Concurrent Algorithms and Data Structures

Concurrent Algorithms and Data Structures

- Slides uploaded to “Studium”
- Please, download the slides first, and only then run the animations
- Report problems with downloading/ watching the slides
- Low-Resolution, No-Animation pdfs are available (around 50 MBs)

Concurrent Algorithms and Data Structures

- Slides uploaded to “Studium”
- Please, download the slides first, and only then run the animations
- Report problems with downloading/ watching the slides
- Low-Resolution, No-Animation pdfs are available (around 50 MBs)

- The first theory assignment is available.
- The first code assignment is available.

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

Overview

Overview



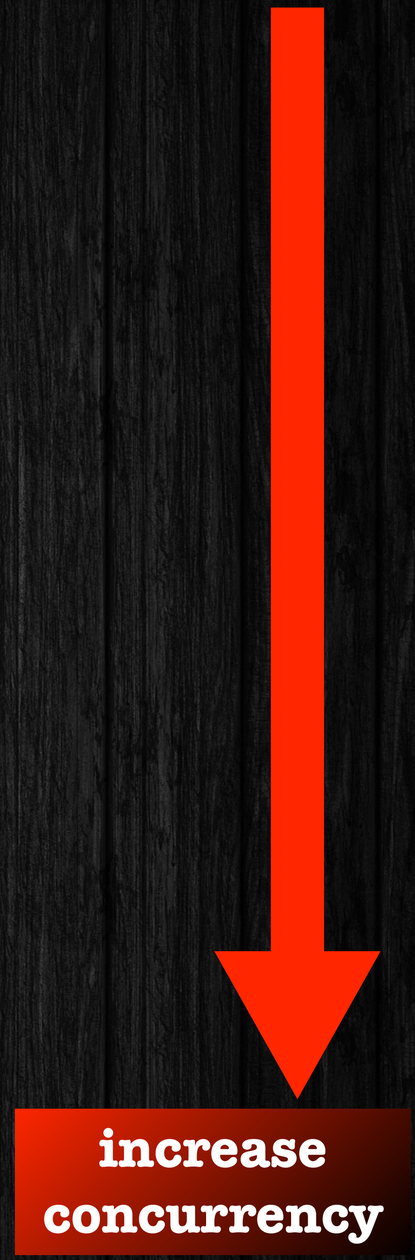
Overview

- Sequential Programs



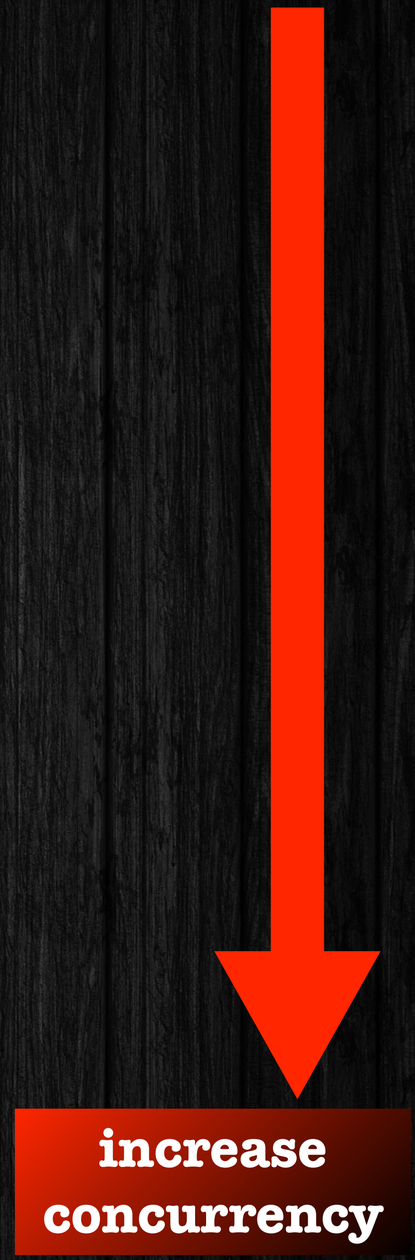
Overview

- Sequential Programs
- Coarse-Grained Synchronization



Overview

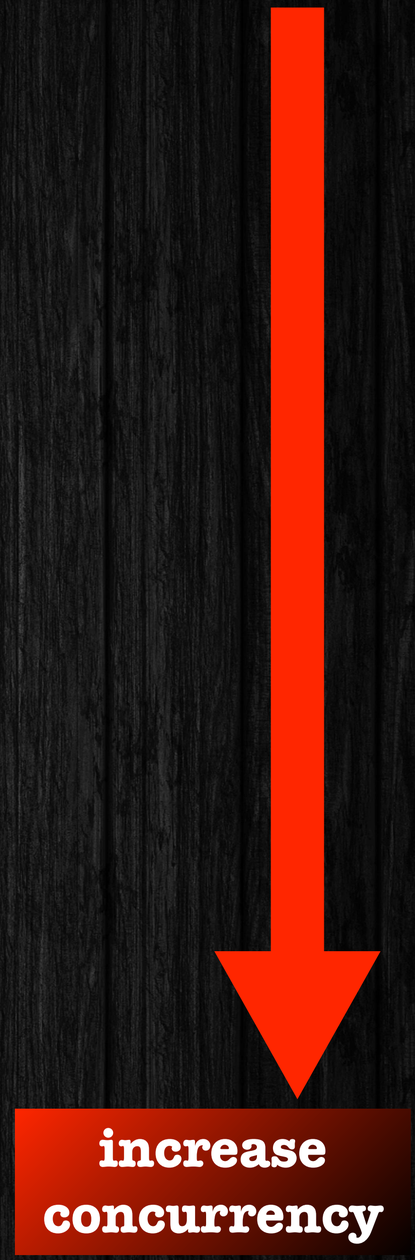
- Sequential Programs
- Coarse-Grained Synchronization
- Single global lock



Overview

- Sequential Programs
- Coarse-Grained Synchronization
- Fine-Grained Synchronization

- Single global lock



Overview

- Sequential Programs
- Coarse-Grained Synchronization
  - Single global lock
- Fine-Grained Synchronization
  - lock per node



Overview

- Sequential Programs
- Coarse-Grained Synchronization
  - Single global lock
- Fine-Grained Synchronization
  - lock per node
  - lock each visited node



Overview

- Sequential Programs
- Coarse-Grained Synchronization
  - Single global lock
- Fine-Grained Synchronization
  - lock per node
  - lock each visited node
- Optimistic Synchronization



Overview

- Sequential Programs
- Coarse-Grained Synchronization
  - Single global lock
- Fine-Grained Synchronization
  - lock per node
  - lock each visited node
- Optimistic Synchronization
  - lock when correct node found



Overview

- Sequential Programs
- Coarse-Grained Synchronization
  - Single global lock
- Fine-Grained Synchronization
  - lock per node
  - lock each visited node
- Optimistic Synchronization
  - lock when correct node found
  - restart search on failure



Overview

- Sequential Programs
- Coarse-Grained Synchronization
  - Single global lock
- Fine-Grained Synchronization
  - lock per node
  - lock each visited node
- Optimistic Synchronization
  - lock when correct node found
  - restart search on failure
- Lazy Synchronization



Overview

- Sequential Programs
- Coarse-Grained Synchronization
  - Single global lock
- Fine-Grained Synchronization
  - lock per node
  - lock each visited node
- Optimistic Synchronization
  - lock when correct node found
  - restart search on failure
- Lazy Synchronization
  - make ctn wait-free



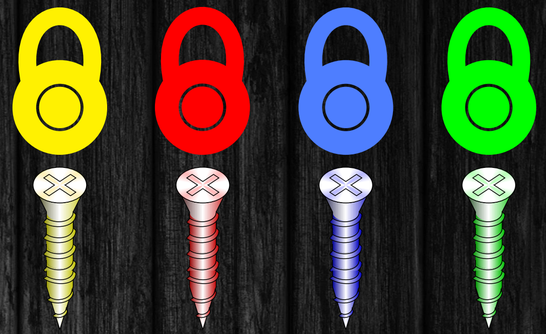
Outline

- 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

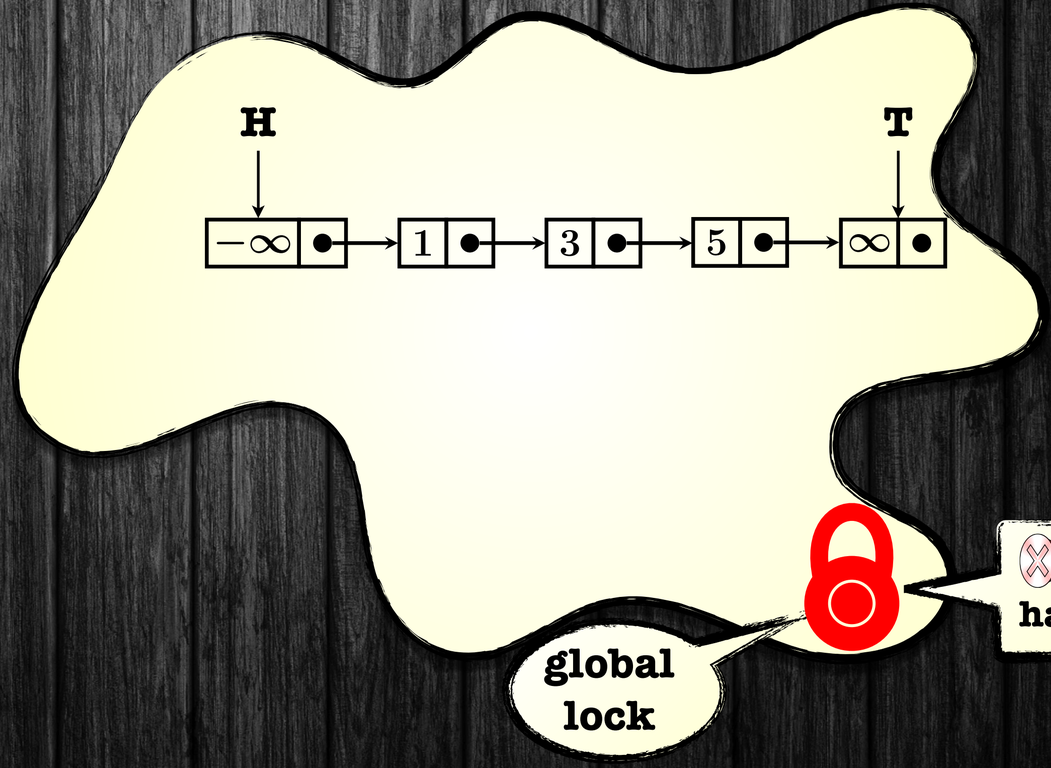
Outline

- 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

Sets Coarse-Grained Synchronization



At most one thread has lock

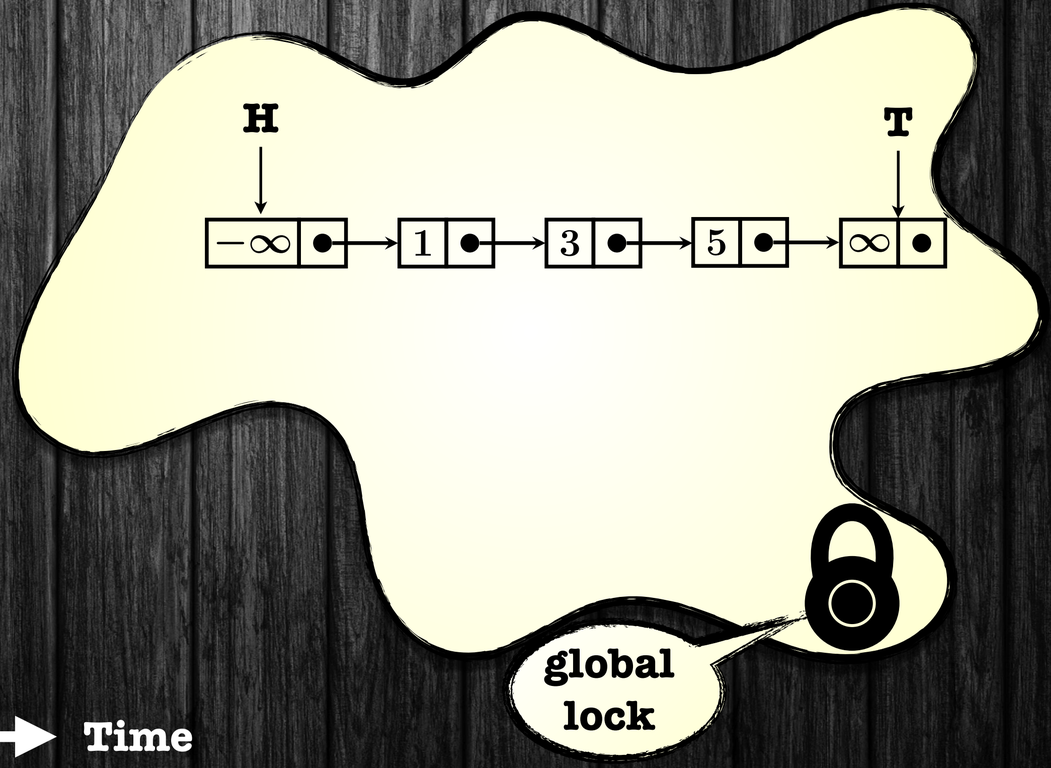


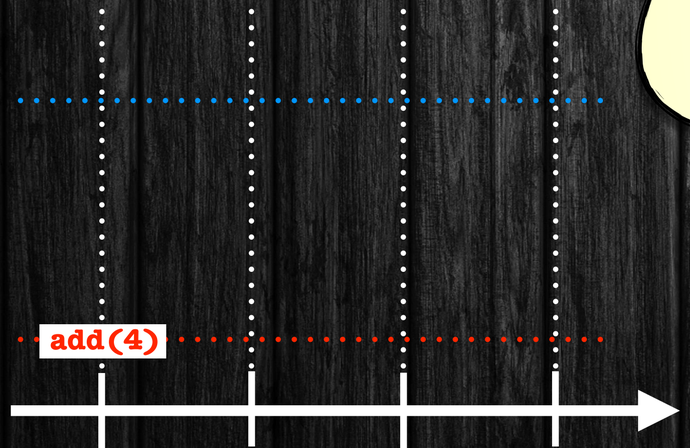
Sets Coarse-Grained Synchronization

```
add(k):
•••
• lock l; LP
• •••
• unlock l
```




```
add(k):
•••
• lock l; LP
• •••
• unlock l
```





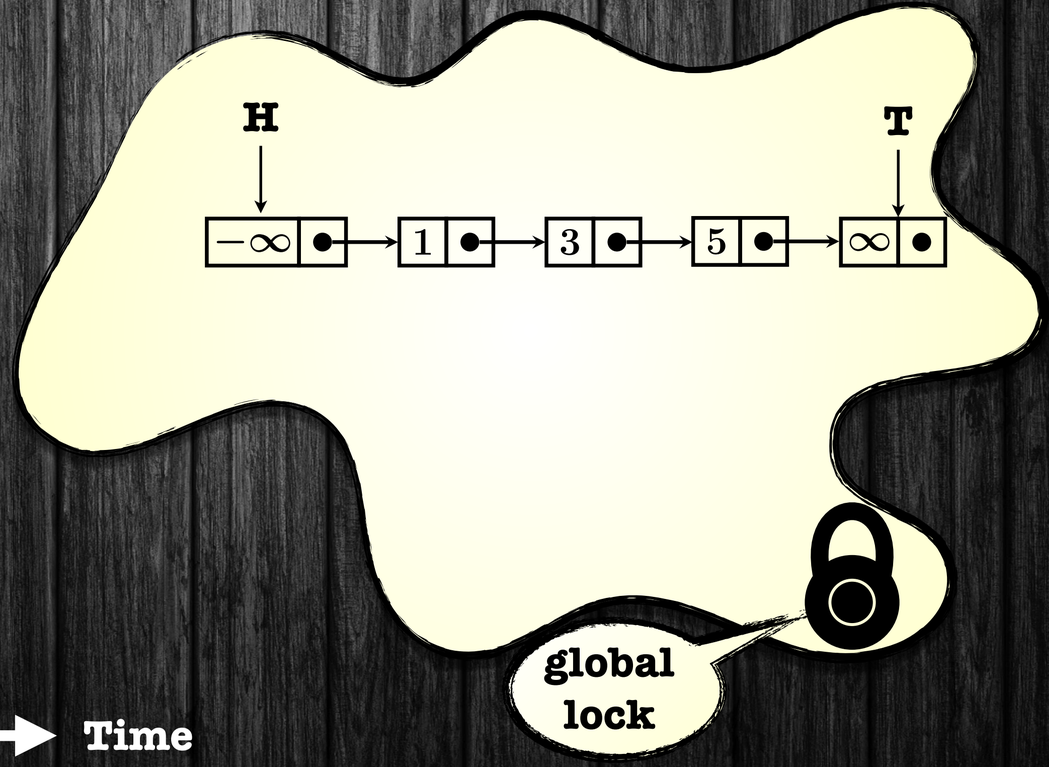
Sets Coarse-Grained Synchronization

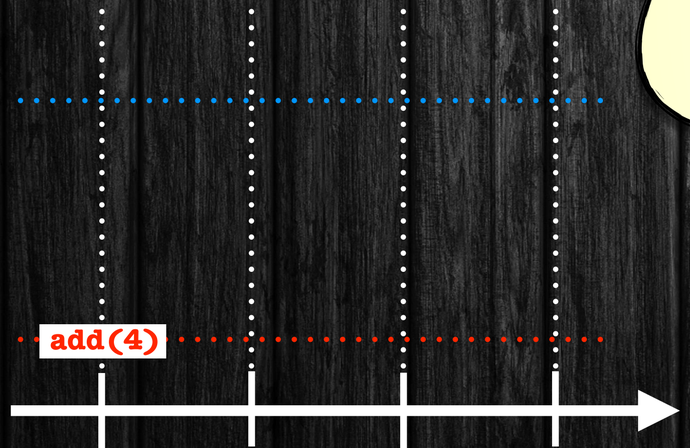
```
add(k):
•••
• lock l; LP
• •••
• unlock l
```

add(4)

add(2)

```
add(k):
•••
• lock l; LP
• •••
• unlock l
```





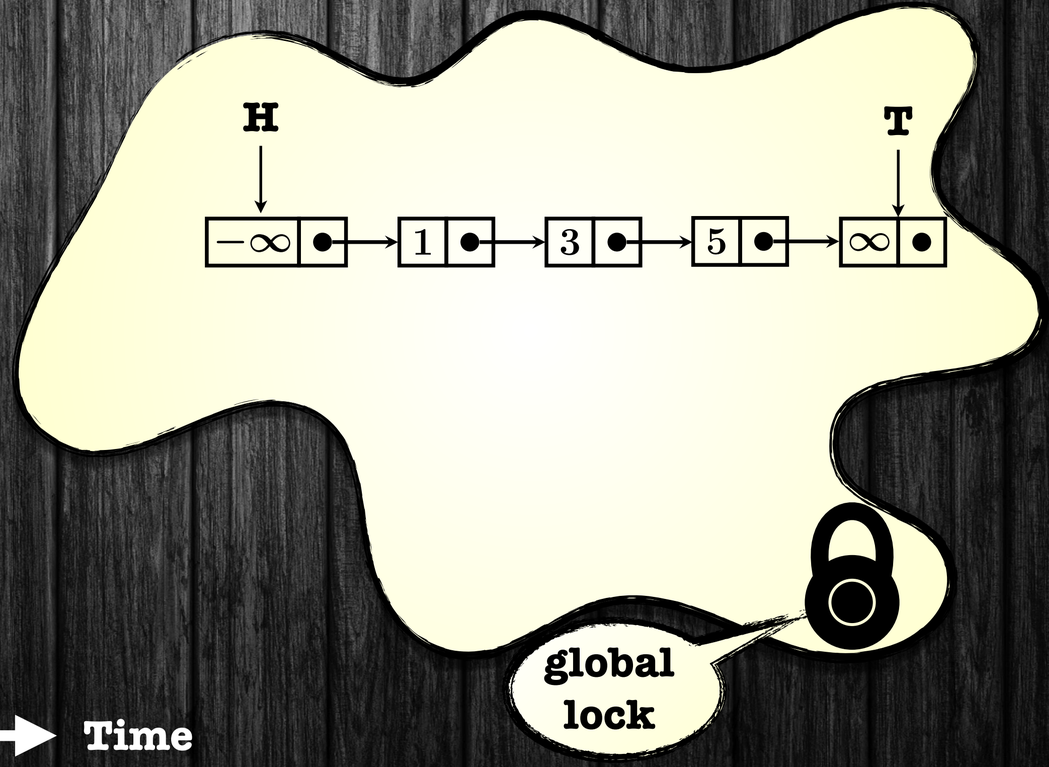
Sets Coarse-Grained Synchronization

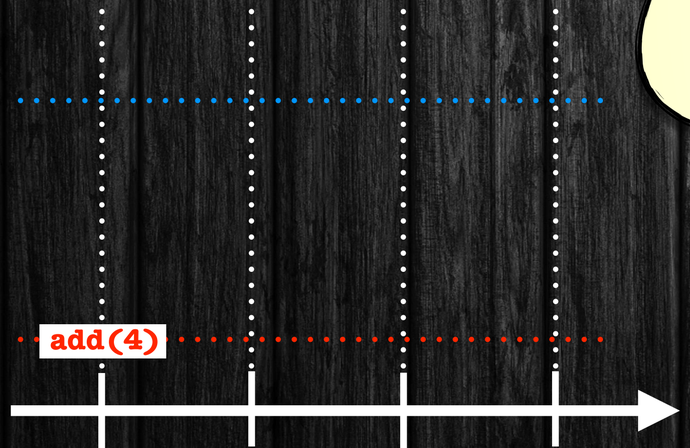
```
add(k):
•••
• lock l; LP
• •••
• unlock l
```

add(4)

add(2)

```
add(k):
•••
• lock l; LP
• •••
• unlock l
```





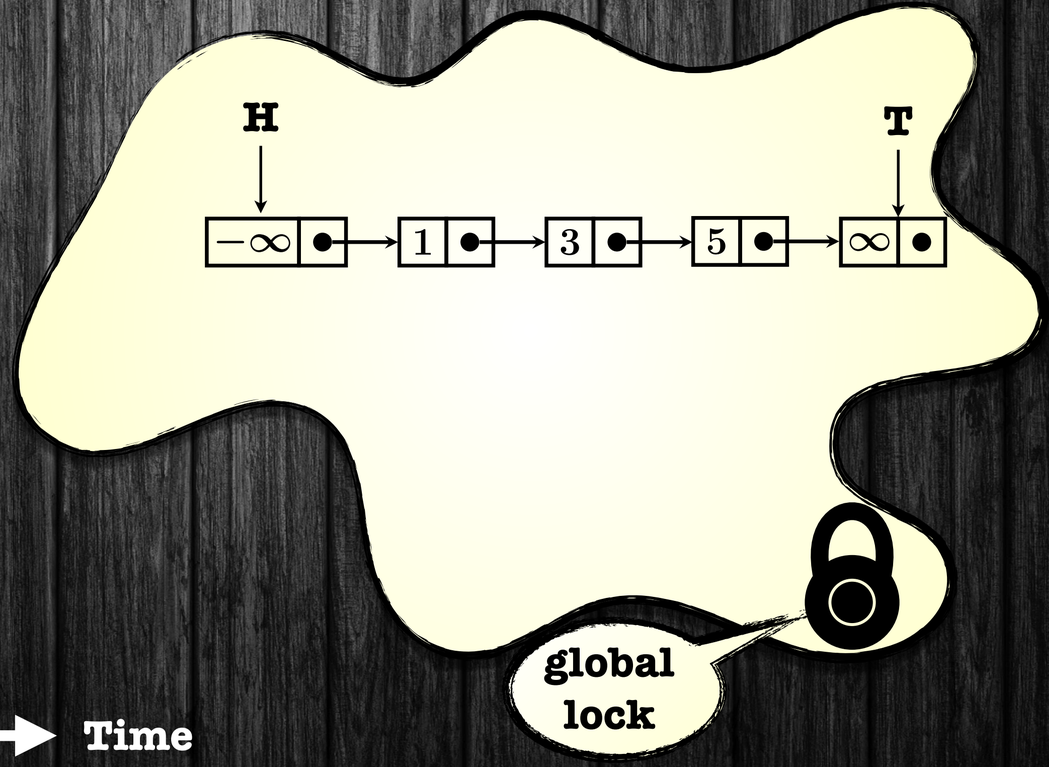
Sets Coarse-Grained Synchronization

```
add(k):
•••
• lock l; LP
• •••
• unlock l
```

add(4)

add(2)

```
add(k):
•••
• lock l; LP
• •••
• unlock l
```





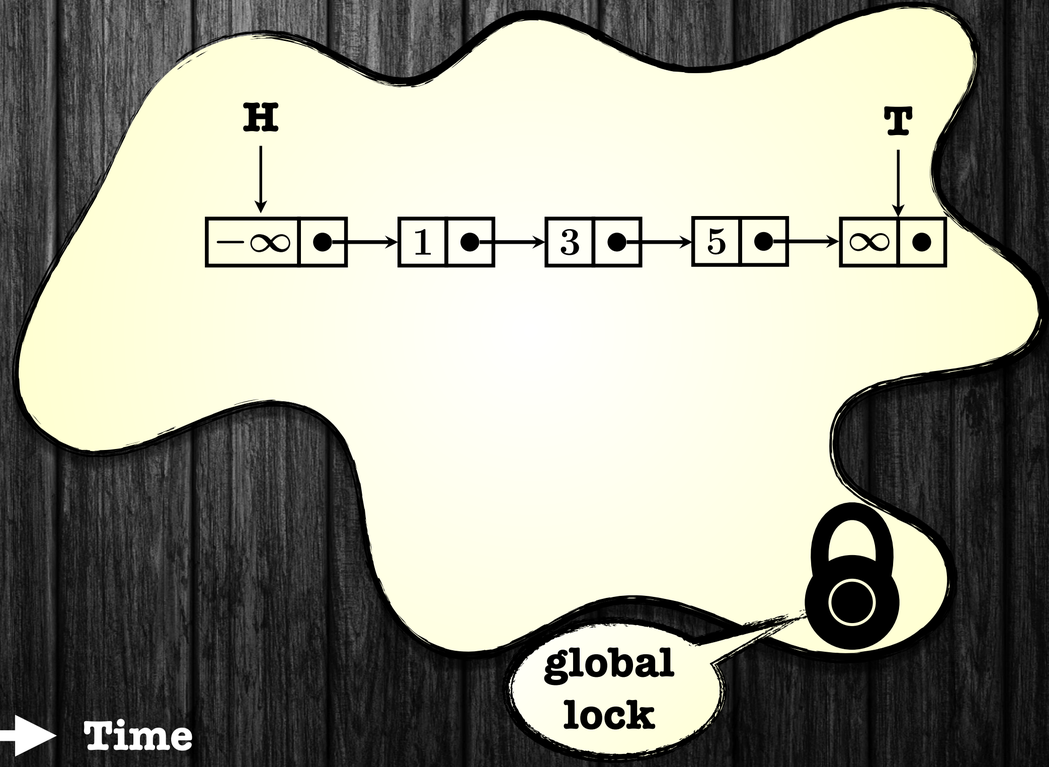
Sets Coarse-Grained Synchronization

```
add(k):
•••
• lock l; LP
• •••
• unlock l
```

add(4)

add(2)

```
add(k):
•••
• lock l; LP
• •••
• unlock l
```





Sets Coarse-Grained Synchronization

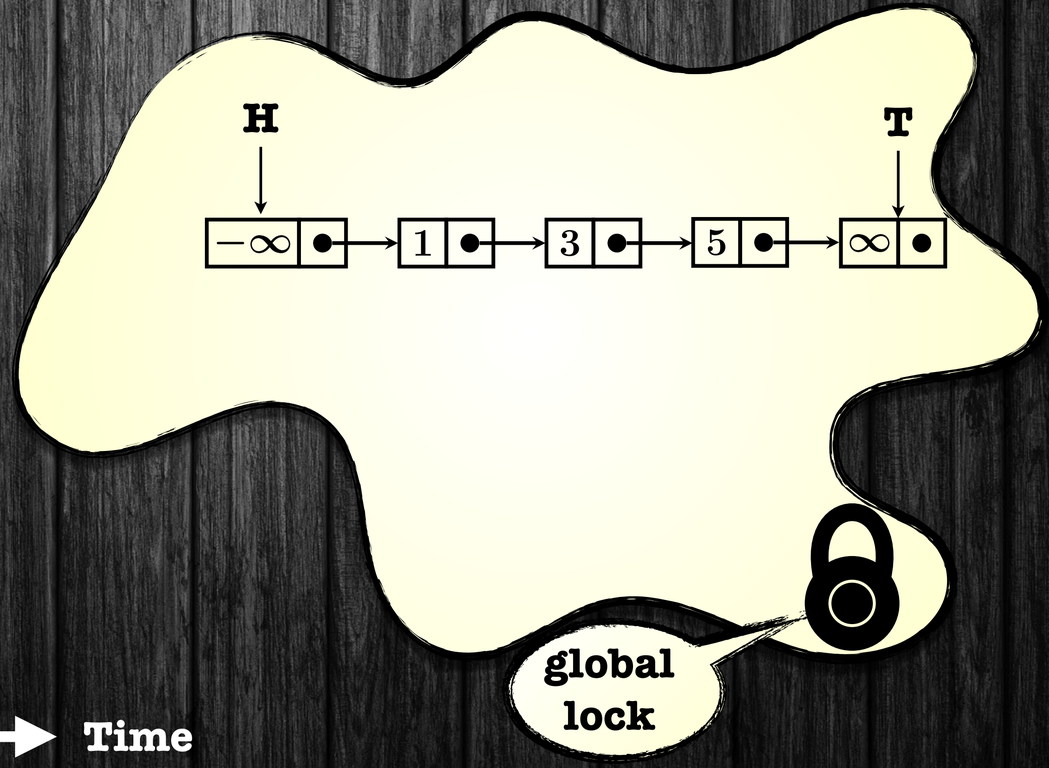
```
add(k):
•••
• lock l; LP
• •••
• unlock l
```

add(4)

add(2)

```
add(k):
•••
• lock l; LP
• •••
• unlock l
```





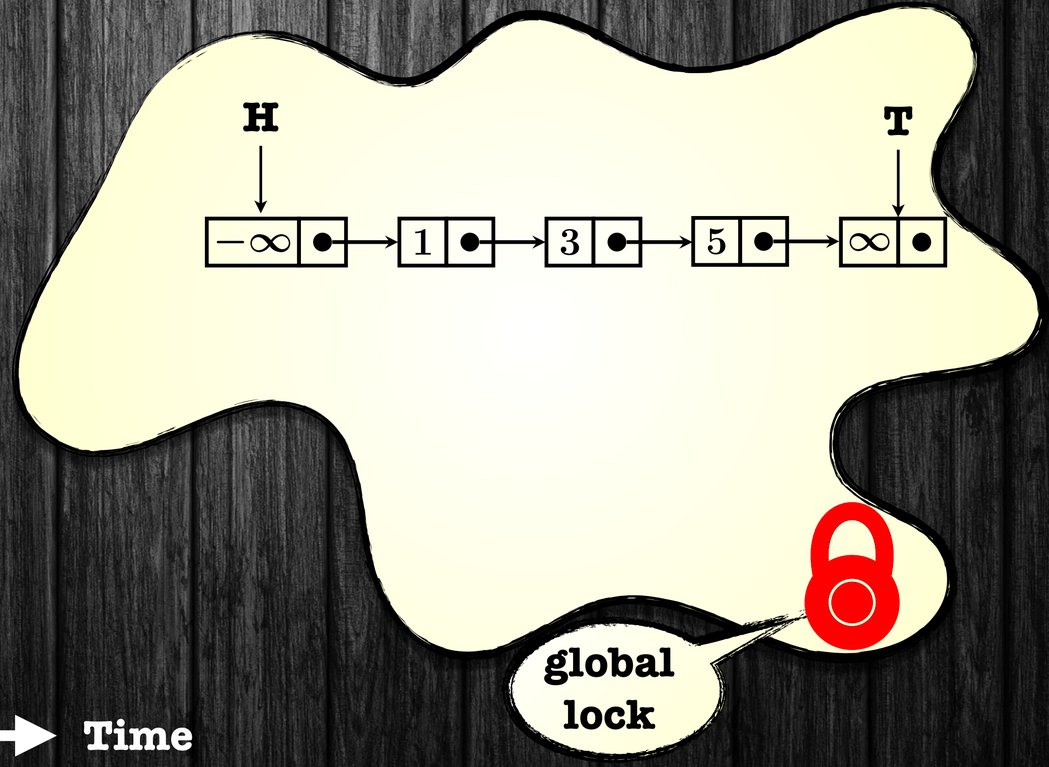
Sets Coarse-Grained Synchronization

```
add(k):
•••
• lock l; LP
• •••
• unlock l
```

add(4)

add(2)

```
add(k):
•••
• lock l; LP
• •••
• unlock l
```





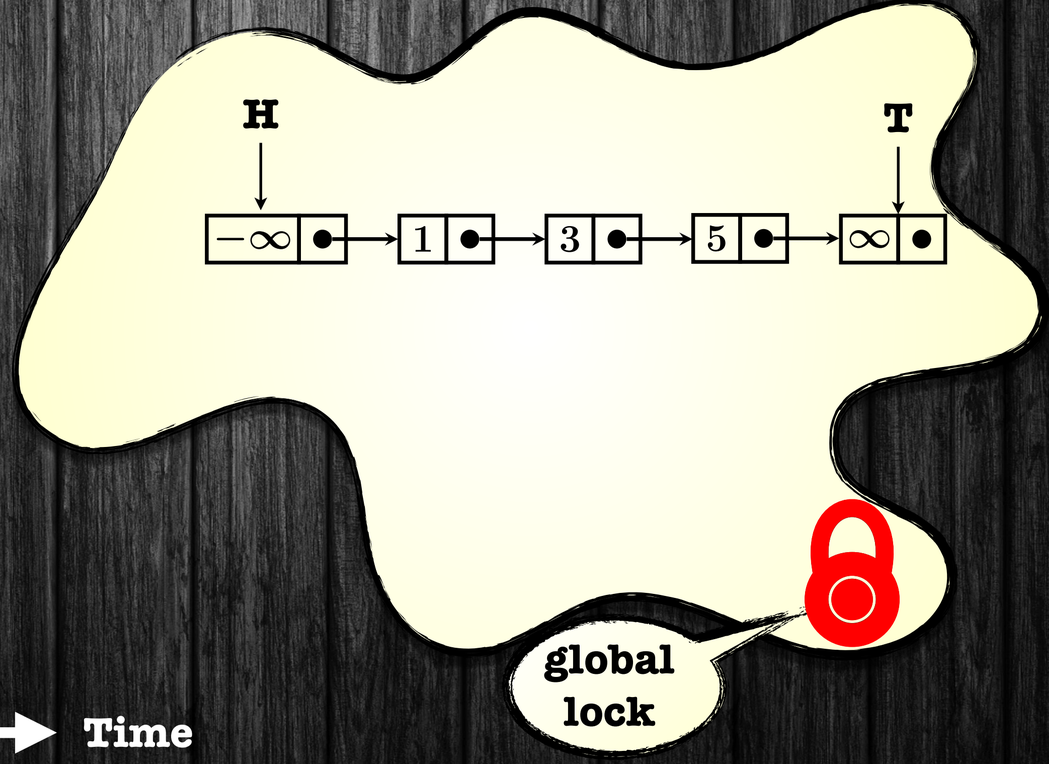
Sets Coarse-Grained Synchronization

```
add(k):
•••
• lock l; LP
• •••
• unlock l
```

add(4)

add(2)

```
add(k):
•••
• lock l; LP
• •••
• unlock l
```





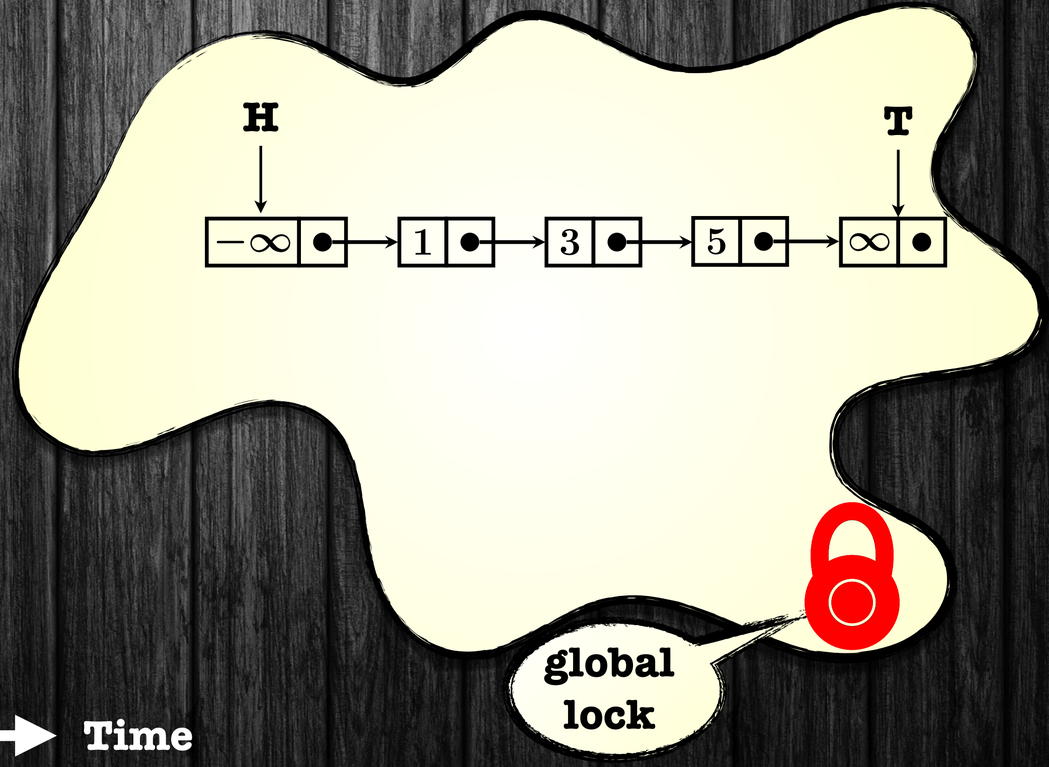
Sets Coarse-Grained Synchronization

```
add(k):
•••
• lock l; LP
• •••
• unlock l
```

add(4)

add(2)

```
add(k):
•••
• lock l; LP
• •••
• unlock l
```

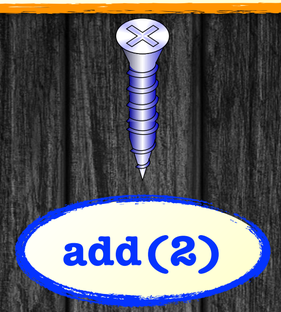




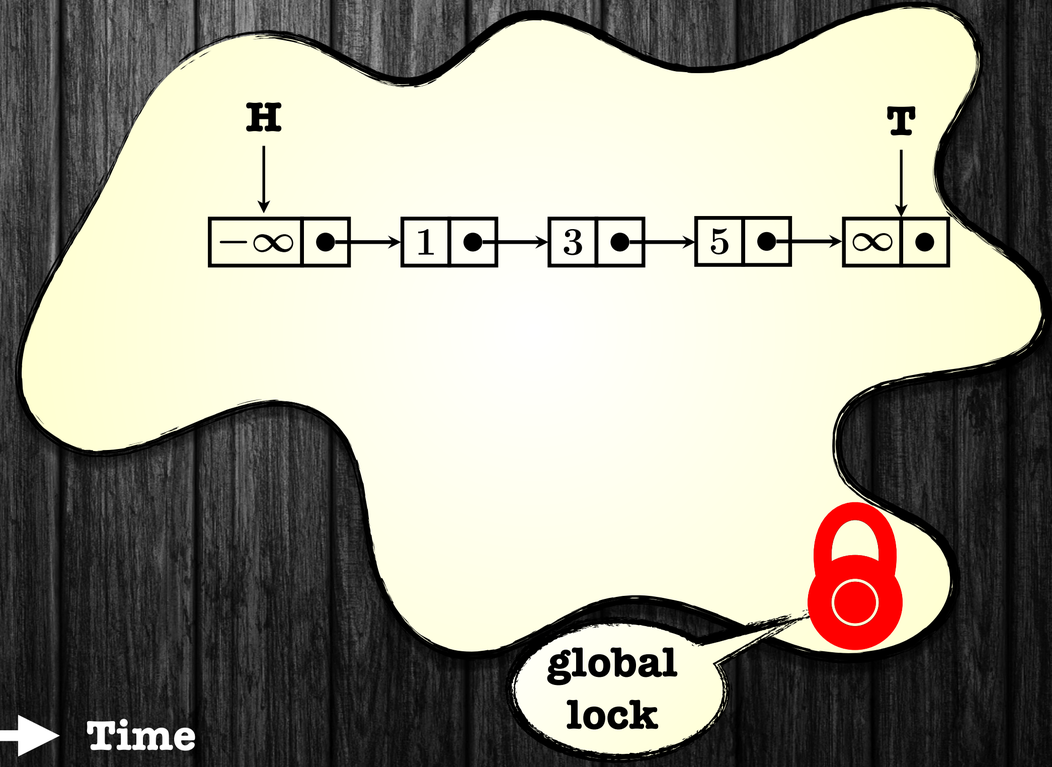
Sets Coarse-Grained Synchronization

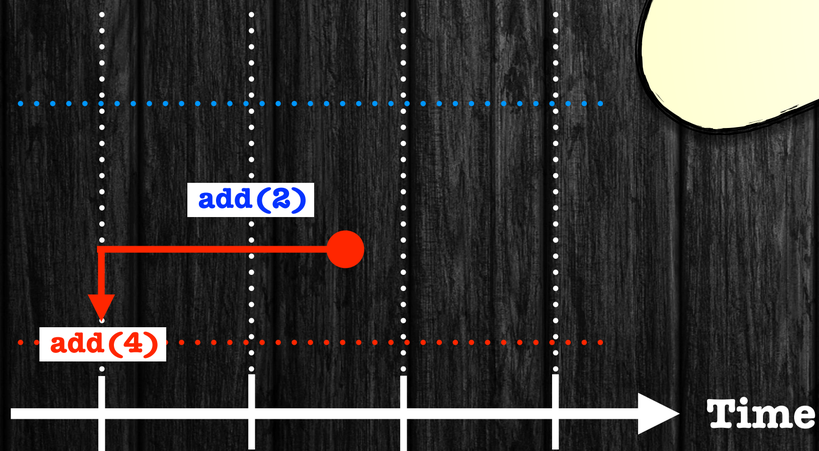
```
add(k):
•••
• lock l;   LP
• •••
• unlock l
```






```
add(k):
•••
• lock l;   LP
• •••
• unlock l
```





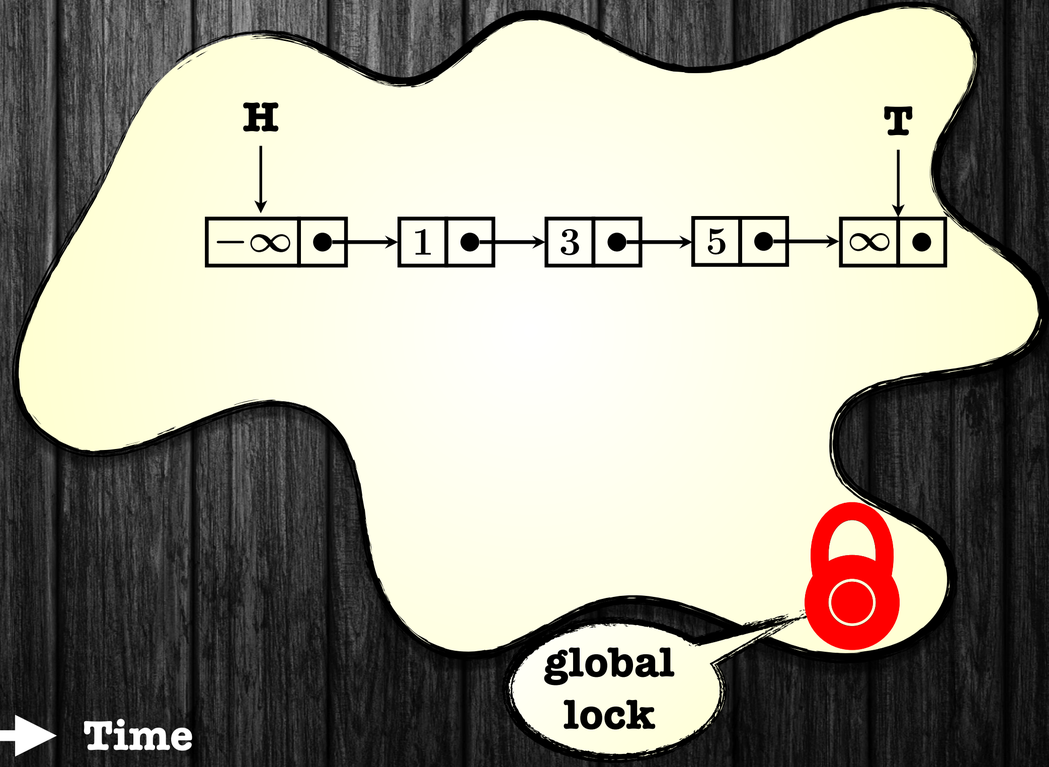
Sets Coarse-Grained Synchronization

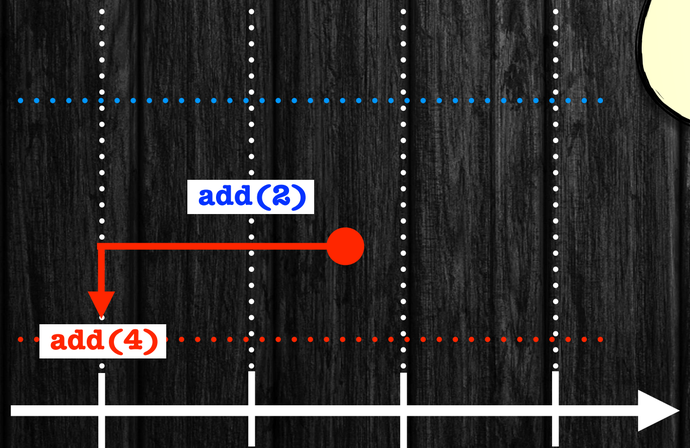
```
add(k):
•••
• lock l; LP
• •••
• unlock l
```

add(4)

add(2)

```
add(k):
•••
• lock l; LP
• •••
• unlock l
```





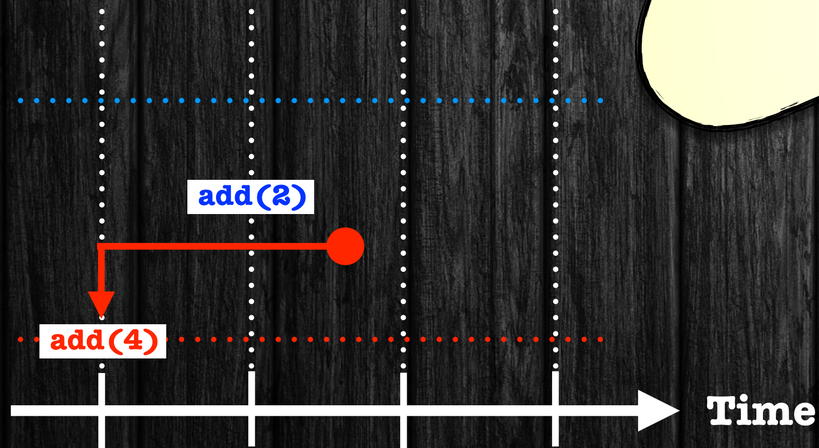
Sets Coarse-Grained Synchronization

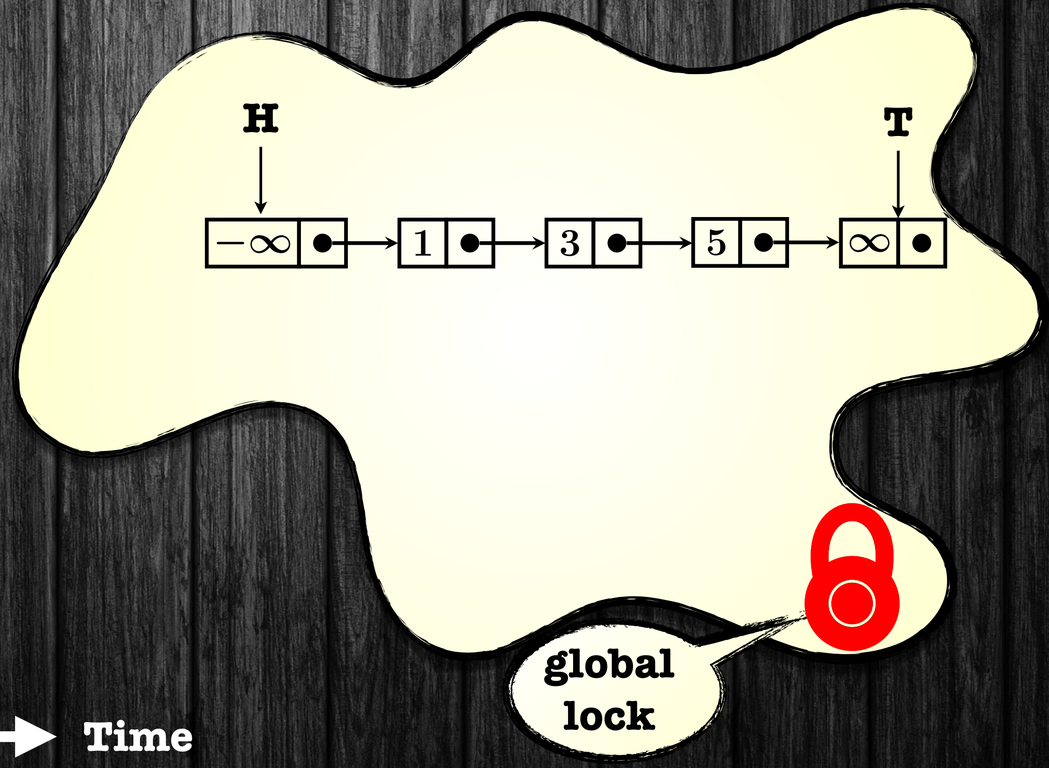
```
add(k):
•••
• lock l; LP
• •••
• unlock l
```

add(4)

add(2)

```
add(k):
•••
• lock l; LP
• •••
• unlock l
```





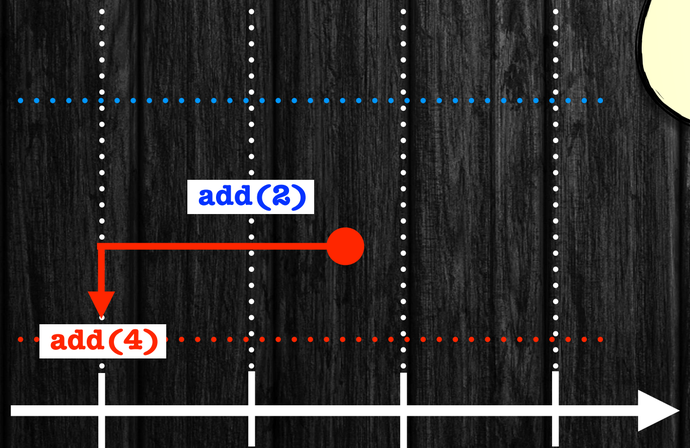
Sets Coarse-Grained Synchronization

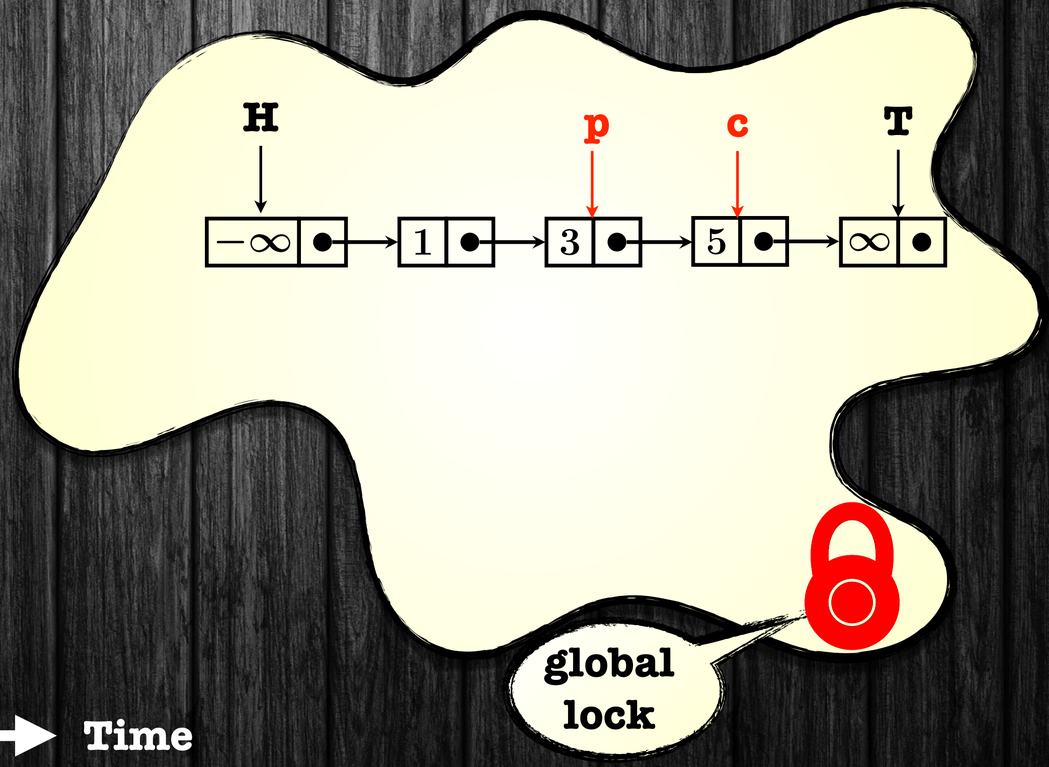
```
add(k):
•••
• lock l; LP
• •••
• unlock l
```

add(4)

add(2)

```
add(k):
•••
• lock l; LP
• •••
• unlock l
```





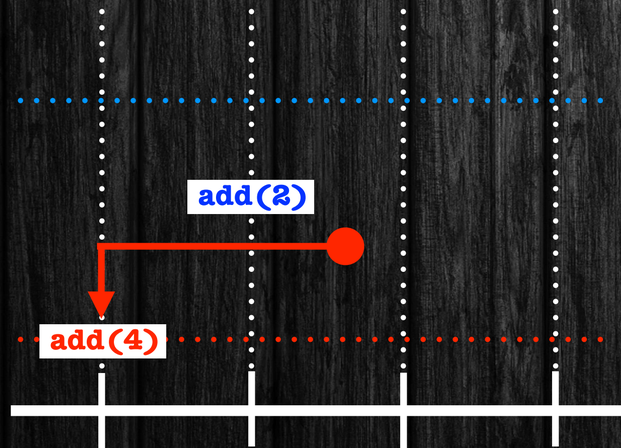
Sets Coarse-Grained Synchronization

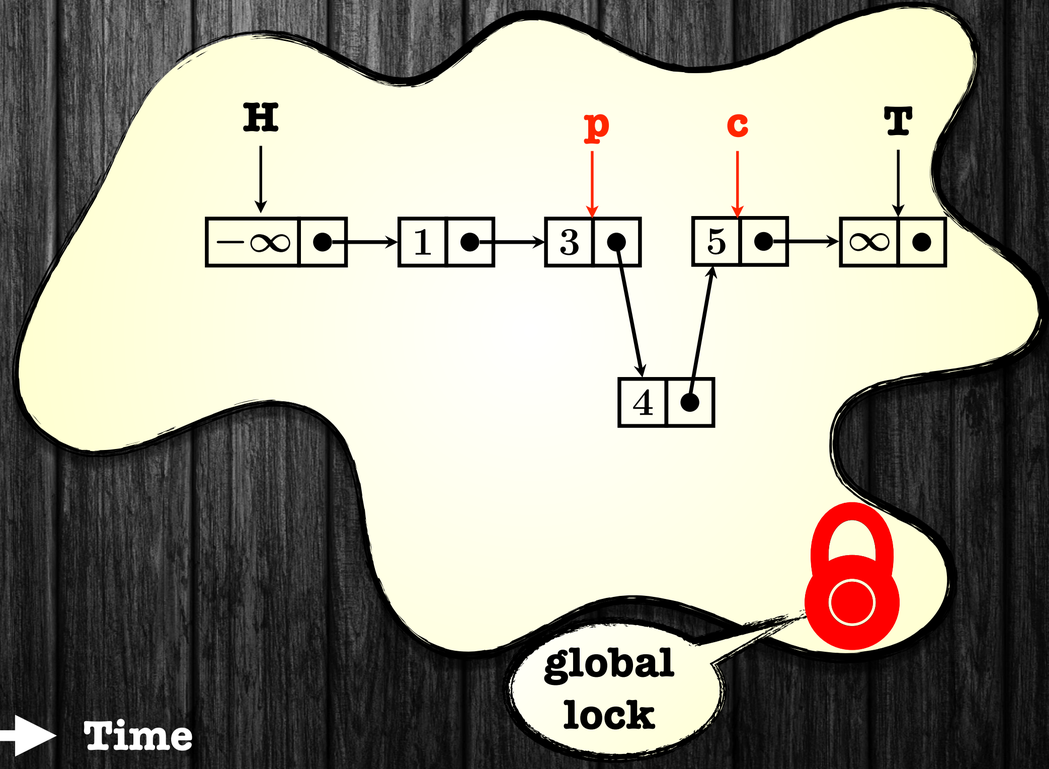
```
add(k):
•••
• lock l; LP
• •••
• unlock l
```

add(4)

add(2)

```
add(k):
•••
• lock l; LP
• •••
• unlock l
```





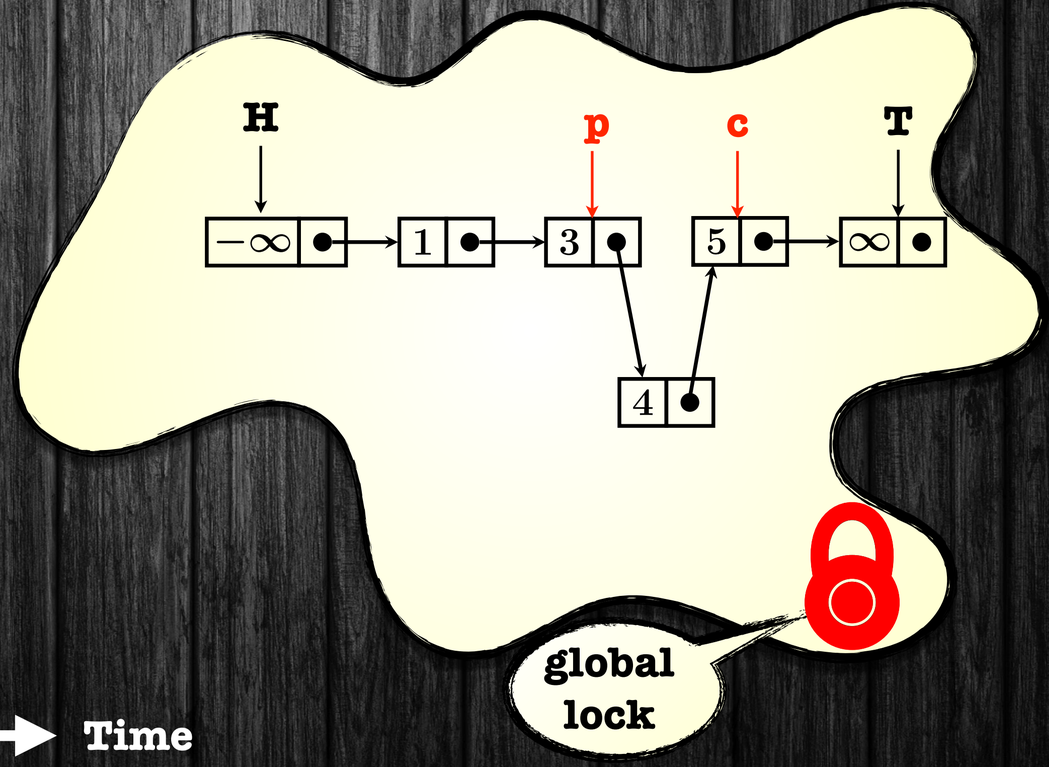
Sets Coarse-Grained Synchronization

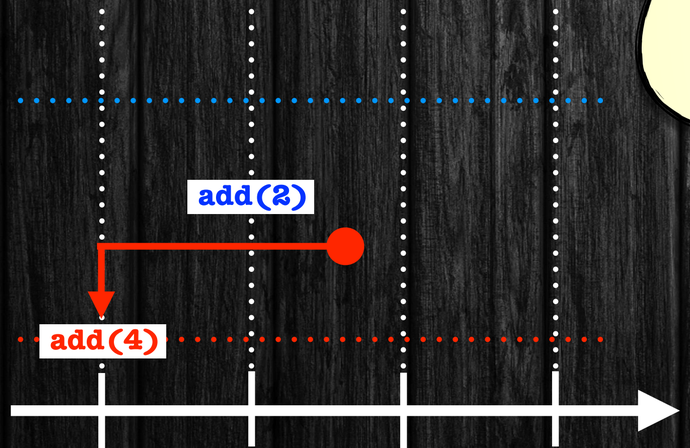
```
add(k):
•••
• lock l; LP
• •••
• unlock l
```

add(4)

add(2)

```
add(k):
•••
• lock l; LP
• •••
• unlock l
```





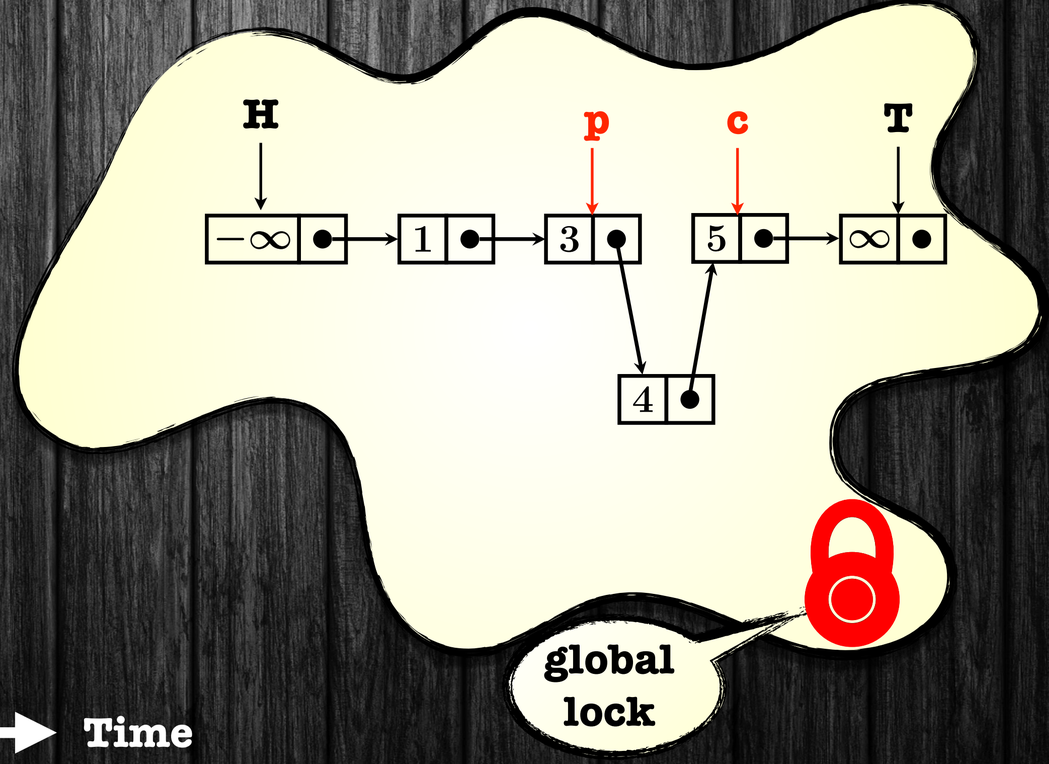
Sets Coarse-Grained Synchronization

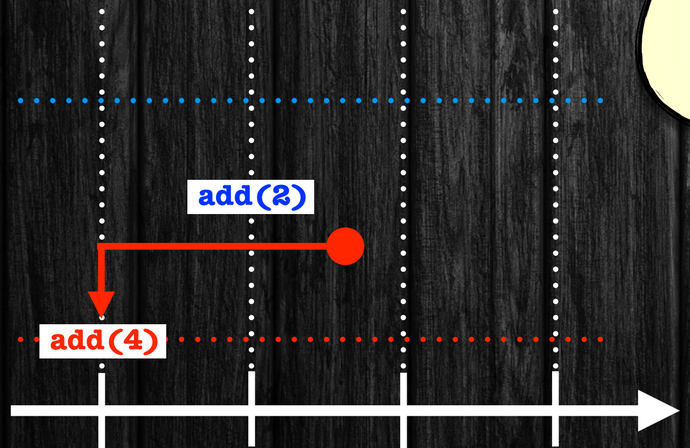
```
add(k):
•••
• lock l; LP
• •••
• unlock l
```

add(4)

add(2)

```
add(k):
•••
• lock l; LP
• •••
• unlock l
```





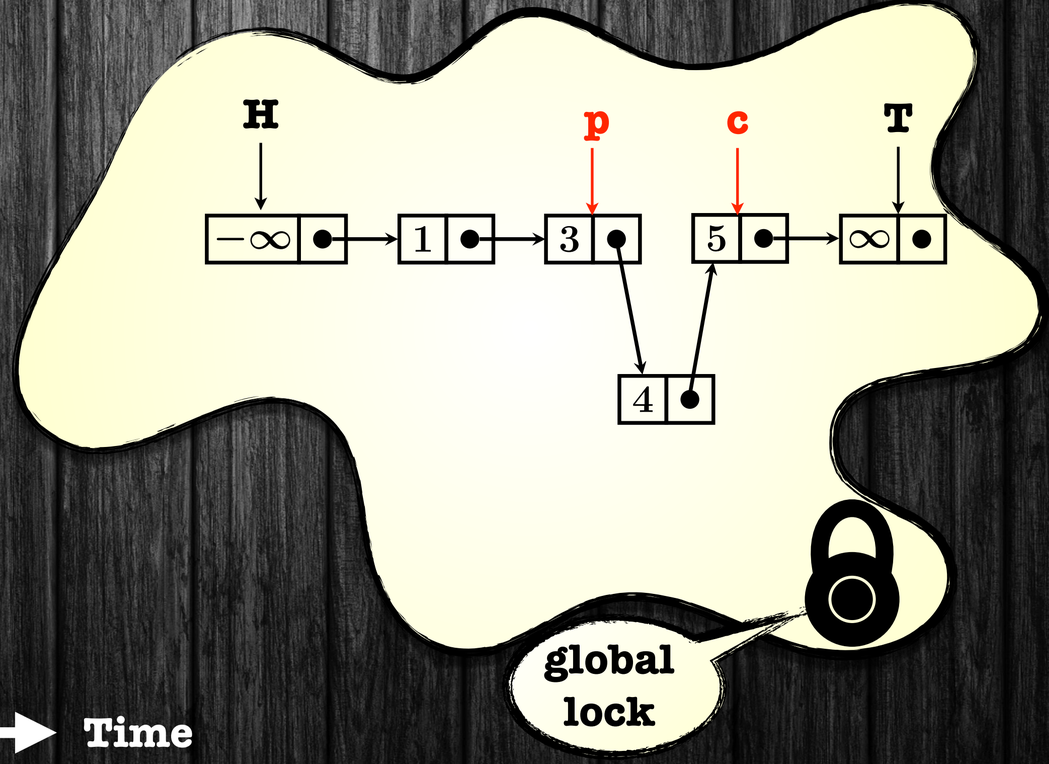
Sets Coarse-Grained Synchronization

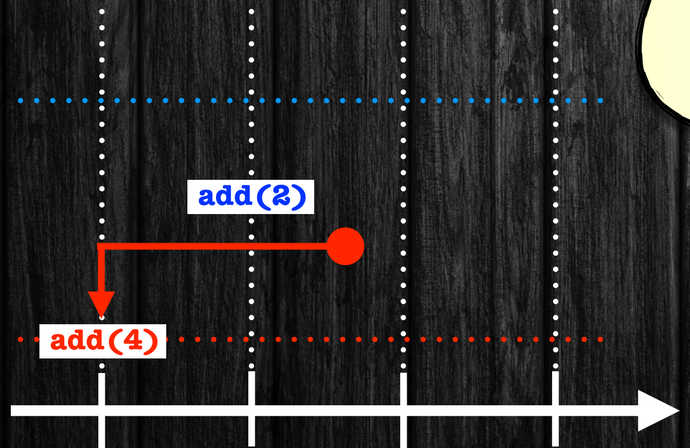
```
add(k):
•••
• lock l; LP
• •••
• unlock l
```

add(4)

add(2)

```
add(k):
•••
• lock l; LP
• •••
• unlock l
```





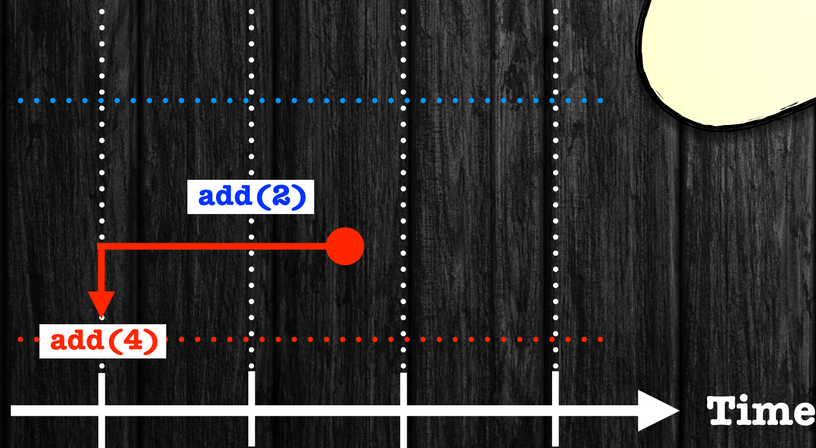
Sets Coarse-Grained Synchronization

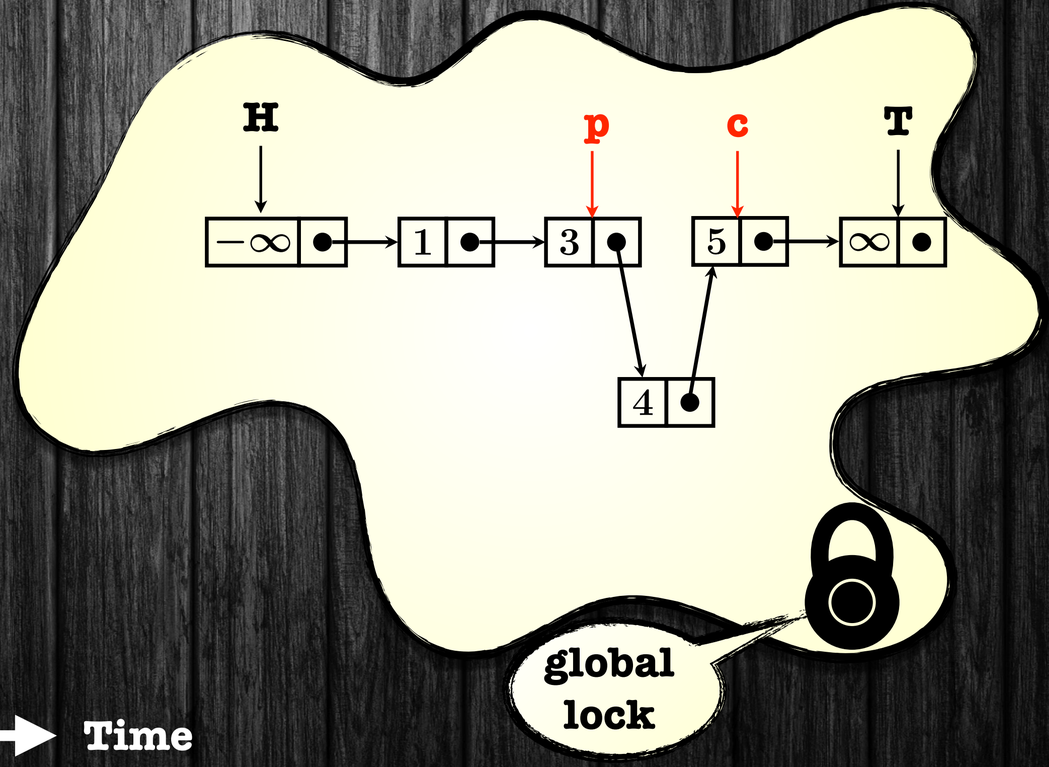
```
add(k):
•••
• lock l; LP
• •••
• unlock l
```

add(4)

add(2)

```
add(k):
•••
• lock l; LP
• •••
• unlock l
```





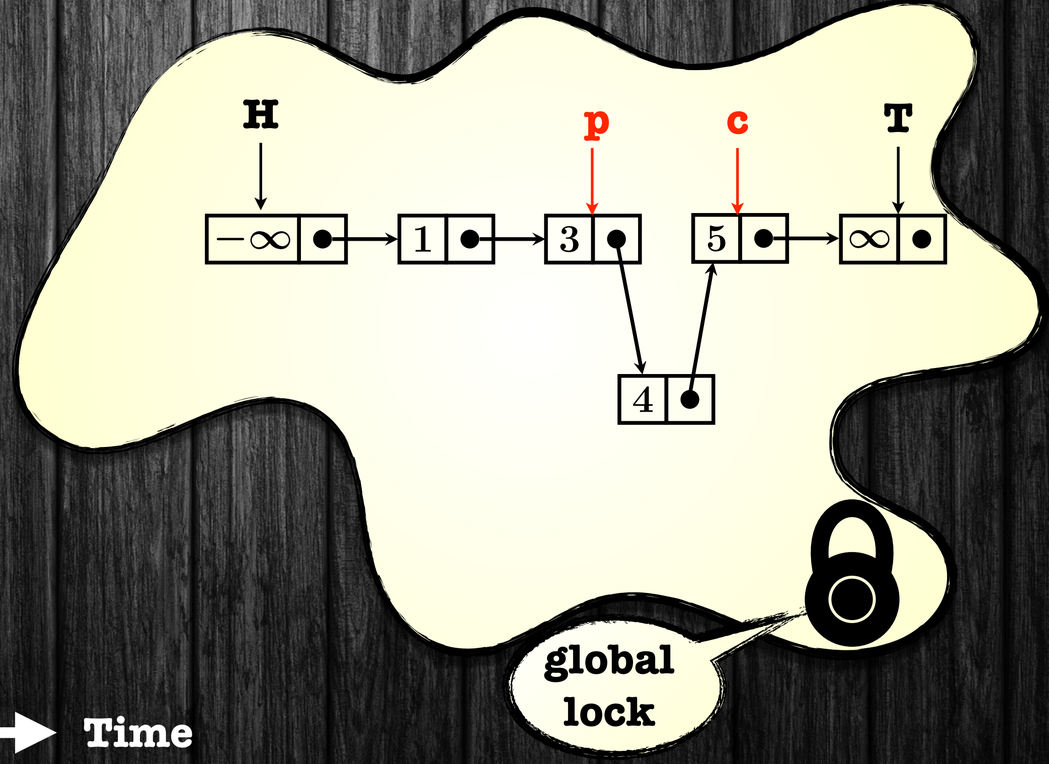
Sets Coarse-Grained Synchronization

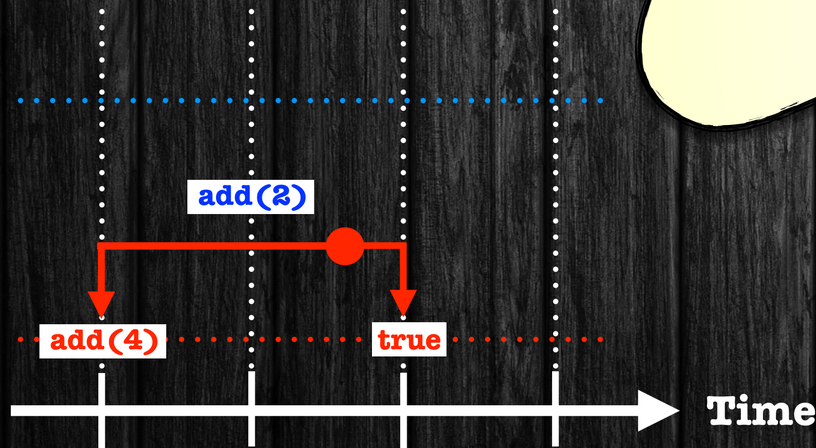
```
add(k):
•••
• lock l; LP
• •••
• unlock l
```

add(4)

add(2)

```
add(k):
•••
• lock l; LP
• •••
• unlock l
```





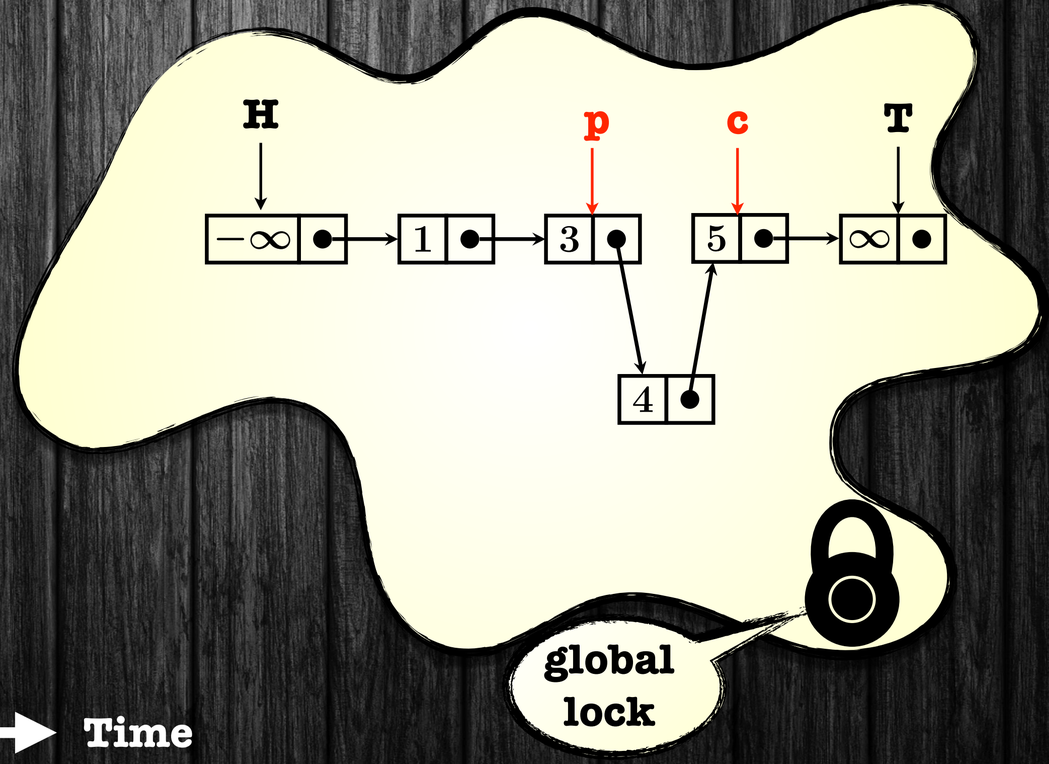
Sets Coarse-Grained Synchronization

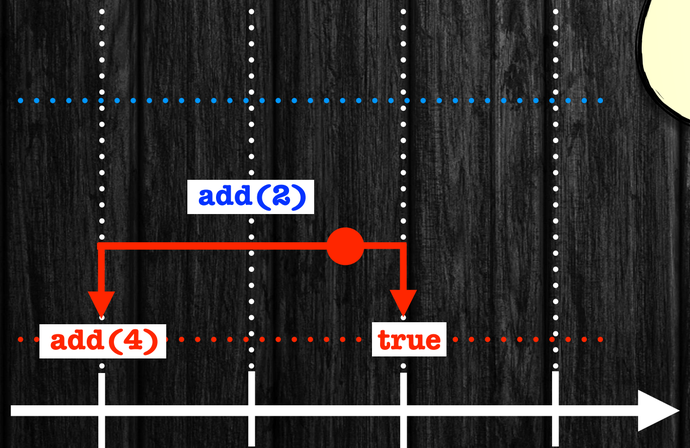
```
add(k):
•••
• lock l; LP
• •••
• unlock l
```

add(4)

add(2)

```
add(k):
•••
• lock l; LP
• •••
• unlock l
```





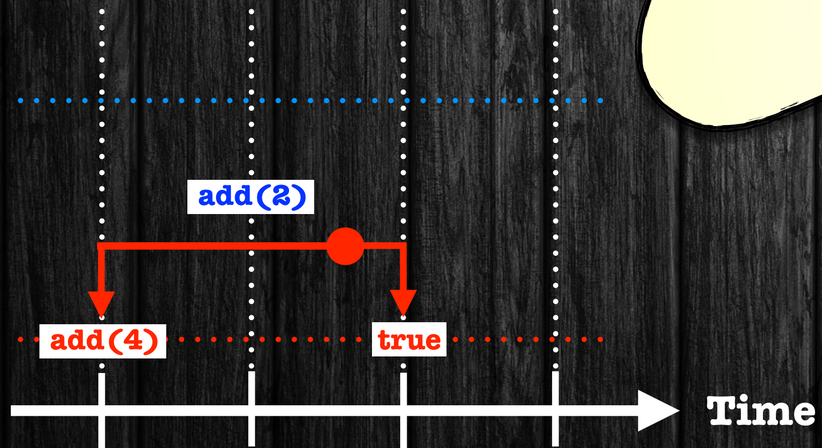
Sets Coarse-Grained Synchronization

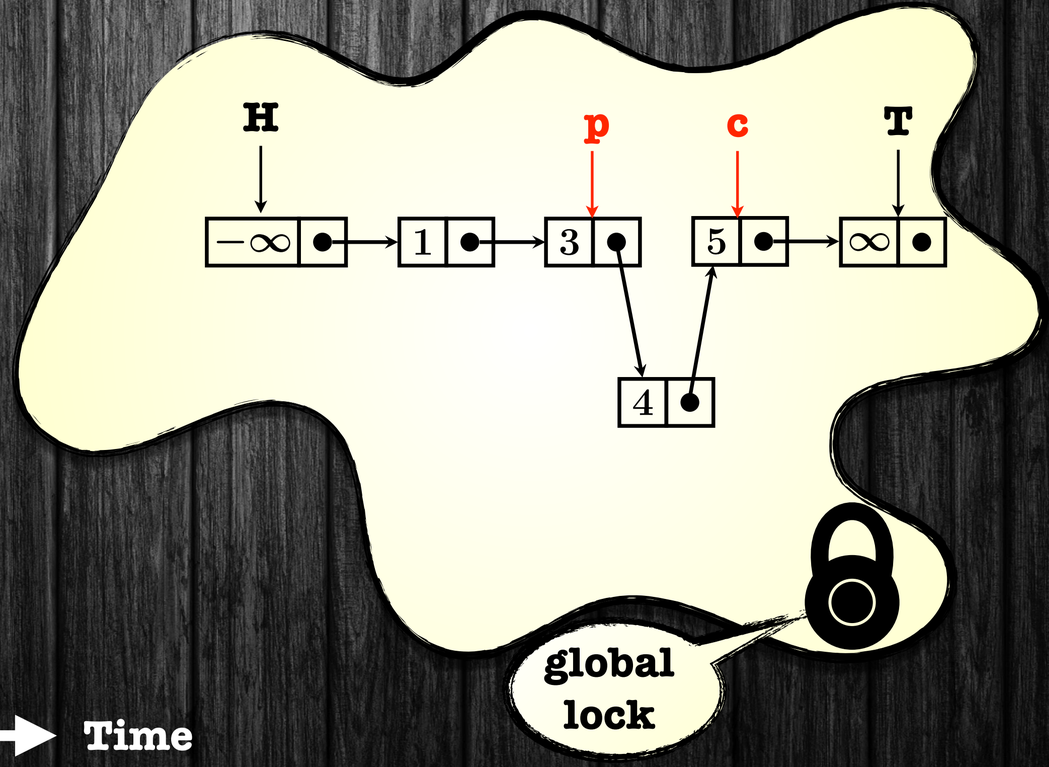
```
add(k):
•••
• lock l; LP
• •••
• unlock l
```

add(4)

add(2)

```
add(k):
•••
• lock l  LP
• •••
• unlock l
```





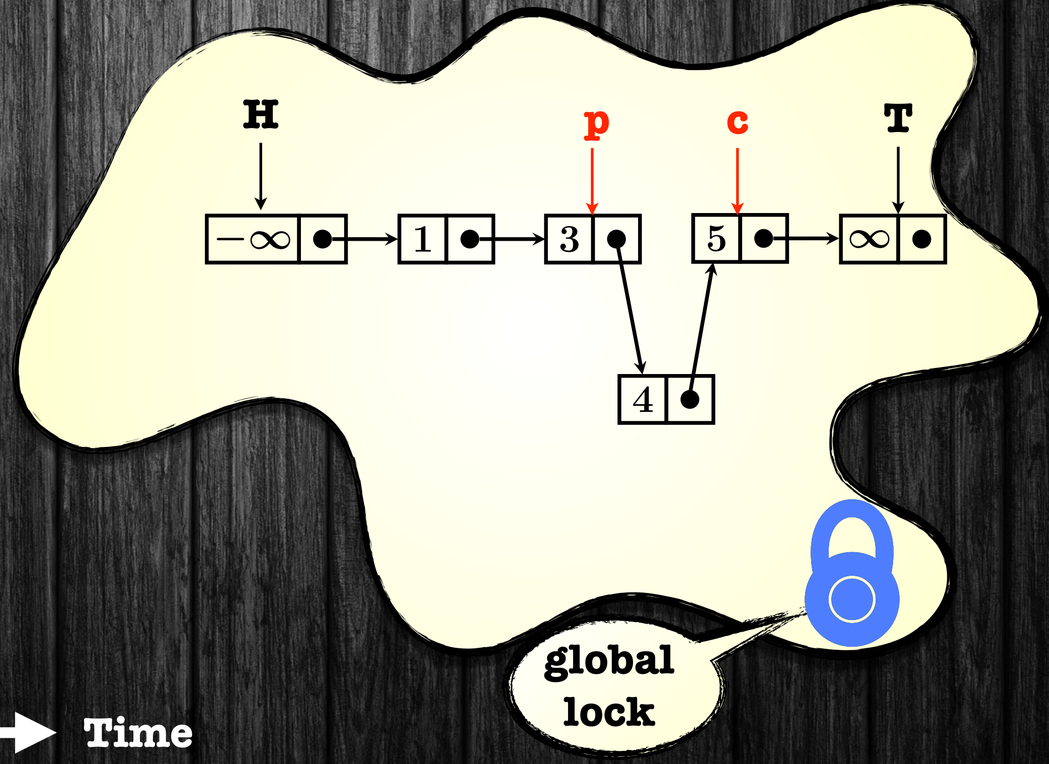
Sets Coarse-Grained Synchronization

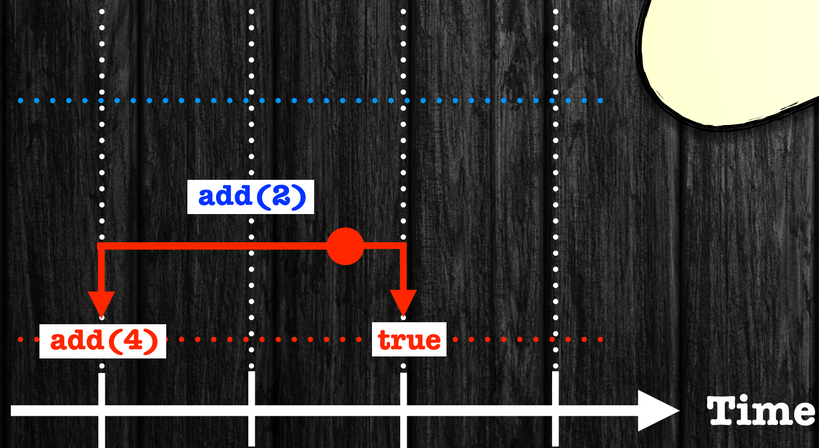
```
add(k):
•••
• lock l; LP
• •••
• unlock l
```

add(4)

add(2)

```
add(k):
•••
• lock l  LP
• •••
• unlock l
```





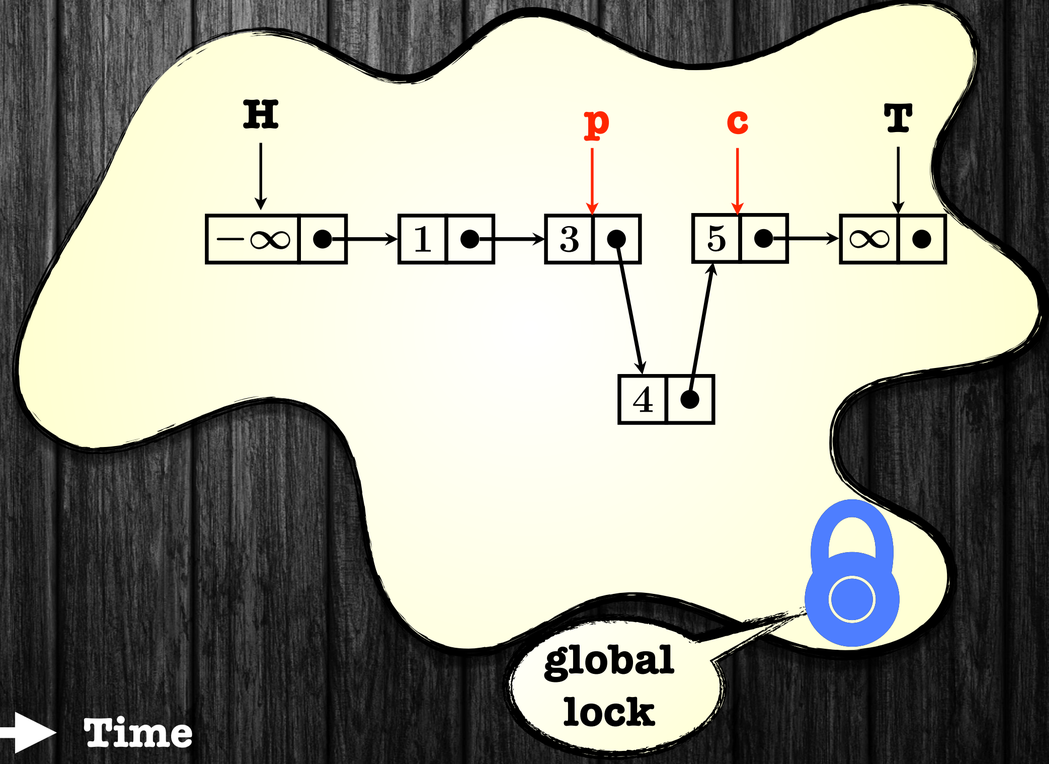
Sets Coarse-Grained Synchronization

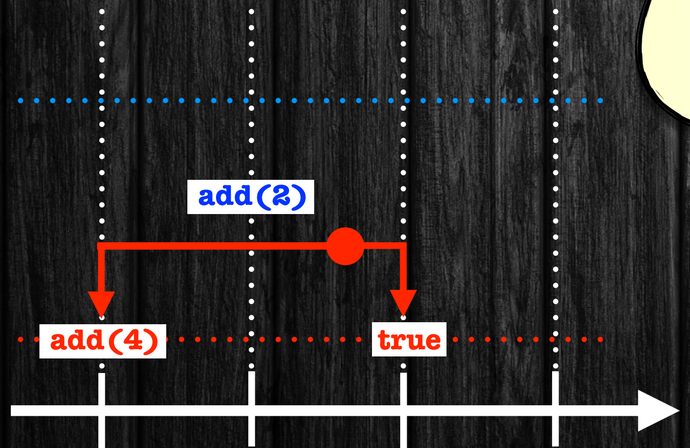
```
add(k):
•••
• lock l; LP
• •••
• unlock l
```

add(4)

add(2)

```
add(k):
•••
• lock l; LP
• •••
• unlock l
```

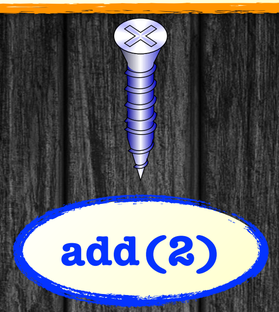




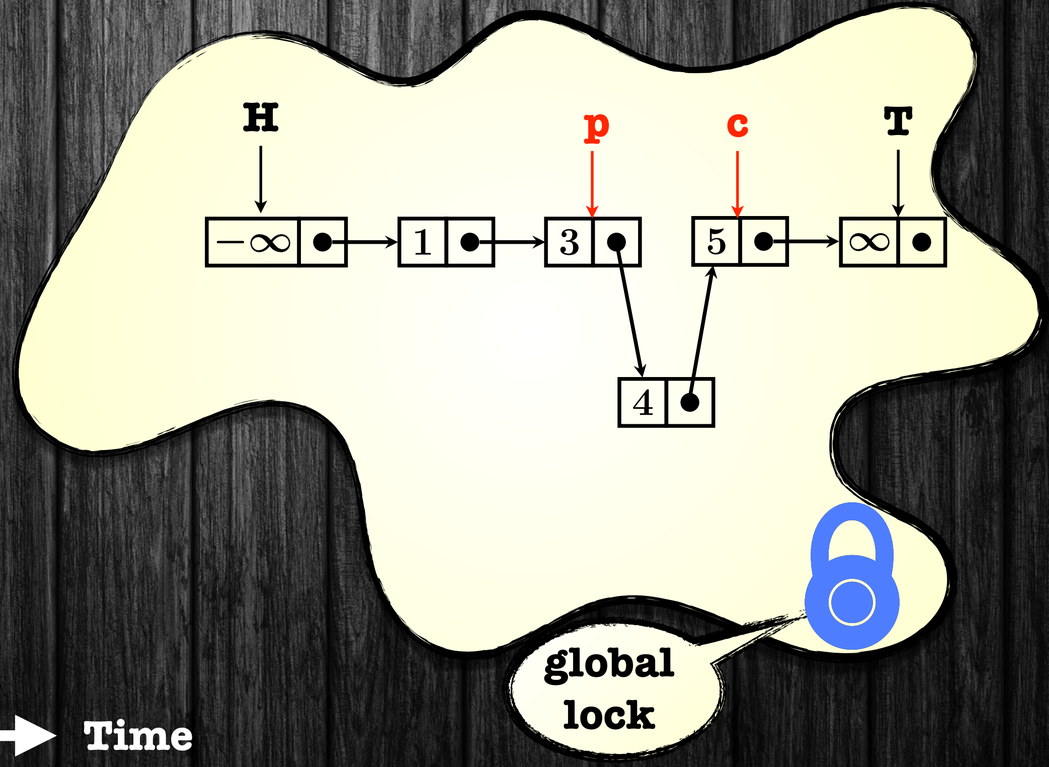
Sets Coarse-Grained Synchronization

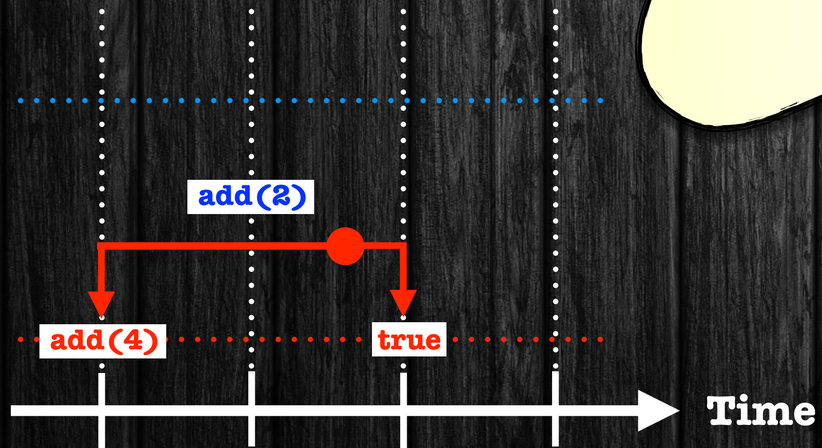
```
add(k):
•••
• lock l; LP
• •••
• unlock l
```






```
add(k):
•••
• lock l; LP
• •••
• unlock l
```





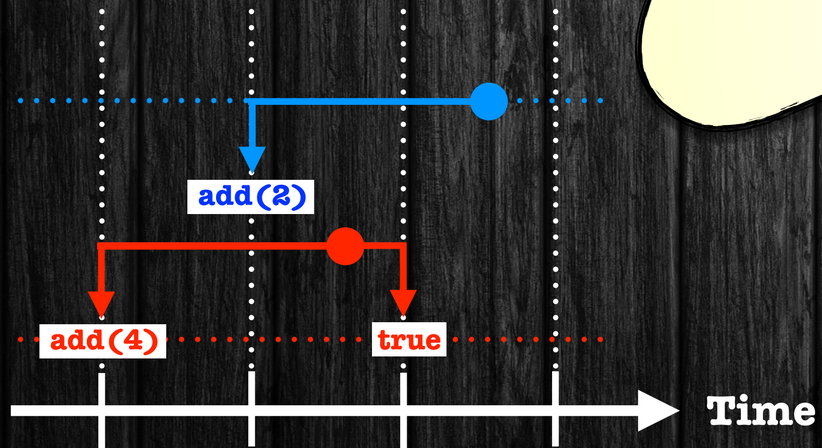
Sets Coarse-Grained Synchronization

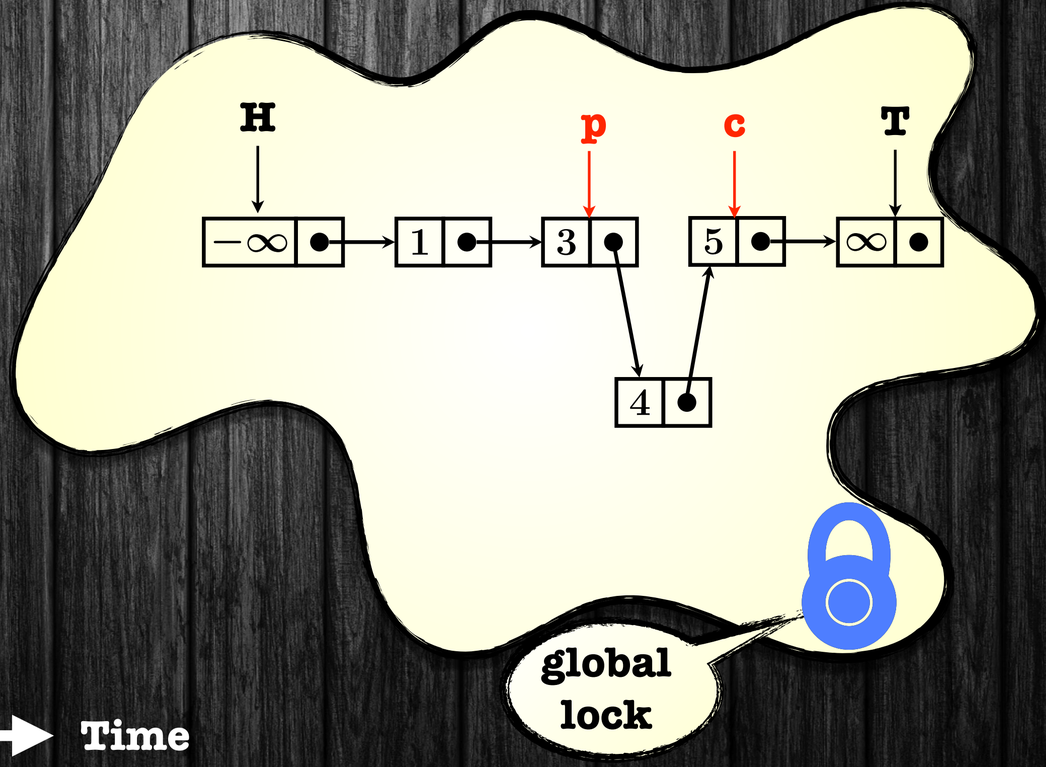
```
add(k):
•••
• lock l; LP
• •••
• unlock l
```

add(4)

add(2)

```
add(k):
•••
• lock l; LP
• •••
• unlock l
```





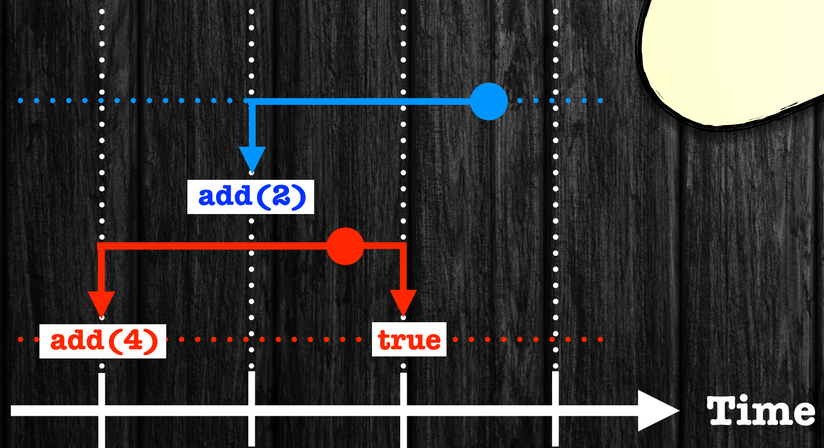
Sets Coarse-Grained Synchronization

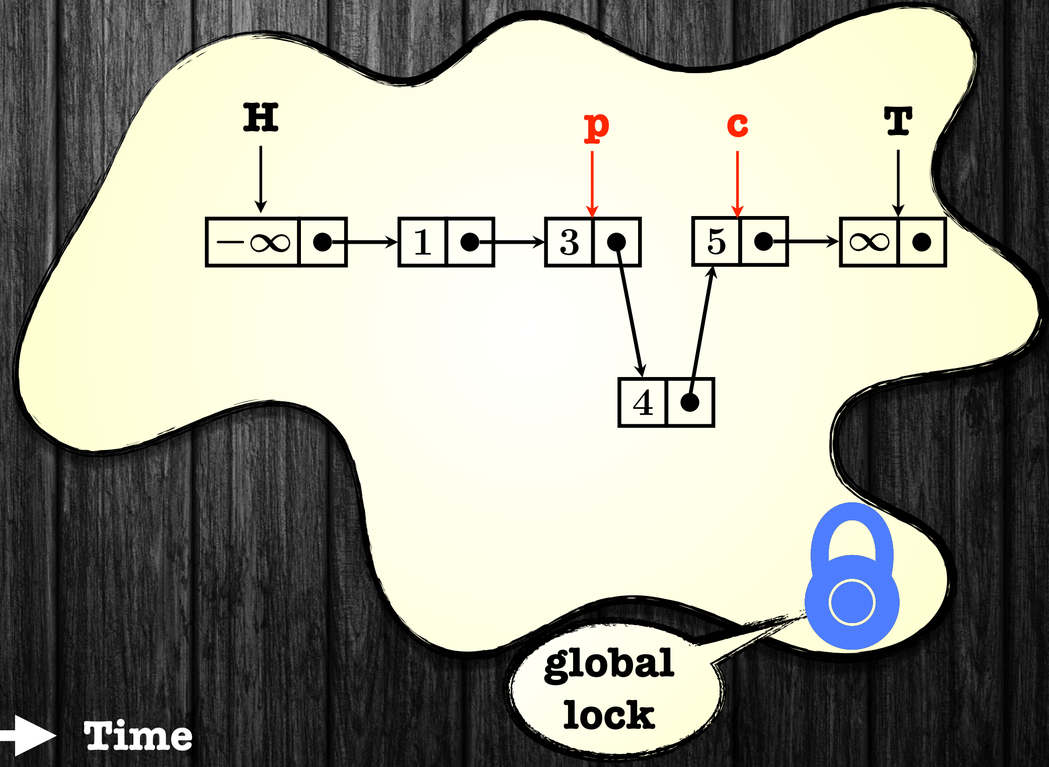
```
add(k):
•••
• lock l; LP
• •••
• unlock l
```

add(4)

add(2)

```
add(k):
•••
• lock l; LP
• •••
• unlock l
```





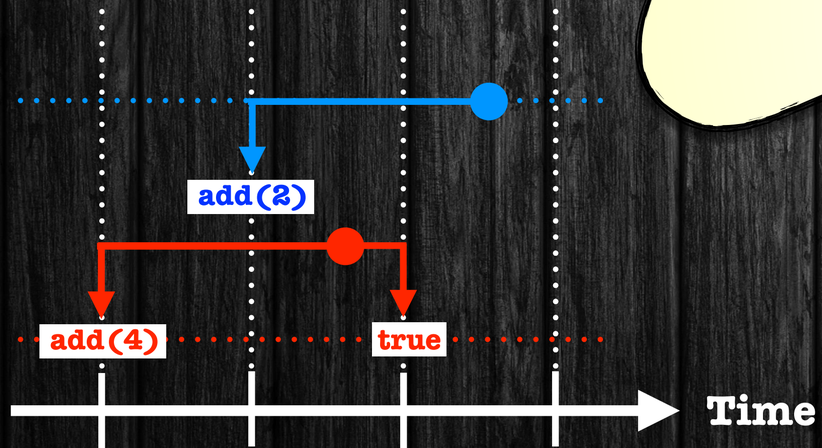
Sets Coarse-Grained Synchronization

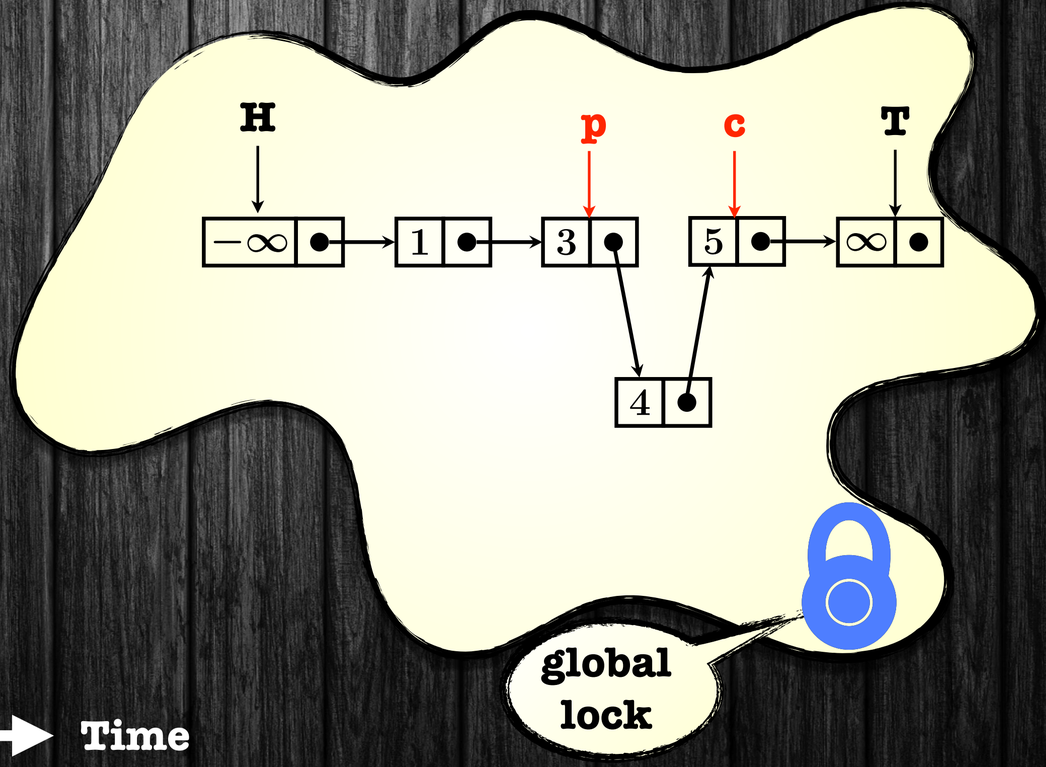
```
add(k):
•••
• lock l; LP
• •••
• unlock l
```

add(4)

add(2)

```
add(k):
•••
• lock l; LP
• •••
• unlock l
```





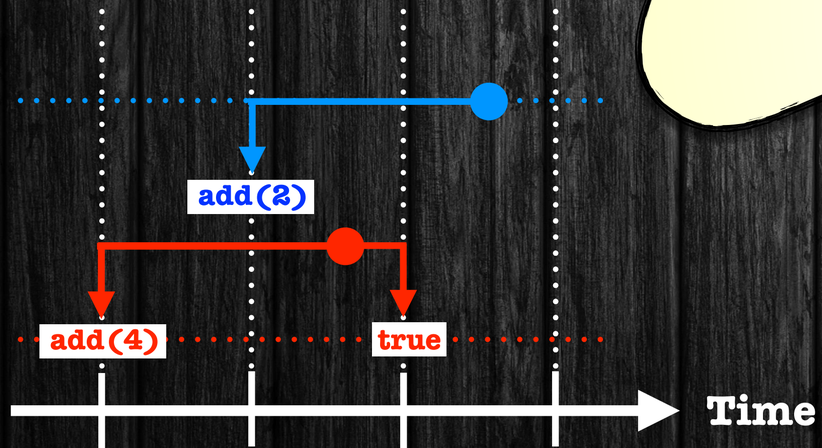
Sets Coarse-Grained Synchronization

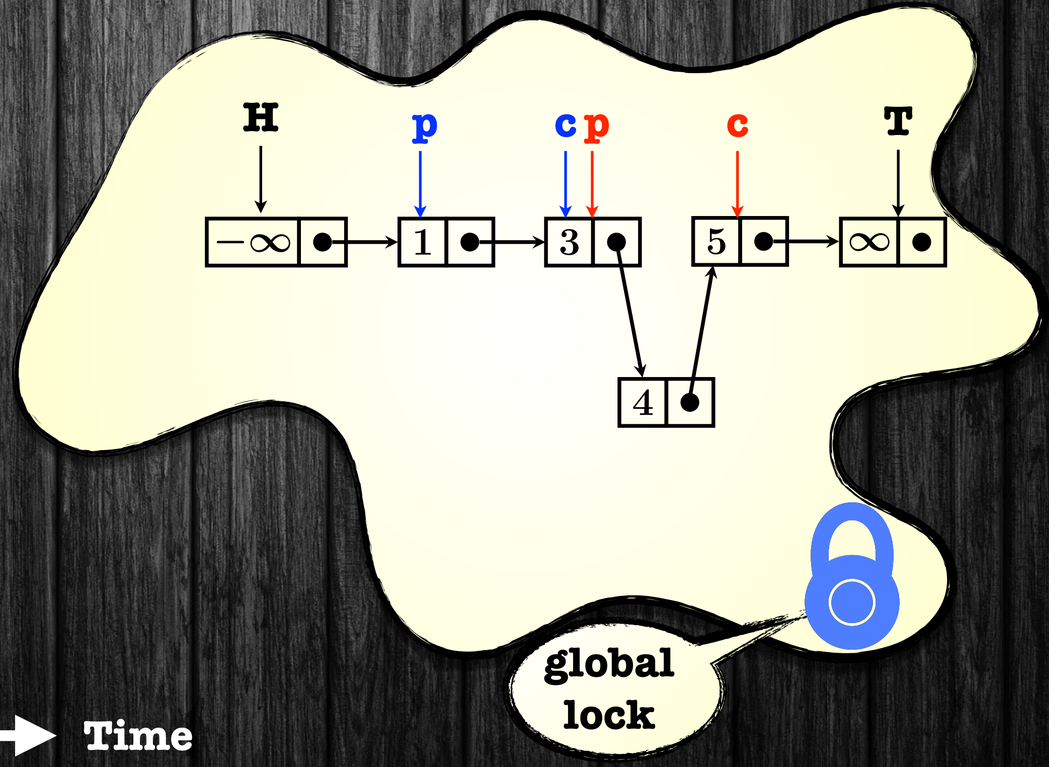
```
add(k):
•••
• lock l; LP
• •••
• unlock l
```

add(4)

add(2)

```
add(k):
•••
• lock l; LP
• •••
• unlock l
```





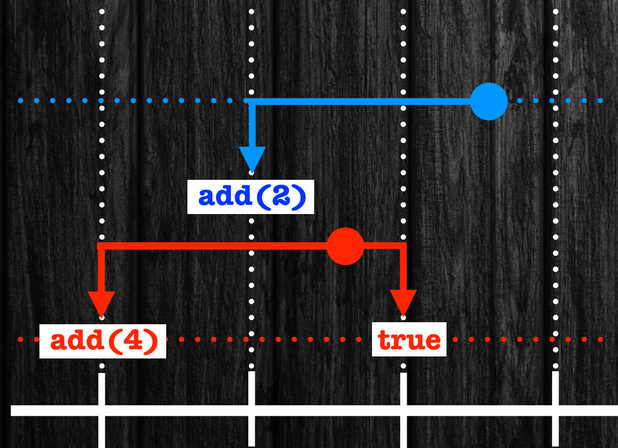
Sets Coarse-Grained Synchronization

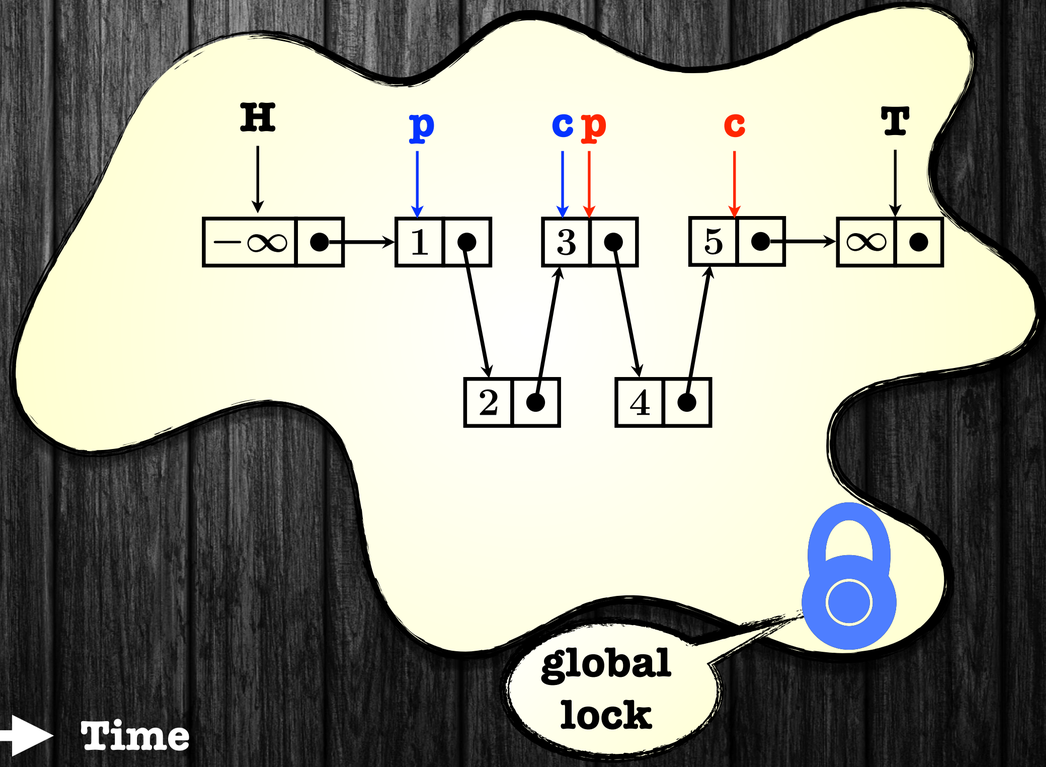
```
add(k):
•••
• lock l; LP
• •••
• unlock l
```

add(4)

add(2)

```
add(k):
•••
• lock l; LP
• •••
• unlock l
```





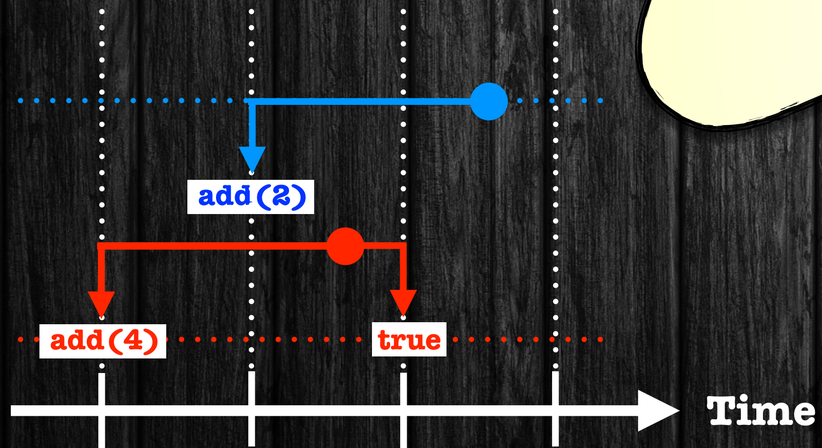
Sets Coarse-Grained Synchronization

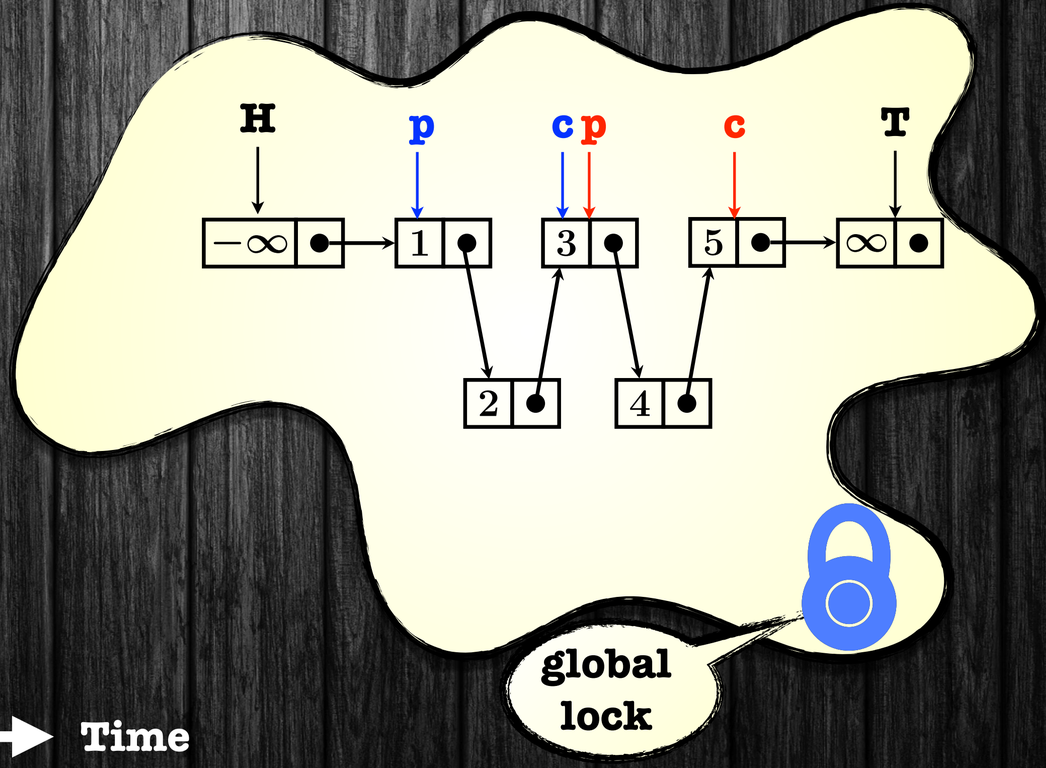
```
add(k):
•••
• lock l; LP
• •••
• unlock l
```

add(4)

add(2)

```
add(k):
•••
• lock l; LP
• •••
• unlock l
```





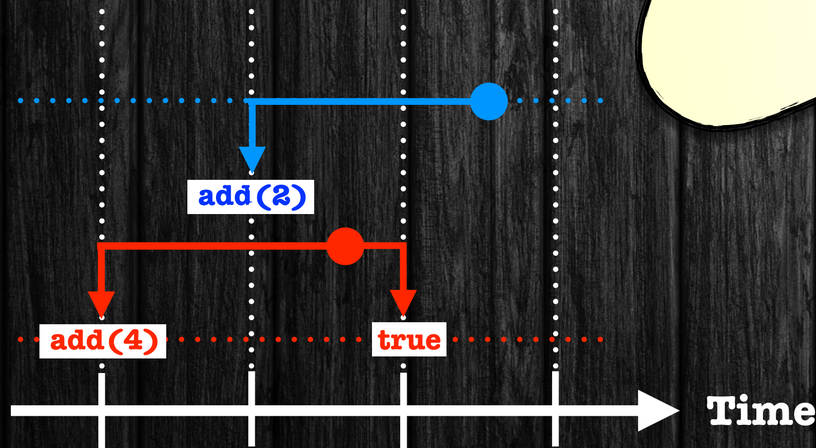
Sets Coarse-Grained Synchronization

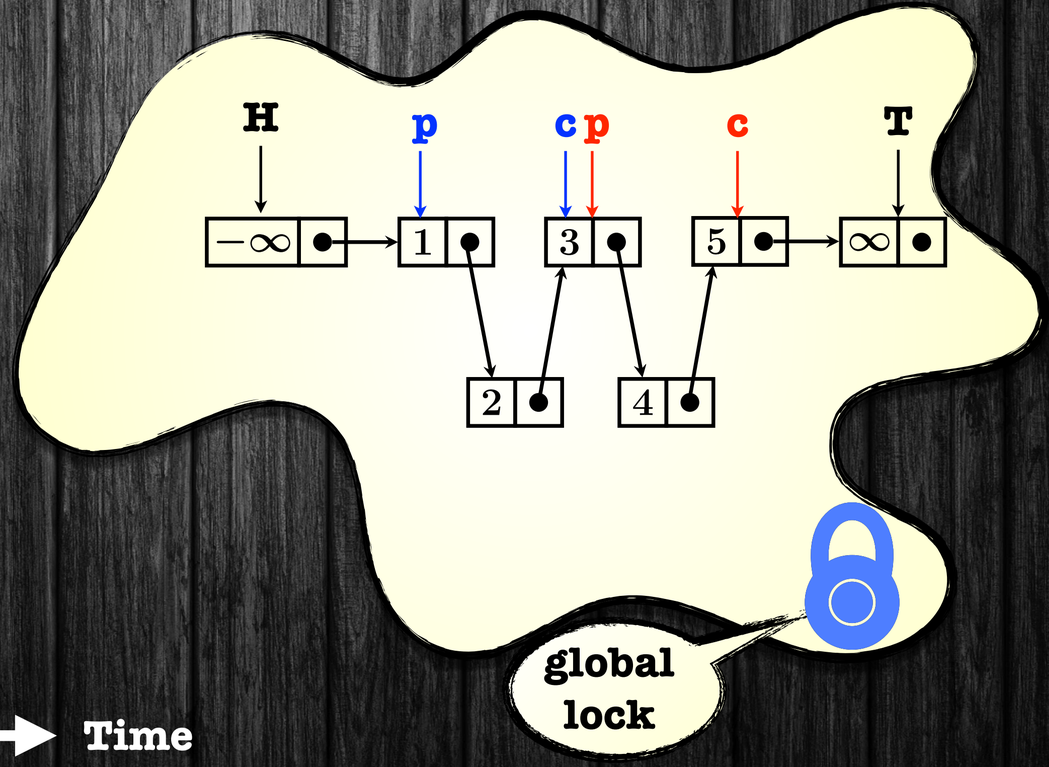
```
add(k):
•••
• lock l; LP
• •••
• unlock l
```

add(4)

add(2)

```
add(k):
•••
• lock l; LP
• •••
• unlock l
```





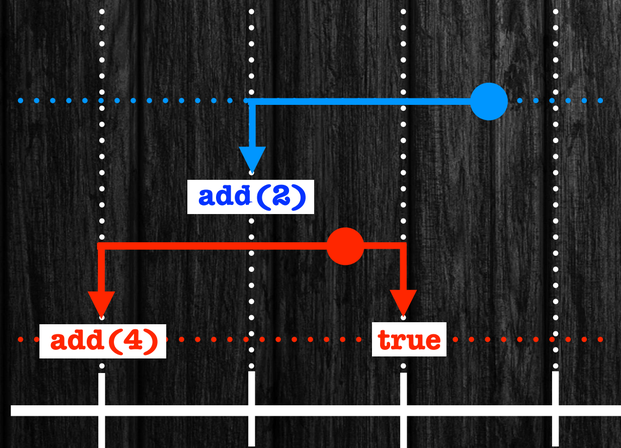
Sets Coarse-Grained Synchronization

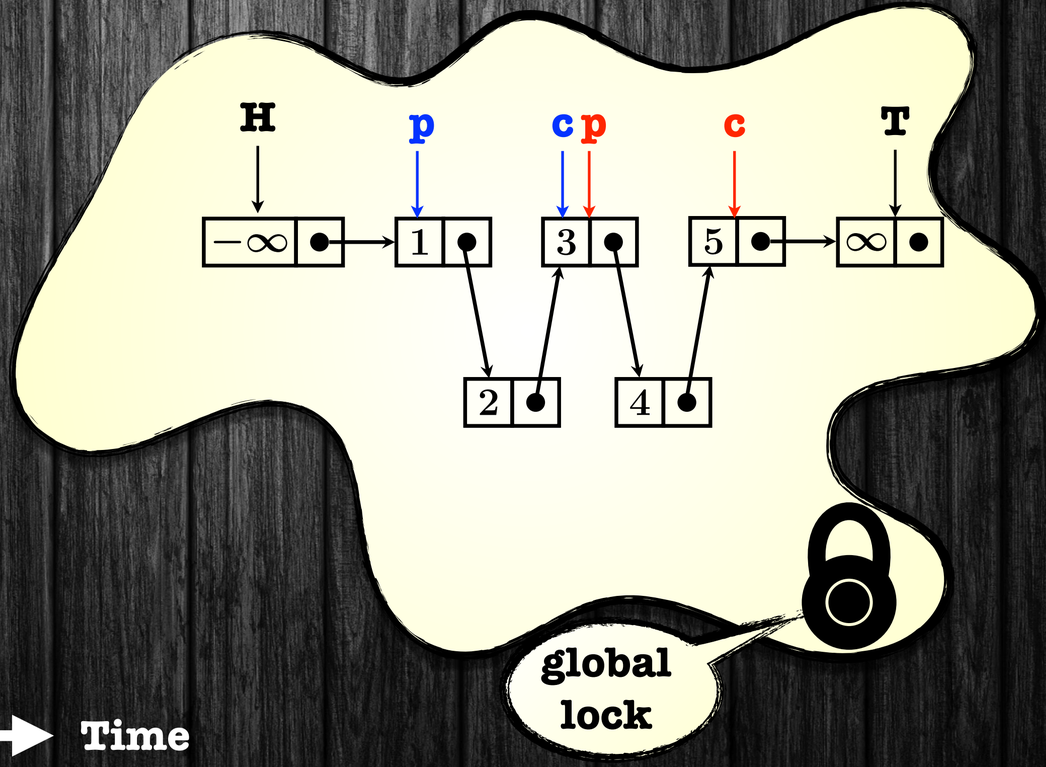
```
add(k):
•••
• lock l; LP
• •••
• unlock l
```

add(4)

add(2)

```
add(k):
•••
• lock l; LP
• •••
• unlock l
```





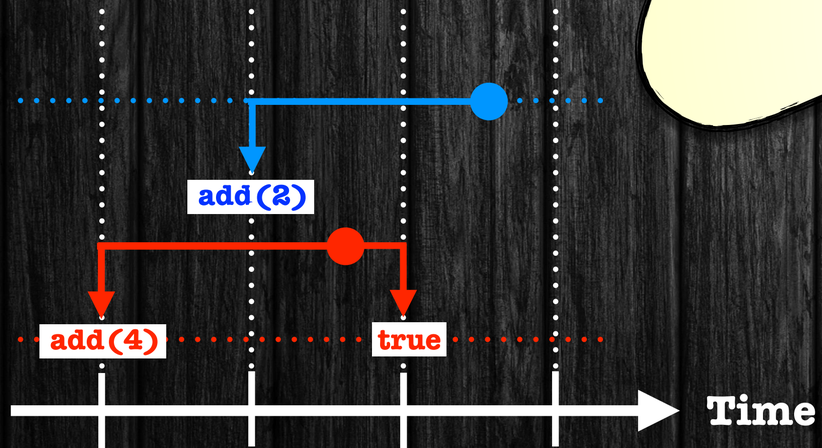
Sets Coarse-Grained Synchronization

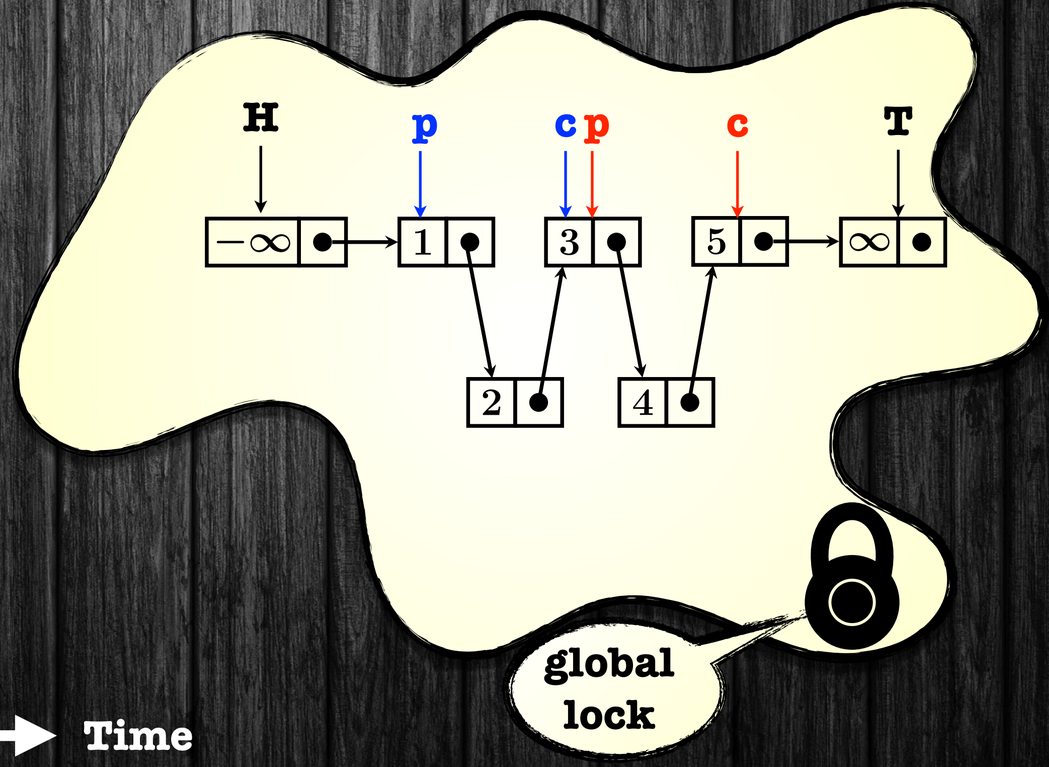
```
add(k):
•••
• lock l; LP
• •••
• unlock l
```

add(4)

add(2)

```
add(k):
•••
• lock l; LP
• •••
• unlock l
```





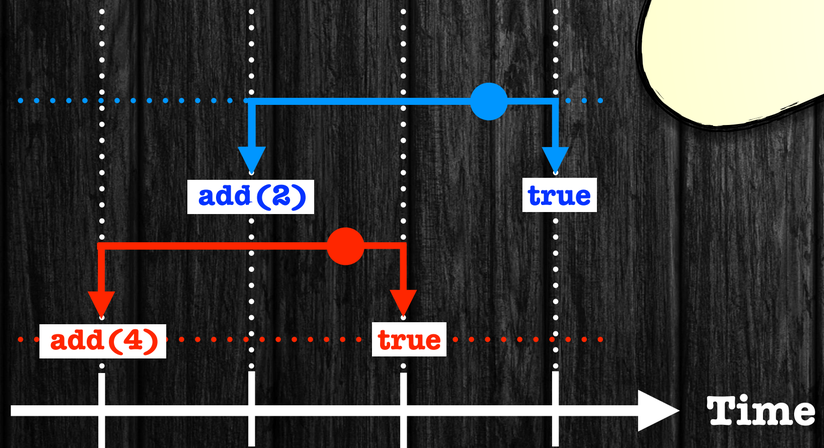
Sets Coarse-Grained Synchronization

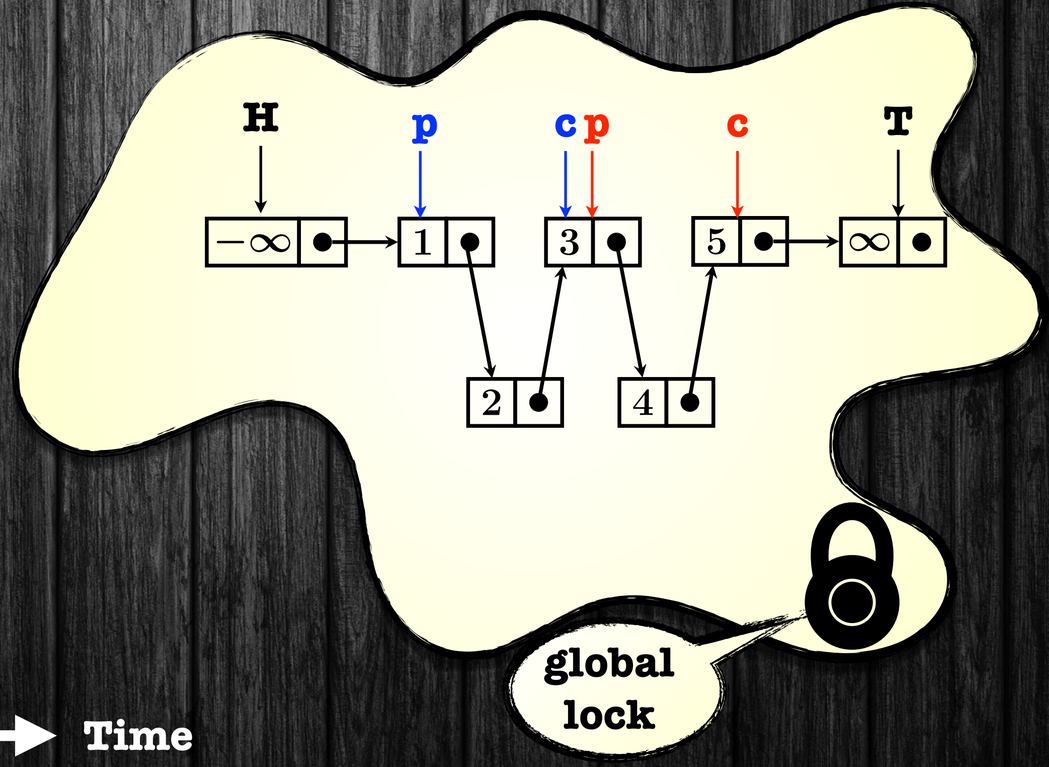
```
add(k):
•••
• lock l; LP
• •••
• unlock l
```

add(4)

add(2)

```
add(k):
•••
• lock l; LP
• •••
• unlock l
```

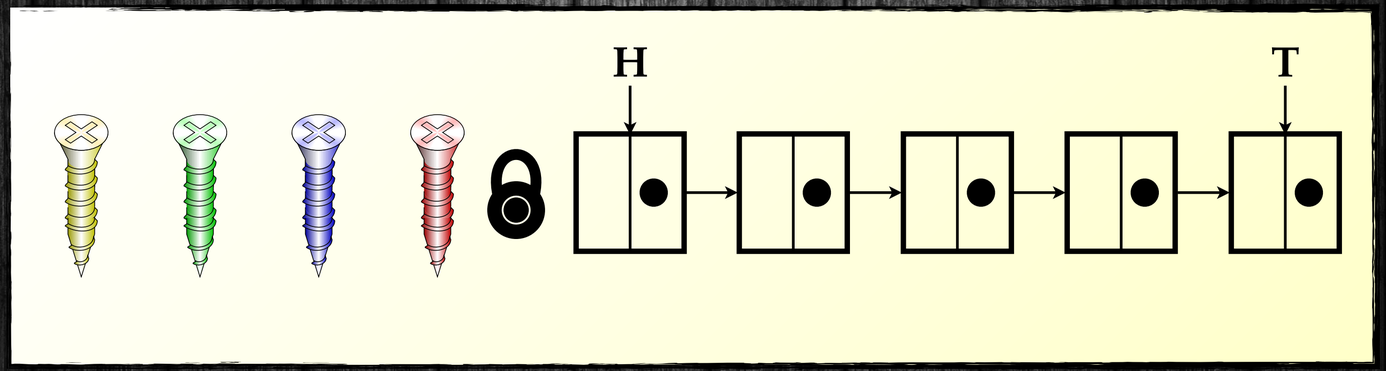




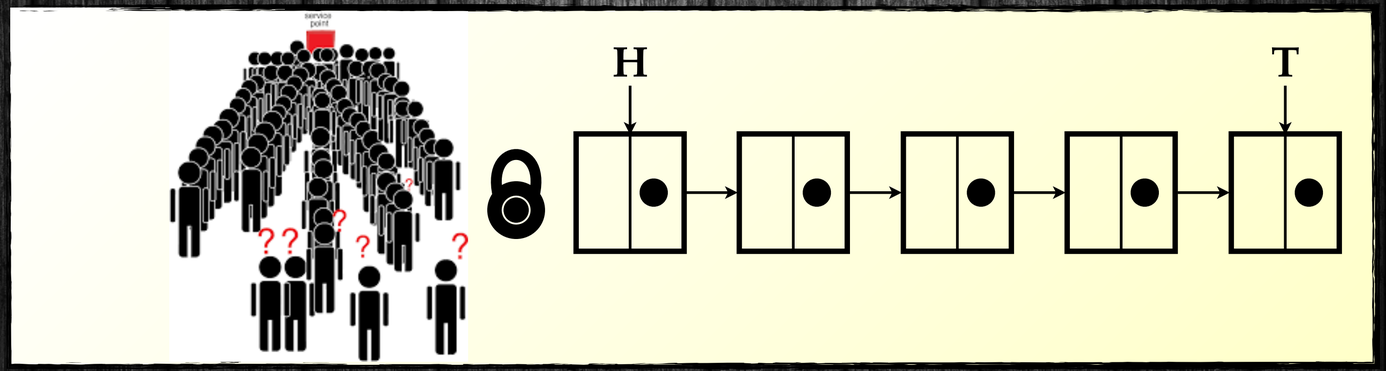
Sets | Coarse-Grained Synchronization

Sets | Coarse-Grained Synchronization

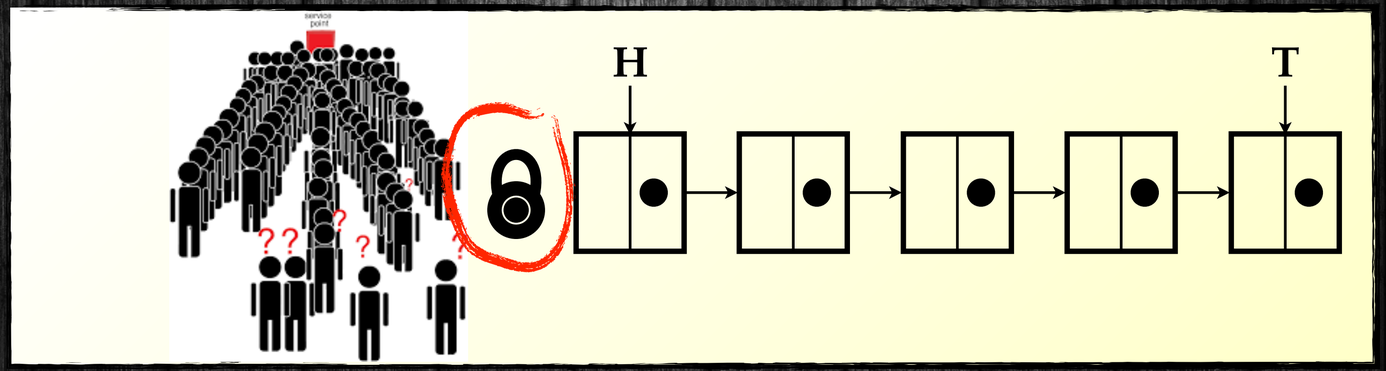
Sets Coarse-Grained Synchronization



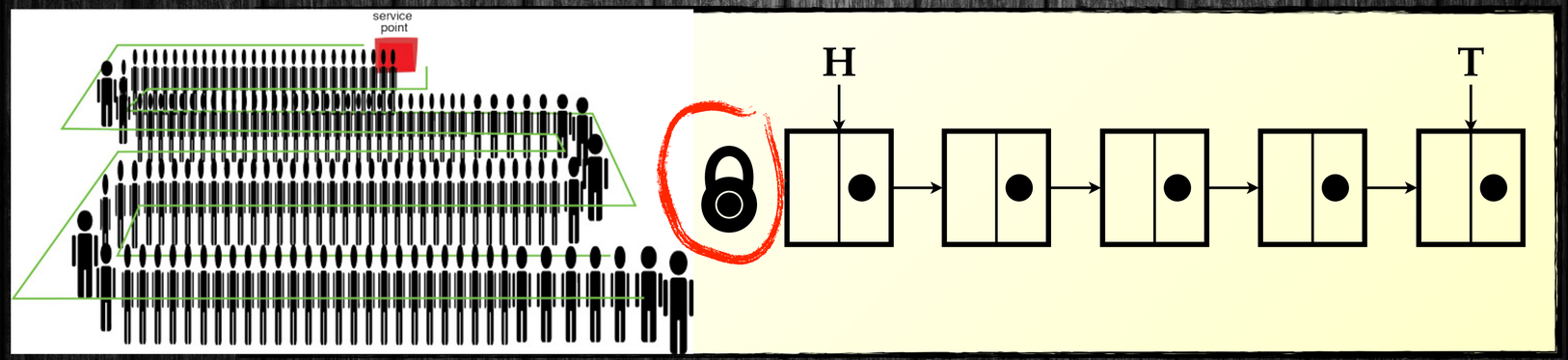
Sets Coarse-Grained Synchronization



Sets Coarse-Grained Synchronization



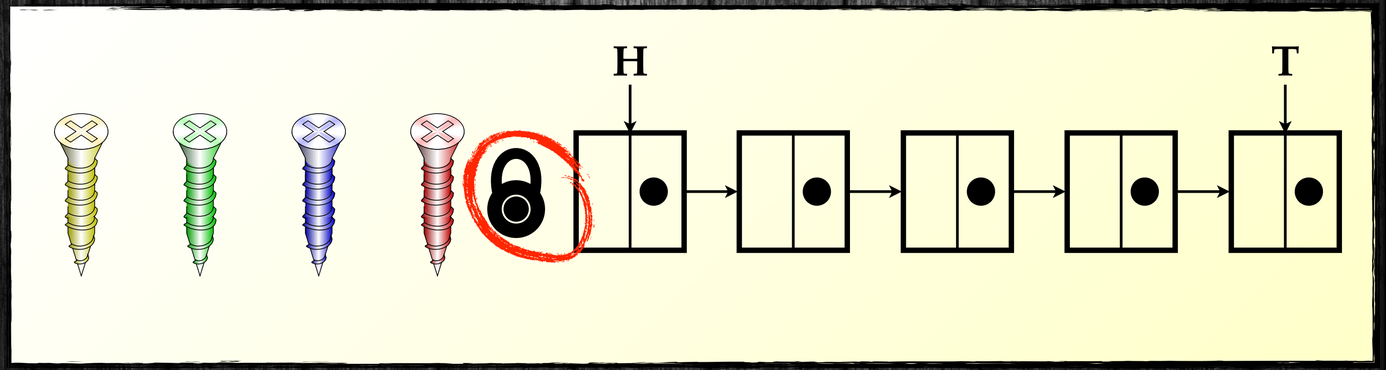
Sets Coarse-Grained Synchronization



Sets Coarse-Grained Synchronization

disadvantages

- inefficient



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

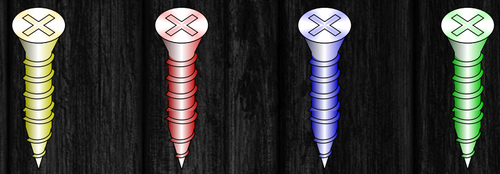
- Overview
- Abstract Data Types
- Sequential Programs
- Concurrent Programs
- Coarse-Grained Synchronization
- Fine-Grained Synchronization
- Optimistic Synchronization
- Lazy Synchronization
- Treiber Stack Algorithm
- Michael&Scott Queue Algorithm

Overview

- Sequential Programs
- Coarse-Grained Synchronization
  - Single global lock
- Fine-Grained Synchronization
  - lock per node
  - lock each visited node
- Optimistic Synchronization
  - lock when correct node found
  - restart search on failure
- Lazy Synchronization
  - make ctn wait-free



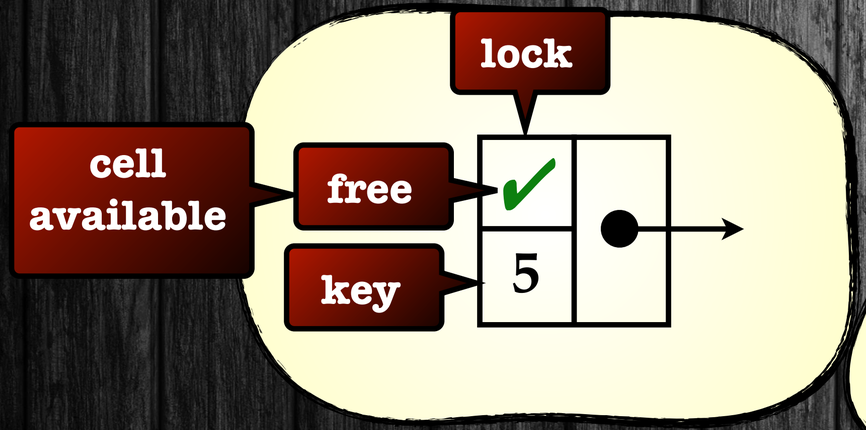
Sets Fine-Grained Synchronization

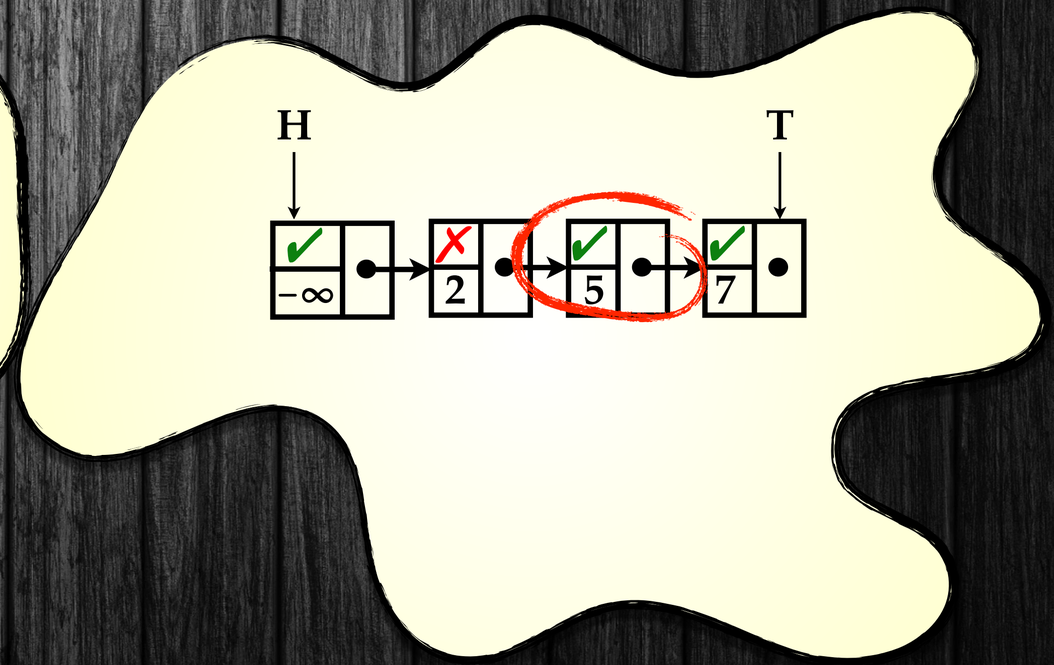


Goal: more concurrency

How?: do not lock whole data structure; lock individual cells

Strategy: "Lock Coupling"; "hand-over-hand lock acquirement"; "acquire lock for next cell before releasing the lock for the current cell"





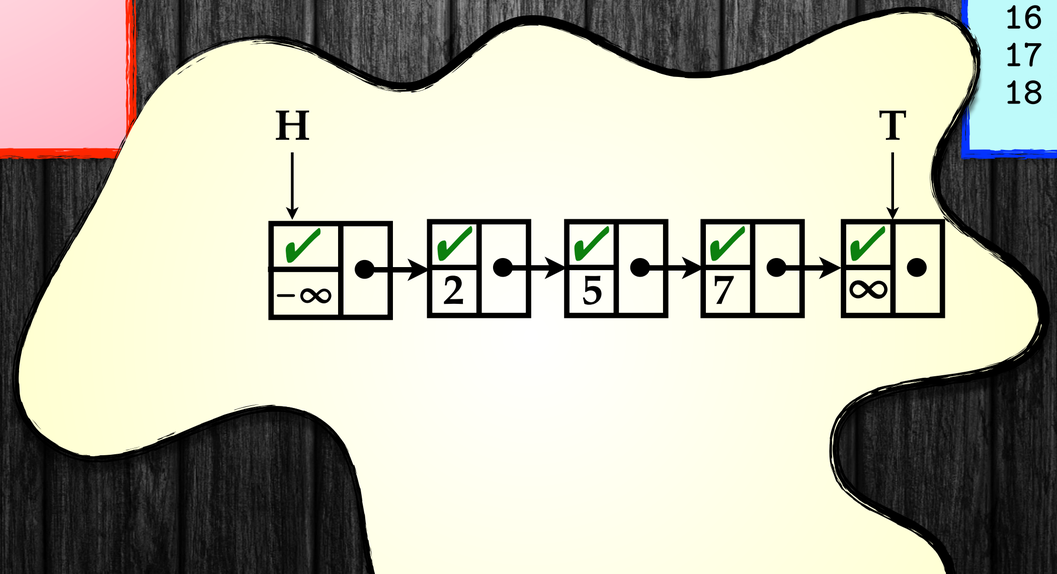
Sets | Fine-Grained Synchronization

```
add(k):
Node p, c
1  lock(H)
2  p = H
3  c = p.next;
4  lock(c);
5  while (c.key < k)
6    unlock(p);
7    p = c;
8    c = c.next;
9    lock(c);
10 if (c.key = k)
11   return false
12 else
13   n = new Node(k,-)
14   n.next = c
15   p.next = n
16   return true
17 unlock(c)
18 unlock(p)
```

add(6)

add(1)


```
add(k):
Node p, c
1  lock(H)
2  p = H
3  c = p.next;
4  lock(c);
5  while (c.key < k)
6    unlock(p);
7    p = c;
8    c = c.next;
9    lock(c);
10 if (c.key = k)
11   return false
12 else
13   n = new Node(k,-)
14   n.next = c
15   p.next = n
16   return true
17 unlock(c)
18 unlock(p)
```

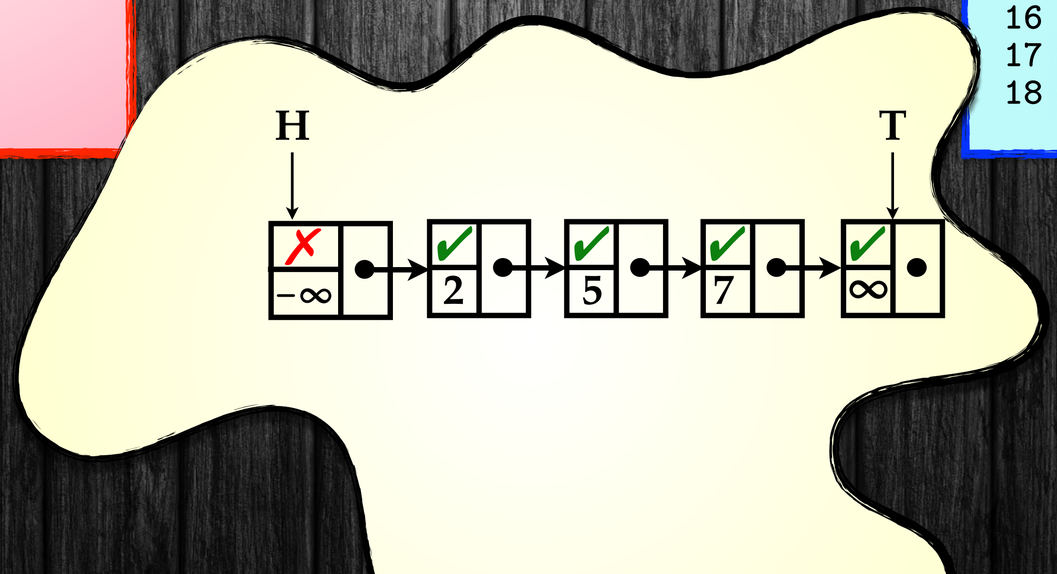
Sets | Fine-Grained Synchronization

```
add(k):
Node p, c
1  lock(H)
2  p = H
3  c = p.next;
4  lock(c);
5  while (c.key < k)
6    unlock(p);
7    p = c;
8    c = c.next;
9    lock(c);
10 if (c.key = k)
11   return false
12 else
13   n = new Node(k,-)
14   n.next = c
15   p.next = n
16   return true
17 unlock(c)
18 unlock(p)
```

add(6)

add(1)


```
add(k):
Node p, c
1  lock(H)
2  p = H
3  c = p.next;
4  lock(c);
5  while (c.key < k)
6    unlock(p);
7    p = c;
8    c = c.next;
9    lock(c);
10 if (c.key = k)
11   return false
12 else
13   n = new Node(k,-)
14   n.next = c
15   p.next = n
16   return true
17 unlock(c)
18 unlock(p)
```

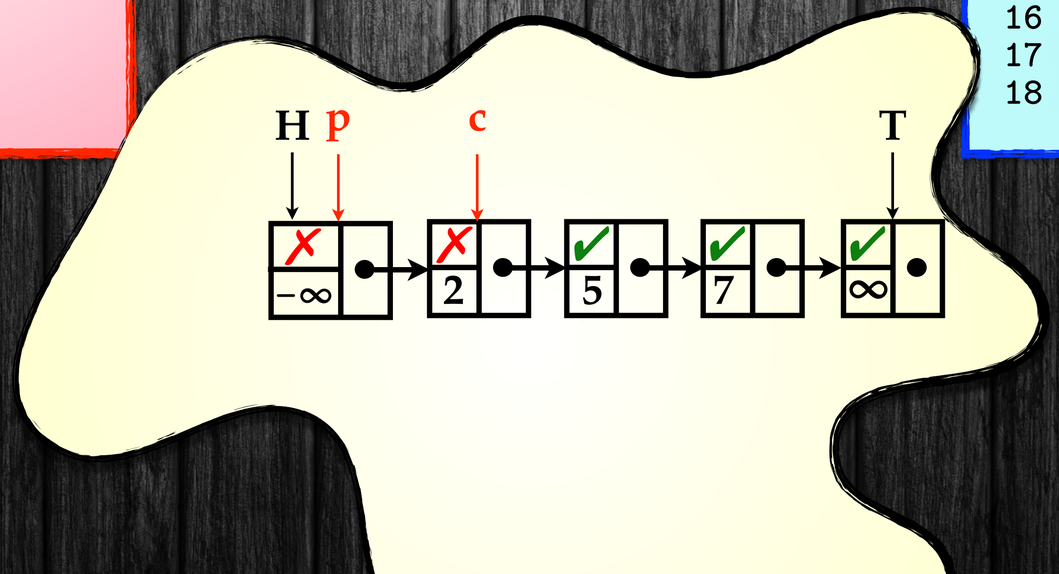
Sets | Fine-Grained Synchronization

```
add(k):
Node p, c
1  lock(H)
2  p = H
3  c = p.next;
4  lock(c);
5  while (c.key < k)
6    unlock(p);
7    p = c;
8    c = c.next;
9    lock(c);
10 if (c.key = k)
11   return false
12 else
13   n = new Node(k,-)
14   n.next = c
15   p.next = n
16   return true
17 unlock(c)
18 unlock(p)
```

add(6)

add(1)

```
add(k):
Node p, c
1  lock(H)
2  p = H
3  c = p.next;
4  lock(c);
5  while (c.key < k)
6    unlock(p);
7    p = c;
8    c = c.next;
9    lock(c);
10 if (c.key = k)
11   return false
12 else
13   n = new Node(k,-)
14   n.next = c
15   p.next = n
16   return true
17 unlock(c)
18 unlock(p)
```



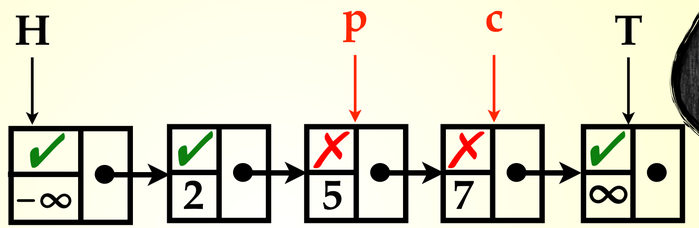
Sets | Fine-Grained Synchronization

```
add(k):
Node p, c
1  lock(H)
2  p = H
3  c = p.next;
4  lock(c);
5  while (c.key < k)
6    unlock(p);
7    p = c;
8    c = c.next;
9    lock(c);
10 if (c.key = k)
11   return false
12 else
13   n = new Node(k,-)
14   n.next = c
15   p.next = n
16   return true
17 unlock(c)
18 unlock(p)
```

add(6)

add(1)

```
add(k):
Node p, c
1  lock(H)
2  p = H
3  c = p.next;
4  lock(c);
5  while (c.key < k)
6    unlock(p);
7    p = c;
8    c = c.next;
9    lock(c);
10 if (c.key = k)
11   return false
12 else
13   n = new Node(k,-)
14   n.next = c
15   p.next = n
16   return true
17 unlock(c)
18 unlock(p)
```



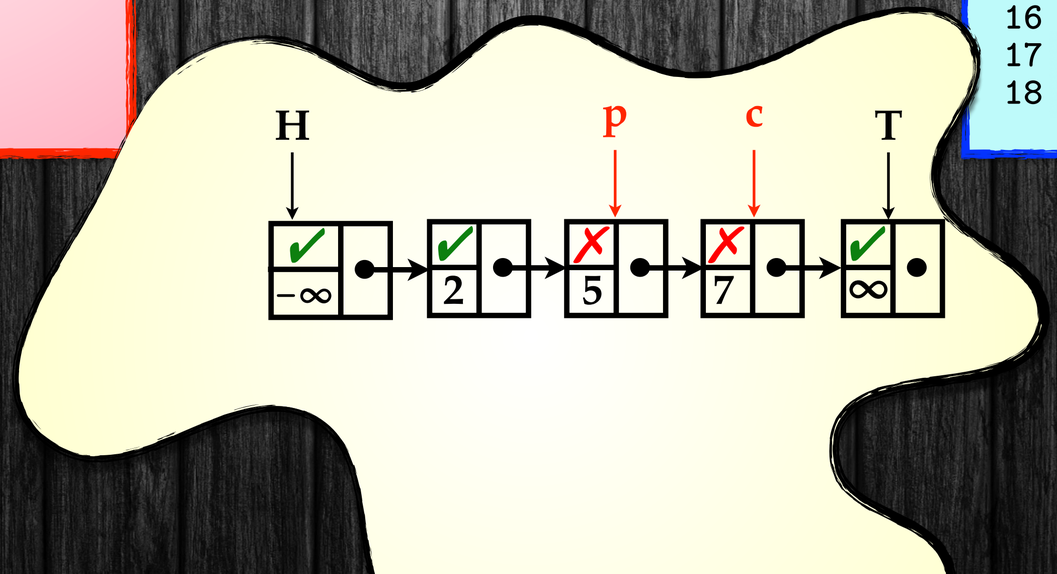
Sets | Fine-Grained Synchronization

```
add(k):
Node p, c
1  lock(H)
2  p = H
3  c = p.next;
4  lock(c);
5  while (c.key < k)
6    unlock(p);
7    p = c;
8    c = c.next;
9    lock(c);
10 if (c.key = k)
11   return false
12 else
13   n = new Node(k,-)
14   n.next = c
15   p.next = n
16   return true
17 unlock(c)
18 unlock(p)
```

add(6)

add(1)

```
add(k):
Node p, c
1  lock(H)
2  p = H
3  c = p.next;
4  lock(c);
5  while (c.key < k)
6    unlock(p);
7    p = c;
8    c = c.next;
9    lock(c);
10 if (c.key = k)
11   return false
12 else
13   n = new Node(k,-)
14   n.next = c
15   p.next = n
16   return true
17 unlock(c)
18 unlock(p)
```



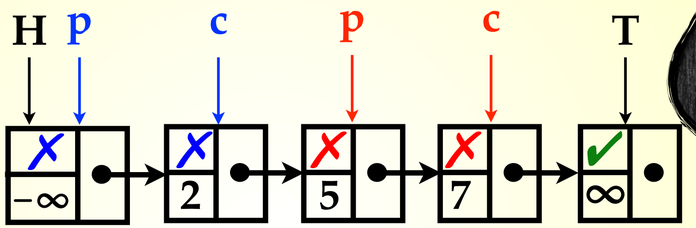
Sets | Fine-Grained Synchronization

```
add(k):
Node p, c
1  lock(H)
2  p = H
3  c = p.next;
4  lock(c);
5  while (c.key < k)
6    unlock(p);
7    p = c;
8    c = c.next;
9    lock(c);
10 if (c.key = k)
11   return false
12 else
13   n = new Node(k,-)
14   n.next = c
15   p.next = n
16   return true
17 unlock(c)
18 unlock(p)
```

add(6)

add(1)

```
add(k):
Node p, c
1  lock(H)
2  p = H
3  c = p.next;
4  lock(c);
5  while (c.key < k)
6    unlock(p);
7    p = c;
8    c = c.next;
9    lock(c);
10 if (c.key = k)
11   return false
12 else
13   n = new Node(k,-)
14   n.next = c
15   p.next = n
16   return true
17 unlock(c)
18 unlock(p)
```



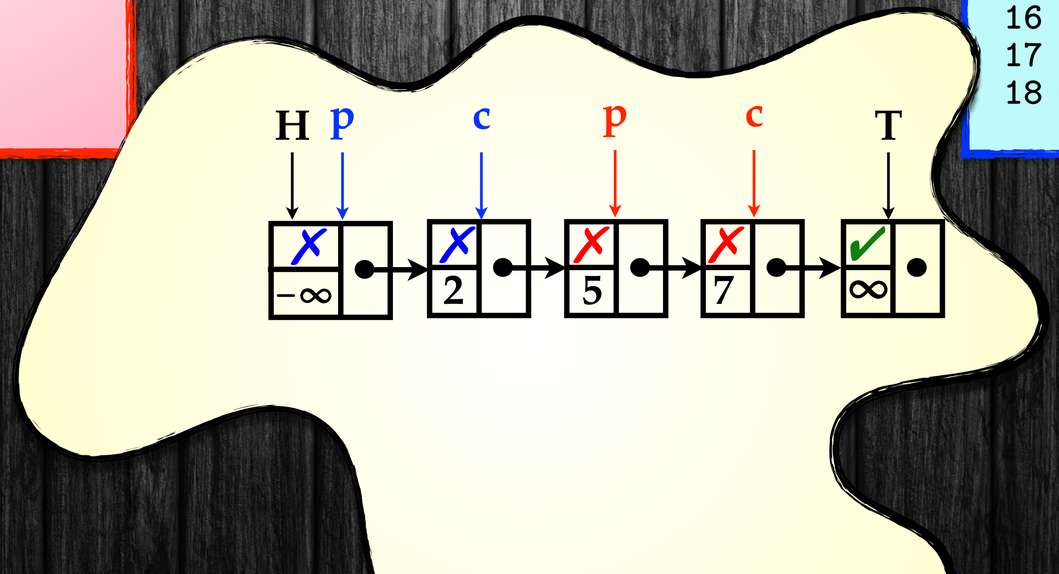
Sets | Fine-Grained Synchronization

```
add(k):
Node p, c
1  lock(H)
2  p = H
3  c = p.next;
4  lock(c);
5  while (c.key < k)
6    unlock(p);
7    p = c;
8    c = c.next;
9    lock(c);
10 if (c.key = k)
11   return false
12 else
13   n = new Node(k,-)
14   n.next = c
15   p.next = n
16   return true
17 unlock(c)
18 unlock(p)
```

add(6)

add(1)

```
add(k):
Node p, c
1  lock(H)
2  p = H
3  c = p.next;
4  lock(c);
5  while (c.key < k)
6    unlock(p);
7    p = c;
8    c = c.next;
9    lock(c);
10 if (c.key = k)
11   return false
12 else
13   n = new Node(k,-)
14   n.next = c
15   p.next = n
16   return true
17 unlock(c)
18 unlock(p)
```



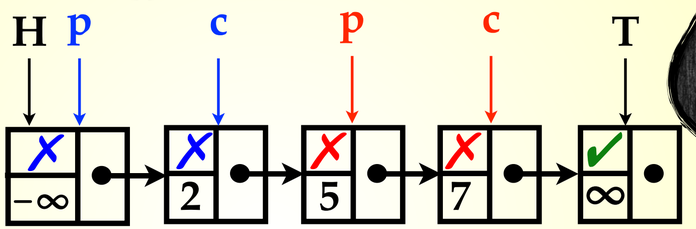
Sets | Fine-Grained Synchronization

```
add(k):
Node p, c
1  lock(H)
2  p = H
3  c = p.next;
4  lock(c);
5  while (c.key < k)
6    unlock(p);
7    p = c;
8    c = c.next;
9    lock(c);
10 if (c.key = k)
11   return false
12 else
13   n = new Node(k,-)
14   n.next = c
15   p.next = n
16   return true
17 unlock(c)
18 unlock(p)
```

add(6)

add(1)

```
add(k):
Node p, c
1  lock(H)
2  p = H
3  c = p.next;
4  lock(c);
5  while (c.key < k)
6    unlock(p);
7    p = c;
8    c = c.next;
9    lock(c);
10 if (c.key = k)
11   return false
12 else
13   n = new Node(k,-)
14   n.next = c
15   p.next = n
16   return true
17 unlock(c)
18 unlock(p)
```



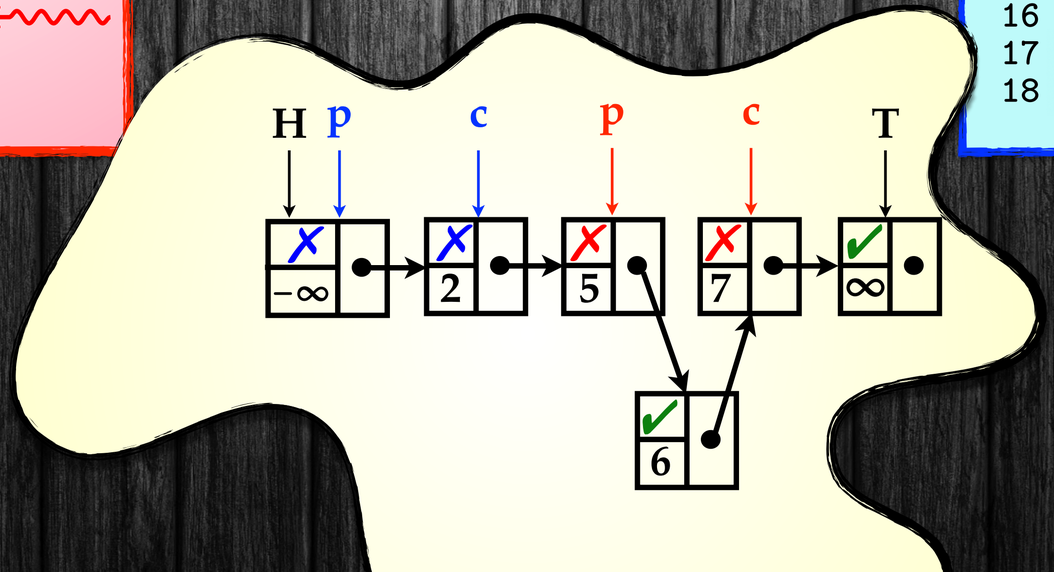
Sets | Fine-Grained Synchronization

```
add(k):
Node p, c
1  lock(H)
2  p = H
3  c = p.next;
4  lock(c);
5  while (c.key < k)
6    unlock(p);
7    p = c;
8    c = c.next;
9    lock(c);
10 if (c.key = k)
11   return false
12 else
13   n = new Node(k,-)
14   n.next = c
15   p.next = n
16   return true
17 unlock(c)
18 unlock(p)
```

add(6)

add(1)

```
add(k):
Node p, c
1  lock(H)
2  p = H
3  c = p.next;
4  lock(c);
5  while (c.key < k)
6    unlock(p);
7    p = c;
8    c = c.next;
9    lock(c);
10 if (c.key = k)
11   return false
12 else
13   n = new Node(k,-)
14   n.next = c
15   p.next = n
16   return true
17 unlock(c)
18 unlock(p)
```



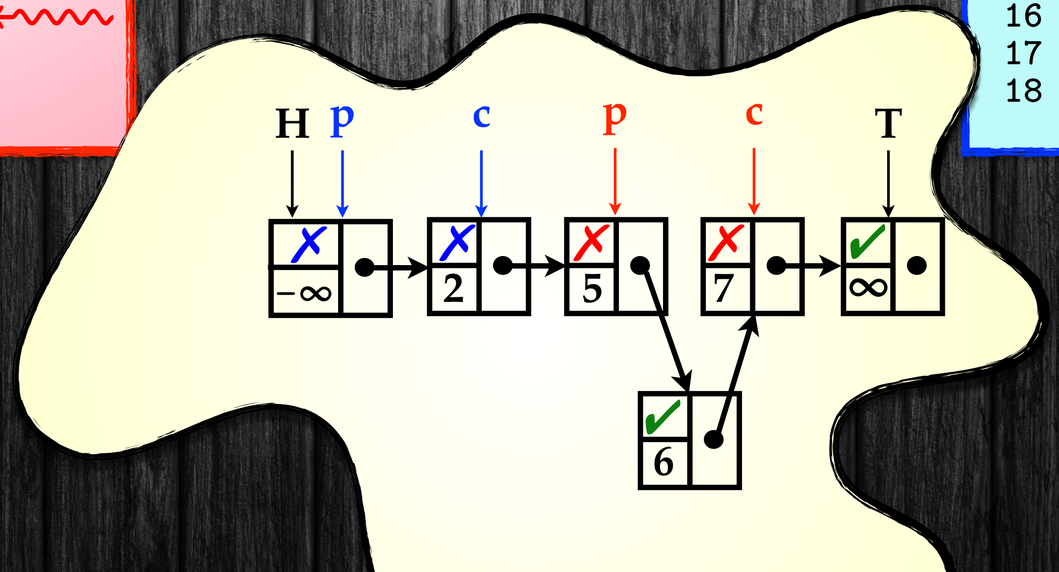
Sets | Fine-Grained Synchronization

```
add(k):
Node p, c
1  lock(H)
2  p = H
3  c = p.next;
4  lock(c);
5  while (c.key < k)
6    unlock(p);
7    p = c;
8    c = c.next;
9    lock(c);
10 if (c.key = k)
11   return false
12 else
13   n = new Node(k,-)
14   n.next = c
15   p.next = n
16   return true
17 unlock(c)
18 unlock(p)
```

add(6)

add(1)

```
add(k):
Node p, c
1  lock(H)
2  p = H
3  c = p.next;
4  lock(c);
5  while (c.key < k)
6    unlock(p);
7    p = c;
8    c = c.next;
9    lock(c);
10 if (c.key = k)
11   return false
12 else
13   n = new Node(k,-)
14   n.next = c
15   p.next = n
16   return true
17 unlock(c)
18 unlock(p)
```



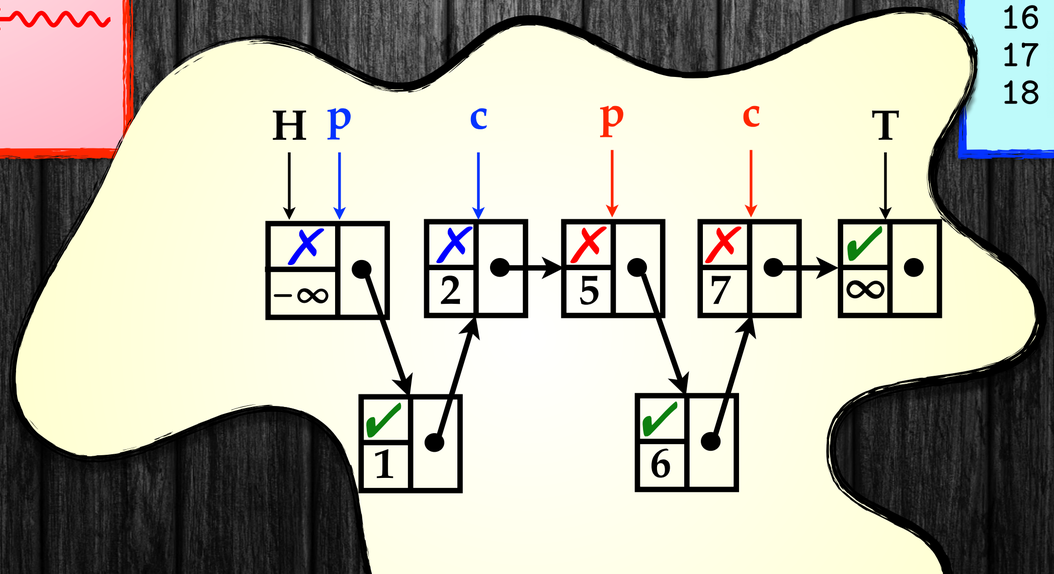
Sets | Fine-Grained Synchronization

```
add(k):
Node p, c
1  lock(H)
2  p = H
3  c = p.next;
4  lock(c);
5  while (c.key < k)
6    unlock(p);
7    p = c;
8    c = c.next;
9    lock(c);
10 if (c.key = k)
11   return false
12 else
13   n = new Node(k,-)
14   n.next = c
15   p.next = n
16   return true
17 unlock(c)
18 unlock(p)
```

add(6)

add(1)

```
add(k):
Node p, c
1  lock(H)
2  p = H
3  c = p.next;
4  lock(c);
5  while (c.key < k)
6    unlock(p);
7    p = c;
8    c = c.next;
9    lock(c);
10 if (c.key = k)
11   return false
12 else
13   n = new Node(k,-)
14   n.next = c
15   p.next = n
16   return true
17 unlock(c)
18 unlock(p)
```



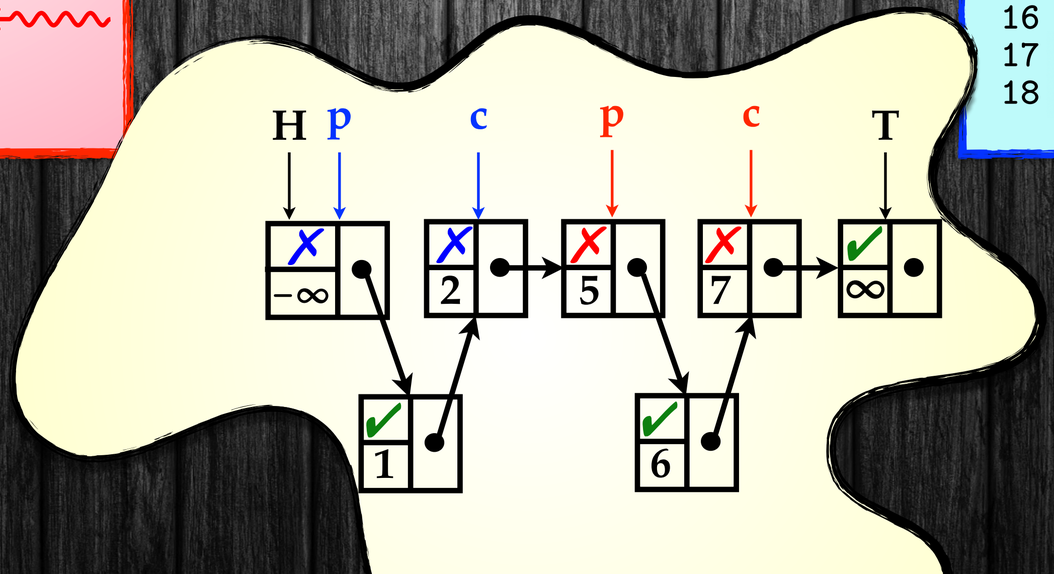
Sets | Fine-Grained Synchronization

```
add(k):
Node p, c
1  lock(H)
2  p = H
3  c = p.next;
4  lock(c);
5  while (c.key < k)
6    unlock(p);
7    p = c;
8    c = c.next;
9    lock(c);
10 if (c.key = k)
11   return false
12 else
13   n = new Node(k,-)
14   n.next = c
15   p.next = n
16   return true
17 unlock(c)
18 unlock(p)
```

add(6)

add(1)

```
add(k):
Node p, c
1  lock(H)
2  p = H
3  c = p.next;
4  lock(c);
5  while (c.key < k)
6    unlock(p);
7    p = c;
8    c = c.next;
9    lock(c);
10 if (c.key = k)
11   return false
12 else
13   n = new Node(k,-)
14   n.next = c
15   p.next = n
16   return true
17 unlock(c)
18 unlock(p)
```



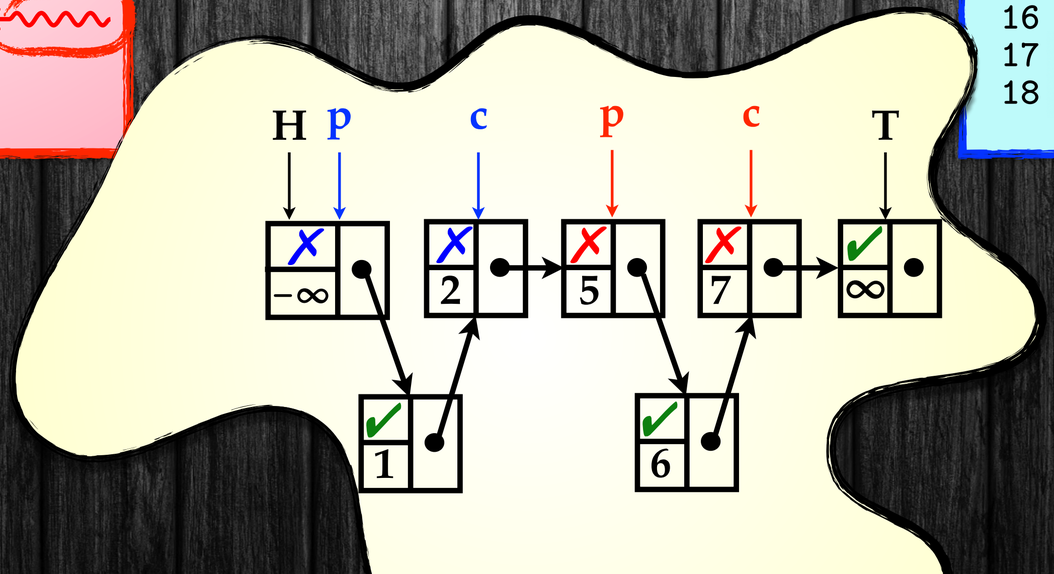
Sets | Fine-Grained Synchronization

```
add(k):
Node p, c
1  lock(H)
2  p = H
3  c = p.next;
4  lock(c);
5  while (c.key < k)
6    unlock(p);
7    p = c;
8    c = c.next;
9    lock(c);
10 if (c.key = k)
11   return false
12 else
13   n = new Node(k,-)
14   n.next = c
15   p.next = n
16   return true
17 unlock(c)
18 unlock(p)
```

add(6)

add(1)

```
add(k):
Node p, c
1  lock(H)
2  p = H
3  c = p.next;
4  lock(c);
5  while (c.key < k)
6    unlock(p);
7    p = c;
8    c = c.next;
9    lock(c);
10 if (c.key = k)
11   return false
12 else
13   n = new Node(k,-)
14   n.next = c
15   p.next = n
16   return true
17 unlock(c)
18 unlock(p)
```



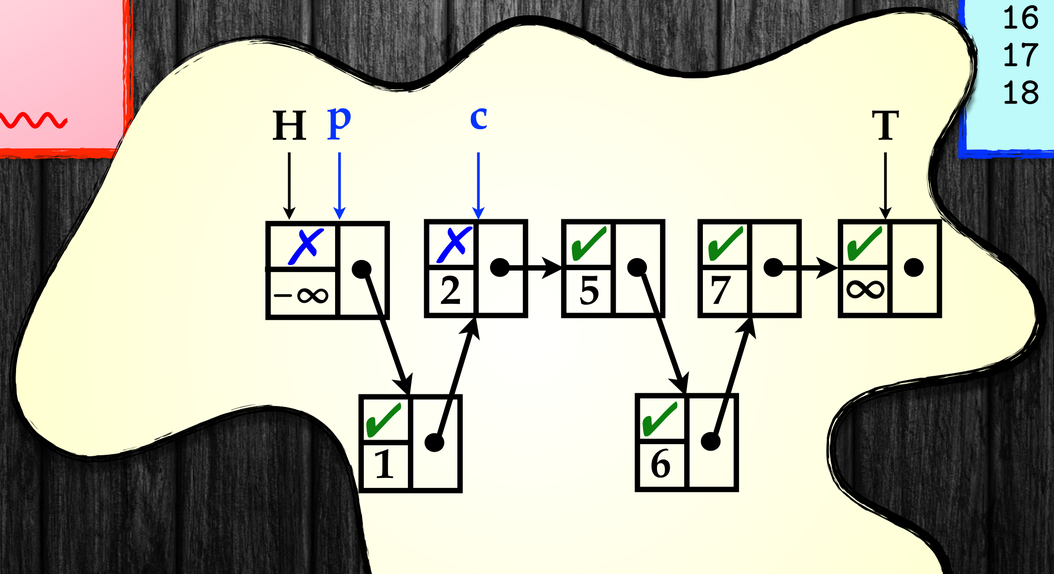
Sets Fine-Grained Synchronization

```
add(k):
Node p, c
1  lock(H)
2  p = H
3  c = p.next;
4  lock(c);
5  while (c.key < k)
6    unlock(p);
7    p = c;
8    c = c.next;
9    lock(c);
10 if (c.key = k)
11   return false
12 else
13   n = new Node(k,-)
14   n.next = c
15   p.next = n
16   return true
17 unlock(c)
18 unlock(p)
```

add(6)

add(1)

```
add(k):
Node p, c
1  lock(H)
2  p = H
3  c = p.next;
4  lock(c);
5  while (c.key < k)
6    unlock(p);
7    p = c;
8    c = c.next;
9    lock(c);
10 if (c.key = k)
11   return false
12 else
13   n = new Node(k,-)
14   n.next = c
15   p.next = n
16   return true
17 unlock(c)
18 unlock(p)
```



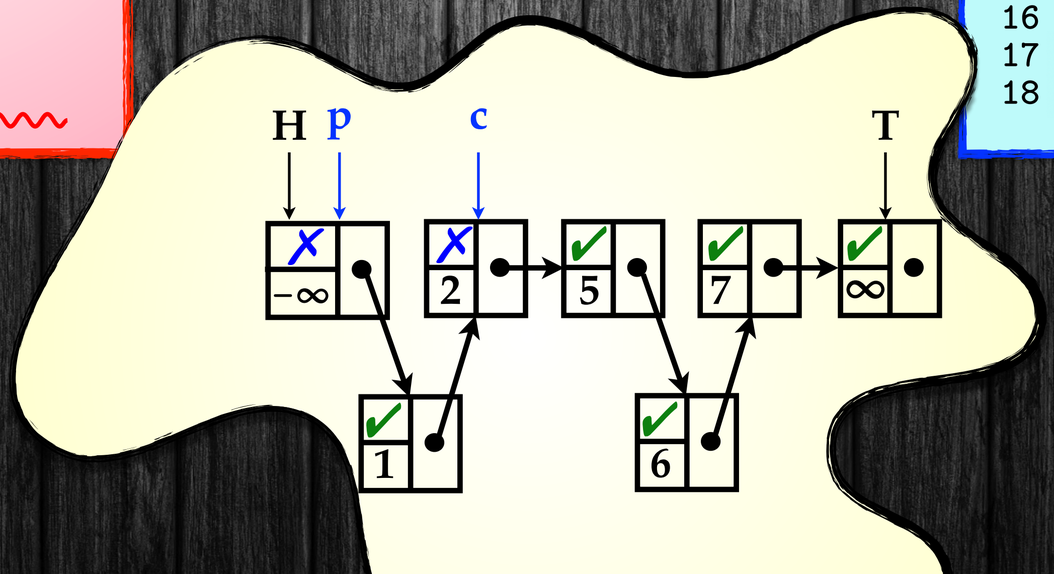
Sets | Fine-Grained Synchronization

```
add(k):
Node p, c
1  lock(H)
2  p = H
3  c = p.next;
4  lock(c);
5  while (c.key < k)
6    unlock(p);
7    p = c;
8    c = c.next;
9    lock(c);
10 if (c.key = k)
11   return false
12 else
13   n = new Node(k,-)
14   n.next = c
15   p.next = n
16   return true
17 unlock(c)
18 unlock(p)
```

add(6)

add(1)

```
add(k):
Node p, c
1  lock(H)
2  p = H
3  c = p.next;
4  lock(c);
5  while (c.key < k)
6    unlock(p);
7    p = c;
8    c = c.next;
9    lock(c);
10 if (c.key = k)
11   return false
12 else
13   n = new Node(k,-)
14   n.next = c
15   p.next = n
16   return true
17 unlock(c)
18 unlock(p)
```



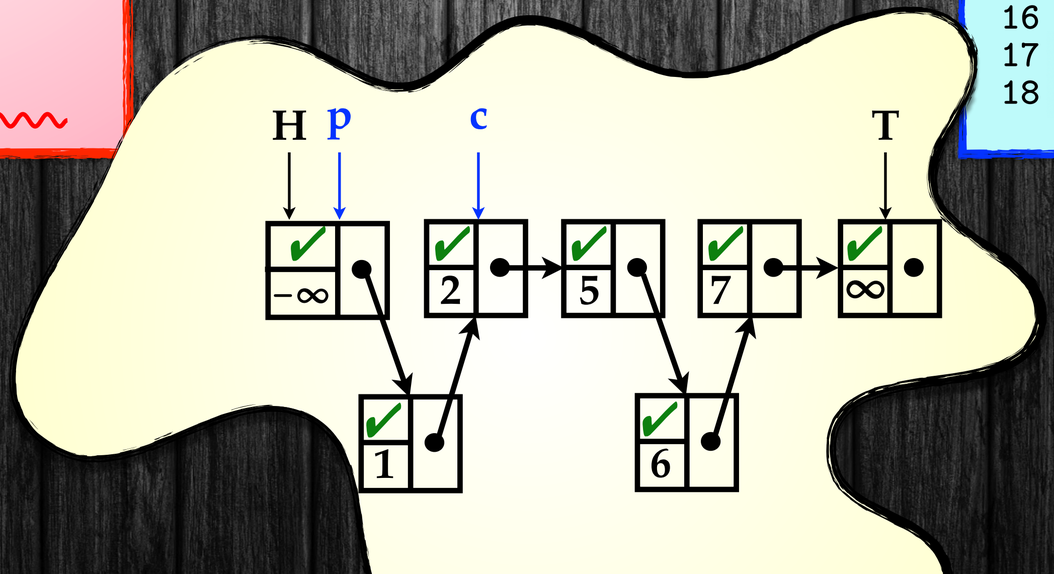
Sets | Fine-Grained Synchronization

```
add(k):
Node p, c
1  lock(H)
2  p = H
3  c = p.next;
4  lock(c);
5  while (c.key < k)
6    unlock(p);
7    p = c;
8    c = c.next;
9    lock(c);
10 if (c.key = k)
11   return false
12 else
13   n = new Node(k,-)
14   n.next = c
15   p.next = n
16   return true
17 unlock(c)
18 unlock(p)
```

add(6)

add(1)

```
add(k):
Node p, c
1  lock(H)
2  p = H
3  c = p.next;
4  lock(c);
5  while (c.key < k)
6    unlock(p);
7    p = c;
8    c = c.next;
9    lock(c);
10 if (c.key = k)
11   return false
12 else
13   n = new Node(k,-)
14   n.next = c
15   p.next = n
16   return true
17 unlock(c)
18 unlock(p)
```

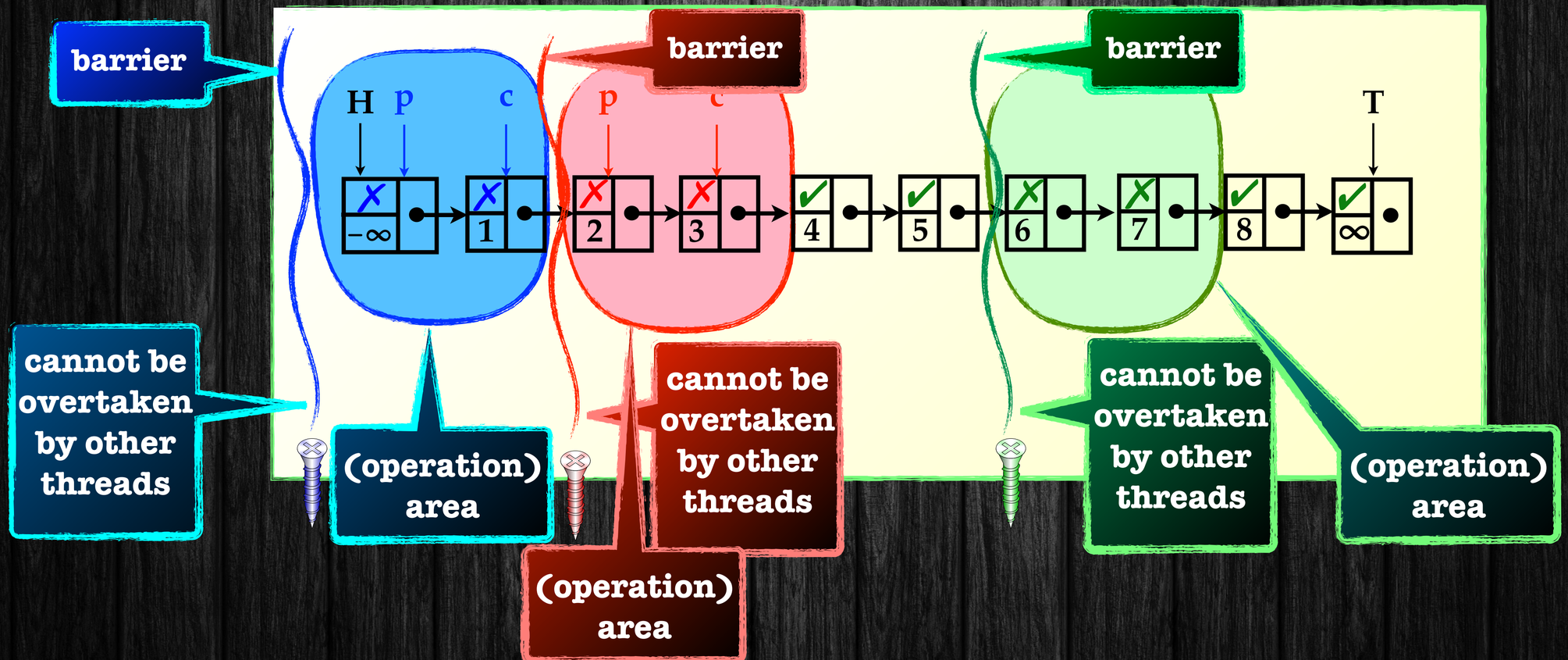


Sets | Fine-Grained Synchronization

threads cannot overtake each other

threads do not interfere with each other

Lock Coupling



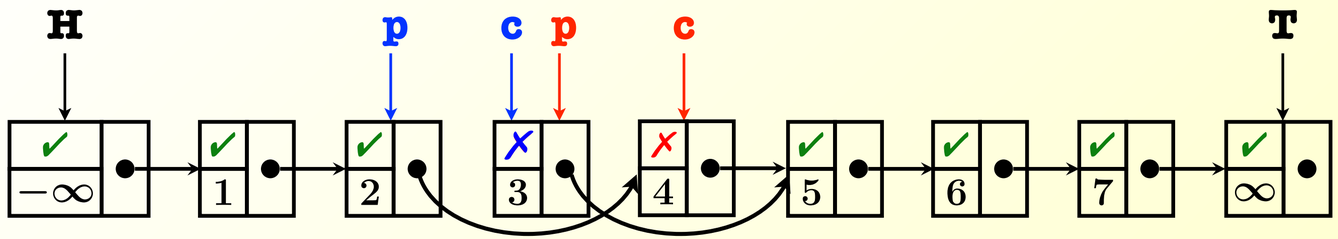
Sets Fine-Grained Synchronization

rmv(4)

~~Lock Coupling~~

Lock only one cell

rmv(3)



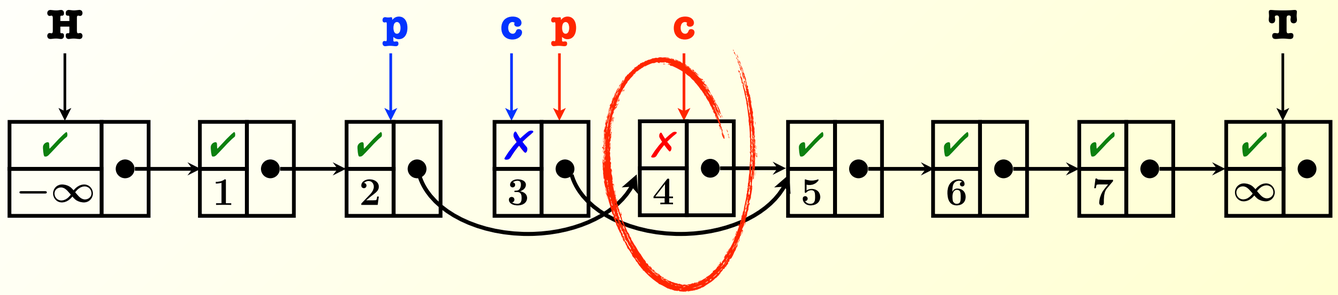
Sets Fine-Grained Synchronization

rmv(4)

~~Lock Coupling~~

Lock only one cell

rmv(3)



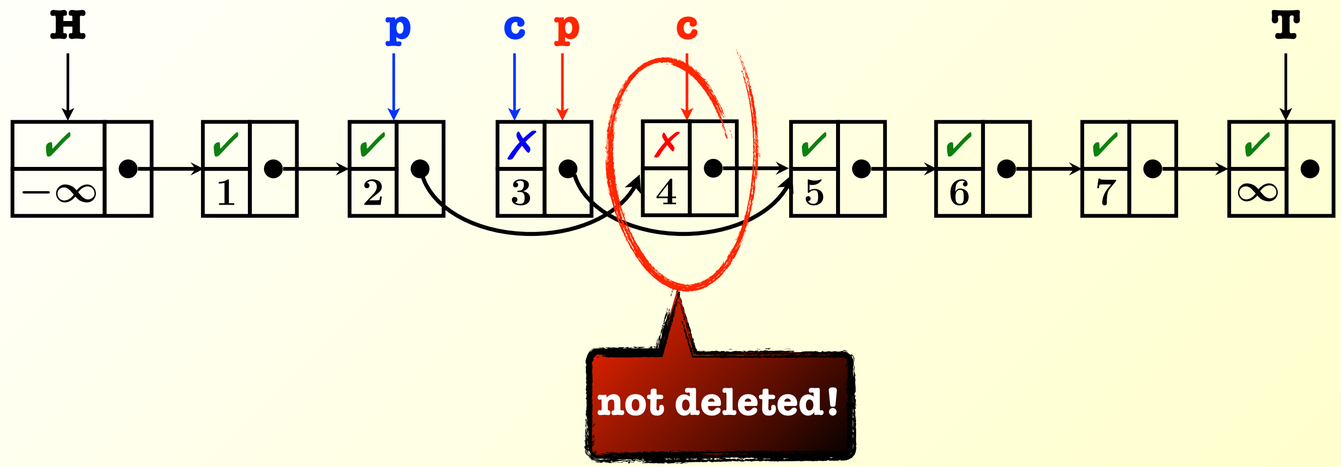
Sets | Fine-Grained Synchronization

rmv(4)

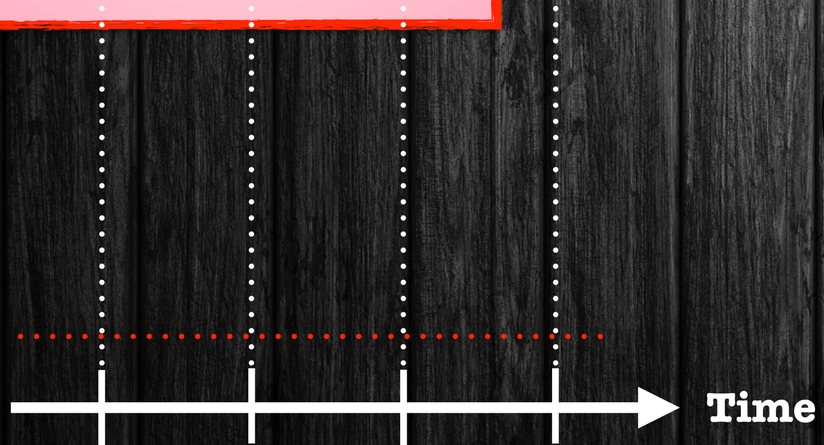
~~Lock Coupling~~

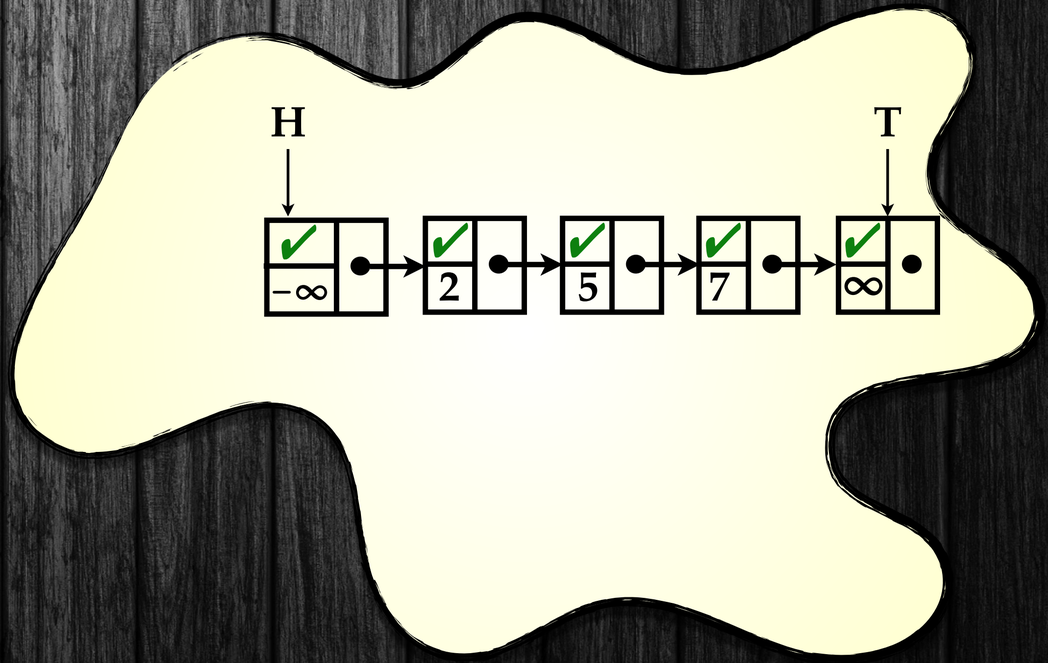
Lock only one cell

rmv(3)



```
add(k):
Node p, c
1  lock(H)
2  p = H
3  c = p.next;
4  lock(c);
5  while (c.key < k)
6    unlock(p);
7    p = c;
8    c = c.next;
9    lock(c);
10 if (c.key = k)
11   return false
12 else
13   n = new Node(k,-)
14   n.next = c
15   p.next = n
16   return true
17   unlock(c)
18   unlock(p)
```



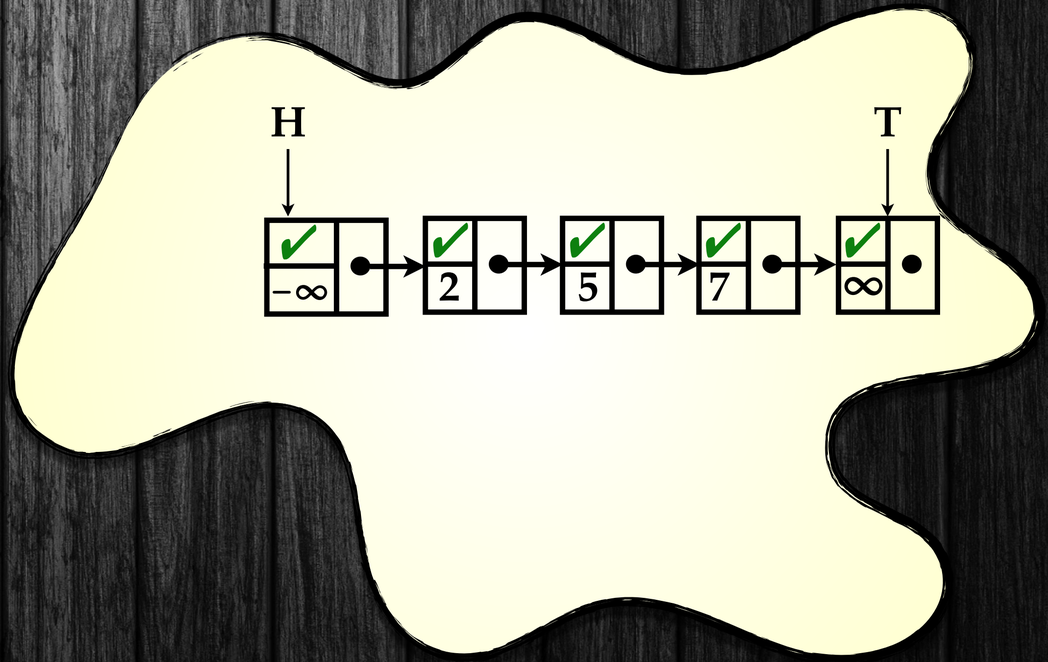


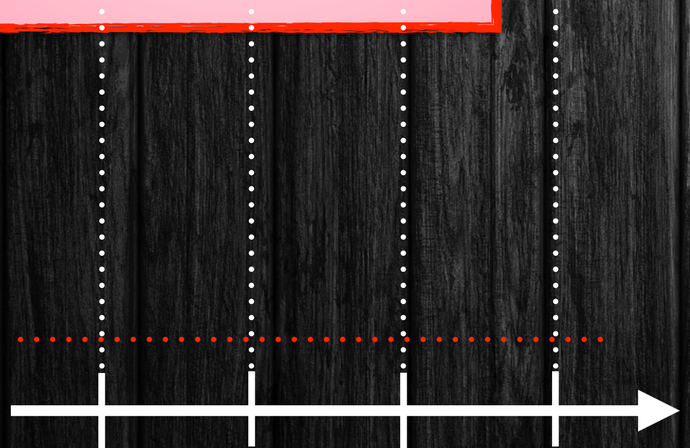
Sets | Fine-Grained Synchronization | Linearization Policy

```
add(k):
Node p, c
1  lock(H)
2  p = H
3  c = p.next;
4  lock(c);
5  while (c.key < k)
6    unlock(p);
7    p = c;
8    c = c.next;
9    lock(c);
10 if (c.key = k)
11   return false
12 else
13   n = new Node(k,-)
14   n.next = c
15   p.next = n
16   return true
17   unlock(c)
18   unlock(p)
```

LP

"time point at which I find a key $\geq$ input key"





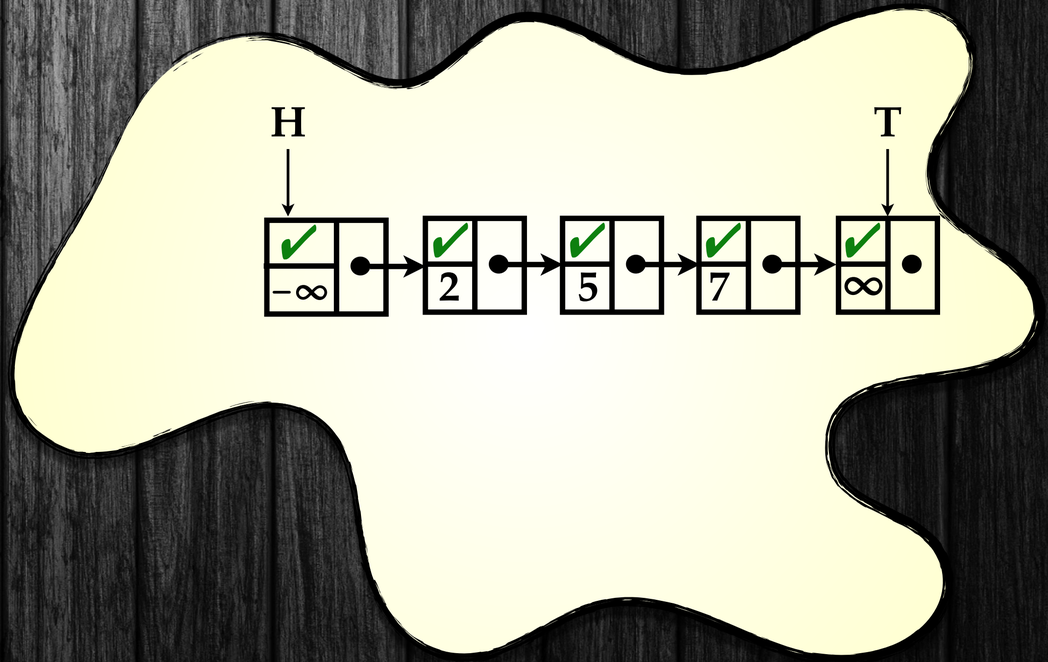
Sets | Fine-Grained Synchronization | Linearization Policy

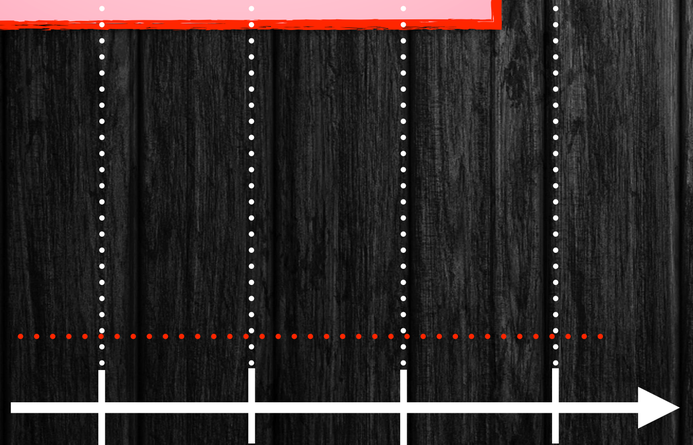
```
add(k):
Node p, c
1  lock(H)
2  p = H
3  c = p.next;
4  lock(c);
5  while (c.key < k)
6    unlock(p);
7    p = c;
8    c = c.next;
9    lock(c);
10 if (c.key = k)
11   return false
12 else
13   n = new Node(k,-)
14   n.next = c
15   p.next = n
16   return true
17   unlock(c)
18   unlock(p)
```

LP

"time point at which I find a key ≥ input key"

"time point at which I lock the cell with key ≥ input key"





Sets | Fine-Grained Synchronization | Linearization Policy

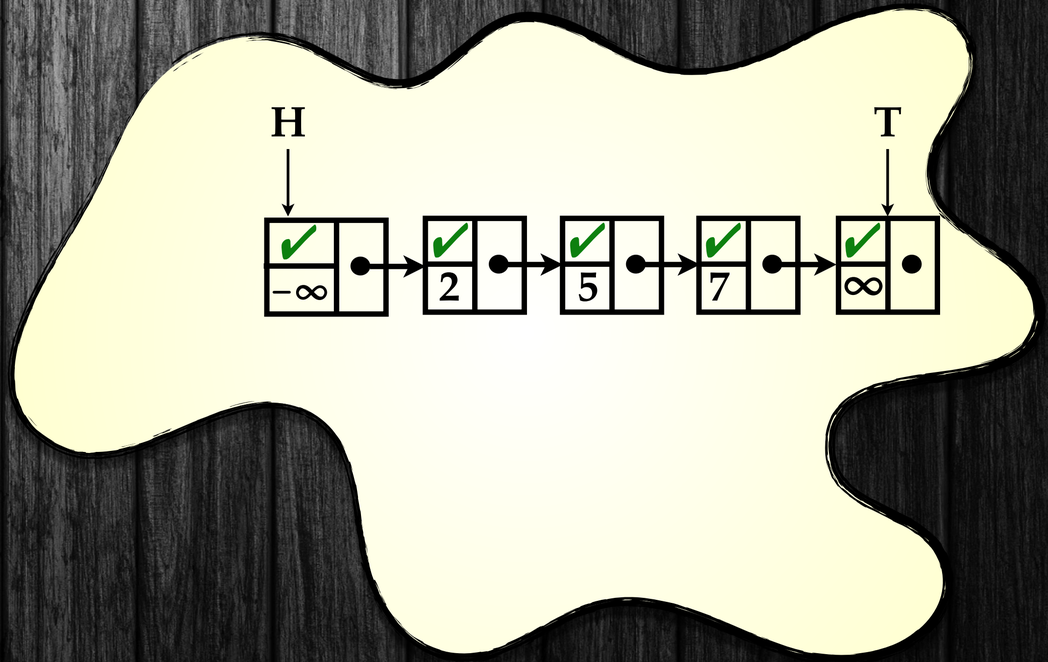
```
add(k):
Node p, c
1  lock(H)
2  p = H
3  c = p.next;
4  lock(c);
5  while (c.key < k)
6    unlock(p);
7    p = c;
8    c = c.next;
9    lock(c);
10 if (c.key = k)
11   return false
12 else
13   n = new Node(k,-)
14   n.next = c
15   p.next = n
16   return true
17   unlock(c)
18   unlock(p)
```

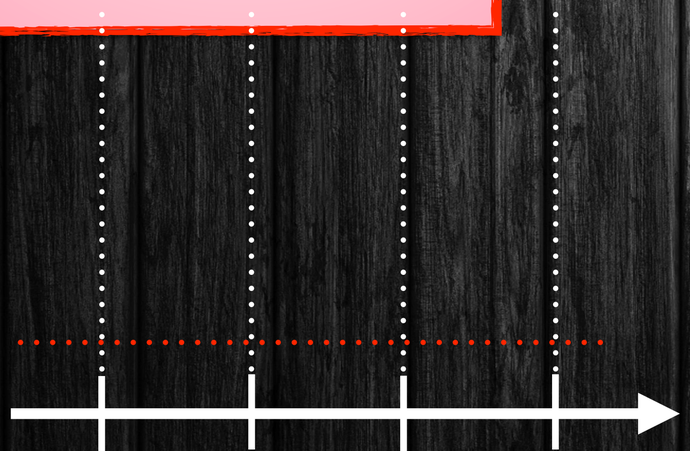
LP

"time point at which I find a key ≥ input key"

"time point at which I lock the cell with key ≥ input key"

add(4)





Sets | Fine-Grained Synchronization | Linearization Policy

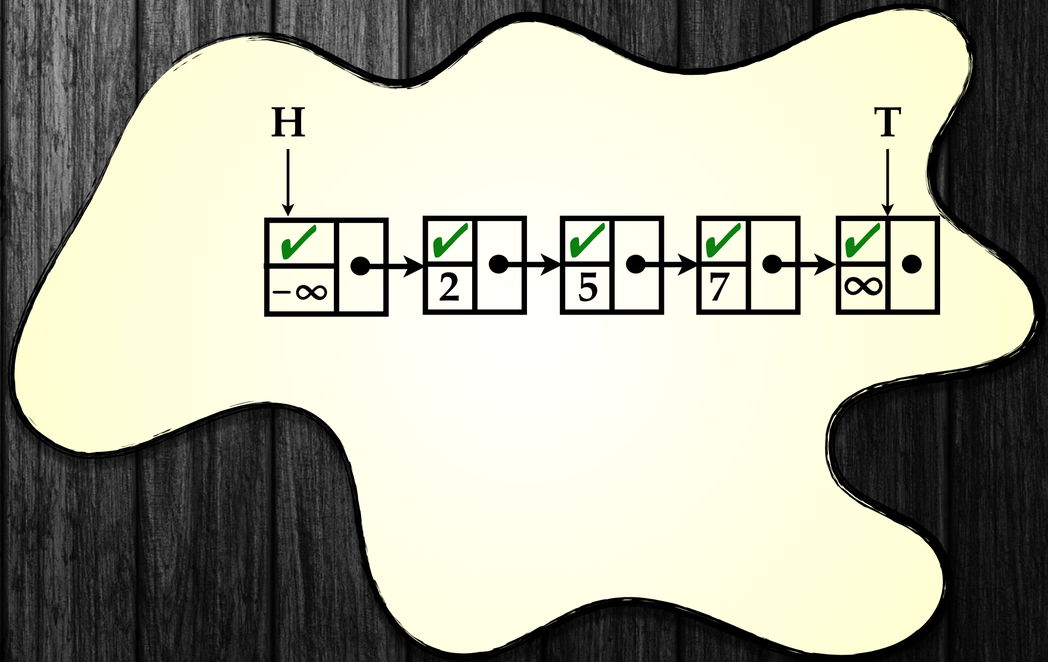
```
add(k):
Node p, c
1  lock(H)
2  p = H
3  c = p.next;
4  lock(c);
5  while (c.key < k)
6    unlock(p);
7    p = c;
8    c = c.next;
9    lock(c);
10 if (c.key = k)
11   return false
12 else
13   n = new Node(k,-)
14   n.next = c
15   p.next = n
16   return true
17   unlock(c)
18   unlock(p)
```

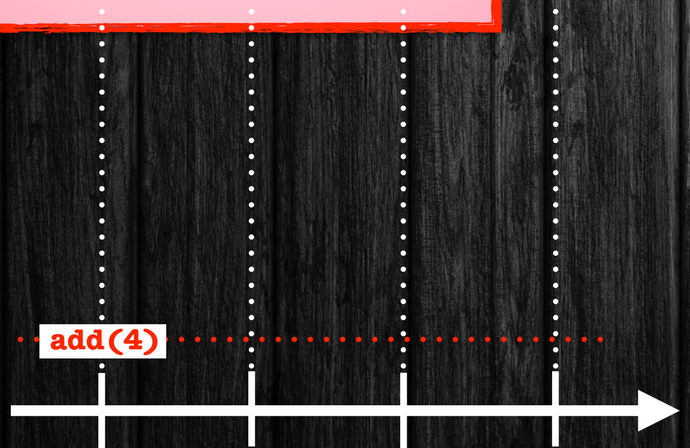
LP

"time point at which I find a key ≥ input key"

"time point at which I lock the cell with key ≥ input key"

add(4)





Sets | Fine-Grained Synchronization | Linearization Policy

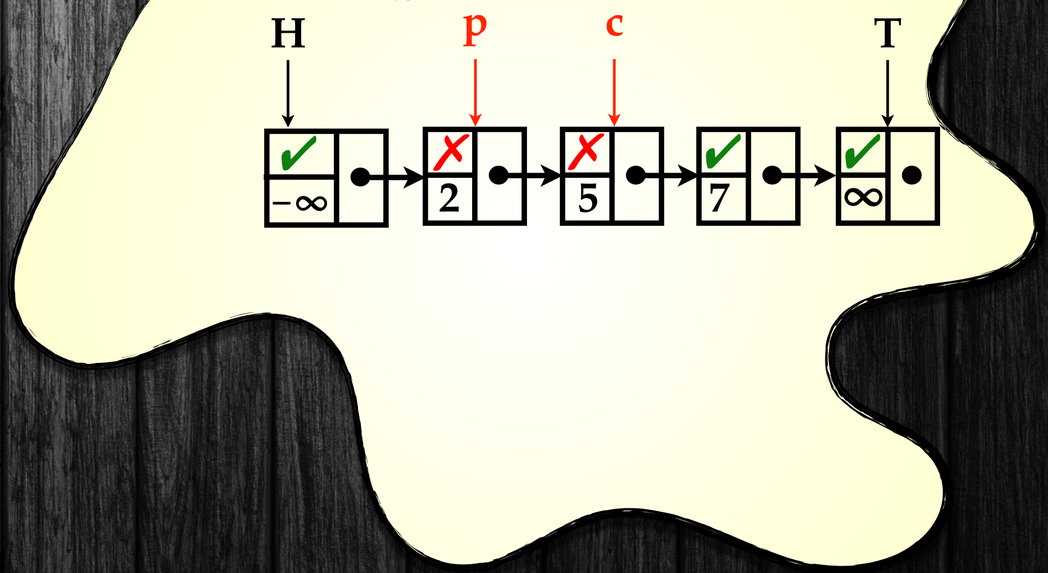
```
add(k):
Node p, c
 1  lock(H)
 2  p = H
 3  c = p.next;
 4  lock(c);
 5  while (c.key < k)
 6    unlock(p);
 7    p = c;
 8    c = c.next;
 9    lock(c);
10  if (c.key = k)
11    return false
12  else
13    n = new Node(k,-)
14    n.next = c
15    p.next = n
16    return true
17    unlock(c)
18    unlock(p)
```

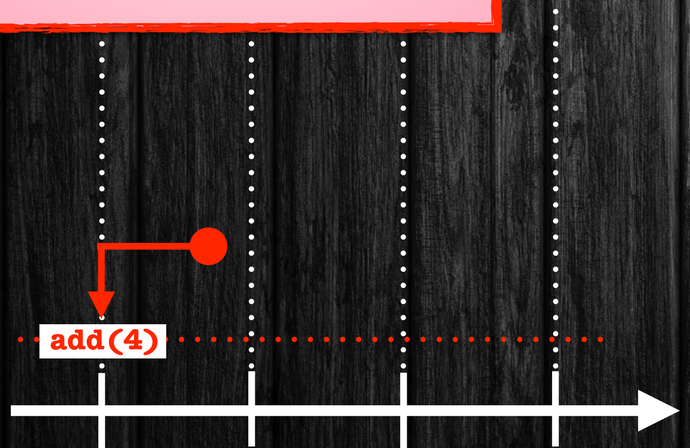


LP

"time point at which I find a key ≥ input key"

add(4)





Sets | Fine-Grained Synchronization | Linearization Policy

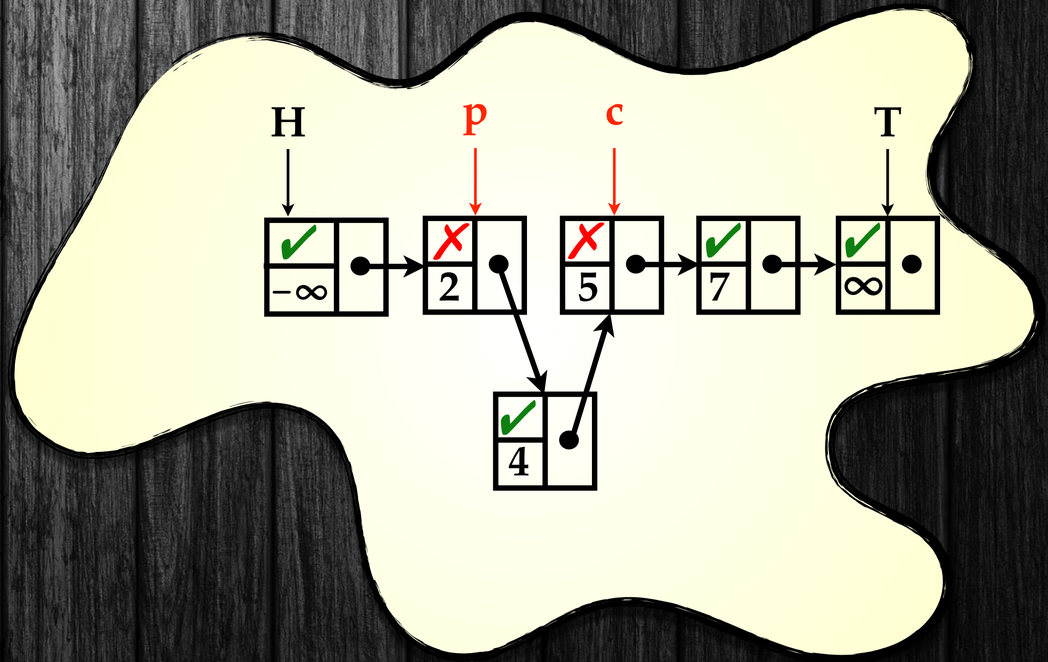
```
add(k):
Node p, c
 1  lock(H)
 2  p = H
 3  c = p.next;
 4  lock(c);
 5  while (c.key < k)
 6    unlock(p);
 7    p = c;
 8    c = c.next;
 9    lock(c);
10  if (c.key = k)
11    return false
12  else
13    n = new Node(k,-)
14    n.next = c
15    p.next = n
16    return true
17    unlock(c)
18    unlock(p)
```

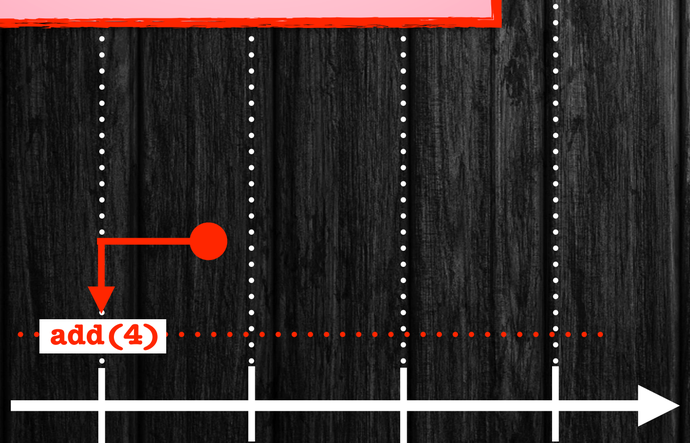


add(4)

LP

"time point at which I find a key ≥ input key"





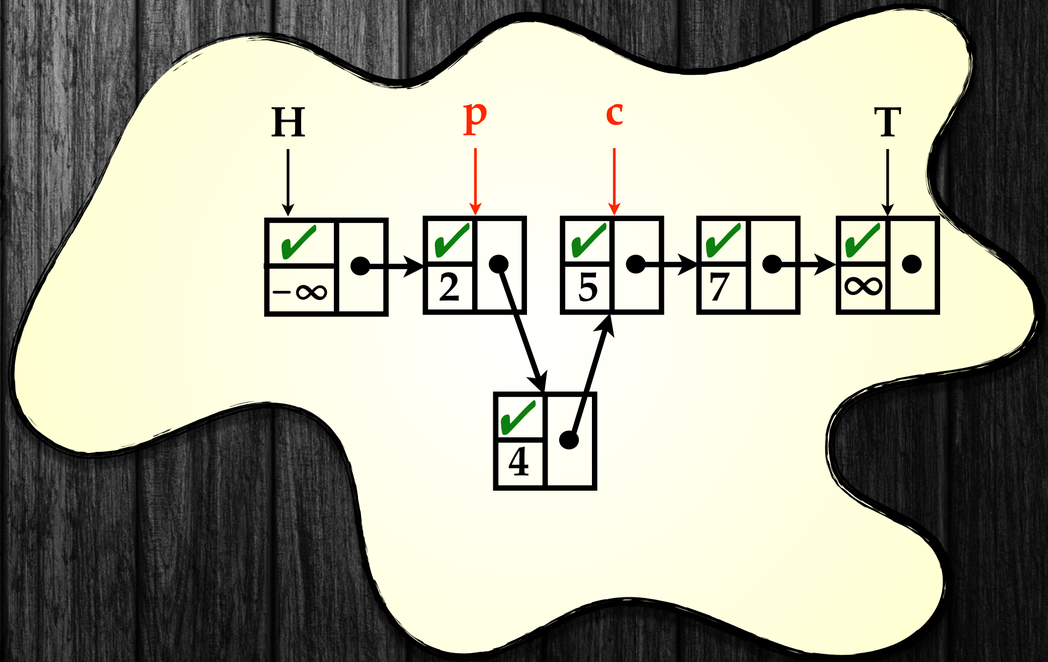
Sets | Fine-Grained Synchronization | Linearization Policy

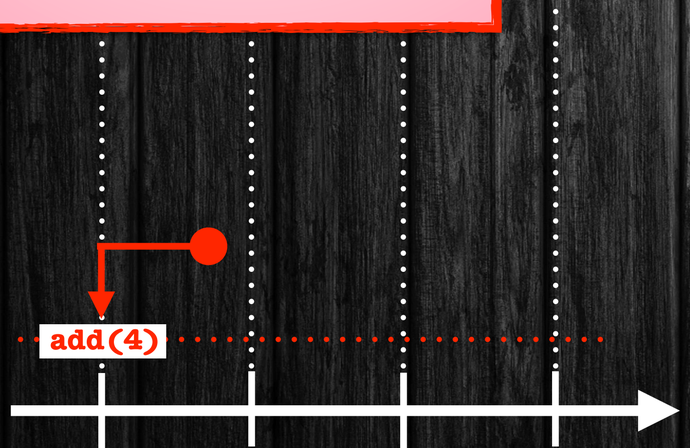
```
add(k):
Node p, c
1  lock(H)
2  p = H
3  c = p.next;
4  lock(c);
5  while (c.key < k)
6    unlock(p);
7    p = c;
8    c = c.next;
9    lock(c);
10 if (c.key = k)
11   return false
12 else
13   n = new Node(k,-)
14   n.next = c
15   p.next = n
16   return true
17   unlock(c)
18   unlock(p)
```

LP

"time point at which I find a key ≥ input key"

add(4)





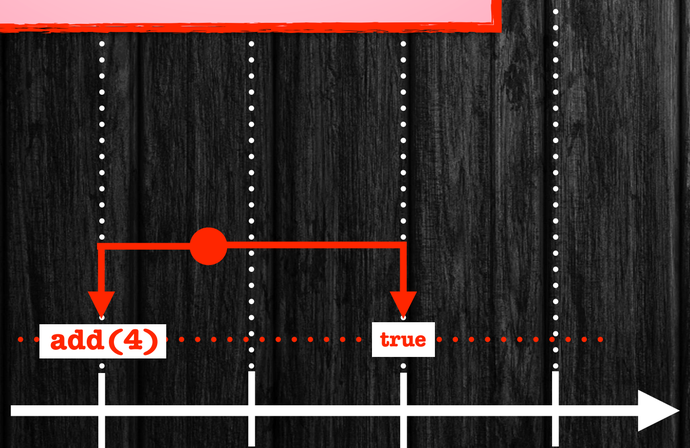
Sets | Fine-Grained Synchronization | Linearization Policy

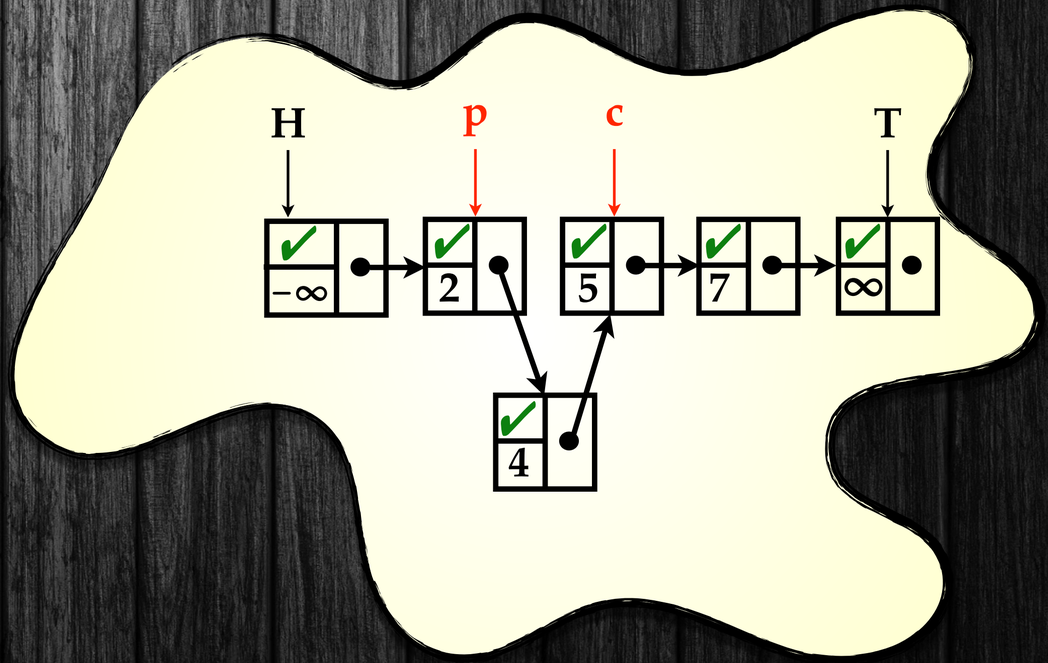
```
add(k):
Node p, c
1  lock(H)
2  p = H
3  c = p.next;
4  lock(c);
5  while (c.key < k)
6    unlock(p);
7    p = c;
8    c = c.next;
9    lock(c);
10 if (c.key = k)
11   return false
12 else
13   n = new Node(k,-)
14   n.next = c
15   p.next = n
16   return true
17   unlock(c)
18   unlock(p)
```

LP

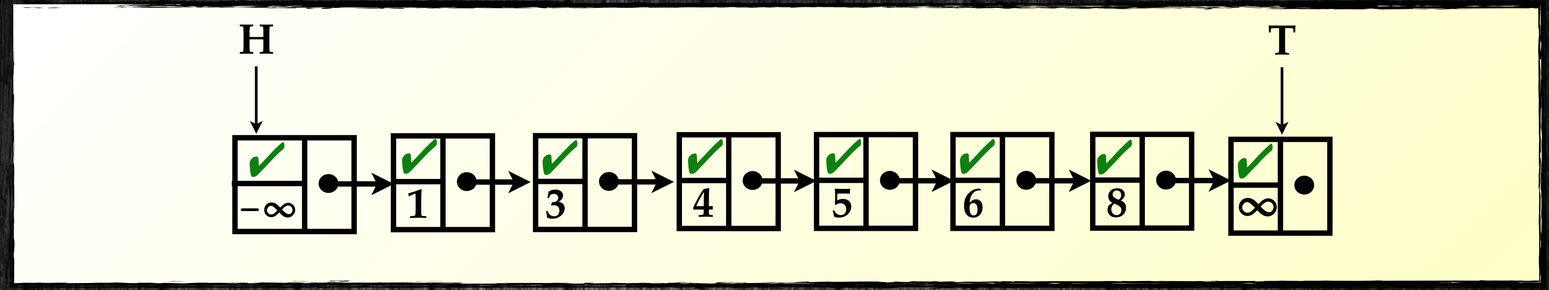
"time point at which I find a key ≥ input key"

add(4)

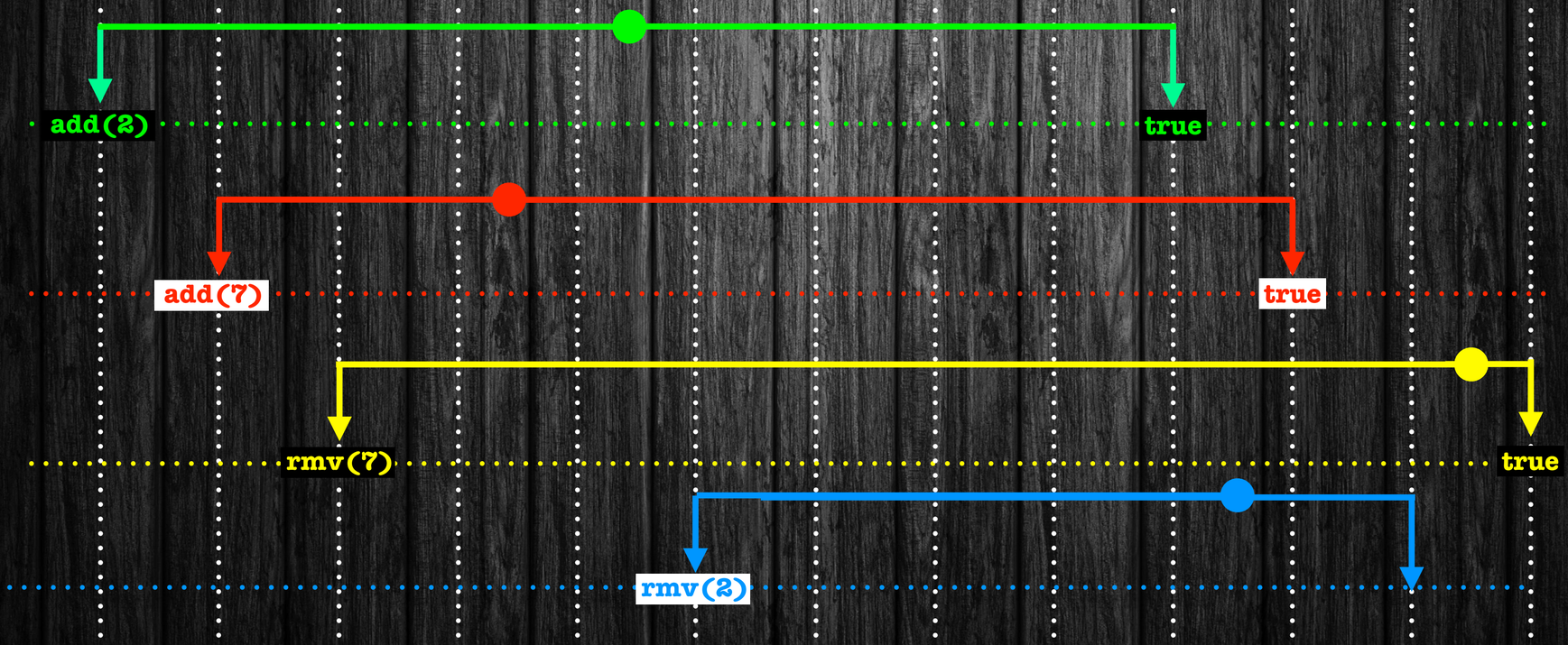




Sets | Fine-Grained Synchronization | Linearization Policy







Outline

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

Outline

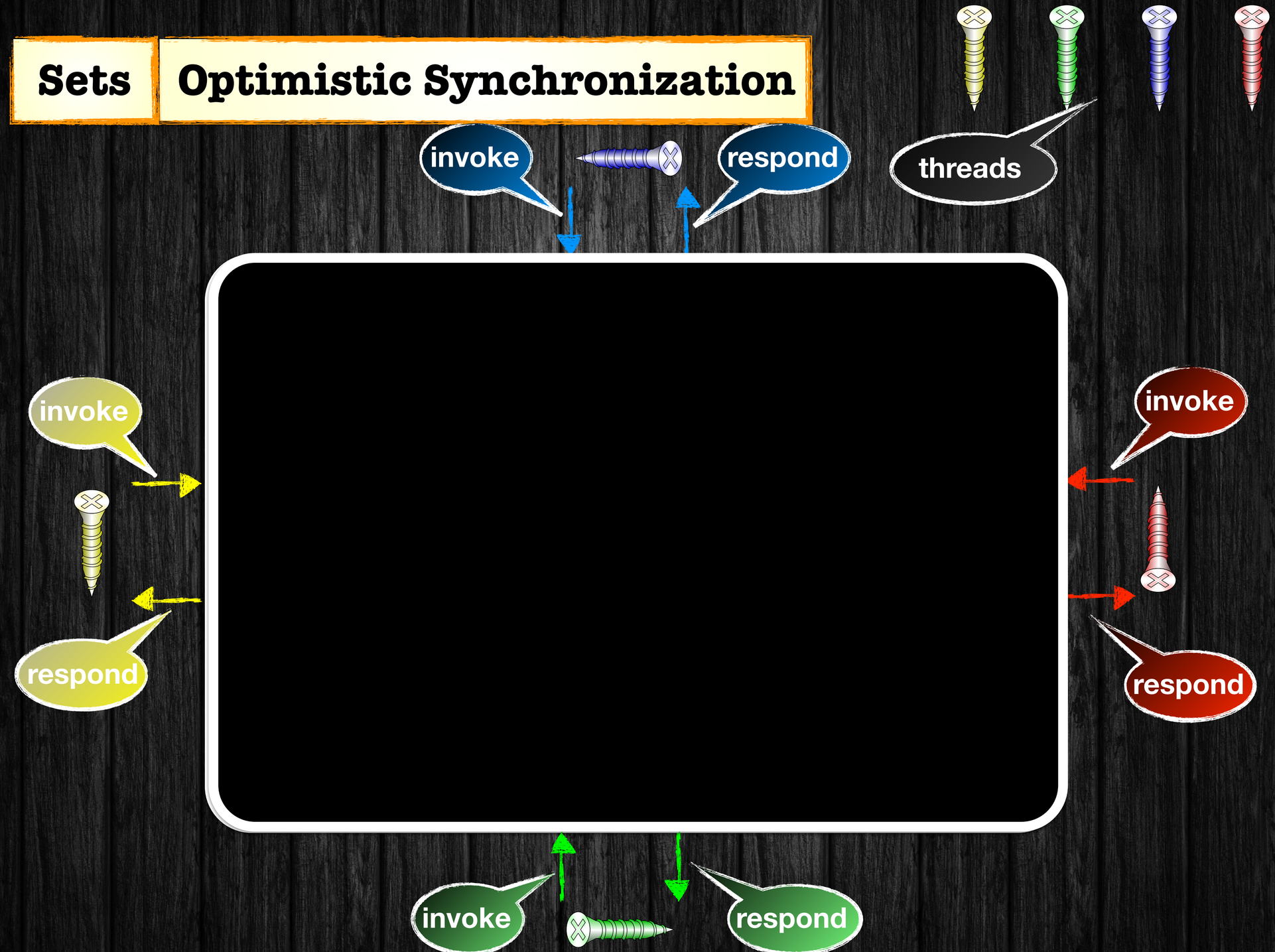
- Overview
- Abstract Data Types
- Sequential Programs
- Concurrent Programs
- Coarse-Grained Synchronization
- Fine-Grained Synchronization
- Optimistic Synchronization
- Lazy Synchronization
- Treiber Stack Algorithm
- Michael&Scott Queue Algorithm

Overview

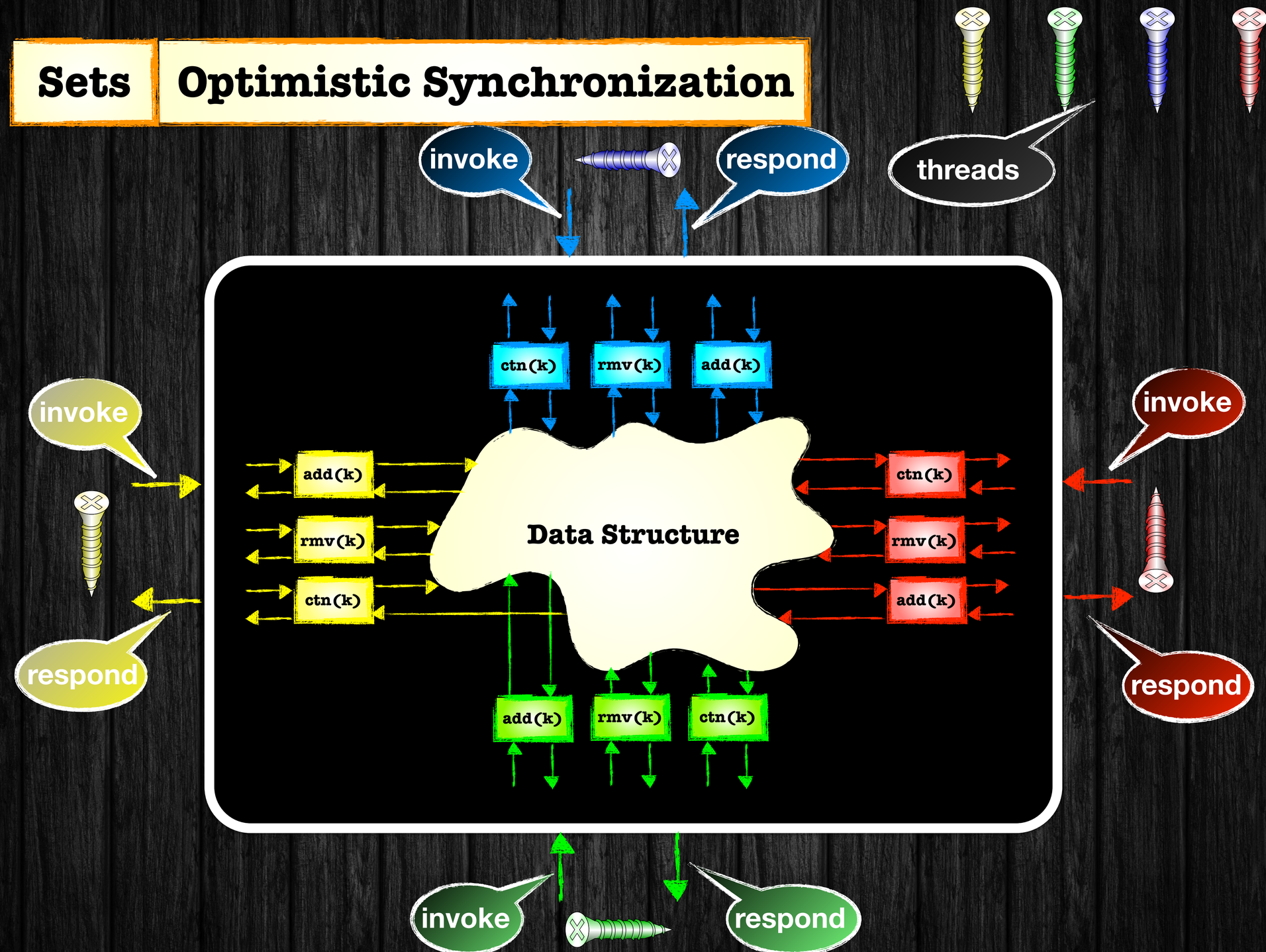
- Sequential Programs
- Coarse-Grained Synchronization
  - Single global lock
- Fine-Grained Synchronization
  - lock per node
  - lock each visited node
- Optimistic Synchronization
  - lock when correct node found
  - restart search on failure
- Lazy Synchronization
  - make ctn wait-free



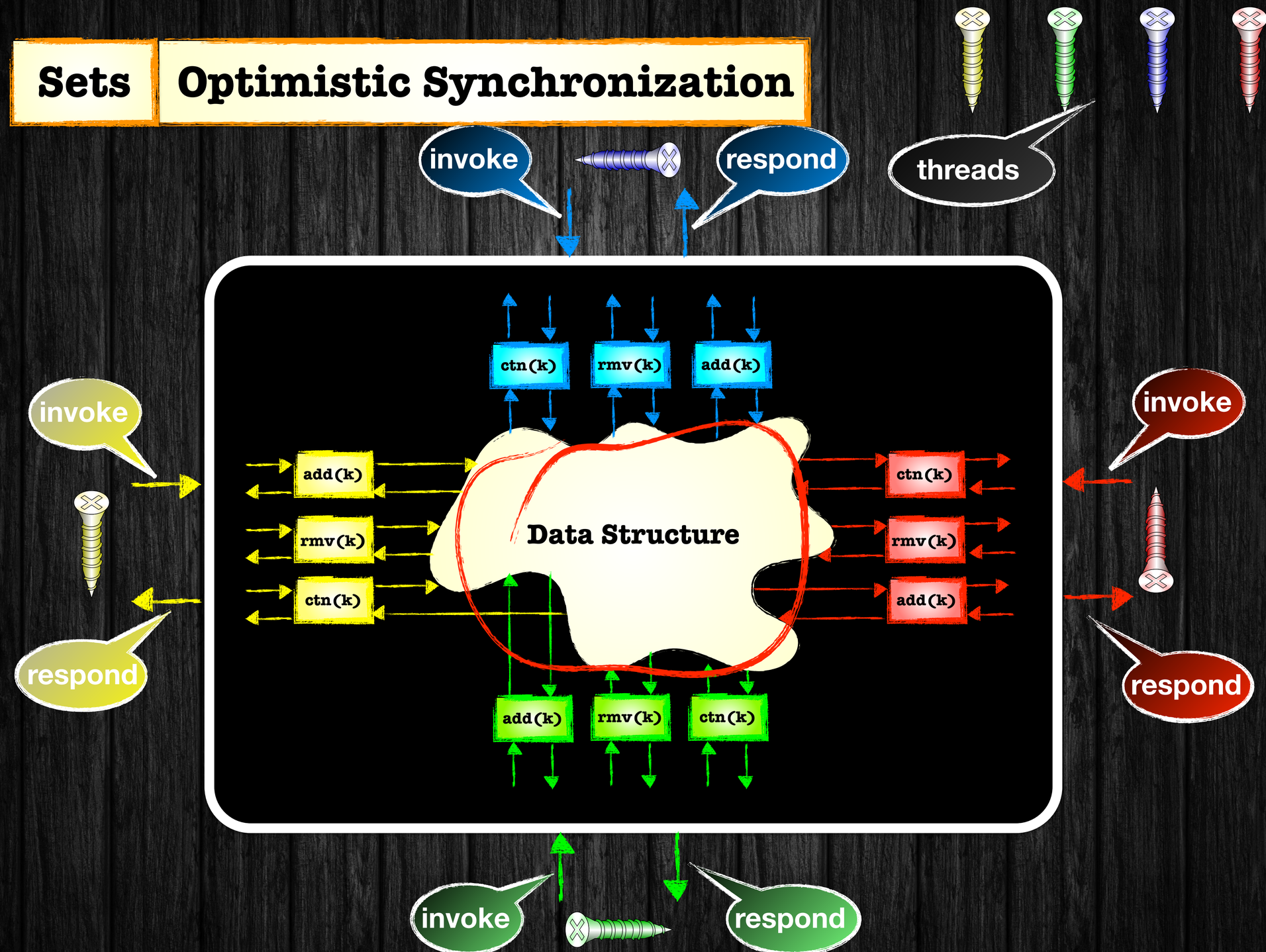
Sets Optimistic Synchronization



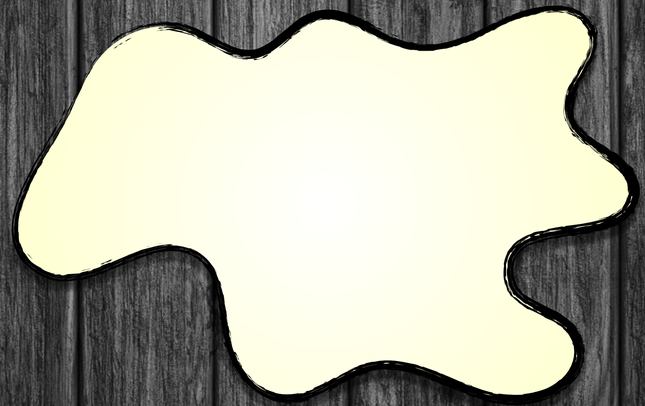
Sets Optimistic Synchronization



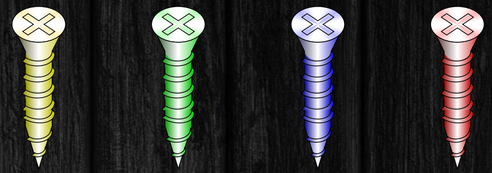
Sets Optimistic Synchronization

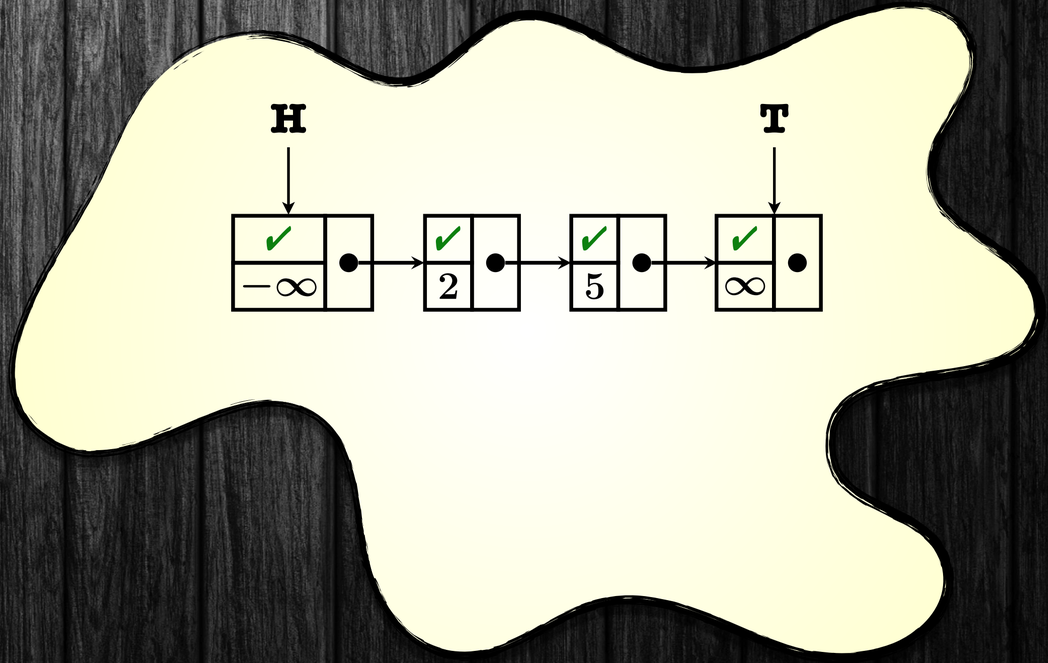


Sets | Optimistic Synchronization



Sets | Optimistic Synchronization

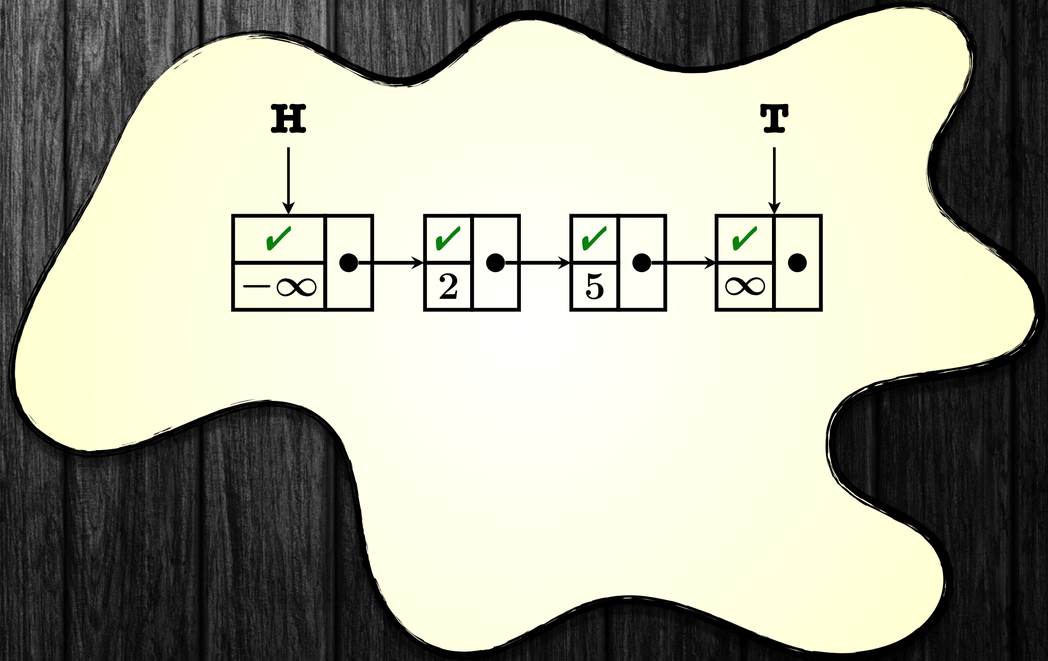




Sets

Optimistic Synchronization

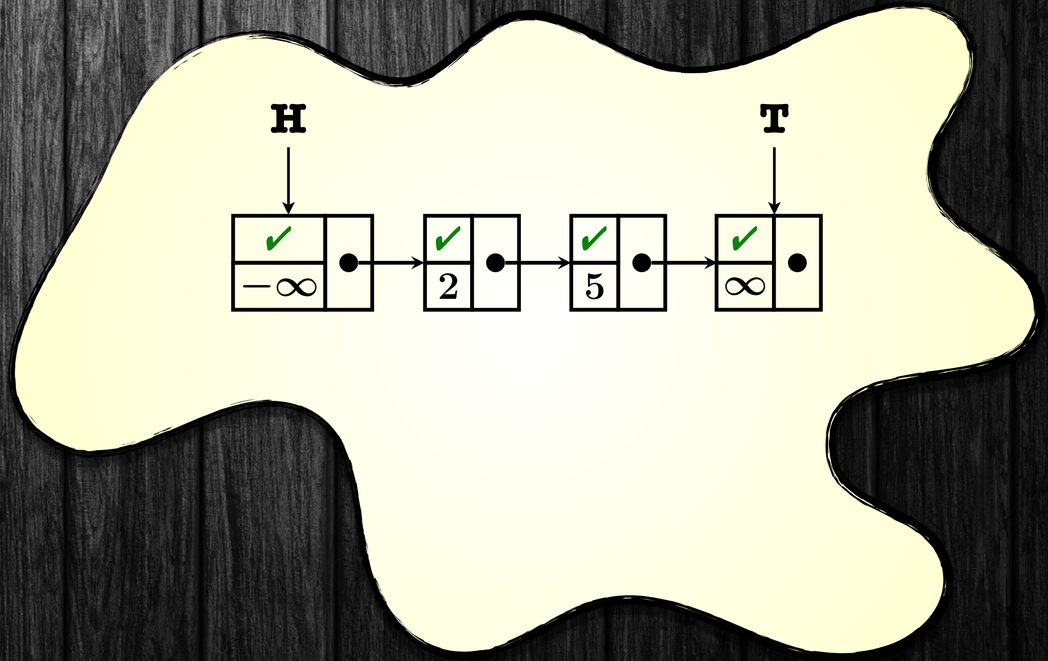
```
add(k):
Node p, c
 1  while (true)
 2    p = H
 3    c = p.next
 4    while (c.key <= k)
 5      p = c
 6      c = c.next
 7    lock(p)
 8    lock(c)
 9    if (validate (p,c))
10      if (c.key=k)
11        return false
12      else
13        n = new Node(k,true,-)
14        n.next = c
15        p.next = n
16        return true
17      unlock p
18      unlock c
19      exit
20    else
21      unlock p
22      unlock c
```



Sets

Optimistic Synchronization

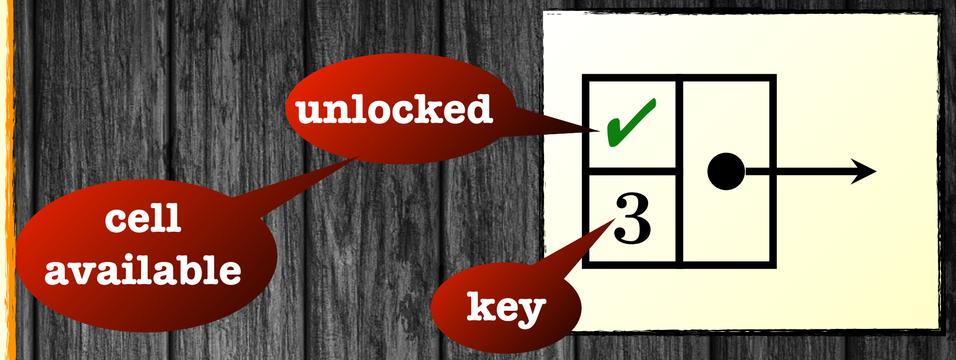
```
add(k):
Node p, c
 1  while (true)
 2    p = H
 3    c = p.next
 4    while (c.key <= k)
 5      p = c
 6      c = c.next
 7    lock(p)
 8    lock(c)
 9    if (validate (p,c))
10      if (c.key=k)
11        return false
12      else
13        n = new Node(k,true,-)
14        n.next = c
15        p.next = n
16        return true
17      unlock p
18      unlock c
19      exit
20    else
21      unlock p
22      unlock c
```

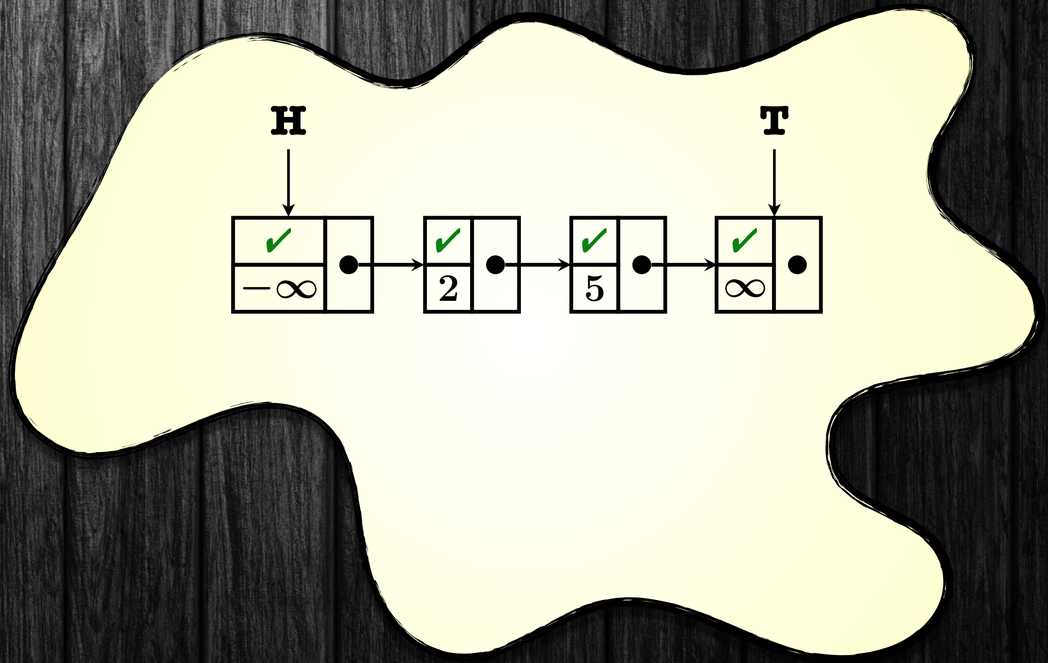


Sets

Optimistic Synchronization

```
add(k):
Node p, c
1  while (true)
2    p = H
3    c = p.next
4    while (c.key <= k)
5      p = c
6      c = c.next
7    lock(p)
8    lock(c)
9    if (validate (p,c))
10     if (c.key=k)
11       return false
12     else
13       n = new Node(k,true,-)
14       n.next = c
15       p.next = n
16       return true
17     unlock p
18     unlock c
19     exit
20   else
21     unlock p
22     unlock c
```

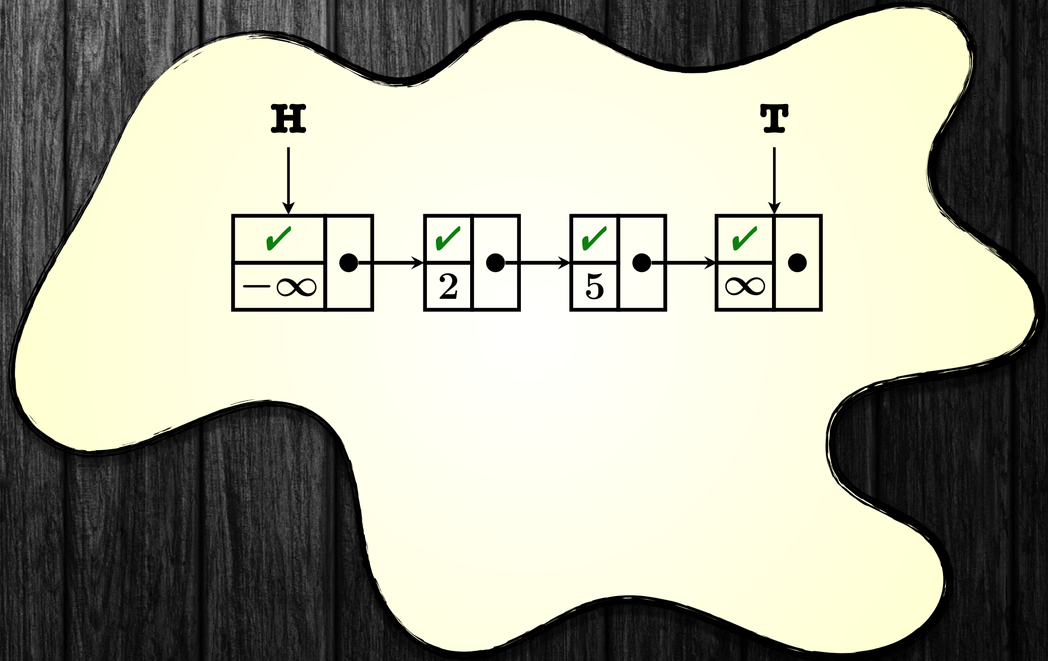




Sets

Optimistic Synchronization

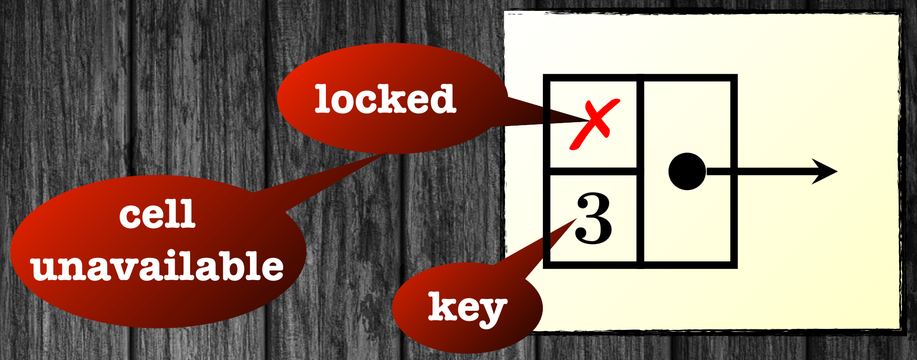
```
add(k):
Node p, c
 1  while (true)
 2    p = H
 3    c = p.next
 4    while (c.key <= k)
 5      p = c
 6      c = c.next
 7    lock(p)
 8    lock(c)
 9    if (validate (p,c))
10      if (c.key=k)
11        return false
12      else
13        n = new Node(k,true,-)
14        n.next = c
15        p.next = n
16        return true
17      unlock p
18      unlock c
19      exit
20    else
21      unlock p
22      unlock c
```

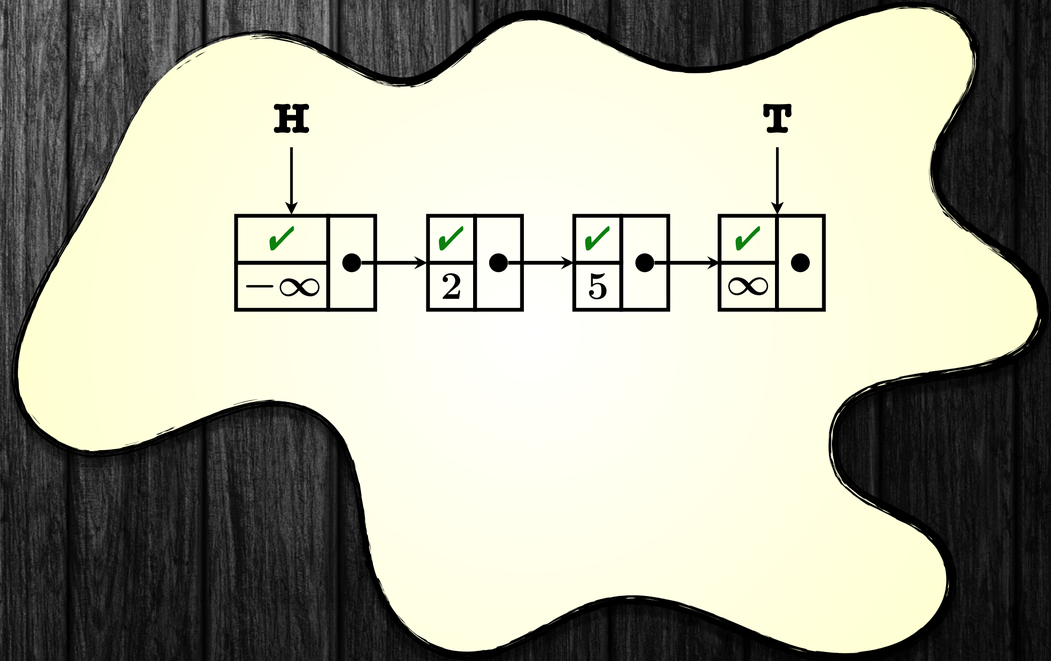


Sets

Optimistic Synchronization

```
add(k):
Node p, c
1  while (true)
2    p = H
3    c = p.next
4    while (c.key <= k)
5      p = c
6      c = c.next
7    lock(p)
8    lock(c)
9    if (validate (p,c))
10     if (c.key=k)
11       return false
12     else
13       n = new Node(k,true,-)
14       n.next = c
15       p.next = n
16       return true
17     unlock p
18     unlock c
19     exit
20   else
21     unlock p
22     unlock c
```

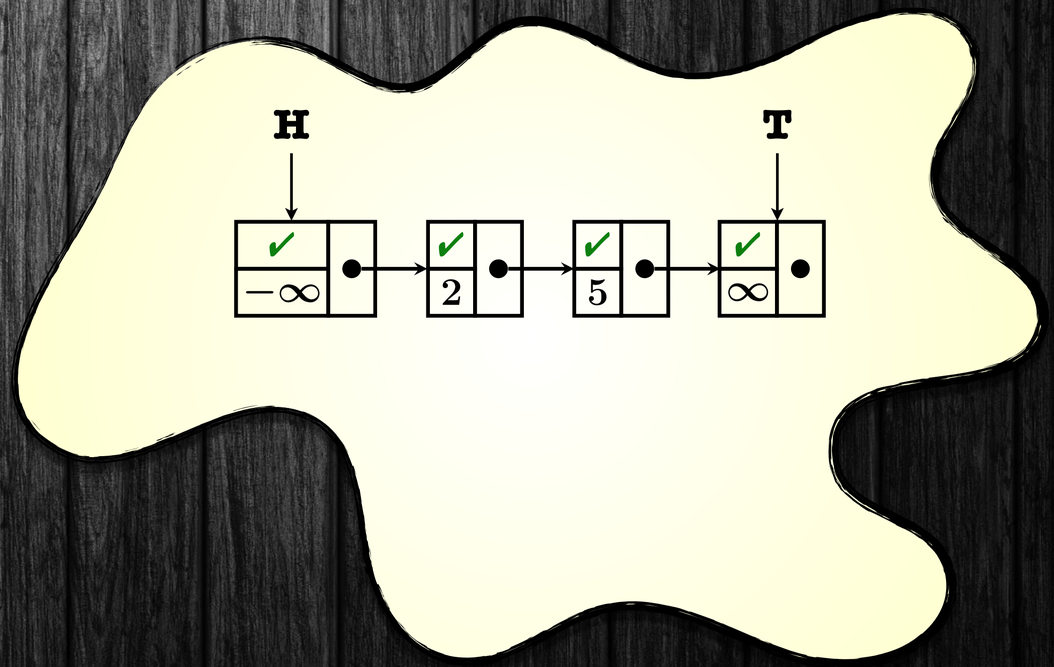




Sets

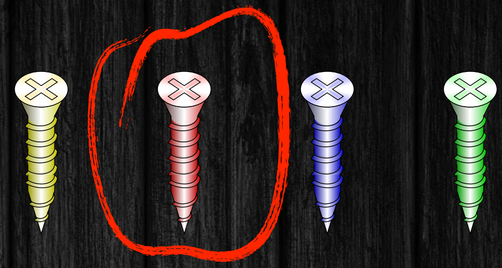
Optimistic Synchronization

```
add(k):
Node p, c
1  while (true)
2    p = H
3    c = p.next
4    while (c.key < k)
5      p = c
6      c = c.next
7    lock(p)
8    lock(c)
9    if (validate (p,c))
10     if (c.key=k)
11       return false
12     else
13       n = new Node(k,true,-)
14       n.next = c
15       p.next = n
16       return true
17     unlock p
18     unlock c
19     exit
20   else
21     unlock p
22     unlock c
```

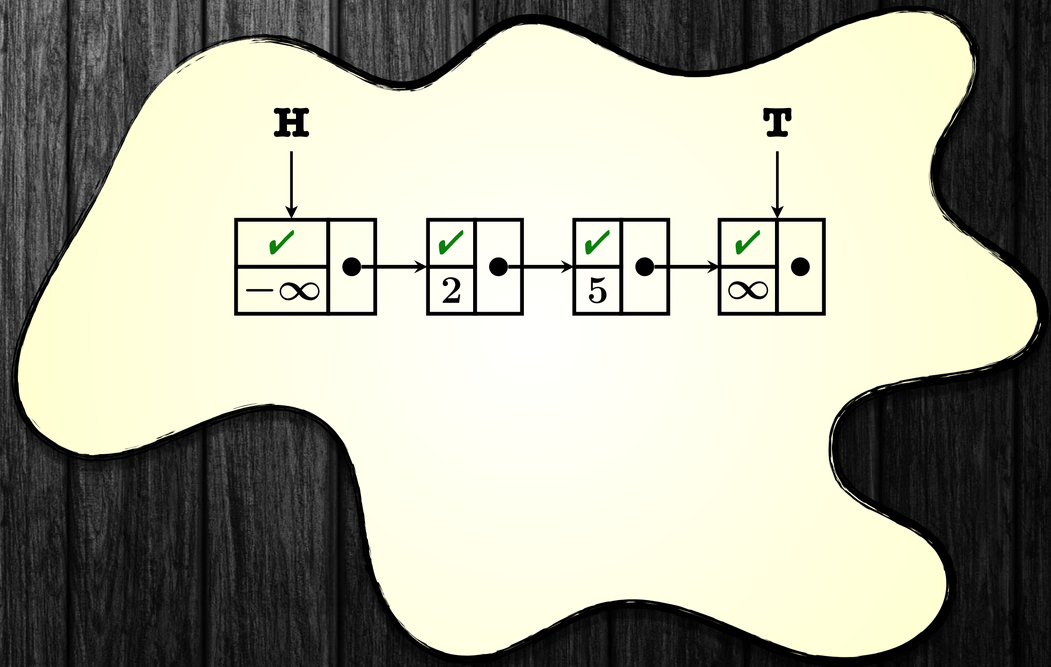


Sets

Optimistic Synchronization



```
add(k):
Node p, c
1  while (true)
2    p = H
3    c = p.next
4    while (c.key < k)
5      p = c
6      c = c.next
7    lock(p)
8    lock(c)
9    if (validate (p,c))
10     if (c.key=k)
11       return false
12     else
13       n = new Node(k,true,-)
14       n.next = c
15       p.next = n
16       return true
17     unlock p
18     unlock c
19     exit
20   else
21     unlock p
22     unlock c
```



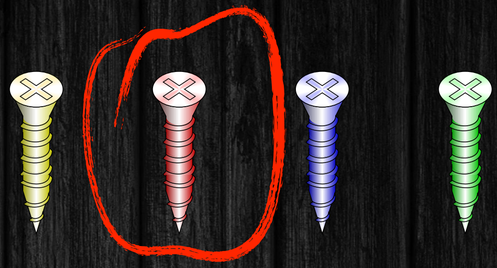
Sets

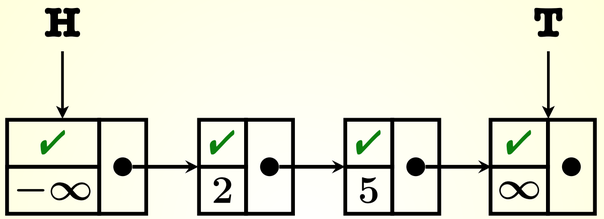
Optimistic Synchronization

```
add(k):
Node p, c
1  while (true)
2    p = H
3    c = p.next
4    while (c.key < k)
5      p = c
6      c = c.next
7    lock(p)
8    lock(c)
9    if (validate (p,c))
10     if (c.key=k)
11       return false
12     else
13       n = new Node(k,true,-)
14       n.next = c
15       p.next = n
16       return true
17     unlock p
18     unlock c
19     exit
20   else
21     unlock p
22     unlock c
```



add(4)





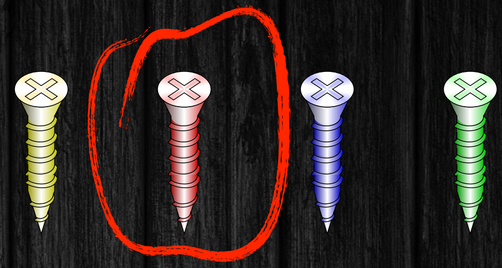
Sets

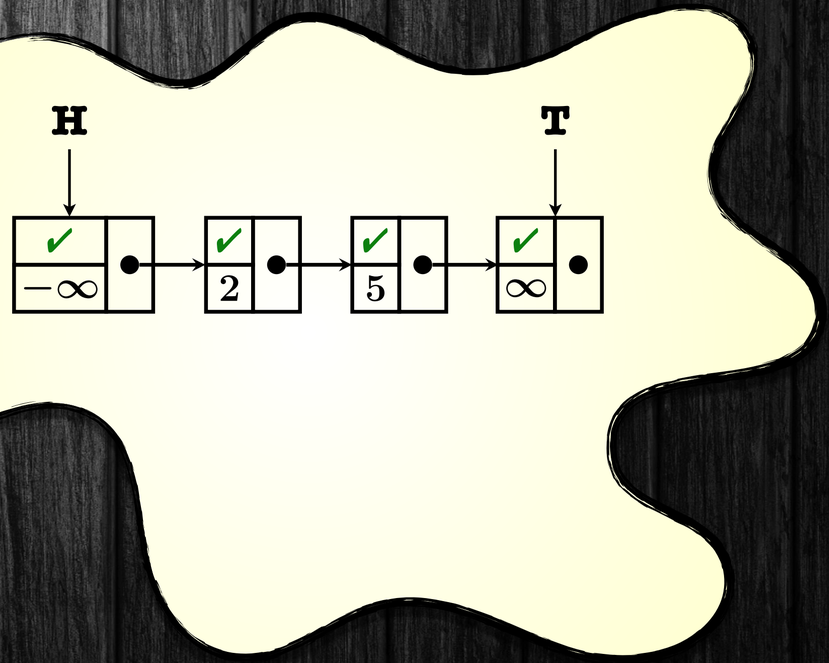
Optimistic Synchronization

```
add(k):
Node p, c
1  while (true)
2    p = H
3    c = p.next
4    while (c.key < k)
5      p = c
6      c = c.next
7    lock(p)
8    lock(c)
9    if (validate (p,c))
10     if (c.key=k)
11       return false
12     else
13       n = new Node(k,true,-)
14       n.next = c
15       p.next = n
16       return true
17     unlock p
18     unlock c
19     exit
20   else
21     unlock p
22     unlock c
```



add(4)





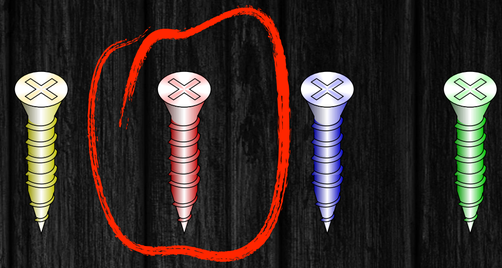
Sets

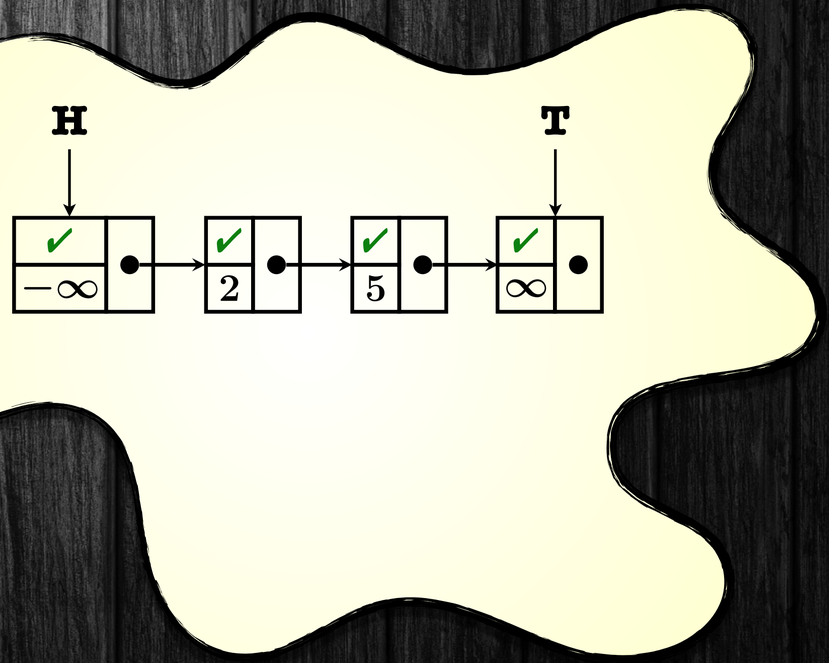
Optimistic Synchronization

```
add(k):
Node p, c
1  while (true)
2    p = H
3    c = p.next
4    while (c.key < k)
5      p = c
6      c = c.next
7    lock(p)
8    lock(c)
9    if (validate (p,c))
10     if (c.key=k)
11       return false
12     else
13       n = new Node(k,true,-)
14       n.next = c
15       p.next = n
16       return true
17     unlock p
18     unlock c
19     exit
20   else
21     unlock p
22     unlock c
```



add(4)



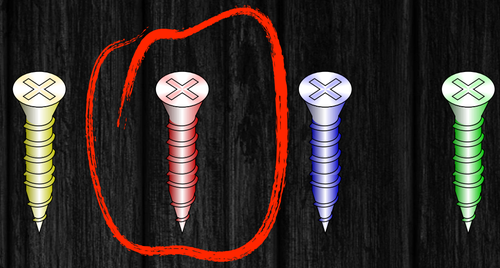


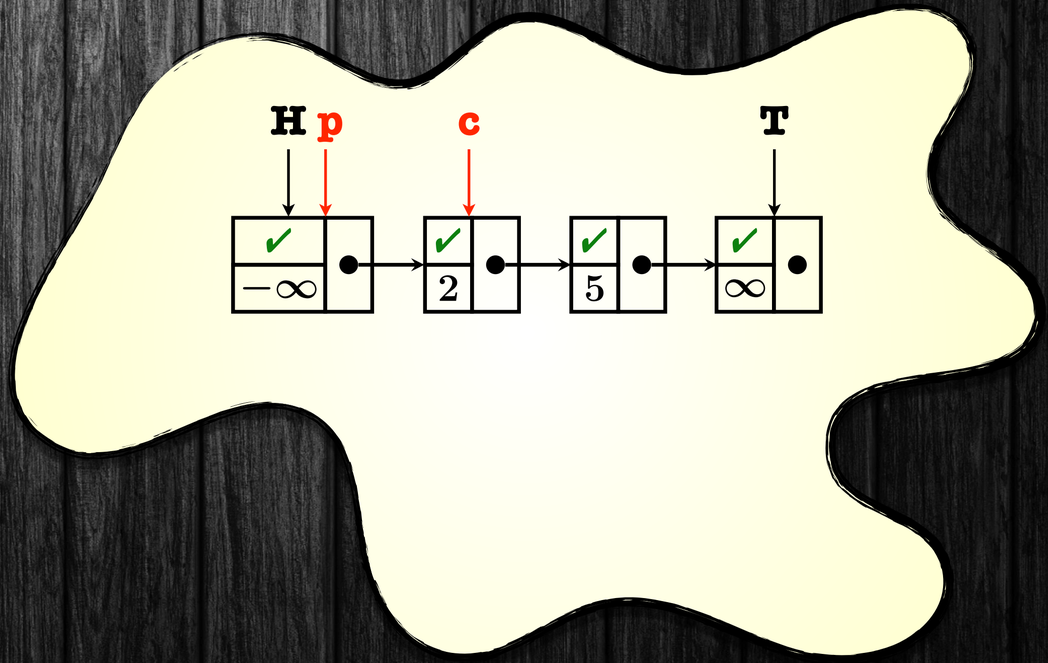
Sets | Optimistic Synchronization

```
add(k):
Node p, c
1  while (true)
2    p = H
3    c = p.next
4    while (c.key < k)
5      p = c
6      c = c.next
7    lock(p)
8    lock(c)
9    if (validate (p,c))
10     if (c.key=k)
11       return false
12     else
13       n = new Node(k,true,-)
14       n.next = c
15       p.next = n
16       return true
17     unlock p
18     unlock c
19     exit
20   else
21     unlock p
22     unlock c
```



add(4)



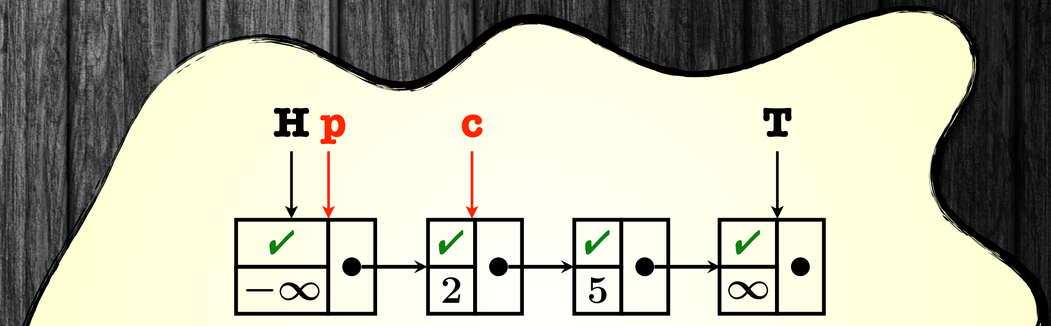


Sets

Optimistic Synchronization

```
add(k):
Node p, c
1  while (true)
2    p = H
3    c = p.next
4    while (c.key < k)
5      p = c
6      c = c.next
7    lock(p)
8    lock(c)
9    if (validate (p,c))
10     if (c.key=k)
11       return false
12     else
13       n = new Node(k,true,-)
14       n.next = c
15       p.next = n
16       return true
17     unlock p
18     unlock c
19     exit
20   else
21     unlock p
22     unlock c
```

add(4)

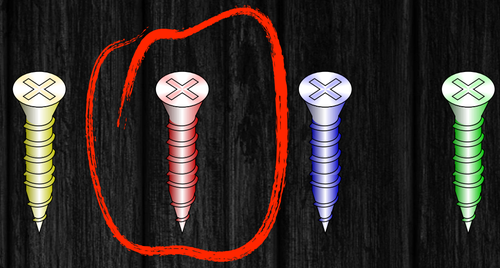


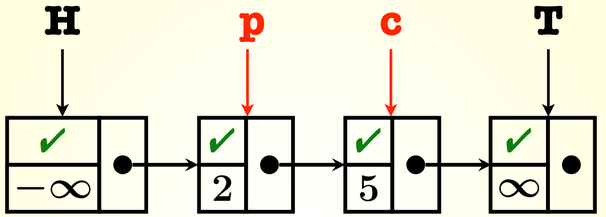
Sets | Optimistic Synchronization

```
add(k):
Node p, c
1  while (true)
2    p = H
3    c = p.next
4    while (c.key < k)
5      p = c
6      c = c.next
7    lock(p)
8    lock(c)
9    if (validate (p,c))
10     if (c.key=k)
11       return false
12     else
13       n = new Node(k,true,-)
14       n.next = c
15       p.next = n
16       return true
17     unlock p
18     unlock c
19     exit
20   else
21     unlock p
22     unlock c
```



add(4)

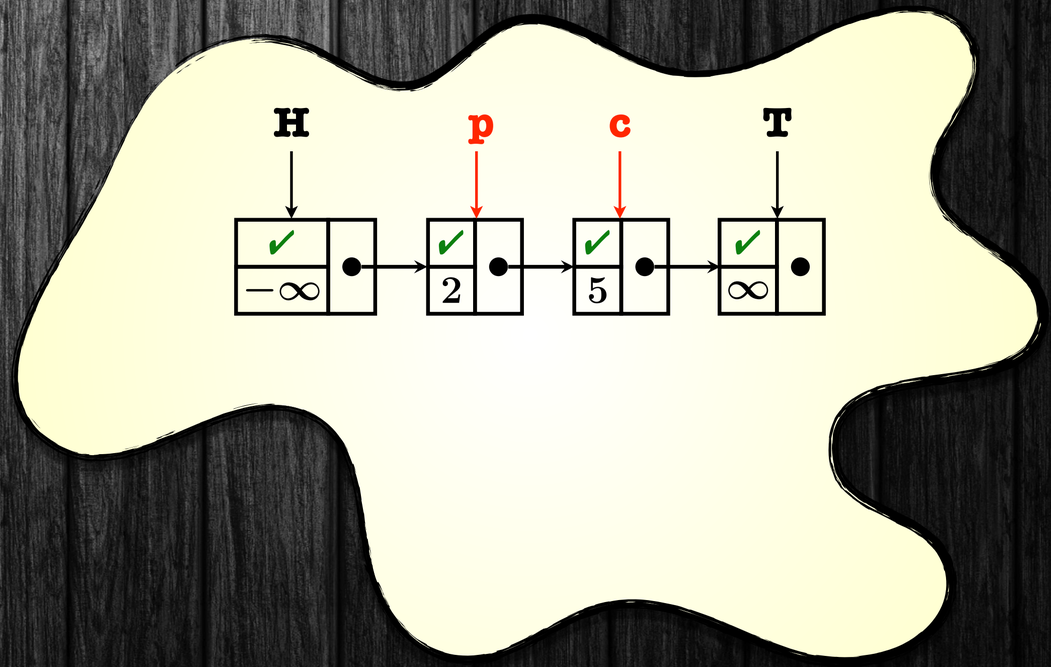




Sets | Optimistic Synchronization

```
add(k):
Node p, c
1  while (true)
2    p = H
3    c = p.next
4    while (c.key < k)
5      p = c
6      c = c.next
7    lock(p)
8    lock(c)
9    if (validate (p,c))
10     if (c.key=k)
11       return false
12     else
13       n = new Node(k,true,-)
14       n.next = c
15       p.next = n
16       return true
17     unlock p
18     unlock c
19     exit
20   else
21     unlock p
22     unlock c
```

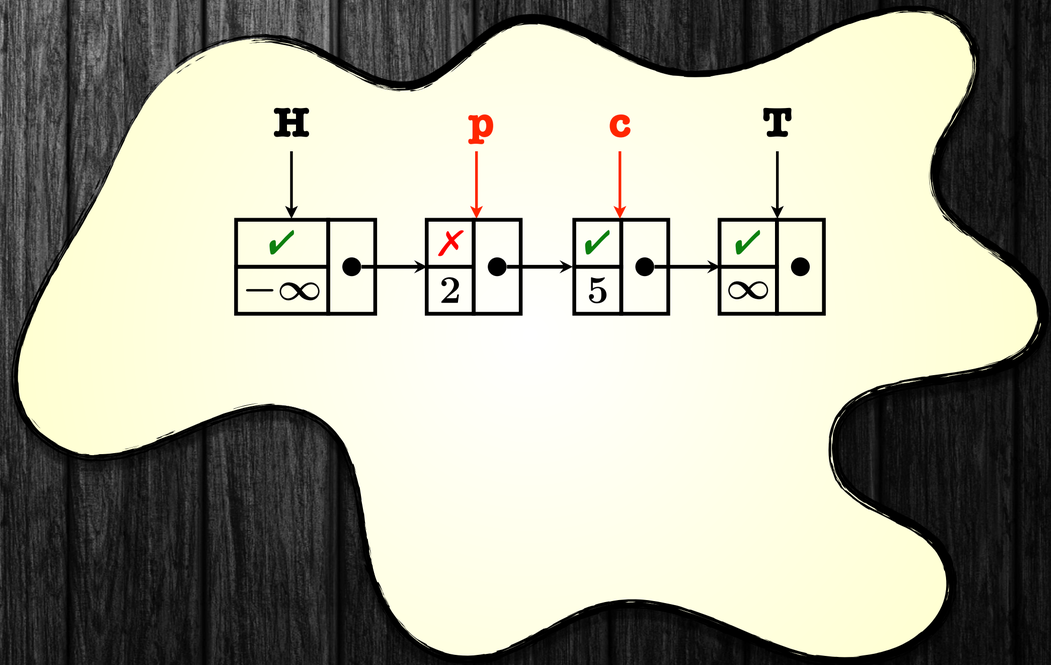
add(4)



Sets | Optimistic Synchronization

```
add(k):
Node p, c
1  while (true)
2    p = H
3    c = p.next
4    while (c.key < k)
5      p = c
6      c = c.next
7    lock(p)
8    lock(c)
9    if (validate (p,c))
10     if (c.key=k)
11       return false
12     else
13       n = new Node(k,true,-)
14       n.next = c
15       p.next = n
16       return true
17     unlock p
18     unlock c
19     exit
20   else
21     unlock p
22     unlock c
```

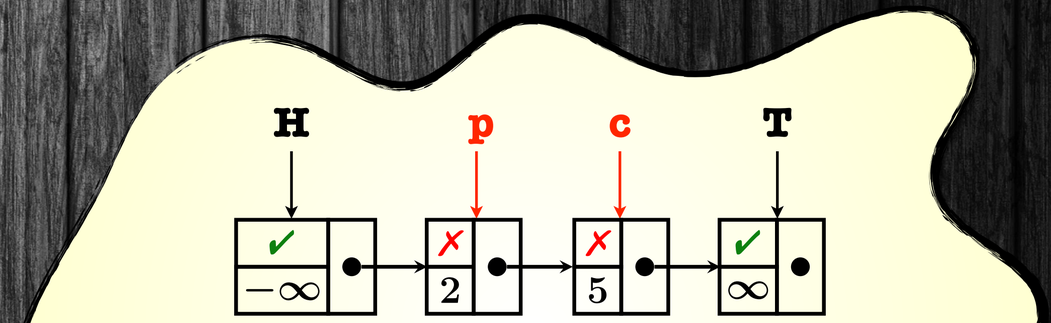
add(4)



Sets | Optimistic Synchronization

```
add(k):
Node p, c
1  while (true)
2    p = H
3    c = p.next
4    while (c.key < k)
5      p = c
6      c = c.next
7    lock(p)
8    lock(c)
9    if (validate (p,c))
10     if (c.key=k)
11       return false
12     else
13       n = new Node(k,true,-)
14       n.next = c
15       p.next = n
16       return true
17     unlock p
18     unlock c
19     exit
20   else
21     unlock p
22     unlock c
```

add(4)

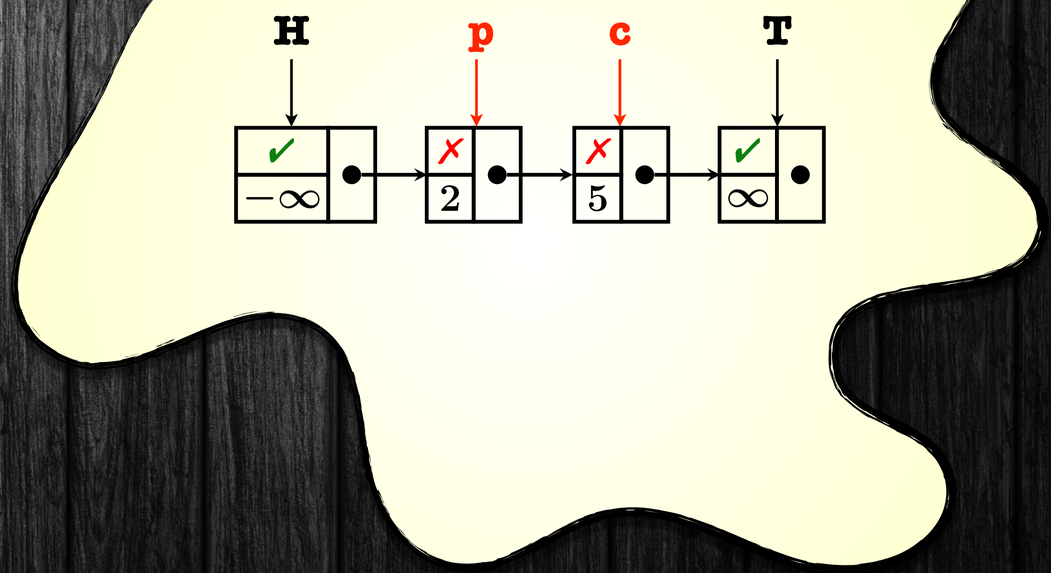


Sets

Optimistic Synchronization

```
add(k):
Node p, c
1  while (true)
2    p = H
3    c = p.next
4    while (c.key < k)
5      p = c
6      c = c.next
7    lock(p)
8    lock(c)
9    if (validate (p,c))
10     if (c.key=k)
11       return false
12     else
13       n = new Node(k,true,-)
14       n.next = c
15       p.next = n
16       return true
17     unlock p
18     unlock c
19     exit
20   else
21     unlock p
22     unlock c
```

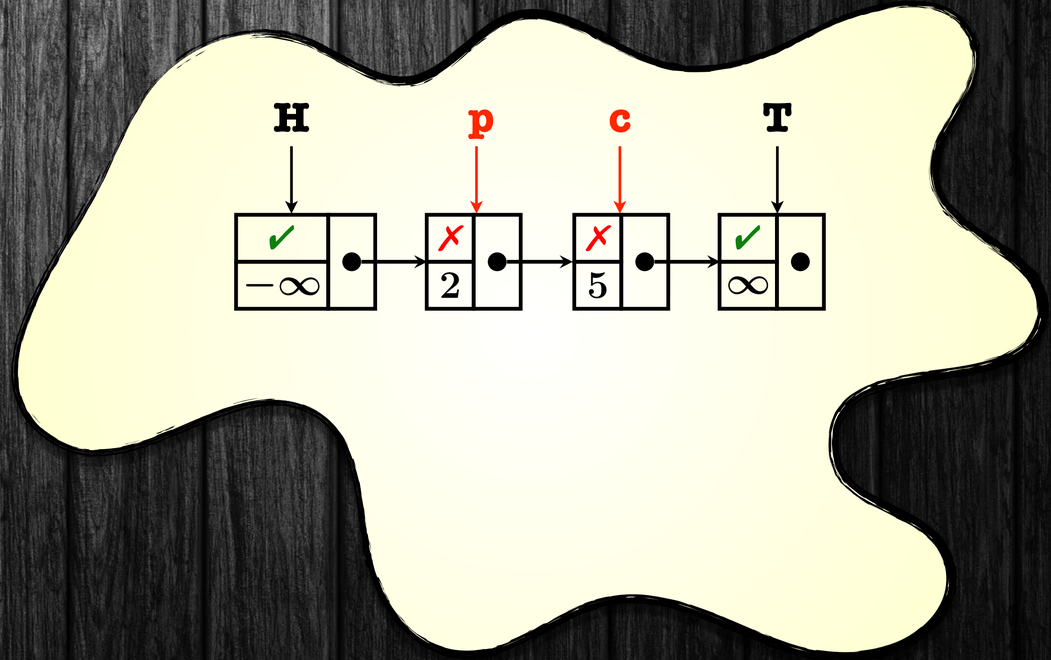
add(4)



Sets | Optimistic Synchronization

```
add(k):
Node p, c
1  while (true)
2    p = H
3    c = p.next
4    while (c.key < k)
5      p = c
6      c = c.next
7    lock(p)
8    lock(c)
9    if (validate (p,c))
10     if (c.key=k)
11       return false
12     else
13       n = new Node(k,true,-)
14       n.next = c
15       p.next = n
16       return true
17     unlock p
18     unlock c
19     exit
20   else
21     unlock p
22     unlock c
```

add(4)



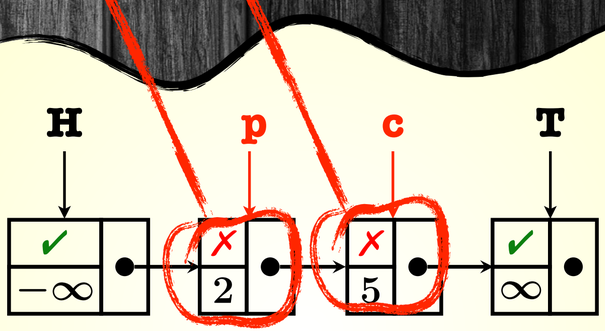
Sets | Optimistic Synchronization

```
add(k):
Node p, c
1  while (true)
2    p = H
3    c = p.next
4    while (c.key < k)
5      p = c
6      c = c.next
7    lock(p)
8    lock(c)
9    if (validate (p,c))
10     if (c.key=k)
11       return false
12     else
13       n = new Node(k,true,-)
14       n.next = c
15       p.next = n
16       return true
17     unlock p
18     unlock c
19     exit
20   else
21     unlock p
22     unlock c
```

validation

add(4)

check whether the two cells are still in the list



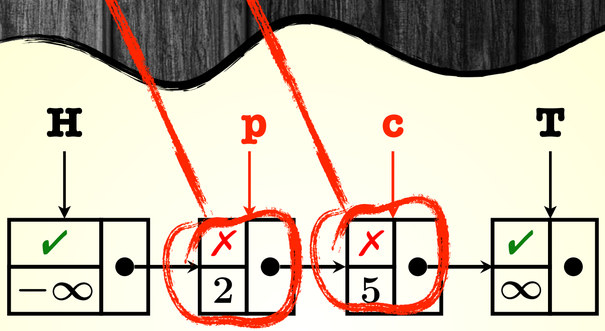
Sets | Optimistic Synchronization

```
add(k):
Node p, c
1  while (true)
2    p = H
3    c = p.next
4    while (c.key < k)
5      p = c
6      c = c.next
7    lock(p)
8    lock(c)
9    if (validate (p,c))
10     if (c.key=k)
11       return false
12     else
13       n = new Node(k,true,-)
14       n.next = c
15       p.next = n
16       return true
17     unlock p
18     unlock c
19     exit
20   else
21     unlock p
22     unlock c
```

validation

add(4)

check whether the two cells are still in the list



Sets | Optimistic Synchronization

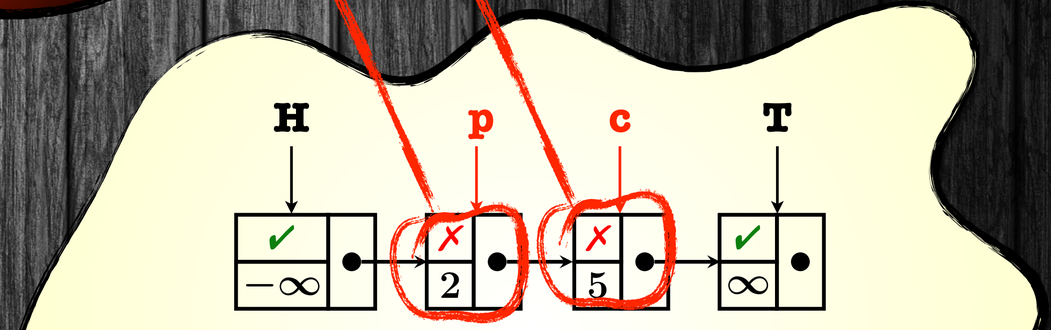
```
add(k):
Node p, c
1  while (true)
2    p = H
3    c = p.next
4    while (c.key < k)
5      p = c
6      c = c.next
7    lock(p)
8    lock(c)
9    if (validate (p,c))
10     if (c.key=k)
11       return false
12     else
13       n = new Node(k,true,-)
14       n.next = c
15       p.next = n
16       return true
17     unlock p
18     unlock c
19     exit
20   else
21     unlock p
22     unlock c
```

validation

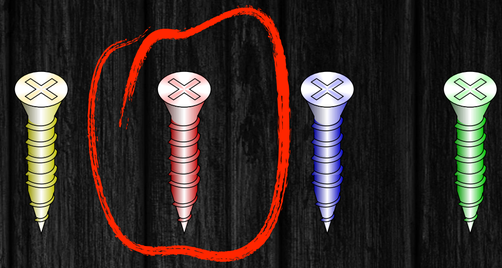
add(4)

check whether the two cells are still in the list

assume yes



Sets | Optimistic Synchronization



```
add(k):
Node p, c
1  while (true)
2    p = H
3    c = p.next
4    while (c.key < k)
5      p = c
6      c = c.next
7    lock(p)
8    lock(c)
9    if (validate (p,c))
10     if (c.key=k)
11       return false
12     else
13       n = new Node(k,true,-)
14       n.next = c
15       p.next = n
16       return true
17     unlock p
18     unlock c
19     exit
20   else
21     unlock p
22     unlock c
```

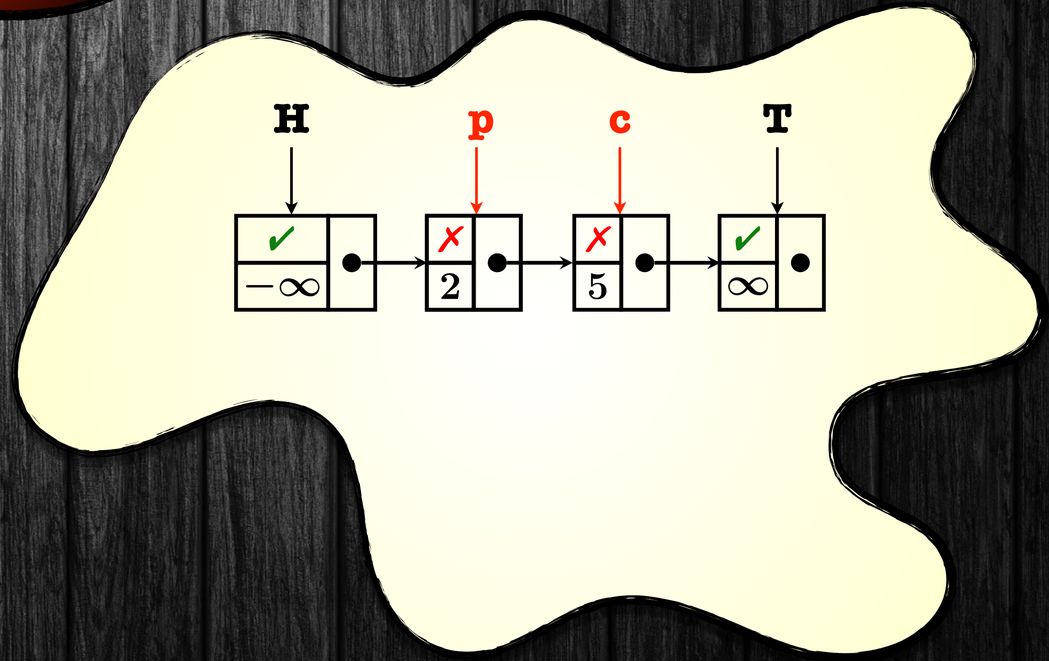


add(4)

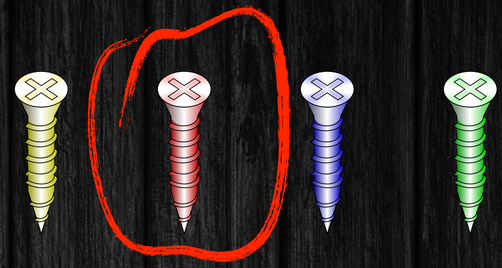
validation

check whether the two cells are still in the list

assume yes



Sets | Optimistic Synchronization



```
add(k):
Node p, c
1  while (true)
2    p = H
3    c = p.next
4    while (c.key < k)
5      p = c
6      c = c.next
7    lock(p)
8    lock(c)
9    if (validate (p,c))
10     if (c.key=k)
11       return false
12     else
13       n = new Node(k,true,-)
14       n.next = c
15       p.next = n
16       return true
17     unlock p
18     unlock c
19     exit
20   else
21     unlock p
22     unlock c
```

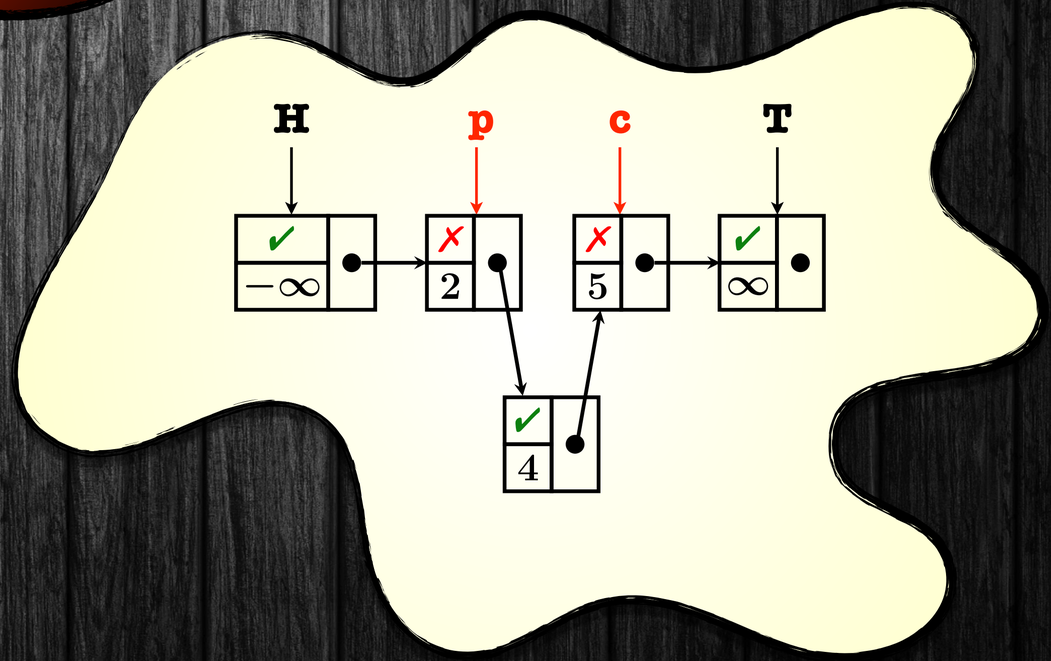


add(4)

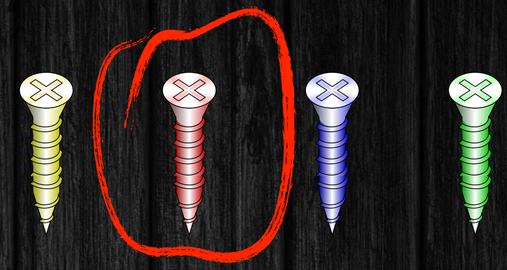
validation

check whether the two cells are still in the list

assume yes



Sets | Optimistic Synchronization



```
add(k):
Node p, c
1  while (true)
2    p = H
3    c = p.next
4    while (c.key < k)
5      p = c
6      c = c.next
7    lock(p)
8    lock(c)
9    if (validate (p,c))
10     if (c.key=k)
11       return false
12     else
13       n = new Node(k,true,-)
14       n.next = c
15       p.next = n
16       return true
17     unlock p
18     unlock c
19     exit
20   else
21     unlock p
22     unlock c
```

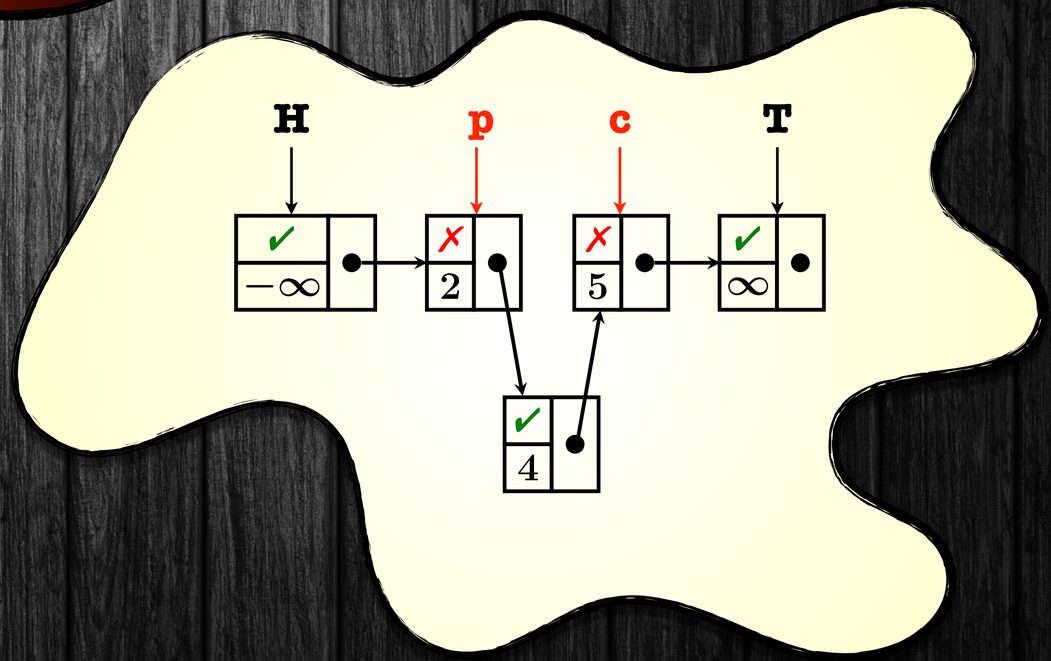


add(4)

validation

check whether the two cells are still in the list

assume yes



Sets | Optimistic Synchronization

```
add(k):
Node p, c
1  while (true)
2    p = H
3    c = p.next
4    while (c.key < k)
5      p = c
6      c = c.next
7    lock(p)
8    lock(c)
9    if (validate (p,c))
10     if (c.key=k)
11       return false
12     else
13       n = new Node(k,true,-)
14       n.next = c
15       p.next = n
16       return true
17     unlock p
18     unlock c
19     exit
20   else
21     unlock p
22     unlock c
```

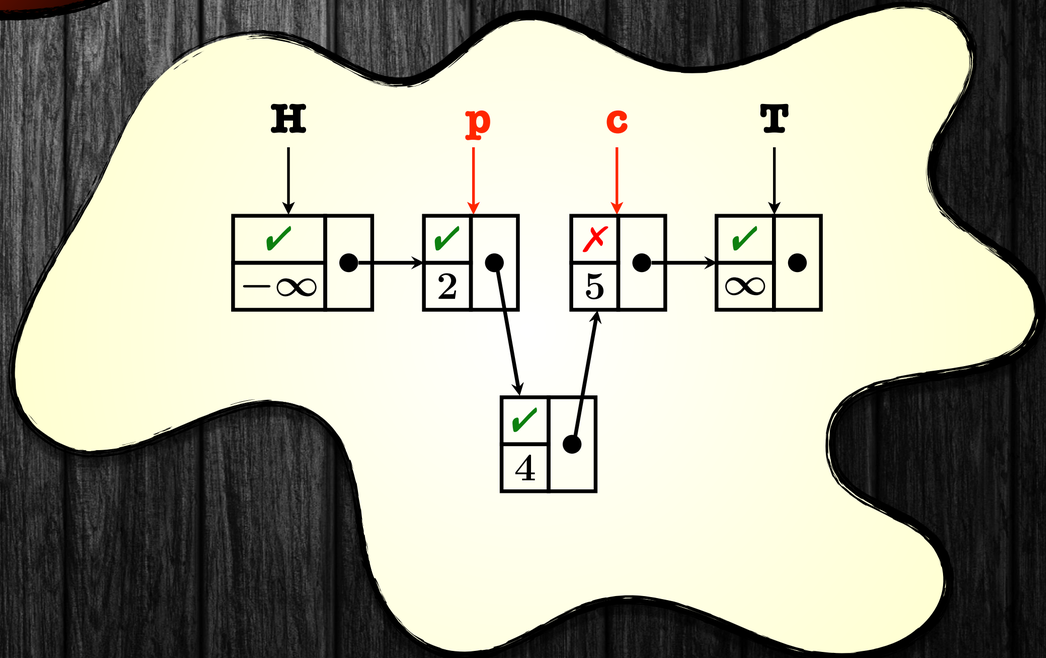


add(4)

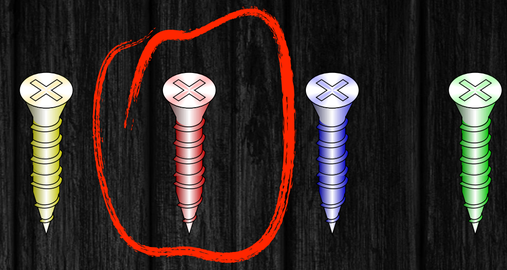
validation

check whether the two cells are still in the list

assume yes



Sets | Optimistic Synchronization



```
add(k):
Node p, c
1  while (true)
2    p = H
3    c = p.next
4    while (c.key < k)
5      p = c
6      c = c.next
7    lock(p)
8    lock(c)
9    if (validate (p,c))
10     if (c.key=k)
11       return false
12     else
13       n = new Node(k,true,-)
14       n.next = c
15       p.next = n
16       return true
17     unlock p
18     unlock c
19     exit
20   else
21     unlock p
22     unlock c
```

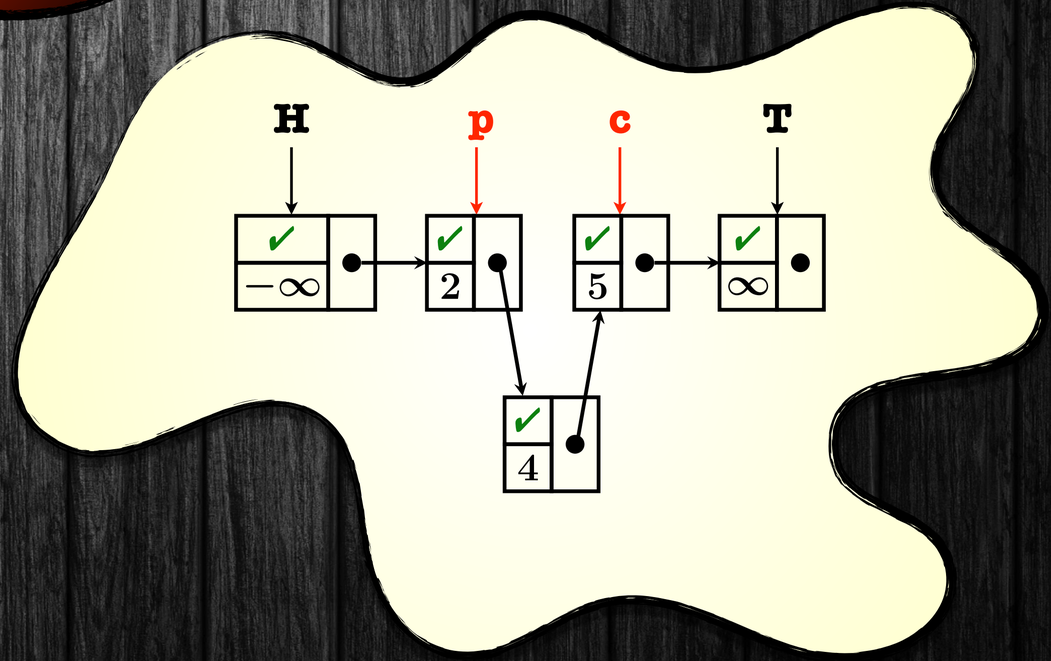


add(4)

validation

check whether the two cells are still in the list

assume yes



Sets | Optimistic Synchronization

```
add(k):
Node p, c
1  while (true)
2    p = H
3    c = p.next
4    while (c.key < k)
5      p = c
6      c = c.next
7    lock(p)
8    lock(c)
9    if (validate (p,c))
10     if (c.key=k)
11       return false
12     else
13       n = new Node(k,true,-)
14       n.next = c
15       p.next = n
16       return true
17     unlock p
18     unlock c
19     exit
20   else
21     unlock p
22     unlock c
```

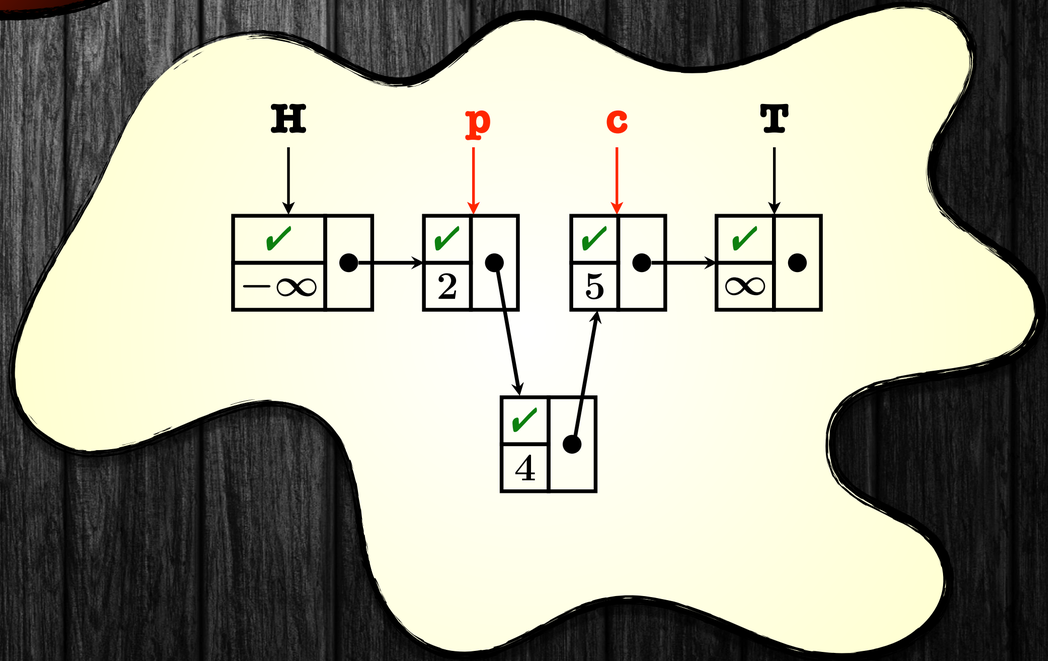


add(4)

validation

check whether the two cells are still in the list

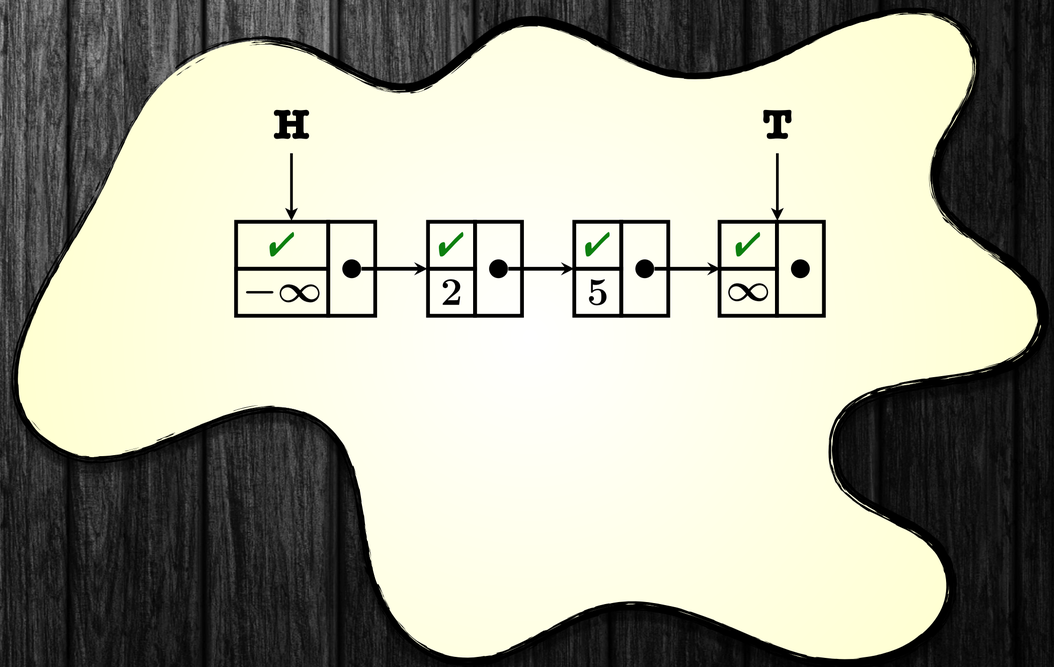
assume yes



Sets

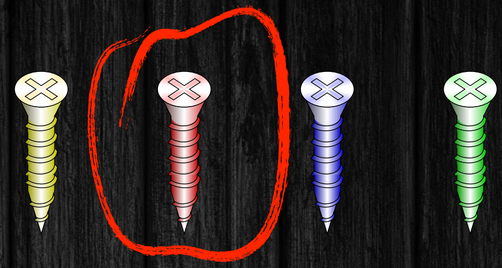
Optimistic Synchronization

```
add(k):
Node p, c
1  while (true)
2    p = H
3    c = p.next
4    while (c.key < k)
5      p = c
6      c = c.next
7    lock(p)
8    lock(c)
9    if (validate (p,c))
10     if (c.key=k)
11       return false
12     else
13       n = new Node(k,true,-)
14       n.next = c
15       p.next = n
16       return true
17     unlock p
18     unlock c
19     exit
20   else
21     unlock p
22     unlock c
```

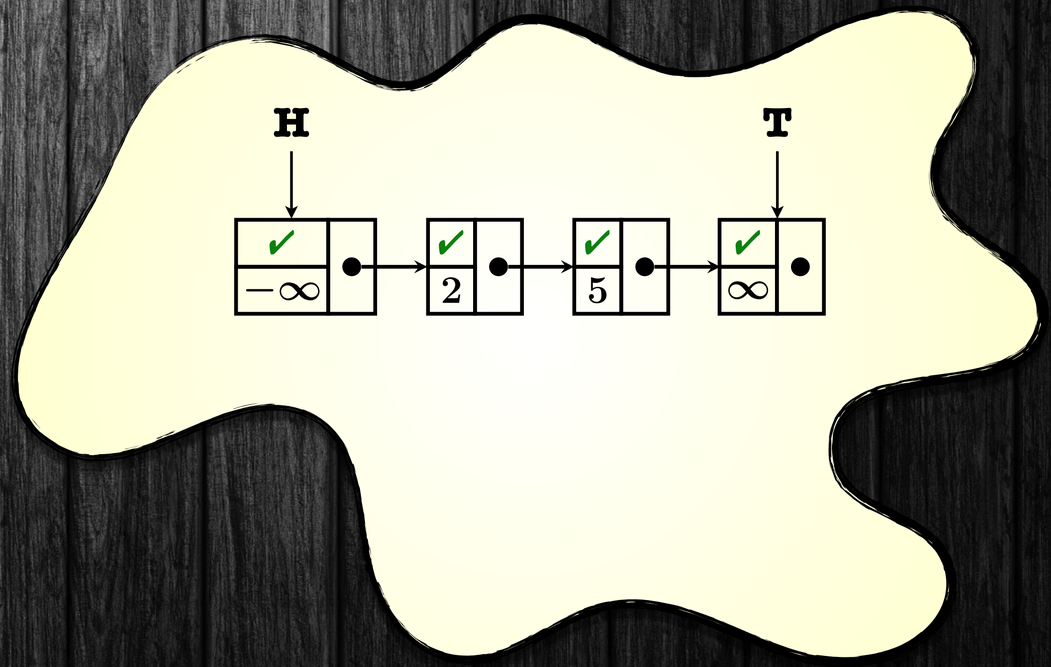


Sets

Optimistic Synchronization



```
add(k):
Node p, c
1  while (true)
2    p = H
3    c = p.next
4    while (c.key < k)
5      p = c
6      c = c.next
7    lock(p)
8    lock(c)
9    if (validate (p,c))
10     if (c.key=k)
11       return false
12     else
13       n = new Node(k,true,-)
14       n.next = c
15       p.next = n
16       return true
17     unlock p
18     unlock c
19     exit
20   else
21     unlock p
22     unlock c
```



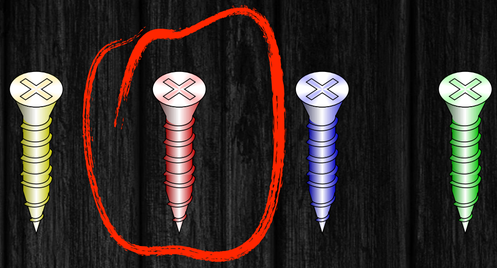
Sets

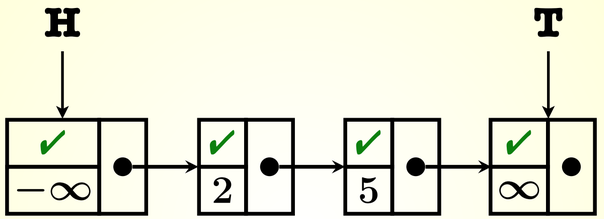
Optimistic Synchronization

```
add(k):
Node p, c
1  while (true)
2    p = H
3    c = p.next
4    while (c.key < k)
5      p = c
6      c = c.next
7    lock(p)
8    lock(c)
9    if (validate (p,c))
10     if (c.key=k)
11       return false
12     else
13       n = new Node(k,true,-)
14       n.next = c
15       p.next = n
16       return true
17     unlock p
18     unlock c
19     exit
20   else
21     unlock p
22     unlock c
```



add(4)





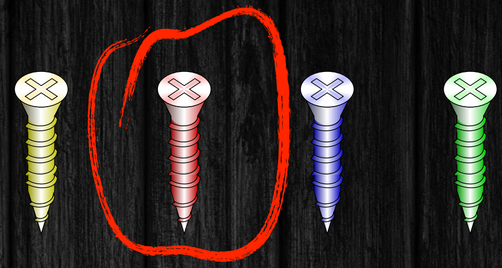
Sets

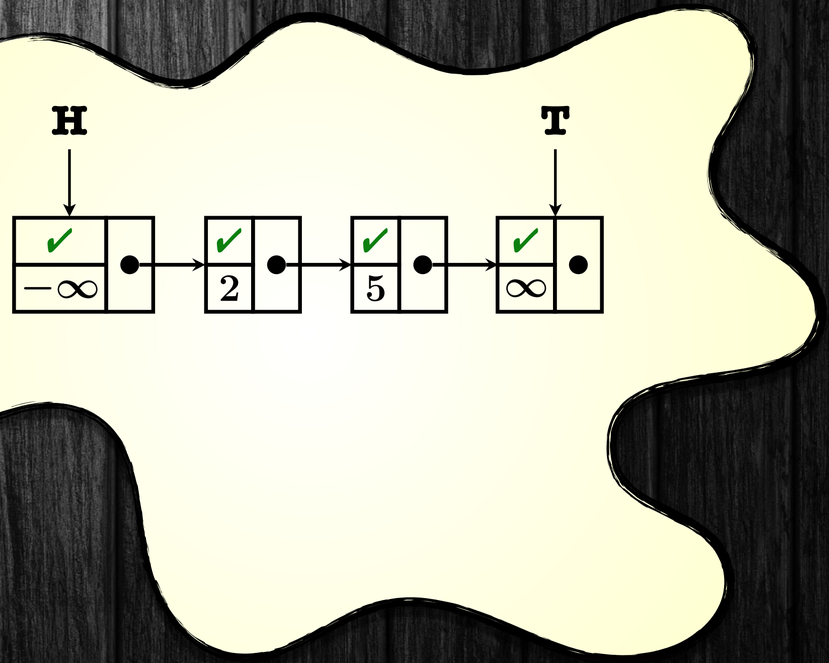
Optimistic Synchronization

```
add(k):
Node p, c
1  while (true)
2    p = H
3    c = p.next
4    while (c.key < k)
5      p = c
6      c = c.next
7    lock(p)
8    lock(c)
9    if (validate (p,c))
10     if (c.key=k)
11       return false
12     else
13       n = new Node(k,true,-)
14       n.next = c
15       p.next = n
16       return true
17     unlock p
18     unlock c
19     exit
20   else
21     unlock p
22     unlock c
```



add(4)





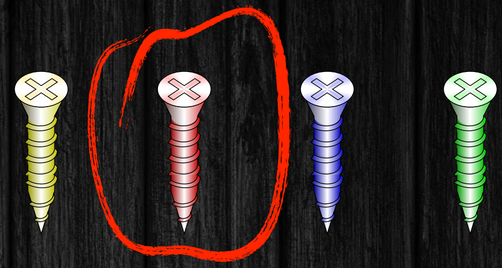
Sets

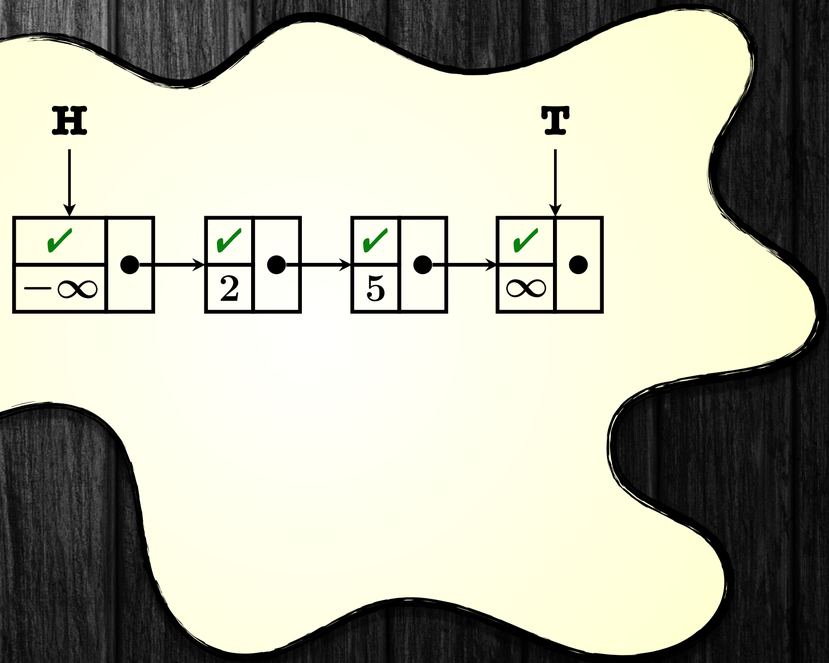
Optimistic Synchronization

```
add(k):
Node p, c
1  while (true)
2    p = H
3    c = p.next
4    while (c.key < k)
5      p = c
6      c = c.next
7    lock(p)
8    lock(c)
9    if (validate (p,c))
10     if (c.key=k)
11       return false
12     else
13       n = new Node(k,true,-)
14       n.next = c
15       p.next = n
16       return true
17     unlock p
18     unlock c
19     exit
20   else
21     unlock p
22     unlock c
```



add(4)



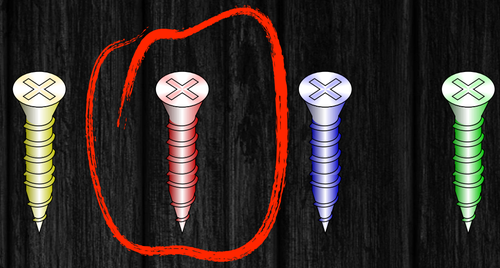


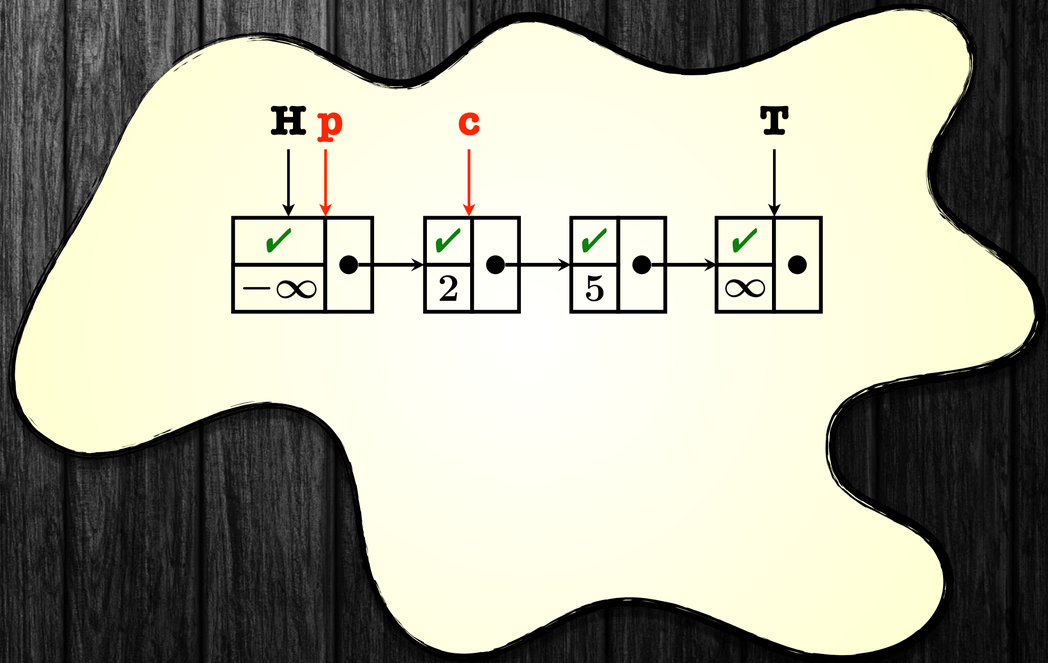
Sets | Optimistic Synchronization

```
add(k):
Node p, c
1  while (true)
2    p = H
3    c = p.next
4    while (c.key < k)
5      p = c
6      c = c.next
7    lock(p)
8    lock(c)
9    if (validate (p,c))
10     if (c.key=k)
11       return false
12     else
13       n = new Node(k,true,-)
14       n.next = c
15       p.next = n
16       return true
17     unlock p
18     unlock c
19     exit
20   else
21     unlock p
22     unlock c
```



add(4)



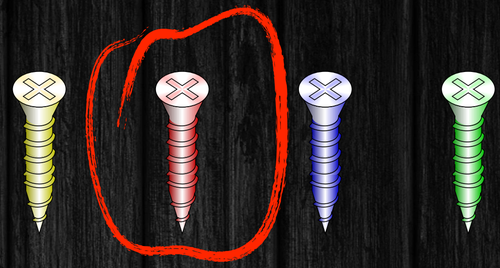


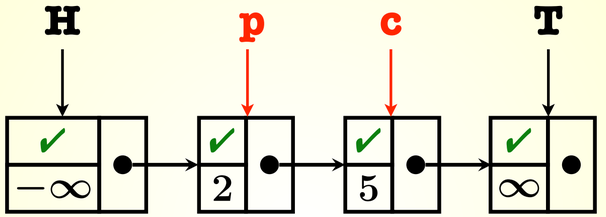
Sets | Optimistic Synchronization

```
add(k):
Node p, c
1  while (true)
2    p = H
3    c = p.next
4    while (c.key < k)
5      p = c
6      c = c.next
7    lock(p)
8    lock(c)
9    if (validate (p,c))
10     if (c.key=k)
11       return false
12     else
13       n = new Node(k,true,-)
14       n.next = c
15       p.next = n
16       return true
17     unlock p
18     unlock c
19     exit
20   else
21     unlock p
22     unlock c
```



add(4)

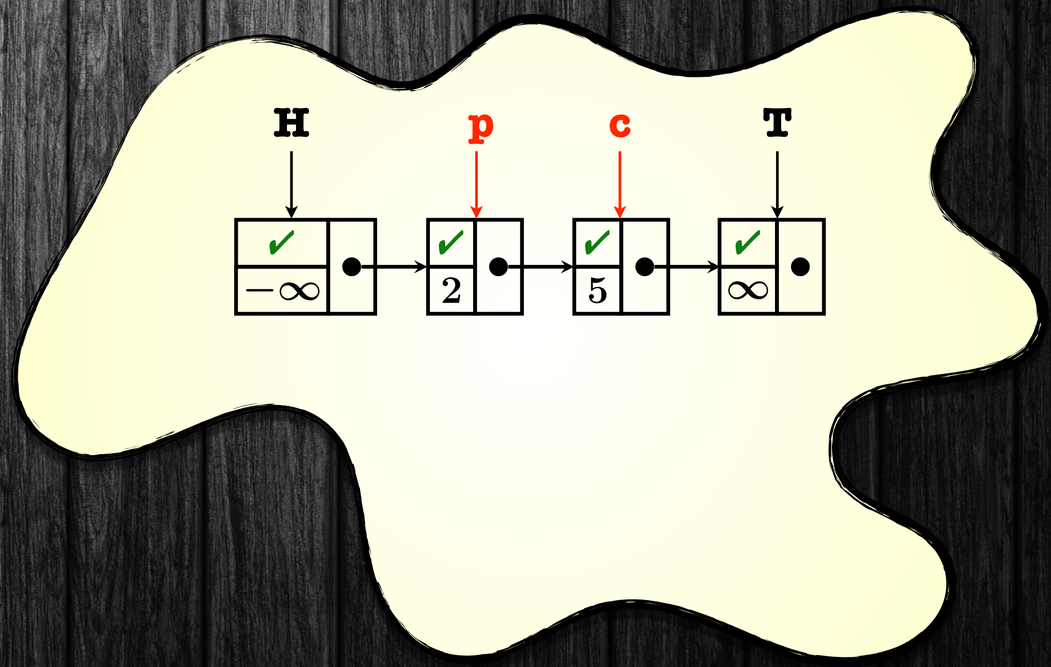




Sets | Optimistic Synchronization

```
add(k):
Node p, c
1  while (true)
2    p = H
3    c = p.next
4    while (c.key < k)
5      p = c
6      c = c.next
7    lock(p)
8    lock(c)
9    if (validate (p,c))
10     if (c.key=k)
11       return false
12     else
13       n = new Node(k,true,-)
14       n.next = c
15       p.next = n
16       return true
17     unlock p
18     unlock c
19     exit
20   else
21     unlock p
22     unlock c
```

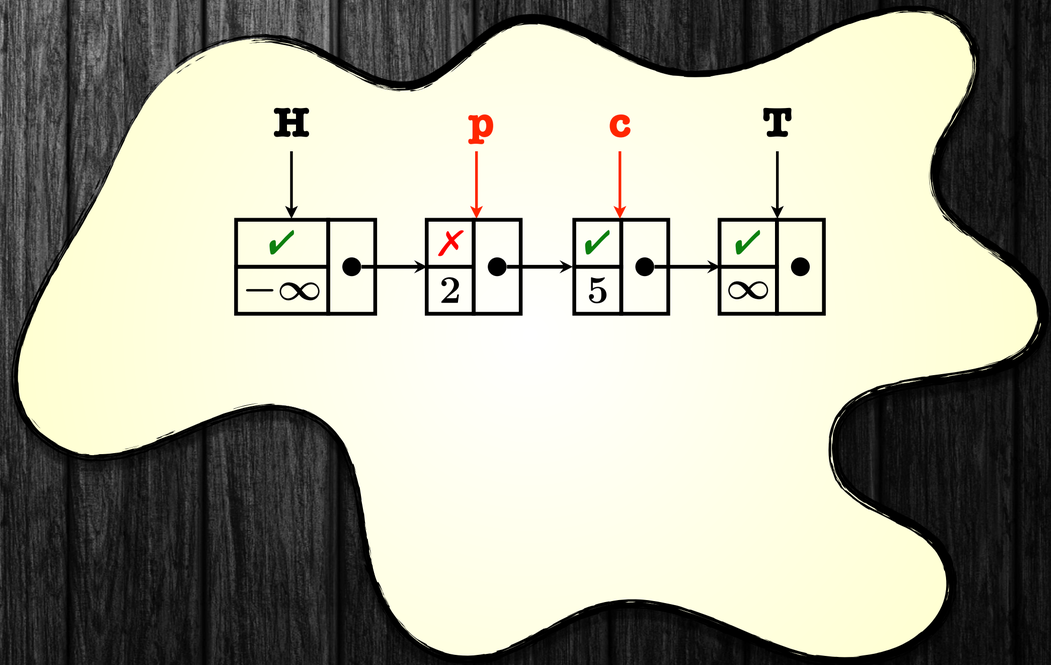
add(4)



Sets | Optimistic Synchronization

```
add(k):
Node p, c
1  while (true)
2    p = H
3    c = p.next
4    while (c.key < k)
5      p = c
6      c = c.next
7    lock(p)
8    lock(c)
9    if (validate (p,c))
10     if (c.key=k)
11       return false
12     else
13       n = new Node(k,true,-)
14       n.next = c
15       p.next = n
16       return true
17     unlock p
18     unlock c
19     exit
20   else
21     unlock p
22     unlock c
```

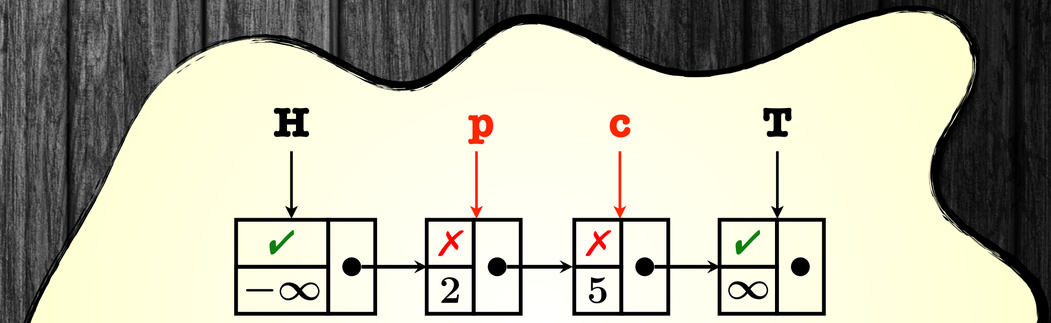
add(4)



Sets | Optimistic Synchronization

```
add(k):
Node p, c
1  while (true)
2    p = H
3    c = p.next
4    while (c.key < k)
5      p = c
6      c = c.next
7    lock(p)
8    lock(c)
9    if (validate (p,c))
10     if (c.key=k)
11       return false
12     else
13       n = new Node(k,true,-)
14       n.next = c
15       p.next = n
16       return true
17     unlock p
18     unlock c
19     exit
20   else
21     unlock p
22     unlock c
```

add(4)

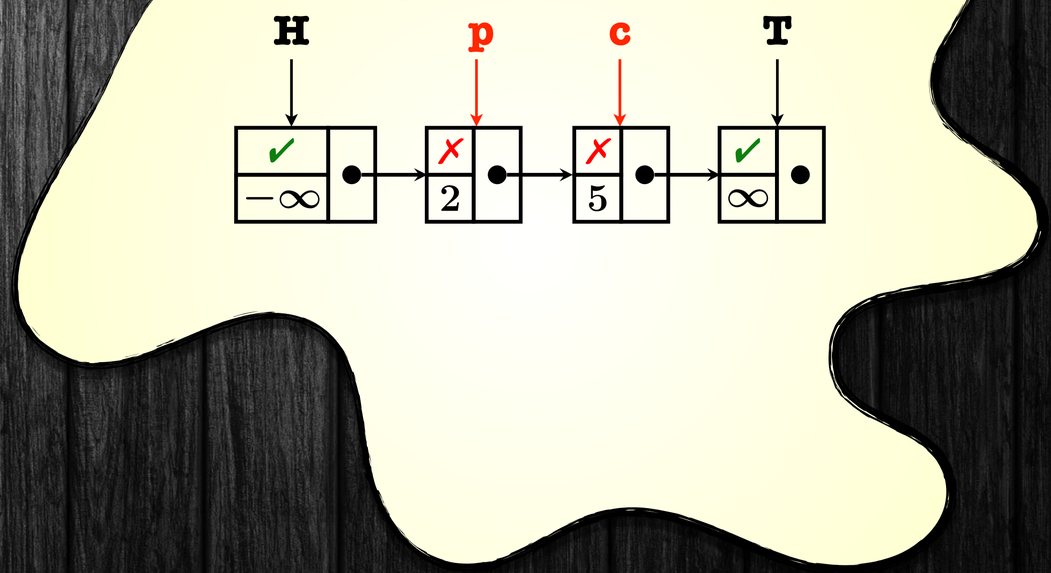


Sets

Optimistic Synchronization

```
add(k):
Node p, c
1  while (true)
2    p = H
3    c = p.next
4    while (c.key < k)
5      p = c
6      c = c.next
7    lock(p)
8    lock(c)
9    if (validate (p,c))
10     if (c.key=k)
11       return false
12     else
13       n = new Node(k,true,-)
14       n.next = c
15       p.next = n
16       return true
17     unlock p
18     unlock c
19     exit
20   else
21     unlock p
22     unlock c
```

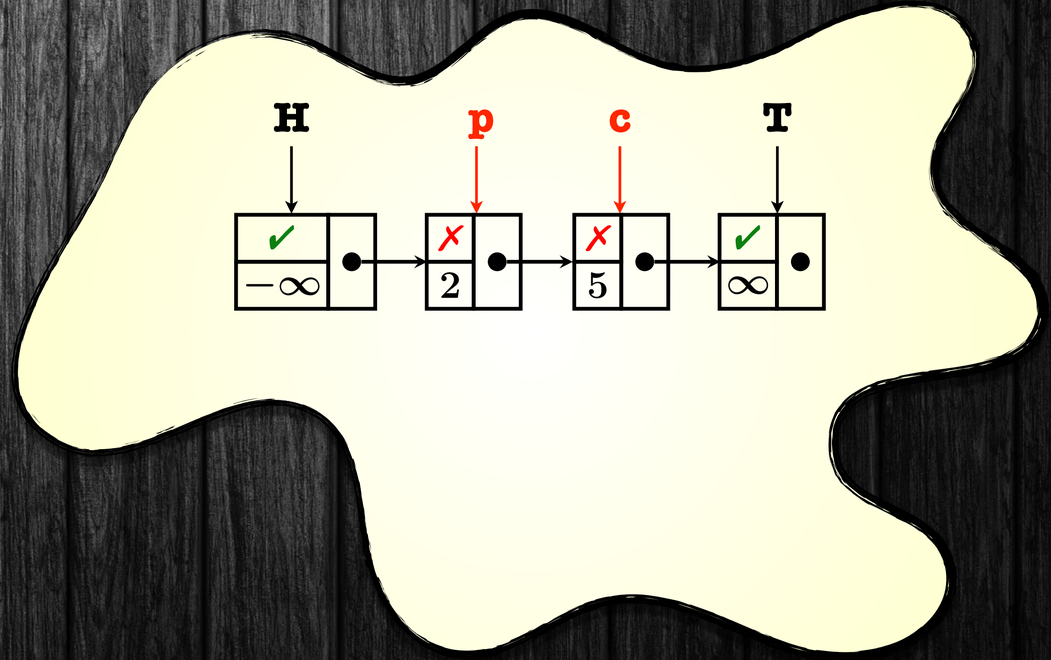
add(4)



Sets | Optimistic Synchronization

```
add(k):
Node p, c
1  while (true)
2    p = H
3    c = p.next
4    while (c.key < k)
5      p = c
6      c = c.next
7    lock(p)
8    lock(c)
9    if (validate (p,c))
10     if (c.key=k)
11       return false
12     else
13       n = new Node(k,true,-)
14       n.next = c
15       p.next = n
16       return true
17     unlock p
18     unlock c
19     exit
20   else
21     unlock p
22     unlock c
```

add(4)



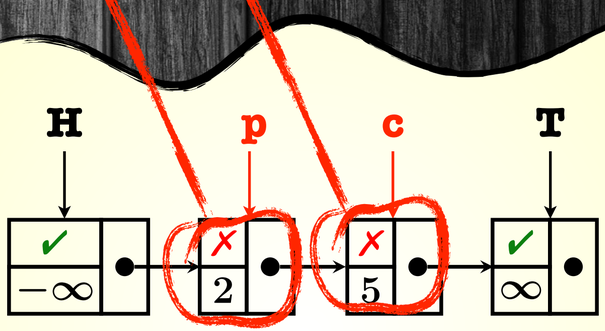
Sets | Optimistic Synchronization

```
add(k):
Node p, c
1  while (true)
2    p = H
3    c = p.next
4    while (c.key < k)
5      p = c
6      c = c.next
7    lock(p)
8    lock(c)
9    if (validate (p,c))
10     if (c.key=k)
11       return false
12     else
13       n = new Node(k,true,-)
14       n.next = c
15       p.next = n
16       return true
17     unlock p
18     unlock c
19     exit
20   else
21     unlock p
22     unlock c
```

validation

add(4)

check whether the two cells are still in the list



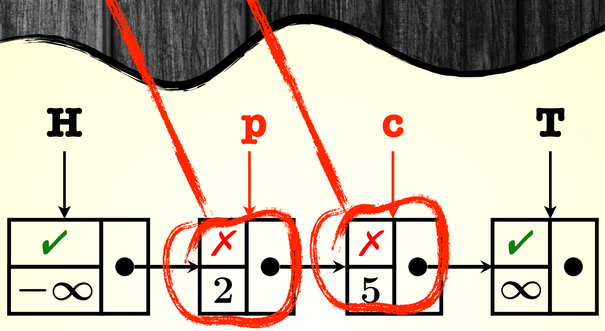
Sets | Optimistic Synchronization

```
add(k):
Node p, c
1  while (true)
2    p = H
3    c = p.next
4    while (c.key < k)
5      p = c
6      c = c.next
7    lock(p)
8    lock(c)
9    if (validate (p,c))
10     if (c.key=k)
11       return false
12     else
13       n = new Node(k,true,-)
14       n.next = c
15       p.next = n
16       return true
17     unlock p
18     unlock c
19     exit
20   else
21     unlock p
22     unlock c
```

validation

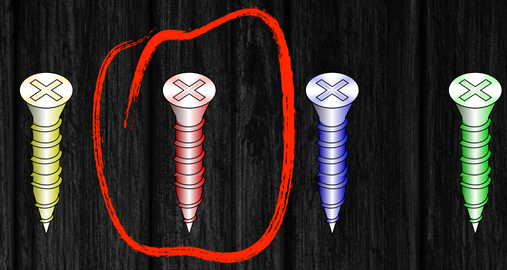
add(4)

check whether the two cells are still in the list



Sets | Optimistic Synchronization

```
add(k):
Node p, c
1  while (true)
2    p = H
3    c = p.next
4    while (c.key < k)
5      p = c
6      c = c.next
7    lock(p)
8    lock(c)
9    if (validate (p,c))
10     if (c.key=k)
11       return false
12     else
13       n = new Node(k,true,-)
14       n.next = c
15       p.next = n
16       return true
17     unlock p
18     unlock c
19     exit
20   else
21     unlock p
22     unlock c
```



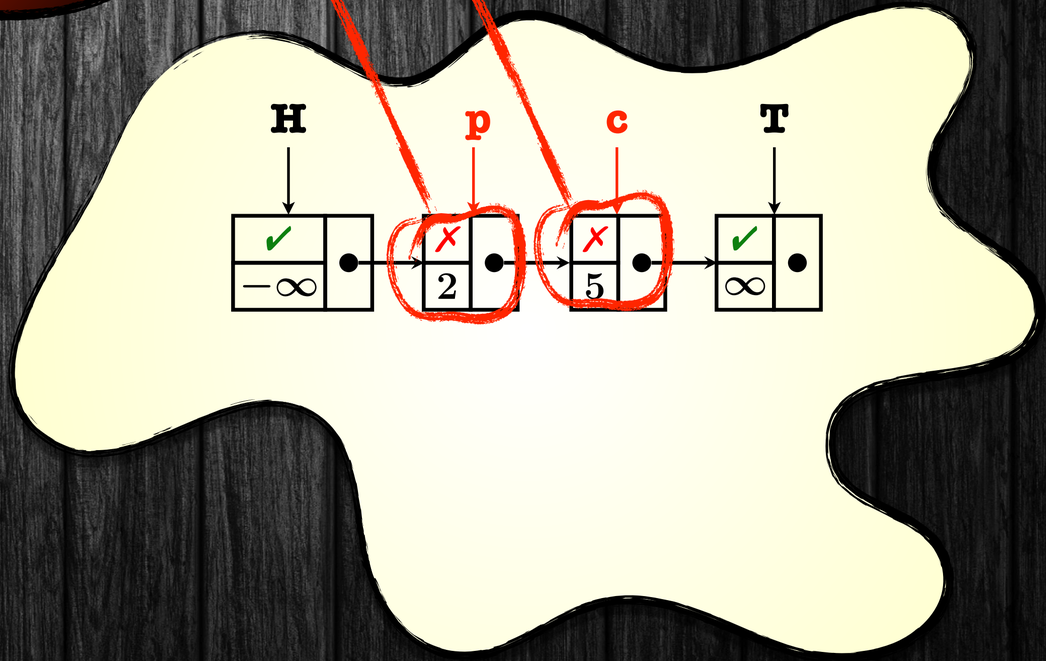


add(4)

validation

check whether the two cells are still in the list

assume no



Sets | Optimistic Synchronization

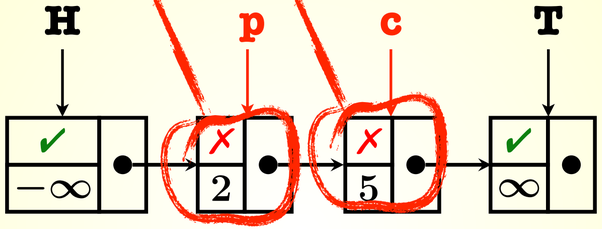
```
add(k):
Node p, c
1  while (true)
2    p = H
3    c = p.next
4    while (c.key < k)
5      p = c
6      c = c.next
7    lock(p)
8    lock(c)
9    if (validate (p,c)
10     if (c.key=k)
11       return false
12     else
13       n = new Node(k,true,-)
14       n.next = c
15       p.next = n
16       return true
17     unlock p
18     unlock c
19     exit
20   else ↜
21     unlock p
22     unlock c
```

validation

add(4)

check whether the two cells are still in the list

assume no



Sets | Optimistic Synchronization

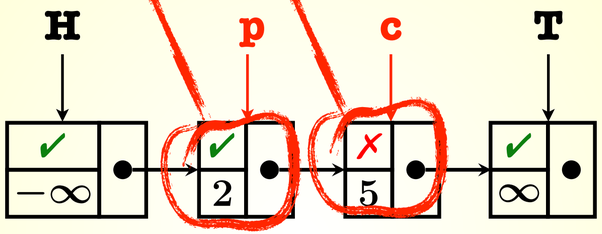
```
add(k):
Node p, c
1  while (true)
2    p = H
3    c = p.next
4    while (c.key < k)
5      p = c
6      c = c.next
7    lock(p)
8    lock(c)
9    if (validate (p,c)
10     if (c.key=k)
11       return false
12     else
13       n = new Node(k,true,-)
14       n.next = c
15       p.next = n
16       return true
17     unlock p
18     unlock c
19     exit
20   else ⟲
21     unlock p
22     unlock c
```

validation

add(4)

check whether the two cells are still in the list

assume no



Sets | Optimistic Synchronization

```
add(k):
Node p, c
1  while (true)
2    p = H
3    c = p.next
4    while (c.key < k)
5      p = c
6      c = c.next
7    lock(p)
8    lock(c)
9    if (validate (p,c)
10     if (c.key=k)
11       return false
12     else
13       n = new Node(k,true,-)
14       n.next = c
15       p.next = n
16       return true
17     unlock p
18     unlock c
19     exit
20   else ⟲
21     unlock p
22     unlock c
```

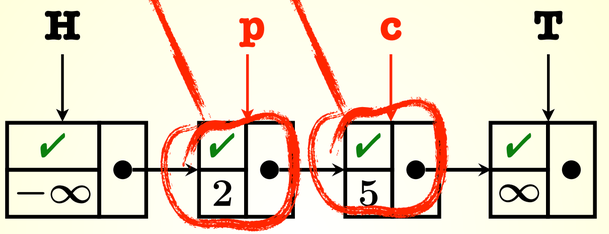


add(4)

validation

check whether the two cells are still in the list

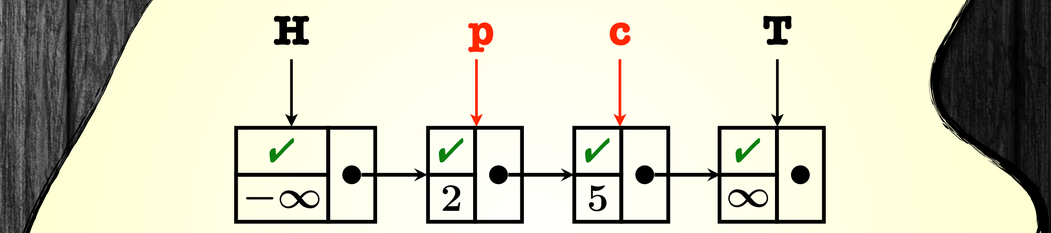
assume no



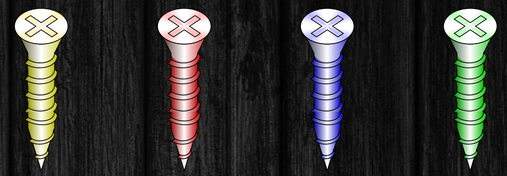
Sets | Optimistic Synchronization

```
add(k):
Node p, c
1  while (true)
2    p = H
3    c = p.next
4    while (c.key < k)
5      p = c
6      c = c.next
7    lock(p)
8    lock(c)
9    if (validate (p,c))
10     if (c.key=k)
11       return false
12     else
13       n = new Node(k,true,-)
14       n.next = c
15       p.next = n
16       return true
17     unlock p
18     unlock c
19     exit
20   else
21     unlock p
22     unlock c
```

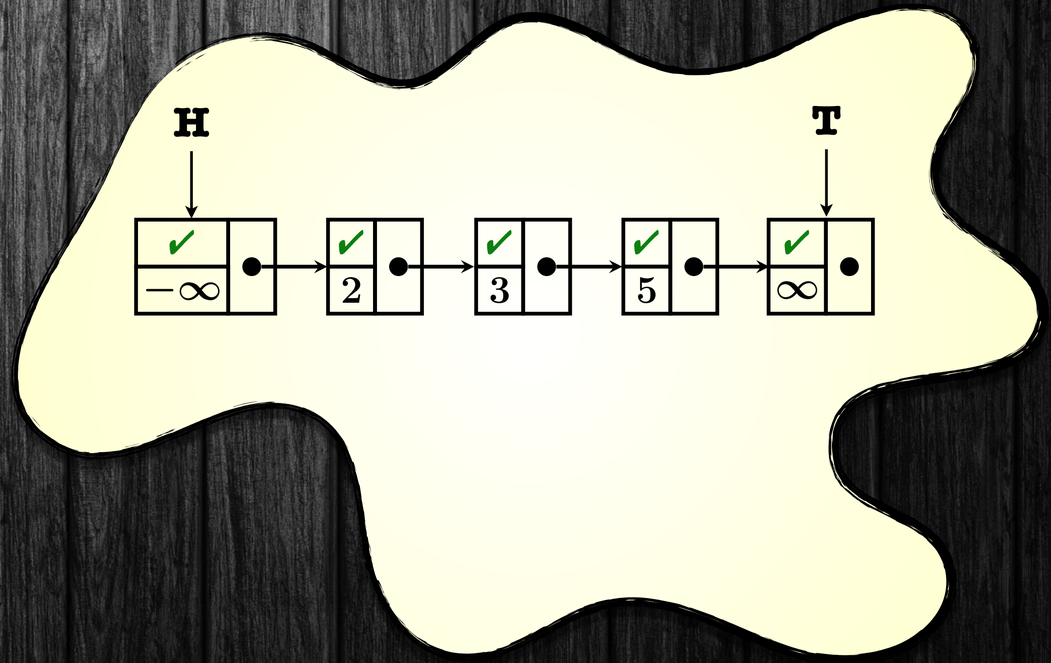
add(4)



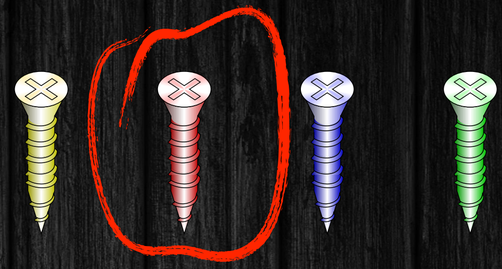
Sets | Optimistic Synchronization



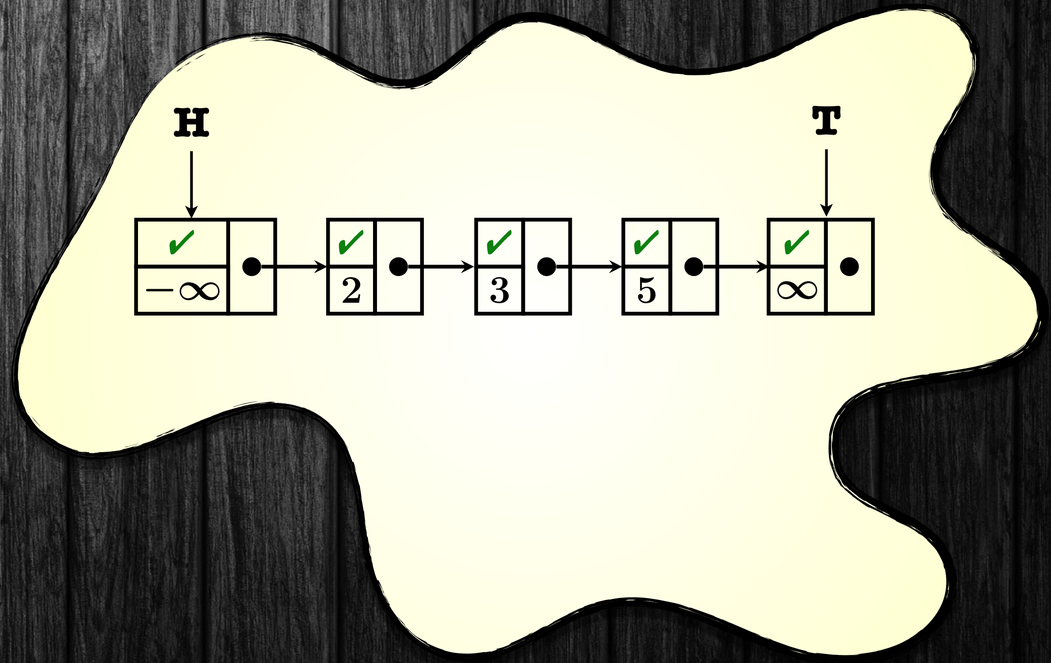
```
rmv(k):
Node p, c
 1  while (true)
 2    p = H
 3    c = p.next
 4    while (c.key < k)
 5      p = c
 6      c = c.next
 7    lock(p)
 8    lock(c)
 9    if (validate (p,c))
10      if (c.key=k)
11        p.next = c.next
12        return true
13      else
14        return false
15      unlock p
16      unlock c
17      exit
18    else
19      unlock p
20      unlock c
```



Sets | Optimistic Synchronization



```
rmv(k):
Node p, c
 1  while (true)
 2    p = H
 3    c = p.next
 4    while (c.key < k)
 5      p = c
 6      c = c.next
 7    lock(p)
 8    lock(c)
 9    if (validate (p,c))
10      if (c.key=k)
11        p.next = c.next
12        return true
13      else
14        return false
15      unlock p
16      unlock c
17      exit
18    else
19      unlock p
20      unlock c
```

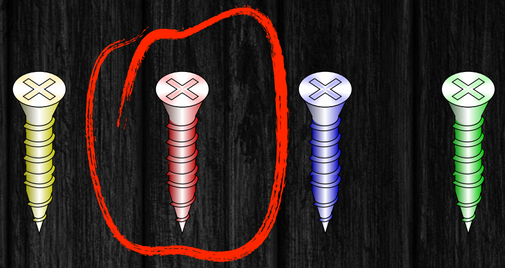


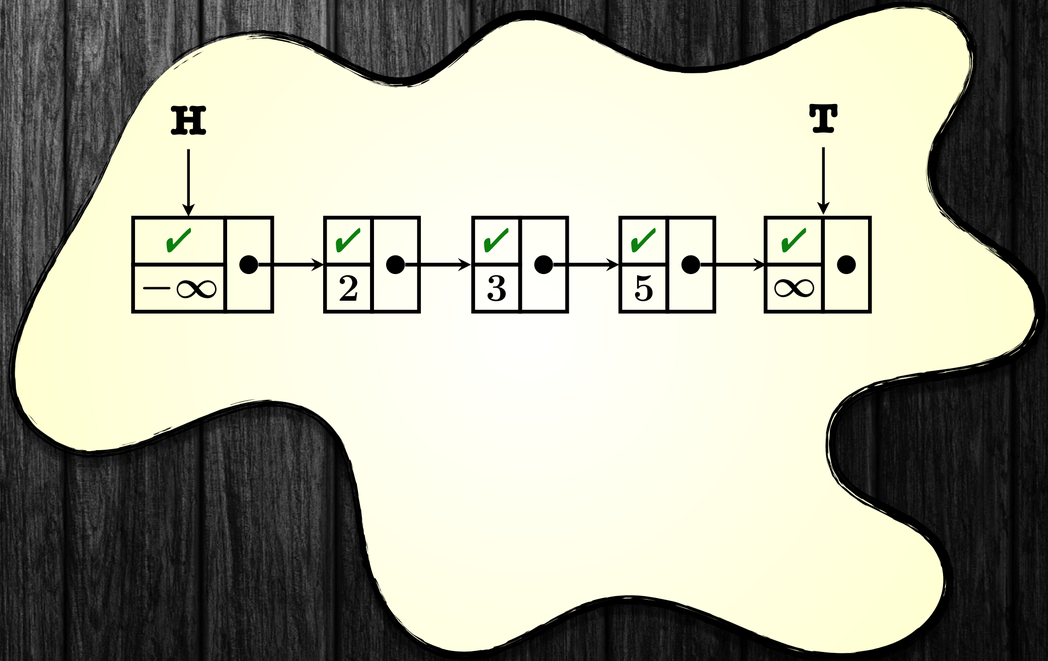
Sets | Optimistic Synchronization

```
rmv(k):
Node p, c
 1  while (true)
 2    p = H
 3    c = p.next
 4    while (c.key < k)
 5      p = c
 6      c = c.next
 7    lock(p)
 8    lock(c)
 9    if (validate (p,c))
10      if (c.key=k)
11        p.next = c.next
12        return true
13      else
14        return false
15      unlock p
16      unlock c
17      exit
18    else
19      unlock p
20      unlock c
```



rmv(5)



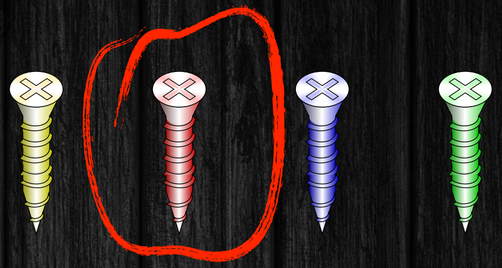


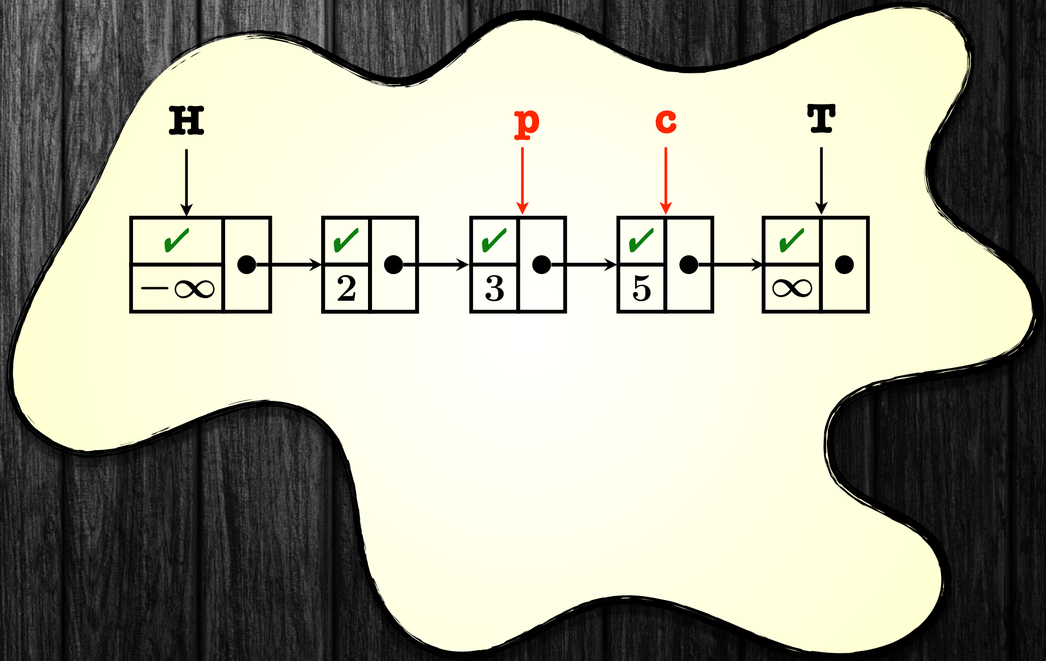
Sets Optimistic Synchronization

```
rmv(k):
Node p, c
1  while (true)
2    p = H
3    c = p.next
4    while (c.key < k)
5      p = c
6      c = c.next
7    lock(p)
8    lock(c)
9    if (validate (p,c))
10     if (c.key=k)
11       p.next = c.next
12       return true
13     else
14       return false
15     unlock p
16     unlock c
17     exit
18   else
19     unlock p
20     unlock c
```



rmv(5)



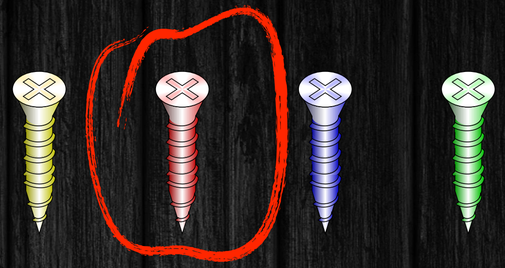


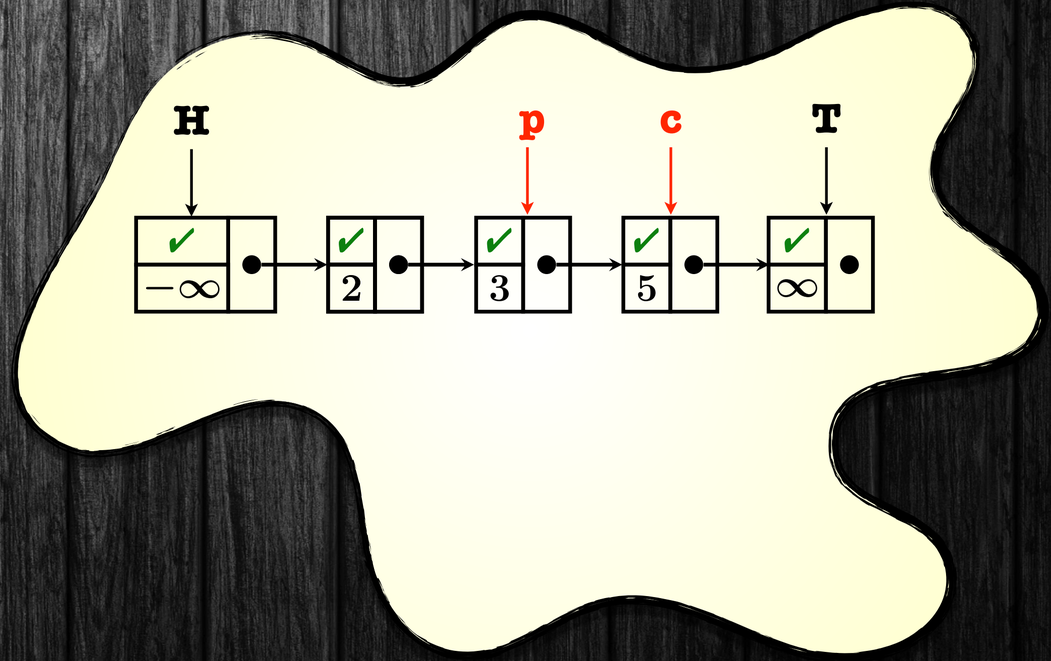
Sets Optimistic Synchronization

```
rmv(k):
Node p, c
1  while (true)
2    p = H
3    c = p.next
4    while (c.key < k)
5      p = c
6      c = c.next
7    lock(p)
8    lock(c)
9    if (validate (p,c))
10     if (c.key=k)
11       p.next = c.next
12       return true
13     else
14       return false
15     unlock p
16     unlock c
17     exit
18   else
19     unlock p
20     unlock c
```



rmv(5)



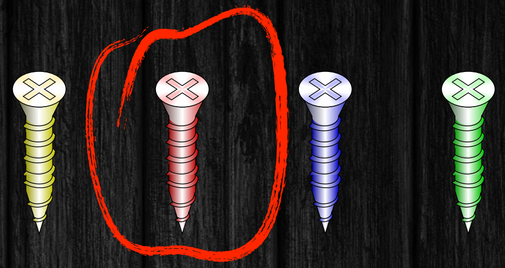


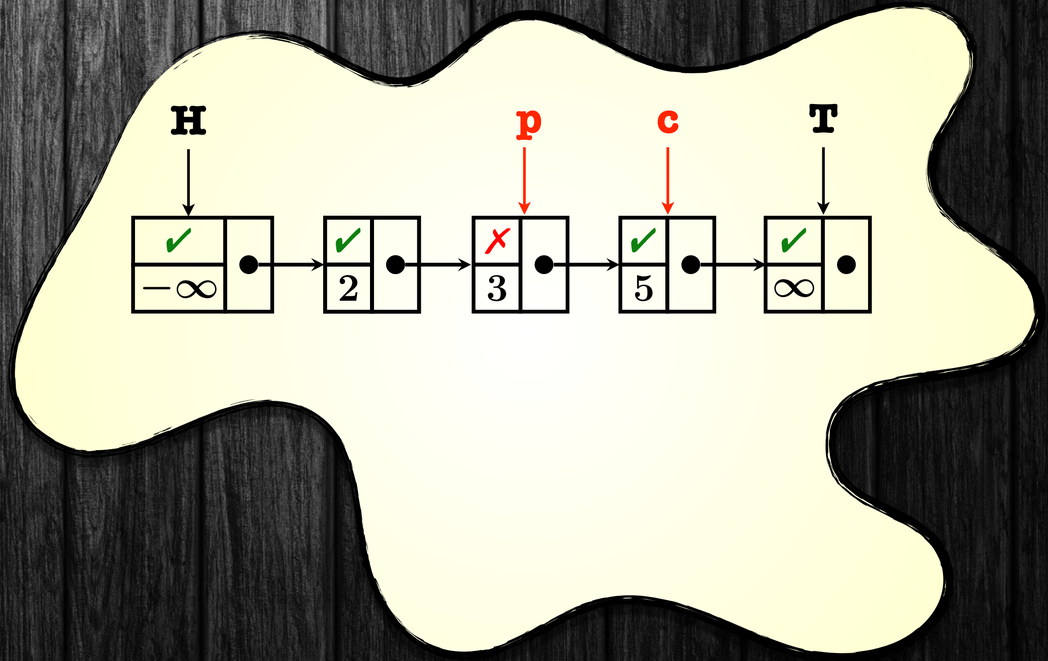
Sets Optimistic Synchronization

```
rmv(k):
Node p, c
1  while (true)
2    p = H
3    c = p.next
4    while (c.key < k)
5      p = c
6      c = c.next
7    lock(p)
8    lock(c)
9    if (validate (p,c))
10     if (c.key=k)
11       p.next = c.next
12       return true
13     else
14       return false
15     unlock p
16     unlock c
17     exit
18   else
19     unlock p
20     unlock c
```



rmv(5)



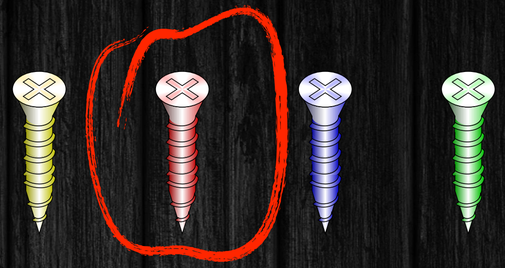


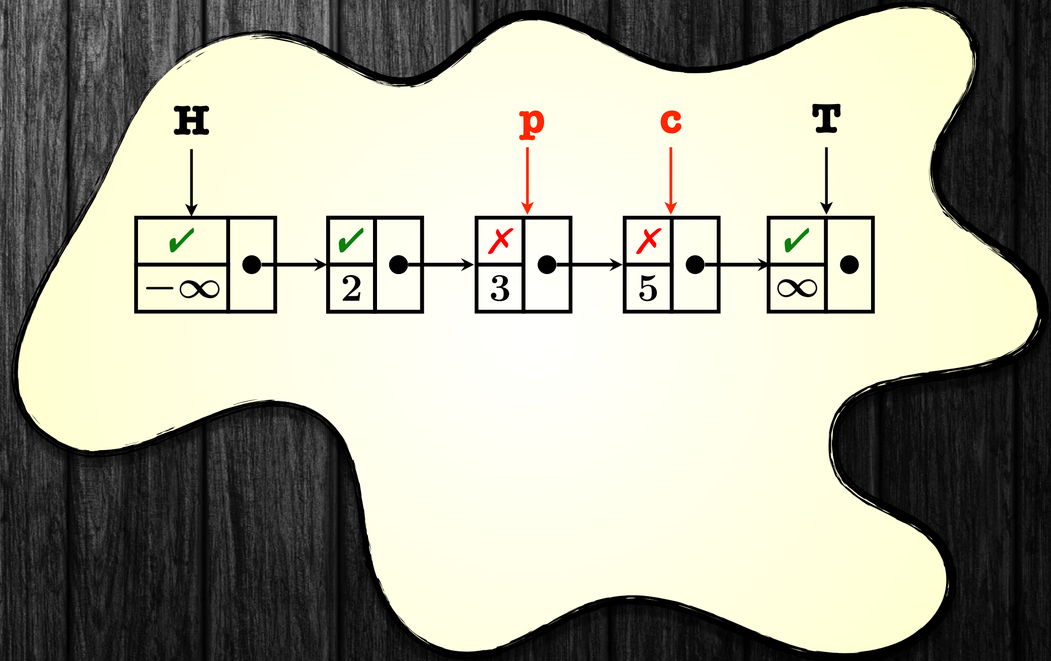
Sets | Optimistic Synchronization

```
rmv(k):
Node p, c
1  while (true)
2    p = H
3    c = p.next
4    while (c.key < k)
5      p = c
6      c = c.next
7    lock(p)
8    lock(c)
9    if (validate (p,c))
10     if (c.key=k)
11       p.next = c.next
12       return true
13     else
14       return false
15     unlock p
16     unlock c
17     exit
18   else
19     unlock p
20     unlock c
```

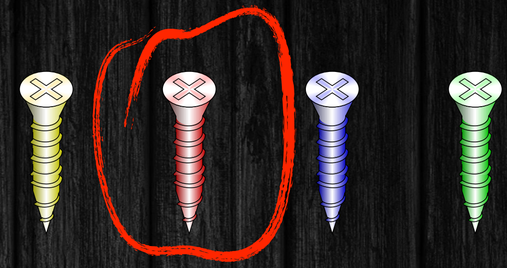


rmv(5)





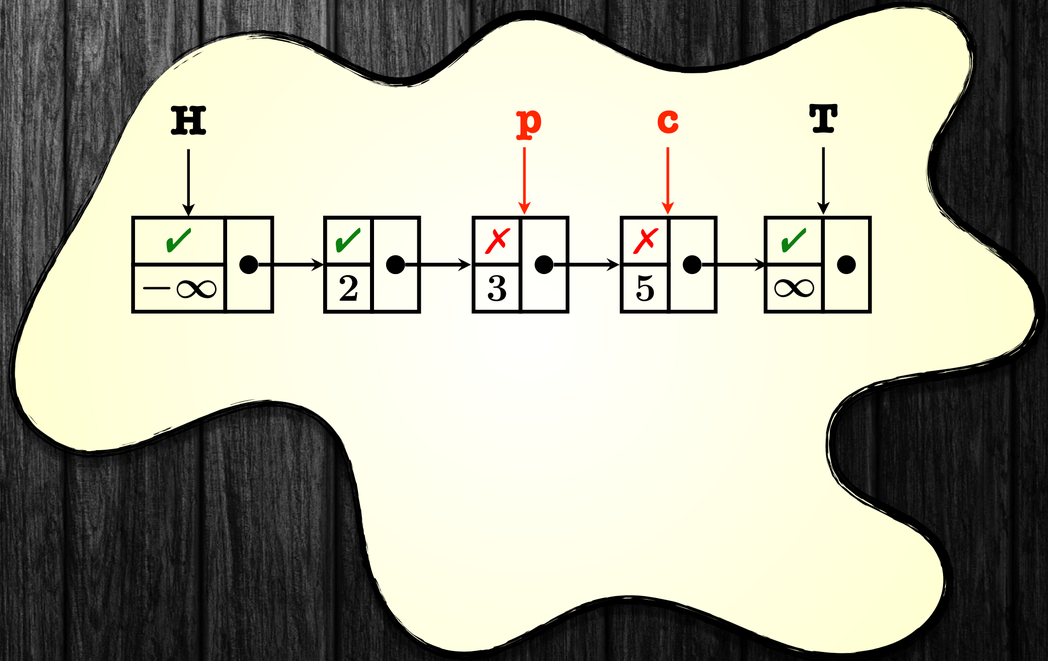
Sets | Optimistic Synchronization



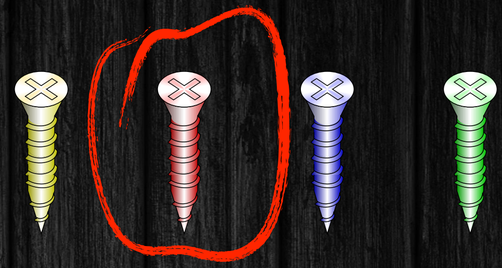
```
rmv(k):
Node p, c
 1  while (true)
 2    p = H
 3    c = p.next
 4    while (c.key < k)
 5      p = c
 6      c = c.next
 7    lock(p)
 8    lock(c)
 9    if (validate (p,c))
10      if (c.key=k)
11        p.next = c.next
12        return true
13      else
14        return false
15      unlock p
16      unlock c
17      exit
18    else
19      unlock p
20      unlock c
```



rmv(5)



Sets | Optimistic Synchronization



```
rmv(k):
Node p, c
1  while (true)
2    p = H
3    c = p.next
4    while (c.key < k)
5      p = c
6      c = c.next
7    lock(p)
8    lock(c)
9    if (validate (p,c))
10     if (c.key=k)
11       p.next = c.next
12       return true
13     else
14       return false
15     unlock p
16     unlock c
17     exit
18   else
19     unlock p
20     unlock c
```

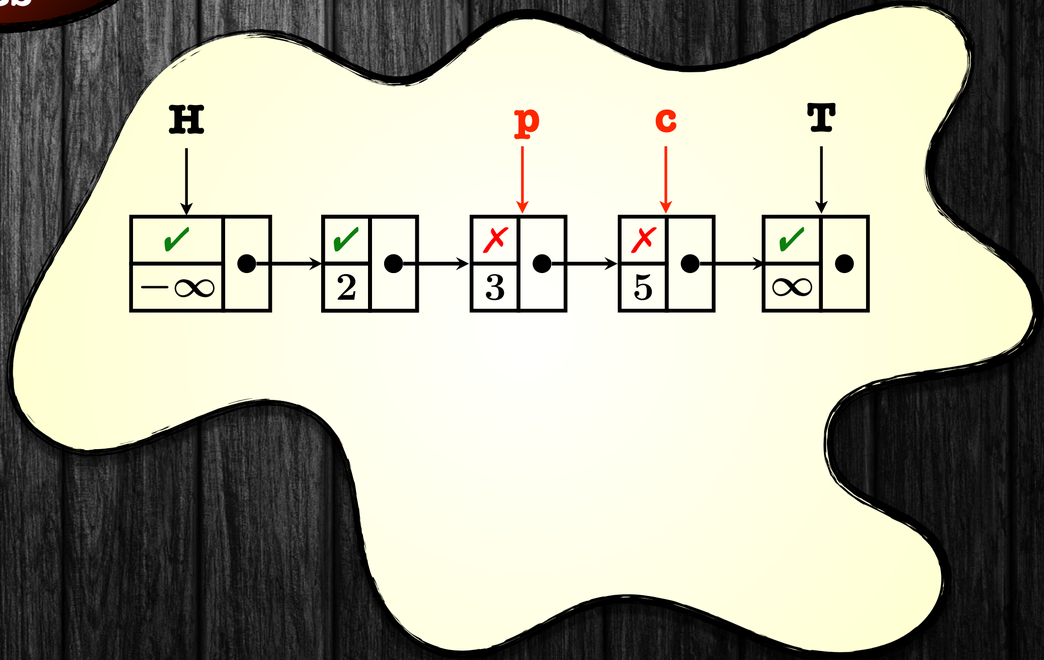


rmv(5)

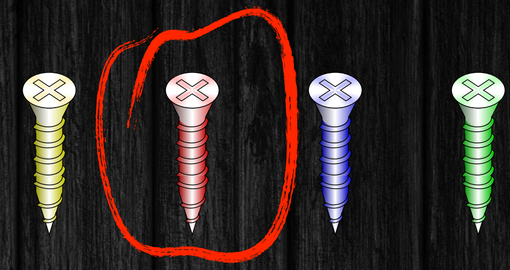
validation

check whether the two cells are still in the list

assume yes



Sets | Optimistic Synchronization



```
rmv(k):
Node p, c
1  while (true)
2    p = H
3    c = p.next
4    while (c.key < k)
5      p = c
6      c = c.next
7    lock(p)
8    lock(c)
9    if (validate (p,c))
10     if (c.key=k)
11       p.next = c.next
12       return true
13     else
14       return false
15     unlock p
16     unlock c
17     exit
18   else
19     unlock p
20     unlock c
```

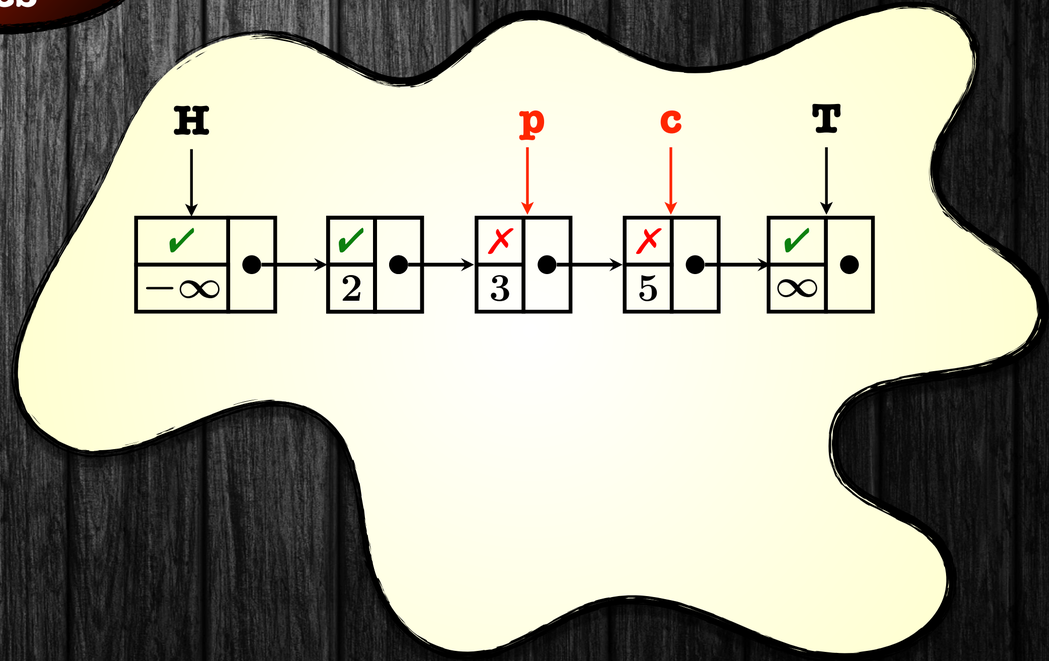


rmv(5)

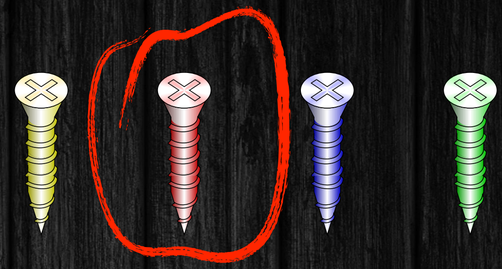
validation

check whether the two cells are still in the list

assume yes



Sets | Optimistic Synchronization



```
rmv(k):
Node p, c
1  while (true)
2    p = H
3    c = p.next
4    while (c.key < k)
5      p = c
6      c = c.next
7    lock(p)
8    lock(c)
9    if (validate (p,c))
10     if (c.key=k)
11       p.next = c.next
12       return true
13     else
14       return false
15     unlock p
16     unlock c
17     exit
18   else
19     unlock p
20     unlock c
```

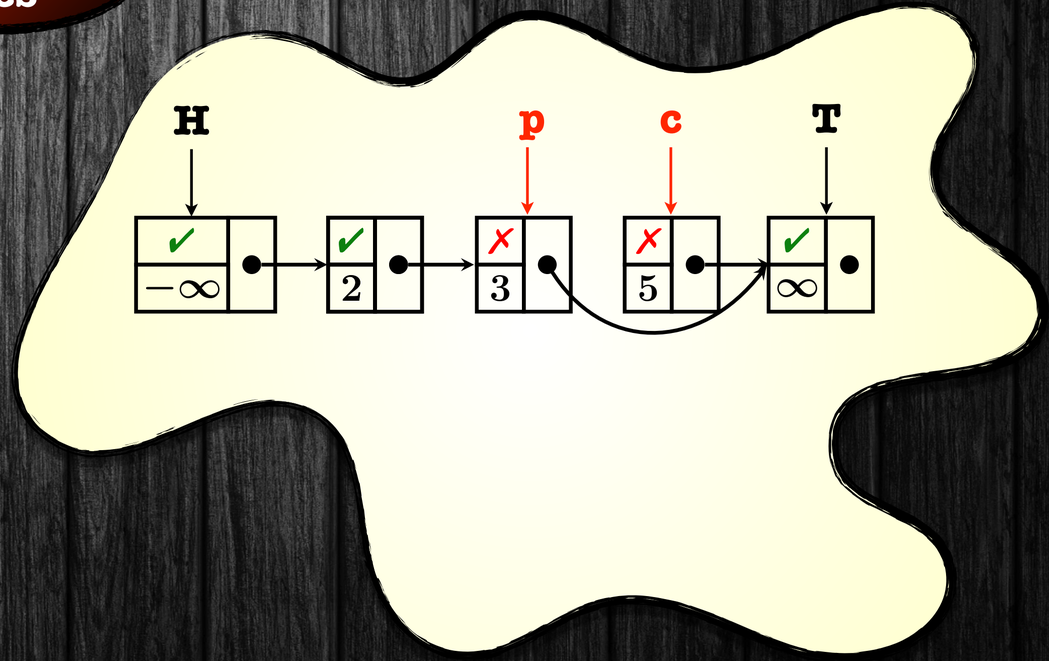


rmv(5)

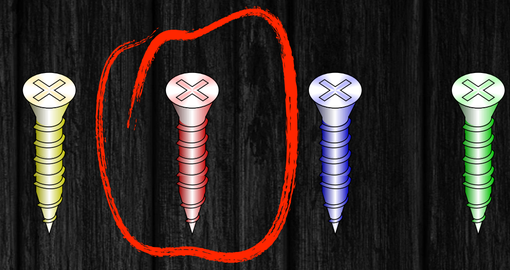
validation

check whether the two cells are still in the list

assume yes



Sets | Optimistic Synchronization



```
rmv(k):
Node p, c
1  while (true)
2    p = H
3    c = p.next
4    while (c.key < k)
5      p = c
6      c = c.next
7    lock(p)
8    lock(c)
9    if (validate (p,c))
10     if (c.key=k)
11       p.next = c.next
12       return true
13     else
14       return false
15     unlock p
16     unlock c
17     exit
18   else
19     unlock p
20     unlock c
```

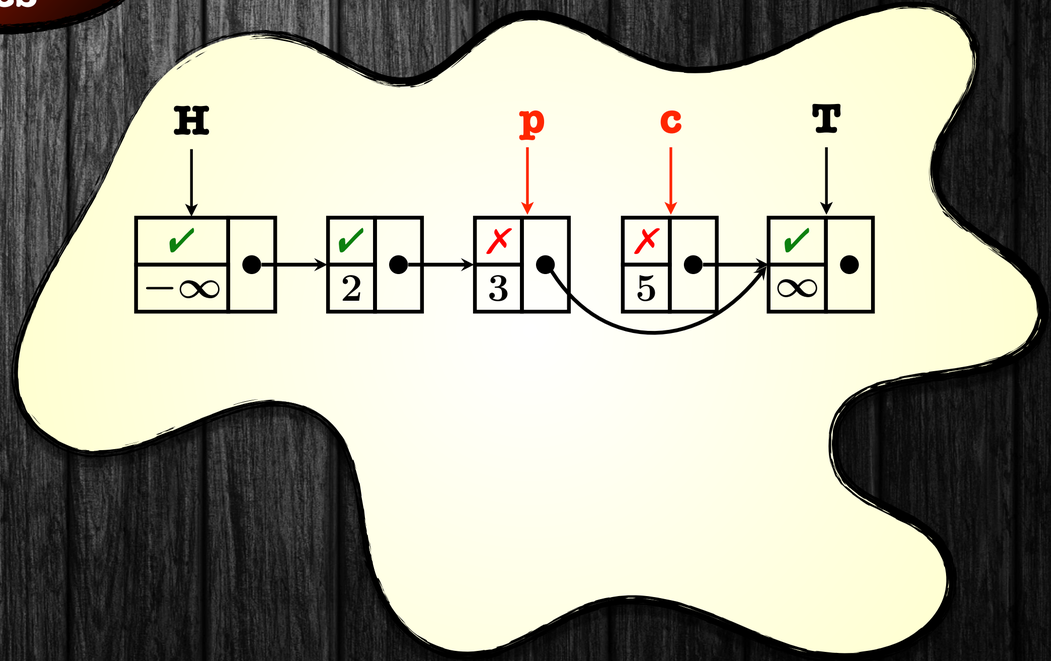


rmv(5)

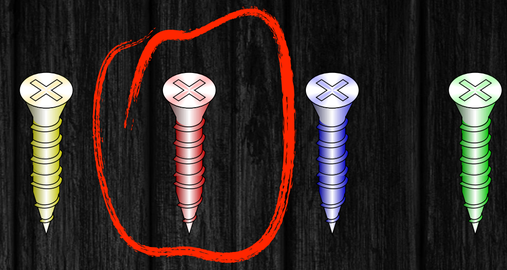
validation

check whether the two cells are still in the list

assume yes



Sets | Optimistic Synchronization



```
rmv(k):
Node p, c
1  while (true)
2    p = H
3    c = p.next
4    while (c.key < k)
5      p = c
6      c = c.next
7    lock(p)
8    lock(c)
9    if (validate (p,c))
10     if (c.key=k)
11       p.next = c.next
12       return true
13     else
14       return false
15     unlock p
16     unlock c
17     exit
18   else
19     unlock p
20     unlock c
```

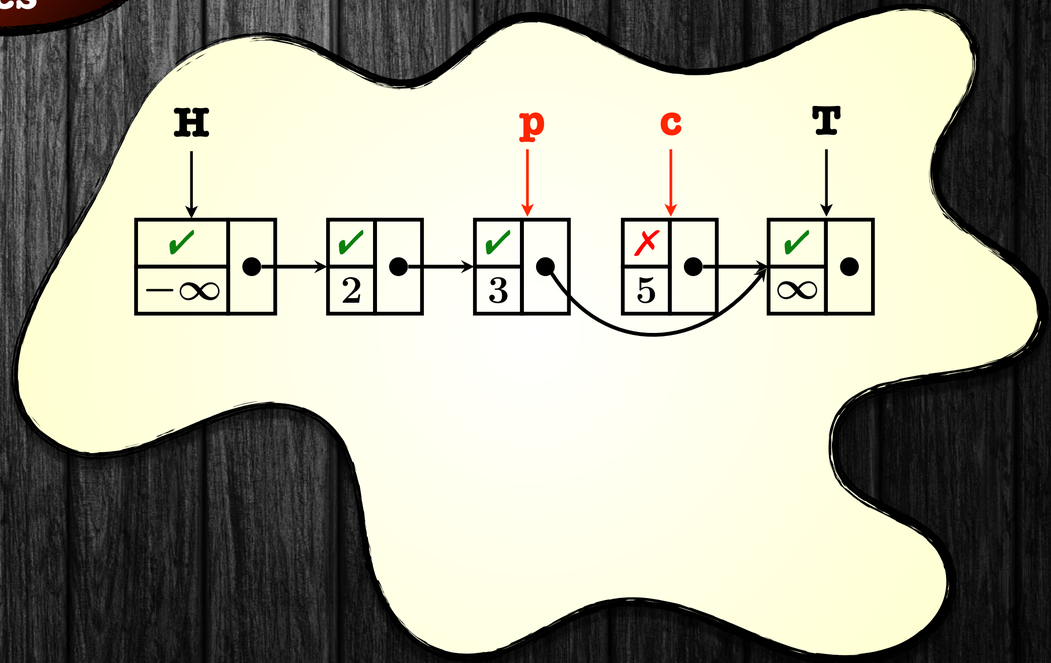


rmv(5)

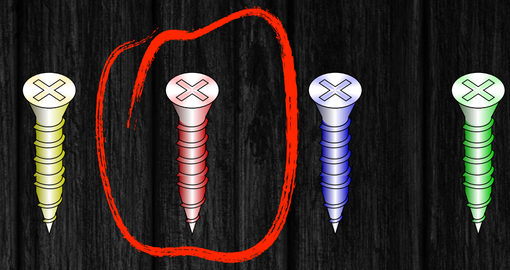
validation

check whether the two cells are still in the list

assume yes



Sets | Optimistic Synchronization



```
rmv(k):
Node p, c
1  while (true)
2    p = H
3    c = p.next
4    while (c.key < k)
5      p = c
6      c = c.next
7    lock(p)
8    lock(c)
9    if (validate (p,c))
10     if (c.key=k)
11       p.next = c.next
12       return true
13     else
14       return false
15     unlock p
16     unlock c
17     exit
18   else
19     unlock p
20     unlock c
```

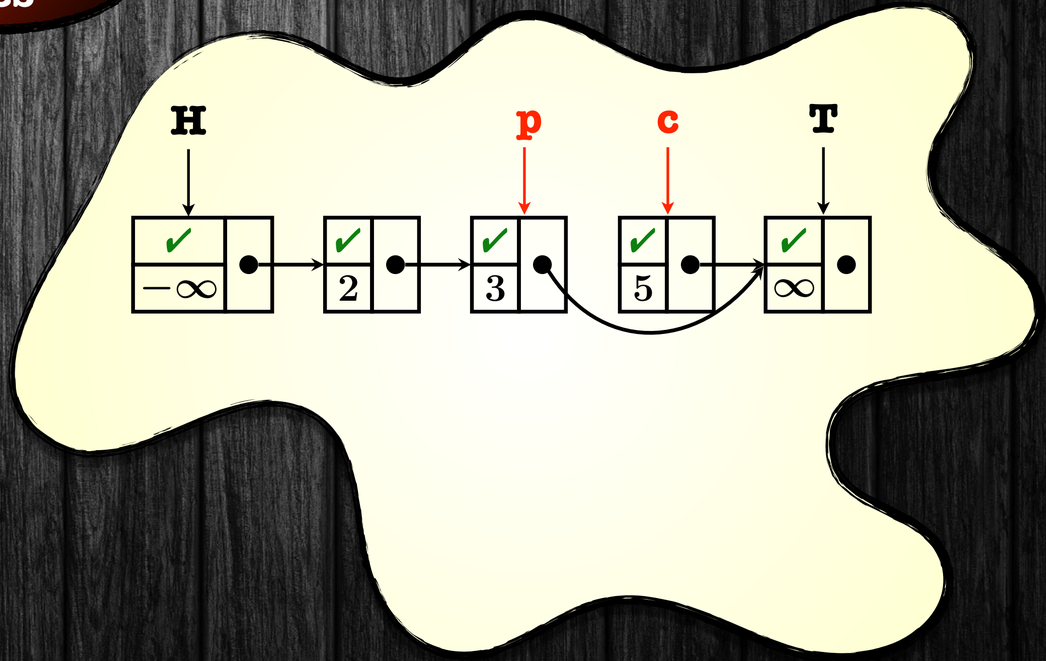


rmv(5)

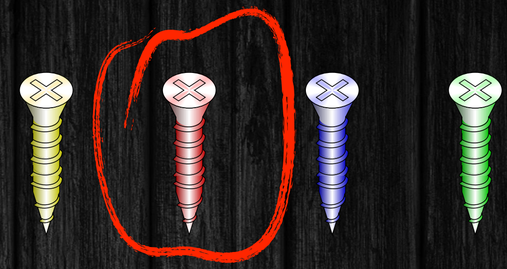
validation

check whether the two cells are still in the list

assume yes



Sets | Optimistic Synchronization



```
rmv(k):
Node p, c
1  while (true)
2    p = H
3    c = p.next
4    while (c.key < k)
5      p = c
6      c = c.next
7    lock(p)
8    lock(c)
9    if (validate (p,c))
10     if (c.key=k)
11       p.next = c.next
12       return true
13     else
14       return false
15     unlock p
16     unlock c
17     exit
18   else
19     unlock p
20     unlock c
```

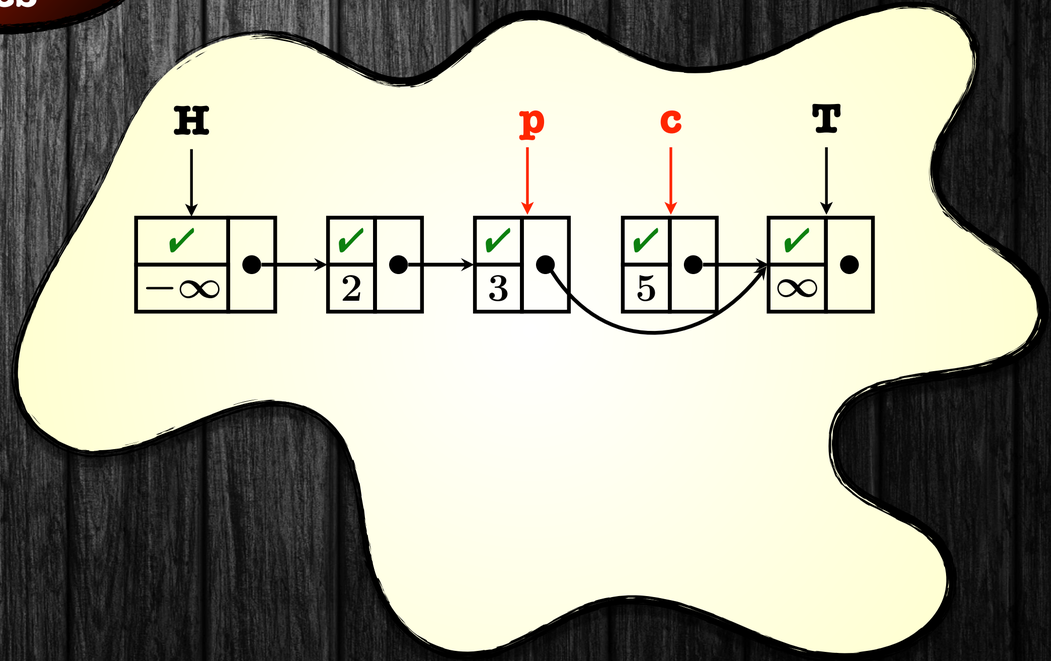


rmv(5)

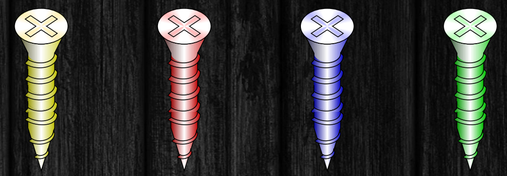
validation

check whether the two cells are still in the list

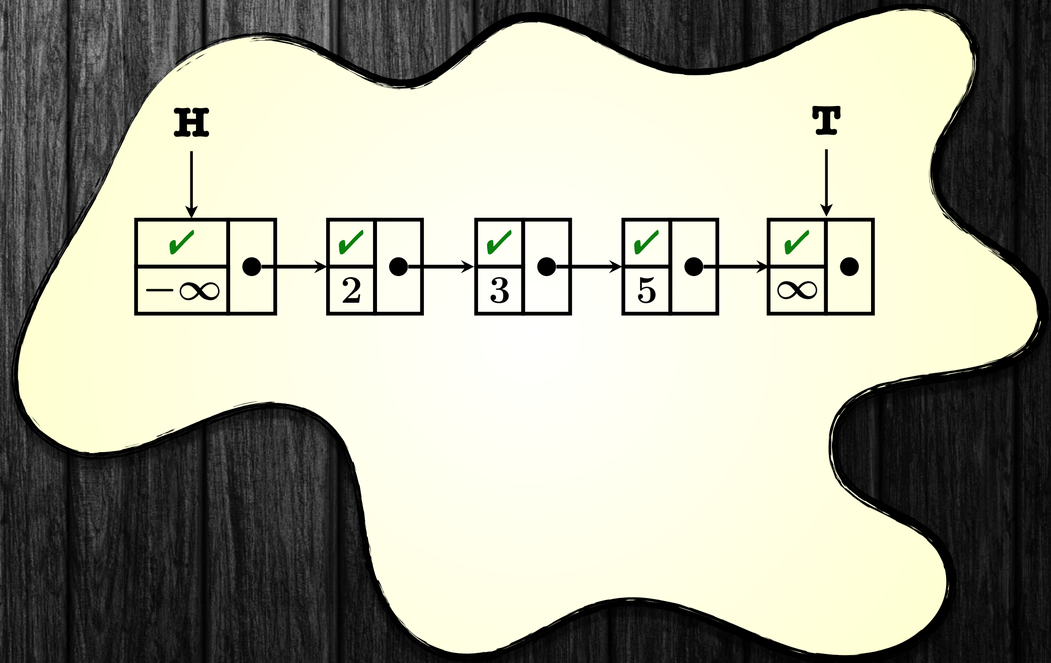
assume yes



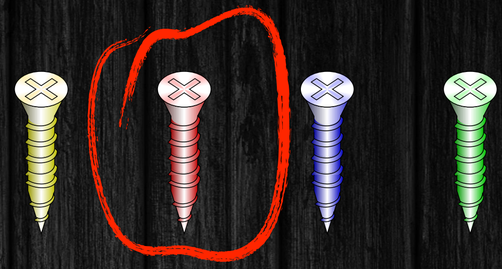
Sets | Optimistic Synchronization



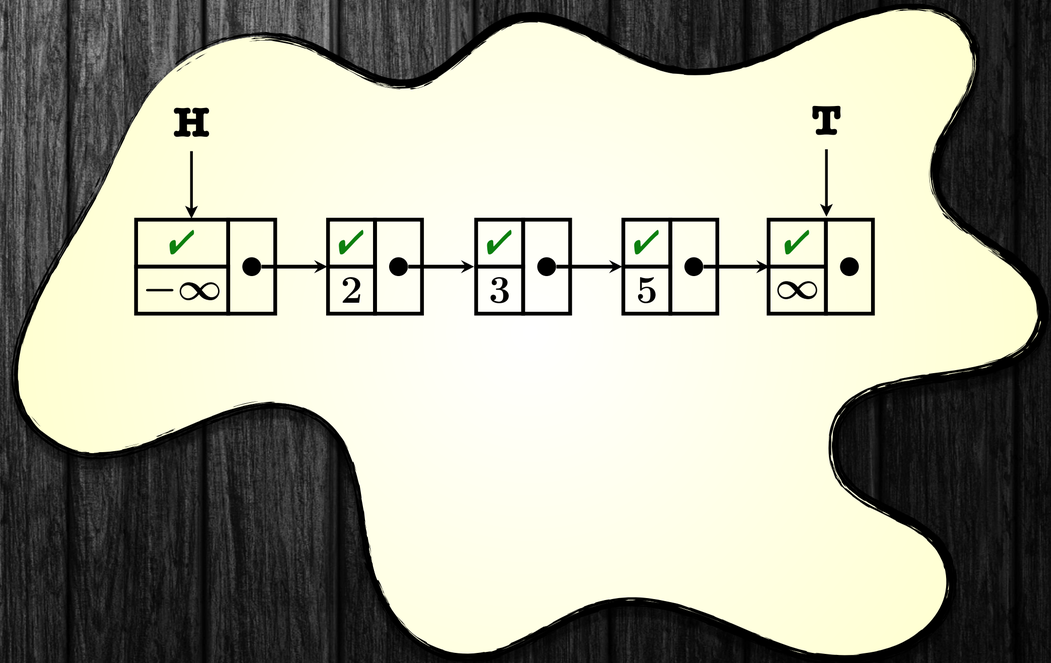
```
rmv(k):
Node p, c
 1  while (true)
 2    p = H
 3    c = p.next
 4    while (c.key < k)
 5      p = c
 6      c = c.next
 7    lock(p)
 8    lock(c)
 9    if (validate (p,c))
10      if (c.key=k)
11        p.next = c.next
12        return true
13      else
14        return false
15      unlock p
16      unlock c
17      exit
18    else
19      unlock p
20      unlock c
```



Sets | Optimistic Synchronization



```
rmv(k):
Node p, c
 1  while (true)
 2    p = H
 3    c = p.next
 4    while (c.key < k)
 5      p = c
 6      c = c.next
 7    lock(p)
 8    lock(c)
 9    if (validate (p,c))
10      if (c.key=k)
11        p.next = c.next
12        return true
13      else
14        return false
15      unlock p
16      unlock c
17      exit
18    else
19      unlock p
20      unlock c
```

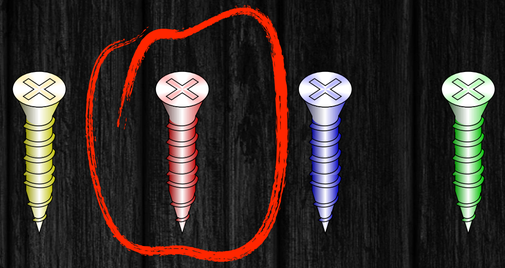


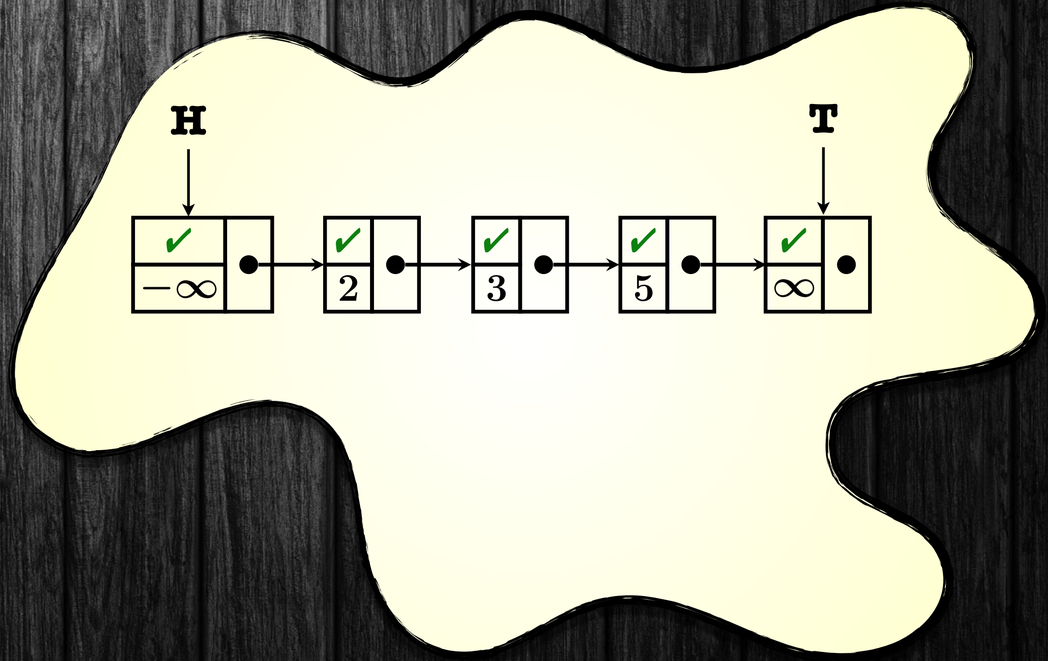
Sets | Optimistic Synchronization

```
rmv(k):
Node p, c
 1  while (true)
 2    p = H
 3    c = p.next
 4    while (c.key < k)
 5      p = c
 6      c = c.next
 7    lock(p)
 8    lock(c)
 9    if (validate (p,c))
10      if (c.key=k)
11        p.next = c.next
12        return true
13      else
14        return false
15      unlock p
16      unlock c
17      exit
18    else
19      unlock p
20      unlock c
```

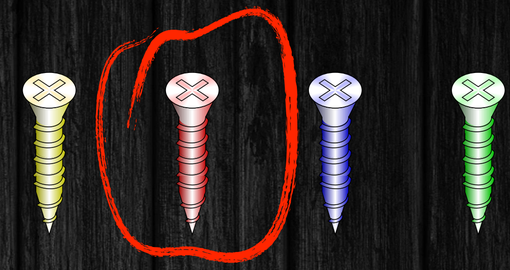


rmv(5)





Sets | Optimistic Synchronization



```
rmv(k):
Node p, c
1  while (true)
2    p = H
3    c = p.next
4    while (c.key < k)
5      p = c
6      c = c.next
7    lock(p)
8    lock(c)
9    if (validate (p,c))
10     if (c.key=k)
11       p.next = c.next
12       return true
13     else
14       return false
15     unlock p
16     unlock c
17     exit
18   else
19     unlock p
20     unlock c
```

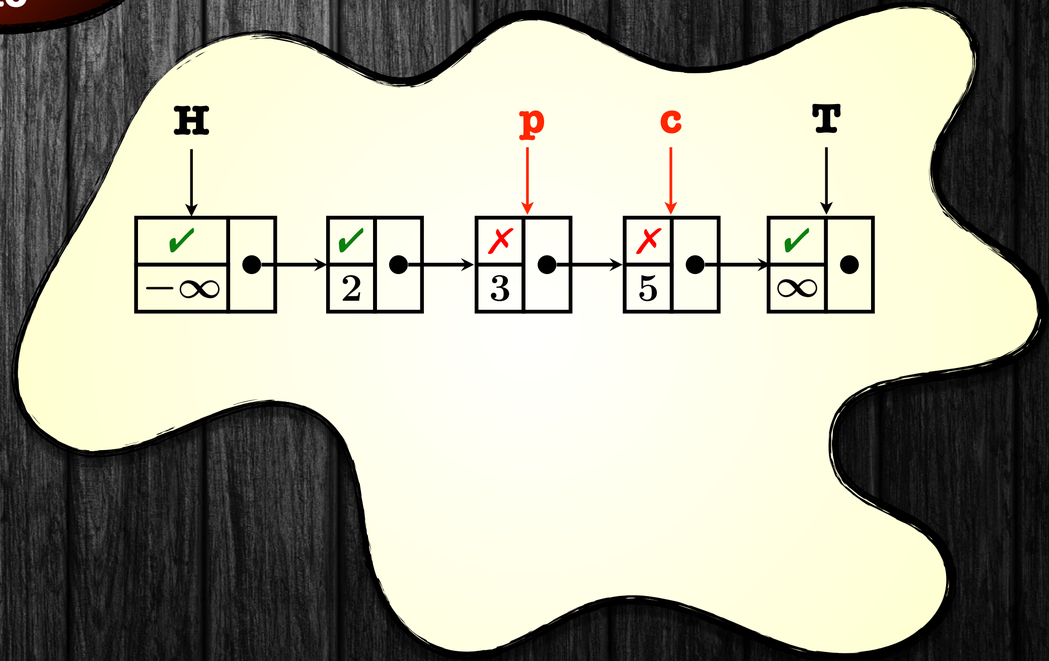


rmv(5)

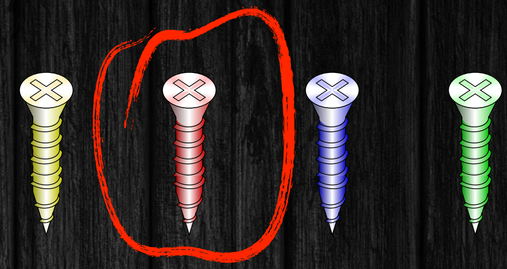
validation

check whether the two cells are still in the list

assume no



Sets | Optimistic Synchronization



```
rmv(k):
Node p, c
1  while (true)
2    p = H
3    c = p.next
4    while (c.key < k)
5      p = c
6      c = c.next
7    lock(p)
8    lock(c)
9    if (validate (p,c))
10     if (c.key=k)
11       p.next = c.next
12       return true
13     else
14       return false
15     unlock p
16     unlock c
17     exit
18   else ⟵
19     unlock p
20     unlock c
```

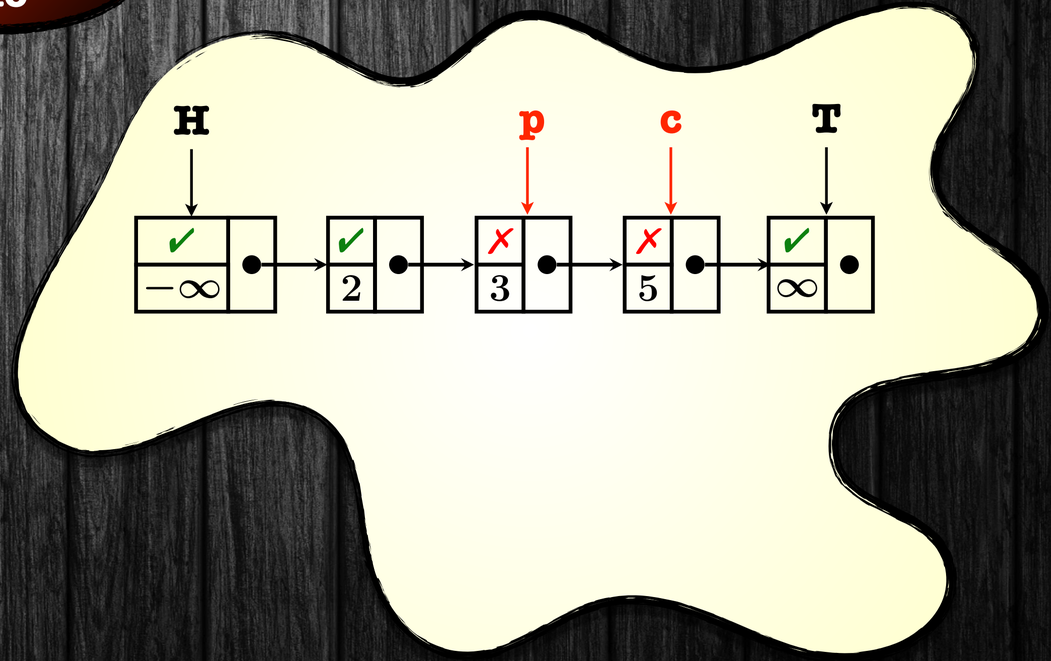


rmv(5)

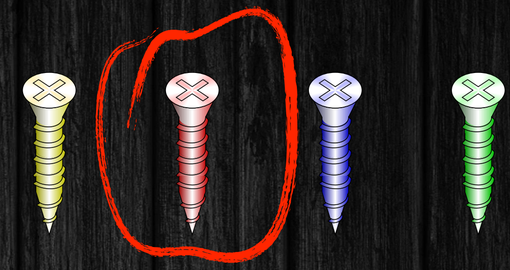
validation

check whether the two cells are still in the list

assume no



Sets | Optimistic Synchronization



```
rmv(k):
Node p, c
1  while (true)
2    p = H
3    c = p.next
4    while (c.key < k)
5      p = c
6      c = c.next
7    lock(p)
8    lock(c)
9    if (validate (p,c))
10     if (c.key=k)
11       p.next = c.next
12       return true
13     else
14       return false
15     unlock p
16     unlock c
17     exit
18   else
19     unlock p
20     unlock c
```

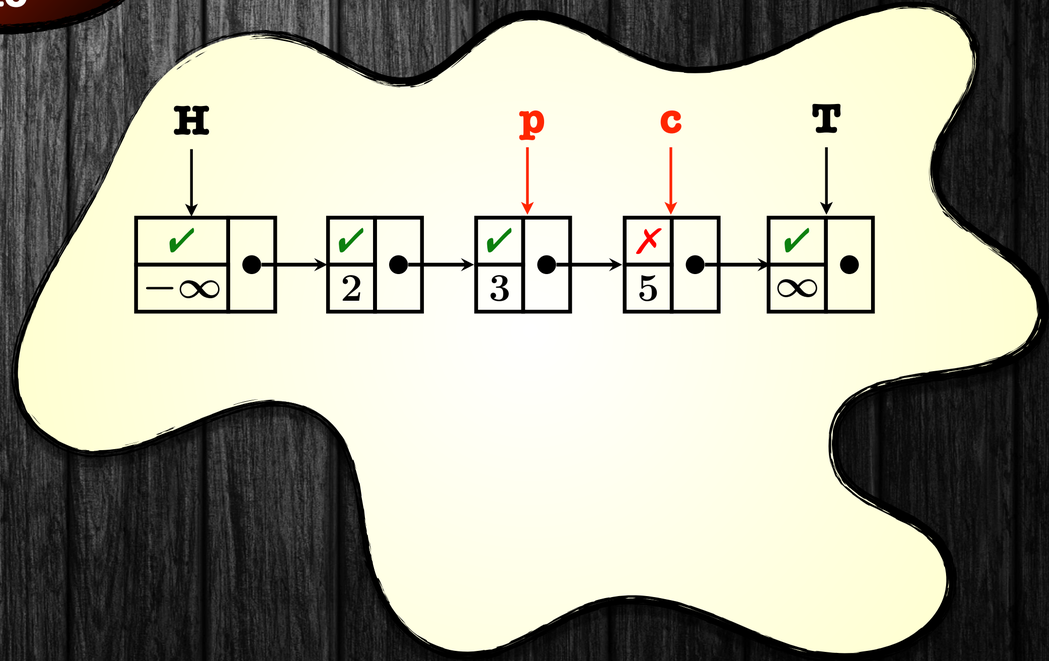


rmv(5)

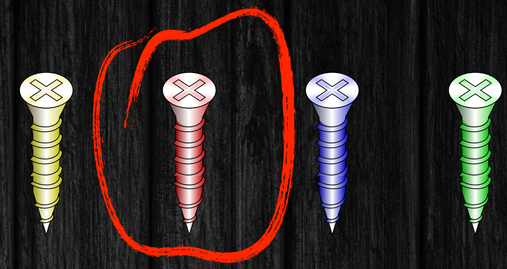
validation

check whether the two cells are still in the list

assume no



Sets | Optimistic Synchronization



```
rmv(k):
Node p, c
 1  while (true)
 2    p = H
 3    c = p.next
 4    while (c.key < k)
 5      p = c
 6      c = c.next
 7    lock(p)
 8    lock(c)
 9    if (validate (p,c))
10      if (c.key=k)
11        p.next = c.next
12        return true
13      else
14        return false
15      unlock p
16      unlock c
17      exit
18    else ⟵
19      unlock p
20      unlock c
```

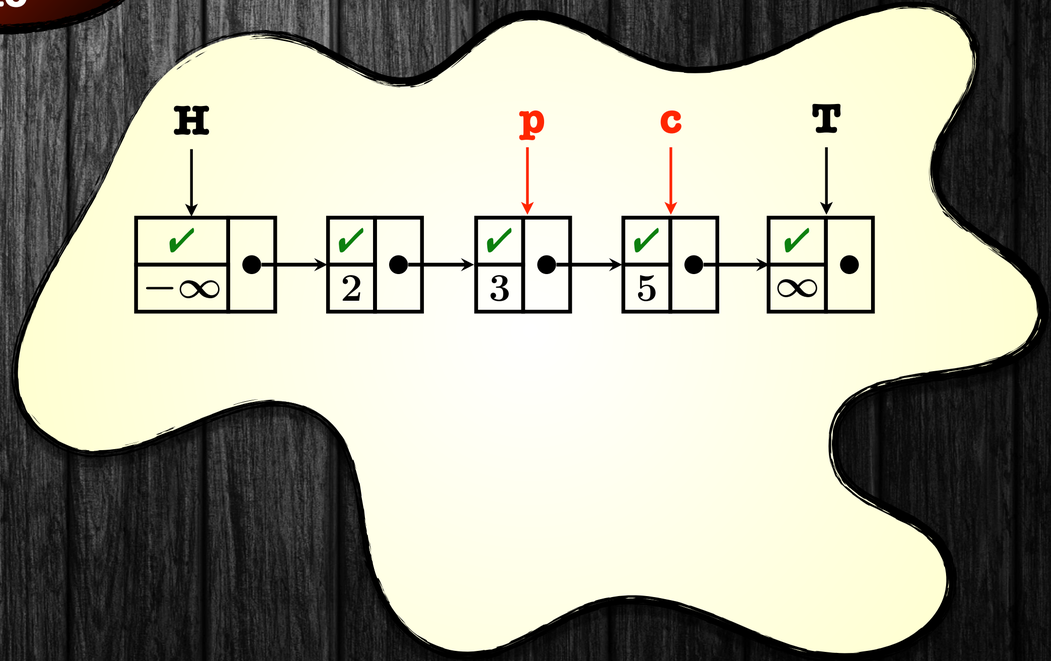


rmv(5)

validation

check whether the two cells are still in the list

assume no

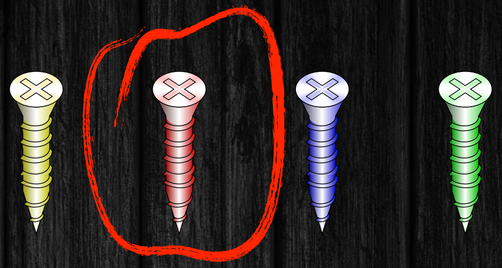


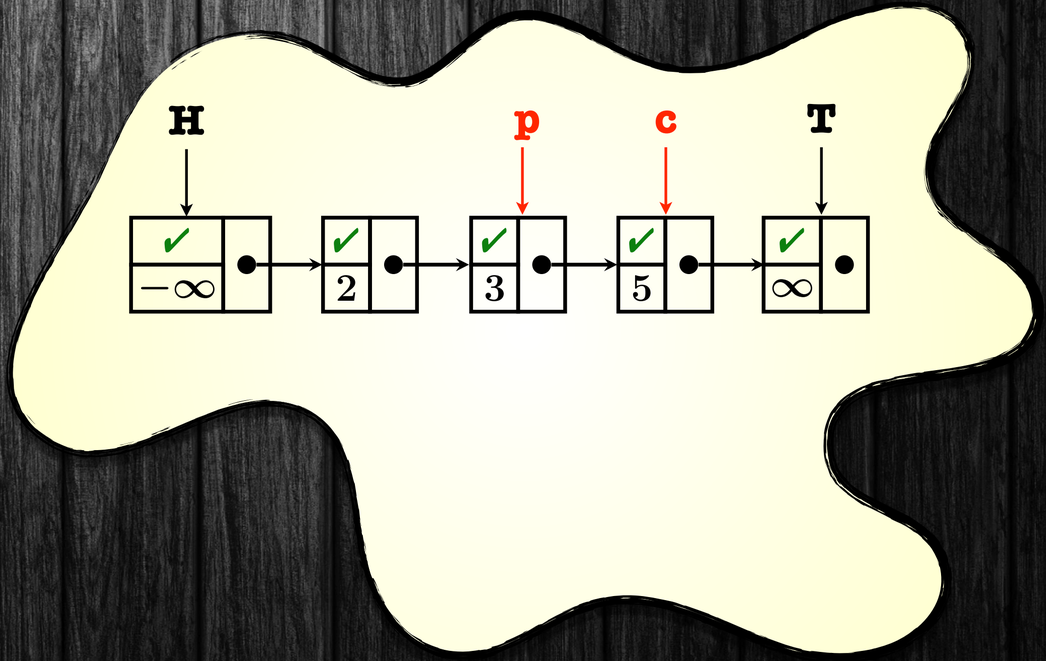
Sets Optimistic Synchronization

```
rmv(k):
Node p, c
1  while (true)
2    p = H
3    c = p.next
4    while (c.key < k)
5      p = c
6      c = c.next
7    lock(p)
8    lock(c)
9    if (validate (p,c))
10     if (c.key=k)
11       p.next = c.next
12       return true
13     else
14       return false
15     unlock p
16     unlock c
17     exit
18   else
19     unlock p
20     unlock c
```



rmv(5)

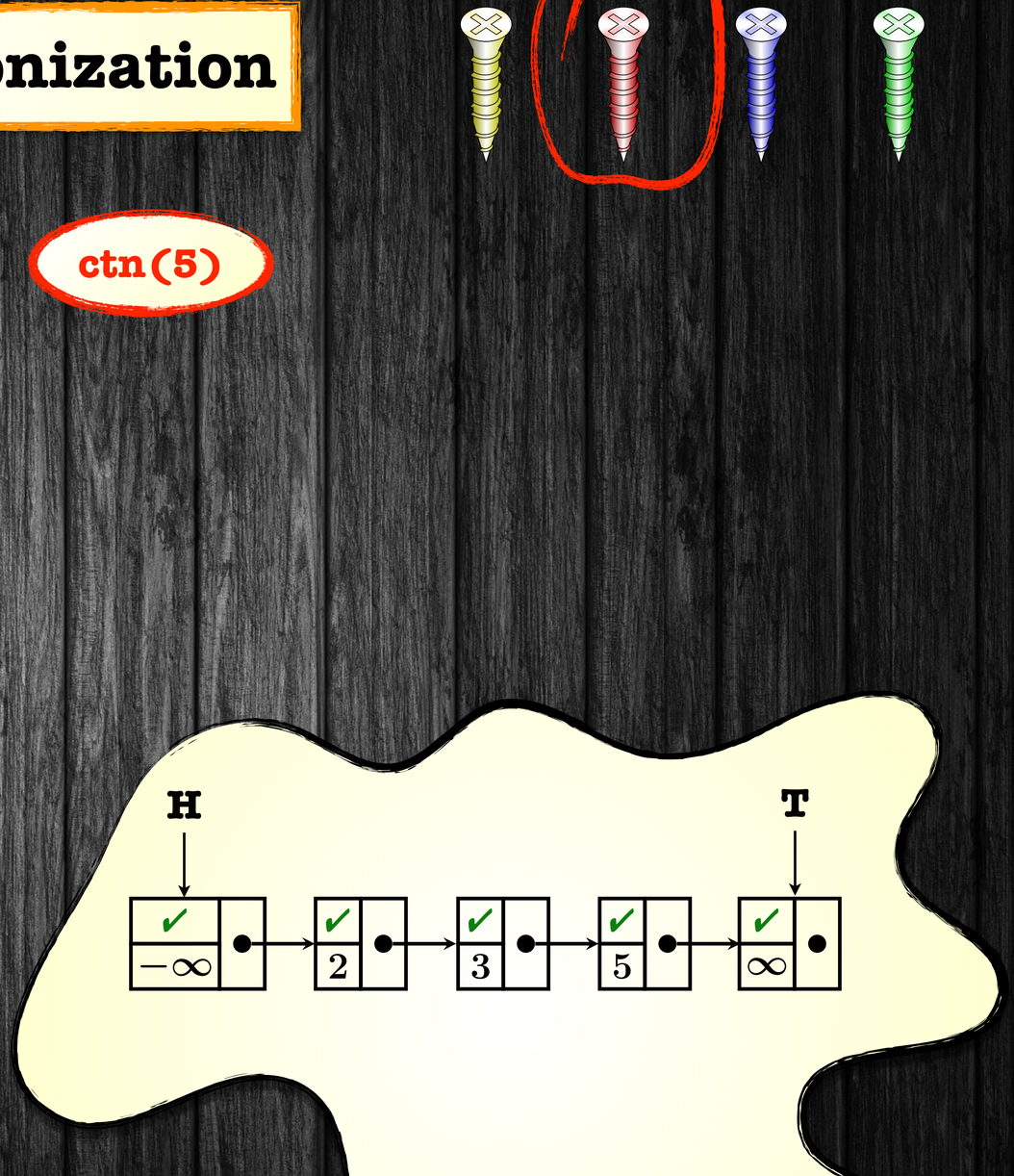




Sets | Optimistic Synchronization

```
ctn(k):
Node p, c
1  while (true)
2    p = H
3    c = p.next
4    while (c.key < k)
5      p = c
6      c = c.next
7    lock(p)
8    lock(c)
9    if (validate (p,c))
10     if (c.key=k)
11       return true
12     else
13       return false
16     unlock p
17     unlock c
18     exit
19   else
20     unlock p
21     unlock c
```

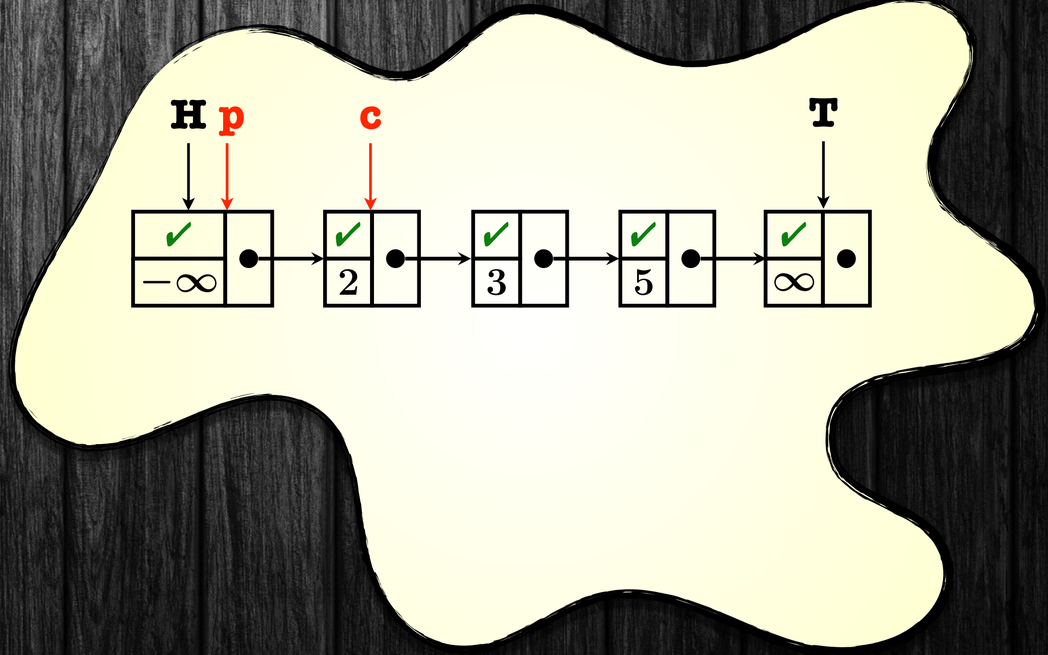
ctn(5)



Sets | Optimistic Synchronization

```
ctn(k):
Node p, c
1  while (true)
2    p = H
3    c = p.next
4    while (c.key < k)
5      p = c
6      c = c.next
7    lock(p)
8    lock(c)
9    if (validate (p,c))
10     if (c.key=k)
11       return true
12     else
13       return false
16     unlock p
17     unlock c
18     exit
19   else
20     unlock p
21     unlock c
```

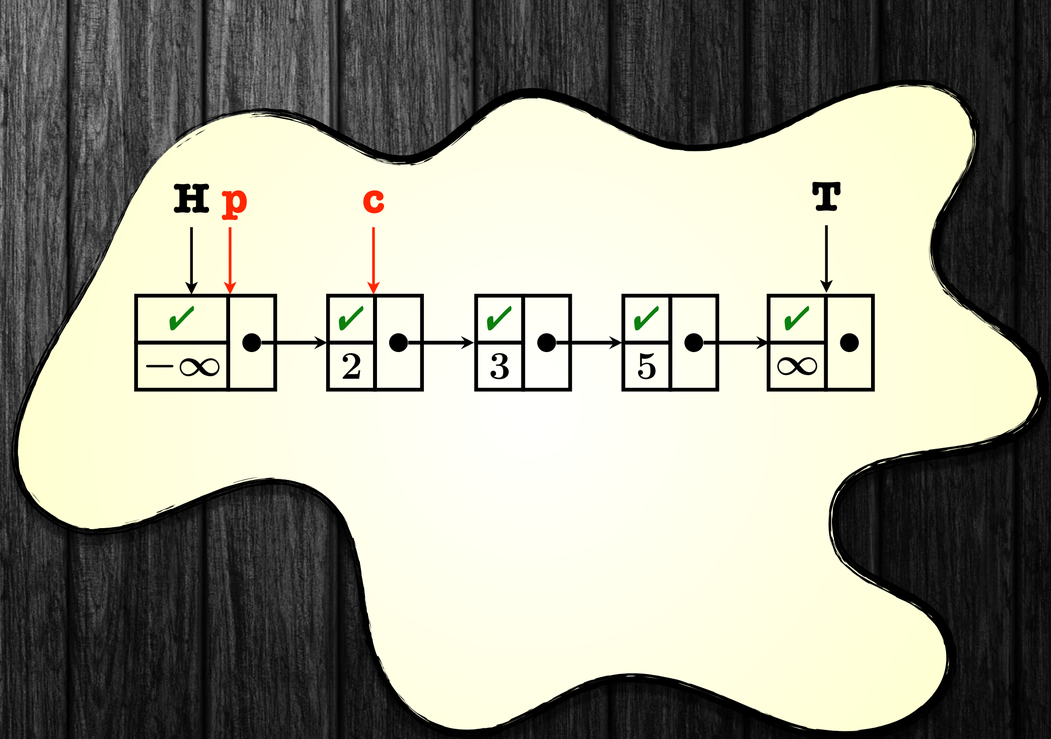
ctn(5)



Sets | Optimistic Synchronization

```
ctn(k):
Node p, c
1  while (true)
2    p = H
3    c = p.next
4    while (c.key < k)
5      p = c
6      c = c.next
7    lock(p)
8    lock(c)
9    if (validate (p,c))
10     if (c.key=k)
11       return true
12     else
13       return false
16     unlock p
17     unlock c
18     exit
19   else
20     unlock p
21     unlock c
```

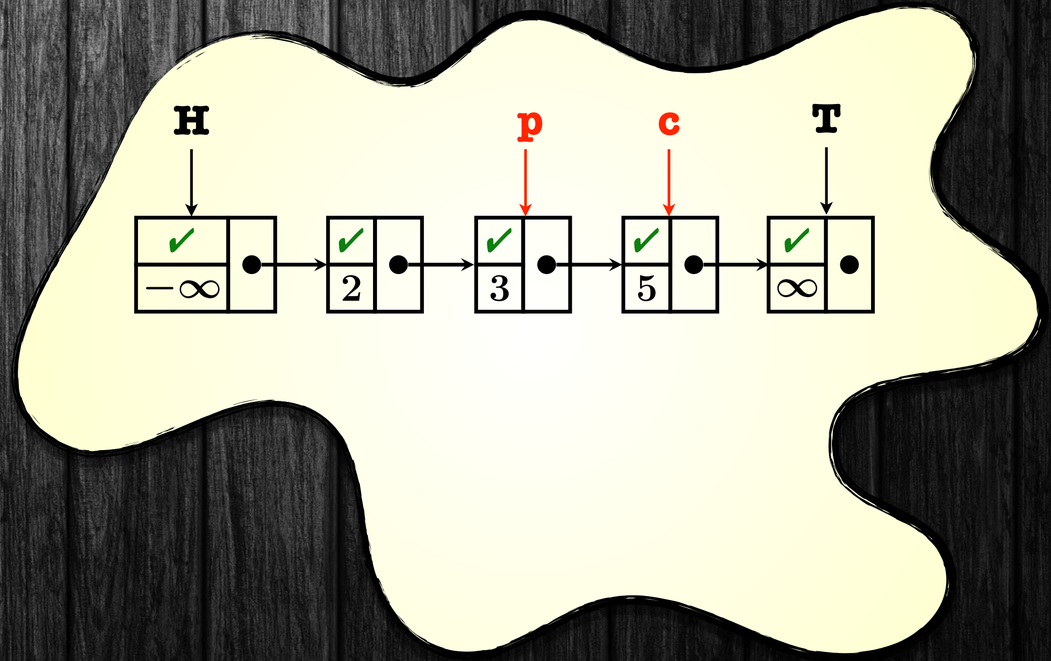
ctn(5)



Sets | Optimistic Synchronization

```
ctn(k):
Node p, c
 1  while (true)
 2    p = H
 3    c = p.next
 4    while (c.key < k)
 5      p = c
 6      c = c.next
 7    lock(p)
 8    lock(c)
 9    if (validate (p,c))
10      if (c.key=k)
11        return true
12      else
13        return false
16      unlock p
17      unlock c
18      exit
19    else
20      unlock p
21      unlock c
```

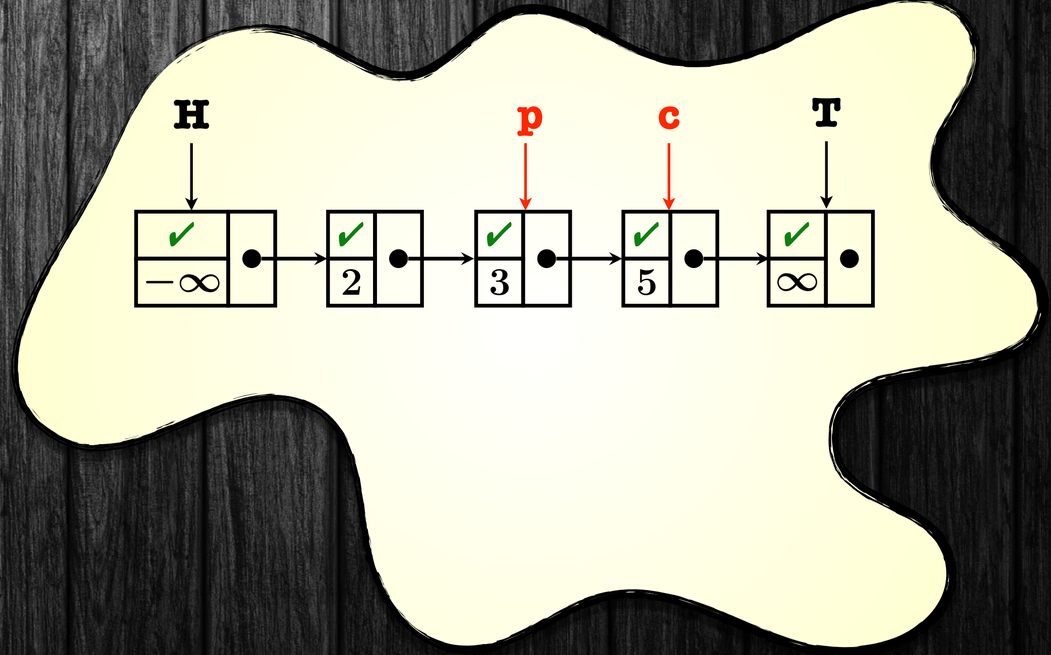
ctn(5)



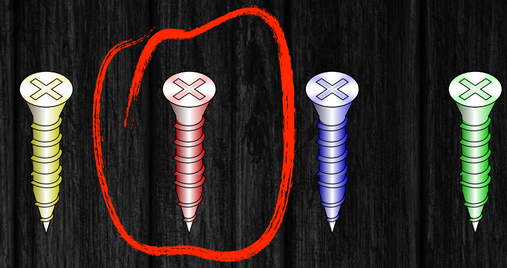
Sets | Optimistic Synchronization

```
ctn(k):
Node p, c
1  while (true)
2    p = H
3    c = p.next
4    while (c.key < k)
5      p = c
6      c = c.next
7    lock(p)
8    lock(c)
9    if (validate (p,c))
10     if (c.key=k)
11       return true
12     else
13       return false
16     unlock p
17     unlock c
18     exit
19   else
20     unlock p
21     unlock c
```

ctn(5)



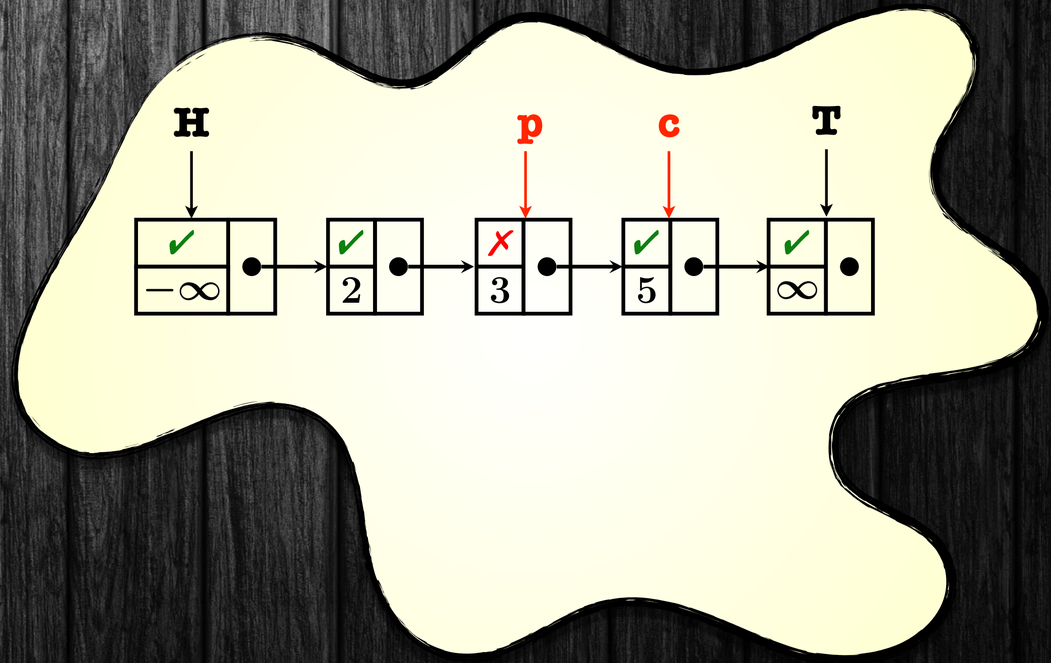
Sets | Optimistic Synchronization



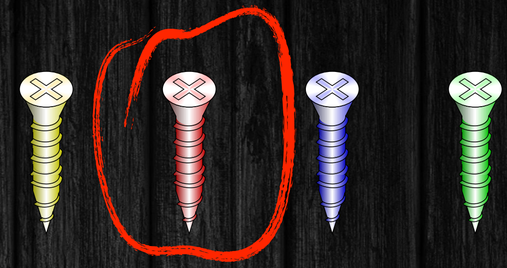
```
ctn(k):
Node p, c
1  while (true)
2    p = H
3    c = p.next
4    while (c.key < k)
5      p = c
6      c = c.next
7    lock(p)
8    lock(c)
9    if (validate (p,c))
10     if (c.key=k)
11       return true
12     else
13       return false
16     unlock p
17     unlock c
18     exit
19   else
20     unlock p
21     unlock c
```



ctn(5)



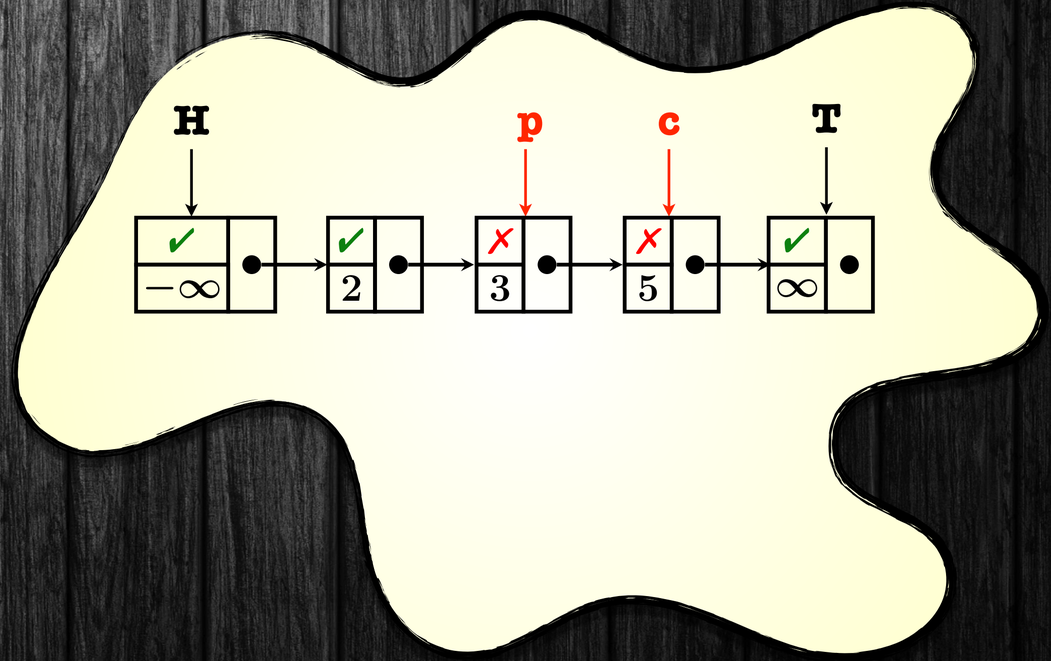
Sets | Optimistic Synchronization



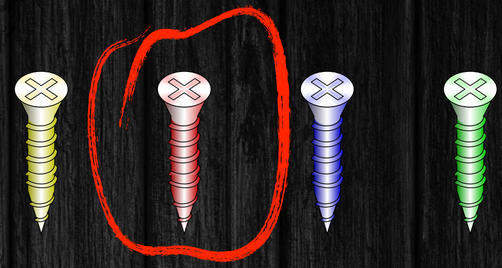
```
ctn(k):
Node p, c
 1  while (true)
 2    p = H
 3    c = p.next
 4    while (c.key < k)
 5      p = c
 6      c = c.next
 7    lock(p)
 8    lock(c)
 9    if (validate (p,c))
10      if (c.key=k)
11        return true
12      else
13        return false
16      unlock p
17      unlock c
18      exit
19    else
20      unlock p
21      unlock c
```



ctn(5)



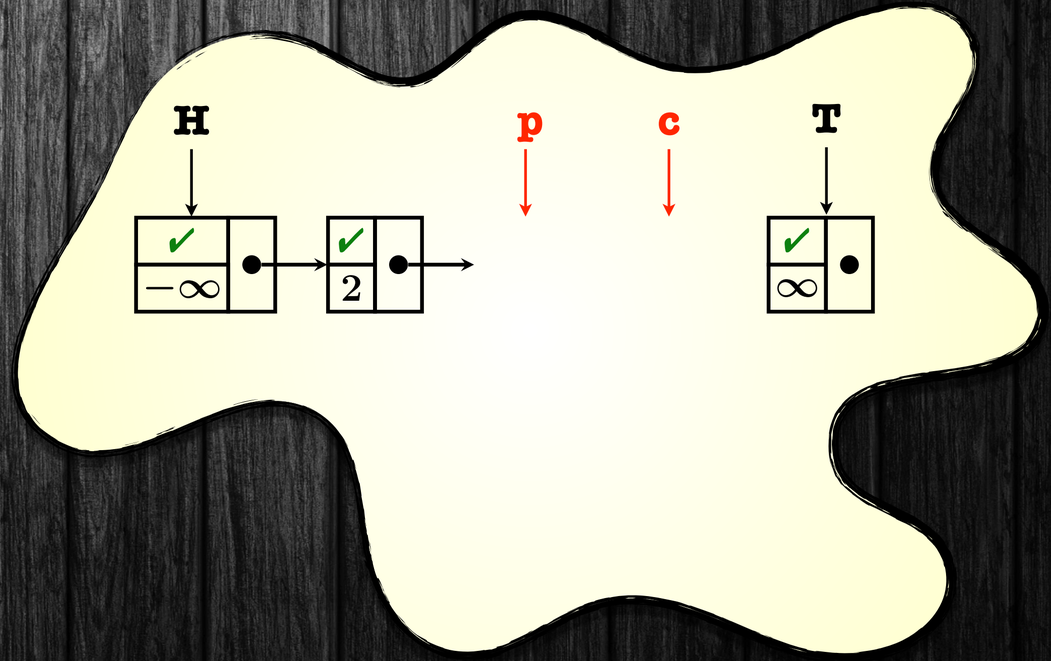
Sets | Optimistic Synchronization



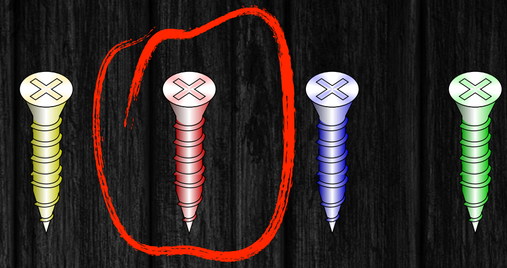
```
ctn(k):
Node p, c
 1  while (true)
 2    p = H
 3    c = p.next
 4    while (c.key < k)
 5      p = c
 6      c = c.next
 7    lock(p)
 8    lock(c)
 9    if (validate (p,c))
10      if (c.key=k)
11        return true
12      else
13        return false
16      unlock p
17      unlock c
18      exit
19    else
20      unlock p
21      unlock c
```



ctn(5)



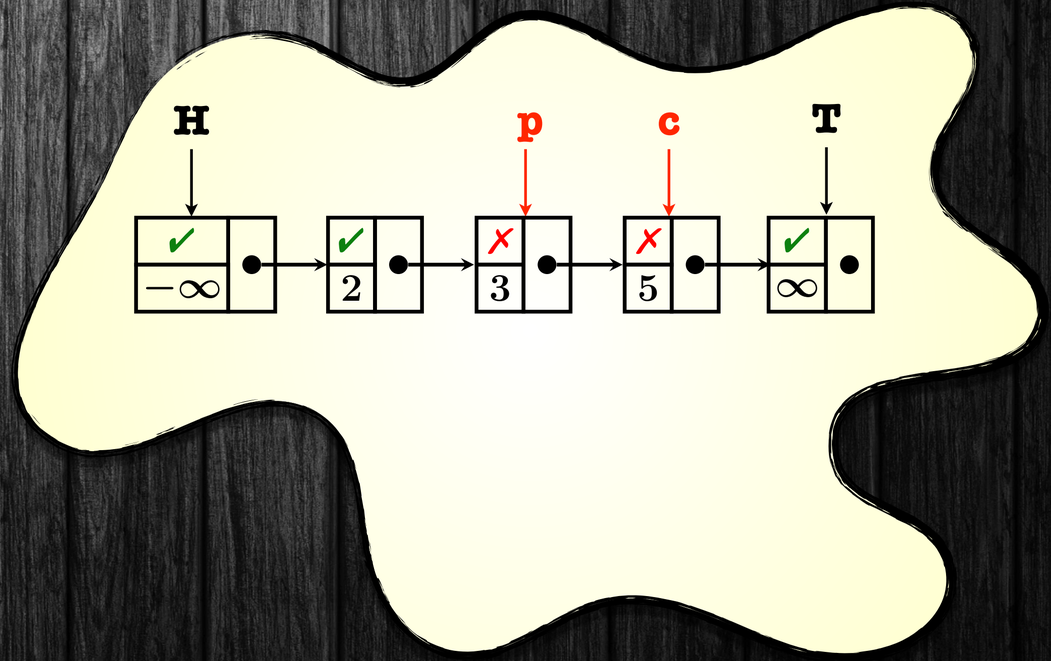
Sets | Optimistic Synchronization



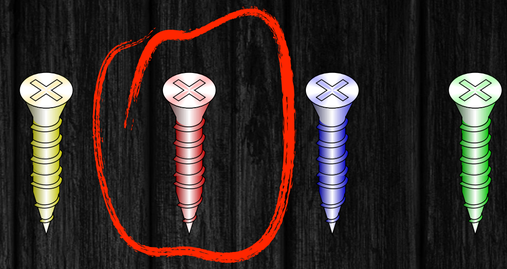
```
ctn(k):
Node p, c
 1  while (true)
 2    p = H
 3    c = p.next
 4    while (c.key < k)
 5      p = c
 6      c = c.next
 7    lock(p)
 8    lock(c)
 9    if (validate (p,c))
10      if (c.key=k)
11        return true
12      else
13        return false
16      unlock p
17      unlock c
18      exit
19    else
20      unlock p
21      unlock c
```



ctn(5)



Sets | Optimistic Synchronization



```
ctn(k):
Node p, c
1  while (true)
2    p = H
3    c = p.next
4    while (c.key < k)
5      p = c
6      c = c.next
7    lock(p)
8    lock(c)
9    if (validate (p,c))
10     if (c.key=k)
11       return true
12     else
13       return false
16     unlock p
17     unlock c
18     exit
19   else
20     unlock p
21     unlock c
```

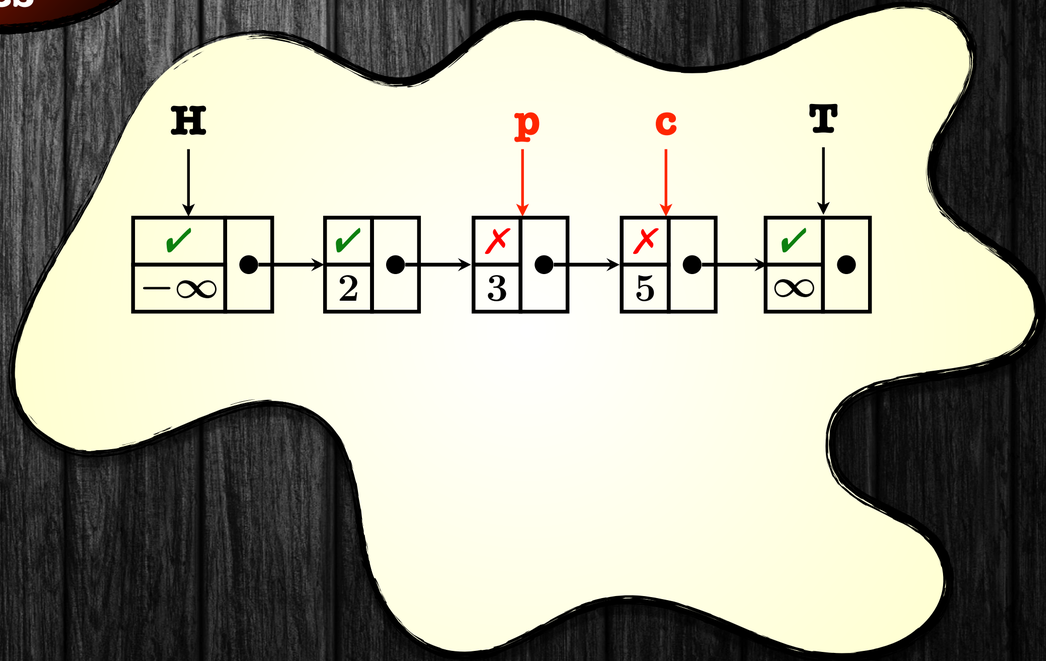


ctn(5)

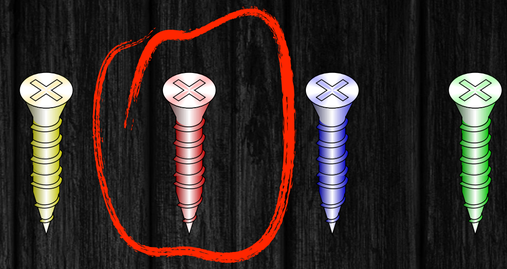
validation

check whether the two cells are still in the list

assume yes



Sets | Optimistic Synchronization



```
ctn(k):
Node p, c
1  while (true)
2    p = H
3    c = p.next
4    while (c.key < k)
5      p = c
6      c = c.next
7    lock(p)
8    lock(c)
9    if (validate (p,c))
10     if (c.key=k)
11       return true
12     else
13       return false
16     unlock p
17     unlock c
18     exit
19   else
20     unlock p
21     unlock c
```

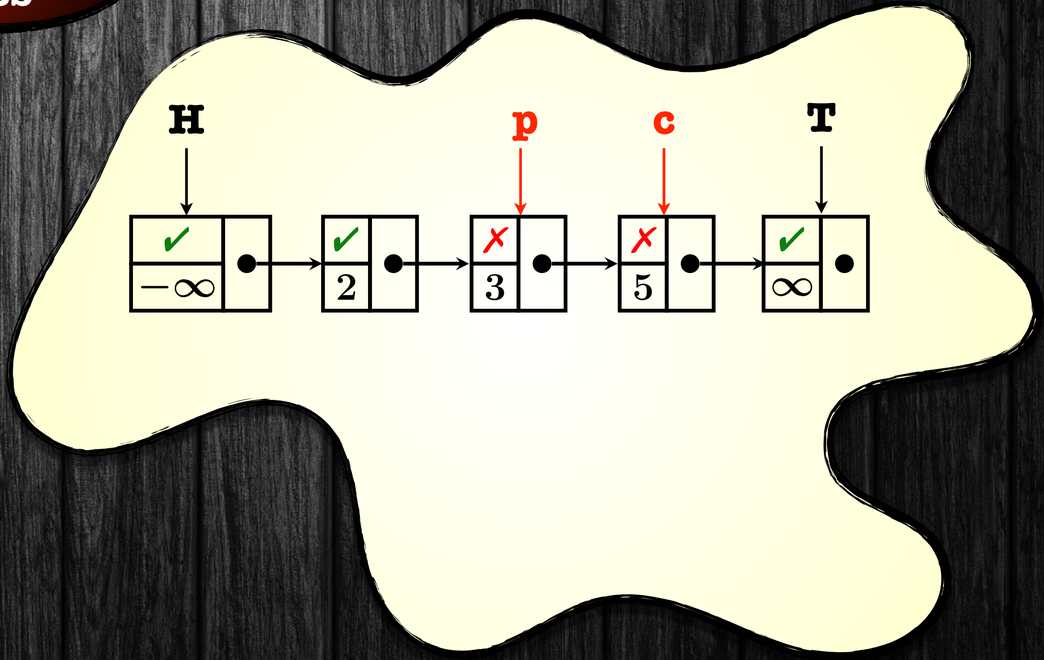


ctn(5)

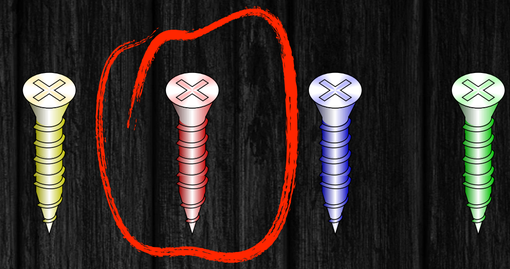
validation

check whether the two cells are still in the list

assume yes



Sets | Optimistic Synchronization



```
ctn(k):
Node p, c
1  while (true)
2    p = H
3    c = p.next
4    while (c.key < k)
5      p = c
6      c = c.next
7    lock(p)
8    lock(c)
9    if (validate (p,c))
10     if (c.key=k)
11       return true
12     else
13       return false
16     unlock p
17     unlock c
18     exit
19   else
20     unlock p
21     unlock c
```

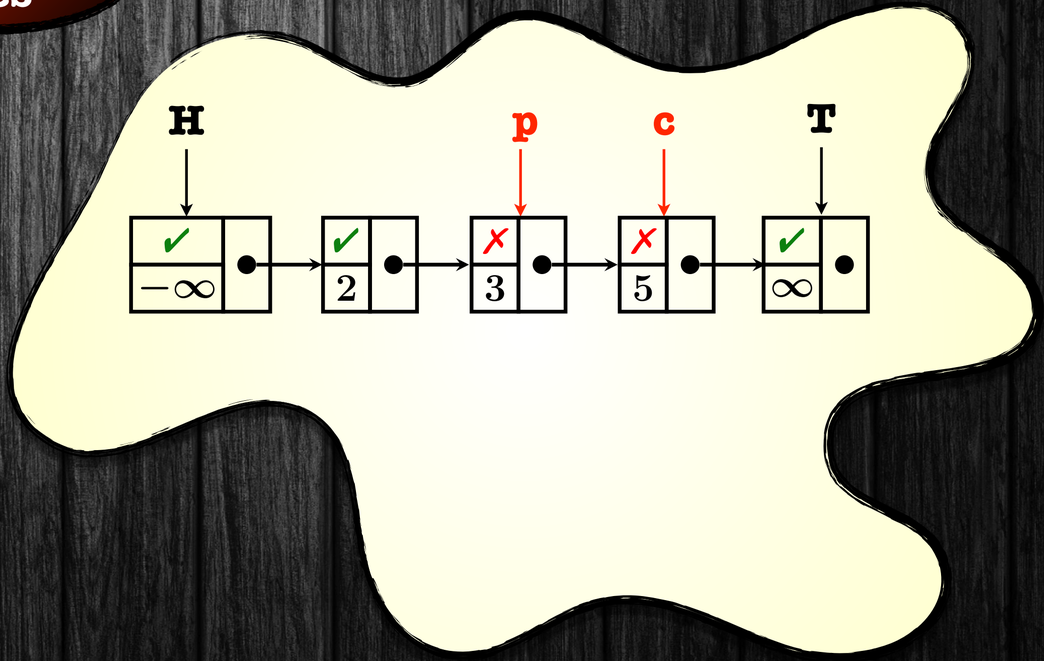


ctn(5)

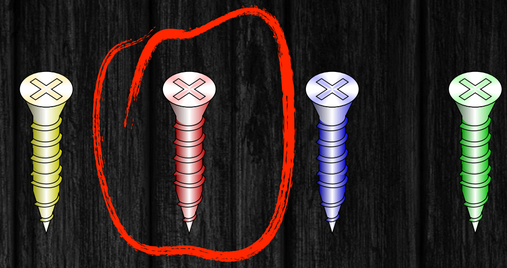
validation

check whether the two cells are still in the list

assume yes



Sets | Optimistic Synchronization



```
ctn(k):
Node p, c
1  while (true)
2    p = H
3    c = p.next
4    while (c.key < k)
5      p = c
6      c = c.next
7    lock(p)
8    lock(c)
9    if (validate (p,c))
10     if (c.key=k)
11       return true
12     else
13       return false
16     unlock p
17     unlock c
18     exit
19   else
20     unlock p
21     unlock c
```

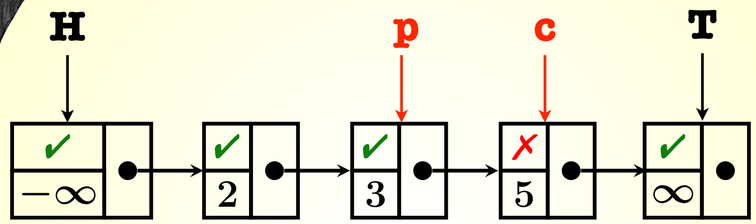


ctn(5)

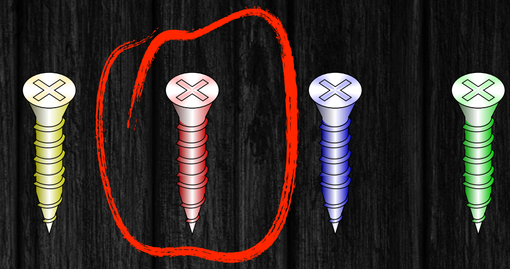
validation

check whether the two cells are still in the list

assume yes



Sets | Optimistic Synchronization



```
ctn(k):
Node p, c
 1  while (true)
 2    p = H
 3    c = p.next
 4    while (c.key < k)
 5      p = c
 6      c = c.next
 7    lock(p)
 8    lock(c)
 9    if (validate (p,c))
10      if (c.key=k)
11        return true
12      else
13        return false
16      unlock p
17      unlock c
18      exit
19    else
20      unlock p
21      unlock c
```

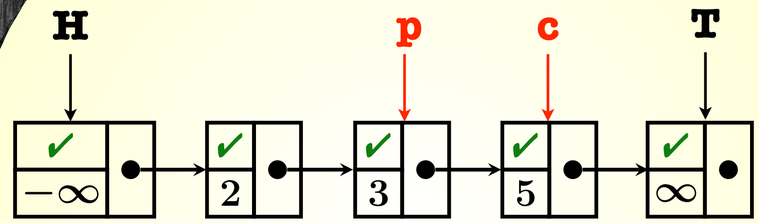


ctn(5)

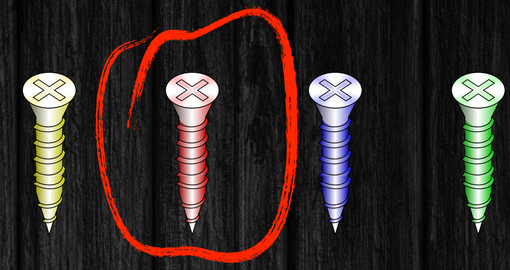
validation

check whether the two cells are still in the list

assume yes



Sets | Optimistic Synchronization



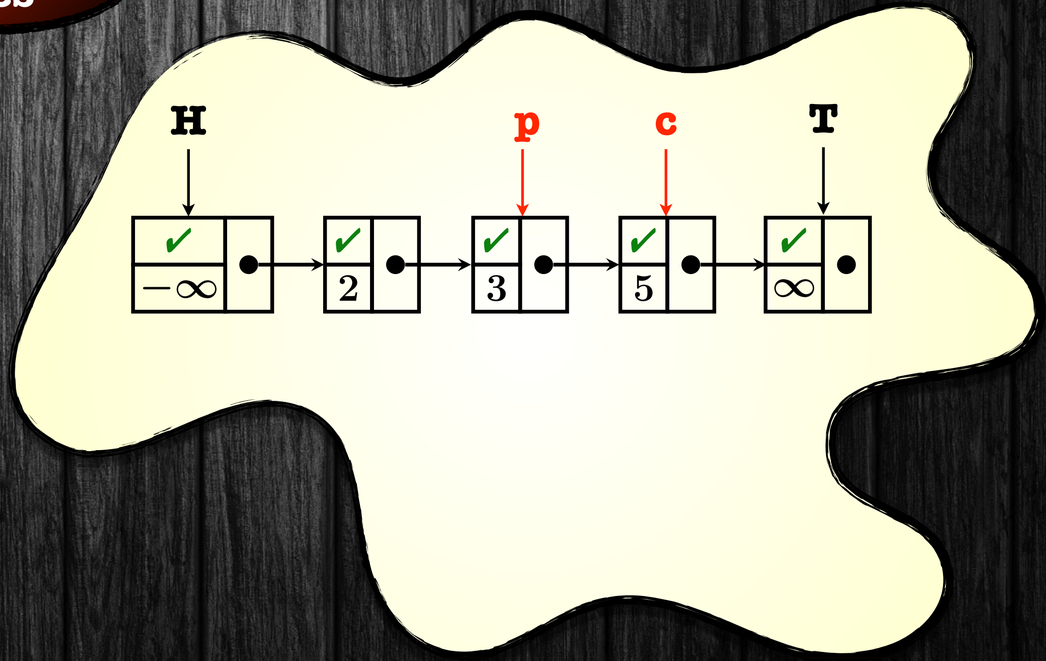
```
ctn(k):
Node p, c
1  while (true)
2    p = H
3    c = p.next
4    while (c.key < k)
5      p = c
6      c = c.next
7    lock(p)
8    lock(c)
9    if (validate (p,c))
10     if (c.key=k)
11       return true
12     else
13       return false
16     unlock p
17     unlock c
18     exit
19   else
20     unlock p
21     unlock c
```

ctn(5)

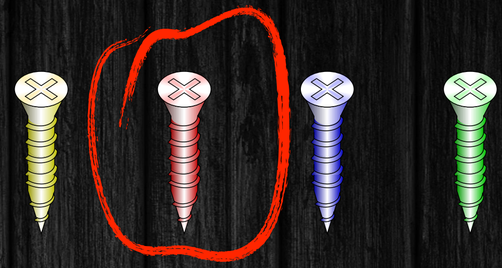
validation

check whether the two cells are still in the list

assume yes



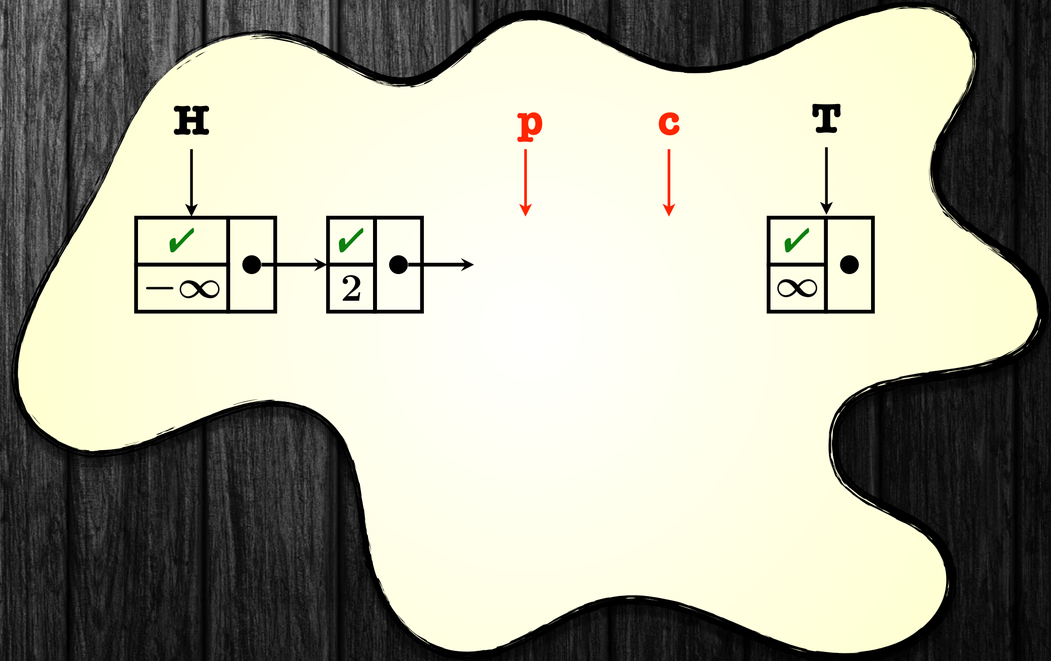
Sets | Optimistic Synchronization



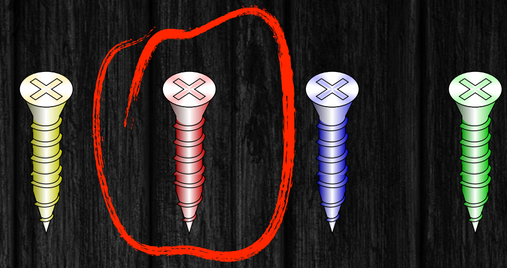
```
ctn(k):
Node p, c
 1  while (true)
 2    p = H
 3    c = p.next
 4    while (c.key < k)
 5      p = c
 6      c = c.next
 7    lock(p)
 8    lock(c)
 9    if (validate (p,c))
10      if (c.key=k)
11        return true
12      else
13        return false
16      unlock p
17      unlock c
18      exit
19    else
20      unlock p
21      unlock c
```



ctn(5)



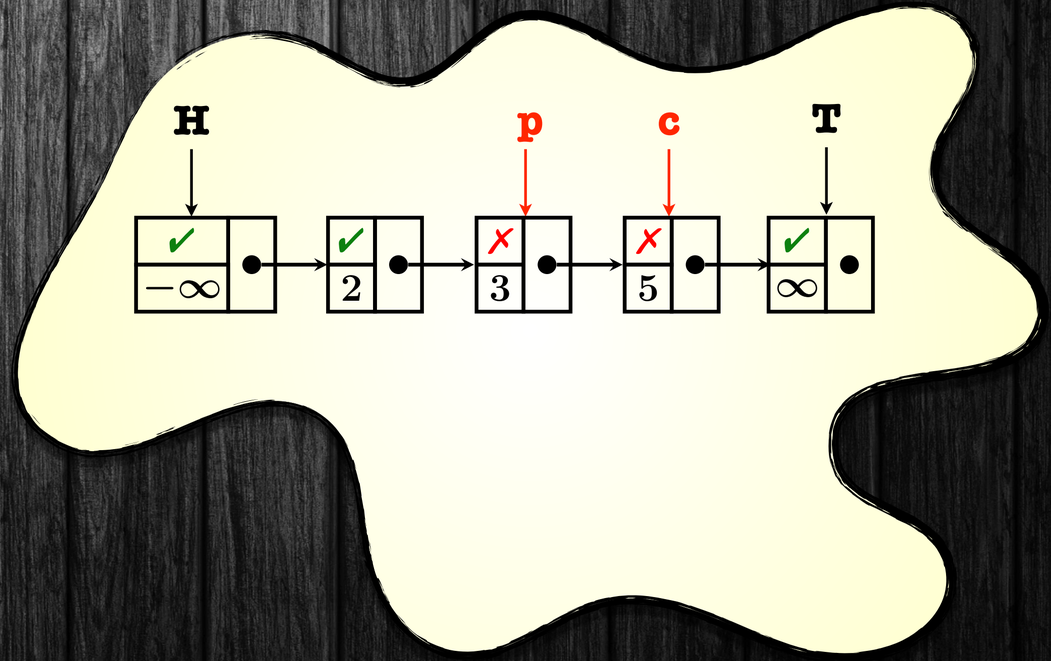
Sets | Optimistic Synchronization



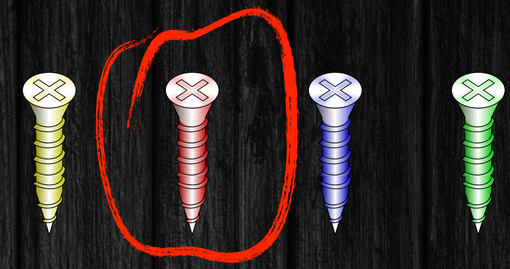
```
ctn(k):
Node p, c
 1  while (true)
 2    p = H
 3    c = p.next
 4    while (c.key < k)
 5      p = c
 6      c = c.next
 7    lock(p)
 8    lock(c)
 9    if (validate (p,c))
10      if (c.key=k)
11        return true
12      else
13        return false
16      unlock p
17      unlock c
18      exit
19    else
20      unlock p
21      unlock c
```



ctn(5)



Sets | Optimistic Synchronization



```
ctn(k):
Node p, c
1  while (true)
2    p = H
3    c = p.next
4    while (c.key < k)
5      p = c
6      c = c.next
7    lock(p)
8    lock(c)
9    if (validate (p,c))
10     if (c.key=k)
11       return true
12     else
13       return false
16     unlock p
17     unlock c
18     exit
19   else
20     unlock p
21     unlock c
```

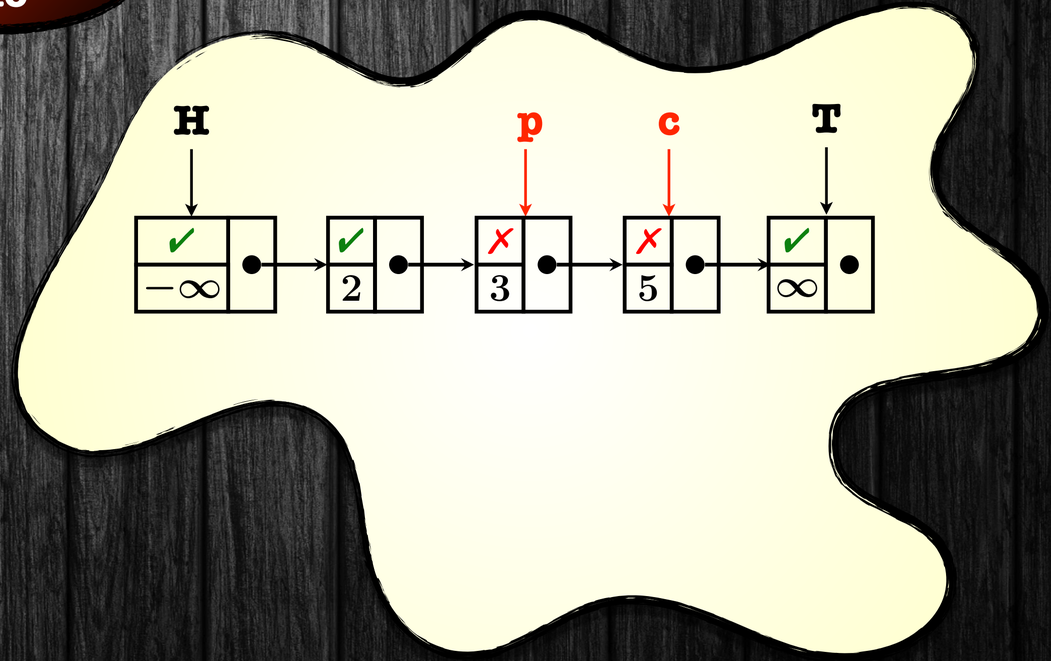


ctn(5)

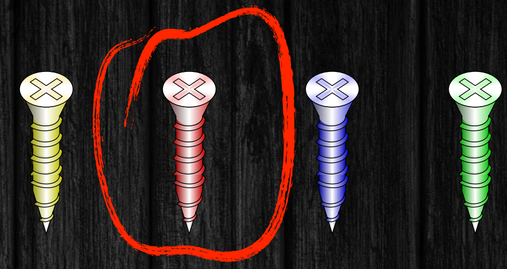
validation

check whether the two cells are still in the list

assume no



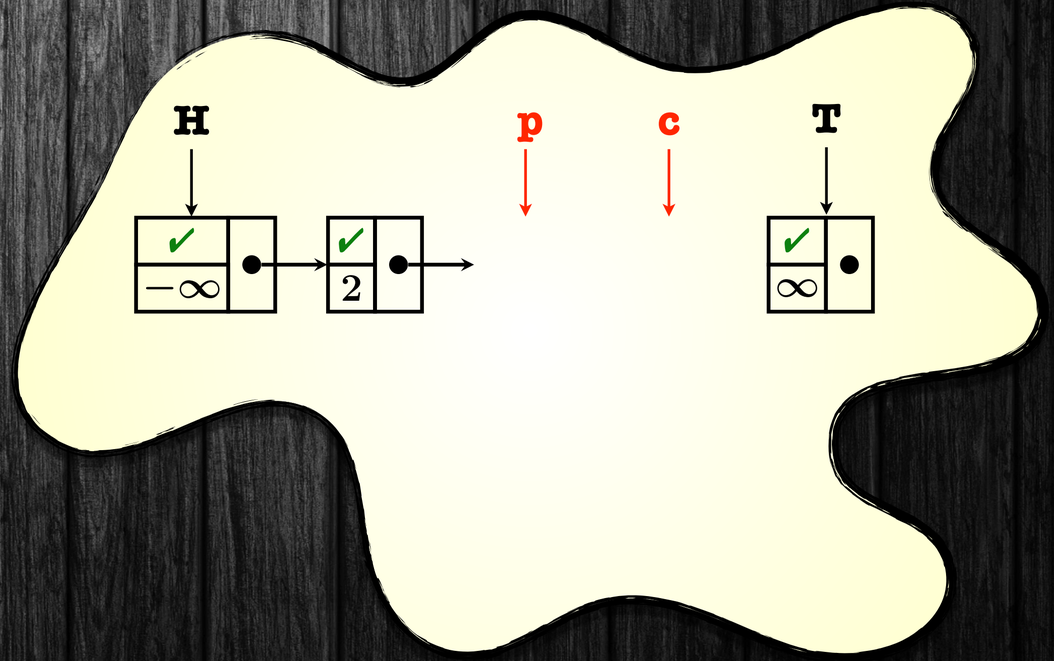
Sets | Optimistic Synchronization



```
ctn(k):
Node p, c
 1  while (true)
 2    p = H
 3    c = p.next
 4    while (c.key < k)
 5      p = c
 6      c = c.next
 7    lock(p)
 8    lock(c)
 9    if (validate (p,c))
10      if (c.key=k)
11        return true
12      else
13        return false
16      unlock p
17      unlock c
18      exit
19    else
20      unlock p
21      unlock c
```



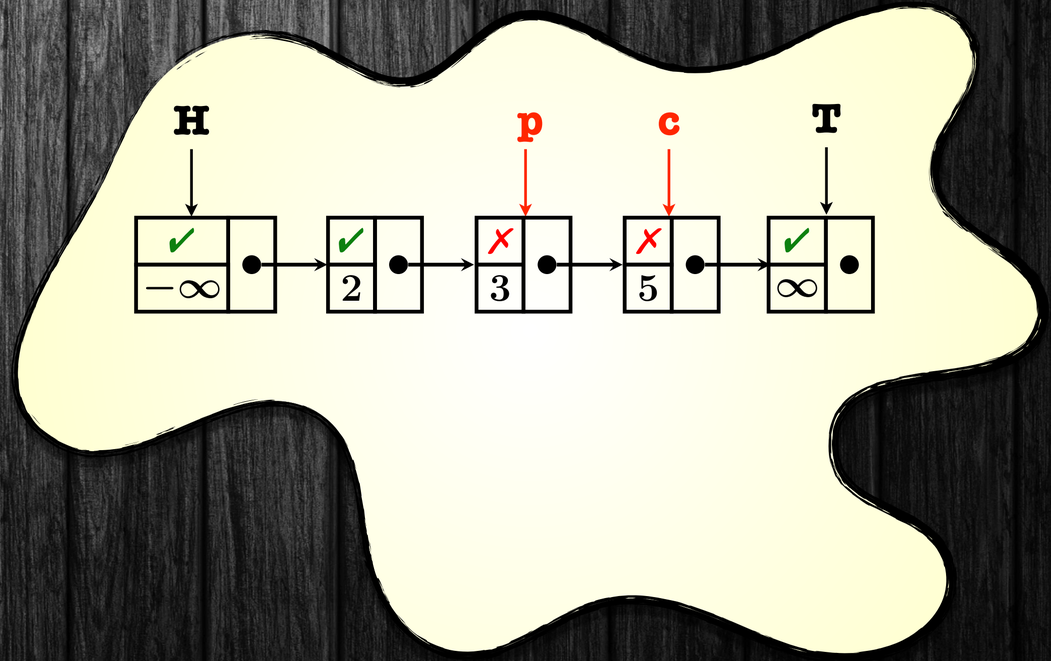
ctn(5)



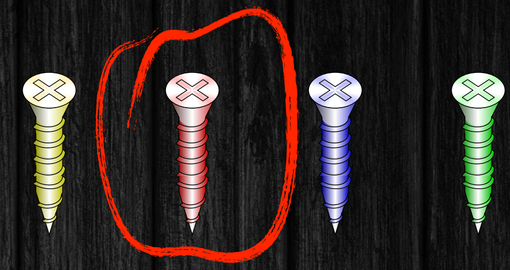
Sets | Optimistic Synchronization

```
ctn(k):
Node p, c
 1  while (true)
 2    p = H
 3    c = p.next
 4    while (c.key < k)
 5      p = c
 6      c = c.next
 7    lock(p)
 8    lock(c)
 9    if (validate (p,c))
10      if (c.key=k)
11        return true
12      else
13        return false
16      unlock p
17      unlock c
18      exit
19    else
20      unlock p
21      unlock c
```

ctn(5)



Sets | Optimistic Synchronization



```
ctn(k):
Node p, c
1  while (true)
2    p = H
3    c = p.next
4    while (c.key < k)
5      p = c
6      c = c.next
7    lock(p)
8    lock(c)
9    if (validate (p,c))
10     if (c.key=k)
11       return true
12     else
13       return false
16     unlock p
17     unlock c
18     exit
19   else
20     unlock p
21     unlock c
```

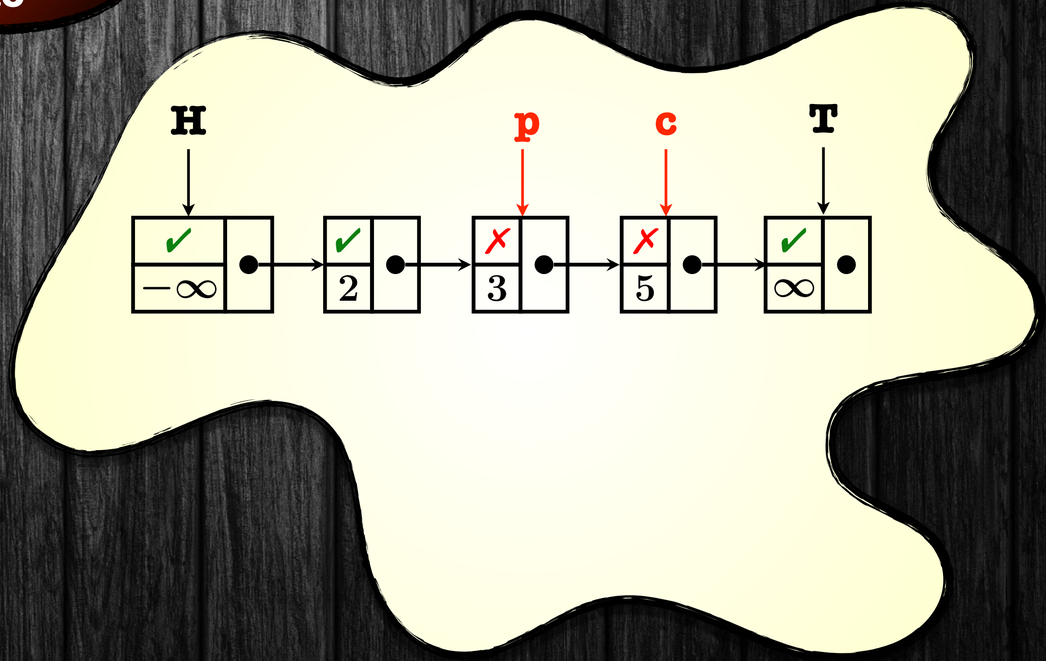


ctn(5)

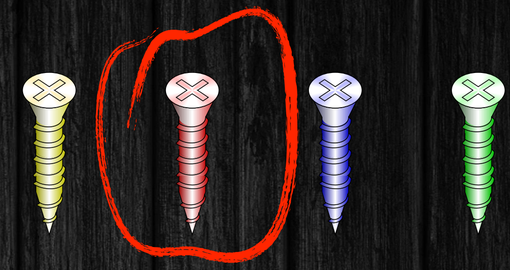
validation

check whether the two cells are still in the list

assume no



Sets | Optimistic Synchronization



```
ctn(k):
Node p, c
1  while (true)
2    p = H
3    c = p.next
4    while (c.key < k)
5      p = c
6      c = c.next
7    lock(p)
8    lock(c)
9    if (validate (p,c))
10     if (c.key=k)
11       return true
12     else
13       return false
16     unlock p
17     unlock c
18     exit
19   else
20     unlock p
21     unlock c
```

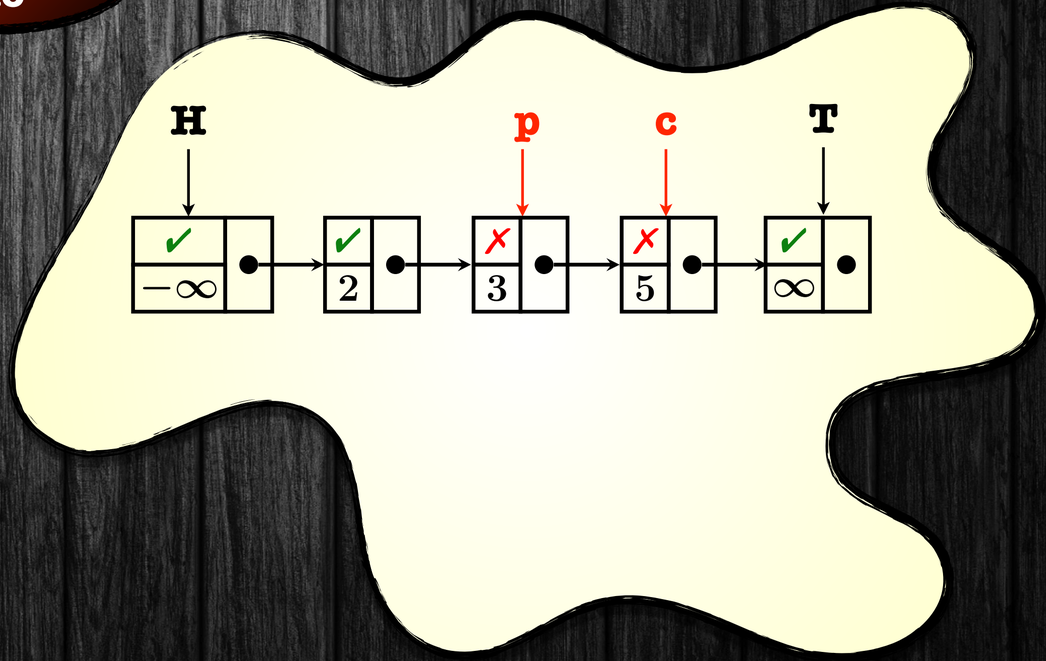


ctn(5)

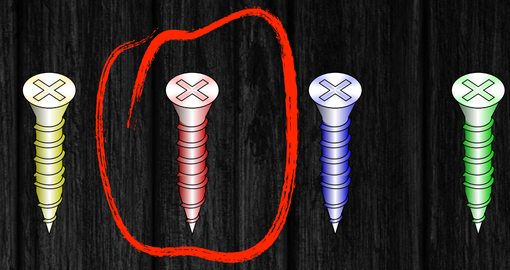
validation

check whether the two cells are still in the list

assume no



Sets | Optimistic Synchronization



```
ctn(k):
Node p, c
1  while (true)
2    p = H
3    c = p.next
4    while (c.key < k)
5      p = c
6      c = c.next
7    lock(p)
8    lock(c)
9    if (validate (p,c))
10     if (c.key=k)
11       return true
12     else
13       return false
16     unlock p
17     unlock c
18     exit
19   else
20     unlock p
21     unlock c
```

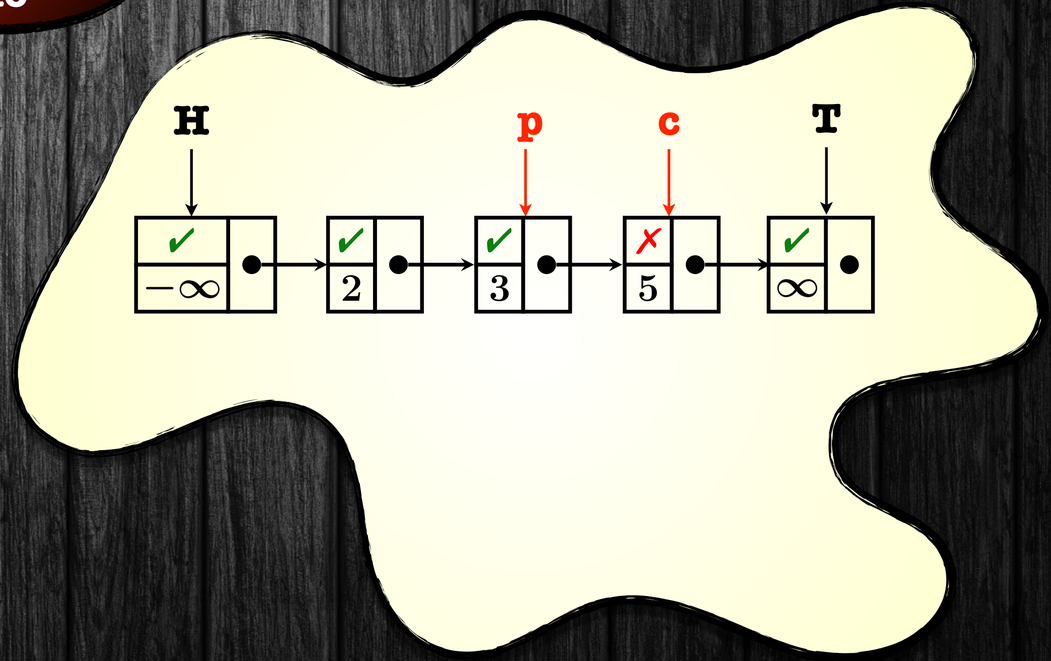


ctn(5)

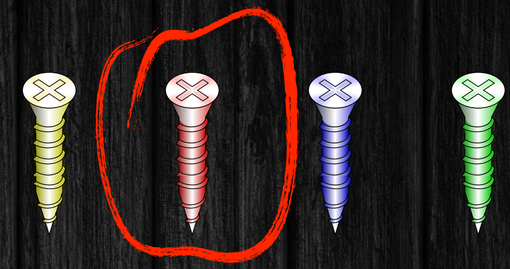
validation

check whether the two cells are still in the list

assume no



Sets | Optimistic Synchronization



```
ctn(k):
Node p, c
1  while (true)
2    p = H
3    c = p.next
4    while (c.key < k)
5      p = c
6      c = c.next
7    lock(p)
8    lock(c)
9    if (validate (p,c))
10     if (c.key=k)
11       return true
12     else
13       return false
16     unlock p
17     unlock c
18     exit
19   else
20     unlock p
21     unlock c
```

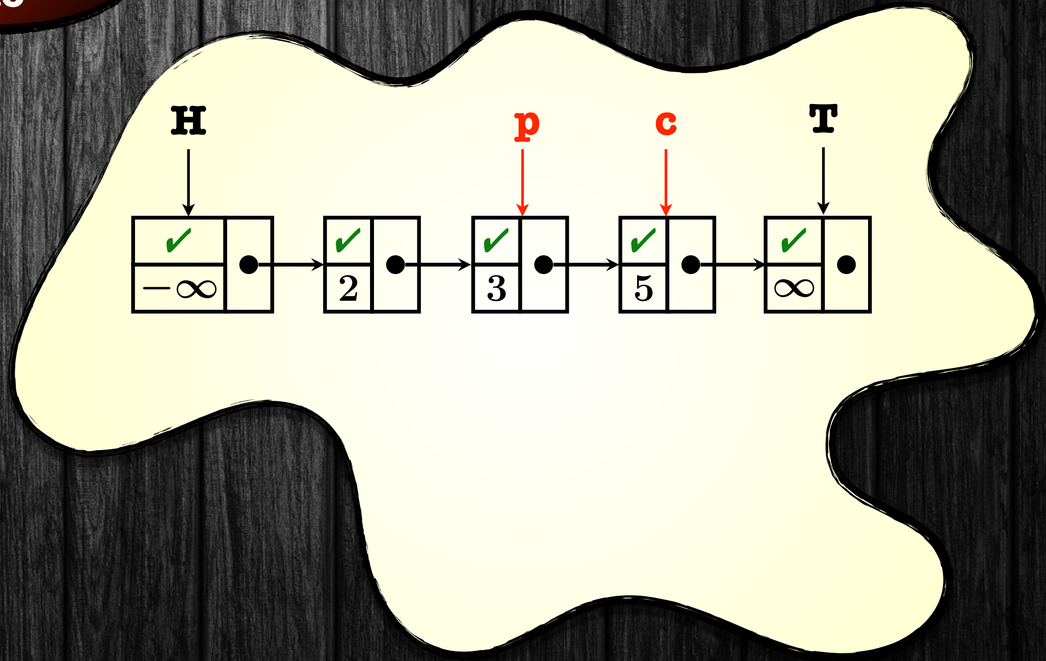


ctn(5)

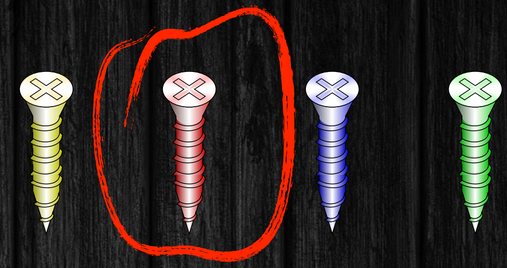
validation

check whether the two cells are still in the list

assume no



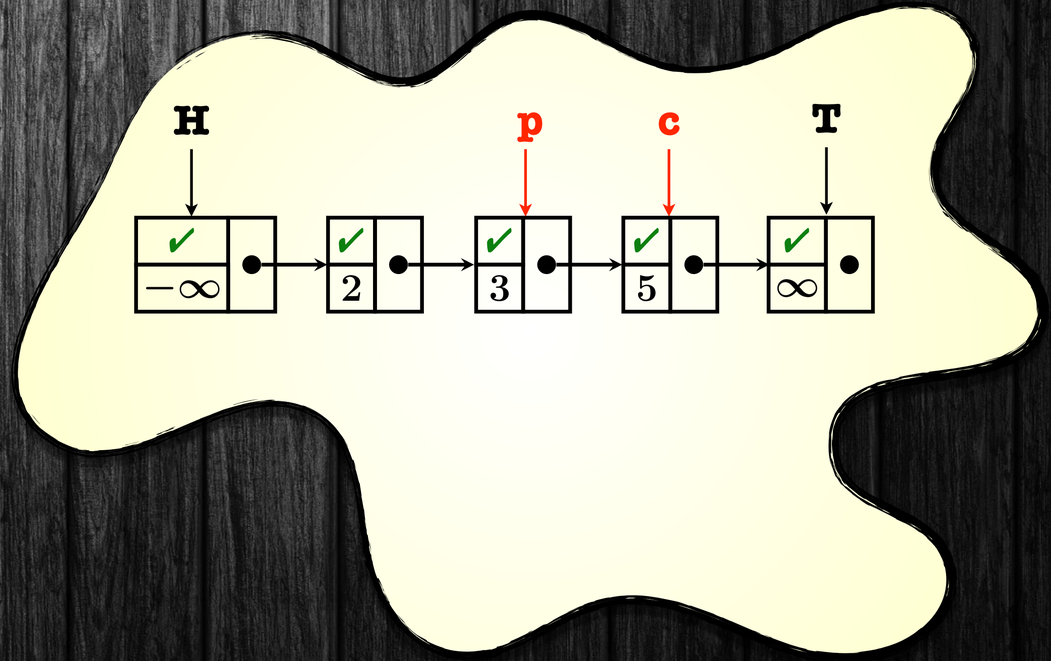
Sets | Optimistic Synchronization



```
ctn(k):
Node p, c
1  while (true)
2    p = H
3    c = p.next
4    while (c.key < k)
5      p = c
6      c = c.next
7    lock(p)
8    lock(c)
9    if (validate (p,c))
10     if (c.key=k)
11       return true
12     else
13       return false
16     unlock p
17     unlock c
18     exit
19   else
20     unlock p
21     unlock c
```



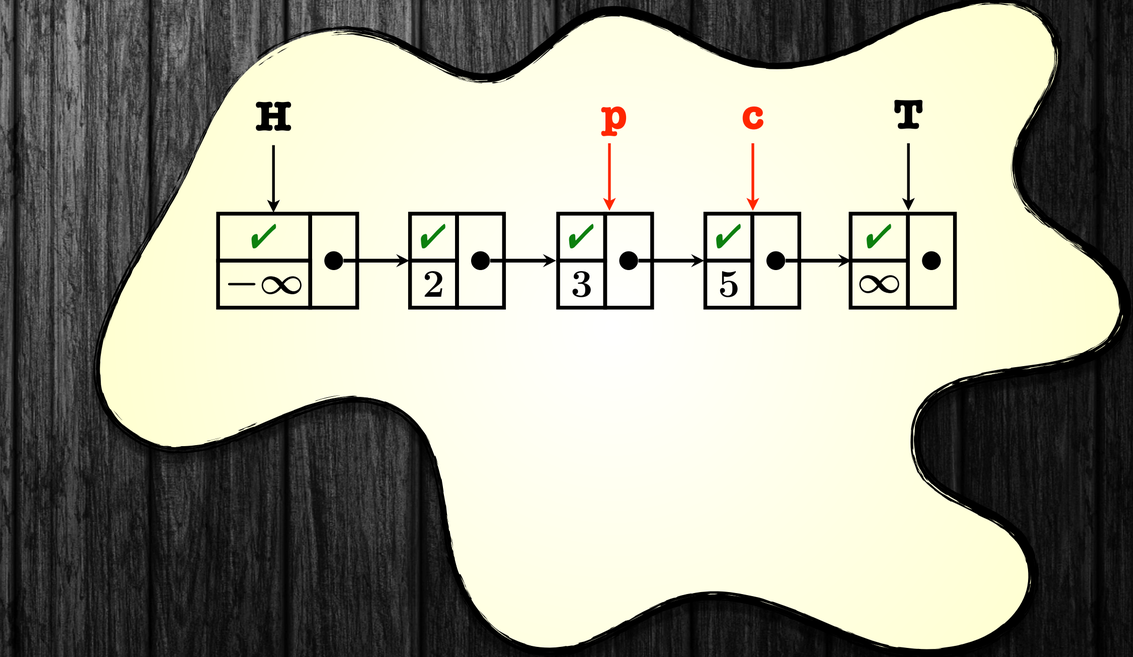
ctn(5)



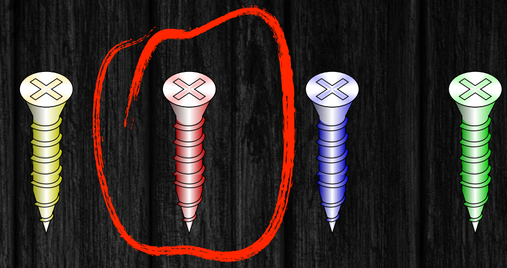
Sets | Optimistic Synchronization

```
validate(p,c):
Node n
1  n=H
2  while (n.key < p.key)
3    n=n.next
4  if ((n=p) and (p.next=c))
5    return true
6  else
7    return false
```

validate



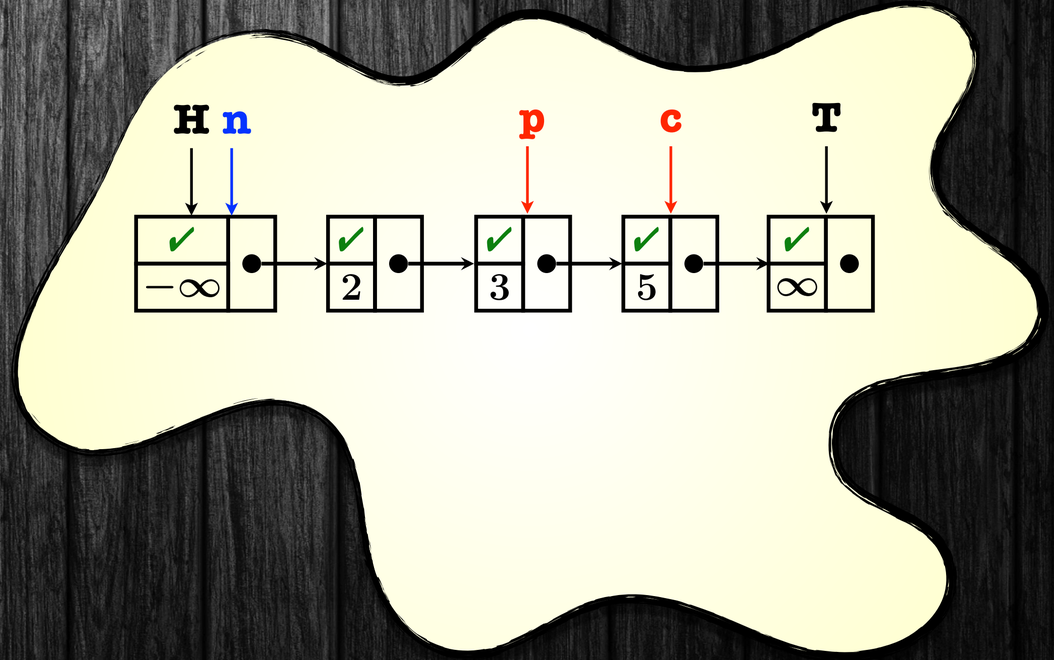
Sets | Optimistic Synchronization



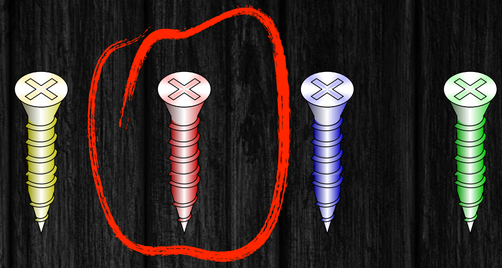
```
validate(p,c):
Node n
1  n=H
2  while (n.key < p.key)
3    n=n.next
4  if ((n=p) and (p.next=c))
5    return true
6  else
7    return false
```



validate



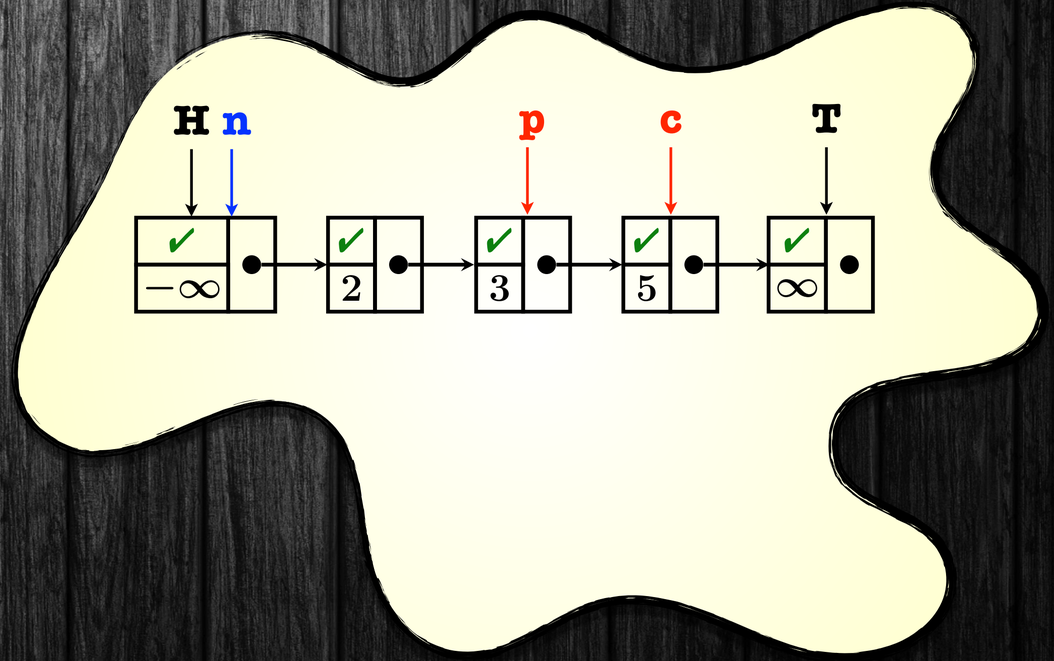
Sets | Optimistic Synchronization



```
validate(p,c):
Node n
1  n=H
2  while (n.key < p.key)
3    n=n.next
4  if ((n=p) and (p.next=c))
5    return true
6  else
7    return false
```



validate

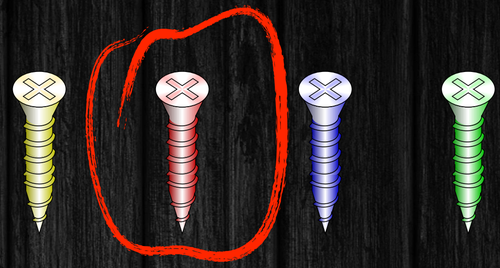


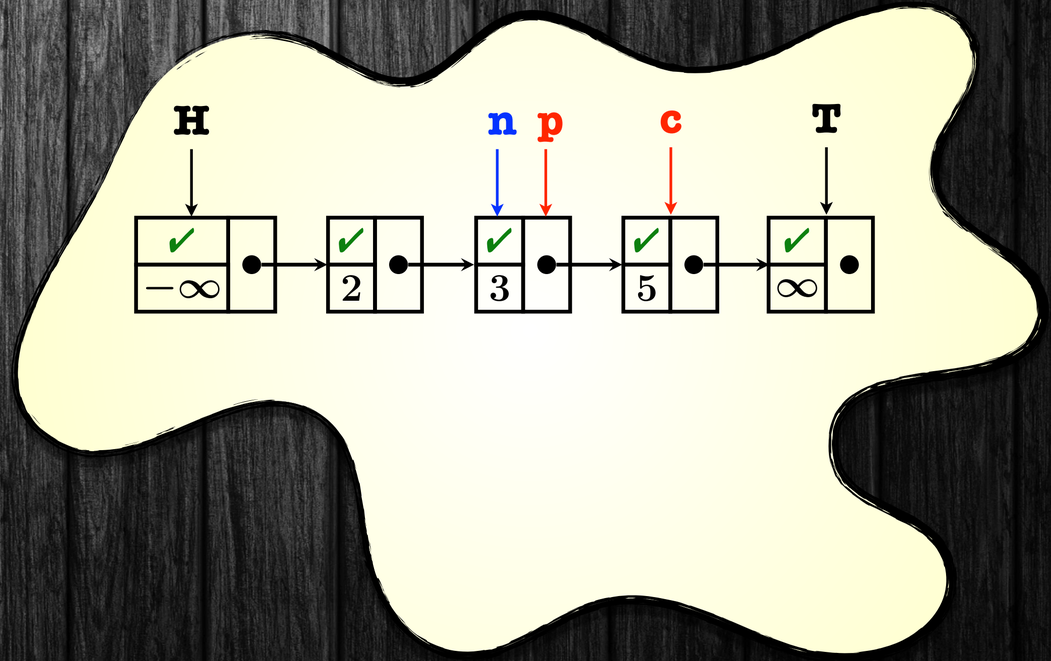
Sets | Optimistic Synchronization

```
validate(p,c):
Node n
1  n=H
2  while (n.key < p.key)
3    n=n.next
4  if ((n=p) and (p.next=c))
5    return true
6  else
7    return false
```

validate





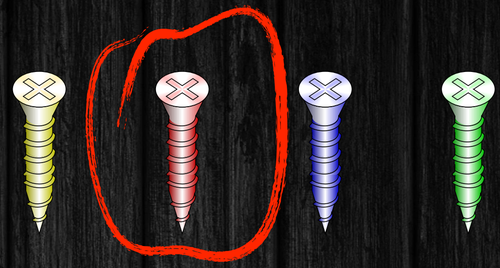


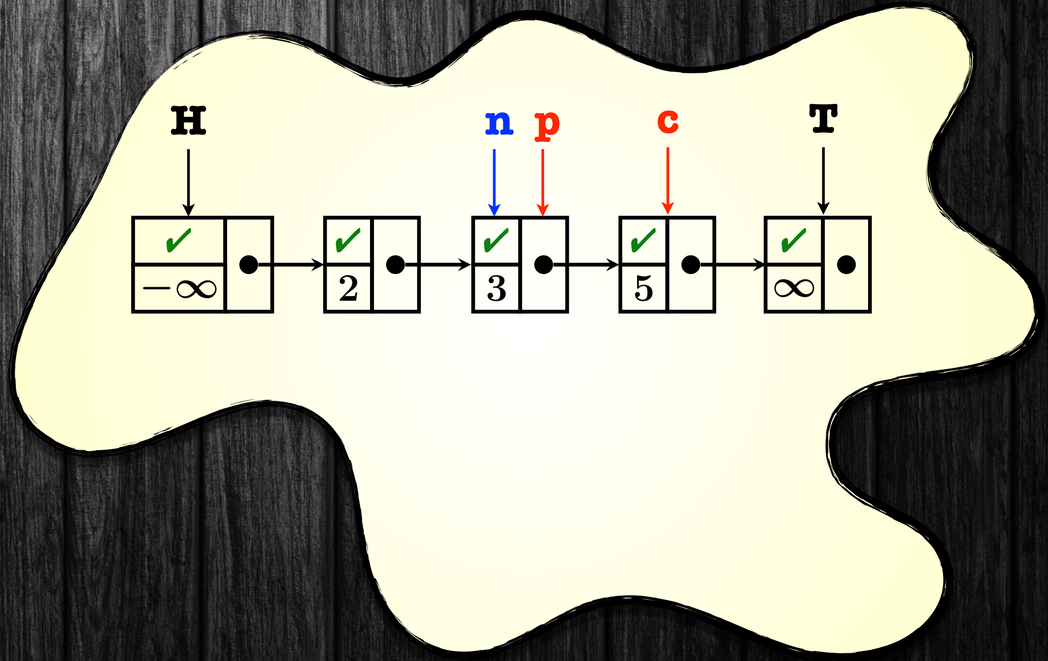
Sets | Optimistic Synchronization

```
validate(p,c):
Node n
1  n=H
2  while (n.key < p.key)
3    n=n.next
4  if ((n=p) and (p.next=c))
5    return true
6  else
7    return false
```



validate





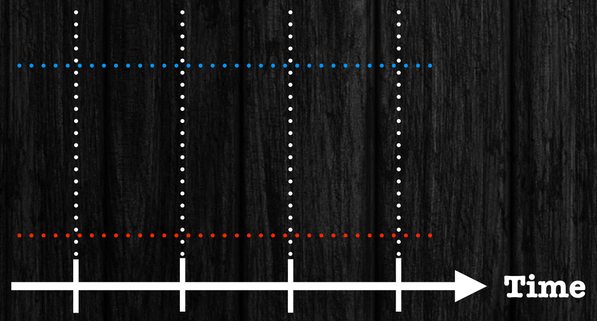
Sets | Optimistic Synchronization | Linearization Policy

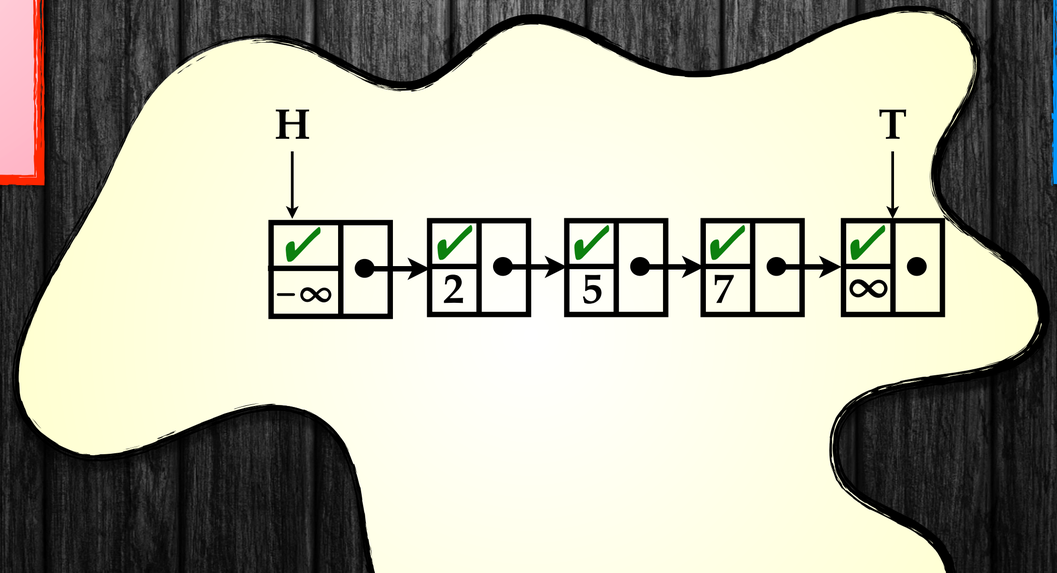
Sets | Optimistic Synchronization | Linea… | LP

"time point at which validation succeeds"

Sets | Optimistic Synchronization | Linearization Policy

```
add(k):
Node p, c
1  while (true)
2    p = H
3    c = p.next
4    while (c.key < k)
5      p = c
6      c = c.next
7    lock(p)
8    lock(c)
9    if (validate (p,c))
10     if (c.key=k)
11       return false
12     else
13       n = new Node(k,true,-)
14       n.next = c
15       p.next = n
16       return true
17     unlock p
18     unlock c
19     exit
20   else
21     unlock p
22     unlock c
```





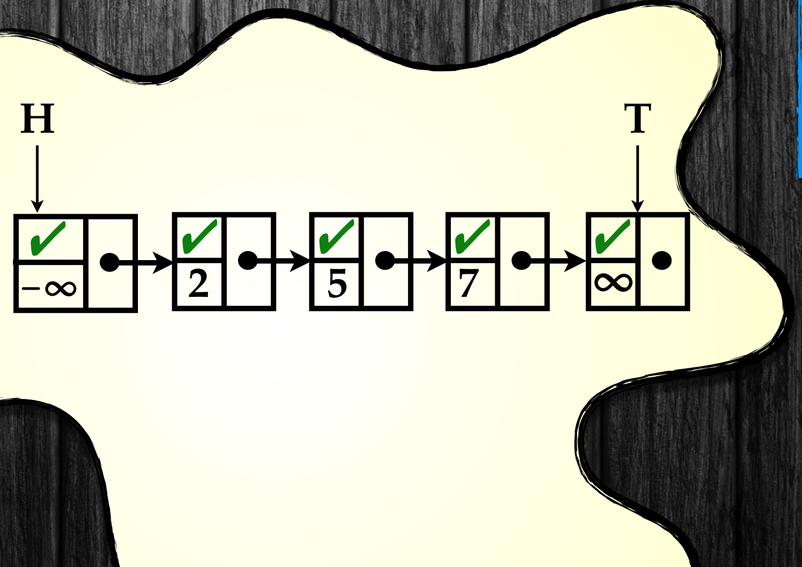

```
add(k):
Node p, c
1  while (true)
2    p = H
3    c = p.next
4    while (c.key < k)
5      p = c
6      c = c.next
7    lock(p)
8    lock(c)
9    if (validate (p,c))
10     if (c.key=k)
11       return false
12     else
13       n = new Node(k,true,-)
14       n.next = c
15       p.next = n
16       return true
17     unlock p
18     unlock c
19     exit
20   else
21     unlock p
22     unlock c
```

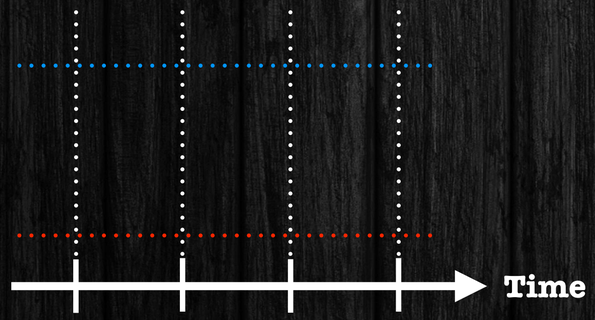
Sets | Optimistic Synchronization | Linearization Policy

```
add(k):
Node p, c
 1  while (true)
 2    p = H
 3    c = p.next
 4    while (c.key < k)
 5      p = c
 6      c = c.next
 7    lock(p)
 8    lock(c)
 9    if (validate (p,c))
10      if (c.key=k)
11        return false
12      else
13        n = new Node(k,true,-)
14        n.next = c
15        p.next = n
16        return true
17      unlock p
18      unlock c
19      exit
20    else
21      unlock p
22      unlock c
```



add(6)



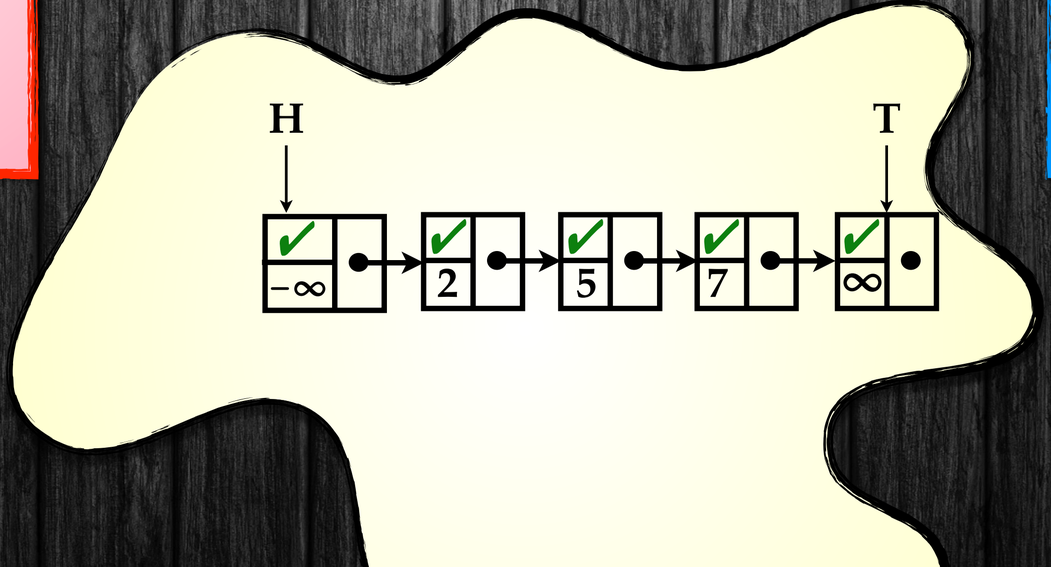


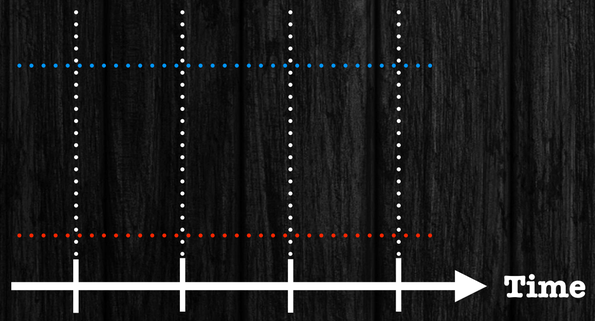

```
add(k):
Node p, c
 1  while (true)
 2    p = H
 3    c = p.next
 4    while (c.key < k)
 5      p = c
 6      c = c.next
 7    lock(p)
 8    lock(c)
 9    if (validate (p,c))
10      if (c.key=k)
11        return false
12      else
13        n = new Node(k,true,-)
14        n.next = c
15        p.next = n
16        return true
17      unlock p
18      unlock c
19      exit
20    else
21      unlock p
22      unlock c
```

Sets | Optimistic Synchronization | Linearization Policy

```
add(k):
Node p, c
1  while (true)
2    p = H
3    c = p.next
4    while (c.key < k)
5      p = c
6      c = c.next
7    lock(p)
8    lock(c)
9    if (validate (p,c))
10     if (c.key=k)
11       return false
12     else
13       n = new Node(k,true,-)
14       n.next = c
15       p.next = n
16       return true
17     unlock p
18     unlock c
19     exit
20   else
21     unlock p
22     unlock c
```





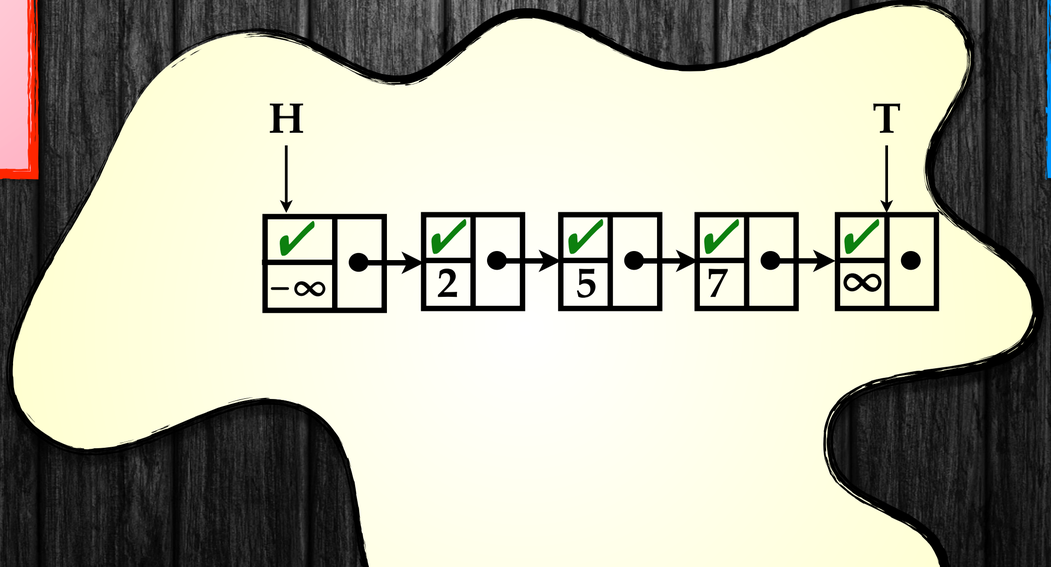


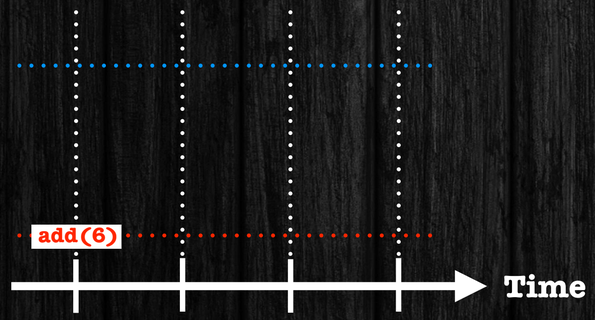

```
add(k):
Node p, c
1  while (true)
2    p = H
3    c = p.next
4    while (c.key < k)
5      p = c
6      c = c.next
7    lock(p)
8    lock(c)
9    if (validate (p,c))
10     if (c.key=k)
11       return false
12     else
13       n = new Node(k,true,-)
14       n.next = c
15       p.next = n
16       return true
17     unlock p
18     unlock c
19     exit
20   else
21     unlock p
22     unlock c
```

Sets | Optimistic Synchronization | Linearization Policy

```
add(k):
Node p, c
1  while (true)
2    p = H
3    c = p.next
4    while (c.key < k)
5      p = c
6      c = c.next
7    lock(p)
8    lock(c)
9    if (validate (p,c))
10     if (c.key=k)
11       return false
12     else
13       n = new Node(k,true,-)
14       n.next = c
15       p.next = n
16       return true
17     unlock p
18     unlock c
19     exit
20   else
21     unlock p
22     unlock c
```



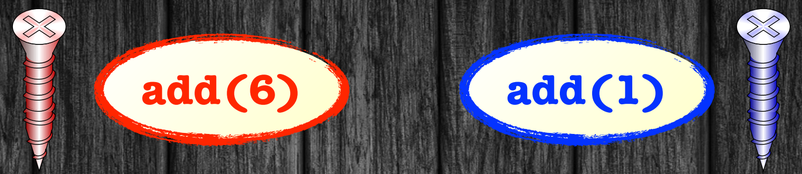




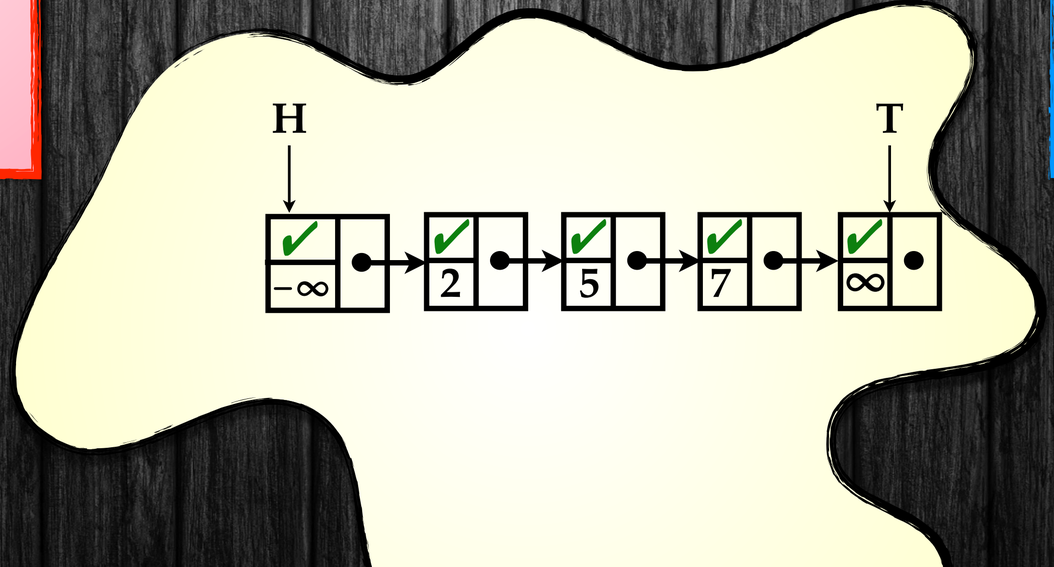

```
add(k):
Node p, c
1  while (true)
2    p = H
3    c = p.next
4    while (c.key < k)
5      p = c
6      c = c.next
7    lock(p)
8    lock(c)
9    if (validate (p,c))
10     if (c.key=k)
11       return false
12     else
13       n = new Node(k,true,-)
14       n.next = c
15       p.next = n
16       return true
17     unlock p
18     unlock c
19     exit
20   else
21     unlock p
22     unlock c
```

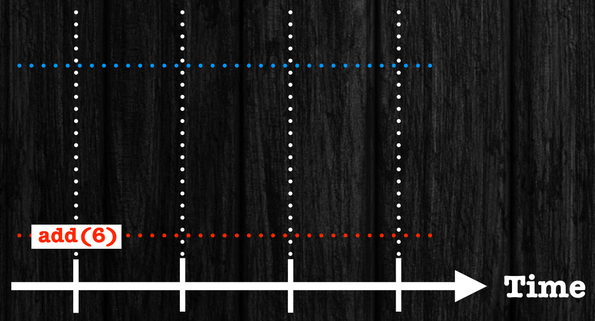
Sets | Optimistic Synchronization | Linearization Policy

```
add(k):
Node p, c
1  while (true)
2    p = H
3    c = p.next
4    while (c.key < k)
5      p = c
6      c = c.next
7    lock(p)
8    lock(c)
9    if (validate (p,c))
10     if (c.key=k)
11       return false
12     else
13       n = new Node(k,true,-)
14       n.next = c
15       p.next = n
16       return true
17     unlock p
18     unlock c
19     exit
20   else
21     unlock p
22     unlock c
```



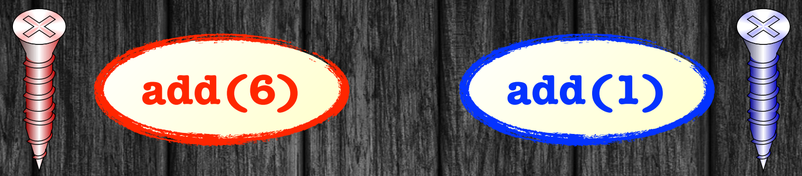

```
add(k):
Node p, c
1  while (true)
2    p = H
3    c = p.next
4    while (c.key < k)
5      p = c
6      c = c.next
7    lock(p)
8    lock(c)
9    if (validate (p,c))
10     if (c.key=k)
11       return false
12     else
13       n = new Node(k,true,-)
14       n.next = c
15       p.next = n
16       return true
17     unlock p
18     unlock c
19     exit
20   else
21     unlock p
22     unlock c
```

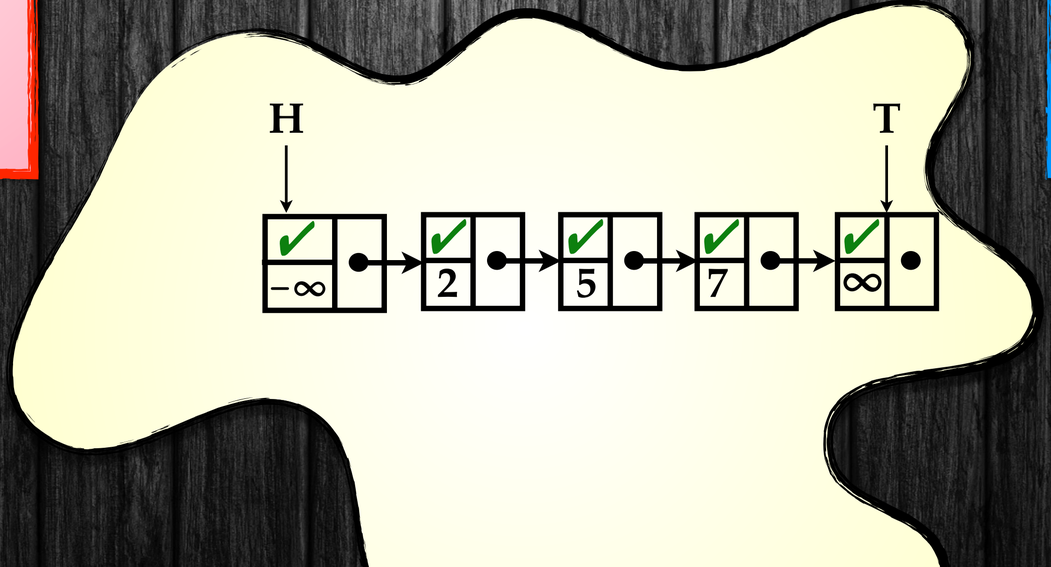


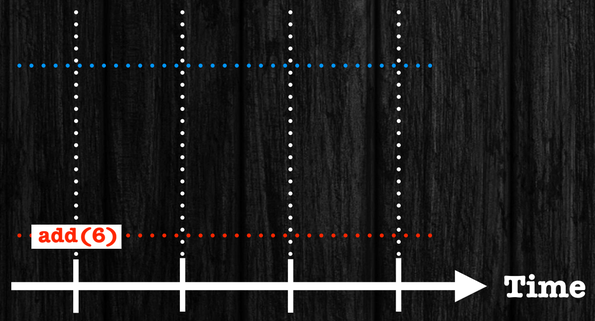


Sets | Optimistic Synchronization | Linearization Policy

```
add(k):
Node p, c
1  while (true)
2    p = H
3    c = p.next
4    while (c.key < k)
5      p = c
6      c = c.next
7    lock(p)
8    lock(c)
9    if (validate (p,c))
10     if (c.key=k)
11       return false
12     else
13       n = new Node(k,true,-)
14       n.next = c
15       p.next = n
16       return true
17     unlock p
18     unlock c
19     exit
20   else
21     unlock p
22     unlock c
```



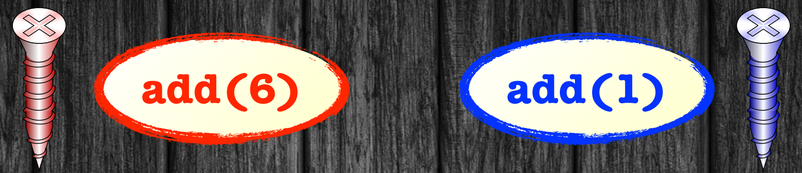


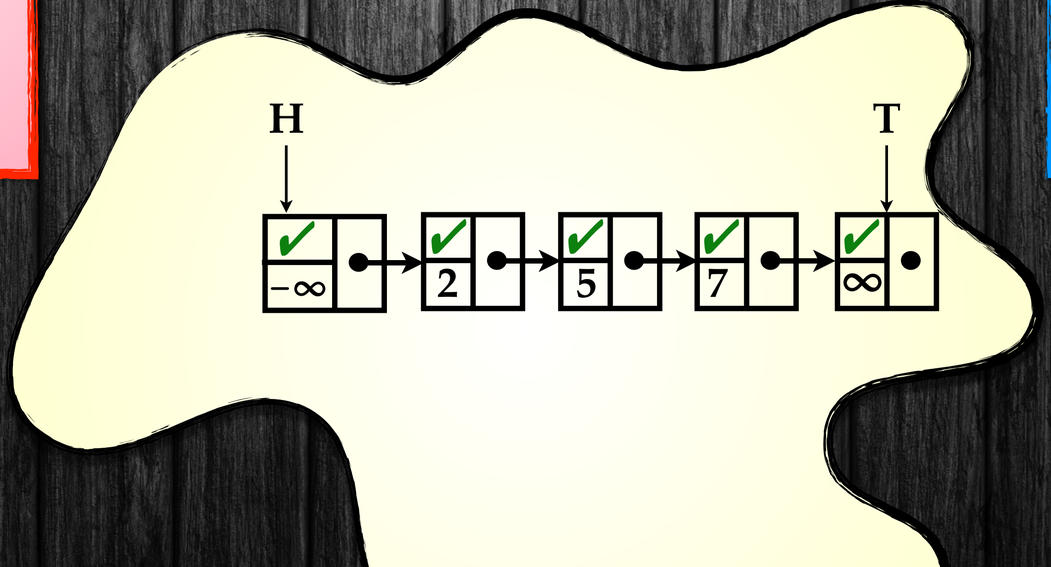


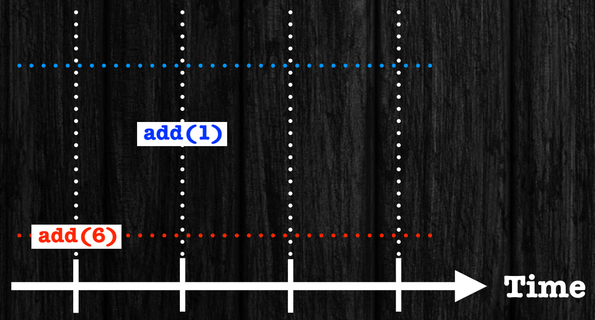

```
add(k):
Node p, c
1  while (true)
2    p = H
3    c = p.next
4    while (c.key < k)
5      p = c
6      c = c.next
7    lock(p)
8    lock(c)
9    if (validate (p,c))
10     if (c.key=k)
11       return false
12     else
13       n = new Node(k,true,-)
14       n.next = c
15       p.next = n
16       return true
17     unlock p
18     unlock c
19     exit
20   else
21     unlock p
22     unlock c
```

Sets | Optimistic Synchronization | Linearization Policy

```
add(k):
Node p, c
1  while (true)
2    p = H
3    c = p.next
4    while (c.key < k)
5      p = c
6      c = c.next
7    lock(p)
8    lock(c)
9    if (validate (p,c))
10     if (c.key=k)
11       return false
12     else
13       n = new Node(k,true,-)
14       n.next = c
15       p.next = n
16       return true
17     unlock p
18     unlock c
19     exit
20   else
21     unlock p
22     unlock c
```








```
add(k):
Node p, c
1  while (true)
2    p = H
3    c = p.next
4    while (c.key < k)
5      p = c
6      c = c.next
7    lock(p)
8    lock(c)
9    if (validate (p,c))
10     if (c.key=k)
11       return false
12     else
13       n = new Node(k,true,-)
14       n.next = c
15       p.next = n
16       return true
17     unlock p
18     unlock c
19     exit
20   else
21     unlock p
22     unlock c
```

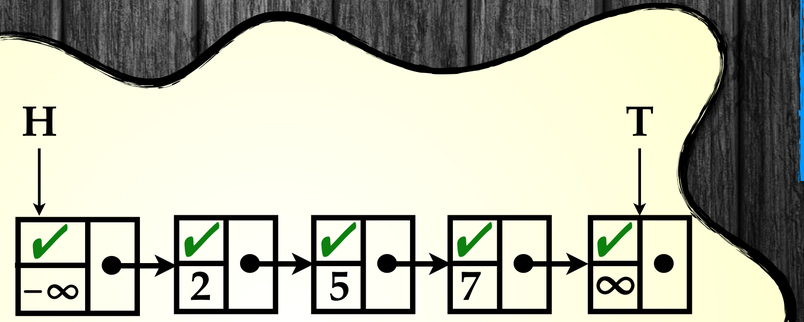
Sets | Optimistic Synchronization | Linearization Policy

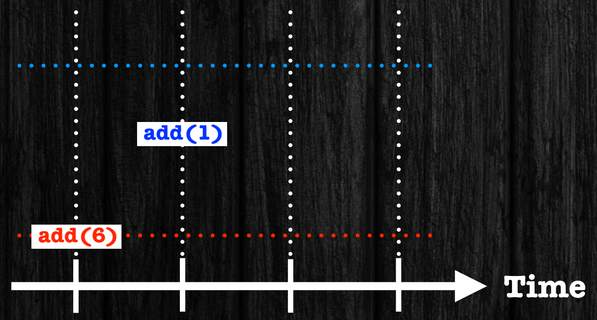
```
add(k):
Node p, c
1  while (true)
2    p = H
3    c = p.next
4    while (c.key < k)
5      p = c
6      c = c.next
7    lock(p)
8    lock(c)
9    if (validate (p,c))
10     if (c.key=k)
11       return false
12     else
13       n = new Node(k,true,-)
14       n.next = c
15       p.next = n
16       return true
17     unlock p
18     unlock c
19     exit
20   else
21     unlock p
22     unlock c
```

add(6)

add(1)

```
add(k):
Node p, c
1  while (true)
2    p = H
3    c = p.next
4    while (c.key < k)
5      p = c
6      c = c.next
7    lock(p)
8    lock(c)
9    if (validate (p,c))
10     if (c.key=k)
11       return false
12     else
13       n = new Node(k,true,-)
14       n.next = c
15       p.next = n
16       return true
17     unlock p
18     unlock c
19     exit
20   else
21     unlock p
22     unlock c
```





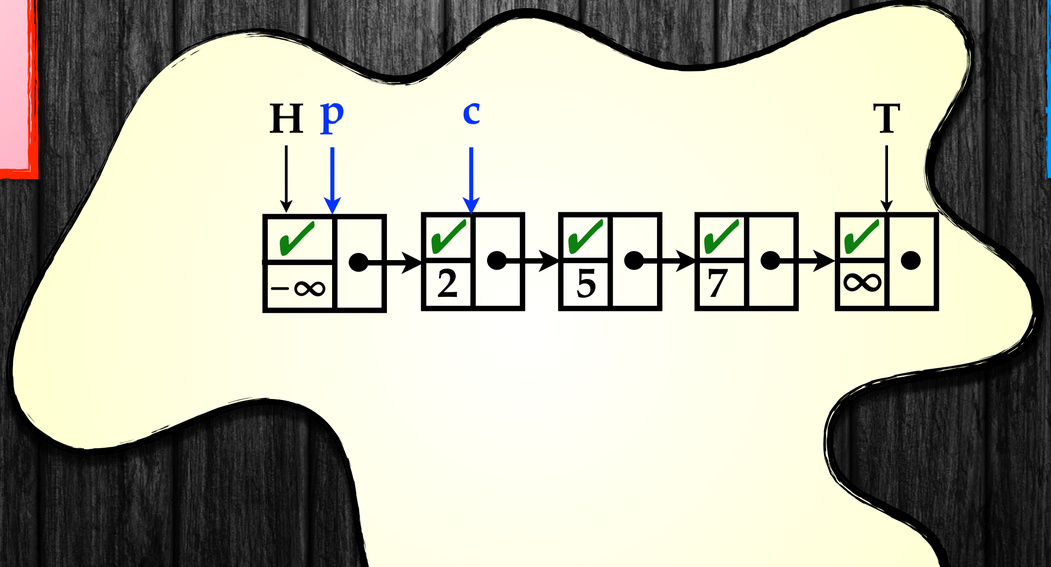
Sets | Optimistic Synchronization | Linearization Policy

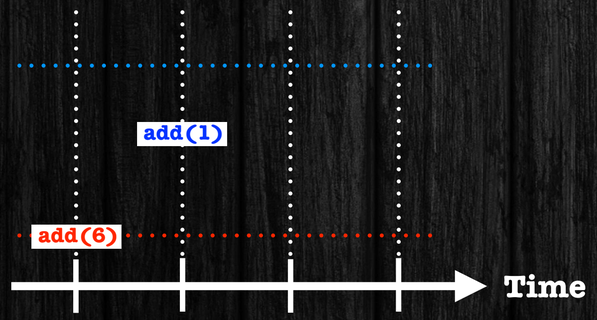
```
add(k):
Node p, c
1  while (true)
2    p = H
3    c = p.next
4    while (c.key < k)
5      p = c
6      c = c.next
7    lock(p)
8    lock(c)
9    if (validate (p,c))
10     if (c.key=k)
11       return false
12     else
13       n = new Node(k,true,-)
14       n.next = c
15       p.next = n
16       return true
17     unlock p
18     unlock c
19     exit
20   else
21     unlock p
22     unlock c
```

add(6)

add(1)

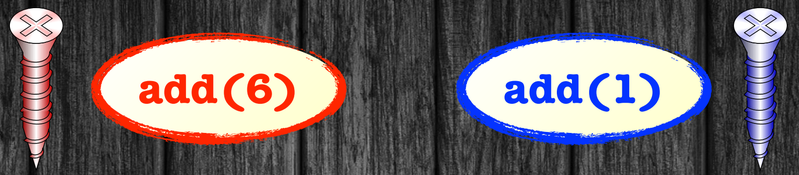
```
add(k):
Node p, c
1  while (true)
2    p = H
3    c = p.next
4    while (c.key < k)
5      p = c
6      c = c.next
7    lock(p)
8    lock(c)
9    if (validate (p,c))
10     if (c.key=k)
11       return false
12     else
13       n = new Node(k,true,-)
14       n.next = c
15       p.next = n
16       return true
17     unlock p
18     unlock c
19     exit
20   else
21     unlock p
22     unlock c
```



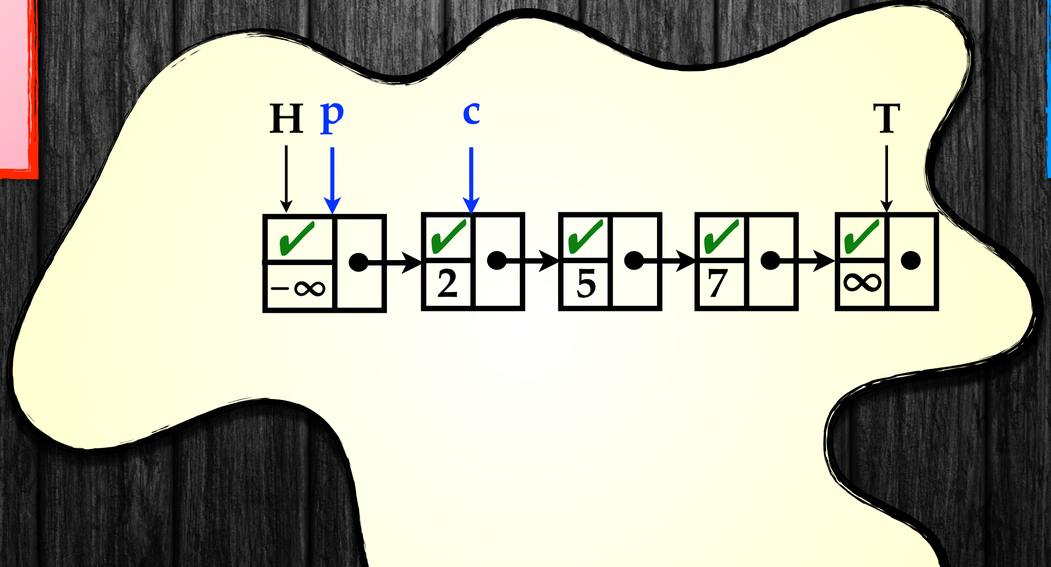


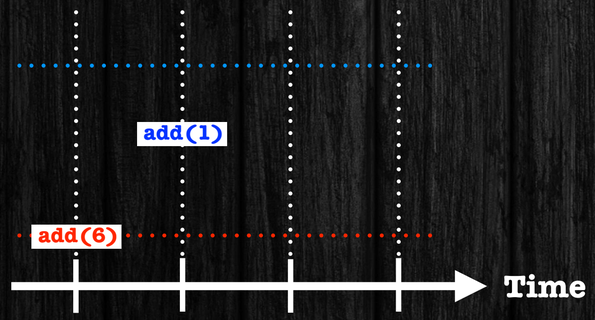
Sets | Optimistic Synchronization | Linearization Policy

```
add(k):
Node p, c
1  while (true)
2    p = H
3    c = p.next
4    while (c.key < k)
5      p = c
6      c = c.next
7    lock(p)
8    lock(c)
9    if (validate (p,c))
10     if (c.key=k)
11       return false
12     else
13       n = new Node(k,true,-)
14       n.next = c
15       p.next = n
16       return true
17     unlock p
18     unlock c
19     exit
20   else
21     unlock p
22     unlock c
```



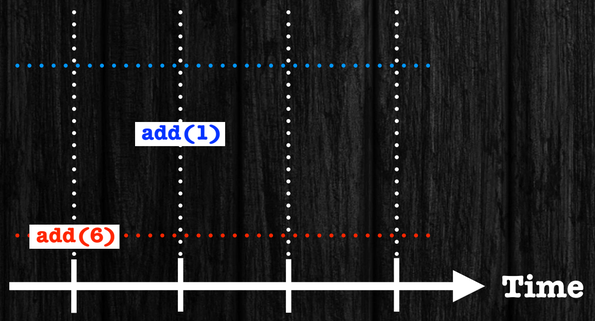

```
add(k):
Node p, c
1  while (true)
2    p = H
3    c = p.next
4    while (c.key < k)
5      p = c
6      c = c.next
7    lock(p)
8    lock(c)
9    if (validate (p,c))
10     if (c.key=k)
11       return false
12     else
13       n = new Node(k,true,-)
14       n.next = c
15       p.next = n
16       return true
17     unlock p
18     unlock c
19     exit
20   else
21     unlock p
22     unlock c
```

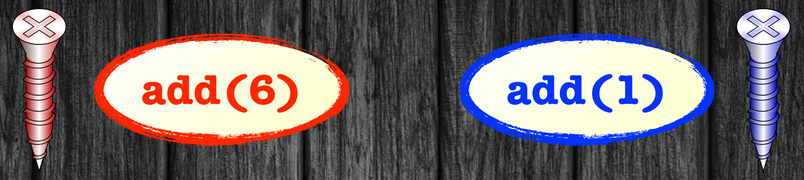


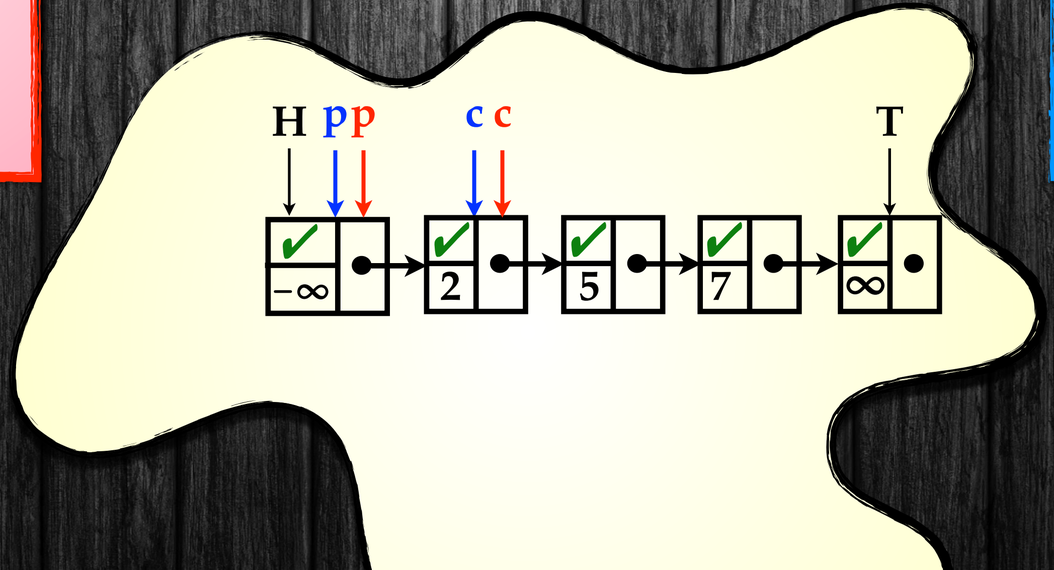


Sets | Optimistic Synchronization | Linearization Policy

```
add(k):
Node p, c
1  while (true)
2    p = H
3    c = p.next
4    while (c.key < k)
5      p = c
6      c = c.next
7    lock(p)
8    lock(c)
9    if (validate (p,c))
10     if (c.key=k)
11       return false
12     else
13       n = new Node(k,true,-)
14       n.next = c
15       p.next = n
16       return true
17     unlock p
18     unlock c
19     exit
20   else
21     unlock p
22     unlock c
```



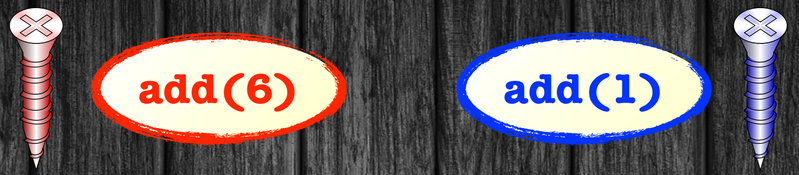


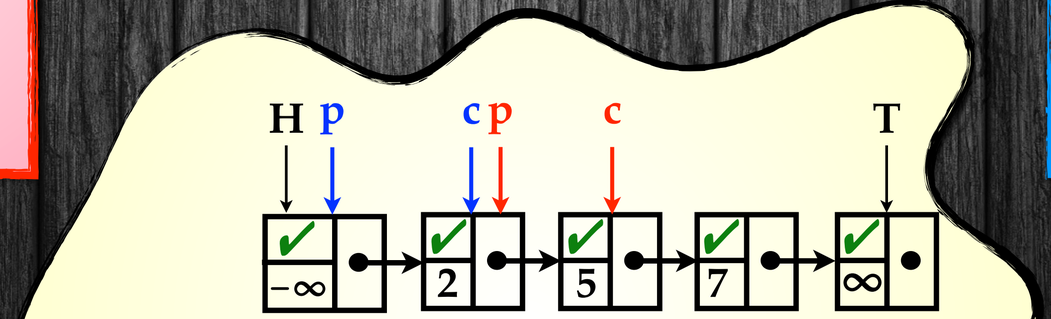


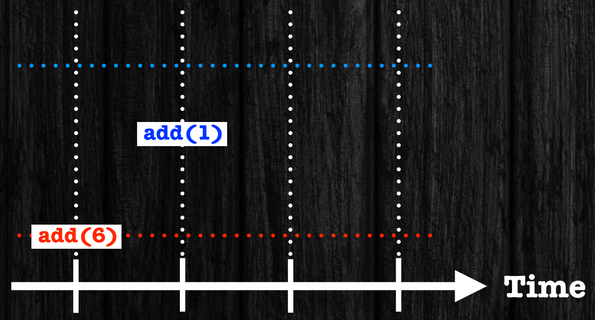

```
add(k):
Node p, c
1  while (true)
2    p = H
3    c = p.next
4    while (c.key < k)
5      p = c
6      c = c.next
7    lock(p)
8    lock(c)
9    if (validate (p,c))
10     if (c.key=k)
11       return false
12     else
13       n = new Node(k,true,-)
14       n.next = c
15       p.next = n
16       return true
17     unlock p
18     unlock c
19     exit
20   else
21     unlock p
22     unlock c
```

Sets | Optimistic Synchronization | Linearization Policy

```
add(k):
Node p, c
1  while (true)
2    p = H
3    c = p.next
4    while (c.key < k)
5      p = c
6      c = c.next
7    lock(p)
8    lock(c)
9    if (validate (p,c))
10     if (c.key=k)
11       return false
12     else
13       n = new Node(k,true,-)
14       n.next = c
15       p.next = n
16       return true
17     unlock p
18     unlock c
19     exit
20   else
21     unlock p
22     unlock c
```



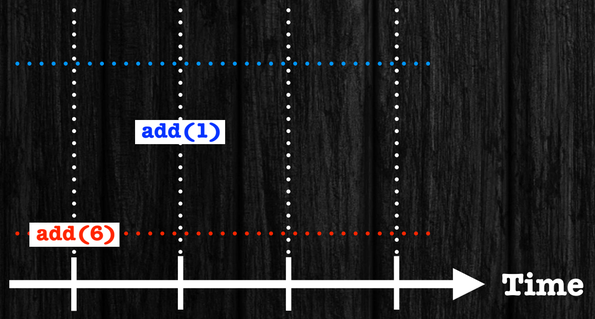


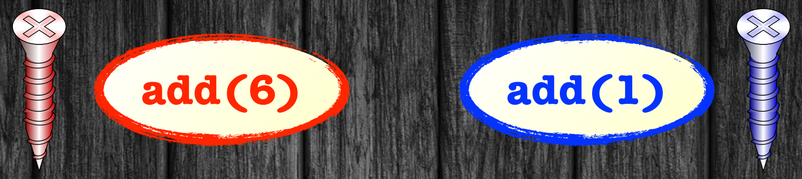


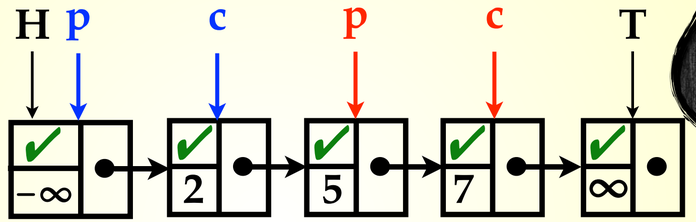

```
add(k):
Node p, c
1  while (true)
2    p = H
3    c = p.next
4    while (c.key < k)
5      p = c
6      c = c.next
7    lock(p)
8    lock(c)
9    if (validate (p,c))
10     if (c.key=k)
11       return false
12     else
13       n = new Node(k,true,-)
14       n.next = c
15       p.next = n
16       return true
17     unlock p
18     unlock c
19     exit
20   else
21     unlock p
22     unlock c
```

Sets | Optimistic Synchronization | Linearization Policy

```
add(k):
Node p, c
1  while (true)
2    p = H
3    c = p.next
4    while (c.key < k)
5      p = c
6      c = c.next
7    lock(p)
8    lock(c)
9    if (validate (p,c))
10     if (c.key=k)
11       return false
12     else
13       n = new Node(k,true,-)
14       n.next = c
15       p.next = n
16       return true
17     unlock p
18     unlock c
19     exit
20   else
21     unlock p
22     unlock c
```



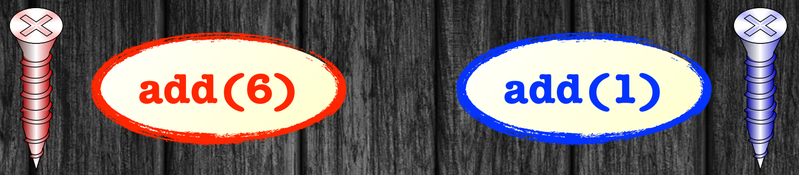




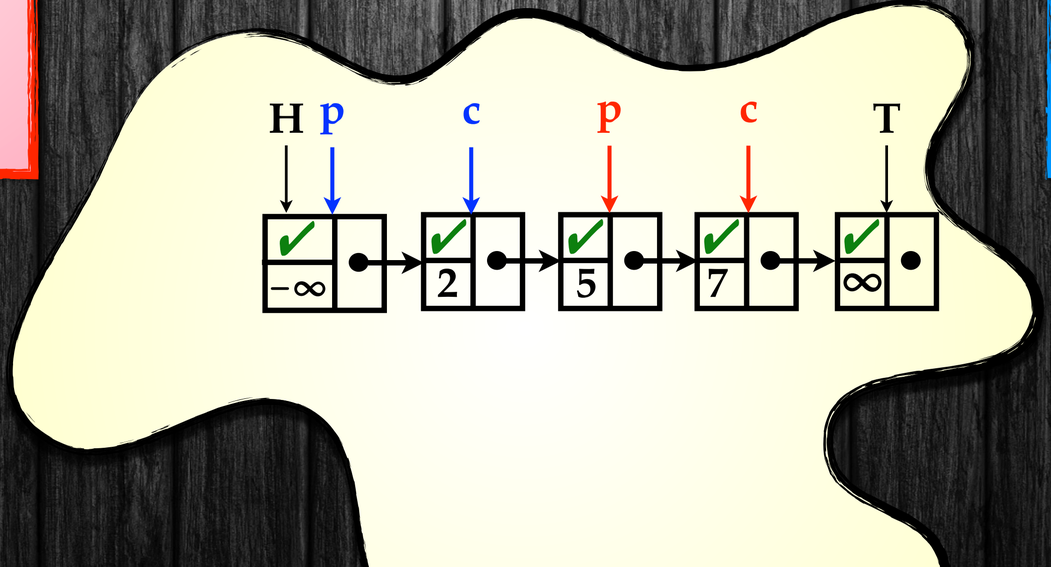

```
add(k):
Node p, c
1  while (true)
2    p = H
3    c = p.next
4    while (c.key < k)
5      p = c
6      c = c.next
7    lock(p)
8    lock(c)
9    if (validate (p,c))
10     if (c.key=k)
11       return false
12     else
13       n = new Node(k,true,-)
14       n.next = c
15       p.next = n
16       return true
17     unlock p
18     unlock c
19     exit
20   else
21     unlock p
22     unlock c
```

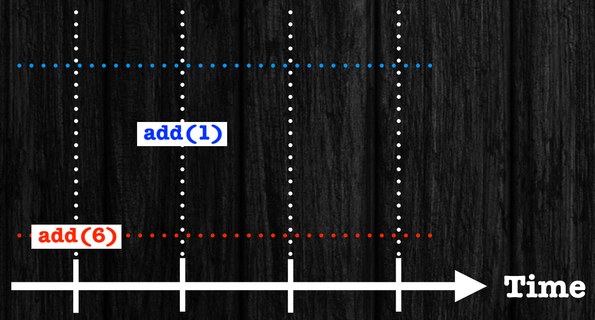
Sets | Optimistic Synchronization | Linearization Policy

```
add(k):
Node p, c
1  while (true)
2    p = H
3    c = p.next
4    while (c.key < k)
5      p = c
6      c = c.next
7    lock(p)
8    lock(c)
9    if (validate (p,c))
10     if (c.key=k)
11       return false
12     else
13       n = new Node(k,true,-)
14       n.next = c
15       p.next = n
16       return true
17     unlock p
18     unlock c
19     exit
20   else
21     unlock p
22     unlock c
```



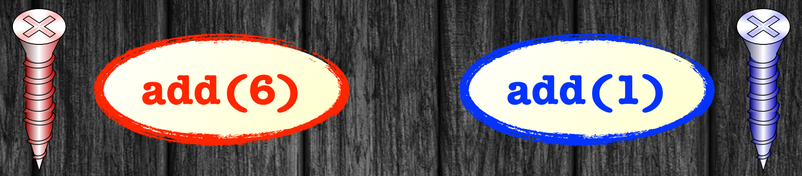

```
add(k):
Node p, c
1  while (true)
2    p = H
3    c = p.next
4    while (c.key < k)
5      p = c
6      c = c.next
7    lock(p)
8    lock(c)
9    if (validate (p,c))
10     if (c.key=k)
11       return false
12     else
13       n = new Node(k,true,-)
14       n.next = c
15       p.next = n
16       return true
17     unlock p
18     unlock c
19     exit
20   else
21     unlock p
22     unlock c
```



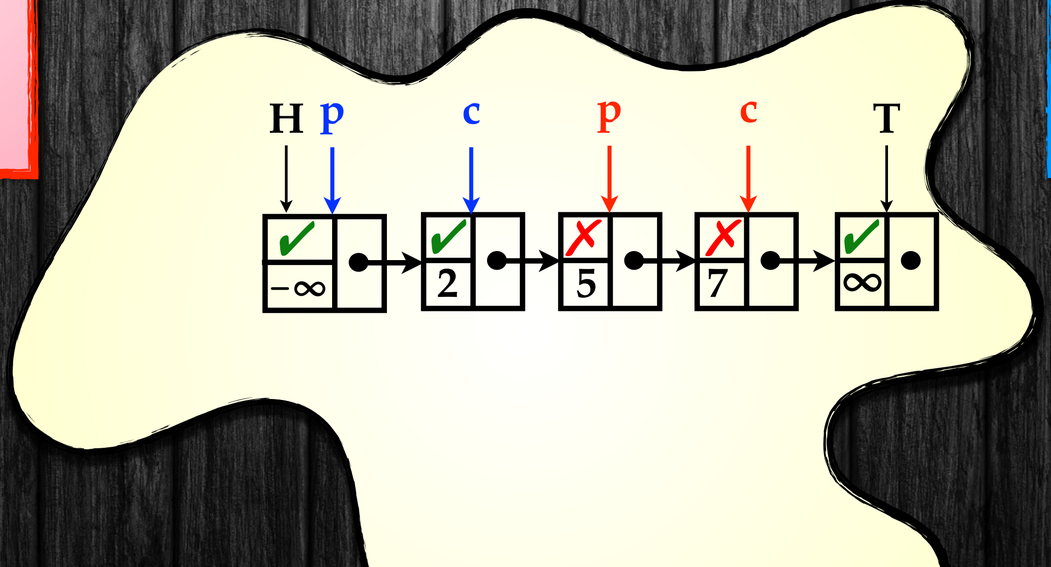


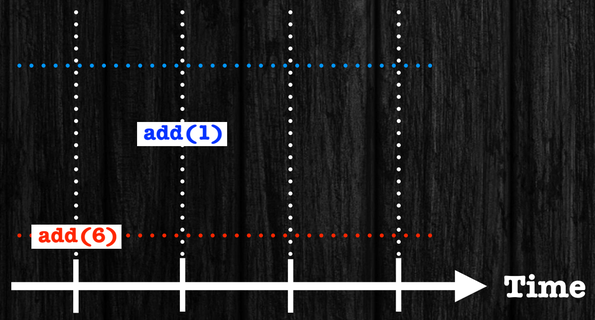
Sets | Optimistic Synchronization | Linearization Policy

```
add(k):
Node p, c
1  while (true)
2    p = H
3    c = p.next
4    while (c.key < k)
5      p = c
6      c = c.next
7    lock(p)
8    lock(c)
9    if (validate (p,c))
10     if (c.key=k)
11       return false
12     else
13       n = new Node(k,true,-)
14       n.next = c
15       p.next = n
16       return true
17     unlock p
18     unlock c
19     exit
20   else
21     unlock p
22     unlock c
```



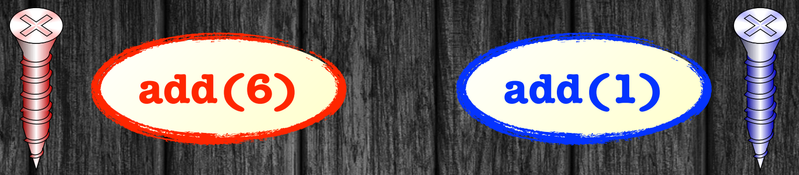

```
add(k):
Node p, c
1  while (true)
2    p = H
3    c = p.next
4    while (c.key < k)
5      p = c
6      c = c.next
7    lock(p)
8    lock(c)
9    if (validate (p,c))
10     if (c.key=k)
11       return false
12     else
13       n = new Node(k,true,-)
14       n.next = c
15       p.next = n
16       return true
17     unlock p
18     unlock c
19     exit
20   else
21     unlock p
22     unlock c
```

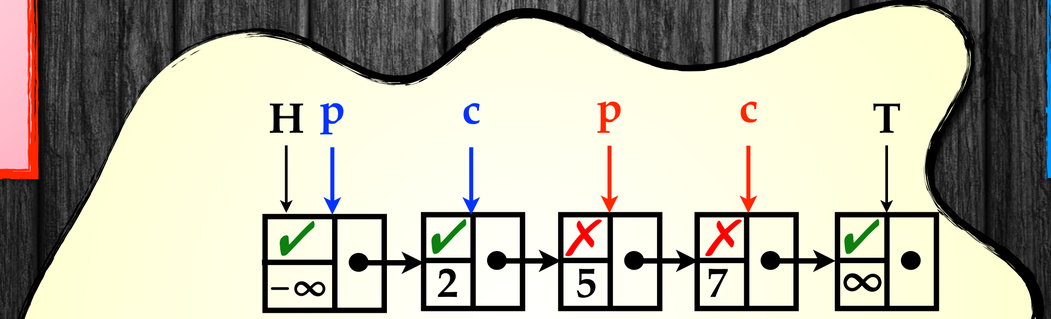


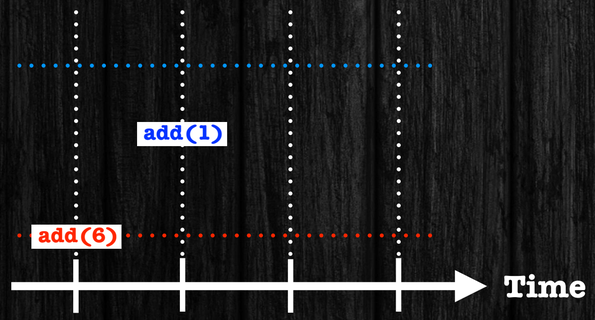


Sets | Optimistic Synchronization | Linearization Policy

```
add(k):
Node p, c
1  while (true)
2    p = H
3    c = p.next
4    while (c.key < k)
5      p = c
6      c = c.next
7    lock(p)
8    lock(c)
9    if (validate (p,c))
10     if (c.key=k)
11       return false
12     else
13       n = new Node(k,true,-)
14       n.next = c
15       p.next = n
16       return true
17     unlock p
18     unlock c
19     exit
20   else
21     unlock p
22     unlock c
```



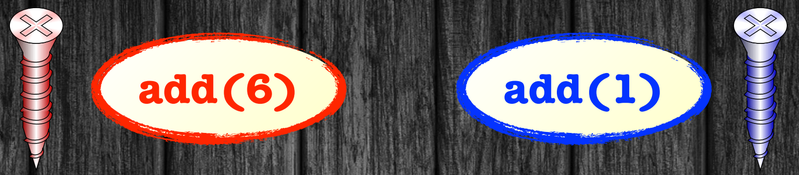


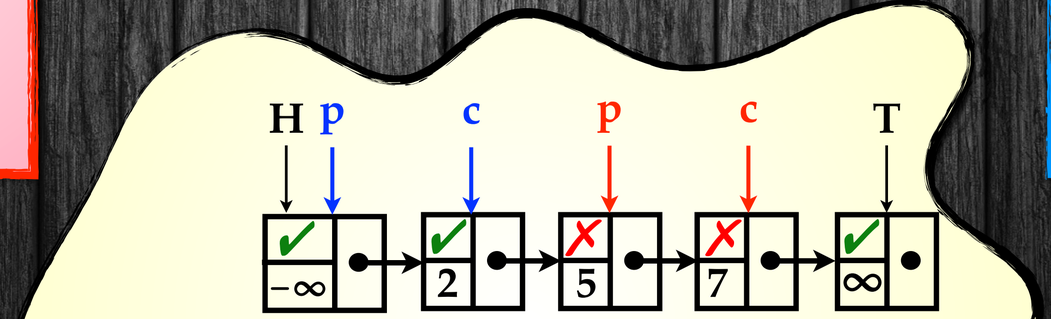


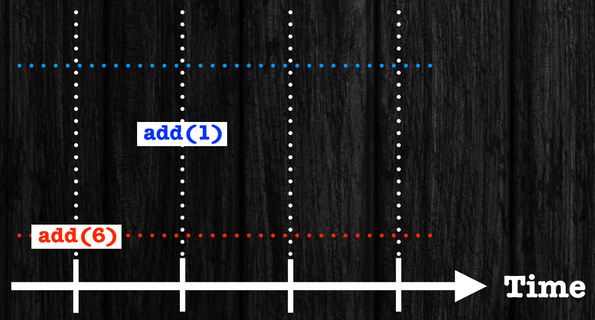

```
add(k):
Node p, c
1  while (true)
2    p = H
3    c = p.next
4    while (c.key < k)
5      p = c
6      c = c.next
7    lock(p)
8    lock(c)
9    if (validate (p,c))
10     if (c.key=k)
11       return false
12     else
13       n = new Node(k,true,-)
14       n.next = c
15       p.next = n
16       return true
17     unlock p
18     unlock c
19     exit
20   else
21     unlock p
22     unlock c
```

Sets | Optimistic Synchronization | Linearization Policy

```
add(k):
Node p, c
1  while (true)
2    p = H
3    c = p.next
4    while (c.key < k)
5      p = c
6      c = c.next
7    lock(p)
8    lock(c)
9    if (validate (p,c)) ←
10     if (c.key=k)
11       return false
12     else
13       n = new Node(k,true,-)
14       n.next = c
15       p.next = n
16       return true
17     unlock p
18     unlock c
19     exit
20   else
21     unlock p
22     unlock c
```



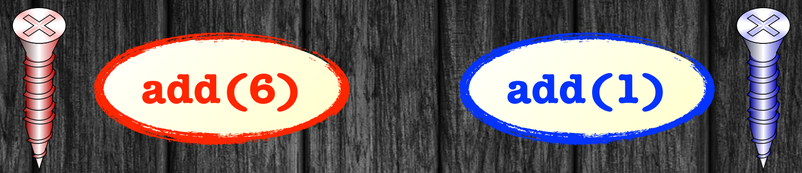


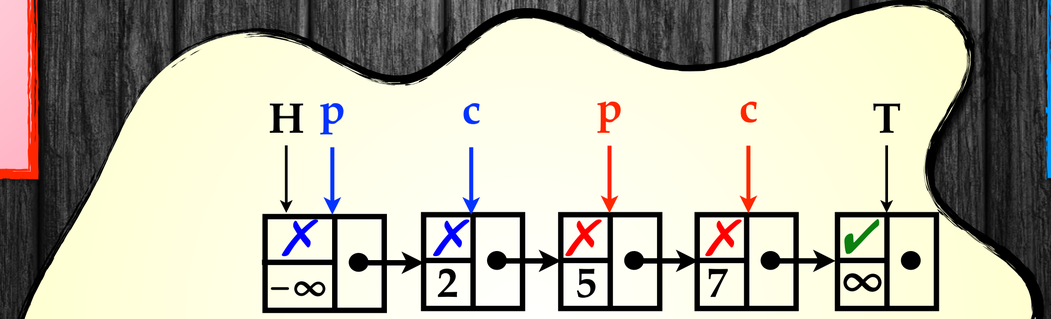


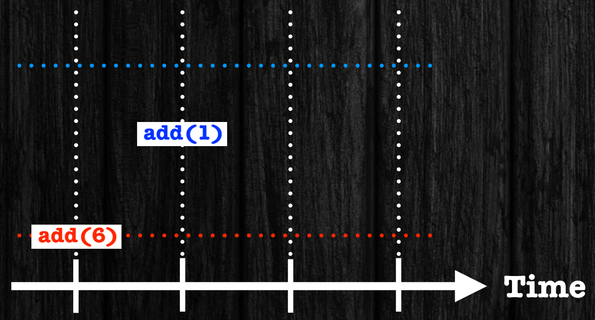

```
add(k):
Node p, c
1  while (true)
2    p = H
3    c = p.next
4    while (c.key < k)
5      p = c
6      c = c.next
7    lock(p) ←
8    lock(c)
9    if (validate (p,c))
10     if (c.key=k)
11       return false
12     else
13       n = new Node(k,true,-)
14       n.next = c
15       p.next = n
16       return true
17     unlock p
18     unlock c
19     exit
20   else
21     unlock p
22     unlock c
```

Sets | Optimistic Synchronization | Linearization Policy

```
add(k):
Node p, c
1  while (true)
2    p = H
3    c = p.next
4    while (c.key < k)
5      p = c
6      c = c.next
7    lock(p)
8    lock(c)
9    if (validate (p,c))
10     if (c.key=k)
11       return false
12     else
13       n = new Node(k,true,-)
14       n.next = c
15       p.next = n
16       return true
17     unlock p
18     unlock c
19     exit
20   else
21     unlock p
22     unlock c
```



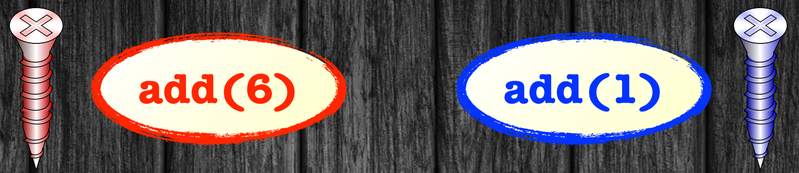


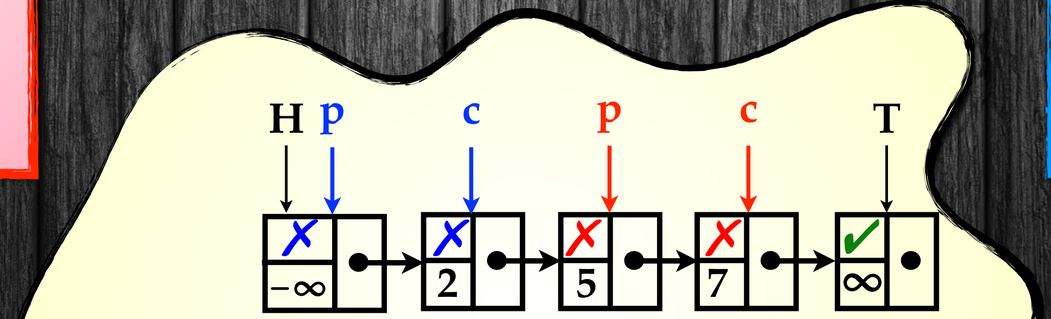


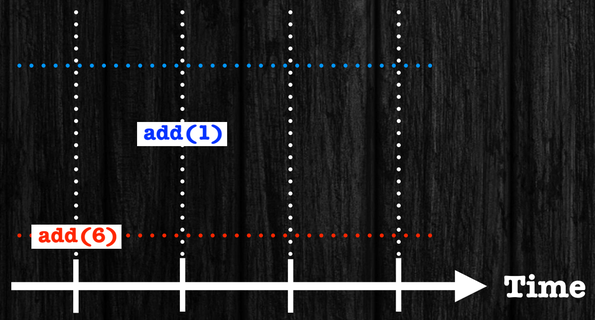

```
add(k):
Node p, c
1  while (true)
2    p = H
3    c = p.next
4    while (c.key < k)
5      p = c
6      c = c.next
7    lock(p)
8    lock(c)
9    if (validate (p,c))
10     if (c.key=k)
11       return false
12     else
13       n = new Node(k,true,-)
14       n.next = c
15       p.next = n
16       return true
17     unlock p
18     unlock c
19     exit
20   else
21     unlock p
22     unlock c
```

Sets | Optimistic Synchronization | Linearization Policy

```
add(k):
Node p, c
1  while (true)
2    p = H
3    c = p.next
4    while (c.key < k)
5      p = c
6      c = c.next
7    lock(p)
8    lock(c)
9    if (validate (p,c))
10     if (c.key=k)
11       return false
12     else
13       n = new Node(k,true,-)
14       n.next = c
15       p.next = n
16       return true
17     unlock p
18     unlock c
19     exit
20   else
21     unlock p
22     unlock c
```



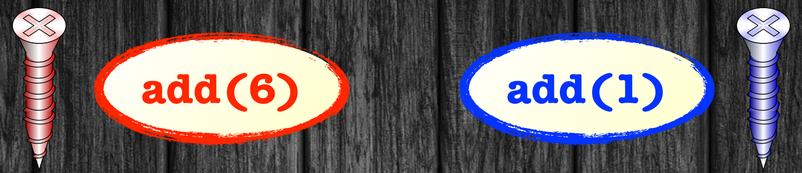


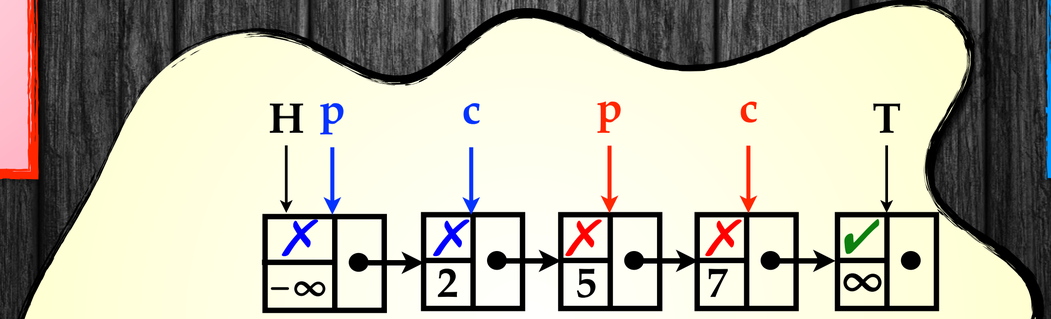


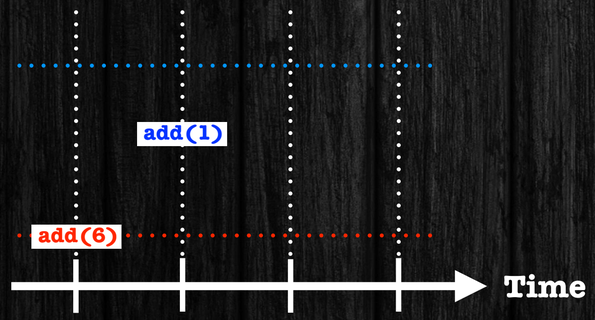

```
add(k):
Node p, c
1  while (true)
2    p = H
3    c = p.next
4    while (c.key < k)
5      p = c
6      c = c.next
7    lock(p)
8    lock(c)
9    if (validate (p,c))
10     if (c.key=k)
11       return false
12     else
13       n = new Node(k,true,-)
14       n.next = c
15       p.next = n
16       return true
17     unlock p
18     unlock c
19     exit
20   else
21     unlock p
22     unlock c
```

Sets | Optimistic Synchronization | Linearization Policy

```
add(k):
Node p, c
1  while (true)
2    p = H
3    c = p.next
4    while (c.key < k)
5      p = c
6      c = c.next
7    lock(p)
8    lock(c)
9    if (validate (p,c))
10     if (c.key=k)
11       return false
12     else
13       n = new Node(k,true,-)
14       n.next = c
15       p.next = n
16       return true
17     unlock p
18     unlock c
19     exit
20   else
21     unlock p
22     unlock c
```








```
add(k):
Node p, c
1  while (true)
2    p = H
3    c = p.next
4    while (c.key < k)
5      p = c
6      c = c.next
7    lock(p)
8    lock(c)
9    if (validate (p,c))
10     if (c.key=k)
11       return false
12     else
13       n = new Node(k,true,-)
14       n.next = c
15       p.next = n
16       return true
17     unlock p
18     unlock c
19     exit
20   else
21     unlock p
22     unlock c
```

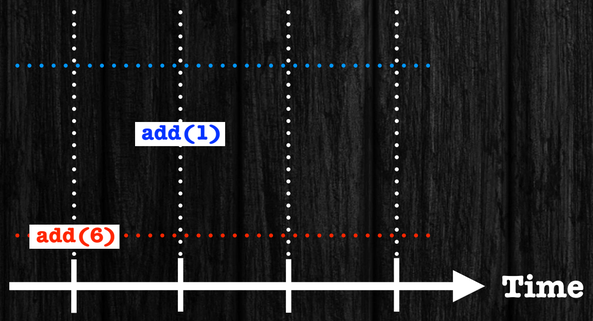
Sets | Optimistic Synchronization | Linearization Policy

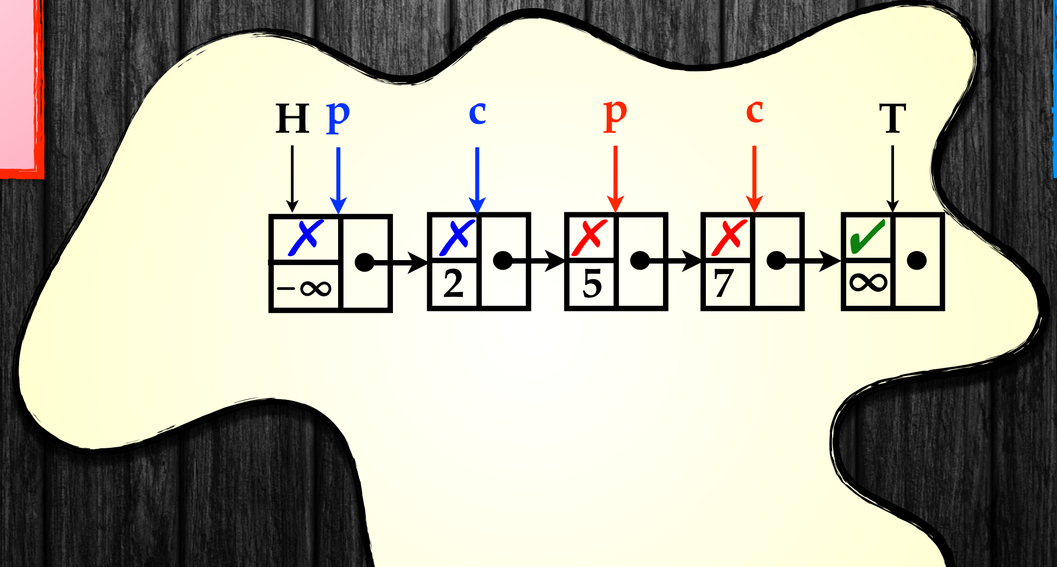
```
add(k):
Node p, c
1  while (true)
2    p = H
3    c = p.next
4    while (c.key < k)
5      p = c
6      c = c.next
7    lock(p)
8    lock(c)
9    if (validate (p,c))
10     if (c.key=k)
11       return false
12     else
13       n = new Node(k,true,-)
14       n.next = c
15       p.next = n
16       return true
17     unlock p
18     unlock c
19     exit
20   else
21     unlock p
22     unlock c
```

add(6)

add(1)

validation






```
add(k):
Node p, c
1  while (true)
2    p = H
3    c = p.next
4    while (c.key < k)
5      p = c
6      c = c.next
7    lock(p)
8    lock(c)
9    if (validate (p,c))
10     if (c.key=k)
11       return false
12     else
13       n = new Node(k,true,-)
14       n.next = c
15       p.next = n
16       return true
17     unlock p
18     unlock c
19     exit
20   else
21     unlock p
22     unlock c
```

Sets | Optimistic Synchronization | Linearization Policy

```
add(k):
Node p, c
1  while (true)
2    p = H
3    c = p.next
4    while (c.key < k)
5      p = c
6      c = c.next
7    lock(p)
8    lock(c)
9    if (validate (p,c))
10     if (c.key=k)
11       return false
12     else
13       n = new Node(k,true,-)
14       n.next = c
15       p.next = n
16       return true
17     unlock p
18     unlock c
19     exit
20   else
21     unlock p
22     unlock c
```

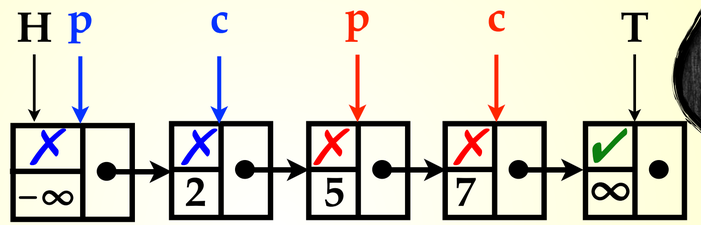
add(6)

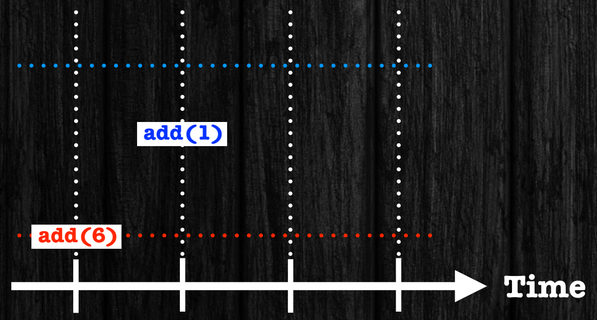
add(1)

validation

check wether the two cells are still in the list

```
add(k):
Node p, c
1  while (true)
2    p = H
3    c = p.next
4    while (c.key < k)
5      p = c
6      c = c.next
7    lock(p)
8    lock(c)
9    if (validate (p,c))
10     if (c.key=k)
11       return false
12     else
13       n = new Node(k,true,-)
14       n.next = c
15       p.next = n
16       return true
17     unlock p
18     unlock c
19     exit
20   else
21     unlock p
22     unlock c
```





Sets | Optimistic Synchronization | Linearization Policy

```
add(k):
Node p, c
1  while (true)
2    p = H
3    c = p.next
4    while (c.key < k)
5      p = c
6      c = c.next
7    lock(p)
8    lock(c)
9    if (validate (p,c))
10     if (c.key=k)
11       return false
12     else
13       n = new Node(k,true,-)
14       n.next = c
15       p.next = n
16       return true
17     unlock p
18     unlock c
19     exit
20   else
21     unlock p
22     unlock c
```

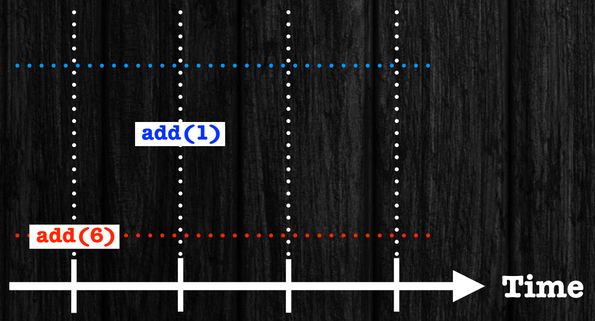
add(6)

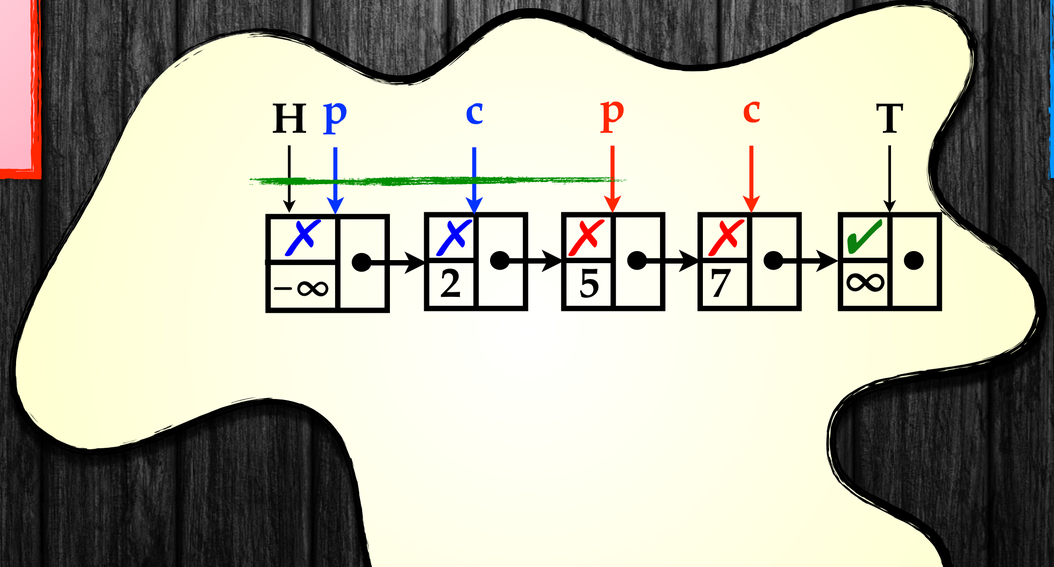
add(1)

validation

check wether the two cells are still in the list

```
add(k):
Node p, c
1  while (true)
2    p = H
3    c = p.next
4    while (c.key < k)
5      p = c
6      c = c.next
7    lock(p)
8    lock(c)
9    if (validate (p,c))
10     if (c.key=k)
11       return false
12     else
13       n = new Node(k,true,-)
14       n.next = c
15       p.next = n
16       return true
17     unlock p
18     unlock c
19     exit
20   else
21     unlock p
22     unlock c
```





Sets | Optimistic Synchronization | Linearization Policy

```
add(k):
Node p, c
1  while (true)
2    p = H
3    c = p.next
4    while (c.key < k)
5      p = c
6      c = c.next
7    lock(p)
8    lock(c)
9    if (validate (p,c))
10     if (c.key=k)
11       return false
12     else
13       n = new Node(k,true,-)
14       n.next = c
15       p.next = n
16       return true
17     unlock p
18     unlock c
19     exit
20   else
21     unlock p
22     unlock c
```

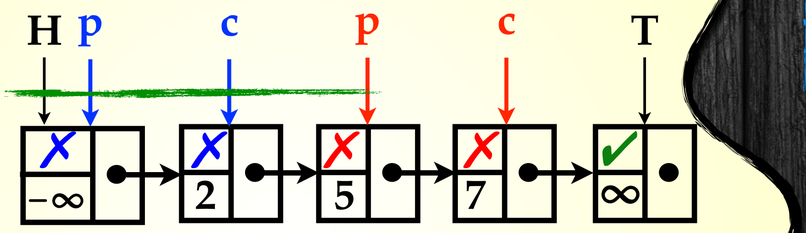
add(6)

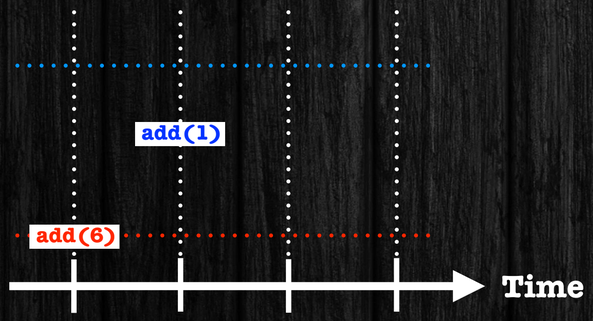
add(1)

validation

check wether the two cells are still in the list

yes






```
add(k):
Node p, c
1  while (true)
2    p = H
3    c = p.next
4    while (c.key < k)
5      p = c
6      c = c.next
7    lock(p)
8    lock(c)
9    if (validate (p,c))
10     if (c.key=k)
11       return false
12     else
13       n = new Node(k,true,-)
14       n.next = c
15       p.next = n
16       return true
17     unlock p
18     unlock c
19     exit
20   else
21     unlock p
22     unlock c
```

Sets | Optimistic Synchronization | Linearization Policy

```
add(k):
Node p, c
1  while (true)
2    p = H
3    c = p.next
4    while (c.key < k)
5      p = c
6      c = c.next
7    lock(p)
8    lock(c)
9    if (validate (p,c))
10     if (c.key=k)
11       return false
12     else
13       n = new Node(k,true,-)
14       n.next = c
15       p.next = n
16       return true
17     unlock p
18     unlock c
19     exit
20   else
21     unlock p
22     unlock c
```

add(6)

add(1)

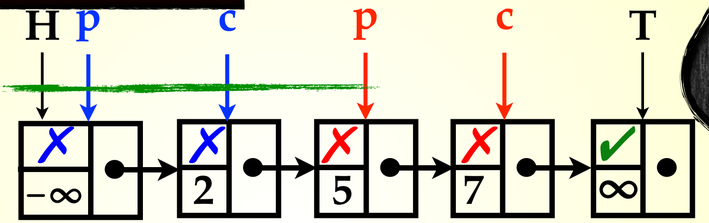
validation

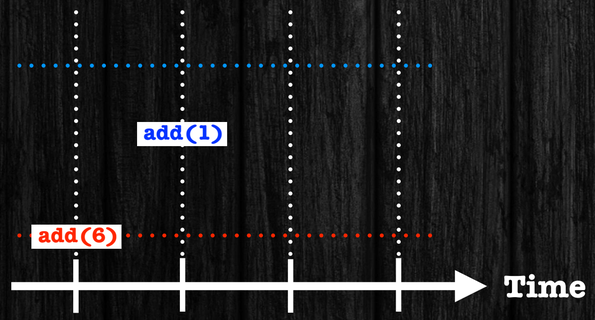
check wether the two cells are still in the list

yes

LP

"time point at which validation succeeds"






```
add(k):
Node p, c
1  while (true)
2    p = H
3    c = p.next
4    while (c.key < k)
5      p = c
6      c = c.next
7    lock(p)
8    lock(c)
9    if (validate (p,c))
10     if (c.key=k)
11       return false
12     else
13       n = new Node(k,true,-)
14       n.next = c
15       p.next = n
16       return true
17     unlock p
18     unlock c
19     exit
20   else
21     unlock p
22     unlock c
```

Sets | Optimistic Synchronization | Linearization Policy

```
add(k):
Node p, c
1  while (true)
2    p = H
3    c = p.next
4    while (c.key < k)
5      p = c
6      c = c.next
7    lock(p)
8    lock(c)
9    if (validate (p,c))
10     if (c.key=k)
11       return false
12     else
13       n = new Node(k,true,-)
14       n.next = c
15       p.next = n
16       return true
17     unlock p
18     unlock c
19     exit
20   else
21     unlock p
22     unlock c
```

add(6)

add(1)

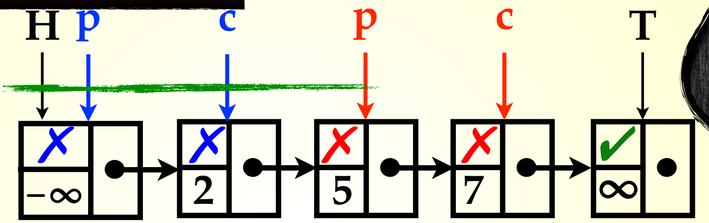
validation

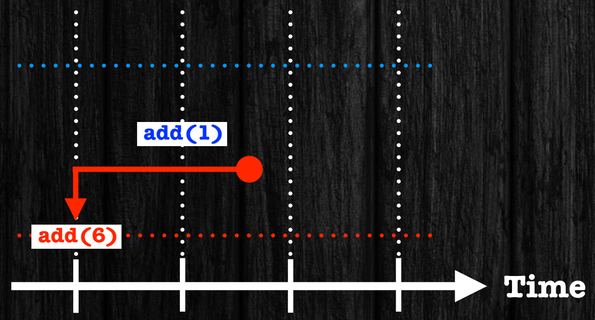
check wether the two cells are still in the list

yes

LP

"time point at which validation succeeds"

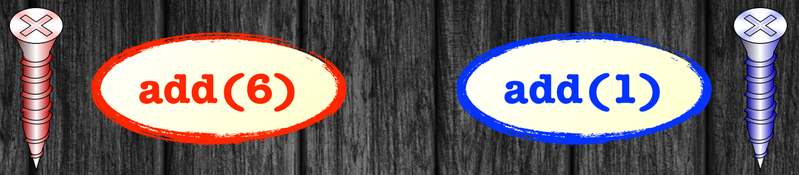




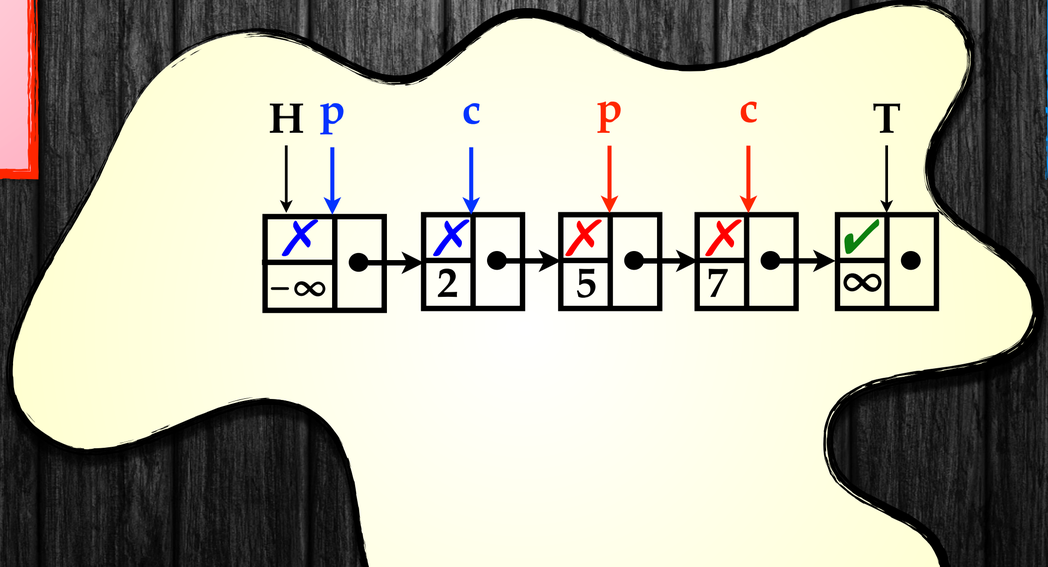

```
add(k):
Node p, c
1  while (true)
2    p = H
3    c = p.next
4    while (c.key < k)
5      p = c
6      c = c.next
7    lock(p)
8    lock(c)
9    if (validate (p,c))
10     if (c.key=k)
11       return false
12     else
13       n = new Node(k,true,-)
14       n.next = c
15       p.next = n
16       return true
17     unlock p
18     unlock c
19     exit
20   else
21     unlock p
22     unlock c
```

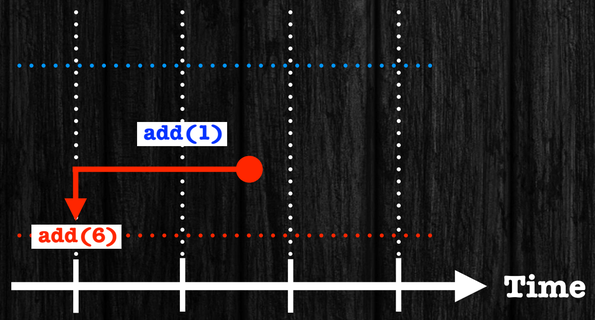
Sets | Optimistic Synchronization | Linearization Policy

```
add(k):
Node p, c
1  while (true)
2    p = H
3    c = p.next
4    while (c.key < k)
5      p = c
6      c = c.next
7    lock(p)
8    lock(c)
9    if (validate (p,c))
10     if (c.key=k)
11       return false
12     else
13       n = new Node(k,true,-)
14       n.next = c
15       p.next = n
16       return true
17     unlock p
18     unlock c
19     exit
20   else
21     unlock p
22     unlock c
```



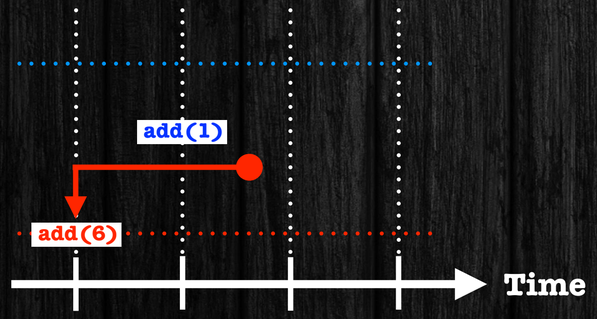

```
add(k):
Node p, c
1  while (true)
2    p = H
3    c = p.next
4    while (c.key < k)
5      p = c
6      c = c.next
7    lock(p)
8    lock(c)
9    if (validate (p,c))
10     if (c.key=k)
11       return false
12     else
13       n = new Node(k,true,-)
14       n.next = c
15       p.next = n
16       return true
17     unlock p
18     unlock c
19     exit
20   else
21     unlock p
22     unlock c
```

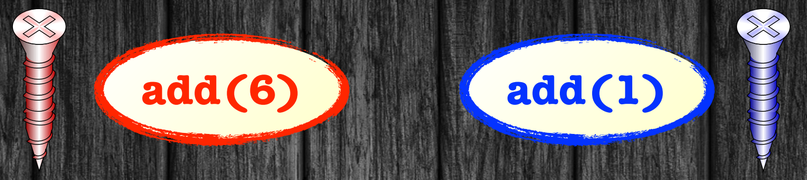


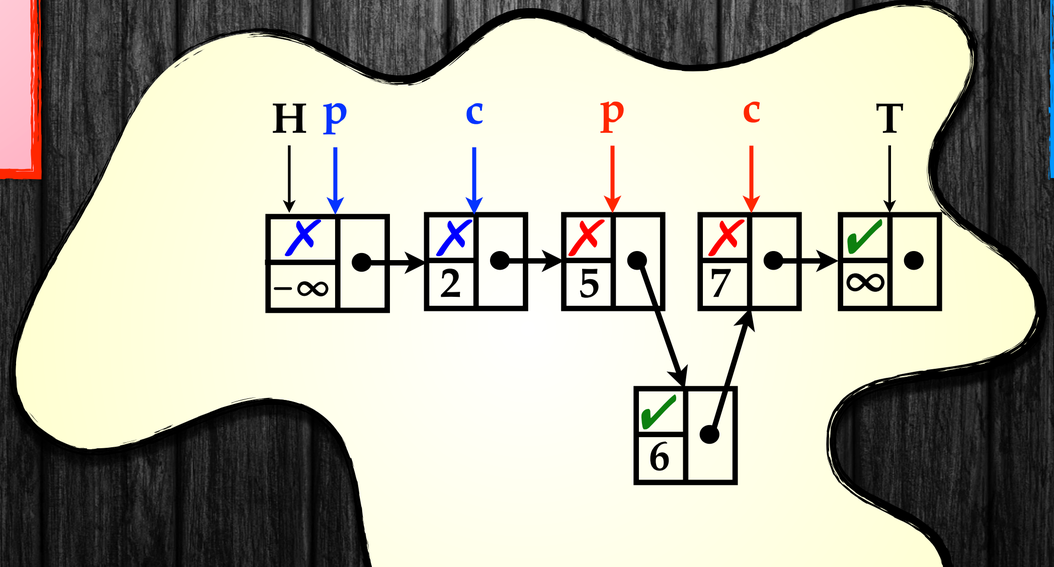


Sets | Optimistic Synchronization | Linearization Policy

```
add(k):
Node p, c
1  while (true)
2    p = H
3    c = p.next
4    while (c.key < k)
5      p = c
6      c = c.next
7    lock(p)
8    lock(c)
9    if (validate (p,c))
10     if (c.key=k)
11       return false
12     else
13       n = new Node(k,true,-)
14       n.next = c
15       p.next = n
16       return true
17     unlock p
18     unlock c
19     exit
20   else
21     unlock p
22     unlock c
```








```
add(k):
Node p, c
1  while (true)
2    p = H
3    c = p.next
4    while (c.key < k)
5      p = c
6      c = c.next
7    lock(p)
8    lock(c)
9    if (validate (p,c))
10     if (c.key=k)
11       return false
12     else
13       n = new Node(k,true,-)
14       n.next = c
15       p.next = n
16       return true
17     unlock p
18     unlock c
19     exit
20   else
21     unlock p
22     unlock c
```

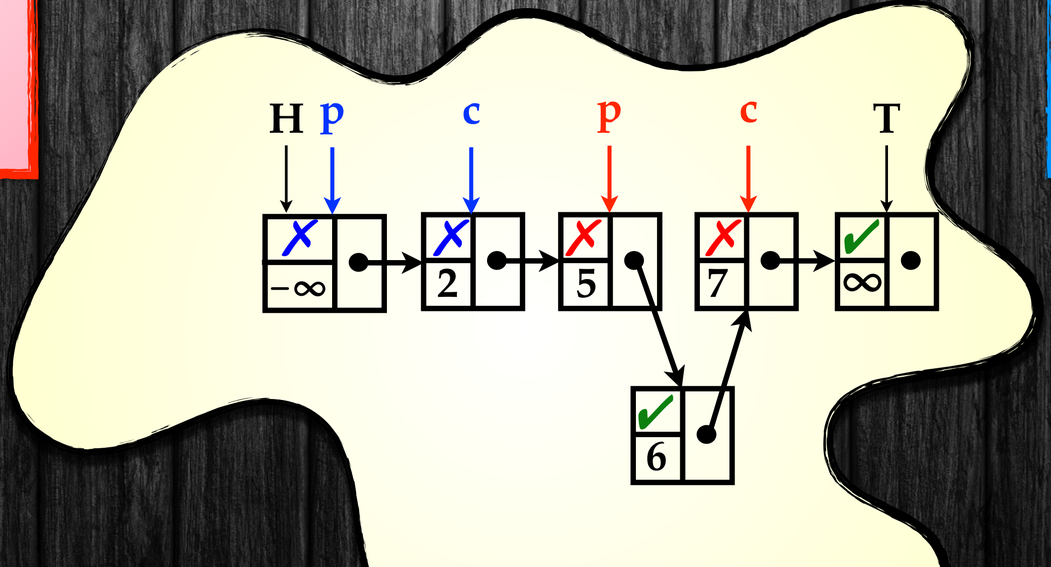
Sets | Optimistic Synchronization | Linearization Policy

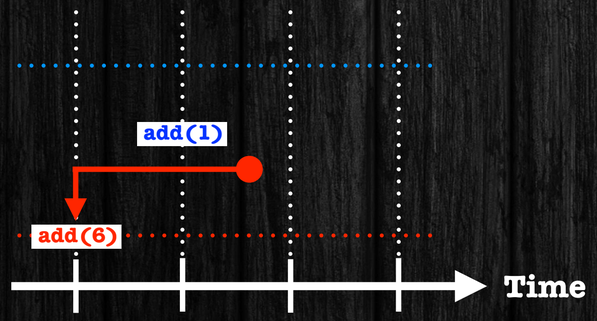
```
add(k):
Node p, c
1  while (true)
2    p = H
3    c = p.next
4    while (c.key < k)
5      p = c
6      c = c.next
7    lock(p)
8    lock(c)
9    if (validate (p,c))
10     if (c.key=k)
11       return false
12     else
13       n = new Node(k,true,-)
14       n.next = c
15       p.next = n
16       return true
17     unlock p
18     unlock c
19     exit
20   else
21     unlock p
22     unlock c
```

add(6)

add(1)

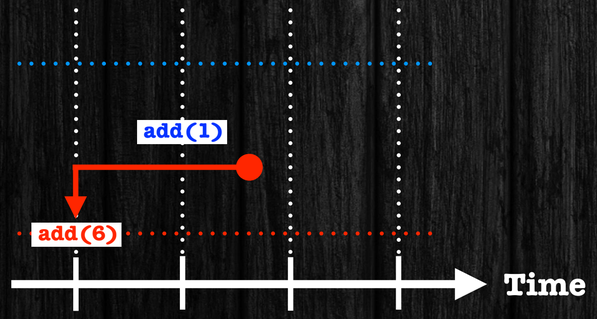
```
add(k):
Node p, c
1  while (true)
2    p = H
3    c = p.next
4    while (c.key < k)
5      p = c
6      c = c.next
7    lock(p)
8    lock(c)
9    if (validate (p,c))
10     if (c.key=k)
11       return false
12     else
13       n = new Node(k,true,-)
14       n.next = c
15       p.next = n
16       return true
17     unlock p
18     unlock c
19     exit
20   else
21     unlock p
22     unlock c
```

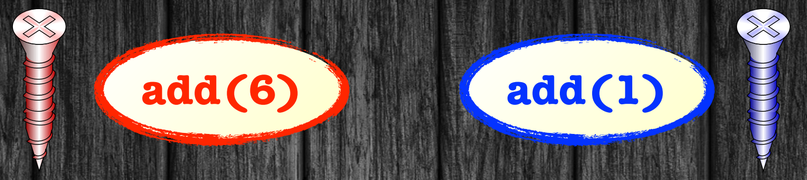


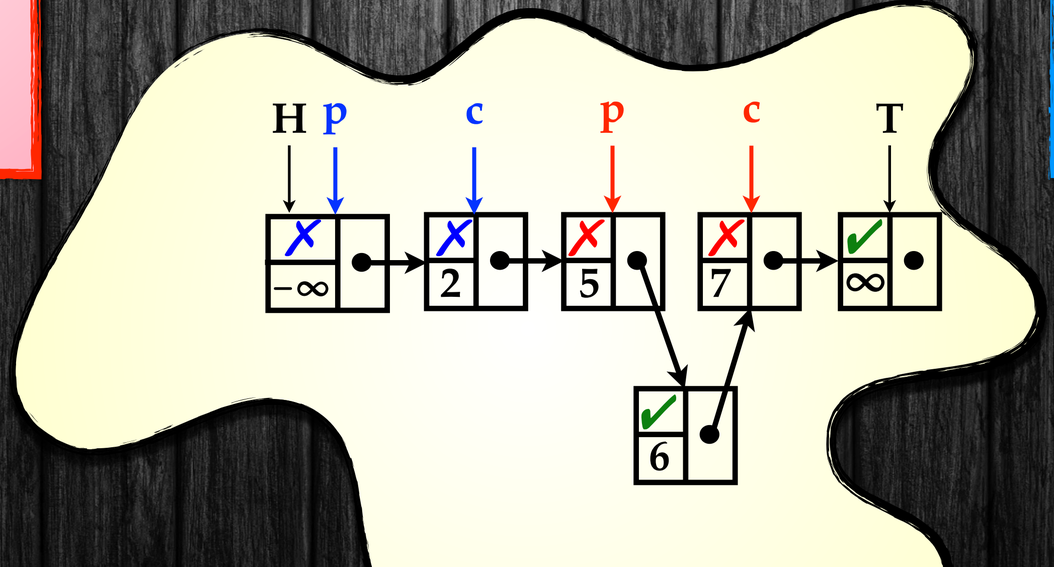


Sets | Optimistic Synchronization | Linearization Policy

```
add(k):
Node p, c
1  while (true)
2    p = H
3    c = p.next
4    while (c.key < k)
5      p = c
6      c = c.next
7    lock(p)
8    lock(c)
9    if (validate (p,c))
10     if (c.key=k)
11       return false
12     else
13       n = new Node(k,true,-)
14       n.next = c
15       p.next = n
16       return true
17     unlock p
18     unlock c
19     exit
20   else
21     unlock p
22     unlock c
```








```
add(k):
Node p, c
1  while (true)
2    p = H
3    c = p.next
4    while (c.key < k)
5      p = c
6      c = c.next
7    lock(p)
8    lock(c)
9    if (validate (p,c))
10     if (c.key=k)
11       return false
12     else
13       n = new Node(k,true,-)
14       n.next = c
15       p.next = n
16       return true
17     unlock p
18     unlock c
19     exit
20   else
21     unlock p
22     unlock c
```

Sets | Optimistic Synchronization | Linearization Policy

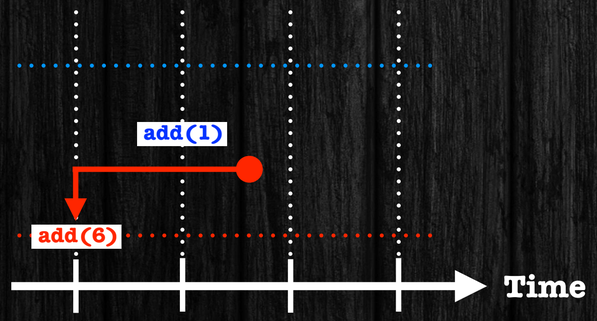
```
add(k):
Node p, c
1  while (true)
2    p = H
3    c = p.next
4    while (c.key < k)
5      p = c
6      c = c.next
7    lock(p)
8    lock(c)
9    if (validate (p,c))
10     if (c.key=k)
11       return false
12     else
13       n = new Node(k,true,-)
14       n.next = c
15       p.next = n
16       return true
17     unlock p
18     unlock c
19     exit
20   else
21     unlock p
22     unlock c
```

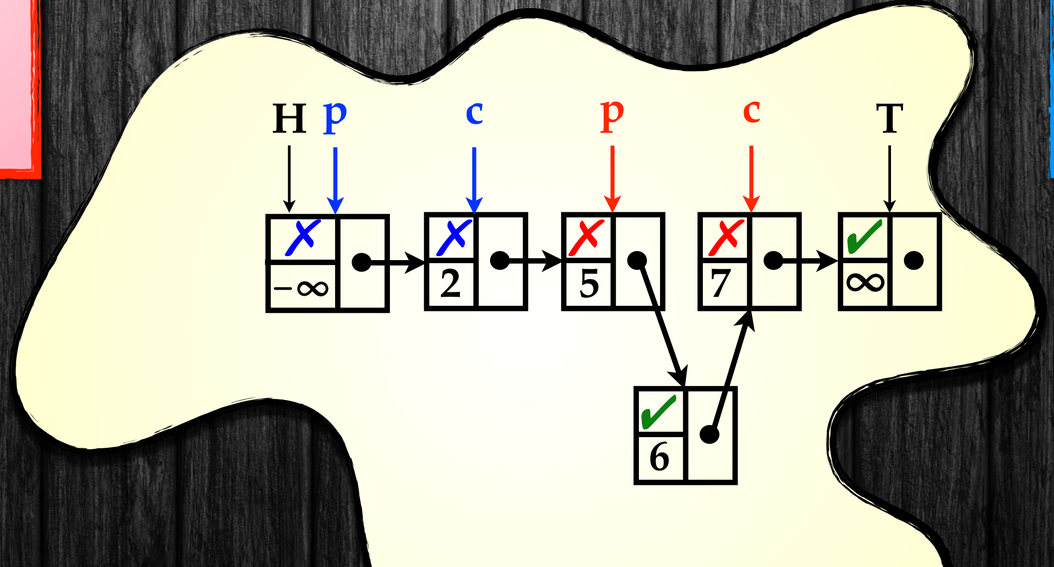
add(6)

add(1)

```
add(k):
Node p, c
1  while (true)
2    p = H
3    c = p.next
4    while (c.key < k)
5      p = c
6      c = c.next
7    lock(p)
8    lock(c)
9    if (validate (p,c))
10     if (c.key=k)
11       return false
12     else
13       n = new Node(k,true,-)
14       n.next = c
15       p.next = n
16       return true
17     unlock p
18     unlock c
19     exit
20   else
21     unlock p
22     unlock c
```

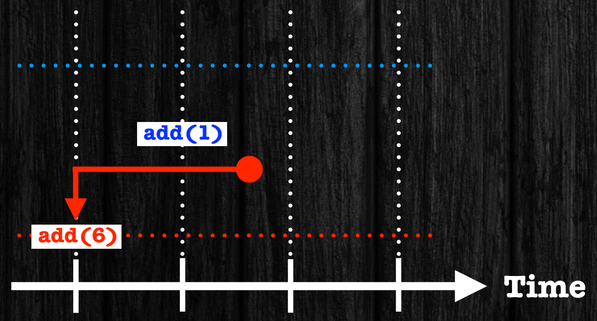
validation





Sets | Optimistic Synchronization | Linearization Policy

```
add(k):
Node p, c
1  while (true)
2    p = H
3    c = p.next
4    while (c.key < k)
5      p = c
6      c = c.next
7    lock(p)
8    lock(c)
9    if (validate (p,c))
10     if (c.key=k)
11       return false
12     else
13       n = new Node(k,true,-)
14       n.next = c
15       p.next = n
16       return true
17     unlock p
18     unlock c
19     exit
20   else
21     unlock p
22     unlock c
```



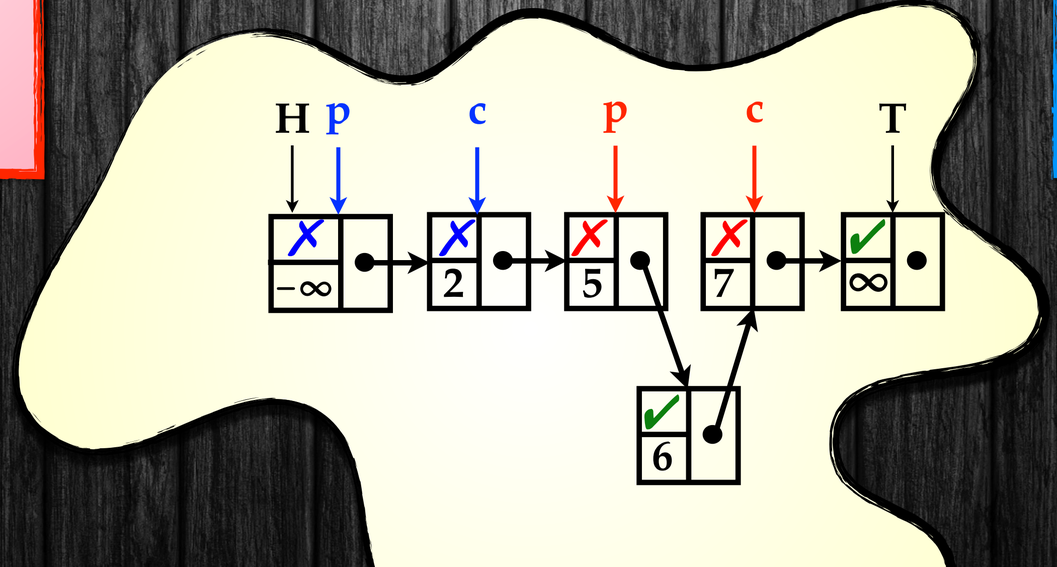
add(6)

add(1)

check wether the two cells are still in the list

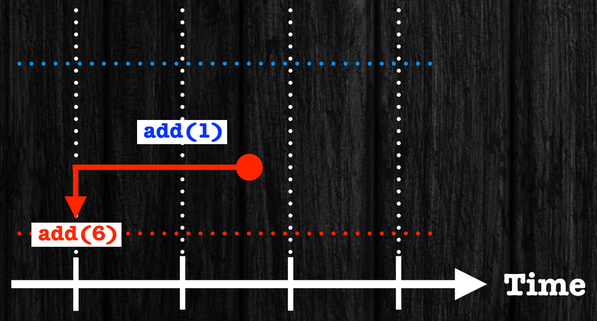
validation

```
add(k):
Node p, c
1  while (true)
2    p = H
3    c = p.next
4    while (c.key < k)
5      p = c
6      c = c.next
7    lock(p)
8    lock(c)
9    if (validate (p,c))
10     if (c.key=k)
11       return false
12     else
13       n = new Node(k,true,-)
14       n.next = c
15       p.next = n
16       return true
17     unlock p
18     unlock c
19     exit
20   else
21     unlock p
22     unlock c
```



Sets | Optimistic Synchronization | Linearization Policy

```
add(k):
Node p, c
1  while (true)
2    p = H
3    c = p.next
4    while (c.key < k)
5      p = c
6      c = c.next
7    lock(p)
8    lock(c)
9    if (validate (p,c))
10     if (c.key=k)
11       return false
12     else
13       n = new Node(k,true,-)
14       n.next = c
15       p.next = n
16       return true
17     unlock p
18     unlock c
19     exit
20   else
21     unlock p
22     unlock c
```



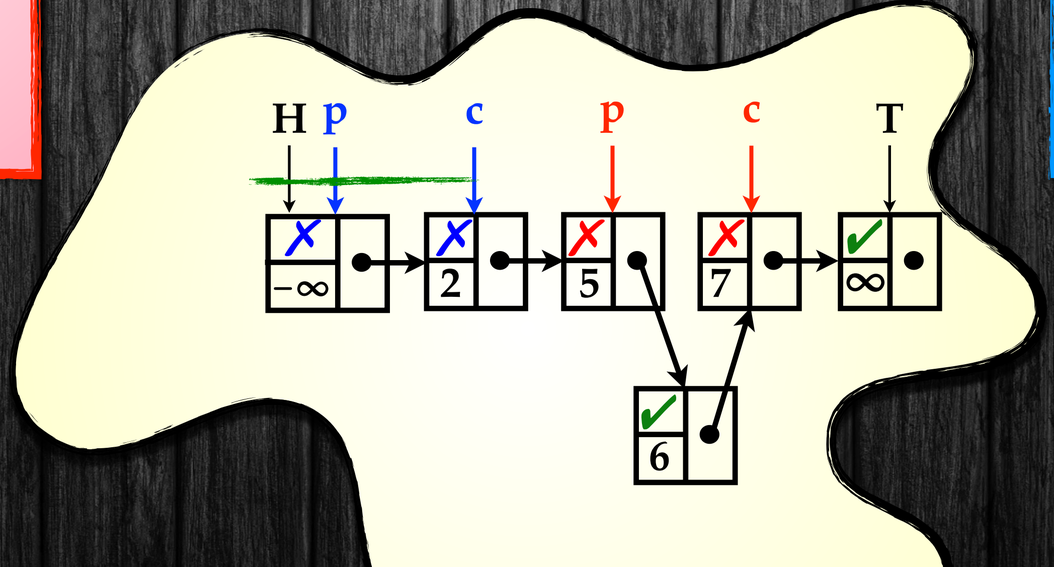
add(6)

add(1)

check wether the two cells are still in the list

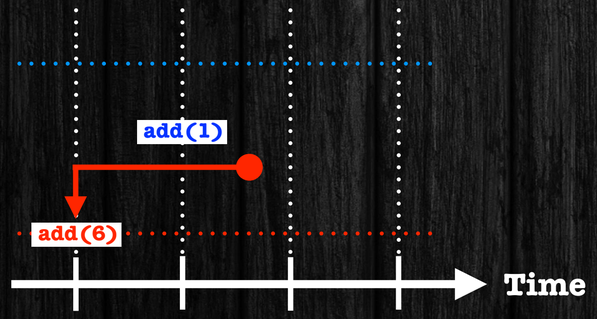
validation

```
add(k):
Node p, c
1  while (true)
2    p = H
3    c = p.next
4    while (c.key < k)
5      p = c
6      c = c.next
7    lock(p)
8    lock(c)
9    if (validate (p,c))
10     if (c.key=k)
11       return false
12     else
13       n = new Node(k,true,-)
14       n.next = c
15       p.next = n
16       return true
17     unlock p
18     unlock c
19     exit
20   else
21     unlock p
22     unlock c
```



Sets | Optimistic Synchronization | Linearization Policy

```
add(k):
Node p, c
1  while (true)
2    p = H
3    c = p.next
4    while (c.key < k)
5      p = c
6      c = c.next
7    lock(p)
8    lock(c)
9    if (validate (p,c))
10     if (c.key=k)
11       return false
12     else
13       n = new Node(k,true,-)
14       n.next = c
15       p.next = n
16       return true
17     unlock p
18     unlock c
19     exit
20   else
21     unlock p
22     unlock c
```



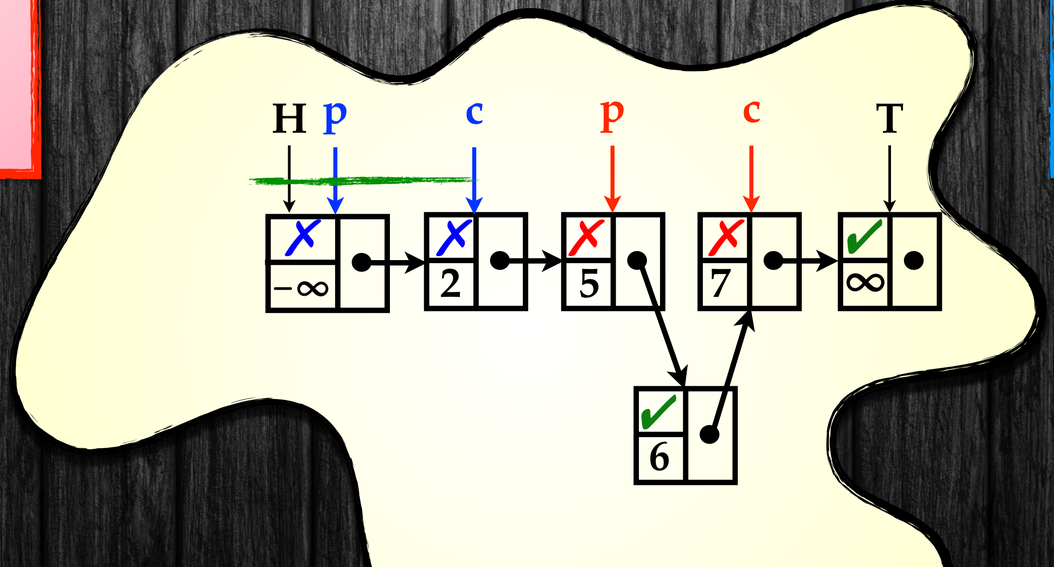
add(6)

add(1)

check wether the two cells are still in the list

yes

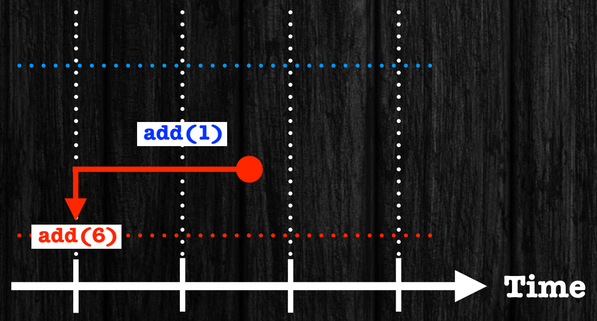
validation




```
add(k):
Node p, c
1  while (true)
2    p = H
3    c = p.next
4    while (c.key < k)
5      p = c
6      c = c.next
7    lock(p)
8    lock(c)
9    if (validate (p,c))
10     if (c.key=k)
11       return false
12     else
13       n = new Node(k,true,-)
14       n.next = c
15       p.next = n
16       return true
17     unlock p
18     unlock c
19     exit
20   else
21     unlock p
22     unlock c
```

Sets | Optimistic Synchronization | Linearization Policy

```
add(k):
Node p, c
1  while (true)
2    p = H
3    c = p.next
4    while (c.key < k)
5      p = c
6      c = c.next
7    lock(p)
8    lock(c)
9    if (validate (p,c))
10     if (c.key=k)
11       return false
12     else
13       n = new Node(k,true,-)
14       n.next = c
15       p.next = n
16       return true
17     unlock p
18     unlock c
19     exit
20   else
21     unlock p
22     unlock c
```



add(6)

add(1)

check wether the two cells are still in the list

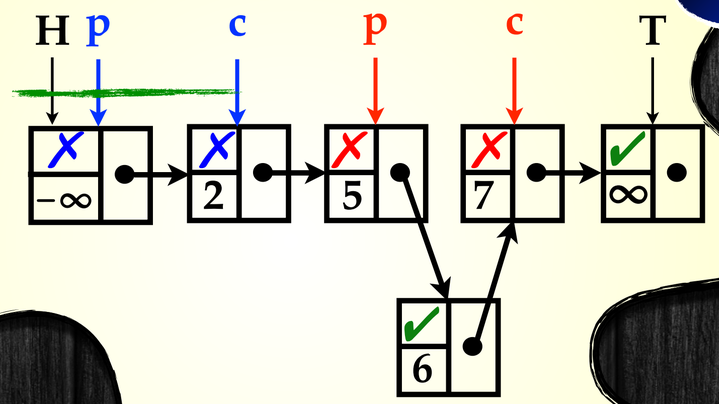
yes

validation

```
add(k):
Node p, c
1  while (true)
2    p = H
3    c = p.next
4    while (c.key < k)
5      p = c
6      c = c.next
7    lock(p)
8    lock(c)
9    if (validate (p,c))
10     if (c.key=k)
11       return false
12     else
13       n = new Node(k,true,-)
14       n.next = c
15       p.next = n
16       return true
17     unlock p
```

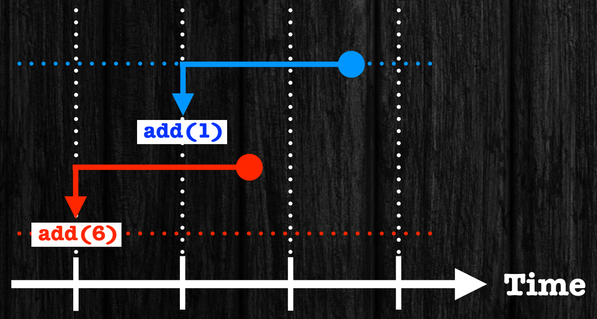
LP

“time point at which validation succeeds”



Sets | Optimistic Synchronization | Linearization Policy

```
add(k):
Node p, c
1  while (true)
2    p = H
3    c = p.next
4    while (c.key < k)
5      p = c
6      c = c.next
7    lock(p)
8    lock(c)
9    if (validate (p,c))
10     if (c.key=k)
11       return false
12     else
13       n = new Node(k,true,-)
14       n.next = c
15       p.next = n
16       return true
17     unlock p
18     unlock c
19     exit
20   else
21     unlock p
22     unlock c
```



add(6)

add(1)

check wether the two cells are still in the list

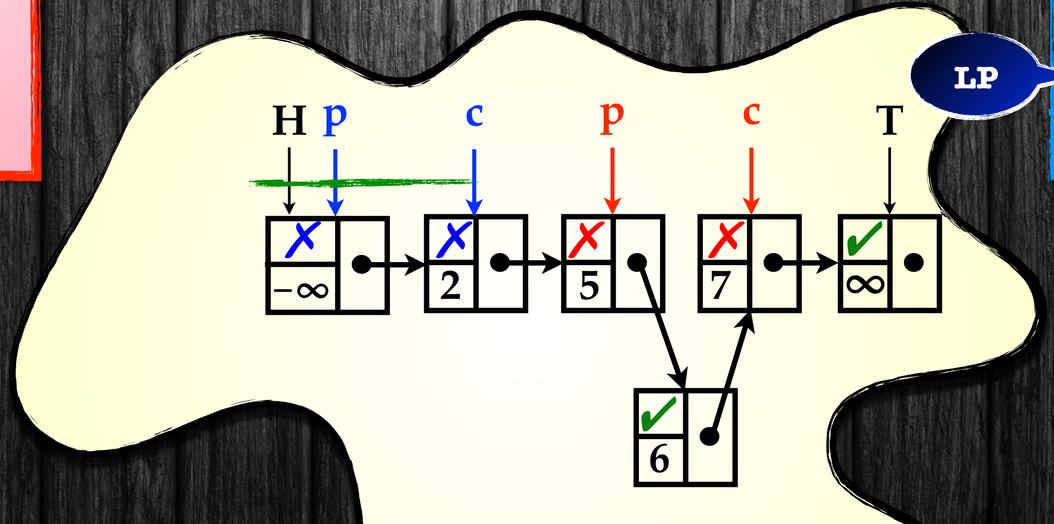
yes

validation

```
add(k):
Node p, c
1  while (true)
2    p = H
3    c = p.next
4    while (c.key < k)
5      p = c
6      c = c.next
7    lock(p)
8    lock(c)
9    if (validate (p,c))
10     if (c.key=k)
11       return false
12     else
13       n = new Node(k,true,-)
14       n.next = c
15       p.next = n
16       return true
17     unlock p
```

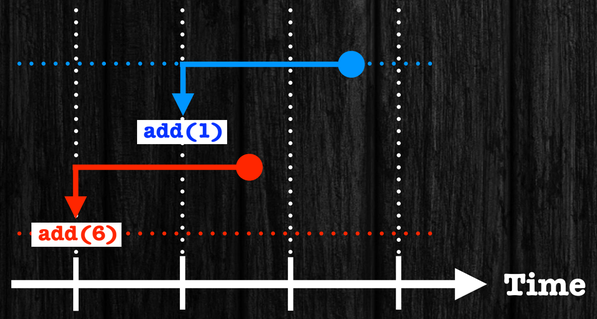
LP

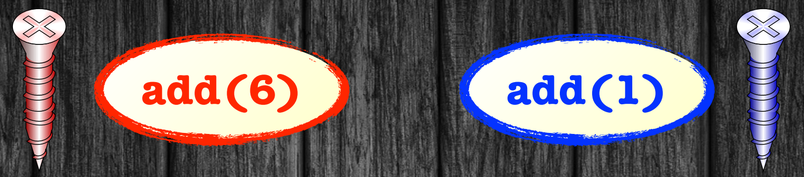
“time point at which validation succeeds”

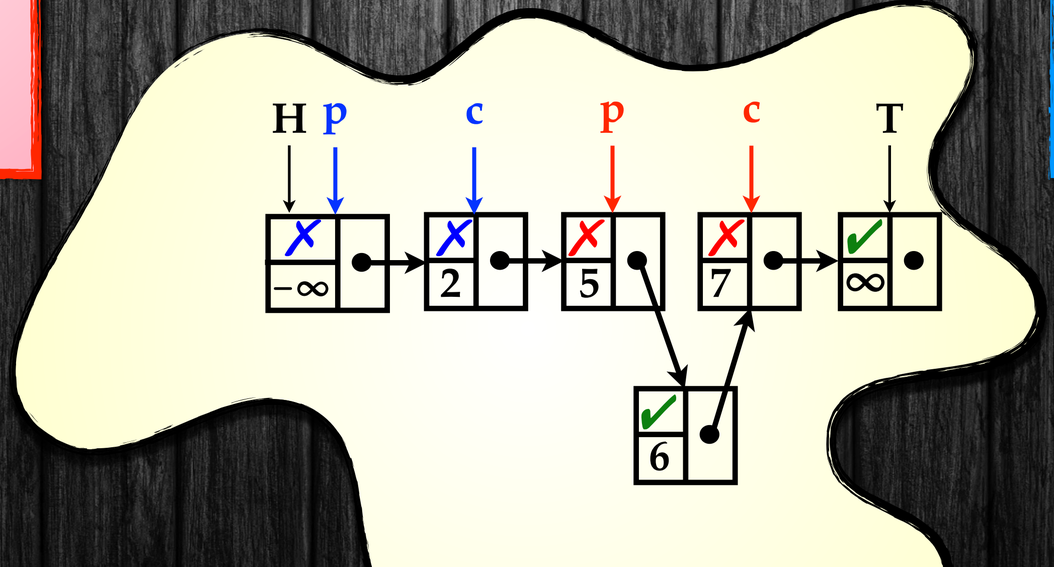


Sets | Optimistic Synchronization | Linearization Policy

```
add(k):
Node p, c
1  while (true)
2    p = H
3    c = p.next
4    while (c.key < k)
5      p = c
6      c = c.next
7    lock(p)
8    lock(c)
9    if (validate (p,c))
10     if (c.key=k)
11       return false
12     else
13       n = new Node(k,true,-)
14       n.next = c
15       p.next = n
16       return true
17     unlock p
18     unlock c
19     exit
20   else
21     unlock p
22     unlock c
```



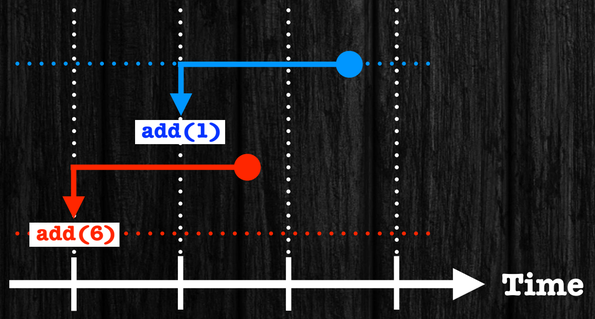


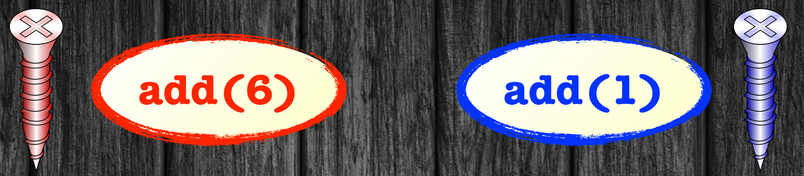


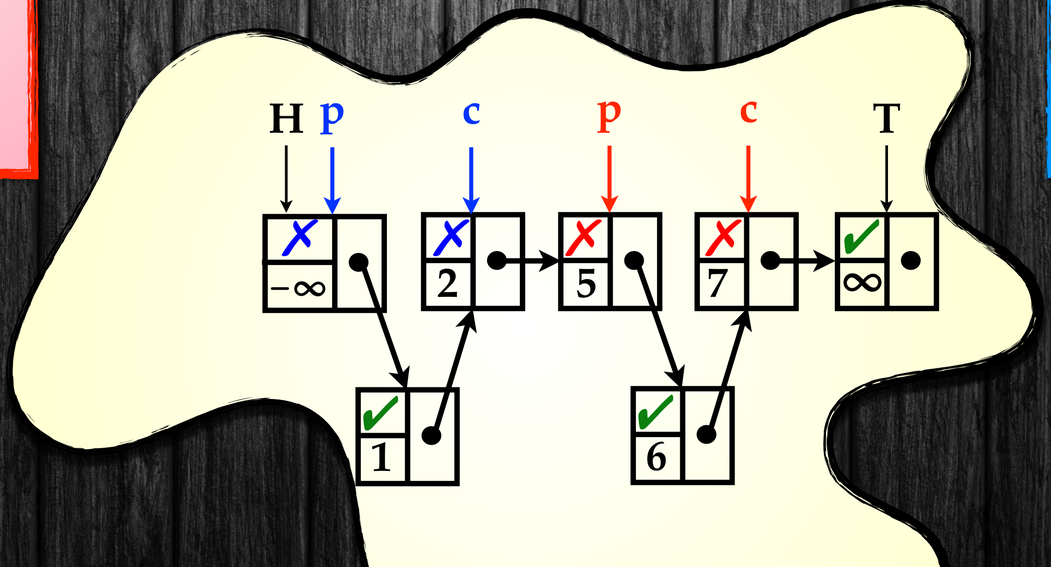

```
add(k):
Node p, c
1  while (true)
2    p = H
3    c = p.next
4    while (c.key < k)
5      p = c
6      c = c.next
7    lock(p)
8    lock(c)
9    if (validate (p,c))
10     if (c.key=k)
11       return false
12     else
13       n = new Node(k,true,-)
14       n.next = c
15       p.next = n
16       return true
17     unlock p
18     unlock c
19     exit
20   else
21     unlock p
22     unlock c
```

Sets | Optimistic Synchronization | Linearization Policy

```
add(k):
Node p, c
1  while (true)
2    p = H
3    c = p.next
4    while (c.key < k)
5      p = c
6      c = c.next
7    lock(p)
8    lock(c)
9    if (validate (p,c))
10     if (c.key=k)
11       return false
12     else
13       n = new Node(k,true,-)
14       n.next = c
15       p.next = n
16       return true
17     unlock p
18     unlock c
19     exit
20   else
21     unlock p
22     unlock c
```



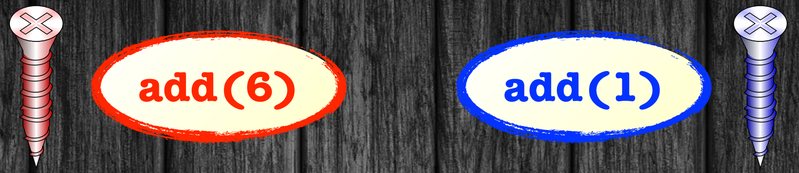


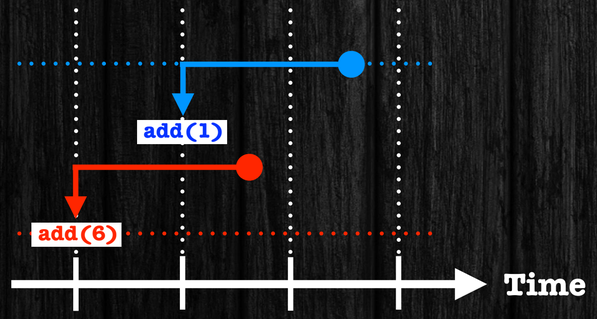


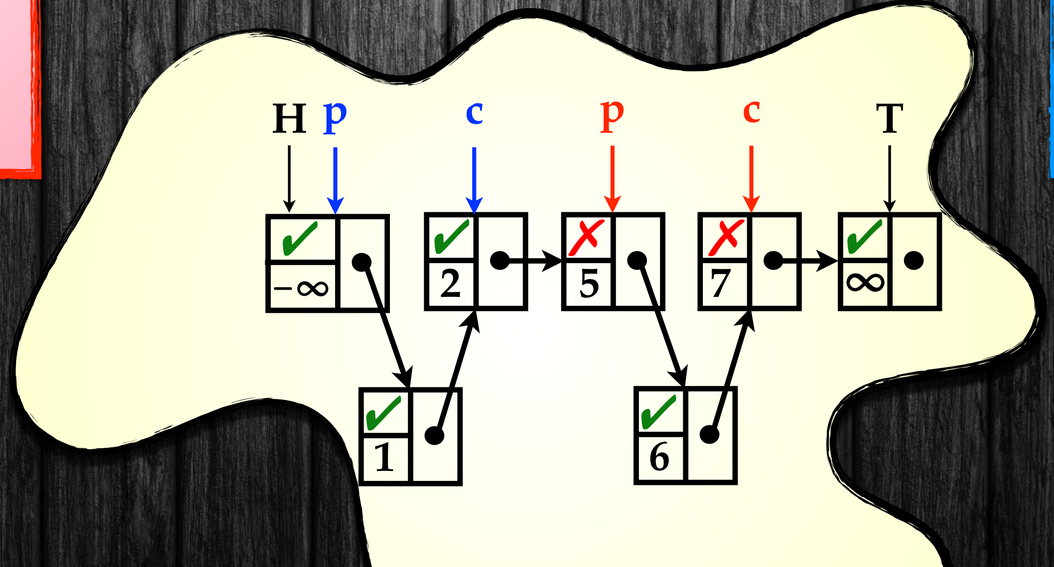

```
add(k):
Node p, c
1  while (true)
2    p = H
3    c = p.next
4    while (c.key < k)
5      p = c
6      c = c.next
7    lock(p)
8    lock(c)
9    if (validate (p,c))
10     if (c.key=k)
11       return false
12     else
13       n = new Node(k,true,-)
14       n.next = c
15       p.next = n
16       return true
17     unlock p
18     unlock c
19     exit
20   else
21     unlock p
22     unlock c
```

Sets | Optimistic Synchronization | Linearization Policy

```
add(k):
Node p, c
1  while (true)
2    p = H
3    c = p.next
4    while (c.key < k)
5      p = c
6      c = c.next
7    lock(p)
8    lock(c)
9    if (validate (p,c))
10     if (c.key=k)
11       return false
12     else
13       n = new Node(k,true,-)
14       n.next = c
15       p.next = n
16       return true
17     unlock p
18     unlock c
19     exit
20   else
21     unlock p
22     unlock c
```



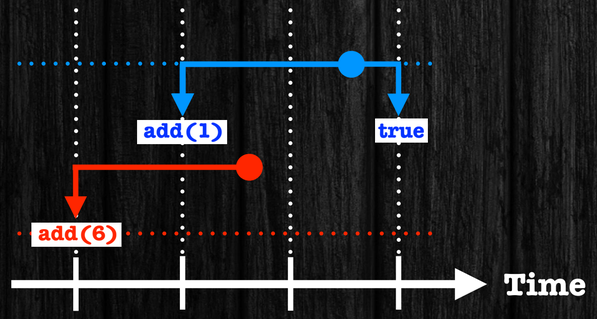


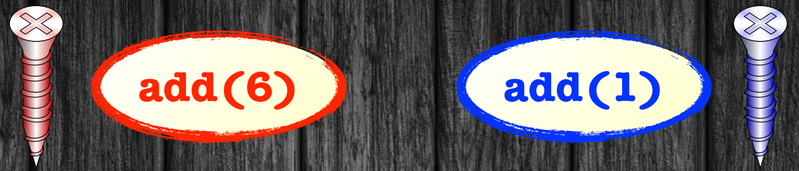


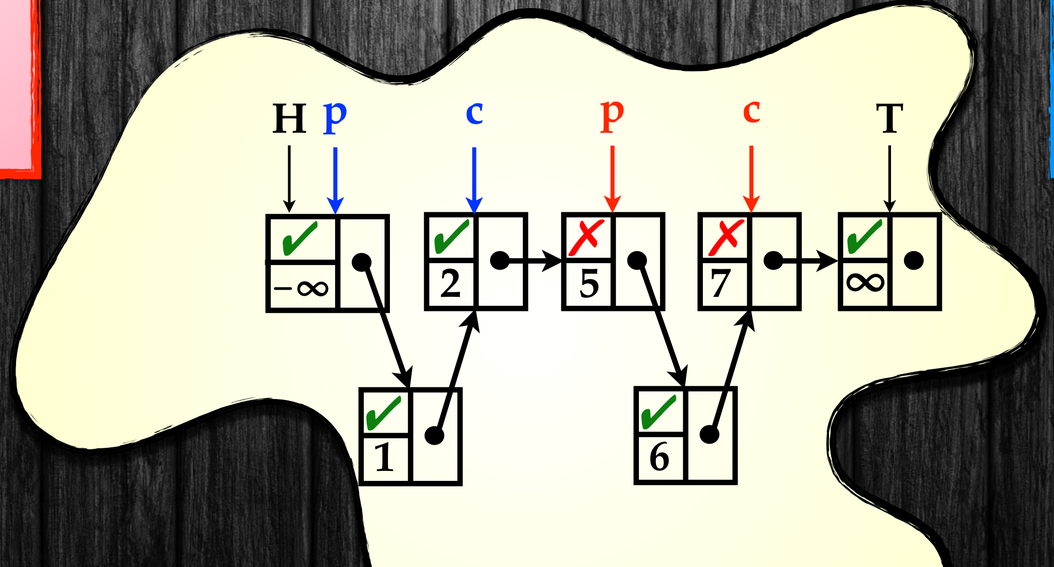

```
add(k):
Node p, c
1  while (true)
2    p = H
3    c = p.next
4    while (c.key < k)
5      p = c
6      c = c.next
7    lock(p)
8    lock(c)
9    if (validate (p,c))
10     if (c.key=k)
11       return false
12     else
13       n = new Node(k,true,-)
14       n.next = c
15       p.next = n
16       return true
17     unlock p
18     unlock c
19     exit
20   else
21     unlock p
22     unlock c
```

Sets | Optimistic Synchronization | Linearization Policy

```
add(k):
Node p, c
1  while (true)
2    p = H
3    c = p.next
4    while (c.key < k)
5      p = c
6      c = c.next
7    lock(p)
8    lock(c)
9    if (validate (p,c))
10     if (c.key=k)
11       return false
12     else
13       n = new Node(k,true,-)
14       n.next = c
15       p.next = n
16       return true
17     unlock p
18     unlock c
19     exit
20   else
21     unlock p
22     unlock c
```



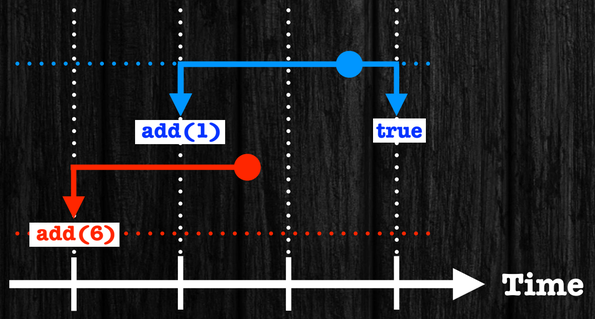


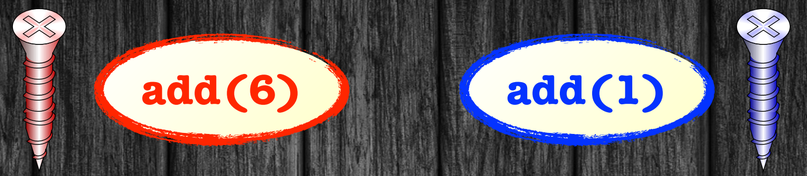


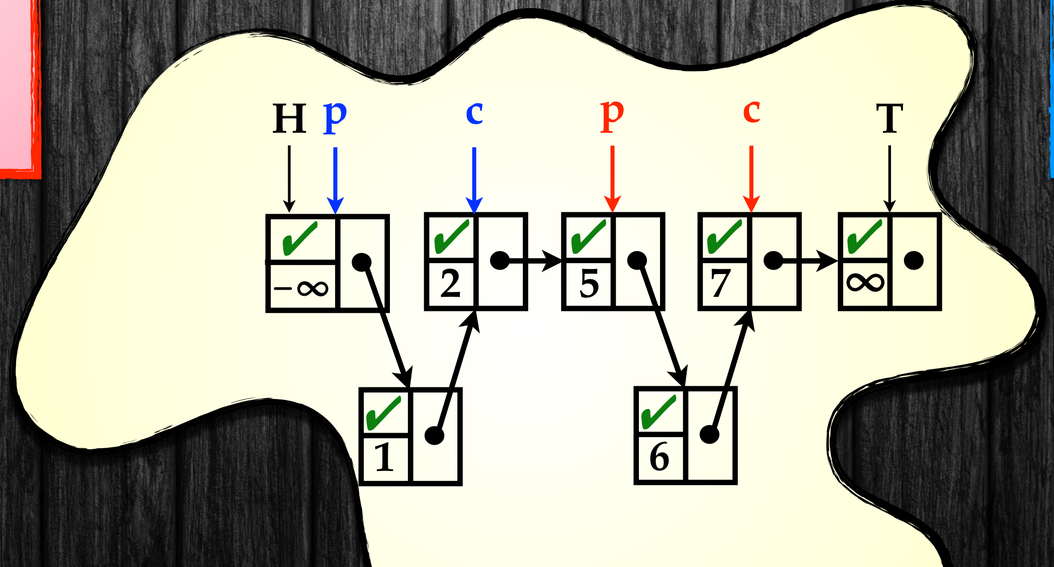

```
add(k):
Node p, c
1  while (true)
2    p = H
3    c = p.next
4    while (c.key < k)
5      p = c
6      c = c.next
7    lock(p)
8    lock(c)
9    if (validate (p,c))
10     if (c.key=k)
11       return false
12     else
13       n = new Node(k,true,-)
14       n.next = c
15       p.next = n
16       return true
17     unlock p
18     unlock c
19     exit
20   else
21     unlock p
22     unlock c
```

Sets | Optimistic Synchronization | Linearization Policy

```
add(k):
Node p, c
1  while (true)
2    p = H
3    c = p.next
4    while (c.key < k)
5      p = c
6      c = c.next
7    lock(p)
8    lock(c)
9    if (validate (p,c))
10     if (c.key=k)
11       return false
12     else
13       n = new Node(k,true,-)
14       n.next = c
15       p.next = n
16       return true
17     unlock p
18     unlock c
19     exit
20   else
21     unlock p
22     unlock c
```



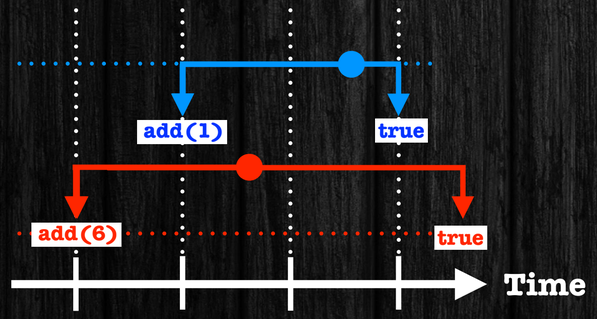


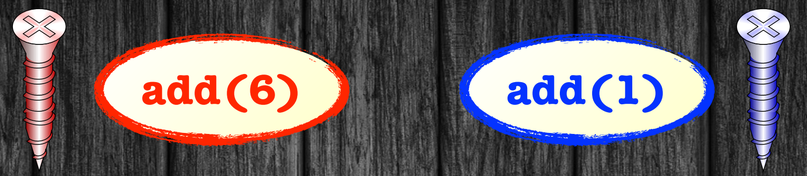


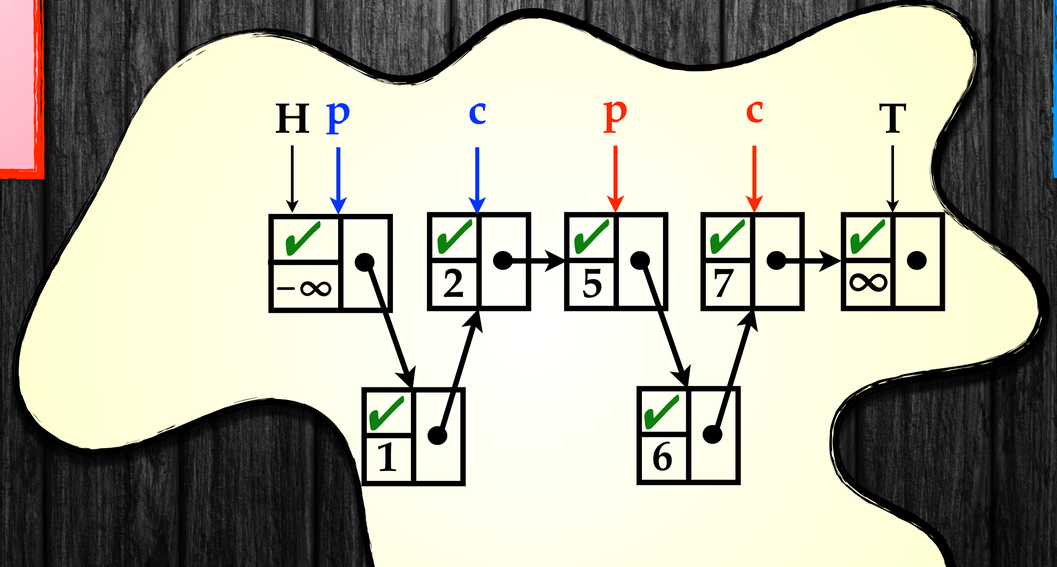

```
add(k):
Node p, c
1  while (true)
2    p = H
3    c = p.next
4    while (c.key < k)
5      p = c
6      c = c.next
7    lock(p)
8    lock(c)
9    if (validate (p,c))
10     if (c.key=k)
11       return false
12     else
13       n = new Node(k,true,-)
14       n.next = c
15       p.next = n
16       return true
17     unlock p
18     unlock c
19     exit
20   else
21     unlock p
22     unlock c
```

Sets | Optimistic Synchronization | Linearization Policy

```
add(k):
Node p, c
1  while (true)
2    p = H
3    c = p.next
4    while (c.key < k)
5      p = c
6      c = c.next
7    lock(p)
8    lock(c)
9    if (validate (p,c))
10     if (c.key=k)
11       return false
12     else
13       n = new Node(k,true,-)
14       n.next = c
15       p.next = n
16       return true
17     unlock p
18     unlock c
19     exit
20   else
21     unlock p
22     unlock c
```





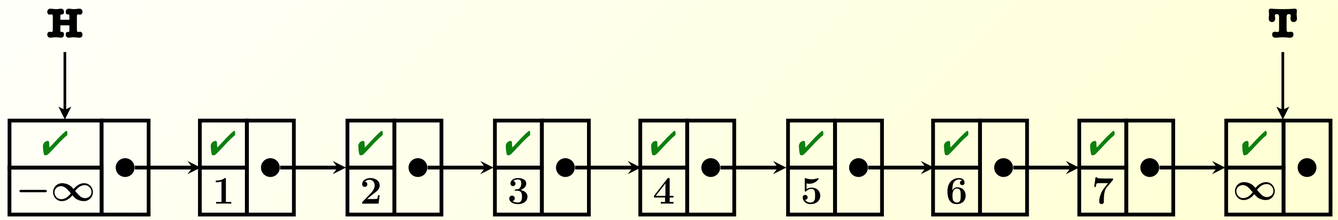



```
add(k):
Node p, c
1  while (true)
2    p = H
3    c = p.next
4    while (c.key < k)
5      p = c
6      c = c.next
7    lock(p)
8    lock(c)
9    if (validate (p,c))
10     if (c.key=k)
11       return false
12     else
13       n = new Node(k,true,-)
14       n.next = c
15       p.next = n
16       return true
17     unlock p
18     unlock c
19     exit
20   else
21     unlock p
22     unlock c
```

Sets

Optimistic Synchronization

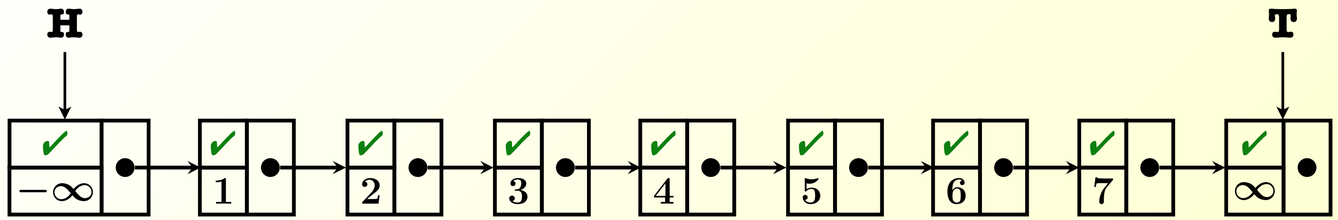
Why Validation?

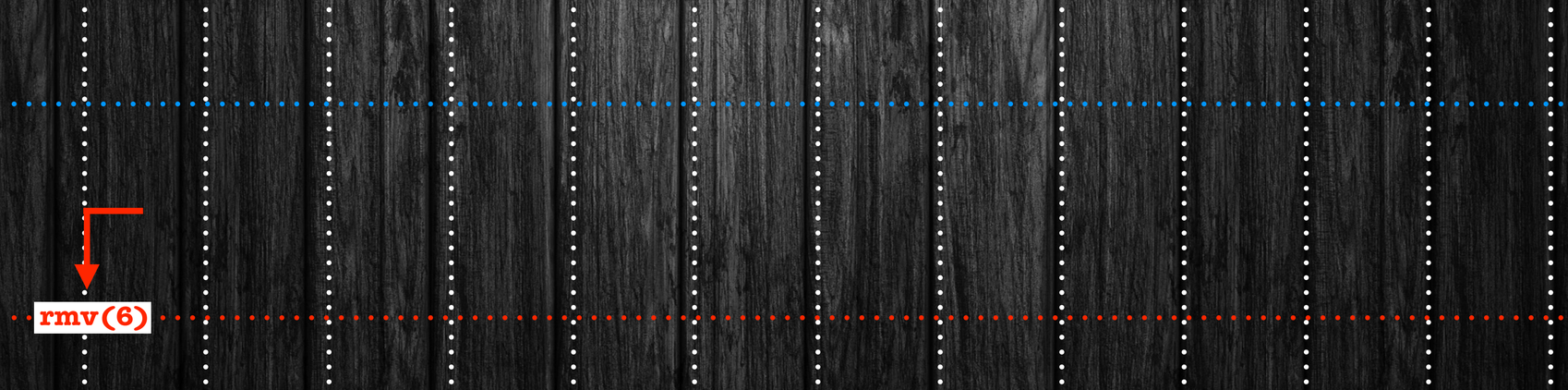


Sets

Optimistic Synchronization

Why Validation?

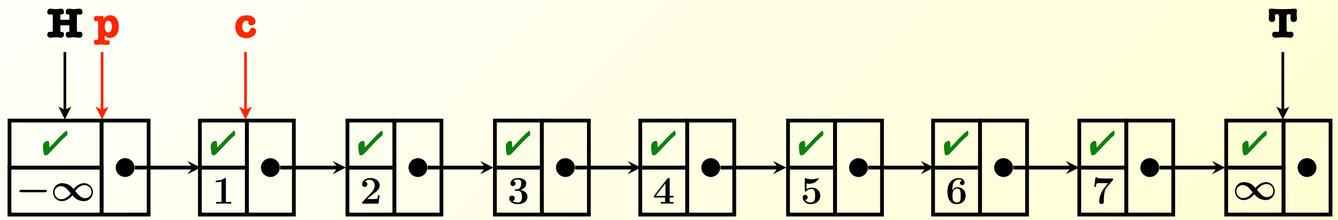


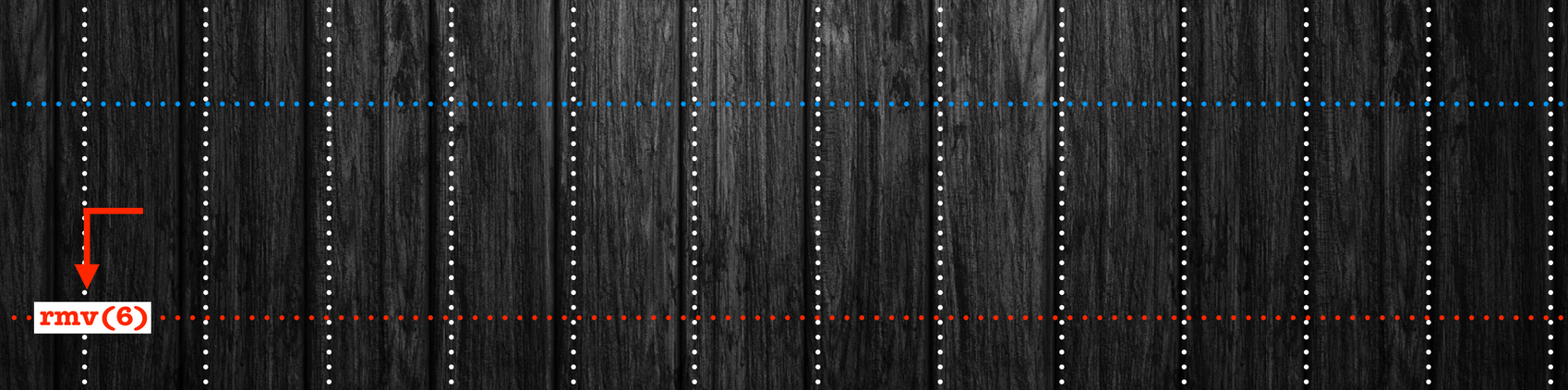


Sets

Optimistic Synchronization

Why Validation?

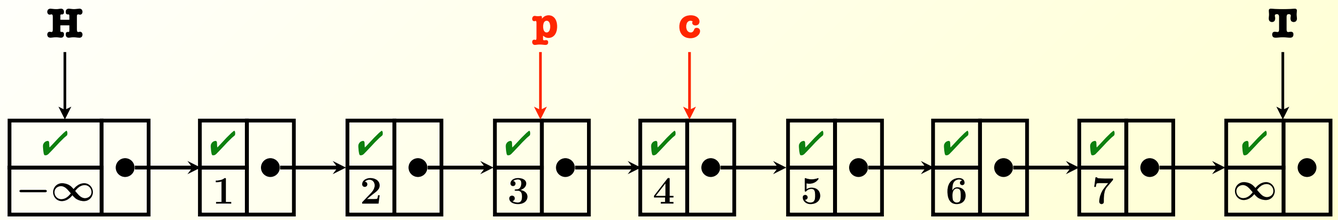


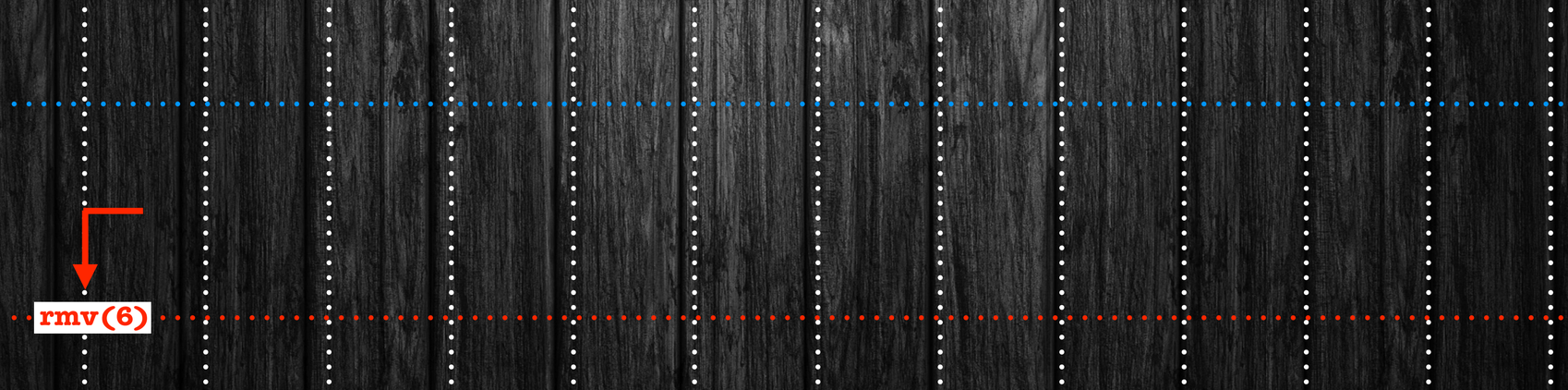


Sets

Optimistic Synchronization

Why Validation?

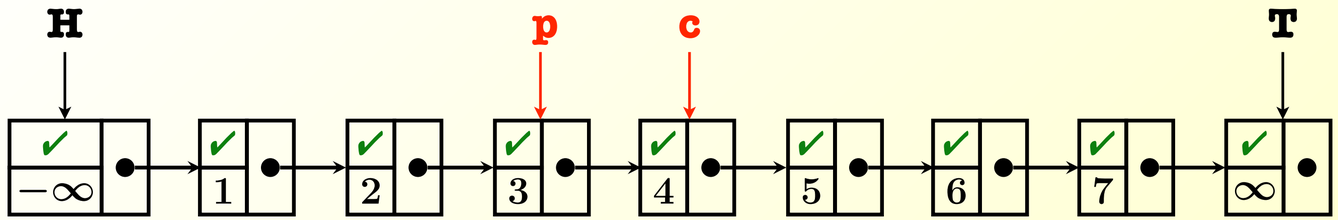


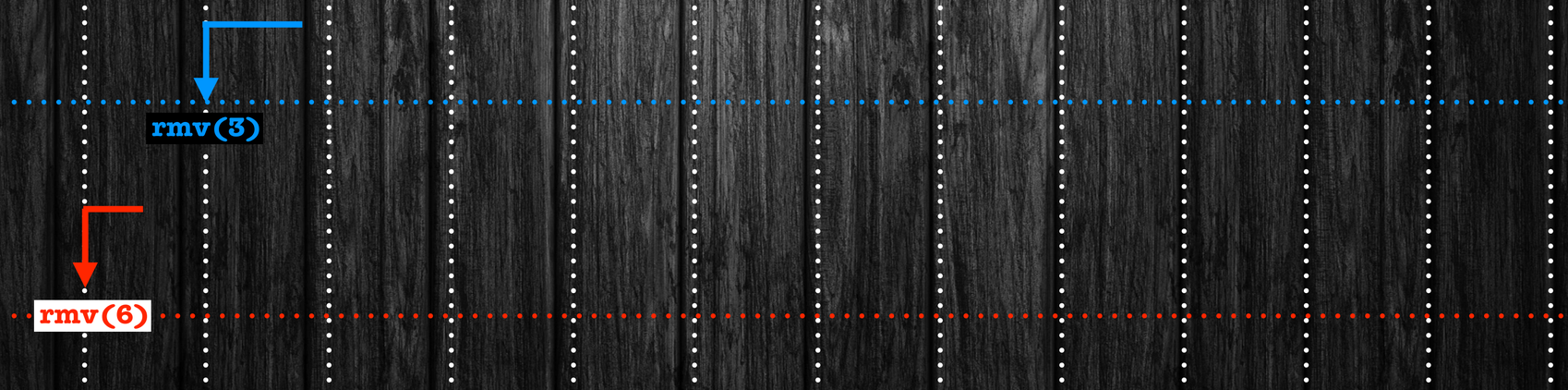


Sets

Optimistic Synchronization

Why Validation?

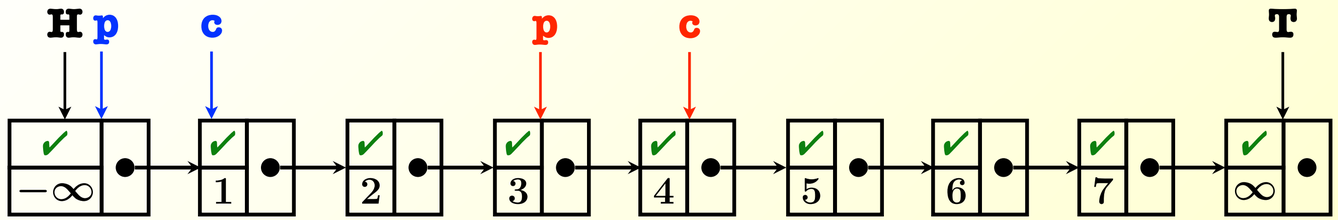


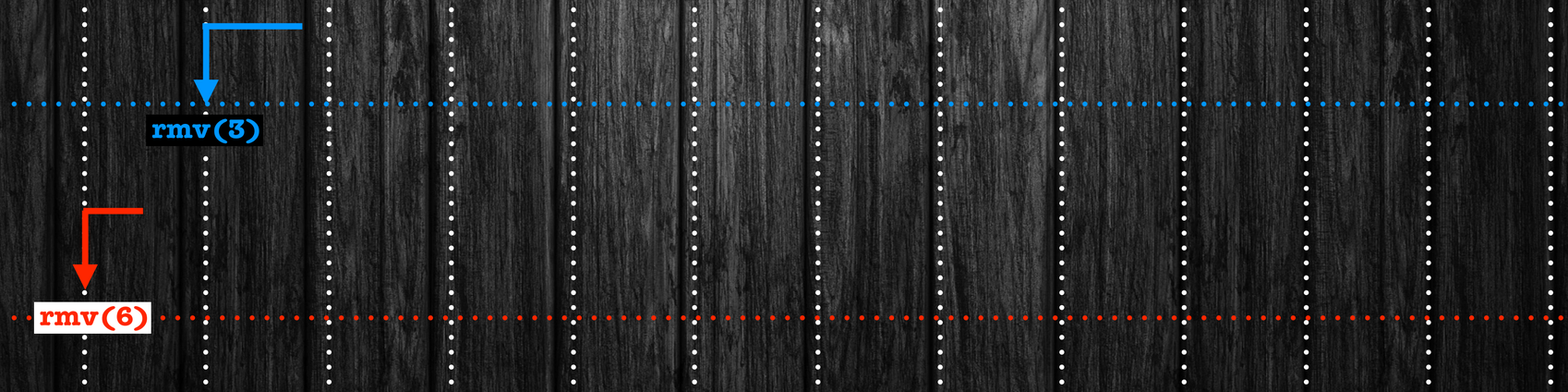


Sets

Optimistic Synchronization

Why Validation?

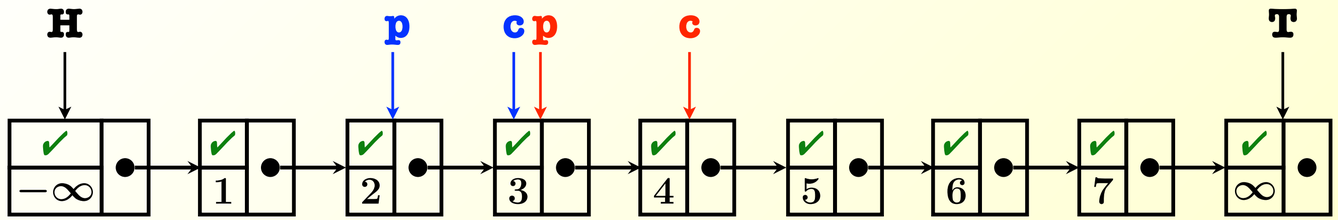


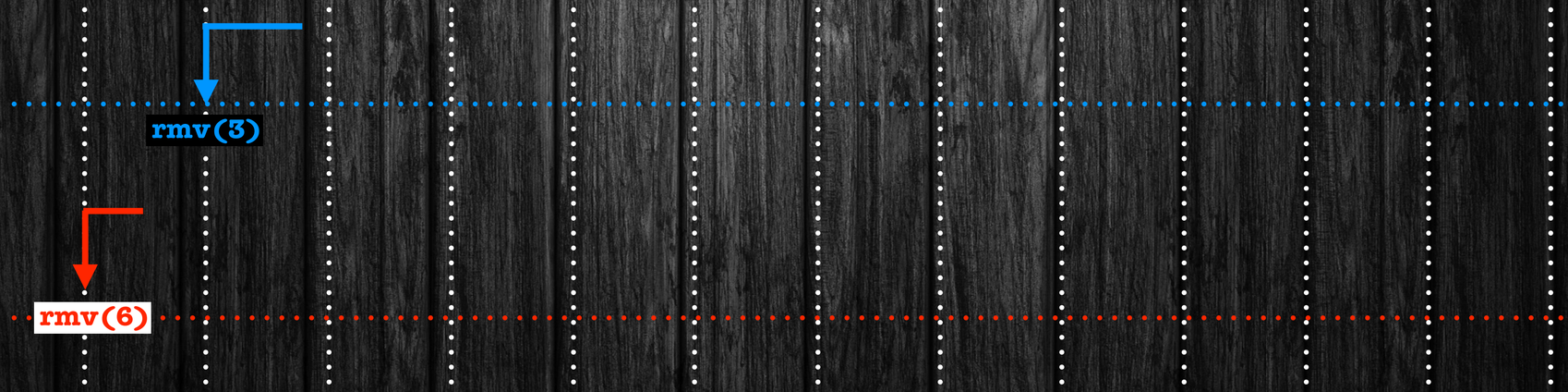


Sets

Optimistic Synchronization

Why Validation?

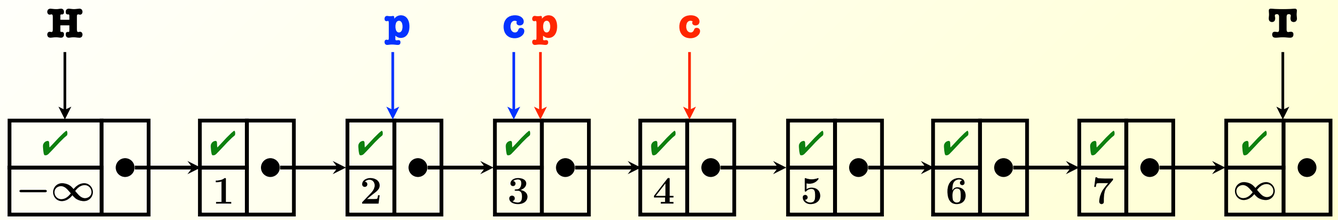


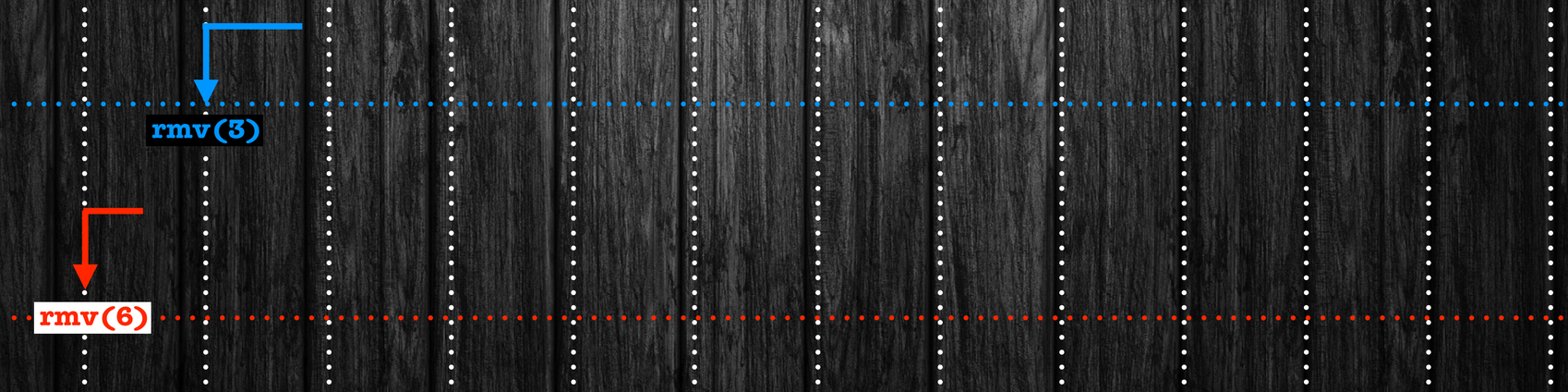


Sets

Optimistic Synchronization

Why Validation?

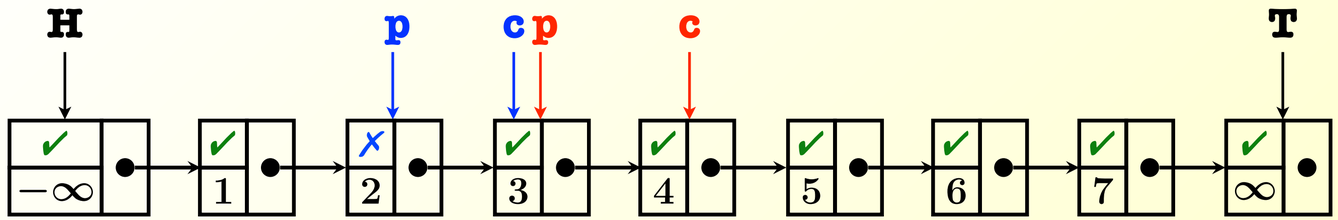


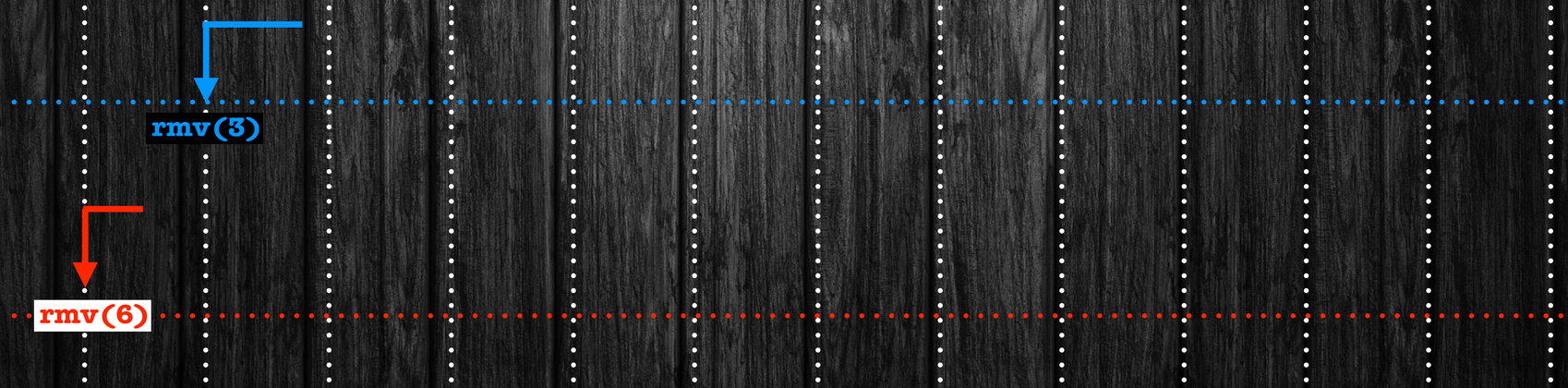


Sets

Optimistic Synchronization

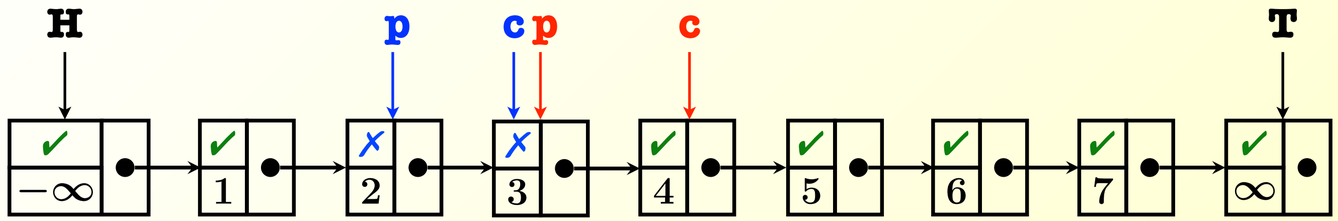
Why Validation?

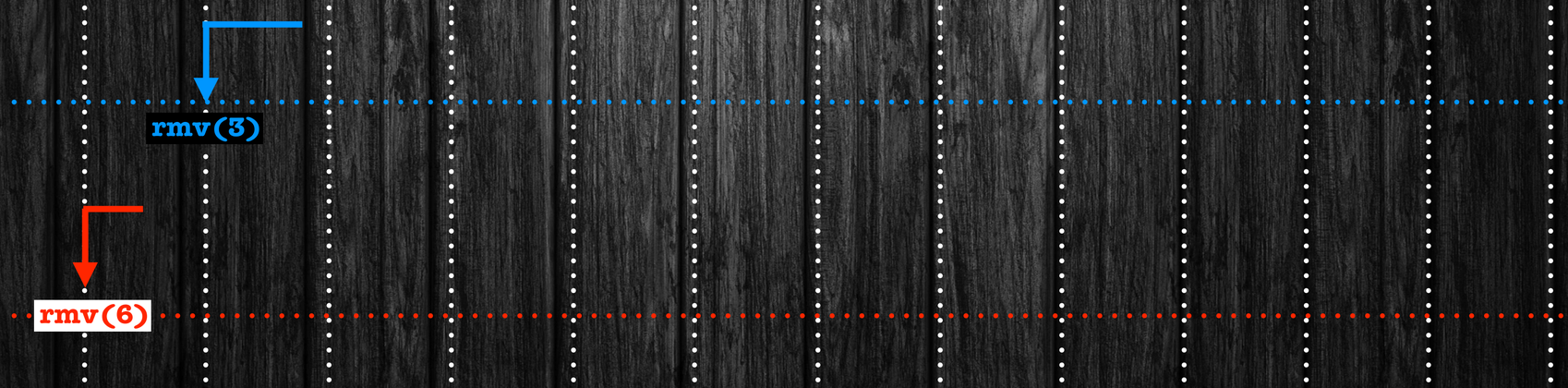




Sets | Optimistic Synchronization

Why Validation?

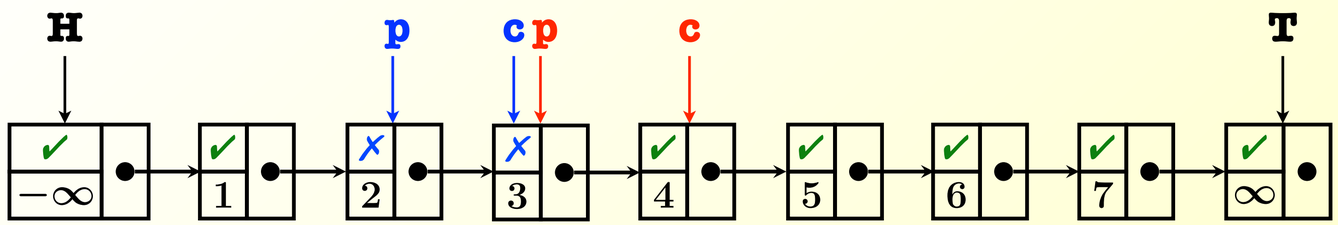


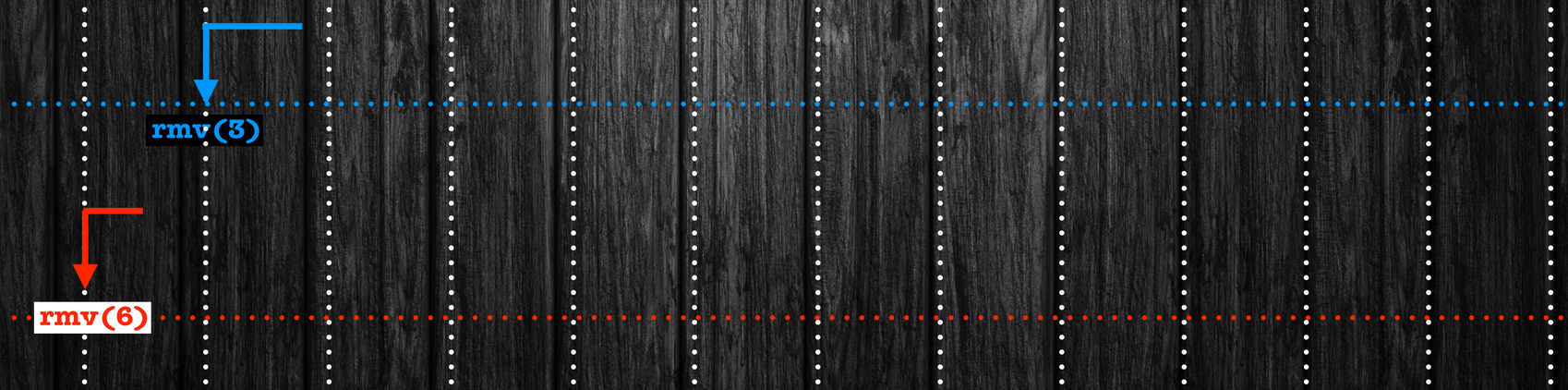


Sets

Optimistic Synchronization

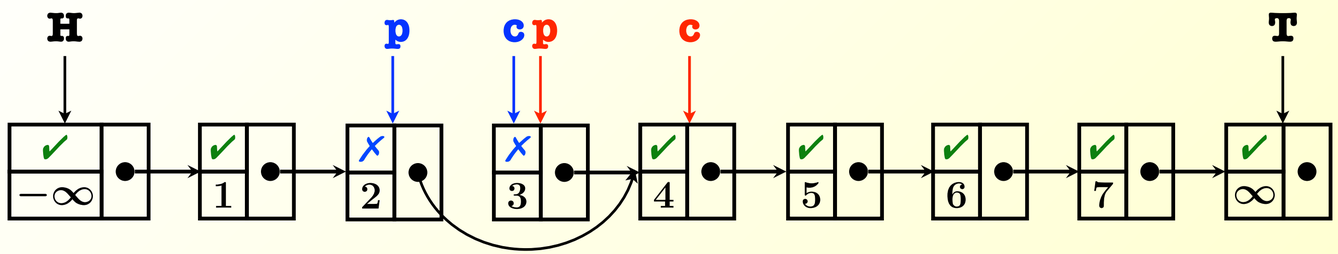
Why Validation?

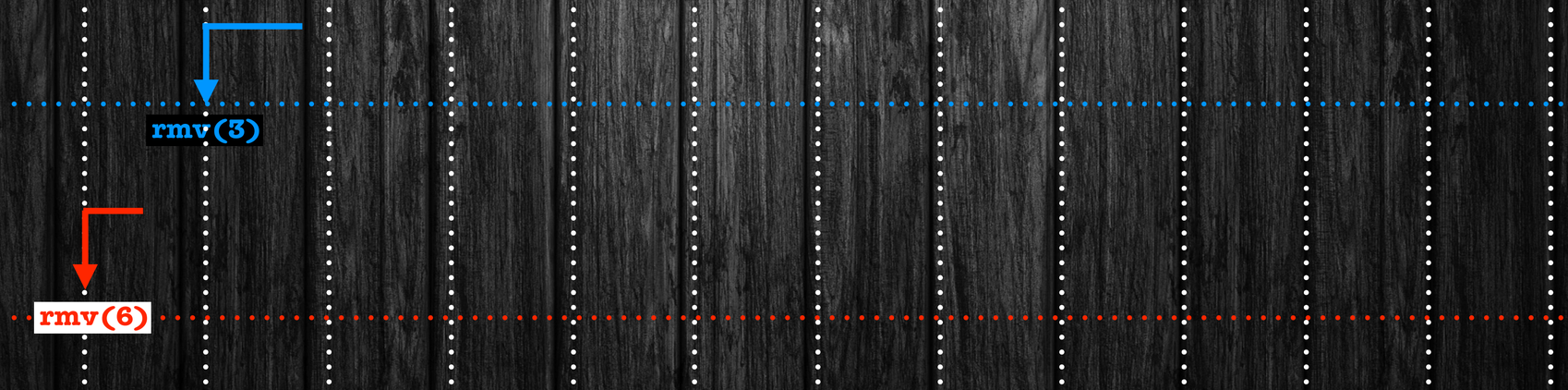




Sets | Optimistic Synchronization

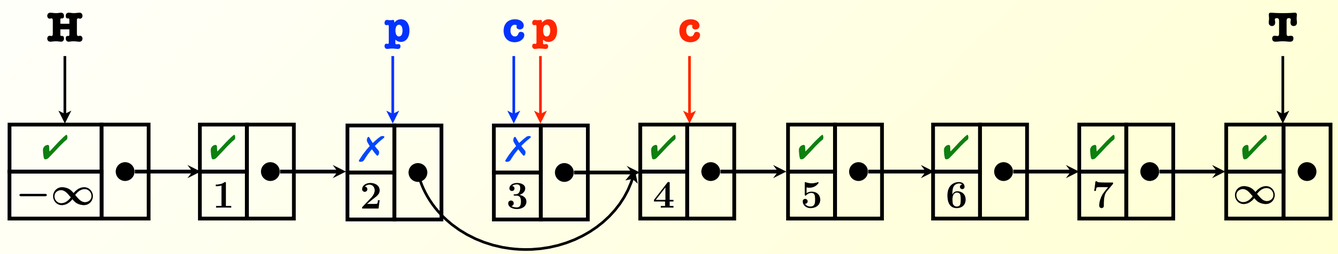
Why Validation?

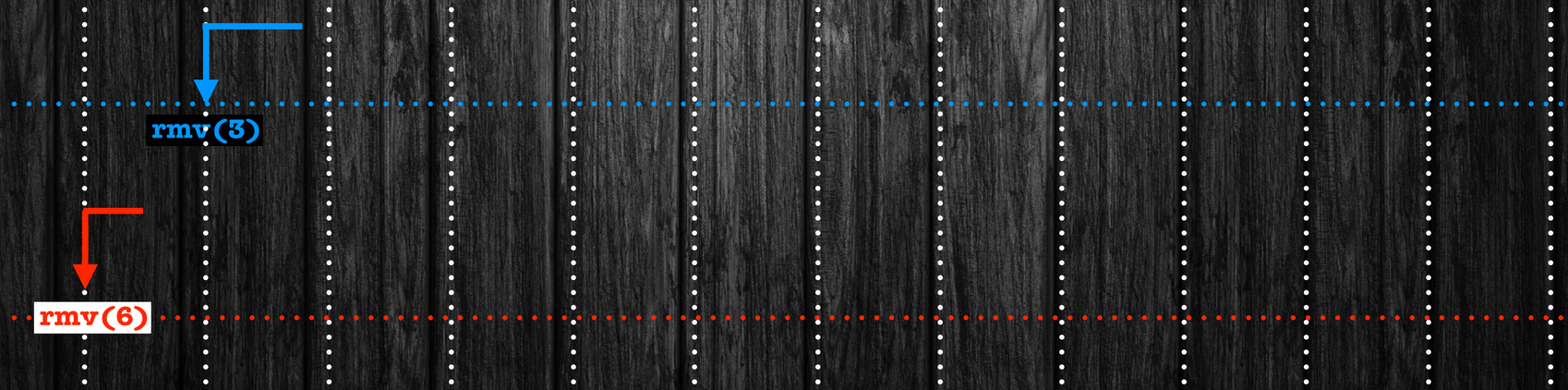




Sets | Optimistic Synchronization

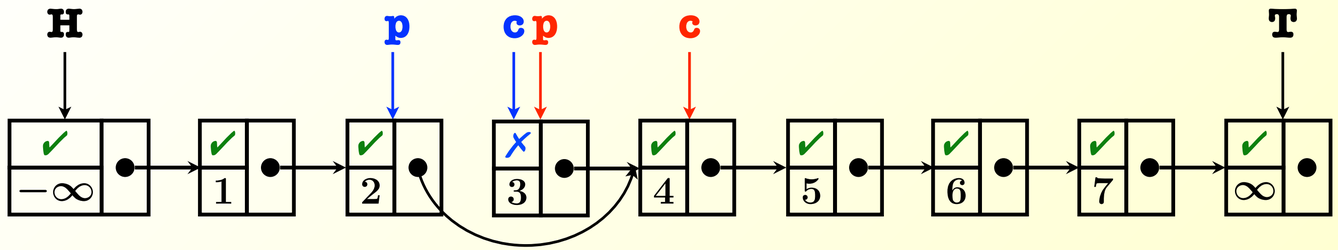
Why Validation?

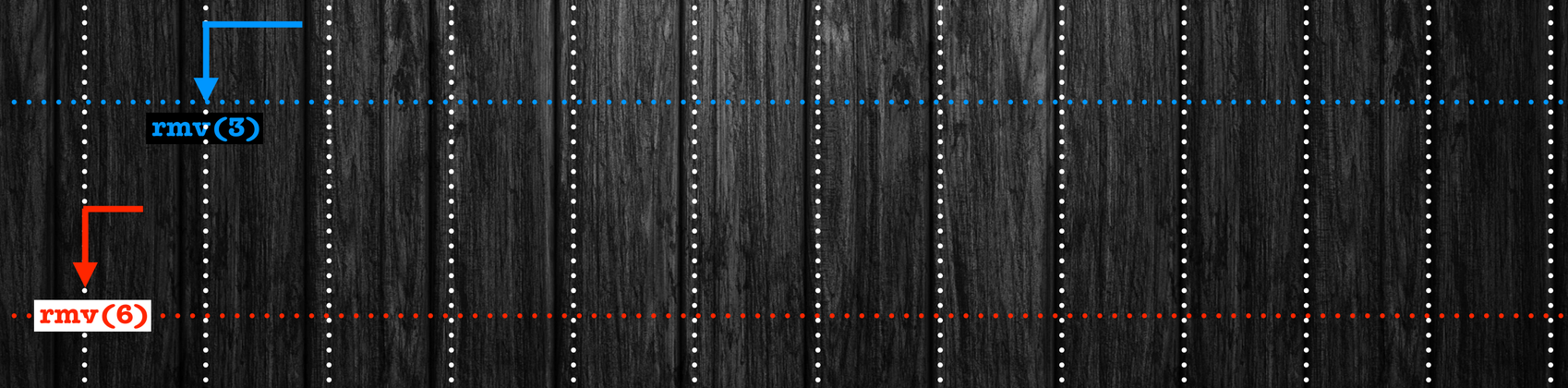




Sets | Optimistic Synchronization

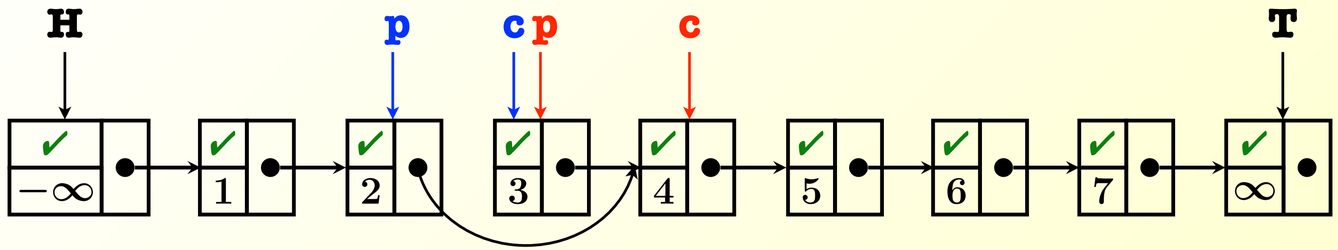
Why Validation?

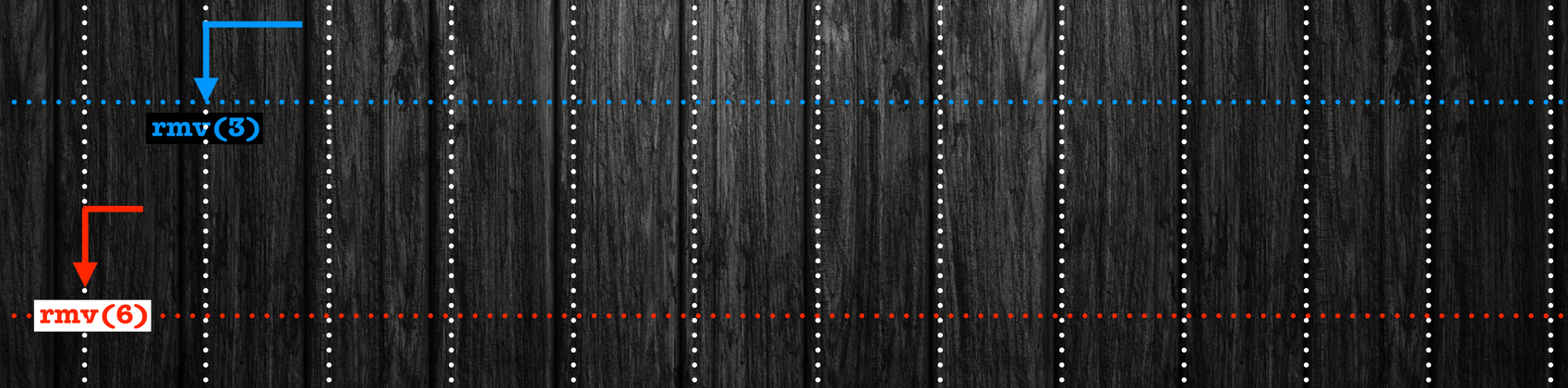




Sets | Optimistic Synchronization

Why Validation?

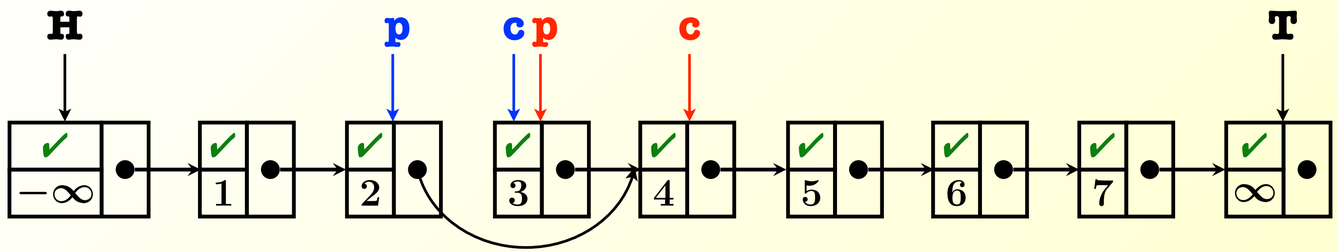


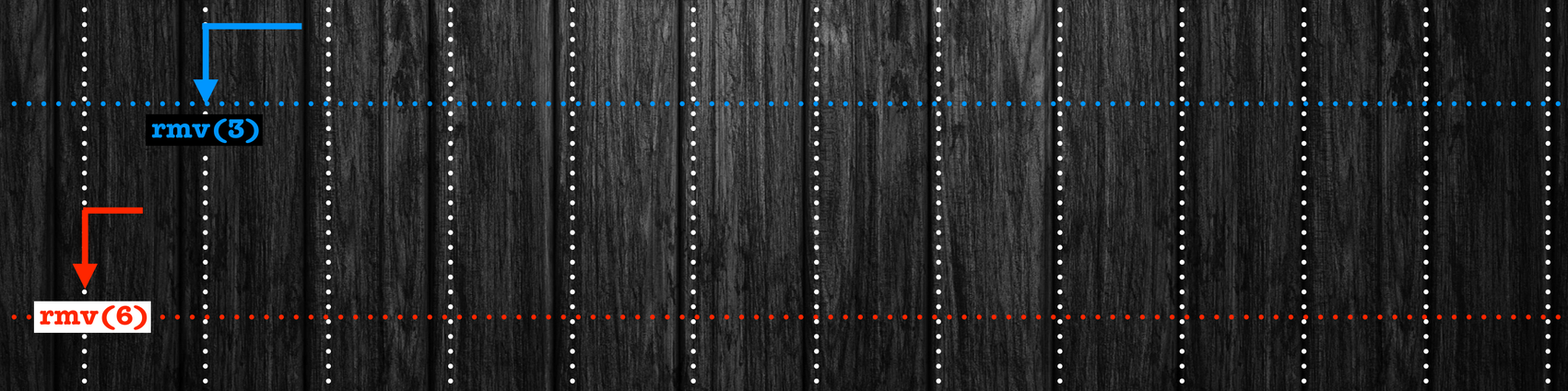


Sets

Optimistic Synchronization

Why Validation?

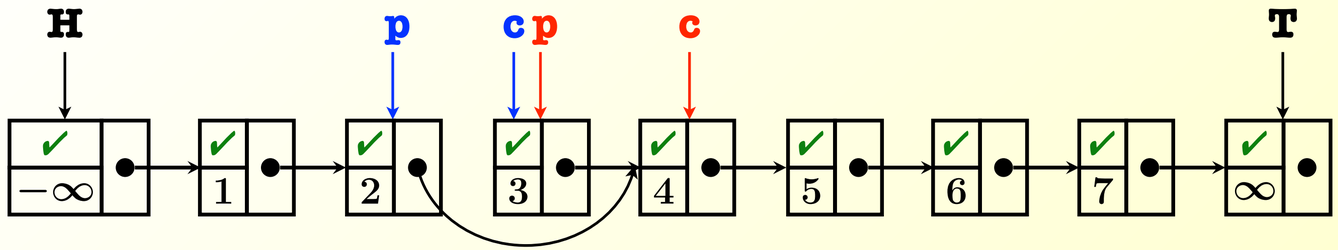


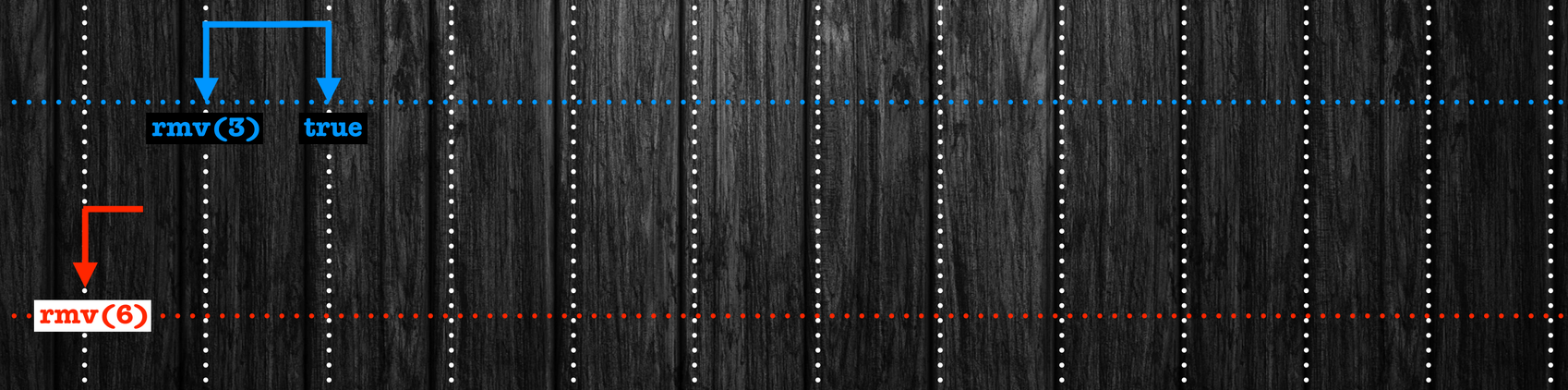


Sets

Optimistic Synchronization

Why Validation?

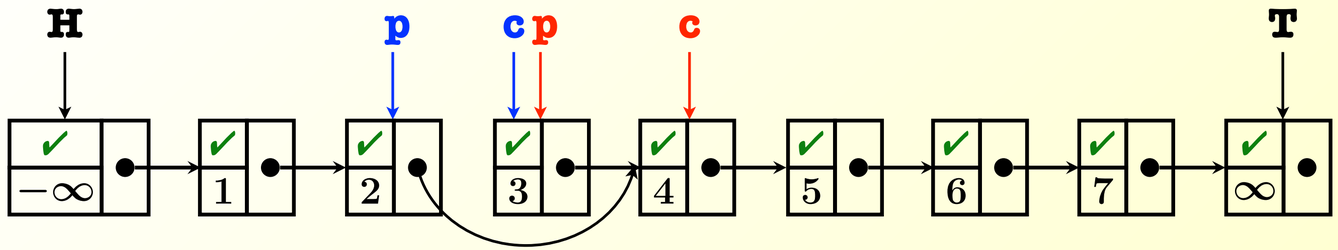


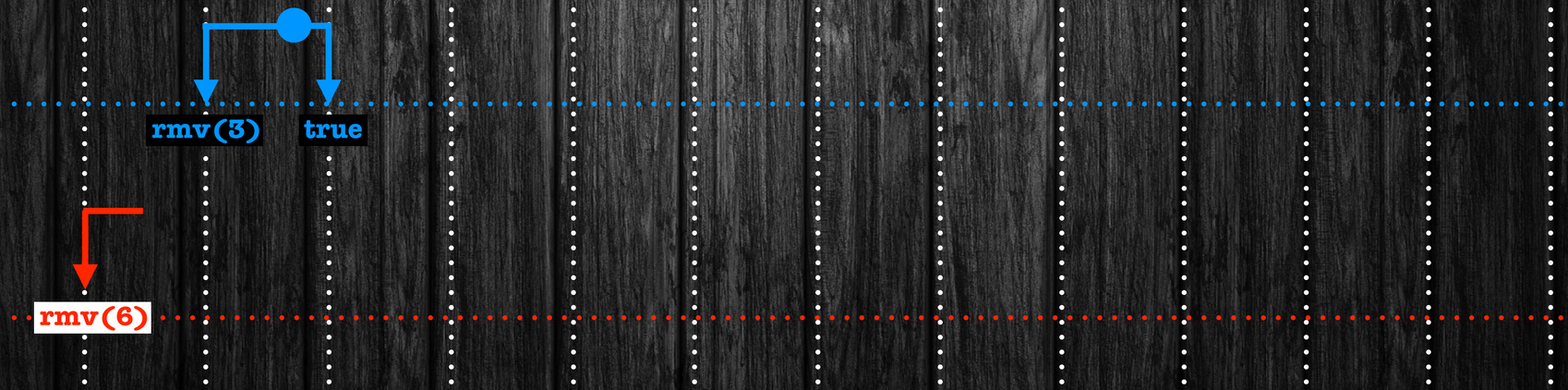


Sets

Optimistic Synchronization

Why Validation?

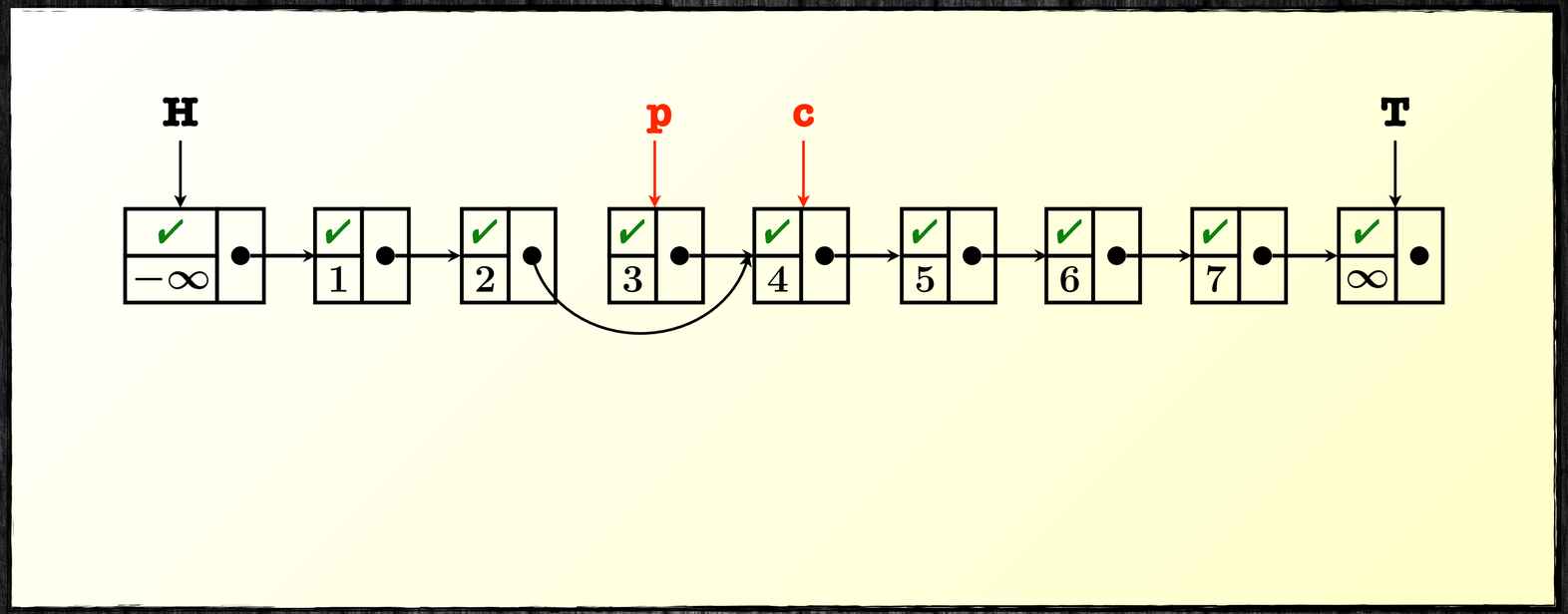


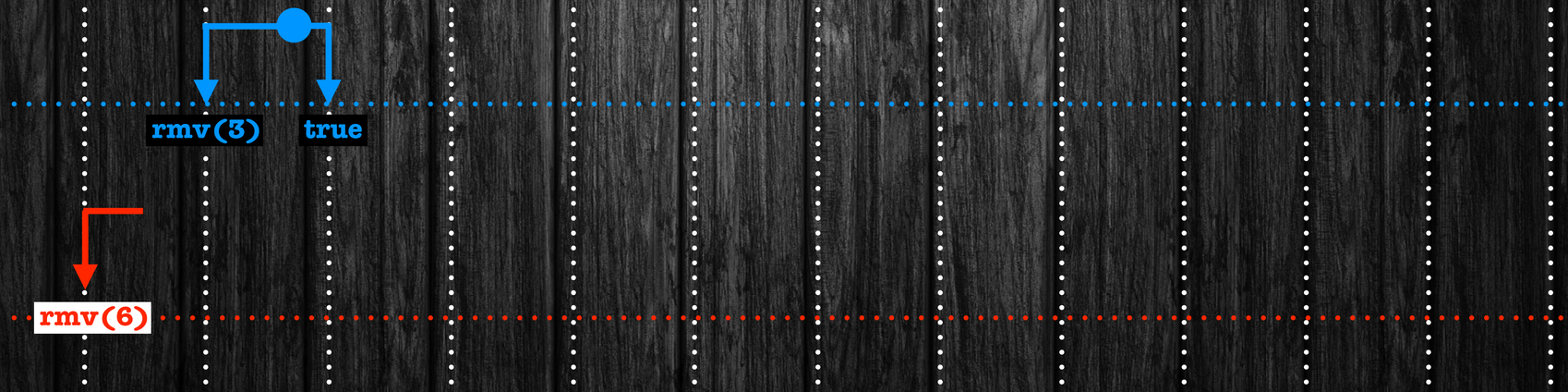


Sets

Optimistic Synchronization

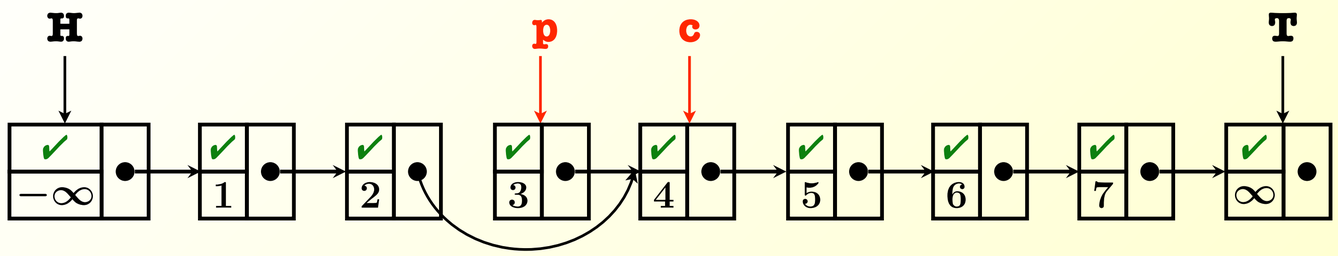
Why Validation?

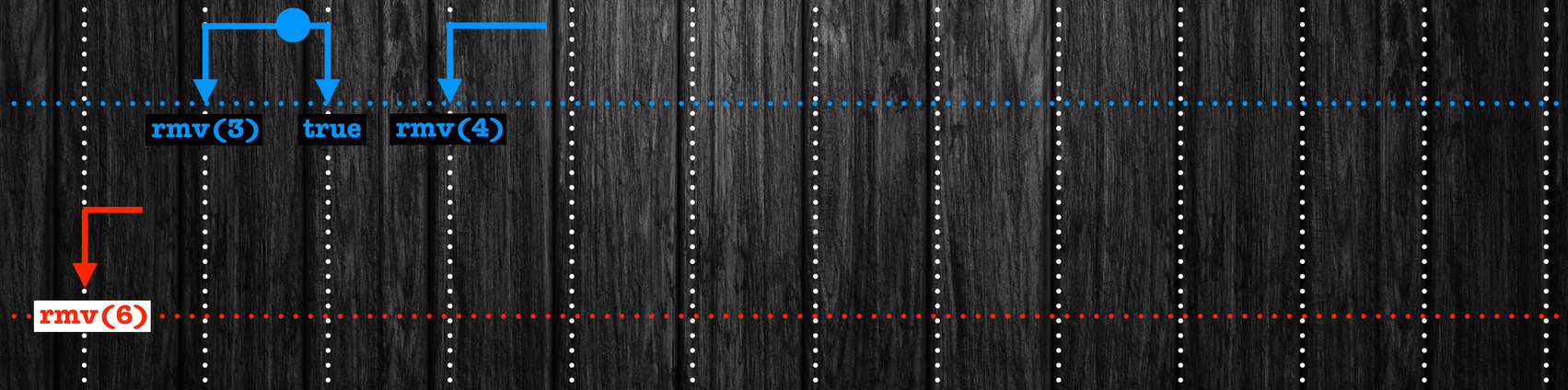




Sets | Optimistic Synchronization

Why Validation?

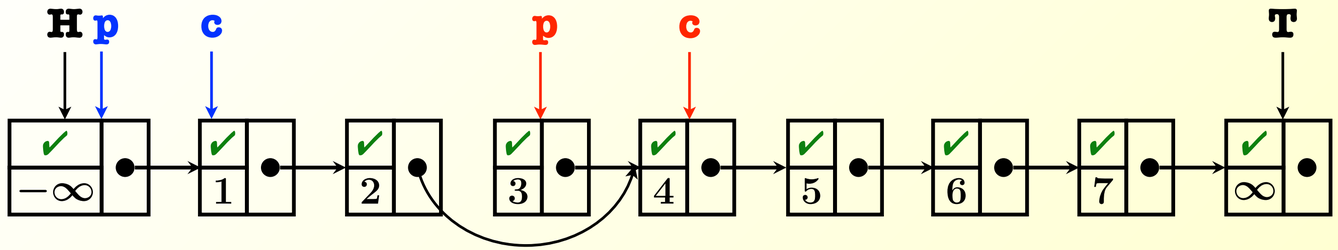


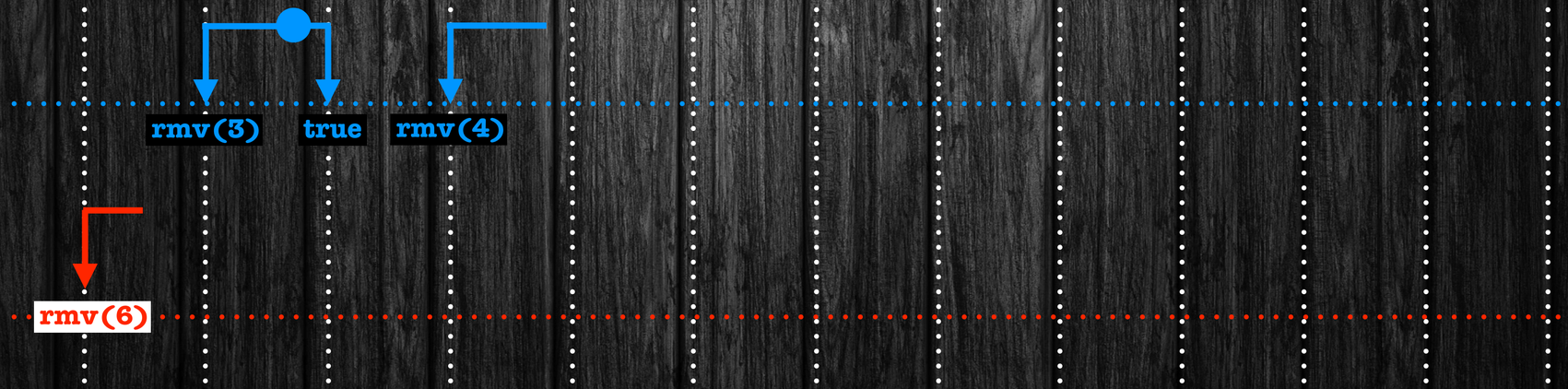


Sets

Optimistic Synchronization

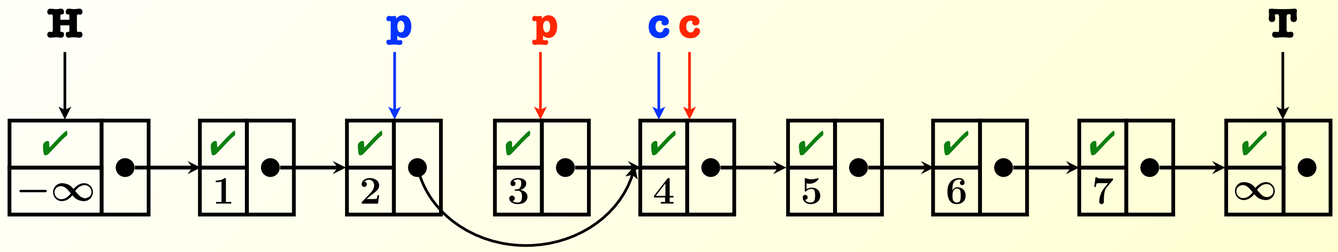
Why Validation?

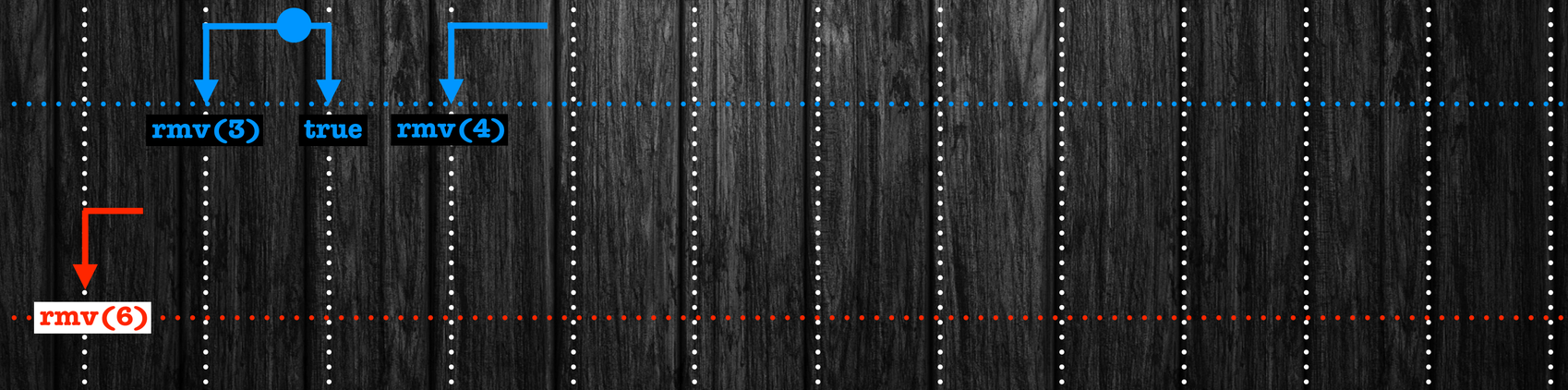




Sets | Optimistic Synchronization

Why Validation?

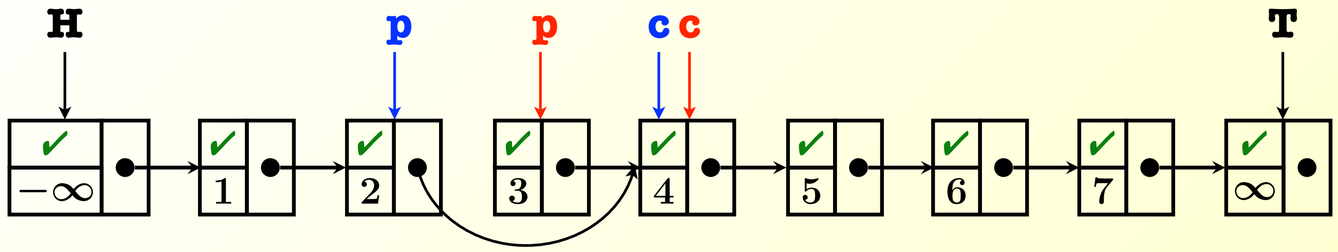


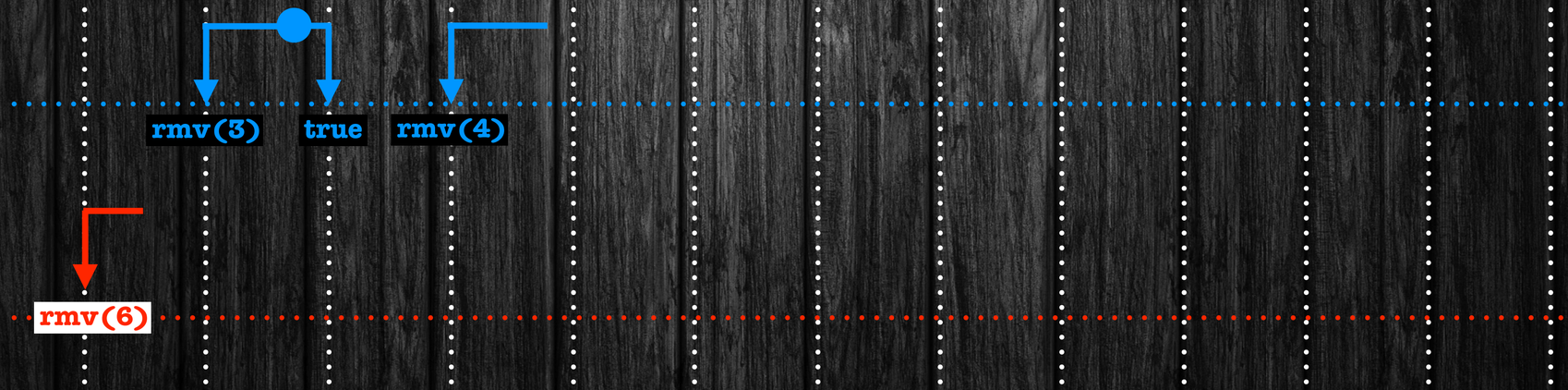


Sets

Optimistic Synchronization

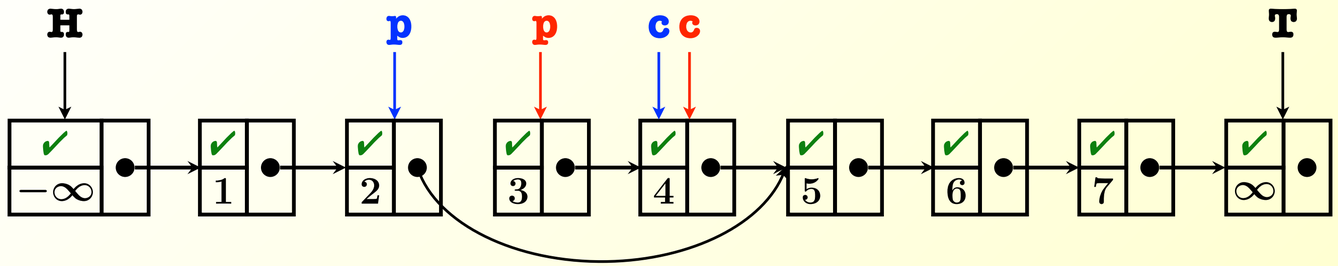
Why Validation?

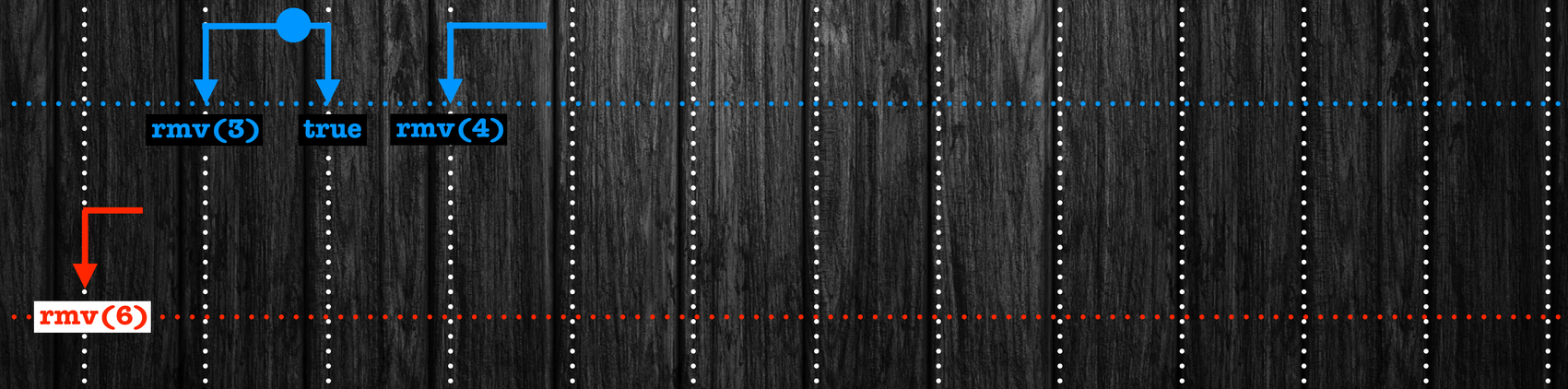




Sets | Optimistic Synchronization

Why Validation?

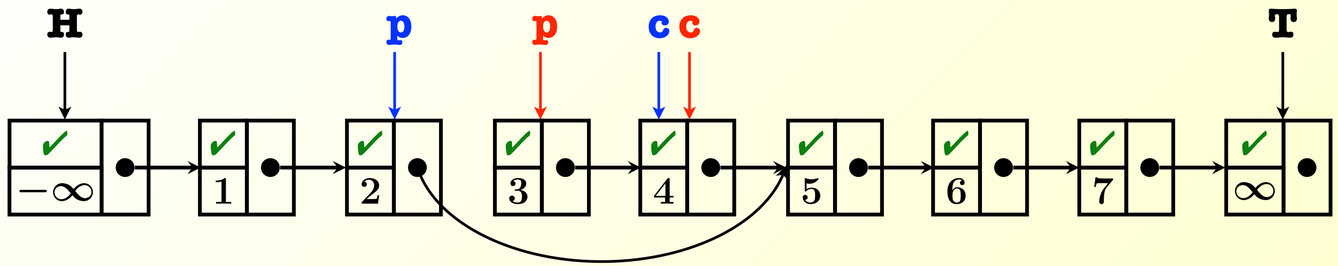


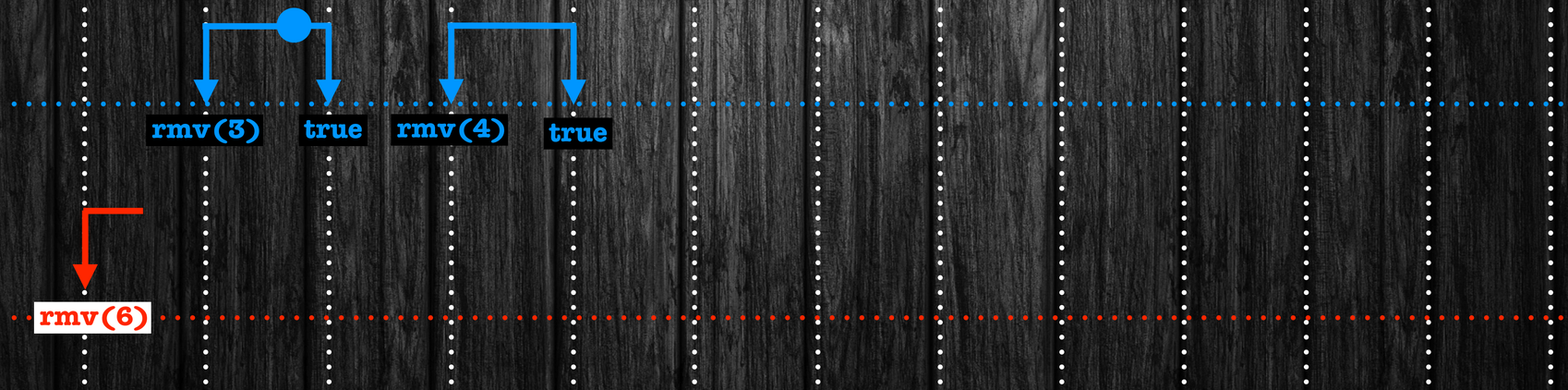


Sets

Optimistic Synchronization

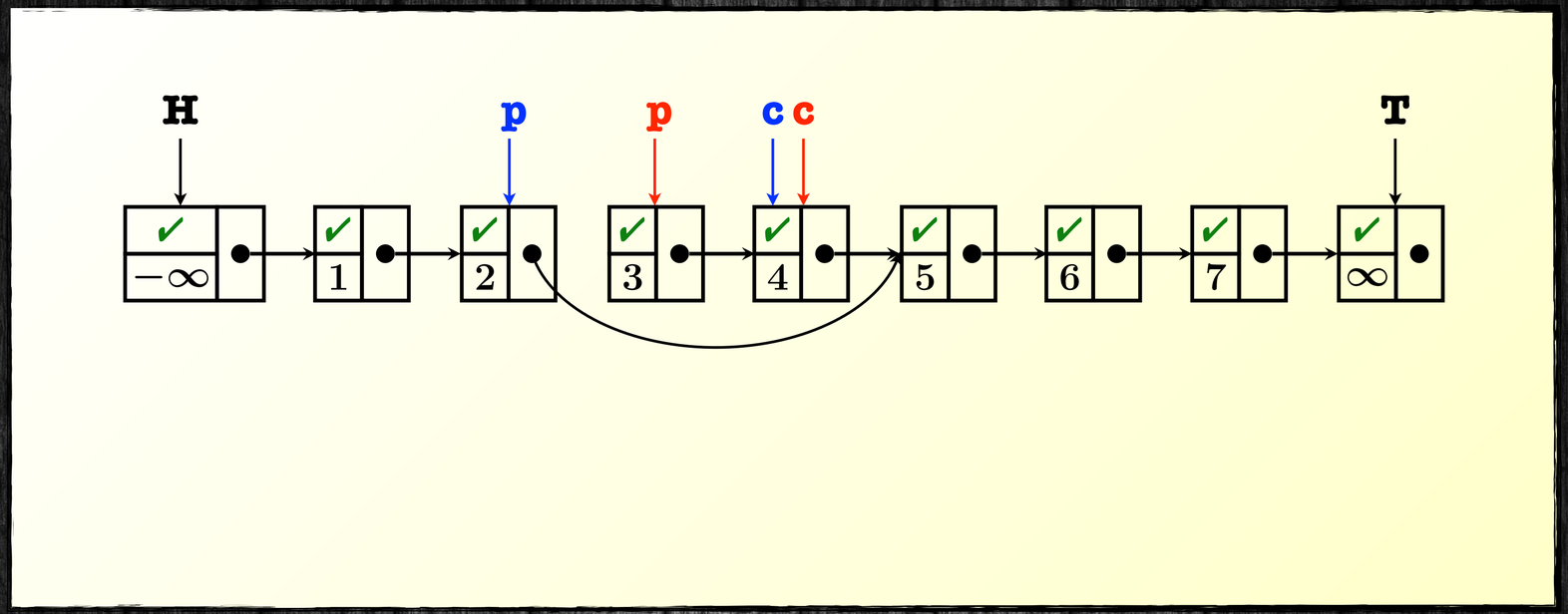
Why Validation?

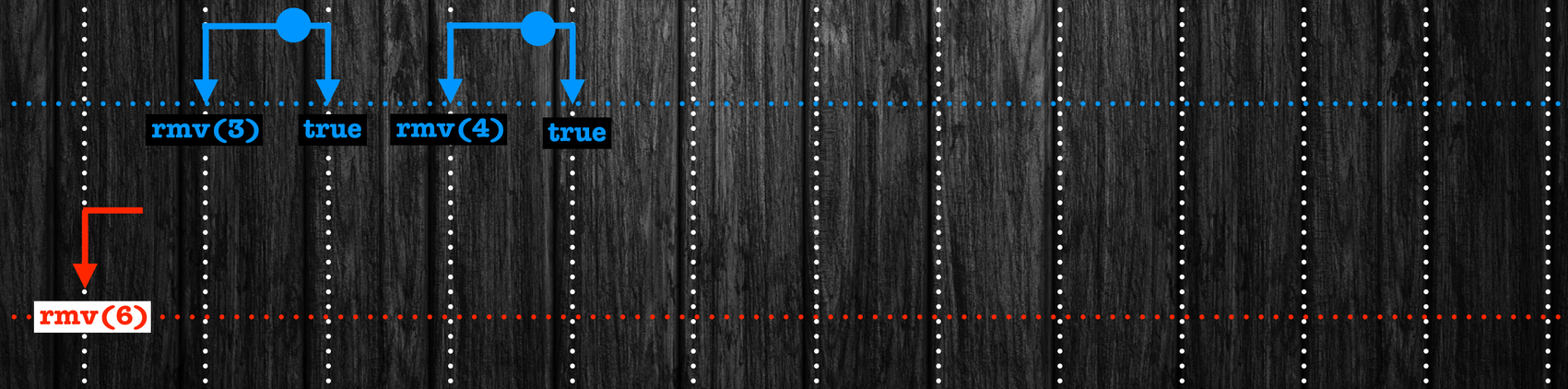




Sets Optimistic Synchronization

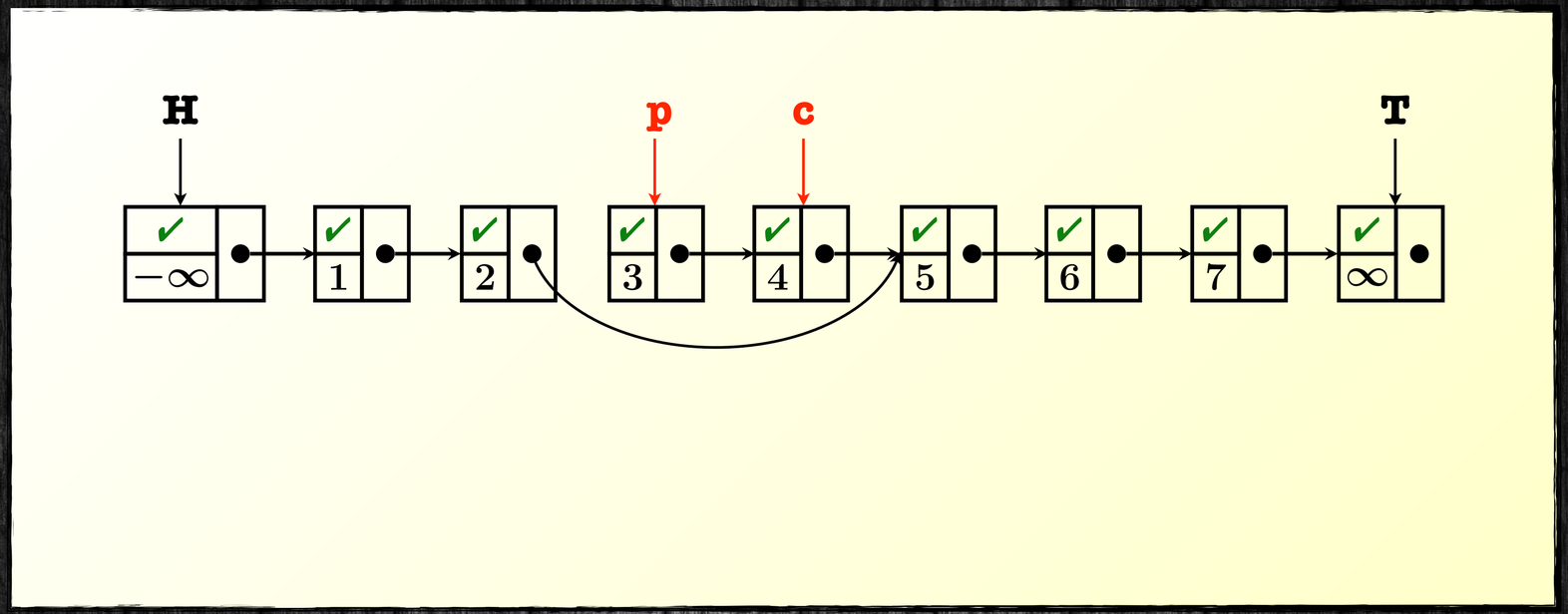
Why Validation?

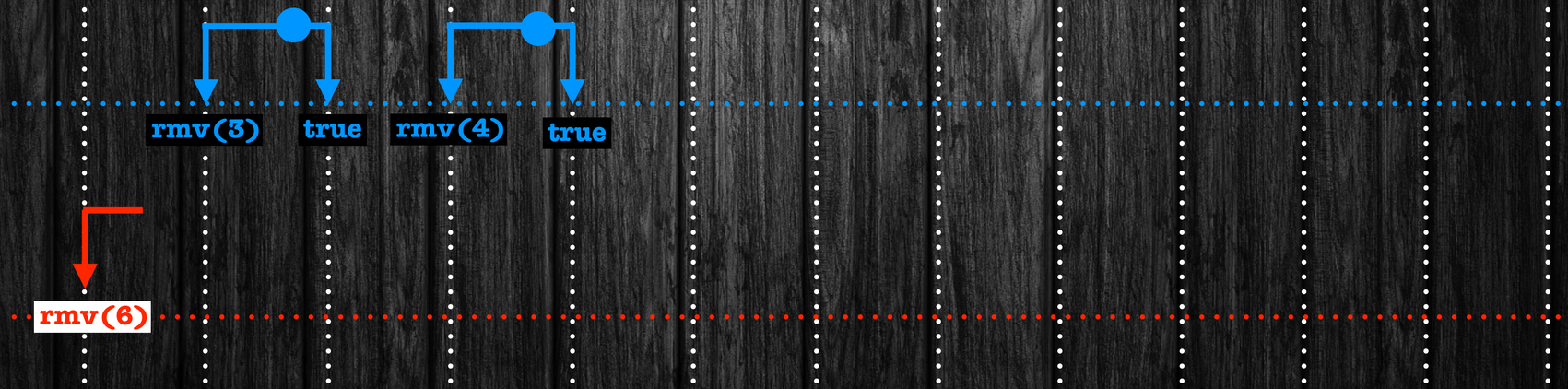




Sets | Optimistic Synchronization

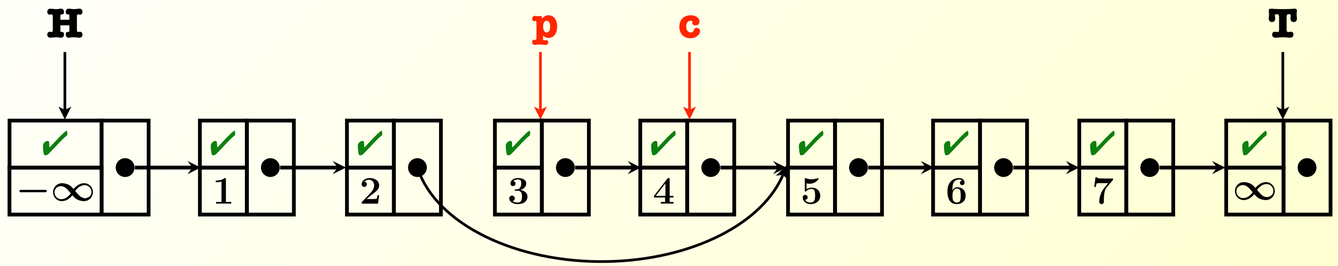
Why Validation?

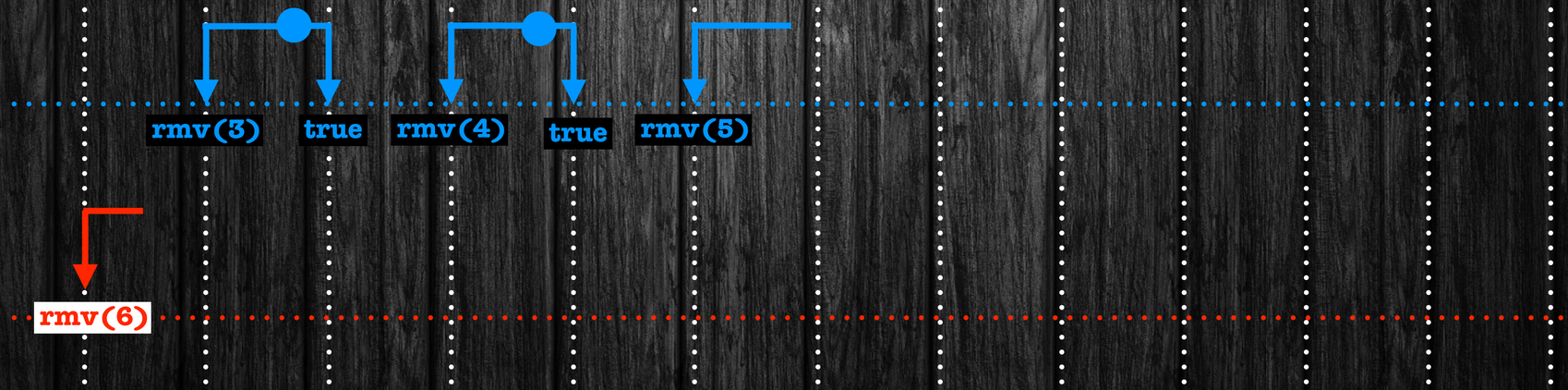




Sets | Optimistic Synchronization

Why Validation?

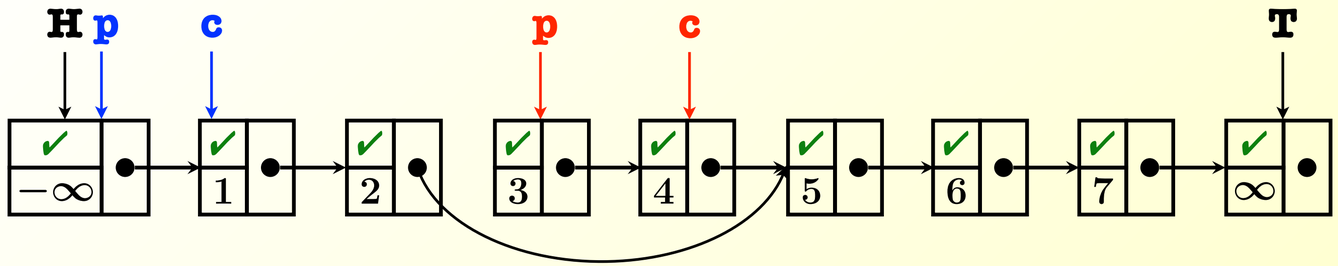


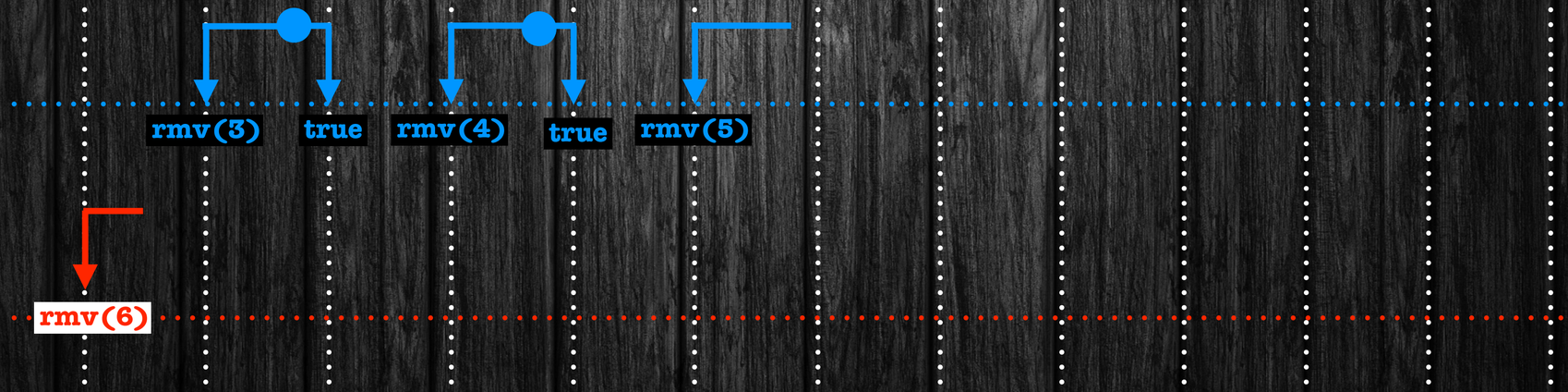


Sets

Optimistic Synchronization

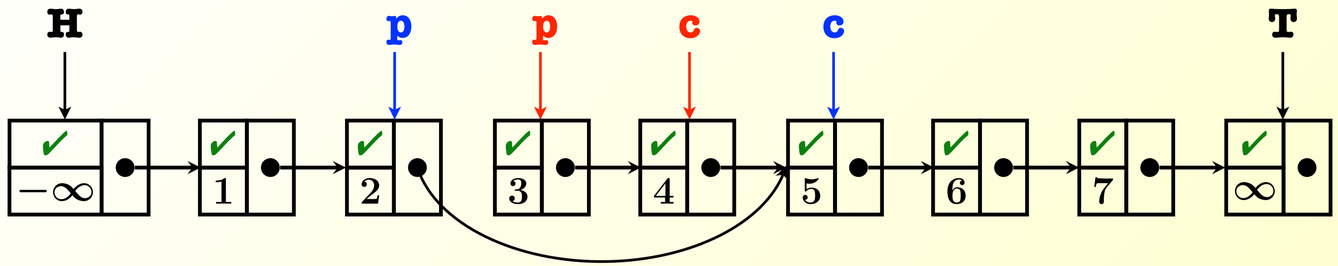
Why Validation?

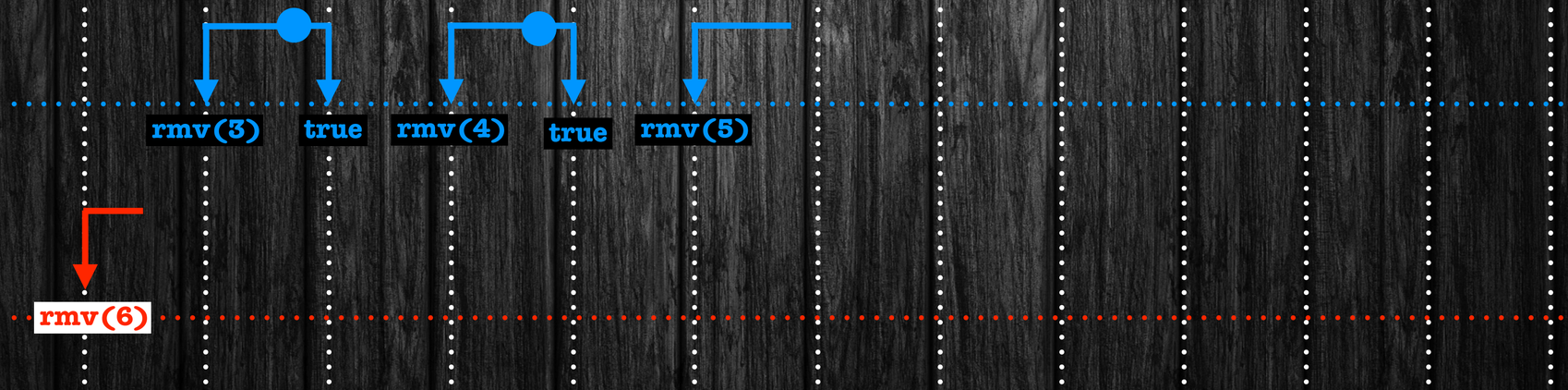




Sets | Optimistic Synchronization

Why Validation?

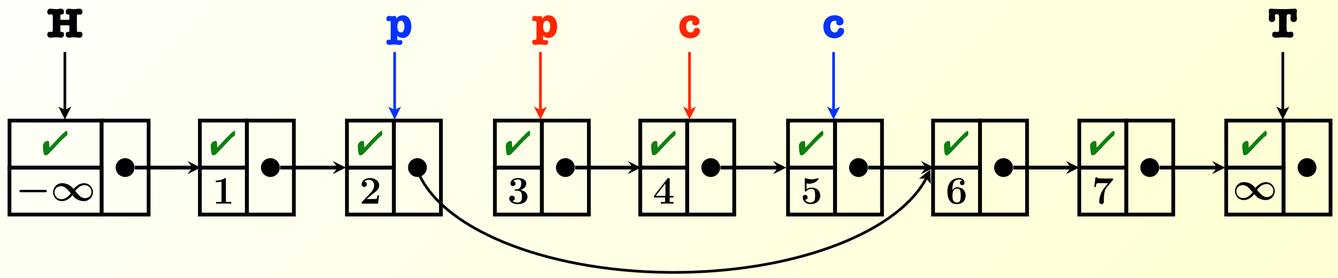


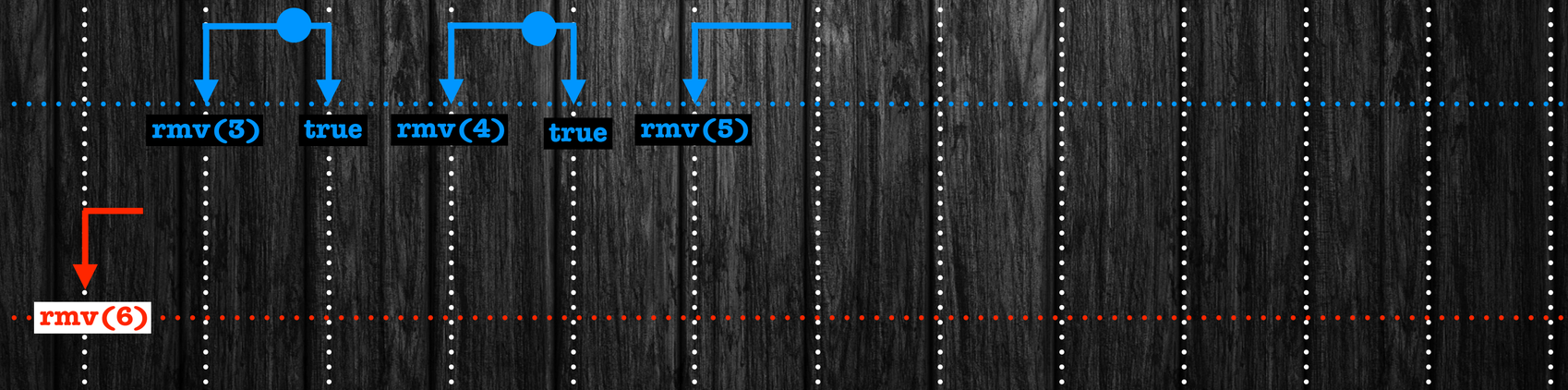


Sets

Optimistic Synchronization

Why Validation?

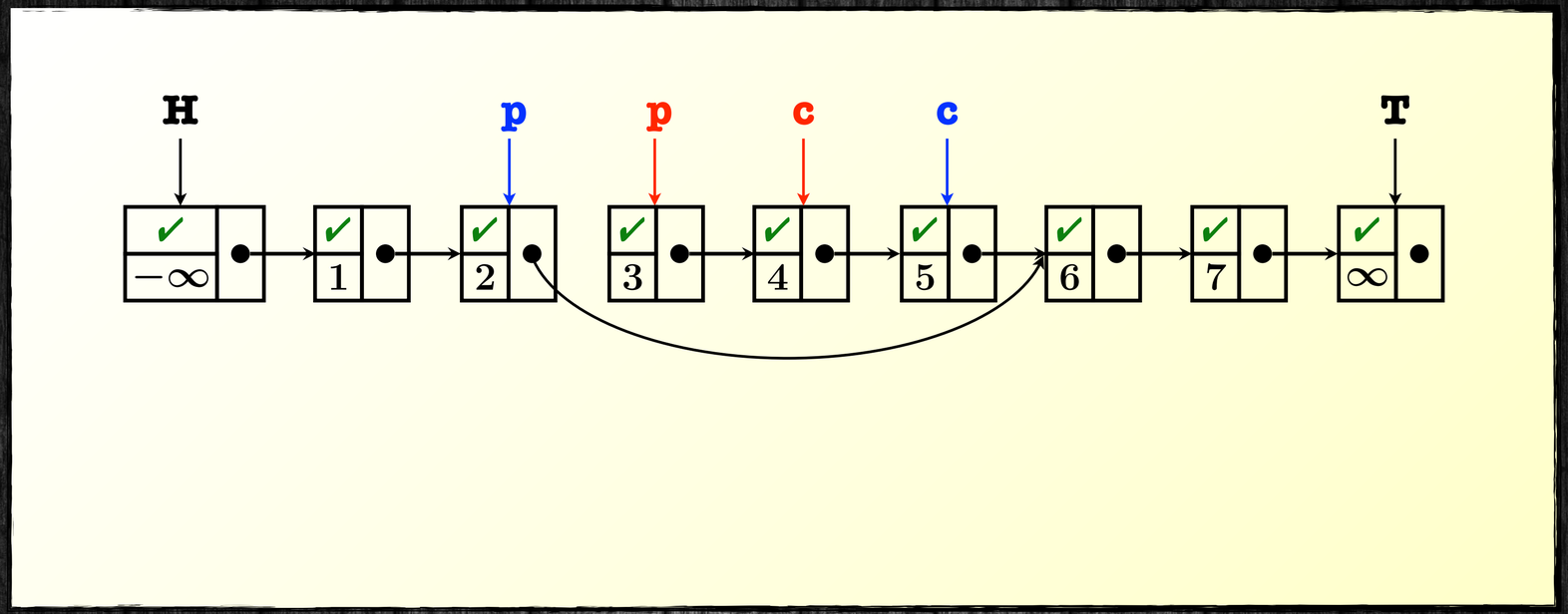


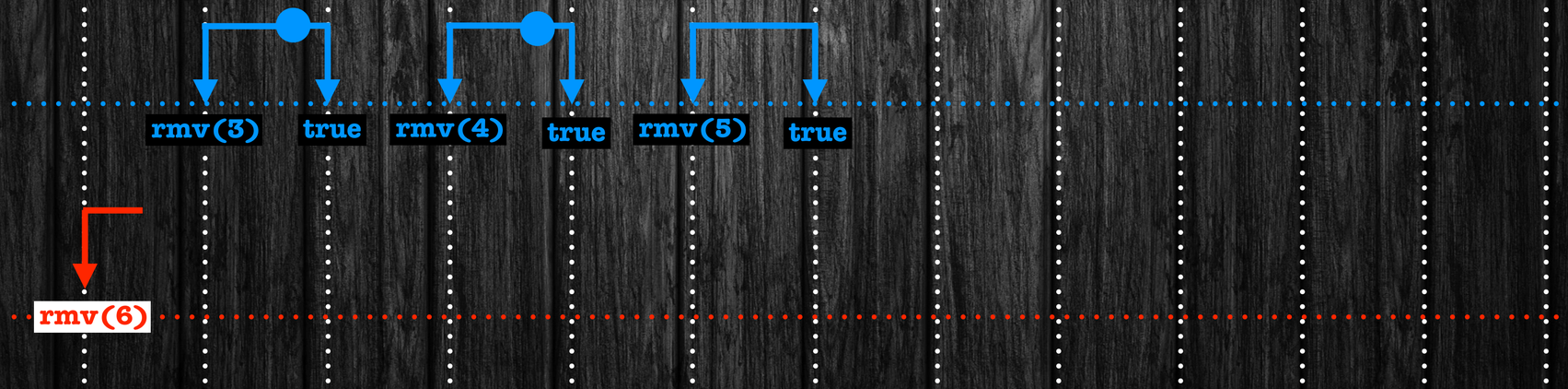


Sets

Optimistic Synchronization

Why Validation?

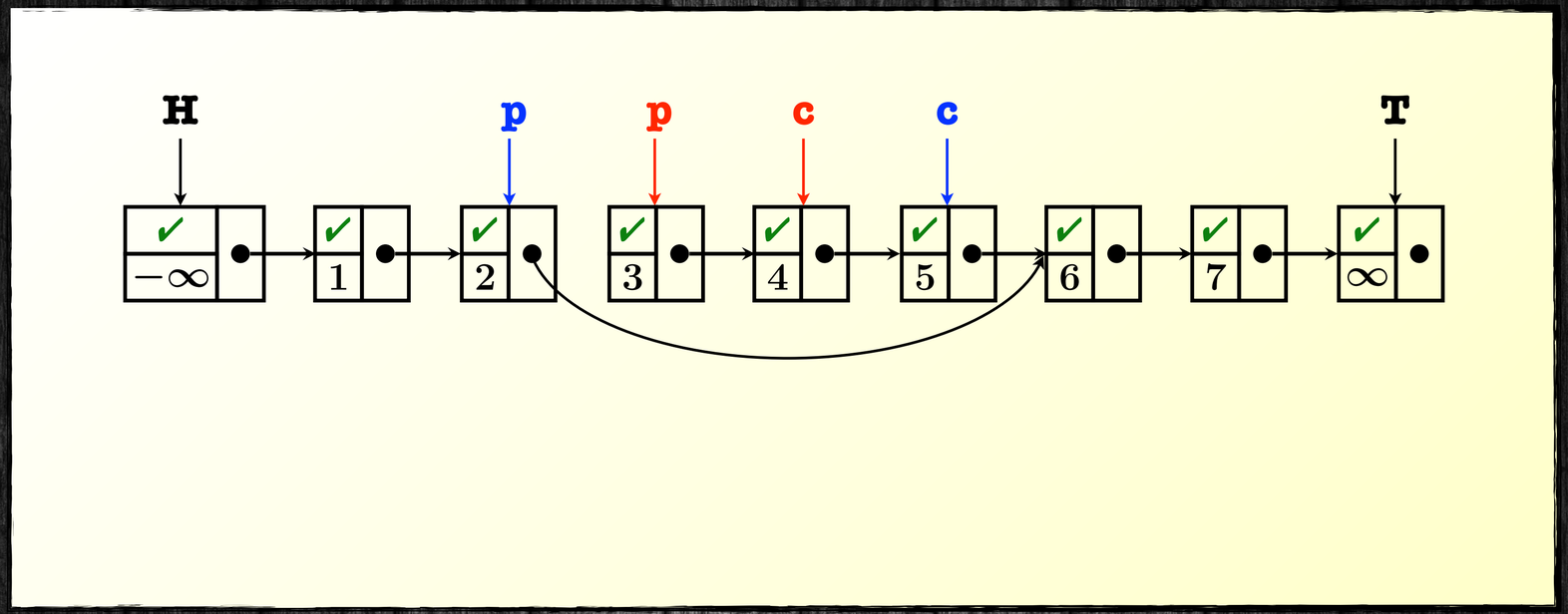


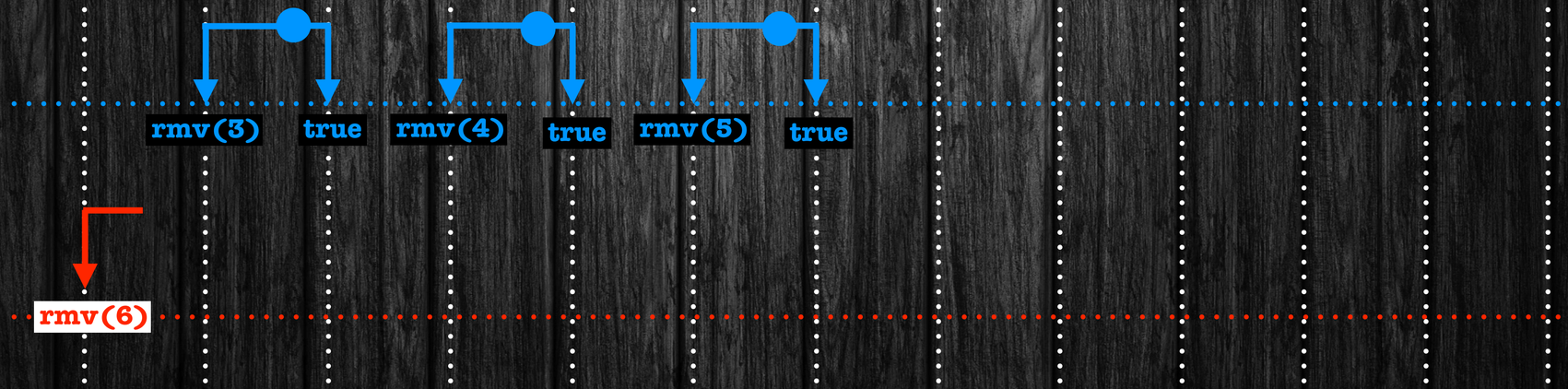


Sets

Optimistic Synchronization

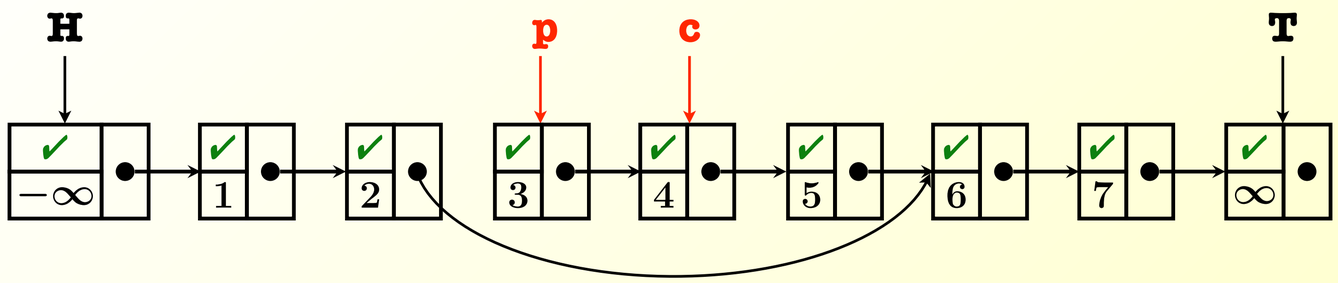
Why Validation?

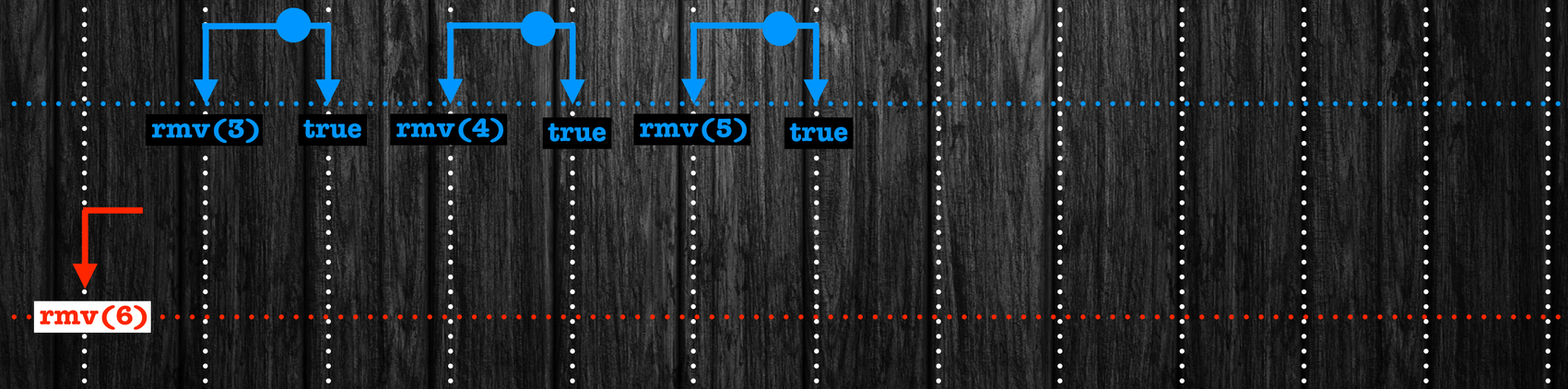




Sets | Optimistic Synchronization

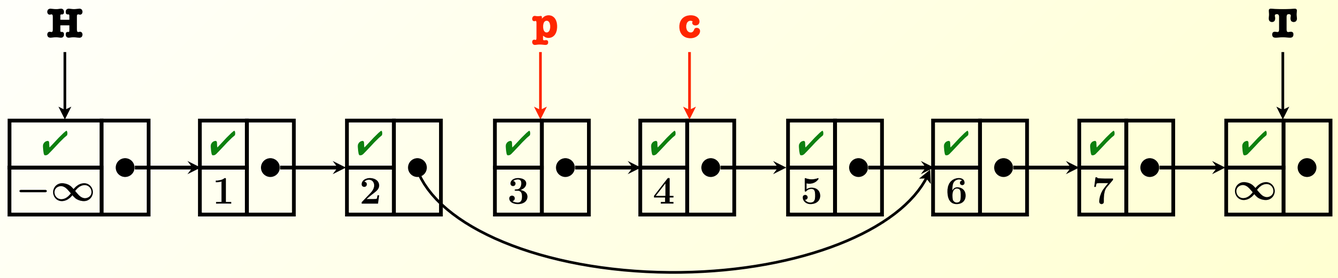
Why Validation?

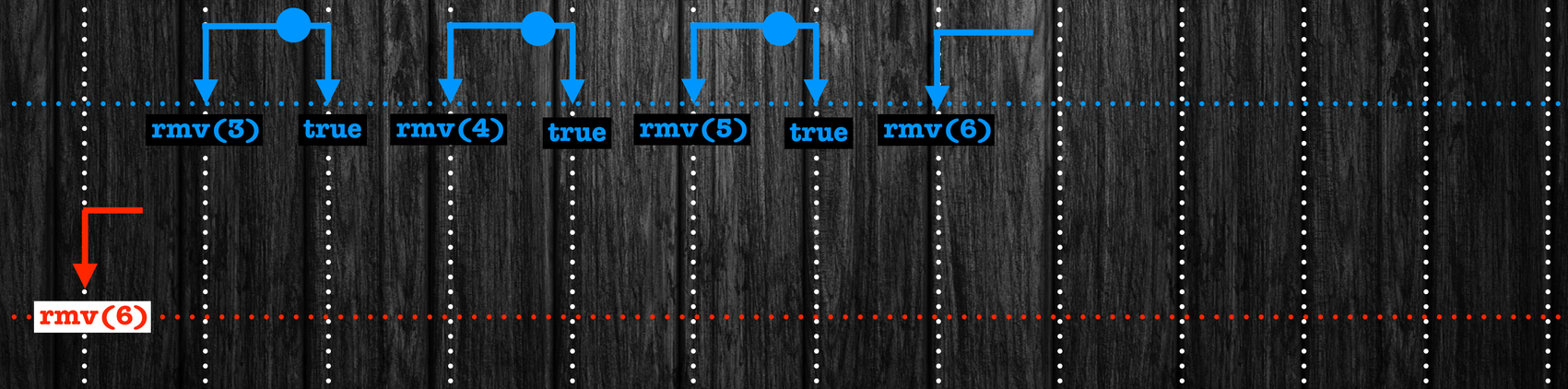




Sets Optimistic Synchronization

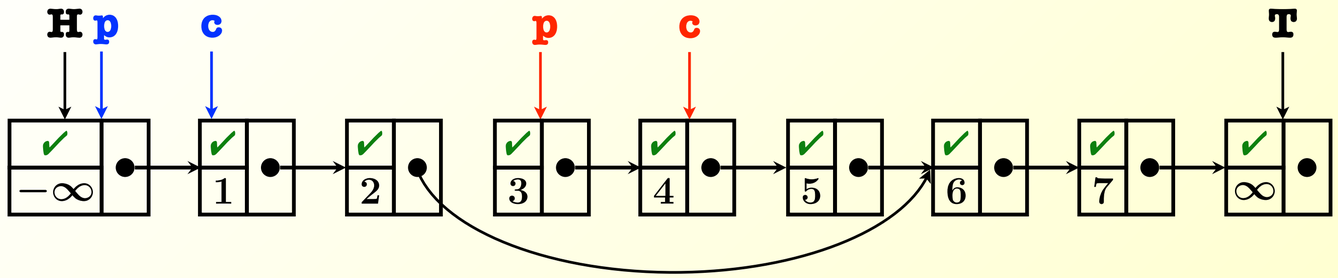
Why Validation?

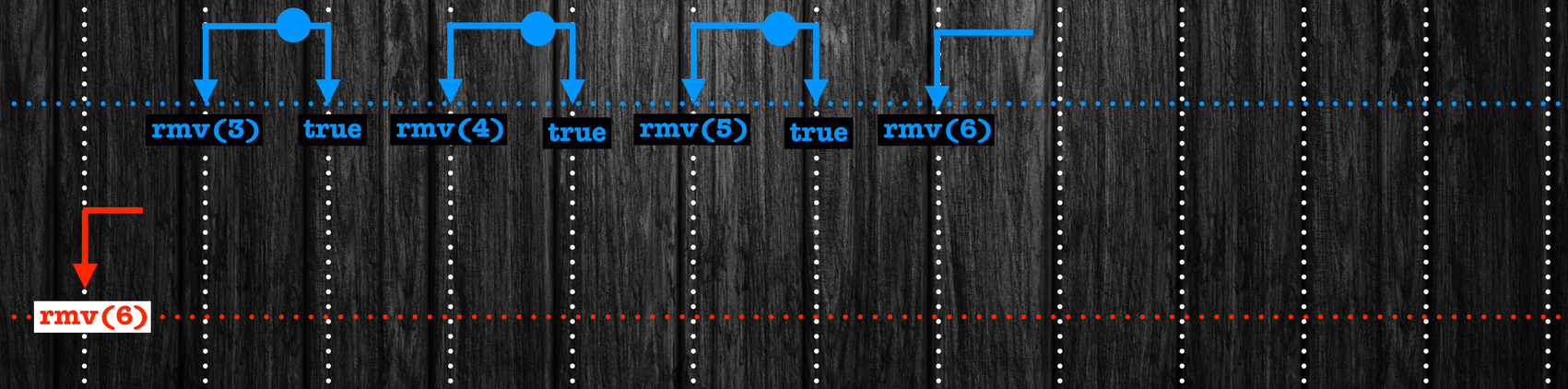




Sets Optimistic Synchronization

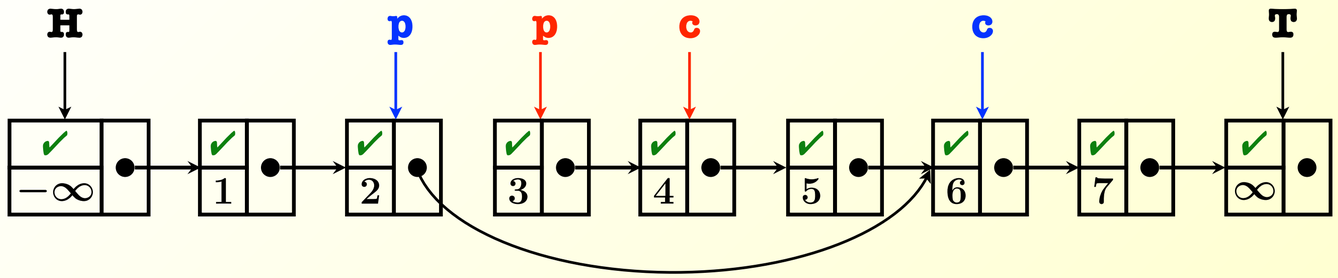
Why Validation?

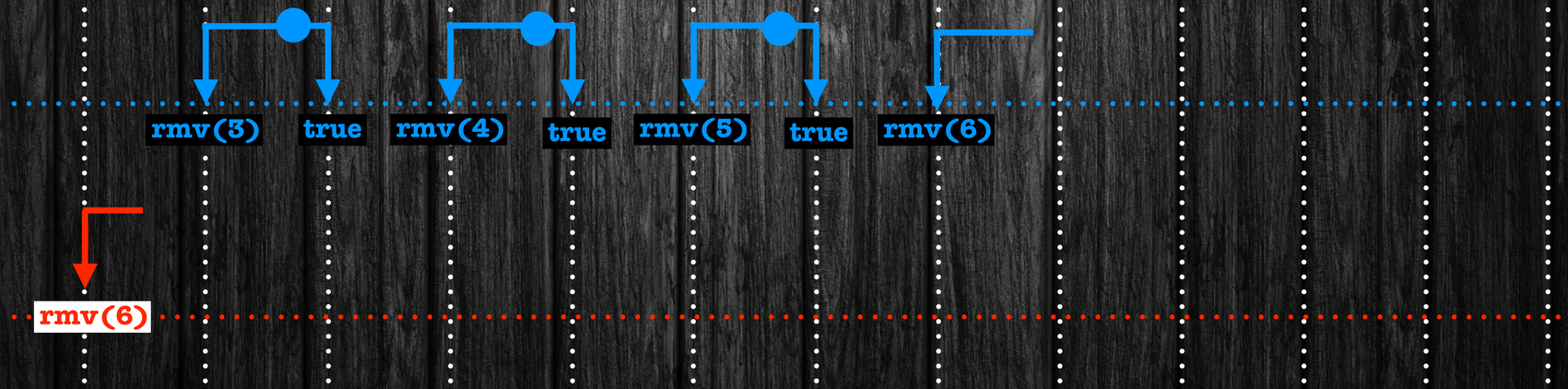




Sets Optimistic Synchronization

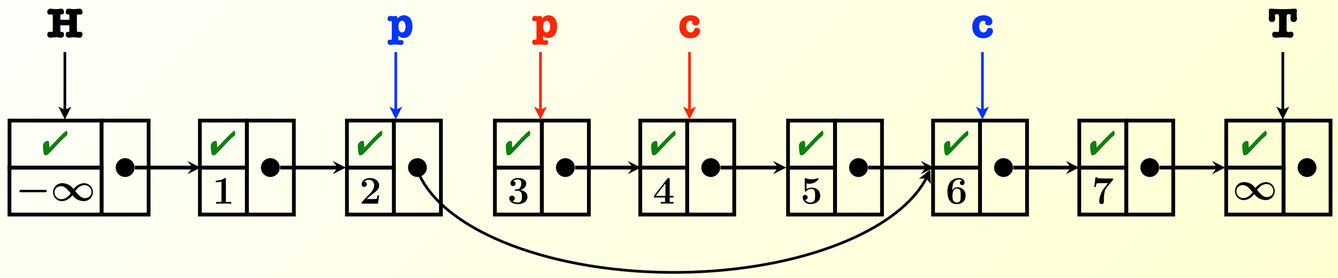
Why Validation?

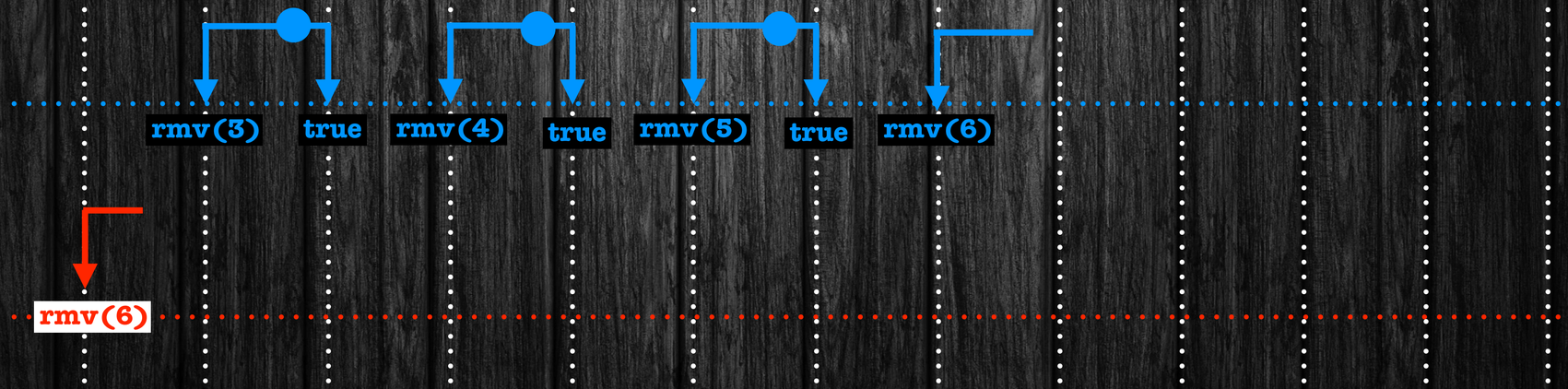




Sets Optimistic Synchronization

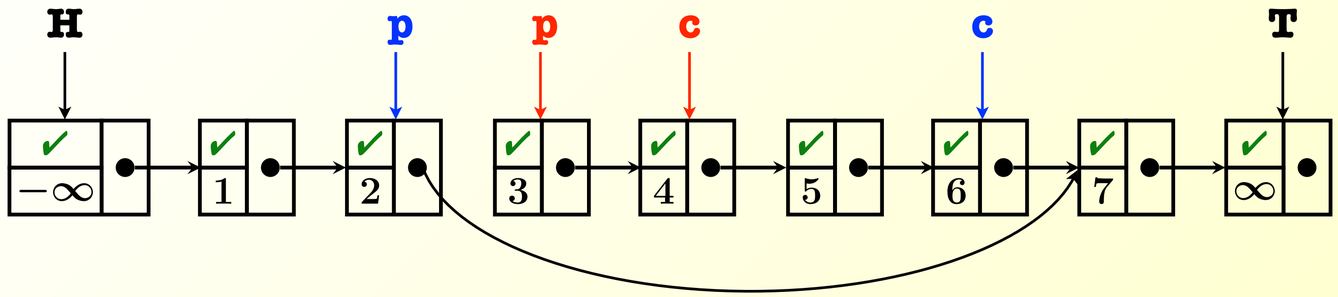
Why Validation?

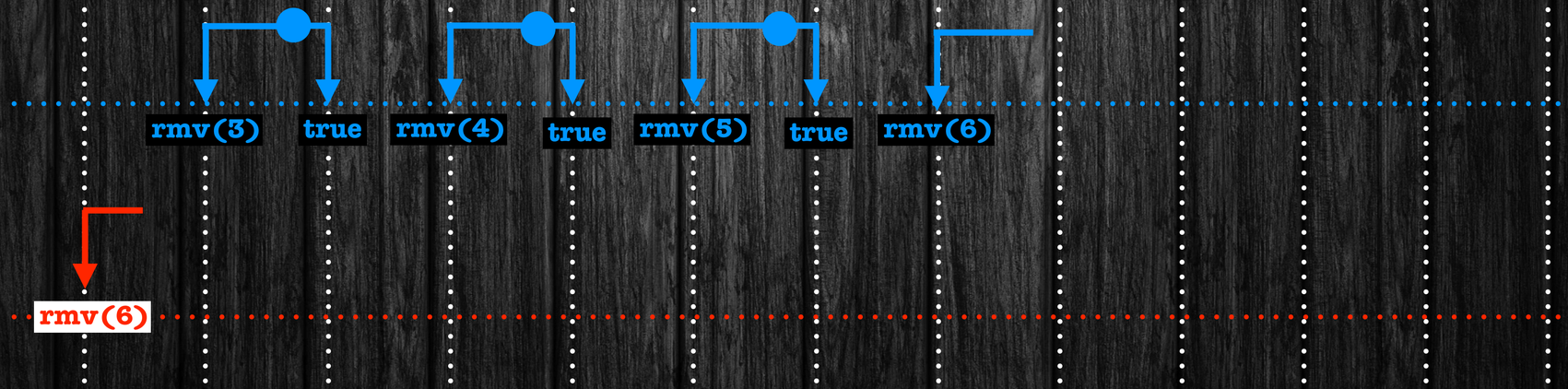




Sets Optimistic Synchronization

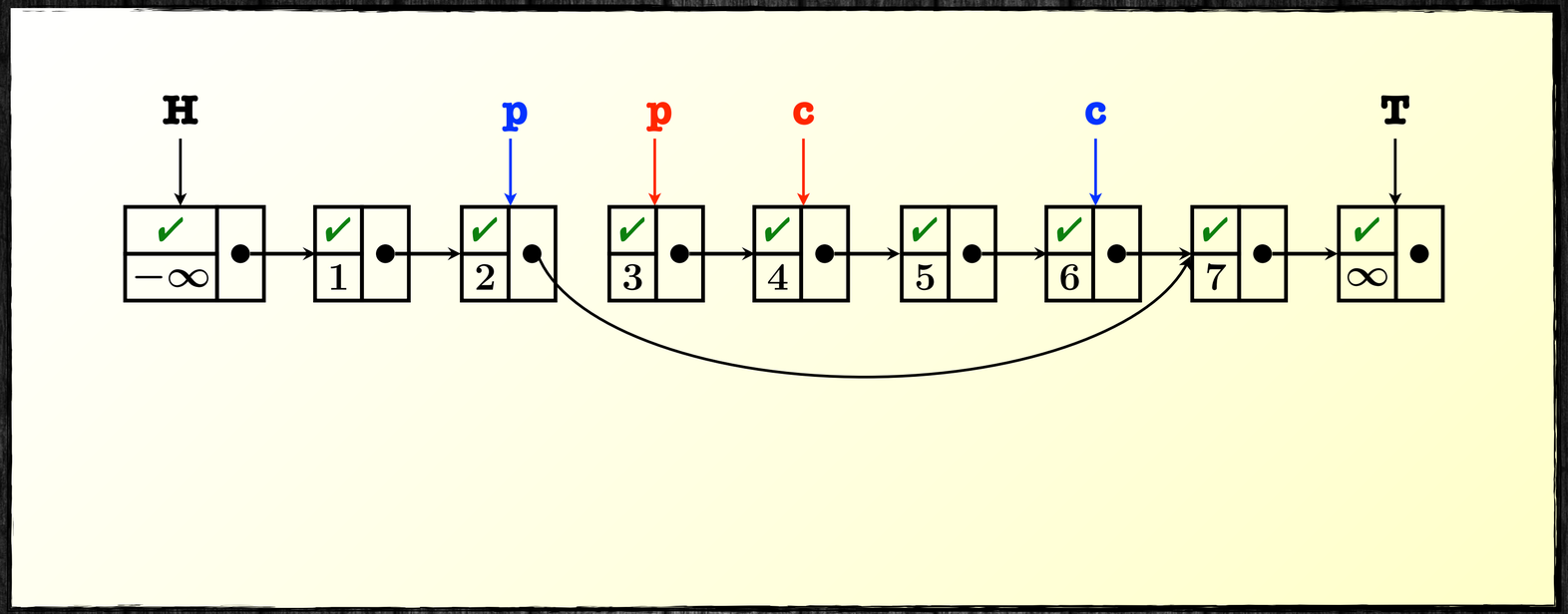
Why Validation?

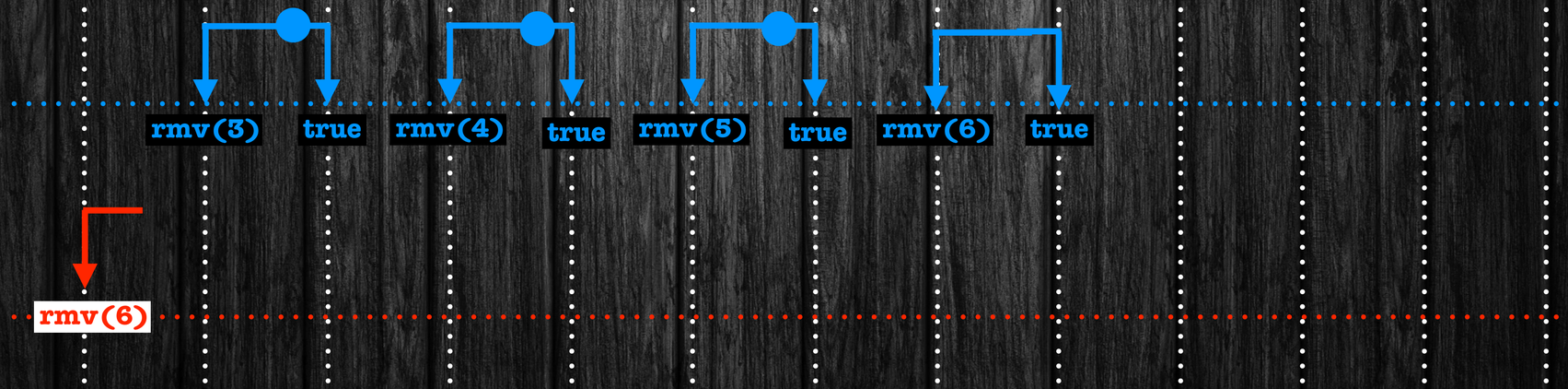




Sets Optimistic Synchronization

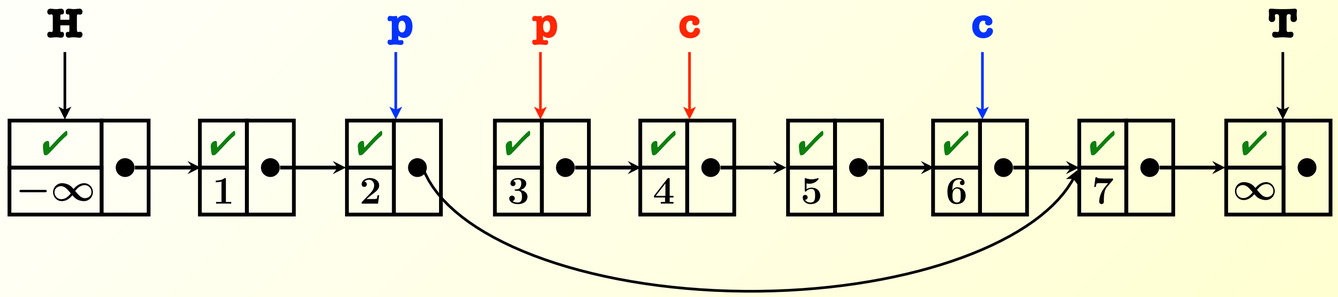
Why Validation?

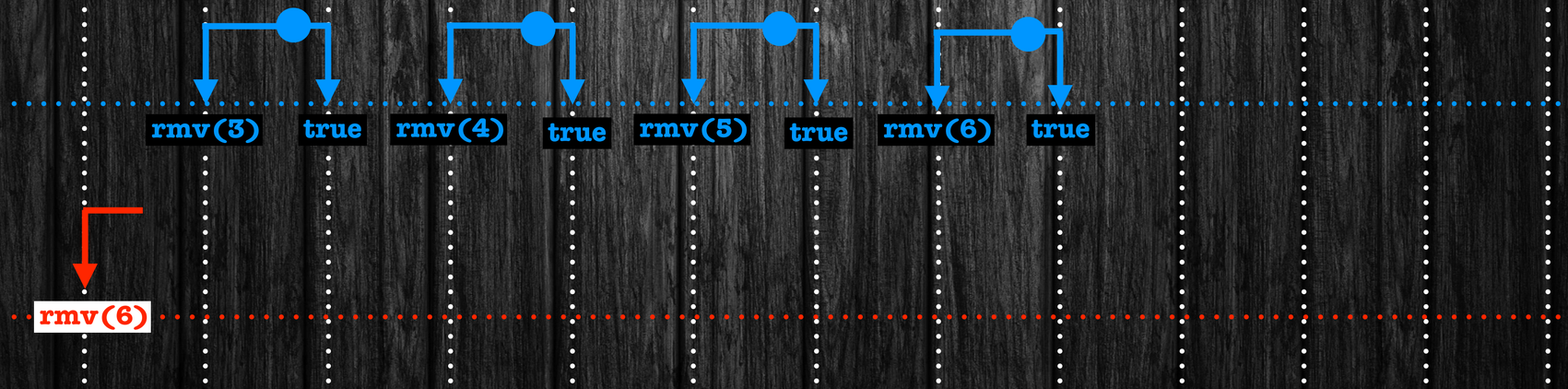




Sets Optimistic Synchronization

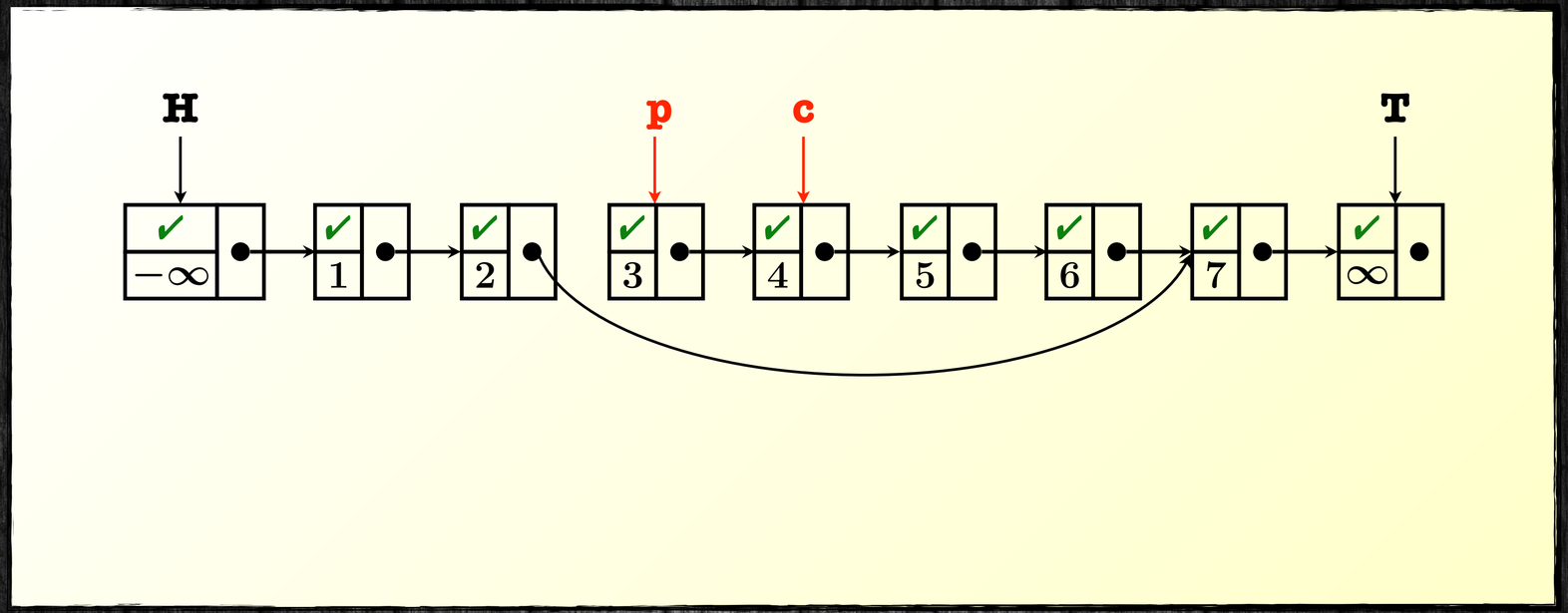
Why Validation?

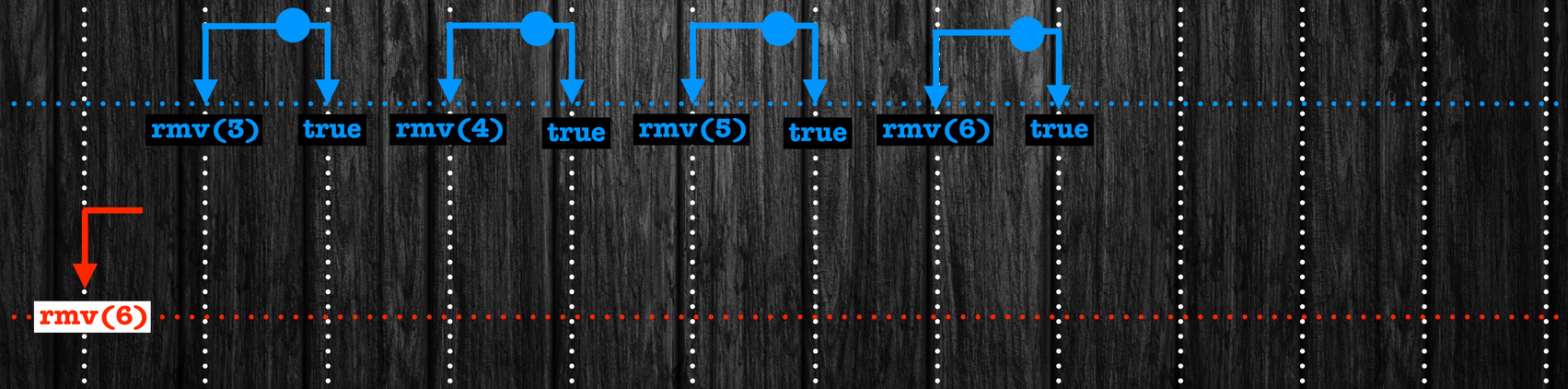




Sets Optimistic Synchronization

Why Validation?

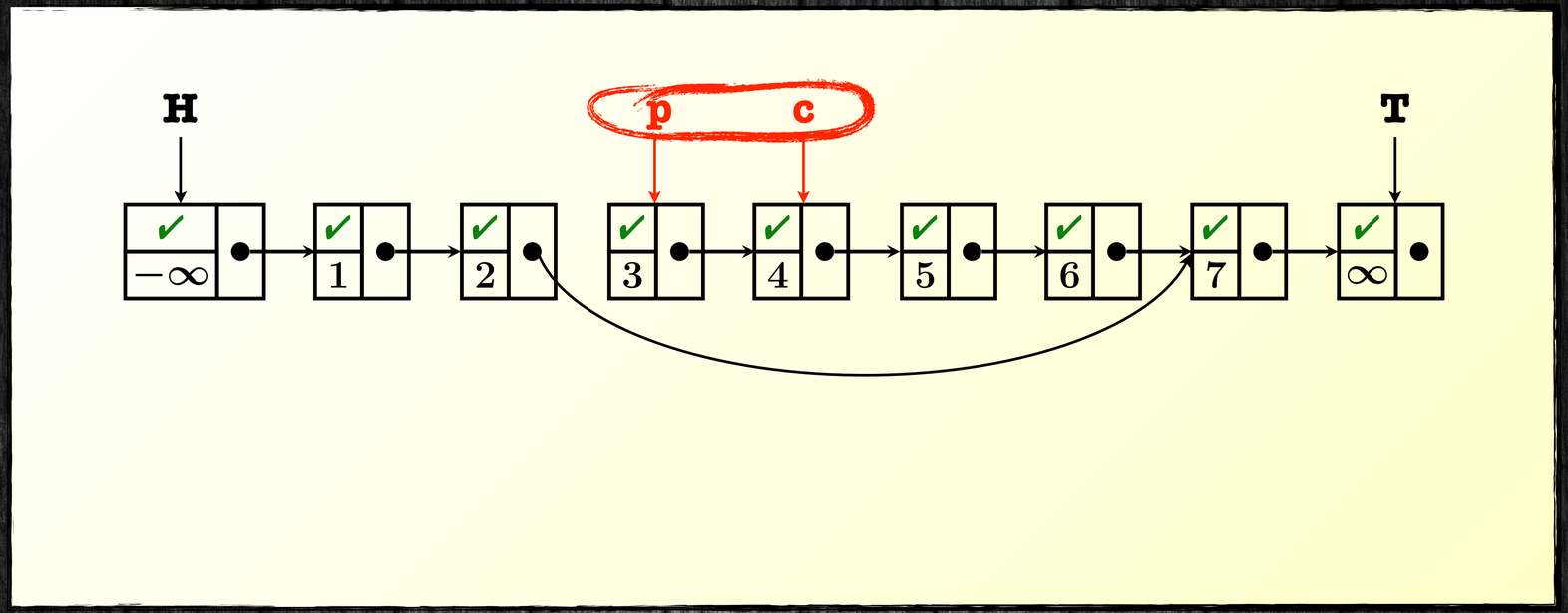


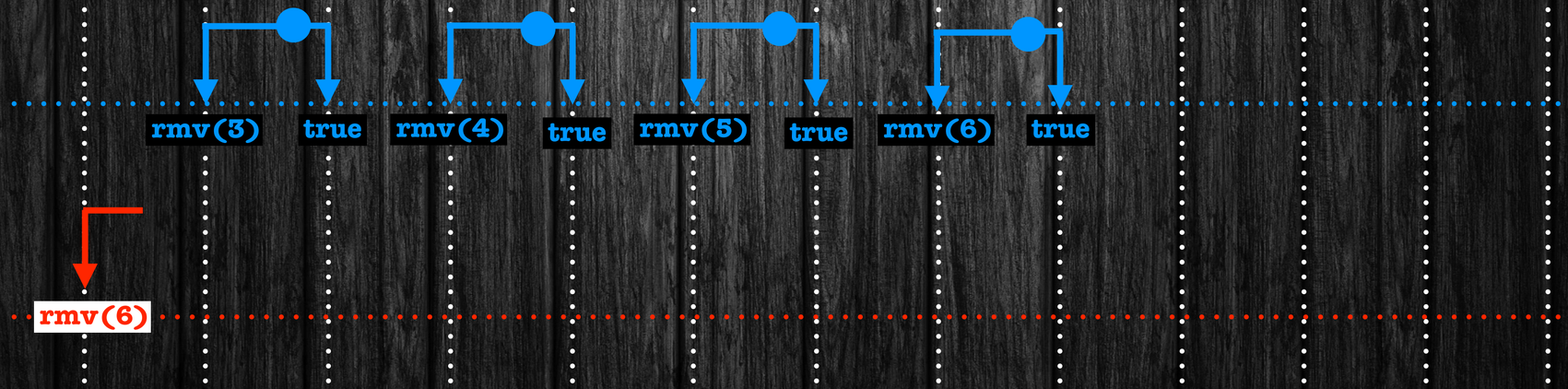


Sets

Optimistic Synchronization

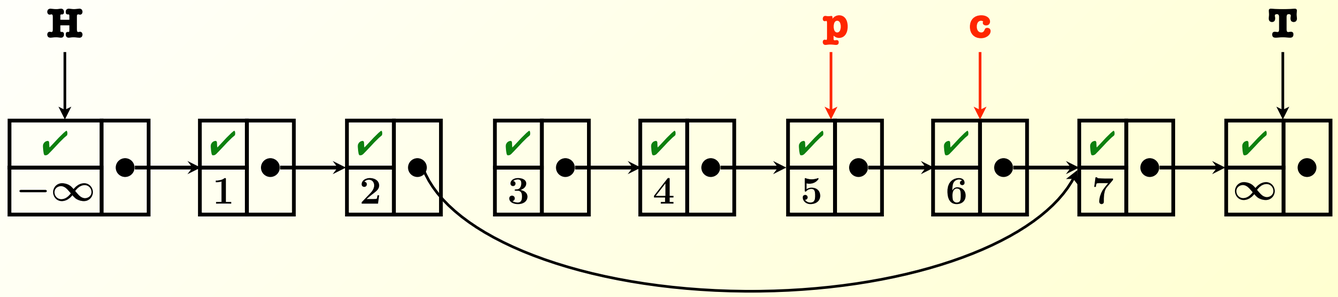
Why Validation?

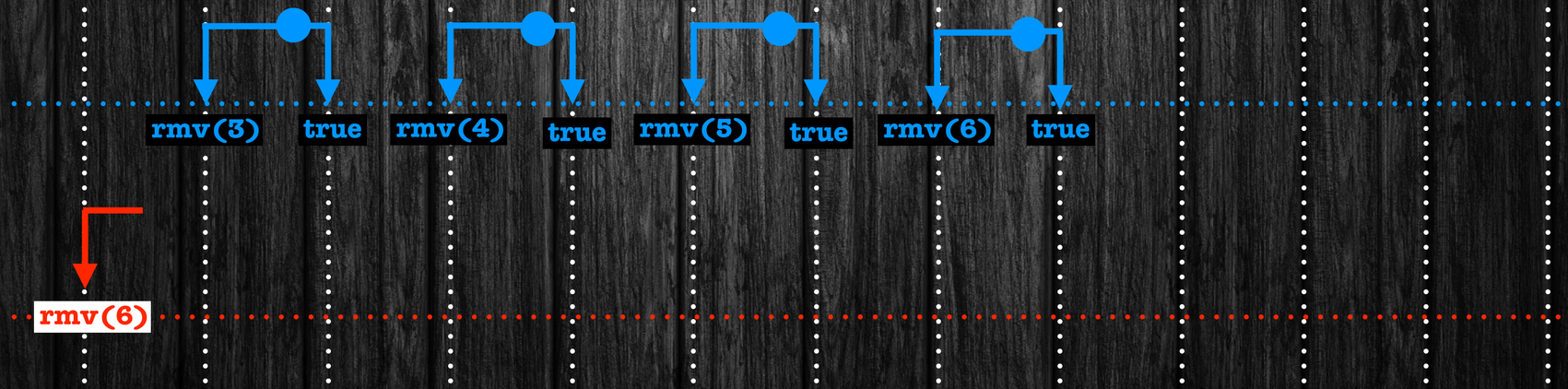




Sets | Optimistic Synchronization

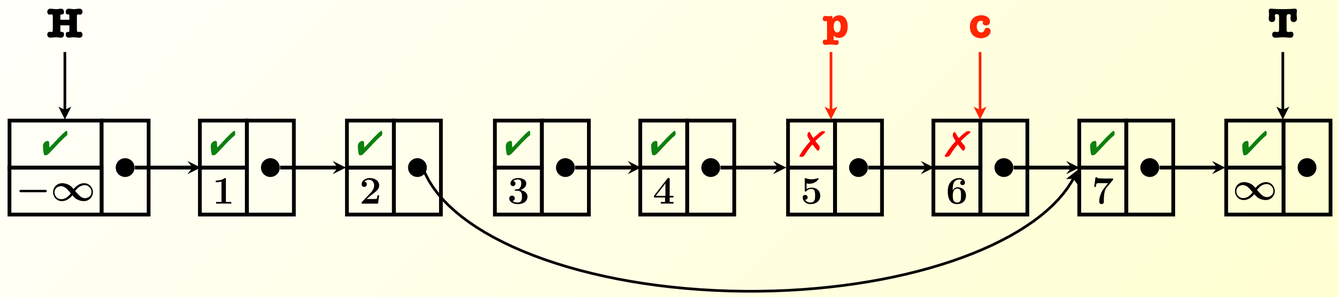
Why Validation?

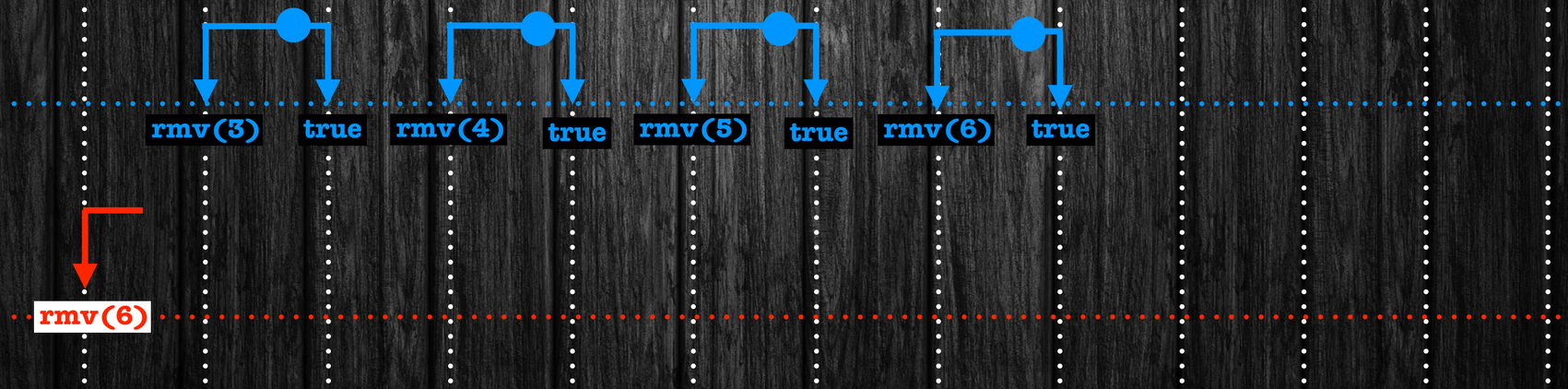




Sets | Optimistic Synchronization

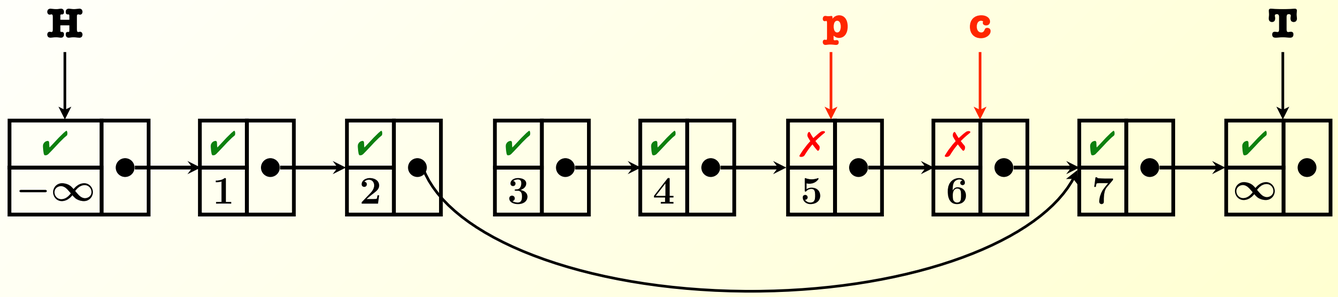
Why Validation?



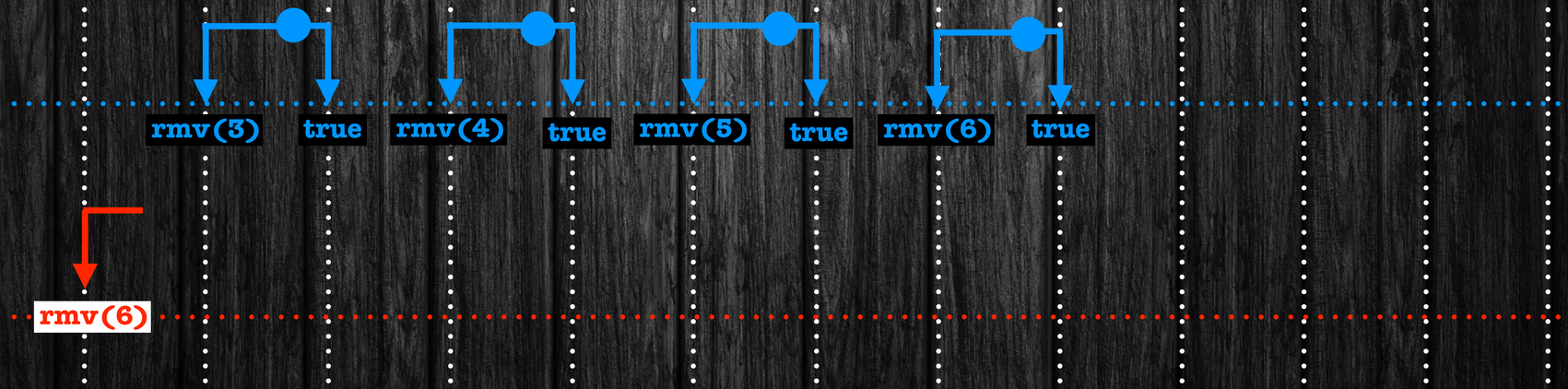


Sets Optimistic Synchronization

Why Validation?

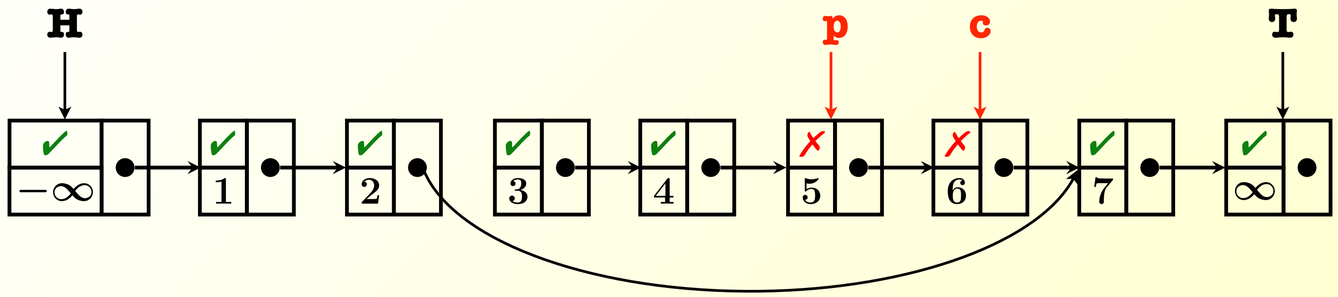


if no validation

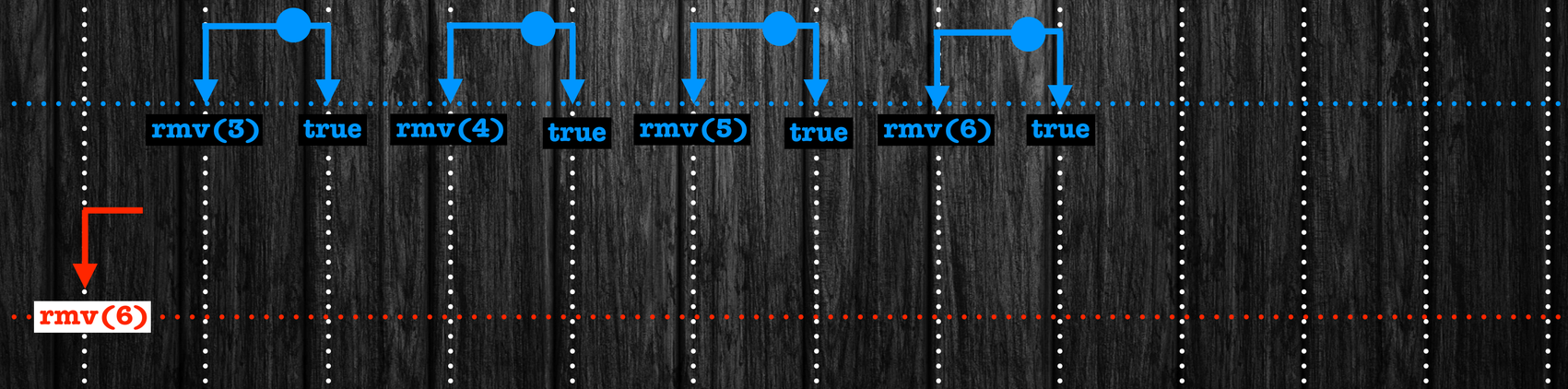


Sets Optimistic Synchronization

Why Validation?

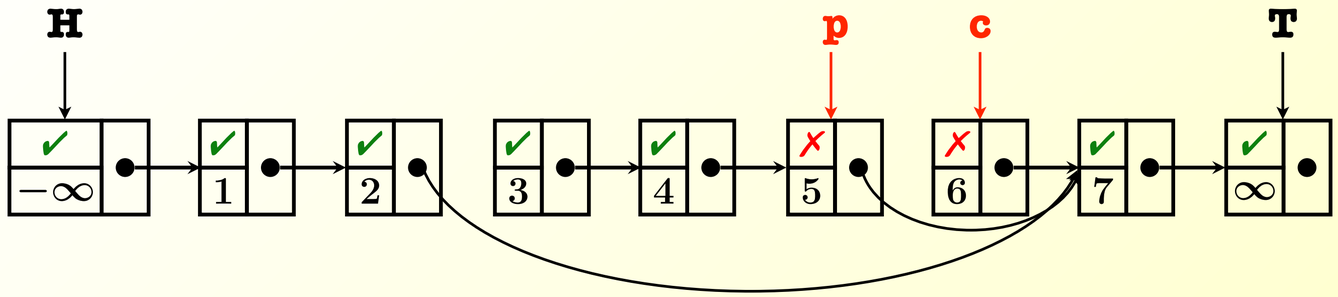


if no validation

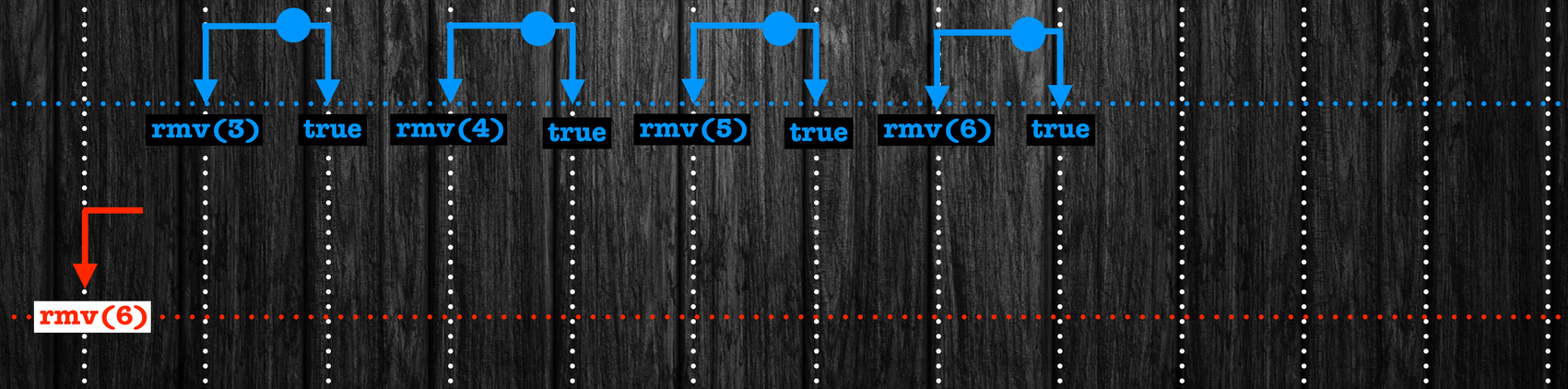


Sets Optimistic Synchronization

Why Validation?

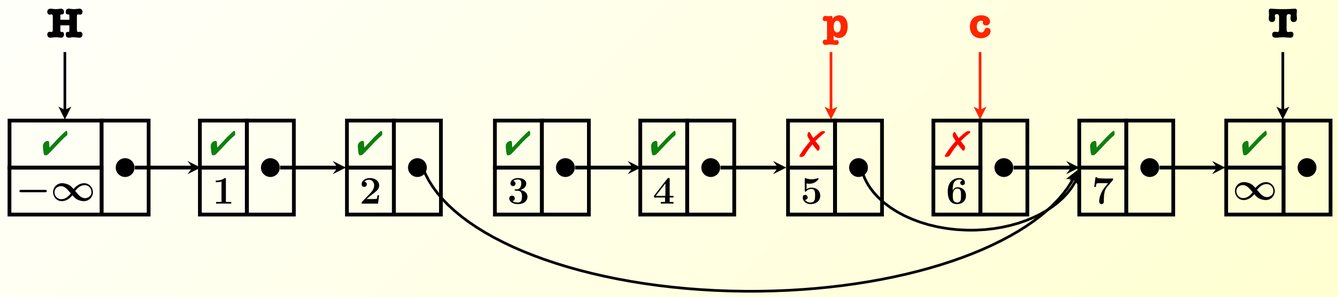


if no validation

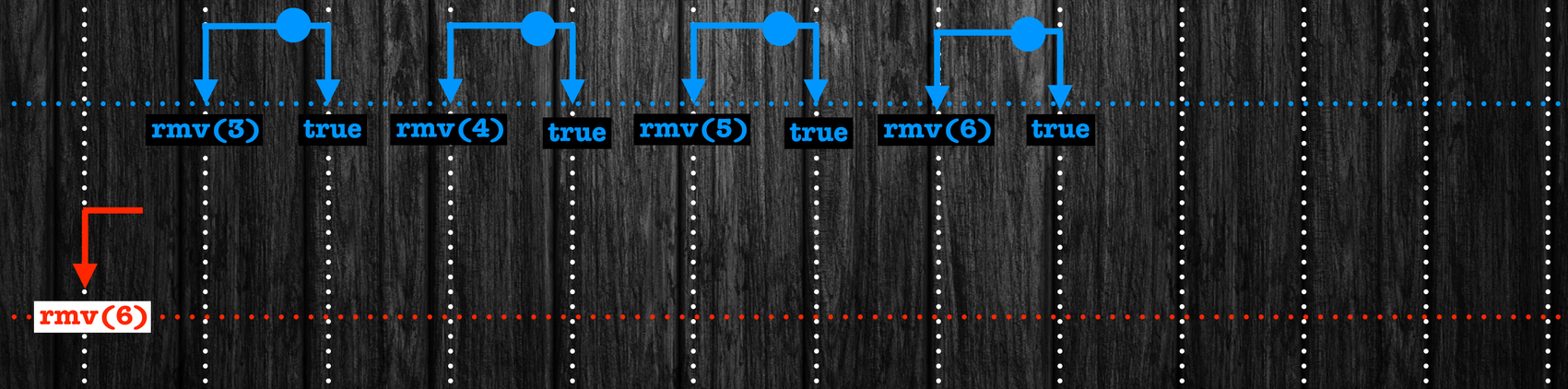


Sets Optimistic Synchronization

Why Validation?

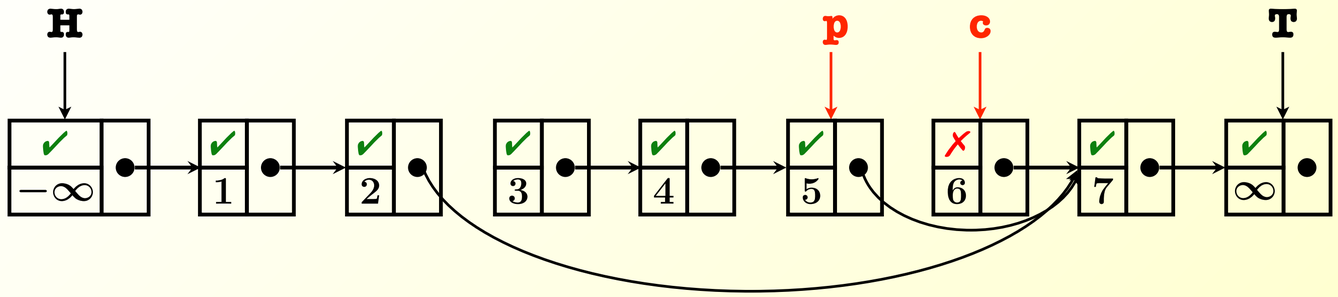


if no validation

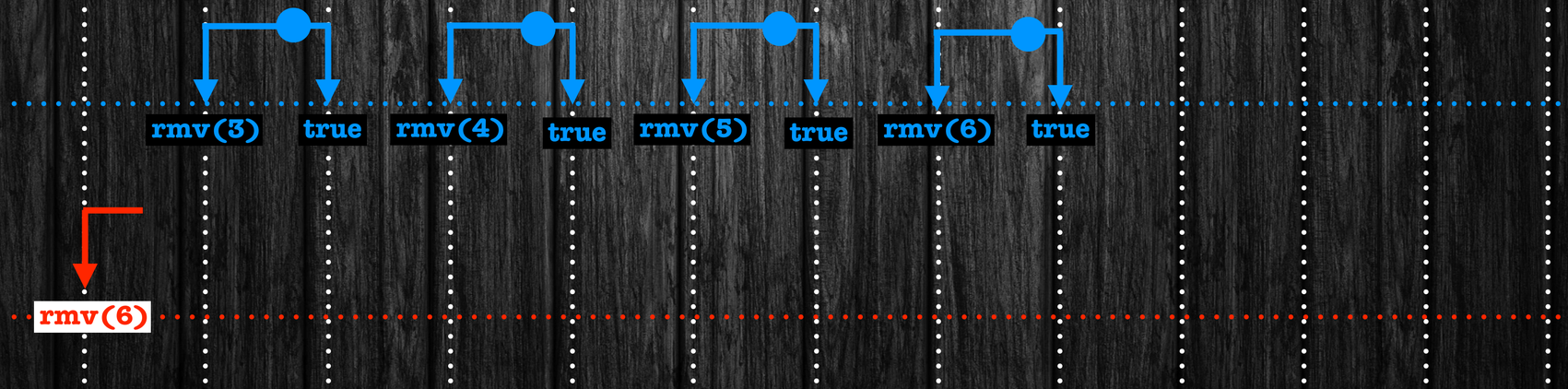


Sets | Optimistic Synchronization

Why Validation?

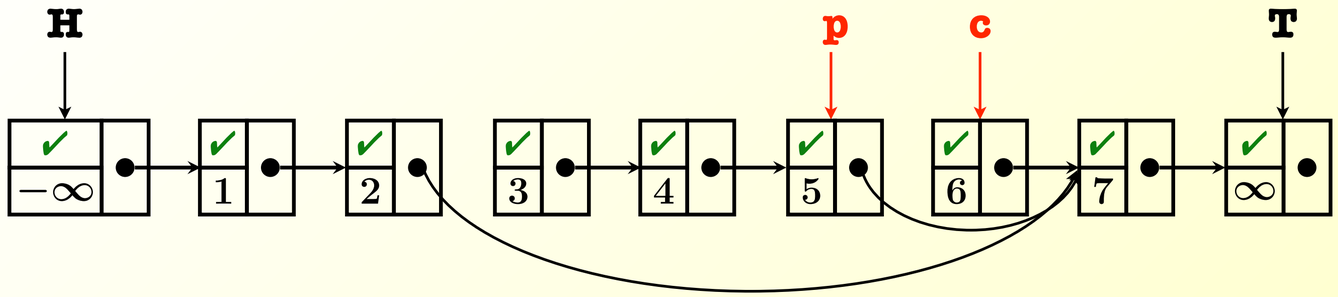


if no validation

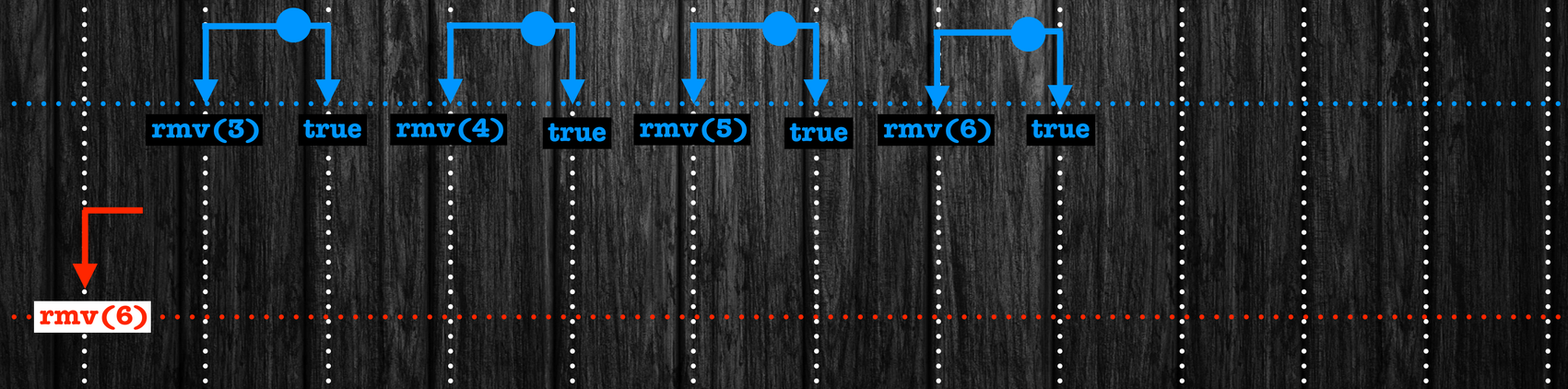


Sets Optimistic Synchronization

Why Validation?

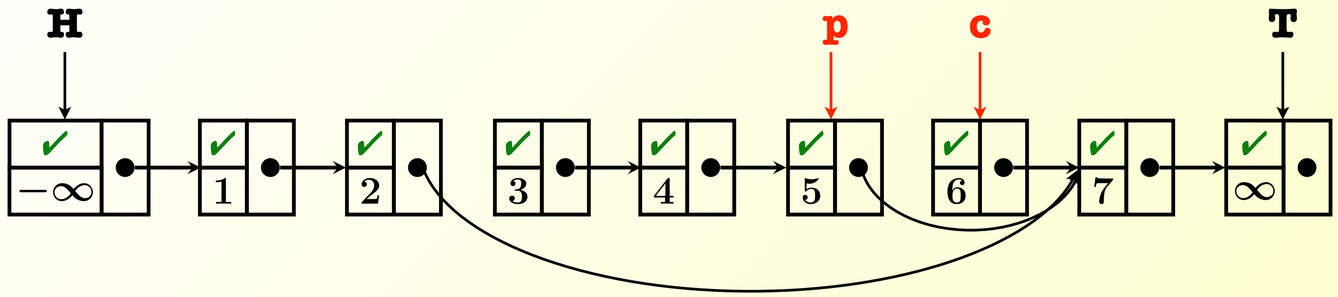


if no validation

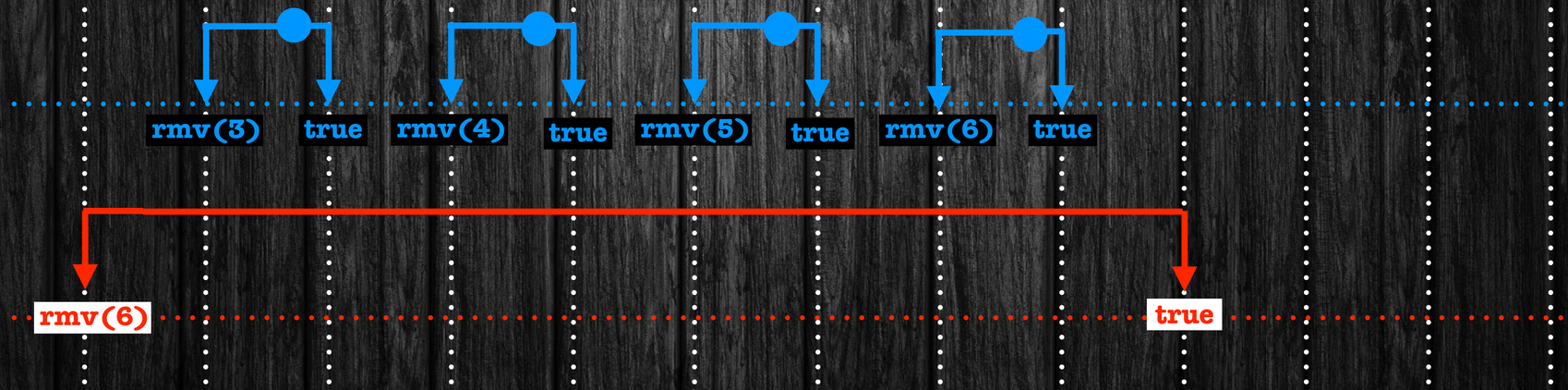


Sets Optimistic Synchronization

Why Validation?

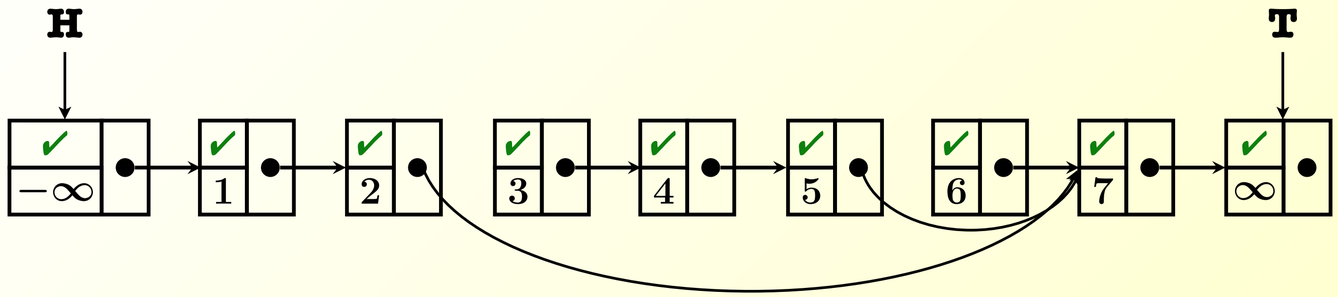


if no validation

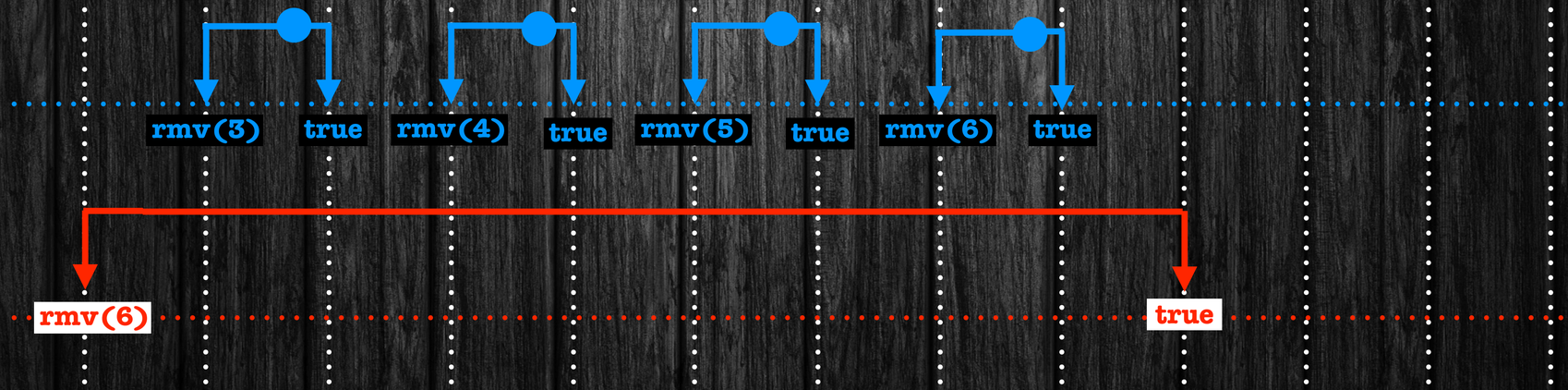


Sets Optimistic Synchronization

Why Validation?

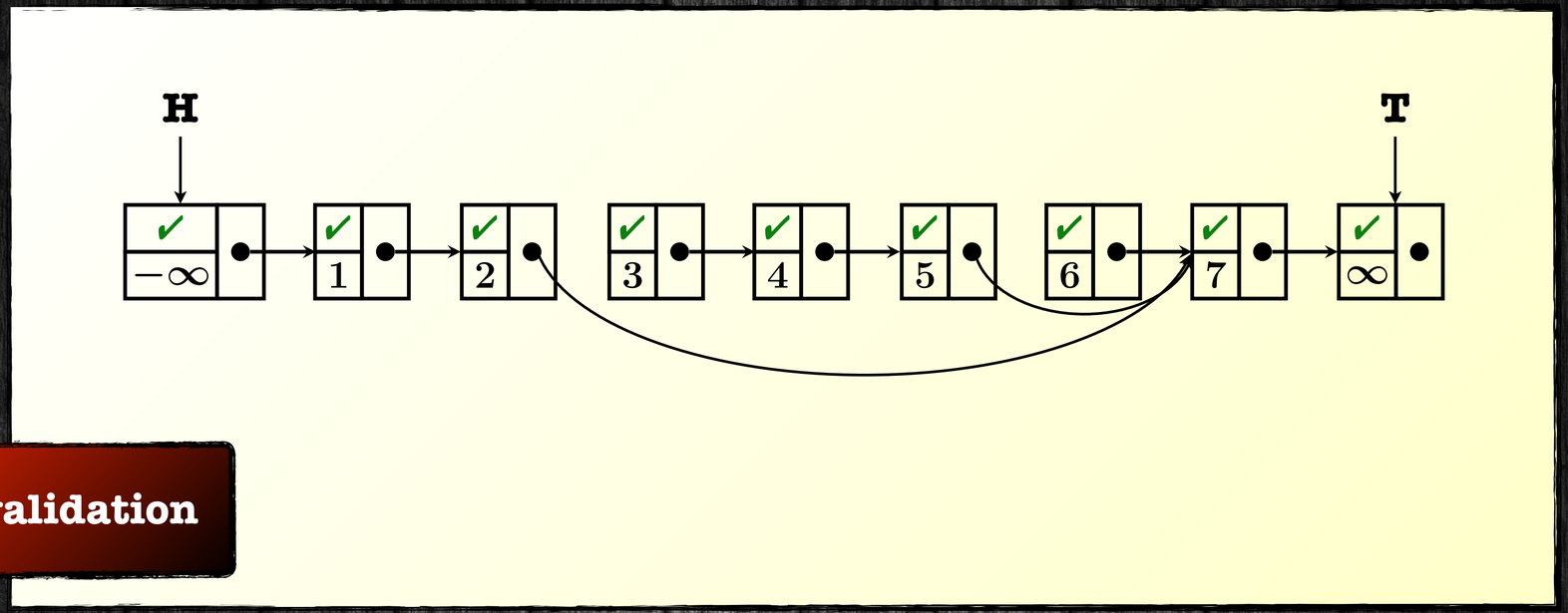


if no validation

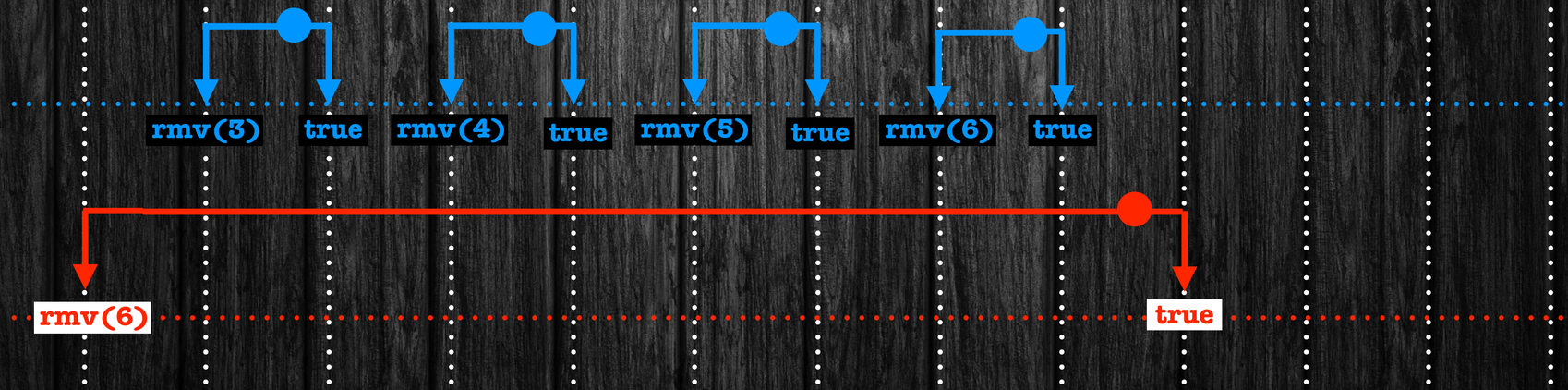


Sets | Optimistic Synchronization

Why Validation?

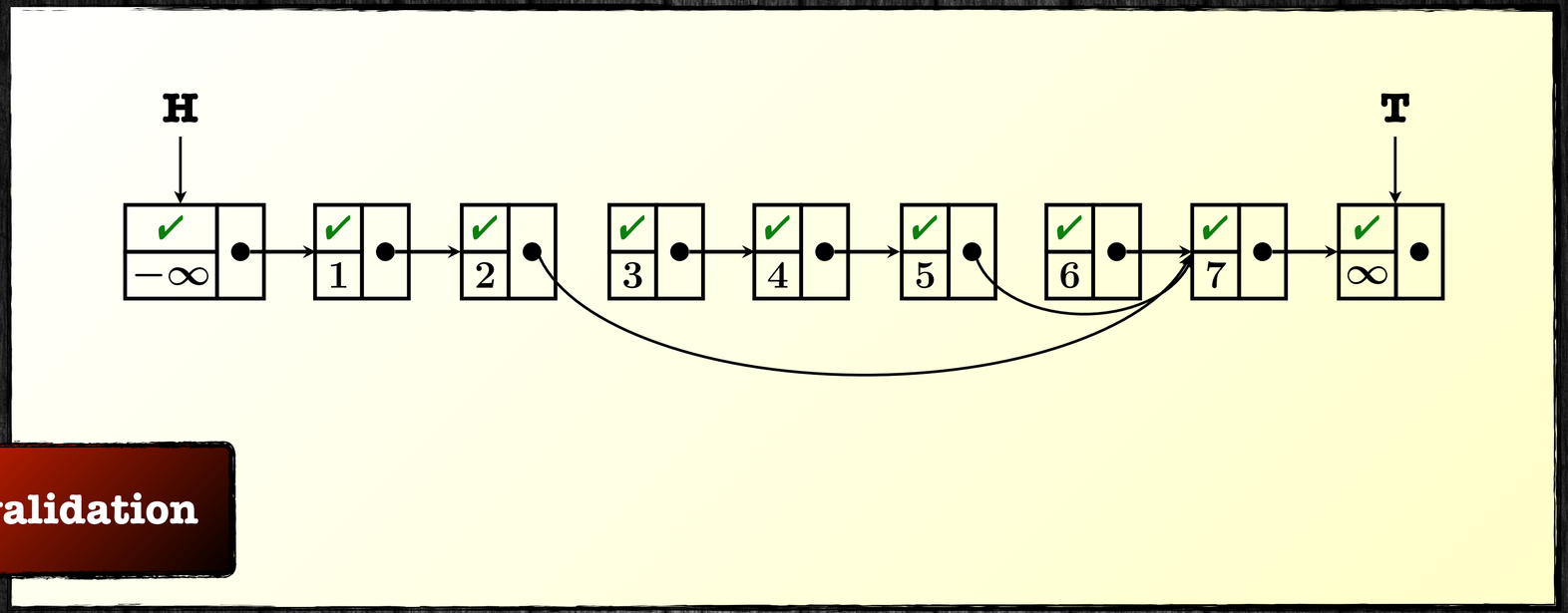


if no validation

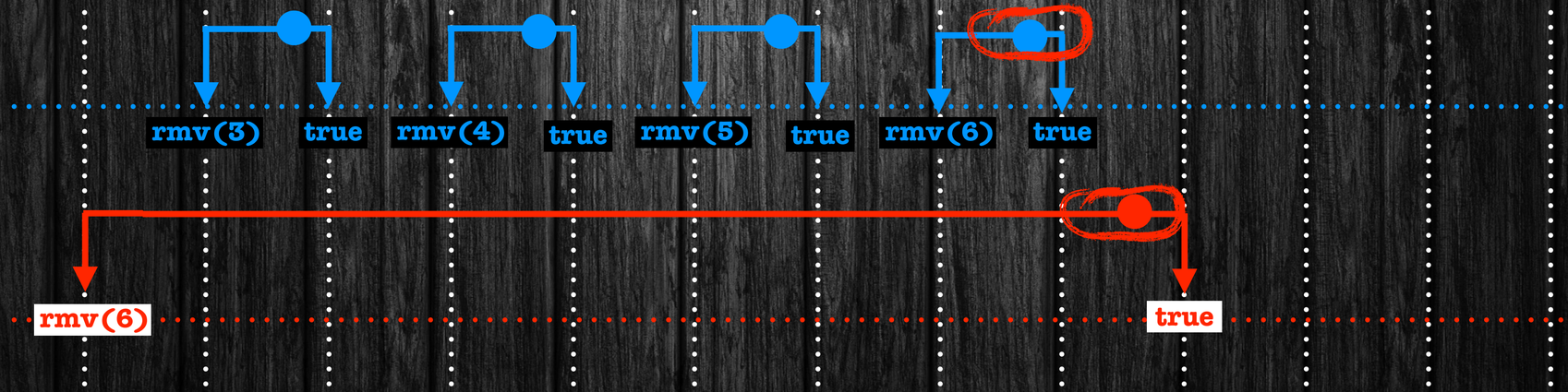


Sets | Optimistic Synchronization

Why Validation?

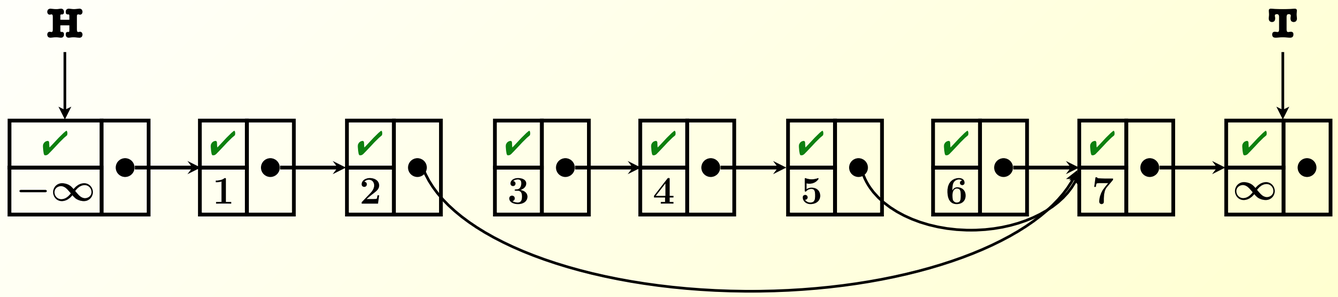


if no validation

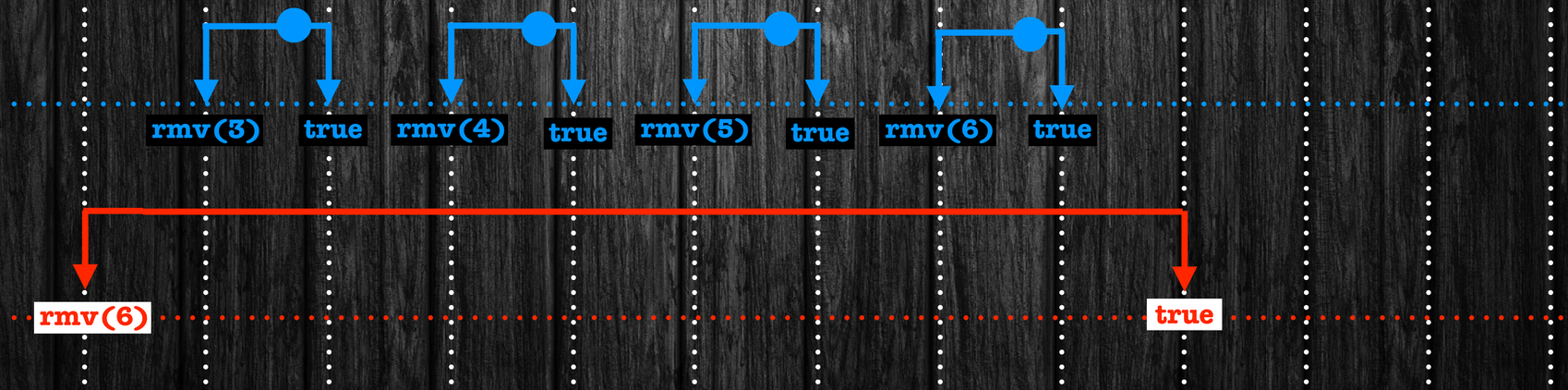


Sets Optimistic Synchronization

Why Validation?

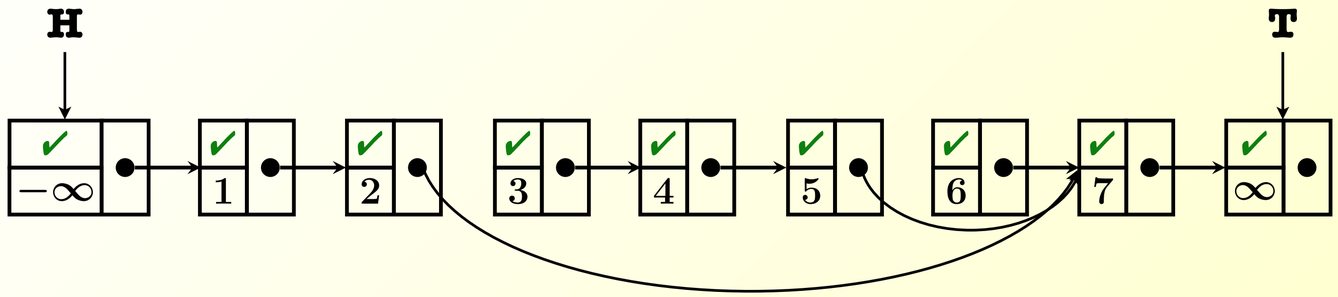


if no validation

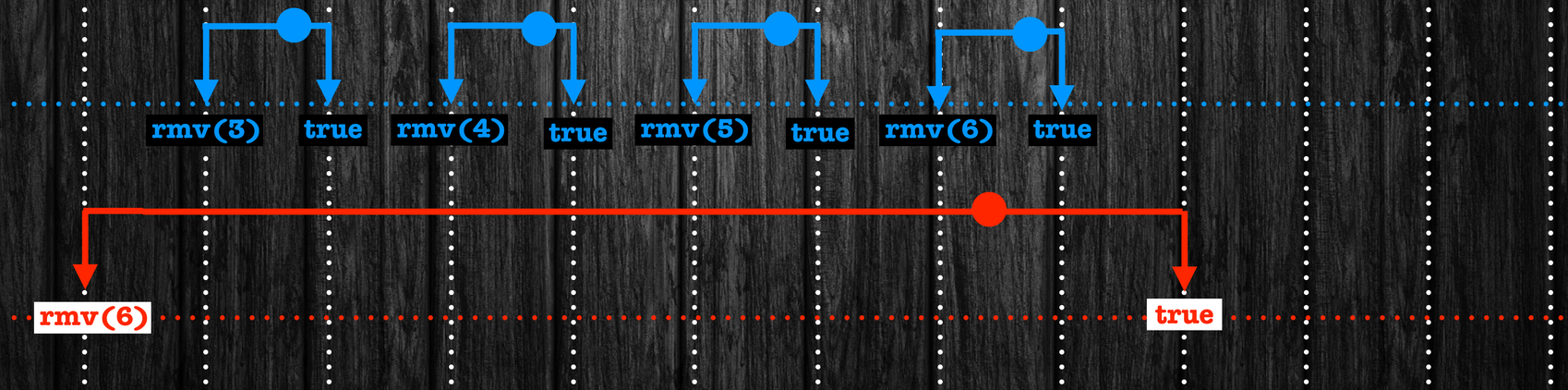


Sets | Optimistic Synchronization

Why Validation?

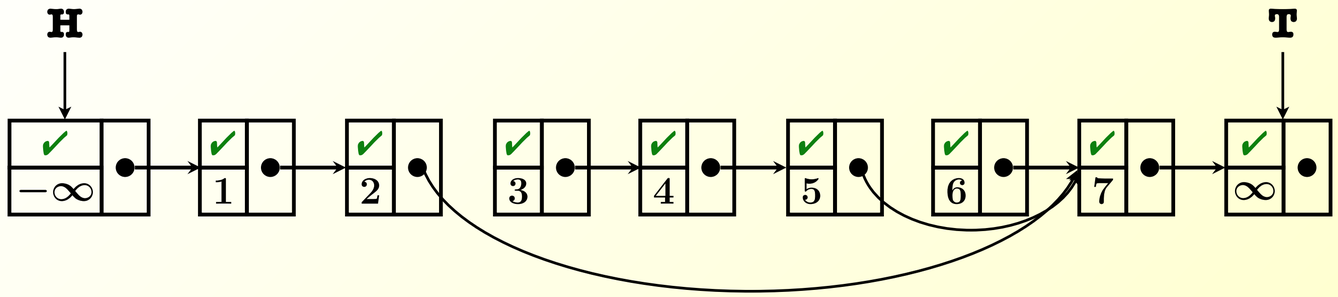


if no validation

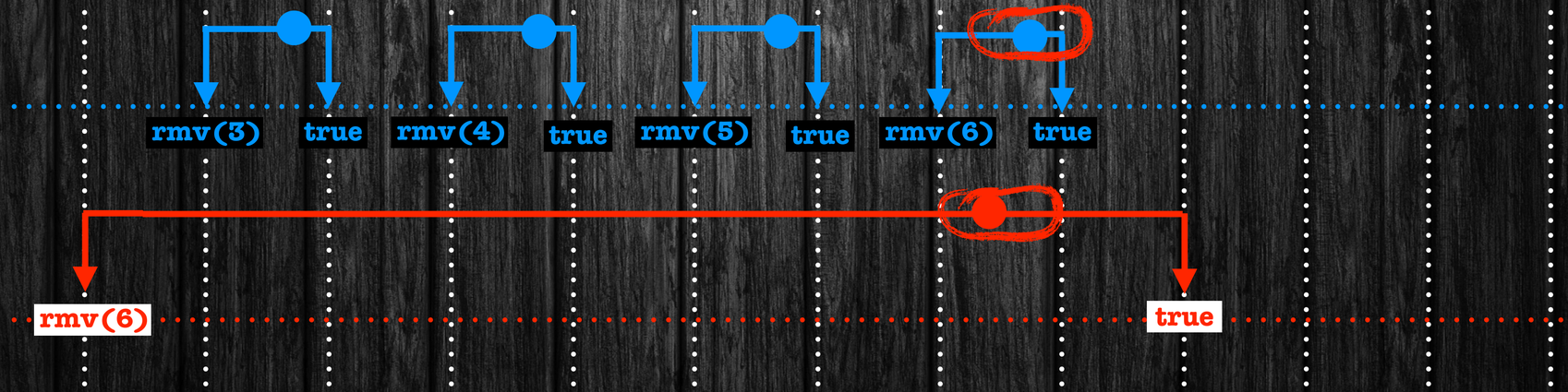


Sets | Optimistic Synchronization

Why Validation?

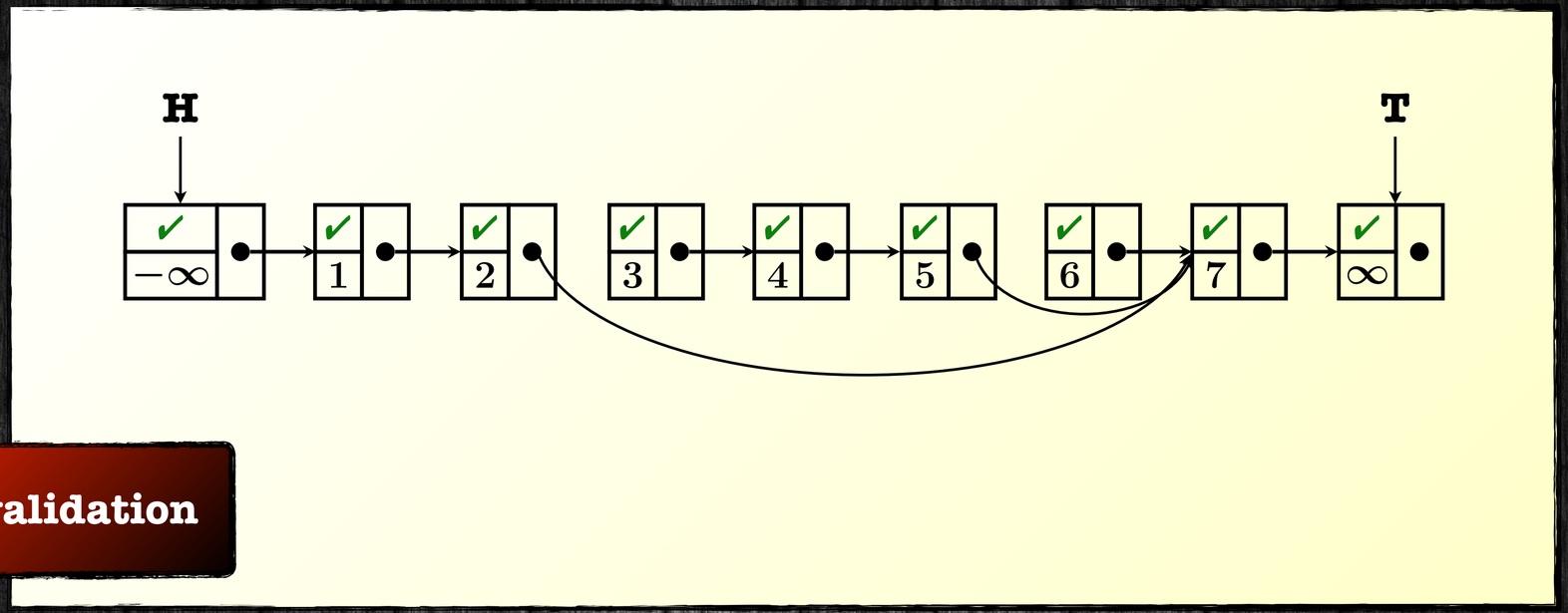


if no validation

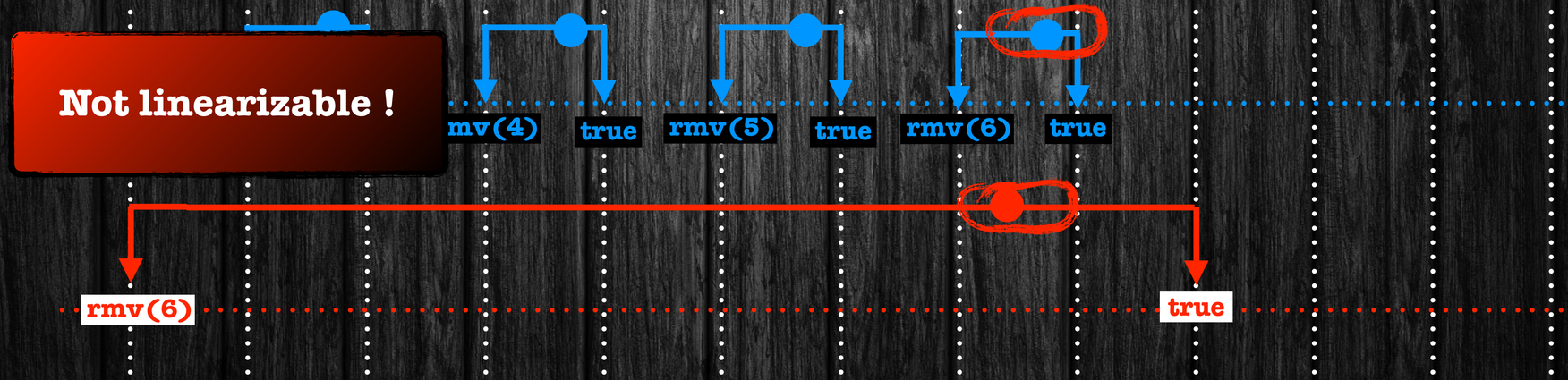


Sets | Optimistic Synchronization

Why Validation?

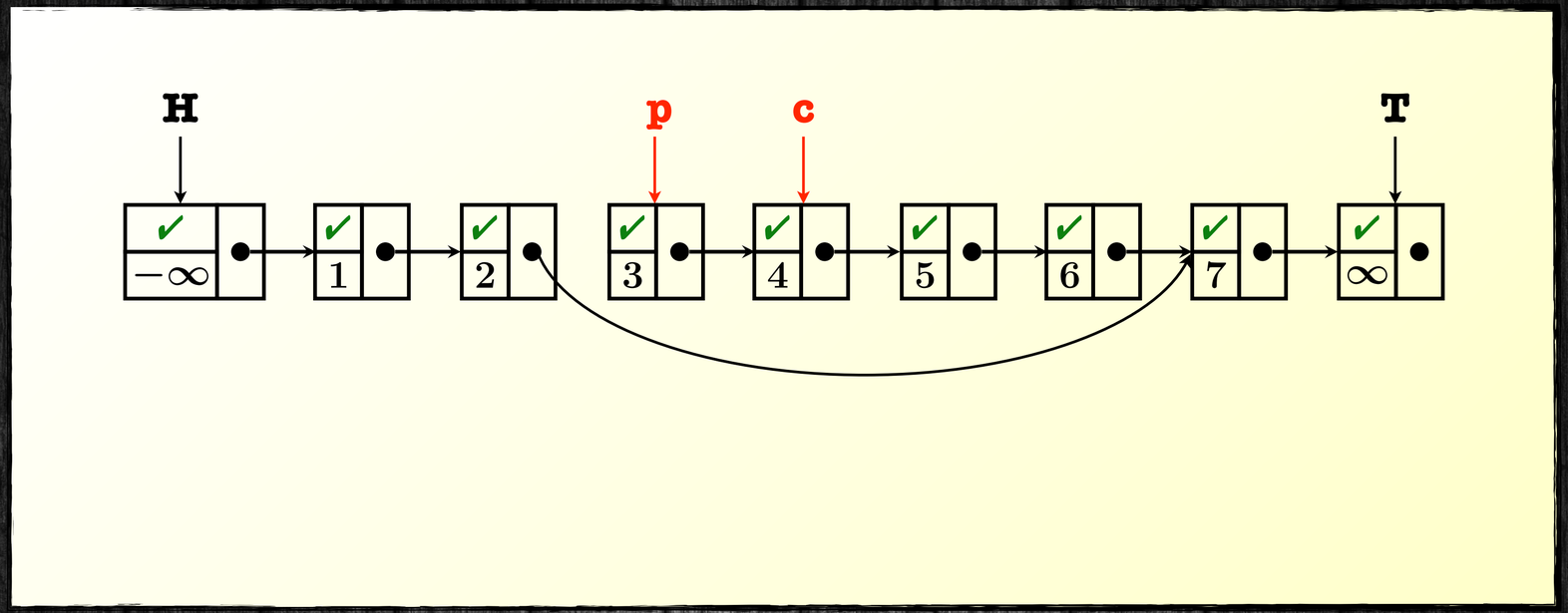


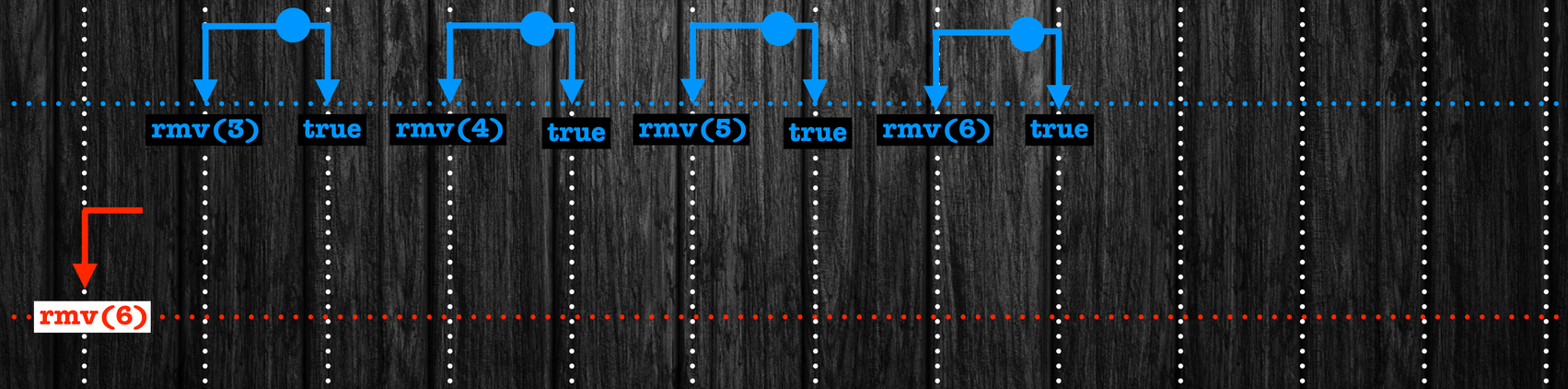
if no validation



Sets Optimistic Synchronization

Why Validation?

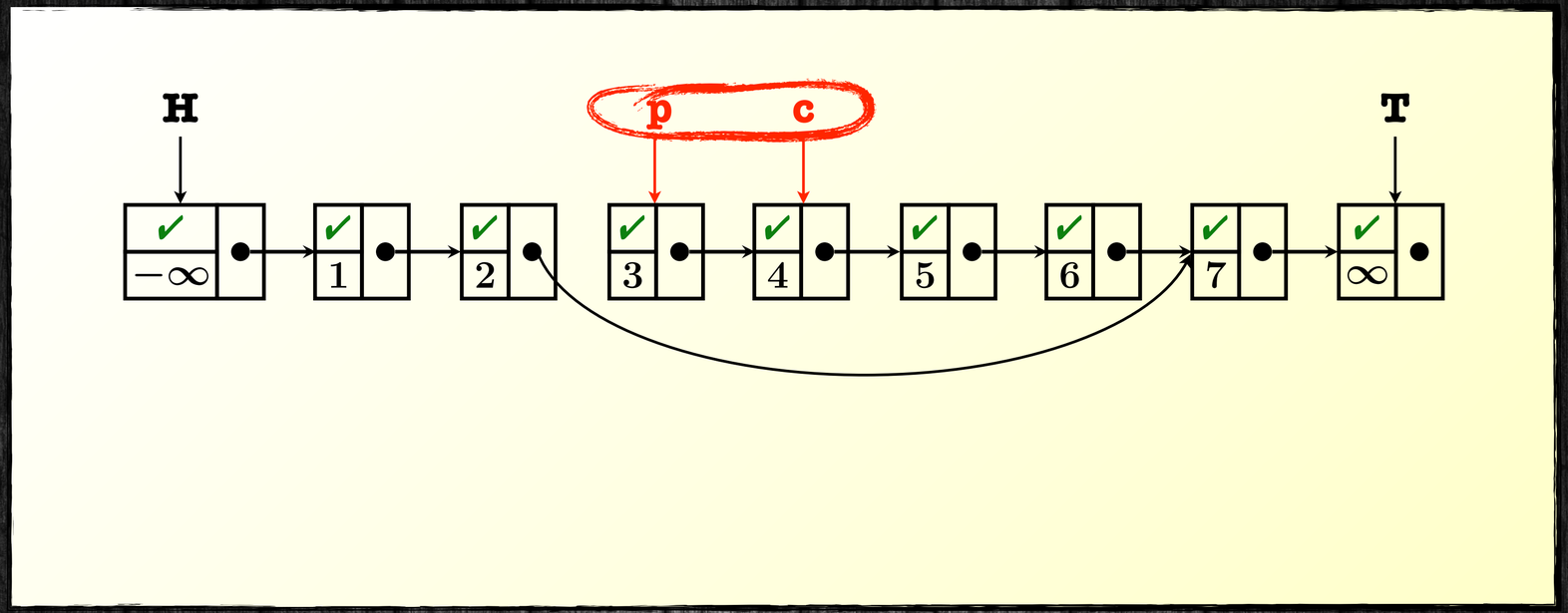


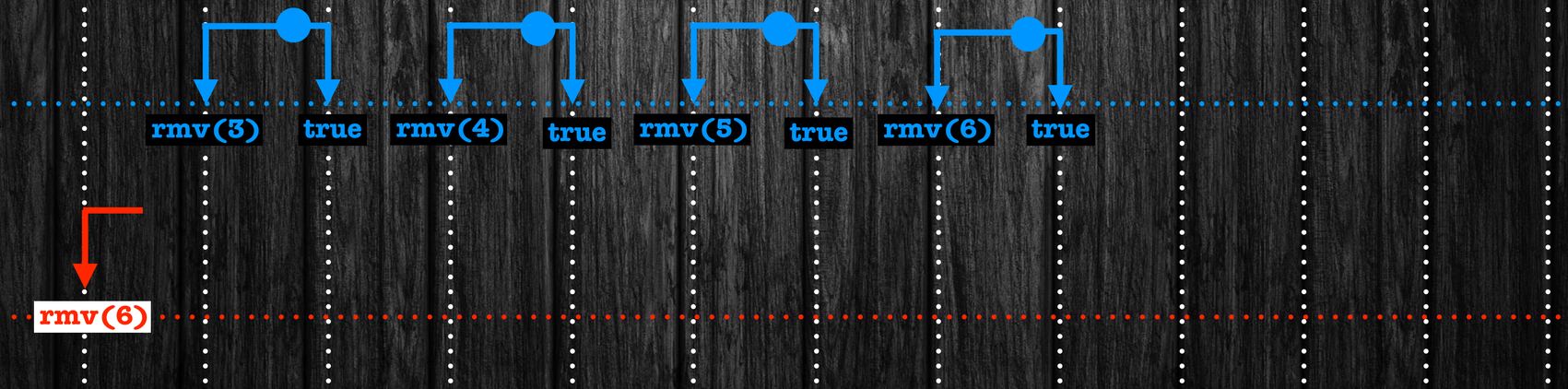


Sets

Optimistic Synchronization

Why Validation?

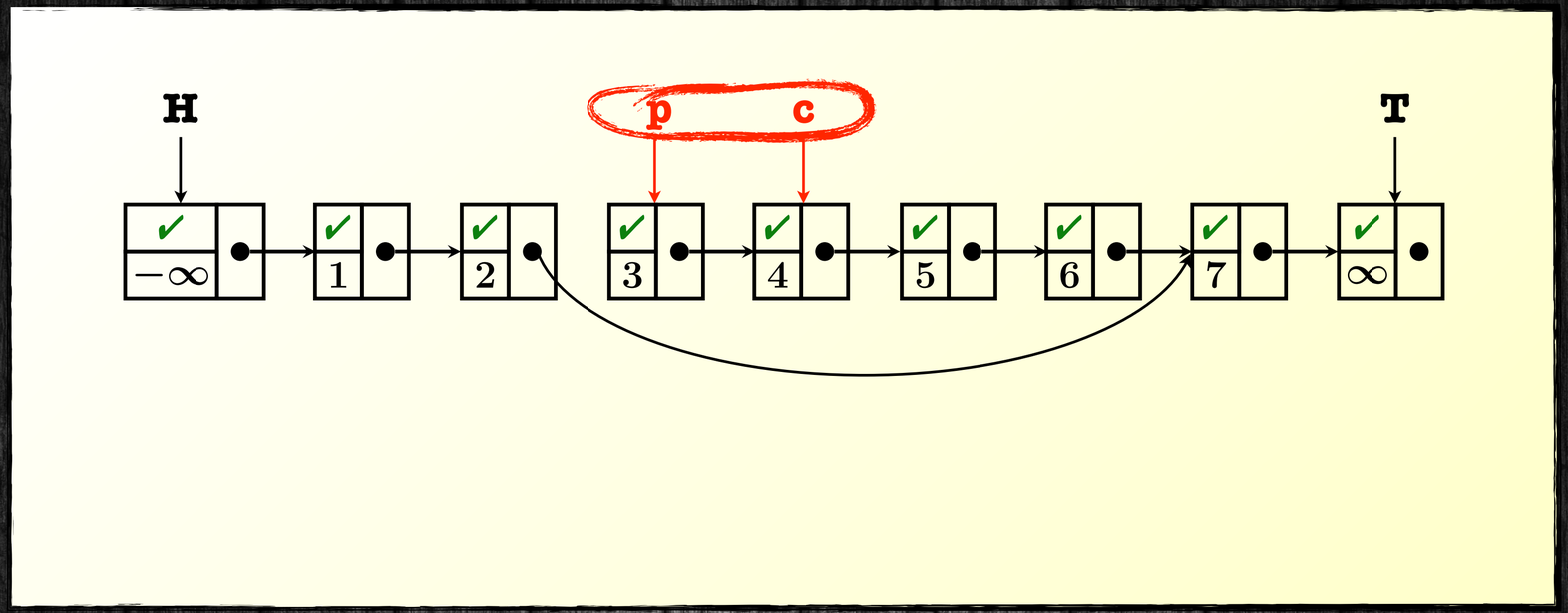


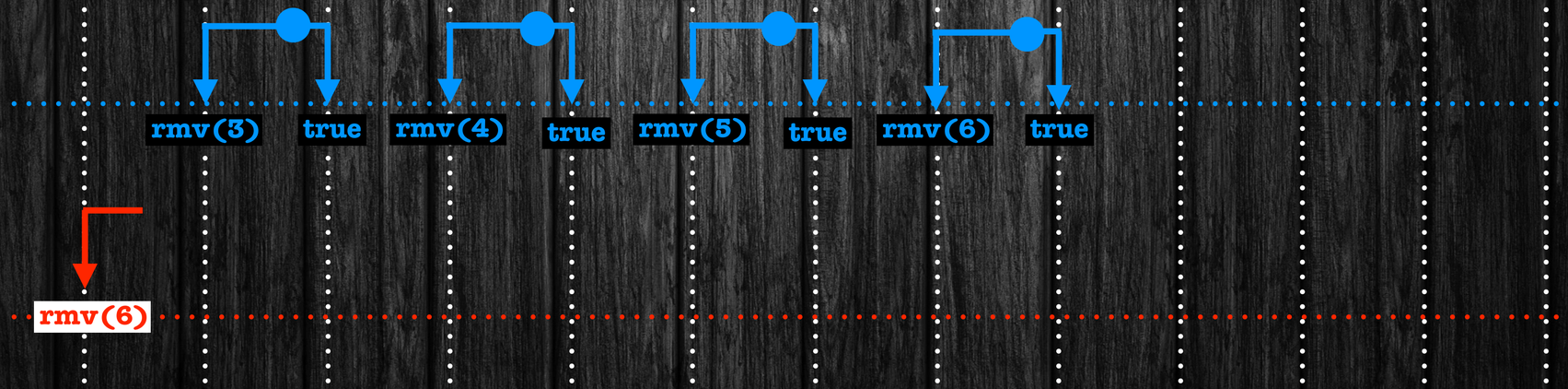


Sets | Optimistic Synchronization

Why Validation?

with validation

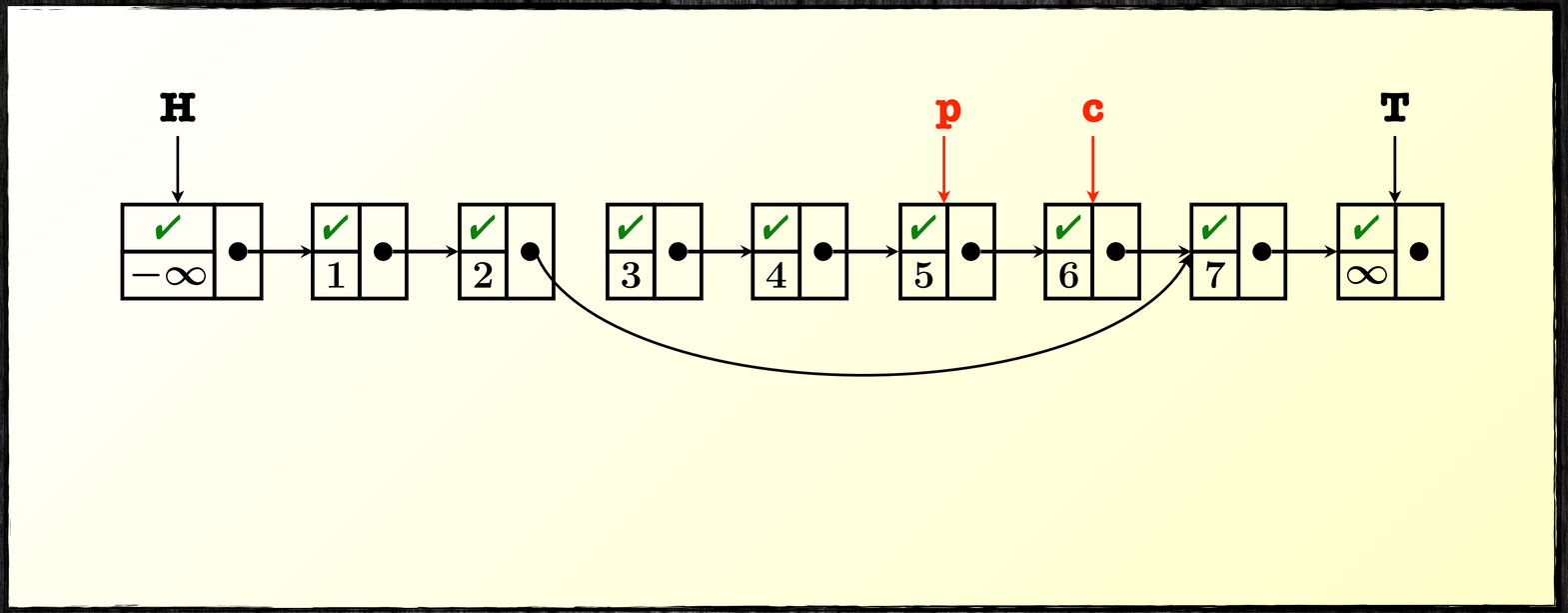


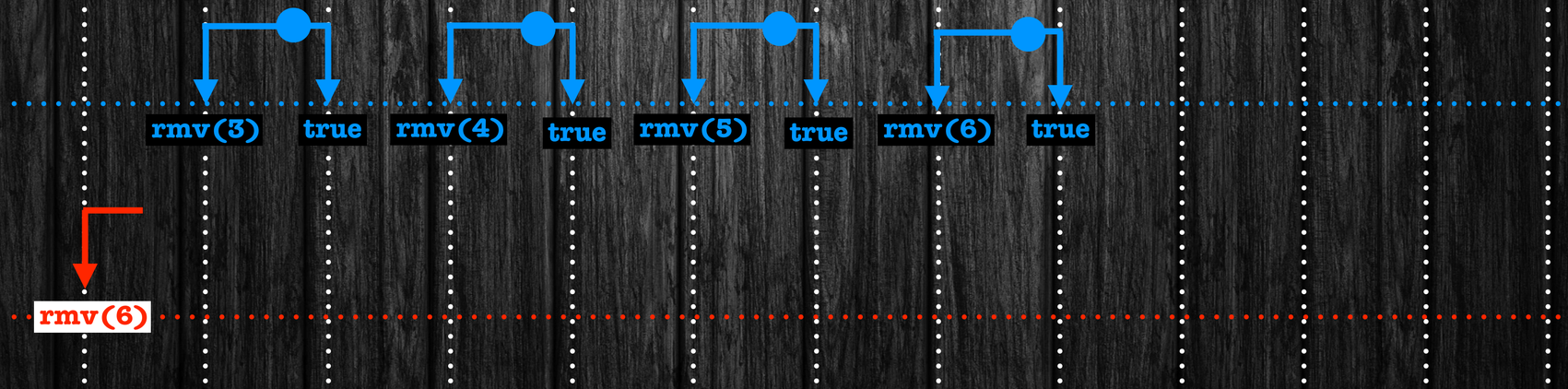


Sets Optimistic Synchronization

Why Validation?

with validation

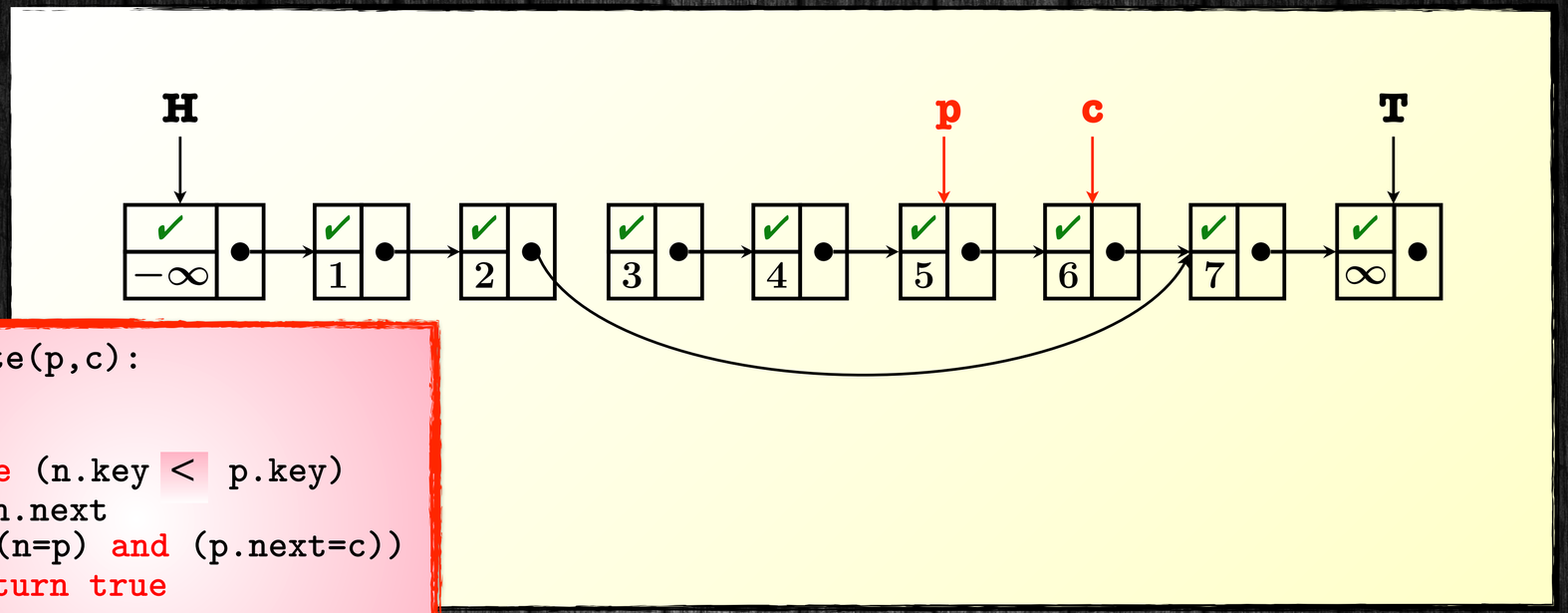




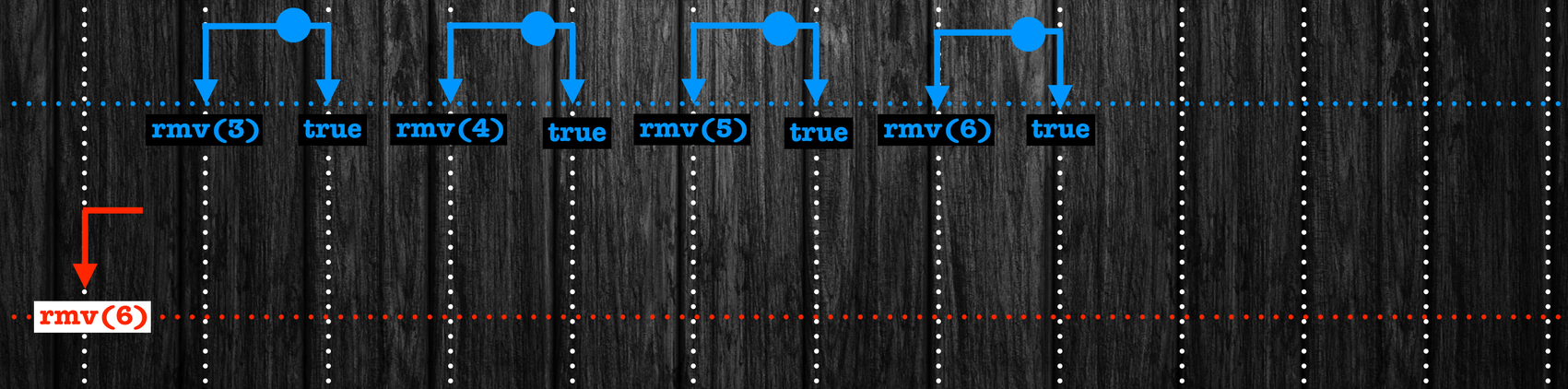
Sets | Optimistic Synchronization

Why Validation?

with validation



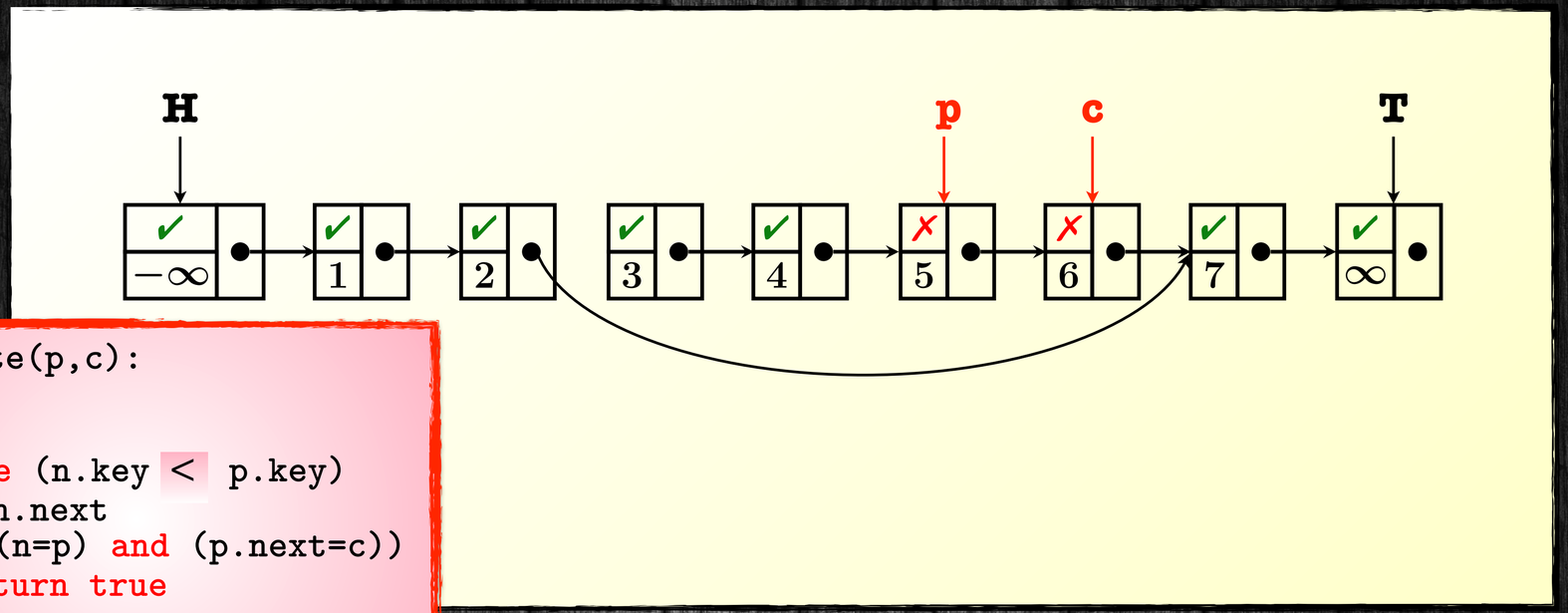

```
validate(p,c):
Node n
1  n=H
2  while (n.key < p.key)
3    n=n.next
4  if ((n=p) and (p.next=c))
5    return true
6  else
7    return false
```



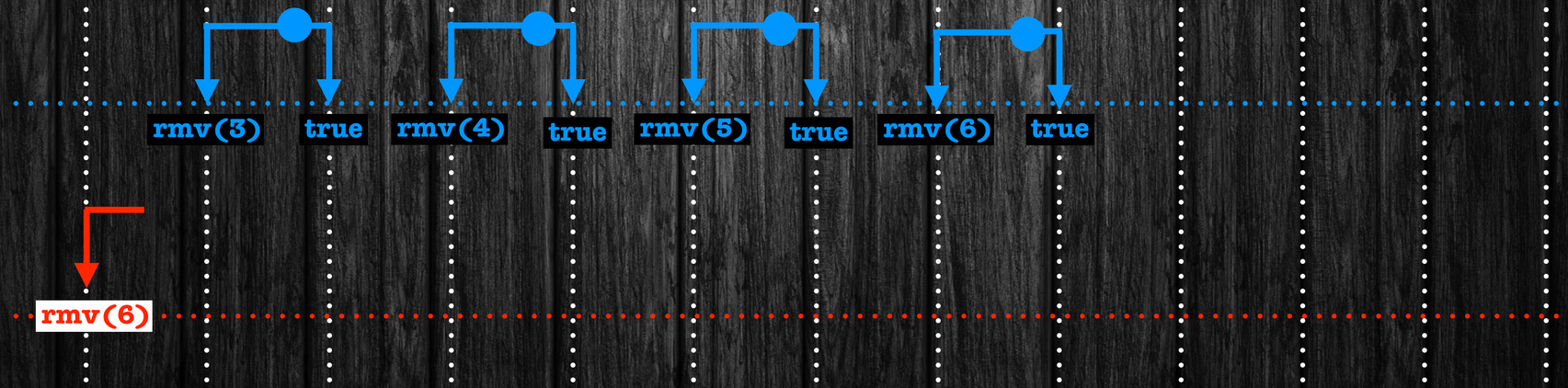
Sets Optimistic Synchronization

Why Validation?

with validation



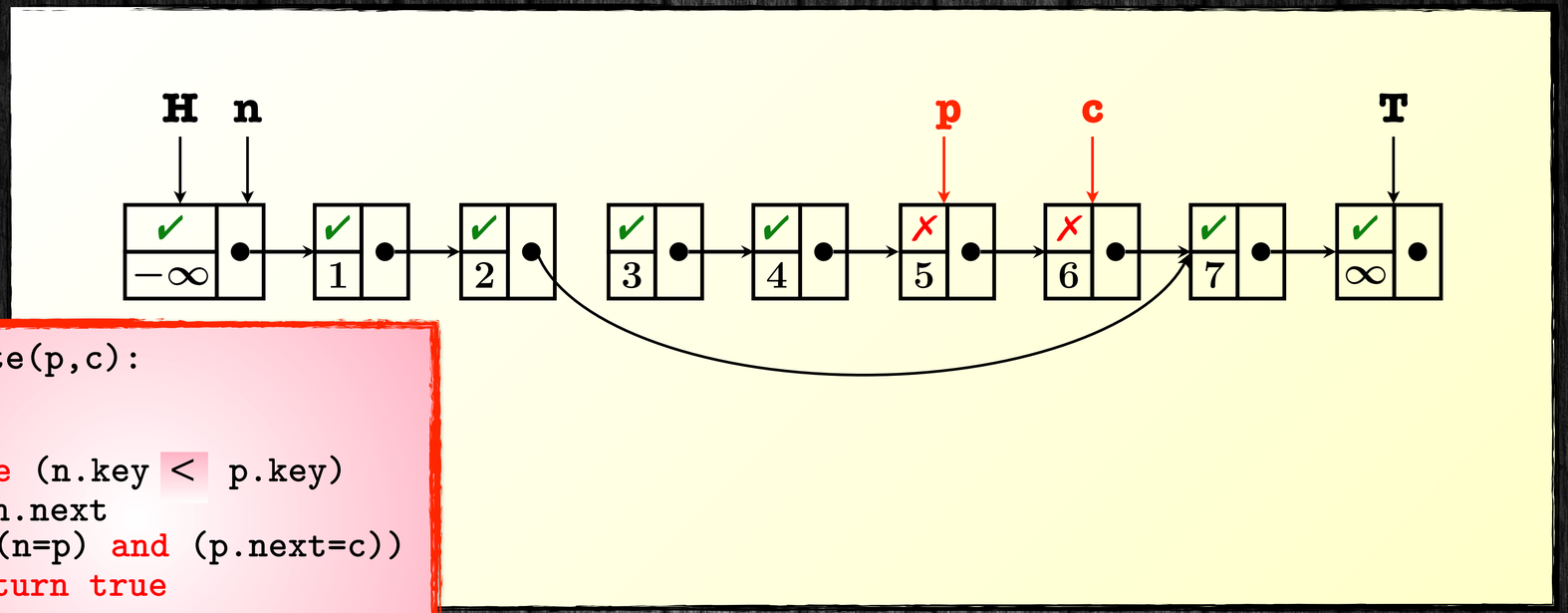

```
validate(p,c):
Node n
1  n=H
2  while (n.key < p.key)
3    n=n.next
4  if ((n=p) and (p.next=c))
5    return true
6  else
7    return false
```



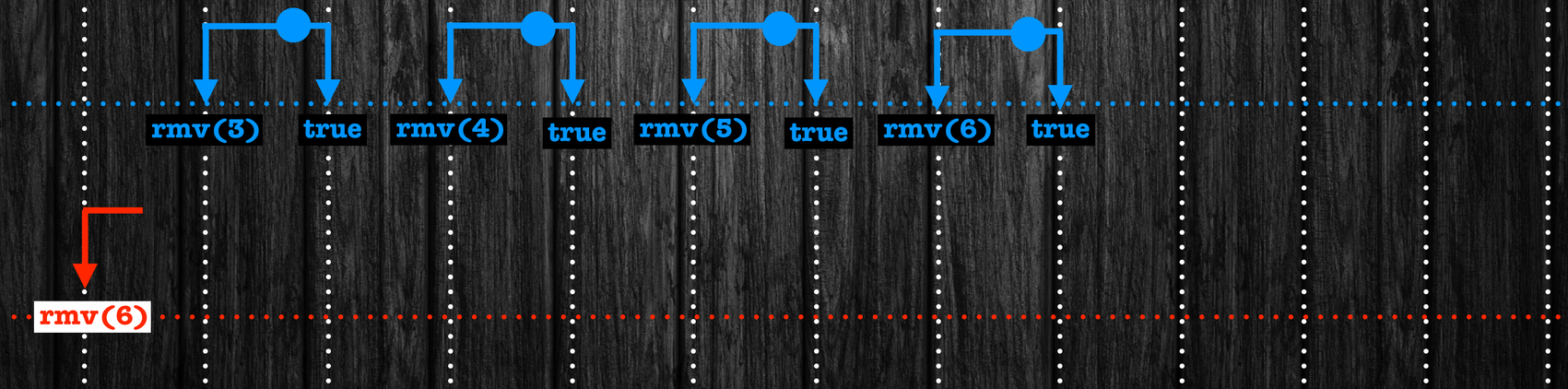
Sets Optimistic Synchronization

Why Validation?

with validation



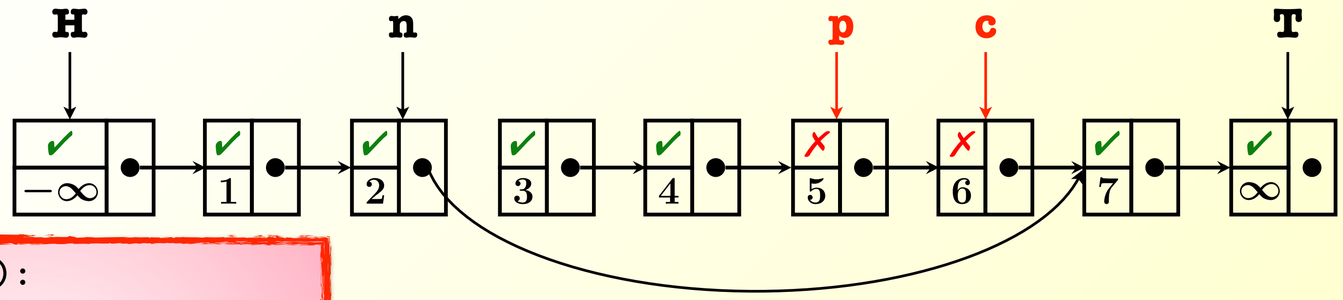

```
validate(p,c):
Node n
1  n=H
2  while (n.key < p.key)
3    n=n.next
4  if ((n=p) and (p.next=c))
5    return true
6  else
7    return false
```



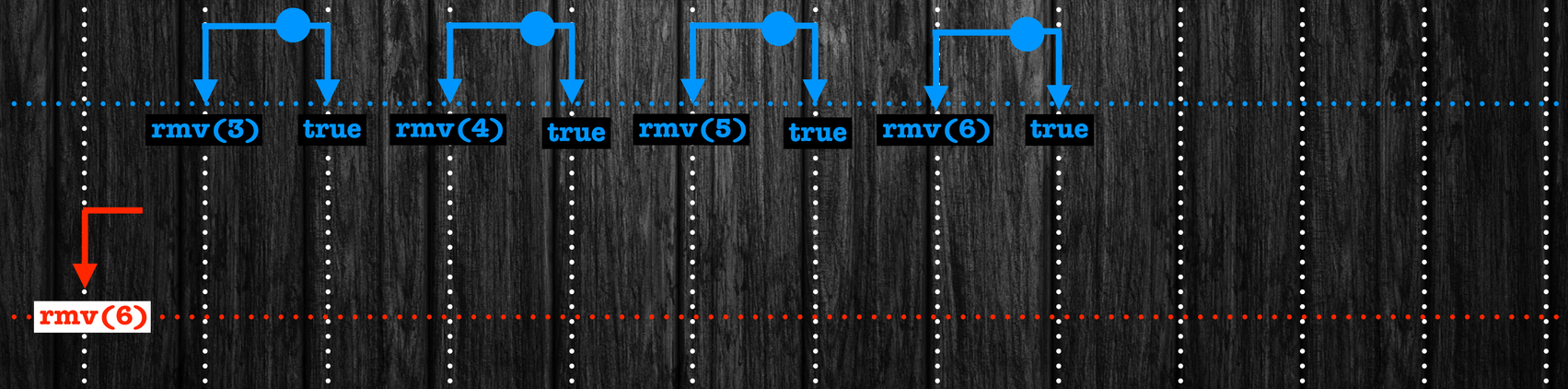
Sets Optimistic Synchronization

Why Validation?

with validation



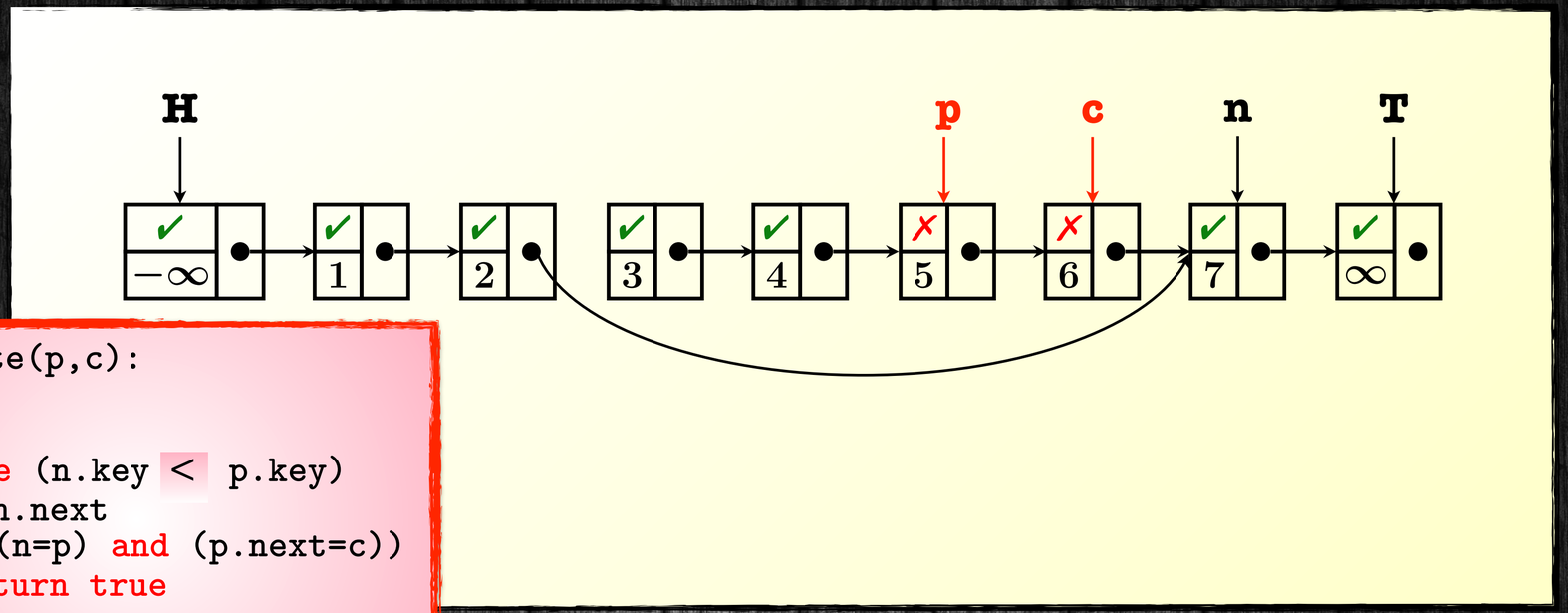

```
validate(p,c):
Node n
1  n=H
2  while (n.key < p.key)
3    n=n.next
4  if ((n=p) and (p.next=c))
5    return true
6  else
7    return false
```



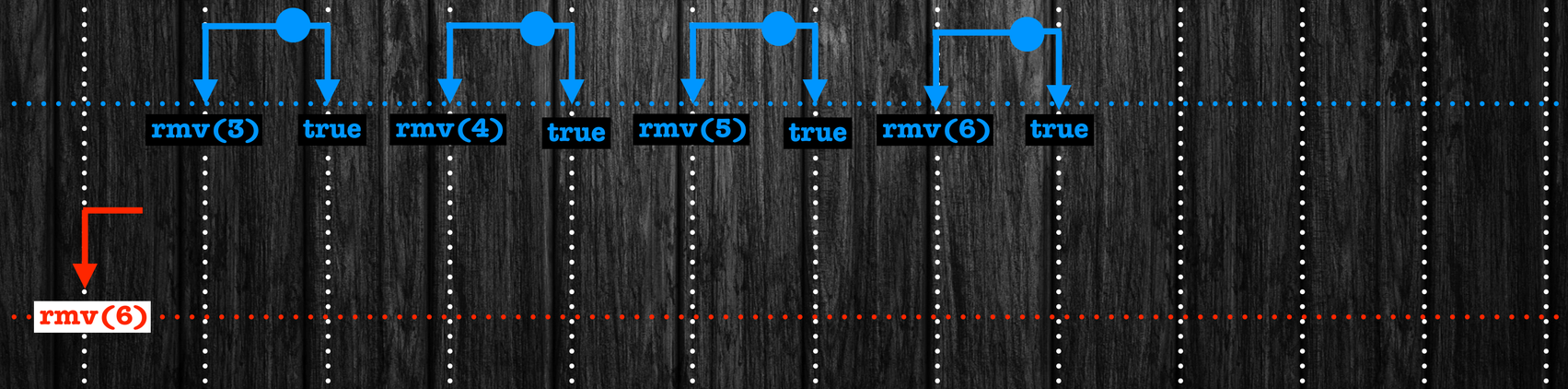
Sets Optimistic Synchronization

Why Validation?

with validation



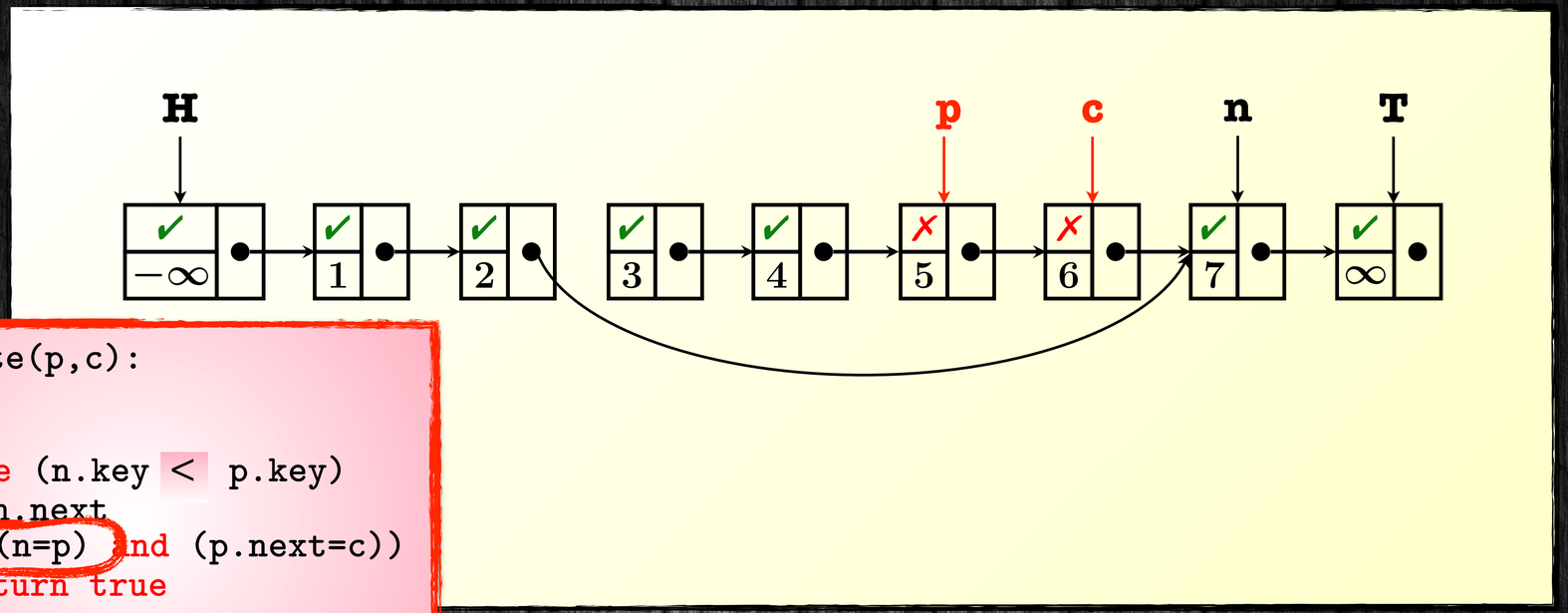

```
validate(p,c):
Node n
1  n=H
2  while (n.key < p.key)
3    n=n.next
4  if ((n=p) and (p.next=c))
5    return true
6  else
7    return false
```



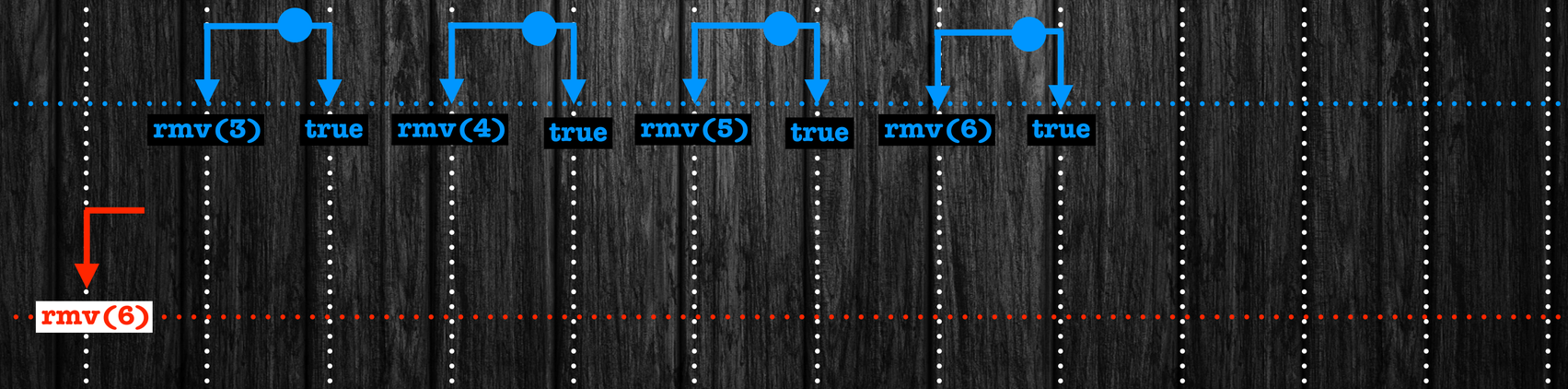
Sets Optimistic Synchronization

Why Validation?

with validation



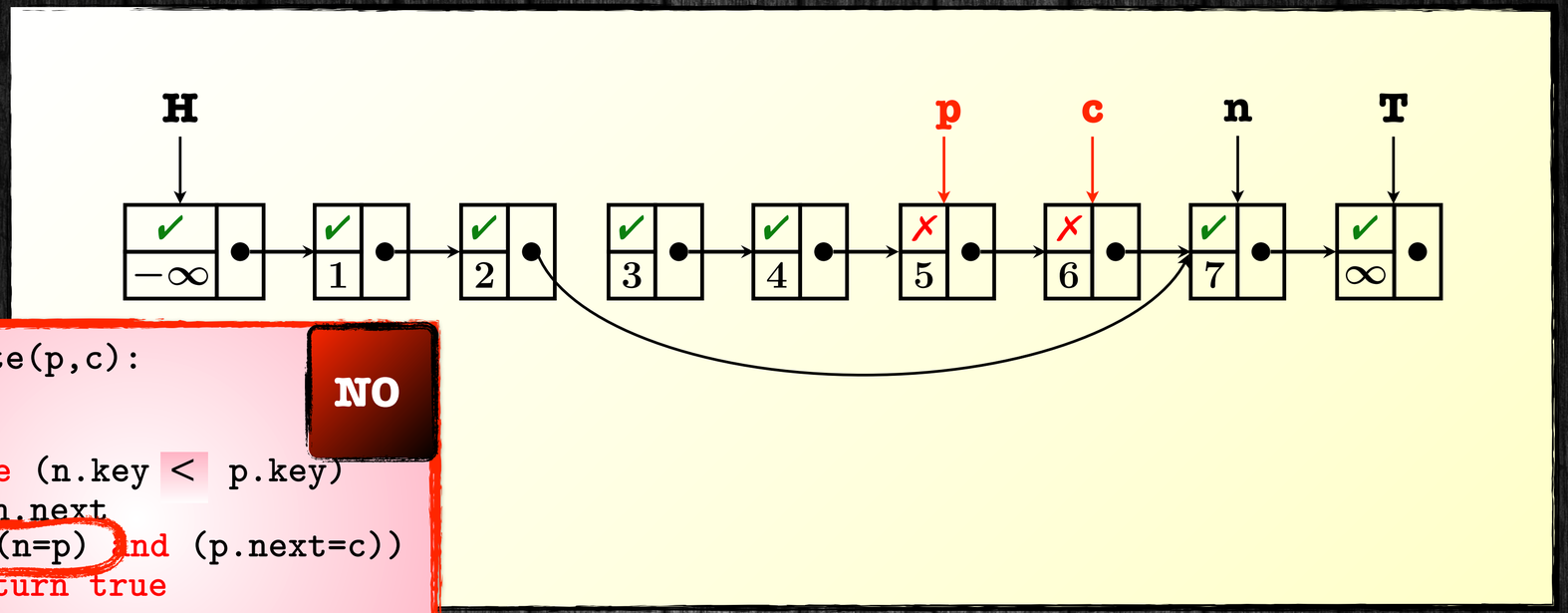

```
validate(p,c):
Node n
1  n=H
2  while (n.key < p.key)
3    n=n.next
4  if ((n=p) and (p.next=c))
5    return true
6  else
7    return false
```



Sets Optimistic Synchronization

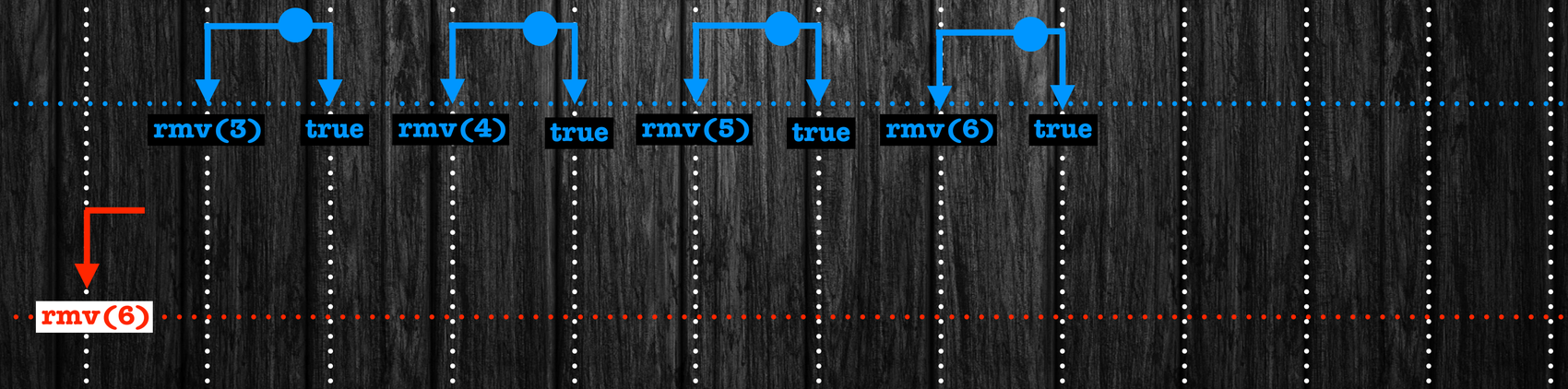
Why Validation?

with validation



NO

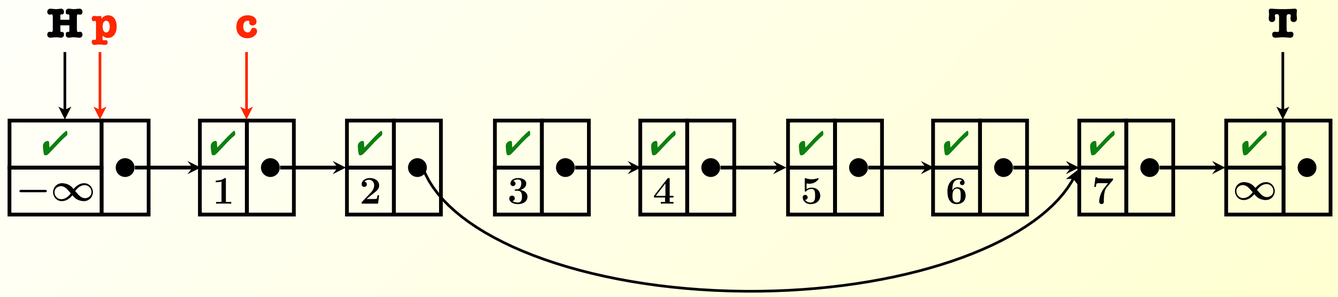
```
validate(p,c):
Node n
1  n=H
2  while (n.key < p.key)
3    n=n.next
4  if ((n=p) and (p.next=c))
5    return true
6  else
7    return false
```

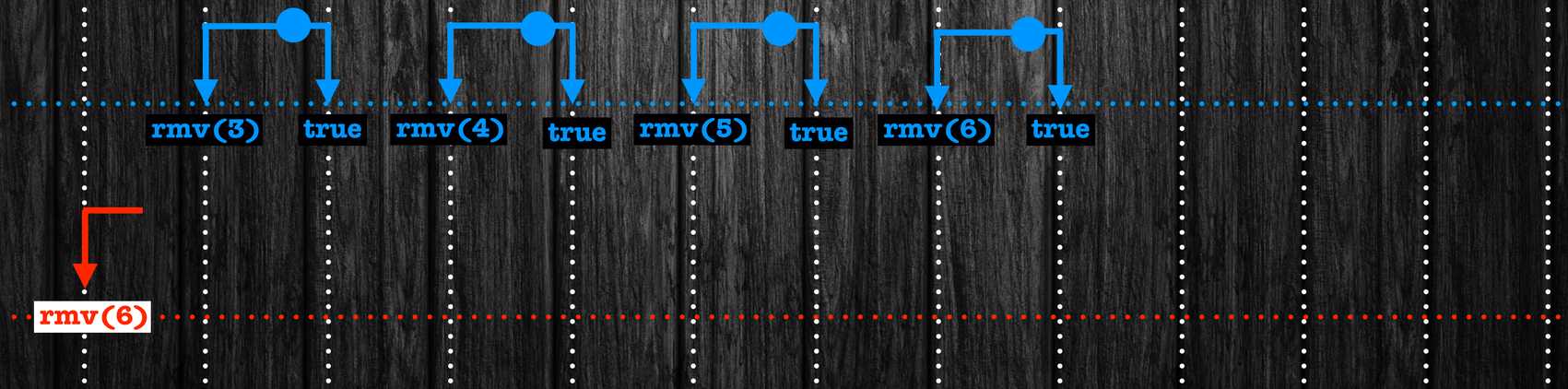


Sets | Optimistic Synchronization

Why Validation?

with validation

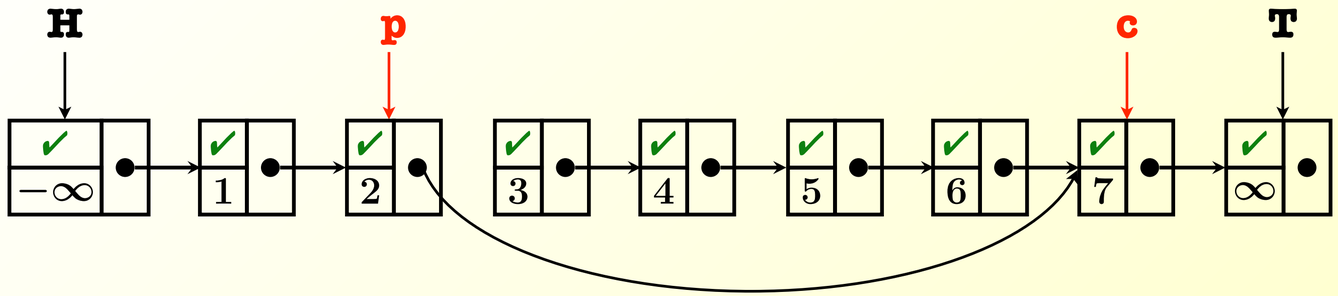


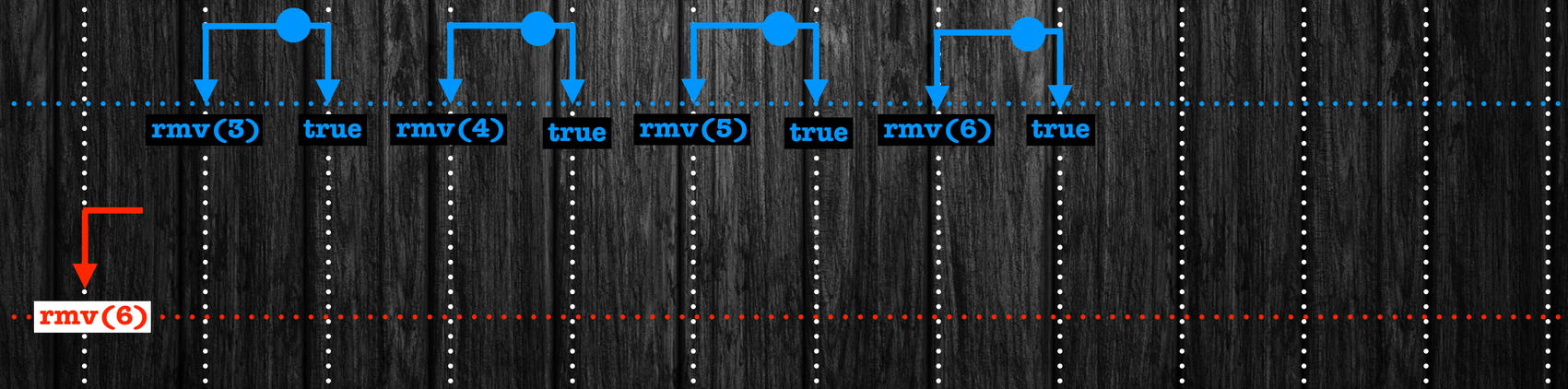


Sets | Optimistic Synchronization

Why Validation?

with validation

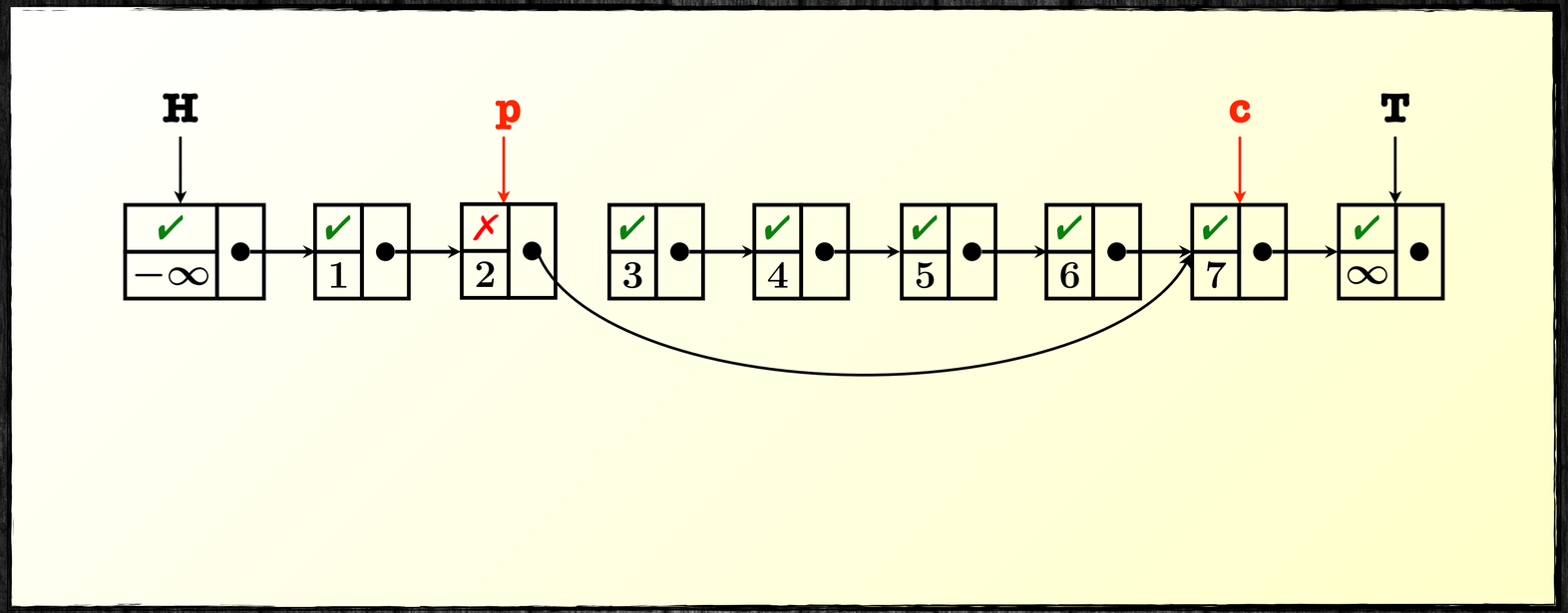


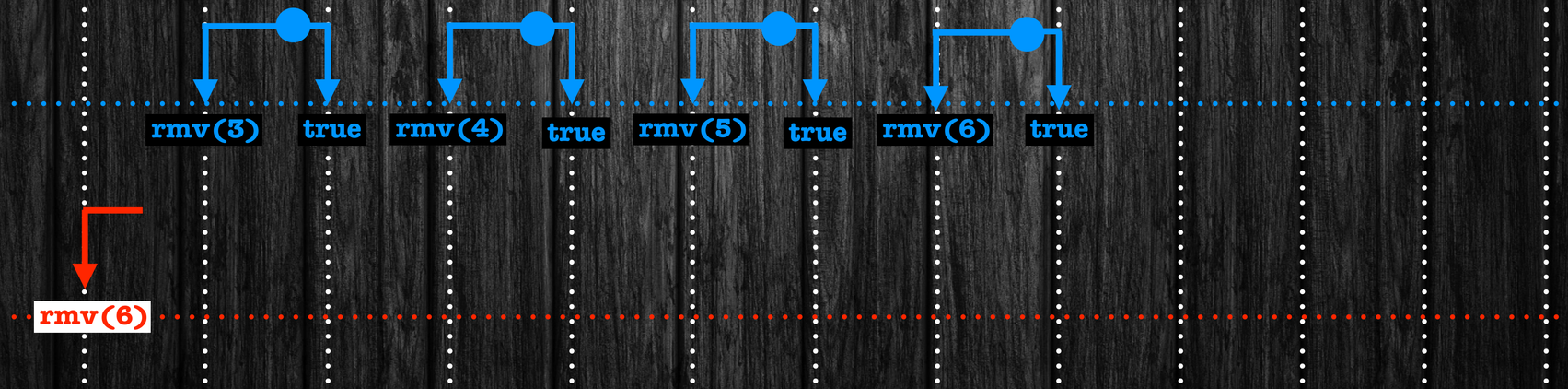


Sets | Optimistic Synchronization

Why Validation?

with validation

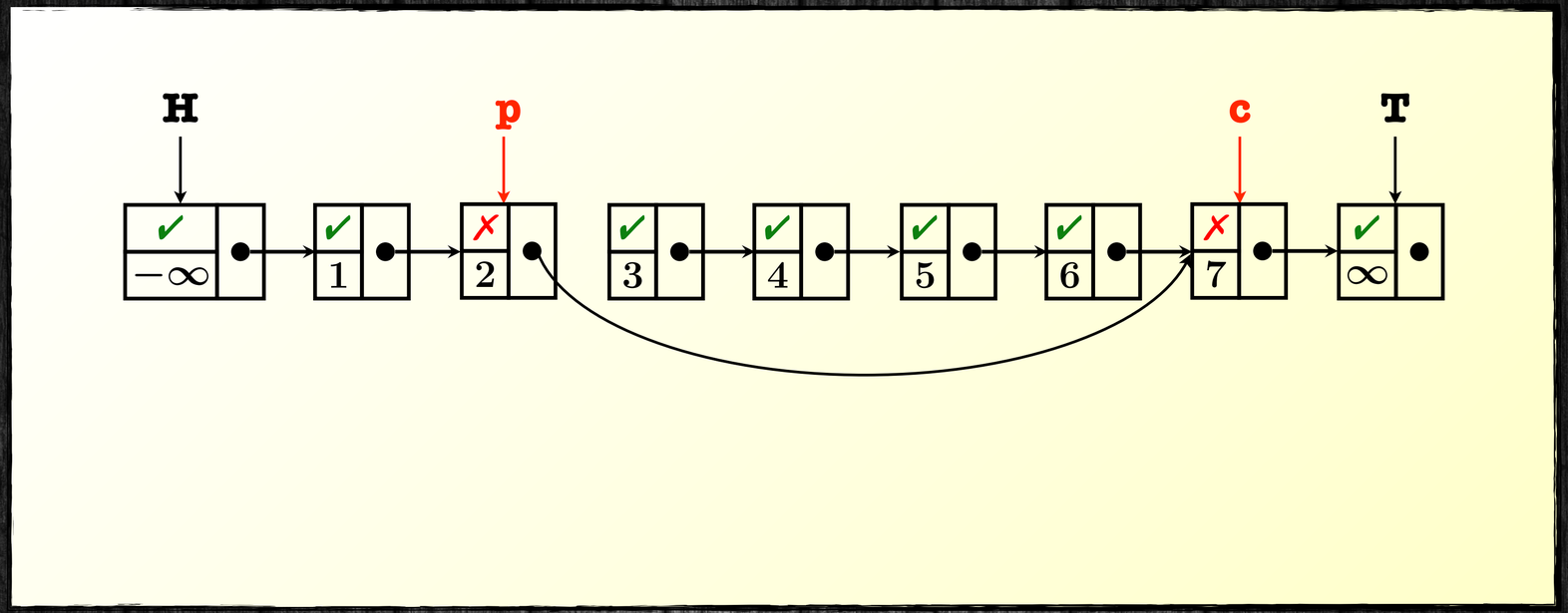


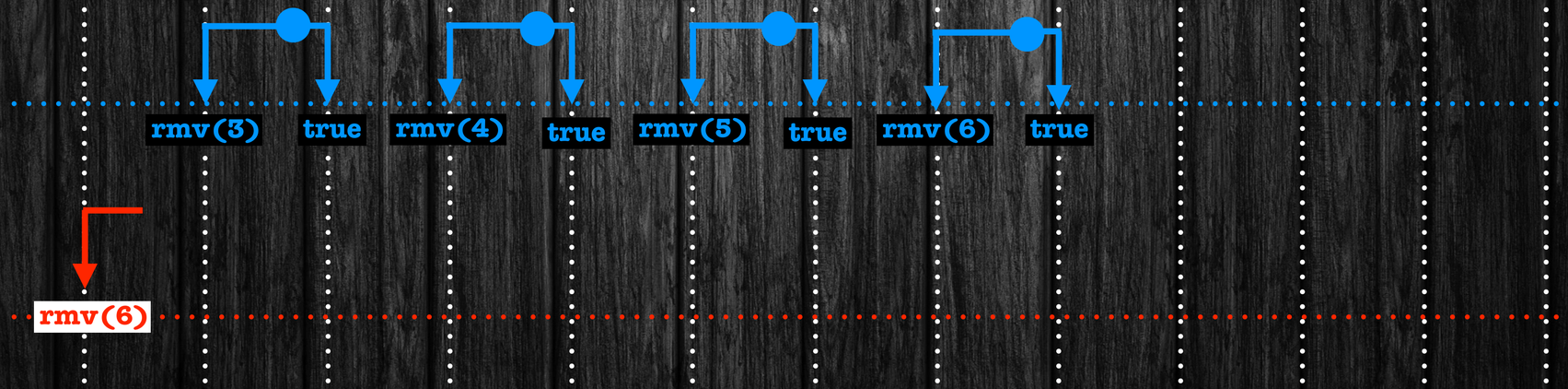


Sets | Optimistic Synchronization

Why Validation?

with validation

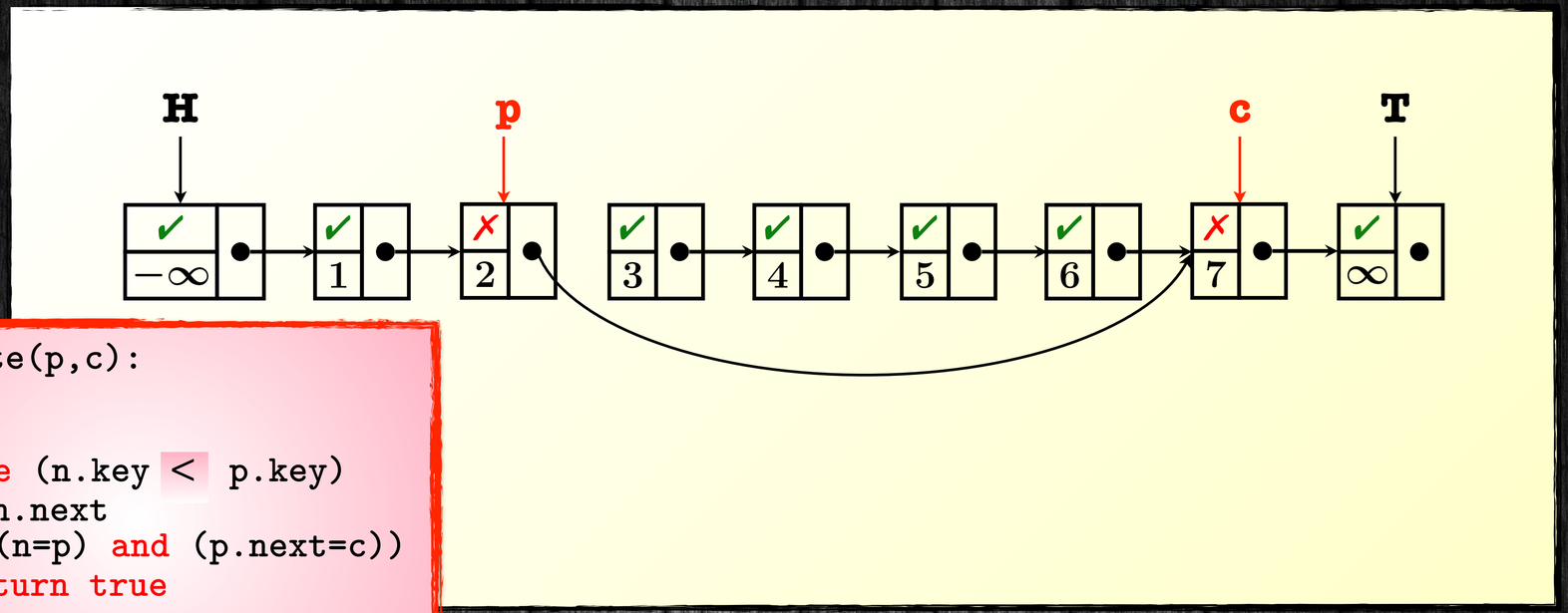




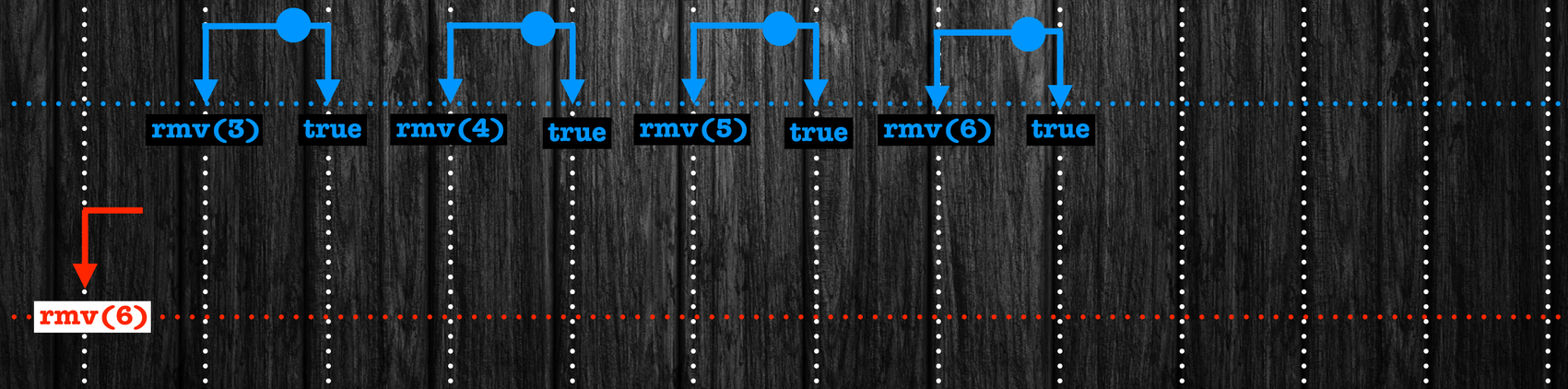
Sets Optimistic Synchronization

Why Validation?

with validation



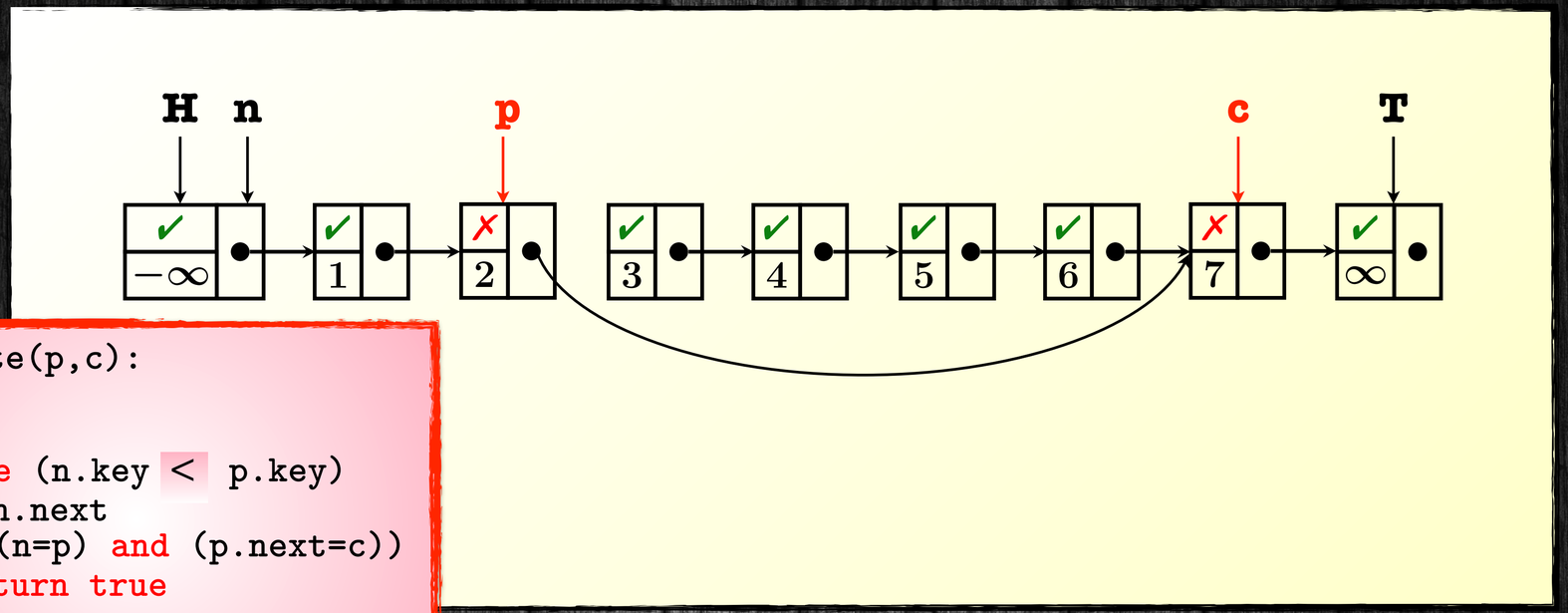

```
validate(p,c):
Node n
1  n=H
2  while (n.key < p.key)
3    n=n.next
4  if ((n=p) and (p.next=c))
5    return true
6  else
7    return false
```



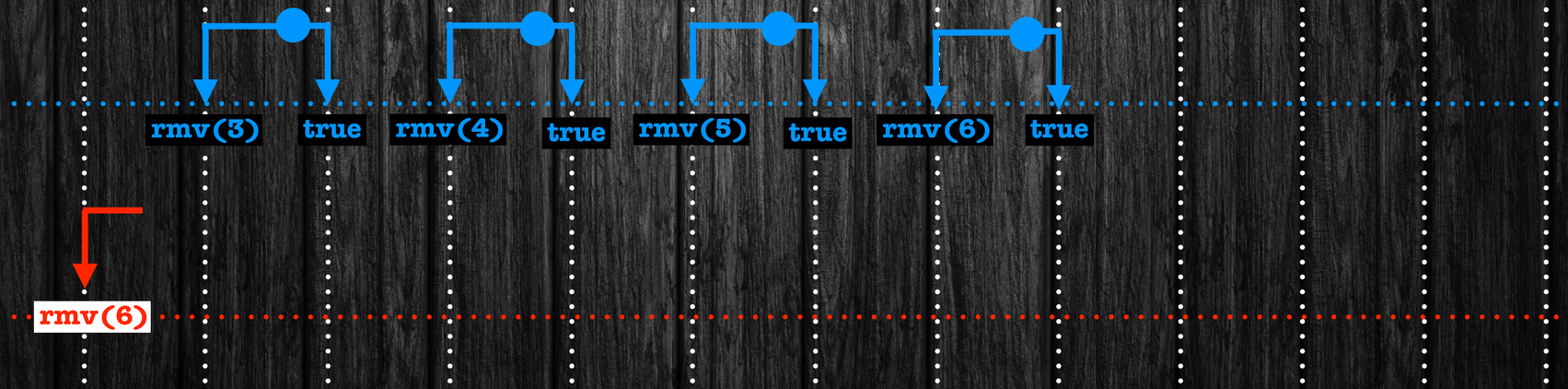
Sets Optimistic Synchronization

Why Validation?

with validation



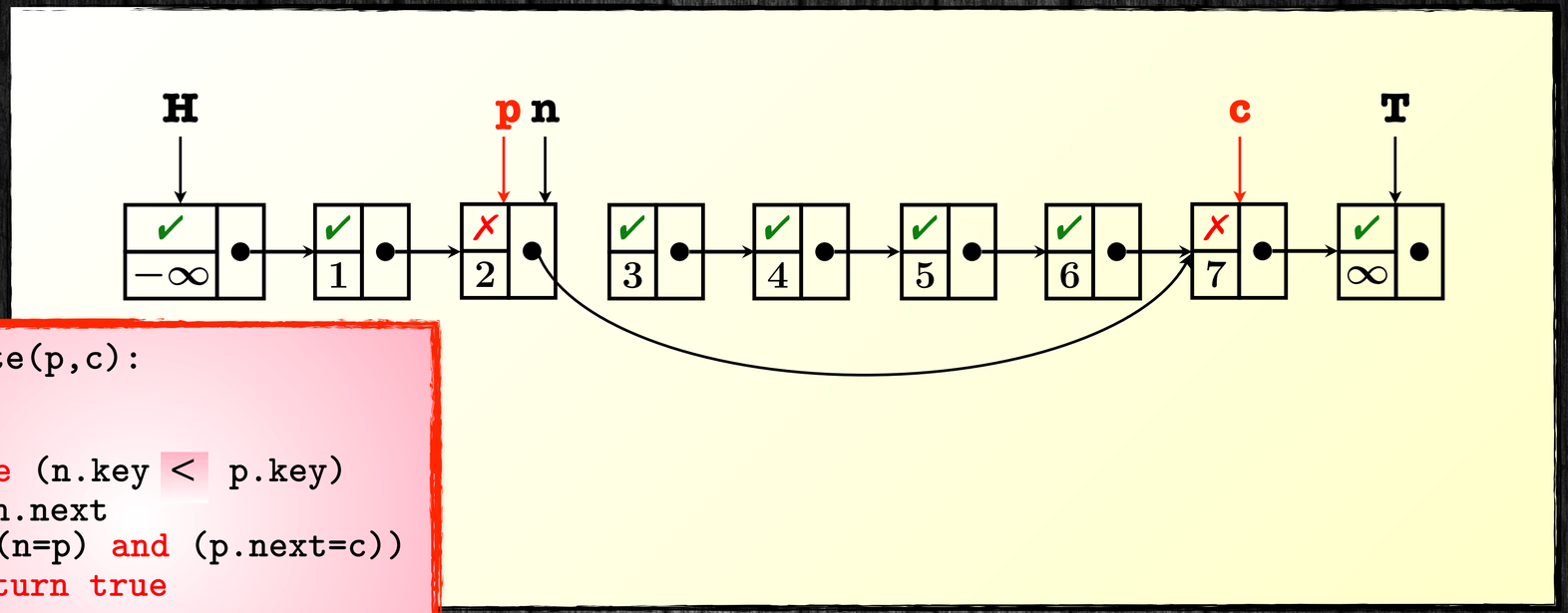

```
validate(p,c):
Node n
1  n=H
2  while (n.key < p.key)
3    n=n.next
4  if ((n=p) and (p.next=c))
5    return true
6  else
7    return false
```



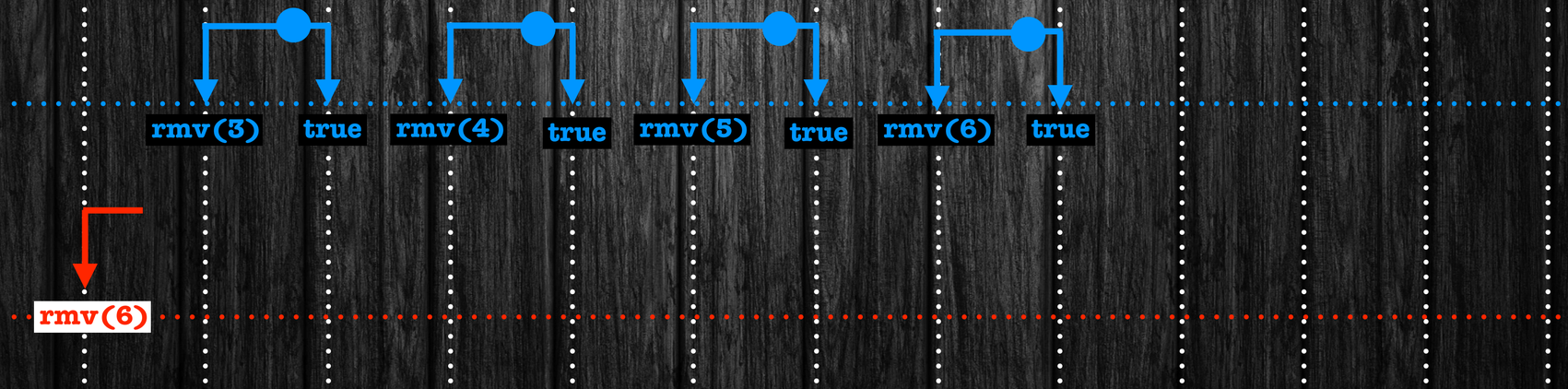
Sets Optimistic Synchronization

Why Validation?

with validation



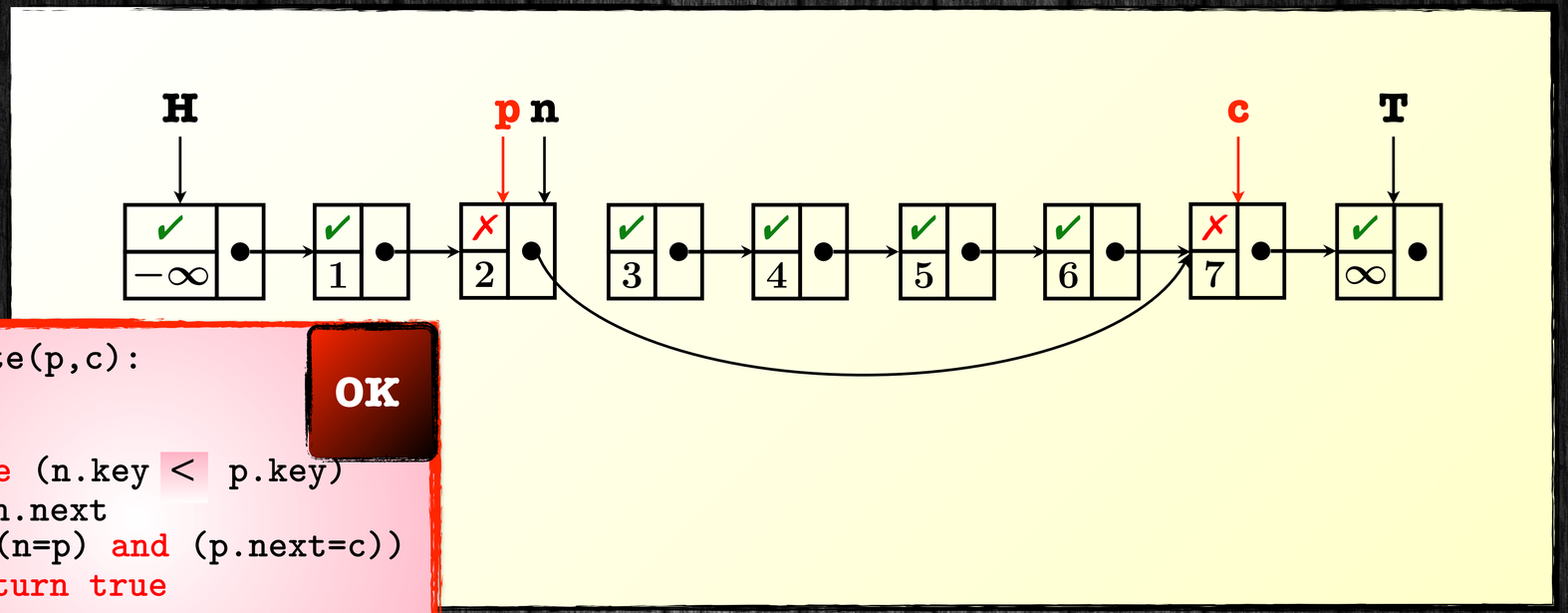

```
validate(p,c):
Node n
1  n=H
2  while (n.key < p.key)
3    n=n.next
4  if ((n=p) and (p.next=c))
5    return true
6  else
7    return false
```



Sets Optimistic Synchronization

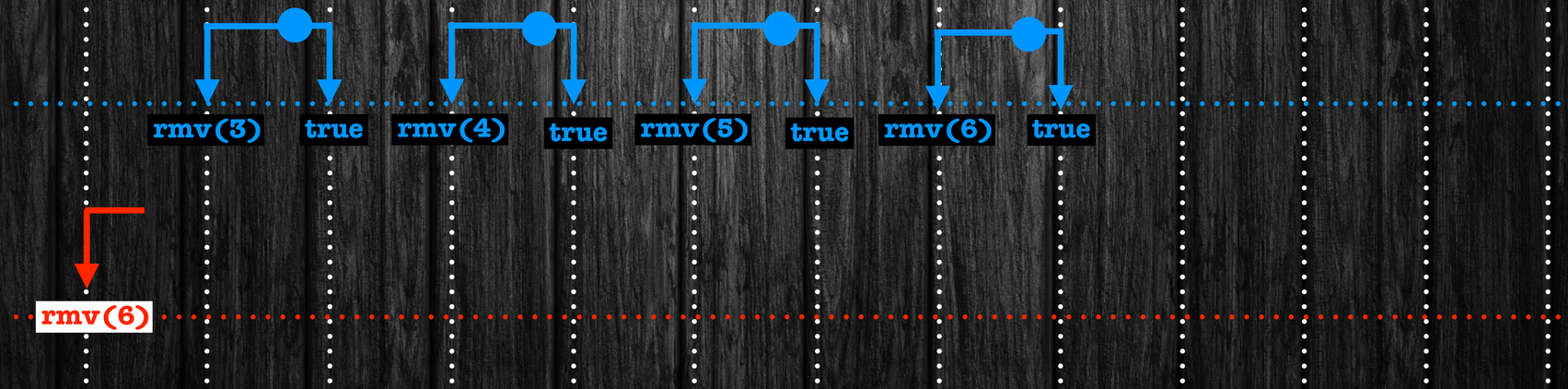
Why Validation?

with validation




```
validate(p,c):
Node n
1  n=H
2  while (n.key < p.key)
3    n=n.next
4  if ((n=p) and (p.next=c))
5    return true
6  else
7    return false
```

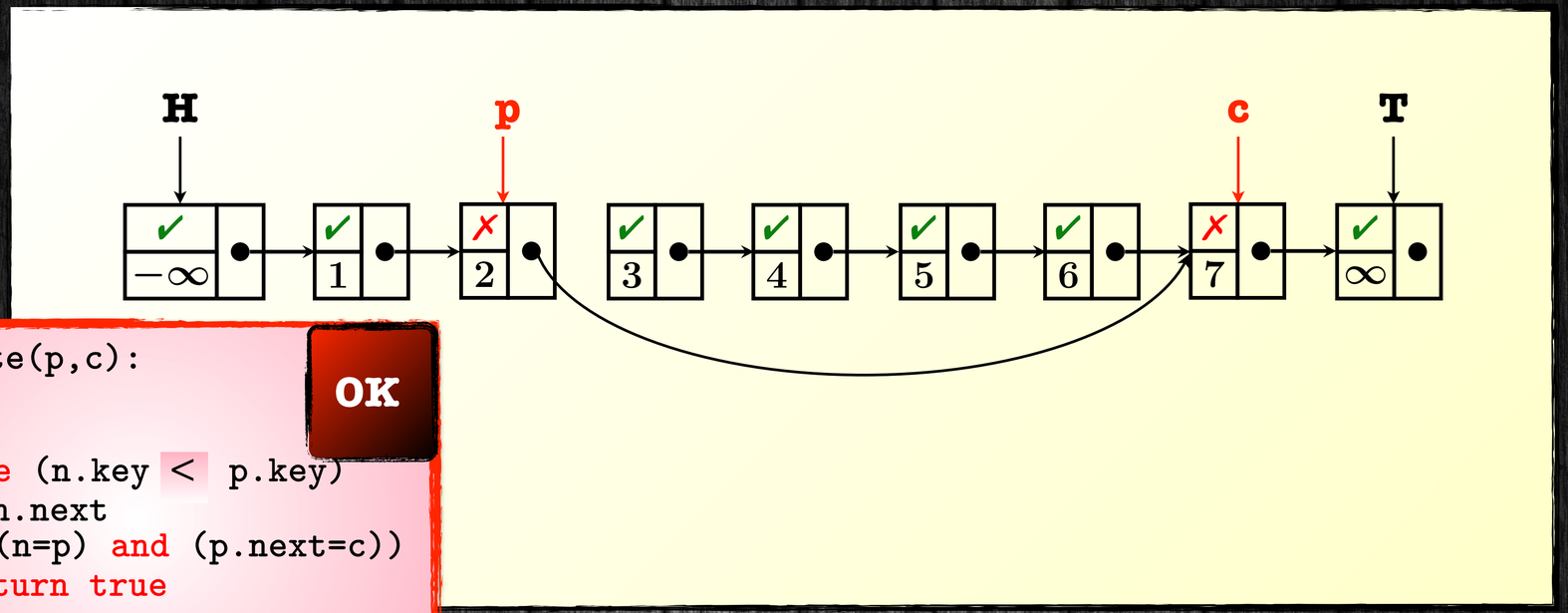
OK



Sets Optimistic Synchronization

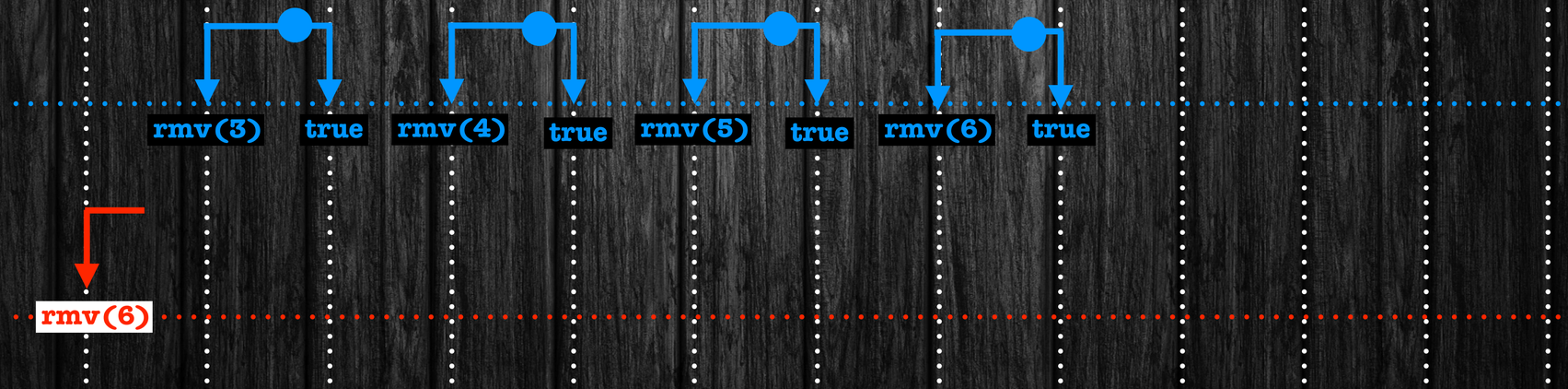
Why Validation?

with validation



OK

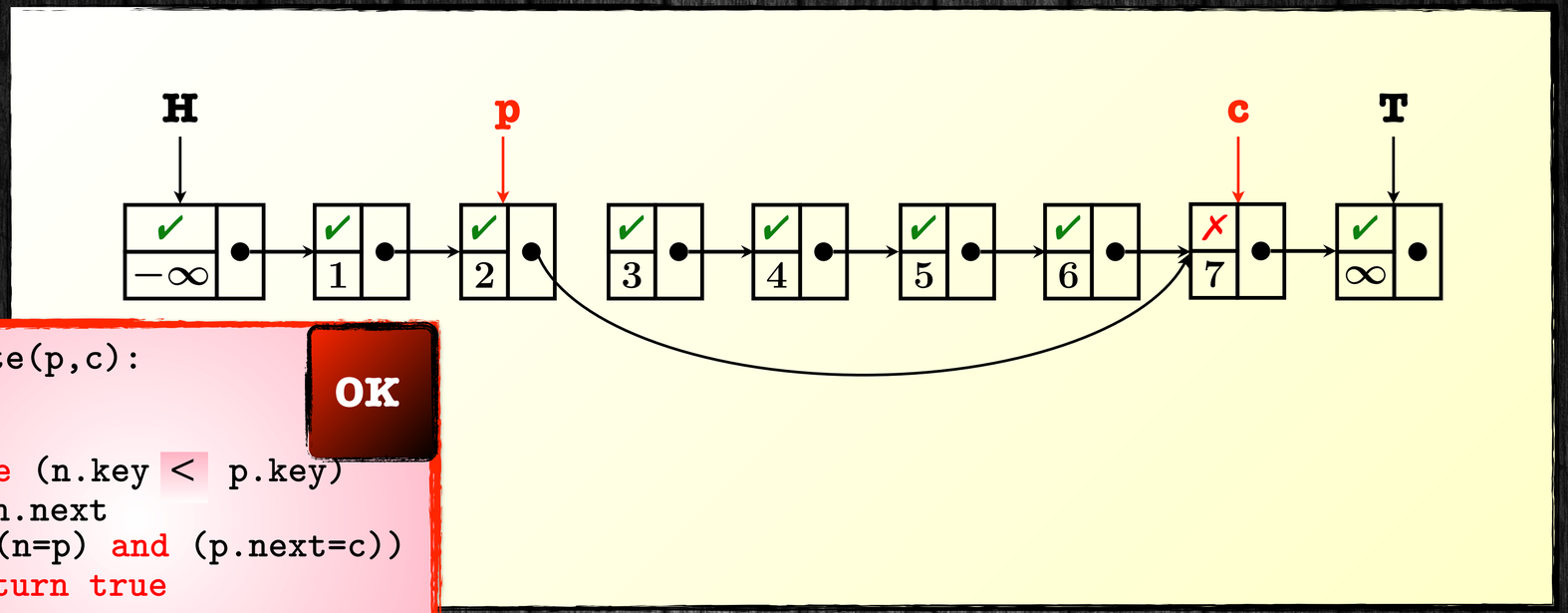
```
validate(p,c):
Node n
1  n=H
2  while (n.key < p.key)
3    n=n.next
4  if ((n=p) and (p.next=c))
5    return true
6  else
7    return false
```



Sets Optimistic Synchronization

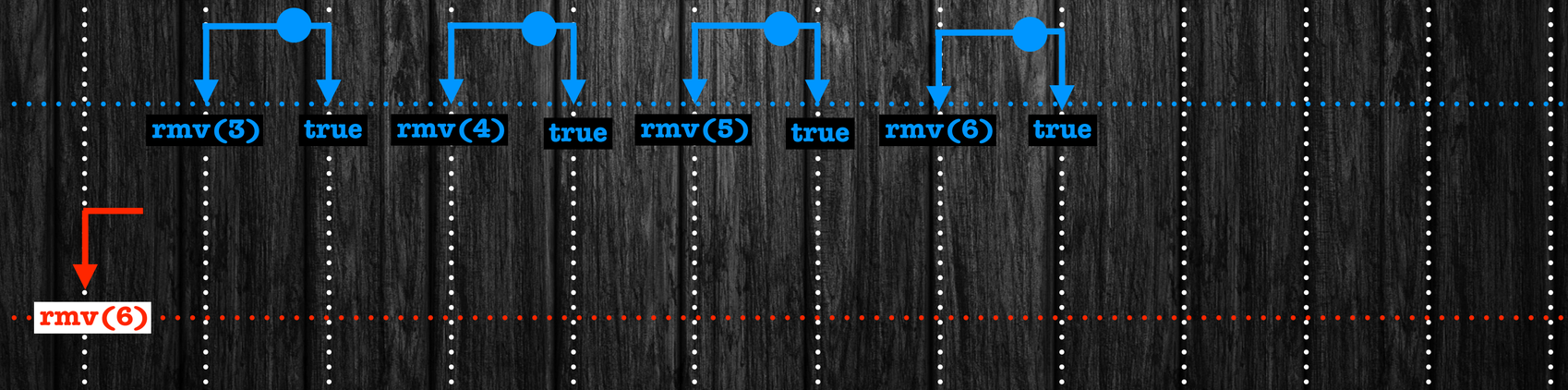
Why Validation?

with validation



OK

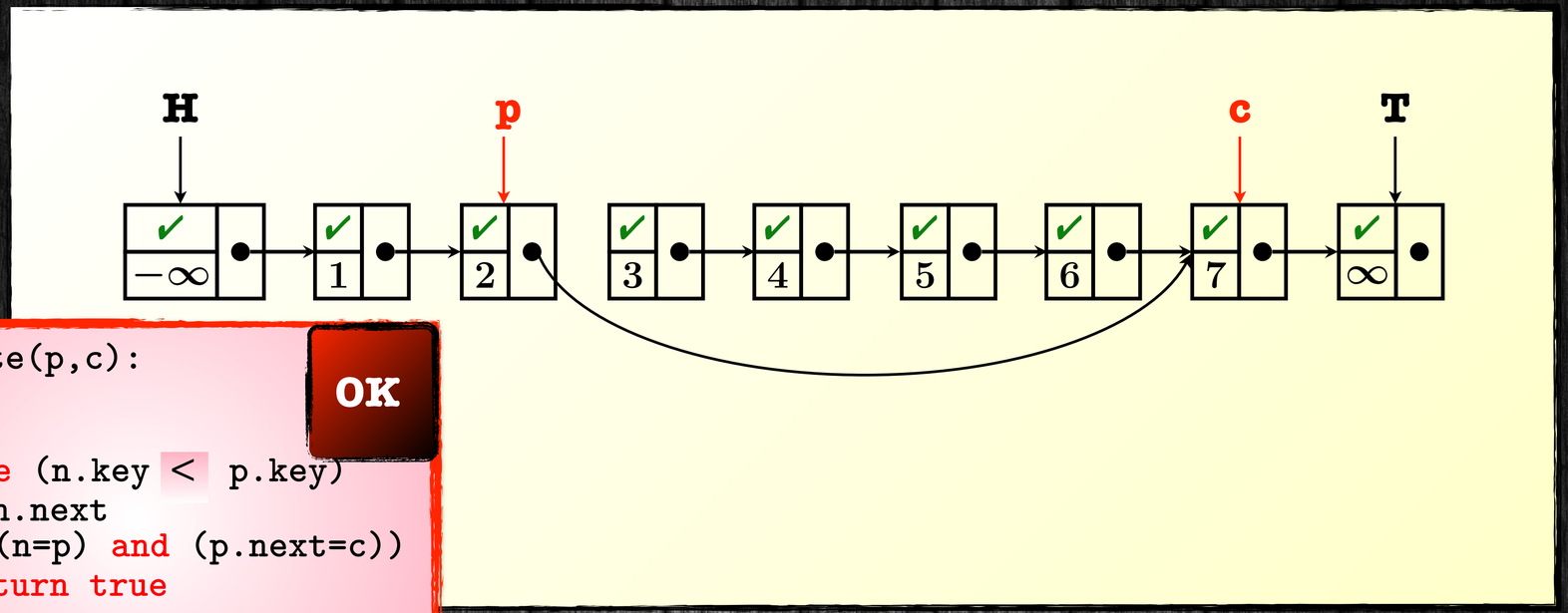
```
validate(p,c):
Node n
1  n=H
2  while (n.key < p.key)
3    n=n.next
4  if ((n=p) and (p.next=c))
5    return true
6  else
7    return false
```



Sets Optimistic Synchronization

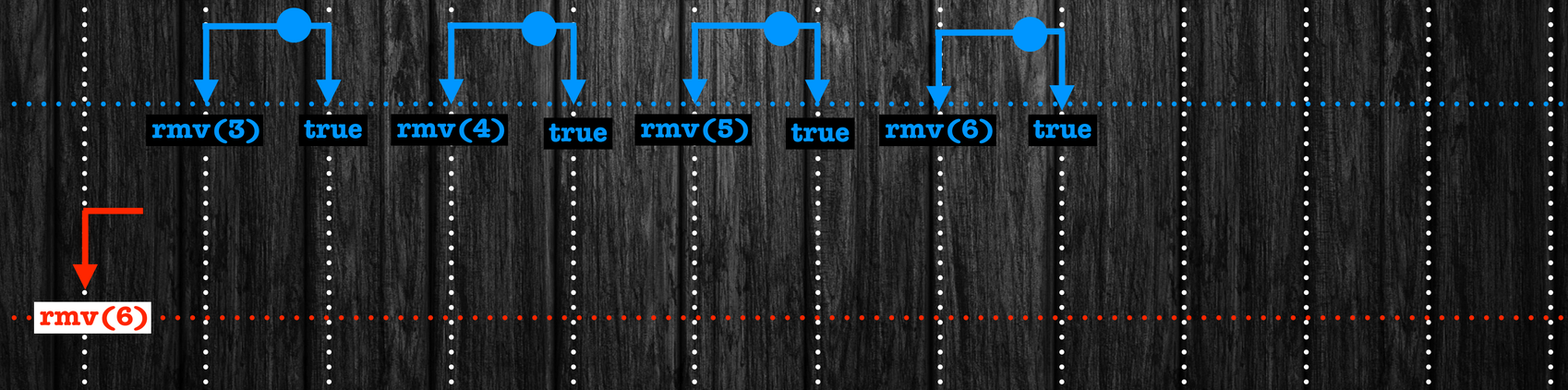
Why Validation?

with validation



OK

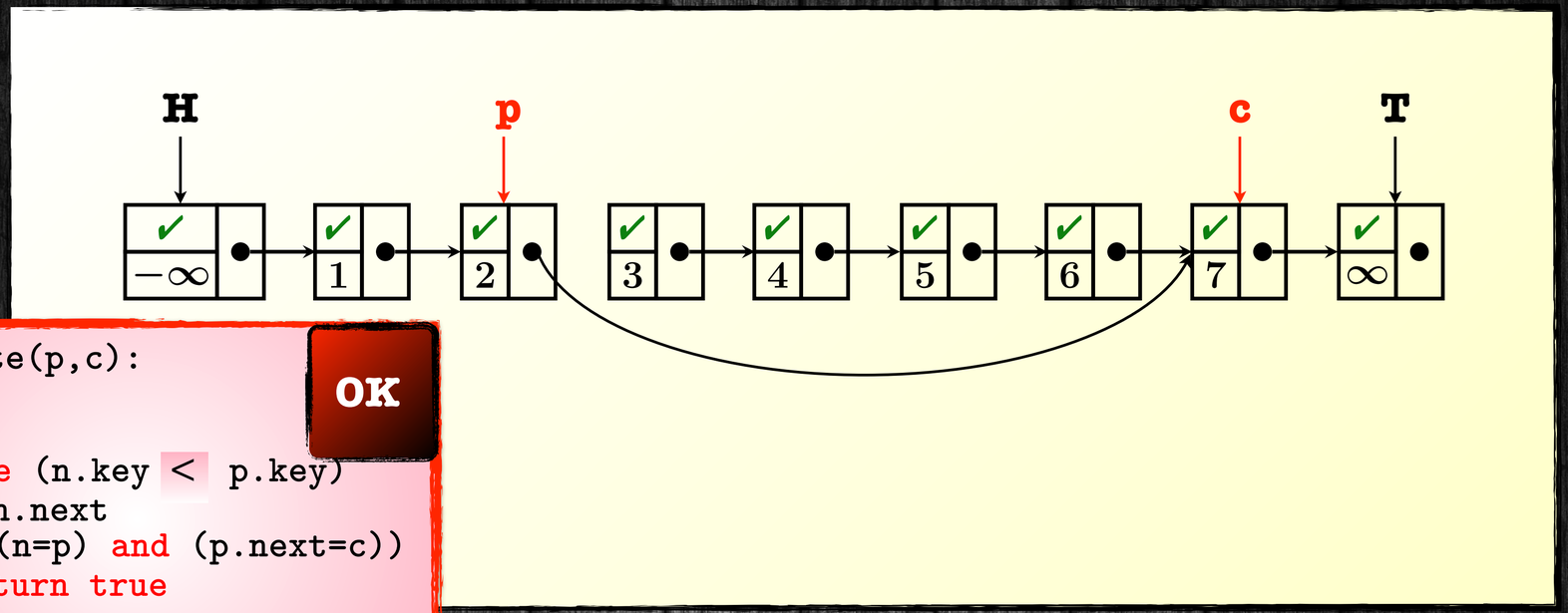
```
validate(p,c):
Node n
1  n=H
2  while (n.key < p.key)
3    n=n.next
4  if ((n=p) and (p.next=c))
5    return true
6  else
7    return false
```



Sets Optimistic Synchronization

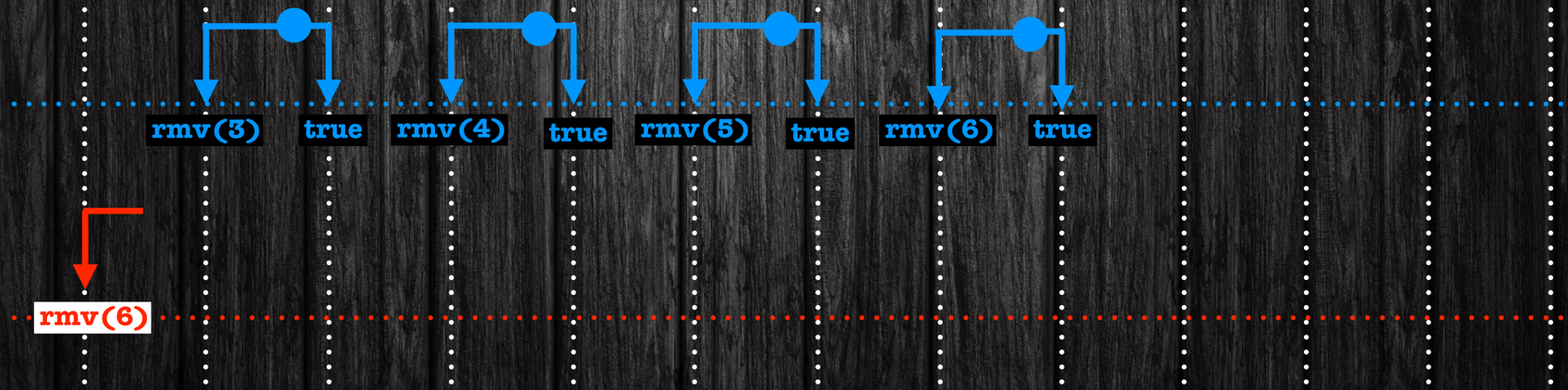
Why Validation?

with validation



OK

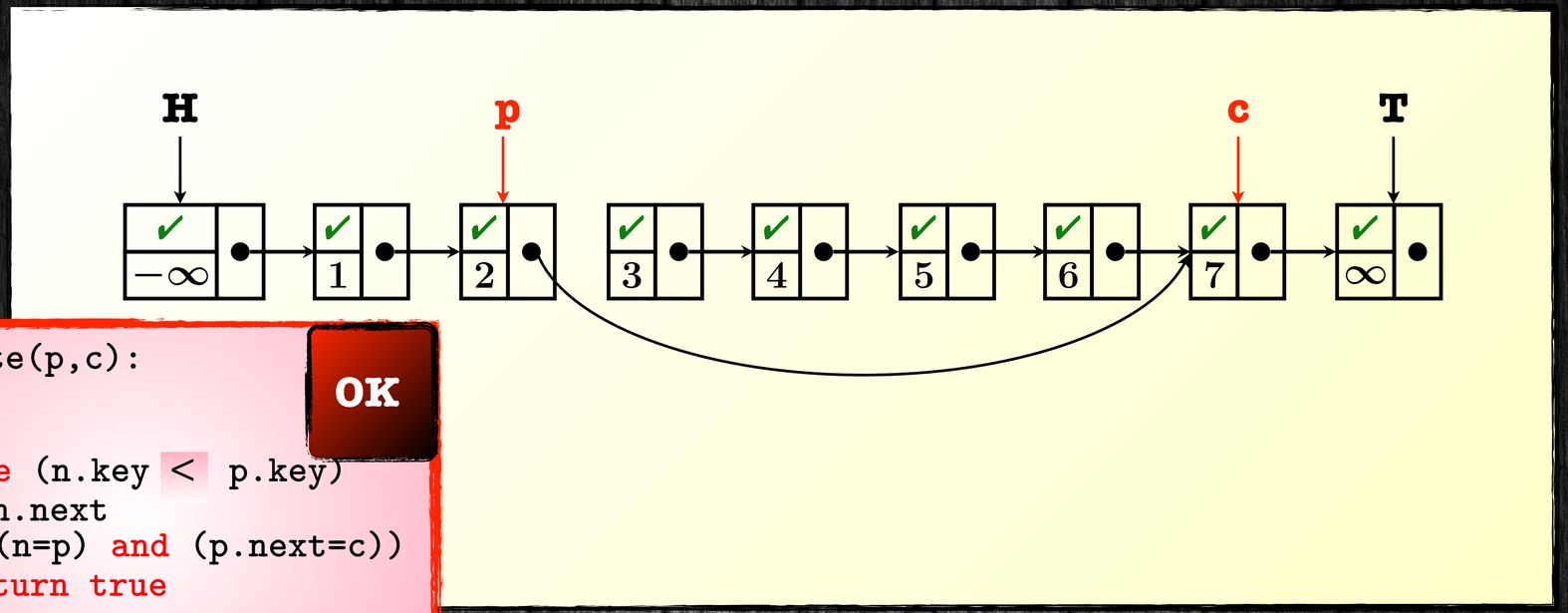
```
validate(p,c):
Node n
1  n=H
2  while (n.key < p.key)
3    n=n.next
4  if ((n=p) and (p.next=c))
5    return true
6  else
7    return false
```



Sets Optimistic Synchronization

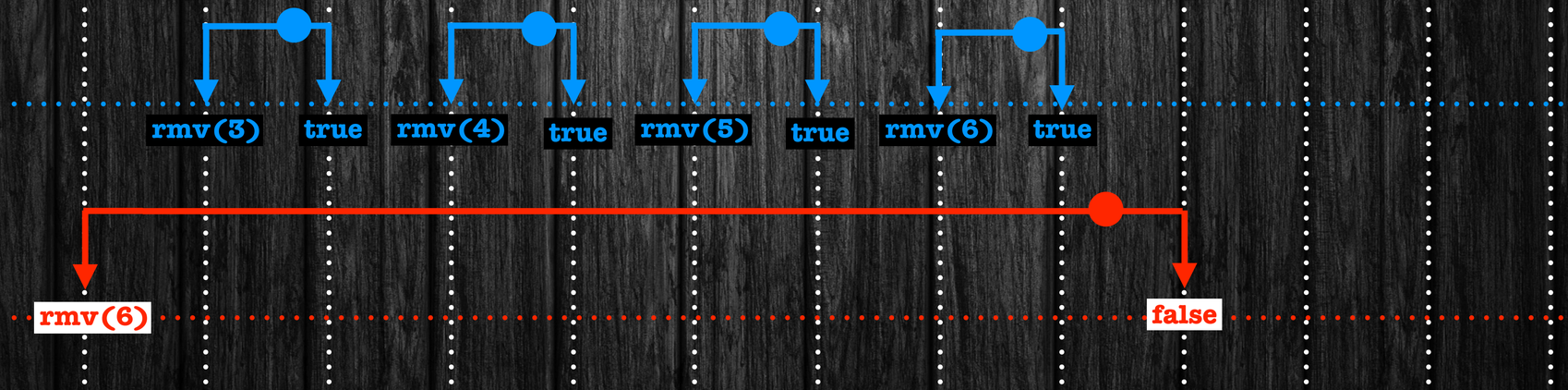
Why Validation?

with validation




```
validate(p,c):
Node n
1  n=H
2  while (n.key < p.key)
3    n=n.next
4  if ((n=p) and (p.next=c))
5    return true
6  else
7    return false
```

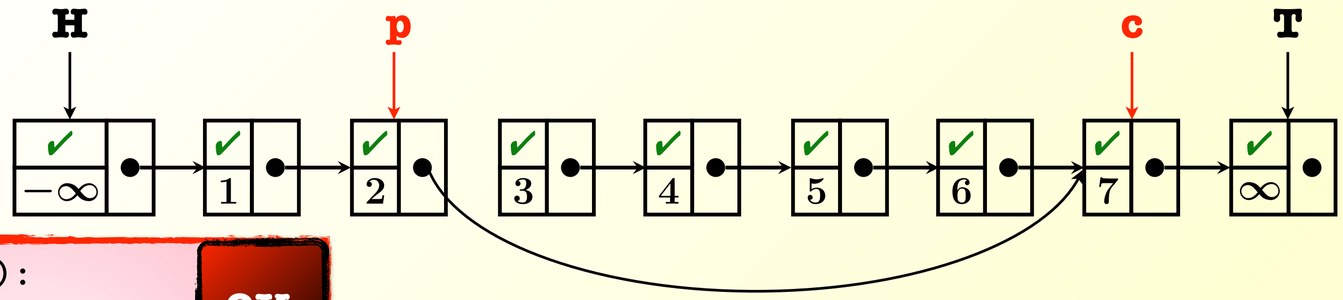
OK



Sets Optimistic Synchronization

Why Validation?

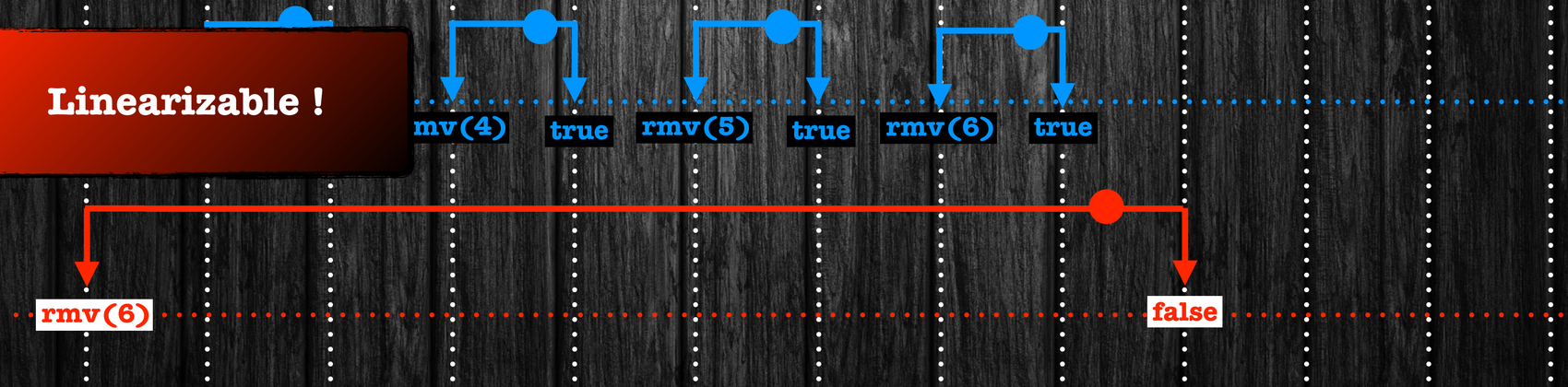
with validation




```
validate(p,c):
Node n
1  n=H
2  while (n.key < p.key)
3    n=n.next
4  if ((n=p) and (p.next=c))
5    return true
6  else
7    return false
```

OK

Linearizable !

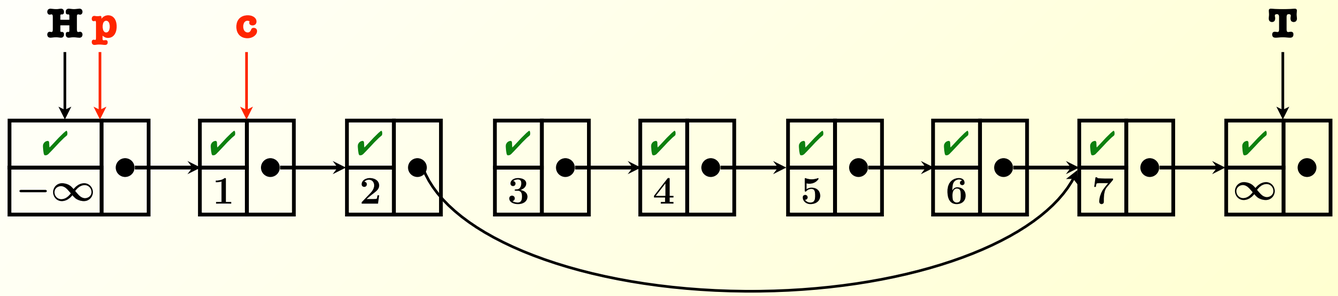


Sets

Optimistic Synchronization

Starvation-Freedom

with validation

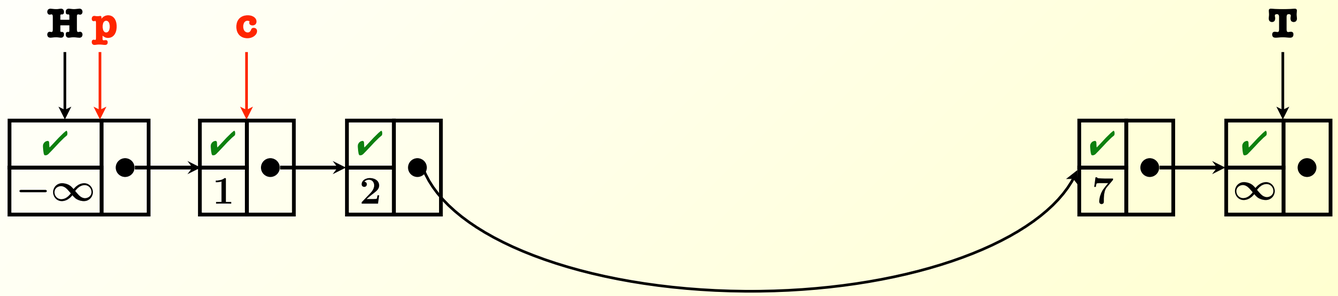


Sets

Optimistic Synchronization

Starvation-Freedom

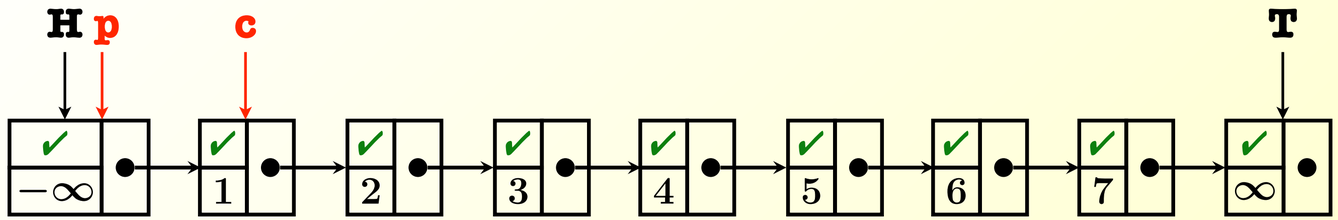
with validation



Sets

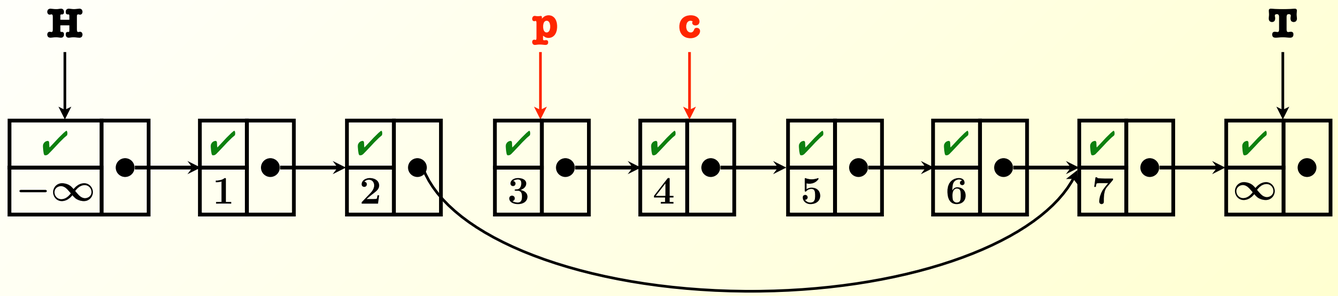
Optimistic Synchronization

Starvation-Freedom



Sets | Optimistic Synchronization

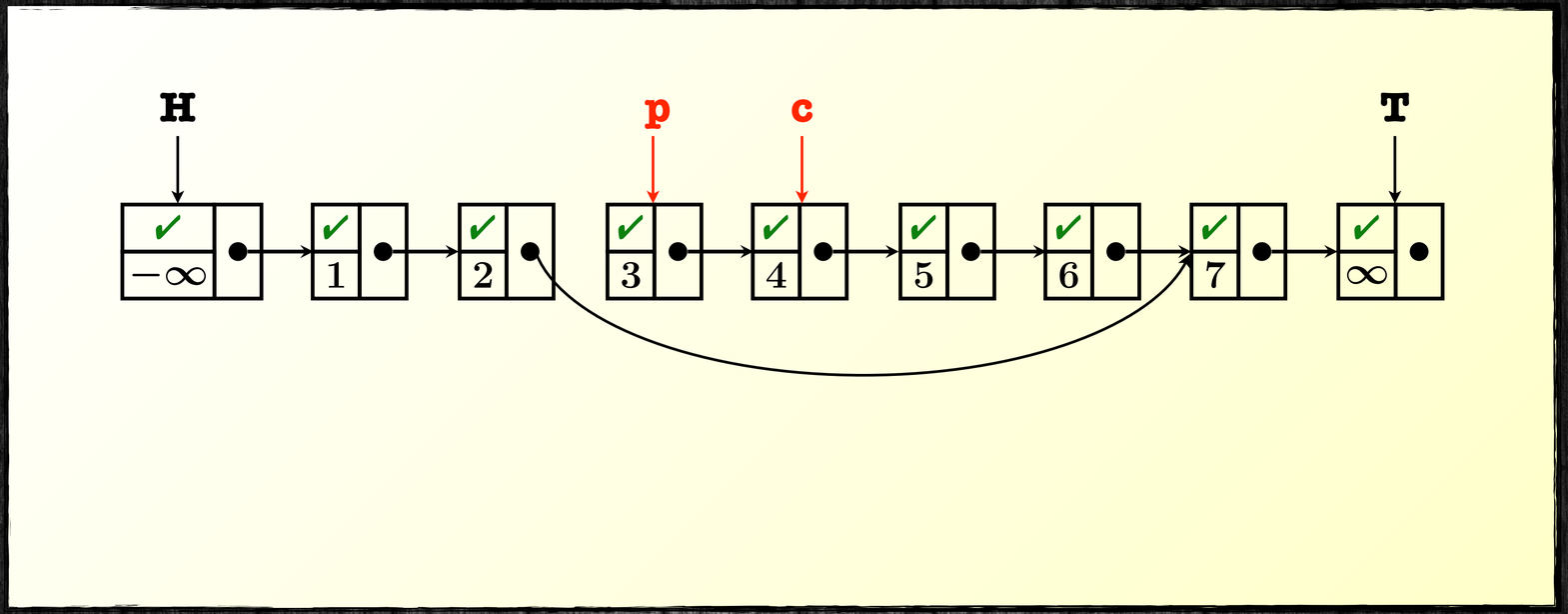
Starvation-Freedom



Sets

Optimistic Synchronization

Starvation-Freedom

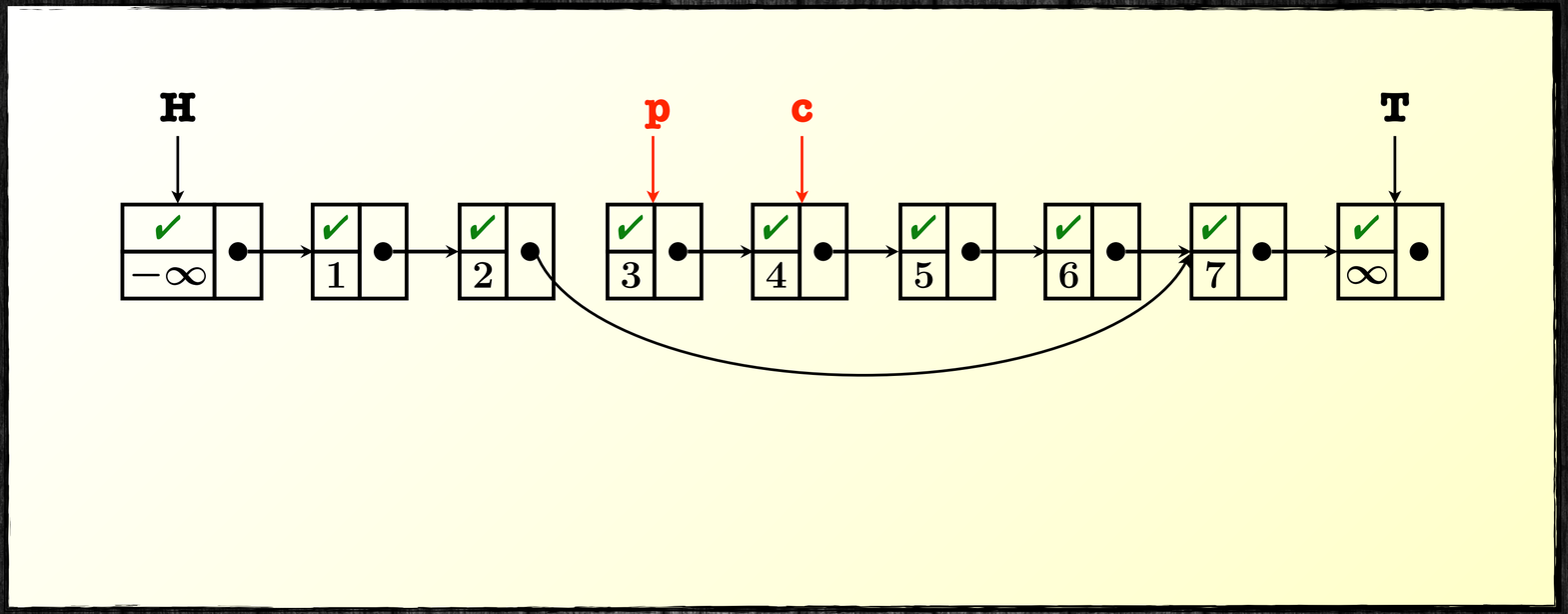


repeated failure

Sets Optimistic Synchronization

Starvation-Freedom

no



repeated failure