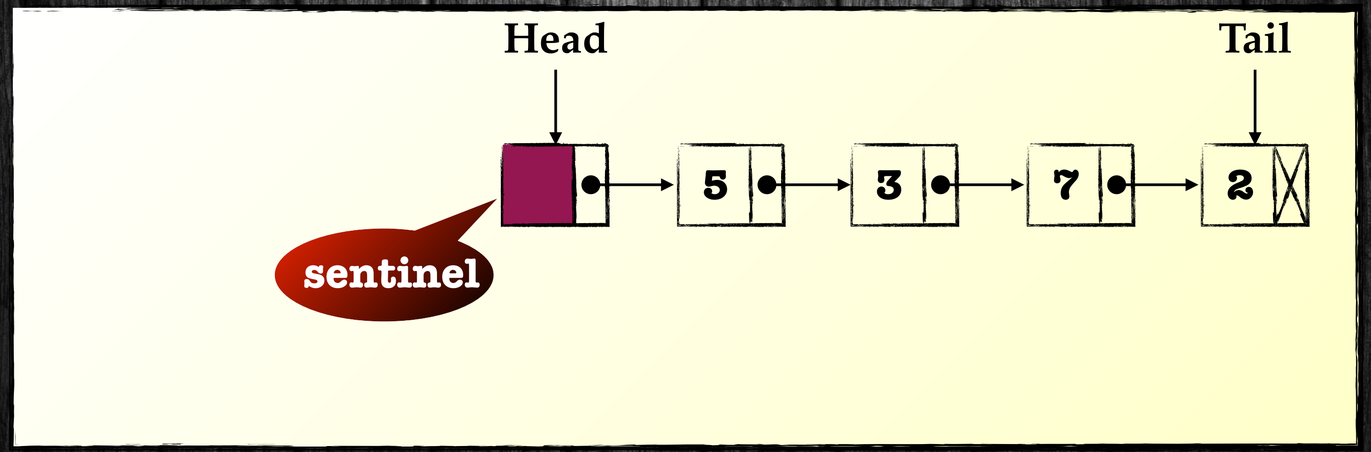
Outline

- 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

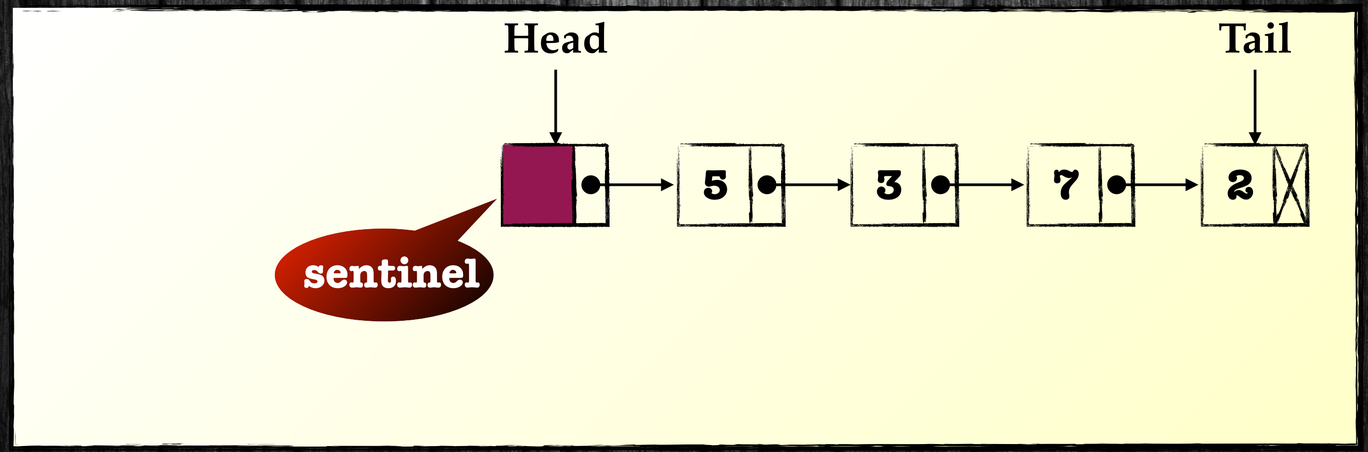
- Overview
- Abstract Data Types
- Sequential Programs
- Concurrent Programs
- Coarse-Grained Synchronization
- Fine-Grained Synchronization
- Optimistic Synchronization
- Lazy Synchronization
- Treiber Stack Algorithm
- Michael&Scott Queue Algorithm

The Michael-Scott Queue Algorithm



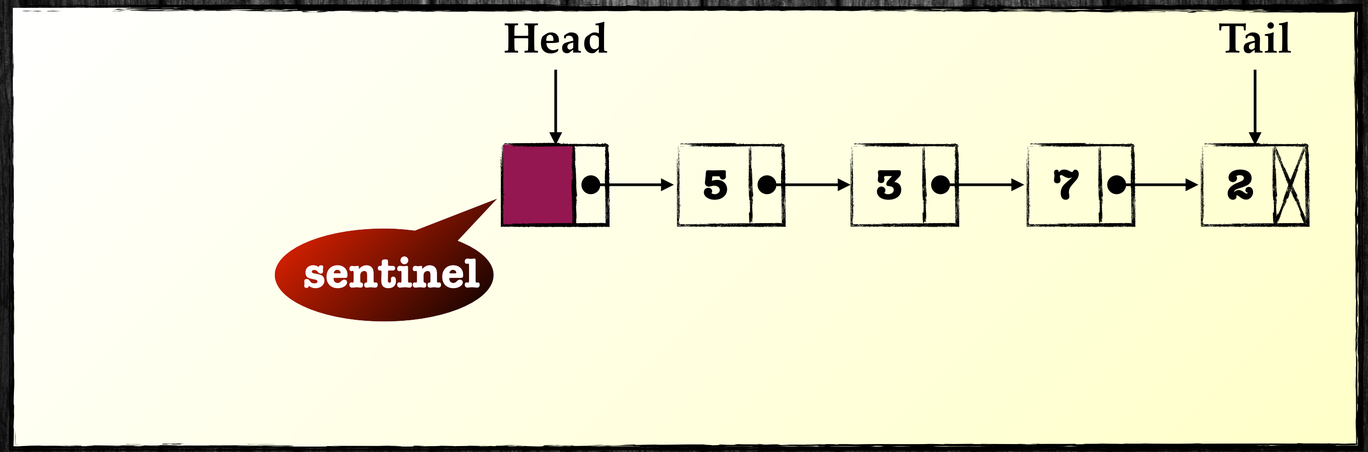
The Michael-Scott Queue Algorithm





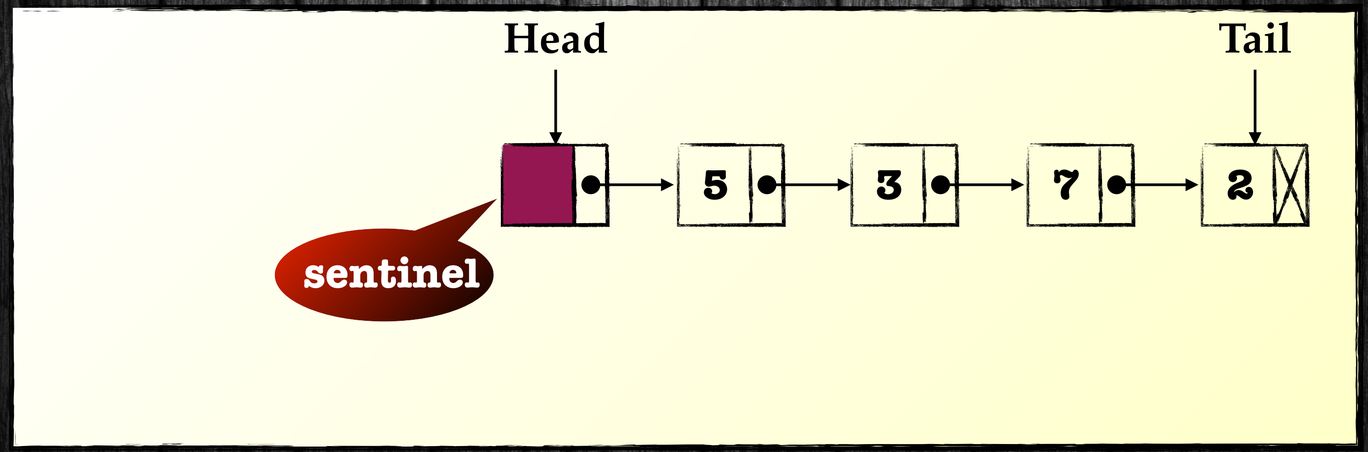
The Michael-Scott Queue Algorithm



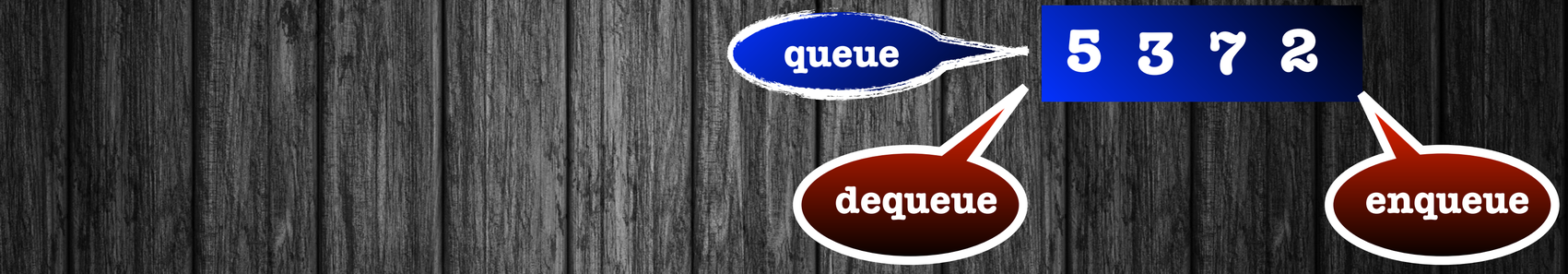


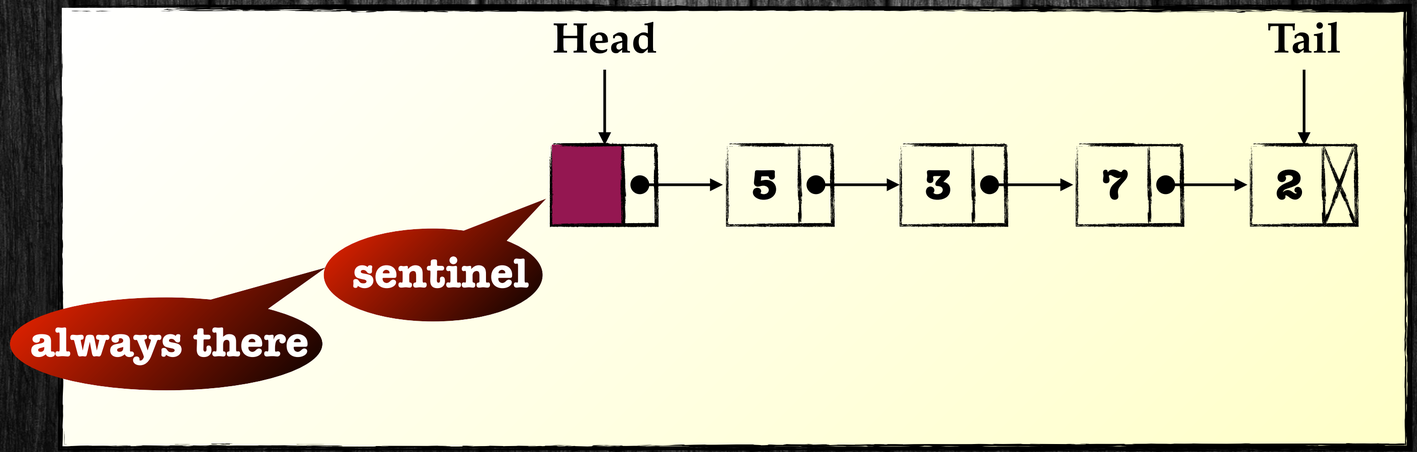
The Michael-Scott Queue Algorithm





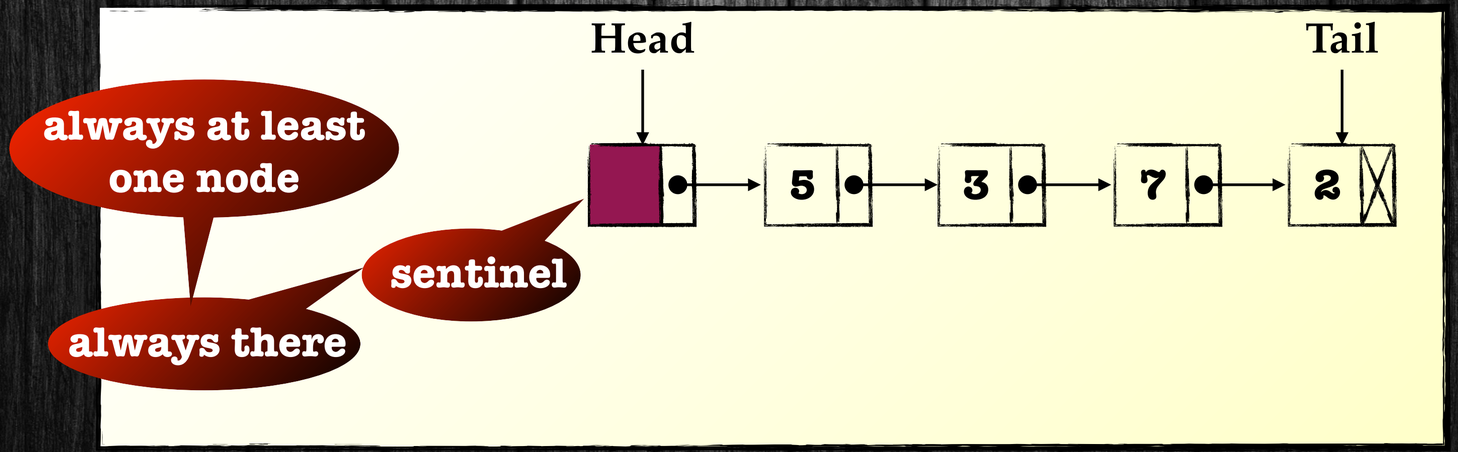
The Michael-Scott Queue Algorithm



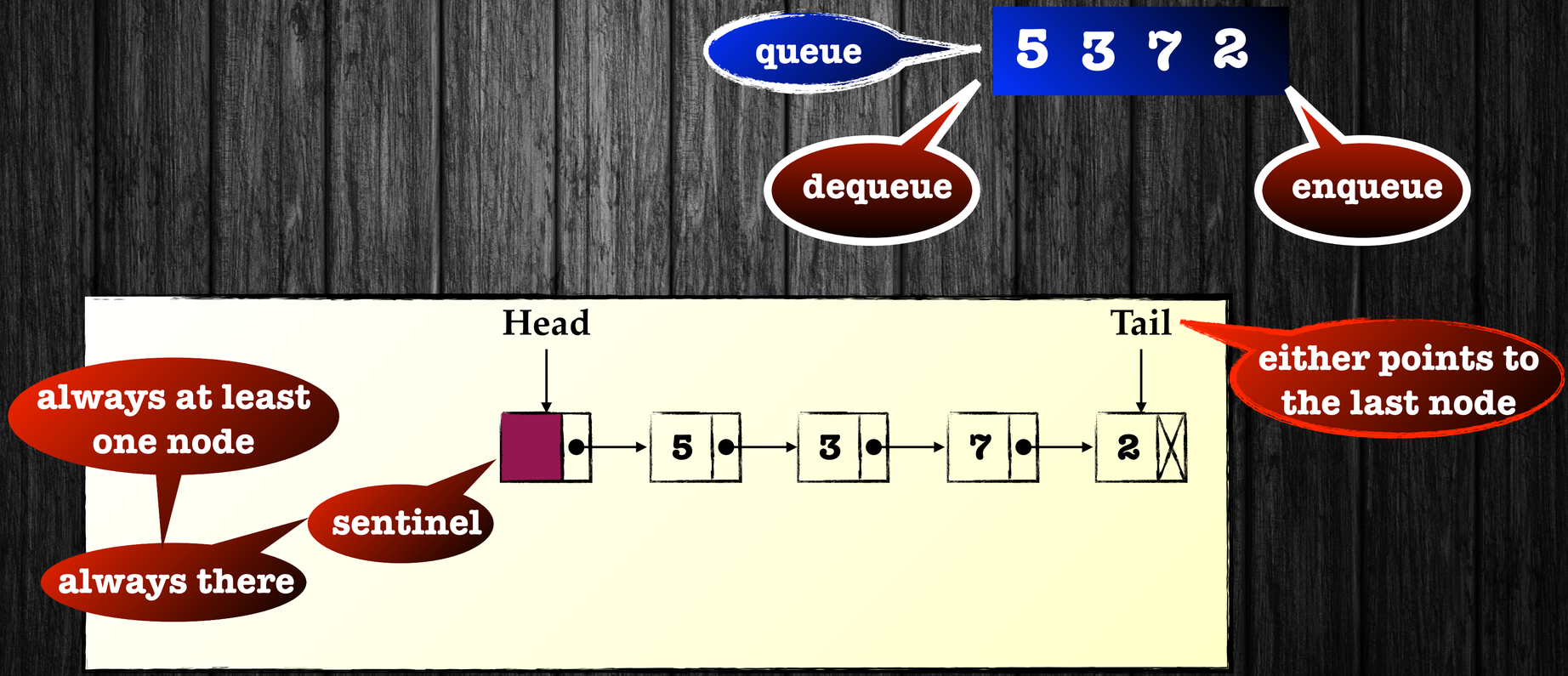


The Michael-Scott Queue Algorithm

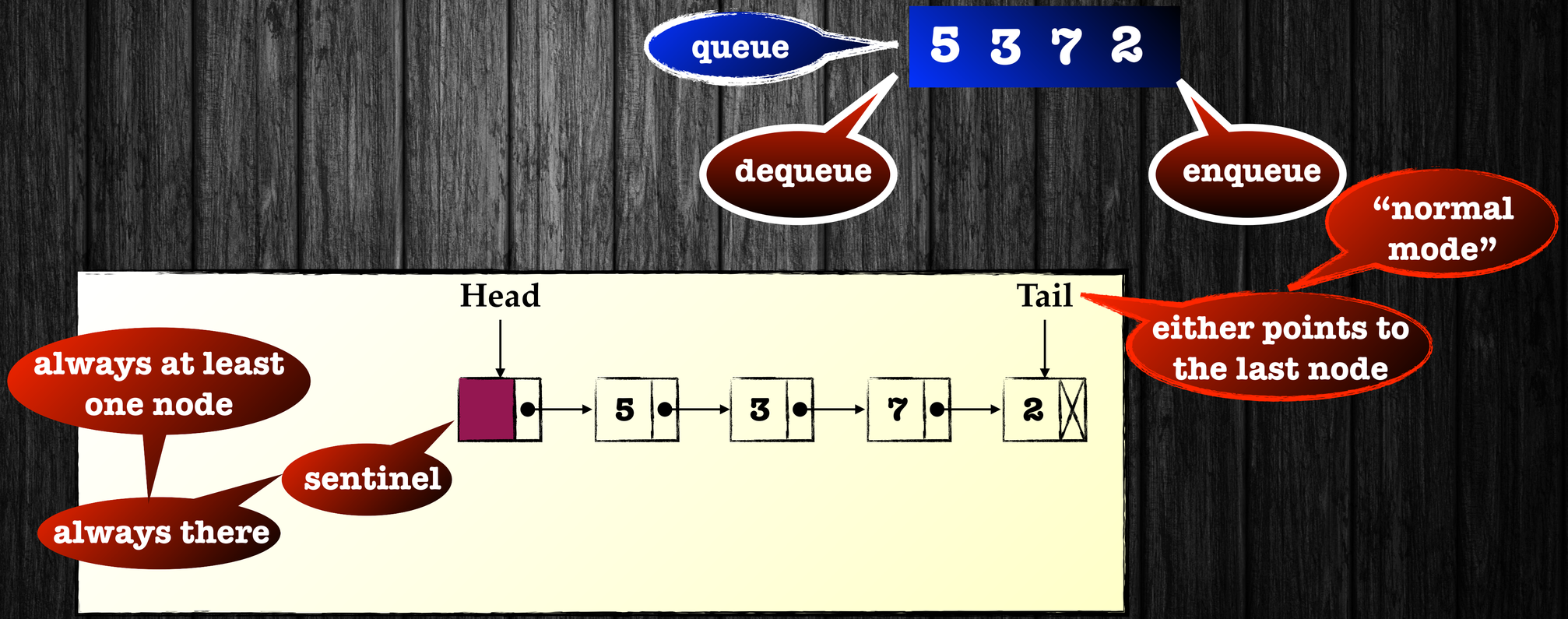




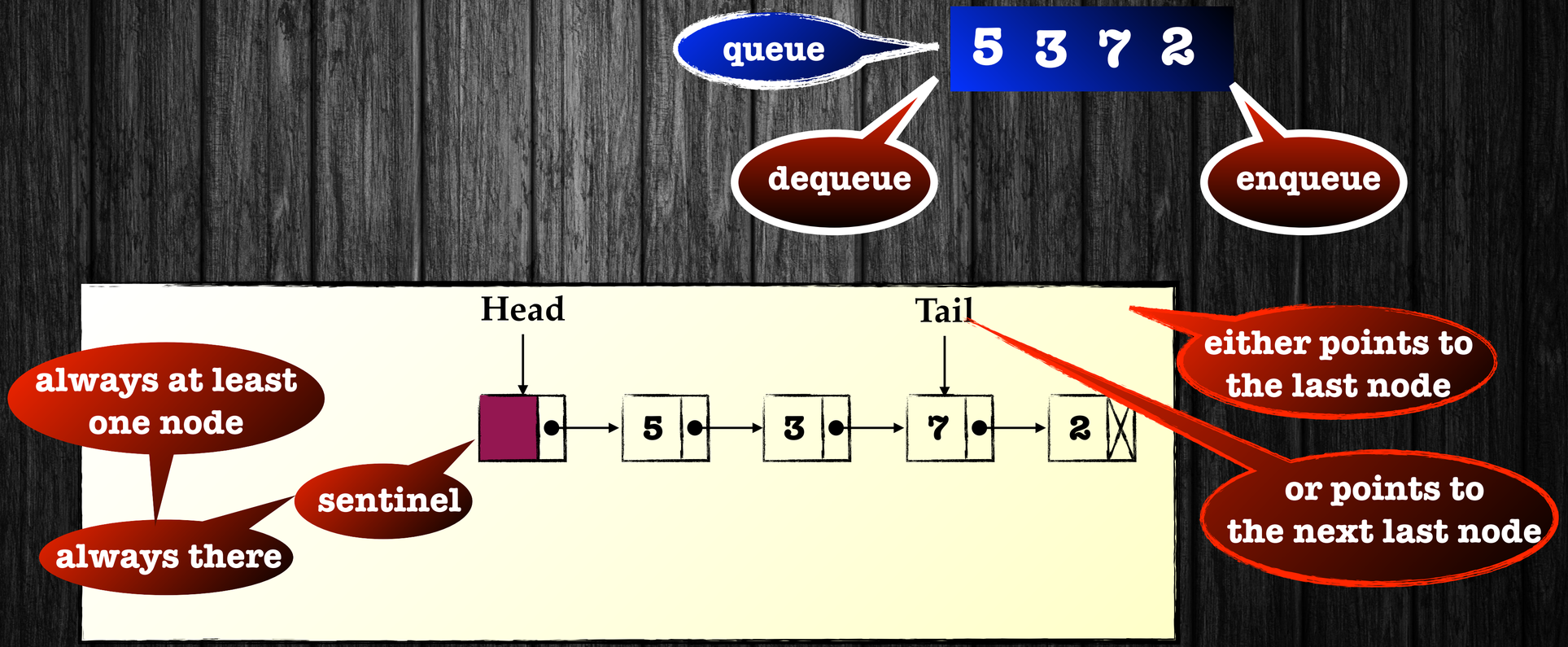
The Michael-Scott Queue Algorithm



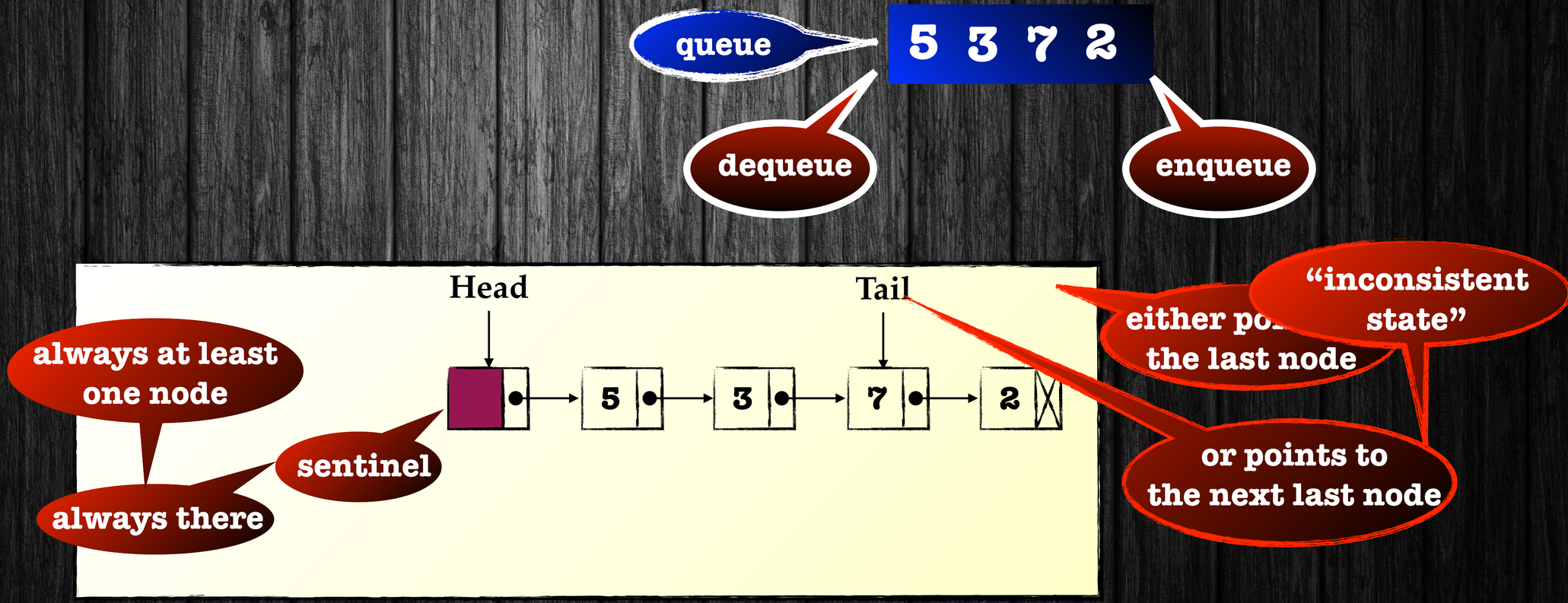
The Michael-Scott Queue Algorithm



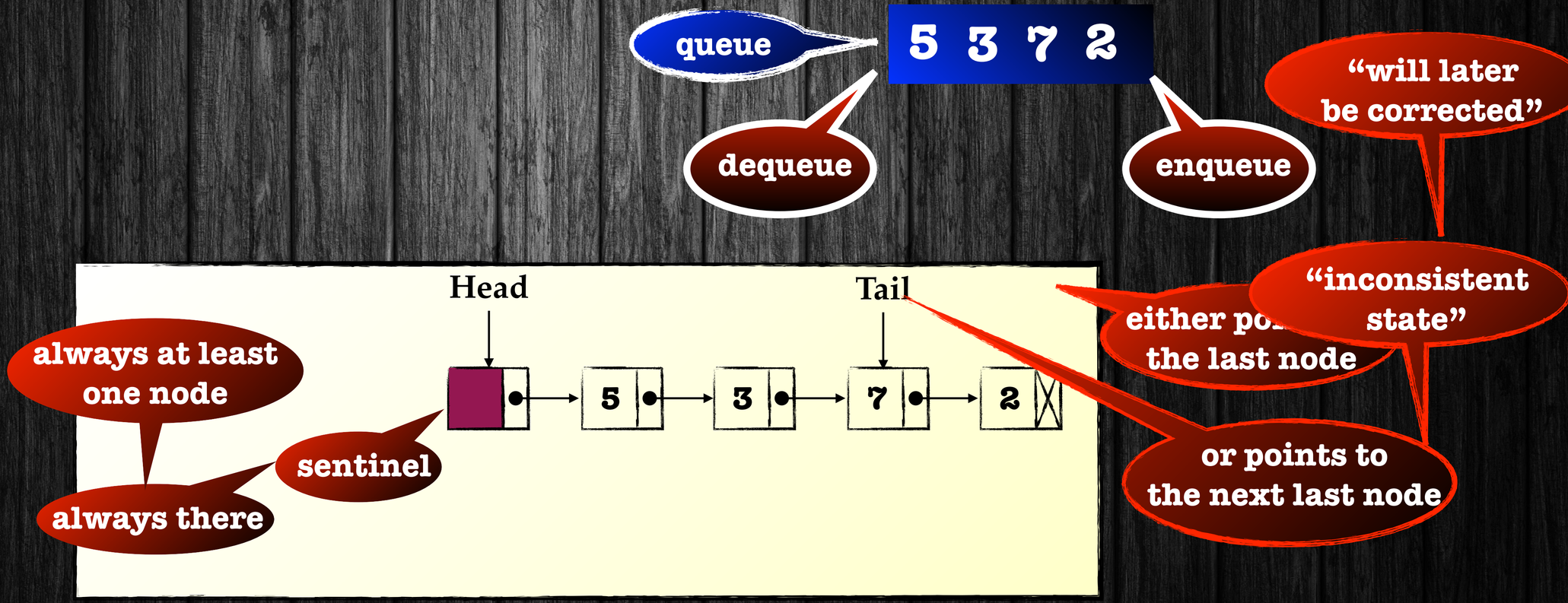
The Michael-Scott Queue Algorithm



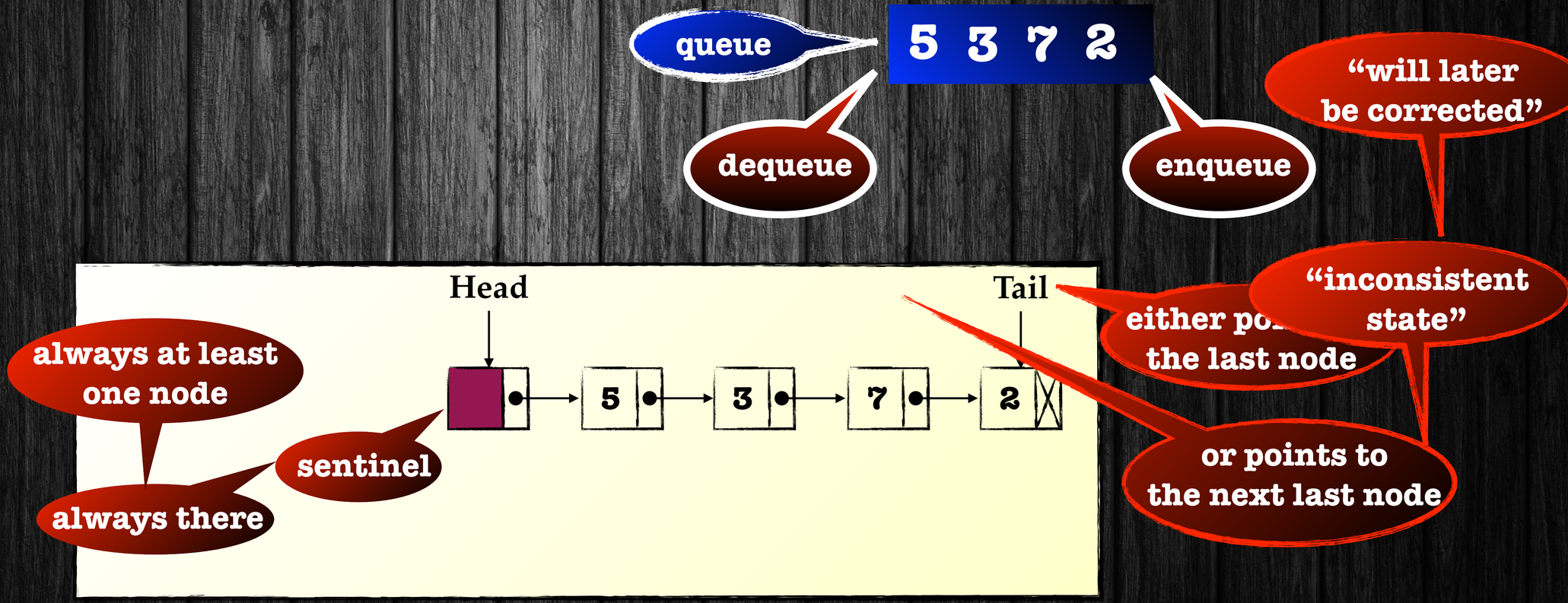
The Michael-Scott Queue Algorithm



The Michael-Scott Queue Algorithm



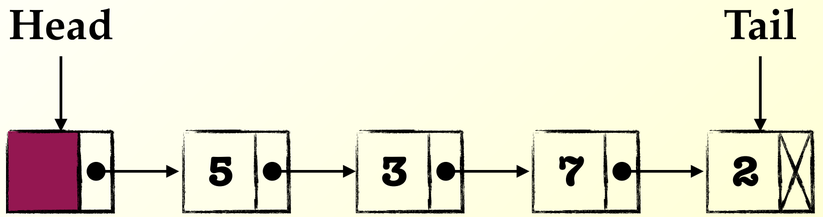
The Michael-Scott Queue Algorithm



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

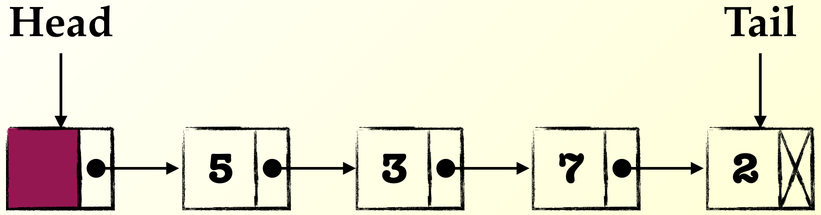
```
enq(k):
Node t,x
 1  n = new Node(k,NULL)
 2  while (true)
 3    t = Tail
 4    x = t.next
 5    if (t == Tail)
 6      if (x == NULL)
 7        if (CAS (t.next,NULL,n))
 8          CAS (Tail,t,n)
 9          exit
10      else
11        CAS (Tail,t,x)
```



enq(4)

queue

5 3 7 2



The Michael-Scott Queue Algorithm

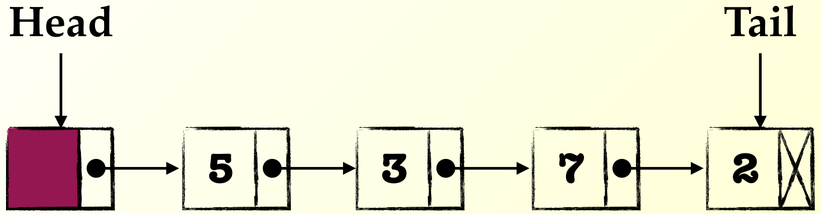
```
enq(k):
Node t,x
 1  n = new Node(k,NULL)
 2  while (true)
 3    t = Tail
 4    x = t.next
 5    if (t == Tail)
 6      if (x == NULL)
 7        if (CAS (t.next,NULL,n))
 8          CAS (Tail,t,n)
 9          exit
10      else
11        CAS (Tail,t,x)
```



enq(4)

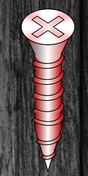
queue

5 3 7 2



The Michael-Scott Queue Algorithm

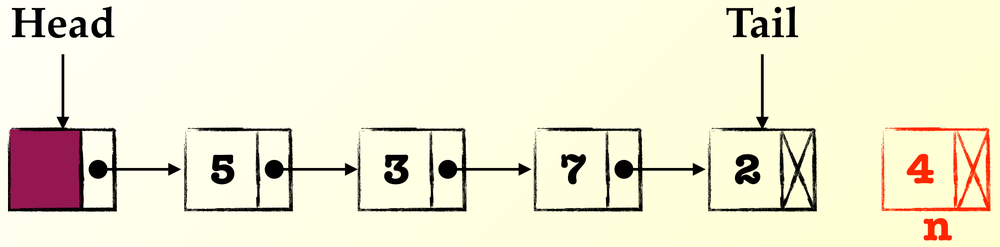
```
enq(k):
Node t,x
1  n = new Node(k,NULL)
2  while (true)
3    t = Tail
4    x = t.next
5    if (t == Tail)
6      if (x == NULL)
7        if (CAS (t.next,NULL,n))
8          CAS (Tail,t,n)
9          exit
10     else
11       CAS (Tail,t,x)
```



enq(4)

queue

5 3 7 2



The Michael-Scott Queue Algorithm

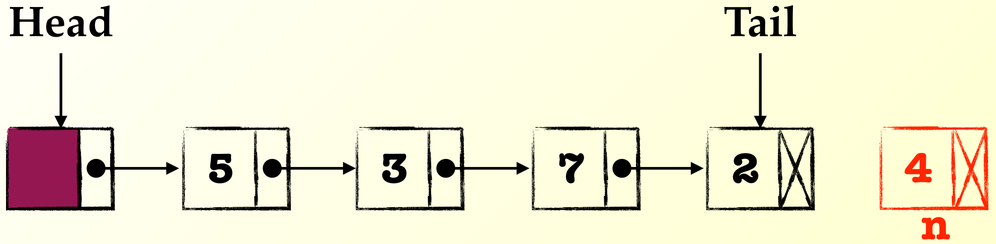
```
enq(k):
Node t,x
 1  n = new Node(k,NULL)
 2  while (true)
 3    t = Tail
 4    x = t.next
 5    if (t == Tail)
 6      if (x == NULL)
 7        if (CAS (t.next,NULL,n))
 8          CAS (Tail,t,n)
 9          exit
10      else
11        CAS (Tail,t,x)
```



enq(4)

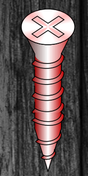
queue

5 3 7 2



The Michael-Scott Queue Algorithm

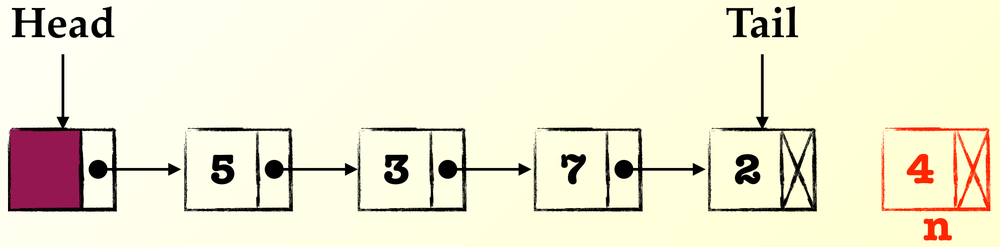
```
enq(k):
Node t,x
1  n = new Node(k,NULL)
2  while (true)
3    t = Tail
4    x = t.next
5    if (t == Tail)
6      if (x == NULL)
7        if (CAS (t.next,NULL,n))
8          CAS (Tail,t,n)
9          exit
10     else
11       CAS (Tail,t,x)
```



enq(4)

queue

5 3 7 2



The Michael-Scott Queue Algorithm

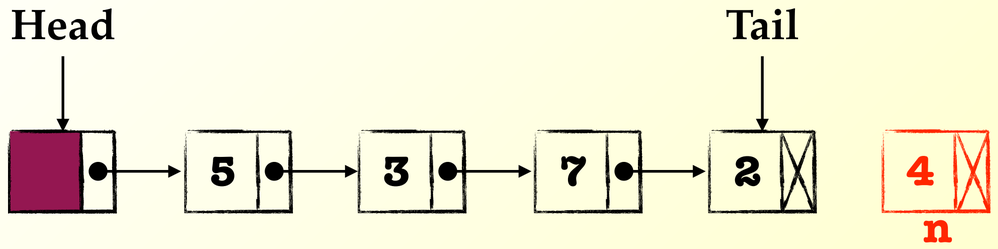
```
enq(k):
Node t,x
1  n = new Node(k,NULL)
2  while (true)
3    t = Tail
4    x = t.next
5    if (t == Tail)
6      if (x == NULL)
7        if (CAS (t.next,NULL,n))
8          CAS (Tail,t,n)
9          exit
10     else
11       CAS (Tail,t,x)
```

try to add the new node

enq(4)

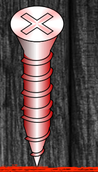
queue

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)
```



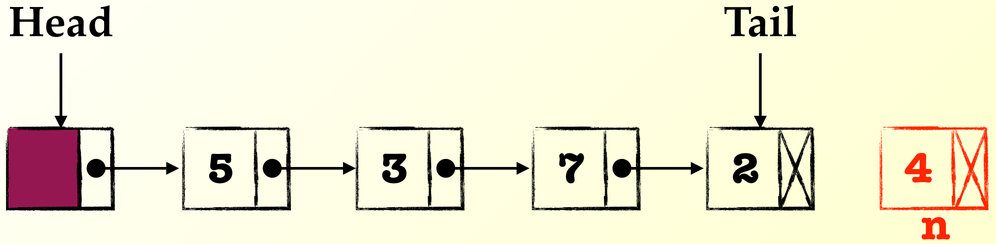
enq(4)

try to add the new node

you may repeatedly fail

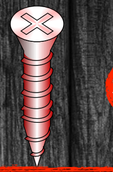
queue

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)

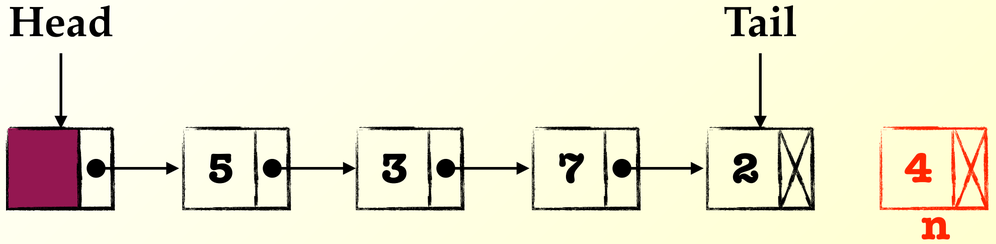
try to add the new node

you may repeatedly fail

"not wait-free"

queue

5 3 7 2



The Michael-Scott Queue Algorithm

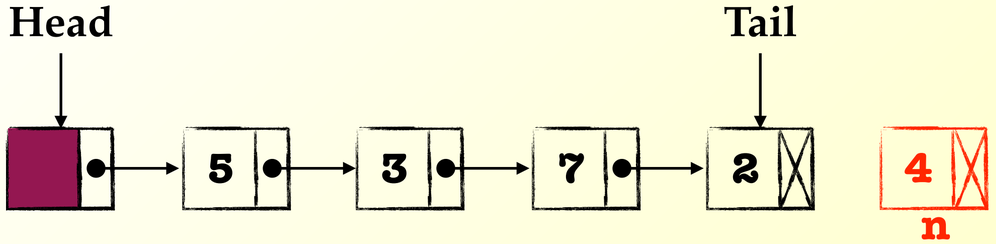
```
enq(k):
Node t,x
 1  n = new Node(k,NULL)
 2  while (true)
 3    t = Tail
 4    x = t.next
 5    if (t == Tail)
 6      if (x == NULL)
 7        if (CAS (t.next,NULL,n))
 8          CAS (Tail,t,n)
 9          exit
10      else
11        CAS (Tail,t,x)
```



enq(4)

queue

5 3 7 2



The Michael-Scott Queue Algorithm

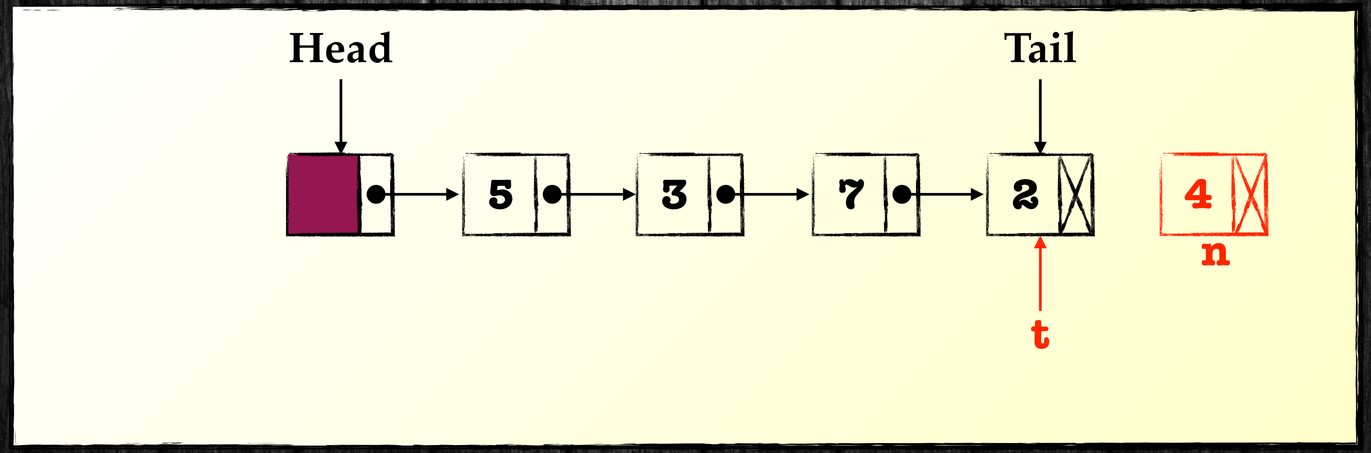
```
enq(k):
Node t,x
 1  n = new Node(k,NULL)
 2  while (true)
 3    t = Tail
 4    x = t.next
 5    if (t == Tail)
 6      if (x == NULL)
 7        if (CAS (t.next,NULL,n))
 8          CAS (Tail,t,n)
 9          exit
10      else
11        CAS (Tail,t,x)
```



enq(4)

queue

5 3 7 2



The Michael-Scott Queue Algorithm

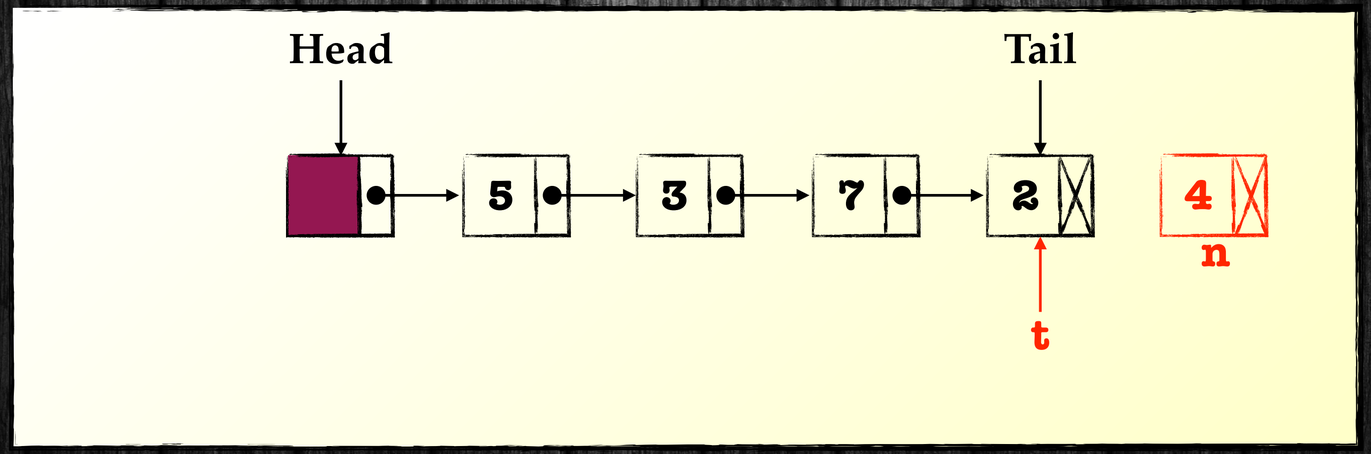
```
enq(k):
Node t,x
 1  n = new Node(k,NULL)
 2  while (true)
 3    t = Tail
 4    x = t.next
 5    if (t == Tail)
 6      if (x == NULL)
 7        if (CAS (t.next,NULL,n))
 8          CAS (Tail,t,n)
 9          exit
10      else
11        CAS (Tail,t,x)
```



enq(4)

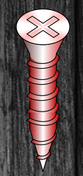
queue

5 3 7 2



The Michael-Scott Queue Algorithm

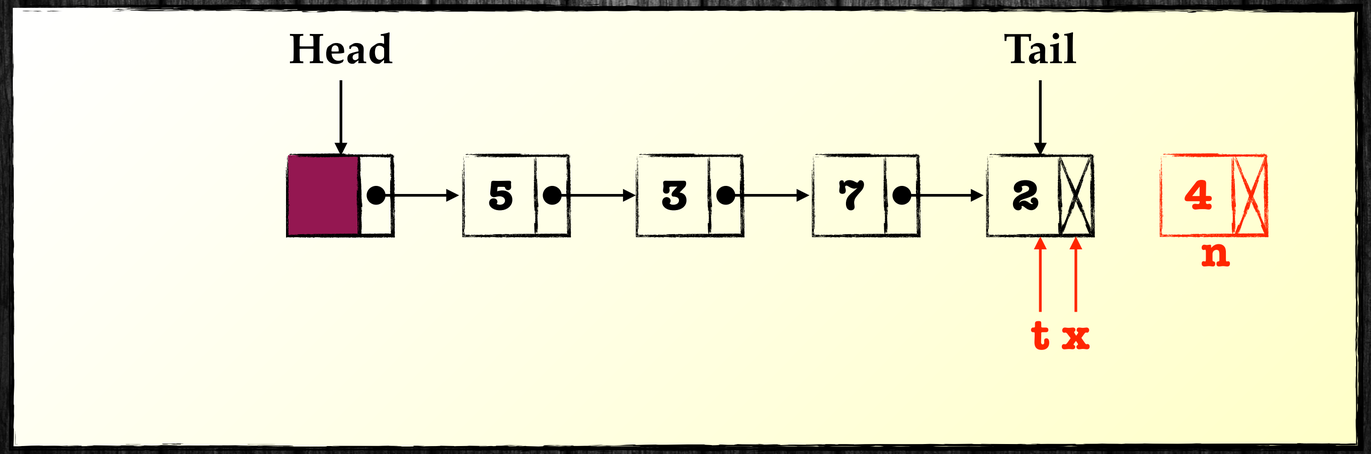
```
enq(k):
Node t,x
 1  n = new Node(k,NULL)
 2  while (true)
 3    t = Tail
 4    x = t.next
 5    if (t == Tail)
 6      if (x == NULL)
 7        if (CAS (t.next,NULL,n))
 8          CAS (Tail,t,n)
 9          exit
10      else
11        CAS (Tail,t,x)
```



enq(4)

queue

5 3 7 2



The Michael-Scott Queue Algorithm

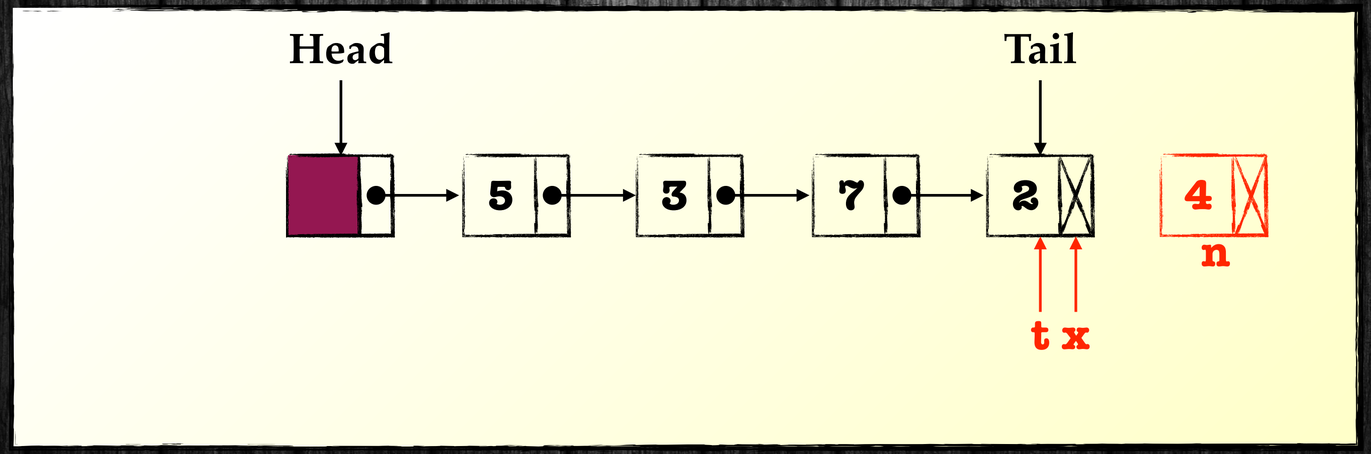
```
enq(k):
Node t,x
 1  n = new Node(k,NULL)
 2  while (true)
 3    t = Tail
 4    x = t.next
 5    if (t == Tail)
 6      if (x == NULL)
 7        if (CAS (t.next,NULL,n))
 8          CAS (Tail,t,n)
 9          exit
10      else
11        CAS (Tail,t,x)
```



enq(4)

queue

5 3 7 2



The Michael-Scott Queue Algorithm

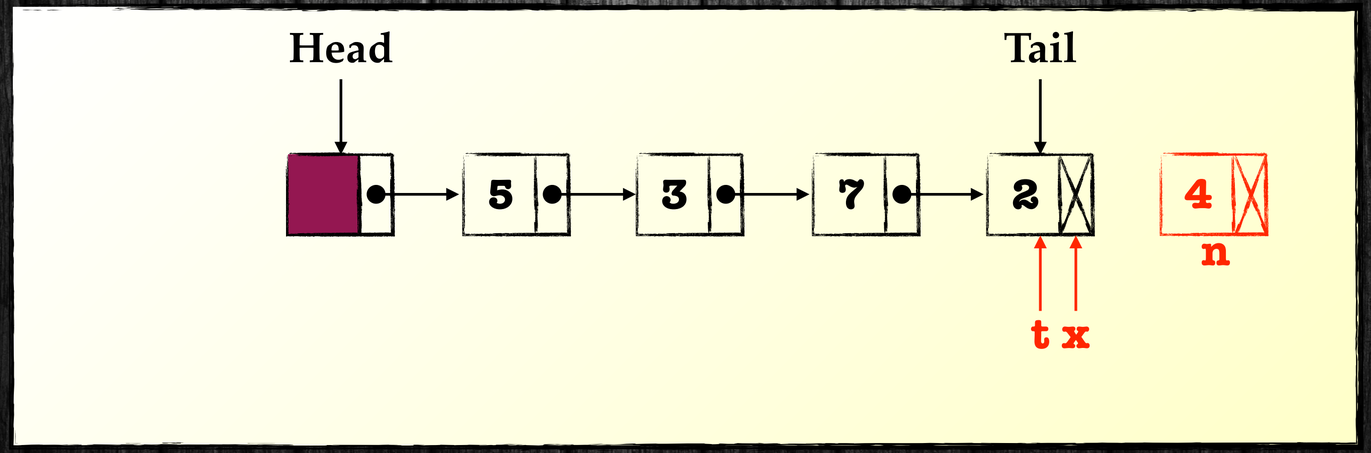
```
enq(k):
Node t,x
 1  n = new Node(k,NULL)
 2  while (true)
 3    t = Tail
 4    x = t.next
 5    if (t == Tail)
 6      if (x == NULL)
 7        if (CAS (t.next,NULL,n))
 8          CAS (Tail,t,n)
 9          exit
10      else
11        CAS (Tail,t,x)
```



enq(4)

queue

5 3 7 2



The Michael-Scott Queue Algorithm

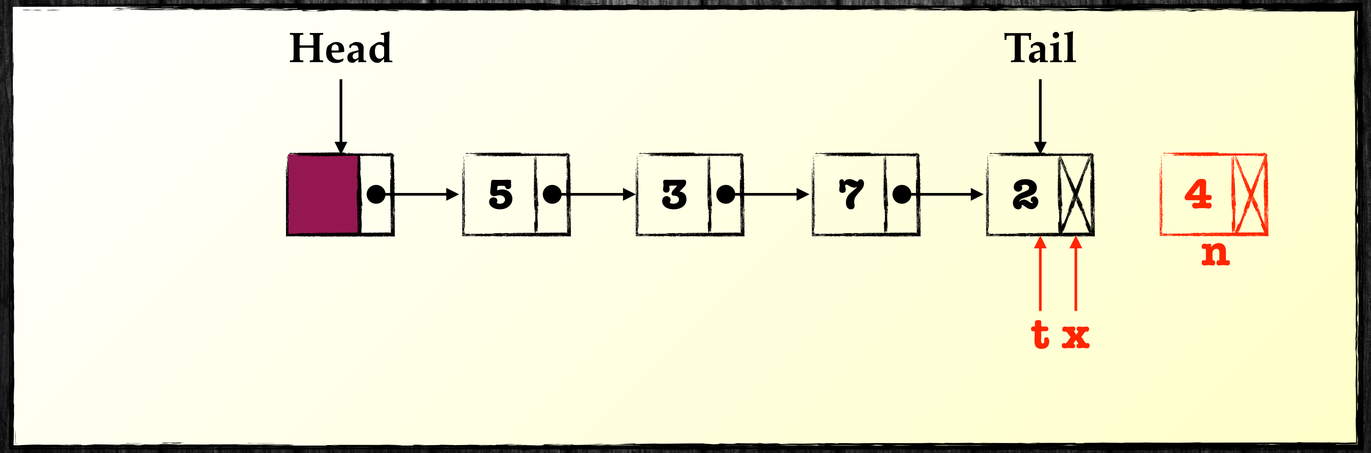
```
enq(k):
Node t,x
1  n = new Node(k,NULL)
2  while (true)
3    t = Tail
4    x = t.next
5    if (t == Tail)
6      if (x == NULL)
7        if (CAS (t.next,NULL,n)
8          CAS (Tail,t,n)
9          exit
10     else
11       CAS (Tail,t,x)
```



enq(4)

"is Tail really pointing to the last node?"

queue 5 3 7 2



The Michael-Scott Queue Algorithm

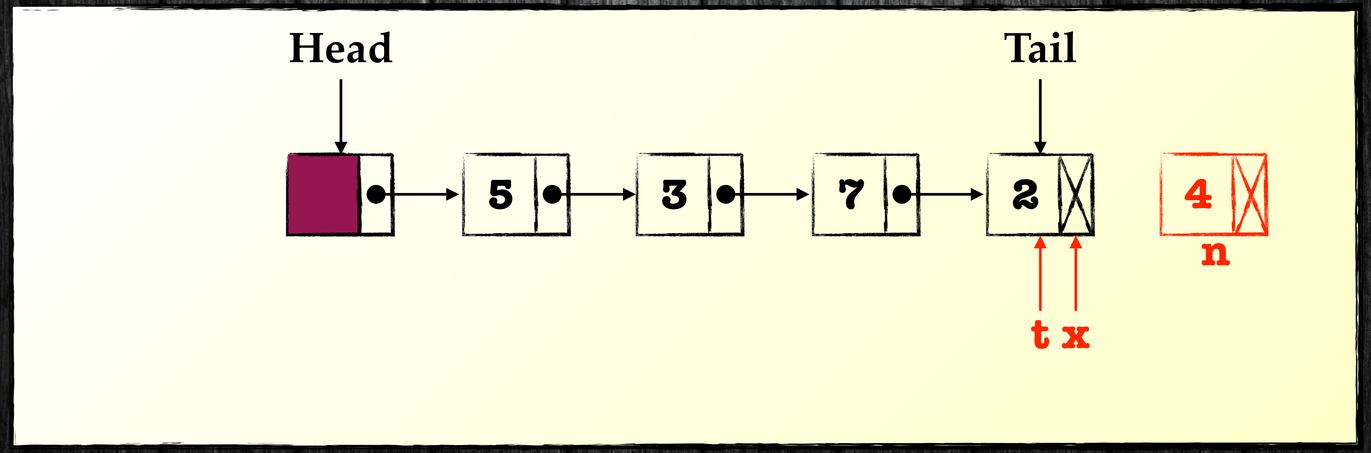
```
enq(k):
Node t,x
 1  n = new Node(k,NULL)
 2  while (true)
 3    t = Tail
 4    x = t.next
 5    if (t == Tail)
 6      if (x == NULL)
 7        if (CAS (t.next,NULL,n))
 8          CAS (Tail,t,n)
 9          exit
10      else
11        CAS (Tail,t,x)
```



enq(4)

queue

5 3 7 2



The Michael-Scott Queue Algorithm

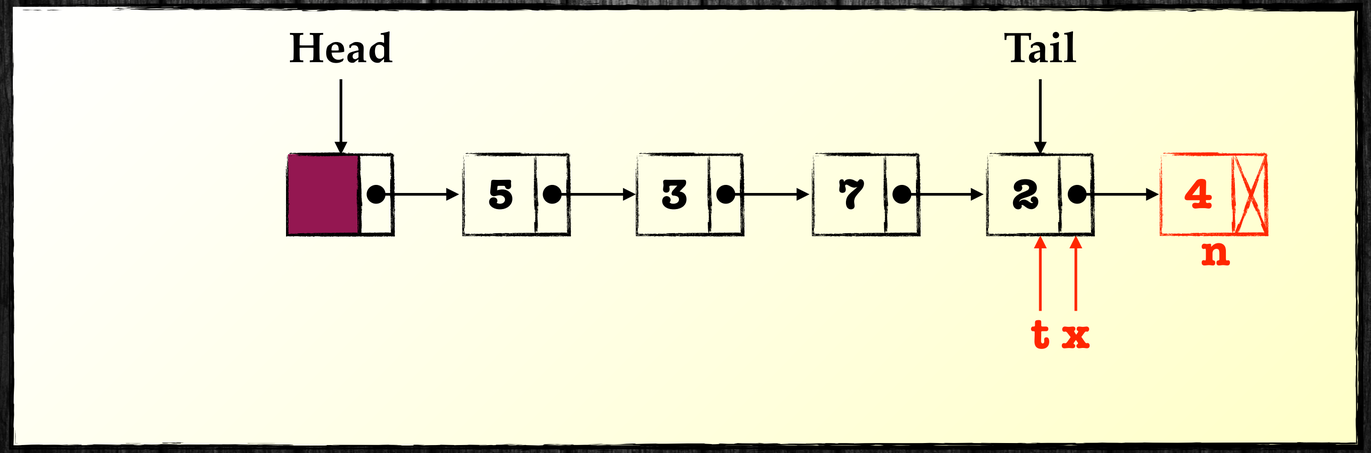
```
enq(k):
Node t,x
 1  n = new Node(k,NULL)
 2  while (true)
 3    t = Tail
 4    x = t.next
 5    if (t == Tail)
 6      if (x == NULL)
 7        if (CAS (t.next,NULL,n))
 8          CAS (Tail,t,n)
 9          exit
10      else
11        CAS (Tail,t,x)
```



enq(4)

queue

5 3 7 2



The Michael-Scott Queue Algorithm

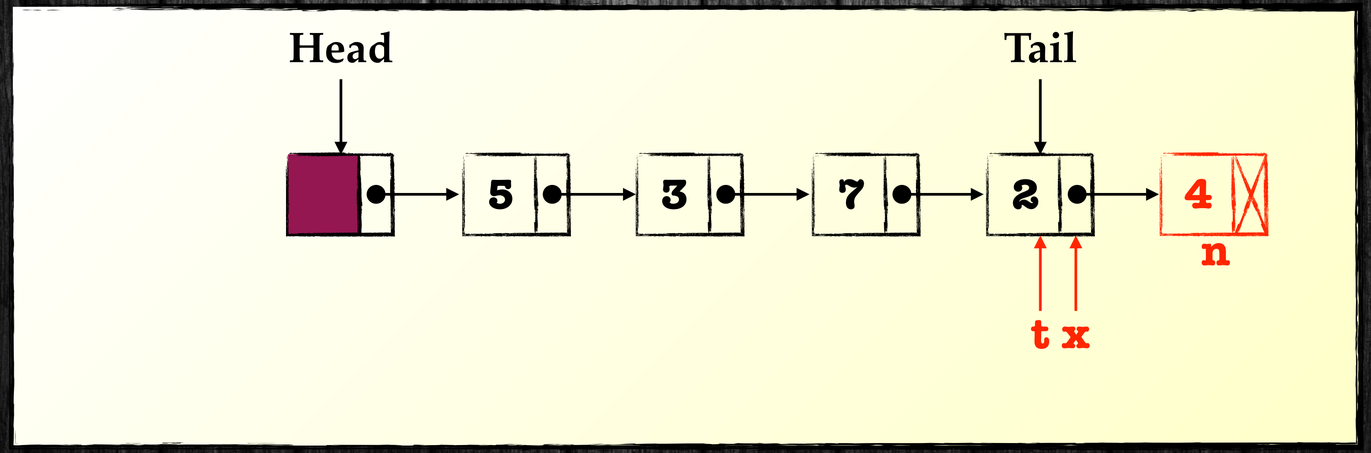
```
enq(k):
Node t,x
 1  n = new Node(k,NULL)
 2  while (true)
 3    t = Tail
 4    x = t.next
 5    if (t == Tail)
 6      if (x == NULL)
 7        if (CAS (t.next,NULL,n))
 8          CAS (Tail,t,n)
 9          exit
10      else
11        CAS (Tail,t,x)
```



enq(4)

queue

5 3 7 2



The Michael-Scott Queue Algorithm

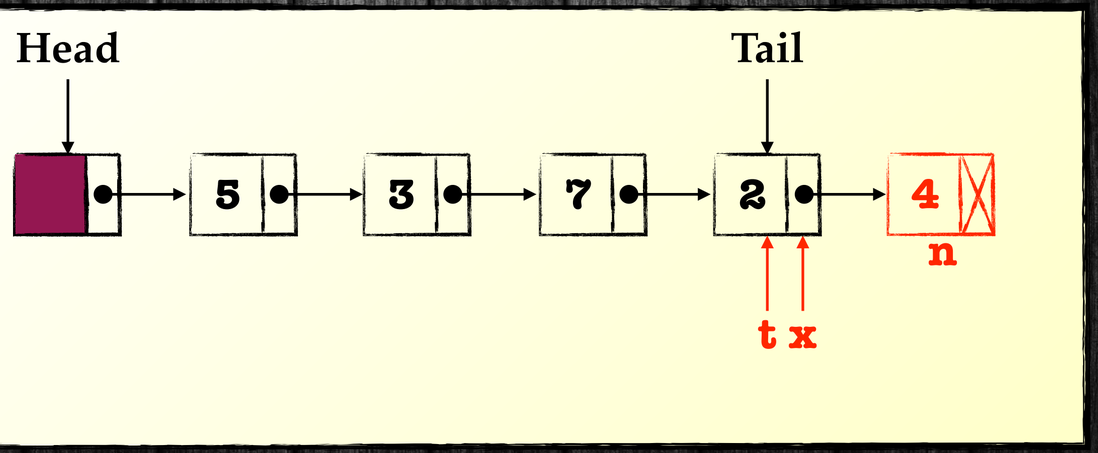
```
enq(k):
Node t,x
1  n = new Node(k,NULL)
2  while (true)
3    t = Tail
4    x = t.next
5    if (t == Tail)
6      if (x == NULL)
7        if (CAS (t.next,NULL,n))
8          CAS (Tail,t,n)
9          exit
10     else
11       CAS (Tail,t,x)
```



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)
```

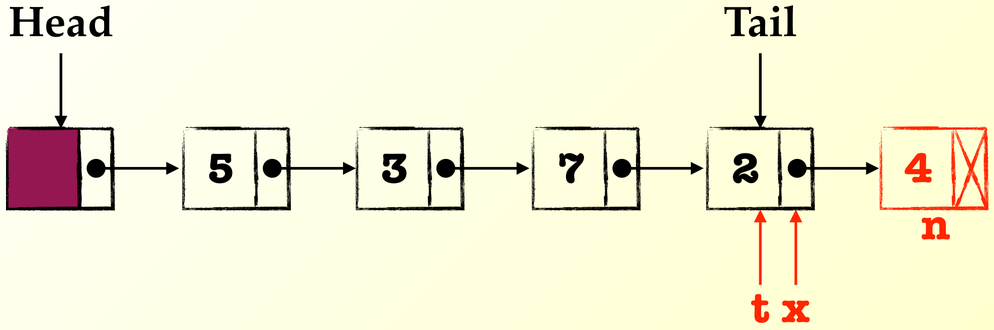


enq(4)

"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)
```



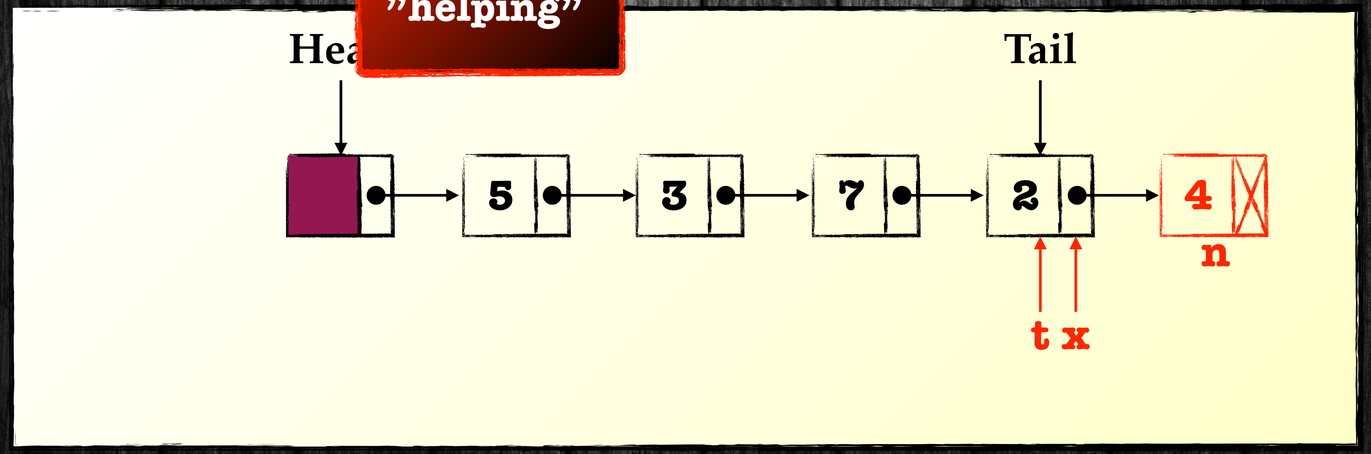
enq(4)

"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)

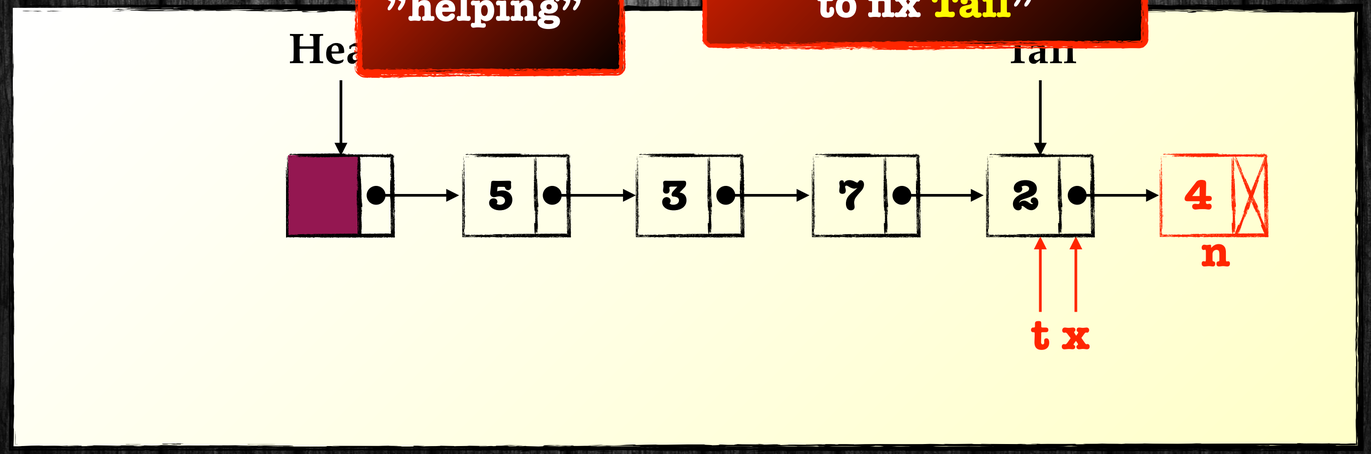
"Fix Tail if needed"

"Fix Tail if needed"

5 3 7 2

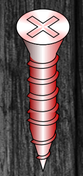
"helping"

"help another thread to fix Tail"



The Michael-Scott Queue Algorithm

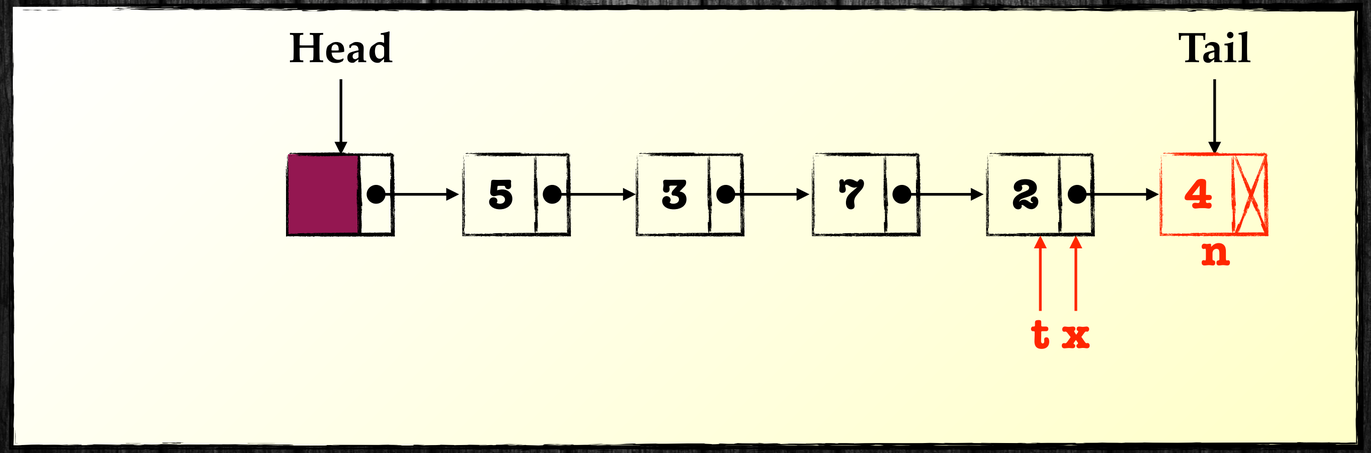
```
enq(k):
Node t,x
 1  n = new Node(k,NULL)
 2  while (true)
 3    t = Tail
 4    x = t.next
 5    if (t == Tail)
 6      if (x == NULL)
 7        if (CAS (t.next,NULL,n))
 8          CAS (Tail,t,n)
 9          exit
10      else
11        CAS (Tail,t,x)
```



enq(4)

queue

5 3 7 2



The Michael-Scott Queue Algorithm

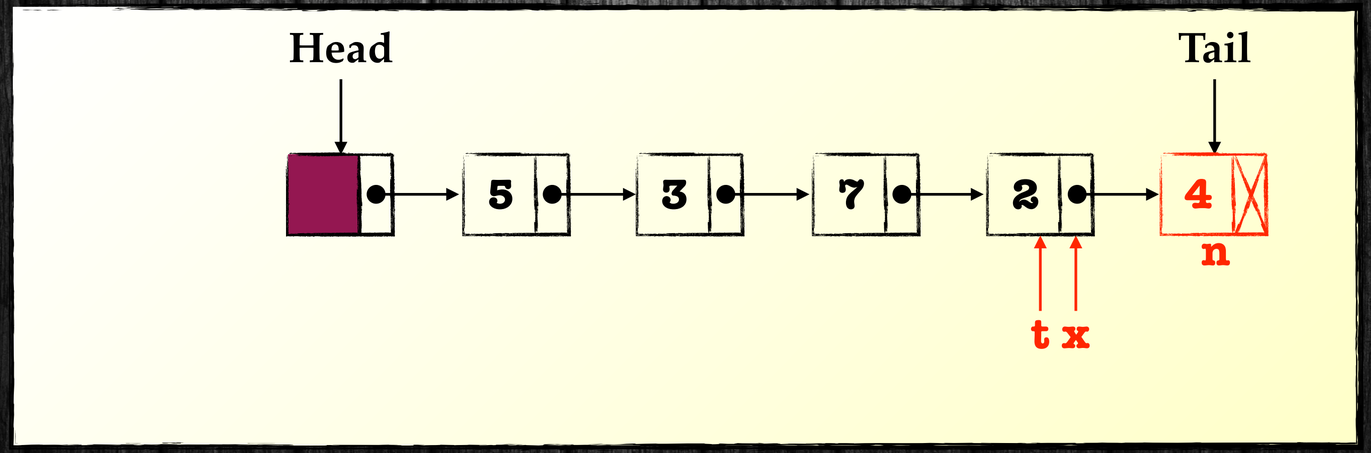
```
enq(k):
Node t,x
 1  n = new Node(k,NULL)
 2  while (true)
 3    t = Tail
 4    x = t.next
 5    if (t == Tail)
 6      if (x == NULL)
 7        if (CAS (t.next,NULL,n))
 8          CAS (Tail,t,n)
 9          exit
10      else
11        CAS (Tail,t,x)
```



enq(4)

queue

5 3 7 2



The Michael-Scott Queue Algorithm

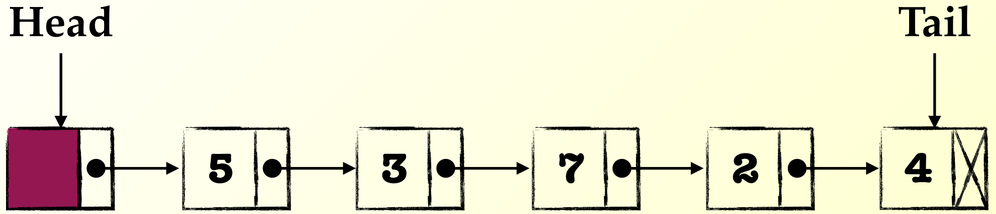
```
enq(k):
Node t,x
 1  n = new Node(k,NULL)
 2  while (true)
 3    t = Tail
 4    x = t.next
 5    if (t == Tail)
 6      if (x == NULL)
 7        if (CAS (t.next,NULL,n))
 8          CAS (Tail,t,n)
 9          exit
10      else
11        CAS (Tail,t,x)
```



enq(4)

queue

5 3 7 1



The Michael-Scott Queue Algorithm

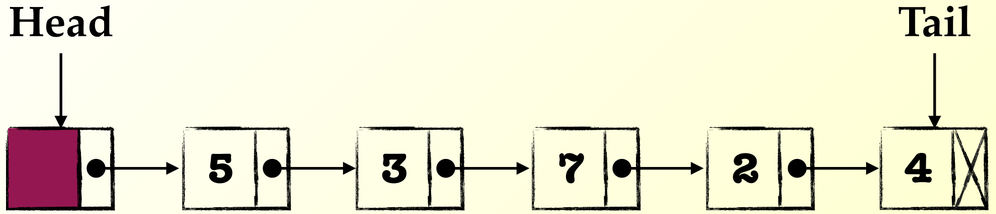
```
enq(k):
Node t,x
 1  n = new Node(k,NULL)
 2  while (true)
 3    t = Tail
 4    x = t.next
 5    if (t == Tail)
 6      if (x == NULL)
 7        if (CAS (t.next,NULL,n))
 8          CAS (Tail,t,n)
 9          exit
10      else
11        CAS (Tail,t,x)
```



enq(4)

queue

5 3 7 2 4

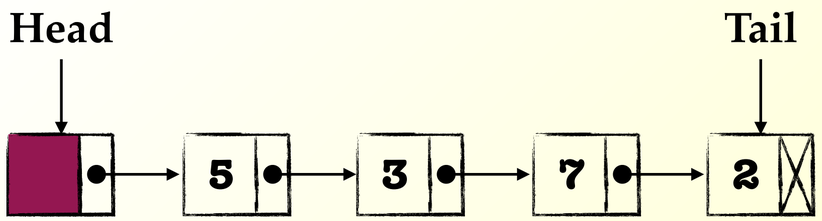


The Michael-Scott Queue Algorithm

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)
```

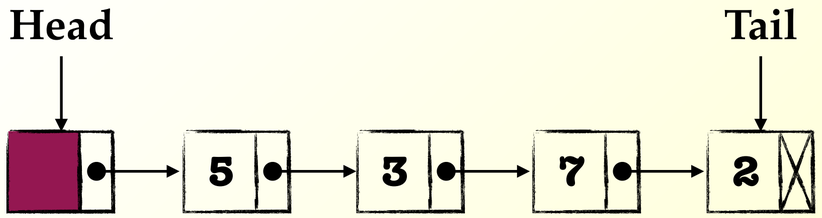
The Michael-Scott Queue Algorithm

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)
```

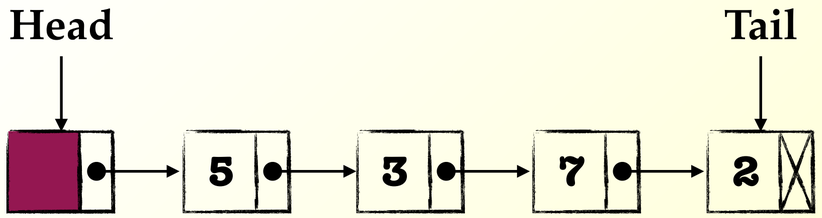
The Michael-Scott Queue Algorithm

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)
```



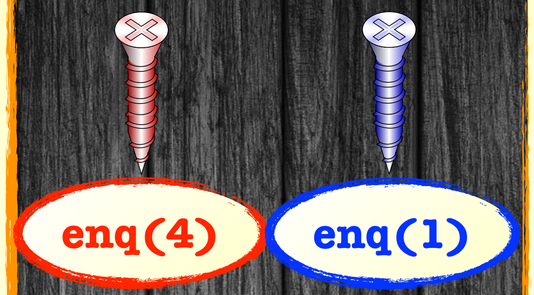

```
enq(k):
Node t,x
1  n = new Node(k,NULL)
2  while (true)
3    t = Tail
4    x = t.next
5    if (t == Tail)
6      if (x == NULL)
7        if (CAS (t.next,NULL,n))
8          CAS (Tail,t,n)
9          exit
10     else
11       CAS (Tail,t,x)
```



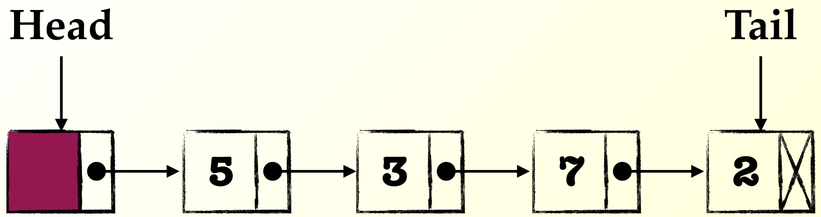
The Michael-Scott Queue Algorithm

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)
```



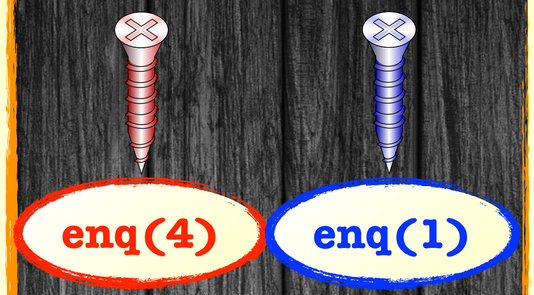

```
enq(k):
Node t,x
1  n = new Node(k,NULL)
2  while (true)
3    t = Tail
4    x = t.next
5    if (t == Tail)
6      if (x == NULL)
7        if (CAS (t.next,NULL,n))
8          CAS (Tail,t,n)
9          exit
10     else
11       CAS (Tail,t,x)
```



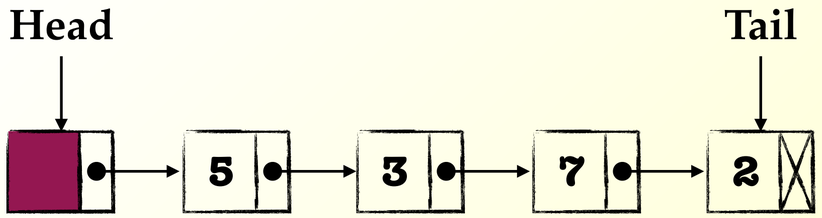
The Michael-Scott Queue Algorithm

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)
```



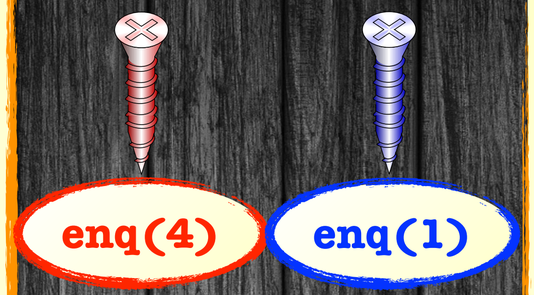

```
enq(k):
Node t,x
1  n = new Node(k,NULL)
2  while (true)
3    t = Tail
4    x = t.next
5    if (t == Tail)
6      if (x == NULL)
7        if (CAS (t.next,NULL,n))
8          CAS (Tail,t,n)
9          exit
10     else
11       CAS (Tail,t,x)
```



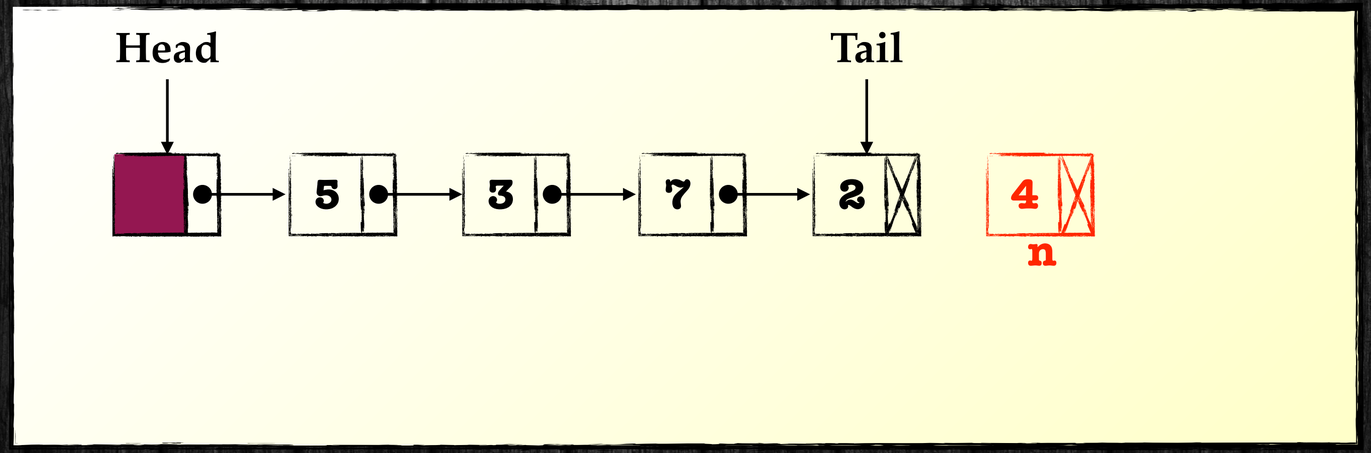
The Michael-Scott Queue Algorithm

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)
```



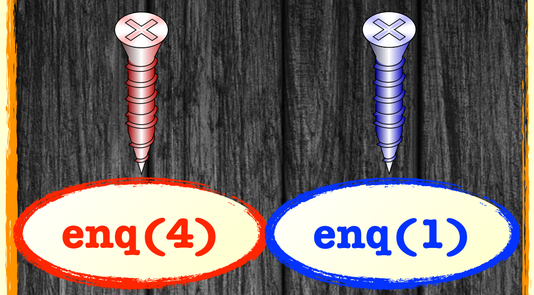

```
enq(k):
Node t,x
1  n = new Node(k,NULL)
2  while (true)
3    t = Tail
4    x = t.next
5    if (t == Tail)
6      if (x == NULL)
7        if (CAS (t.next,NULL,n))
8          CAS (Tail,t,n)
9          exit
10     else
11       CAS (Tail,t,x)
```

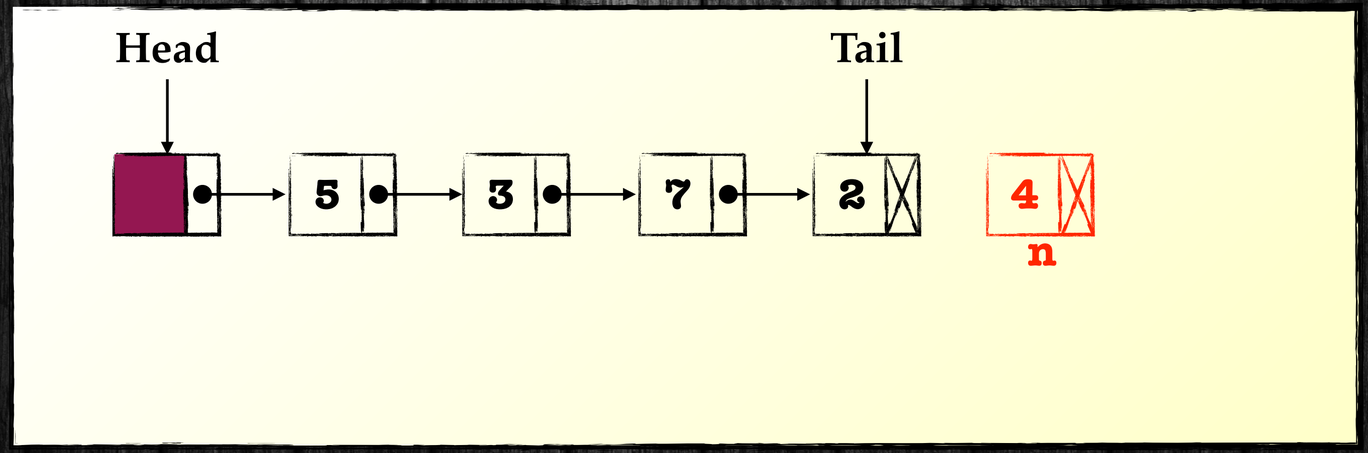
The Michael-Scott Queue Algorithm

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)
```



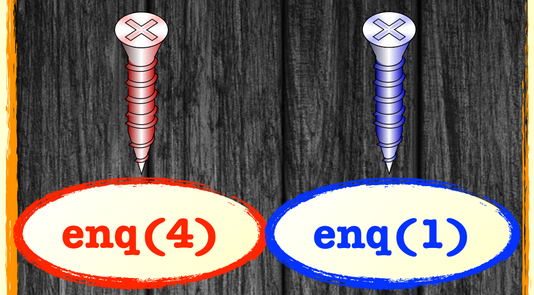

```
enq(k):
Node t,x
1  n = new Node(k,NULL)
2  while (true)
3    t = Tail
4    x = t.next
5    if (t == Tail)
6      if (x == NULL)
7        if (CAS (t.next,NULL,n))
8          CAS (Tail,t,n)
9          exit
10     else
11       CAS (Tail,t,x)
```



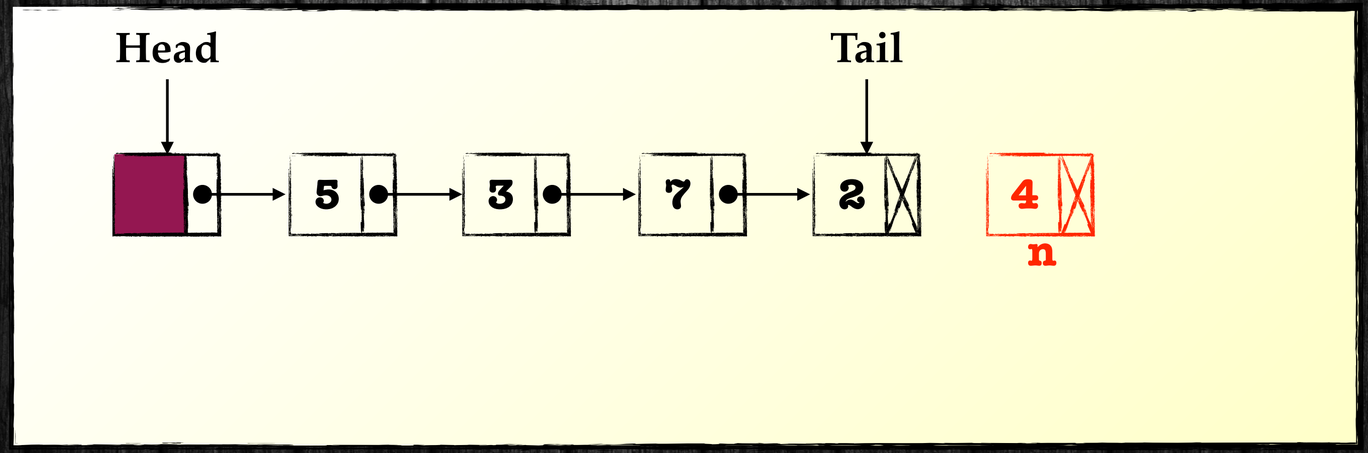
The Michael-Scott Queue Algorithm

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)
```



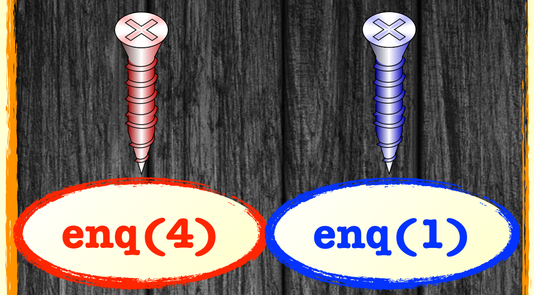

```
enq(k):
Node t,x
1  n = new Node(k,NULL)
2  while (true)
3    t = Tail
4    x = t.next
5    if (t == Tail)
6      if (x == NULL)
7        if (CAS (t.next,NULL,n))
8          CAS (Tail,t,n)
9          exit
10     else
11       CAS (Tail,t,x)
```



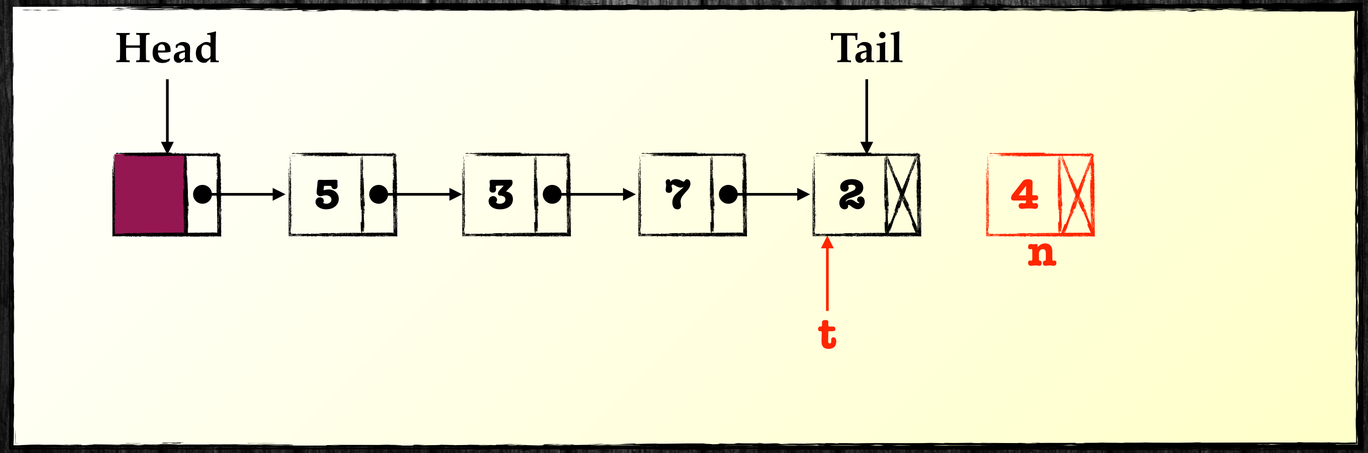
The Michael-Scott Queue Algorithm

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)
```



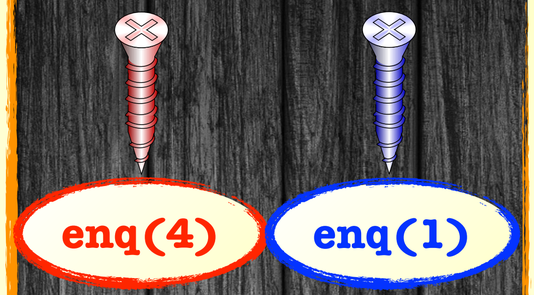

```
enq(k):
Node t,x
1  n = new Node(k,NULL)
2  while (true)
3    t = Tail
4    x = t.next
5    if (t == Tail)
6      if (x == NULL)
7        if (CAS (t.next,NULL,n))
8          CAS (Tail,t,n)
9          exit
10     else
11       CAS (Tail,t,x)
```



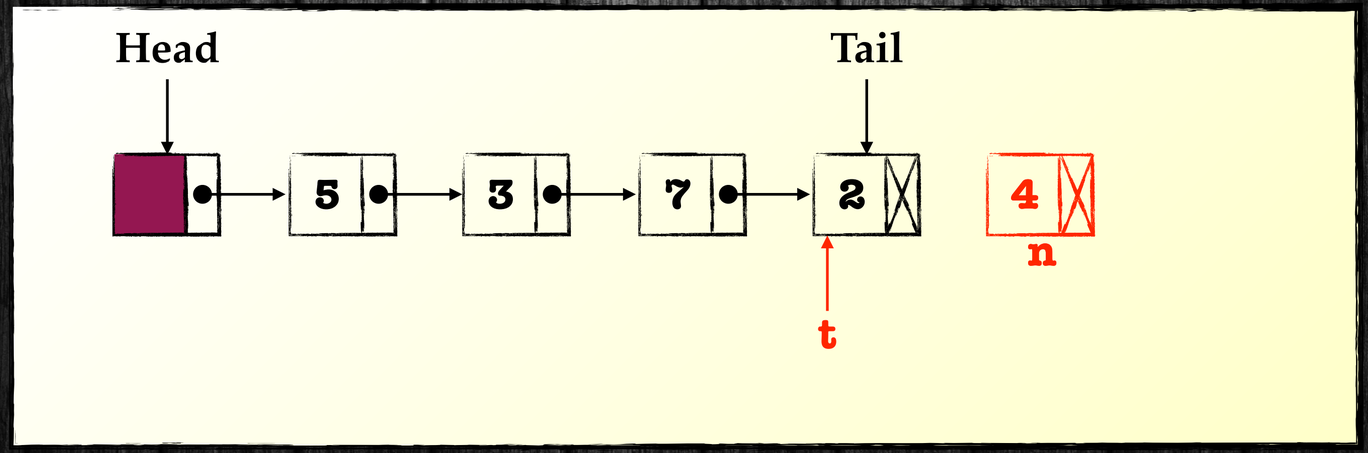
The Michael-Scott Queue Algorithm

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)
```



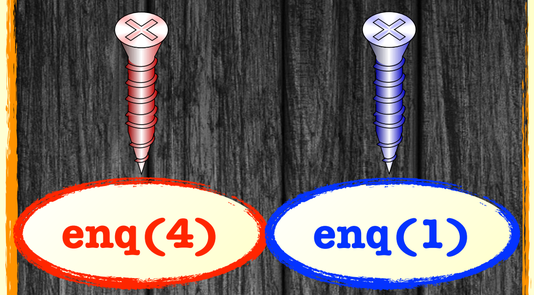

```
enq(k):
Node t,x
1  n = new Node(k,NULL)
2  while (true)
3    t = Tail
4    x = t.next
5    if (t == Tail)
6      if (x == NULL)
7        if (CAS (t.next,NULL,n))
8          CAS (Tail,t,n)
9          exit
10     else
11       CAS (Tail,t,x)
```



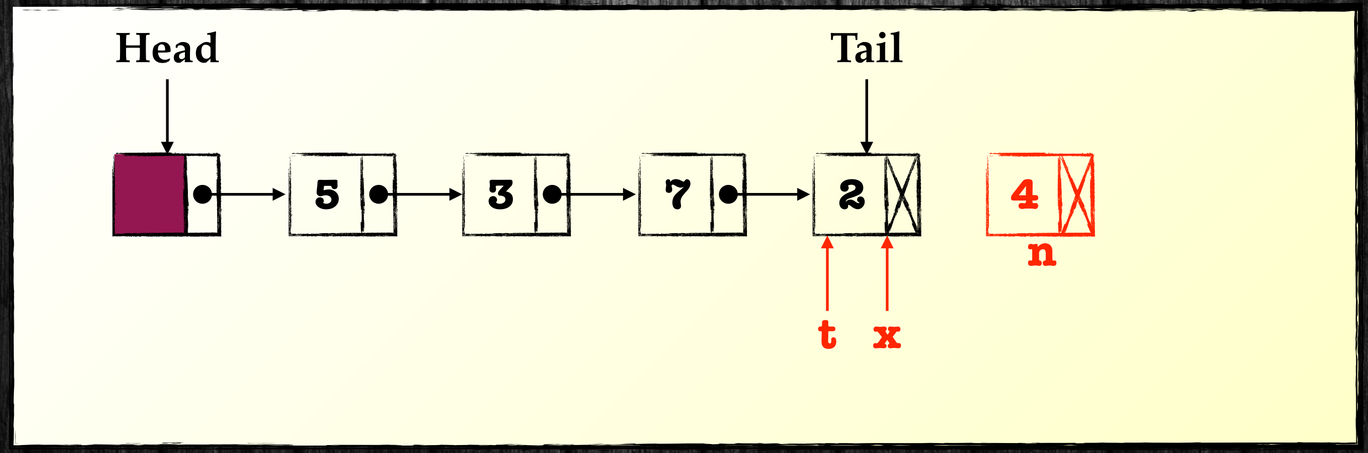
The Michael-Scott Queue Algorithm

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)
```



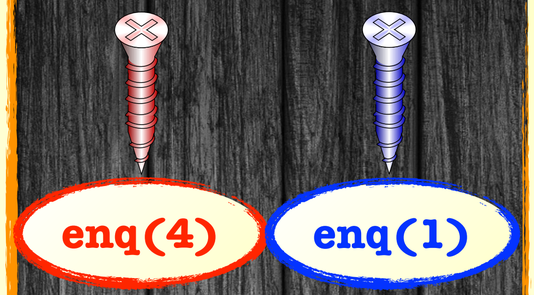

```
enq(k):
Node t,x
1  n = new Node(k,NULL)
2  while (true)
3    t = Tail
4    x = t.next
5    if (t == Tail)
6      if (x == NULL)
7        if (CAS (t.next,NULL,n))
8          CAS (Tail,t,n)
9          exit
10     else
11       CAS (Tail,t,x)
```



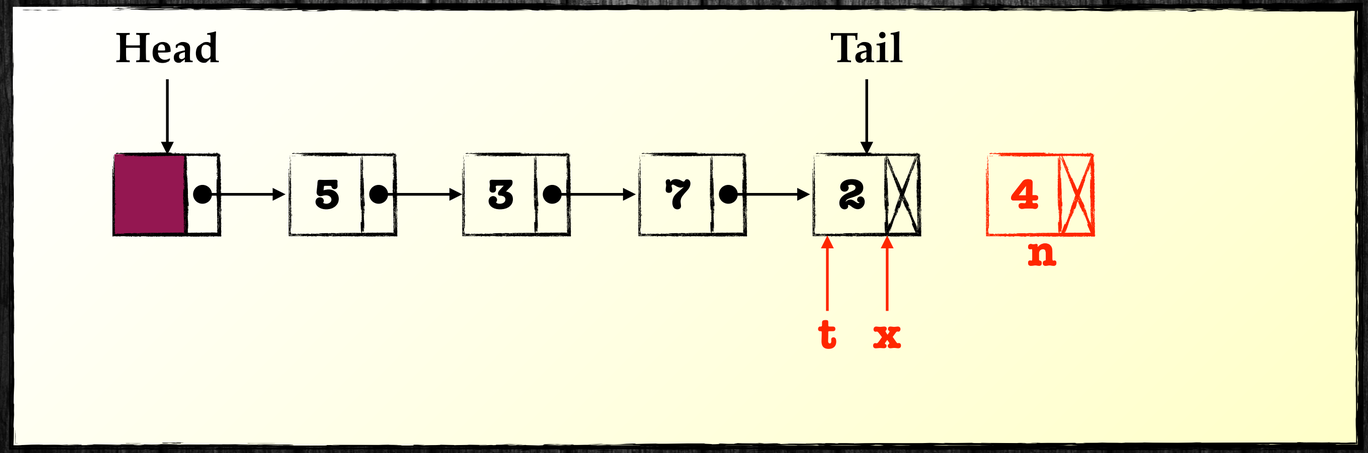
The Michael-Scott Queue Algorithm

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)
```



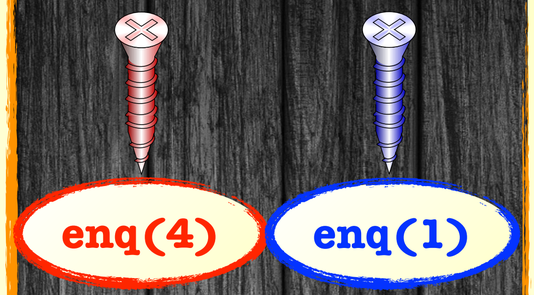

```
enq(k):
Node t,x
1  n = new Node(k,NULL)
2  while (true)
3    t = Tail
4    x = t.next
5    if (t == Tail)
6      if (x == NULL)
7        if (CAS (t.next,NULL,n))
8          CAS (Tail,t,n)
9          exit
10     else
11       CAS (Tail,t,x)
```



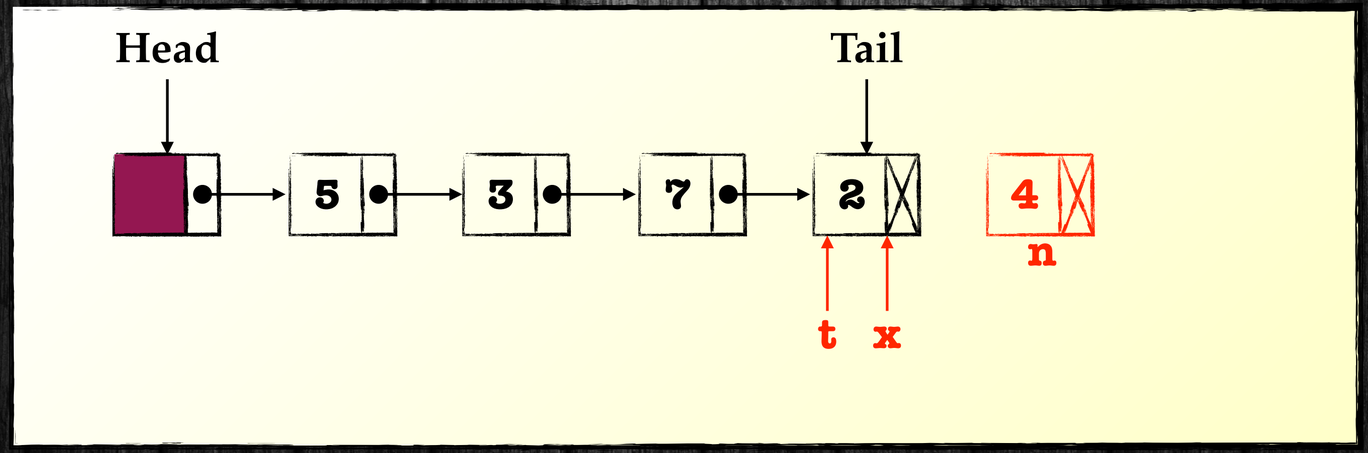
The Michael-Scott Queue Algorithm

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)
```



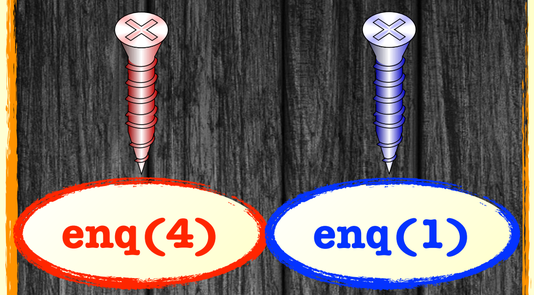

```
enq(k):
Node t,x
1  n = new Node(k,NULL)
2  while (true)
3    t = Tail
4    x = t.next
5    if (t == Tail)
6      if (x == NULL)
7        if (CAS (t.next,NULL,n))
8          CAS (Tail,t,n)
9          exit
10     else
11       CAS (Tail,t,x)
```



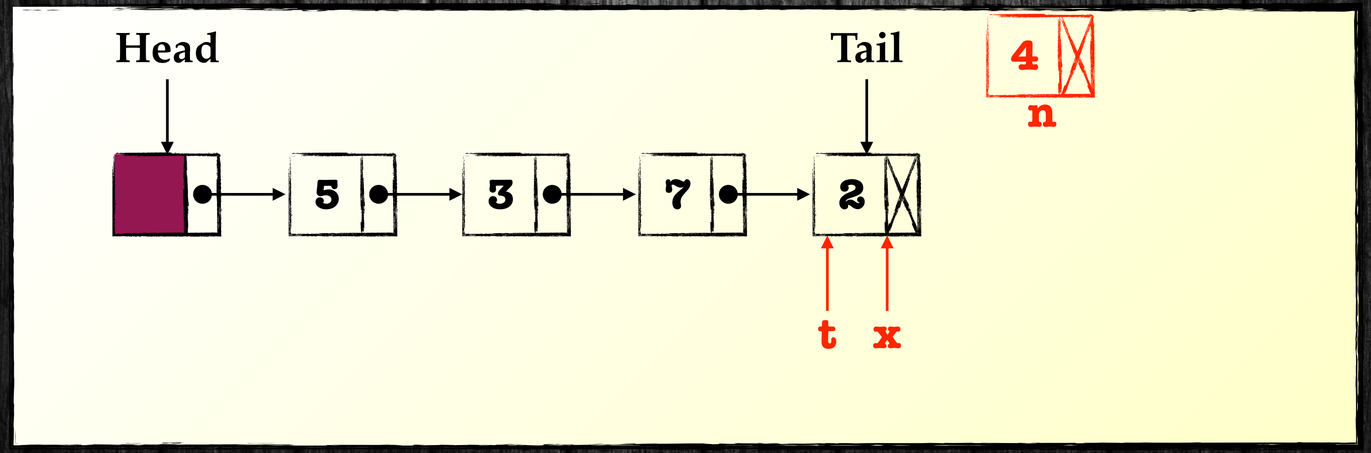
The Michael-Scott Queue Algorithm

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)
```



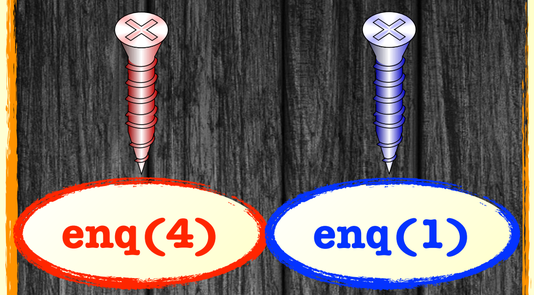

```
enq(k):
Node t,x
1  n = new Node(k,NULL)
2  while (true)
3    t = Tail
4    x = t.next
5    if (t == Tail)
6      if (x == NULL)
7        if (CAS (t.next,NULL,n))
8          CAS (Tail,t,n)
9          exit
10     else
11       CAS (Tail,t,x)
```



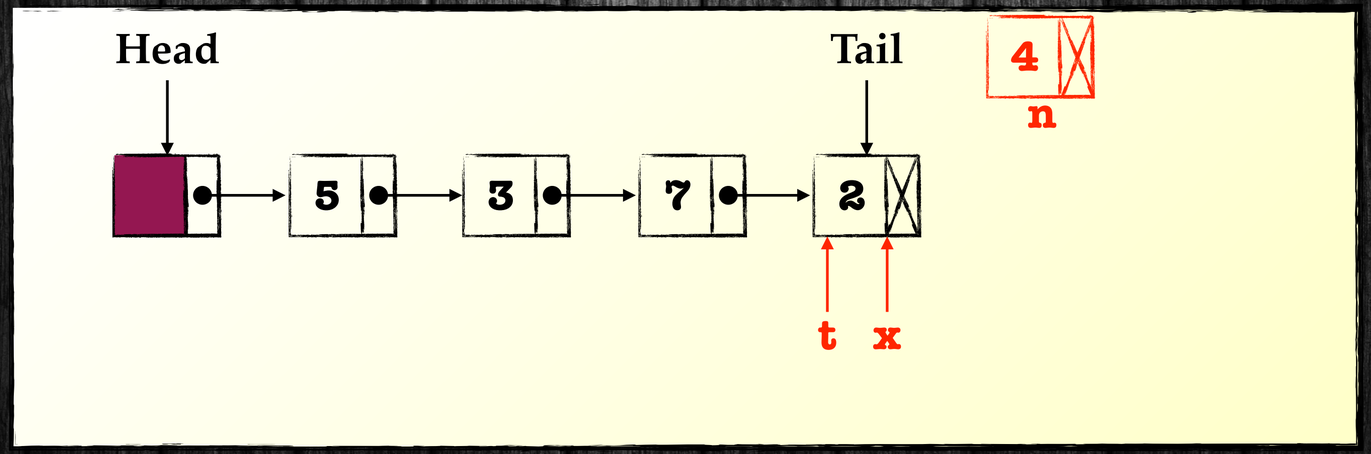
The Michael-Scott Queue Algorithm

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)
```



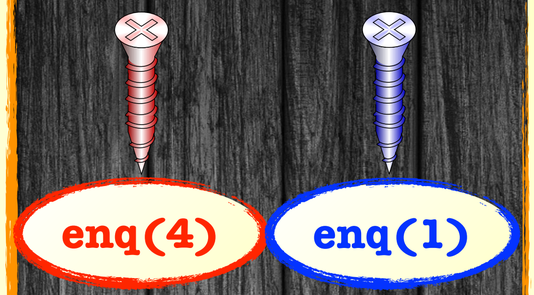

```
enq(k):
Node t,x
1  n = new Node(k,NULL)
2  while (true)
3    t = Tail
4    x = t.next
5    if (t == Tail)
6      if (x == NULL)
7        if (CAS (t.next,NULL,n))
8          CAS (Tail,t,n)
9          exit
10     else
11       CAS (Tail,t,x)
```



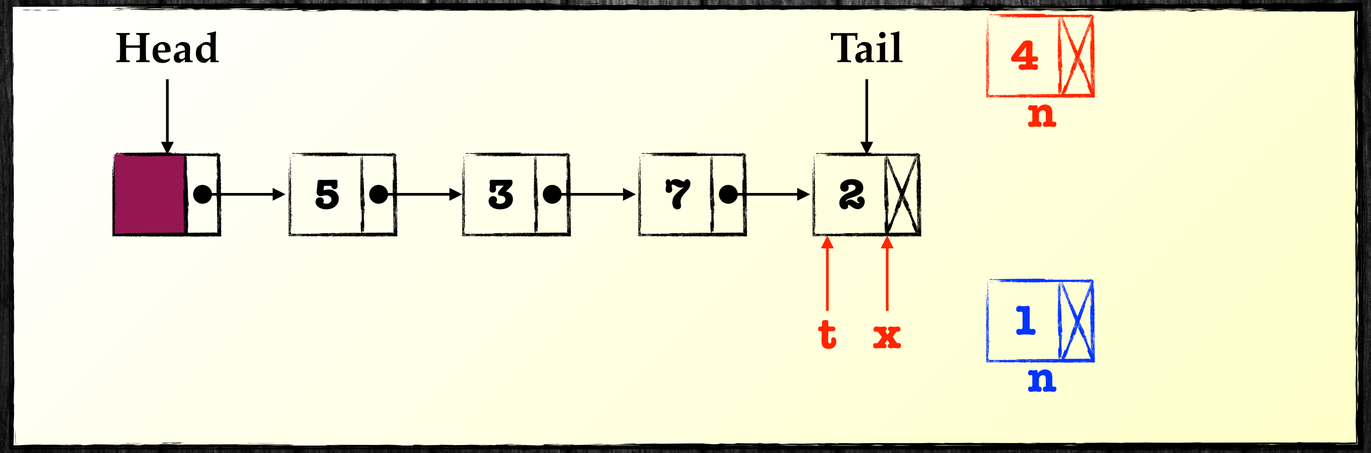
The Michael-Scott Queue Algorithm

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)
```



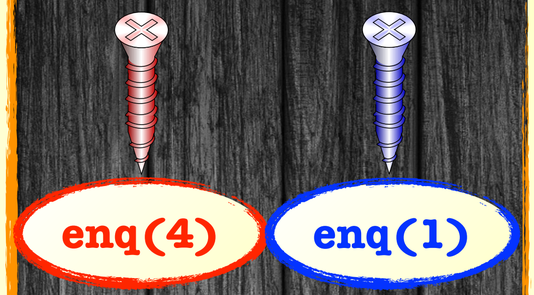

```
enq(k):
Node t,x
1  n = new Node(k,NULL)
2  while (true)
3    t = Tail
4    x = t.next
5    if (t == Tail)
6      if (x == NULL)
7        if (CAS (t.next,NULL,n))
8          CAS (Tail,t,n)
9          exit
10     else
11       CAS (Tail,t,x)
```



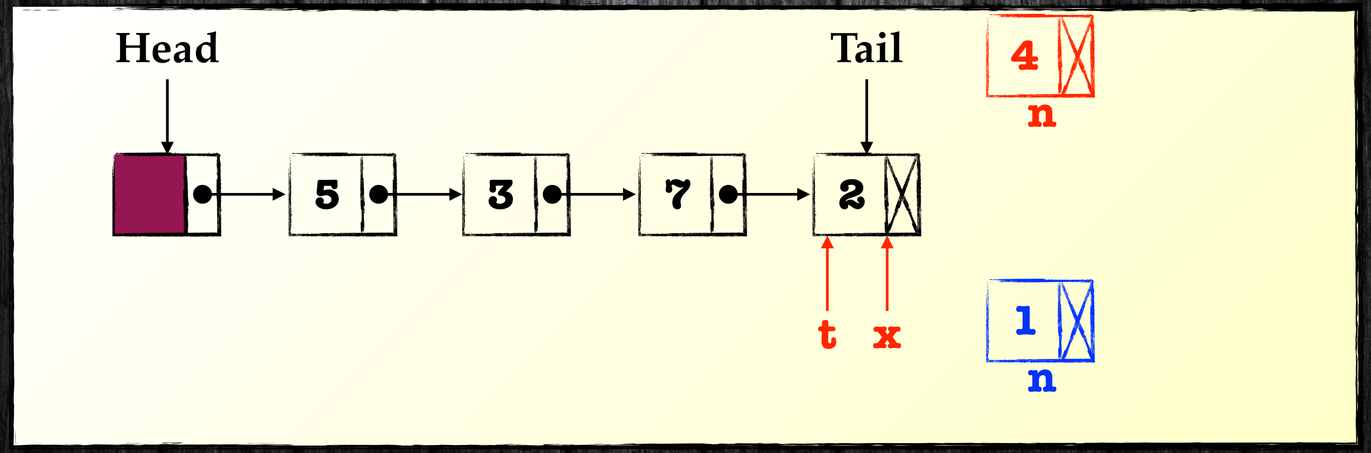
The Michael-Scott Queue Algorithm

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)
```



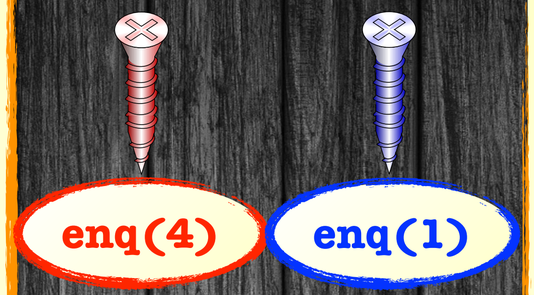

```
enq(k):
Node t,x
1  n = new Node(k,NULL)
2  while (true)
3    t = Tail
4    x = t.next
5    if (t == Tail)
6      if (x == NULL)
7        if (CAS (t.next,NULL,n))
8          CAS (Tail,t,n)
9          exit
10     else
11       CAS (Tail,t,x)
```



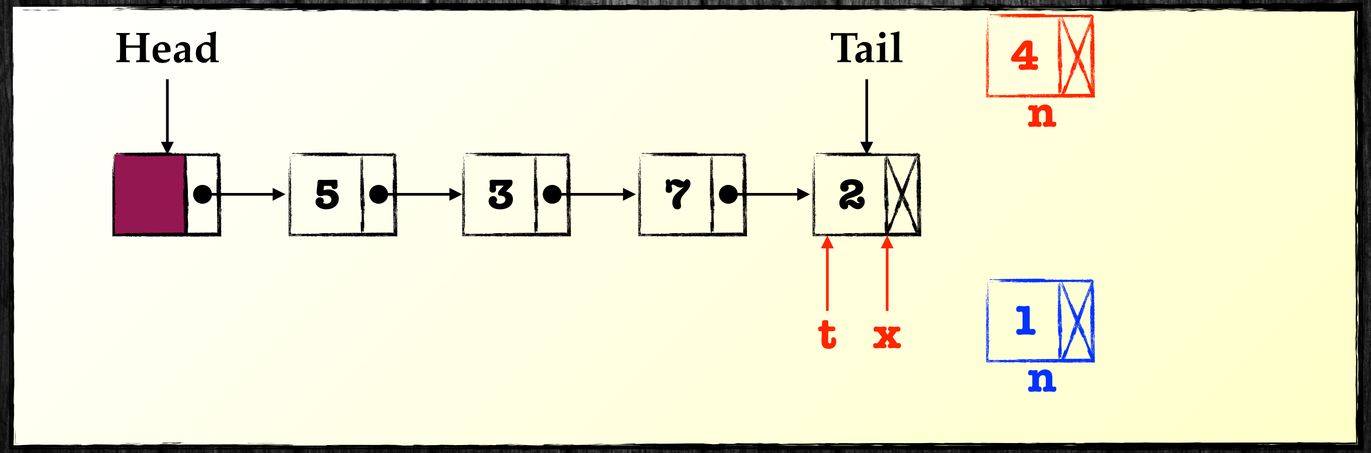
The Michael-Scott Queue Algorithm

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)
```



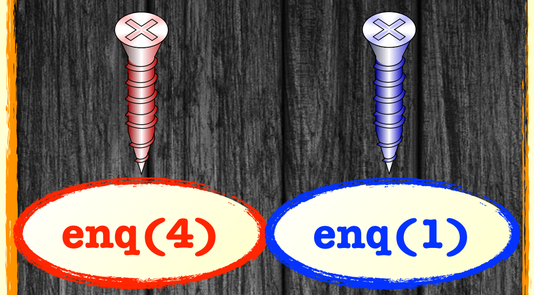

```
enq(k):
Node t,x
1  n = new Node(k,NULL)
2  while (true)
3    t = Tail
4    x = t.next
5    if (t == Tail)
6      if (x == NULL)
7        if (CAS (t.next,NULL,n))
8          CAS (Tail,t,n)
9          exit
10     else
11       CAS (Tail,t,x)
```



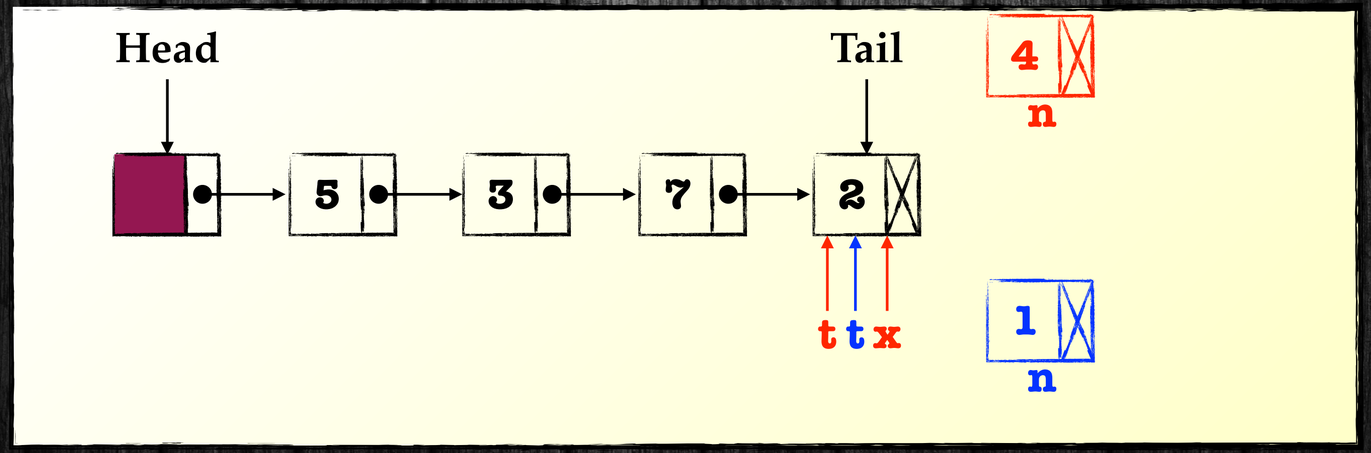
The Michael-Scott Queue Algorithm

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)
```



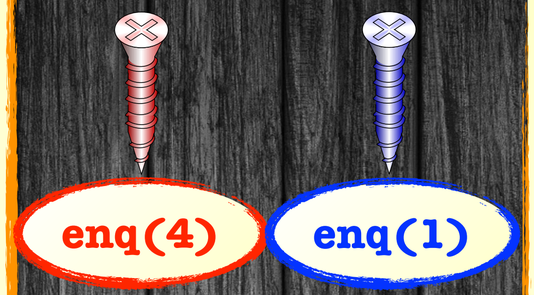

```
enq(k):
Node t,x
1  n = new Node(k,NULL)
2  while (true)
3    t = Tail
4    x = t.next
5    if (t == Tail)
6      if (x == NULL)
7        if (CAS (t.next,NULL,n))
8          CAS (Tail,t,n)
9          exit
10     else
11       CAS (Tail,t,x)
```



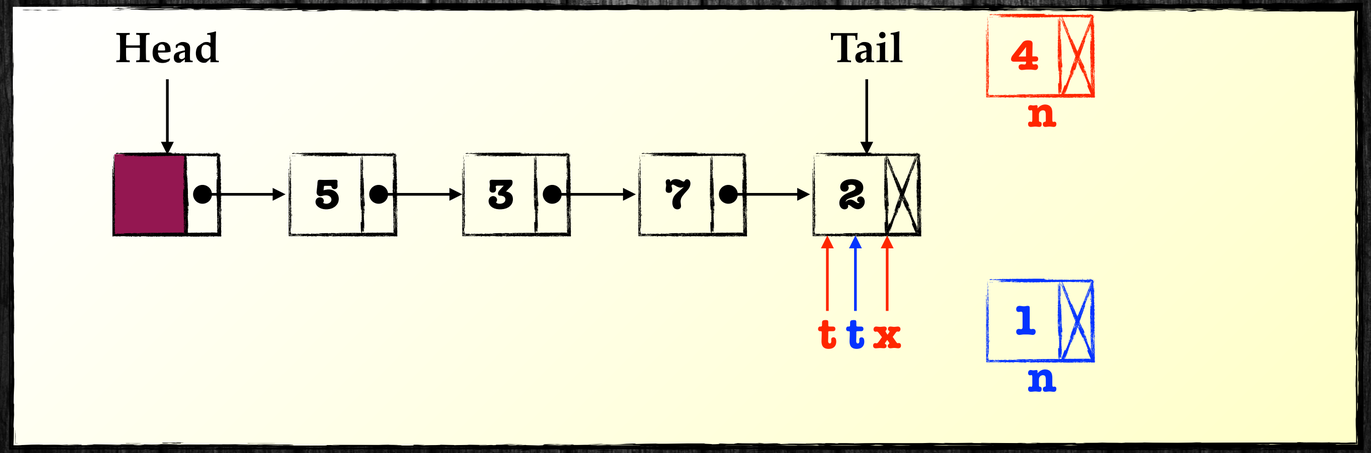
The Michael-Scott Queue Algorithm

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)
```



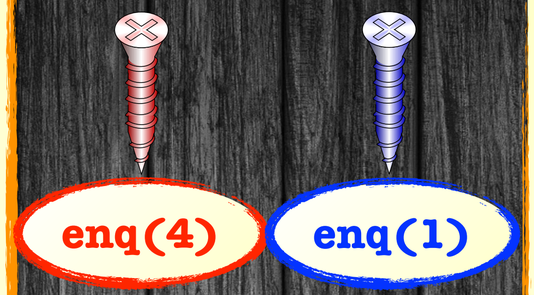

```
enq(k):
Node t,x
1  n = new Node(k,NULL)
2  while (true)
3    t = Tail
4    x = t.next
5    if (t == Tail)
6      if (x == NULL)
7        if (CAS (t.next,NULL,n))
8          CAS (Tail,t,n)
9          exit
10     else
11       CAS (Tail,t,x)
```



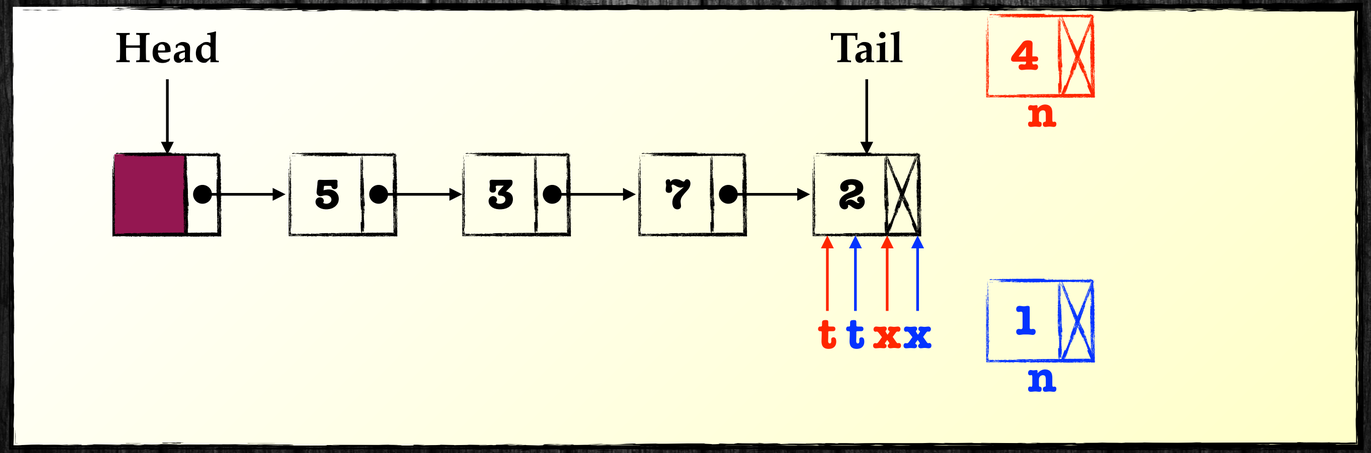
The Michael-Scott Queue Algorithm

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)
```



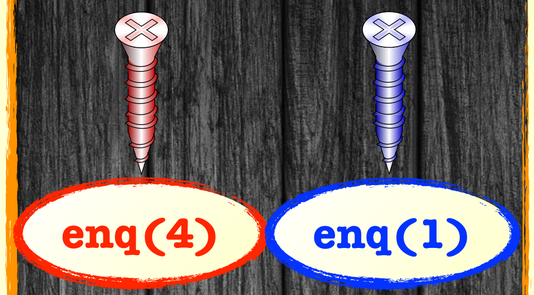

```
enq(k):
Node t,x
1  n = new Node(k,NULL)
2  while (true)
3    t = Tail
4    x = t.next
5    if (t == Tail)
6      if (x == NULL)
7        if (CAS (t.next,NULL,n))
8          CAS (Tail,t,n)
9          exit
10     else
11       CAS (Tail,t,x)
```



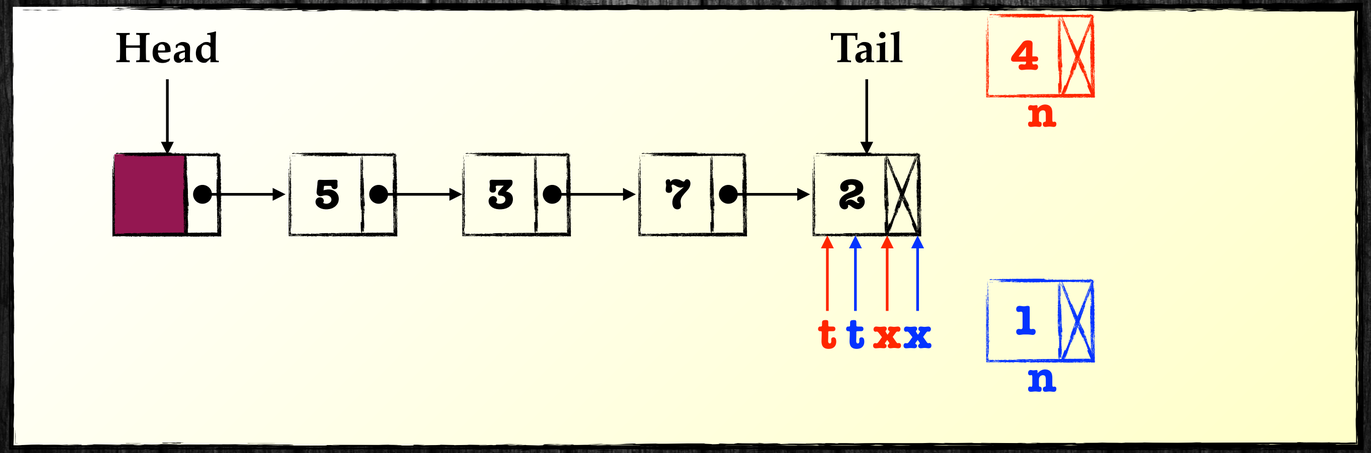
The Michael-Scott Queue Algorithm

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)
```



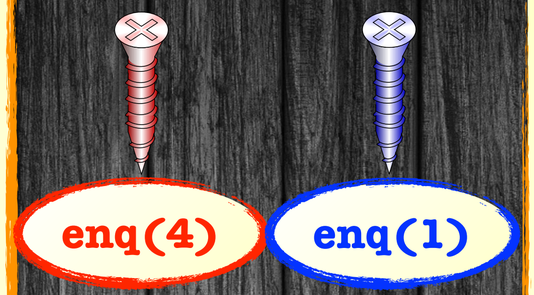

```
enq(k):
Node t,x
1  n = new Node(k,NULL)
2  while (true)
3    t = Tail
4    x = t.next
5    if (t == Tail)
6      if (x == NULL)
7        if (CAS (t.next,NULL,n))
8          CAS (Tail,t,n)
9          exit
10     else
11       CAS (Tail,t,x)
```



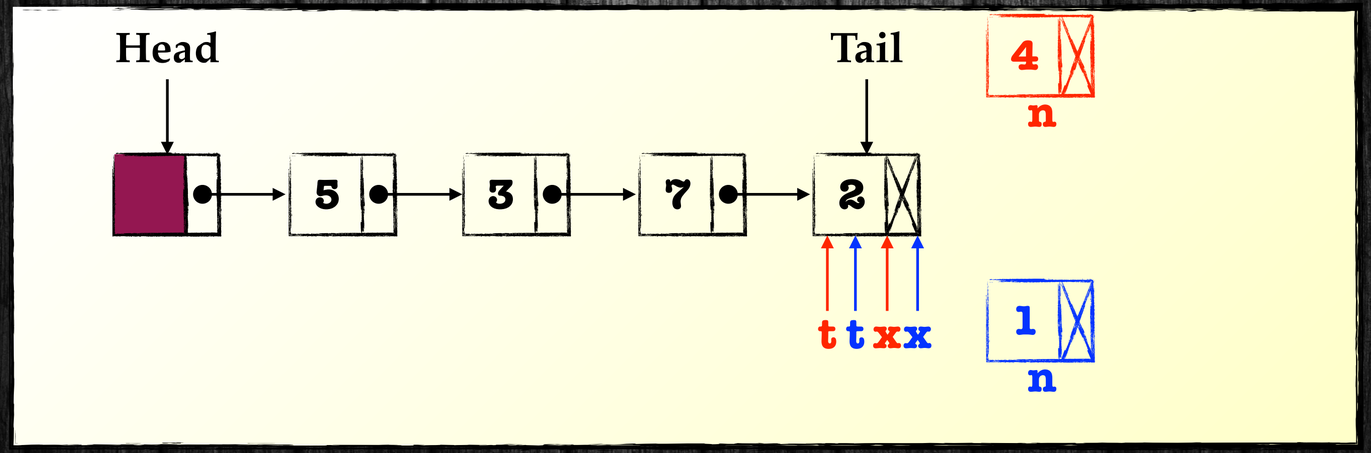
The Michael-Scott Queue Algorithm

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)
```



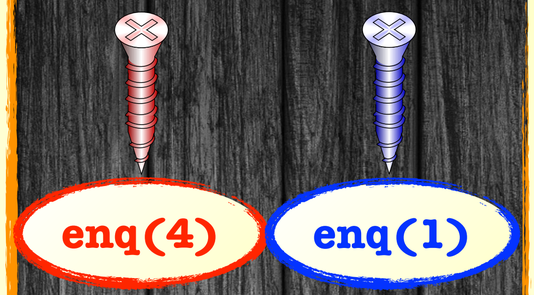

```
enq(k):
Node t,x
1  n = new Node(k,NULL)
2  while (true)
3    t = Tail
4    x = t.next
5    if (t == Tail)
6      if (x == NULL)
7        if (CAS (t.next,NULL,n))
8          CAS (Tail,t,n)
9          exit
10     else
11       CAS (Tail,t,x)
```



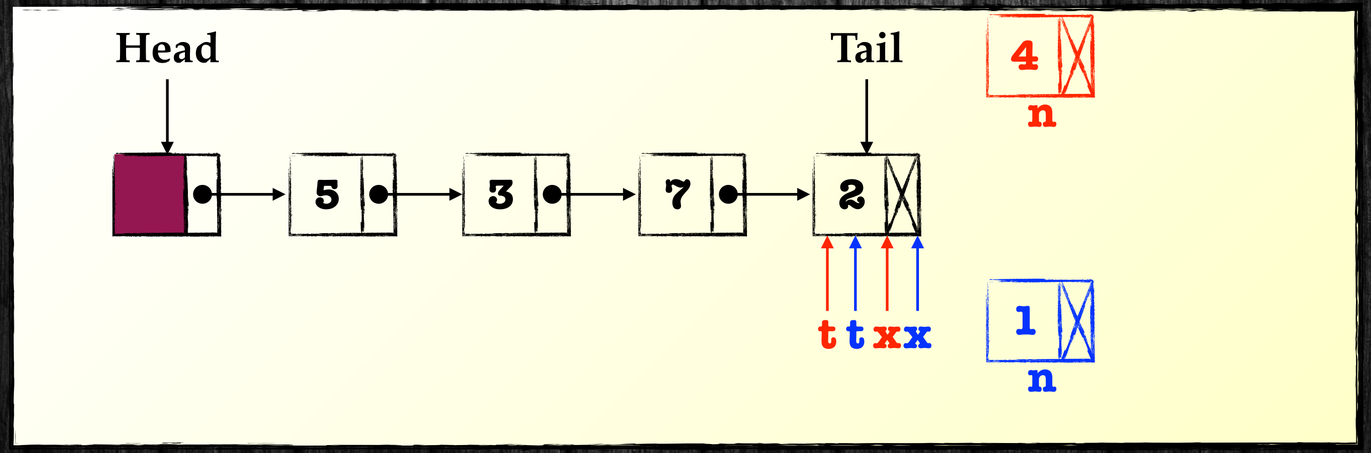
The Michael-Scott Queue Algorithm

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)
```



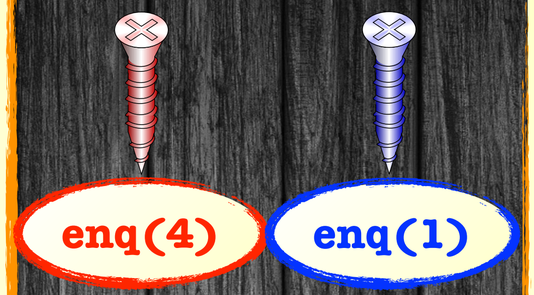

```
enq(k):
Node t,x
1  n = new Node(k,NULL)
2  while (true)
3    t = Tail
4    x = t.next
5    if (t == Tail)
6      if (x == NULL)
7        if (CAS (t.next,NULL,n))
8          CAS (Tail,t,n)
9          exit
10     else
11       CAS (Tail,t,x)
```



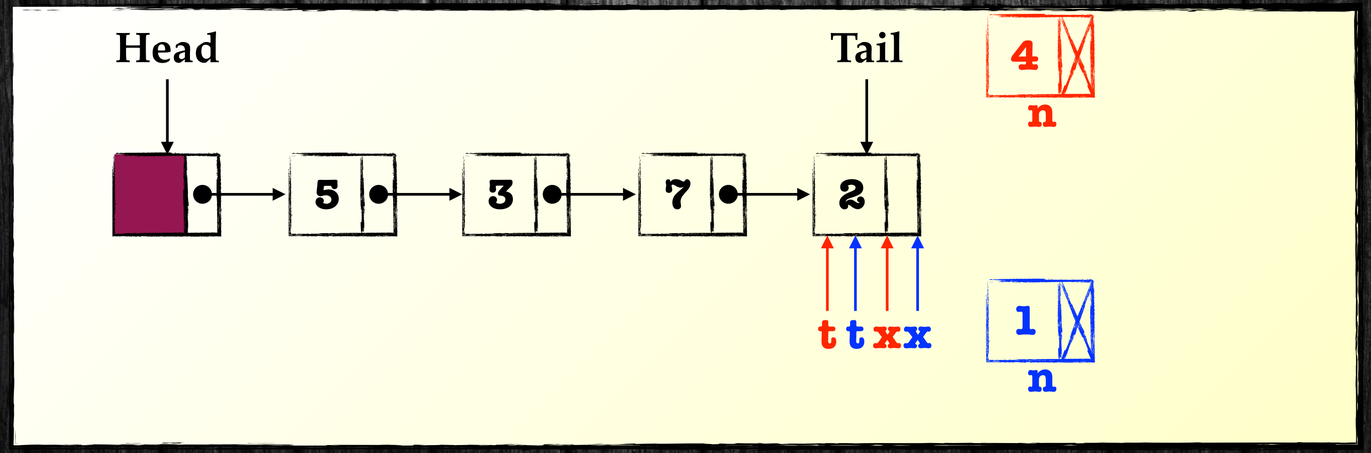
The Michael-Scott Queue Algorithm

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)
```



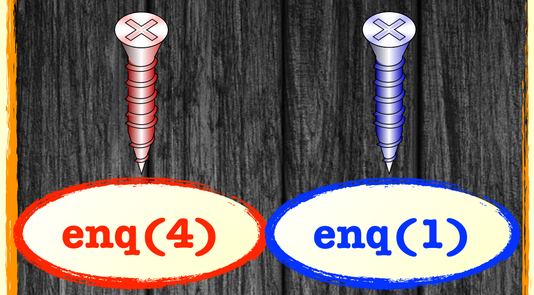

```
enq(k):
Node t,x
1  n = new Node(k,NULL)
2  while (true)
3    t = Tail
4    x = t.next
5    if (t == Tail)
6      if (x == NULL)
7        if (CAS (t.next,NULL,n))
8          CAS (Tail,t,n)
9          exit
10     else
11       CAS (Tail,t,x)
```



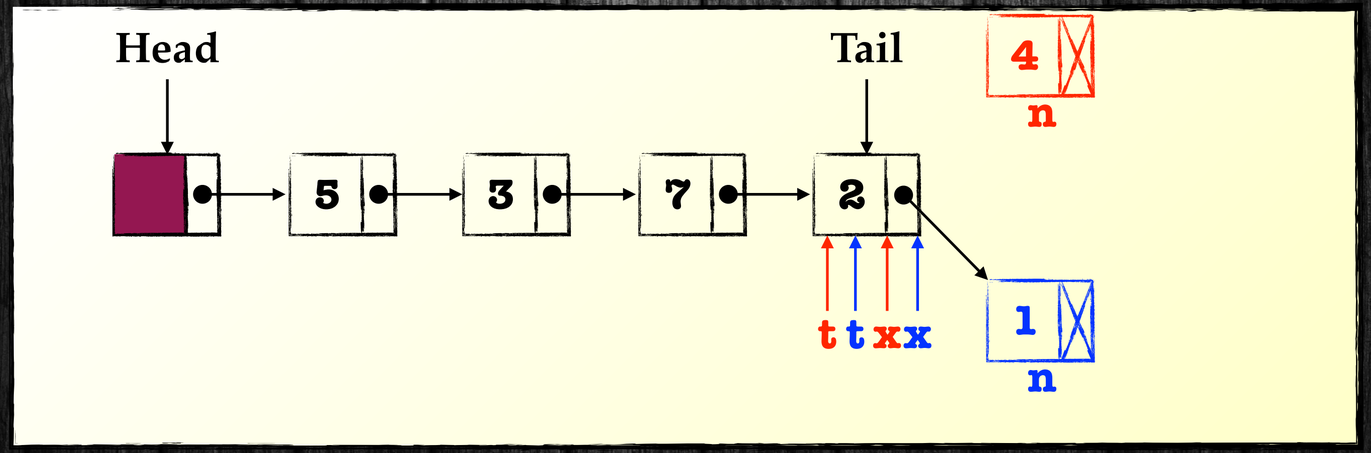
The Michael-Scott Queue Algorithm

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)
```



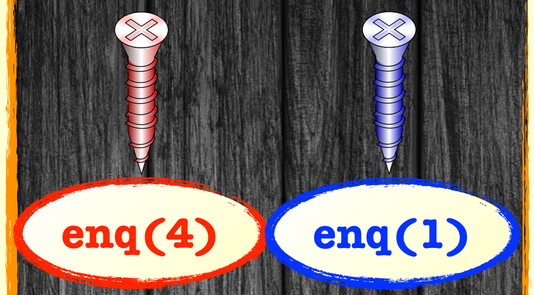

```
enq(k):
Node t,x
1  n = new Node(k,NULL)
2  while (true)
3    t = Tail
4    x = t.next
5    if (t == Tail)
6      if (x == NULL)
7        if (CAS (t.next,NULL,n))
8          CAS (Tail,t,n)
9          exit
10     else
11       CAS (Tail,t,x)
```



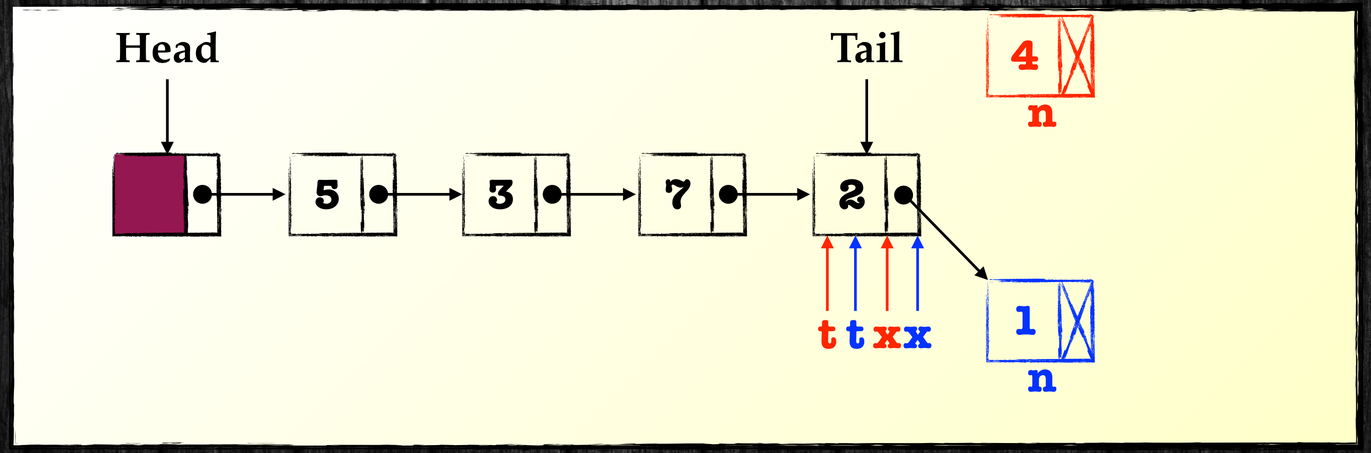
The Michael-Scott Queue Algorithm

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)
```



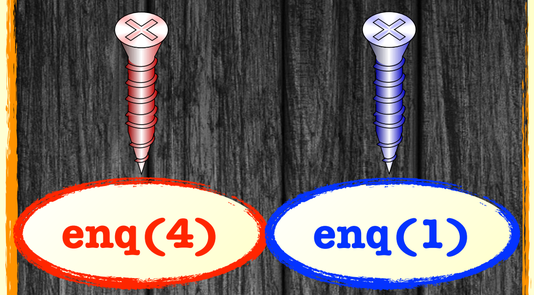

```
enq(k):
Node t,x
1  n = new Node(k,NULL)
2  while (true)
3    t = Tail
4    x = t.next
5    if (t == Tail)
6      if (x == NULL)
7        if (CAS (t.next,NULL,n))
8          CAS (Tail,t,n)
9          exit
10     else
11       CAS (Tail,t,x)
```



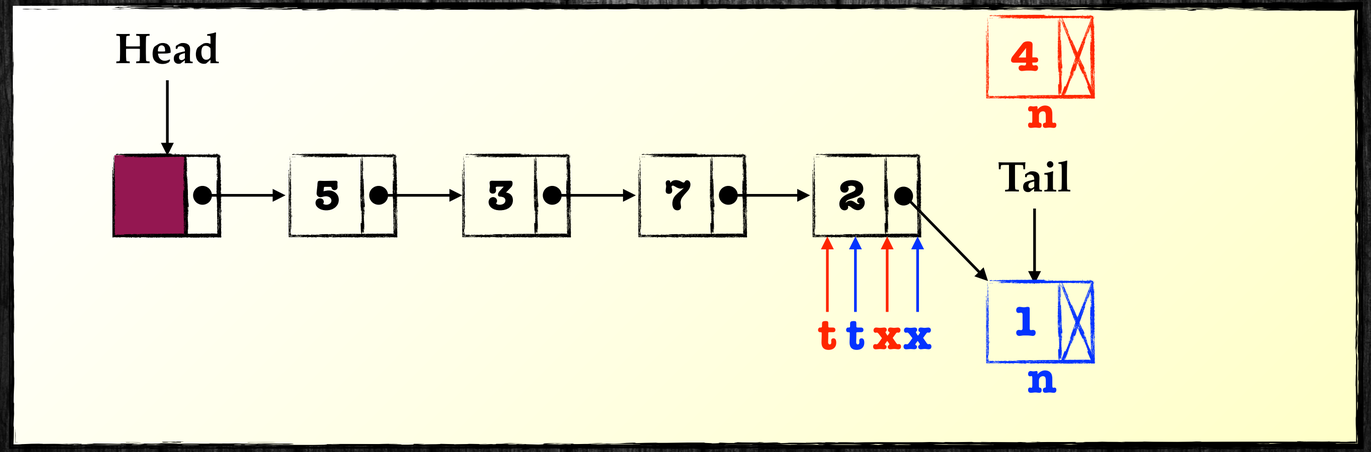
The Michael-Scott Queue Algorithm

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)
```



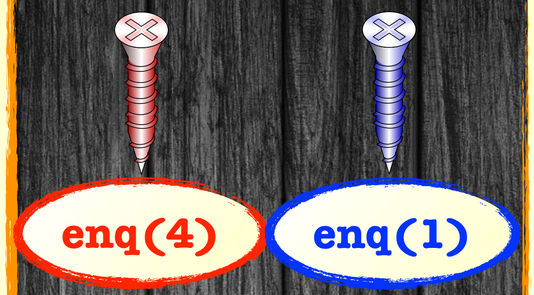

```
enq(k):
Node t,x
1  n = new Node(k,NULL)
2  while (true)
3    t = Tail
4    x = t.next
5    if (t == Tail)
6      if (x == NULL)
7        if (CAS (t.next,NULL,n))
8          CAS (Tail,t,n)
9          exit
10     else
11       CAS (Tail,t,x)
```



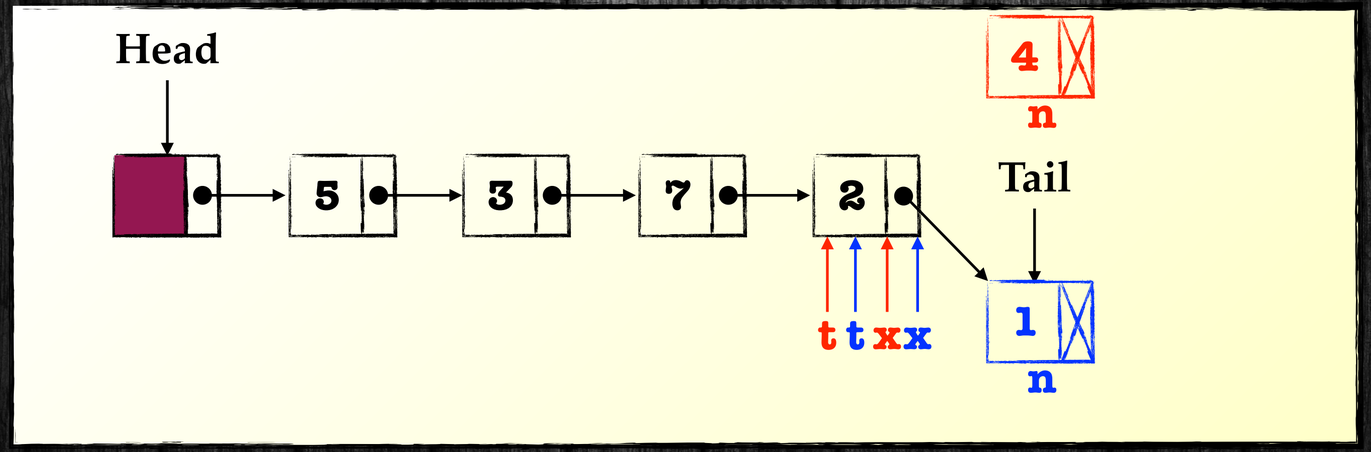
The Michael-Scott Queue Algorithm

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)
```

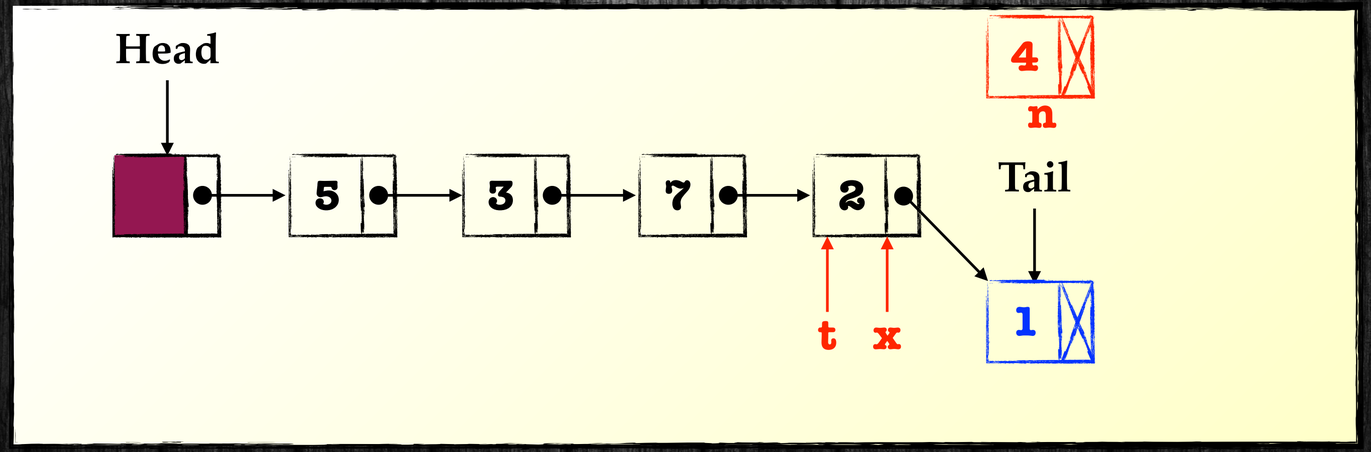


The Michael-Scott Queue Algorithm

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)
```

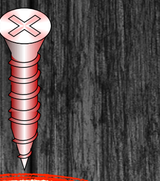




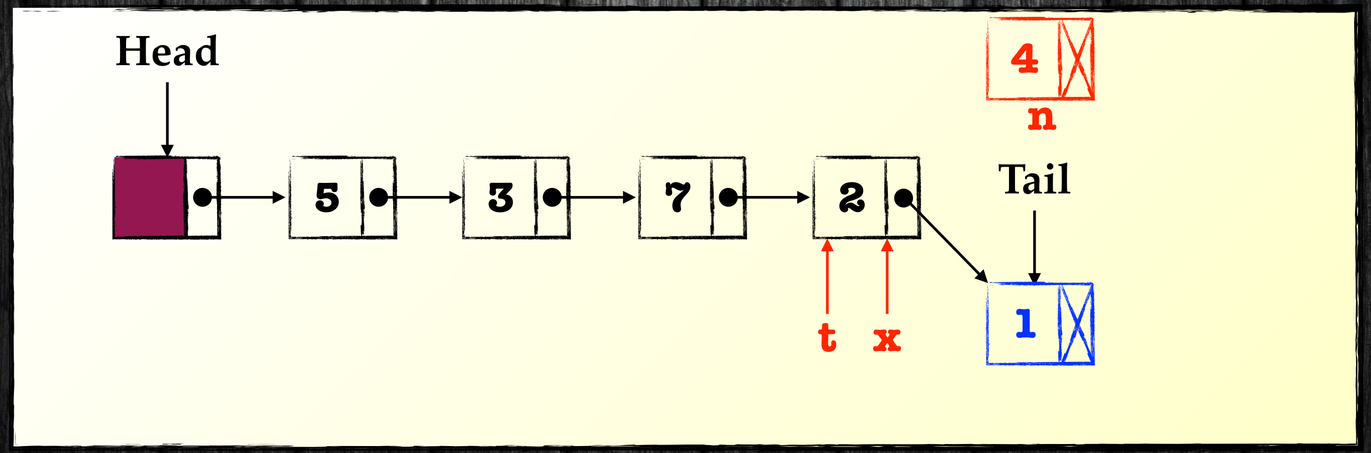
The Michael-Scott Queue Algorithm

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)
```



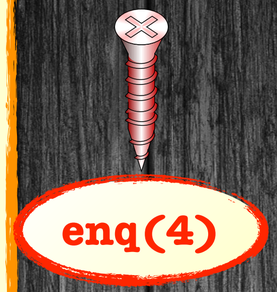
enq(4)

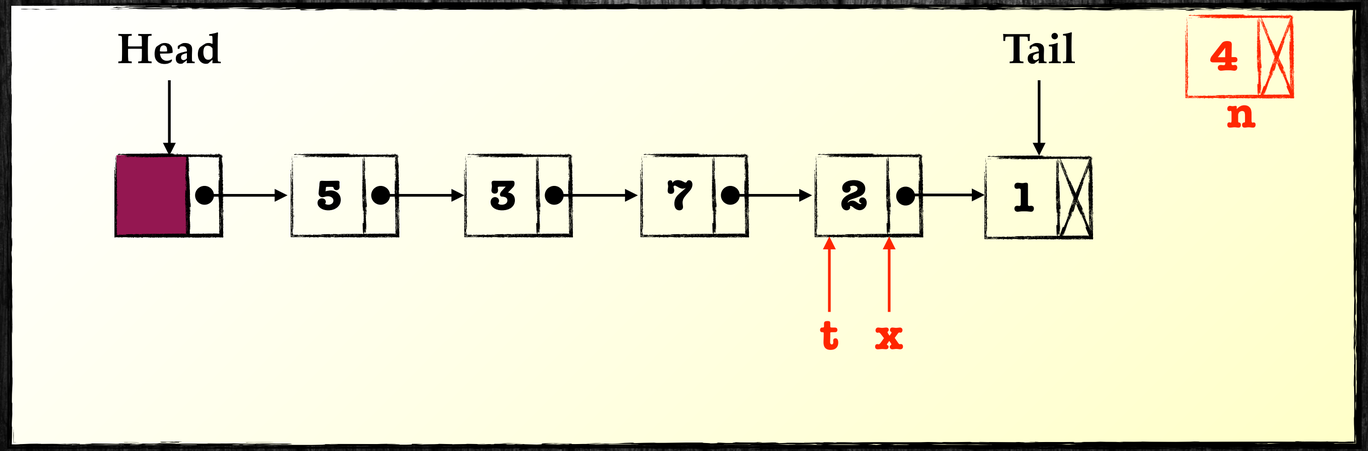


The Michael-Scott Queue Algorithm

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)
```

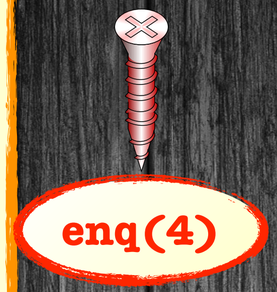


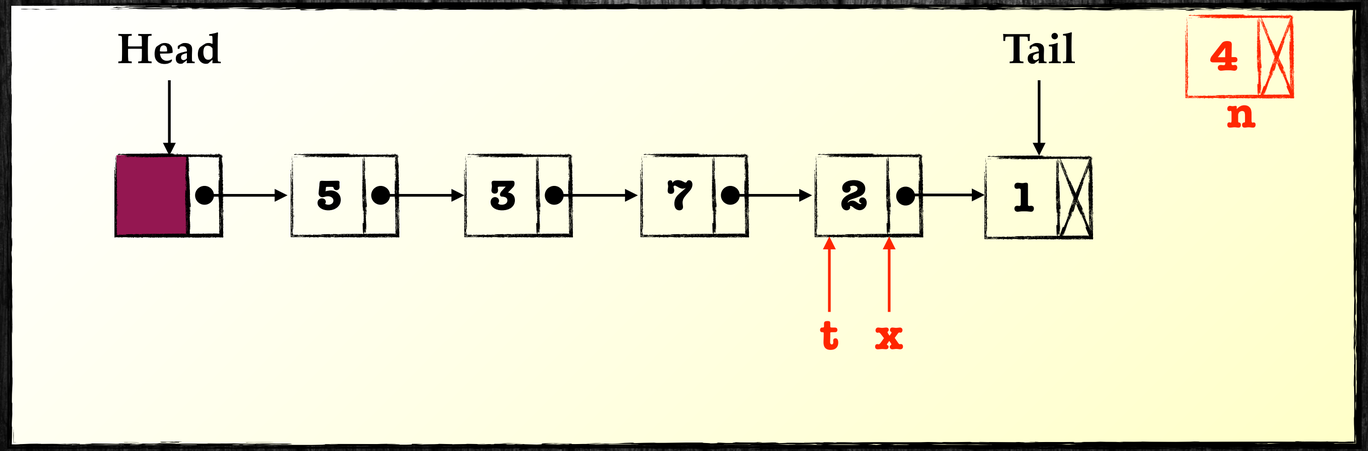


The Michael-Scott Queue Algorithm

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)
```



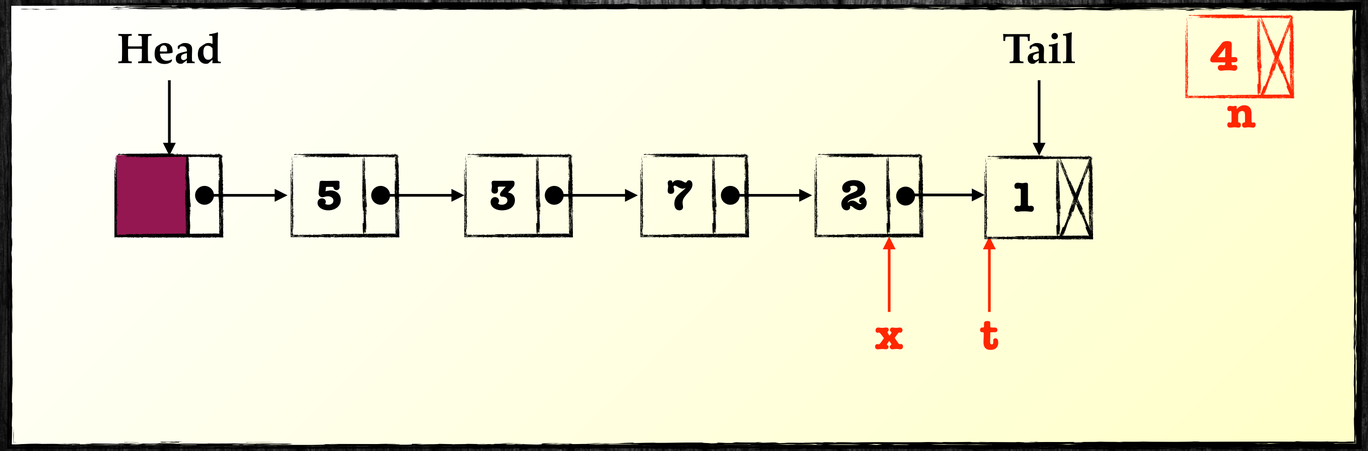


The Michael-Scott Queue Algorithm

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)
```



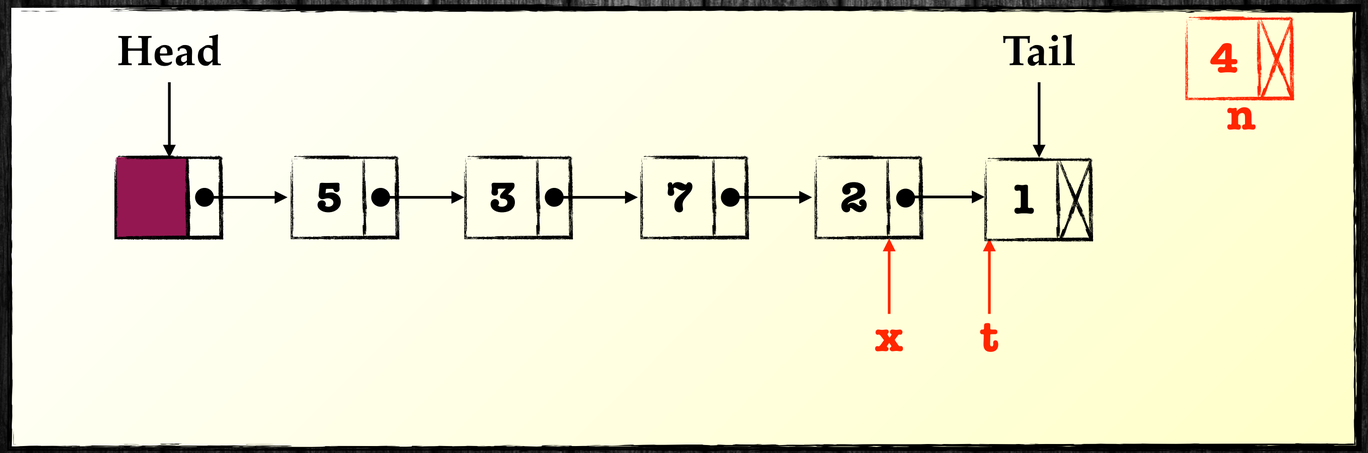


The Michael-Scott Queue Algorithm

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)
```

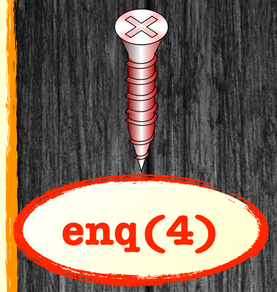


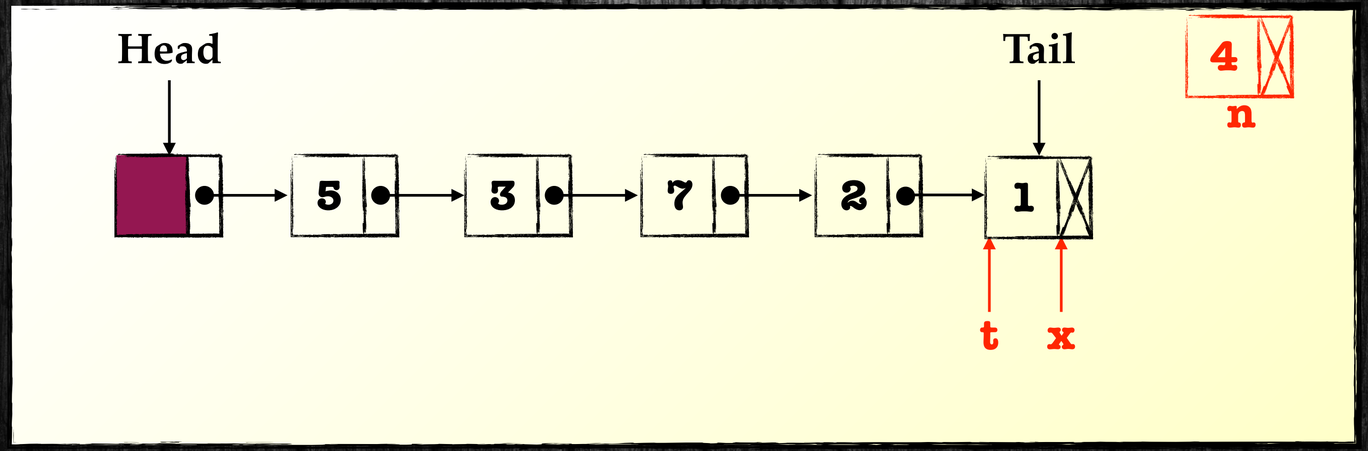


The Michael-Scott Queue Algorithm

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)
```



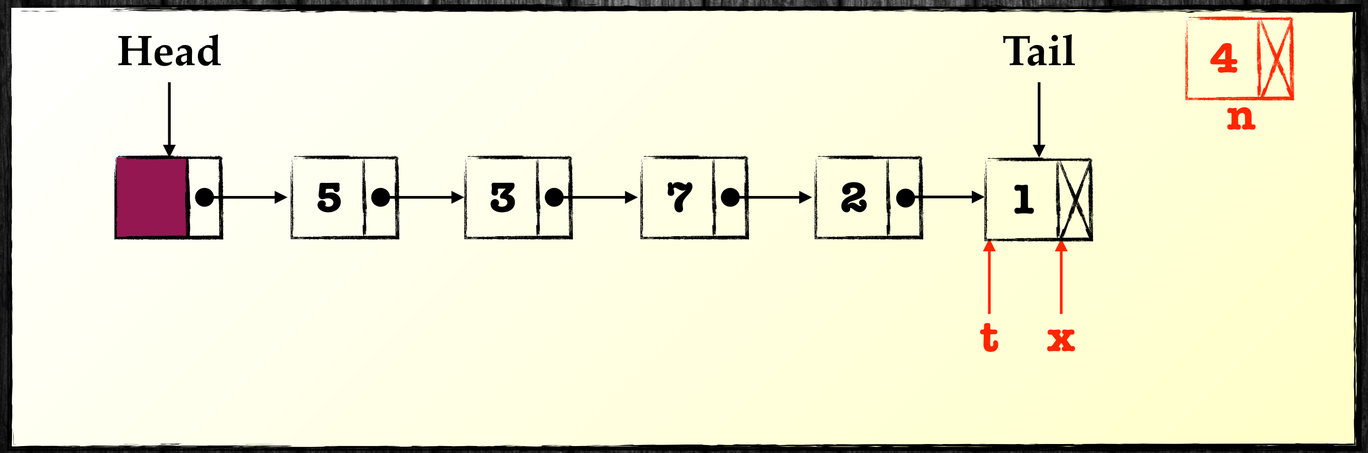


The Michael-Scott Queue Algorithm

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)
```



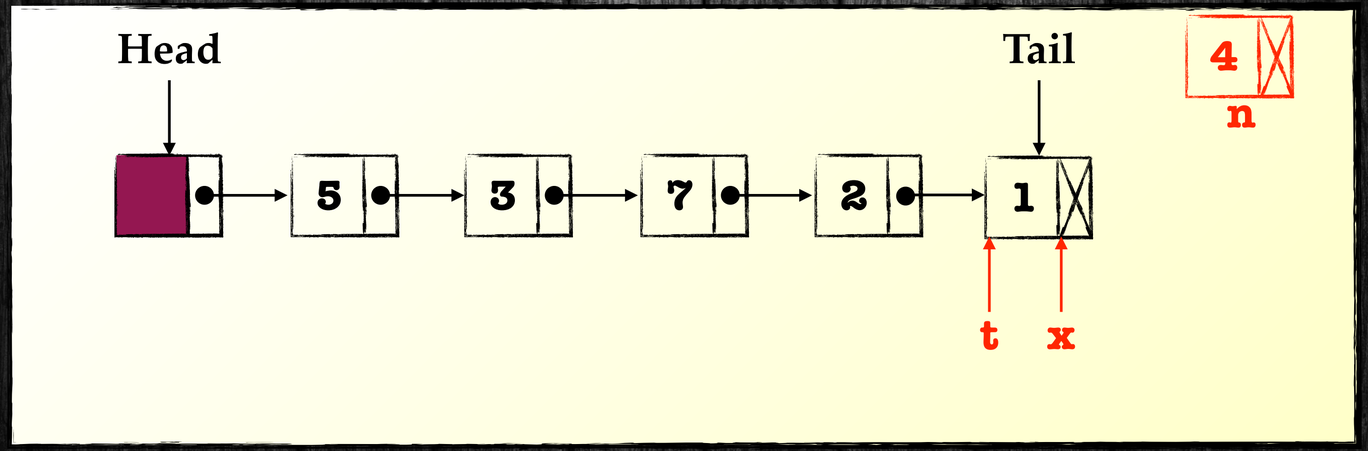


The Michael-Scott Queue Algorithm

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)
```

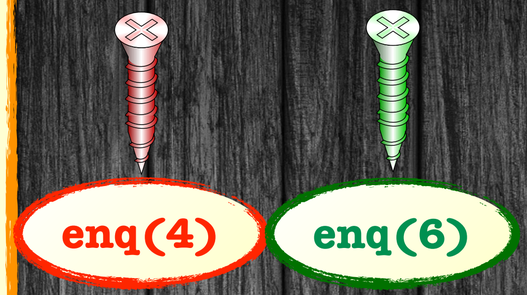


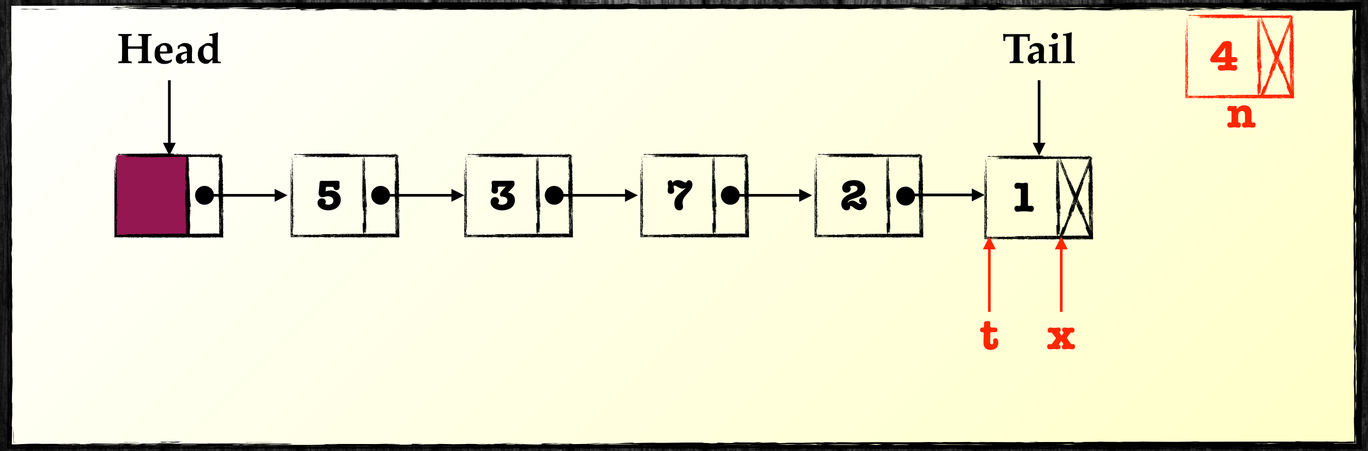


The Michael-Scott Queue Algorithm

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)
```

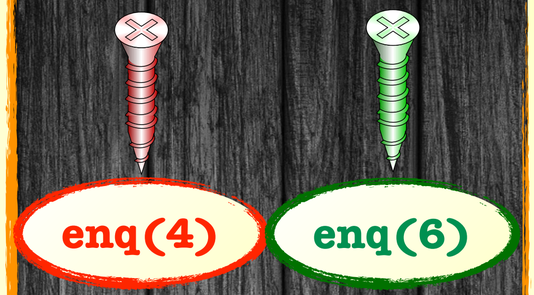




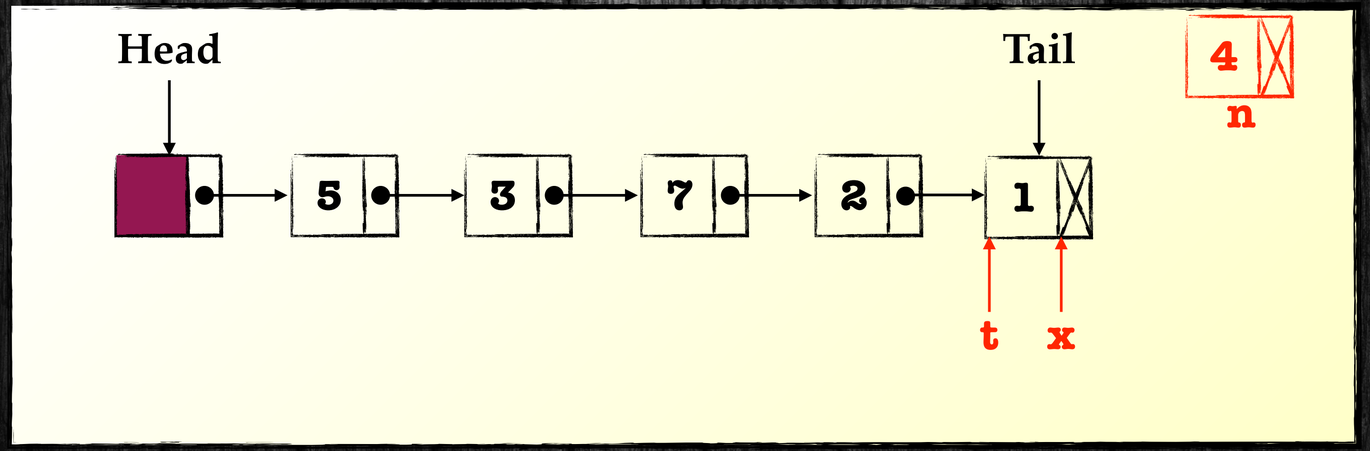
The Michael-Scott Queue Algorithm

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)
```



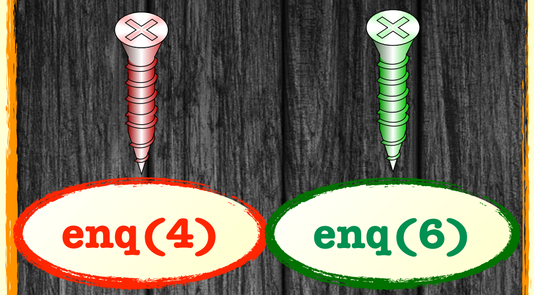

```
enq(k):
Node t,x
1  n = new Node(k,NULL)
2  while (true)
3    t = Tail
4    x = t.next
5    if (t == Tail)
6      if (x == NULL)
7        if (CAS (t.next,NULL,n))
8          CAS (Tail,t,n)
9          exit
10     else
11       CAS (Tail,t,x)
```



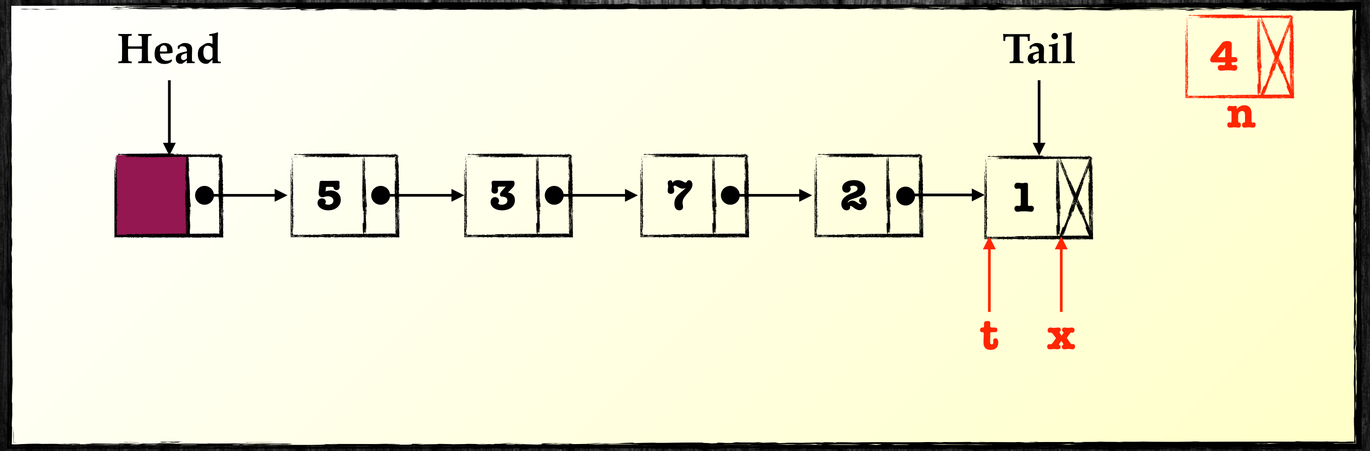
The Michael-Scott Queue Algorithm

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)
```



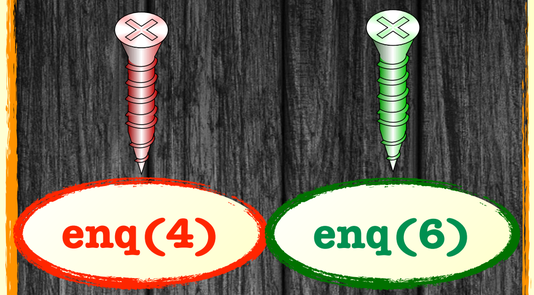

```
enq(k):
Node t,x
1  n = new Node(k,NULL)
2  while (true)
3    t = Tail
4    x = t.next
5    if (t == Tail)
6      if (x == NULL)
7        if (CAS (t.next,NULL,n))
8          CAS (Tail,t,n)
9          exit
10     else
11       CAS (Tail,t,x)
```



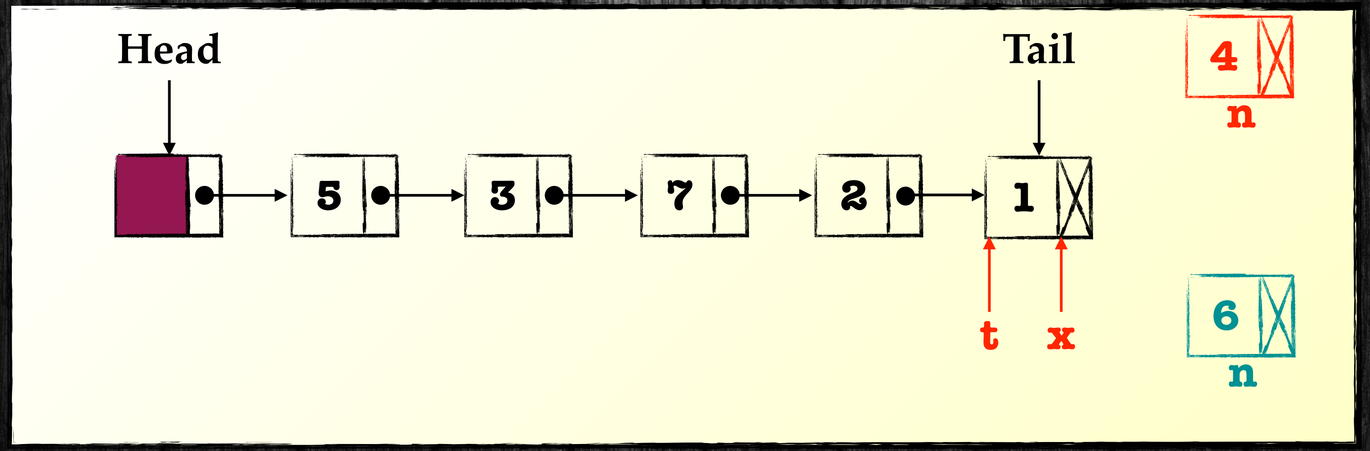
The Michael-Scott Queue Algorithm

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)
```



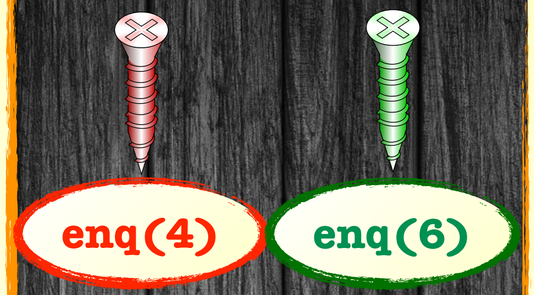

```
enq(k):
Node t,x
1  n = new Node(k,NULL)
2  while (true)
3    t = Tail
4    x = t.next
5    if (t == Tail)
6      if (x == NULL)
7        if (CAS (t.next,NULL,n))
8          CAS (Tail,t,n)
9          exit
10     else
11       CAS (Tail,t,x)
```



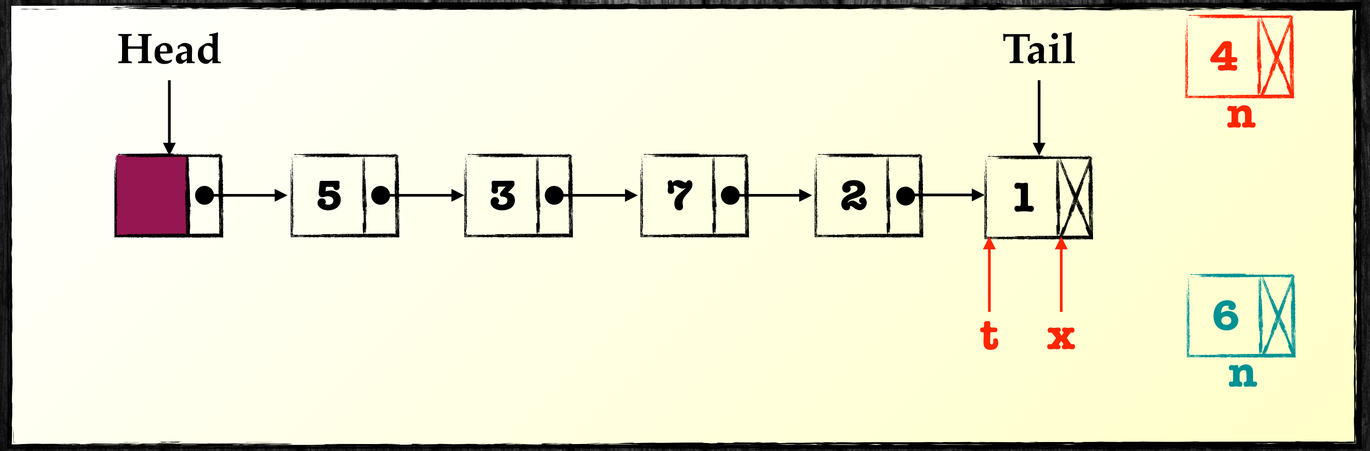
The Michael-Scott Queue Algorithm

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)
```



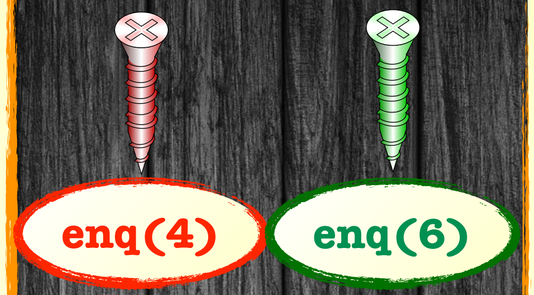

```
enq(k):
Node t,x
1  n = new Node(k,NULL)
2  while (true)
3    t = Tail
4    x = t.next
5    if (t == Tail)
6      if (x == NULL)
7        if (CAS (t.next,NULL,n))
8          CAS (Tail,t,n)
9          exit
10     else
11       CAS (Tail,t,x)
```



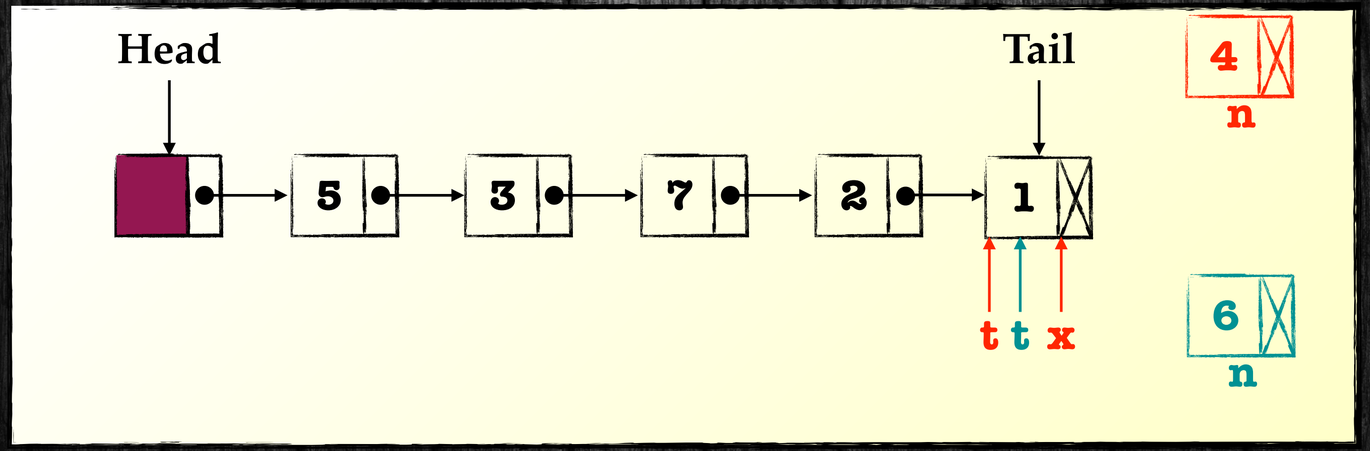
The Michael-Scott Queue Algorithm

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)
```



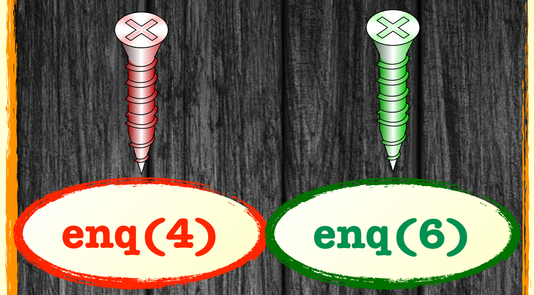

```
enq(k):
Node t,x
1  n = new Node(k,NULL)
2  while (true)
3    t = Tail
4    x = t.next
5    if (t == Tail)
6      if (x == NULL)
7        if (CAS (t.next,NULL,n))
8          CAS (Tail,t,n)
9          exit
10     else
11       CAS (Tail,t,x)
```



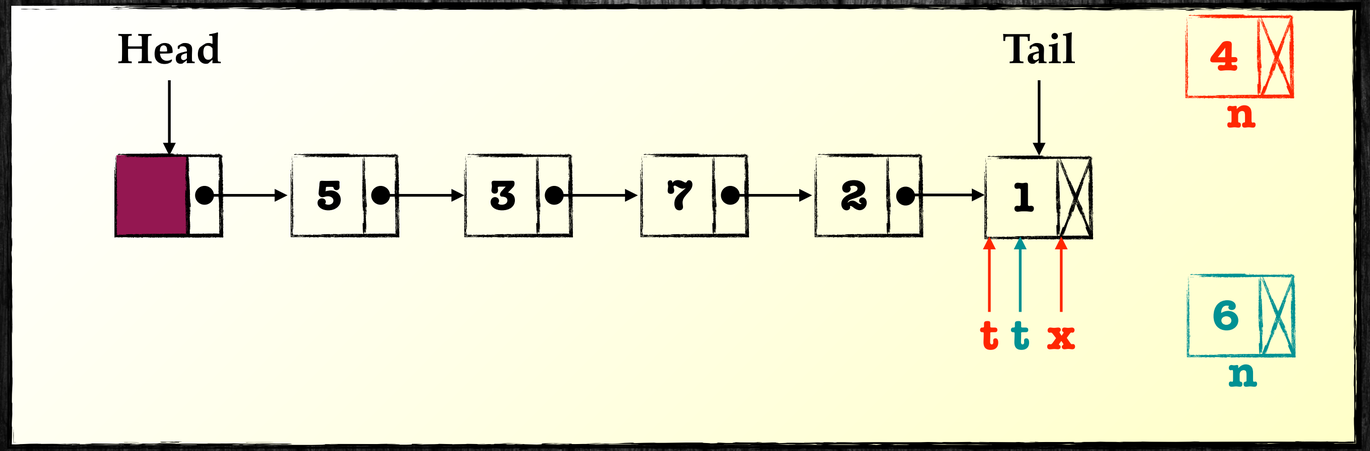
The Michael-Scott Queue Algorithm

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)
```



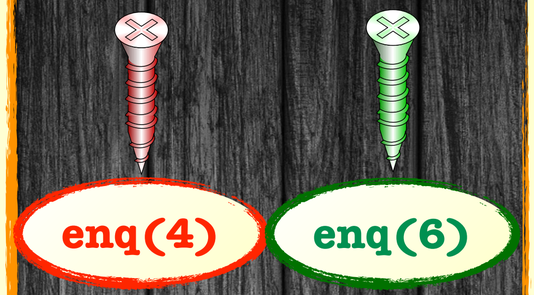

```
enq(k):
Node t,x
1  n = new Node(k,NULL)
2  while (true)
3    t = Tail
4    x = t.next
5    if (t == Tail)
6      if (x == NULL)
7        if (CAS (t.next,NULL,n))
8          CAS (Tail,t,n)
9          exit
10     else
11       CAS (Tail,t,x)
```



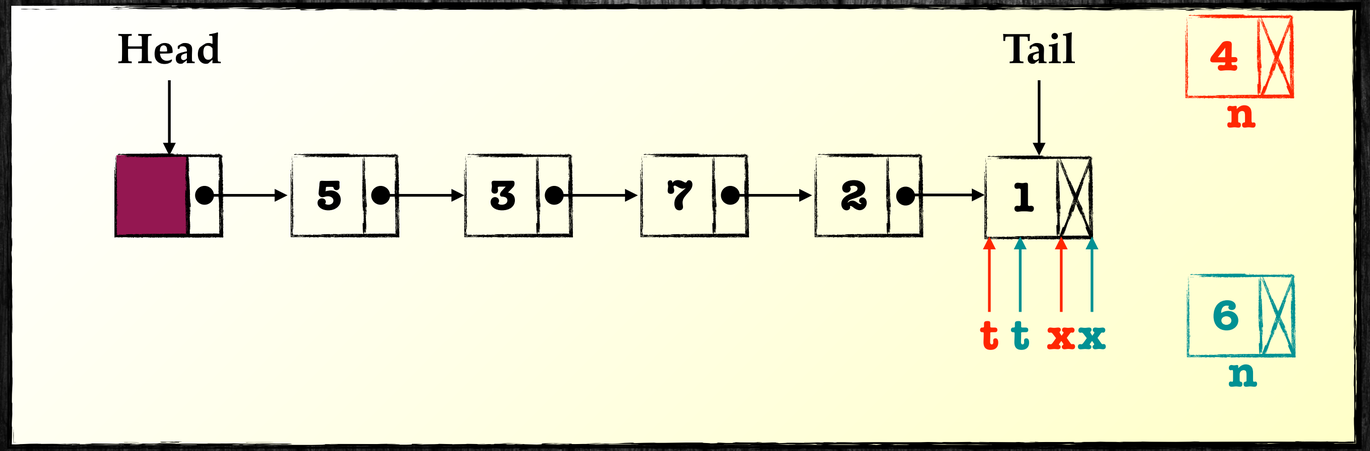
The Michael-Scott Queue Algorithm

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)
```



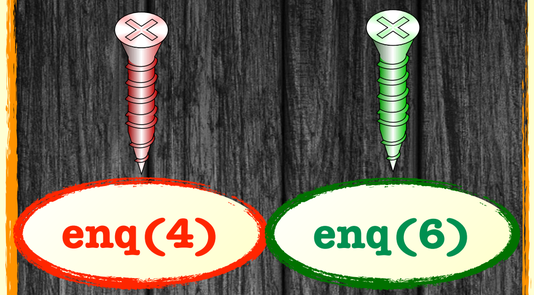

```
enq(k):
Node t,x
1  n = new Node(k,NULL)
2  while (true)
3    t = Tail
4    x = t.next
5    if (t == Tail)
6      if (x == NULL)
7        if (CAS (t.next,NULL,n))
8          CAS (Tail,t,n)
9          exit
10     else
11       CAS (Tail,t,x)
```



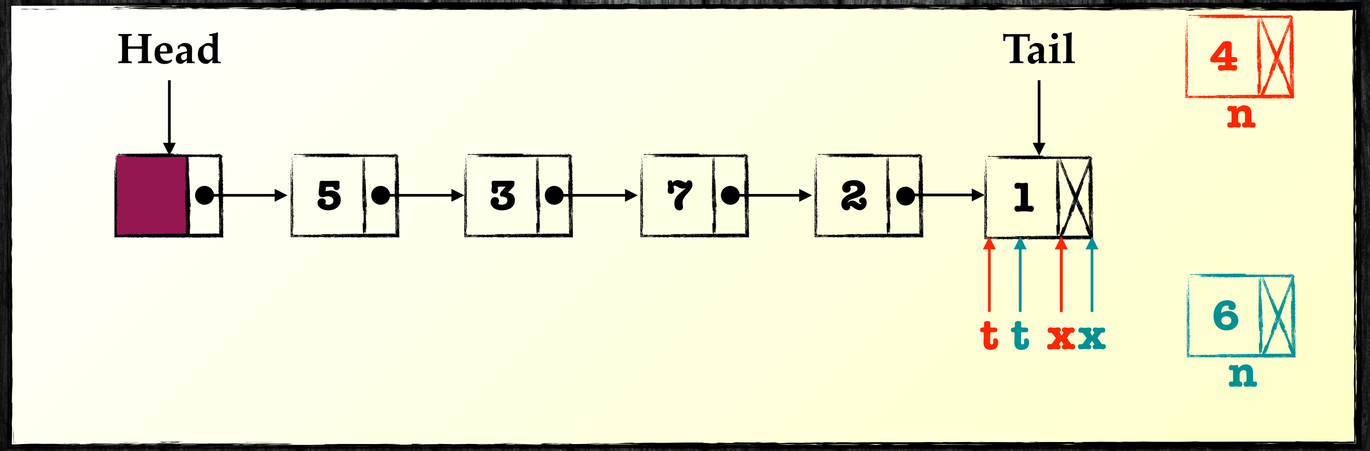
The Michael-Scott Queue Algorithm

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)
```



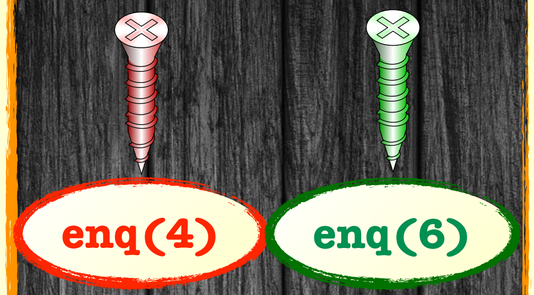

```
enq(k):
Node t,x
1  n = new Node(k,NULL)
2  while (true)
3    t = Tail
4    x = t.next
5    if (t == Tail)
6      if (x == NULL)
7        if (CAS (t.next,NULL,n))
8          CAS (Tail,t,n)
9          exit
10     else
11       CAS (Tail,t,x)
```



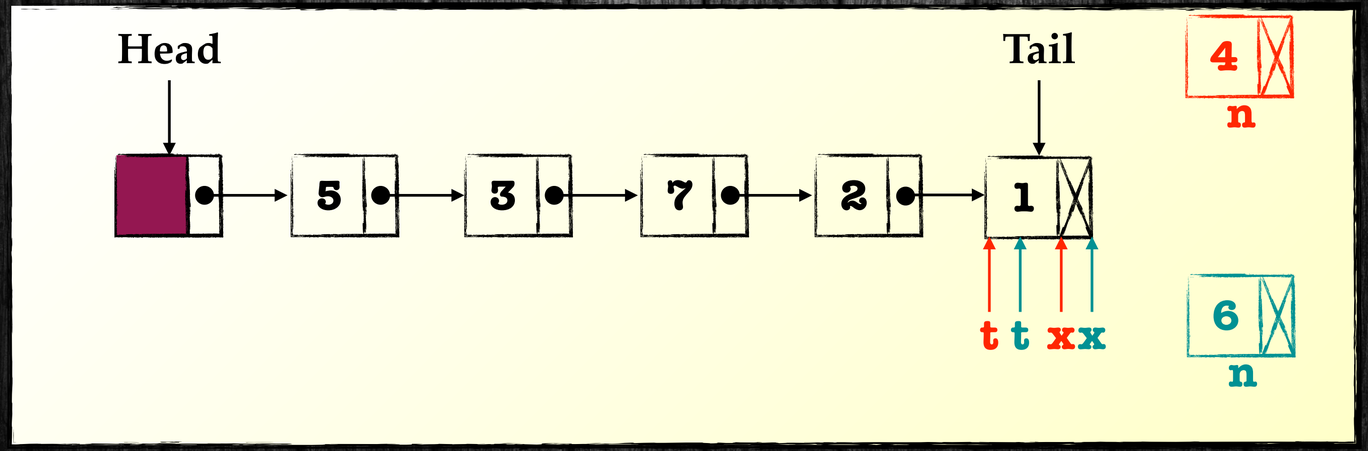
The Michael-Scott Queue Algorithm

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)
```



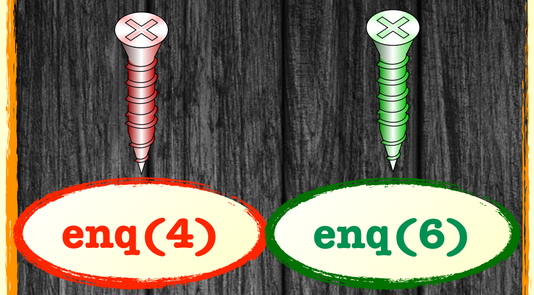

```
enq(k):
Node t,x
1  n = new Node(k,NULL)
2  while (true)
3    t = Tail
4    x = t.next
5    if (t == Tail)
6      if (x == NULL)
7        if (CAS (t.next,NULL,n))
8          CAS (Tail,t,n)
9          exit
10     else
11       CAS (Tail,t,x)
```

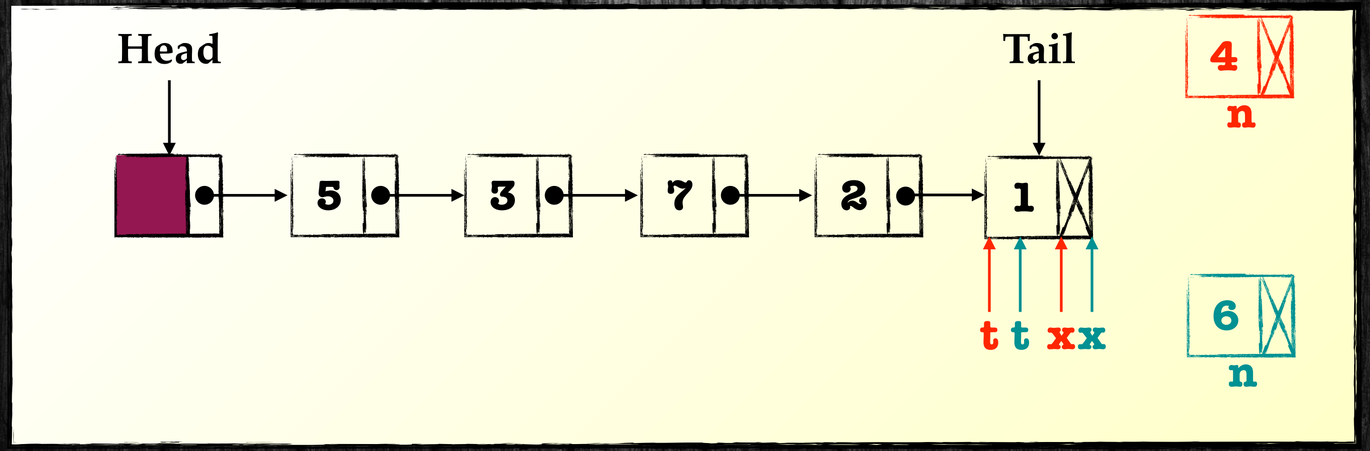
The Michael-Scott Queue Algorithm

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)
```



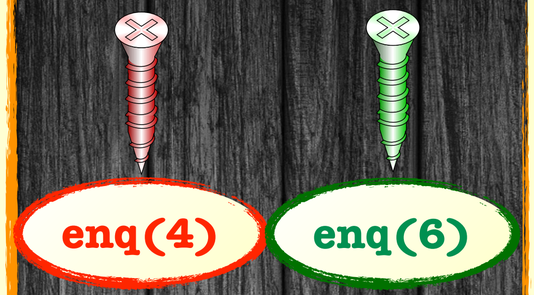

```
enq(k):
Node t,x
1  n = new Node(k,NULL)
2  while (true)
3    t = Tail
4    x = t.next
5    if (t == Tail)
6      if (x == NULL)
7        if (CAS (t.next,NULL,n))
8          CAS (Tail,t,n)
9          exit
10     else
11       CAS (Tail,t,x)
```



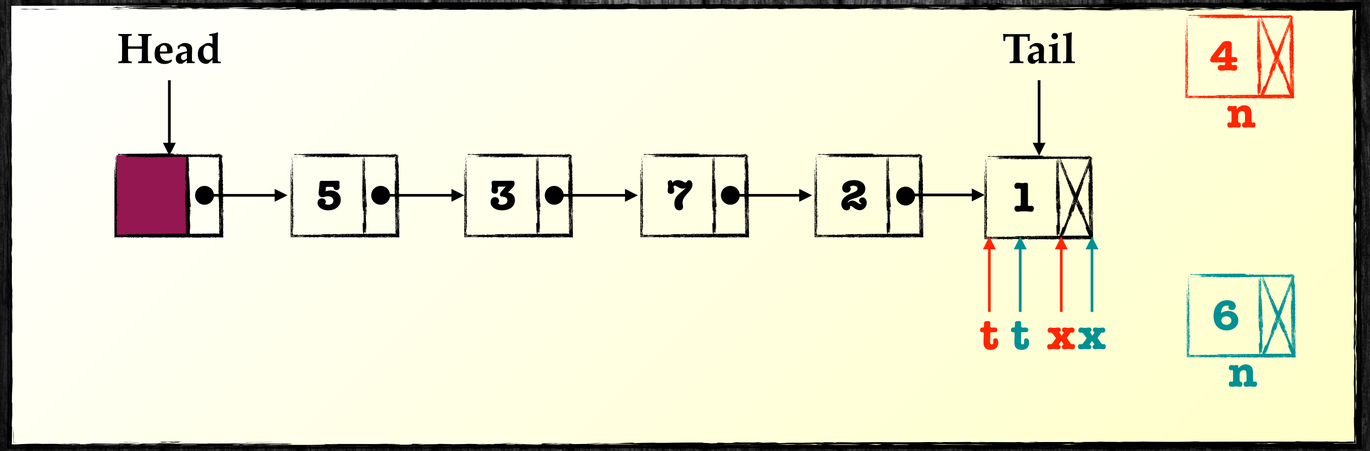
The Michael-Scott Queue Algorithm

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)
```



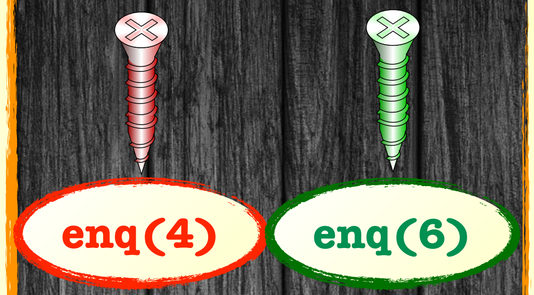

```
enq(k):
Node t,x
1  n = new Node(k,NULL)
2  while (true)
3    t = Tail
4    x = t.next
5    if (t == Tail)
6      if (x == NULL)
7        if (CAS (t.next,NULL,n))
8          CAS (Tail,t,n)
9          exit
10     else
11       CAS (Tail,t,x)
```



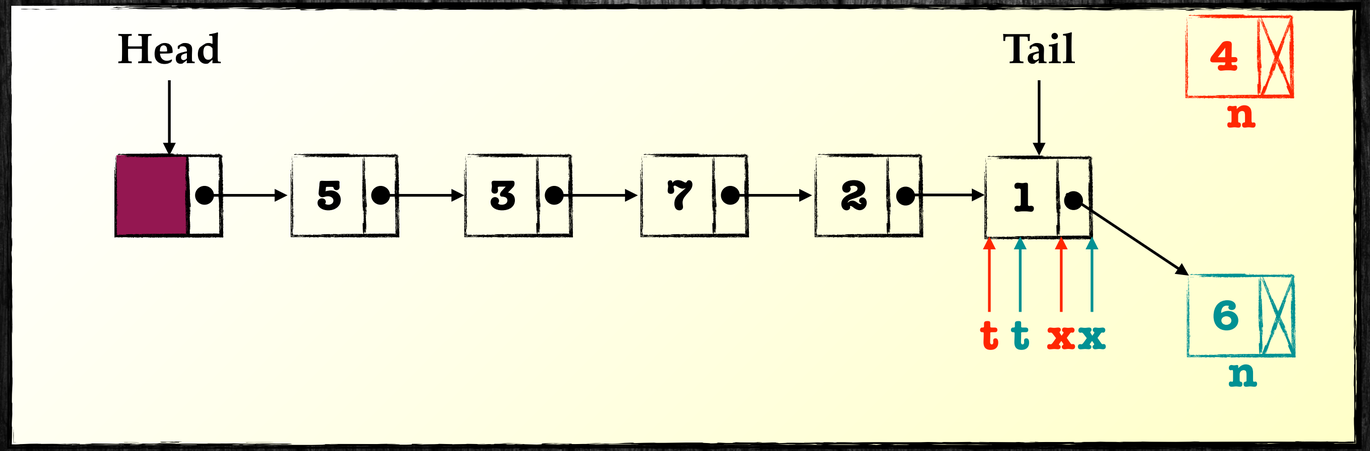
The Michael-Scott Queue Algorithm

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)
```



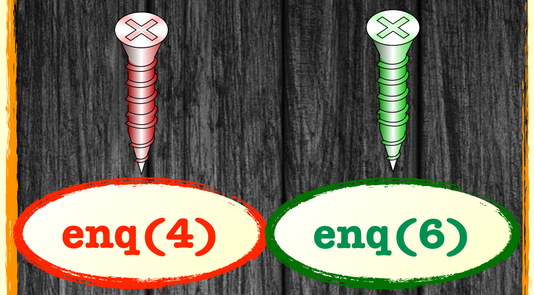

```
enq(k):
Node t,x
1  n = new Node(k,NULL)
2  while (true)
3    t = Tail
4    x = t.next
5    if (t == Tail)
6      if (x == NULL)
7        if (CAS (t.next,NULL,n))
8          CAS (Tail,t,n)
9          exit
10     else
11       CAS (Tail,t,x)
```



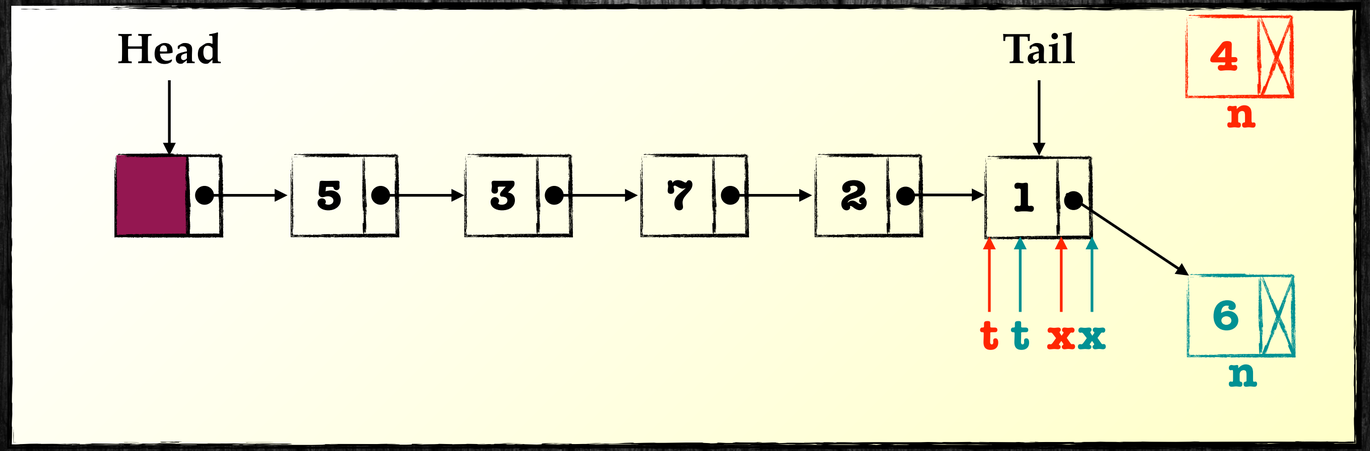
The Michael-Scott Queue Algorithm

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)
```



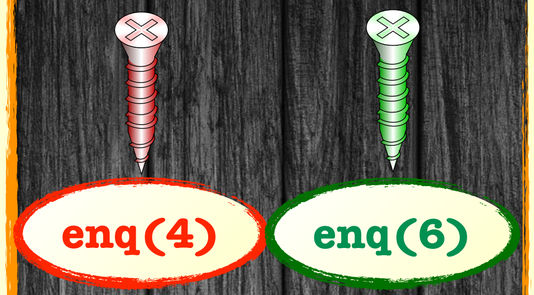

```
enq(k):
Node t,x
1  n = new Node(k,NULL)
2  while (true)
3    t = Tail
4    x = t.next
5    if (t == Tail)
6      if (x == NULL)
7        if (CAS (t.next,NULL,n))
8          CAS (Tail,t,n)
9          exit
10     else
11       CAS (Tail,t,x)
```



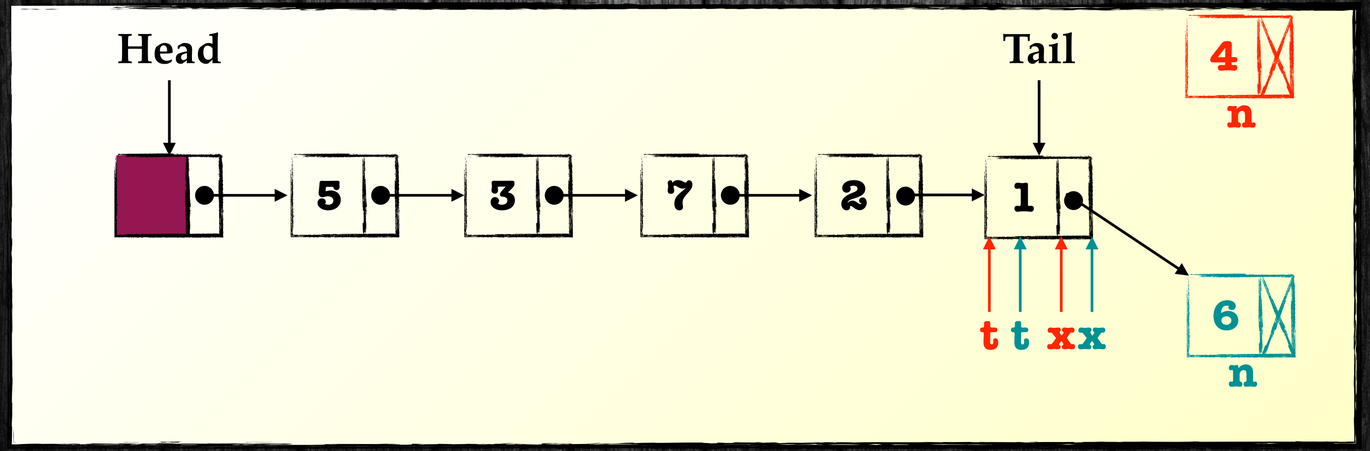
The Michael-Scott Queue Algorithm

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)
```



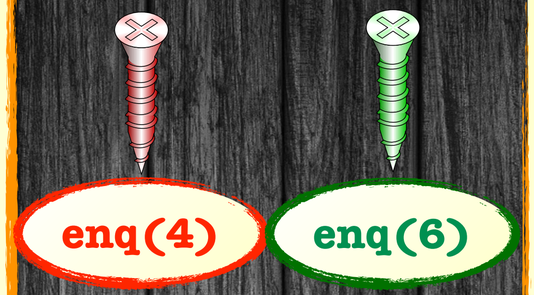

```
enq(k):
Node t,x
1  n = new Node(k,NULL)
2  while (true)
3    t = Tail
4    x = t.next
5    if (t == Tail)
6      if (x == NULL)
7        if (CAS (t.next,NULL,n))
8          CAS (Tail,t,n)
9          exit
10     else
11       CAS (Tail,t,x)
```



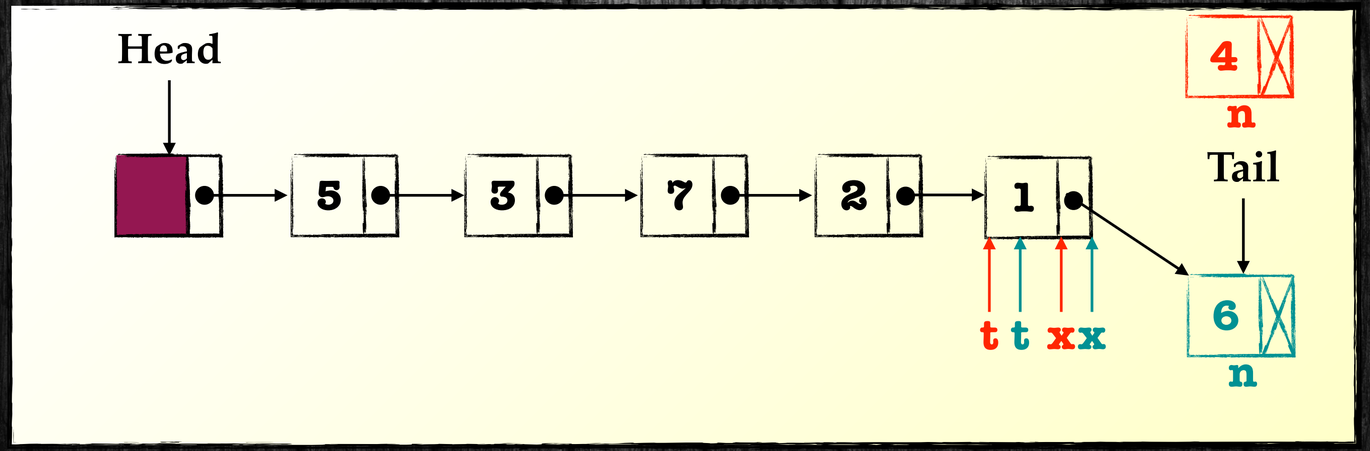
The Michael-Scott Queue Algorithm

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)
```



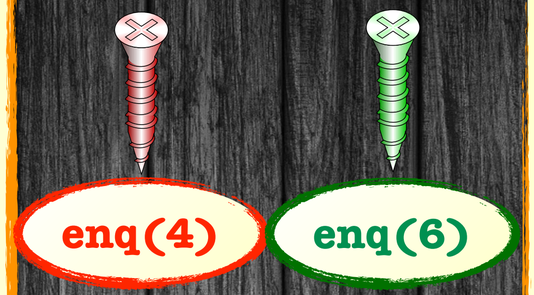

```
enq(k):
Node t,x
1  n = new Node(k,NULL)
2  while (true)
3    t = Tail
4    x = t.next
5    if (t == Tail)
6      if (x == NULL)
7        if (CAS (t.next,NULL,n))
8          CAS (Tail,t,n)
9          exit
10     else
11       CAS (Tail,t,x)
```



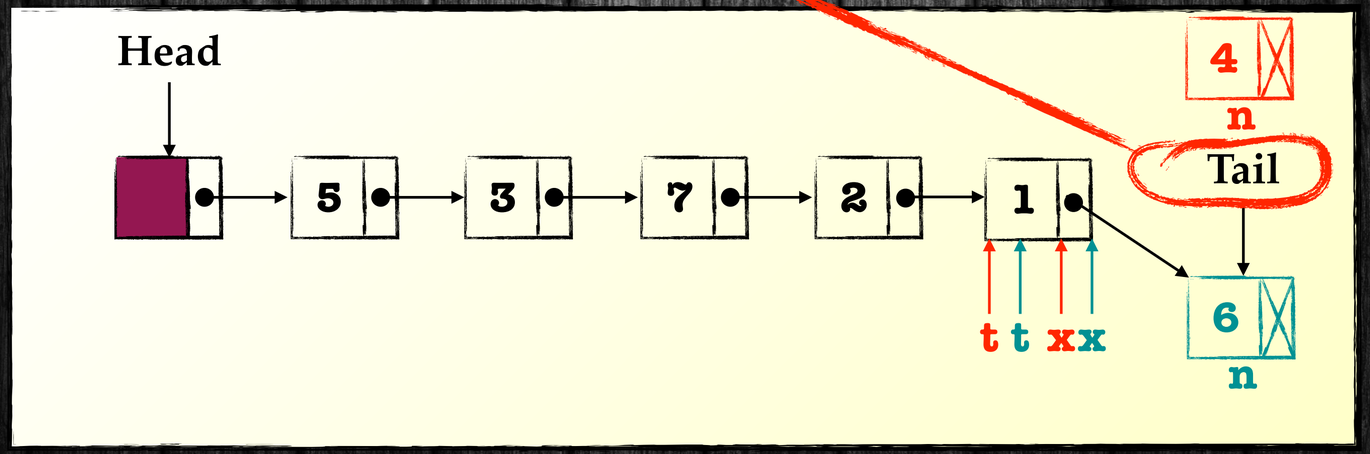
The Michael-Scott Queue Algorithm

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)
```



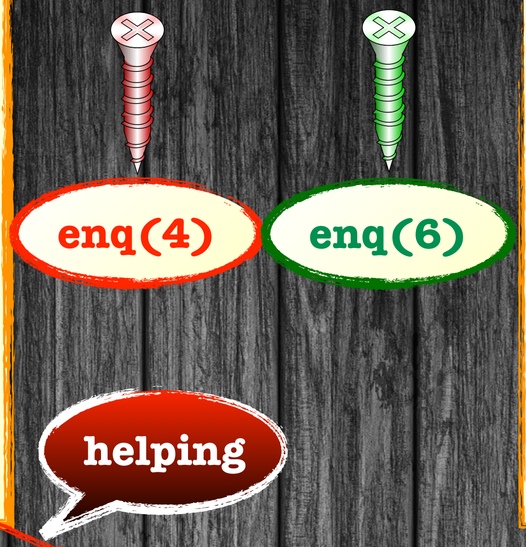

```
enq(k):
Node t,x
1  n = new Node(k,NULL)
2  while (true)
3    t = Tail
4    x = t.next
5    if (t == Tail)
6      if (x == NULL)
7        if (CAS (t.next,NULL,n))
8          CAS (Tail,t,n)
9          exit
10     else
11       CAS (Tail,t,x)
```



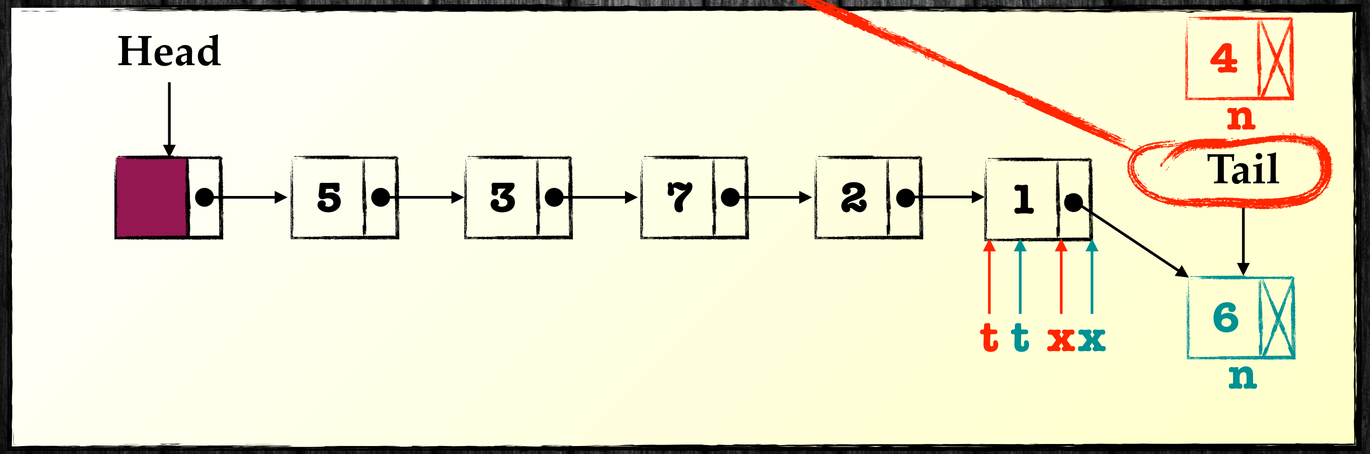
The Michael-Scott Queue Algorithm

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)
```



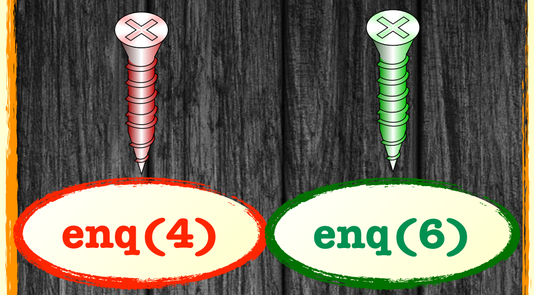

```
enq(k):
Node t,x
1  n = new Node(k,NULL)
2  while (true)
3    t = Tail
4    x = t.next
5    if (t == Tail)
6      if (x == NULL)
7        if (CAS (t.next,NULL,n))
8          CAS (Tail,t,n)
9          exit
10     else
11       CAS (Tail,t,x)
```



The Michael-Scott Queue Algorithm

Concurrent enqueue operations

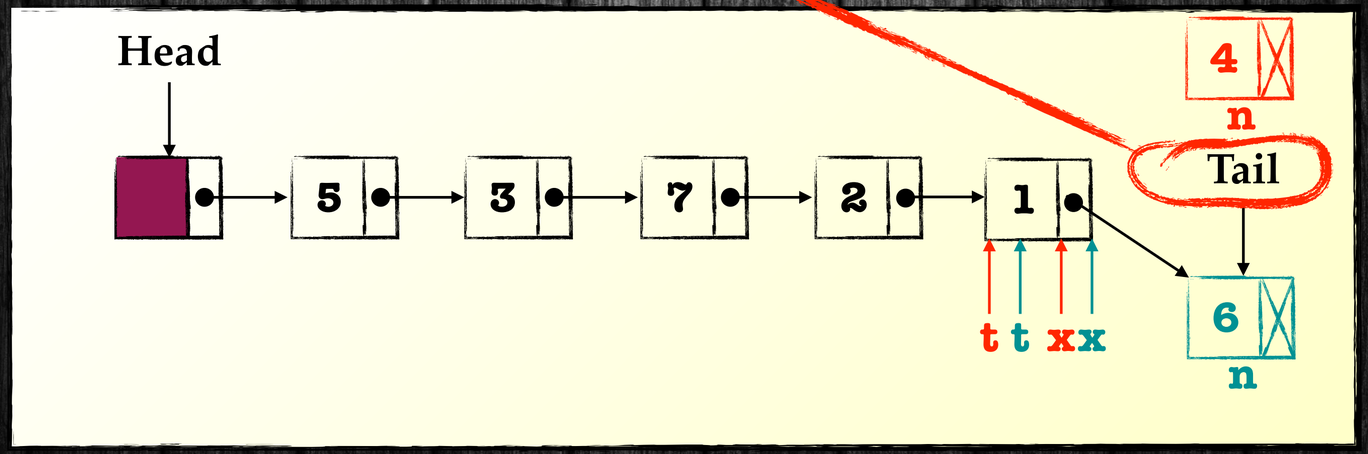
```
enq(k):
Node t,x
1  n = new Node(k,NULL)
2  while (true)
3    t = Tail
4    x = t.next
5    if (t == Tail)
6      if (x == NULL)
7        if (CAS (t.next,NULL,n))
8          CAS (Tail,t,n)
9          exit
10     else
11       CAS (Tail,t,x)
```




```
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

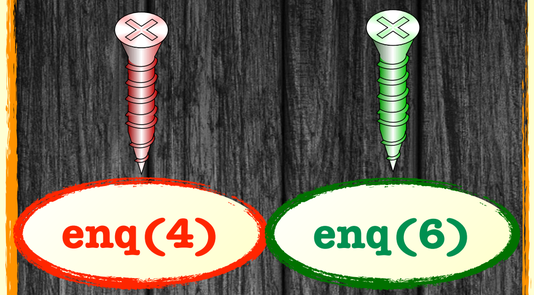
one thread helps another



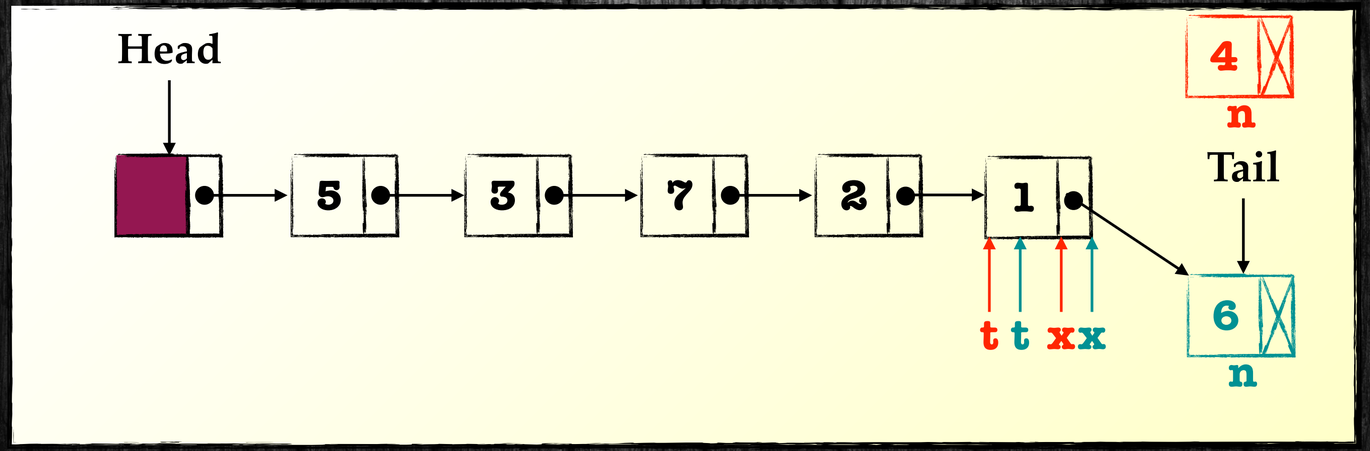
The Michael-Scott Queue Algorithm

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)
```



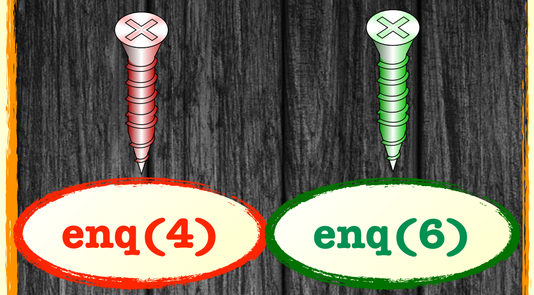

```
enq(k):
Node t,x
1  n = new Node(k,NULL)
2  while (true)
3    t = Tail
4    x = t.next
5    if (t == Tail)
6      if (x == NULL)
7        if (CAS (t.next,NULL,n))
8          CAS (Tail,t,n)
9          exit
10     else
11       CAS (Tail,t,x)
```



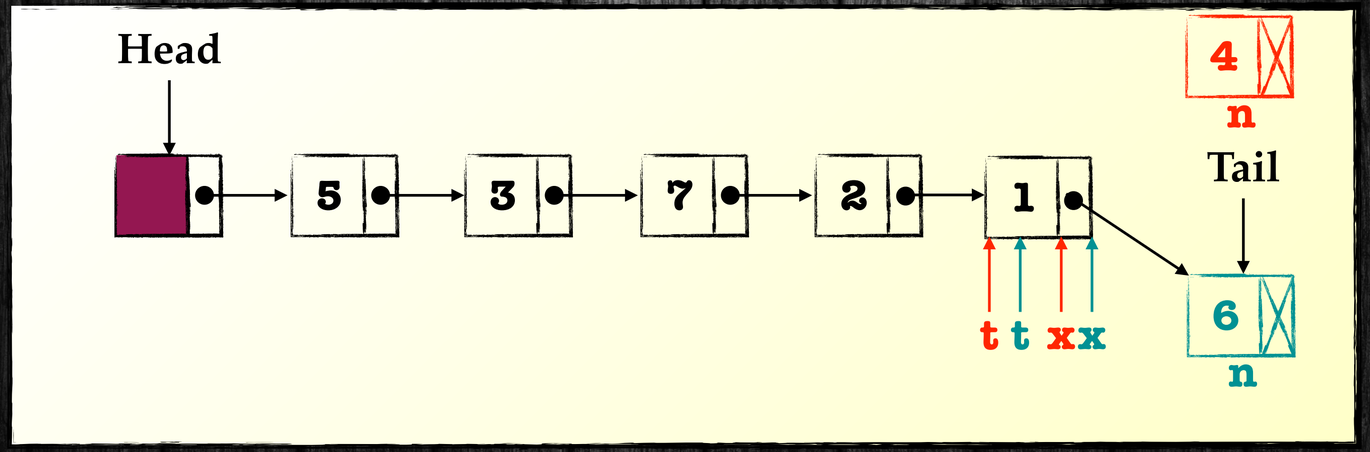
The Michael-Scott Queue Algorithm

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)
```



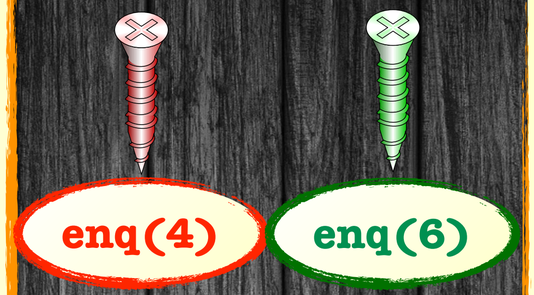

```
enq(k):
Node t,x
1  n = new Node(k,NULL)
2  while (true)
3    t = Tail
4    x = t.next
5    if (t == Tail)
6      if (x == NULL)
7        if (CAS (t.next,NULL,n))
8          CAS (Tail,t,n)
9          exit
10     else
11       CAS (Tail,t,x)
```



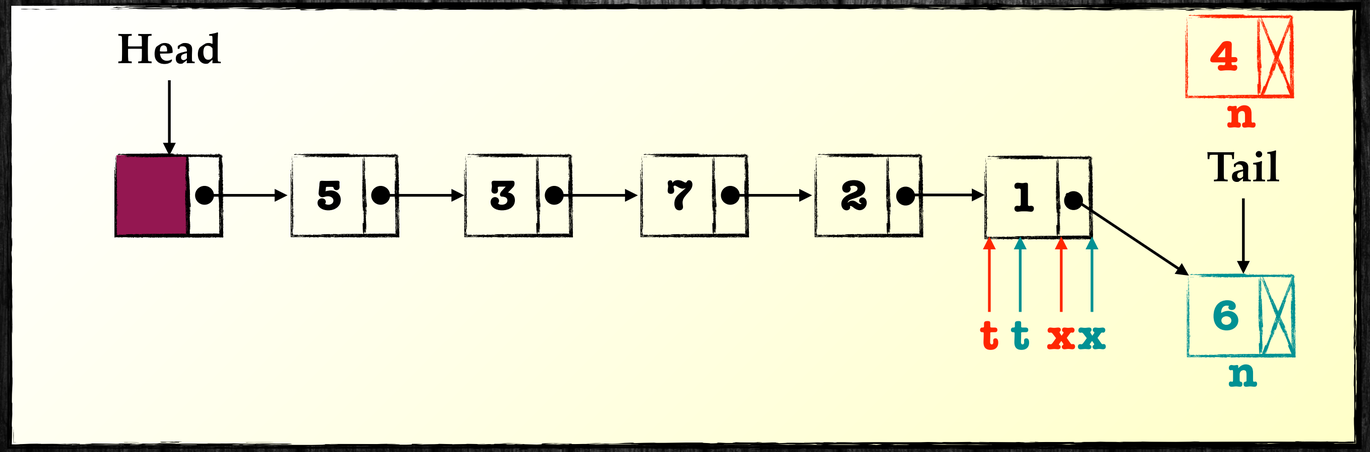
The Michael-Scott Queue Algorithm

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)
```



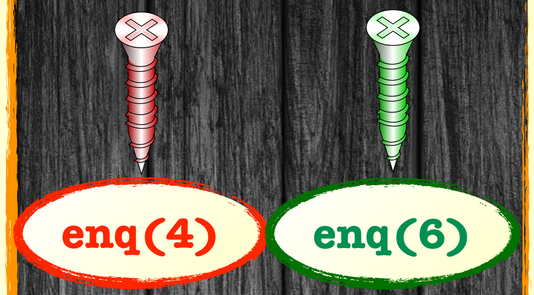

```
enq(k):
Node t,x
1  n = new Node(k,NULL)
2  while (true)
3    t = Tail
4    x = t.next
5    if (t == Tail)
6      if (x == NULL)
7        if (CAS (t.next,NULL,n))
8          CAS (Tail,t,n)
9          exit
10     else
11       CAS (Tail,t,x)
```



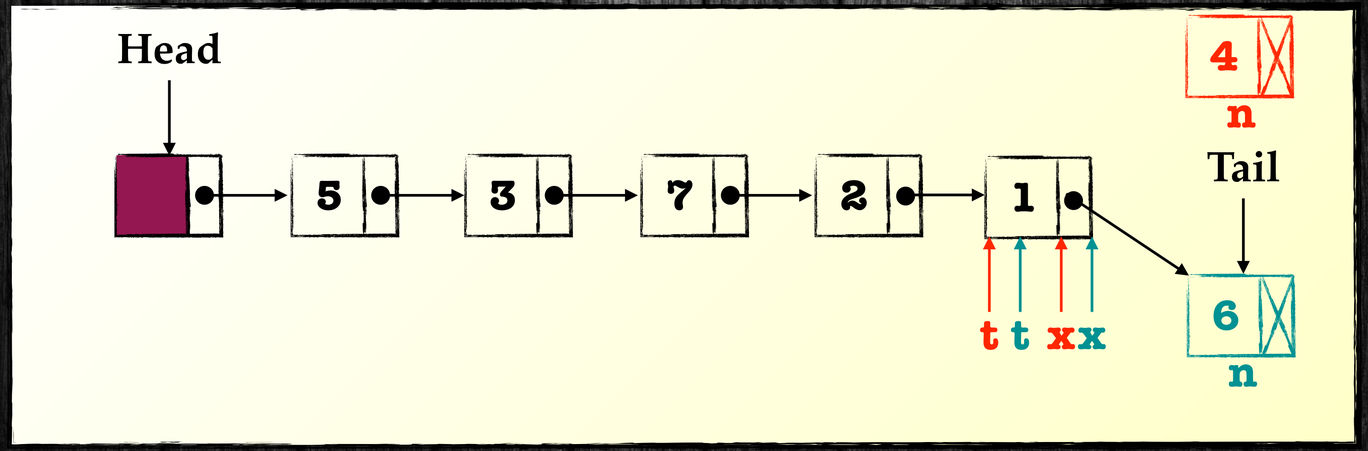
The Michael-Scott Queue Algorithm

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)
```

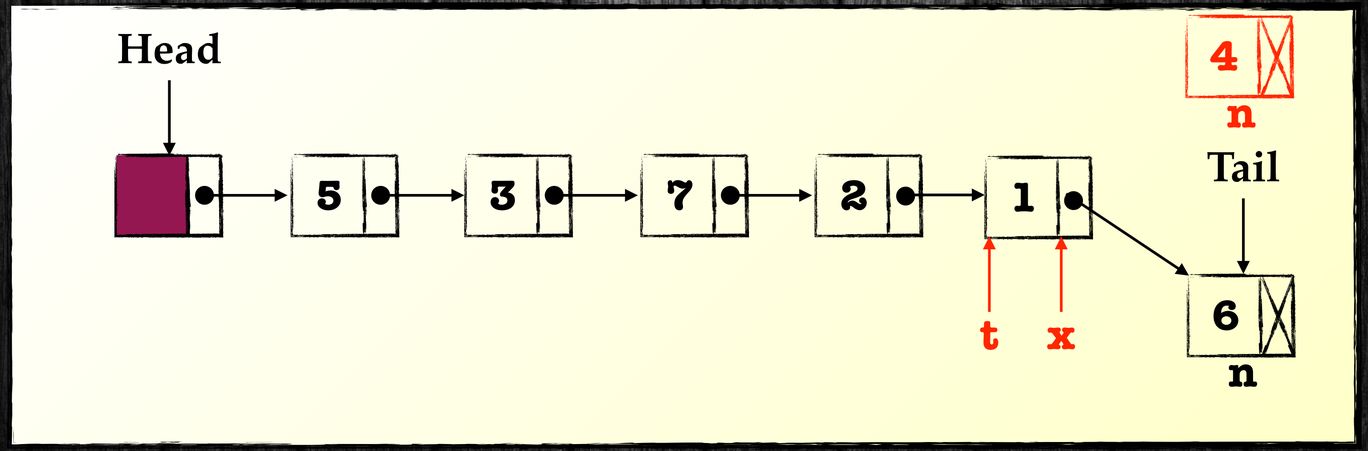


The Michael-Scott Queue Algorithm

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)
```

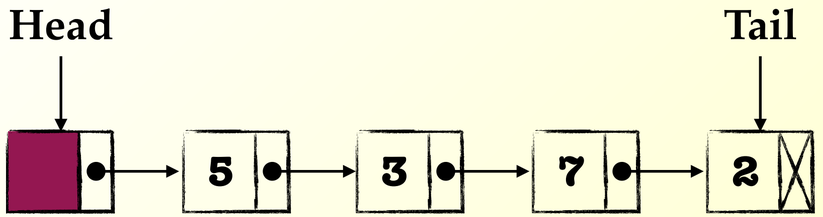




The Michael-Scott Queue Algorithm

```
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
```

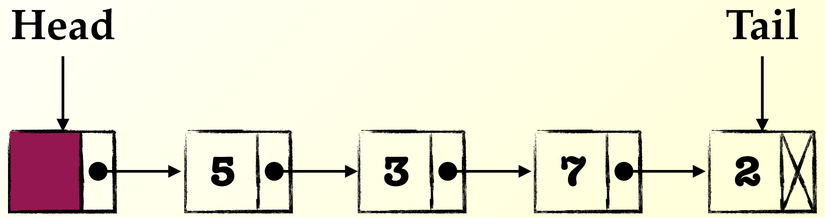




The Michael-Scott Queue Algorithm

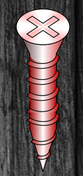
```
deq(k):
Node h,t,x
Integer v
1  while (true)
2    h = Head
3    t = Tail
4    x = h.next
5    if (h == Head)
6      if (h == t)
7        if (x == NULL)
8          return *
9          exit
10       CAS (Tail,t,x)
11     else
12       v = x.key
13       if (CAS (Head,h,x))
14         return v
15  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





The Michael-Scott Queue Algorithm

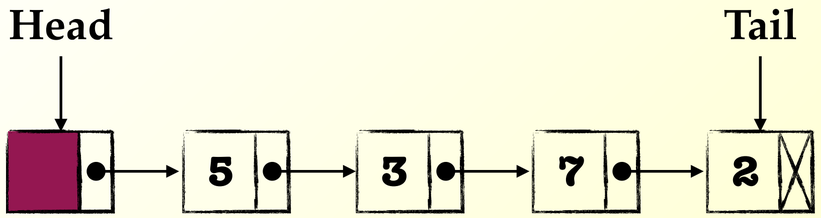
```
deq(k):
Node h,t,x
Integer v
1  while (true)
2    h = Head
3    t = Tail
4    x = h.next
5    if (h == Head)
6      if (h == t)
7        if (x == NULL)
8          return *
9          exit
10       CAS (Tail,t,x)
11     else
12       v = x.key
13       if (CAS (Head,h,x))
14         return v
15  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



deq

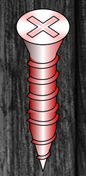
queue

5 3 7 2



The Michael-Scott Queue Algorithm

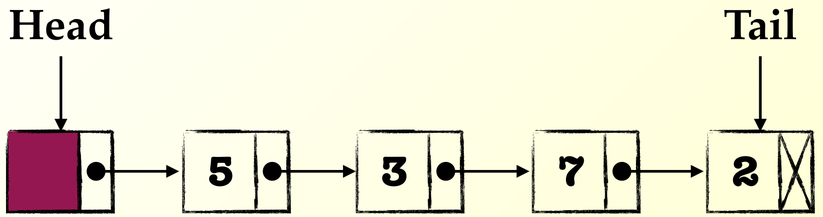
```
deq(k):
Node h,t,x
Integer v
1  while (true)
2    h = Head
3    t = Tail
4    x = h.next
5    if (h == Head)
6      if (h == t)
7        if (x == NULL)
8          return *
9          exit
10       CAS (Tail,t,x)
11     else
12       v = x.key
13       if (CAS (Head,h,x))
14         return v
15  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



deq

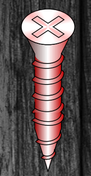
queue

5 3 7 2



The Michael-Scott Queue Algorithm

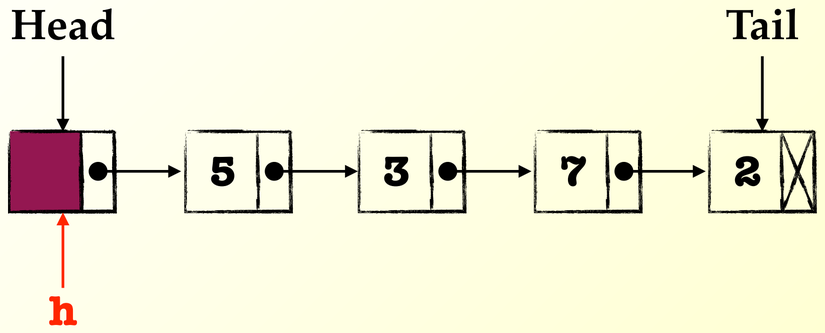
```
deq(k):
Node h,t,x
Integer v
1  while (true)
2    h = Head
3    t = Tail
4    x = h.next
5    if (h == Head)
6      if (h == t)
7        if (x == NULL)
8          return *
9          exit
10       CAS (Tail,t,x)
11     else
12       v = x.key
13       if (CAS (Head,h,x))
14         return v
15  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



deq

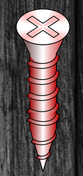
queue

5 3 7 2



The Michael-Scott Queue Algorithm

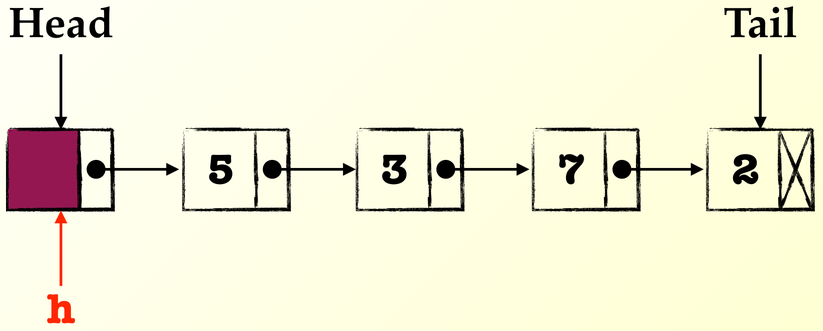
```
deq(k):
Node h,t,x
Integer v
1  while (true)
2    h = Head
3    t = Tail
4    x = h.next
5    if (h == Head)
6      if (h == t)
7        if (x == NULL)
8          return *
9          exit
10       CAS (Tail,t,x)
11     else
12       v = x.key
13       if (CAS (Head,h,x))
14         return v
15  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



deq

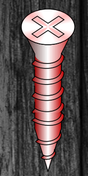
queue

5 3 7 2



The Michael-Scott Queue Algorithm

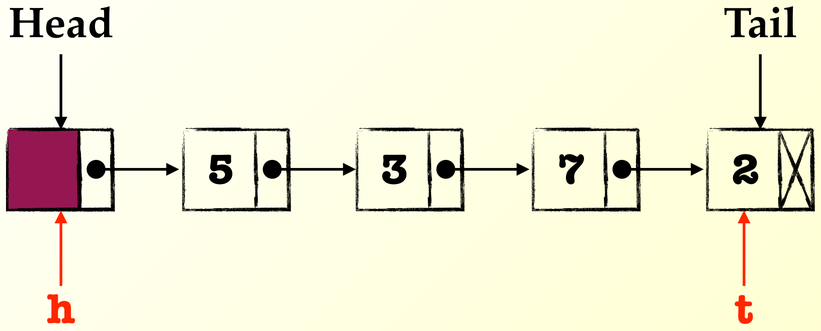
```
deq(k):
Node h,t,x
Integer v
1  while (true)
2    h = Head
3    t = Tail
4    x = h.next
5    if (h == Head)
6      if (h == t)
7        if (x == NULL)
8          return *
9          exit
10       CAS (Tail,t,x)
11     else
12       v = x.key
13       if (CAS (Head,h,x))
14         return v
15  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



deq

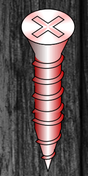
queue

5 3 7 2



The Michael-Scott Queue Algorithm

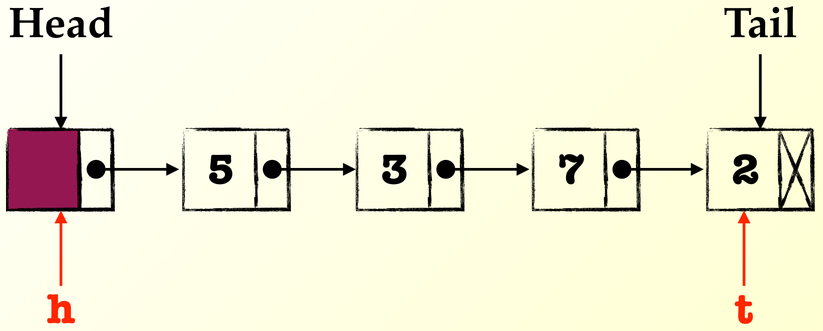
```
deq(k):
Node h,t,x
Integer v
1  while (true)
2    h = Head
3    t = Tail
4    x = h.next
5    if (h == Head)
6      if (h == t)
7        if (x == NULL)
8          return *
9          exit
10       CAS (Tail,t,x)
11     else
12       v = x.key
13       if (CAS (Head,h,x))
14         return v
15  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



deq

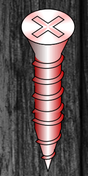
queue

5 3 7 2



The Michael-Scott Queue Algorithm

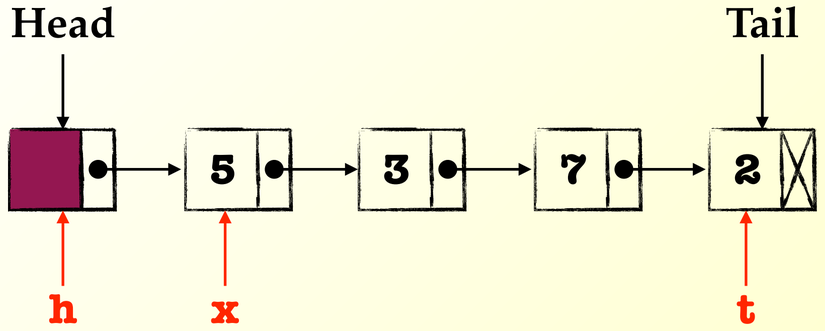
```
deq(k):
Node h,t,x
Integer v
1  while (true)
2    h = Head
3    t = Tail
4    x = h.next
5    if (h == Head)
6      if (h == t)
7        if (x == NULL)
8          return *
9          exit
10       CAS (Tail,t,x)
11     else
12       v = x.key
13       if (CAS (Head,h,x))
14         return v
15  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



deq

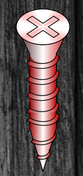
queue

5 3 7 2



The Michael-Scott Queue Algorithm

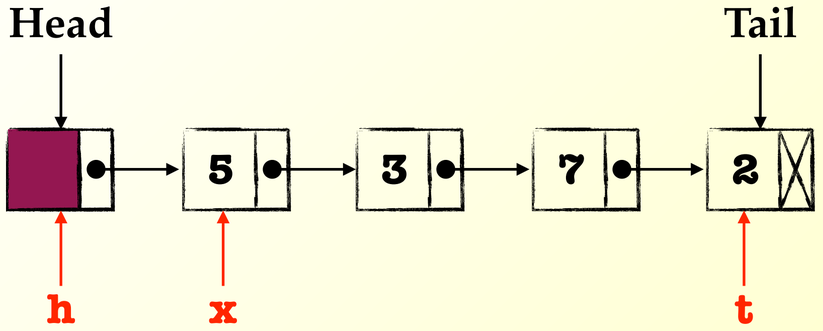
```
deq(k):
Node h,t,x
Integer v
1  while (true)
2    h = Head
3    t = Tail
4    x = h.next
5    if (h == Head)
6      if (h == t)
7        if (x == NULL)
8          return *
9          exit
10       CAS (Tail,t,x)
11     else
12       v = x.key
13       if (CAS (Head,h,x))
14         return v
15  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



deq

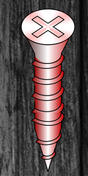
queue

5 3 7 2



The Michael-Scott Queue Algorithm

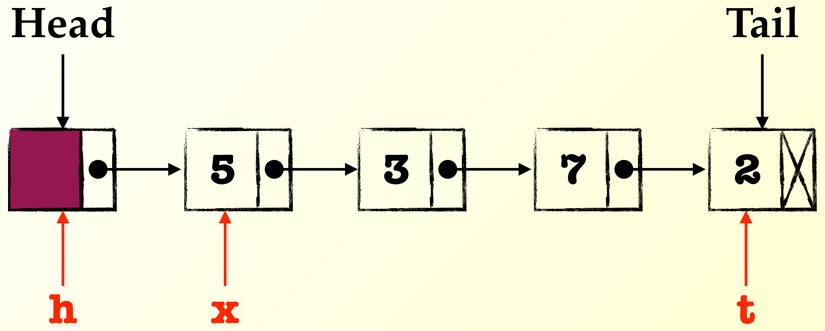
```
deq(k):
Node h,t,x
Integer v
1  while (true)
2    h = Head
3    t = Tail
4    x = h.next
5    if (h == Head)
6      if (h == t)
7        if (x == NULL)
8          return *
9          exit
10       CAS (Tail,t,x)
11     else
12       v = x.key
13       if (CAS (Head,h,x))
14         return v
15  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



deq

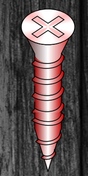
queue

5 3 7 2



The Michael-Scott Queue Algorithm

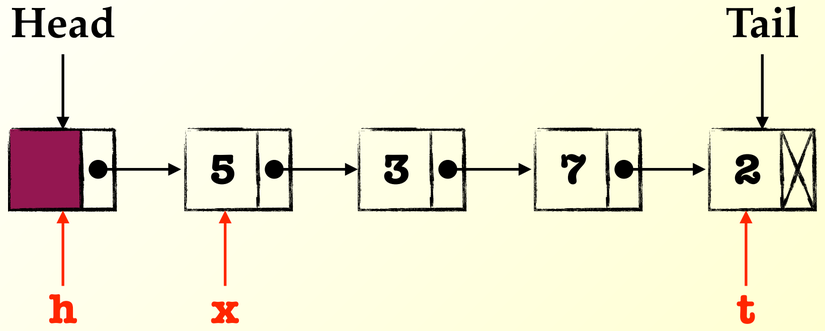
```
deq(k):
Node h,t,x
Integer v
1  while (true)
2    h = Head
3    t = Tail
4    x = h.next
5    if (h == Head)
6      if (h == t)
7        if (x == NULL)
8          return *
9          exit
10       CAS (Tail,t,x)
11     else
12       v = x.key
13       if (CAS (Head,h,x))
14         return v
15  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



deq

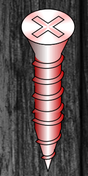
queue

5 3 7 2



The Michael-Scott Queue Algorithm

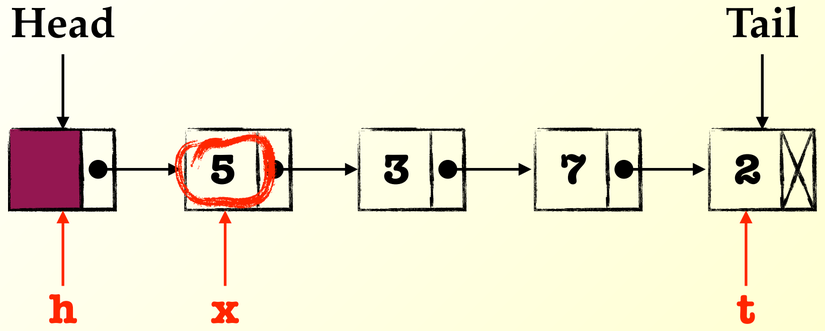
```
deq(k):
Node h,t,x
Integer v
1  while (true)
2    h = Head
3    t = Tail
4    x = h.next
5    if (h == Head)
6      if (h == t)
7        if (x == NULL)
8          return *
9          exit
10       CAS (Tail,t,x)
11     else
12       v = x.key
13       if (CAS (Head,h,x))
14         return v
15  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



deq

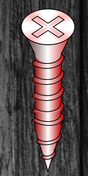
queue

5 3 7 2



The Michael-Scott Queue Algorithm

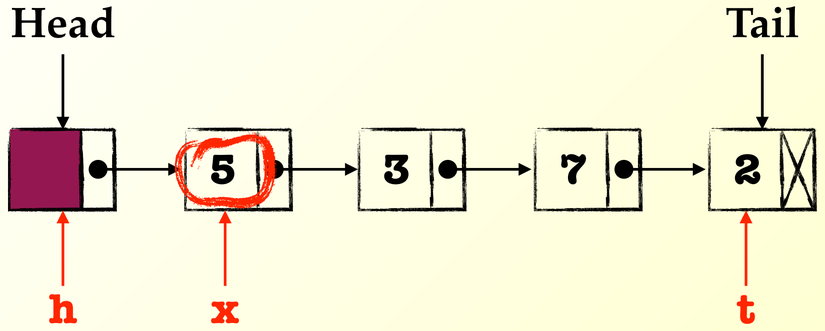
```
deq(k):
Node h,t,x
Integer v
1  while (true)
2    h = Head
3    t = Tail
4    x = h.next
5    if (h == Head)
6      if (h == t)
7        if (x == NULL)
8          return *
9          exit
10       CAS (Tail,t,x)
11     else
12       v = x.key
13       if (CAS (Head,h,x))
14         return v
15  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



deq

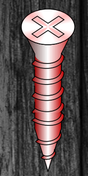
queue

5 3 7 2



The Michael-Scott Queue Algorithm

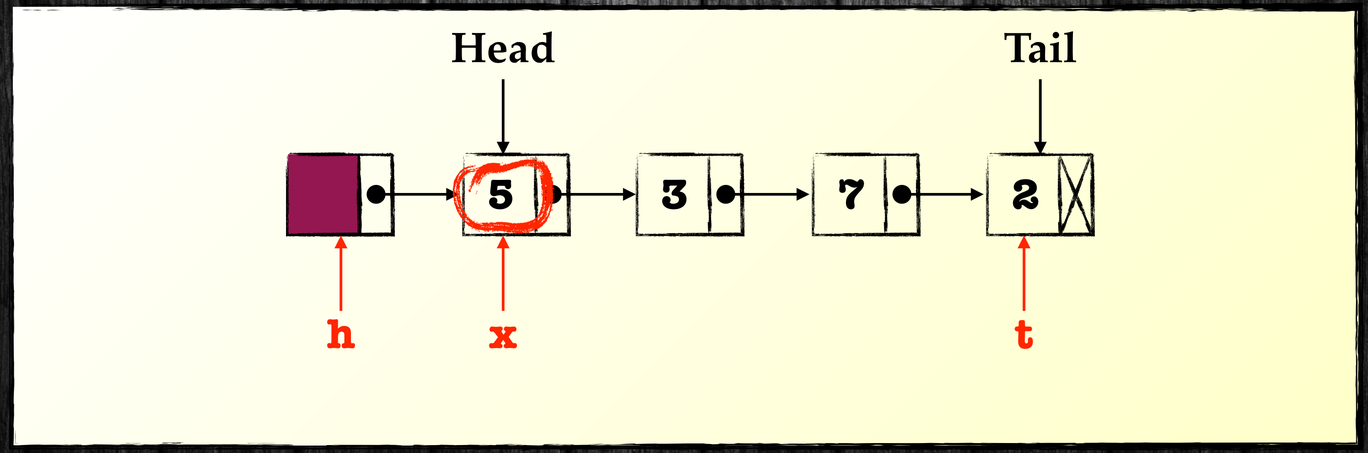
```
deq(k):
Node h,t,x
Integer v
1  while (true)
2    h = Head
3    t = Tail
4    x = h.next
5    if (h == Head)
6      if (h == t)
7        if (x == NULL)
8          return *
9          exit
10       CAS (Tail,t,x)
11     else
12       v = x.key
13       if (CAS (Head,h,x))
14         return v
15  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



deq

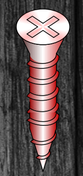
queue

5 3 7 2



The Michael-Scott Queue Algorithm

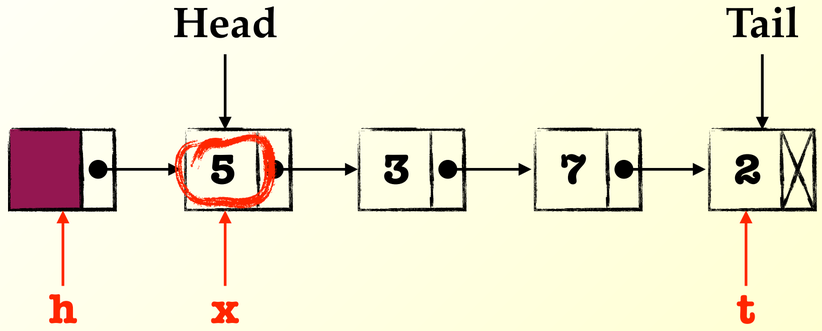
```
deq(k):
Node h,t,x
Integer v
1  while (true)
2    h = Head
3    t = Tail
4    x = h.next
5    if (h == Head)
6      if (h == t)
7        if (x == NULL)
8          return *
9          exit
10       CAS (Tail,t,x)
11     else
12       v = x.key
13       if (CAS (Head,h,x))
14         return v
15  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



deq

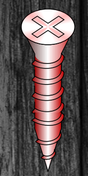
queue

5 3 7 2



The Michael-Scott Queue Algorithm

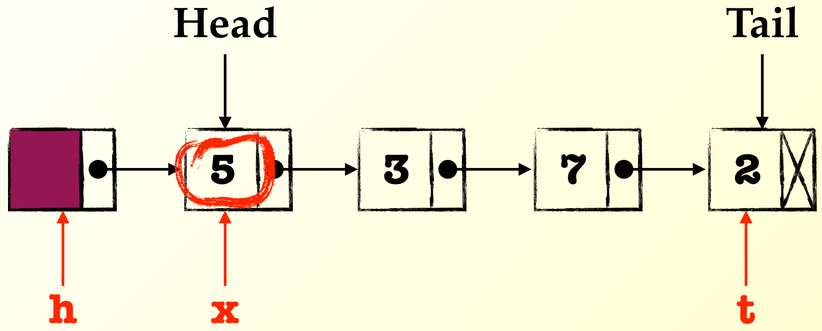
```
deq(k):
Node h,t,x
Integer v
1  while (true)
2    h = Head
3    t = Tail
4    x = h.next
5    if (h == Head)
6      if (h == t)
7        if (x == NULL)
8          return *
9          exit
10       CAS (Tail,t,x)
11     else
12       v = x.key
13       if (CAS (Head,h,x))
14         return v
15  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



deq

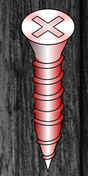
queue

5 3 7 2



The Michael-Scott Queue Algorithm

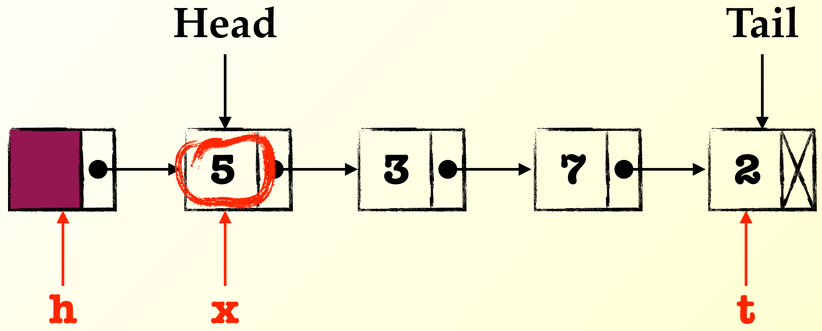
```
deq(k):
Node h,t,x
Integer v
1  while (true)
2    h = Head
3    t = Tail
4    x = h.next
5    if (h == Head)
6      if (h == t)
7        if (x == NULL)
8          return *
9          exit
10       CAS (Tail,t,x)
11     else
12       v = x.key
13       if (CAS (Head,h,x))
14         return v
15  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



deq

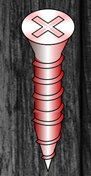
queue

5 3 7 2



The Michael-Scott Queue Algorithm

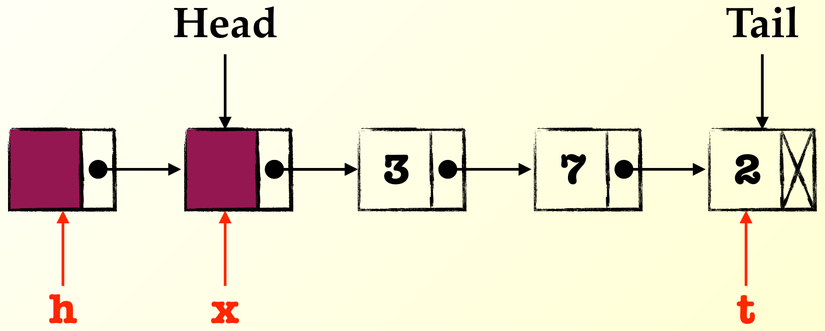
```
deq(k):
Node h,t,x
Integer v
1  while (true)
2    h = Head
3    t = Tail
4    x = h.next
5    if (h == Head)
6      if (h == t)
7        if (x == NULL)
8          return *
9          exit
10       CAS (Tail,t,x)
11     else
12       v = x.key
13       if (CAS (Head,h,x))
14         return v
15  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



deq

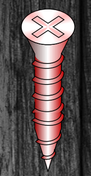
queue

5 3 7 2



The Michael-Scott Queue Algorithm

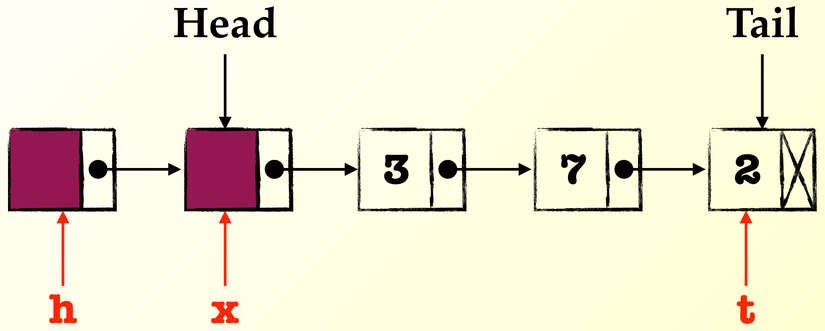
```
deq(k):
Node h,t,x
Integer v
1  while (true)
2    h = Head
3    t = Tail
4    x = h.next
5    if (h == Head)
6      if (h == t)
7        if (x == NULL)
8          return *
9          exit
10       CAS (Tail,t,x)
11     else
12       v = x.key
13       if (CAS (Head,h,x))
14         return v
15  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



deq

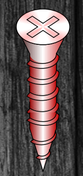
queue

5 3 7 2



The Michael-Scott Queue Algorithm

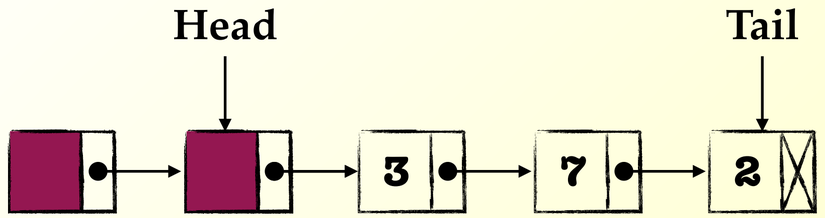
```
deq(k):
Node h,t,x
Integer v
1  while (true)
2    h = Head
3    t = Tail
4    x = h.next
5    if (h == Head)
6      if (h == t)
7        if (x == NULL)
8          return *
9          exit
10       CAS (Tail,t,x)
11     else
12       v = x.key
13       if (CAS (Head,h,x))
14         return v
15  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



deq

queue

5 3 7 2



The Michael-Scott Queue Algorithm

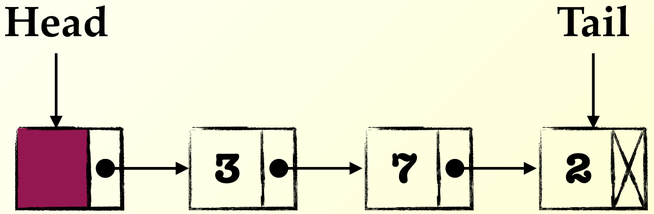
```
deq(k):
Node h,t,x
Integer v
1  while (true)
2    h = Head
3    t = Tail
4    x = h.next
5    if (h == Head)
6      if (h == t)
7        if (x == NULL)
8          return *
9          exit
10       CAS (Tail,t,x)
11     else
12       v = x.key
13       if (CAS (Head,h,x))
14         return v
15  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



deq

queue

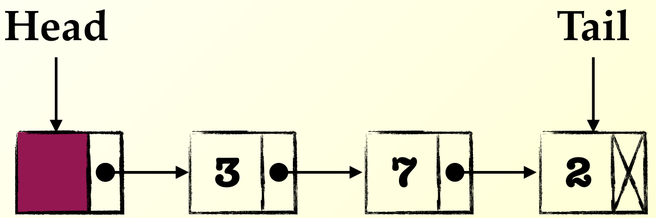
5 3 7 2



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
```

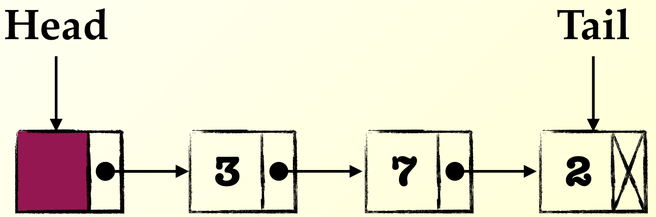




The Michael-Scott Queue Algorithm

```
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
```



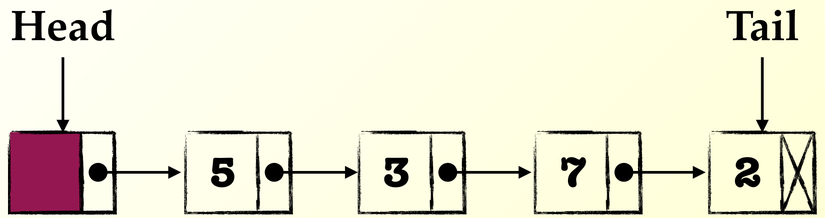


The Michael-Scott Queue Algorithm

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
```



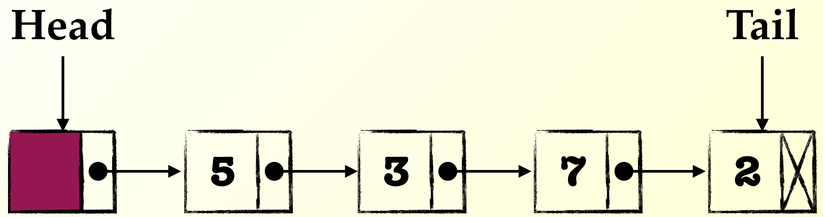
The Michael-Scott Queue Algorithm

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
```



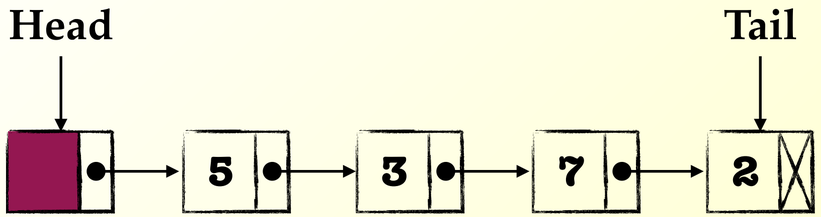
The Michael-Scott Queue Algorithm

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
```



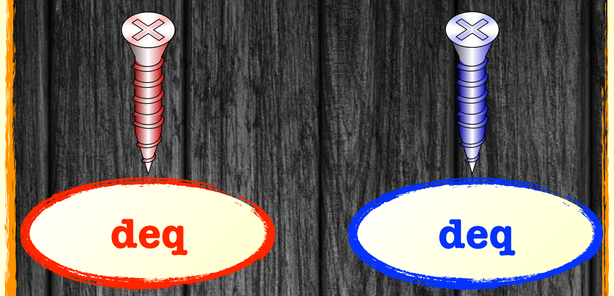

```
deq(k):
Node h,t,x
Integer v
1  while (true)
2    h = Head
3    t = Tail
4    x = h.next
5    if (h == Head)
6      if (h == t)
7        if (x == NULL)
8          return *
9          exit
10       CAS (Tail,t,x)
11     else
12       v = x.key
13       if (CAS (Head,h,x))
14         return v
15  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



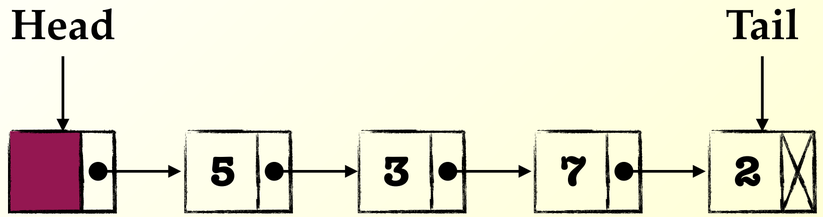
The Michael-Scott Queue Algorithm

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
```



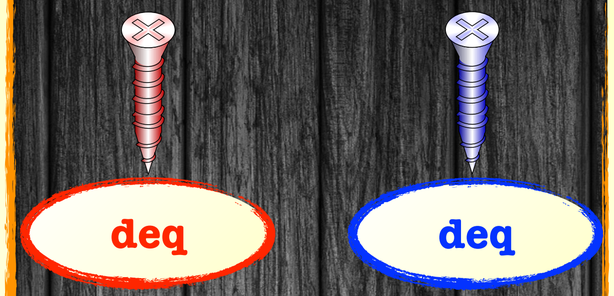

```
deq(k):
Node h,t,x
Integer v
1  while (true)
2    h = Head
3    t = Tail
4    x = h.next
5    if (h == Head)
6      if (h == t)
7        if (x == NULL)
8          return *
9          exit
10       CAS (Tail,t,x)
11     else
12       v = x.key
13       if (CAS (Head,h,x))
14         return v
15  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



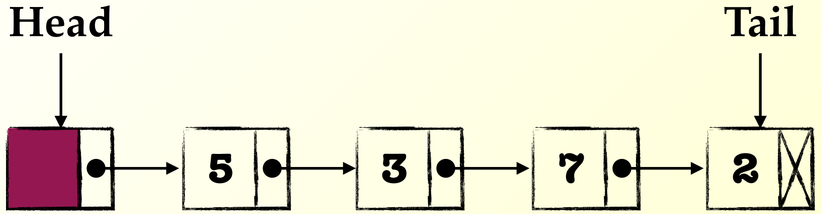
The Michael-Scott Queue Algorithm

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
```



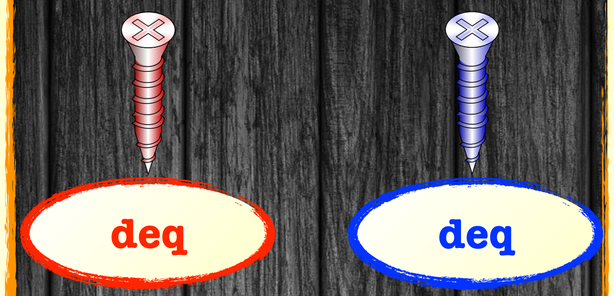

```
deq(k):
Node h,t,x
Integer v
1  while (true)
2    h = Head
3    t = Tail
4    x = h.next
5    if (h == Head)
6      if (h == t)
7        if (x == NULL)
8          return *
9          exit
10       CAS (Tail,t,x)
11     else
12       v = x.key
13       if (CAS (Head,h,x))
14         return v
15  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



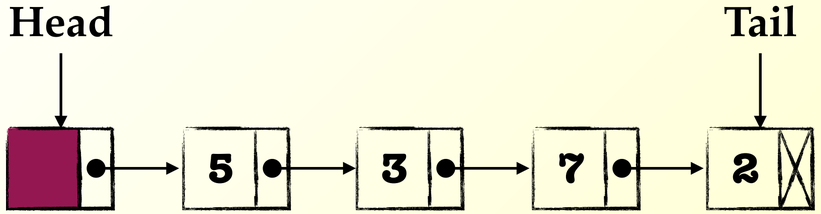
The Michael-Scott Queue Algorithm

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
```



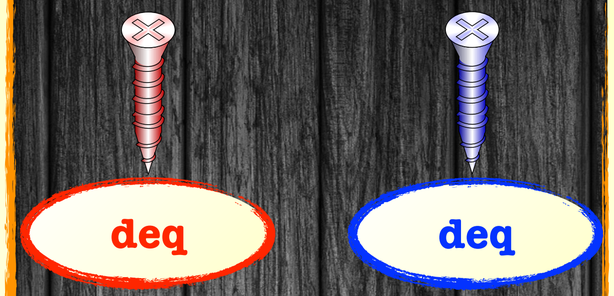

```
deq(k):
Node h,t,x
Integer v
1  while (true)
2    h = Head
3    t = Tail
4    x = h.next
5    if (h == Head)
6      if (h == t)
7        if (x == NULL)
8          return *
9          exit
10       CAS (Tail,t,x)
11     else
12       v = x.key
13       if (CAS (Head,h,x))
14         return v
15  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



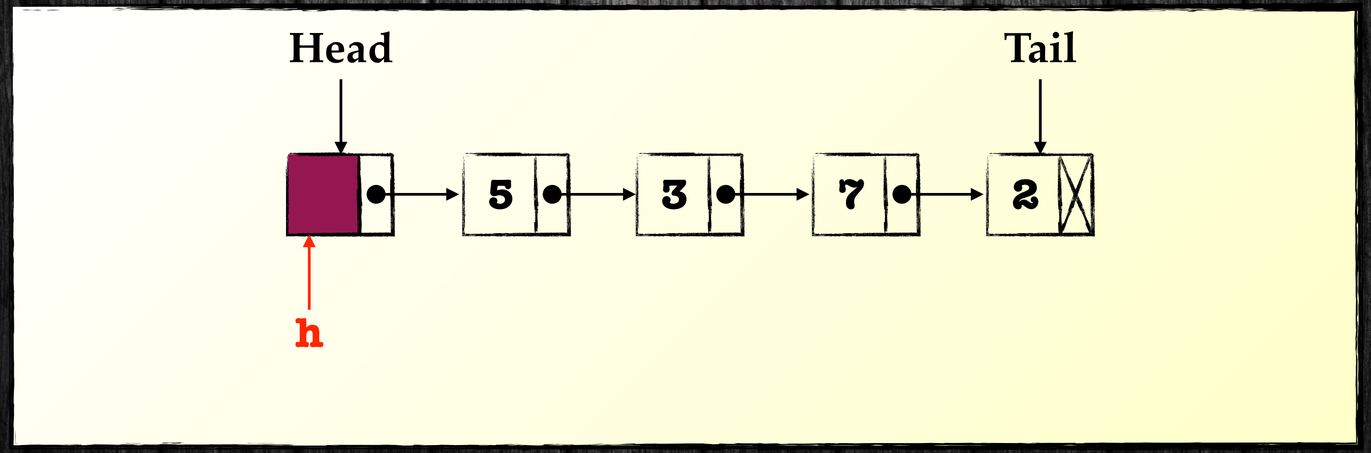
The Michael-Scott Queue Algorithm

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
```



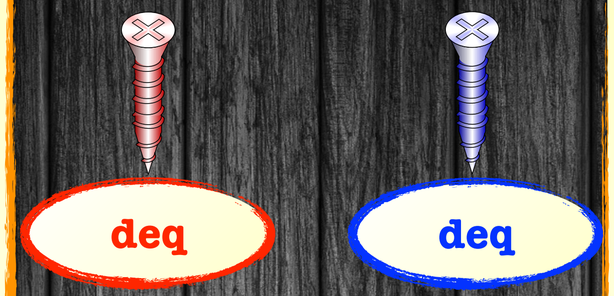

```
deq(k):
Node h,t,x
Integer v
1  while (true)
2    h = Head
3    t = Tail
4    x = h.next
5    if (h == Head)
6      if (h == t)
7        if (x == NULL)
8          return *
9          exit
10       CAS (Tail,t,x)
11     else
12       v = x.key
13       if (CAS (Head,h,x))
14         return v
15  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



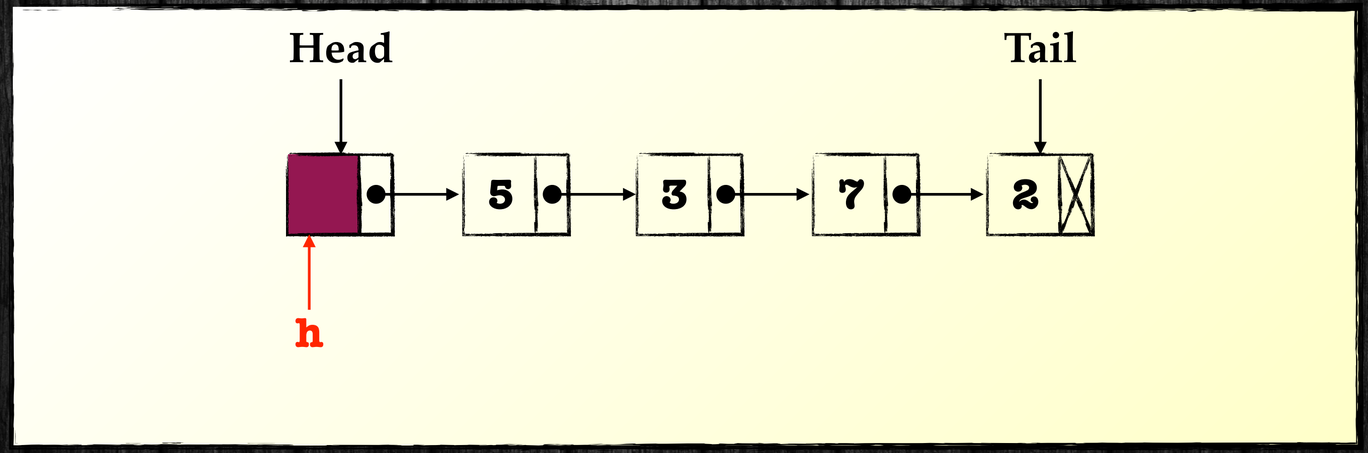
The Michael-Scott Queue Algorithm

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
```



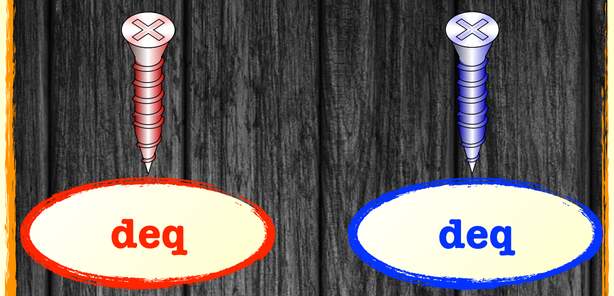

```
deq(k):
Node h,t,x
Integer v
1  while (true)
2    h = Head
3    t = Tail
4    x = h.next
5    if (h == Head)
6      if (h == t)
7        if (x == NULL)
8          return *
9          exit
10       CAS (Tail,t,x)
11     else
12       v = x.key
13       if (CAS (Head,h,x))
14         return v
15  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



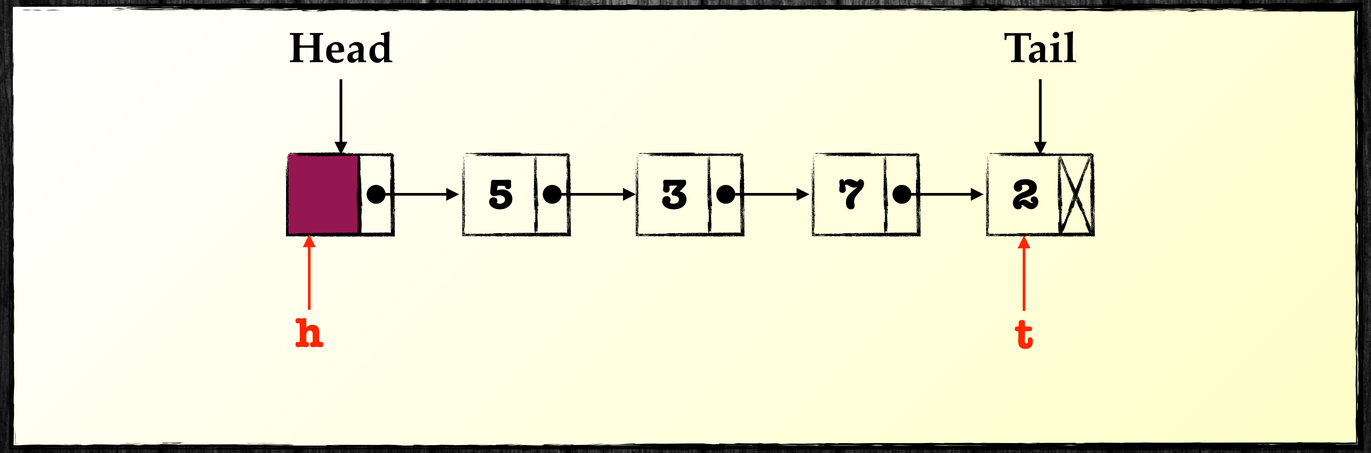
The Michael-Scott Queue Algorithm

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
```



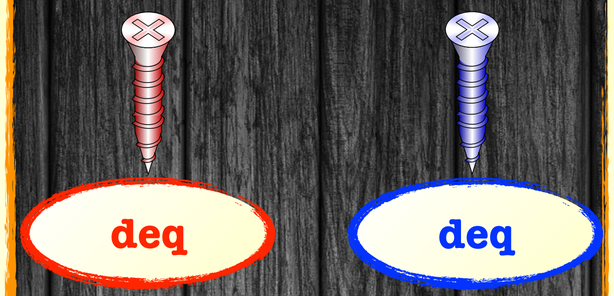

```
deq(k):
Node h,t,x
Integer v
1  while (true)
2    h = Head
3    t = Tail
4    x = h.next
5    if (h == Head)
6      if (h == t)
7        if (x == NULL)
8          return *
9          exit
10       CAS (Tail,t,x)
11     else
12       v = x.key
13       if (CAS (Head,h,x))
14         return v
15  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



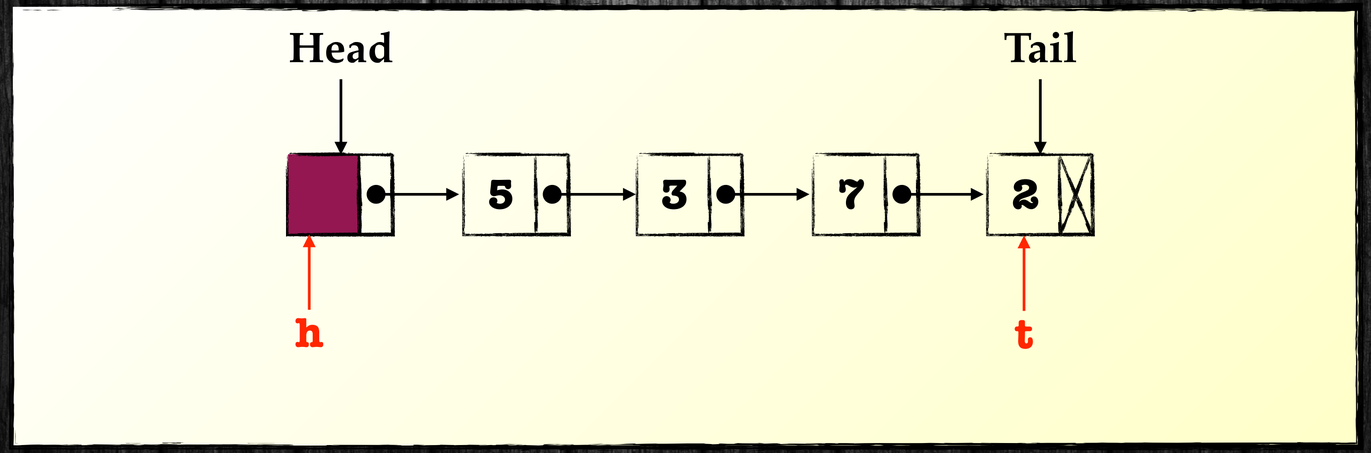
The Michael-Scott Queue Algorithm

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
```



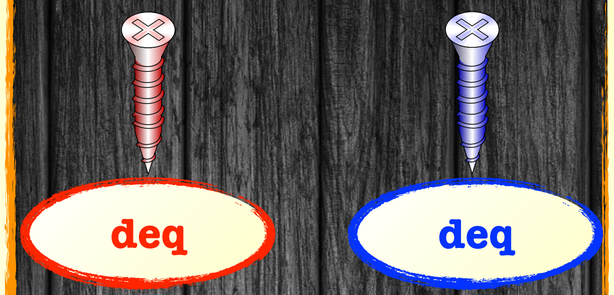

```
deq(k):
Node h,t,x
Integer v
1  while (true)
2    h = Head
3    t = Tail
4    x = h.next
5    if (h == Head)
6      if (h == t)
7        if (x == NULL)
8          return *
9          exit
10       CAS (Tail,t,x)
11     else
12       v = x.key
13       if (CAS (Head,h,x))
14         return v
15  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



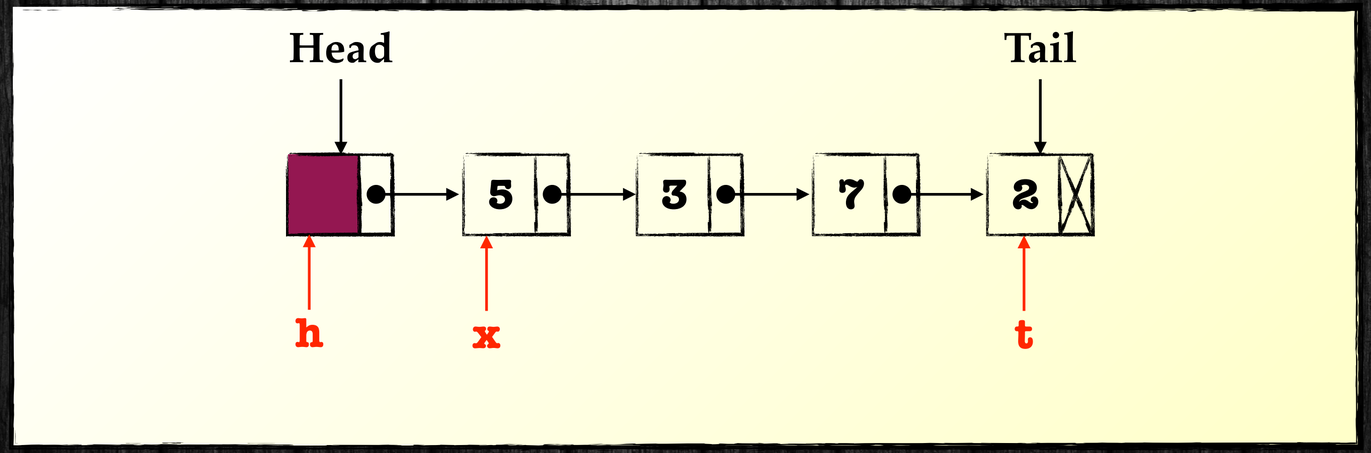
The Michael-Scott Queue Algorithm

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
```



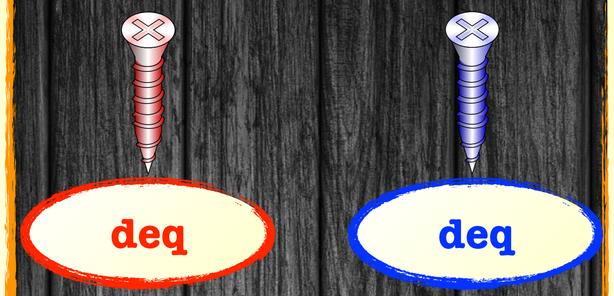

```
deq(k):
Node h,t,x
Integer v
1  while (true)
2    h = Head
3    t = Tail
4    x = h.next
5    if (h == Head)
6      if (h == t)
7        if (x == NULL)
8          return *
9          exit
10       CAS (Tail,t,x)
11     else
12       v = x.key
13       if (CAS (Head,h,x))
14         return v
15  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



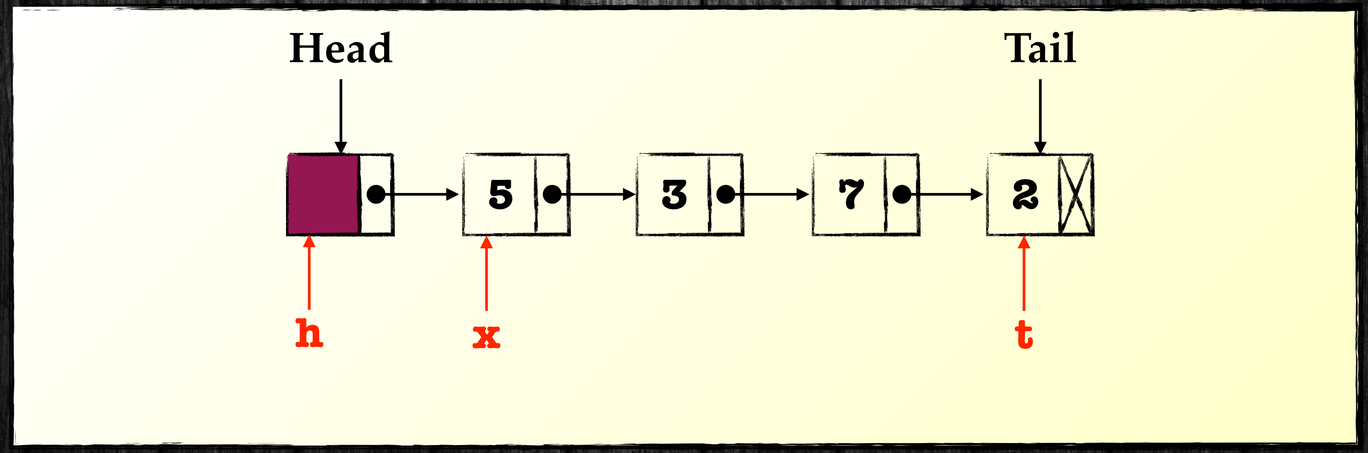
The Michael-Scott Queue Algorithm

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
```



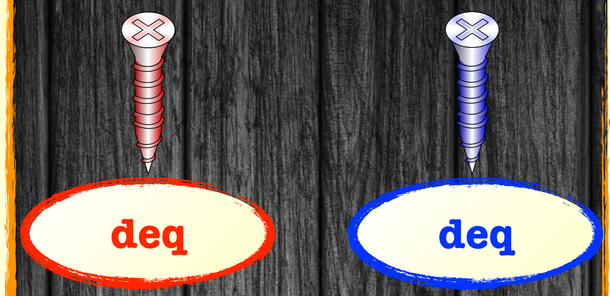

```
deq(k):
Node h,t,x
Integer v
1  while (true)
2    h = Head
3    t = Tail
4    x = h.next
5    if (h == Head)
6      if (h == t)
7        if (x == NULL)
8          return *
9          exit
10       CAS (Tail,t,x)
11     else
12       v = x.key
13       if (CAS (Head,h,x))
14         return v
15  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



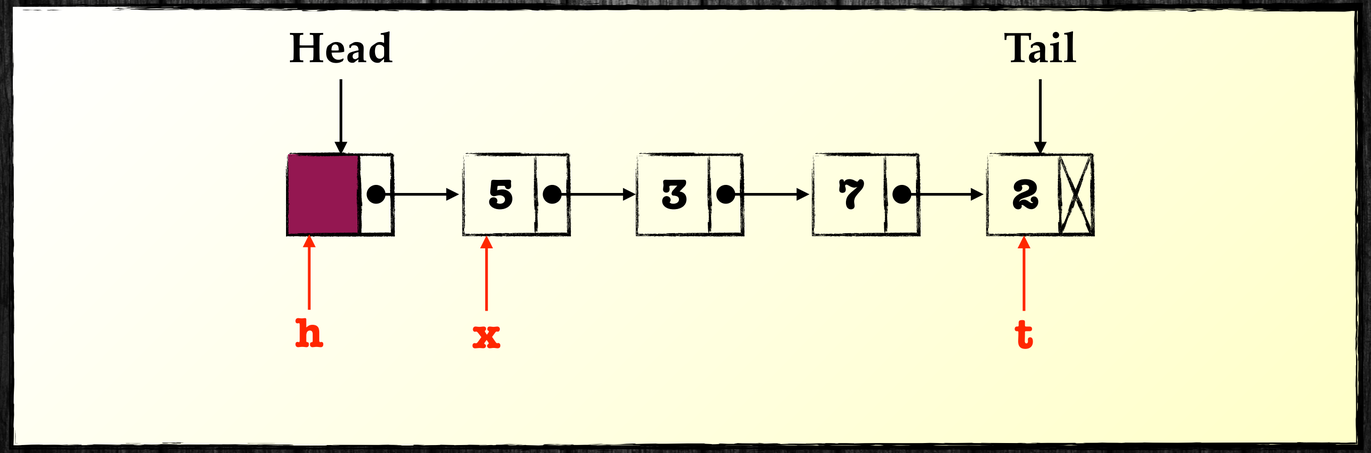
The Michael-Scott Queue Algorithm

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
```



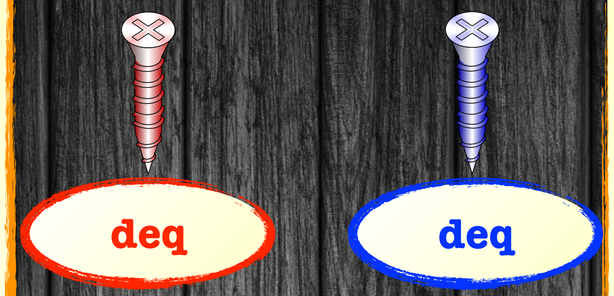

```
deq(k):
Node h,t,x
Integer v
1  while (true)
2    h = Head
3    t = Tail
4    x = h.next
5    if (h == Head)
6      if (h == t)
7        if (x == NULL)
8          return *
9          exit
10       CAS (Tail,t,x)
11     else
12       v = x.key
13       if (CAS (Head,h,x))
14         return v
15  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



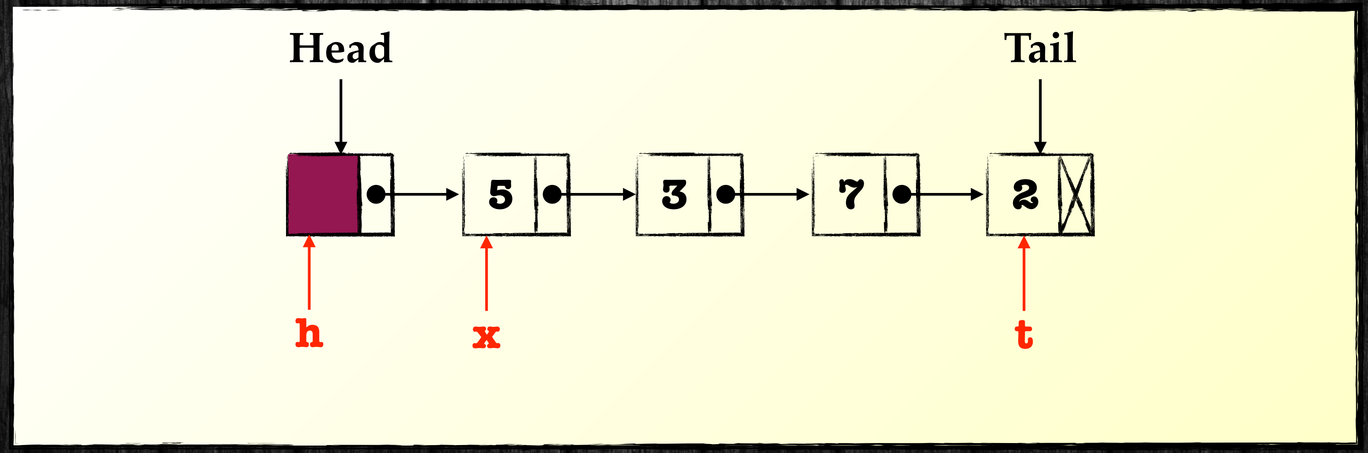
The Michael-Scott Queue Algorithm

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
```



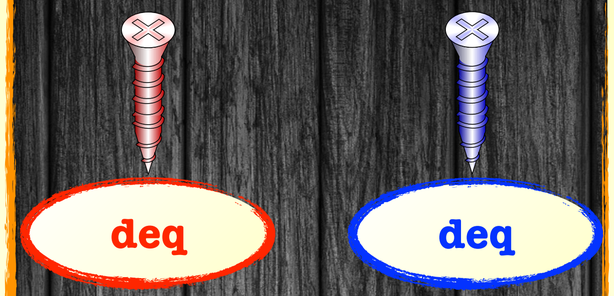

```
deq(k):
Node h,t,x
Integer v
1  while (true)
2    h = Head
3    t = Tail
4    x = h.next
5    if (h == Head)
6      if (h == t)
7        if (x == NULL)
8          return *
9          exit
10       CAS (Tail,t,x)
11     else
12       v = x.key
13       if (CAS (Head,h,x))
14         return v
15  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



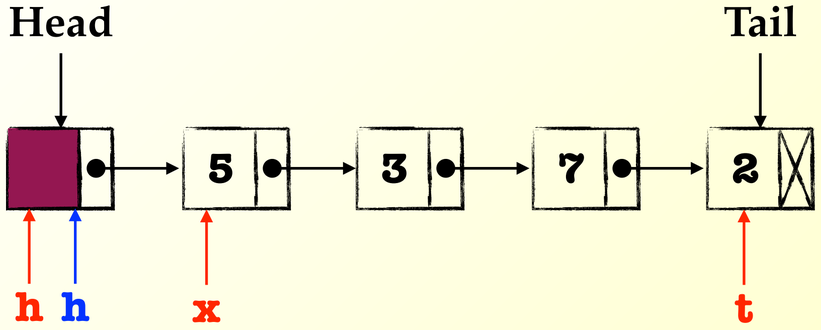
The Michael-Scott Queue Algorithm

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
```



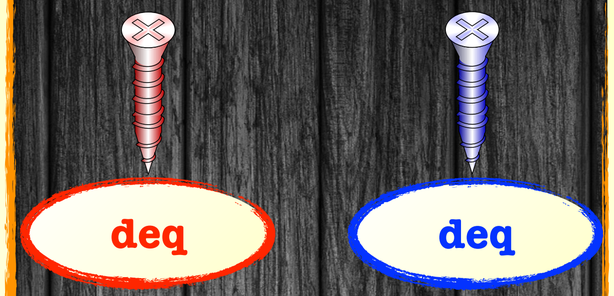

```
deq(k):
Node h,t,x
Integer v
1  while (true)
2    h = Head
3    t = Tail
4    x = h.next
5    if (h == Head)
6      if (h == t)
7        if (x == NULL)
8          return *
9          exit
10       CAS (Tail,t,x)
11     else
12       v = x.key
13       if (CAS (Head,h,x))
14         return v
15  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



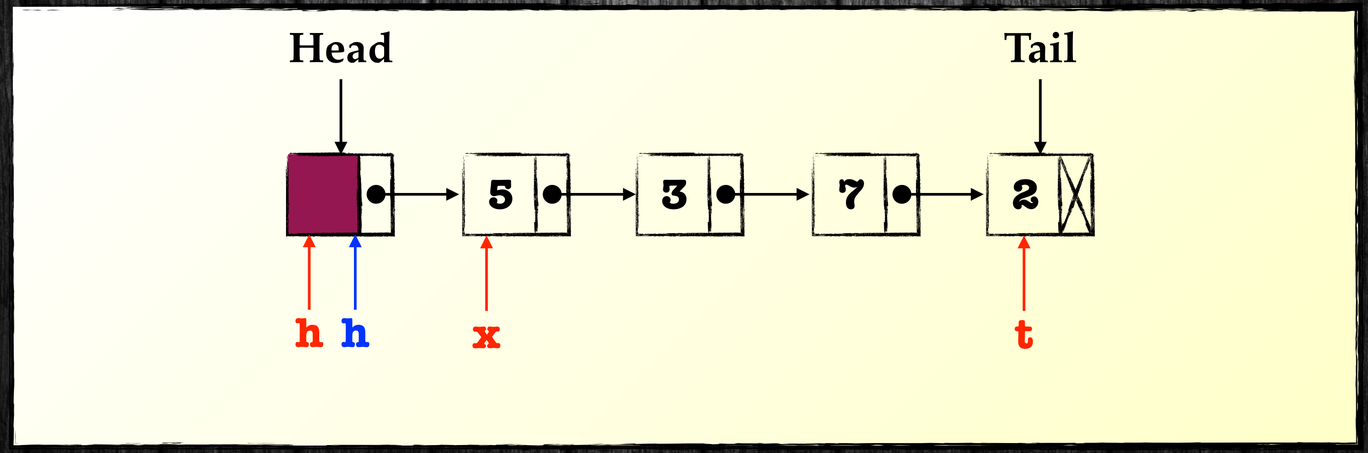
The Michael-Scott Queue Algorithm

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
```



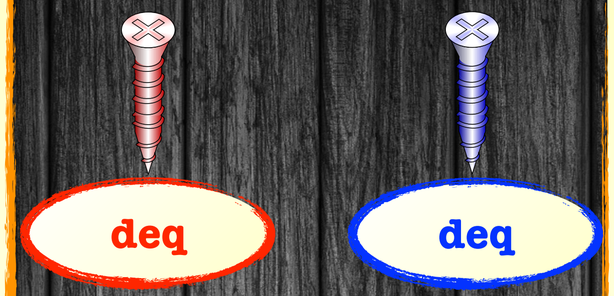

```
deq(k):
Node h,t,x
Integer v
1  while (true)
2    h = Head
3    t = Tail
4    x = h.next
5    if (h == Head)
6      if (h == t)
7        if (x == NULL)
8          return *
9          exit
10       CAS (Tail,t,x)
11     else
12       v = x.key
13       if (CAS (Head,h,x))
14         return v
15  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



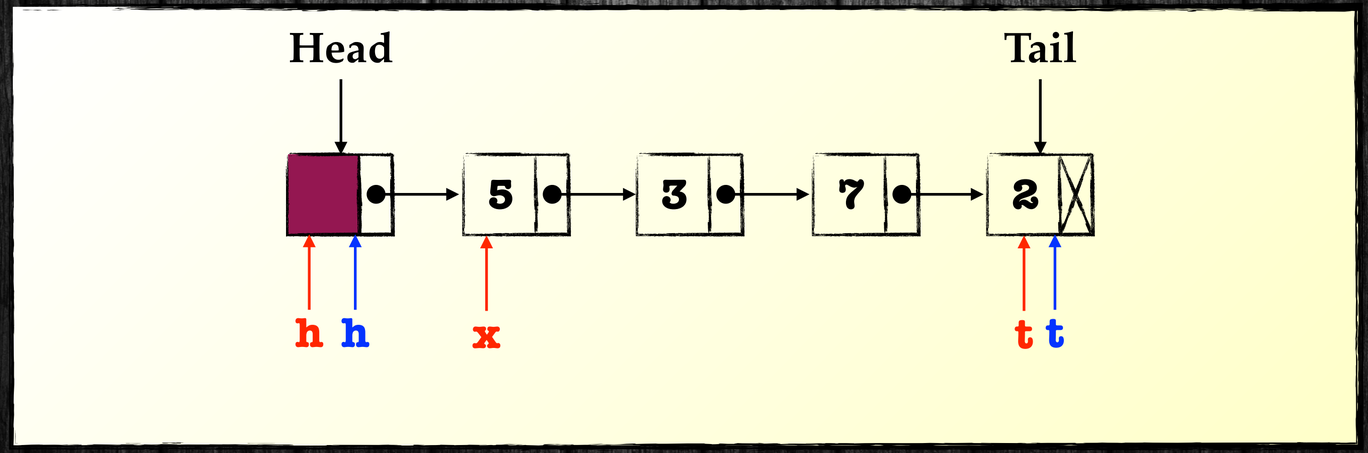
The Michael-Scott Queue Algorithm

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
```



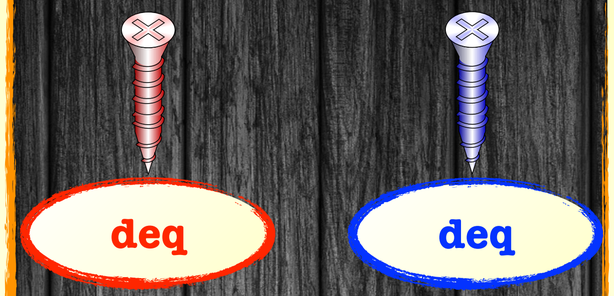

```
deq(k):
Node h,t,x
Integer v
1  while (true)
2    h = Head
3    t = Tail
4    x = h.next
5    if (h == Head)
6      if (h == t)
7        if (x == NULL)
8          return *
9          exit
10       CAS (Tail,t,x)
11     else
12       v = x.key
13       if (CAS (Head,h,x))
14         return v
15  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



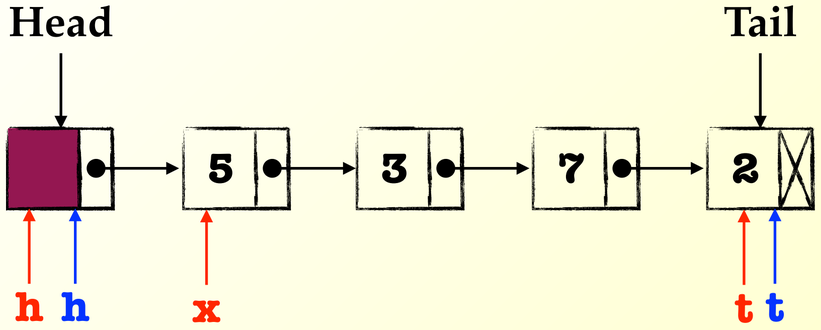
The Michael-Scott Queue Algorithm

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
```



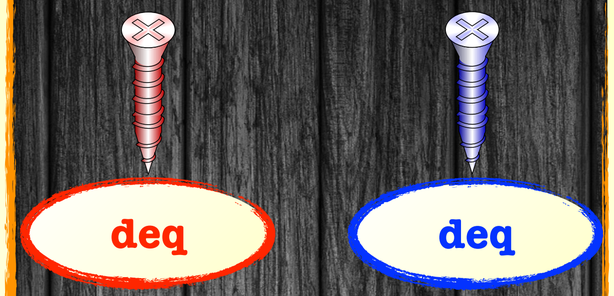

```
deq(k):
Node h,t,x
Integer v
1  while (true)
2    h = Head
3    t = Tail
4    x = h.next
5    if (h == Head)
6      if (h == t)
7        if (x == NULL)
8          return *
9          exit
10       CAS (Tail,t,x)
11     else
12       v = x.key
13       if (CAS (Head,h,x))
14         return v
15  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



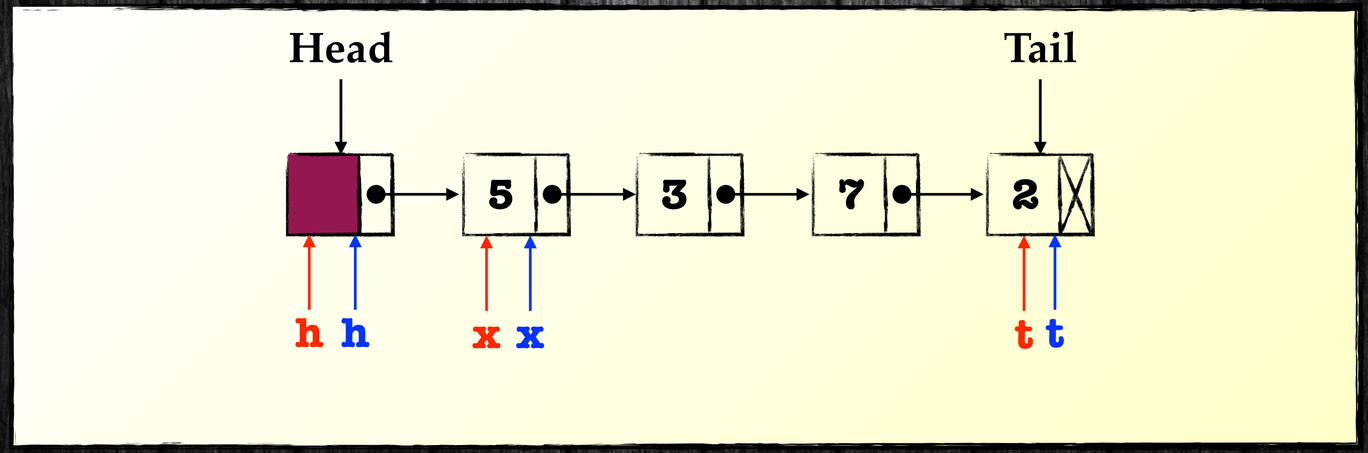
The Michael-Scott Queue Algorithm

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
```



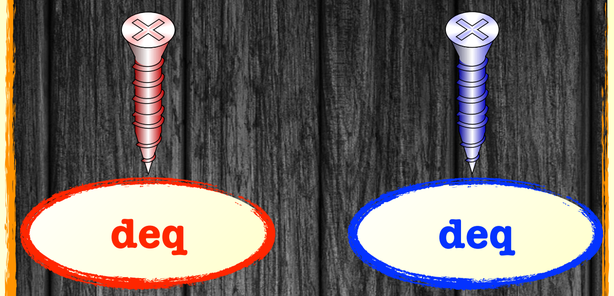

```
deq(k):
Node h,t,x
Integer v
1  while (true)
2    h = Head
3    t = Tail
4    x = h.next
5    if (h == Head)
6      if (h == t)
7        if (x == NULL)
8          return *
9          exit
10       CAS (Tail,t,x)
11     else
12       v = x.key
13       if (CAS (Head,h,x))
14         return v
15  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



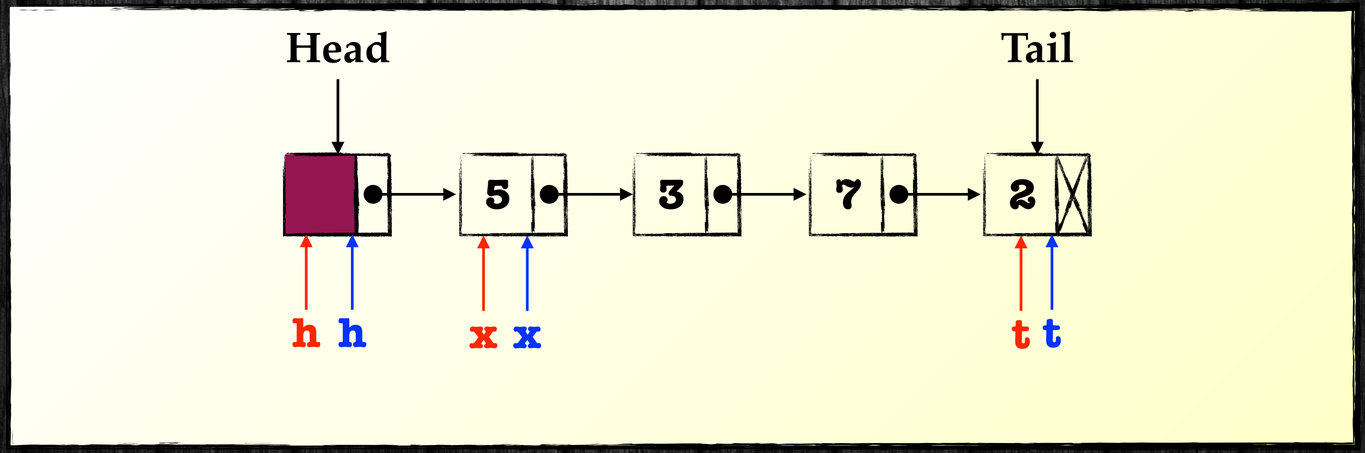
The Michael-Scott Queue Algorithm

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
```



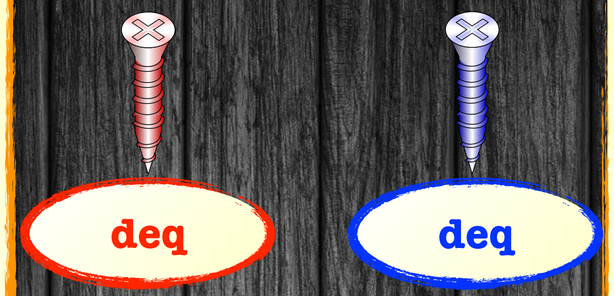

```
deq(k):
Node h,t,x
Integer v
1  while (true)
2    h = Head
3    t = Tail
4    x = h.next
5    if (h == Head)
6      if (h == t)
7        if (x == NULL)
8          return *
9          exit
10       CAS (Tail,t,x)
11     else
12       v = x.key
13       if (CAS (Head,h,x))
14         return v
15  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



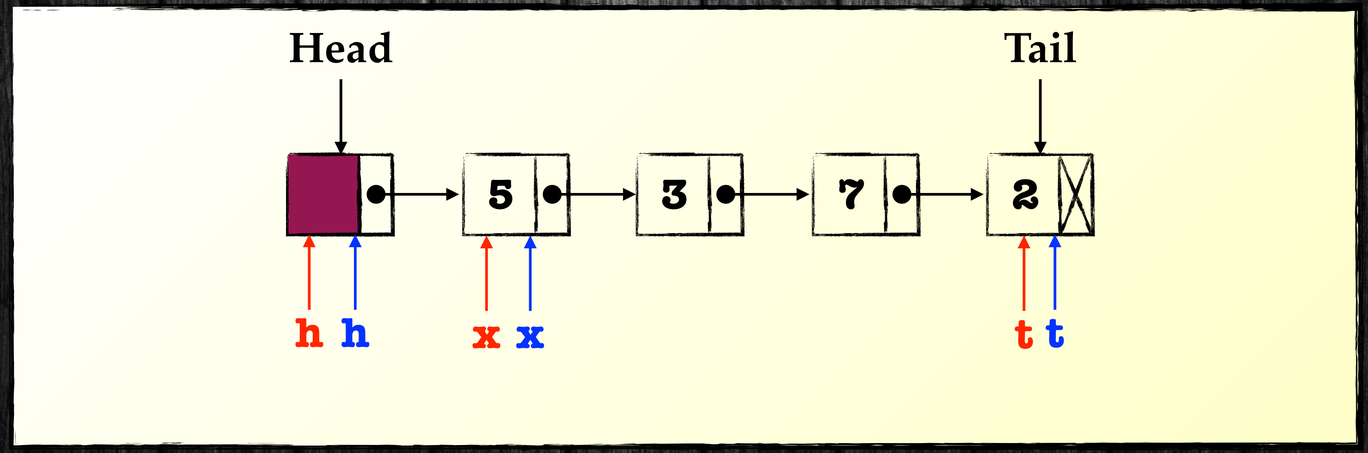
The Michael-Scott Queue Algorithm

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
```



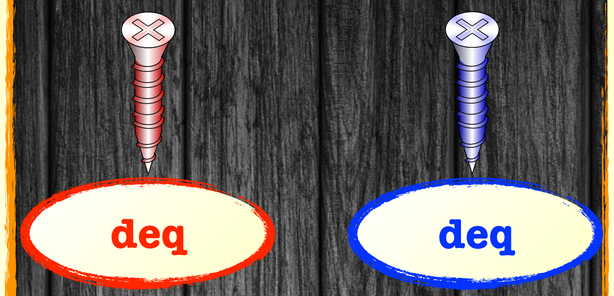

```
deq(k):
Node h,t,x
Integer v
1  while (true)
2    h = Head
3    t = Tail
4    x = h.next
5    if (h == Head)
6      if (h == t)
7        if (x == NULL)
8          return *
9          exit
10       CAS (Tail,t,x)
11     else
12       v = x.key
13       if (CAS (Head,h,x))
14         return v
15  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



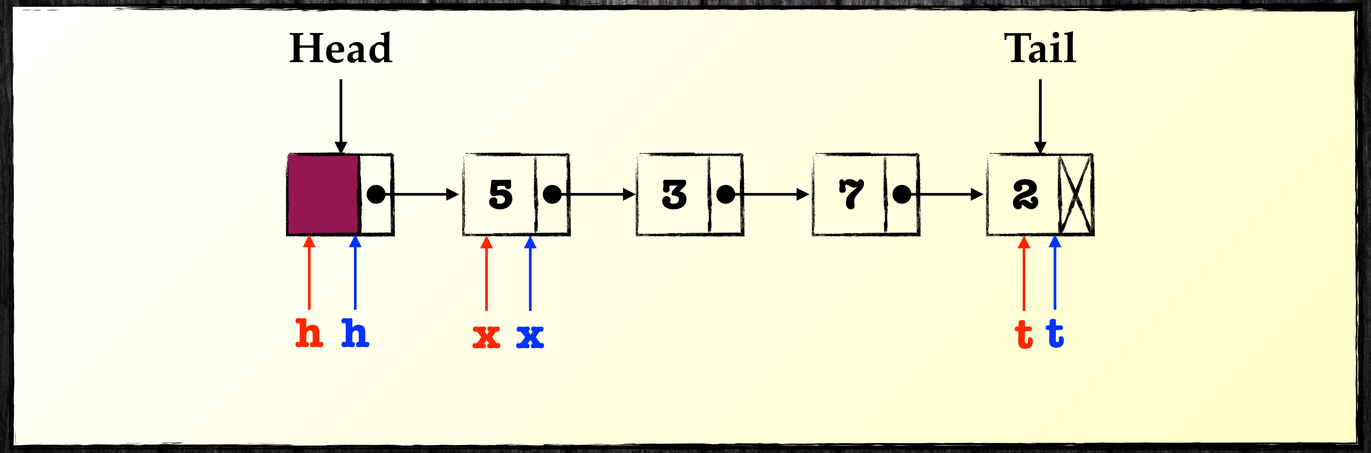
The Michael-Scott Queue Algorithm

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
```



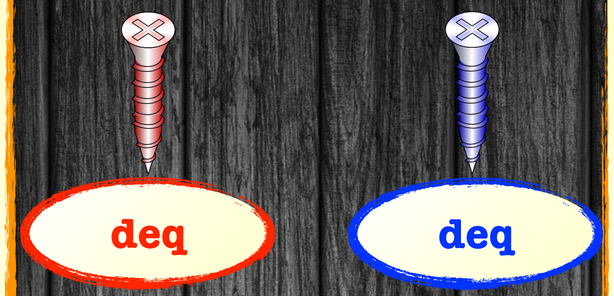

```
deq(k):
Node h,t,x
Integer v
1  while (true)
2    h = Head
3    t = Tail
4    x = h.next
5    if (h == Head)
6      if (h == t)
7        if (x == NULL)
8          return *
9          exit
10       CAS (Tail,t,x)
11     else
12       v = x.key
13       if (CAS (Head,h,x))
14         return v
15  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



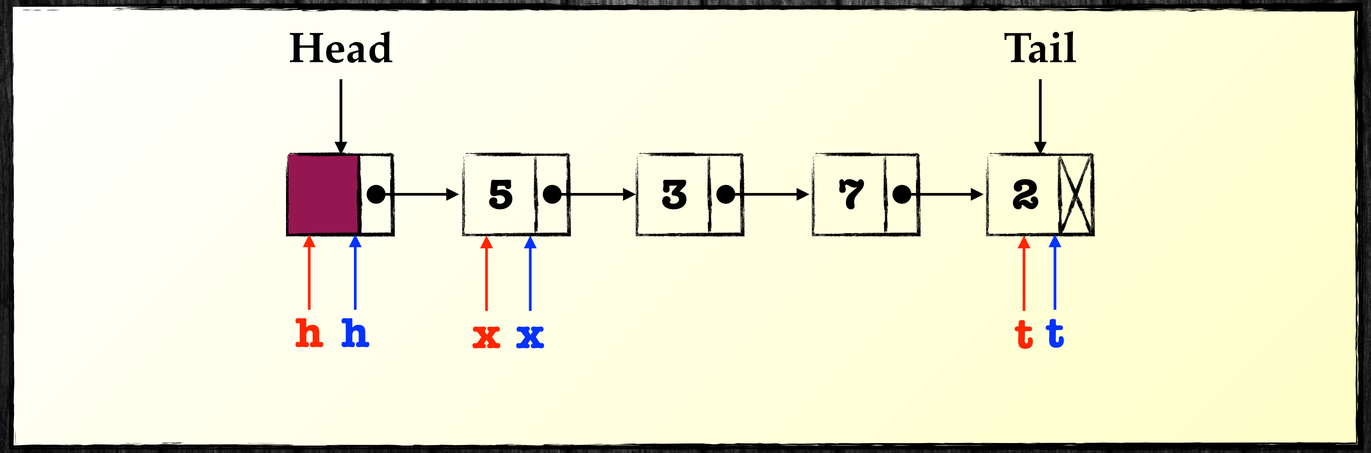
The Michael-Scott Queue Algorithm

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
```



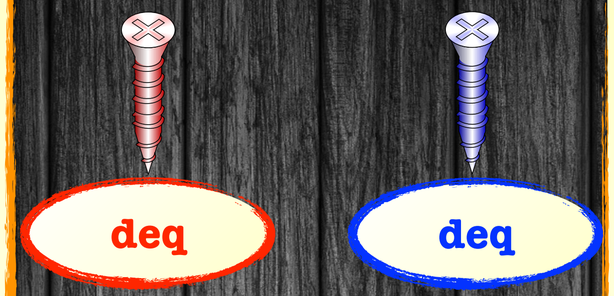

```
deq(k):
Node h,t,x
Integer v
1  while (true)
2    h = Head
3    t = Tail
4    x = h.next
5    if (h == Head)
6      if (h == t)
7        if (x == NULL)
8          return *
9          exit
10       CAS (Tail,t,x)
11     else
12       v = x.key
13       if (CAS (Head,h,x))
14         return v
15  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



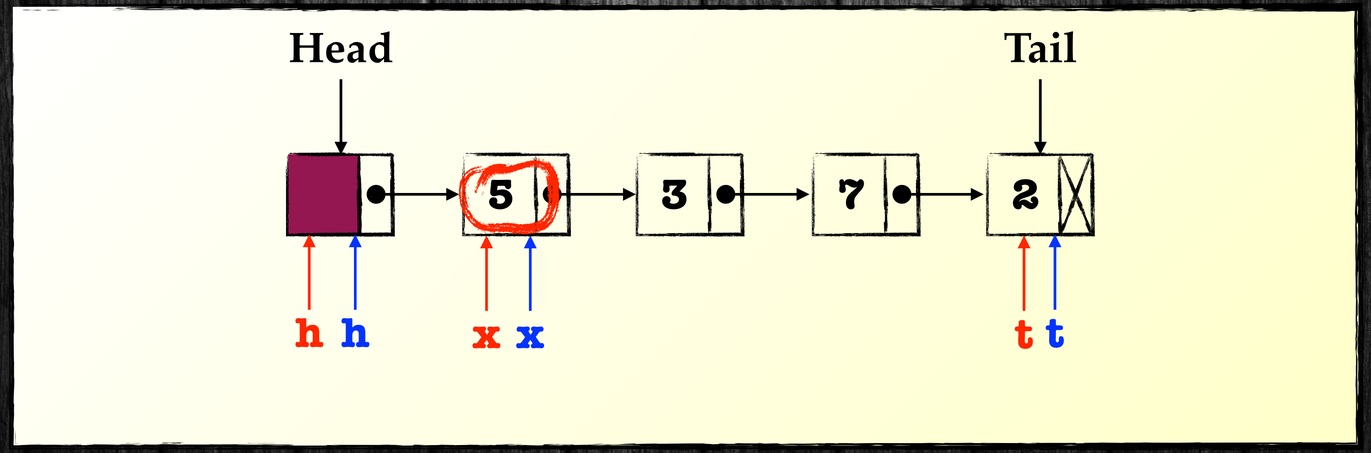
The Michael-Scott Queue Algorithm

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
```



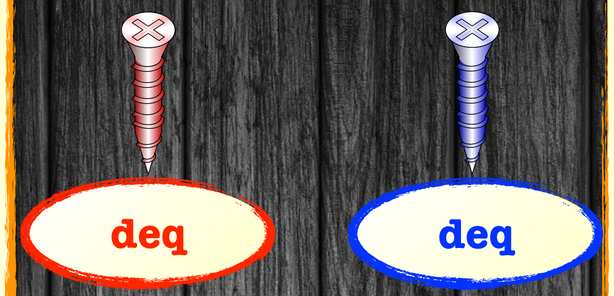

```
deq(k):
Node h,t,x
Integer v
1  while (true)
2    h = Head
3    t = Tail
4    x = h.next
5    if (h == Head)
6      if (h == t)
7        if (x == NULL)
8          return *
9          exit
10       CAS (Tail,t,x)
11     else
12       v = x.key
13       if (CAS (Head,h,x))
14         return v
15  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



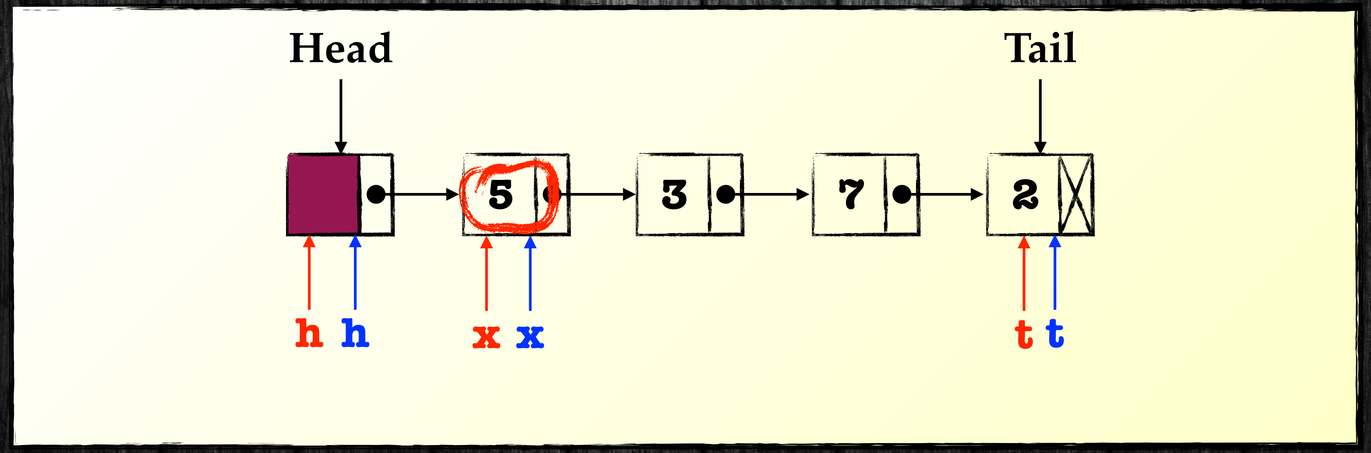
The Michael-Scott Queue Algorithm

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
```



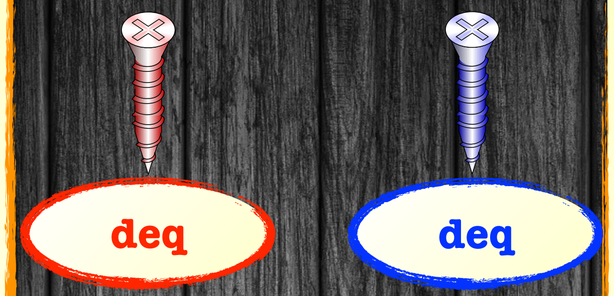

```
deq(k):
Node h,t,x
Integer v
1  while (true)
2    h = Head
3    t = Tail
4    x = h.next
5    if (h == Head)
6      if (h == t)
7        if (x == NULL)
8          return *
9          exit
10       CAS (Tail,t,x)
11     else
12       v = x.key
13       if (CAS (Head,h,x))
14         return v
15  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



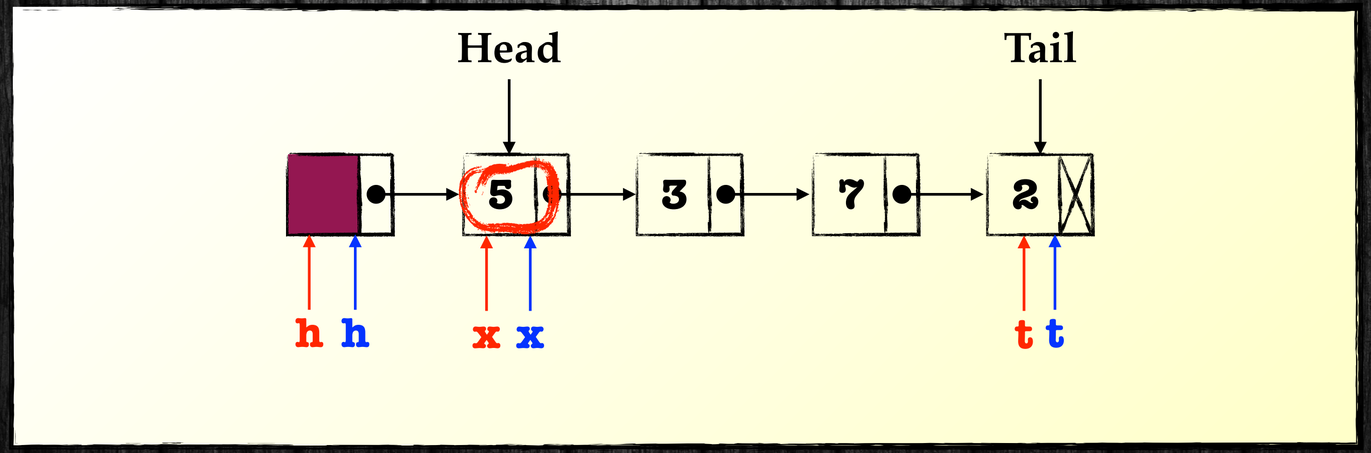
The Michael-Scott Queue Algorithm

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
```



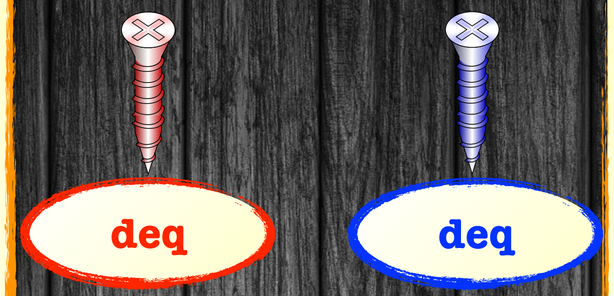

```
deq(k):
Node h,t,x
Integer v
1  while (true)
2    h = Head
3    t = Tail
4    x = h.next
5    if (h == Head)
6      if (h == t)
7        if (x == NULL)
8          return *
9          exit
10       CAS (Tail,t,x)
11     else
12       v = x.key
13       if (CAS (Head,h,x))
14         return v
15  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



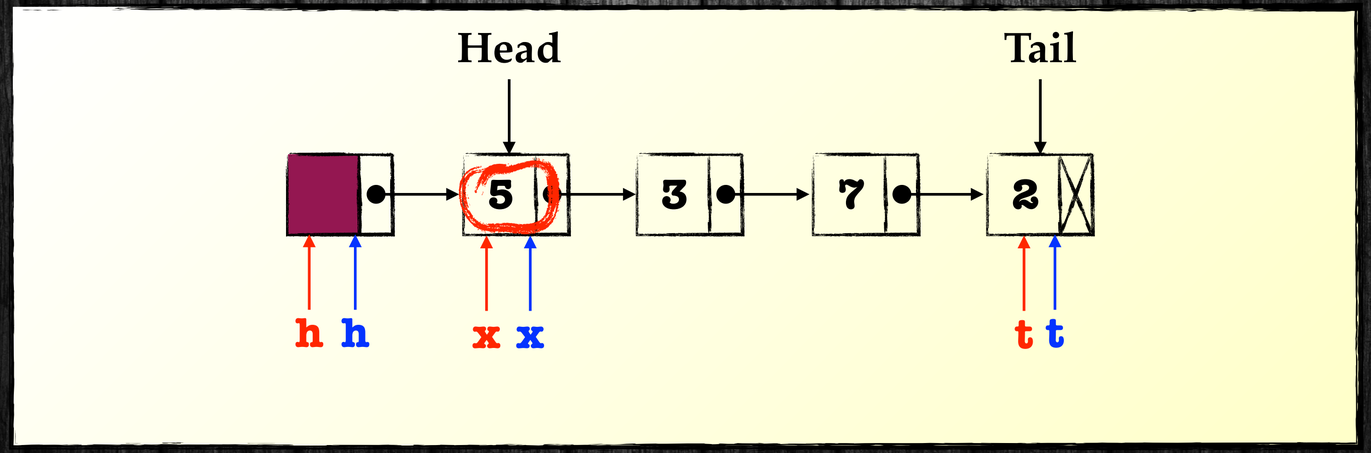
The Michael-Scott Queue Algorithm

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
```



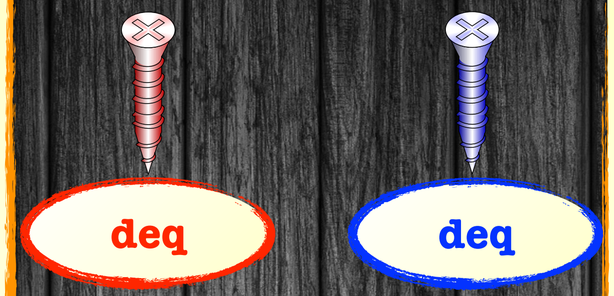

```
deq(k):
Node h,t,x
Integer v
1  while (true)
2    h = Head
3    t = Tail
4    x = h.next
5    if (h == Head)
6      if (h == t)
7        if (x == NULL)
8          return *
9          exit
10       CAS (Tail,t,x)
11     else
12       v = x.key
13       if (CAS (Head,h,x))
14         return v
15  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



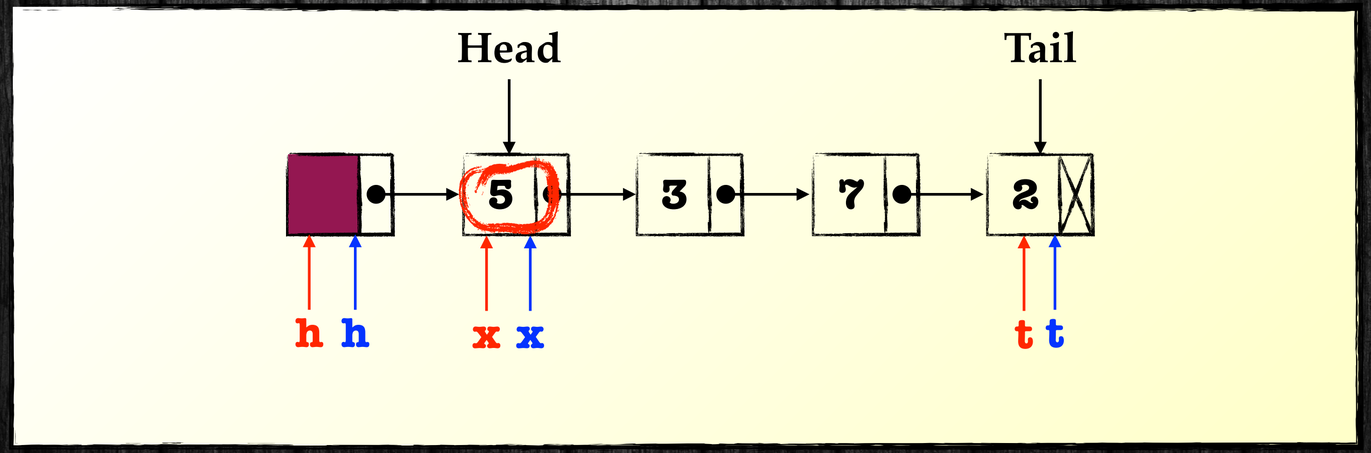
The Michael-Scott Queue Algorithm

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
```



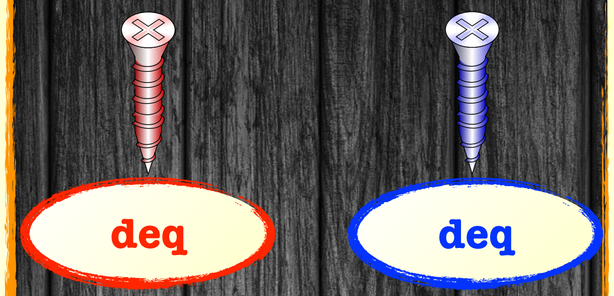

```
deq(k):
Node h,t,x
Integer v
1  while (true)
2    h = Head
3    t = Tail
4    x = h.next
5    if (h == Head)
6      if (h == t)
7        if (x == NULL)
8          return *
9          exit
10       CAS (Tail,t,x)
11     else
12       v = x.key
13       if (CAS (Head,h,x))
14         return v
15  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



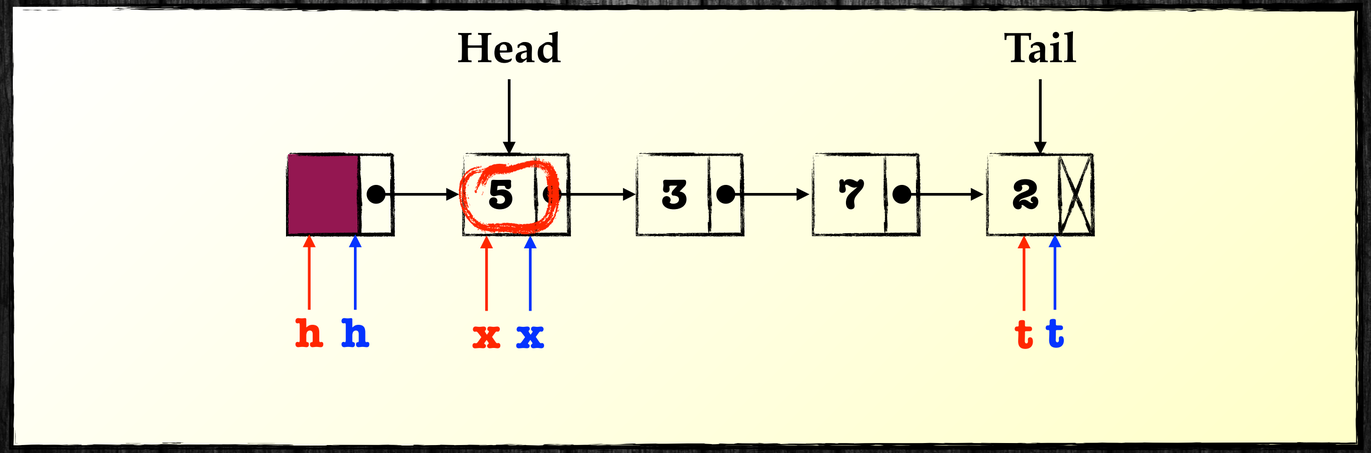
The Michael-Scott Queue Algorithm

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
```



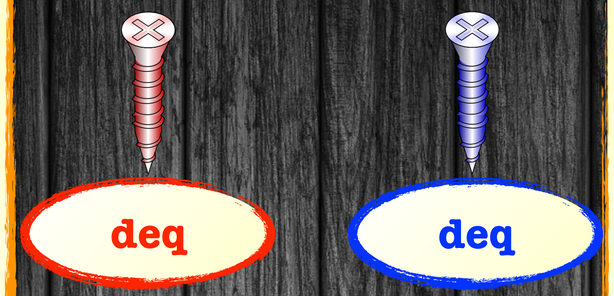

```
deq(k):
Node h,t,x
Integer v
1  while (true)
2    h = Head
3    t = Tail
4    x = h.next
5    if (h == Head)
6      if (h == t)
7        if (x == NULL)
8          return *
9          exit
10       CAS (Tail,t,x)
11     else
12       v = x.key
13       if (CAS (Head,h,x))
14         return v
15  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



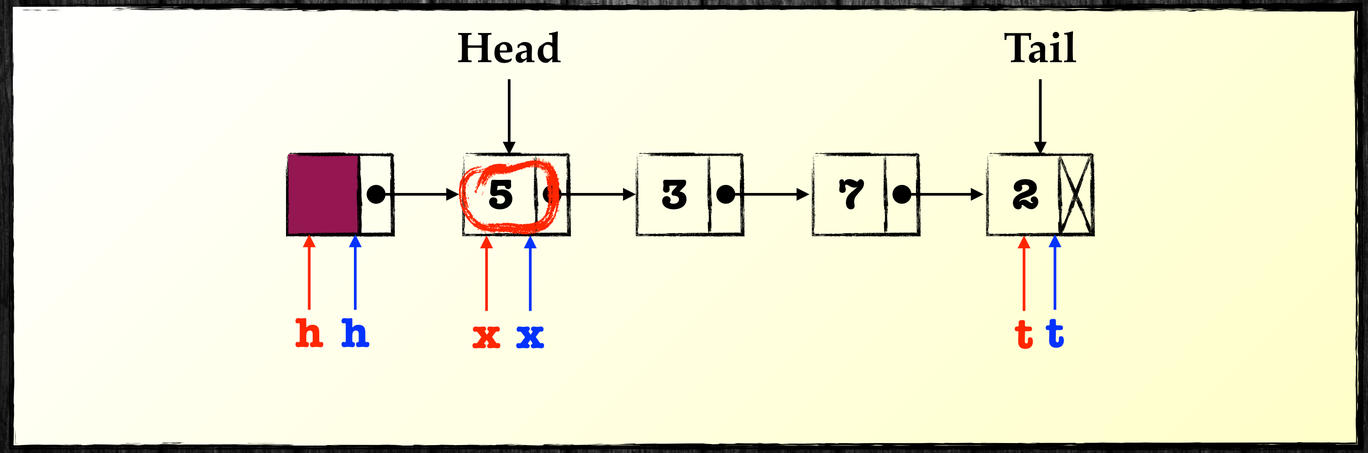
The Michael-Scott Queue Algorithm

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
```



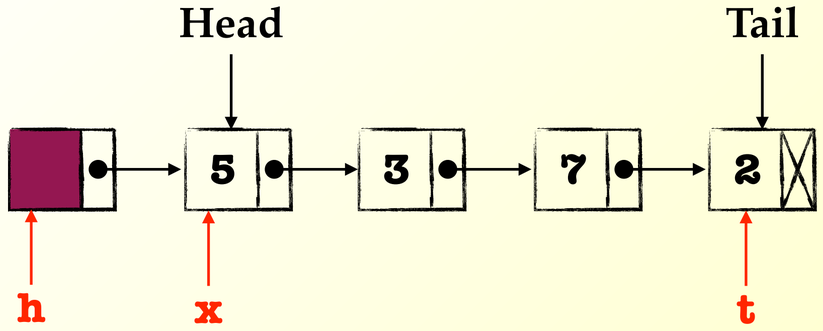
The Michael-Scott Queue Algorithm

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
```



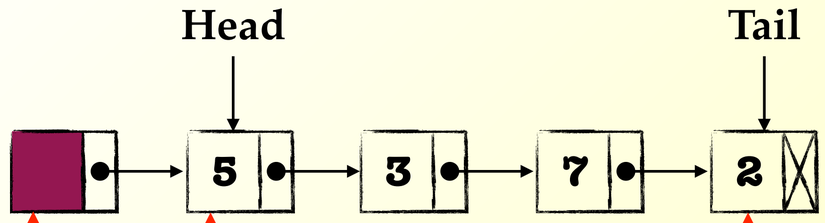
The Michael-Scott Queue Algorithm

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
```



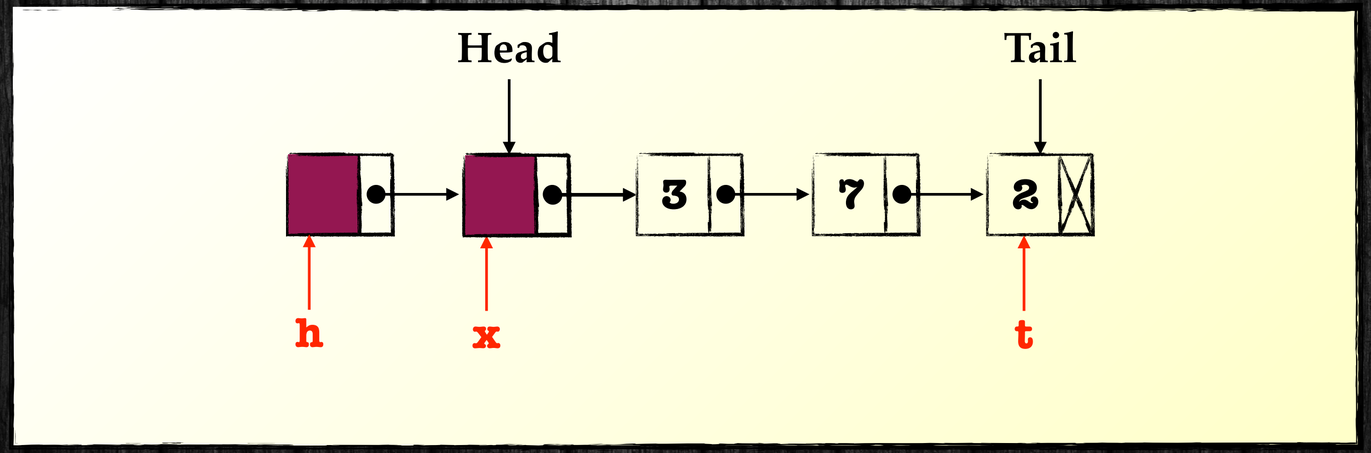
The Michael-Scott Queue Algorithm

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
```



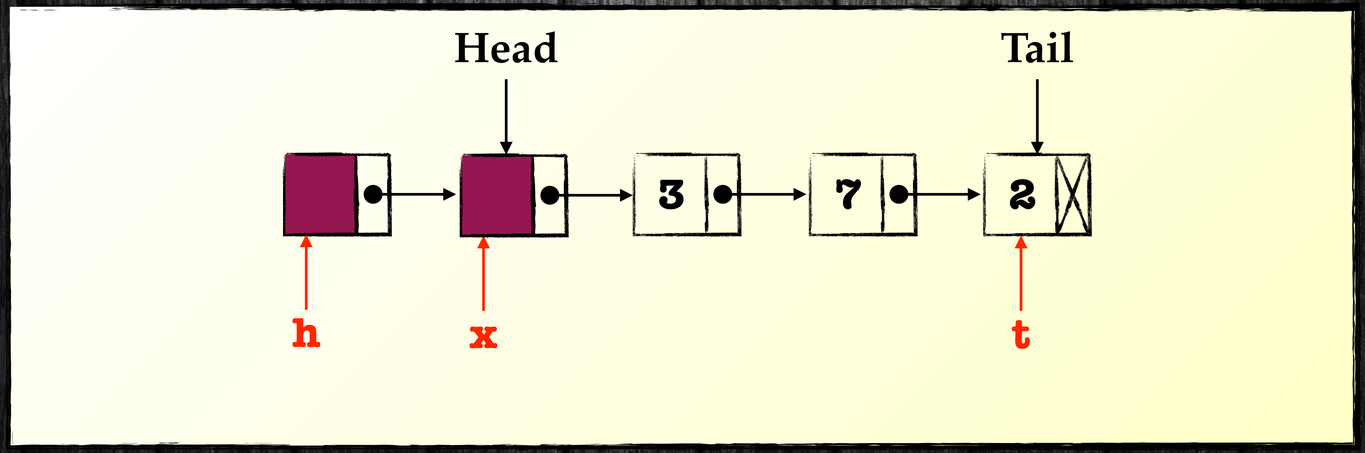
The Michael-Scott Queue Algorithm

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
```



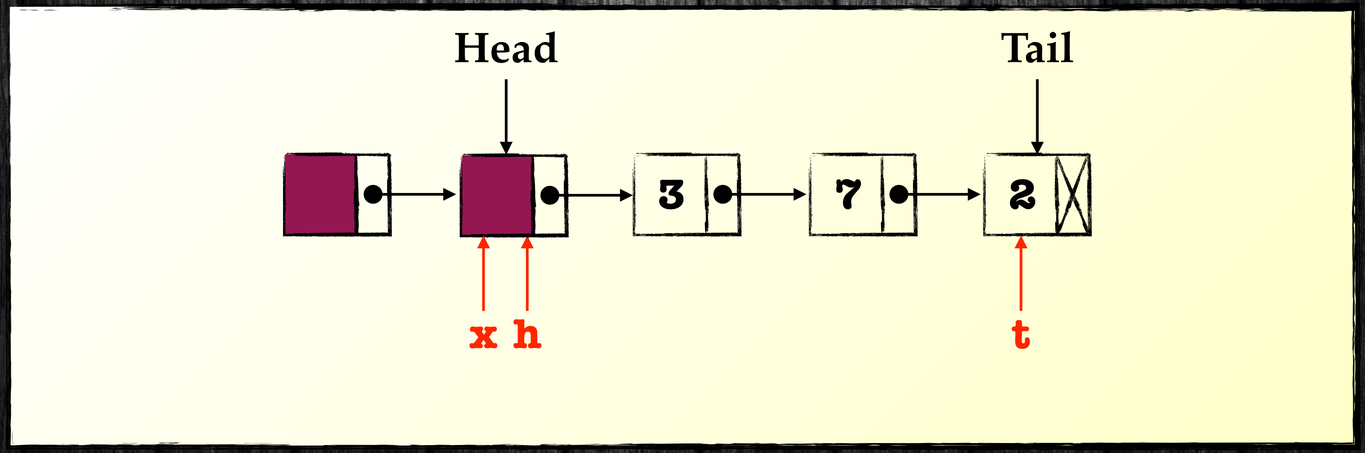
The Michael-Scott Queue Algorithm

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
```



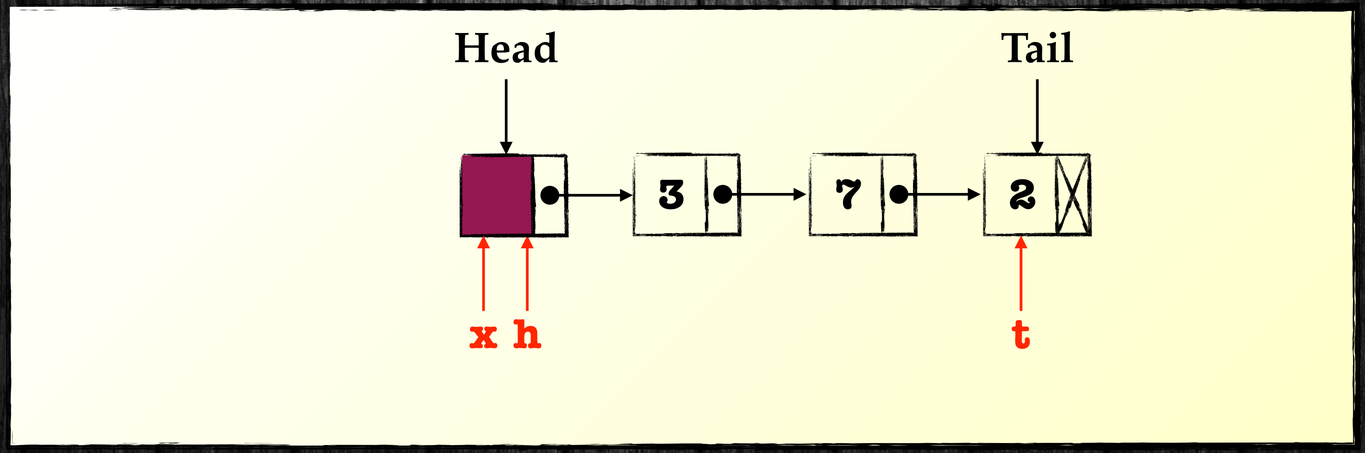
The Michael-Scott Queue Algorithm

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
```



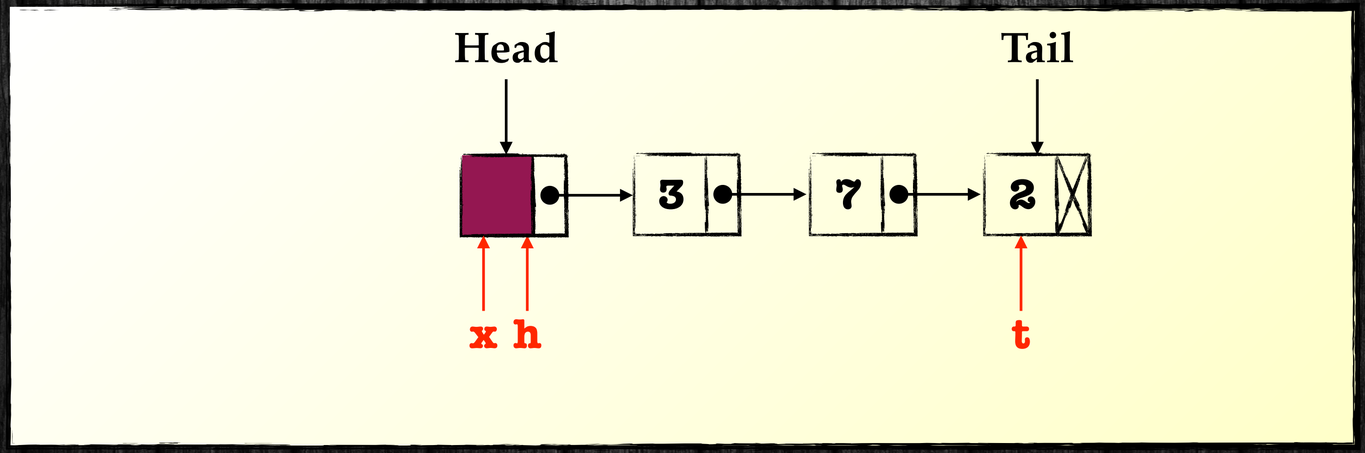
The Michael-Scott Queue Algorithm

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
```



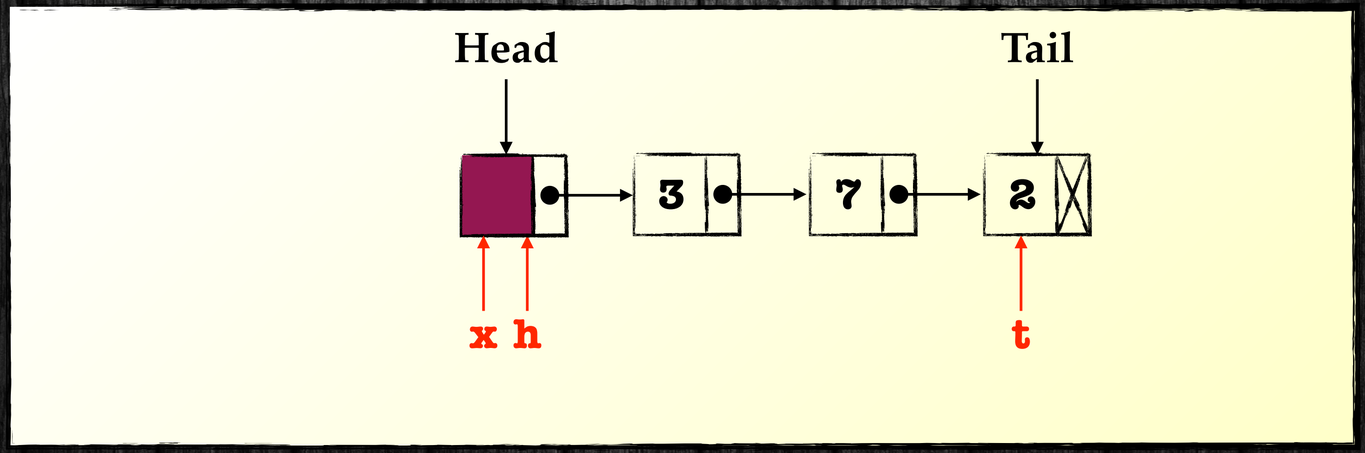
The Michael-Scott Queue Algorithm

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
```



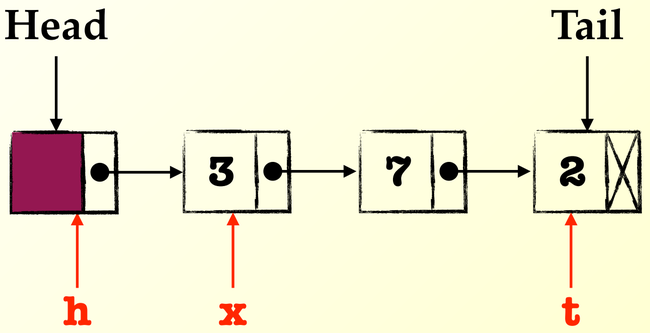
The Michael-Scott Queue Algorithm

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
```



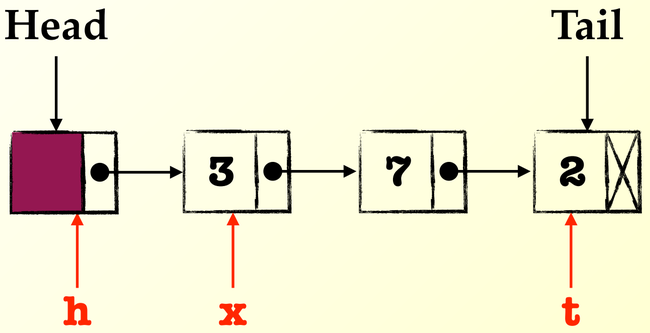
The Michael-Scott Queue Algorithm

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
```



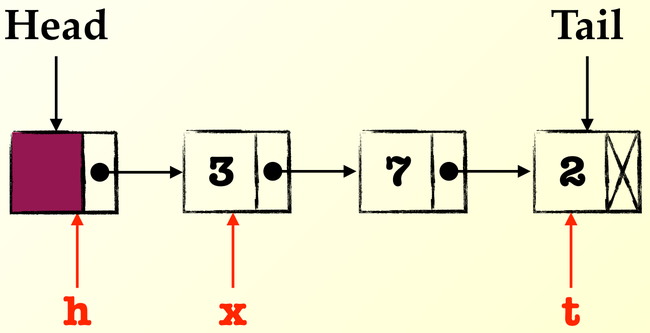
The Michael-Scott Queue Algorithm

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
```



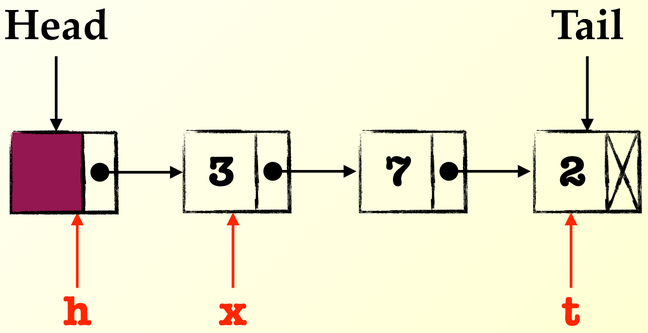
The Michael-Scott Queue Algorithm

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
```



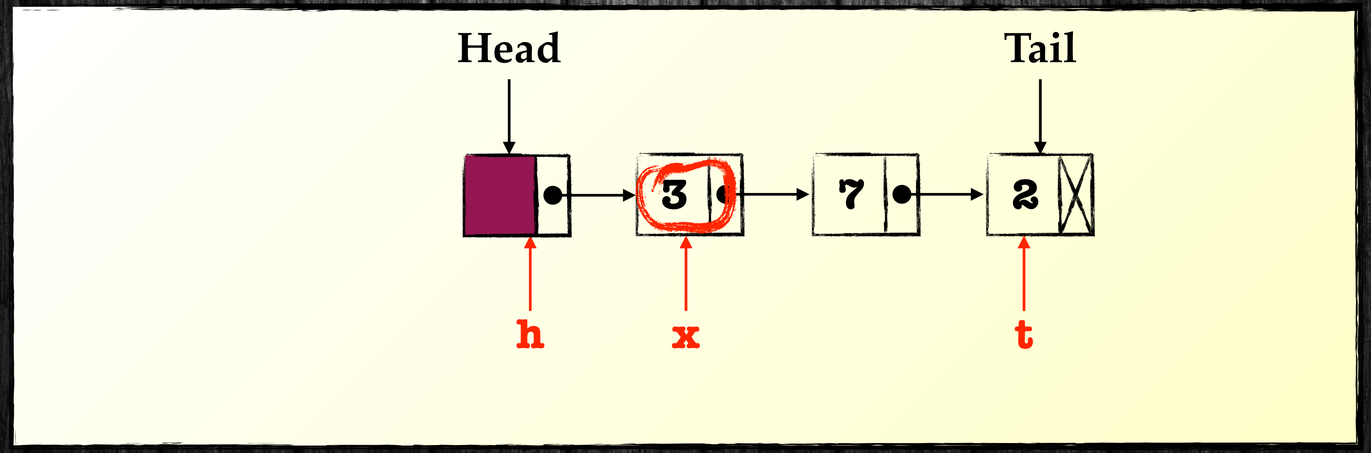
The Michael-Scott Queue Algorithm

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
```



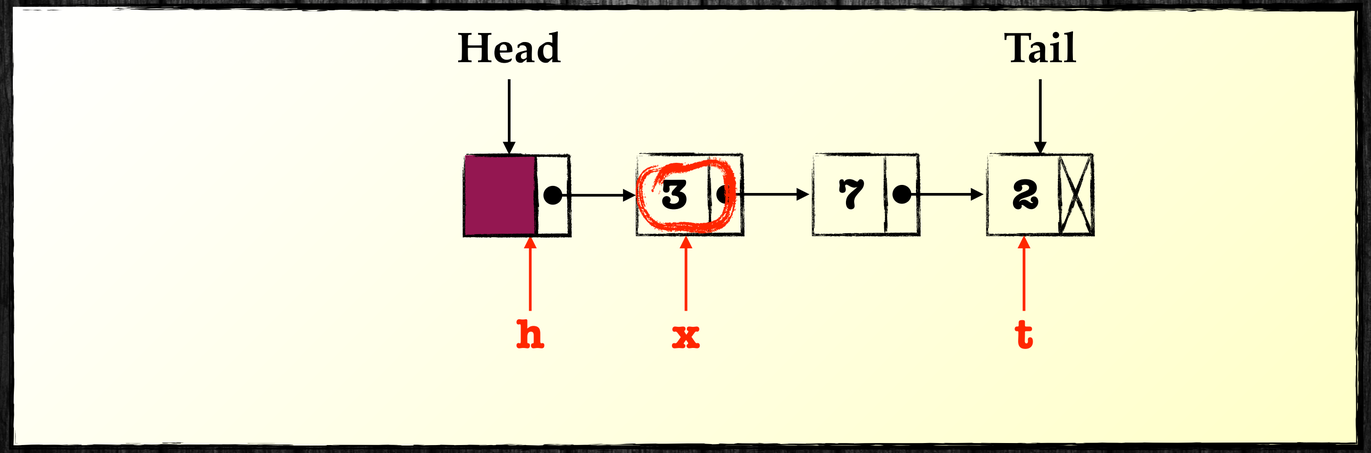
The Michael-Scott Queue Algorithm

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
```



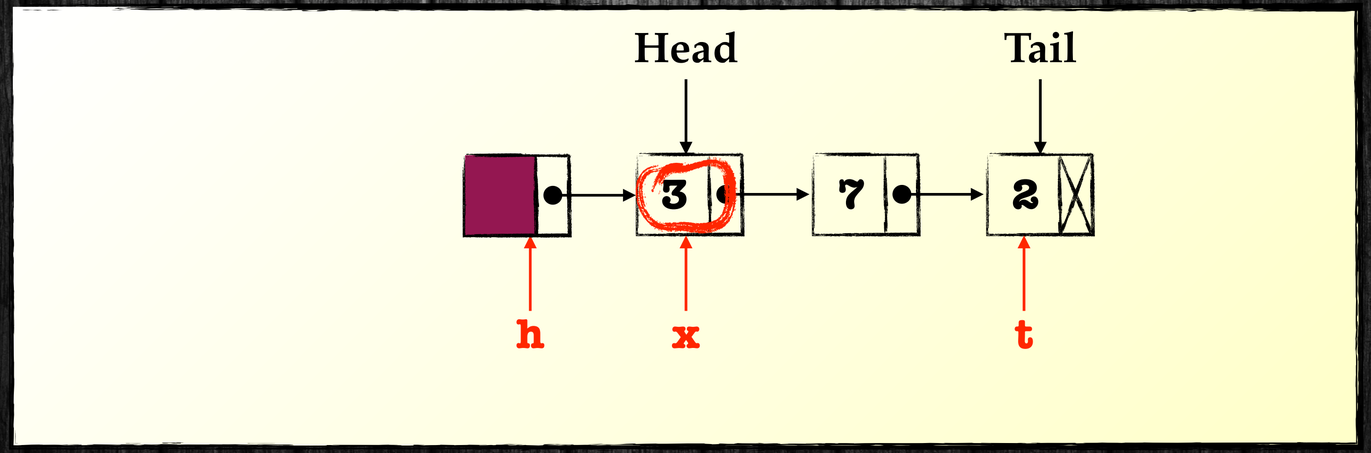
The Michael-Scott Queue Algorithm

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
```



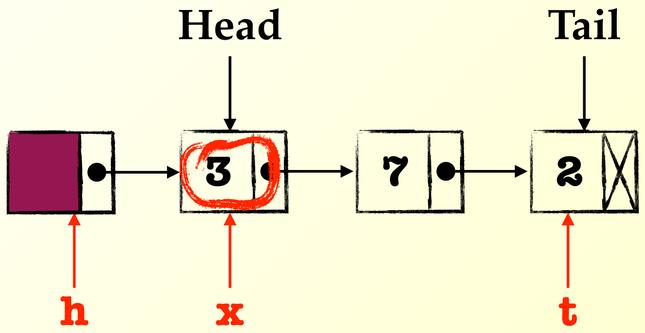
The Michael-Scott Queue Algorithm

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
```



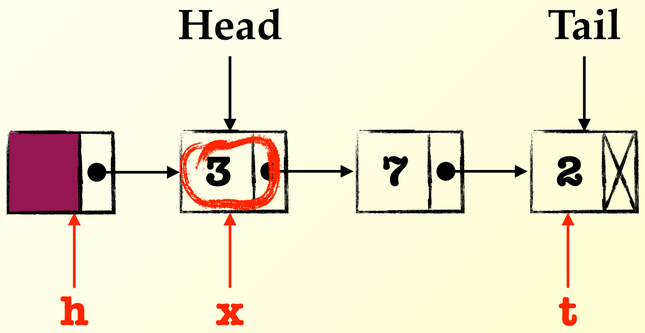
The Michael-Scott Queue Algorithm

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
```

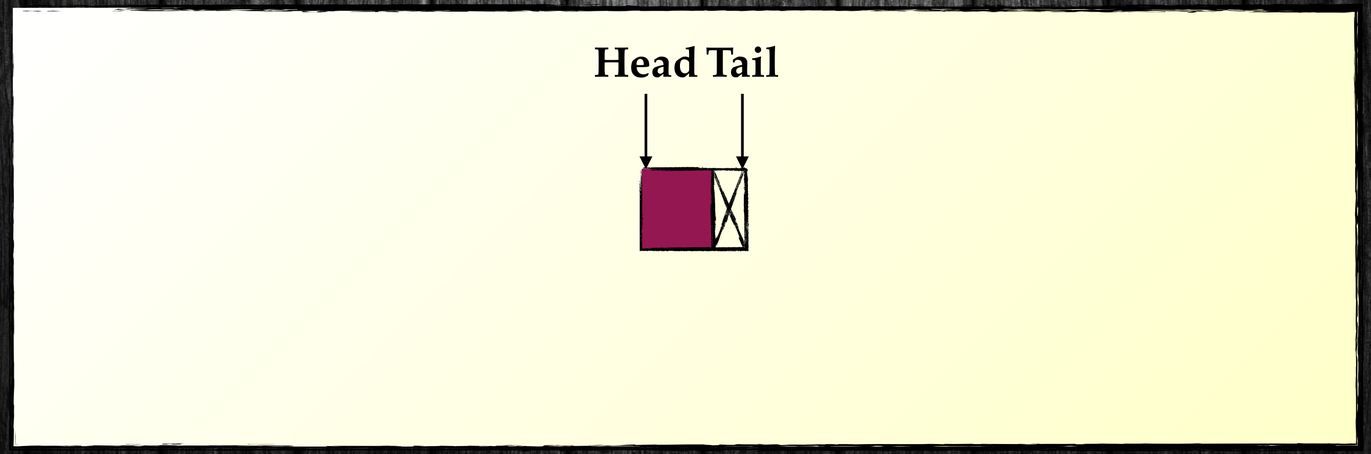


The Michael-Scott Queue Algorithm

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
```



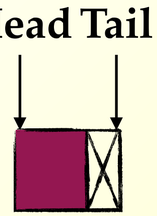


The Michael-Scott Queue Algorithm

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
```



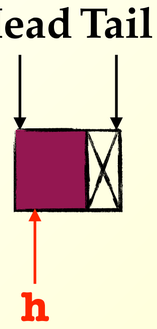


The Michael-Scott Queue Algorithm

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
```



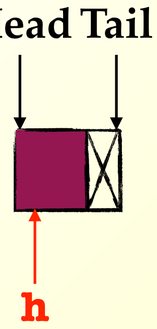


The Michael-Scott Queue Algorithm

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
```



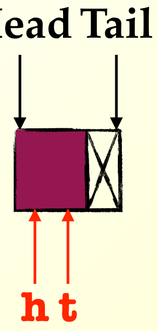


The Michael-Scott Queue Algorithm

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
```



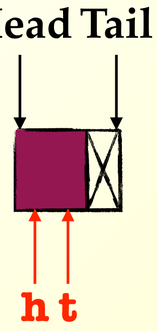


The Michael-Scott Queue Algorithm

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
```



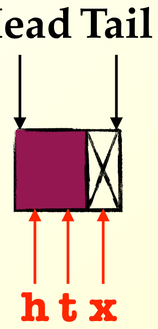


The Michael-Scott Queue Algorithm

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
```



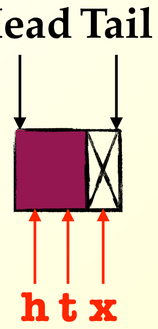


The Michael-Scott Queue Algorithm

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
```



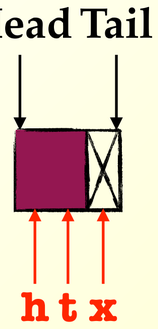


The Michael-Scott Queue Algorithm

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
```



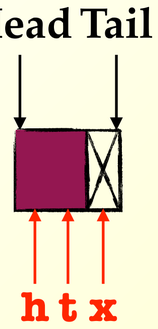


The Michael-Scott Queue Algorithm

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
```



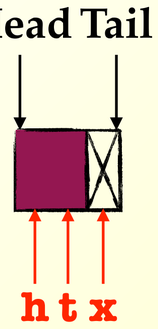


The Michael-Scott Queue Algorithm

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
```



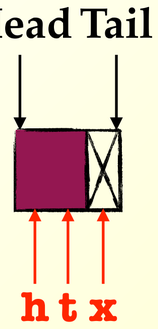


The Michael-Scott Queue Algorithm

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
```



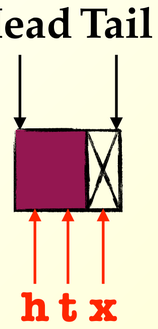


The Michael-Scott Queue Algorithm

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

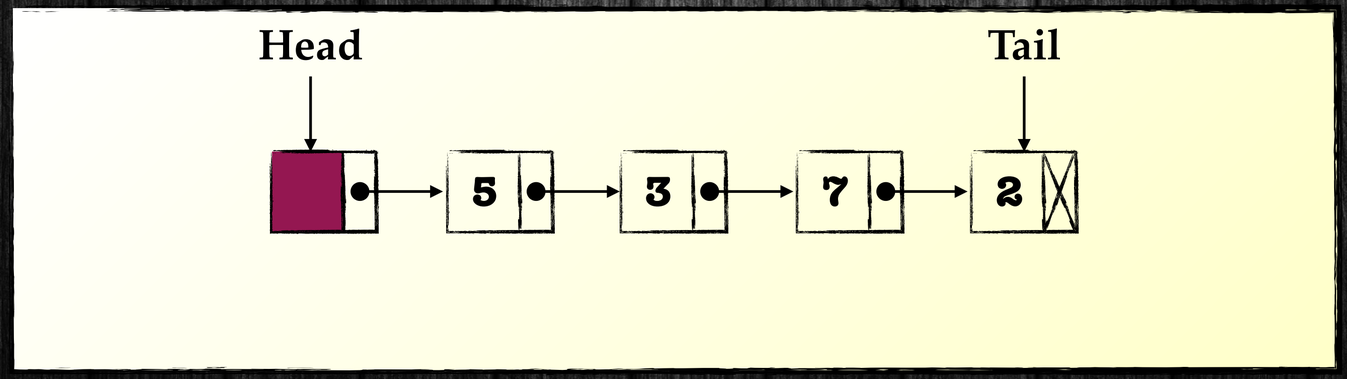
when successful

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
```

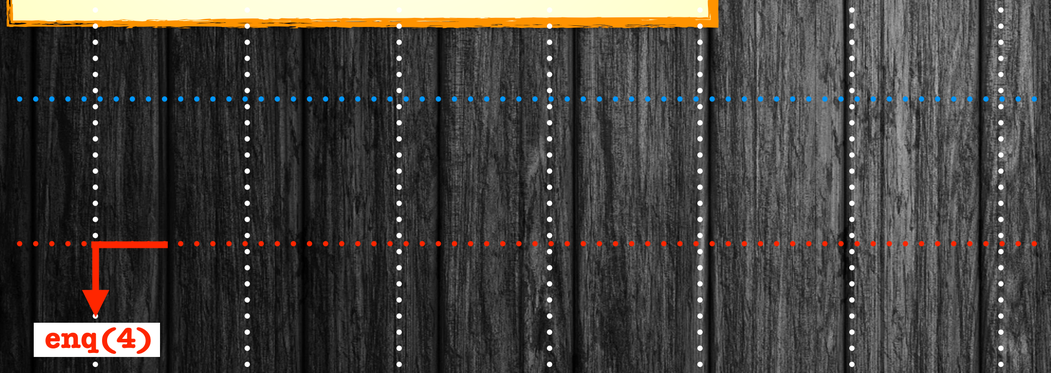


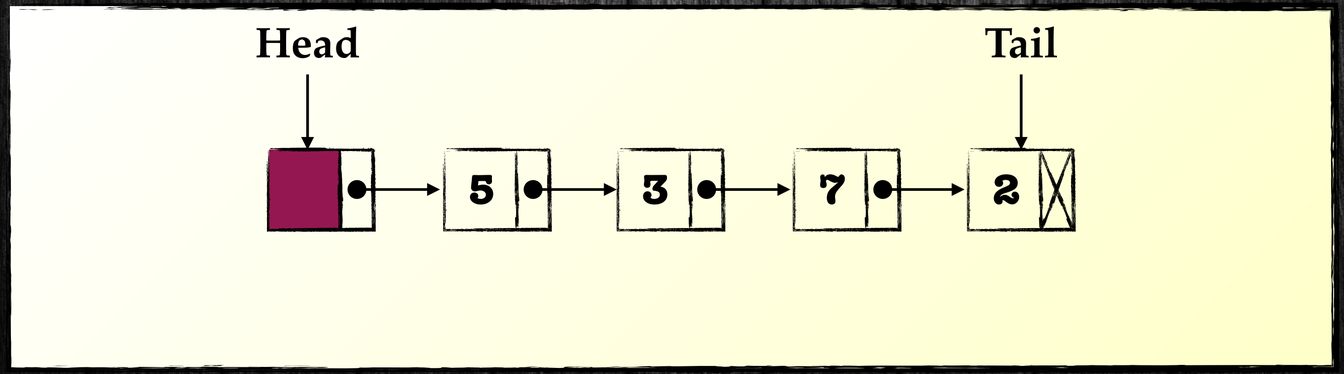
The Michael-Scott Queue Algorithm

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
```





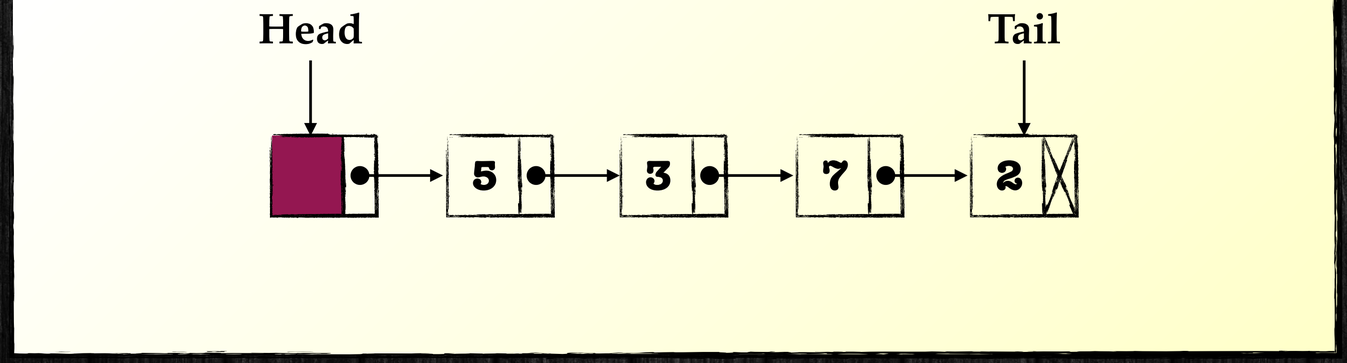
The Michael-Scott Queue Algorithm

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)



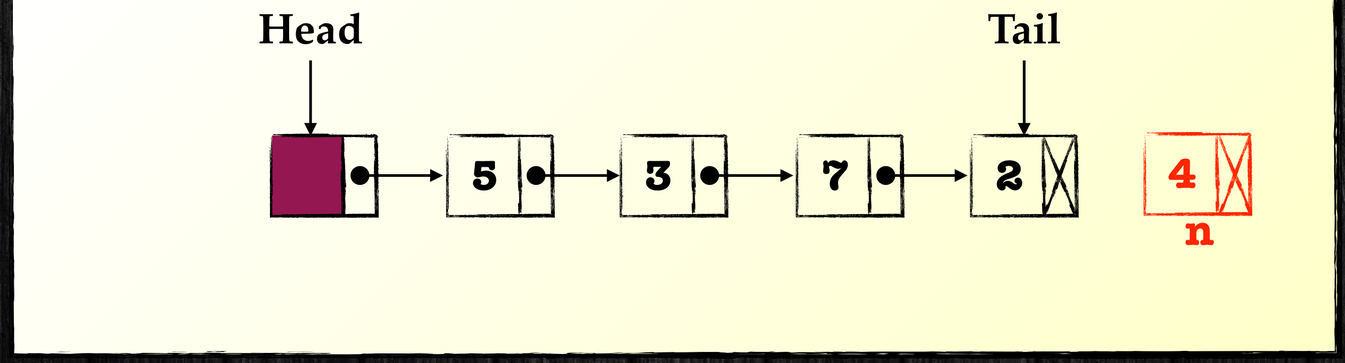
The Michael-Scott Queue Algorithm

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)



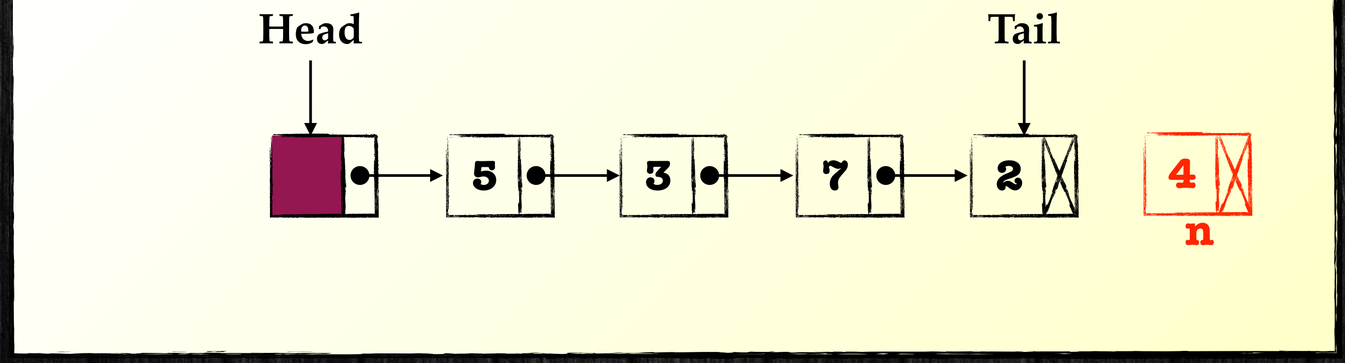
The Michael-Scott Queue Algorithm

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)



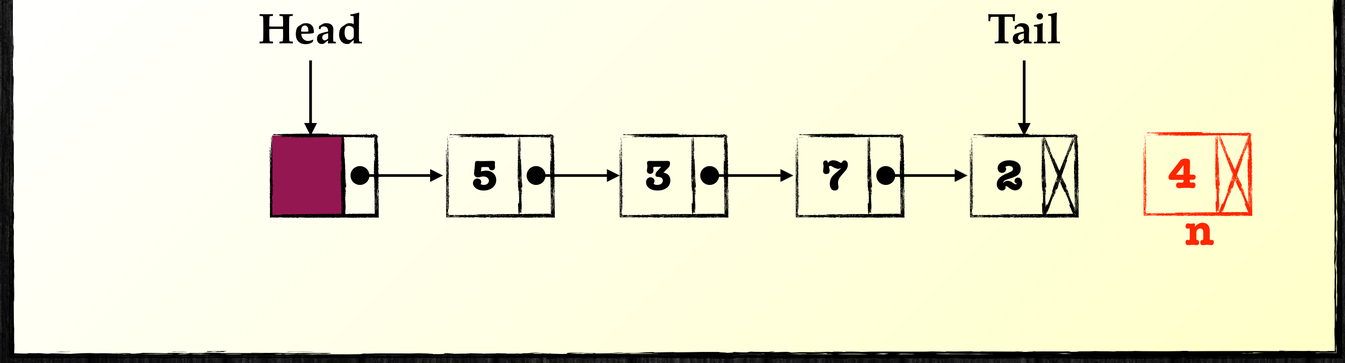
The Michael-Scott Queue Algorithm

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)



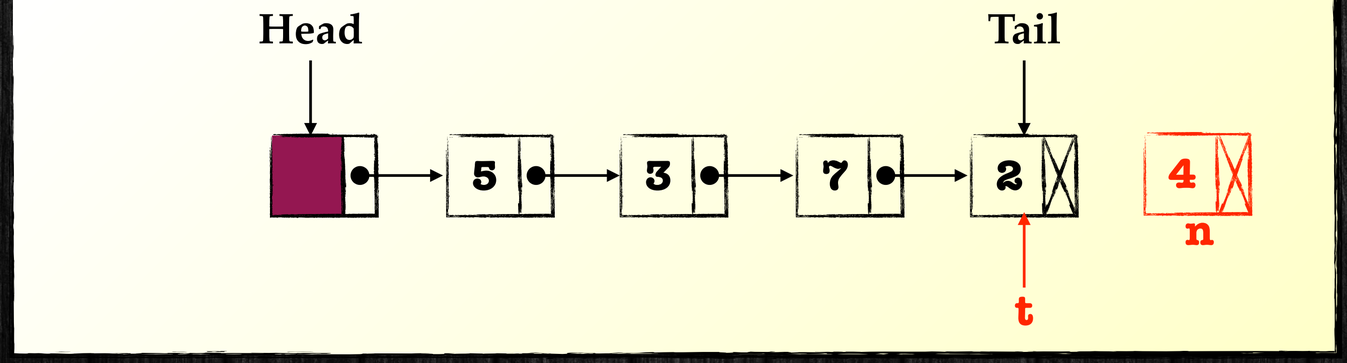
The Michael-Scott Queue Algorithm

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)



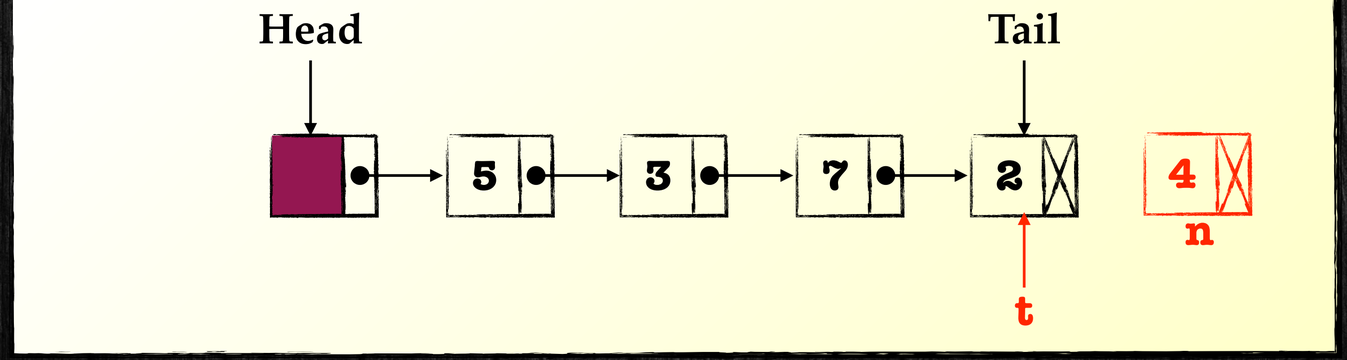
The Michael-Scott Queue Algorithm

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)



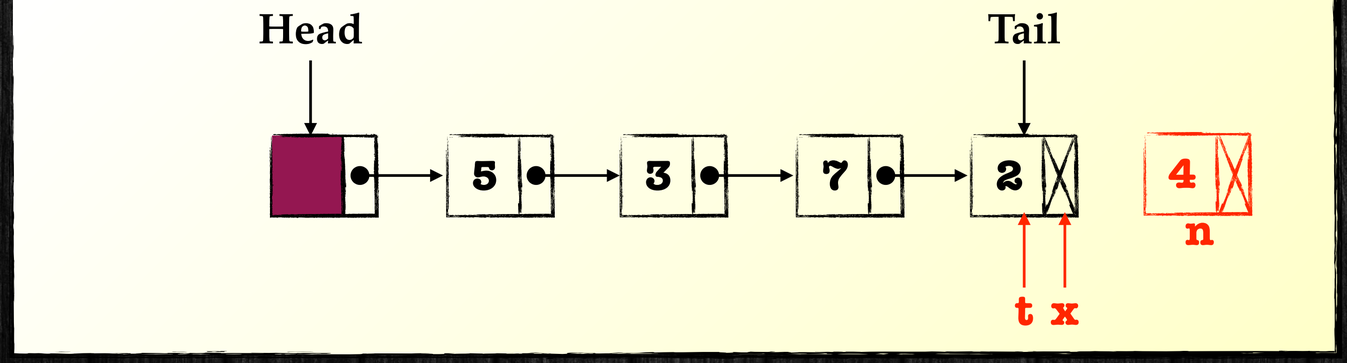
The Michael-Scott Queue Algorithm

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)



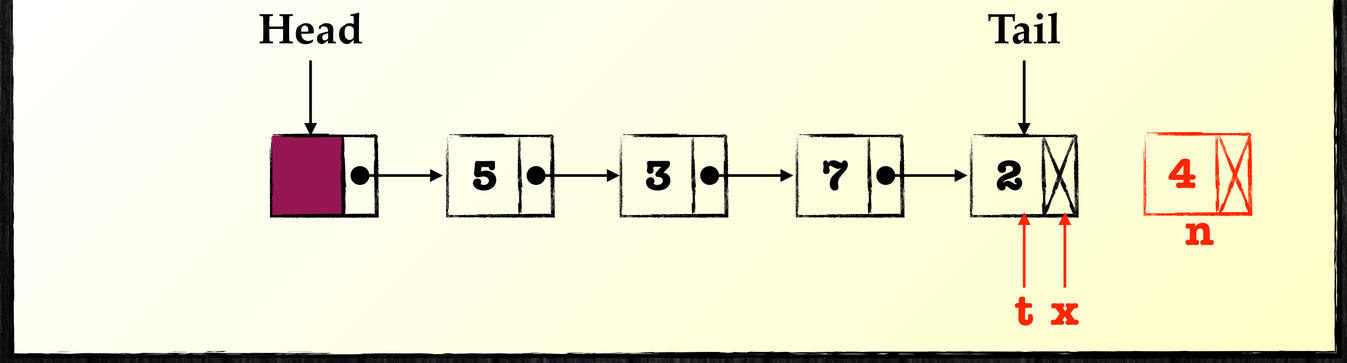
The Michael-Scott Queue Algorithm

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)



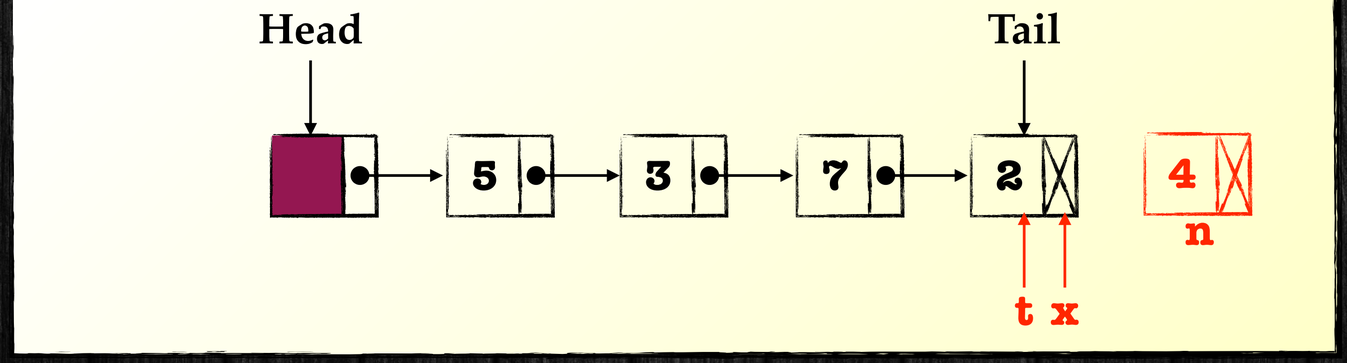
The Michael-Scott Queue Algorithm

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)



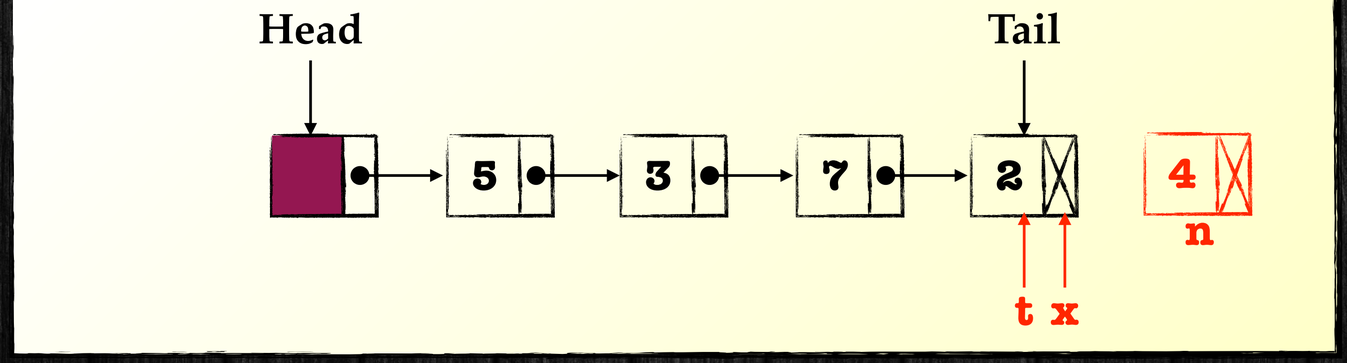
The Michael-Scott Queue Algorithm

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)



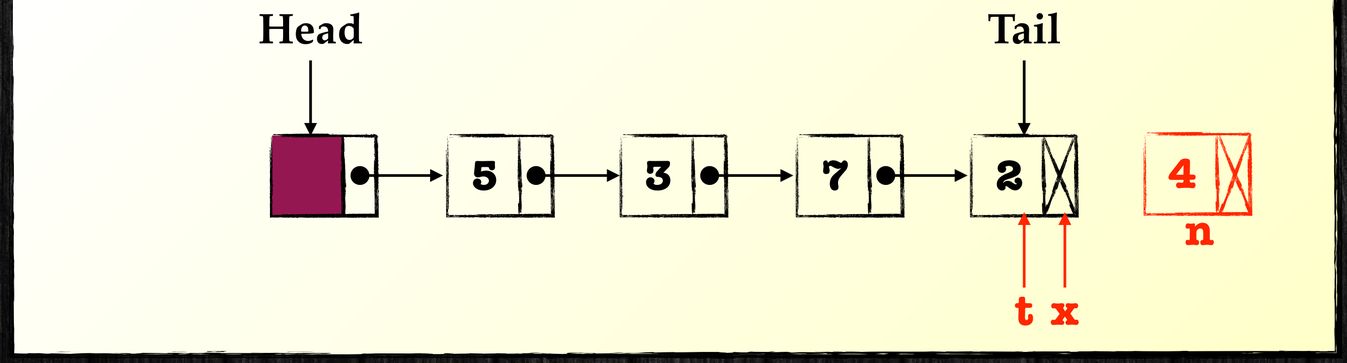
The Michael-Scott Queue Algorithm

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)

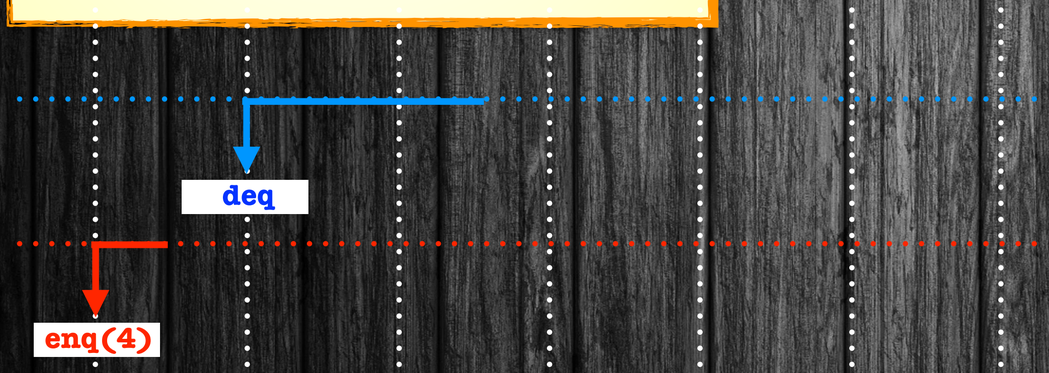


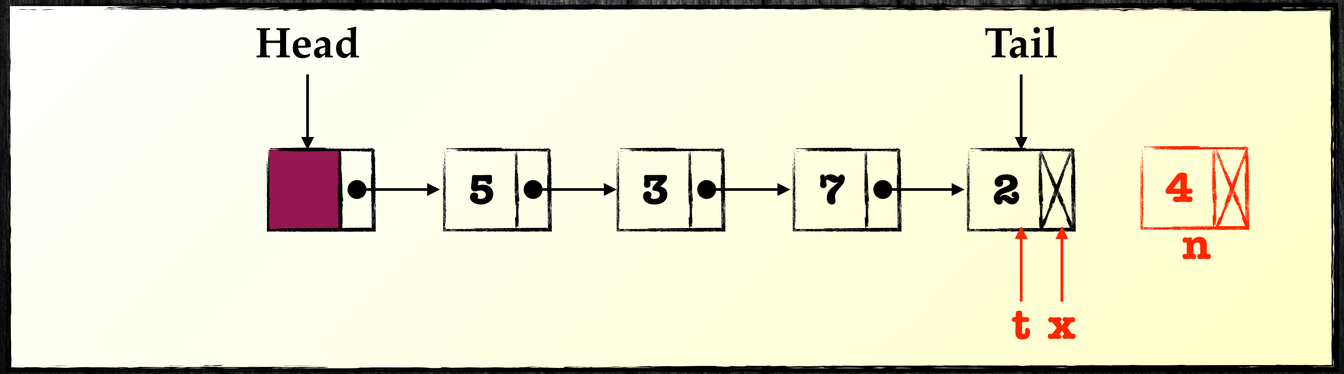
The Michael-Scott Queue Algorithm

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
```



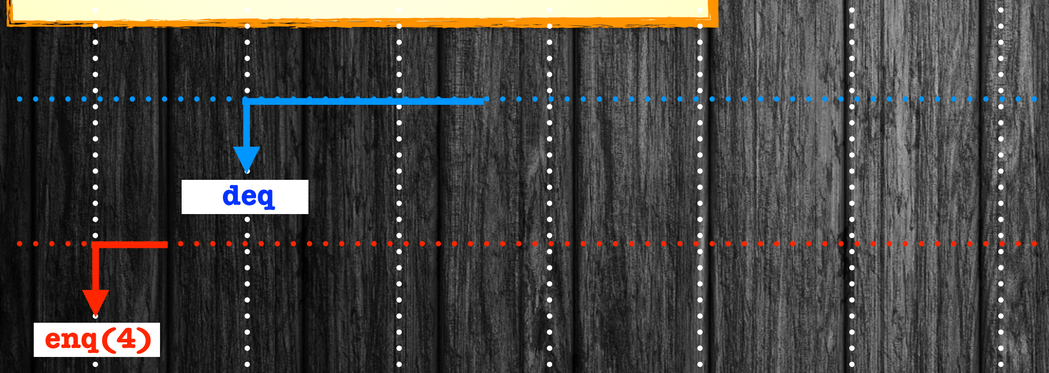


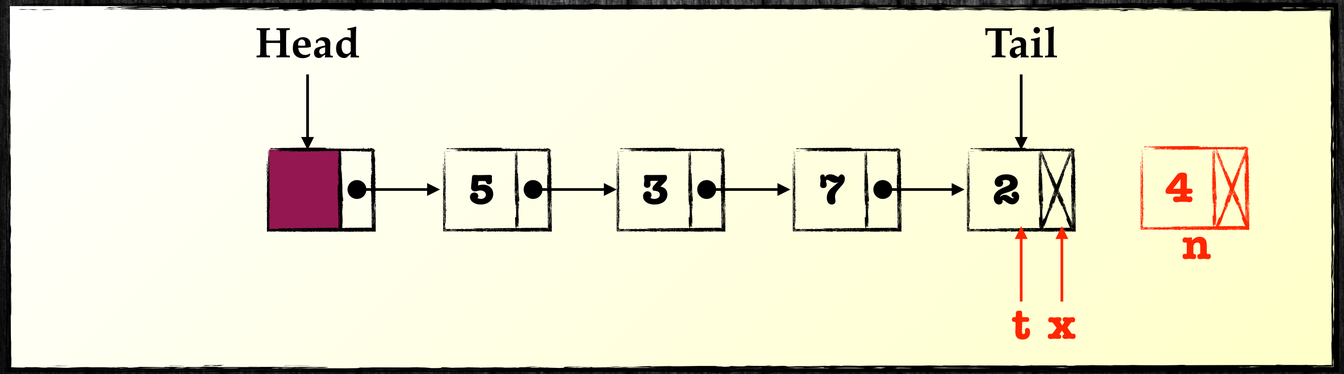
The Michael-Scott Queue Algorithm

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
```



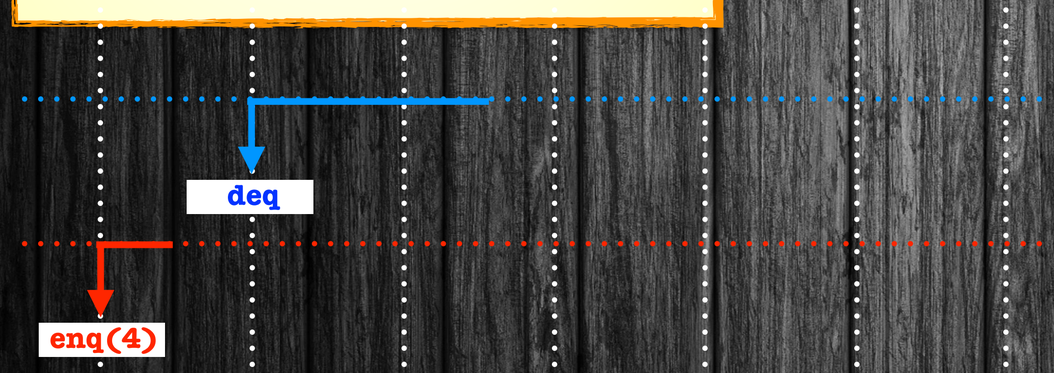


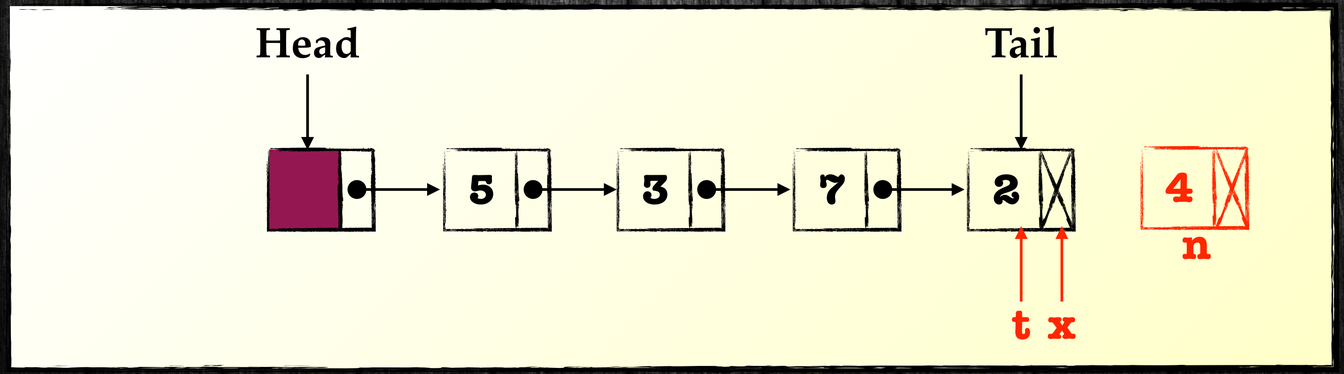
The Michael-Scott Queue Algorithm

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
```



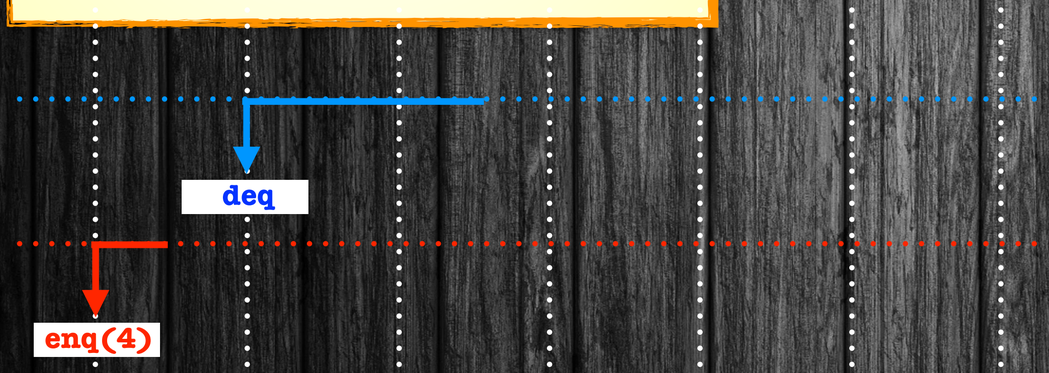


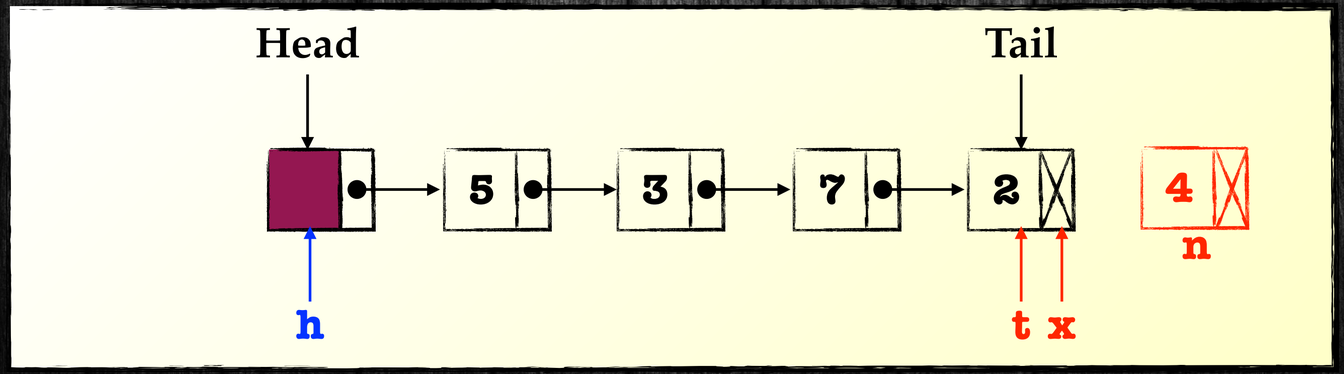
The Michael-Scott Queue Algorithm

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
```



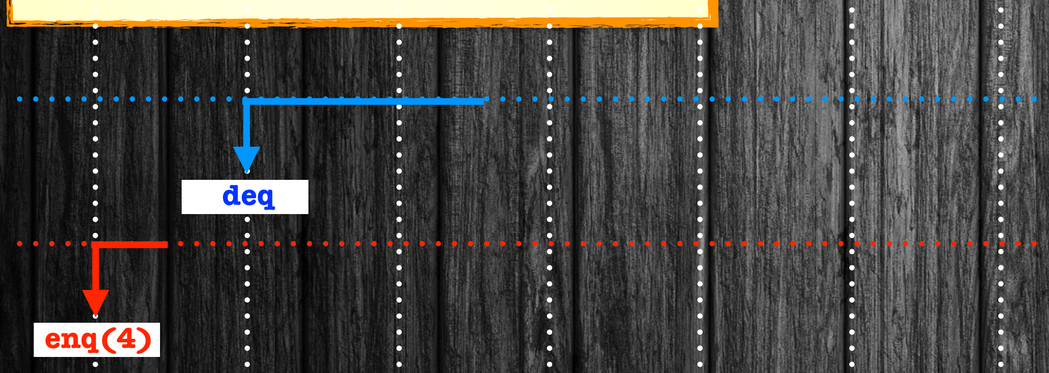


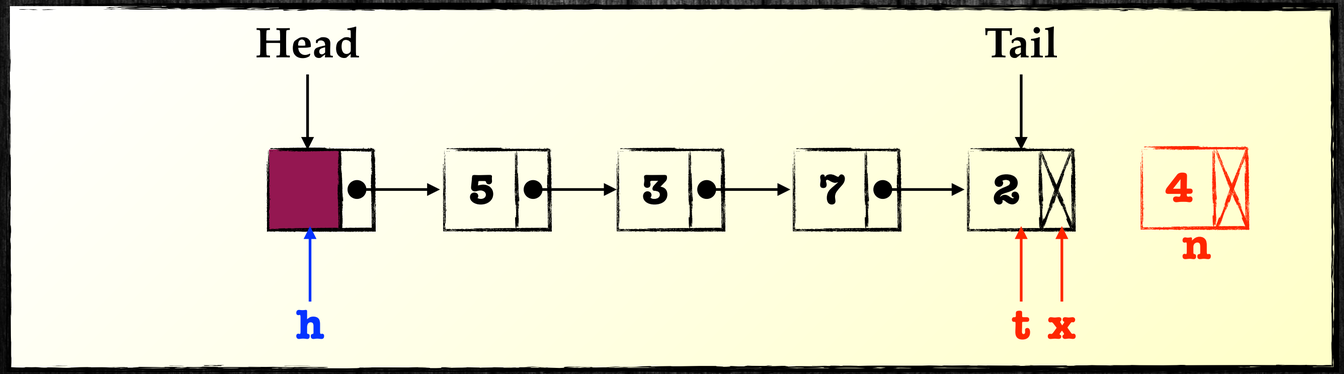
The Michael-Scott Queue Algorithm

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

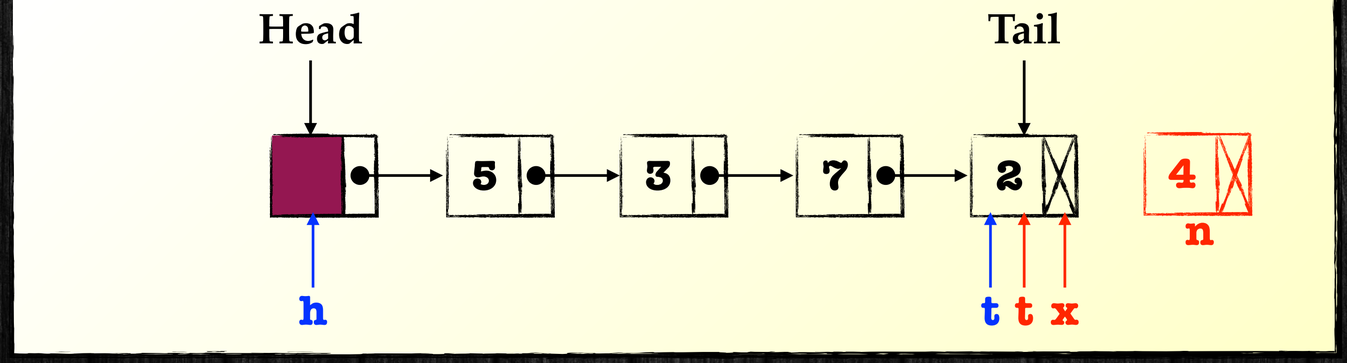
Linearization Policy

```
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

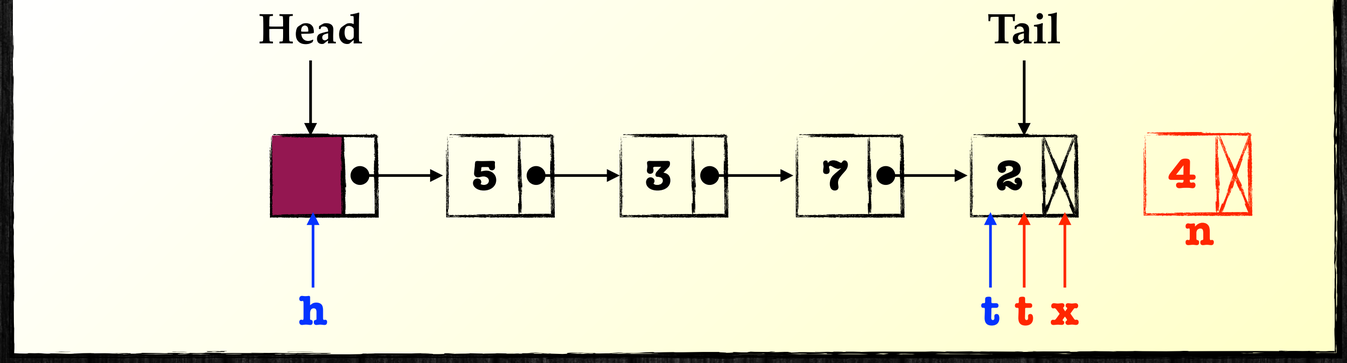
Linearization Policy

```
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)

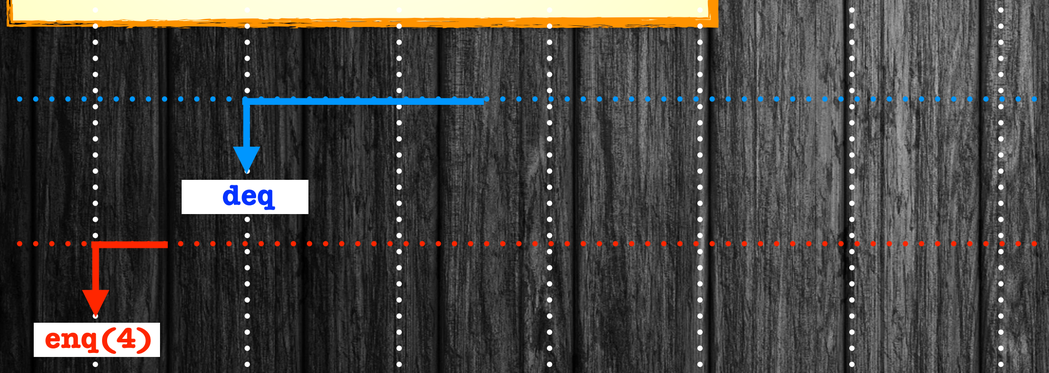


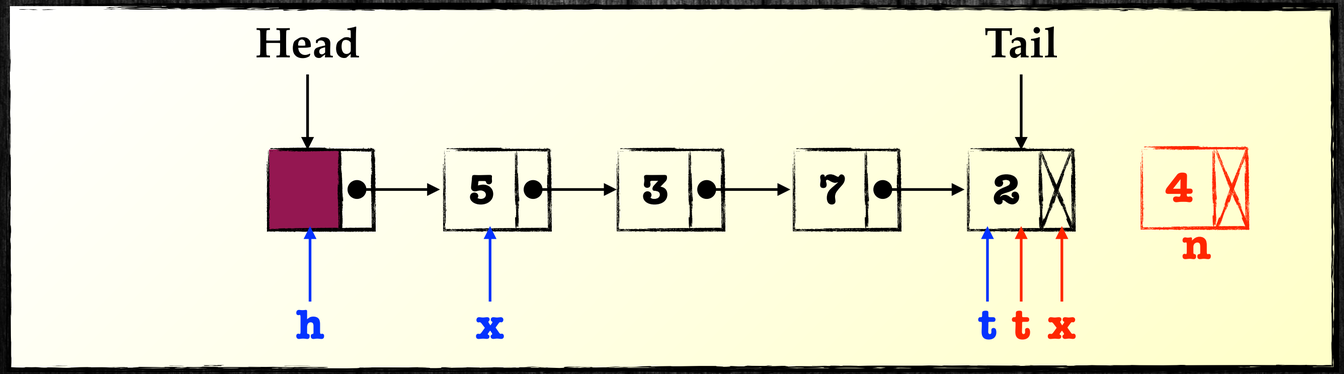
The Michael-Scott Queue Algorithm

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
```



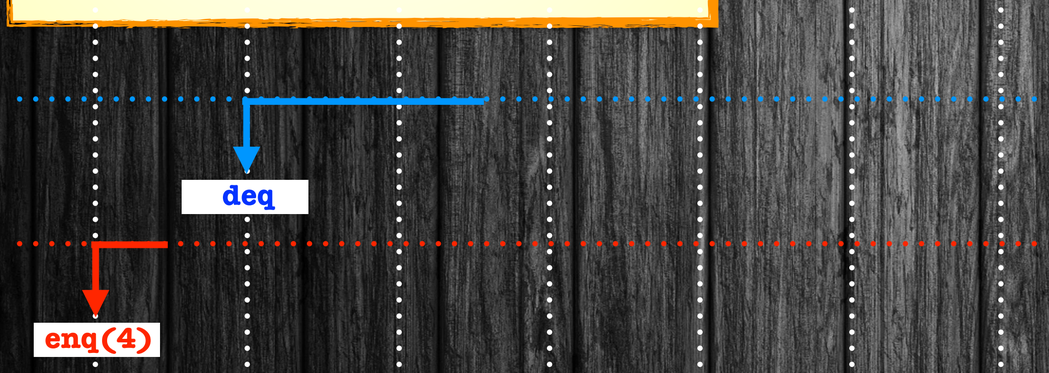


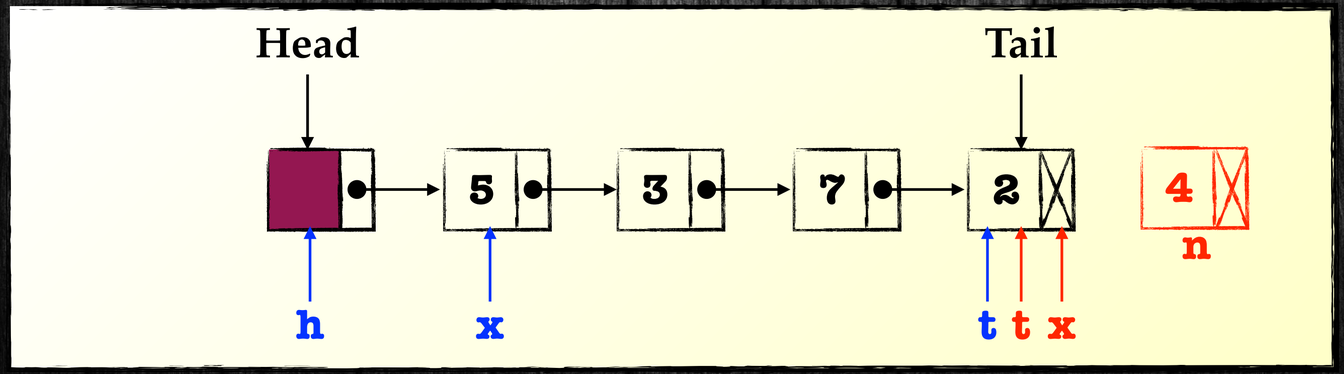
The Michael-Scott Queue Algorithm

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
```



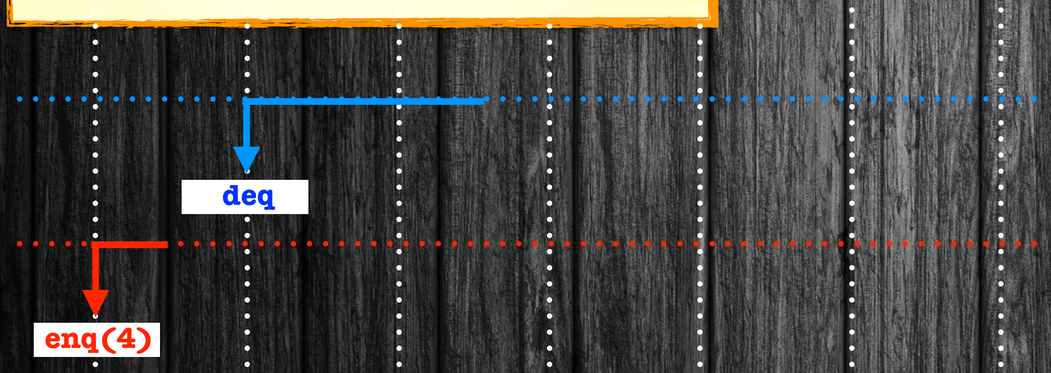


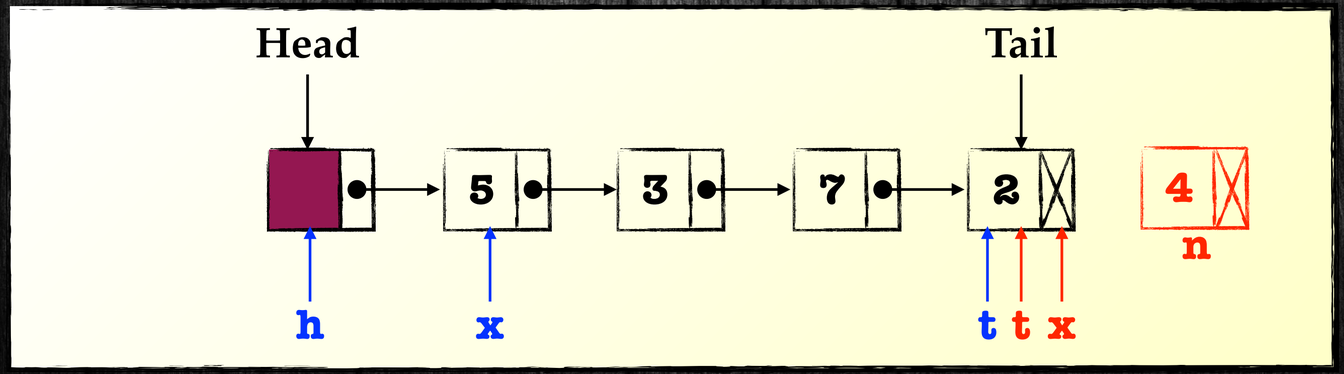
The Michael-Scott Queue Algorithm

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

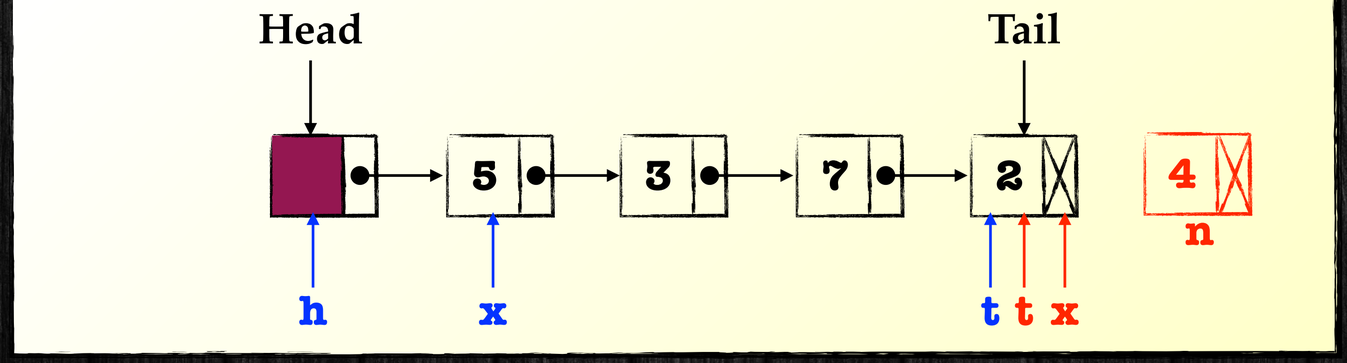
Linearization Policy

```
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

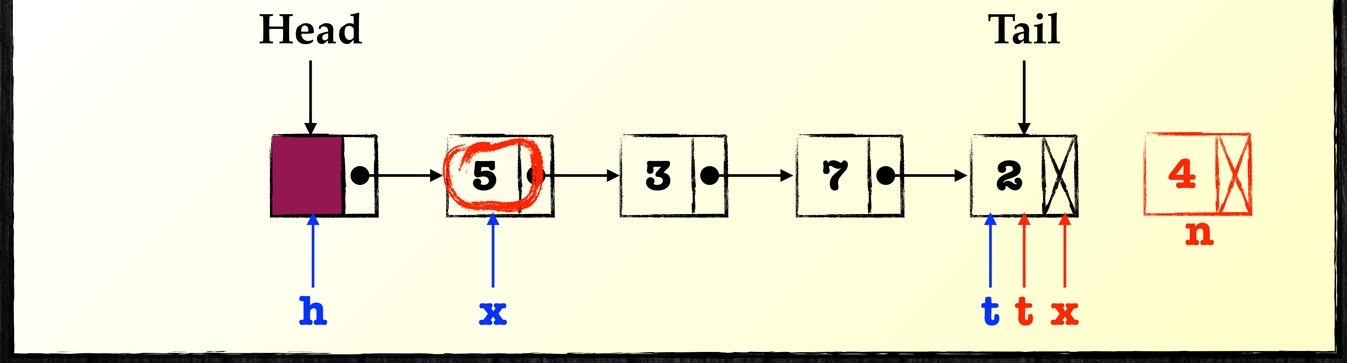
Linearization Policy

```
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

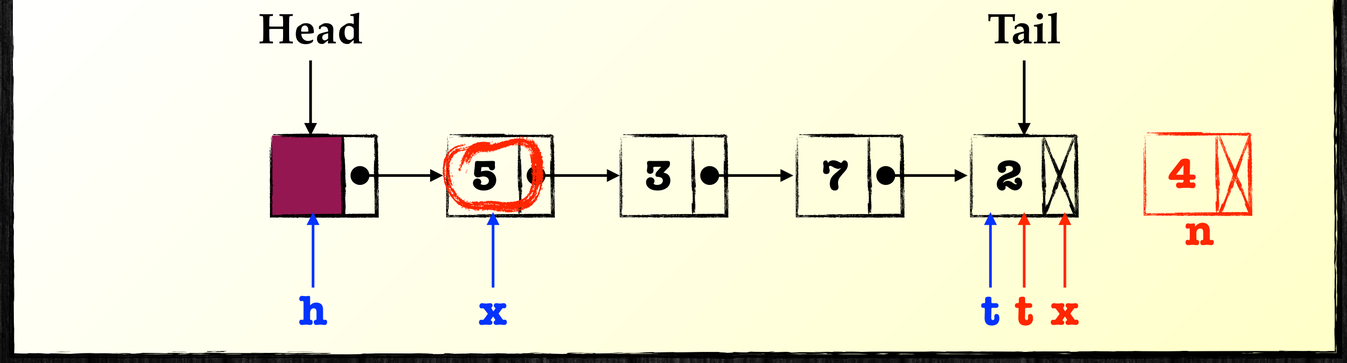
Linearization Policy

```
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

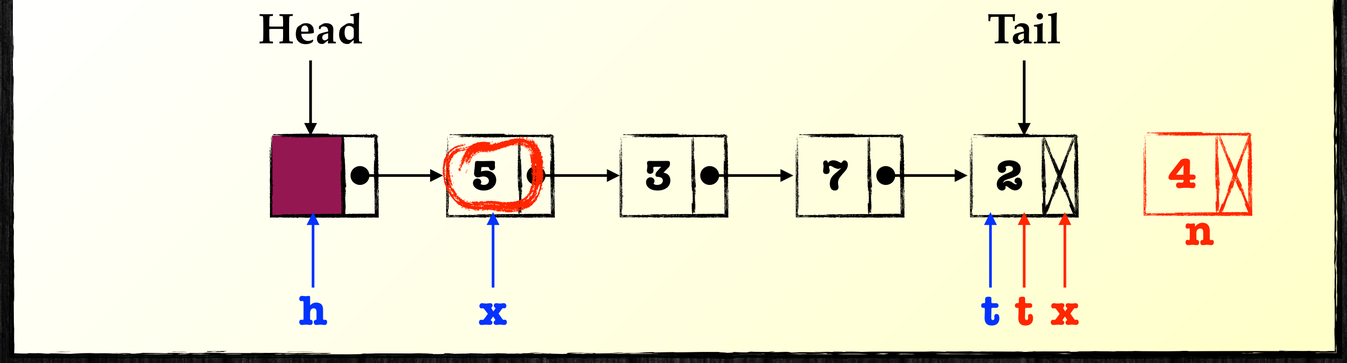
Linearization Policy

```
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

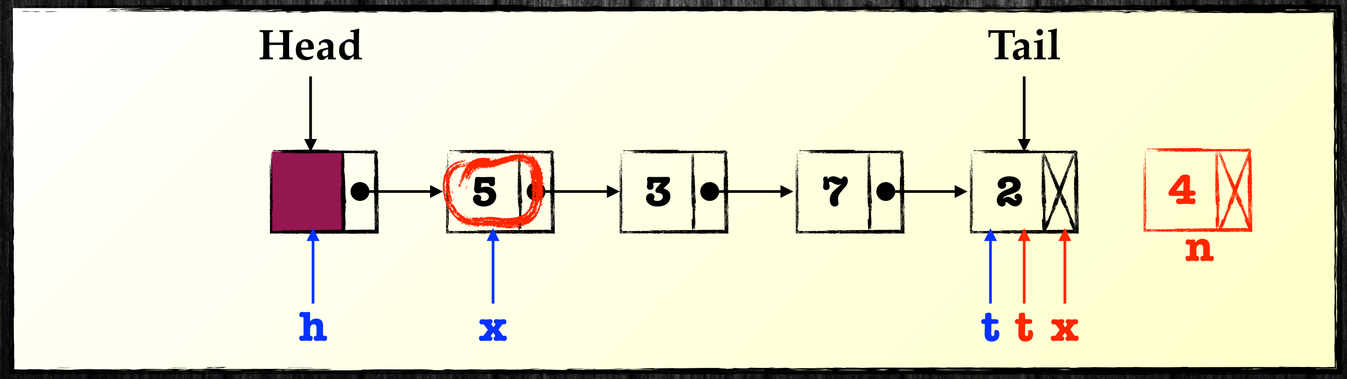
```
enq(k):
Node t,x
1  n = new Node(k,NULL)
2  while (true)
3    t = Tail
4    x = t.next
5    if (t == Tail)
6      if (x == NULL)
7        if (CAS (t.next,NULL,n))
8          CAS (Tail,t,n)
9          exit
10     else
11       CAS (Tail,t,x)
```

```
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)



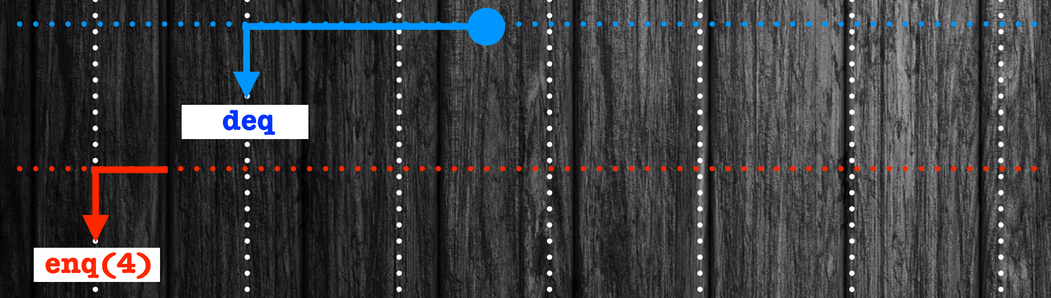
The Michael-Scott Queue Algorithm

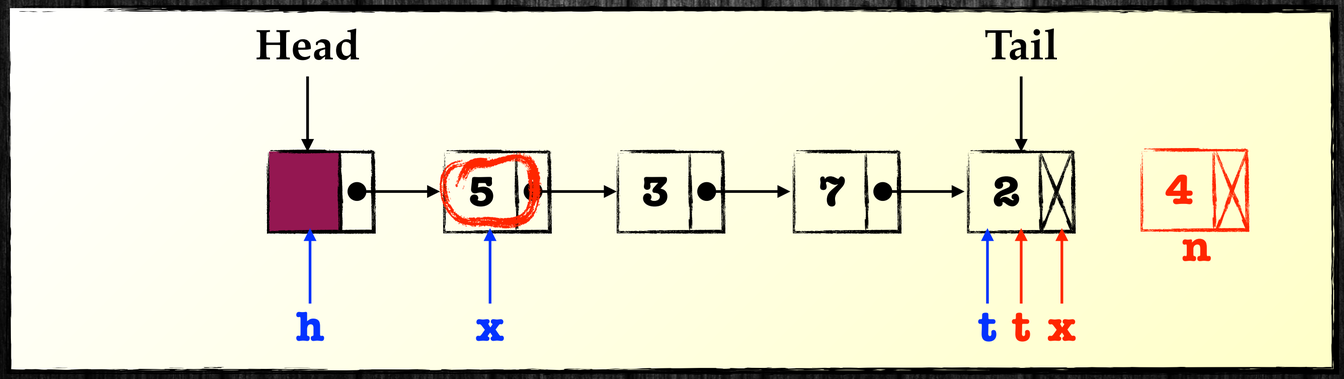
Linearization Policy

```
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



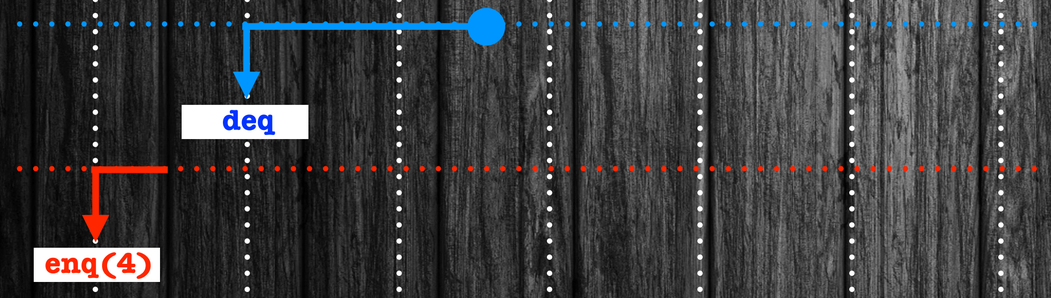


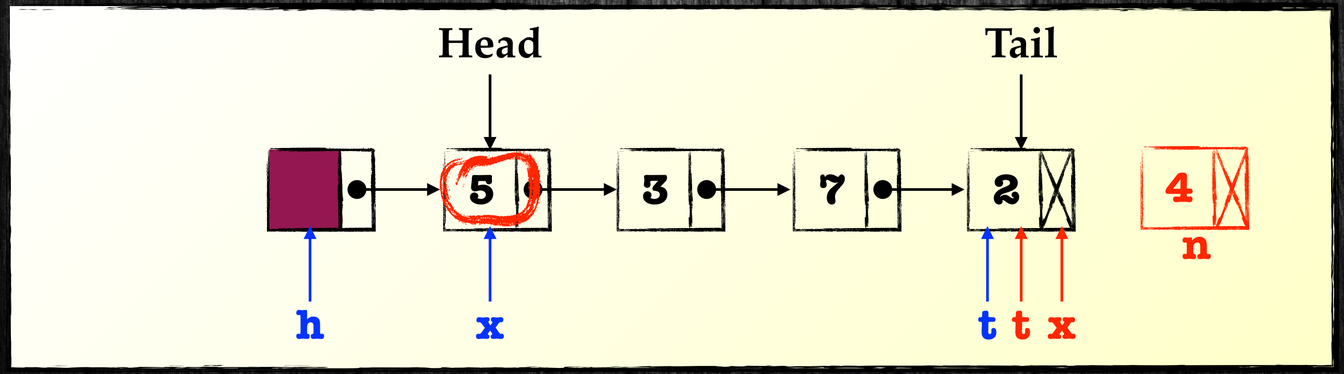
The Michael-Scott Queue Algorithm

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
```



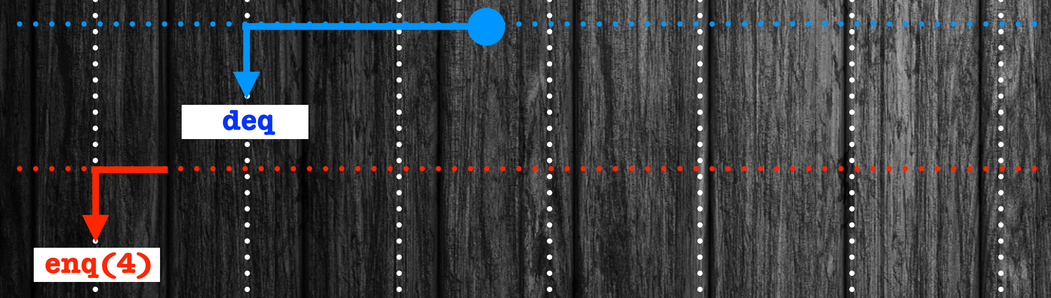


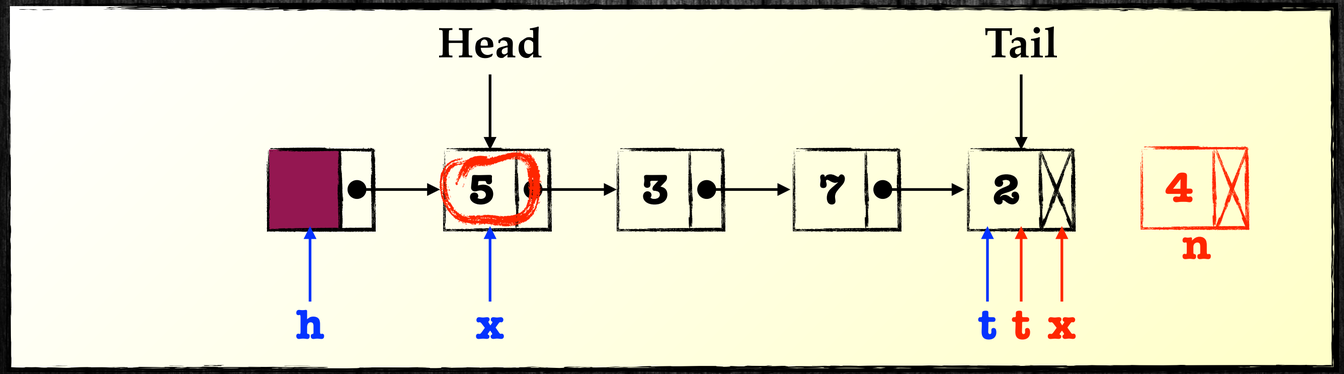
The Michael-Scott Queue Algorithm

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
```



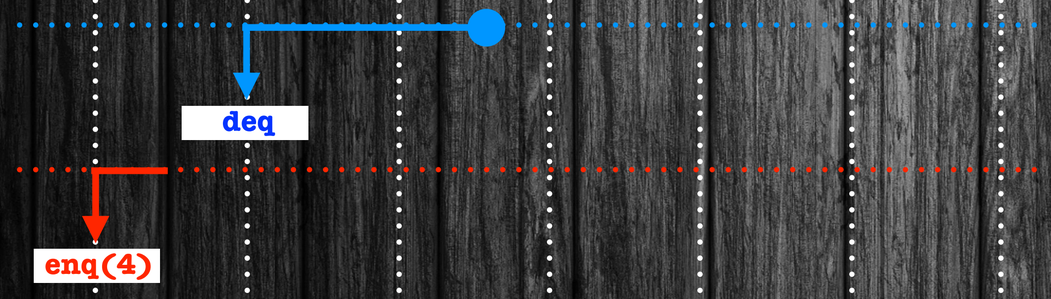


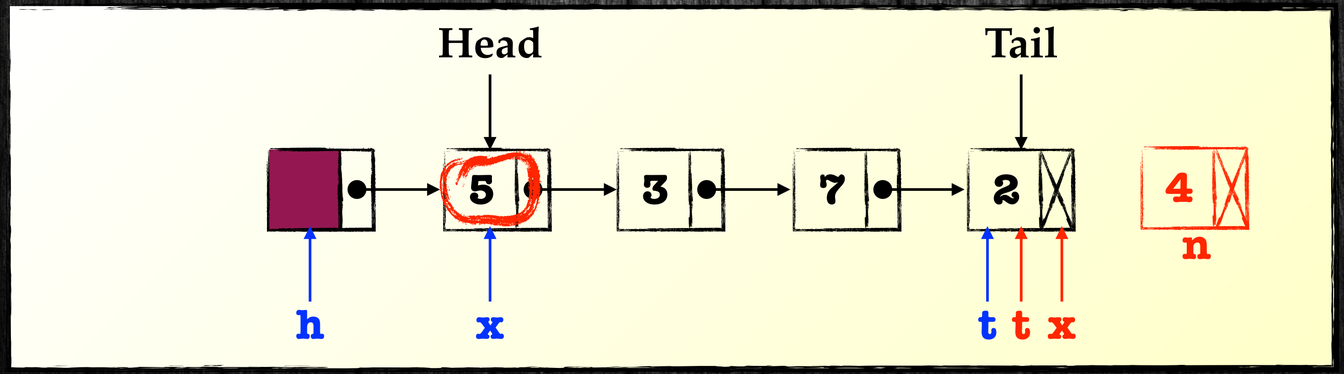
The Michael-Scott Queue Algorithm

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
```



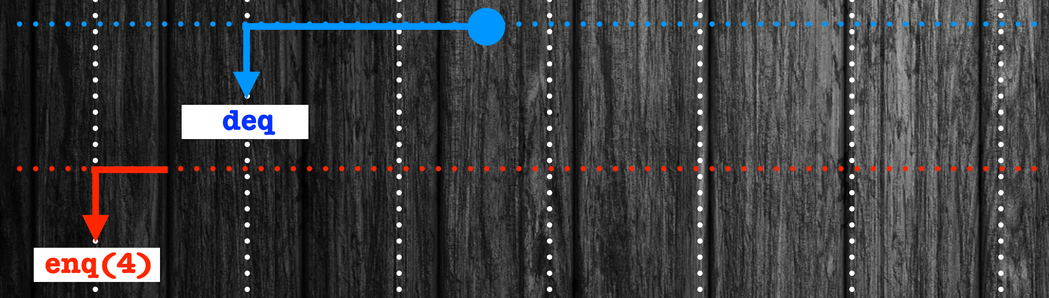


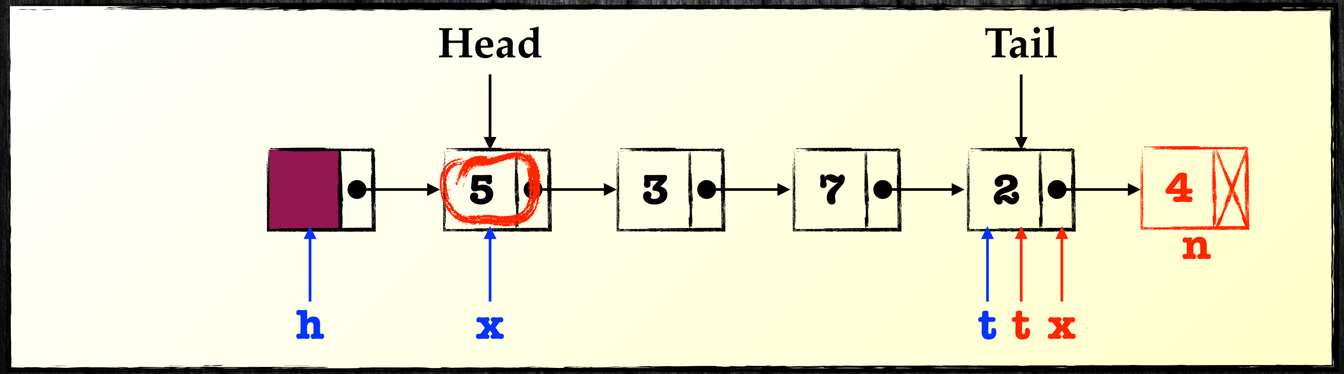
The Michael-Scott Queue Algorithm

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

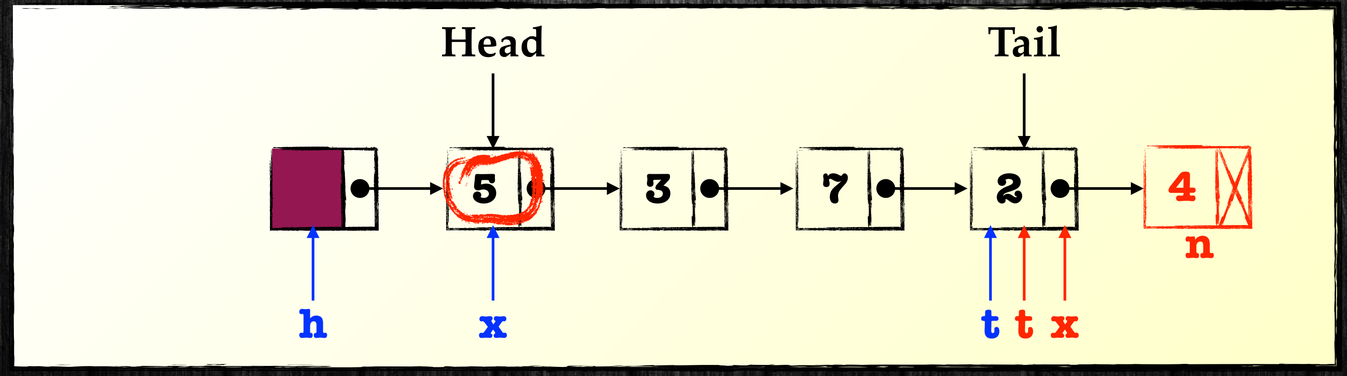
```
enq(k):
Node t,x
1  n = new Node(k,NULL)
2  while (true)
3    t = Tail
4    x = t.next
5    if (t == Tail)
6      if (x == NULL)
7        if (CAS (t.next,NULL,n))
8          CAS (Tail,t,n)
9          exit
10     else
11       CAS (Tail,t,x)
```

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

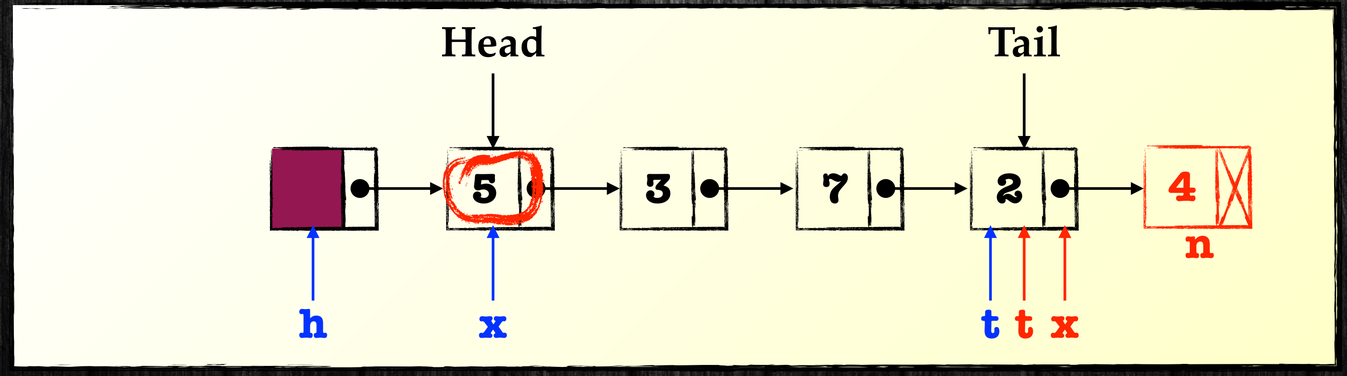
```
enq(k):
Node t,x
1  n = new Node(k,NULL)
2  while (true)
3    t = Tail
4    x = t.next
5    if (t == Tail)
6      if (x == NULL)
7        if (CAS (t.next,NULL,n))
8          CAS (Tail,t,n)
9          exit
10     else
11       CAS (Tail,t,x)
```

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)

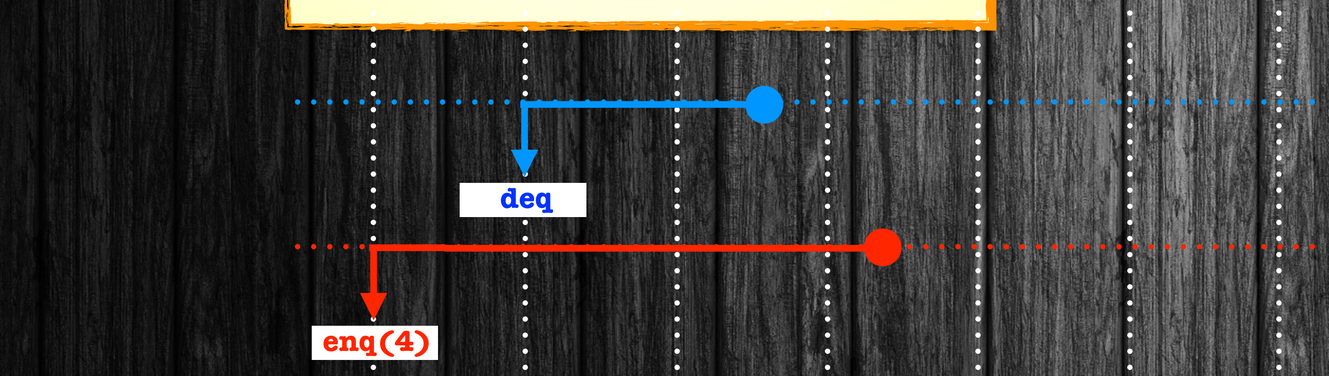


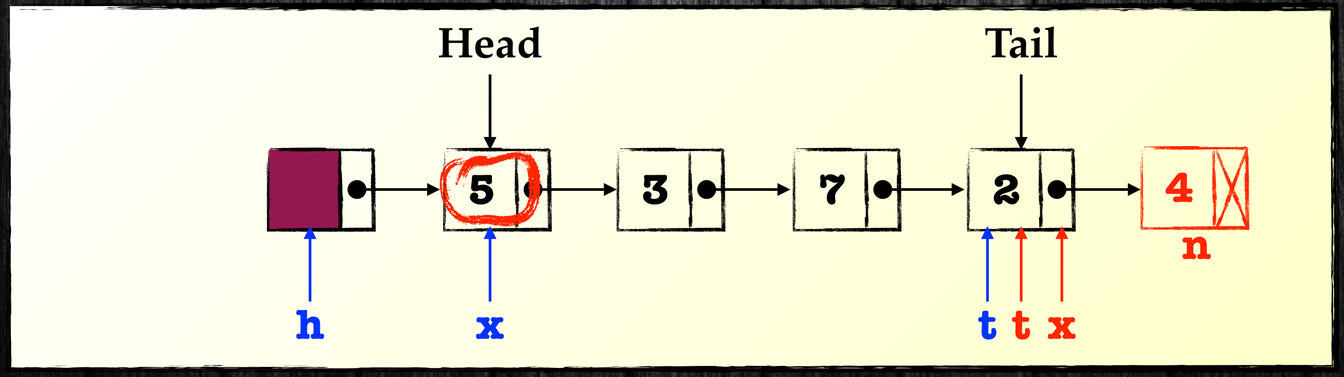
The Michael-Scott Queue Algorithm

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
```



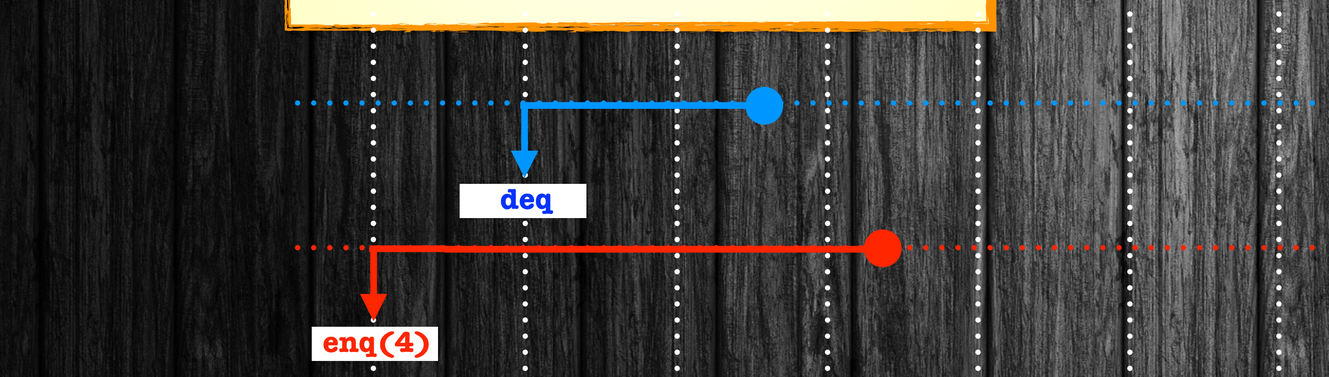


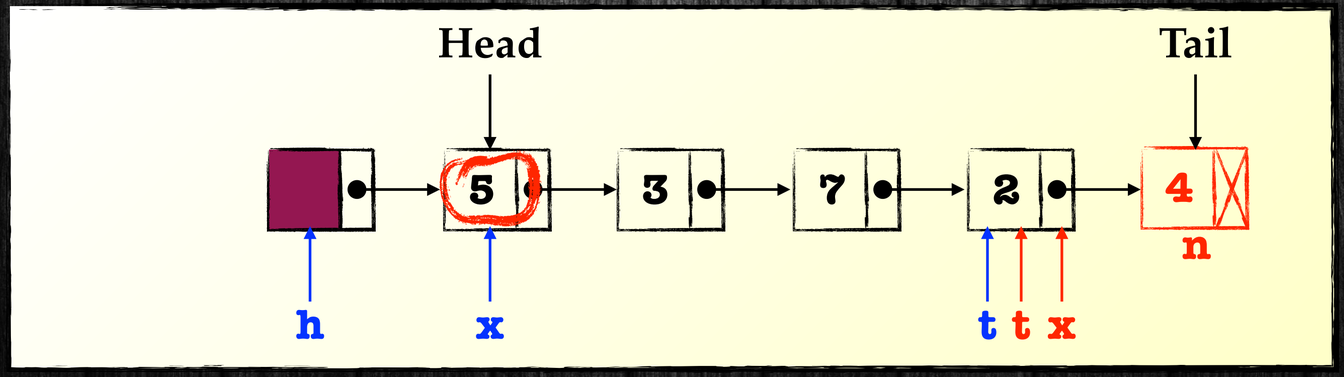
The Michael-Scott Queue Algorithm

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
```



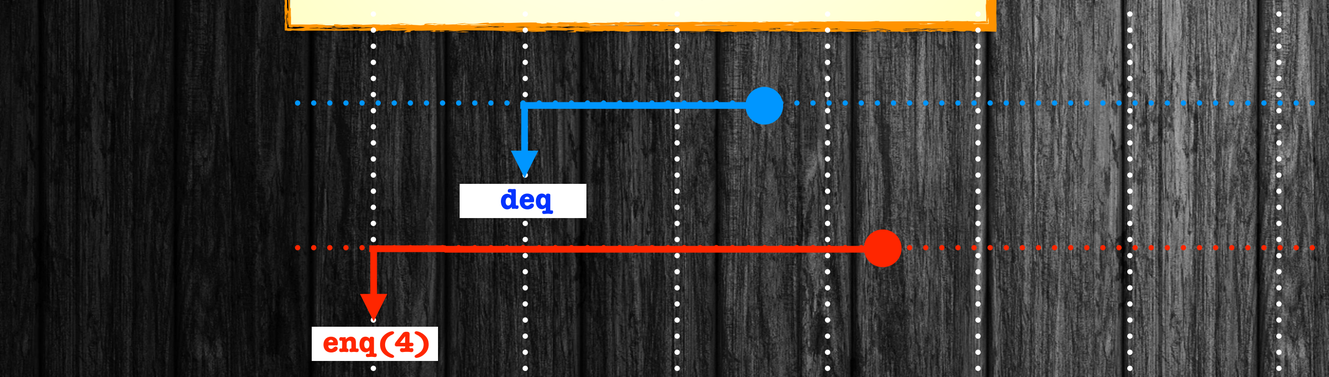


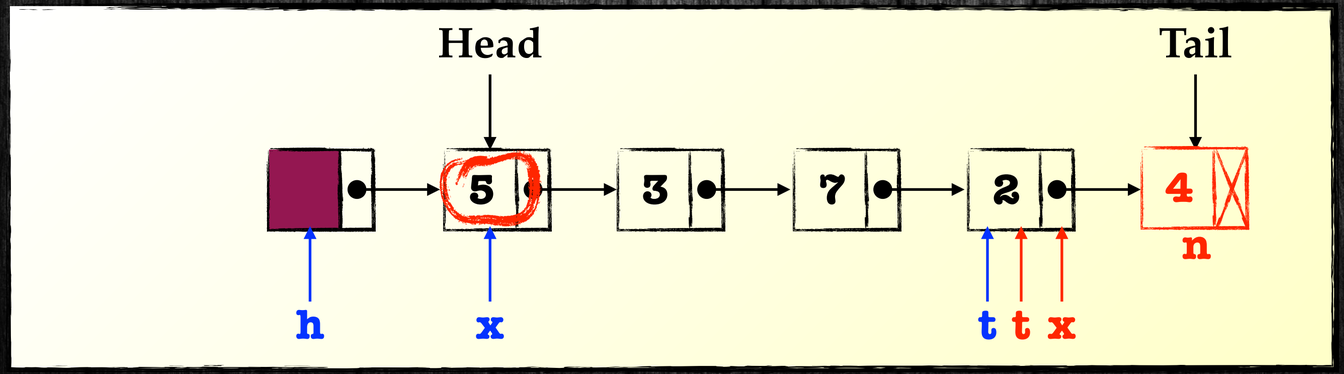
The Michael-Scott Queue Algorithm

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
```



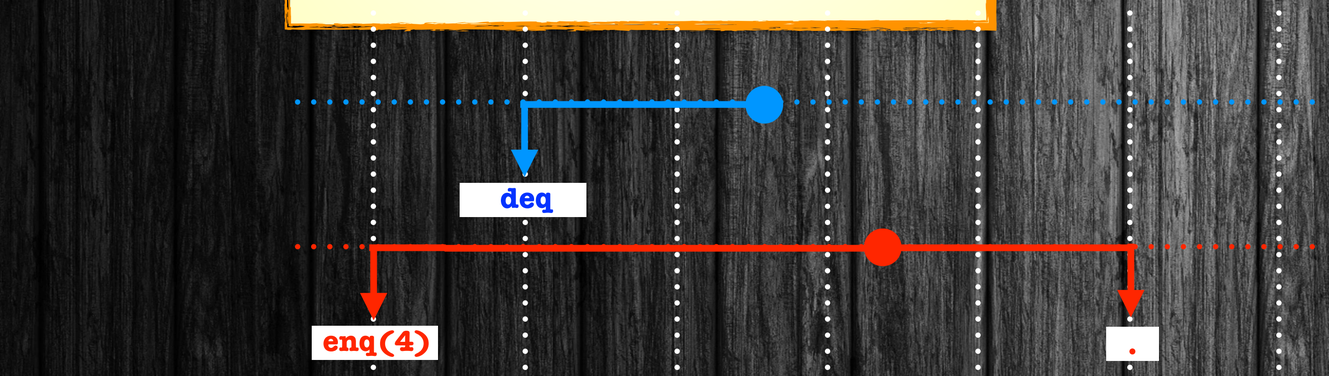


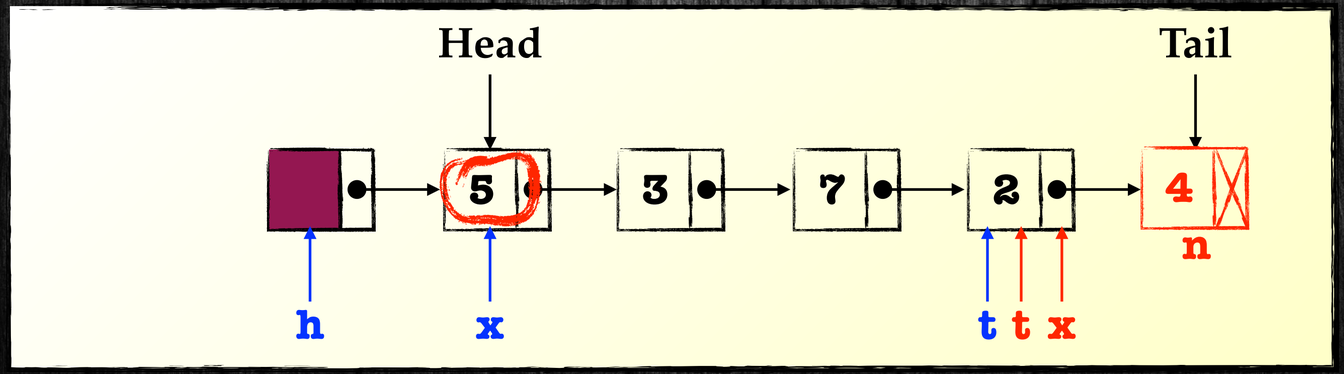
The Michael-Scott Queue Algorithm

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
```



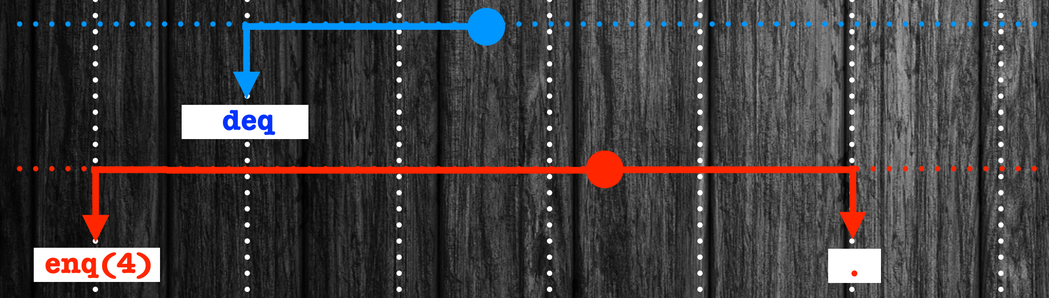


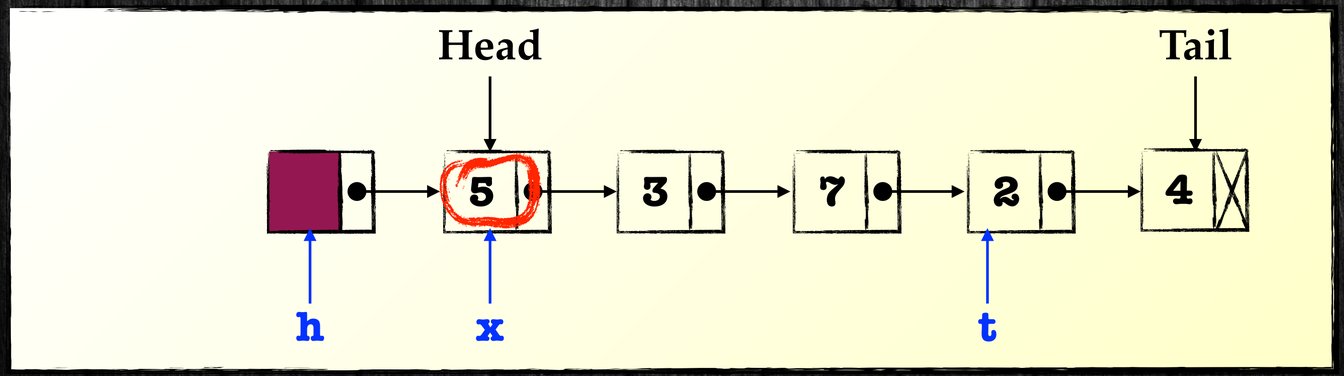
The Michael-Scott Queue Algorithm

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
```



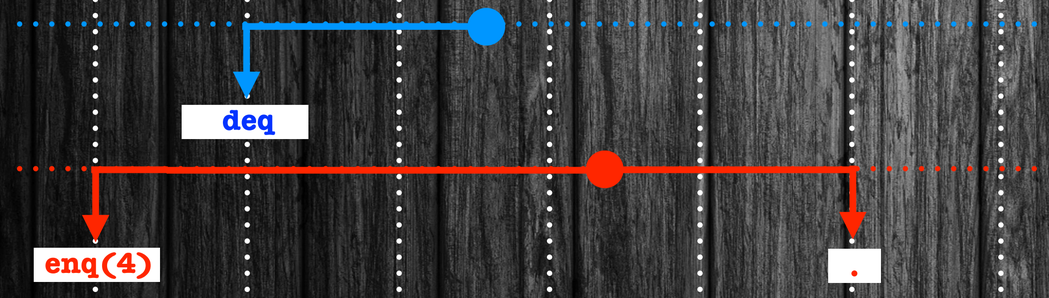


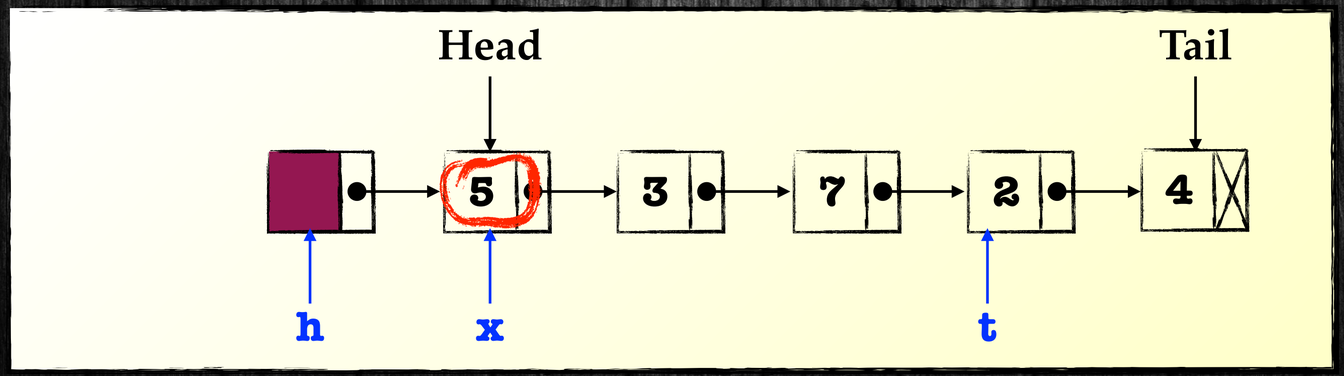
The Michael-Scott Queue Algorithm

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
```



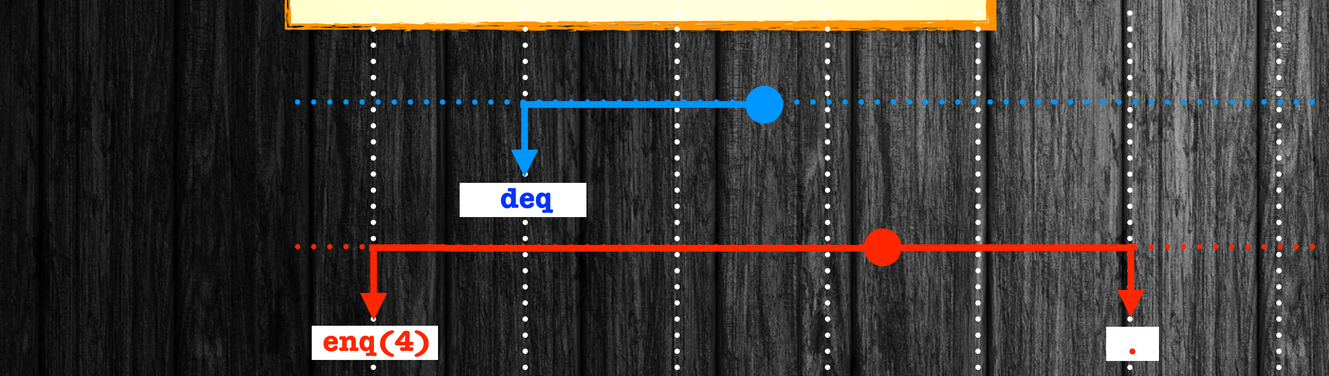


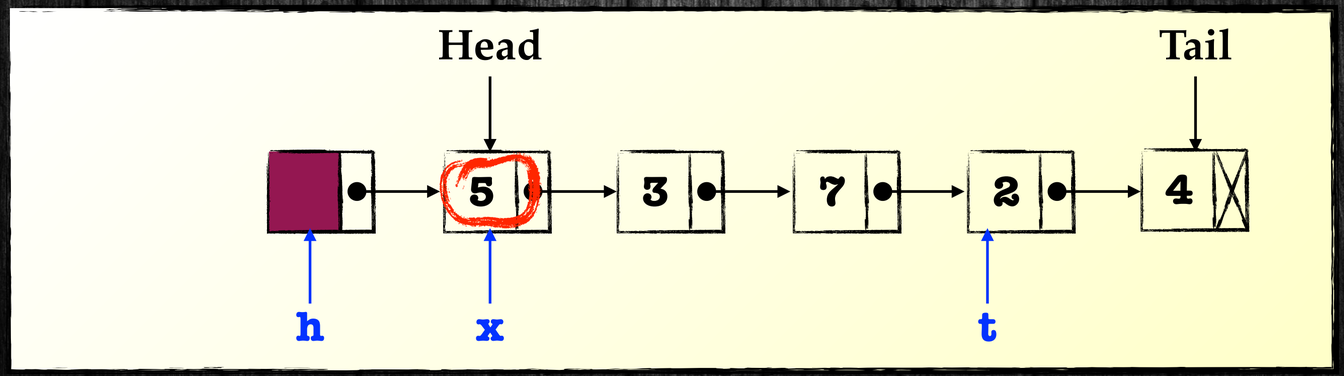
The Michael-Scott Queue Algorithm

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
```



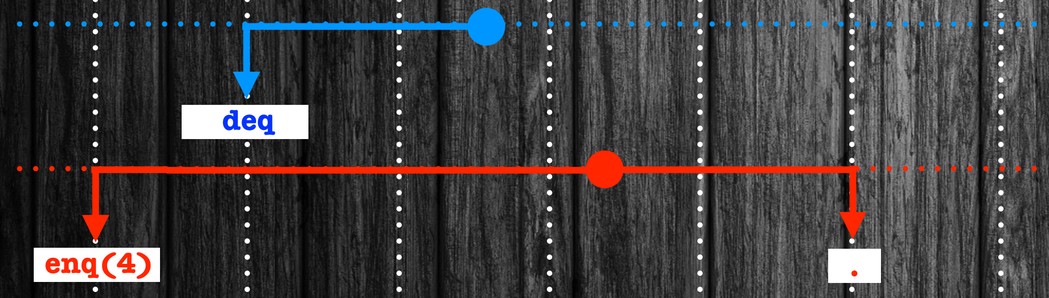


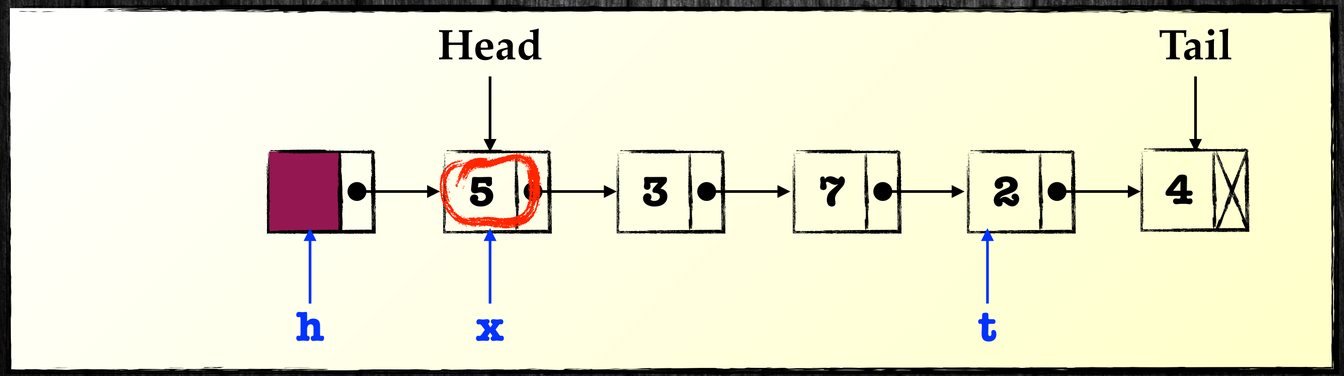
The Michael-Scott Queue Algorithm

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
```



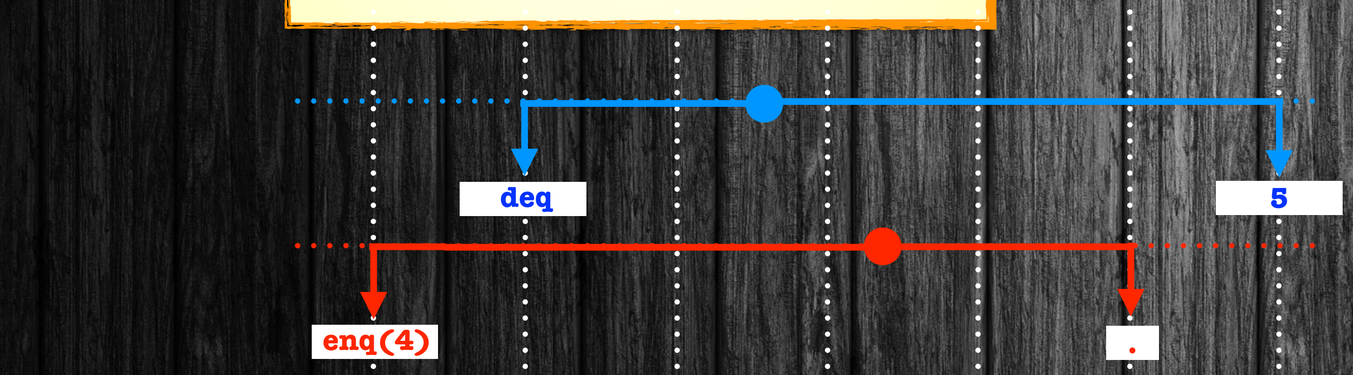


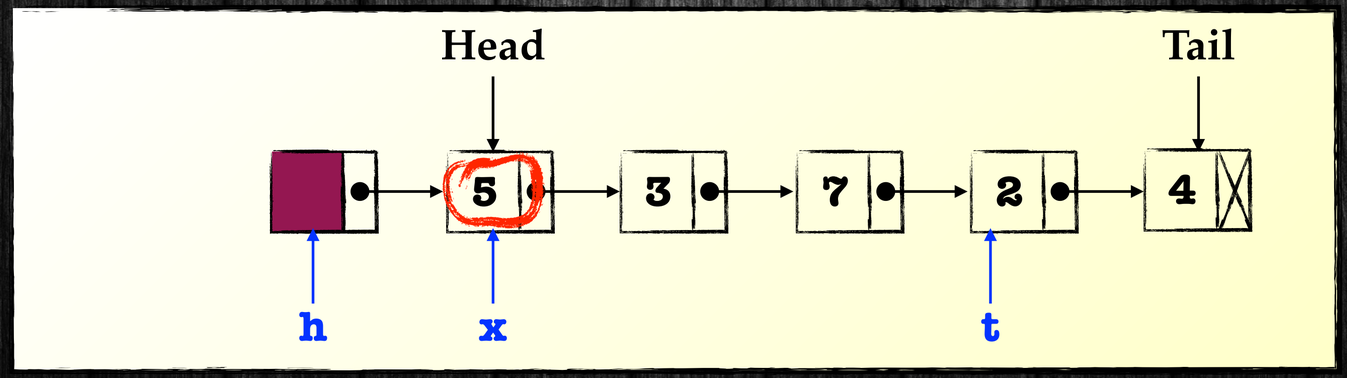
The Michael-Scott Queue Algorithm

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
```



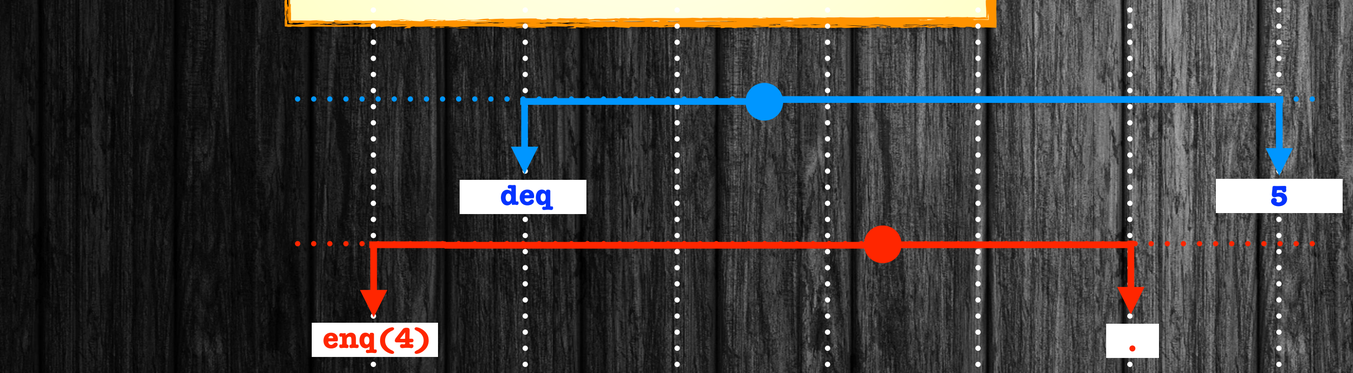


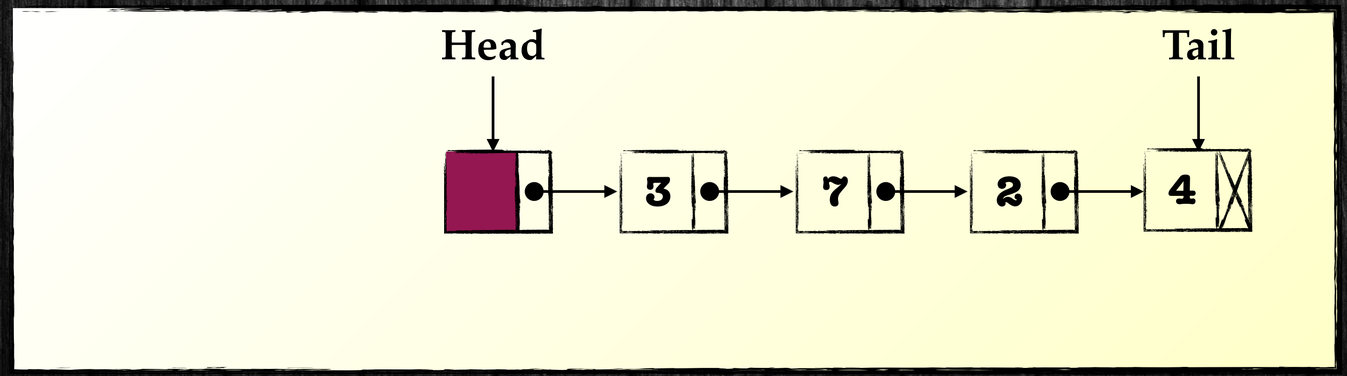
The Michael-Scott Queue Algorithm

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
```



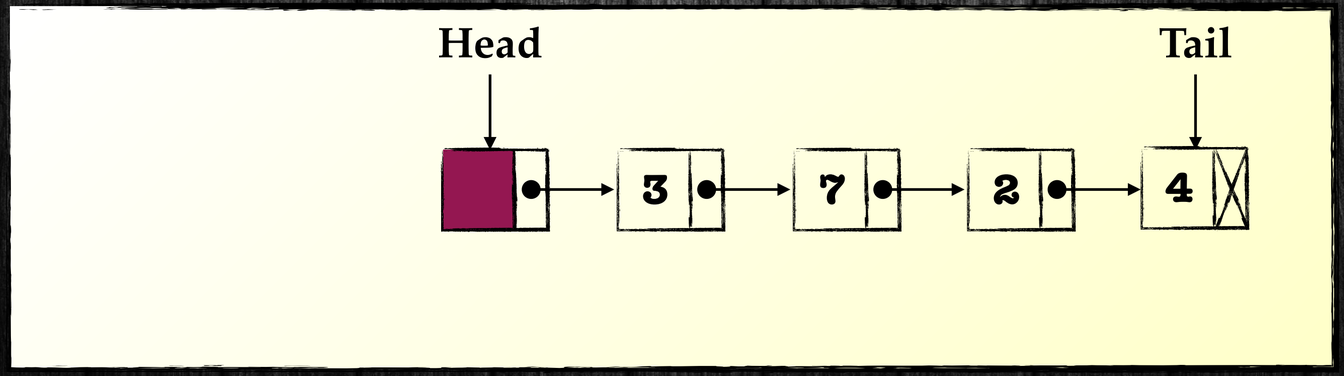


The Michael-Scott Queue Algorithm

The ABA Problem

```
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
```



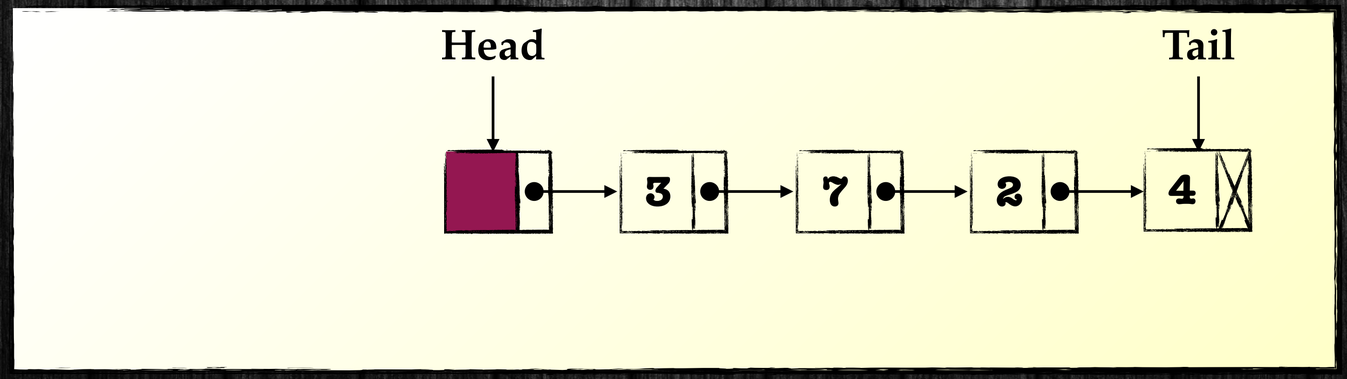
The Michael-Scott Queue Algorithm

The ABA Problem

```
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
```



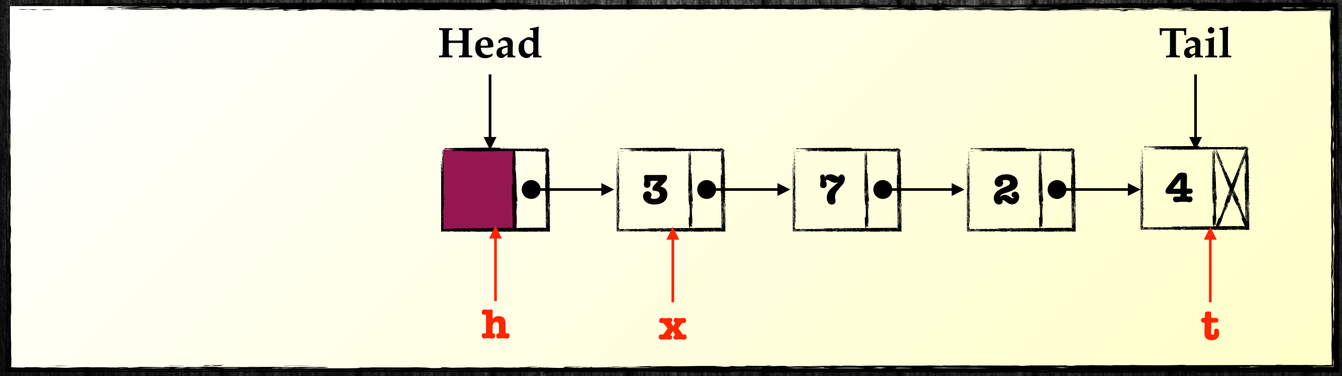
The Michael-Scott Queue Algorithm

The ABA Problem

```
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
```



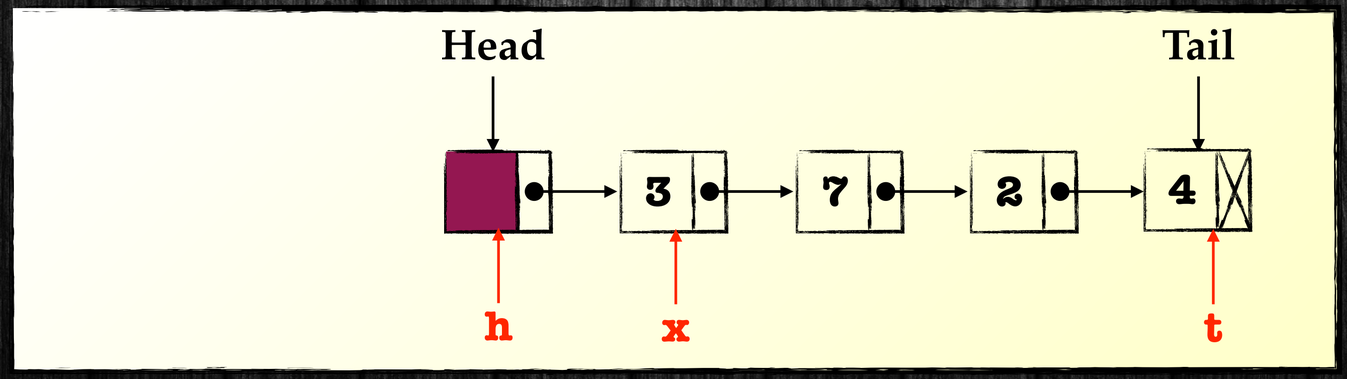
The Michael-Scott Queue Algorithm

The ABA Problem

```
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

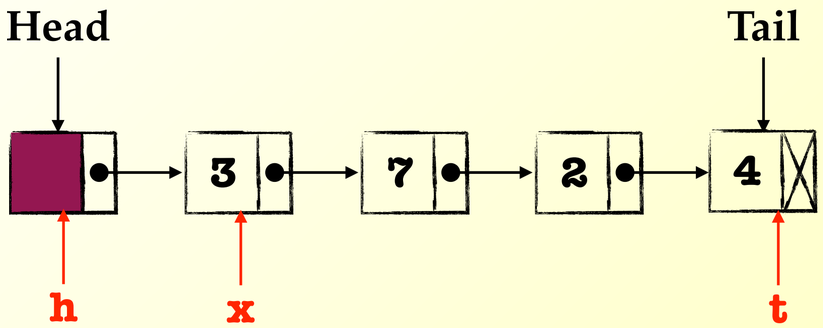
The ABA Problem

```
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
```

sleep

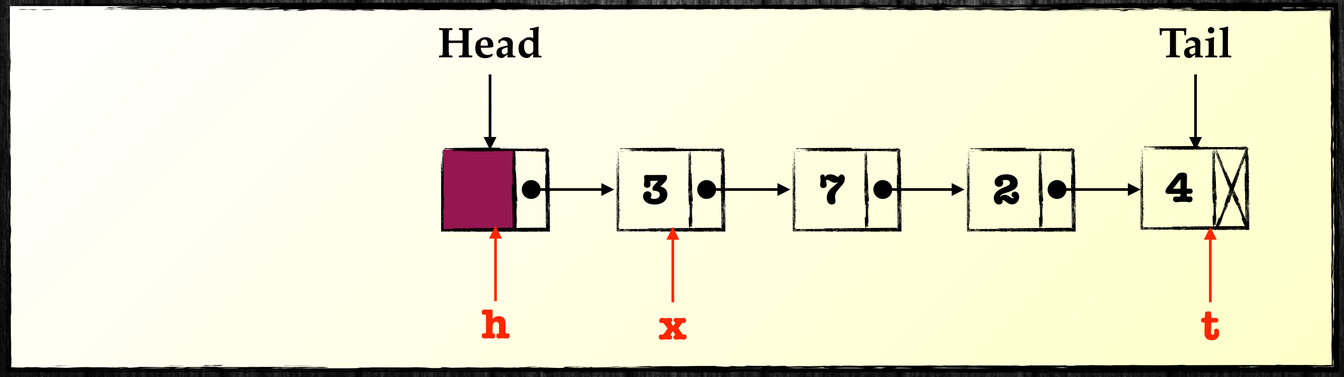


The Michael-Scott Queue Algorithm

The ABA Problem

```
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
```



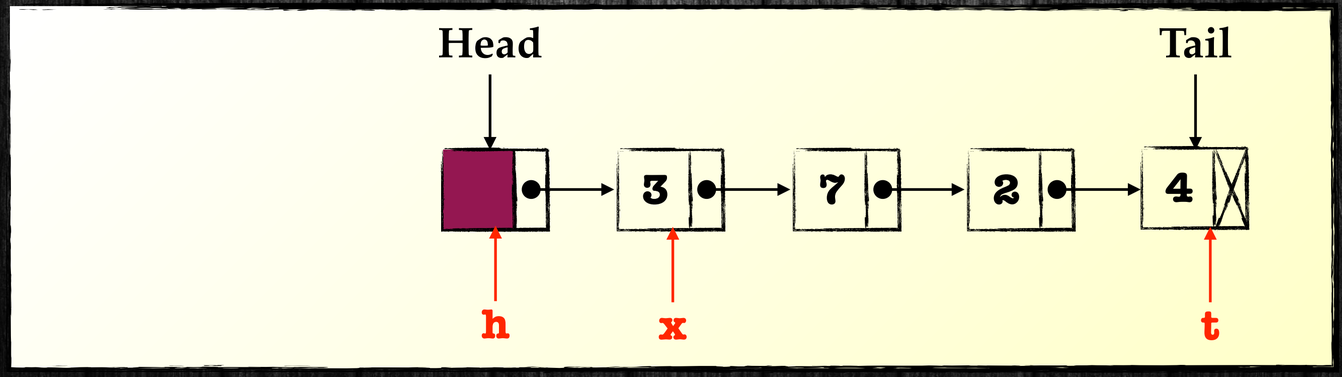
The Michael-Scott Queue Algorithm

The ABA Problem

```
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
```



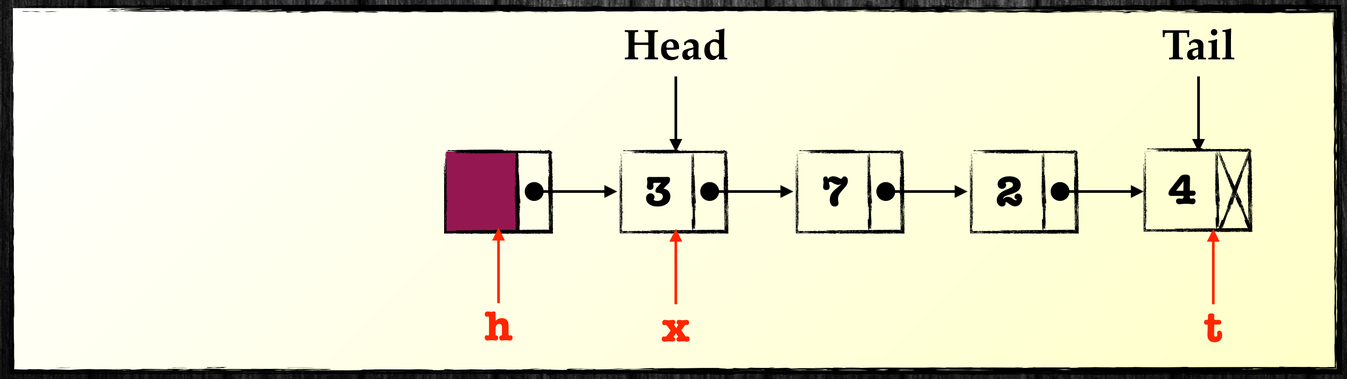
The Michael-Scott Queue Algorithm

The ABA Problem

```
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
```



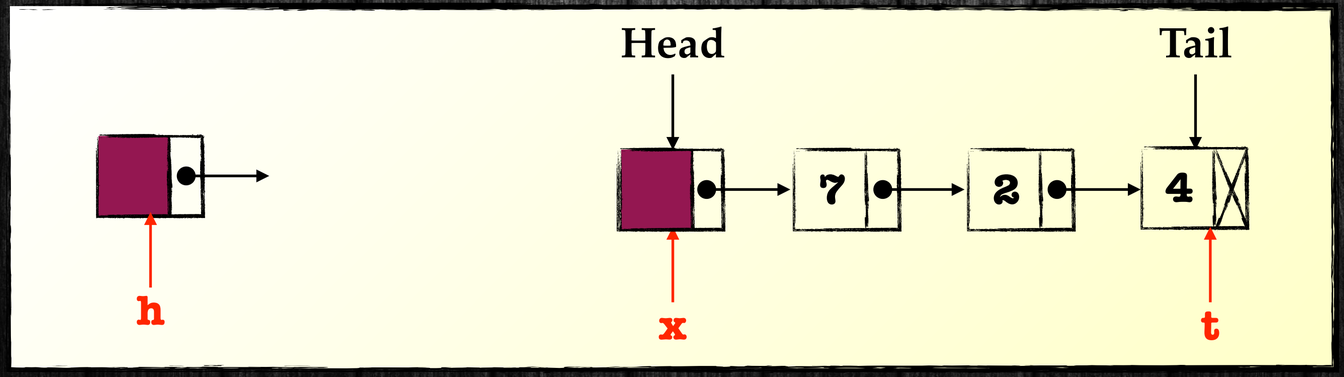
The Michael-Scott Queue Algorithm

The ABA Problem

```
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
```

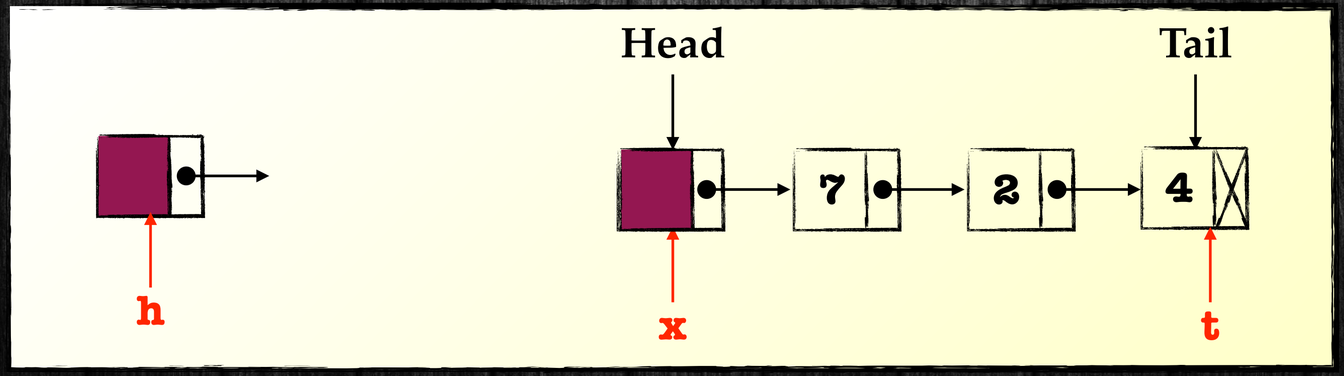


The Michael-Scott Queue Algorithm

The ABA Problem

```
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
```



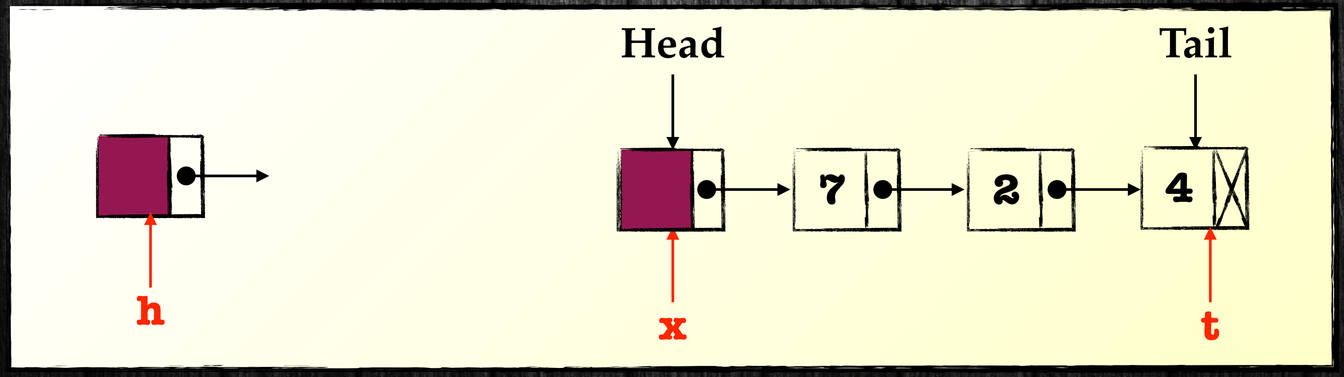
The Michael-Scott Queue Algorithm

The ABA Problem

```
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
```



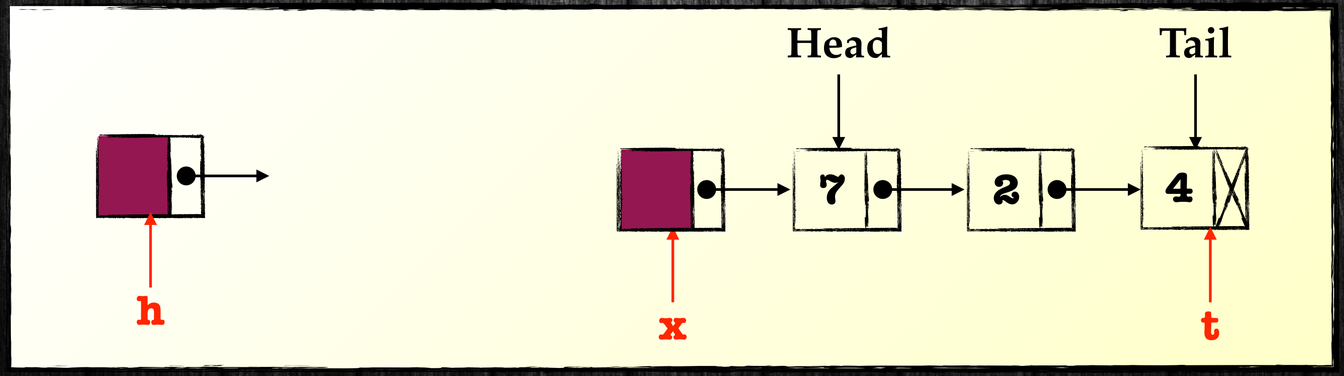
The Michael-Scott Queue Algorithm

The ABA Problem

```
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
```



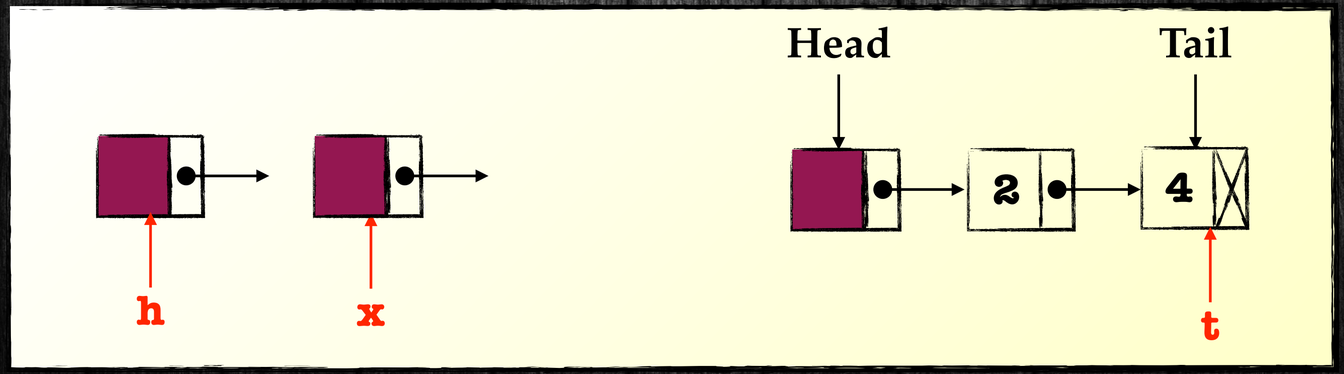
The Michael-Scott Queue Algorithm

The ABA Problem

```
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
```

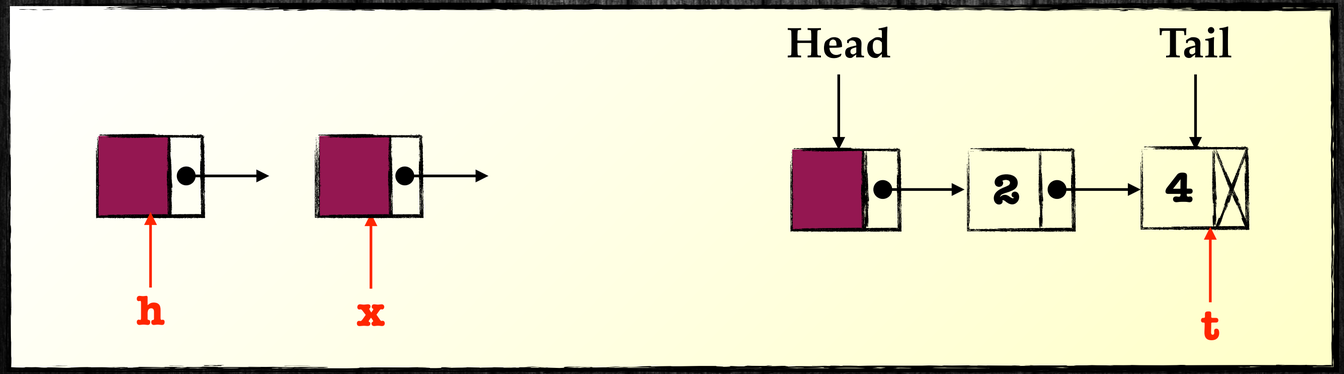


The Michael-Scott Queue Algorithm

```
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 ABA Problem

```
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
```



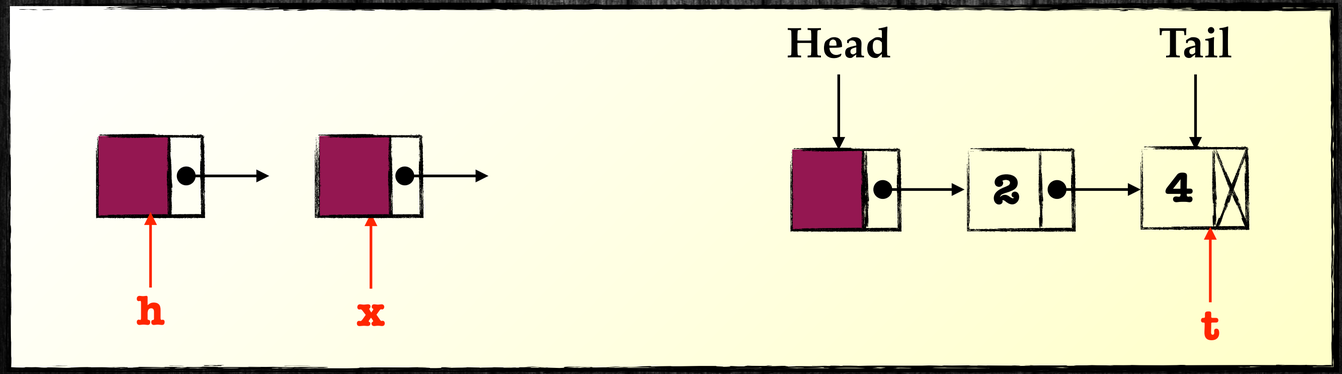
The Michael-Scott Queue Algorithm

The ABA Problem

```
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
```



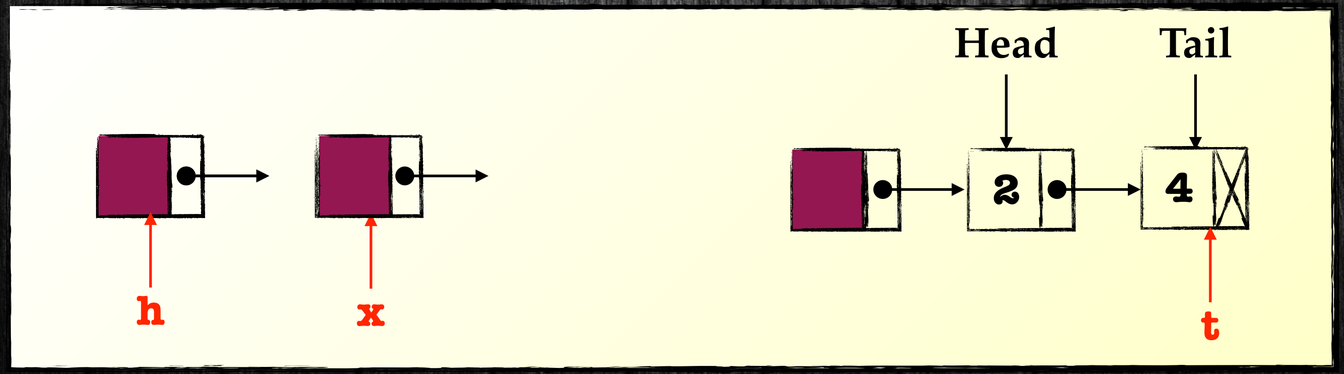
The Michael-Scott Queue Algorithm

The ABA Problem

```
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
```

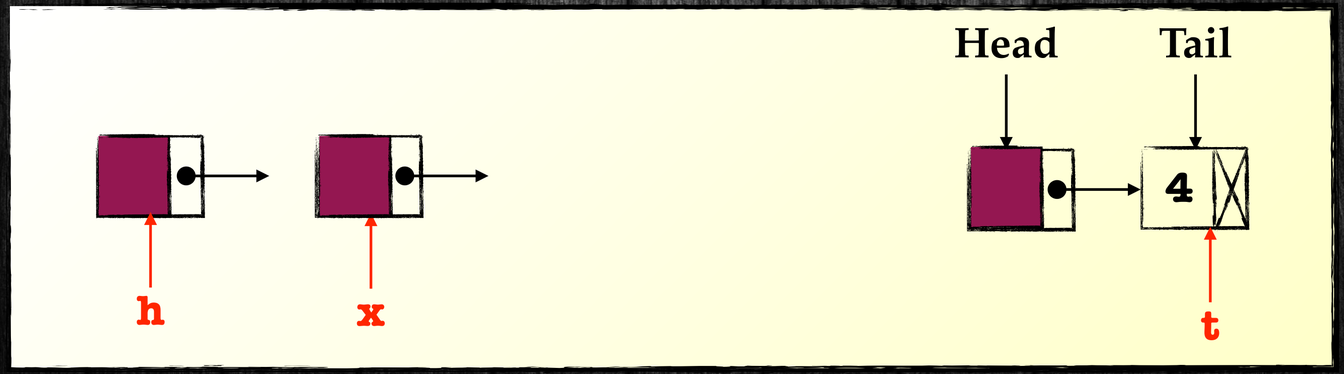


The Michael-Scott Queue Algorithm

The ABA Problem

```
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)
```

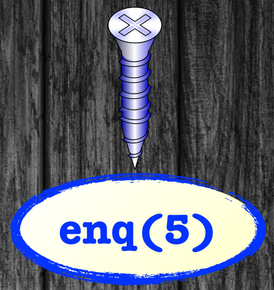
```
deq(k):
Node h,t,x
Integer v
 1  while (true)
 2    h = Head
 3    t = Tail
 4    x = h.next
 5    if (h == Head)
 6      if (h == t)
 7        if (x == NULL)
 8          return *
 9          exit
10        CAS (Tail,t,x)
11      else
12        v = x.key
13        if (CAS (Head,h,x))
14         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



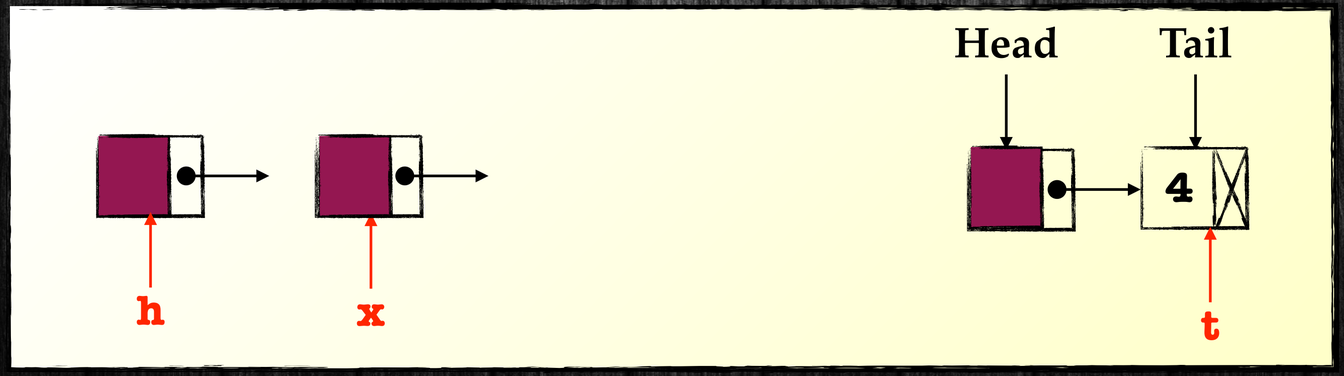
The Michael-Scott Queue Algorithm

The ABA Problem

```
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)
```



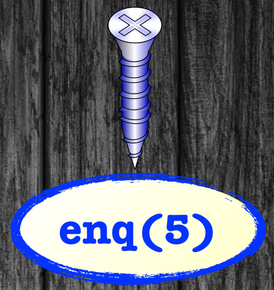

```
deq(k):
Node h,t,x
Integer v
1  while (true)
2    h = Head
3    t = Tail
4    x = h.next
5    if (h == Head)
6      if (h == t)
7        if (x == NULL)
8          return *
9          exit
10       CAS (Tail,t,x)
11     else
12       v = x.key
13       if (CAS (Head,h,x))
14         return v
15  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



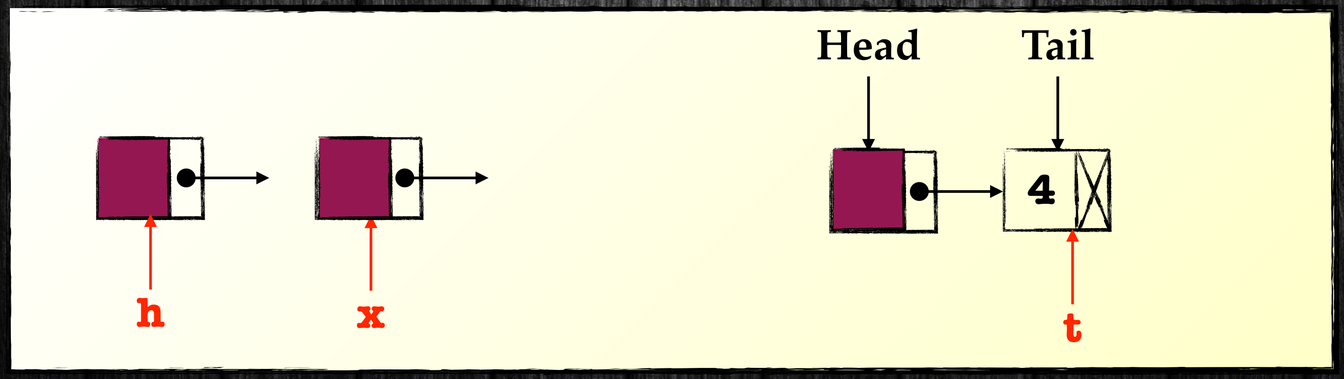
The Michael-Scott Queue Algorithm

The ABA Problem

```
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)
```



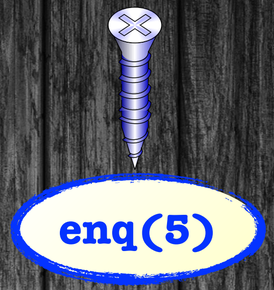

```
deq(k):
Node h,t,x
Integer v
1  while (true)
2    h = Head
3    t = Tail
4    x = h.next
5    if (h == Head)
6      if (h == t)
7        if (x == NULL)
8          return *
9          exit
10       CAS (Tail,t,x)
11     else
12       v = x.key
13       if (CAS (Head,h,x))
14         return v
15  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



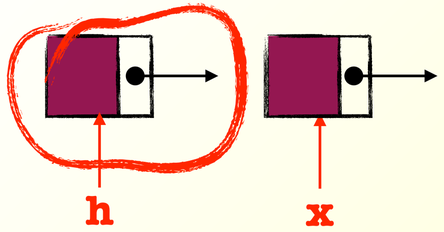
The Michael-Scott Queue Algorithm

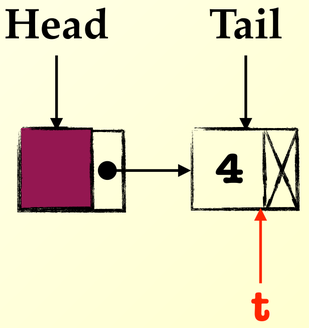
The ABA Problem

```
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
```

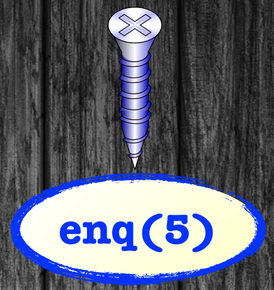




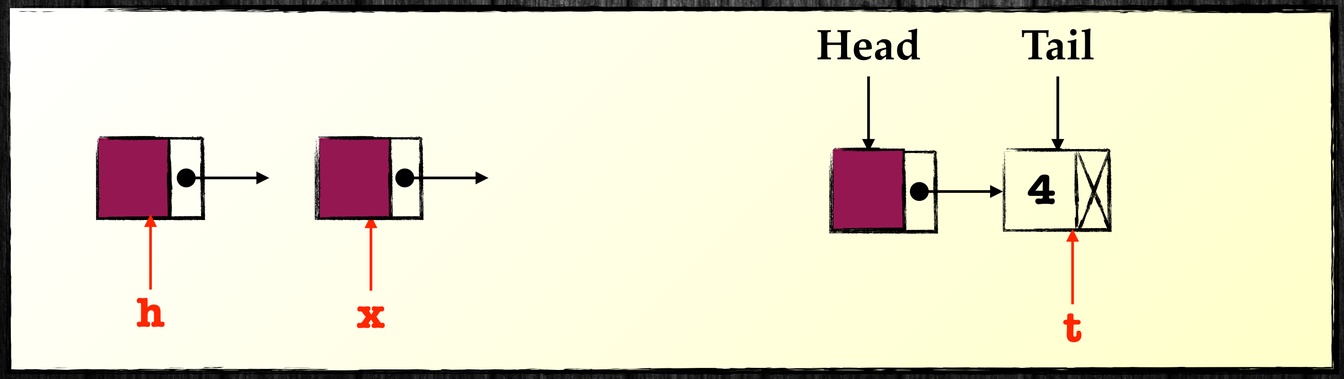
The Michael-Scott Queue Algorithm

The ABA Problem

```
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)
```



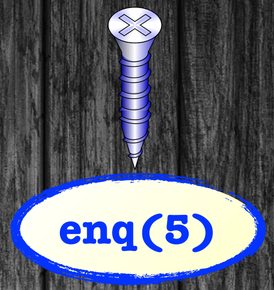

```
deq(k):
Node h,t,x
Integer v
1  while (true)
2    h = Head
3    t = Tail
4    x = h.next
5    if (h == Head)
6      if (h == t)
7        if (x == NULL)
8          return *
9          exit
10       CAS (Tail,t,x)
11     else
12       v = x.key
13       if (CAS (Head,h,x))
14         return v
15  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



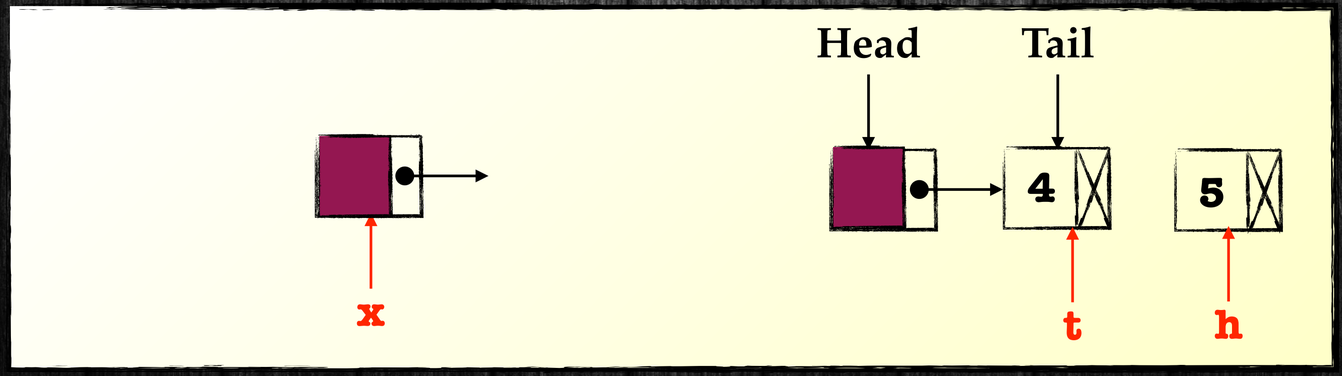
The Michael-Scott Queue Algorithm

The ABA Problem

```
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)
```



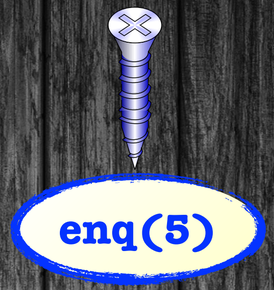

```
deq(k):
Node h,t,x
Integer v
1  while (true)
2    h = Head
3    t = Tail
4    x = h.next
5    if (h == Head)
6      if (h == t)
7        if (x == NULL)
8          return *
9          exit
10       CAS (Tail,t,x)
11     else
12       v = x.key
13       if (CAS (Head,h,x))
14         return v
15  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



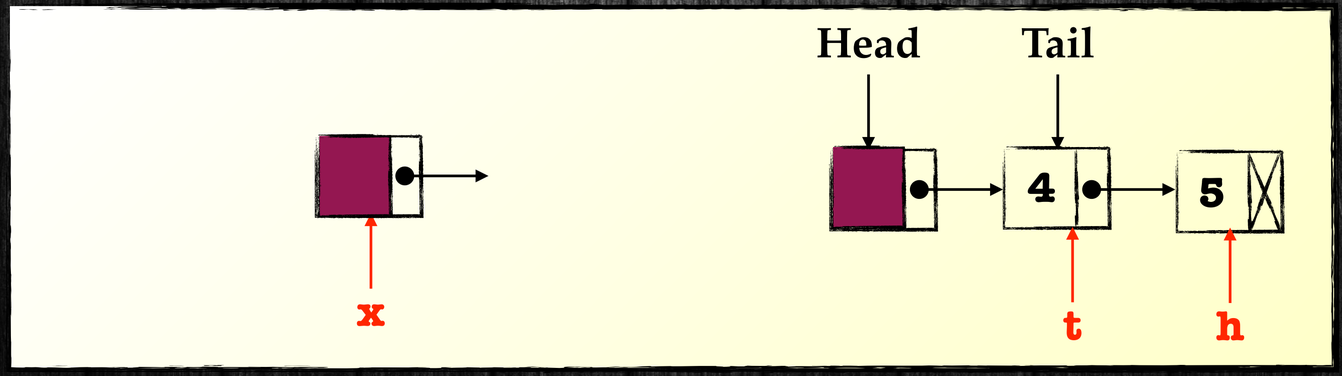
The Michael-Scott Queue Algorithm

The ABA Problem

```
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)
```



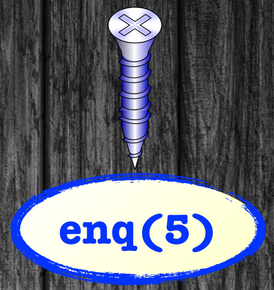

```
deq(k):
Node h,t,x
Integer v
1  while (true)
2    h = Head
3    t = Tail
4    x = h.next
5    if (h == Head)
6      if (h == t)
7        if (x == NULL)
8          return *
9          exit
10       CAS (Tail,t,x)
11     else
12       v = x.key
13       if (CAS (Head,h,x))
14         return v
15  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



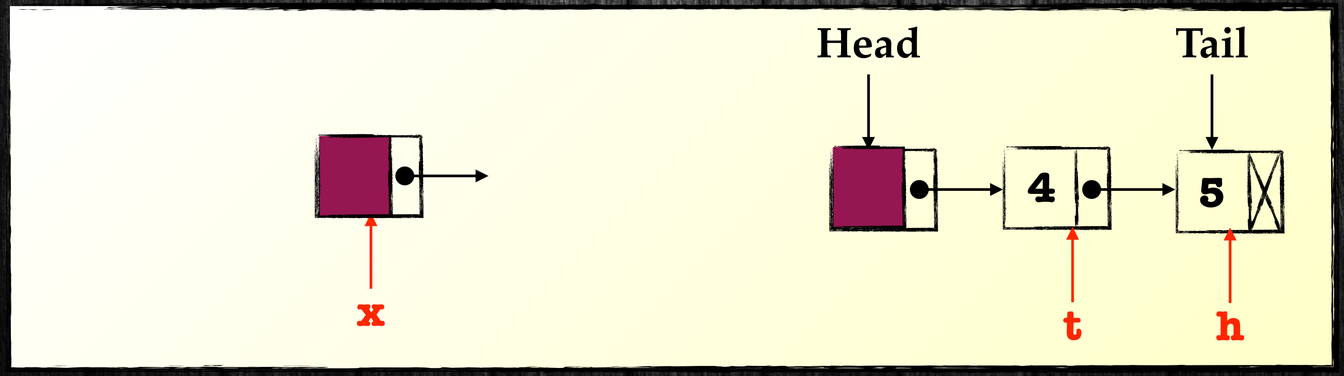
The Michael-Scott Queue Algorithm

The ABA Problem

```
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
```

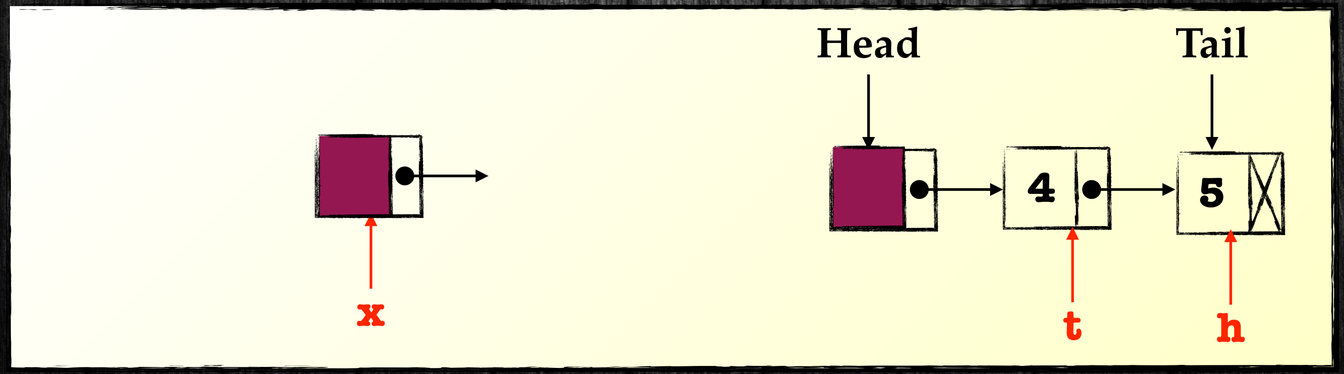


The Michael-Scott Queue Algorithm

The ABA Problem

```
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)
```

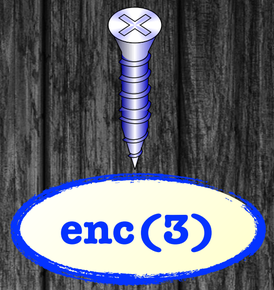
```
deq(k):
Node h,t,x
Integer v
 1  while (true)
 2    h = Head
 3    t = Tail
 4    x = h.next
 5    if (h == Head)
 6      if (h == t)
 7        if (x == NULL)
 8          return *
 9          exit
10        CAS (Tail,t,x)
11      else
12        v = x.key
13        if (CAS (Head,h,x))
14         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



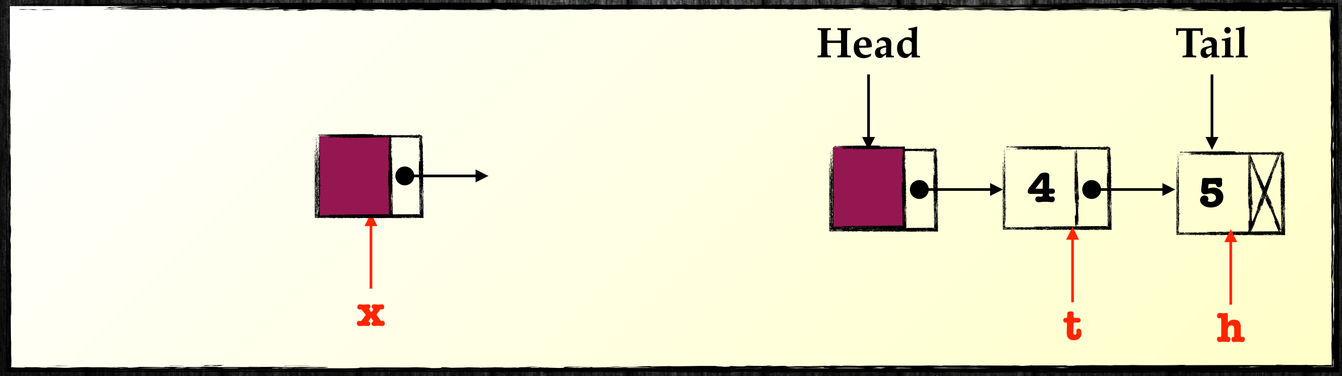
The Michael-Scott Queue Algorithm

The ABA Problem

```
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)
```



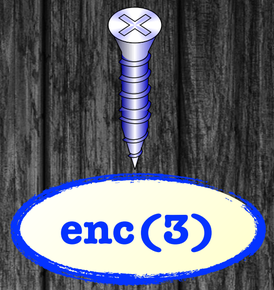

```
deq(k):
Node h,t,x
Integer v
1  while (true)
2    h = Head
3    t = Tail
4    x = h.next
5    if (h == Head)
6      if (h == t)
7        if (x == NULL)
8          return *
9          exit
10       CAS (Tail,t,x)
11     else
12       v = x.key
13       if (CAS (Head,h,x))
14         return v
15  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



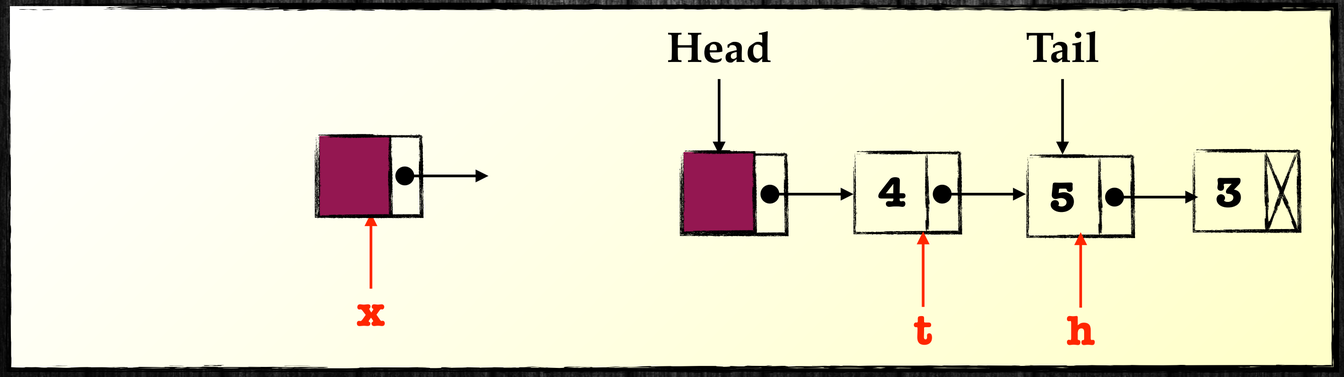
The Michael-Scott Queue Algorithm

The ABA Problem

```
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)
```



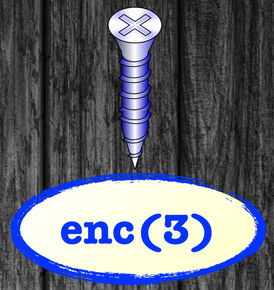

```
deq(k):
Node h,t,x
Integer v
1  while (true)
2    h = Head
3    t = Tail
4    x = h.next
5    if (h == Head)
6      if (h == t)
7        if (x == NULL)
8          return *
9          exit
10       CAS (Tail,t,x)
11     else
12       v = x.key
13       if (CAS (Head,h,x))
14         return v
15  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



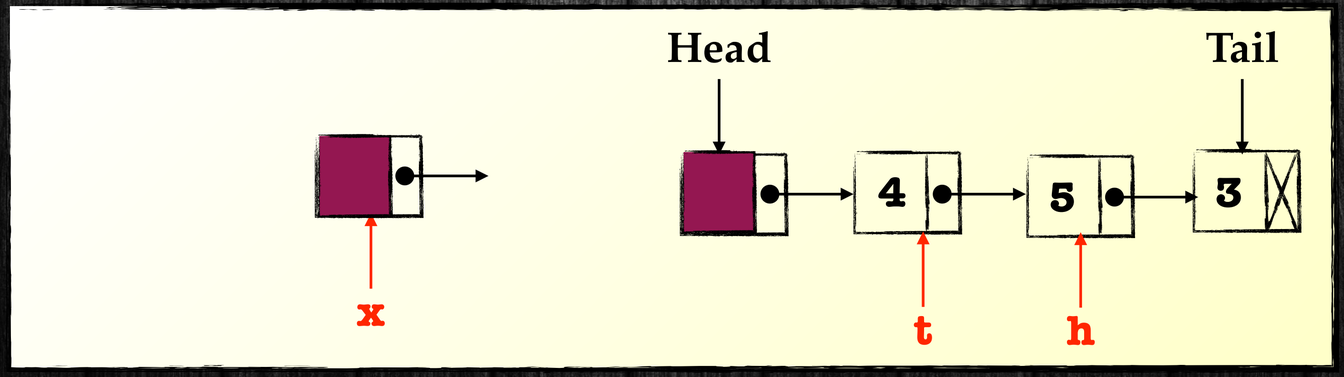
The Michael-Scott Queue Algorithm

The ABA Problem

```
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
```

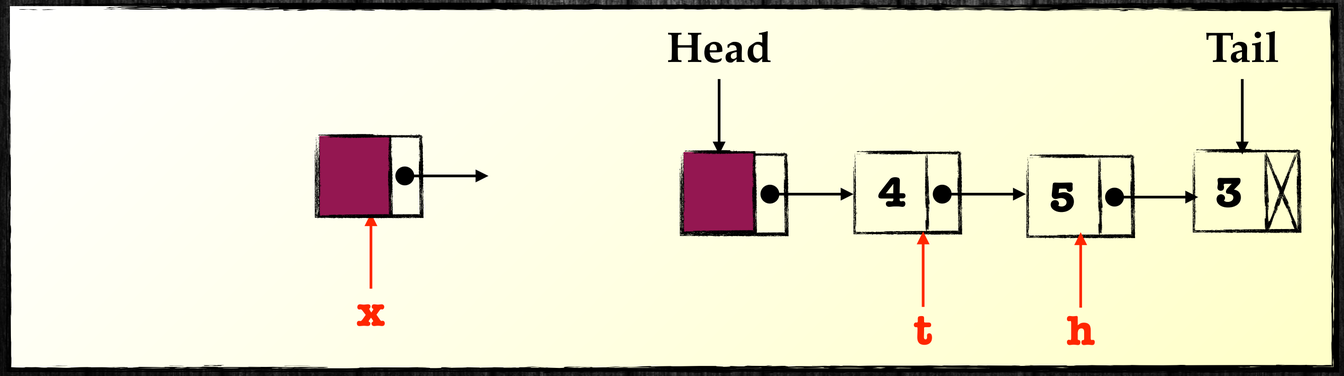


The Michael-Scott Queue Algorithm

The ABA Problem

```
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
```



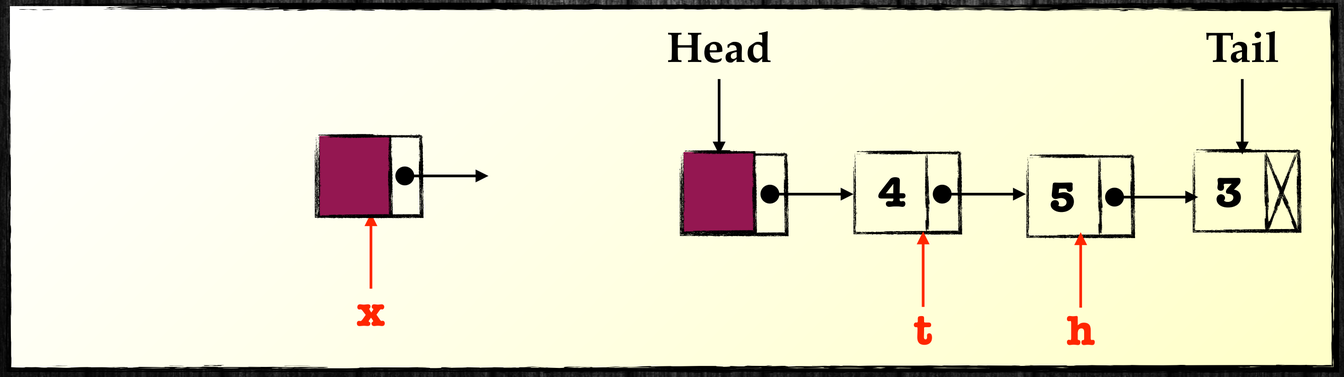
The Michael-Scott Queue Algorithm

The ABA Problem

```
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
```



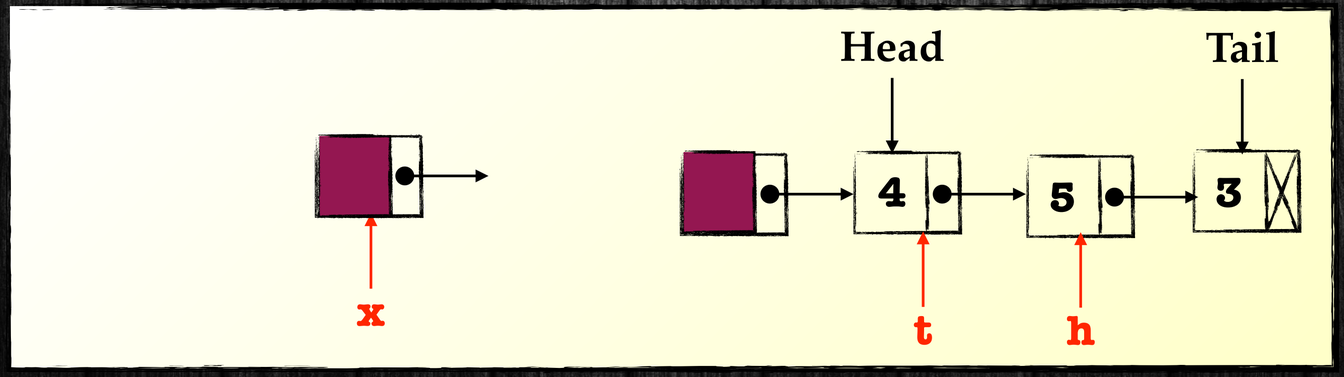
The Michael-Scott Queue Algorithm

The ABA Problem

```
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

The ABA Problem

```
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

The ABA Problem

```
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

The ABA Problem

```
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
```



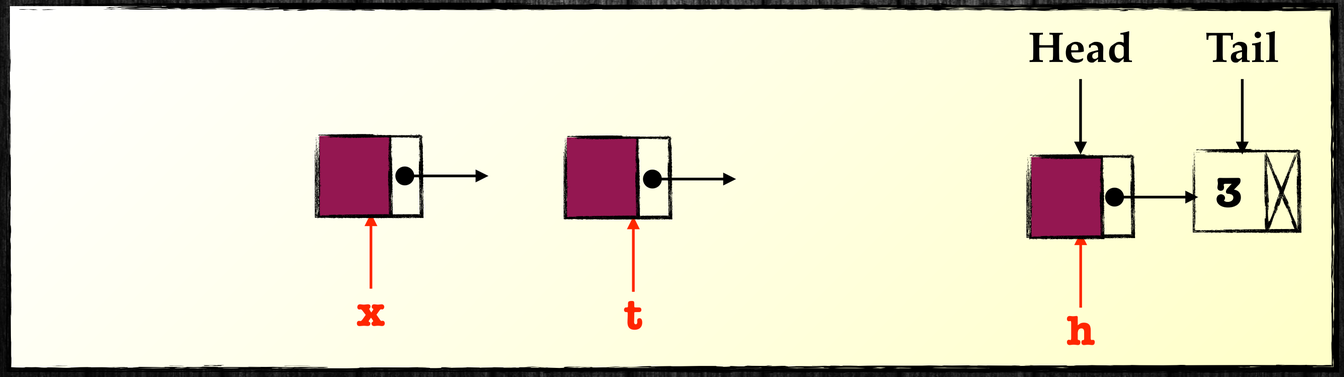
The Michael-Scott Queue Algorithm

The ABA Problem

```
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

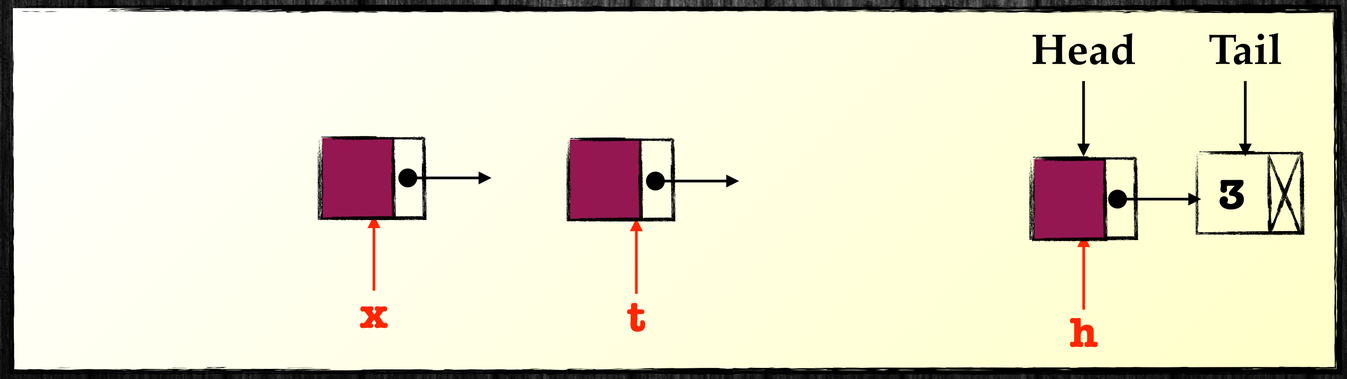
The ABA Problem

```
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
```

wake up



The Michael-Scott Queue Algorithm

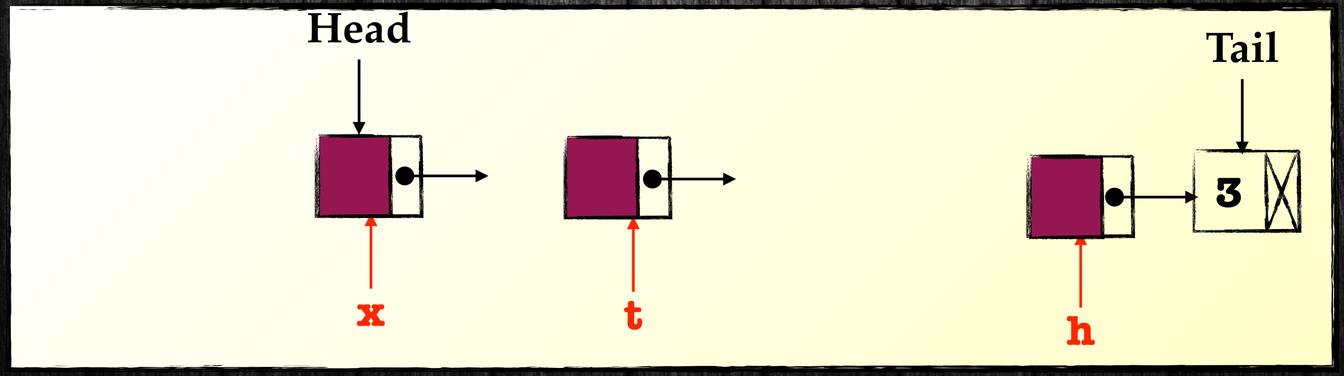
The ABA Problem

```
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
```

wake up



The Michael-Scott Queue Algorithm

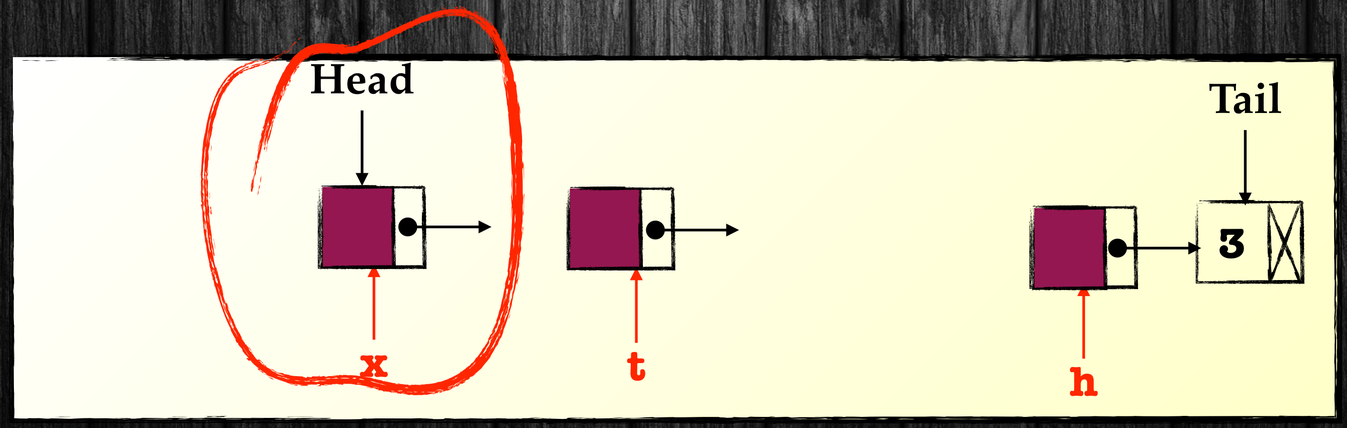
The ABA Problem

```
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
```

wake up



The Michael-Scott Queue Algorithm

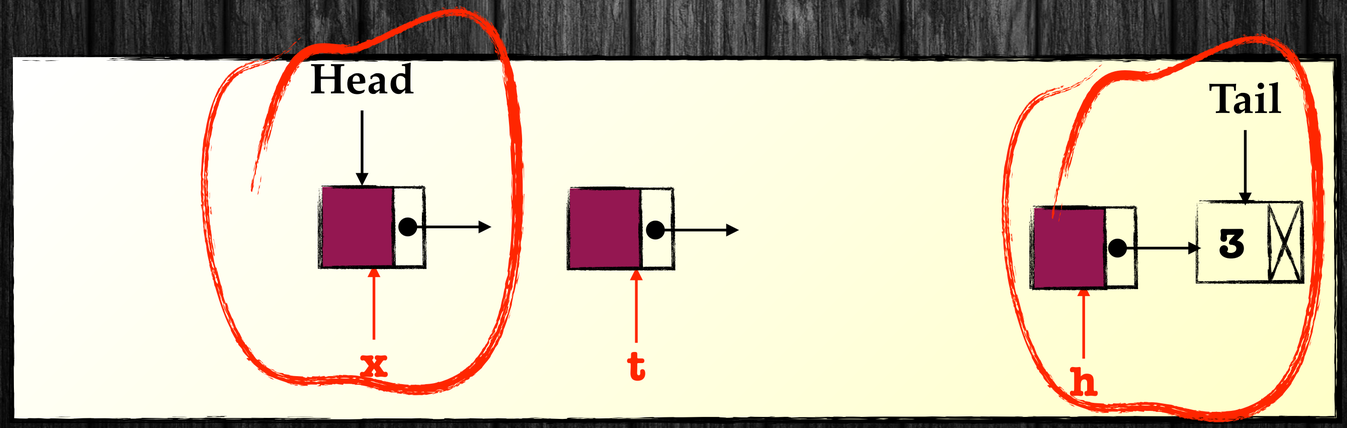
The ABA Problem

```
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
```

wake up



Concurrent Algorithms and Data Structures

Outline

Concurrent Algorithms and Data Structures

Outline

- Overview

Concurrent Algorithms and Data Structures

Outline

- Overview
- Abstract Data Types

Concurrent Algorithms and Data Structures

Outline

- Overview
- Abstract Data Types
- Sequential Programs

Concurrent Algorithms and Data Structures

Outline

- Overview
- Abstract Data Types
- Sequential Programs
- Concurrent Programs

Concurrent Algorithms and Data Structures

Outline

- Overview
- Abstract Data Types
- Sequential Programs
- Concurrent Programs
- Coarse-Grained Locking

Concurrent Algorithms and Data Structures

Outline

- Overview
- Abstract Data Types
- Sequential Programs
- Concurrent Programs
- Coarse-Grained Locking
- Fine-Grained Locking

Concurrent Algorithms and Data Structures

Outline

- Overview
- Abstract Data Types
- Sequential Programs
- Concurrent Programs
- Coarse-Grained Locking
- Fine-Grained Locking
- Lazy Lists

Concurrent Algorithms and Data Structures

Outline

- Overview
- Abstract Data Types
- Sequential Programs
- Concurrent Programs
- Coarse-Grained Locking
- Fine-Grained Locking
- Lazy Lists
- Optimistic Lists

Concurrent Algorithms and Data Structures

Outline

- Overview
- Abstract Data Types
- Sequential Programs
- Concurrent Programs
- Coarse-Grained Locking
- Fine-Grained Locking
- Lazy Lists
- Optimistic Lists

Set Algorithms

Concurrent Algorithms and Data Structures

Outline

- Overview
- Abstract Data Types
- Sequential Programs
- Concurrent Programs
- Coarse-Grained Locking
- Fine-Grained Locking
- Lazy Lists
- Optimistic Lists
- Treiber Stack Algorithm

Set Algorithms

Concurrent Algorithms and Data Structures

Outline

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

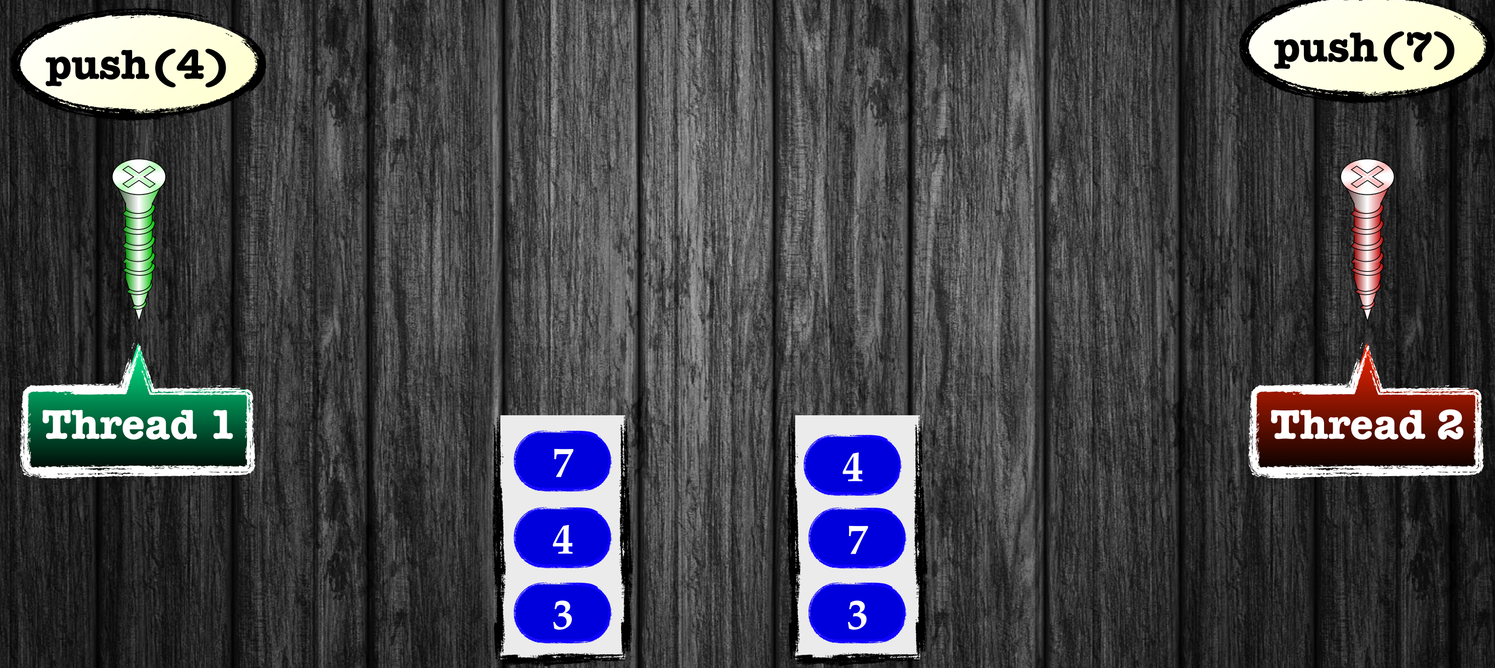
Set Algorithms

Overview

- Algorithms and Data Structures
- Concurrency
- Efficiency & Correctness
- Concepts
  - Specifications
  - Safety Properties
  - Liveness Properties

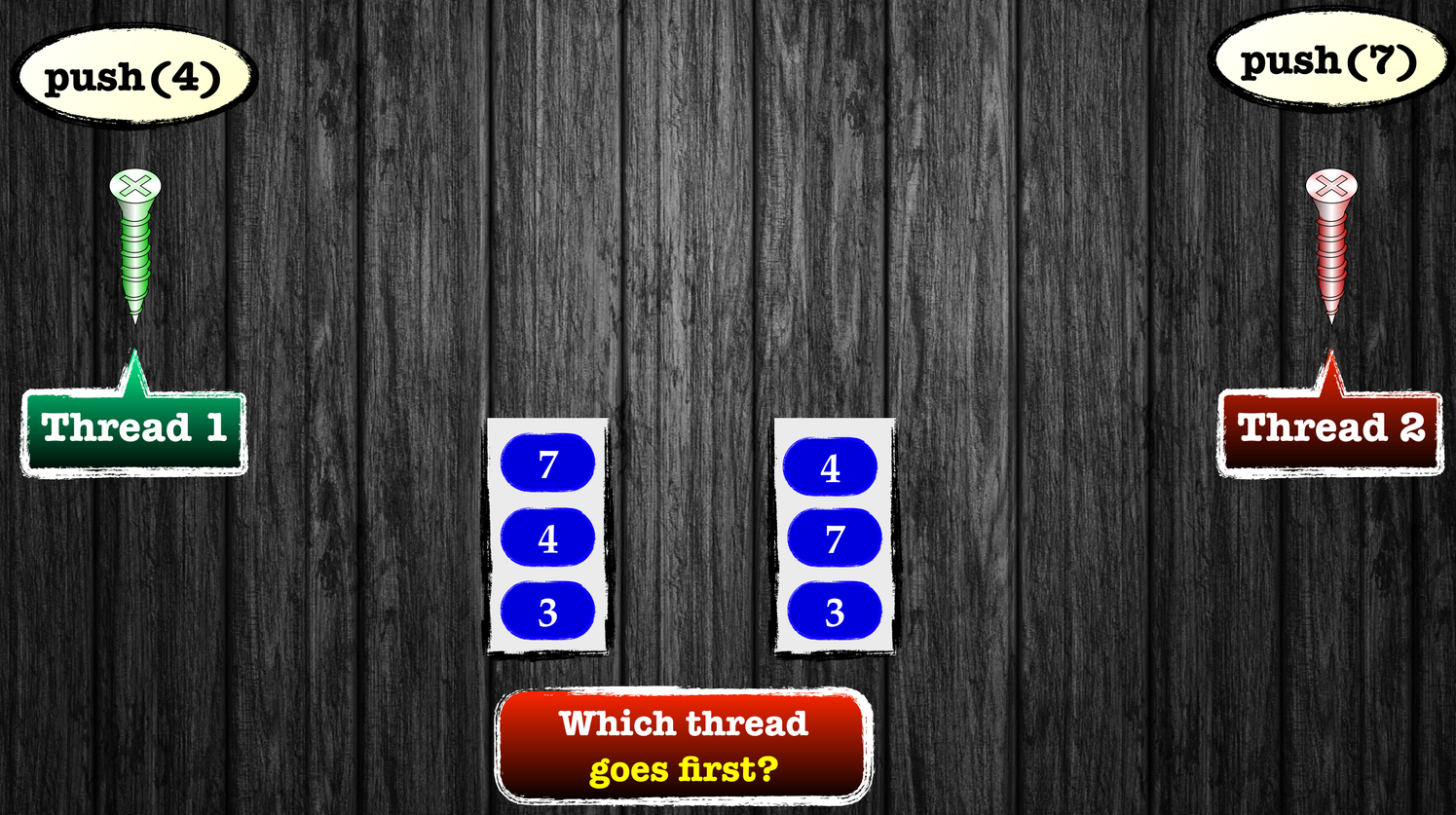
Overview | Goal → concurrent setting

Example | Stack | (Basic) Algorithms & Data Structures



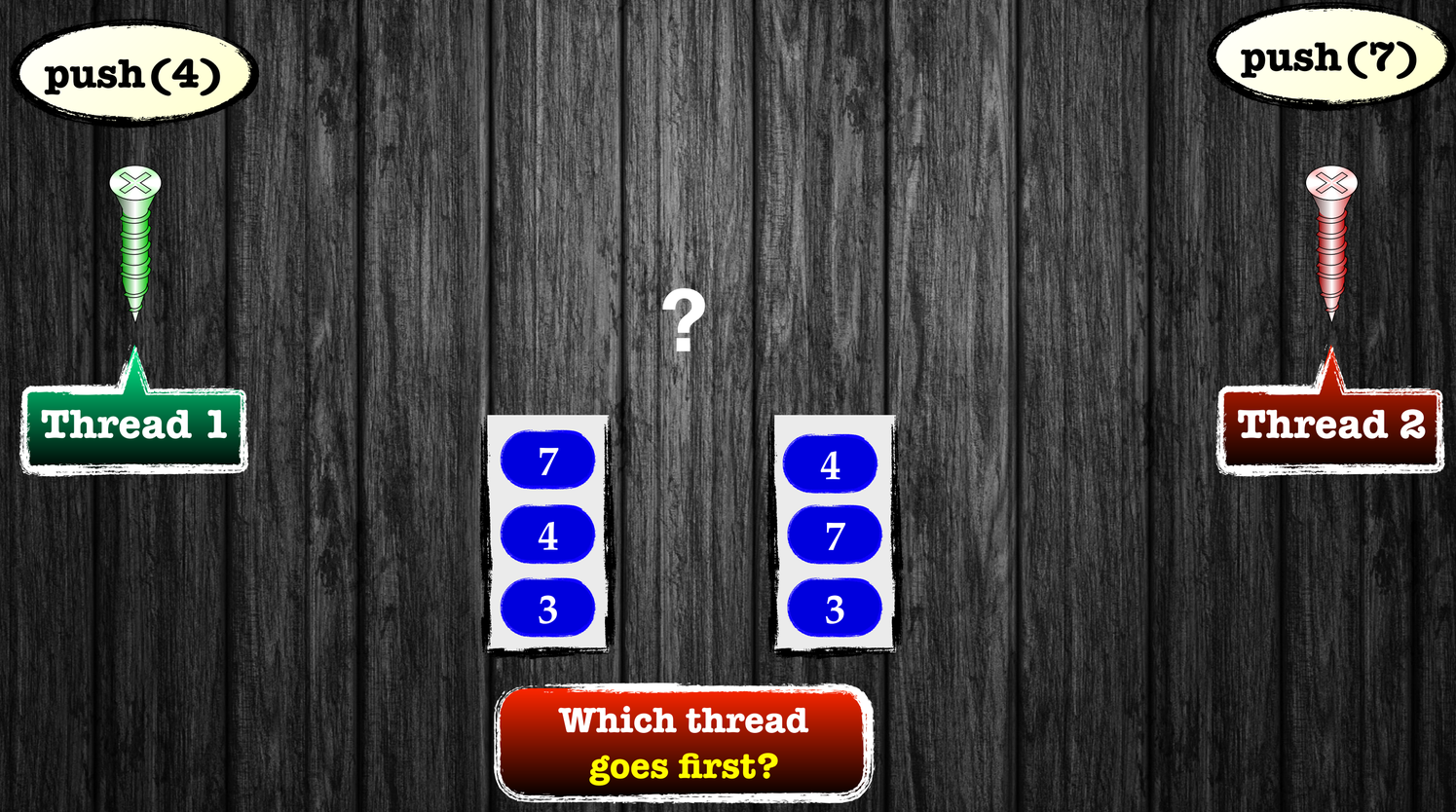
Overview | Goal → concurrent setting

Example | Stack | (Basic) Algorithms & Data Structures



Overview | Goal → concurrent setting

Example | Stack | (Basic) Algorithms & Data Structures










thank you




thank you


thank you


thank you


thank you