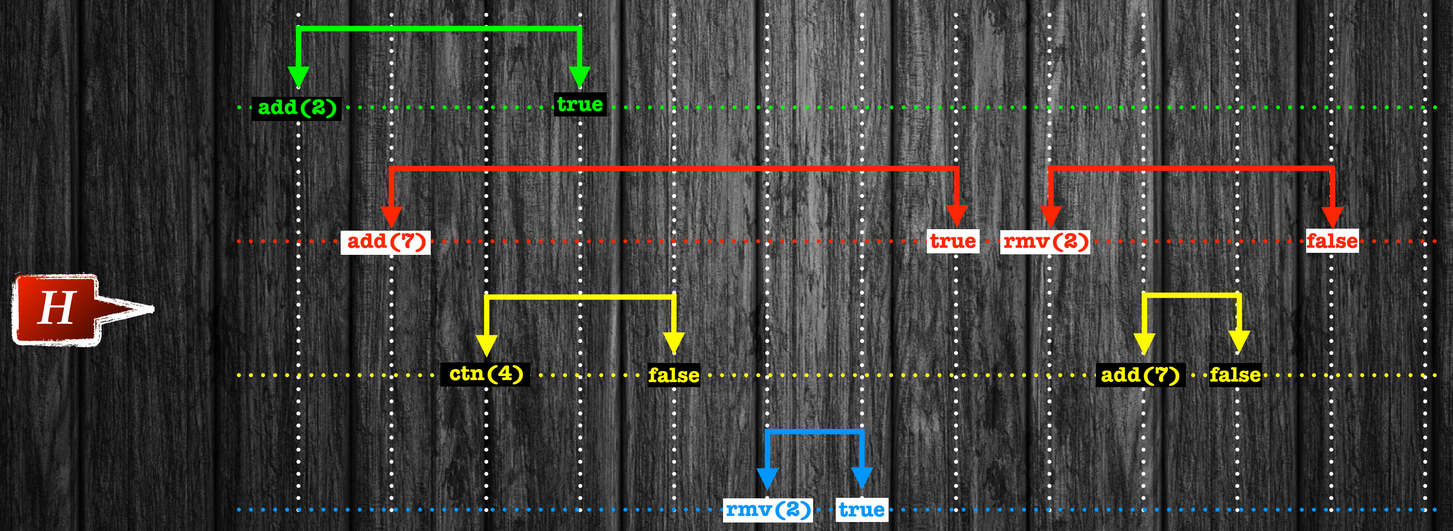
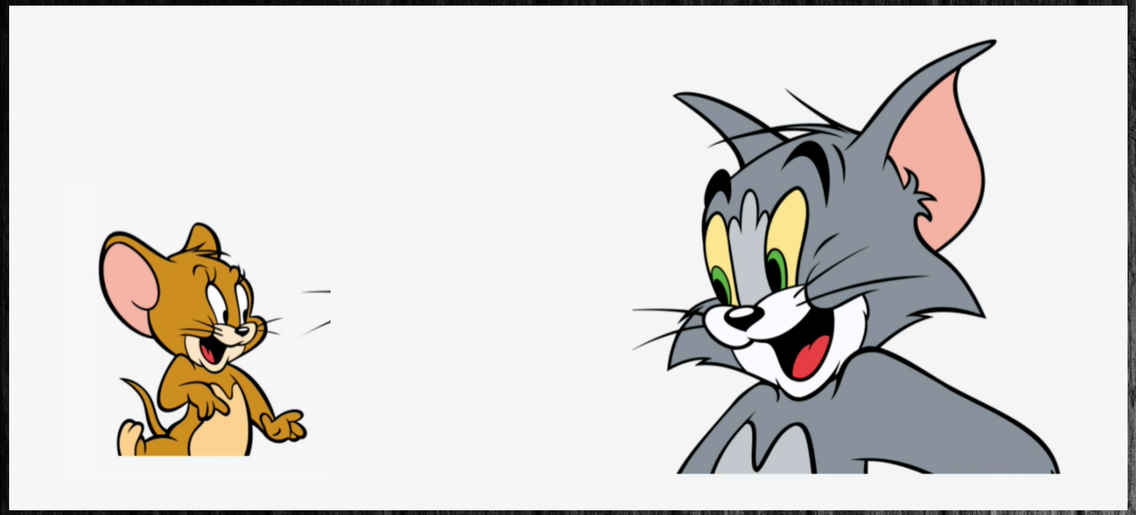
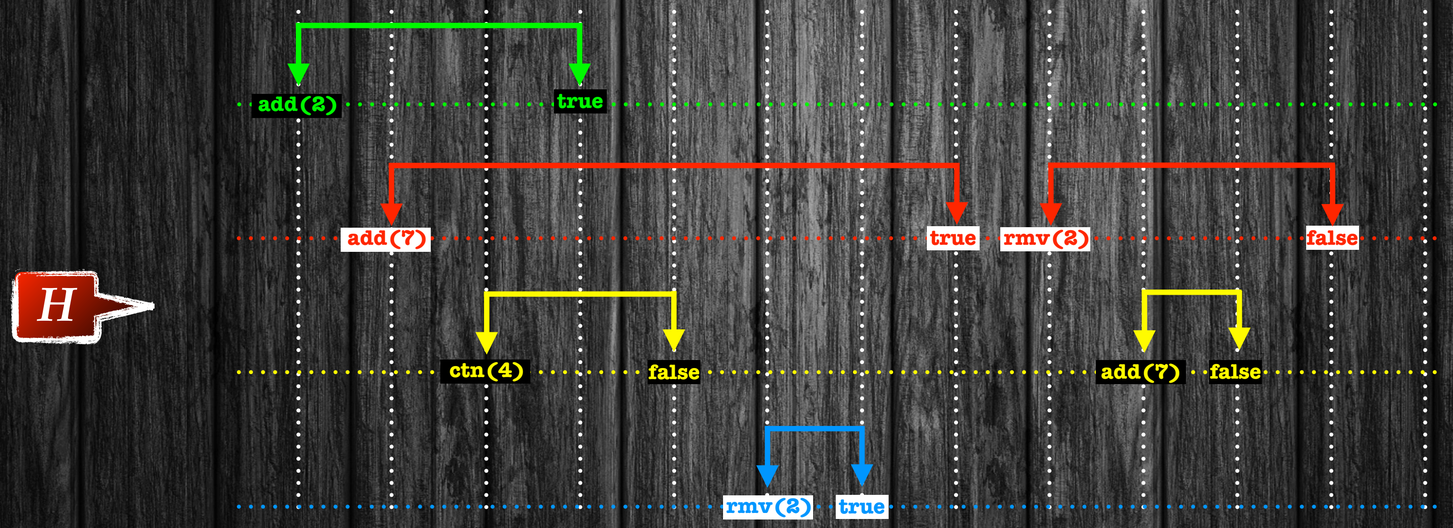
Outline

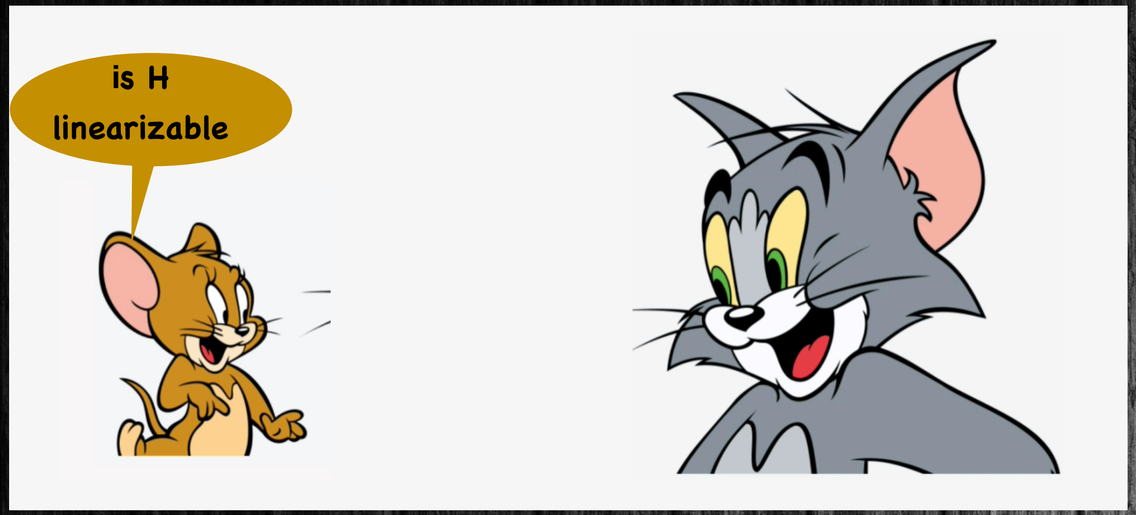
- Overview
- Abstract Data Types
- Sequential Programs
- Concurrent Programs
- Coarse-Grained Synchronization
- Fine-Grained Synchronization
- Optimistic Synchronization
- Lazy Synchronization
- Treiber Stack Algorithm
- Michael&Scott Queue Algorithm

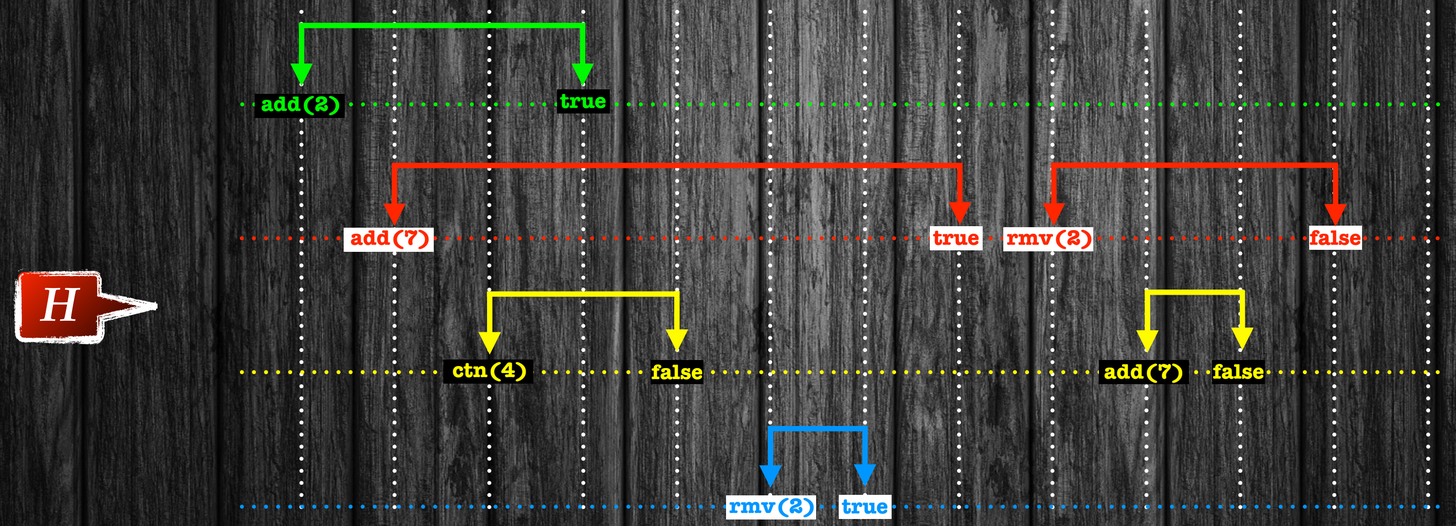
“Linearizability in a nutshell”

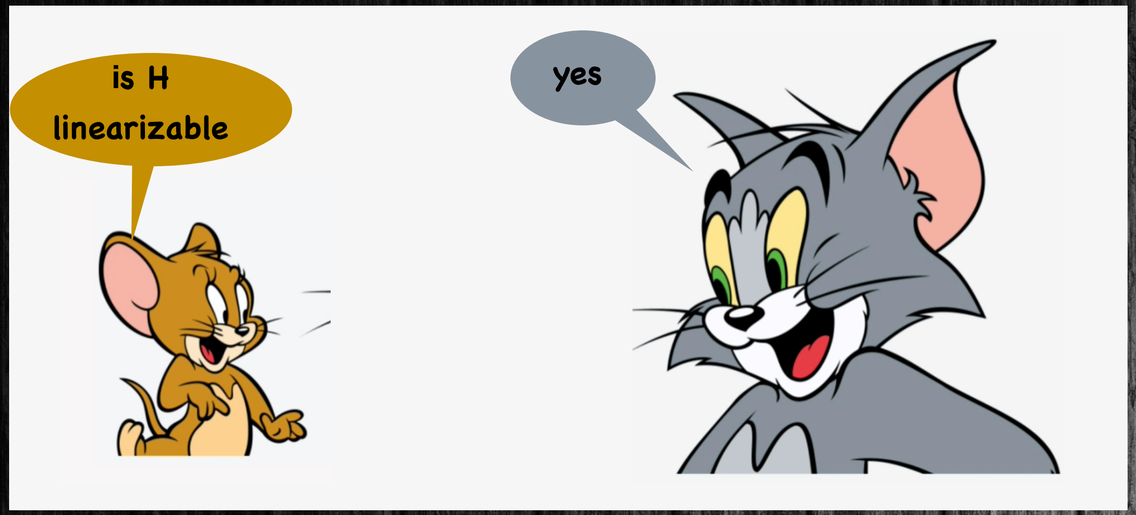

add(2)
true
add(7)
true
rmv(2)
false
H
ctn(4)
false
add(7)
false
rmv(2)
true

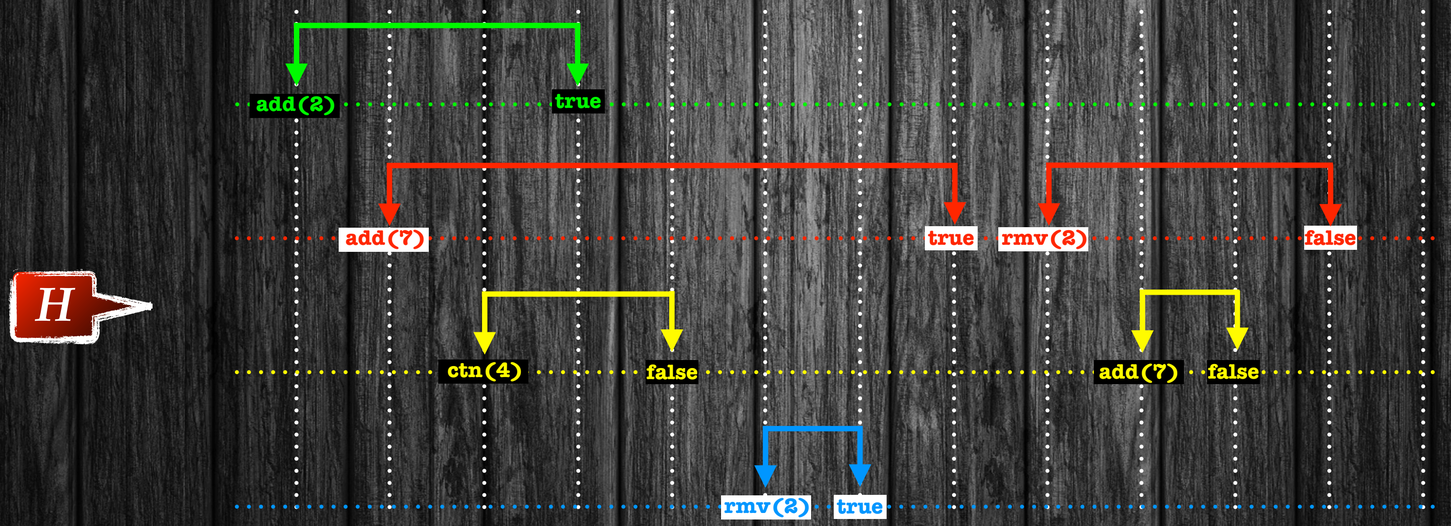


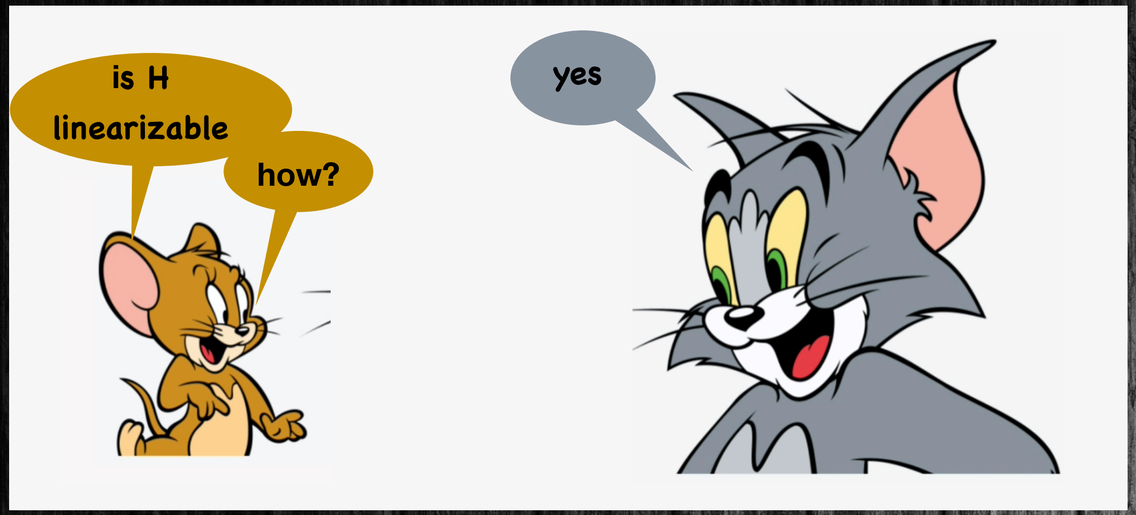

add(2)
true
add(7)
true
rmv(2)
false
ctn(4)
false
add(7)
false
rmv(2)
true

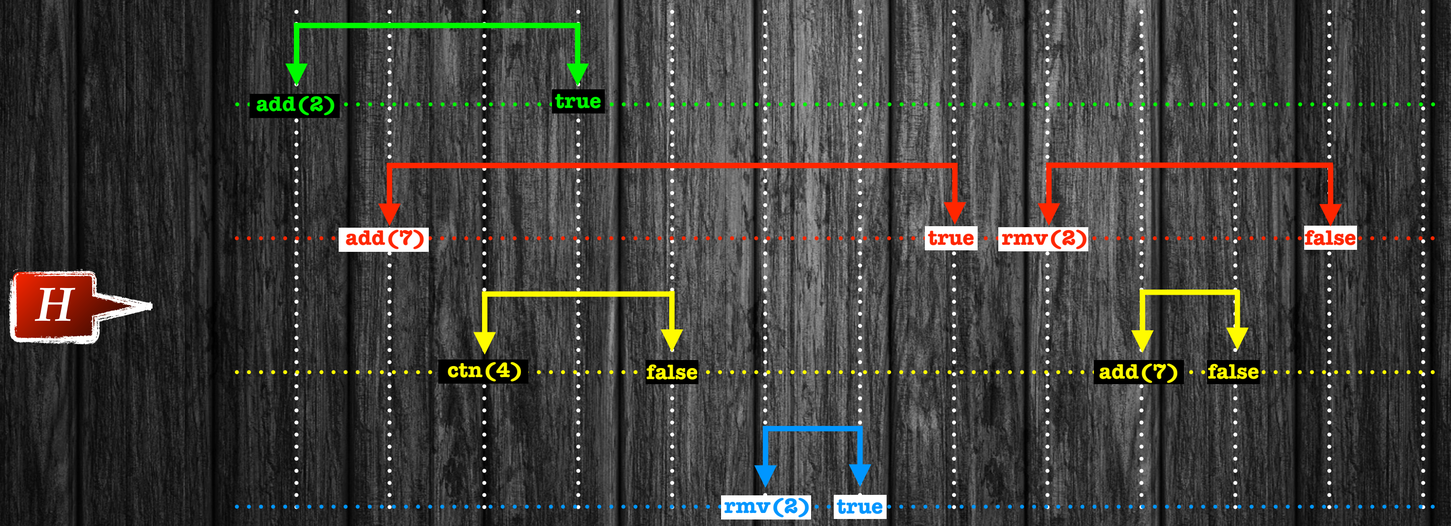

is H
linearizable

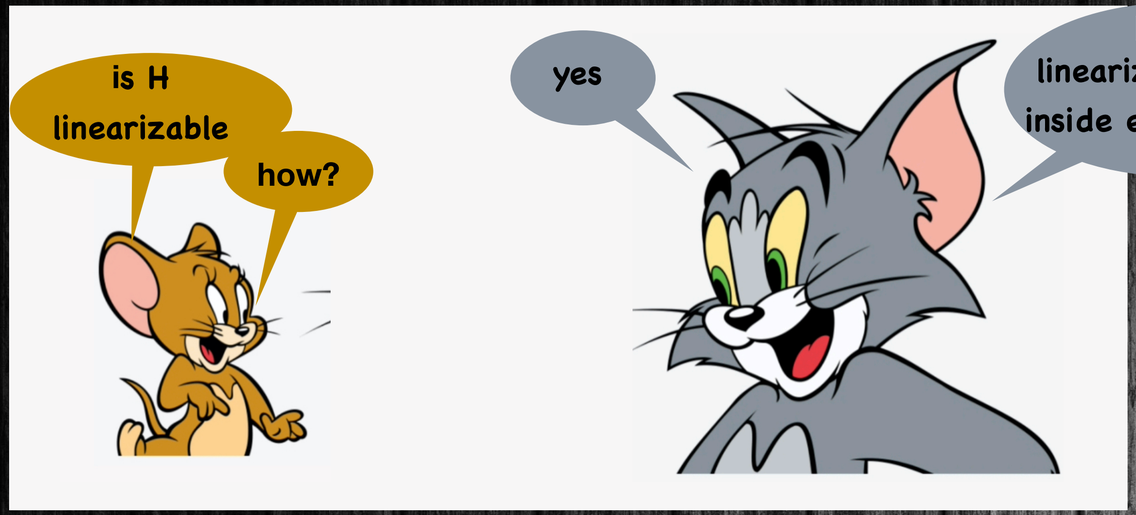

H
add(2)
true
add(7)
true
rmv(2)
false
ctn(4)
false
add(7)
false
rmv(2)
true

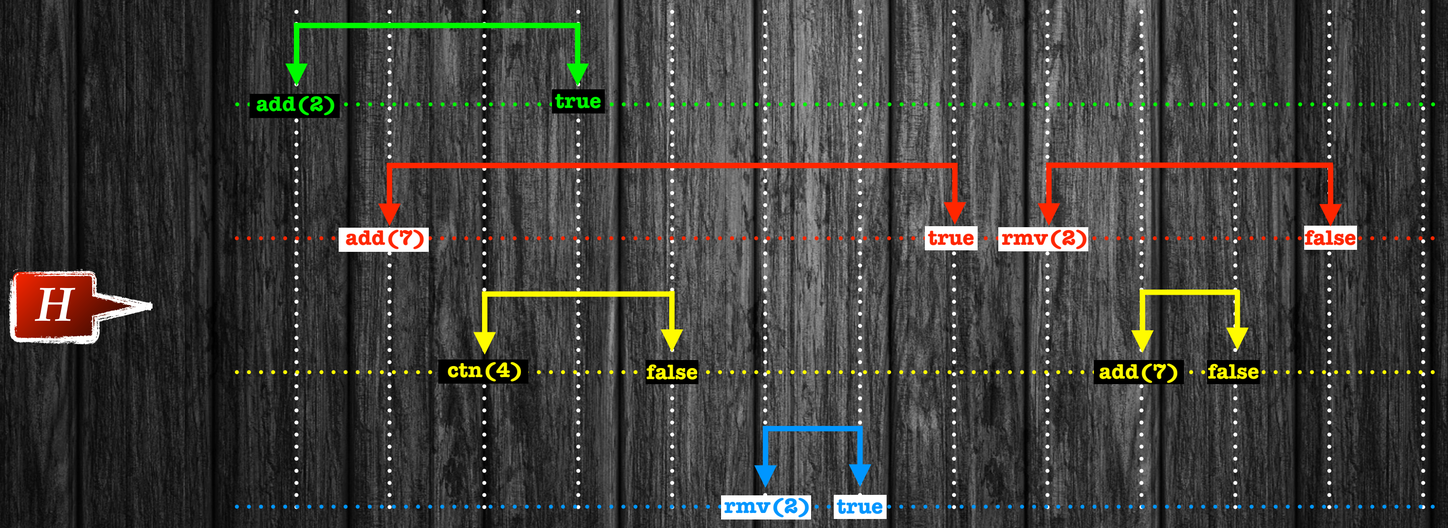

is H
linearizable
yes

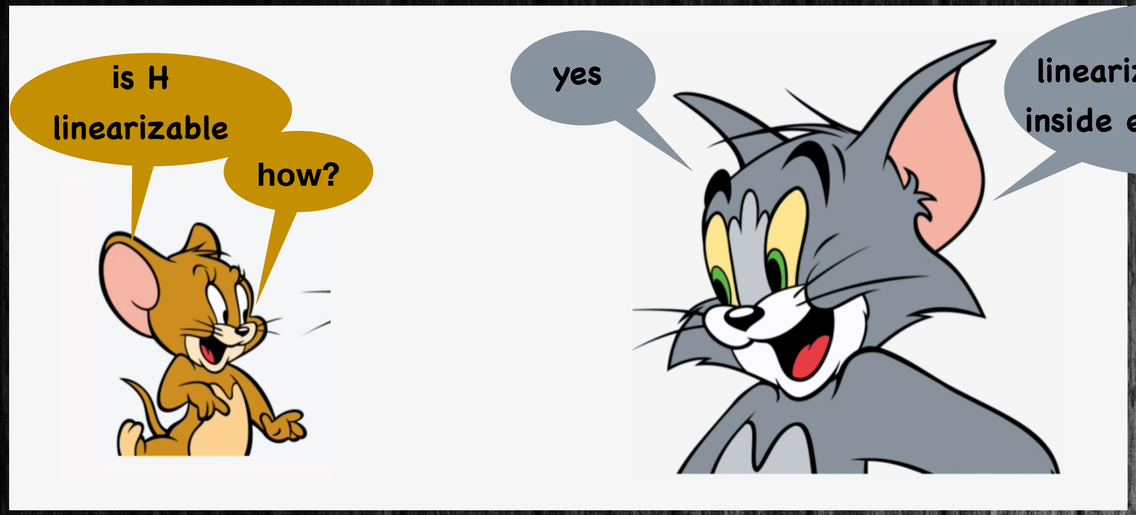

H
add(2)
true
add(7)
true
rmv(2)
false
ctn(4)
false
add(7)
false
rmv(2)
true

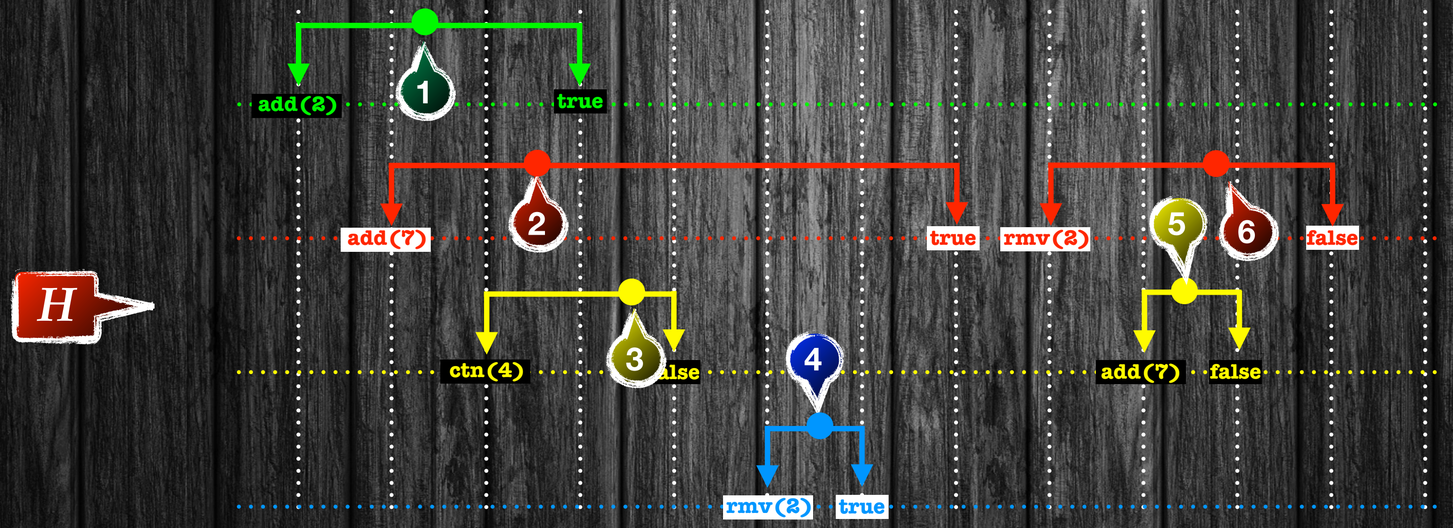

is H
linearizable
how?
yes

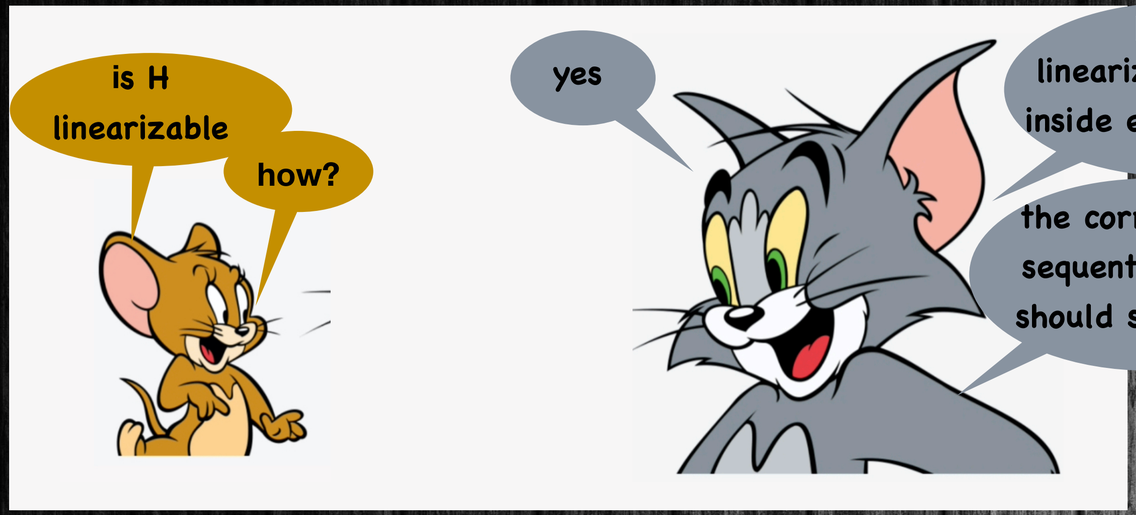

H
add(2)
true
add(7)
true
rmv(2)
false
ctn(4)
false
add(7)
false
rmv(2)
true

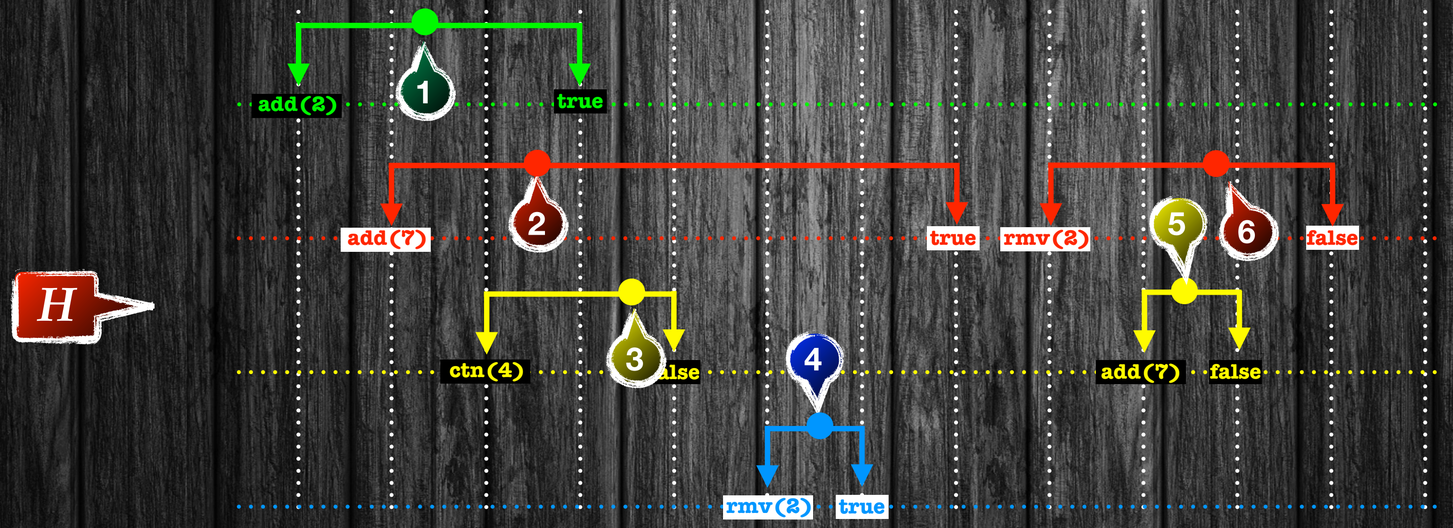

is H linearizable
how?
yes
put a linearization point inside each interval

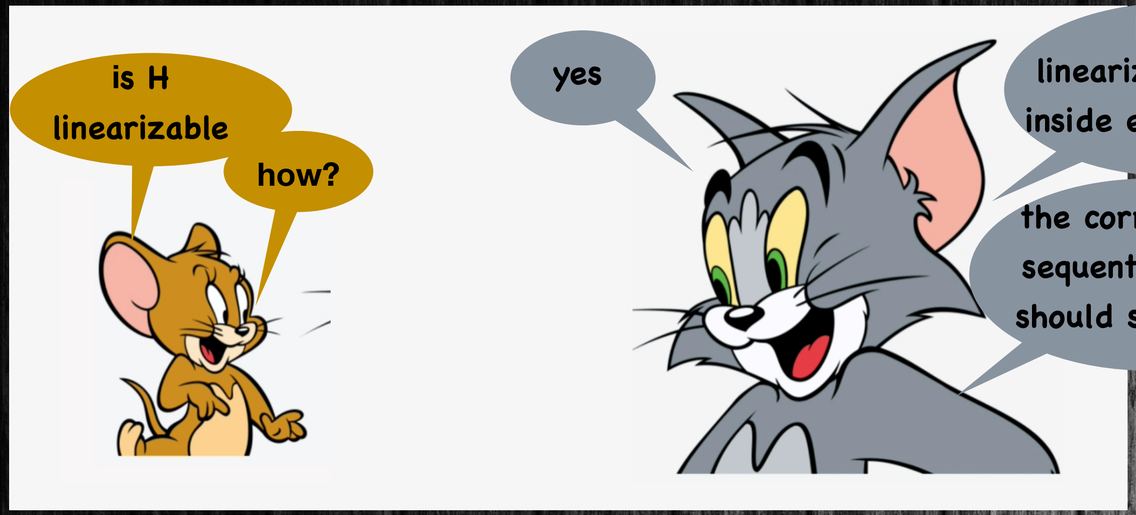

H
add(2)
true
add(7)
true
rmv(2)
false
ctn(4)
false
add(7)
false
rmv(2)
true

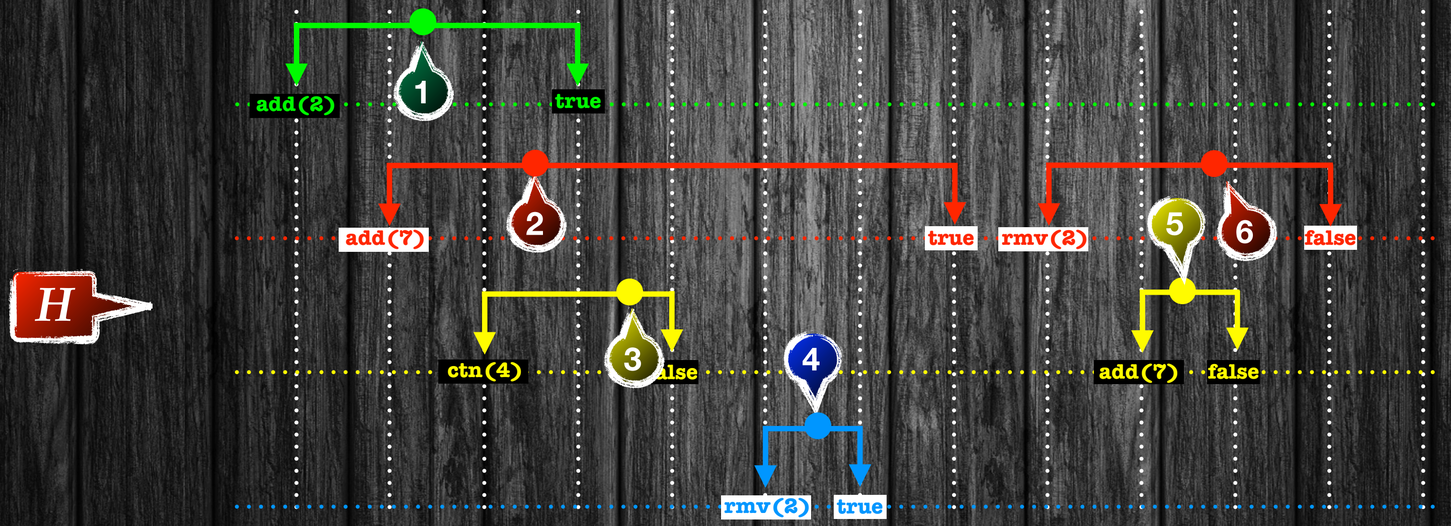

is H linearizable
how?
yes
put a linearization point inside each interval


H
add(2)
1
true
add(7)
2
true
rmv(2)
5
6
false
ctn(4)
3
4
add(7)
false
rmv(2)
true


is H linearizable
how?
yes
put a linearization point inside each interval
the corresponding sequential history should satisfy SET


H
add(2)
1
true
add(7)
2
true
rmv(2)
5
6
false
ctn(4)
3
4
add(7)
false
rmv(2)
true



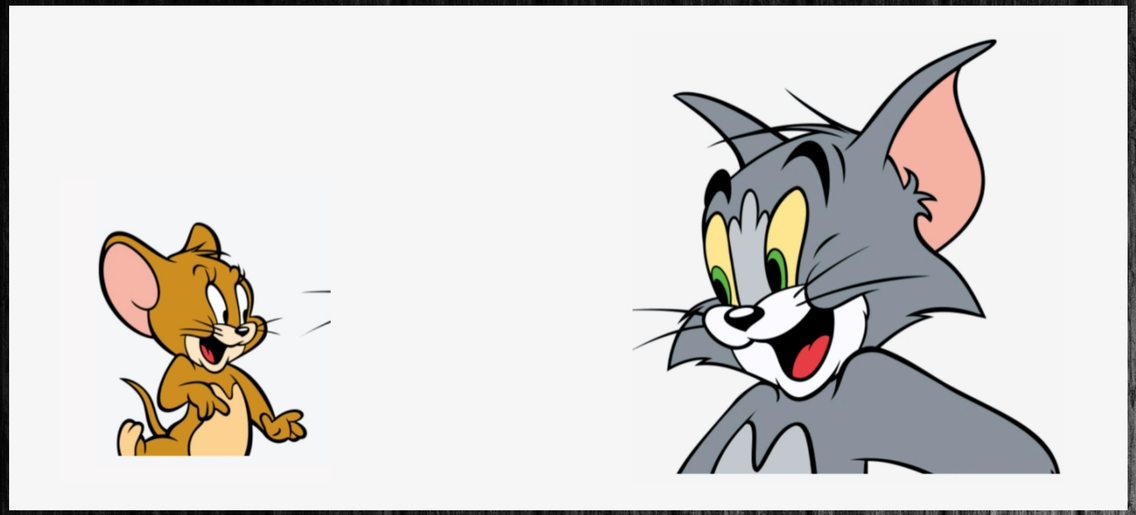


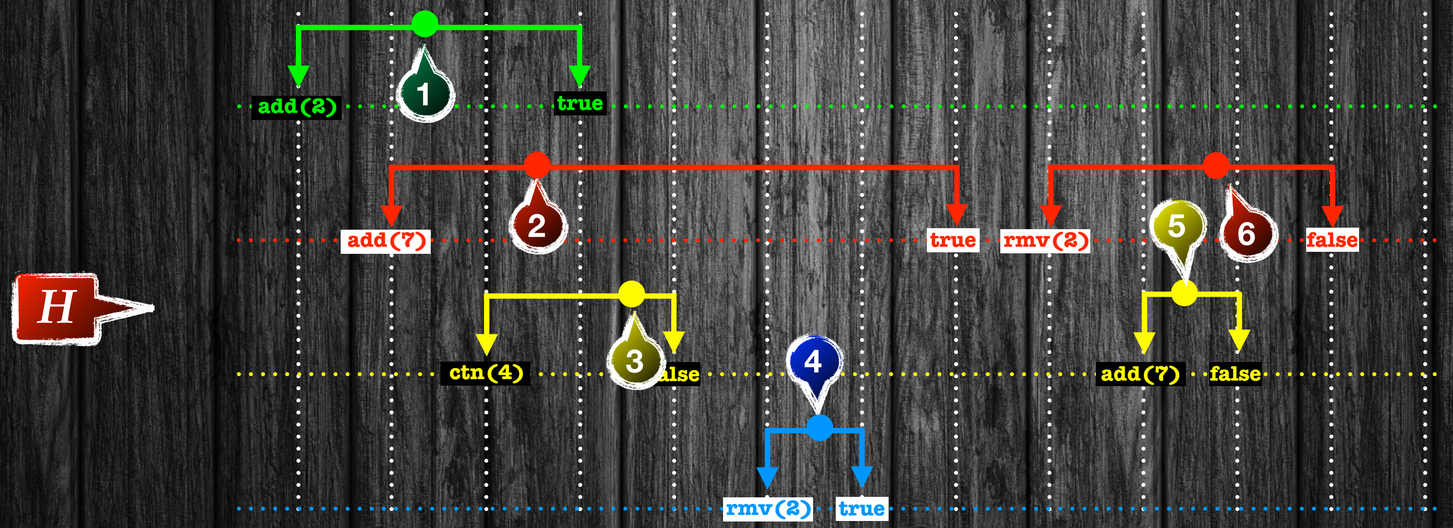
add(2) true | add(7) true | ctn(4) false | rmv(2) true | add(7) false | rmv(2) false

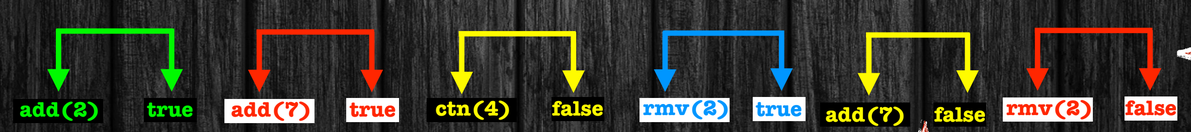
H′

Sequential history

$H' \in \text{linearizations}(H)$ H





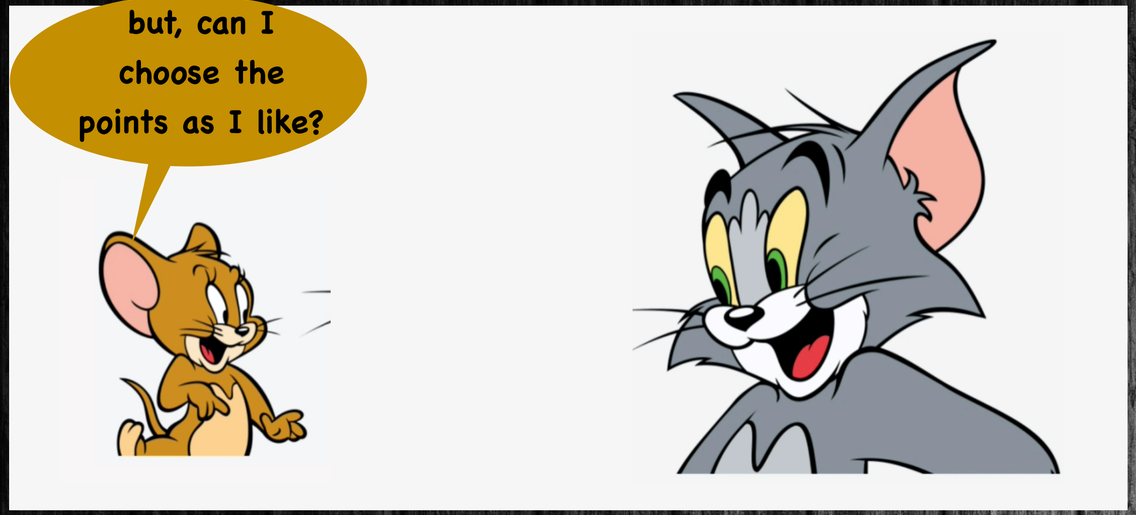


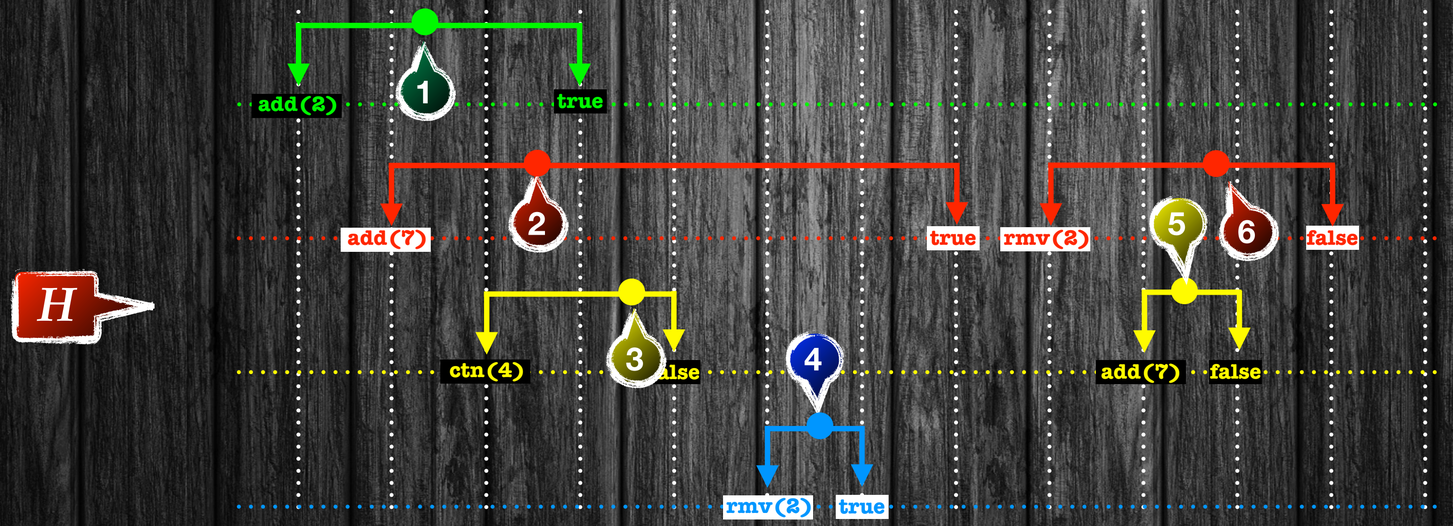
$H' \in$ linearizations(H)

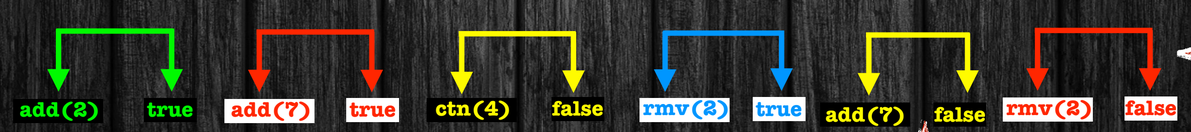
H

H'

Sequential history

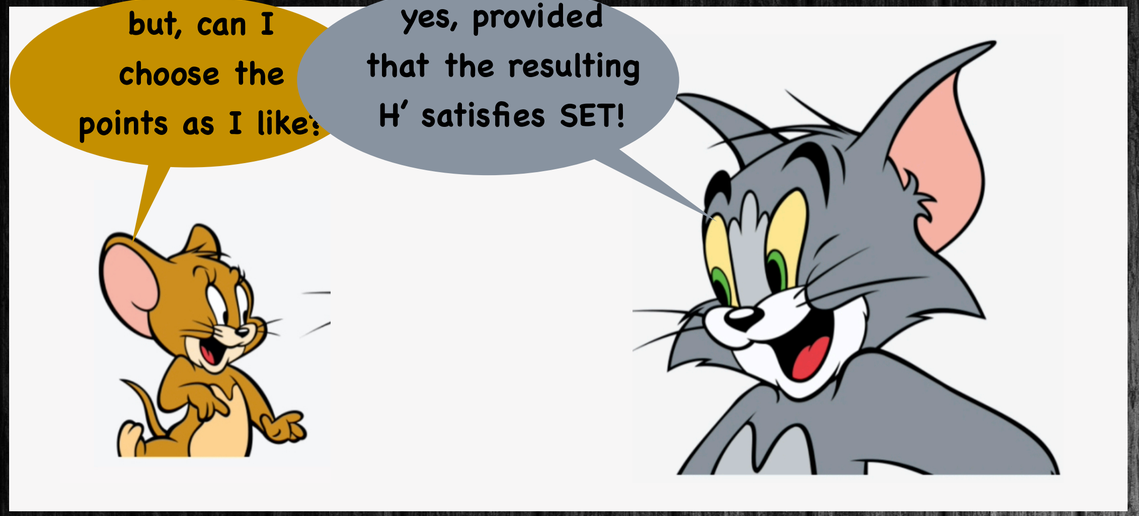


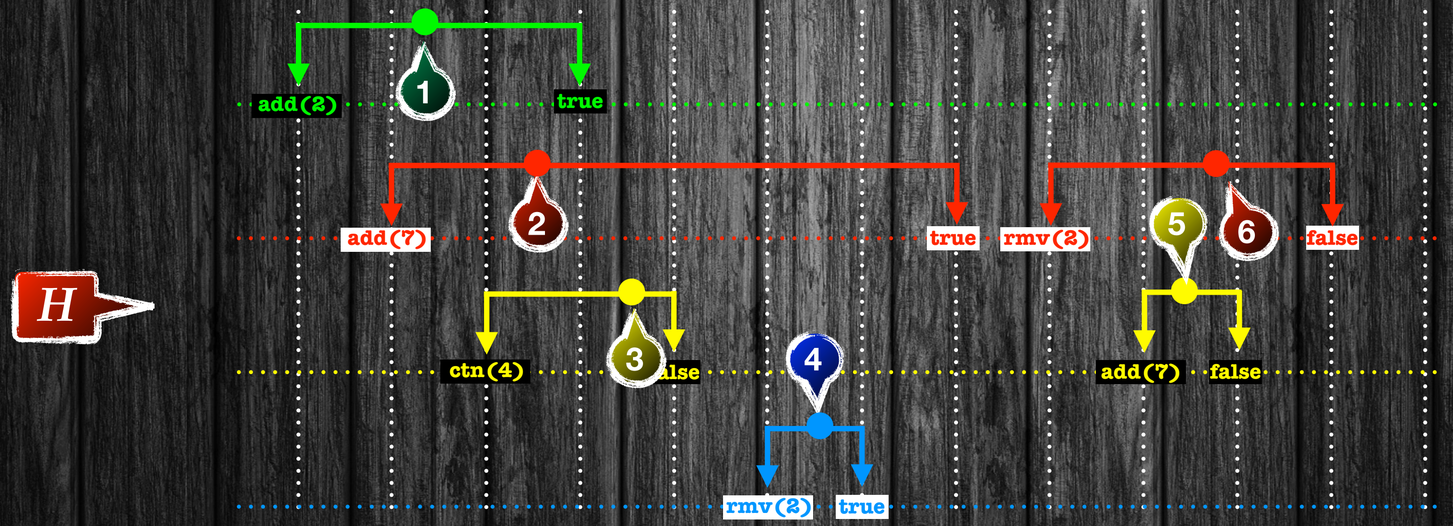


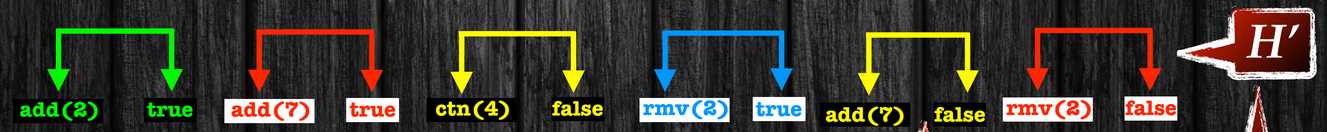


$H' \in \text{linearizations}(H)$ H

Sequential history







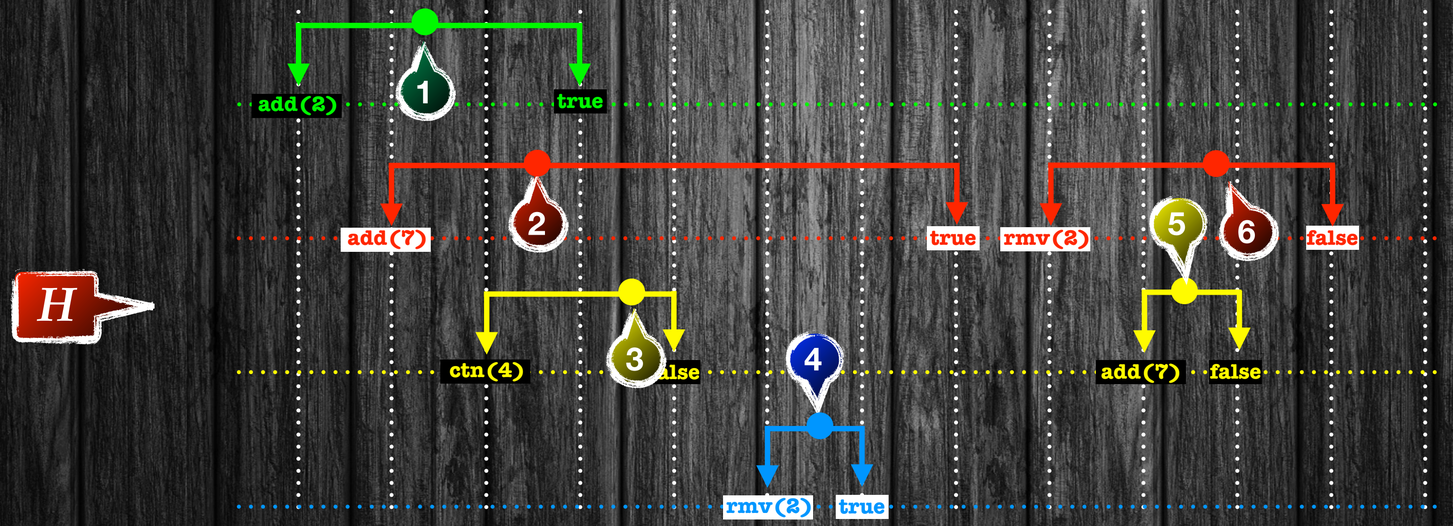
$H' \in$ linearizations(H)

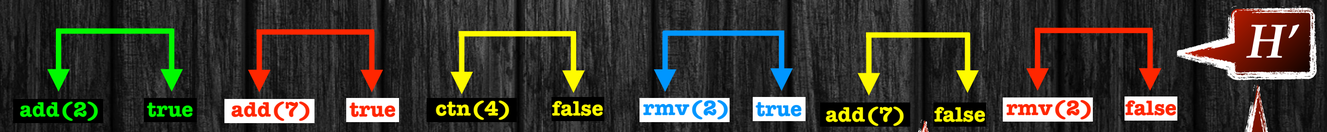
H

H'

Sequential history

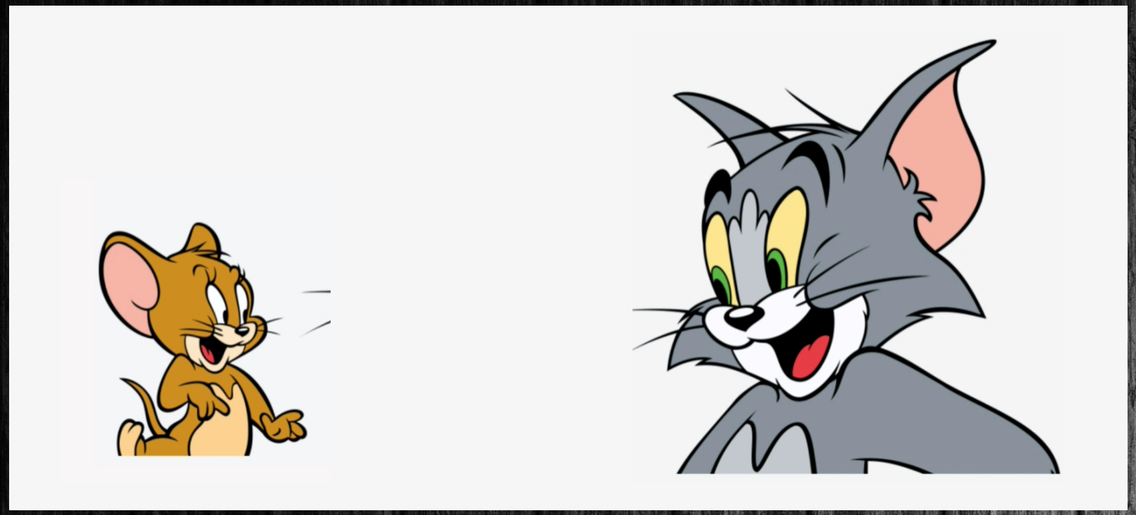






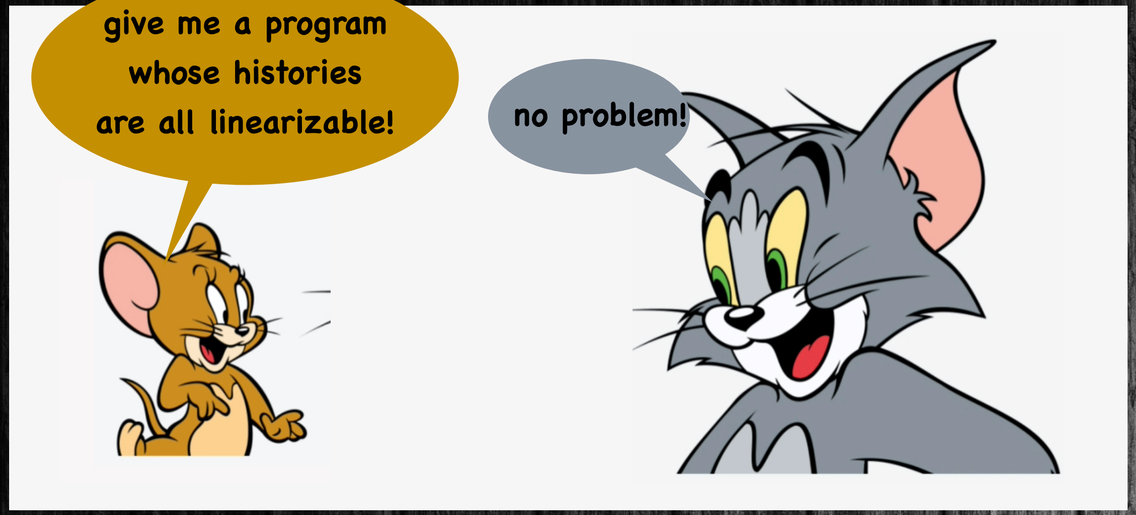
$H' \in \text{linearizations}(H)$ H

Sequential history




give me a program
whose histories
are all linearizable!

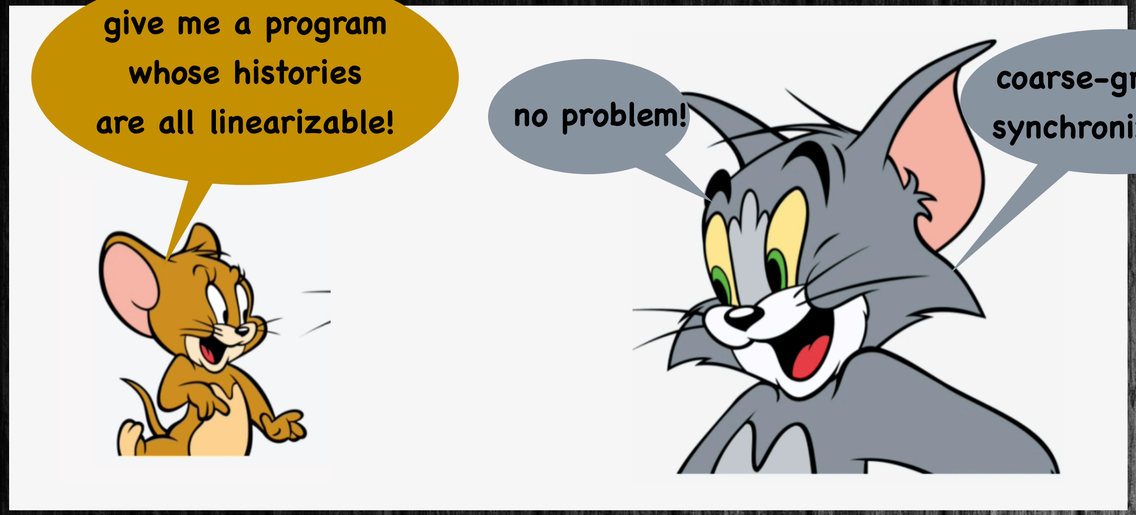

give me a program whose histories are all linearizable!
no problem!




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

```
remove(k):
Node p, c
Lock l
1  lock l;
2  p = H;
3  c = p.next
4  while (c.key < k)
5    p = c;
6    c = c.next;
7  if (c.key = k)
8    p.next = c.next
9    return true
10 else
11   return false
12 unlock l
```

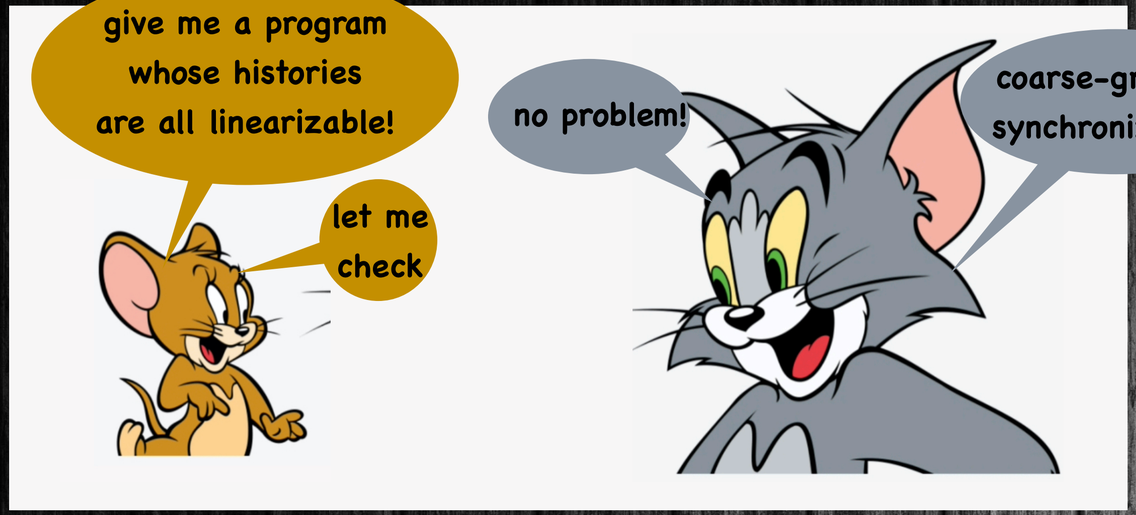
```
ctn(k):
Node p, c
Lock l
1  lock l;
2  p = H;
3  c = p.next
4  while (c.key < k)
5    p = c;
6    c = c.next;
7  if (c.key = k)
8    return true
9  else
10   return false
11 unlock l
```




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

```
remove(k):
Node p, c
Lock l
1  lock l;
2  p = H;
3  c = p.next
4  while (c.key < k)
5    p = c;
6    c = c.next;
7  if (c.key = k)
8    p.next = c.next
9    return true
10 else
11   return false
12 unlock l
```

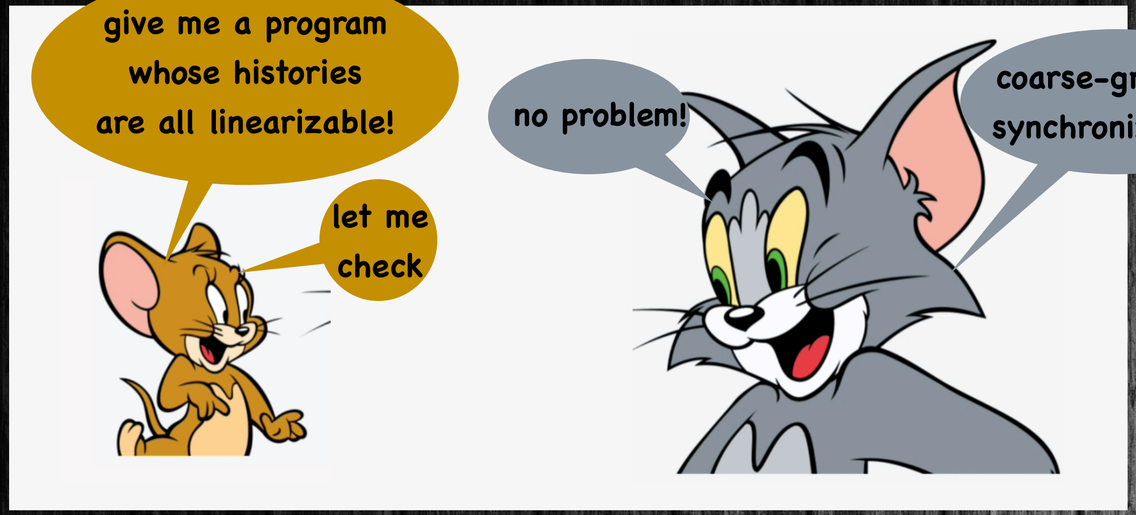
```
ctn(k):
Node p, c
Lock l
1  lock l;
2  p = H;
3  c = p.next
4  while (c.key < k)
5    p = c;
6    c = c.next;
7  if (c.key = k)
8    return true
9  else
10   return false
11 unlock l
```




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

```
remove(k):
Node p, c
Lock l
1  lock l;
2  p = H;
3  c = p.next
4  while (c.key < k)
5    p = c;
6    c = c.next;
7  if (c.key = k)
8    p.next = c.next
9    return true
10 else
11   return false
12 unlock l
```

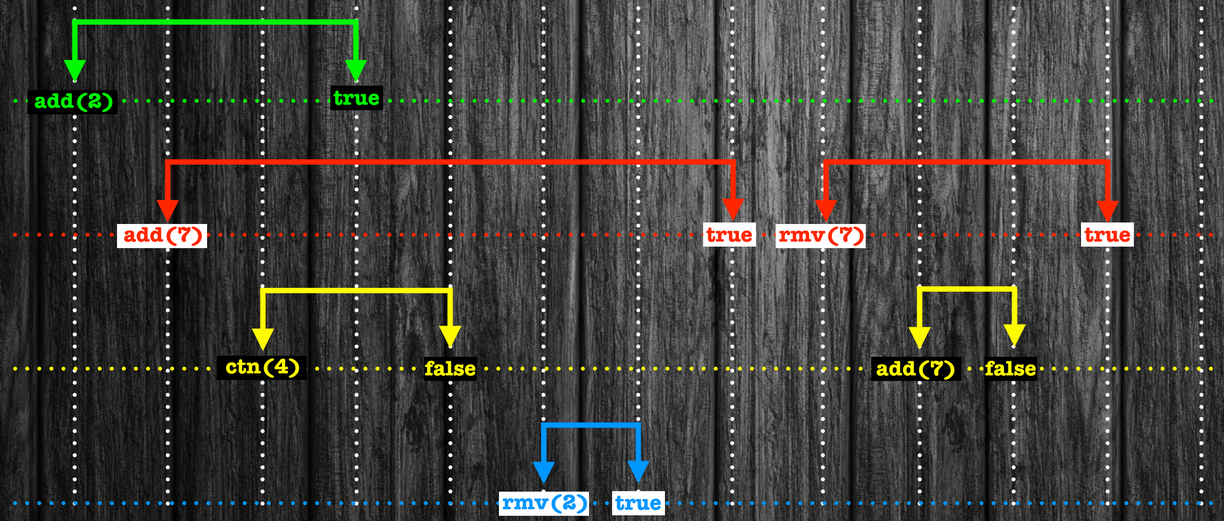
```
ctn(k):
Node p, c
Lock l
1  lock l;
2  p = H;
3  c = p.next
4  while (c.key < k)
5    p = c;
6    c = c.next;
7  if (c.key = k)
8    return true
9  else
10   return false
11 unlock l
```

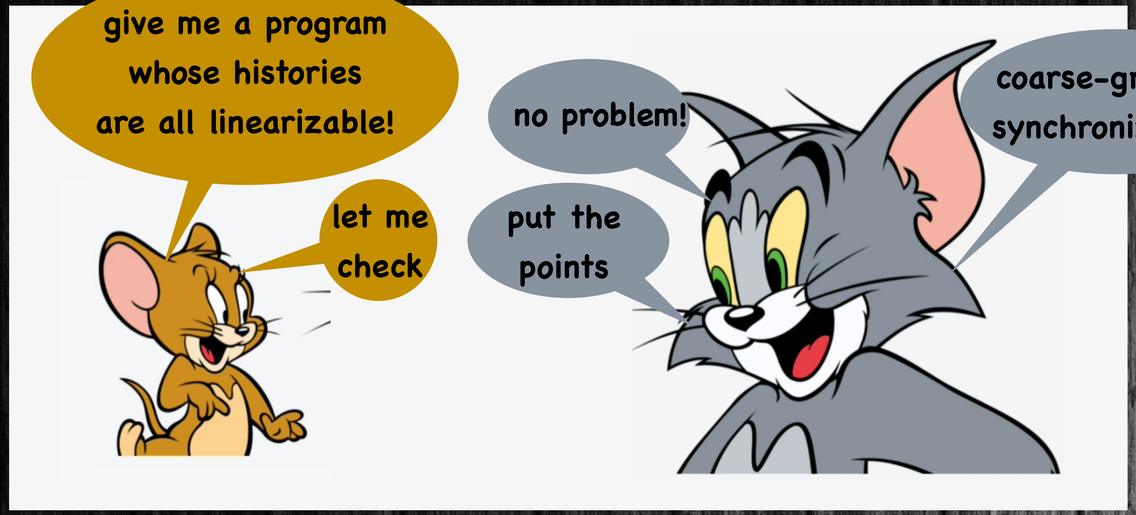



```
add(k):
Node p, c
Lock l
1  lock l;
2  p = H;
3  c = p.next
4  while (c.key < k)
5    p = c;
6    c = c.next;
7  if (c.key = k)
8    return false
9  else
10   n = new Node(k,-)
11   n.next = c
12   p.next = n
13   return true
14 unlock l
```

```
remove(k):
Node p, c
Lock l
1  lock l;
2  p = H;
3  c = p.next
4  while (c.key < k)
5    p = c;
6    c = c.next;
7  if (c.key = k)
8    p.next = c.next
9    return true
10 else
11   return false
12 unlock l
```

```
ctn(k):
Node p, c
Lock l
1  lock l;
2  p = H;
3  c = p.next
4  while (c.key < k)
5    p = c;
6    c = c.next;
7  if (c.key = k)
8    return true
9  else
10   return false
11 unlock l
```

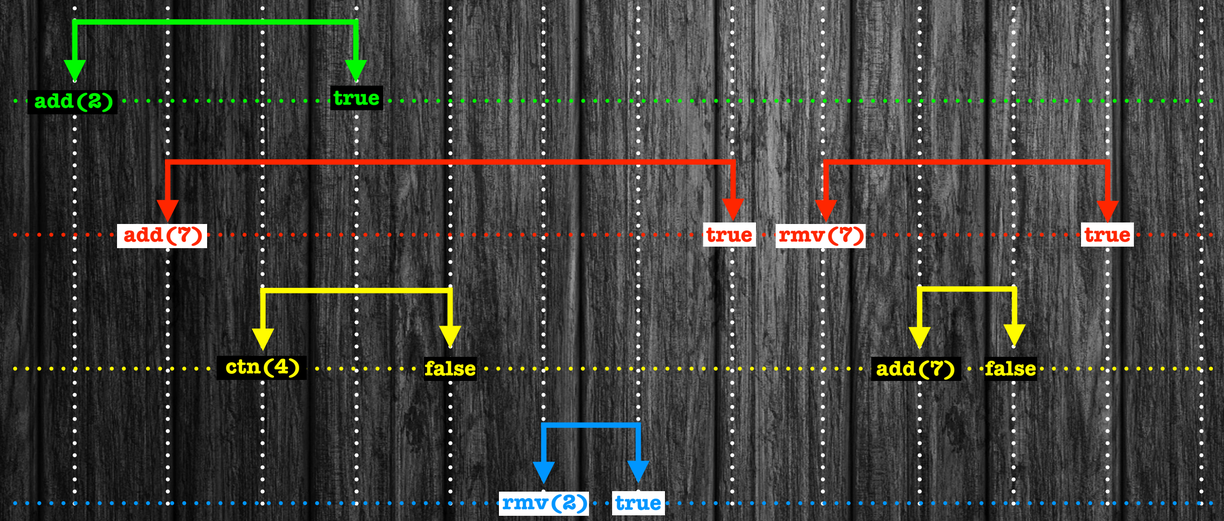


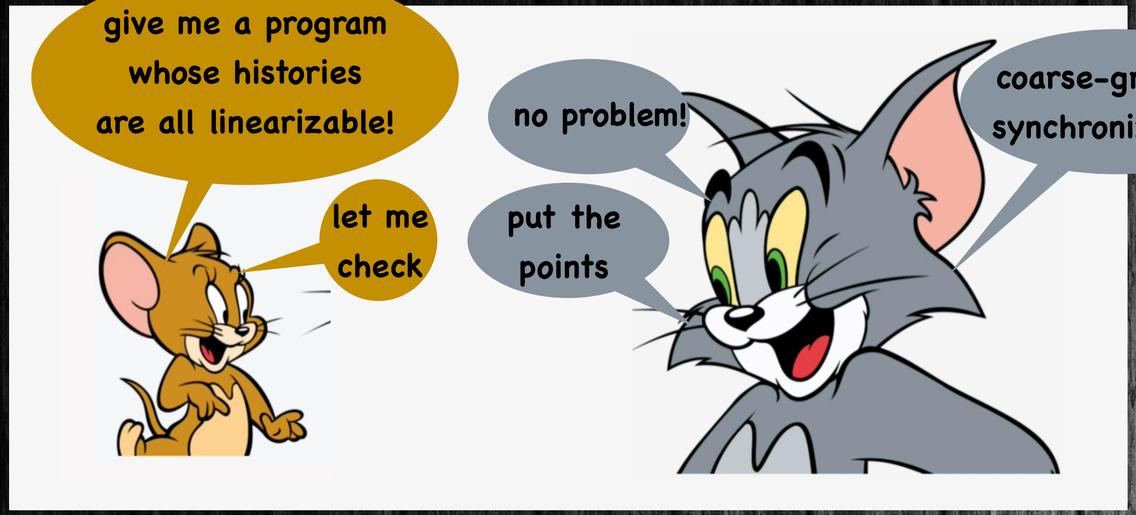



```
add(k):
Node p, c
Lock l
1  lock l;
2  p = H;
3  c = p.next
4  while (c.key < k)
5    p = c;
6    c = c.next;
7  if (c.key = k)
8    return false
9  else
10   n = new Node(k,-)
11   n.next = c
12   p.next = n
13   return true
14 unlock l
```

```
remove(k):
Node p, c
Lock l
1  lock l;
2  p = H;
3  c = p.next
4  while (c.key < k)
5    p = c;
6    c = c.next;
7  if (c.key = k)
8    p.next = c.next
9    return true
10 else
11   return false
12 unlock l
```

```
ctn(k):
Node p, c
Lock l
1  lock l;
2  p = H;
3  c = p.next
4  while (c.key < k)
5    p = c;
6    c = c.next;
7  if (c.key = k)
8    return true
9  else
10   return false
11 unlock l
```

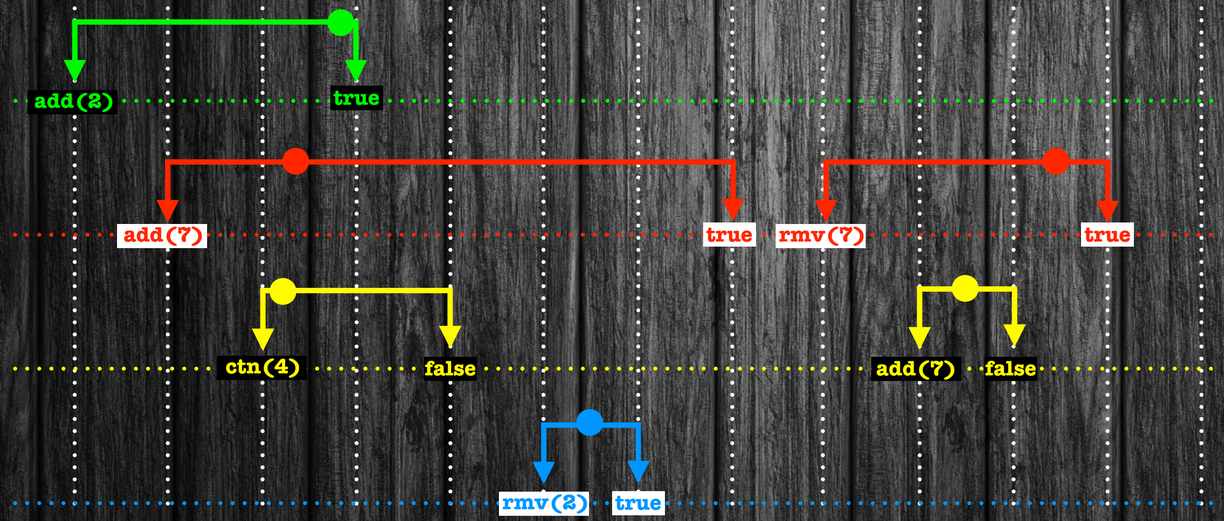


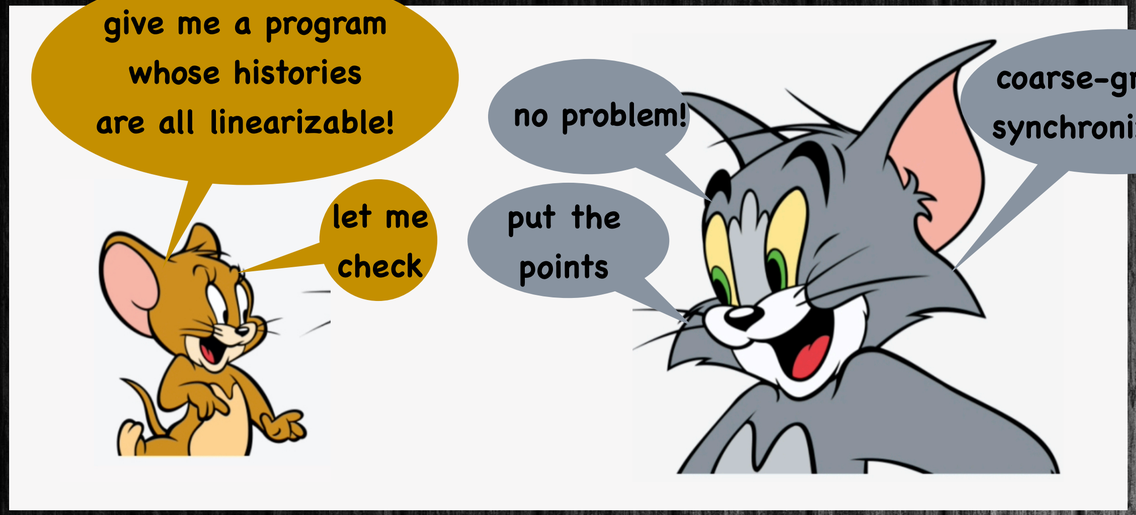



```
add(k):
Node p, c
Lock l
1  lock l;
2  p = H;
3  c = p.next
4  while (c.key < k)
5    p = c;
6    c = c.next;
7  if (c.key = k)
8    return false
9  else
10   n = new Node(k,-)
11   n.next = c
12   p.next = n
13   return true
14 unlock l
```

```
remove(k):
Node p, c
Lock l
1  lock l;
2  p = H;
3  c = p.next
4  while (c.key < k)
5    p = c;
6    c = c.next;
7  if (c.key = k)
8    p.next = c.next
9    return true
10 else
11   return false
12 unlock l
```

```
ctn(k):
Node p, c
Lock l
1  lock l;
2  p = H;
3  c = p.next
4  while (c.key < k)
5    p = c;
6    c = c.next;
7  if (c.key = k)
8    return true
9  else
10   return false
11 unlock l
```

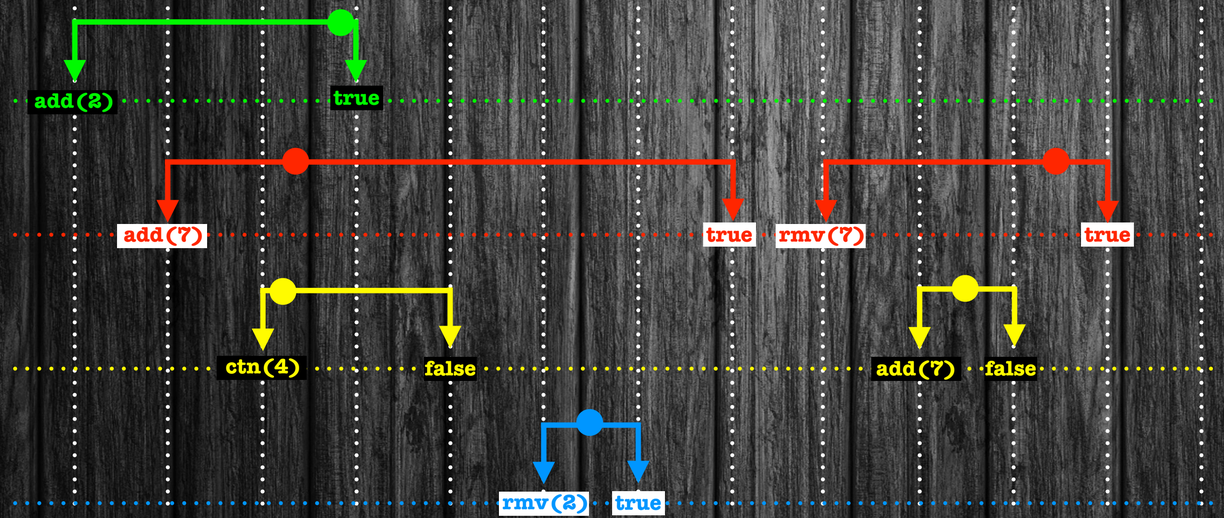




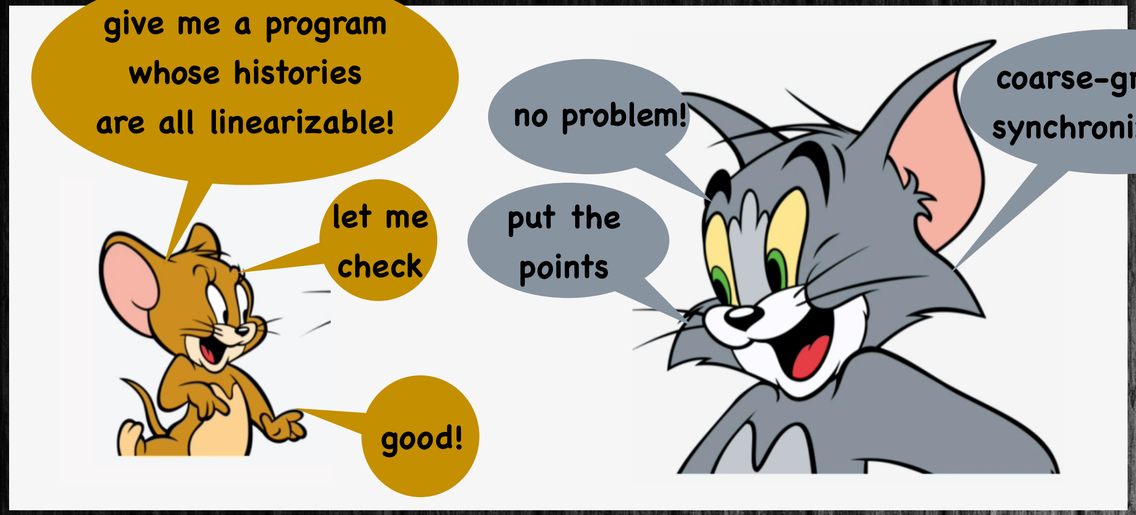

```
add(k):
Node p, c
Lock l
1  lock l;
2  p = H;
3  c = p.next
4  while (c.key < k)
5    p = c;
6    c = c.next;
7  if (c.key = k)
8    return false
9  else
10   n = new Node(k,-)
11   n.next = c
12   p.next = n
13   return true
14 unlock l
```

```
remove(k):
Node p, c
Lock l
1  lock l;
2  p = H;
3  c = p.next
4  while (c.key < k)
5    p = c;
6    c = c.next;
7  if (c.key = k)
8    p.next = c.next
9    return true
10 else
11   return false
12 unlock l
```

```
ctn(k):
Node p, c
Lock l
1  lock l;
2  p = H;
3  c = p.next
4  while (c.key < k)
5    p = c;
6    c = c.next;
7  if (c.key = k)
8    return true
9  else
10   return false
11 unlock l
```



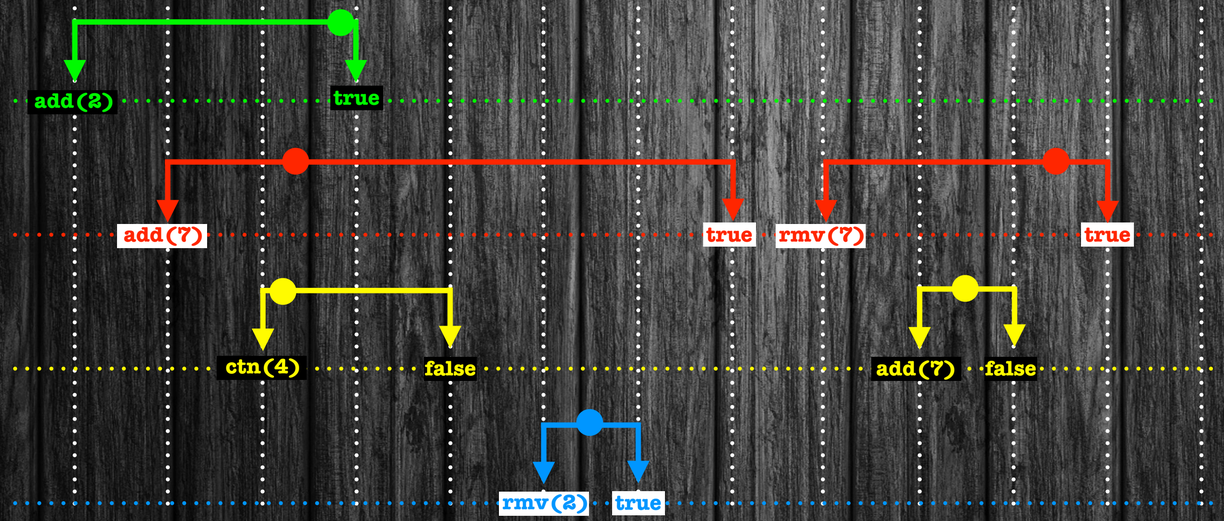
✔



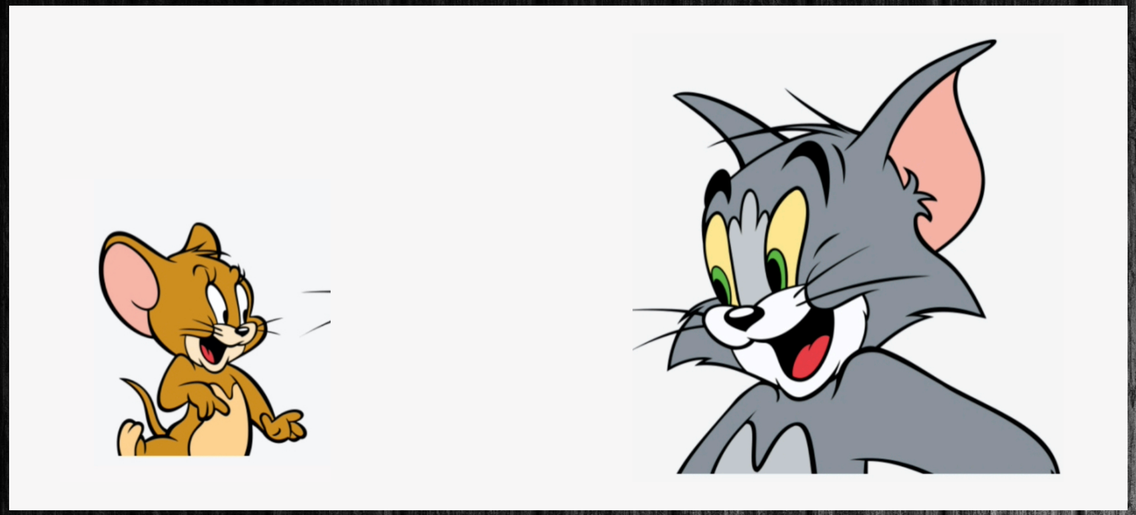

```
add(k):
Node p, c
Lock l
1  lock l;
2  p = H;
3  c = p.next
4  while (c.key < k)
5    p = c;
6    c = c.next;
7  if (c.key = k)
8    return false
9  else
10   n = new Node(k,-)
11   n.next = c
12   p.next = n
13   return true
14 unlock l
```

```
remove(k):
Node p, c
Lock l
1  lock l;
2  p = H;
3  c = p.next
4  while (c.key < k)
5    p = c;
6    c = c.next;
7  if (c.key = k)
8    p.next = c.next
9    return true
10 else
11   return false
12 unlock l
```

```
ctn(k):
Node p, c
Lock l
1  lock l;
2  p = H;
3  c = p.next
4  while (c.key < k)
5    p = c;
6    c = c.next;
7  if (c.key = k)
8    return true
9  else
10   return false
11 unlock l
```



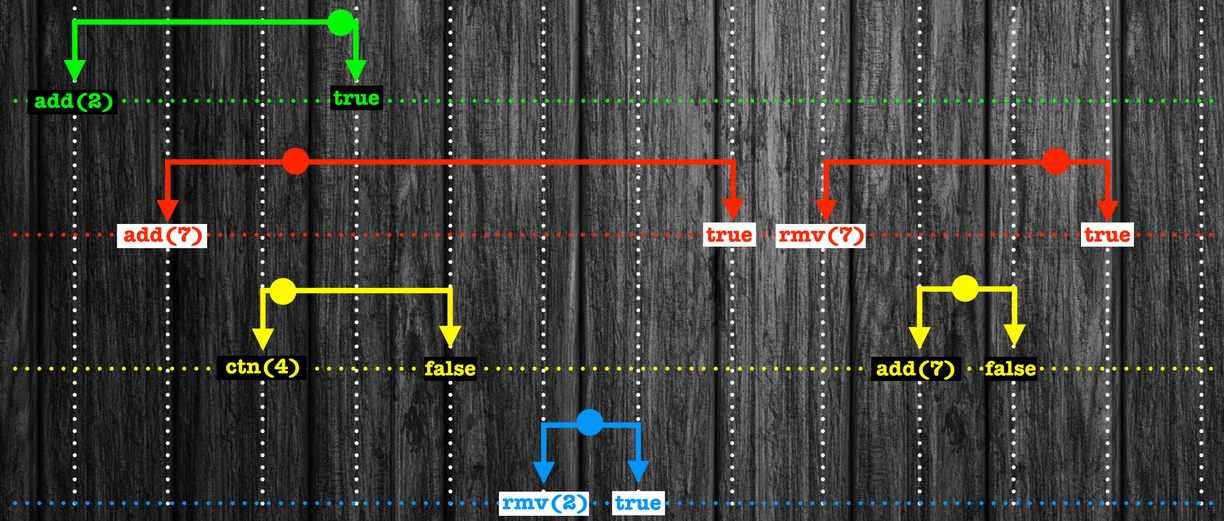
✔



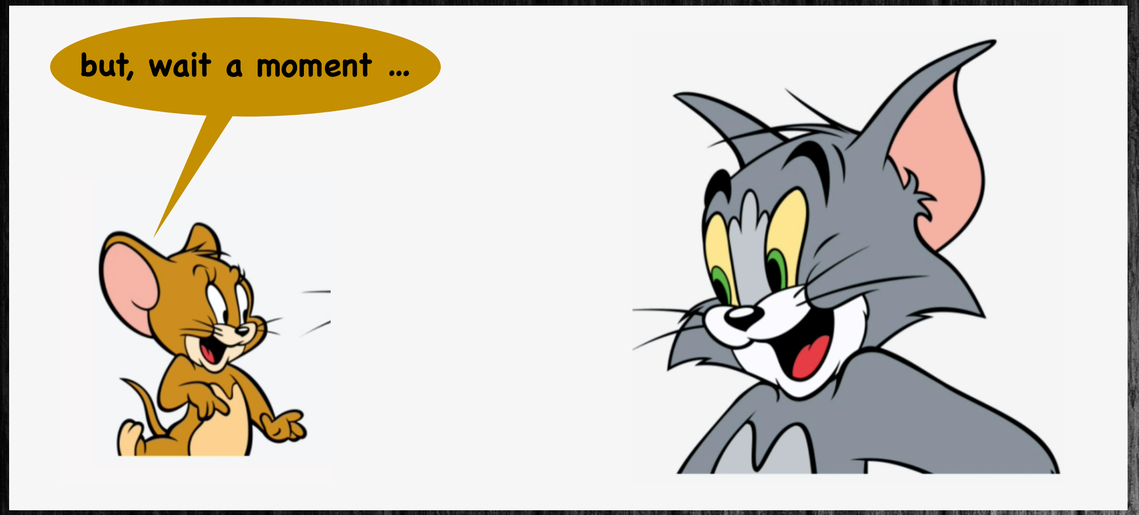
```
add(k):
Node p, c
Lock l
1  lock l;
2  p = H;
3  c = p.next
4  while (c.key < k)
5    p = c;
6    c = c.next;
7  if (c.key = k)
8    return false
9  else
10   n = new Node(k,-)
11   n.next = c
12   p.next = n
13   return true
14 unlock l
```

```
remove(k):
Node p, c
Lock l
1  lock l;
2  p = H;
3  c = p.next
4  while (c.key < k)
5    p = c;
6    c = c.next;
7  if (c.key = k)
8    p.next = c.next
9    return true
10 else
11   return false
12 unlock l
```

```
ctn(k):
Node p, c
Lock l
1  lock l;
2  p = H;
3  c = p.next
4  while (c.key < k)
5    p = c;
6    c = c.next;
7  if (c.key = k)
8    return true
9  else
10   return false
11 unlock l
```



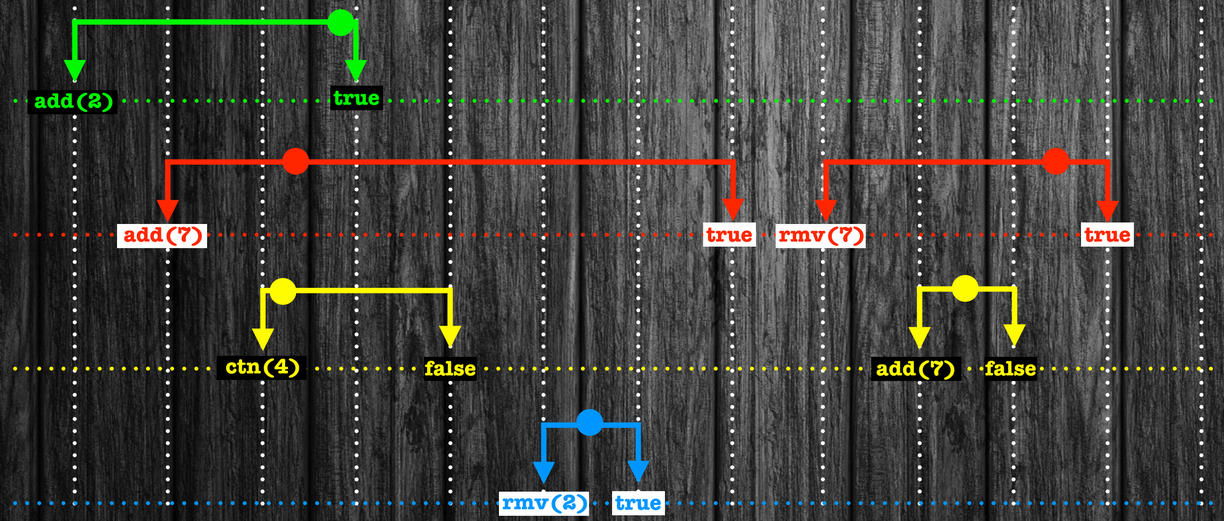
✔



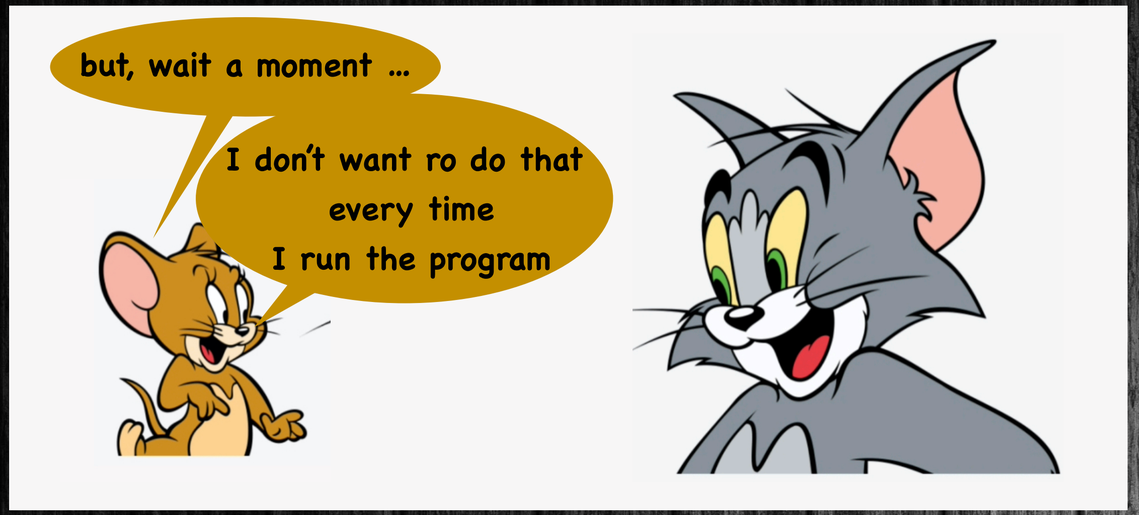

```
add(k):
Node p, c
Lock l
1  lock l;
2  p = H;
3  c = p.next
4  while (c.key < k)
5    p = c;
6    c = c.next;
7  if (c.key = k)
8    return false
9  else
10   n = new Node(k,-)
11   n.next = c
12   p.next = n
13   return true
14 unlock l
```

```
remove(k):
Node p, c
Lock l
1  lock l;
2  p = H;
3  c = p.next
4  while (c.key < k)
5    p = c;
6    c = c.next;
7  if (c.key = k)
8    p.next = c.next
9    return true
10 else
11   return false
12 unlock l
```

```
ctn(k):
Node p, c
Lock l
1  lock l;
2  p = H;
3  c = p.next
4  while (c.key < k)
5    p = c;
6    c = c.next;
7  if (c.key = k)
8    return true
9  else
10   return false
11 unlock l
```



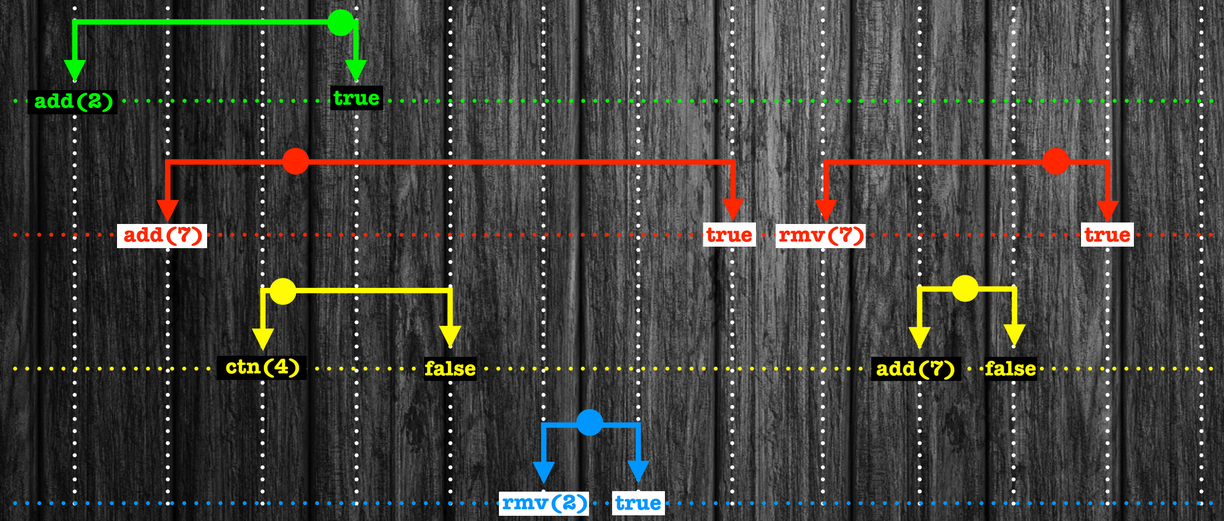
✔



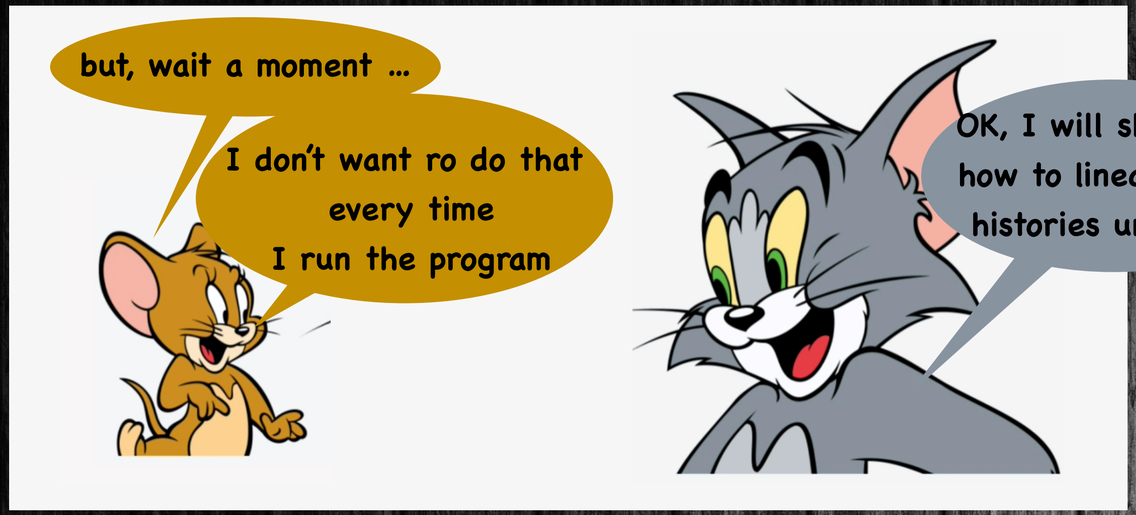

```
add(k):
Node p, c
Lock l
1  lock l;
2  p = H;
3  c = p.next
4  while (c.key < k)
5    p = c;
6    c = c.next;
7  if (c.key = k)
8    return false
9  else
10   n = new Node(k,-)
11   n.next = c
12   p.next = n
13   return true
14 unlock l
```

```
remove(k):
Node p, c
Lock l
1  lock l;
2  p = H;
3  c = p.next
4  while (c.key < k)
5    p = c;
6    c = c.next;
7  if (c.key = k)
8    p.next = c.next
9    return true
10 else
11   return false
12 unlock l
```

```
ctn(k):
Node p, c
Lock l
1  lock l;
2  p = H;
3  c = p.next
4  while (c.key < k)
5    p = c;
6    c = c.next;
7  if (c.key = k)
8    return true
9  else
10   return false
11 unlock l
```



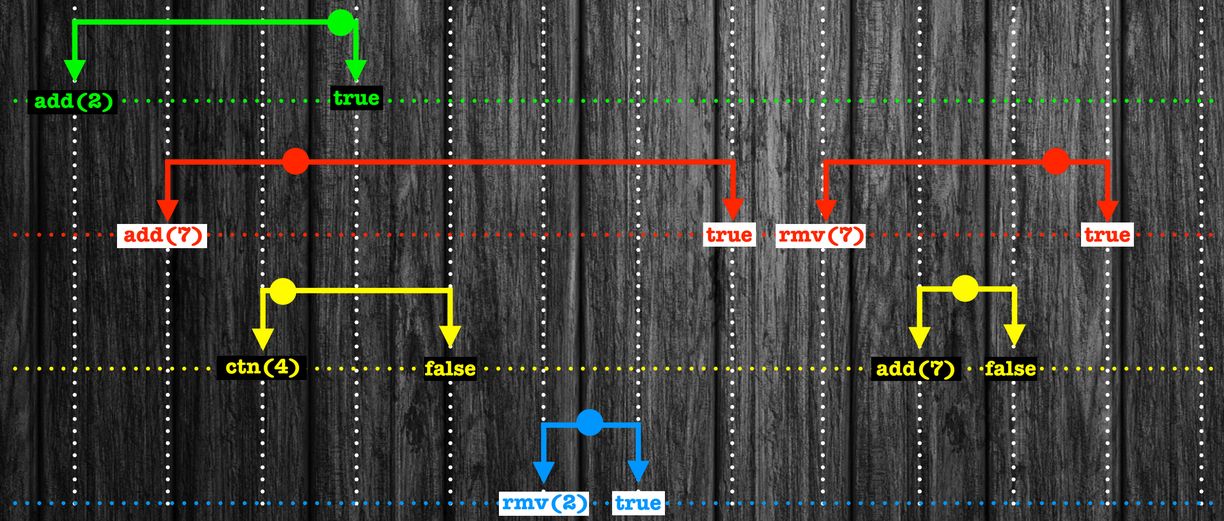
✔



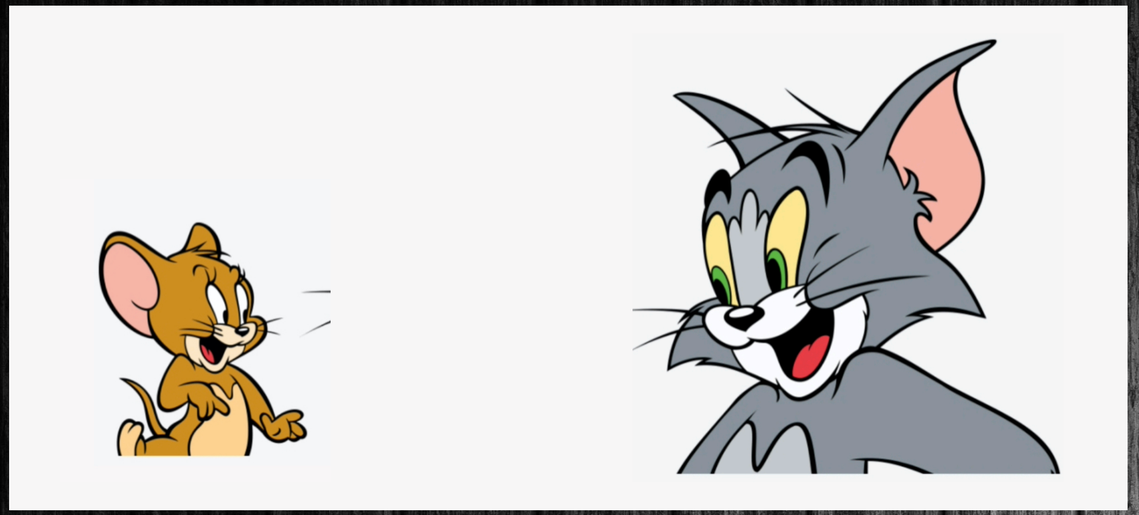

```
add(k):
Node p, c
Lock l
1  lock l;
2  p = H;
3  c = p.next
4  while (c.key < k)
5    p = c;
6    c = c.next;
7  if (c.key = k)
8    return false
9  else
10   n = new Node(k,-)
11   n.next = c
12   p.next = n
13   return true
14 unlock l
```

```
remove(k):
Node p, c
Lock l
1  lock l;
2  p = H;
3  c = p.next
4  while (c.key < k)
5    p = c;
6    c = c.next;
7  if (c.key = k)
8    p.next = c.next
9    return true
10 else
11   return false
12 unlock l
```

```
ctn(k):
Node p, c
Lock l
1  lock l;
2  p = H;
3  c = p.next
4  while (c.key < k)
5    p = c;
6    c = c.next;
7  if (c.key = k)
8    return true
9  else
10   return false
11 unlock l
```



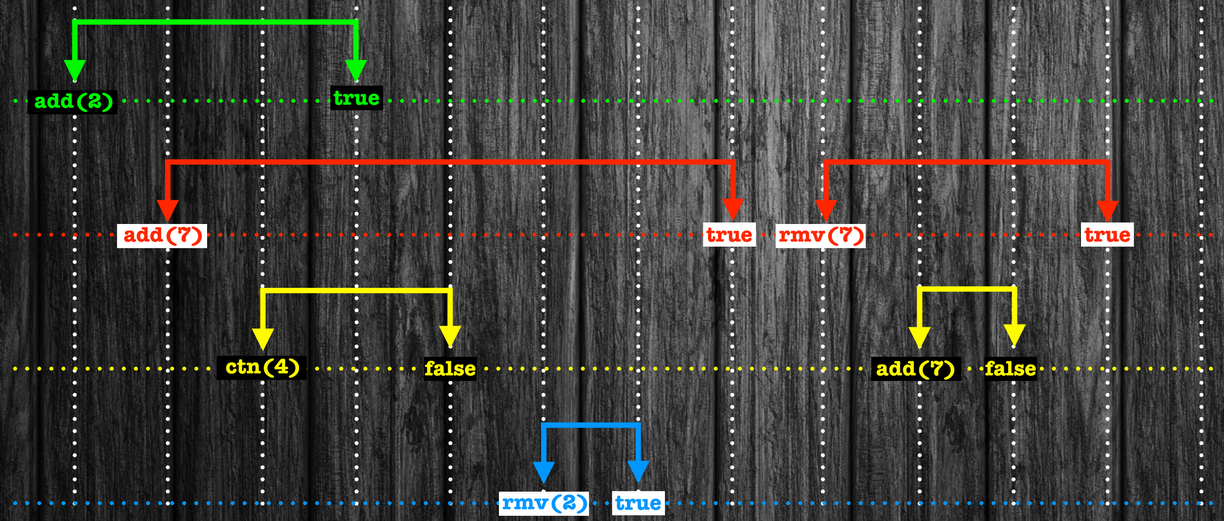
✔

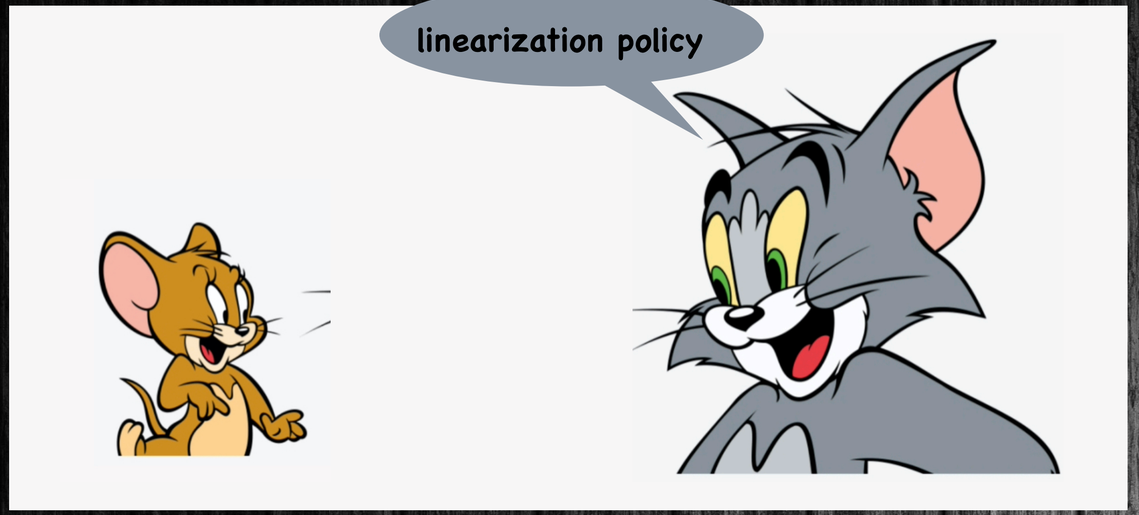


```
add(k):
Node p, c
Lock l
1  lock l;
2  p = H;
3  c = p.next
4  while (c.key < k)
5    p = c;
6    c = c.next;
7  if (c.key = k)
8    return false
9  else
10   n = new Node(k,-)
11   n.next = c
12   p.next = n
13   return true
14 unlock l
```

```
remove(k):
Node p, c
Lock l
1  lock l;
2  p = H;
3  c = p.next
4  while (c.key < k)
5    p = c;
6    c = c.next;
7  if (c.key = k)
8    p.next = c.next
9    return true
10 else
11   return false
12 unlock l
```

```
ctn(k):
Node p, c
Lock l
1  lock l;
2  p = H;
3  c = p.next
4  while (c.key < k)
5    p = c;
6    c = c.next;
7  if (c.key = k)
8    return true
9  else
10   return false
11 unlock l
```

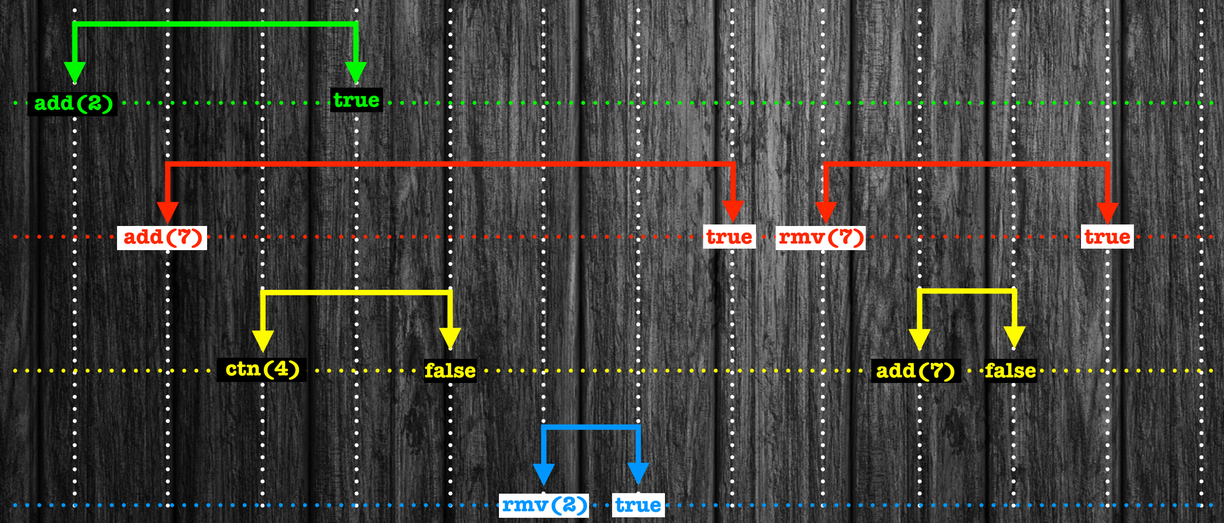


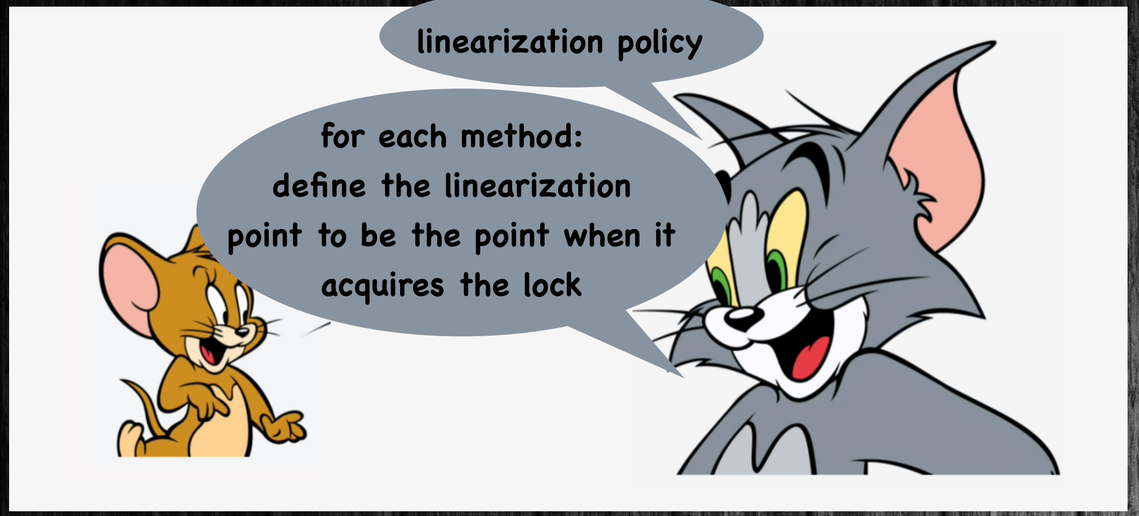



```
add(k):
Node p, c
Lock l
1  lock l;
2  p = H;
3  c = p.next
4  while (c.key < k)
5    p = c;
6    c = c.next;
7  if (c.key = k)
8    return false
9  else
10   n = new Node(k,-)
11   n.next = c
12   p.next = n
13   return true
14 unlock l
```

```
remove(k):
Node p, c
Lock l
1  lock l;
2  p = H;
3  c = p.next
4  while (c.key < k)
5    p = c;
6    c = c.next;
7  if (c.key = k)
8    p.next = c.next
9    return true
10 else
11   return false
12 unlock l
```

```
ctn(k):
Node p, c
Lock l
1  lock l;
2  p = H;
3  c = p.next
4  while (c.key < k)
5    p = c;
6    c = c.next;
7  if (c.key = k)
8    return true
9  else
10   return false
11 unlock l
```

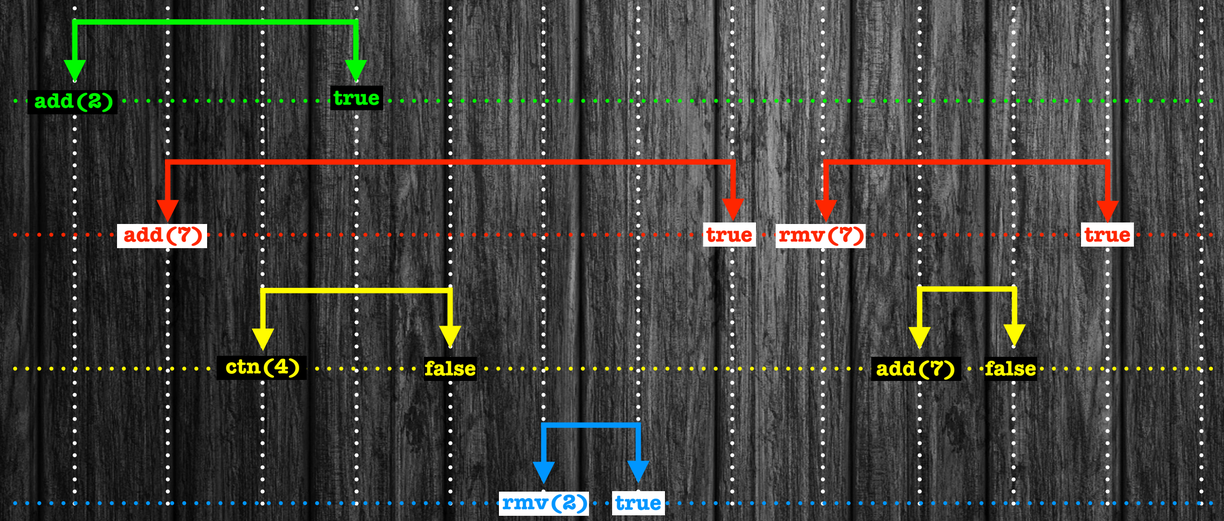


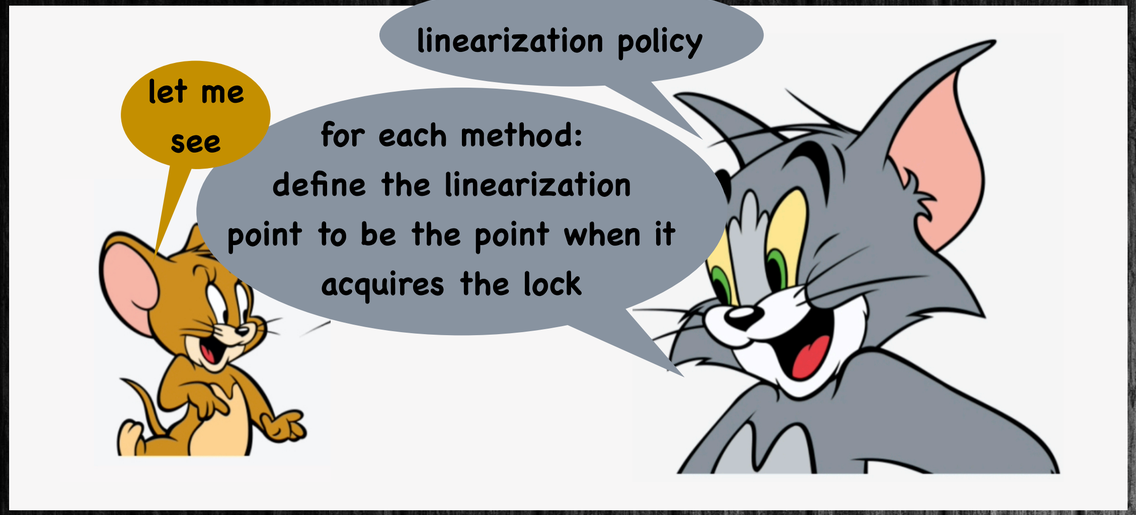



```
add(k):
Node p, c
Lock l
1  lock l;
2  p = H;
3  c = p.next
4  while (c.key < k)
5    p = c;
6    c = c.next;
7  if (c.key = k)
8    return false
9  else
10   n = new Node(k,-)
11   n.next = c
12   p.next = n
13   return true
14 unlock l
```

```
remove(k):
Node p, c
Lock l
1  lock l;
2  p = H;
3  c = p.next
4  while (c.key < k)
5    p = c;
6    c = c.next;
7  if (c.key = k)
8    p.next = c.next
9    return true
10 else
11   return false
12 unlock l
```

```
ctn(k):
Node p, c
Lock l
1  lock l;
2  p = H;
3  c = p.next
4  while (c.key < k)
5    p = c;
6    c = c.next;
7  if (c.key = k)
8    return true
9  else
10   return false
11 unlock l
```

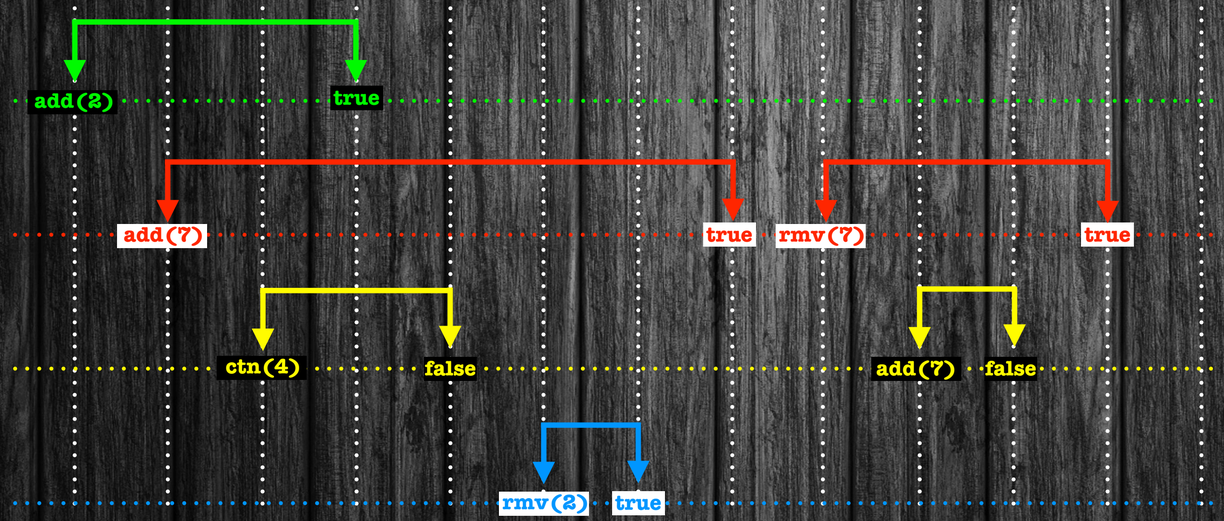


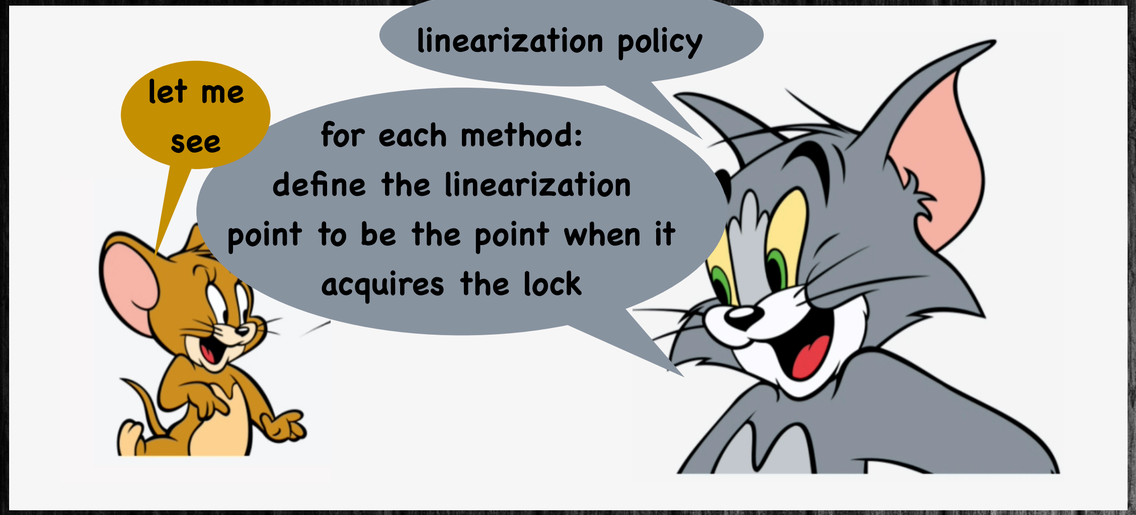



```
add(k):
Node p, c
Lock l
1  lock l;
2  p = H;
3  c = p.next
4  while (c.key < k)
5    p = c;
6    c = c.next;
7  if (c.key = k)
8    return false
9  else
10   n = new Node(k,-)
11   n.next = c
12   p.next = n
13   return true
14 unlock l
```

```
remove(k):
Node p, c
Lock l
1  lock l;
2  p = H;
3  c = p.next
4  while (c.key < k)
5    p = c;
6    c = c.next;
7  if (c.key = k)
8    p.next = c.next
9    return true
10 else
11   return false
12 unlock l
```

```
ctn(k):
Node p, c
Lock l
1  lock l;
2  p = H;
3  c = p.next
4  while (c.key < k)
5    p = c;
6    c = c.next;
7  if (c.key = k)
8    return true
9  else
10   return false
11 unlock l
```

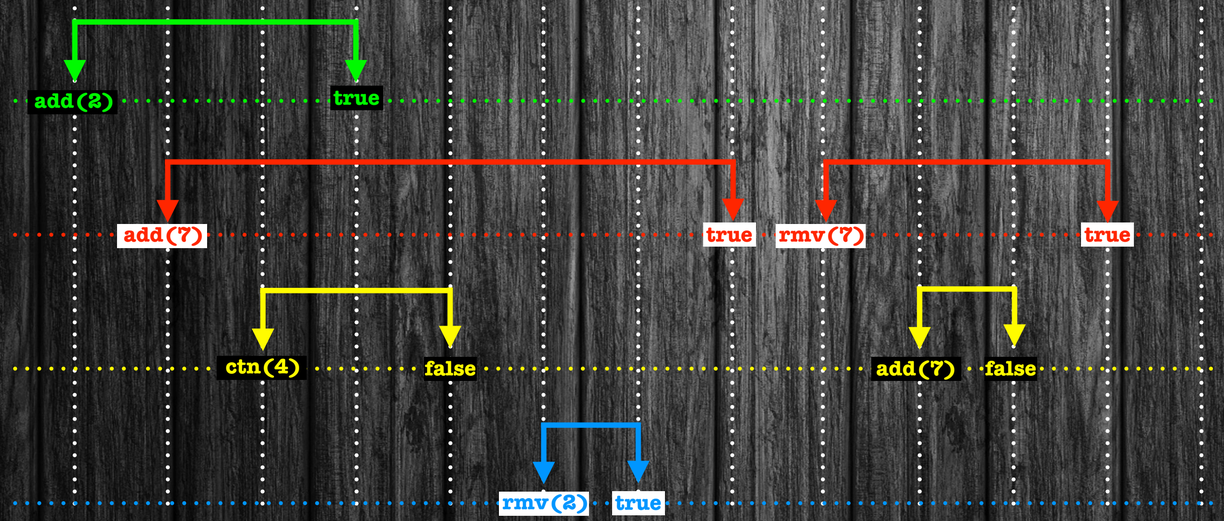


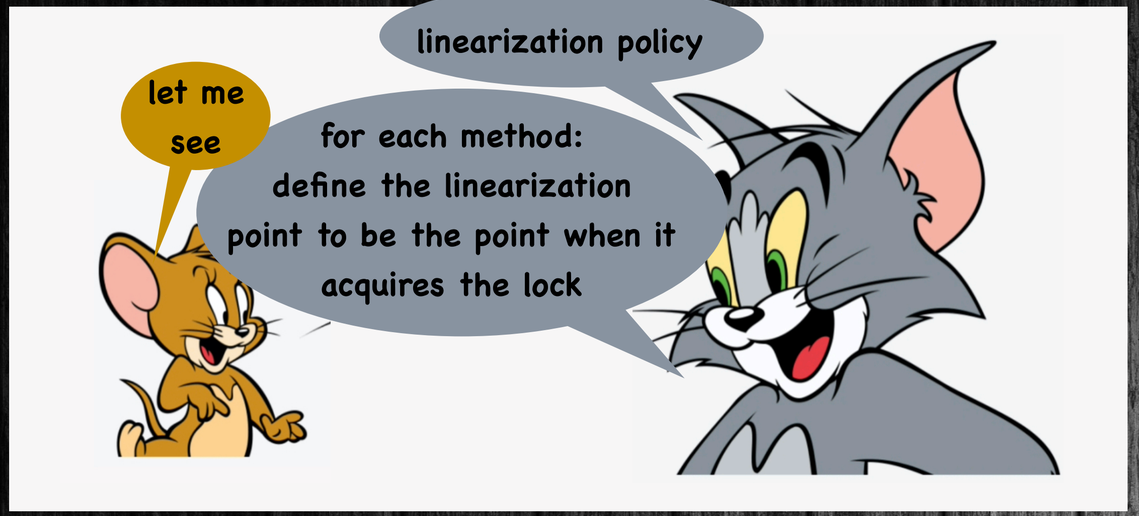



```
add(k):
Node p, c
Lock l
1  lock l;
2  p = H;
3  c = p.next
4  while (c.key < k)
5    p = c;
6    c = c.next;
7  if (c.key = k)
8    return false
9  else
10   n = new Node(k,-)
11   n.next = c
12   p.next = n
13   return true
14 unlock l
```

```
remove(k):
Node p, c
Lock l
1  lock l;
2  p = H;
3  c = p.next
4  while (c.key < k)
5    p = c;
6    c = c.next;
7  if (c.key = k)
8    p.next = c.next
9    return true
10 else
11   return false
12 unlock l
```

```
ctn(k):
Node p, c
Lock l
1  lock l;
2  p = H;
3  c = p.next
4  while (c.key < k)
5    p = c;
6    c = c.next;
7  if (c.key = k)
8    return true
9  else
10   return false
11 unlock l
```

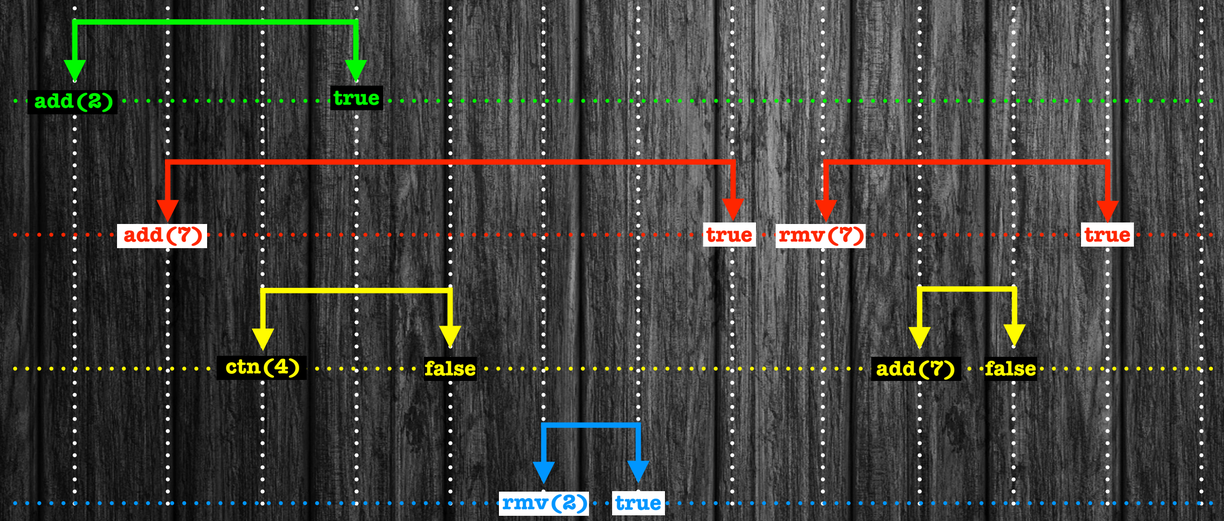


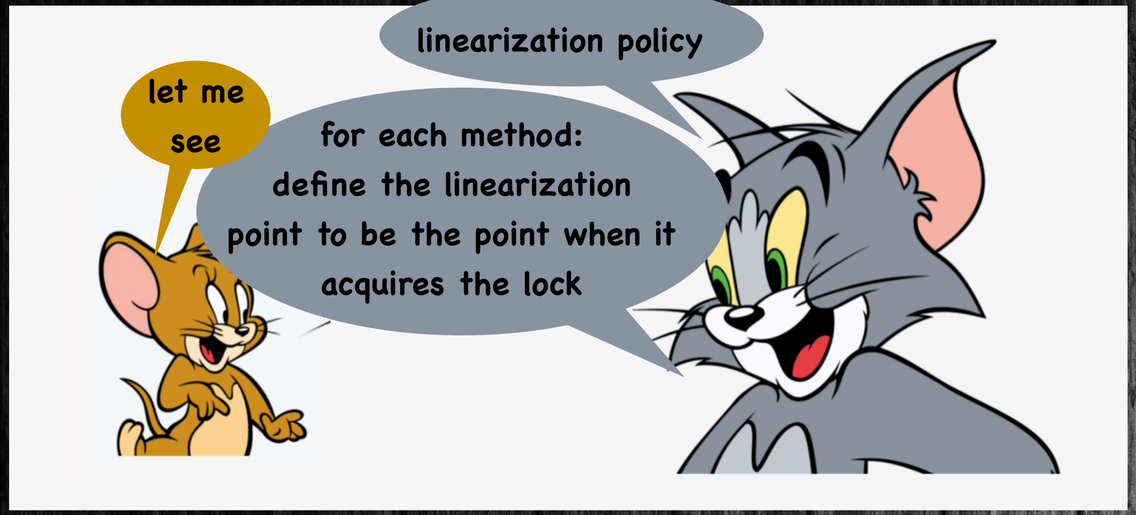



```
add(k):
Node p, c
Lock l
1  lock l;
2  p = H;
3  c = p.next
4  while (c.key < k)
5    p = c;
6    c = c.next;
7  if (c.key = k)
8    return false
9  else
10   n = new Node(k,-)
11   n.next = c
12   p.next = n
13   return true
14 unlock l
```

```
remove(k):
Node p, c
Lock l
1  lock l;
2  p = H;
3  c = p.next
4  while (c.key < k)
5    p = c;
6    c = c.next;
7  if (c.key = k)
8    p.next = c.next
9    return true
10 else
11   return false
12 unlock l
```

```
ctn(k):
Node p, c
Lock l
1  lock l;
2  p = H;
3  c = p.next
4  while (c.key < k)
5    p = c;
6    c = c.next;
7  if (c.key = k)
8    return true
9  else
10   return false
11 unlock l
```

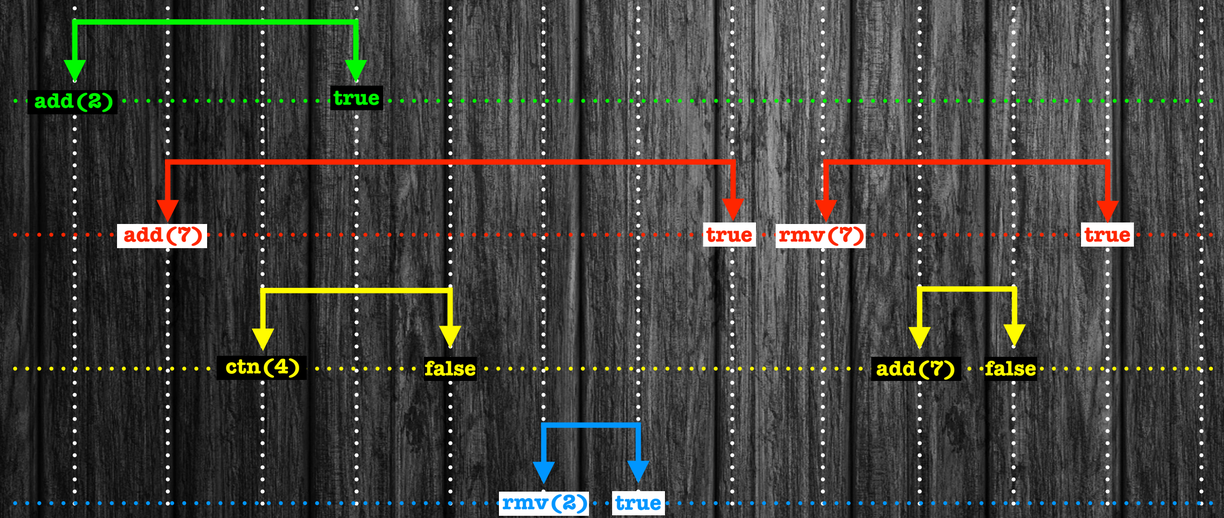


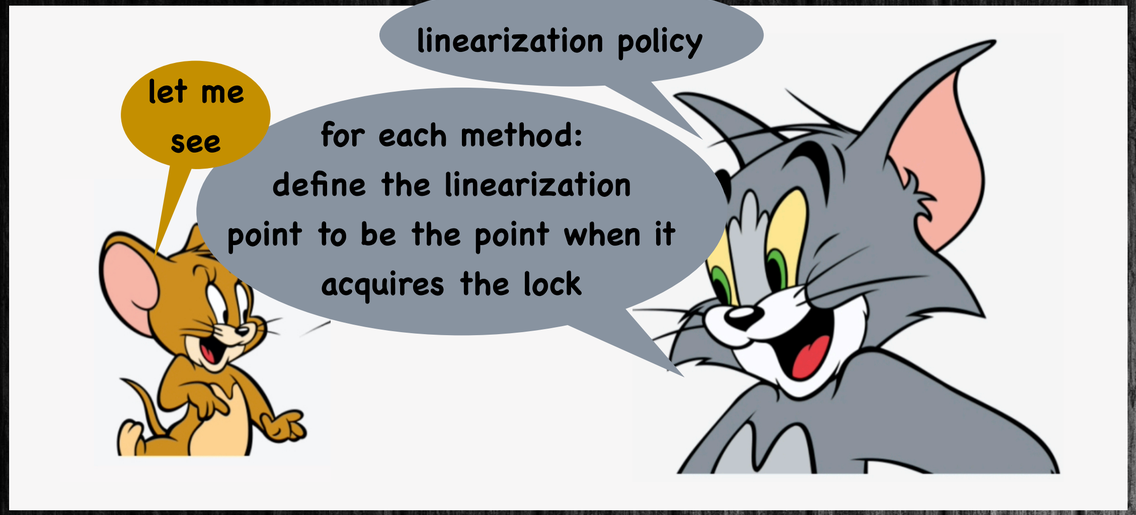



```
add(k):
Node p, c
Lock l
1  lock l;
2  p = H;
3  c = p.next
4  while (c.key < k)
5    p = c;
6    c = c.next;
7  if (c.key = k)
8    return false
9  else
10   n = new Node(k,-)
11   n.next = c
12   p.next = n
13   return true
14 unlock l
```

```
remove(k):
Node p, c
Lock l
1  lock l;
2  p = H;
3  c = p.next
4  while (c.key < k)
5    p = c;
6    c = c.next;
7  if (c.key = k)
8    p.next = c.next
9    return true
10 else
11   return false
12 unlock l
```

```
ctn(k):
Node p, c
Lock l
1  lock l;
2  p = H;
3  c = p.next
4  while (c.key < k)
5    p = c;
6    c = c.next;
7  if (c.key = k)
8    return true
9  else
10   return false
11 unlock l
```

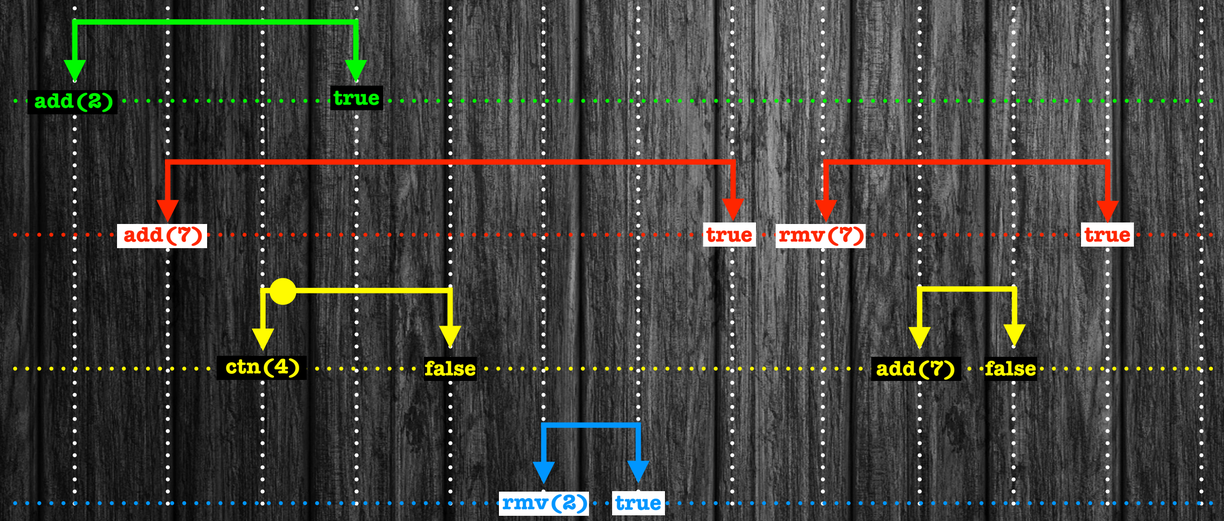


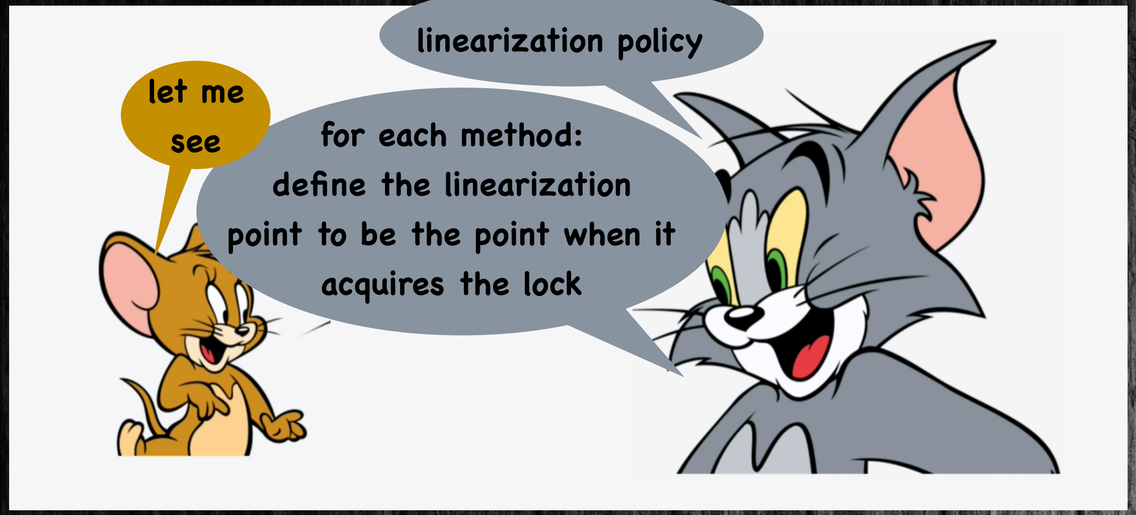



```
add(k):
Node p, c
Lock l
1  lock l;
2  p = H;
3  c = p.next
4  while (c.key < k)
5    p = c;
6    c = c.next;
7  if (c.key = k)
8    return false
9  else
10   n = new Node(k,-)
11   n.next = c
12   p.next = n
13   return true
14 unlock l
```

```
remove(k):
Node p, c
Lock l
1  lock l;
2  p = H;
3  c = p.next
4  while (c.key < k)
5    p = c;
6    c = c.next;
7  if (c.key = k)
8    p.next = c.next
9    return true
10 else
11   return false
12 unlock l
```

```
ctn(k):
Node p, c
Lock l
1  lock l;
2  p = H;
3  c = p.next
4  while (c.key < k)
5    p = c;
6    c = c.next;
7  if (c.key = k)
8    return true
9  else
10   return false
11 unlock l
```

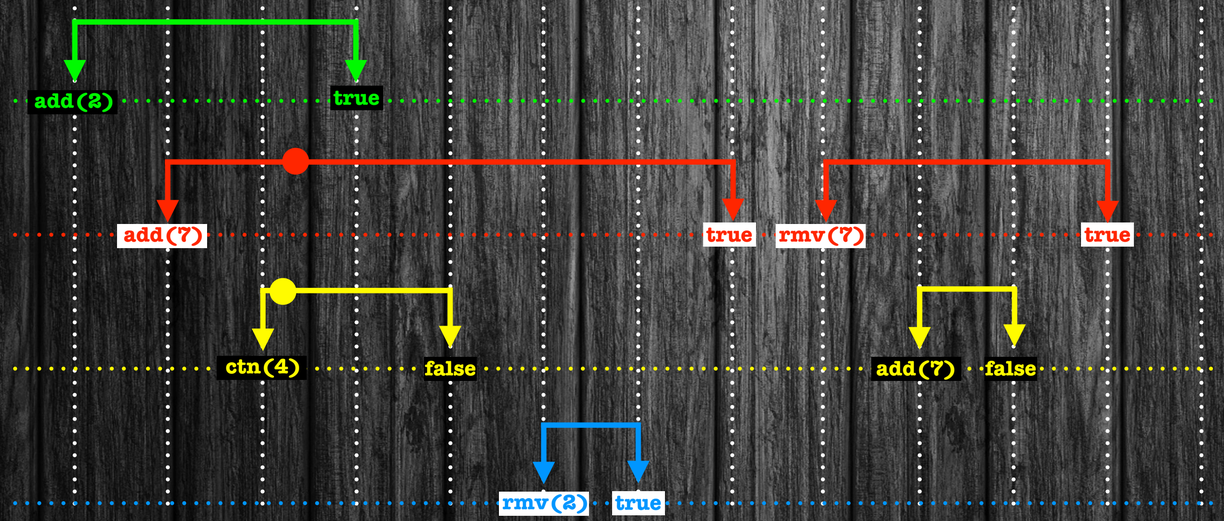


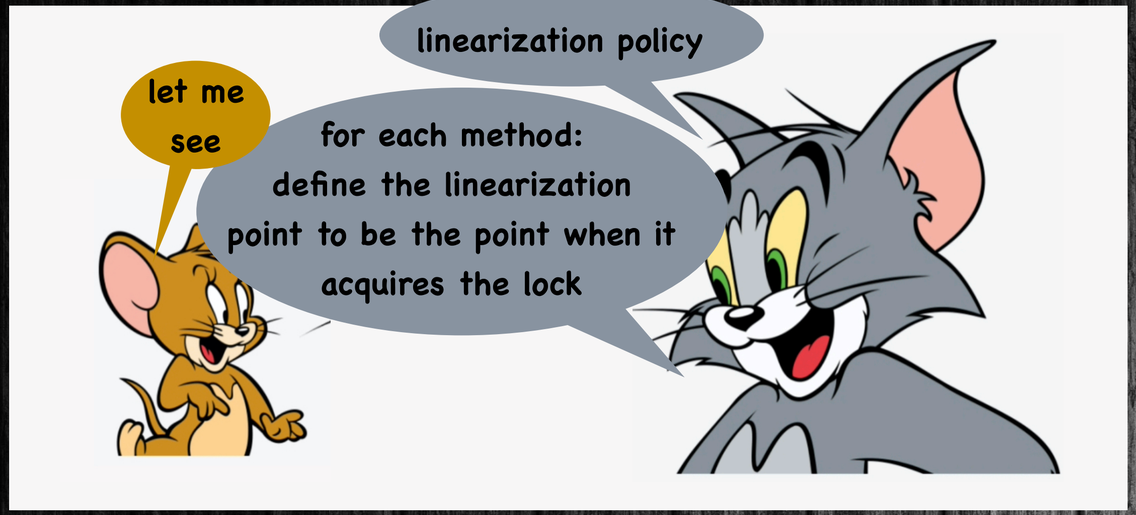



```
add(k):
Node p, c
Lock l
1  lock l;
2  p = H;
3  c = p.next
4  while (c.key < k)
5    p = c;
6    c = c.next;
7  if (c.key = k)
8    return false
9  else
10   n = new Node(k,-)
11   n.next = c
12   p.next = n
13   return true
14 unlock l
```

```
remove(k):
Node p, c
Lock l
1  lock l;
2  p = H;
3  c = p.next
4  while (c.key < k)
5    p = c;
6    c = c.next;
7  if (c.key = k)
8    p.next = c.next
9    return true
10 else
11   return false
12 unlock l
```

```
ctn(k):
Node p, c
Lock l
1  lock l;
2  p = H;
3  c = p.next
4  while (c.key < k)
5    p = c;
6    c = c.next;
7  if (c.key = k)
8    return true
9  else
10   return false
11 unlock l
```

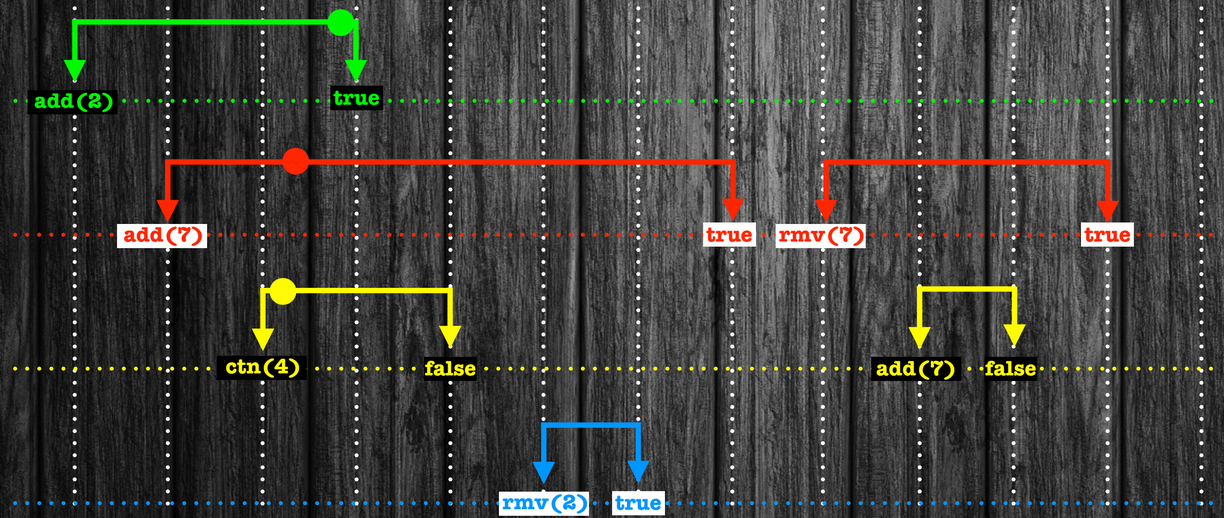


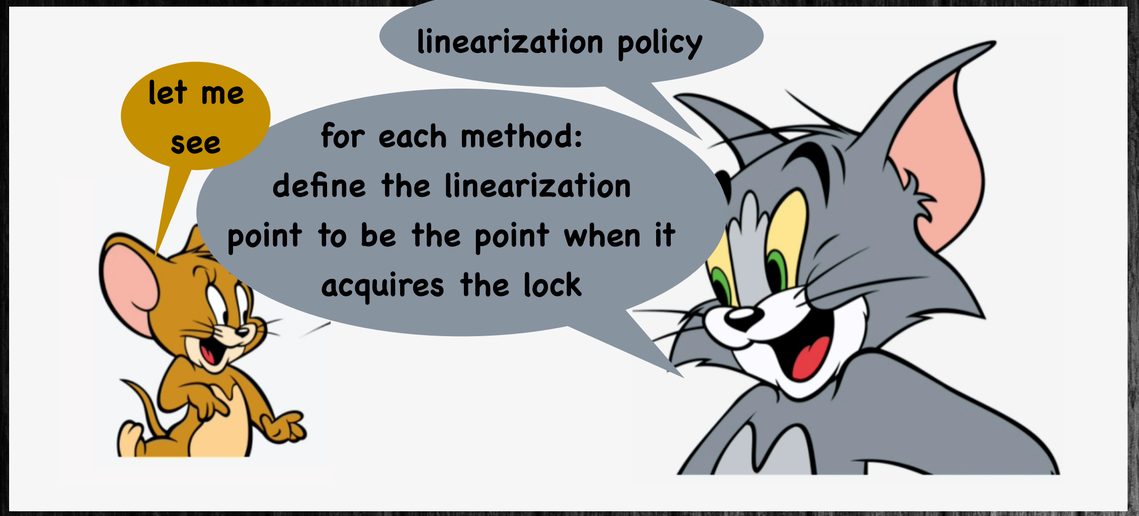



```
add(k):
Node p, c
Lock l
1  lock l;
2  p = H;
3  c = p.next
4  while (c.key < k)
5    p = c;
6    c = c.next;
7  if (c.key = k)
8    return false
9  else
10   n = new Node(k,-)
11   n.next = c
12   p.next = n
13   return true
14 unlock l
```

```
remove(k):
Node p, c
Lock l
1  lock l;
2  p = H;
3  c = p.next
4  while (c.key < k)
5    p = c;
6    c = c.next;
7  if (c.key = k)
8    p.next = c.next
9    return true
10 else
11   return false
12 unlock l
```

```
ctn(k):
Node p, c
Lock l
1  lock l;
2  p = H;
3  c = p.next
4  while (c.key < k)
5    p = c;
6    c = c.next;
7  if (c.key = k)
8    return true
9  else
10   return false
11 unlock l
```

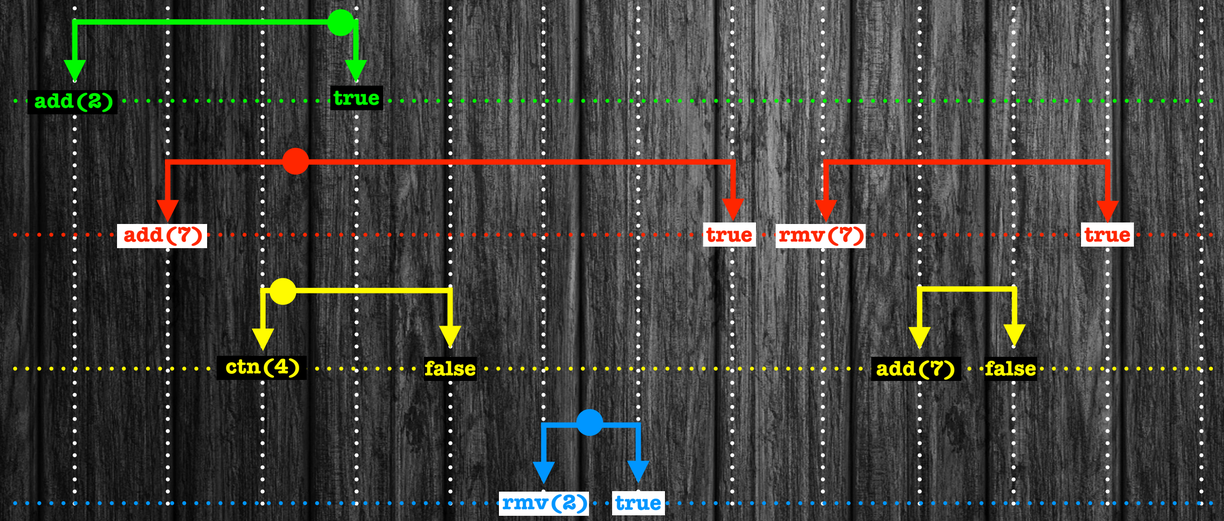


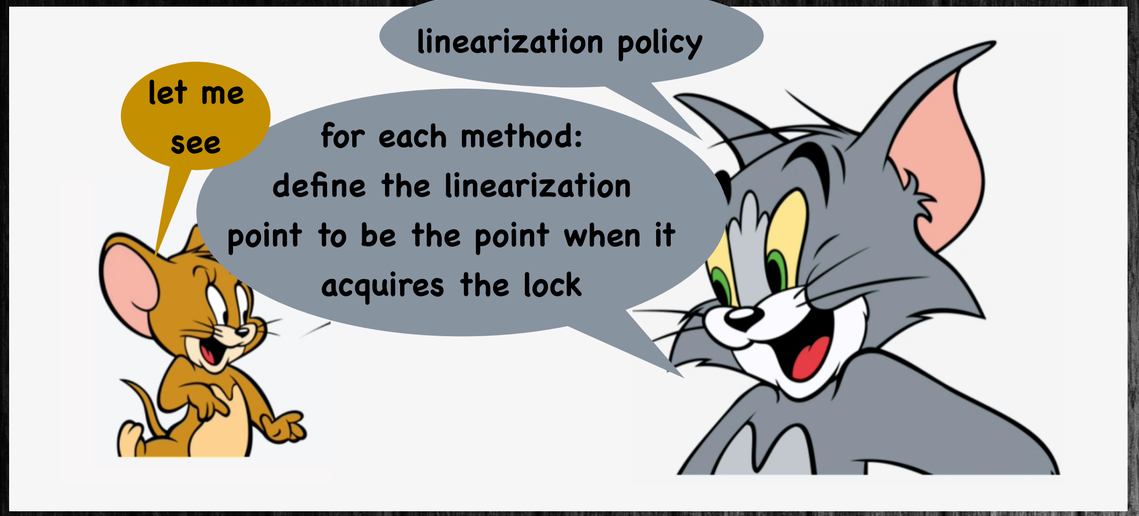



```
add(k):
Node p, c
Lock l
1  lock l;
2  p = H;
3  c = p.next
4  while (c.key < k)
5    p = c;
6    c = c.next;
7  if (c.key = k)
8    return false
9  else
10   n = new Node(k,-)
11   n.next = c
12   p.next = n
13   return true
14 unlock l
```

```
remove(k):
Node p, c
Lock l
1  lock l;
2  p = H;
3  c = p.next
4  while (c.key < k)
5    p = c;
6    c = c.next;
7  if (c.key = k)
8    p.next = c.next
9    return true
10 else
11   return false
12 unlock l
```

```
ctn(k):
Node p, c
Lock l
1  lock l;
2  p = H;
3  c = p.next
4  while (c.key < k)
5    p = c;
6    c = c.next;
7  if (c.key = k)
8    return true
9  else
10   return false
11 unlock l
```

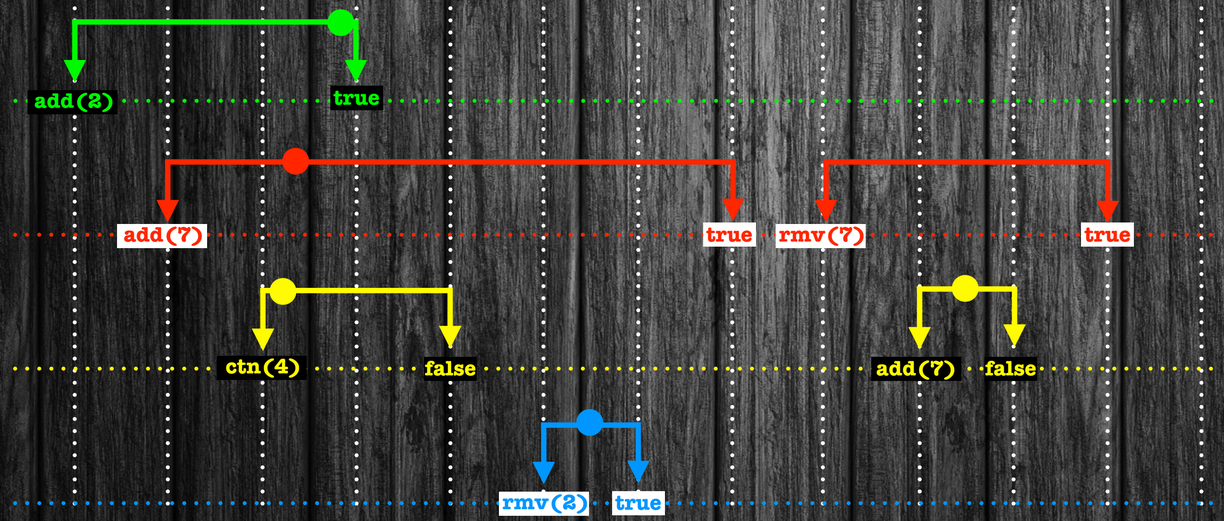


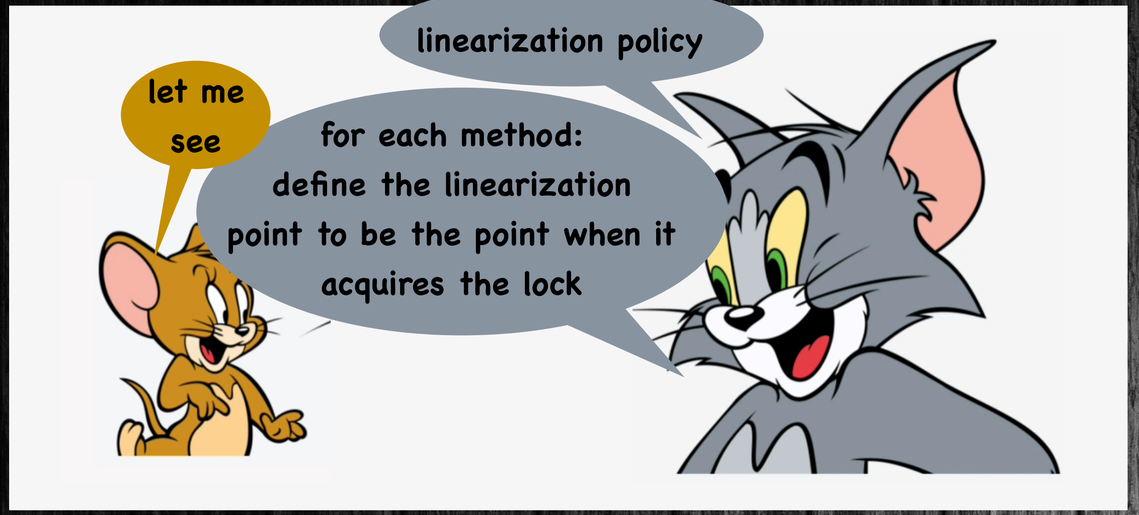



```
add(k):
Node p, c
Lock l
1  lock l;
2  p = H;
3  c = p.next
4  while (c.key < k)
5    p = c;
6    c = c.next;
7  if (c.key = k)
8    return false
9  else
10   n = new Node(k,-)
11   n.next = c
12   p.next = n
13   return true
14 unlock l
```

```
remove(k):
Node p, c
Lock l
1  lock l;
2  p = H;
3  c = p.next
4  while (c.key < k)
5    p = c;
6    c = c.next;
7  if (c.key = k)
8    p.next = c.next
9    return true
10 else
11   return false
12 unlock l
```

```
ctn(k):
Node p, c
Lock l
1  lock l;
2  p = H;
3  c = p.next
4  while (c.key < k)
5    p = c;
6    c = c.next;
7  if (c.key = k)
8    return true
9  else
10   return false
11 unlock l
```

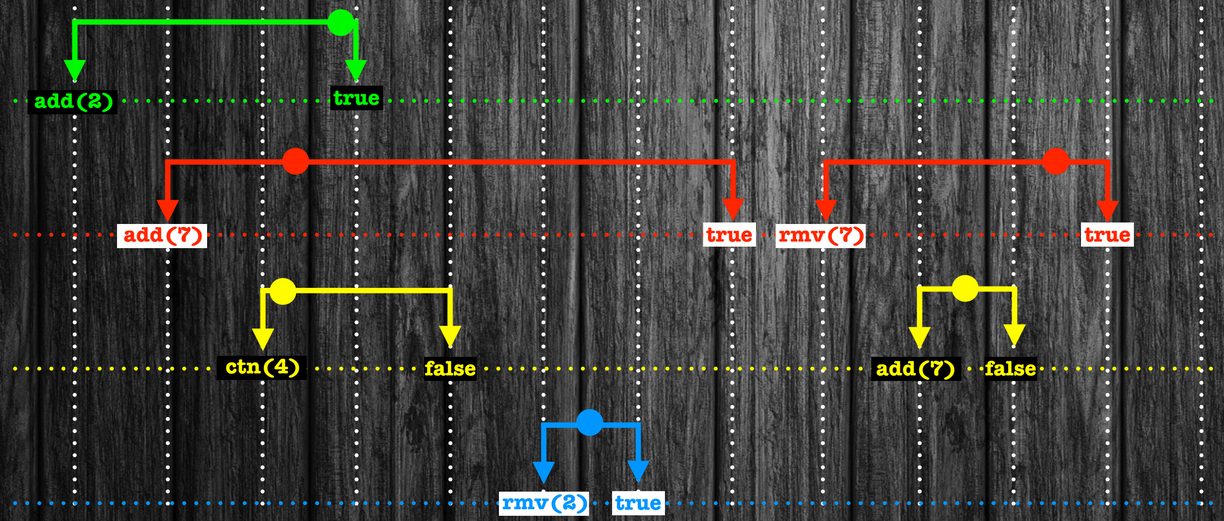


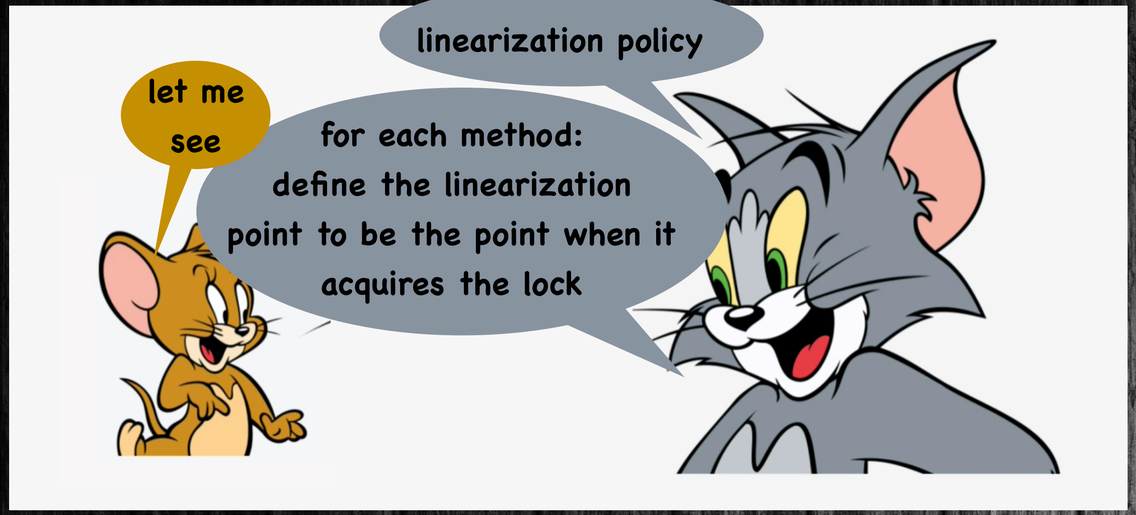



```
add(k):
Node p, c
Lock l
1  lock l;
2  p = H;
3  c = p.next
4  while (c.key < k)
5    p = c;
6    c = c.next;
7  if (c.key = k)
8    return false
9  else
10   n = new Node(k,-)
11   n.next = c
12   p.next = n
13   return true
14 unlock l
```

```
remove(k):
Node p, c
Lock l
1  lock l;
2  p = H;
3  c = p.next
4  while (c.key < k)
5    p = c;
6    c = c.next;
7  if (c.key = k)
8    p.next = c.next
9    return true
10 else
11   return false
12 unlock l
```

```
ctn(k):
Node p, c
Lock l
1  lock l;
2  p = H;
3  c = p.next
4  while (c.key < k)
5    p = c;
6    c = c.next;
7  if (c.key = k)
8    return true
9  else
10   return false
11 unlock l
```

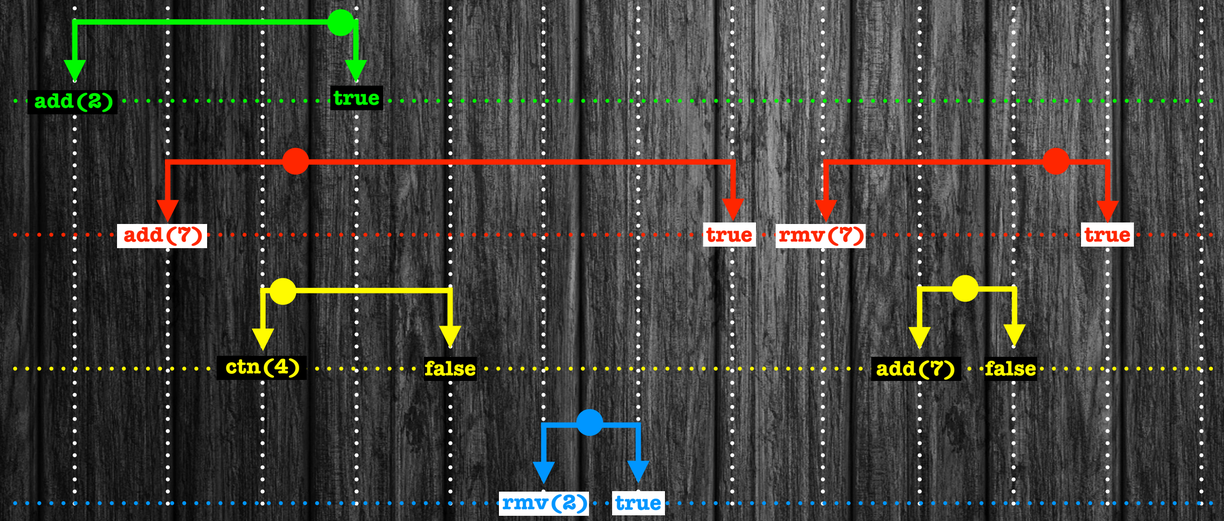




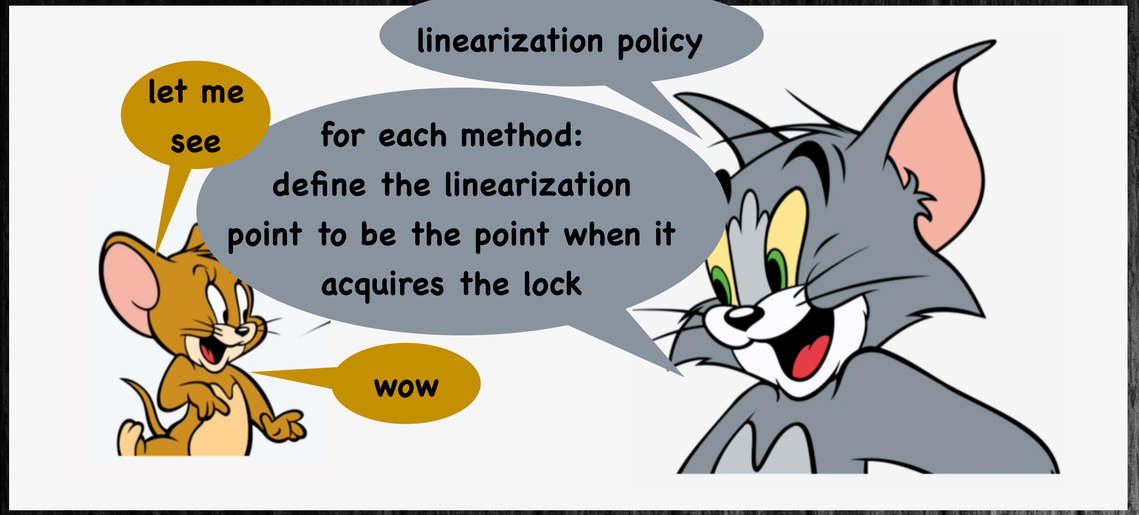

```
add(k):
Node p, c
Lock l
1  lock l;
2  p = H;
3  c = p.next
4  while (c.key < k)
5    p = c;
6    c = c.next;
7  if (c.key = k)
8    return false
9  else
10   n = new Node(k,-)
11   n.next = c
12   p.next = n
13   return true
14 unlock l
```

```
remove(k):
Node p, c
Lock l
1  lock l;
2  p = H;
3  c = p.next
4  while (c.key < k)
5    p = c;
6    c = c.next;
7  if (c.key = k)
8    p.next = c.next
9    return true
10 else
11   return false
12 unlock l
```

```
ctn(k):
Node p, c
Lock l
1  lock l;
2  p = H;
3  c = p.next
4  while (c.key < k)
5    p = c;
6    c = c.next;
7  if (c.key = k)
8    return true
9  else
10   return false
11 unlock l
```



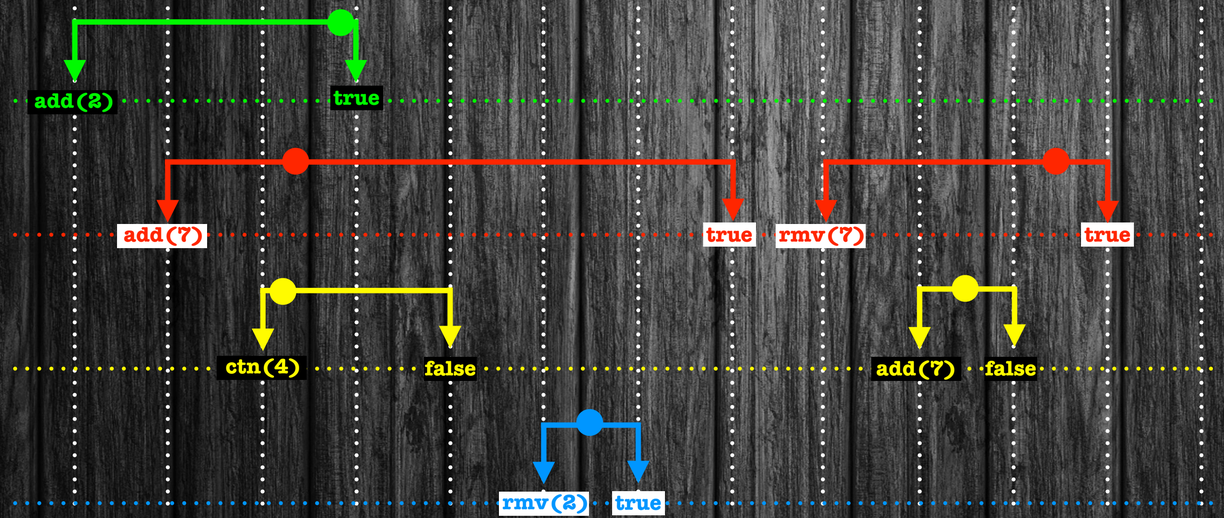
✔



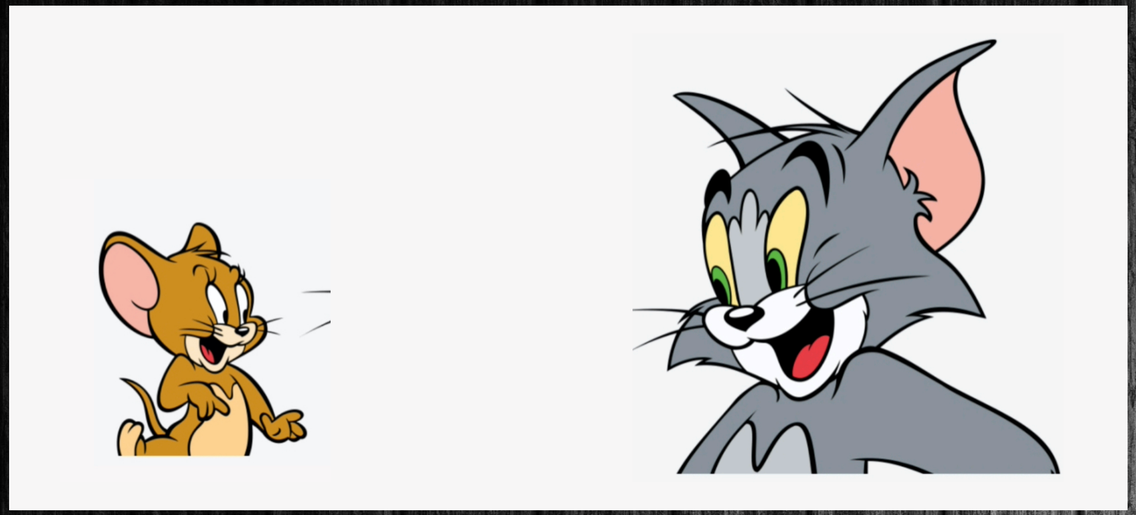

```
add(k):
Node p, c
Lock l
1  lock l;
2  p = H;
3  c = p.next
4  while (c.key < k)
5    p = c;
6    c = c.next;
7  if (c.key = k)
8    return false
9  else
10   n = new Node(k,-)
11   n.next = c
12   p.next = n
13   return true
14 unlock l
```

```
remove(k):
Node p, c
Lock l
1  lock l;
2  p = H;
3  c = p.next
4  while (c.key < k)
5    p = c;
6    c = c.next;
7  if (c.key = k)
8    p.next = c.next
9    return true
10 else
11   return false
12 unlock l
```

```
ctn(k):
Node p, c
Lock l
1  lock l;
2  p = H;
3  c = p.next
4  while (c.key < k)
5    p = c;
6    c = c.next;
7  if (c.key = k)
8    return true
9  else
10   return false
11 unlock l
```



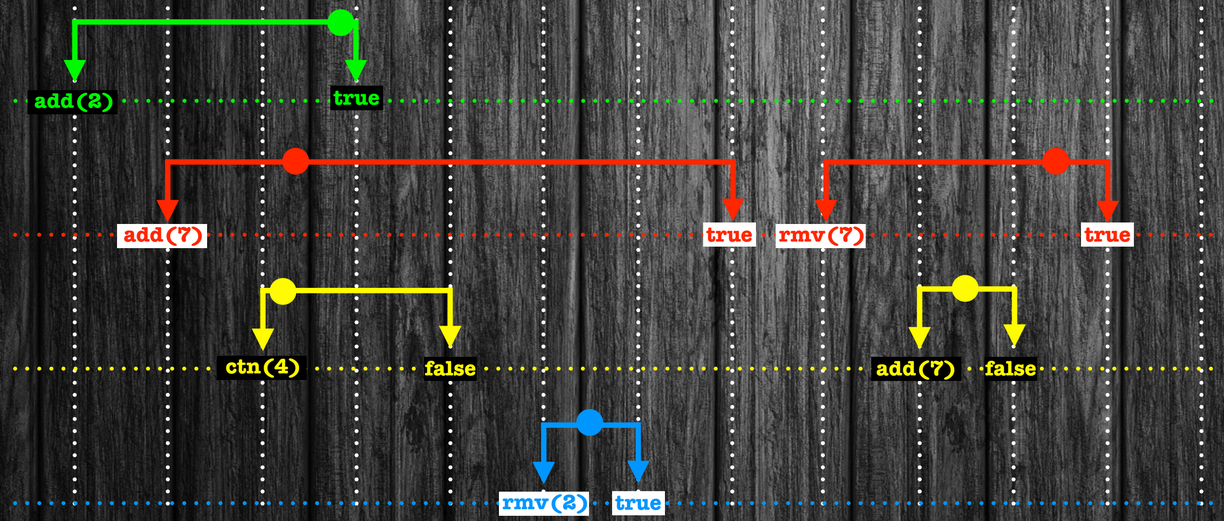
✔

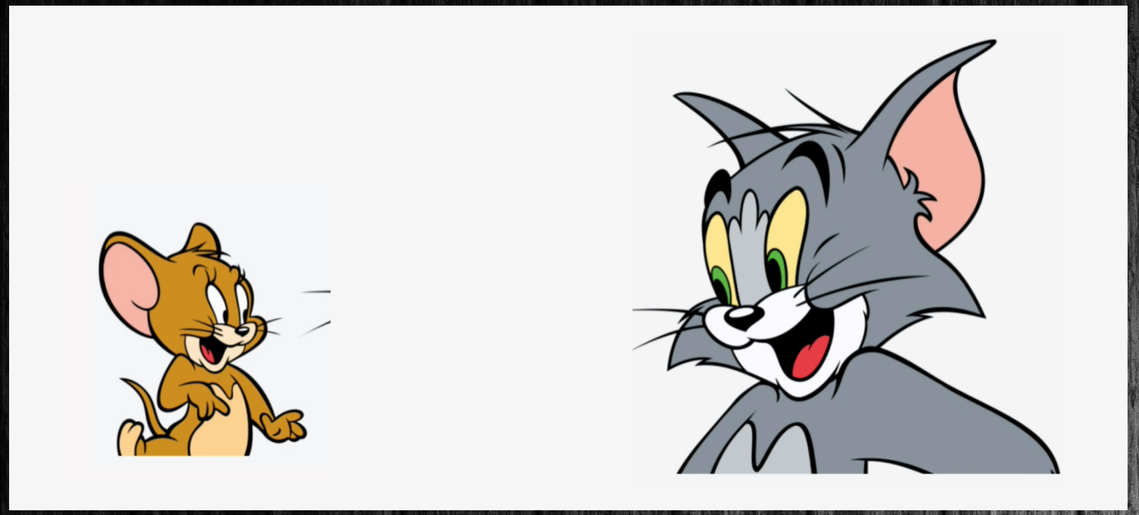


```
add(k):
Node p, c
Lock l
1  lock l;
2  p = H;
3  c = p.next
4  while (c.key < k)
5    p = c;
6    c = c.next;
7  if (c.key = k)
8    return false
9  else
10   n = new Node(k,-)
11   n.next = c
12   p.next = n
13   return true
14 unlock l
```

```
remove(k):
Node p, c
Lock l
1  lock l;
2  p = H;
3  c = p.next
4  while (c.key < k)
5    p = c;
6    c = c.next;
7  if (c.key = k)
8    p.next = c.next
9    return true
10 else
11   return false
12 unlock l
```

```
ctn(k):
Node p, c
Lock l
1  lock l;
2  p = H;
3  c = p.next
4  while (c.key < k)
5    p = c;
6    c = c.next;
7  if (c.key = k)
8    return true
9  else
10   return false
11 unlock l
```

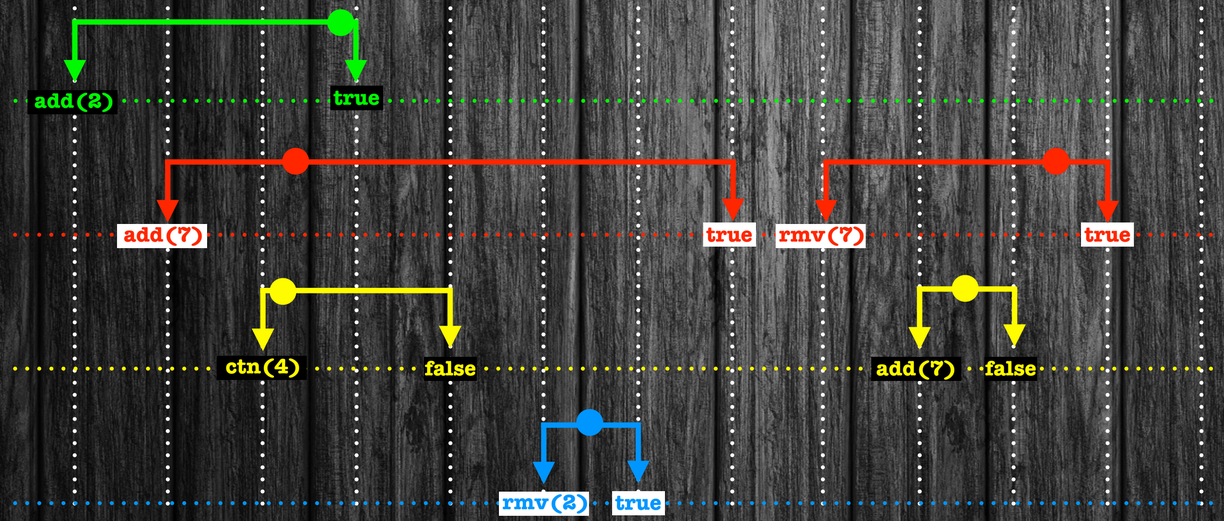




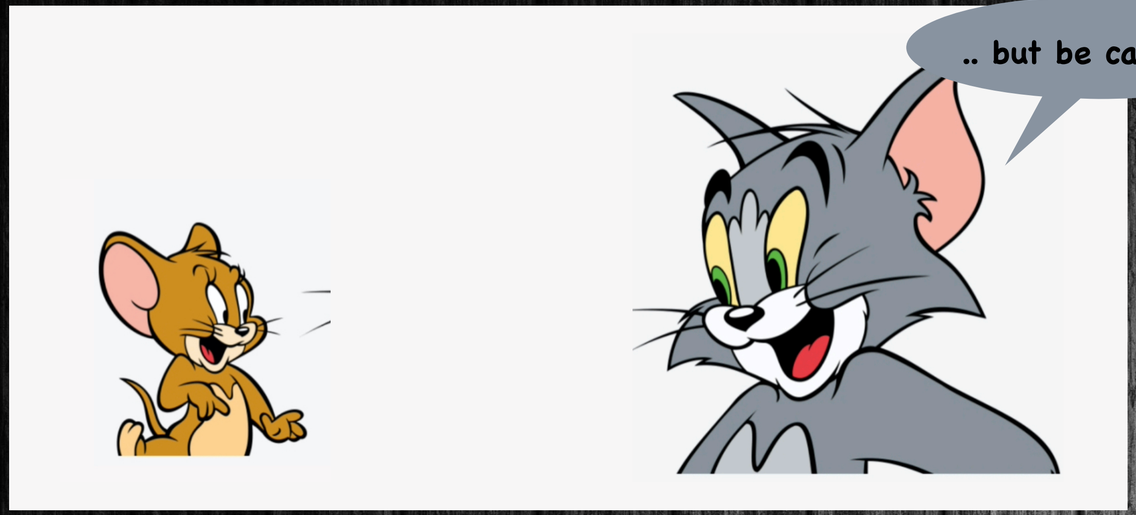
```
add(k):
Node p, c
Lock l
1  lock l;
2  p = H;
3  c = p.next
4  while (c.key < k)
5    p = c;
6    c = c.next;
7  if (c.key = k)
8    return false
9  else
10   n = new Node(k,-)
11   n.next = c
12   p.next = n
13   return true
14 unlock l
```

```
remove(k):
Node p, c
Lock l
1  lock l;
2  p = H;
3  c = p.next
4  while (c.key < k)
5    p = c;
6    c = c.next;
7  if (c.key = k)
8    p.next = c.next
9    return true
10 else
11   return false
12 unlock l
```

```
ctn(k):
Node p, c
Lock l
1  lock l;
2  p = H;
3  c = p.next
4  while (c.key < k)
5    p = c;
6    c = c.next;
7  if (c.key = k)
8    return true
9  else
10   return false
11 unlock l
```



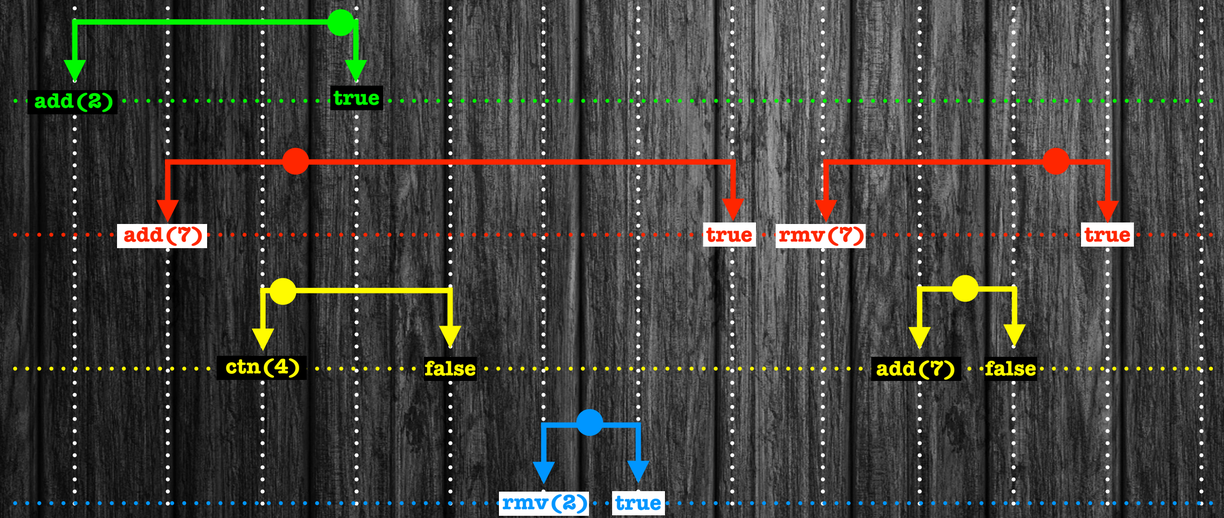
✔




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

```
remove(k):
Node p, c
Lock l
1  lock l;
2  p = H;
3  c = p.next
4  while (c.key < k)
5    p = c;
6    c = c.next;
7  if (c.key = k)
8    p.next = c.next
9    return true
10 else
11   return false
12 unlock l
```

```
ctn(k):
Node p, c
Lock l
1  lock l;
2  p = H;
3  c = p.next
4  while (c.key < k)
5    p = c;
6    c = c.next;
7  if (c.key = k)
8    return true
9  else
10   return false
11 unlock l
```



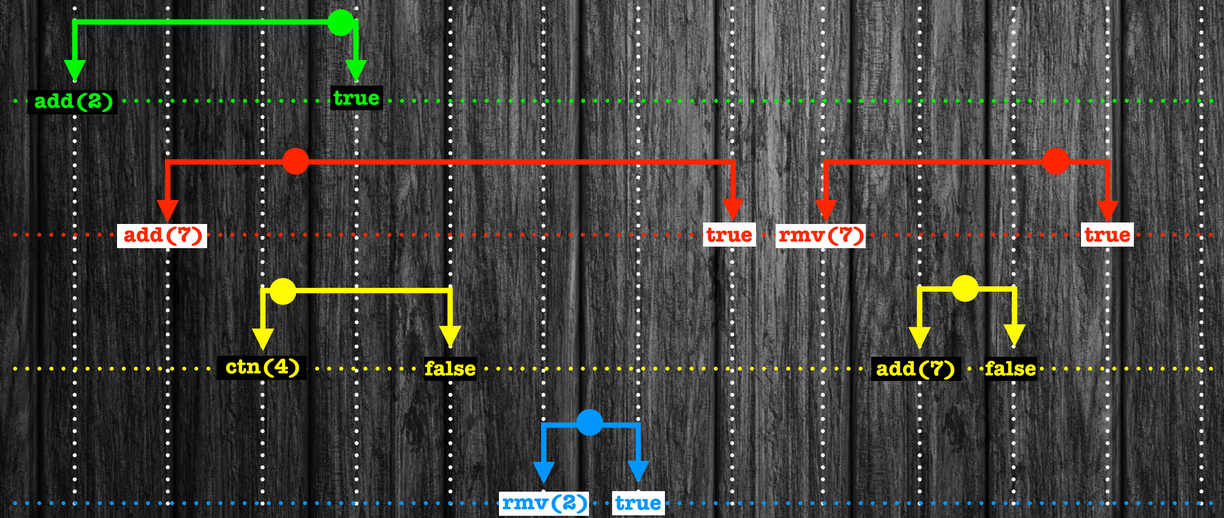
✔



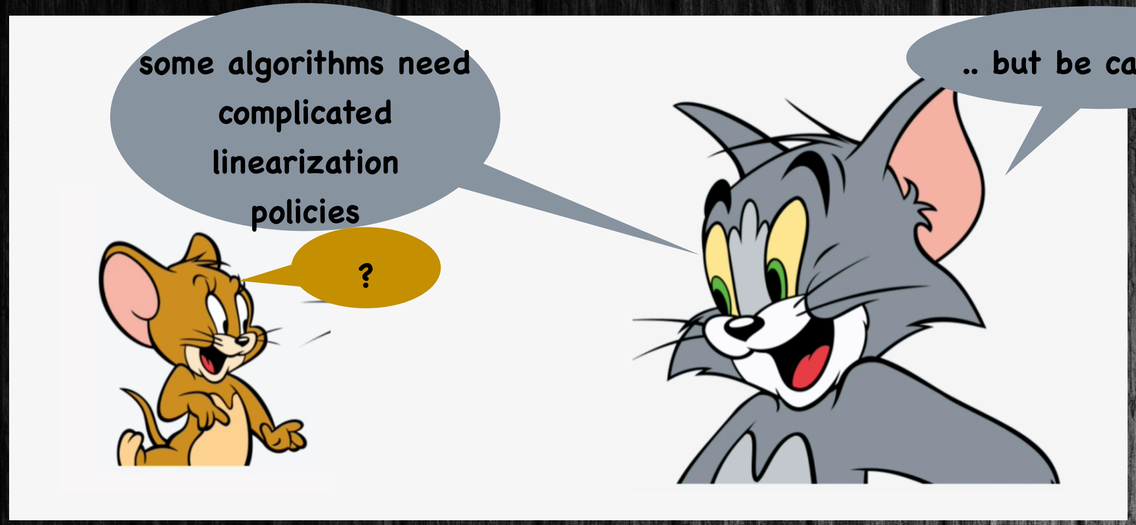

```
add(k):
Node p, c
Lock l
1  lock l;
2  p = H;
3  c = p.next
4  while (c.key < k)
5    p = c;
6    c = c.next;
7  if (c.key = k)
8    return false
9  else
10   n = new Node(k,-)
11   n.next = c
12   p.next = n
13   return true
14 unlock l
```

```
remove(k):
Node p, c
Lock l
1  lock l;
2  p = H;
3  c = p.next
4  while (c.key < k)
5    p = c;
6    c = c.next;
7  if (c.key = k)
8    p.next = c.next
9    return true
10 else
11   return false
12 unlock l
```

```
ctn(k):
Node p, c
Lock l
1  lock l;
2  p = H;
3  c = p.next
4  while (c.key < k)
5    p = c;
6    c = c.next;
7  if (c.key = k)
8    return true
9  else
10   return false
11 unlock l
```



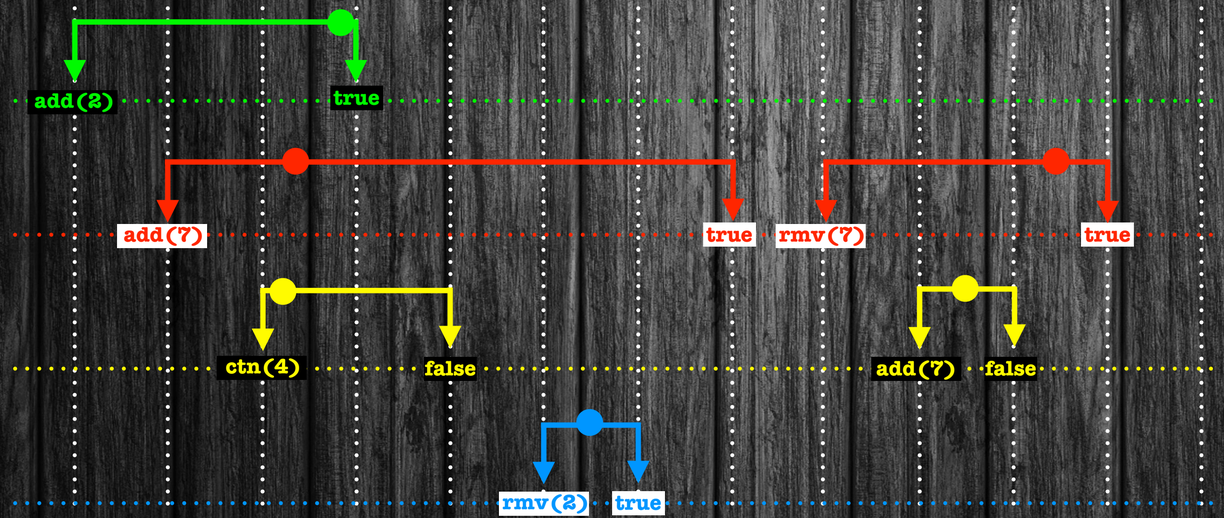
✔



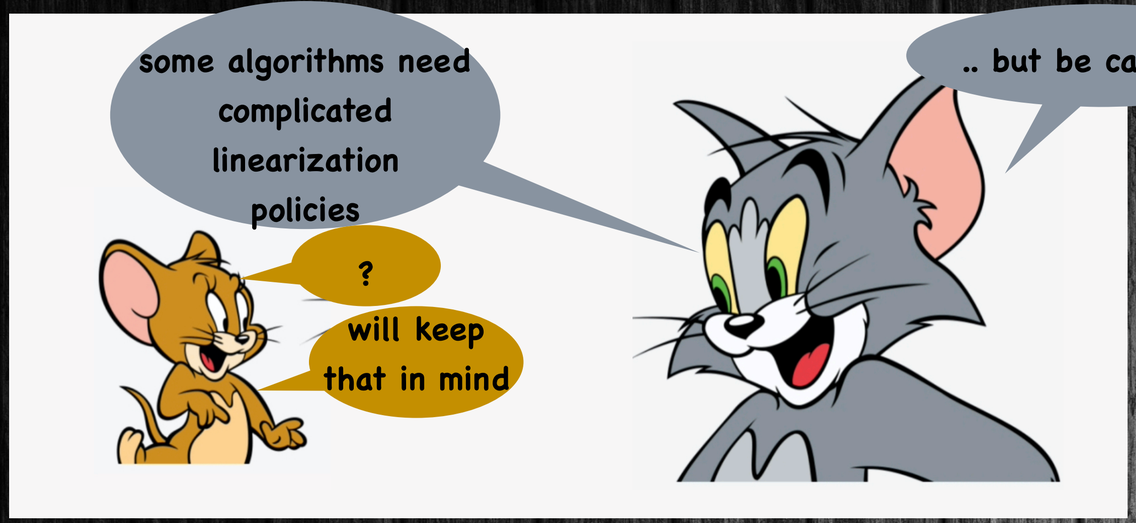

```
add(k):
Node p, c
Lock l
1  lock l;
2  p = H;
3  c = p.next
4  while (c.key < k)
5    p = c;
6    c = c.next;
7  if (c.key = k)
8    return false
9  else
10   n = new Node(k,-)
11   n.next = c
12   p.next = n
13   return true
14 unlock l
```

```
remove(k):
Node p, c
Lock l
1  lock l;
2  p = H;
3  c = p.next
4  while (c.key < k)
5    p = c;
6    c = c.next;
7  if (c.key = k)
8    p.next = c.next
9    return true
10 else
11   return false
12 unlock l
```

```
ctn(k):
Node p, c
Lock l
1  lock l;
2  p = H;
3  c = p.next
4  while (c.key < k)
5    p = c;
6    c = c.next;
7  if (c.key = k)
8    return true
9  else
10   return false
11 unlock l
```



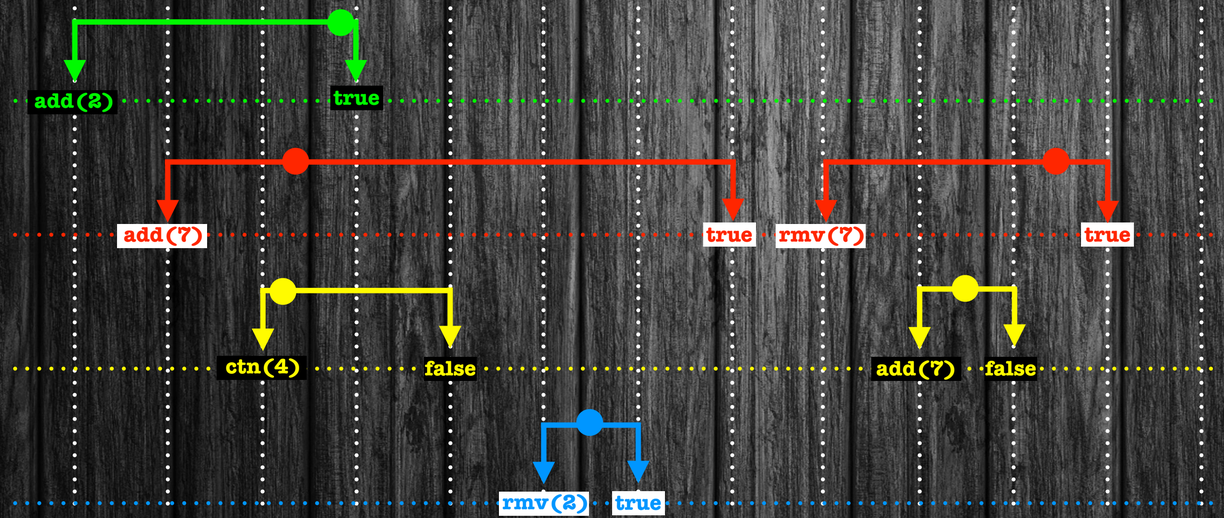
✔




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

```
remove(k):
Node p, c
Lock l
1  lock l;
2  p = H;
3  c = p.next
4  while (c.key < k)
5    p = c;
6    c = c.next;
7  if (c.key = k)
8    p.next = c.next
9    return true
10 else
11   return false
12 unlock l
```

```
ctn(k):
Node p, c
Lock l
1  lock l;
2  p = H;
3  c = p.next
4  while (c.key < k)
5    p = c;
6    c = c.next;
7  if (c.key = k)
8    return true
9  else
10   return false
11 unlock l
```



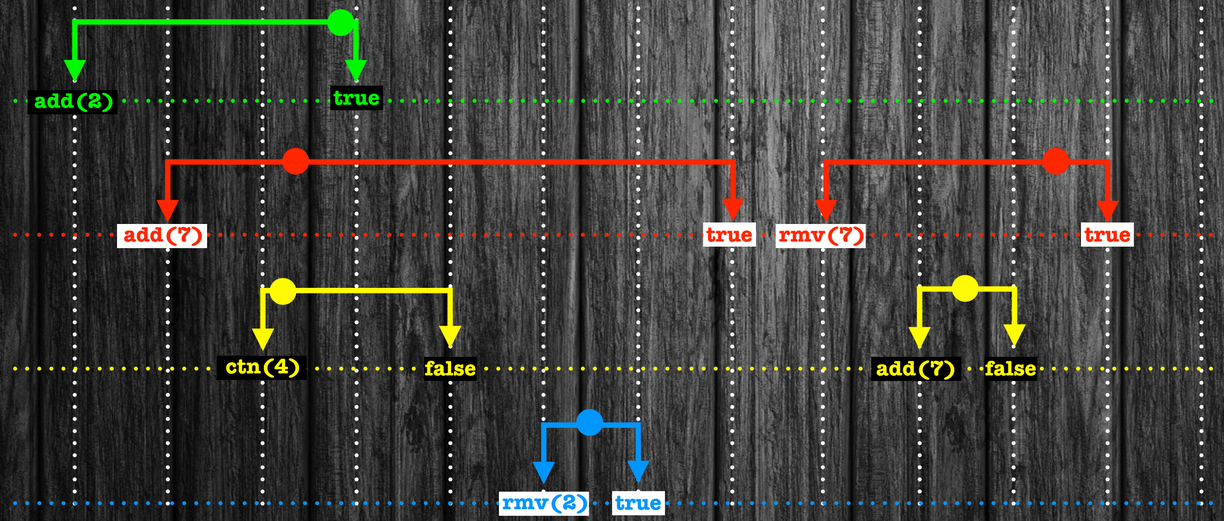
✔




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

```
remove(k):
Node p, c
Lock l
1  lock l;
2  p = H;
3  c = p.next
4  while (c.key < k)
5    p = c;
6    c = c.next;
7  if (c.key = k)
8    p.next = c.next
9    return true
10 else
11   return false
12 unlock l
```

```
ctn(k):
Node p, c
Lock l
1  lock l;
2  p = H;
3  c = p.next
4  while (c.key < k)
5    p = c;
6    c = c.next;
7  if (c.key = k)
8    return true
9  else
10   return false
11 unlock l
```



✔


let us look
at more complicated
algorithms

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 | Lazy Synchronization

Goal — even more concurrency

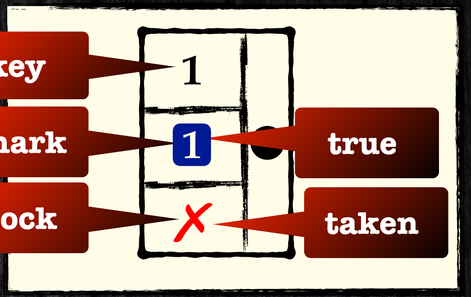
How? — ctn need not acquire locks

Strategy — logical deletion

Typical application:

- 90% ctn
- 9% add
- 1% rcv

logically deleted? → Boolean flag



set mark = 1

rmv

1. delete logically
2. delete physically

ctn returns false if:

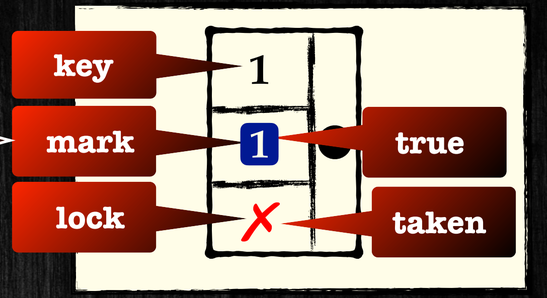
1. node logically deleted

or

2. node absent

	linearization point
successful add	“when p.next is set”
unsuccessful add	“when unmarked entry is observed”
successful rmv	“when the cell is marked”
unsuccessful rmv	“first unmarked entry with larger key”
successful ctn	“when unmarked entry with input key found”
unsuccessful ctn	?

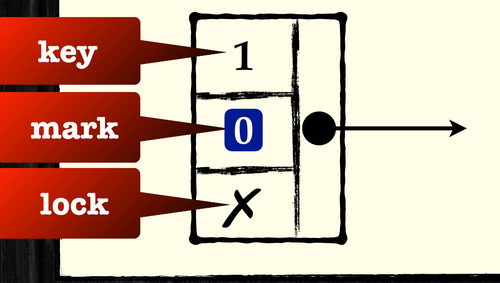
logically deleted? → Boolean flag

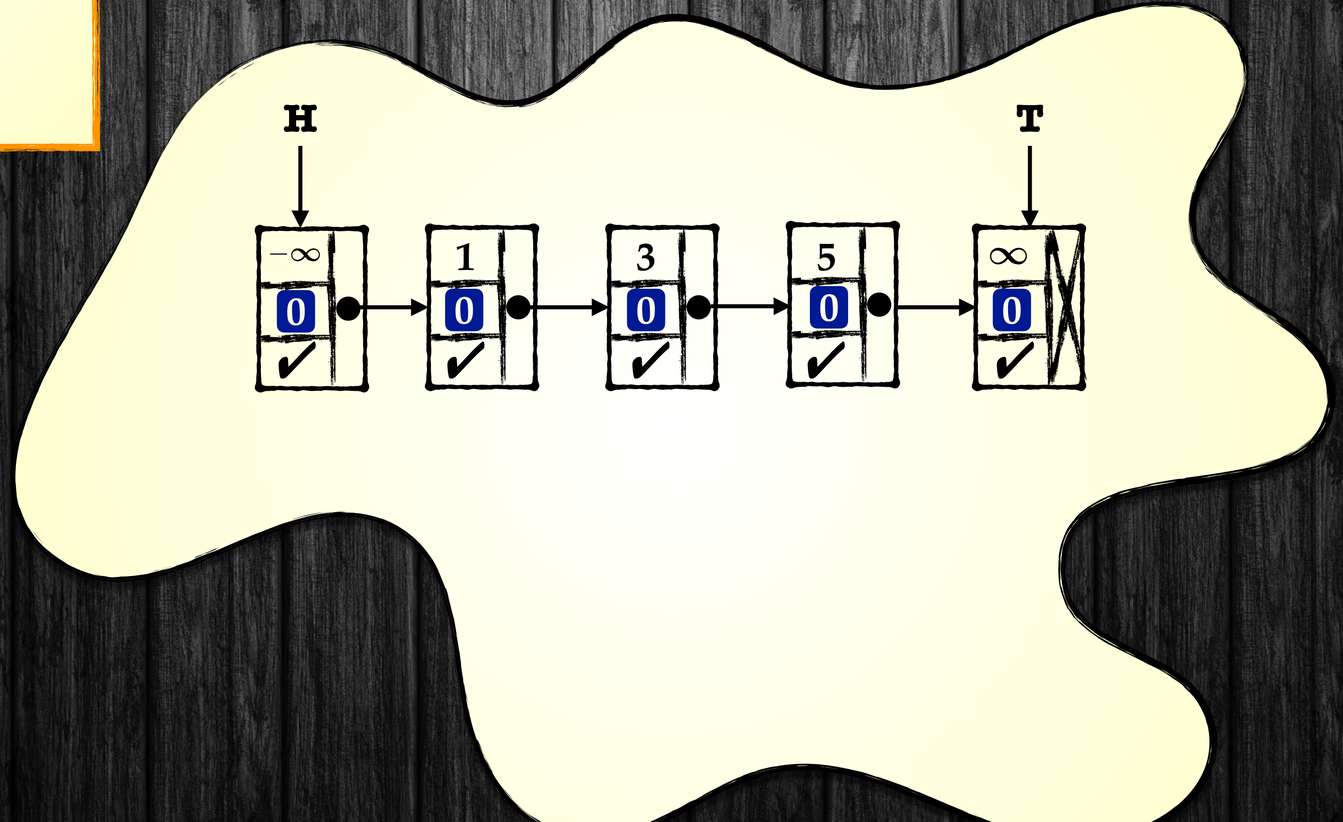


Sets | Lazy Synchronization

```
locate(e):
local p, c
1  while (true)
2     p := head;
3     c := p.next;
4     while (c.val < e)
5        p := c;
6        c := c.next
7     lock(p); lock(c);
8     if (!p.mark && !c.mark && p.next=c)
9        return(p,c);
10    else
11       unlock(p);
12       unlock(c);
```

locate(4)

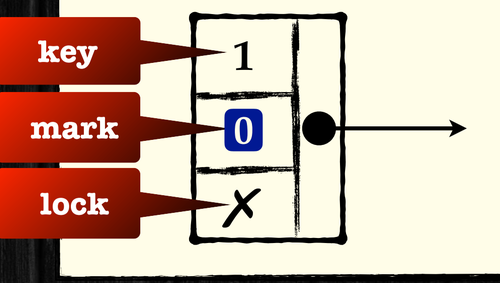


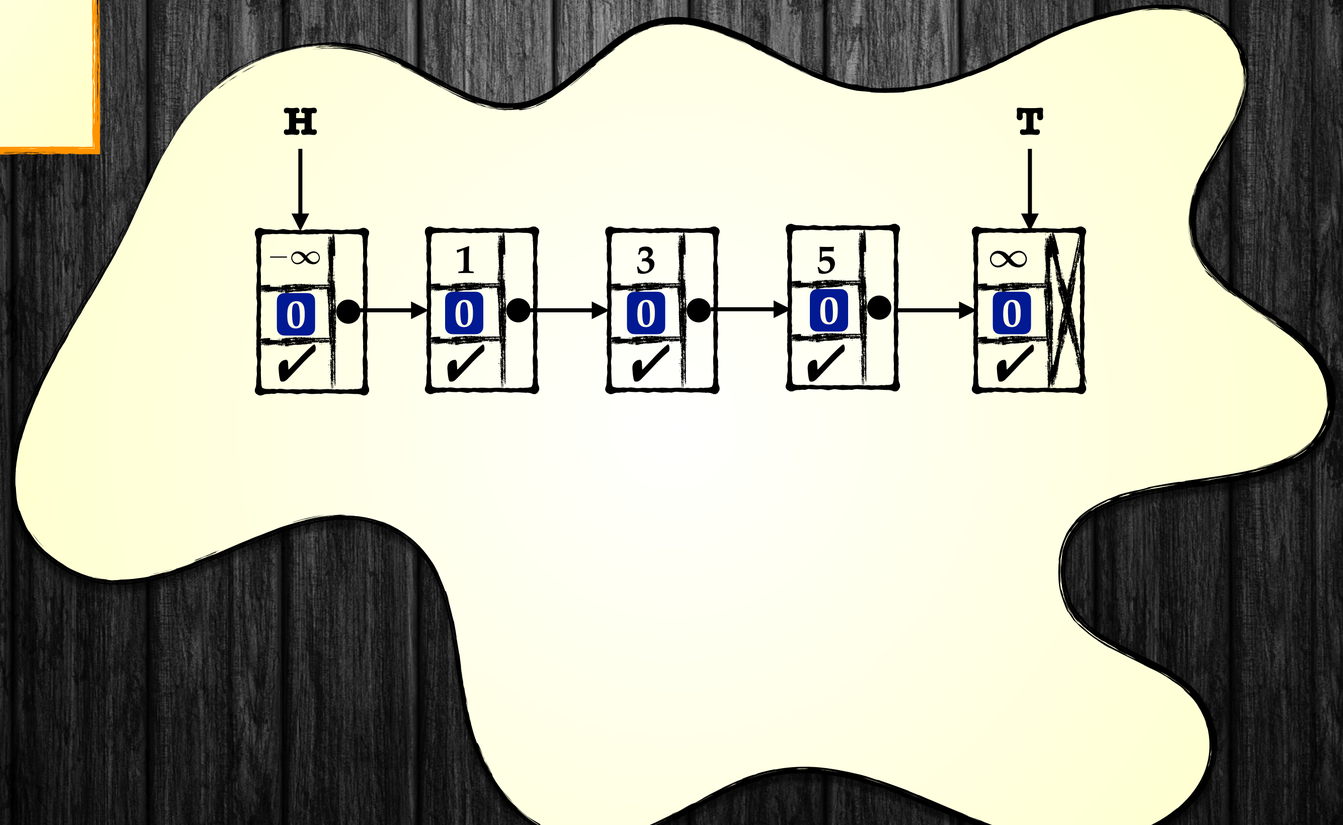


Sets | Lazy Synchronization

```
locate(e):
local p, c
1  while (true)
2     p := head;
3     c := p.next;
4     while (c.val < e)
5        p := c;
6        c := c.next
7     lock(p); lock(c);
8     if (!p.mark && !c.mark && p.next=c)
9        return(p,c);
10    else
11       unlock(p);
12       unlock(c);
```

locate(4)

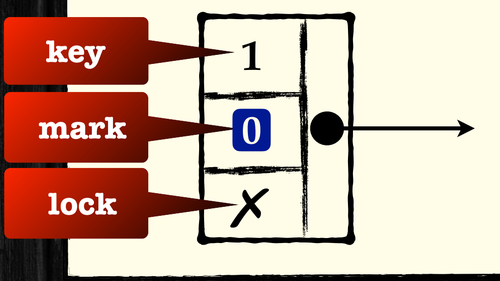


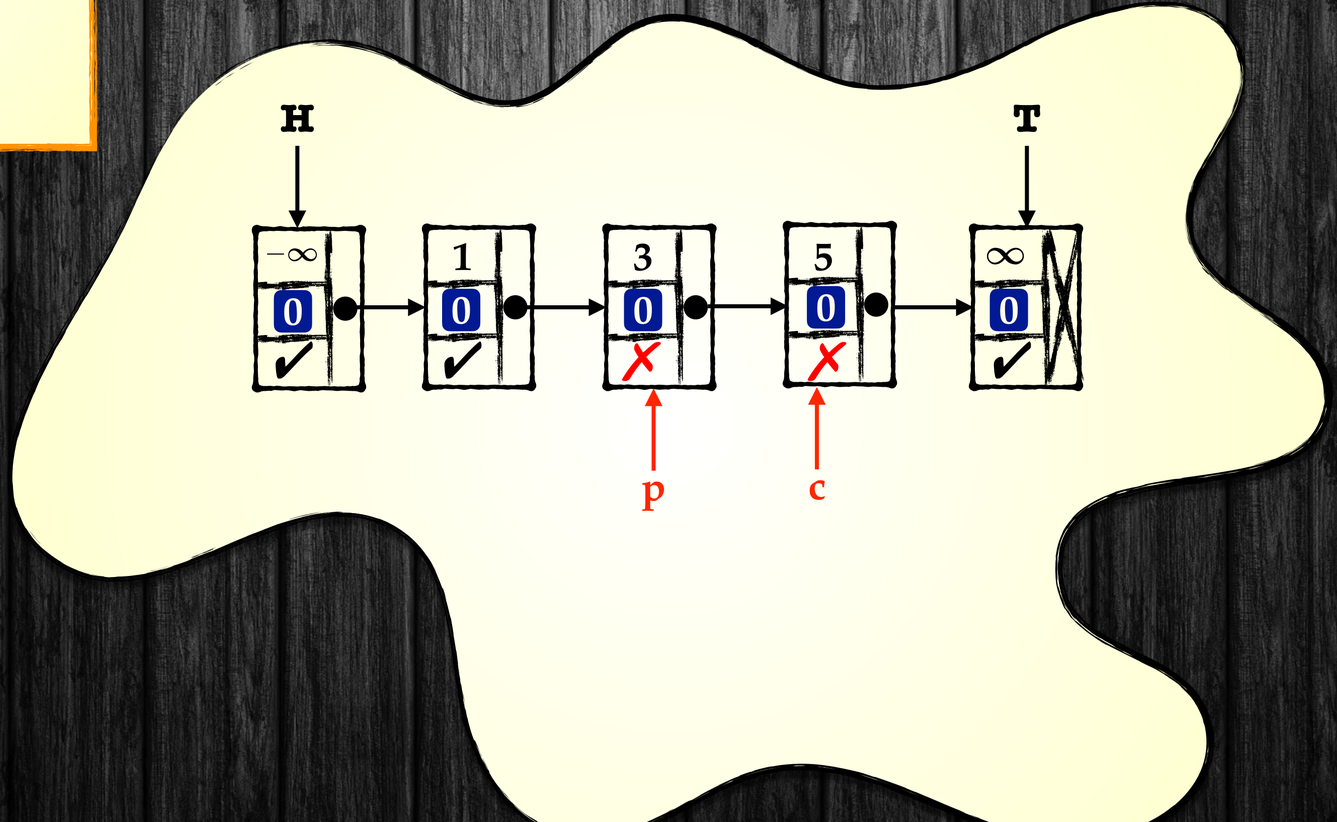


Sets | Lazy Synchronization

```
locate(e):
local p, c
1  while (true)
2     p := head;
3     c := p.next;
4     while (c.val < e)
5        p := c;
6        c := c.next
7     lock(p); lock(c);
8     if (!p.mark && !c.mark && p.next=c)
9        return(p,c);
10    else
11       unlock(p);
12       unlock(c);
```



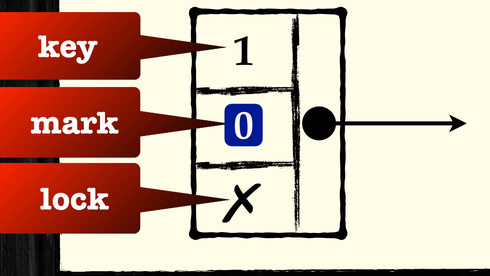


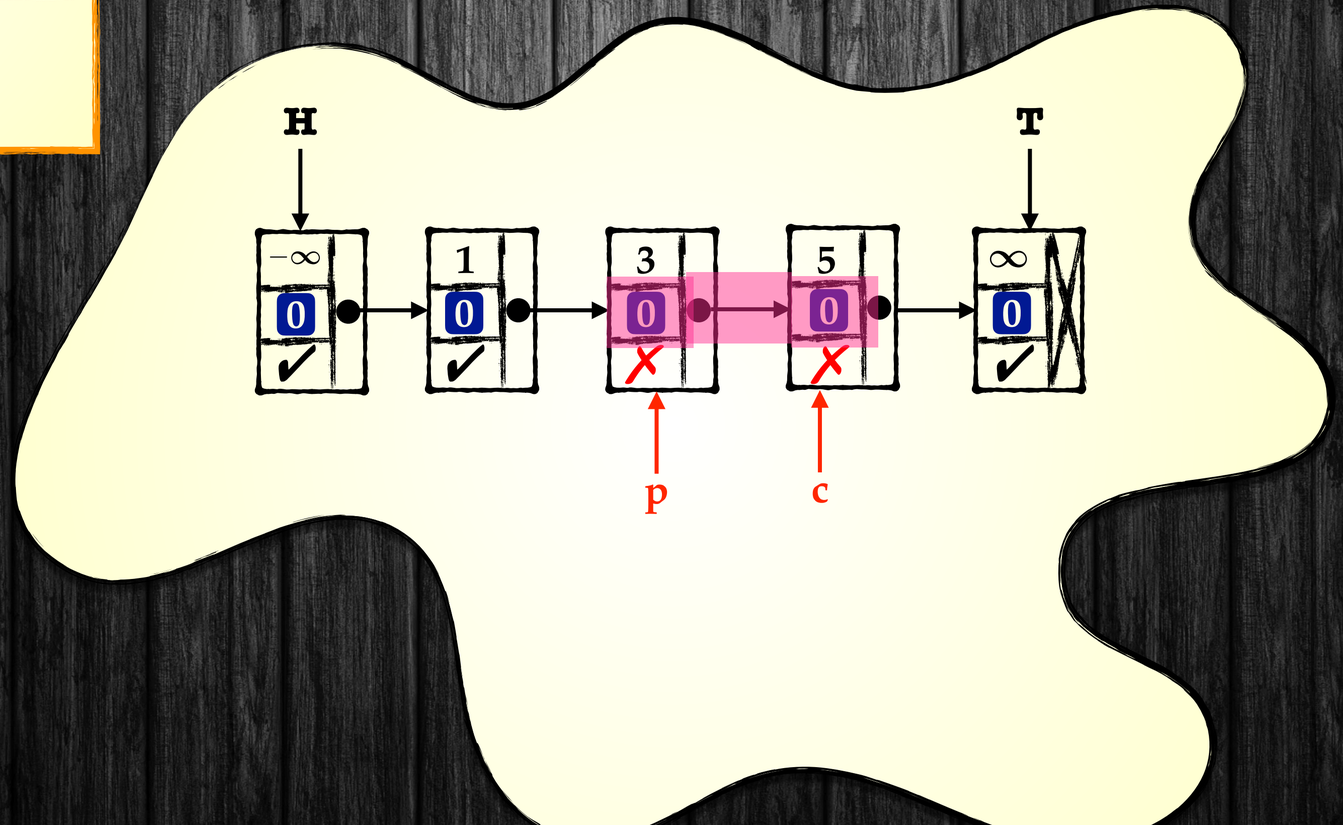


Sets | Lazy Synchronization

```
locate(e):
local p, c
1  while (true)
2     p := head;
3     c := p.next;
4     while (c.val < e)
5        p := c;
6        c := c.next
7     lock(p); lock(c);
8     if (!p.mark && !c.mark && p.next=c)
9        return(p,c);
10    else
11       unlock(p);
12       unlock(c);
```

locate(4)

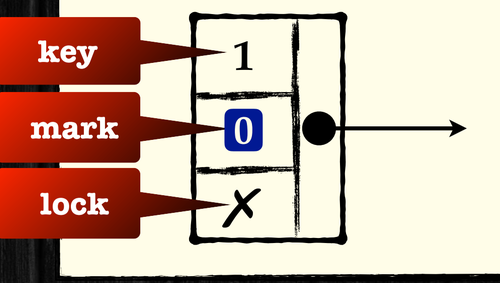


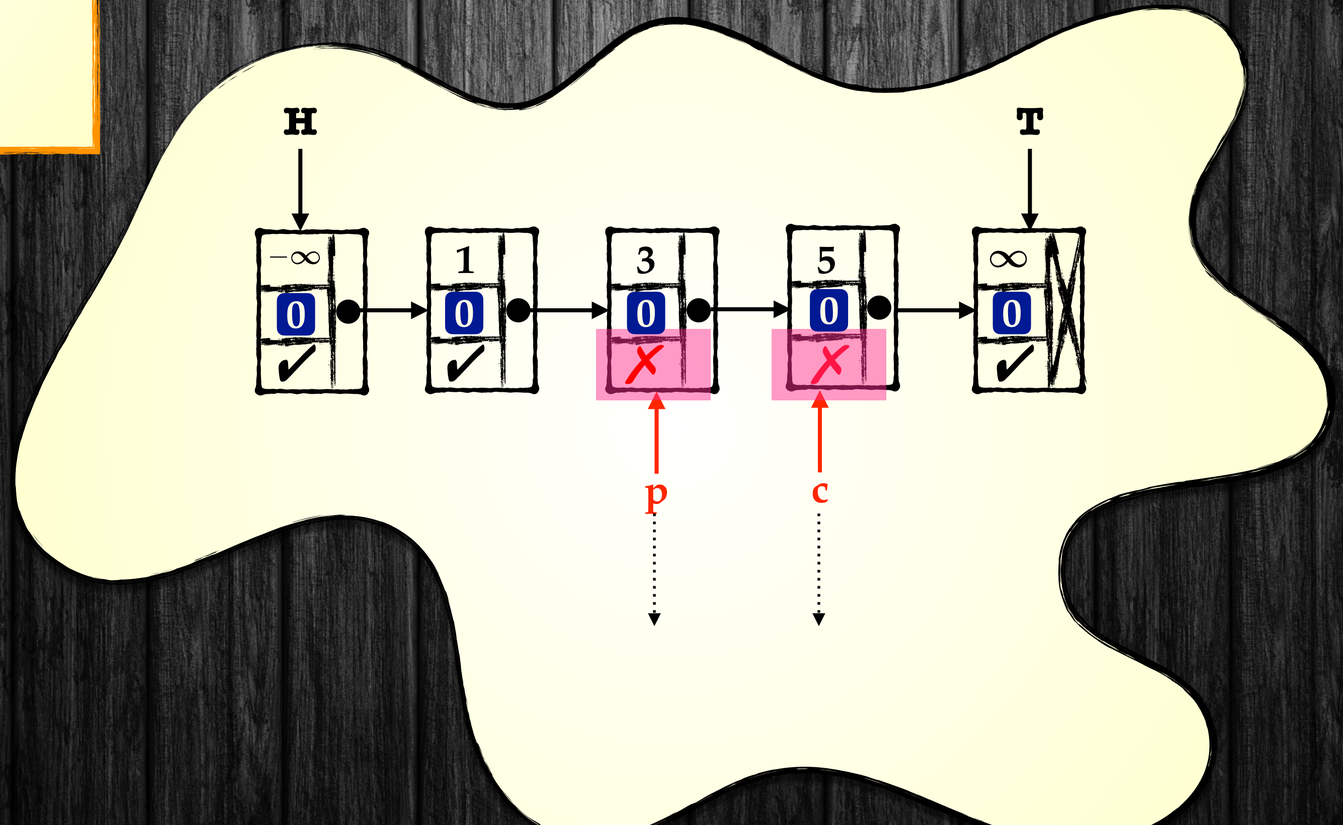


Sets | Lazy Synchronization

```
locate(e):
local p, c
1  while (true)
2     p := head;
3     c := p.next;
4     while (c.val < e)
5        p := c;
6        c := c.next
7     lock(p); lock(c);
8     if (!p.mark && !c.mark && p.next=c)
9        return(p,c);
10    else
11       unlock(p);
12       unlock(c);
```

locate(4)



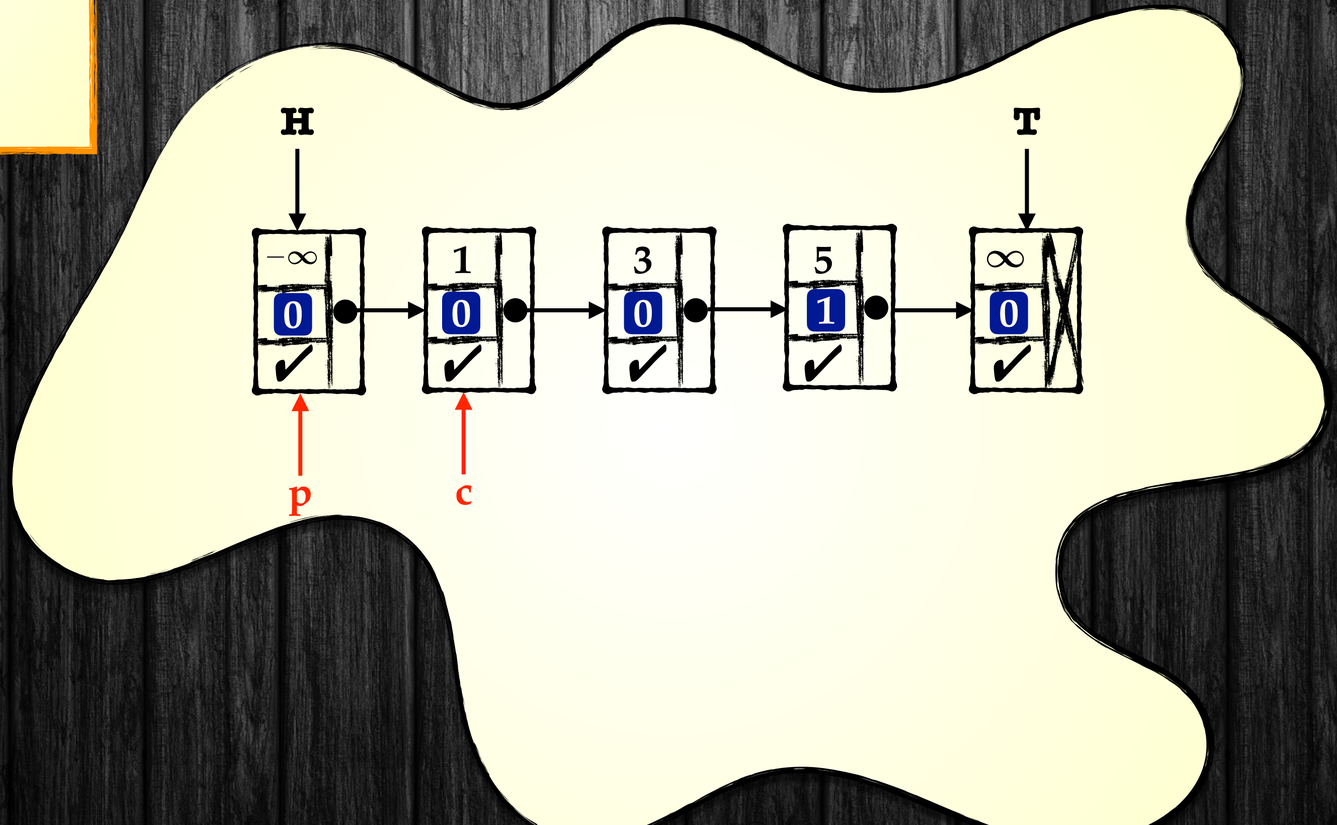


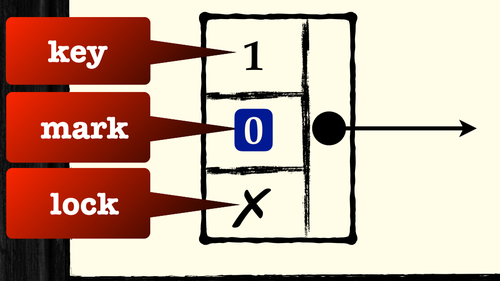
Sets | Lazy Synchronization

```
locate(e):
local p, c
1  while (true)
2     p := head;
3     c := p.next;
4     while (c.val < e)
5        p := c;
6        c := c.next
7     lock(p); lock(c);
8     if (!p.mark && !c.mark && p.next=c)
9        return(p,c);
10    else
11       unlock(p);
12       unlock(c);
```

locate(4)

different scenario



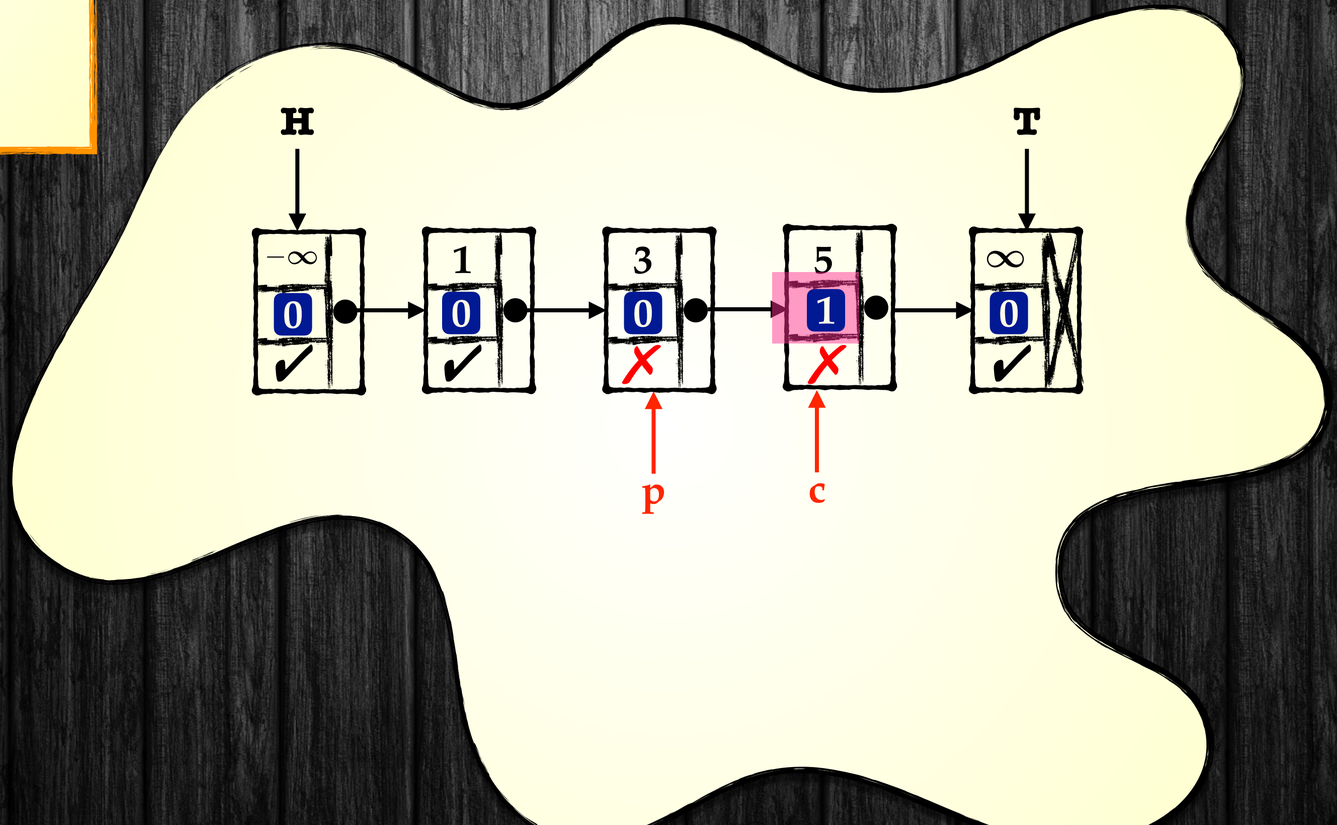


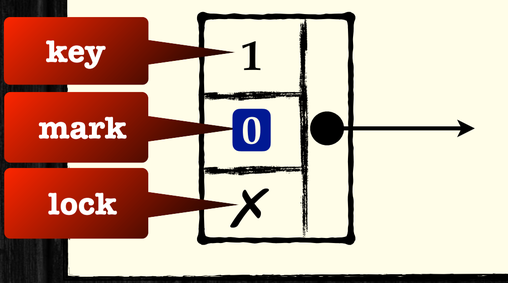
Sets | Lazy Synchronization

```
locate(e):
local p, c
1  while (true)
2     p := head;
3     c := p.next;
4     while (c.val < e)
5        p := c;
6        c := c.next
7     lock(p); lock(c);
8     if (!p.mark && !c.mark && p.next=c)
9        return(p,c);
10    else
11       unlock(p);
12       unlock(c);
```

locate(4)

different scenario





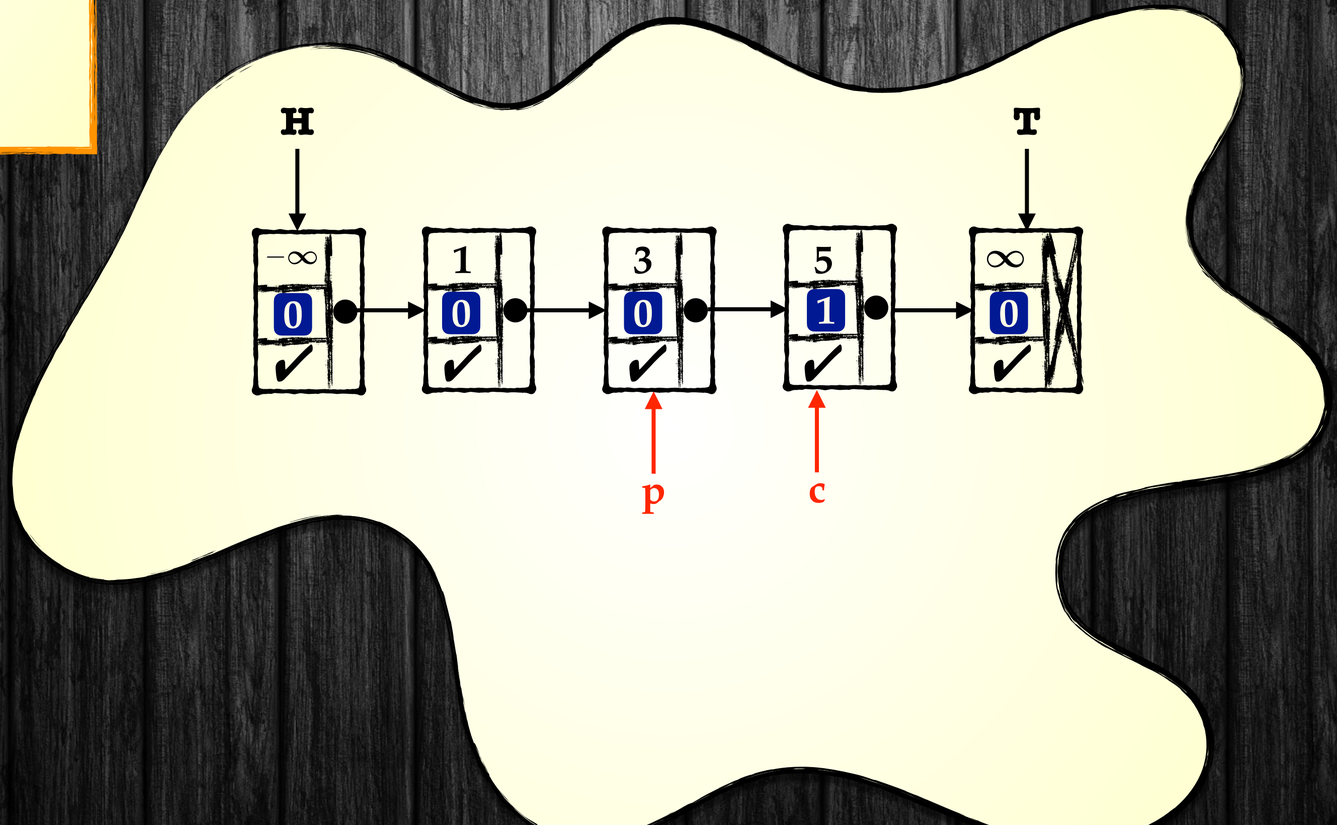
Sets | Lazy Synchronization

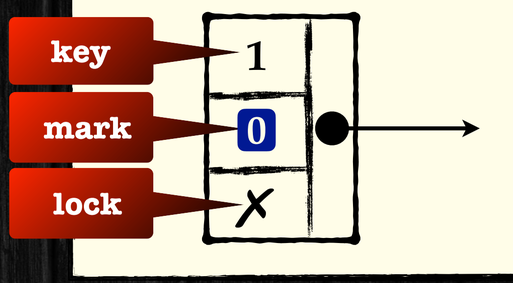
```
locate(e):
local p, c
1  while (true)
2     p := head;
3     c := p.next;
4     while (c.val < e)
5        p := c;
6        c := c.next
7     lock(p); lock(c);
8     if (!p.mark && !c.mark && p.next=c)
9        return(p,c);
10    else
11       unlock(p);
12       unlock(c);
```

try again

locate(4)

different scenario





Sets | Lazy Synchronization

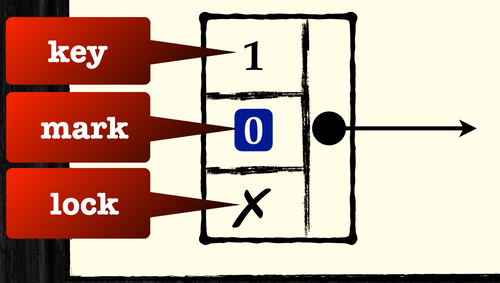
```
add(e):
   local p, c, n, r
   1  (p,c) := locate(e);
   2  if (c.val <> e)
   3     n := new Node(0,e,c,false);
   4     p.next := n;
   5     r := true;
   6  else r := false;
   7  unlock(p);
   8  unlock(c);
   9  return r;
```

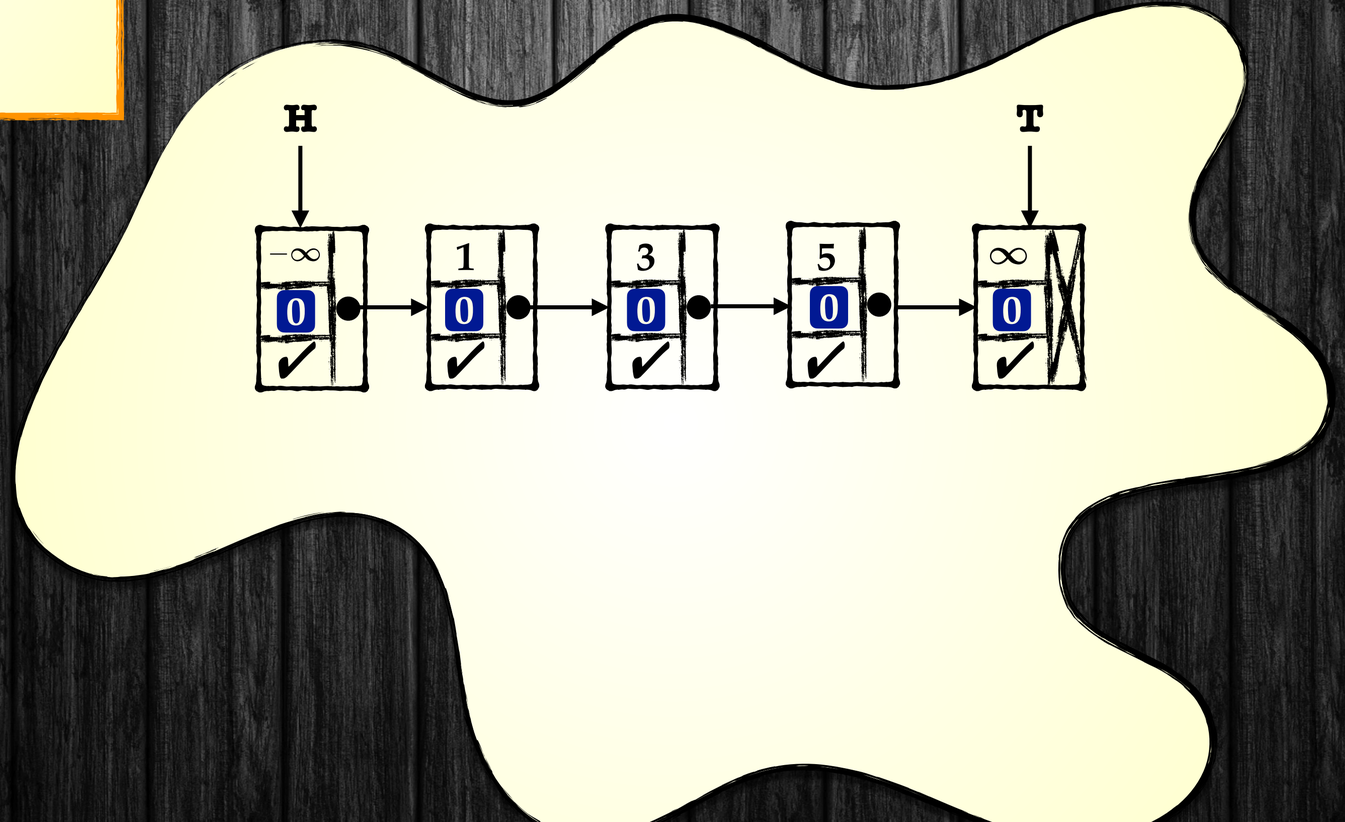
result

Boolean



add(4)

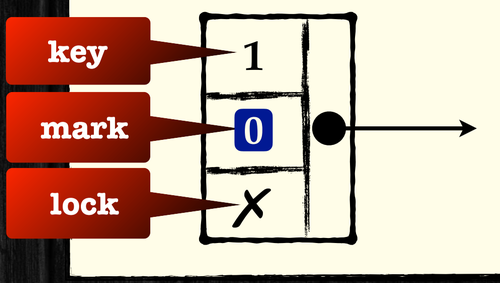


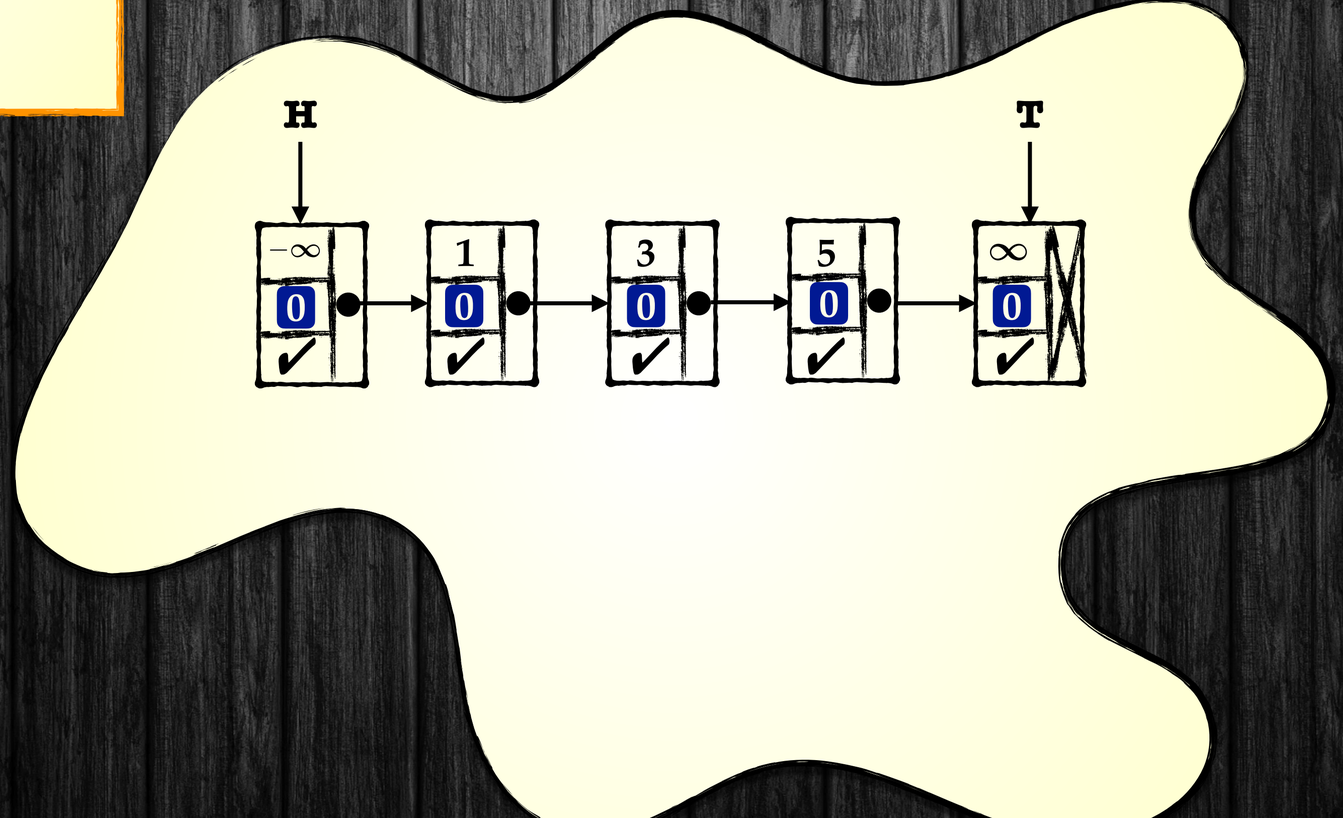


Sets | Lazy Synchronization

```
add(e):
   local p, c, n, r
1  (p,c) := locate(e);
2  if (c.val <> e)
3     n := new Node(0,e,c,false);
4     p.next := n;
5     r := true;
6  else r := false;
7  unlock(p);
8  unlock(c);
9  return r;
```



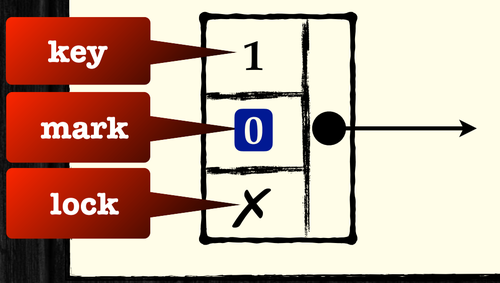


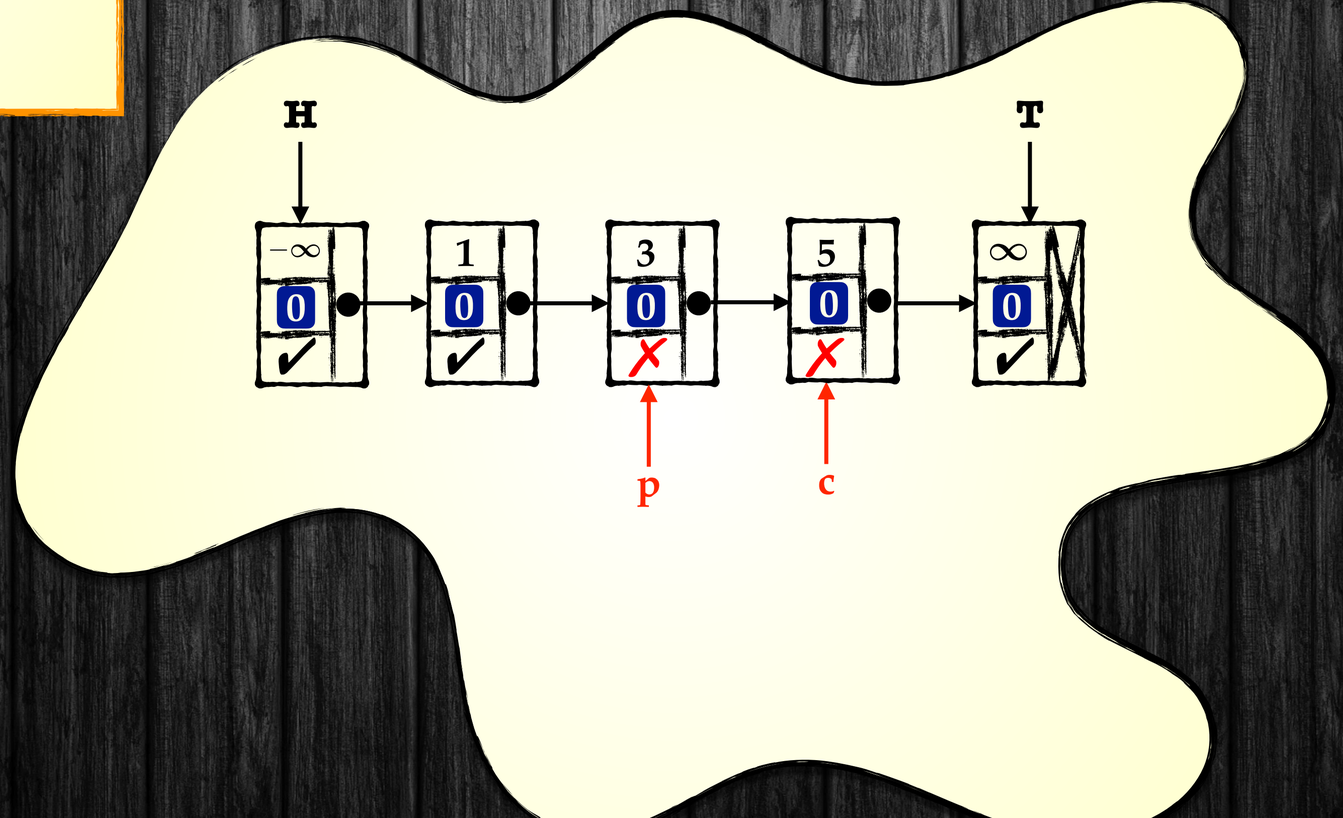


Sets | Lazy Synchronization

```
add(e):
   local p, c, n, r
1  (p,c) := locate(e);
2  if (c.val <> e)
3     n := new Node(0,e,c,false);
4     p.next := n;
5     r := true;
6  else r := false;
7  unlock(p);
8  unlock(c);
9  return r;
```



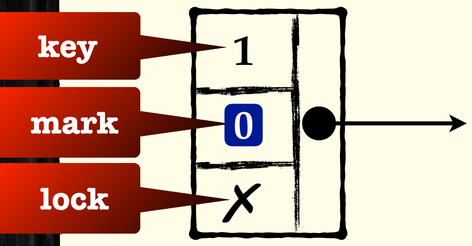


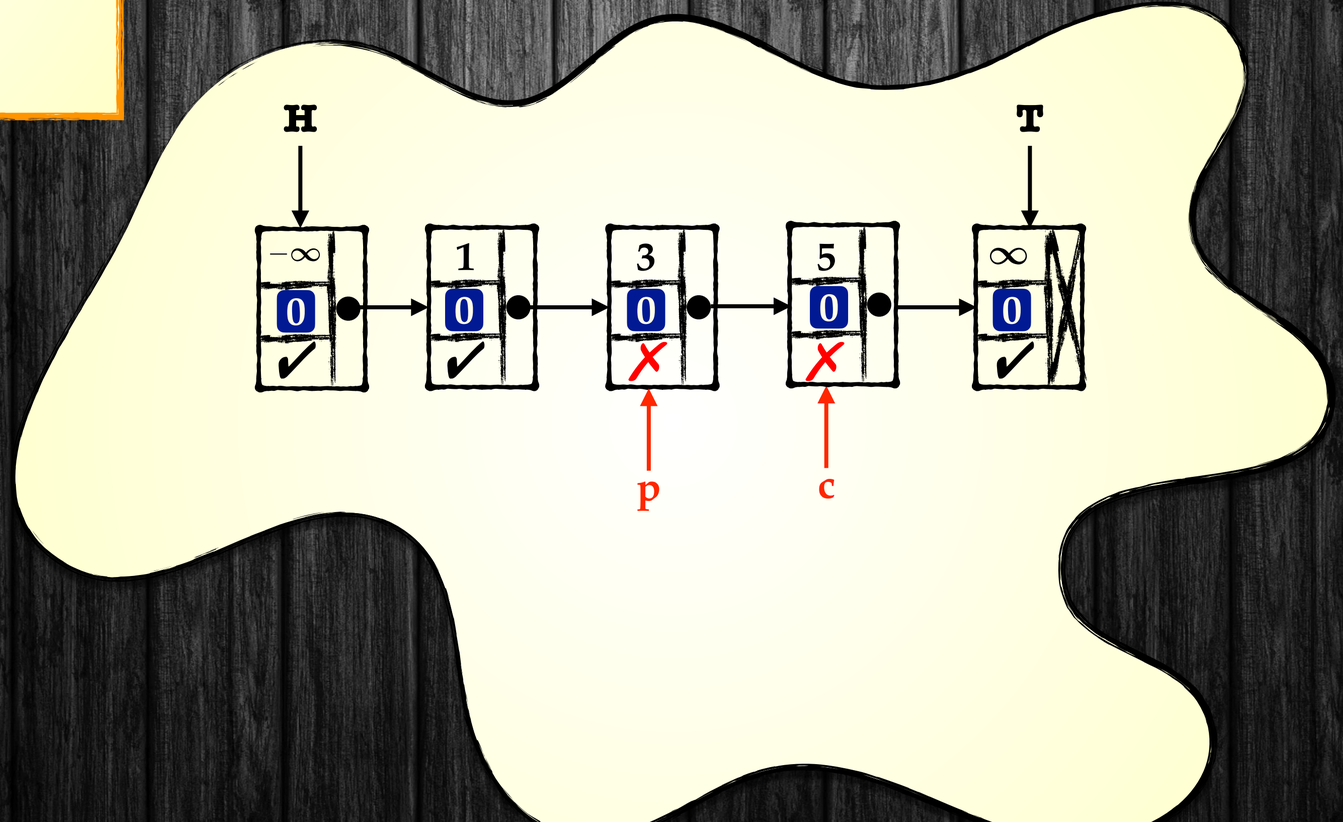


Sets | Lazy Synchronization

```
add(e):
   local p, c, n, r
1  (p,c) := locate(e);
2  if (c.val <> e)
3     n := new Node(0,e,c,false);
4     p.next := n;
5     r := true;
6  else r := false;
7  unlock(p);
8  unlock(c);
9  return r;
```

add(4)



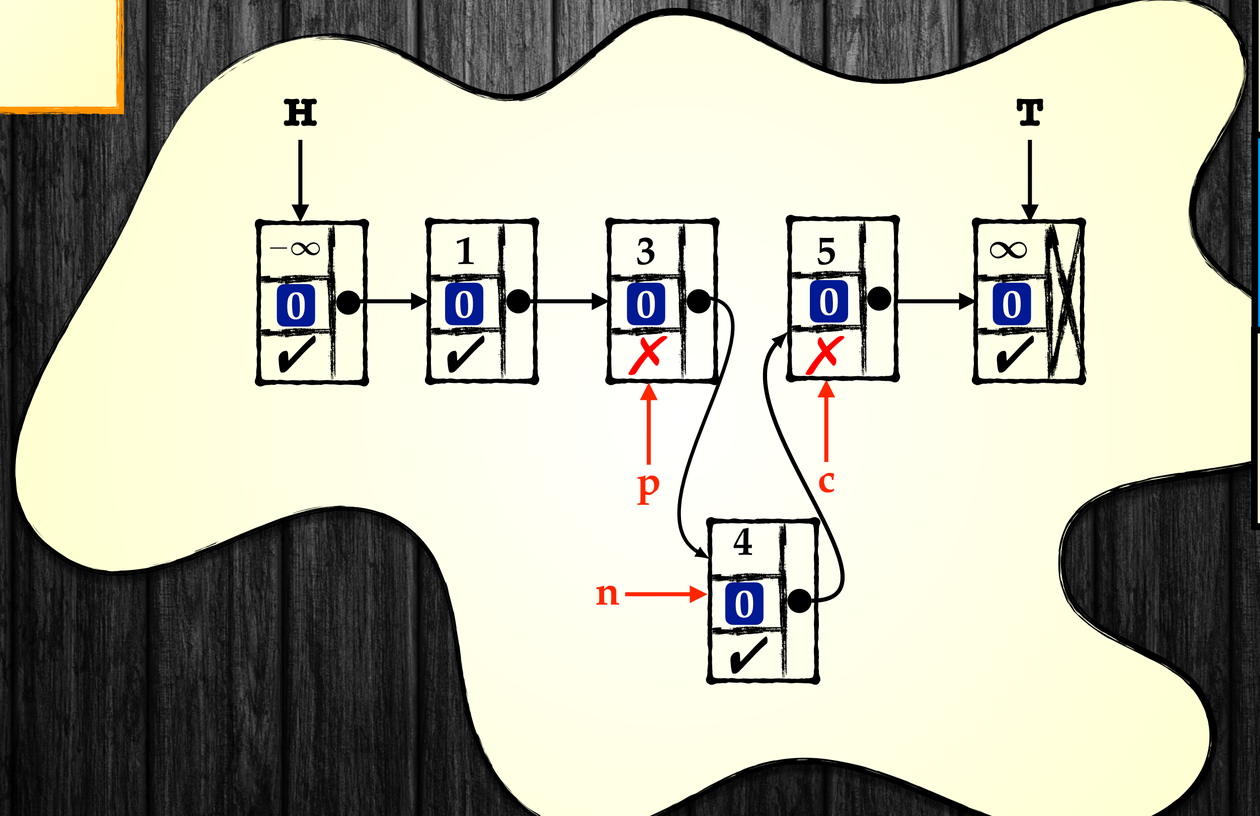


Sets | Lazy Synchronization

```
add(e):
   local p, c, n, r
1  (p,c) := locate(e);
2  if (c.val <> e)
3     n := new Node(0,e,c,false);
4     p.next := n;
5     r := true;
6  else r := false;
7  unlock(p);
8  unlock(c);
9  return r;
```

add(4)

linearization point

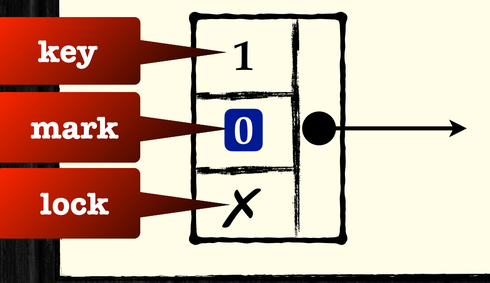


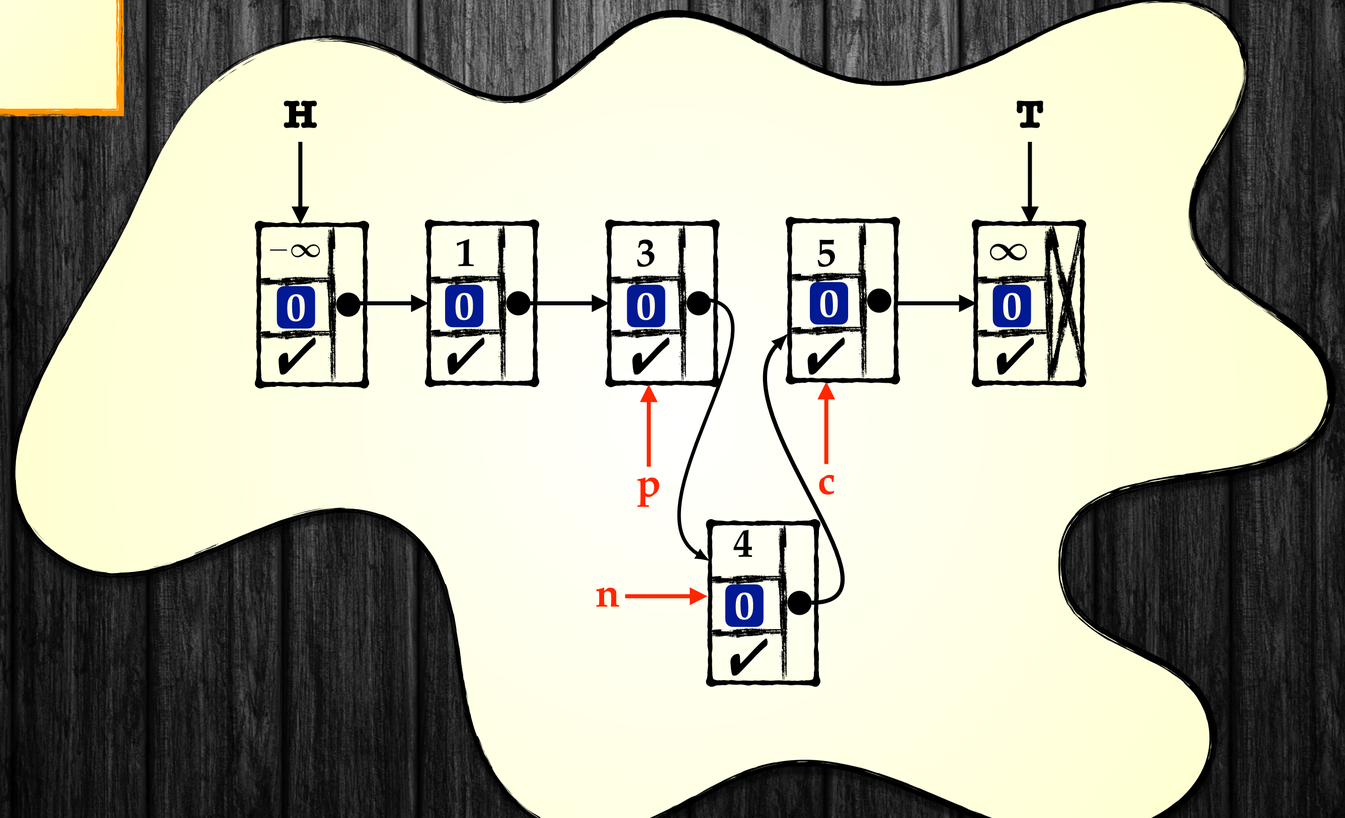
successful add	"when p.next is set"
unsuccessful add	"when unmarked entry is observed"
successful rmv	"when the cell is marked"
unsuccessful rmv	"first unmarked entry with larger key"
successful ctn	"when unmarked entry with input key found"
unsuccessful ctn	?

Sets | Lazy Synchronization

```
add(e):
   local p, c, n, r
1  (p,c) := locate(e);
2  if (c.val <> e)
3     n := new Node(0,e,c,false);
4     p.next := n;
5     r := true;
6  else r := false;
7  unlock(p);
8  unlock(c);
9  return r;
```

add(4)



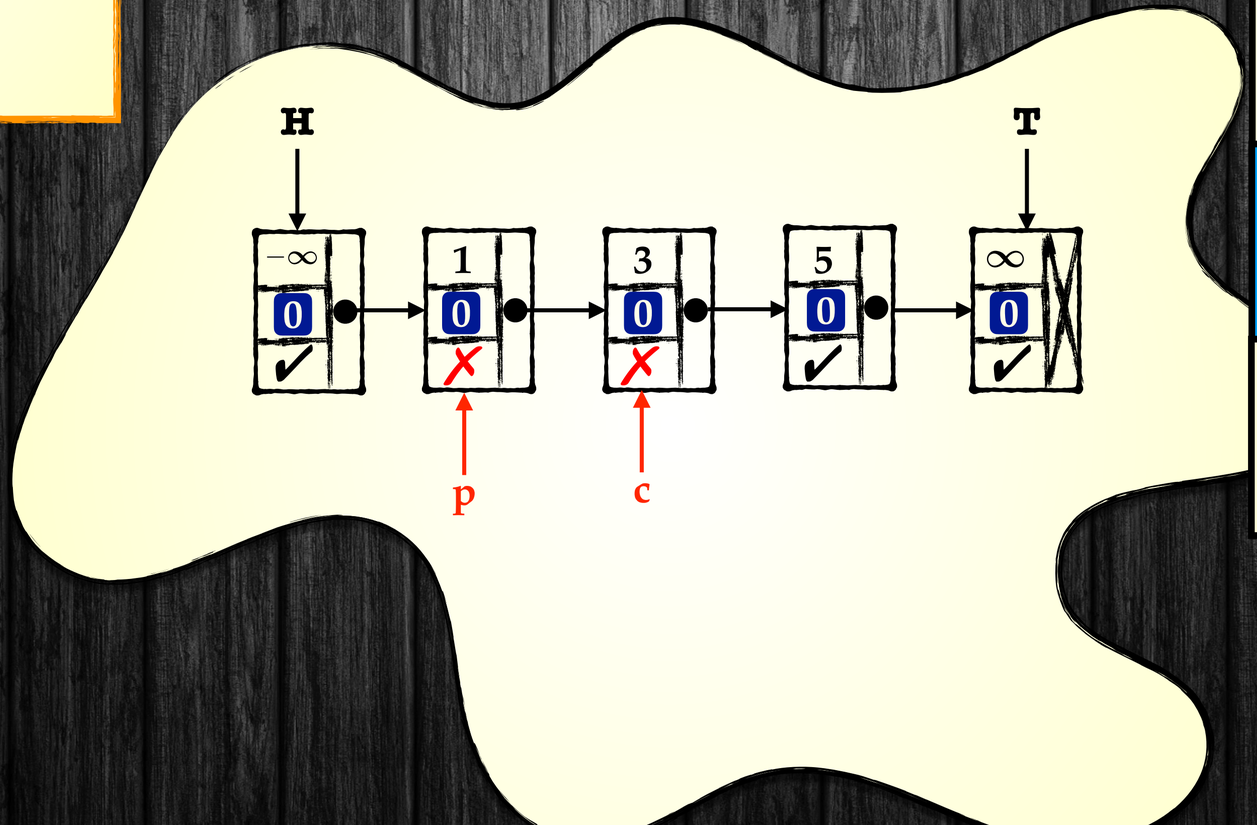


Sets | Lazy Synchronization

```
add(e):
   local p, c, n, r
1  (p,c) := locate(e);
2  if (c.val <> e)
3     n := new Node(0,e,c,false);
4     p.next := n;
5     r := true;
6  else r := false;
7  unlock(p);
8  unlock(c);
9  return r;
```

add(3)

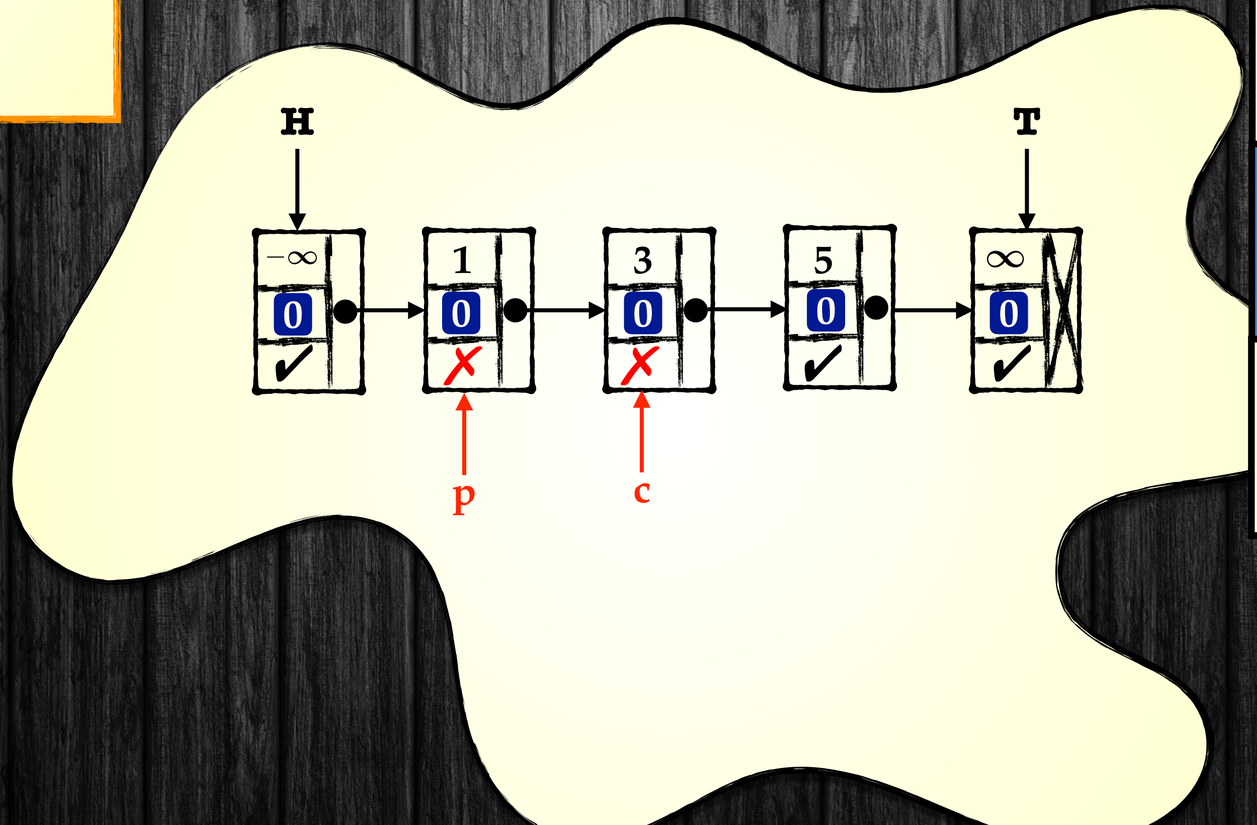
linearization point



successful add	“when p.next is set”
unsuccessful add	“when unmarked entry is observed”
successful rmv	“when the cell is marked”
unsuccessful rmv	“first unmarked entry with larger key”
successful ctn	“when unmarked entry with input key found”
unsuccessful ctn	?

Sets | Lazy Synchronization

```
add(e):
   local p, c, n, r
1  (p,c) := locate(e);
2  if (c.val <> e)
3     n := new Node(0,e,c,false);
4     p.next := n;
5     r := true;
6  else r := false;
7  unlock(p);
8  unlock(c);
9  return r;
```



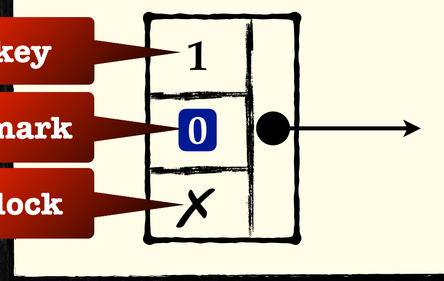
linearization point

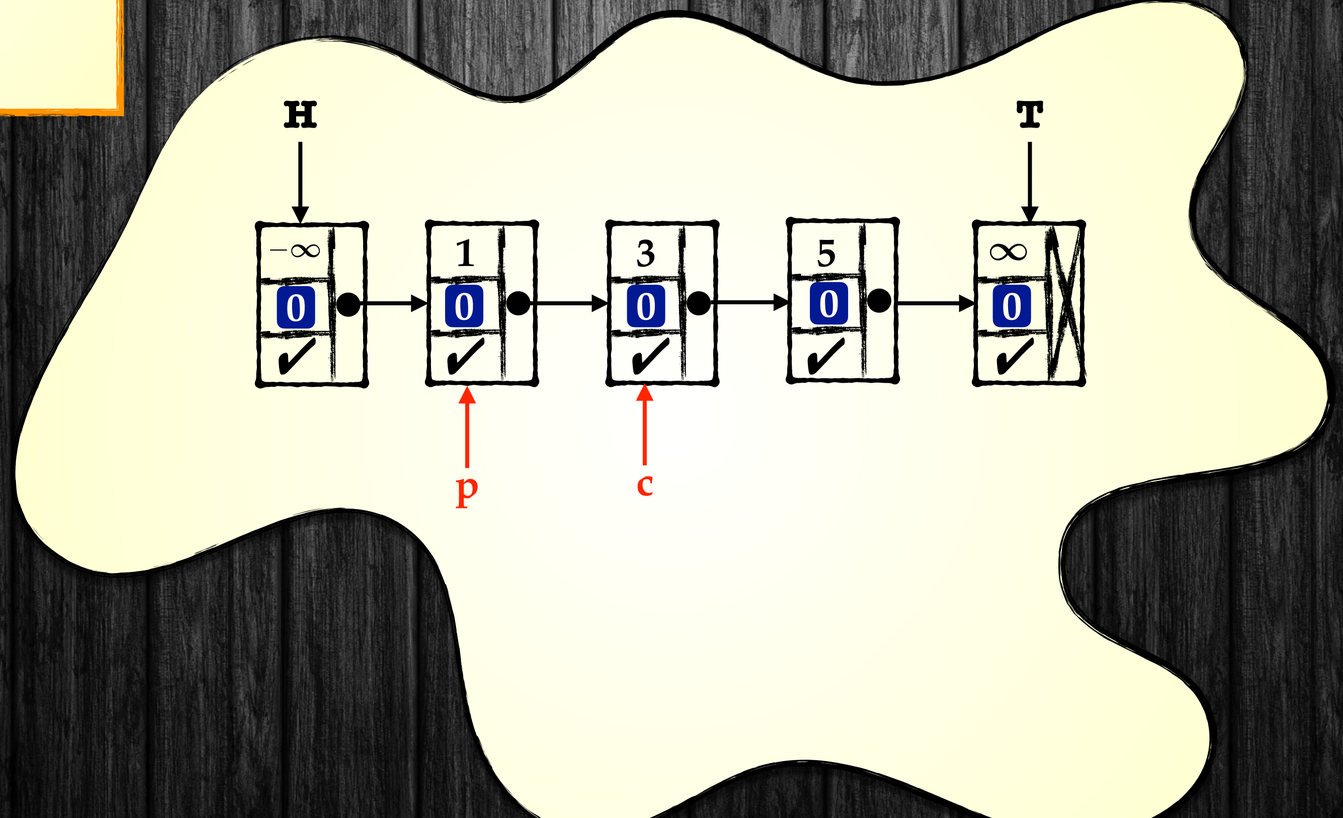
successful add	“when p.next is set”
unsuccessful add	“when unmarked entry is observed”
successful rmv	“when the cell is marked”
unsuccessful rmv	“first unmarked entry with larger key”
successful ctn	“when unmarked entry with input key found”
unsuccessful ctn	?

Sets | Lazy Synchronization

```
add(e):
   local p, c, n, r
1  (p,c) := locate(e);
2  if (c.val <> e)
3     n := new Node(0,e,c,false);
4     p.next := n;
5     r := true;
6  else r := false;
7  unlock(p);
8  unlock(c);
9  return r;
```

add(3)

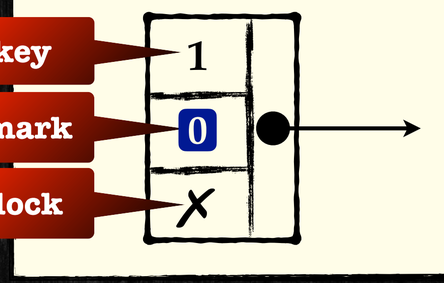


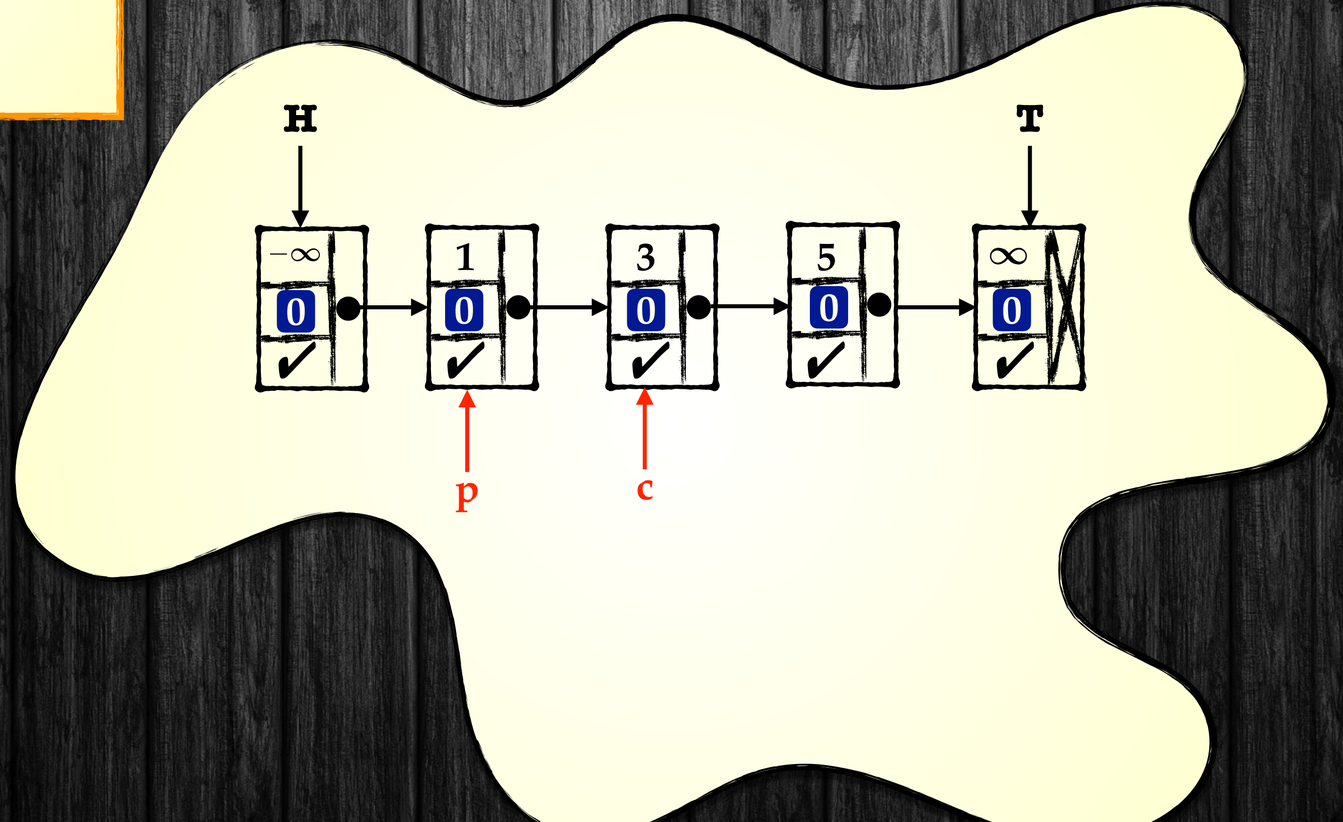


Sets | Lazy Synchronization

```
add(e):
   local p, c, n, r
1  (p,c) := locate(e);
2  if (c.val <> e)
3     n := new Node(0,e,c,false);
4     p.next := n;
5     r := true;
6  else r := false;
7  unlock(p);
8  unlock(c);
9  return r;
```

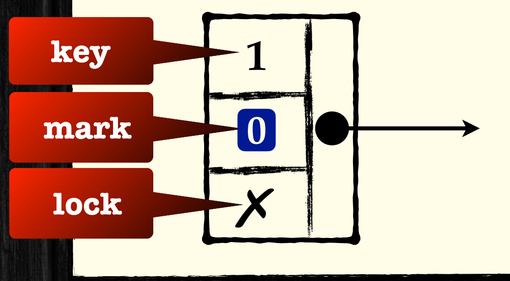
add(3)

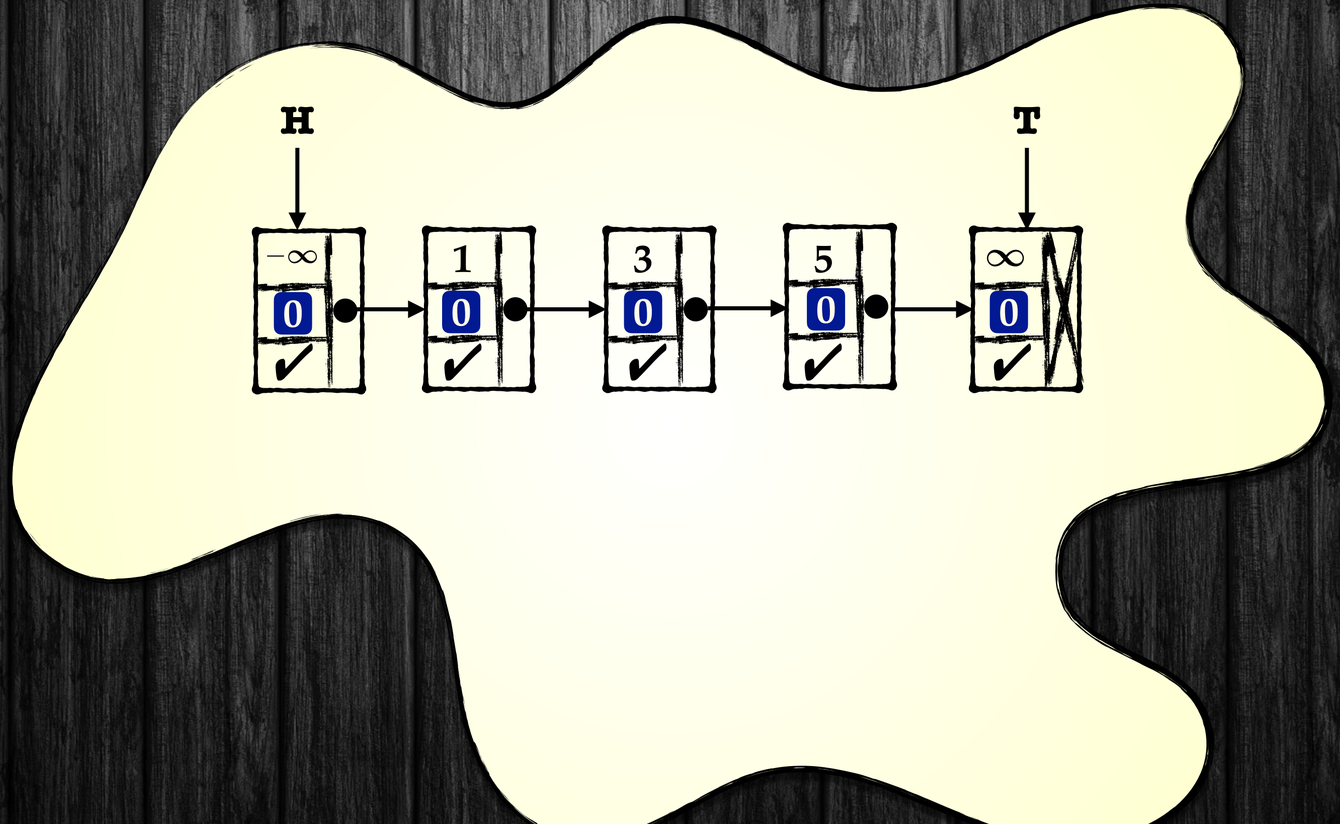




```
rmv(e):
local p, c, n, r
1  (p,c) := locate(e);
2  if (c.val = e)
3     c.mark := true
4     n := c.next;
5     p.next := n;
6     r := true;
7  else r := false;
8  unlock(p);
9  unlock(c);
10 return r;
```

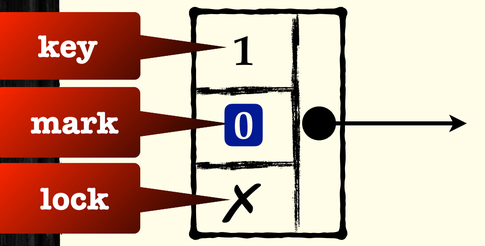


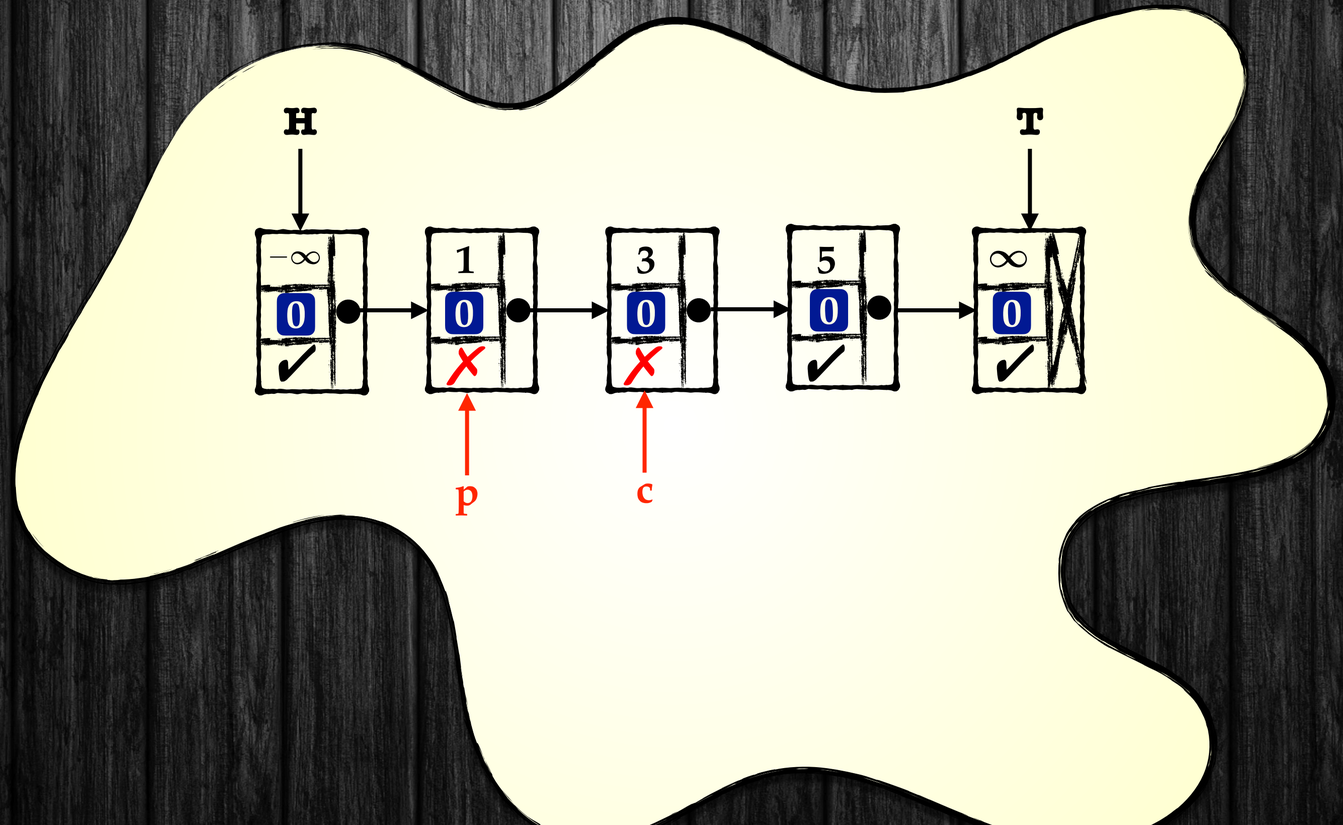




```
rmv(e):
local p, c, n, r
1  (p,c) := locate(e);
2  if (c.val = e)
3     c.mark := true
4     n := c.next;
5     p.next := n;
6     r := true;
7  else r := false;
8  unlock(p);
9  unlock(c);
10 return r;
```

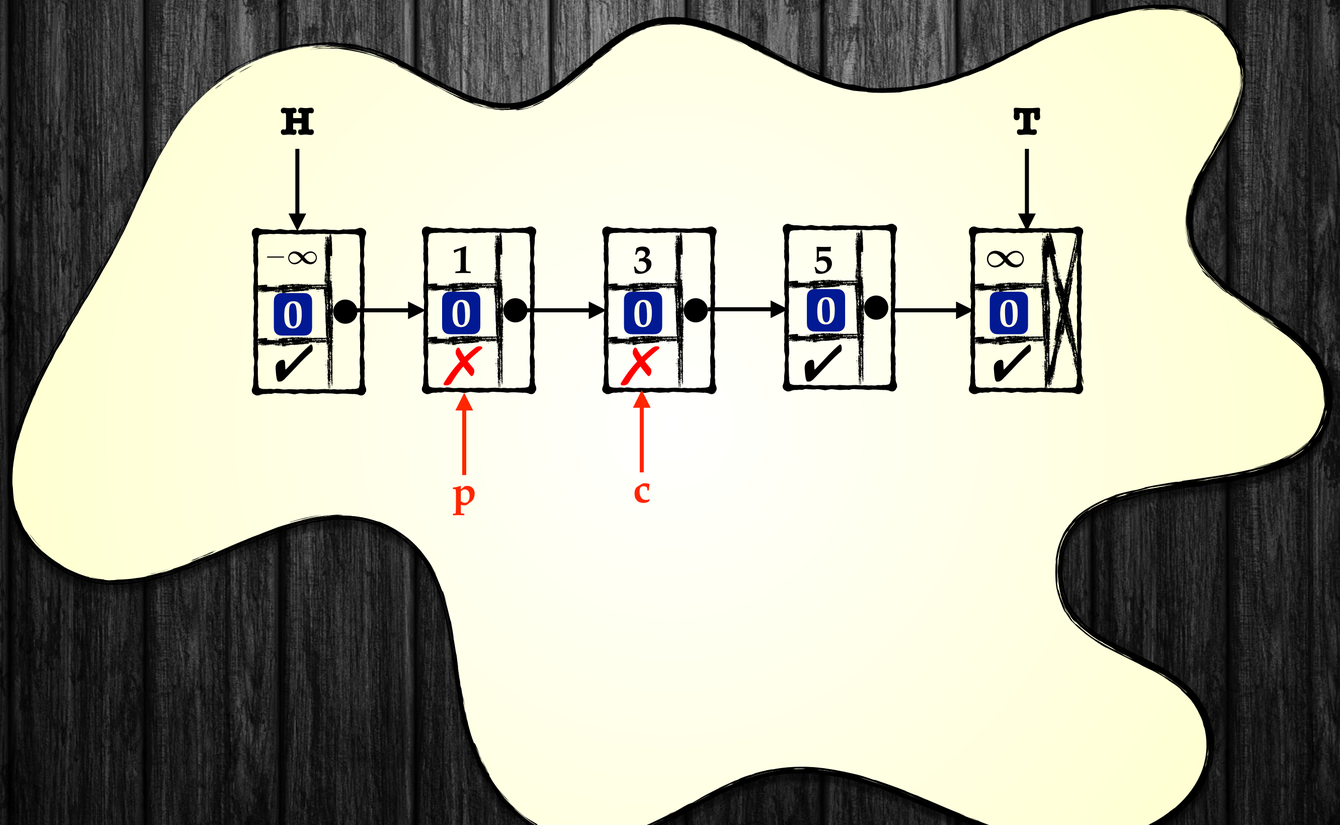
rmv(3)

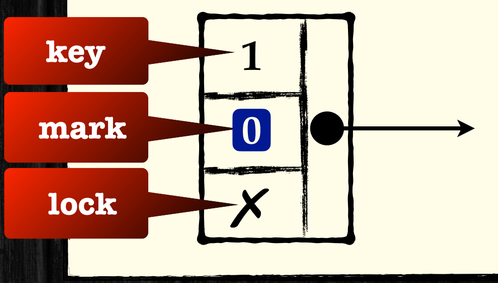




```
rmv(e):
local p, c, n, r
1  (p,c) := locate(e);
2  if (c.val = e)
3     c.mark := true
4     n := c.next;
5     p.next := n;
6     r := true;
7  else r := false;
8  unlock(p);
9  unlock(c);
10 return r;
```

rmv(3)

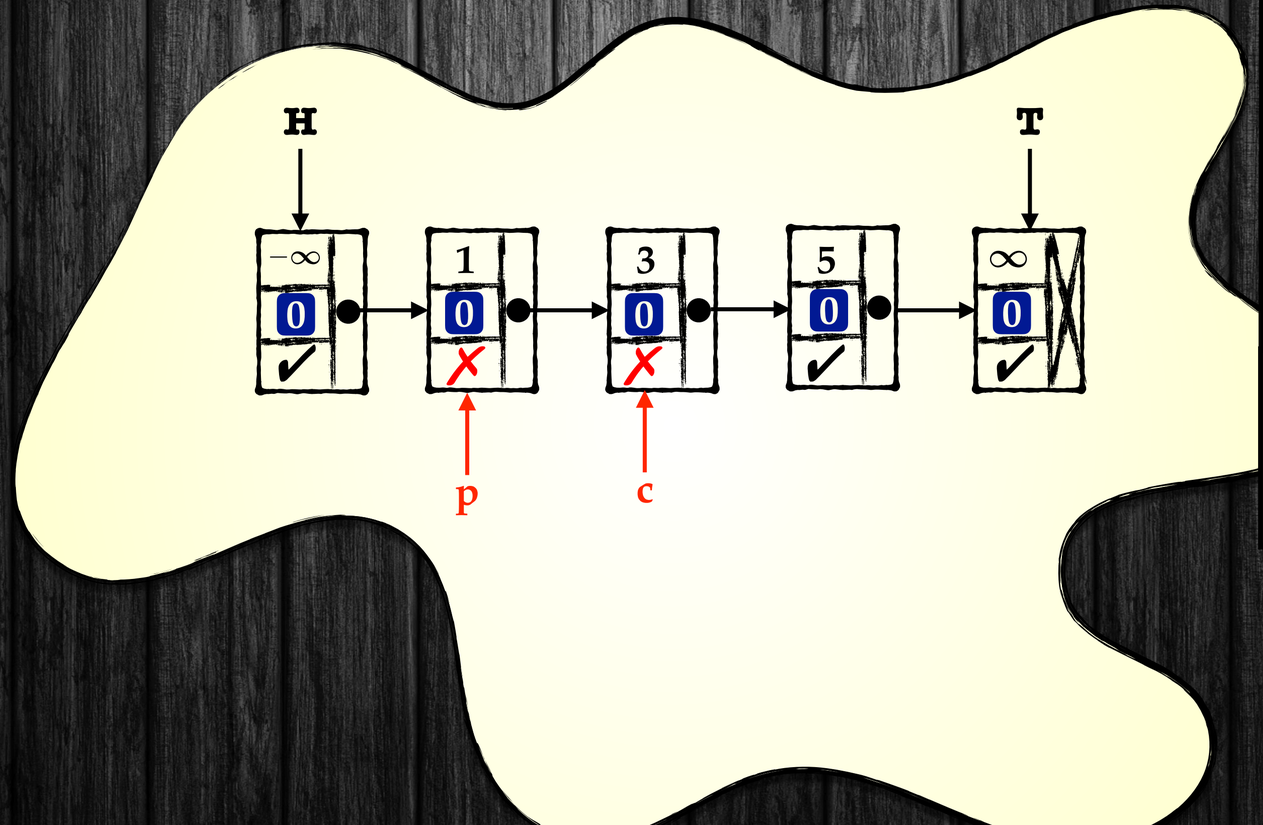




```
rmv(e):
local p, c, n, r
1  (p,c) := locate(e);
2  if (c.val = e)
3     c.mark := true
4     n := c.next;
5     p.next := n;
6     r := true;
7  else r := false;
8  unlock(p);
9  unlock(c);
10 return r;
```

rmv(3)

linearization point



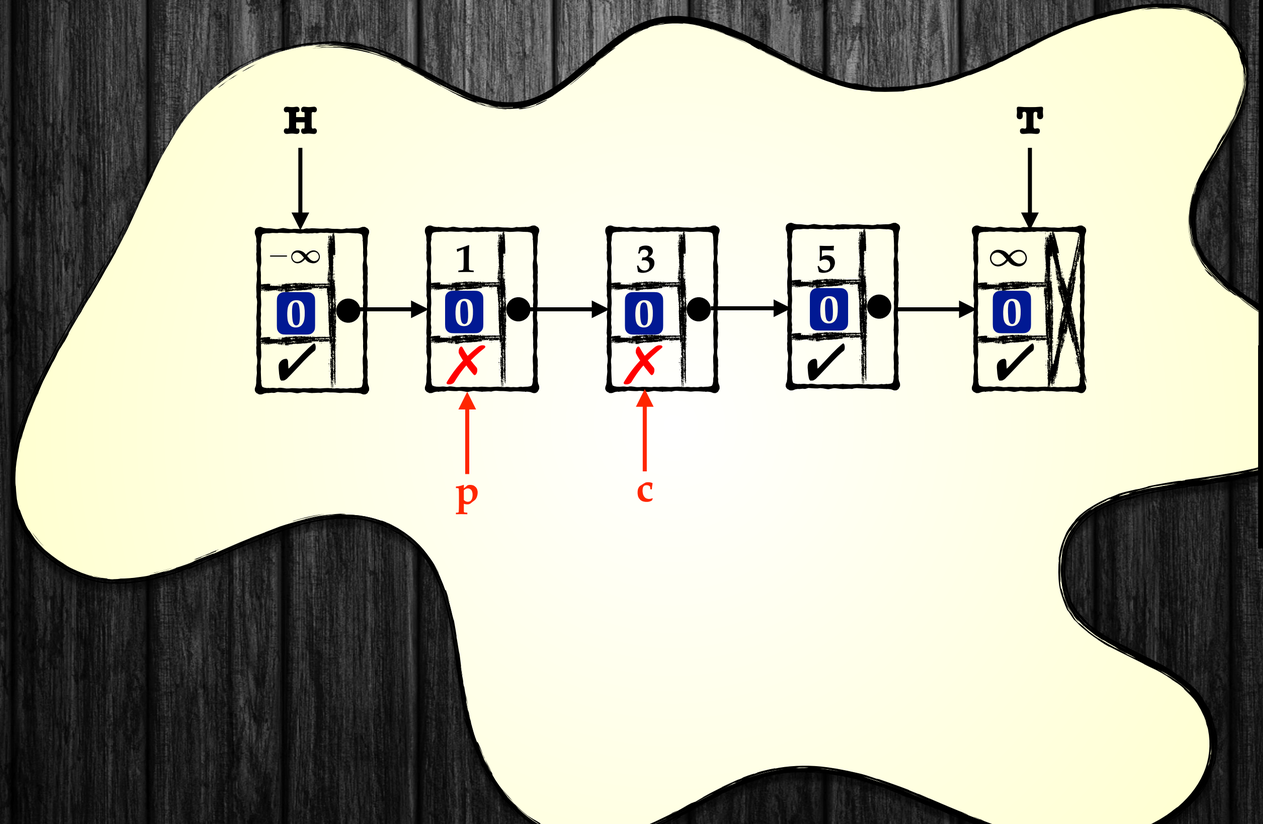
successful add	“when p.next is set”
unsuccessful add	“when unmarked entry is observed”
successful rmv	“when the cell is marked”
unsuccessful rmv	“first unmarked entry with larger key”
successful ctn	“when unmarked entry with input key found”
unsuccessful ctn	?

```
rmv(e):
local p, c, n, r
1  (p,c) := locate(e);
2  if (c.val = e)
3     c.mark := true
4     n := c.next;
5     p.next := n;
6     r := true;
7  else r := false;
8  unlock(p);
9  unlock(c);
10 return r;
```

rmv(3)

logical deletion

linearization point



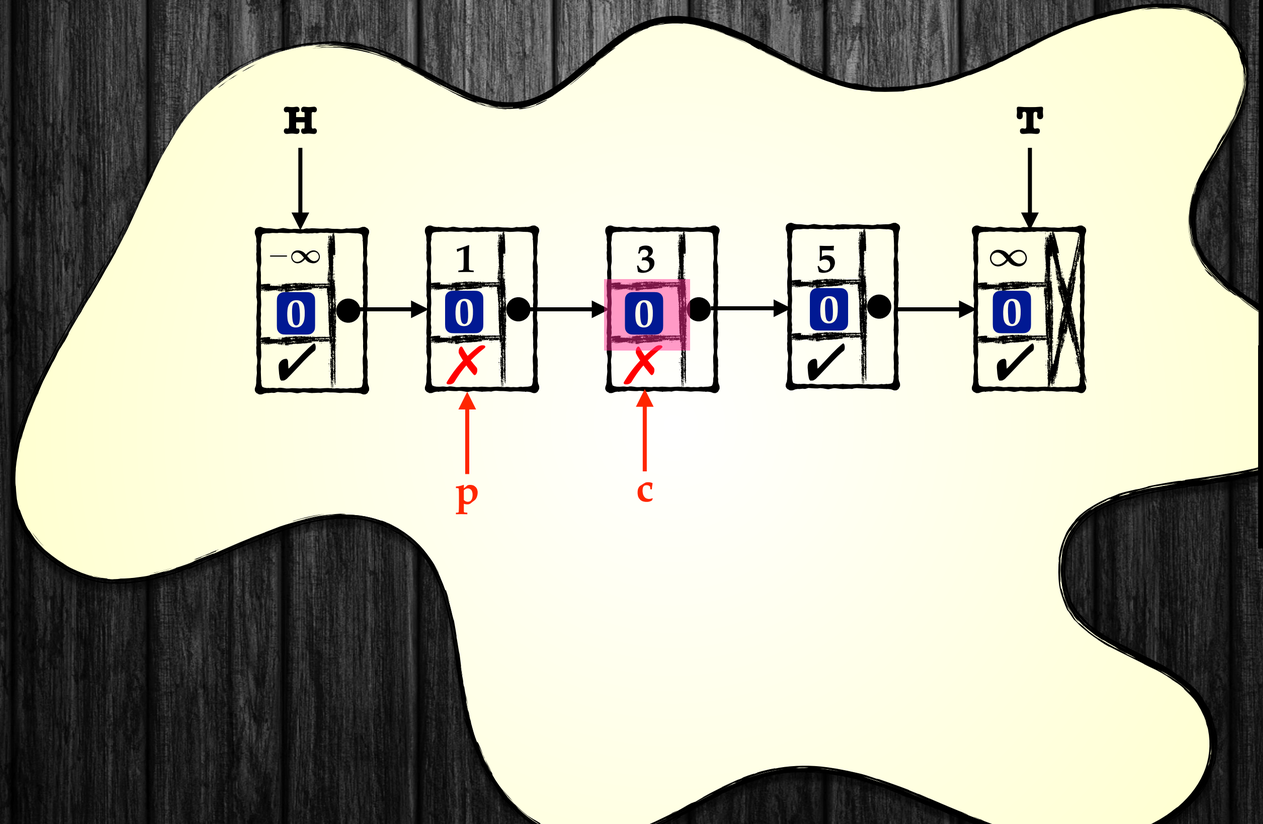
successful add	“when p.next is set”
unsuccessful add	“when unmarked entry is observed”
successful rmv	“when the cell is marked”
unsuccessful rmv	“first unmarked entry with larger key”
successful ctn	“when unmarked entry with input key found”
unsuccessful ctn	?

```
rmv(e):
local p, c, n, r
1  (p,c) := locate(e);
2  if (c.val = e)
3     c.mark := true
4     n := c.next;
5     p.next := n;
6     r := true;
7  else r := false;
8  unlock(p);
9  unlock(c);
10 return r;
```

rmv(3)

logical deletion

linearization point



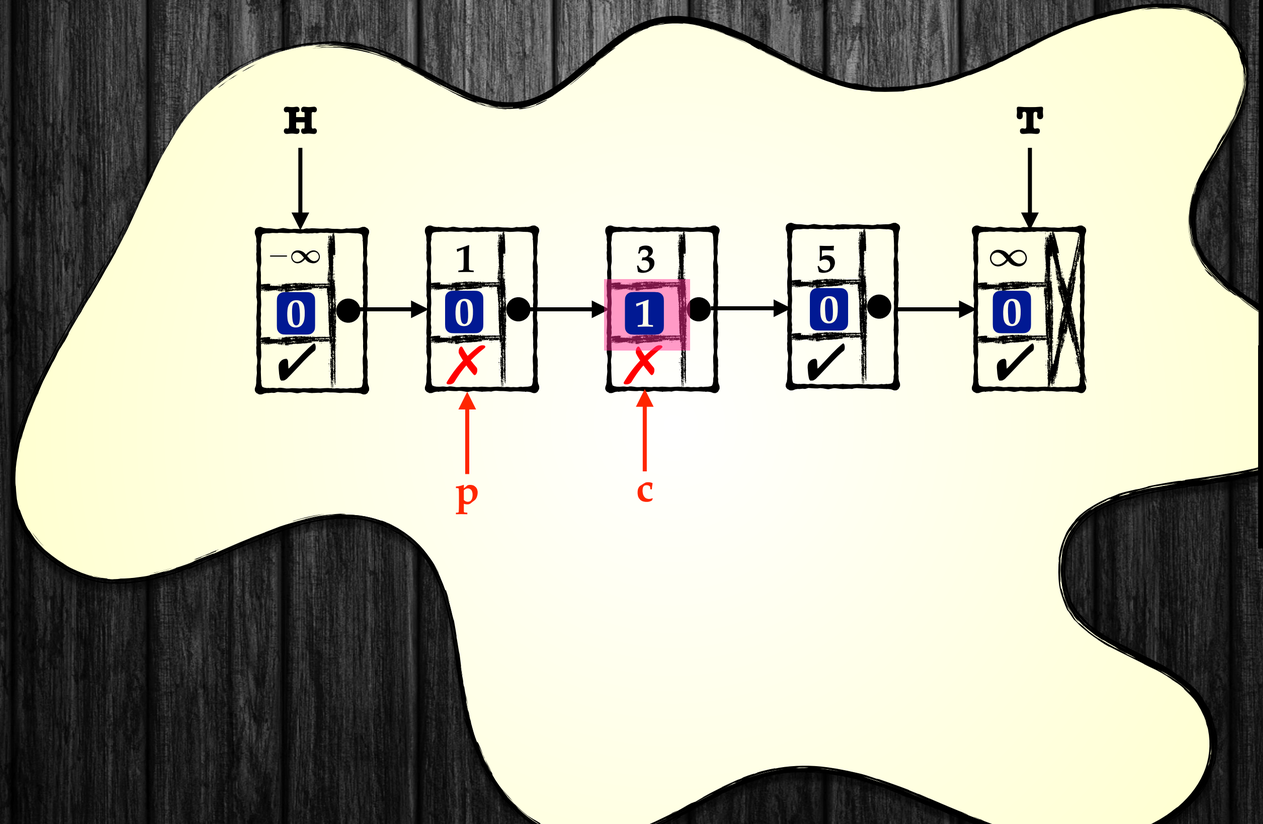
successful add	“when p.next is set”
unsuccessful add	“when unmarked entry is observed”
successful rmv	“when the cell is marked”
unsuccessful rmv	“first unmarked entry with larger key”
successful ctn	“when unmarked entry with input key found”
unsuccessful ctn	?

```
rmv(e):
local p, c, n, r
1  (p,c) := locate(e);
2  if (c.val = e)
3     c.mark := true
4     n := c.next;
5     p.next := n;
6     r := true;
7  else r := false;
8  unlock(p);
9  unlock(c);
10 return r;
```

rmv(3)

logical deletion

linearization point



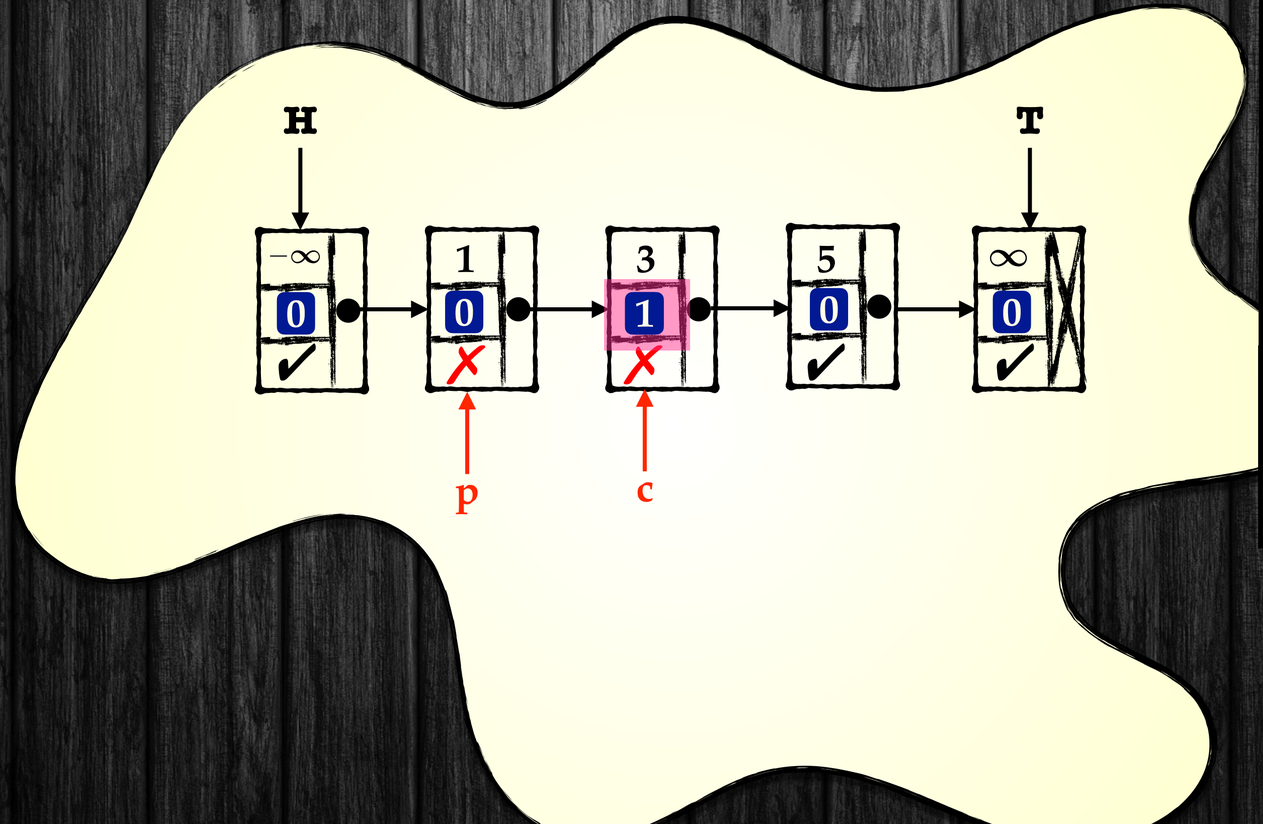
successful add	“when p.next is set”
unsuccessful add	“when unmarked entry is observed”
successful rmv	“when the cell is marked”
unsuccessful rmv	“first unmarked entry with larger key”
successful ctn	“when unmarked entry with input key found”
unsuccessful ctn	?

```
rmv(e):
local p, c, n, r
1  (p,c) := locate(e);
2  if (c.val = e)
3     c.mark := true
4     n := c.next;
5     p.next := n;
6     r := true;
7  else r := false;
8  unlock(p);
9  unlock(c);
10 return r;
```

rmv(3)

logical deletion

linearization point

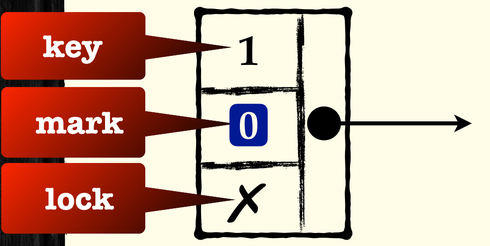


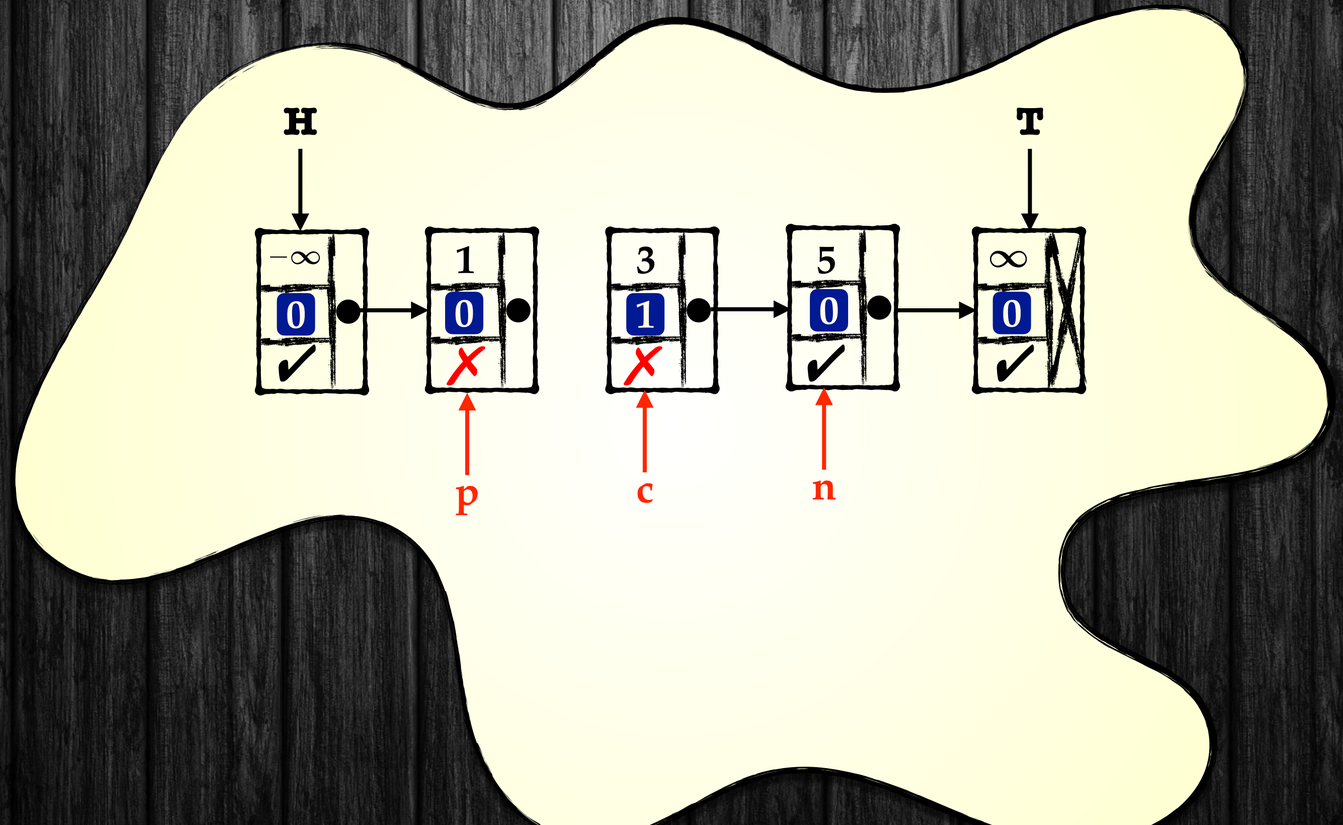
successful add	“when p.next is set”
unsuccessful add	“when unmarked entry is observed”
successful rmv	“when the cell is marked”
unsuccessful rmv	“first unmarked entry with larger key”
successful ctn	“when unmarked entry with input key found”
unsuccessful ctn	?

```
rmv(e):
local p, c, n, r
1  (p,c) := locate(e);
2  if (c.val = e)
3     c.mark := true
4     n := c.next;
5     p.next := n;
6     r := true;
7  else r := false;
8  unlock(p);
9  unlock(c);
10 return r;
```

rmv(3)

physical deletion

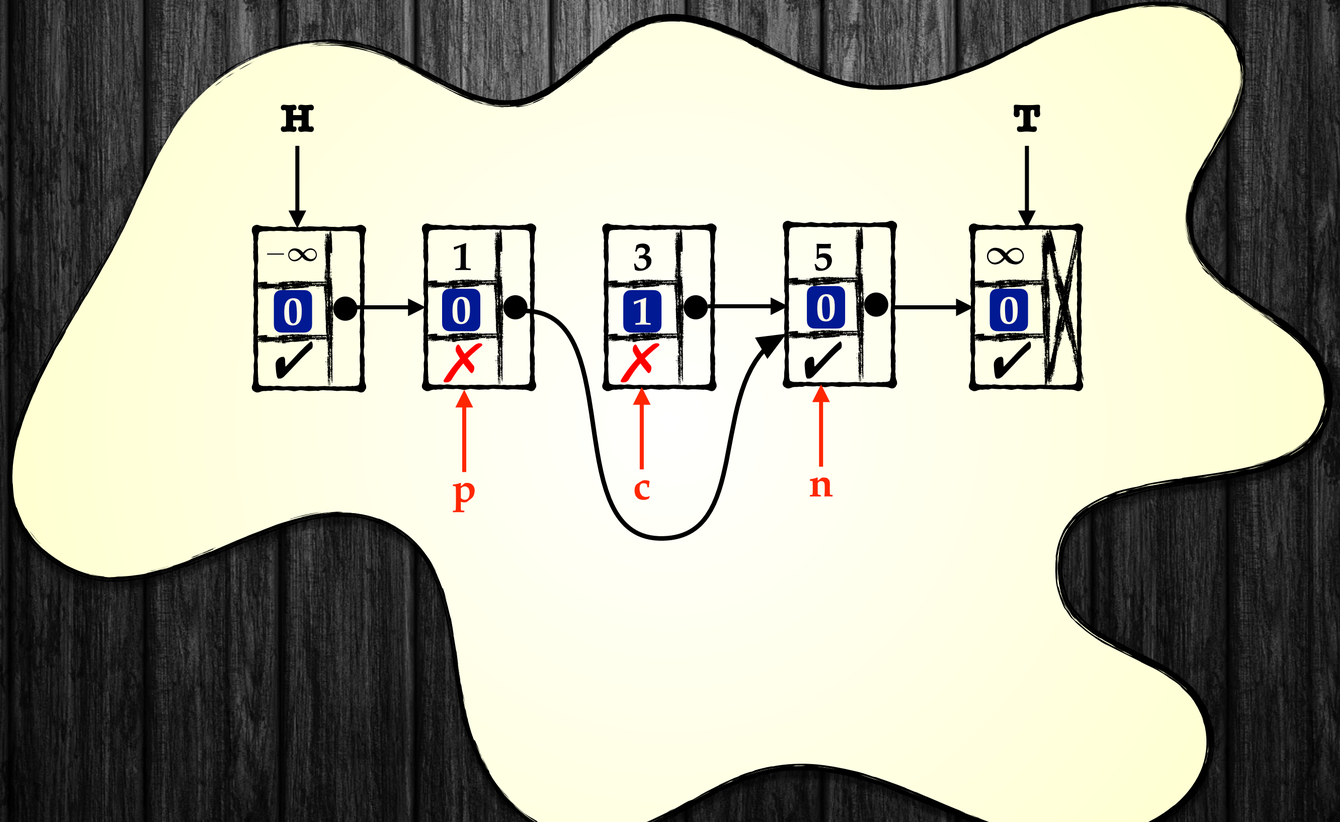


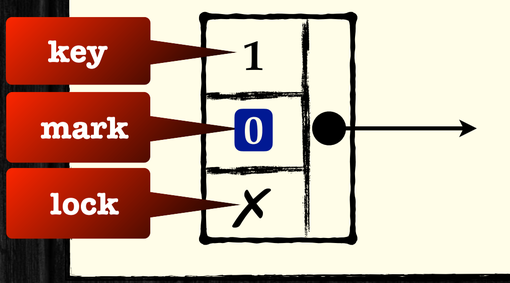


```
rmv(e):
local p, c, n, r
1  (p,c) := locate(e);
2  if (c.val = e)
3     c.mark := true
4     n := c.next;
5     p.next := n;
6     r := true;
7  else r := false;
8  unlock(p);
9  unlock(c);
10 return r;
```

rmv(3)

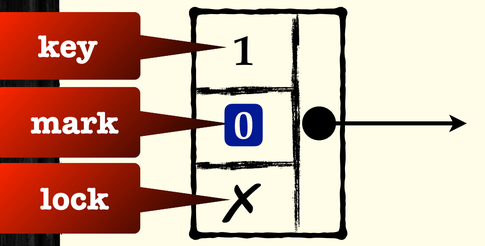
physical deletion

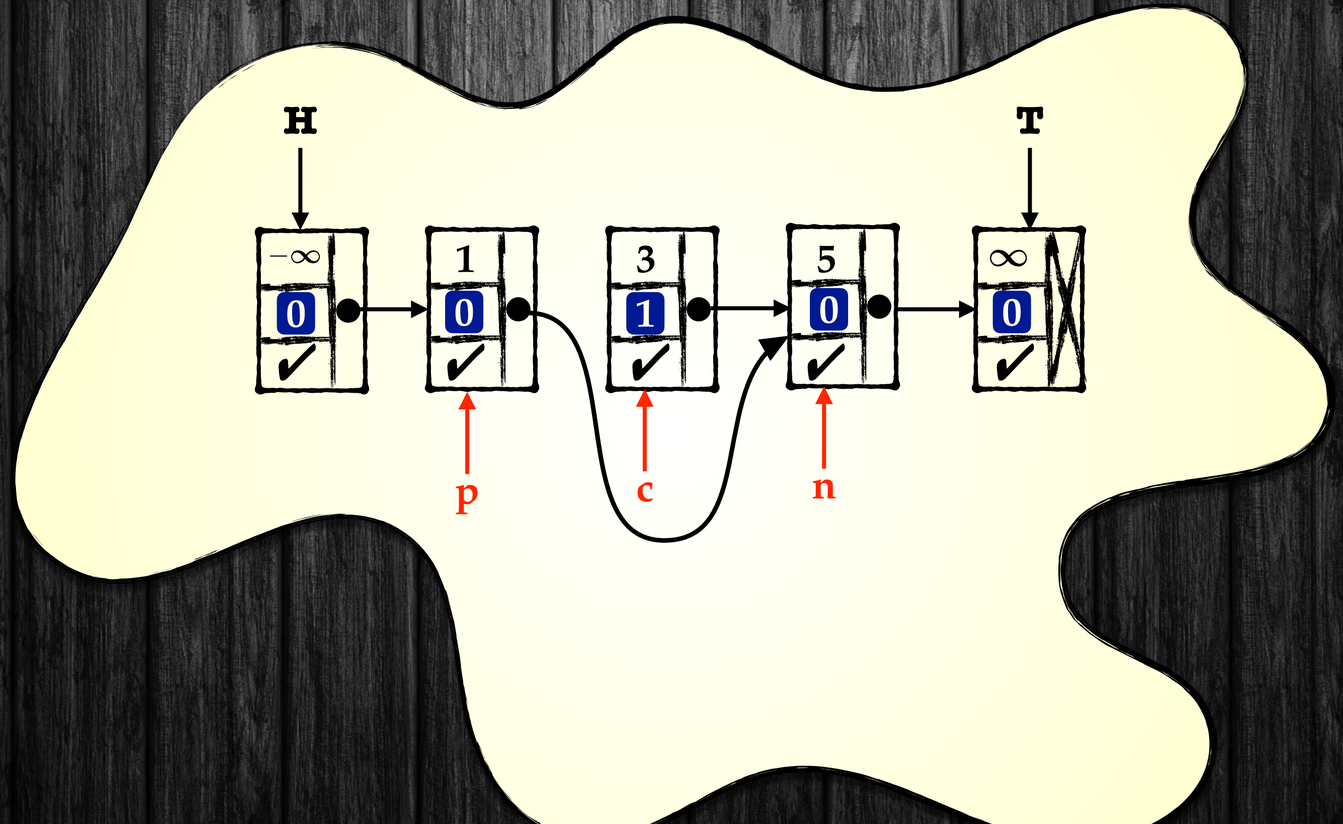




```
rmv(e):
local p, c, n, r
1  (p,c) := locate(e);
2  if (c.val = e)
3     c.mark := true
4     n := c.next;
5     p.next := n;
6     r := true;
7  else r := false;
8  unlock(p);
9  unlock(c);
10 return r;
```

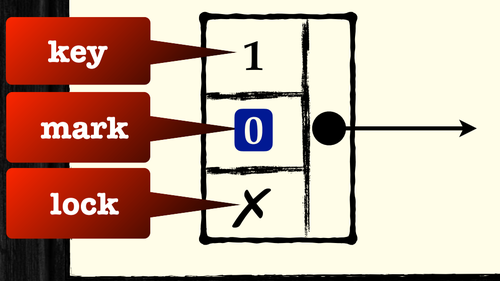
rmv(3)

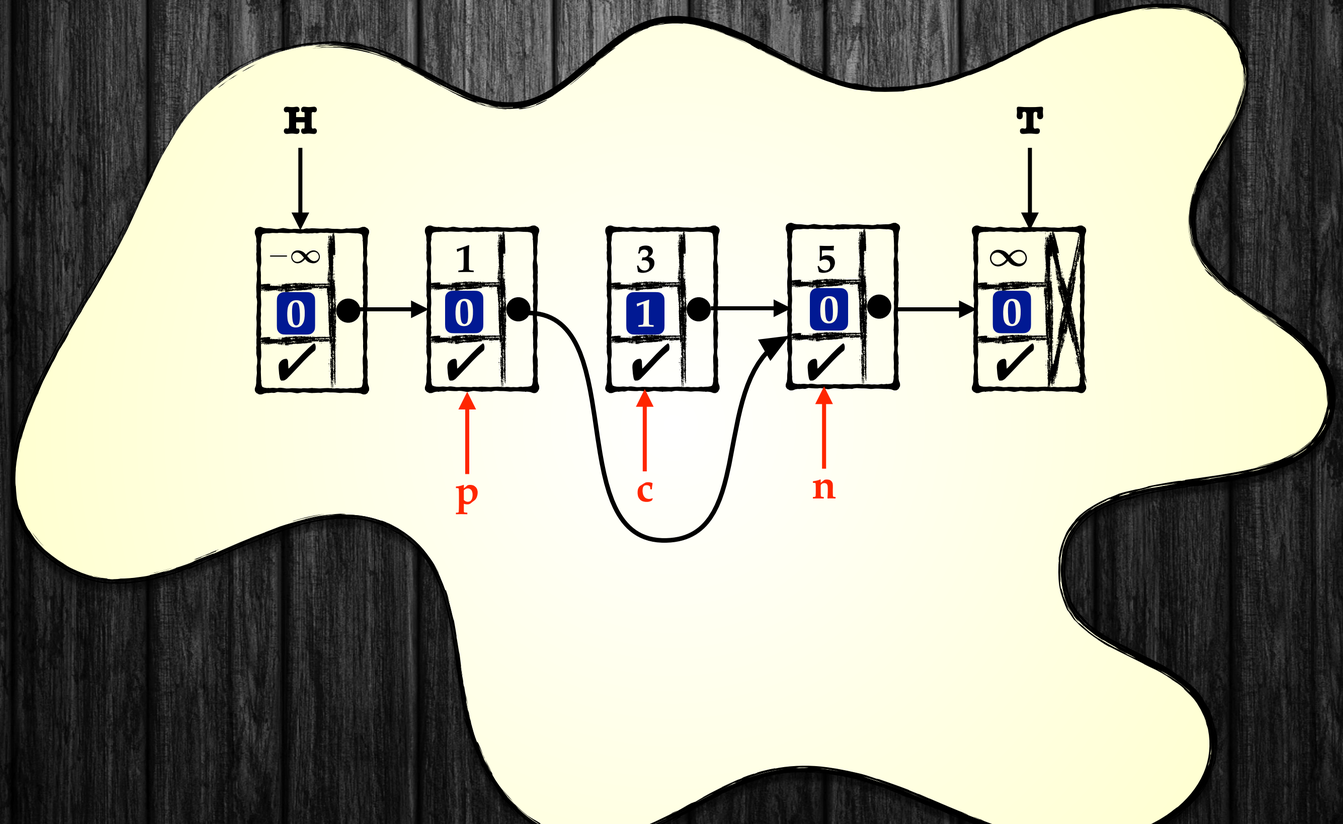




```
rmv(e):
local p, c, n, r
1  (p,c) := locate(e);
2  if (c.val = e)
3     c.mark := true
4     n := c.next;
5     p.next := n;
6     r := true;
7  else r := false;
8  unlock(p);
9  unlock(c);
10 return r;
```

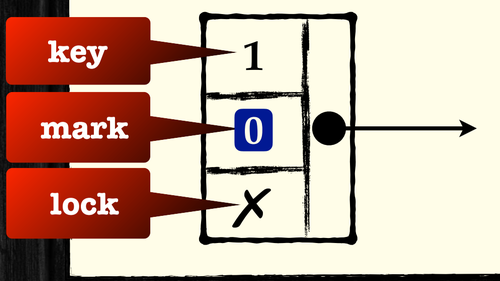


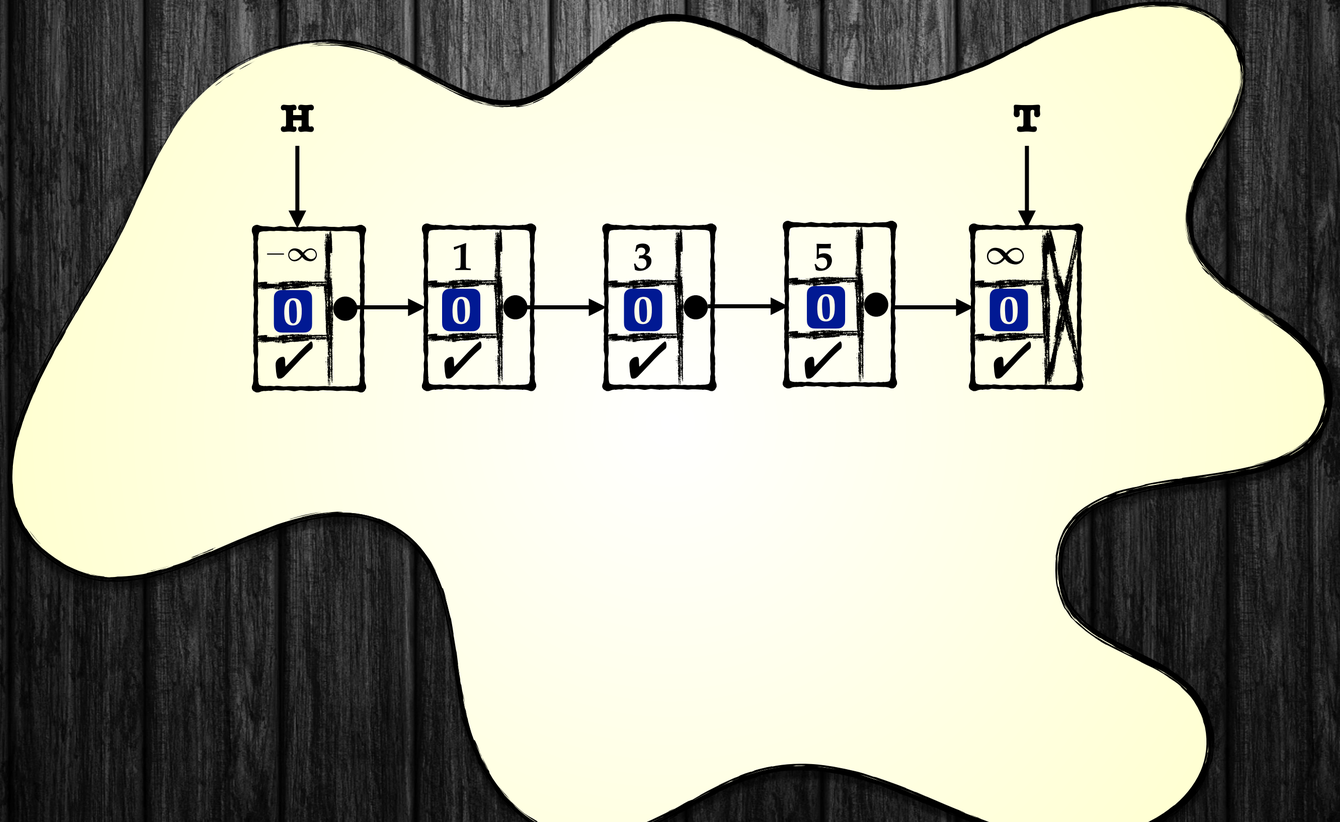




```
rmv(e):
local p, c, n, r
1  (p,c) := locate(e);
2  if (c.val = e)
3     c.mark := true
4     n := c.next;
5     p.next := n;
6     r := true;
7  else r := false;
8  unlock(p);
9  unlock(c);
10 return r;
```

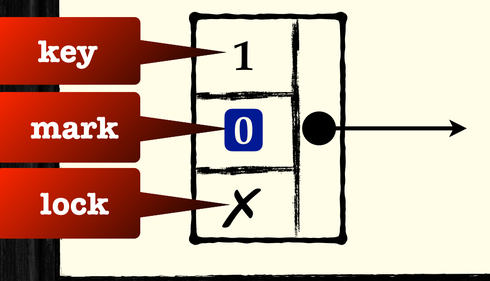


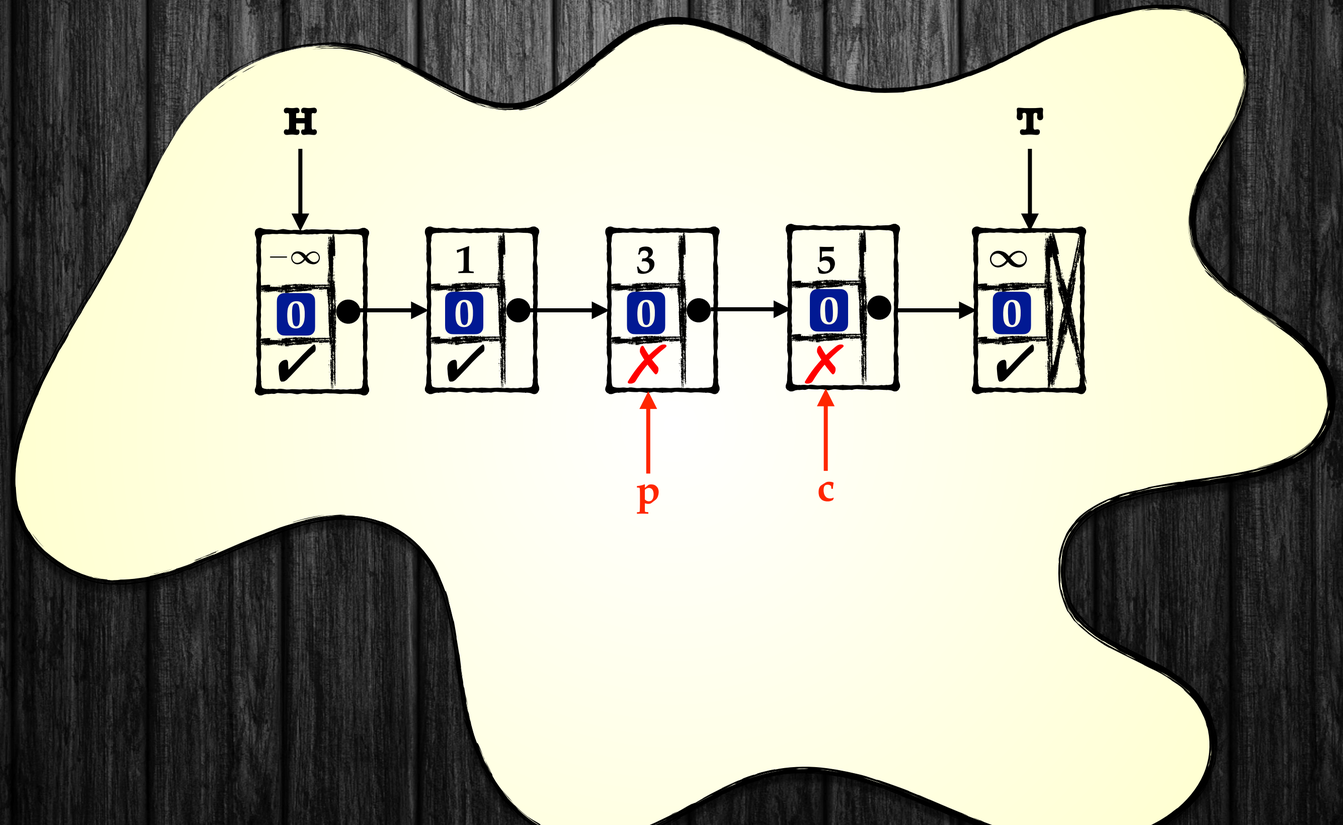




```
rmv(e):
local p, c, n, r
1  (p,c) := locate(e);
2  if (c.val = e)
3     c.mark := true
4     n := c.next;
5     p.next := n;
6     r := true;
7  else r := false;
8  unlock(p);
9  unlock(c);
10 return r;
```

rmv(4)

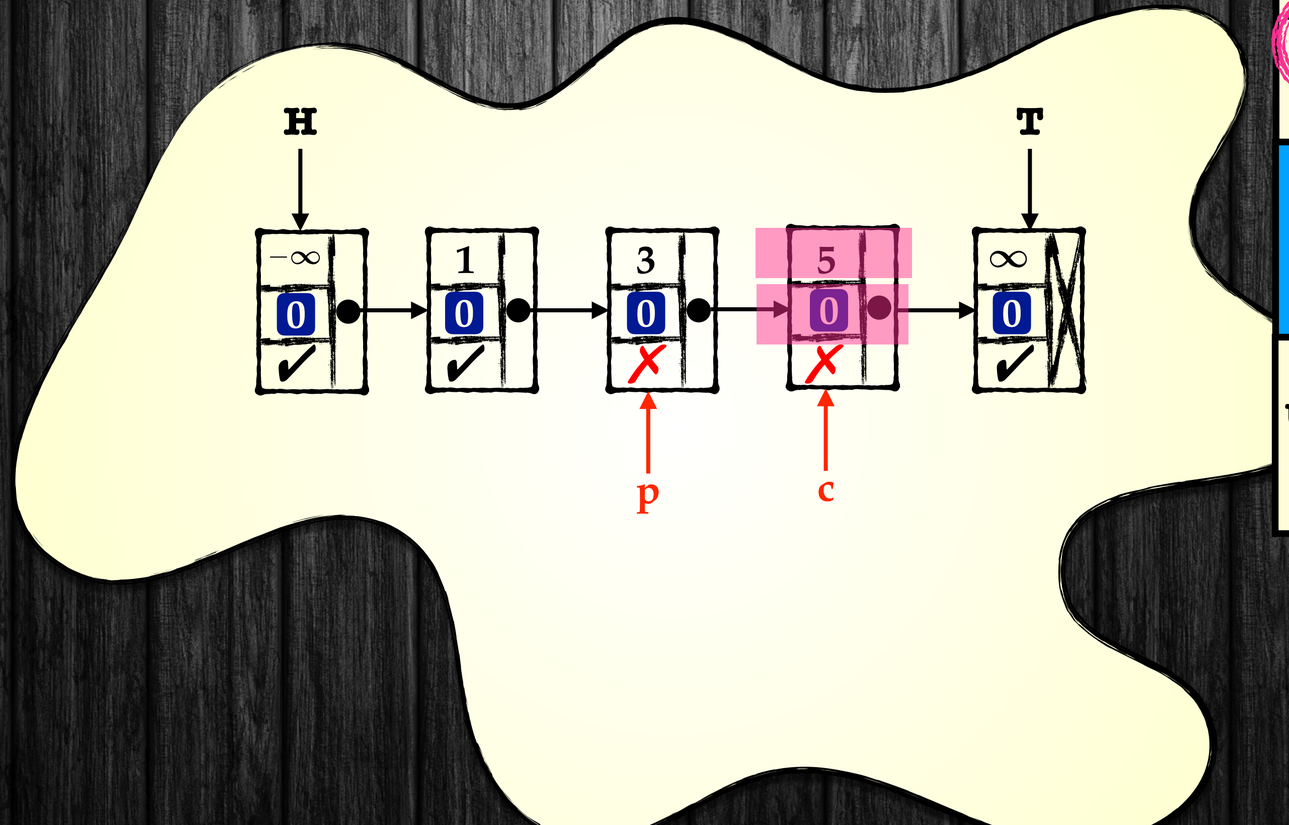




```
rmv(e):
local p, c, n, r
1  (p,c) := locate(e);
2  if (c.val = e)
3     c.mark := true
4     n := c.next;
5     p.next := n;
6     r := true;
7  else r := false;
8  unlock(p);
9  unlock(c);
10 return r;
```

rmv(4)

linearization point

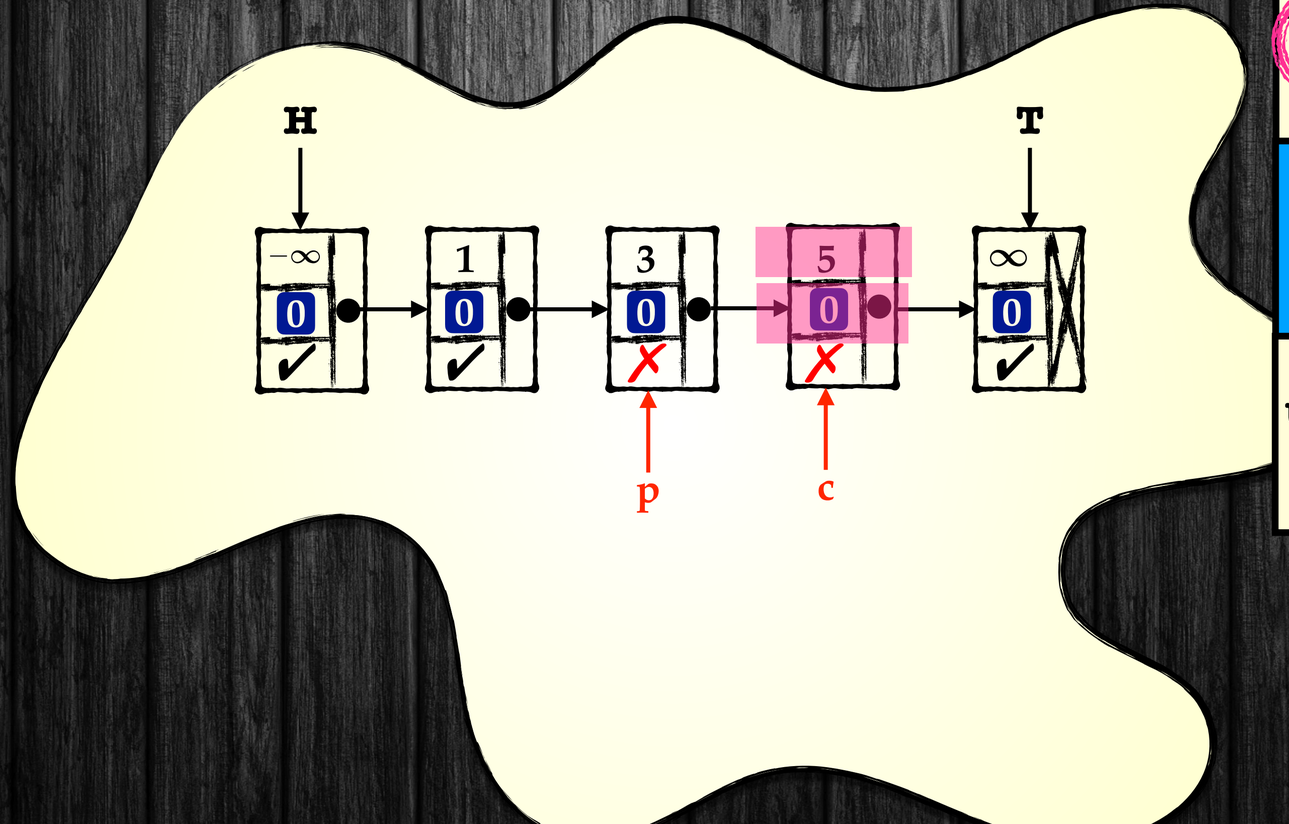


successful add	“when p.next is set”
unsuccessful add	“when unmarked entry is observed”
successful rmv	“when the cell is marked”
unsuccessful rmv	“first unmarked entry with larger key”
successful ctn	“when unmarked entry with input key found”
unsuccessful ctn	?

```
rmv(e):
local p, c, n, r
1  (p,c) := locate(e);
2  if (c.val = e)
3     c.mark := true
4     n := c.next;
5     p.next := n;
6     r := true;
7  else r := false;
8  unlock(p);
9  unlock(c);
10 return r;
```

rmv(4)

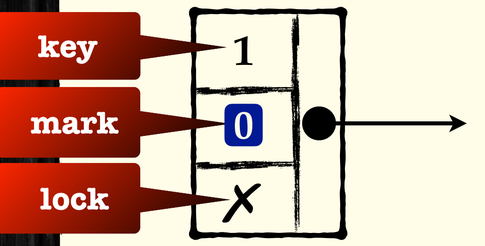
linearization point

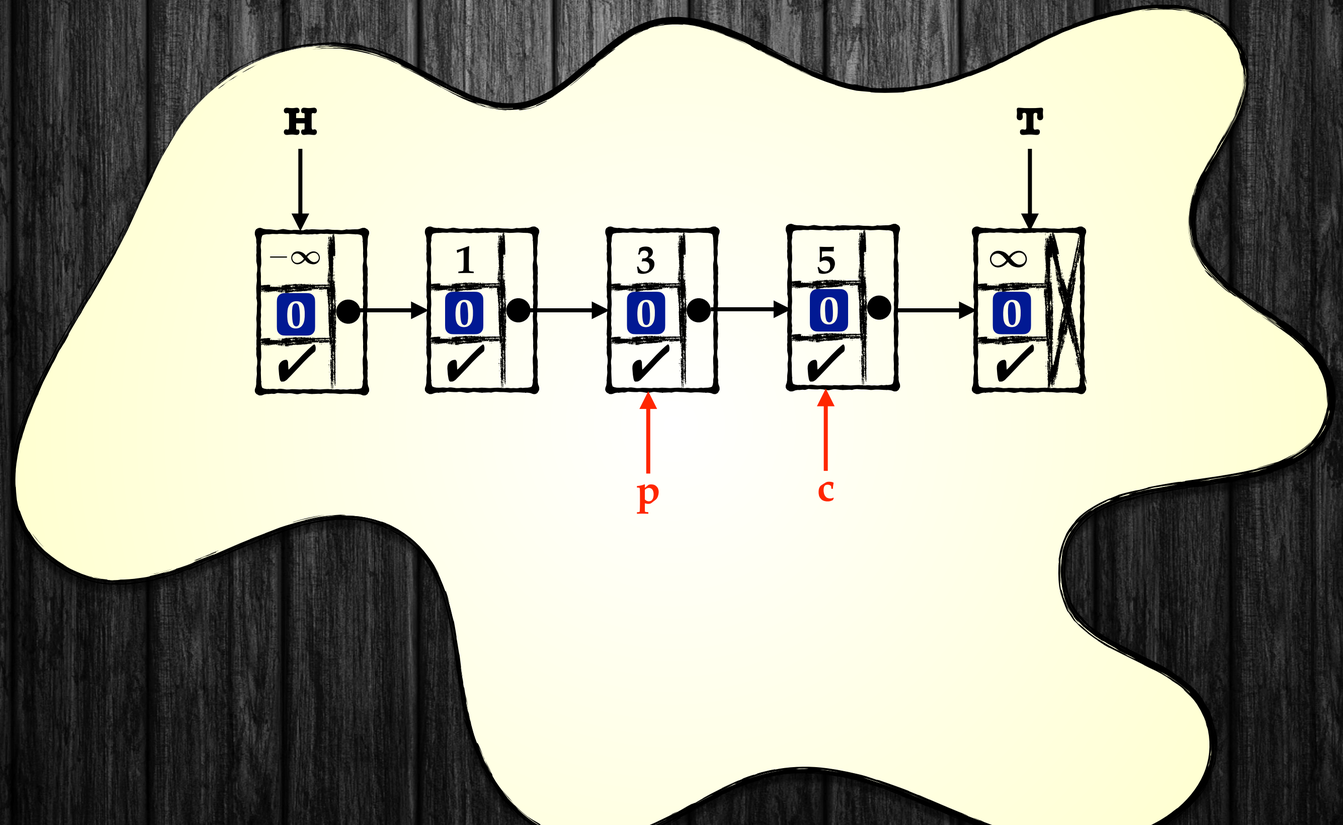


successful add	“when p.next is set”
unsuccessful add	“when unmarked entry is observed”
successful rmv	“when the cell is marked”
unsuccessful rmv	“first unmarked entry with larger key”
successful ctn	“when unmarked entry with input key found”
unsuccessful ctn	?

```
rmv(e):
local p, c, n, r
1  (p,c) := locate(e);
2  if (c.val = e)
3     c.mark := true
4     n := c.next;
5     p.next := n;
6     r := true;
7  else r := false;
8  unlock(p);
9  unlock(c);
10 return r;
```

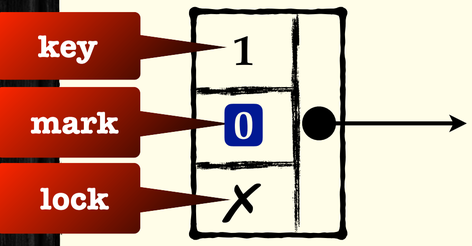
rmv(4)

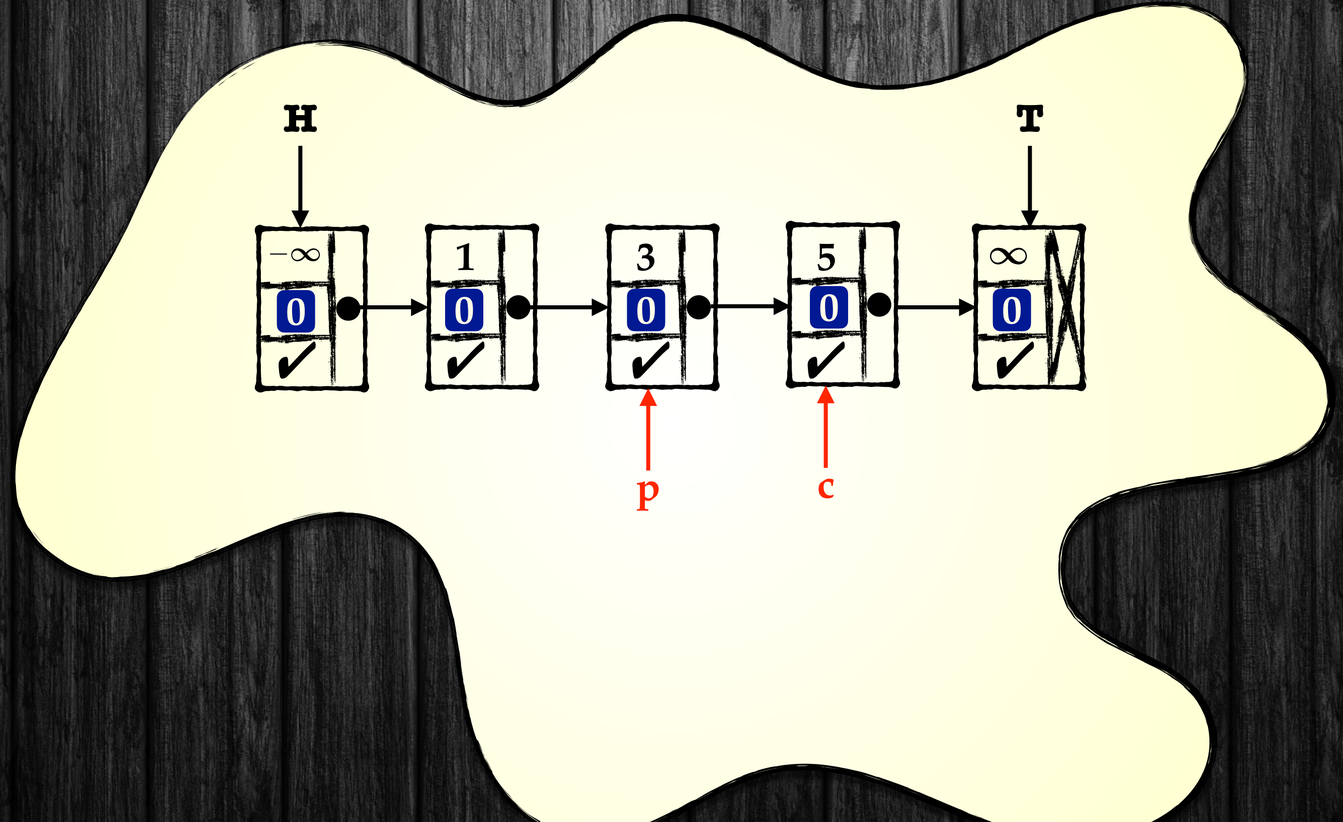




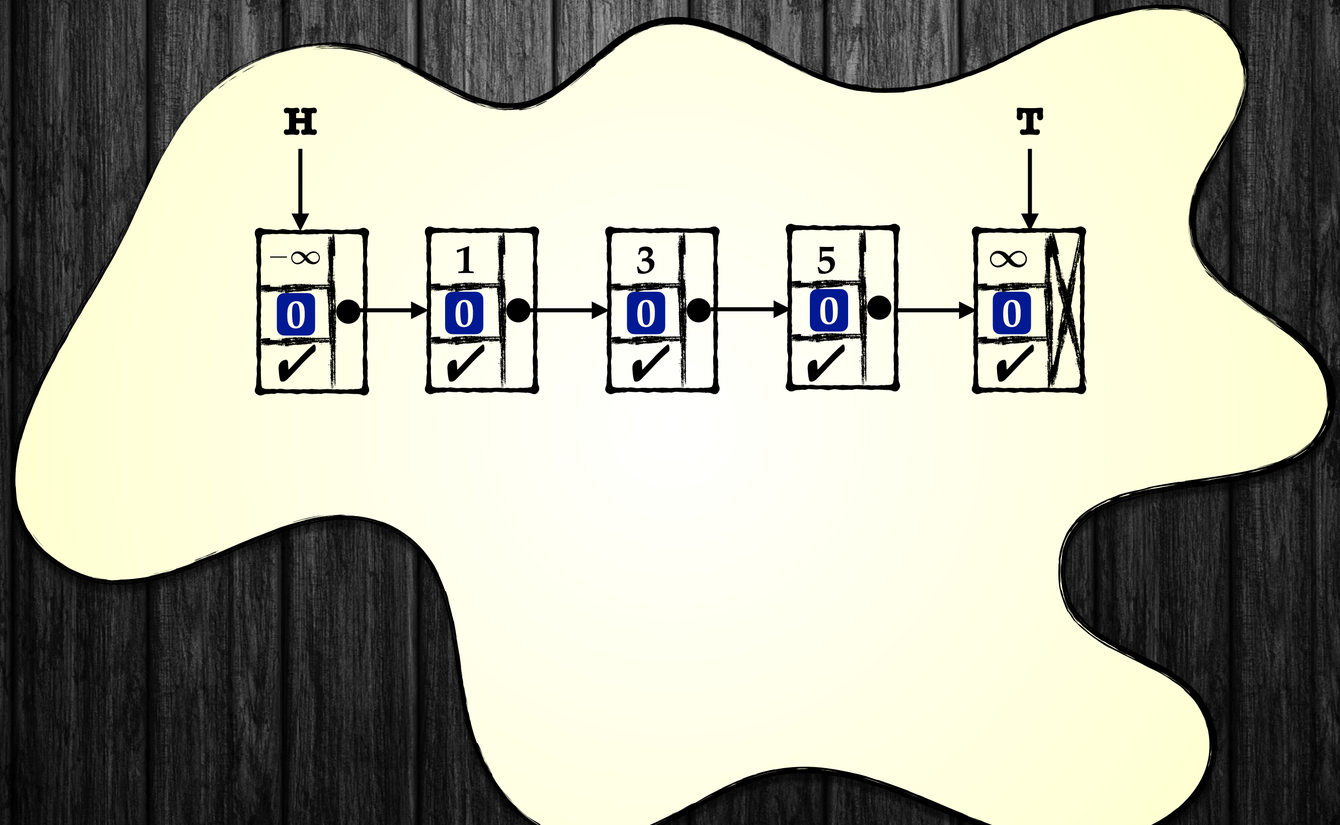
```
rmv(e):
local p, c, n, r
1  (p,c) := locate(e);
2  if (c.val = e)
3     c.mark := true
4     n := c.next;
5     p.next := n;
6     r := true;
7  else r := false;
8  unlock(p);
9  unlock(c);
10 return r;
```

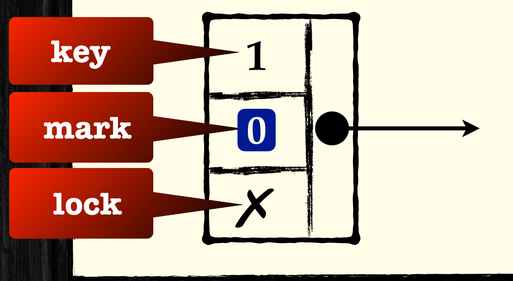
rmv(4)



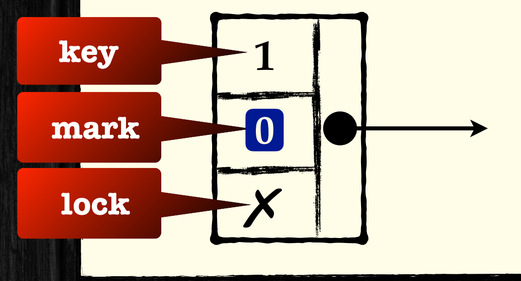


```
ctn(e):
local c, b
1  c := Head;
2  while (c.val < e)
3     c := c.next
4  b := c.mark
5  if (!b && c.val = e)
6     return true;
7  else return false;
```

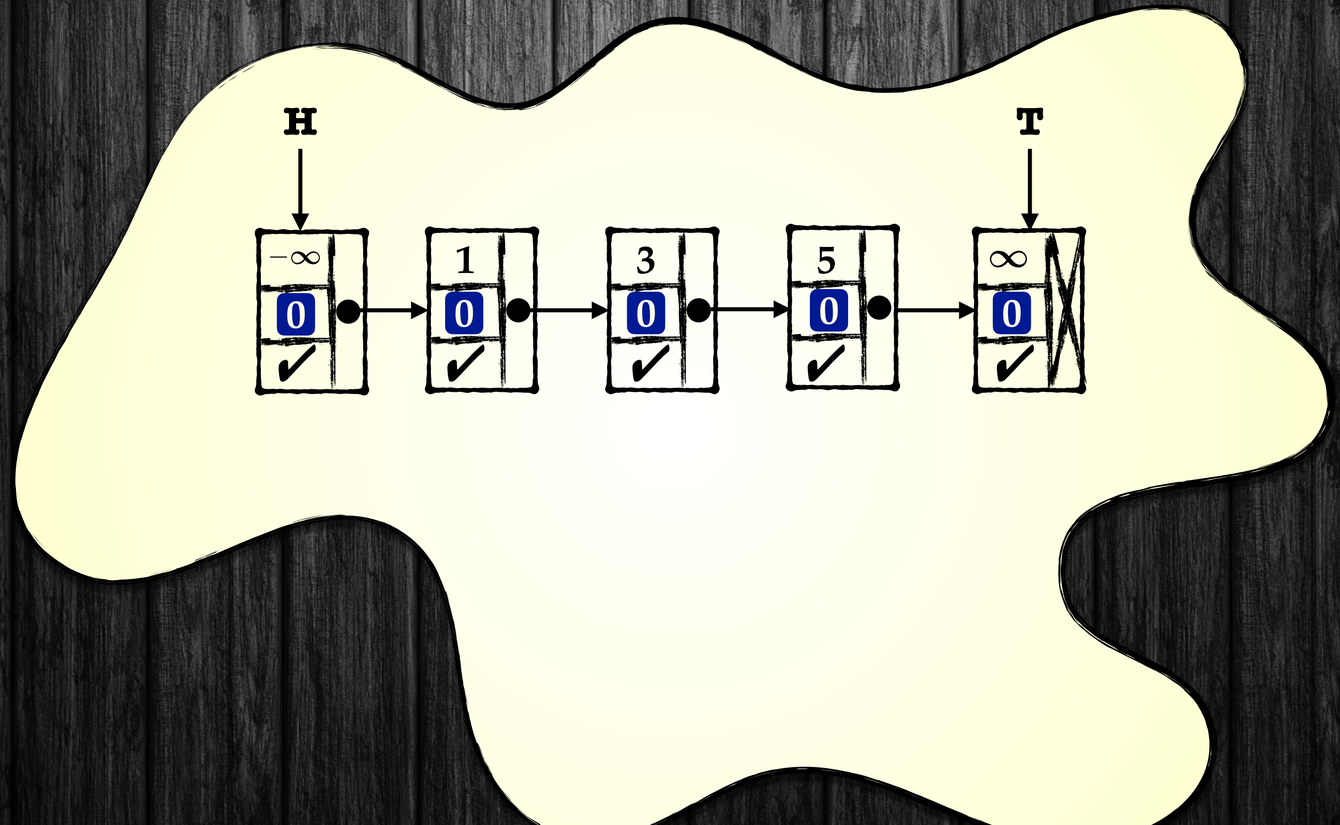




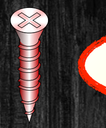
```
ctn(e):
local c, b
1  c := Head;
2  while (c.val < e)
3     c := c.next
4  b := c.mark
5  if (!b && c.val = e)
6     return true;
7  else return false;
```



no locks acquired

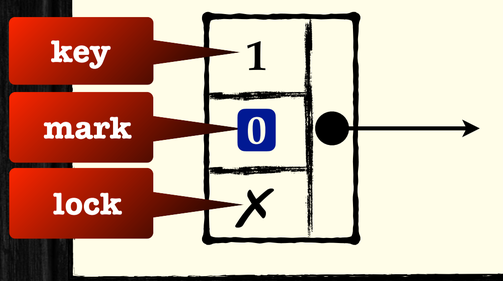


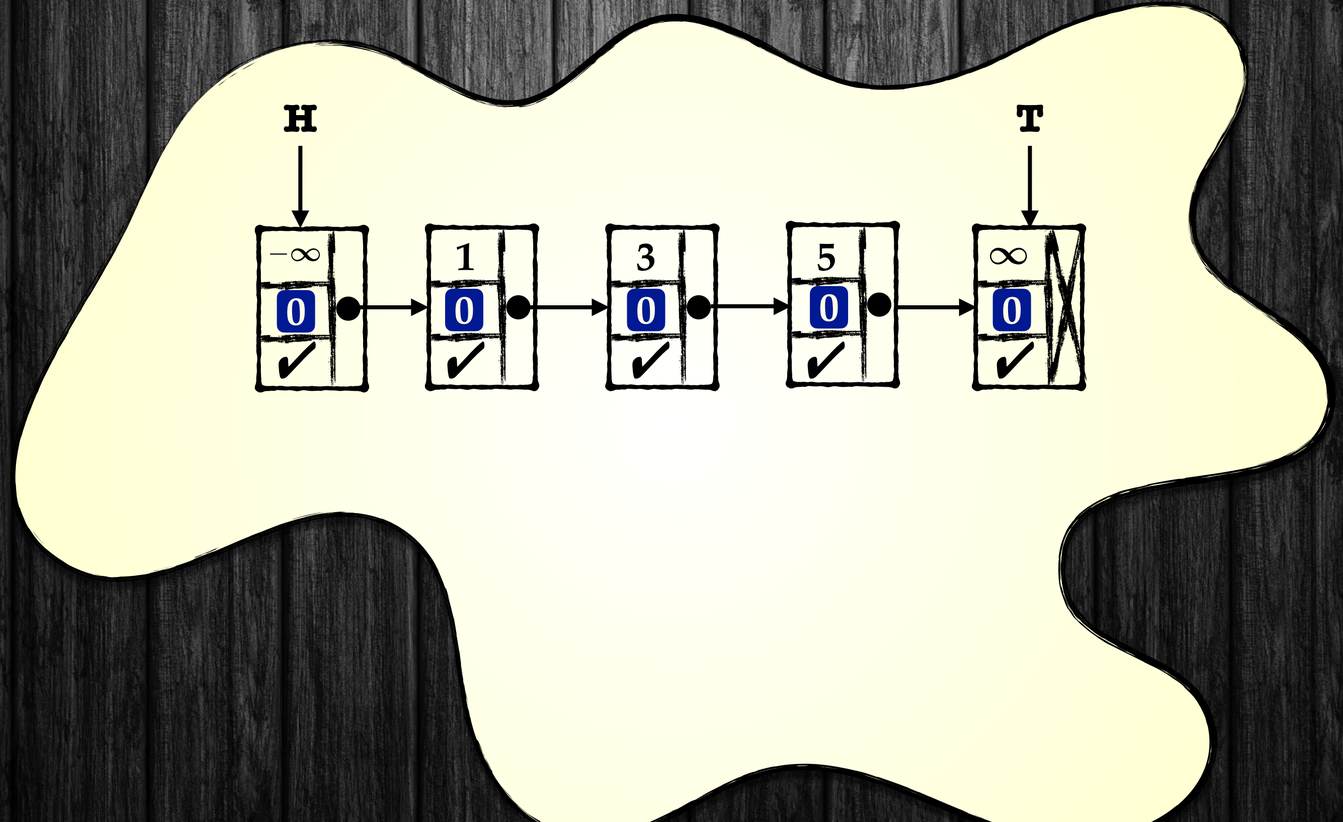
```
ctn(e):
local c, b
1  c := Head;
2  while (c.val < e)
3     c := c.next
4  b := c.mark
5  if (!b && c.val = e)
6     return true;
7  else return false;
```



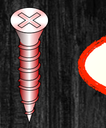
ctn(3)

no locks acquired



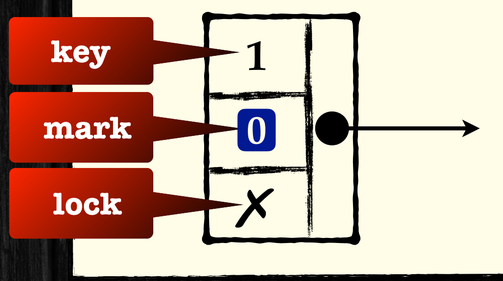


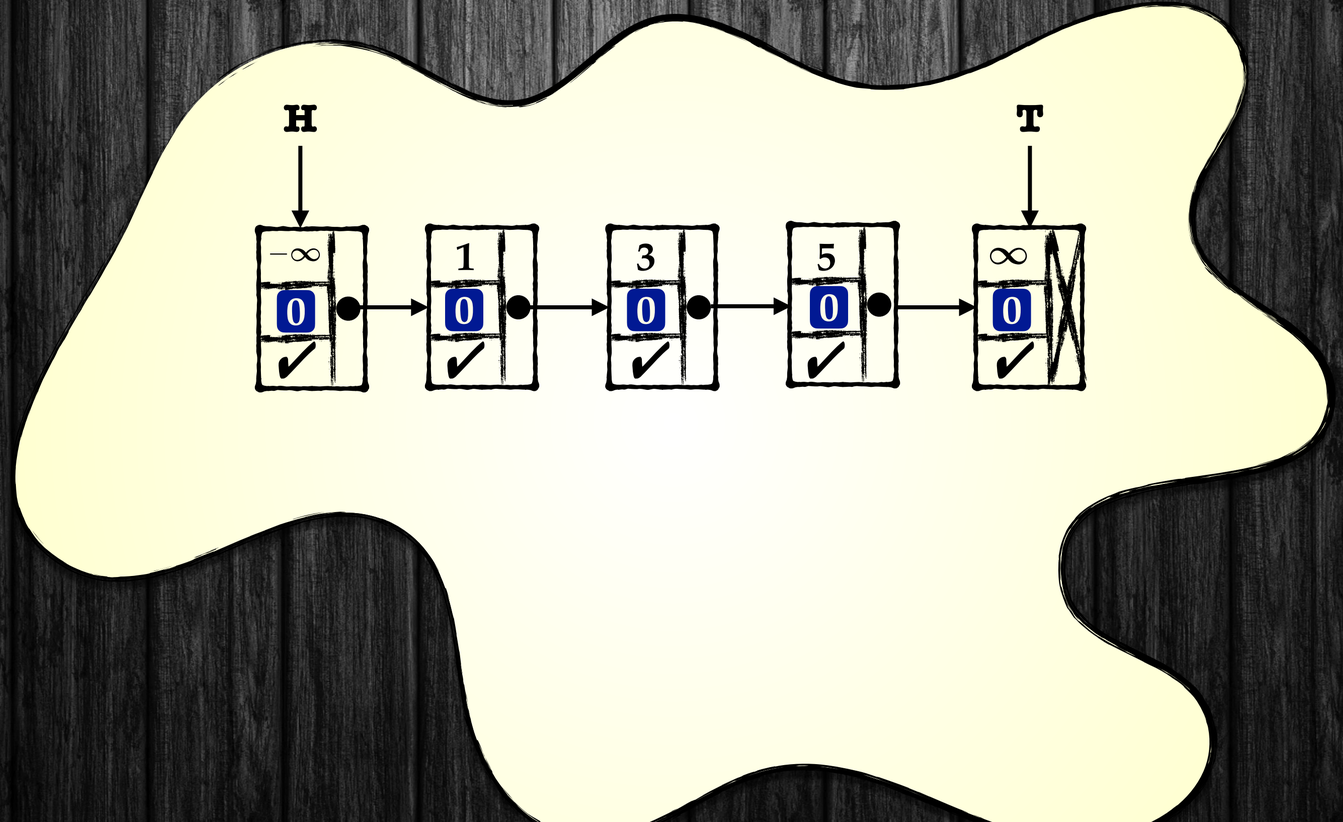
```
ctn(e):
local c, b
1  c := Head;
2  while (c.val < e)
3     c := c.next
4  b := c.mark
5  if (!b && c.val = e)
6     return true;
7  else return false;
```



ctn(3)

no locks acquired

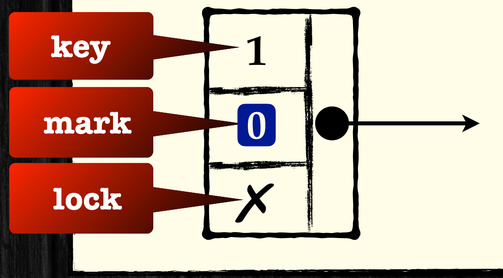


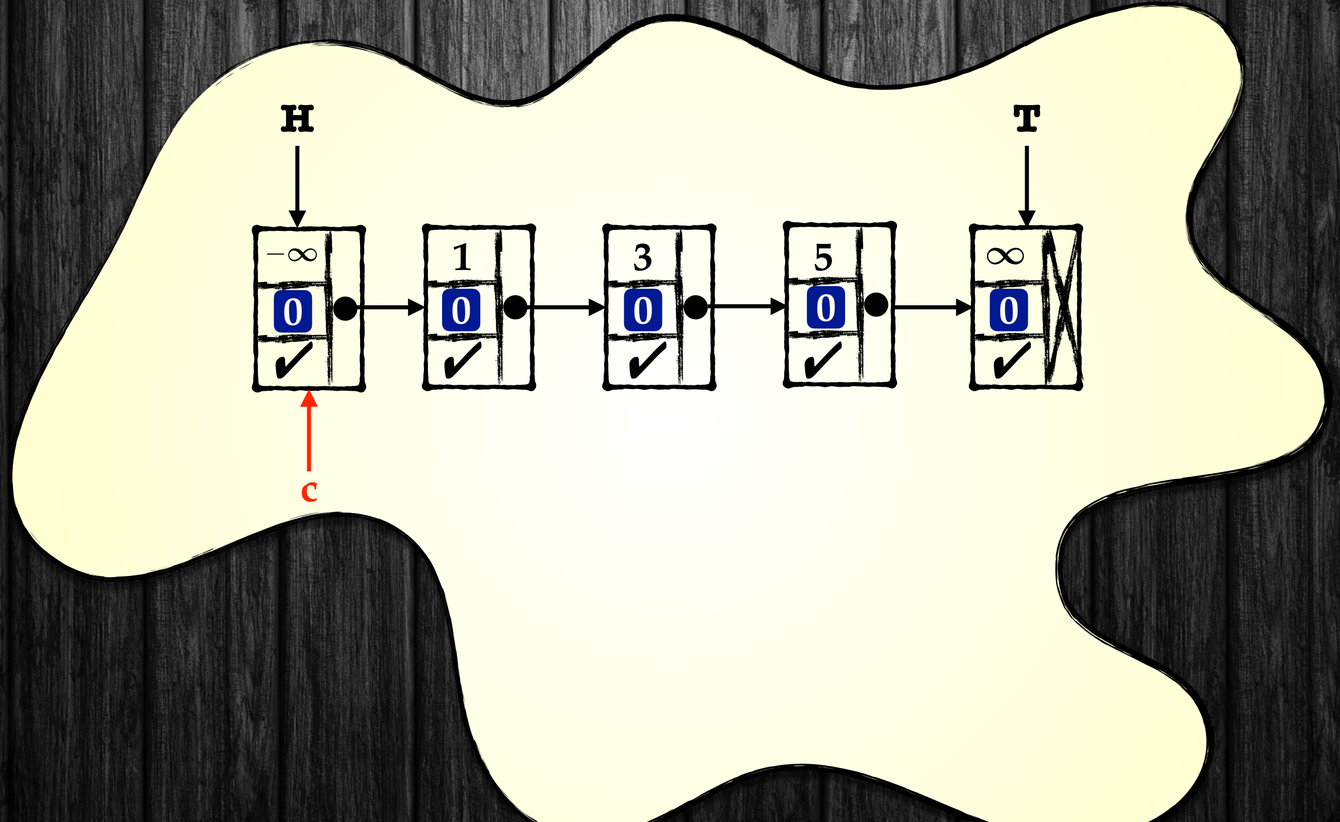


Sets | Lazy Synchronization

```
ctn(e):
local c, b
1  c := Head;
2  while (c.val < e)
3     c := c.next
4  b := c.mark
5  if (!b && c.val = e)
6     return true;
7  else return false;
```

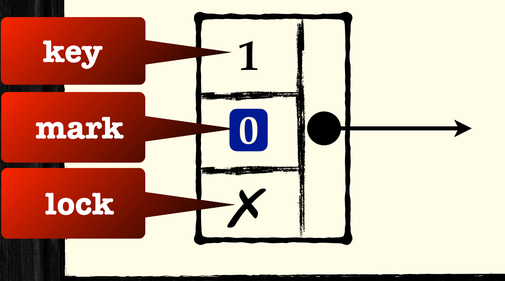


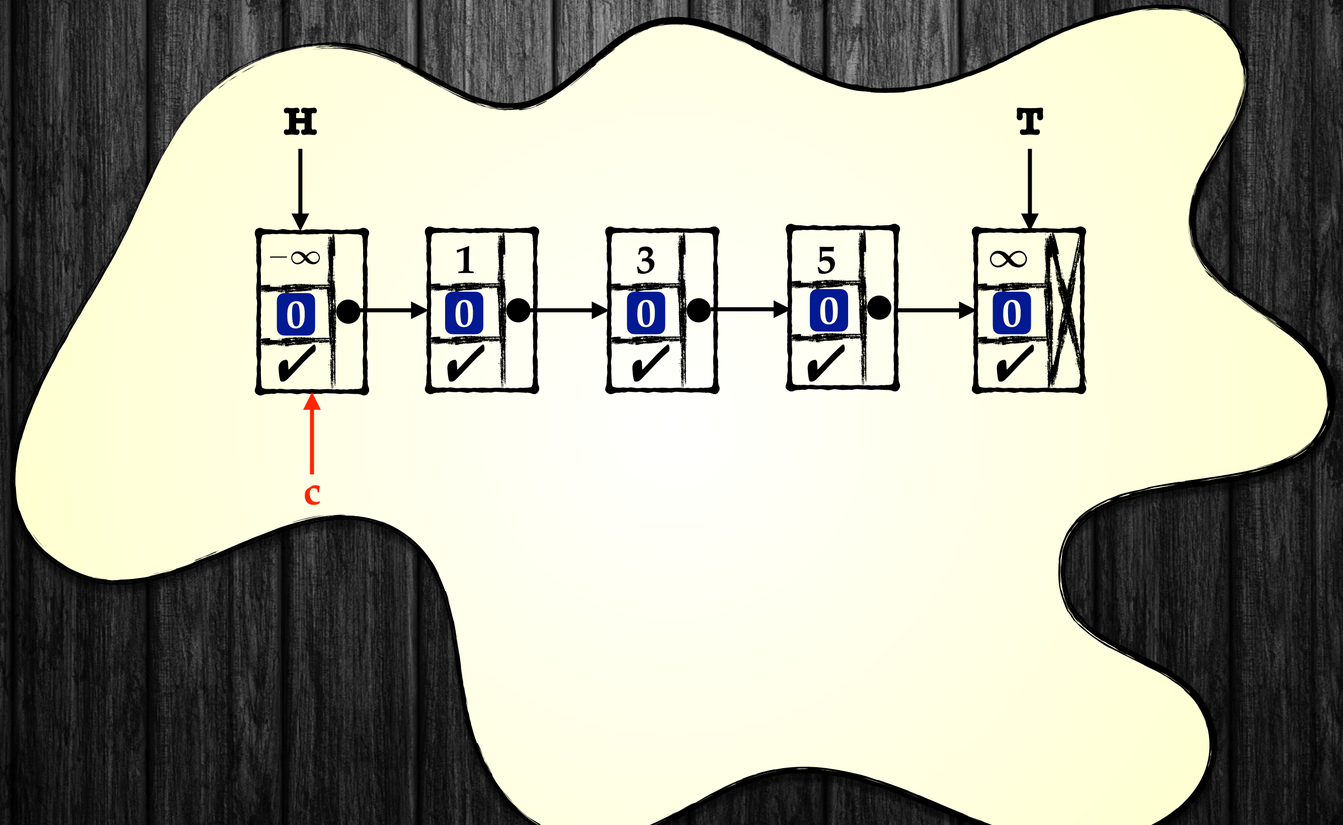




```
ctn(e):
local c, b
1  c := Head;
2  while (c.val < e)
3     c := c.next
4  b := c.mark
5  if (!b && c.val = e)
6     return true;
7  else return false;
```

ctn(3)

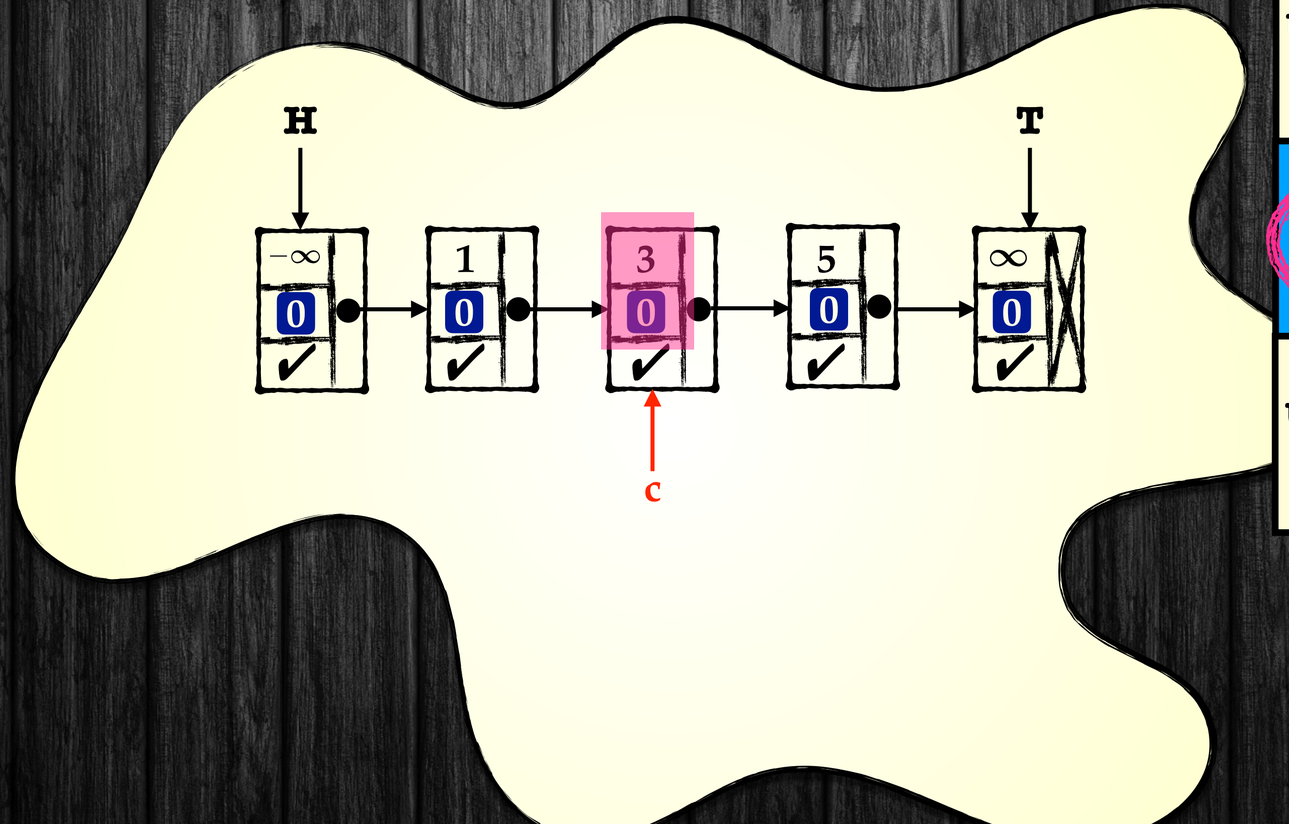




```
ctn(e):
local c, b
1  c := Head;
2  while (c.val < e)
3     c := c.next
4  b := c.mark
5  if (!b && c.val = e)
6     return true;
7  else return false;
```

ctn(3)

linearization point

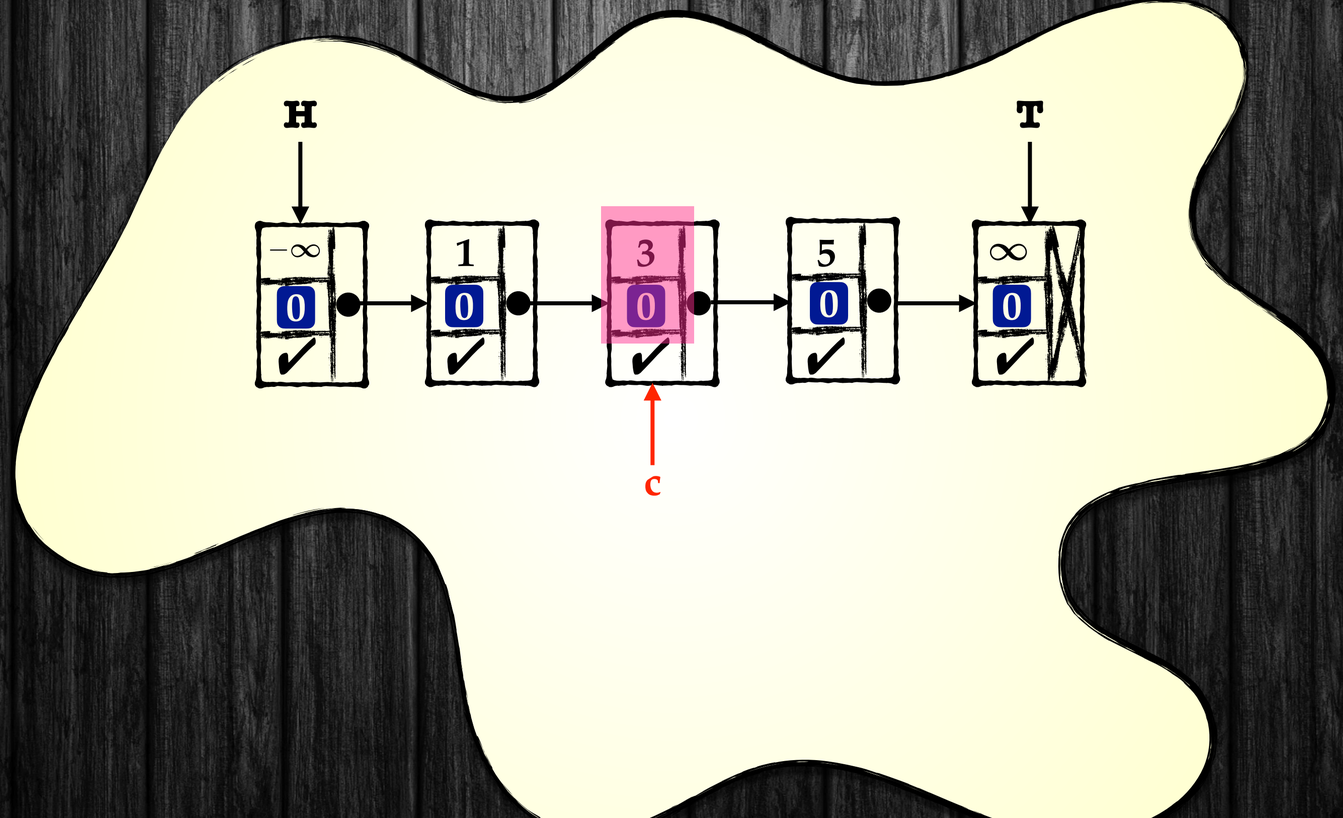


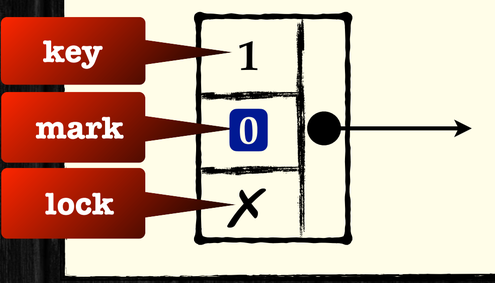
successful add	“when p.next is set”
unsuccessful add	“when unmarked entry is observed”
successful rmv	“when the cell is marked”
unsuccessful rmv	“first unmarked entry with larger key”
successful ctn	“when unmarked entry with input key found”
unsuccessful ctn	?

Sets | Lazy Synchronization

```
ctn(e):
local c, b
1  c := Head;
2  while (c.val < e)
3     c := c.next
4  b := c.mark
5  if (!b && c.val = e)
6     return true;
7  else return false;
```

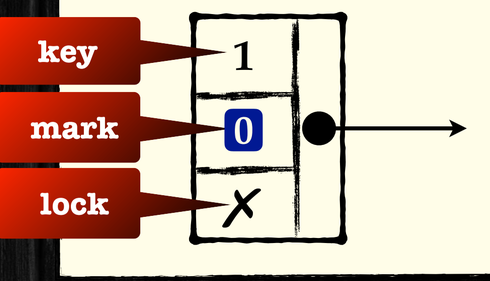
ctn(3)

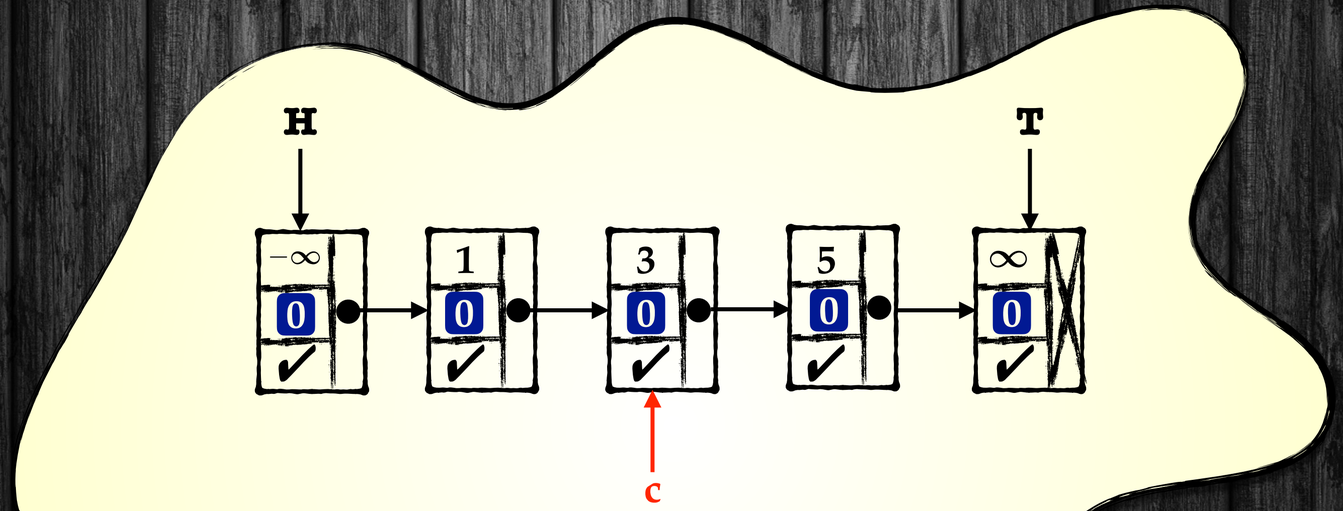




```
ctn(e):
local c, b
1  c := Head;
2  while (c.val < e)
3     c := c.next
4  b := c.mark
5  if (!b && c.val = e)
6     return true;
7  else return false;
```

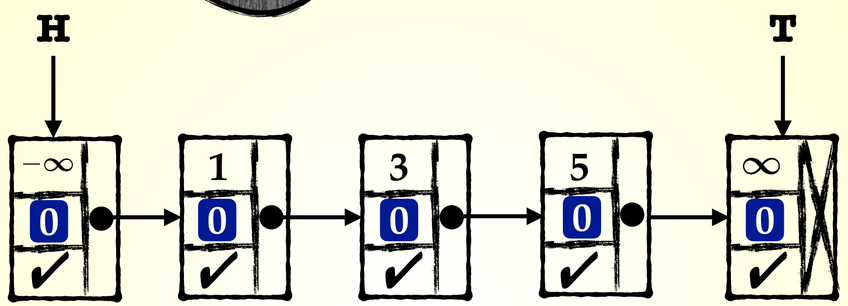
ctn(3)





```
ctn(e):
local c, b
1  c := Head;
2  while (c.val < e)
3     c := c.next
4  b := c.mark
5  if (!b && c.val = e)
6     return true;
7  else return false;
```

unsuccessful contains linearization point



wrong linearization policy

successful add	“when p.next is set”
unsuccessful add	“when unmarked entry is observed”
successful rmv	“when the cell is marked”
unsuccessful rmv	“first unmarked entry with larger key”
successful ctn	“when unmarked entry with input key found”
unsuccessful ctn	?

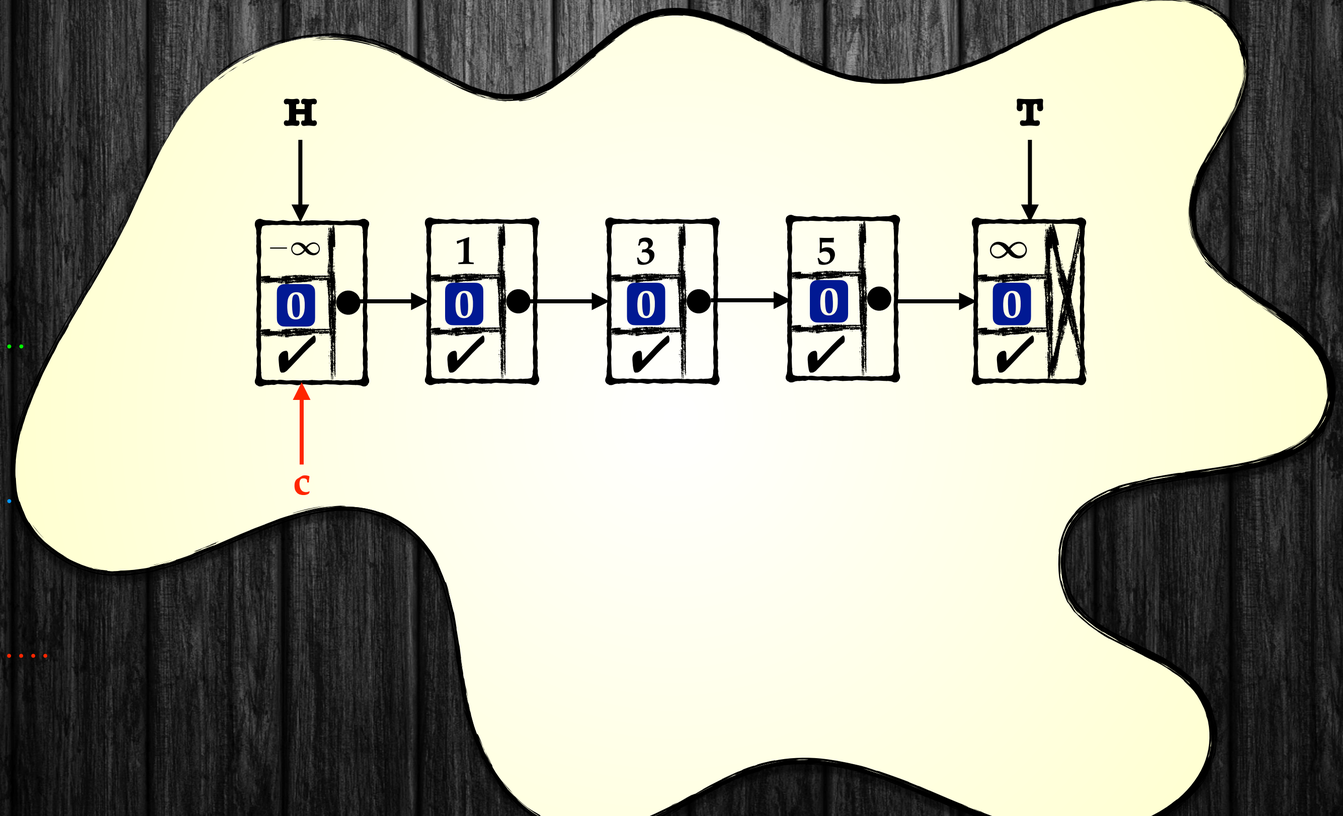
“when finding the cell with the input key marked”

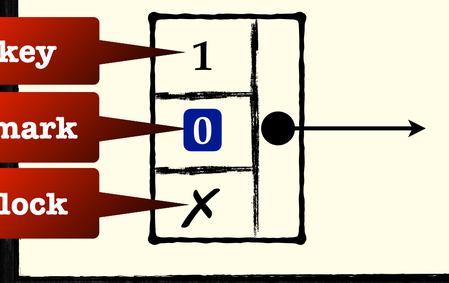
“complicated linearization policy”

Sets | Lazy Synchronization

```
ctn(e):
local c, b
1  c := Head;
2  while (c.val < e)
3     c := c.next
4  b := c.mark
5  if (!b && c.val = e)
6     return true;
7  else return false;
```

ctn(3)



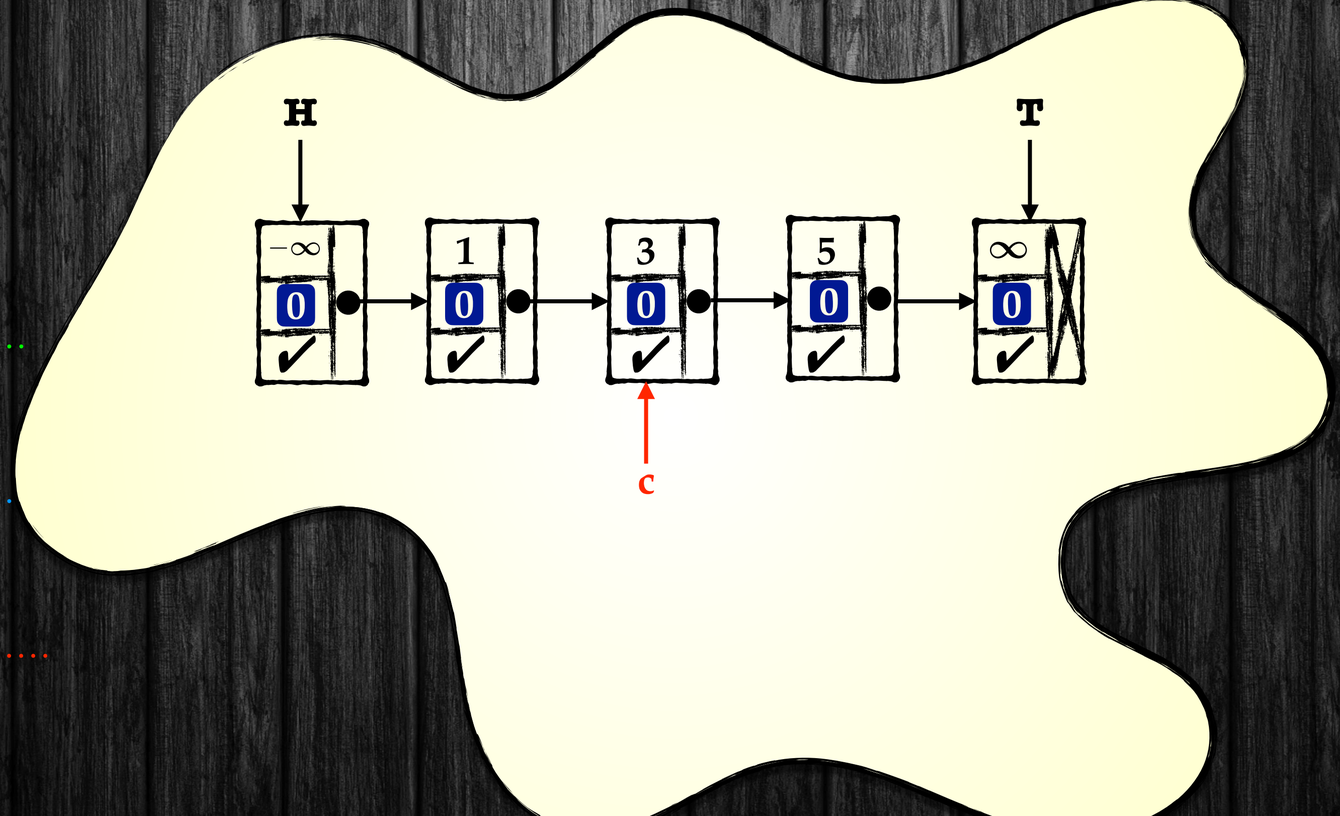


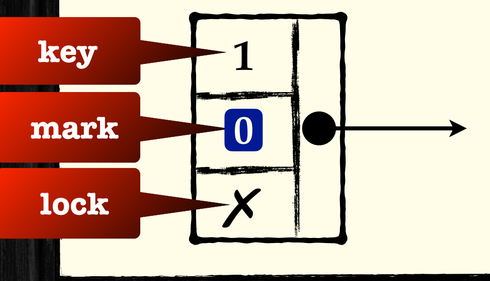
ctn(3)

```
ctn(e):
local c, b
1  c := Head;
2  while (c.val < e)
3     c := c.next
4  b := c.mark
5  if (!b && c.val = e)
6     return true;
7  else return false;
```

ctn(3)

not executed yet





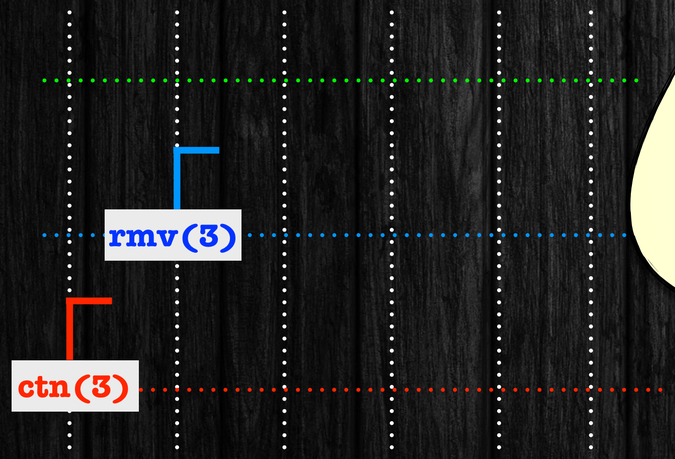
ctn(3)

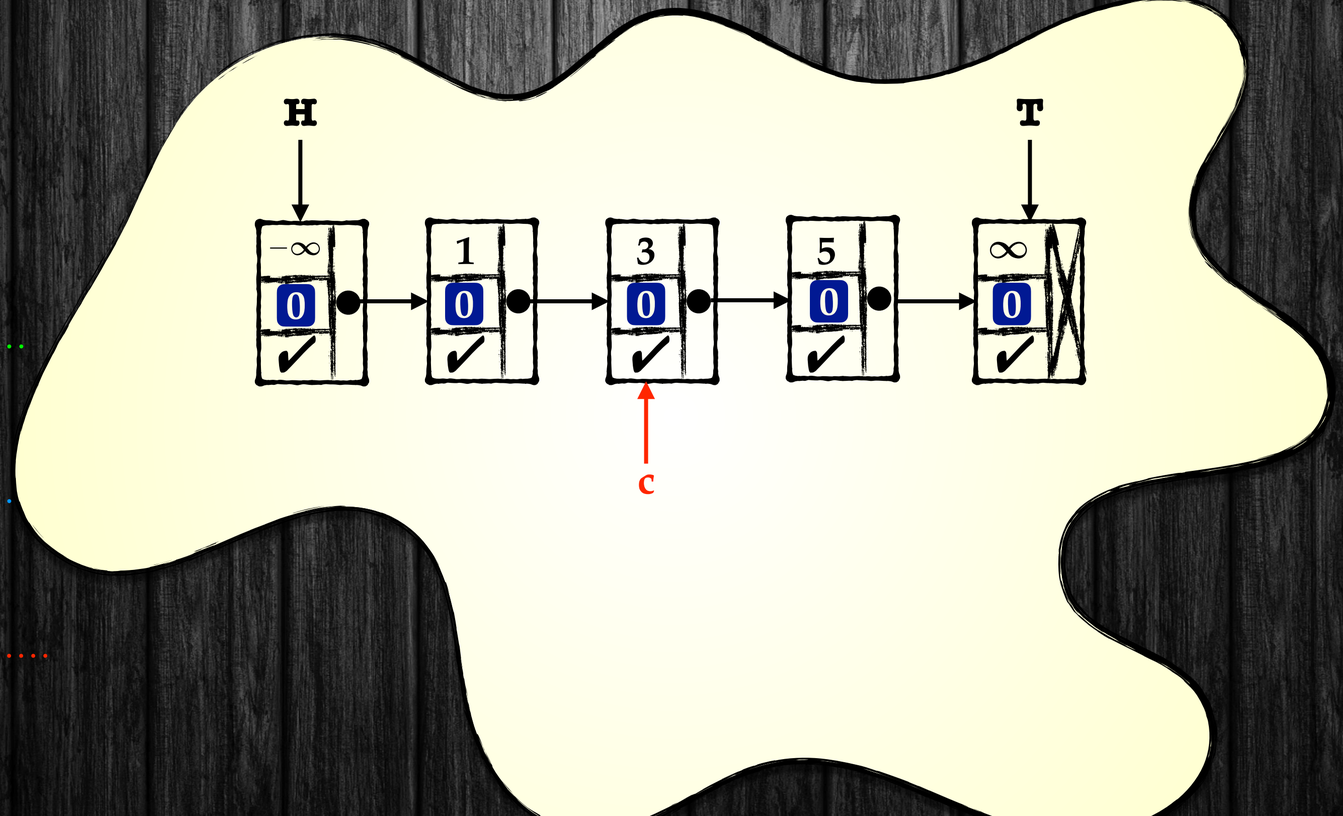
Sets | Lazy Synchronization

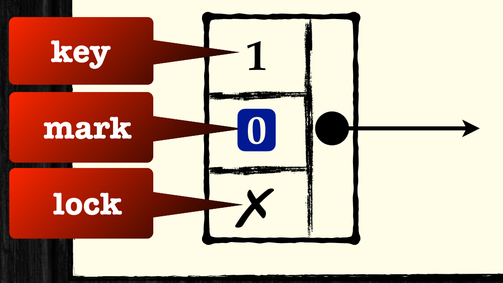
```
ctn(e):
local c, b
1  c := Head;
2  while (c.val < e)
3     c := c.next
4  b := c.mark
5  if (!b && c.val = e)
6     return true;
7  else return false;
```

ctn(3)

not executed yet







rmv(3)

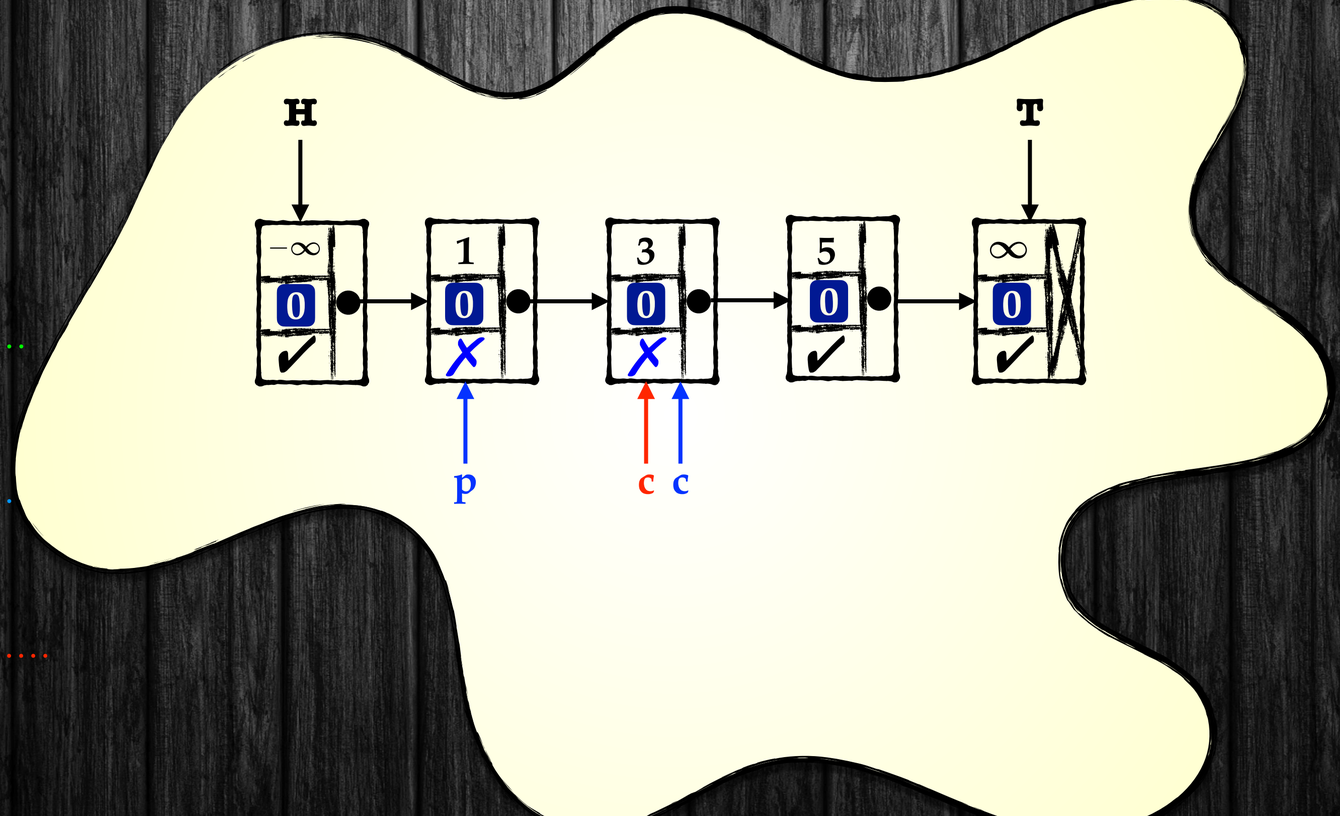
```
rmv(e):
local p, c, n, r
1  (p,c) := locate(e);
2  if (c.val = e)
3     c.mark := true
4     n := c.next;
5     p.next := n;
6     r := true;
7  else r := false;
8  unlock(p);
9  unlock(c);
10 return r;
```

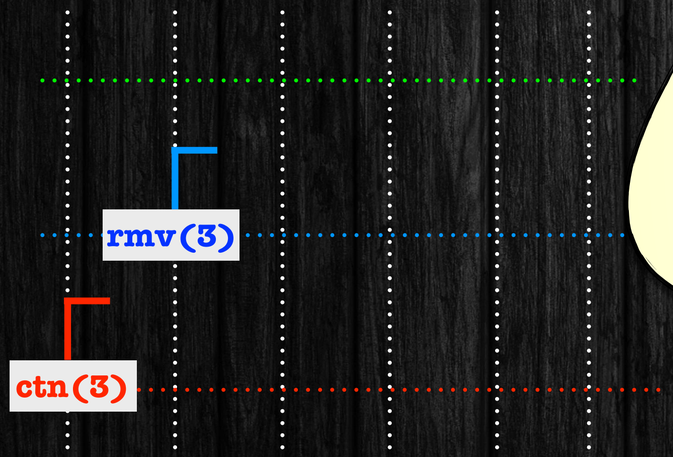
Sets | Lazy Synchronization

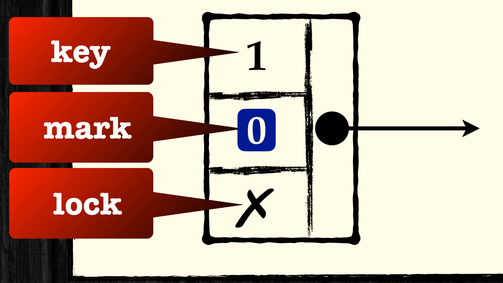
```
ctn(e):
local c, b
1  c := Head;
2  while (c.val < e)
3     c := c.next
4  b := c.mark
5  if (!b && c.val = e)
6     return true;
7  else return false;
```

ctn(3)

not executed yet







rmv(3)

```
rmv(e):
local p, c, n, r
1  (p,c) := locate(e);
2  if (c.val = e)
3     c.mark := true
4     n := c.next;
5     p.next := n;
6     r := true;
7  else r := false;
8  unlock(p);
9  unlock(c);
10 return r;
```

Sets | **Lazy Synchronization**

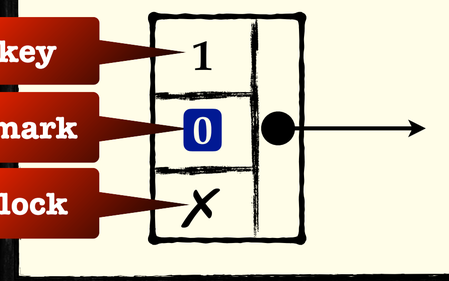
```
ctn(e):
local c, b
1  c := Head;
2  while (c.val < e)
3     c := c.next
4  b := c.mark
5  if (!b && c.val = e)
6     return true;
7  else return false;
```

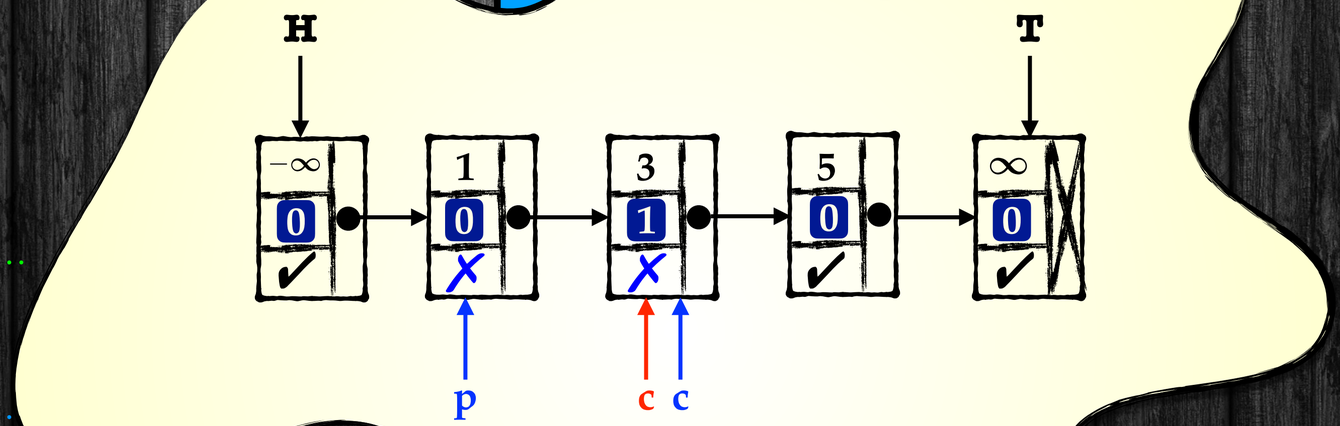
ctn(3)

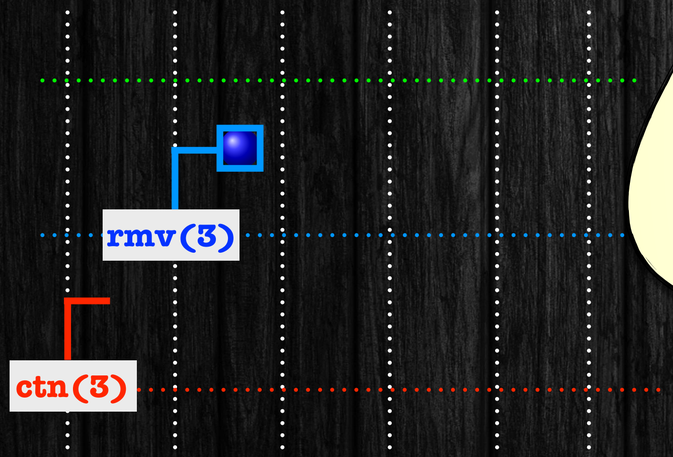
not executed yet

linearization point

unsuccessful add	"when unmarked entry is observed"
successful rmv	"when the cell is marked"
unsuccessful rmv	"first unmarked entry with larger key"
suc…	"when unm… ent…







rmv(3)

```
rmv(e):
local p, c, n, r
1  (p,c) := locate(e);
2  if (c.val = e)
3     c.mark := true
4     n := c.next;
5     p.next := n;
6     r := true;
7  else r := false;
8  unlock(p);
9  unlock(c);
10 return r;
```

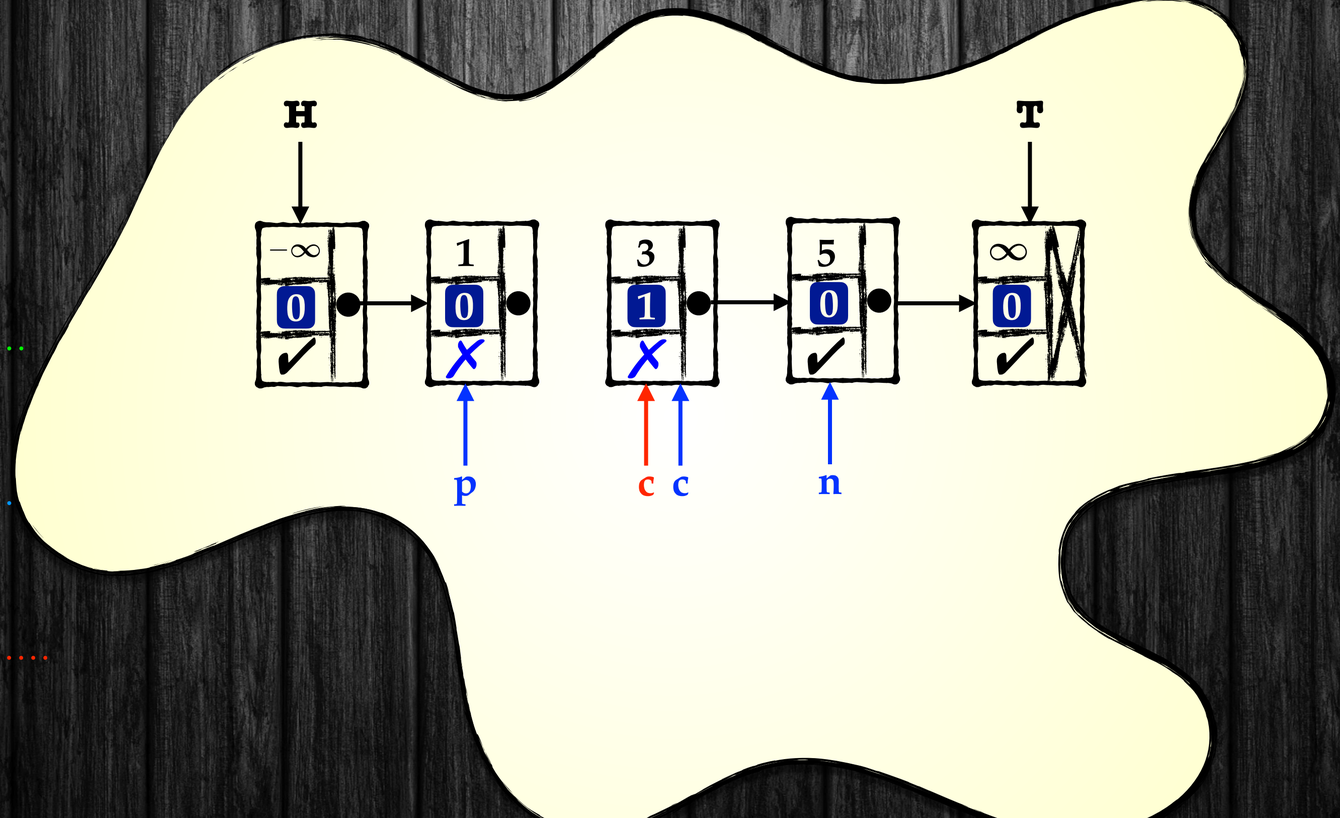
Sets | Lazy Synchronization

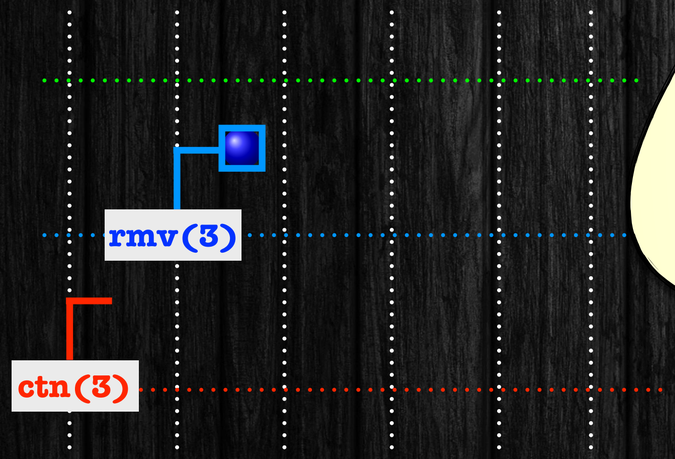
```
ctn(e):
local c, b
1  c := Head;
2  while (c.val < e)
3     c := c.next
4  b := c.mark
5  if (!b && c.val = e)
6     return true;
7  else return false;
```

ctn(3)

not executed yet







rmv(3)

```
rmv(e):
local p, c, n, r
1  (p,c) := locate(e);
2  if (c.val = e)
3     c.mark := true
4     n := c.next;
5     p.next := n;
6     r := true;
7  else r := false;
8  unlock(p);
9  unlock(c);
10 return r;
```

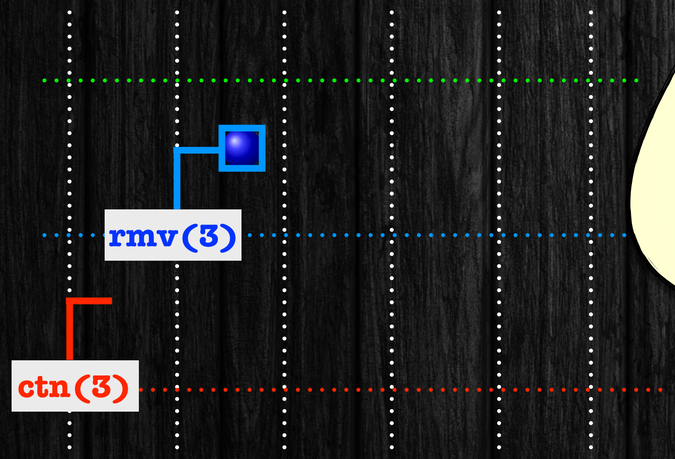
Sets | Lazy Synchronization

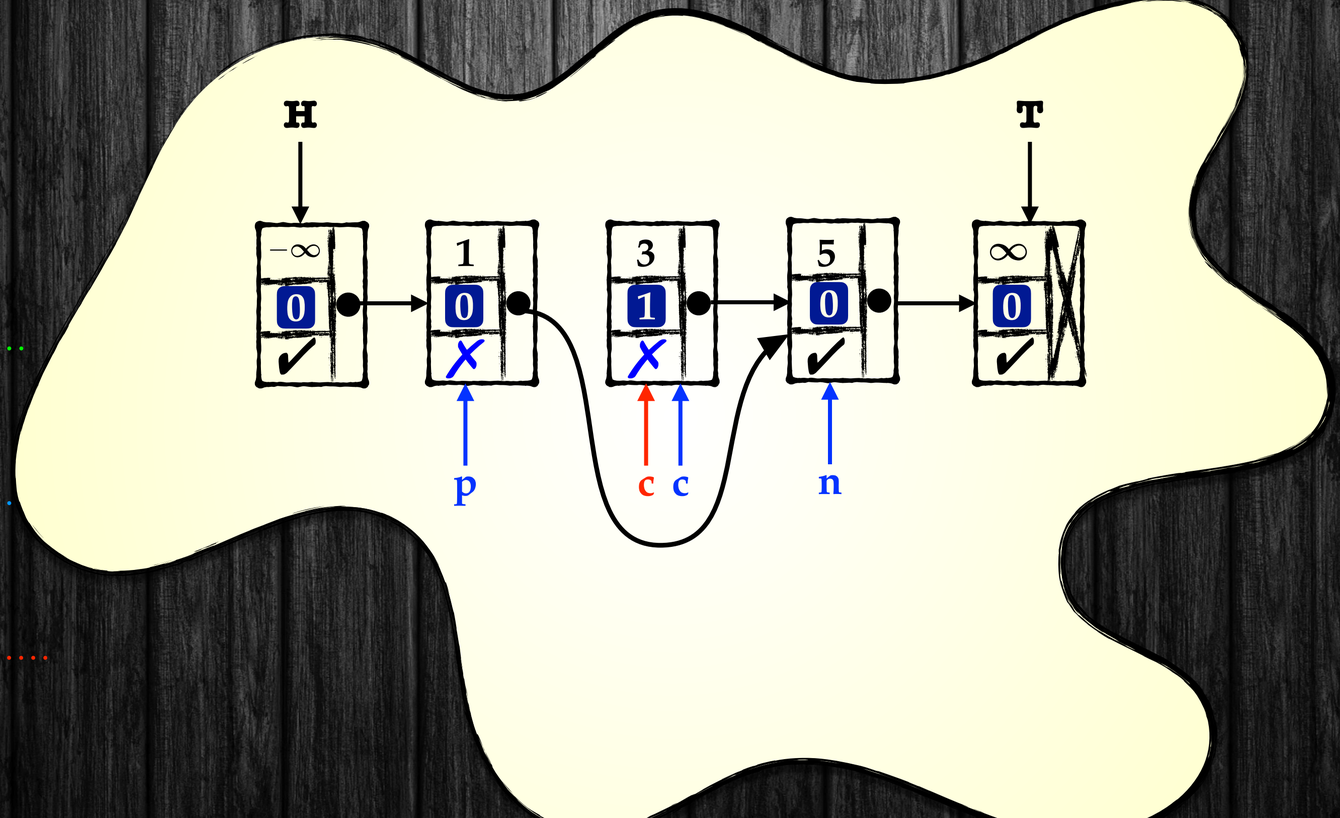
```
ctn(e):
local c, b
1  c := Head;
2  while (c.val < e)
3     c := c.next
4  b := c.mark
5  if (!b && c.val = e)
6     return true;
7  else return false;
```

ctn(3)

not executed yet







rmv(3)

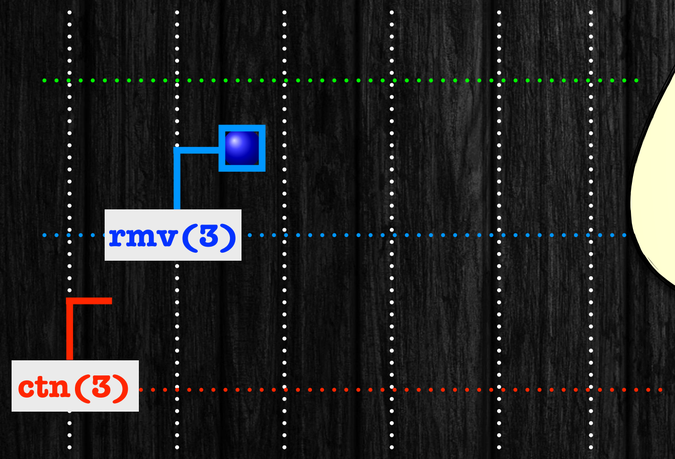
```
rmv(e):
local p, c, n, r
1  (p,c) := locate(e);
2  if (c.val = e)
3     c.mark := true
4     n := c.next;
5     p.next := n;
6     r := true;
7  else r := false;
8  unlock(p);
9  unlock(c);
10 return r;
```

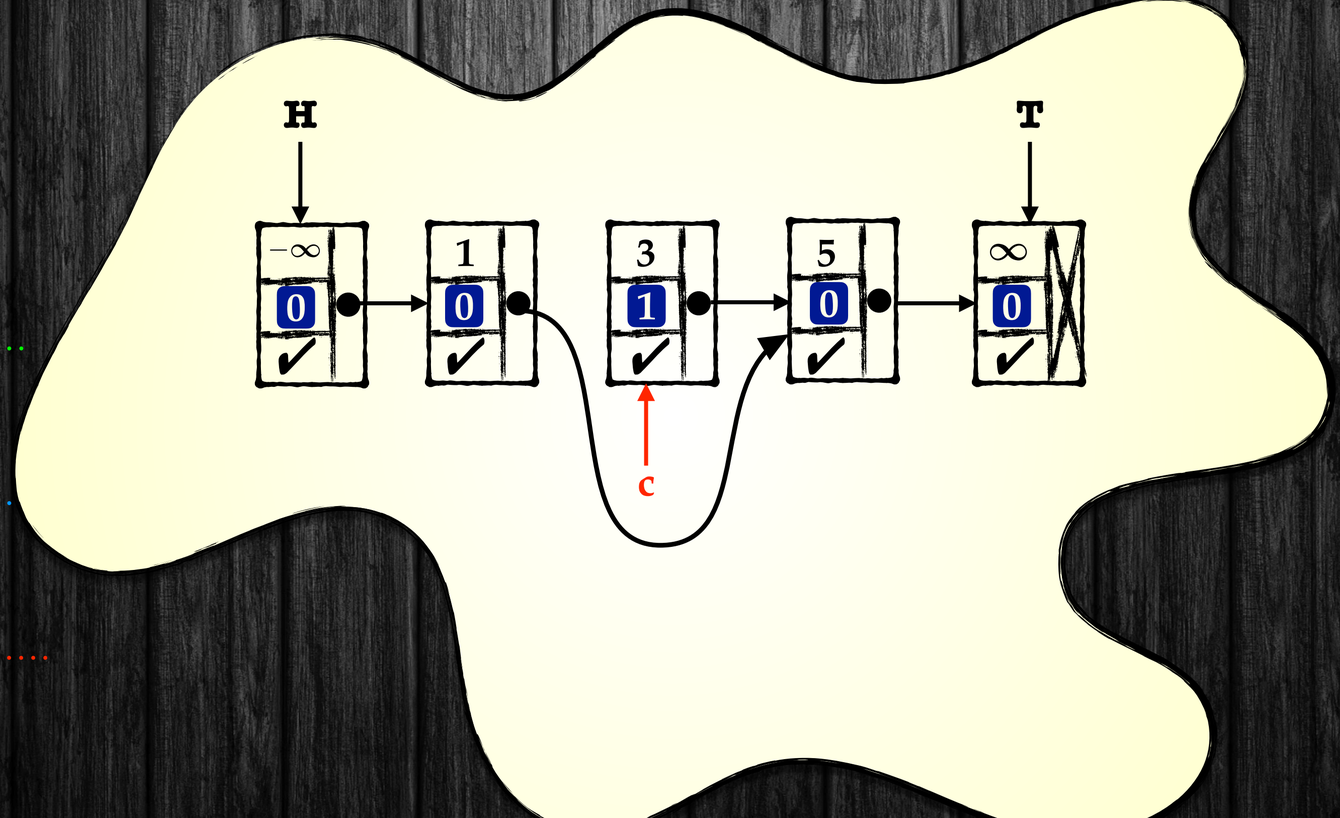
Sets | Lazy Synchronization

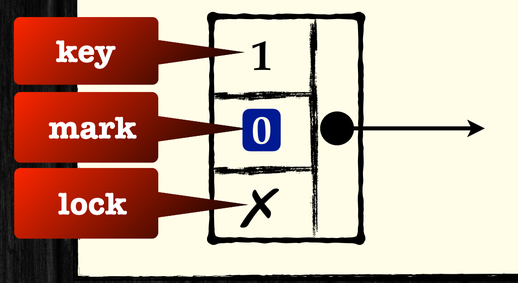
```
ctn(e):
local c, b
1  c := Head;
2  while (c.val < e)
3     c := c.next
4  b := c.mark
5  if (!b && c.val = e)
6     return true;
7  else return false;
```

ctn(3)

not executed yet







rmv(3)

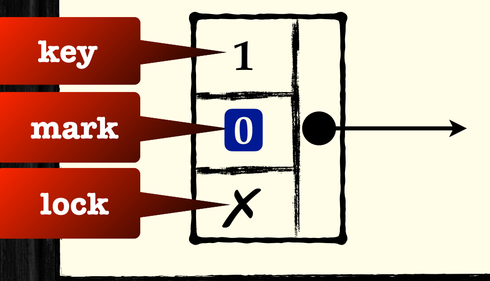
```
rmv(e):
local p, c, n, r
1  (p,c) := locate(e);
2  if (c.val = e)
3     c.mark := true
4     n := c.next;
5     p.next := n;
6     r := true;
7  else r := false;
8  unlock(p);
9  unlock(c);
10 return r;
```

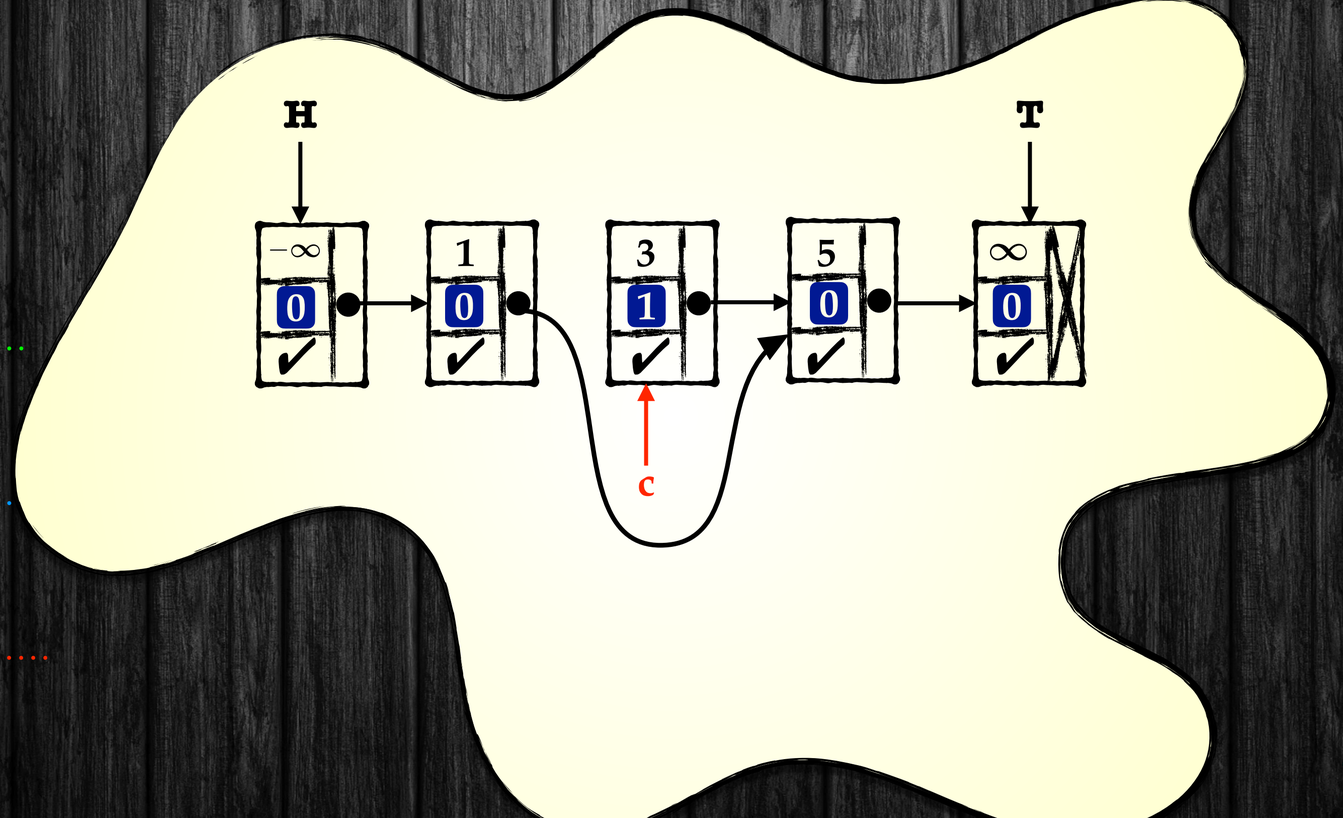
Sets | Lazy Synchronization

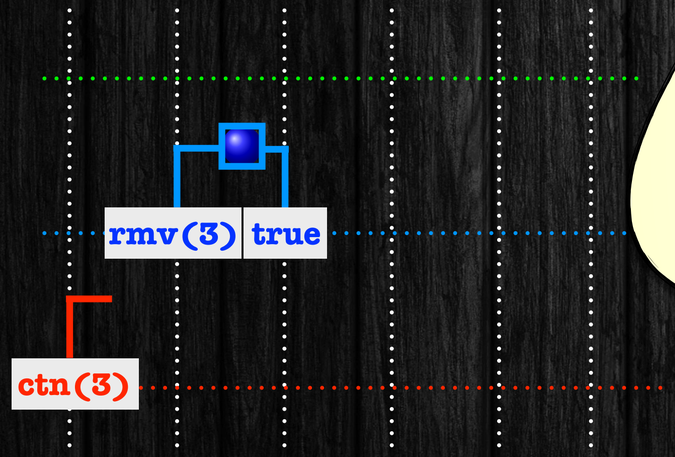
```
ctn(e):
local c, b
1  c := Head;
2  while (c.val < e)
3     c := c.next
4  b := c.mark
5  if (!b && c.val = e)
6     return true;
7  else return false;
```

ctn(3)

not executed yet







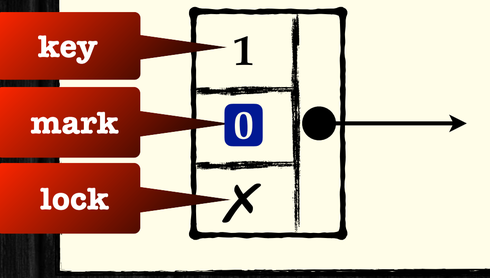

```
rmv(e):                                  rmv(3)
local p, c, n, r
1  (p,c) := locate(e);
2  if (c.val = e)
3     c.mark := true
4     n := c.next;
5     p.next := n;
6     r := true;
7  else r := false;
8  unlock(p);
9  unlock(c);
10 return r;
```

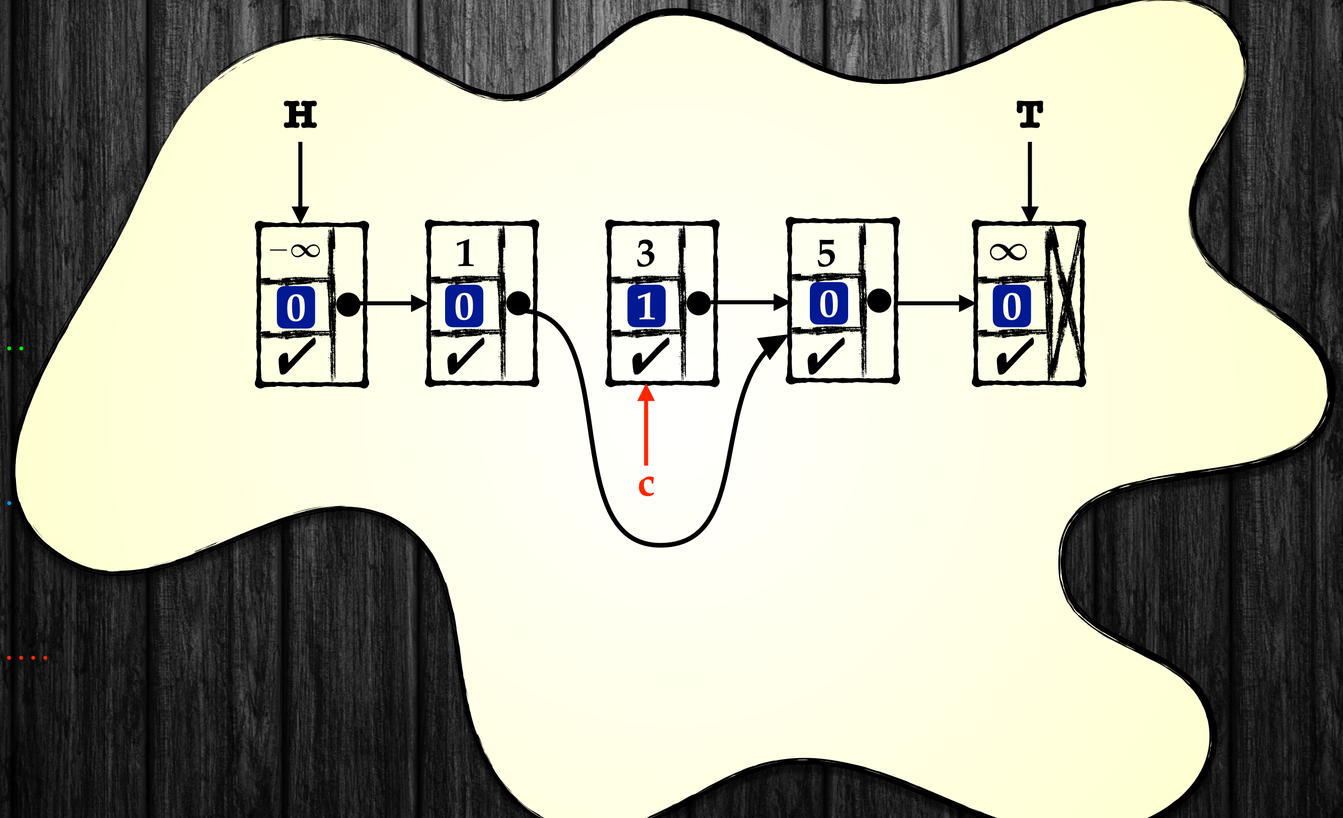
Sets | Lazy Synchronization

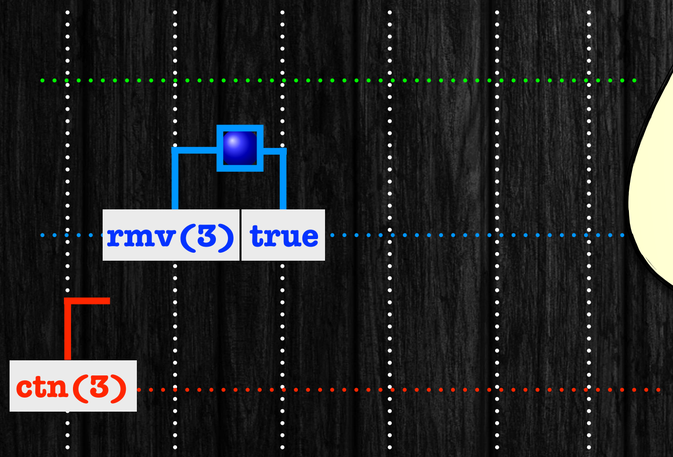
```
ctn(e):
local c, b
1  c := Head;
2  while (c.val < e)
3     c := c.next
4  b := c.mark
5  if (!b && c.val = e)
6     return true;
7  else return false;
```

ctn(3)

not executed yet



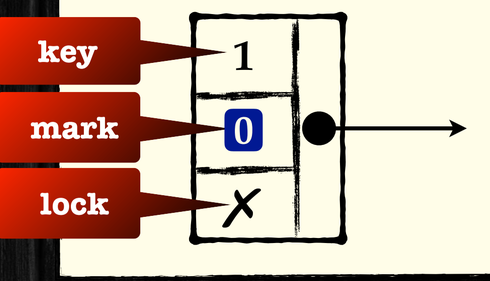


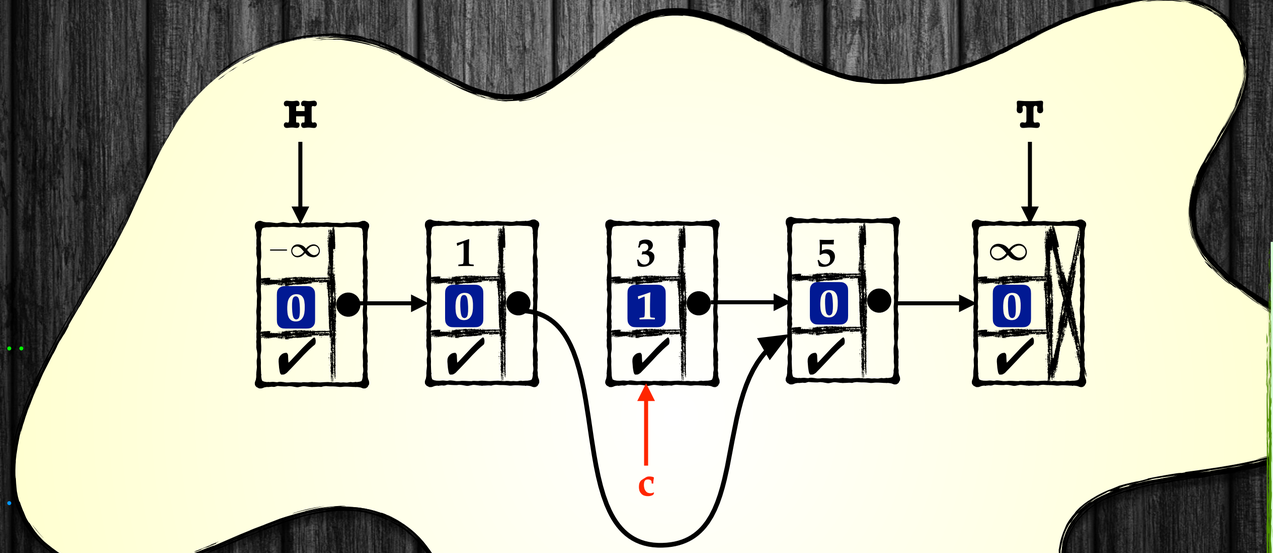


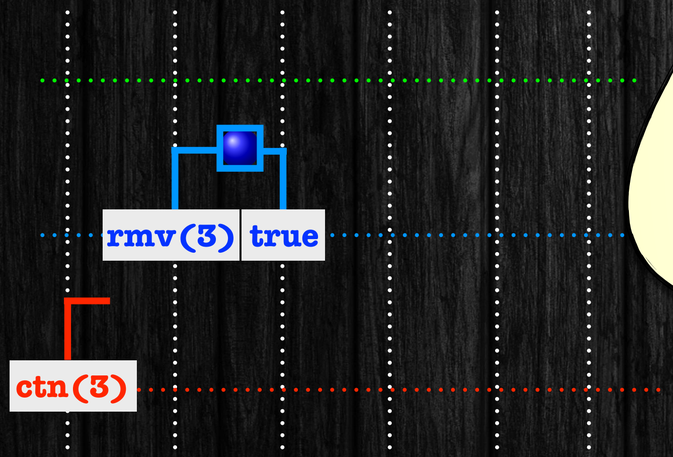
```
ctn(e):
local c, b
1  c := Head;
2  while (c.val < e)
3     c := c.next
4  b := c.mark
5  if (!b && c.val = e)
6     return true;
7  else return false;
```

ctn(3)

not executed yet







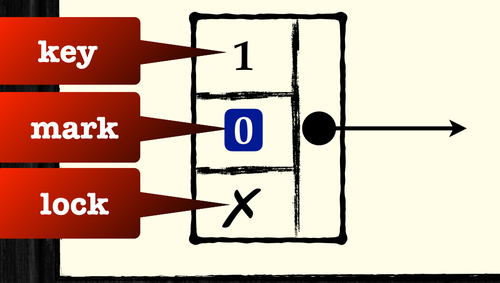
add(3)

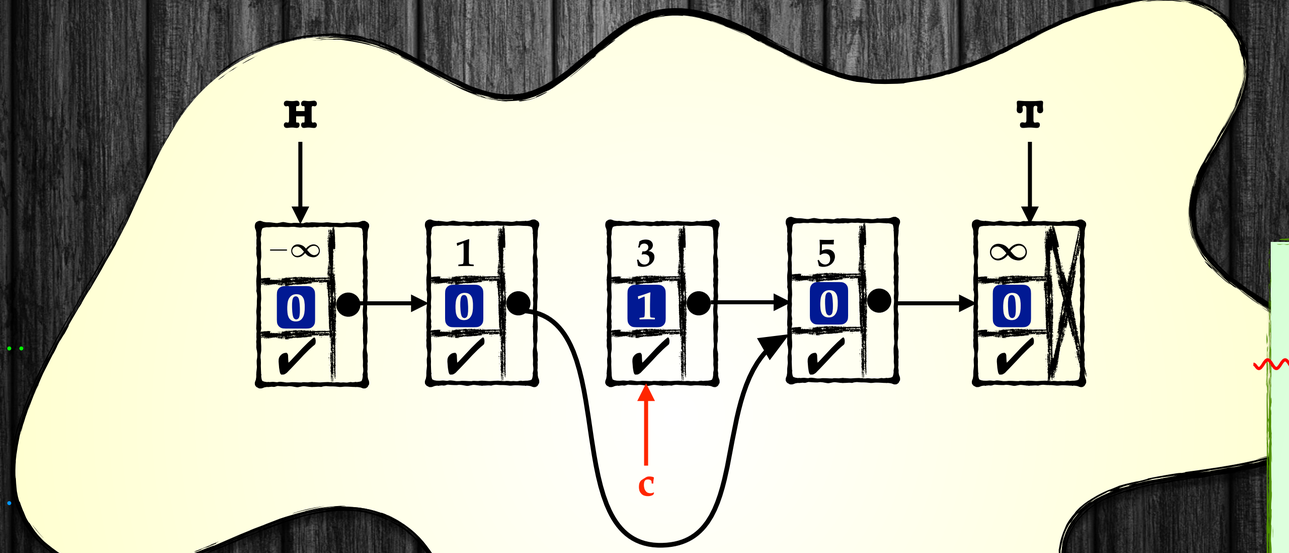
```
add(e):
   local p, c, n, r
   1  (p,c) := locate(e);
   2  if (c.val <> e)
   3     n := new Node(0,e,c,false);
   4     p.next := n;
   5     r := true;
   6  else r := false;
   7  unlock(p);
   8  unlock(c);
   9  return r;
```

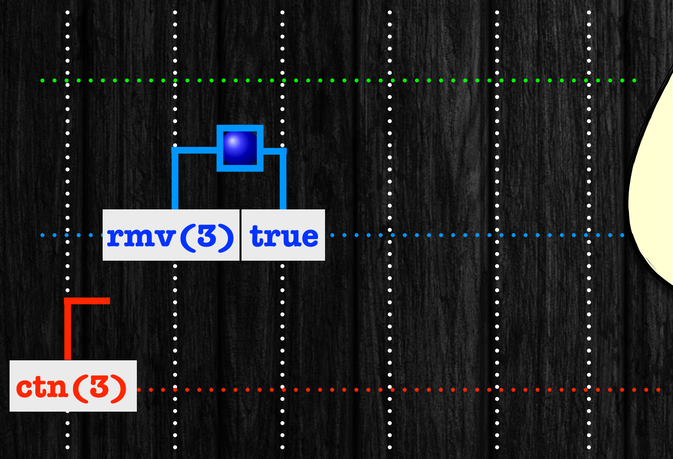
```
ctn(e):
local c, b
1  c := Head;
2  while (c.val < e)
3     c := c.next
4  b := c.mark
5  if (!b && c.val = e)
6     return true;
7  else return false;
```

ctn(3)

not executed yet







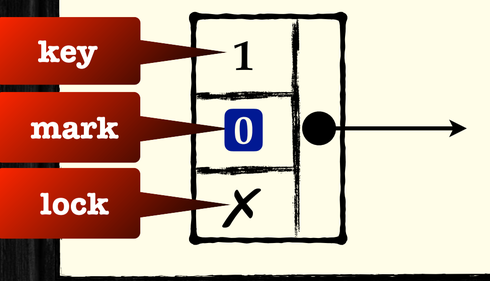
add(3)

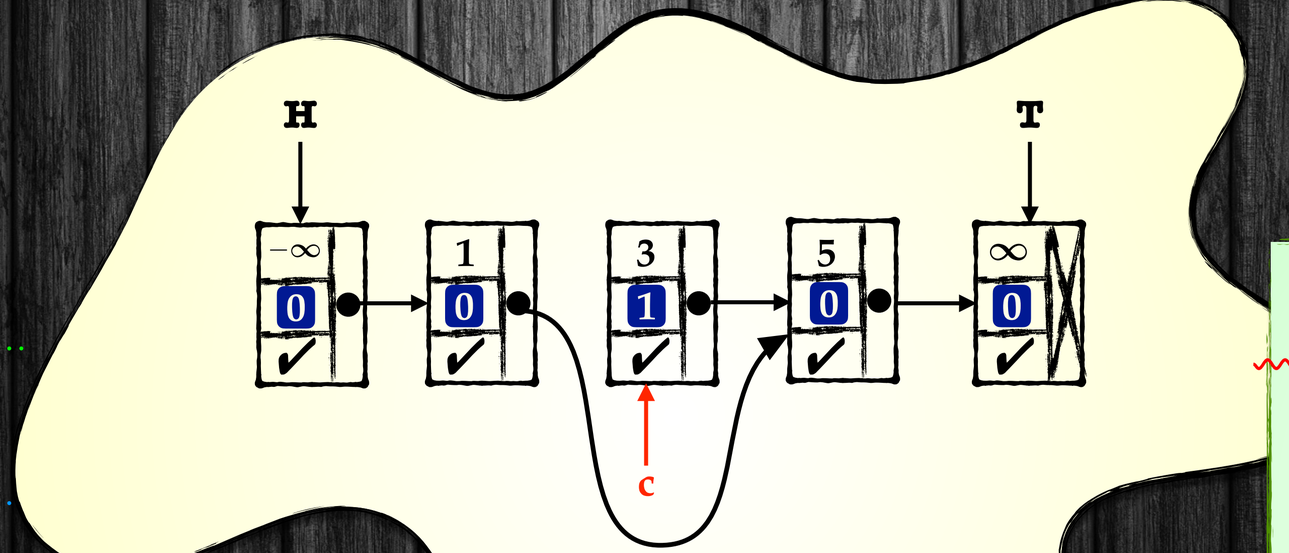
```
add(e):
   local p, c, n, r
1  (p,c) := locate(e);
2  if (c.val <> e)
3     n := new Node(0,e,c,false);
4     p.next := n;
5     r := true;
6  else r := false;
7  unlock(p);
8  unlock(c);
9  return r;
```

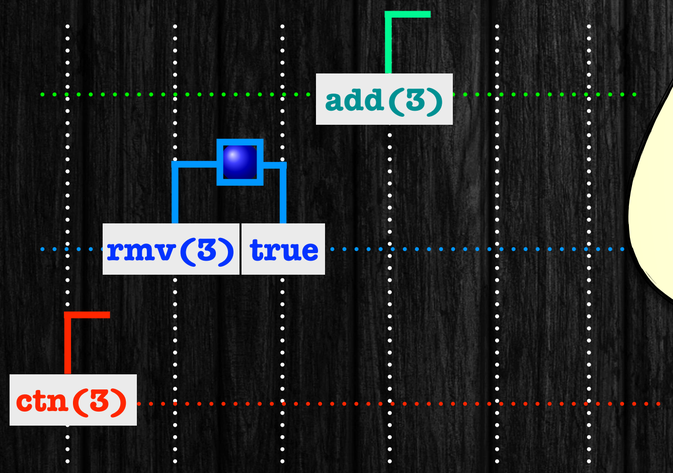
```
ctn(e):
local c, b
1  c := Head;
2  while (c.val < e)
3     c := c.next
4  b := c.mark
5  if (!b && c.val = e)
6     return true;
7  else return false;
```

ctn(3)

not executed yet







add(3)

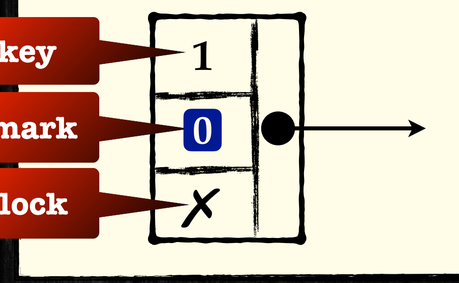
```
add(e):
   local p, c, n, r
1  (p,c) := locate(e);
2  if (c.val <> e)
3     n := new Node(0,e,c,false);
4     p.next := n;
5     r := true;
6  else r := false;
7  unlock(p);
8  unlock(c);
9  return r;
```

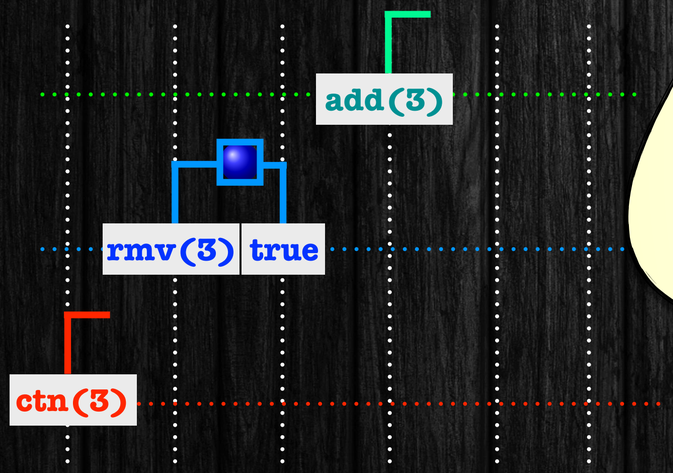
Sets | Lazy Synchronization

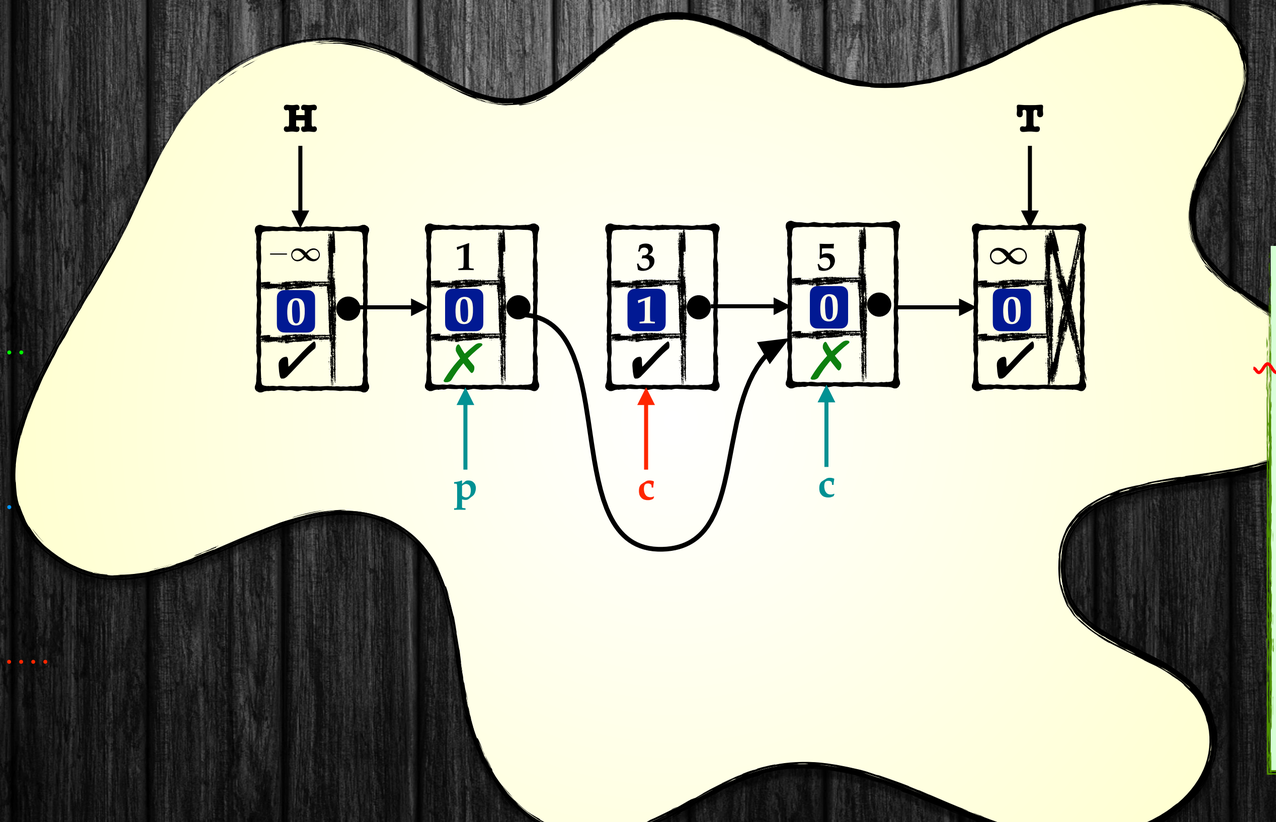
```
ctn(e):
local c, b
1  c := Head;
2  while (c.val < e)
3     c := c.next
4  b := c.mark
5  if (!b && c.val = e)
6     return true;
7  else return false;
```

ctn(3)

not executed yet







add(3)

```
add(e):
   local p, c, n, r
1  (p,c) := locate(e);
2  if (c.val <> e)
3     n := new Node(0,e,c,false);
4     p.next := n;
5     r := true;
6  else r := false;
7  unlock(p);
8  unlock(c);
9  return r;
```

Sets | Lazy Synchronization

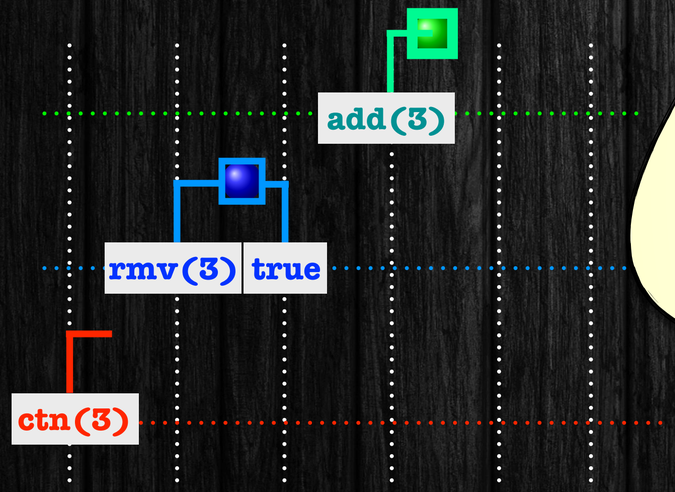
```
ctn(e):
local c, b
1  c := Head;
2  while (c.val < e)
3     c := c.next
4  b := c.mark
5  if (!b && c.val = e)
6     return true;
7  else return false;
```

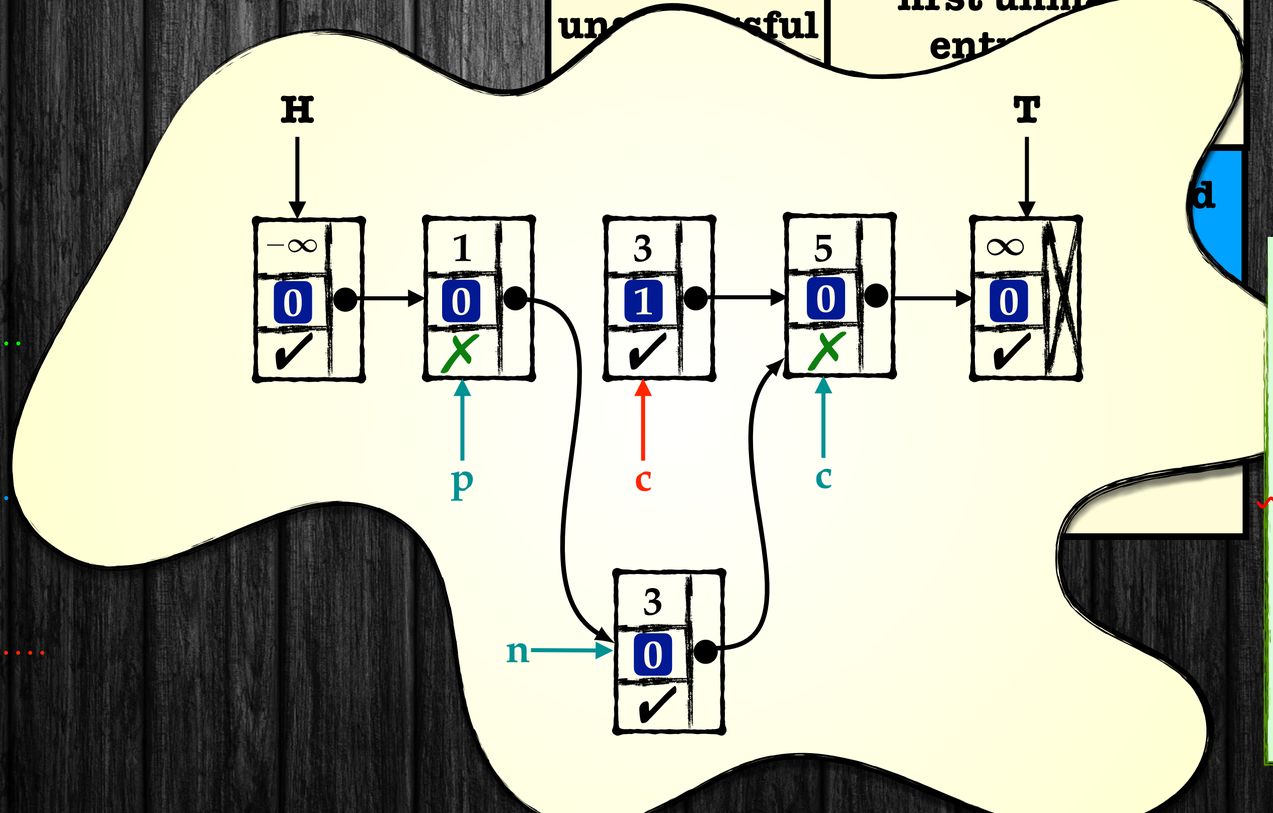
ctn(3)

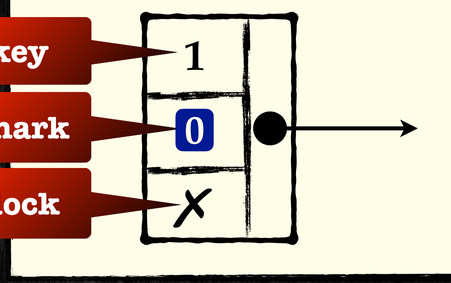
not executed yet

linearization point

successful add	“when p.next is set”
unsuccessful add	“when unmarked entry is observed”
successful rmv	“when the cell is marked”
un… sful	“first unmar… ent…”







add(3)

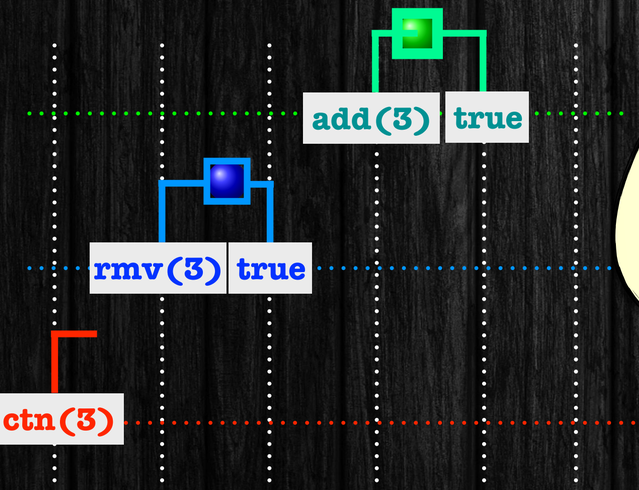
```
add(e):
   local p, c, n, r
   1  (p,c) := locate(e);
   2  if (c.val <> e)
   3     n := new Node(0,e,c,false);
   4     p.next := n;
   5     r := true;
   6  else r := false;
   7  unlock(p);
   8  unlock(c);
   9  return r;
```

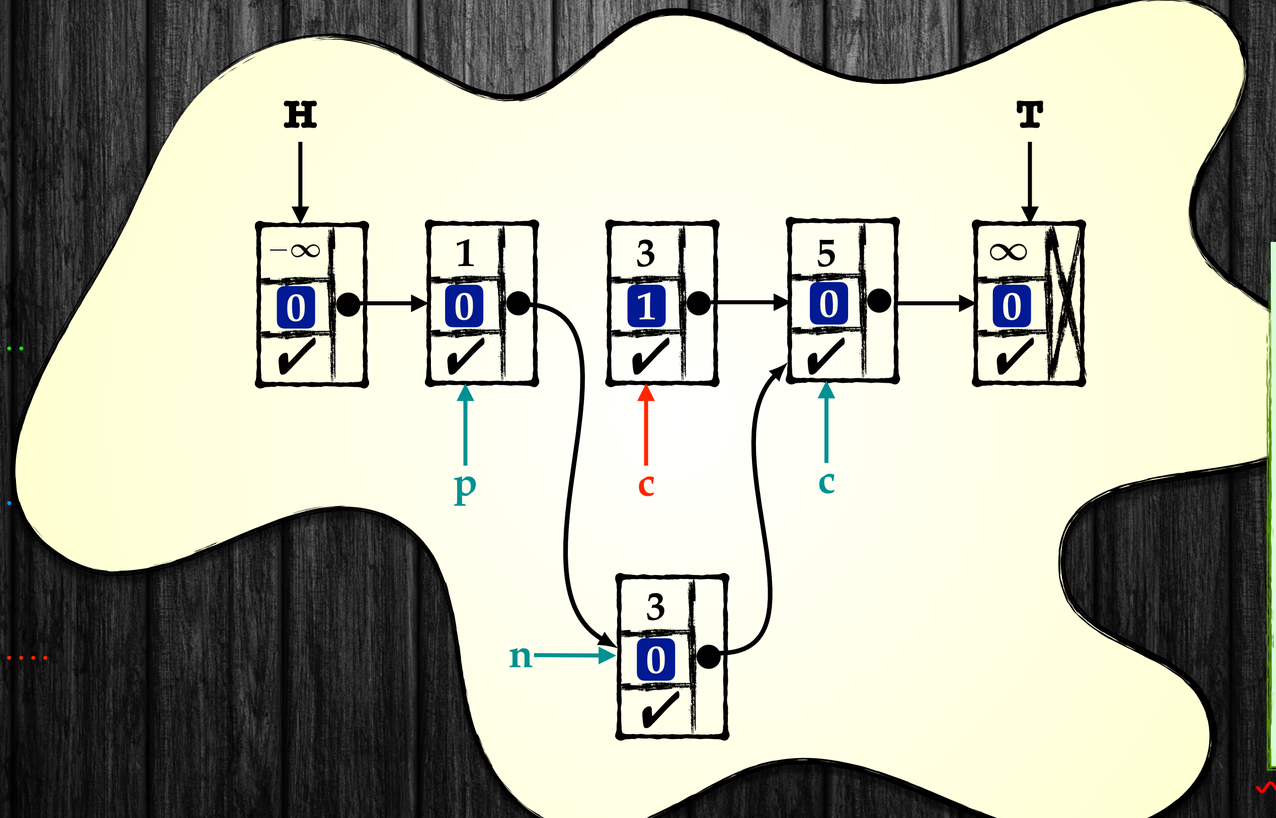
Sets | Lazy Synchronization

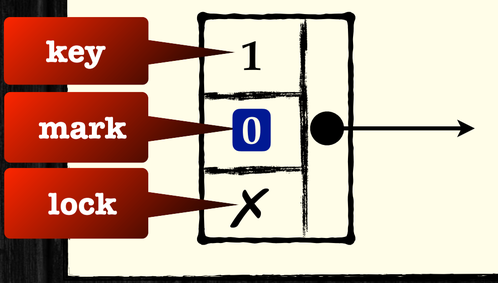
```
ctn(e):
local c, b
1  c := Head;
2  while (c.val < e)
3     c := c.next
4  b := c.mark
5  if (!b && c.val = e)
6     return true;
7  else return false;
```

ctn(3)

not executed yet







add(3)

```
add(e):
   local p, c, n, r
1  (p,c) := locate(e);
2  if (c.val <> e)
3     n := new Node(0,e,c,false);
4     p.next := n;
5     r := true;
6  else r := false;
7  unlock(p);
8  unlock(c);
9  return r;
```

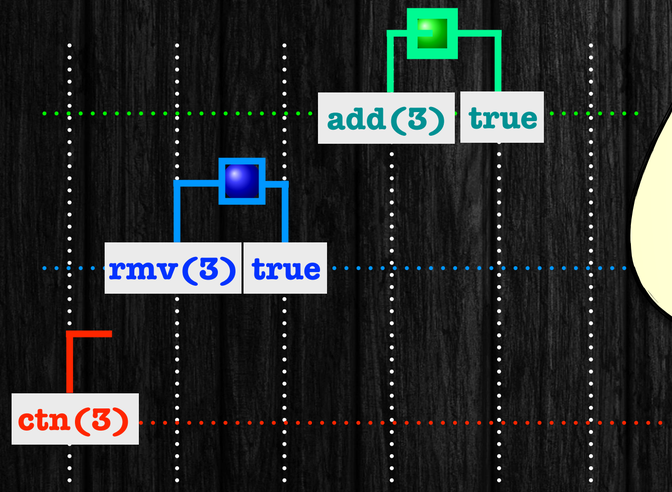
Sets | Lazy Synchronization

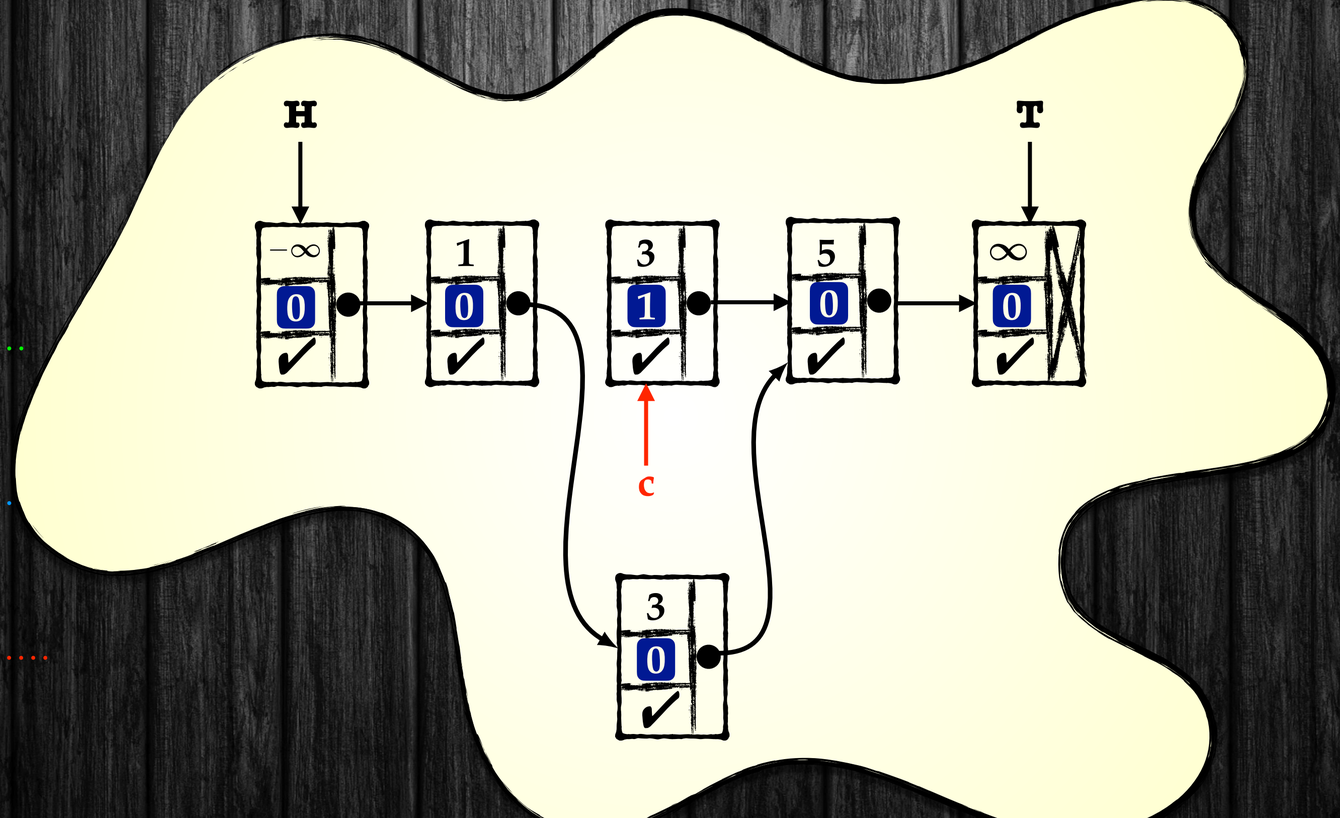
```
ctn(e):
local c, b
1  c := Head;
2  while (c.val < e)
3     c := c.next
4  b := c.mark
5  if (!b && c.val = e)
6     return true;
7  else return false;
```

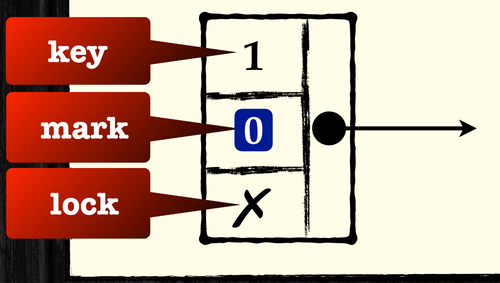
ctn(3)

not executed yet

becomes active again

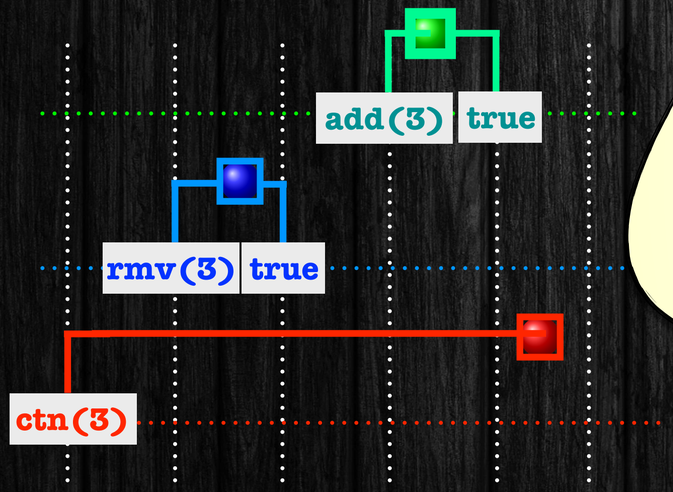


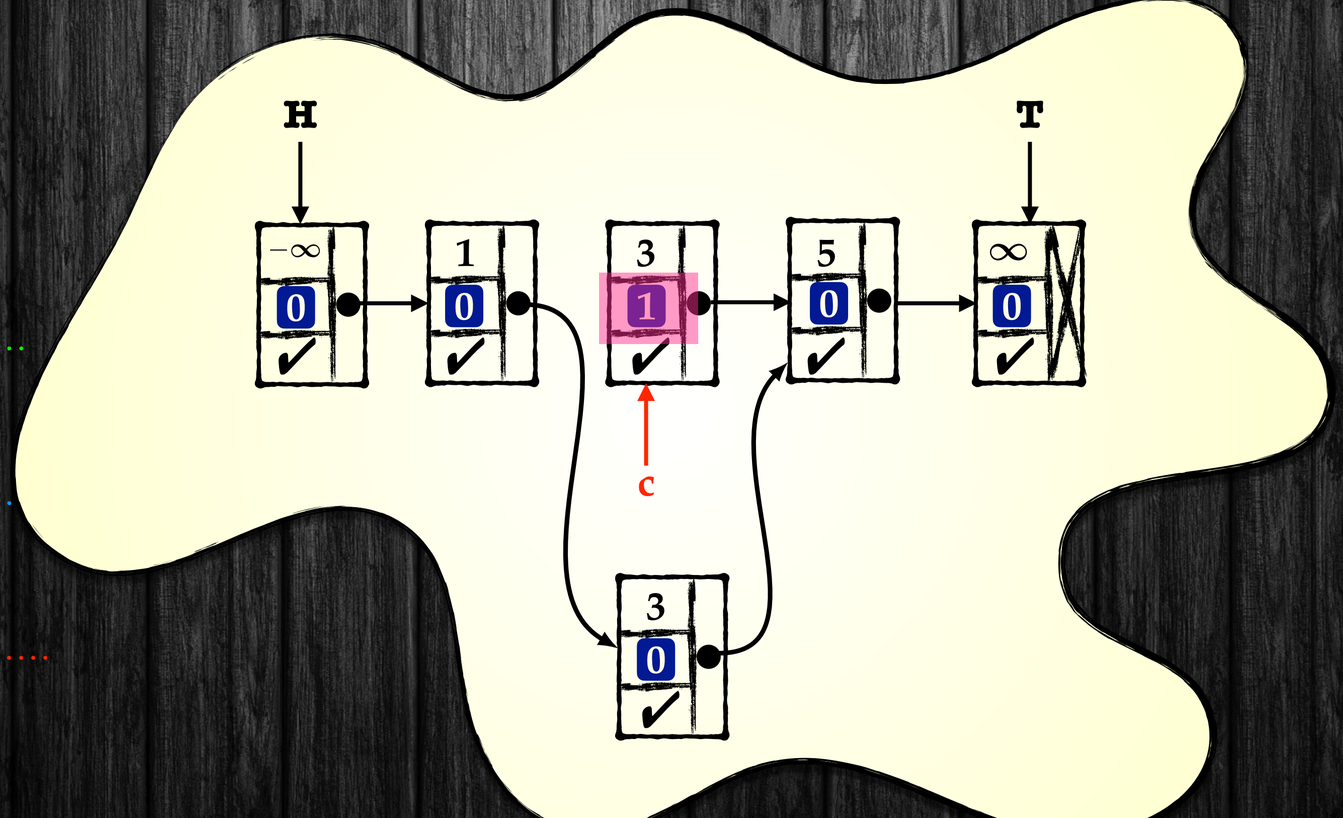


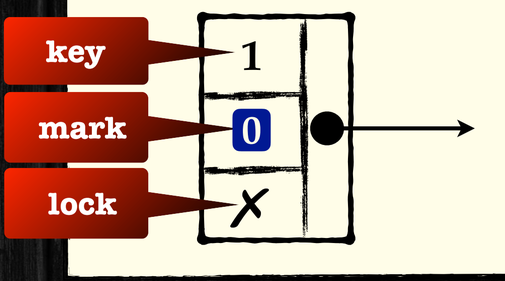


```
ctn(e):
local c, b
1  c := Head;
2  while (c.val < e)
3     c := c.next
4  b := c.mark
5  if (!b && c.val = e)
6     return true;
7  else return false;
```

ctn(3)



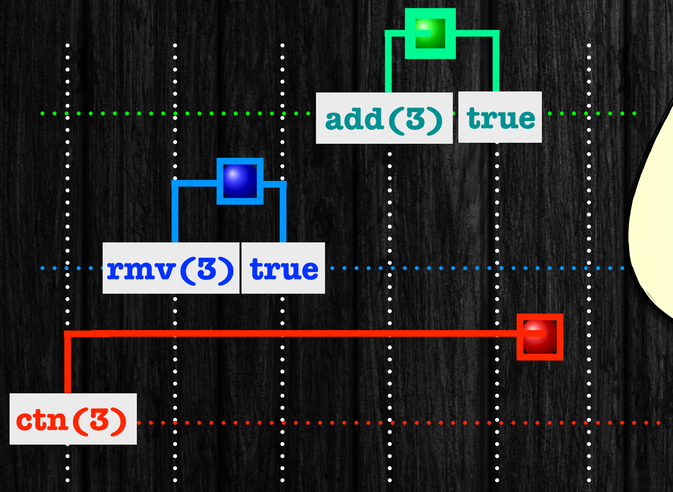


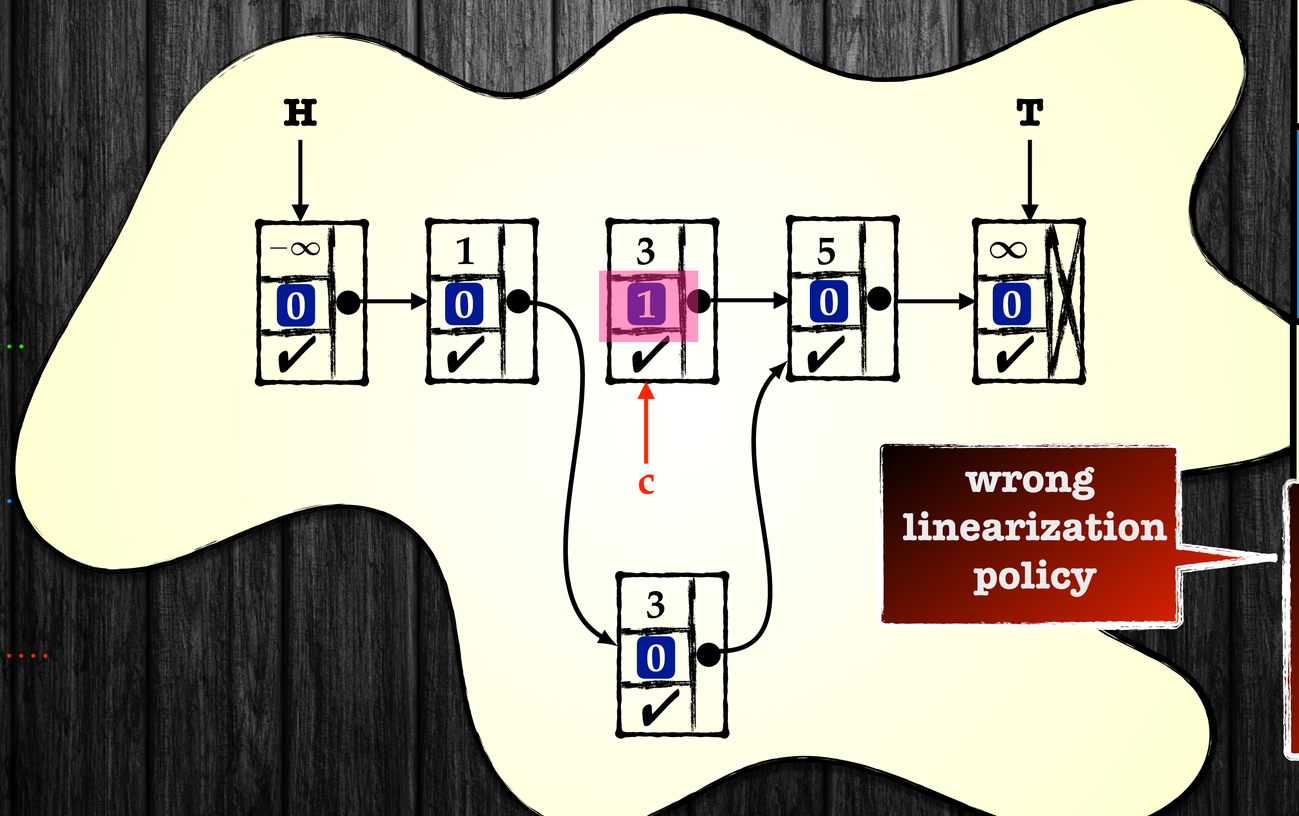


Sets | Lazy Synchronization

```
ctn(e):
local c, b
1  c := Head;
2  while (c.val < e)
3     c := c.next
4  b := c.mark
5  if (!b && c.val = e)
6     return true;
7  else return false;
```

ctn(3)





linearization point

wrong linearization policy

successful add	“when p.next is set”
unsuccessful add	“when unmarked entry is observed”
successful rmv	“when the cell is marked”
unsuccessful rmv	“first unmarked entry with larger key”
successful ctn	“when unmarked entry with input key found”
unsuccessful ctn	?

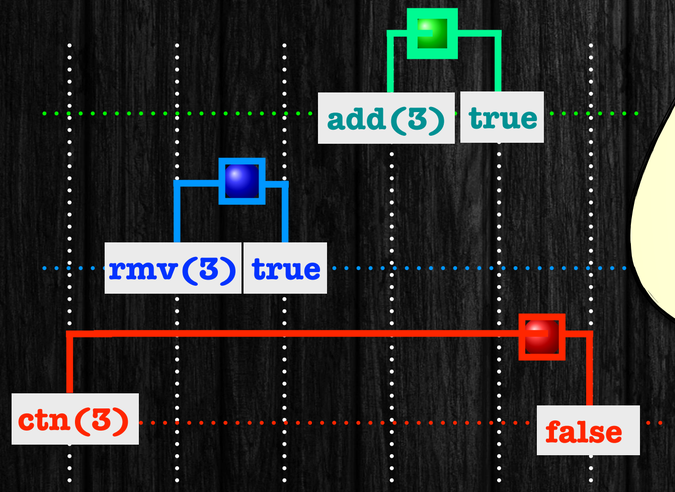
“when finding the cell with the input key marked”

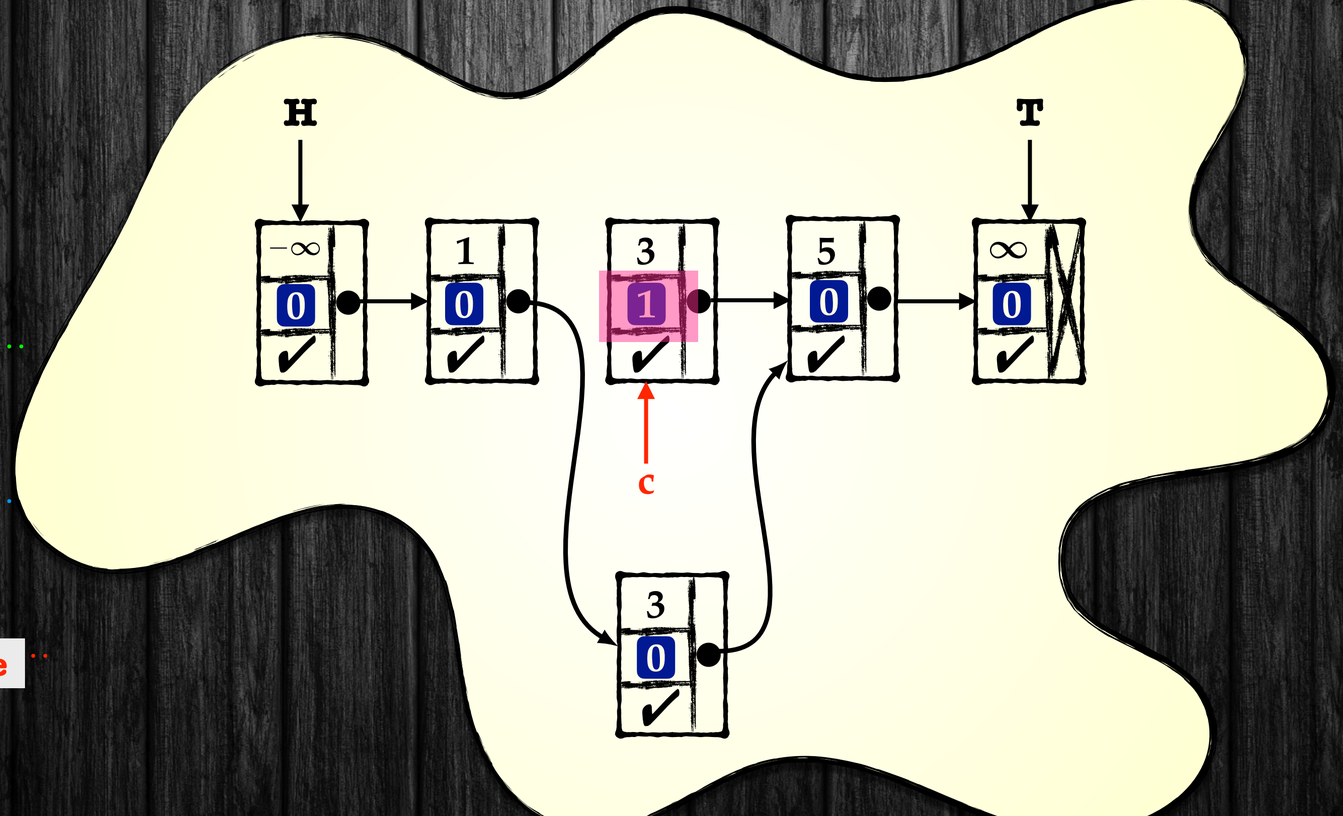
“complicated linearization policy”

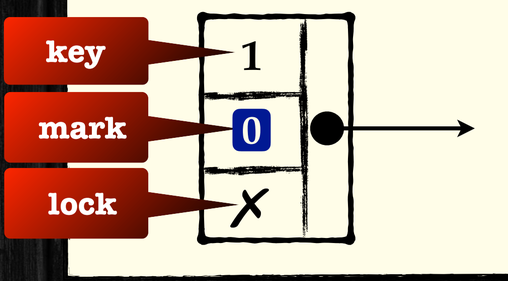
Sets | Lazy Synchronization

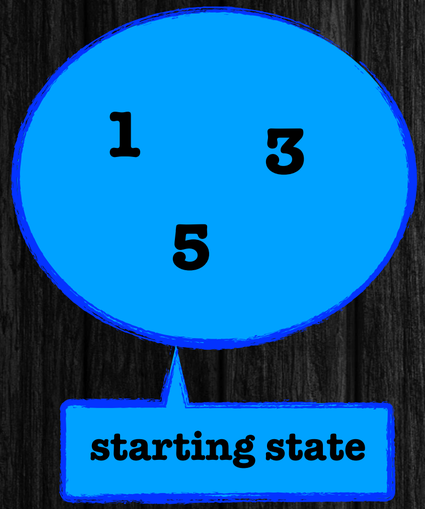
```
ctn(e):
local c, b
1  c := Head;
2  while (c.val < e)
3     c := c.next
4  b := c.mark
5  if (!b && c.val = e)
6     return true;
7  else return false;
```

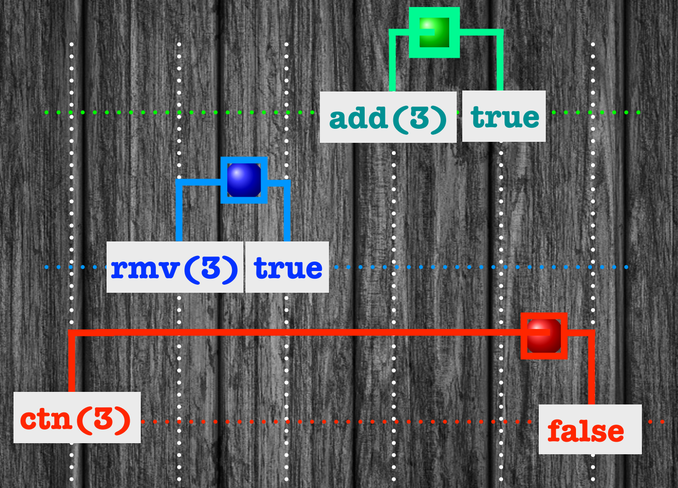
ctn(3)

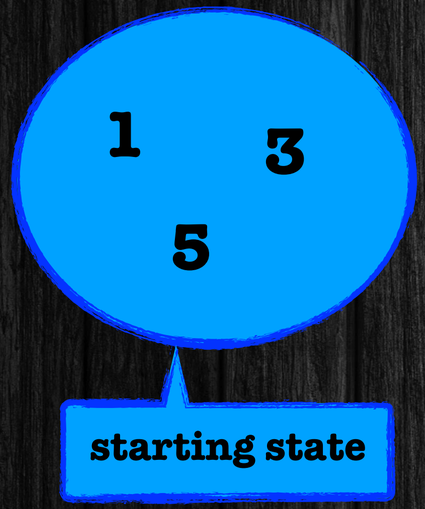


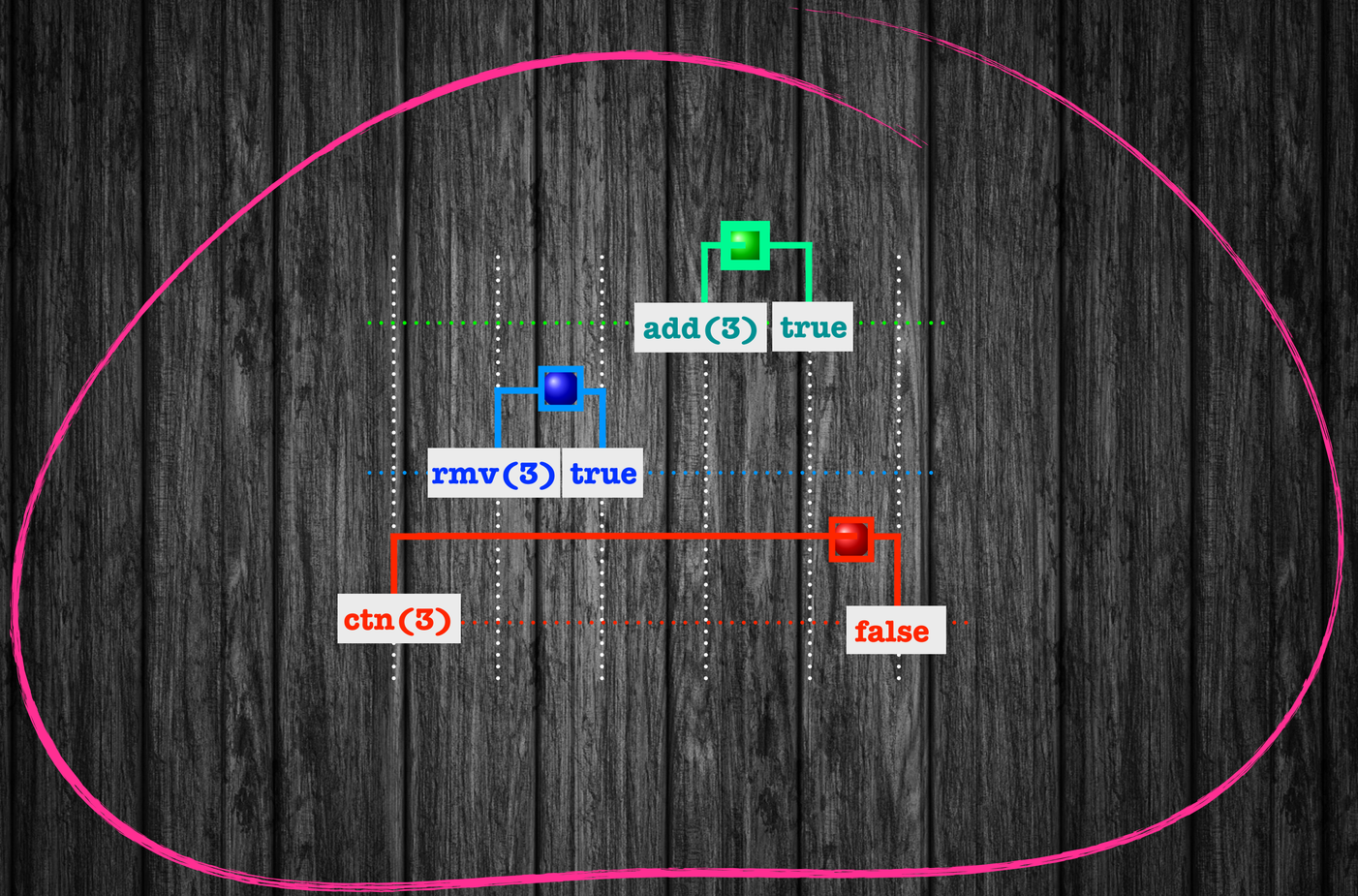


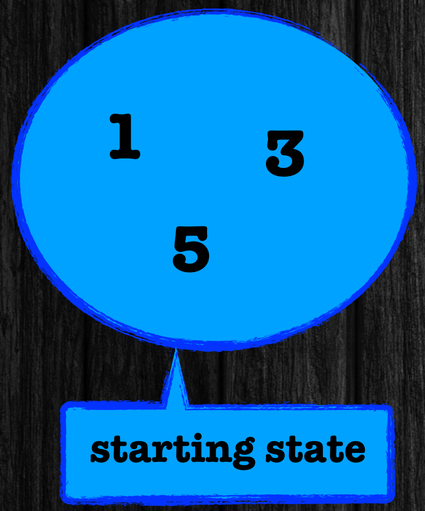


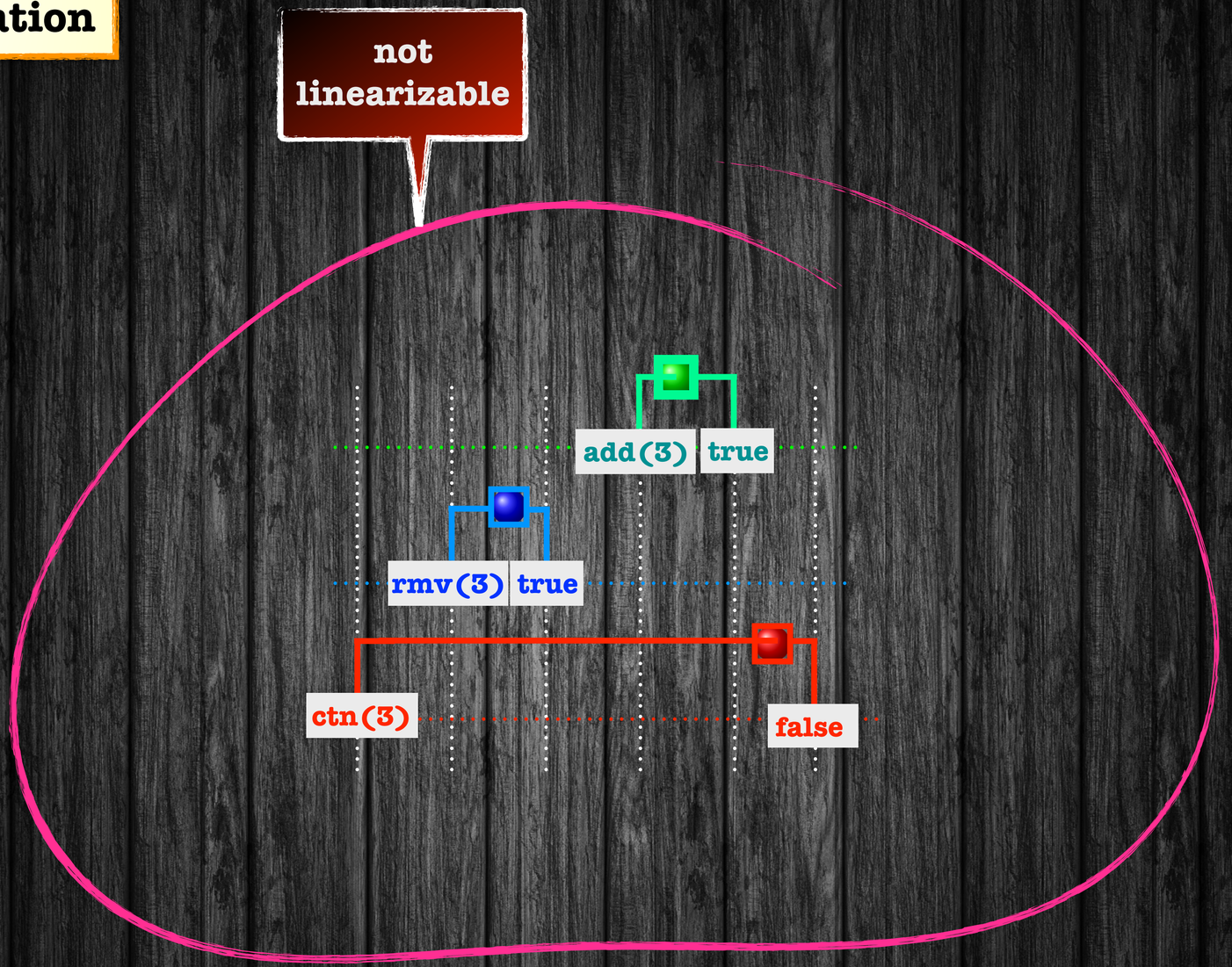

1
3
5
starting state


add(3)
true
rmv(3)
true
ctn(3)
false

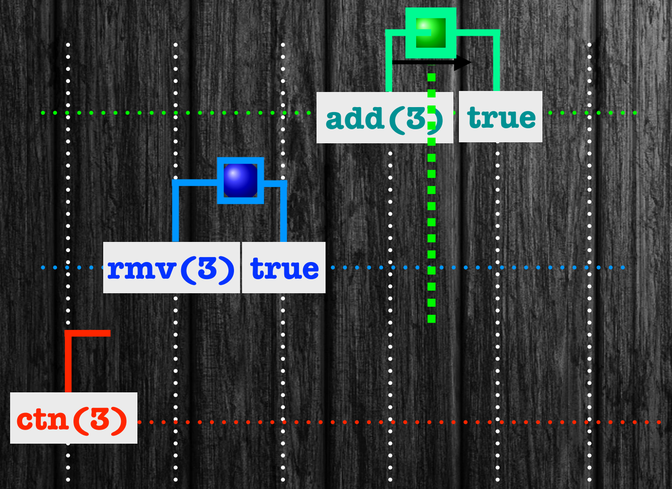

1
3
5
starting state


add(3)
true
rmv(3)
true
ctn(3)
false


1
3
5
starting state


not linearizable
add(3)
true
rmv(3)
true
ctn(3)
false

Sets | Lazy Synchronization

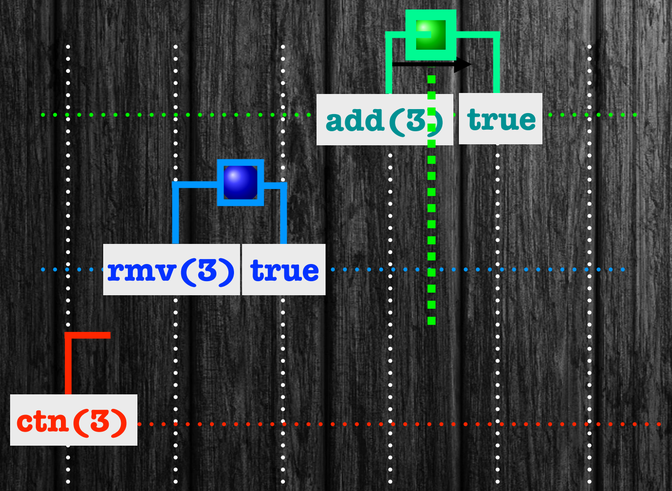


linearization point

the earliest time point where

1. we find node with key > input key
2. find marked node with key = input key
3. just before a new node added with key = input key

unsuccessful add	“when unmark entry is observe
successful rmv	“when the cel is marked”
unsuccessful rmv	“first unmarke entry with larger key”
successful ctn	“when unmark entry with input key found
unsuccessful ctn	?



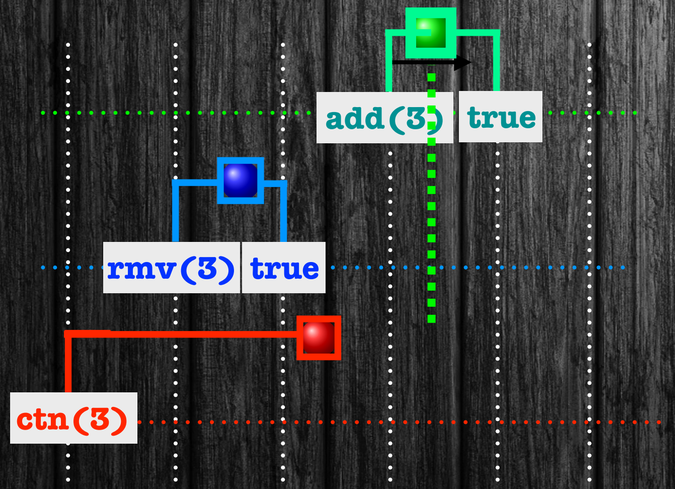
linearization point

the earliest time point where

1. we find node with key > input key
2. find marked node with key = input key
3. just before a new node added with key = input key

unsuccessful add	“when unmark entry is observe
successful rmv	“when the cel is marked”
unsuccessful rmv	“first unmarke entry with larger key”
successful ctn	“when unmark entry with input key found
unsuccessful ctn	?

correct linearization policy



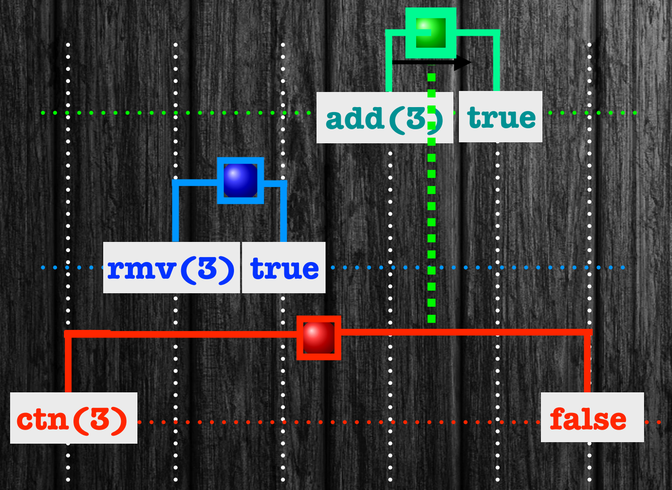
linearization point

the earliest time point where

1. we find node with key > input key
2. find marked node with key = input key
3. just before a new node added with key = input key

unsuccessful add	“when unmark entry is observe
successful rmv	“when the cel is marked”
unsuccessful rmv	“first unmarke entry with larger key”
successful ctn	“when unmark entry with input key found
unsuccessful ctn	?

correct linearization policy



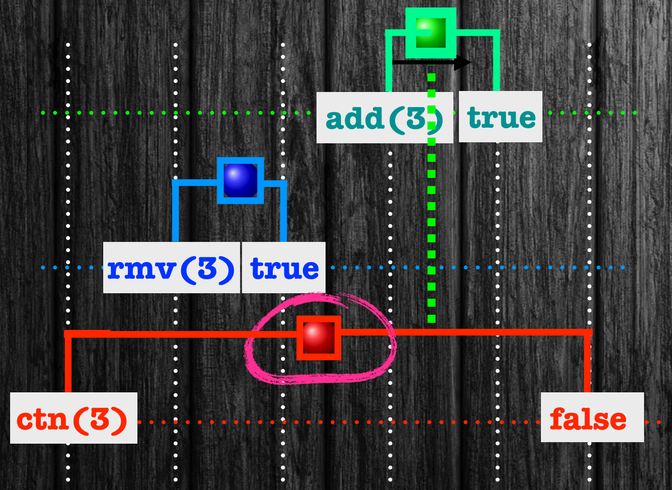
linearization point

the earliest time point where

1. we find node with key > input key
2. find marked node with key = input key
3. just before a new node added with key = input key

unsuccessful add	“when unmark entry is observe
successful rmv	“when the cel is marked”
unsuccessful rmv	“first unmarke entry with larger key”
successful ctn	“when unmark entry with input key found
unsuccessful ctn	?

correct linearization policy



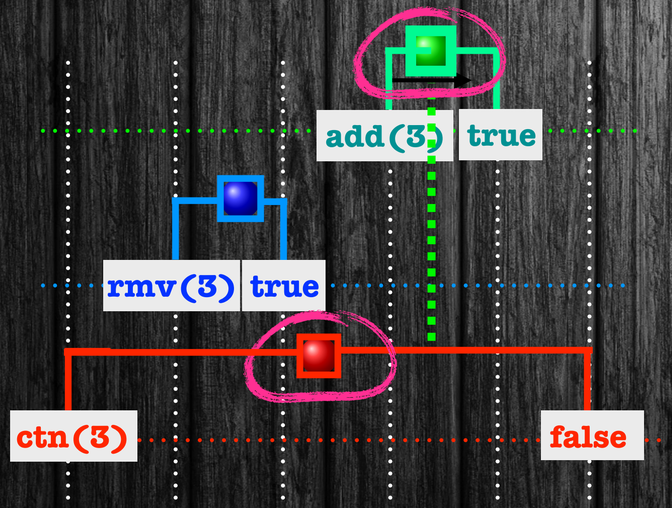
linearization point

the earliest time point where

1. we find node with key > input key
2. find marked node with key = input key
3. just before a new node added with key = input key

unsuccessful add	“when unmark entry is observe
successful rmv	“when the cel is marked”
unsuccessful rmv	“first unmarke entry with larger key”
successful ctn	“when unmark entry with input key found
unsuccessful ctn	?

correct linearization policy



linearization point

unsuccessful add	"when unmark entry is observe
successful rmv	"when the cel is marked"
unsuccessful rmv	"first unmarke entry with larger key"
successful ctn	"when unmark entry with input key found
unsuccessful ctn	?

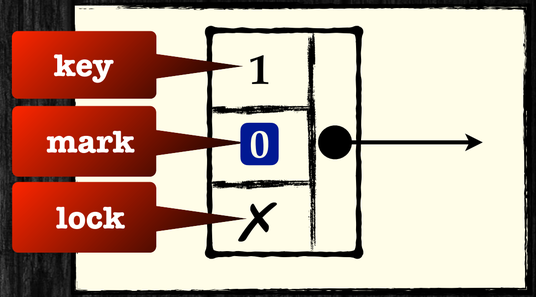
the earliest time point where

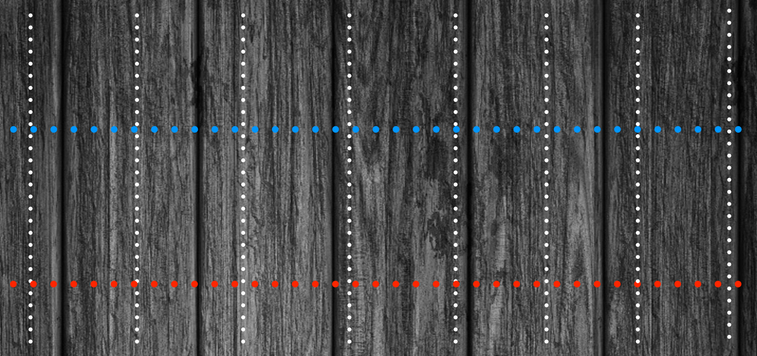
1. we find node with key > input key
2. find marked node with key = input key
3. just before a new node added with key = input key

correct linearization policy

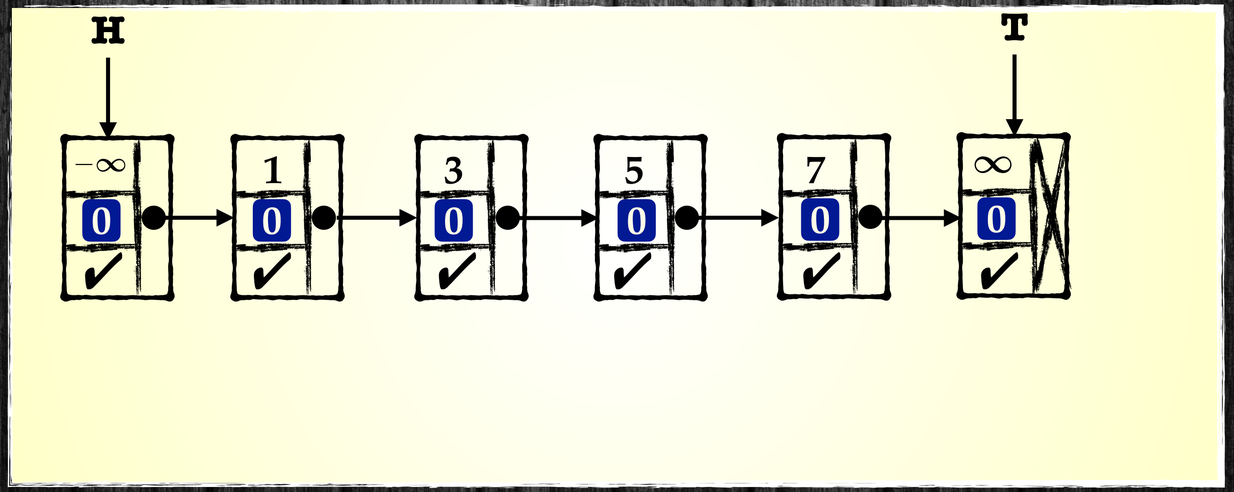
Scenario 1

- no logical deletion
- only physical deletion



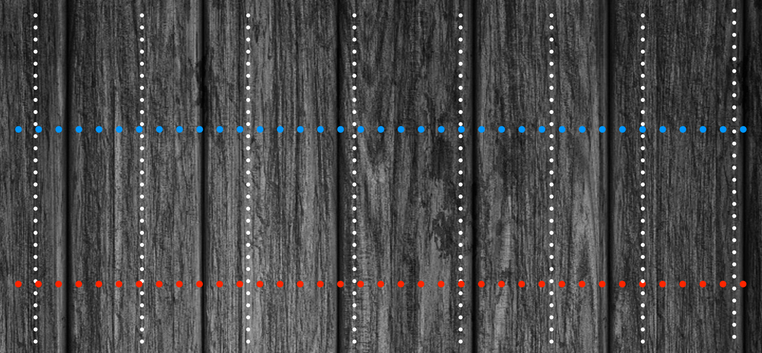


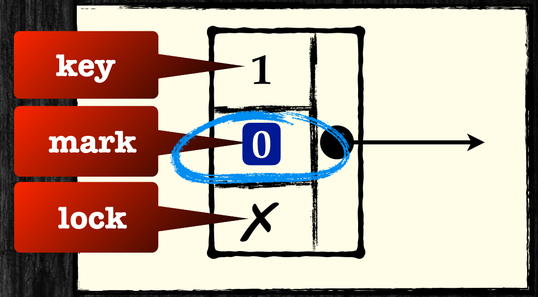
```
rmv(e):
local p, c, n, r
1  (p,c) := locate(e);
2  if (c.val = e)
3     c.mark := true
4     n := c.next;
5     p.next := n;
6     r := true;
7  else r := false;
8  unlock(p);
9  unlock(c);
10 return r;
```



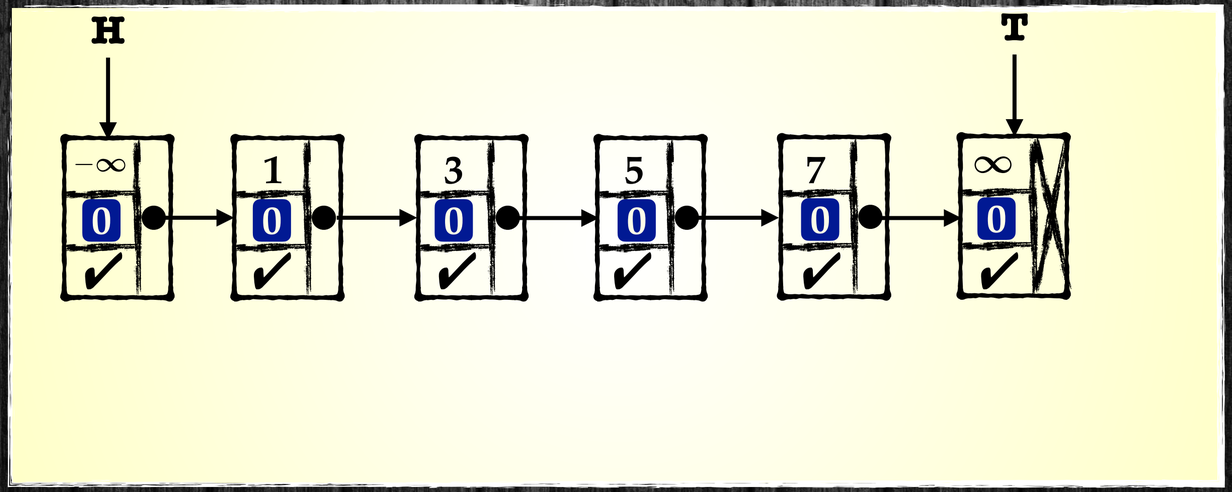
Scenario 1

- no logical deletion
- only physical deletion



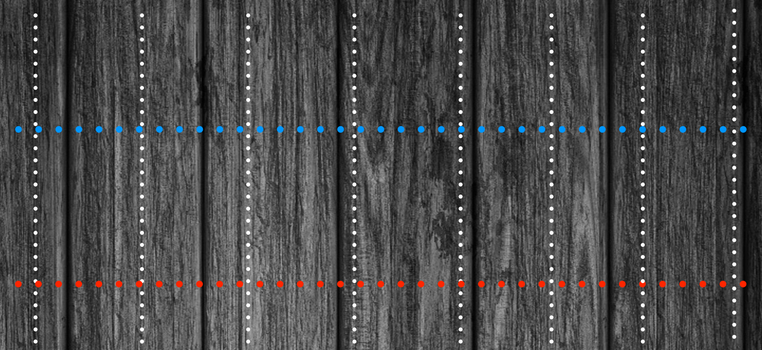


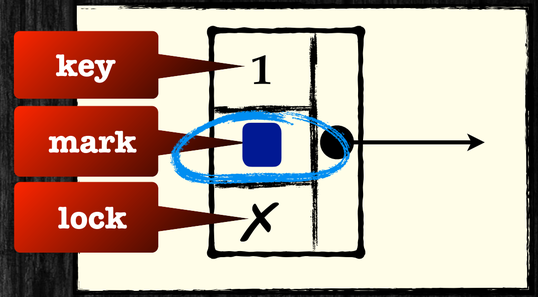

```
rmv(e):
local p, c, n, r
1  (p,c) := locate(e);
2  if (c.val = e)
3     c.mark := true
4     n := c.next;
5     p.next := n;
6     r := true;
7  else r := false;
8  unlock(p);
9  unlock(c);
10 return r;
```

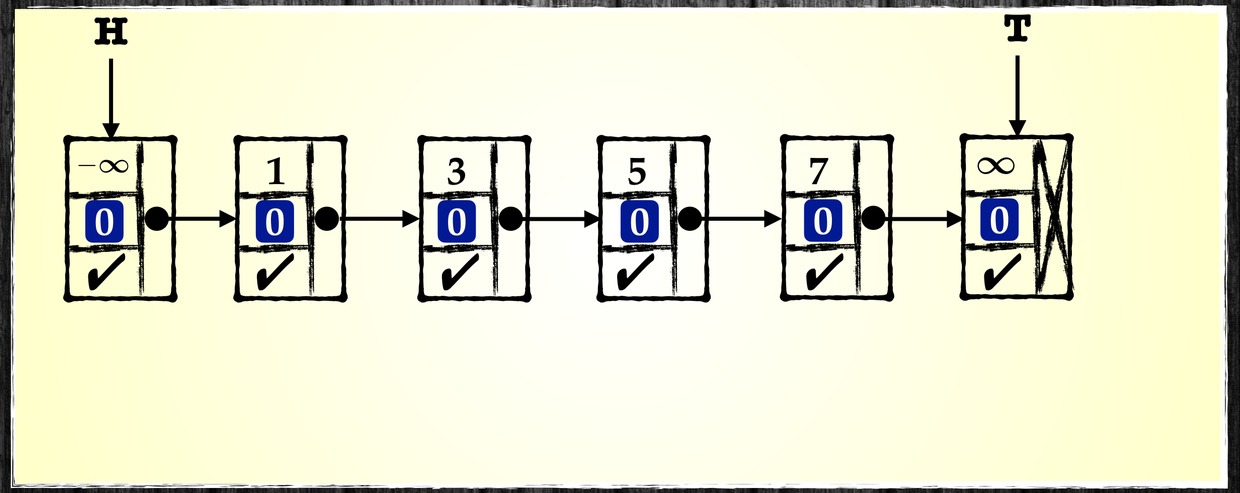


Scenario 1

- no logical deletion
- only physical deletion



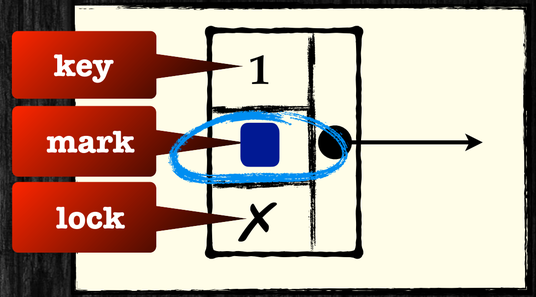


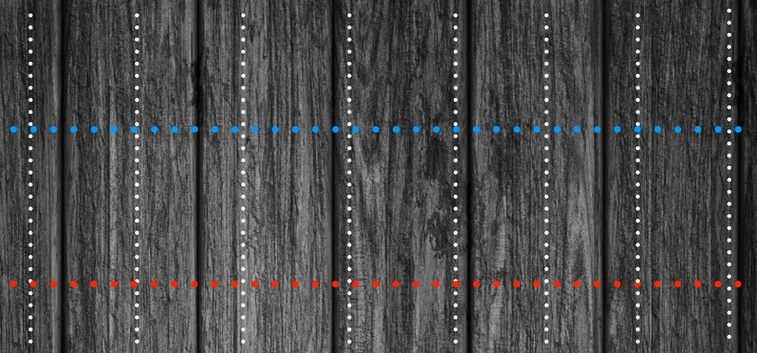


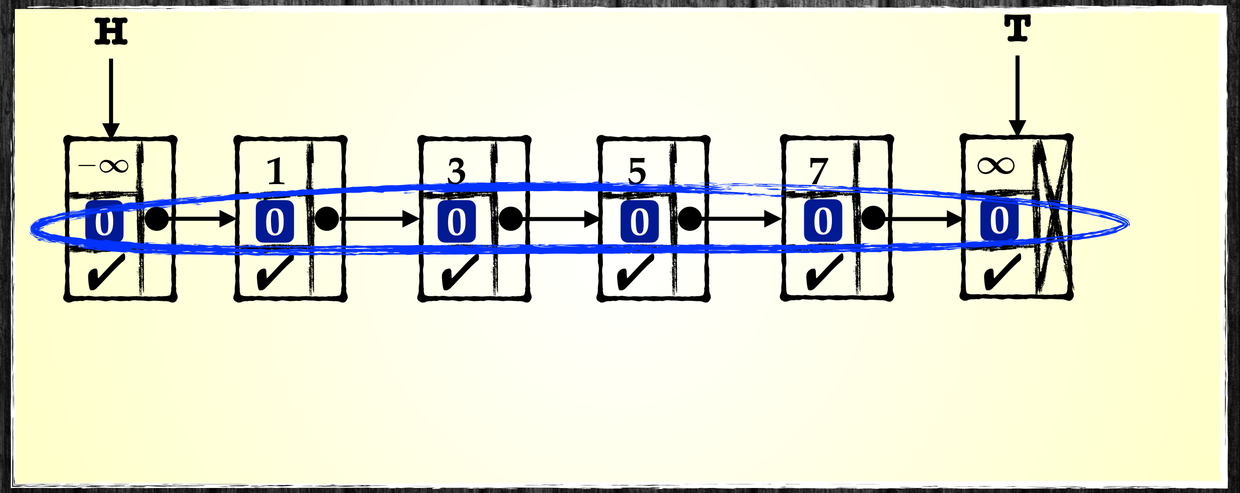

```
rmv(e):
local p, c, n, r
1  (p,c) := locate(e);
2  if (c.val = e)
3     c.mark := true
4     n := c.next;
5     p.next := n;
6     r := true;
7  else r := false;
8  unlock(p);
9  unlock(c);
10 return r;
```

Scenario 1

- no logical deletion
- only physical deletion



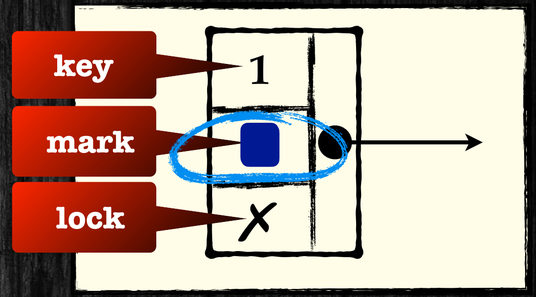


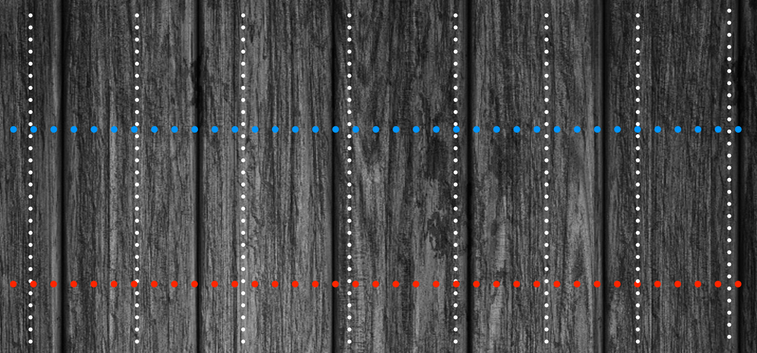


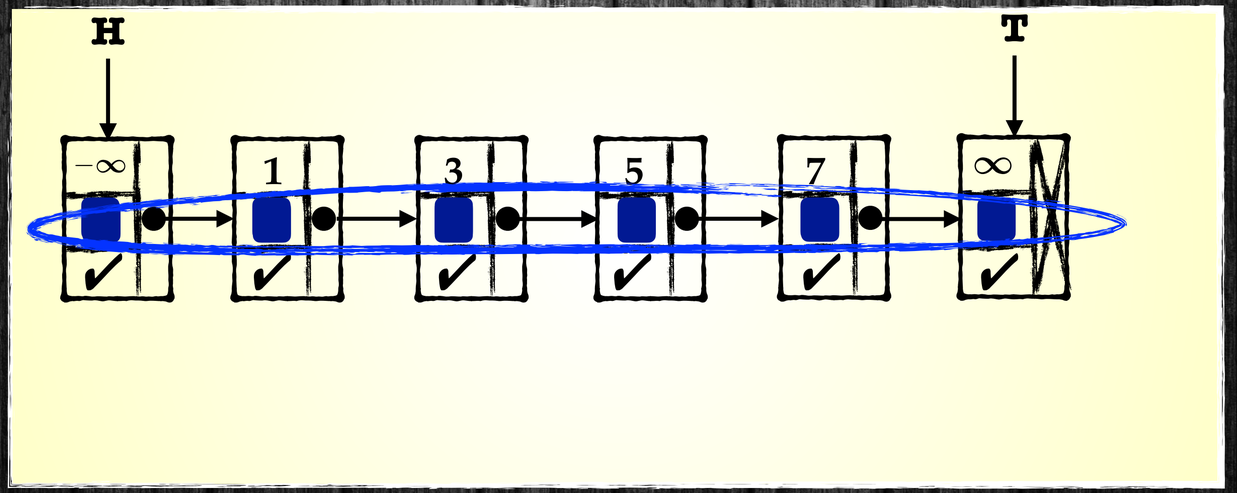

```
rmv(e):
local p, c, n, r
1  (p,c) := locate(e);
2  if (c.val = e)
3     c.mark := true
4     n := c.next;
5     p.next := n;
6     r := true;
7  else r := false;
8  unlock(p);
9  unlock(c);
10 return r;
```

Scenario 1

- no logical deletion
- only physical deletion



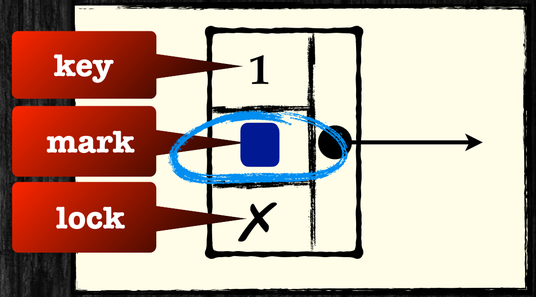


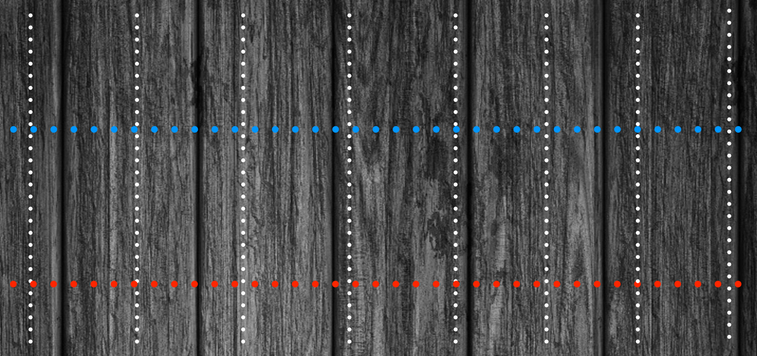


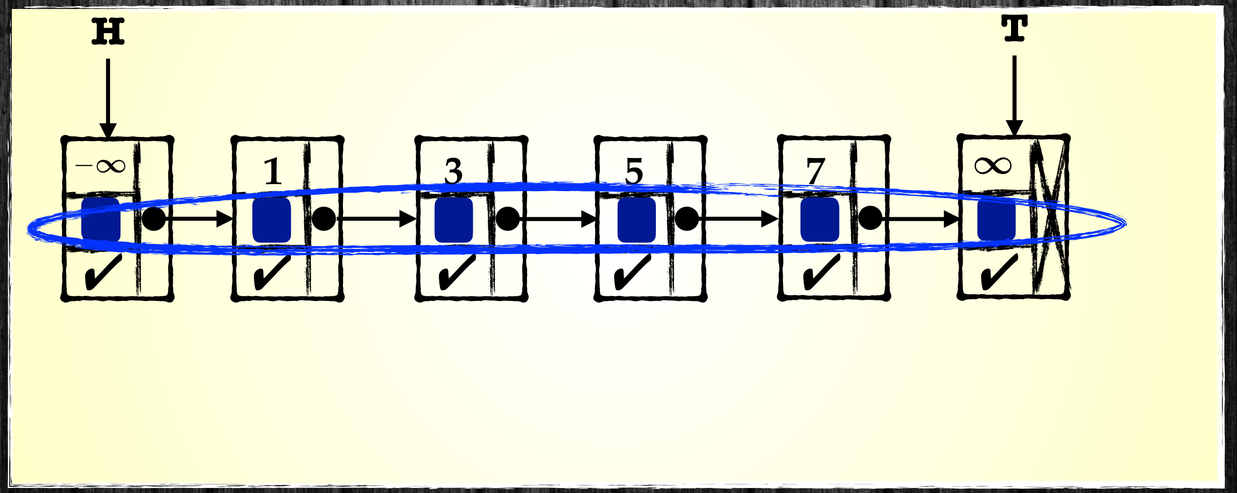

```
rmv(e):
local p, c, n, r
1  (p,c) := locate(e);
2  if (c.val = e)
3     c.mark := true
4     n := c.next;
5     p.next := n;
6     r := true;
7  else r := false;
8  unlock(p);
9  unlock(c);
10 return r;
```

Scenario 1

- no logical deletion
- only physical deletion



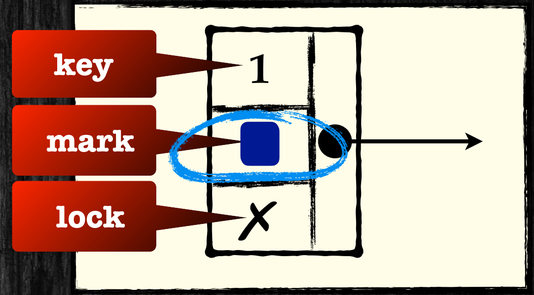


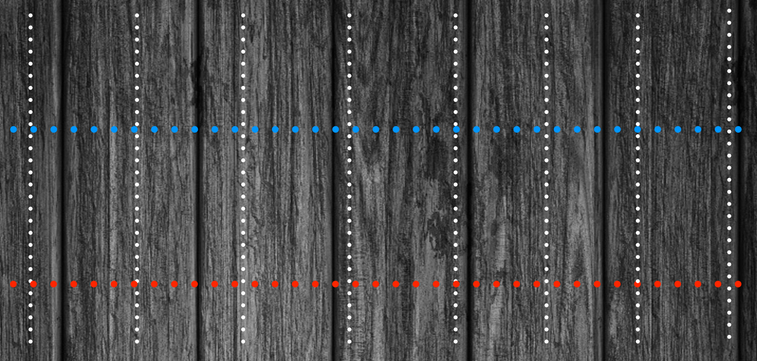



```
rmv(e):
local p, c, n, r
1  (p,c) := locate(e);
2  if (c.val = e)
3    c.mark := true
4    n := c.next;
5    p.next := n;
6    r := true;
7  else r := false;
8  unlock(p);
9  unlock(c);
10 return r;
```

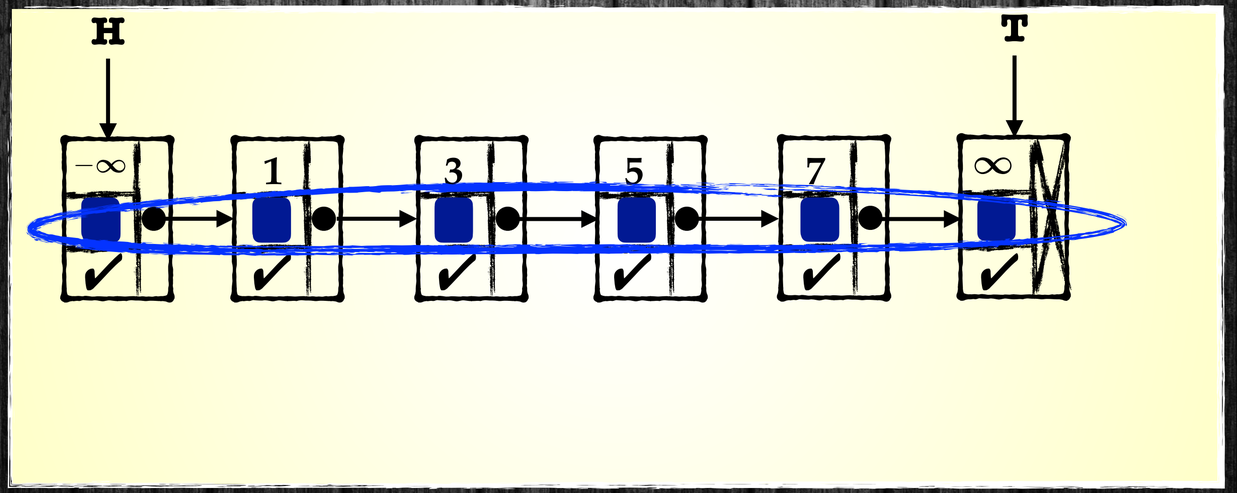
Scenario 1

- no logical deletion
- only physical deletion



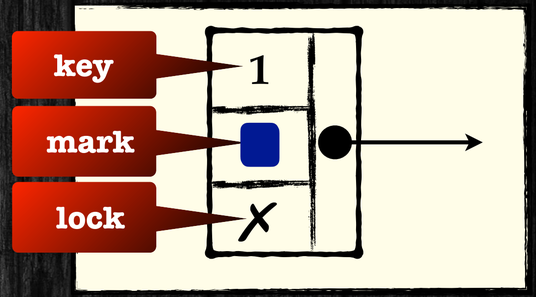


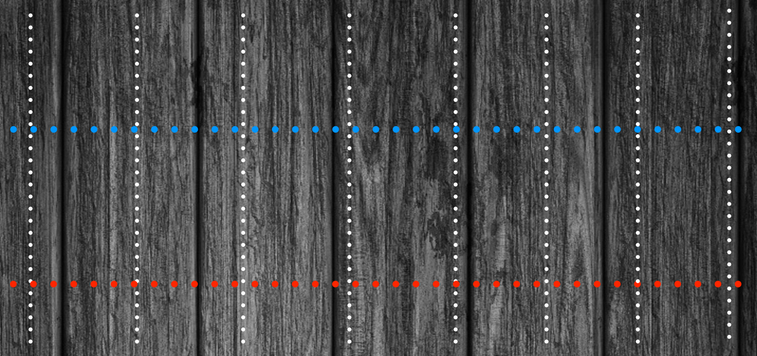
```
rmv(e):
local p, c, n, r
1  (p,c) := locate(e);
2  if (c.val = e)
3
4     n := c.next;
5     p.next := n;
6     r := true;
7  else r := false;
8  unlock(p);
9  unlock(c);
10 return r;
```



Scenario 1

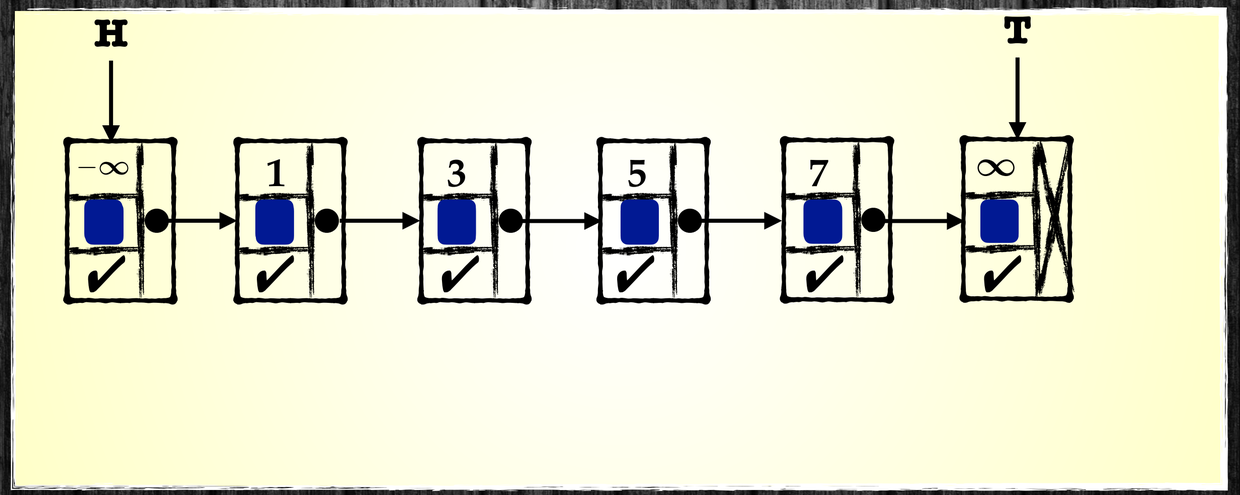
- no logical deletion
- only physical deletion





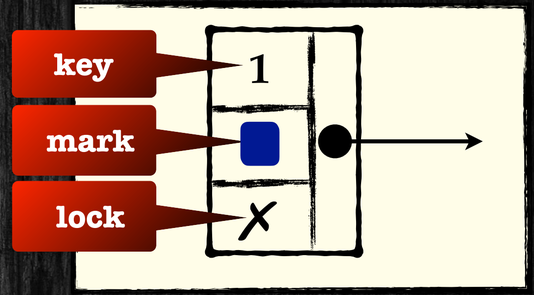
rmv(5)

```
rmv(e):
local p, c, n, r
1  (p,c) := locate(e);
2  if (c.val = e)
3
4     n := c.next;
5     p.next := n;
6     r := true;
7  else r := false;
8  unlock(p);
9  unlock(c);
10 return r;
```



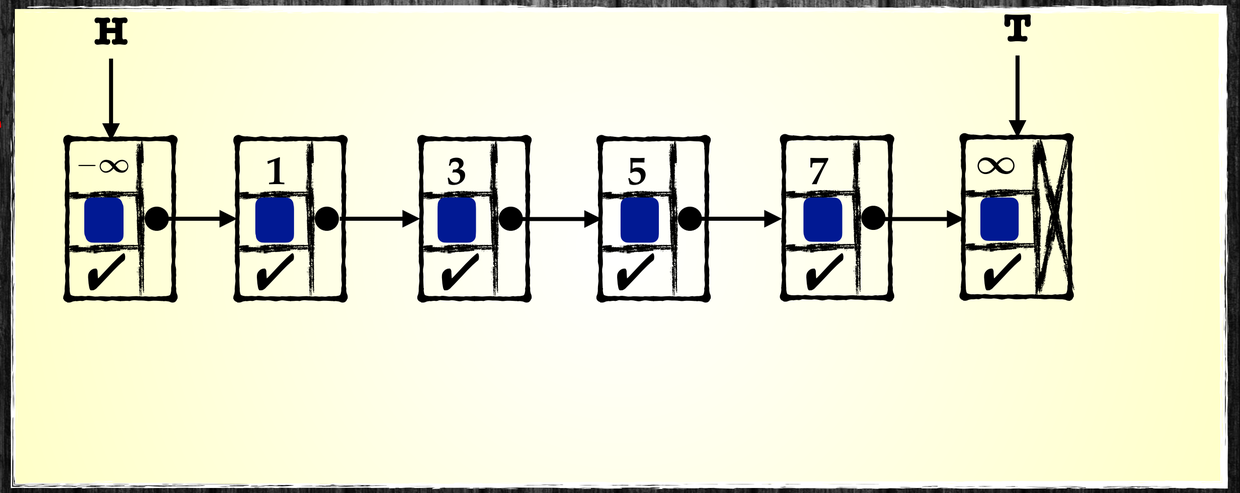
Scenario 1

- no logical deletion
- only physical deletion



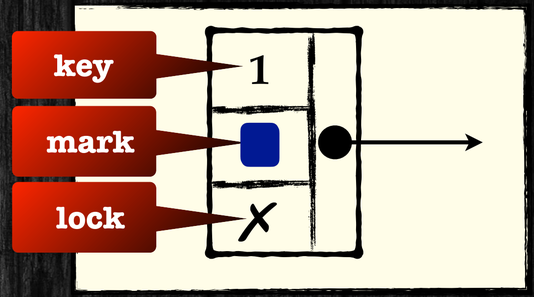
rmv(5)

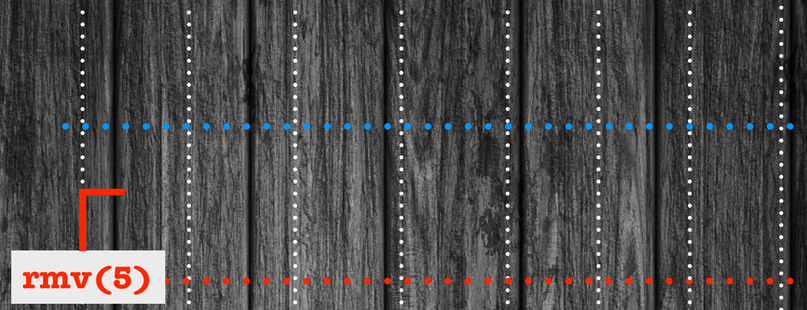
```
rmv(e):
local p, c, n, r
1  (p,c) := locate(e);
2  if (c.val = e)
3
4     n := c.next;
5     p.next := n;
6     r := true;
7  else r := false;
8  unlock(p);
9  unlock(c);
10 return r;
```



Scenario 1

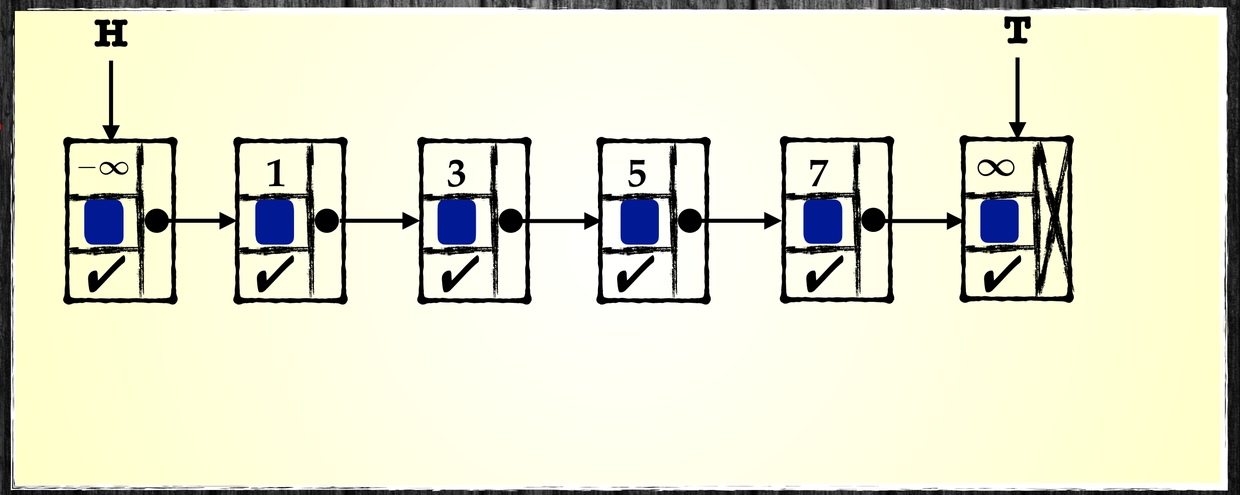
- no logical deletion
- only physical deletion





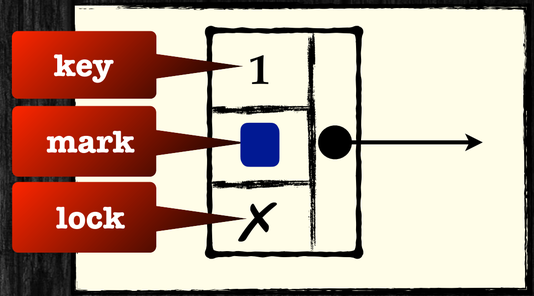
rmv(5)

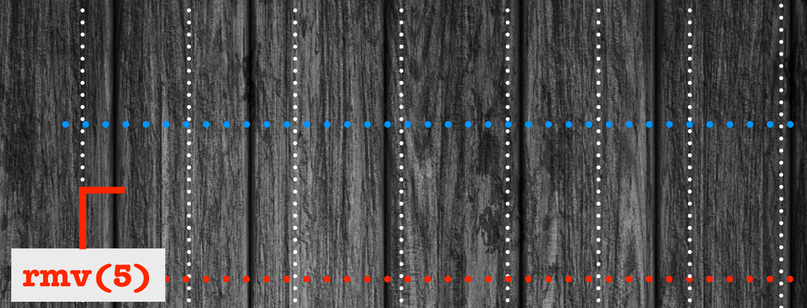
```
rmv(e):
local p, c, n, r
1  (p,c) := locate(e);
2  if (c.val = e)
3
4     n := c.next;
5     p.next := n;
6     r := true;
7  else r := false;
8  unlock(p);
9  unlock(c);
10 return r;
```



Scenario 1

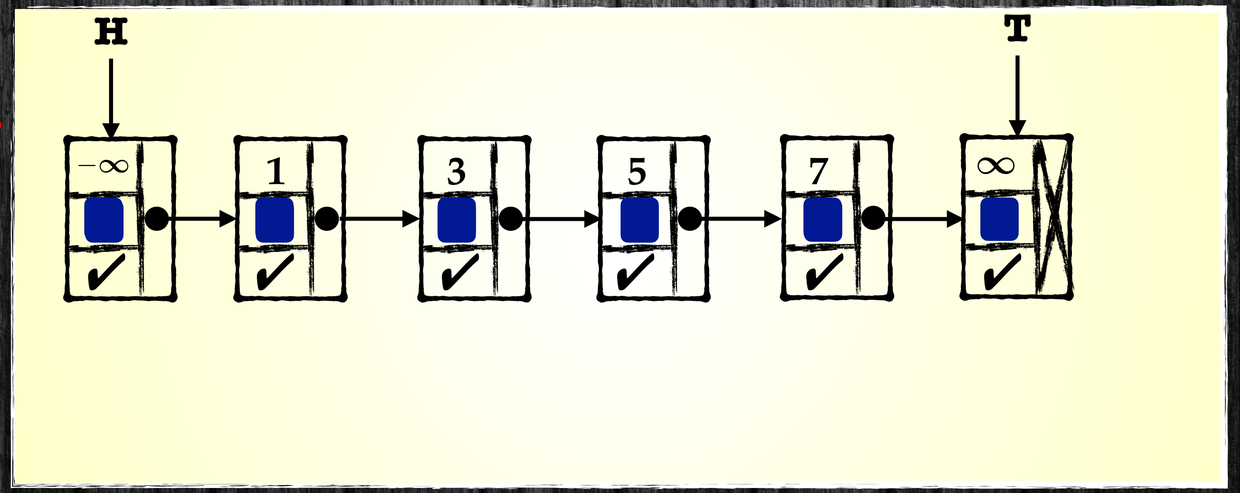
- no logical deletion
- only physical deletion





rmv(5)

```
rmv(e):
local p, c, n, r
1  (p,c) := locate(e);
2  if (c.val = e)
3
4     n := c.next;
5     p.next := n;
6     r := true;
7  else r := false;
8  unlock(p);
9  unlock(c);
10 return r;
```

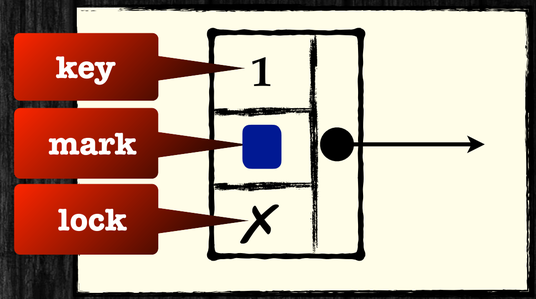


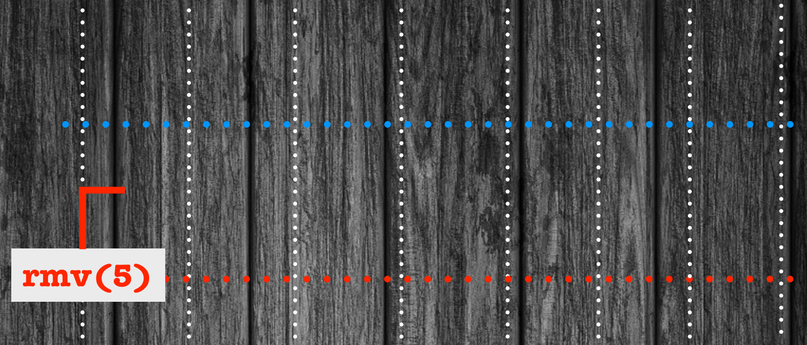
Sets | Lazy Synchronization

Scenario 1

- no logical deletion
- only physical deletion

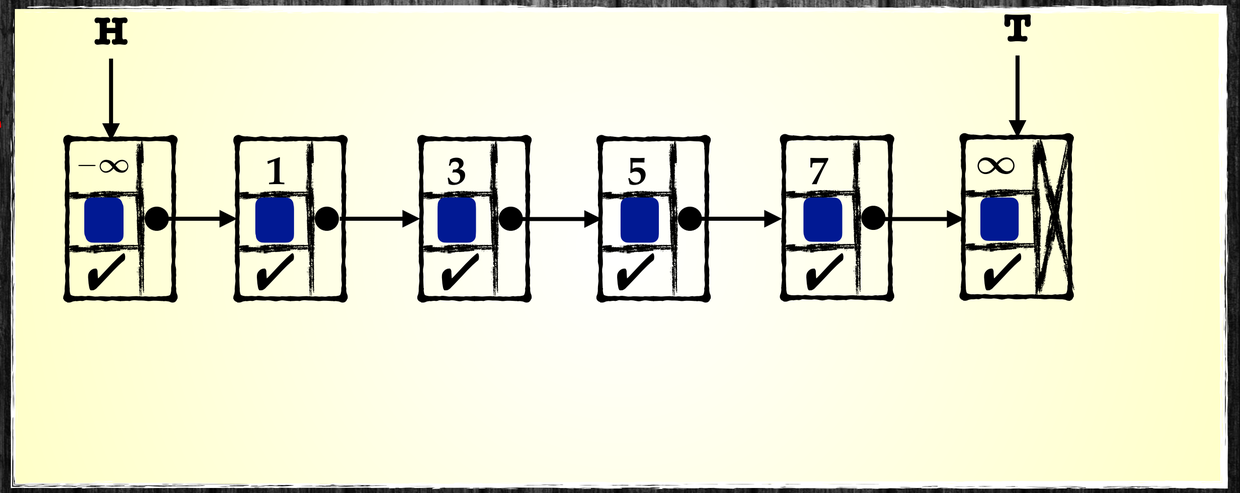
```
locate(e):
local p, c
1  while (true)
2     p := head;
3     c := p.next;
4     while (c.val < e)
5        p := c;
6        c := c.next
7     lock(p); lock(c);
8     if (!p.mark && !c.mark && p.next=c)
9        return(p,c);
10    else
11       unlock(p);
12       unlock(c);
```





rmv(5)

```
rmv(e):
local p, c, n, r
1  (p,c) := locate(e);
2  if (c.val = e)
3
4     n := c.next;
5     p.next := n;
6     r := true;
7  else r := false;
8  unlock(p);
9  unlock(c);
10 return r;
```



Sets | Lazy Synchronization

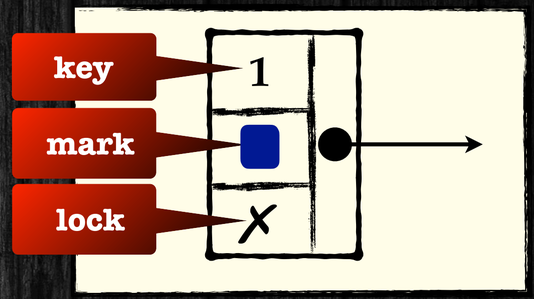
Scenario 1

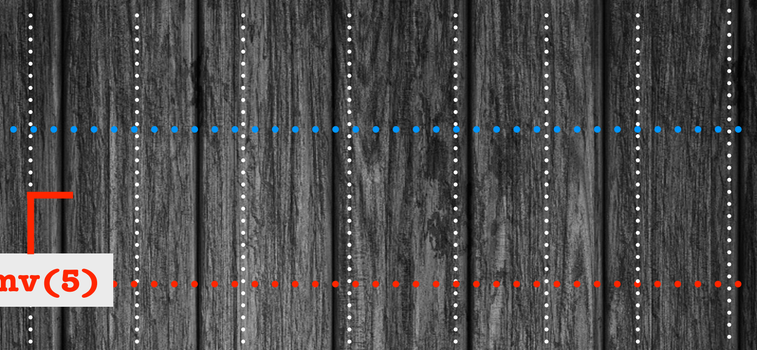
- no logical deletion
- only physical deletion

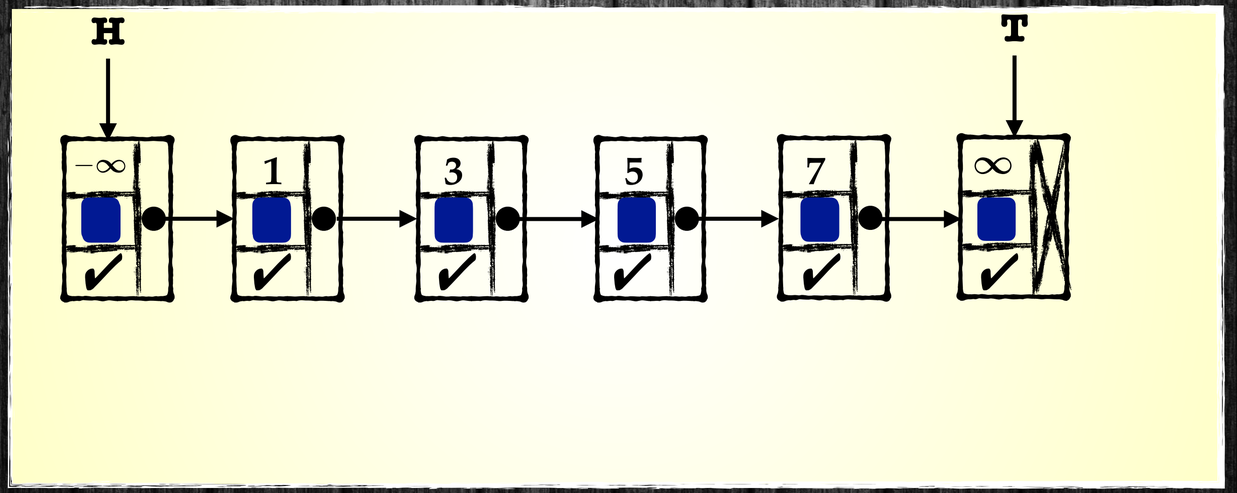
```
locate(e):
local p, c
1  while (true)
2     p := head;
3     c := p.next;
4     while (c.val < e)
5        p := c;
6        c := c.next
7     lock(p); lock(c);
8     if (!p.mark && !c.mark && p.next=c)
9        return(p,c);
10    else
11       unlock(p);
12       unlock(c);
```

rmv(5)

```
rmv(e):
local p, c, n, r
1  (p,c) := locate(e);
2  if (c.val = e)
3
4     n := c.next;
5     p.next := n;
6     r := true;
7  else r := false;
8  unlock(p);
9  unlock(c);
10 return r;
```





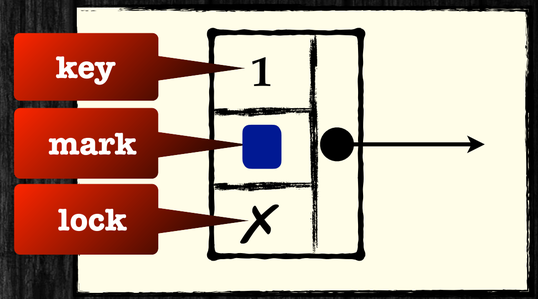


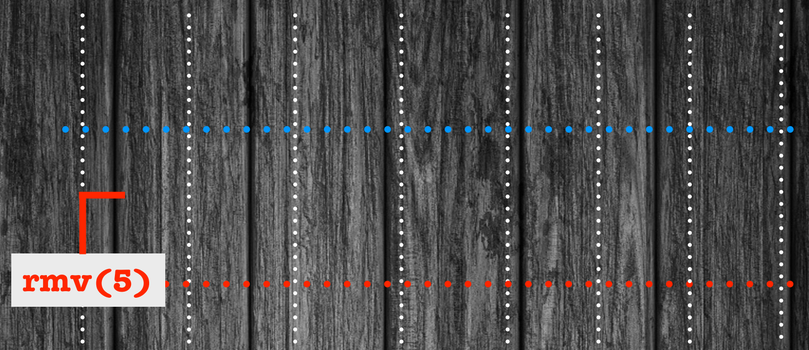
Sets | Lazy Synchronization

Scenario 1

- no logical deletion
- only physical deletion

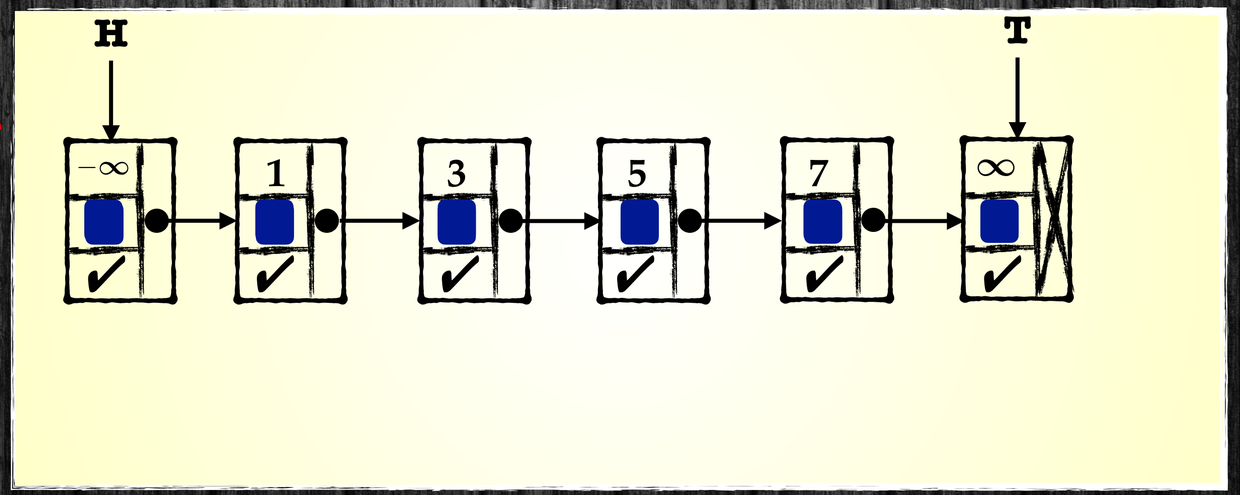
```
locate(e):
local p, c
 1  while (true)
 2     p := head;
 3     c := p.next;
 4     while (c.val < e)
 5        p := c;
 6        c := c.next
 7     lock(p); lock(c);
 8     if (                    p.next=c)
 9        return(p,c);
10     else
11        unlock(p);
12        unlock(c);
```





rmv(5)

```
rmv(e):
local p, c, n, r
 1  (p,c) := locate(e);
 2  if (c.val = e)
 3
 4     n := c.next;
 5     p.next := n;
 6     r := true;
 7  else r := false;
 8  unlock(p);
 9  unlock(c);
10  return r;
```

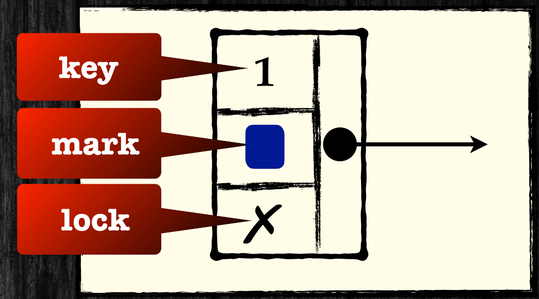


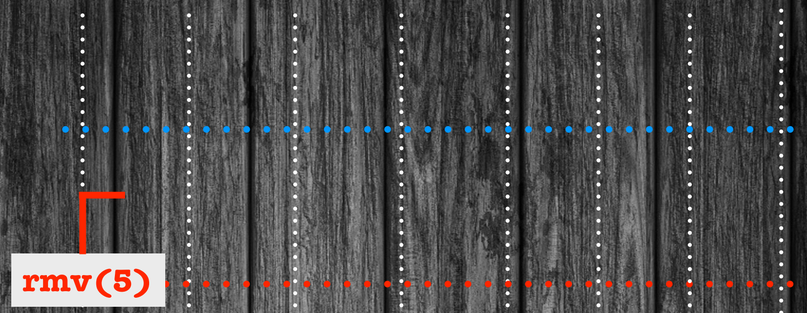
Sets | Lazy Synchronization

Scenario 1

- no logical deletion
- only physical deletion

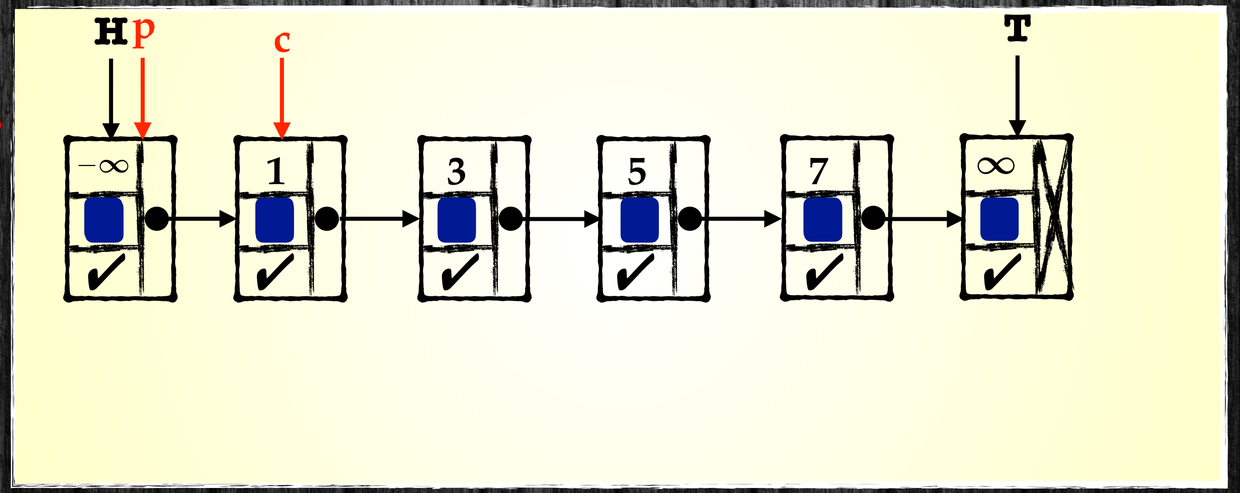
```
locate(e):
local p, c
1  while (true)
2     p := head;
3     c := p.next;
4     while (c.val < e)
5        p := c;
6        c := c.next
7     lock(p); lock(c);
8     if (                          p.next=c)
9        return(p,c);
10    else
11       unlock(p);
12       unlock(c);
```





rmv(5)

```
rmv(e):
local p, c, n, r
1  (p,c) := locate(e);
2  if (c.val = e)
3
4     n := c.next;
5     p.next := n;
6     r := true;
7  else r := false;
8  unlock(p);
9  unlock(c);
10 return r;
```

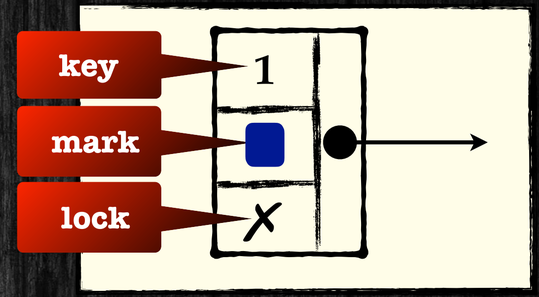


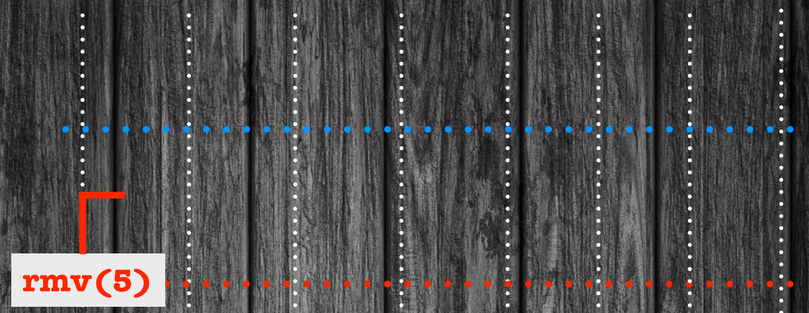
Sets | Lazy Synchronization

Scenario 1

- no logical deletion
- only physical deletion

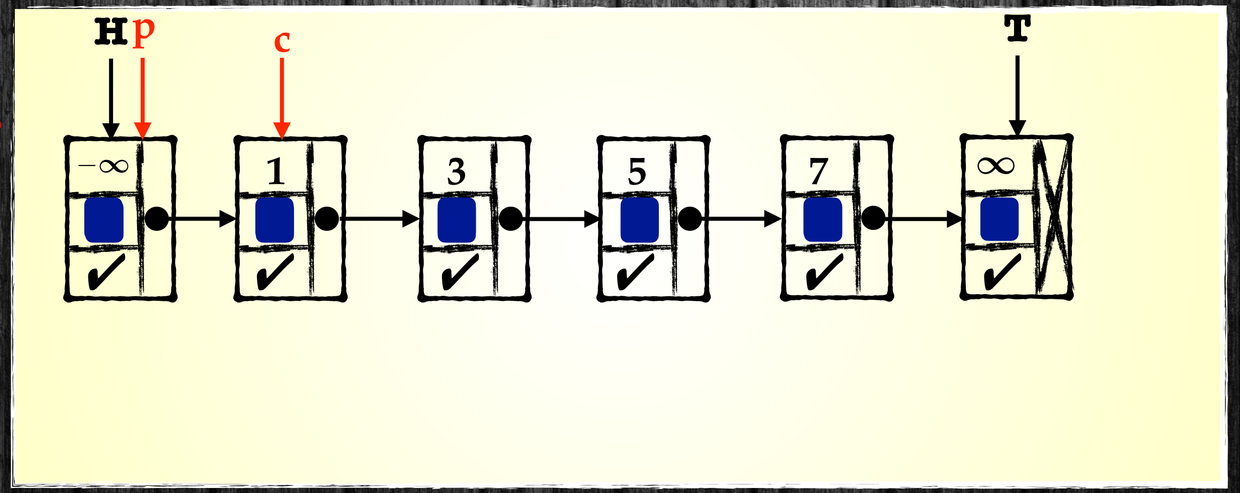
```
locate(e):
local p, c
1  while (true)
2     p := head;
3     c := p.next;
4     while (c.val < e)
5        p := c;
6        c := c.next
7     lock(p); lock(c);
8     if (                          p.next=c)
9        return(p,c);
10    else
11       unlock(p);
12       unlock(c);
```





rmv(5)

```
rmv(e):
local p, c, n, r
1  (p,c) := locate(e);
2  if (c.val = e)
3
4     n := c.next;
5     p.next := n;
6     r := true;
7  else r := false;
8  unlock(p);
9  unlock(c);
10 return r;
```

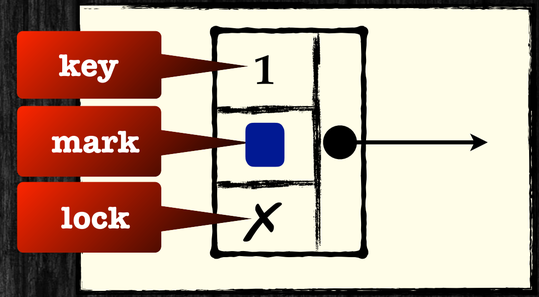


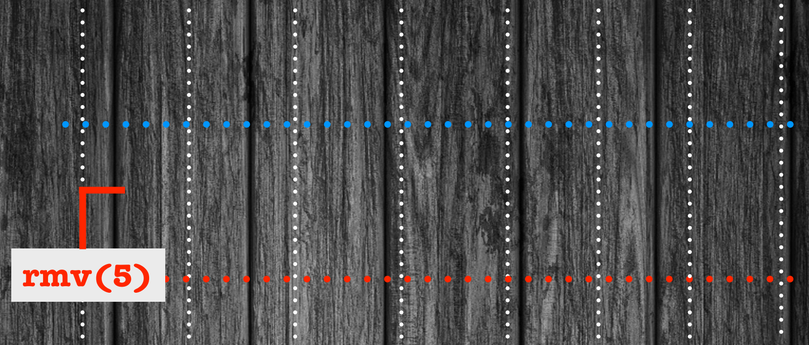
Sets | Lazy Synchronization

Scenario 1

- no logical deletion
- only physical deletion

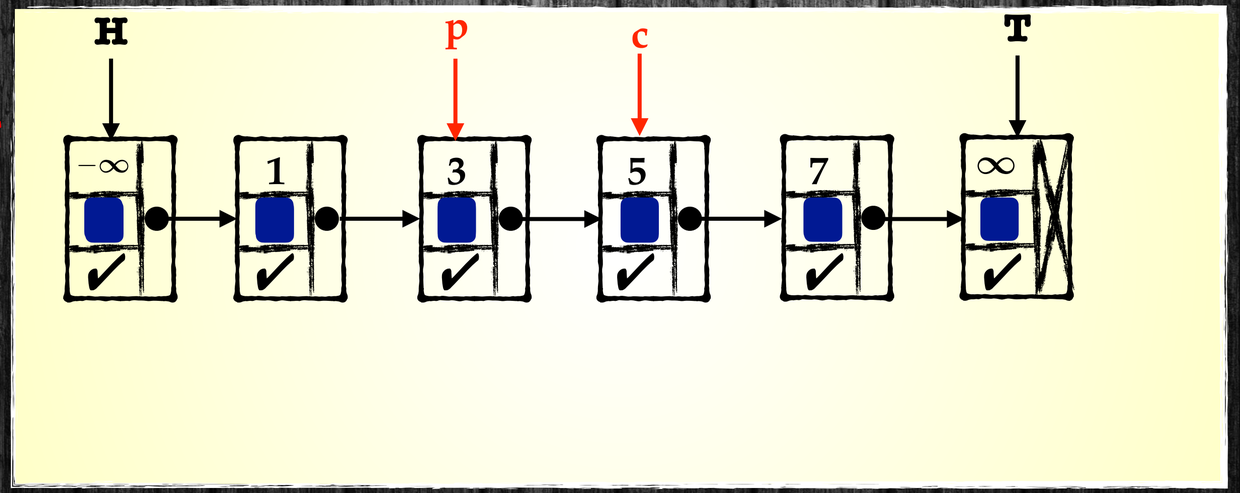
```
locate(e):
local p, c
1  while (true)
2     p := head;
3     c := p.next;
4     while (c.val < e)
5        p := c;
6        c := c.next
7     lock(p); lock(c);
8     if (                          p.next=c)
9        return(p,c);
10    else
11       unlock(p);
12       unlock(c);
```





rmv(5)

```
rmv(e):
local p, c, n, r
1  (p,c) := locate(e);
2  if (c.val = e)
3
4     n := c.next;
5     p.next := n;
6     r := true;
7  else r := false;
8  unlock(p);
9  unlock(c);
10 return r;
```



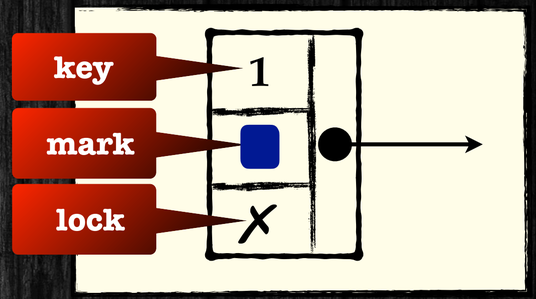
Sets | Lazy Synchronization

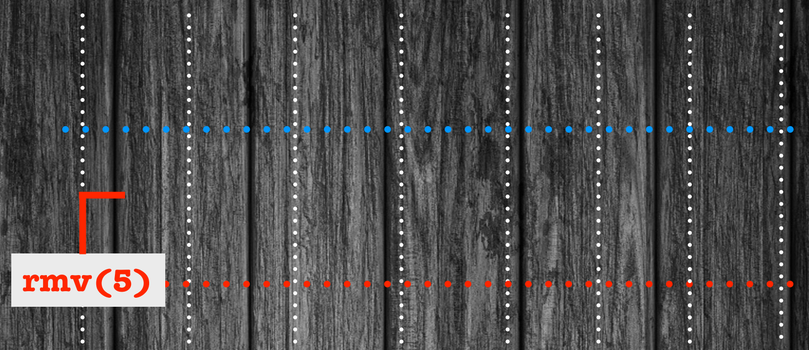
Scenario 1

- no logical deletion
- only physical deletion

```
locate(e):
local p, c
1  while (true)
2     p := head;
3     c := p.next;
4     while (c.val < e)
5        p := c;
6        c := c.next
7     lock(p); lock(c);
8     if (                          p.next=c)
9        return(p,c);
10    else
11       unlock(p);
12       unlock(c);
```

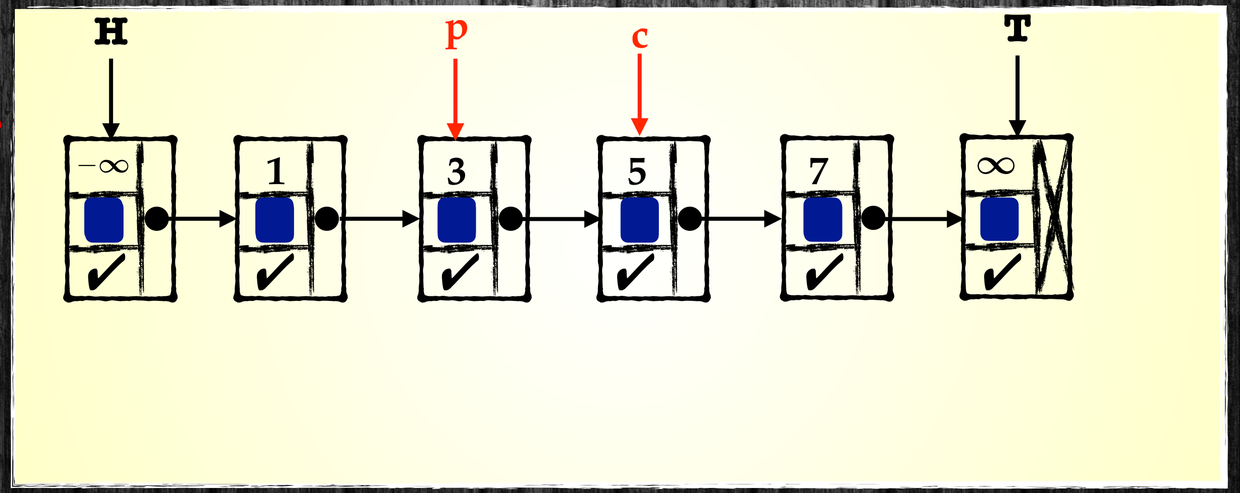
go to sleep





rmv(5)

```
rmv(e):
local p, c, n, r
1  (p,c) := locate(e);
2  if (c.val = e)
3
4     n := c.next;
5     p.next := n;
6     r := true;
7  else r := false;
8  unlock(p);
9  unlock(c);
10 return r;
```



Sets | Lazy Synchronization

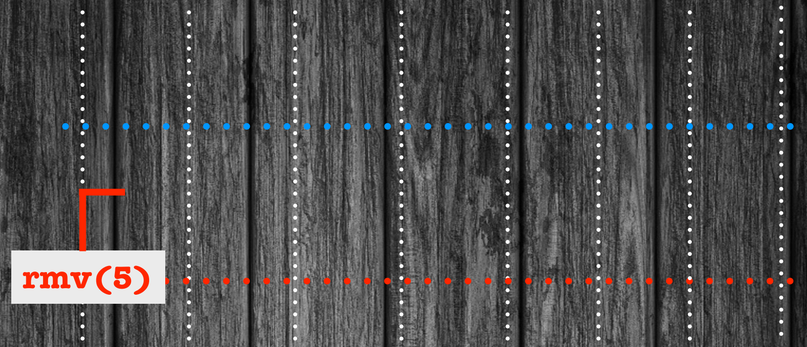
```
locate(e):
local p, c
1  while (true)
2    p := head;
3    c := p.next;
4    while (c.val < e)
5      p := c;
6      c := c.next
7    lock(p); lock(c);
8    if (!p.mark && !c.mark && p.next=c)
9      return(p,c);
10   else
11     unlock(p);
12     unlock(c);
```

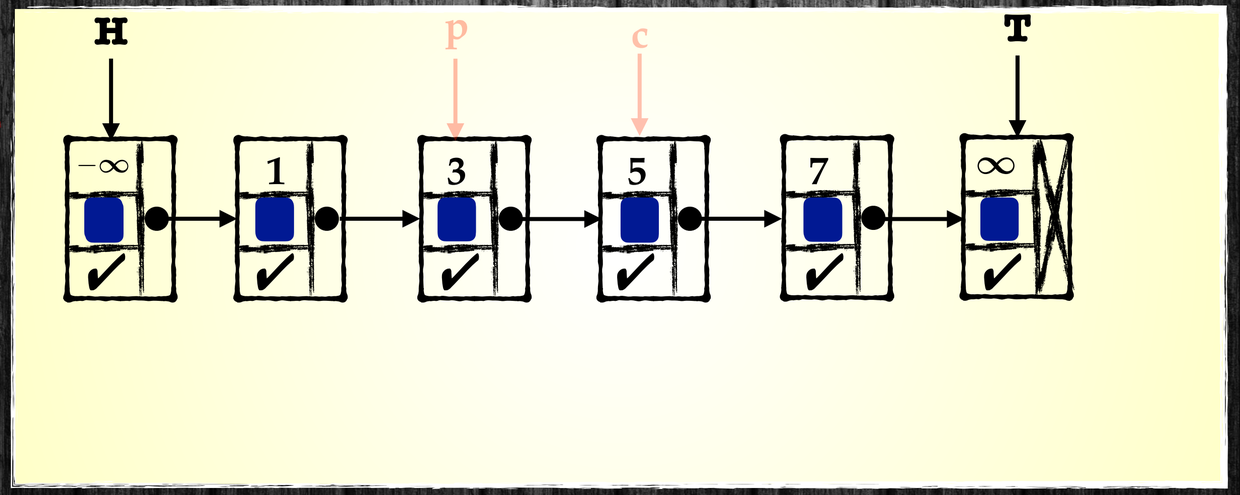
```
rmv(e):
local p, c, n, r
1  (p,c) := locate(e);
2  if (c.val = e)
3    c.mark := true
4    n := c.next;
5    p.next := n;
6    r := true;
7  else r := false;
8  unlock(p);
9  unlock(c);
10 return r;
```

rmv(5)

Scenario 1

- no logical deletion
- only physical deletion





Sets | Lazy Synchronization

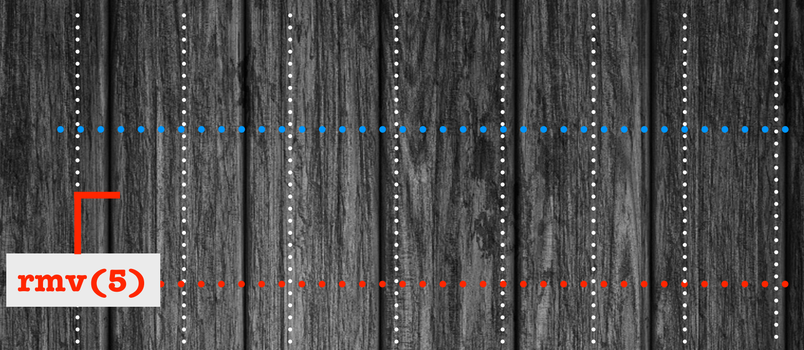
Scenario 1

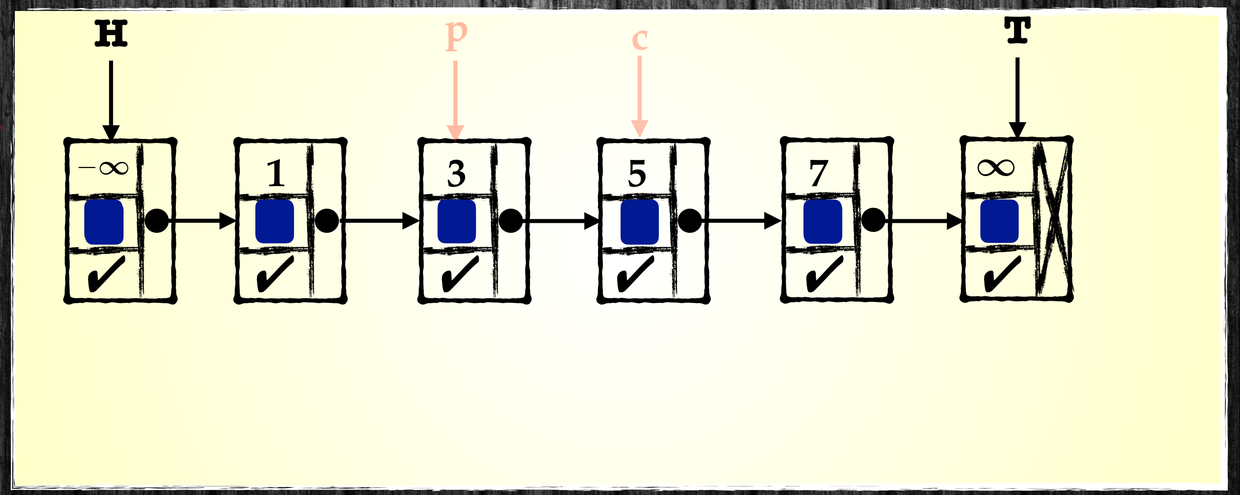
- no logical deletion
- only physical deletion

```
locate(e):
local p, c
1  while (true)
2     p := head;
3     c := p.next;
4     while (c.val < e)
5        p := c;
6        c := c.next
7     lock(p); lock(c);
8     if (!p.mark && !c.mark && p.next=c)
9        return(p,c);
10    else
11       unlock(p);
12       unlock(c);
```

rmv(5)

```
rmv(e):
local p, c, n, r
1  (p,c) := locate(e);
2  if (c.val = e)
3     c.mark := true
4     n := c.next;
5     p.next := n;
6     r := true;
7  else r := false;
8  unlock(p);
9  unlock(c);
10 return r;
```





rmv(3)

```
rmv(e):
local p, c, n, r
1  (p,c) := locate(e);
2  if (c.val = e)
3     c.mark := true
4     n := c.next;
5     p.next := n;
6     r := true;
7  else r := false;
8  unlock(p);
9  unlock(c);
10 return r;
```

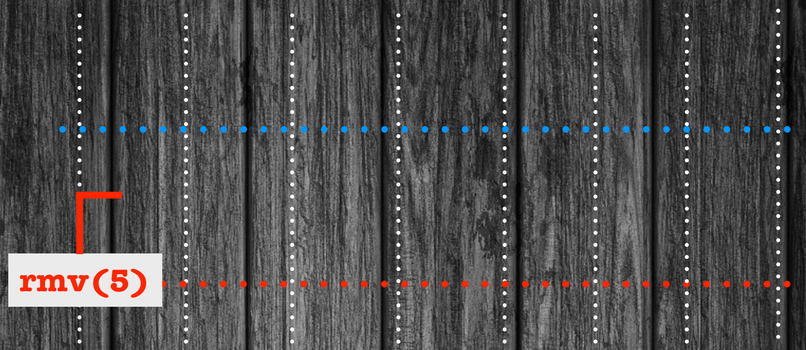
Scenario 1

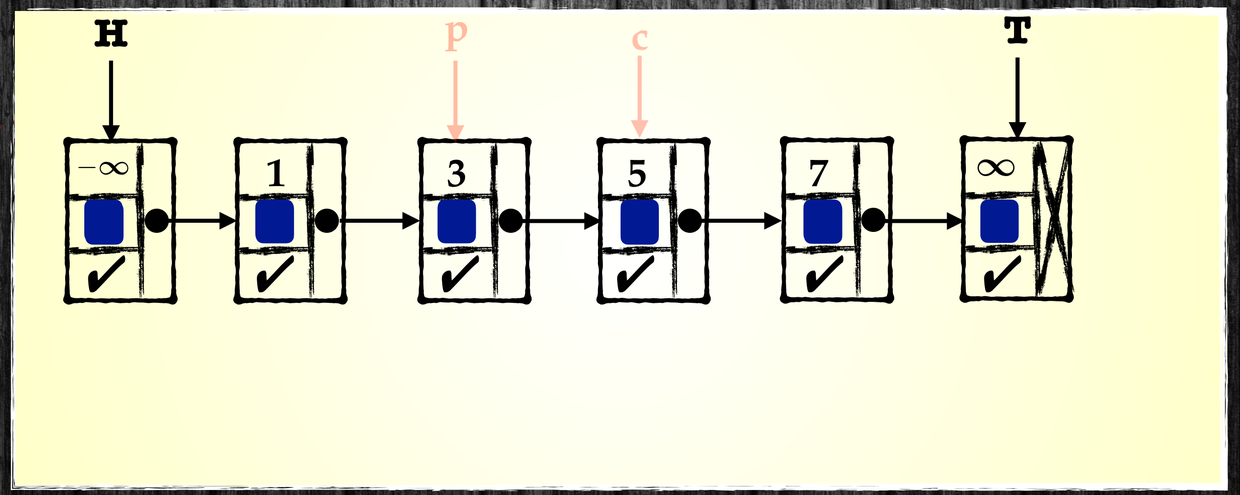
- no logical deletion
- only physical deletion

```
locate(e):
local p, c
1  while (true)
2    p := head;
3    c := p.next;
4    while (c.val < e)
5      p := c;
6      c := c.next
7    lock(p); lock(c);
8    if (!p.mark && !c.mark && p.next=c)
9      return(p,c);
10   else
11     unlock(p);
12     unlock(c);
```

rmv(5)

```
rmv(e):
local p, c, n, r
1  (p,c) := locate(e);
2  if (c.val = e)
3    c.mark := true
4    n := c.next;
5    p.next := n;
6    r := true;
7  else r := false;
8  unlock(p);
9  unlock(c);
10 return r;
```





rmv(3)

```
rmv(e):
local p, c, n, r
1  (p,c) := locate(e);
2  if (c.val = e)
3
4    n := c.next;
5    p.next := n;
6    r := true;
7  else r := false;
8  unlock(p);
9  unlock(c);
10 return r;
```

Sets | Lazy Synchronization

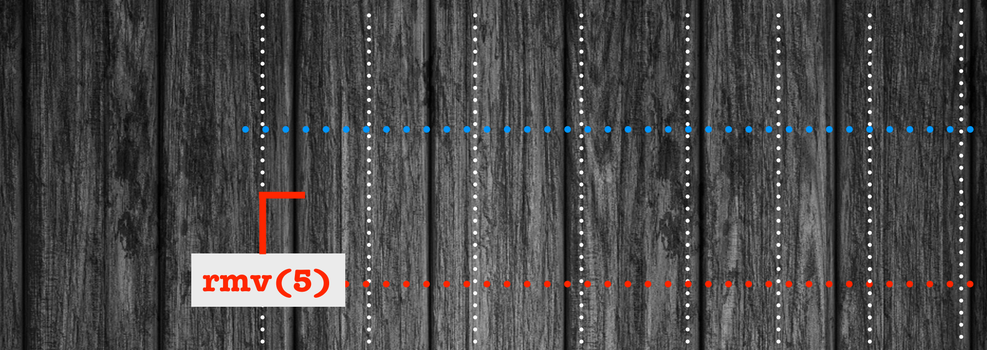
Scenario 1

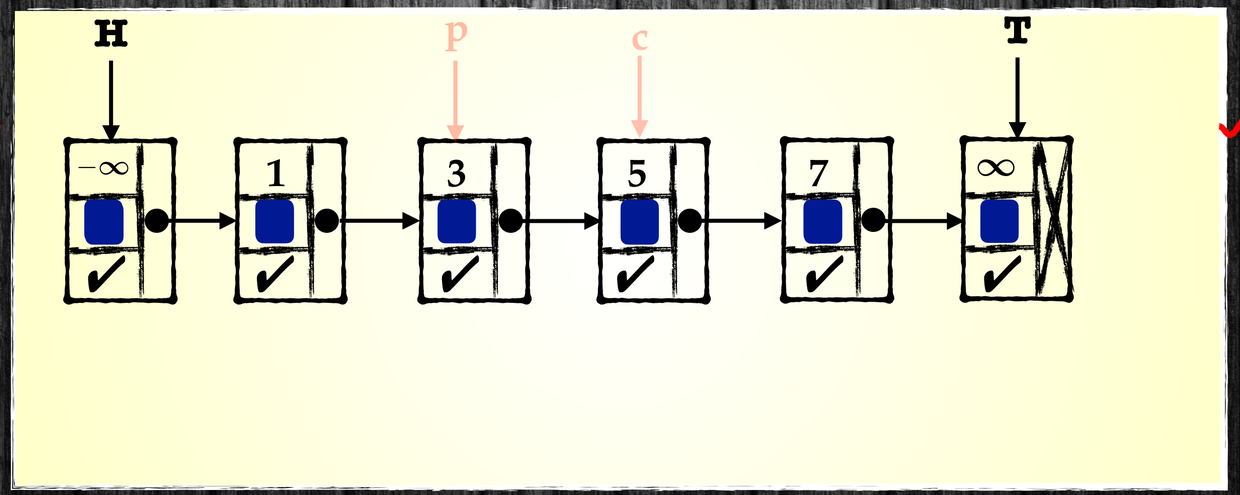
- no logical deletion
- only physical deletion

```
locate(e):
local p, c
1  while (true)
2     p := head;
3     c := p.next;
4     while (c.val < e)
5        p := c;
6        c := c.next
7     lock(p); lock(c);
8     if (!p.mark && !c.mark && p.next=c)
9        return(p,c);
10    else
11       unlock(p);
12       unlock(c);
```

rmv(5)

```
rmv(e):
local p, c, n, r
1  (p,c) := locate(e);
2  if (c.val = e)
3     c.mark := true
4     n := c.next;
5     p.next := n;
6     r := true;
7  else r := false;
8  unlock(p);
9  unlock(c);
10 return r;
```





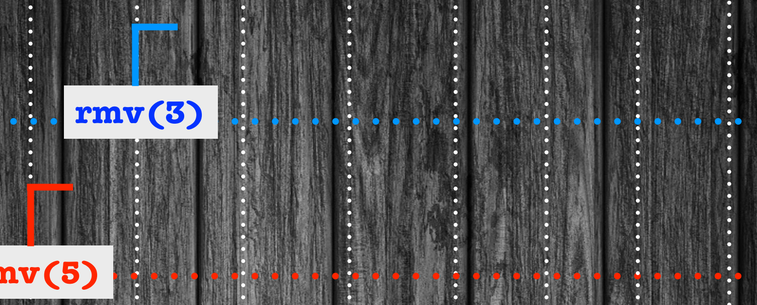
rmv(3)

```
rmv(e):
local p, c, n, r
1  (p,c) := locate(e);
2  if (c.val = e)
3
4     n := c.next;
5     p.next := n;
6     r := true;
7  else r := false;
8  unlock(p);
9  unlock(c);
10 return r;
```

Scenario 1

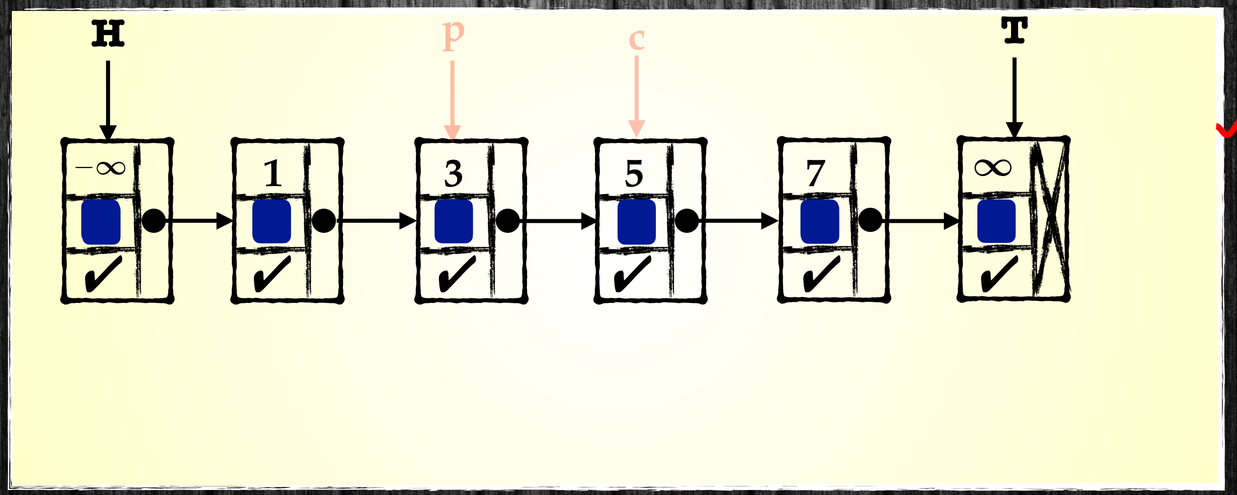
- no logical deletion
- only physical deletion

```
locate(e):
local p, c
1  while (true)
2     p := head;
3     c := p.next;
4     while (c.val < e)
5        p := c;
6        c := c.next
7     lock(p); lock(c);
8     if (!p.mark && !c.mark && p.next=c)
9        return(p,c);
10    else
11       unlock(p);
12       unlock(c);
```



rmv(5)

```
rmv(e):
local p, c, n, r
1  (p,c) := locate(e);
2  if (c.val = e)
3     c.mark := true
4     n := c.next;
5     p.next := n;
6     r := true;
7  else r := false;
8  unlock(p);
9  unlock(c);
10 return r;
```



rmv(3)

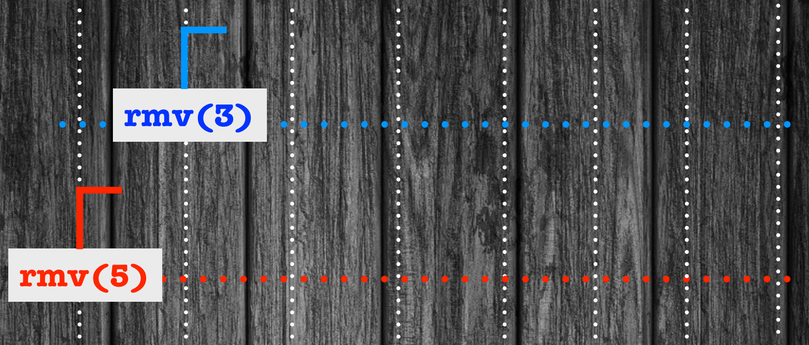
```
rmv(e):
local p, c, n, r
1  (p,c) := locate(e);
2  if (c.val = e)
3
4     n := c.next;
5     p.next := n;
6     r := true;
7  else r := false;
8  unlock(p);
9  unlock(c);
10 return r;
```

Sets | Lazy Synchronization

Scenario 1

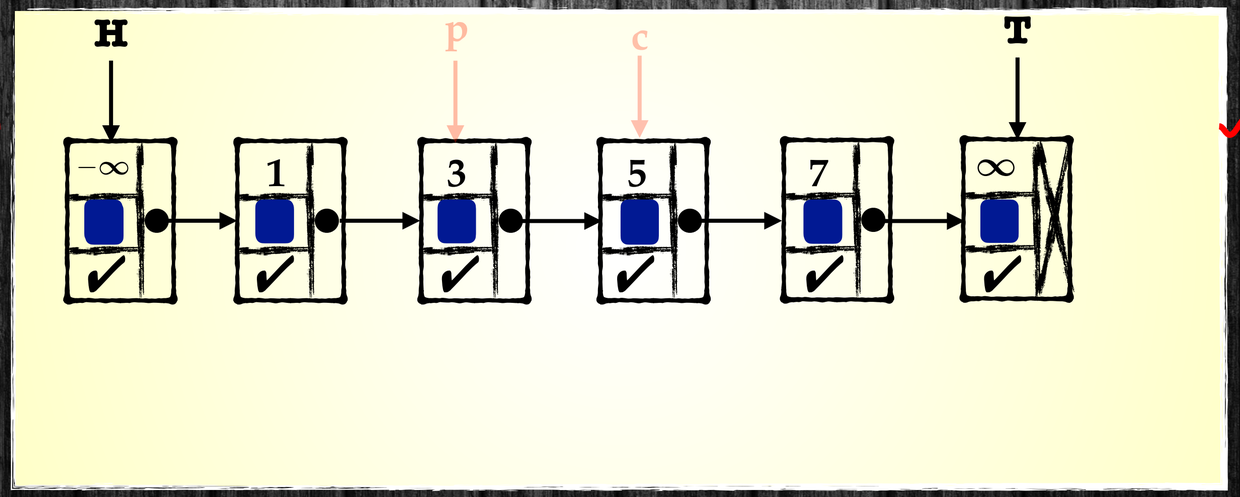
- no logical deletion
- only physical deletion

```
locate(e):
local p, c
1  while (true)
2     p := head;
3     c := p.next;
4     while (c.val < e)
5        p := c;
6        c := c.next
7     lock(p); lock(c);
8     if (!p.mark && !c.mark && p.next=c)
9        return(p,c);
10    else
11       unlock(p);
12       unlock(c);
```



rmv(5)

```
rmv(e):
local p, c, n, r
1  (p,c) := locate(e);
2  if (c.val = e)
3     c.mark := true
4     n := c.next;
5     p.next := n;
6     r := true;
7  else r := false;
8  unlock(p);
9  unlock(c);
10 return r;
```



rmv(3)

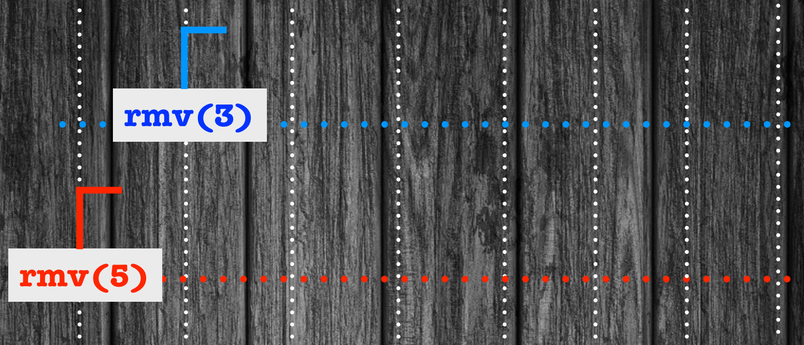
```
rmv(e):
local p, c, n, r
1  (p,c) := locate(e);
2  if (c.val = e)
3
4     n := c.next;
5     p.next := n;
6     r := true;
7  else r := false;
8  unlock(p);
9  unlock(c);
10 return r;
```

Sets | Lazy Synchronization

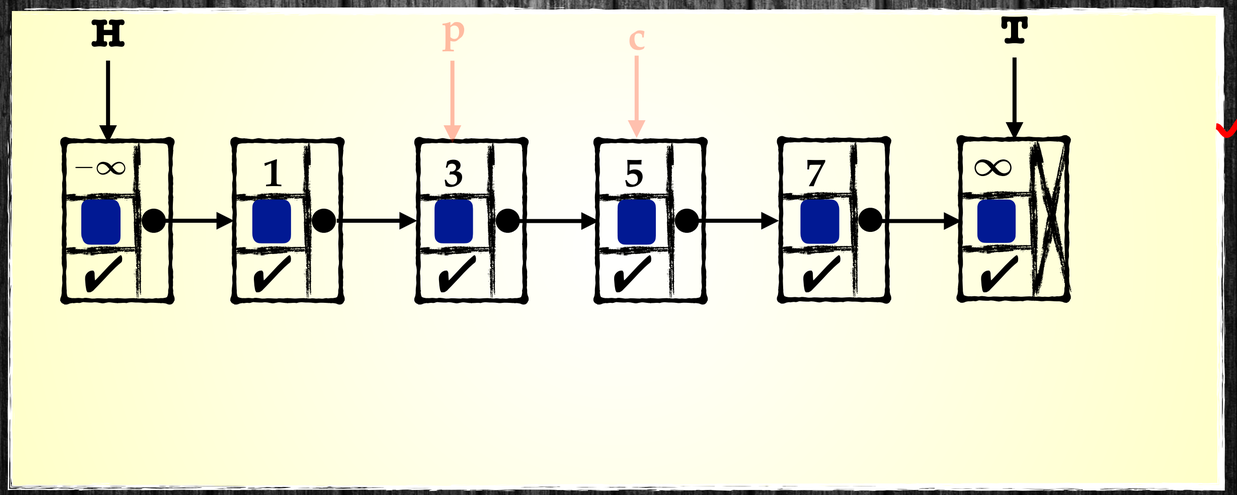
Scenario 1

- no logical deletion
- only physical deletion

```
locate(e):
local p, c
1  while (true)
2     p := head;
3     c := p.next;
4     while (c.val < e)
5        p := c;
6        c := c.next
7     lock(p); lock(c);
8     if (!p.mark && !c.mark && p.next=c)
9        return(p,c);
10    else
11       unlock(p);
12       unlock(c);
```




```
rmv(e):                    rmv(5)
local p, c, n, r
1  (p,c) := locate(e);
2  if (c.val = e)
3     c.mark := true
4     n := c.next;
5     p.next := n;
6     r := true;
7  else r := false;
8  unlock(p);
9  unlock(c);
10 return r;
```



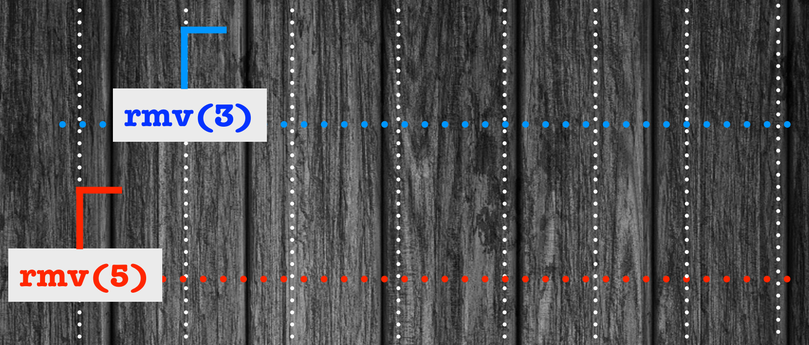

```
rmv(e):                    rmv(3)
local p, c, n, r
1  (p,c) := locate(e);
2  if (c.val = e)
3
4     n := c.next;
5     p.next := n;
6     r := true;
7  else r := false;
8  unlock(p);
9  unlock(c);
10 return r;
```

Sets | Lazy Synchronization

Scenario 1

- no logical deletion
- only physical deletion

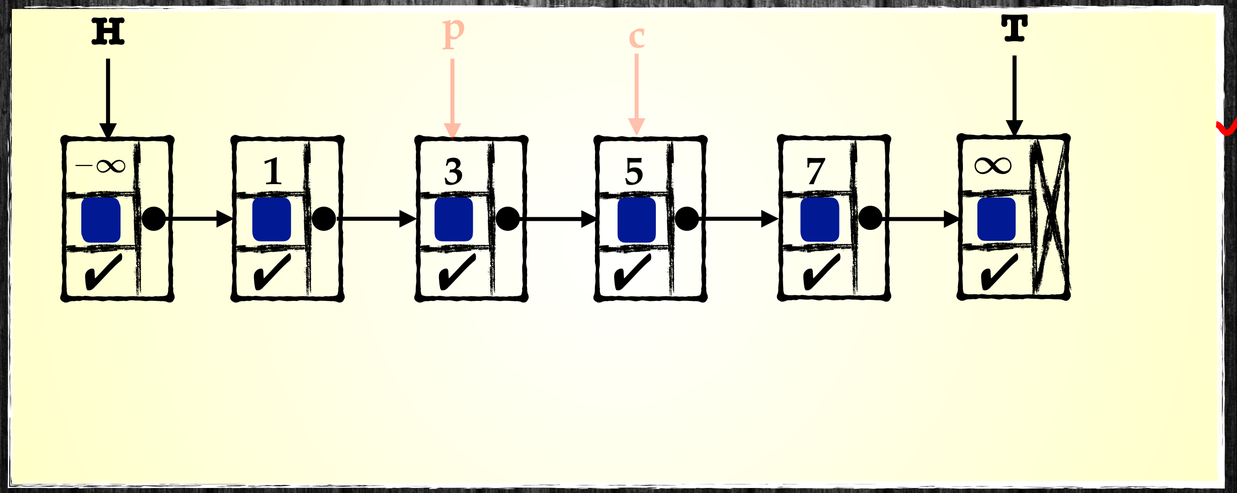
```
locate(e):
local p, c
1  while (true)
2     p := head;
3     c := p.next;
4     while (c.val < e)
5        p := c;
6        c := c.next
7     lock(p); lock(c);
8     if (!p.mark && !c.mark && p.next=c)
9        return(p,c);
10    else
11       unlock(p);
12       unlock(c);
```




```
locate(e):
local p, c
1  while (true)
2     p := head;
3     c := p.next;
4     while (c.val < e)
5        p := c;
6        c := c.next
7     lock(p); lock(c);
8     if (                    p.next=c)
9        return(p,c);
10    else
11       unlock(p);
12       unlock(c);
```

rmv(5)

```
rmv(e):
local p, c, n, r
1  (p,c) := locate(e);
2  if (c.val = e)
3     c.mark := true
4     n := c.next;
5     p.next := n;
6     r := true;
7  else r := false;
8  unlock(p);
9  unlock(c);
10 return r;
```



rmv(3)

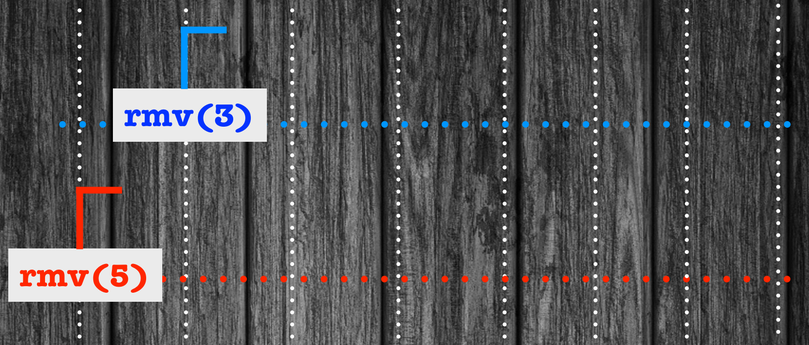
```
rmv(e):
local p, c, n, r
1  (p,c) := locate(e);
2  if (c.val = e)
3
4     n := c.next;
5     p.next := n;
6     r := true;
7  else r := false;
8  unlock(p);
9  unlock(c);
10 return r;
```

Sets | Lazy Synchronization

Scenario 1

- no logical deletion
- only physical deletion

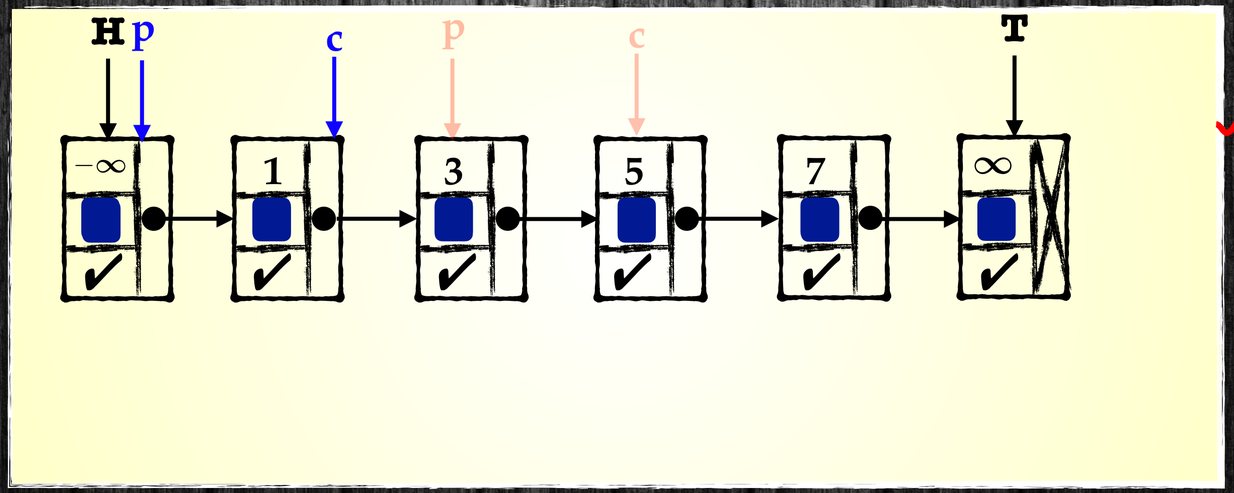
```
locate(e):
local p, c
1  while (true)
2     p := head;
3     c := p.next;
4     while (c.val < e)
5        p := c;
6        c := c.next
7     lock(p); lock(c);
8     if (!p.mark && !c.mark && p.next=c)
9        return(p,c);
10    else
11       unlock(p);
12       unlock(c);
```




```
locate(e):
local p, c
1  while (true)
2     p := head;
3     c := p.next;
4     while (c.val < e)
5        p := c;
6        c := c.next
7     lock(p); lock(c);
8     if (                    p.next=c)
9        return(p,c);
10    else
11       unlock(p);
12       unlock(c);
```

rmv(5)

```
rmv(e):
local p, c, n, r
1  (p,c) := locate(e);
2  if (c.val = e)
3     c.mark := true
4     n := c.next;
5     p.next := n;
6     r := true;
7  else r := false;
8  unlock(p);
9  unlock(c);
10 return r;
```



rmv(3)

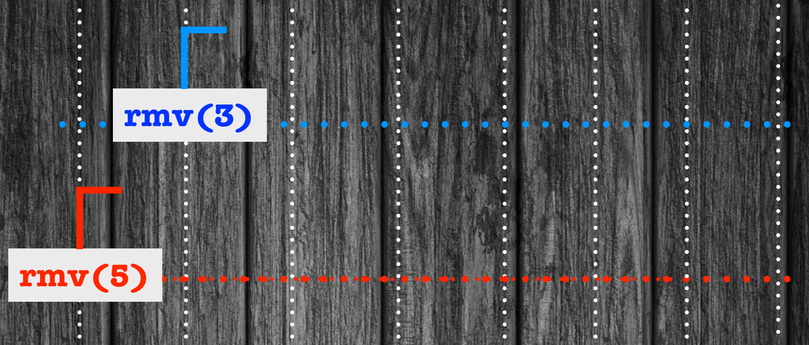
```
rmv(e):
local p, c, n, r
1  (p,c) := locate(e);
2  if (c.val = e)
3
4     n := c.next;
5     p.next := n;
6     r := true;
7  else r := false;
8  unlock(p);
9  unlock(c);
10 return r;
```

Sets | Lazy Synchronization

Scenario 1

- no logical deletion
- only physical deletion

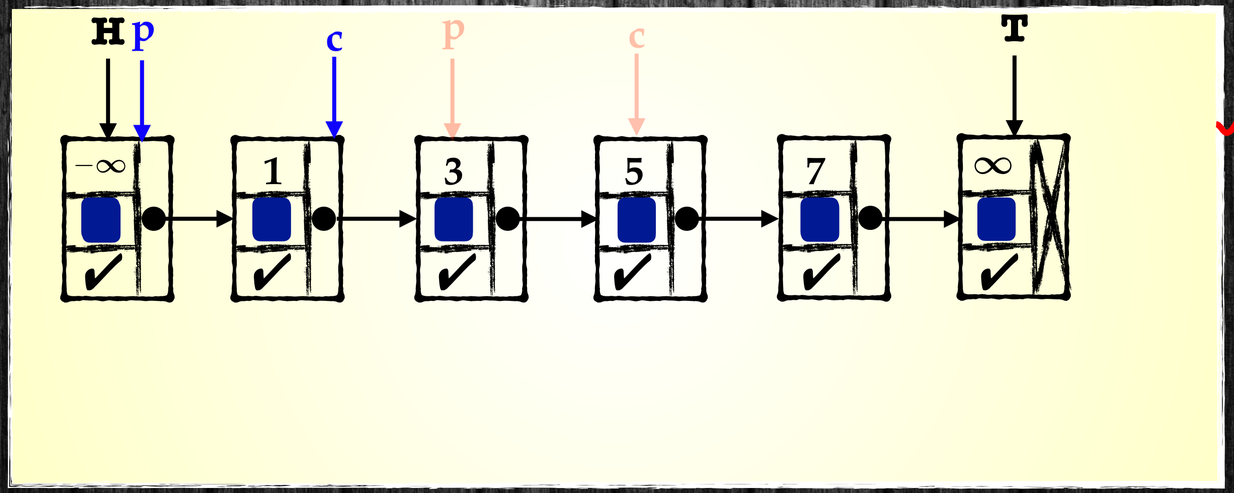
```
locate(e):
local p, c
1  while (true)
2     p := head;
3     c := p.next;
4     while (c.val < e)
5        p := c;
6        c := c.next
7     lock(p); lock(c);
8     if (!p.mark && !c.mark && p.next=c)
9        return(p,c);
10    else
11       unlock(p);
12       unlock(c);
```




```
locate(e):
local p, c
1  while (true)
2     p := head;
3     c := p.next;
4     while (c.val < e)
5        p := c;
6        c := c.next
7     lock(p); lock(c);
8     if (                    p.next=c)
9        return(p,c);
10    else
11       unlock(p);
12       unlock(c);
```

rmv(5)

```
rmv(e):
local p, c, n, r
1  (p,c) := locate(e);
2  if (c.val = e)
3     c.mark := true
4     n := c.next;
5     p.next := n;
6     r := true;
7  else r := false;
8  unlock(p);
9  unlock(c);
10 return r;
```



rmv(3)

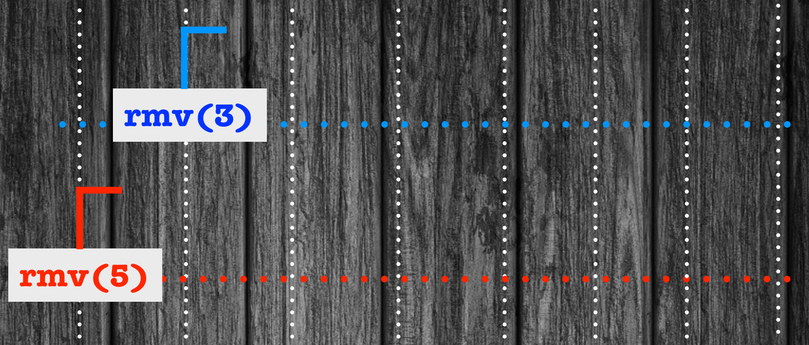
```
rmv(e):
local p, c, n, r
1  (p,c) := locate(e);
2  if (c.val = e)
3
4     n := c.next;
5     p.next := n;
6     r := true;
7  else r := false;
8  unlock(p);
9  unlock(c);
10 return r;
```

Sets | Lazy Synchronization

Scenario 1

- no logical deletion
- only physical deletion

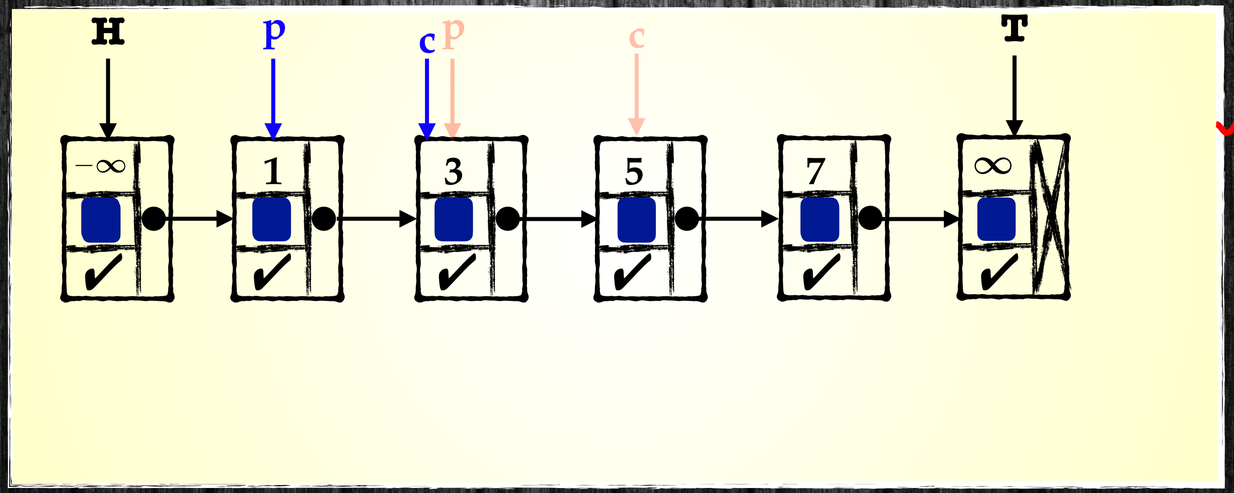
```
locate(e):
local p, c
1  while (true)
2     p := head;
3     c := p.next;
4     while (c.val < e)
5        p := c;
6        c := c.next
7     lock(p); lock(c);
8     if (!p.mark && !c.mark && p.next=c)
9        return(p,c);
10    else
11       unlock(p);
12       unlock(c);
```




```
locate(e):
local p, c
1  while (true)
2     p := head;
3     c := p.next;
4     while (c.val < e)
5        p := c;
6        c := c.next
7     lock(p); lock(c);
8     if (                    p.next=c)
9        return(p,c);
10    else
11       unlock(p);
12       unlock(c);
```

rmv(5)

```
rmv(e):
local p, c, n, r
1  (p,c) := locate(e);
2  if (c.val = e)
3     c.mark := true
4     n := c.next;
5     p.next := n;
6     r := true;
7  else r := false;
8  unlock(p);
9  unlock(c);
10 return r;
```



rmv(3)

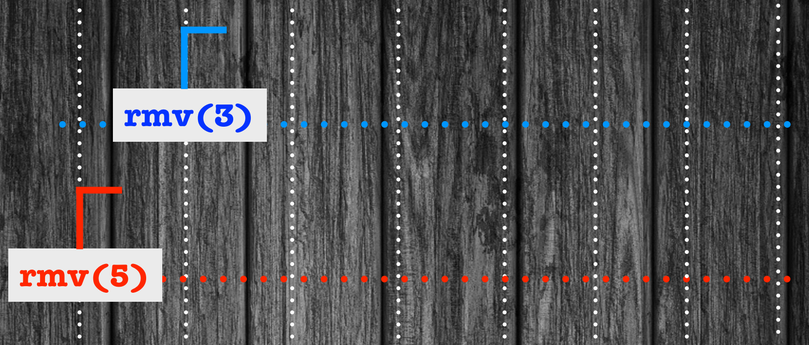
```
rmv(e):
local p, c, n, r
1  (p,c) := locate(e);
2  if (c.val = e)
3
4     n := c.next;
5     p.next := n;
6     r := true;
7  else r := false;
8  unlock(p);
9  unlock(c);
10 return r;
```

Sets | Lazy Synchronization

Scenario 1

- no logical deletion
- only physical deletion

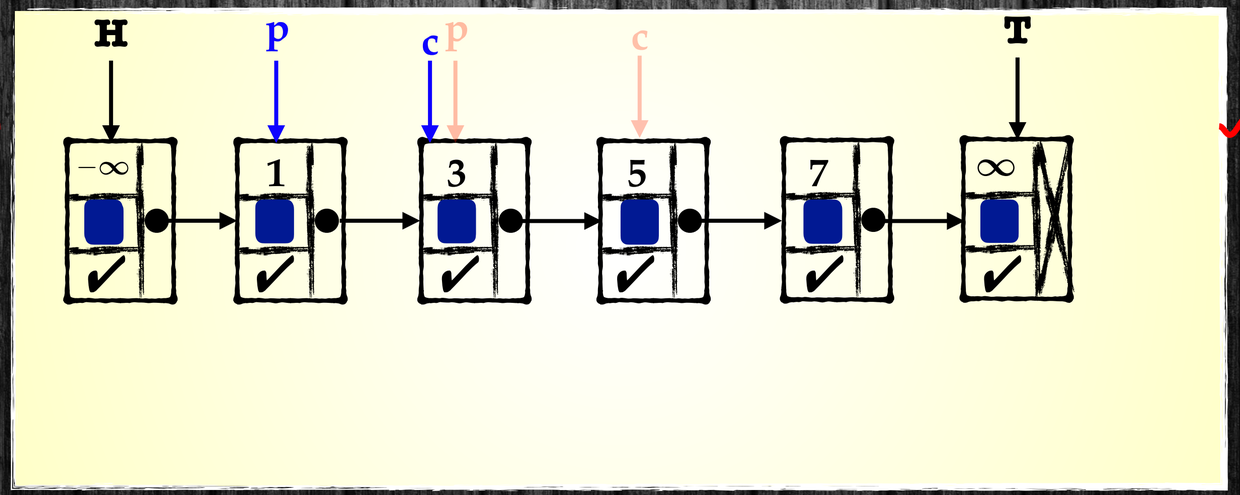
```
locate(e):
local p, c
1  while (true)
2     p := head;
3     c := p.next;
4     while (c.val < e)
5        p := c;
6        c := c.next
7     lock(p); lock(c);
8     if (!p.mark && !c.mark && p.next=c)
9        return(p,c);
10    else
11       unlock(p);
12       unlock(c);
```




```
locate(e):
local p, c
1  while (true)
2     p := head;
3     c := p.next;
4     while (c.val < e)
5        p := c;
6        c := c.next
7     lock(p); lock(c);
8     if (                              p.next=c)
9        return(p,c);
10    else
11       unlock(p);
12       unlock(c);
```

rmv(5)

```
rmv(e):
local p, c, n, r
1  (p,c) := locate(e);
2  if (c.val = e)
3     c.mark := true
4     n := c.next;
5     p.next := n;
6     r := true;
7  else r := false;
8  unlock(p);
9  unlock(c);
10 return r;
```



rmv(3)

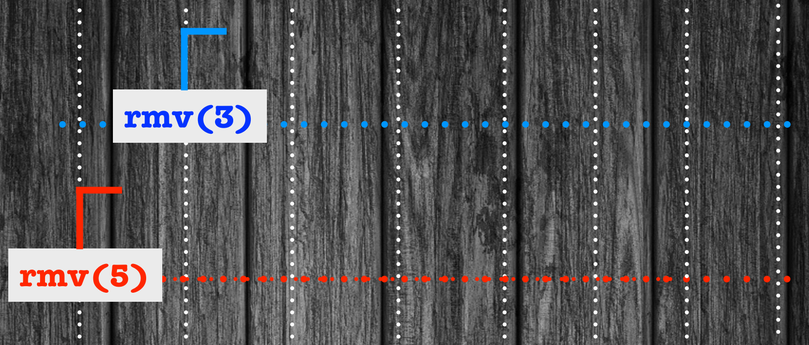
```
rmv(e):
local p, c, n, r
1  (p,c) := locate(e);
2  if (c.val = e)
3
4     n := c.next;
5     p.next := n;
6     r := true;
7  else r := false;
8  unlock(p);
9  unlock(c);
10 return r;
```

Sets | Lazy Synchronization

Scenario 1

- no logical deletion
- only physical deletion

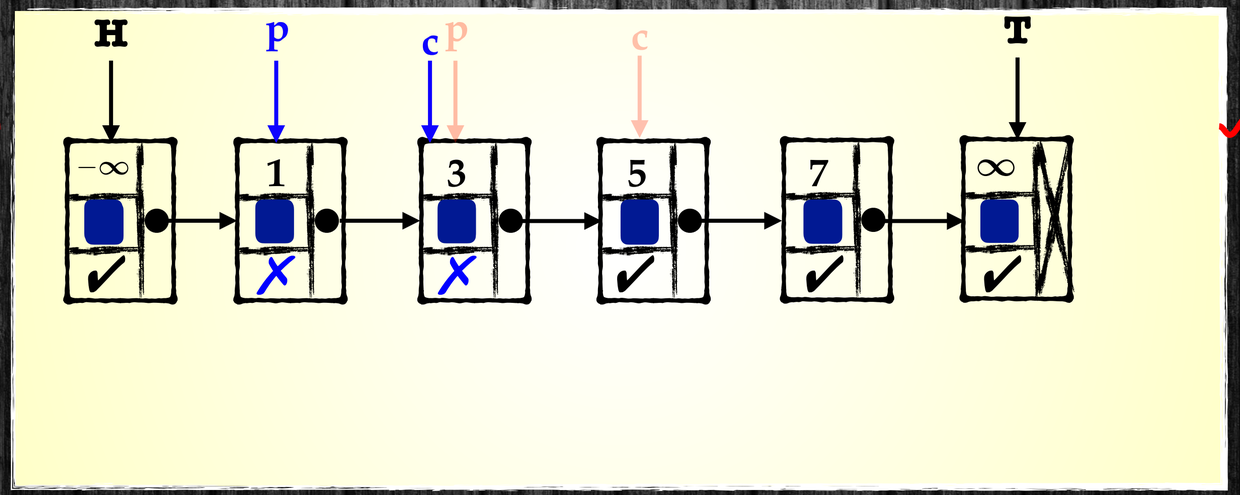
```
locate(e):
local p, c
1  while (true)
2     p := head;
3     c := p.next;
4     while (c.val < e)
5        p := c;
6        c := c.next
7     lock(p); lock(c);
8     if (!p.mark && !c.mark && p.next=c)
9        return(p,c);
10    else
11       unlock(p);
12       unlock(c);
```




```
locate(e):
local p, c
1  while (true)
2     p := head;
3     c := p.next;
4     while (c.val < e)
5        p := c;
6        c := c.next
7     lock(p); lock(c);
8     if (                              p.next=c)
9        return(p,c);
10    else
11       unlock(p);
12       unlock(c);
```

rmv(5)

```
rmv(e):
local p, c, n, r
1  (p,c) := locate(e);
2  if (c.val = e)
3     c.mark := true
4     n := c.next;
5     p.next := n;
6     r := true;
7  else r := false;
8  unlock(p);
9  unlock(c);
10 return r;
```



rmv(3)

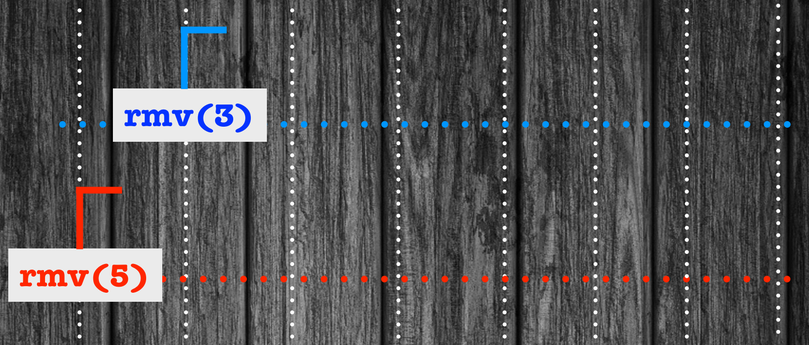
```
rmv(e):
local p, c, n, r
1  (p,c) := locate(e);
2  if (c.val = e)
3
4     n := c.next;
5     p.next := n;
6     r := true;
7  else r := false;
8  unlock(p);
9  unlock(c);
10 return r;
```

Sets | Lazy Synchronization

Scenario 1

- no logical deletion
- only physical deletion

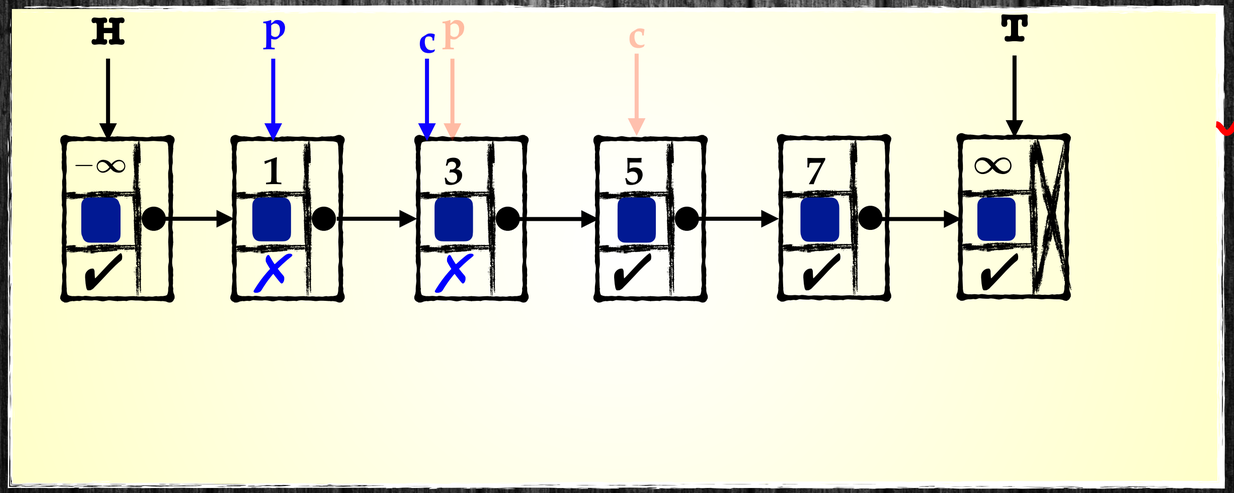
```
locate(e):
local p, c
1  while (true)
2     p := head;
3     c := p.next;
4     while (c.val < e)
5        p := c;
6        c := c.next
7     lock(p); lock(c);
8     if (!p.mark && !c.mark && p.next=c)
9        return(p,c);
10    else
11       unlock(p);
12       unlock(c);
```




```
locate(e):
local p, c
1  while (true)
2     p := head;
3     c := p.next;
4     while (c.val < e)
5        p := c;
6        c := c.next
7     lock(p); lock(c);
8     if (                              p.next=c)
9        return(p,c);
10    else
11       unlock(p);
12       unlock(c);
```

rmv(5)

```
rmv(e):
local p, c, n, r
1  (p,c) := locate(e);
2  if (c.val = e)
3     c.mark := true
4     n := c.next;
5     p.next := n;
6     r := true;
7  else r := false;
8  unlock(p);
9  unlock(c);
10 return r;
```



rmv(3)

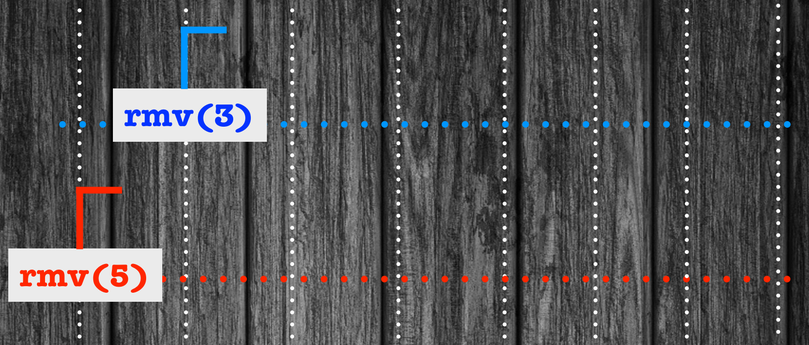
```
rmv(e):
local p, c, n, r
1  (p,c) := locate(e);
2  if (c.val = e)
3
4     n := c.next;
5     p.next := n;
6     r := true;
7  else r := false;
8  unlock(p);
9  unlock(c);
10 return r;
```

Sets | Lazy Synchronization

Scenario 1

- no logical deletion
- only physical deletion

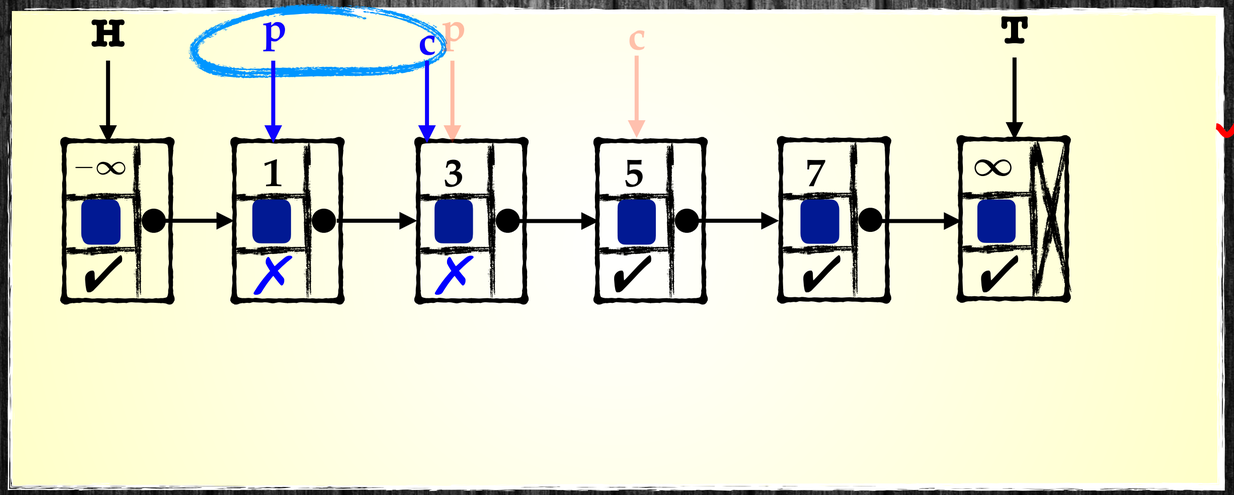
```
locate(e):
local p, c
1  while (true)
2     p := head;
3     c := p.next;
4     while (c.val < e)
5        p := c;
6        c := c.next
7     lock(p); lock(c);
8     if (!p.mark && !c.mark && p.next=c)
9        return(p,c);
10    else
11       unlock(p);
12       unlock(c);
```




```
locate(e):
local p, c
1  while (true)
2     p := head;
3     c := p.next;
4     while (c.val < e)
5        p := c;
6        c := c.next
7     lock(p); lock(c);
8     if (                          p.next=c)
9        return(p,c);
10    else
11       unlock(p);
12       unlock(c);
```

rmv(5)

```
rmv(e):
local p, c, n, r
1  (p,c) := locate(e);
2  if (c.val = e)
3     c.mark := true
4     n := c.next;
5     p.next := n;
6     r := true;
7  else r := false;
8  unlock(p);
9  unlock(c);
10 return r;
```



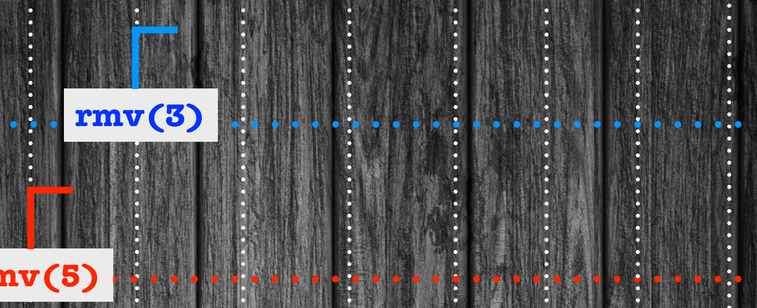
rmv(3)

```
rmv(e):
local p, c, n, r
1  (p,c) := locate(e);
2  if (c.val = e)
3
4     n := c.next;
5     p.next := n;
6     r := true;
7  else r := false;
8  unlock(p);
9  unlock(c);
10 return r;
```

Scenario 1

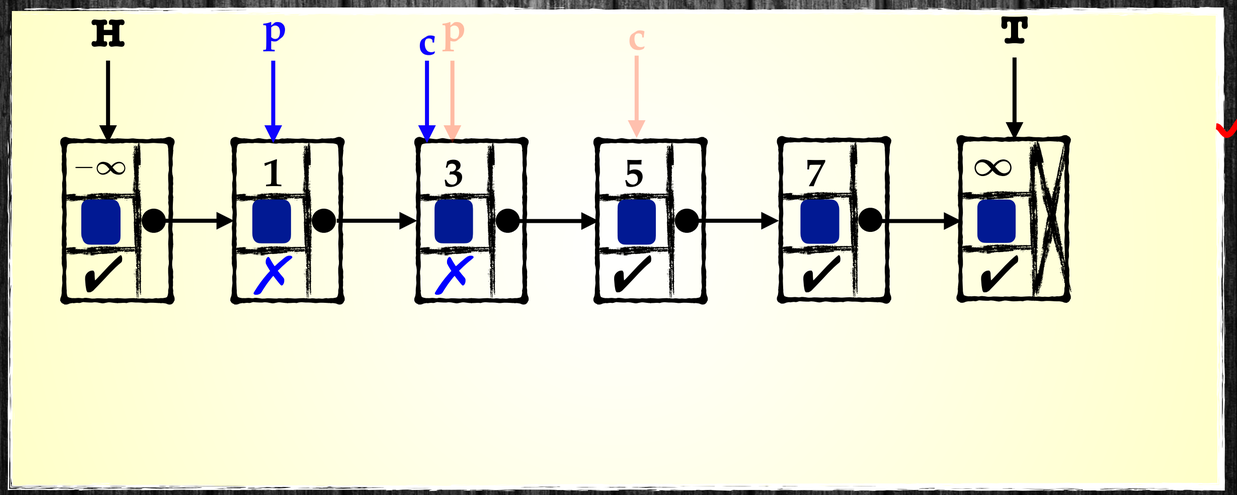
- no logical deletion
- only physical deletion

```
locate(e):
local p, c
1  while (true)
2     p := head;
3     c := p.next;
4     while (c.val < e)
5        p := c;
6        c := c.next
7     lock(p); lock(c);
8     if (!p.mark && !c.mark && p.next=c)
9        return(p,c);
10    else
11       unlock(p);
12       unlock(c);
```



rmv(5)

```
rmv(e):
local p, c, n, r
1  (p,c) := locate(e);
2  if (c.val = e)
3     c.mark := true
4     n := c.next;
5     p.next := n;
6     r := true;
7  else r := false;
8  unlock(p);
9  unlock(c);
10 return r;
```



rmv(3)

```
rmv(e):
local p, c, n, r
1  (p,c) := locate(e);
2  if (c.val = e)
3
4     n := c.next;
5     p.next := n;
6     r := true;
7  else r := false;
8  unlock(p);
9  unlock(c);
10 return r;
```

Sets | Lazy Synchronization

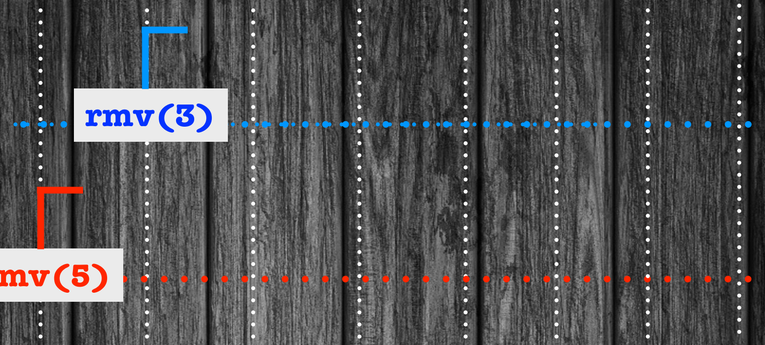
```
locate(e):
local p, c
1  while (true)
2     p := head;
3     c := p.next;
4     while (c.val < e)
5        p := c;
6        c := c.next
7     lock(p); lock(c);
8     if (!p.mark && !c.mark && p.next=c)
9        return(p,c);
10    else
11       unlock(p);
12       unlock(c);
```

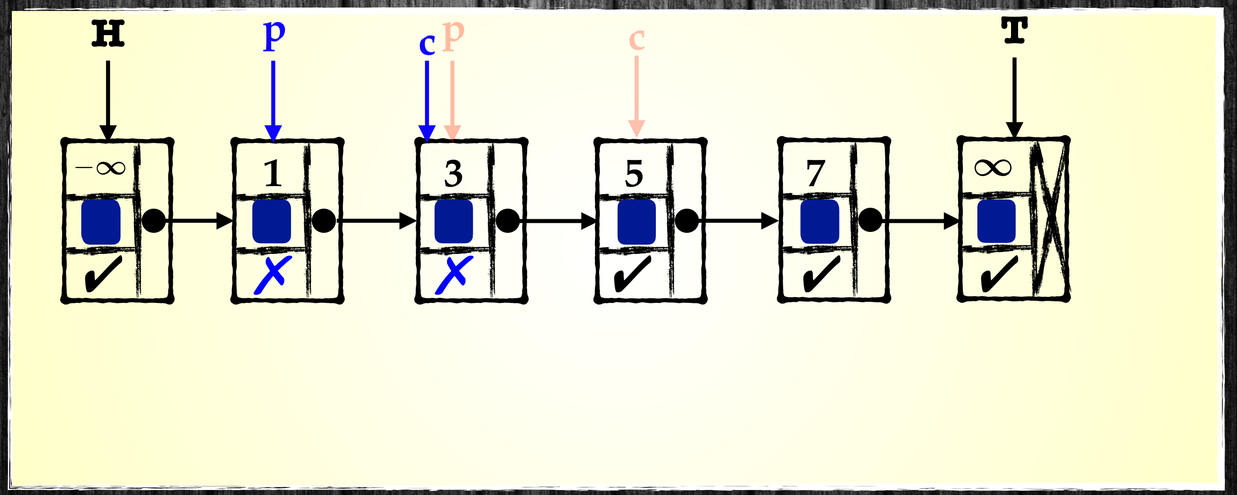
```
rmv(e):
local p, c, n, r
1  (p,c) := locate(e);
2  if (c.val = e)
3     c.mark := true
4     n := c.next;
5     p.next := n;
6     r := true;
7  else r := false;
8  unlock(p);
9  unlock(c);
10 return r;
```

rmv(5)

Scenario 1

- no logical deletion
- only physical deletion





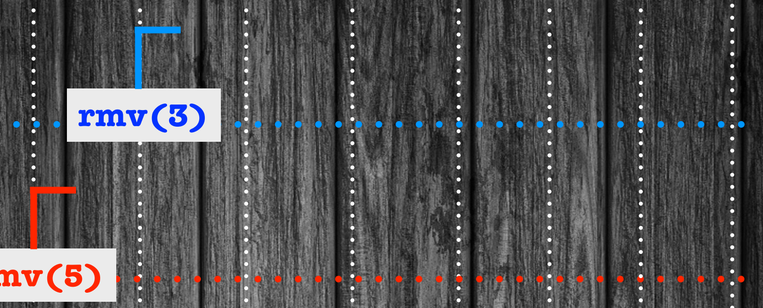
rmv(3)

```
rmv(e):
local p, c, n, r
1  (p,c) := locate(e);
2  if (c.val = e)
3
4     n := c.next;
5     p.next := n;
6     r := true;
7  else r := false;
8  unlock(p);
9  unlock(c);
10 return r;
```

Scenario 1

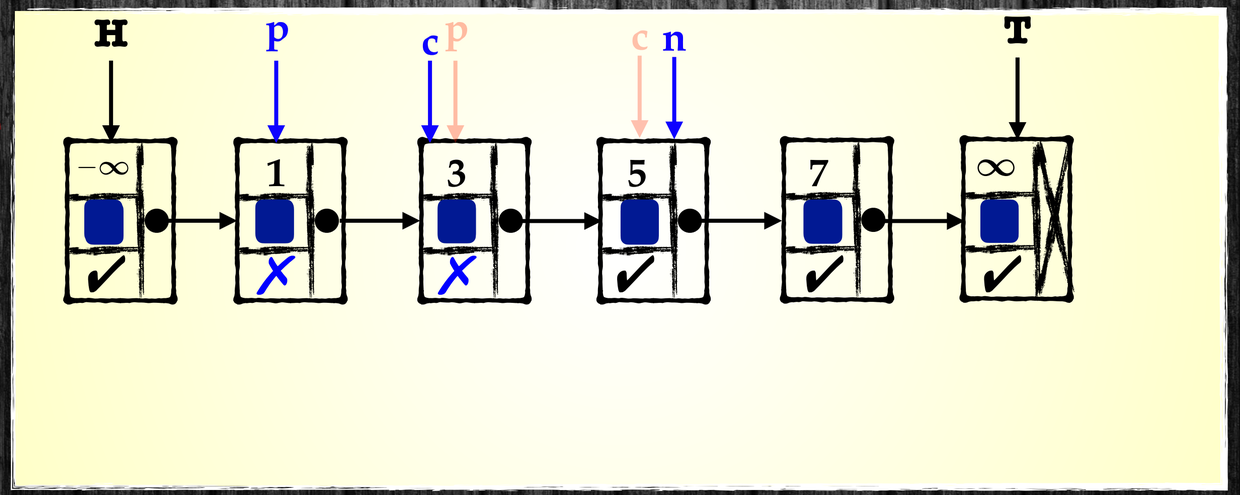
- no logical deletion
- only physical deletion

```
locate(e):
local p, c
1  while (true)
2     p := head;
3     c := p.next;
4     while (c.val < e)
5        p := c;
6        c := c.next
7     lock(p); lock(c);
8     if (!p.mark && !c.mark && p.next=c)
9        return(p,c);
10    else
11       unlock(p);
12       unlock(c);
```



rmv(5)

```
rmv(e):
local p, c, n, r
1  (p,c) := locate(e);
2  if (c.val = e)
3     c.mark := true
4     n := c.next;
5     p.next := n;
6     r := true;
7  else r := false;
8  unlock(p);
9  unlock(c);
10 return r;
```



rmv(3)

```
rmv(e):
local p, c, n, r
1  (p,c) := locate(e);
2  if (c.val = e)
3
4     n := c.next;
5     p.next := n;
6     r := true;
7  else r := false;
8  unlock(p);
9  unlock(c);
10 return r;
```

Sets | Lazy Synchronization

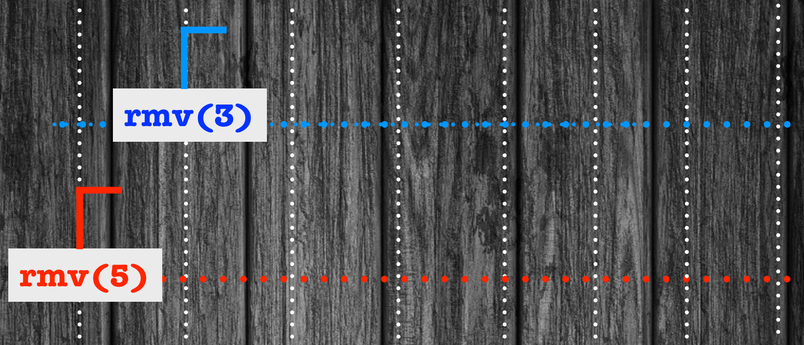
```
locate(e):
local p, c
1  while (true)
2     p := head;
3     c := p.next;
4     while (c.val < e)
5        p := c;
6        c := c.next
7     lock(p); lock(c);
8     if (!p.mark && !c.mark && p.next=c)
9        return(p,c);
10    else
11       unlock(p);
12       unlock(c);
```

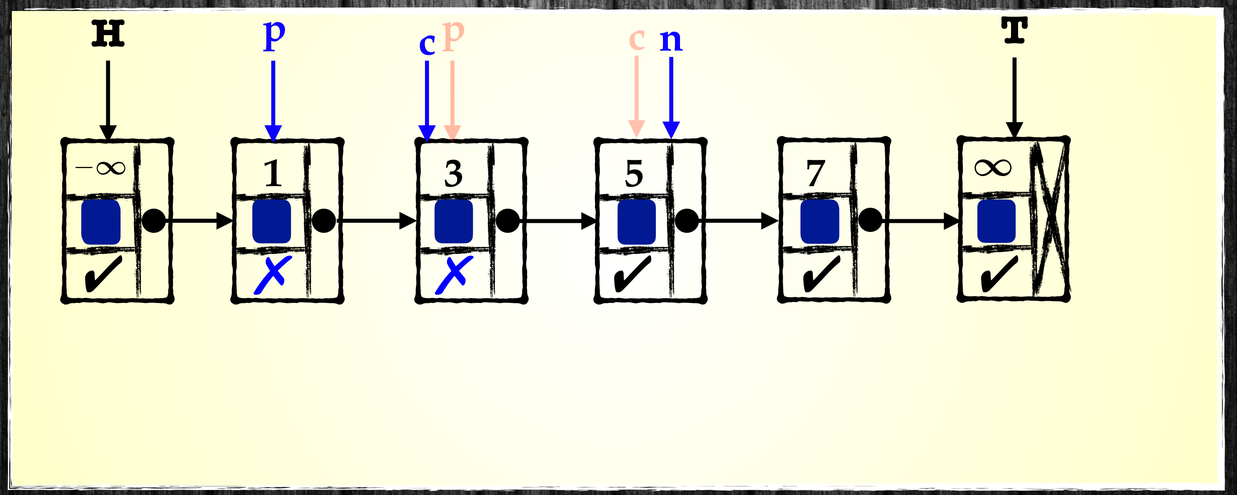
```
rmv(e):
local p, c, n, r
1  (p,c) := locate(e);
2  if (c.val = e)
3     c.mark := true
4     n := c.next;
5     p.next := n;
6     r := true;
7  else r := false;
8  unlock(p);
9  unlock(c);
10 return r;
```

rmv(5)

Scenario 1

- no logical deletion
- only physical deletion






```
rmv(e):
local p, c, n, r
1  (p,c) := locate(e);
2  if (c.val = e)
3
4     n := c.next;
5     p.next := n;
6     r := true;
7  else r := false;
8  unlock(p);
9  unlock(c);
10 return r;
```

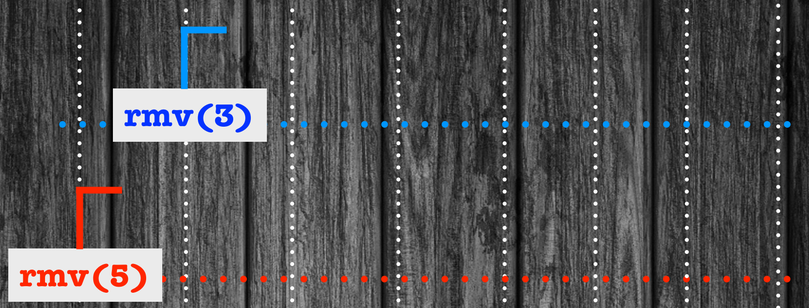
rmv(3)

Sets | Lazy Synchronization

Scenario 1

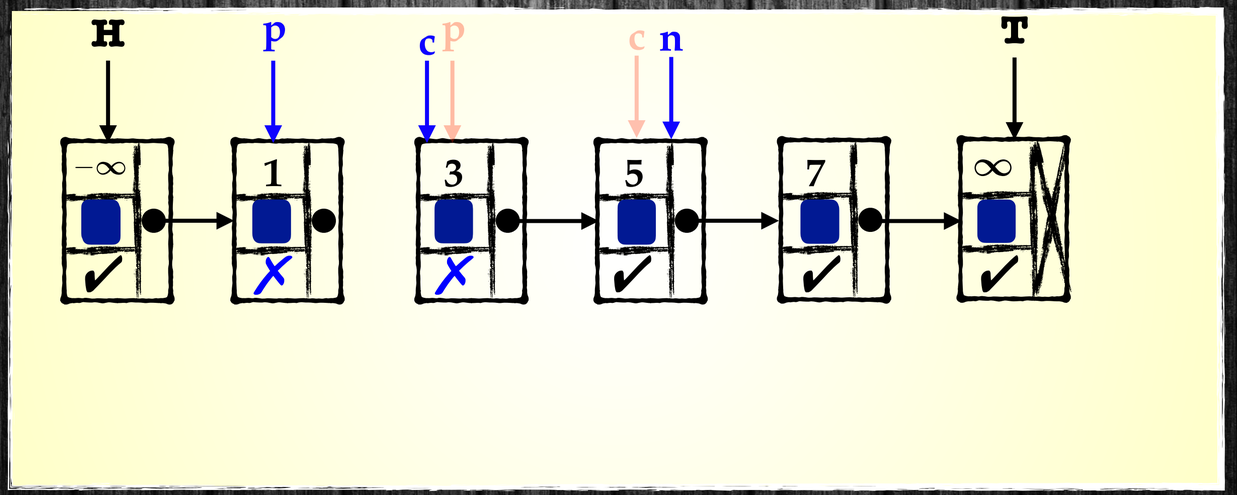
- no logical deletion
- only physical deletion

```
locate(e):
local p, c
1  while (true)
2     p := head;
3     c := p.next;
4     while (c.val < e)
5        p := c;
6        c := c.next
7     lock(p); lock(c);
8     if (!p.mark && !c.mark && p.next=c)
9        return(p,c);
10    else
11       unlock(p);
12       unlock(c);
```



rmv(5)

```
rmv(e):
local p, c, n, r
1  (p,c) := locate(e);
2  if (c.val = e)
3     c.mark := true
4     n := c.next;
5     p.next := n;
6     r := true;
7  else r := false;
8  unlock(p);
9  unlock(c);
10 return r;
```



rmv(3)

```
rmv(e):
local p, c, n, r
1  (p,c) := locate(e);
2  if (c.val = e)
3
4     n := c.next;
5     p.next := n;
6     r := true;
7  else r := false;
8  unlock(p);
9  unlock(c);
10 return r;
```

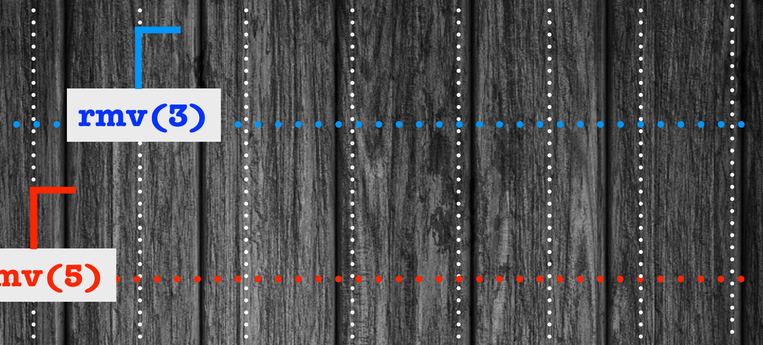
```
locate(e):
local p, c
1  while (true)
2    p := head;
3    c := p.next;
4    while (c.val < e)
5      p := c;
6      c := c.next
7    lock(p); lock(c);
8    if (!p.mark && !c.mark && p.next=c)
9      return(p,c);
10   else
11     unlock(p);
12     unlock(c);
```

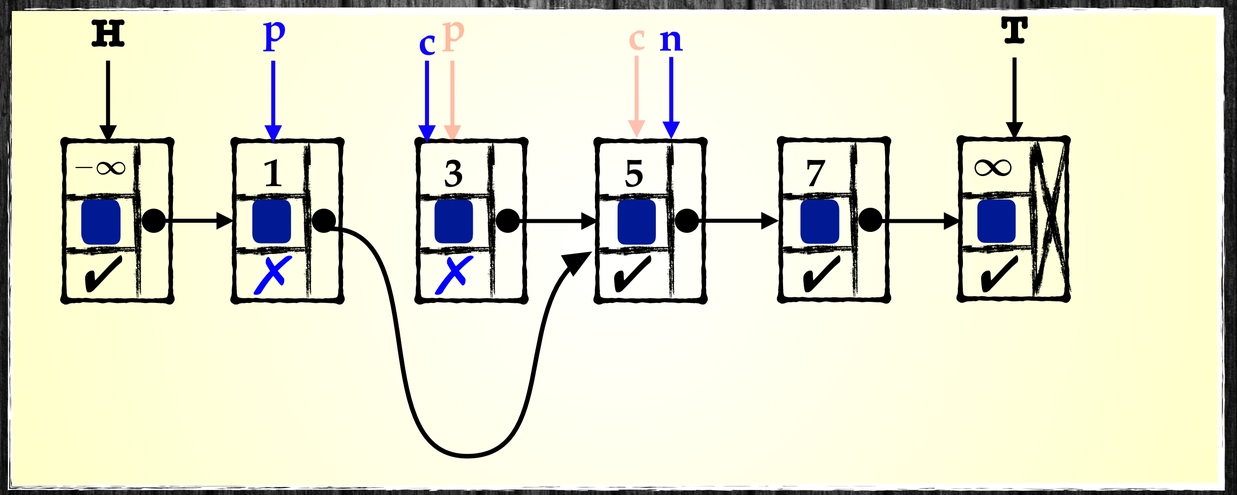
```
rmv(e):
local p, c, n, r
1  (p,c) := locate(e);
2  if (c.val = e)
3    c.mark := true
4    n := c.next;
5    p.next := n;
6    r := true;
7  else r := false;
8  unlock(p);
9  unlock(c);
10 return r;
```

rmv(5)

Scenario 1

- no logical deletion
- only physical deletion





rmv(3)

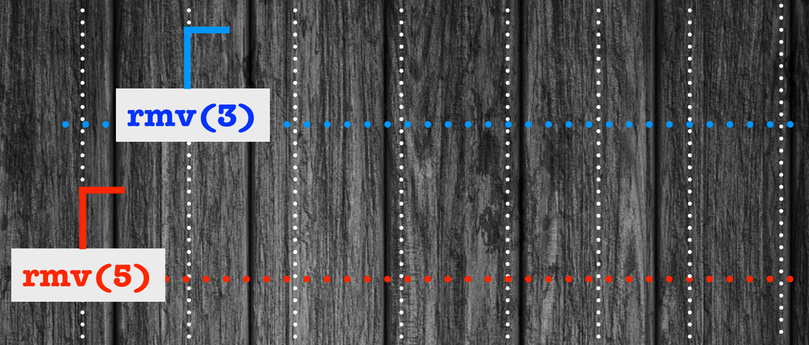
```
rmv(e):
local p, c, n, r
1  (p,c) := locate(e);
2  if (c.val = e)
3
4    n := c.next;
5    p.next := n;
6    r := true;
7  else r := false;
8  unlock(p);
9  unlock(c);
10 return r;
```

Sets | Lazy Synchronization

```
locate(e):
local p, c
1  while (true)
2     p := head;
3     c := p.next;
4     while (c.val < e)
5        p := c;
6        c := c.next
7     lock(p); lock(c);
8     if (!p.mark && !c.mark && p.next=c)
9        return(p,c);
10    else
11       unlock(p);
12       unlock(c);
```

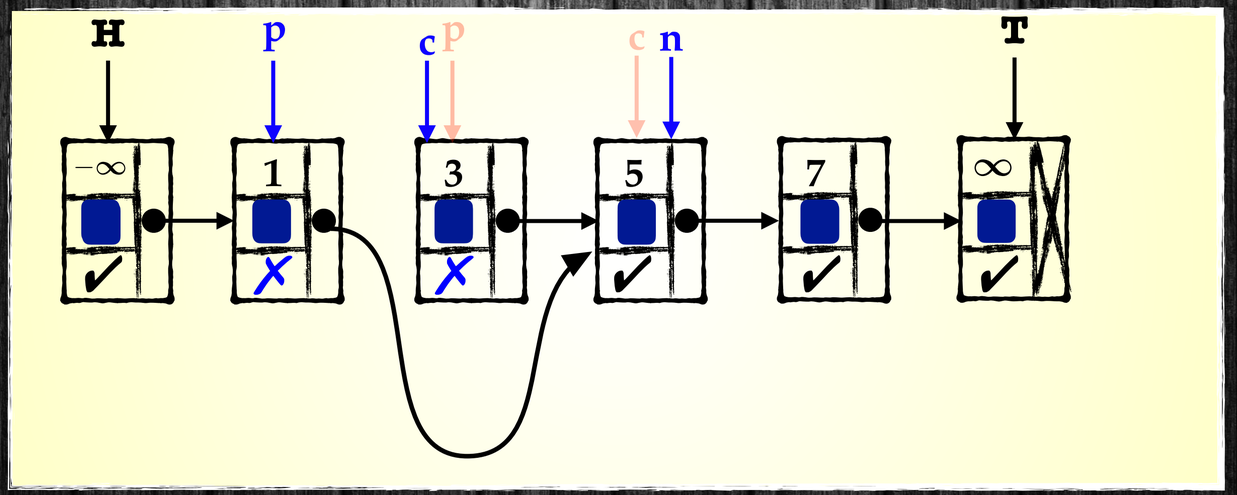
Scenario 1

- no logical deletion
- only physical deletion



rmv(5)

```
rmv(e):
local p, c, n, r
1  (p,c) := locate(e);
2  if (c.val = e)
3     c.mark := true
4     n := c.next;
5     p.next := n;
6     r := true;
7  else r := false;
8  unlock(p);
9  unlock(c);
10 return r;
```



rmv(3)

```
rmv(e):
local p, c, n, r
1  (p,c) := locate(e);
2  if (c.val = e)
3     c.mark := true
4     n := c.next;
5     p.next := n;
6     r := true;
7  else r := false;
8  unlock(p);
9  unlock(c);
10 return r;
```

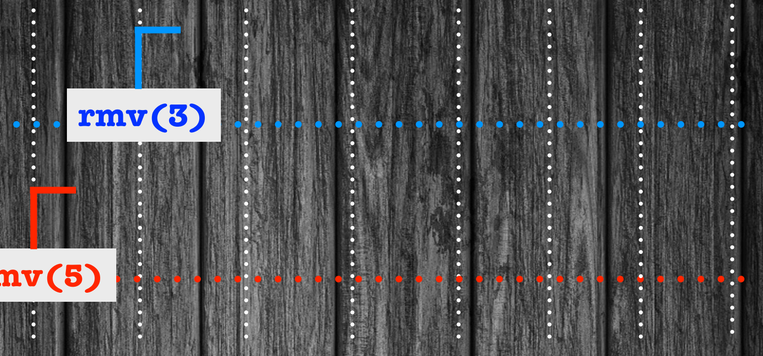
```
locate(e):
local p, c
1  while (true)
2     p := head;
3     c := p.next;
4     while (c.val < e)
5        p := c;
6        c := c.next
7     lock(p); lock(c);
8     if (!p.mark && !c.mark && p.next=c)
9        return(p,c);
10    else
11       unlock(p);
12       unlock(c);
```

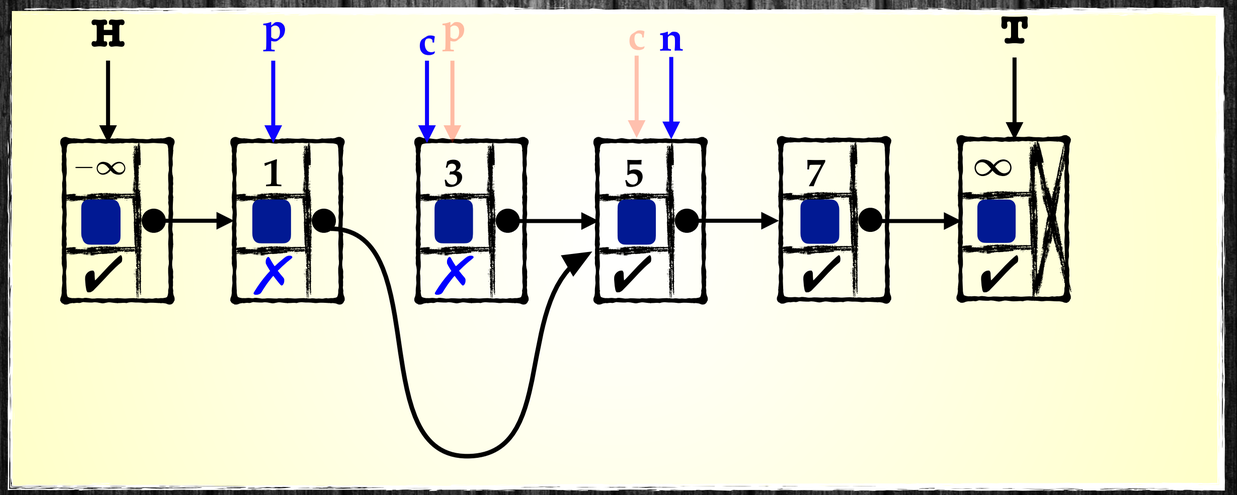
```
rmv(e):
local p, c, n, r
1  (p,c) := locate(e);
2  if (c.val = e)
3     c.mark := true
4     n := c.next;
5     p.next := n;
6     r := true;
7  else r := false;
8  unlock(p);
9  unlock(c);
10 return r;
```

rmv(5)

Scenario 1

- no logical deletion
- only physical deletion





rmv(3)

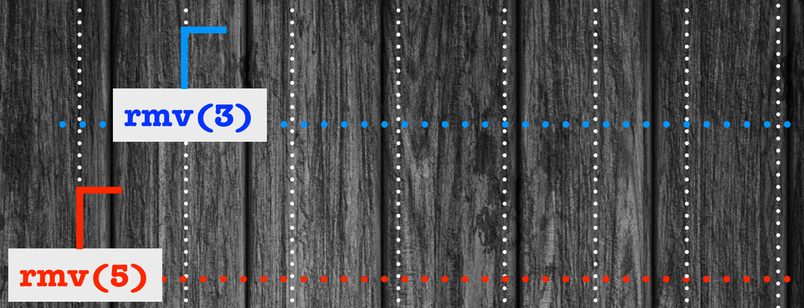
```
rmv(e):
local p, c, n, r
1  (p,c) := locate(e);
2  if (c.val = e)
3
4     n := c.next;
5     p.next := n;
6     r := true;
7  else r := false;
8  unlock(p);
9  unlock(c);
10 return r;
```

Sets | Lazy Synchronization

Scenario 1

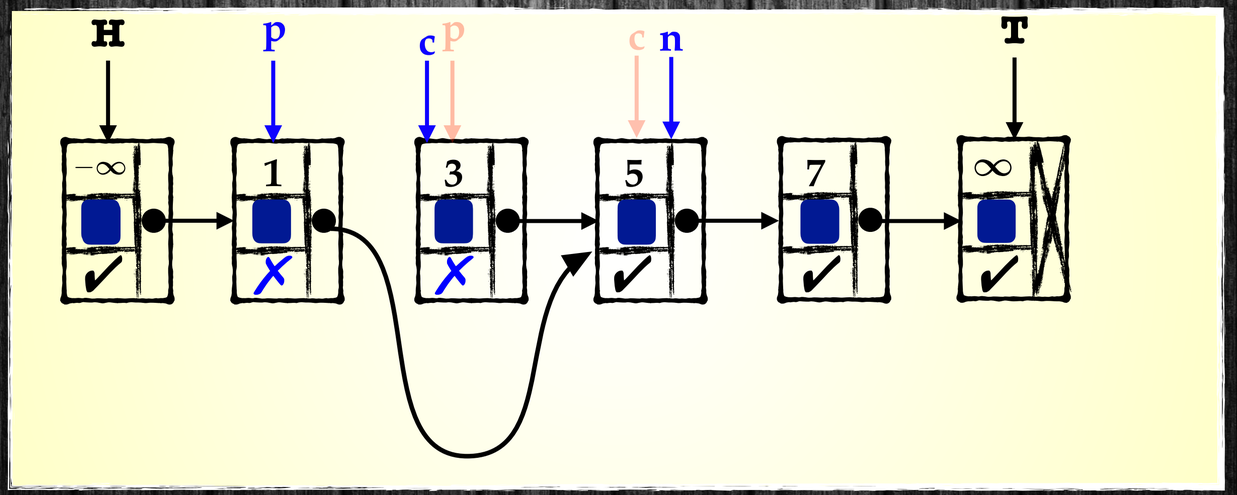
- no logical deletion
- only physical deletion

```
locate(e):
local p, c
1  while (true)
2    p := head;
3    c := p.next;
4    while (c.val < e)
5      p := c;
6      c := c.next
7    lock(p); lock(c);
8    if (!p.mark && !c.mark && p.next=c)
9      return(p,c);
10   else
11     unlock(p);
12     unlock(c);
```



rmv(5)

```
rmv(e):
local p, c, n, r
1  (p,c) := locate(e);
2  if (c.val = e)
3    c.mark := true
4    n := c.next;
5    p.next := n;
6    r := true;
7  else r := false;
8  unlock(p);
9  unlock(c);
10 return r;
```



rmv(3)

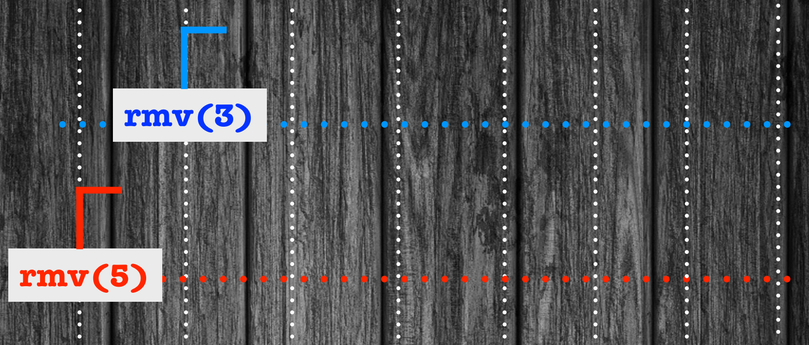
```
rmv(e):
local p, c, n, r
1  (p,c) := locate(e);
2  if (c.val = e)
3
4    n := c.next;
5    p.next := n;
6    r := true;
7  else r := false;
8  unlock(p);
9  unlock(c);
10 return r;
```

Sets | Lazy Synchronization

Scenario 1

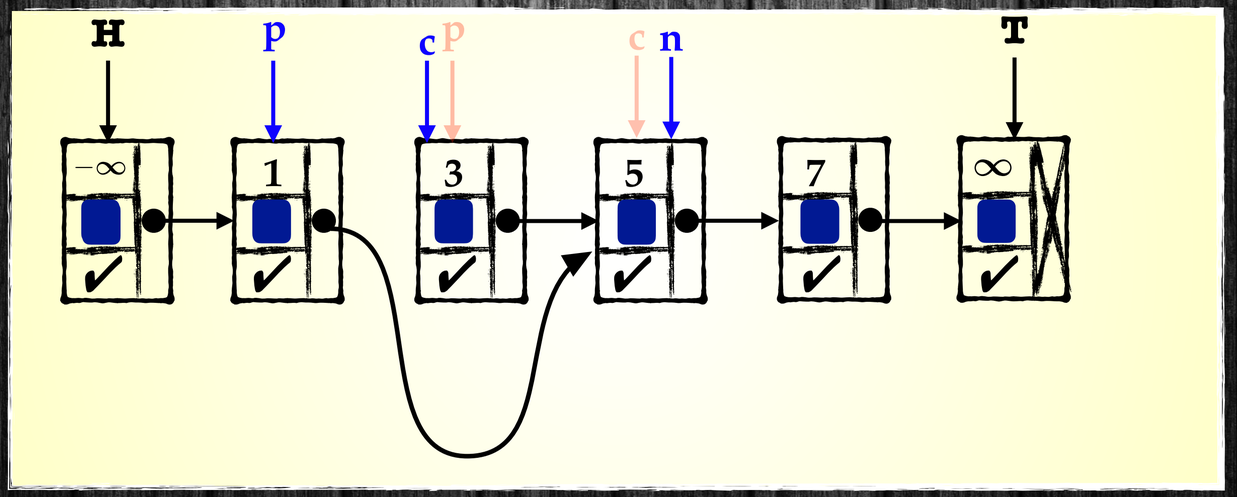
- no logical deletion
- only physical deletion

```
locate(e):
local p, c
1  while (true)
2     p := head;
3     c := p.next;
4     while (c.val < e)
5        p := c;
6        c := c.next
7     lock(p); lock(c);
8     if (!p.mark && !c.mark && p.next=c)
9        return(p,c);
10    else
11       unlock(p);
12       unlock(c);
```



rmv(5)

```
rmv(e):
local p, c, n, r
1  (p,c) := locate(e);
2  if (c.val = e)
3     c.mark := true
4     n := c.next;
5     p.next := n;
6     r := true;
7  else r := false;
8  unlock(p);
9  unlock(c);
10 return r;
```



rmv(3)

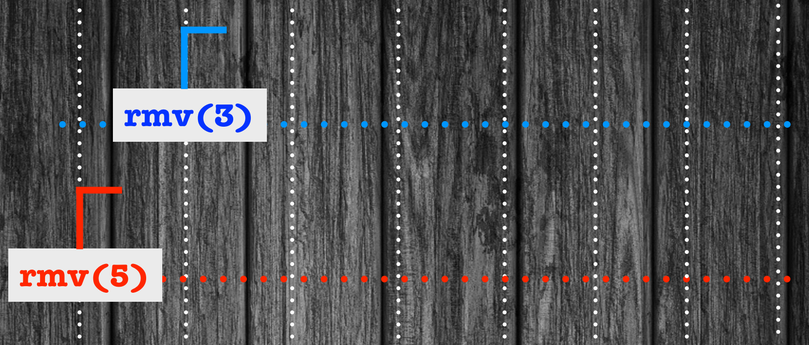
```
rmv(e):
local p, c, n, r
1  (p,c) := locate(e);
2  if (c.val = e)
3
4     n := c.next;
5     p.next := n;
6     r := true;
7  else r := false;
8  unlock(p);
9  unlock(c);
10 return r;
```

Sets | Lazy Synchronization

Scenario 1

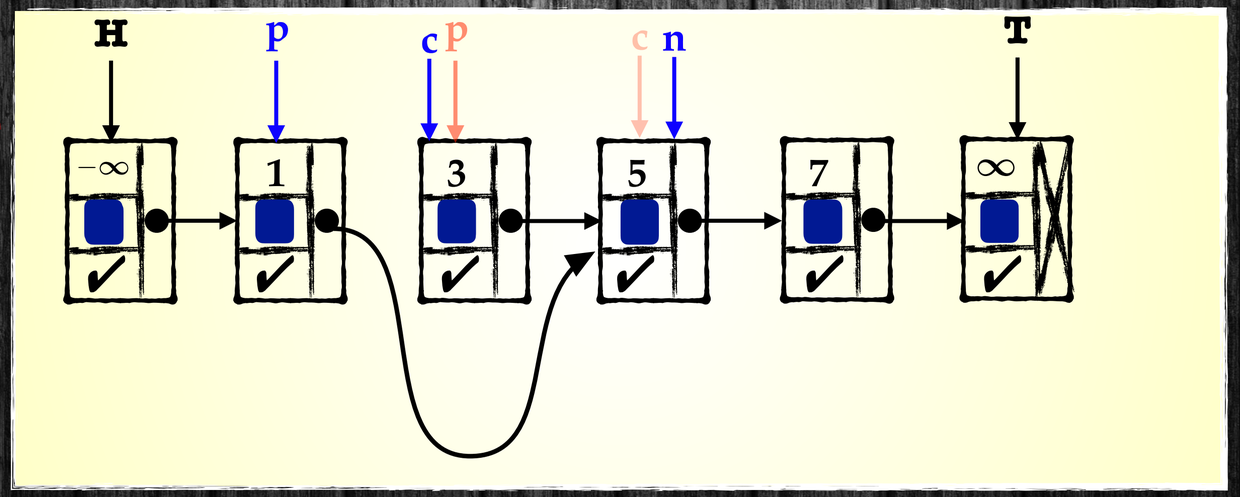
- no logical deletion
- only physical deletion

```
locate(e):
local p, c
1  while (true)
2     p := head;
3     c := p.next;
4     while (c.val < e)
5        p := c;
6        c := c.next
7     lock(p); lock(c);
8     if (!p.mark && !c.mark && p.next=c)
9        return(p,c);
10    else
11       unlock(p);
12       unlock(c);
```



rmv(5)

```
rmv(e):
local p, c, n, r
1  (p,c) := locate(e);
2  if (c.val = e)
3     c.mark := true
4     n := c.next;
5     p.next := n;
6     r := true;
7  else r := false;
8  unlock(p);
9  unlock(c);
10 return r;
```



rmv(3)

```
rmv(e):
local p, c, n, r
1  (p,c) := locate(e);
2  if (c.val = e)
3
4     n := c.next;
5     p.next := n;
6     r := true;
7  else r := false;
8  unlock(p);
9  unlock(c);
10 return r;
```

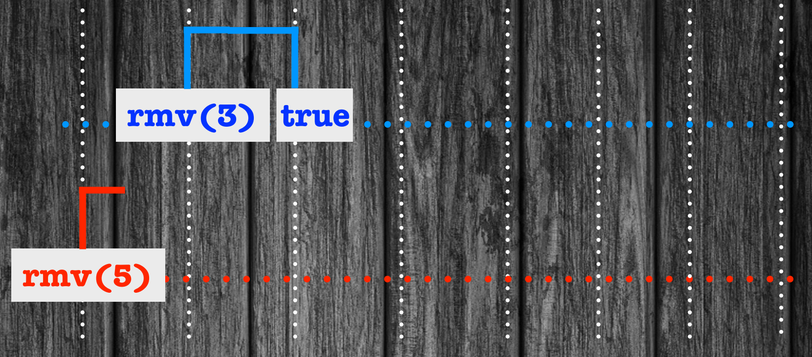
```
locate(e):
local p, c
1  while (true)
2     p := head;
3     c := p.next;
4     while (c.val < e)
5        p := c;
6        c := c.next
7     lock(p); lock(c);
8     if (!p.mark && !c.mark && p.next=c)
9        return(p,c);
10    else
11       unlock(p);
12       unlock(c);
```

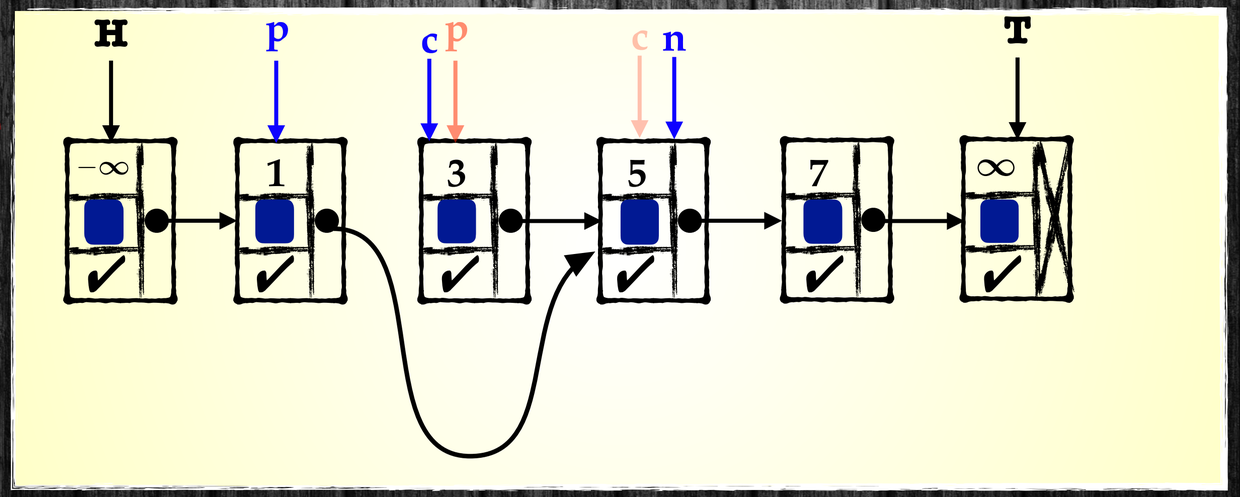
```
rmv(e):
local p, c, n, r
1  (p,c) := locate(e);
2  if (c.val = e)
3     c.mark := true
4     n := c.next;
5     p.next := n;
6     r := true;
7  else r := false;
8  unlock(p);
9  unlock(c);
10 return r;
```

rmv(5)

Scenario 1

- no logical deletion
- only physical deletion





rmv(3)

```
rmv(e):
local p, c, n, r
1  (p,c) := locate(e);
2  if (c.val = e)
3
4     n := c.next;
5     p.next := n;
6     r := true;
7  else r := false;
8  unlock(p);
9  unlock(c);
10 return r;
```

Sets | Lazy Synchronization

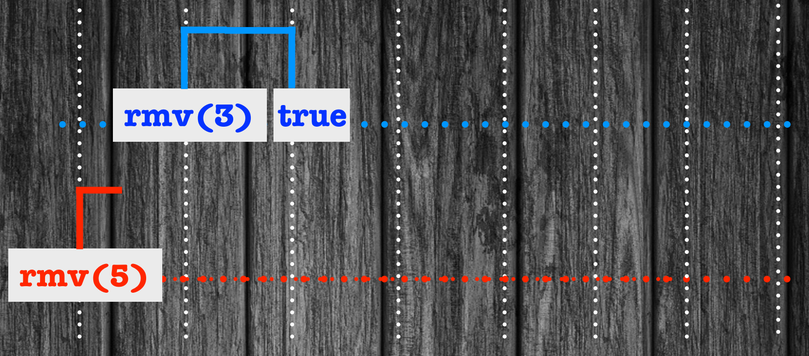
```
locate(e):
local p, c
1  while (true)
2    p := head;
3    c := p.next;
4    while (c.val < e)
5      p := c;
6      c := c.next
7    lock(p); lock(c);
8    if (!p.mark && !c.mark && p.next=c)
9      return(p,c);
10   else
11     unlock(p);
12     unlock(c);
```

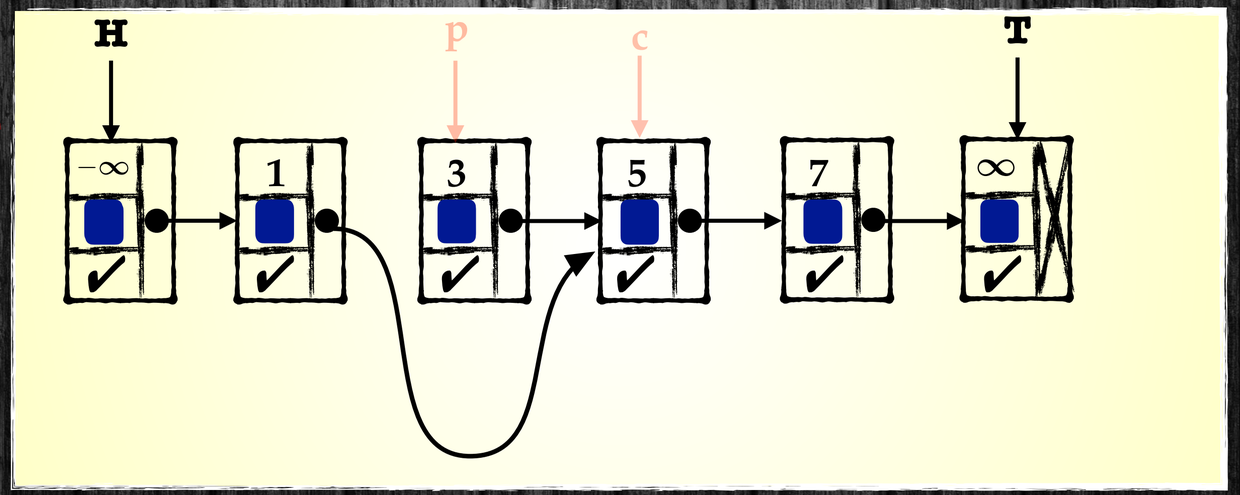
```
rmv(e):
local p, c, n, r
1  (p,c) := locate(e);
2  if (c.val = e)
3    c.mark := true
4    n := c.next;
5    p.next := n;
6    r := true;
7  else r := false;
8  unlock(p);
9  unlock(c);
10 return r;
```

rmv(5)

Scenario 1

- no logical deletion
- only physical deletion

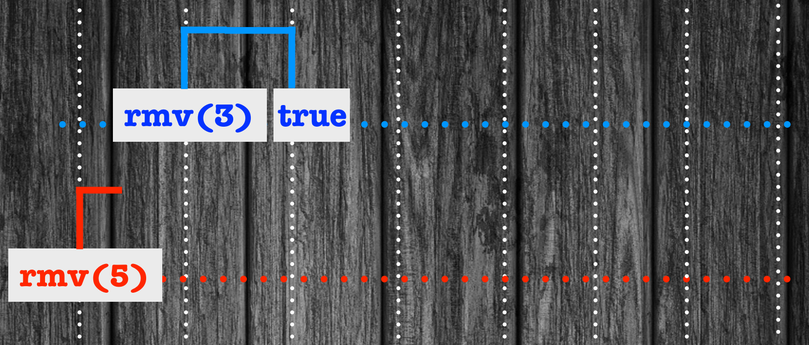




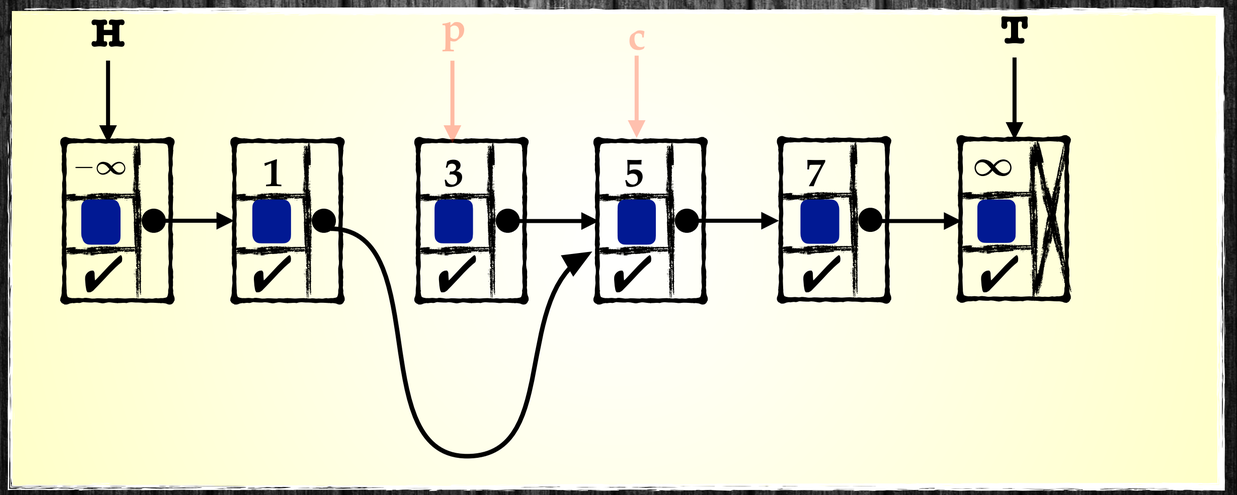
Scenario 1

- no logical deletion
- only physical deletion

```
locate(e):
local p, c
1  while (true)
2     p := head;
3     c := p.next;
4     while (c.val < e)
5        p := c;
6        c := c.next
7     lock(p); lock(c);
8     if (!p.mark && !c.mark && p.next=c)
9        return(p,c);
10    else
11       unlock(p);
12       unlock(c);
```




```
rmv(e):                    rmv(5)
local p, c, n, r
1  (p,c) := locate(e);
2  if (c.val = e)
3     c.mark := true
4     n := c.next;
5     p.next := n;
6     r := true;
7  else r := false;
8  unlock(p);
9  unlock(c);
10 return r;
```

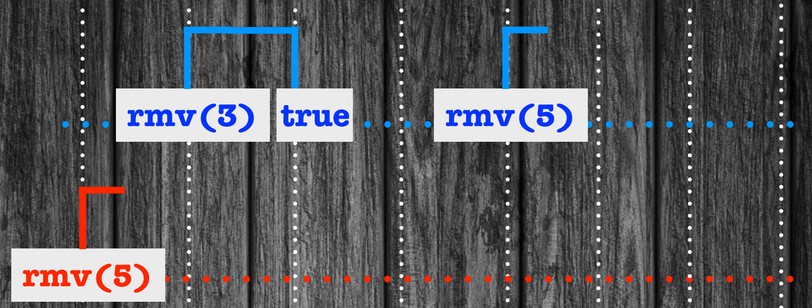



```
rmv(e):                    rmv(5)
local p, c, n, r
1  (p,c) := locate(e);
2  if (c.val = e)
3
4     n := c.next;
5     p.next := n;
6     r := true;
7  else r := false;
8  unlock(p);
9  unlock(c);
10 return r;
```

Scenario 1

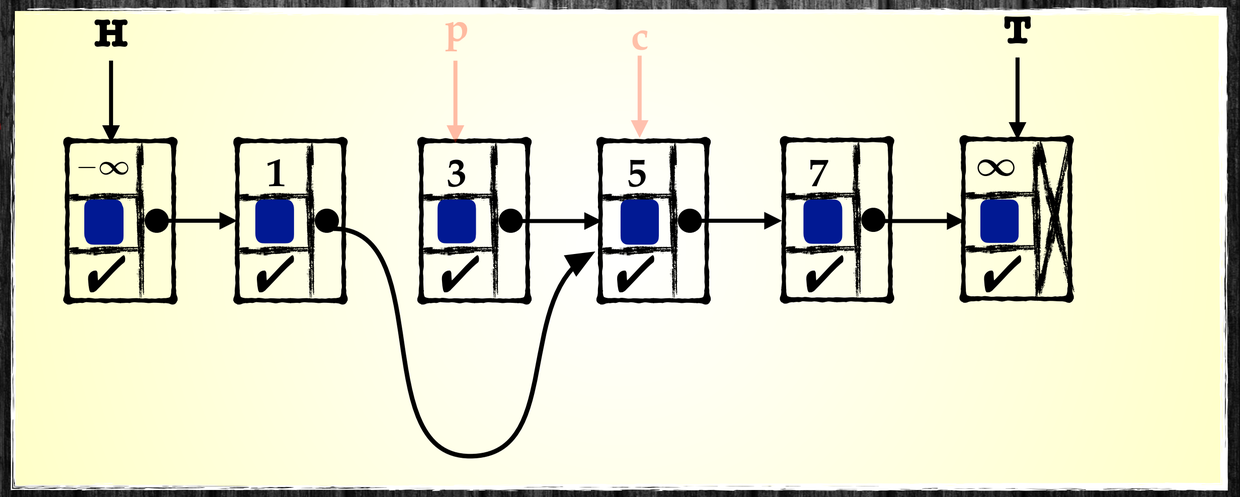
- no logical deletion
- only physical deletion

```
locate(e):
local p, c
1  while (true)
2    p := head;
3    c := p.next;
4    while (c.val < e)
5      p := c;
6      c := c.next
7    lock(p); lock(c);
8    if (!p.mark && !c.mark && p.next=c)
9      return(p,c);
10   else
11     unlock(p);
12     unlock(c);
```



rmv(5)

```
rmv(e):
local p, c, n, r
1  (p,c) := locate(e);
2  if (c.val = e)
3    c.mark := true
4    n := c.next;
5    p.next := n;
6    r := true;
7  else r := false;
8  unlock(p);
9  unlock(c);
10 return r;
```



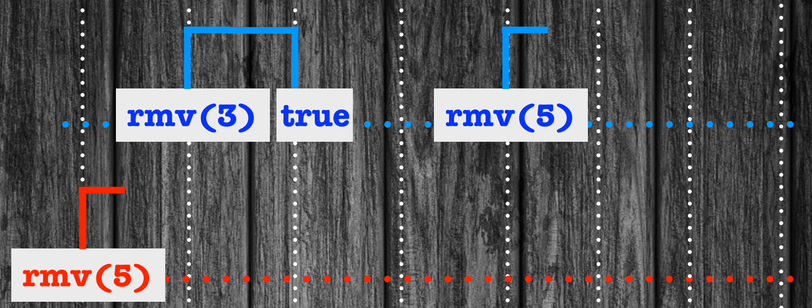
rmv(5)

```
rmv(e):
local p, c, n, r
1  (p,c) := locate(e);
2  if (c.val = e)
3
4    n := c.next;
5    p.next := n;
6    r := true;
7  else r := false;
8  unlock(p);
9  unlock(c);
10 return r;
```

Scenario 1

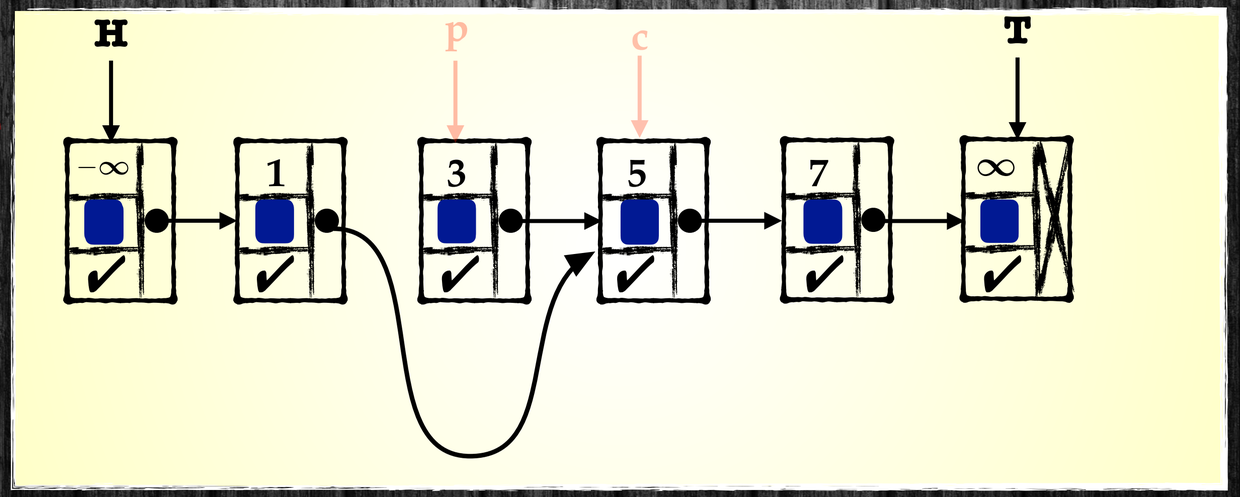
- no logical deletion
- only physical deletion

```
locate(e):
local p, c
1  while (true)
2    p := head;
3    c := p.next;
4    while (c.val < e)
5      p := c;
6      c := c.next
7    lock(p); lock(c);
8    if (!p.mark && !c.mark && p.next=c)
9      return(p,c);
10   else
11     unlock(p);
12     unlock(c);
```



rmv(5)

```
rmv(e):
local p, c, n, r
1  (p,c) := locate(e);
2  if (c.val = e)
3    c.mark := true
4    n := c.next;
5    p.next := n;
6    r := true;
7  else r := false;
8  unlock(p);
9  unlock(c);
10 return r;
```



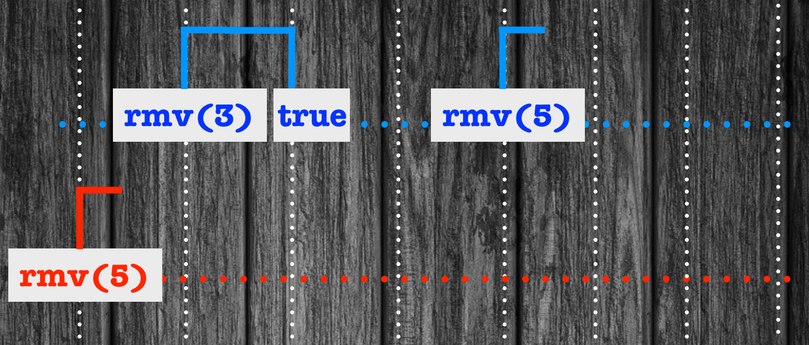
rmv(5)

```
rmv(e):
local p, c, n, r
1  (p,c) := locate(e);
2  if (c.val = e)
3
4    n := c.next;
5    p.next := n;
6    r := true;
7  else r := false;
8  unlock(p);
9  unlock(c);
10 return r;
```

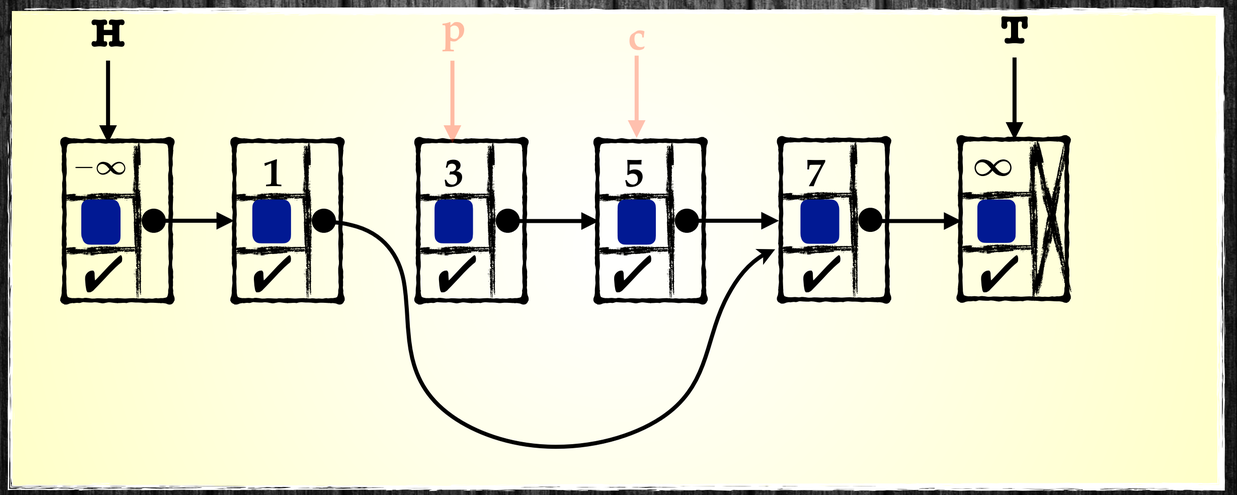
Scenario 1

- no logical deletion
- only physical deletion

```
locate(e):
local p, c
1  while (true)
2     p := head;
3     c := p.next;
4     while (c.val < e)
5        p := c;
6        c := c.next
7     lock(p); lock(c);
8     if (!p.mark && !c.mark && p.next=c)
9        return(p,c);
10    else
11       unlock(p);
12       unlock(c);
```




```
rmv(e):                    rmv(5)
local p, c, n, r
1  (p,c) := locate(e);
2  if (c.val = e)
3     c.mark := true
4     n := c.next;
5     p.next := n;
6     r := true;
7  else r := false;
8  unlock(p);
9  unlock(c);
10 return r;
```

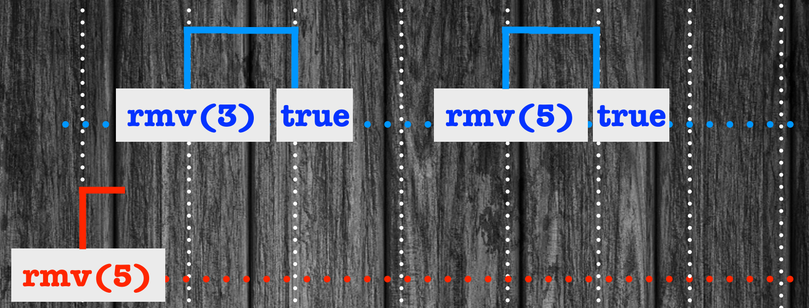



```
rmv(e):                    rmv(5)
local p, c, n, r
1  (p,c) := locate(e);
2  if (c.val = e)
3     c.mark := true
4     n := c.next;
5     p.next := n;
6     r := true;
7  else r := false;
8  unlock(p);
9  unlock(c);
10 return r;
```

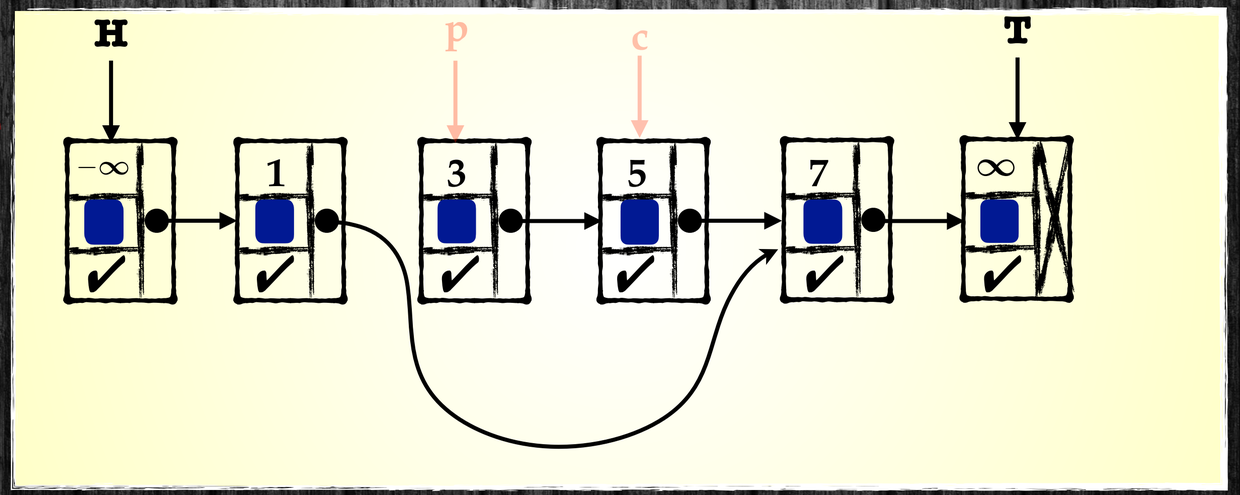
Scenario 1

- no logical deletion
- only physical deletion

```
locate(e):
local p, c
1  while (true)
2     p := head;
3     c := p.next;
4     while (c.val < e)
5        p := c;
6        c := c.next
7     lock(p); lock(c);
8     if (!p.mark && !c.mark && p.next=c)
9        return(p,c);
10    else
11       unlock(p);
12       unlock(c);
```




```
rmv(e):                    rmv(5)
local p, c, n, r
1  (p,c) := locate(e);
2  if (c.val = e)
3     c.mark := true
4     n := c.next;
5     p.next := n;
6     r := true;
7  else r := false;
8  unlock(p);
9  unlock(c);
10 return r;
```




```
rmv(e):                    rmv(5)
local p, c, n, r
1  (p,c) := locate(e);
2  if (c.val = e)
3     c.mark := true
4     n := c.next;
5     p.next := n;
6     r := true;
7  else r := false;
8  unlock(p);
9  unlock(c);
10 return r;
```

Sets | Lazy Synchronization

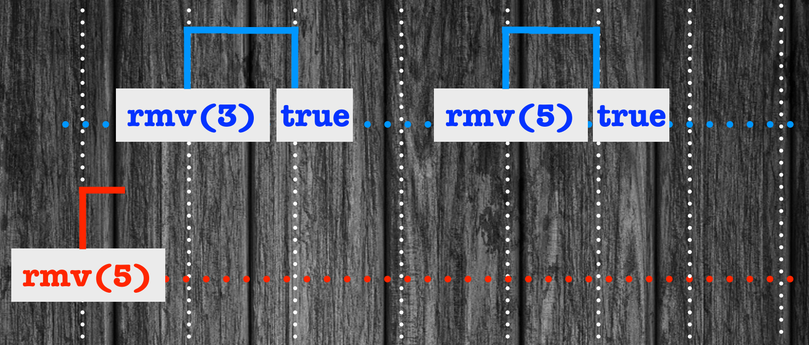
```
locate(e):
local p, c
1  while (true)
2    p := head;
3    c := p.next;
4    while (c.val < e)
5      p := c;
6      c := c.next
7    lock(p); lock(c);
8    if (!p.mark && !c.mark && p.next=c)
9      return(p,c);
10   else
11     unlock(p);
12     unlock(c);
```

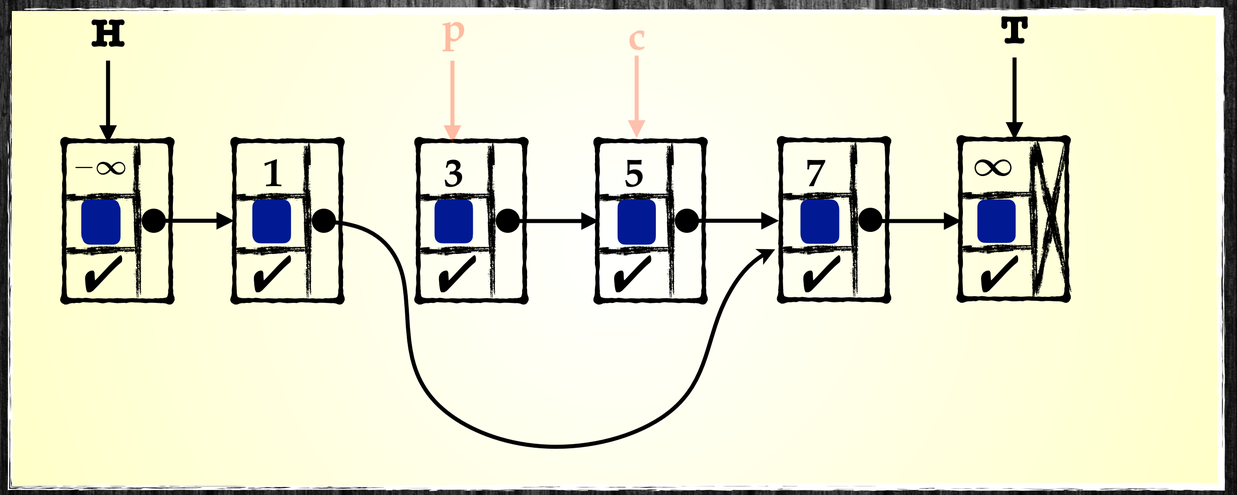
```
rmv(e):
local p, c, n, r
1  (p,c) := locate(e);
2  if (c.val = e)
3    c.mark := true
4    n := c.next;
5    p.next := n;
6    r := true;
7  else r := false;
8  unlock(p);
9  unlock(c);
10 return r;
```

rmv(5)

Scenario 1

- no logical deletion
- only physical deletion





Sets | Lazy Synchronization

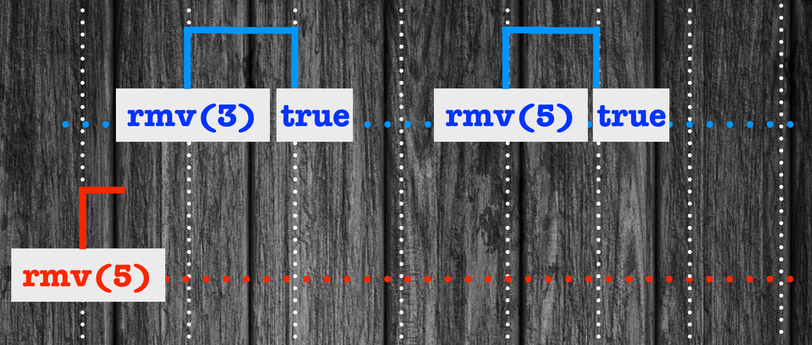
```
locate(e):
local p, c
1  while (true)
2     p := head;
3     c := p.next;
4     while (c.val < e)
5        p := c;
6        c := c.next
7     lock(p); lock(c);
8     if (                          p.next=c)
9        return(p,c);
10    else
11       unlock(p);
12       unlock(c);
```

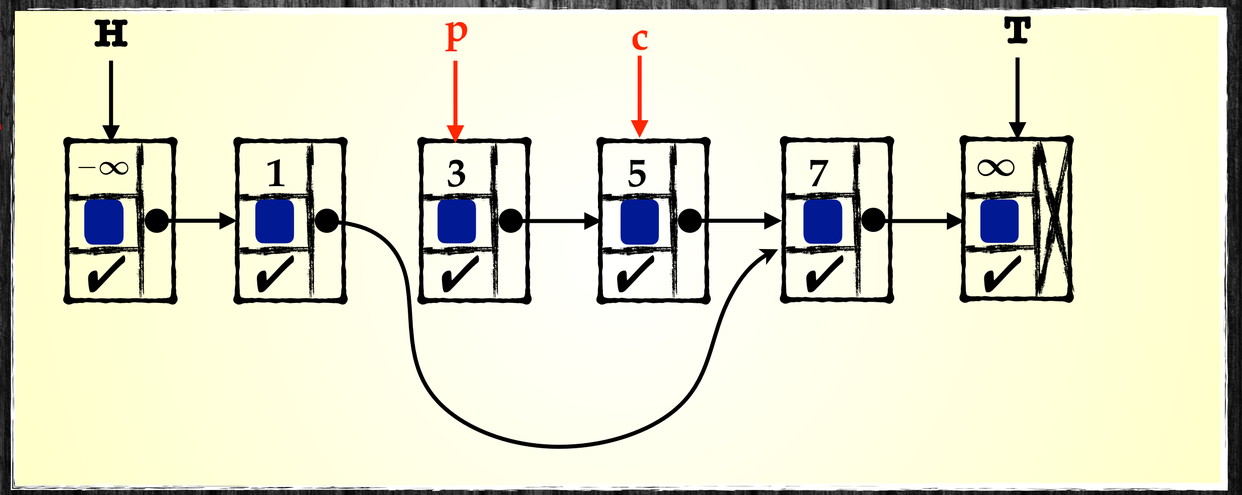
```
rmv(e):
local p, c, n, r
1  (p,c) := locate(e);
2  if (c.val = e)
3
4     n := c.next;
5     p.next := n;
6     r := true;
7  else r := false;
8  unlock(p);
9  unlock(c);
10 return r;
```

rmv(5)

Scenario 1

- no logical deletion
- only physical deletion





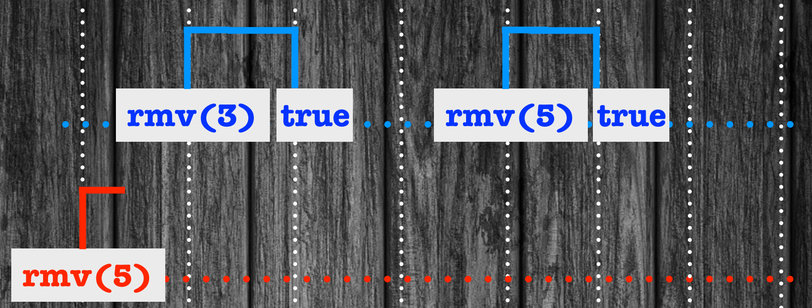
Sets | Lazy Synchronization

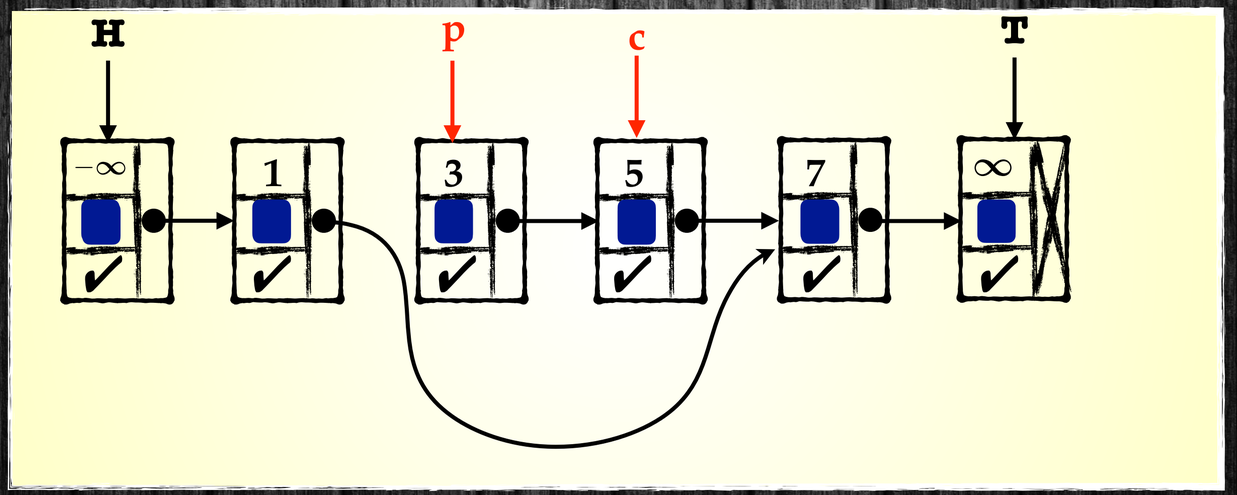
```
locate(e):
local p, c
1  while (true)
2     p := head;
3     c := p.next;
4     while (c.val < e)
5        p := c;
6        c := c.next
7     lock(p); lock(c);
8     if (                          p.next=c)
9        return(p,c);
10    else
11       unlock(p);
12       unlock(c);
```

```
rmv(e):                    rmv(5)
local p, c, n, r
1  (p,c) := locate(e);
2  if (c.val = e)
3
4     n := c.next;
5     p.next := n;
6     r := true;
7  else r := false;
8  unlock(p);
9  unlock(c);
10 return r;
```

Scenario 1

- no logical deletion
- only physical deletion





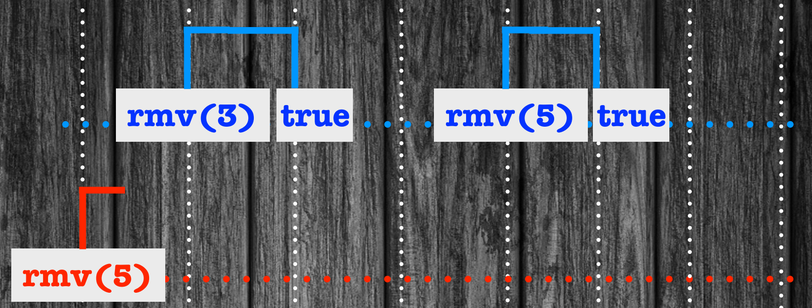
Sets | Lazy Synchronization

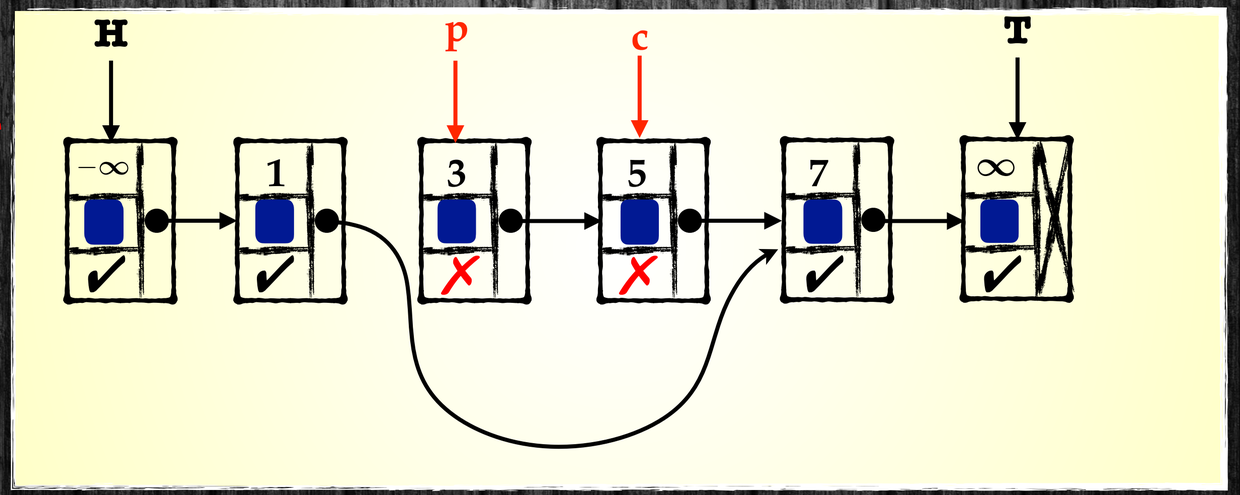
```
locate(e):
local p, c
1  while (true)
2     p := head;
3     c := p.next;
4     while (c.val < e)
5        p := c;
6        c := c.next
7     lock(p); lock(c);
8     if (                          p.next=c)
9        return(p,c);
10    else
11       unlock(p);
12       unlock(c);
```

```
rmv(e):                    rmv(5)
local p, c, n, r
1  (p,c) := locate(e);
2  if (c.val = e)
3
4     n := c.next;
5     p.next := n;
6     r := true;
7  else r := false;
8  unlock(p);
9  unlock(c);
10 return r;
```

Scenario 1

- no logical deletion
- only physical deletion





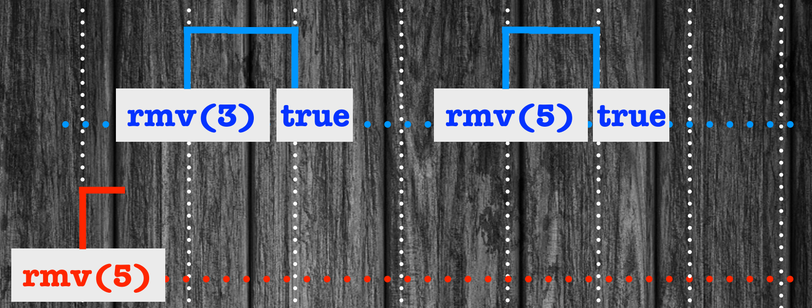
Sets | Lazy Synchronization

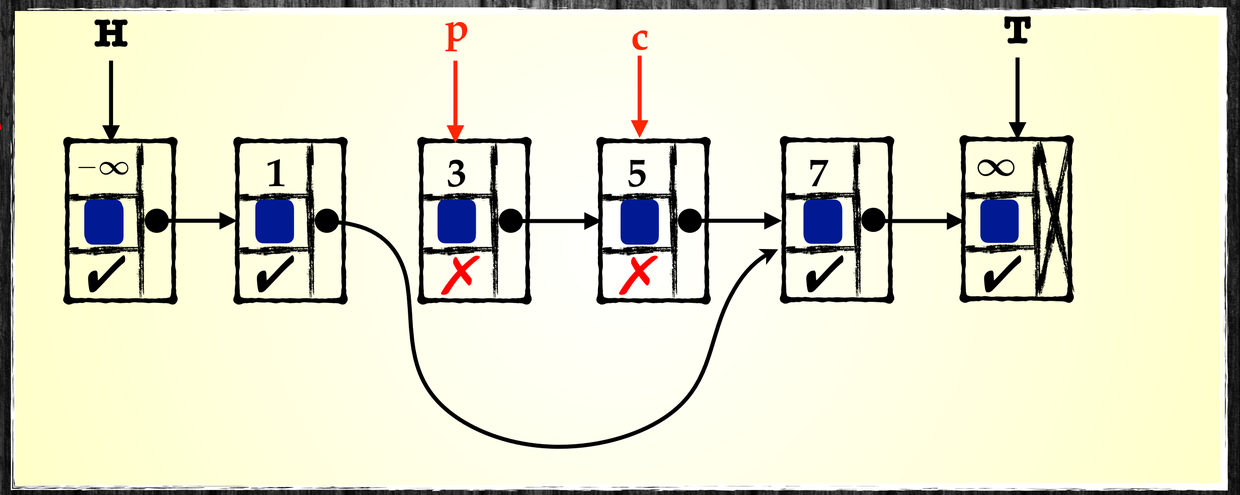
```
locate(e):
local p, c
1  while (true)
2     p := head;
3     c := p.next;
4     while (c.val < e)
5        p := c;
6        c := c.next
7     lock(p); lock(c);
8     if (                          p.next=c)
9        return(p,c);
10    else
11       unlock(p);
12       unlock(c);
```

```
rmv(e):                    rmv(5)
local p, c, n, r
1  (p,c) := locate(e);
2  if (c.val = e)
3
4     n := c.next;
5     p.next := n;
6     r := true;
7  else r := false;
8  unlock(p);
9  unlock(c);
10 return r;
```

Scenario 1

- no logical deletion
- only physical deletion





Sets | Lazy Synchronization

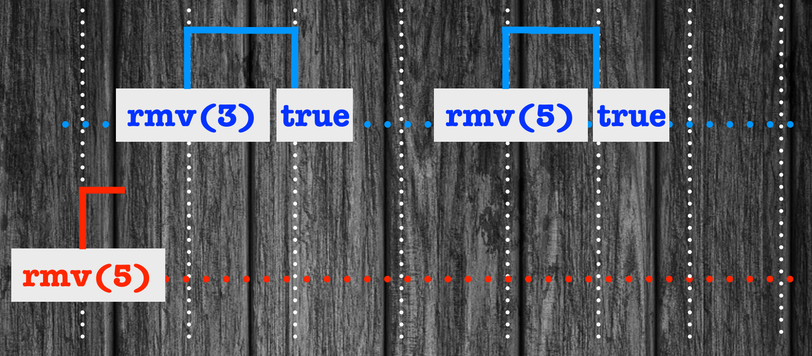
```
locate(e):
local p, c
1  while (true)
2     p := head;
3     c := p.next;
4     while (c.val < e)
5        p := c;
6        c := c.next
7     lock(p); lock(c);
8     if (                          p.next=c)
9        return(p,c);
10    else
11       unlock(p);
12       unlock(c);
```

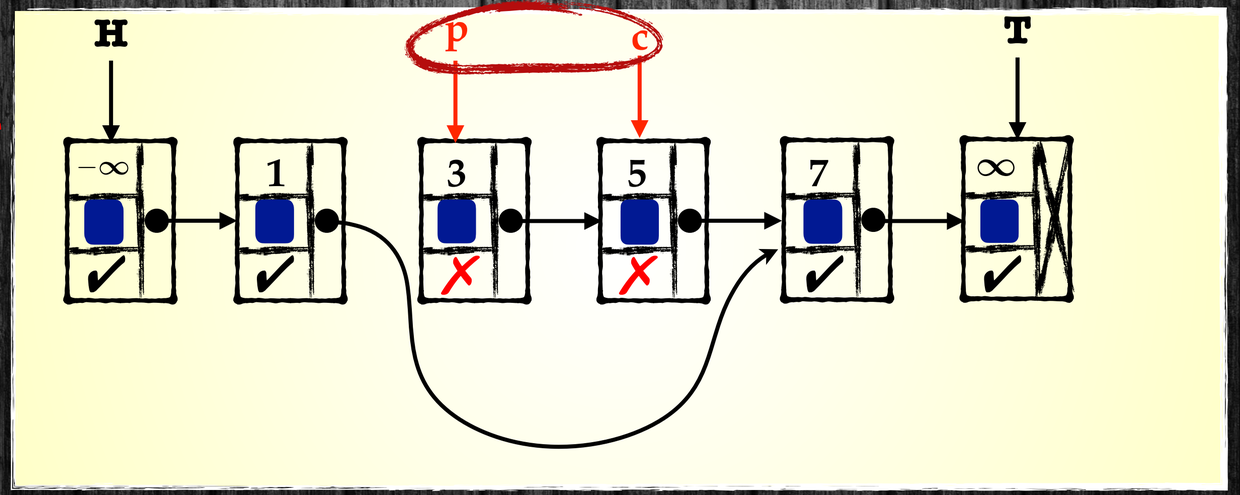
```
rmv(e):
local p, c, n, r
1  (p,c) := locate(e);
2  if (c.val = e)
3
4     n := c.next;
5     p.next := n;
6     r := true;
7  else r := false;
8  unlock(p);
9  unlock(c);
10 return r;
```

rmv(5)

Scenario 1

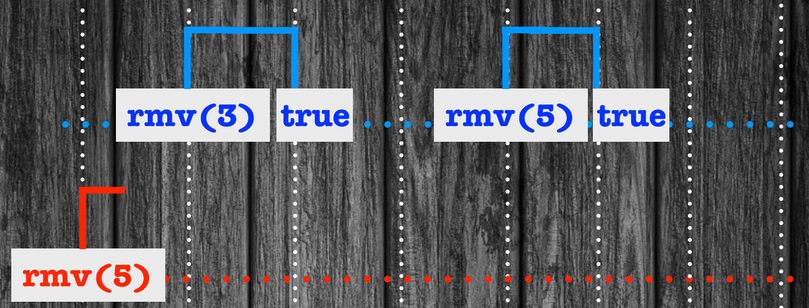
- no logical deletion
- only physical deletion





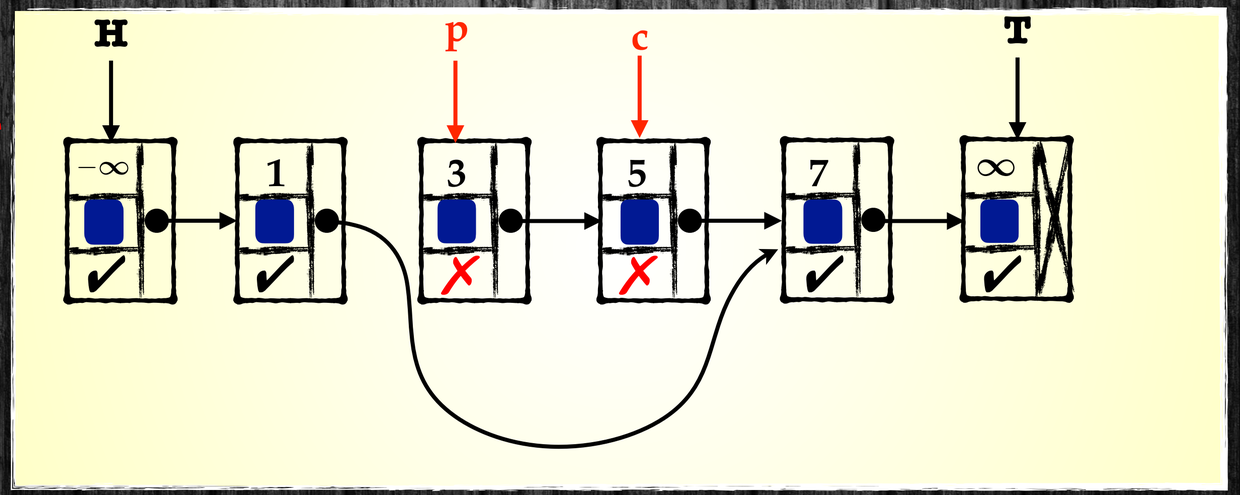
Scenario 1

- no logical deletion
- only physical deletion



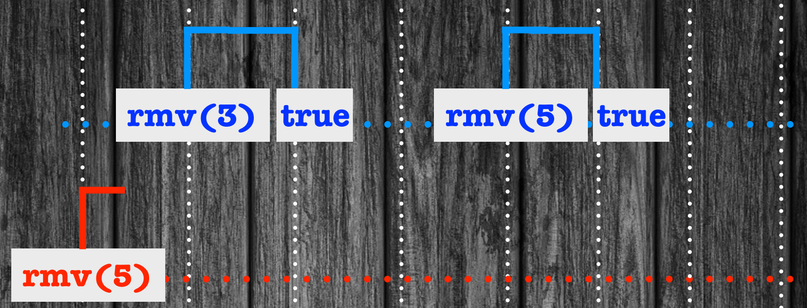
rmv(5)

```
rmv(e):
local p, c, n, r
1  (p,c) := locate(e);
2  if (c.val = e)
3
4     n := c.next;
5     p.next := n;
6     r := true;
7  else r := false;
8  unlock(p);
9  unlock(c);
10 return r;
```



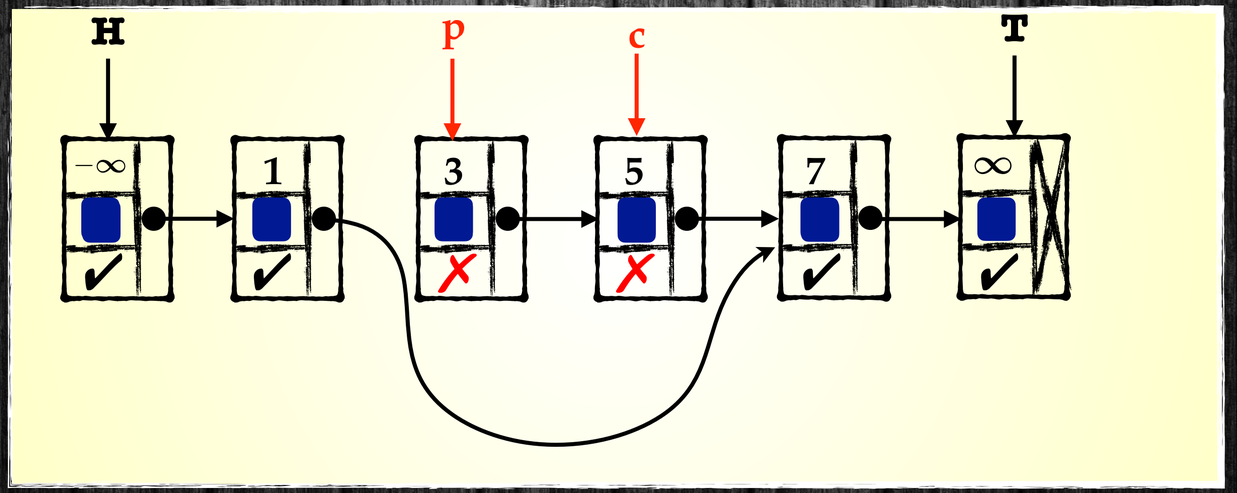
Scenario 1

- no logical deletion
- only physical deletion



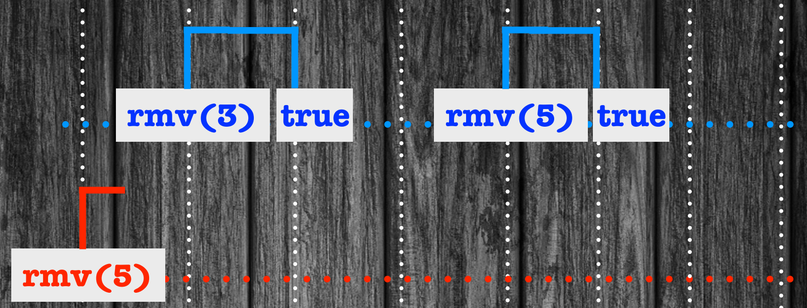
rmv(5)

```
rmv(e):
local p, c, n, r
1  (p,c) := locate(e);
2  if (c.val = e)
3
4     n := c.next;
5     p.next := n;
6     r := true;
7  else r := false;
8  unlock(p);
9  unlock(c);
10 return r;
```



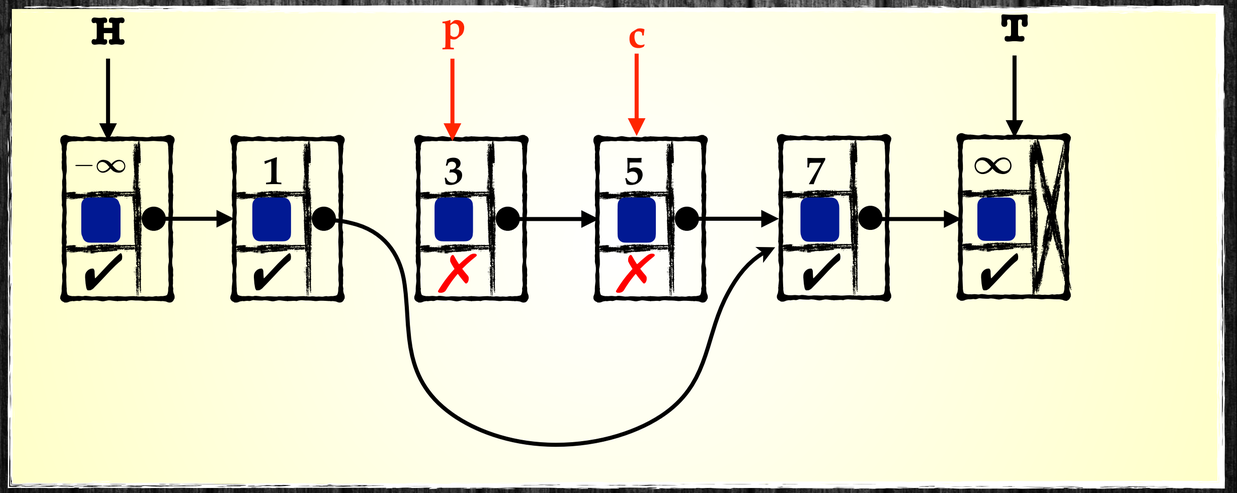
Scenario 1

- no logical deletion
- only physical deletion



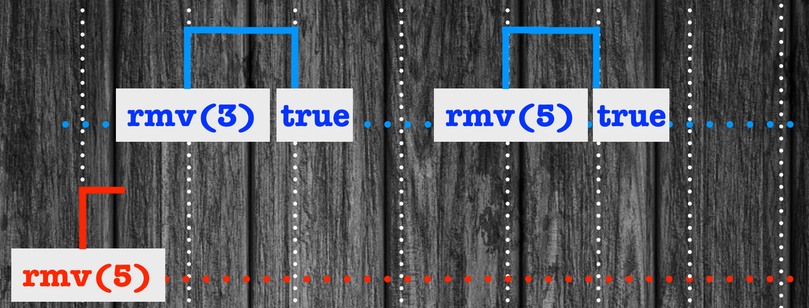
rmv(5)

```
rmv(e):
local p, c, n, r
1  (p,c) := locate(e);
2  if (c.val = e)
3
4     n := c.next;
5     p.next := n;
6     r := true;
7  else r := false;
8  unlock(p);
9  unlock(c);
10 return r;
```



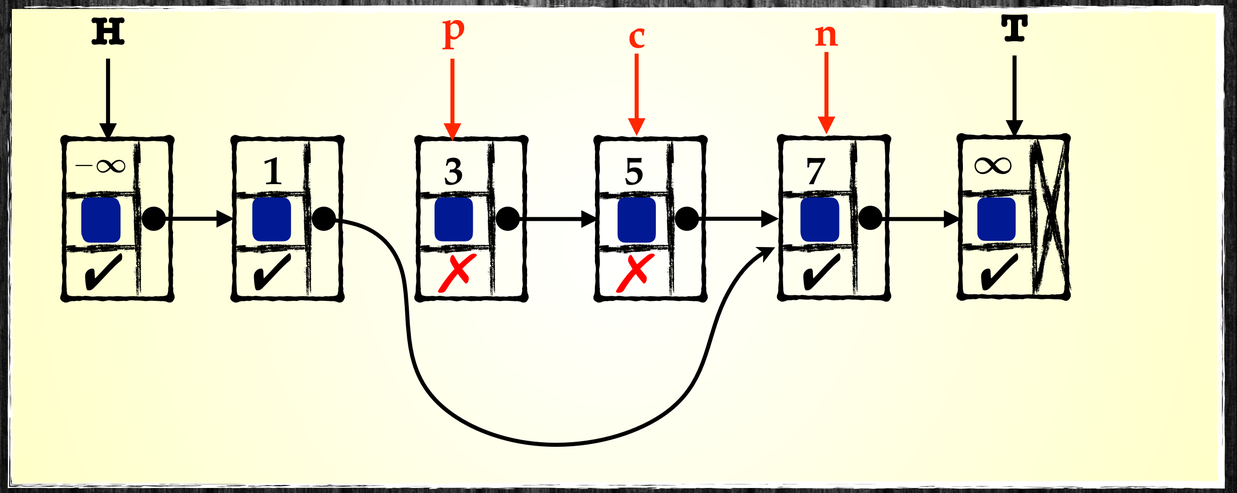
Scenario 1

- no logical deletion
- only physical deletion



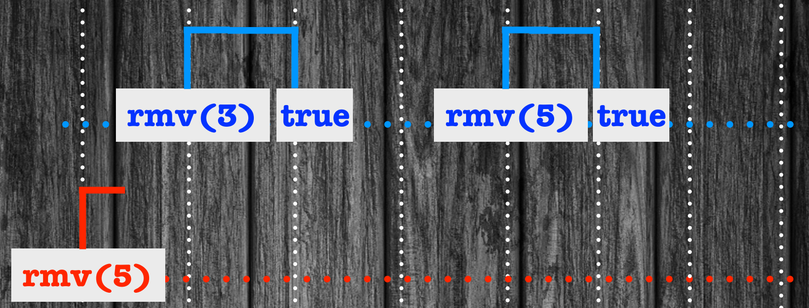
rmv(5)

```
rmv(e):
local p, c, n, r
1  (p,c) := locate(e);
2  if (c.val = e)
3
4    n := c.next;
5    p.next := n;
6    r := true;
7  else r := false;
8  unlock(p);
9  unlock(c);
10 return r;
```



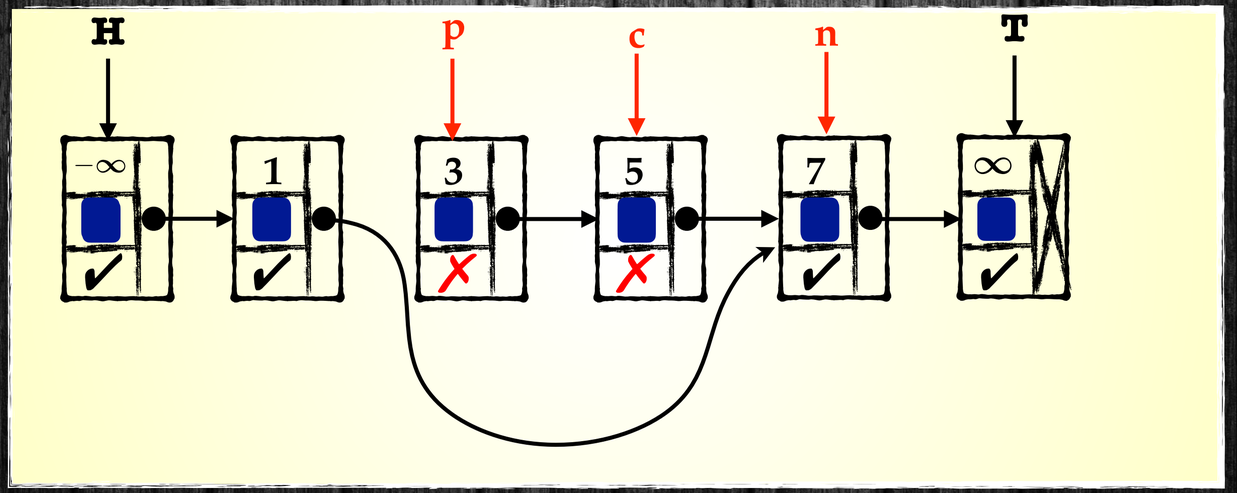
Scenario 1

- no logical deletion
- only physical deletion



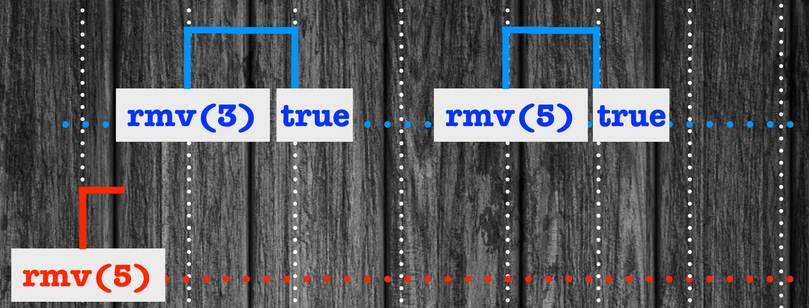
rmv(5)

```
rmv(e):
local p, c, n, r
1  (p,c) := locate(e);
2  if (c.val = e)
3
4     n := c.next;
5     p.next := n;
6     r := true;
7  else r := false;
8  unlock(p);
9  unlock(c);
10 return r;
```



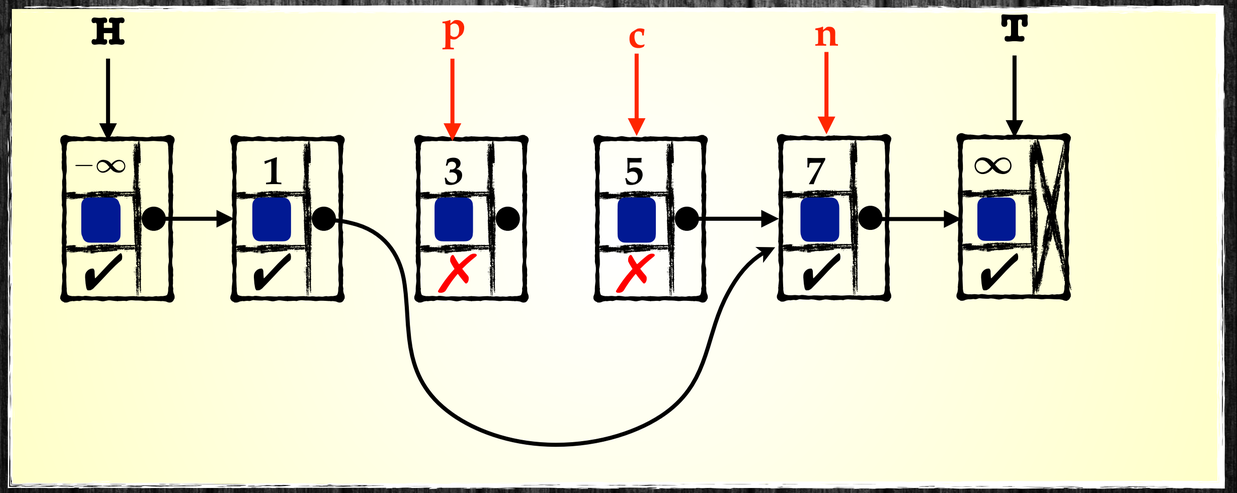
Scenario 1

- no logical deletion
- only physical deletion



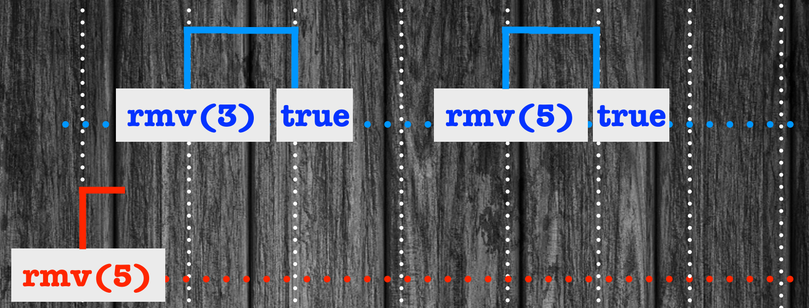
rmv(5)

```
rmv(e):
local p, c, n, r
1  (p,c) := locate(e);
2  if (c.val = e)
3
4     n := c.next;
5     p.next := n;
6     r := true;
7  else r := false;
8  unlock(p);
9  unlock(c);
10 return r;
```



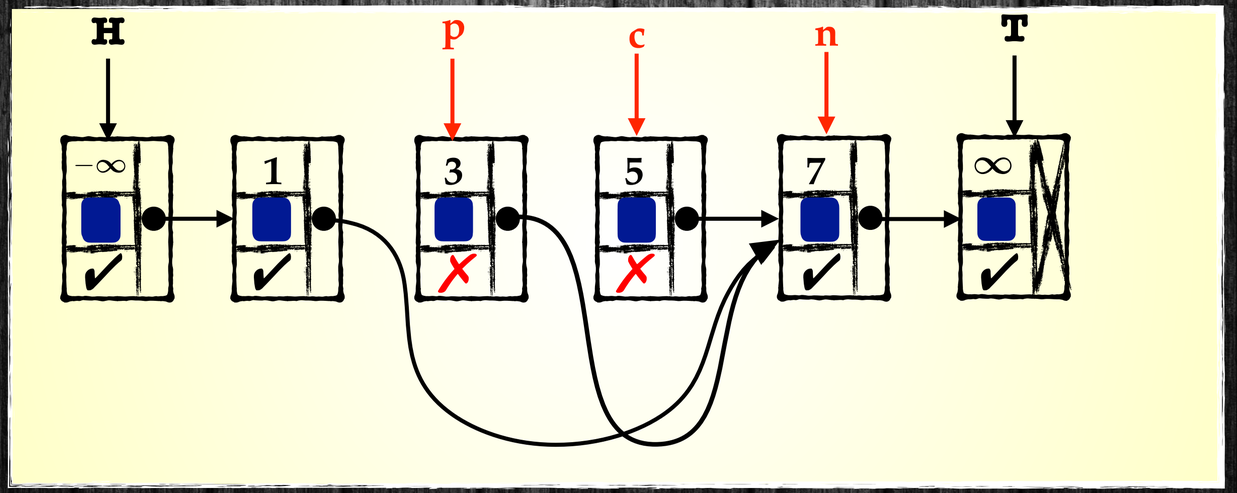
Scenario 1

- no logical deletion
- only physical deletion



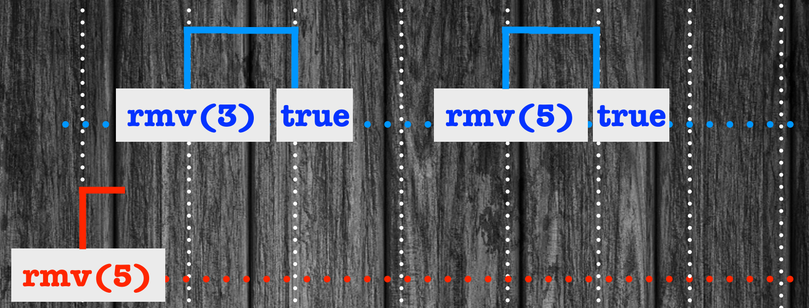
rmv(5)

```
rmv(e):
local p, c, n, r
1  (p,c) := locate(e);
2  if (c.val = e)
3
4     n := c.next;
5     p.next := n;
6     r := true;
7  else r := false;
8  unlock(p);
9  unlock(c);
10 return r;
```



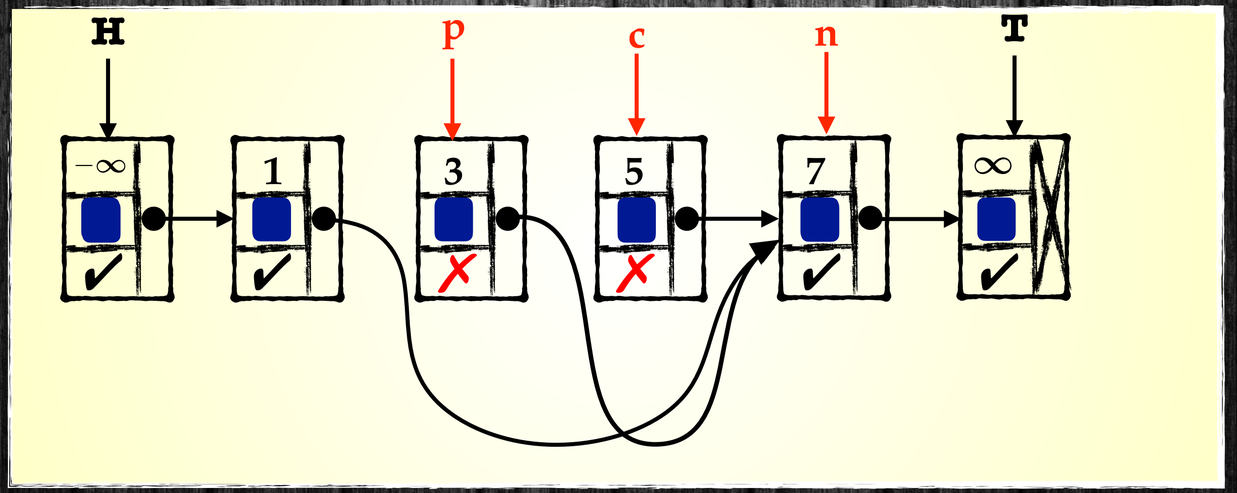
Scenario 1

- no logical deletion
- only physical deletion



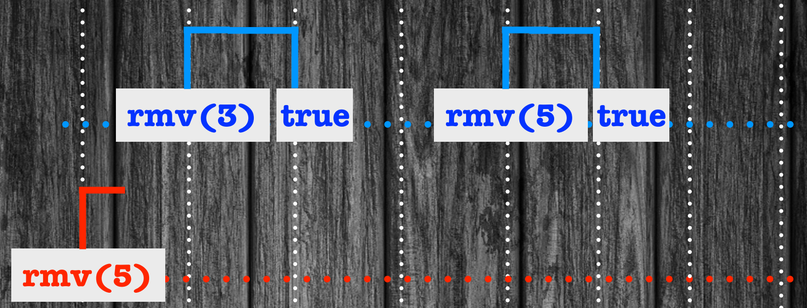
rmv(5)

```
rmv(e):
local p, c, n, r
1  (p,c) := locate(e);
2  if (c.val = e)
3
4     n := c.next;
5     p.next := n;
6     r := true;
7  else r := false;
8  unlock(p);
9  unlock(c);
10 return r;
```



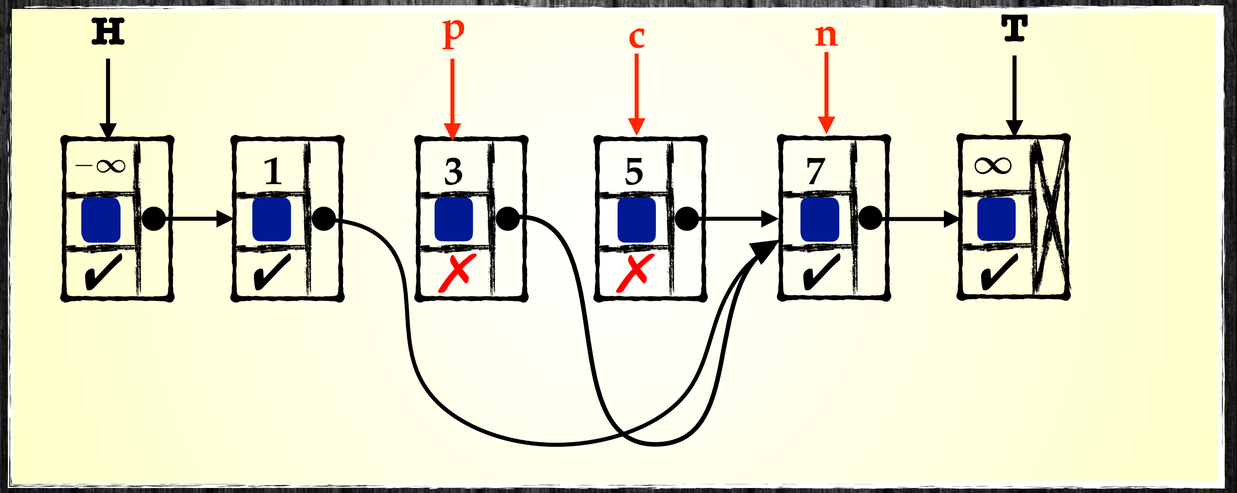
Scenario 1

- no logical deletion
- only physical deletion



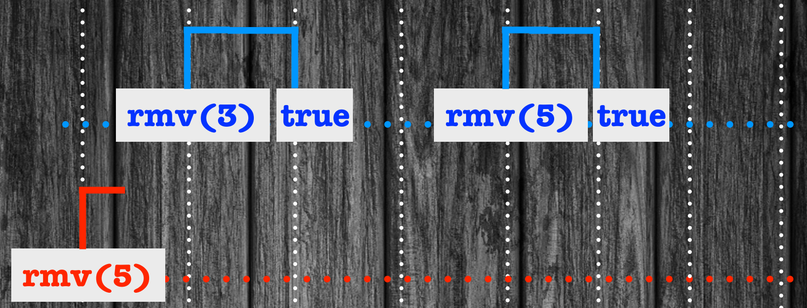
rmv(5)

```
rmv(e):
local p, c, n, r
1  (p,c) := locate(e);
2  if (c.val = e)
3
4     n := c.next;
5     p.next := n;
6     r := true;
7  else r := false;
8  unlock(p);
9  unlock(c);
10 return r;
```



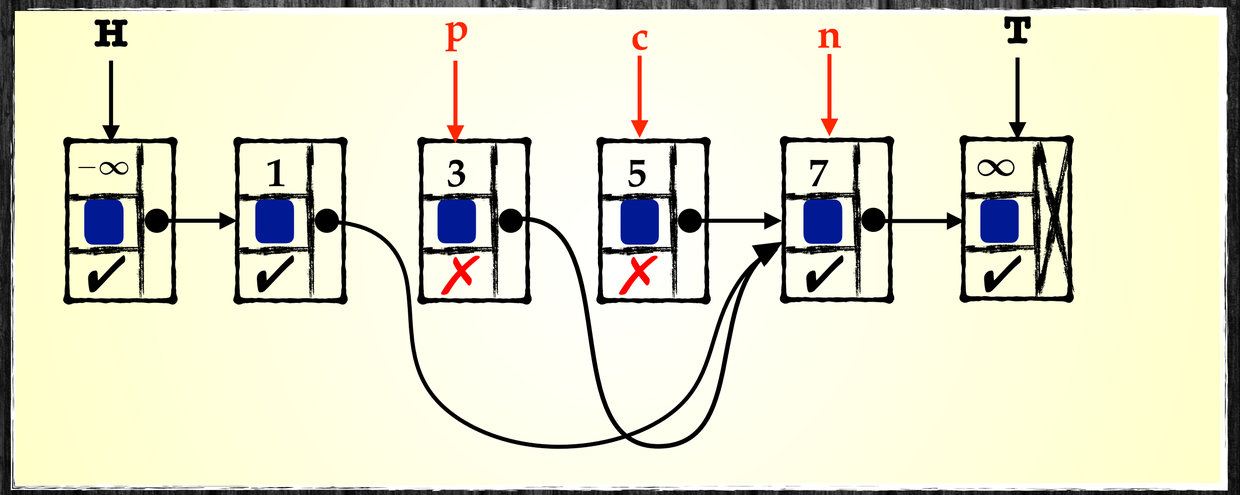
Scenario 1

- no logical deletion
- only physical deletion



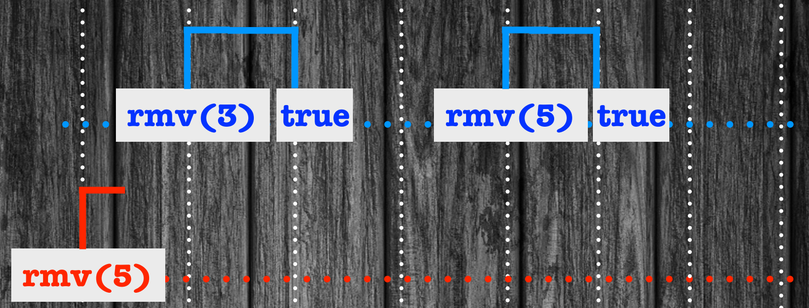
rmv(5)

```
rmv(e):
local p, c, n, r
1  (p,c) := locate(e);
2  if (c.val = e)
3
4     n := c.next;
5     p.next := n;
6     r := true;
7  else r := false;
8  unlock(p);
9  unlock(c);
10 return r;
```



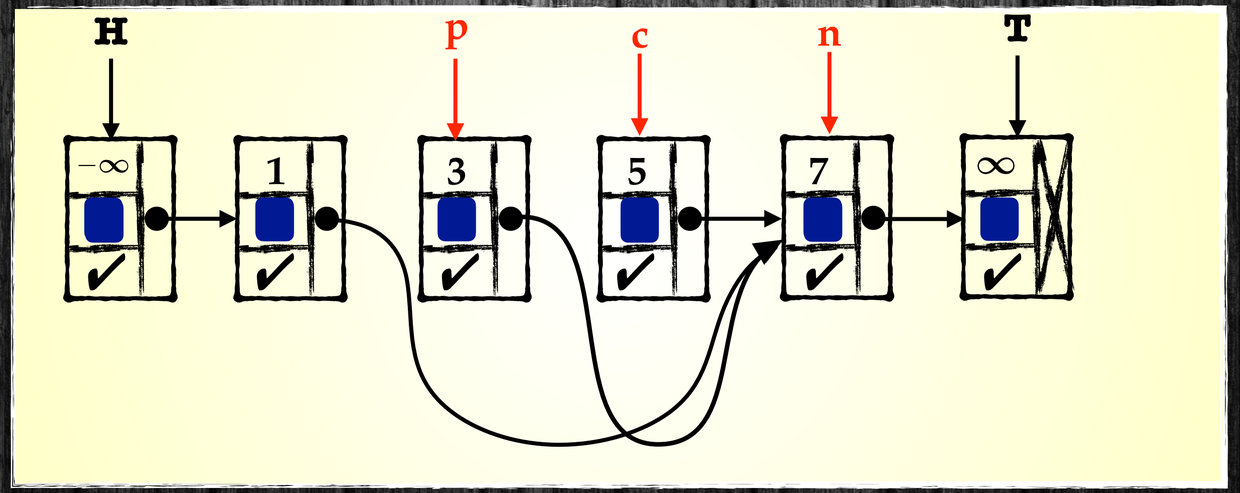
Scenario 1

- no logical deletion
- only physical deletion



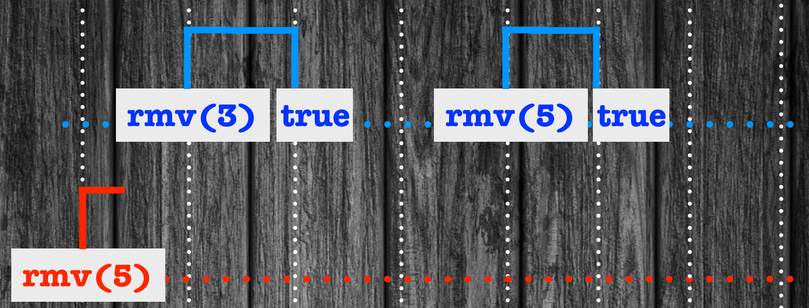
rmv(5)

```
rmv(e):
local p, c, n, r
1  (p,c) := locate(e);
2  if (c.val = e)
3
4     n := c.next;
5     p.next := n;
6     r := true;
7  else r := false;
8  unlock(p);
9  unlock(c);
10 return r;
```



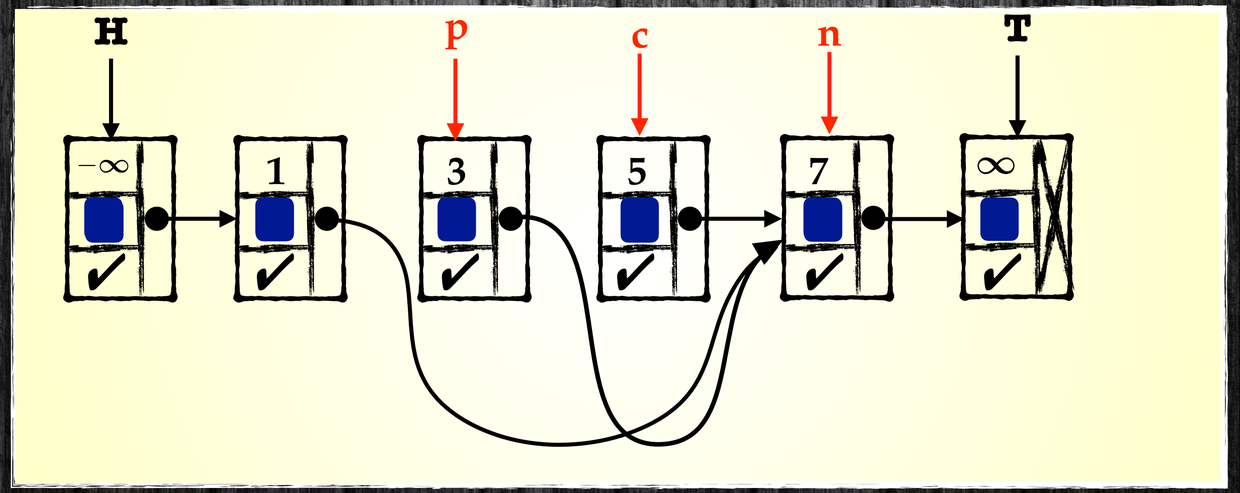
Scenario 1

- no logical deletion
- only physical deletion



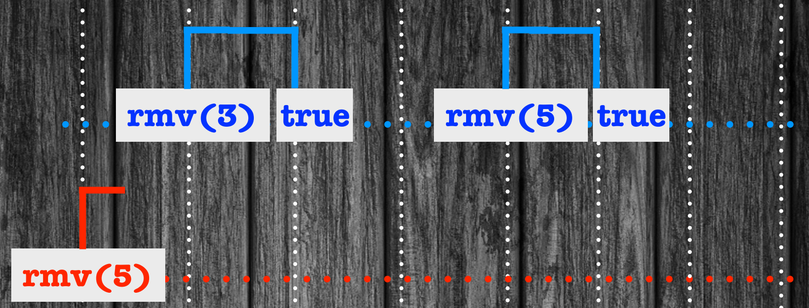
rmv(5)

```
rmv(e):
local p, c, n, r
1  (p,c) := locate(e);
2  if (c.val = e)
3
4     n := c.next;
5     p.next := n;
6     r := true;
7  else r := false;
8  unlock(p);
9  unlock(c);
10 return r;
```



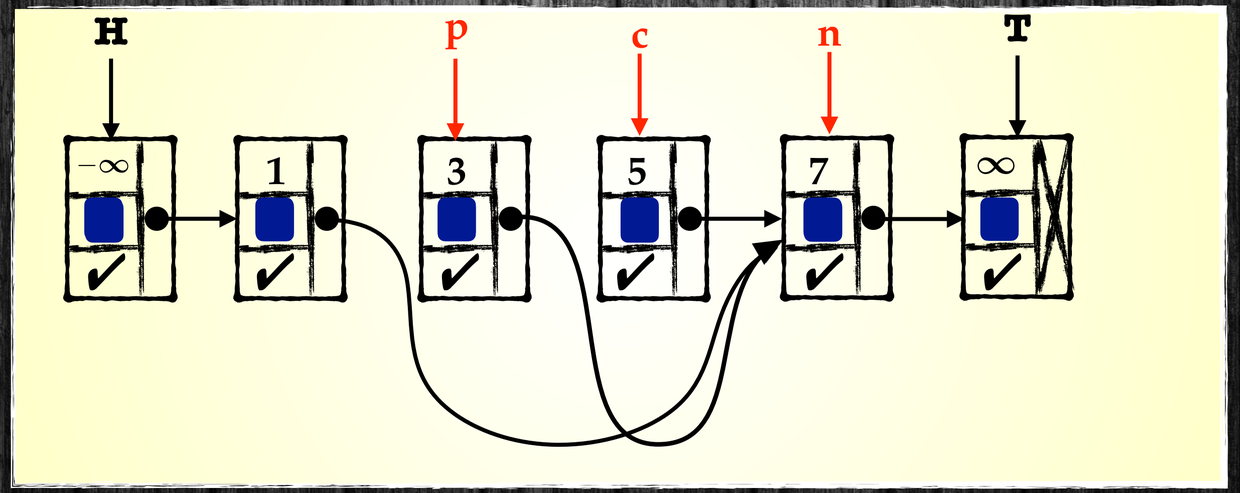
Scenario 1

- no logical deletion
- only physical deletion



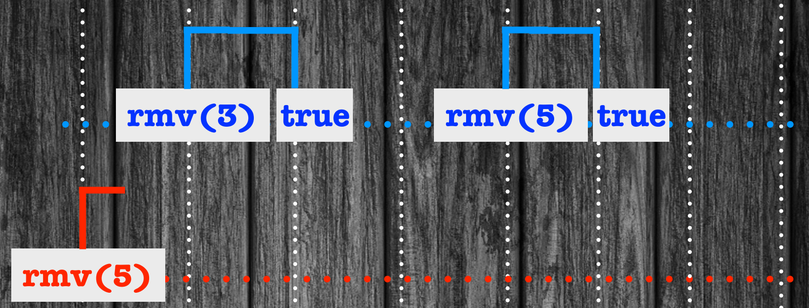
rmv(5)

```
rmv(e):
local p, c, n, r
1  (p,c) := locate(e);
2  if (c.val = e)
3
4     n := c.next;
5     p.next := n;
6     r := true;
7  else r := false;
8  unlock(p);
9  unlock(c);
10 return r;
```



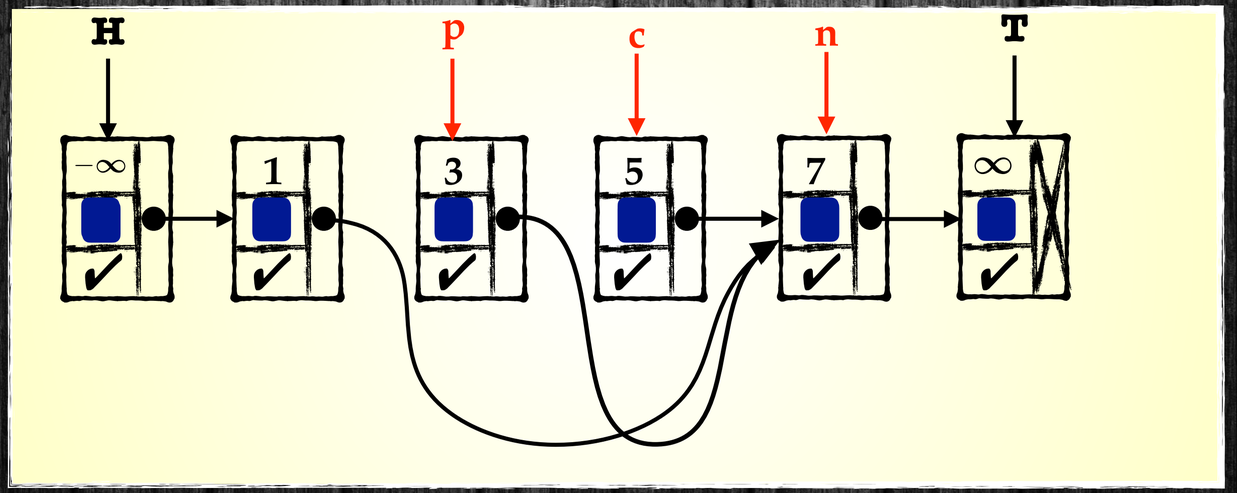
Scenario 1

- no logical deletion
- only physical deletion



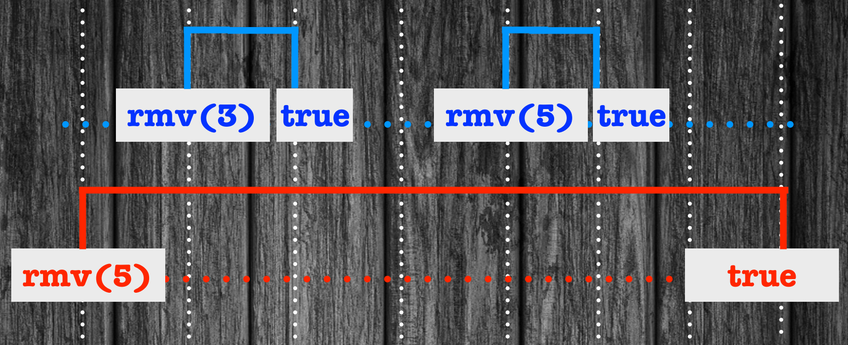
rmv(5)

```
rmv(e):
local p, c, n, r
1  (p,c) := locate(e);
2  if (c.val = e)
3
4     n := c.next;
5     p.next := n;
6     r := true;
7  else r := false;
8  unlock(p);
9  unlock(c);
10 return r;
```



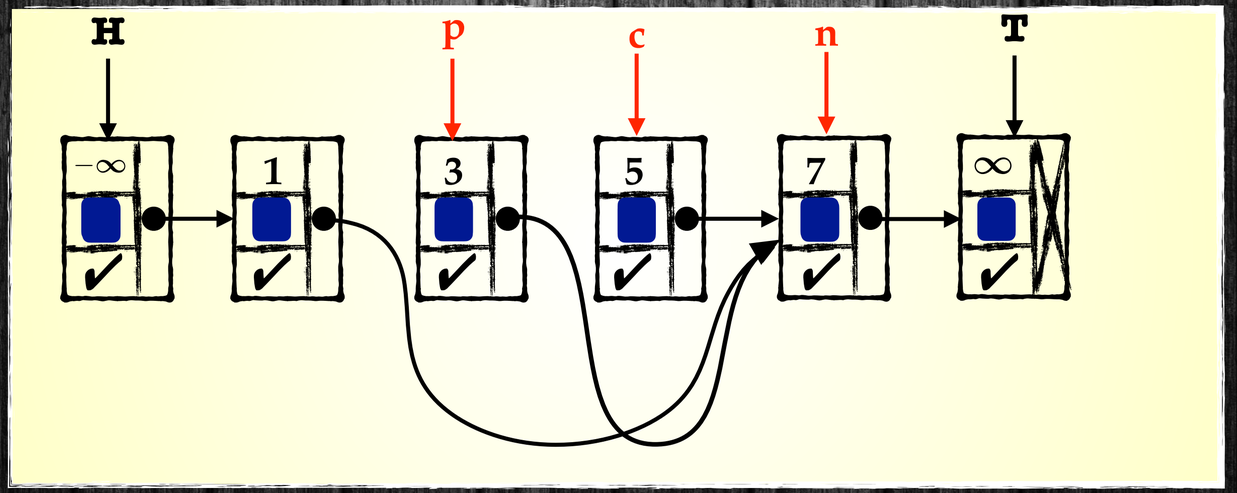
Scenario 1

- no logical deletion
- only physical deletion



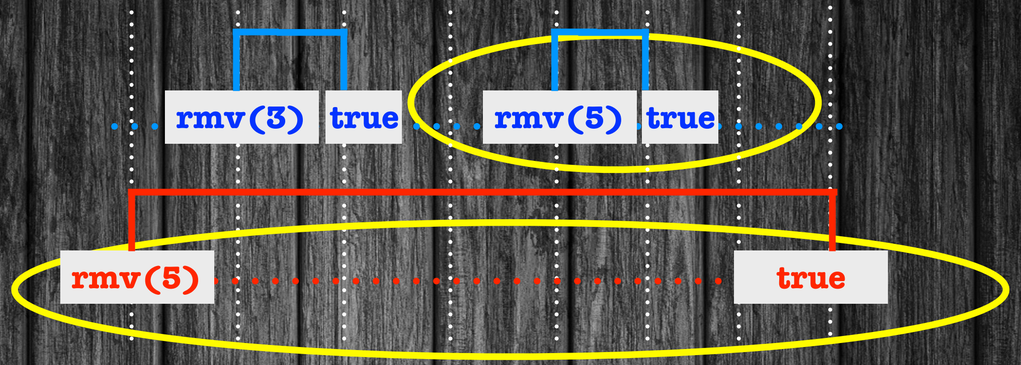
rmv(5)

```
rmv(e):
local p, c, n, r
1  (p,c) := locate(e);
2  if (c.val = e)
3
4     n := c.next;
5     p.next := n;
6     r := true;
7  else r := false;
8  unlock(p);
9  unlock(c);
10 return r;
```



Scenario 1

- no logical deletion
- only physical deletion



Sets | Lazy Synchronization

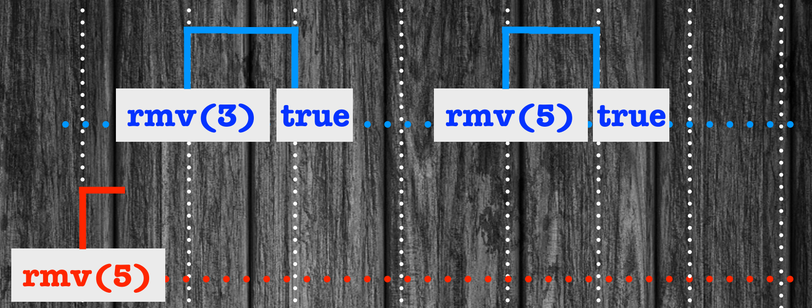
```
locate(e):
local p, c
1  while (true)
2     p := head;
3     c := p.next;
4     while (c.val < e)
5        p := c;
6        c := c.next
7     lock(p); lock(c);
8     if (!p.mark && !c.mark && p.next=c)
9        return(p,c);
10    else
11       unlock(p);
12       unlock(c);
```

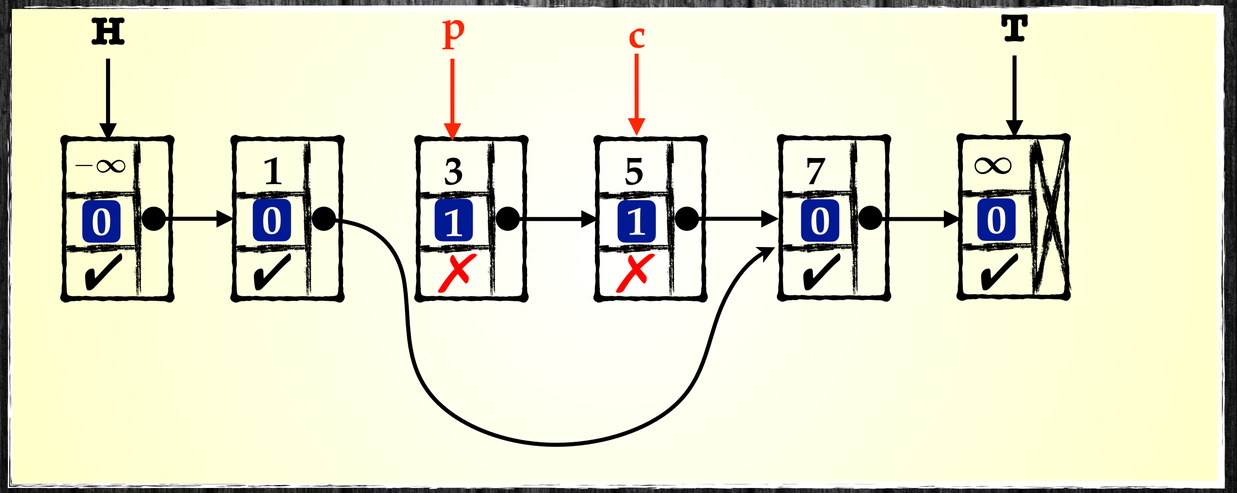
```
rmv(e):
local p, c, n, r
1  (p,c) := locate(e);
2  if (c.val = e)
3     c.mark := true
4     n := c.next;
5     p.next := n;
6     r := true;
7  else r := false;
8  unlock(p);
9  unlock(c);
10 return r;
```

rmv(5)

Scenario 1

- no logical deletion
- only physical deletion



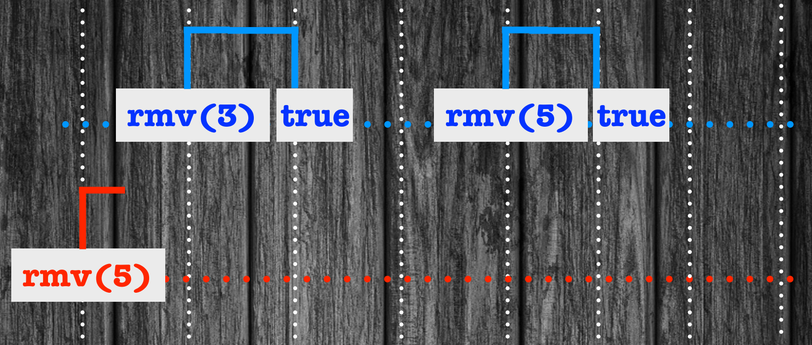


Sets | Lazy Synchronization

Scenario 1

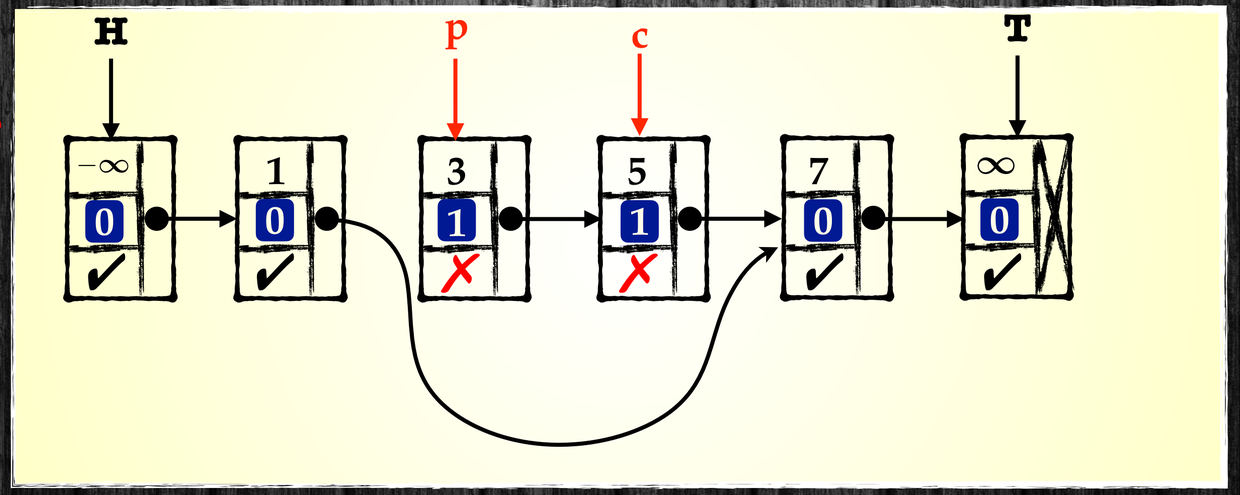
- no logical deletion
- only physical deletion

```
locate(e):
local p, c
1  while (true)
2     p := head;
3     c := p.next;
4     while (c.val < e)
5        p := c;
6        c := c.next
7     lock(p); lock(c);
8     if (!p.mark && !c.mark && p.next=c)
9        return(p,c);
10    else
11       unlock(p);
12       unlock(c);
```



rmv(5)

```
rmv(e):
local p, c, n, r
1  (p,c) := locate(e);
2  if (c.val = e)
3     c.mark := true
4     n := c.next;
5     p.next := n;
6     r := true;
7  else r := false;
8  unlock(p);
9  unlock(c);
10 return r;
```



Sets | Lazy Synchronization

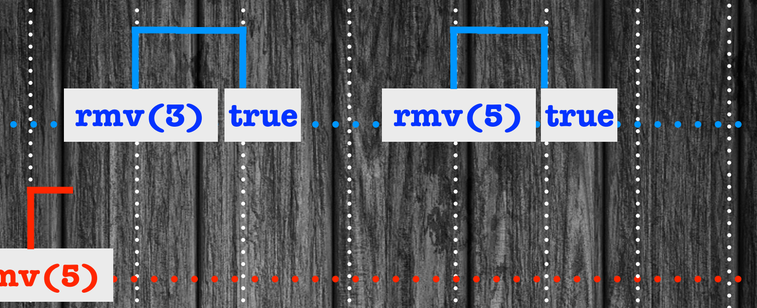
```
locate(e):
local p, c
1  while (true)
2     p := head;
3     c := p.next;
4     while (c.val < e)
5        p := c;
6        c := c.next
7     lock(p); lock(c);
8     if (!p.mark && !c.mark && p.next=c)
9        return(p,c);
10    else
11       unlock(p);
12       unlock(c);
```

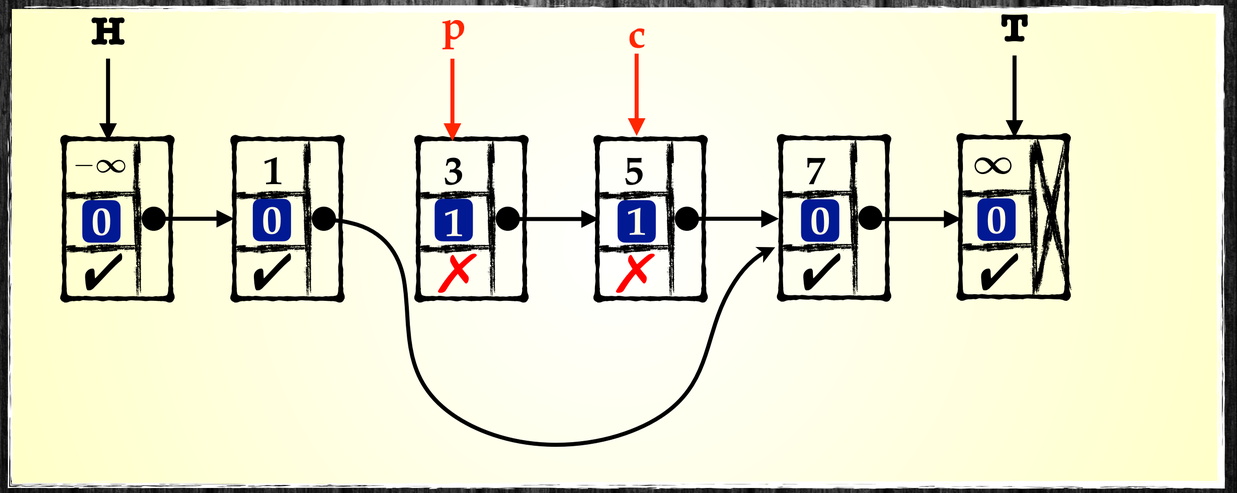
```
rmv(e):
local p, c, n, r
1  (p,c) := locate(e);
2  if (c.val = e)
3     c.mark := true
4     n := c.next;
5     p.next := n;
6     r := true;
7  else r := false;
8  unlock(p);
9  unlock(c);
10 return r;
```

rmv(5)

Scenario 1

- no logical deletion
- only physical deletion





Sets | Lazy Synchronization

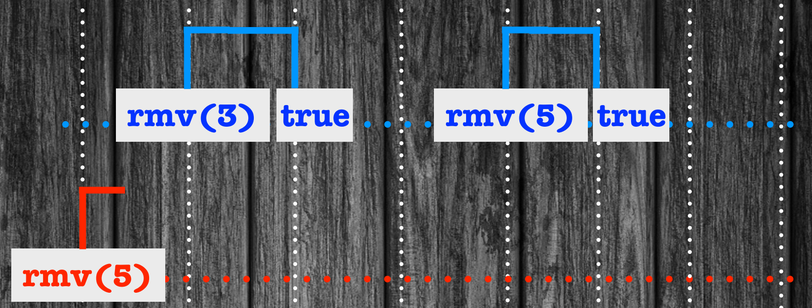
```
locate(e):
local p, c
1  while (true)
2     p := head;
3     c := p.next;
4     while (c.val < e)
5        p := c;
6        c := c.next
7     lock(p); lock(c);
8     if (!p.mark && !c.mark && p.next=c)
9        return(p,c);
10    else
11       unlock(p);
12       unlock(c);
```

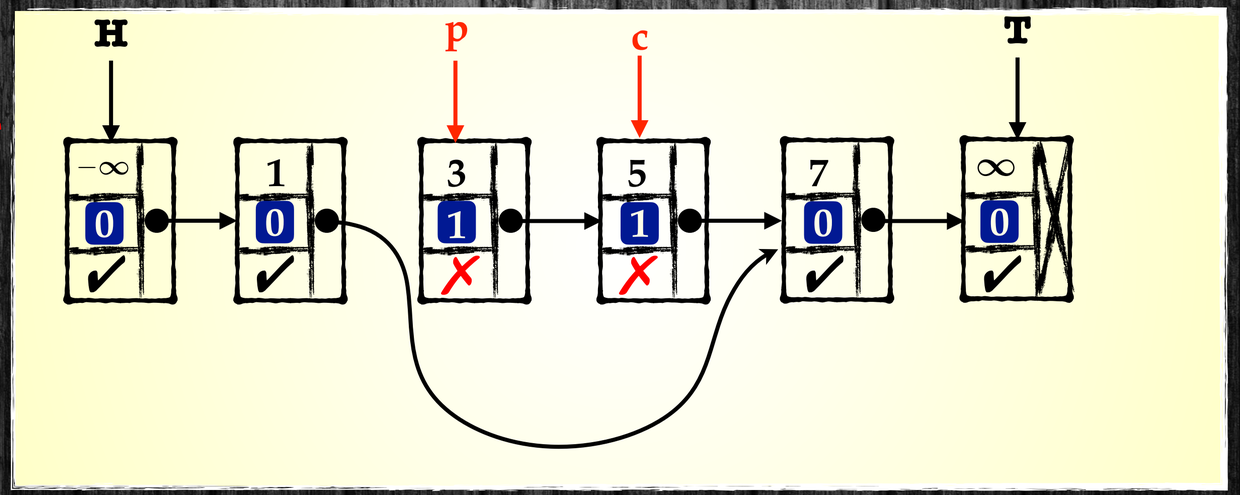
```
rmv(e):
local p, c, n, r
1  (p,c) := locate(e);
2  if (c.val = e)
3     c.mark := true
4     n := c.next;
5     p.next := n;
6     r := true;
7  else r := false;
8  unlock(p);
9  unlock(c);
10 return r;
```

rmv(5)

Scenario 1

- no logical deletion
- only physical deletion





Sets | Lazy Synchronization

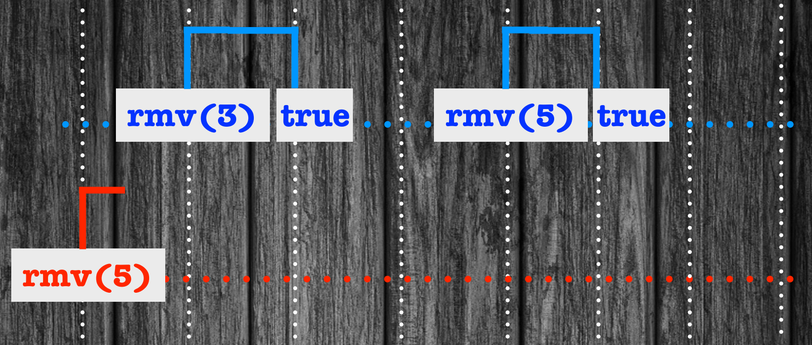
```
locate(e):
local p, c
1  while (true)
2     p := head;
3     c := p.next;
4     while (c.val < e)
5        p := c;
6        c := c.next
7     lock(p); lock(c);
8     if (!p.mark && !c.mark && p.next=c)
9        return(p,c);
10    else
11       unlock(p);
12       unlock(c);
```

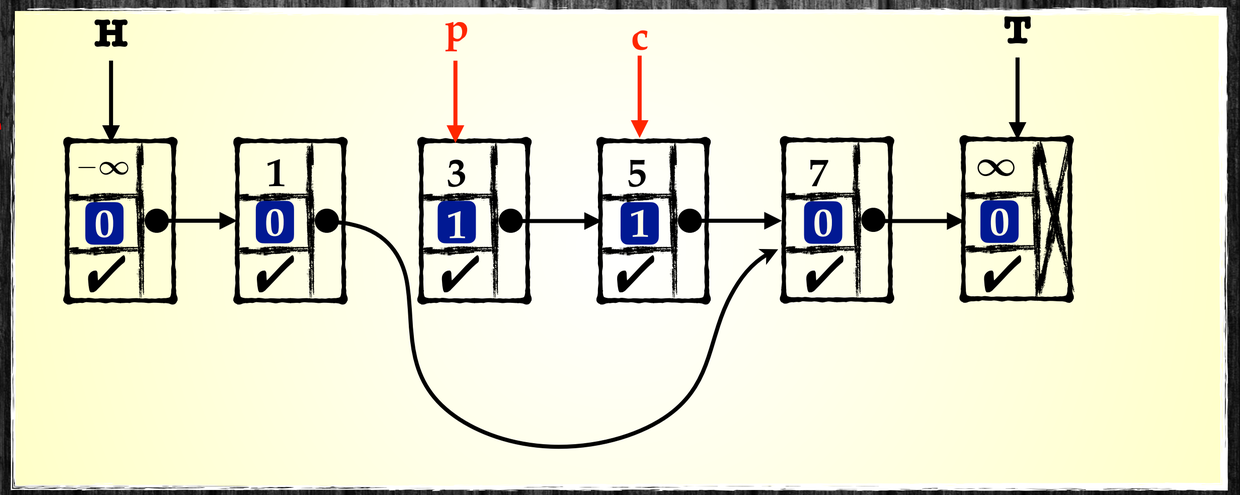
```
rmv(e):
local p, c, n, r
1  (p,c) := locate(e);
2  if (c.val = e)
3     c.mark := true
4     n := c.next;
5     p.next := n;
6     r := true;
7  else r := false;
8  unlock(p);
9  unlock(c);
10 return r;
```

rmv(5)

Scenario 1

- no logical deletion
- only physical deletion





Sets | Lazy Synchronization

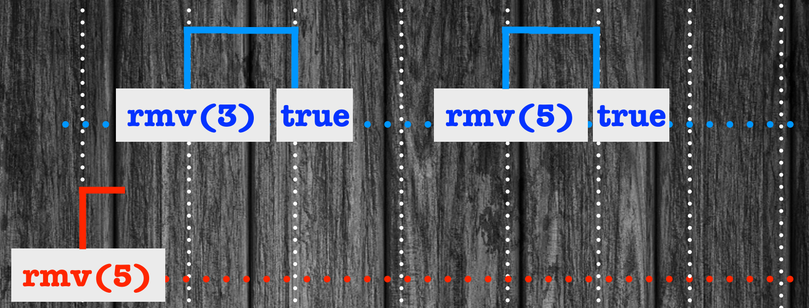
```
locate(e):
local p, c
1  while (true)
2     p := head;
3     c := p.next;
4     while (c.val < e)
5        p := c;
6        c := c.next
7     lock(p); lock(c);
8     if (!p.mark && !c.mark && p.next=c)
9        return(p,c);
10    else
11       unlock(p);
12       unlock(c);
```

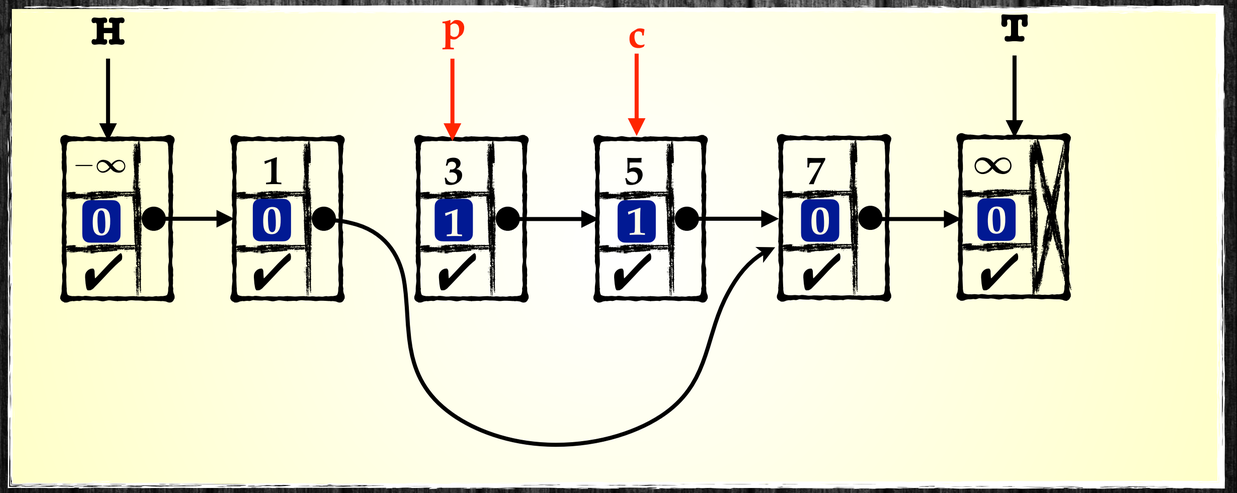
```
rmv(e):
local p, c, n, r
1  (p,c) := locate(e);
2  if (c.val = e)
3     c.mark := true
4     n := c.next;
5     p.next := n;
6     r := true;
7  else r := false;
8  unlock(p);
9  unlock(c);
10 return r;
```

rmv(5)

Scenario 1

- no logical deletion
- only physical deletion



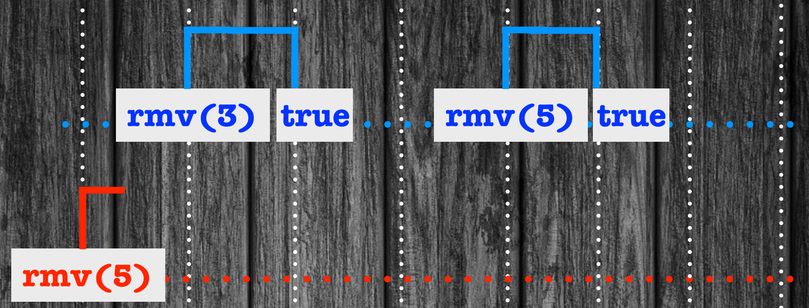


Sets | Lazy Synchronization

```
locate(e):
local p, c
1  while (true)
2     p := head;
3     c := p.next;
4     while (c.val < e)
5        p := c;
6        c := c.next
7     lock(p); lock(c);
8     if (!p.mark && !c.mark && p.next=c)
9        return(p,c);
10    else
11       unlock(p);
12       unlock(c);
```

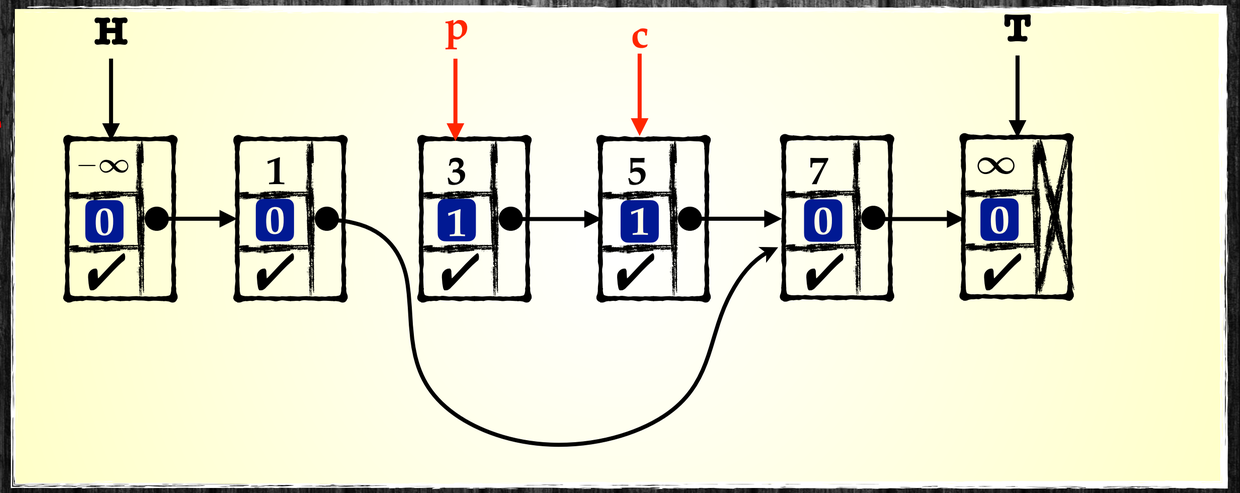
Scenario 1

- no logical deletion
- only physical deletion



rmv(5)

```
rmv(e):
local p, c, n, r
1  (p,c) := locate(e);
2  if (c.val = e)
3     c.mark := true
4     n := c.next;
5     p.next := n;
6     r := true;
7  else r := false;
8  unlock(p);
9  unlock(c);
10 return r;
```



Sets | Lazy Synchronization

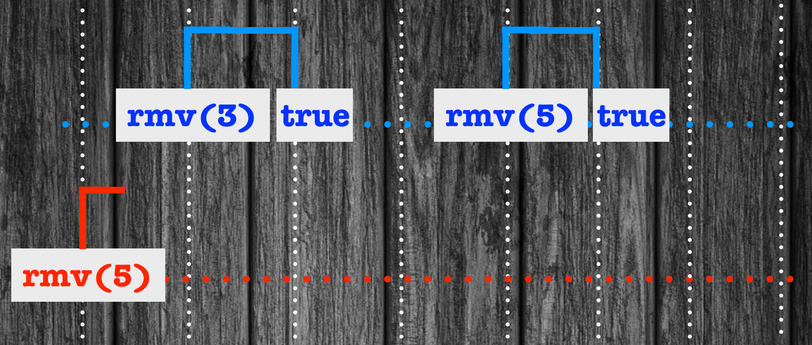
```
locate(e):
local p, c
1  while (true)
2     p := head;
3     c := p.next;
4     while (c.val < e)
5        p := c;
6        c := c.next
7     lock(p); lock(c);
8     if (!p.mark && !c.mark && p.next=c)
9        return(p,c);
10    else
11       unlock(p);
12       unlock(c);
```

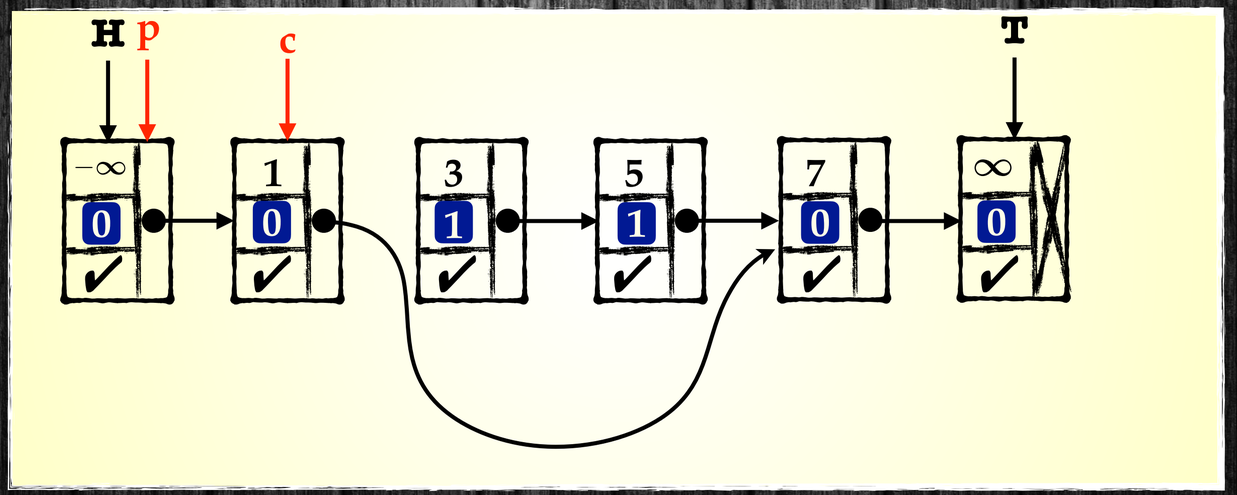
```
rmv(e):
local p, c, n, r
1  (p,c) := locate(e);
2  if (c.val = e)
3     c.mark := true
4     n := c.next;
5     p.next := n;
6     r := true;
7  else r := false;
8  unlock(p);
9  unlock(c);
10 return r;
```

rmv(5)

Scenario 1

- no logical deletion
- only physical deletion





Sets | Lazy Synchronization

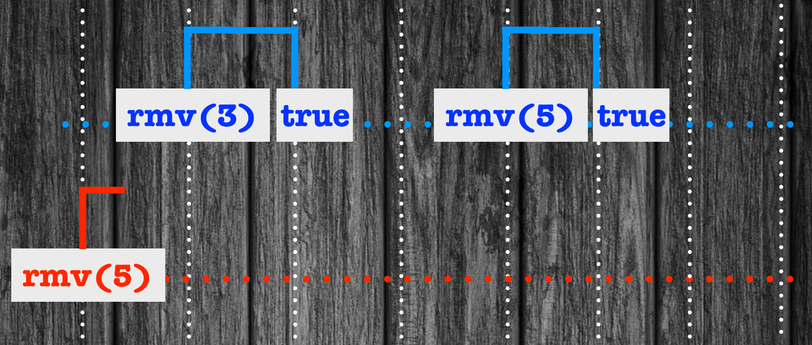
```
locate(e):
local p, c
1  while (true)
2     p := head;
3     c := p.next;
4     while (c.val < e)
5        p := c;
6        c := c.next
7     lock(p); lock(c);
8     if (!p.mark && !c.mark && p.next=c)
9        return(p,c);
10    else
11       unlock(p);
12       unlock(c);
```

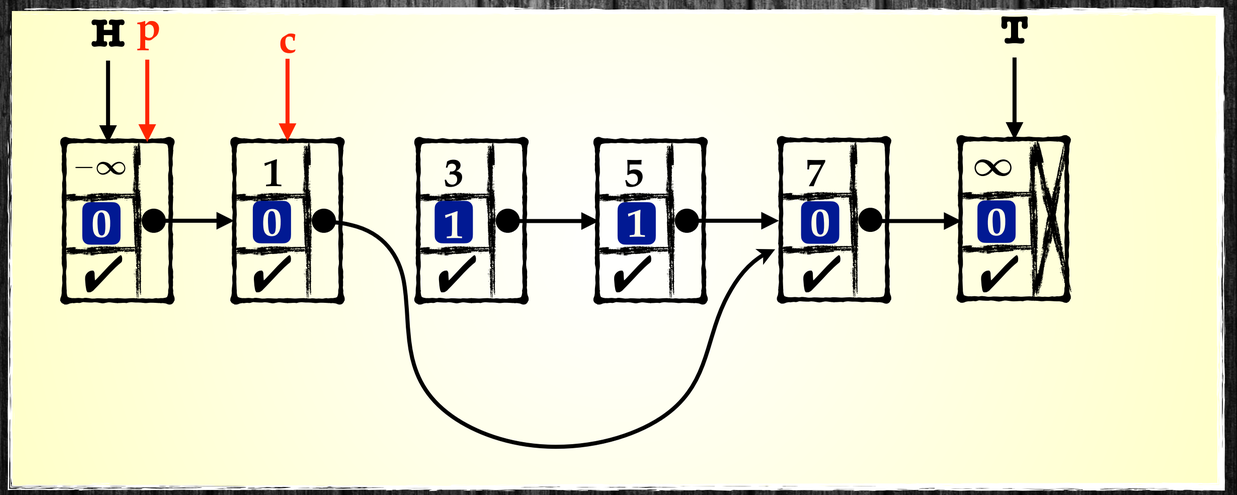
```
rmv(e):
local p, c, n, r
1  (p,c) := locate(e);
2  if (c.val = e)
3     c.mark := true
4     n := c.next;
5     p.next := n;
6     r := true;
7  else r := false;
8  unlock(p);
9  unlock(c);
10 return r;
```

rmv(5)

Scenario 1

- no logical deletion
- only physical deletion





Sets | Lazy Synchronization

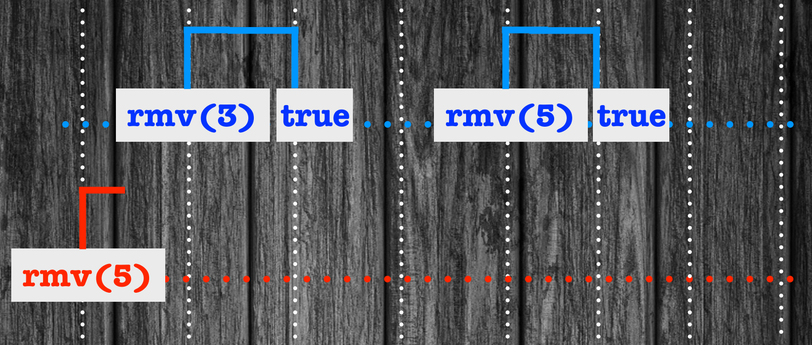
```
locate(e):
local p, c
1  while (true)
2     p := head;
3     c := p.next;
4     while (c.val < e)
5        p := c;
6        c := c.next
7     lock(p); lock(c);
8     if (!p.mark && !c.mark && p.next=c)
9        return(p,c);
10    else
11       unlock(p);
12       unlock(c);
```

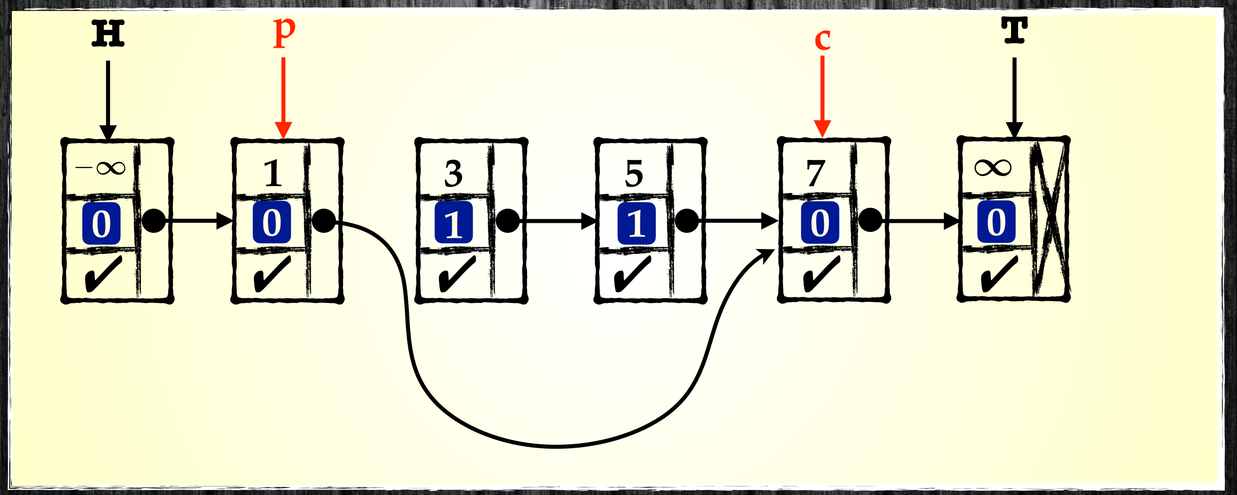
```
rmv(e):
local p, c, n, r
1  (p,c) := locate(e);
2  if (c.val = e)
3     c.mark := true
4     n := c.next;
5     p.next := n;
6     r := true;
7  else r := false;
8  unlock(p);
9  unlock(c);
10 return r;
```

rmv(5)

Scenario 1

- no logical deletion
- only physical deletion





Sets | Lazy Synchronization

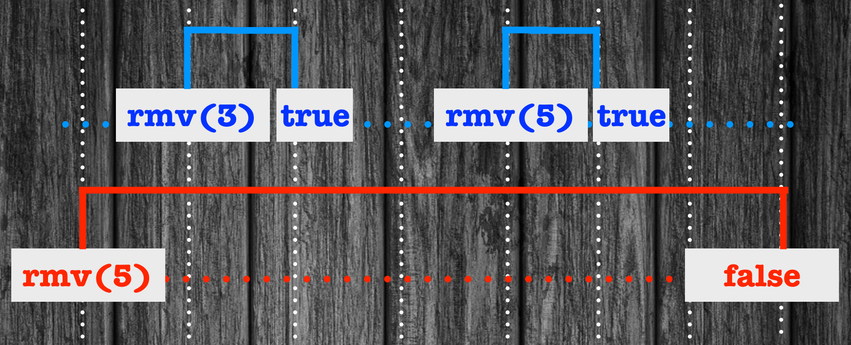
```
locate(e):
local p, c
1  while (true)
2     p := head;
3     c := p.next;
4     while (c.val < e)
5        p := c;
6        c := c.next
7     lock(p); lock(c);
8     if (!p.mark && !c.mark && p.next=c)
9        return(p,c);
10    else
11       unlock(p);
12       unlock(c);
```

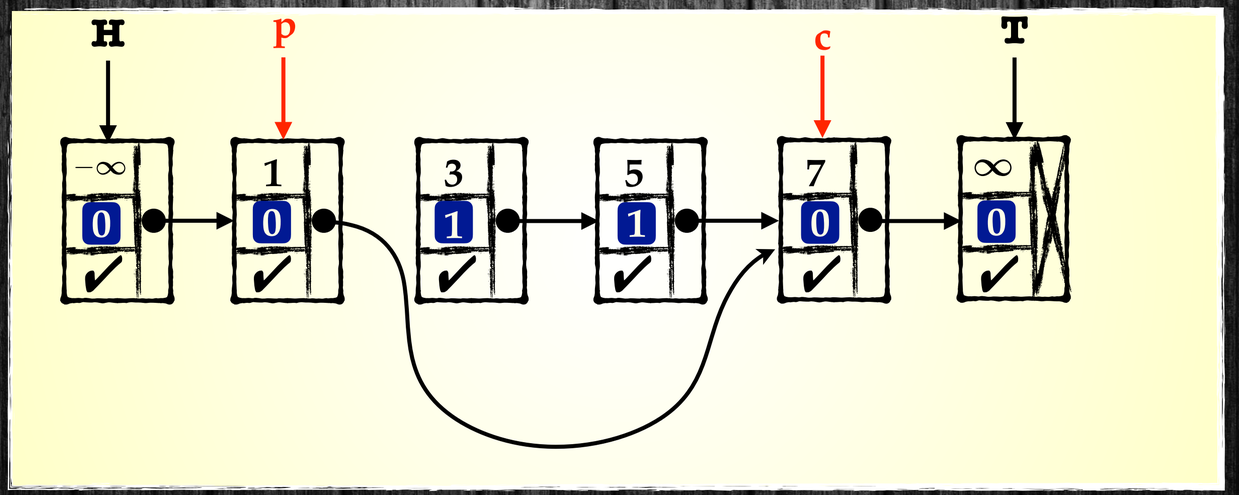
```
rmv(e):
local p, c, n, r
1  (p,c) := locate(e);
2  if (c.val = e)
3     c.mark := true
4     n := c.next;
5     p.next := n;
6     r := true;
7  else r := false;
8  unlock(p);
9  unlock(c);
10 return r;
```

rmv(5)

Scenario 1

- no logical deletion
- only physical deletion



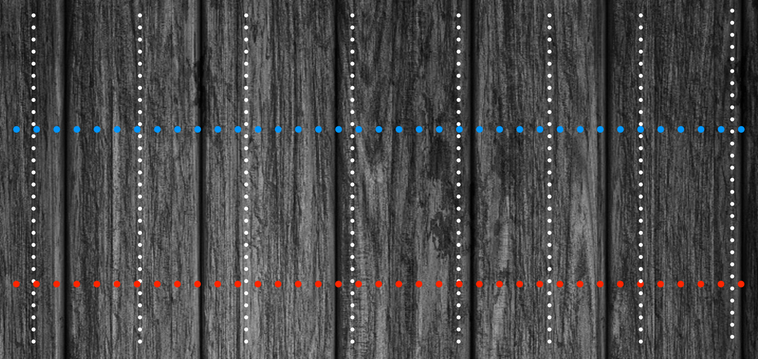


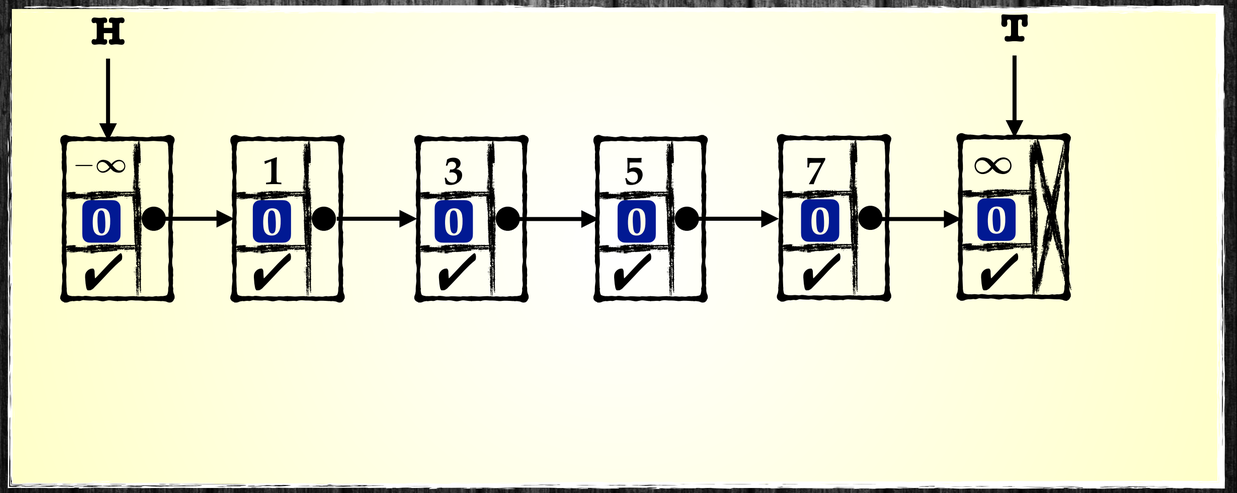
ctn(5)

```
ctn(e):
local c, b
1  c := Head;
2  while (c.val < e)
3     c := c.next
4  b := c.mark
5  if (!b && c.val = e)
6     return true;
7  else return false;
```

Scenario 2

- what if “mark” is first read
- and then set?



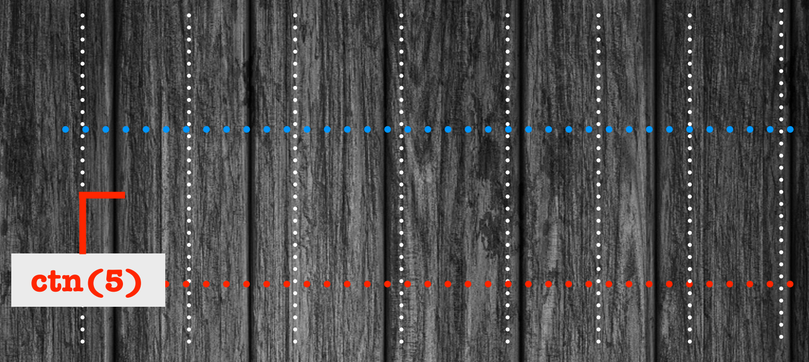


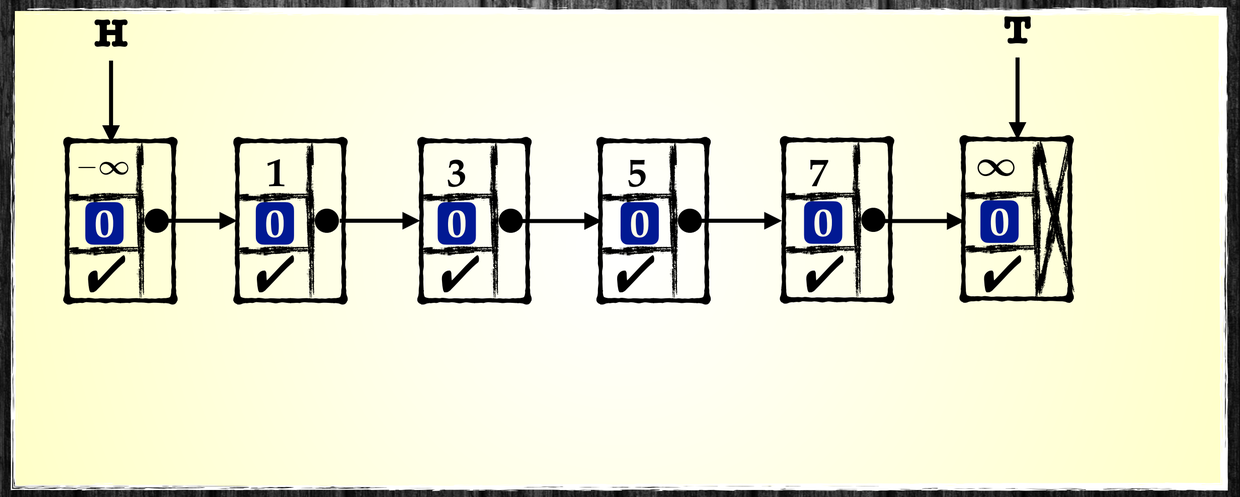
ctn(5)

```
ctn(e):
local c, b
1  c := Head;
2  while (c.val < e)
3     c := c.next
4  b := c.mark
5  if (!b && c.val = e)
6     return true;
7  else return false;
```

Scenario 2

- what if “mark” is first read
- and then set?



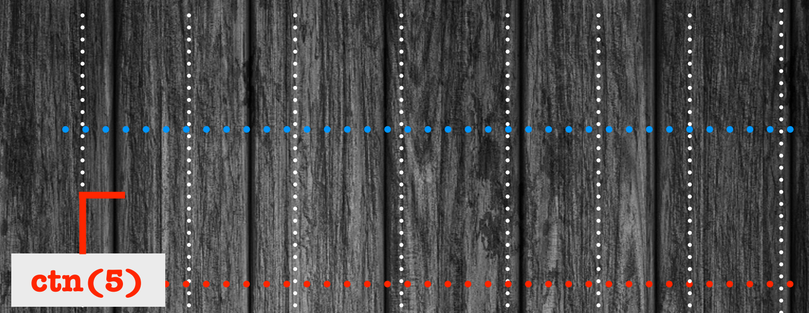


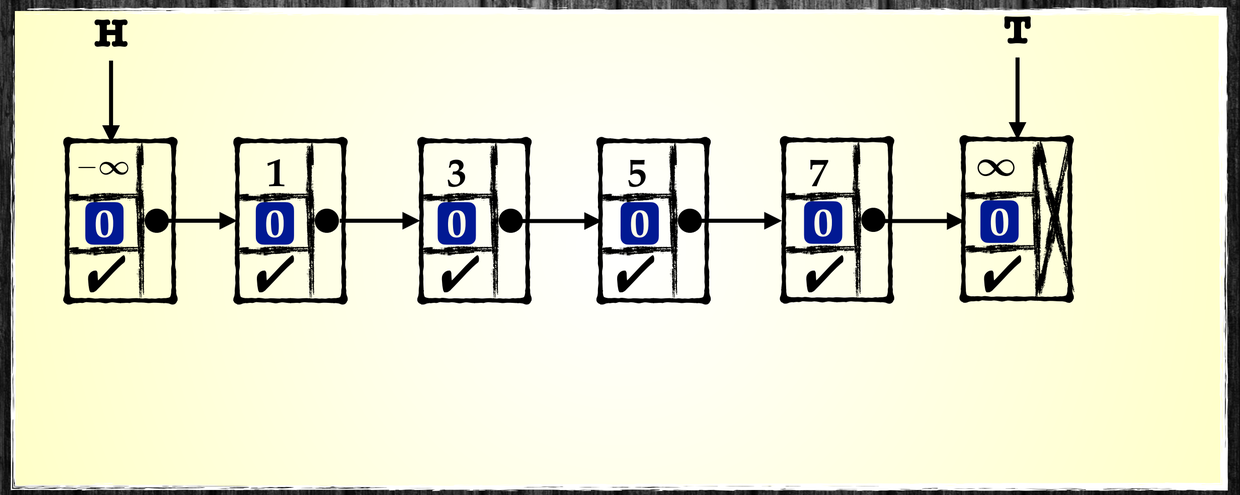
ctn(5)

```
ctn(e):
local c, b
1  c := Head;
2  while (c.val < e)
3     c := c.next
4  b := c.mark
5  if (!b && c.val = e)
6     return true;
7  else return false;
```

Scenario 2

- what if “mark” is first read
- and then set?



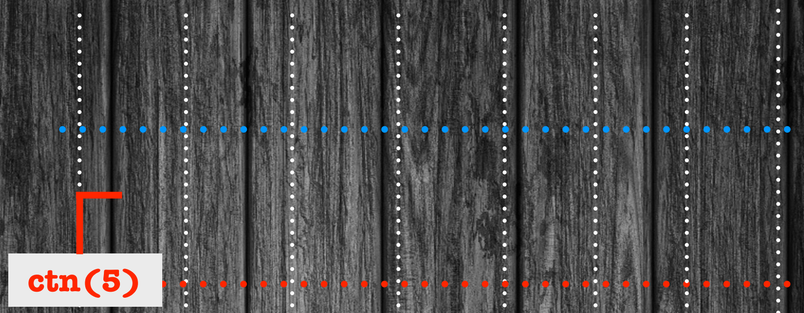


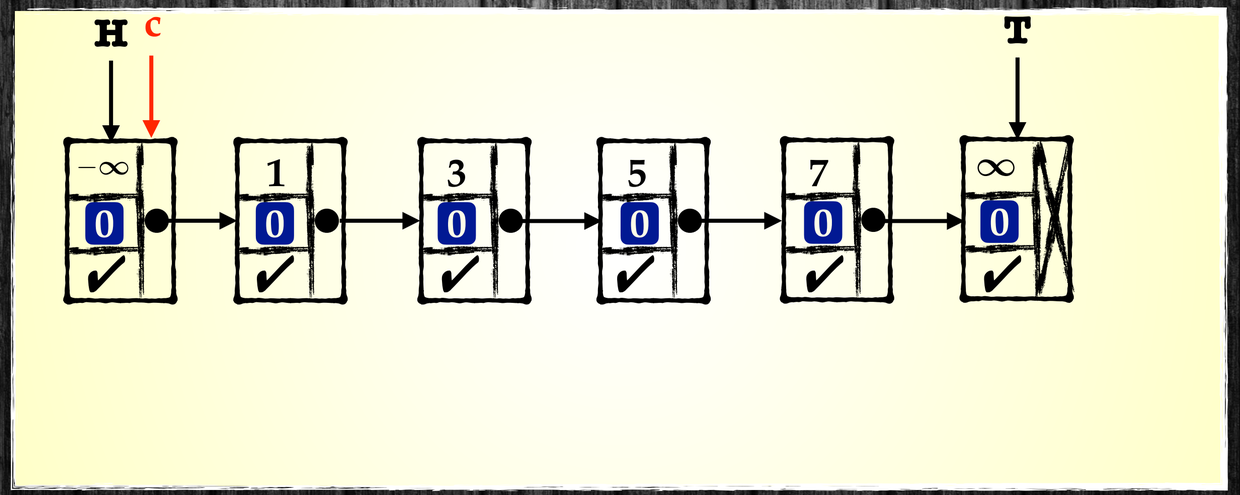
ctn(5)

```
ctn(e):
local c, b
1  c := Head;
2  while (c.val < e)
3     c := c.next
4  b := c.mark
5  if (!b && c.val = e)
6     return true;
7  else return false;
```

Scenario 2

- what if “mark” is first read
- and then set?



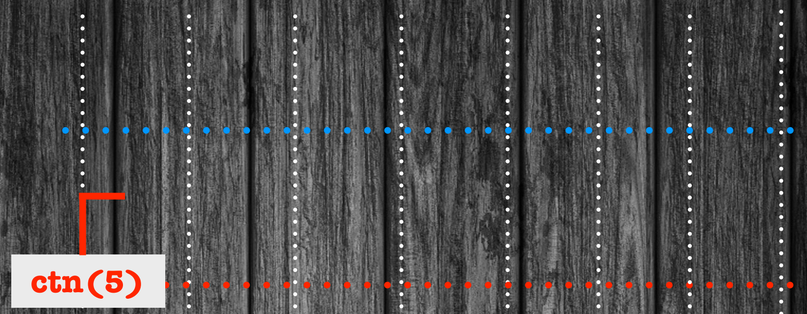


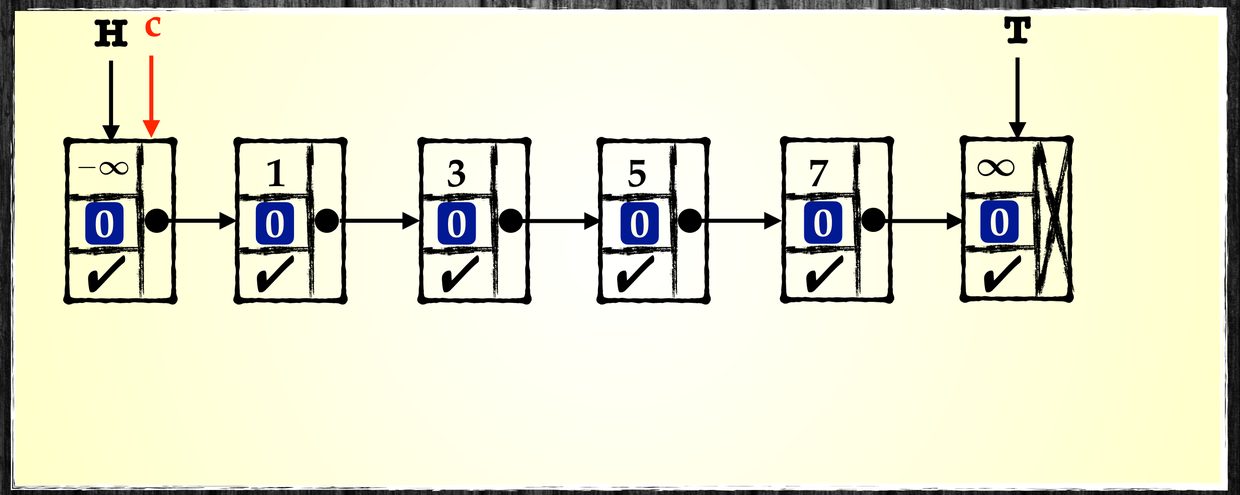
ctn(5)

```
ctn(e):
local c, b
1  c := Head;
2  while (c.val < e)
3     c := c.next
4  b := c.mark
5  if (!b && c.val = e)
6     return true;
7  else return false;
```

Scenario 2

- what if “mark” is first read
- and then set?



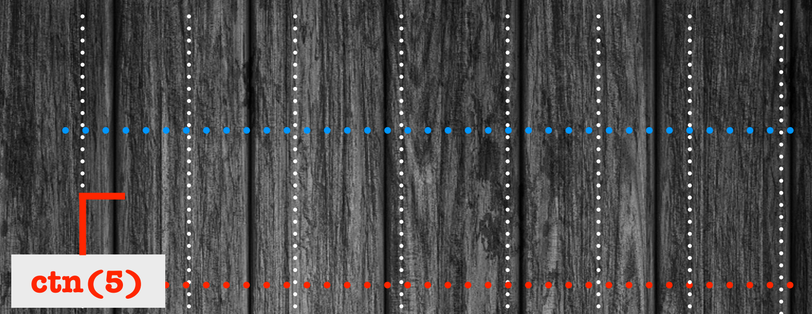


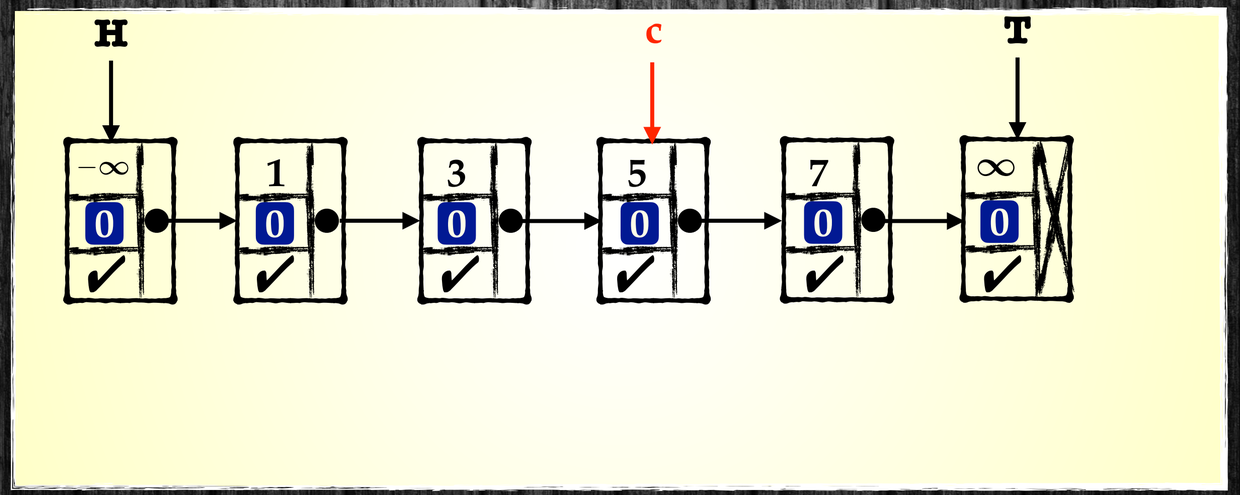
ctn(5)

```
ctn(e):
local c, b
1  c := Head;
2  while (c.val < e)
3     c := c.next
4  b := c.mark
5  if (!b && c.val = e)
6     return true;
7  else return false;
```

Scenario 2

- what if “mark” is first read
- and then set?



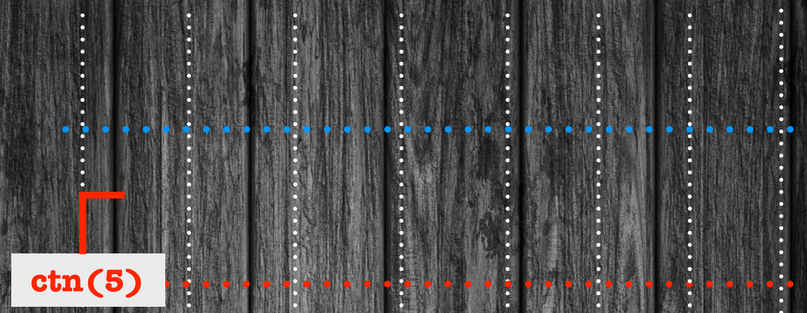


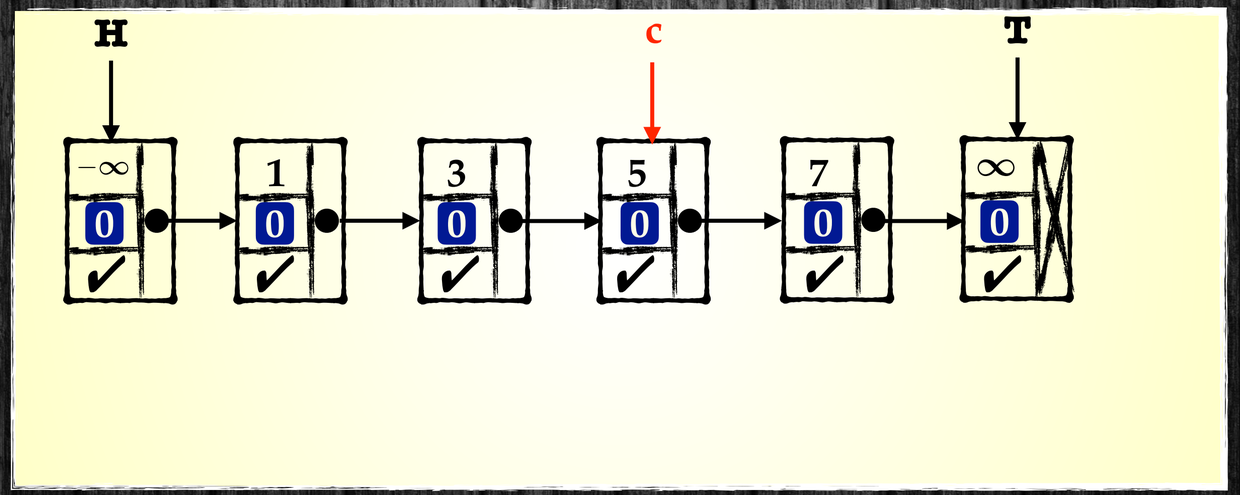
ctn(5)

```
ctn(e):
local c, b
1  c := Head;
2  while (c.val < e)
3     c := c.next
4  b := c.mark
5  if (!b && c.val = e)
6     return true;
7  else return false;
```

Scenario 2

- what if “mark” is first read
- and then set?



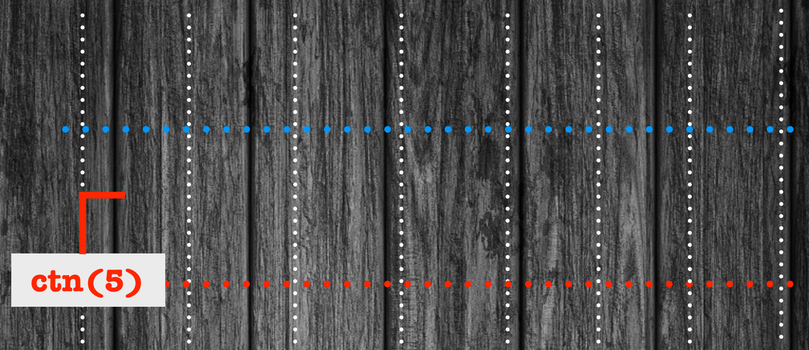


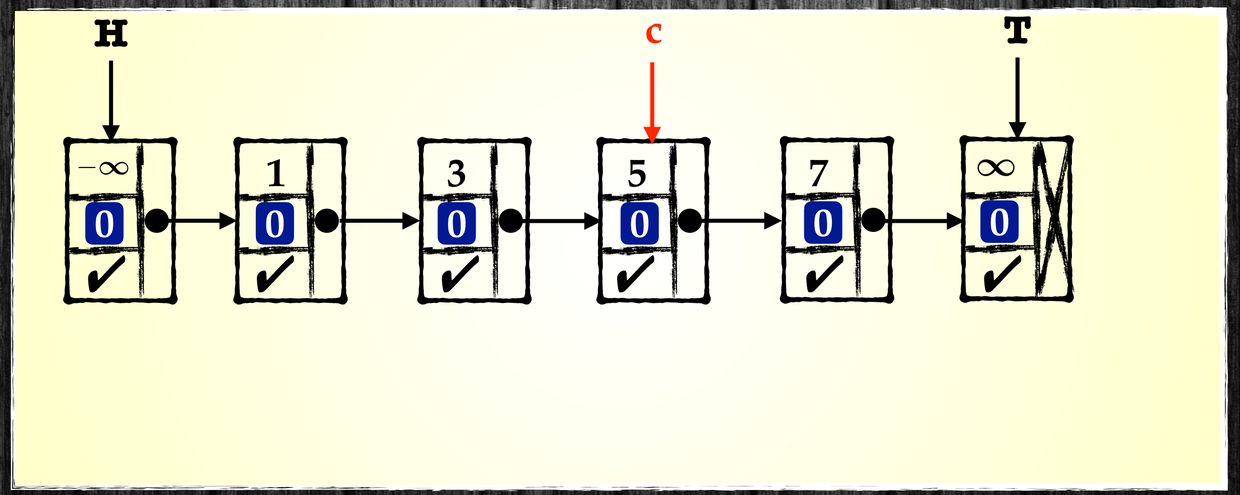
ctn(5)

```
ctn(e):
local c, b
1  c := Head;
2  while (c.val < e)
3     c := c.next
4  b := c.mark
5  if (!b && c.val = e)
6     return true;
7  else return false;
```

Scenario 2

- what if “mark” is first read
- and then set?



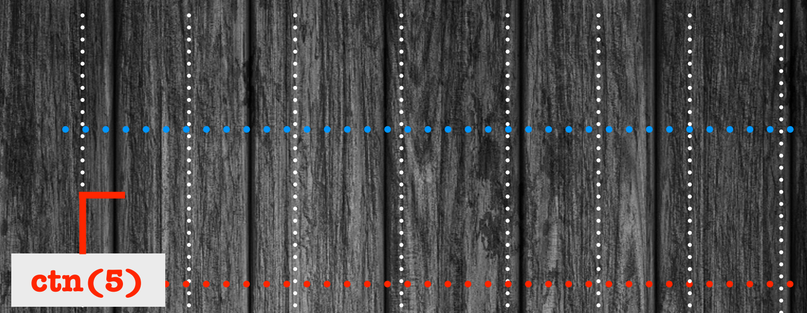


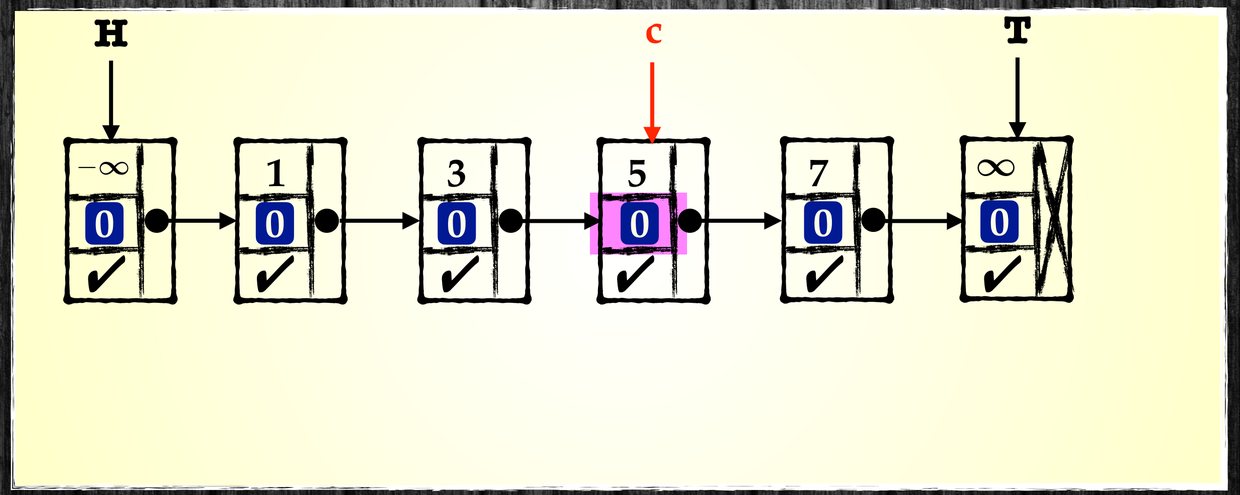
ctn(5)

```
ctn(e):
local c, b
1  c := Head;
2  while (c.val < e)
3     c := c.next
4  b := c.mark
5  if (!b && c.val = e)
6     return true;
7  else return false;
```

Scenario 2

- what if “mark” is first read
- and then set?





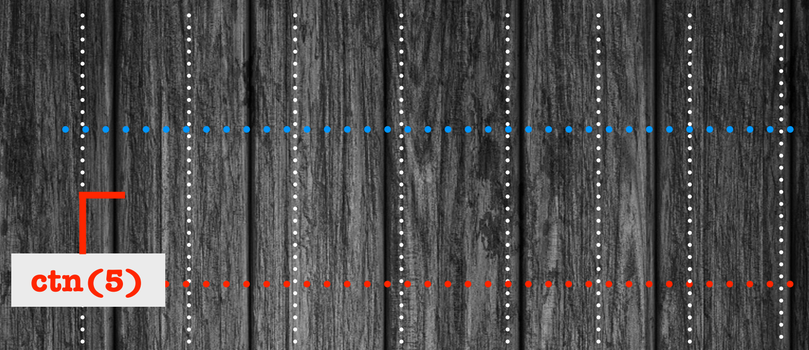
ctn(5)

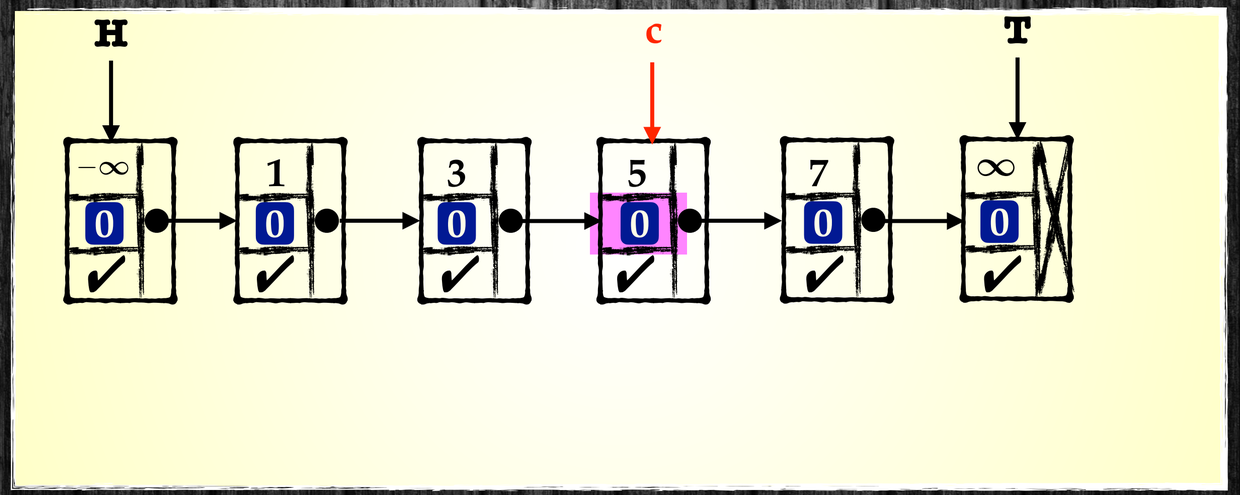
```
ctn(e):
local c, b
1  c := Head;
2  while (c.val < e)
3     c := c.next
4  b := c.mark
5  if (!b && c.val = e)
6     return true;
7  else return false;
```

0

Scenario 2

- what if “mark” is first read
- and then set?





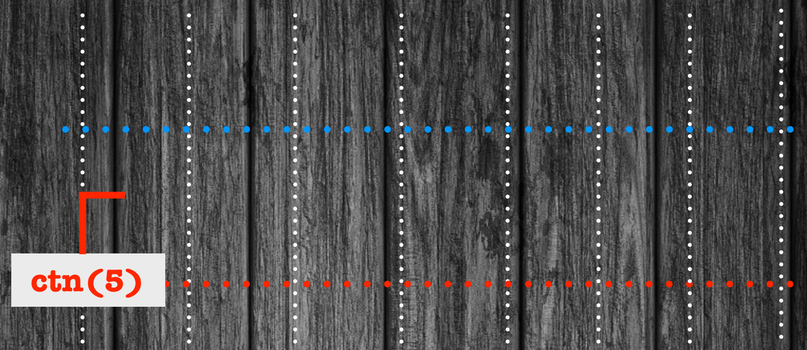
ctn(5)

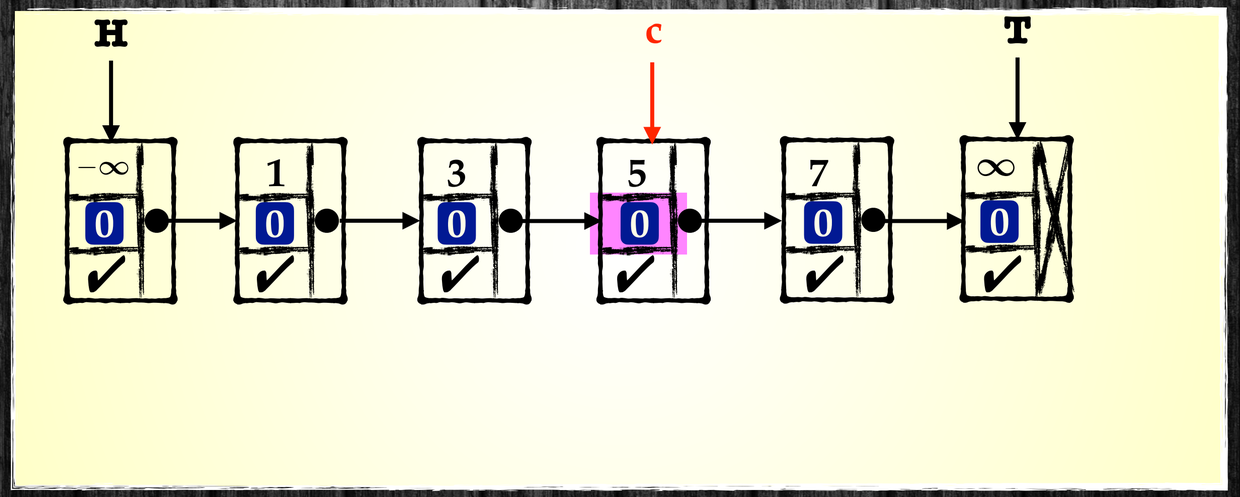
```
ctn(e):
local c, b
1  c := Head;
2  while (c.val < e)
3     c := c.next
4  b := c.mark
5  if (!b && c.val = e)
6     return true;
7  else return false;
```

0

Scenario 2

- what if “mark” is first read
- and then set?





linearization point

ctn(5)

```
ctn(e):
local c, b
1  c := Head;
2  while (c.val < e)
3     c := c.next
4  b := c.mark
5  if (!b && c.val = e)
6     return true;
7  else return false;
```

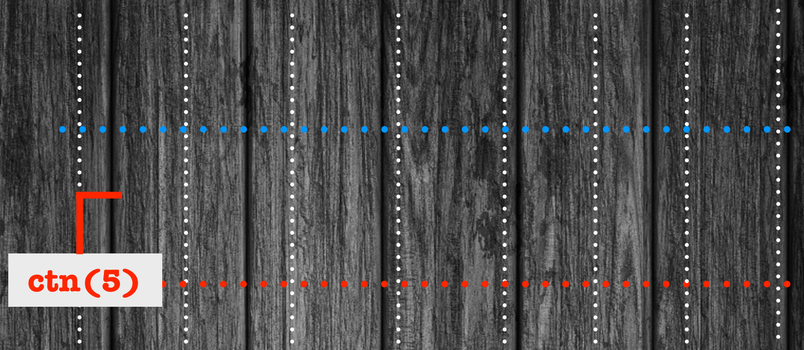
0

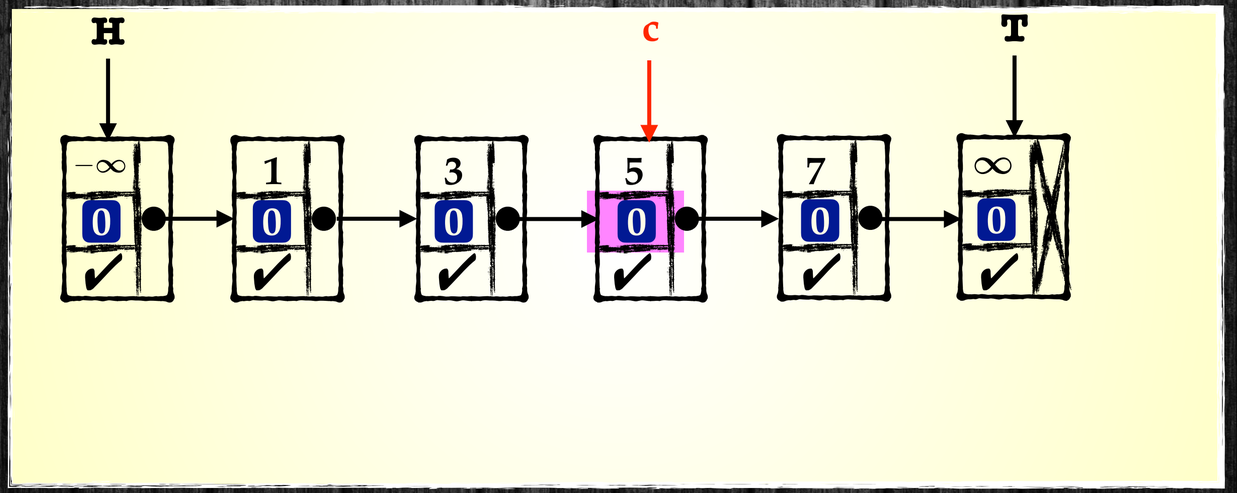
successful contains

linearization point

Scenario 2

- what if “mark” is first read
- and then set?





ctn(5)

```
ctn(e):
local c, b
1  c := Head;
2  while (c.val < e)
3     c := c.next
4  b := c.mark
5  if (!b && c.val = e)
6     return true;
7  else return false;
```

0

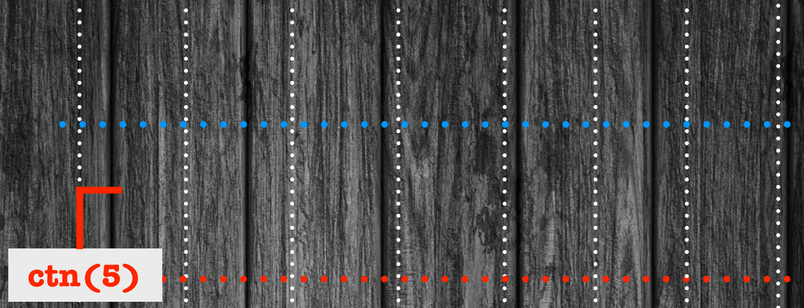
"unmarked entry with input key"

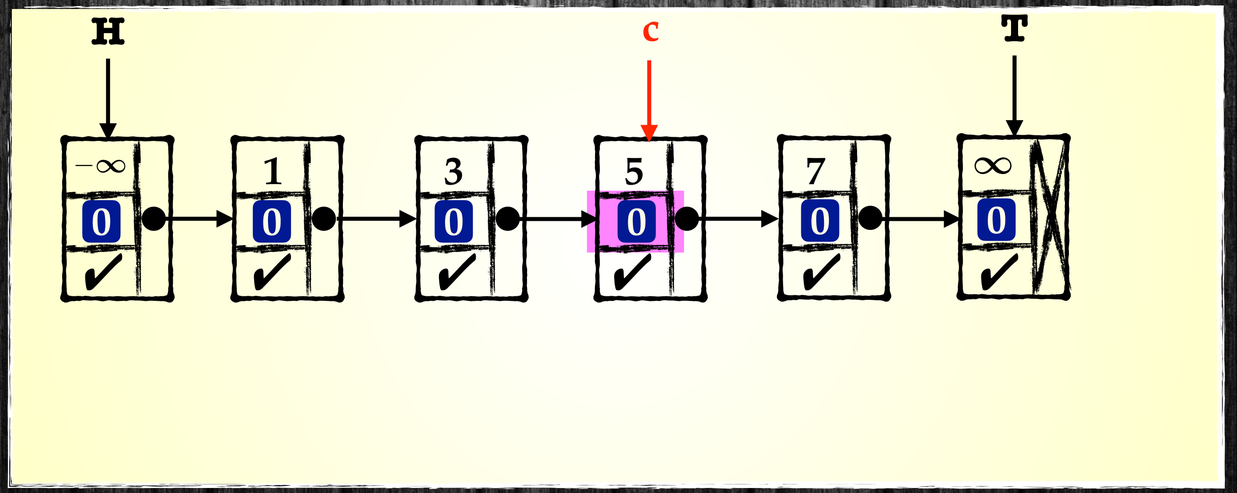
successful contains

linearization point

Scenario 2

- what if "mark" is first read
- and then set?





ctn(5)

```
ctn(e):
local c, b
1  c := Head;
2  while (c.val < e)
3     c := c.next
4  b := c.mark
5  if (!b && c.val = e)
6     return true;
7  else return false;
```

0

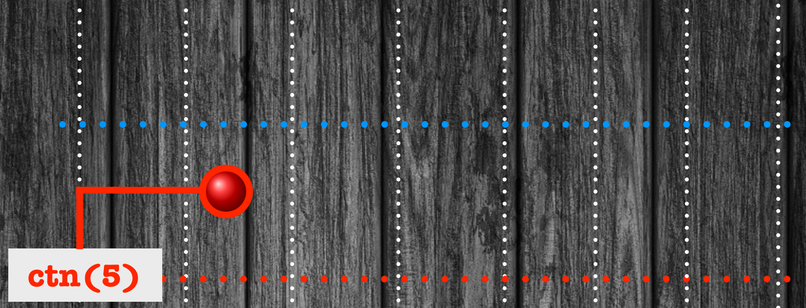
"unmarked entry with input key"

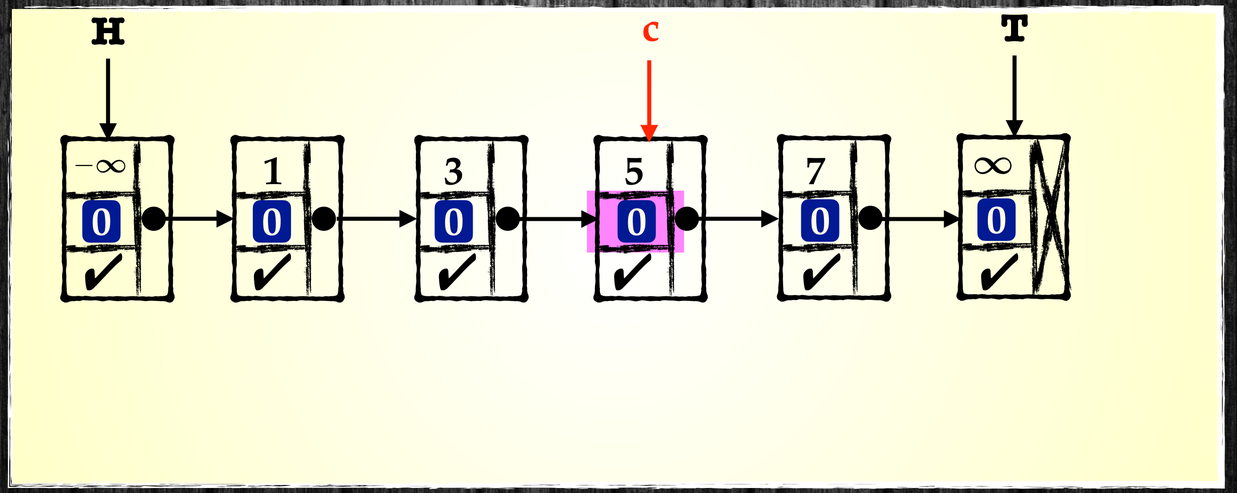
successful contains

linearization point

Scenario 2

- what if "mark" is first read
- and then set?





ctn(5)

```
ctn(e):
local c, b
1  c := Head;
2  while (c.val < e)
3     c := c.next
4  b := c.mark
5  if (!b && c.val = e)
6     return true;
7  else return false;
```

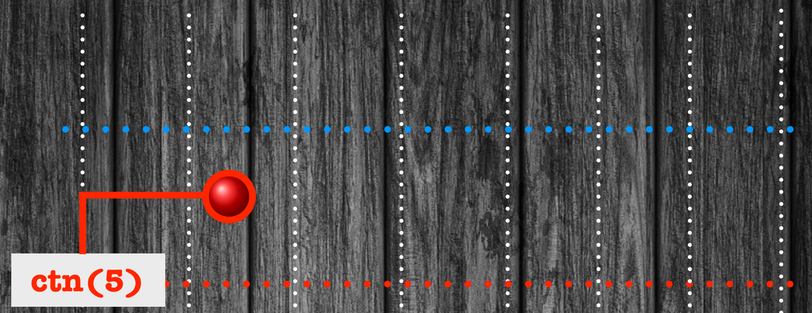
0

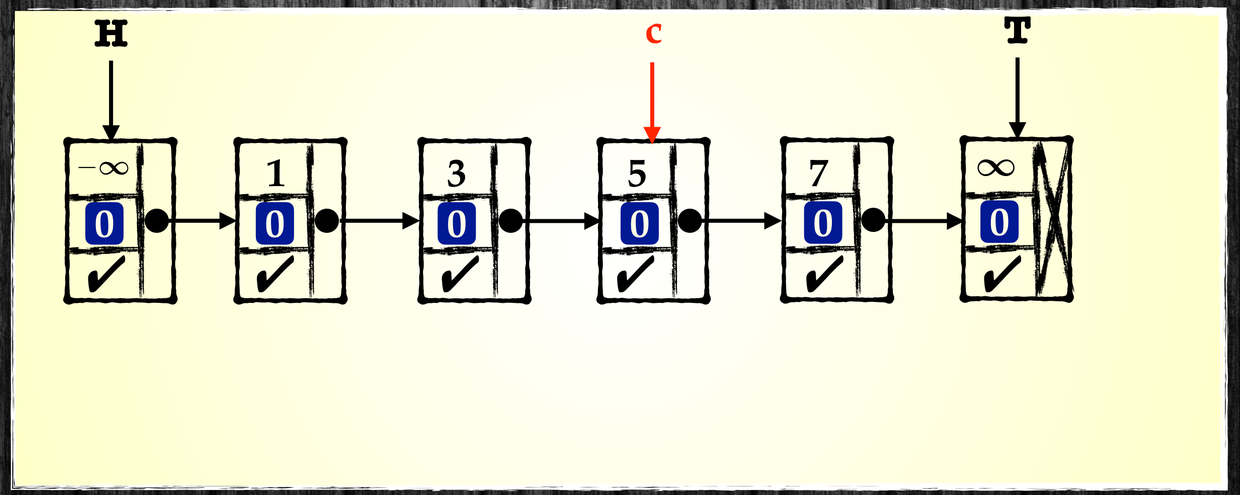
rmv(5)

```
rmv(e):
local p, c, n, r
1  (p,c) := locate(e);
2  if (c.val = e)
3     c.mark := true
4     n := c.next;
5     p.next := n;
6     r := true;
7  else r := false;
8  unlock(p);
9  unlock(c);
10 return r;
```

Scenario 2

- what if “mark” is first read
- and then set?





ctn(5)

```
ctn(e):
local c, b
1  c := Head;
2  while (c.val < e)
3     c := c.next
4  b := c.mark
5  if (!b && c.val = e)
6     return true;
7  else return false;
```

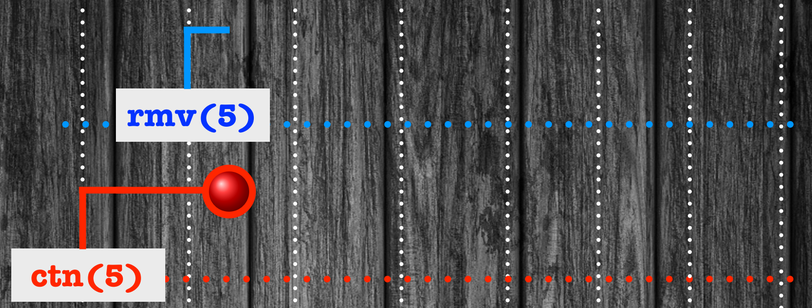
0

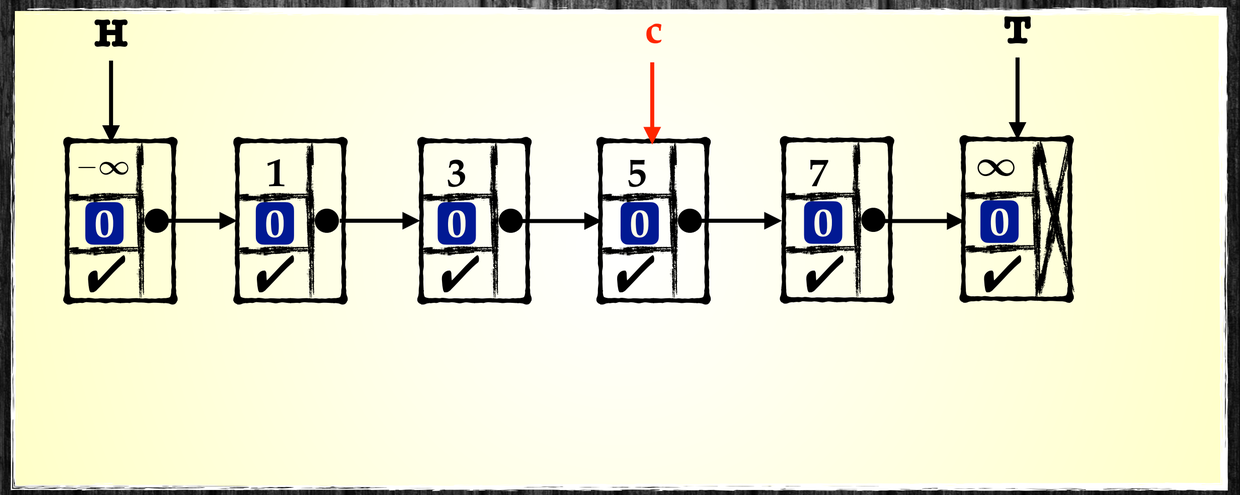
rmv(e): rmv(5)

```
local p, c, n, r
1  (p,c) := locate(e);
2  if (c.val = e)
3     c.mark := true
4     n := c.next;
5     p.next := n;
6     r := true;
7  else r := false;
8  unlock(p);
9  unlock(c);
10 return r;
```

Scenario 2

- what if “mark” is first read
- and then set?





ctn(5)

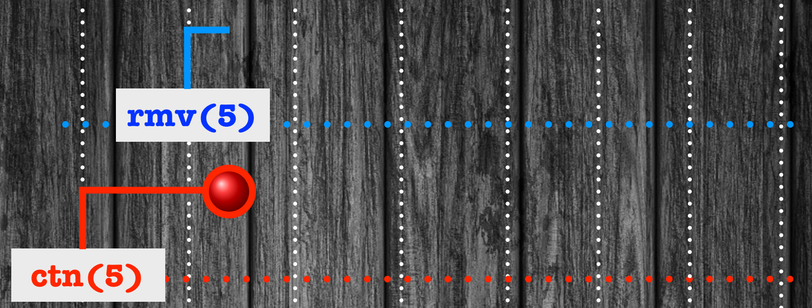
```
ctn(e):
local c, b
1  c := Head;
2  while (c.val < e)
3     c := c.next
4  b := c.mark
5  if (!b && c.val = e)
6     return true;
7  else return false;
```

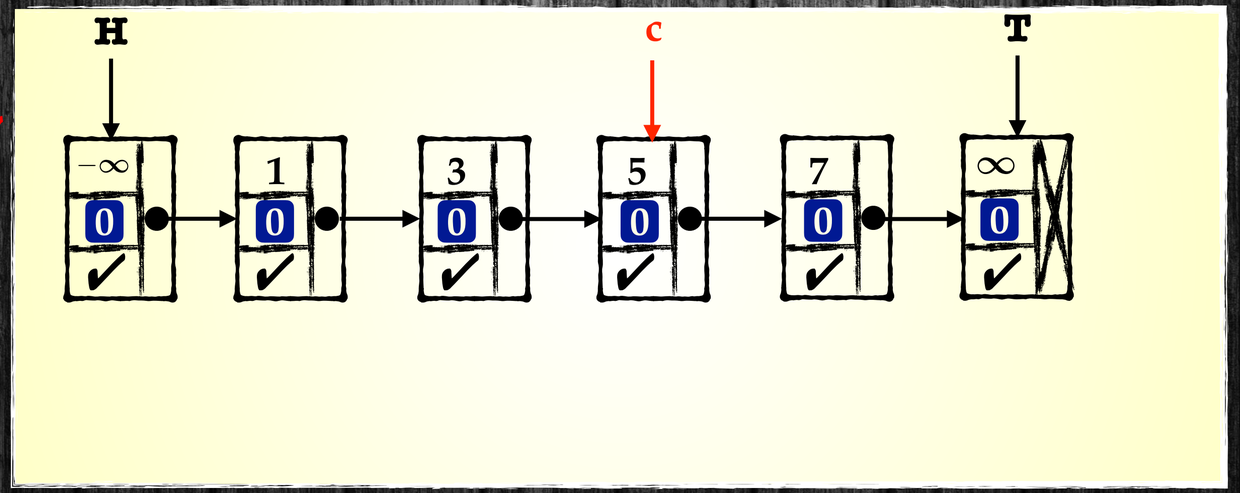
rmv(e): rmv(5)

```
local p, c, n, r
1  (p,c) := locate(e);
2  if (c.val = e)
3     c.mark := true
4     n := c.next;
5     p.next := n;
6     r := true;
7  else r := false;
8  unlock(p);
9  unlock(c);
10 return r;
```

Scenario 2

- what if “mark” is first read
- and then set?





ctn(5)

```
ctn(e):
local c, b
1  c := Head;
2  while (c.val < e)
3     c := c.next
4  b := c.mark
5  if (!b && c.val = e)
6     return true;
7  else return false;
```

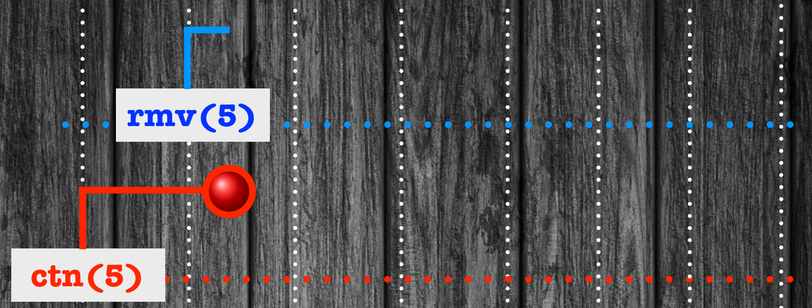
0

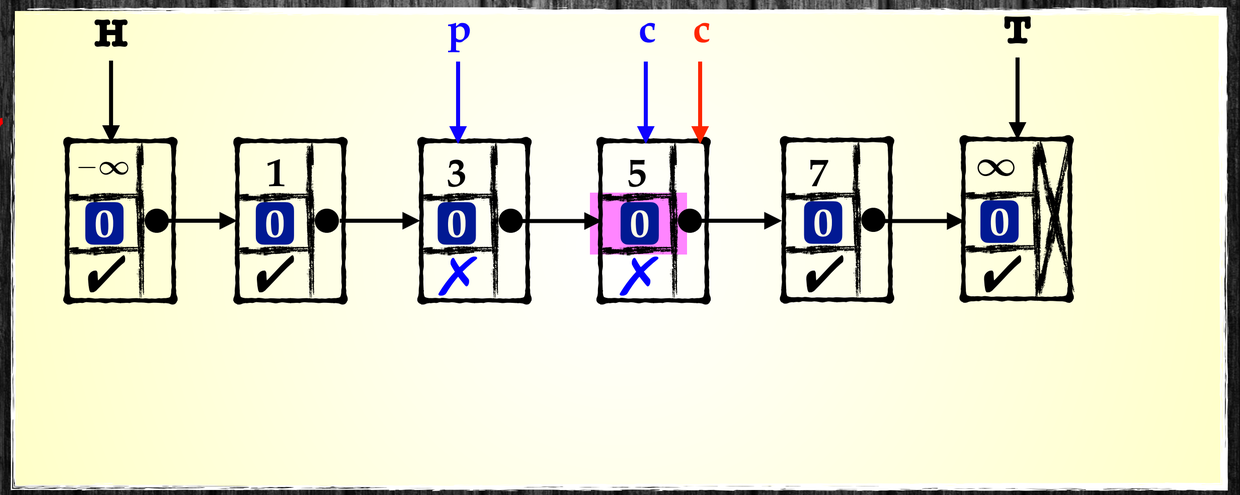
rmv(e): rmv(5)

```
local p, c, n, r
1  (p,c) := locate(e);
2  if (c.val = e)
3     c.mark := true
4     n := c.next;
5     p.next := n;
6     r := true;
7  else r := false;
8  unlock(p);
9  unlock(c);
10 return r;
```

Scenario 2

- what if “mark” is first read
- and then set?





ctn(5)

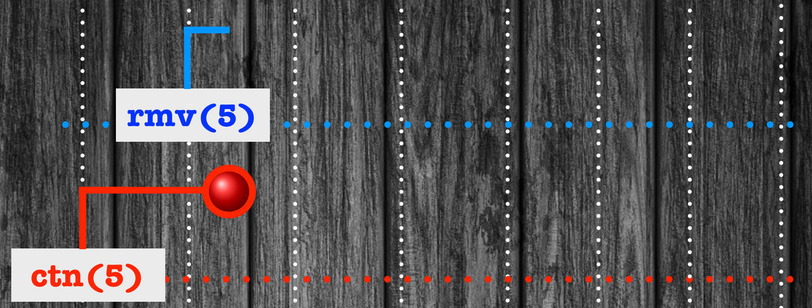
```
ctn(e):
local c, b
1  c := Head;
2  while (c.val < e)
3     c := c.next
4  b := c.mark
5  if (!b && c.val = e)
6     return true;
7  else return false;
```

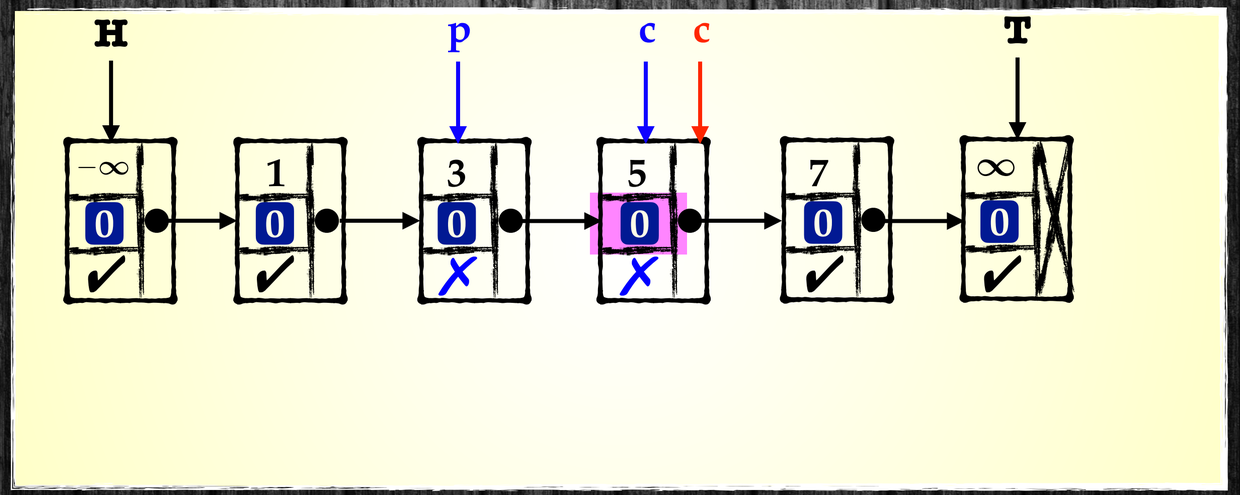
rmv(5)

```
rmv(e):
local p, c, n, r
1  (p,c) := locate(e);
2  if (c.val = e)
3     c.mark := true
4     n := c.next;
5     p.next := n;
6     r := true;
7  else r := false;
8  unlock(p);
9  unlock(c);
10 return r;
```

Scenario 2

- what if “mark” is first read
- and then set?





ctn(5)

```
ctn(e):
local c, b
1  c := Head;
2  while (c.val < e)
3     c := c.next
4  b := c.mark
5  if (!b && c.val = e)
6     return true;
7  else return false;
```

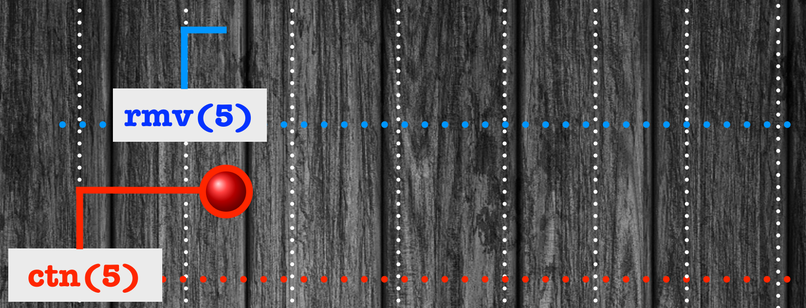
0

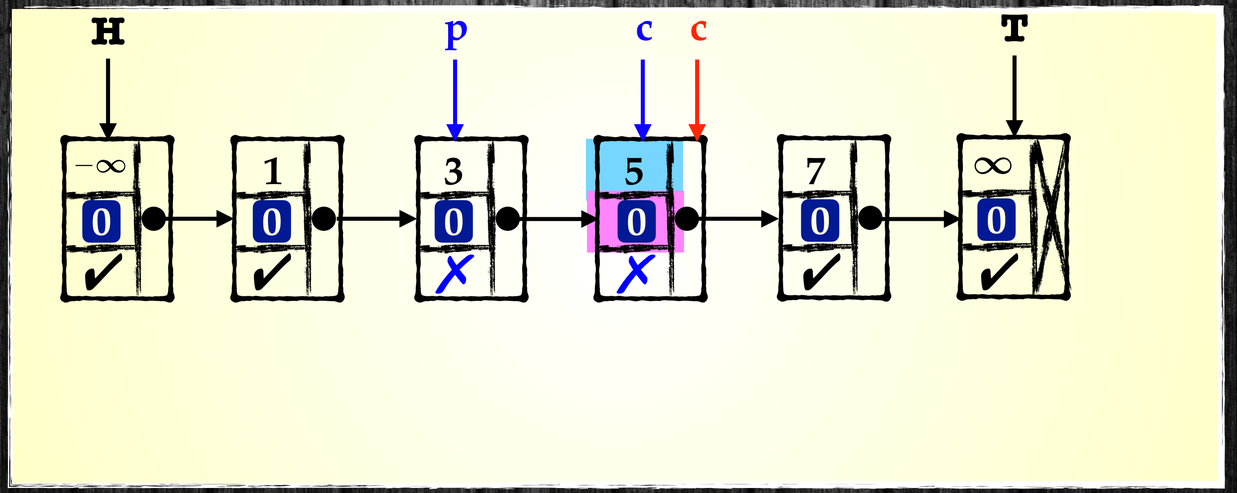
rmv(e): rmv(5)

```
local p, c, n, r
1  (p,c) := locate(e);
2  if (c.val = e)
3     c.mark := true
4     n := c.next;
5     p.next := n;
6     r := true;
7  else r := false;
8  unlock(p);
9  unlock(c);
10 return r;
```

Scenario 2

- what if “mark” is first read
- and then set?





ctn(5)

```
ctn(e):
local c, b
1  c := Head;
2  while (c.val < e)
3     c := c.next
4  b := c.mark
5  if (!b && c.val = e)
6     return true;
7  else return false;
```

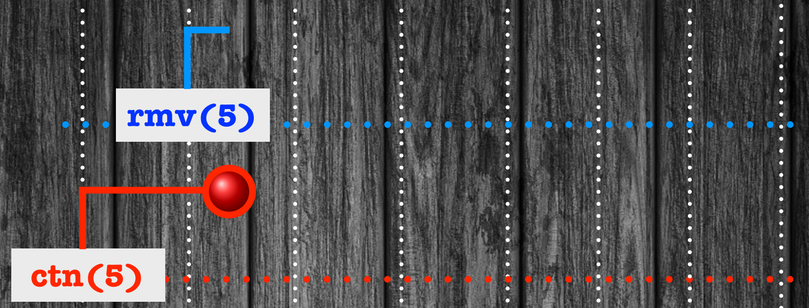
0

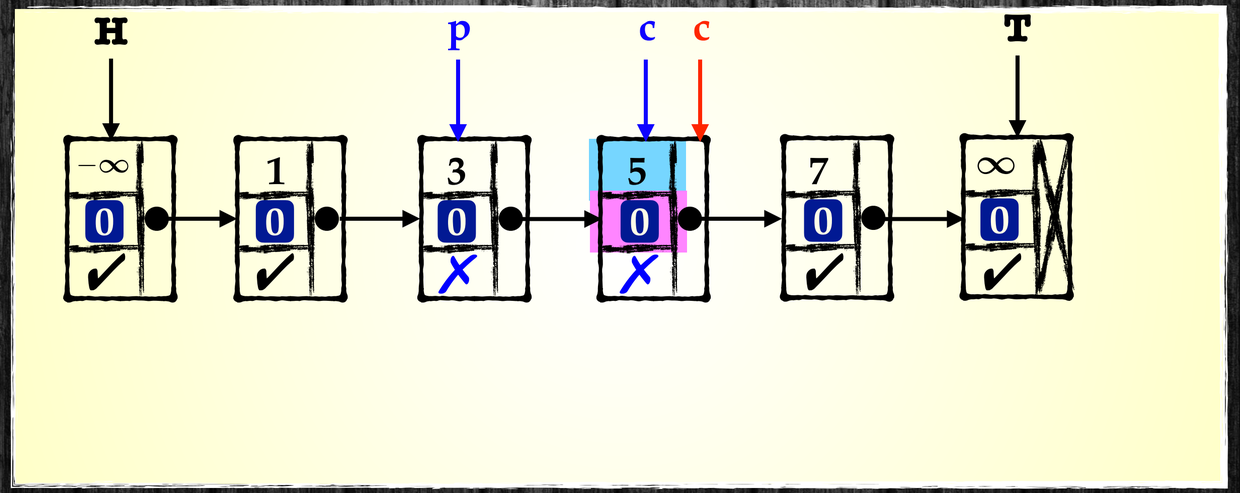
rmv(e): rmv(5)

```
local p, c, n, r
1  (p,c) := locate(e);
2  if (c.val = e)
3     c.mark := true
4     n := c.next;
5     p.next := n;
6     r := true;
7  else r := false;
8  unlock(p);
9  unlock(c);
10 return r;
```

Scenario 2

- what if “mark” is first read
- and then set?





ctn(5)

```
ctn(e):
local c, b
1  c := Head;
2  while (c.val < e)
3     c := c.next
4  b := c.mark
5  if (!b && c.val = e)
6     return true;
7  else return false;
```

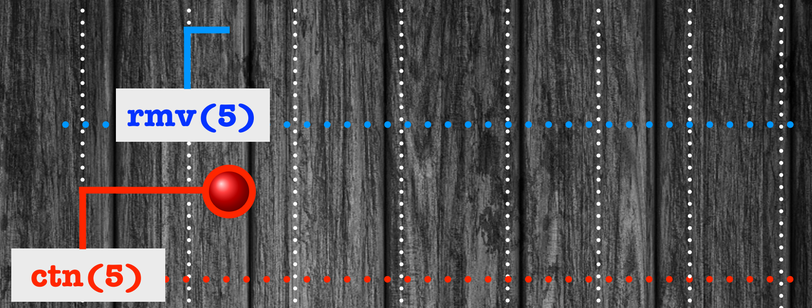
0

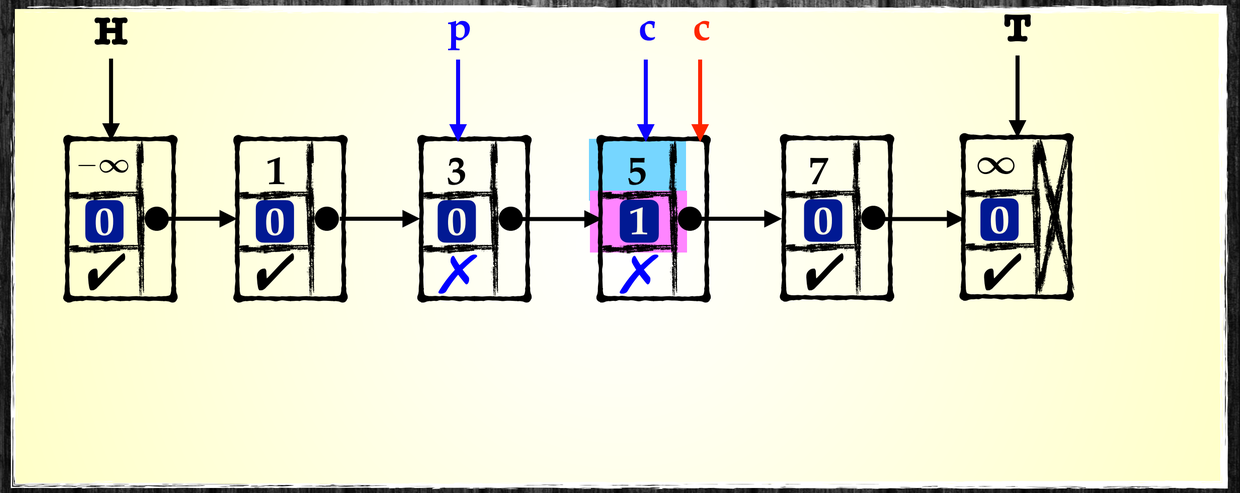
rmv(e): rmv(5)

```
local p, c, n, r
1  (p,c) := locate(e);
2  if (c.val = e)
3     c.mark := true
4     n := c.next;
5     p.next := n;
6     r := true;
7  else r := false;
8  unlock(p);
9  unlock(c);
10 return r;
```

Scenario 2

- what if “mark” is first read
- and then set?





ctn(5)

```
ctn(e):
local c, b
1  c := Head;
2  while (c.val < e)
3     c := c.next
4  b := c.mark
5  if (!b && c.val = e)
6     return true;
7  else return false;
```

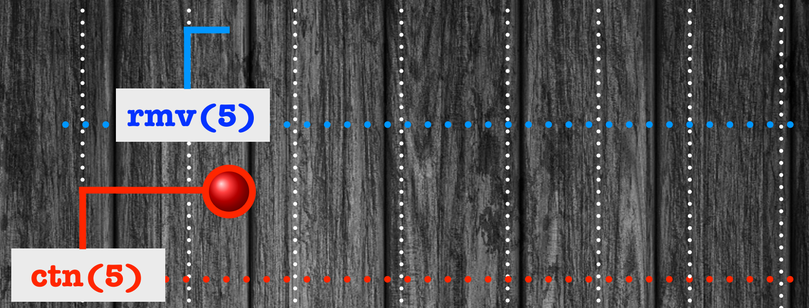
0

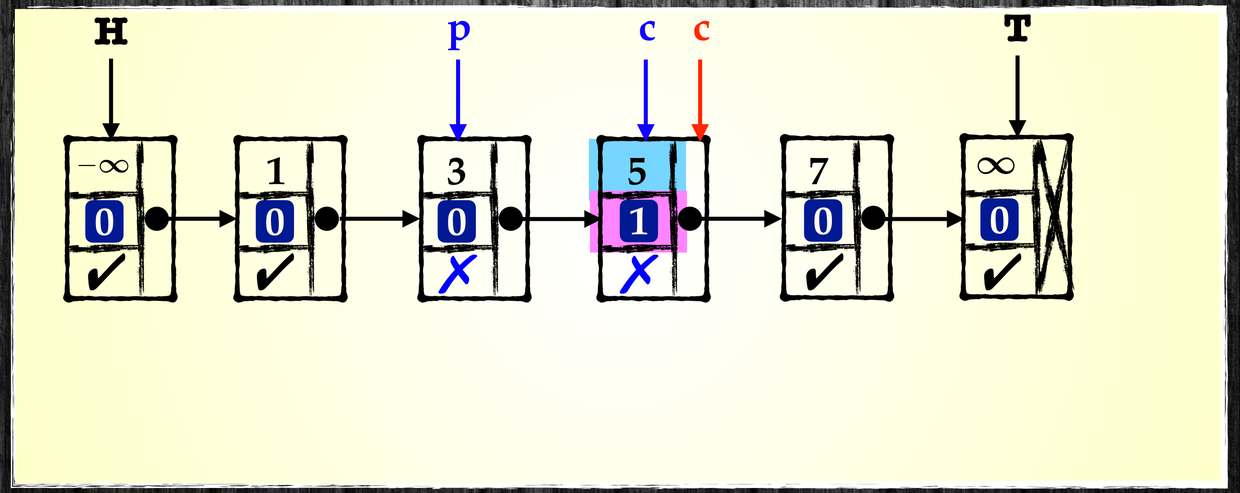
rmv(e): rmv(5)

```
local p, c, n, r
1  (p,c) := locate(e);
2  if (c.val = e)
3     c.mark := true
4     n := c.next;
5     p.next := n;
6     r := true;
7  else r := false;
8  unlock(p);
9  unlock(c);
10 return r;
```

Scenario 2

- what if “mark” is first read
- and then set?





ctn(5)

```
ctn(e):
local c, b
1  c := Head;
2  while (c.val < e)
3     c := c.next
4  b := c.mark
5  if (!b && c.val = e)
6     return true;
7  else return false;
```

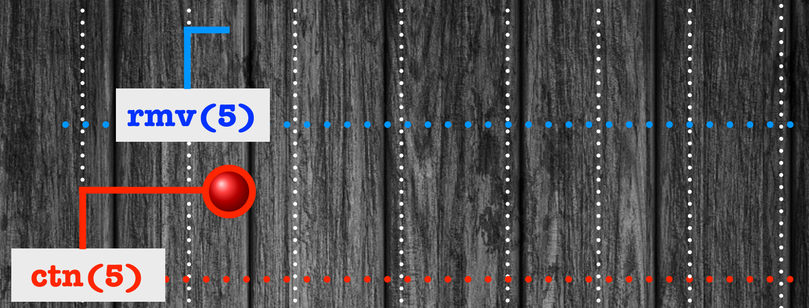
0

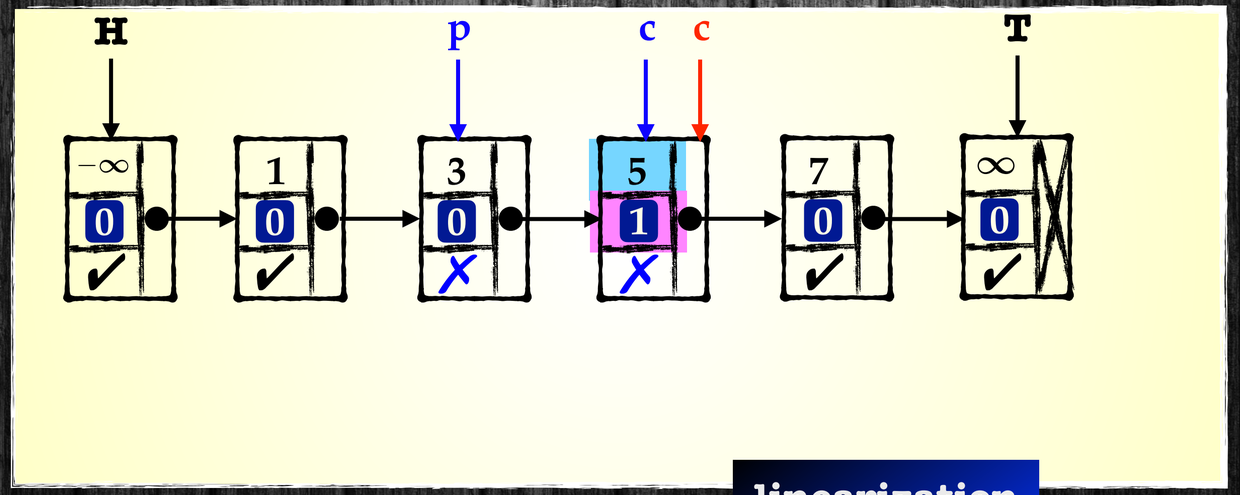
rmv(e): rmv(5)

```
local p, c, n, r
1  (p,c) := locate(e);
2  if (c.val = e)
3     c.mark := true
4     n := c.next;
5     p.next := n;
6     r := true;
7  else r := false;
8  unlock(p);
9  unlock(c);
10 return r;
```

Scenario 2

- what if “mark” is first read
- and then set?





linearization point

ctn(5)

```
ctn(e):
local c, b
1  c := Head;
2  while (c.val < e)
3     c := c.next
4  b := c.mark
5  if (!b && c.val = e)
6     return true;
7  else return false;
```

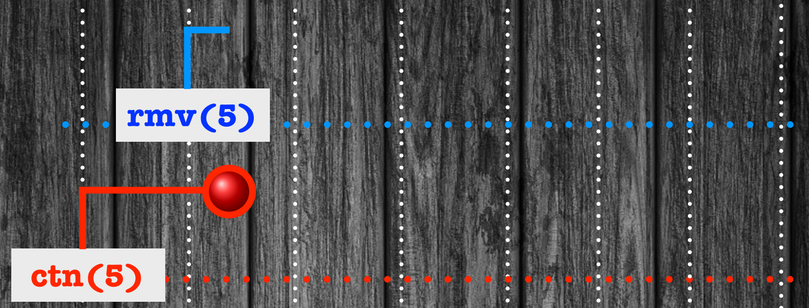
0

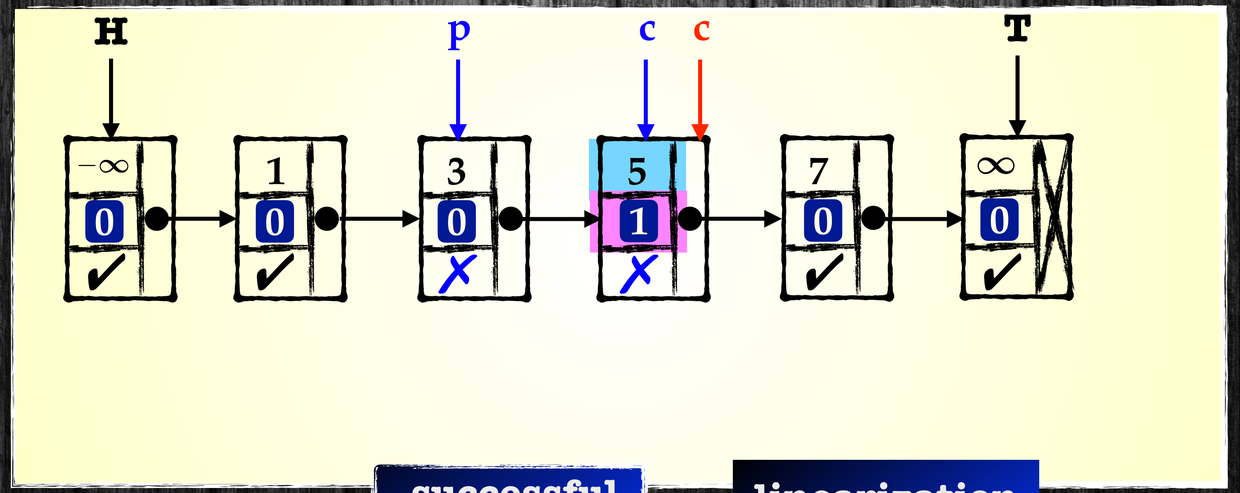
rmv(e): rmv(5)

```
local p, c, n, r
1  (p,c) := locate(e);
2  if (c.val = e)
3     c.mark := true
4     n := c.next;
5     p.next := n;
6     r := true;
7  else r := false;
8  unlock(p);
9  unlock(c);
10 return r;
```

Scenario 2

- what if “mark” is first read
- and then set?





successful remove

linearization point

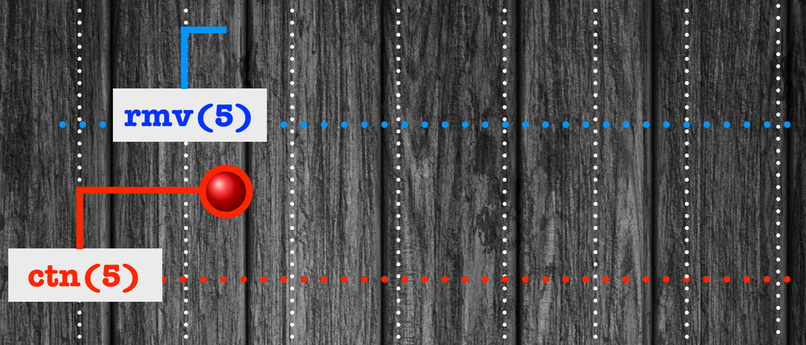
ctn(5)

```
ctn(e):
local c, b
1  c := Head;
2  while (c.val < e)
3     c := c.next
4  b := c.mark
5  if (!b && c.val = e)
6     return true;
7  else return false;
```

0

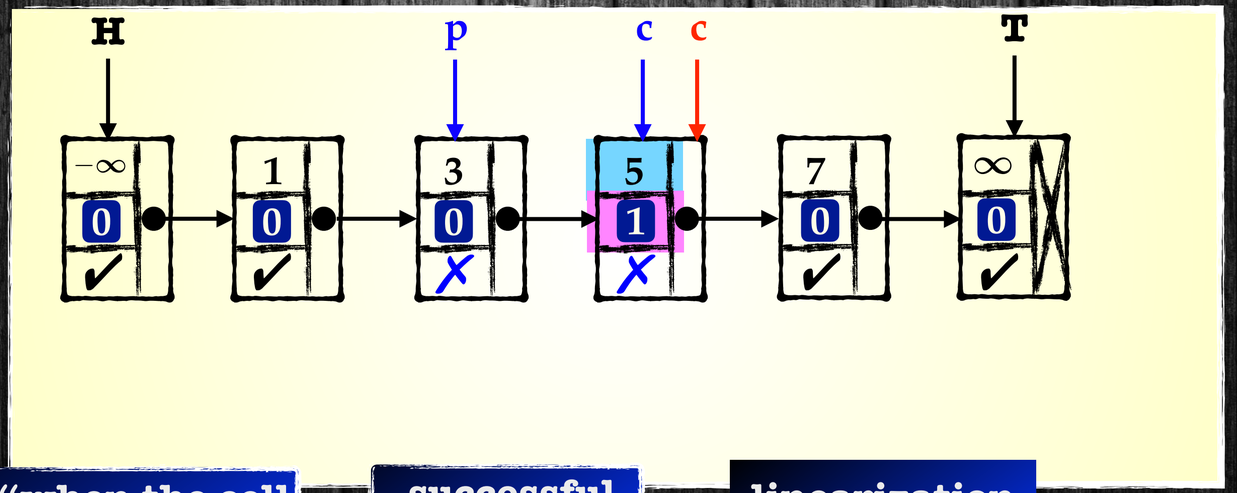
Scenario 2

- what if “mark” is first read
- and then set?



rmv(5)

```
rmv(e):
local p, c, n, r
1  (p,c) := locate(e);
2  if (c.val = e)
3     c.mark := true
4     n := c.next;
5     p.next := n;
6     r := true;
7  else r := false;
8  unlock(p);
9  unlock(c);
10 return r;
```



“when the cell is marked”

successful remove

linearization point

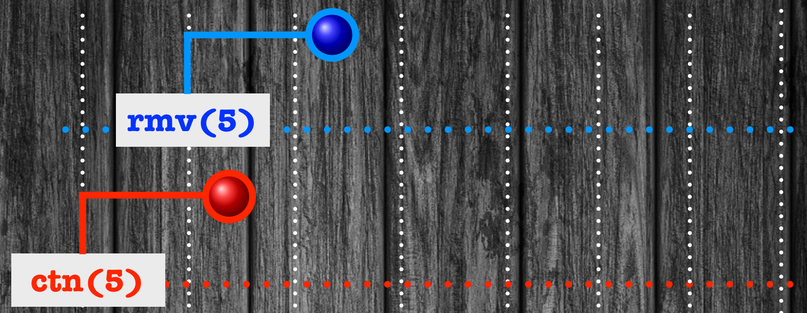
ctn(5)

```
ctn(e):
local c, b
1  c := Head;
2  while (c.val < e)
3     c := c.next
4  b := c.mark
5  if (!b && c.val = e)
6     return true;
7  else return false;
```

0

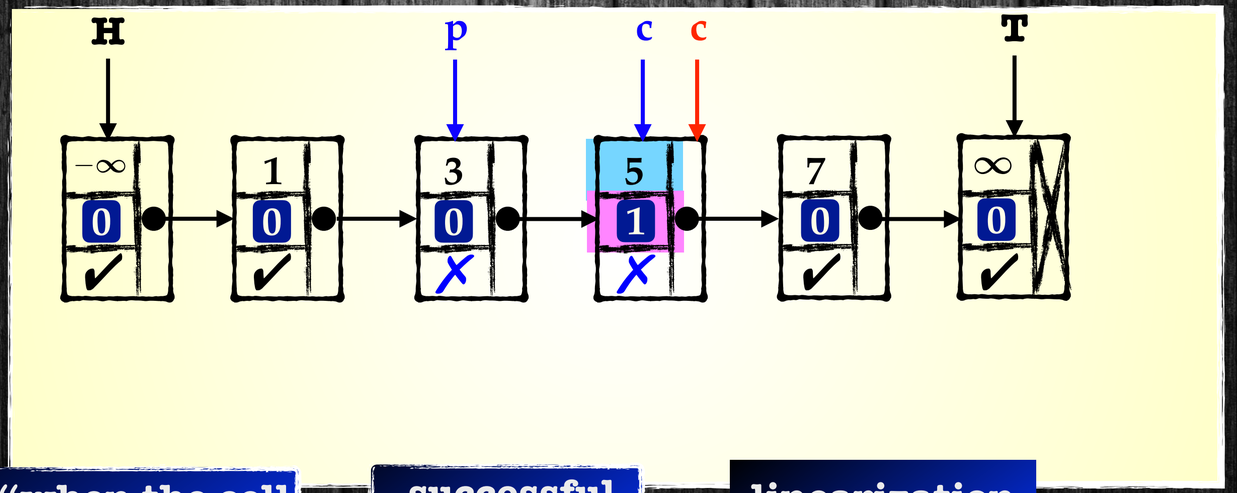
Scenario 2

- what if “mark” is first read
- and then set?



rmv(5)

```
rmv(e):
local p, c, n, r
1  (p,c) := locate(e);
2  if (c.val = e)
3     c.mark := true
4     n := c.next;
5     p.next := n;
6     r := true;
7  else r := false;
8  unlock(p);
9  unlock(c);
10 return r;
```



“when the cell is marked”

successful remove

linearization point

ctn(5)

```
ctn(e):
local c, b
1  c := Head;
2  while (c.val < e)
3     c := c.next
4  b := c.mark
5  if (!b && c.val = e)
6     return true;
7  else return false;
```

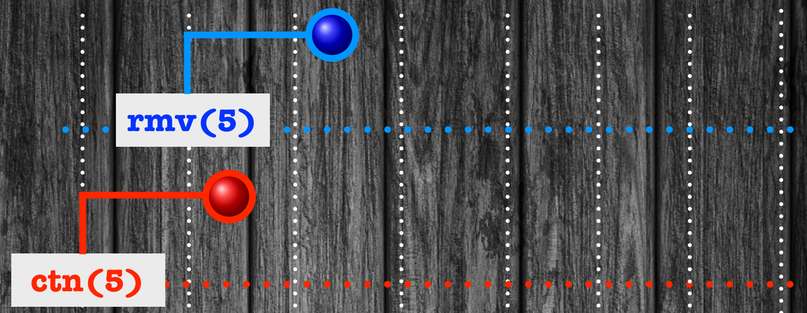
0

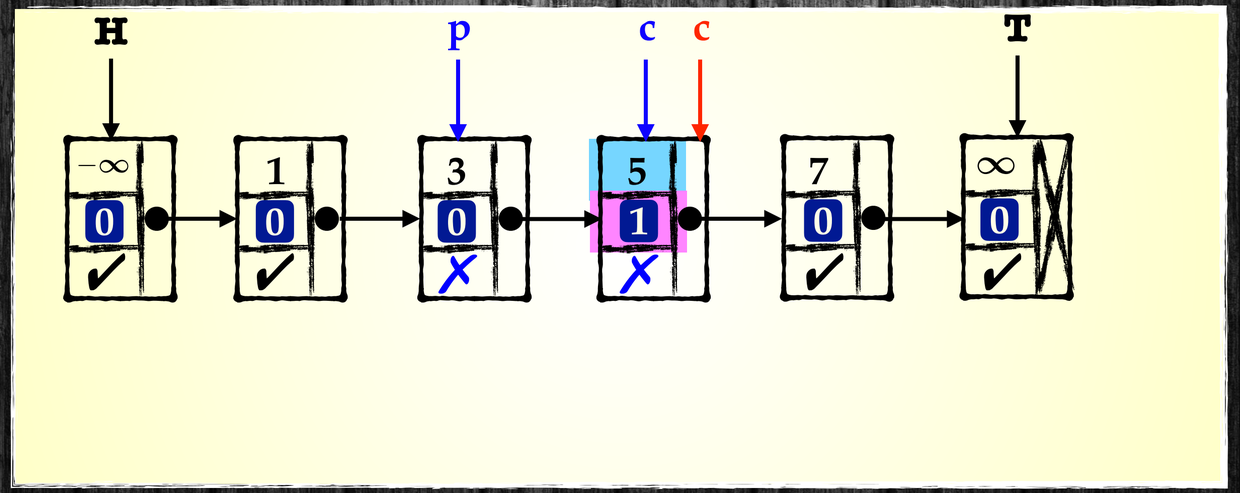
rmv(e): rmv(5)

```
local p, c, n, r
1  (p,c) := locate(e);
2  if (c.val = e)
3     c.mark := true
4     n := c.next;
5     p.next := n;
6     r := true;
7  else r := false;
8  unlock(p);
9  unlock(c);
10 return r;
```

Scenario 2

- what if “mark” is first read
- and then set?





ctn(5)

```
ctn(e):
local c, b
1  c := Head;
2  while (c.val < e)
3     c := c.next
4  b := c.mark
5  if (!b && c.val = e)
6     return true;
7  else return false;
```

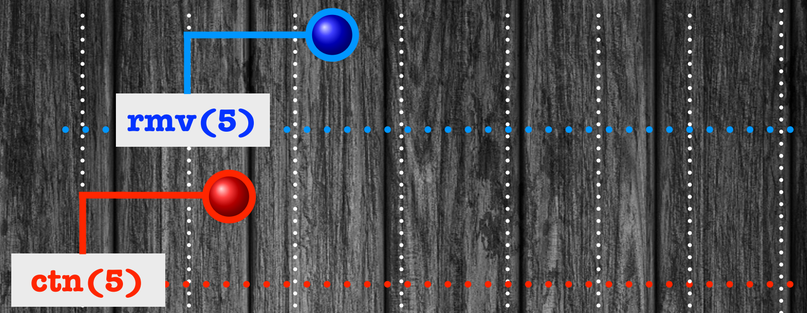
0

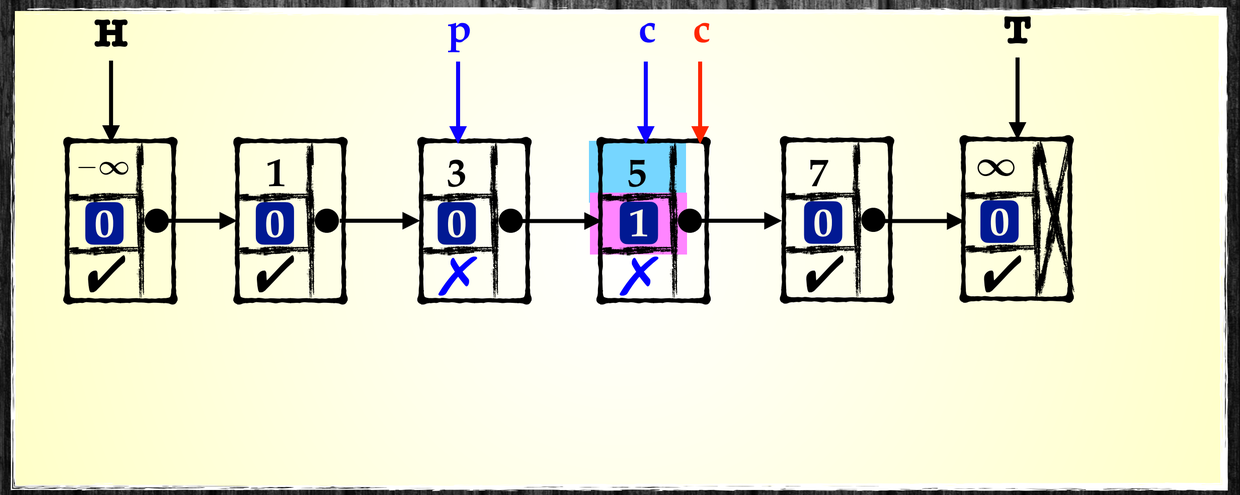
rmv(e): rmv(5)

```
local p, c, n, r
1  (p,c) := locate(e);
2  if (c.val = e)
3     c.mark := true
4     n := c.next;
5     p.next := n;
6     r := true;
7  else r := false;
8  unlock(p);
9  unlock(c);
10 return r;
```

Scenario 2

- what if “mark” is first read
- and then set?





ctn(5)

```
ctn(e):
local c, b
1  c := Head;
2  while (c.val < e)
3     c := c.next
4  b := c.mark
5  if (!b && c.val = e)
6     return true;
7  else return false;
```

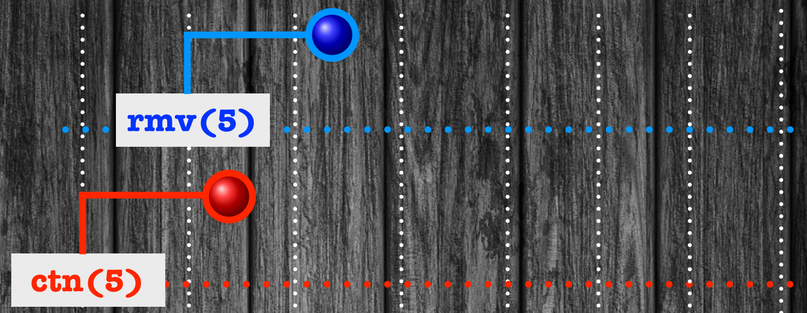
0

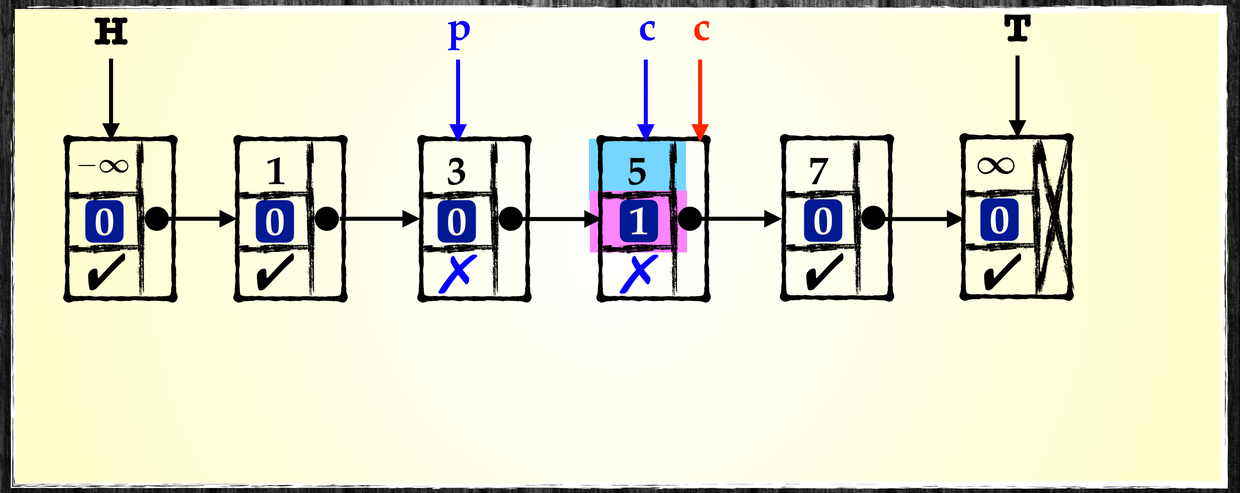
rmv(e): rmv(5)

```
local p, c, n, r
1  (p,c) := locate(e);
2  if (c.val = e)
3     c.mark := true
4     n := c.next;
5     p.next := n;
6     r := true;
7  else r := false;
8  unlock(p);
9  unlock(c);
10 return r;
```

Scenario 2

- what if “mark” is first read
- and then set?





ctn(5)

```
ctn(e):
local c, b
1  c := Head;
2  while (c.val < e)
3     c := c.next
4  b := c.mark
5  if (!b && c.val = e)
6     return true;
7  else return false;
```

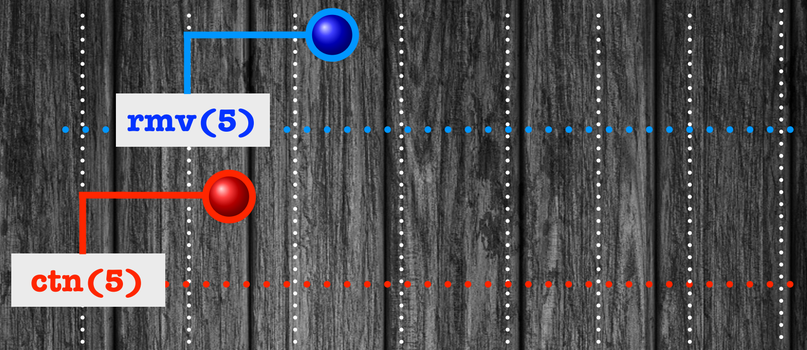
0

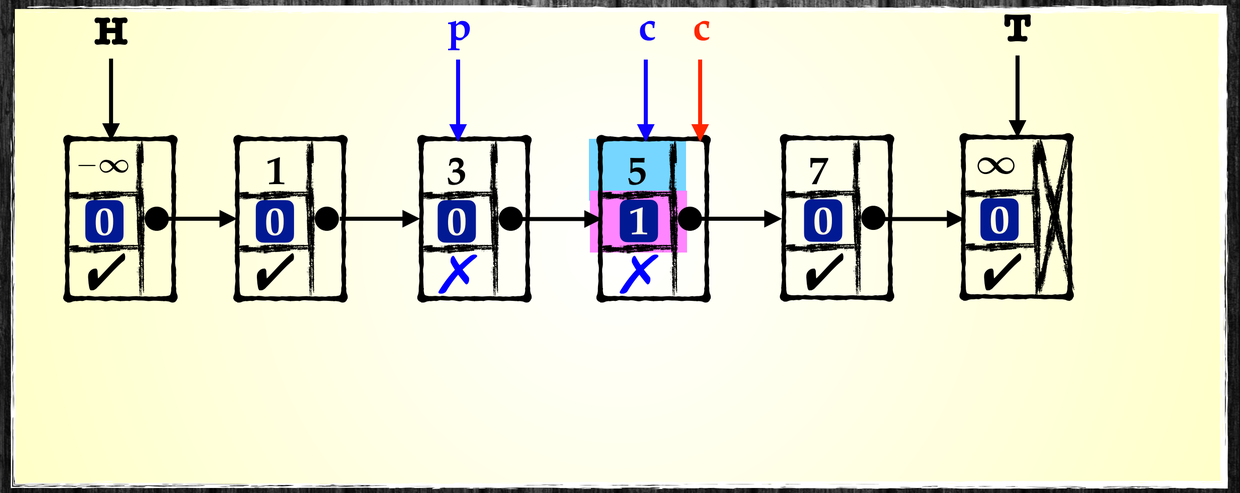
rmv(5)

```
rmv(e):
local p, c, n, r
1  (p,c) := locate(e);
2  if (c.val = e)
3     c.mark := true
4     n := c.next;
5     p.next := n;
6     r := true;
7  else r := false;
8  unlock(p);
9  unlock(c);
10 return r;
```

Scenario 2

- what if “mark” is first read
- and then set?





ctn(5)

```
ctn(e):
local c, b
1  c := Head;
2  while (c.val < e)
3     c := c.next
4  b := c.mark
5  if (!b && c.val = e)
6     return true;
7  else return false;
```

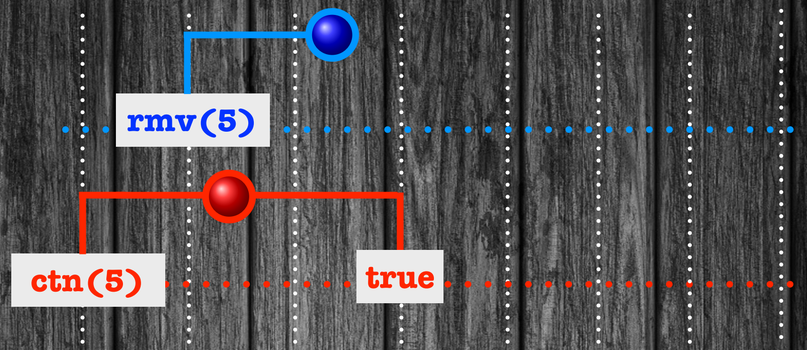
0

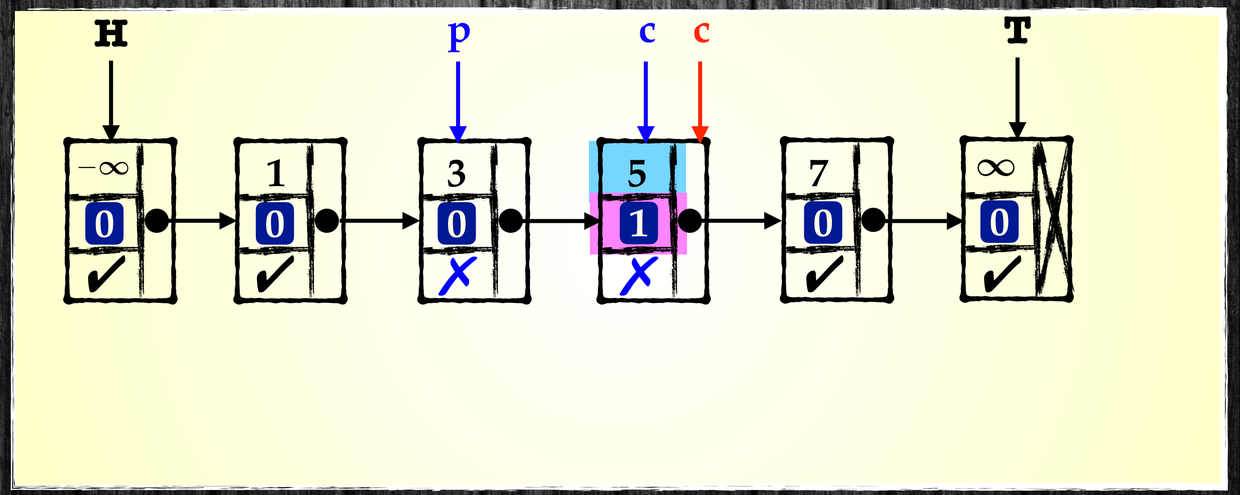
rmv(e): rmv(5)

```
local p, c, n, r
1  (p,c) := locate(e);
2  if (c.val = e)
3     c.mark := true
4     n := c.next;
5     p.next := n;
6     r := true;
7  else r := false;
8  unlock(p);
9  unlock(c);
10 return r;
```

Scenario 2

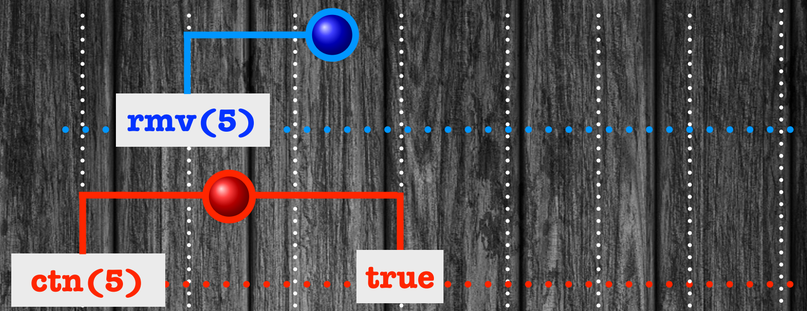
- what if “mark” is first read
- and then set?





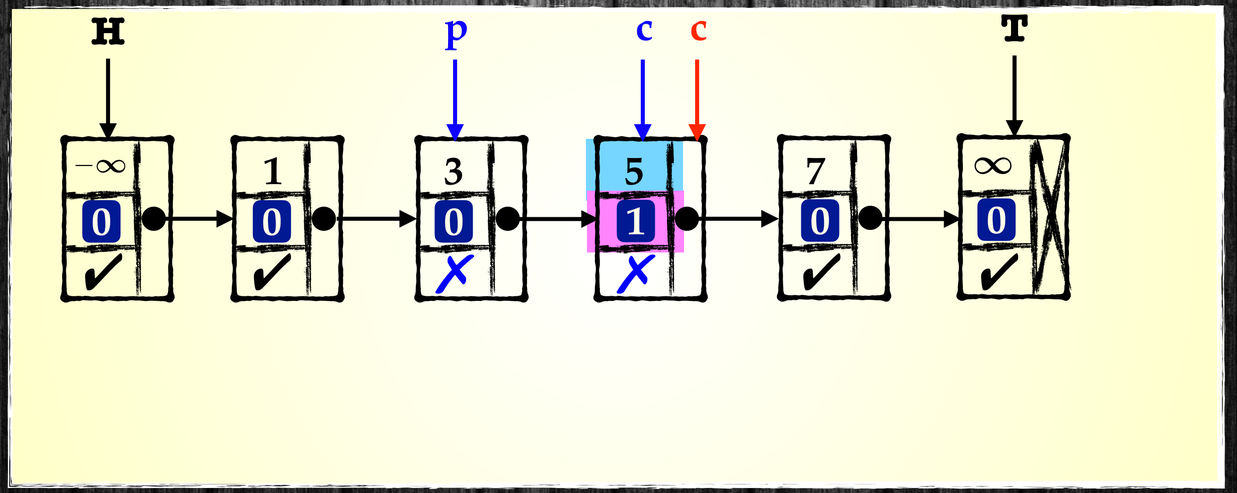
Scenario 2

- what if “mark” is first read
- and then set?



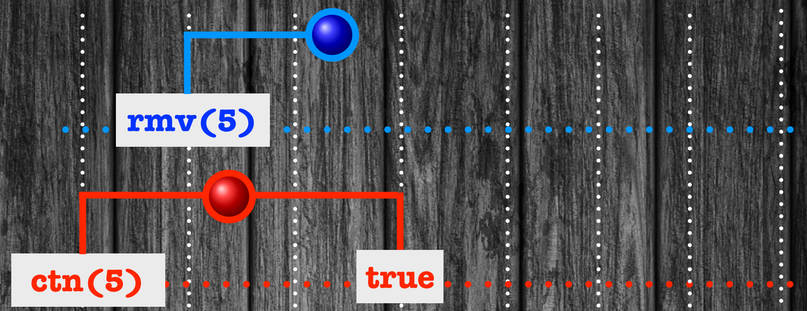
rmv(5)

```
rmv(e):
local p, c, n, r
1  (p,c) := locate(e);
2  if (c.val = e)
3     c.mark := true
4     n := c.next;
5     p.next := n;
6     r := true;
7  else r := false;
8  unlock(p);
9  unlock(c);
10 return r;
```



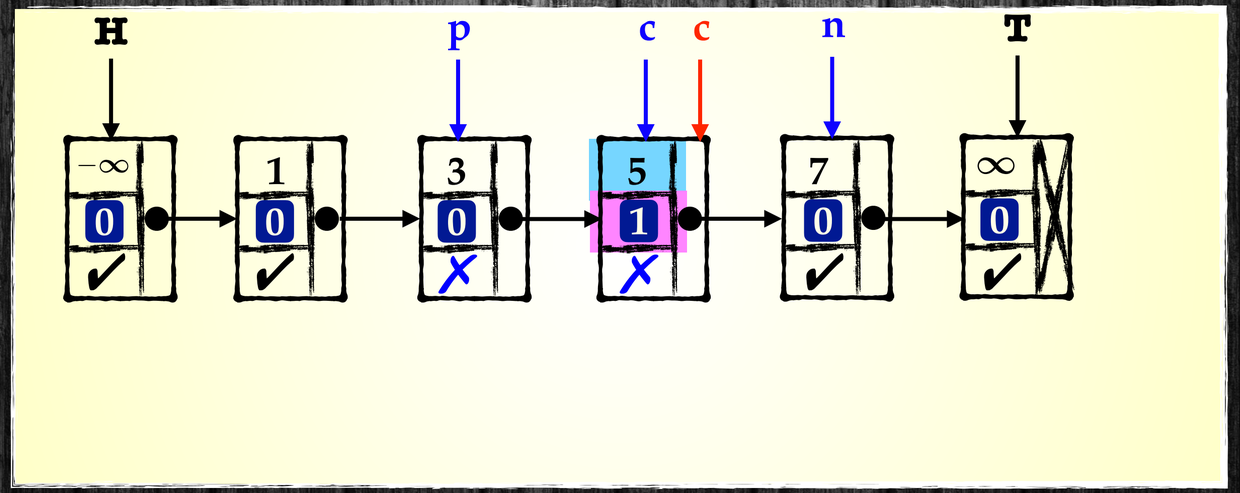
Scenario 2

- what if “mark” is first read
- and then set?



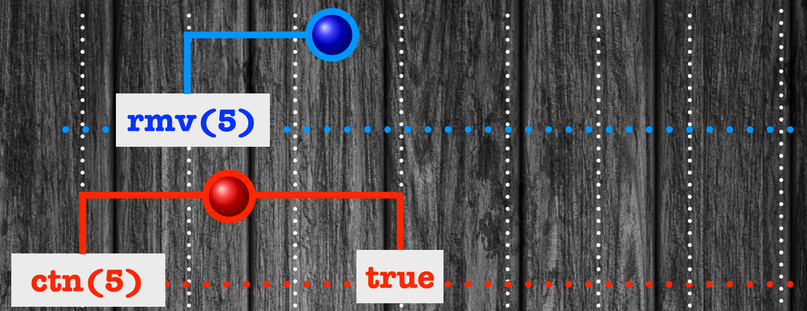
rmv(5)

```
rmv(e):
local p, c, n, r
1  (p,c) := locate(e);
2  if (c.val = e)
3     c.mark := true
4     n := c.next;
5     p.next := n;
6     r := true;
7  else r := false;
8  unlock(p);
9  unlock(c);
10 return r;
```



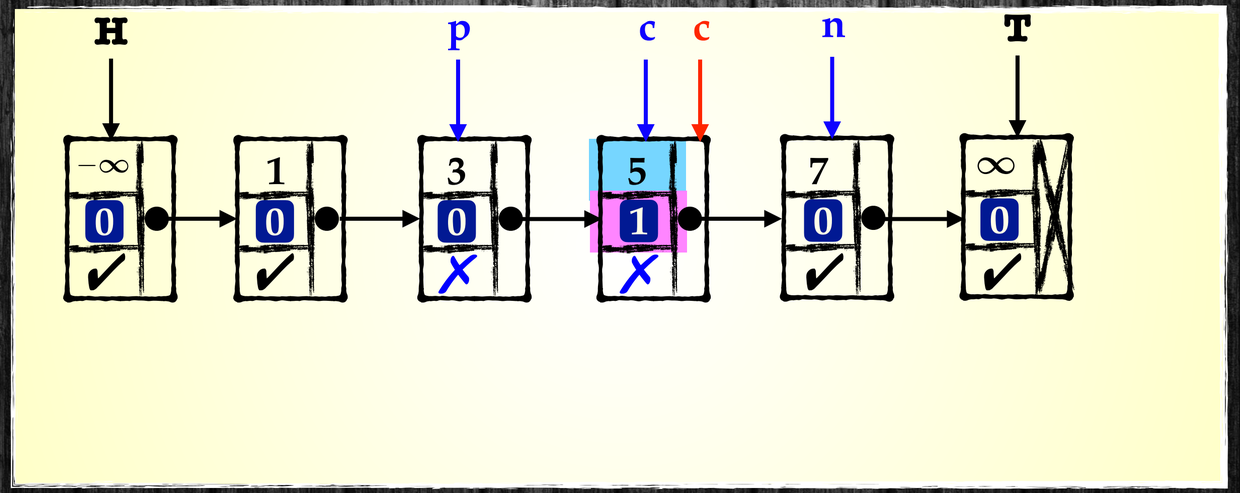
Scenario 2

- what if “mark” is first read
- and then set?



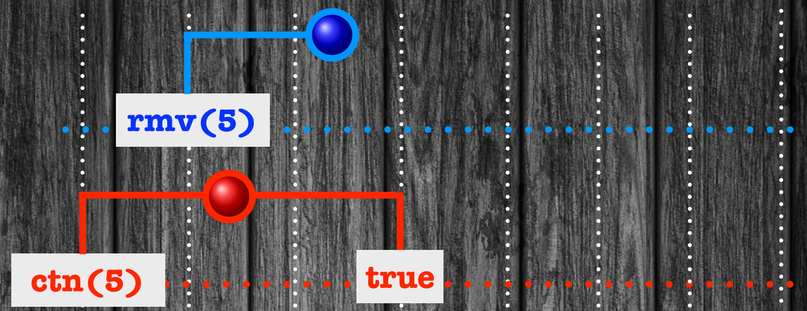
rmv(5)

```
rmv(e):
local p, c, n, r
1  (p,c) := locate(e);
2  if (c.val = e)
3     c.mark := true
4     n := c.next;
5     p.next := n;
6     r := true;
7  else r := false;
8  unlock(p);
9  unlock(c);
10 return r;
```



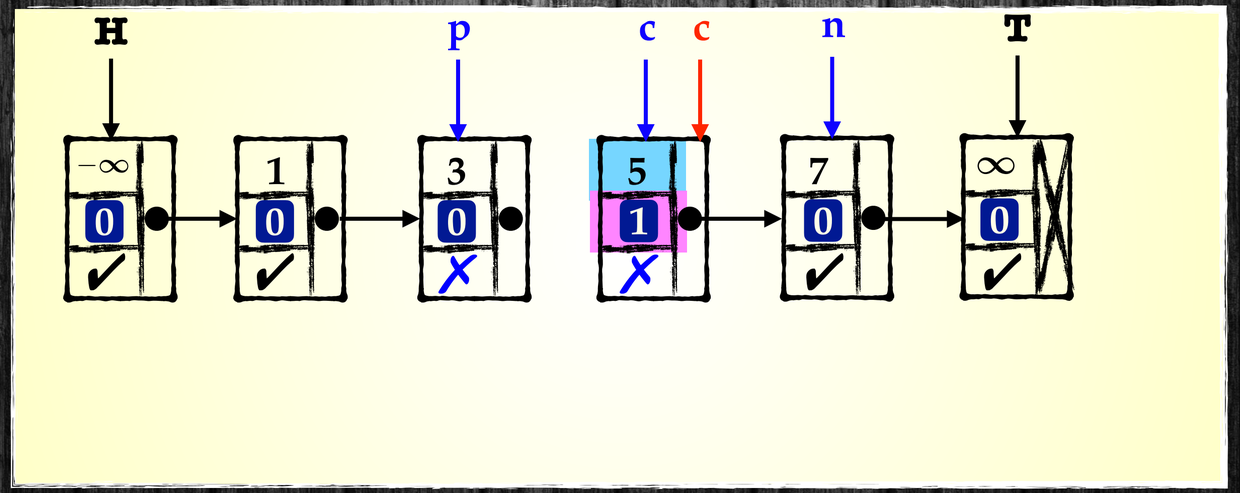
Scenario 2

- what if “mark” is first read
- and then set?



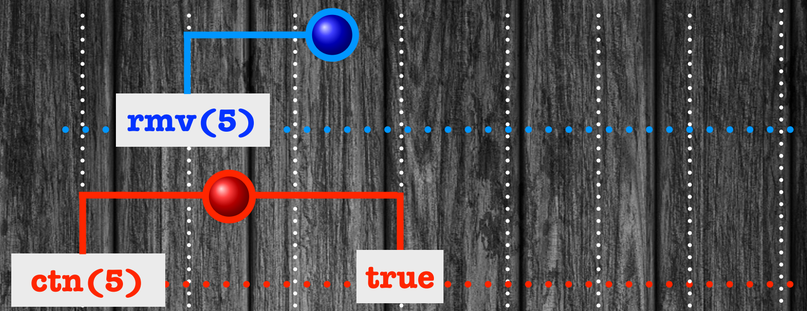
rmv(5)

```
rmv(e):
local p, c, n, r
1  (p,c) := locate(e);
2  if (c.val = e)
3     c.mark := true
4     n := c.next;
5     p.next := n;
6     r := true;
7  else r := false;
8  unlock(p);
9  unlock(c);
10 return r;
```



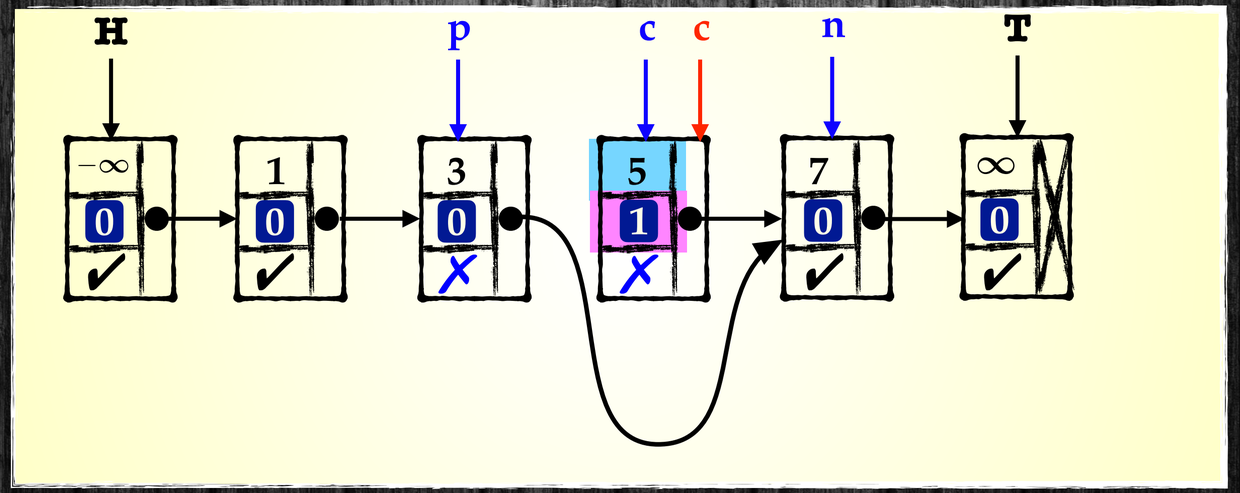
Scenario 2

- what if “mark” is first read
- and then set?



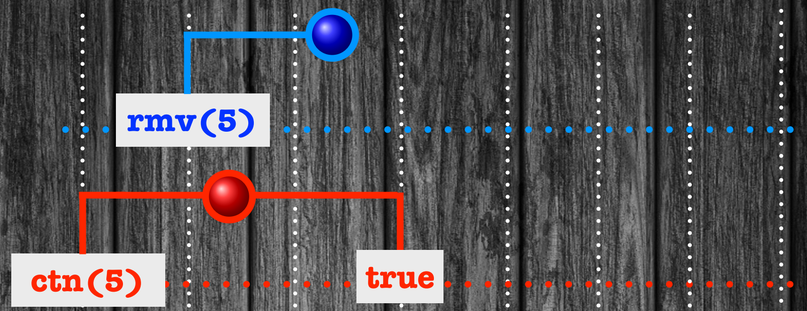
rmv(5)

```
rmv(e):
local p, c, n, r
1  (p,c) := locate(e);
2  if (c.val = e)
3     c.mark := true
4     n := c.next;
5     p.next := n;
6     r := true;
7  else r := false;
8  unlock(p);
9  unlock(c);
10 return r;
```



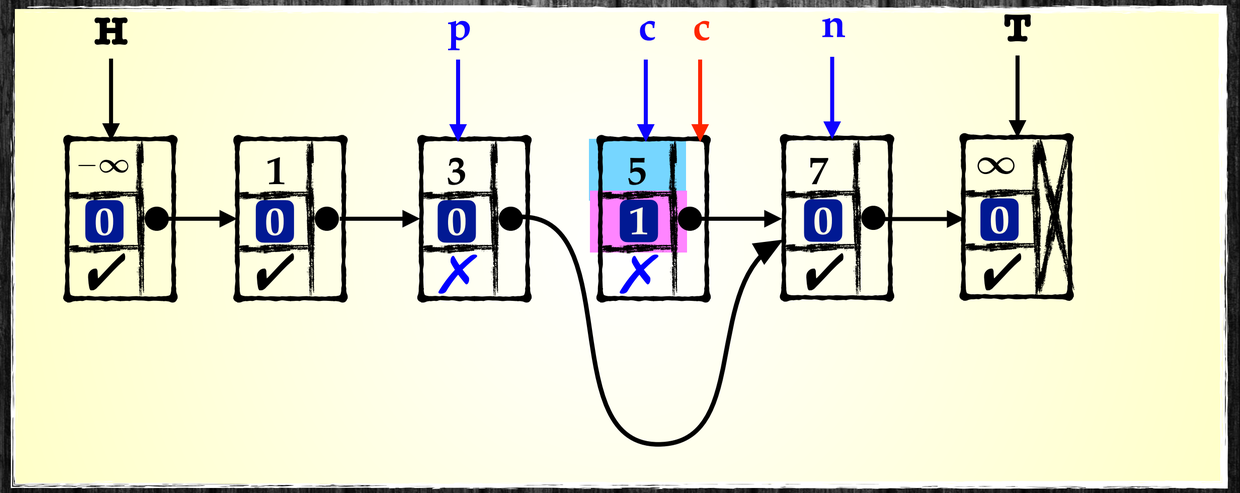
Scenario 2

- what if “mark” is first read
- and then set?



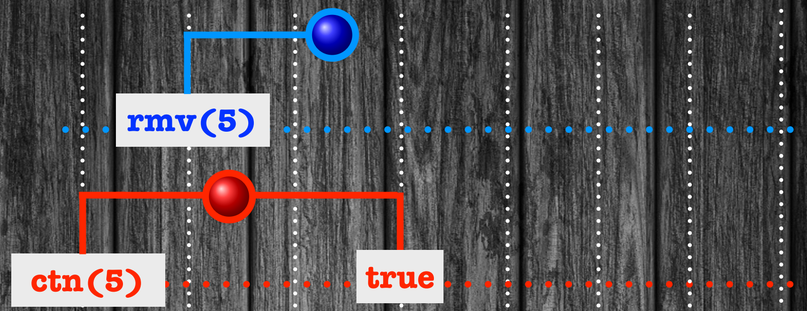
rmv(5)

```
rmv(e):
local p, c, n, r
1  (p,c) := locate(e);
2  if (c.val = e)
3     c.mark := true
4     n := c.next;
5     p.next := n;
6     r := true;
7  else r := false;
8  unlock(p);
9  unlock(c);
10 return r;
```



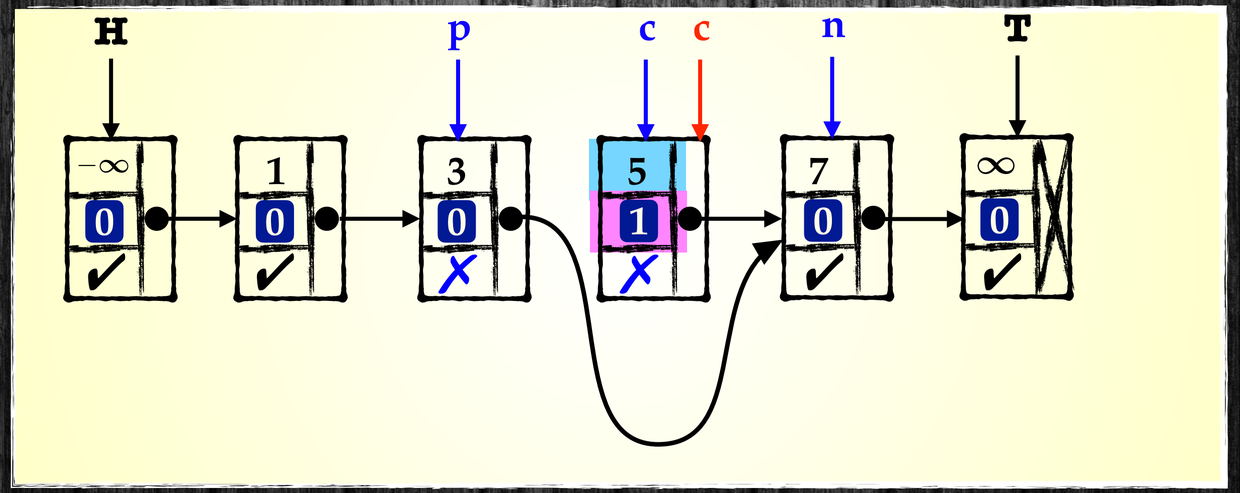
Scenario 2

- what if “mark” is first read
- and then set?



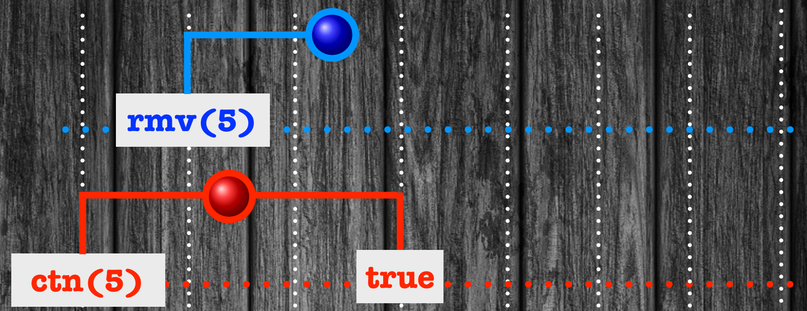
rmv(5)

```
rmv(e):
local p, c, n, r
1  (p,c) := locate(e);
2  if (c.val = e)
3     c.mark := true
4     n := c.next;
5     p.next := n;
6     r := true;
7  else r := false;
8  unlock(p);
9  unlock(c);
10 return r;
```



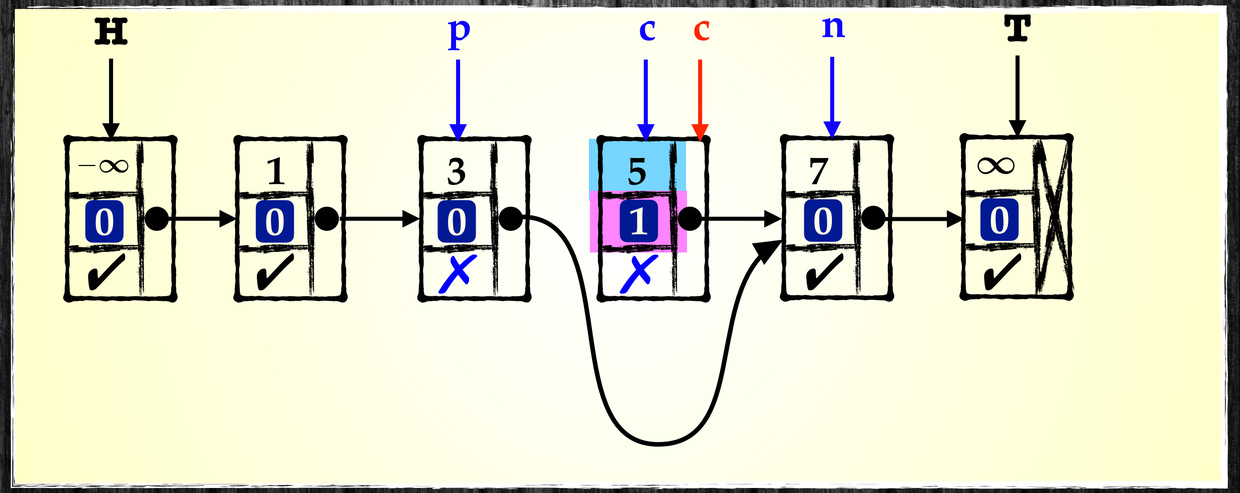
Scenario 2

- what if “mark” is first read
- and then set?



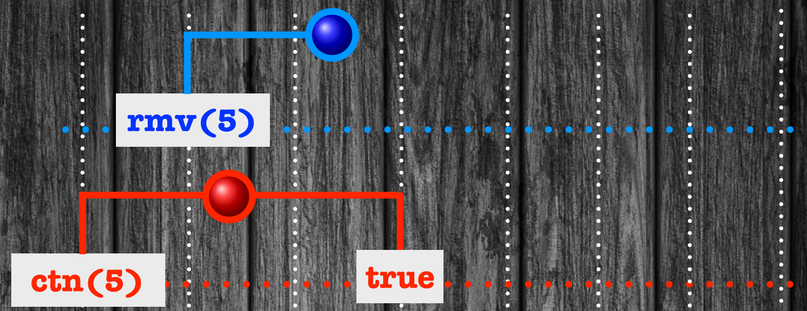
rmv(5)

```
rmv(e):
local p, c, n, r
1  (p,c) := locate(e);
2  if (c.val = e)
3     c.mark := true
4     n := c.next;
5     p.next := n;
6     r := true;
7  else r := false;
8  unlock(p);
9  unlock(c);
10 return r;
```



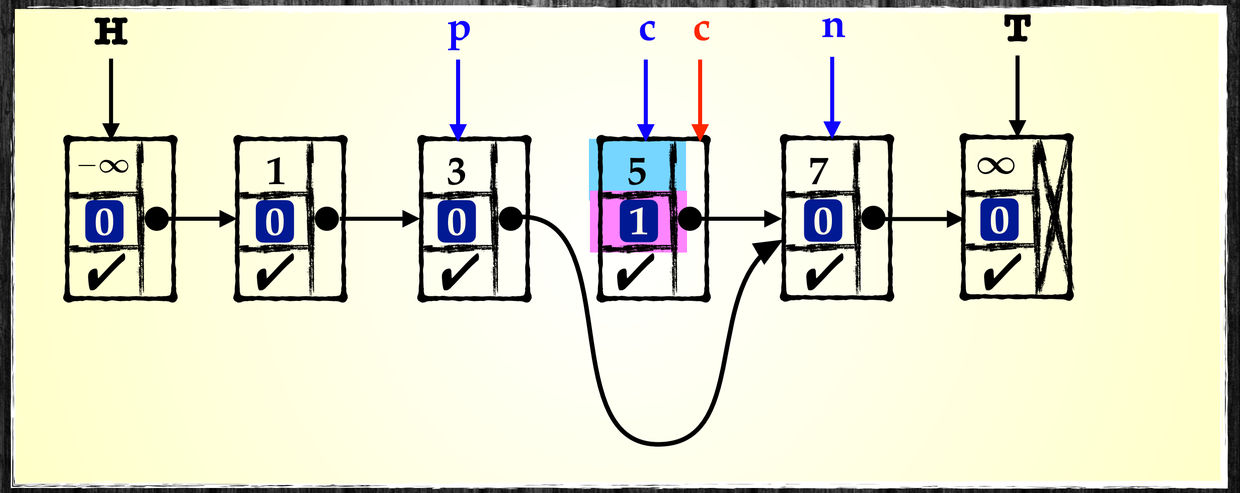
Scenario 2

- what if “mark” is first read
- and then set?



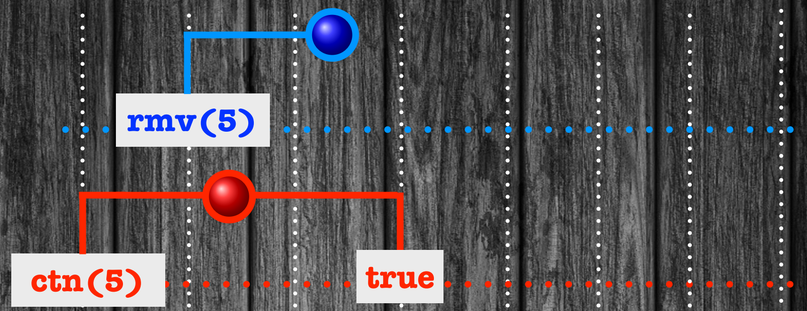
rmv(5)

```
rmv(e):
local p, c, n, r
1  (p,c) := locate(e);
2  if (c.val = e)
3     c.mark := true
4     n := c.next;
5     p.next := n;
6     r := true;
7  else r := false;
8  unlock(p);
9  unlock(c);
10 return r;
```



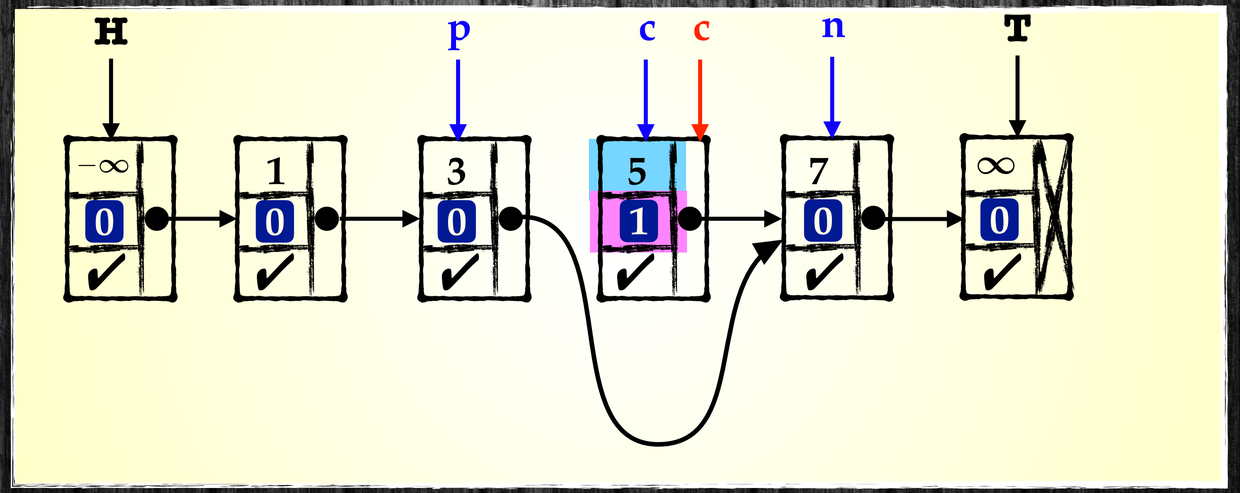
Scenario 2

- what if “mark” is first read
- and then set?



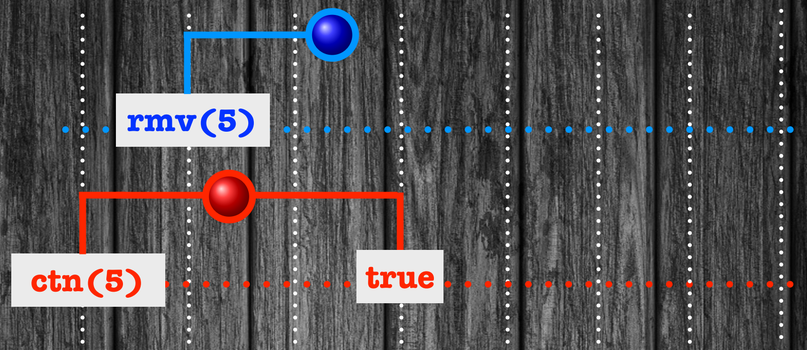
rmv(5)

```
rmv(e):
local p, c, n, r
1  (p,c) := locate(e);
2  if (c.val = e)
3     c.mark := true
4     n := c.next;
5     p.next := n;
6     r := true;
7  else r := false;
8  unlock(p);
9  unlock(c);
10 return r;
```



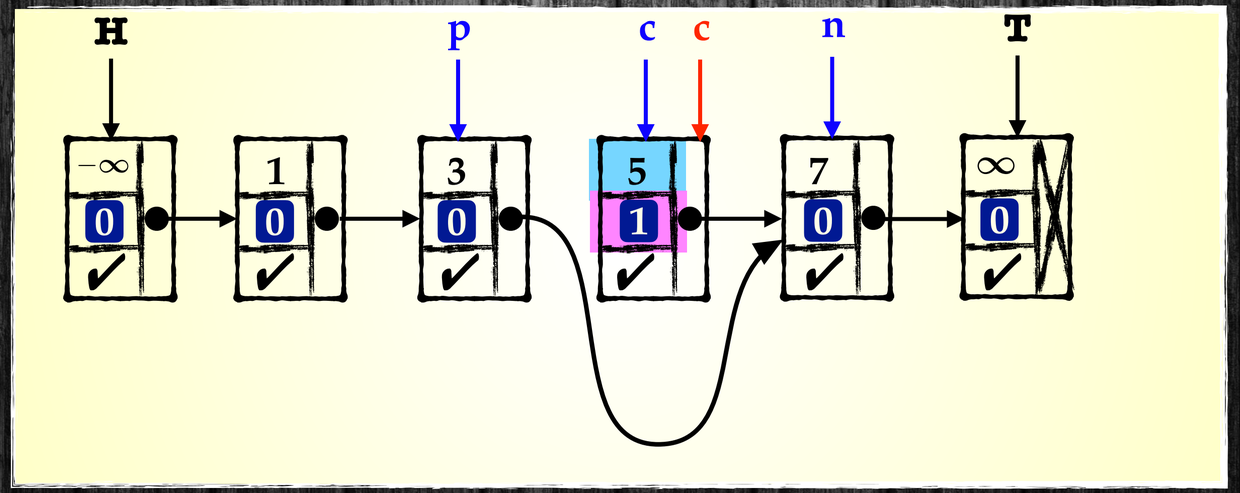
Scenario 2

- what if “mark” is first read
- and then set?



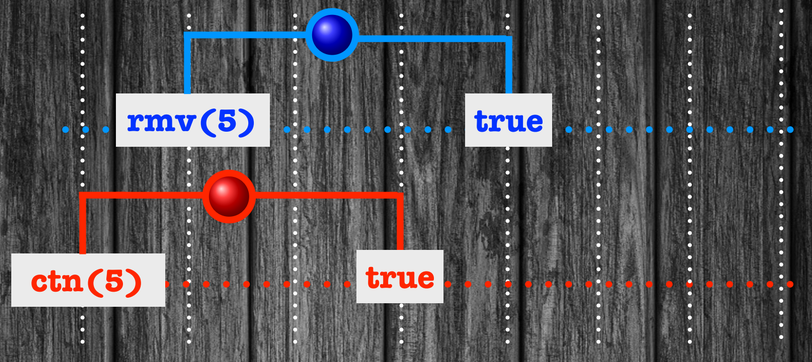
rmv(5)

```
rmv(e):
local p, c, n, r
1  (p,c) := locate(e);
2  if (c.val = e)
3     c.mark := true
4     n := c.next;
5     p.next := n;
6     r := true;
7  else r := false;
8  unlock(p);
9  unlock(c);
10 return r;
```



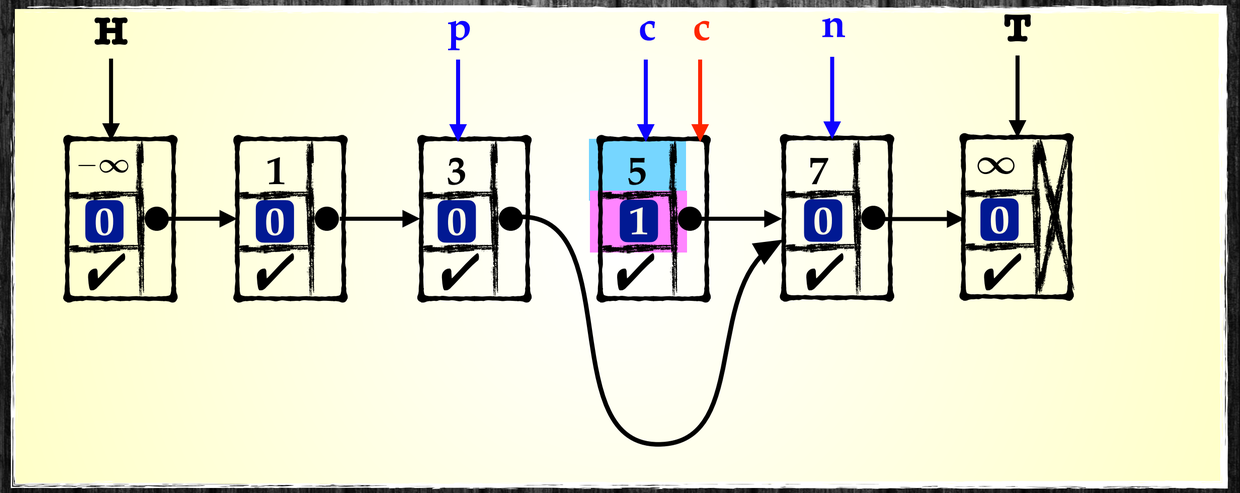
Scenario 2

- what if “mark” is first read
- and then set?



rmv(5)

```
rmv(e):
local p, c, n, r
1  (p,c) := locate(e);
2  if (c.val = e)
3     c.mark := true
4     n := c.next;
5     p.next := n;
6     r := true;
7  else r := false;
8  unlock(p);
9  unlock(c);
10 return r;
```



Scenario 2

- what if “mark” is first read
- and then set?

