Outline

- Overview
- Abstract Data Types
- Sequential Programs
- Concurrent Programs
- Coarse-Grained Synchronization
- Fine-Grained Synchronization
- Optimistic Synchronization
- Lazy Synchronization
- Treiber Stack Algorithm
- Michael&Scott Queue Algorithm

Outline

- Overview
- Abstract Data Types
- Sequential Programs
- Concurrent Programs
- Coarse-Grained Synchronization
- Fine-Grained Synchronization
- Optimistic Synchronization
- Lazy Synchronization
- Treiber Stack Algorithm
- Michael&Scott Queue Algorithm

Abstract Data Type Stack

Abstract Data Type **Stack**

- States
- Methods
- Rules
- initial state

Abstract Data Type Stack

- States
- Methods
- Rules
- initial state

word over integers

state



Abstract Data Type Stack

- States
- Methods
- Rules
- initial state

word over integers

state

2 5 1 2

- *push*
- *pop*

methods

Abstract Data Type Stack

- States
- Methods
- Rules
- initial state

$$w \xrightarrow{push(a)} a \bullet w$$

$$a \bullet w \xrightarrow{pop(a)} w$$

$$\epsilon \xrightarrow{pop(*)} \epsilon$$

rules

guarded commands

word over integers

state

2 5 1 2

- *push*
- *pop*

methods

Abstract Data Type Stack

- States
- Methods
- Rules
- initial state

$$w \xrightarrow{push(a)} a \bullet w$$

$$a \bullet w \xrightarrow{pop(a)} w$$

$$\epsilon \xrightarrow{pop(*)} \epsilon$$

rules

guarded commands

word over integers

state

2 5 1 2

- *push*
- *pop*

methods

Abstract Data Type Stack

- States
- Methods
- Rules
- initial state

$$w \xrightarrow{push(a)} a \bullet w$$

$$a \bullet w \xrightarrow{pop(a)} w$$

$$\epsilon \xrightarrow{pop(*)} \epsilon$$

initially ϵ

rules

guarded commands

word over integers

state

2 5 1 2

- *push*
- *pop*

methods

Abstract Data Type Stack

- States
- Methods
- Rules
- initial state

$$w \xrightarrow{push(a)} a \bullet w$$

$$a \bullet w \xrightarrow{pop(a)} w$$

$$\epsilon \xrightarrow{pop(*)} \epsilon$$

initially ϵ

rules

guarded commands

word over integers

state

- *push*
- *pop*

methods

Abstract Data Type Stack

- States
- Methods
- Rules
- initial state

$$w \xrightarrow{push(a)} a \bullet w$$

$$a \bullet w \xrightarrow{pop(a)} w$$

$$\epsilon \xrightarrow{pop(*)} \epsilon$$

initially ϵ

rules

guarded commands

word over integers

state

- *push*
- *pop*

methods

Abstract Data Type Stack

- States
- Methods
- Rules
- initial state

$$w \xrightarrow{push(a)} a \bullet w$$

$$a \bullet w \xrightarrow{pop(a)} w$$

$$\epsilon \xrightarrow{pop(*)} \epsilon$$

initially ϵ

rules

guarded commands

word over integers

state

- *push*
- *pop*

methods

push(3)

Abstract Data Type Stack

- States
- Methods
- Rules
- initial state

$$w \xrightarrow{push(a)} a \bullet w$$

$$a \bullet w \xrightarrow{pop(a)} w$$

$$\epsilon \xrightarrow{pop(*)} \epsilon$$

initially ϵ

rules

guarded commands

word over integers

state

3

- *push*
- *pop*

methods

push(3)

Abstract Data Type Stack

- States
- Methods
- Rules
- initial state

$$w \xrightarrow{push(a)} a \bullet w$$

$$a \bullet w \xrightarrow{pop(a)} w$$

$$\epsilon \xrightarrow{pop(*)} \epsilon$$

initially ϵ

rules

guarded commands

word over integers

state

3

- *push*
- *pop*

methods

push(3) push(7)

Abstract Data Type Stack

- States
- Methods
- Rules
- initial state

$$w \xrightarrow{push(a)} a \bullet w$$

$$a \bullet w \xrightarrow{pop(a)} w$$

$$\epsilon \xrightarrow{pop(*)} \epsilon$$

initially ϵ

rules

guarded commands

word over integers

state

7 3

- *push*
- *pop*

methods

push(3) push(7)

Abstract Data Type Stack

- States
- Methods
- Rules
- initial state

$$w \xrightarrow{push(a)} a \bullet w$$

$$a \bullet w \xrightarrow{pop(a)} w$$

$$\epsilon \xrightarrow{pop(*)} \epsilon$$

initially ϵ

rules

guarded commands

word over integers

state

7 3

- *push*
- *pop*

methods

push(3) push(7) push(5)

Abstract Data Type Stack

- States
- Methods
- Rules
- initial state

$$w \xrightarrow{push(a)} a \bullet w$$

$$a \bullet w \xrightarrow{pop(a)} w$$

$$\epsilon \xrightarrow{pop(*)} \epsilon$$

initially ϵ

rules

guarded commands

word over integers

state

5 7 3

- *push*
- *pop*

methods

push(3) push(7) push(5)

Abstract Data Type Stack

- States
- Methods
- Rules
- initial state

$$w \xrightarrow{push(a)} a \bullet w$$

$$a \bullet w \xrightarrow{pop(a)} w$$

$$\epsilon \xrightarrow{pop(*)} \epsilon$$

initially ϵ

rules

guarded commands

word over integers

state

5 7 3

- *push*
- *pop*

methods

push(3) push(7) push(5) pop(5)

Abstract Data Type Stack

- States
- Methods
- Rules
- initial state

$$w \xrightarrow{push(a)} a \bullet w$$

$$a \bullet w \xrightarrow{pop(a)} w$$

$$\epsilon \xrightarrow{pop(*)} \epsilon$$

initially ϵ

rules

guarded commands

word over integers

state

7 3

- *push*
- *pop*

methods

push(3) push(7) push(5) pop(5)

Abstract Data Type Stack

- States
- Methods
- Rules
- initial state

$$w \xrightarrow{push(a)} a \bullet w$$

$$a \bullet w \xrightarrow{pop(a)} w$$

$$\epsilon \xrightarrow{pop(*)} \epsilon$$

initially ϵ

rules

guarded commands

word over integers

state

7 3

- *push*
- *pop*

methods

push(3) push(7) push(5) pop(5) pop(7)

Abstract Data Type Stack

- States
- Methods
- Rules
- initial state

$$w \xrightarrow{push(a)} a \bullet w$$

$$a \bullet w \xrightarrow{pop(a)} w$$

$$\epsilon \xrightarrow{pop(*)} \epsilon$$

initially ϵ

rules

guarded commands

word over integers

state

3

- *push*
- *pop*

methods

push(3) push(7) push(5) pop(5) pop(7)

Abstract Data Type Stack

- States
- Methods
- Rules
- initial state

$$w \xrightarrow{push(a)} a \bullet w$$

$$a \bullet w \xrightarrow{pop(a)} w$$

$$\epsilon \xrightarrow{pop(*)} \epsilon$$

initially ϵ

rules

guarded commands

word over integers

state

3

- *push*
- *pop*

methods

trace

τ

push(3) push(7) push(5) pop(5) pop(7)

Abstract Data Type Stack

- States
- Methods
- Rules
- initial state

$$w \xrightarrow{push(a)} a \bullet w$$

$$a \bullet w \xrightarrow{pop(a)} w$$

$$\epsilon \xrightarrow{pop(*)} \epsilon$$

initially ϵ

rules

guarded commands

word over integers

state

3

- *push*
- *pop*

methods

$\tau \vDash \mathrm{STACK}$

trace

τ

push(3) push(7) push(5) pop(5) pop(7)

The Treiber Stack Algorithm

The Treiber Stack Algorithm

CAS: Compare And Swap

The Treiber Stack Algorithm

CAS: Compare And Swap

```
CAS (x,expected,new):
Integer tmp
1  Atomically
2    tmp = x
3    if (tmp == expected)
4      x = new
5      return true
6    else
7      return false
```

The Treiber Stack Algorithm

CAS: Compare And Swap

```
CAS (x,expected,new):
Integer tmp
1  Atomically
2    tmp = x
3    if (tmp == expected)
4      x = new
5      return true
6    else
7      return false
```

hardware

software

The Treiber Stack Algorithm

CAS: Compare And Swap

```
CAS (x,expected,new):
Integer tmp
1  Atomically
2    tmp = x
3    if (tmp == expected)
4      x = new
5      return true
6    else
7      return false
```

- **hardware**
  - **Intel, AMD** — CMPXCHG
  - **SPARC** — CAS
- **software**

The Treiber Stack Algorithm

CAS: Compare And Swap

```
CAS (x,expected,new):
Integer tmp
1  Atomically
2    tmp = x
3    if (tmp == expected)
4      x = new
5      return true
6    else
7      return false
```

- **hardware**
  - **Intel, AMD** — CMPXCHG
  - **SPARC** — CAS
- **software**
  - **java** — CompareAndSet
  - **C#** — CompareExchange

The Treiber Stack Algorithm

CAS: Compare And Swap

```
CAS (x,expected,new):
Integer tmp
1  Atomically
2    tmp = x
3    if (tmp == expected)
4      x = new
5      return true
6    else
7      return false
```

- **hardware**
  - **Intel, AMD** — CMPXCHG
  - **SPARC** — CAS
- **software**
  - **java** — CompareAndSet
  - **C#** — CompareExchange

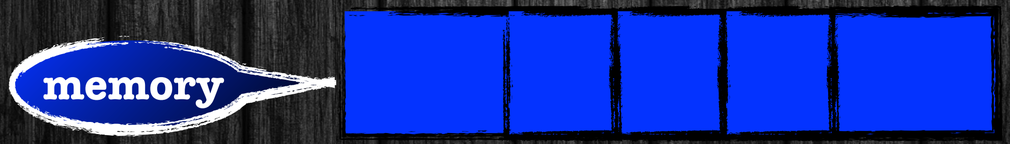


The Treiber Stack Algorithm

CAS: Compare And Swap

```
CAS (x,expected,new):
Integer tmp
1  Atomically
2    tmp = x
3    if (tmp == expected)
4      x = new
5      return true
6    else
7      return false
```

- **hardware**
  - **Intel, AMD** — CMPXCHG
  - **SPARC** — CAS
- **software**
  - **java** — CompareAndSet
  - **C#** — CompareExchange

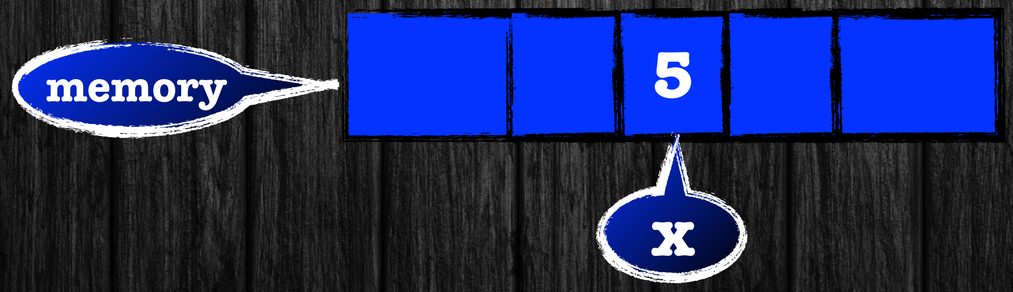


The Treiber Stack Algorithm

CAS: Compare And Swap

```
CAS (x,expected,new):
Integer tmp
1  Atomically
2    tmp = x
3    if (tmp == expected)
4      x = new
5      return true
6    else
7      return false
```

- hardware
  - Intel, AMD — CMPXCHG
  - SPARC — CAS
- software
  - java — CompareAndSet
  - C# — CompareExchange

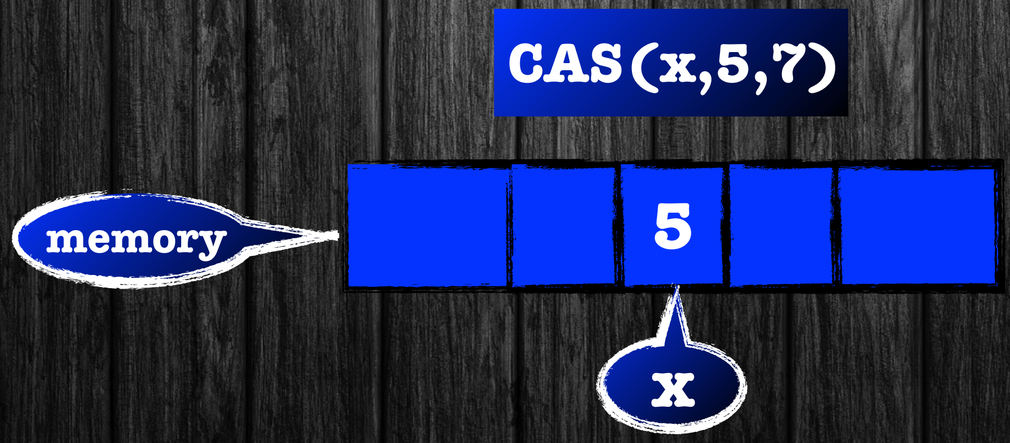


The Treiber Stack Algorithm

CAS: Compare And Swap

```
CAS (x,expected,new):
Integer tmp
1  Atomically
2    tmp = x
3    if (tmp == expected)
4      x = new
5      return true
6    else
7      return false
```

- hardware
  - Intel, AMD — CMPXCHG
  - SPARC — CAS
- software
  - java — CompareAndSet
  - C# — CompareExchange

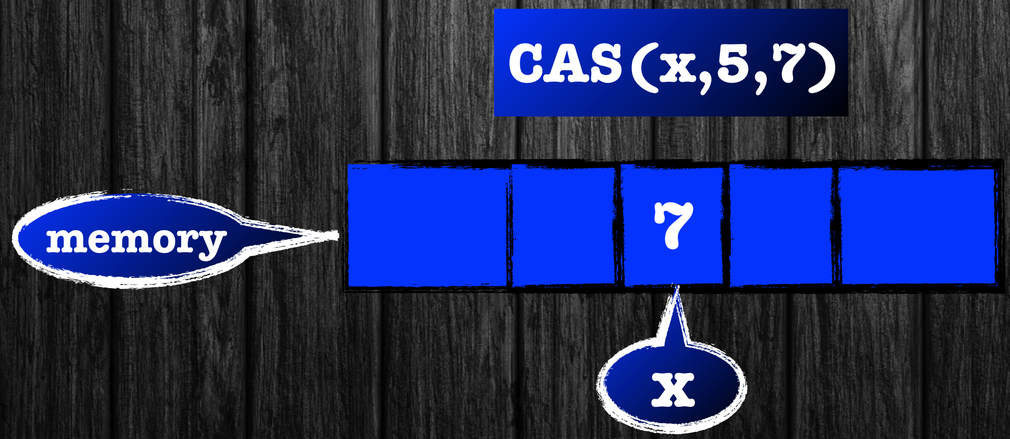


The Treiber Stack Algorithm

CAS: Compare And Swap

```
CAS (x,expected,new):
Integer tmp
1  Atomically
2    tmp = x
3    if (tmp == expected)
4      x = new
5      return true
6    else
7      return false
```

- hardware
  - Intel, AMD — CMPXCHG
  - SPARC — CAS
- software
  - java — CompareAndSet
  - C# — CompareExchange



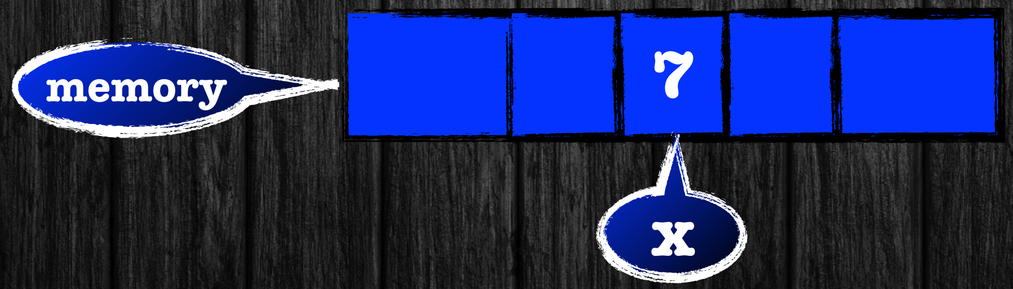
The Treiber Stack Algorithm

CAS: Compare And Swap

```
CAS (x,expected,new):
Integer tmp
1  Atomically
2    tmp = x
3    if (tmp == expected)
4      x = new
5      return true
6    else
7      return false
```

- hardware
  - Intel, AMD — CMPXCHG
  - SPARC — CAS
- software
  - java — CompareAndSet
  - C# — CompareExchange

CAS(x,5,7) → **true**

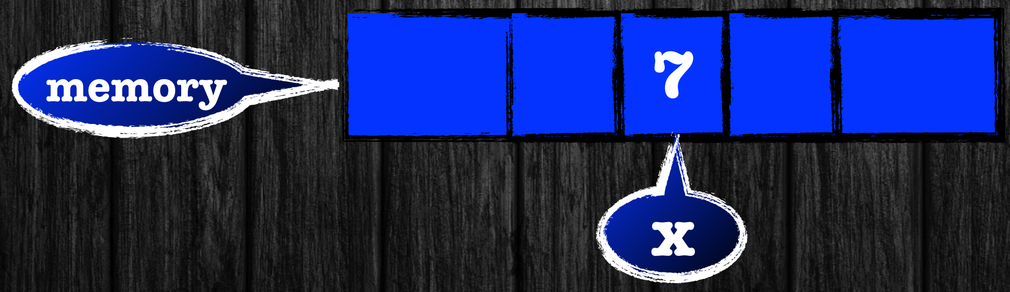


The Treiber Stack Algorithm

CAS: Compare And Swap

```
CAS (x,expected,new):
Integer tmp
1  Atomically
2    tmp = x
3    if (tmp == expected)
4      x = new
5      return true
6    else
7      return false
```

- **hardware**
  - **Intel, AMD** — CMPXCHG
  - **SPARC** — CAS
- **software**
  - **java** — CompareAndSet
  - **C#** — CompareExchange

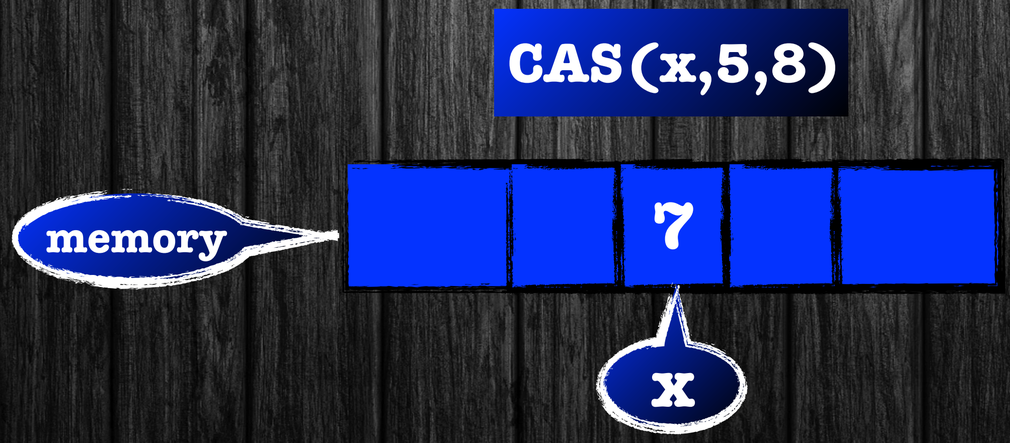


The Treiber Stack Algorithm

CAS: Compare And Swap

```
CAS (x,expected,new):
Integer tmp
1  Atomically
2    tmp = x
3    if (tmp == expected)
4      x = new
5      return true
6    else
7      return false
```

- hardware
  - Intel, AMD — CMPXCHG
  - SPARC — CAS
- software
  - java — CompareAndSet
  - C# — CompareExchange



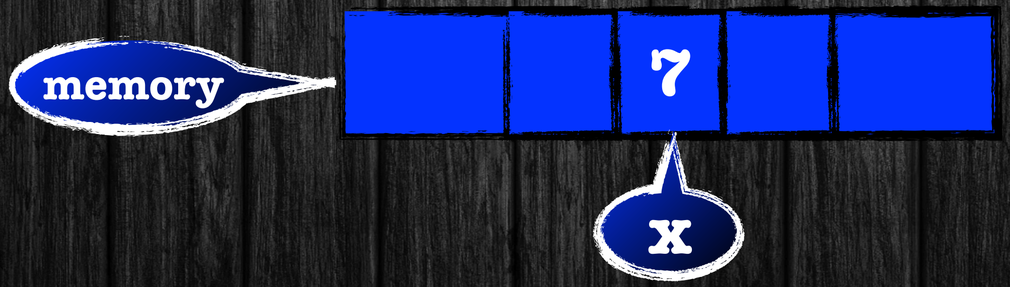
The Treiber Stack Algorithm

CAS: Compare And Swap

```
CAS (x,expected,new):
Integer tmp
1  Atomically
2    tmp = x
3    if (tmp == expected)
4      x = new
5      return true
6    else
7      return false
```

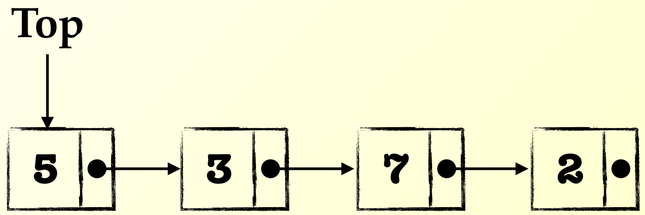
- hardware
  - Intel, AMD — CMPXCHG
  - SPARC — CAS
- software
  - java — CompareAndSet
  - C# — CompareExchange

CAS(x,5,8) **false**



The Treiber Stack Algorithm

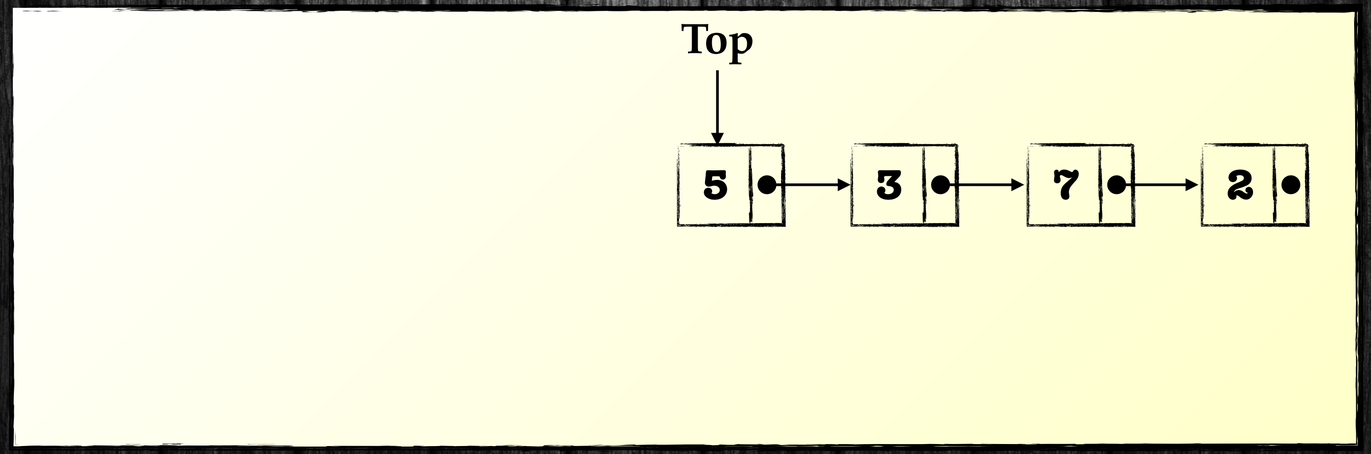




The Treiber Stack Algorithm

```
push(k):
Node t
1  n = new Node(k,-)
2  while (true)
3    t = Top
4    n.next = t
5    if (CAS (Top,t,n))
6      exit
```





The Treiber Stack Algorithm

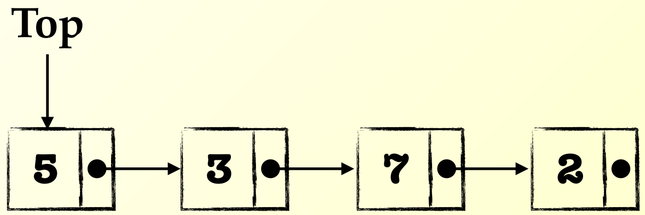
```
push(k):
Node t
1  n = new Node(k,-)
2  while (true)
3    t = Top
4    n.next = t
5    if (CAS (Top,t,n))
6      exit
```



push(4)

stack

5 3 7 2



The Treiber Stack Algorithm

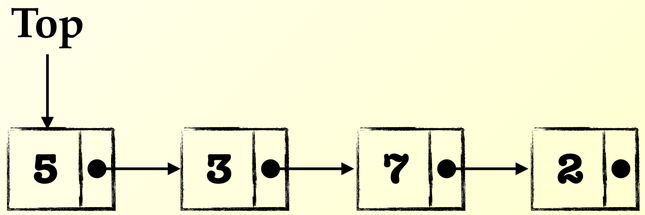
```
push(k):
Node t
1  n = new Node(k,-)
2  while (true)
3    t = Top
4    n.next = t
5    if (CAS (Top,t,n))
6      exit
```



push(4)

stack

5 3 7 2



The Treiber Stack Algorithm

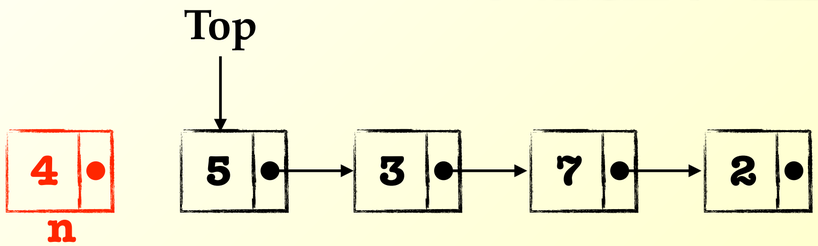
```
push(k):
Node t
1  n = new Node(k,-)
2  while (true)
3    t = Top
4    n.next = t
5    if (CAS (Top,t,n))
6      exit
```



push(4)

stack

5 3 7 2



The Treiber Stack Algorithm

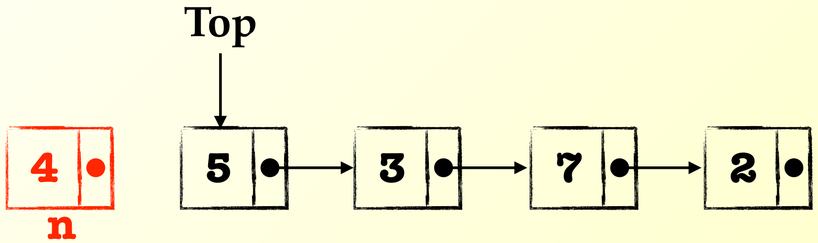
```
push(k):
Node t
1  n = new Node(k,-)
2  while (true)
3    t = Top
4    n.next = t
5    if (CAS (Top,t,n))
6      exit
```



push(4)

stack

5 3 7 2



The Treiber Stack Algorithm

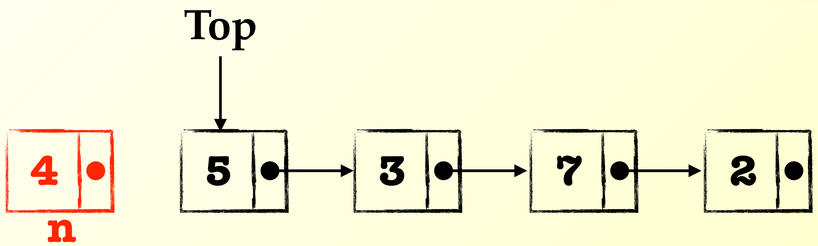
```
push(k):
Node t
1  n = new Node(k,-)
2  while (true)
3    t = Top
4    n.next = t
5    if (CAS (Top,t,n))
6      exit
```



push(4)

stack

5 3 7 2



The Treiber Stack Algorithm

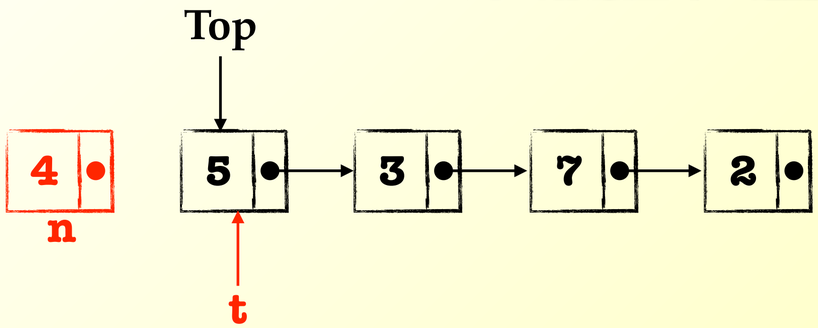
```
push(k):
Node t
1  n = new Node(k,-)
2  while (true)
3    t = Top
4    n.next = t
5    if (CAS (Top,t,n))
6      exit
```



push(4)

stack

5 3 7 2



The Treiber Stack Algorithm

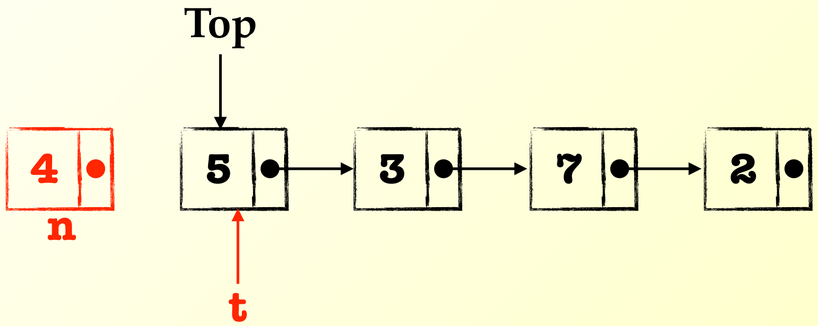
```
push(k):
Node t
1  n = new Node(k,-)
2  while (true)
3    t = Top
4    n.next = t
5    if (CAS (Top,t,n))
6      exit
```



push(4)

stack

5 3 7 2



The Treiber Stack Algorithm

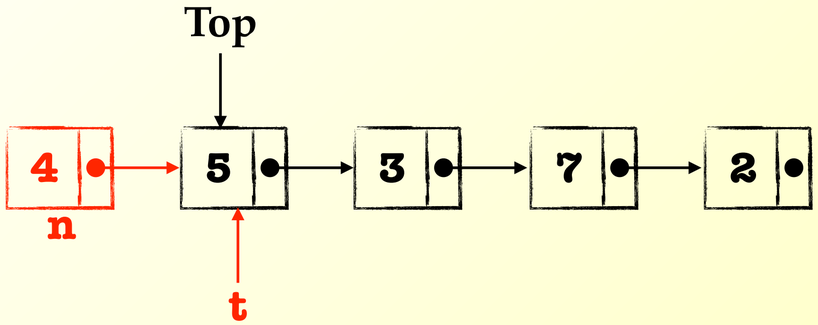
```
push(k):
Node t
1  n = new Node(k,-)
2  while (true)
3    t = Top
4    n.next = t
5    if (CAS (Top,t,n))
6      exit
```



push(4)

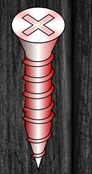
stack

5 3 7 2



The Treiber Stack Algorithm

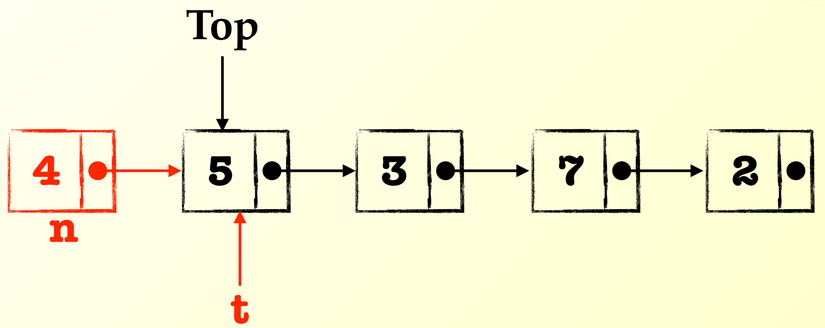
```
push(k):
Node t
1  n = new Node(k,-)
2  while (true)
3    t = Top
4    n.next = t
5    if (CAS (Top,t,n))
6      exit
```



push(4)

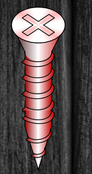
stack

5 3 7 2



The Treiber Stack Algorithm

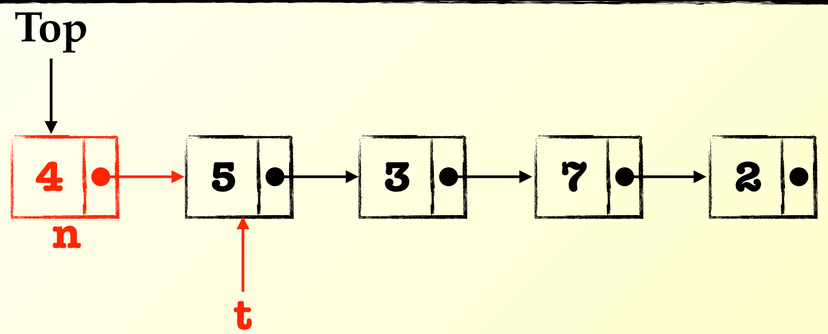
```
push(k):
Node t
1  n = new Node(k,-)
2  while (true)
3    t = Top
4    n.next = t
5    if (CAS (Top,t,n))
6      exit
```



push(4)

stack

5 3 7 2



The Treiber Stack Algorithm

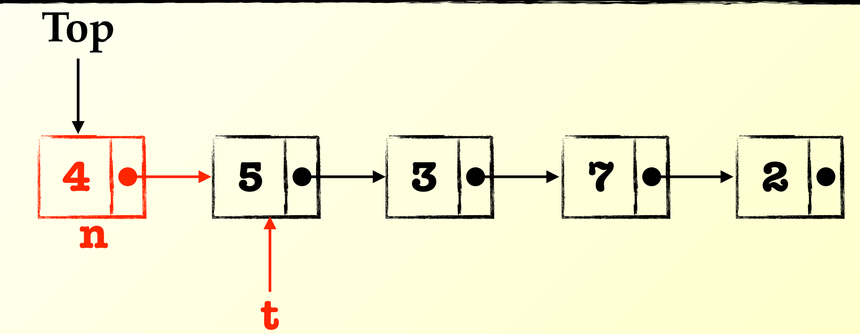
```
push(k):
Node t
1  n = new Node(k,-)
2  while (true)
3    t = Top
4    n.next = t
5    if (CAS (Top,t,n))
6      exit
```



push(4)

stack

5 3 7 2



The Treiber Stack Algorithm

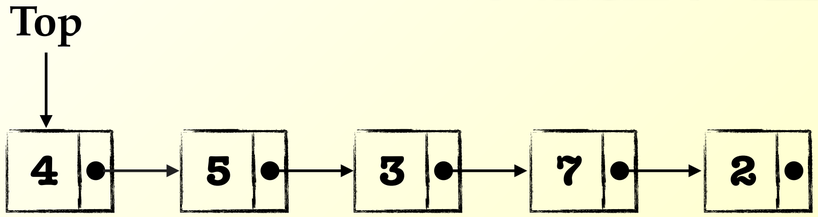
```
push(k):
Node t
1  n = new Node(k,-)
2  while (true)
3    t = Top
4    n.next = t
5    if (CAS (Top,t,n))
6      exit
```



push(4)

stack

5 3 7 2



The Treiber Stack Algorithm

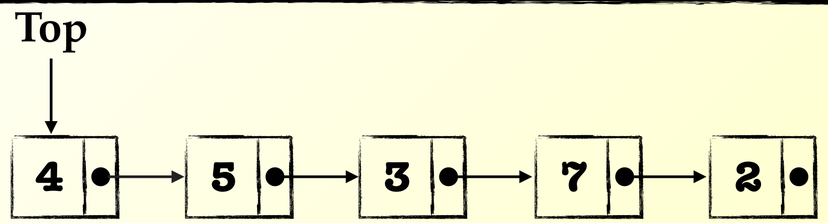
```
push(k):
Node t
1  n = new Node(k,-)
2  while (true)
3    t = Top
4    n.next = t
5    if (CAS (Top,t,n))
6      exit
```



push(4)

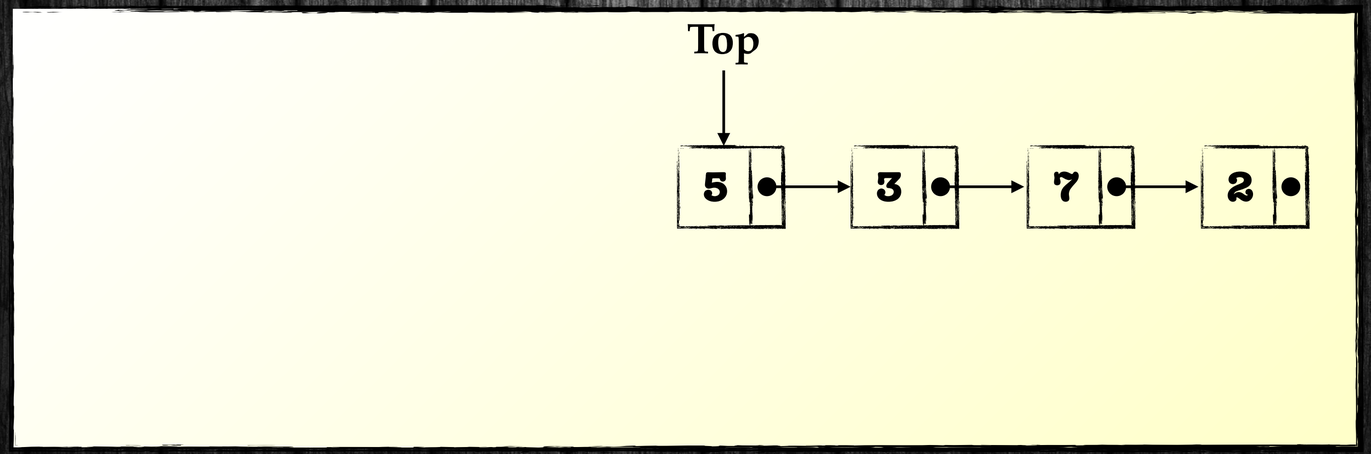
stack

4
5
3
7
2



The Treiber Stack Algorithm

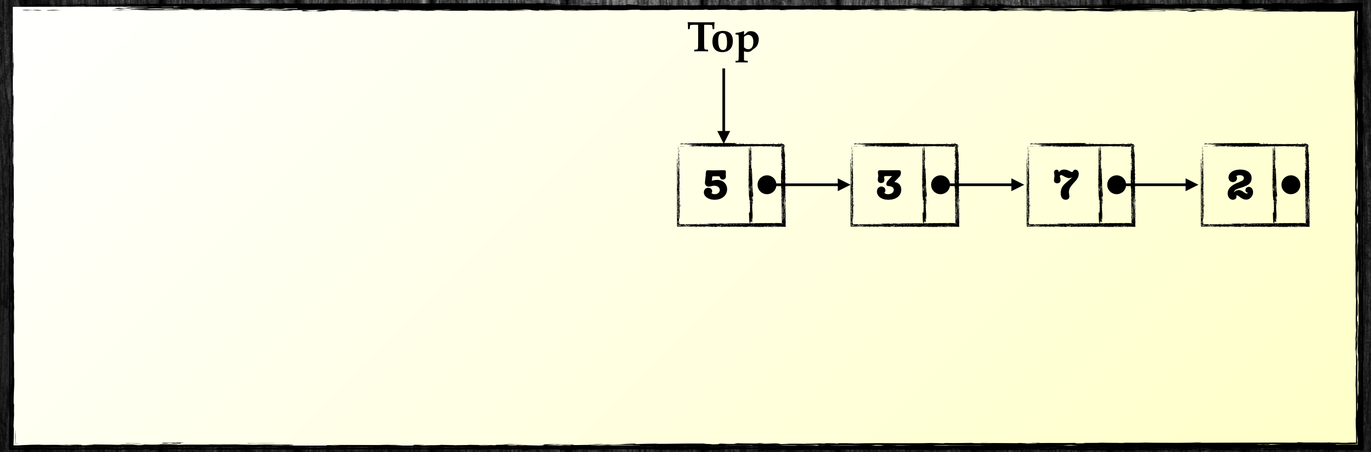
Concurrent push operations



The Treiber Stack Algorithm

Concurrent push operations

```
push(k):
Node t
1  n = new Node(k,-)
2  while (true)
3    t = Top
4    n.next = t
5    if (CAS (Top,t,n))
6      exit
```



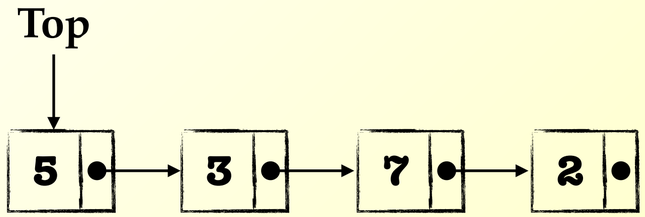
The Treiber Stack Algorithm

Concurrent push operations

```
push(k):
Node t
1  n = new Node(k,-)
2  while (true)
3    t = Top
4    n.next = t
5    if (CAS (Top,t,n))
6      exit
```



push(4)



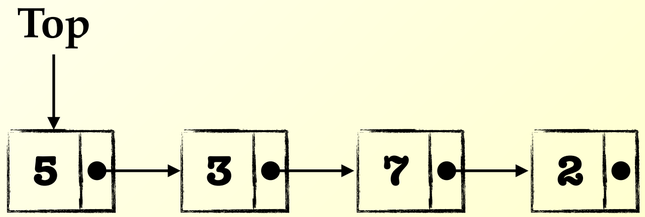
The Treiber Stack Algorithm

Concurrent push operations

```
push(k):
Node t
1  n = new Node(k,-)
2  while (true)
3    t = Top
4    n.next = t
5    if (CAS (Top,t,n))
6      exit
```



push(4)



The Treiber Stack Algorithm

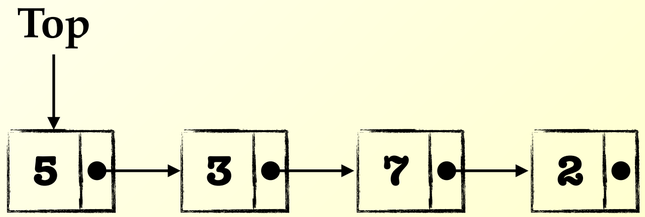
Concurrent push operations

```
push(k):
Node t
1  n = new Node(k,-)
2  while (true)
3    t = Top
4    n.next = t
5    if (CAS (Top,t,n))
6      exit
```



push(4)

```
push(k):
Node t
1  n = new Node(k,-)
2  while (true)
3    t = Top
4    n.next = t
5    if (CAS (Top,t,n))
6      exit
```



The Treiber Stack Algorithm

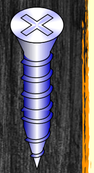
Concurrent push operations

```
push(k):
Node t
1  n = new Node(k,-)
2  while (true)
3    t = Top
4    n.next = t
5    if (CAS (Top,t,n))
6      exit
```

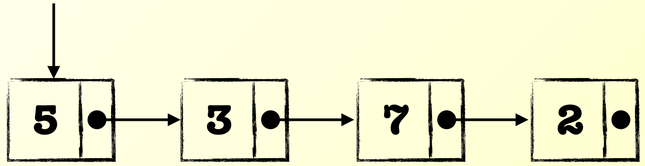


push(4)

push(2)



```
push(k):
Node t
1  n = new Node(k,-)
2  while (true)
3    t = Top
4    n.next = t
5    if (CAS (Top,t,n))
6      exit
```



The Treiber Stack Algorithm

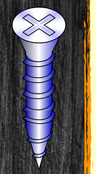
Concurrent push operations

```
push(k):
Node t
1  n = new Node(k,-)
2  while (true)
3    t = Top
4    n.next = t
5    if (CAS (Top,t,n))
6      exit
```

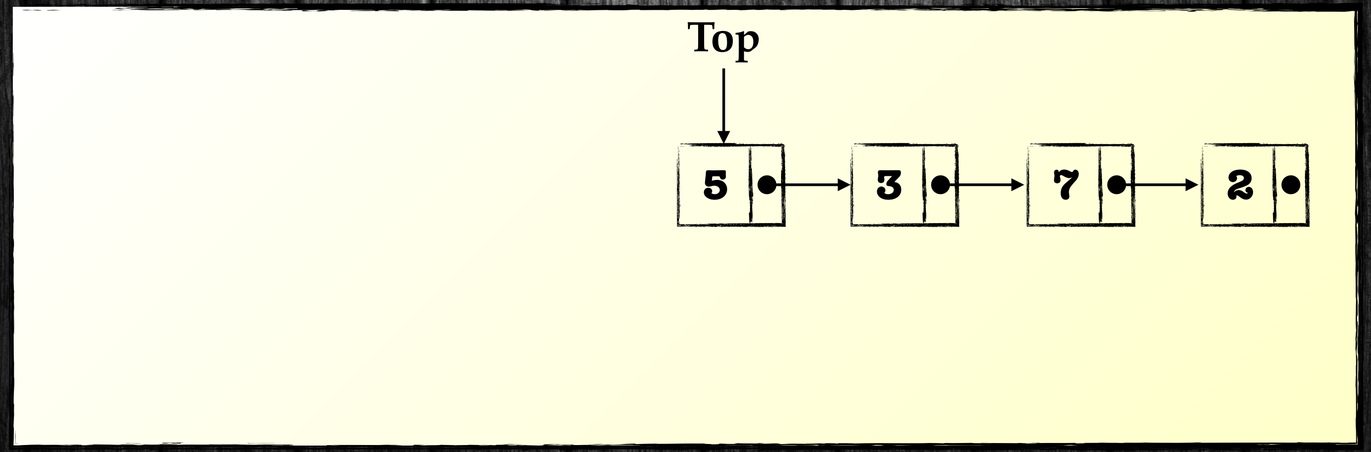


push(4)

push(2)



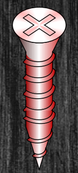
```
push(k):
Node t
1  n = new Node(k,-)
2  while (true)
3    t = Top
4    n.next = t
5    if (CAS (Top,t,n))
6      exit
```



The Treiber Stack Algorithm

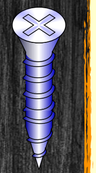
Concurrent push operations

```
push(k):
Node t
1  n = new Node(k,-)
2  while (true)
3    t = Top
4    n.next = t
5    if (CAS (Top,t,n))
6      exit
```

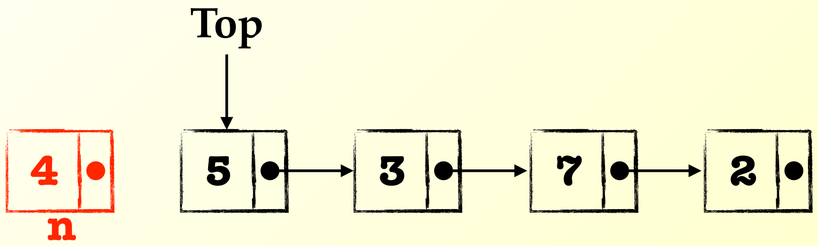


push(4)

push(2)



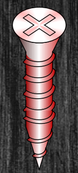
```
push(k):
Node t
1  n = new Node(k,-)
2  while (true)
3    t = Top
4    n.next = t
5    if (CAS (Top,t,n))
6      exit
```



The Treiber Stack Algorithm

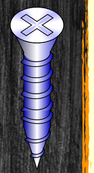
Concurrent push operations

```
push(k):
Node t
1  n = new Node(k,-)
2  while (true)
3    t = Top
4    n.next = t
5    if (CAS (Top,t,n))
6      exit
```

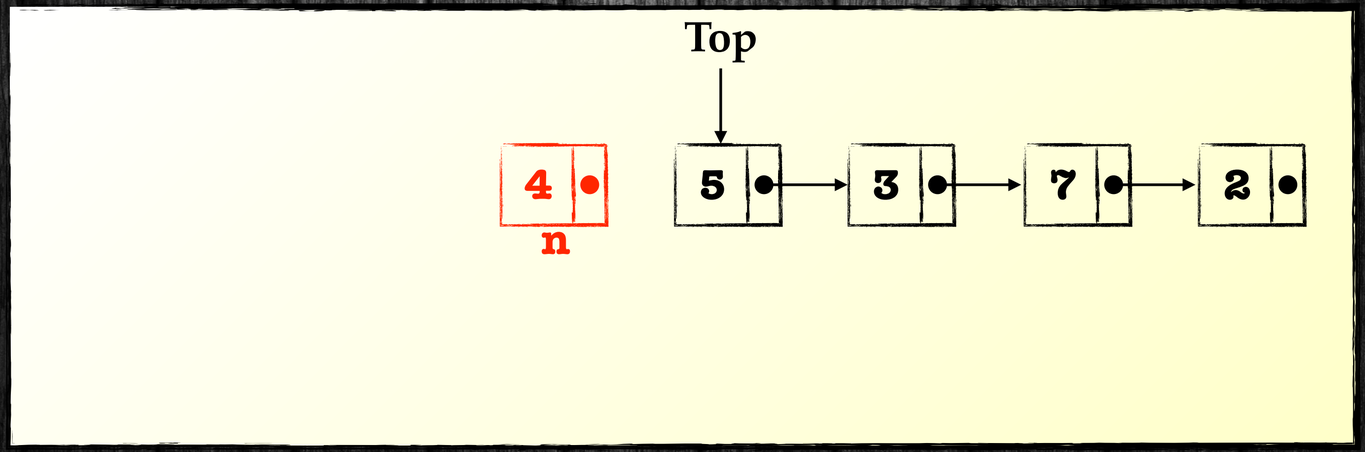


push(4)

push(2)



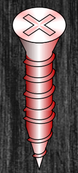
```
push(k):
Node t
1  n = new Node(k,-)
2  while (true)
3    t = Top
4    n.next = t
5    if (CAS (Top,t,n))
6      exit
```



The Treiber Stack Algorithm

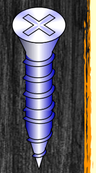
Concurrent push operations

```
push(k):
Node t
1  n = new Node(k,-)
2  while (true)
3    t = Top
4    n.next = t
5    if (CAS (Top,t,n))
6      exit
```

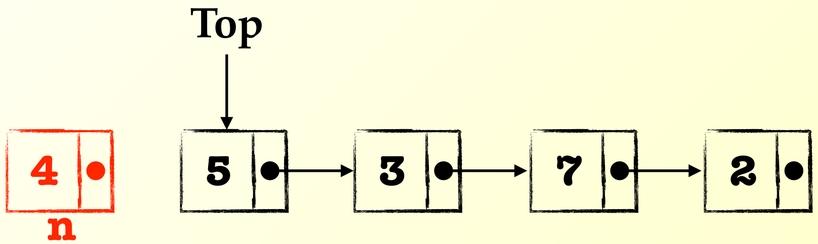


push(4)

push(2)



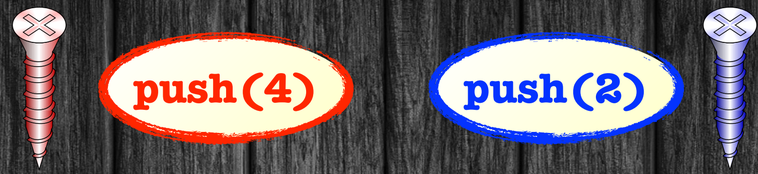
```
push(k):
Node t
1  n = new Node(k,-)
2  while (true)
3    t = Top
4    n.next = t
5    if (CAS (Top,t,n))
6      exit
```



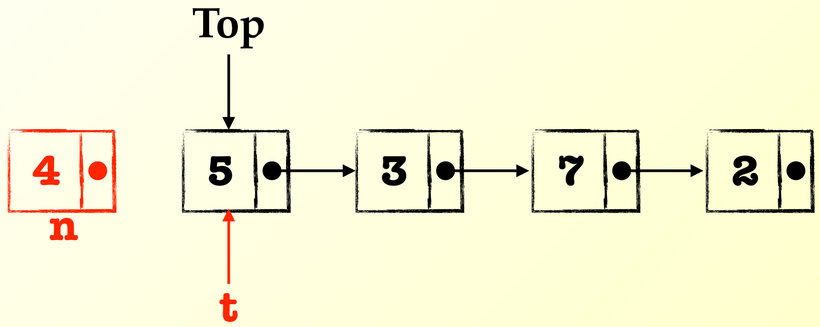
The Treiber Stack Algorithm

Concurrent push operations

```
push(k):
Node t
1  n = new Node(k,-)
2  while (true)
3    t = Top
4    n.next = t
5    if (CAS (Top,t,n))
6      exit
```



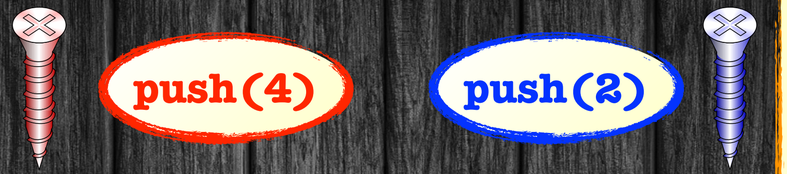

```
push(k):
Node t
1  n = new Node(k,-)
2  while (true)
3    t = Top
4    n.next = t
5    if (CAS (Top,t,n))
6      exit
```



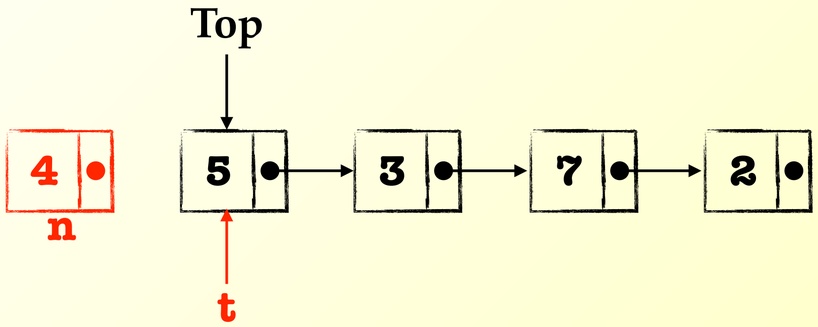
The Treiber Stack Algorithm

Concurrent push operations

```
push(k):
Node t
1  n = new Node(k,-)
2  while (true)
3    t = Top
4    n.next = t
5    if (CAS (Top,t,n))
6      exit
```



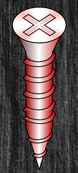

```
push(k):
Node t
1  n = new Node(k,-)
2  while (true)
3    t = Top
4    n.next = t
5    if (CAS (Top,t,n))
6      exit
```



The Treiber Stack Algorithm

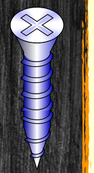
Concurrent push operations

```
push(k):
Node t
1  n = new Node(k,-)
2  while (true)
3    t = Top
4    n.next = t
5    if (CAS (Top,t,n))
6      exit
```

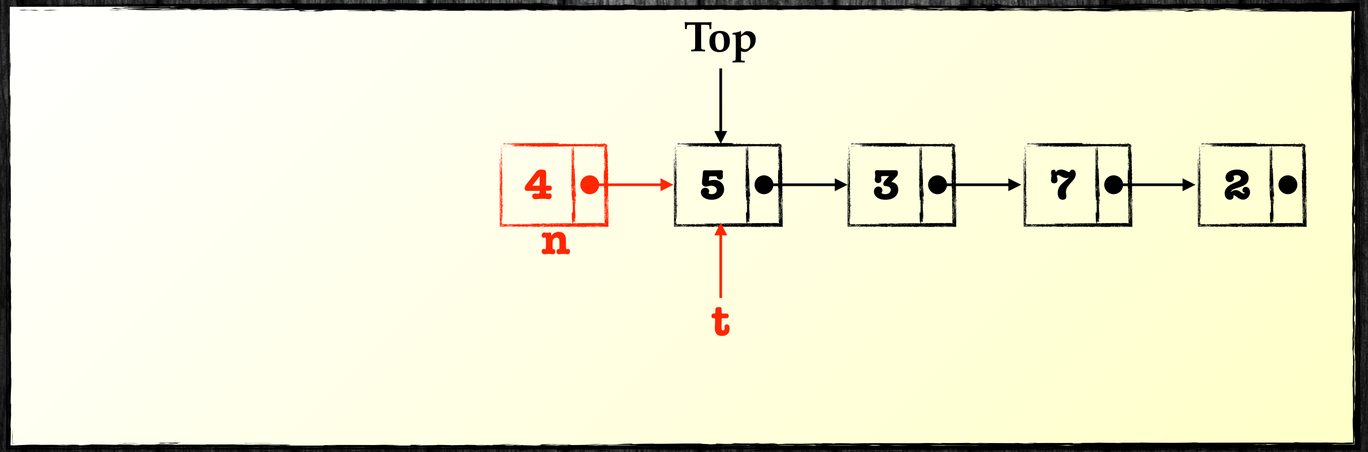


push(4)

push(2)



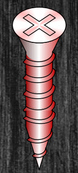
```
push(k):
Node t
1  n = new Node(k,-)
2  while (true)
3    t = Top
4    n.next = t
5    if (CAS (Top,t,n))
6      exit
```



The Treiber Stack Algorithm

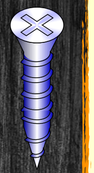
Concurrent push operations

```
push(k):
Node t
1  n = new Node(k,-)
2  while (true)
3    t = Top
4    n.next = t
5    if (CAS (Top,t,n))
6      exit
```

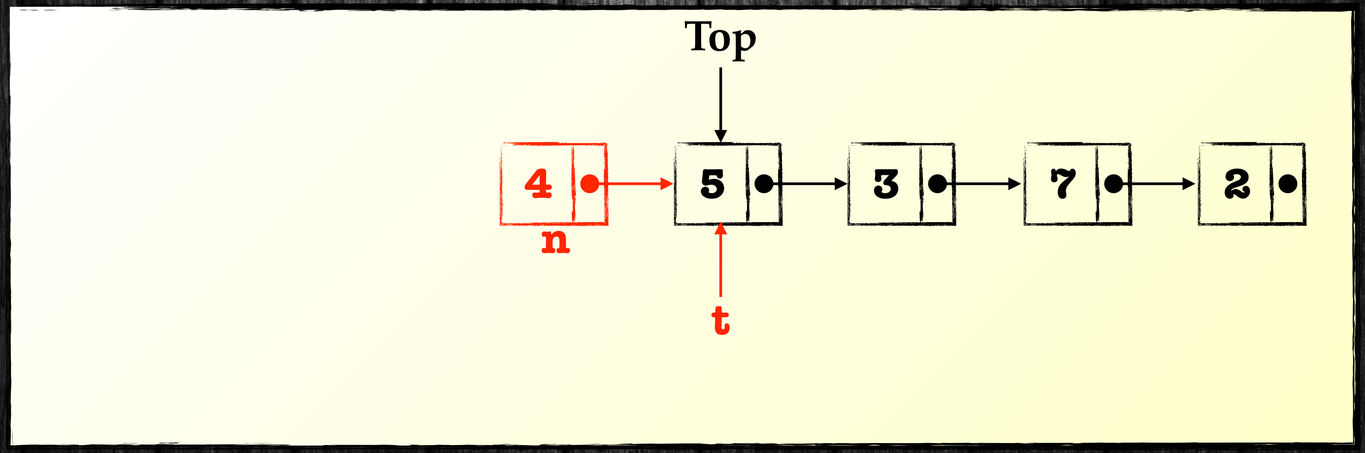


push(4)

push(2)



```
push(k):
Node t
1  n = new Node(k,-)
2  while (true)
3    t = Top
4    n.next = t
5    if (CAS (Top,t,n))
6      exit
```



The Treiber Stack Algorithm

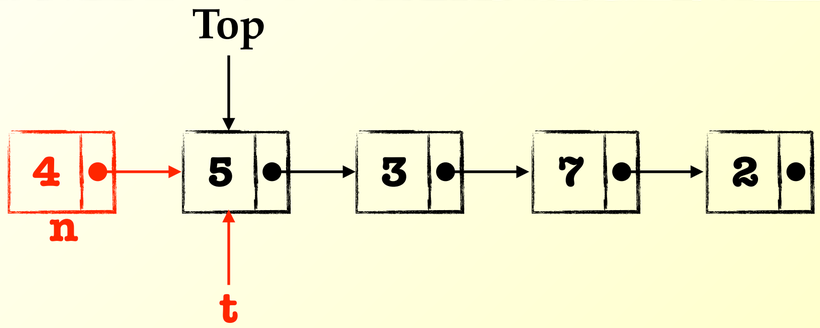
Concurrent push operations

```
push(k):
Node t
1  n = new Node(k,-)
2  while (true)
3    t = Top
4    n.next = t
5    if (CAS (Top,t,n))
6      exit
```

push(4)

push(2)

```
push(k):
Node t
1  n = new Node(k,-)
2  while (true)
3    t = Top
4    n.next = t
5    if (CAS (Top,t,n))
6      exit
```



The Treiber Stack Algorithm

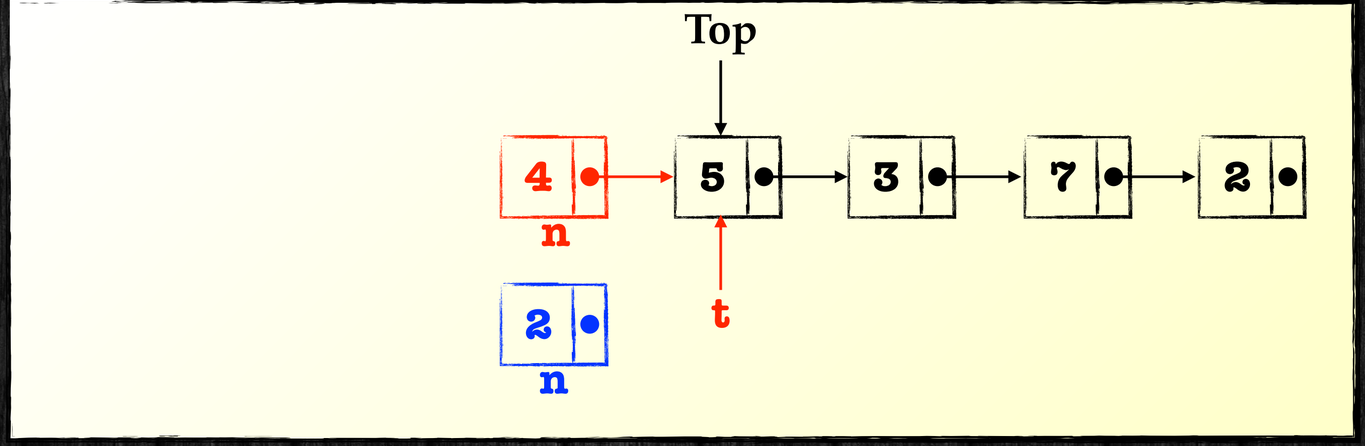
Concurrent push operations

```
push(k):
Node t
1  n = new Node(k,-)
2  while (true)
3    t = Top
4    n.next = t
5    if (CAS (Top,t,n))
6      exit
```

push(4)

push(2)

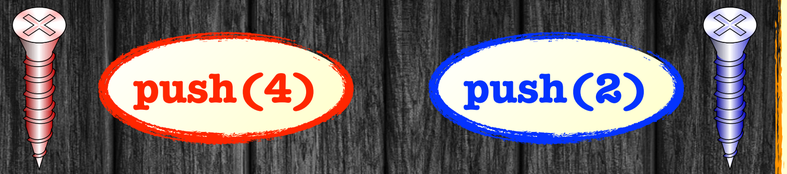
```
push(k):
Node t
1  n = new Node(k,-)
2  while (true)
3    t = Top
4    n.next = t
5    if (CAS (Top,t,n))
6      exit
```



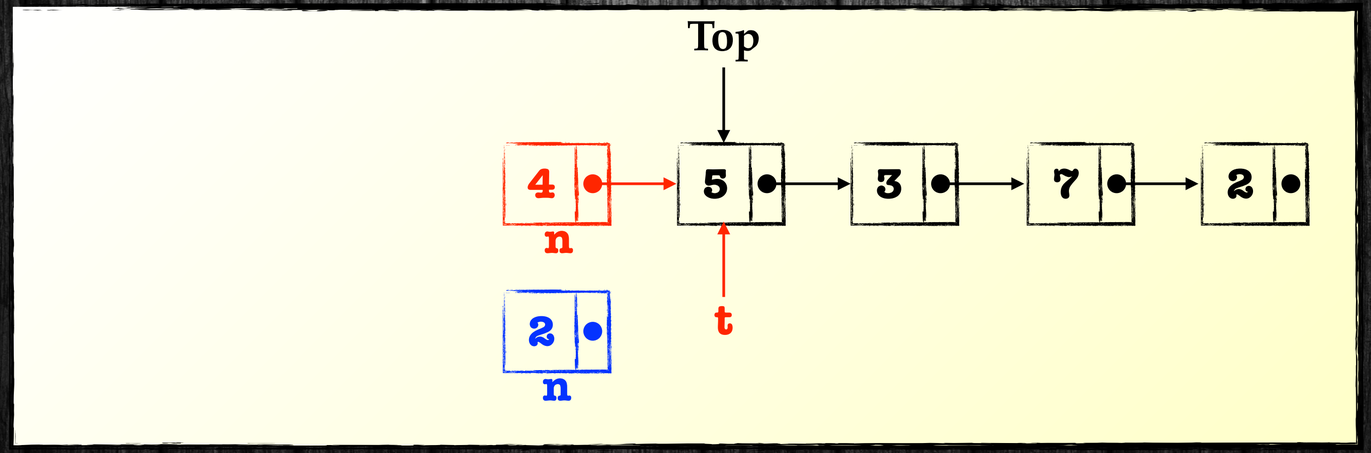
The Treiber Stack Algorithm

Concurrent push operations

```
push(k):
Node t
1  n = new Node(k,-)
2  while (true)
3    t = Top
4    n.next = t
5    if (CAS (Top,t,n))
6      exit
```



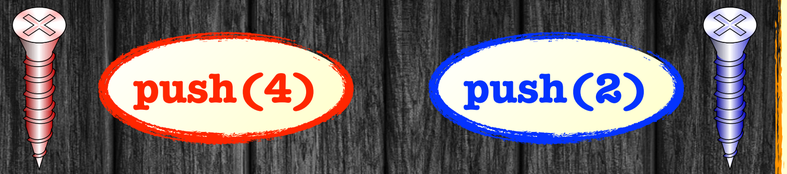

```
push(k):
Node t
1  n = new Node(k,-)
2  while (true)
3    t = Top
4    n.next = t
5    if (CAS (Top,t,n))
6      exit
```



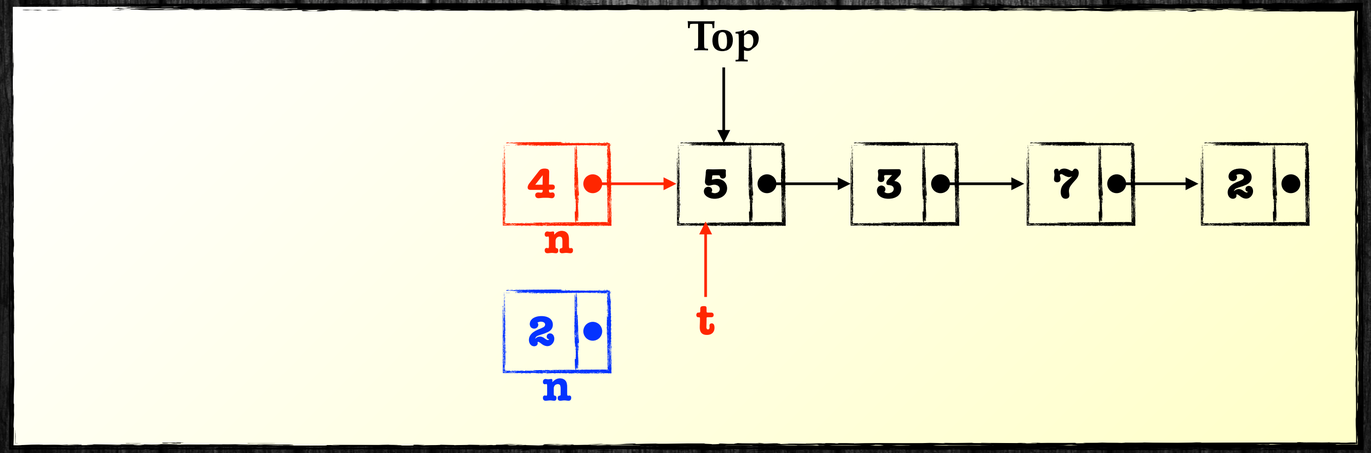
The Treiber Stack Algorithm

Concurrent push operations

```
push(k):
Node t
1  n = new Node(k,-)
2  while (true)
3    t = Top
4    n.next = t
5    if (CAS (Top,t,n))
6      exit
```




```
push(k):
Node t
1  n = new Node(k,-)
2  while (true)
3    t = Top
4    n.next = t
5    if (CAS (Top,t,n))
6      exit
```



The Treiber Stack Algorithm

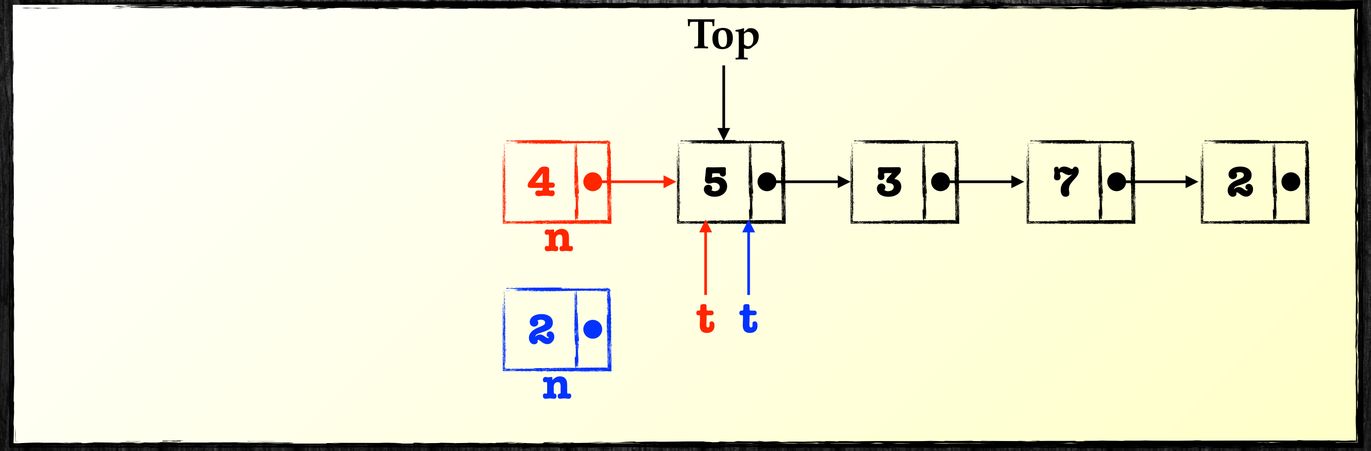
Concurrent push operations

```
push(k):
Node t
1  n = new Node(k,-)
2  while (true)
3    t = Top
4    n.next = t
5    if (CAS (Top,t,n))
6      exit
```

push(4)

push(2)

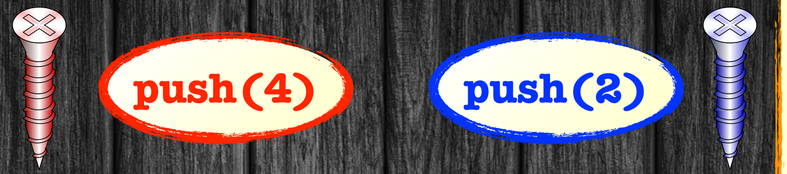
```
push(k):
Node t
1  n = new Node(k,-)
2  while (true)
3    t = Top
4    n.next = t
5    if (CAS (Top,t,n))
6      exit
```



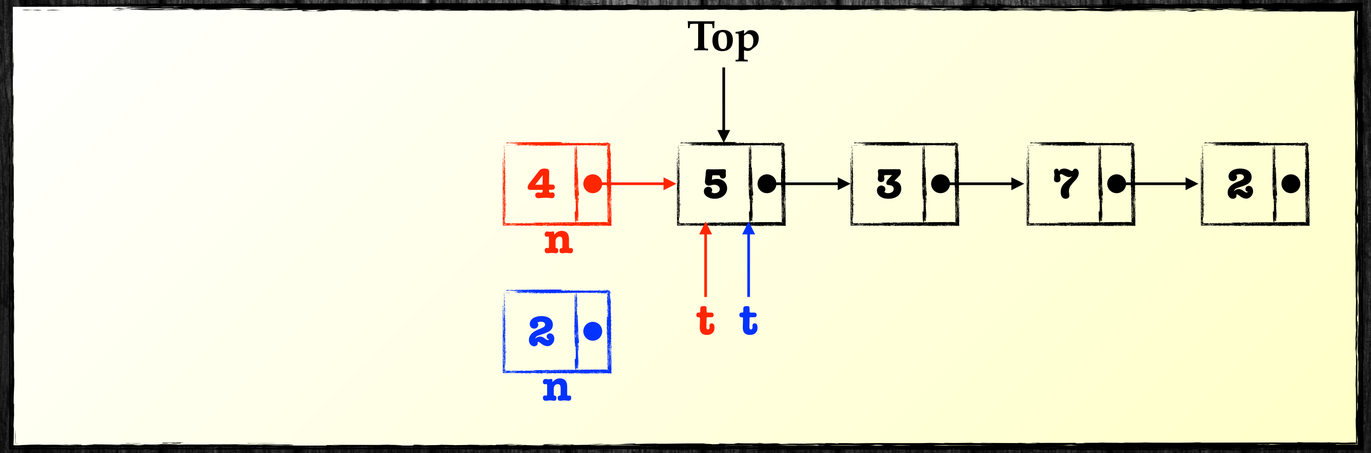
The Treiber Stack Algorithm

Concurrent push operations

```
push(k):
Node t
1  n = new Node(k,-)
2  while (true)
3    t = Top
4    n.next = t
5    if (CAS (Top,t,n))
6      exit
```



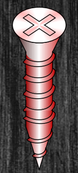

```
push(k):
Node t
1  n = new Node(k,-)
2  while (true)
3    t = Top
4    n.next = t
5    if (CAS (Top,t,n))
6      exit
```



The Treiber Stack Algorithm

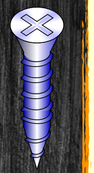
Concurrent push operations

```
push(k):
Node t
1  n = new Node(k,-)
2  while (true)
3    t = Top
4    n.next = t
5    if (CAS (Top,t,n))
6      exit
```

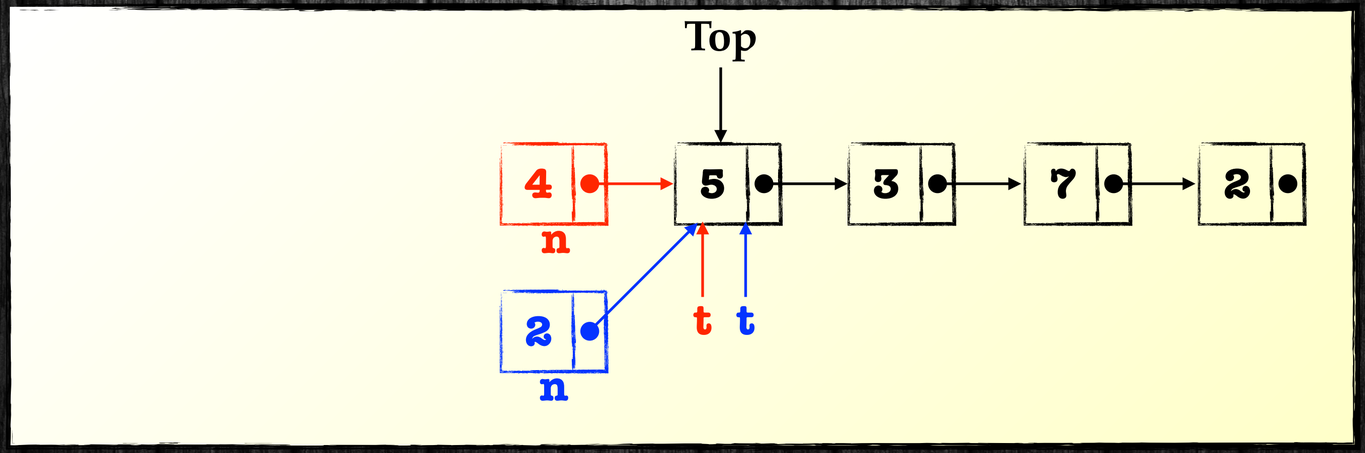


push(4)

push(2)



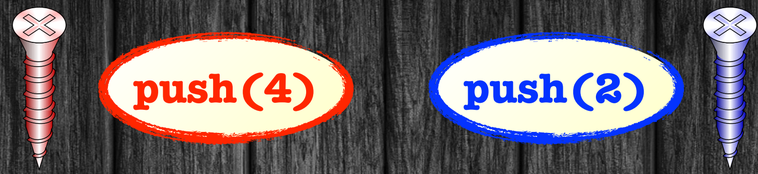
```
push(k):
Node t
1  n = new Node(k,-)
2  while (true)
3    t = Top
4    n.next = t
5    if (CAS (Top,t,n))
6      exit
```



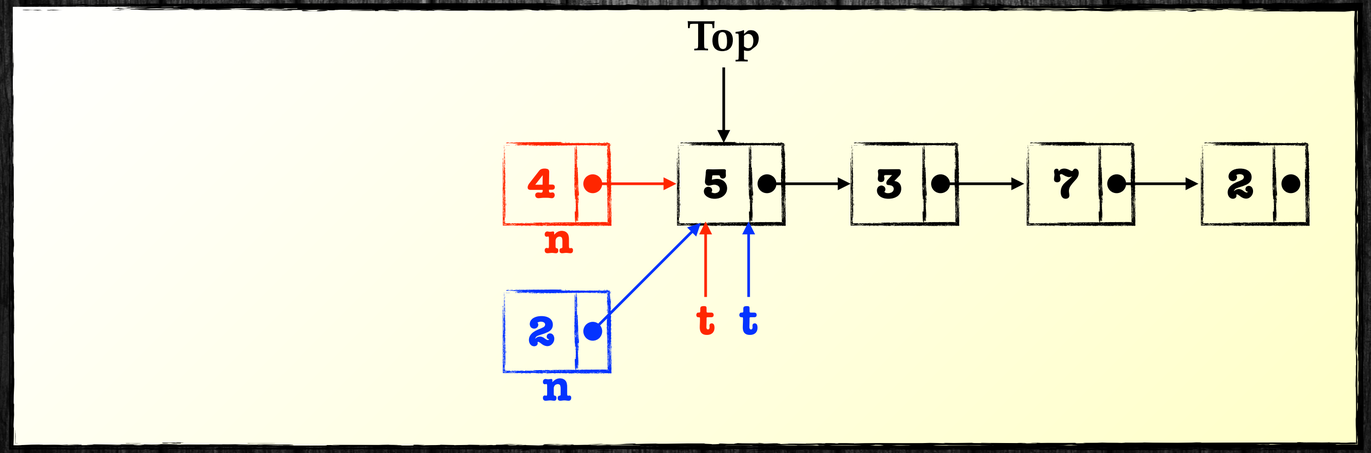
The Treiber Stack Algorithm

Concurrent push operations

```
push(k):
Node t
1  n = new Node(k,-)
2  while (true)
3    t = Top
4    n.next = t
5    if (CAS (Top,t,n))
6      exit
```




```
push(k):
Node t
1  n = new Node(k,-)
2  while (true)
3    t = Top
4    n.next = t
5    if (CAS (Top,t,n))
6      exit
```



The Treiber Stack Algorithm

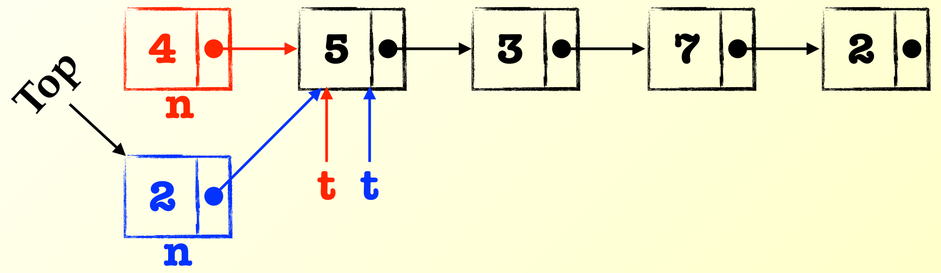
Concurrent push operations

```
push(k):
Node t
1  n = new Node(k,-)
2  while (true)
3    t = Top
4    n.next = t
5    if (CAS (Top,t,n))
6      exit
```

push(4)

push(2)

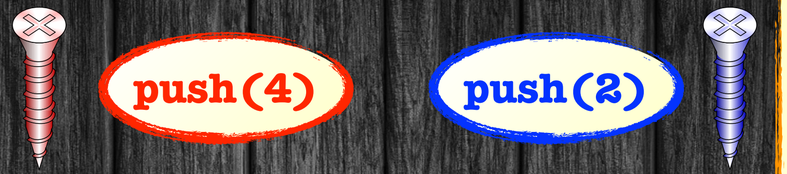
```
push(k):
Node t
1  n = new Node(k,-)
2  while (true)
3    t = Top
4    n.next = t
5    if (CAS (Top,t,n))
6      exit
```



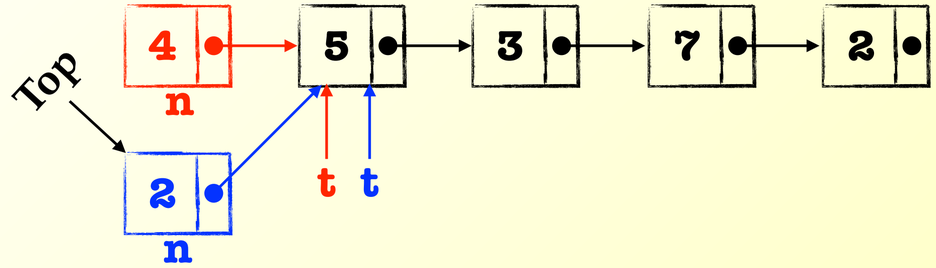
The Treiber Stack Algorithm

Concurrent push operations

```
push(k):
Node t
1  n = new Node(k,-)
2  while (true)
3    t = Top
4    n.next = t
5    if (CAS (Top,t,n))
6      exit
```



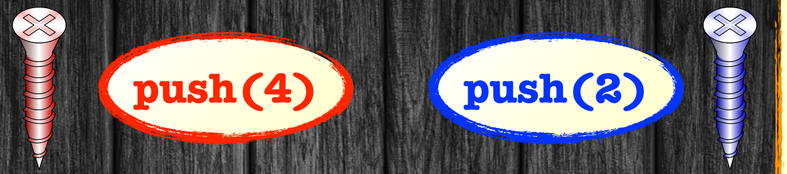

```
push(k):
Node t
1  n = new Node(k,-)
2  while (true)
3    t = Top
4    n.next = t
5    if (CAS (Top,t,n))
6      exit
```



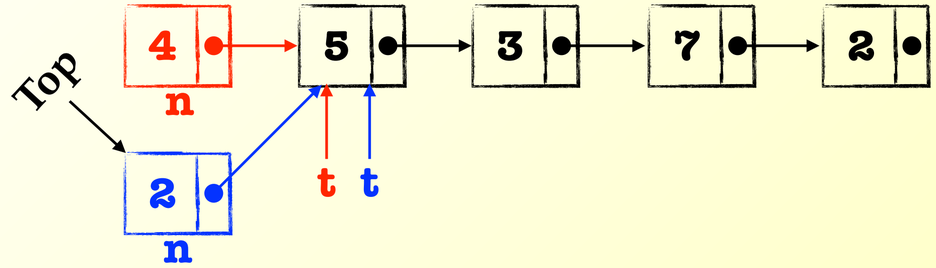
The Treiber Stack Algorithm

Concurrent push operations

```
push(k):
Node t
1  n = new Node(k,-)
2  while (true)
3    t = Top
4    n.next = t
5    if (CAS (Top,t,n))
6      exit
```




```
push(k):
Node t
1  n = new Node(k,-)
2  while (true)
3    t = Top
4    n.next = t
5    if (CAS (Top,t,n))
6      exit
```



The Treiber Stack Algorithm

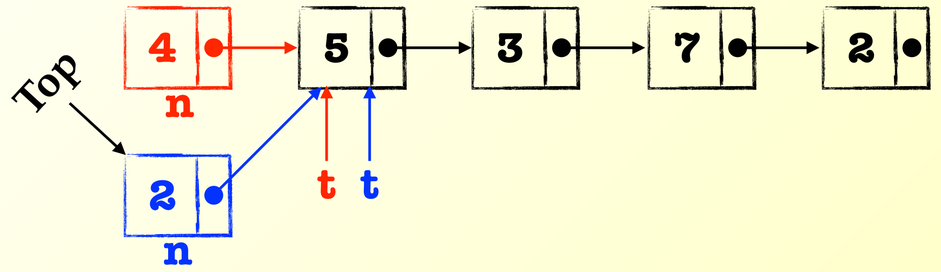
Concurrent push operations

```
push(k):
Node t
1  n = new Node(k,-)
2  while (true)
3    t = Top
4    n.next = t
5    if (CAS (Top,t,n))
6      exit
```

push(4)

push(2)

```
push(k):
Node t
1  n = new Node(k,-)
2  while (true)
3    t = Top
4    n.next = t
5    if (CAS (Top,t,n))
6      exit
```



The Treiber Stack Algorithm

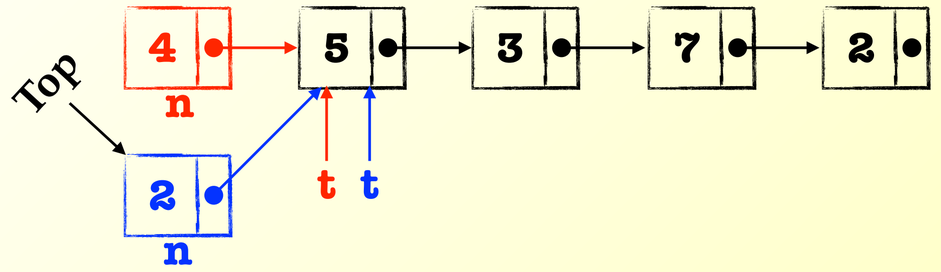
Concurrent push operations

```
push(k):
Node t
1  n = new Node(k,-)
2  while (true)
3    t = Top
4    n.next = t
5    if (CAS (Top,t,n))
6      exit
```

push(4)

push(2)

```
push(k):
Node t
1  n = new Node(k,-)
2  while (true)
3    t = Top
4    n.next = t
5    if (CAS (Top,t,n))
6      exit
```



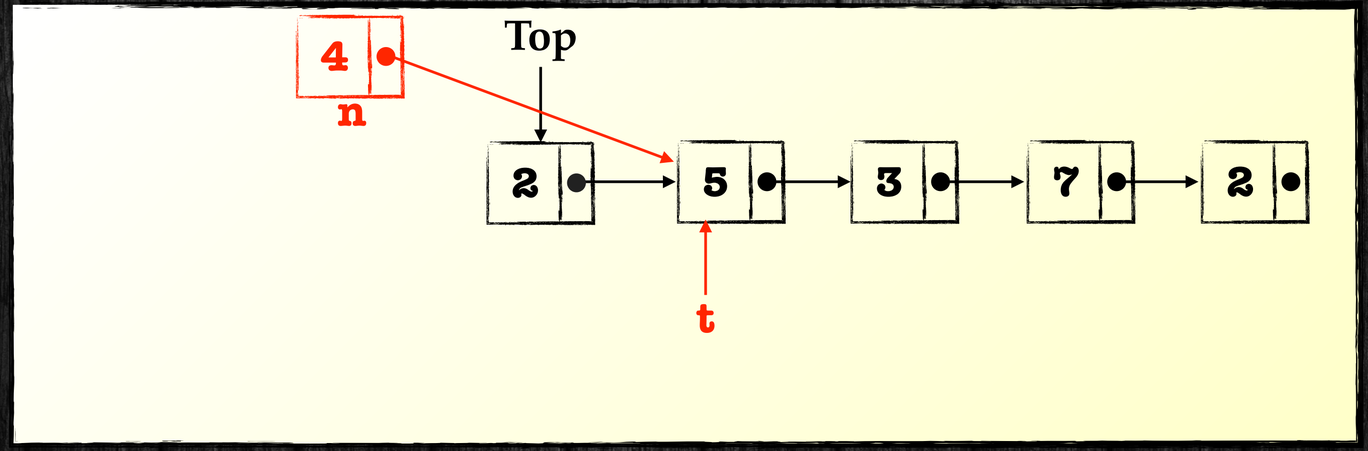
The Treiber Stack Algorithm

Concurrent push operations

```
push(k):
Node t
1  n = new Node(k,-)
2  while (true)
3    t = Top
4    n.next = t
5    if (CAS (Top,t,n))
6      exit
```



push(4)



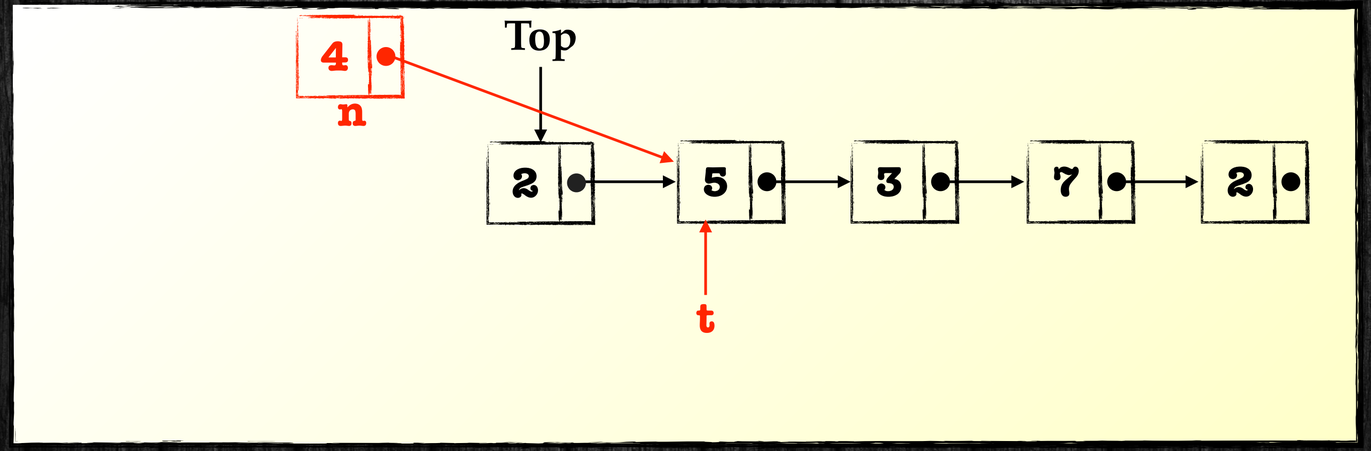
The Treiber Stack Algorithm

Concurrent push operations

```
push(k):
Node t
1  n = new Node(k,-)
2  while (true)
3    t = Top
4    n.next = t
5    if (CAS (Top,t,n))
6      exit
```



push(4)



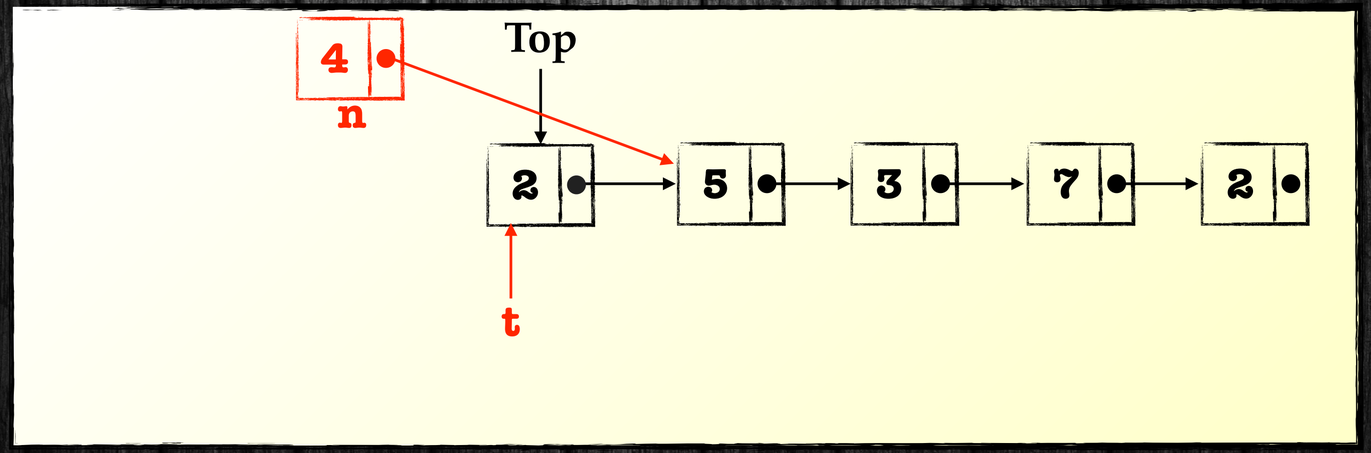
The Treiber Stack Algorithm

Concurrent push operations

```
push(k):
Node t
1  n = new Node(k,-)
2  while (true)
3    t = Top
4    n.next = t
5    if (CAS (Top,t,n))
6      exit
```



push(4)



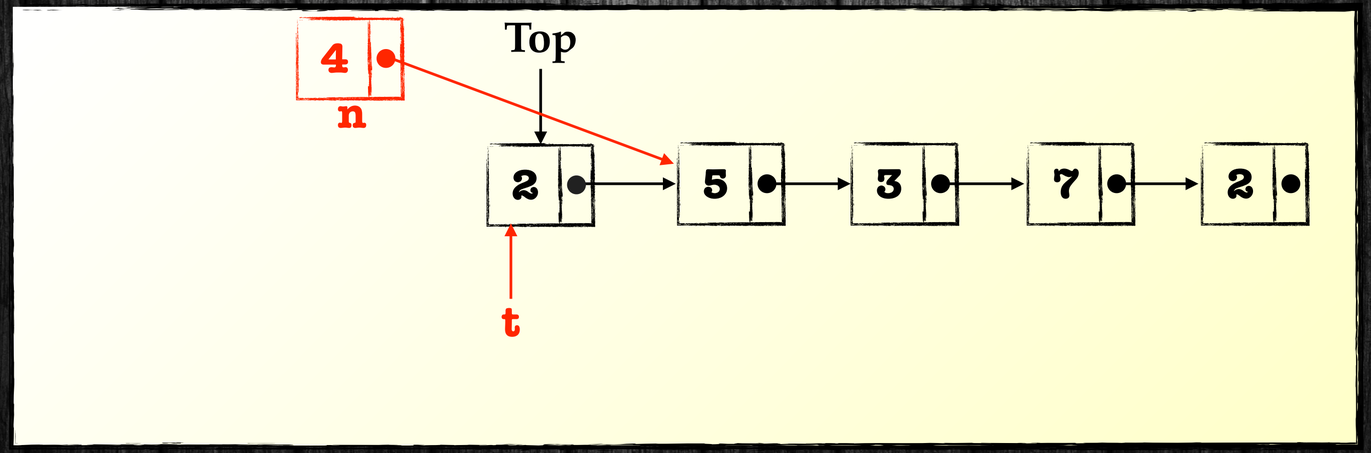
The Treiber Stack Algorithm

Concurrent push operations

```
push(k):
Node t
1  n = new Node(k,-)
2  while (true)
3    t = Top
4    n.next = t
5    if (CAS (Top,t,n))
6      exit
```



push(4)



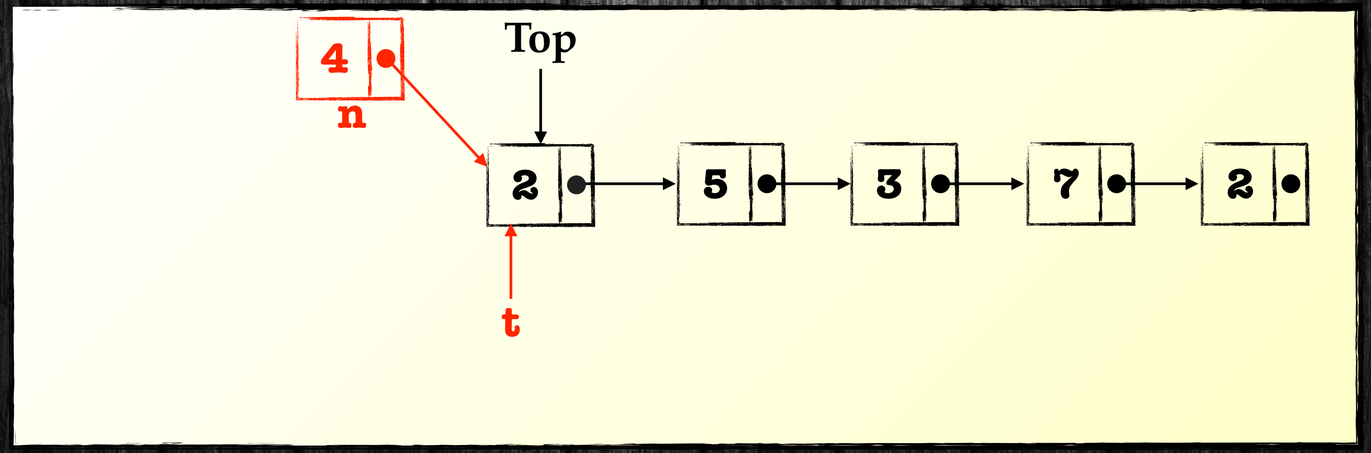
The Treiber Stack Algorithm

Concurrent push operations

```
push(k):
Node t
1  n = new Node(k,-)
2  while (true)
3    t = Top
4    n.next = t
5    if (CAS (Top,t,n))
6      exit
```



push(4)



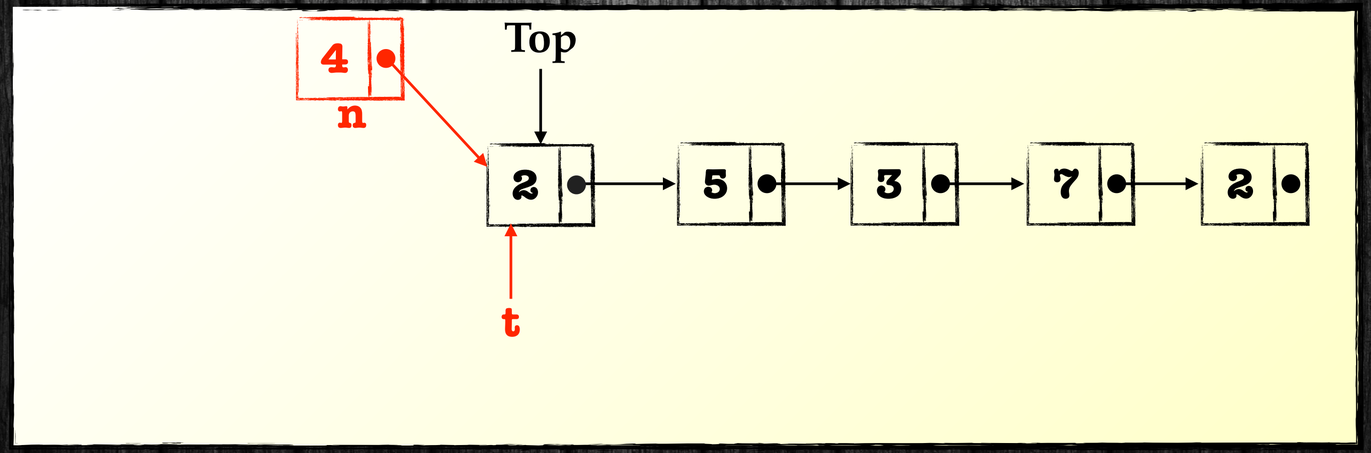
The Treiber Stack Algorithm

Concurrent push operations

```
push(k):
Node t
1  n = new Node(k,-)
2  while (true)
3    t = Top
4    n.next = t
5    if (CAS (Top,t,n))
6      exit
```



push(4)

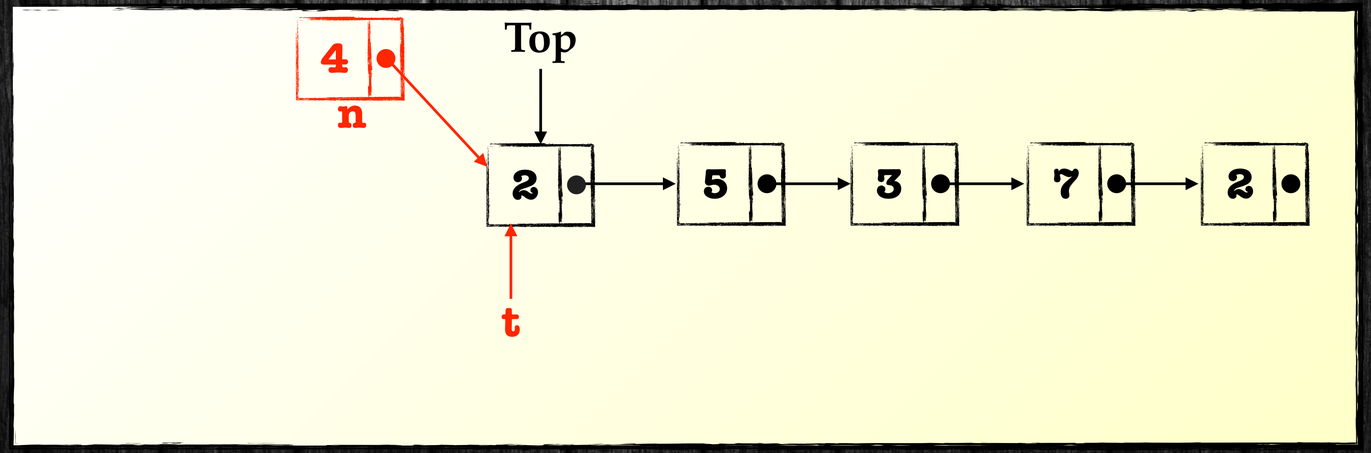


The Treiber Stack Algorithm

Concurrent push operations

```
push(k):
Node t
1  n = new Node(k,-)
2  while (true)
3    t = Top
4    n.next = t
5    if (CAS (Top,t,n))
6      exit
```





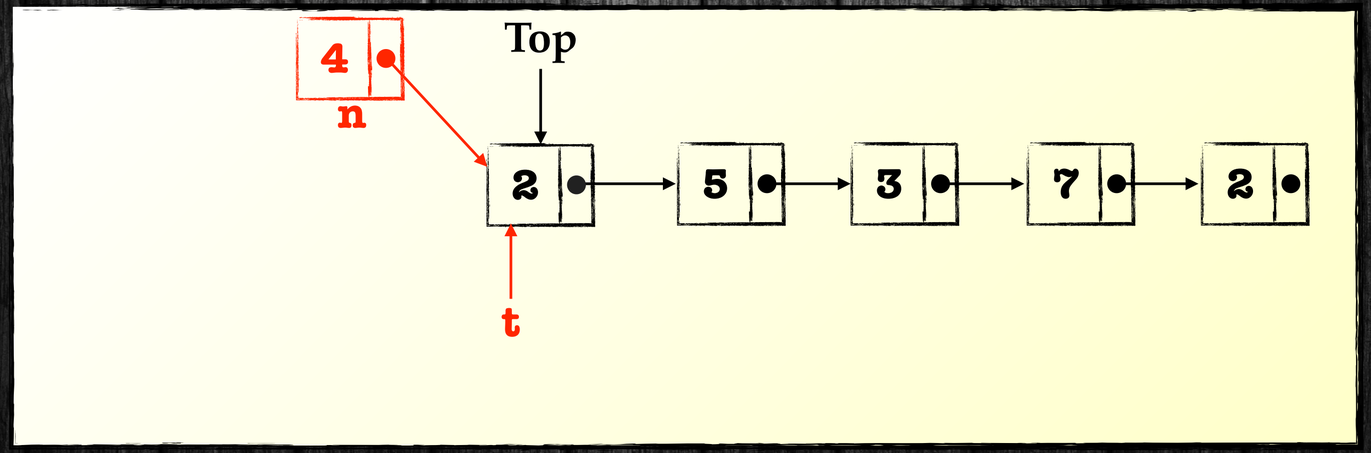
The Treiber Stack Algorithm

Concurrent push operations

```
push(k):
Node t
1  n = new Node(k,-)
2  while (true)
3    t = Top
4    n.next = t
5    if (CAS (Top,t,n))
6      exit
```



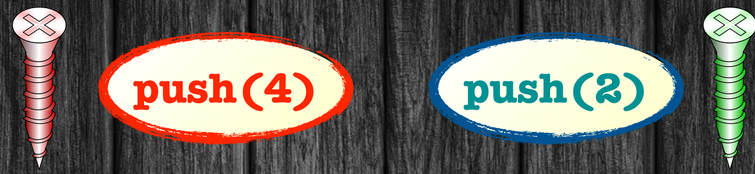

```
push(k):
Node t
1  n = new Node(k,-)
2  while (true)
3    t = Top
4    n.next = t
5    if (CAS (Top,t,n))
6      exit
```



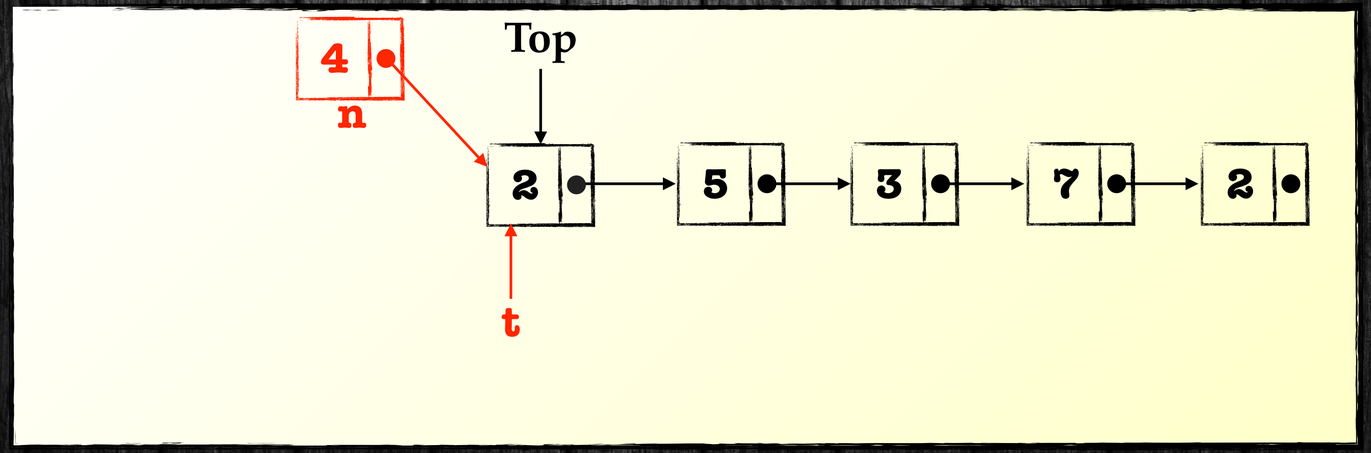
The Treiber Stack Algorithm

Concurrent push operations

```
push(k):
Node t
1  n = new Node(k,-)
2  while (true)
3    t = Top
4    n.next = t
5    if (CAS (Top,t,n))
6      exit
```




```
push(k):
Node t
1  n = new Node(k,-)
2  while (true)
3    t = Top
4    n.next = t
5    if (CAS (Top,t,n))
6      exit
```



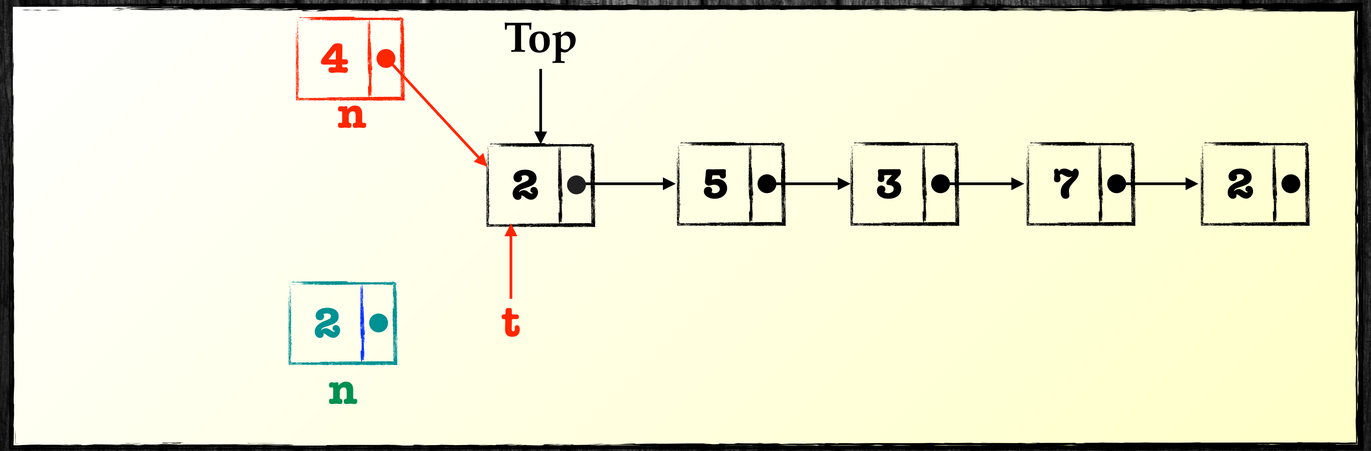
The Treiber Stack Algorithm

Concurrent push operations

```
push(k):
Node t
1  n = new Node(k,-)
2  while (true)
3    t = Top
4    n.next = t
5    if (CAS (Top,t,n))
6      exit
```




```
push(k):
Node t
1  n = new Node(k,-)
2  while (true)
3    t = Top
4    n.next = t
5    if (CAS (Top,t,n))
6      exit
```



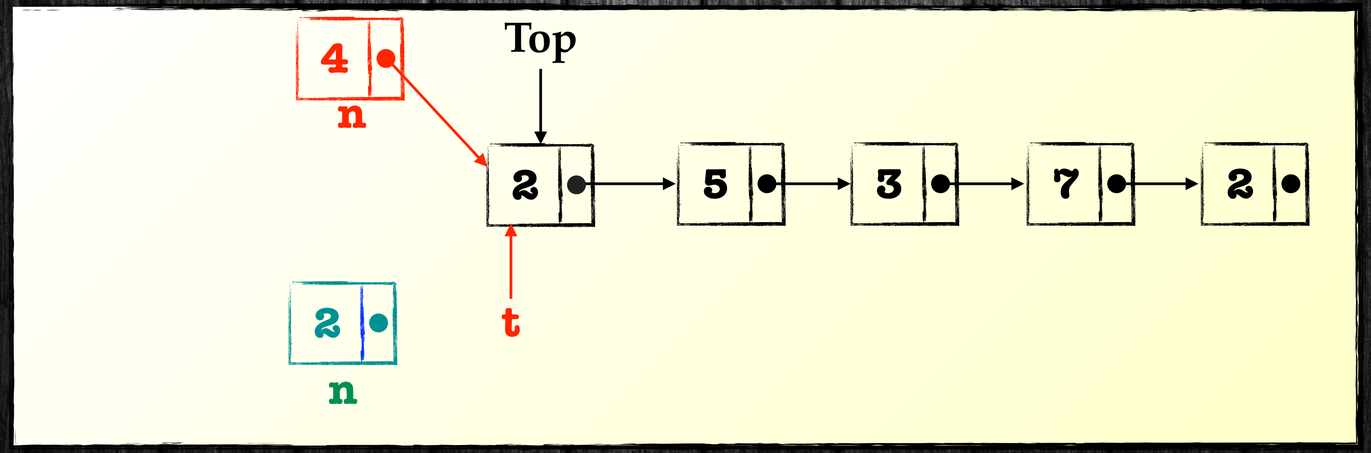
The Treiber Stack Algorithm

Concurrent push operations

```
push(k):
Node t
1  n = new Node(k,-)
2  while (true)
3    t = Top
4    n.next = t
5    if (CAS (Top,t,n))
6      exit
```




```
push(k):
Node t
1  n = new Node(k,-)
2  while (true)
3    t = Top
4    n.next = t
5    if (CAS (Top,t,n))
6      exit
```



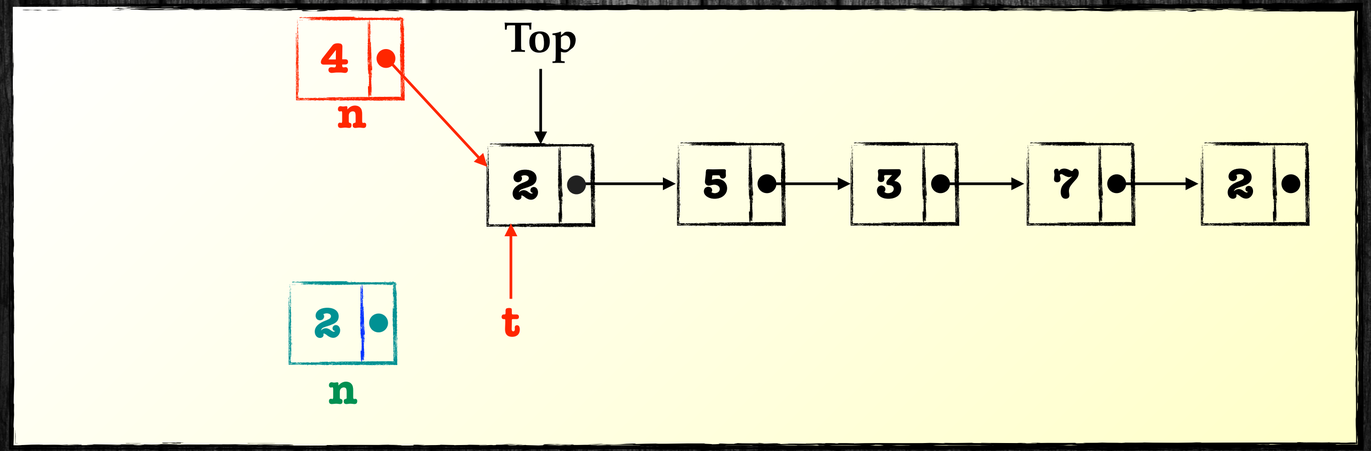
The Treiber Stack Algorithm

Concurrent push operations

```
push(k):
Node t
1  n = new Node(k,-)
2  while (true)
3    t = Top
4    n.next = t
5    if (CAS (Top,t,n))
6      exit
```




```
push(k):
Node t
1  n = new Node(k,-)
2  while (true)
3    t = Top
4    n.next = t
5    if (CAS (Top,t,n))
6      exit
```



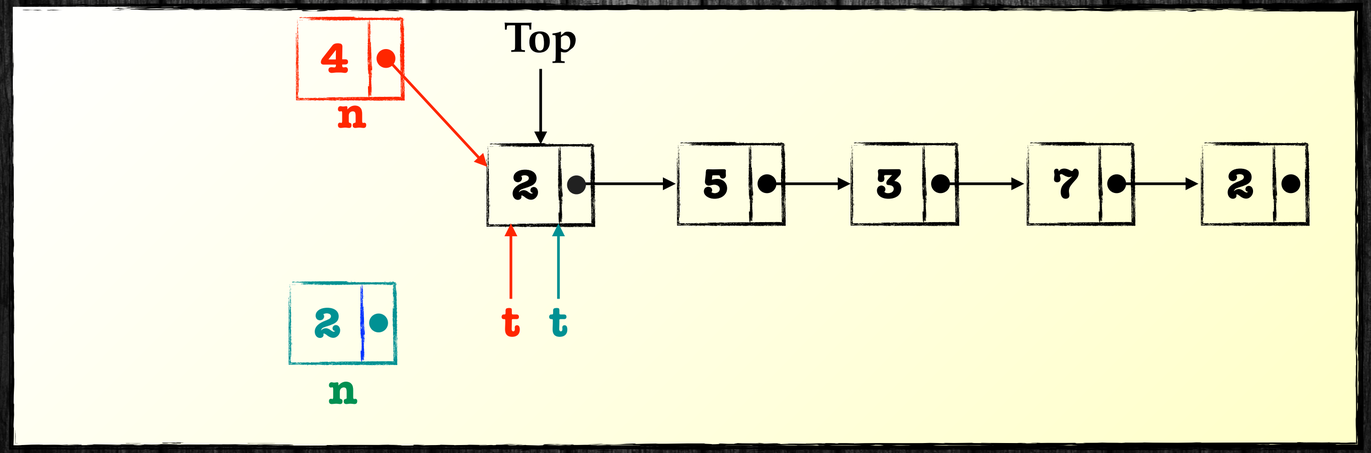
The Treiber Stack Algorithm

Concurrent push operations

```
push(k):
Node t
1  n = new Node(k,-)
2  while (true)
3    t = Top
4    n.next = t
5    if (CAS (Top,t,n))
6      exit
```




```
push(k):
Node t
1  n = new Node(k,-)
2  while (true)
3    t = Top
4    n.next = t
5    if (CAS (Top,t,n))
6      exit
```



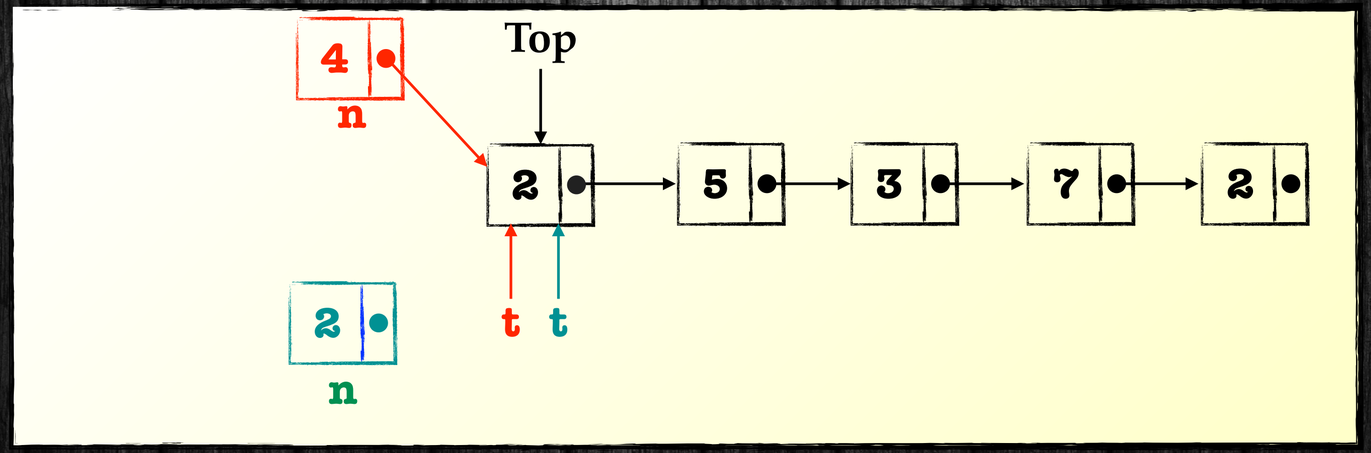
The Treiber Stack Algorithm

Concurrent push operations

```
push(k):
Node t
1  n = new Node(k,-)
2  while (true)
3    t = Top
4    n.next = t
5    if (CAS (Top,t,n))
6      exit
```




```
push(k):
Node t
1  n = new Node(k,-)
2  while (true)
3    t = Top
4    n.next = t
5    if (CAS (Top,t,n))
6      exit
```



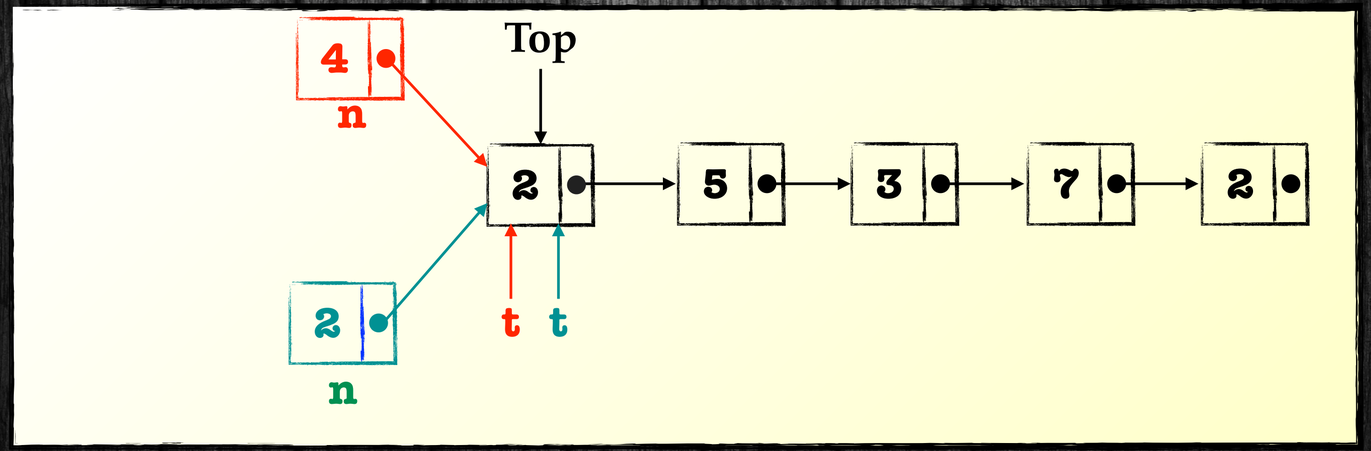
The Treiber Stack Algorithm

Concurrent push operations

```
push(k):
Node t
1  n = new Node(k,-)
2  while (true)
3    t = Top
4    n.next = t
5    if (CAS (Top,t,n))
6      exit
```




```
push(k):
Node t
1  n = new Node(k,-)
2  while (true)
3    t = Top
4    n.next = t
5    if (CAS (Top,t,n))
6      exit
```



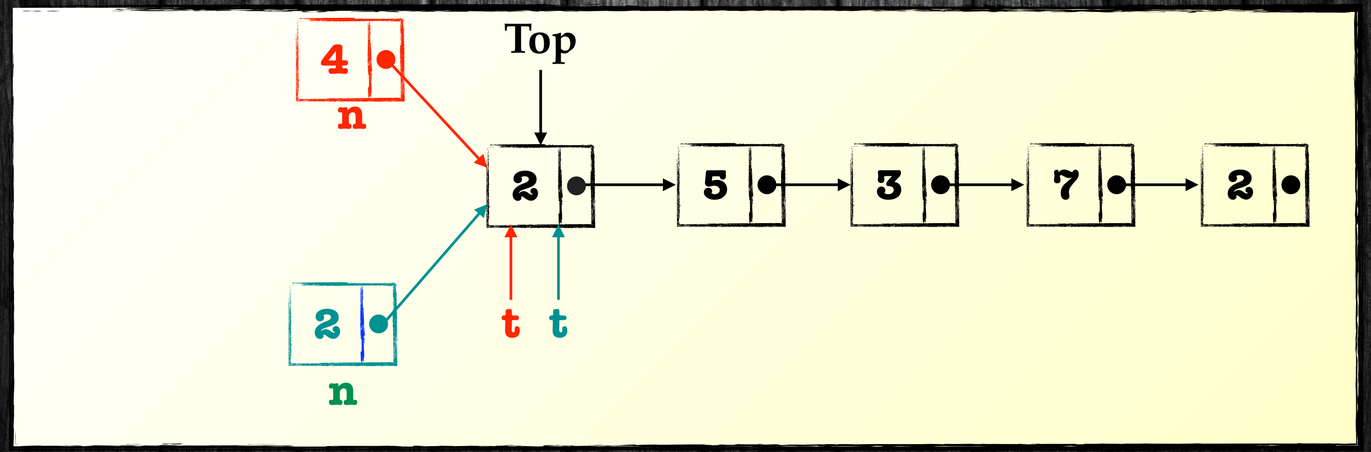
The Treiber Stack Algorithm

Concurrent push operations

```
push(k):
Node t
1  n = new Node(k,-)
2  while (true)
3    t = Top
4    n.next = t
5    if (CAS (Top,t,n))
6      exit
```




```
push(k):
Node t
1  n = new Node(k,-)
2  while (true)
3    t = Top
4    n.next = t
5    if (CAS (Top,t,n))
6      exit
```



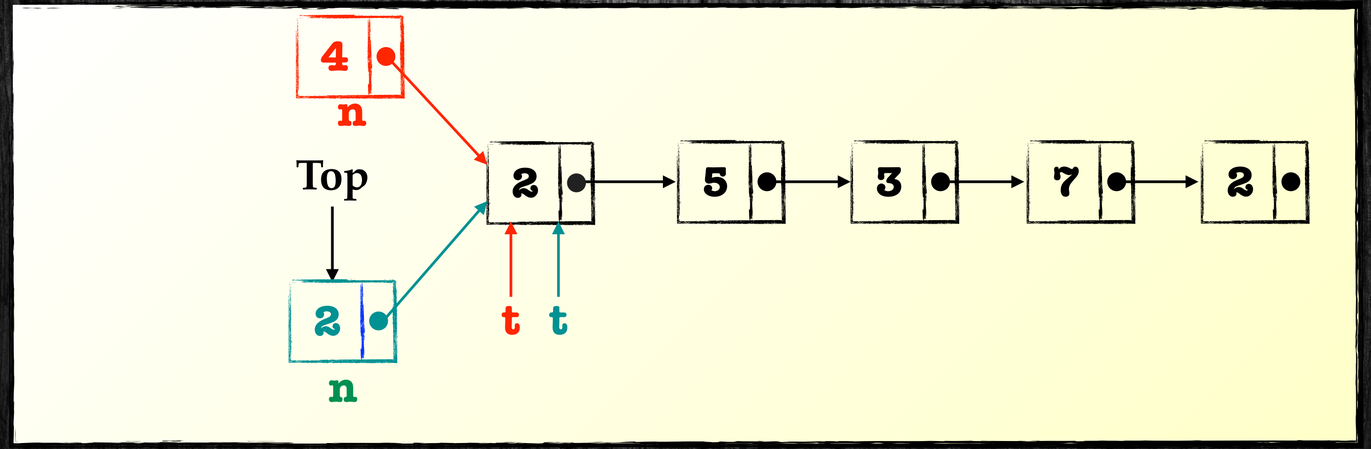
The Treiber Stack Algorithm

Concurrent push operations

```
push(k):
Node t
1  n = new Node(k,-)
2  while (true)
3    t = Top
4    n.next = t
5    if (CAS (Top,t,n))
6      exit
```




```
push(k):
Node t
1  n = new Node(k,-)
2  while (true)
3    t = Top
4    n.next = t
5    if (CAS (Top,t,n))
6      exit
```



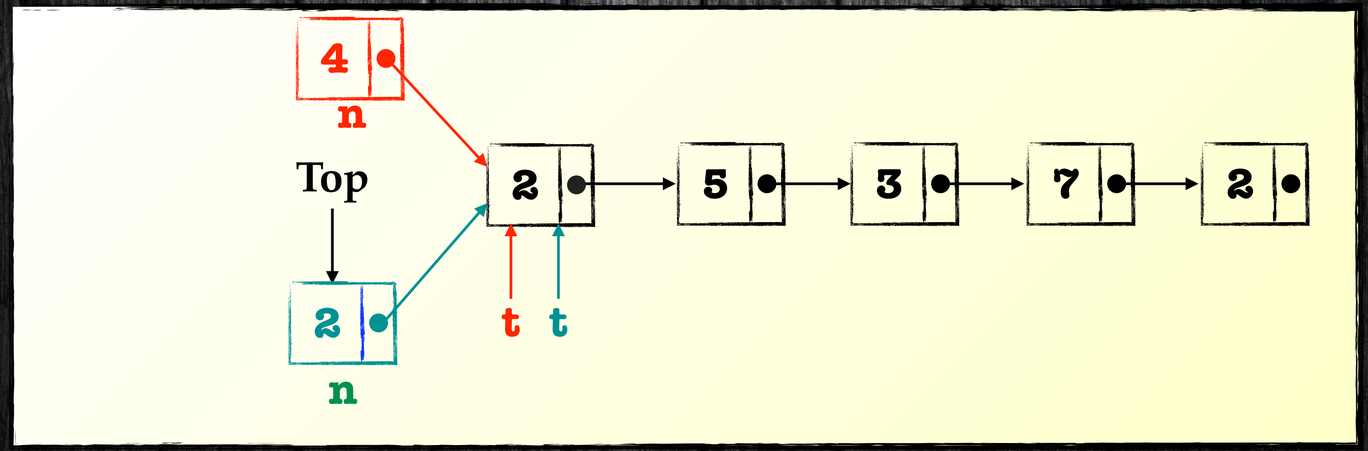
The Treiber Stack Algorithm

Concurrent push operations

```
push(k):
Node t
1  n = new Node(k,-)
2  while (true)
3    t = Top
4    n.next = t
5    if (CAS (Top,t,n))
6      exit
```




```
push(k):
Node t
1  n = new Node(k,-)
2  while (true)
3    t = Top
4    n.next = t
5    if (CAS (Top,t,n))
6      exit
```



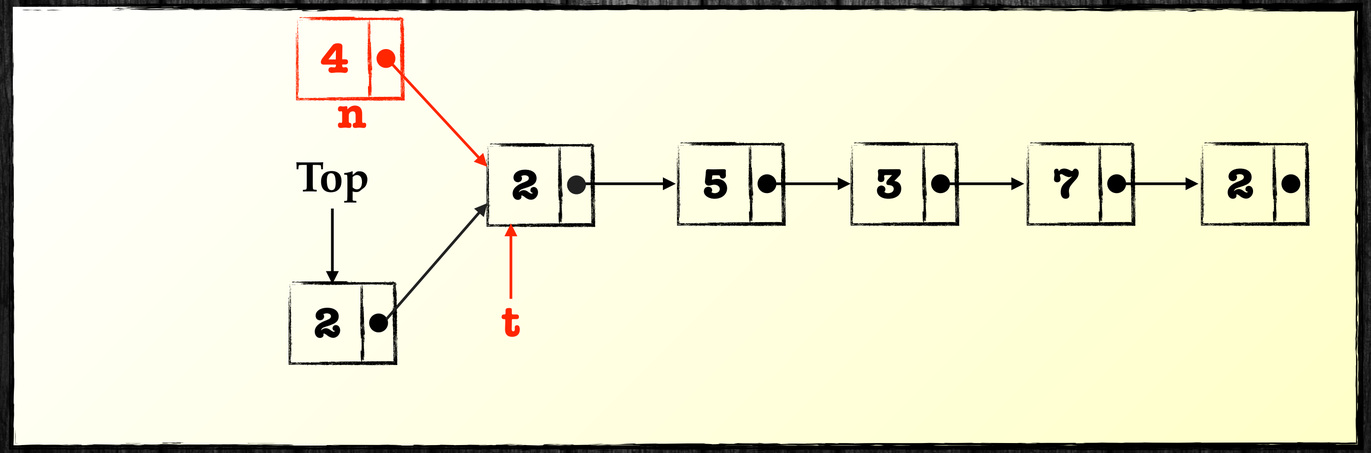
The Treiber Stack Algorithm

Concurrent push operations

```
push(k):
Node t
1  n = new Node(k,-)
2  while (true)
3    t = Top
4    n.next = t
5    if (CAS (Top,t,n))
6      exit
```



push(4)



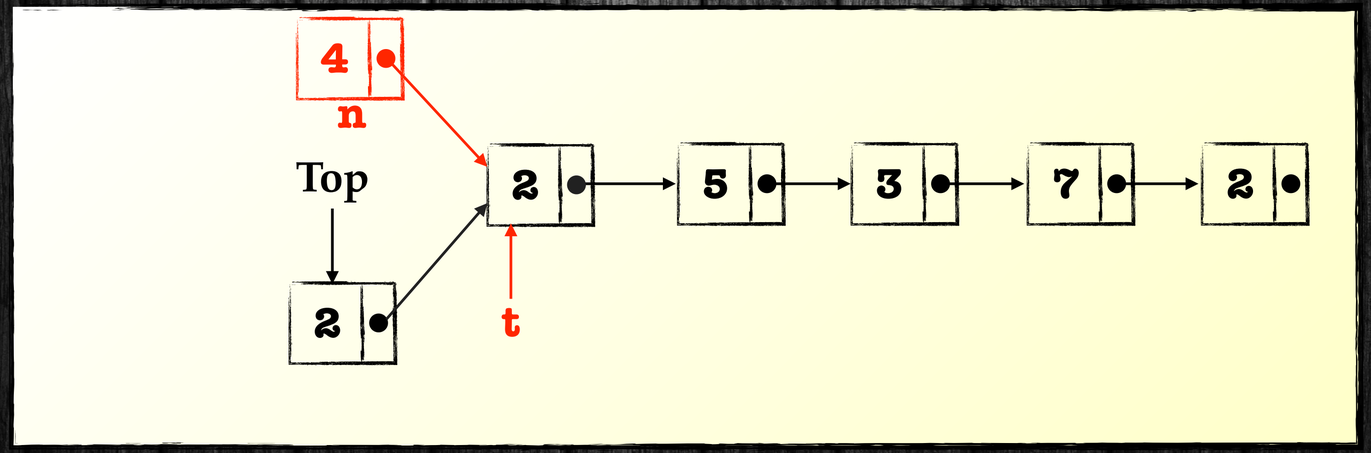
The Treiber Stack Algorithm

Concurrent push operations

```
push(k):
Node t
1  n = new Node(k,-)
2  while (true)
3    t = Top
4    n.next = t
5    if (CAS (Top,t,n))
6      exit
```



push(4)

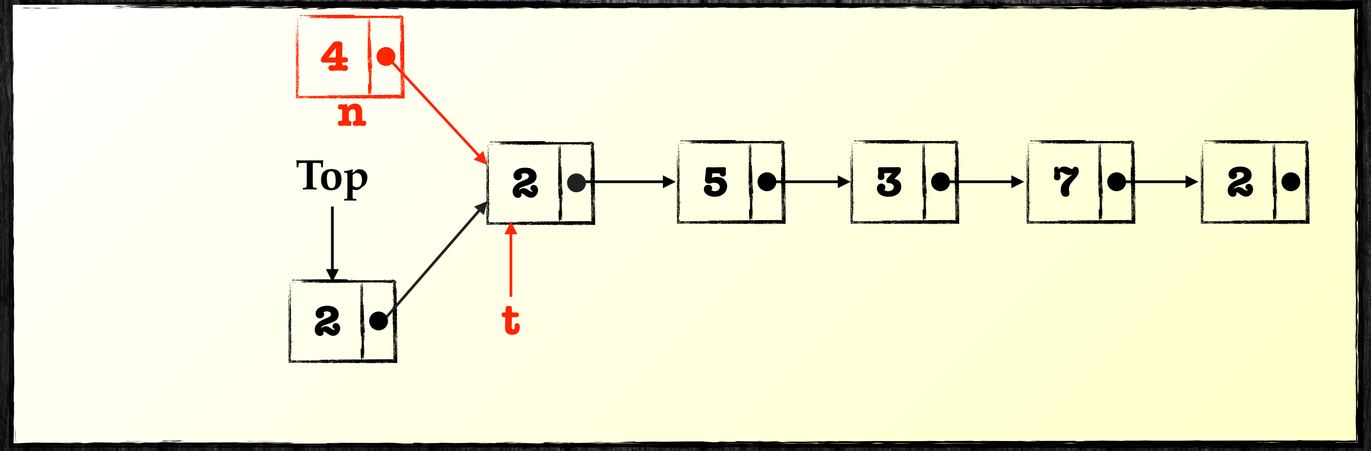


The Treiber Stack Algorithm

```
push(k):
Node t
1  n = new Node(k,-)
2  while (true)
3    t = Top
4    n.next = t
5    if (CAS (Top,t,n))
6      exit
```



push(4)



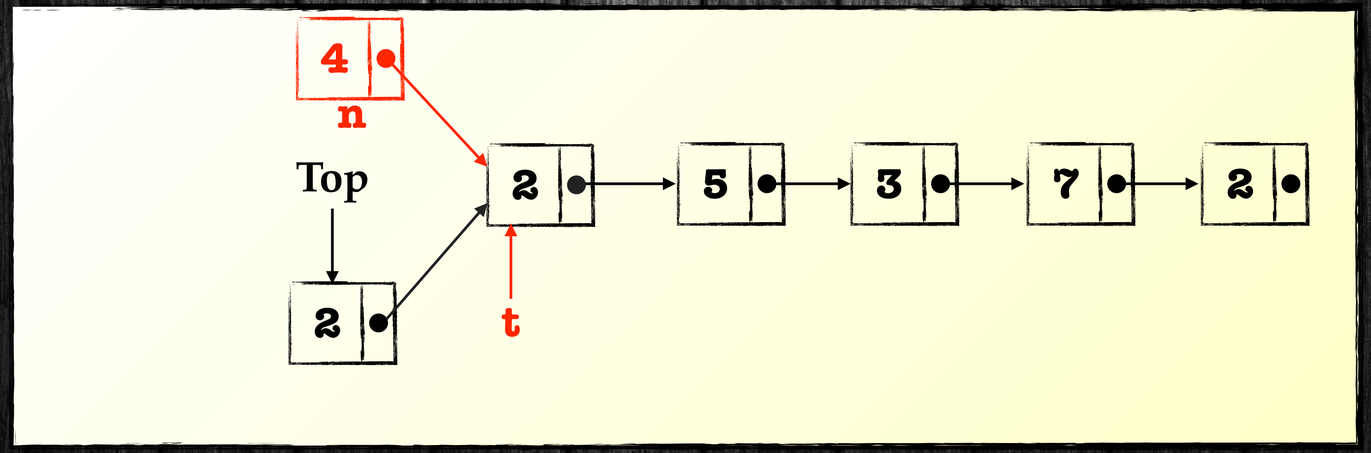
The Treiber Stack Algorithm

```
push(k):
Node t
1  n = new Node(k,-)
2  while (true)
3    t = Top
4    n.next = t
5    if (CAS (Top,t,n))
6      exit
```



push(4)

not wait-free (starvation)



The Treiber Stack Algorithm

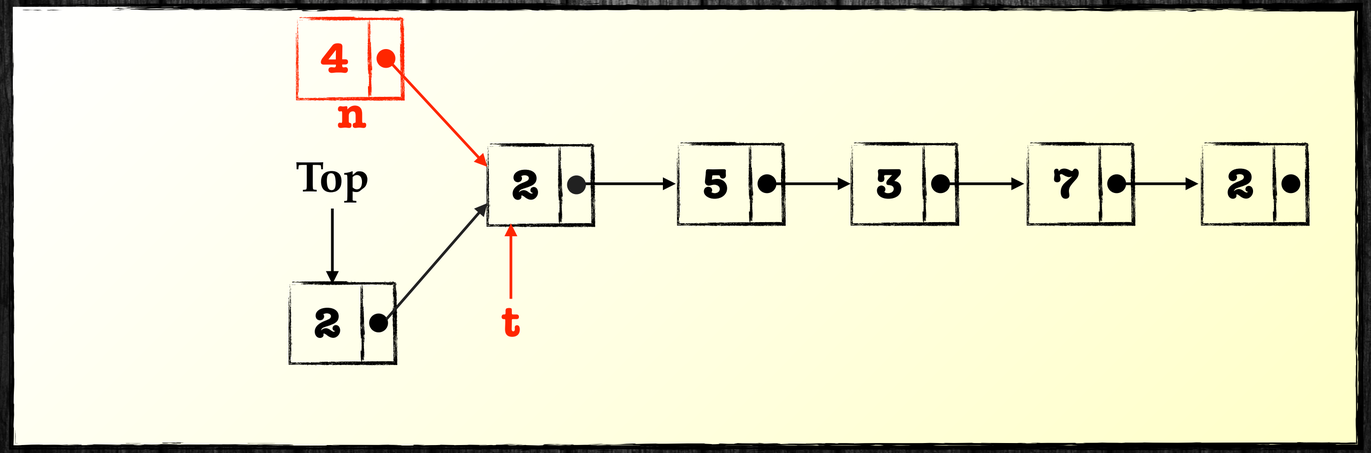
```
push(k):
Node t
1  n = new Node(k,-)
2  while (true)
3    t = Top
4    n.next = t
5    if (CAS (Top,t,n))
6      exit
```



push(4)

not wait-free (starvation)

lock-free



The Treiber Stack Algorithm

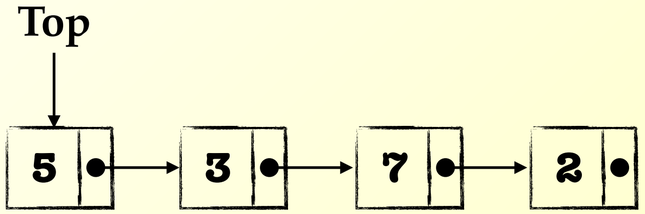
```
push(k):
Node t
1  n = new Node(k,-)
2  while (true)
3    t = Top
4    n.next = t
5    if (CAS (Top,t,n))
6      exit
```



push(4)

not wait-free (starvation)

lock-free



The Treiber Stack Algorithm

```
push(k):
Node t
1  n = new Node(k,-)
2  while (true)
3    t = Top
4    n.next = t
5    if (CAS (Top,t,n))
6      exit
```

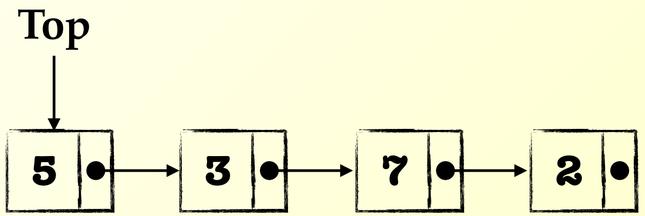


push(4)

not wait-free (starvation)

lock-free

essentially sequential



The Treiber Stack Algorithm

```
push(k):
Node t
1  n = new Node(k,-)
2  while (true)
3    t = Top
4    n.next = t
5    if (CAS (Top,t,n))
6      exit
```

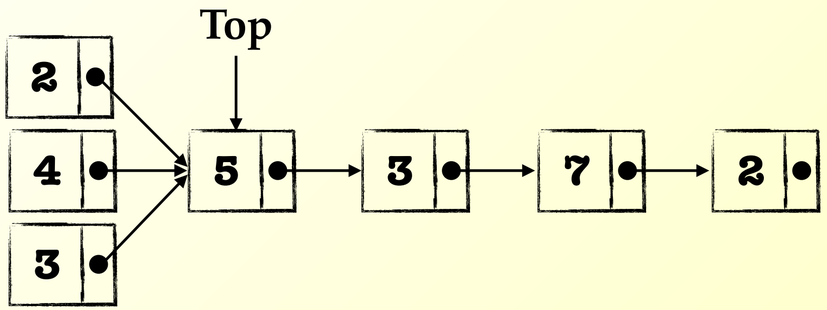


push(4)

not wait-free (starvation)

lock-free

essentially sequential



The Treiber Stack Algorithm

```
push(k):
Node t
1  n = new Node(k,-)
2  while (true)
3    t = Top
4    n.next = t
5    if (CAS (Top,t,n))
6      exit
```

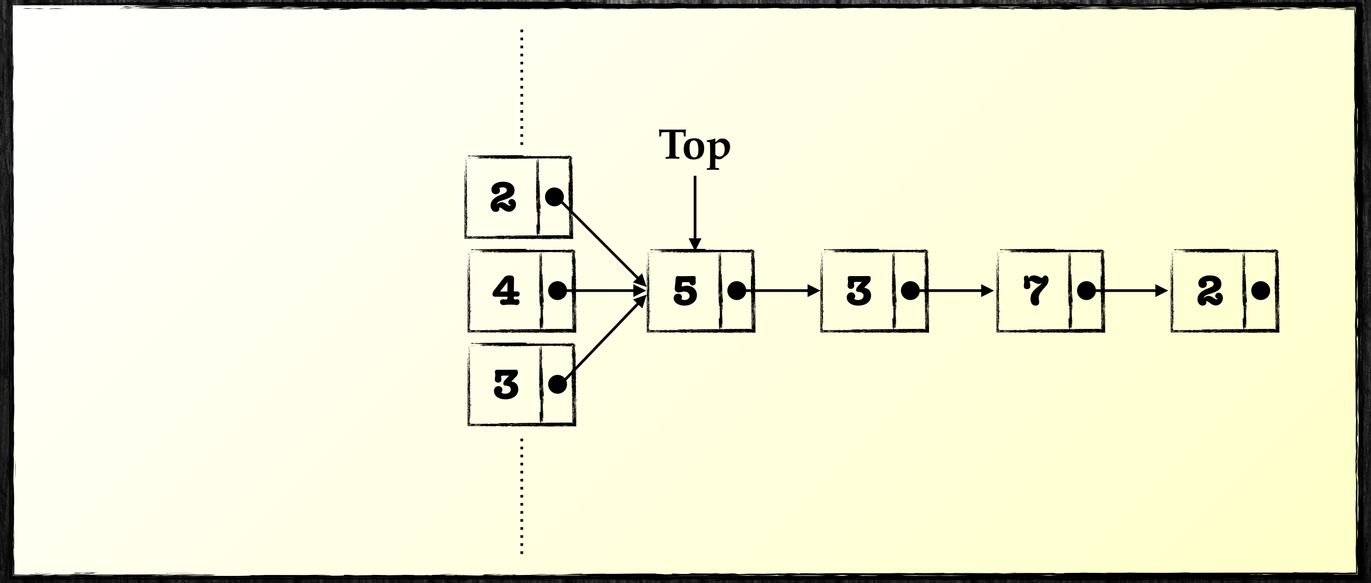


push(4)

not wait-free (starvation)

lock-free

essentially sequential



The Treiber Stack Algorithm

```
push(k):
Node t
1  n = new Node(k,-)
2  while (true)
3    t = Top
4    n.next = t
5    if (CAS (Top,t,n))
6      exit
```

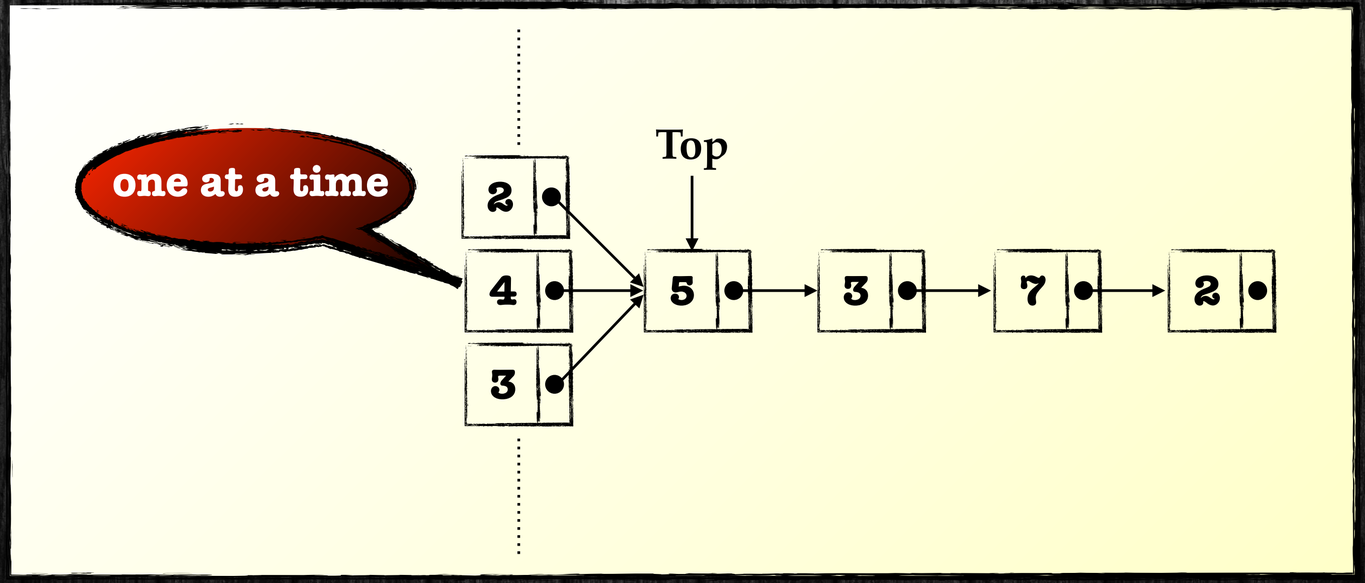


push(4)

not wait-free (starvation)

lock-free

essentially sequential



The Treiber Stack Algorithm

```
push(k):
Node t
1  n = new Node(k,-)
2  while (true)
3    t = Top
4    n.next = t
5    if (CAS (Top,t,n))
6      exit
```



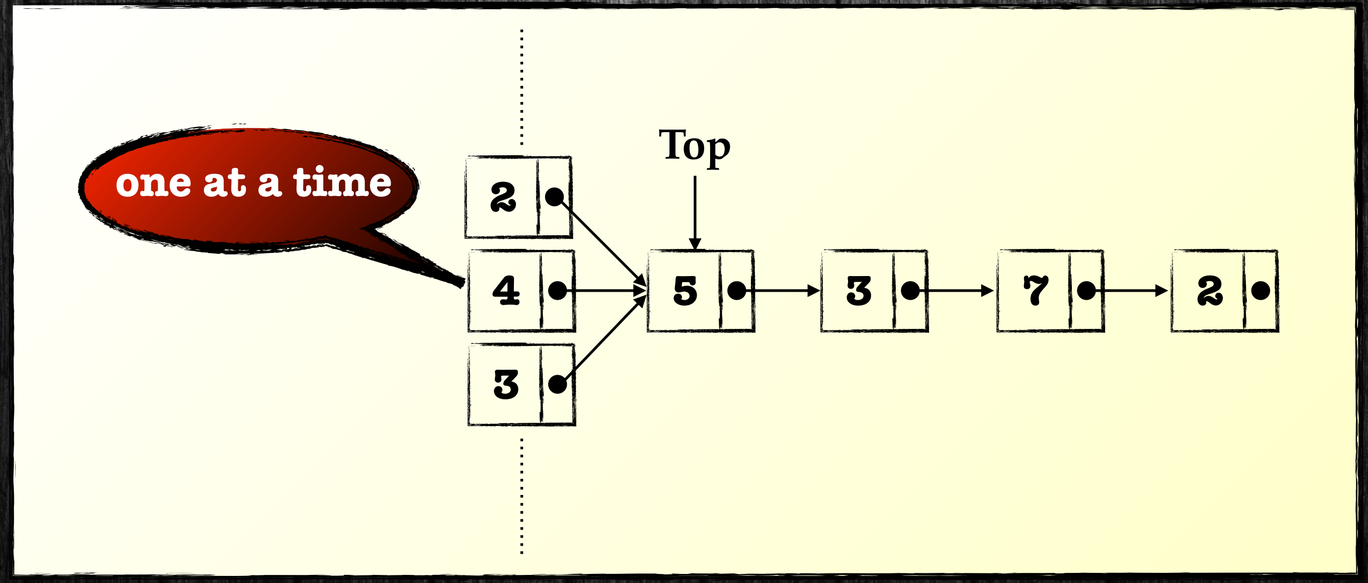
push(4)

not wait-free (starvation)

lock-free

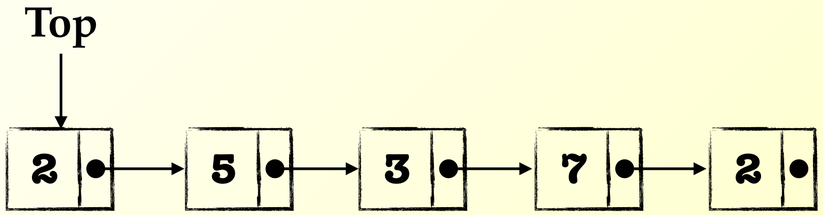


essentially sequential



The Treiber Stack Algorithm

```
pop:
Node t
1  while (true)
2    t = Top
3    if (t = NULL)
4      return *
5      exit
6    if (CAS (Top,t,t.next))
7      return t.val
8      exit
```

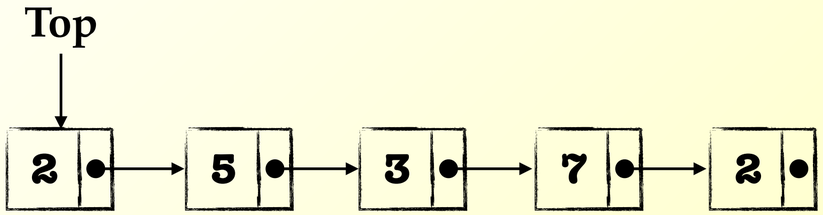


The Treiber Stack Algorithm

```
pop:
Node t
1  while (true)
2    t = Top
3    if (t = NULL)
4      return *
5      exit
6    if (CAS (Top,t,t.next))
7      return t.val
8      exit
```



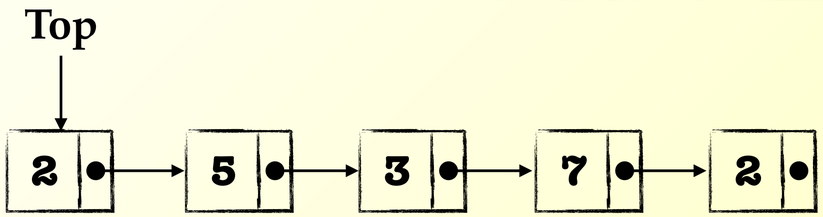
pop



The Treiber Stack Algorithm

```
pop:
Node t
1  while (true)
2    t = Top
3    if (t = NULL)
4      return *
5      exit
6    if (CAS (Top,t,t.next))
7      return t.val
8      exit
```



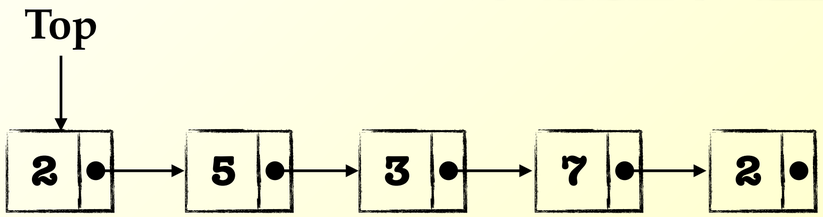


The Treiber Stack Algorithm

```
pop:
Node t
1  while (true)
2    t = Top
3    if (t = NULL)
4      return *
5      exit
6    if (CAS (Top,t,t.next))
7      return t.val
8      exit
```



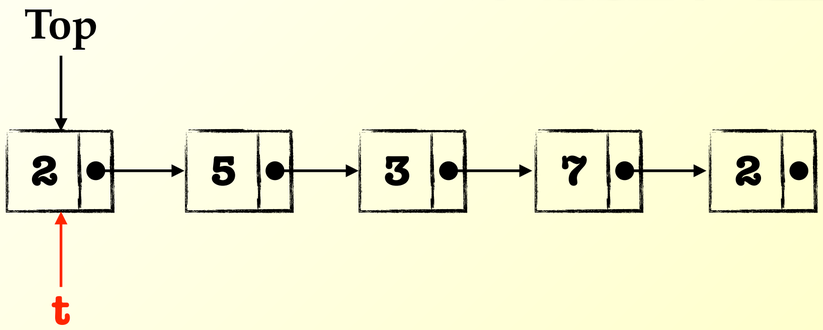
pop



The Treiber Stack Algorithm

```
pop:
Node t
1  while (true)
2    t = Top
3    if (t = NULL)
4      return *
5      exit
6    if (CAS (Top,t,t.next))
7      return t.val
8      exit
```



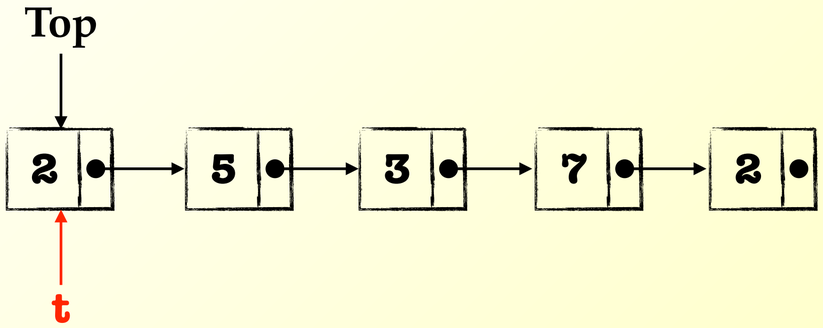


The Treiber Stack Algorithm

```
pop:
Node t
1  while (true)
2    t = Top
3    if (t = NULL)
4      return *
5      exit
6    if (CAS (Top,t,t.next))
7      return t.val
8      exit
```



pop

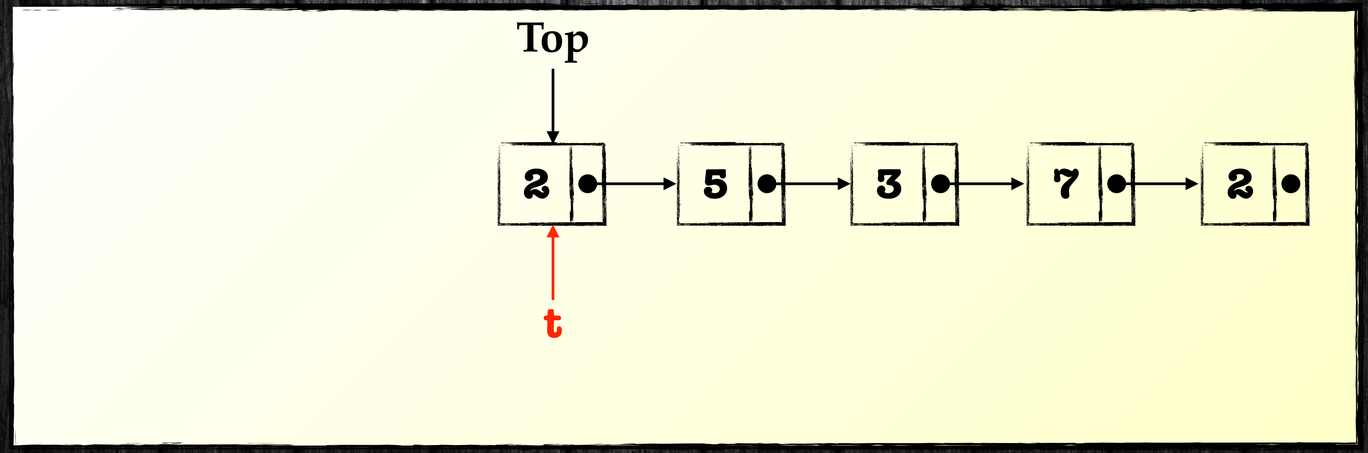


The Treiber Stack Algorithm

```
pop:
Node t
1  while (true)
2    t = Top
3    if (t = NULL)
4      return *
5      exit
6    if (CAS (Top,t,t.next))
7      return t.val
8      exit
```



pop

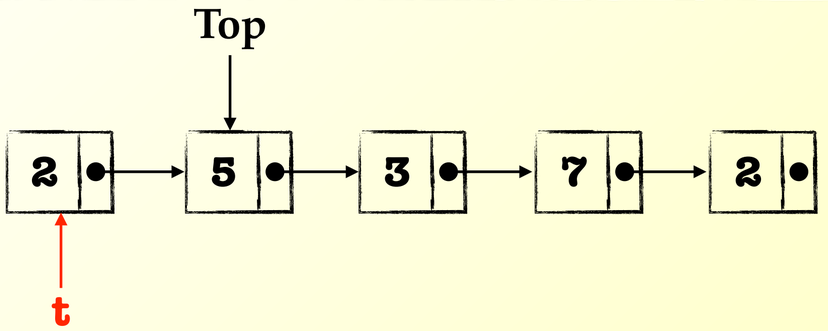


The Treiber Stack Algorithm

```
pop:
Node t
1  while (true)
2    t = Top
3    if (t = NULL)
4      return *
5      exit
6    if (CAS (Top,t,t.next))
7      return t.val
8      exit
```



pop

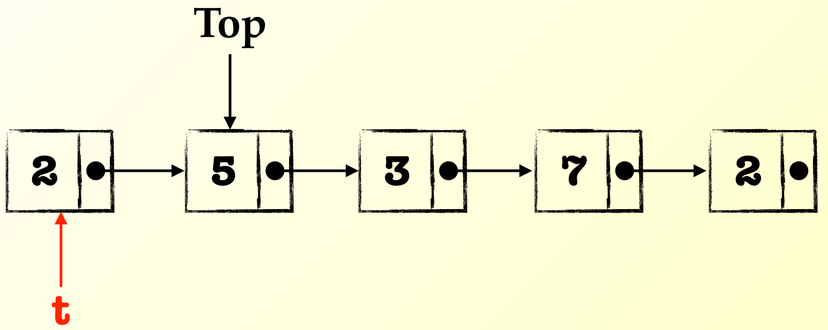


The Treiber Stack Algorithm

```
pop:
Node t
1  while (true)
2    t = Top
3    if (t = NULL)
4      return *
5      exit
6    if (CAS (Top,t,t.next))
7      return t.val
8      exit
```



pop

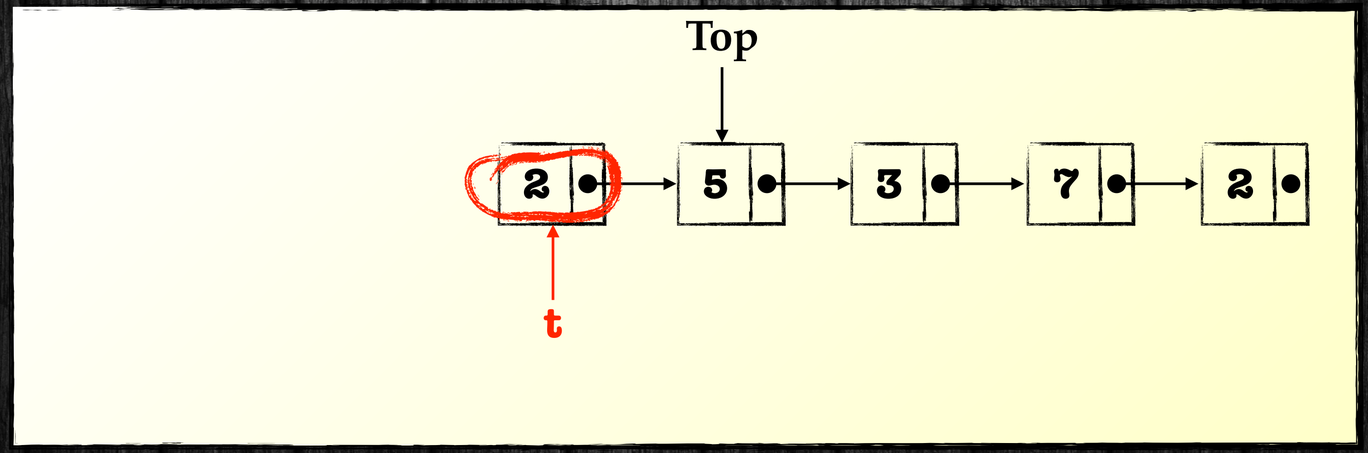


The Treiber Stack Algorithm

```
pop:
Node t
1  while (true)
2    t = Top
3    if (t = NULL)
4      return *
5      exit
6    if (CAS (Top,t,t.next))
7      return t.val
8      exit
```

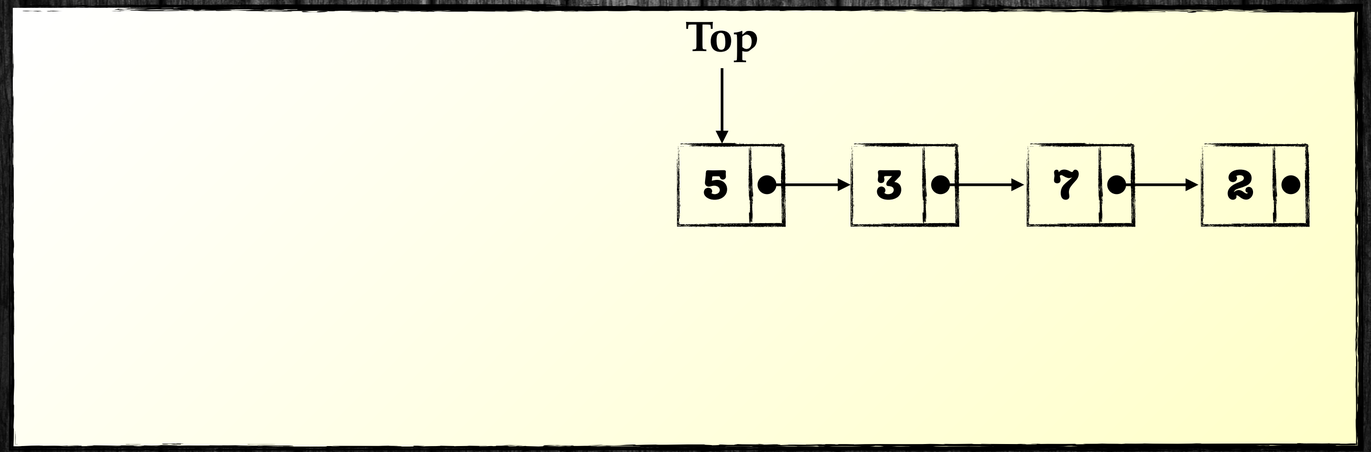


pop



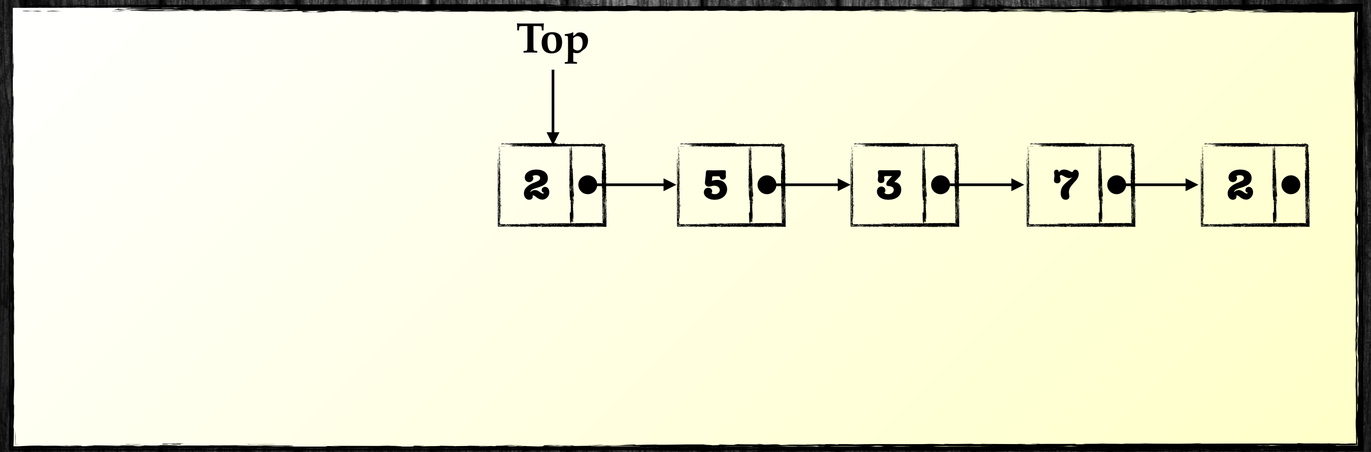
The Treiber Stack Algorithm

Concurrent pop operations



The Treiber Stack Algorithm

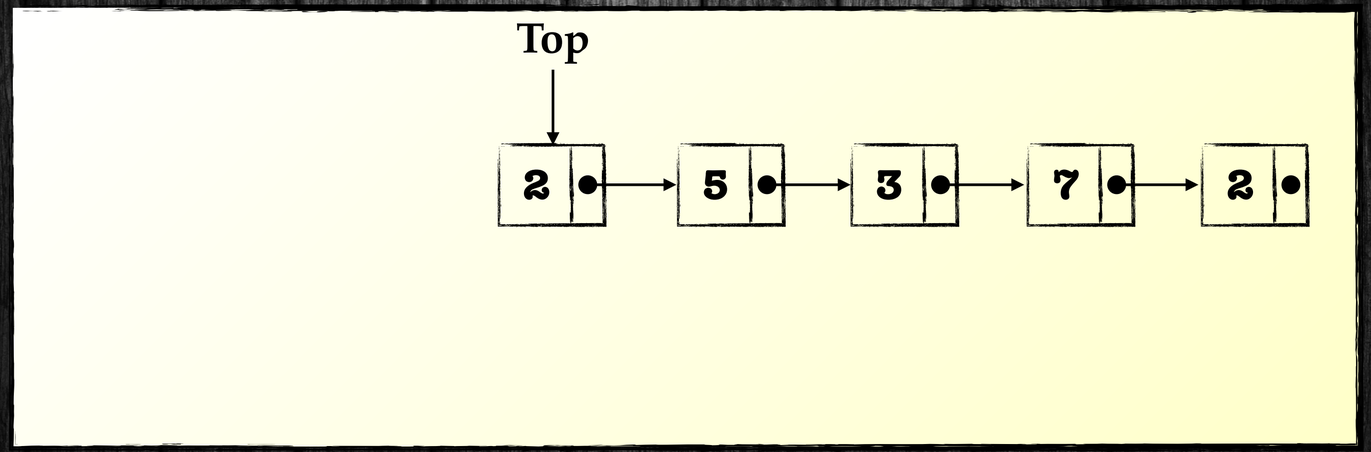
Concurrent pop operations



The Treiber Stack Algorithm

Concurrent pop operations

```
pop:
Node t
1  while (true)
2    t = Top
3    if (t = NULL)
4      return *
5      exit
6    if (CAS (Top,t,t.next))
7      return t.val
8      exit
```

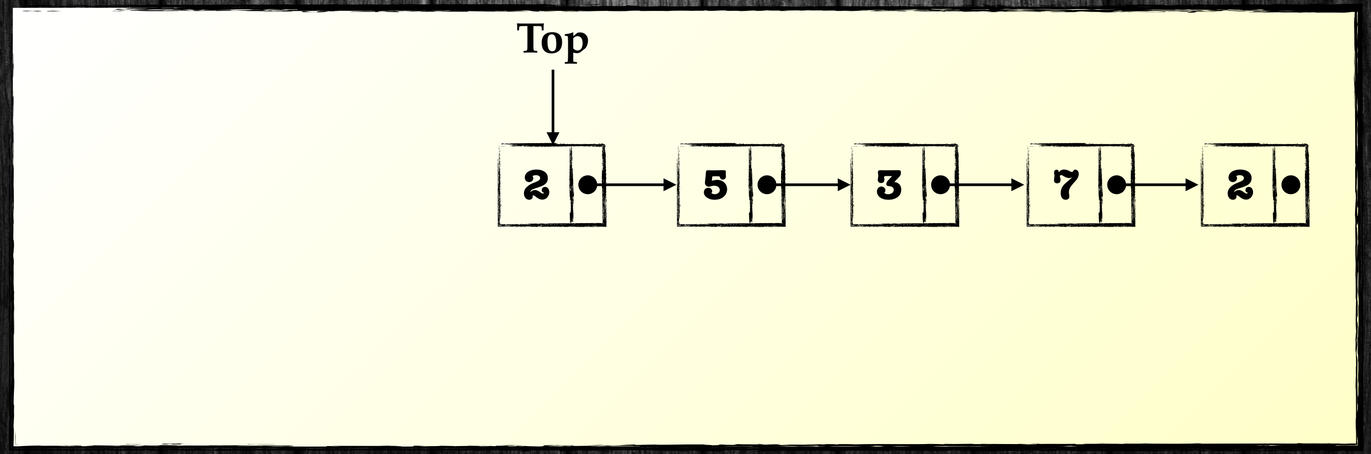


The Treiber Stack Algorithm

Concurrent pop operations

```
pop:
Node t
1  while (true)
2    t = Top
3    if (t = NULL)
4      return *
5      exit
6    if (CAS (Top,t,t.next))
7      return t.val
8      exit
```



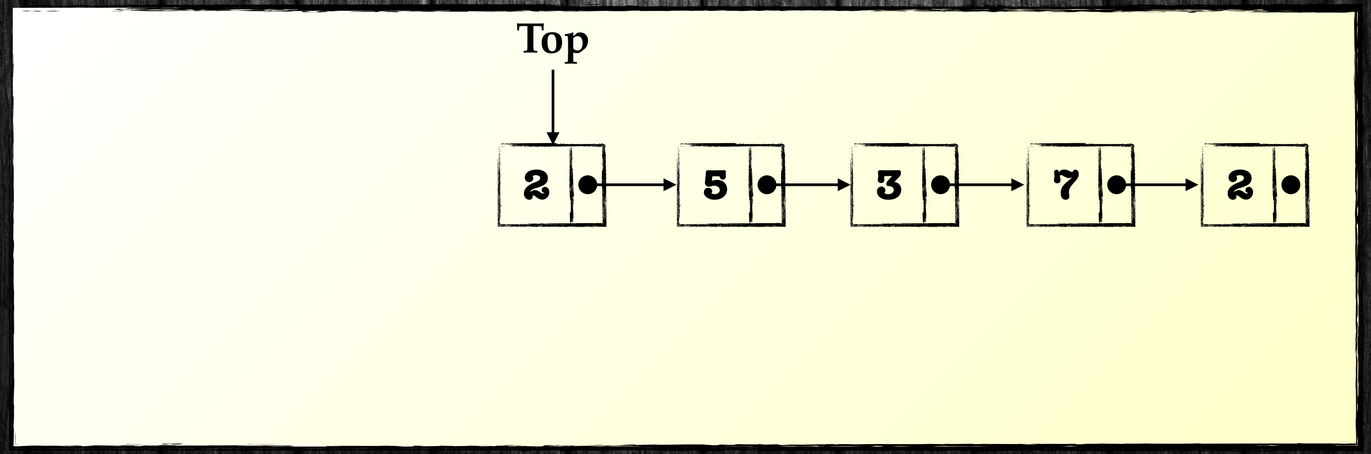


The Treiber Stack Algorithm

Concurrent pop operations

```
pop:
Node t
1  while (true)
2    t = Top
3    if (t = NULL)
4      return *
5      exit
6    if (CAS (Top,t,t.next))
7      return t.val
8      exit
```



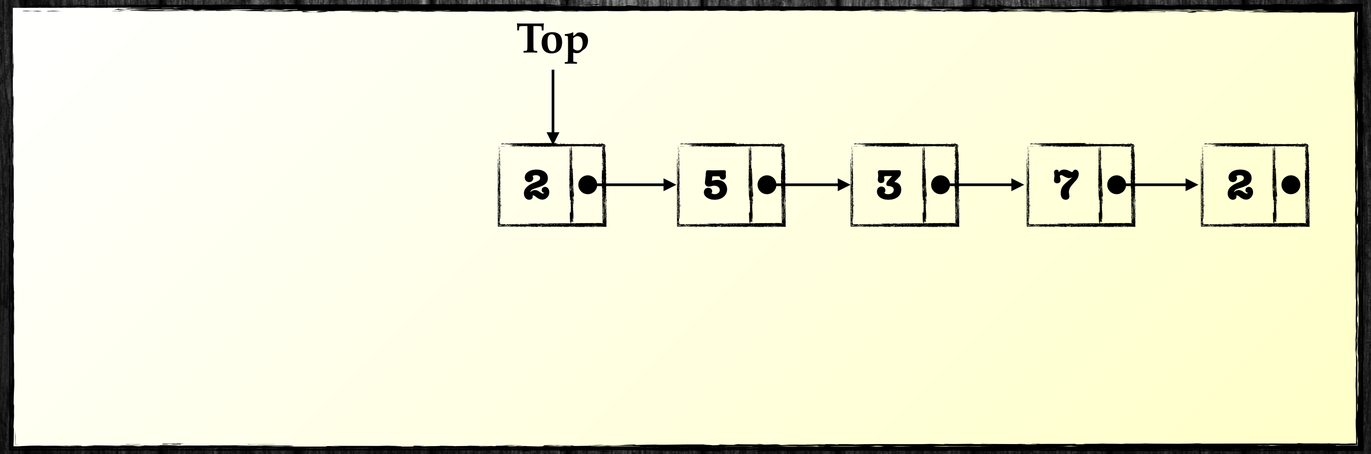


The Treiber Stack Algorithm

Concurrent pop operations

```
pop:
Node t
1  while (true)
2    t = Top
3    if (t = NULL)
4      return *
5      exit
6    if (CAS (Top,t,t.next))
7      return t.val
8      exit
```



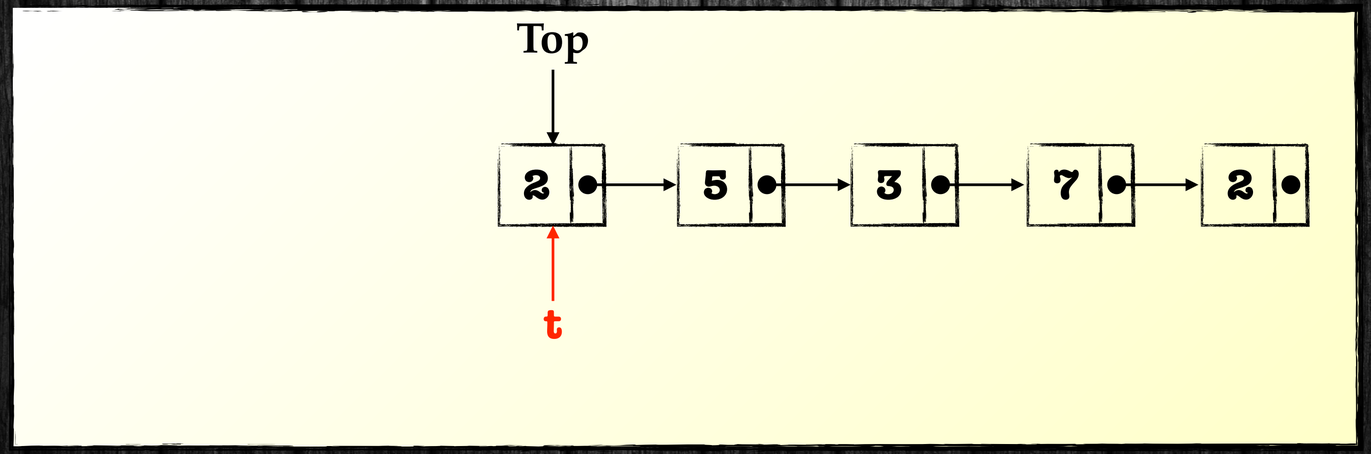


The Treiber Stack Algorithm

Concurrent pop operations

```
pop:
Node t
1  while (true)
2    t = Top
3    if (t = NULL)
4      return *
5      exit
6    if (CAS (Top,t,t.next))
7      return t.val
8      exit
```



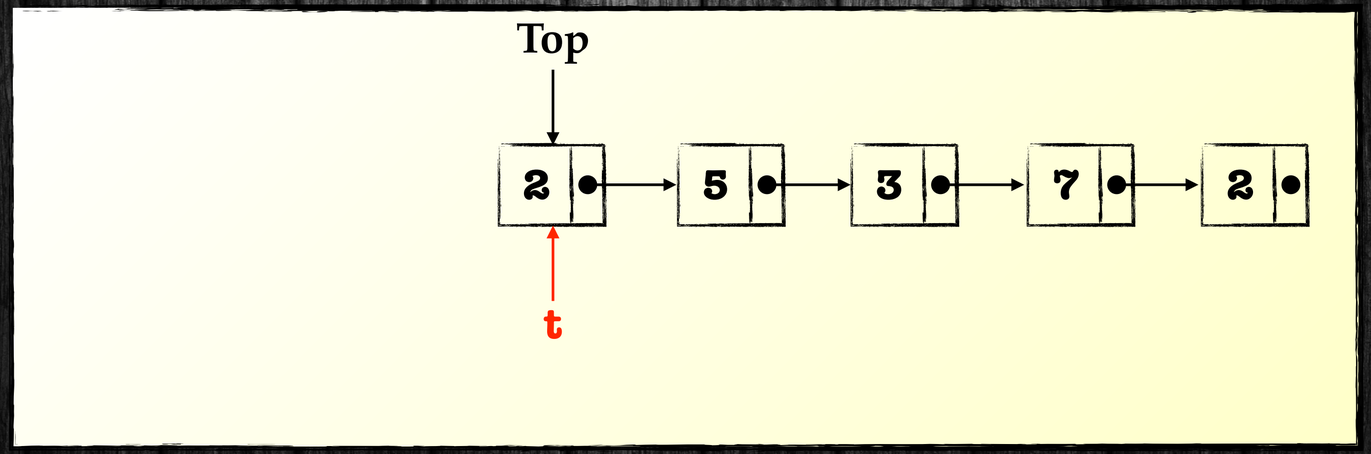


The Treiber Stack Algorithm

Concurrent pop operations

```
pop:
Node t
1  while (true)
2    t = Top
3    if (t = NULL)
4      return *
5      exit
6    if (CAS (Top,t,t.next))
7      return t.val
8      exit
```

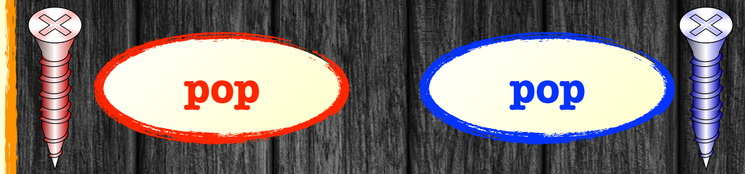


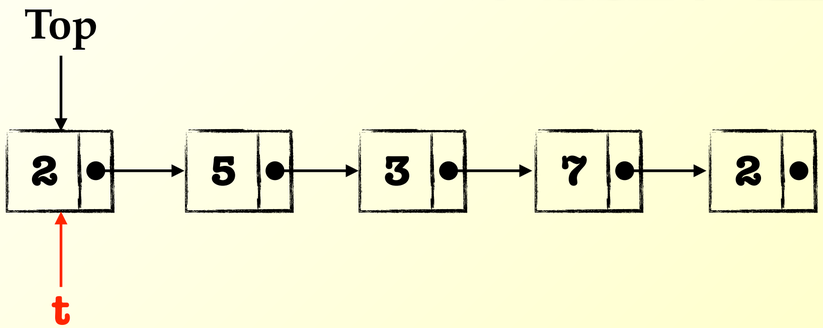


The Treiber Stack Algorithm

Concurrent pop operations

```
pop:
Node t
1  while (true)
2    t = Top
3    if (t = NULL)
4      return *
5      exit
6    if (CAS (Top,t,t.next))
7      return t.val
8      exit
```

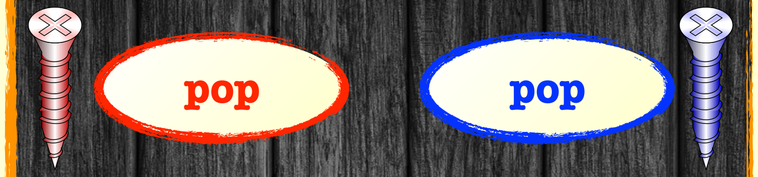




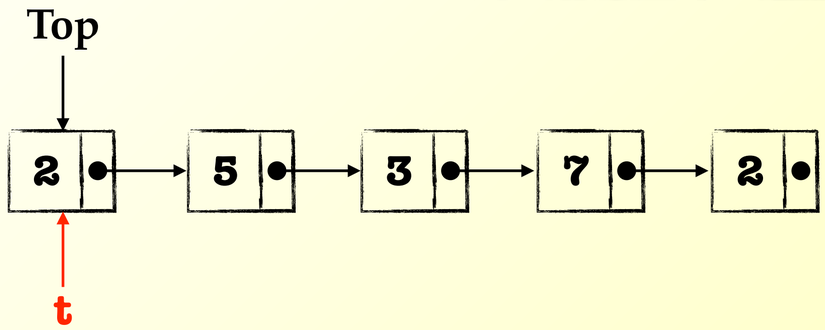
The Treiber Stack Algorithm

Concurrent pop operations

```
pop:
Node t
1  while (true)
2    t = Top
3    if (t = NULL)
4      return *
5      exit
6    if (CAS (Top,t,t.next))
7      return t.val
8      exit
```



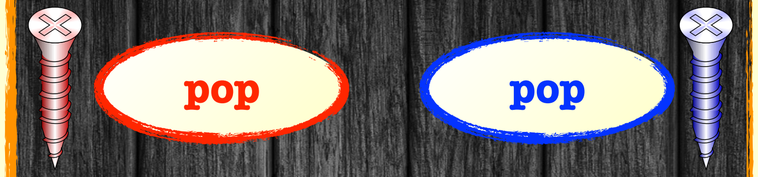

```
pop:
Node t
1  while (true)
2    t = Top
3    if (t = NULL)
4      return *
5      exit
6    if (CAS (Top,t,t.next))
7      return t.val
8      exit
```



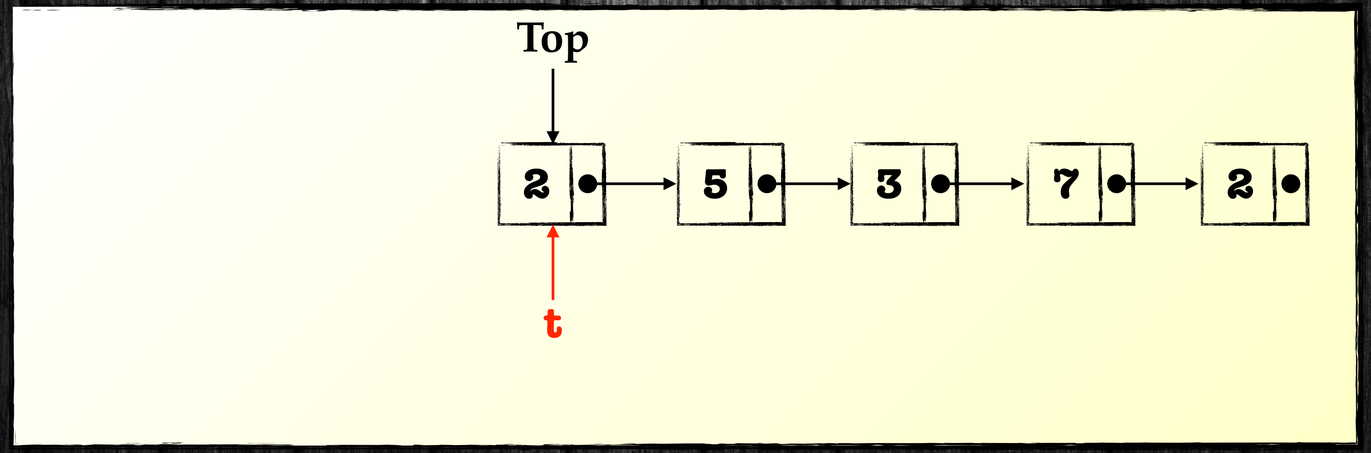
The Treiber Stack Algorithm

Concurrent pop operations

```
pop:
Node t
1  while (true)
2    t = Top
3    if (t = NULL)
4      return *
5      exit
6    if (CAS (Top,t,t.next))
7      return t.val
8      exit
```



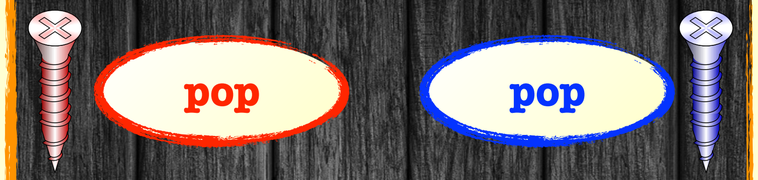

```
pop:
Node t
1  while (true)
2    t = Top
3    if (t = NULL)
4      return *
5      exit
6    if (CAS (Top,t,t.next))
7      return t.val
8      exit
```



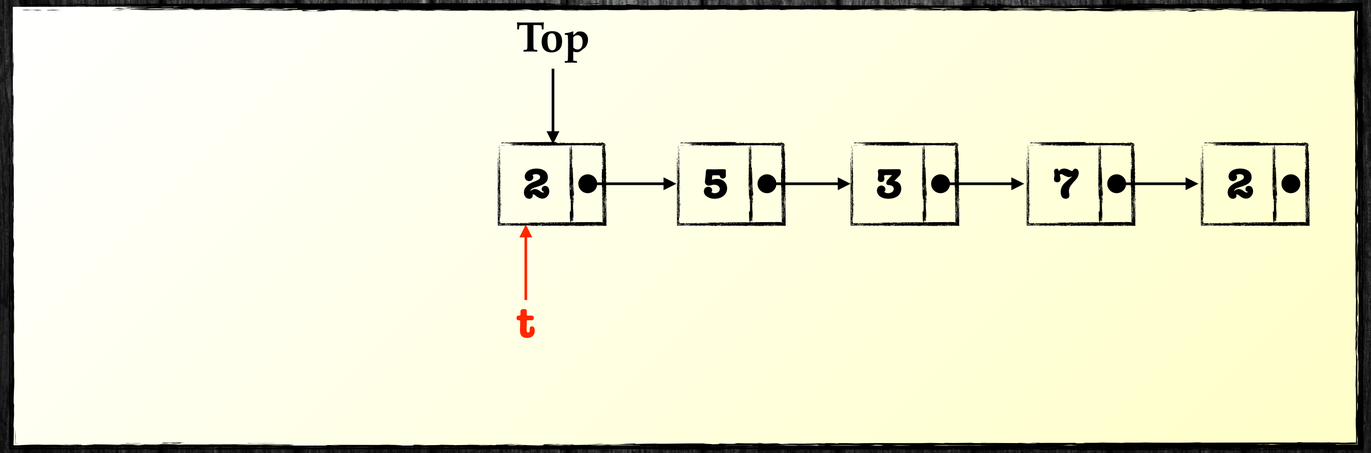
The Treiber Stack Algorithm

Concurrent pop operations

```
pop:
Node t
1  while (true)
2    t = Top
3    if (t = NULL)
4      return *
5      exit
6    if (CAS (Top,t,t.next))
7      return t.val
8      exit
```




```
pop:
Node t
1  while (true)
2    t = Top
3    if (t = NULL)
4      return *
5      exit
6    if (CAS (Top,t,t.next))
7      return t.val
8      exit
```



The Treiber Stack Algorithm

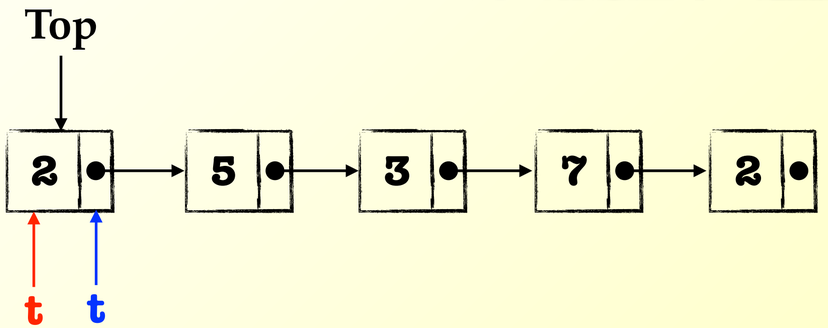
Concurrent pop operations

```
pop:
Node t
1  while (true)
2    t = Top
3    if (t = NULL)
4      return *
5      exit
6    if (CAS (Top,t,t.next))
7      return t.val
8      exit
```

pop

pop

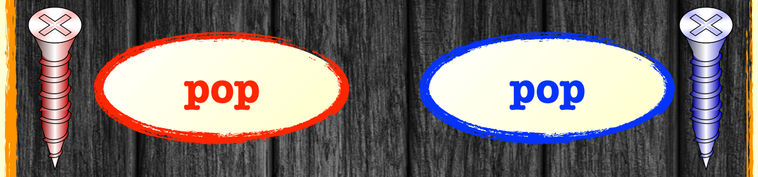
```
pop:
Node t
1  while (true)
2    t = Top
3    if (t = NULL)
4      return *
5      exit
6    if (CAS (Top,t,t.next))
7      return t.val
8      exit
```



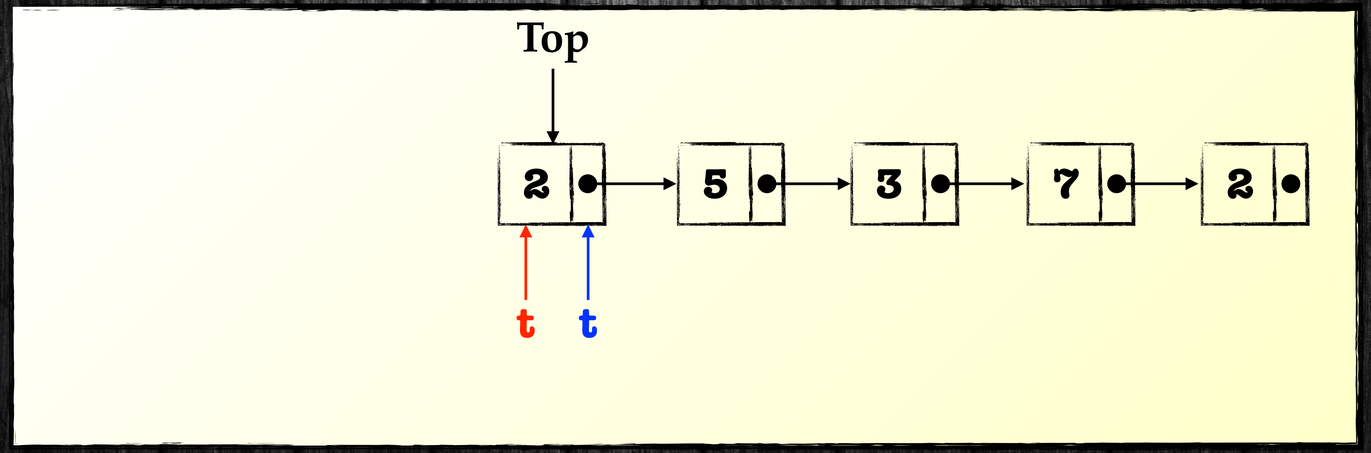
The Treiber Stack Algorithm

Concurrent pop operations

```
pop:
Node t
1  while (true)
2    t = Top
3    if (t = NULL)
4      return *
5      exit
6    if (CAS (Top,t,t.next))
7      return t.val
8      exit
```



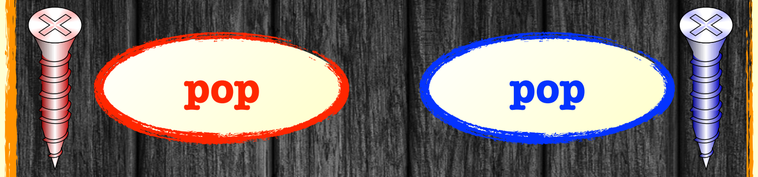

```
pop:
Node t
1  while (true)
2    t = Top
3    if (t = NULL)
4      return *
5      exit
6    if (CAS (Top,t,t.next))
7      return t.val
8      exit
```



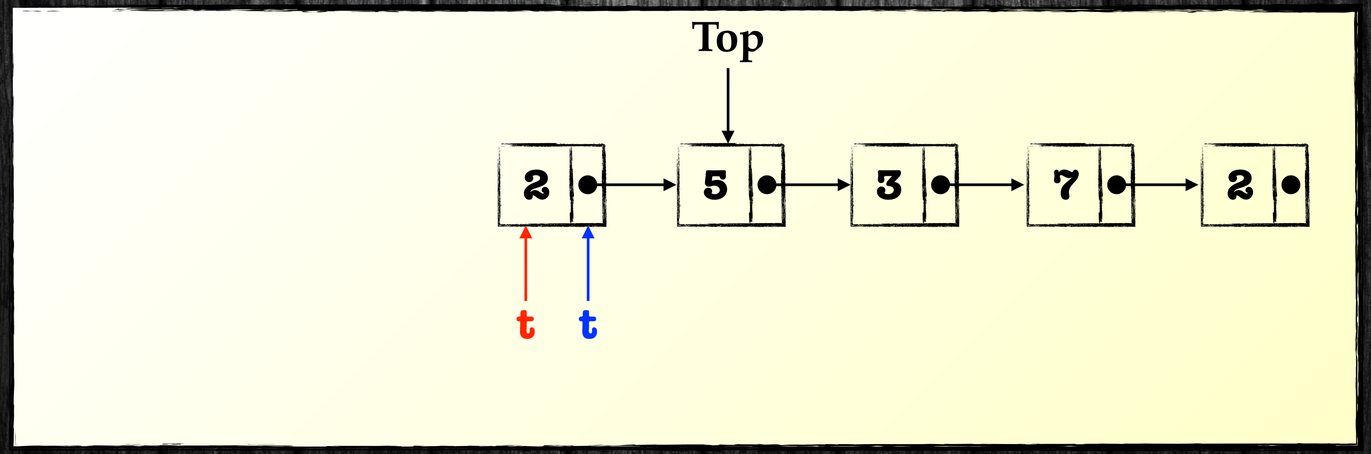
The Treiber Stack Algorithm

Concurrent pop operations

```
pop:
Node t
1  while (true)
2    t = Top
3    if (t = NULL)
4      return *
5      exit
6    if (CAS (Top,t,t.next))
7      return t.val
8      exit
```



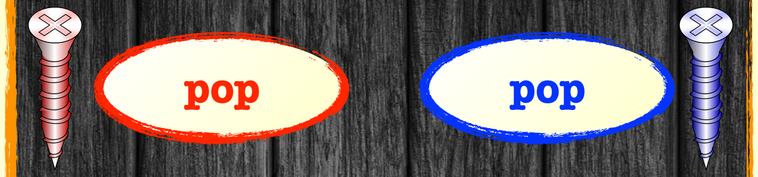

```
pop:
Node t
1  while (true)
2    t = Top
3    if (t = NULL)
4      return *
5      exit
6    if (CAS (Top,t,t.next))
7      return t.val
8      exit
```



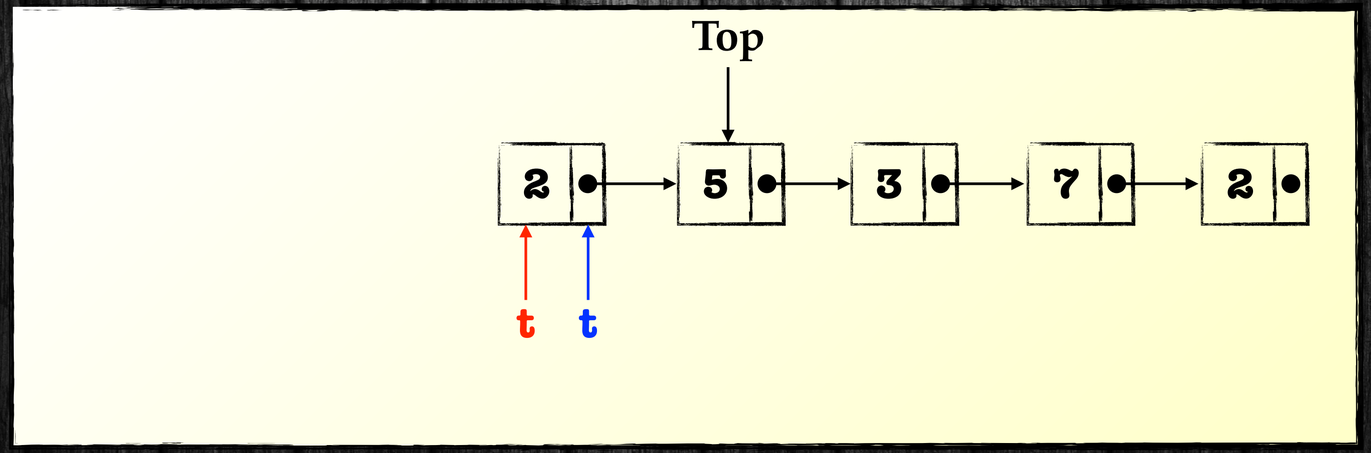
The Treiber Stack Algorithm

Concurrent pop operations

```
pop:
Node t
1  while (true)
2    t = Top
3    if (t = NULL)
4      return *
5      exit
6    if (CAS (Top,t,t.next))
7      return t.val
8      exit
```




```
pop:
Node t
1  while (true)
2    t = Top
3    if (t = NULL)
4      return *
5      exit
6    if (CAS (Top,t,t.next))
7      return t.val
8      exit
```



The Treiber Stack Algorithm

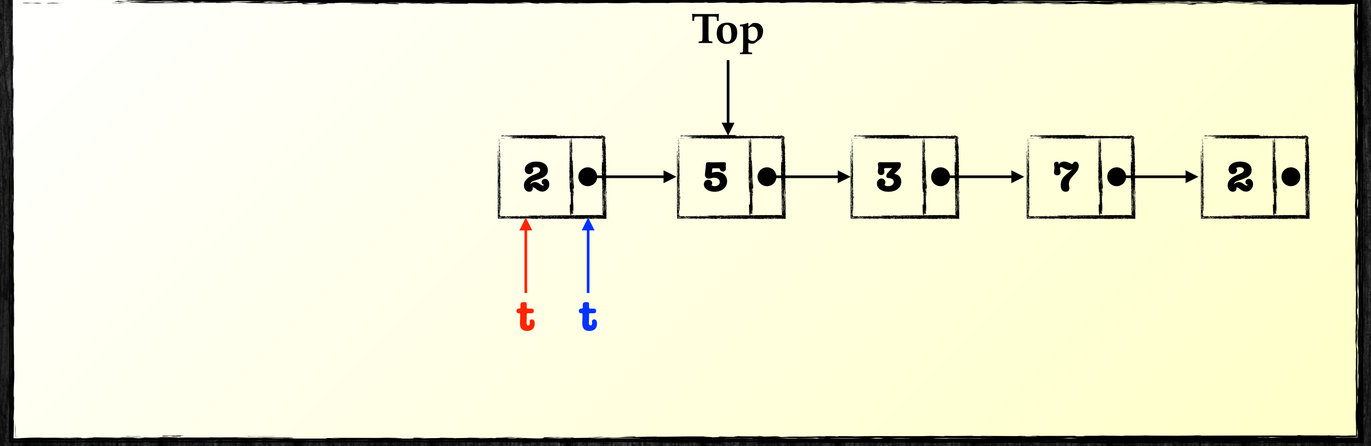
Concurrent pop operations

```
pop:
Node t
1  while (true)
2    t = Top
3    if (t = NULL)
4      return *
5      exit
6    if (CAS (Top,t,t.next))
7      return t.val
8      exit
```

pop

pop

```
pop:
Node t
1  while (true)
2    t = Top
3    if (t = NULL)
4      return *
5      exit
6    if (CAS (Top,t,t.next))
7      return t.val
8      exit
```



The Treiber Stack Algorithm

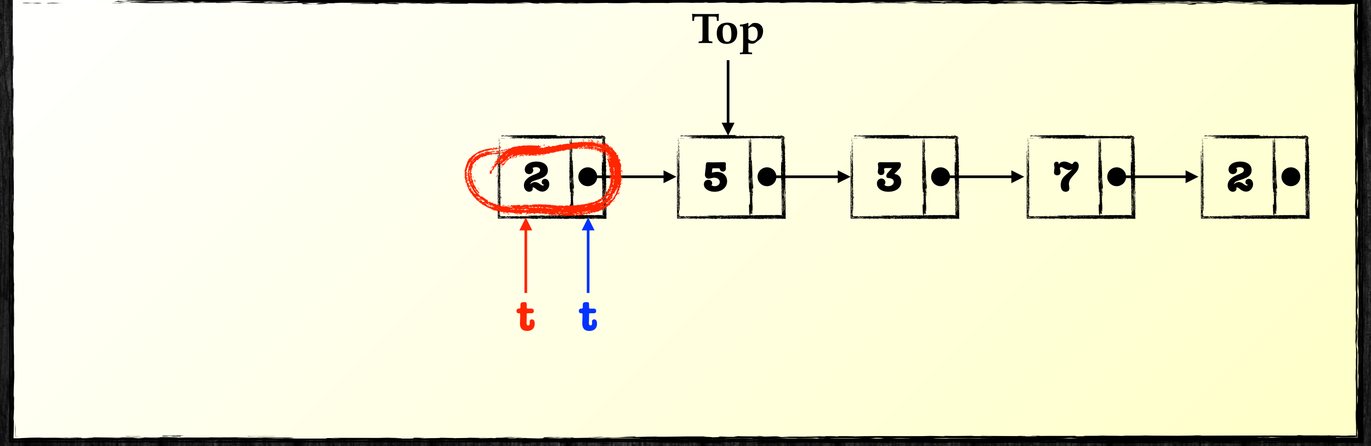
Concurrent pop operations

```
pop:
Node t
1  while (true)
2    t = Top
3    if (t = NULL)
4      return *
5      exit
6    if (CAS (Top,t,t.next))
7      return t.val
8      exit
```

pop

pop

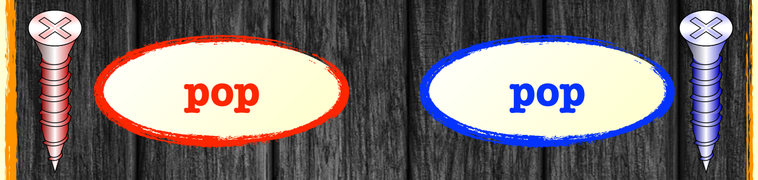
```
pop:
Node t
1  while (true)
2    t = Top
3    if (t = NULL)
4      return *
5      exit
6    if (CAS (Top,t,t.next))
7      return t.val
8      exit
```



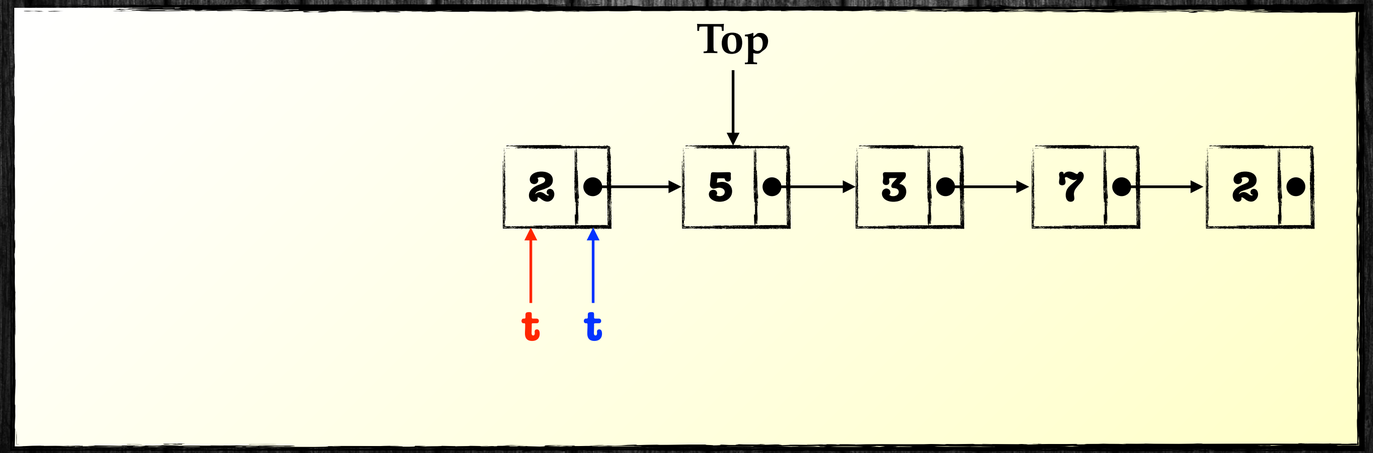
The Treiber Stack Algorithm

Concurrent pop operations

```
pop:
Node t
1  while (true)
2    t = Top
3    if (t = NULL)
4      return *
5      exit
6    if (CAS (Top,t,t.next))
7      return t.val
8      exit
```




```
pop:
Node t
1  while (true)
2    t = Top
3    if (t = NULL)
4      return *
5      exit
6    if (CAS (Top,t,t.next))
7      return t.val
8      exit
```

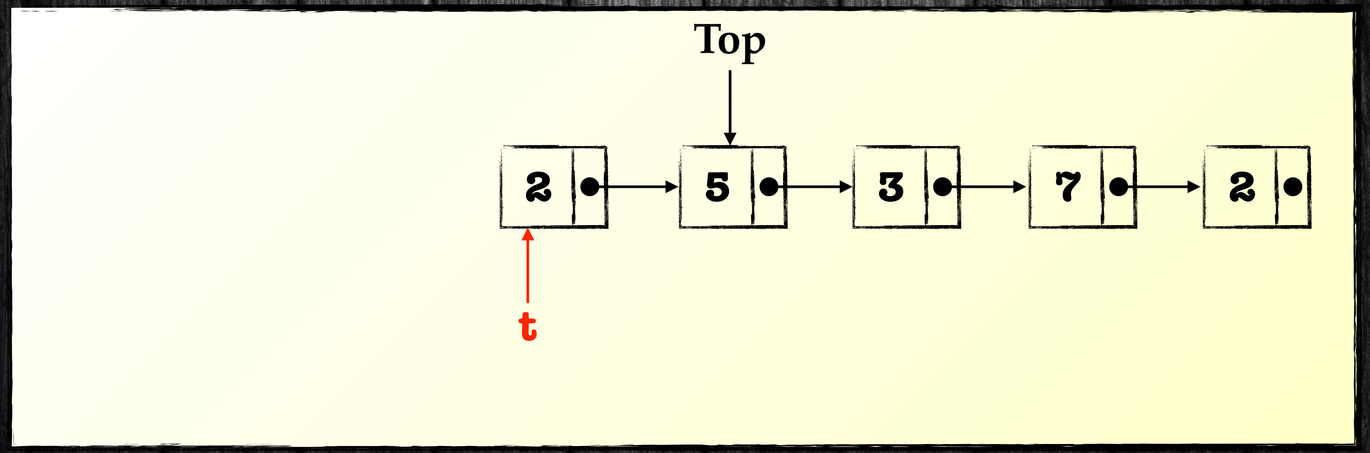


The Treiber Stack Algorithm

Concurrent pop operations

```
pop:
Node t
1  while (true)
2    t = Top
3    if (t = NULL)
4      return *
5      exit
6    if (CAS (Top,t,t.next))
7      return t.val
8      exit
```



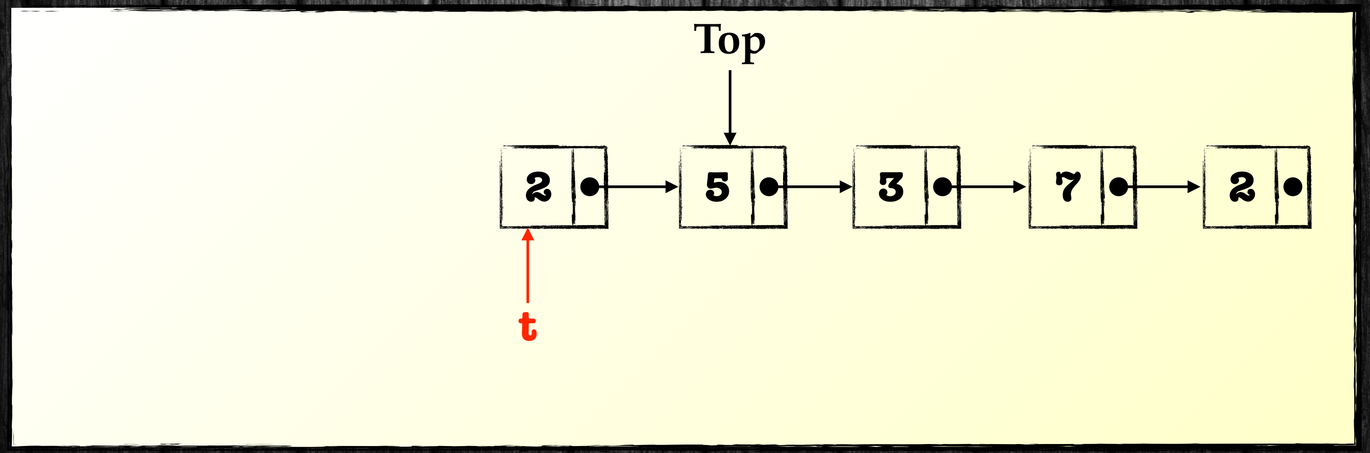


The Treiber Stack Algorithm

Concurrent pop operations

```
pop:
Node t
1  while (true)
2    t = Top
3    if (t = NULL)
4      return *
5      exit
6    if (CAS (Top,t,t.next))
7      return t.val
8      exit
```



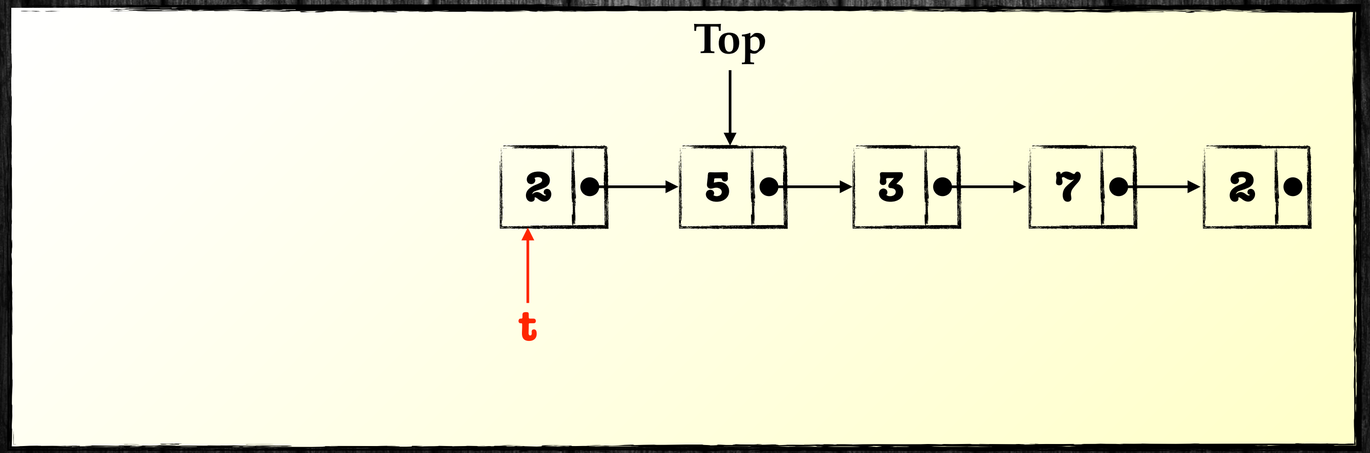


The Treiber Stack Algorithm

Concurrent pop operations

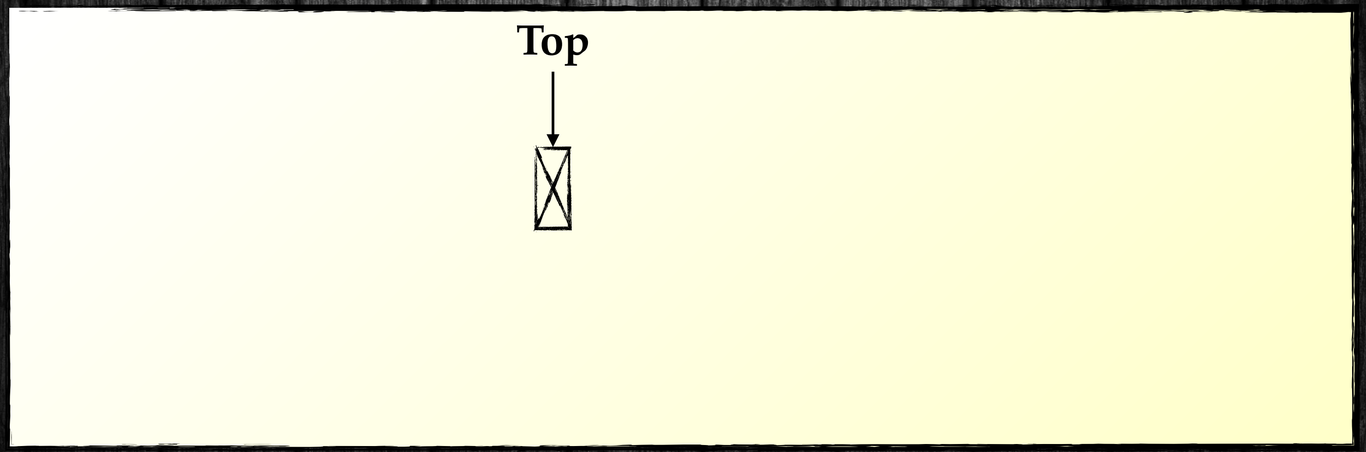
```
pop:
Node t
1  while (true)
2    t = Top
3    if (t = NULL)
4      return *
5      exit
6    if (CAS (Top,t,t.next))
7      return t.val
8      exit
```





The Treiber Stack Algorithm

```
pop:
Node t
1  while (true)
2    t = Top
3    if (t = NULL)
4      return *
5      exit
6    if (CAS (Top,t,t.next))
7      return t.val
8      exit
```

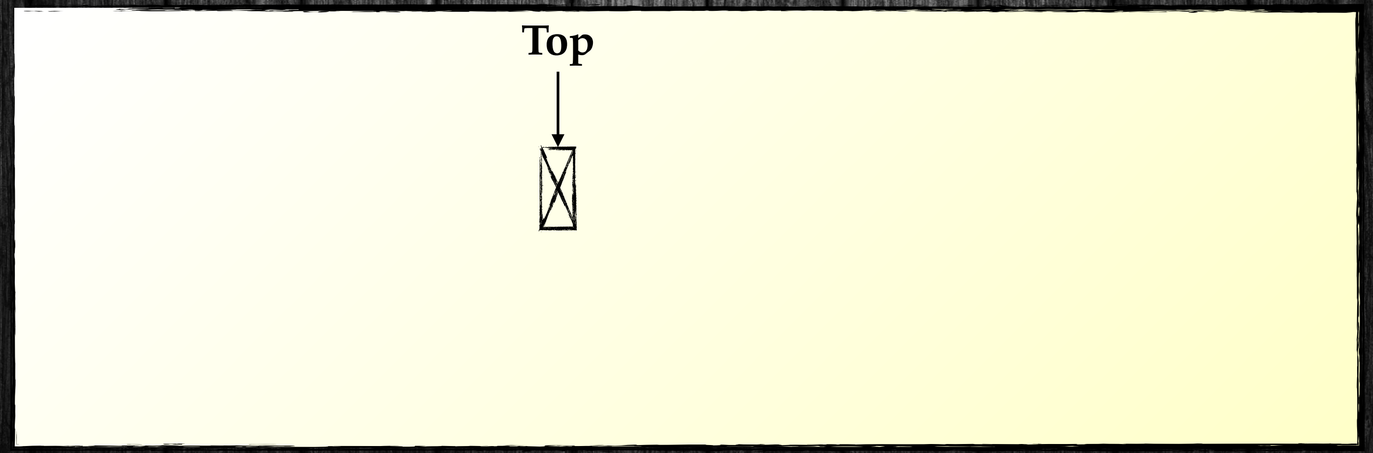


The Treiber Stack Algorithm

```
pop:
Node t
1  while (true)
2    t = Top
3    if (t = NULL)
4      return *
5      exit
6    if (CAS (Top,t,t.next))
7      return t.val
8      exit
```



pop

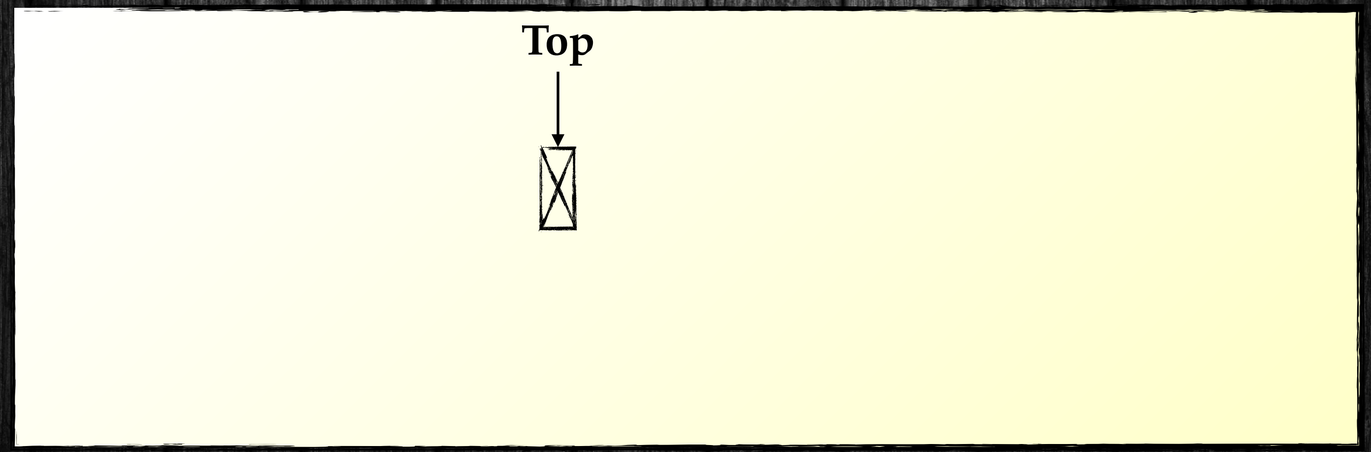


The Treiber Stack Algorithm

```
pop:
Node t
1  while (true)
2    t = Top
3    if (t = NULL)
4      return *
5      exit
6    if (CAS (Top,t,t.next))
7      return t.val
8      exit
```



pop

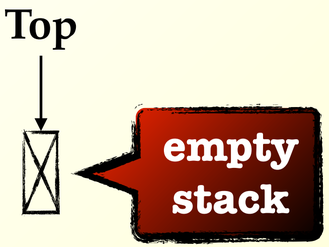


The Treiber Stack Algorithm

```
pop:
Node t
1  while (true)
2    t = Top
3    if (t = NULL)
4      return *
5      exit
6    if (CAS (Top,t,t.next))
7      return t.val
8      exit
```



pop

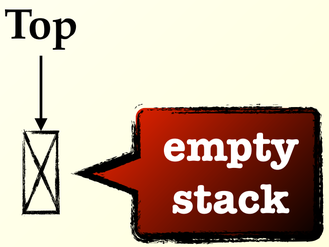


The Treiber Stack Algorithm

```
pop:
Node t
1  while (true)
2    t = Top
3    if (t = NULL)
4      return *
5      exit
6    if (CAS (Top,t,t.next))
7      return t.val
8      exit
```



pop

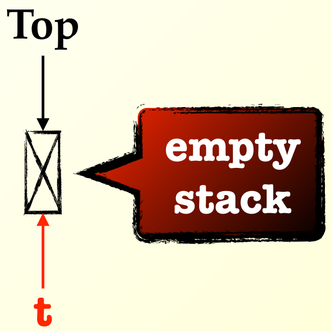


The Treiber Stack Algorithm

```
pop:
Node t
1  while (true)
2    t = Top
3    if (t = NULL)
4      return *
5      exit
6    if (CAS (Top,t,t.next))
7      return t.val
8      exit
```



pop

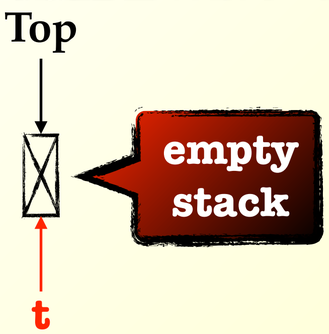


The Treiber Stack Algorithm

```
pop:
Node t
1  while (true)
2    t = Top
3    if (t = NULL)
4      return *
5      exit
6    if (CAS (Top,t,t.next))
7      return t.val
8      exit
```



pop

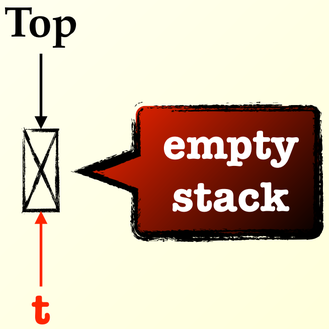


The Treiber Stack Algorithm

```
pop:
Node t
1  while (true)
2    t = Top
3    if (t = NULL)
4      return *
5      exit
6    if (CAS (Top,t,t.next))
7      return t.val
8      exit
```





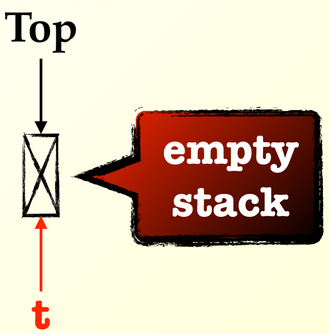


The Treiber Stack Algorithm

```
pop:
Node t
1  while (true)
2    t = Top
3    if (t = NULL)
4      return *
5      exit
6    if (CAS (Top,t,t.next))
7      return t.val
8      exit
```



pop

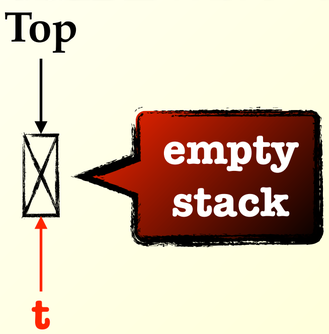


The Treiber Stack Algorithm

```
pop:
Node t
1  while (true)
2    t = Top
3    if (t = NULL)
4      return *
5      exit
6    if (CAS (Top,t,t.next))
7      return t.val
8      exit
```



pop



The Treiber Stack Algorithm

```
push(k):
Node t
1  n = new Node(k,-)
2  while (true)
3    t = Top
4    n.next = t
5    if (CAS (Top,t,n))
6      exit
```

Linearization Policy

```
pop:
Node t
1  while (true)
2    t = Top
3    if (t = NULL)
4      return *
5      exit
6    if (CAS (Top,t,t.next))
7      return t.val
8      exit
```

The Treiber Stack Algorithm

```
push(k):
Node t
1  n = new Node(k,-)
2  while (true)
3    t = Top
4    n.next = t
5    if (CAS (Top,t,n))
6      exit
```



Linearization Policy

```
pop:
Node t
1  while (true)
2    t = Top
3    if (t = NULL)
4      return *
5      exit
6    if (CAS (Top,t,t.next))
7      return t.val
8      exit
```

The Treiber Stack Algorithm

```
push(k):
Node t
1  n = new Node(k,-)
2  while (true)
3    t = Top
4    n.next = t
5    if (CAS (Top,t,n))
6      exit
```



Linearization Policy

```
pop:
Node t
1  while (true)
2    t = Top
3    if (t = NULL)
4      return *
5      exit
6    if (CAS (Top,t,t.next))
7      return t.val
8      exit
```

The Treiber Stack Algorithm

```
push(k):
Node t
1  n = new Node(k,-)
2  while (true)
3    t = Top
4    n.next = t
5    if (CAS (Top,t,n))
6      exit
```

when successful

Linearization Policy

```
pop:
Node t
1  while (true)
2    t = Top
3    if (t = NULL)
4      return *
5      exit
6    if (CAS (Top,t,t.next))
7      return t.val
8      exit
```

The Treiber Stack Algorithm

```
push(k):
Node t
1  n = new Node(k,-)
2  while (true)
3    t = Top
4    n.next = t
5    if (CAS (Top,t,n))
6      exit
```

when successful

Linearization Policy

```
pop:
Node t
1  while (true)
2    t = Top
3    if (t = NULL)
4      return *
5      exit
6    if (CAS (Top,t,t.next))
7      return t.val
8      exit
```

when successful

The Treiber Stack Algorithm

```
push(k):
Node t
1  n = new Node(k,-)
2  while (true)
3    t = Top
4    n.next = t
5    if (CAS (Top,t,n))
6      exit
```

when successful

Linearization Policy

```
pop:
Node t
1  while (true)
2    t = Top
3    if (t = NULL)
4      return *
5      exit
6    if (CAS (Top,t,t.next))
7      return t.val
8      exit
```

when successful

The Treiber Stack Algorithm

```
push(k):
Node t
1  n = new Node(k,-)
2  while (true)
3    t = Top
4    n.next = t
5    if (CAS (Top,t,n))
6      exit
```

when successful

Linearization Policy

```
pop:
Node t
1  while (true)
2    t = Top
3    if (t = NULL)
4      return *
5      exit
6    if (CAS (Top,t,t.next))
7      return t.val
8      exit
```

when empty

when successful

The Treiber Stack Algorithm

```
push(k):
Node t
1  n = new Node(k,-)
2  while (true)
3    t = Top
4    n.next = t
5    if (CAS (Top,t,n))
6      exit
```

when successful

Linearization Policy

```
pop:
Node t
1  while (true)
2    t = Top
3    if (t = NULL)
4      return *
5      exit
6    if (CAS (Top,t,t.next))
7      return t.val
8      exit
```

when empty

when successful

The Treiber Stack Algorithm

```
push(k):
Node t
1  n = new Node(k,-)
2  while (true)
3    t = Top
4    n.next = t
5    if (CAS (Top,t,n))
6      exit
```

when successful

Linearization Policy

```
pop:
Node t
1  while (true)
2    t = Top
3    if (t = NULL)
4      return *
5      exit
6    if (CAS (Top,t,t.next))
7      return t.val
8      exit
```

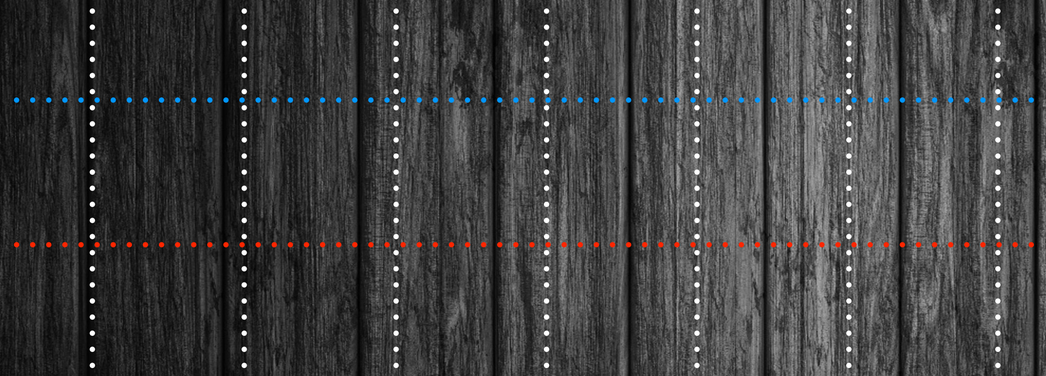
when empty

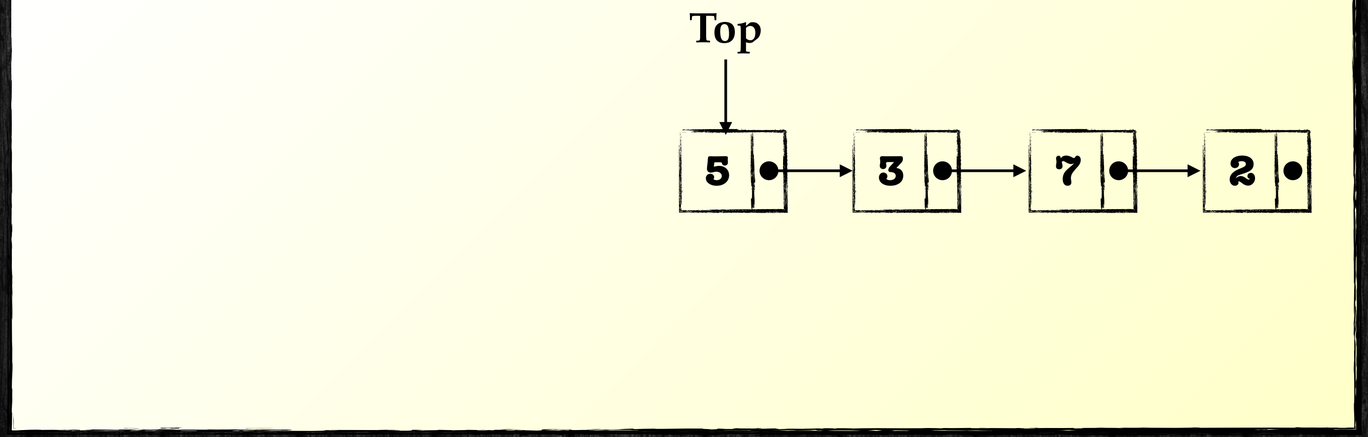
?

when successful

The Treiber Stack Algorithm

Linearization Policy

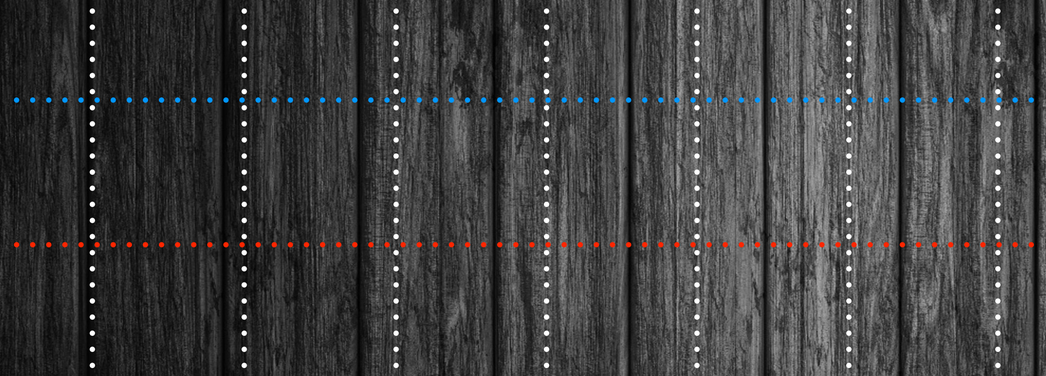


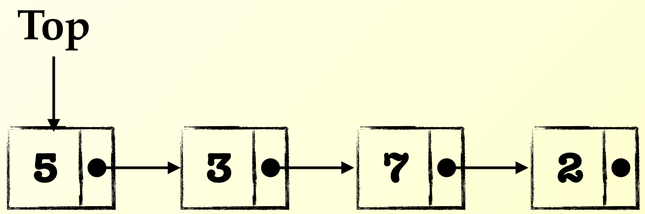


The Treiber Stack Algorithm

Linearization Policy







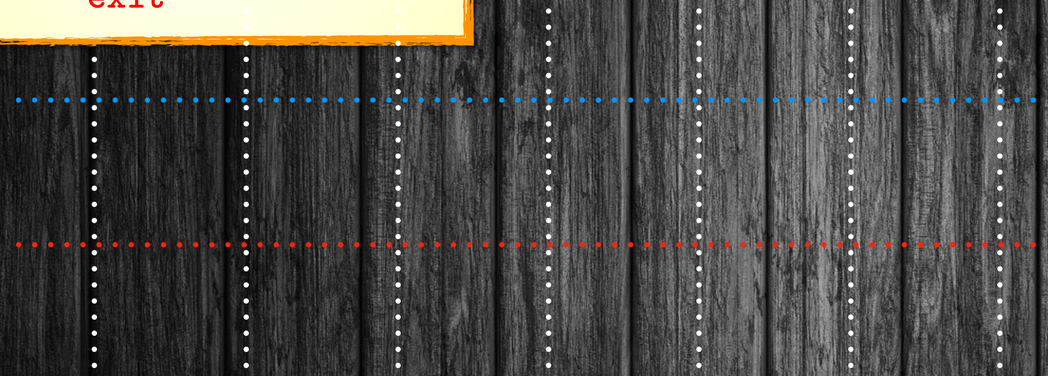
The Treiber Stack Algorithm

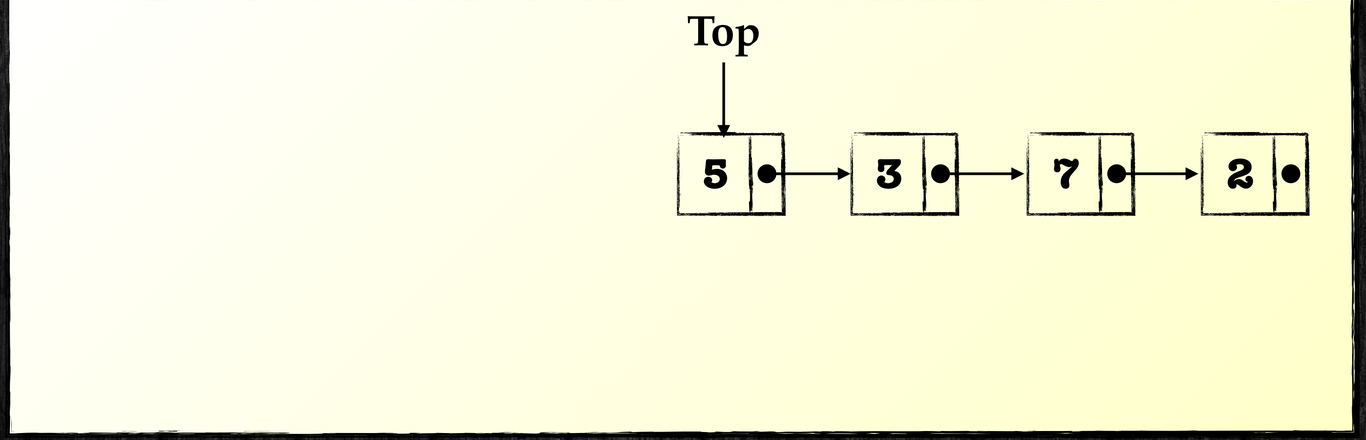
Linearization Policy

```
push(k):
Node t
1  n = new Node(k,-)
2  while (true)
3    t = Top
4    n.next = t
5    if (CAS (Top,t,n))
6      exit
```



push(4)





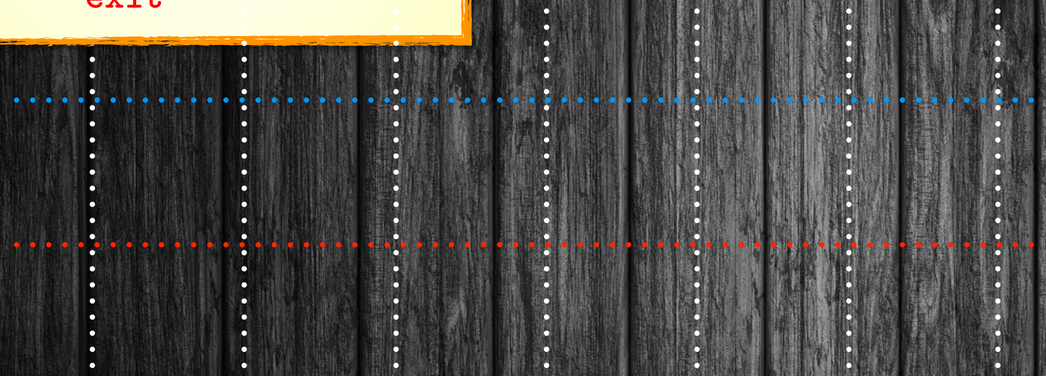
The Treiber Stack Algorithm

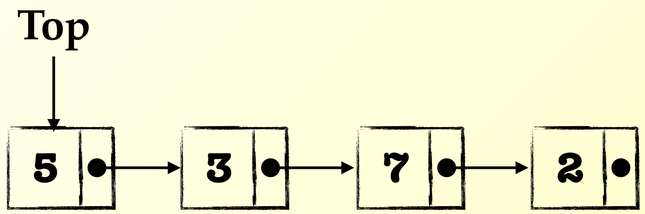
Linearization Policy

```
push(k):
Node t
1  n = new Node(k,-)
2  while (true)
3    t = Top
4    n.next = t
5    if (CAS (Top,t,n))
6      exit
```



push(4)





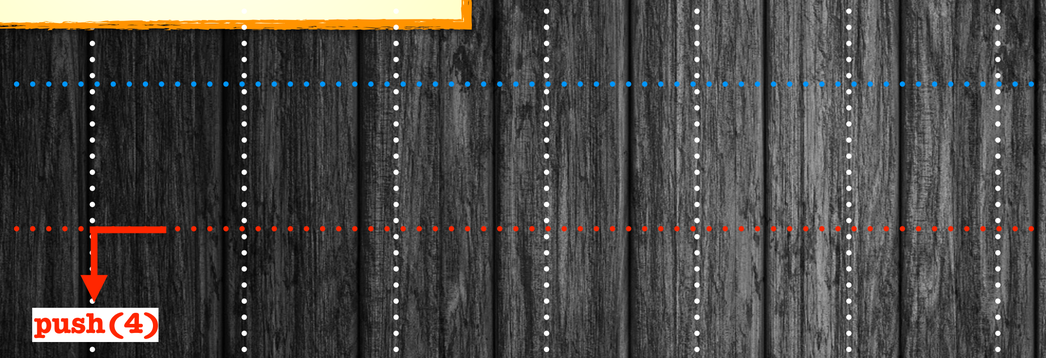
The Treiber Stack Algorithm

Linearization Policy

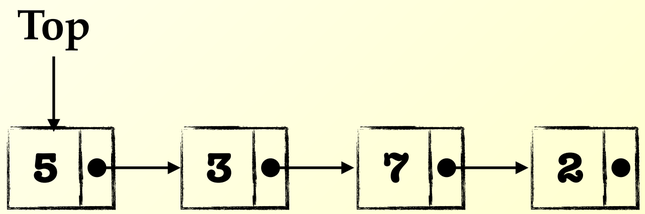
```
push(k):
Node t
1  n = new Node(k,-)
2  while (true)
3    t = Top
4    n.next = t
5    if (CAS (Top,t,n))
6      exit
```



push(4)



push(4)



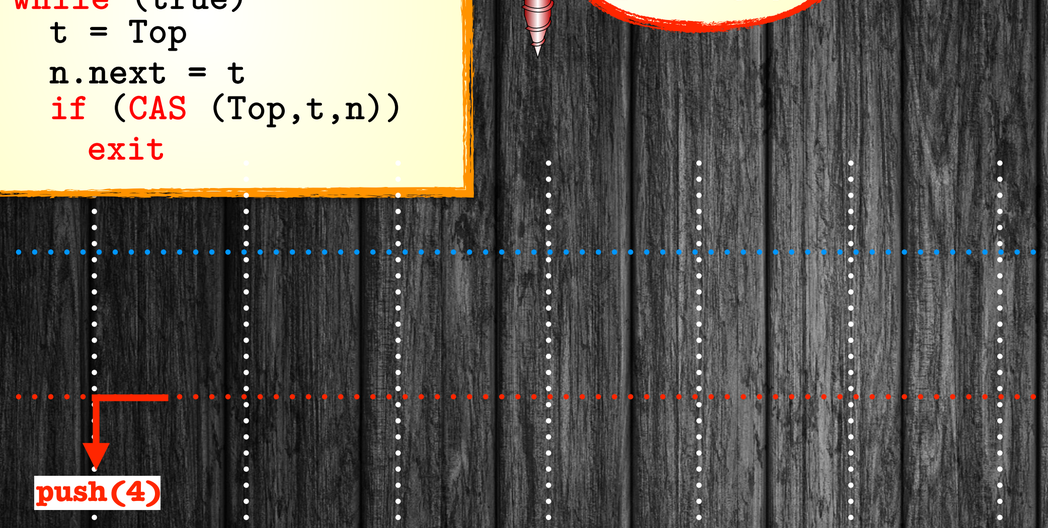
The Treiber Stack Algorithm

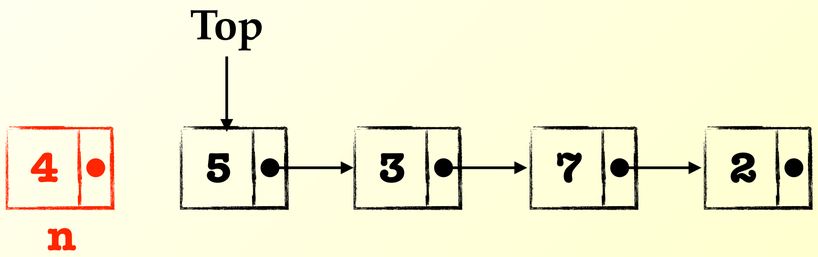
Linearization Policy

```
push(k):
Node t
1  n = new Node(k,-)
2  while (true)
3    t = Top
4    n.next = t
5    if (CAS (Top,t,n))
6      exit
```



push(4)





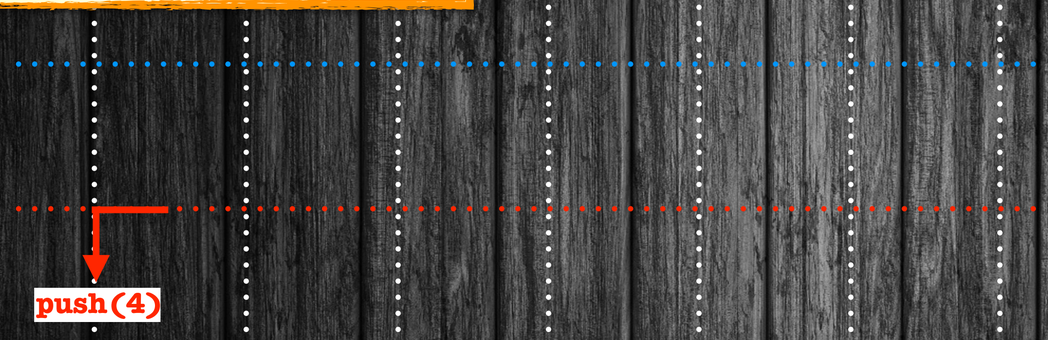
The Treiber Stack Algorithm

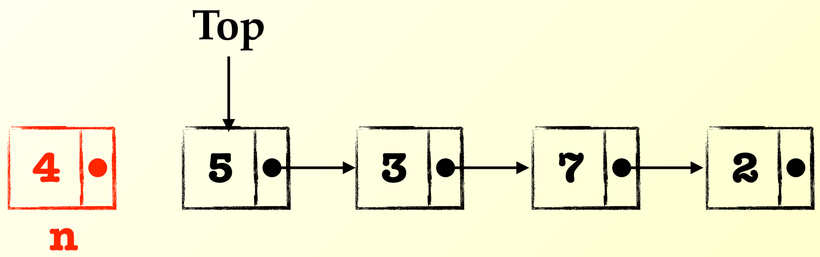
Linearization Policy

```
push(k):
Node t
1  n = new Node(k,-)
2  while (true)
3    t = Top
4    n.next = t
5    if (CAS (Top,t,n))
6      exit
```



push(4)





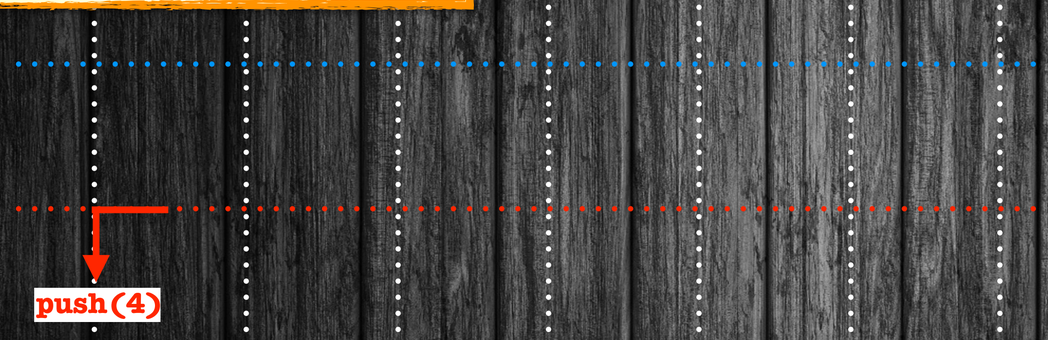
The Treiber Stack Algorithm

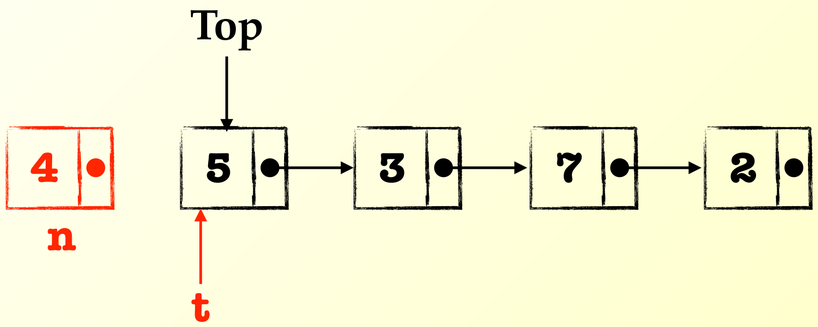
Linearization Policy

```
push(k):
Node t
1  n = new Node(k,-)
2  while (true)
3    t = Top
4    n.next = t
5    if (CAS (Top,t,n))
6      exit
```



push(4)





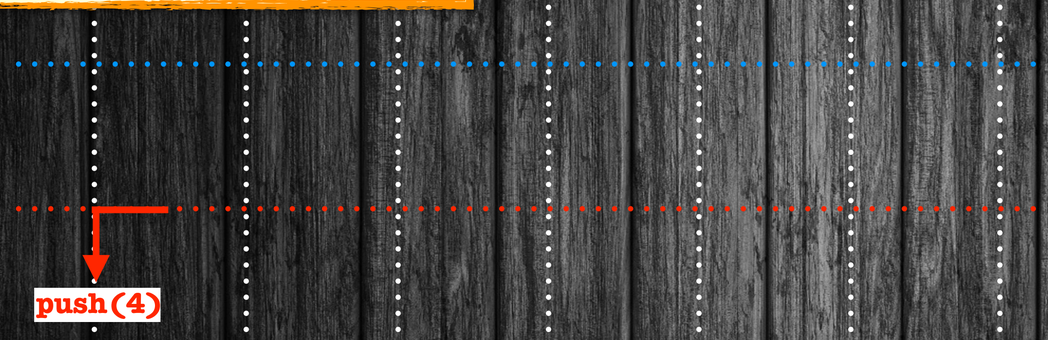
The Treiber Stack Algorithm

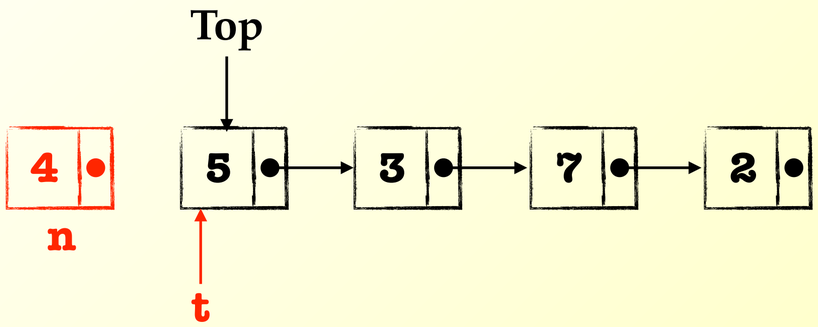
Linearization Policy

```
push(k):
Node t
1  n = new Node(k,-)
2  while (true)
3    t = Top
4    n.next = t
5    if (CAS (Top,t,n))
6      exit
```



push(4)





The Treiber Stack Algorithm

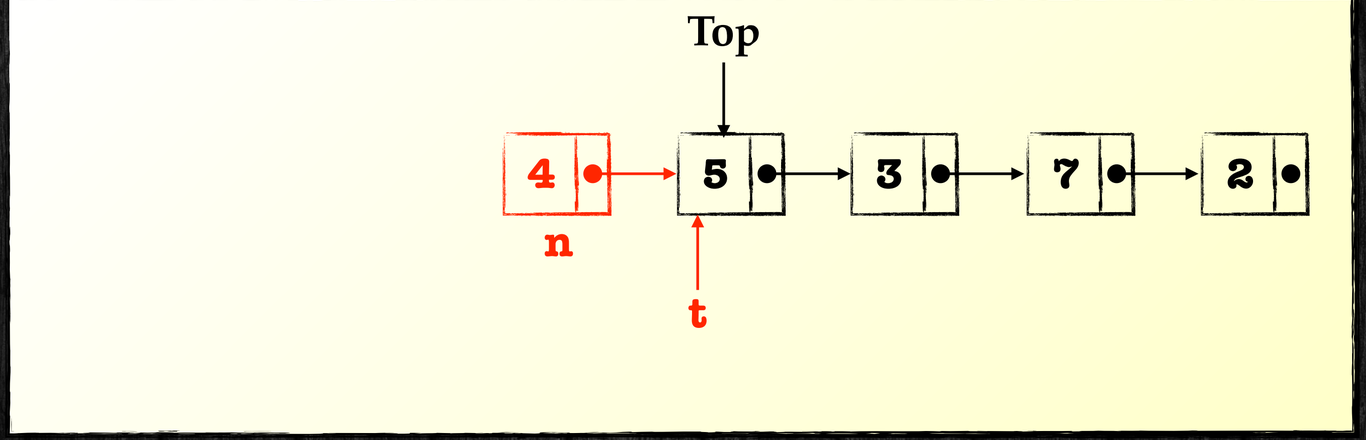
Linearization Policy

```
push(k):
Node t
1  n = new Node(k,-)
2  while (true)
3    t = Top
4    n.next = t
5    if (CAS (Top,t,n))
6      exit
```



push(4)

push(4)



The Treiber Stack Algorithm

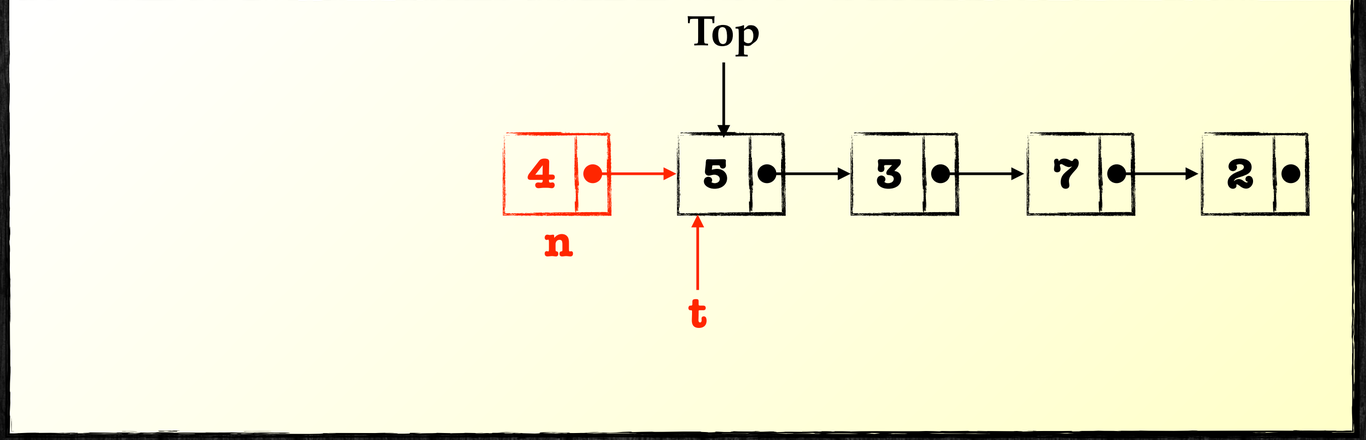
Linearization Policy

```
push(k):
Node t
1  n = new Node(k,-)
2  while (true)
3    t = Top
4    n.next = t
5    if (CAS (Top,t,n))
6      exit
```



push(4)

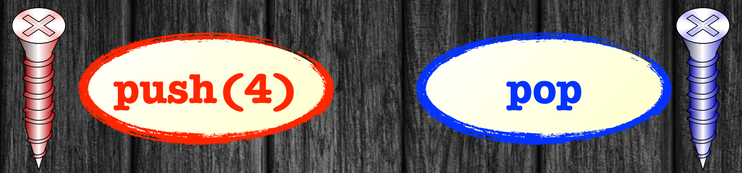
push(4)

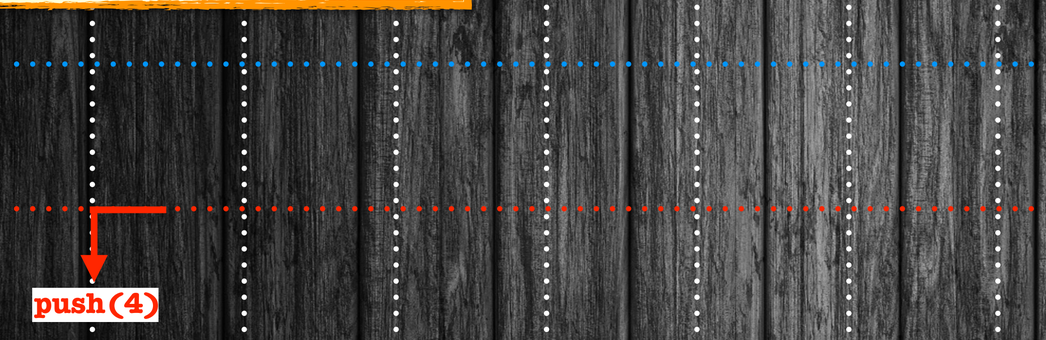


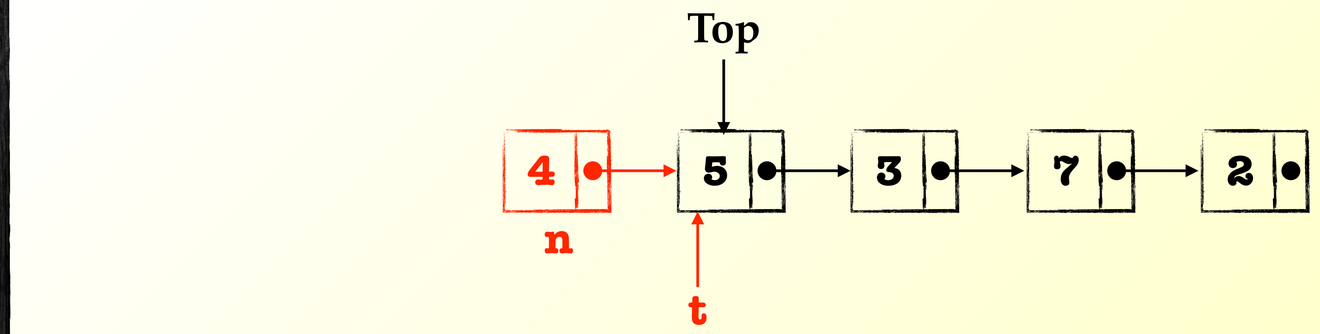
The Treiber Stack Algorithm

Linearization Policy

```
push(k):
Node t
1  n = new Node(k,-)
2  while (true)
3    t = Top
4    n.next = t
5    if (CAS (Top,t,n))
6      exit
```



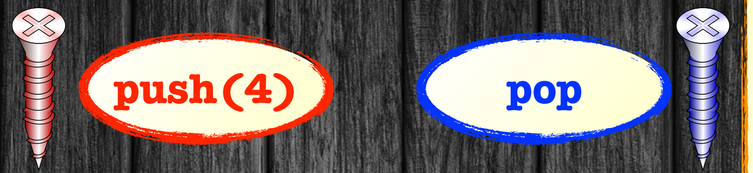




The Treiber Stack Algorithm

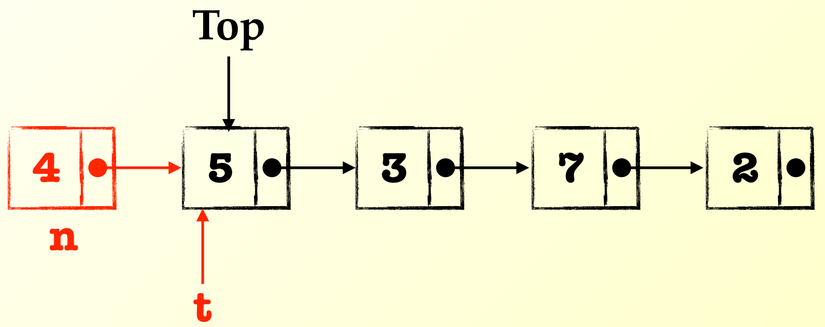
Linearization Policy

```
push(k):
Node t
1  n = new Node(k,-)
2  while (true)
3    t = Top
4    n.next = t
5    if (CAS (Top,t,n))
6      exit
```




```
pop:
Node t
1  while (true)
2    t = Top
3    if (t = NULL)
4      return *
5      exit
6    if (CAS (Top,t,t.next))
7      return t.val
8      exit
```



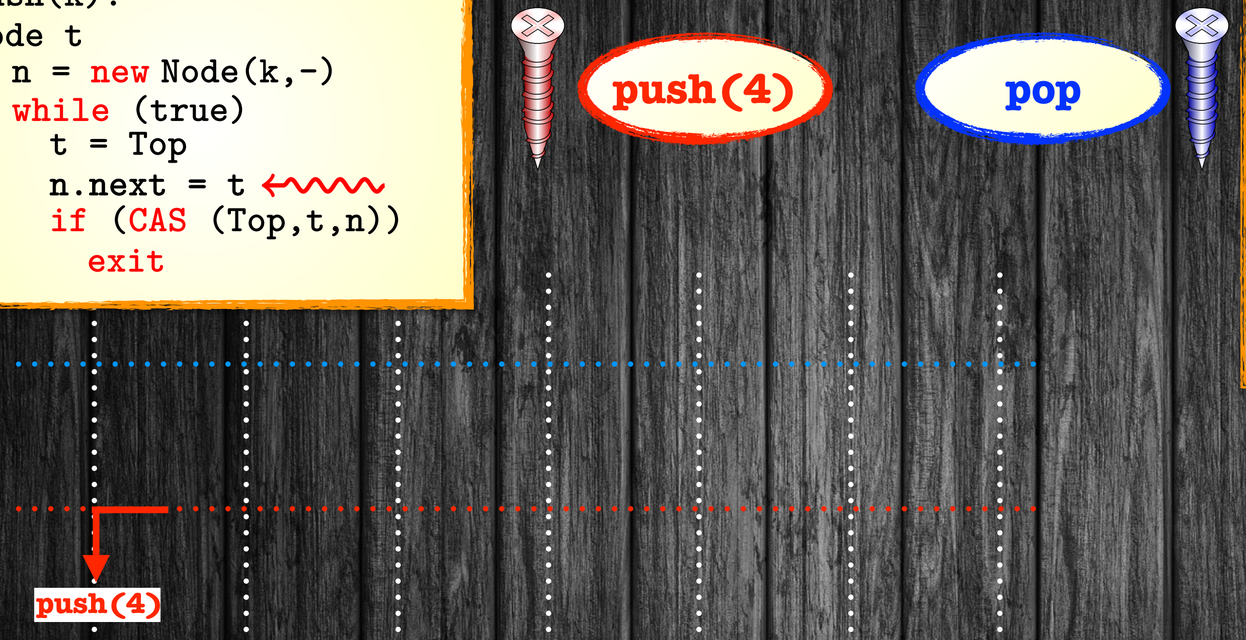


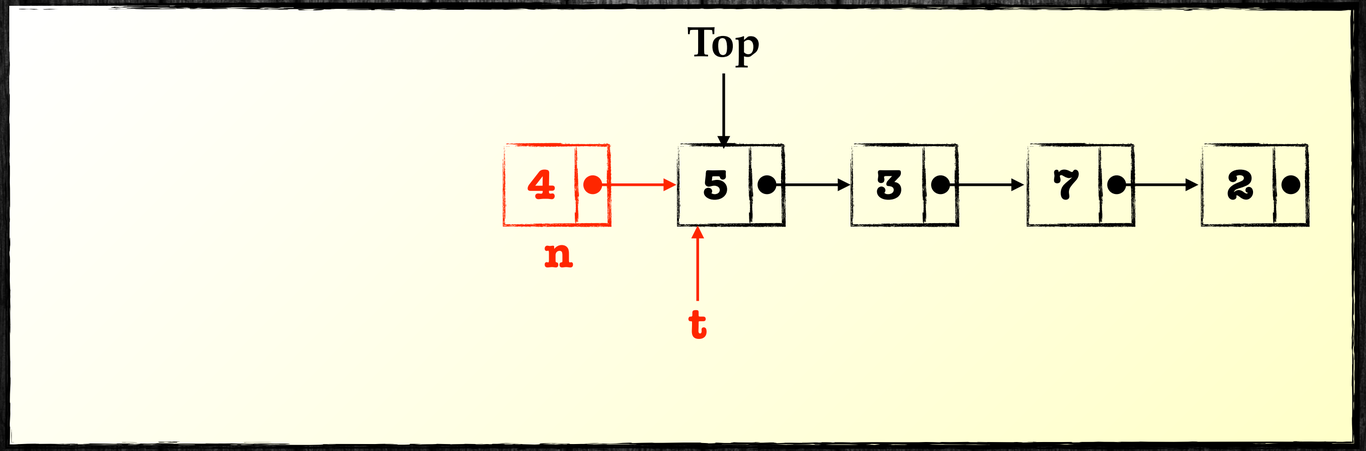
The Treiber Stack Algorithm

Linearization Policy

```
push(k):
Node t
1  n = new Node(k,-)
2  while (true)
3    t = Top
4    n.next = t
5    if (CAS (Top,t,n))
6      exit
```

```
pop:
Node t
1  while (true)
2    t = Top
3    if (t = NULL)
4      return *
5      exit
6    if (CAS (Top,t,t.next))
7      return t.val
8      exit
```



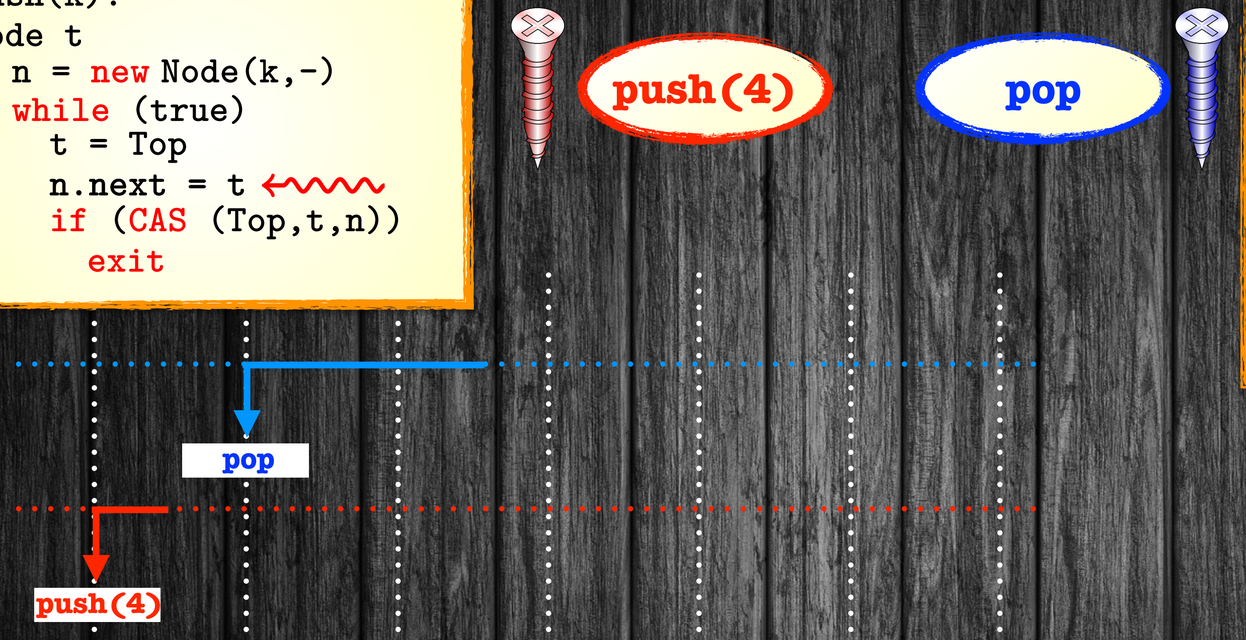


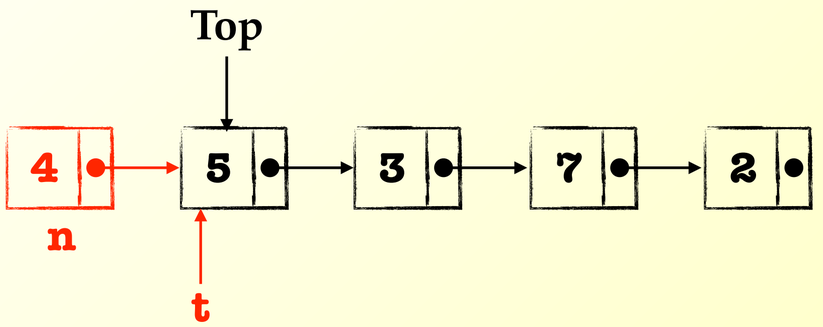
The Treiber Stack Algorithm

Linearization Policy

```
push(k):
Node t
1  n = new Node(k,-)
2  while (true)
3    t = Top
4    n.next = t
5    if (CAS (Top,t,n))
6      exit
```

```
pop:
Node t
1  while (true)
2    t = Top
3    if (t = NULL)
4      return *
5      exit
6    if (CAS (Top,t,t.next))
7      return t.val
8      exit
```



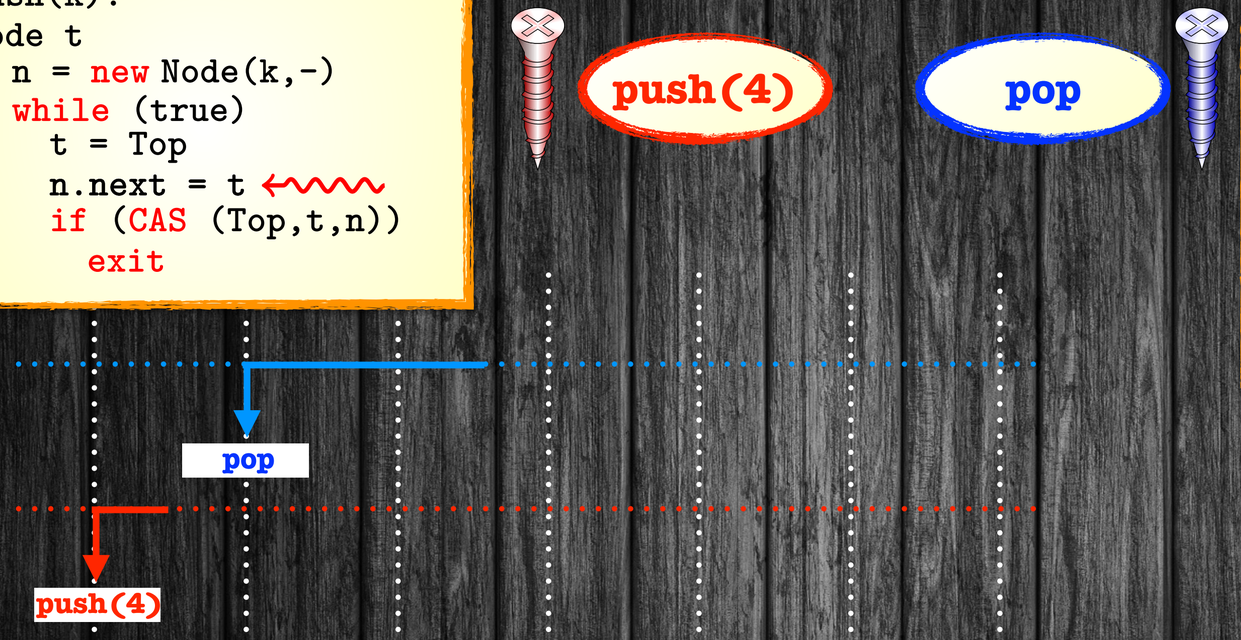


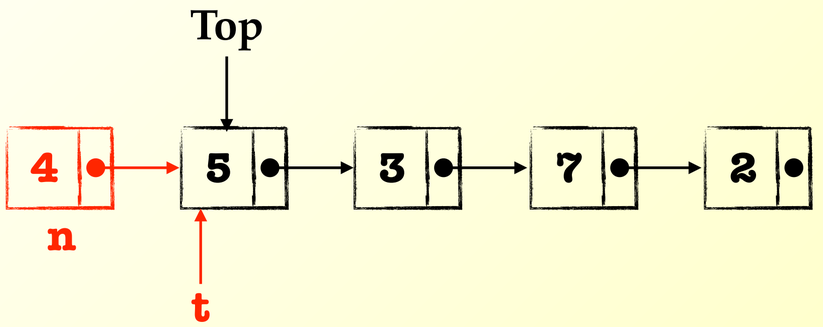
The Treiber Stack Algorithm

Linearization Policy

```
push(k):
Node t
1  n = new Node(k,-)
2  while (true)
3    t = Top
4    n.next = t
5    if (CAS (Top,t,n))
6      exit
```

```
pop:
Node t
1  while (true)
2    t = Top
3    if (t = NULL)
4      return *
5      exit
6    if (CAS (Top,t,t.next))
7      return t.val
8      exit
```



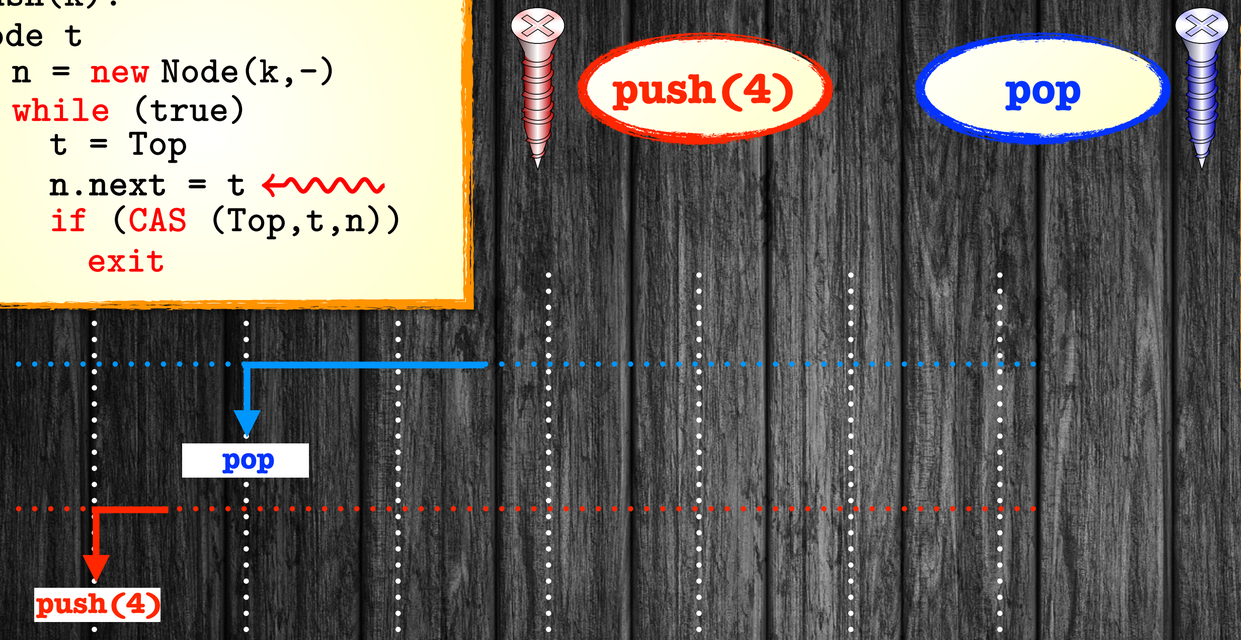


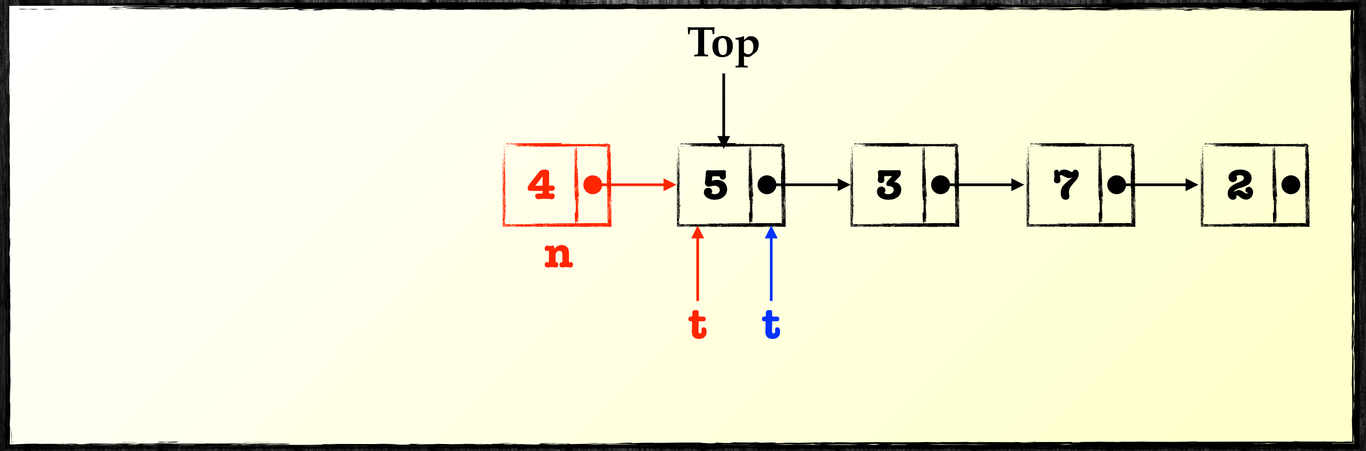
The Treiber Stack Algorithm

Linearization Policy

```
push(k):
Node t
1  n = new Node(k,-)
2  while (true)
3    t = Top
4    n.next = t
5    if (CAS (Top,t,n))
6      exit
```

```
pop:
Node t
1  while (true)
2    t = Top
3    if (t = NULL)
4      return *
5      exit
6    if (CAS (Top,t,t.next))
7      return t.val
8      exit
```



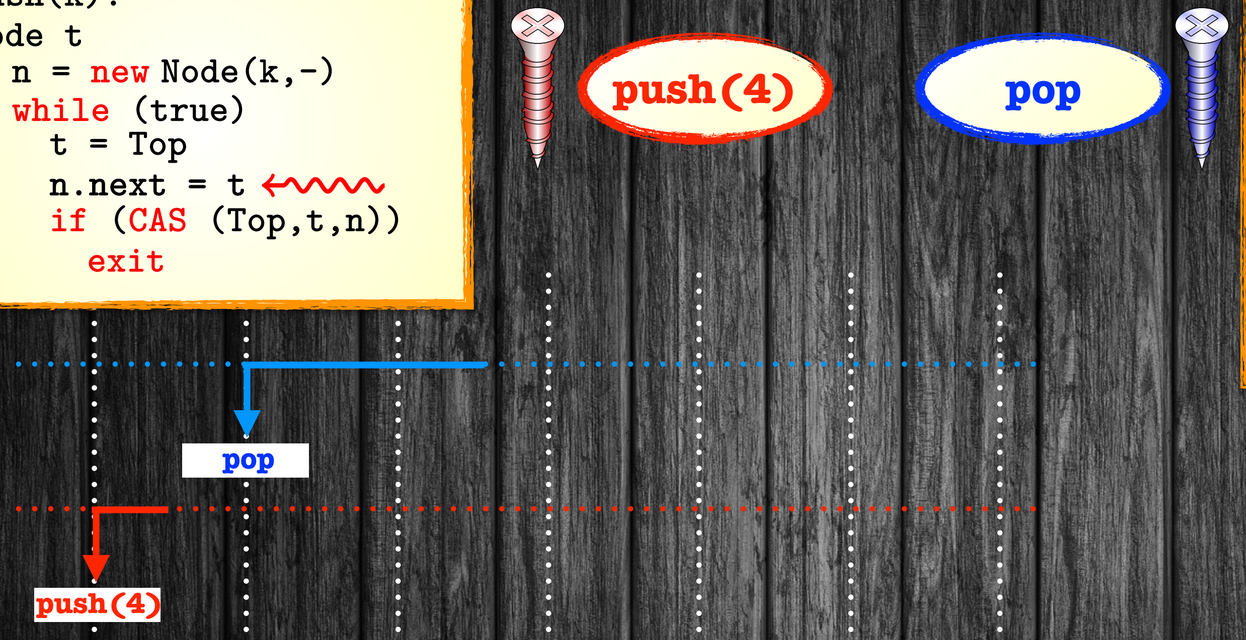


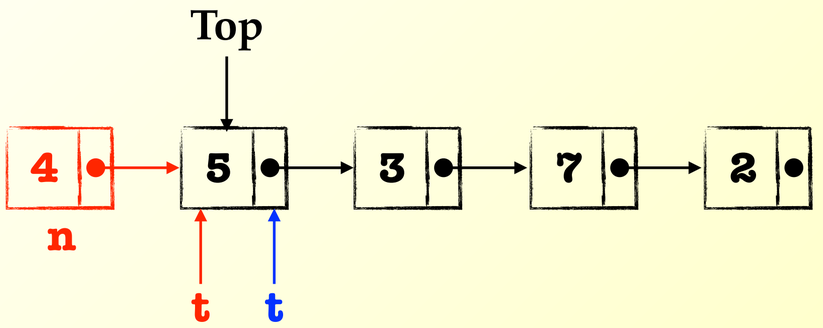
The Treiber Stack Algorithm

Linearization Policy

```
push(k):
Node t
1  n = new Node(k,-)
2  while (true)
3    t = Top
4    n.next = t
5    if (CAS (Top,t,n))
6      exit
```

```
pop:
Node t
1  while (true)
2    t = Top
3    if (t = NULL)
4      return *
5      exit
6    if (CAS (Top,t,t.next))
7      return t.val
8      exit
```



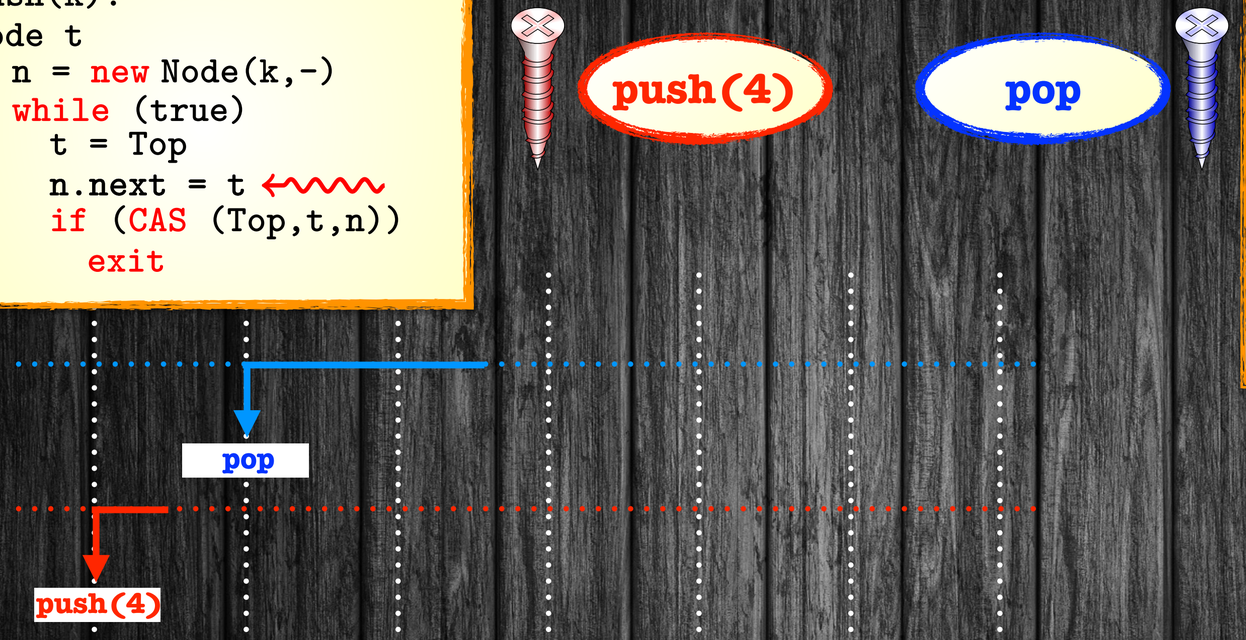


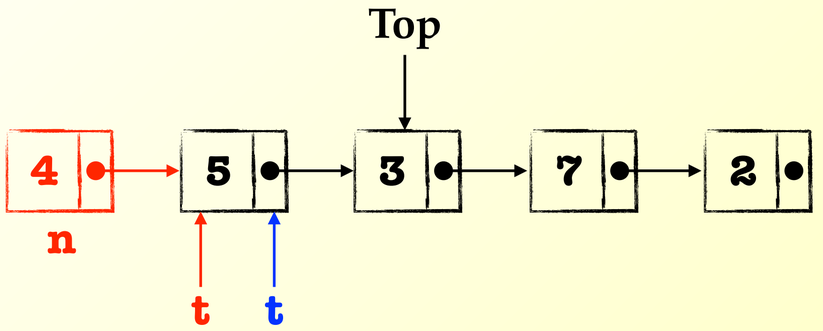
The Treiber Stack Algorithm

Linearization Policy

```
push(k):
Node t
1  n = new Node(k,-)
2  while (true)
3    t = Top
4    n.next = t
5    if (CAS (Top,t,n))
6      exit
```

```
pop:
Node t
1  while (true)
2    t = Top
3    if (t = NULL)
4      return *
5      exit
6    if (CAS (Top,t,t.next))
7      return t.val
8      exit
```



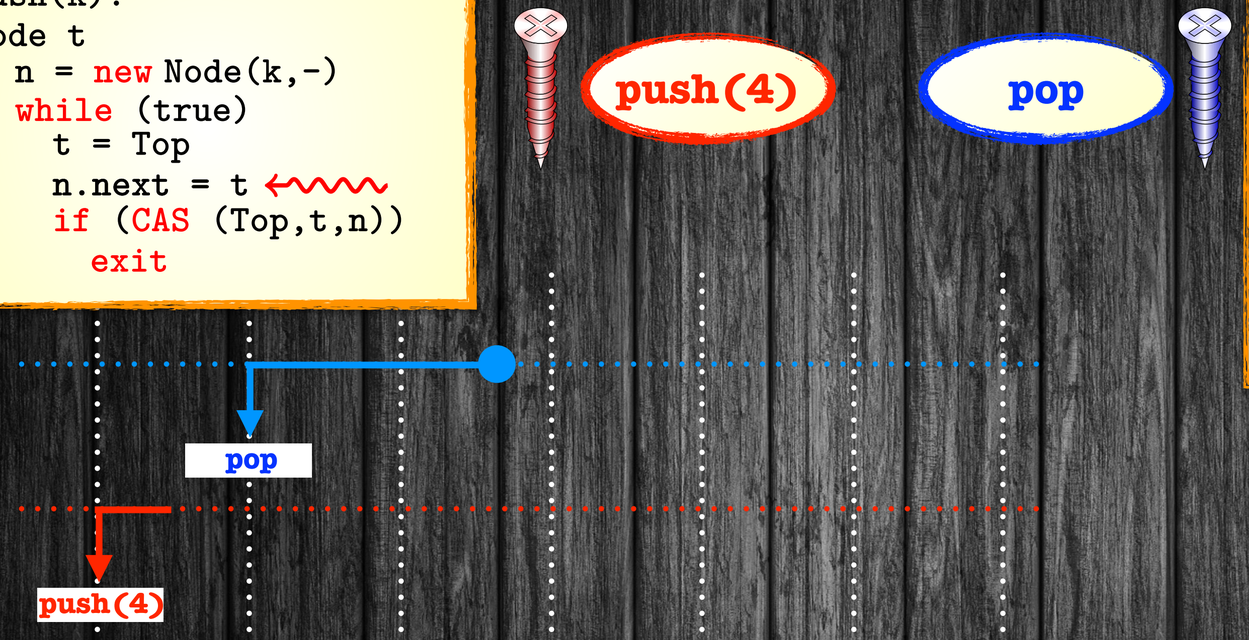


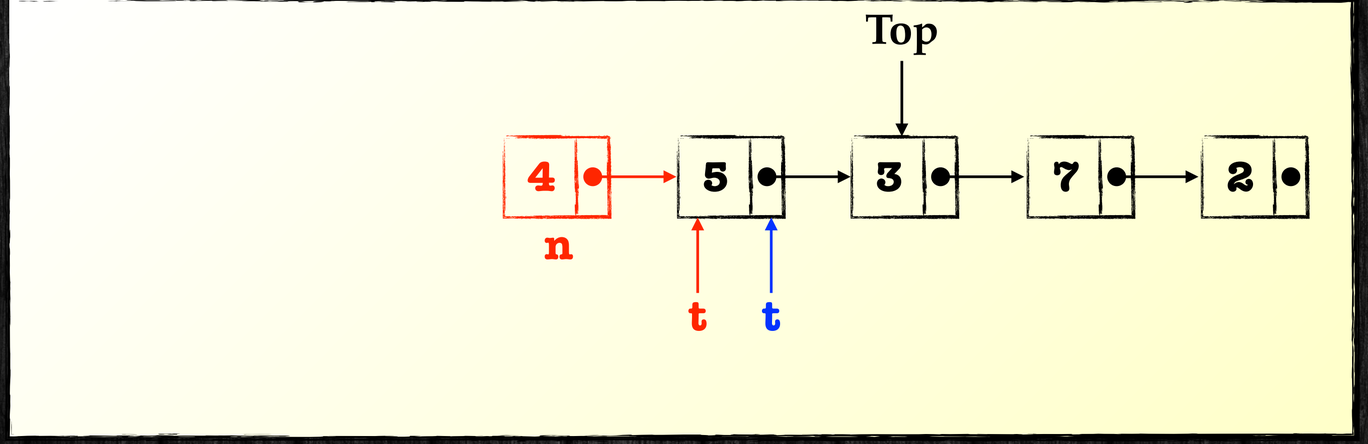
The Treiber Stack Algorithm

Linearization Policy

```
push(k):
Node t
1  n = new Node(k,-)
2  while (true)
3    t = Top
4    n.next = t
5    if (CAS (Top,t,n))
6      exit
```

```
pop:
Node t
1  while (true)
2    t = Top
3    if (t = NULL)
4      return *
5      exit
6    if (CAS (Top,t,t.next))
7      return t.val
8      exit
```



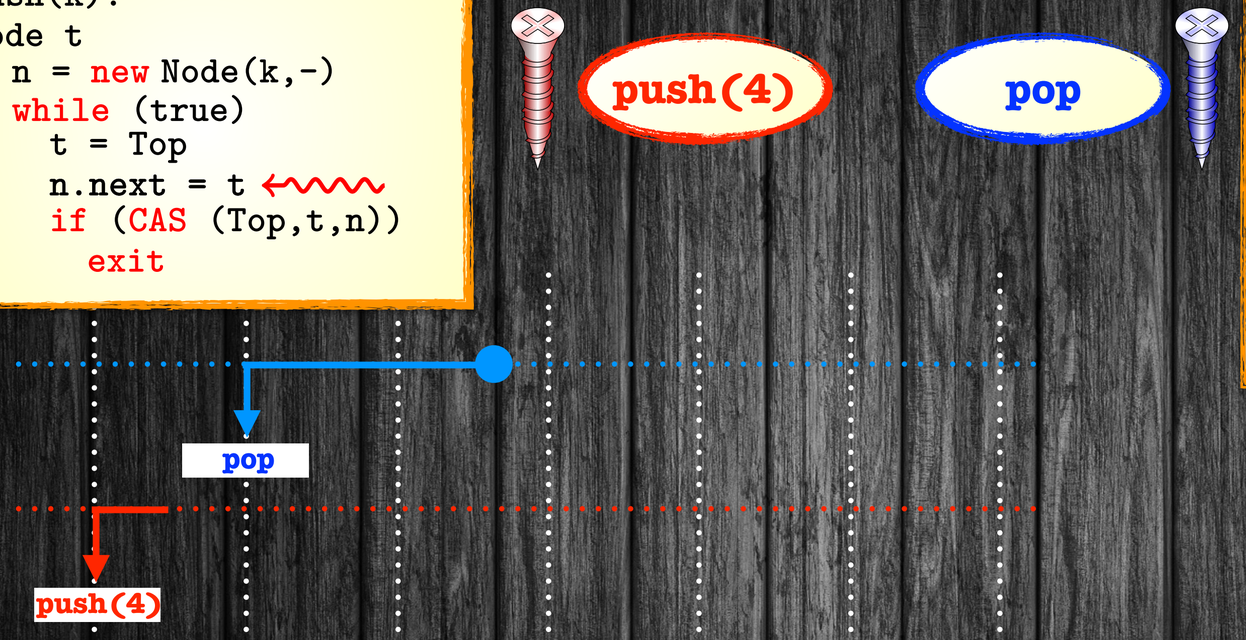


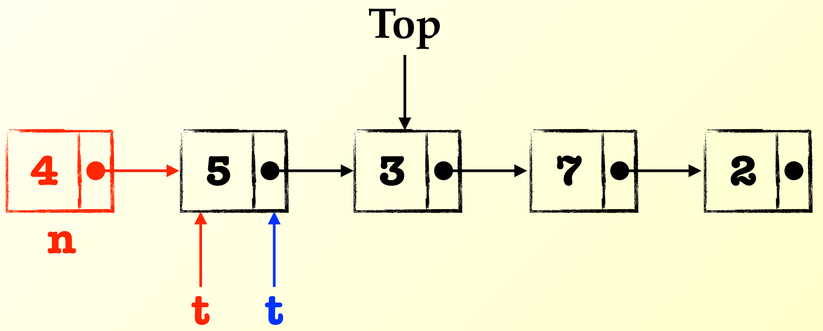
The Treiber Stack Algorithm

Linearization Policy

```
push(k):
Node t
1  n = new Node(k,-)
2  while (true)
3    t = Top
4    n.next = t
5    if (CAS (Top,t,n))
6      exit
```

```
pop:
Node t
1  while (true)
2    t = Top
3    if (t = NULL)
4      return *
5      exit
6    if (CAS (Top,t,t.next))
7      return t.val
8      exit
```



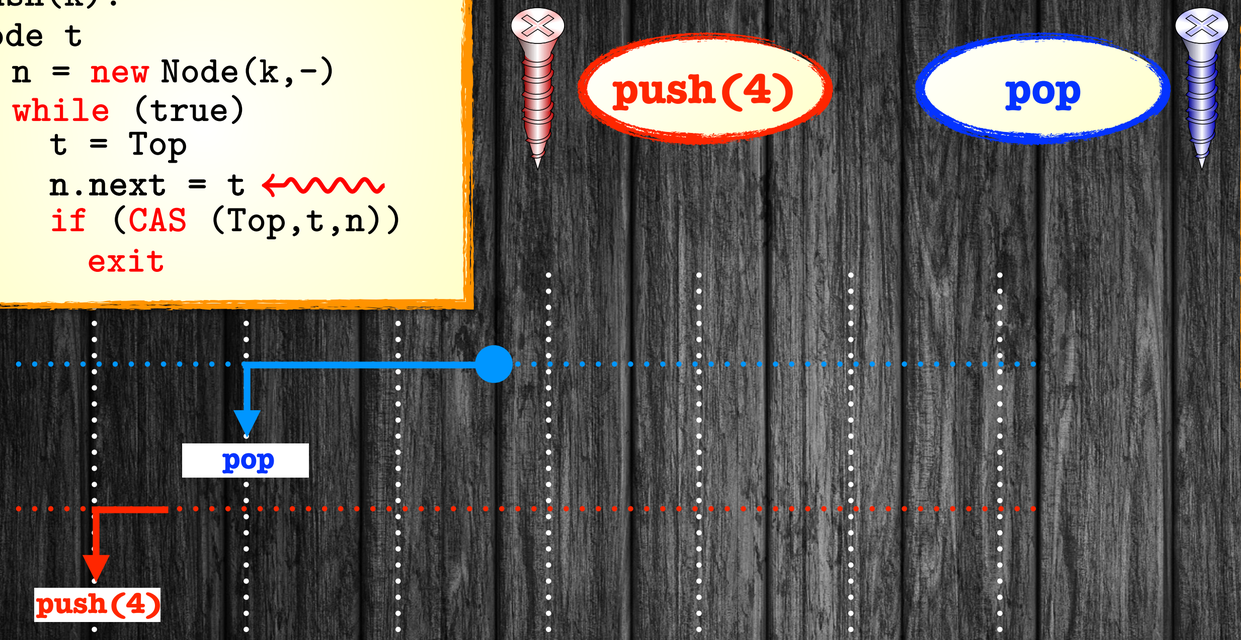


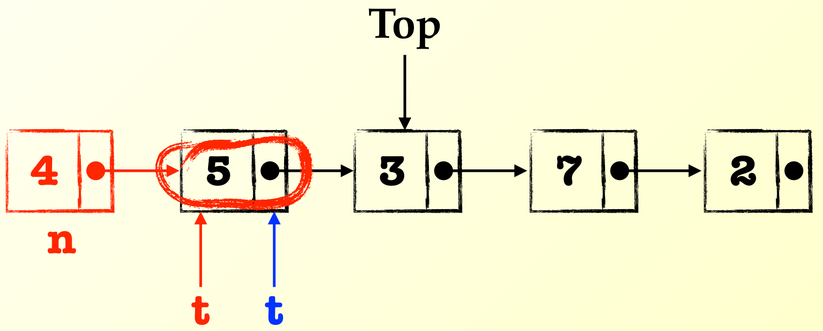
The Treiber Stack Algorithm

Linearization Policy

```
push(k):
Node t
1  n = new Node(k,-)
2  while (true)
3    t = Top
4    n.next = t
5    if (CAS (Top,t,n))
6      exit
```

```
pop:
Node t
1  while (true)
2    t = Top
3    if (t = NULL)
4      return *
5      exit
6    if (CAS (Top,t,t.next))
7      return t.val
8      exit
```



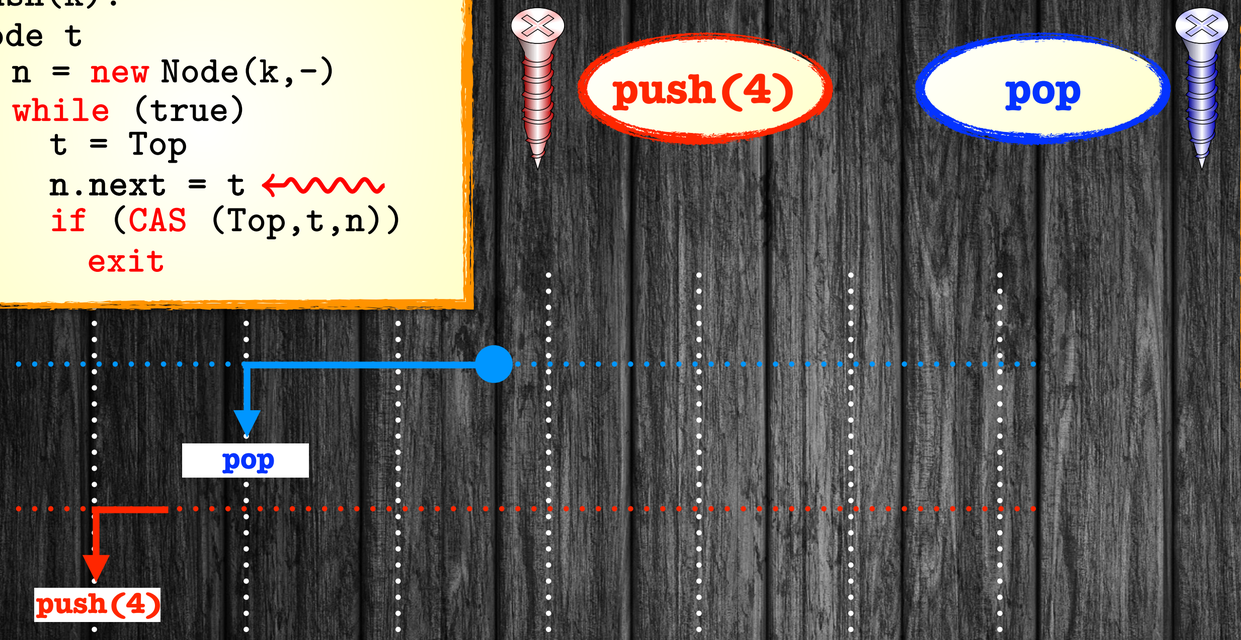


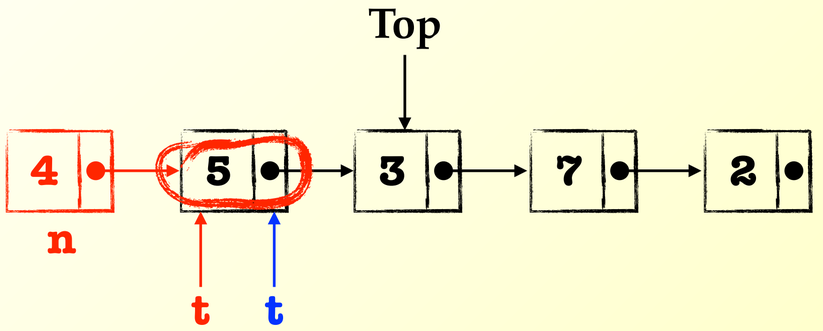
The Treiber Stack Algorithm

Linearization Policy

```
push(k):
Node t
1  n = new Node(k,-)
2  while (true)
3    t = Top
4    n.next = t
5    if (CAS (Top,t,n))
6      exit
```

```
pop:
Node t
1  while (true)
2    t = Top
3    if (t = NULL)
4      return *
5      exit
6    if (CAS (Top,t,t.next))
7      return t.val
8      exit
```



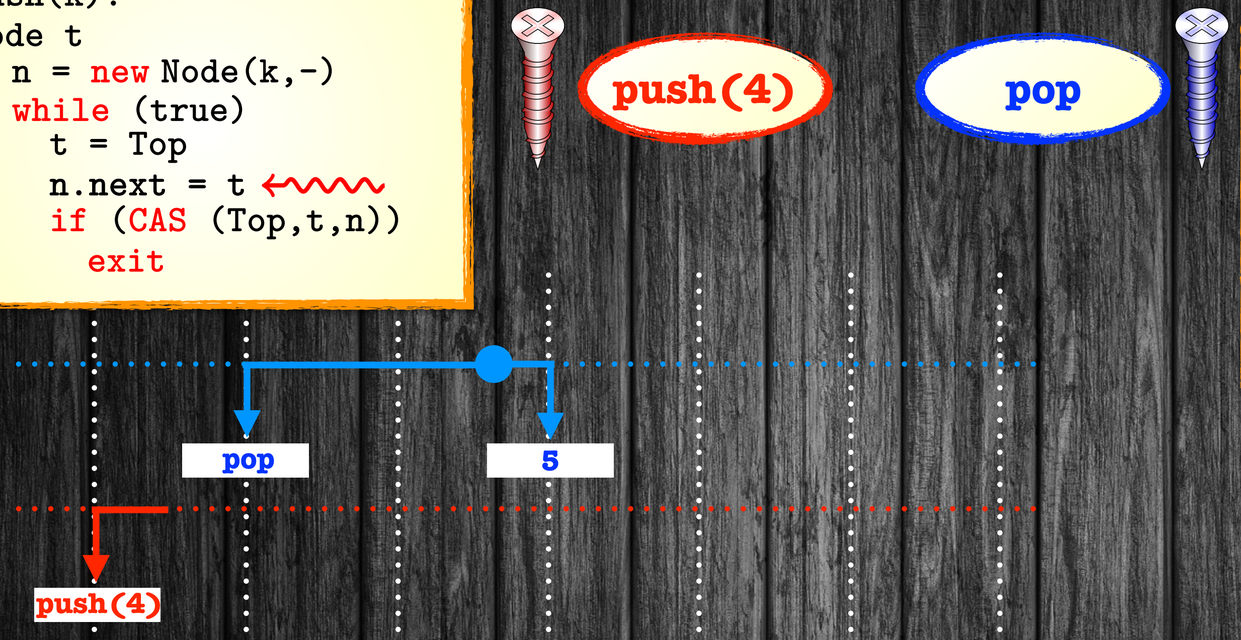


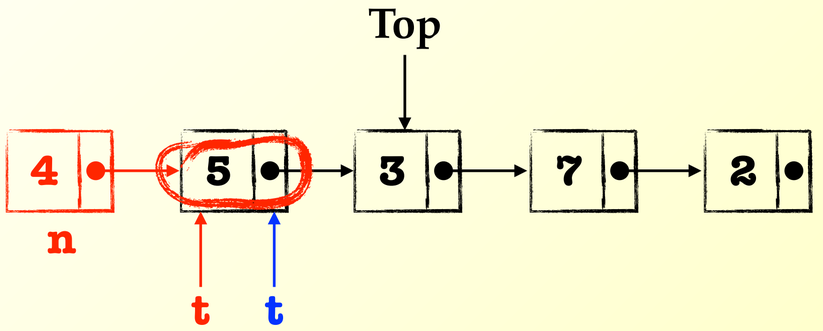
The Treiber Stack Algorithm

Linearization Policy

```
push(k):
Node t
1  n = new Node(k,-)
2  while (true)
3    t = Top
4    n.next = t
5    if (CAS (Top,t,n))
6      exit
```

```
pop:
Node t
1  while (true)
2    t = Top
3    if (t = NULL)
4      return *
5      exit
6    if (CAS (Top,t,t.next))
7      return t.val
8      exit
```





The Treiber Stack Algorithm

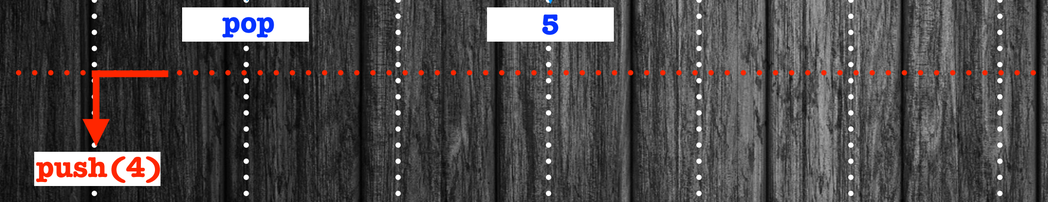
Linearization Policy

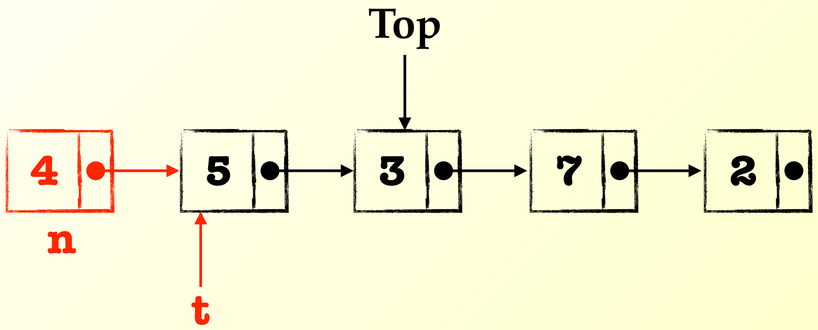
```
push(k):
Node t
1  n = new Node(k,-)
2  while (true)
3    t = Top
4    n.next = t
5    if (CAS (Top,t,n))
6      exit
```



push(4)

```
pop:
Node t
1  while (true)
2    t = Top
3    if (t = NULL)
4      return *
5      exit
6    if (CAS (Top,t,t.next))
7      return t.val
8      exit
```





The Treiber Stack Algorithm

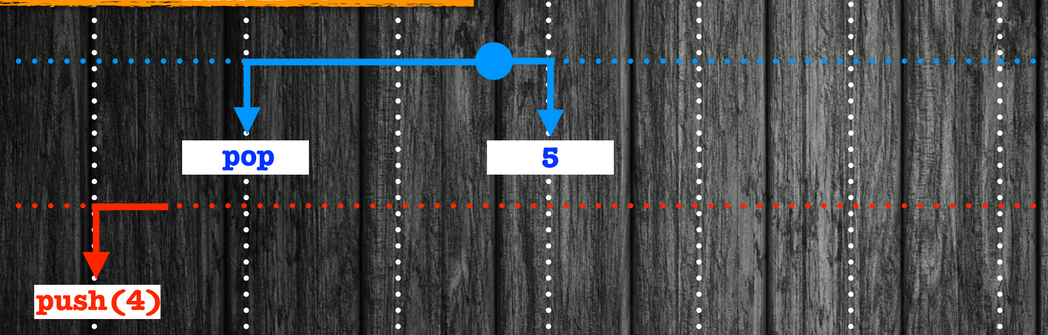
Linearization Policy

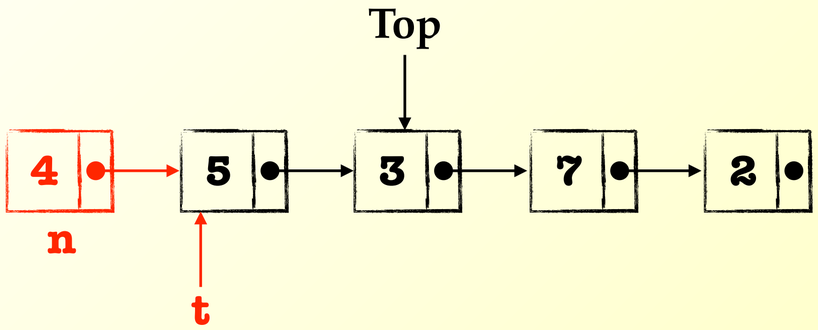
```
push(k):
Node t
1  n = new Node(k,-)
2  while (true)
3    t = Top
4    n.next = t
5    if (CAS (Top,t,n))
6      exit
```



push(4)

```
pop:
Node t
1  while (true)
2    t = Top
3    if (t = NULL)
4      return *
5      exit
6    if (CAS (Top,t,t.next))
7      return t.val
8      exit
```



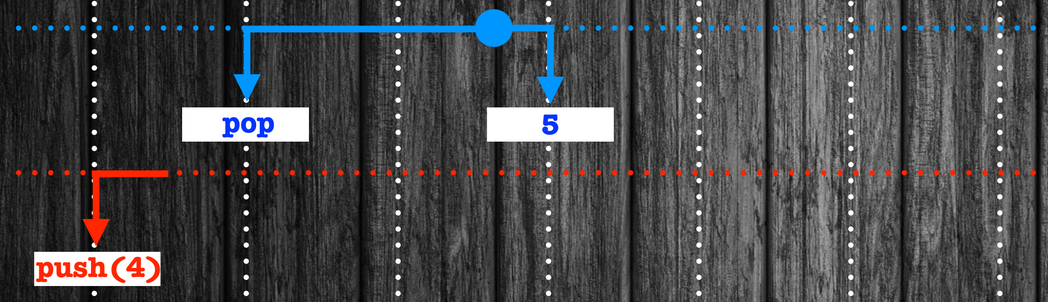


The Treiber Stack Algorithm

```
push(k):
Node t
1  n = new Node(k,-)
2  while (true)
3    t = Top
4    n.next = t
5    if (CAS (Top,t,n))
6      exit
```

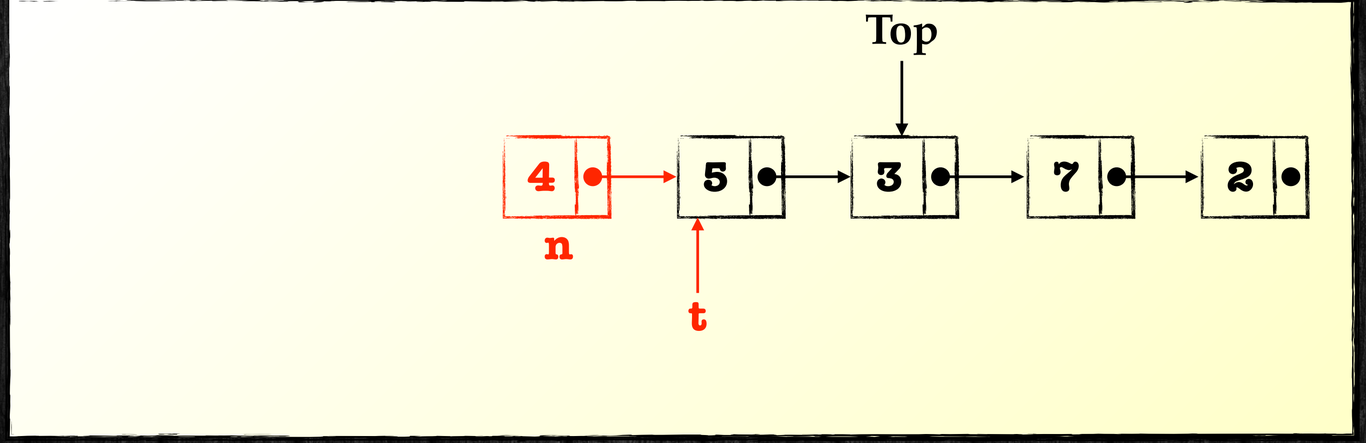


push(4)



Linearization Policy

```
pop:
Node t
1  while (true)
2    t = Top
3    if (t = NULL)
4      return *
5      exit
6    if (CAS (Top,t,t.next))
7      return t.val
8      exit
```

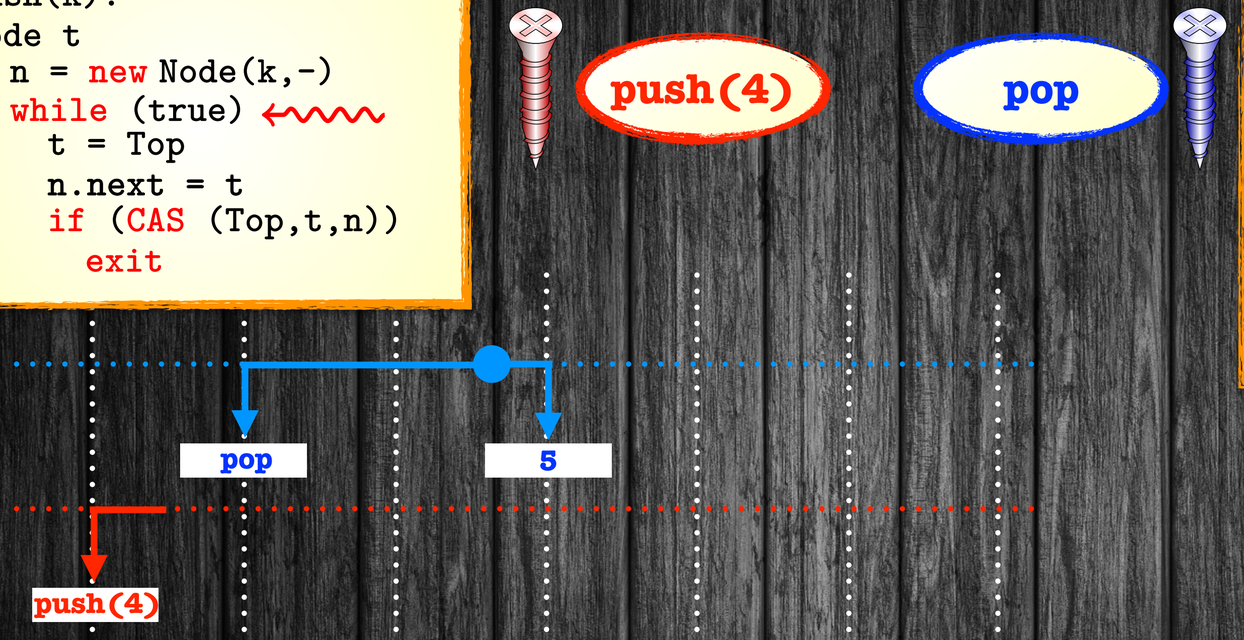


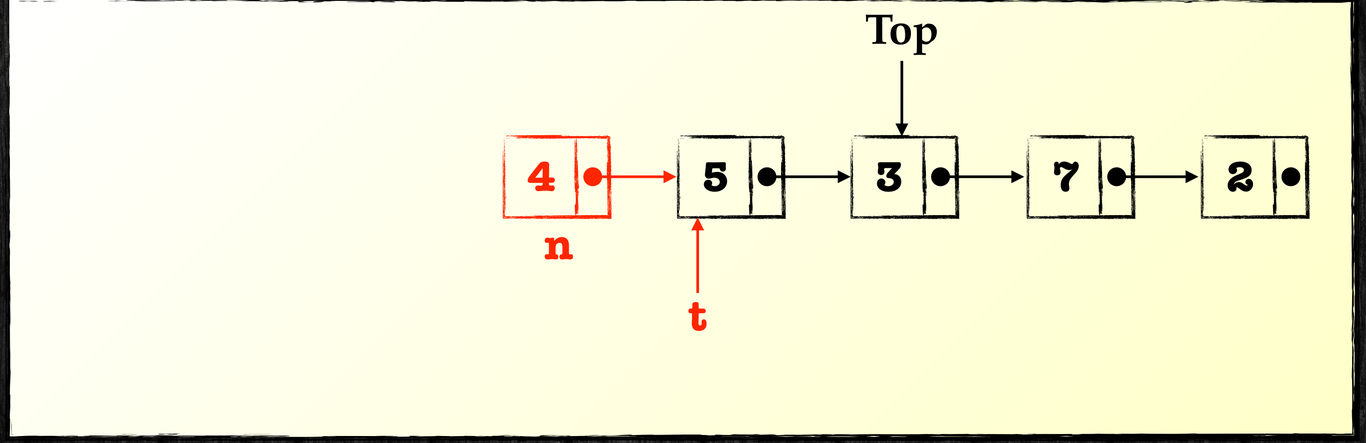
The Treiber Stack Algorithm

Linearization Policy

```
push(k):
Node t
1  n = new Node(k,-)
2  while (true)
3    t = Top
4    n.next = t
5    if (CAS (Top,t,n))
6      exit
```

```
pop:
Node t
1  while (true)
2    t = Top
3    if (t = NULL)
4      return *
5      exit
6    if (CAS (Top,t,t.next))
7      return t.val
8      exit
```



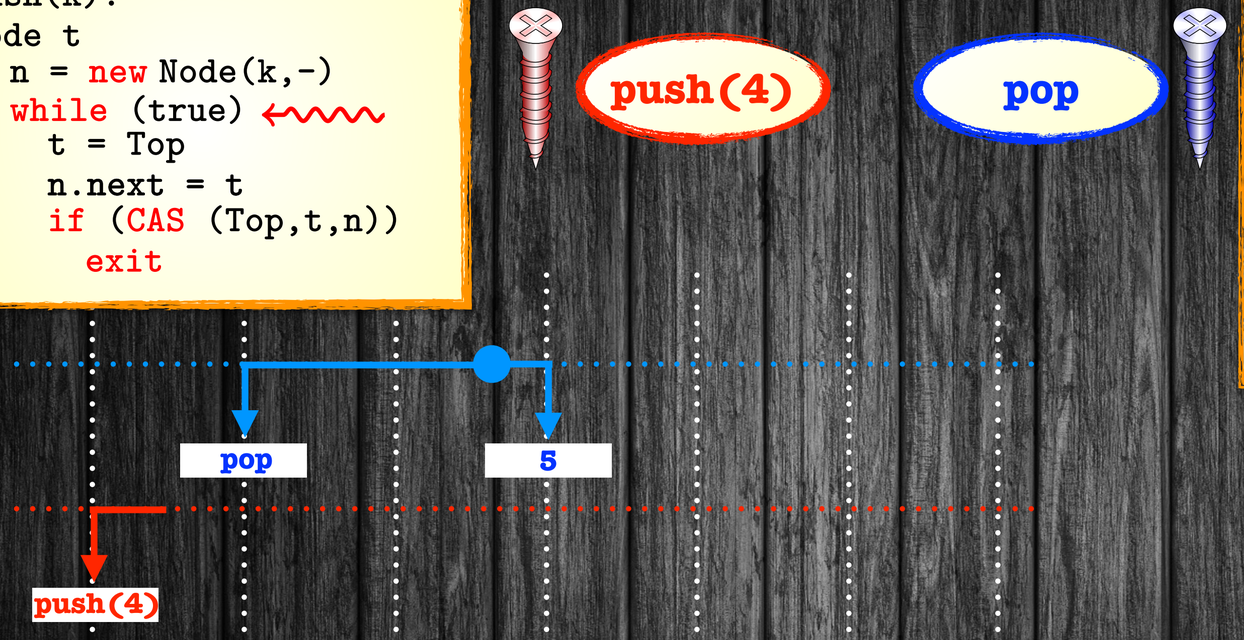


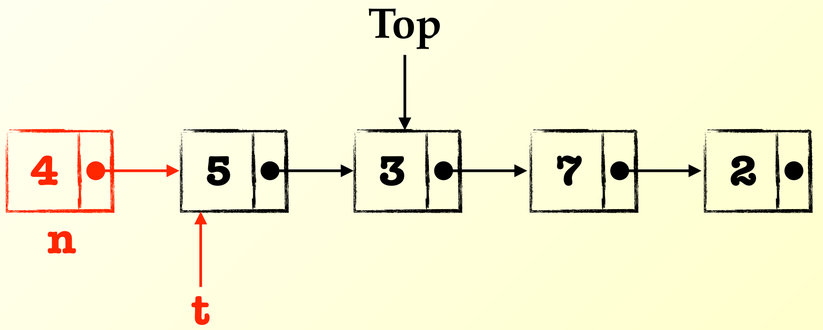
The Treiber Stack Algorithm

Linearization Policy

```
push(k):
Node t
1  n = new Node(k,-)
2  while (true)
3    t = Top
4    n.next = t
5    if (CAS (Top,t,n))
6      exit
```

```
pop:
Node t
1  while (true)
2    t = Top
3    if (t = NULL)
4      return *
5      exit
6    if (CAS (Top,t,t.next))
7      return t.val
8      exit
```



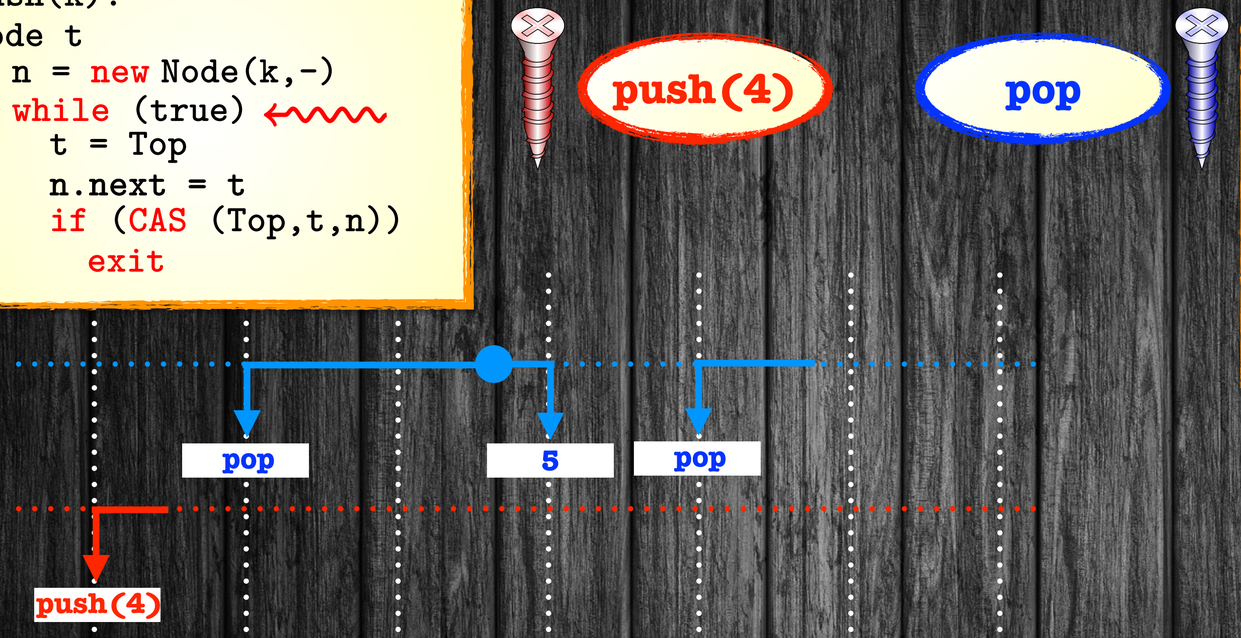


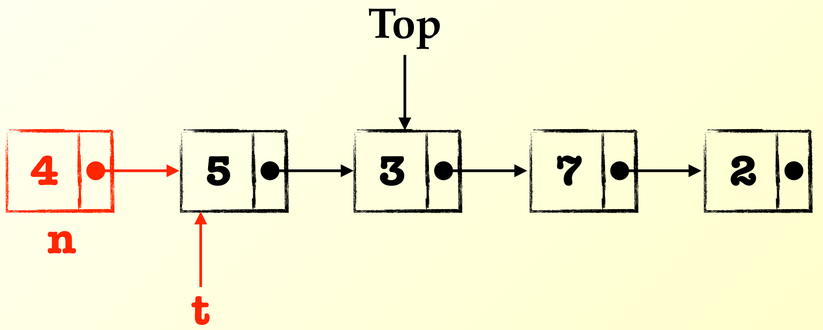
The Treiber Stack Algorithm

Linearization Policy

```
push(k):
Node t
1  n = new Node(k,-)
2  while (true)
3    t = Top
4    n.next = t
5    if (CAS (Top,t,n))
6      exit
```

```
pop:
Node t
1  while (true)
2    t = Top
3    if (t = NULL)
4      return *
5      exit
6    if (CAS (Top,t,t.next))
7      return t.val
8      exit
```



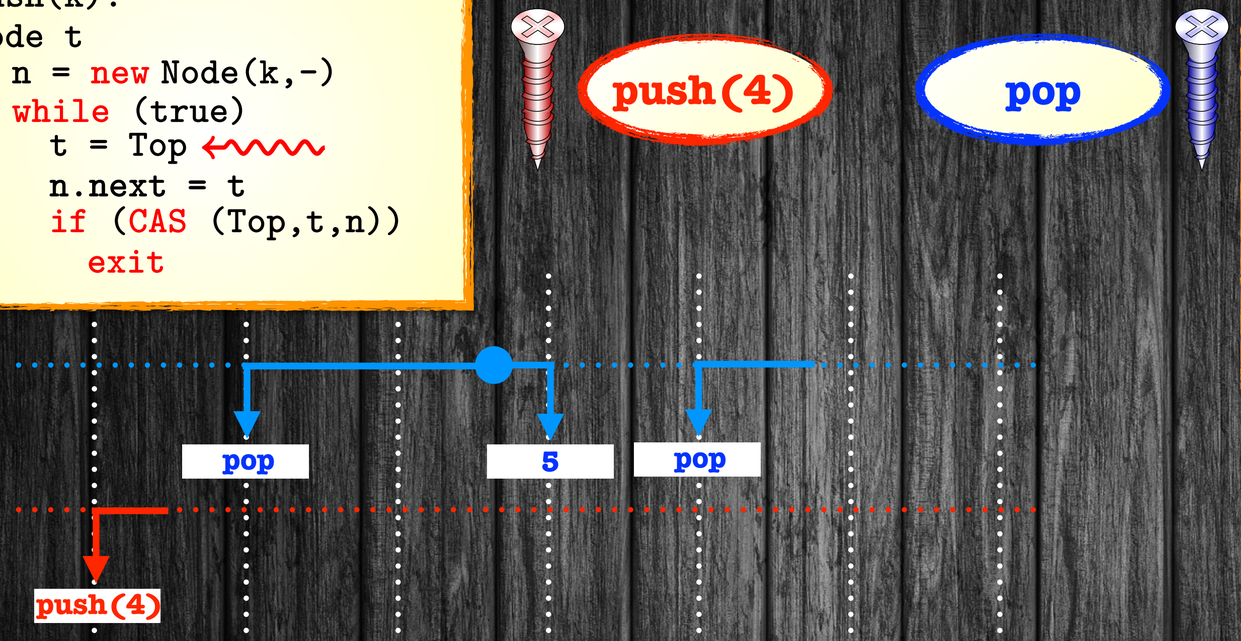


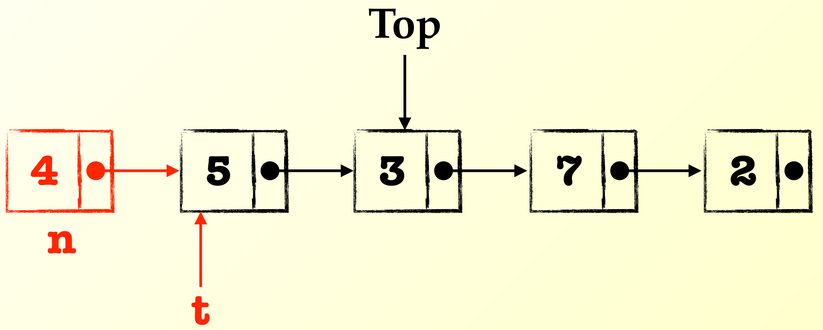
The Treiber Stack Algorithm

Linearization Policy

```
push(k):
Node t
1  n = new Node(k,-)
2  while (true)
3    t = Top
4    n.next = t
5    if (CAS (Top,t,n))
6      exit
```

```
pop:
Node t
1  while (true)
2    t = Top
3    if (t = NULL)
4      return *
5      exit
6    if (CAS (Top,t,t.next))
7      return t.val
8      exit
```



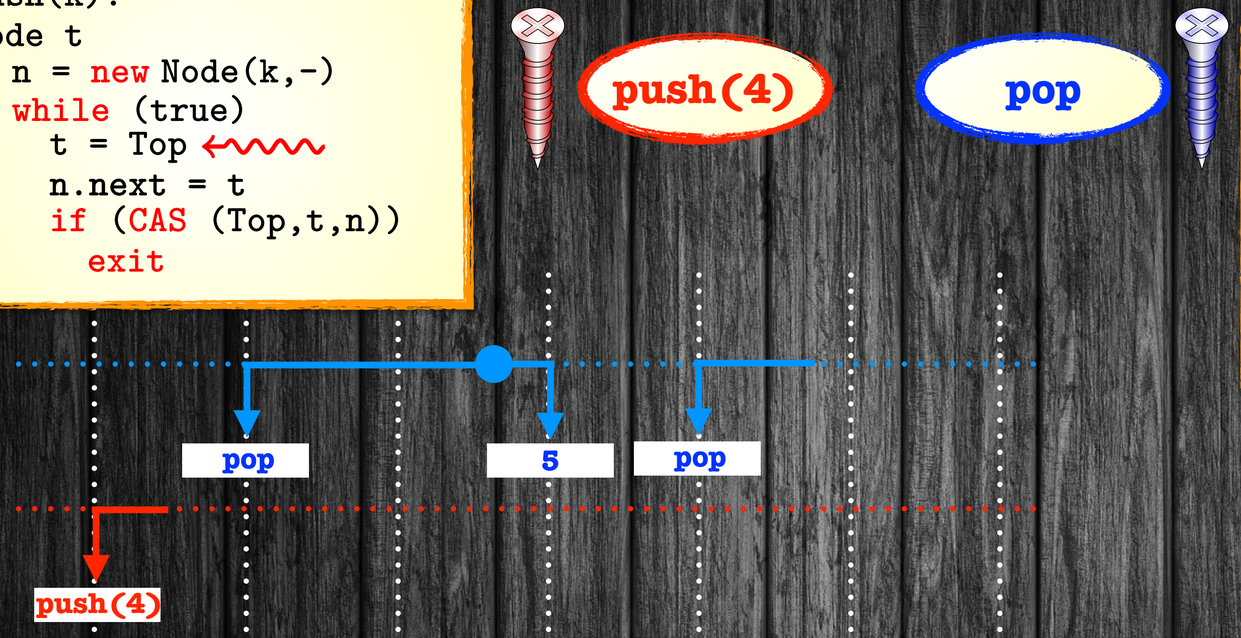


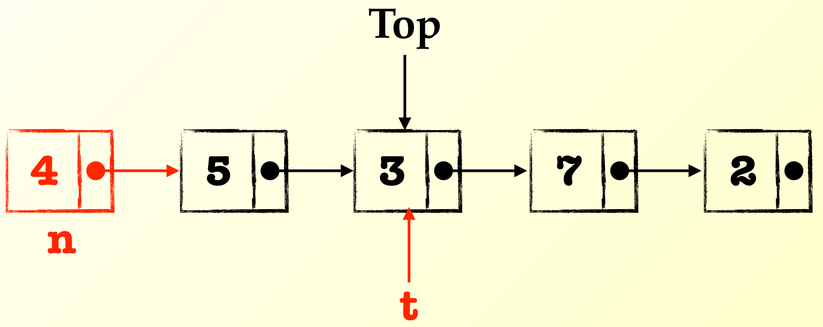
The Treiber Stack Algorithm

Linearization Policy

```
push(k):
Node t
1  n = new Node(k,-)
2  while (true)
3    t = Top
4    n.next = t
5    if (CAS (Top,t,n))
6      exit
```

```
pop:
Node t
1  while (true)
2    t = Top
3    if (t = NULL)
4      return *
5      exit
6    if (CAS (Top,t,t.next))
7      return t.val
8      exit
```



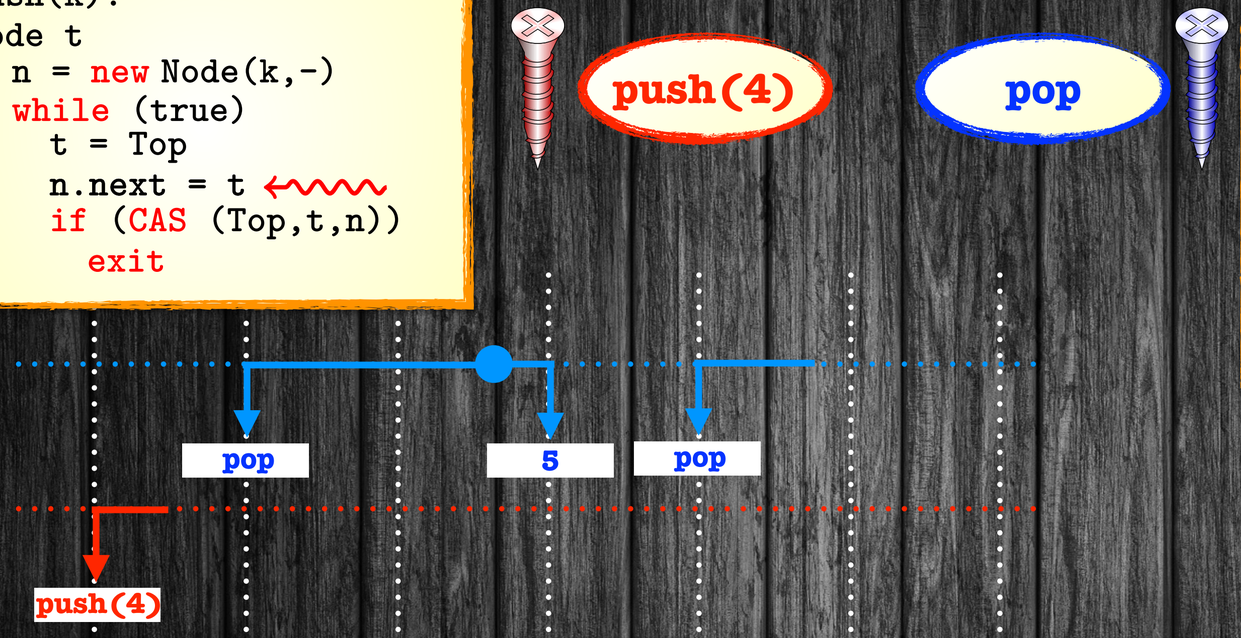


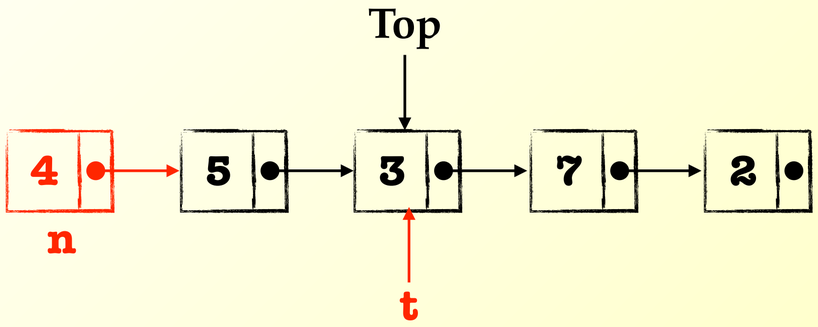
The Treiber Stack Algorithm

Linearization Policy

```
push(k):
Node t
1  n = new Node(k,-)
2  while (true)
3    t = Top
4    n.next = t
5    if (CAS (Top,t,n))
6      exit
```

```
pop:
Node t
1  while (true)
2    t = Top
3    if (t = NULL)
4      return *
5      exit
6    if (CAS (Top,t,t.next))
7      return t.val
8      exit
```



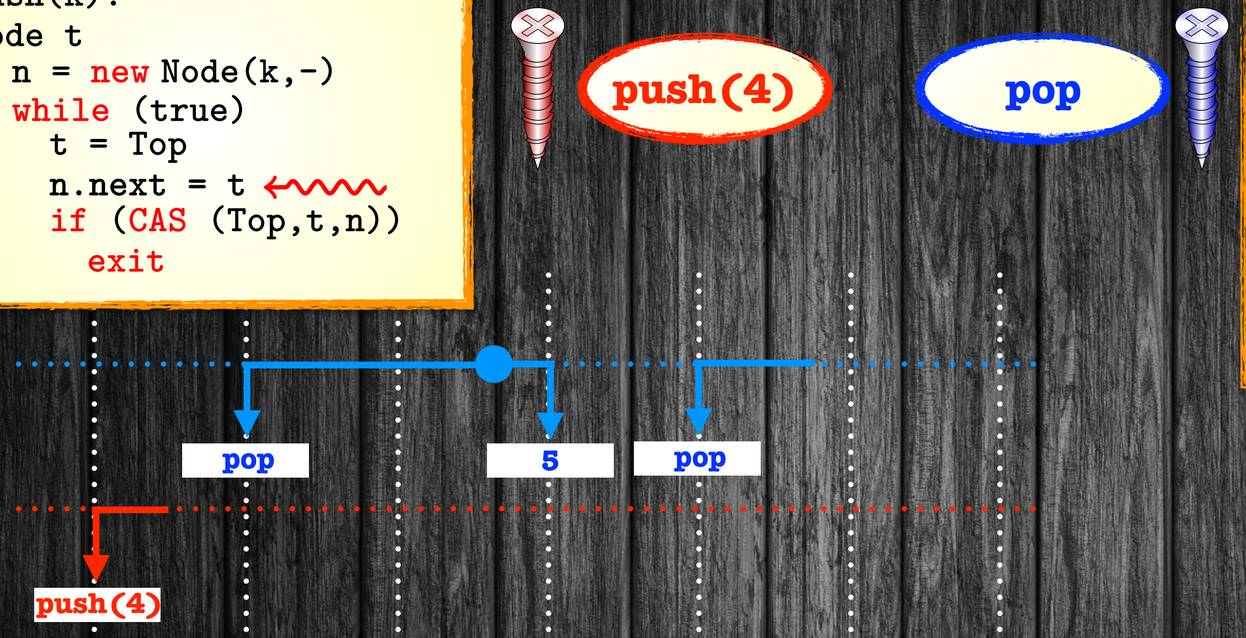


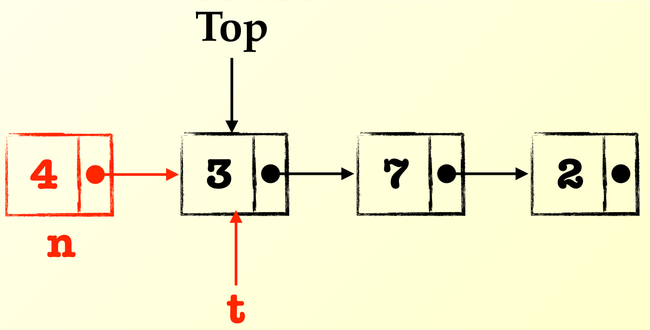
The Treiber Stack Algorithm

Linearization Policy

```
push(k):
Node t
1  n = new Node(k,-)
2  while (true)
3    t = Top
4    n.next = t
5    if (CAS (Top,t,n))
6      exit
```

```
pop:
Node t
1  while (true)
2    t = Top
3    if (t = NULL)
4      return *
5      exit
6    if (CAS (Top,t,t.next))
7      return t.val
8      exit
```





The Treiber Stack Algorithm

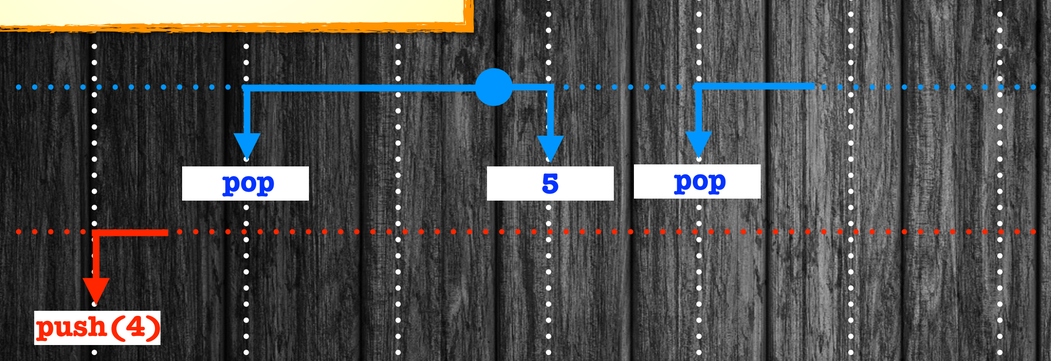
Linearization Policy

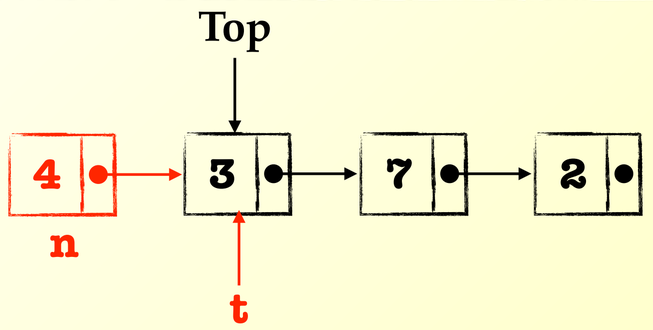
```
push(k):
Node t
1  n = new Node(k,-)
2  while (true)
3    t = Top
4    n.next = t
5    if (CAS (Top,t,n))
6      exit
```

push(4)

pop

```
pop:
Node t
1  while (true)
2    t = Top
3    if (t = NULL)
4      return *
5      exit
6    if (CAS (Top,t,t.next))
7      return t.val
8      exit
```





The Treiber Stack Algorithm

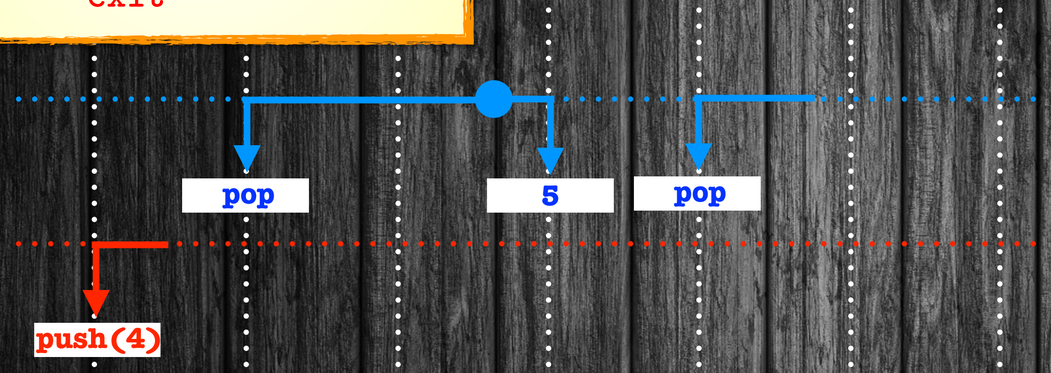
Linearization Policy

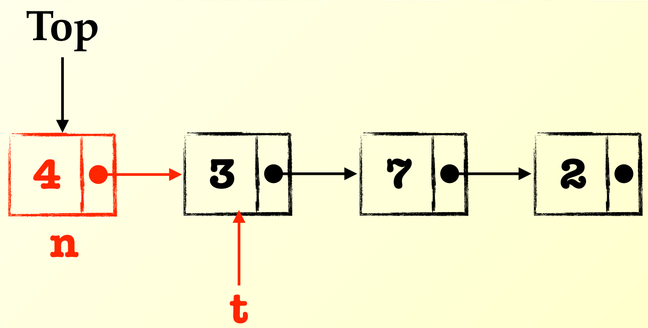
```
push(k):
Node t
1  n = new Node(k,-)
2  while (true)
3    t = Top
4    n.next = t
5    if (CAS (Top,t,n))
6      exit
```

push(4)

pop

```
pop:
Node t
1  while (true)
2    t = Top
3    if (t = NULL)
4      return *
5      exit
6    if (CAS (Top,t,t.next))
7      return t.val
8      exit
```





The Treiber Stack Algorithm

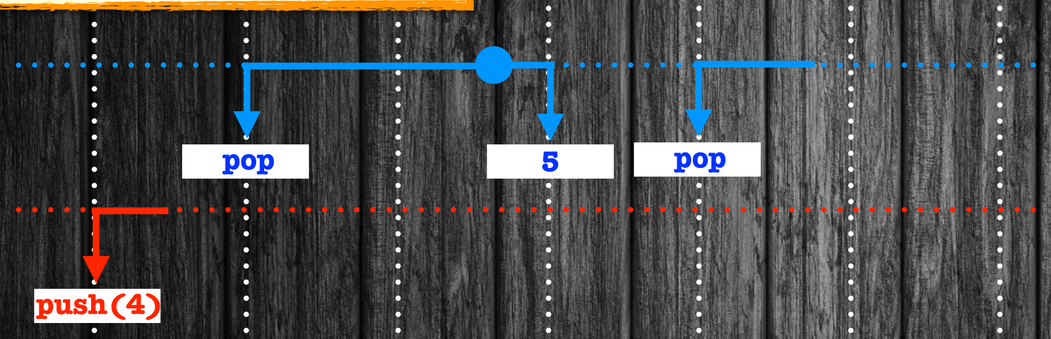
Linearization Policy

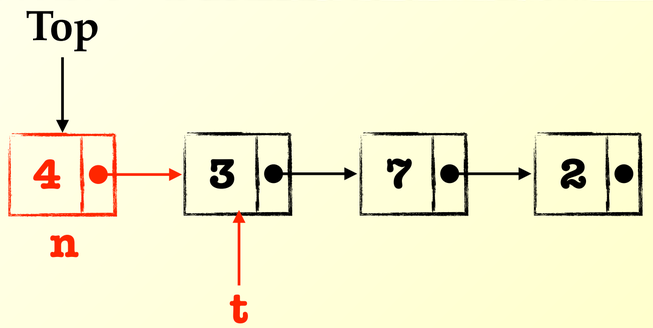
```
push(k):
Node t
1  n = new Node(k,-)
2  while (true)
3    t = Top
4    n.next = t
5    if (CAS (Top,t,n))
6      exit
```

push(4)

pop

```
pop:
Node t
1  while (true)
2    t = Top
3    if (t = NULL)
4      return *
5      exit
6    if (CAS (Top,t,t.next))
7      return t.val
8      exit
```



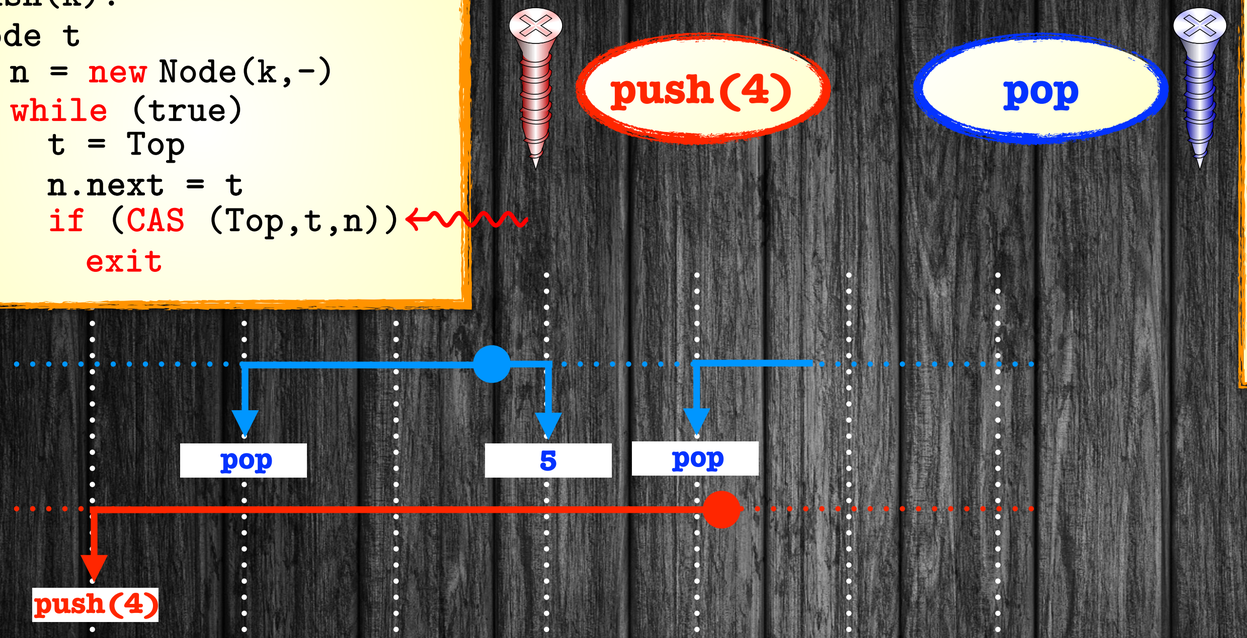


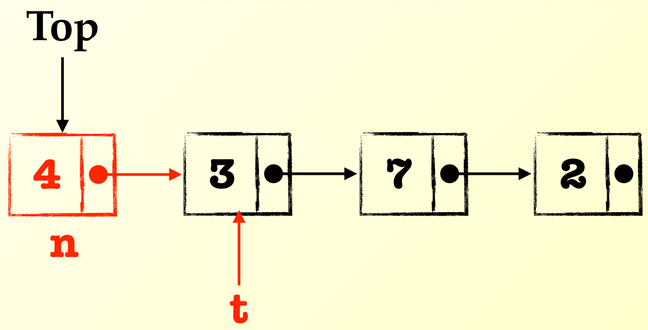
The Treiber Stack Algorithm

Linearization Policy

```
push(k):
Node t
1  n = new Node(k,-)
2  while (true)
3    t = Top
4    n.next = t
5    if (CAS (Top,t,n))
6      exit
```

```
pop:
Node t
1  while (true)
2    t = Top
3    if (t = NULL)
4      return *
5      exit
6    if (CAS (Top,t,t.next))
7      return t.val
8      exit
```



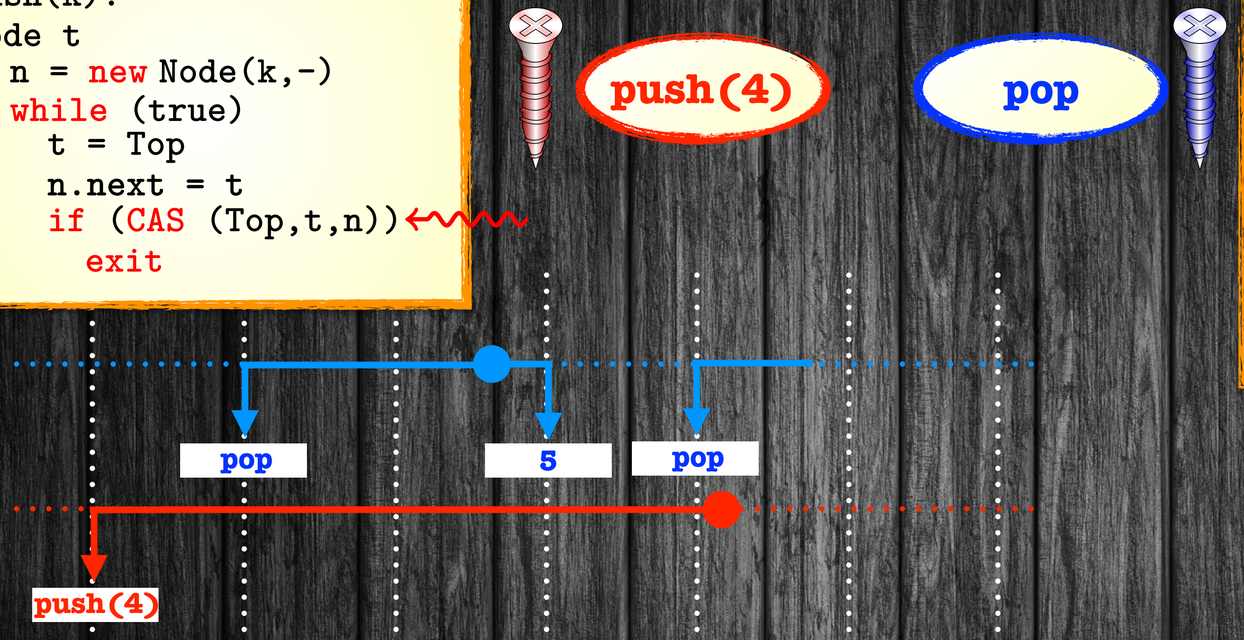


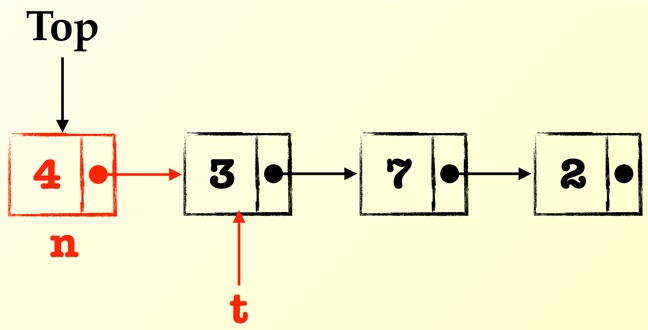
The Treiber Stack Algorithm

Linearization Policy

```
push(k):
Node t
1  n = new Node(k,-)
2  while (true)
3    t = Top
4    n.next = t
5    if (CAS (Top,t,n))
6      exit
```

```
pop:
Node t
1  while (true)
2    t = Top
3    if (t = NULL)
4      return *
5      exit
6    if (CAS (Top,t,t.next))
7      return t.val
8      exit
```





The Treiber Stack Algorithm

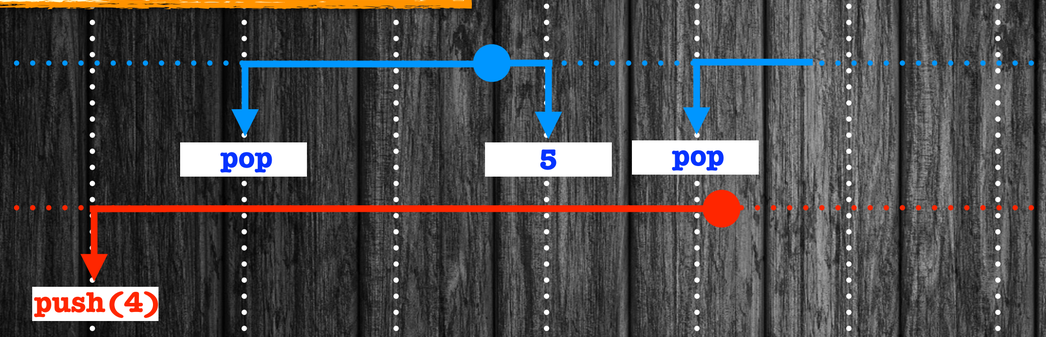
Linearization Policy

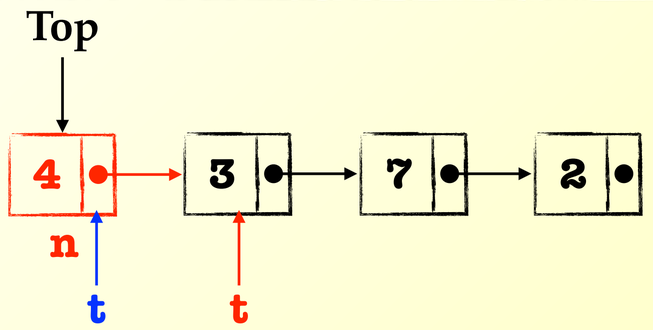
```
push(k):
Node t
1  n = new Node(k,-)
2  while (true)
3    t = Top
4    n.next = t
5    if (CAS (Top,t,n))
6      exit
```

push(4)

pop

```
pop:
Node t
1  while (true)
2    t = Top
3    if (t = NULL)
4      return *
5      exit
6    if (CAS (Top,t,t.next))
7      return t.val
8      exit
```



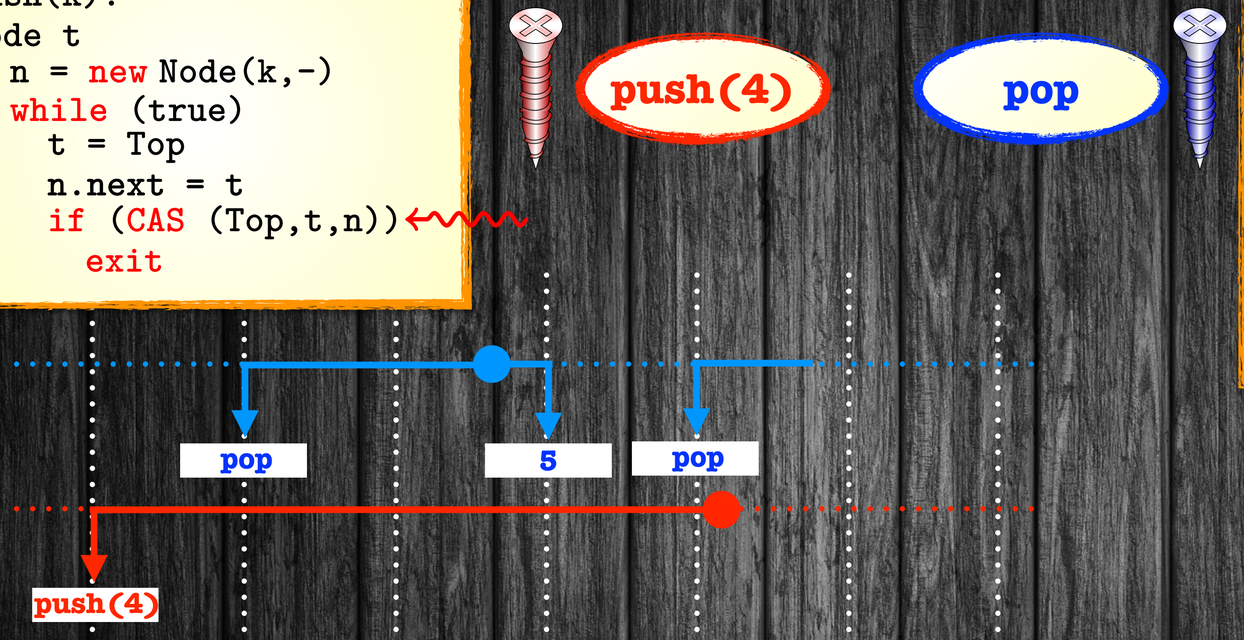


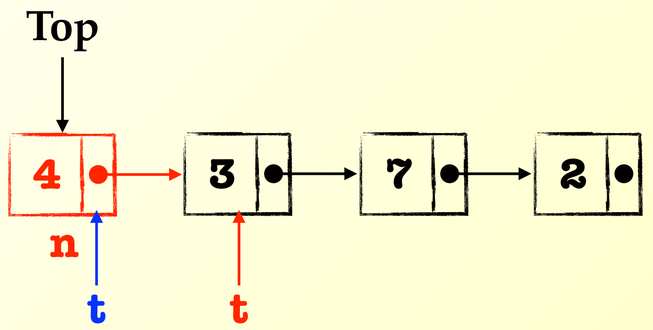
The Treiber Stack Algorithm

Linearization Policy

```
push(k):
Node t
1  n = new Node(k,-)
2  while (true)
3    t = Top
4    n.next = t
5    if (CAS (Top,t,n))
6      exit
```

```
pop:
Node t
1  while (true)
2    t = Top
3    if (t = NULL)
4      return *
5      exit
6    if (CAS (Top,t,t.next))
7      return t.val
8      exit
```



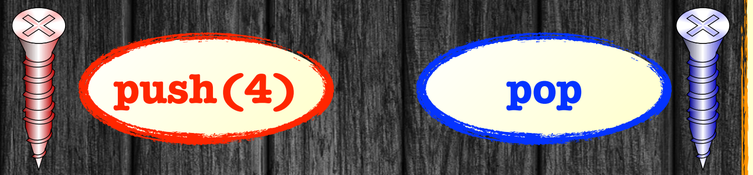


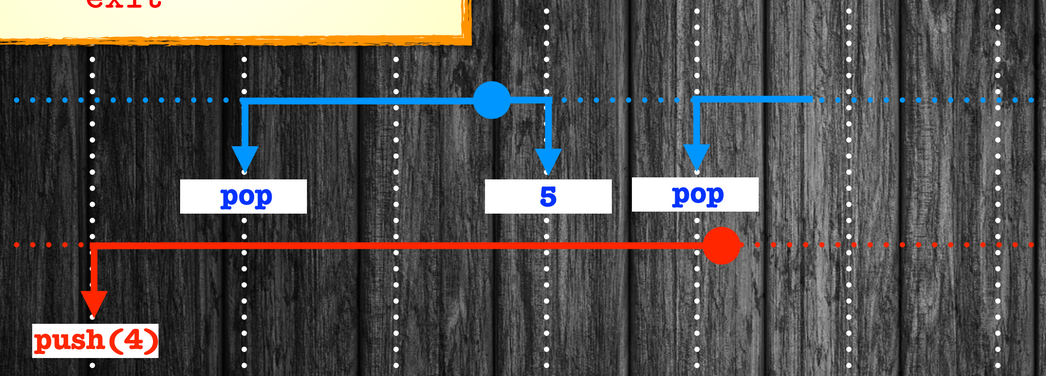
The Treiber Stack Algorithm

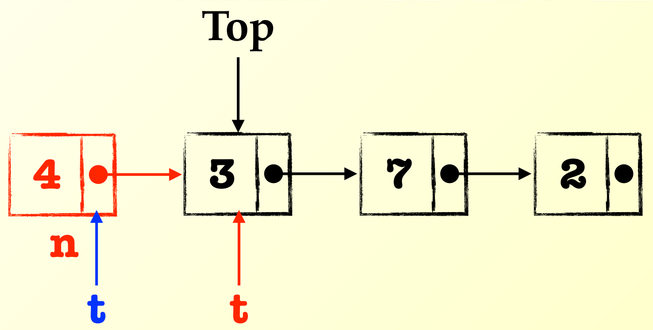
Linearization Policy

```
push(k):
Node t
1  n = new Node(k,-)
2  while (true)
3    t = Top
4    n.next = t
5    if (CAS (Top,t,n))
6      exit
```

```
pop:
Node t
1  while (true)
2    t = Top
3    if (t = NULL)
4      return *
5      exit
6    if (CAS (Top,t,t.next))
7      return t.val
8      exit
```





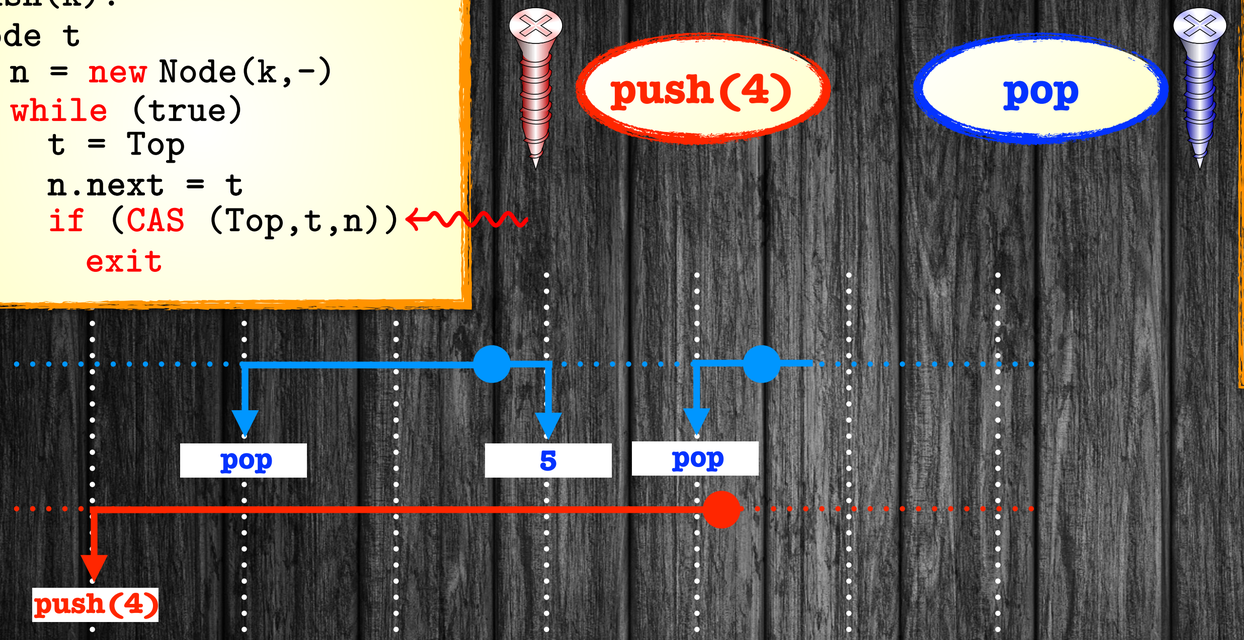


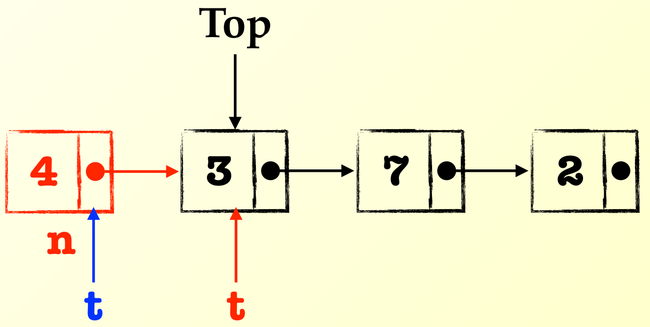
The Treiber Stack Algorithm

Linearization Policy

```
push(k):
Node t
1  n = new Node(k,-)
2  while (true)
3    t = Top
4    n.next = t
5    if (CAS (Top,t,n))
6      exit
```

```
pop:
Node t
1  while (true)
2    t = Top
3    if (t = NULL)
4      return *
5      exit
6    if (CAS (Top,t,t.next))
7      return t.val
8      exit
```





The Treiber Stack Algorithm

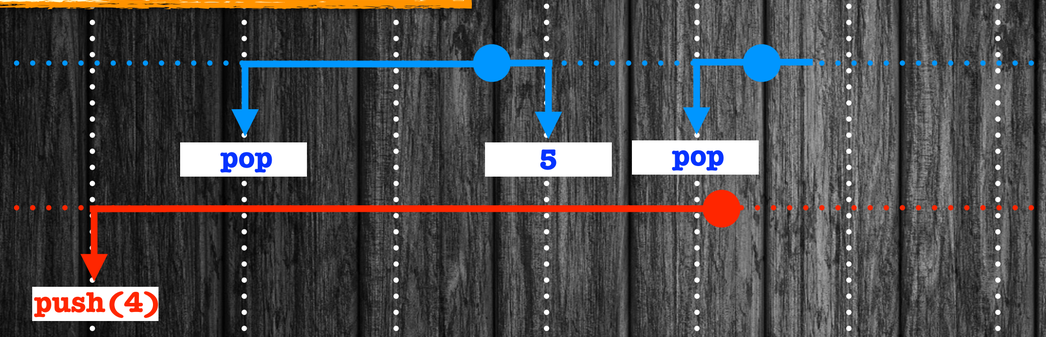
Linearization Policy

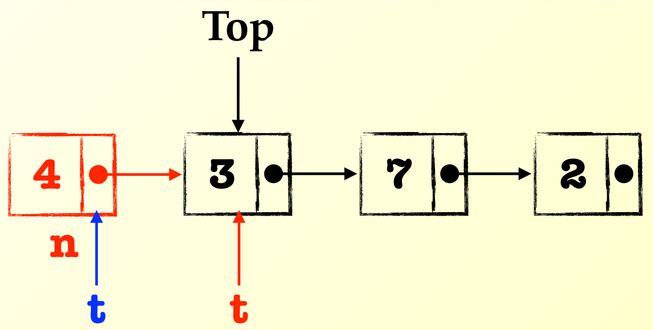
```
push(k):
Node t
1  n = new Node(k,-)
2  while (true)
3    t = Top
4    n.next = t
5    if (CAS (Top,t,n))
6      exit
```

push(4)

pop

```
pop:
Node t
1  while (true)
2    t = Top
3    if (t = NULL)
4      return *
5      exit
6    if (CAS (Top,t,t.next))
7      return t.val
8      exit
```





The Treiber Stack Algorithm

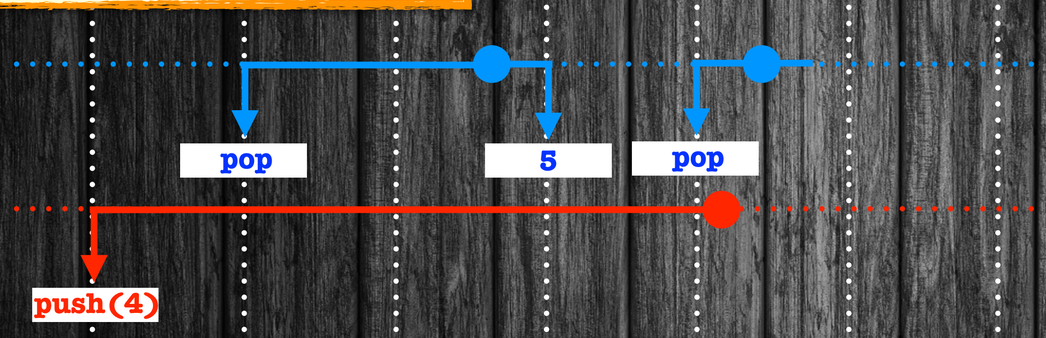
Linearization Policy

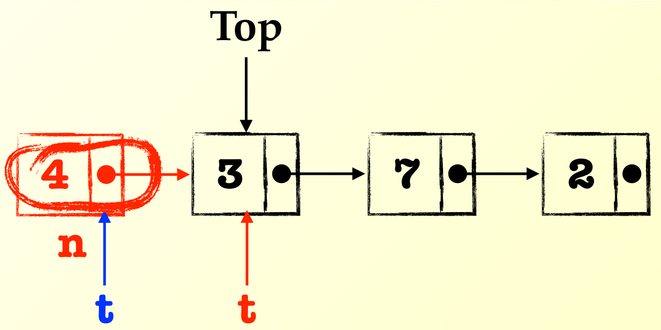
```
push(k):
Node t
1  n = new Node(k,-)
2  while (true)
3    t = Top
4    n.next = t
5    if (CAS (Top,t,n))
6      exit
```

push(4)

pop

```
pop:
Node t
1  while (true)
2    t = Top
3    if (t = NULL)
4      return *
5      exit
6    if (CAS (Top,t,t.next))
7      return t.val
8      exit
```





The Treiber Stack Algorithm

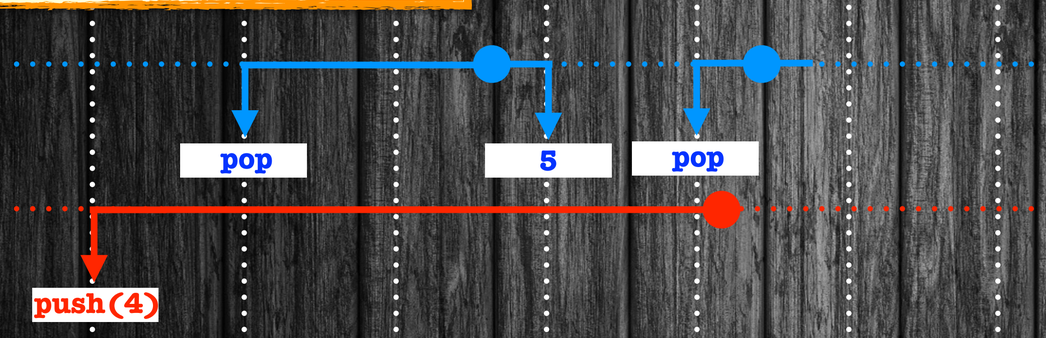
Linearization Policy

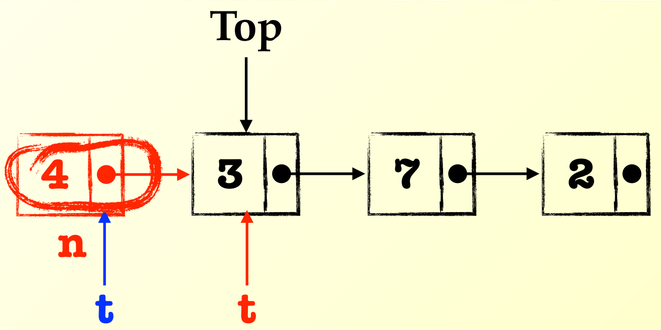
```
push(k):
Node t
1  n = new Node(k,-)
2  while (true)
3    t = Top
4    n.next = t
5    if (CAS (Top,t,n))
6      exit
```

push(4)

pop

```
pop:
Node t
1  while (true)
2    t = Top
3    if (t = NULL)
4      return *
5      exit
6    if (CAS (Top,t,t.next))
7      return t.val
8      exit
```





The Treiber Stack Algorithm

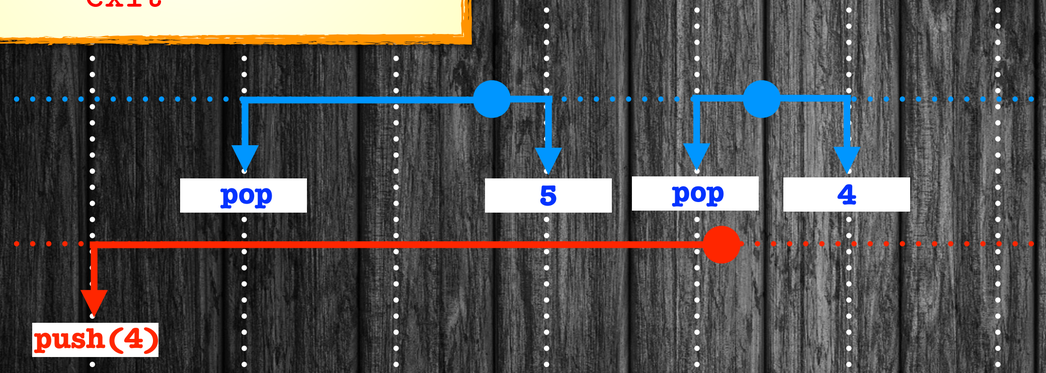
Linearization Policy

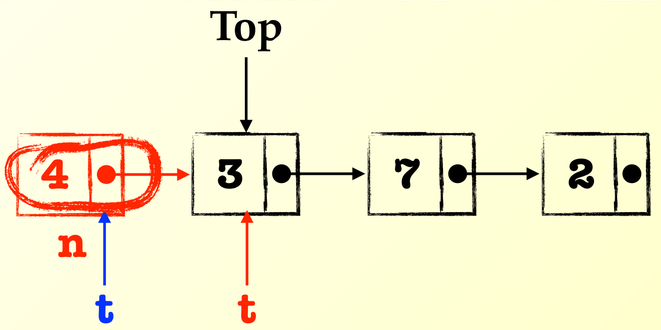
```
push(k):
Node t
1  n = new Node(k,-)
2  while (true)
3    t = Top
4    n.next = t
5    if (CAS (Top,t,n))
6      exit
```

push(4)

pop

```
pop:
Node t
1  while (true)
2    t = Top
3    if (t = NULL)
4      return *
5      exit
6    if (CAS (Top,t,t.next))
7      return t.val
8      exit
```



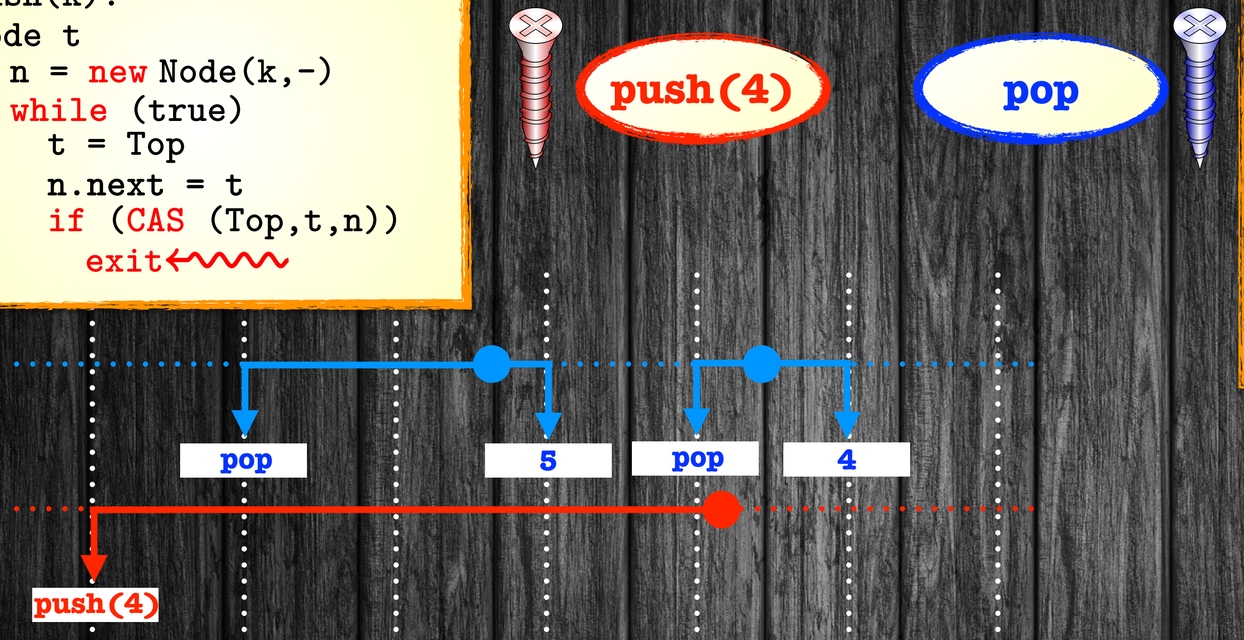


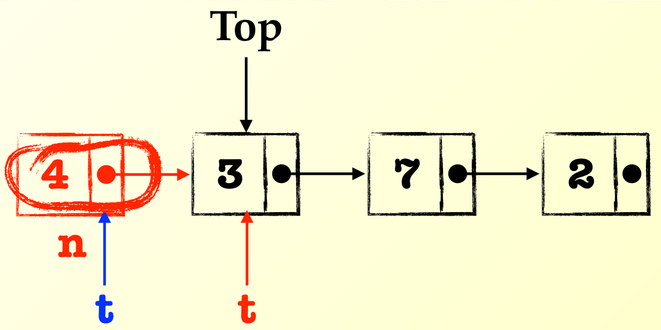
The Treiber Stack Algorithm

Linearization Policy

```
push(k):
Node t
1  n = new Node(k,-)
2  while (true)
3    t = Top
4    n.next = t
5    if (CAS (Top,t,n))
6      exit
```

```
pop:
Node t
1  while (true)
2    t = Top
3    if (t = NULL)
4      return *
5      exit
6    if (CAS (Top,t,t.next))
7      return t.val
8      exit
```



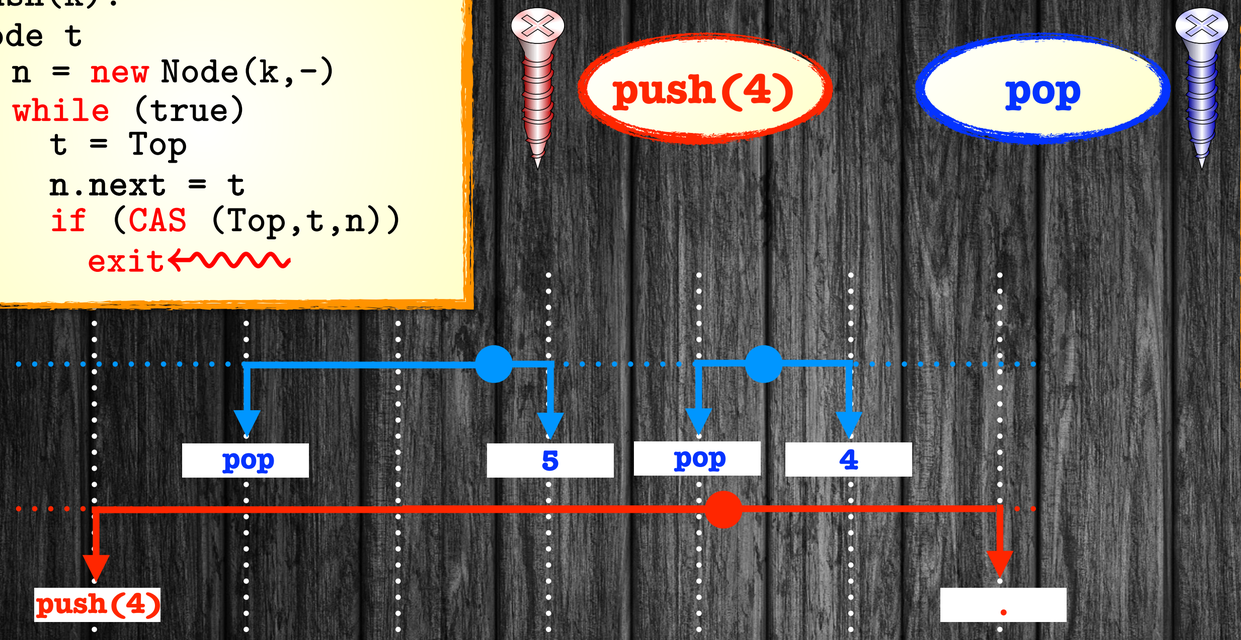


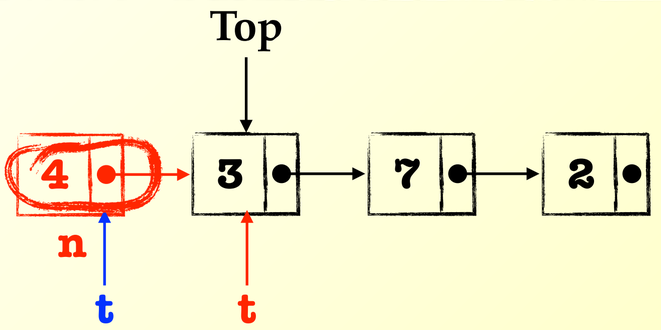
The Treiber Stack Algorithm

Linearization Policy

```
push(k):
Node t
1  n = new Node(k,-)
2  while (true)
3    t = Top
4    n.next = t
5    if (CAS (Top,t,n))
6      exit
```

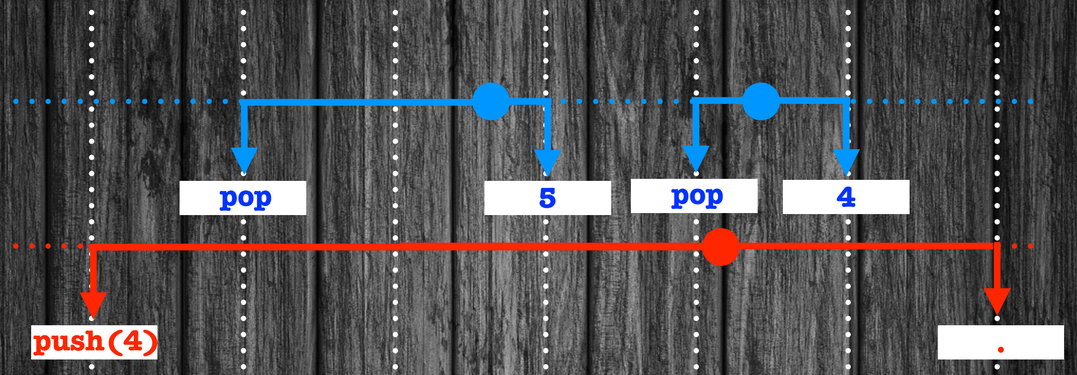
```
pop:
Node t
1  while (true)
2    t = Top
3    if (t = NULL)
4      return *
5      exit
6    if (CAS (Top,t,t.next))
7      return t.val
8      exit
```





The Treiber Stack Algorithm

Linearization Policy



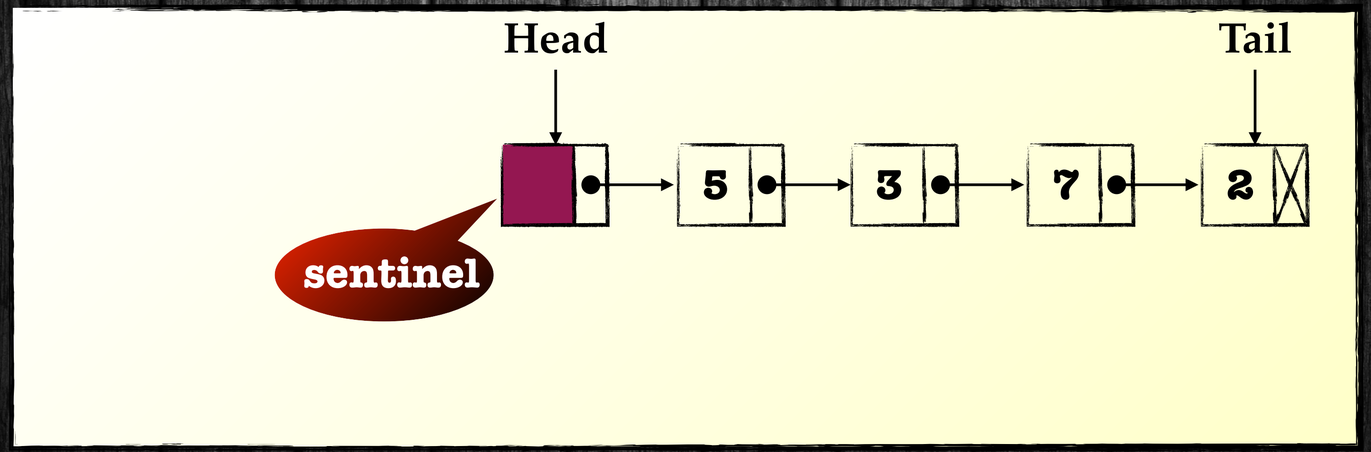
Outline

- Overview
- Abstract Data Types
- Sequential Programs
- Concurrent Programs
- Coarse-Grained Synchronization
- Fine-Grained Synchronization
- Optimistic Synchronization
- Lazy Synchronization
- Treiber Stack Algorithm
- Michael&Scott Queue Algorithm

Outline

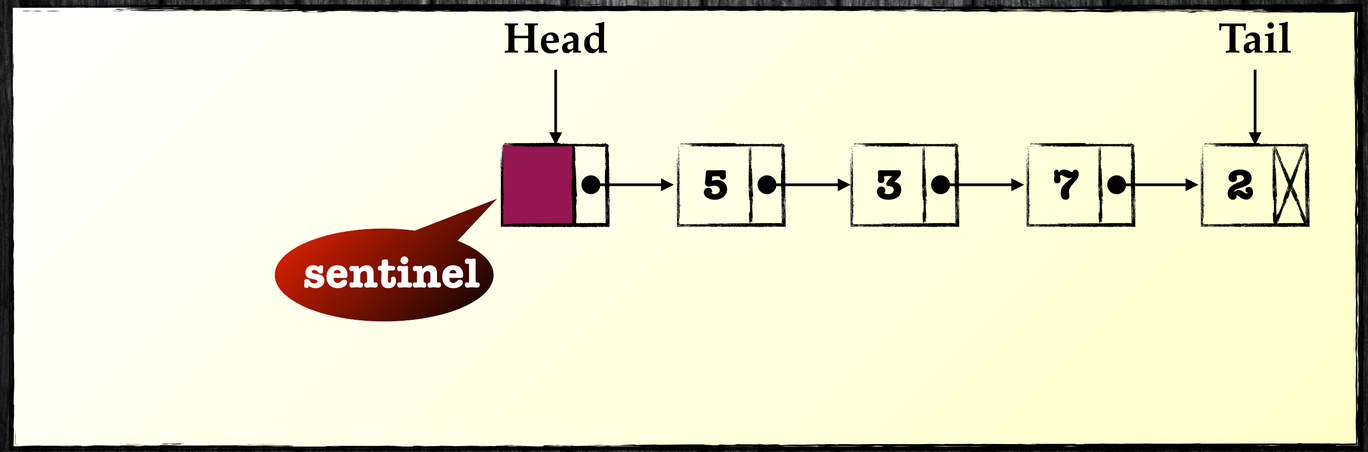
- Overview
- Abstract Data Types
- Sequential Programs
- Concurrent Programs
- Coarse-Grained Synchronization
- Fine-Grained Synchronization
- Optimistic Synchronization
- Lazy Synchronization
- Treiber Stack Algorithm
- Michael&Scott Queue Algorithm

The Michael-Scott Queue Algorithm



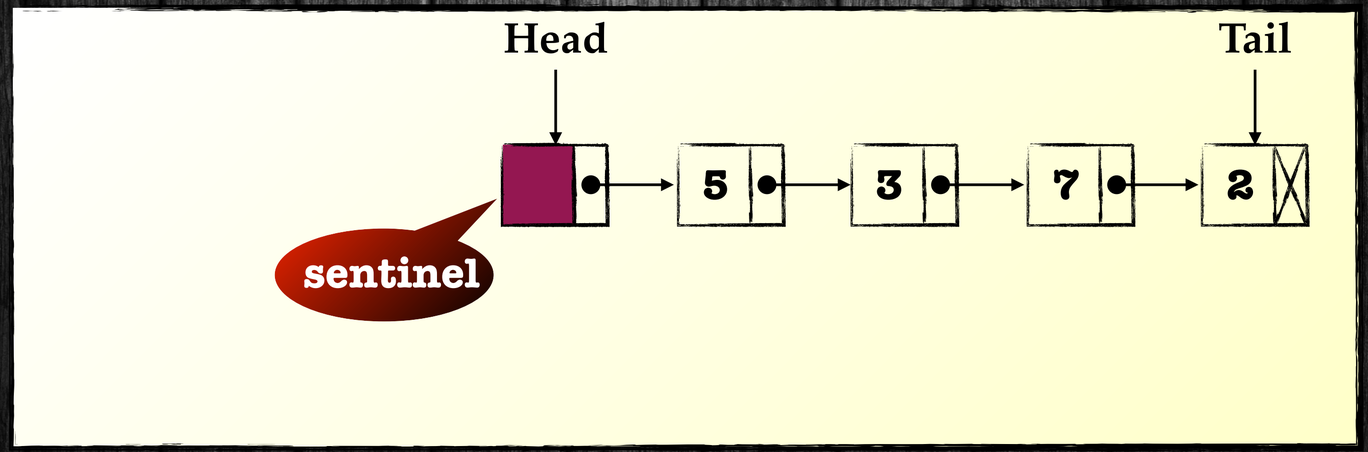
The Michael-Scott Queue Algorithm





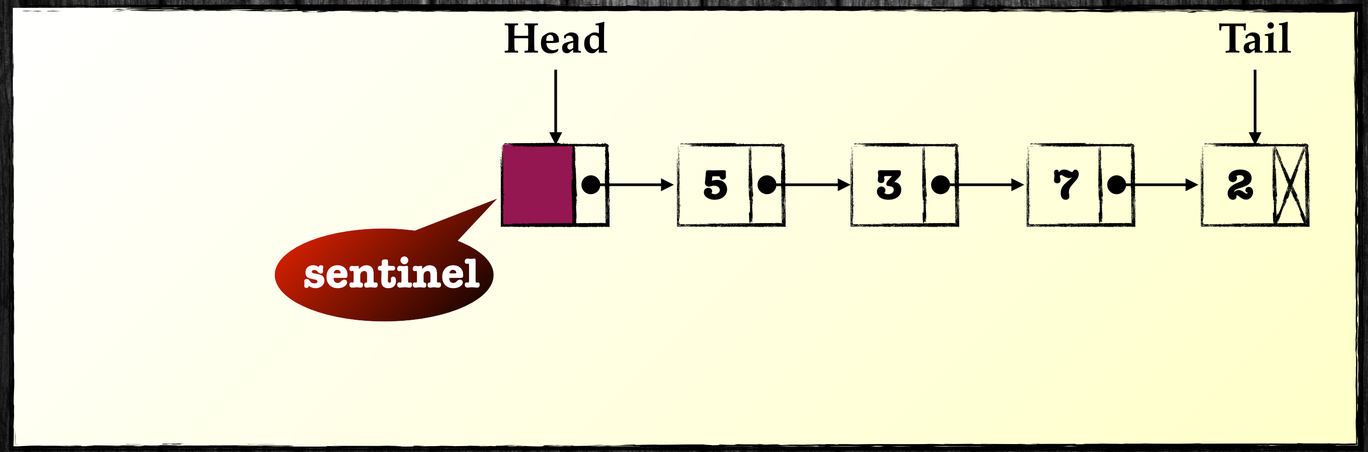
The Michael-Scott Queue Algorithm



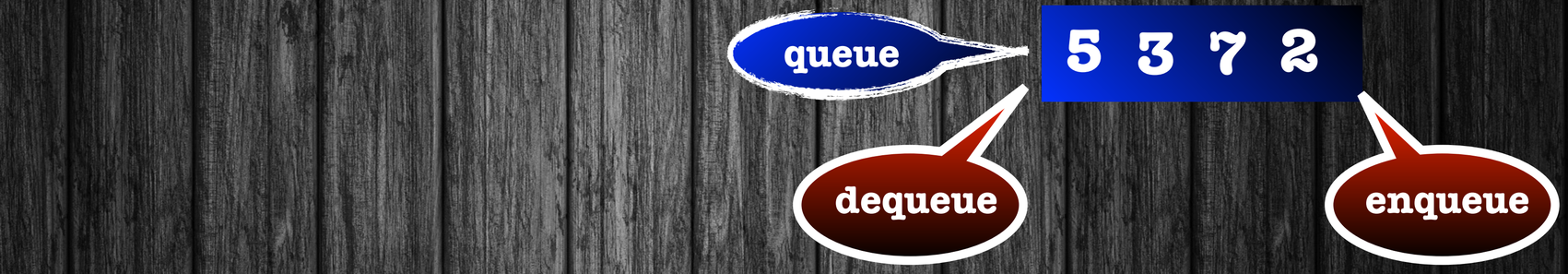


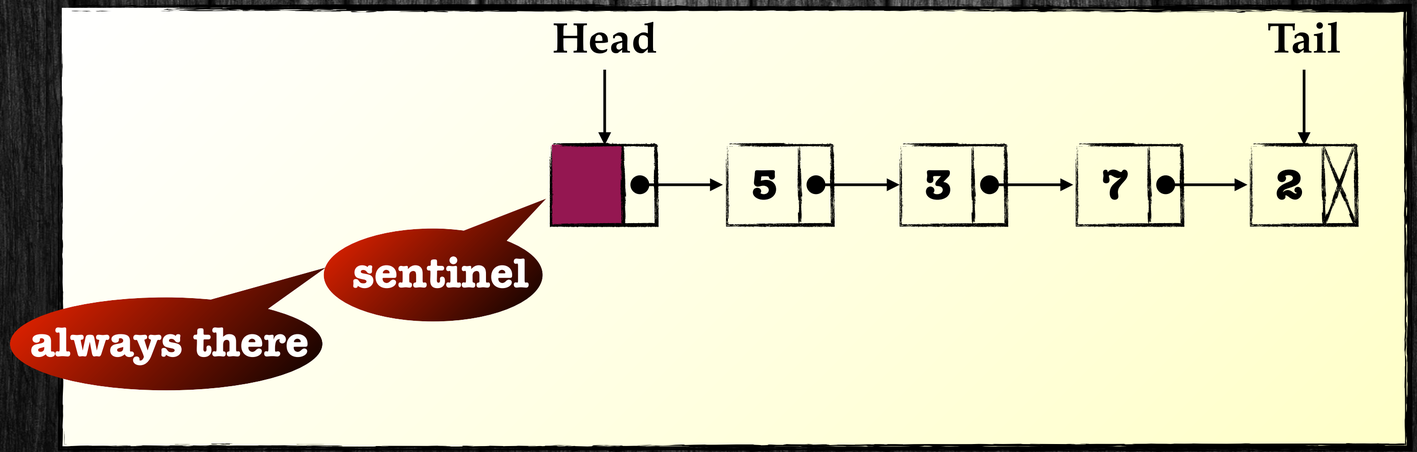
The Michael-Scott Queue Algorithm





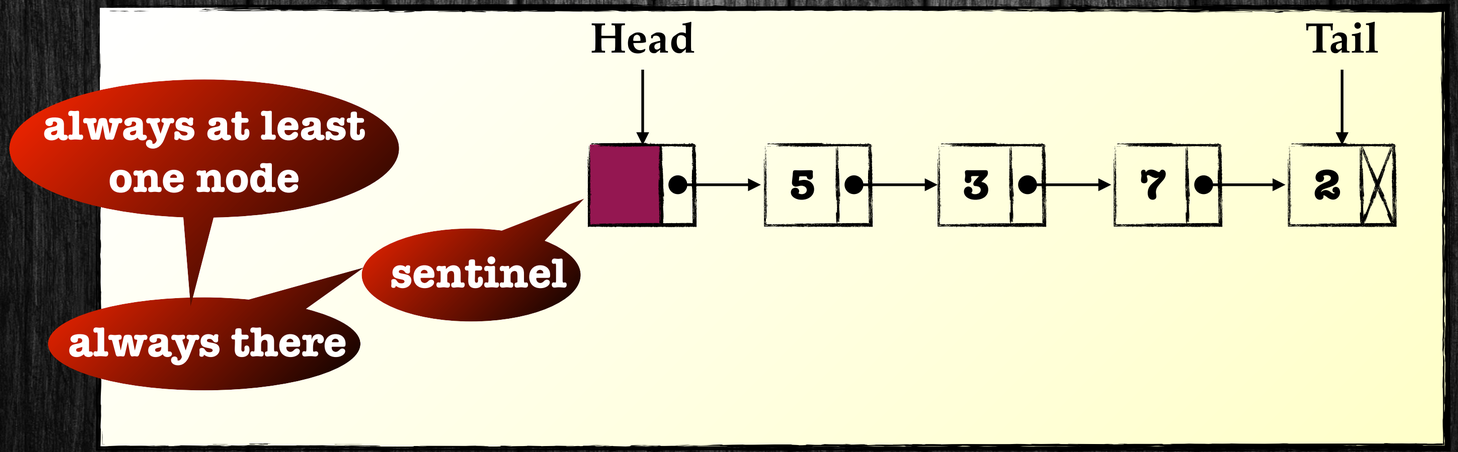
The Michael-Scott Queue Algorithm



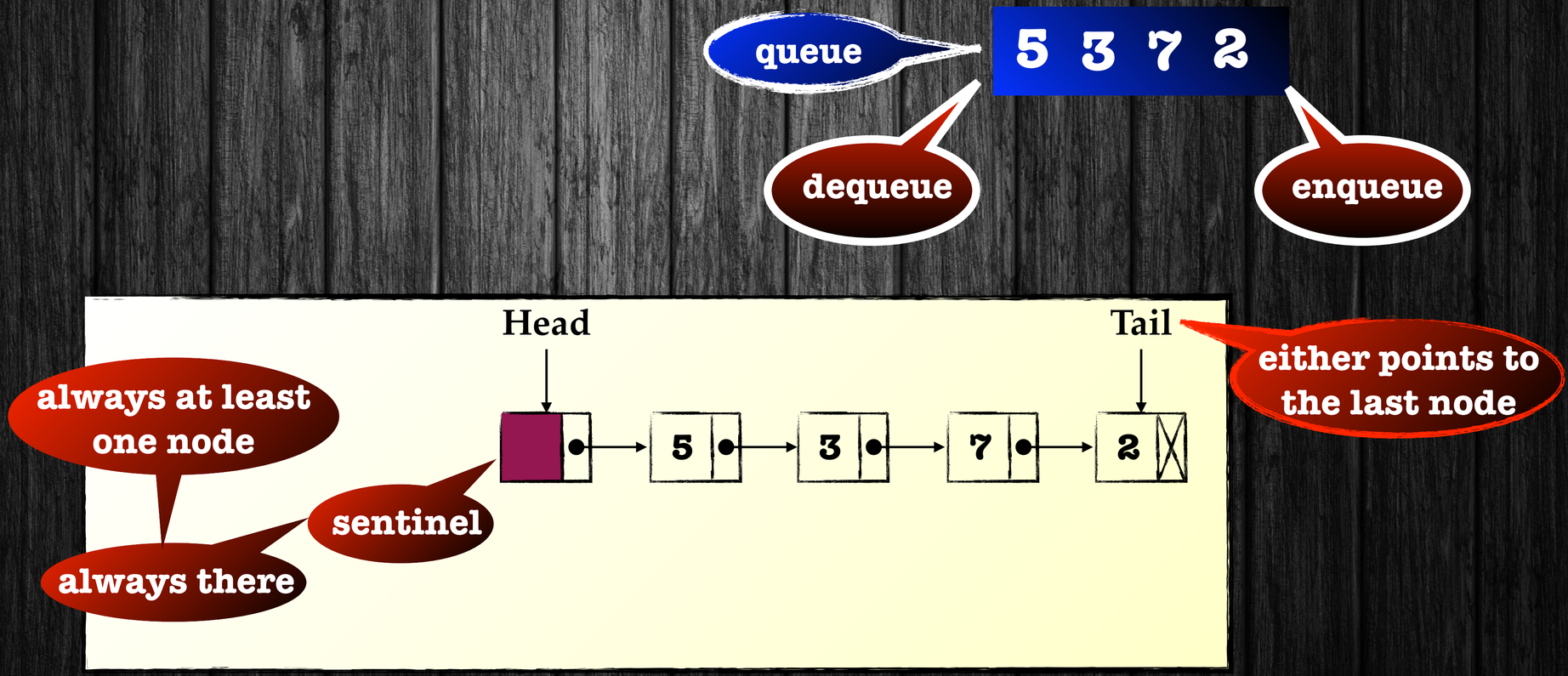


The Michael-Scott Queue Algorithm

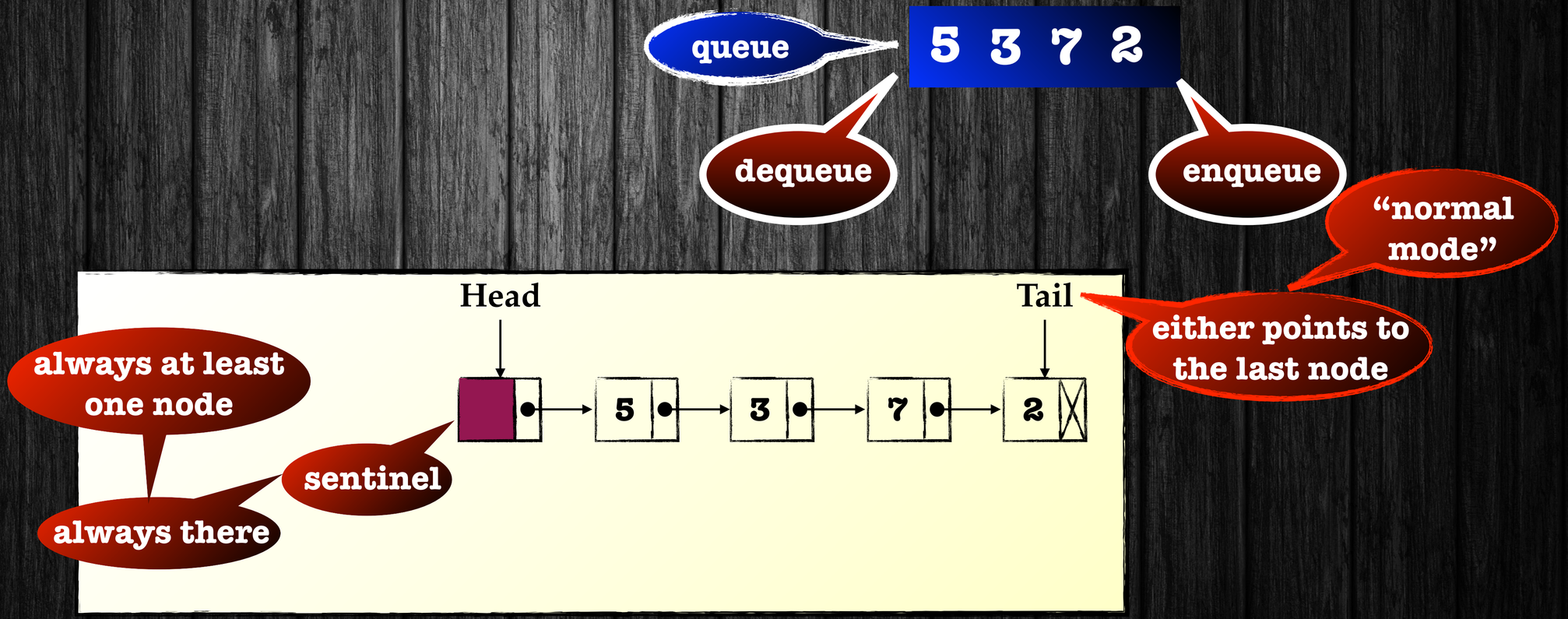




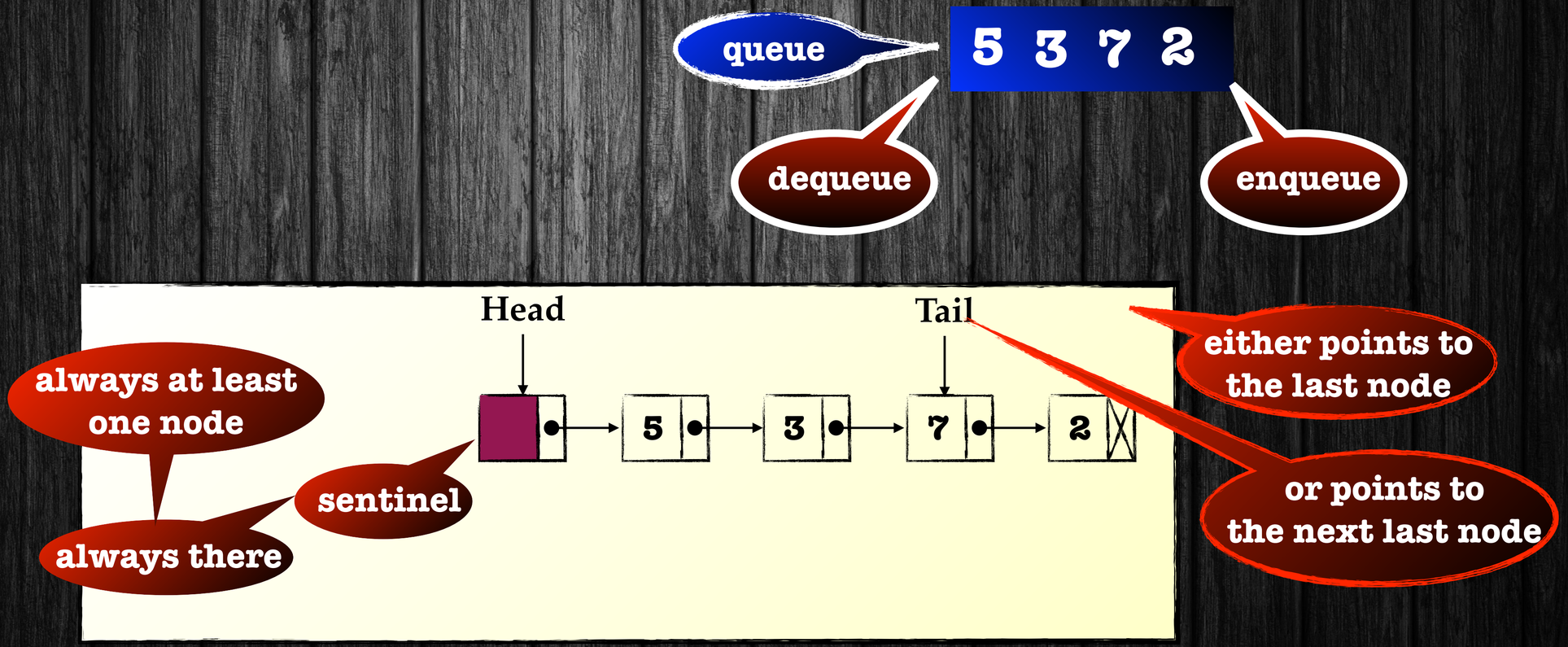
The Michael-Scott Queue Algorithm



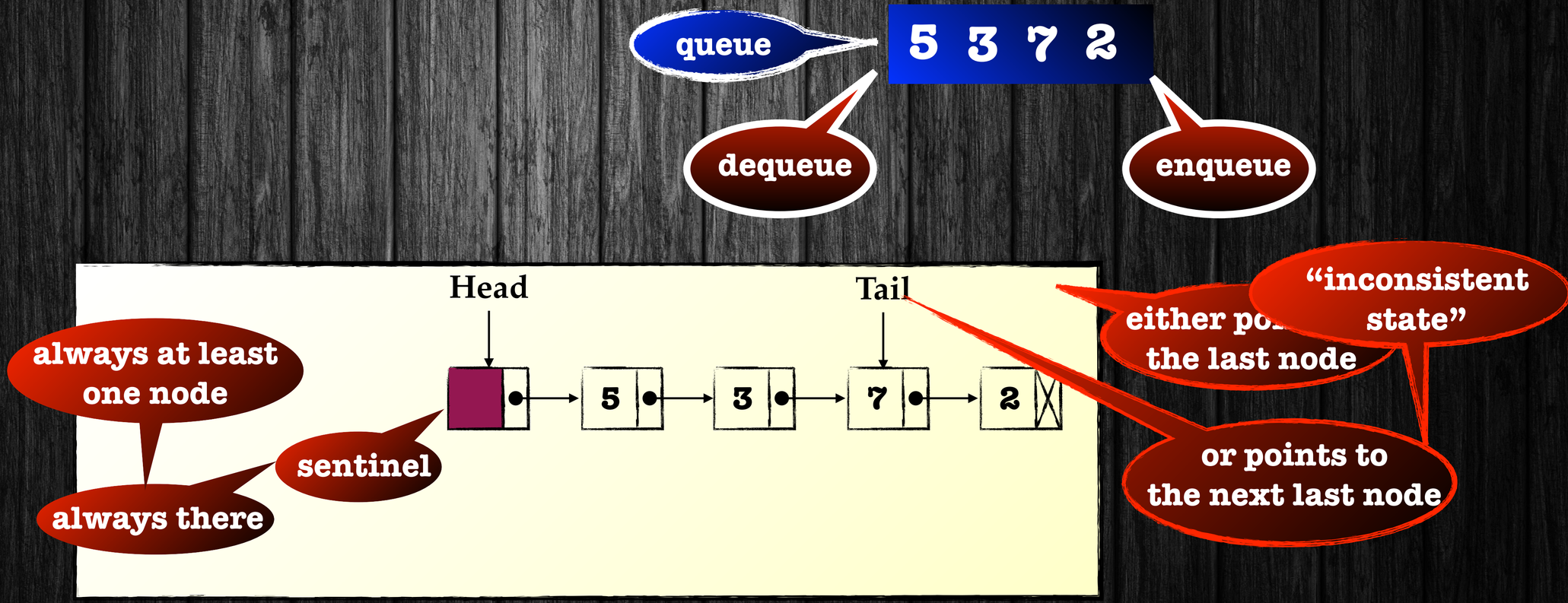
The Michael-Scott Queue Algorithm



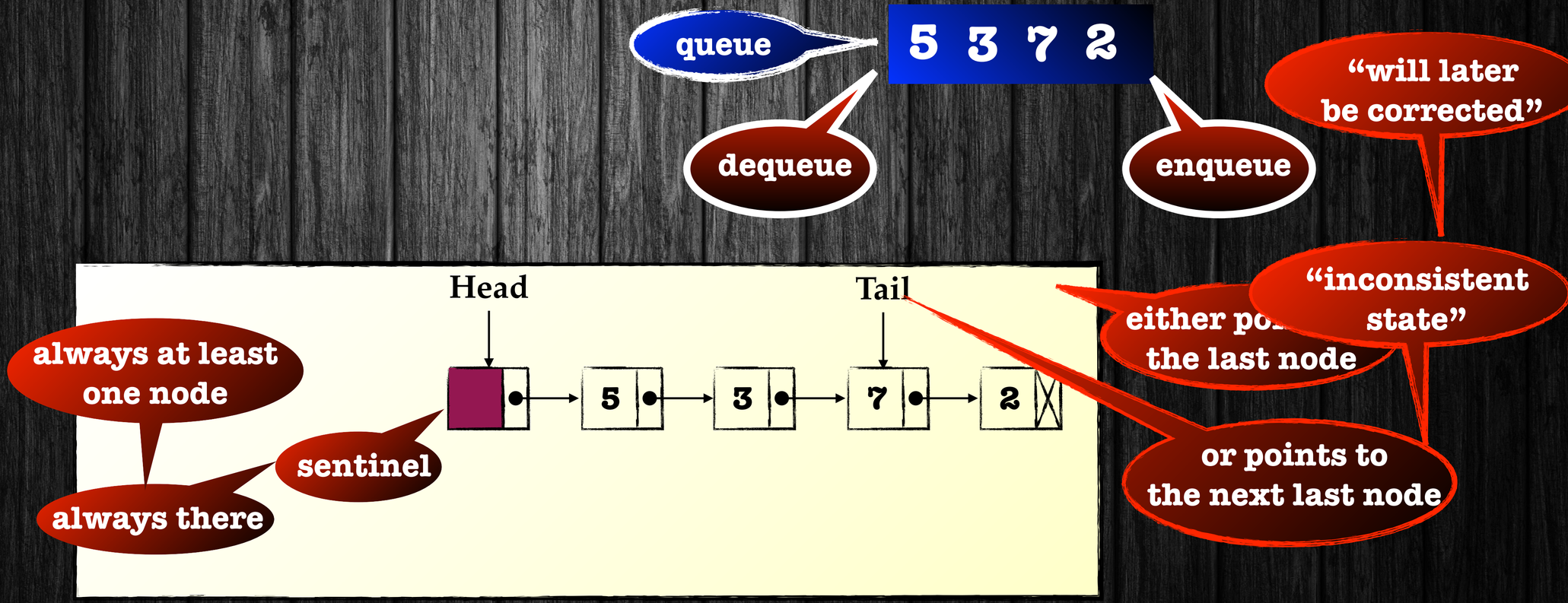
The Michael-Scott Queue Algorithm



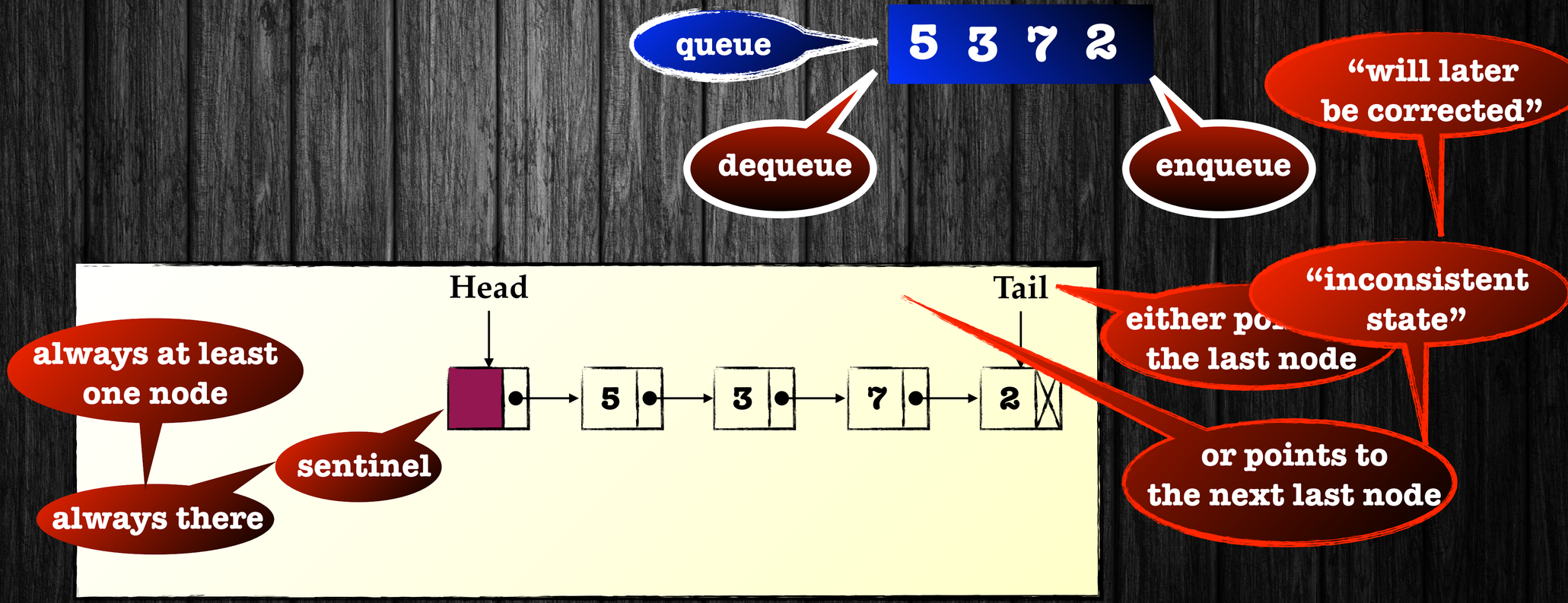
The Michael-Scott Queue Algorithm



The Michael-Scott Queue Algorithm



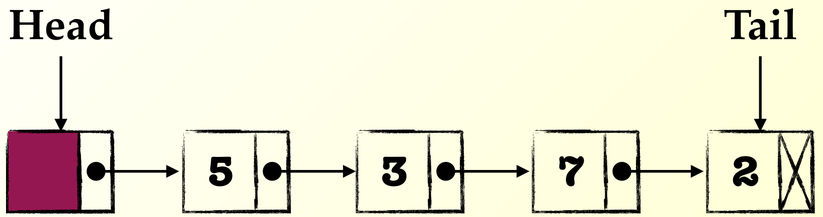
The Michael-Scott Queue Algorithm



The Michael-Scott Queue Algorithm

```
enq(k):
Node t,x
 1  n = new Node(k,NULL)
 2  while (true)
 3    t = Tail
 4    x = t.next
 5    if (t == Tail)
 6      if (x == NULL)
 7        if (CAS (t.next,NULL,n))
 8          CAS (Tail,t,n)
 9          exit
10      else
11        CAS (Tail,t,x)
```





The Michael-Scott Queue Algorithm

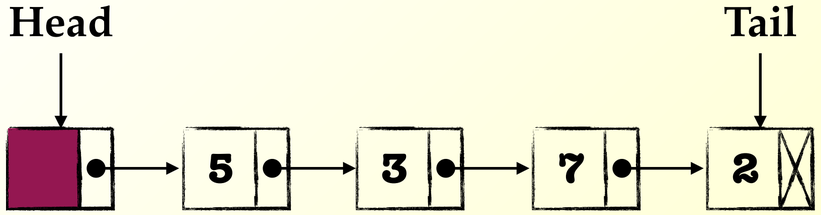
```
enq(k):
Node t,x
 1  n = new Node(k,NULL)
 2  while (true)
 3    t = Tail
 4    x = t.next
 5    if (t == Tail)
 6      if (x == NULL)
 7        if (CAS (t.next,NULL,n))
 8          CAS (Tail,t,n)
 9          exit
10      else
11        CAS (Tail,t,x)
```



enq(4)

queue

5 3 7 2



The Michael-Scott Queue Algorithm

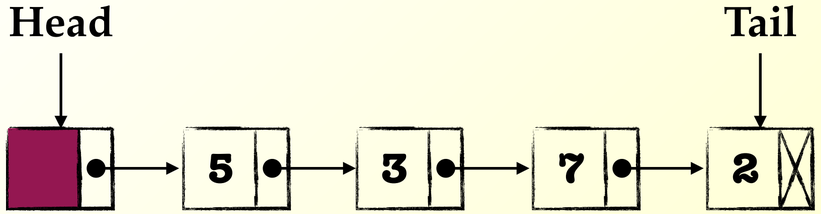
```
enq(k):
Node t,x
 1  n = new Node(k,NULL)
 2  while (true)
 3    t = Tail
 4    x = t.next
 5    if (t == Tail)
 6      if (x == NULL)
 7        if (CAS (t.next,NULL,n))
 8          CAS (Tail,t,n)
 9          exit
10      else
11        CAS (Tail,t,x)
```



enq(4)

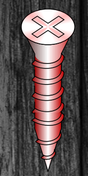
queue

5 3 7 2



The Michael-Scott Queue Algorithm

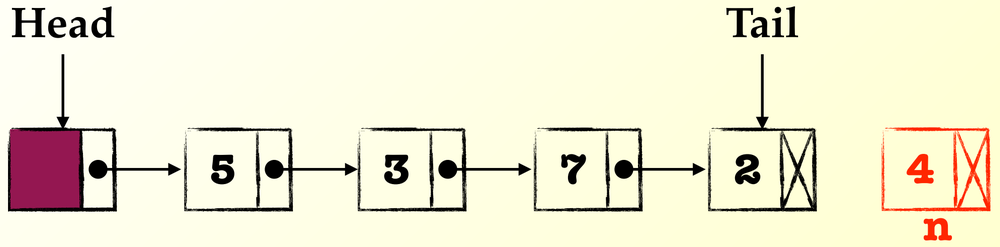
```
enq(k):
Node t,x
1  n = new Node(k,NULL)
2  while (true)
3    t = Tail
4    x = t.next
5    if (t == Tail)
6      if (x == NULL)
7        if (CAS (t.next,NULL,n))
8          CAS (Tail,t,n)
9          exit
10     else
11       CAS (Tail,t,x)
```



enq(4)

queue

5 3 7 2



The Michael-Scott Queue Algorithm

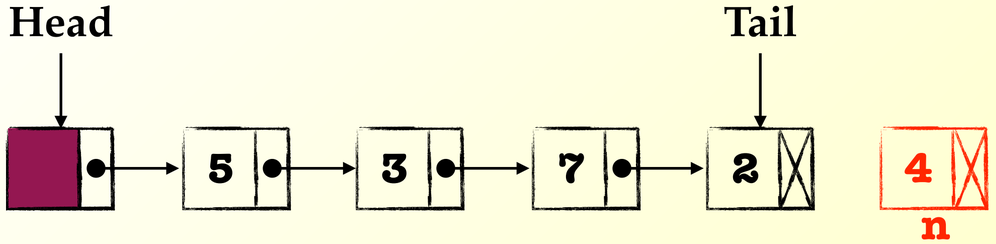
```
enq(k):
Node t,x
 1  n = new Node(k,NULL)
 2  while (true)
 3    t = Tail
 4    x = t.next
 5    if (t == Tail)
 6      if (x == NULL)
 7        if (CAS (t.next,NULL,n))
 8          CAS (Tail,t,n)
 9          exit
10      else
11        CAS (Tail,t,x)
```



enq(4)

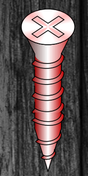
queue

5 3 7 2



The Michael-Scott Queue Algorithm

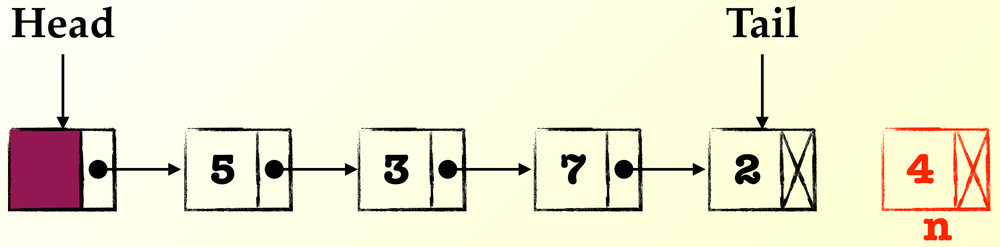
```
enq(k):
Node t,x
1  n = new Node(k,NULL)
2  while (true)
3    t = Tail
4    x = t.next
5    if (t == Tail)
6      if (x == NULL)
7        if (CAS (t.next,NULL,n))
8          CAS (Tail,t,n)
9          exit
10     else
11       CAS (Tail,t,x)
```



enq(4)

queue

5 3 7 2



The Michael-Scott Queue Algorithm

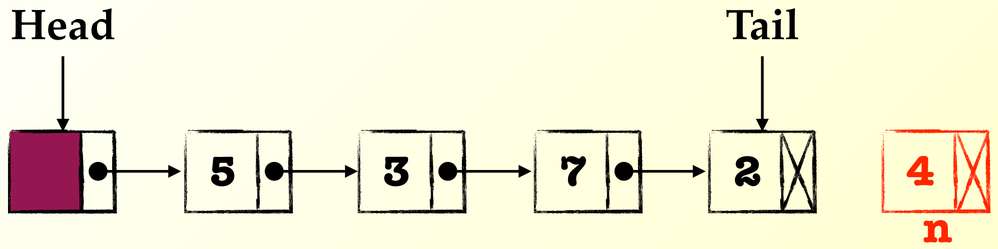
```
enq(k):
Node t,x
1  n = new Node(k,NULL)
2  while (true)
3    t = Tail
4    x = t.next
5    if (t == Tail)
6      if (x == NULL)
7        if (CAS (t.next,NULL,n))
8          CAS (Tail,t,n)
9          exit
10     else
11       CAS (Tail,t,x)
```

try to add the new node

enq(4)

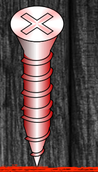
queue

5 3 7 2



The Michael-Scott Queue Algorithm

```
enq(k):
Node t,x
1  n = new Node(k,NULL)
2  while (true)
3    t = Tail
4    x = t.next
5    if (t == Tail)
6      if (x == NULL)
7        if (CAS (t.next,NULL,n))
8          CAS (Tail,t,n)
9          exit
10     else
11       CAS (Tail,t,x)
```



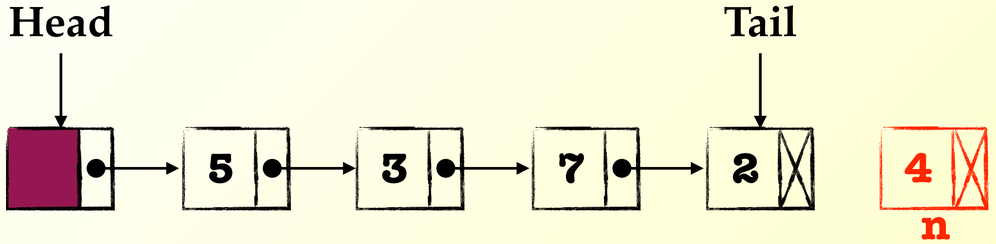
enq(4)

try to add the new node

you may repeatedly fail

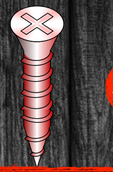
queue

5 3 7 2



The Michael-Scott Queue Algorithm

```
enq(k):
Node t,x
1  n = new Node(k,NULL)
2  while (true)
3    t = Tail
4    x = t.next
5    if (t == Tail)
6      if (x == NULL)
7        if (CAS (t.next,NULL,n))
8          CAS (Tail,t,n)
9          exit
10     else
11       CAS (Tail,t,x)
```



enq(4)

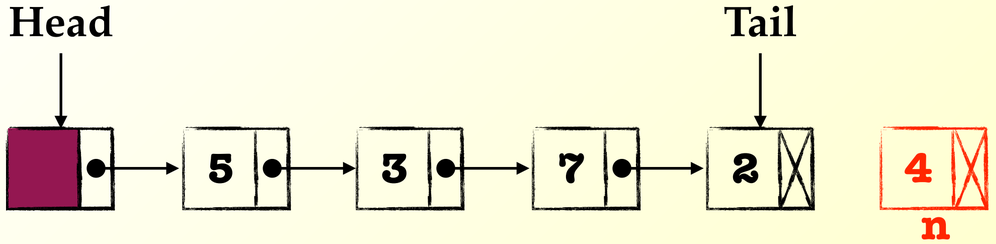
try to add the new node

you may repeatedly fail

"not wait-free"

queue

5 3 7 2



The Michael-Scott Queue Algorithm

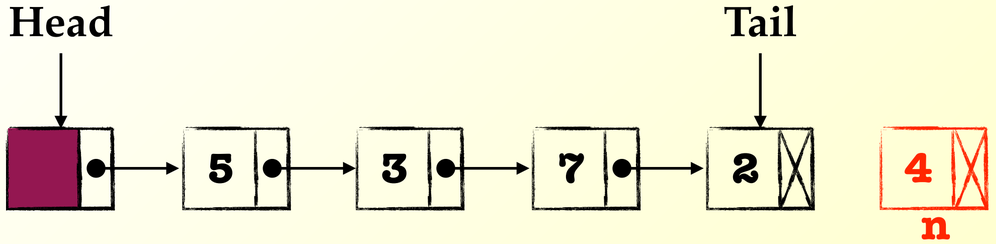
```
enq(k):
Node t,x
 1  n = new Node(k,NULL)
 2  while (true)
 3    t = Tail
 4    x = t.next
 5    if (t == Tail)
 6      if (x == NULL)
 7        if (CAS (t.next,NULL,n))
 8          CAS (Tail,t,n)
 9          exit
10      else
11        CAS (Tail,t,x)
```



enq(4)

queue

5 3 7 2



The Michael-Scott Queue Algorithm

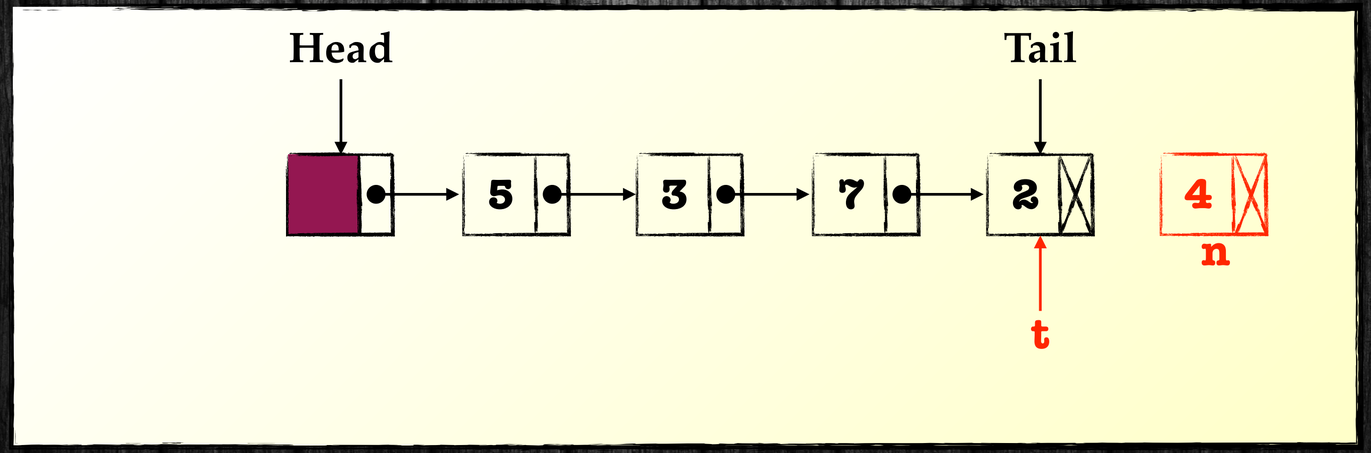
```
enq(k):
Node t,x
 1  n = new Node(k,NULL)
 2  while (true)
 3    t = Tail
 4    x = t.next
 5    if (t == Tail)
 6      if (x == NULL)
 7        if (CAS (t.next,NULL,n))
 8          CAS (Tail,t,n)
 9          exit
10      else
11        CAS (Tail,t,x)
```



enq(4)

queue

5 3 7 2



The Michael-Scott Queue Algorithm

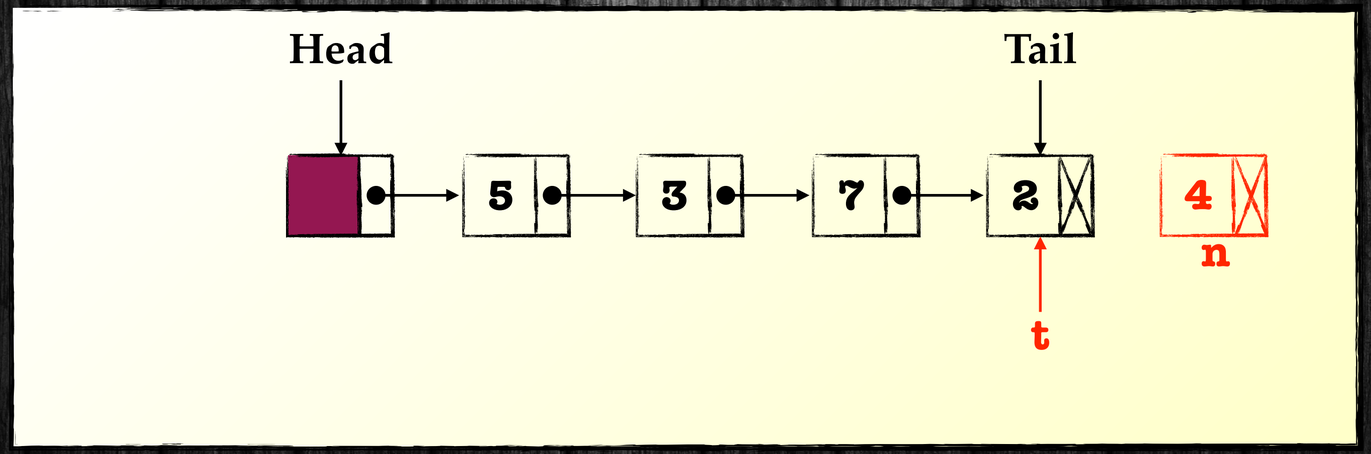
```
enq(k):
Node t,x
 1  n = new Node(k,NULL)
 2  while (true)
 3    t = Tail
 4    x = t.next
 5    if (t == Tail)
 6      if (x == NULL)
 7        if (CAS (t.next,NULL,n))
 8          CAS (Tail,t,n)
 9          exit
10      else
11        CAS (Tail,t,x)
```



enq(4)

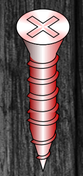
queue

5 3 7 2



The Michael-Scott Queue Algorithm

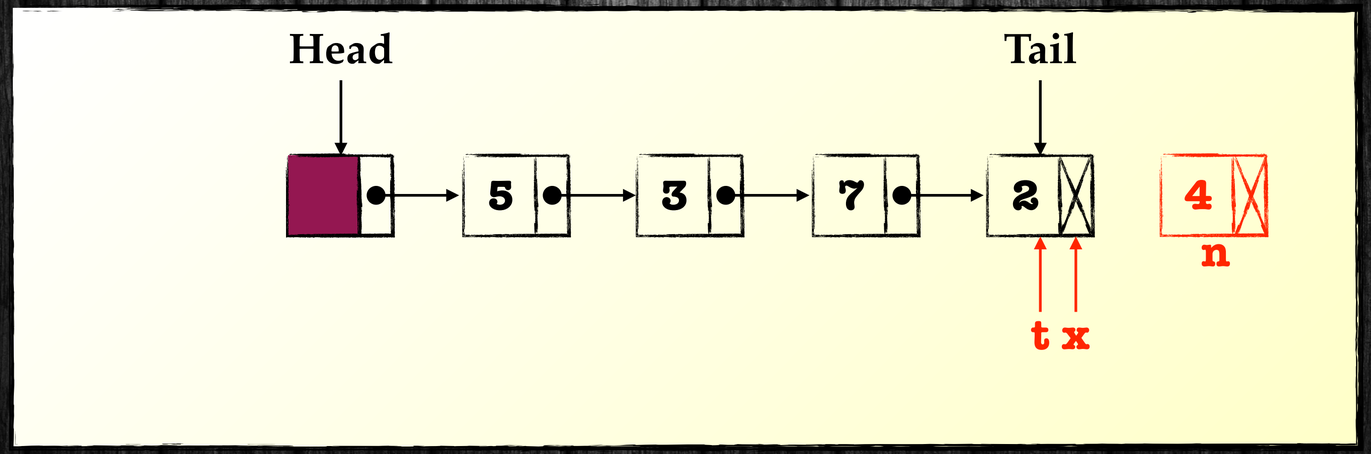
```
enq(k):
Node t,x
 1  n = new Node(k,NULL)
 2  while (true)
 3    t = Tail
 4    x = t.next
 5    if (t == Tail)
 6      if (x == NULL)
 7        if (CAS (t.next,NULL,n))
 8          CAS (Tail,t,n)
 9          exit
10      else
11        CAS (Tail,t,x)
```



enq(4)

queue

5 3 7 2



The Michael-Scott Queue Algorithm

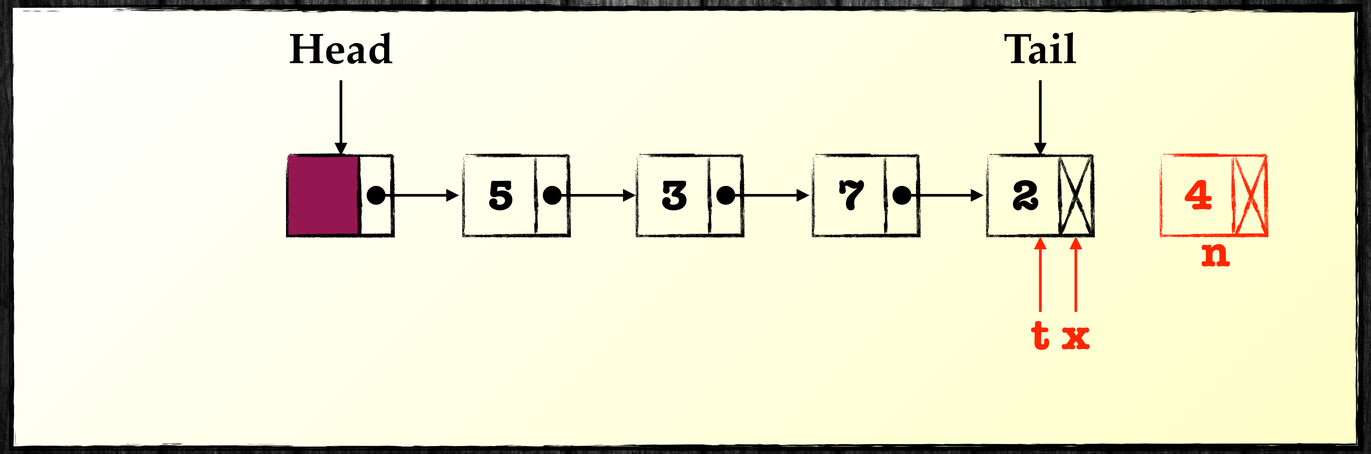
```
enq(k):
Node t,x
 1  n = new Node(k,NULL)
 2  while (true)
 3    t = Tail
 4    x = t.next
 5    if (t == Tail)
 6      if (x == NULL)
 7        if (CAS (t.next,NULL,n))
 8          CAS (Tail,t,n)
 9          exit
10      else
11        CAS (Tail,t,x)
```



enq(4)

queue

5 3 7 2



The Michael-Scott Queue Algorithm

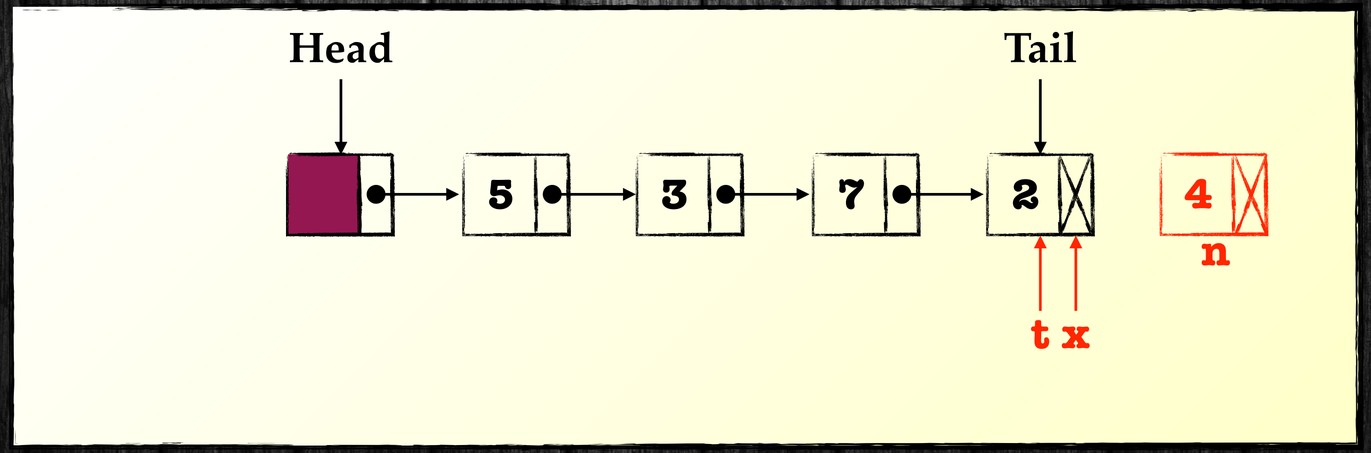
```
enq(k):
Node t,x
 1  n = new Node(k,NULL)
 2  while (true)
 3    t = Tail
 4    x = t.next
 5    if (t == Tail)
 6      if (x == NULL)
 7        if (CAS (t.next,NULL,n))
 8          CAS (Tail,t,n)
 9          exit
10      else
11        CAS (Tail,t,x)
```



enq(4)

queue

5 3 7 2



The Michael-Scott Queue Algorithm

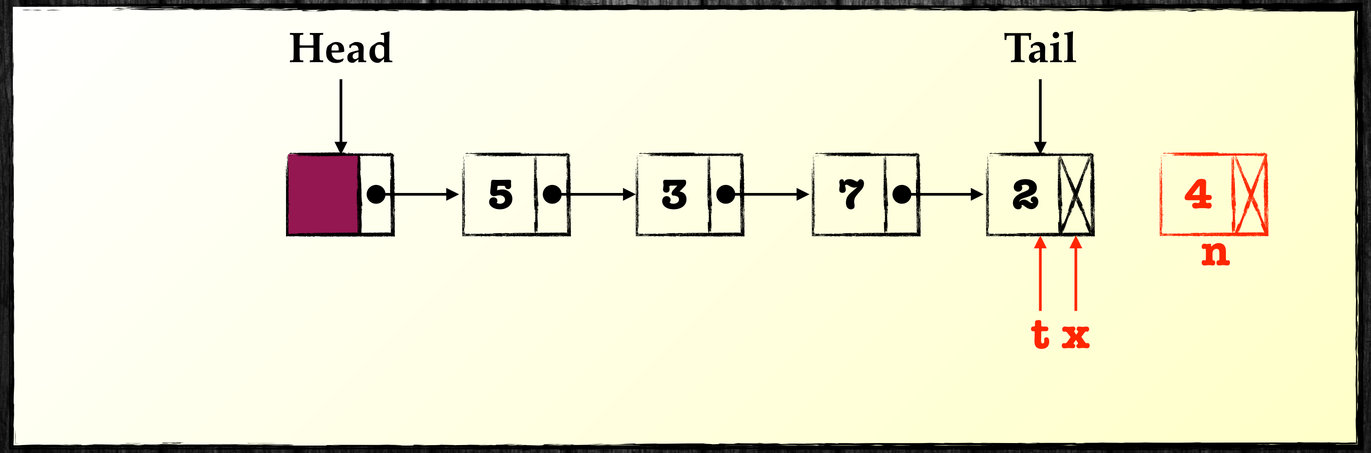
```
enq(k):
Node t,x
1  n = new Node(k,NULL)
2  while (true)
3    t = Tail
4    x = t.next
5    if (t == Tail)
6      if (x == NULL)
7        if (CAS (t.next,NULL,n)
8          CAS (Tail,t,n)
9          exit
10     else
11       CAS (Tail,t,x)
```



enq(4)

"is Tail really pointing to the last node?"

queue 5 3 7 2



The Michael-Scott Queue Algorithm

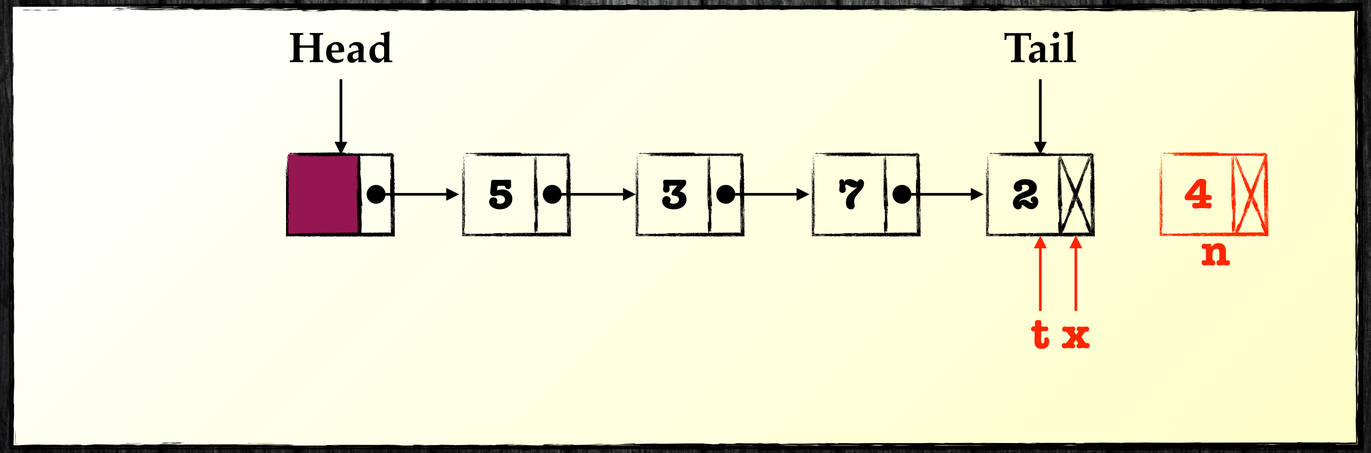
```
enq(k):
Node t,x
 1  n = new Node(k,NULL)
 2  while (true)
 3    t = Tail
 4    x = t.next
 5    if (t == Tail)
 6      if (x == NULL)
 7        if (CAS (t.next,NULL,n))
 8          CAS (Tail,t,n)
 9          exit
10      else
11        CAS (Tail,t,x)
```



enq(4)

queue

5 3 7 2



The Michael-Scott Queue Algorithm

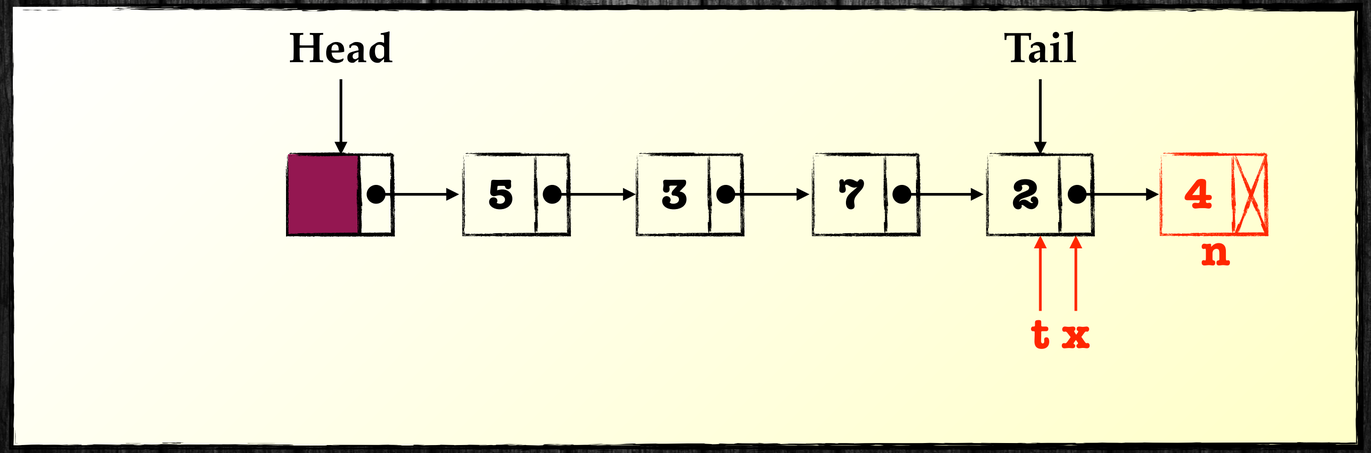
```
enq(k):
Node t,x
 1  n = new Node(k,NULL)
 2  while (true)
 3    t = Tail
 4    x = t.next
 5    if (t == Tail)
 6      if (x == NULL)
 7        if (CAS (t.next,NULL,n))
 8          CAS (Tail,t,n)
 9          exit
10      else
11        CAS (Tail,t,x)
```



enq(4)

queue

5 3 7 2



The Michael-Scott Queue Algorithm

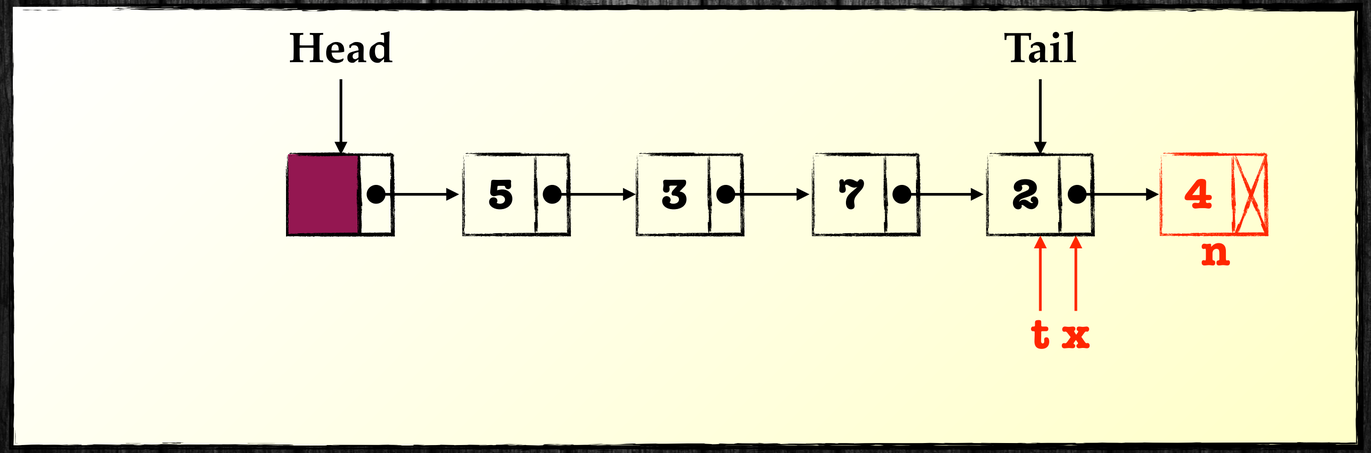
```
enq(k):
Node t,x
 1  n = new Node(k,NULL)
 2  while (true)
 3    t = Tail
 4    x = t.next
 5    if (t == Tail)
 6      if (x == NULL)
 7        if (CAS (t.next,NULL,n))
 8          CAS (Tail,t,n)
 9          exit
10      else
11        CAS (Tail,t,x)
```



enq(4)

queue

5 3 7 2



The Michael-Scott Queue Algorithm

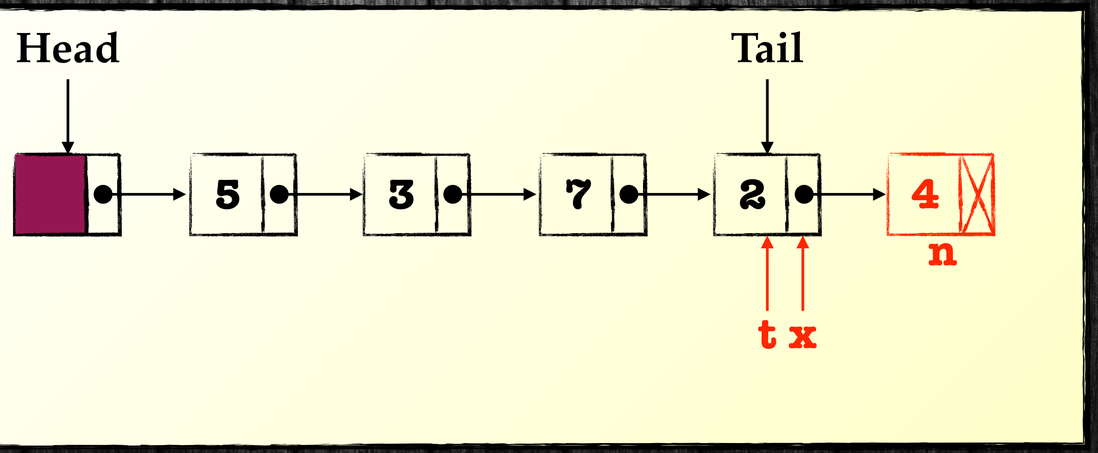
```
enq(k):
Node t,x
1  n = new Node(k,NULL)
2  while (true)
3    t = Tail
4    x = t.next
5    if (t == Tail)
6      if (x == NULL)
7        if (CAS (t.next,NULL,n))
8          CAS (Tail,t,n)
9          exit
10     else
11       CAS (Tail,t,x)
```



enq(4)

"Fix Tail if needed"

5 3 7 2



The Michael-Scott Queue Algorithm

```
enq(k):
Node t,x
1  n = new Node(k,NULL)
2  while (true)
3    t = Tail
4    x = t.next
5    if (t == Tail)
6      if (x == NULL)
7        if (CAS (t.next,NULL,n))
8          CAS (Tail,t,n)
9          exit
10     else
11       CAS (Tail,t,x)
```

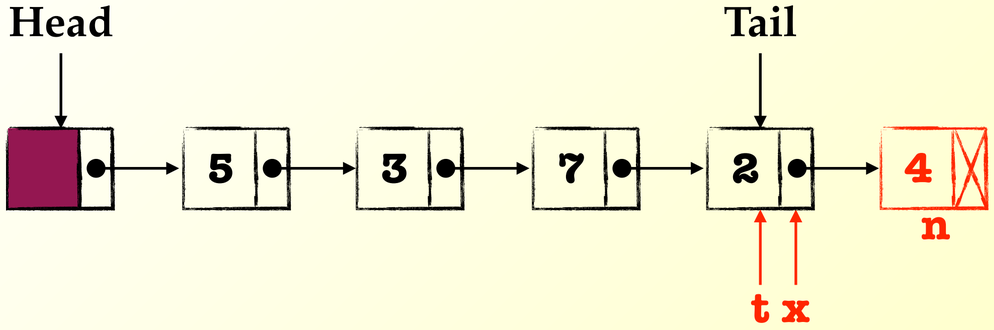


enq(4)

"Fix Tail if needed"

"Fix Tail if needed"

5 3 7 2



The Michael-Scott Queue Algorithm

```
enq(k):
Node t,x
1  n = new Node(k,NULL)
2  while (true)
3    t = Tail
4    x = t.next
5    if (t == Tail)
6      if (x == NULL)
7        if (CAS (t.next,NULL,n))
8          CAS (Tail,t,n)
9          exit
10     else
11       CAS (Tail,t,x)
```



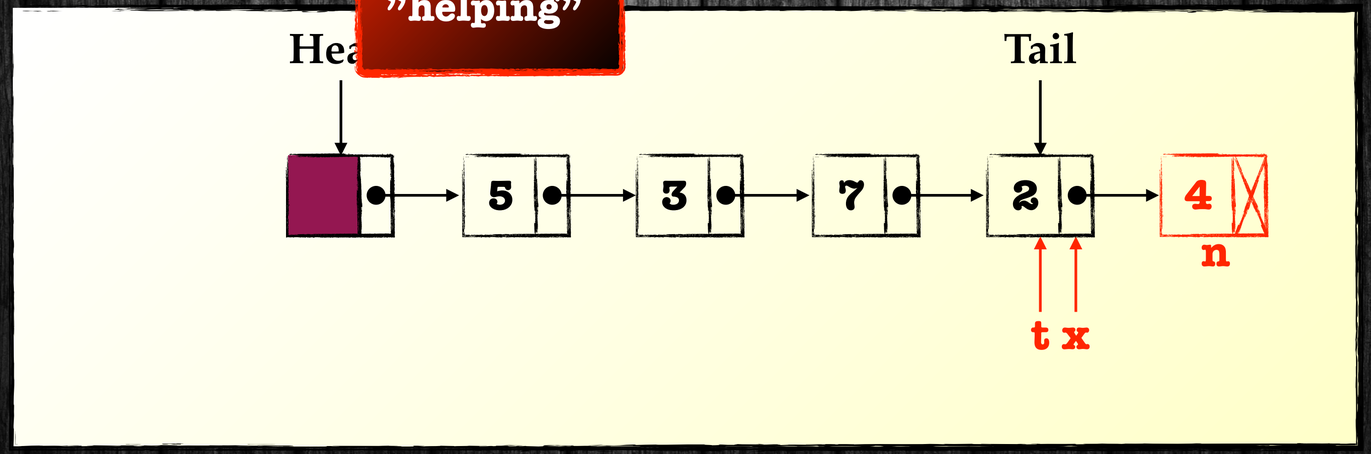
enq(4)

"Fix Tail if needed"

"Fix Tail if needed"

"helping"

5 3 7 2



The Michael-Scott Queue Algorithm

```
enq(k):
Node t,x
1  n = new Node(k,NULL)
2  while (true)
3    t = Tail
4    x = t.next
5    if (t == Tail)
6      if (x == NULL)
7        if (CAS (t.next,NULL,n))
8          CAS (Tail,t,n)
9          exit
10     else
11       CAS (Tail,t,x)
```



enq(4)

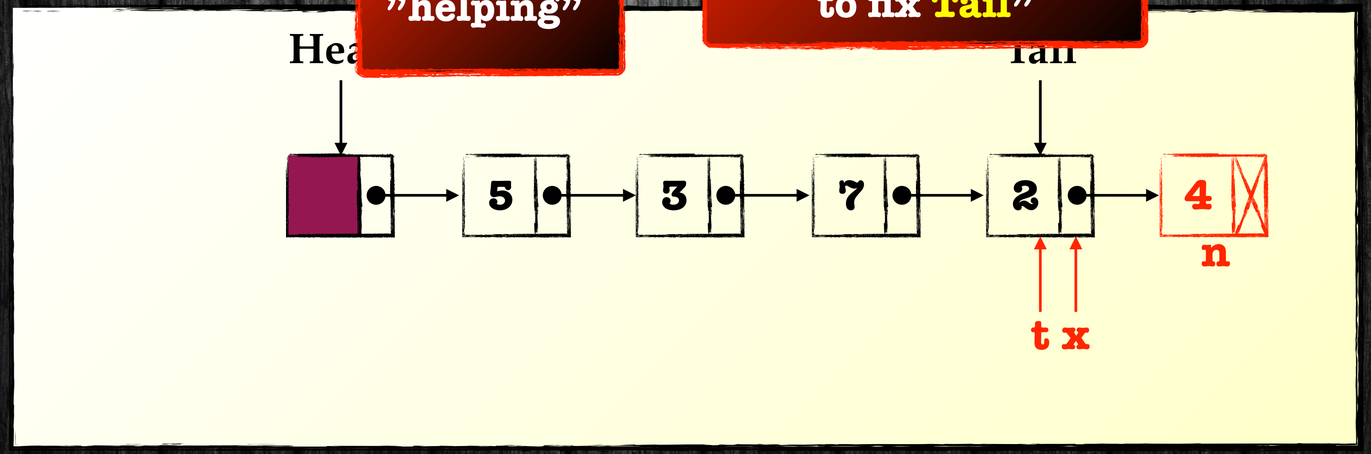
"Fix Tail if needed"

"Fix Tail if needed"

5 3 7 2

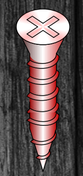
"helping"

"help another thread to fix Tail"



The Michael-Scott Queue Algorithm

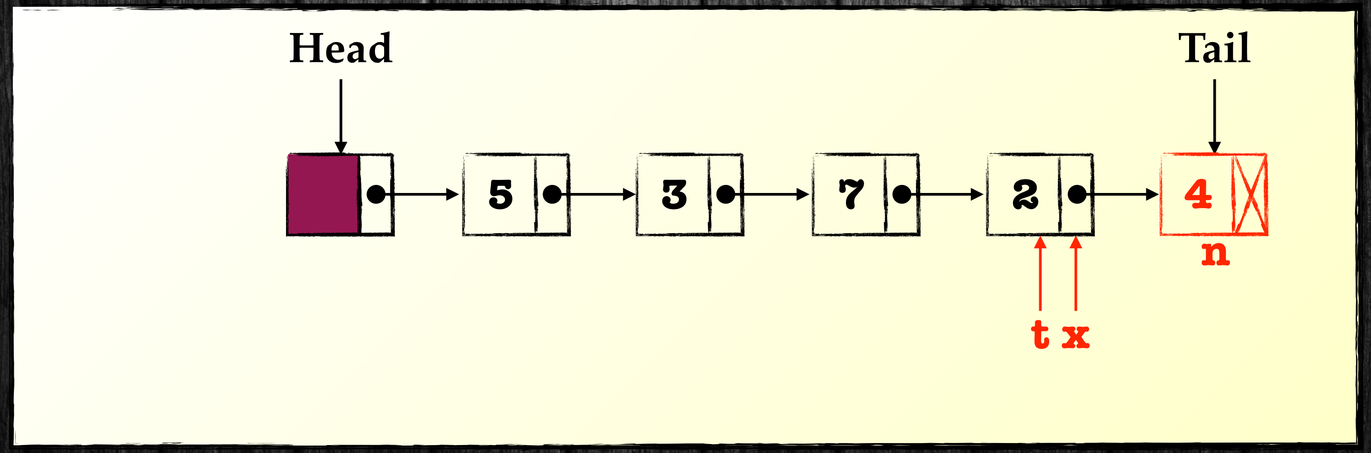
```
enq(k):
Node t,x
 1  n = new Node(k,NULL)
 2  while (true)
 3    t = Tail
 4    x = t.next
 5    if (t == Tail)
 6      if (x == NULL)
 7        if (CAS (t.next,NULL,n))
 8          CAS (Tail,t,n)
 9          exit
10      else
11        CAS (Tail,t,x)
```



enq(4)

queue

5 3 7 2



The Michael-Scott Queue Algorithm

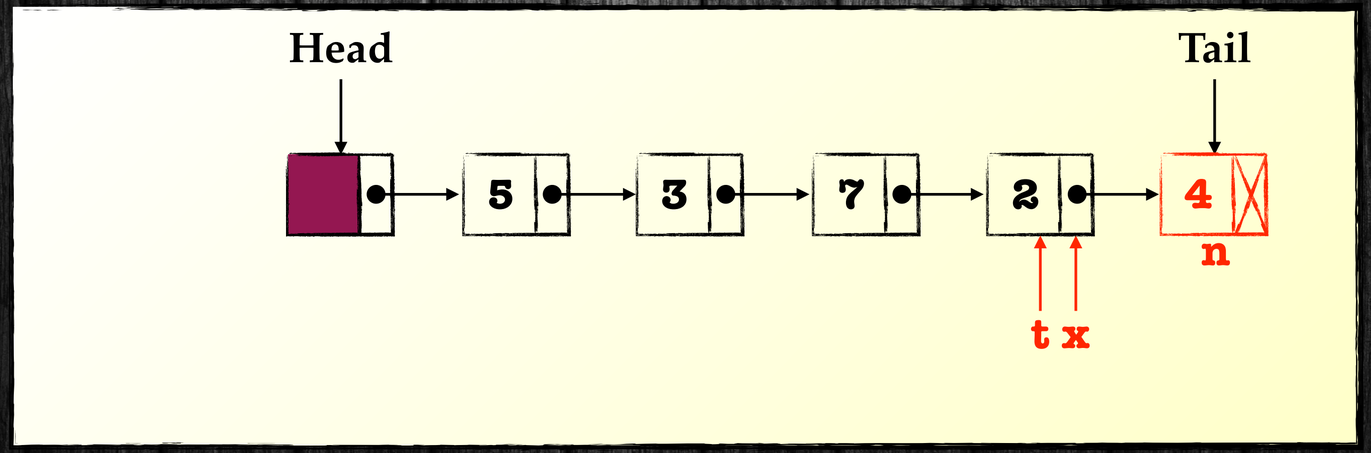
```
enq(k):
Node t,x
 1  n = new Node(k,NULL)
 2  while (true)
 3    t = Tail
 4    x = t.next
 5    if (t == Tail)
 6      if (x == NULL)
 7        if (CAS (t.next,NULL,n))
 8          CAS (Tail,t,n)
 9          exit
10      else
11        CAS (Tail,t,x)
```



enq(4)

queue

5 3 7 2



The Michael-Scott Queue Algorithm

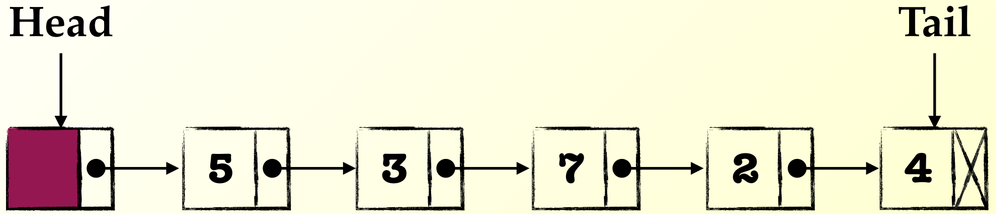
```
enq(k):
Node t,x
 1  n = new Node(k,NULL)
 2  while (true)
 3    t = Tail
 4    x = t.next
 5    if (t == Tail)
 6      if (x == NULL)
 7        if (CAS (t.next,NULL,n))
 8          CAS (Tail,t,n)
 9          exit
10      else
11        CAS (Tail,t,x)
```



enq(4)

queue

5 3 7 1



The Michael-Scott Queue Algorithm

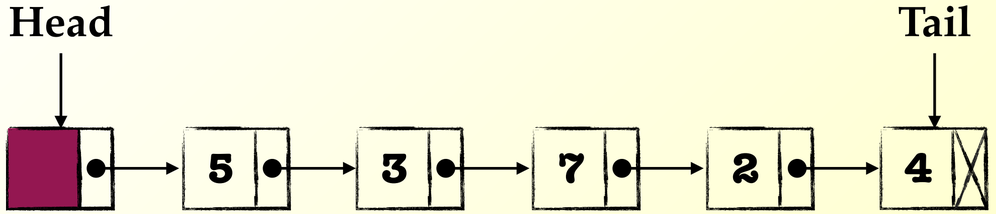
```
enq(k):
Node t,x
 1  n = new Node(k,NULL)
 2  while (true)
 3    t = Tail
 4    x = t.next
 5    if (t == Tail)
 6      if (x == NULL)
 7        if (CAS (t.next,NULL,n))
 8          CAS (Tail,t,n)
 9          exit
10      else
11        CAS (Tail,t,x)
```



enq(4)

queue

5 3 7 2 4

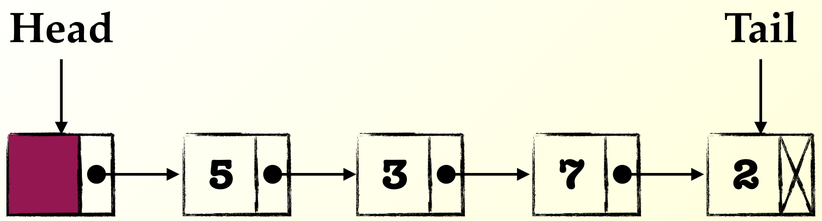


The Michael-Scott Queue Algorithm

Concurrent enqueue operations

```
enq(k):
Node t,x
1  n = new Node(k,NULL)
2  while (true)
3    t = Tail
4    x = t.next
5    if (t == Tail)
6      if (x == NULL)
7        if (CAS (t.next,NULL,n))
8          CAS (Tail,t,n)
9          exit
10     else
11       CAS (Tail,t,x)
```


```
enq(k):
Node t,x
1  n = new Node(k,NULL)
2  while (true)
3    t = Tail
4    x = t.next
5    if (t == Tail)
6      if (x == NULL)
7        if (CAS (t.next,NULL,n))
8          CAS (Tail,t,n)
9          exit
10     else
11       CAS (Tail,t,x)
```

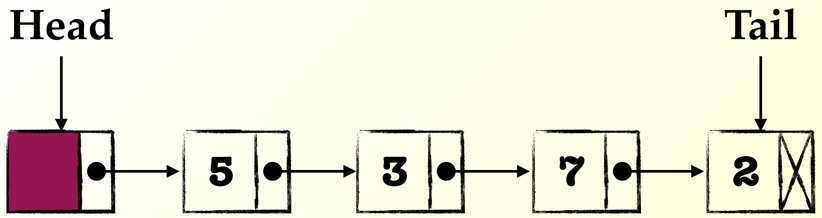
The Michael-Scott Queue Algorithm

Concurrent enqueue operations

```
enq(k):
Node t,x
 1  n = new Node(k,NULL)
 2  while (true)
 3    t = Tail
 4    x = t.next
 5    if (t == Tail)
 6      if (x == NULL)
 7        if (CAS (t.next,NULL,n))
 8          CAS (Tail,t,n)
 9          exit
10      else
11        CAS (Tail,t,x)
```




```
enq(k):
Node t,x
 1  n = new Node(k,NULL)
 2  while (true)
 3    t = Tail
 4    x = t.next
 5    if (t == Tail)
 6      if (x == NULL)
 7        if (CAS (t.next,NULL,n))
 8          CAS (Tail,t,n)
 9          exit
10      else
11        CAS (Tail,t,x)
```

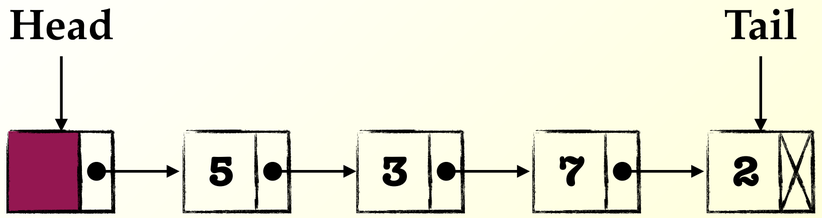
The Michael-Scott Queue Algorithm

Concurrent enqueue operations

```
enq(k):
Node t,x
1  n = new Node(k,NULL)
2  while (true)
3    t = Tail
4    x = t.next
5    if (t == Tail)
6      if (x == NULL)
7        if (CAS (t.next,NULL,n))
8          CAS (Tail,t,n)
9          exit
10     else
11       CAS (Tail,t,x)
```



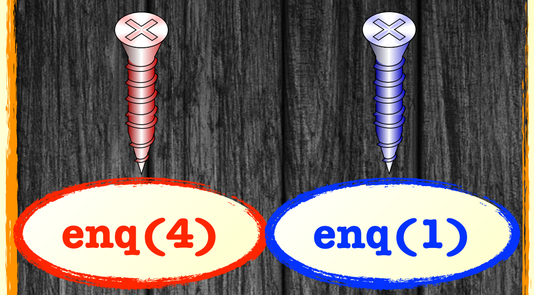

```
enq(k):
Node t,x
1  n = new Node(k,NULL)
2  while (true)
3    t = Tail
4    x = t.next
5    if (t == Tail)
6      if (x == NULL)
7        if (CAS (t.next,NULL,n))
8          CAS (Tail,t,n)
9          exit
10     else
11       CAS (Tail,t,x)
```



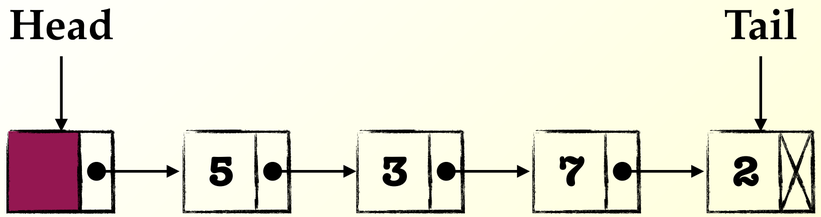
The Michael-Scott Queue Algorithm

Concurrent enqueue operations

```
enq(k):
Node t,x
1  n = new Node(k,NULL)
2  while (true)
3    t = Tail
4    x = t.next
5    if (t == Tail)
6      if (x == NULL)
7        if (CAS (t.next,NULL,n))
8          CAS (Tail,t,n)
9          exit
10     else
11       CAS (Tail,t,x)
```



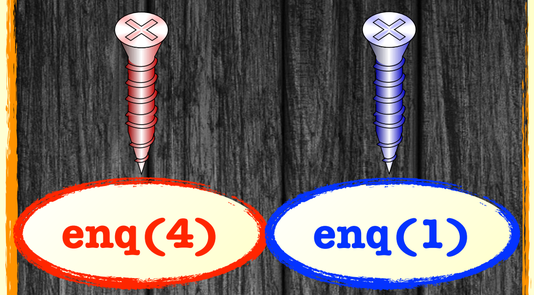

```
enq(k):
Node t,x
1  n = new Node(k,NULL)
2  while (true)
3    t = Tail
4    x = t.next
5    if (t == Tail)
6      if (x == NULL)
7        if (CAS (t.next,NULL,n))
8          CAS (Tail,t,n)
9          exit
10     else
11       CAS (Tail,t,x)
```



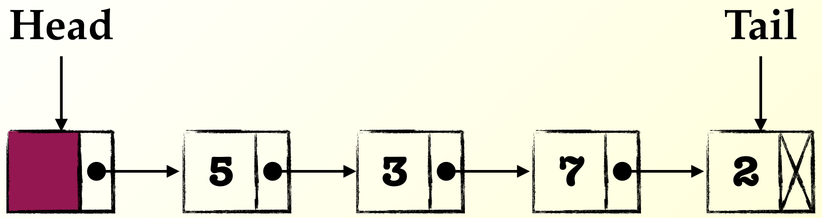
The Michael-Scott Queue Algorithm

Concurrent enqueue operations

```
enq(k):
Node t,x
1  n = new Node(k,NULL)
2  while (true)
3    t = Tail
4    x = t.next
5    if (t == Tail)
6      if (x == NULL)
7        if (CAS (t.next,NULL,n))
8          CAS (Tail,t,n)
9          exit
10     else
11       CAS (Tail,t,x)
```



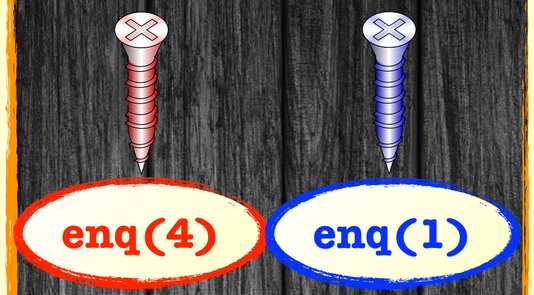

```
enq(k):
Node t,x
1  n = new Node(k,NULL)
2  while (true)
3    t = Tail
4    x = t.next
5    if (t == Tail)
6      if (x == NULL)
7        if (CAS (t.next,NULL,n))
8          CAS (Tail,t,n)
9          exit
10     else
11       CAS (Tail,t,x)
```



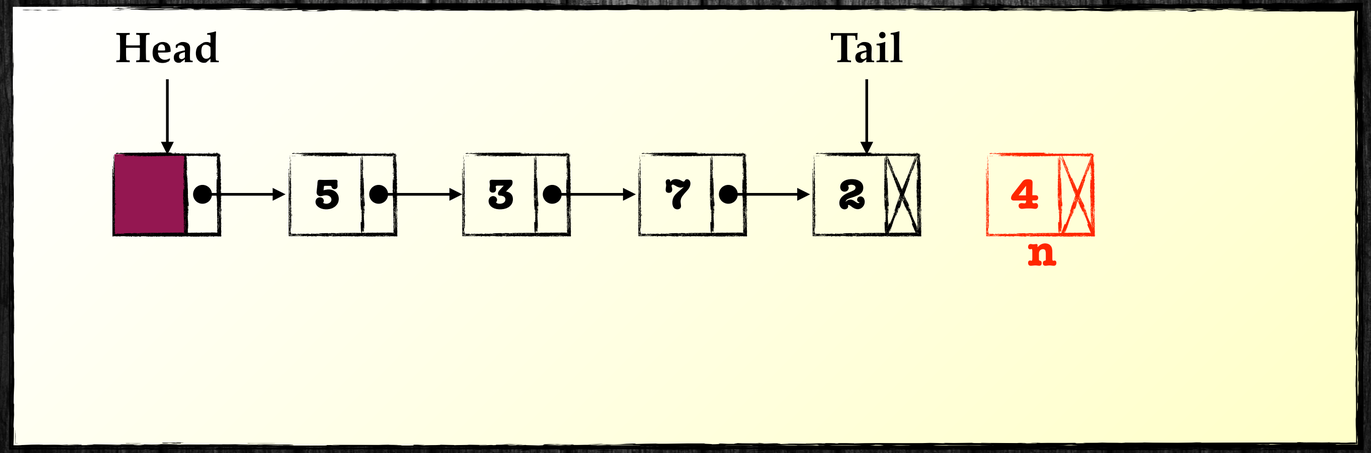
The Michael-Scott Queue Algorithm

Concurrent enqueue operations

```
enq(k):
Node t,x
1  n = new Node(k,NULL)
2  while (true)
3    t = Tail
4    x = t.next
5    if (t == Tail)
6      if (x == NULL)
7        if (CAS (t.next,NULL,n))
8          CAS (Tail,t,n)
9          exit
10     else
11       CAS (Tail,t,x)
```



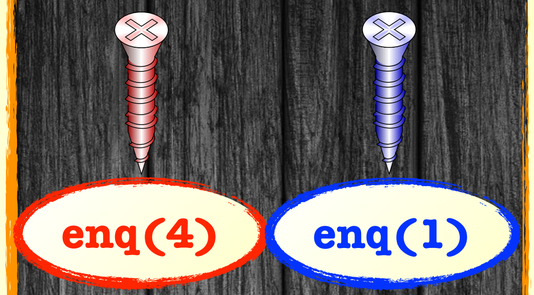

```
enq(k):
Node t,x
1  n = new Node(k,NULL)
2  while (true)
3    t = Tail
4    x = t.next
5    if (t == Tail)
6      if (x == NULL)
7        if (CAS (t.next,NULL,n))
8          CAS (Tail,t,n)
9          exit
10     else
11       CAS (Tail,t,x)
```

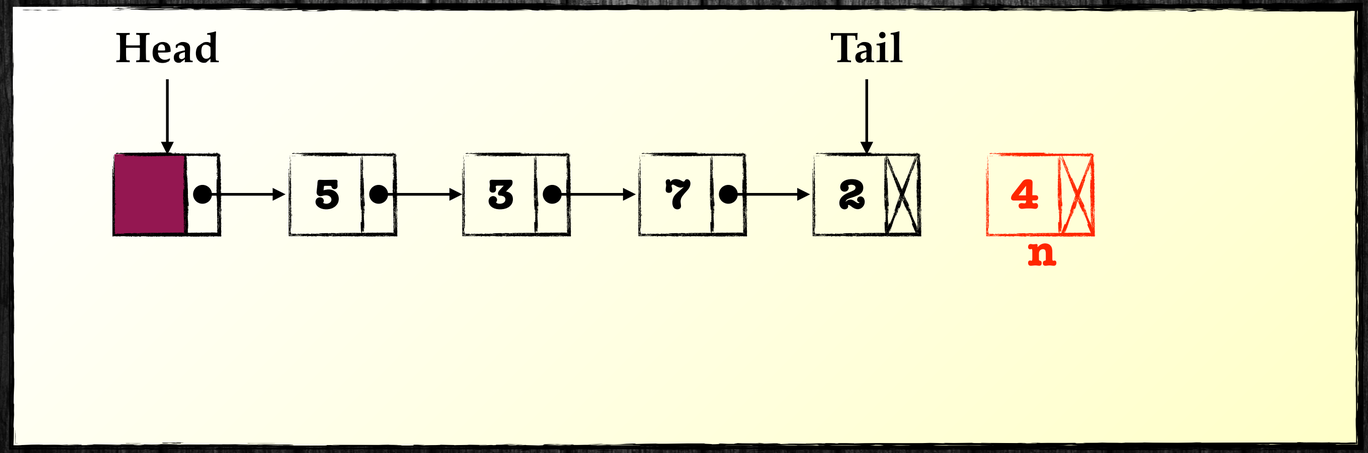
The Michael-Scott Queue Algorithm

Concurrent enqueue operations

```
enq(k):
Node t,x
1  n = new Node(k,NULL)
2  while (true)
3    t = Tail
4    x = t.next
5    if (t == Tail)
6      if (x == NULL)
7        if (CAS (t.next,NULL,n))
8          CAS (Tail,t,n)
9          exit
10     else
11       CAS (Tail,t,x)
```



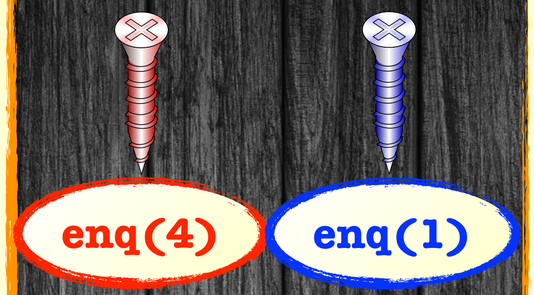

```
enq(k):
Node t,x
1  n = new Node(k,NULL)
2  while (true)
3    t = Tail
4    x = t.next
5    if (t == Tail)
6      if (x == NULL)
7        if (CAS (t.next,NULL,n))
8          CAS (Tail,t,n)
9          exit
10     else
11       CAS (Tail,t,x)
```



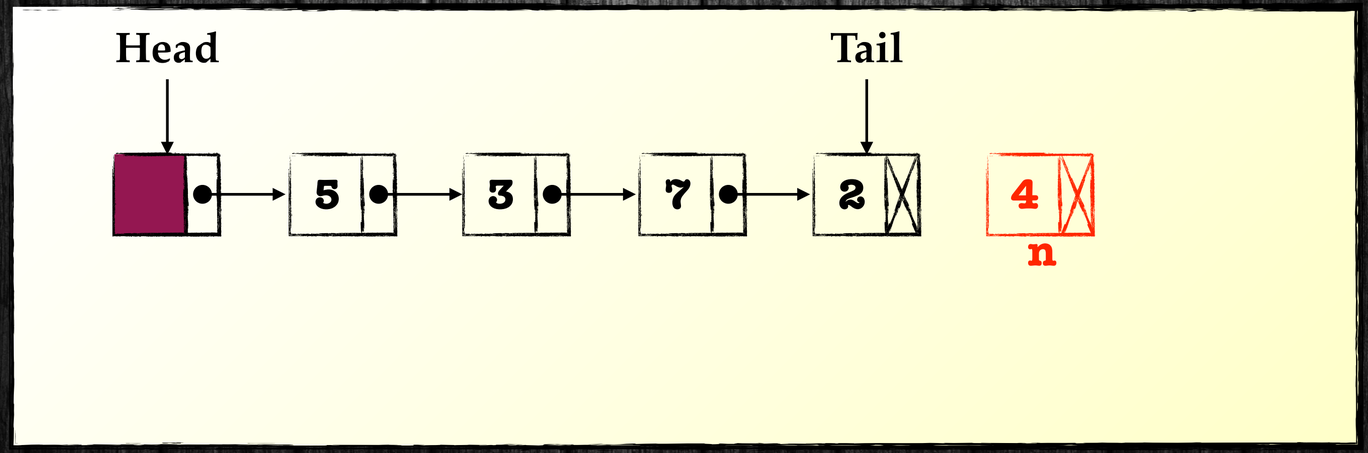
The Michael-Scott Queue Algorithm

Concurrent enqueue operations

```
enq(k):
Node t,x
1  n = new Node(k,NULL)
2  while (true)
3    t = Tail
4    x = t.next
5    if (t == Tail)
6      if (x == NULL)
7        if (CAS (t.next,NULL,n))
8          CAS (Tail,t,n)
9          exit
10     else
11       CAS (Tail,t,x)
```



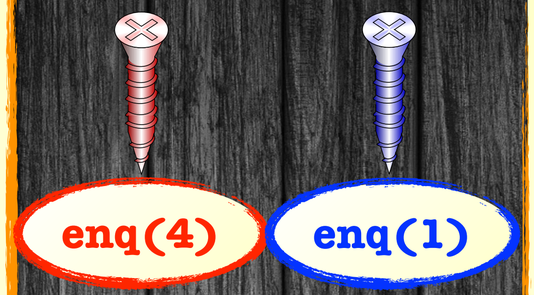

```
enq(k):
Node t,x
1  n = new Node(k,NULL)
2  while (true)
3    t = Tail
4    x = t.next
5    if (t == Tail)
6      if (x == NULL)
7        if (CAS (t.next,NULL,n))
8          CAS (Tail,t,n)
9          exit
10     else
11       CAS (Tail,t,x)
```



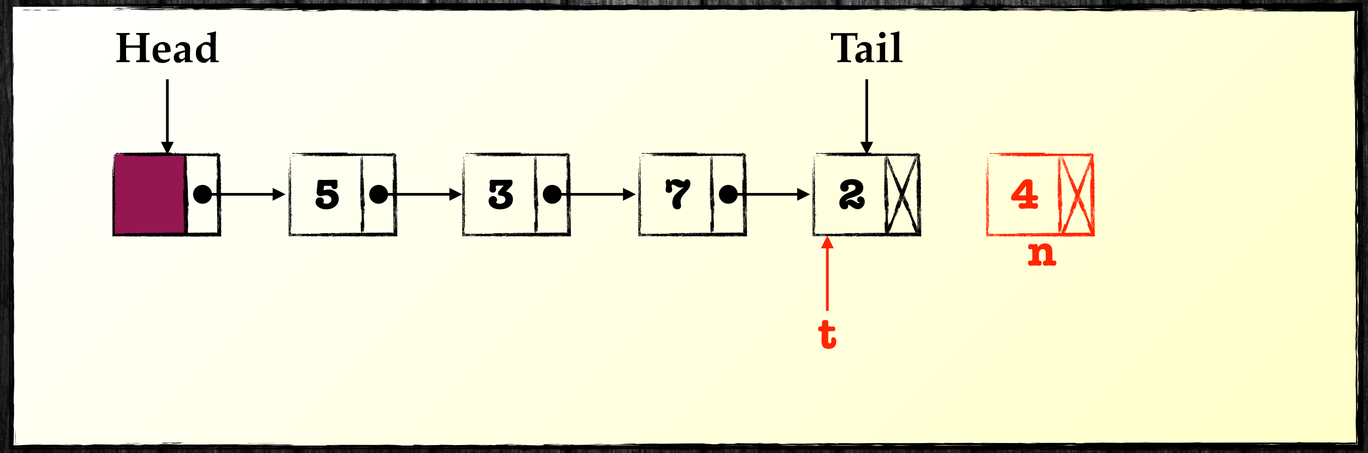
The Michael-Scott Queue Algorithm

Concurrent enqueue operations

```
enq(k):
Node t,x
1  n = new Node(k,NULL)
2  while (true)
3    t = Tail
4    x = t.next
5    if (t == Tail)
6      if (x == NULL)
7        if (CAS (t.next,NULL,n))
8          CAS (Tail,t,n)
9          exit
10     else
11       CAS (Tail,t,x)
```



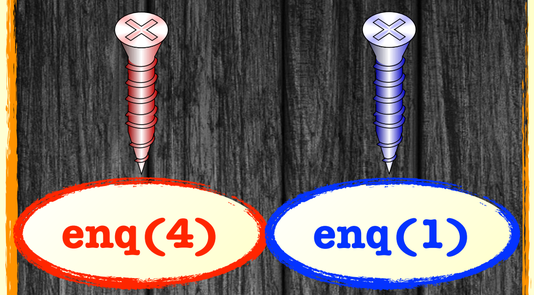

```
enq(k):
Node t,x
1  n = new Node(k,NULL)
2  while (true)
3    t = Tail
4    x = t.next
5    if (t == Tail)
6      if (x == NULL)
7        if (CAS (t.next,NULL,n))
8          CAS (Tail,t,n)
9          exit
10     else
11       CAS (Tail,t,x)
```



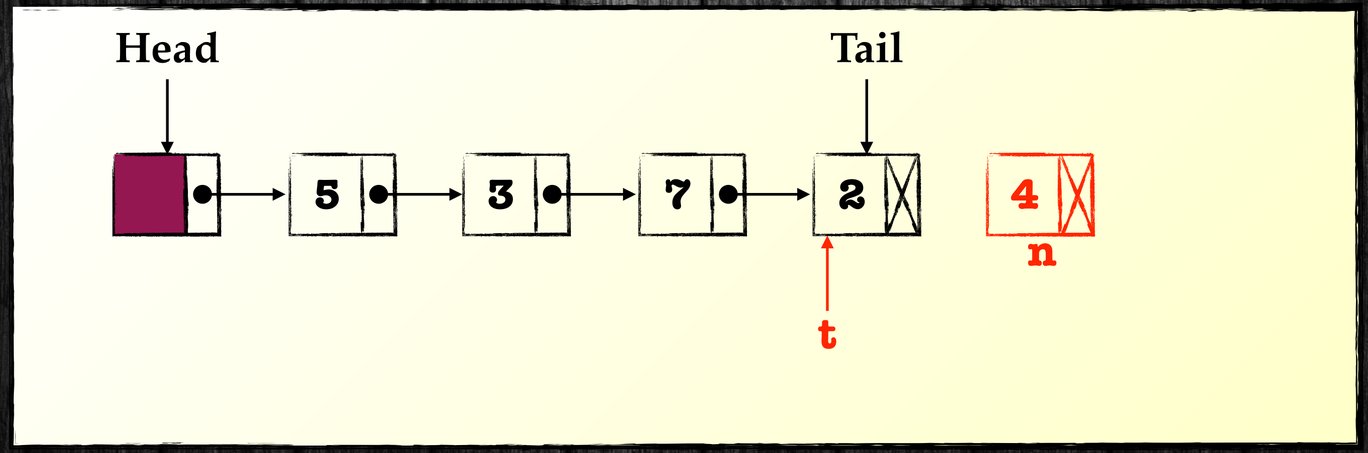
The Michael-Scott Queue Algorithm

Concurrent enqueue operations

```
enq(k):
Node t,x
1  n = new Node(k,NULL)
2  while (true)
3    t = Tail
4    x = t.next
5    if (t == Tail)
6      if (x == NULL)
7        if (CAS (t.next,NULL,n))
8          CAS (Tail,t,n)
9          exit
10     else
11       CAS (Tail,t,x)
```



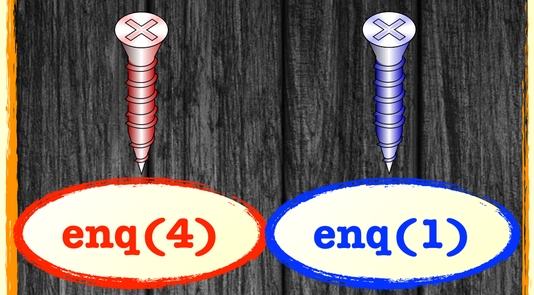

```
enq(k):
Node t,x
1  n = new Node(k,NULL)
2  while (true)
3    t = Tail
4    x = t.next
5    if (t == Tail)
6      if (x == NULL)
7        if (CAS (t.next,NULL,n))
8          CAS (Tail,t,n)
9          exit
10     else
11       CAS (Tail,t,x)
```



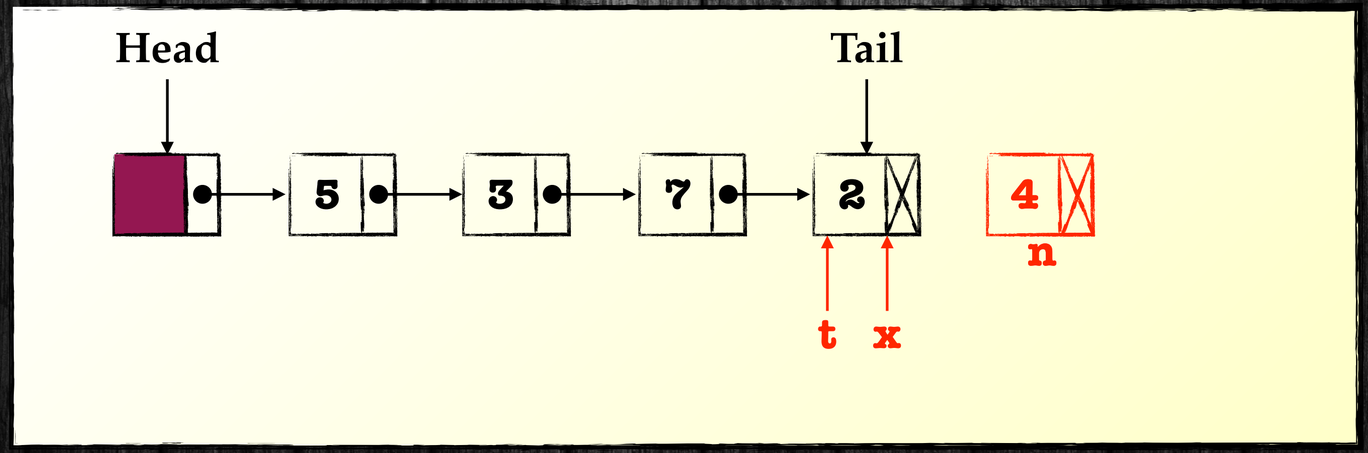
The Michael-Scott Queue Algorithm

Concurrent enqueue operations

```
enq(k):
Node t,x
1  n = new Node(k,NULL)
2  while (true)
3    t = Tail
4    x = t.next
5    if (t == Tail)
6      if (x == NULL)
7        if (CAS (t.next,NULL,n))
8          CAS (Tail,t,n)
9          exit
10     else
11       CAS (Tail,t,x)
```



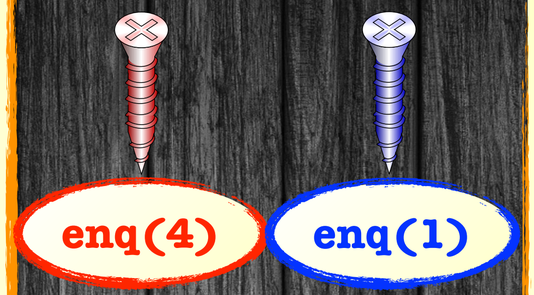

```
enq(k):
Node t,x
1  n = new Node(k,NULL)
2  while (true)
3    t = Tail
4    x = t.next
5    if (t == Tail)
6      if (x == NULL)
7        if (CAS (t.next,NULL,n))
8          CAS (Tail,t,n)
9          exit
10     else
11       CAS (Tail,t,x)
```



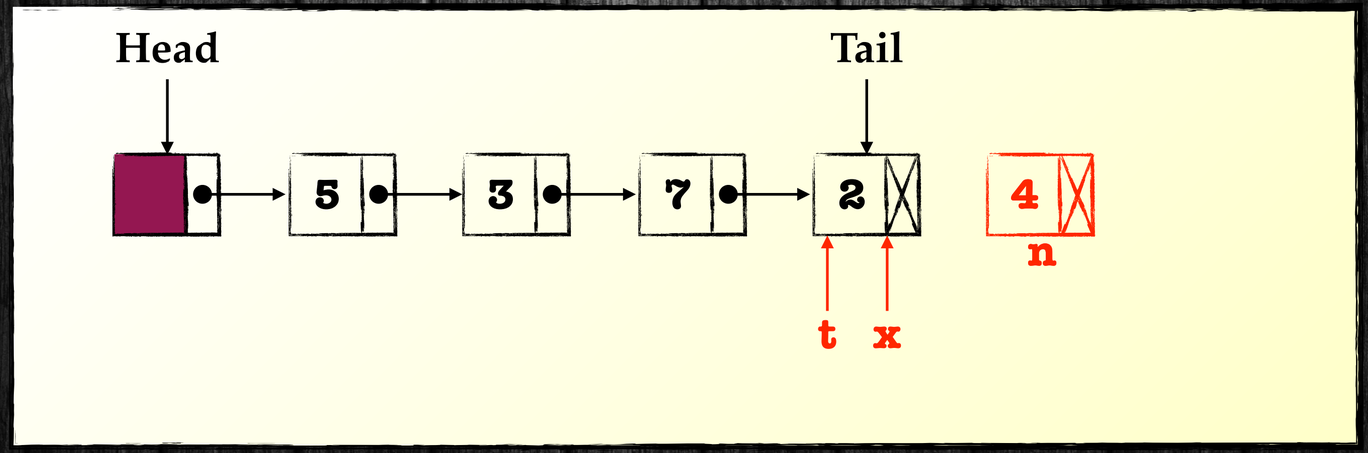
The Michael-Scott Queue Algorithm

Concurrent enqueue operations

```
enq(k):
Node t,x
1  n = new Node(k,NULL)
2  while (true)
3    t = Tail
4    x = t.next
5    if (t == Tail)
6      if (x == NULL)
7        if (CAS (t.next,NULL,n))
8          CAS (Tail,t,n)
9          exit
10     else
11       CAS (Tail,t,x)
```



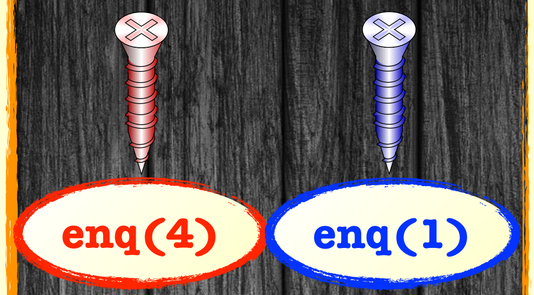

```
enq(k):
Node t,x
1  n = new Node(k,NULL)
2  while (true)
3    t = Tail
4    x = t.next
5    if (t == Tail)
6      if (x == NULL)
7        if (CAS (t.next,NULL,n))
8          CAS (Tail,t,n)
9          exit
10     else
11       CAS (Tail,t,x)
```



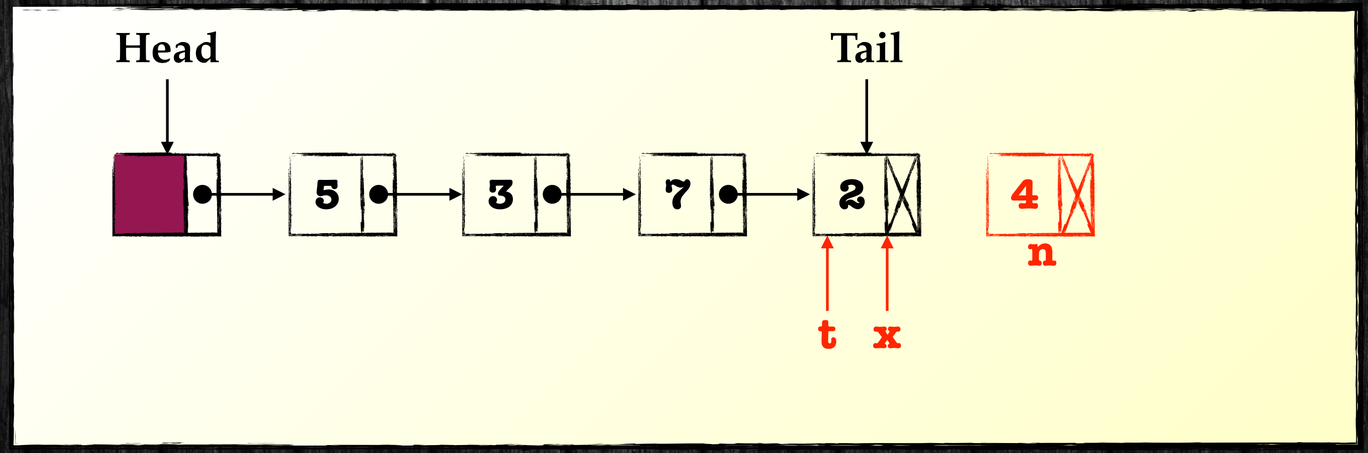
The Michael-Scott Queue Algorithm

Concurrent enqueue operations

```
enq(k):
Node t,x
1  n = new Node(k,NULL)
2  while (true)
3    t = Tail
4    x = t.next
5    if (t == Tail)
6      if (x == NULL)
7        if (CAS (t.next,NULL,n))
8          CAS (Tail,t,n)
9          exit
10     else
11       CAS (Tail,t,x)
```



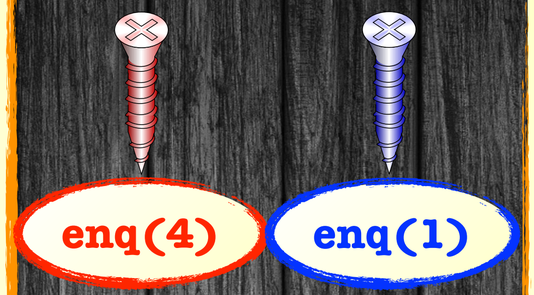

```
enq(k):
Node t,x
1  n = new Node(k,NULL)
2  while (true)
3    t = Tail
4    x = t.next
5    if (t == Tail)
6      if (x == NULL)
7        if (CAS (t.next,NULL,n))
8          CAS (Tail,t,n)
9          exit
10     else
11       CAS (Tail,t,x)
```



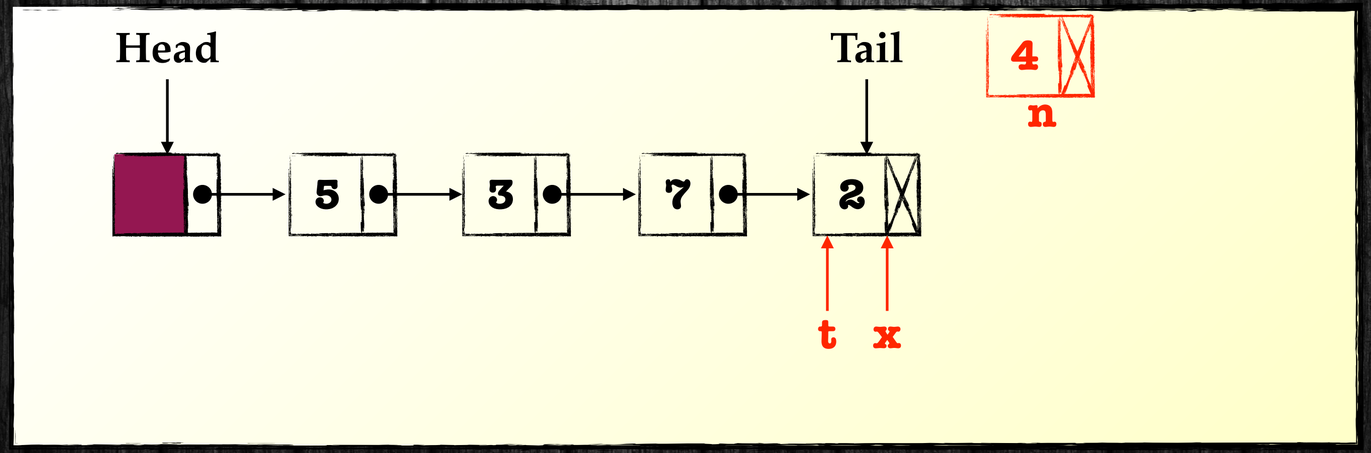
The Michael-Scott Queue Algorithm

Concurrent enqueue operations

```
enq(k):
Node t,x
1  n = new Node(k,NULL)
2  while (true)
3    t = Tail
4    x = t.next
5    if (t == Tail)
6      if (x == NULL)
7        if (CAS (t.next,NULL,n))
8          CAS (Tail,t,n)
9          exit
10     else
11       CAS (Tail,t,x)
```



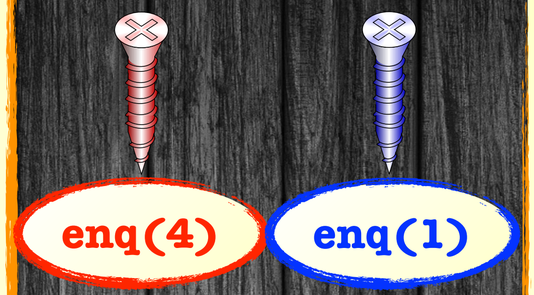

```
enq(k):
Node t,x
1  n = new Node(k,NULL)
2  while (true)
3    t = Tail
4    x = t.next
5    if (t == Tail)
6      if (x == NULL)
7        if (CAS (t.next,NULL,n))
8          CAS (Tail,t,n)
9          exit
10     else
11       CAS (Tail,t,x)
```



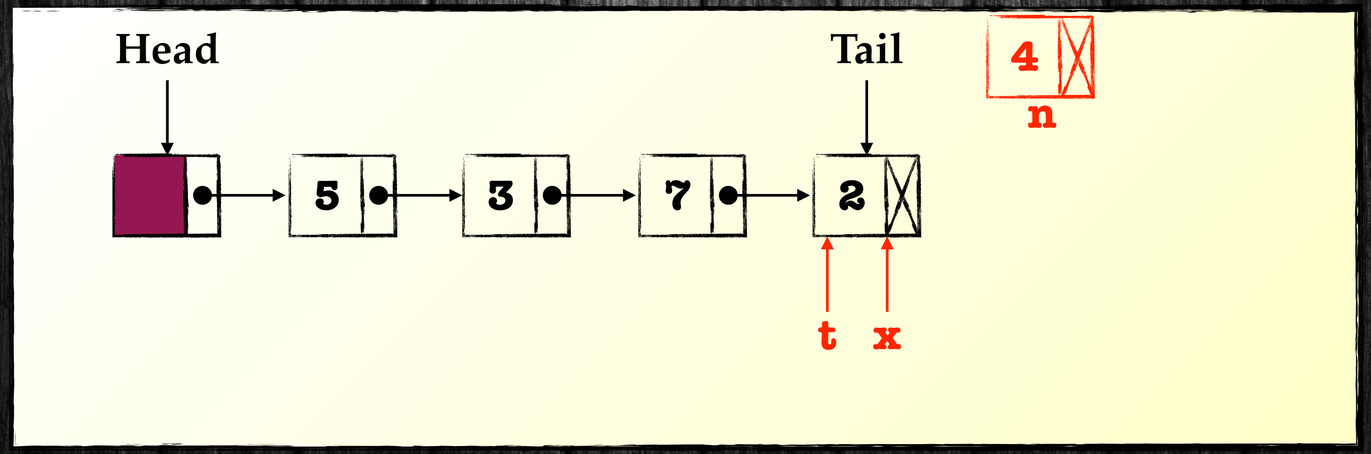
The Michael-Scott Queue Algorithm

Concurrent enqueue operations

```
enq(k):
Node t,x
1  n = new Node(k,NULL)
2  while (true)
3    t = Tail
4    x = t.next
5    if (t == Tail)
6      if (x == NULL)
7        if (CAS (t.next,NULL,n))
8          CAS (Tail,t,n)
9          exit
10     else
11       CAS (Tail,t,x)
```



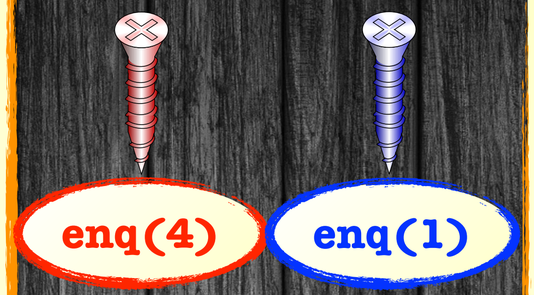

```
enq(k):
Node t,x
1  n = new Node(k,NULL)
2  while (true)
3    t = Tail
4    x = t.next
5    if (t == Tail)
6      if (x == NULL)
7        if (CAS (t.next,NULL,n))
8          CAS (Tail,t,n)
9          exit
10     else
11       CAS (Tail,t,x)
```



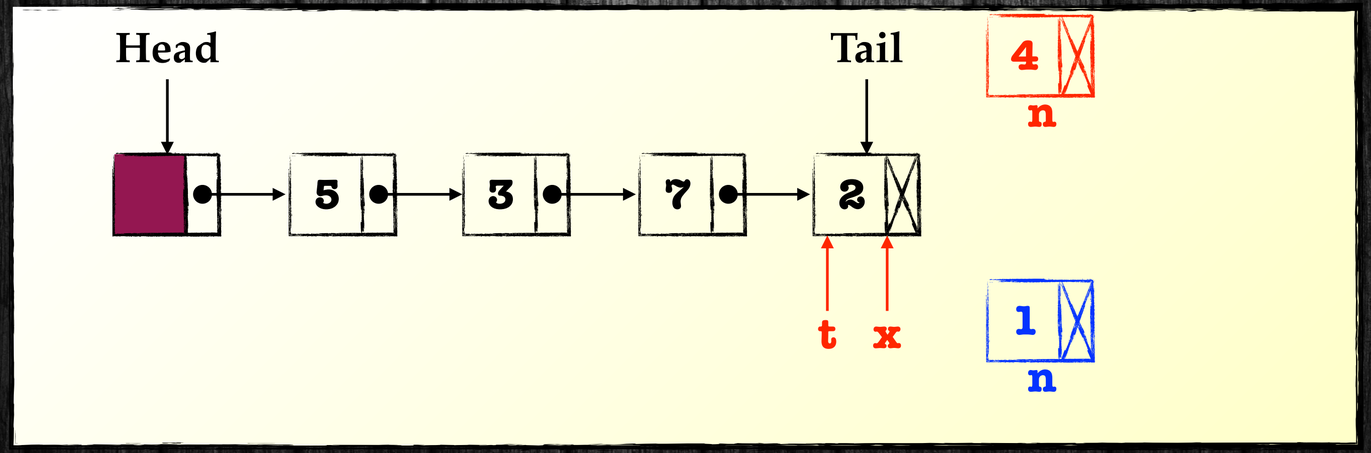
The Michael-Scott Queue Algorithm

Concurrent enqueue operations

```
enq(k):
Node t,x
1  n = new Node(k,NULL)
2  while (true)
3    t = Tail
4    x = t.next
5    if (t == Tail)
6      if (x == NULL)
7        if (CAS (t.next,NULL,n))
8          CAS (Tail,t,n)
9          exit
10     else
11       CAS (Tail,t,x)
```



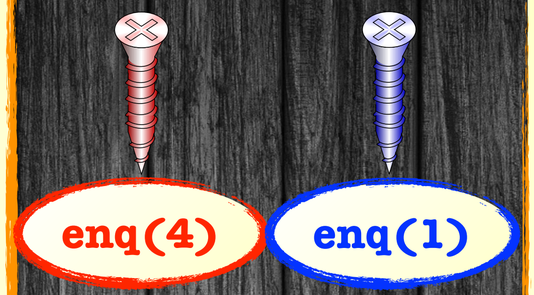

```
enq(k):
Node t,x
1  n = new Node(k,NULL)
2  while (true)
3    t = Tail
4    x = t.next
5    if (t == Tail)
6      if (x == NULL)
7        if (CAS (t.next,NULL,n))
8          CAS (Tail,t,n)
9          exit
10     else
11       CAS (Tail,t,x)
```



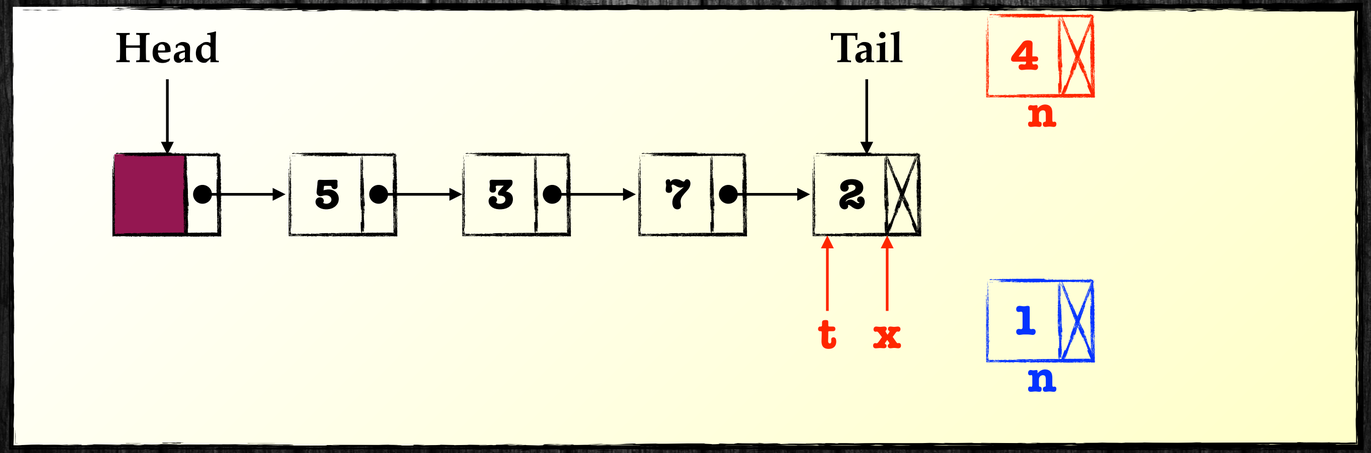
The Michael-Scott Queue Algorithm

Concurrent enqueue operations

```
enq(k):
Node t,x
1  n = new Node(k,NULL)
2  while (true)
3    t = Tail
4    x = t.next
5    if (t == Tail)
6      if (x == NULL)
7        if (CAS (t.next,NULL,n))
8          CAS (Tail,t,n)
9          exit
10     else
11       CAS (Tail,t,x)
```



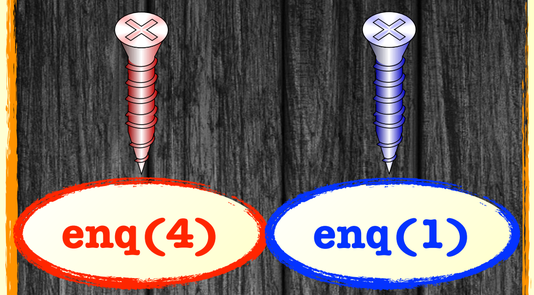

```
enq(k):
Node t,x
1  n = new Node(k,NULL)
2  while (true)
3    t = Tail
4    x = t.next
5    if (t == Tail)
6      if (x == NULL)
7        if (CAS (t.next,NULL,n))
8          CAS (Tail,t,n)
9          exit
10     else
11       CAS (Tail,t,x)
```



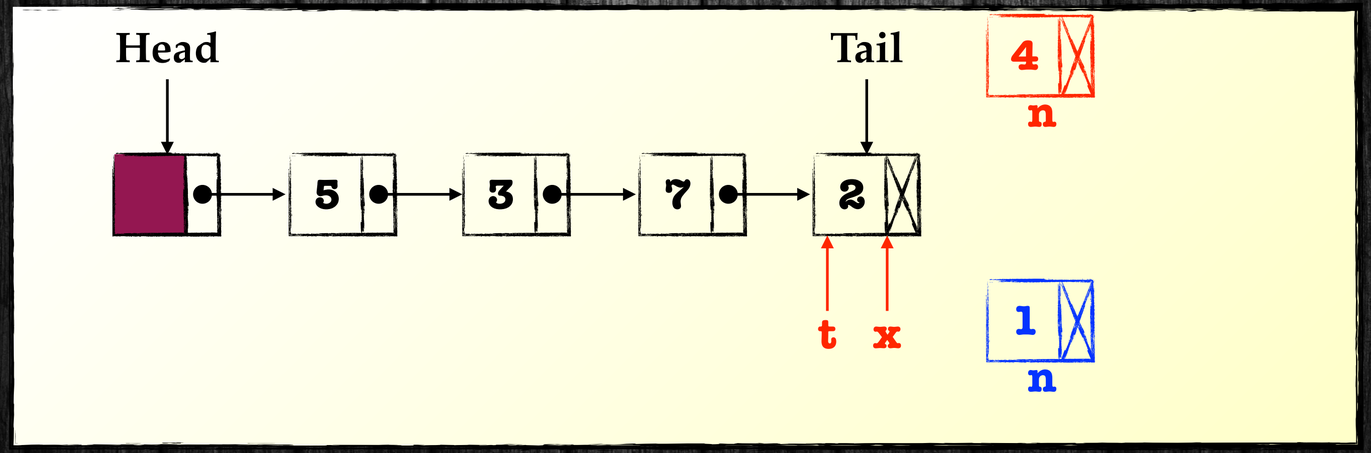
The Michael-Scott Queue Algorithm

Concurrent enqueue operations

```
enq(k):
Node t,x
1  n = new Node(k,NULL)
2  while (true)
3    t = Tail
4    x = t.next
5    if (t == Tail)
6      if (x == NULL)
7        if (CAS (t.next,NULL,n))
8          CAS (Tail,t,n)
9          exit
10     else
11       CAS (Tail,t,x)
```



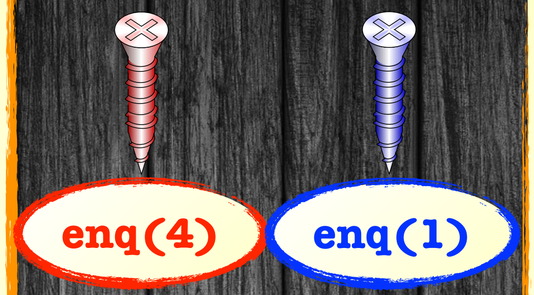

```
enq(k):
Node t,x
1  n = new Node(k,NULL)
2  while (true)
3    t = Tail
4    x = t.next
5    if (t == Tail)
6      if (x == NULL)
7        if (CAS (t.next,NULL,n))
8          CAS (Tail,t,n)
9          exit
10     else
11       CAS (Tail,t,x)
```



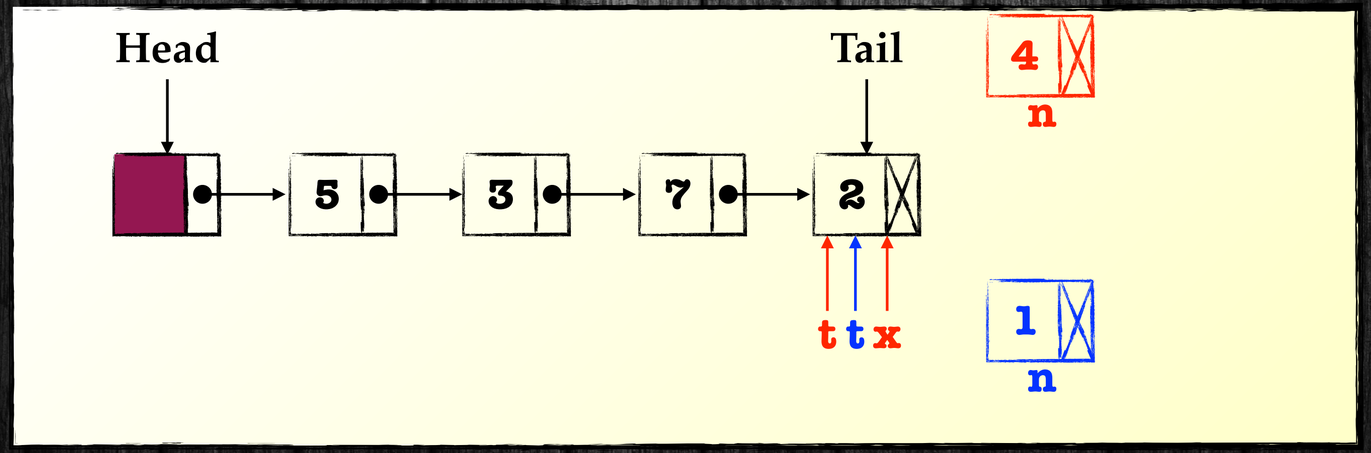
The Michael-Scott Queue Algorithm

Concurrent enqueue operations

```
enq(k):
Node t,x
1  n = new Node(k,NULL)
2  while (true)
3    t = Tail
4    x = t.next
5    if (t == Tail)
6      if (x == NULL)
7        if (CAS (t.next,NULL,n))
8          CAS (Tail,t,n)
9          exit
10     else
11       CAS (Tail,t,x)
```



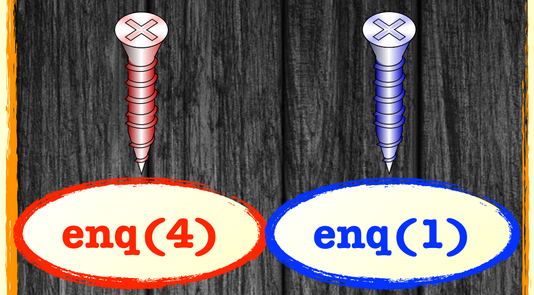

```
enq(k):
Node t,x
1  n = new Node(k,NULL)
2  while (true)
3    t = Tail
4    x = t.next
5    if (t == Tail)
6      if (x == NULL)
7        if (CAS (t.next,NULL,n))
8          CAS (Tail,t,n)
9          exit
10     else
11       CAS (Tail,t,x)
```



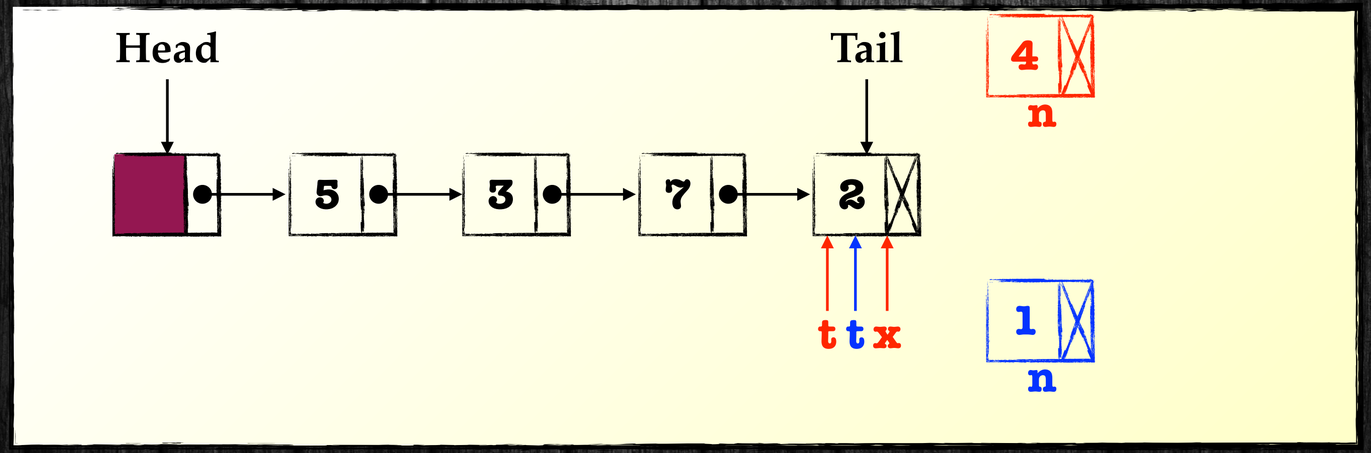
The Michael-Scott Queue Algorithm

Concurrent enqueue operations

```
enq(k):
Node t,x
1  n = new Node(k,NULL)
2  while (true)
3    t = Tail
4    x = t.next
5    if (t == Tail)
6      if (x == NULL)
7        if (CAS (t.next,NULL,n))
8          CAS (Tail,t,n)
9          exit
10     else
11       CAS (Tail,t,x)
```



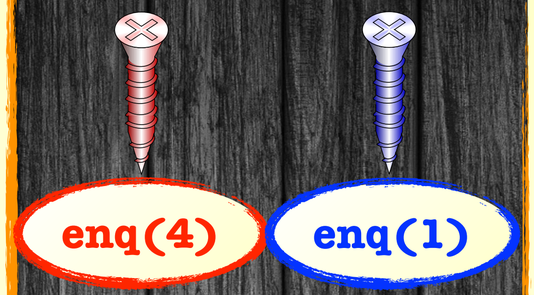

```
enq(k):
Node t,x
1  n = new Node(k,NULL)
2  while (true)
3    t = Tail
4    x = t.next
5    if (t == Tail)
6      if (x == NULL)
7        if (CAS (t.next,NULL,n))
8          CAS (Tail,t,n)
9          exit
10     else
11       CAS (Tail,t,x)
```



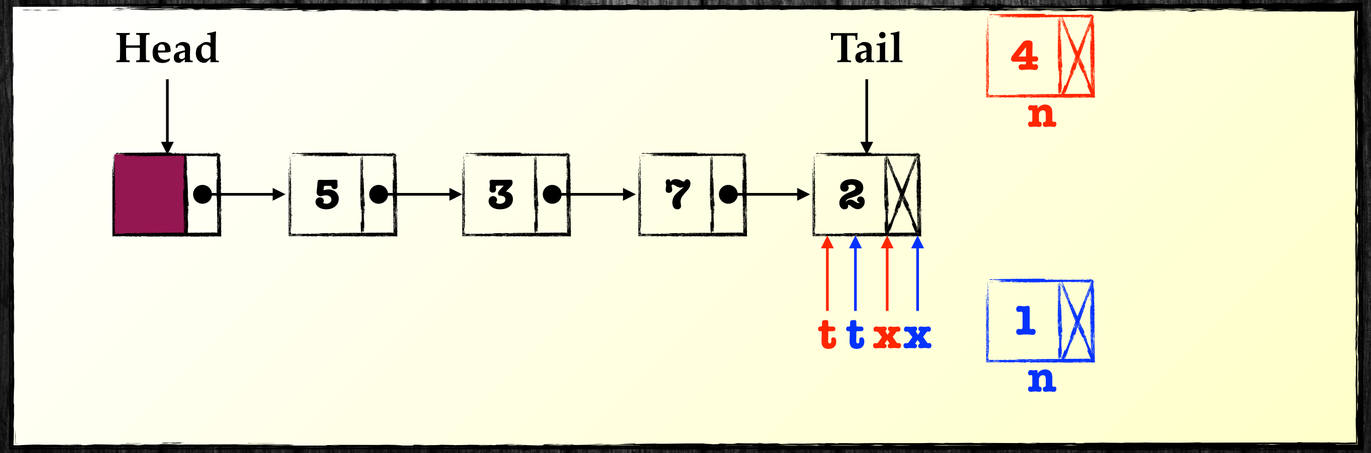
The Michael-Scott Queue Algorithm

Concurrent enqueue operations

```
enq(k):
Node t,x
1  n = new Node(k,NULL)
2  while (true)
3    t = Tail
4    x = t.next
5    if (t == Tail)
6      if (x == NULL)
7        if (CAS (t.next,NULL,n))
8          CAS (Tail,t,n)
9          exit
10     else
11       CAS (Tail,t,x)
```



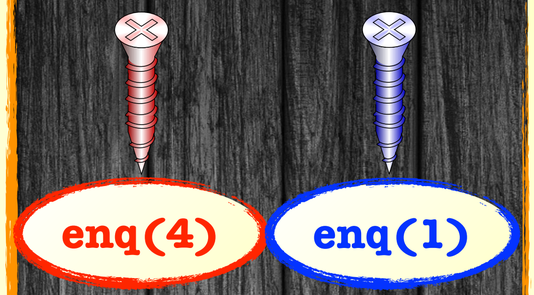

```
enq(k):
Node t,x
1  n = new Node(k,NULL)
2  while (true)
3    t = Tail
4    x = t.next
5    if (t == Tail)
6      if (x == NULL)
7        if (CAS (t.next,NULL,n))
8          CAS (Tail,t,n)
9          exit
10     else
11       CAS (Tail,t,x)
```



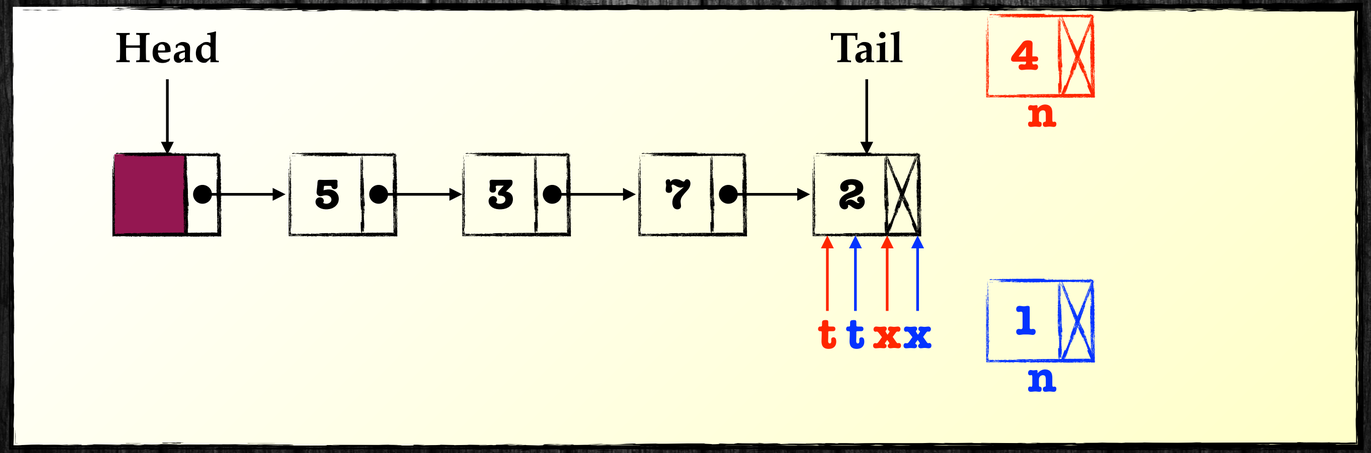
The Michael-Scott Queue Algorithm

Concurrent enqueue operations

```
enq(k):
Node t,x
1  n = new Node(k,NULL)
2  while (true)
3    t = Tail
4    x = t.next
5    if (t == Tail)
6      if (x == NULL)
7        if (CAS (t.next,NULL,n))
8          CAS (Tail,t,n)
9          exit
10     else
11       CAS (Tail,t,x)
```



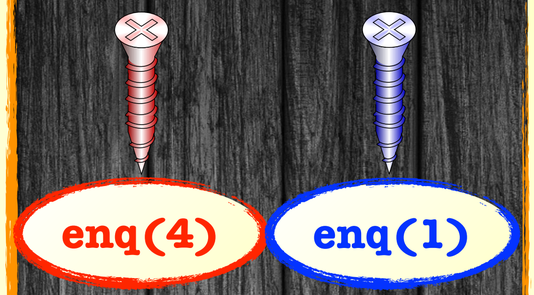

```
enq(k):
Node t,x
1  n = new Node(k,NULL)
2  while (true)
3    t = Tail
4    x = t.next
5    if (t == Tail)
6      if (x == NULL)
7        if (CAS (t.next,NULL,n))
8          CAS (Tail,t,n)
9          exit
10     else
11       CAS (Tail,t,x)
```



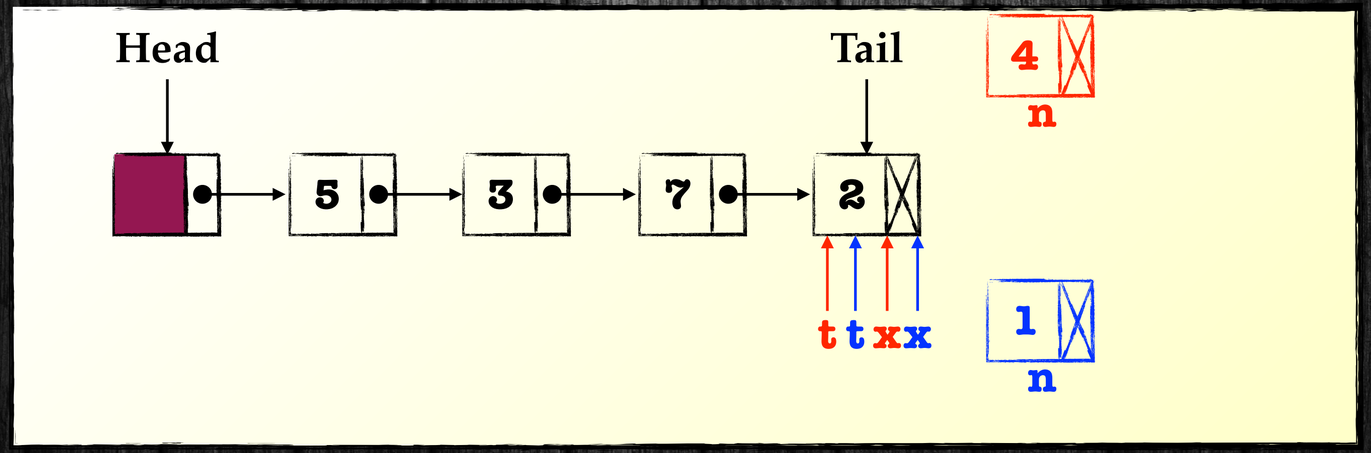
The Michael-Scott Queue Algorithm

Concurrent enqueue operations

```
enq(k):
Node t,x
1  n = new Node(k,NULL)
2  while (true)
3    t = Tail
4    x = t.next
5    if (t == Tail)
6      if (x == NULL)
7        if (CAS (t.next,NULL,n))
8          CAS (Tail,t,n)
9          exit
10     else
11       CAS (Tail,t,x)
```



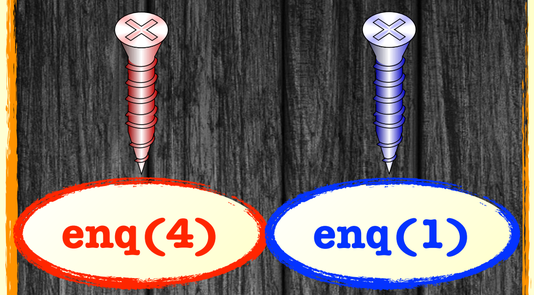

```
enq(k):
Node t,x
1  n = new Node(k,NULL)
2  while (true)
3    t = Tail
4    x = t.next
5    if (t == Tail)
6      if (x == NULL)
7        if (CAS (t.next,NULL,n))
8          CAS (Tail,t,n)
9          exit
10     else
11       CAS (Tail,t,x)
```



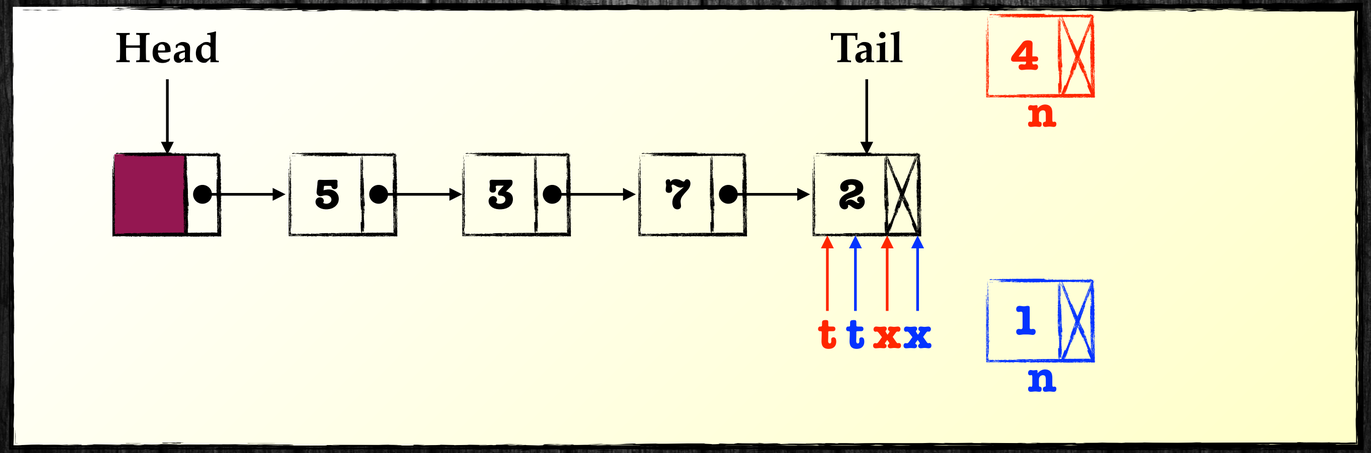
The Michael-Scott Queue Algorithm

Concurrent enqueue operations

```
enq(k):
Node t,x
1  n = new Node(k,NULL)
2  while (true)
3    t = Tail
4    x = t.next
5    if (t == Tail)
6      if (x == NULL)
7        if (CAS (t.next,NULL,n))
8          CAS (Tail,t,n)
9          exit
10     else
11       CAS (Tail,t,x)
```



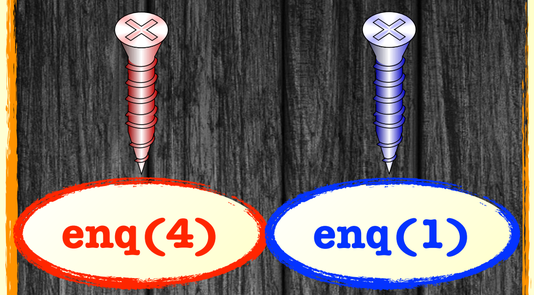

```
enq(k):
Node t,x
1  n = new Node(k,NULL)
2  while (true)
3    t = Tail
4    x = t.next
5    if (t == Tail)
6      if (x == NULL)
7        if (CAS (t.next,NULL,n))
8          CAS (Tail,t,n)
9          exit
10     else
11       CAS (Tail,t,x)
```



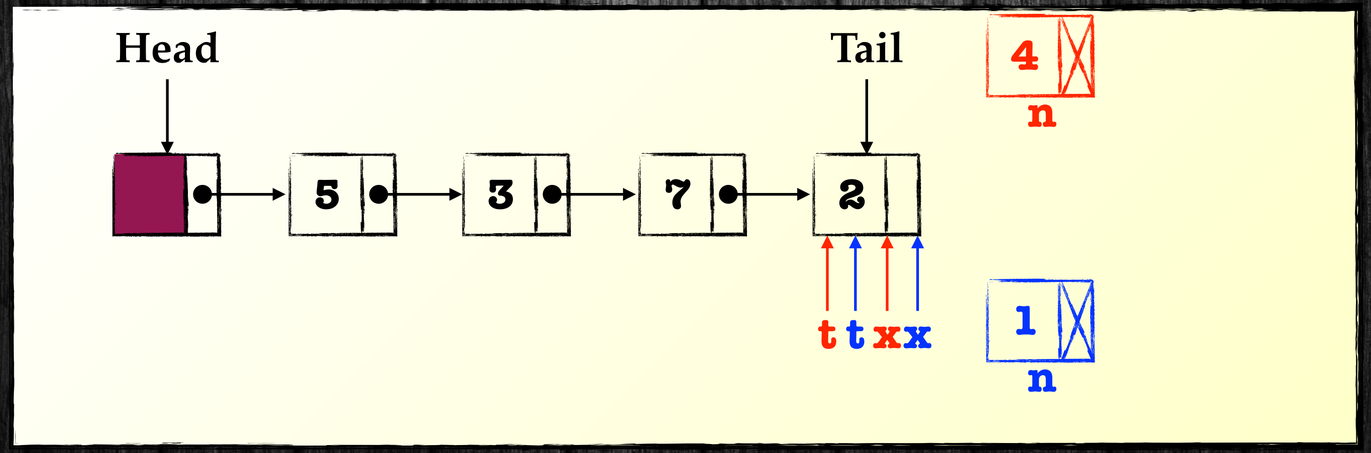
The Michael-Scott Queue Algorithm

Concurrent enqueue operations

```
enq(k):
Node t,x
1  n = new Node(k,NULL)
2  while (true)
3    t = Tail
4    x = t.next
5    if (t == Tail)
6      if (x == NULL)
7        if (CAS (t.next,NULL,n))
8          CAS (Tail,t,n)
9          exit
10     else
11       CAS (Tail,t,x)
```



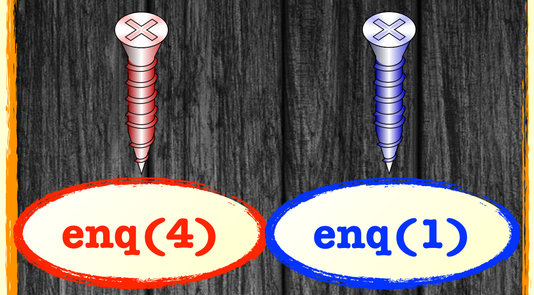

```
enq(k):
Node t,x
1  n = new Node(k,NULL)
2  while (true)
3    t = Tail
4    x = t.next
5    if (t == Tail)
6      if (x == NULL)
7        if (CAS (t.next,NULL,n))
8          CAS (Tail,t,n)
9          exit
10     else
11       CAS (Tail,t,x)
```



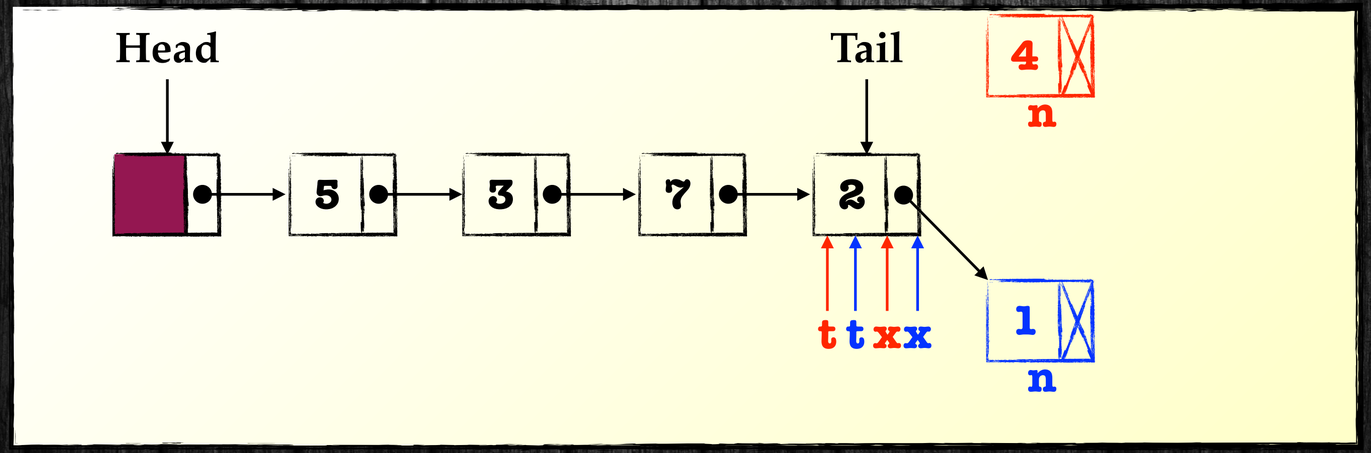
The Michael-Scott Queue Algorithm

Concurrent enqueue operations

```
enq(k):
Node t,x
1  n = new Node(k,NULL)
2  while (true)
3    t = Tail
4    x = t.next
5    if (t == Tail)
6      if (x == NULL)
7        if (CAS (t.next,NULL,n))
8          CAS (Tail,t,n)
9          exit
10     else
11       CAS (Tail,t,x)
```



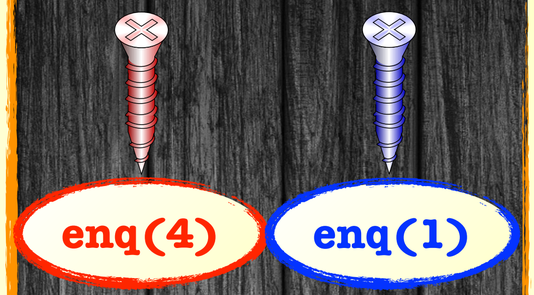

```
enq(k):
Node t,x
1  n = new Node(k,NULL)
2  while (true)
3    t = Tail
4    x = t.next
5    if (t == Tail)
6      if (x == NULL)
7        if (CAS (t.next,NULL,n))
8          CAS (Tail,t,n)
9          exit
10     else
11       CAS (Tail,t,x)
```



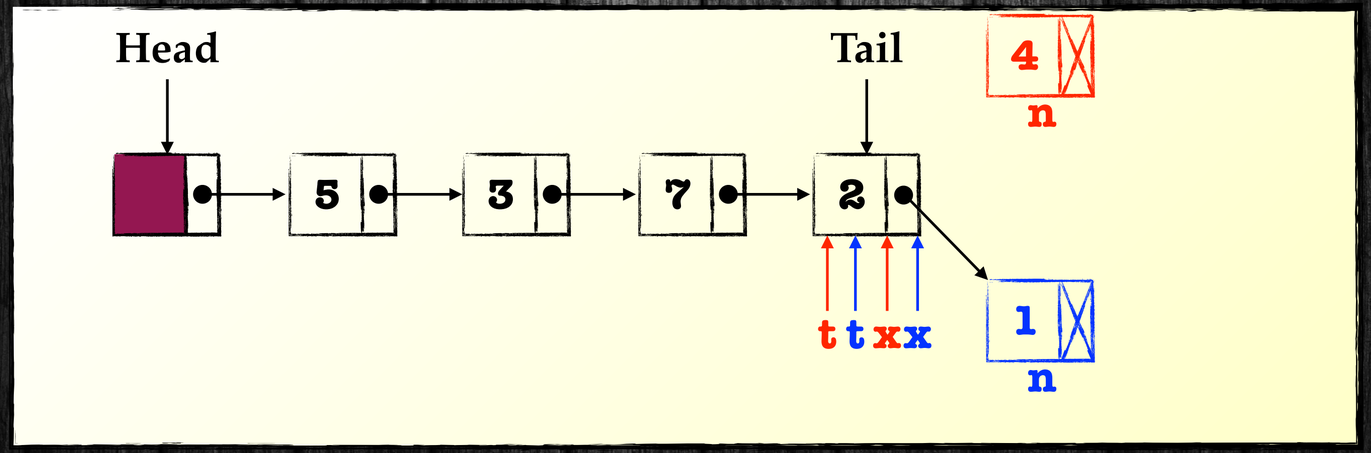
The Michael-Scott Queue Algorithm

Concurrent enqueue operations

```
enq(k):
Node t,x
1  n = new Node(k,NULL)
2  while (true)
3    t = Tail
4    x = t.next
5    if (t == Tail)
6      if (x == NULL)
7        if (CAS (t.next,NULL,n))
8          CAS (Tail,t,n)
9          exit
10     else
11       CAS (Tail,t,x)
```



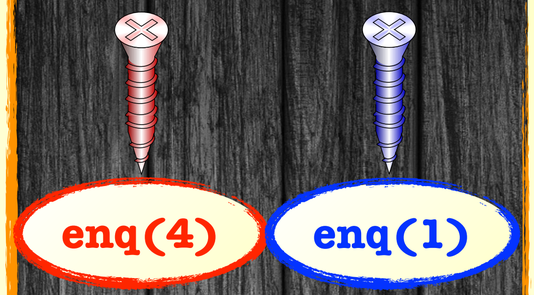

```
enq(k):
Node t,x
1  n = new Node(k,NULL)
2  while (true)
3    t = Tail
4    x = t.next
5    if (t == Tail)
6      if (x == NULL)
7        if (CAS (t.next,NULL,n))
8          CAS (Tail,t,n)
9          exit
10     else
11       CAS (Tail,t,x)
```



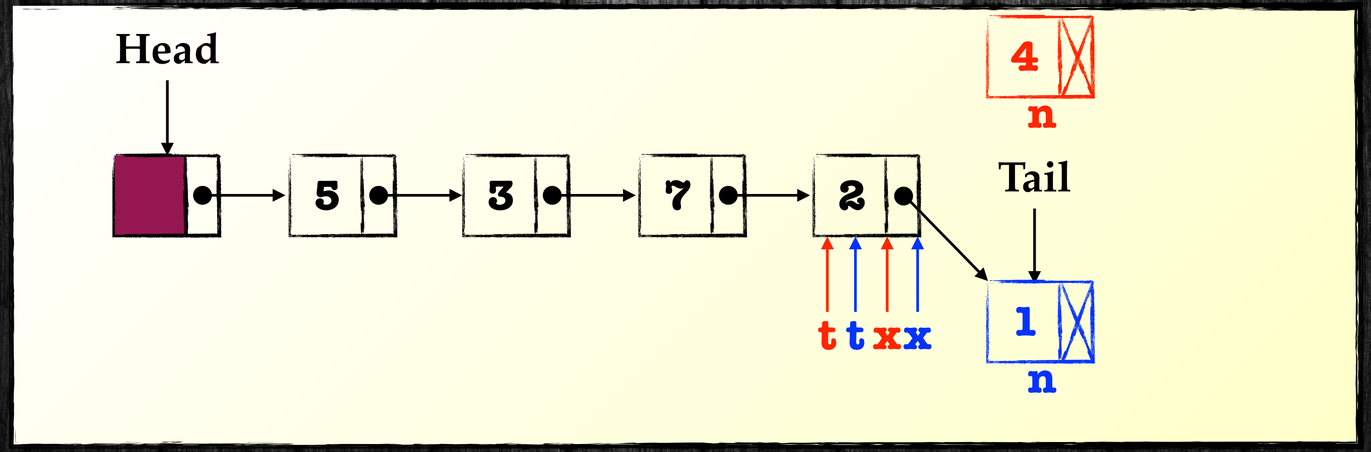
The Michael-Scott Queue Algorithm

Concurrent enqueue operations

```
enq(k):
Node t,x
1  n = new Node(k,NULL)
2  while (true)
3    t = Tail
4    x = t.next
5    if (t == Tail)
6      if (x == NULL)
7        if (CAS (t.next,NULL,n))
8          CAS (Tail,t,n)
9          exit
10     else
11       CAS (Tail,t,x)
```



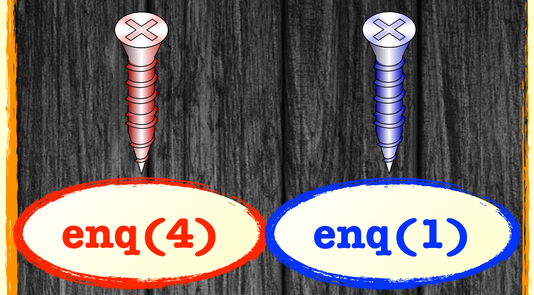

```
enq(k):
Node t,x
1  n = new Node(k,NULL)
2  while (true)
3    t = Tail
4    x = t.next
5    if (t == Tail)
6      if (x == NULL)
7        if (CAS (t.next,NULL,n))
8          CAS (Tail,t,n)
9          exit
10     else
11       CAS (Tail,t,x)
```



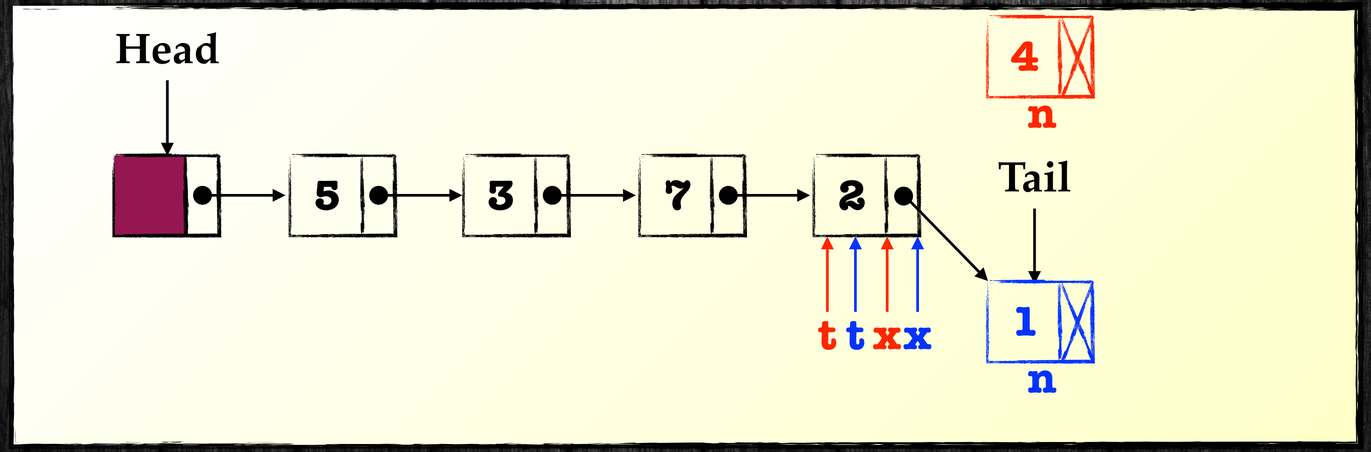
The Michael-Scott Queue Algorithm

Concurrent enqueue operations

```
enq(k):
Node t,x
1  n = new Node(k,NULL)
2  while (true)
3    t = Tail
4    x = t.next
5    if (t == Tail)
6      if (x == NULL)
7        if (CAS (t.next,NULL,n))
8          CAS (Tail,t,n)
9          exit
10     else
11       CAS (Tail,t,x)
```




```
enq(k):
Node t,x
1  n = new Node(k,NULL)
2  while (true)
3    t = Tail
4    x = t.next
5    if (t == Tail)
6      if (x == NULL)
7        if (CAS (t.next,NULL,n))
8          CAS (Tail,t,n)
9          exit
10     else
11       CAS (Tail,t,x)
```

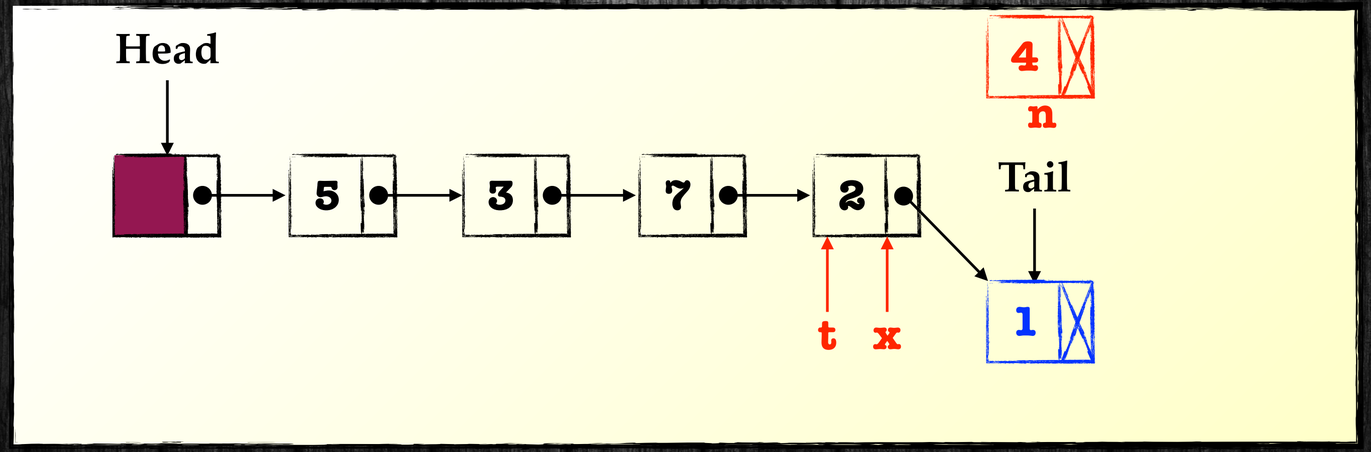


The Michael-Scott Queue Algorithm

Concurrent enqueue operations

```
enq(k):
Node t,x
 1  n = new Node(k,NULL)
 2  while (true)
 3    t = Tail
 4    x = t.next
 5    if (t == Tail)
 6      if (x == NULL)
 7        if (CAS (t.next,NULL,n))
 8          CAS (Tail,t,n)
 9          exit
10      else
11        CAS (Tail,t,x)
```

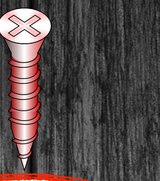




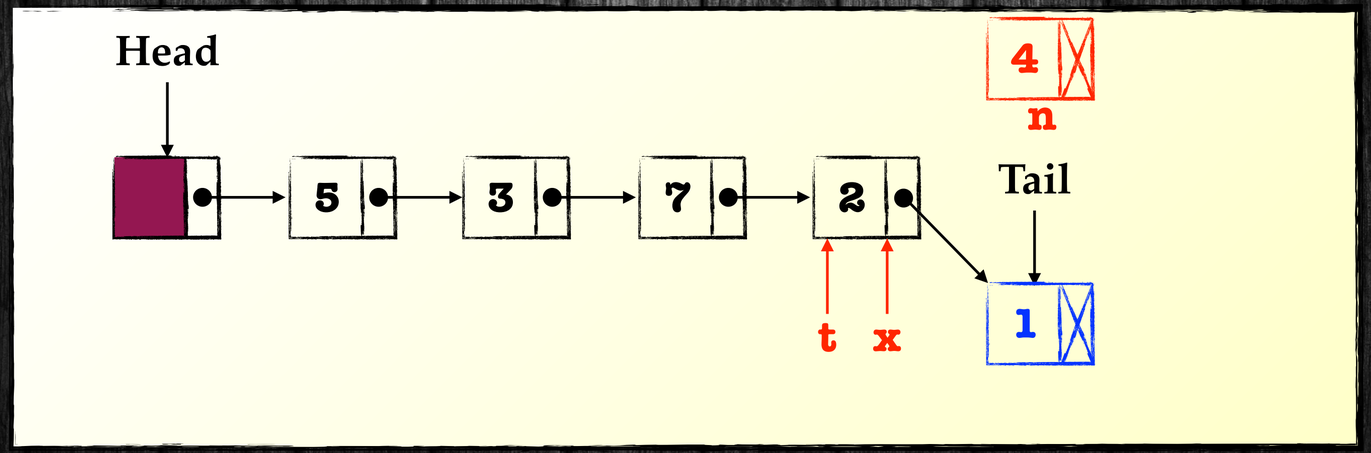
The Michael-Scott Queue Algorithm

Concurrent enqueue operations

```
enq(k):
Node t,x
 1  n = new Node(k,NULL)
 2  while (true)
 3    t = Tail
 4    x = t.next
 5    if (t == Tail)
 6      if (x == NULL)
 7        if (CAS (t.next,NULL,n))
 8          CAS (Tail,t,n)
 9          exit
10      else
11        CAS (Tail,t,x)
```



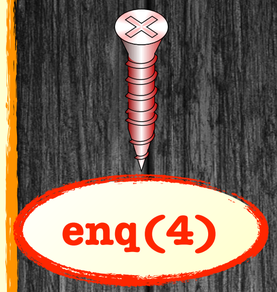
enq(4)

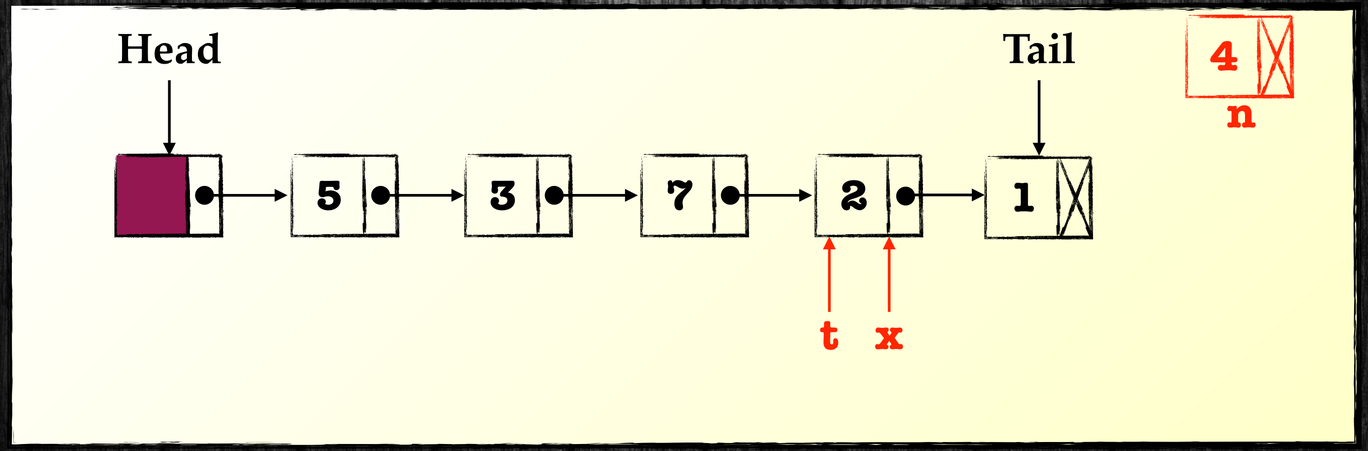


The Michael-Scott Queue Algorithm

Concurrent enqueue operations

```
enq(k):
Node t,x
 1  n = new Node(k,NULL)
 2  while (true)
 3    t = Tail
 4    x = t.next
 5    if (t == Tail)
 6      if (x == NULL)
 7        if (CAS (t.next,NULL,n))
 8          CAS (Tail,t,n)
 9          exit
10      else
11        CAS (Tail,t,x)
```

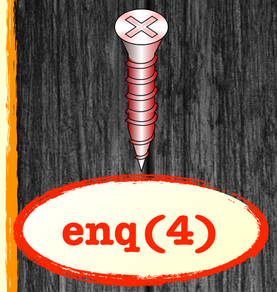


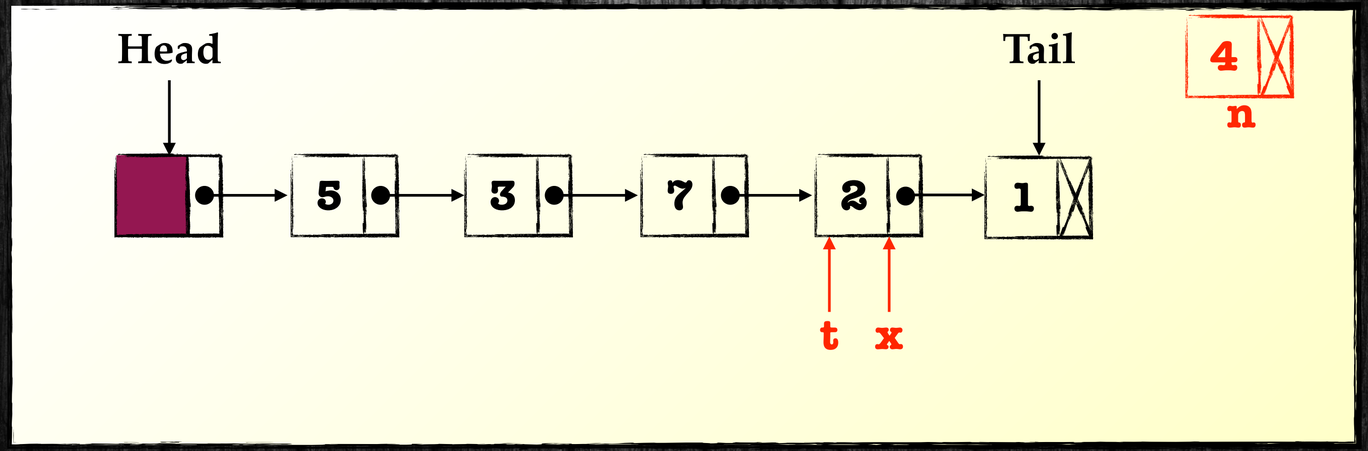


The Michael-Scott Queue Algorithm

Concurrent enqueue operations

```
enq(k):
Node t,x
 1  n = new Node(k,NULL)
 2  while (true)
 3    t = Tail
 4    x = t.next
 5    if (t == Tail)
 6      if (x == NULL)
 7        if (CAS (t.next,NULL,n))
 8          CAS (Tail,t,n)
 9          exit
10      else
11        CAS (Tail,t,x)
```



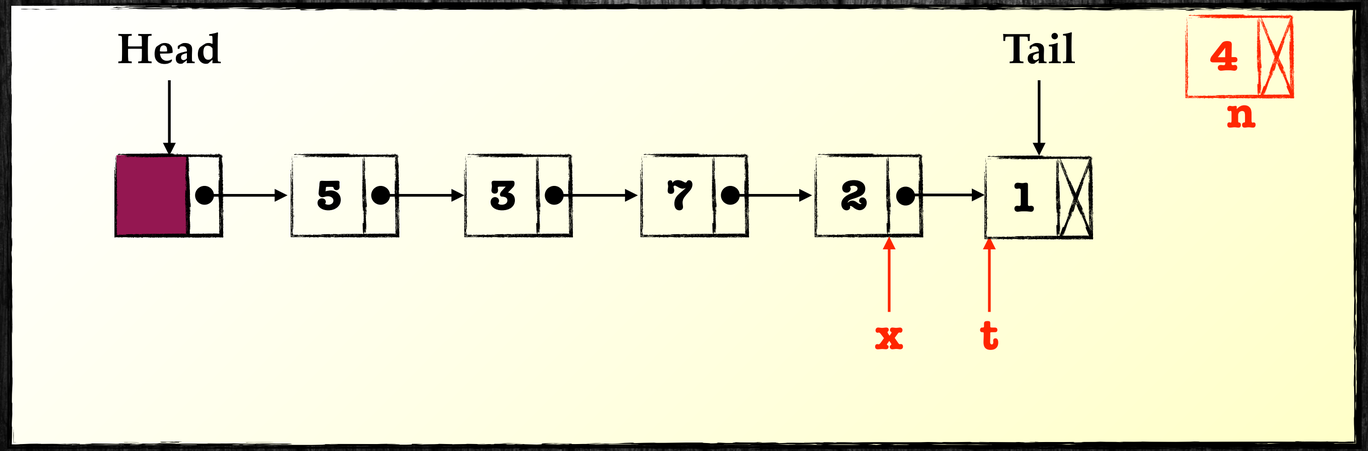


The Michael-Scott Queue Algorithm

Concurrent enqueue operations

```
enq(k):
Node t,x
 1  n = new Node(k,NULL)
 2  while (true)
 3    t = Tail
 4    x = t.next
 5    if (t == Tail)
 6      if (x == NULL)
 7        if (CAS (t.next,NULL,n))
 8          CAS (Tail,t,n)
 9          exit
10      else
11        CAS (Tail,t,x)
```



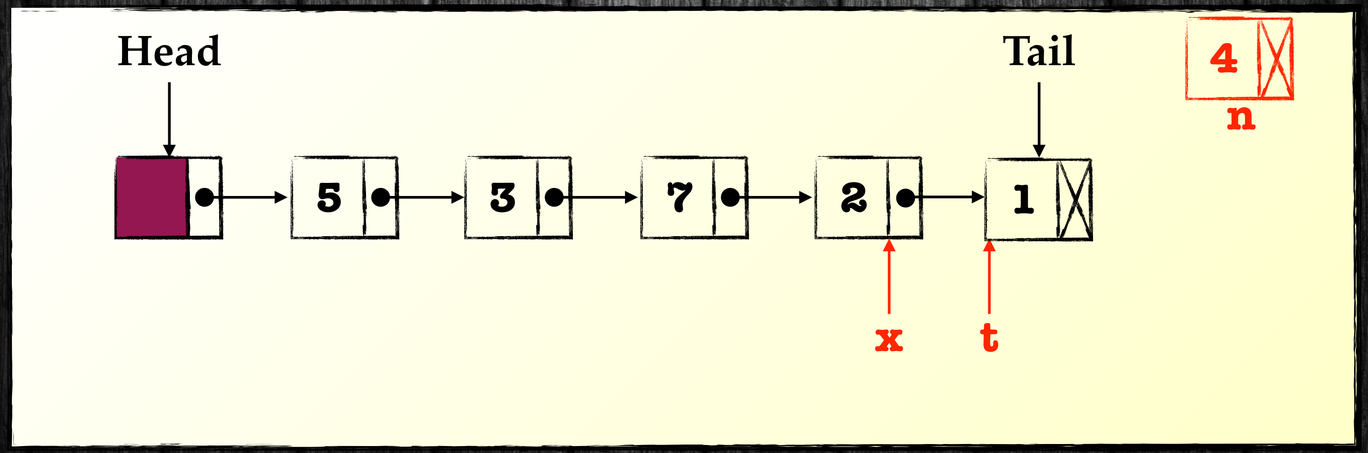


The Michael-Scott Queue Algorithm

Concurrent enqueue operations

```
enq(k):
Node t,x
 1  n = new Node(k,NULL)
 2  while (true)
 3    t = Tail
 4    x = t.next
 5    if (t == Tail)
 6      if (x == NULL)
 7        if (CAS (t.next,NULL,n))
 8          CAS (Tail,t,n)
 9          exit
10      else
11        CAS (Tail,t,x)
```

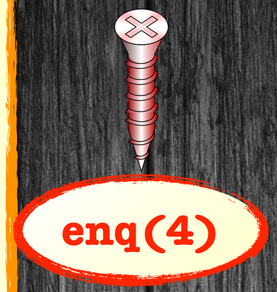


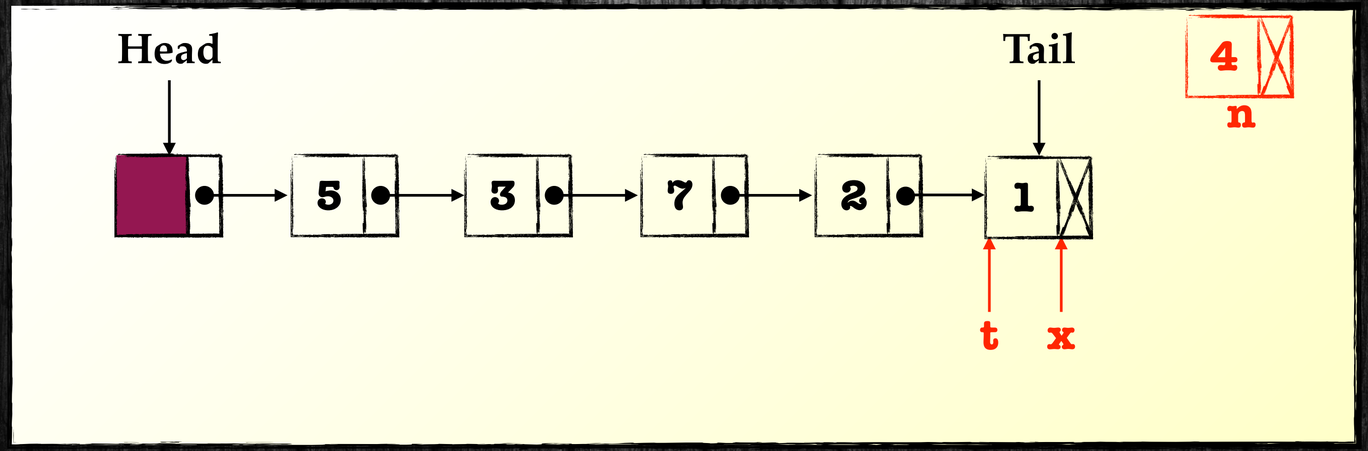


The Michael-Scott Queue Algorithm

Concurrent enqueue operations

```
enq(k):
Node t,x
 1  n = new Node(k,NULL)
 2  while (true)
 3    t = Tail
 4    x = t.next
 5    if (t == Tail)
 6      if (x == NULL)
 7        if (CAS (t.next,NULL,n))
 8          CAS (Tail,t,n)
 9          exit
10      else
11        CAS (Tail,t,x)
```



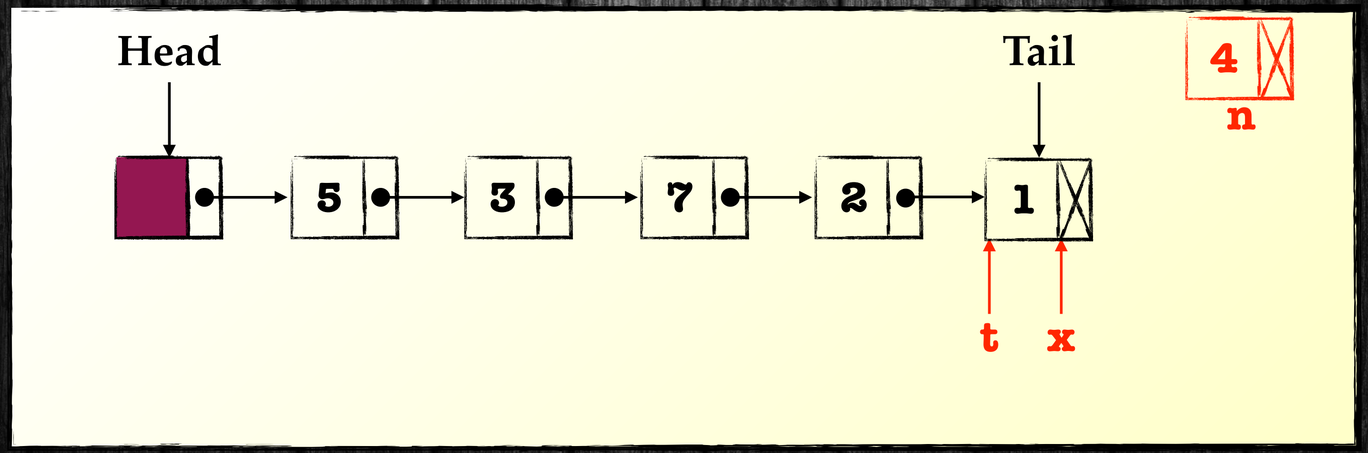


The Michael-Scott Queue Algorithm

Concurrent enqueue operations

```
enq(k):
Node t,x
 1  n = new Node(k,NULL)
 2  while (true)
 3    t = Tail
 4    x = t.next
 5    if (t == Tail)
 6      if (x == NULL)
 7        if (CAS (t.next,NULL,n))
 8          CAS (Tail,t,n)
 9          exit
10      else
11        CAS (Tail,t,x)
```



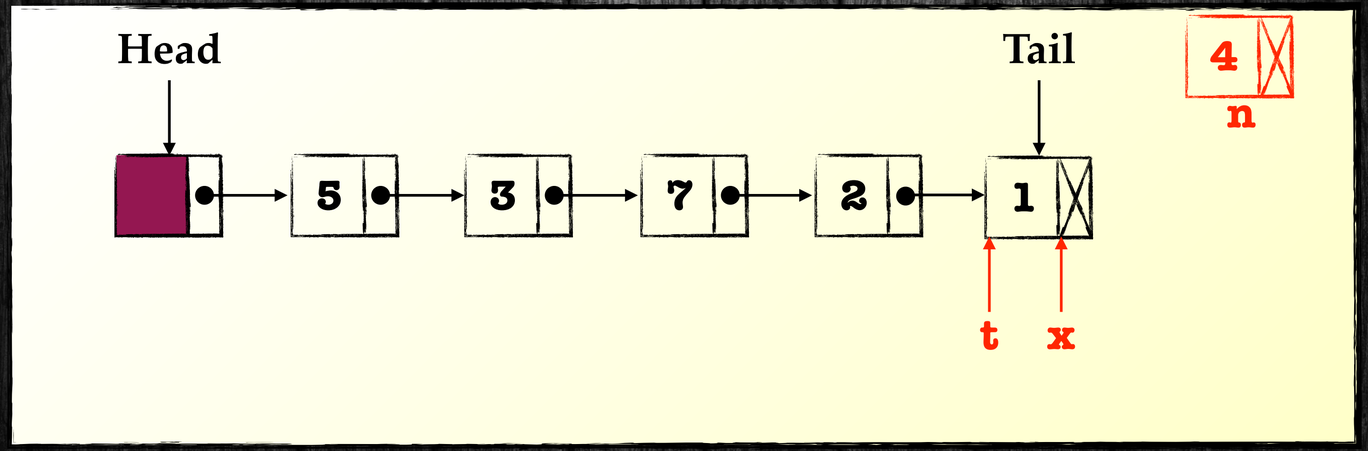


The Michael-Scott Queue Algorithm

Concurrent enqueue operations

```
enq(k):
Node t,x
 1  n = new Node(k,NULL)
 2  while (true)
 3    t = Tail
 4    x = t.next
 5    if (t == Tail)
 6      if (x == NULL)
 7        if (CAS (t.next,NULL,n))
 8          CAS (Tail,t,n)
 9          exit
10      else
11        CAS (Tail,t,x)
```

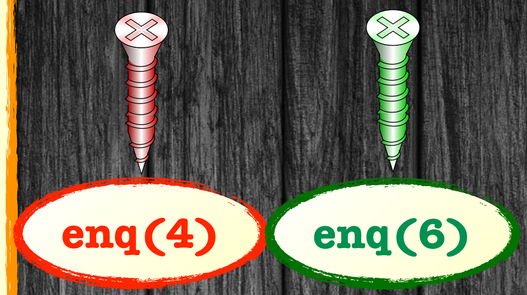


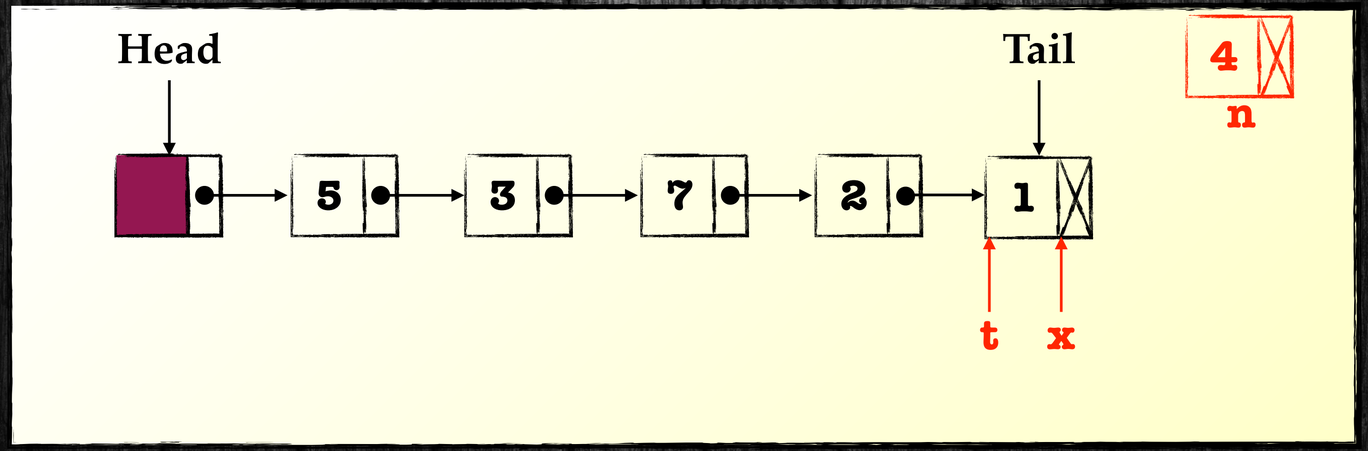


The Michael-Scott Queue Algorithm

Concurrent enqueue operations

```
enq(k):
Node t,x
 1  n = new Node(k,NULL)
 2  while (true)
 3    t = Tail
 4    x = t.next
 5    if (t == Tail)
 6      if (x == NULL)
 7        if (CAS (t.next,NULL,n))
 8          CAS (Tail,t,n)
 9          exit
10      else
11        CAS (Tail,t,x)
```

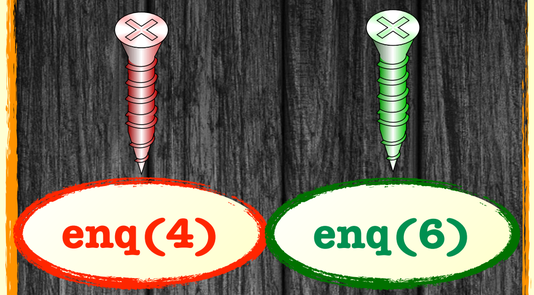




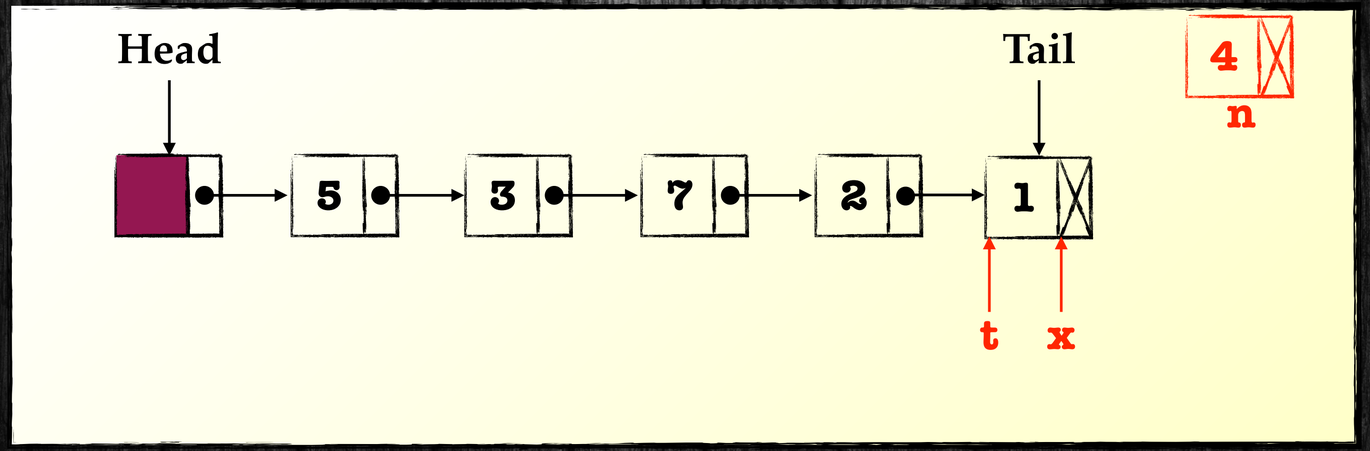
The Michael-Scott Queue Algorithm

Concurrent enqueue operations

```
enq(k):
Node t,x
1  n = new Node(k,NULL)
2  while (true)
3    t = Tail
4    x = t.next
5    if (t == Tail)
6      if (x == NULL)
7        if (CAS (t.next,NULL,n))
8          CAS (Tail,t,n)
9          exit
10     else
11       CAS (Tail,t,x)
```



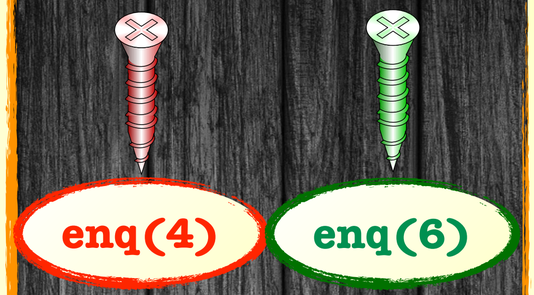

```
enq(k):
Node t,x
1  n = new Node(k,NULL)
2  while (true)
3    t = Tail
4    x = t.next
5    if (t == Tail)
6      if (x == NULL)
7        if (CAS (t.next,NULL,n))
8          CAS (Tail,t,n)
9          exit
10     else
11       CAS (Tail,t,x)
```



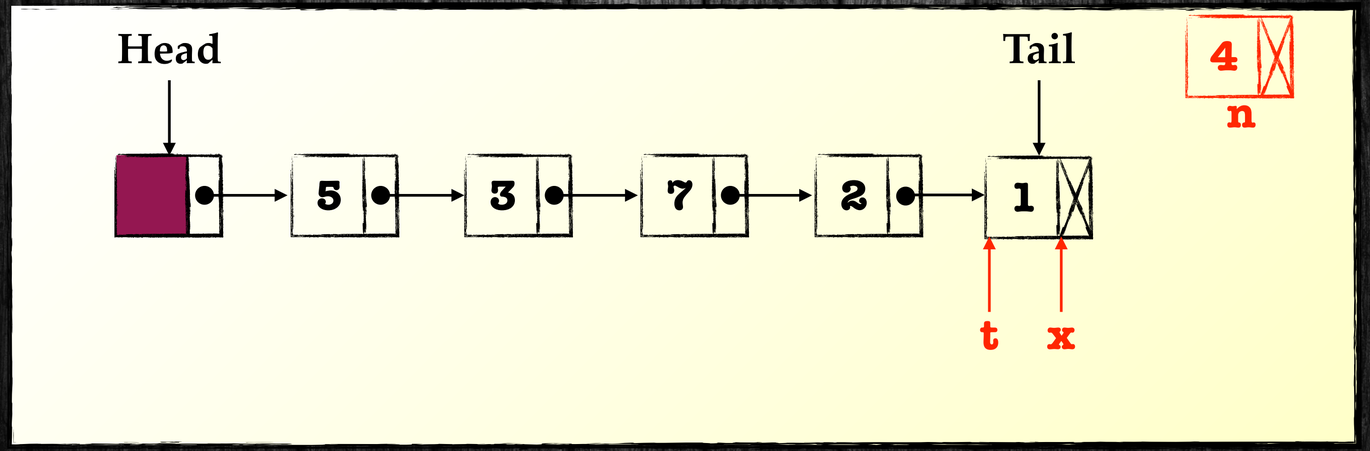
The Michael-Scott Queue Algorithm

Concurrent enqueue operations

```
enq(k):
Node t,x
1  n = new Node(k,NULL)
2  while (true)
3    t = Tail
4    x = t.next
5    if (t == Tail)
6      if (x == NULL)
7        if (CAS (t.next,NULL,n))
8          CAS (Tail,t,n)
9          exit
10     else
11       CAS (Tail,t,x)
```



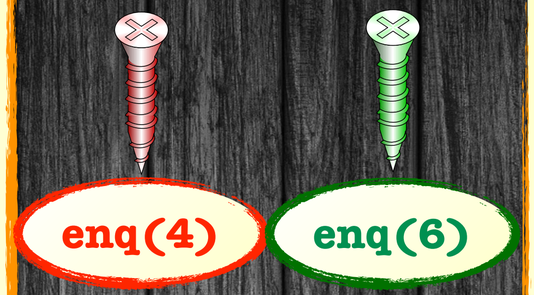

```
enq(k):
Node t,x
1  n = new Node(k,NULL)
2  while (true)
3    t = Tail
4    x = t.next
5    if (t == Tail)
6      if (x == NULL)
7        if (CAS (t.next,NULL,n))
8          CAS (Tail,t,n)
9          exit
10     else
11       CAS (Tail,t,x)
```



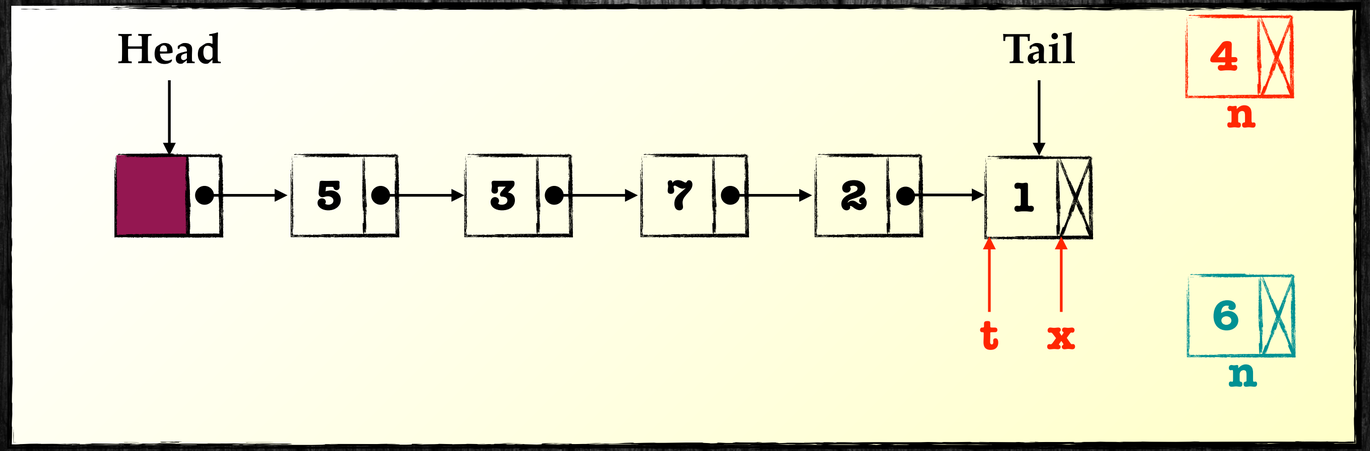
The Michael-Scott Queue Algorithm

Concurrent enqueue operations

```
enq(k):
Node t,x
1  n = new Node(k,NULL)
2  while (true)
3    t = Tail
4    x = t.next
5    if (t == Tail)
6      if (x == NULL)
7        if (CAS (t.next,NULL,n))
8          CAS (Tail,t,n)
9          exit
10     else
11       CAS (Tail,t,x)
```



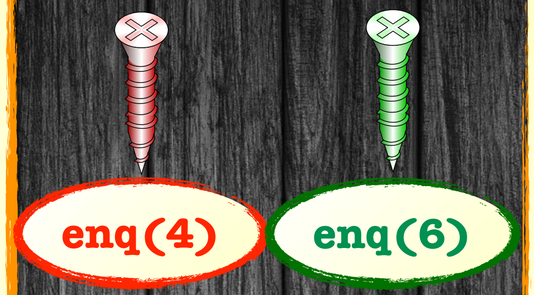

```
enq(k):
Node t,x
1  n = new Node(k,NULL)
2  while (true)
3    t = Tail
4    x = t.next
5    if (t == Tail)
6      if (x == NULL)
7        if (CAS (t.next,NULL,n))
8          CAS (Tail,t,n)
9          exit
10     else
11       CAS (Tail,t,x)
```



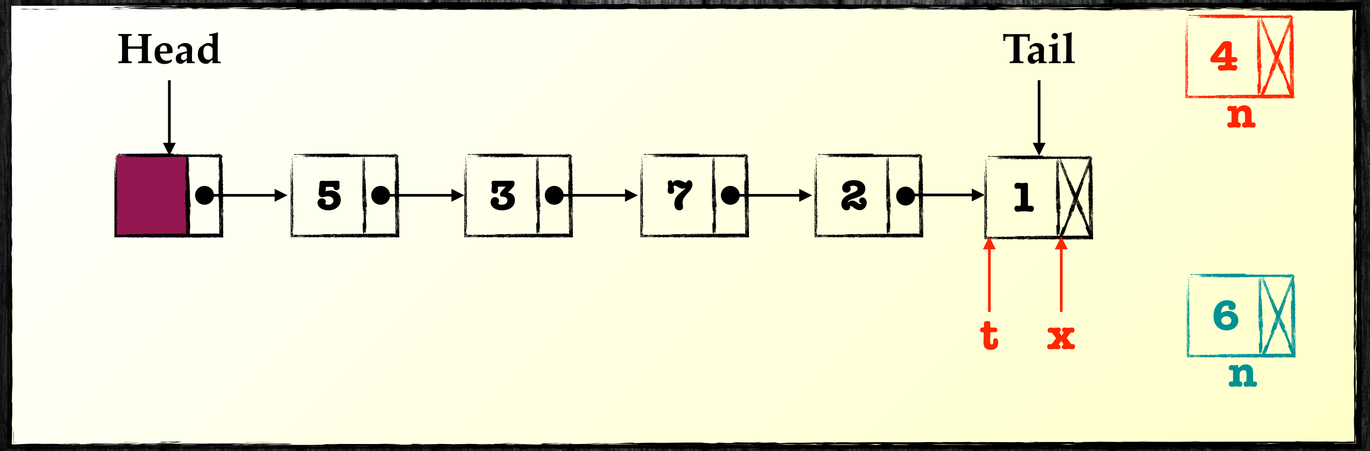
The Michael-Scott Queue Algorithm

Concurrent enqueue operations

```
enq(k):
Node t,x
1  n = new Node(k,NULL)
2  while (true)
3    t = Tail
4    x = t.next
5    if (t == Tail)
6      if (x == NULL)
7        if (CAS (t.next,NULL,n))
8          CAS (Tail,t,n)
9          exit
10     else
11       CAS (Tail,t,x)
```



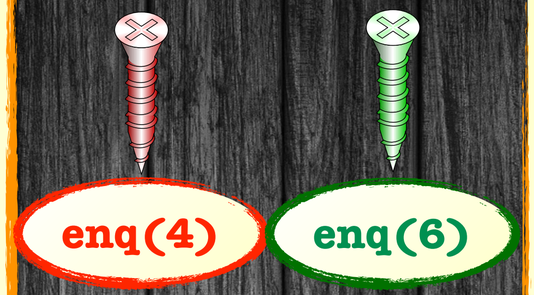

```
enq(k):
Node t,x
1  n = new Node(k,NULL)
2  while (true)
3    t = Tail
4    x = t.next
5    if (t == Tail)
6      if (x == NULL)
7        if (CAS (t.next,NULL,n))
8          CAS (Tail,t,n)
9          exit
10     else
11       CAS (Tail,t,x)
```



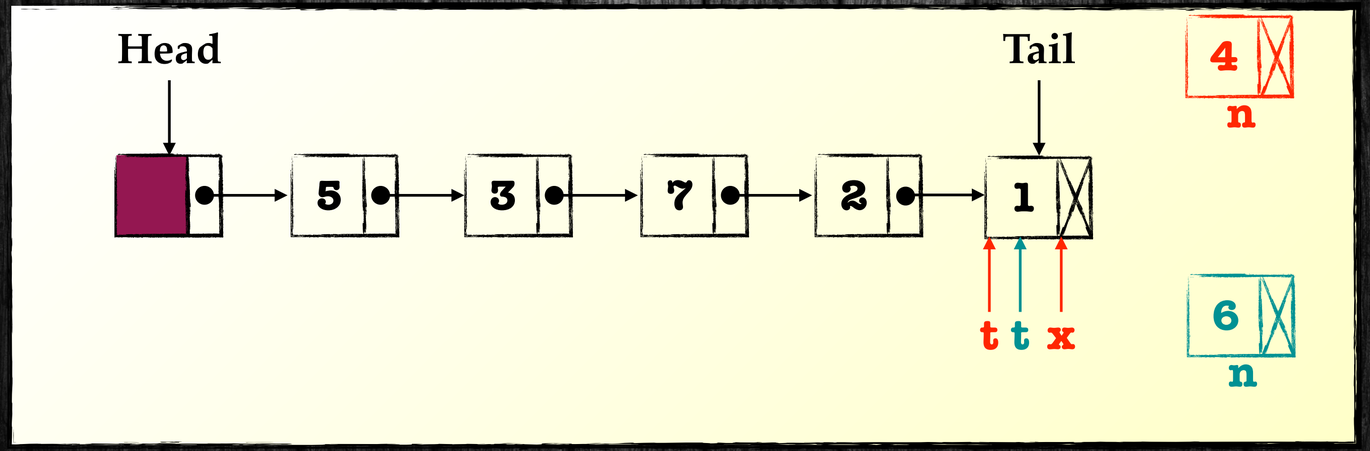
The Michael-Scott Queue Algorithm

Concurrent enqueue operations

```
enq(k):
Node t,x
1  n = new Node(k,NULL)
2  while (true)
3    t = Tail
4    x = t.next
5    if (t == Tail)
6      if (x == NULL)
7        if (CAS (t.next,NULL,n))
8          CAS (Tail,t,n)
9          exit
10     else
11       CAS (Tail,t,x)
```



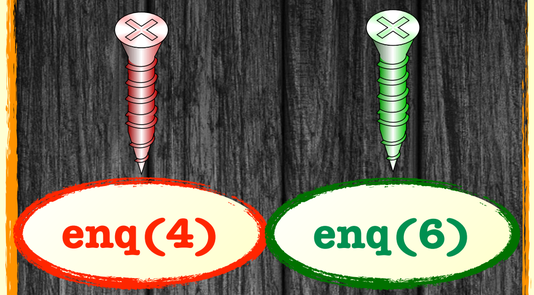

```
enq(k):
Node t,x
1  n = new Node(k,NULL)
2  while (true)
3    t = Tail
4    x = t.next
5    if (t == Tail)
6      if (x == NULL)
7        if (CAS (t.next,NULL,n))
8          CAS (Tail,t,n)
9          exit
10     else
11       CAS (Tail,t,x)
```



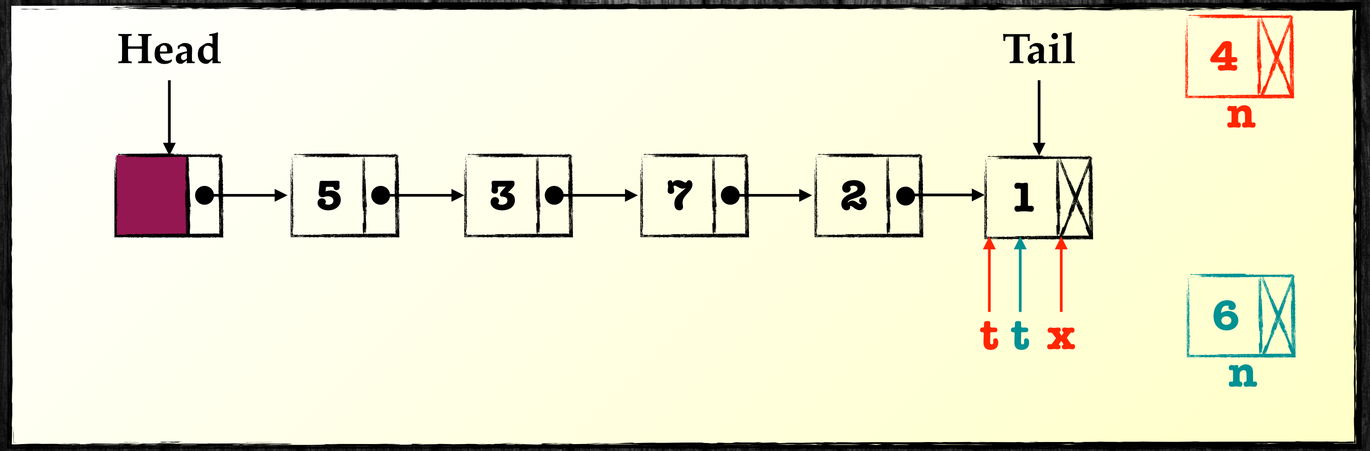
The Michael-Scott Queue Algorithm

Concurrent enqueue operations

```
enq(k):
Node t,x
1  n = new Node(k,NULL)
2  while (true)
3    t = Tail
4    x = t.next
5    if (t == Tail)
6      if (x == NULL)
7        if (CAS (t.next,NULL,n))
8          CAS (Tail,t,n)
9          exit
10     else
11       CAS (Tail,t,x)
```



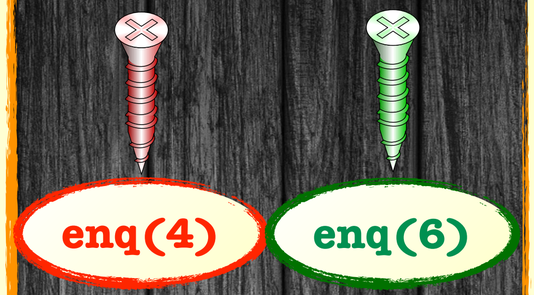

```
enq(k):
Node t,x
1  n = new Node(k,NULL)
2  while (true)
3    t = Tail
4    x = t.next
5    if (t == Tail)
6      if (x == NULL)
7        if (CAS (t.next,NULL,n))
8          CAS (Tail,t,n)
9          exit
10     else
11       CAS (Tail,t,x)
```



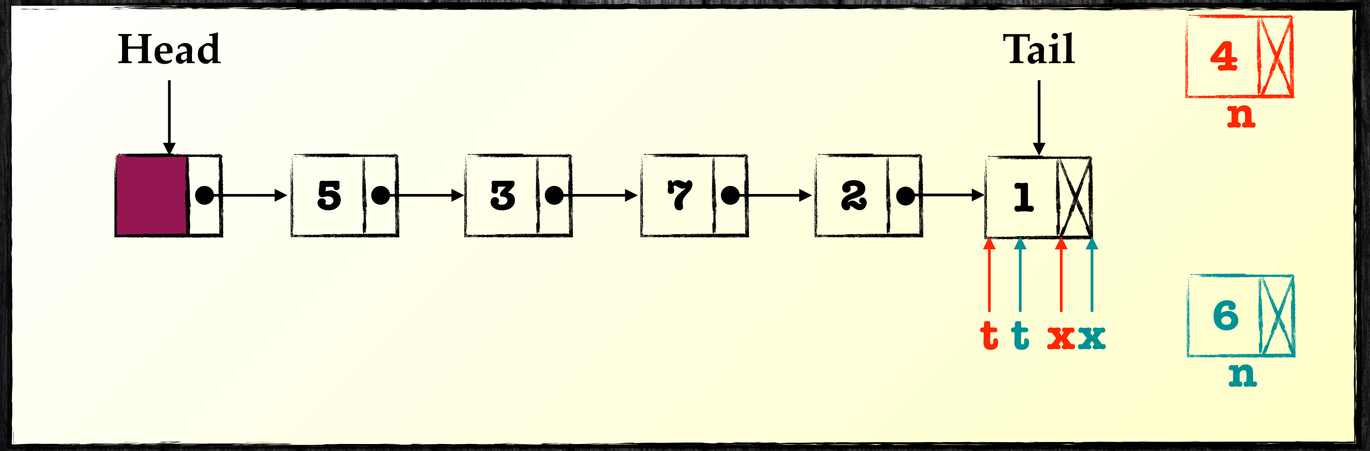
The Michael-Scott Queue Algorithm

Concurrent enqueue operations

```
enq(k):
Node t,x
1  n = new Node(k,NULL)
2  while (true)
3    t = Tail
4    x = t.next
5    if (t == Tail)
6      if (x == NULL)
7        if (CAS (t.next,NULL,n))
8          CAS (Tail,t,n)
9          exit
10     else
11       CAS (Tail,t,x)
```



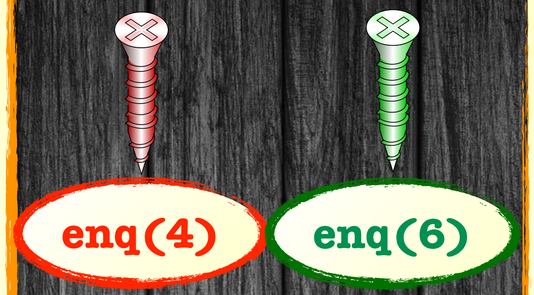

```
enq(k):
Node t,x
1  n = new Node(k,NULL)
2  while (true)
3    t = Tail
4    x = t.next
5    if (t == Tail)
6      if (x == NULL)
7        if (CAS (t.next,NULL,n))
8          CAS (Tail,t,n)
9          exit
10     else
11       CAS (Tail,t,x)
```



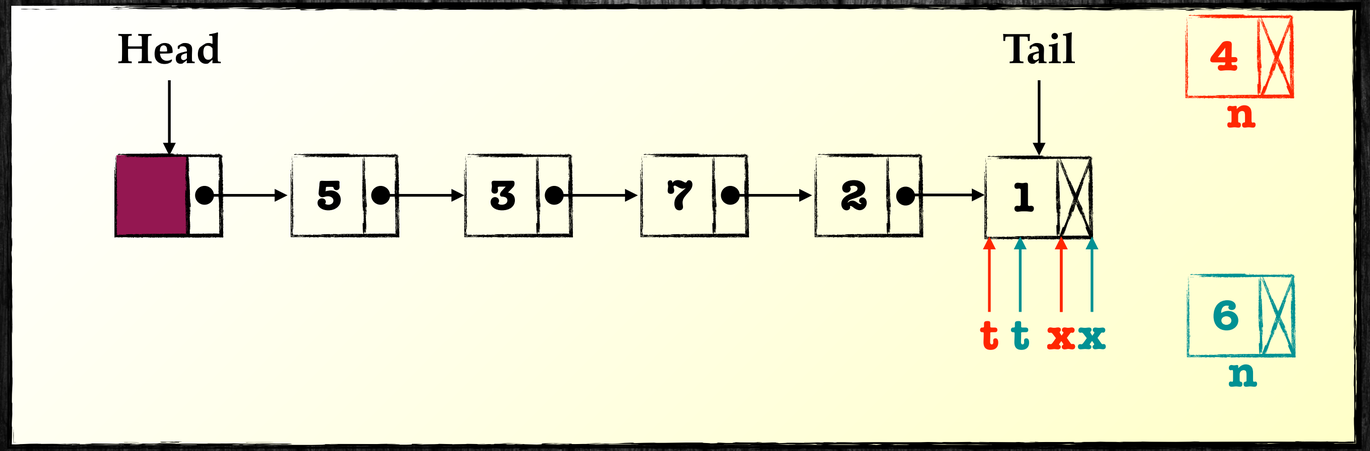
The Michael-Scott Queue Algorithm

Concurrent enqueue operations

```
enq(k):
Node t,x
1  n = new Node(k,NULL)
2  while (true)
3    t = Tail
4    x = t.next
5    if (t == Tail)
6      if (x == NULL)
7        if (CAS (t.next,NULL,n))
8          CAS (Tail,t,n)
9          exit
10     else
11       CAS (Tail,t,x)
```



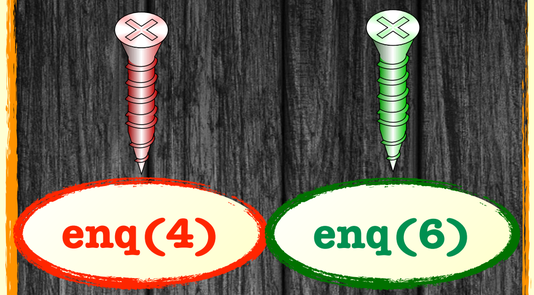

```
enq(k):
Node t,x
1  n = new Node(k,NULL)
2  while (true)
3    t = Tail
4    x = t.next
5    if (t == Tail)
6      if (x == NULL)
7        if (CAS (t.next,NULL,n))
8          CAS (Tail,t,n)
9          exit
10     else
11       CAS (Tail,t,x)
```



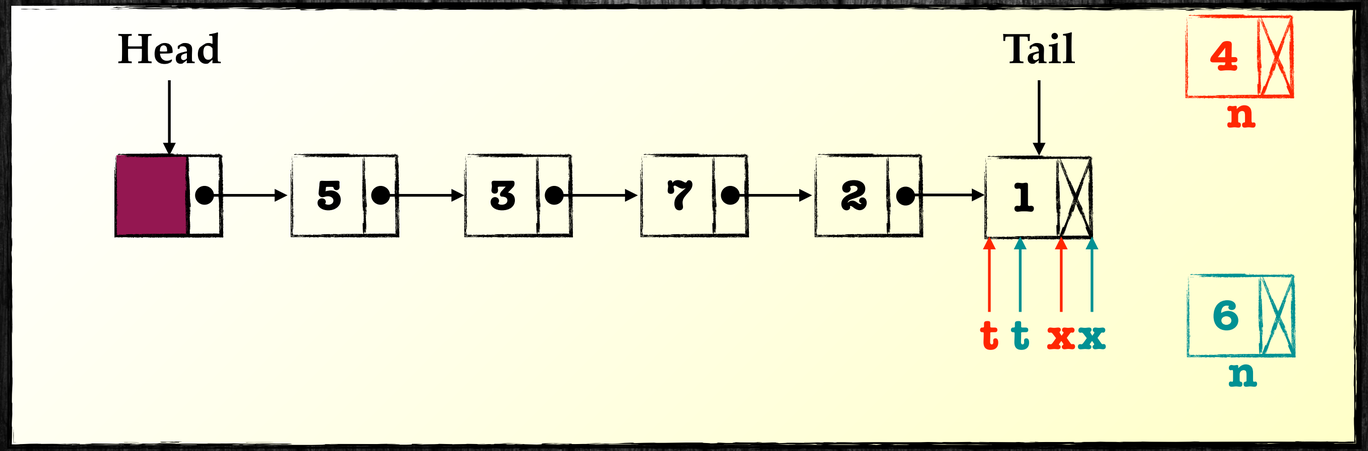
The Michael-Scott Queue Algorithm

Concurrent enqueue operations

```
enq(k):
Node t,x
1  n = new Node(k,NULL)
2  while (true)
3    t = Tail
4    x = t.next
5    if (t == Tail)
6      if (x == NULL)
7        if (CAS (t.next,NULL,n))
8          CAS (Tail,t,n)
9          exit
10     else
11       CAS (Tail,t,x)
```



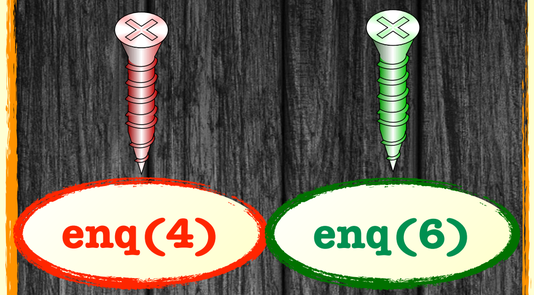

```
enq(k):
Node t,x
1  n = new Node(k,NULL)
2  while (true)
3    t = Tail
4    x = t.next
5    if (t == Tail)
6      if (x == NULL)
7        if (CAS (t.next,NULL,n))
8          CAS (Tail,t,n)
9          exit
10     else
11       CAS (Tail,t,x)
```

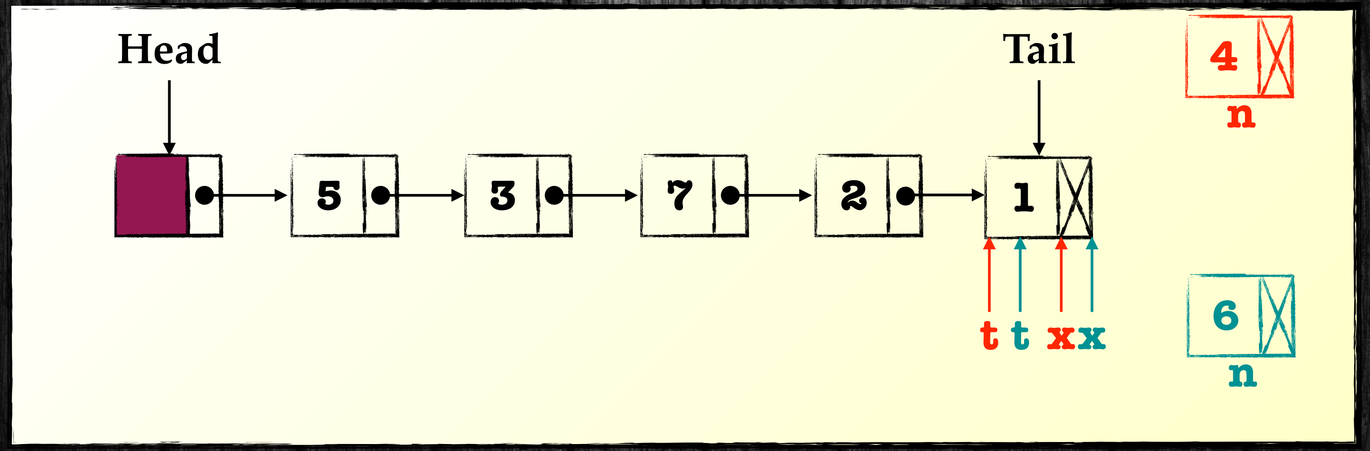
The Michael-Scott Queue Algorithm

Concurrent enqueue operations

```
enq(k):
Node t,x
1  n = new Node(k,NULL)
2  while (true)
3    t = Tail
4    x = t.next
5    if (t == Tail)
6      if (x == NULL)
7        if (CAS (t.next,NULL,n))
8          CAS (Tail,t,n)
9          exit
10     else
11       CAS (Tail,t,x)
```



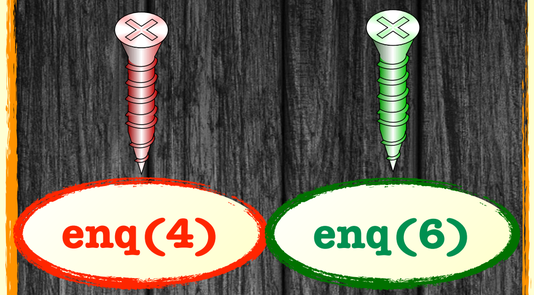

```
enq(k):
Node t,x
1  n = new Node(k,NULL)
2  while (true)
3    t = Tail
4    x = t.next
5    if (t == Tail)
6      if (x == NULL)
7        if (CAS (t.next,NULL,n))
8          CAS (Tail,t,n)
9          exit
10     else
11       CAS (Tail,t,x)
```



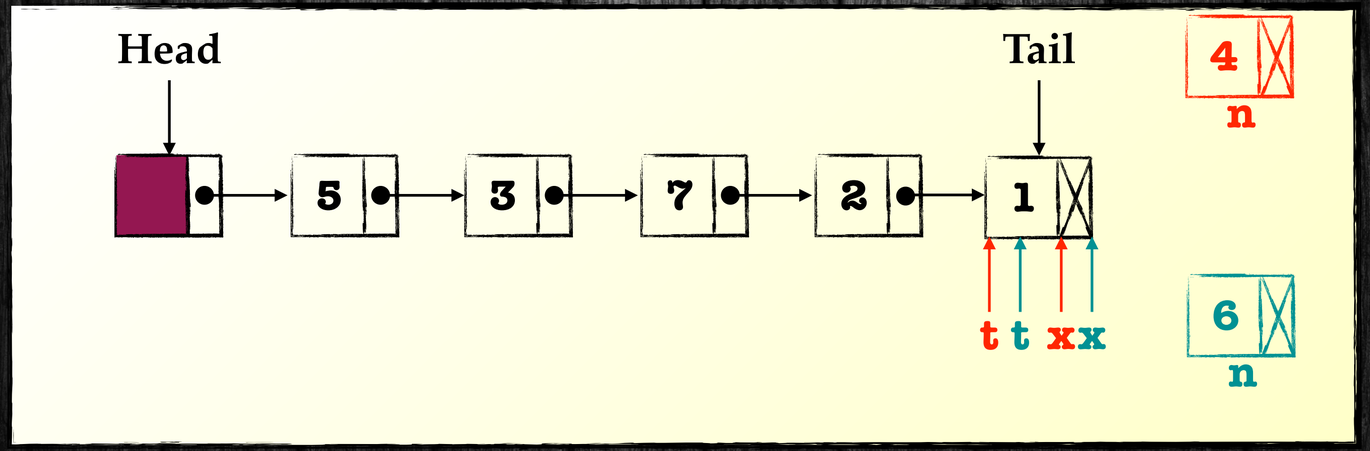
The Michael-Scott Queue Algorithm

Concurrent enqueue operations

```
enq(k):
Node t,x
1  n = new Node(k,NULL)
2  while (true)
3    t = Tail
4    x = t.next
5    if (t == Tail)
6      if (x == NULL)
7        if (CAS (t.next,NULL,n))
8          CAS (Tail,t,n)
9          exit
10     else
11       CAS (Tail,t,x)
```



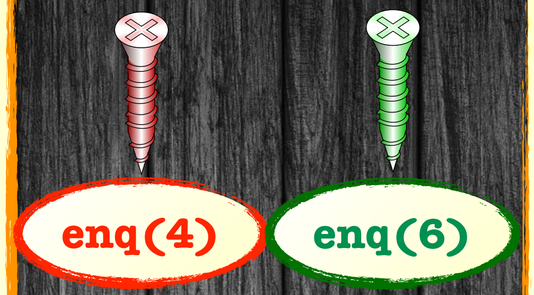

```
enq(k):
Node t,x
1  n = new Node(k,NULL)
2  while (true)
3    t = Tail
4    x = t.next
5    if (t == Tail)
6      if (x == NULL)
7        if (CAS (t.next,NULL,n))
8          CAS (Tail,t,n)
9          exit
10     else
11       CAS (Tail,t,x)
```



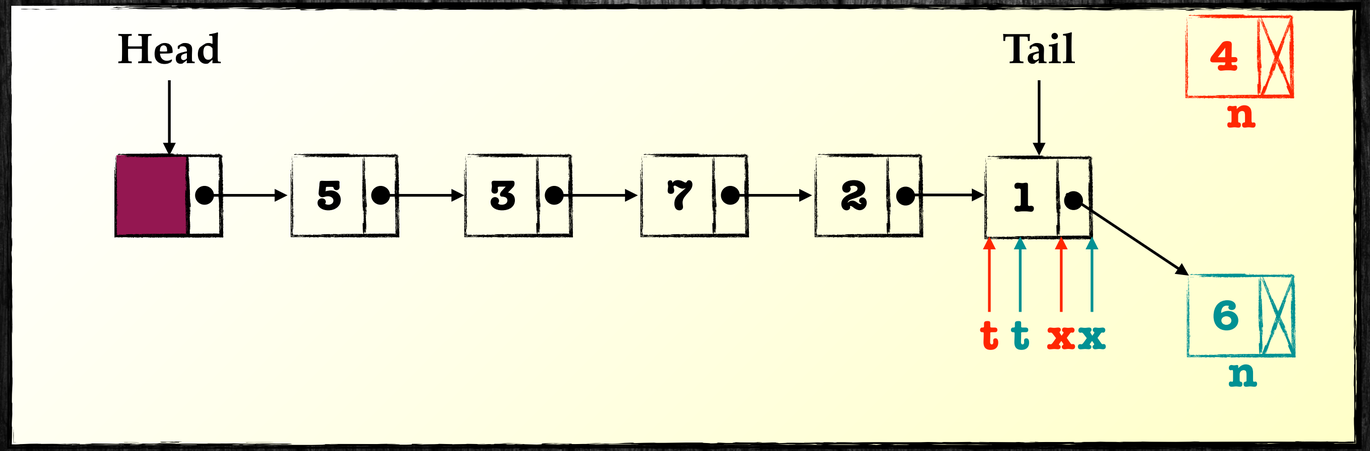
The Michael-Scott Queue Algorithm

Concurrent enqueue operations

```
enq(k):
Node t,x
1  n = new Node(k,NULL)
2  while (true)
3    t = Tail
4    x = t.next
5    if (t == Tail)
6      if (x == NULL)
7        if (CAS (t.next,NULL,n))
8          CAS (Tail,t,n)
9          exit
10     else
11       CAS (Tail,t,x)
```



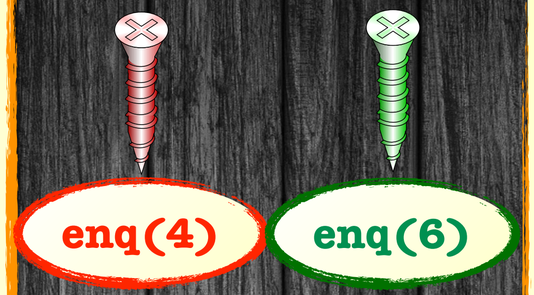

```
enq(k):
Node t,x
1  n = new Node(k,NULL)
2  while (true)
3    t = Tail
4    x = t.next
5    if (t == Tail)
6      if (x == NULL)
7        if (CAS (t.next,NULL,n))
8          CAS (Tail,t,n)
9          exit
10     else
11       CAS (Tail,t,x)
```



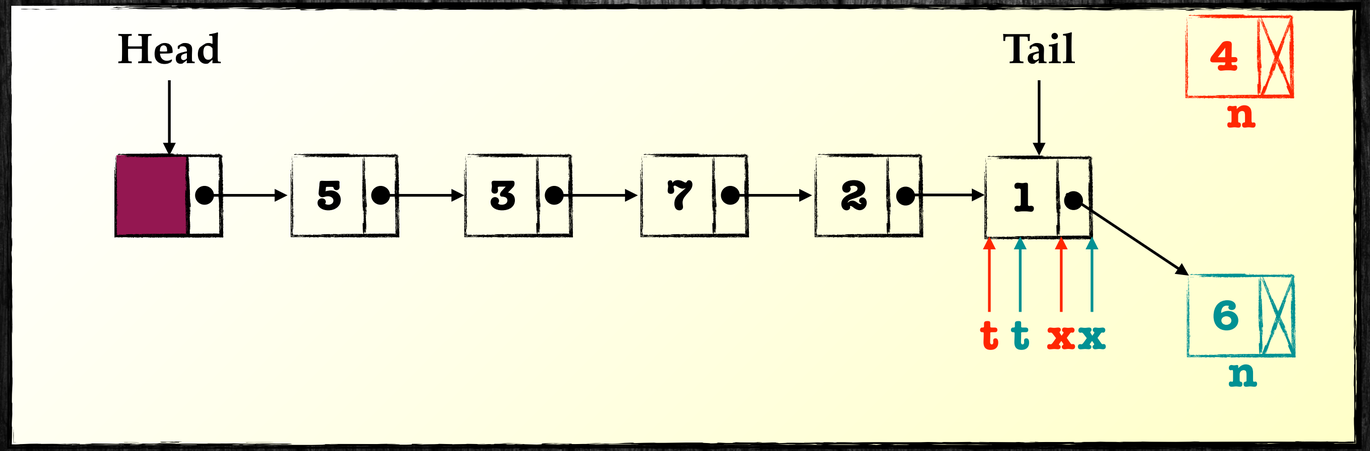
The Michael-Scott Queue Algorithm

Concurrent enqueue operations

```
enq(k):
Node t,x
1  n = new Node(k,NULL)
2  while (true)
3    t = Tail
4    x = t.next
5    if (t == Tail)
6      if (x == NULL)
7        if (CAS (t.next,NULL,n))
8          CAS (Tail,t,n)
9          exit
10     else
11       CAS (Tail,t,x)
```



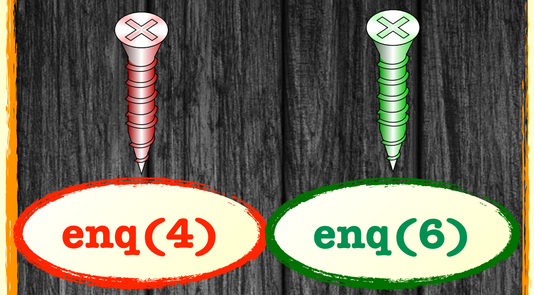

```
enq(k):
Node t,x
1  n = new Node(k,NULL)
2  while (true)
3    t = Tail
4    x = t.next
5    if (t == Tail)
6      if (x == NULL)
7        if (CAS (t.next,NULL,n))
8          CAS (Tail,t,n)
9          exit
10     else
11       CAS (Tail,t,x)
```



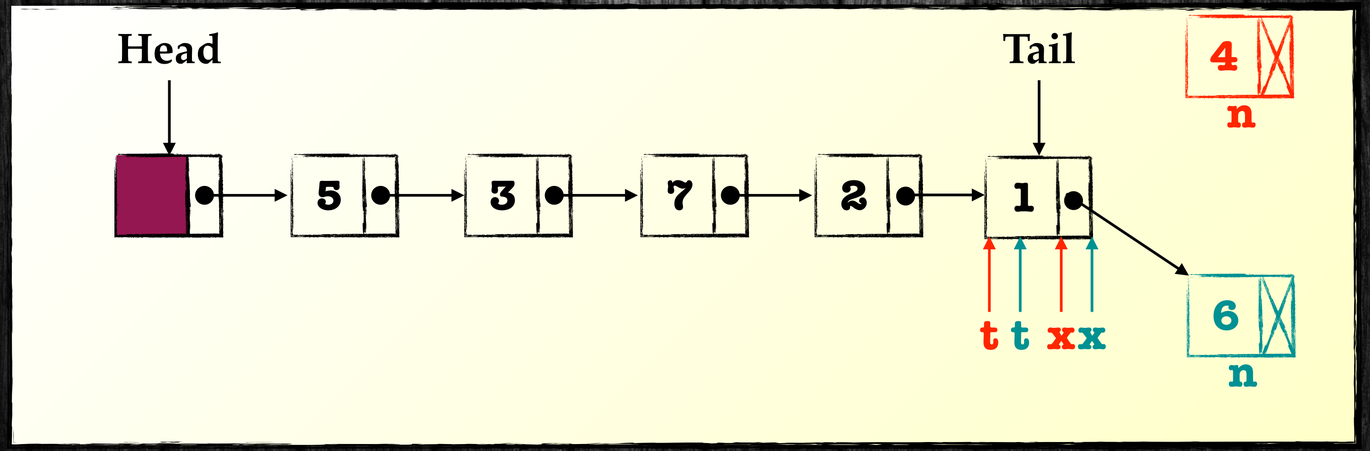
The Michael-Scott Queue Algorithm

Concurrent enqueue operations

```
enq(k):
Node t,x
1  n = new Node(k,NULL)
2  while (true)
3    t = Tail
4    x = t.next
5    if (t == Tail)
6      if (x == NULL)
7        if (CAS (t.next,NULL,n))
8          CAS (Tail,t,n)
9          exit
10     else
11       CAS (Tail,t,x)
```



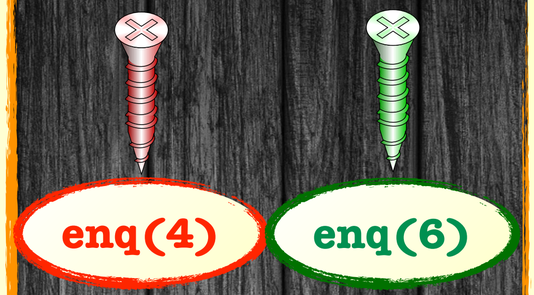

```
enq(k):
Node t,x
1  n = new Node(k,NULL)
2  while (true)
3    t = Tail
4    x = t.next
5    if (t == Tail)
6      if (x == NULL)
7        if (CAS (t.next,NULL,n))
8          CAS (Tail,t,n)
9          exit
10     else
11       CAS (Tail,t,x)
```



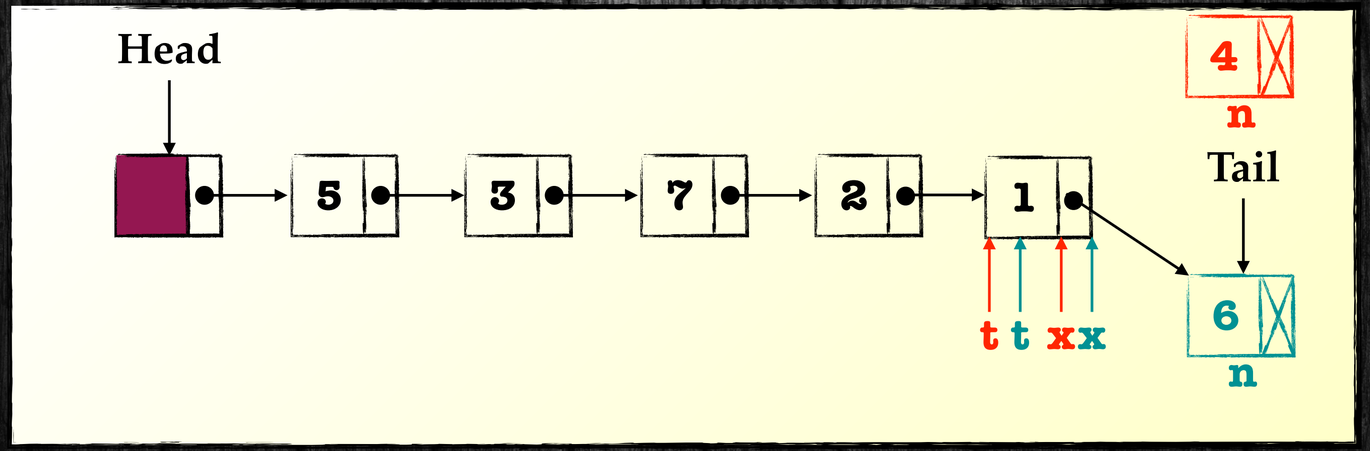
The Michael-Scott Queue Algorithm

Concurrent enqueue operations

```
enq(k):
Node t,x
1  n = new Node(k,NULL)
2  while (true)
3    t = Tail
4    x = t.next
5    if (t == Tail)
6      if (x == NULL)
7        if (CAS (t.next,NULL,n))
8          CAS (Tail,t,n)
9          exit
10     else
11       CAS (Tail,t,x)
```



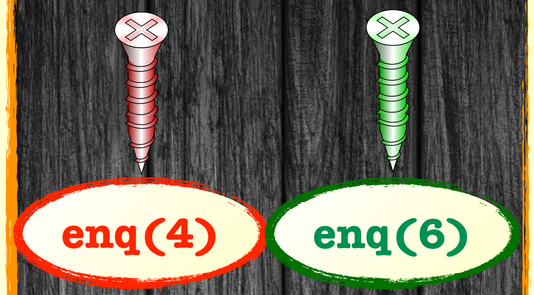

```
enq(k):
Node t,x
1  n = new Node(k,NULL)
2  while (true)
3    t = Tail
4    x = t.next
5    if (t == Tail)
6      if (x == NULL)
7        if (CAS (t.next,NULL,n))
8          CAS (Tail,t,n)
9          exit
10     else
11       CAS (Tail,t,x)
```



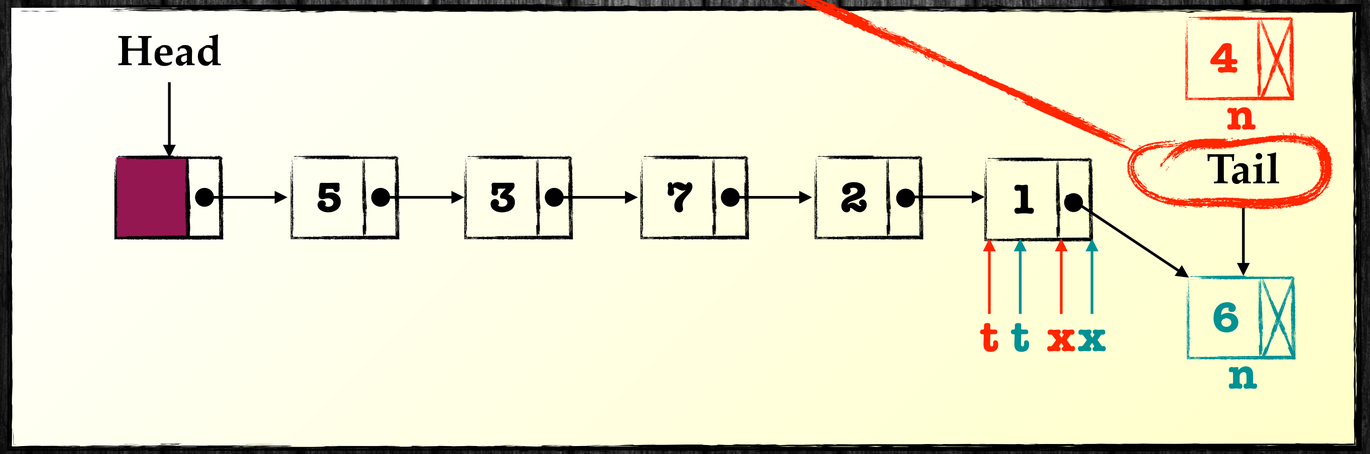
The Michael-Scott Queue Algorithm

Concurrent enqueue operations

```
enq(k):
Node t,x
1  n = new Node(k,NULL)
2  while (true)
3    t = Tail
4    x = t.next
5    if (t == Tail)
6      if (x == NULL)
7        if (CAS (t.next,NULL,n))
8          CAS (Tail,t,n)
9          exit
10     else
11       CAS (Tail,t,x)
```



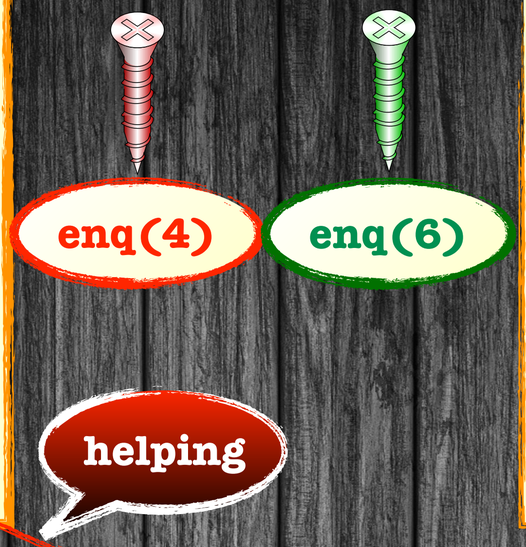

```
enq(k):
Node t,x
1  n = new Node(k,NULL)
2  while (true)
3    t = Tail
4    x = t.next
5    if (t == Tail)
6      if (x == NULL)
7        if (CAS (t.next,NULL,n))
8          CAS (Tail,t,n)
9          exit
10     else
11       CAS (Tail,t,x)
```



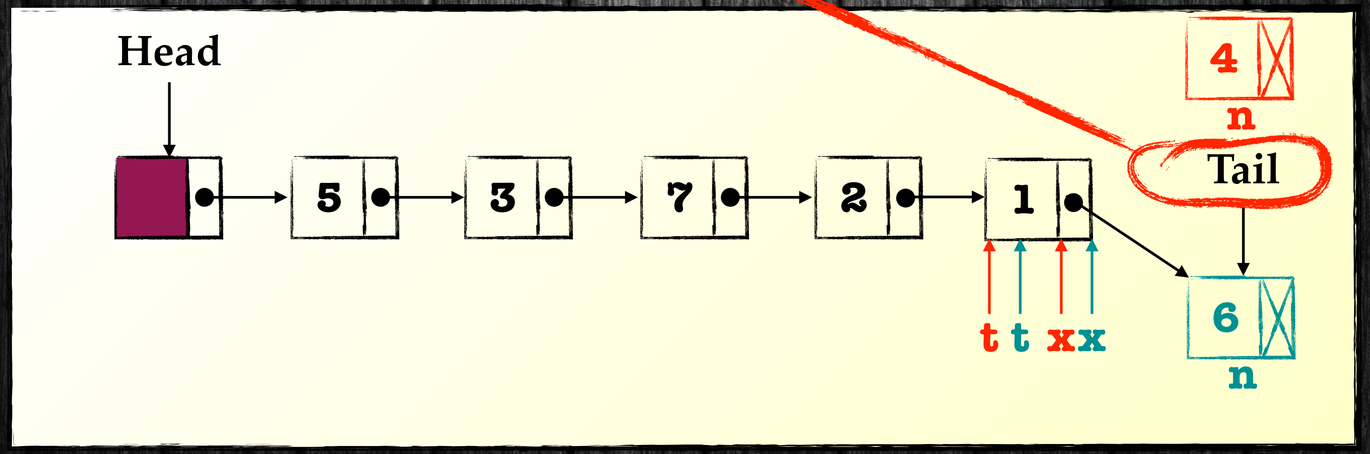
The Michael-Scott Queue Algorithm

Concurrent enqueue operations

```
enq(k):
Node t,x
1  n = new Node(k,NULL)
2  while (true)
3    t = Tail
4    x = t.next
5    if (t == Tail)
6      if (x == NULL)
7        if (CAS (t.next,NULL,n))
8          CAS (Tail,t,n)
9          exit
10     else
11       CAS (Tail,t,x)
```



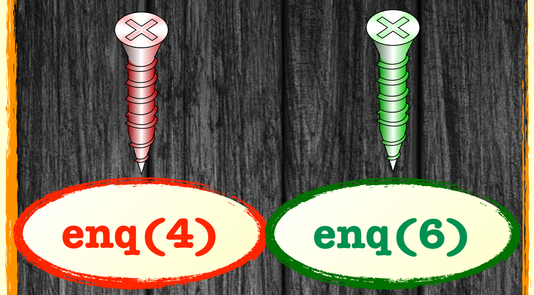

```
enq(k):
Node t,x
1  n = new Node(k,NULL)
2  while (true)
3    t = Tail
4    x = t.next
5    if (t == Tail)
6      if (x == NULL)
7        if (CAS (t.next,NULL,n))
8          CAS (Tail,t,n)
9          exit
10     else
11       CAS (Tail,t,x)
```



The Michael-Scott Queue Algorithm

Concurrent enqueue operations

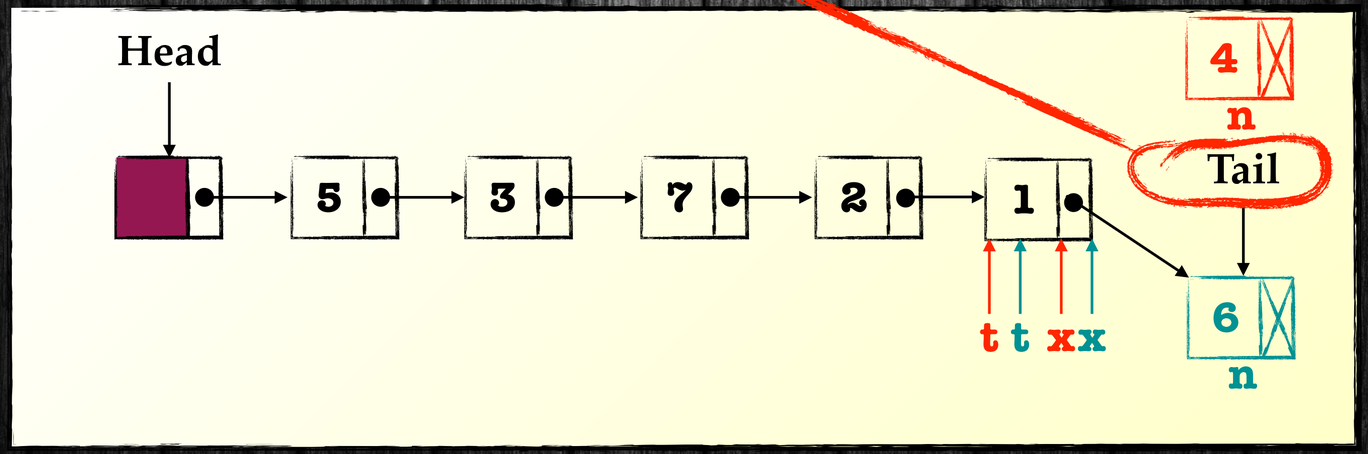
```
enq(k):
Node t,x
1  n = new Node(k,NULL)
2  while (true)
3    t = Tail
4    x = t.next
5    if (t == Tail)
6      if (x == NULL)
7        if (CAS (t.next,NULL,n))
8          CAS (Tail,t,n)
9          exit
10     else
11       CAS (Tail,t,x)
```




```
enq(k):
Node t,x
1  n = new Node(k,NULL)
2  while (true)
3    t = Tail
4    x = t.next
5    if (t == Tail)
6      if (x == NULL)
7        if (CAS (t.next,NULL,n))
8          CAS (Tail,t,n)
9          exit
10     else
11       CAS (Tail,t,x)
```

helping

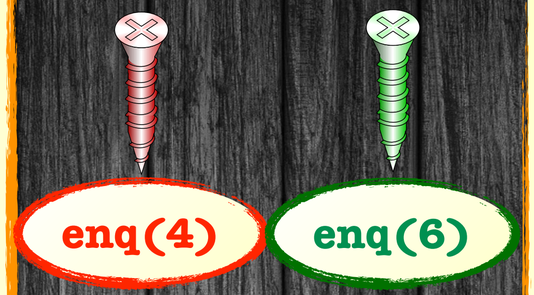
one thread helps another



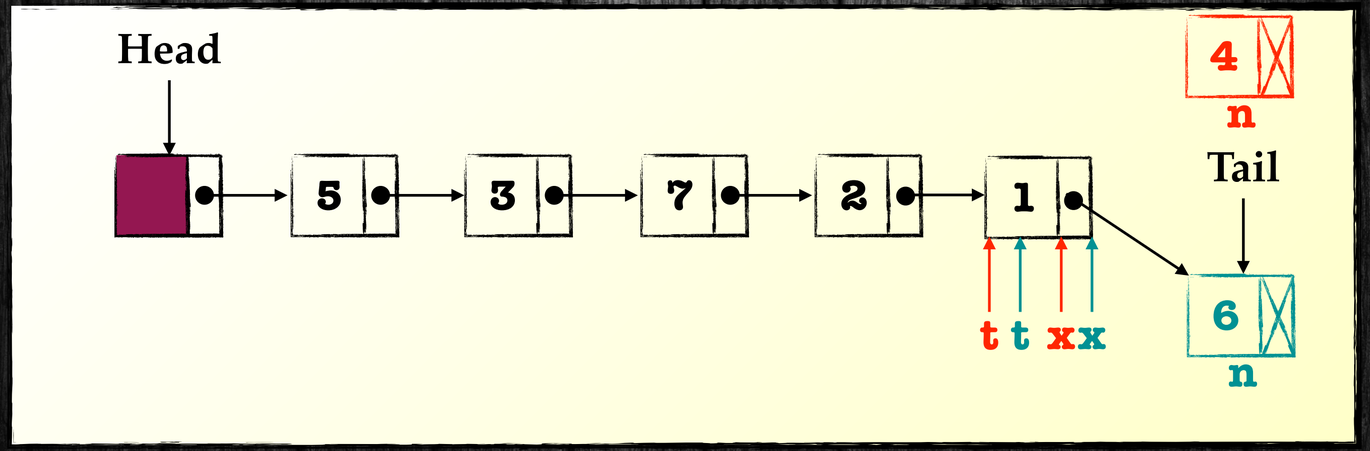
The Michael-Scott Queue Algorithm

Concurrent enqueue operations

```
enq(k):
Node t,x
1  n = new Node(k,NULL)
2  while (true)
3    t = Tail
4    x = t.next
5    if (t == Tail)
6      if (x == NULL)
7        if (CAS (t.next,NULL,n))
8          CAS (Tail,t,n)
9          exit
10     else
11       CAS (Tail,t,x)
```



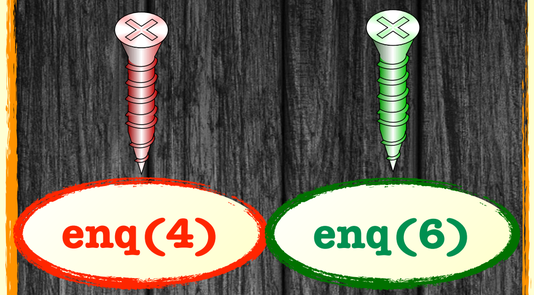

```
enq(k):
Node t,x
1  n = new Node(k,NULL)
2  while (true)
3    t = Tail
4    x = t.next
5    if (t == Tail)
6      if (x == NULL)
7        if (CAS (t.next,NULL,n))
8          CAS (Tail,t,n)
9          exit
10     else
11       CAS (Tail,t,x)
```



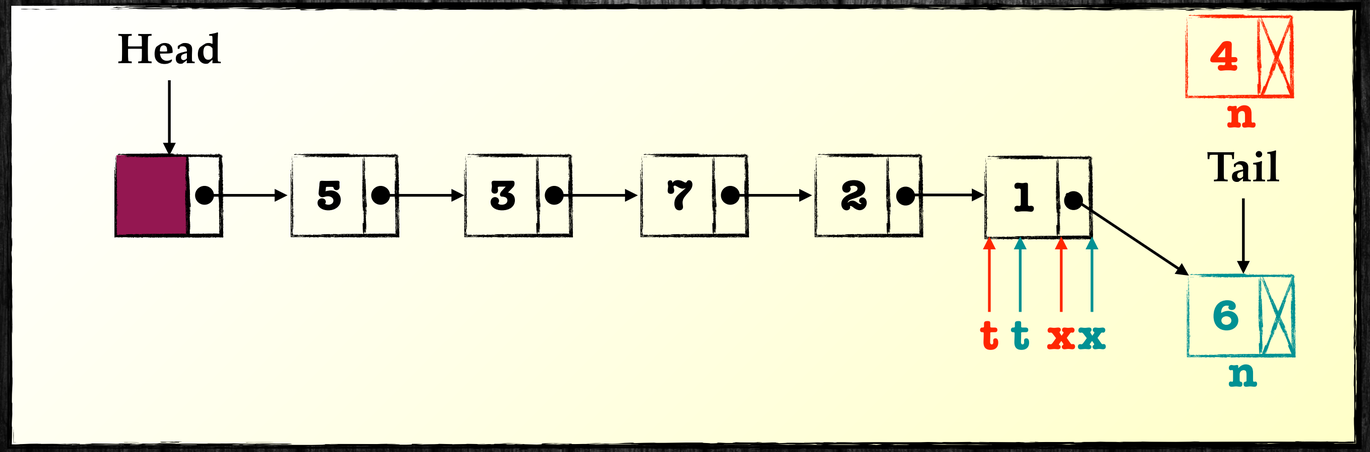
The Michael-Scott Queue Algorithm

Concurrent enqueue operations

```
enq(k):
Node t,x
1  n = new Node(k,NULL)
2  while (true)
3    t = Tail
4    x = t.next
5    if (t == Tail)
6      if (x == NULL)
7        if (CAS (t.next,NULL,n))
8          CAS (Tail,t,n)
9          exit
10     else
11       CAS (Tail,t,x)
```



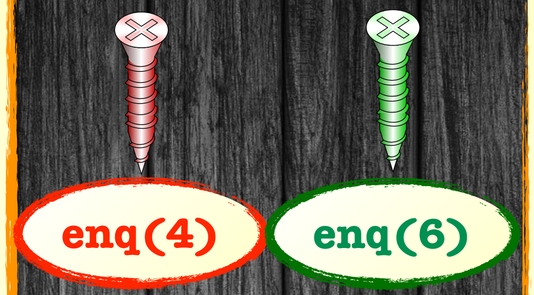

```
enq(k):
Node t,x
1  n = new Node(k,NULL)
2  while (true)
3    t = Tail
4    x = t.next
5    if (t == Tail)
6      if (x == NULL)
7        if (CAS (t.next,NULL,n))
8          CAS (Tail,t,n)
9          exit
10     else
11       CAS (Tail,t,x)
```



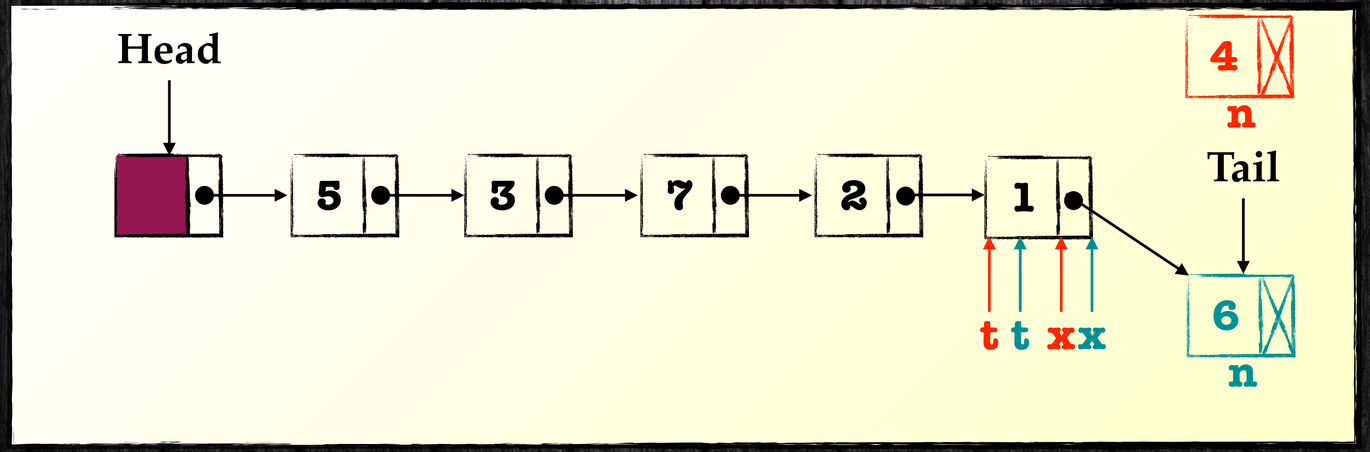
The Michael-Scott Queue Algorithm

Concurrent enqueue operations

```
enq(k):
Node t,x
1  n = new Node(k,NULL)
2  while (true)
3    t = Tail
4    x = t.next
5    if (t == Tail)
6      if (x == NULL)
7        if (CAS (t.next,NULL,n))
8          CAS (Tail,t,n)
9          exit
10     else
11       CAS (Tail,t,x)
```



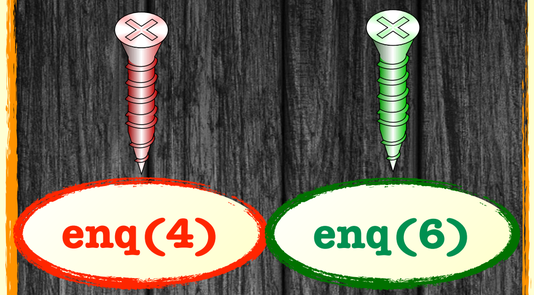

```
enq(k):
Node t,x
1  n = new Node(k,NULL)
2  while (true)
3    t = Tail
4    x = t.next
5    if (t == Tail)
6      if (x == NULL)
7        if (CAS (t.next,NULL,n))
8          CAS (Tail,t,n)
9          exit
10     else
11       CAS (Tail,t,x)
```



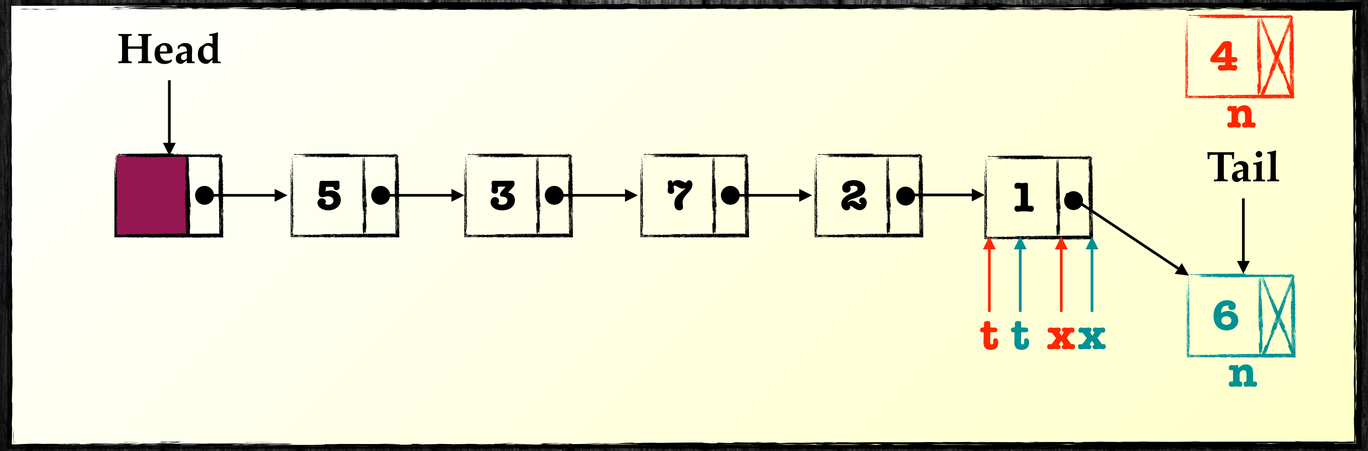
The Michael-Scott Queue Algorithm

Concurrent enqueue operations

```
enq(k):
Node t,x
1  n = new Node(k,NULL)
2  while (true)
3    t = Tail
4    x = t.next
5    if (t == Tail)
6      if (x == NULL)
7        if (CAS (t.next,NULL,n))
8          CAS (Tail,t,n)
9          exit
10     else
11       CAS (Tail,t,x)
```




```
enq(k):
Node t,x
1  n = new Node(k,NULL)
2  while (true)
3    t = Tail
4    x = t.next
5    if (t == Tail)
6      if (x == NULL)
7        if (CAS (t.next,NULL,n))
8          CAS (Tail,t,n)
9          exit
10     else
11       CAS (Tail,t,x)
```

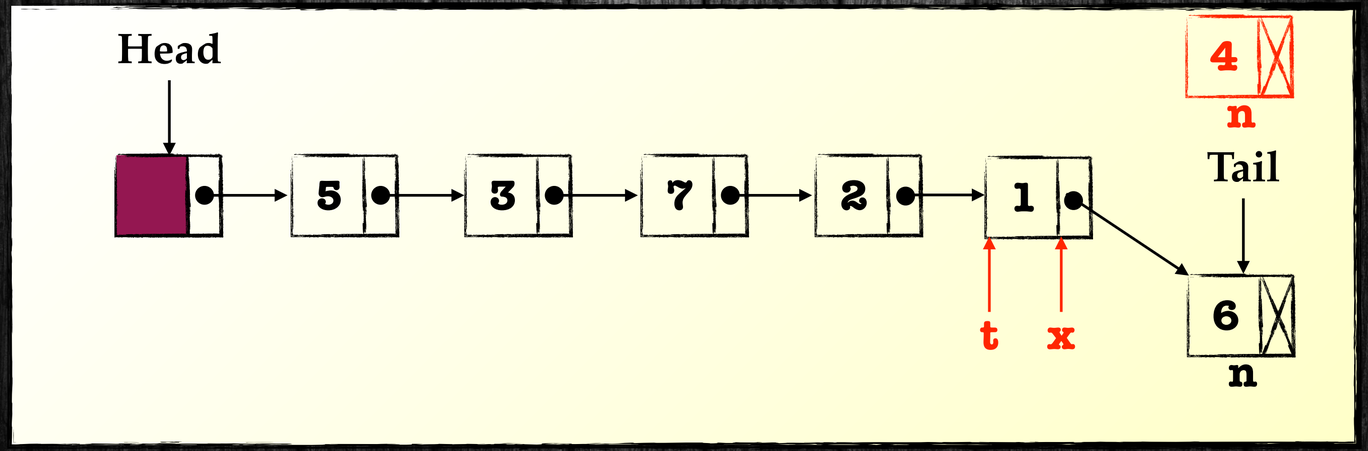


The Michael-Scott Queue Algorithm

Concurrent enqueue operations

```
enq(k):
Node t,x
 1  n = new Node(k,NULL)
 2  while (true)
 3    t = Tail
 4    x = t.next
 5    if (t == Tail)
 6      if (x == NULL)
 7        if (CAS (t.next,NULL,n))
 8          CAS (Tail,t,n)
 9          exit
10      else
11        CAS (Tail,t,x)
```

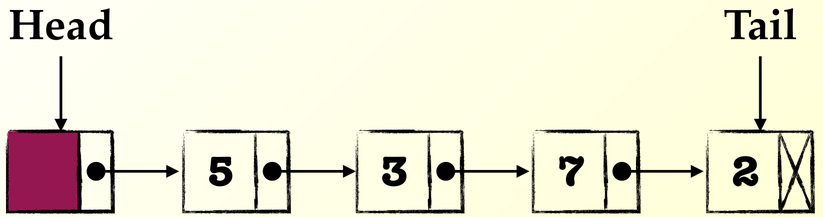




The Michael-Scott Queue Algorithm

```
deq(k):
Node h,t,x
Integer v
1  while (true)
2    h = Head
3    t = Tail
4    x = h.next
5    if (h == Head)
6      if (h == t)
7        if (x == NULL)
8          return *
9          exit
10       CAS (Tail,t,x)
11     else
12       v = x.key
13       if (CAS (Head,h,x))
14         return v
15         exit
```

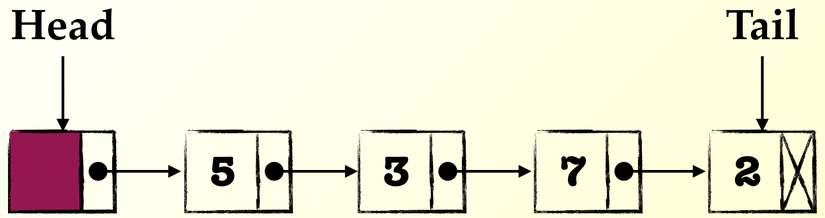




The Michael-Scott Queue Algorithm

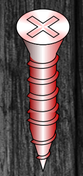
```
deq(k):
Node h,t,x
Integer v
1  while (true)
2    h = Head
3    t = Tail
4    x = h.next
5    if (h == Head)
6      if (h == t)
7        if (x == NULL)
8          return *
9          exit
10       CAS (Tail,t,x)
11     else
12       v = x.key
13       if (CAS (Head,h,x))
14         return v
15         exit
```





The Michael-Scott Queue Algorithm

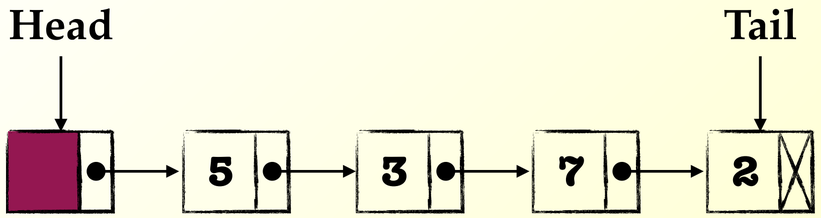
```
deq(k):
Node h,t,x
Integer v
1  while (true)
2    h = Head
3    t = Tail
4    x = h.next
5    if (h == Head)
6      if (h == t)
7        if (x == NULL)
8          return *
9          exit
10       CAS (Tail,t,x)
11     else
12       v = x.key
13       if (CAS (Head,h,x))
14         return v
15         exit
```



deq

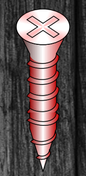
queue

5 3 7 2



The Michael-Scott Queue Algorithm

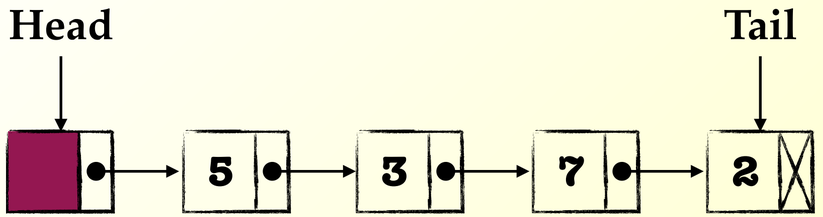
```
deq(k):
Node h,t,x
Integer v
1  while (true)
2    h = Head
3    t = Tail
4    x = h.next
5    if (h == Head)
6      if (h == t)
7        if (x == NULL)
8          return *
9          exit
10       CAS (Tail,t,x)
11     else
12       v = x.key
13       if (CAS (Head,h,x))
14         return v
15         exit
```



deq

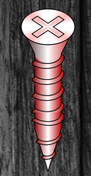
queue

5 3 7 2



The Michael-Scott Queue Algorithm

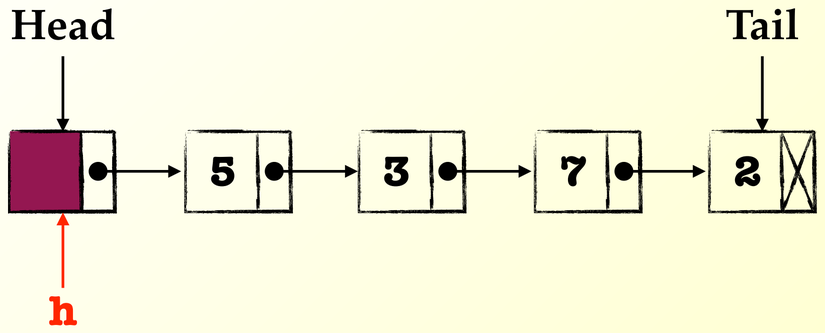
```
deq(k):
Node h,t,x
Integer v
1  while (true)
2    h = Head
3    t = Tail
4    x = h.next
5    if (h == Head)
6      if (h == t)
7        if (x == NULL)
8          return *
9          exit
10       CAS (Tail,t,x)
11     else
12       v = x.key
13       if (CAS (Head,h,x))
14         return v
15         exit
```



deq

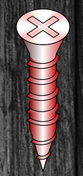
queue

5 3 7 2



The Michael-Scott Queue Algorithm

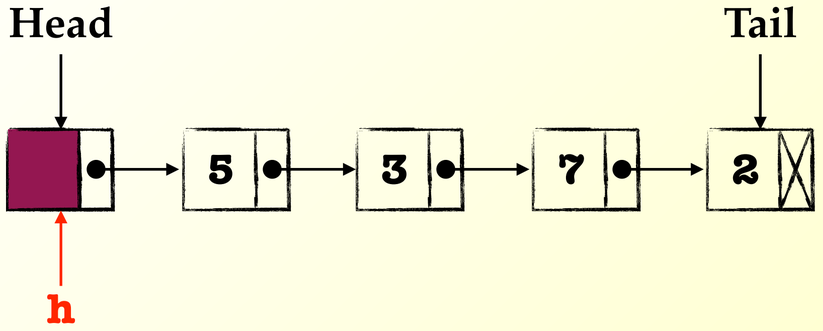
```
deq(k):
Node h,t,x
Integer v
1  while (true)
2    h = Head
3    t = Tail
4    x = h.next
5    if (h == Head)
6      if (h == t)
7        if (x == NULL)
8          return *
9          exit
10       CAS (Tail,t,x)
11     else
12       v = x.key
13       if (CAS (Head,h,x))
14         return v
15         exit
```



deq

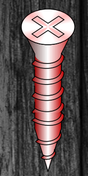
queue

5 3 7 2



The Michael-Scott Queue Algorithm

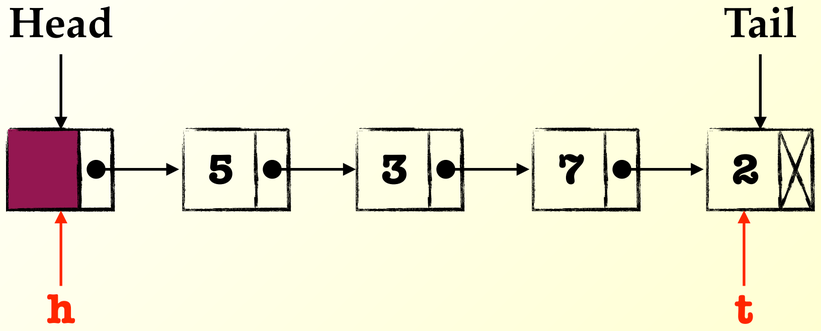
```
deq(k):
Node h,t,x
Integer v
1  while (true)
2    h = Head
3    t = Tail
4    x = h.next
5    if (h == Head)
6      if (h == t)
7        if (x == NULL)
8          return *
9          exit
10       CAS (Tail,t,x)
11     else
12       v = x.key
13       if (CAS (Head,h,x))
14         return v
15         exit
```



deq

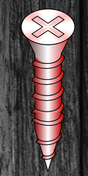
queue

5 3 7 2



The Michael-Scott Queue Algorithm

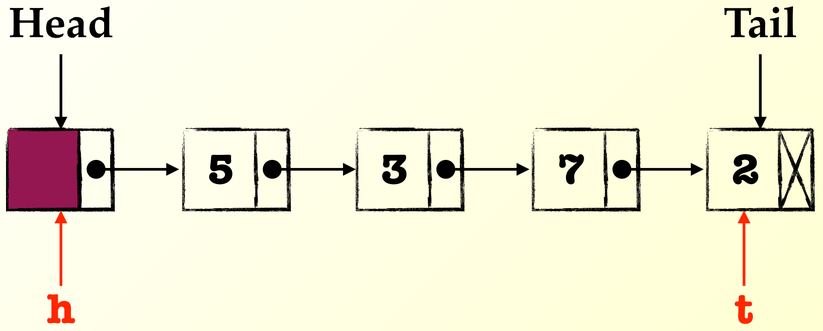
```
deq(k):
Node h,t,x
Integer v
1  while (true)
2    h = Head
3    t = Tail
4    x = h.next
5    if (h == Head)
6      if (h == t)
7        if (x == NULL)
8          return *
9          exit
10       CAS (Tail,t,x)
11     else
12       v = x.key
13       if (CAS (Head,h,x))
14         return v
15         exit
```



deq

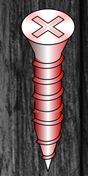
queue

5 3 7 2



The Michael-Scott Queue Algorithm

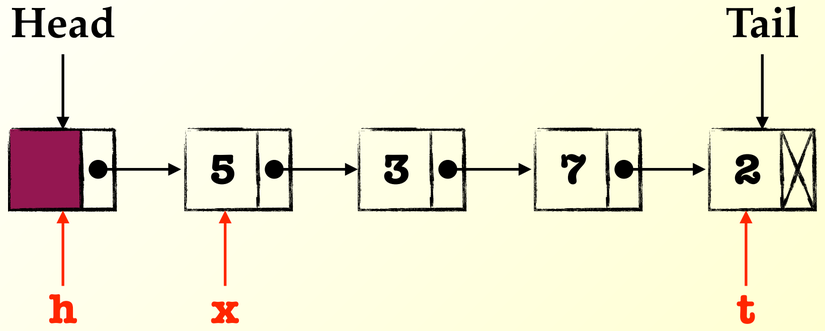
```
deq(k):
Node h,t,x
Integer v
1  while (true)
2    h = Head
3    t = Tail
4    x = h.next
5    if (h == Head)
6      if (h == t)
7        if (x == NULL)
8          return *
9          exit
10       CAS (Tail,t,x)
11     else
12       v = x.key
13       if (CAS (Head,h,x))
14         return v
15         exit
```



deq

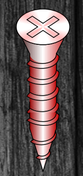
queue

5 3 7 2



The Michael-Scott Queue Algorithm

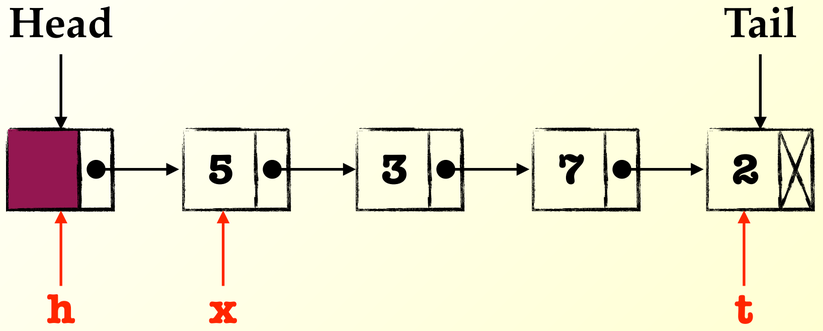
```
deq(k):
Node h,t,x
Integer v
1  while (true)
2    h = Head
3    t = Tail
4    x = h.next
5    if (h == Head)
6      if (h == t)
7        if (x == NULL)
8          return *
9          exit
10       CAS (Tail,t,x)
11     else
12       v = x.key
13       if (CAS (Head,h,x))
14         return v
15         exit
```



deq

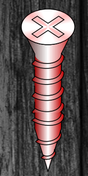
queue

5 3 7 2



The Michael-Scott Queue Algorithm

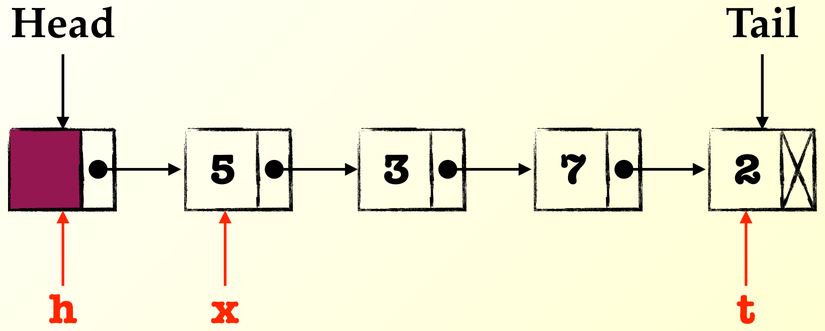
```
deq(k):
Node h,t,x
Integer v
1  while (true)
2    h = Head
3    t = Tail
4    x = h.next
5    if (h == Head)
6      if (h == t)
7        if (x == NULL)
8          return *
9          exit
10       CAS (Tail,t,x)
11     else
12       v = x.key
13       if (CAS (Head,h,x))
14         return v
15         exit
```



deq

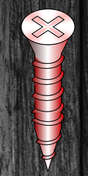
queue

5 3 7 2



The Michael-Scott Queue Algorithm

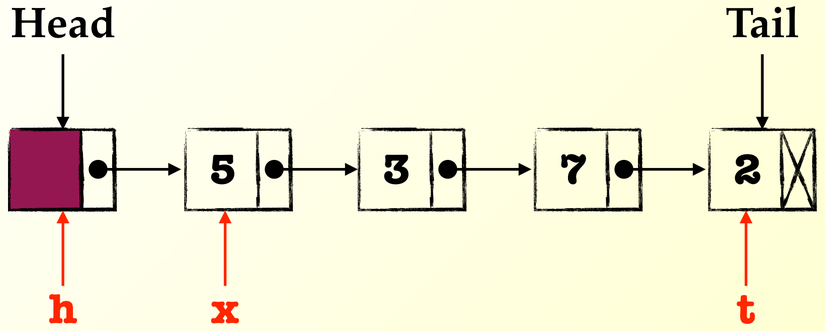
```
deq(k):
Node h,t,x
Integer v
1  while (true)
2    h = Head
3    t = Tail
4    x = h.next
5    if (h == Head)
6      if (h == t)
7        if (x == NULL)
8          return *
9          exit
10       CAS (Tail,t,x)
11     else
12       v = x.key
13       if (CAS (Head,h,x))
14         return v
15         exit
```



deq

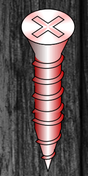
queue

5 3 7 2



The Michael-Scott Queue Algorithm

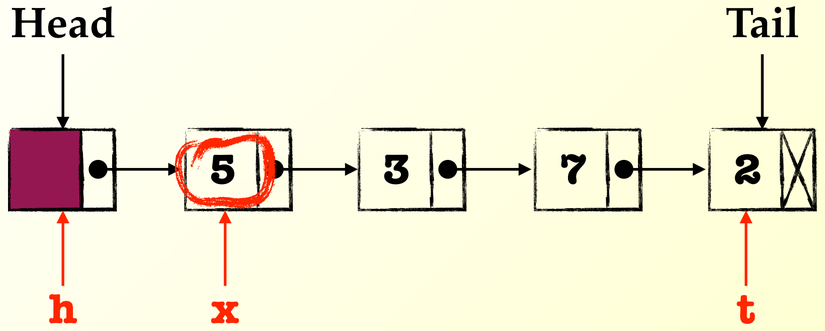
```
deq(k):
Node h,t,x
Integer v
1  while (true)
2    h = Head
3    t = Tail
4    x = h.next
5    if (h == Head)
6      if (h == t)
7        if (x == NULL)
8          return *
9          exit
10       CAS (Tail,t,x)
11     else
12       v = x.key
13       if (CAS (Head,h,x))
14         return v
15         exit
```



deq

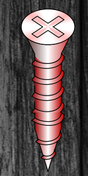
queue

5 3 7 2



The Michael-Scott Queue Algorithm

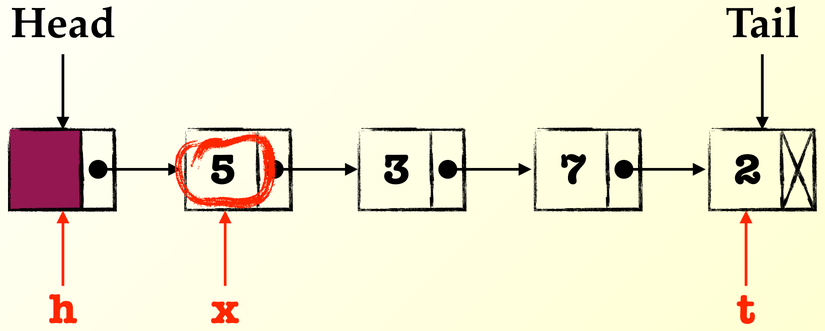
```
deq(k):
Node h,t,x
Integer v
1  while (true)
2    h = Head
3    t = Tail
4    x = h.next
5    if (h == Head)
6      if (h == t)
7        if (x == NULL)
8          return *
9          exit
10       CAS (Tail,t,x)
11     else
12       v = x.key
13       if (CAS (Head,h,x))
14         return v
15         exit
```



deq

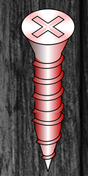
queue

5 3 7 2



The Michael-Scott Queue Algorithm

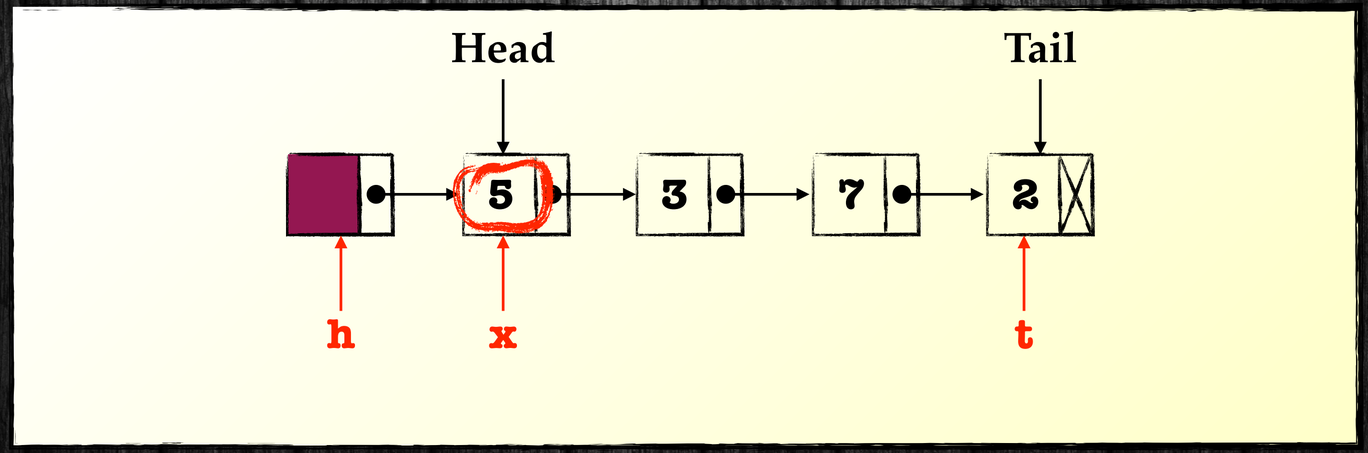
```
deq(k):
Node h,t,x
Integer v
1  while (true)
2    h = Head
3    t = Tail
4    x = h.next
5    if (h == Head)
6      if (h == t)
7        if (x == NULL)
8          return *
9          exit
10       CAS (Tail,t,x)
11     else
12       v = x.key
13       if (CAS (Head,h,x))
14         return v
15         exit
```



deq

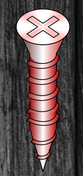
queue

5 3 7 2



The Michael-Scott Queue Algorithm

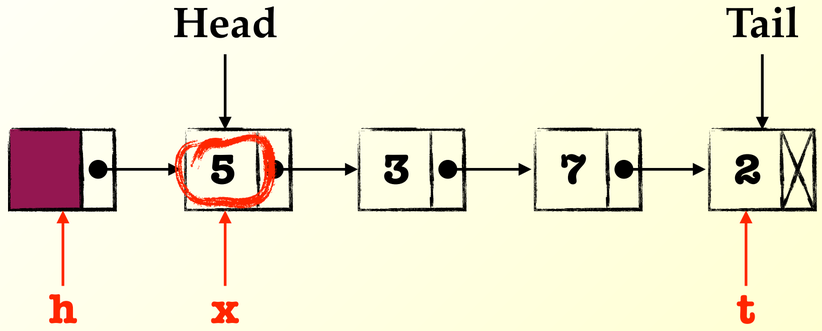
```
deq(k):
Node h,t,x
Integer v
1  while (true)
2    h = Head
3    t = Tail
4    x = h.next
5    if (h == Head)
6      if (h == t)
7        if (x == NULL)
8          return *
9          exit
10       CAS (Tail,t,x)
11     else
12       v = x.key
13       if (CAS (Head,h,x))
14         return v
15         exit
```



deq

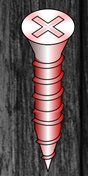
queue

5 3 7 2



The Michael-Scott Queue Algorithm

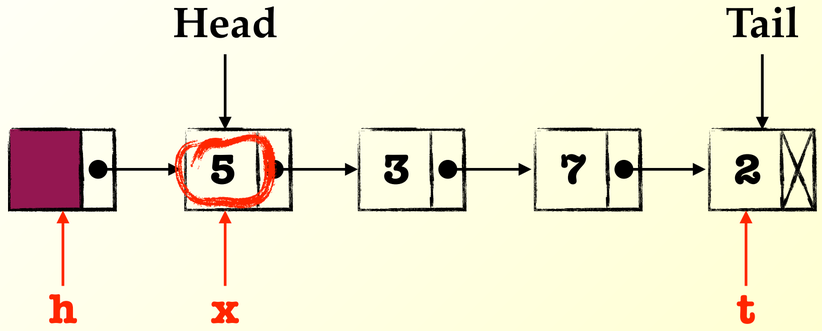
```
deq(k):
Node h,t,x
Integer v
1  while (true)
2    h = Head
3    t = Tail
4    x = h.next
5    if (h == Head)
6      if (h == t)
7        if (x == NULL)
8          return *
9          exit
10       CAS (Tail,t,x)
11     else
12       v = x.key
13       if (CAS (Head,h,x))
14         return v
15         exit
```



deq

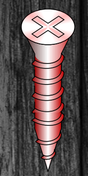
queue

5 3 7 2



The Michael-Scott Queue Algorithm

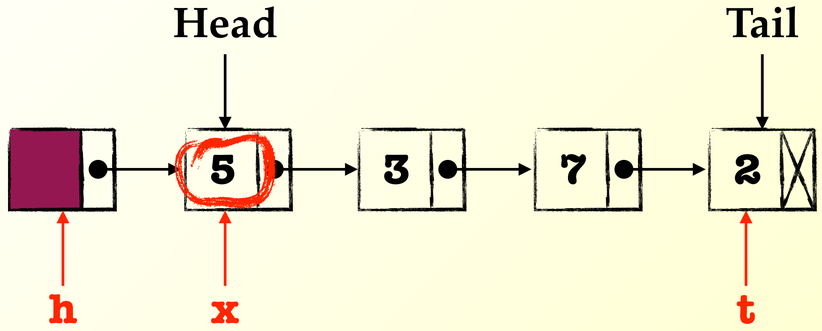
```
deq(k):
Node h,t,x
Integer v
1  while (true)
2    h = Head
3    t = Tail
4    x = h.next
5    if (h == Head)
6      if (h == t)
7        if (x == NULL)
8          return *
9          exit
10       CAS (Tail,t,x)
11     else
12       v = x.key
13       if (CAS (Head,h,x))
14         return v
15         exit
```



deq

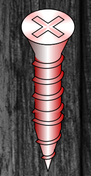
queue

5 3 7 2



The Michael-Scott Queue Algorithm

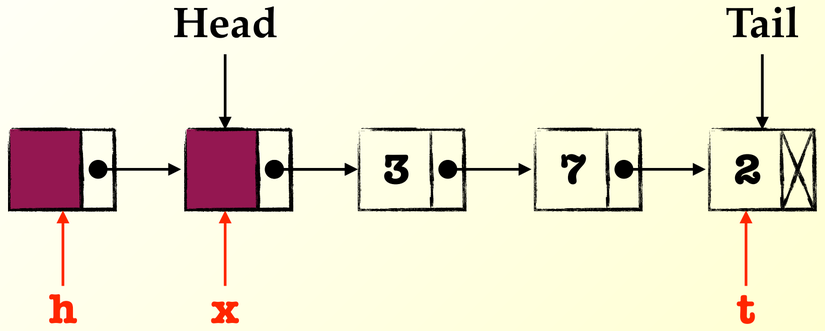
```
deq(k):
Node h,t,x
Integer v
1  while (true)
2    h = Head
3    t = Tail
4    x = h.next
5    if (h == Head)
6      if (h == t)
7        if (x == NULL)
8          return *
9          exit
10       CAS (Tail,t,x)
11     else
12       v = x.key
13       if (CAS (Head,h,x))
14         return v
15         exit
```



deq

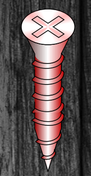
queue

5 3 7 2



The Michael-Scott Queue Algorithm

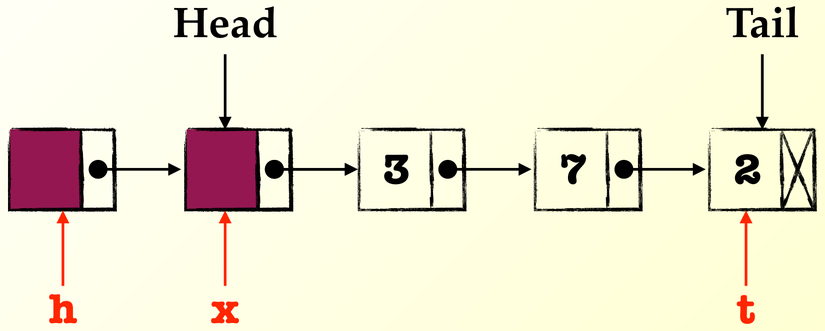
```
deq(k):
Node h,t,x
Integer v
1  while (true)
2    h = Head
3    t = Tail
4    x = h.next
5    if (h == Head)
6      if (h == t)
7        if (x == NULL)
8          return *
9          exit
10       CAS (Tail,t,x)
11     else
12       v = x.key
13       if (CAS (Head,h,x))
14         return v
15         exit
```



deq

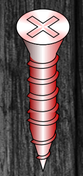
queue

5 3 7 2



The Michael-Scott Queue Algorithm

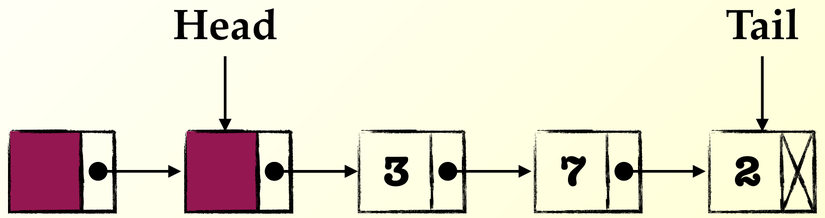
```
deq(k):
Node h,t,x
Integer v
1  while (true)
2    h = Head
3    t = Tail
4    x = h.next
5    if (h == Head)
6      if (h == t)
7        if (x == NULL)
8          return *
9          exit
10       CAS (Tail,t,x)
11     else
12       v = x.key
13       if (CAS (Head,h,x))
14         return v
15         exit
```



deq

queue

5 3 7 2



The Michael-Scott Queue Algorithm

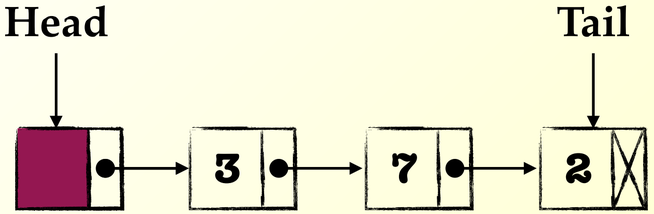
```
deq(k):
Node h,t,x
Integer v
1  while (true)
2    h = Head
3    t = Tail
4    x = h.next
5    if (h == Head)
6      if (h == t)
7        if (x == NULL)
8          return *
9          exit
10       CAS (Tail,t,x)
11     else
12       v = x.key
13       if (CAS (Head,h,x))
14         return v
15         exit
```



deq

queue

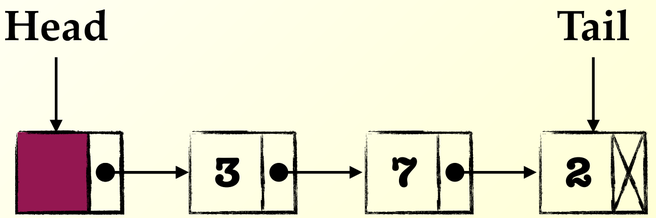
5 3 7 2



The Michael-Scott Queue Algorithm

```
deq(k):
Node h,t,x
Integer v
1  while (true)
2    h = Head
3    t = Tail
4    x = h.next
5    if (h == Head)
6      if (h == t)
7        if (x == NULL)
8          return *
9          exit
10       CAS (Tail,t,x)
11     else
12       v = x.key
13       if (CAS (Head,h,x))
14         return v
15         exit
```

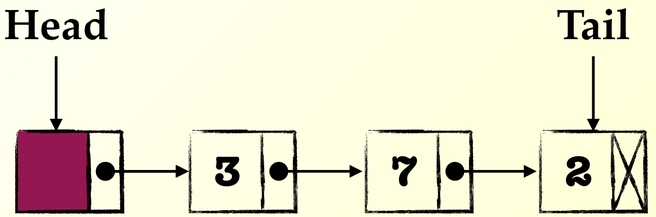




The Michael-Scott Queue Algorithm

```
deq(k):
Node h,t,x
Integer v
1  while (true)
2    h = Head
3    t = Tail
4    x = h.next
5    if (h == Head)
6      if (h == t)
7        if (x == NULL)
8          return *
9          exit
10       CAS (Tail,t,x)
11     else
12       v = x.key
13       if (CAS (Head,h,x))
14         return v
15         exit
```



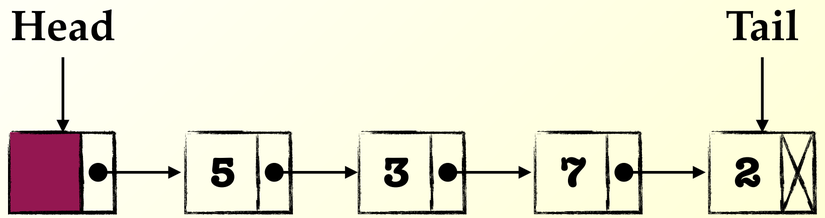


The Michael-Scott Queue Algorithm

Concurrent dequeue operations

```
deq(k):
Node h,t,x
Integer v
1  while (true)
2    h = Head
3    t = Tail
4    x = h.next
5    if (h == Head)
6      if (h == t)
7        if (x == NULL)
8          return *
9          exit
10       CAS (Tail,t,x)
11     else
12       v = x.key
13       if (CAS (Head,h,x))
14         return v
15         exit
```

```
deq(k):
Node h,t,x
Integer v
1  while (true)
2    h = Head
3    t = Tail
4    x = h.next
5    if (h == Head)
6      if (h == t)
7        if (x == NULL)
8          return *
9          exit
10       CAS (Tail,t,x)
11     else
12       v = x.key
13       if (CAS (Head,h,x))
14         return v
15         exit
```



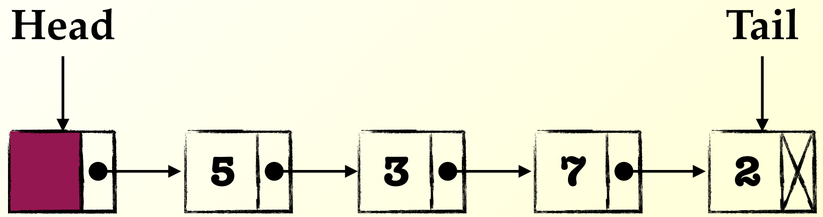
The Michael-Scott Queue Algorithm

Concurrent dequeue operations

```
deq(k):
Node h,t,x
Integer v
1  while (true)
2    h = Head
3    t = Tail
4    x = h.next
5    if (h == Head)
6      if (h == t)
7        if (x == NULL)
8          return *
9          exit
10       CAS (Tail,t,x)
11     else
12       v = x.key
13       if (CAS (Head,h,x))
14         return v
15         exit
```




```
deq(k):
Node h,t,x
Integer v
1  while (true)
2    h = Head
3    t = Tail
4    x = h.next
5    if (h == Head)
6      if (h == t)
7        if (x == NULL)
8          return *
9          exit
10       CAS (Tail,t,x)
11     else
12       v = x.key
13       if (CAS (Head,h,x))
14         return v
15         exit
```



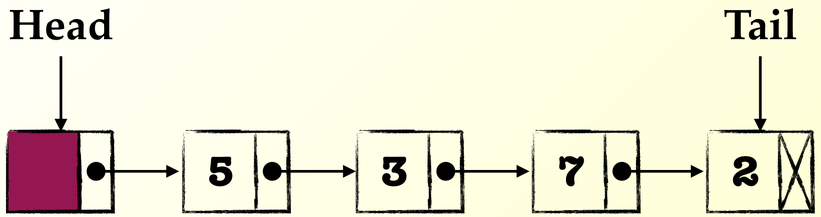
The Michael-Scott Queue Algorithm

Concurrent dequeue operations

```
deq(k):
Node h,t,x
Integer v
1  while (true)
2    h = Head
3    t = Tail
4    x = h.next
5    if (h == Head)
6      if (h == t)
7        if (x == NULL)
8          return *
9          exit
10       CAS (Tail,t,x)
11     else
12       v = x.key
13       if (CAS (Head,h,x))
14         return v
15         exit
```



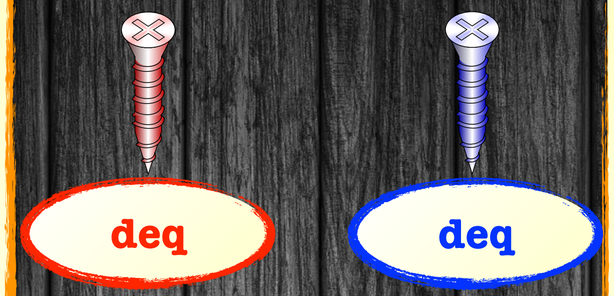

```
deq(k):
Node h,t,x
Integer v
1  while (true)
2    h = Head
3    t = Tail
4    x = h.next
5    if (h == Head)
6      if (h == t)
7        if (x == NULL)
8          return *
9          exit
10       CAS (Tail,t,x)
11     else
12       v = x.key
13       if (CAS (Head,h,x))
14         return v
15         exit
```



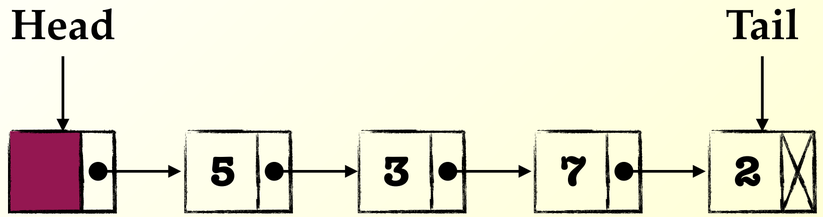
The Michael-Scott Queue Algorithm

Concurrent dequeue operations

```
deq(k):
Node h,t,x
Integer v
1  while (true)
2    h = Head
3    t = Tail
4    x = h.next
5    if (h == Head)
6      if (h == t)
7        if (x == NULL)
8          return *
9          exit
10       CAS (Tail,t,x)
11     else
12       v = x.key
13       if (CAS (Head,h,x))
14         return v
15         exit
```



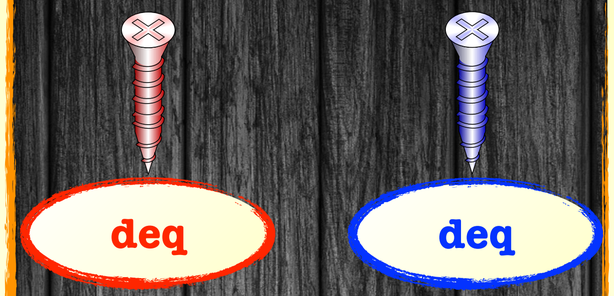

```
deq(k):
Node h,t,x
Integer v
1  while (true)
2    h = Head
3    t = Tail
4    x = h.next
5    if (h == Head)
6      if (h == t)
7        if (x == NULL)
8          return *
9          exit
10       CAS (Tail,t,x)
11     else
12       v = x.key
13       if (CAS (Head,h,x))
14         return v
15         exit
```



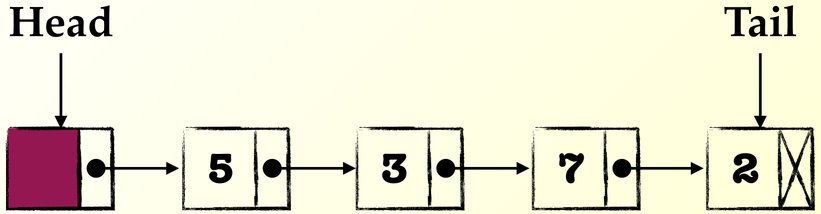
The Michael-Scott Queue Algorithm

Concurrent dequeue operations

```
deq(k):
Node h,t,x
Integer v
1  while (true)
2    h = Head
3    t = Tail
4    x = h.next
5    if (h == Head)
6      if (h == t)
7        if (x == NULL)
8          return *
9          exit
10       CAS (Tail,t,x)
11     else
12       v = x.key
13       if (CAS (Head,h,x))
14         return v
15         exit
```



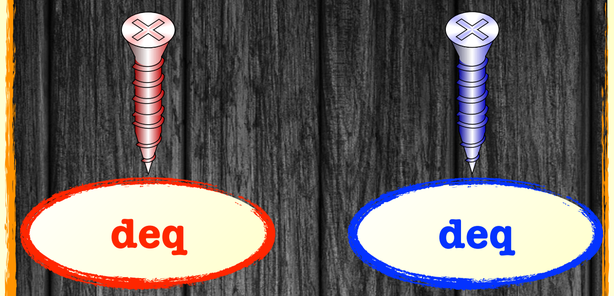

```
deq(k):
Node h,t,x
Integer v
1  while (true)
2    h = Head
3    t = Tail
4    x = h.next
5    if (h == Head)
6      if (h == t)
7        if (x == NULL)
8          return *
9          exit
10       CAS (Tail,t,x)
11     else
12       v = x.key
13       if (CAS (Head,h,x))
14         return v
15         exit
```



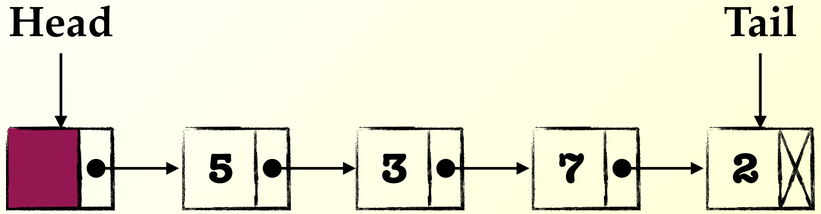
The Michael-Scott Queue Algorithm

Concurrent dequeue operations

```
deq(k):
Node h,t,x
Integer v
1  while (true)
2    h = Head
3    t = Tail
4    x = h.next
5    if (h == Head)
6      if (h == t)
7        if (x == NULL)
8          return *
9          exit
10       CAS (Tail,t,x)
11     else
12       v = x.key
13       if (CAS (Head,h,x))
14         return v
15         exit
```



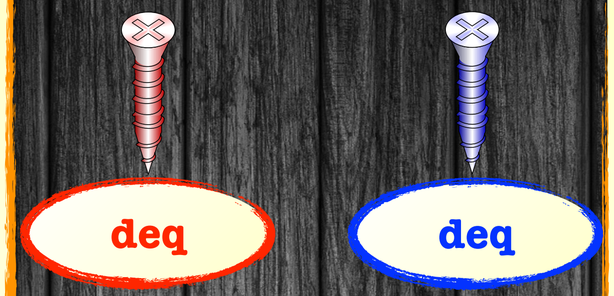

```
deq(k):
Node h,t,x
Integer v
1  while (true)
2    h = Head
3    t = Tail
4    x = h.next
5    if (h == Head)
6      if (h == t)
7        if (x == NULL)
8          return *
9          exit
10       CAS (Tail,t,x)
11     else
12       v = x.key
13       if (CAS (Head,h,x))
14         return v
15         exit
```



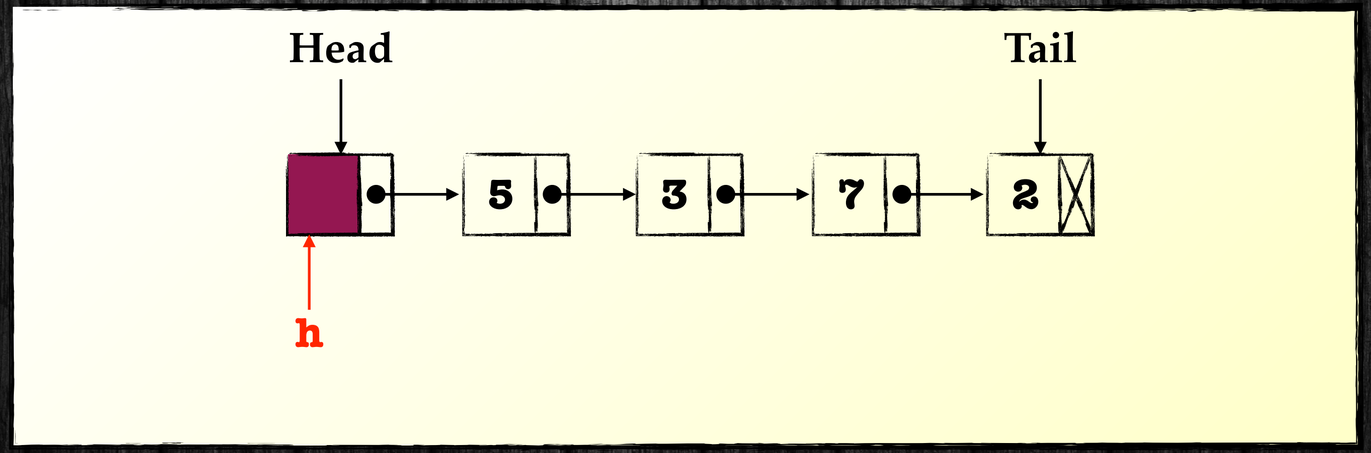
The Michael-Scott Queue Algorithm

Concurrent dequeue operations

```
deq(k):
Node h,t,x
Integer v
1  while (true)
2    h = Head
3    t = Tail
4    x = h.next
5    if (h == Head)
6      if (h == t)
7        if (x == NULL)
8          return *
9          exit
10       CAS (Tail,t,x)
11     else
12       v = x.key
13       if (CAS (Head,h,x))
14         return v
15         exit
```



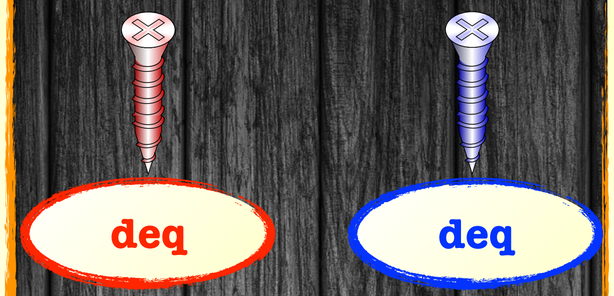

```
deq(k):
Node h,t,x
Integer v
1  while (true)
2    h = Head
3    t = Tail
4    x = h.next
5    if (h == Head)
6      if (h == t)
7        if (x == NULL)
8          return *
9          exit
10       CAS (Tail,t,x)
11     else
12       v = x.key
13       if (CAS (Head,h,x))
14         return v
15         exit
```



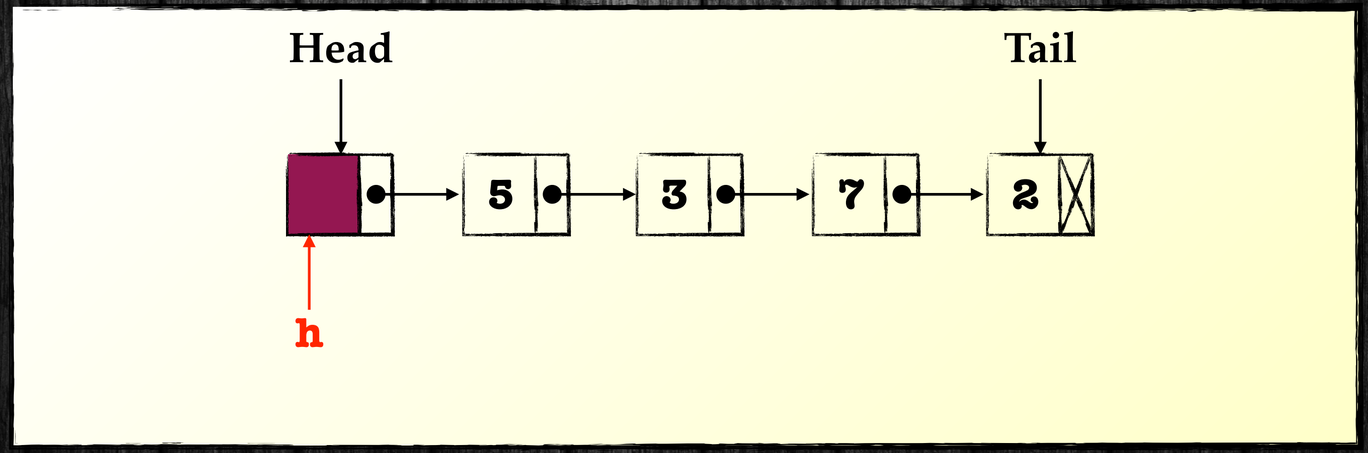
The Michael-Scott Queue Algorithm

Concurrent dequeue operations

```
deq(k):
Node h,t,x
Integer v
1  while (true)
2    h = Head
3    t = Tail
4    x = h.next
5    if (h == Head)
6      if (h == t)
7        if (x == NULL)
8          return *
9          exit
10       CAS (Tail,t,x)
11     else
12       v = x.key
13       if (CAS (Head,h,x))
14         return v
15         exit
```



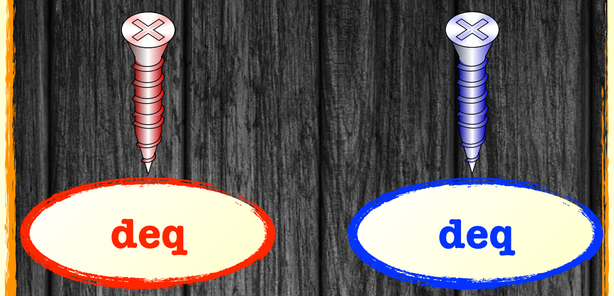

```
deq(k):
Node h,t,x
Integer v
1  while (true)
2    h = Head
3    t = Tail
4    x = h.next
5    if (h == Head)
6      if (h == t)
7        if (x == NULL)
8          return *
9          exit
10       CAS (Tail,t,x)
11     else
12       v = x.key
13       if (CAS (Head,h,x))
14         return v
15         exit
```



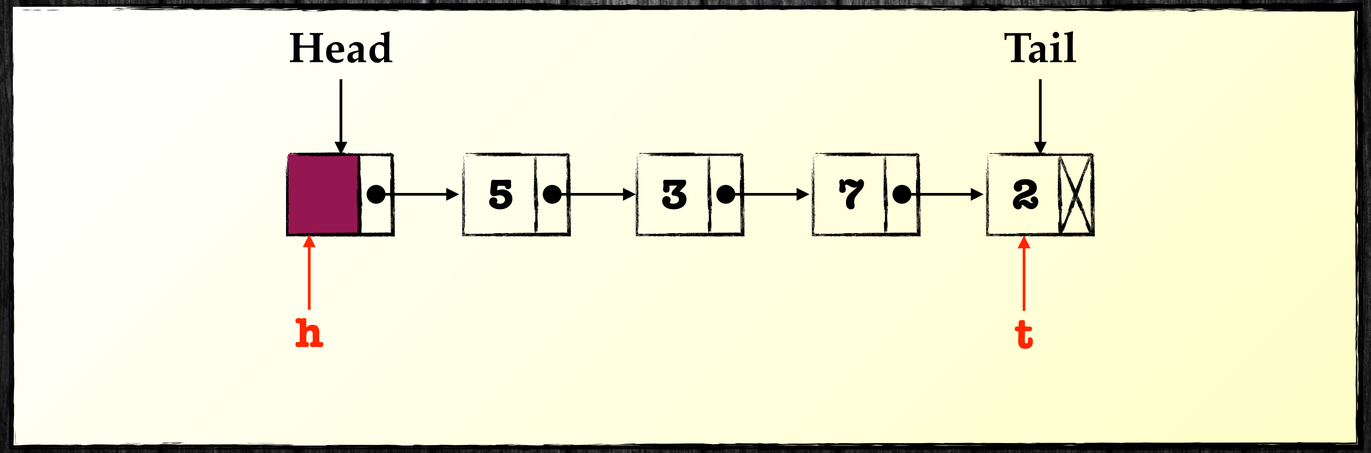
The Michael-Scott Queue Algorithm

Concurrent dequeue operations

```
deq(k):
Node h,t,x
Integer v
1  while (true)
2    h = Head
3    t = Tail
4    x = h.next
5    if (h == Head)
6      if (h == t)
7        if (x == NULL)
8          return *
9          exit
10       CAS (Tail,t,x)
11     else
12       v = x.key
13       if (CAS (Head,h,x))
14         return v
15         exit
```



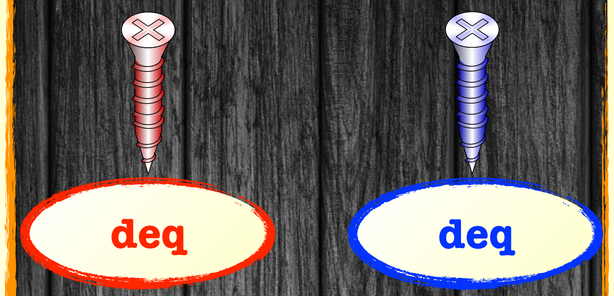

```
deq(k):
Node h,t,x
Integer v
1  while (true)
2    h = Head
3    t = Tail
4    x = h.next
5    if (h == Head)
6      if (h == t)
7        if (x == NULL)
8          return *
9          exit
10       CAS (Tail,t,x)
11     else
12       v = x.key
13       if (CAS (Head,h,x))
14         return v
15         exit
```



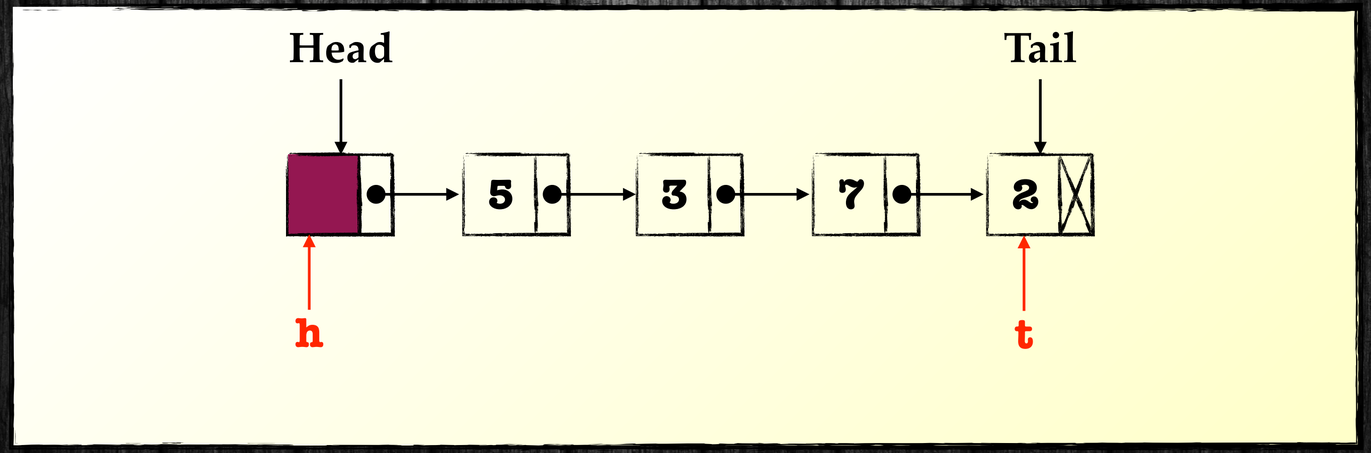
The Michael-Scott Queue Algorithm

Concurrent dequeue operations

```
deq(k):
Node h,t,x
Integer v
1  while (true)
2    h = Head
3    t = Tail
4    x = h.next
5    if (h == Head)
6      if (h == t)
7        if (x == NULL)
8          return *
9          exit
10       CAS (Tail,t,x)
11     else
12       v = x.key
13       if (CAS (Head,h,x))
14         return v
15         exit
```



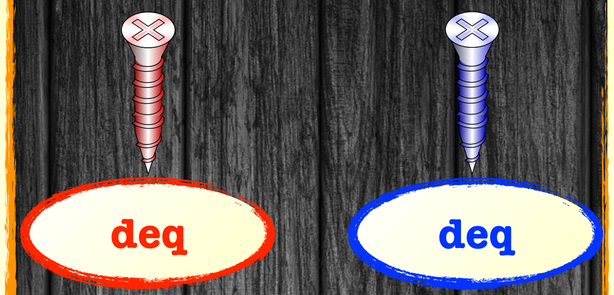

```
deq(k):
Node h,t,x
Integer v
1  while (true)
2    h = Head
3    t = Tail
4    x = h.next
5    if (h == Head)
6      if (h == t)
7        if (x == NULL)
8          return *
9          exit
10       CAS (Tail,t,x)
11     else
12       v = x.key
13       if (CAS (Head,h,x))
14         return v
15         exit
```



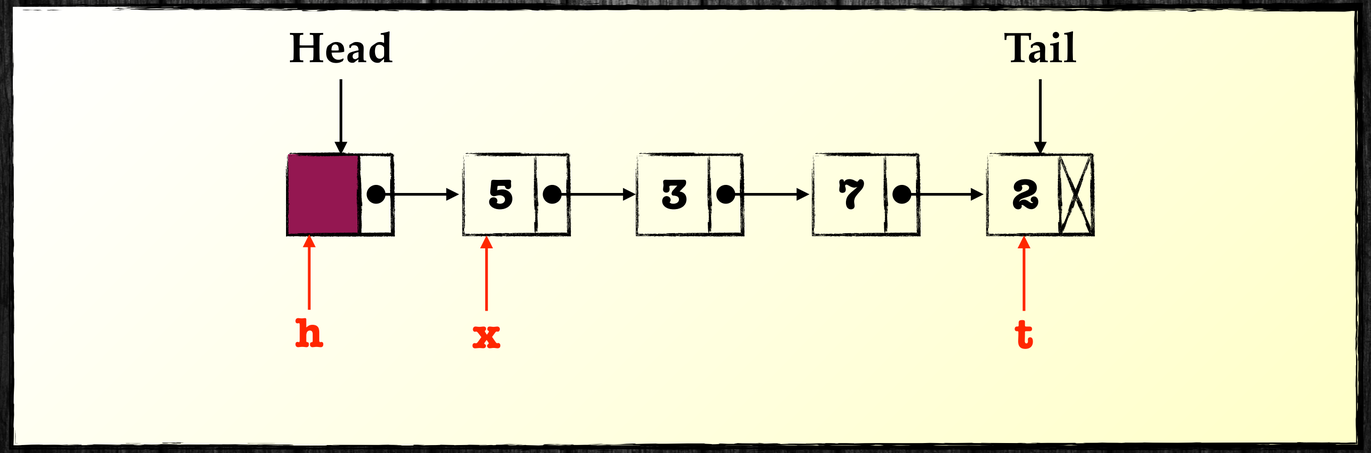
The Michael-Scott Queue Algorithm

Concurrent dequeue operations

```
deq(k):
Node h,t,x
Integer v
1  while (true)
2    h = Head
3    t = Tail
4    x = h.next
5    if (h == Head)
6      if (h == t)
7        if (x == NULL)
8          return *
9          exit
10       CAS (Tail,t,x)
11     else
12       v = x.key
13       if (CAS (Head,h,x))
14         return v
15         exit
```



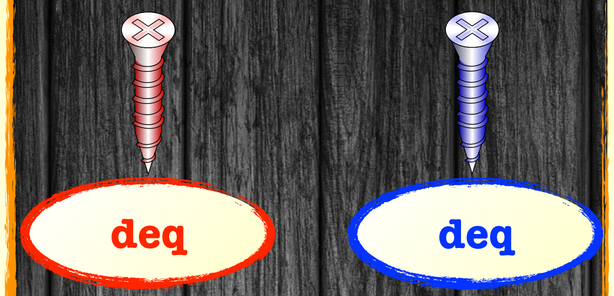

```
deq(k):
Node h,t,x
Integer v
1  while (true)
2    h = Head
3    t = Tail
4    x = h.next
5    if (h == Head)
6      if (h == t)
7        if (x == NULL)
8          return *
9          exit
10       CAS (Tail,t,x)
11     else
12       v = x.key
13       if (CAS (Head,h,x))
14         return v
15         exit
```



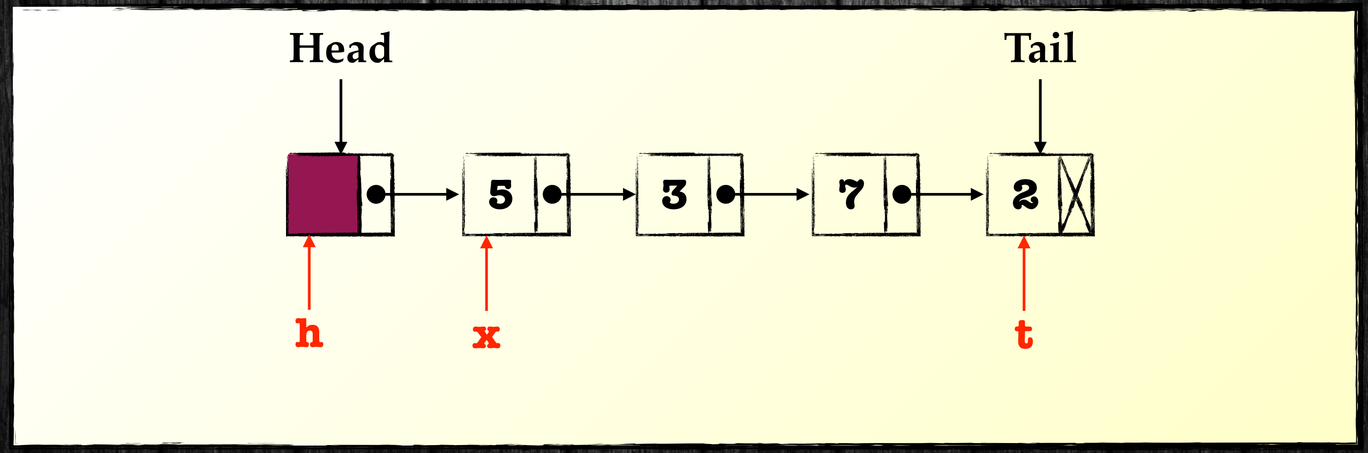
The Michael-Scott Queue Algorithm

Concurrent dequeue operations

```
deq(k):
Node h,t,x
Integer v
1  while (true)
2    h = Head
3    t = Tail
4    x = h.next
5    if (h == Head)
6      if (h == t)
7        if (x == NULL)
8          return *
9          exit
10       CAS (Tail,t,x)
11     else
12       v = x.key
13       if (CAS (Head,h,x))
14         return v
15         exit
```



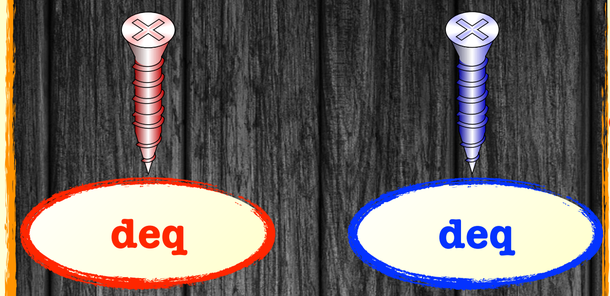

```
deq(k):
Node h,t,x
Integer v
1  while (true)
2    h = Head
3    t = Tail
4    x = h.next
5    if (h == Head)
6      if (h == t)
7        if (x == NULL)
8          return *
9          exit
10       CAS (Tail,t,x)
11     else
12       v = x.key
13       if (CAS (Head,h,x))
14         return v
15         exit
```



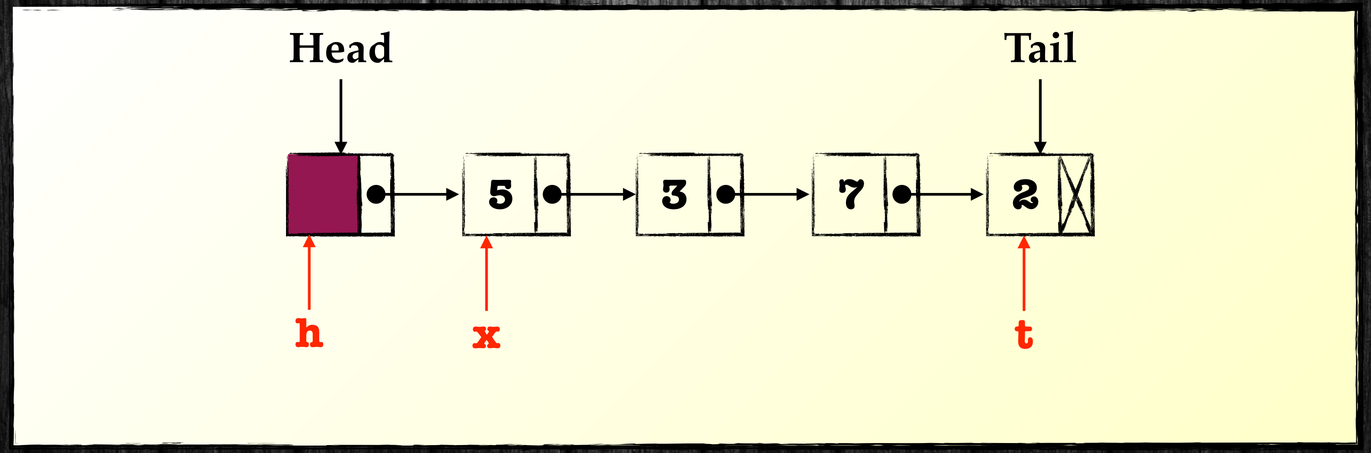
The Michael-Scott Queue Algorithm

Concurrent dequeue operations

```
deq(k):
Node h,t,x
Integer v
1  while (true)
2    h = Head
3    t = Tail
4    x = h.next
5    if (h == Head)
6      if (h == t)
7        if (x == NULL)
8          return *
9          exit
10       CAS (Tail,t,x)
11     else
12       v = x.key
13       if (CAS (Head,h,x))
14         return v
15         exit
```



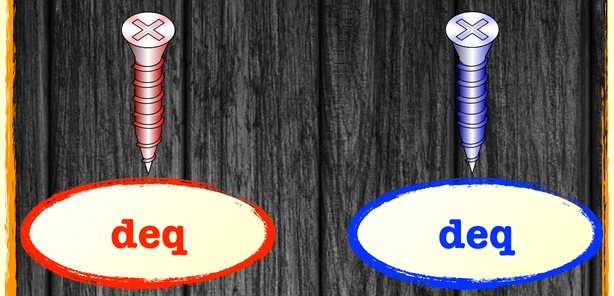

```
deq(k):
Node h,t,x
Integer v
1  while (true)
2    h = Head
3    t = Tail
4    x = h.next
5    if (h == Head)
6      if (h == t)
7        if (x == NULL)
8          return *
9          exit
10       CAS (Tail,t,x)
11     else
12       v = x.key
13       if (CAS (Head,h,x))
14         return v
15         exit
```



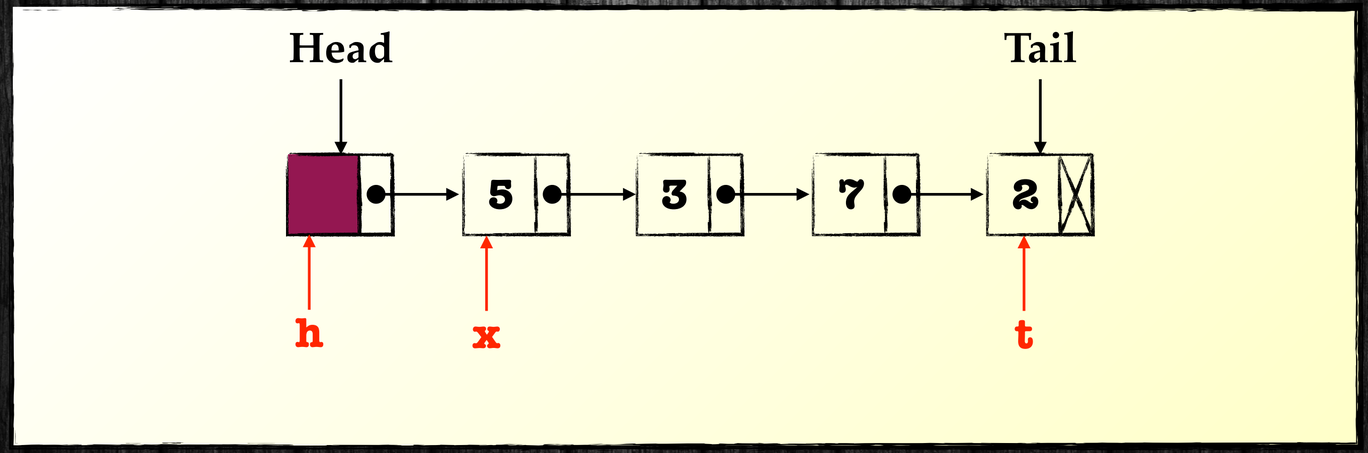
The Michael-Scott Queue Algorithm

Concurrent dequeue operations

```
deq(k):
Node h,t,x
Integer v
1  while (true)
2    h = Head
3    t = Tail
4    x = h.next
5    if (h == Head)
6      if (h == t)
7        if (x == NULL)
8          return *
9          exit
10       CAS (Tail,t,x)
11     else
12       v = x.key
13       if (CAS (Head,h,x))
14         return v
15         exit
```



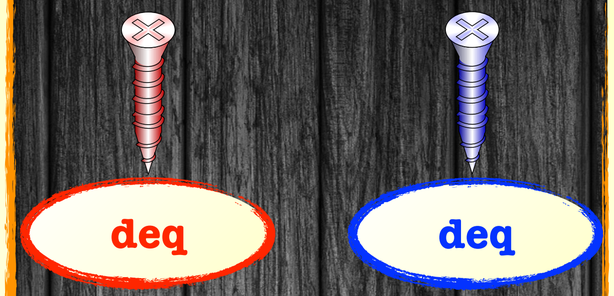

```
deq(k):
Node h,t,x
Integer v
1  while (true)
2    h = Head
3    t = Tail
4    x = h.next
5    if (h == Head)
6      if (h == t)
7        if (x == NULL)
8          return *
9          exit
10       CAS (Tail,t,x)
11     else
12       v = x.key
13       if (CAS (Head,h,x))
14         return v
15         exit
```



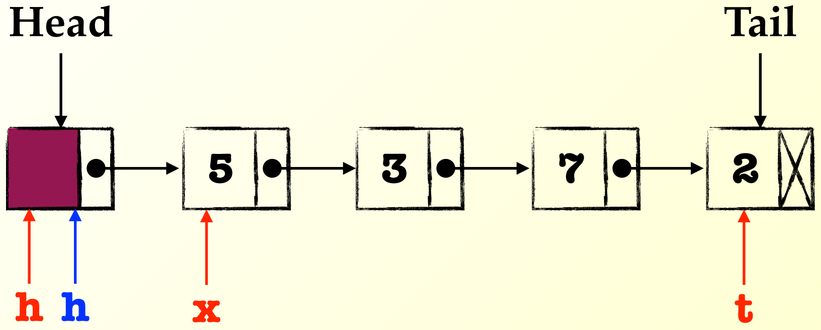
The Michael-Scott Queue Algorithm

Concurrent dequeue operations

```
deq(k):
Node h,t,x
Integer v
1  while (true)
2    h = Head
3    t = Tail
4    x = h.next
5    if (h == Head)
6      if (h == t)
7        if (x == NULL)
8          return *
9          exit
10       CAS (Tail,t,x)
11     else
12       v = x.key
13       if (CAS (Head,h,x))
14         return v
15         exit
```



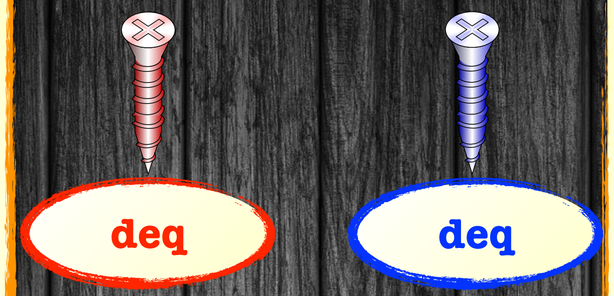

```
deq(k):
Node h,t,x
Integer v
1  while (true)
2    h = Head
3    t = Tail
4    x = h.next
5    if (h == Head)
6      if (h == t)
7        if (x == NULL)
8          return *
9          exit
10       CAS (Tail,t,x)
11     else
12       v = x.key
13       if (CAS (Head,h,x))
14         return v
15         exit
```



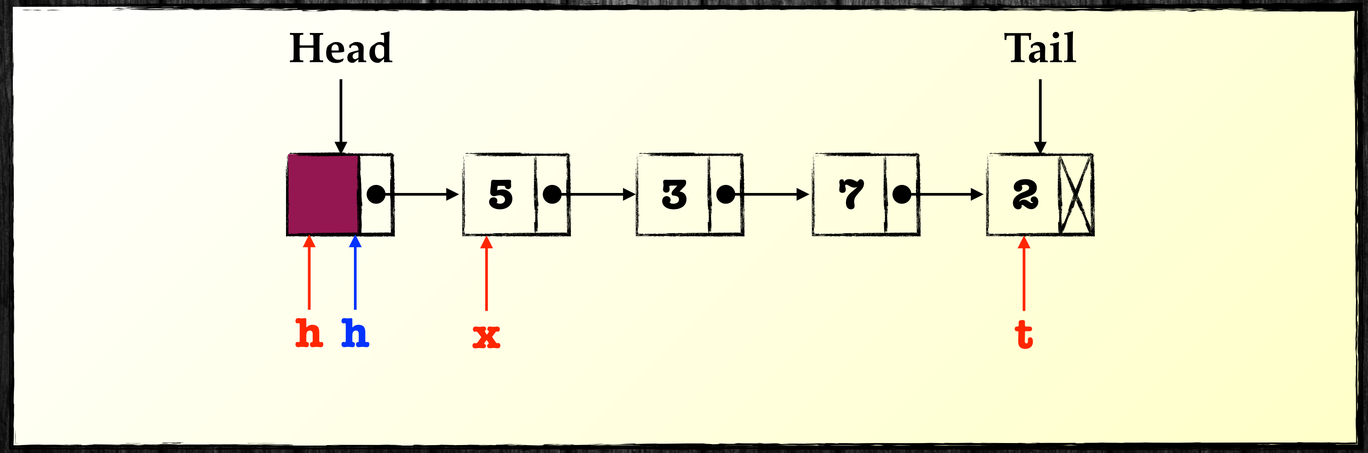
The Michael-Scott Queue Algorithm

Concurrent dequeue operations

```
deq(k):
Node h,t,x
Integer v
1  while (true)
2    h = Head
3    t = Tail
4    x = h.next
5    if (h == Head)
6      if (h == t)
7        if (x == NULL)
8          return *
9          exit
10       CAS (Tail,t,x)
11     else
12       v = x.key
13       if (CAS (Head,h,x))
14         return v
15         exit
```



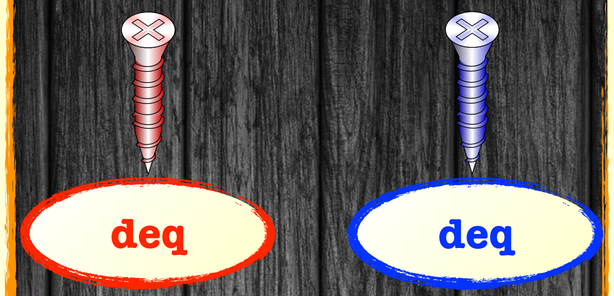

```
deq(k):
Node h,t,x
Integer v
1  while (true)
2    h = Head
3    t = Tail
4    x = h.next
5    if (h == Head)
6      if (h == t)
7        if (x == NULL)
8          return *
9          exit
10       CAS (Tail,t,x)
11     else
12       v = x.key
13       if (CAS (Head,h,x))
14         return v
15         exit
```



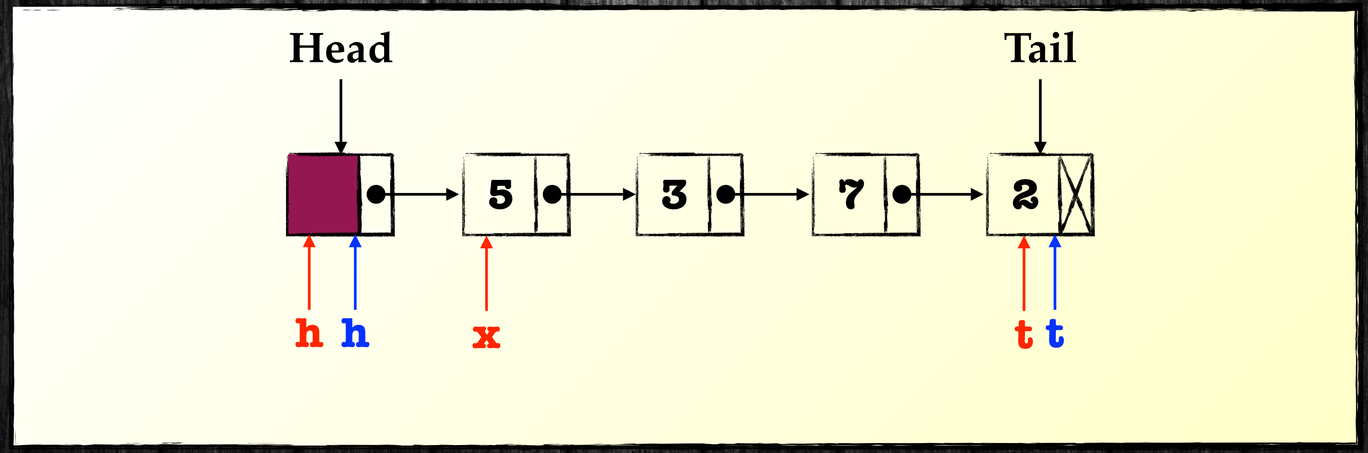
The Michael-Scott Queue Algorithm

Concurrent dequeue operations

```
deq(k):
Node h,t,x
Integer v
1  while (true)
2    h = Head
3    t = Tail
4    x = h.next
5    if (h == Head)
6      if (h == t)
7        if (x == NULL)
8          return *
9          exit
10       CAS (Tail,t,x)
11     else
12       v = x.key
13       if (CAS (Head,h,x))
14         return v
15         exit
```



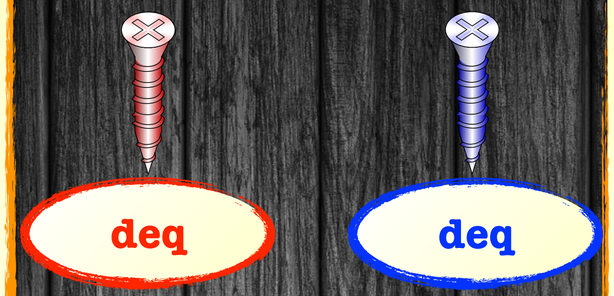

```
deq(k):
Node h,t,x
Integer v
1  while (true)
2    h = Head
3    t = Tail
4    x = h.next
5    if (h == Head)
6      if (h == t)
7        if (x == NULL)
8          return *
9          exit
10       CAS (Tail,t,x)
11     else
12       v = x.key
13       if (CAS (Head,h,x))
14         return v
15         exit
```



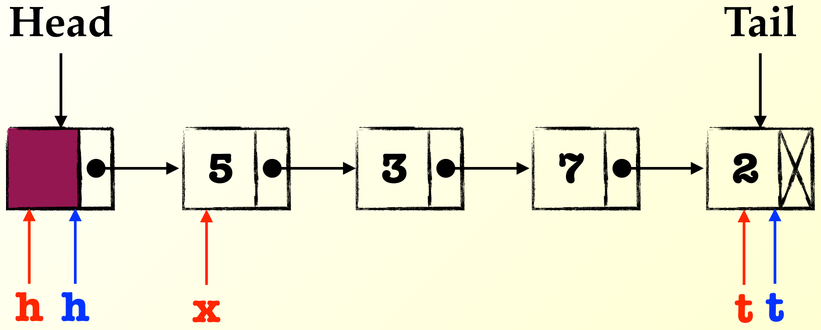
The Michael-Scott Queue Algorithm

Concurrent dequeue operations

```
deq(k):
Node h,t,x
Integer v
1  while (true)
2    h = Head
3    t = Tail
4    x = h.next
5    if (h == Head)
6      if (h == t)
7        if (x == NULL)
8          return *
9          exit
10       CAS (Tail,t,x)
11     else
12       v = x.key
13       if (CAS (Head,h,x))
14         return v
15         exit
```



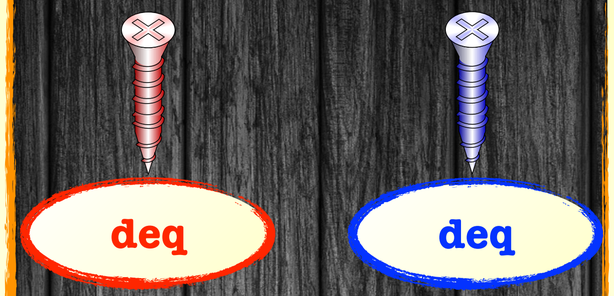

```
deq(k):
Node h,t,x
Integer v
1  while (true)
2    h = Head
3    t = Tail
4    x = h.next
5    if (h == Head)
6      if (h == t)
7        if (x == NULL)
8          return *
9          exit
10       CAS (Tail,t,x)
11     else
12       v = x.key
13       if (CAS (Head,h,x))
14         return v
15         exit
```



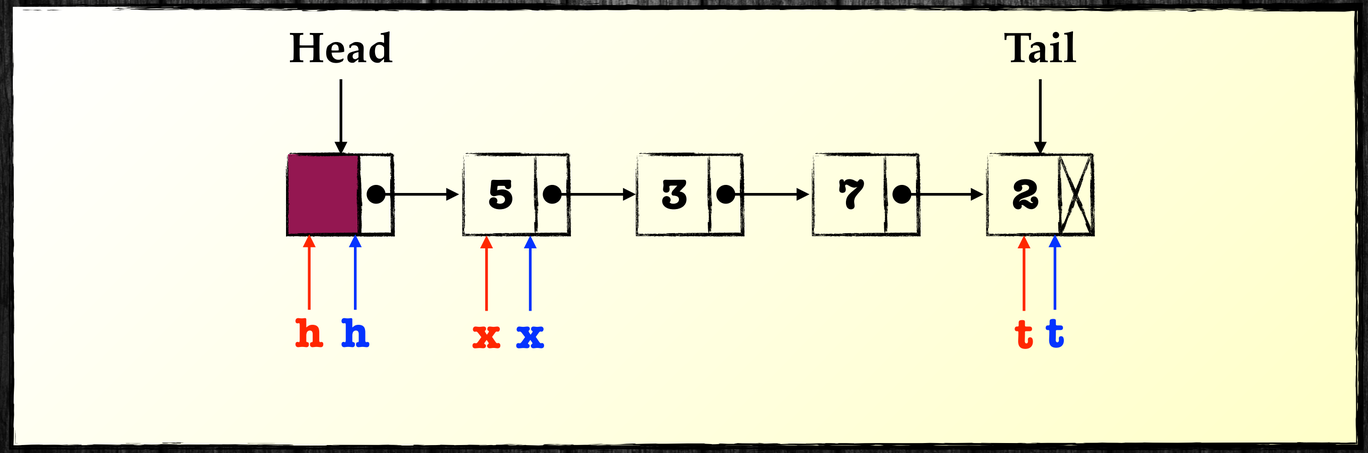
The Michael-Scott Queue Algorithm

Concurrent dequeue operations

```
deq(k):
Node h,t,x
Integer v
1  while (true)
2    h = Head
3    t = Tail
4    x = h.next
5    if (h == Head)
6      if (h == t)
7        if (x == NULL)
8          return *
9          exit
10       CAS (Tail,t,x)
11     else
12       v = x.key
13       if (CAS (Head,h,x))
14         return v
15         exit
```



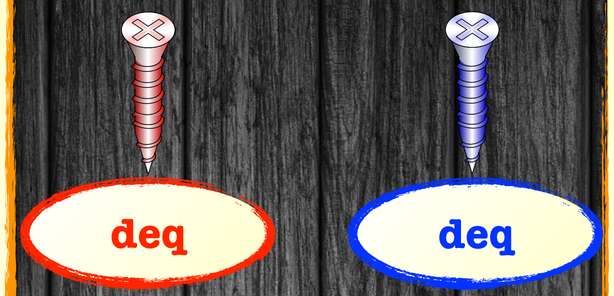

```
deq(k):
Node h,t,x
Integer v
1  while (true)
2    h = Head
3    t = Tail
4    x = h.next
5    if (h == Head)
6      if (h == t)
7        if (x == NULL)
8          return *
9          exit
10       CAS (Tail,t,x)
11     else
12       v = x.key
13       if (CAS (Head,h,x))
14         return v
15         exit
```



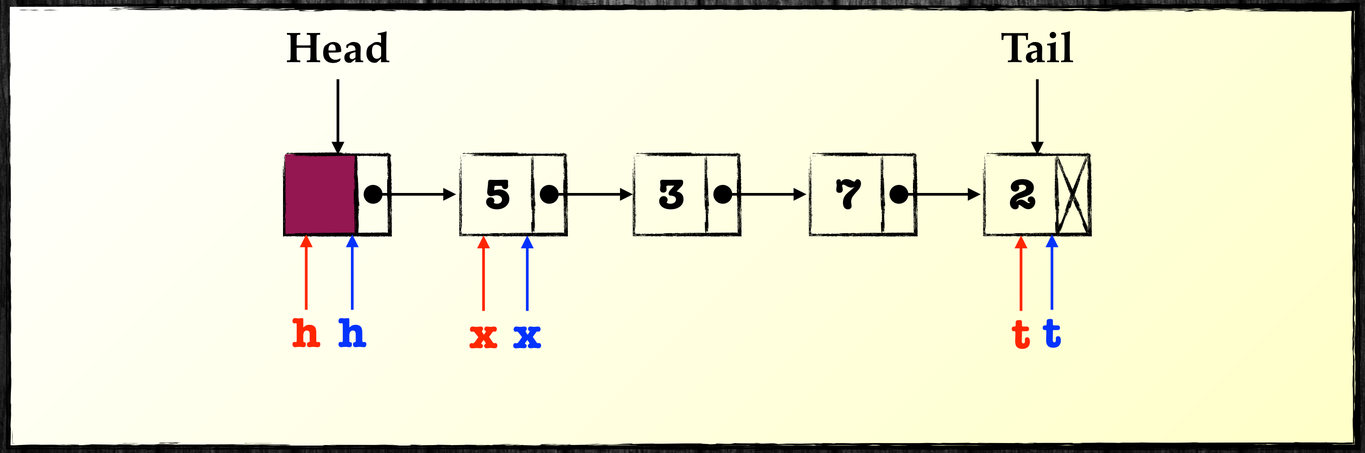
The Michael-Scott Queue Algorithm

Concurrent dequeue operations

```
deq(k):
Node h,t,x
Integer v
1  while (true)
2    h = Head
3    t = Tail
4    x = h.next
5    if (h == Head)
6      if (h == t)
7        if (x == NULL)
8          return *
9          exit
10       CAS (Tail,t,x)
11     else
12       v = x.key
13       if (CAS (Head,h,x))
14         return v
15         exit
```



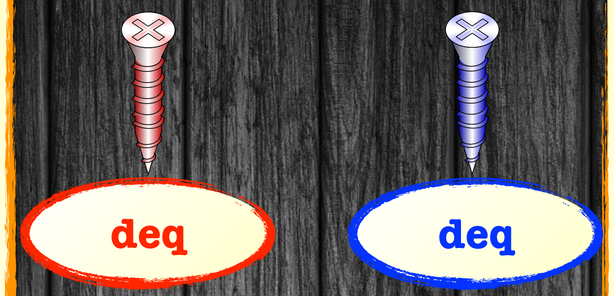

```
deq(k):
Node h,t,x
Integer v
1  while (true)
2    h = Head
3    t = Tail
4    x = h.next
5    if (h == Head)
6      if (h == t)
7        if (x == NULL)
8          return *
9          exit
10       CAS (Tail,t,x)
11     else
12       v = x.key
13       if (CAS (Head,h,x))
14         return v
15         exit
```



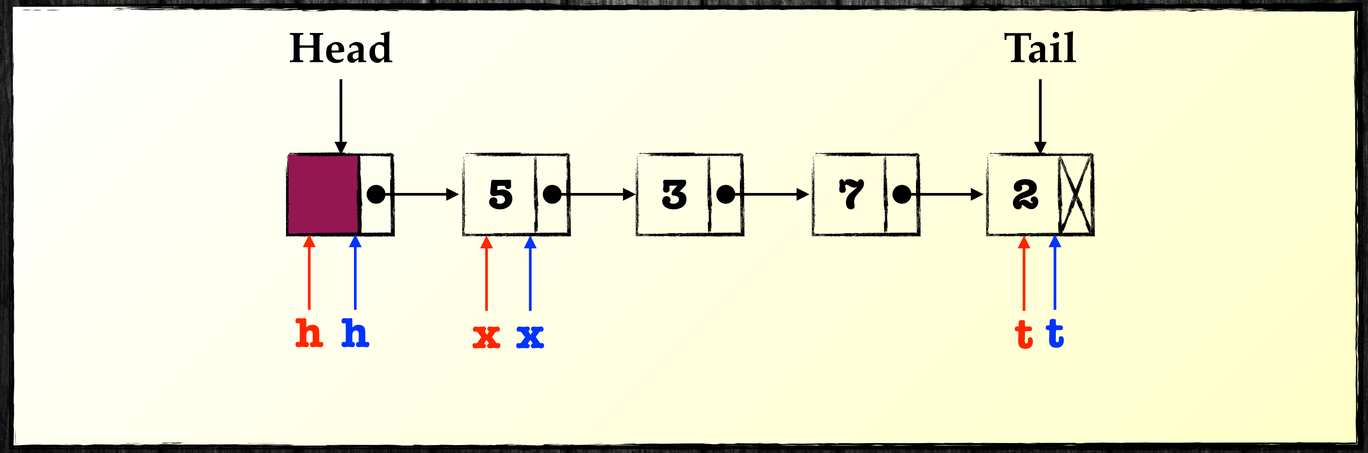
The Michael-Scott Queue Algorithm

Concurrent dequeue operations

```
deq(k):
Node h,t,x
Integer v
1  while (true)
2    h = Head
3    t = Tail
4    x = h.next
5    if (h == Head)
6      if (h == t)
7        if (x == NULL)
8          return *
9          exit
10       CAS (Tail,t,x)
11     else
12       v = x.key
13       if (CAS (Head,h,x))
14         return v
15         exit
```



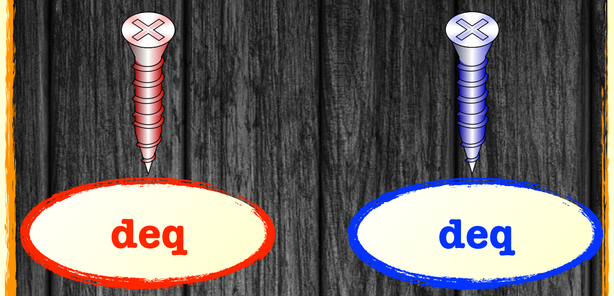

```
deq(k):
Node h,t,x
Integer v
1  while (true)
2    h = Head
3    t = Tail
4    x = h.next
5    if (h == Head)
6      if (h == t)
7        if (x == NULL)
8          return *
9          exit
10       CAS (Tail,t,x)
11     else
12       v = x.key
13       if (CAS (Head,h,x))
14         return v
15         exit
```



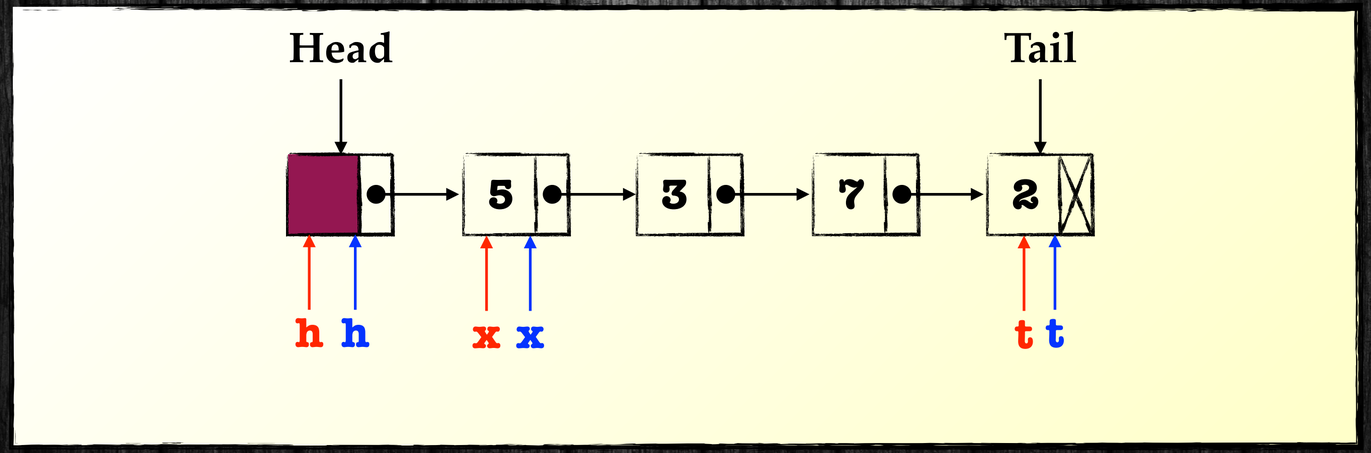
The Michael-Scott Queue Algorithm

Concurrent dequeue operations

```
deq(k):
Node h,t,x
Integer v
1  while (true)
2    h = Head
3    t = Tail
4    x = h.next
5    if (h == Head)
6      if (h == t)
7        if (x == NULL)
8          return *
9          exit
10       CAS (Tail,t,x)
11     else
12       v = x.key
13       if (CAS (Head,h,x))
14         return v
15         exit
```



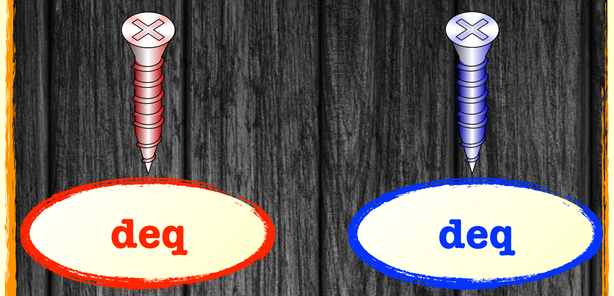

```
deq(k):
Node h,t,x
Integer v
1  while (true)
2    h = Head
3    t = Tail
4    x = h.next
5    if (h == Head)
6      if (h == t)
7        if (x == NULL)
8          return *
9          exit
10       CAS (Tail,t,x)
11     else
12       v = x.key
13       if (CAS (Head,h,x))
14         return v
15         exit
```



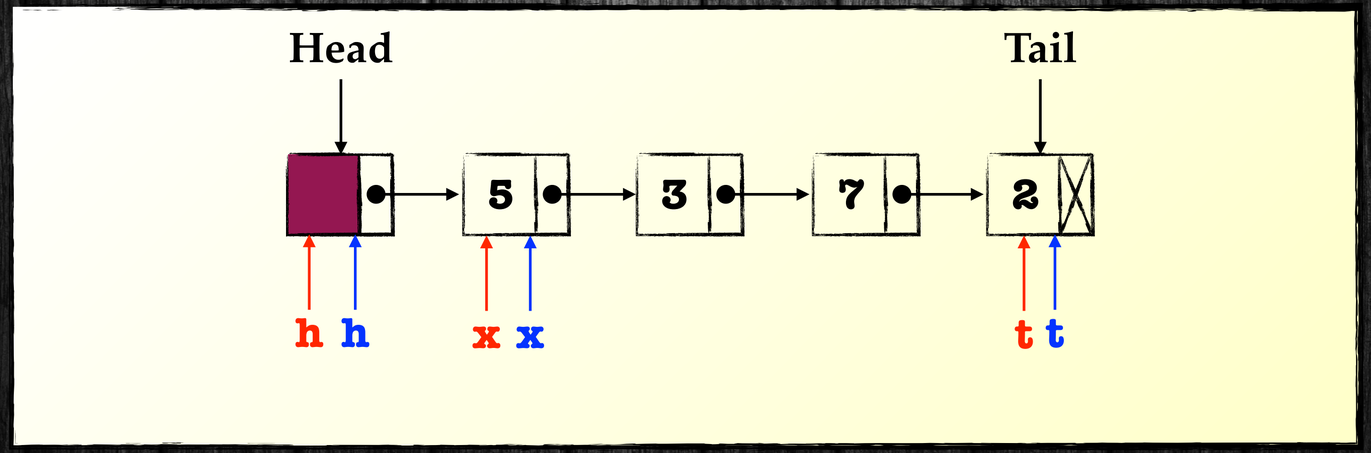
The Michael-Scott Queue Algorithm

Concurrent dequeue operations

```
deq(k):
Node h,t,x
Integer v
1  while (true)
2    h = Head
3    t = Tail
4    x = h.next
5    if (h == Head)
6      if (h == t)
7        if (x == NULL)
8          return *
9          exit
10       CAS (Tail,t,x)
11     else
12       v = x.key
13       if (CAS (Head,h,x))
14         return v
15         exit
```



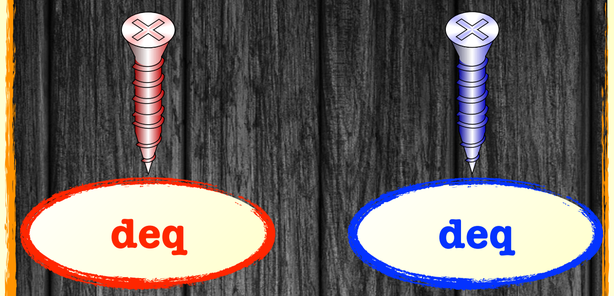

```
deq(k):
Node h,t,x
Integer v
1  while (true)
2    h = Head
3    t = Tail
4    x = h.next
5    if (h == Head)
6      if (h == t)
7        if (x == NULL)
8          return *
9          exit
10       CAS (Tail,t,x)
11     else
12       v = x.key
13       if (CAS (Head,h,x))
14         return v
15         exit
```



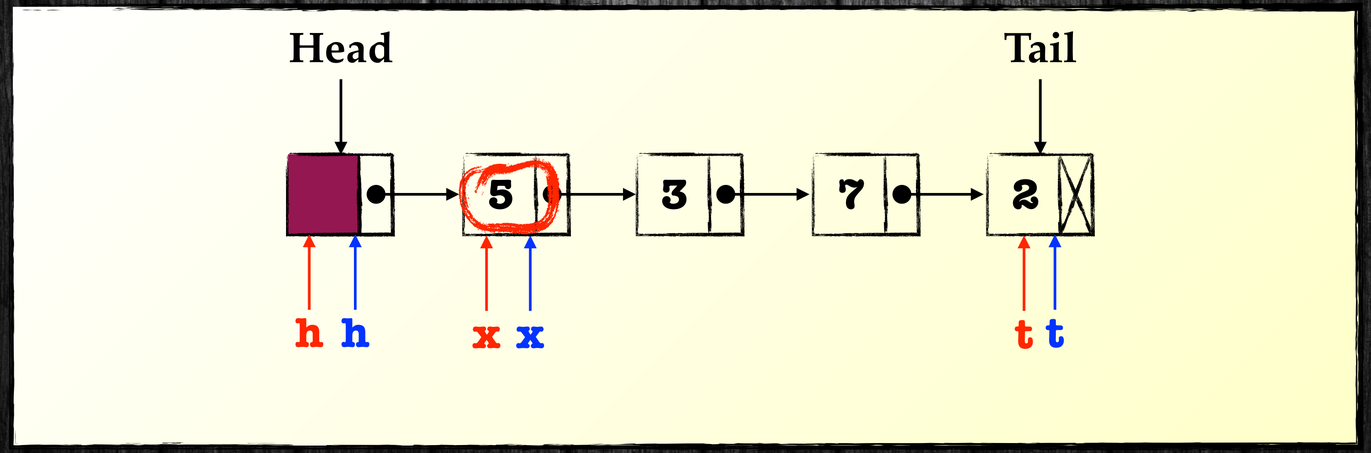
The Michael-Scott Queue Algorithm

Concurrent dequeue operations

```
deq(k):
Node h,t,x
Integer v
1  while (true)
2    h = Head
3    t = Tail
4    x = h.next
5    if (h == Head)
6      if (h == t)
7        if (x == NULL)
8          return *
9          exit
10       CAS (Tail,t,x)
11     else
12       v = x.key
13       if (CAS (Head,h,x))
14         return v
15         exit
```



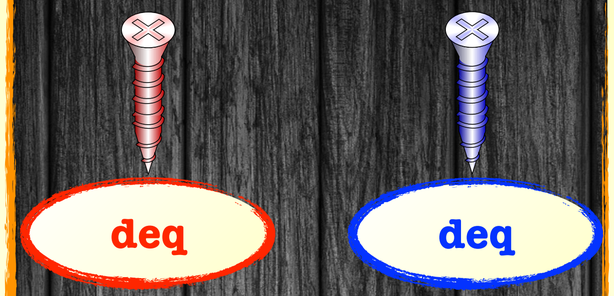

```
deq(k):
Node h,t,x
Integer v
1  while (true)
2    h = Head
3    t = Tail
4    x = h.next
5    if (h == Head)
6      if (h == t)
7        if (x == NULL)
8          return *
9          exit
10       CAS (Tail,t,x)
11     else
12       v = x.key
13       if (CAS (Head,h,x))
14         return v
15         exit
```



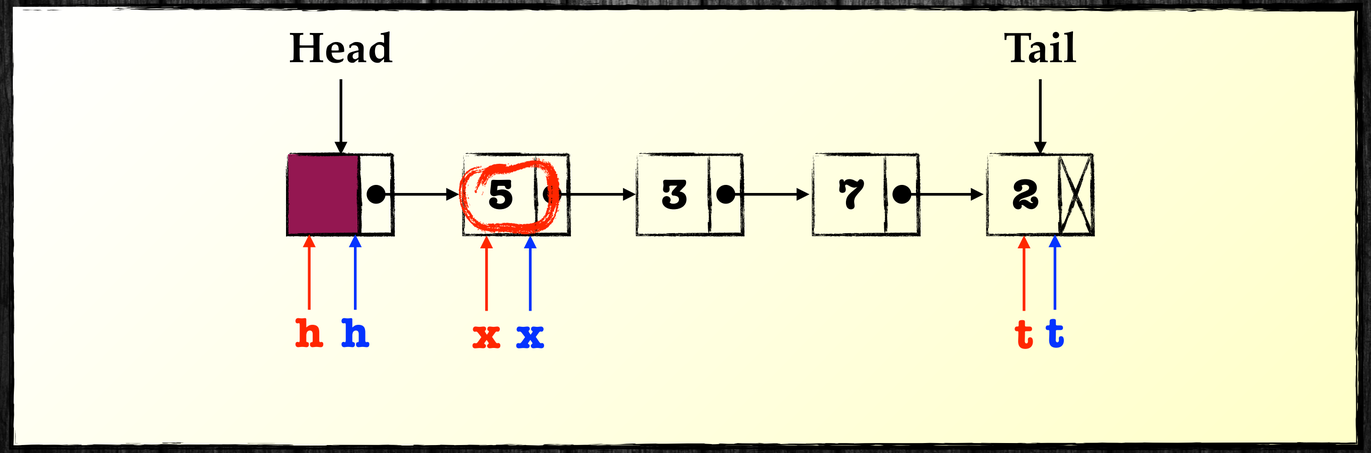
The Michael-Scott Queue Algorithm

Concurrent dequeue operations

```
deq(k):
Node h,t,x
Integer v
1  while (true)
2    h = Head
3    t = Tail
4    x = h.next
5    if (h == Head)
6      if (h == t)
7        if (x == NULL)
8          return *
9          exit
10       CAS (Tail,t,x)
11     else
12       v = x.key
13       if (CAS (Head,h,x))
14         return v
15         exit
```



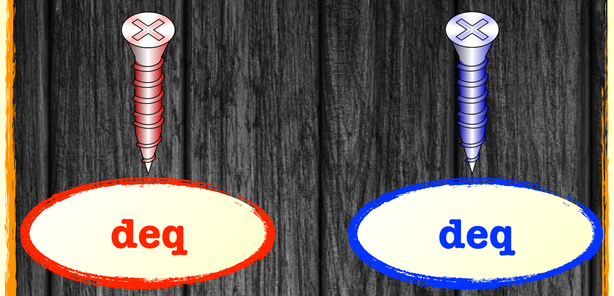

```
deq(k):
Node h,t,x
Integer v
1  while (true)
2    h = Head
3    t = Tail
4    x = h.next
5    if (h == Head)
6      if (h == t)
7        if (x == NULL)
8          return *
9          exit
10       CAS (Tail,t,x)
11     else
12       v = x.key
13       if (CAS (Head,h,x))
14         return v
15         exit
```



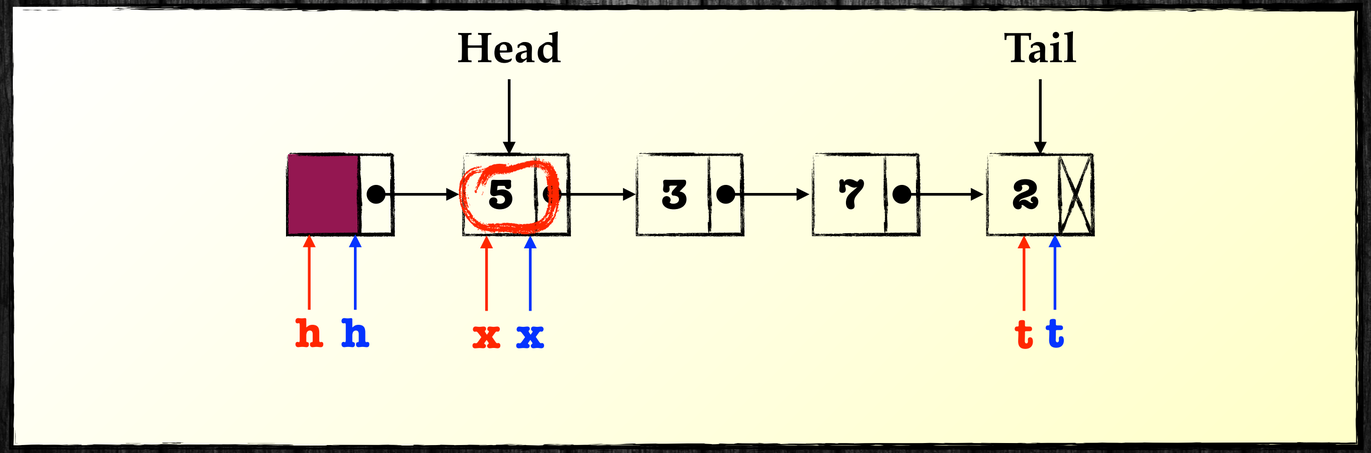
The Michael-Scott Queue Algorithm

Concurrent dequeue operations

```
deq(k):
Node h,t,x
Integer v
1  while (true)
2    h = Head
3    t = Tail
4    x = h.next
5    if (h == Head)
6      if (h == t)
7        if (x == NULL)
8          return *
9          exit
10       CAS (Tail,t,x)
11     else
12       v = x.key
13       if (CAS (Head,h,x))
14         return v
15         exit
```



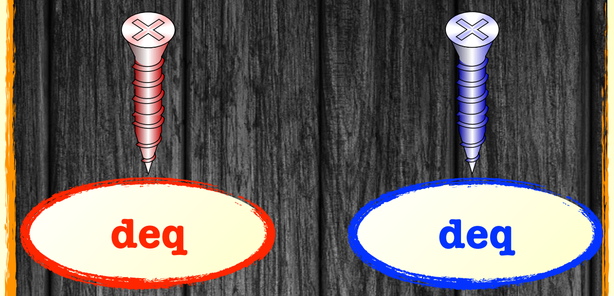

```
deq(k):
Node h,t,x
Integer v
1  while (true)
2    h = Head
3    t = Tail
4    x = h.next
5    if (h == Head)
6      if (h == t)
7        if (x == NULL)
8          return *
9          exit
10       CAS (Tail,t,x)
11     else
12       v = x.key
13       if (CAS (Head,h,x))
14         return v
15         exit
```



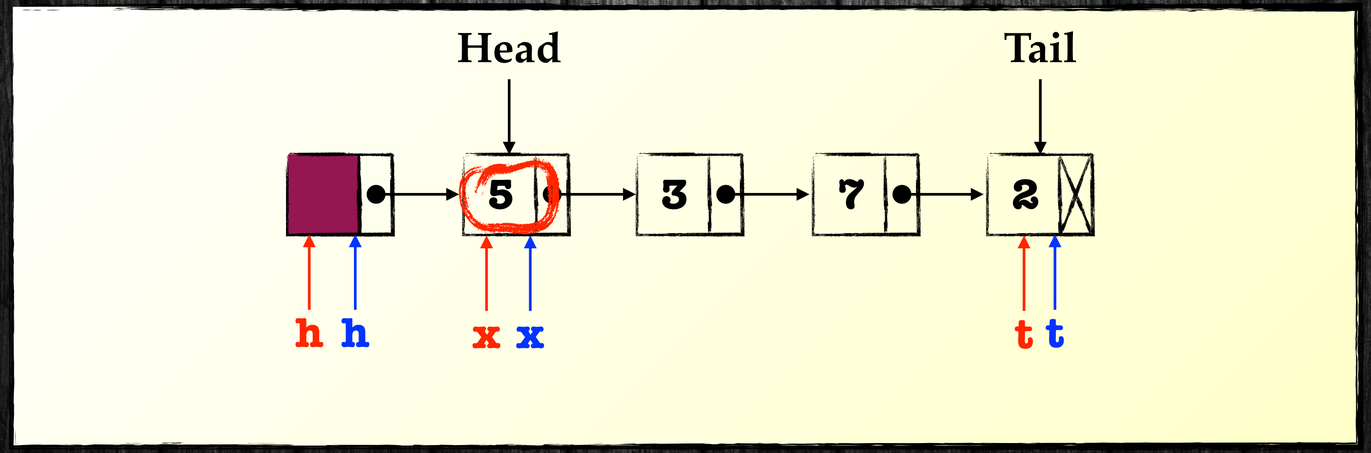
The Michael-Scott Queue Algorithm

Concurrent dequeue operations

```
deq(k):
Node h,t,x
Integer v
1  while (true)
2    h = Head
3    t = Tail
4    x = h.next
5    if (h == Head)
6      if (h == t)
7        if (x == NULL)
8          return *
9          exit
10       CAS (Tail,t,x)
11     else
12       v = x.key
13       if (CAS (Head,h,x))
14         return v
15         exit
```



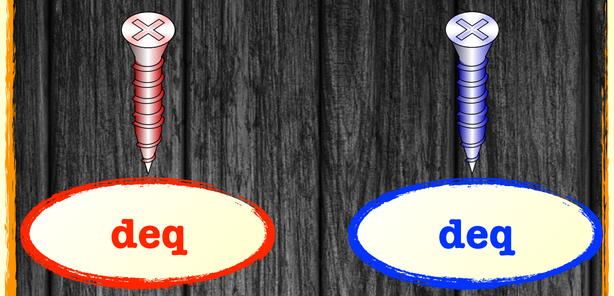

```
deq(k):
Node h,t,x
Integer v
1  while (true)
2    h = Head
3    t = Tail
4    x = h.next
5    if (h == Head)
6      if (h == t)
7        if (x == NULL)
8          return *
9          exit
10       CAS (Tail,t,x)
11     else
12       v = x.key
13       if (CAS (Head,h,x))
14         return v
15         exit
```



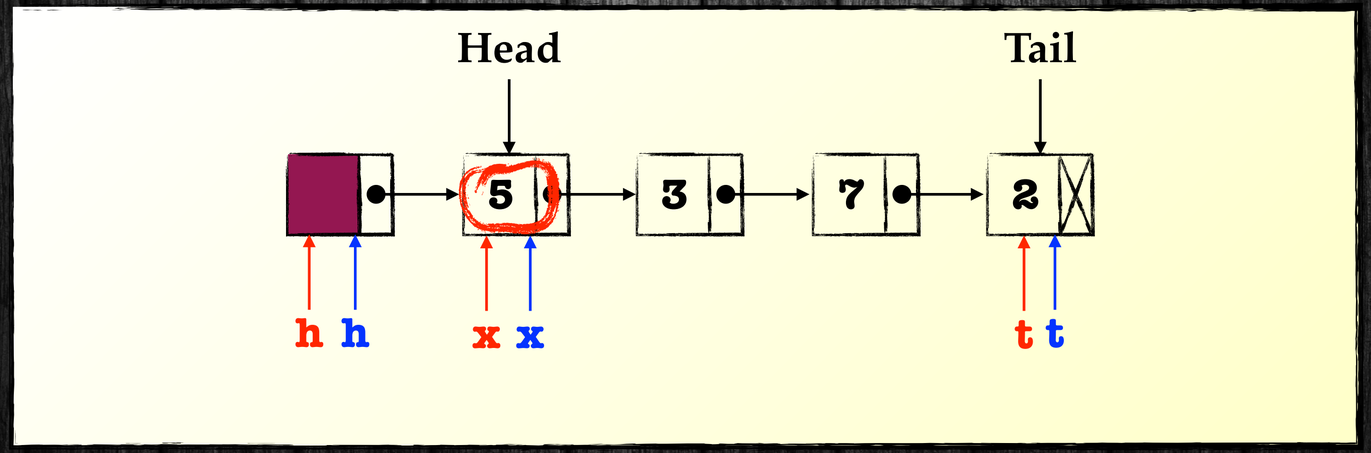
The Michael-Scott Queue Algorithm

Concurrent dequeue operations

```
deq(k):
Node h,t,x
Integer v
1  while (true)
2    h = Head
3    t = Tail
4    x = h.next
5    if (h == Head)
6      if (h == t)
7        if (x == NULL)
8          return *
9          exit
10       CAS (Tail,t,x)
11     else
12       v = x.key
13       if (CAS (Head,h,x))
14         return v
15         exit
```



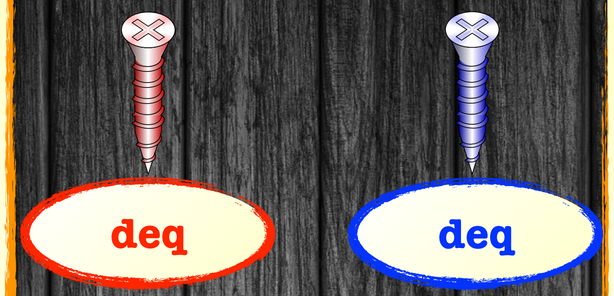

```
deq(k):
Node h,t,x
Integer v
1  while (true)
2    h = Head
3    t = Tail
4    x = h.next
5    if (h == Head)
6      if (h == t)
7        if (x == NULL)
8          return *
9          exit
10       CAS (Tail,t,x)
11     else
12       v = x.key
13       if (CAS (Head,h,x))
14         return v
15         exit
```



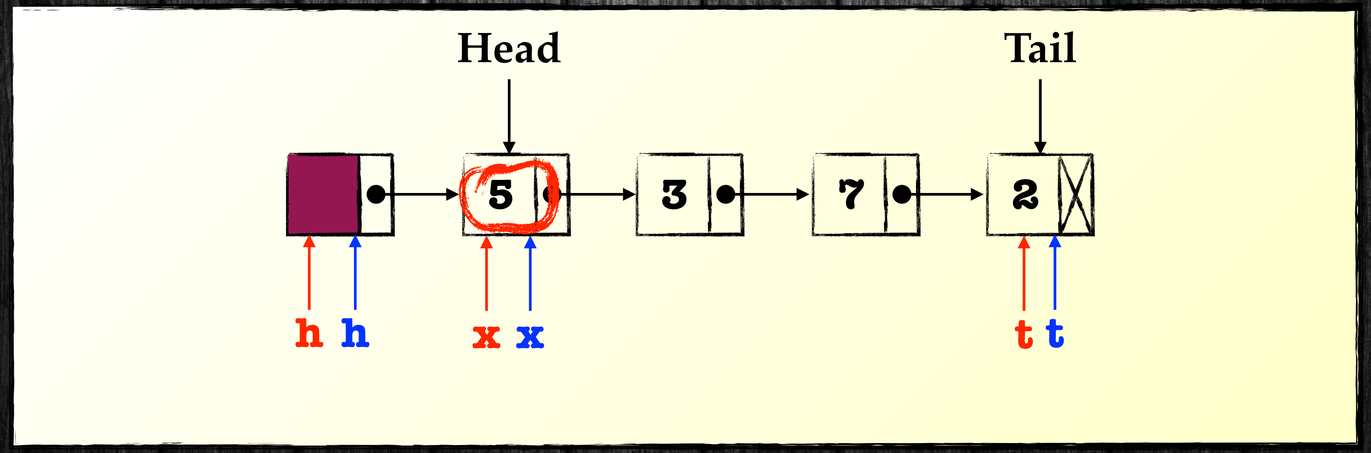
The Michael-Scott Queue Algorithm

Concurrent dequeue operations

```
deq(k):
Node h,t,x
Integer v
1  while (true)
2    h = Head
3    t = Tail
4    x = h.next
5    if (h == Head)
6      if (h == t)
7        if (x == NULL)
8          return *
9          exit
10       CAS (Tail,t,x)
11     else
12       v = x.key
13       if (CAS (Head,h,x))
14         return v
15         exit
```



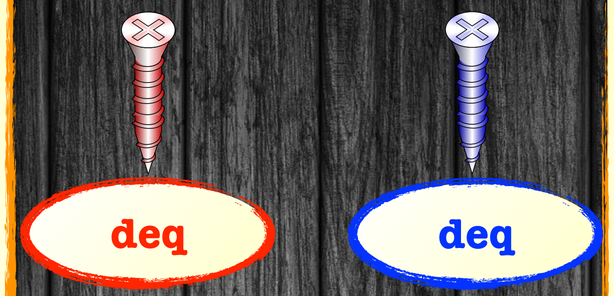

```
deq(k):
Node h,t,x
Integer v
1  while (true)
2    h = Head
3    t = Tail
4    x = h.next
5    if (h == Head)
6      if (h == t)
7        if (x == NULL)
8          return *
9          exit
10       CAS (Tail,t,x)
11     else
12       v = x.key
13       if (CAS (Head,h,x))
14         return v
15         exit
```



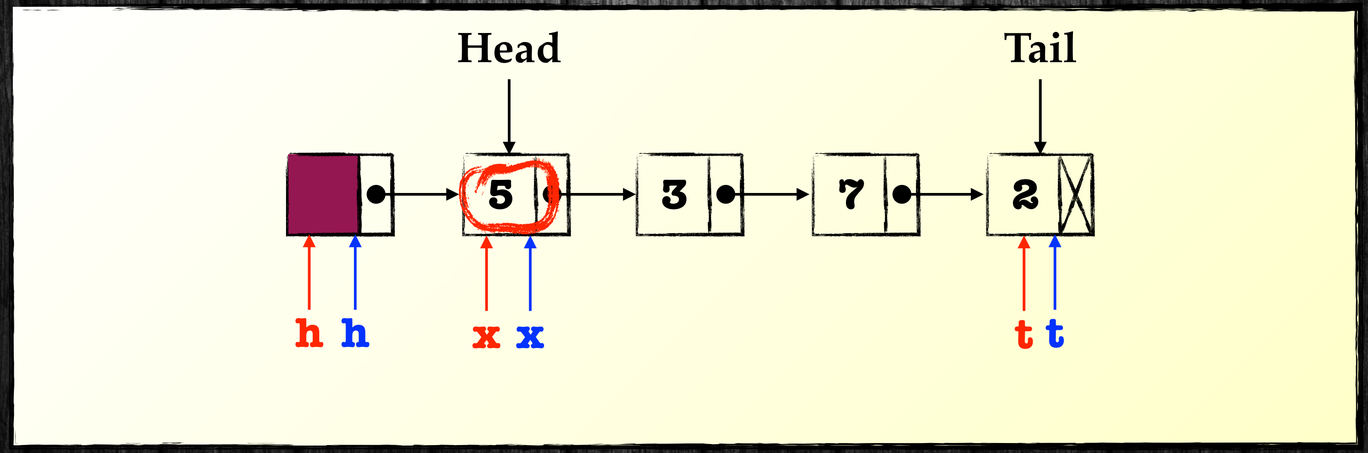
The Michael-Scott Queue Algorithm

Concurrent dequeue operations

```
deq(k):
Node h,t,x
Integer v
1  while (true)
2    h = Head
3    t = Tail
4    x = h.next
5    if (h == Head)
6      if (h == t)
7        if (x == NULL)
8          return *
9          exit
10       CAS (Tail,t,x)
11     else
12       v = x.key
13       if (CAS (Head,h,x))
14         return v
15         exit
```




```
deq(k):
Node h,t,x
Integer v
1  while (true)
2    h = Head
3    t = Tail
4    x = h.next
5    if (h == Head)
6      if (h == t)
7        if (x == NULL)
8          return *
9          exit
10       CAS (Tail,t,x)
11     else
12       v = x.key
13       if (CAS (Head,h,x))
14         return v
15         exit
```



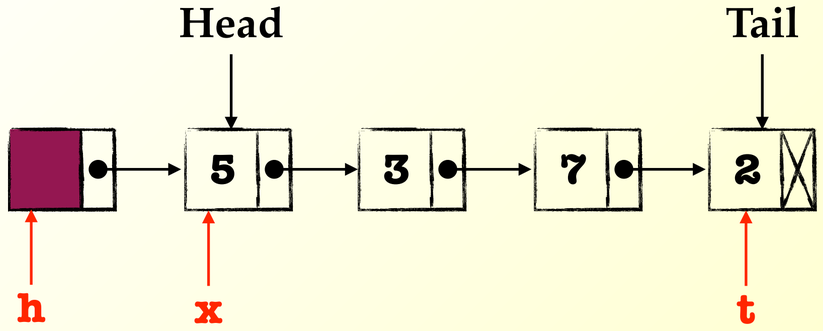
The Michael-Scott Queue Algorithm

Concurrent dequeue operations

```
deq(k):
Node h,t,x
Integer v
1  while (true)
2    h = Head
3    t = Tail
4    x = h.next
5    if (h == Head)
6      if (h == t)
7        if (x == NULL)
8          return *
9          exit
10       CAS (Tail,t,x)
11     else
12       v = x.key
13       if (CAS (Head,h,x))
14         return v
15         exit
```




```
deq(k):
Node h,t,x
Integer v
1  while (true)
2    h = Head
3    t = Tail
4    x = h.next
5    if (h == Head)
6      if (h == t)
7        if (x == NULL)
8          return *
9          exit
10       CAS (Tail,t,x)
11     else
12       v = x.key
13       if (CAS (Head,h,x))
14         return v
15         exit
```



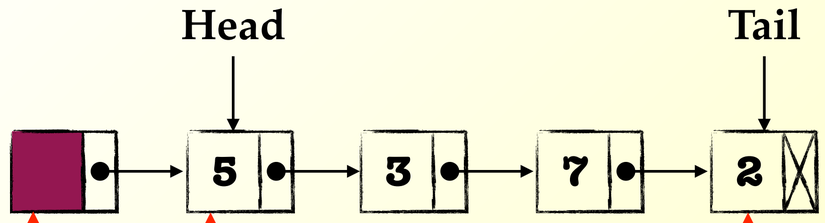
The Michael-Scott Queue Algorithm

Concurrent dequeue operations

```
deq(k):
Node h,t,x
Integer v
1  while (true)
2    h = Head
3    t = Tail
4    x = h.next
5    if (h == Head)
6      if (h == t)
7        if (x == NULL)
8          return *
9          exit
10       CAS (Tail,t,x)
11     else
12       v = x.key
13       if (CAS (Head,h,x))
14         return v
15         exit
```




```
deq(k):
Node h,t,x
Integer v
1  while (true)
2    h = Head
3    t = Tail
4    x = h.next
5    if (h == Head)
6      if (h == t)
7        if (x == NULL)
8          return *
9          exit
10       CAS (Tail,t,x)
11     else
12       v = x.key
13       if (CAS (Head,h,x))
14         return v
15         exit
```



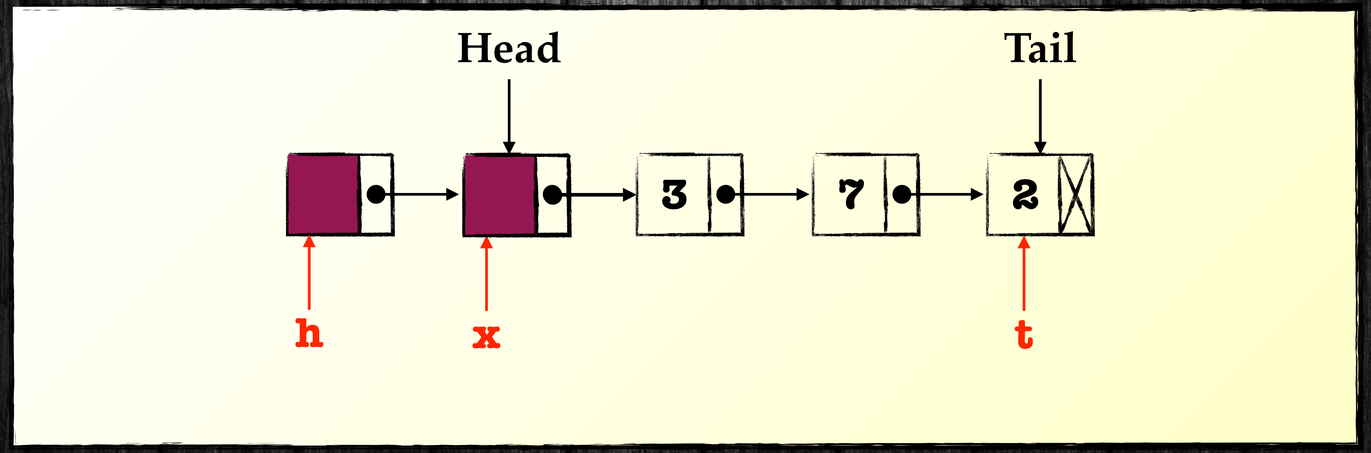
The Michael-Scott Queue Algorithm

Concurrent dequeue operations

```
deq(k):
Node h,t,x
Integer v
1  while (true)
2    h = Head
3    t = Tail
4    x = h.next
5    if (h == Head)
6      if (h == t)
7        if (x == NULL)
8          return *
9          exit
10       CAS (Tail,t,x)
11     else
12       v = x.key
13       if (CAS (Head,h,x))
14         return v
15         exit
```




```
deq(k):
Node h,t,x
Integer v
1  while (true)
2    h = Head
3    t = Tail
4    x = h.next
5    if (h == Head)
6      if (h == t)
7        if (x == NULL)
8          return *
9          exit
10       CAS (Tail,t,x)
11     else
12       v = x.key
13       if (CAS (Head,h,x))
14         return v
15         exit
```



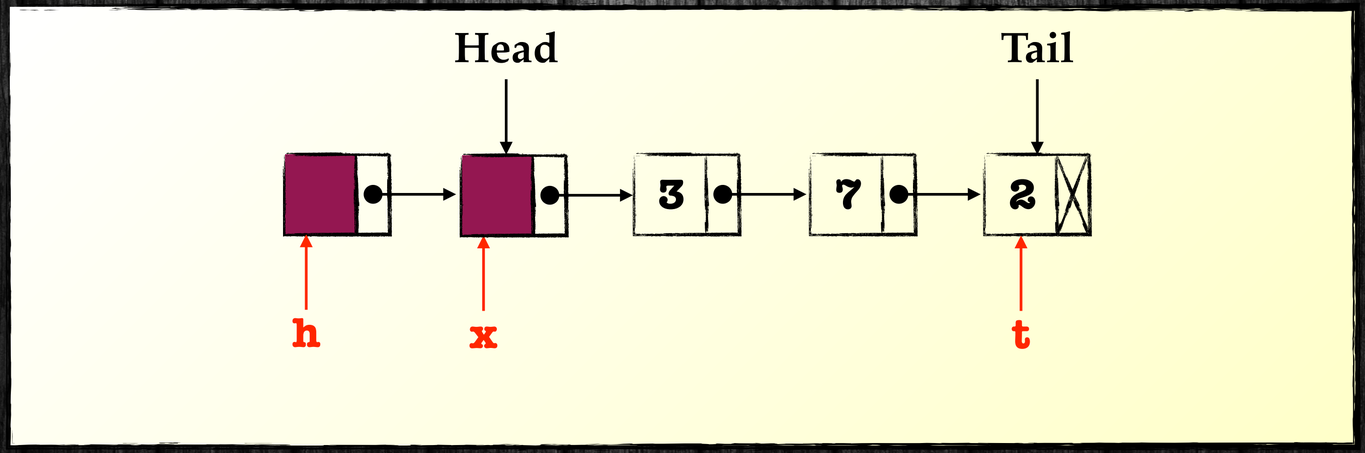
The Michael-Scott Queue Algorithm

Concurrent dequeue operations

```
deq(k):
Node h,t,x
Integer v
1  while (true)
2    h = Head
3    t = Tail
4    x = h.next
5    if (h == Head)
6      if (h == t)
7        if (x == NULL)
8          return *
9          exit
10       CAS (Tail,t,x)
11     else
12       v = x.key
13       if (CAS (Head,h,x))
14         return v
15         exit
```




```
deq(k):
Node h,t,x
Integer v
1  while (true)
2    h = Head
3    t = Tail
4    x = h.next
5    if (h == Head)
6      if (h == t)
7        if (x == NULL)
8          return *
9          exit
10       CAS (Tail,t,x)
11     else
12       v = x.key
13       if (CAS (Head,h,x))
14         return v
15         exit
```



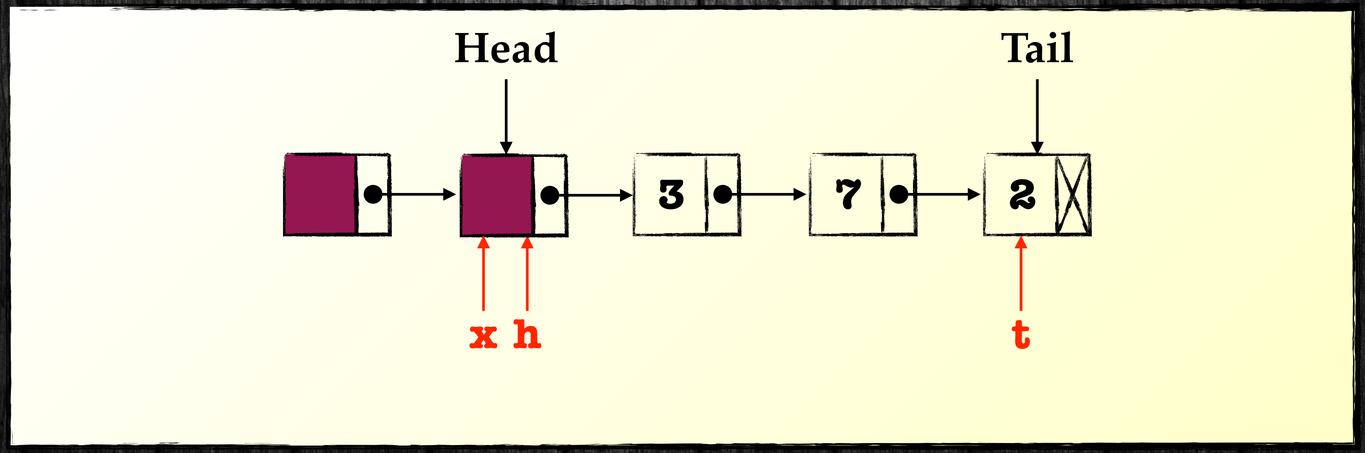
The Michael-Scott Queue Algorithm

Concurrent dequeue operations

```
deq(k):
Node h,t,x
Integer v
1  while (true)
2    h = Head
3    t = Tail
4    x = h.next
5    if (h == Head)
6      if (h == t)
7        if (x == NULL)
8          return *
9          exit
10       CAS (Tail,t,x)
11     else
12       v = x.key
13       if (CAS (Head,h,x))
14         return v
15         exit
```




```
deq(k):
Node h,t,x
Integer v
1  while (true)
2    h = Head
3    t = Tail
4    x = h.next
5    if (h == Head)
6      if (h == t)
7        if (x == NULL)
8          return *
9          exit
10       CAS (Tail,t,x)
11     else
12       v = x.key
13       if (CAS (Head,h,x))
14         return v
15         exit
```



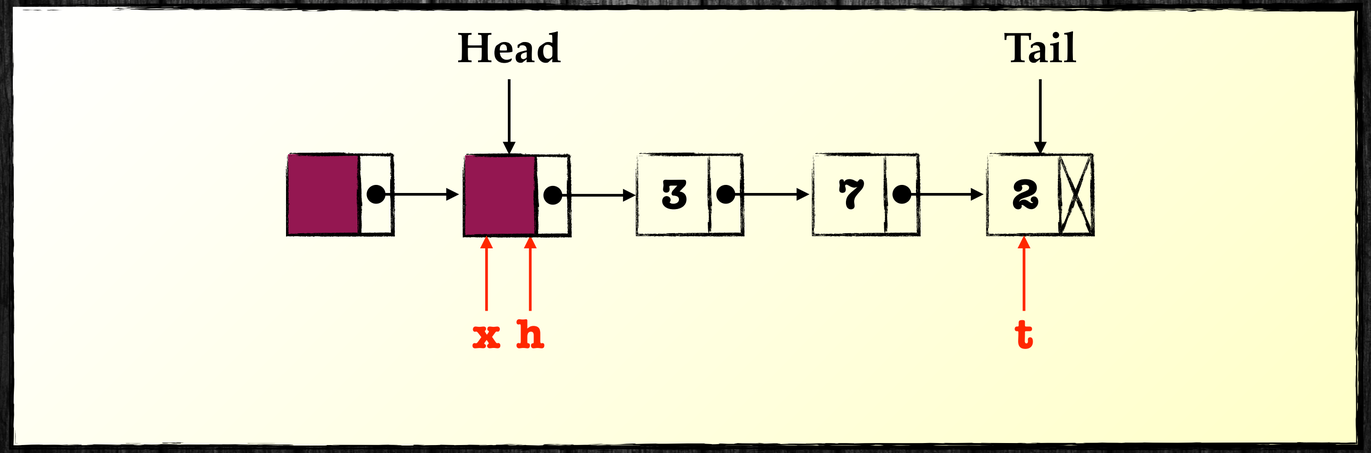
The Michael-Scott Queue Algorithm

Concurrent dequeue operations

```
deq(k):
Node h,t,x
Integer v
1  while (true)
2    h = Head
3    t = Tail
4    x = h.next
5    if (h == Head)
6      if (h == t)
7        if (x == NULL)
8          return *
9          exit
10       CAS (Tail,t,x)
11     else
12       v = x.key
13       if (CAS (Head,h,x))
14         return v
15         exit
```




```
deq(k):
Node h,t,x
Integer v
1  while (true)
2    h = Head
3    t = Tail
4    x = h.next
5    if (h == Head)
6      if (h == t)
7        if (x == NULL)
8          return *
9          exit
10       CAS (Tail,t,x)
11     else
12       v = x.key
13       if (CAS (Head,h,x))
14         return v
15         exit
```



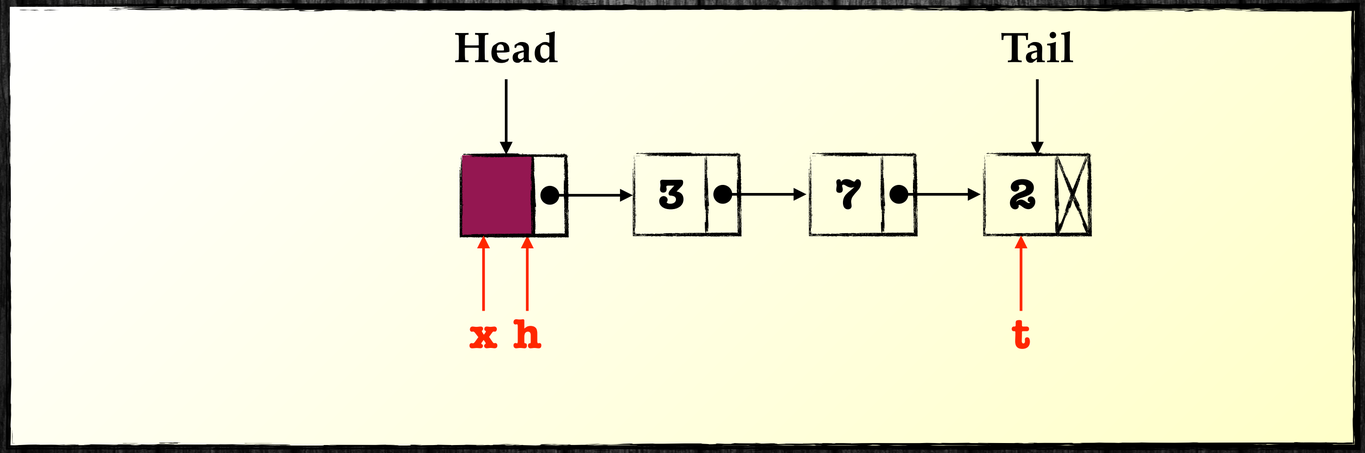
The Michael-Scott Queue Algorithm

Concurrent dequeue operations

```
deq(k):
Node h,t,x
Integer v
1  while (true)
2    h = Head
3    t = Tail
4    x = h.next
5    if (h == Head)
6      if (h == t)
7        if (x == NULL)
8          return *
9          exit
10       CAS (Tail,t,x)
11     else
12       v = x.key
13       if (CAS (Head,h,x))
14         return v
15         exit
```




```
deq(k):
Node h,t,x
Integer v
1  while (true)
2    h = Head
3    t = Tail
4    x = h.next
5    if (h == Head)
6      if (h == t)
7        if (x == NULL)
8          return *
9          exit
10       CAS (Tail,t,x)
11     else
12       v = x.key
13       if (CAS (Head,h,x))
14         return v
15         exit
```



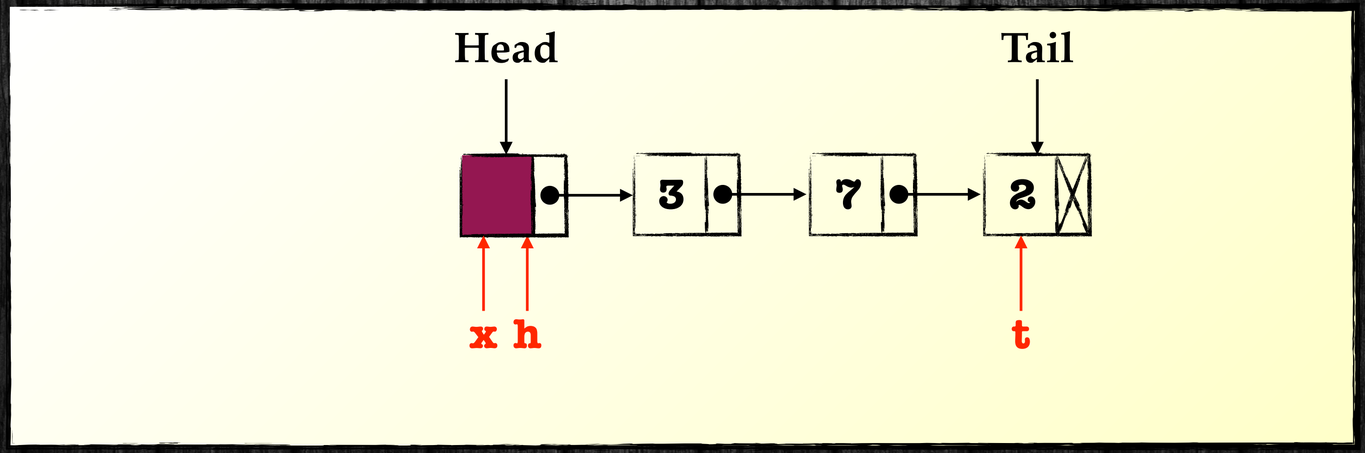
The Michael-Scott Queue Algorithm

Concurrent dequeue operations

```
deq(k):
Node h,t,x
Integer v
1  while (true)
2    h = Head
3    t = Tail
4    x = h.next
5    if (h == Head)
6      if (h == t)
7        if (x == NULL)
8          return *
9          exit
10       CAS (Tail,t,x)
11     else
12       v = x.key
13       if (CAS (Head,h,x))
14         return v
15         exit
```




```
deq(k):
Node h,t,x
Integer v
1  while (true)
2    h = Head
3    t = Tail
4    x = h.next
5    if (h == Head)
6      if (h == t)
7        if (x == NULL)
8          return *
9          exit
10       CAS (Tail,t,x)
11     else
12       v = x.key
13       if (CAS (Head,h,x))
14         return v
15         exit
```



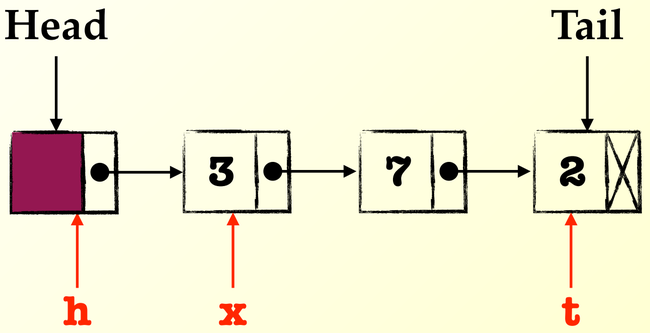
The Michael-Scott Queue Algorithm

Concurrent dequeue operations

```
deq(k):
Node h,t,x
Integer v
1  while (true)
2    h = Head
3    t = Tail
4    x = h.next
5    if (h == Head)
6      if (h == t)
7        if (x == NULL)
8          return *
9          exit
10       CAS (Tail,t,x)
11     else
12       v = x.key
13       if (CAS (Head,h,x))
14         return v
15         exit
```




```
deq(k):
Node h,t,x
Integer v
1  while (true)
2    h = Head
3    t = Tail
4    x = h.next
5    if (h == Head)
6      if (h == t)
7        if (x == NULL)
8          return *
9          exit
10       CAS (Tail,t,x)
11     else
12       v = x.key
13       if (CAS (Head,h,x))
14         return v
15         exit
```



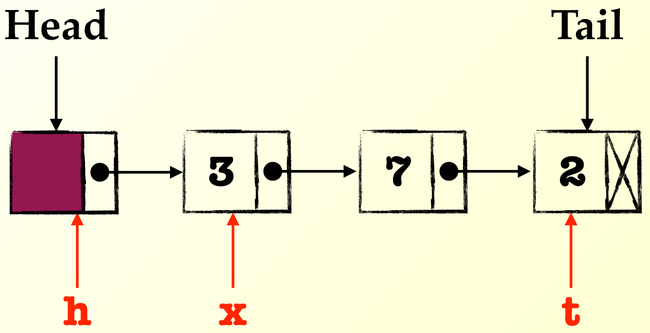
The Michael-Scott Queue Algorithm

Concurrent dequeue operations

```
deq(k):
Node h,t,x
Integer v
1  while (true)
2    h = Head
3    t = Tail
4    x = h.next
5    if (h == Head)
6      if (h == t)
7        if (x == NULL)
8          return *
9          exit
10       CAS (Tail,t,x)
11     else
12       v = x.key
13       if (CAS (Head,h,x))
14         return v
15         exit
```




```
deq(k):
Node h,t,x
Integer v
1  while (true)
2    h = Head
3    t = Tail
4    x = h.next
5    if (h == Head)
6      if (h == t)
7        if (x == NULL)
8          return *
9          exit
10       CAS (Tail,t,x)
11     else
12       v = x.key
13       if (CAS (Head,h,x))
14         return v
15         exit
```



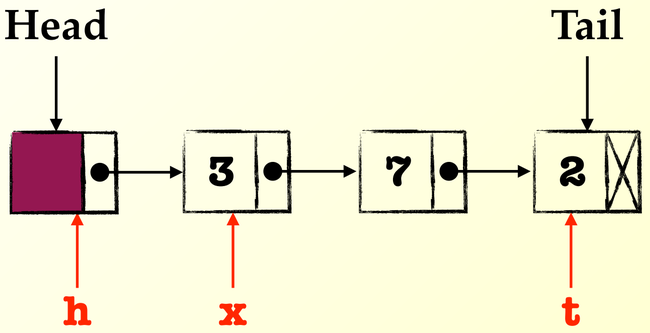
The Michael-Scott Queue Algorithm

Concurrent dequeue operations

```
deq(k):
Node h,t,x
Integer v
1  while (true)
2    h = Head
3    t = Tail
4    x = h.next
5    if (h == Head)
6      if (h == t)
7        if (x == NULL)
8          return *
9          exit
10       CAS (Tail,t,x)
11     else
12       v = x.key
13       if (CAS (Head,h,x))
14         return v
15         exit
```




```
deq(k):
Node h,t,x
Integer v
1  while (true)
2    h = Head
3    t = Tail
4    x = h.next
5    if (h == Head)
6      if (h == t)
7        if (x == NULL)
8          return *
9          exit
10       CAS (Tail,t,x)
11     else
12       v = x.key
13       if (CAS (Head,h,x))
14         return v
15         exit
```



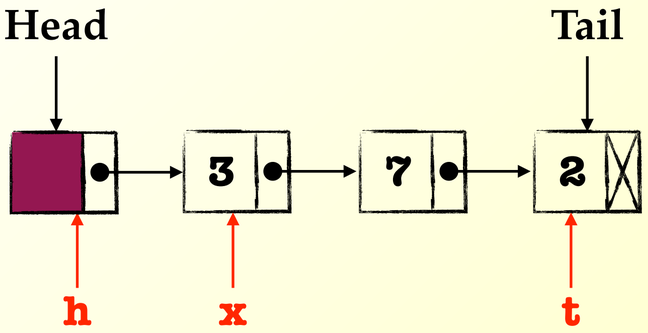
The Michael-Scott Queue Algorithm

Concurrent dequeue operations

```
deq(k):
Node h,t,x
Integer v
1  while (true)
2    h = Head
3    t = Tail
4    x = h.next
5    if (h == Head)
6      if (h == t)
7        if (x == NULL)
8          return *
9          exit
10       CAS (Tail,t,x)
11     else
12       v = x.key
13       if (CAS (Head,h,x))
14         return v
15         exit
```




```
deq(k):
Node h,t,x
Integer v
1  while (true)
2    h = Head
3    t = Tail
4    x = h.next
5    if (h == Head)
6      if (h == t)
7        if (x == NULL)
8          return *
9          exit
10       CAS (Tail,t,x)
11     else
12       v = x.key
13       if (CAS (Head,h,x))
14         return v
15         exit
```



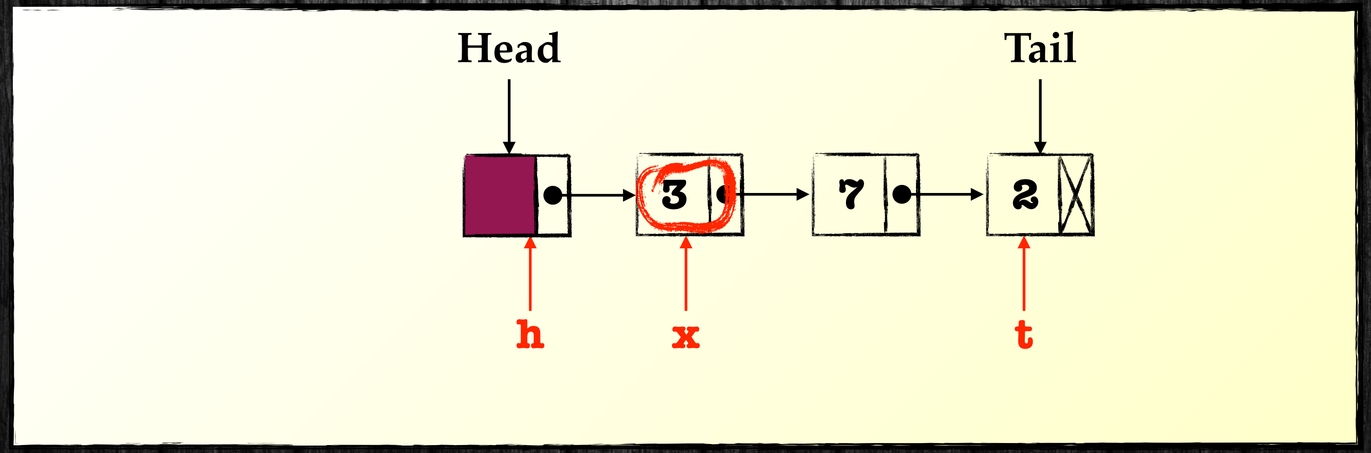
The Michael-Scott Queue Algorithm

Concurrent dequeue operations

```
deq(k):
Node h,t,x
Integer v
1  while (true)
2    h = Head
3    t = Tail
4    x = h.next
5    if (h == Head)
6      if (h == t)
7        if (x == NULL)
8          return *
9          exit
10       CAS (Tail,t,x)
11     else
12       v = x.key
13       if (CAS (Head,h,x))
14         return v
15         exit
```




```
deq(k):
Node h,t,x
Integer v
1  while (true)
2    h = Head
3    t = Tail
4    x = h.next
5    if (h == Head)
6      if (h == t)
7        if (x == NULL)
8          return *
9          exit
10       CAS (Tail,t,x)
11     else
12       v = x.key
13       if (CAS (Head,h,x))
14         return v
15         exit
```



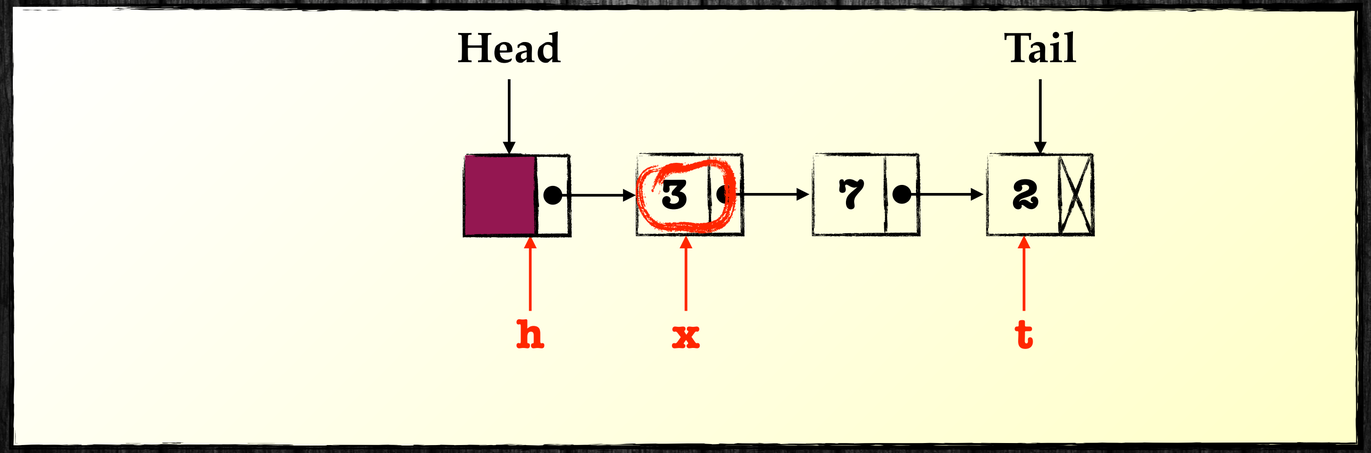
The Michael-Scott Queue Algorithm

Concurrent dequeue operations

```
deq(k):
Node h,t,x
Integer v
1  while (true)
2    h = Head
3    t = Tail
4    x = h.next
5    if (h == Head)
6      if (h == t)
7        if (x == NULL)
8          return *
9          exit
10       CAS (Tail,t,x)
11     else
12       v = x.key
13       if (CAS (Head,h,x))
14         return v
15         exit
```




```
deq(k):
Node h,t,x
Integer v
1  while (true)
2    h = Head
3    t = Tail
4    x = h.next
5    if (h == Head)
6      if (h == t)
7        if (x == NULL)
8          return *
9          exit
10       CAS (Tail,t,x)
11     else
12       v = x.key
13       if (CAS (Head,h,x))
14         return v
15         exit
```



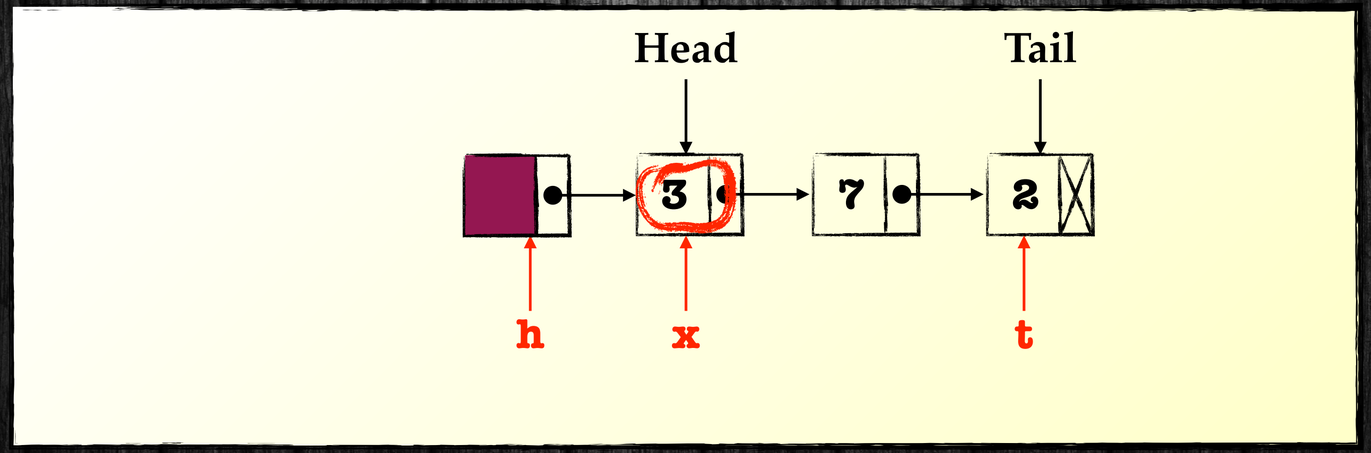
The Michael-Scott Queue Algorithm

Concurrent dequeue operations

```
deq(k):
Node h,t,x
Integer v
1  while (true)
2    h = Head
3    t = Tail
4    x = h.next
5    if (h == Head)
6      if (h == t)
7        if (x == NULL)
8          return *
9          exit
10       CAS (Tail,t,x)
11     else
12       v = x.key
13       if (CAS (Head,h,x))
14         return v
15         exit
```




```
deq(k):
Node h,t,x
Integer v
1  while (true)
2    h = Head
3    t = Tail
4    x = h.next
5    if (h == Head)
6      if (h == t)
7        if (x == NULL)
8          return *
9          exit
10       CAS (Tail,t,x)
11     else
12       v = x.key
13       if (CAS (Head,h,x))
14         return v
15         exit
```



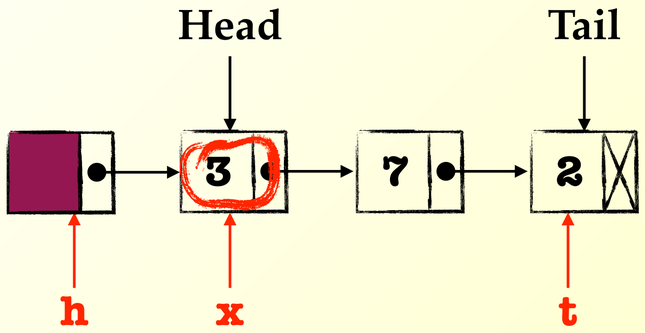
The Michael-Scott Queue Algorithm

Concurrent dequeue operations

```
deq(k):
Node h,t,x
Integer v
1  while (true)
2    h = Head
3    t = Tail
4    x = h.next
5    if (h == Head)
6      if (h == t)
7        if (x == NULL)
8          return *
9          exit
10       CAS (Tail,t,x)
11     else
12       v = x.key
13       if (CAS (Head,h,x))
14         return v
15         exit
```




```
deq(k):
Node h,t,x
Integer v
1  while (true)
2    h = Head
3    t = Tail
4    x = h.next
5    if (h == Head)
6      if (h == t)
7        if (x == NULL)
8          return *
9          exit
10       CAS (Tail,t,x)
11     else
12       v = x.key
13       if (CAS (Head,h,x))
14         return v
15         exit
```



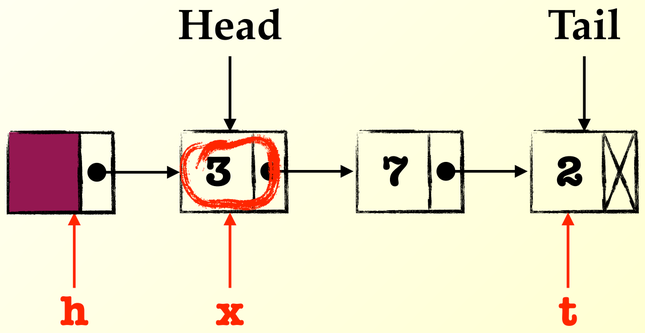
The Michael-Scott Queue Algorithm

Concurrent dequeue operations

```
deq(k):
Node h,t,x
Integer v
1  while (true)
2    h = Head
3    t = Tail
4    x = h.next
5    if (h == Head)
6      if (h == t)
7        if (x == NULL)
8          return *
9          exit
10       CAS (Tail,t,x)
11     else
12       v = x.key
13       if (CAS (Head,h,x))
14         return v
15         exit
```




```
deq(k):
Node h,t,x
Integer v
1  while (true)
2    h = Head
3    t = Tail
4    x = h.next
5    if (h == Head)
6      if (h == t)
7        if (x == NULL)
8          return *
9          exit
10       CAS (Tail,t,x)
11     else
12       v = x.key
13       if (CAS (Head,h,x))
14         return v
15         exit
```

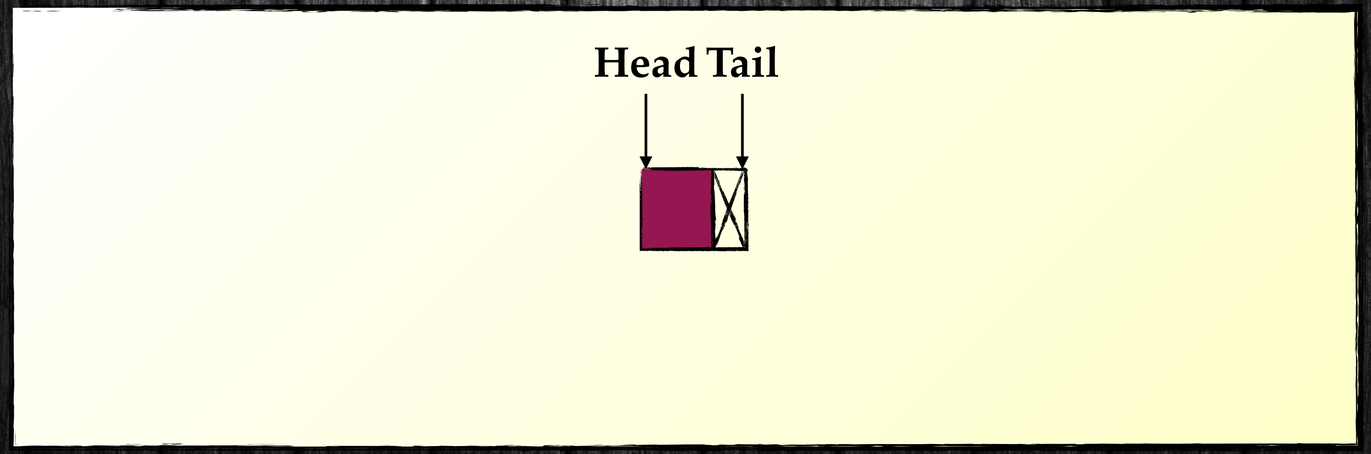


The Michael-Scott Queue Algorithm

Empty queue

```
deq(k):
Node h,t,x
Integer v
1  while (true)
2    h = Head
3    t = Tail
4    x = h.next
5    if (h == Head)
6      if (h == t)
7        if (x == NULL)
8          return *
9          exit
10       CAS (Tail,t,x)
11     else
12       v = x.key
13       if (CAS (Head,h,x))
14         return v
15         exit
```



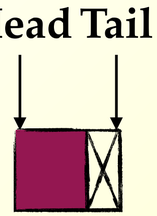


The Michael-Scott Queue Algorithm

Empty queue

```
deq(k):
Node h,t,x
Integer v
1  while (true)
2    h = Head
3    t = Tail
4    x = h.next
5    if (h == Head)
6      if (h == t)
7        if (x == NULL)
8          return *
9          exit
10       CAS (Tail,t,x)
11     else
12       v = x.key
13       if (CAS (Head,h,x))
14         return v
15         exit
```



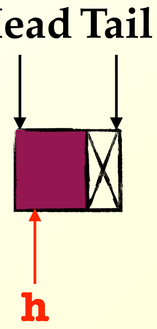


The Michael-Scott Queue Algorithm

Empty queue

```
deq(k):
Node h,t,x
Integer v
1  while (true)
2    h = Head
3    t = Tail
4    x = h.next
5    if (h == Head)
6      if (h == t)
7        if (x == NULL)
8          return *
9          exit
10       CAS (Tail,t,x)
11     else
12       v = x.key
13       if (CAS (Head,h,x))
14         return v
15         exit
```



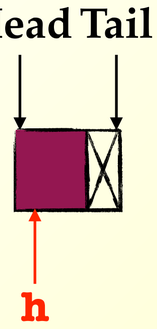


The Michael-Scott Queue Algorithm

Empty queue

```
deq(k):
Node h,t,x
Integer v
1  while (true)
2    h = Head
3    t = Tail
4    x = h.next
5    if (h == Head)
6      if (h == t)
7        if (x == NULL)
8          return *
9          exit
10       CAS (Tail,t,x)
11     else
12       v = x.key
13       if (CAS (Head,h,x))
14         return v
15         exit
```



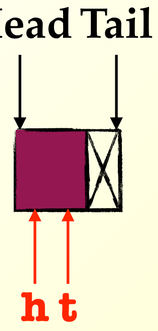


The Michael-Scott Queue Algorithm

Empty queue

```
deq(k):
Node h,t,x
Integer v
1  while (true)
2    h = Head
3    t = Tail
4    x = h.next
5    if (h == Head)
6      if (h == t)
7        if (x == NULL)
8          return *
9          exit
10       CAS (Tail,t,x)
11     else
12       v = x.key
13       if (CAS (Head,h,x))
14         return v
15         exit
```



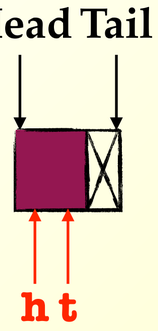


The Michael-Scott Queue Algorithm

Empty queue

```
deq(k):
Node h,t,x
Integer v
1  while (true)
2    h = Head
3    t = Tail
4    x = h.next
5    if (h == Head)
6      if (h == t)
7        if (x == NULL)
8          return *
9          exit
10       CAS (Tail,t,x)
11     else
12       v = x.key
13       if (CAS (Head,h,x))
14         return v
15         exit
```



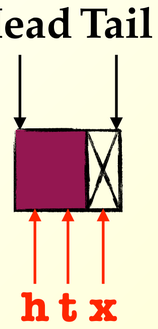


The Michael-Scott Queue Algorithm

Empty queue

```
deq(k):
Node h,t,x
Integer v
1  while (true)
2    h = Head
3    t = Tail
4    x = h.next
5    if (h == Head)
6      if (h == t)
7        if (x == NULL)
8          return *
9          exit
10       CAS (Tail,t,x)
11     else
12       v = x.key
13       if (CAS (Head,h,x))
14         return v
15         exit
```



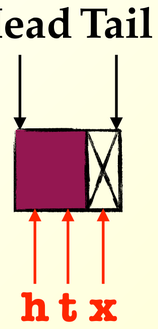


The Michael-Scott Queue Algorithm

Empty queue

```
deq(k):
Node h,t,x
Integer v
1  while (true)
2    h = Head
3    t = Tail
4    x = h.next
5    if (h == Head)
6      if (h == t)
7        if (x == NULL)
8          return *
9          exit
10       CAS (Tail,t,x)
11     else
12       v = x.key
13       if (CAS (Head,h,x))
14         return v
15         exit
```



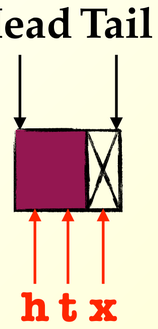


The Michael-Scott Queue Algorithm

Empty queue

```
deq(k):
Node h,t,x
Integer v
1  while (true)
2    h = Head
3    t = Tail
4    x = h.next
5    if (h == Head)
6      if (h == t)
7        if (x == NULL)
8          return *
9          exit
10       CAS (Tail,t,x)
11     else
12       v = x.key
13       if (CAS (Head,h,x))
14         return v
15         exit
```



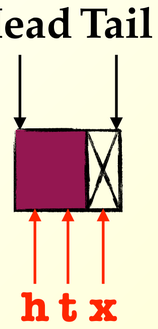


The Michael-Scott Queue Algorithm

Empty queue

```
deq(k):
Node h,t,x
Integer v
1  while (true)
2    h = Head
3    t = Tail
4    x = h.next
5    if (h == Head)
6      if (h == t)
7        if (x == NULL)
8          return *
9          exit
10       CAS (Tail,t,x)
11     else
12       v = x.key
13       if (CAS (Head,h,x))
14         return v
15         exit
```



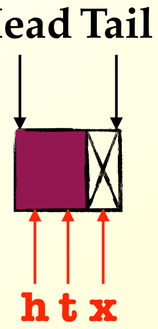


The Michael-Scott Queue Algorithm

Empty queue

```
deq(k):
Node h,t,x
Integer v
1  while (true)
2    h = Head
3    t = Tail
4    x = h.next
5    if (h == Head)
6      if (h == t)
7        if (x == NULL)
8          return *
9          exit
10       CAS (Tail,t,x)
11     else
12       v = x.key
13       if (CAS (Head,h,x))
14         return v
15         exit
```



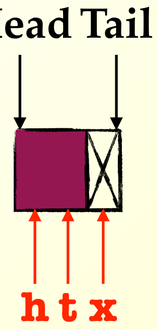


The Michael-Scott Queue Algorithm

Empty queue

```
deq(k):
Node h,t,x
Integer v
1  while (true)
2    h = Head
3    t = Tail
4    x = h.next
5    if (h == Head)
6      if (h == t)
7        if (x == NULL)
8          return *
9          exit
10       CAS (Tail,t,x)
11     else
12       v = x.key
13       if (CAS (Head,h,x))
14         return v
15         exit
```



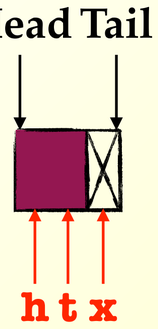


The Michael-Scott Queue Algorithm

Empty queue

```
deq(k):
Node h,t,x
Integer v
1  while (true)
2    h = Head
3    t = Tail
4    x = h.next
5    if (h == Head)
6      if (h == t)
7        if (x == NULL)
8          return *
9          exit
10       CAS (Tail,t,x)
11     else
12       v = x.key
13       if (CAS (Head,h,x))
14         return v
15         exit
```





The Michael-Scott Queue Algorithm

Linearization Policy

```
 enq(k):
 Node t,x
 1  n = new Node(k,NULL)
 2  while (true)
 3    t = Tail
 4    x = t.next
 5    if (t == Tail)
 6      if (x == NULL)
 7        if (CAS (t.next,NULL,n))
 8          CAS (Tail,t,n)
 9          exit
10      else
11        CAS (Tail,t,x)
```

```
 deq(k):
 Node h,t,x
 Integer v
 1  while (true)
 2    h = Head
 3    t = Tail
 4    x = h.next
 5    if (h == Head)
 6      if (h == t)
 7        if (x == NULL)
 8          return *
 9          exit
10        CAS (Tail,t,x)
11      else
12        v = x.key
13        if (CAS (Head,h,x))
14          return v
15          exit
```

The Michael-Scott Queue Algorithm

Linearization Policy

```
enq(k):
Node t,x
 1  n = new Node(k,NULL)
 2  while (true)
 3    t = Tail
 4    x = t.next
 5    if (t == Tail)
 6      if (x == NULL)
 7        if (CAS (t.next,NULL,n))
 8          CAS (Tail,t,n)
 9          exit
10      else
11        CAS (Tail,t,x)
```



```
deq(k):
Node h,t,x
Integer v
 1  while (true)
 2    h = Head
 3    t = Tail
 4    x = h.next
 5    if (h == Head)
 6      if (h == t)
 7        if (x == NULL)
 8          return *
 9          exit
10        CAS (Tail,t,x)
11      else
12        v = x.key
13        if (CAS (Head,h,x))
14          return v
15          exit
```

The Michael-Scott Queue Algorithm

Linearization Policy

```
enq(k):
Node t,x
 1  n = new Node(k,NULL)
 2  while (true)
 3    t = Tail
 4    x = t.next
 5    if (t == Tail)
 6      if (x == NULL)
 7        if (CAS (t.next,NULL,n))
 8          CAS (Tail,t,n)
 9          exit
10      else
11        CAS (Tail,t,x)
```

when successful

```
deq(k):
Node h,t,x
Integer v
 1  while (true)
 2    h = Head
 3    t = Tail
 4    x = h.next
 5    if (h == Head)
 6      if (h == t)
 7        if (x == NULL)
 8          return *
 9          exit
10        CAS (Tail,t,x)
11      else
12        v = x.key
13        if (CAS (Head,h,x))
14          return v
15          exit
```

The Michael-Scott Queue Algorithm

Linearization Policy

```
enq(k):
Node t,x
 1  n = new Node(k,NULL)
 2  while (true)
 3    t = Tail
 4    x = t.next
 5    if (t == Tail)
 6      if (x == NULL)
 7        if (CAS (t.next,NULL,n))
 8          CAS (Tail,t,n)
 9          exit
10      else
11        CAS (Tail,t,x)
```

when successful

```
deq(k):
Node h,t,x
Integer v
 1  while (true)
 2    h = Head
 3    t = Tail
 4    x = h.next
 5    if (h == Head)
 6      if (h == t)
 7        if (x == NULL)
 8          return *
 9          exit
10        CAS (Tail,t,x)
11      else
12        v = x.key
13        if (CAS (Head,h,x))
14          return v
15          exit
```

The Michael-Scott Queue Algorithm

Linearization Policy

```
enq(k):
Node t,x
 1  n = new Node(k,NULL)
 2  while (true)
 3    t = Tail
 4    x = t.next
 5    if (t == Tail)
 6      if (x == NULL)
 7        if (CAS (t.next,NULL,n))
 8          CAS (Tail,t,n)
 9          exit
10      else
11        CAS (Tail,t,x)
```

when successful

```
deq(k):
Node h,t,x
Integer v
 1  while (true)
 2    h = Head
 3    t = Tail
 4    x = h.next
 5    if (h == Head)
 6      if (h == t)
 7        if (x == NULL)
 8          return *
 9          exit
10        CAS (Tail,t,x)
11      else
12        v = x.key
13        if (CAS (Head,h,x))
14          return v
15          exit
```

when successful

The Michael-Scott Queue Algorithm

Linearization Policy

```
enq(k):
Node t,x
 1  n = new Node(k,NULL)
 2  while (true)
 3    t = Tail
 4    x = t.next
 5    if (t == Tail)
 6      if (x == NULL)
 7        if (CAS (t.next,NULL,n))
 8          CAS (Tail,t,n)
 9          exit
10      else
11        CAS (Tail,t,x)
```

when successful

```
deq(k):
Node h,t,x
Integer v
 1  while (true)
 2    h = Head
 3    t = Tail
 4    x = h.next
 5    if (h == Head)
 6      if (h == t)
 7        if (x == NULL)
 8          return *
 9          exit
10        CAS (Tail,t,x)
11      else
12        v = x.key
13        if (CAS (Head,h,x))
14          return v
15          exit
```

when successful

The Michael-Scott Queue Algorithm

Linearization Policy

```
enq(k):
Node t,x
 1  n = new Node(k,NULL)
 2  while (true)
 3    t = Tail
 4    x = t.next
 5    if (t == Tail)
 6      if (x == NULL)
 7        if (CAS (t.next,NULL,n))
 8          CAS (Tail,t,n)
 9          exit
10      else
11        CAS (Tail,t,x)
```

when successful

```
deq(k):
Node h,t,x
Integer v
 1  while (true)
 2    h = Head
 3    t = Tail
 4    x = h.next
 5    if (h == Head)
 6      if (h == t)
 7        if (x == NULL)
 8          return *
 9          exit
10        CAS (Tail,t,x)
11      else
12        v = x.key
13        if (CAS (Head,h,x))
14          return v
15          exit
```

when empty

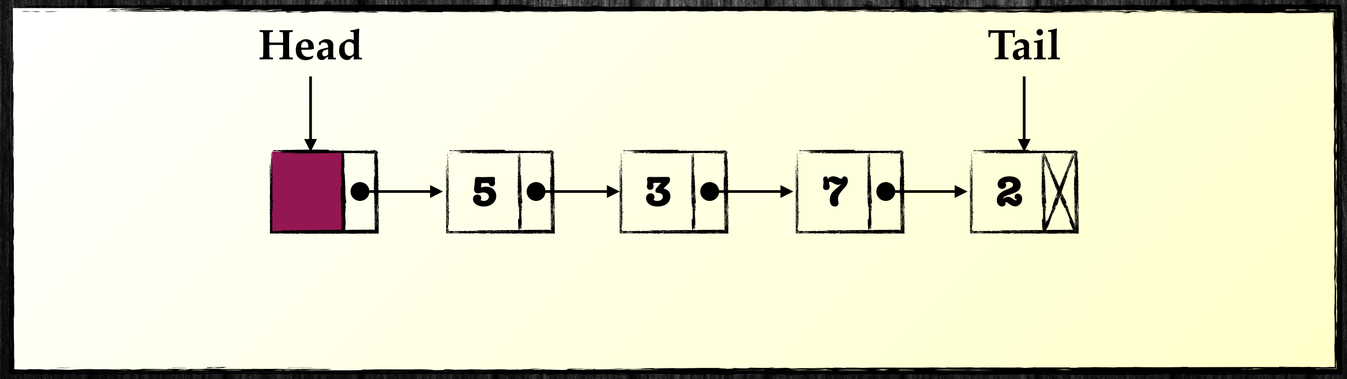
when successful

The Michael-Scott Queue Algorithm

Linearization Policy

```
enq(k):
Node t,x
1  n = new Node(k,NULL)
2  while (true)
3    t = Tail
4    x = t.next
5    if (t == Tail)
6      if (x == NULL)
7        if (CAS (t.next,NULL,n))
8          CAS (Tail,t,n)
9          exit
10     else
11       CAS (Tail,t,x)
```

```
deq(k):
Node h,t,x
Integer v
1  while (true)
2    h = Head
3    t = Tail
4    x = h.next
5    if (h == Head)
6      if (h == t)
7        if (x == NULL)
8          return *
9          exit
10       CAS (Tail,t,x)
11     else
12       v = x.key
13       if (CAS (Head,h,x))
14         return v
15         exit
```

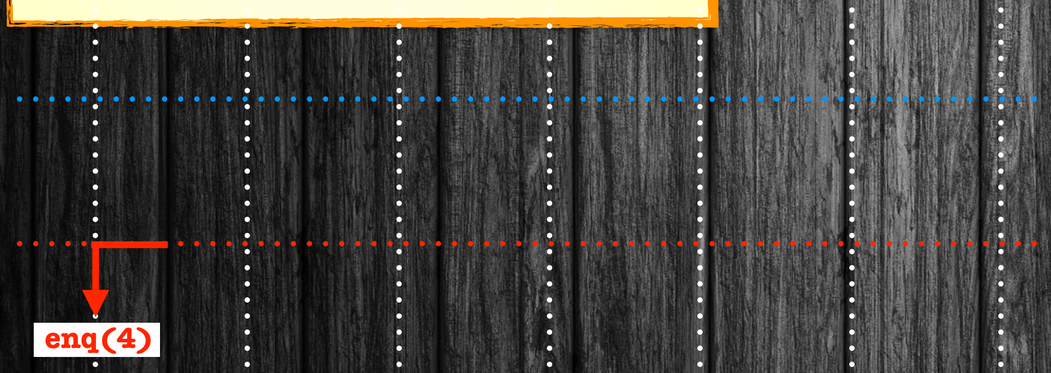


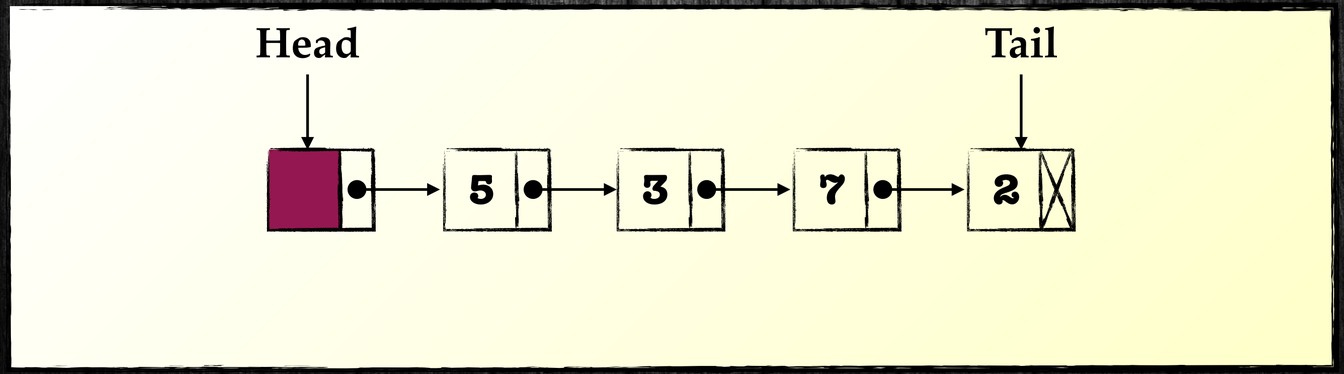
The Michael-Scott Queue Algorithm

Linearization Policy

```
enq(k):
Node t,x
1  n = new Node(k,NULL)
2  while (true)
3    t = Tail
4    x = t.next
5    if (t == Tail)
6      if (x == NULL)
7        if (CAS (t.next,NULL,n))
8          CAS (Tail,t,n)
9          exit
10     else
11       CAS (Tail,t,x)
```

```
deq(k):
Node h,t,x
Integer v
1  while (true)
2    h = Head
3    t = Tail
4    x = h.next
5    if (h == Head)
6      if (h == t)
7        if (x == NULL)
8          return *
9          exit
10       CAS (Tail,t,x)
11     else
12       v = x.key
13       if (CAS (Head,h,x))
14         return v
15         exit
```





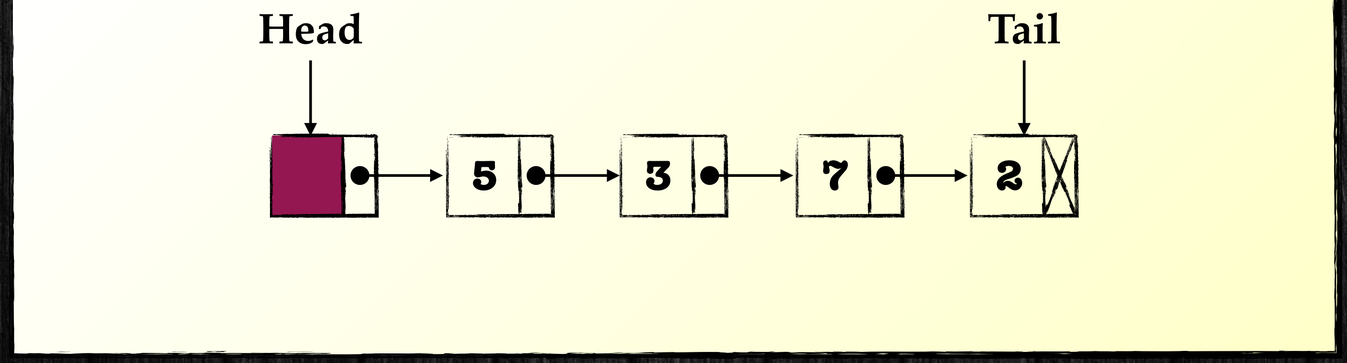
The Michael-Scott Queue Algorithm

Linearization Policy

```
enq(k):
Node t,x
1  n = new Node(k,NULL)
2  while (true)
3    t = Tail
4    x = t.next
5    if (t == Tail)
6      if (x == NULL)
7        if (CAS (t.next,NULL,n))
8          CAS (Tail,t,n)
9          exit
10       else
11         CAS (Tail,t,x)
```

```
deq(k):
Node h,t,x
Integer v
1  while (true)
2    h = Head
3    t = Tail
4    x = h.next
5    if (h == Head)
6      if (h == t)
7        if (x == NULL)
8          return *
9          exit
10       CAS (Tail,t,x)
11     else
12       v = x.key
13       if (CAS (Head,h,x))
14         return v
15         exit
```

enq(4)



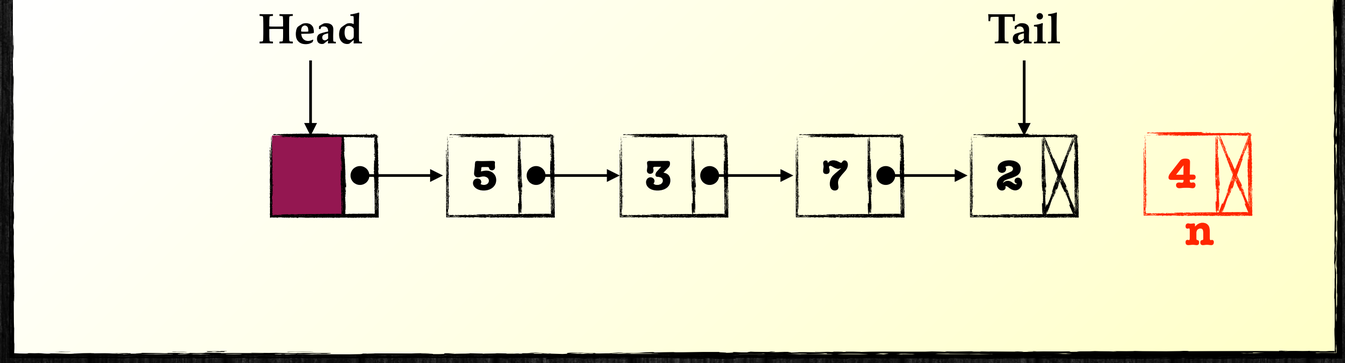
The Michael-Scott Queue Algorithm

Linearization Policy

```
enq(k):
Node t,x
1  n = new Node(k,NULL)
2  while (true)
3    t = Tail
4    x = t.next
5    if (t == Tail)
6      if (x == NULL)
7        if (CAS (t.next,NULL,n))
8          CAS (Tail,t,n)
9          exit
10     else
11       CAS (Tail,t,x)
```

```
deq(k):
Node h,t,x
Integer v
1  while (true)
2    h = Head
3    t = Tail
4    x = h.next
5    if (h == Head)
6      if (h == t)
7        if (x == NULL)
8          return *
9          exit
10       CAS (Tail,t,x)
11     else
12       v = x.key
13       if (CAS (Head,h,x))
14         return v
15         exit
```

enq(4)



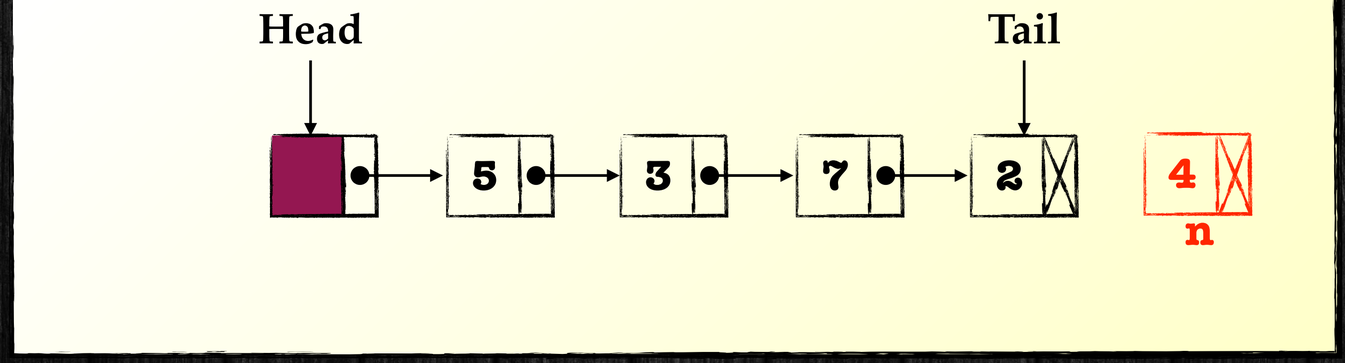
The Michael-Scott Queue Algorithm

Linearization Policy

```
enq(k):
Node t,x
1  n = new Node(k,NULL)
2  while (true)
3    t = Tail
4    x = t.next
5    if (t == Tail)
6      if (x == NULL)
7        if (CAS (t.next,NULL,n))
8          CAS (Tail,t,n)
9          exit
10     else
11       CAS (Tail,t,x)
```

```
deq(k):
Node h,t,x
Integer v
1  while (true)
2    h = Head
3    t = Tail
4    x = h.next
5    if (h == Head)
6      if (h == t)
7        if (x == NULL)
8          return *
9          exit
10       CAS (Tail,t,x)
11     else
12       v = x.key
13       if (CAS (Head,h,x))
14         return v
15         exit
```

enq(4)



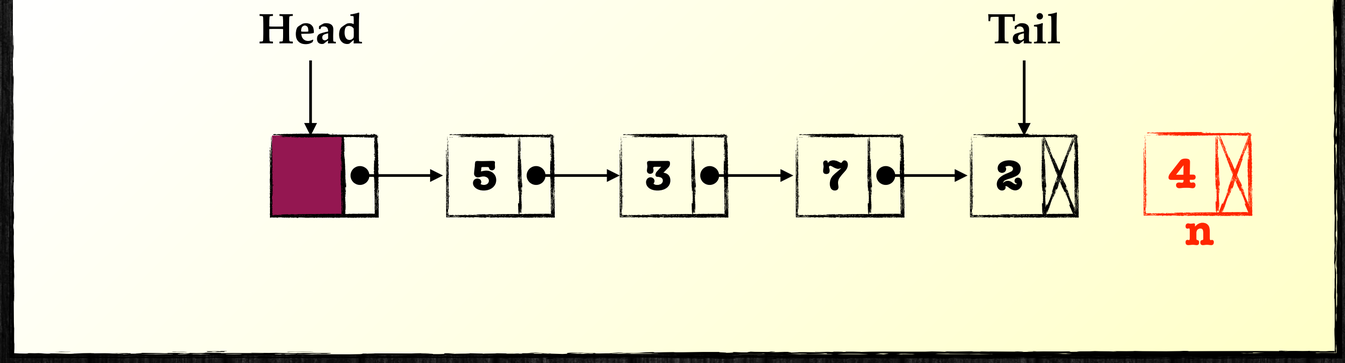
The Michael-Scott Queue Algorithm

Linearization Policy

```
enq(k):
Node t,x
1  n = new Node(k,NULL)
2  while (true)
3    t = Tail
4    x = t.next
5    if (t == Tail)
6      if (x == NULL)
7        if (CAS (t.next,NULL,n))
8          CAS (Tail,t,n)
9          exit
10     else
11       CAS (Tail,t,x)
```

```
deq(k):
Node h,t,x
Integer v
1  while (true)
2    h = Head
3    t = Tail
4    x = h.next
5    if (h == Head)
6      if (h == t)
7        if (x == NULL)
8          return *
9          exit
10       CAS (Tail,t,x)
11     else
12       v = x.key
13       if (CAS (Head,h,x))
14         return v
15         exit
```

enq(4)



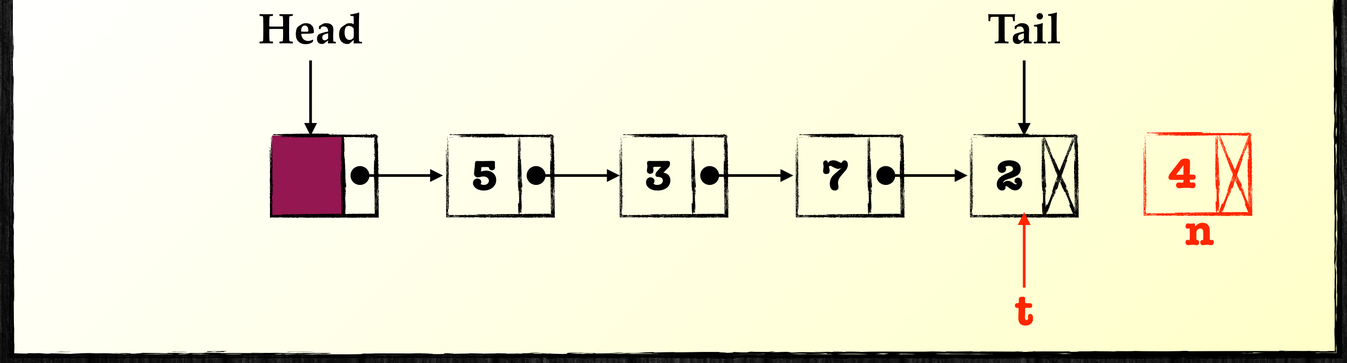
The Michael-Scott Queue Algorithm

Linearization Policy

```
enq(k):
Node t,x
1  n = new Node(k,NULL)
2  while (true)
3    t = Tail
4    x = t.next
5    if (t == Tail)
6      if (x == NULL)
7        if (CAS (t.next,NULL,n))
8          CAS (Tail,t,n)
9          exit
10     else
11       CAS (Tail,t,x)
```

```
deq(k):
Node h,t,x
Integer v
1  while (true)
2    h = Head
3    t = Tail
4    x = h.next
5    if (h == Head)
6      if (h == t)
7        if (x == NULL)
8          return *
9          exit
10       CAS (Tail,t,x)
11     else
12       v = x.key
13       if (CAS (Head,h,x))
14         return v
15         exit
```

enq(4)



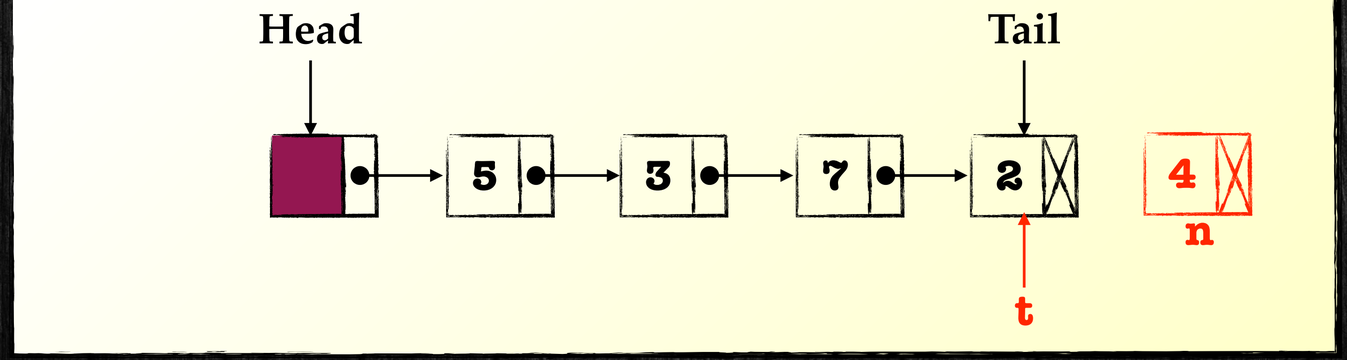
The Michael-Scott Queue Algorithm

Linearization Policy

```
enq(k):
Node t,x
1  n = new Node(k,NULL)
2  while (true)
3    t = Tail
4    x = t.next
5    if (t == Tail)
6      if (x == NULL)
7        if (CAS (t.next,NULL,n))
8          CAS (Tail,t,n)
9          exit
10     else
11       CAS (Tail,t,x)
```

```
deq(k):
Node h,t,x
Integer v
1  while (true)
2    h = Head
3    t = Tail
4    x = h.next
5    if (h == Head)
6      if (h == t)
7        if (x == NULL)
8          return *
9          exit
10       CAS (Tail,t,x)
11     else
12       v = x.key
13       if (CAS (Head,h,x))
14         return v
15         exit
```

enq(4)



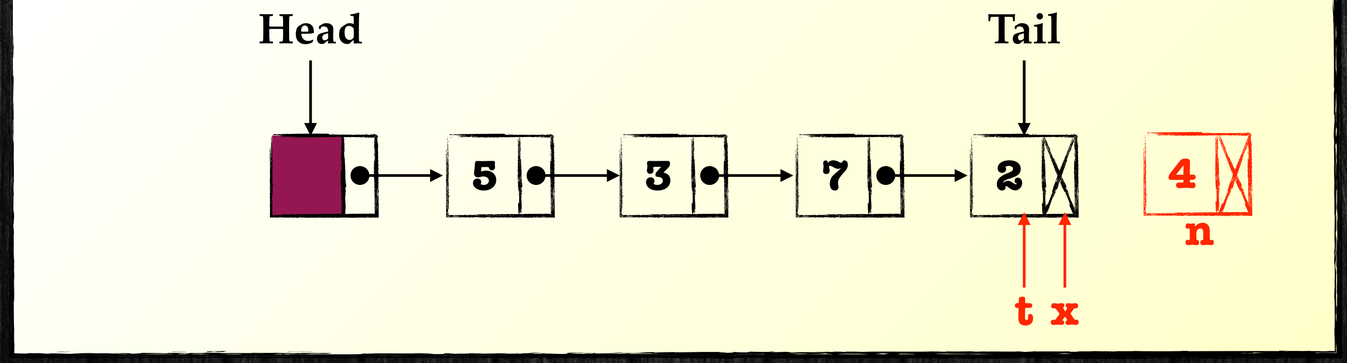
The Michael-Scott Queue Algorithm

Linearization Policy

```
enq(k):
Node t,x
1  n = new Node(k,NULL)
2  while (true)
3    t = Tail
4    x = t.next
5    if (t == Tail)
6      if (x == NULL)
7        if (CAS (t.next,NULL,n))
8          CAS (Tail,t,n)
9          exit
10     else
11       CAS (Tail,t,x)
```

```
deq(k):
Node h,t,x
Integer v
1  while (true)
2    h = Head
3    t = Tail
4    x = h.next
5    if (h == Head)
6      if (h == t)
7        if (x == NULL)
8          return *
9          exit
10       CAS (Tail,t,x)
11     else
12       v = x.key
13       if (CAS (Head,h,x))
14         return v
15         exit
```

enq(4)



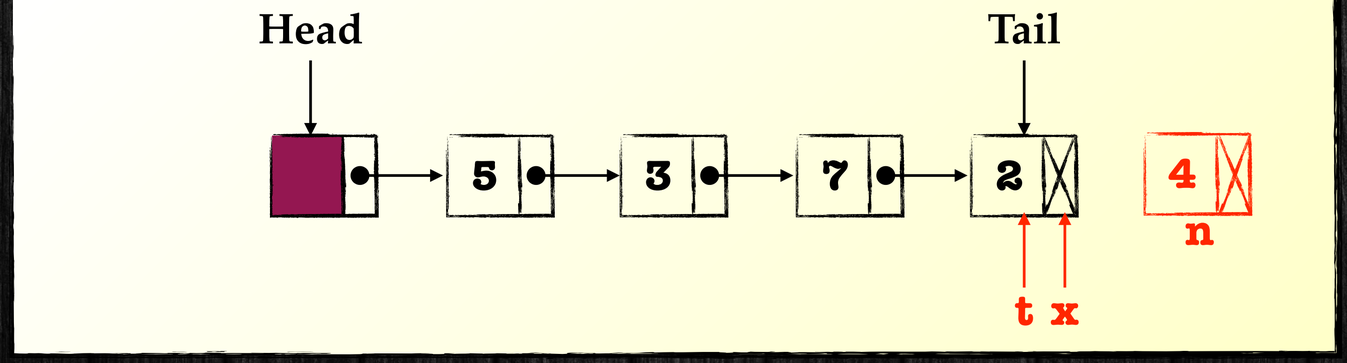
The Michael-Scott Queue Algorithm

Linearization Policy

```
enq(k):
Node t,x
1  n = new Node(k,NULL)
2  while (true)
3    t = Tail
4    x = t.next
5    if (t == Tail)
6      if (x == NULL)
7        if (CAS (t.next,NULL,n))
8          CAS (Tail,t,n)
9          exit
10     else
11       CAS (Tail,t,x)
```

```
deq(k):
Node h,t,x
Integer v
1  while (true)
2    h = Head
3    t = Tail
4    x = h.next
5    if (h == Head)
6      if (h == t)
7        if (x == NULL)
8          return *
9          exit
10       CAS (Tail,t,x)
11     else
12       v = x.key
13       if (CAS (Head,h,x))
14         return v
15         exit
```

enq(4)



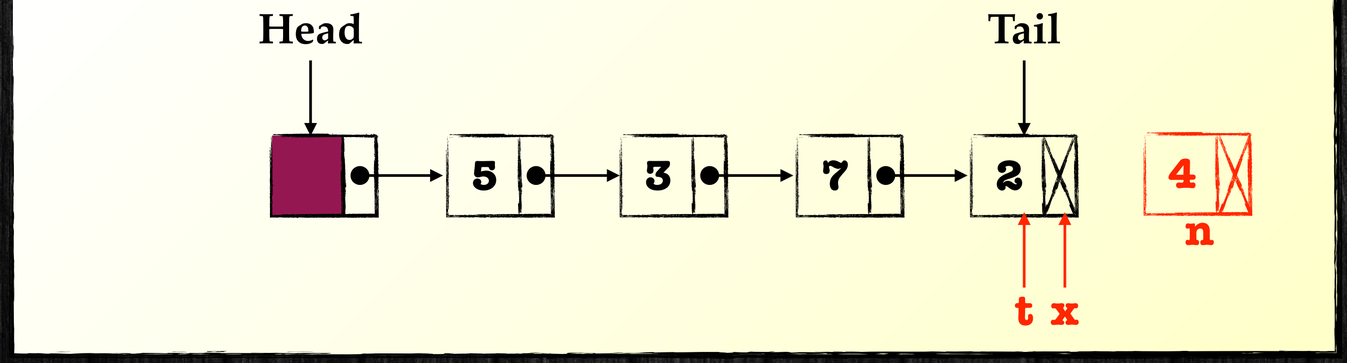
The Michael-Scott Queue Algorithm

Linearization Policy

```
enq(k):
Node t,x
1  n = new Node(k,NULL)
2  while (true)
3    t = Tail
4    x = t.next
5    if (t == Tail)
6      if (x == NULL)
7        if (CAS (t.next,NULL,n))
8          CAS (Tail,t,n)
9          exit
10       else
11         CAS (Tail,t,x)
```

```
deq(k):
Node h,t,x
Integer v
1  while (true)
2    h = Head
3    t = Tail
4    x = h.next
5    if (h == Head)
6      if (h == t)
7        if (x == NULL)
8          return *
9          exit
10       CAS (Tail,t,x)
11     else
12       v = x.key
13       if (CAS (Head,h,x))
14         return v
15         exit
```

enq(4)



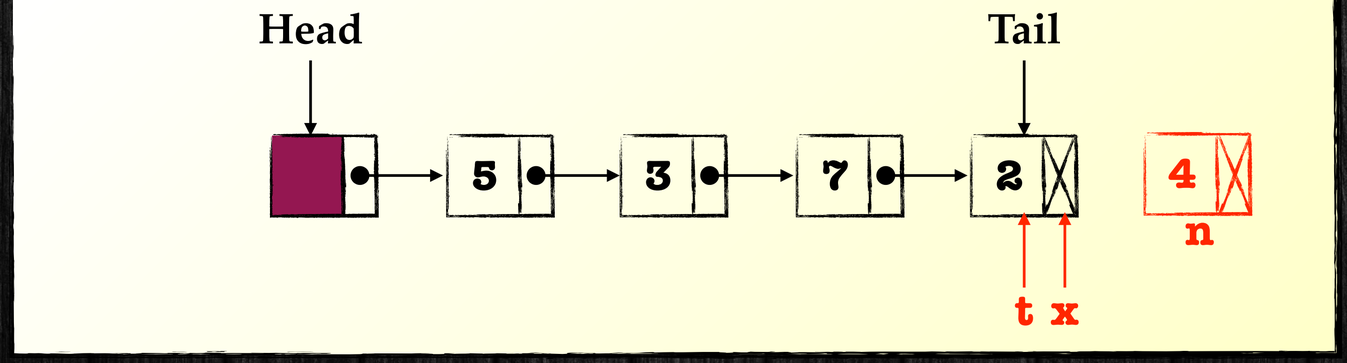
The Michael-Scott Queue Algorithm

Linearization Policy

```
enq(k):
Node t,x
1  n = new Node(k,NULL)
2  while (true)
3    t = Tail
4    x = t.next
5    if (t == Tail)
6      if (x == NULL)
7        if (CAS (t.next,NULL,n))
8          CAS (Tail,t,n)
9          exit
10     else
11       CAS (Tail,t,x)
```

```
deq(k):
Node h,t,x
Integer v
1  while (true)
2    h = Head
3    t = Tail
4    x = h.next
5    if (h == Head)
6      if (h == t)
7        if (x == NULL)
8          return *
9          exit
10       CAS (Tail,t,x)
11     else
12       v = x.key
13       if (CAS (Head,h,x))
14         return v
15         exit
```

enq(4)



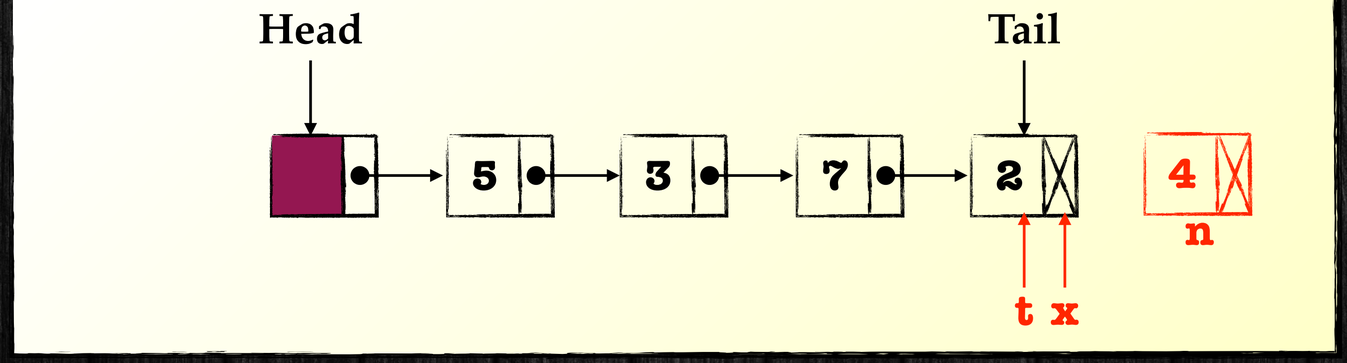
The Michael-Scott Queue Algorithm

Linearization Policy

```
enq(k):
Node t,x
1  n = new Node(k,NULL)
2  while (true)
3    t = Tail
4    x = t.next
5    if (t == Tail)
6      if (x == NULL)
7        if (CAS (t.next,NULL,n))
8          CAS (Tail,t,n)
9          exit
10     else
11       CAS (Tail,t,x)
```

```
deq(k):
Node h,t,x
Integer v
1  while (true)
2    h = Head
3    t = Tail
4    x = h.next
5    if (h == Head)
6      if (h == t)
7        if (x == NULL)
8          return *
9          exit
10       CAS (Tail,t,x)
11     else
12       v = x.key
13       if (CAS (Head,h,x))
14         return v
15         exit
```

enq(4)

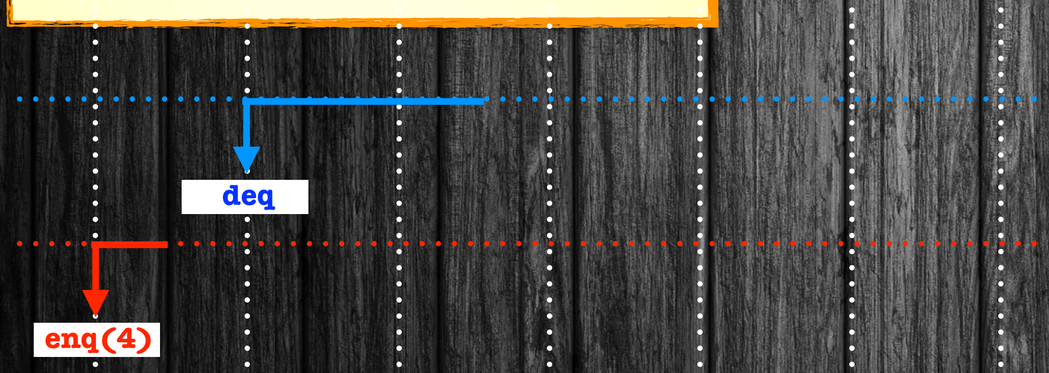


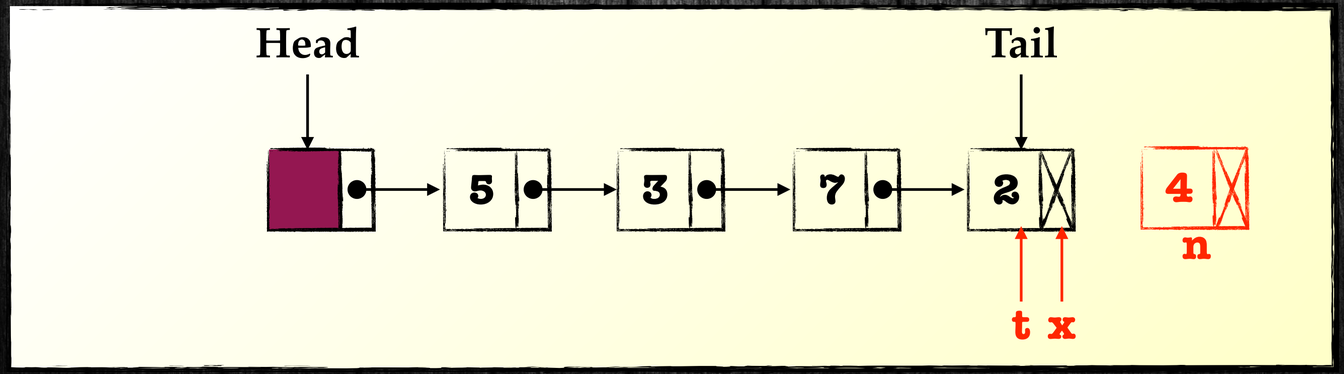
The Michael-Scott Queue Algorithm

Linearization Policy

```
enq(k):
Node t,x
1  n = new Node(k,NULL)
2  while (true)
3    t = Tail
4    x = t.next
5    if (t == Tail)
6      if (x == NULL)
7        if (CAS (t.next,NULL,n))
8          CAS (Tail,t,n)
9          exit
10     else
11       CAS (Tail,t,x)
```

```
deq(k):
Node h,t,x
Integer v
1  while (true)
2    h = Head
3    t = Tail
4    x = h.next
5    if (h == Head)
6      if (h == t)
7        if (x == NULL)
8          return *
9          exit
10       CAS (Tail,t,x)
11     else
12       v = x.key
13       if (CAS (Head,h,x))
14         return v
15         exit
```



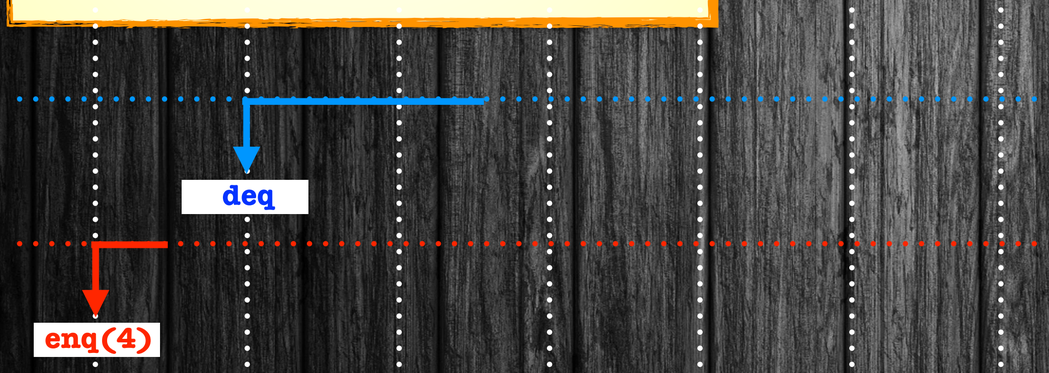


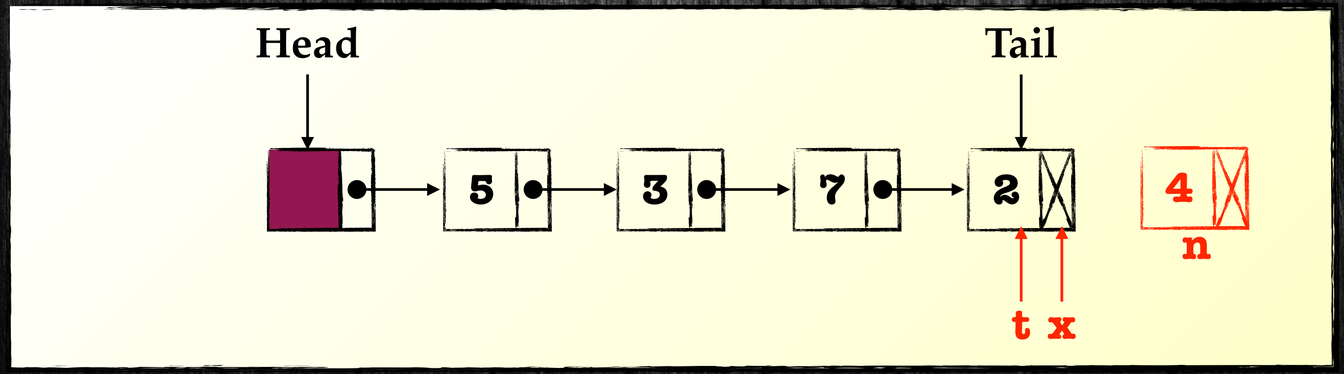
The Michael-Scott Queue Algorithm

Linearization Policy

```
enq(k):
Node t,x
1  n = new Node(k,NULL)
2  while (true)
3    t = Tail
4    x = t.next
5    if (t == Tail)
6      if (x == NULL)
7        if (CAS (t.next,NULL,n))
8          CAS (Tail,t,n)
9          exit
10     else
11       CAS (Tail,t,x)
```

```
deq(k):
Node h,t,x
Integer v
1  while (true)
2    h = Head
3    t = Tail
4    x = h.next
5    if (h == Head)
6      if (h == t)
7        if (x == NULL)
8          return *
9          exit
10       CAS (Tail,t,x)
11     else
12       v = x.key
13       if (CAS (Head,h,x))
14         return v
15         exit
```



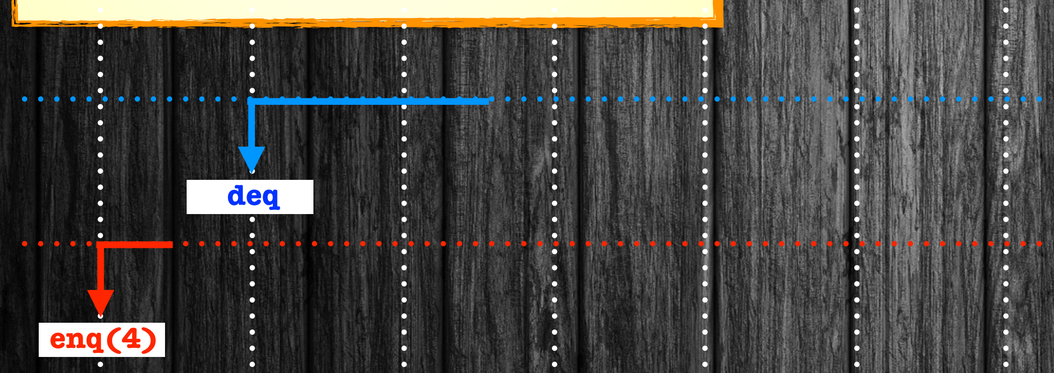


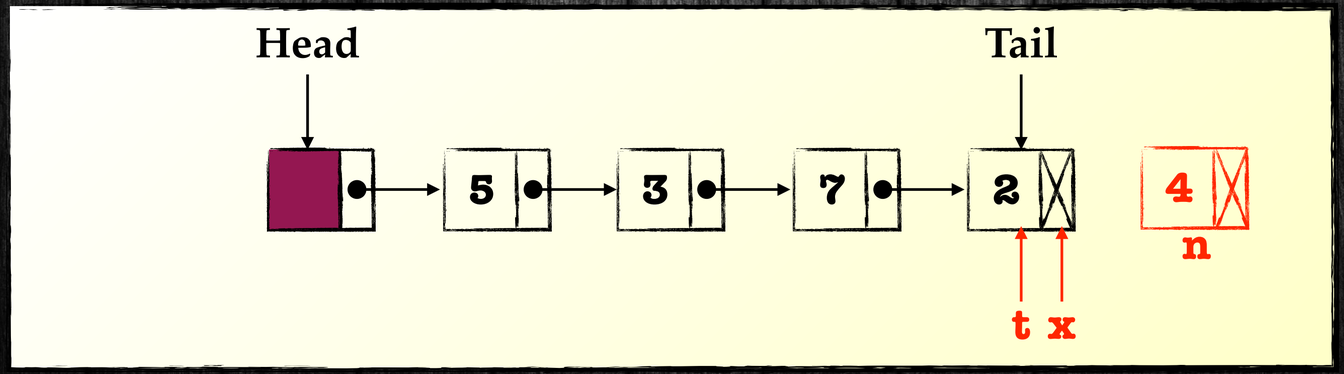
The Michael-Scott Queue Algorithm

Linearization Policy

```
enq(k):
Node t,x
1  n = new Node(k,NULL)
2  while (true)
3    t = Tail
4    x = t.next
5    if (t == Tail)
6      if (x == NULL)
7        if (CAS (t.next,NULL,n))
8          CAS (Tail,t,n)
9          exit
10     else
11       CAS (Tail,t,x)
```

```
deq(k):
Node h,t,x
Integer v
1  while (true)
2    h = Head
3    t = Tail
4    x = h.next
5    if (h == Head)
6      if (h == t)
7        if (x == NULL)
8          return *
9          exit
10       CAS (Tail,t,x)
11     else
12       v = x.key
13       if (CAS (Head,h,x))
14         return v
15         exit
```



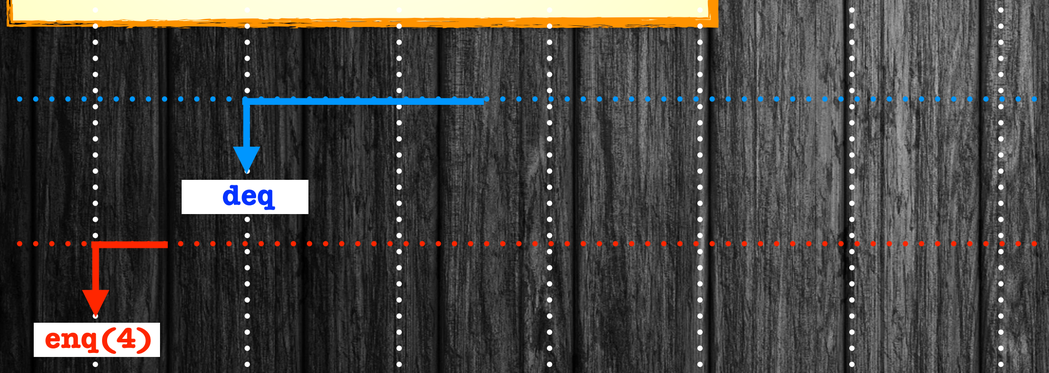


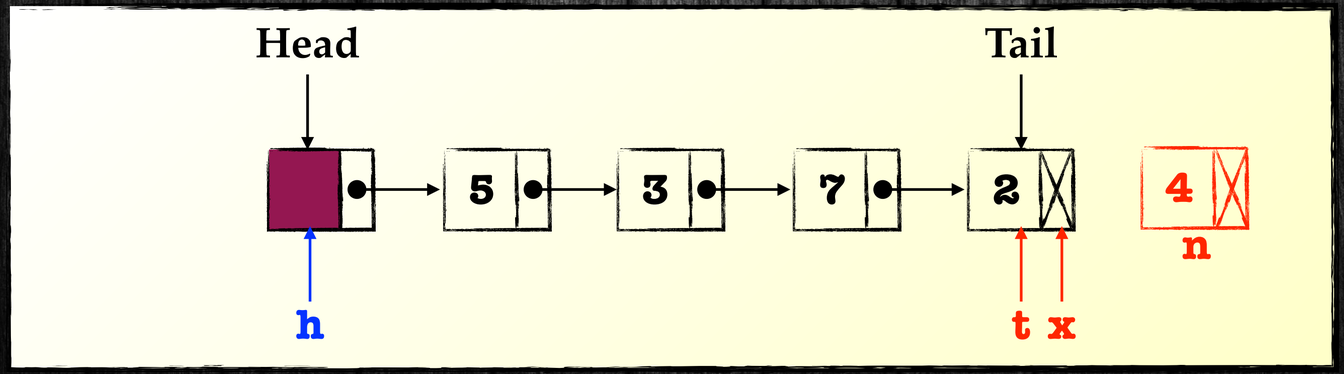
The Michael-Scott Queue Algorithm

Linearization Policy

```
enq(k):
Node t,x
1  n = new Node(k,NULL)
2  while (true)
3    t = Tail
4    x = t.next
5    if (t == Tail)
6      if (x == NULL)
7        if (CAS (t.next,NULL,n))
8          CAS (Tail,t,n)
9          exit
10     else
11       CAS (Tail,t,x)
```

```
deq(k):
Node h,t,x
Integer v
1  while (true)
2    h = Head
3    t = Tail
4    x = h.next
5    if (h == Head)
6      if (h == t)
7        if (x == NULL)
8          return *
9          exit
10       CAS (Tail,t,x)
11     else
12       v = x.key
13       if (CAS (Head,h,x))
14         return v
15         exit
```



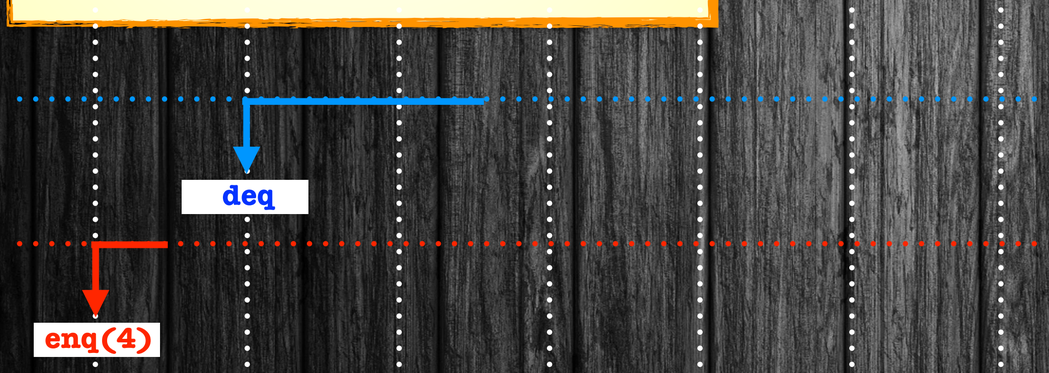


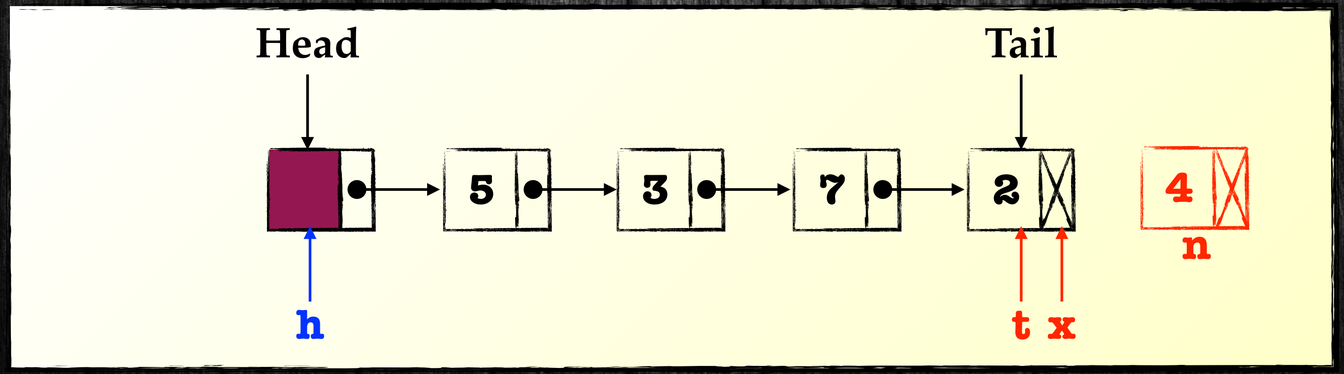
The Michael-Scott Queue Algorithm

Linearization Policy

```
enq(k):
Node t,x
1  n = new Node(k,NULL)
2  while (true)
3    t = Tail
4    x = t.next
5    if (t == Tail)
6      if (x == NULL)
7        if (CAS (t.next,NULL,n))
8          CAS (Tail,t,n)
9          exit
10     else
11       CAS (Tail,t,x)
```

```
deq(k):
Node h,t,x
Integer v
1  while (true)
2    h = Head
3    t = Tail
4    x = h.next
5    if (h == Head)
6      if (h == t)
7        if (x == NULL)
8          return *
9          exit
10       CAS (Tail,t,x)
11     else
12       v = x.key
13       if (CAS (Head,h,x))
14         return v
15         exit
```





The Michael-Scott Queue Algorithm

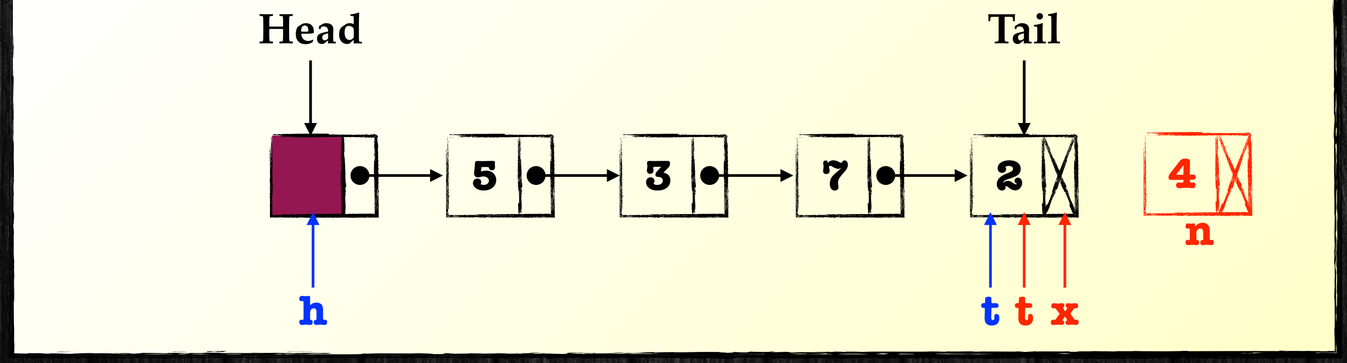
Linearization Policy

```
enq(k):
Node t,x
1  n = new Node(k,NULL)
2  while (true)
3    t = Tail
4    x = t.next
5    if (t == Tail)
6      if (x == NULL)
7        if (CAS (t.next,NULL,n))
8          CAS (Tail,t,n)
9          exit
10     else
11       CAS (Tail,t,x)
```

```
deq(k):
Node h,t,x
Integer v
1  while (true)
2    h = Head
3    t = Tail
4    x = h.next
5    if (h == Head)
6      if (h == t)
7        if (x == NULL)
8          return *
9          exit
10       CAS (Tail,t,x)
11     else
12       v = x.key
13       if (CAS (Head,h,x))
14         return v
15         exit
```

deq

enq(4)



The Michael-Scott Queue Algorithm

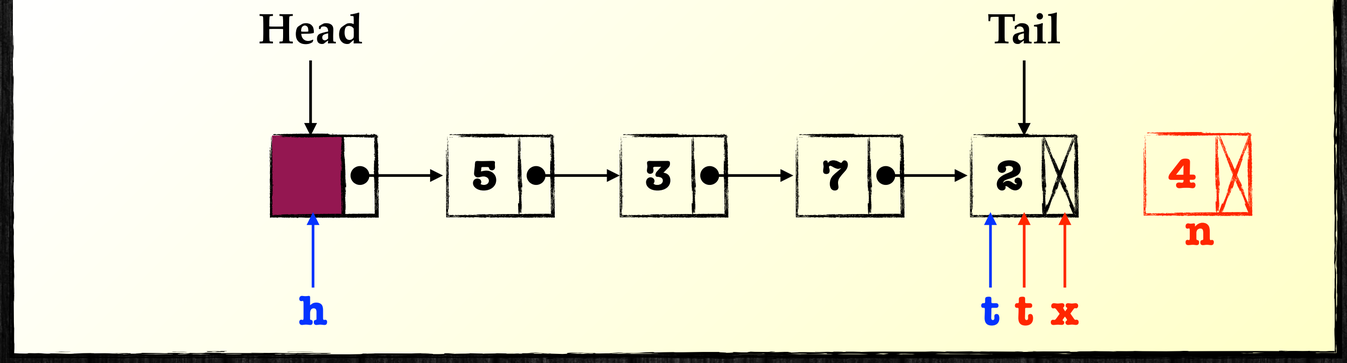
Linearization Policy

```
enq(k):
Node t,x
1  n = new Node(k,NULL)
2  while (true)
3    t = Tail
4    x = t.next
5    if (t == Tail)
6      if (x == NULL)
7        if (CAS (t.next,NULL,n))
8          CAS (Tail,t,n)
9          exit
10     else
11       CAS (Tail,t,x)
```

```
deq(k):
Node h,t,x
Integer v
1  while (true)
2    h = Head
3    t = Tail
4    x = h.next
5    if (h == Head)
6      if (h == t)
7        if (x == NULL)
8          return *
9          exit
10       CAS (Tail,t,x)
11     else
12       v = x.key
13       if (CAS (Head,h,x))
14         return v
15         exit
```

deq

enq(4)

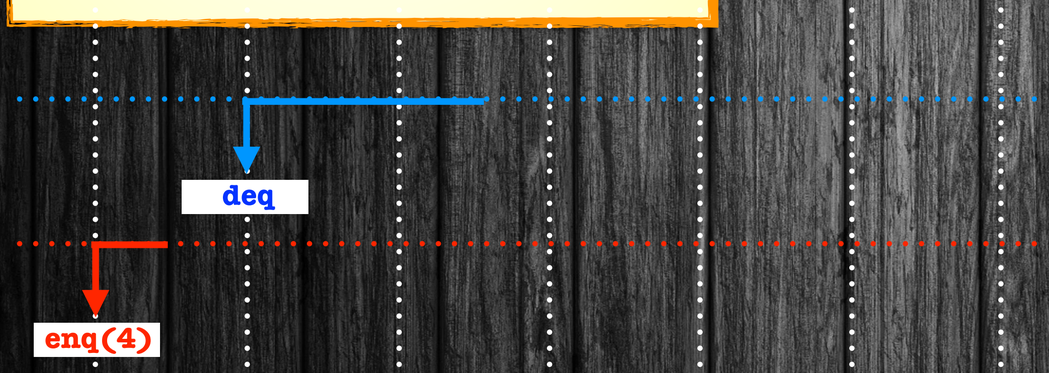


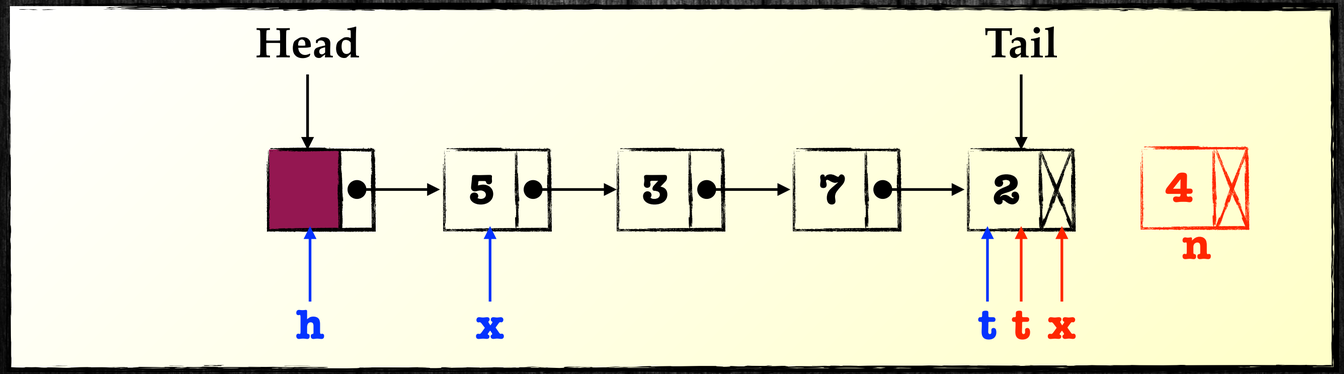
The Michael-Scott Queue Algorithm

Linearization Policy

```
enq(k):
Node t,x
1  n = new Node(k,NULL)
2  while (true)
3    t = Tail
4    x = t.next
5    if (t == Tail)
6      if (x == NULL)
7        if (CAS (t.next,NULL,n))
8          CAS (Tail,t,n)
9          exit
10     else
11       CAS (Tail,t,x)
```

```
deq(k):
Node h,t,x
Integer v
1  while (true)
2    h = Head
3    t = Tail
4    x = h.next
5    if (h == Head)
6      if (h == t)
7        if (x == NULL)
8          return *
9          exit
10       CAS (Tail,t,x)
11     else
12       v = x.key
13       if (CAS (Head,h,x))
14         return v
15         exit
```



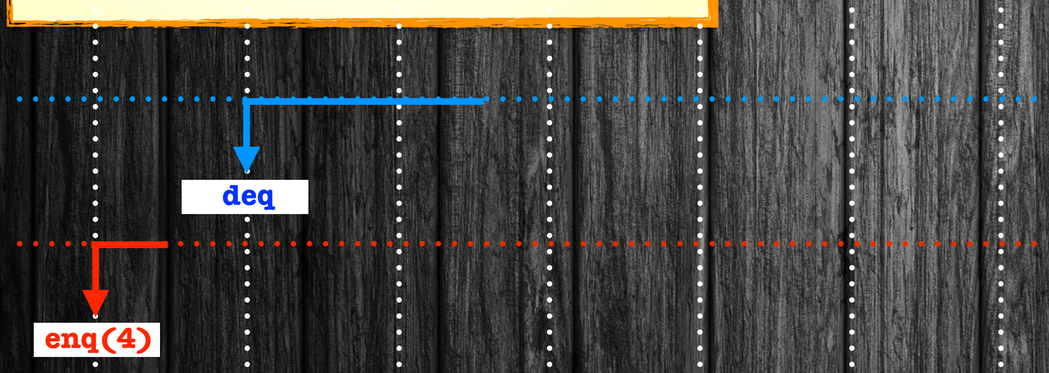


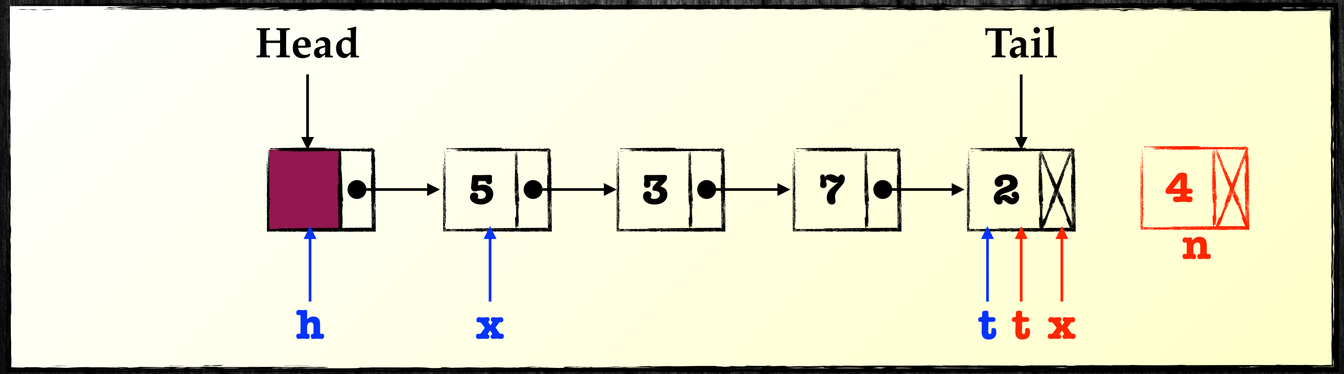
The Michael-Scott Queue Algorithm

Linearization Policy

```
enq(k):
Node t,x
1  n = new Node(k,NULL)
2  while (true)
3    t = Tail
4    x = t.next
5    if (t == Tail)
6      if (x == NULL)
7        if (CAS (t.next,NULL,n))
8          CAS (Tail,t,n)
9          exit
10     else
11       CAS (Tail,t,x)
```

```
deq(k):
Node h,t,x
Integer v
1  while (true)
2    h = Head
3    t = Tail
4    x = h.next
5    if (h == Head)
6      if (h == t)
7        if (x == NULL)
8          return *
9          exit
10       CAS (Tail,t,x)
11     else
12       v = x.key
13       if (CAS (Head,h,x))
14         return v
15         exit
```



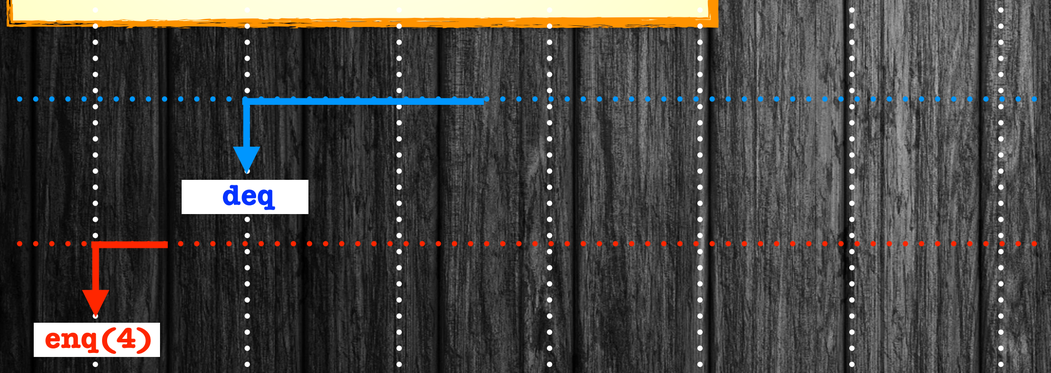


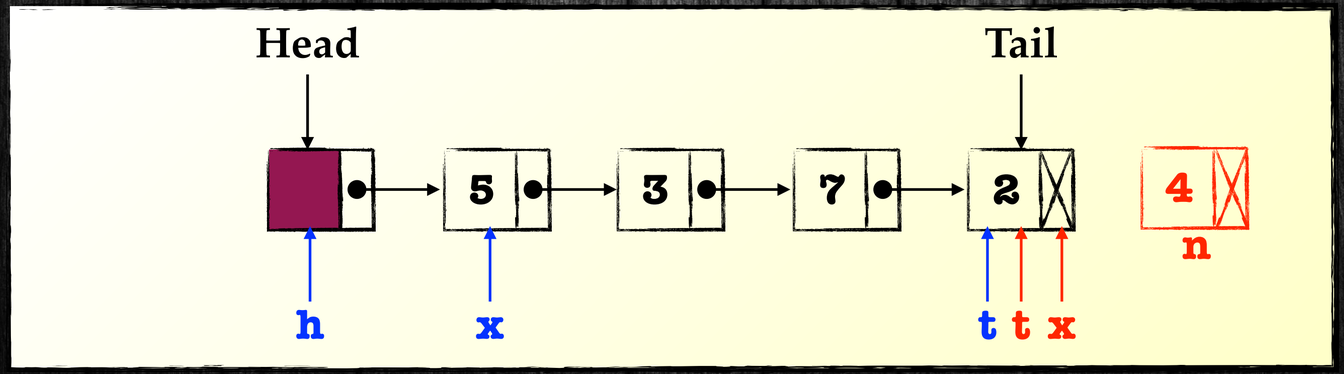
The Michael-Scott Queue Algorithm

Linearization Policy

```
enq(k):
Node t,x
1  n = new Node(k,NULL)
2  while (true)
3    t = Tail
4    x = t.next
5    if (t == Tail)
6      if (x == NULL)
7        if (CAS (t.next,NULL,n))
8          CAS (Tail,t,n)
9          exit
10     else
11       CAS (Tail,t,x)
```

```
deq(k):
Node h,t,x
Integer v
1  while (true)
2    h = Head
3    t = Tail
4    x = h.next
5    if (h == Head)
6      if (h == t)
7        if (x == NULL)
8          return *
9          exit
10       CAS (Tail,t,x)
11     else
12       v = x.key
13       if (CAS (Head,h,x))
14         return v
15         exit
```





The Michael-Scott Queue Algorithm

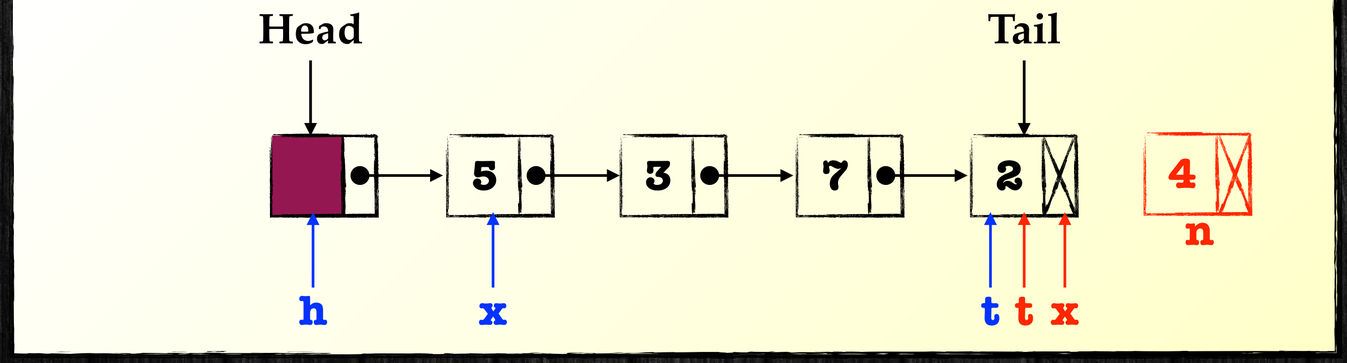
Linearization Policy

```
enq(k):
Node t,x
1  n = new Node(k,NULL)
2  while (true)
3    t = Tail
4    x = t.next
5    if (t == Tail)
6      if (x == NULL)
7        if (CAS (t.next,NULL,n))
8          CAS (Tail,t,n)
9          exit
10     else
11       CAS (Tail,t,x)
```

```
deq(k):
Node h,t,x
Integer v
1  while (true)
2    h = Head
3    t = Tail
4    x = h.next
5    if (h == Head)
6      if (h == t)
7        if (x == NULL)
8          return *
9          exit
10       CAS (Tail,t,x)
11     else
12       v = x.key
13       if (CAS (Head,h,x))
14         return v
15         exit
```

deq

enq(4)



The Michael-Scott Queue Algorithm

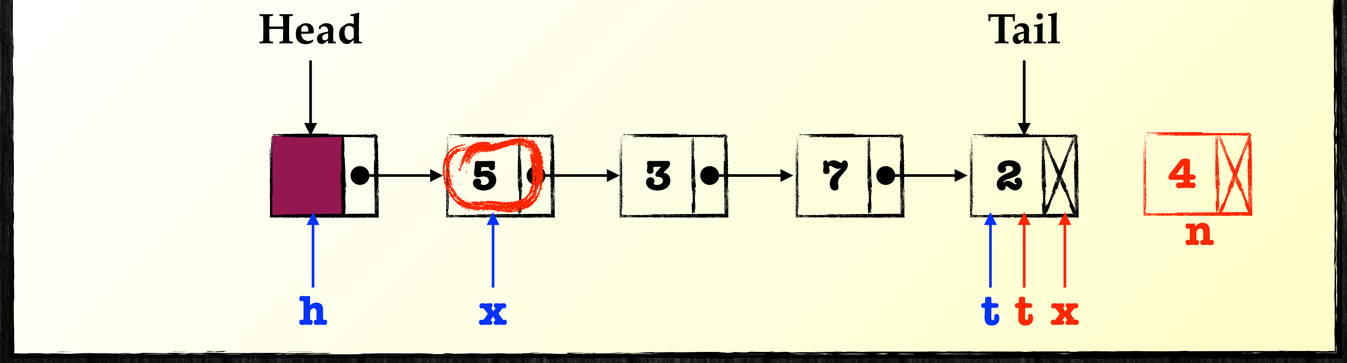
Linearization Policy

```
enq(k):
Node t,x
1  n = new Node(k,NULL)
2  while (true)
3    t = Tail
4    x = t.next
5    if (t == Tail)
6      if (x == NULL)
7        if (CAS (t.next,NULL,n))
8          CAS (Tail,t,n)
9          exit
10     else
11       CAS (Tail,t,x)
```

```
deq(k):
Node h,t,x
Integer v
1  while (true)
2    h = Head
3    t = Tail
4    x = h.next
5    if (h == Head)
6      if (h == t)
7        if (x == NULL)
8          return *
9          exit
10       CAS (Tail,t,x)
11     else
12       v = x.key
13       if (CAS (Head,h,x))
14         return v
15         exit
```

deq

enq(4)



The Michael-Scott Queue Algorithm

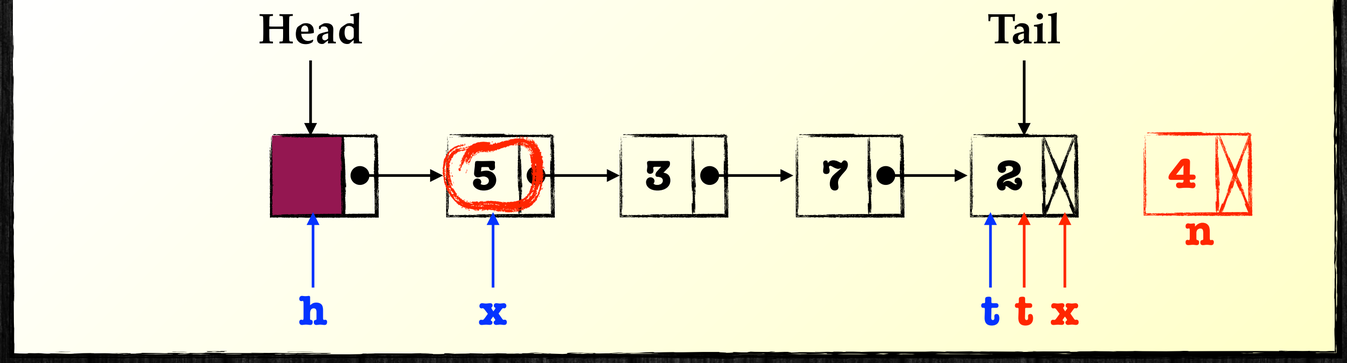
Linearization Policy

```
enq(k):
Node t,x
1  n = new Node(k,NULL)
2  while (true)
3    t = Tail
4    x = t.next
5    if (t == Tail)
6      if (x == NULL)
7        if (CAS (t.next,NULL,n))
8          CAS (Tail,t,n)
9          exit
10     else
11       CAS (Tail,t,x)
```

```
deq(k):
Node h,t,x
Integer v
1  while (true)
2    h = Head
3    t = Tail
4    x = h.next
5    if (h == Head)
6      if (h == t)
7        if (x == NULL)
8          return *
9          exit
10       CAS (Tail,t,x)
11     else
12       v = x.key
13       if (CAS (Head,h,x))
14         return v
15         exit
```

deq

enq(4)



The Michael-Scott Queue Algorithm

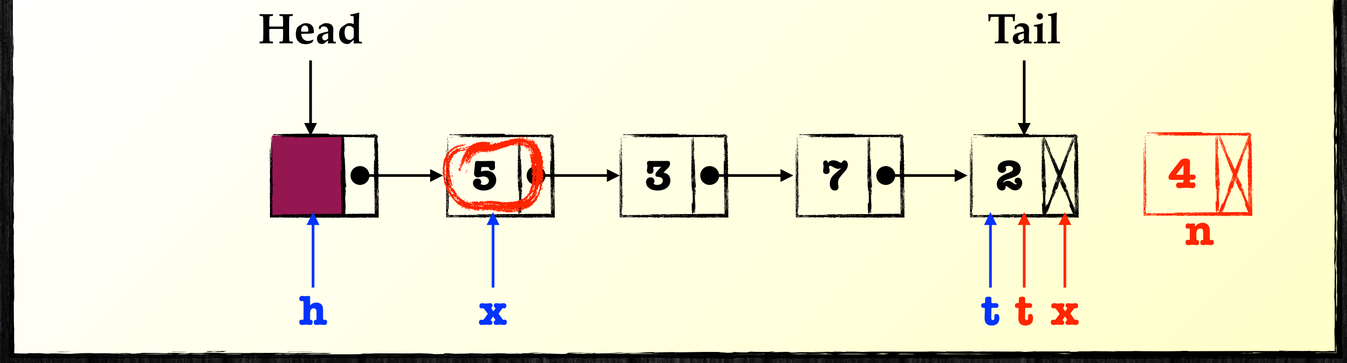
Linearization Policy

```
enq(k):
Node t,x
1  n = new Node(k,NULL)
2  while (true)
3    t = Tail
4    x = t.next
5    if (t == Tail)
6      if (x == NULL)
7        if (CAS (t.next,NULL,n))
8          CAS (Tail,t,n)
9          exit
10     else
11       CAS (Tail,t,x)
```

```
deq(k):
Node h,t,x
Integer v
1  while (true)
2    h = Head
3    t = Tail
4    x = h.next
5    if (h == Head)
6      if (h == t)
7        if (x == NULL)
8          return *
9          exit
10       CAS (Tail,t,x)
11     else
12       v = x.key
13       if (CAS (Head,h,x))
14         return v
15         exit
```

deq

enq(4)



The Michael-Scott Queue Algorithm

Linearization Policy

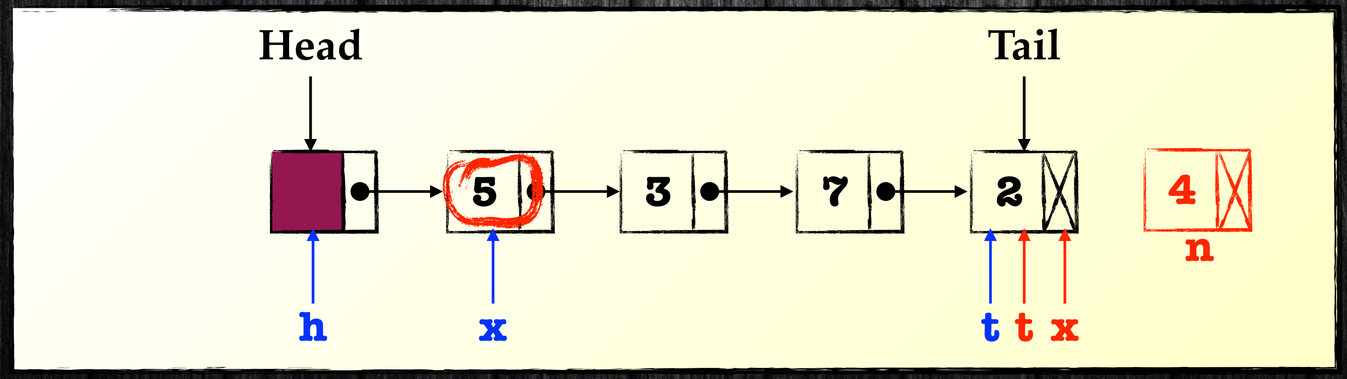
```
enq(k):
Node t,x
1  n = new Node(k,NULL)
2  while (true)
3    t = Tail
4    x = t.next
5    if (t == Tail)
6      if (x == NULL)
7        if (CAS (t.next,NULL,n))
8          CAS (Tail,t,n)
9          exit
10     else
11       CAS (Tail,t,x)
```

```
deq(k):
Node h,t,x
Integer v
1  while (true)
2    h = Head
3    t = Tail
4    x = h.next
5    if (h == Head)
6      if (h == t)
7        if (x == NULL)
8          return *
9          exit
10       CAS (Tail,t,x)
11     else
12       v = x.key
13       if (CAS (Head,h,x))
14         return v
15         exit
```

when successful

deq

enq(4)



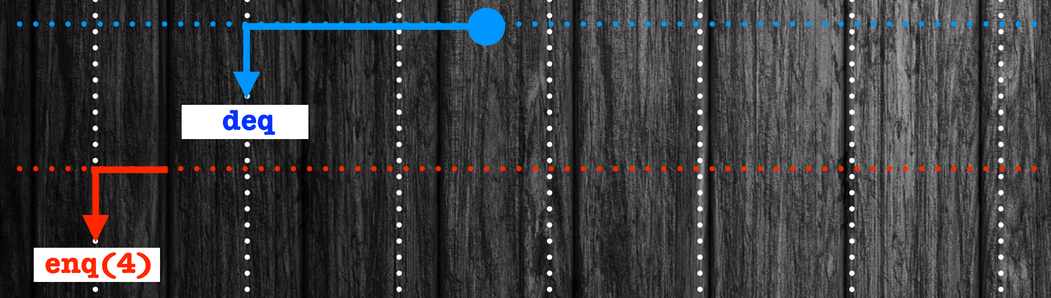
The Michael-Scott Queue Algorithm

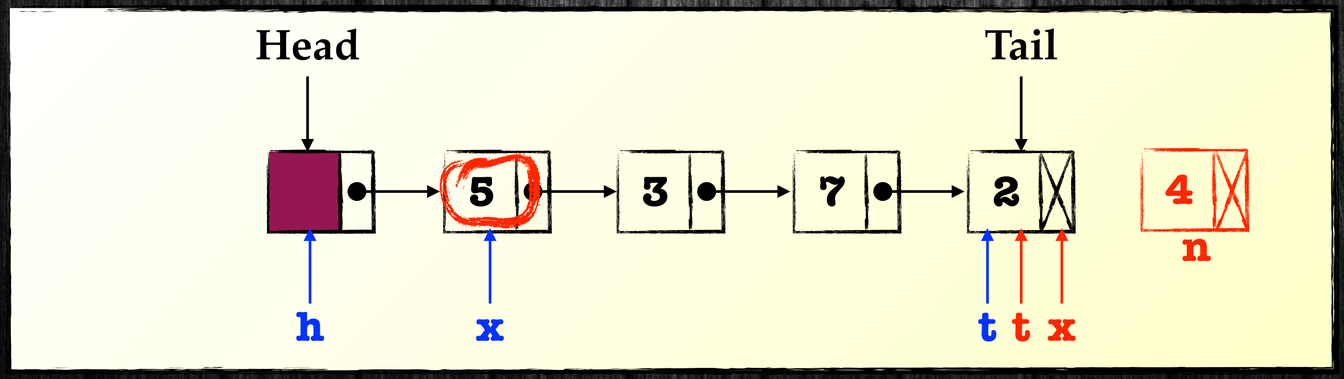
Linearization Policy

```
enq(k):
Node t,x
1  n = new Node(k,NULL)
2  while (true)
3    t = Tail
4    x = t.next
5    if (t == Tail)
6      if (x == NULL)
7        if (CAS (t.next,NULL,n))
8          CAS (Tail,t,n)
9          exit
10     else
11       CAS (Tail,t,x)
```

```
deq(k):
Node h,t,x
Integer v
1  while (true)
2    h = Head
3    t = Tail
4    x = h.next
5    if (h == Head)
6      if (h == t)
7        if (x == NULL)
8          return *
9          exit
10       CAS (Tail,t,x)
11     else
12       v = x.key
13       if (CAS (Head,h,x))
14         return v
15         exit
```

when successful



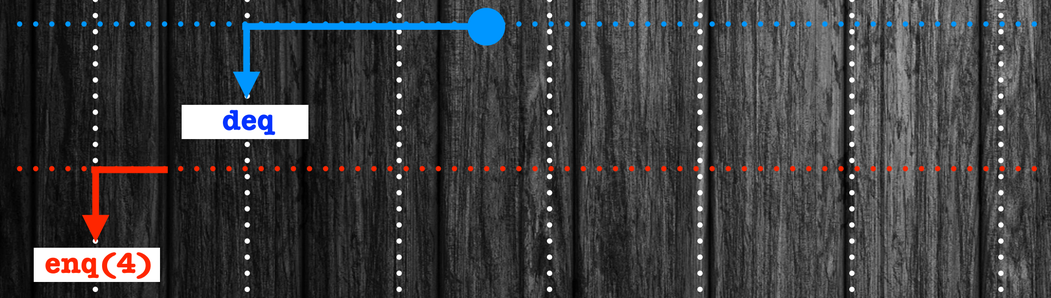


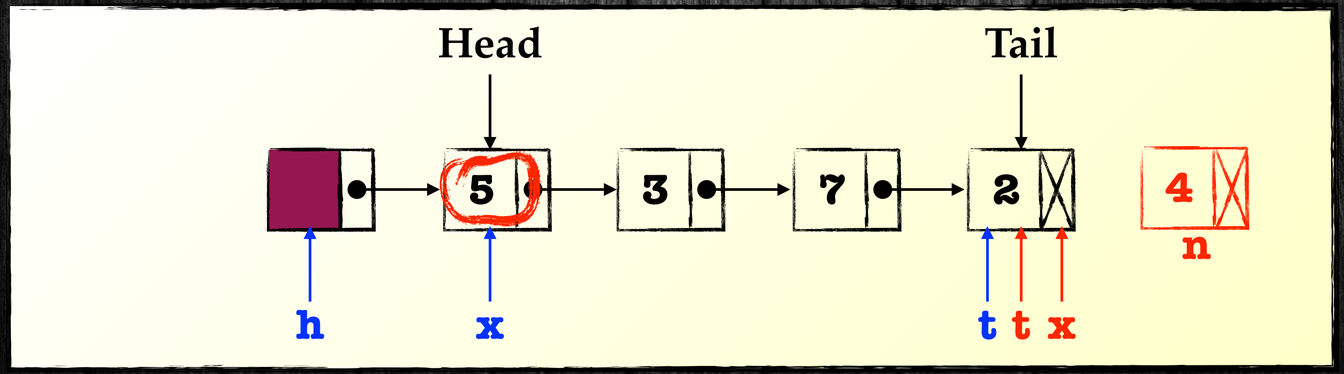
The Michael-Scott Queue Algorithm

Linearization Policy

```
enq(k):
Node t,x
1  n = new Node(k,NULL)
2  while (true)
3    t = Tail
4    x = t.next
5    if (t == Tail)
6      if (x == NULL)
7        if (CAS (t.next,NULL,n))
8          CAS (Tail,t,n)
9          exit
10     else
11       CAS (Tail,t,x)
```

```
deq(k):
Node h,t,x
Integer v
1  while (true)
2    h = Head
3    t = Tail
4    x = h.next
5    if (h == Head)
6      if (h == t)
7        if (x == NULL)
8          return *
9          exit
10       CAS (Tail,t,x)
11     else
12       v = x.key
13       if (CAS (Head,h,x))
14         return v
15         exit
```



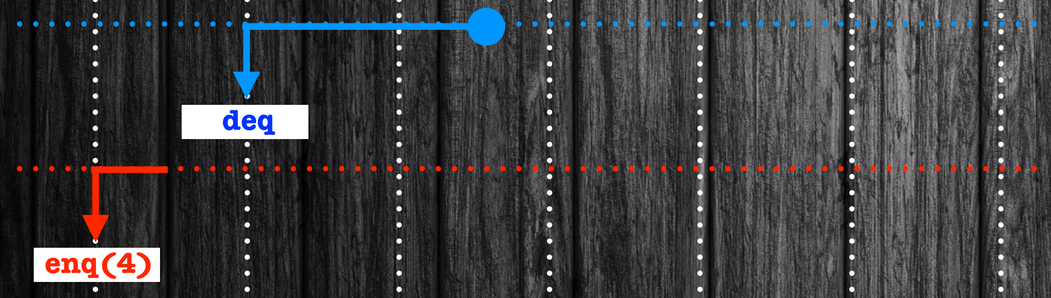


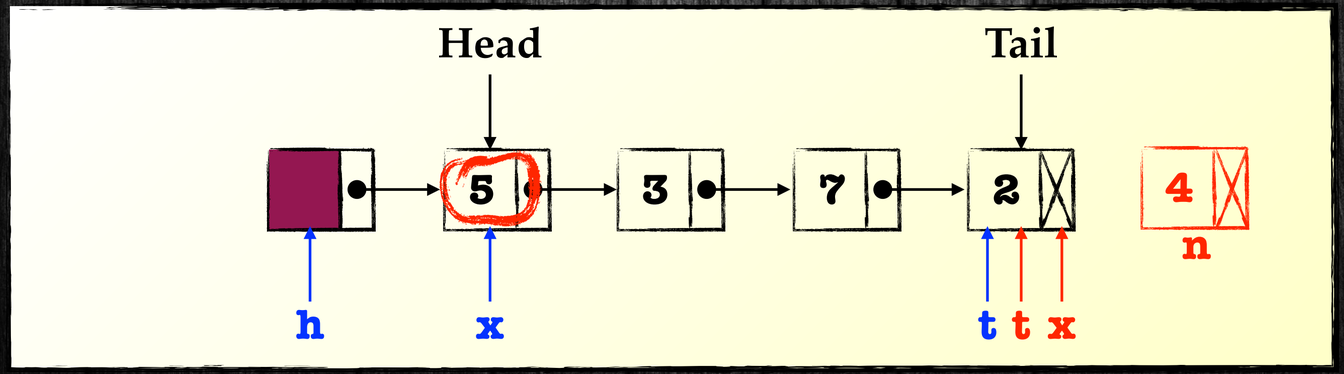
The Michael-Scott Queue Algorithm

Linearization Policy

```
enq(k):
Node t,x
1  n = new Node(k,NULL)
2  while (true)
3    t = Tail
4    x = t.next
5    if (t == Tail)
6      if (x == NULL)
7        if (CAS (t.next,NULL,n))
8          CAS (Tail,t,n)
9          exit
10     else
11       CAS (Tail,t,x)
```

```
deq(k):
Node h,t,x
Integer v
1  while (true)
2    h = Head
3    t = Tail
4    x = h.next
5    if (h == Head)
6      if (h == t)
7        if (x == NULL)
8          return *
9          exit
10       CAS (Tail,t,x)
11     else
12       v = x.key
13       if (CAS (Head,h,x))
14         return v
15         exit
```



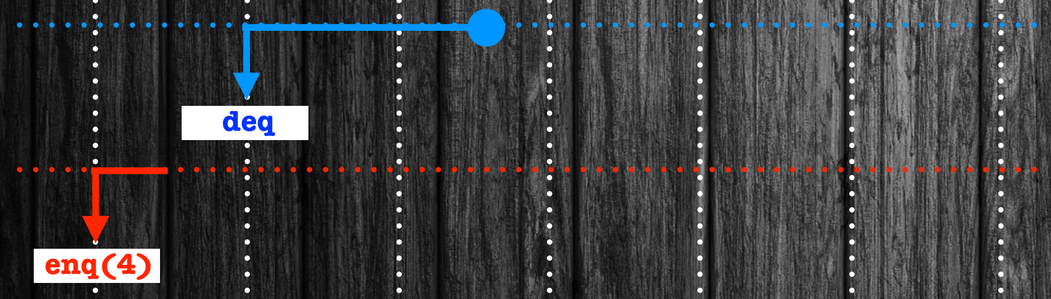


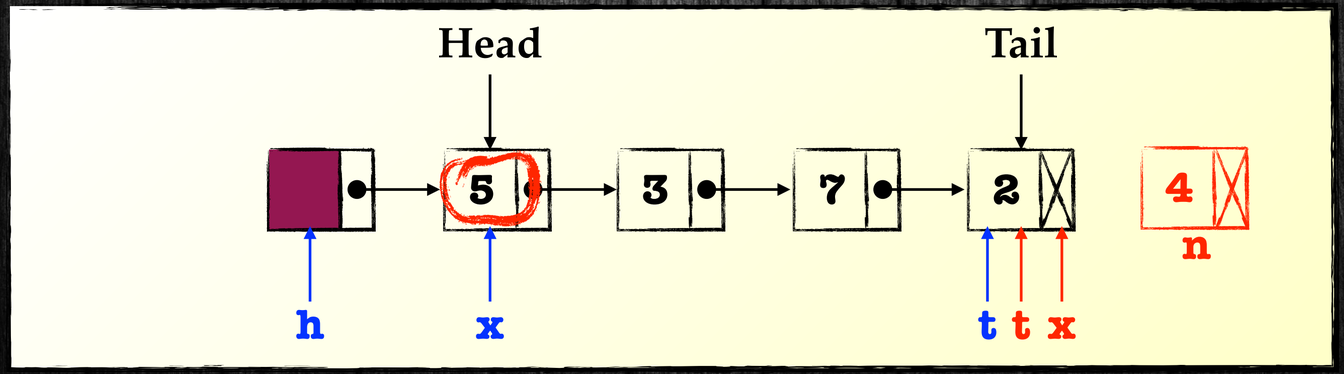
The Michael-Scott Queue Algorithm

Linearization Policy

```
enq(k):
Node t,x
1  n = new Node(k,NULL)
2  while (true)
3    t = Tail
4    x = t.next
5    if (t == Tail)
6      if (x == NULL)
7        if (CAS (t.next,NULL,n))
8          CAS (Tail,t,n)
9          exit
10     else
11       CAS (Tail,t,x)
```

```
deq(k):
Node h,t,x
Integer v
1  while (true)
2    h = Head
3    t = Tail
4    x = h.next
5    if (h == Head)
6      if (h == t)
7        if (x == NULL)
8          return *
9          exit
10       CAS (Tail,t,x)
11     else
12       v = x.key
13       if (CAS (Head,h,x))
14         return v
15         exit
```



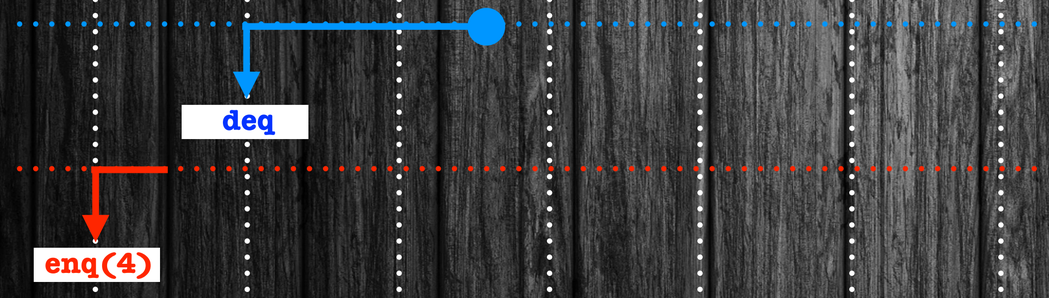


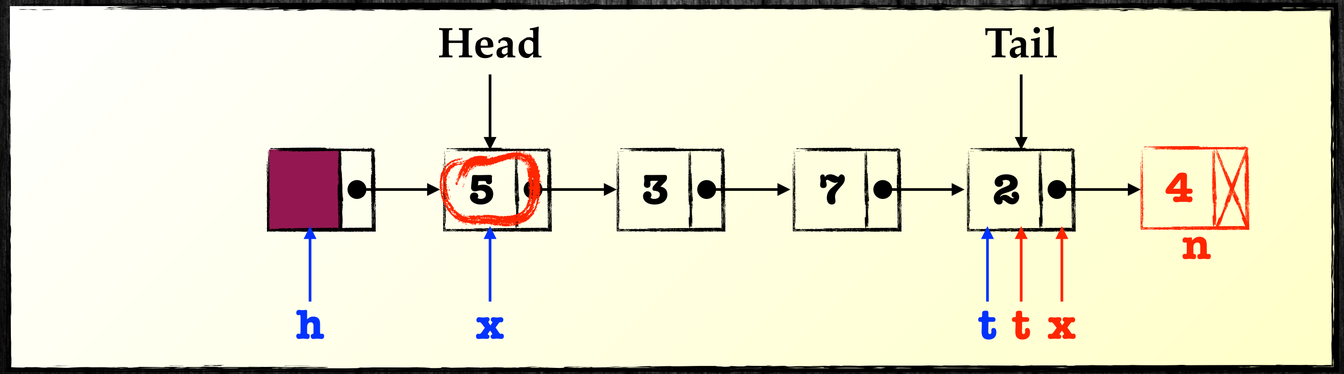
The Michael-Scott Queue Algorithm

Linearization Policy

```
enq(k):
Node t,x
1  n = new Node(k,NULL)
2  while (true)
3    t = Tail
4    x = t.next
5    if (t == Tail)
6      if (x == NULL)
7        if (CAS (t.next,NULL,n))
8          CAS (Tail,t,n)
9          exit
10     else
11       CAS (Tail,t,x)
```

```
deq(k):
Node h,t,x
Integer v
1  while (true)
2    h = Head
3    t = Tail
4    x = h.next
5    if (h == Head)
6      if (h == t)
7        if (x == NULL)
8          return *
9          exit
10       CAS (Tail,t,x)
11     else
12       v = x.key
13       if (CAS (Head,h,x))
14         return v
15         exit
```





The Michael-Scott Queue Algorithm

Linearization Policy

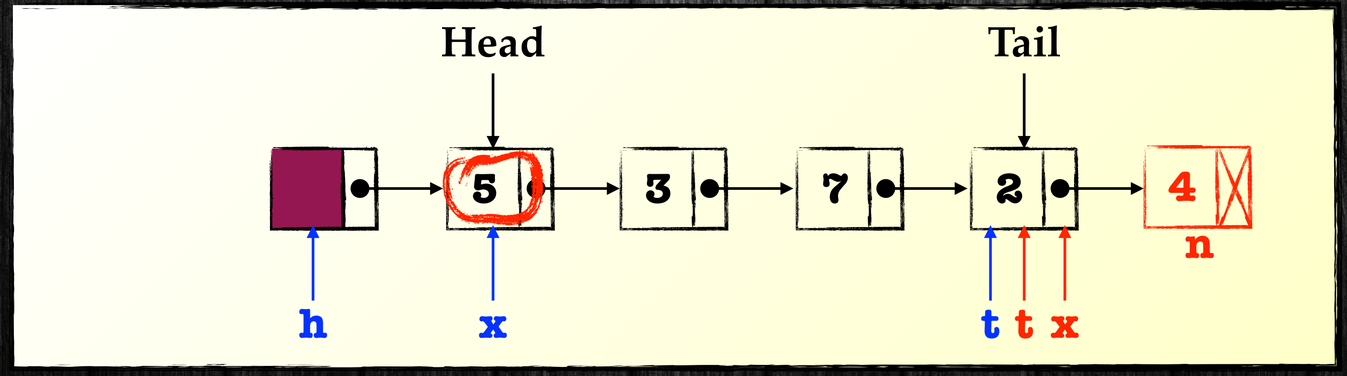
```
enq(k):
Node t,x
1  n = new Node(k,NULL)
2  while (true)
3    t = Tail
4    x = t.next
5    if (t == Tail)
6      if (x == NULL)
7        if (CAS (t.next,NULL,n))
8          CAS (Tail,t,n)
9          exit
10     else
11       CAS (Tail,t,x)
```

when successful

```
deq(k):
Node h,t,x
Integer v
1  while (true)
2    h = Head
3    t = Tail
4    x = h.next
5    if (h == Head)
6      if (h == t)
7        if (x == NULL)
8          return *
9          exit
10       CAS (Tail,t,x)
11     else
12       v = x.key
13       if (CAS (Head,h,x))
14         return v
15         exit
```

deq

enq(4)



The Michael-Scott Queue Algorithm

Linearization Policy

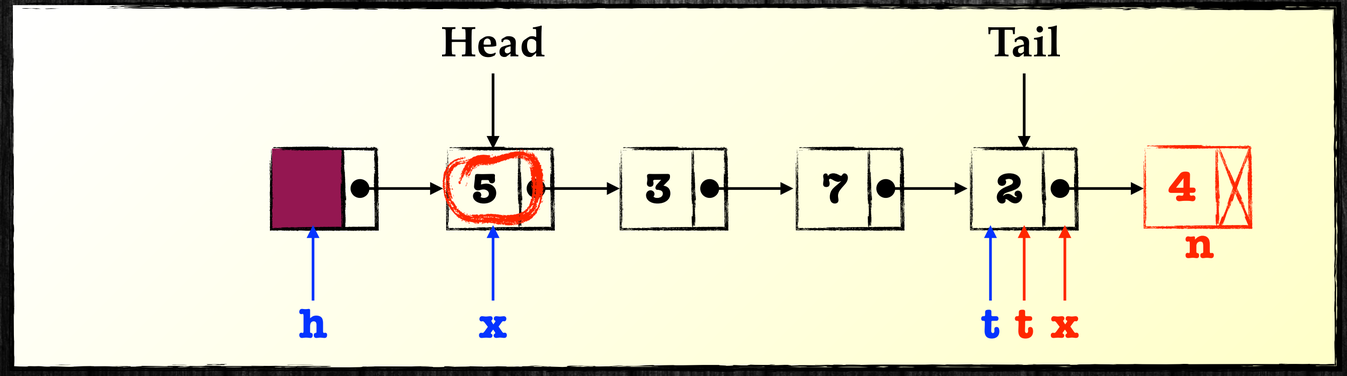
```
enq(k):
Node t,x
1  n = new Node(k,NULL)
2  while (true)
3    t = Tail
4    x = t.next
5    if (t == Tail)
6      if (x == NULL)
7        if (CAS (t.next,NULL,n))
8          CAS (Tail,t,n)
9          exit
10     else
11       CAS (Tail,t,x)
```

when successful

```
deq(k):
Node h,t,x
Integer v
1  while (true)
2    h = Head
3    t = Tail
4    x = h.next
5    if (h == Head)
6      if (h == t)
7        if (x == NULL)
8          return *
9          exit
10       CAS (Tail,t,x)
11     else
12       v = x.key
13       if (CAS (Head,h,x))
14         return v
15         exit
```

deq

enq(4)

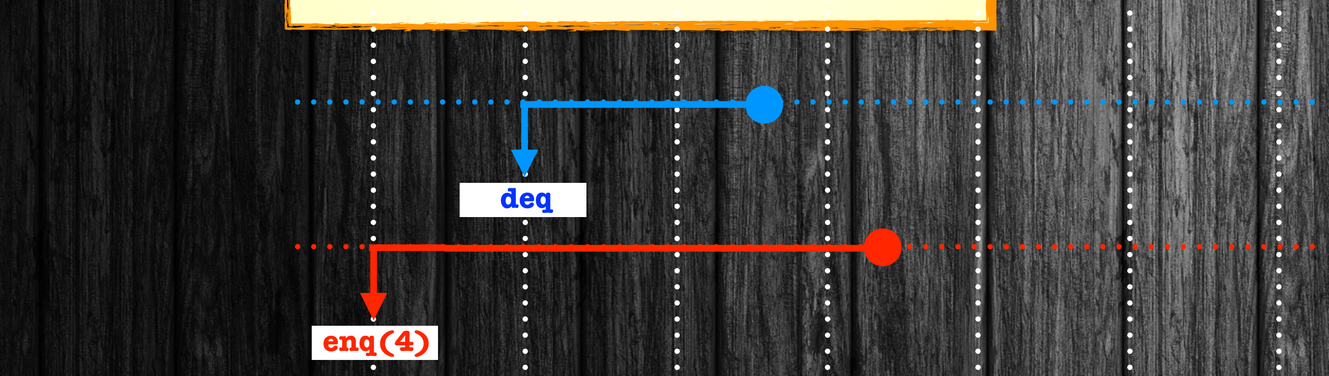


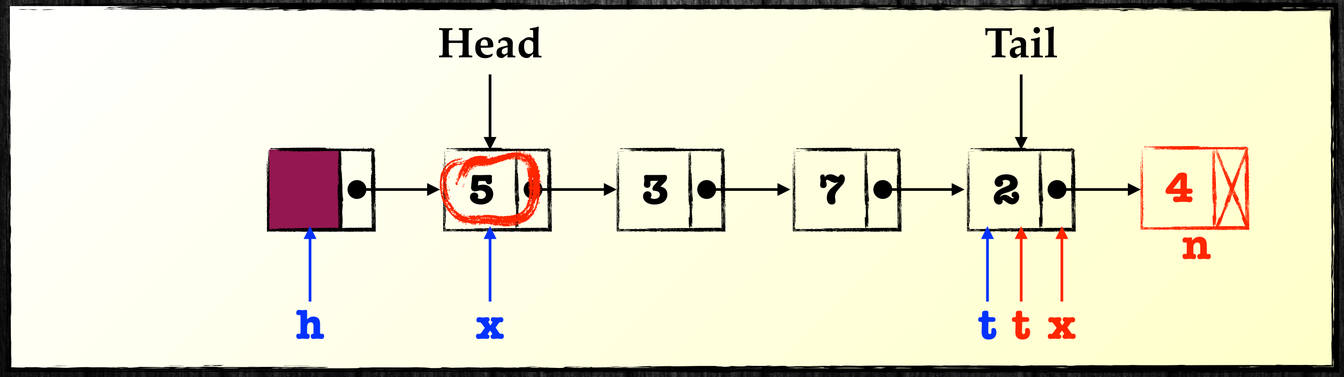
The Michael-Scott Queue Algorithm

Linearization Policy

```
enq(k):
Node t,x
1  n = new Node(k,NULL)
2  while (true)
3    t = Tail
4    x = t.next
5    if (t == Tail)
6      if (x == NULL)
7        if (CAS (t.next,NULL,n))
8          CAS (Tail,t,n)
9          exit
10     else
11       CAS (Tail,t,x)
```

```
deq(k):
Node h,t,x
Integer v
1  while (true)
2    h = Head
3    t = Tail
4    x = h.next
5    if (h == Head)
6      if (h == t)
7        if (x == NULL)
8          return *
9          exit
10       CAS (Tail,t,x)
11     else
12       v = x.key
13       if (CAS (Head,h,x))
14         return v
15         exit
```



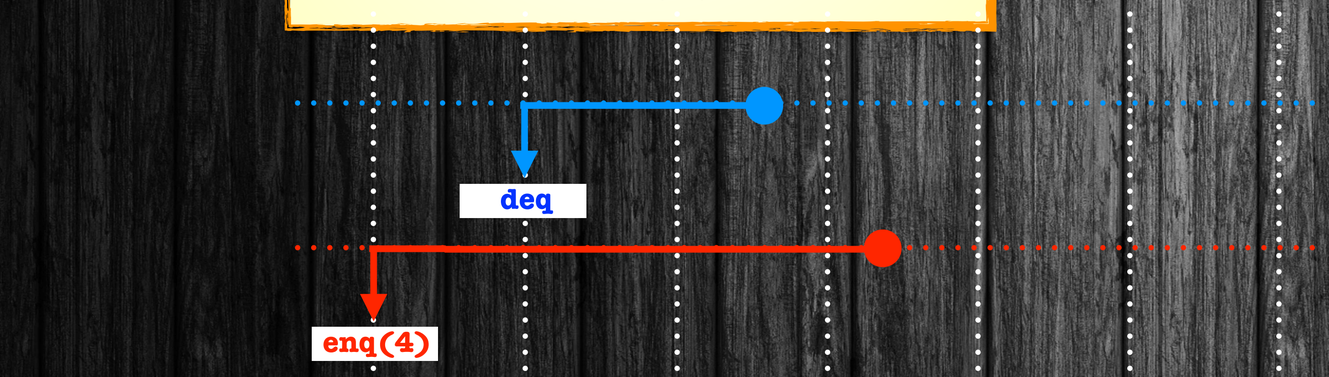


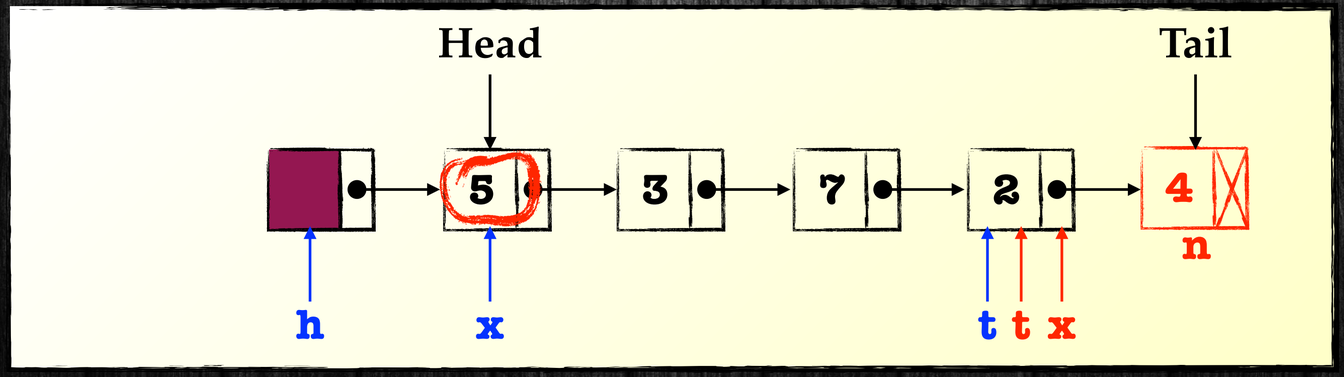
The Michael-Scott Queue Algorithm

Linearization Policy

```
enq(k):
Node t,x
1  n = new Node(k,NULL)
2  while (true)
3    t = Tail
4    x = t.next
5    if (t == Tail)
6      if (x == NULL)
7        if (CAS (t.next,NULL,n))
8          CAS (Tail,t,n)
9          exit
10     else
11       CAS (Tail,t,x)
```

```
deq(k):
Node h,t,x
Integer v
1  while (true)
2    h = Head
3    t = Tail
4    x = h.next
5    if (h == Head)
6      if (h == t)
7        if (x == NULL)
8          return *
9          exit
10       CAS (Tail,t,x)
11     else
12       v = x.key
13       if (CAS (Head,h,x))
14         return v
15         exit
```



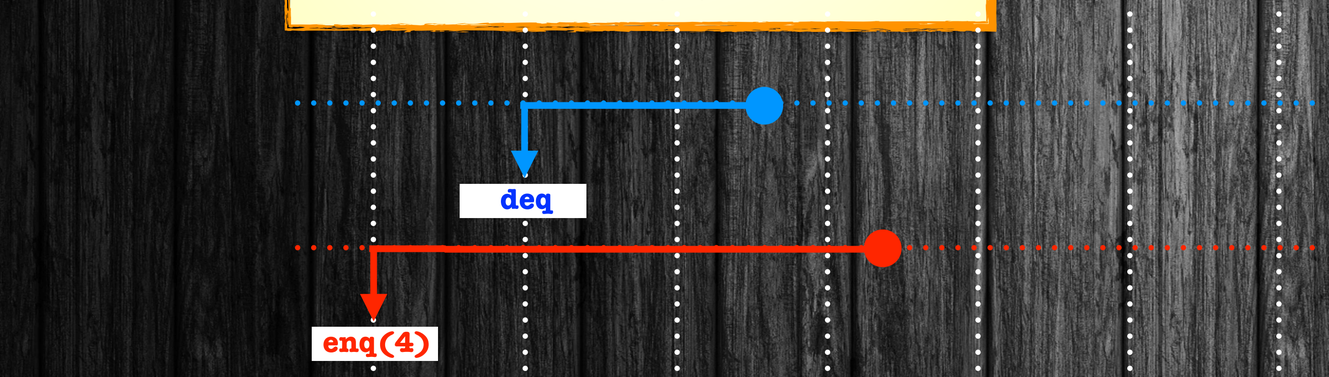


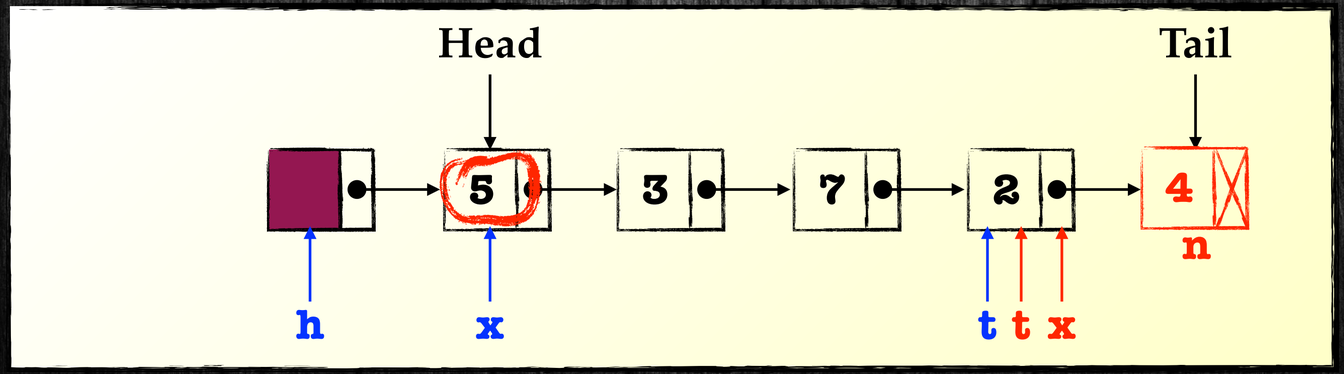
The Michael-Scott Queue Algorithm

Linearization Policy

```
enq(k):
Node t,x
1  n = new Node(k,NULL)
2  while (true)
3    t = Tail
4    x = t.next
5    if (t == Tail)
6      if (x == NULL)
7        if (CAS (t.next,NULL,n))
8          CAS (Tail,t,n)
9          exit
10     else
11       CAS (Tail,t,x)
```

```
deq(k):
Node h,t,x
Integer v
1  while (true)
2    h = Head
3    t = Tail
4    x = h.next
5    if (h == Head)
6      if (h == t)
7        if (x == NULL)
8          return *
9          exit
10       CAS (Tail,t,x)
11     else
12       v = x.key
13       if (CAS (Head,h,x))
14         return v
15         exit
```



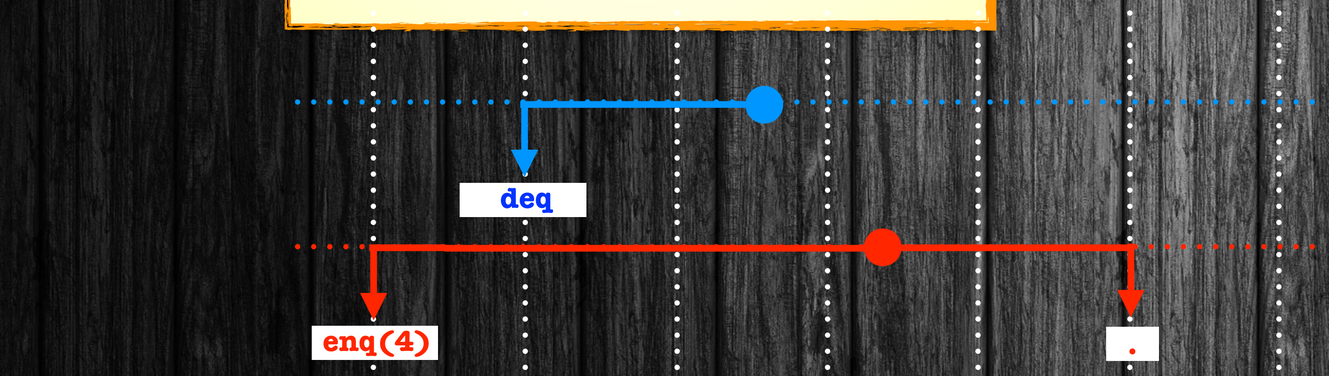


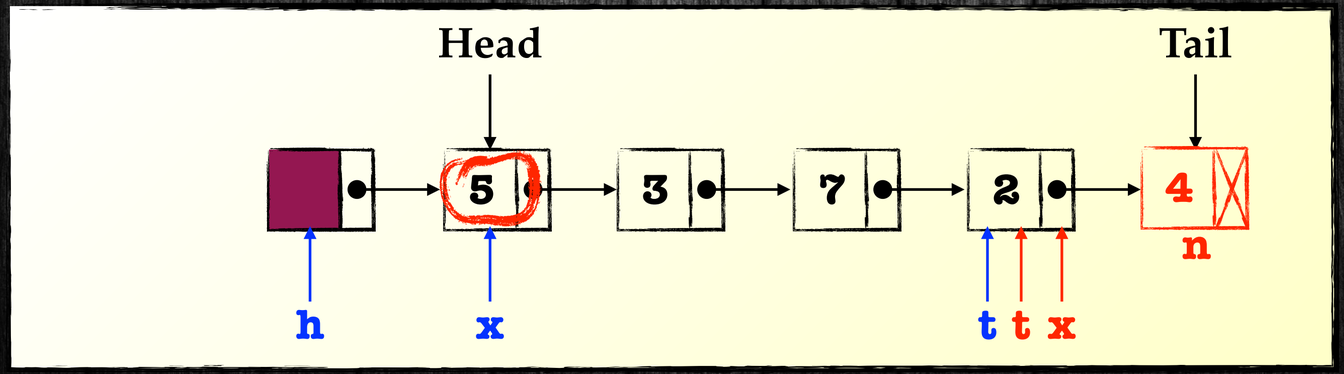
The Michael-Scott Queue Algorithm

Linearization Policy

```
enq(k):
Node t,x
1  n = new Node(k,NULL)
2  while (true)
3    t = Tail
4    x = t.next
5    if (t == Tail)
6      if (x == NULL)
7        if (CAS (t.next,NULL,n))
8          CAS (Tail,t,n)
9          exit
10     else
11       CAS (Tail,t,x)
```

```
deq(k):
Node h,t,x
Integer v
1  while (true)
2    h = Head
3    t = Tail
4    x = h.next
5    if (h == Head)
6      if (h == t)
7        if (x == NULL)
8          return *
9          exit
10       CAS (Tail,t,x)
11     else
12       v = x.key
13       if (CAS (Head,h,x))
14         return v
15         exit
```



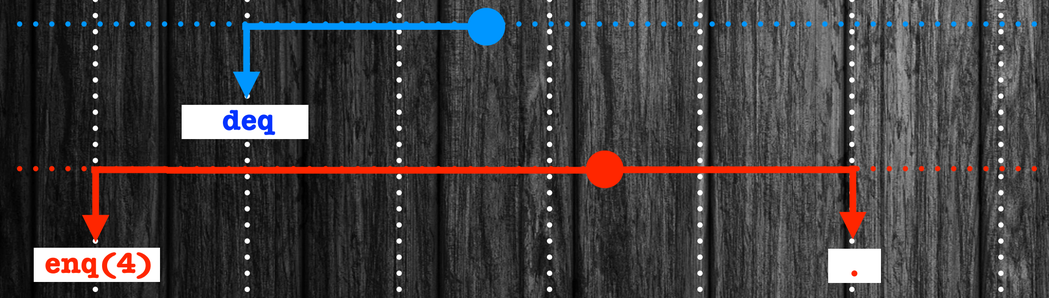


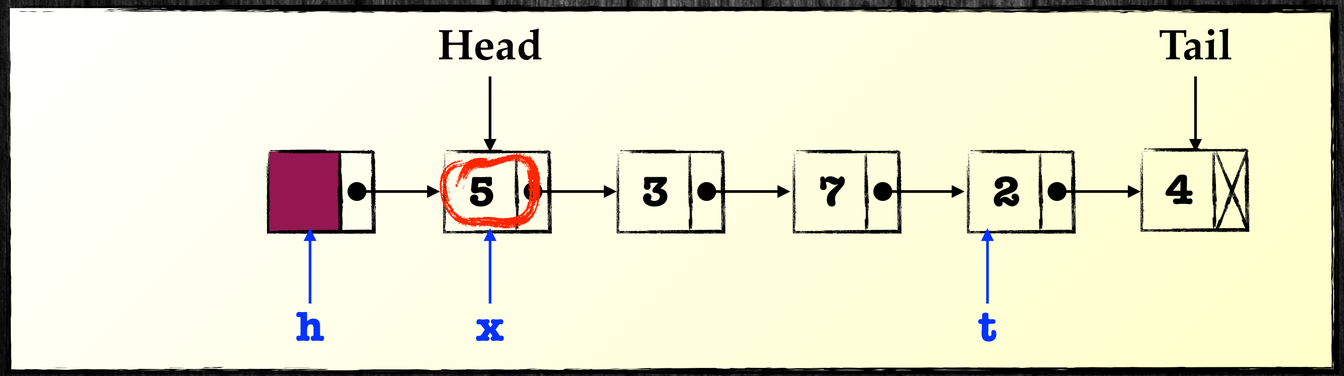
The Michael-Scott Queue Algorithm

Linearization Policy

```
enq(k):
Node t,x
1  n = new Node(k,NULL)
2  while (true)
3    t = Tail
4    x = t.next
5    if (t == Tail)
6      if (x == NULL)
7        if (CAS (t.next,NULL,n))
8          CAS (Tail,t,n)
9          exit
10     else
11       CAS (Tail,t,x)
```

```
deq(k):
Node h,t,x
Integer v
1  while (true)
2    h = Head
3    t = Tail
4    x = h.next
5    if (h == Head)
6      if (h == t)
7        if (x == NULL)
8          return *
9          exit
10       CAS (Tail,t,x)
11     else
12       v = x.key
13       if (CAS (Head,h,x))
14         return v
15         exit
```



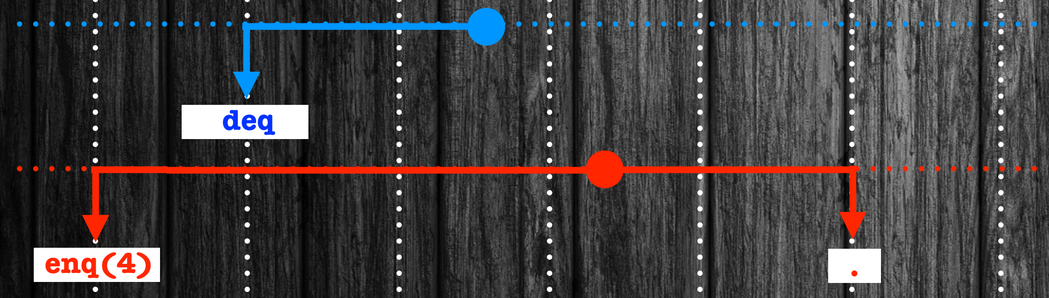


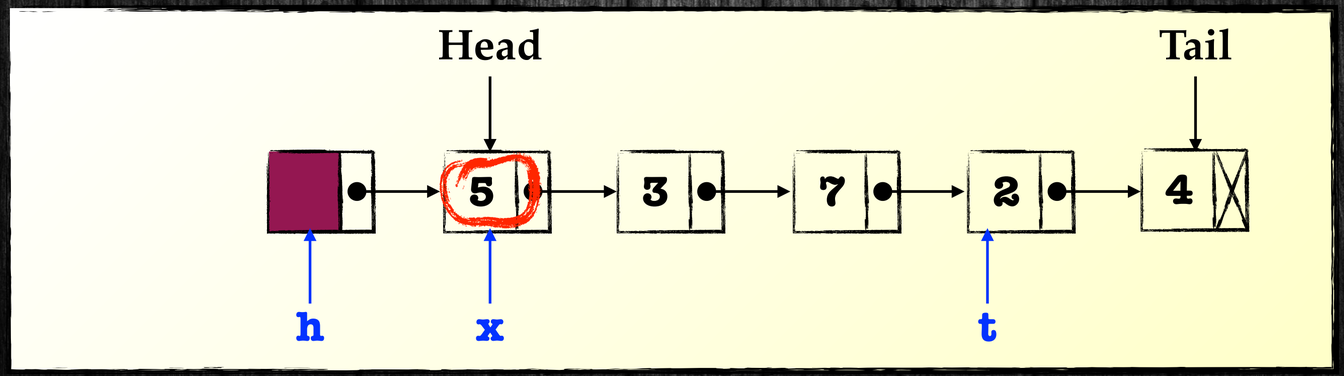
The Michael-Scott Queue Algorithm

Linearization Policy

```
enq(k):
Node t,x
1  n = new Node(k,NULL)
2  while (true)
3    t = Tail
4    x = t.next
5    if (t == Tail)
6      if (x == NULL)
7        if (CAS (t.next,NULL,n))
8          CAS (Tail,t,n)
9          exit
10     else
11       CAS (Tail,t,x)
```

```
deq(k):
Node h,t,x
Integer v
1  while (true)
2    h = Head
3    t = Tail
4    x = h.next
5    if (h == Head)
6      if (h == t)
7        if (x == NULL)
8          return *
9          exit
10       CAS (Tail,t,x)
11     else
12       v = x.key
13       if (CAS (Head,h,x))
14         return v
15         exit
```



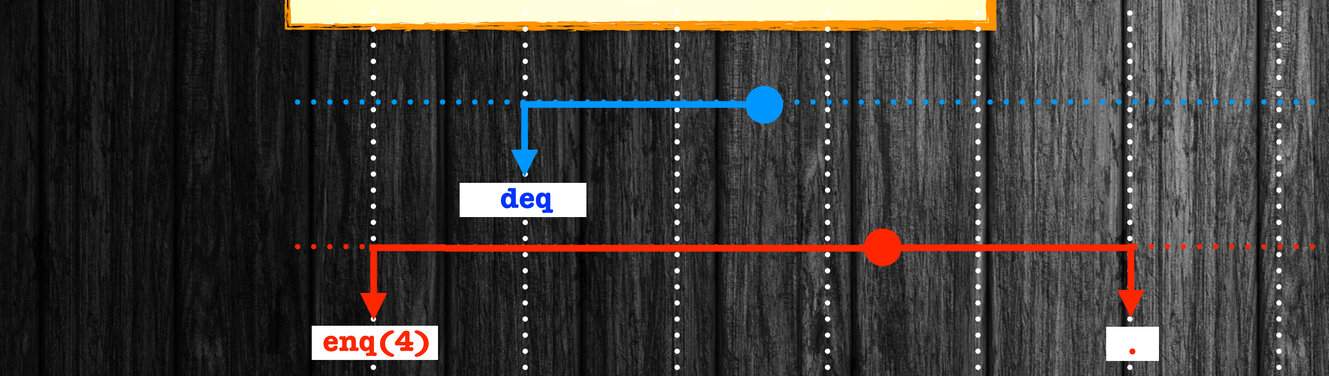


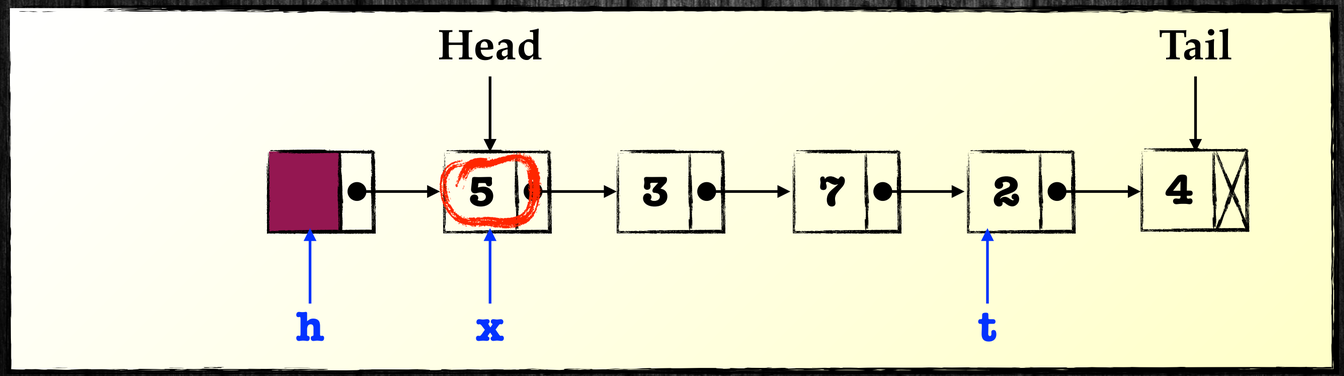
The Michael-Scott Queue Algorithm

Linearization Policy

```
enq(k):
Node t,x
1  n = new Node(k,NULL)
2  while (true)
3    t = Tail
4    x = t.next
5    if (t == Tail)
6      if (x == NULL)
7        if (CAS (t.next,NULL,n))
8          CAS (Tail,t,n)
9          exit
10     else
11       CAS (Tail,t,x)
```

```
deq(k):
Node h,t,x
Integer v
1  while (true)
2    h = Head
3    t = Tail
4    x = h.next
5    if (h == Head)
6      if (h == t)
7        if (x == NULL)
8          return *
9          exit
10       CAS (Tail,t,x)
11     else
12       v = x.key
13       if (CAS (Head,h,x))
14         return v
15         exit
```



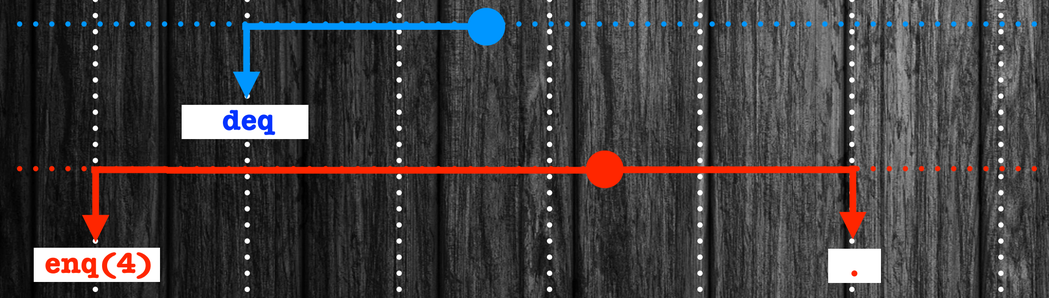


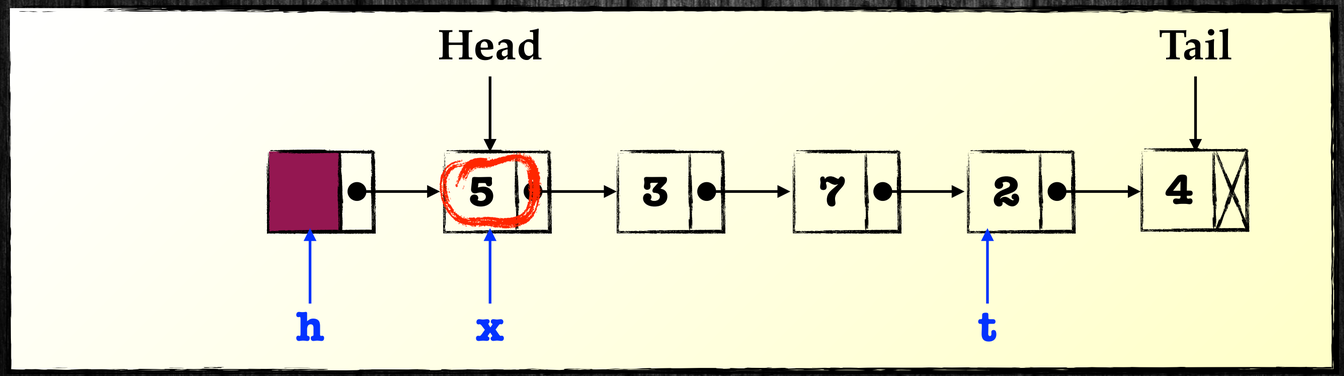
The Michael-Scott Queue Algorithm

Linearization Policy

```
enq(k):
Node t,x
1  n = new Node(k,NULL)
2  while (true)
3    t = Tail
4    x = t.next
5    if (t == Tail)
6      if (x == NULL)
7        if (CAS (t.next,NULL,n))
8          CAS (Tail,t,n)
9          exit
10     else
11       CAS (Tail,t,x)
```

```
deq(k):
Node h,t,x
Integer v
1  while (true)
2    h = Head
3    t = Tail
4    x = h.next
5    if (h == Head)
6      if (h == t)
7        if (x == NULL)
8          return *
9          exit
10       CAS (Tail,t,x)
11     else
12       v = x.key
13       if (CAS (Head,h,x))
14         return v
15         exit
```



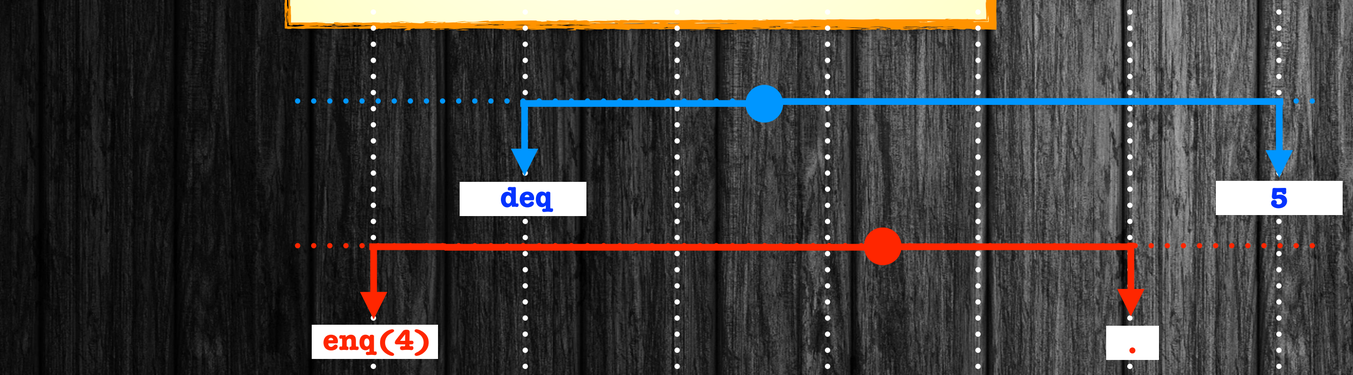


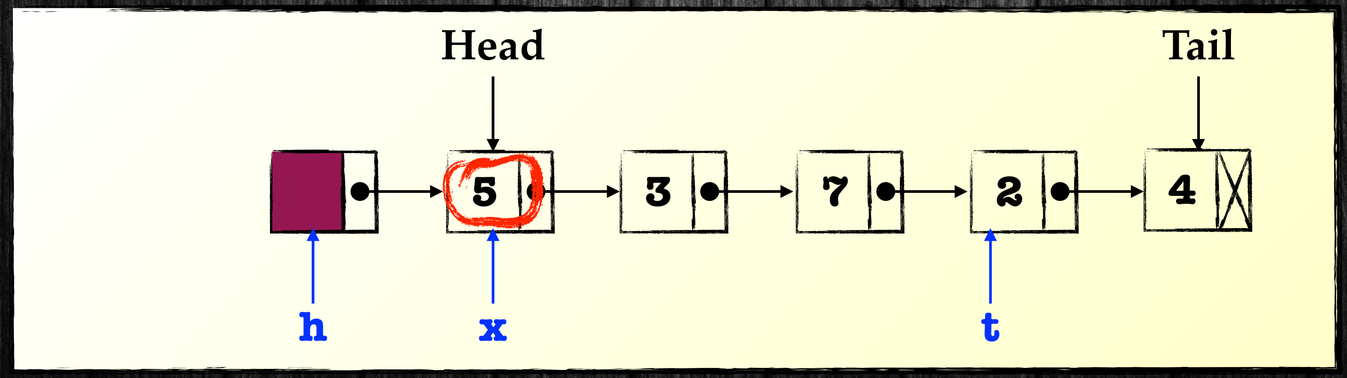
The Michael-Scott Queue Algorithm

Linearization Policy

```
enq(k):
Node t,x
1  n = new Node(k,NULL)
2  while (true)
3    t = Tail
4    x = t.next
5    if (t == Tail)
6      if (x == NULL)
7        if (CAS (t.next,NULL,n))
8          CAS (Tail,t,n)
9          exit
10     else
11       CAS (Tail,t,x)
```

```
deq(k):
Node h,t,x
Integer v
1  while (true)
2    h = Head
3    t = Tail
4    x = h.next
5    if (h == Head)
6      if (h == t)
7        if (x == NULL)
8          return *
9          exit
10       CAS (Tail,t,x)
11     else
12       v = x.key
13       if (CAS (Head,h,x))
14         return v
15         exit
```





The Michael-Scott Queue Algorithm

Linearization Policy

```
enq(k):
Node t,x
1  n = new Node(k,NULL)
2  while (true)
3    t = Tail
4    x = t.next
5    if (t == Tail)
6      if (x == NULL)
7        if (CAS (t.next,NULL,n))
8          CAS (Tail,t,n)
9          exit
10     else
11       CAS (Tail,t,x)
```

```
deq(k):
Node h,t,x
Integer v
1  while (true)
2    h = Head
3    t = Tail
4    x = h.next
5    if (h == Head)
6      if (h == t)
7        if (x == NULL)
8          return *
9          exit
10       CAS (Tail,t,x)
11     else
12       v = x.key
13       if (CAS (Head,h,x))
14         return v
15         exit
```

