Concurrent Algorithms and Data Structures

Concurrent Algorithms and Data Structures

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

Concurrent Algorithms and Data Structures

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

- The first theory assignment is available.
  - Deadline on Sunday.
  - No extension possible! (for fairness reasons)
- The first code assignment is available.

Outline

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

Overview



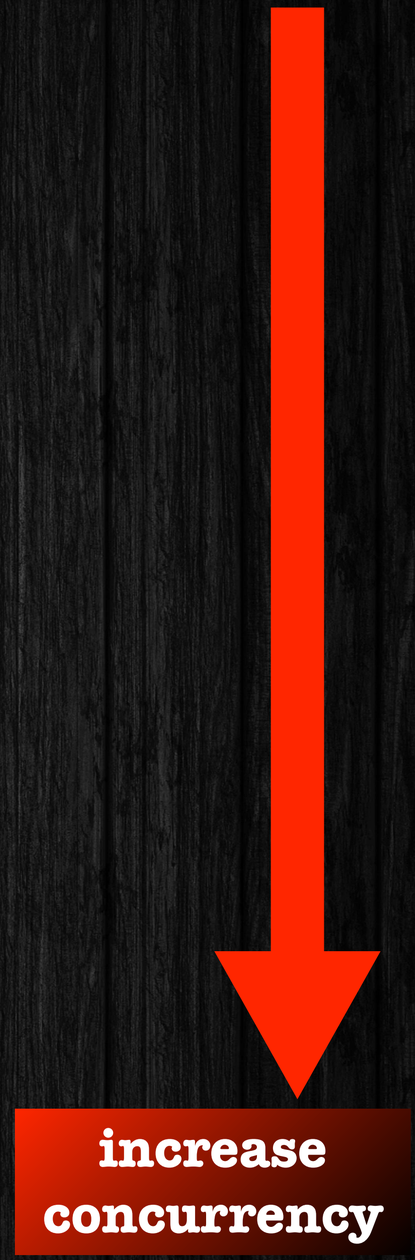
Overview

- Sequential Programs



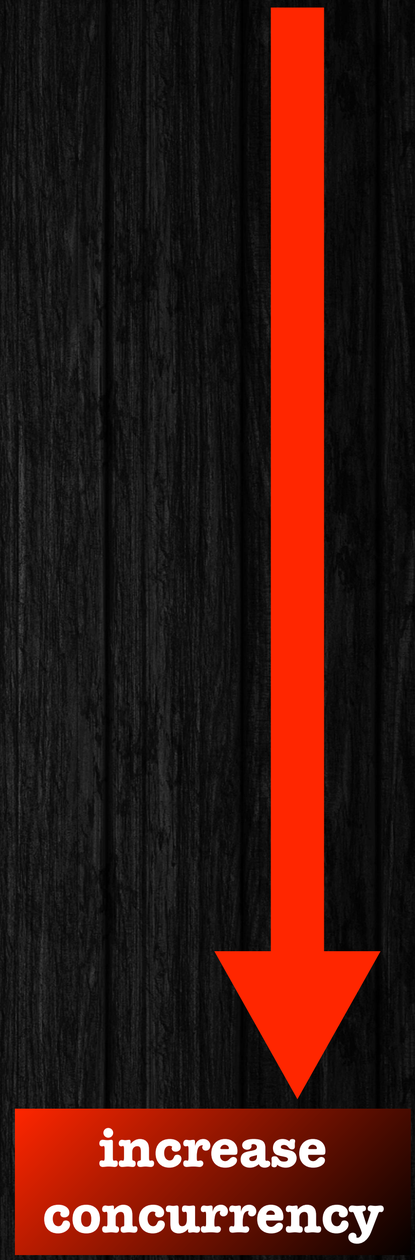
Overview

- Sequential Programs
- Coarse-Grained Synchronization



Overview

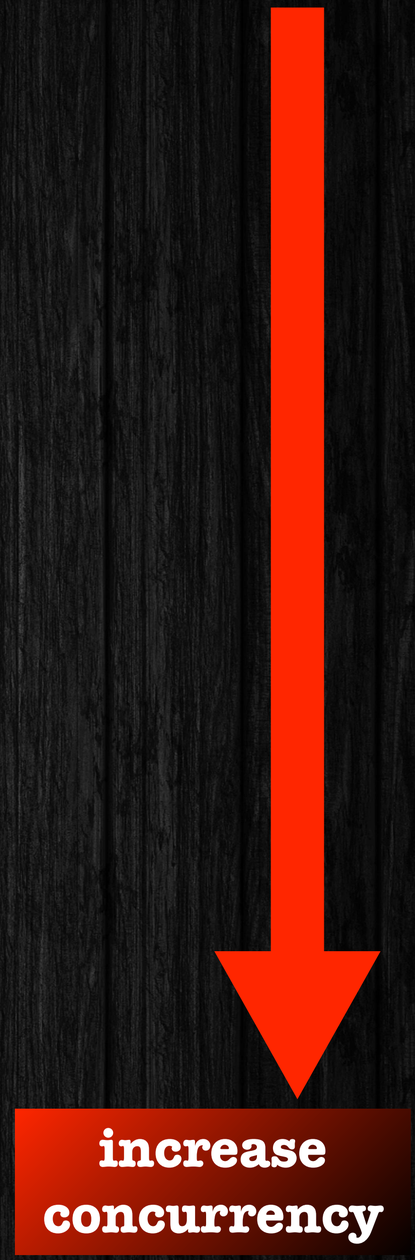
- Sequential Programs
- Coarse-Grained Synchronization
- Single global lock



Overview

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

- Single global lock



Overview

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



Overview

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



Overview

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



Overview

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



Overview

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



Overview

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



Overview

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



Outline

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

Outline

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

Goal

even more concurrency

Sets | Lazy Synchronization

Goal — even more concurrency

How? — ctn need not acquire locks

Sets | Lazy Synchronization

Goal — even more concurrency

How? — ctn need not acquire locks

Typical application:

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

Sets | Lazy Synchronization

Goal — even more concurrency

How? — ctn need not acquire locks

Strategy — logical deletion

Typical application:

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

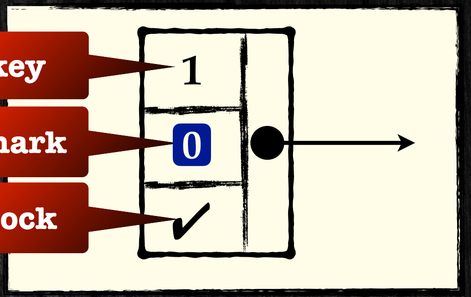
logically deleted?

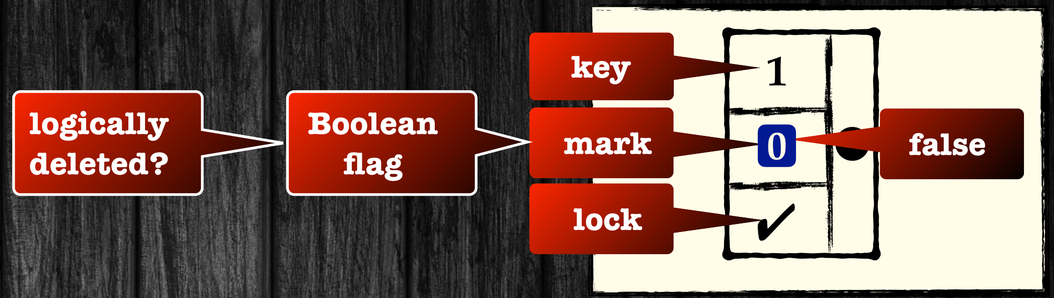
Boolean flag

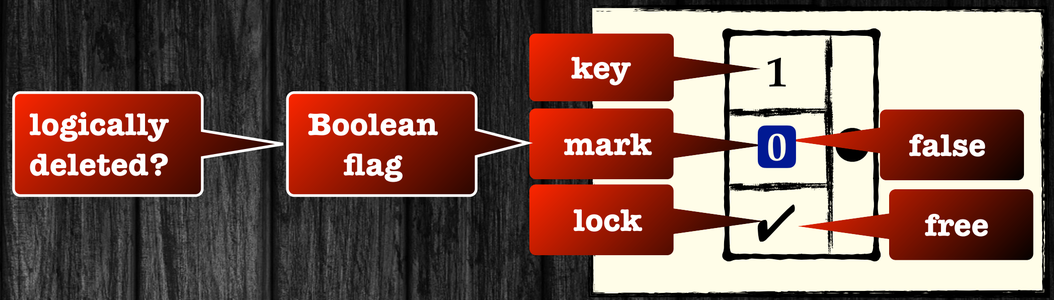
key

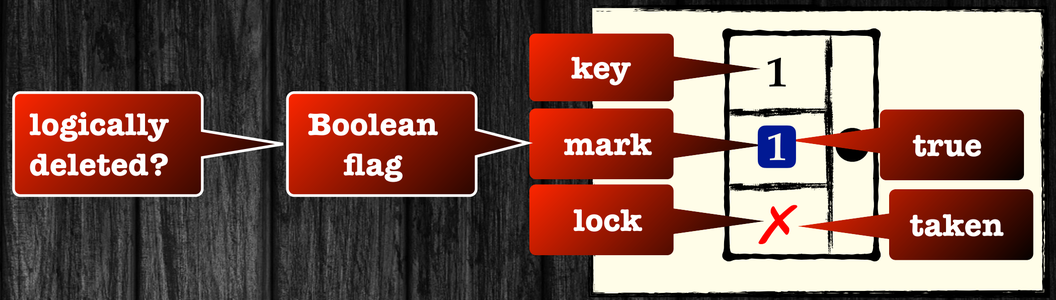
mark

lock

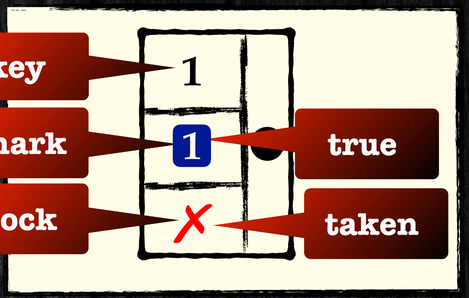



logically deleted?
Boolean flag
key
mark
lock
1
0
✓
false


logically deleted?
Boolean flag
key
mark
lock
1
0
✓
false
free


logically deleted?
Boolean flag
key
mark
lock
1
1
true
taken

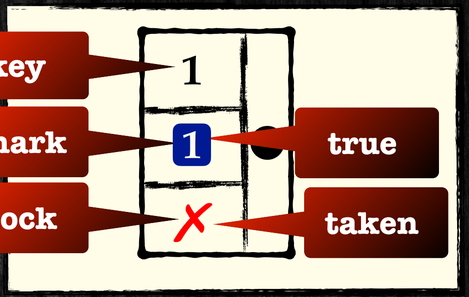
logically deleted? → Boolean flag



rmv

1. delete logically
2. delete physically

logically deleted? → Boolean flag

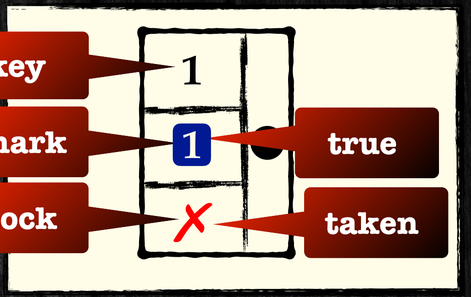


set mark = 1

rmv

1. delete logically
2. delete physically

logically deleted? → Boolean flag



set mark = 1

rmv

1. delete logically
2. delete physically

ctn returns false if:

1. node logically deleted

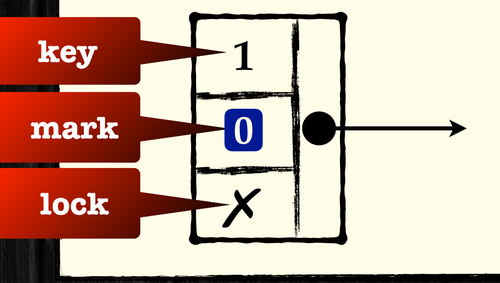
or

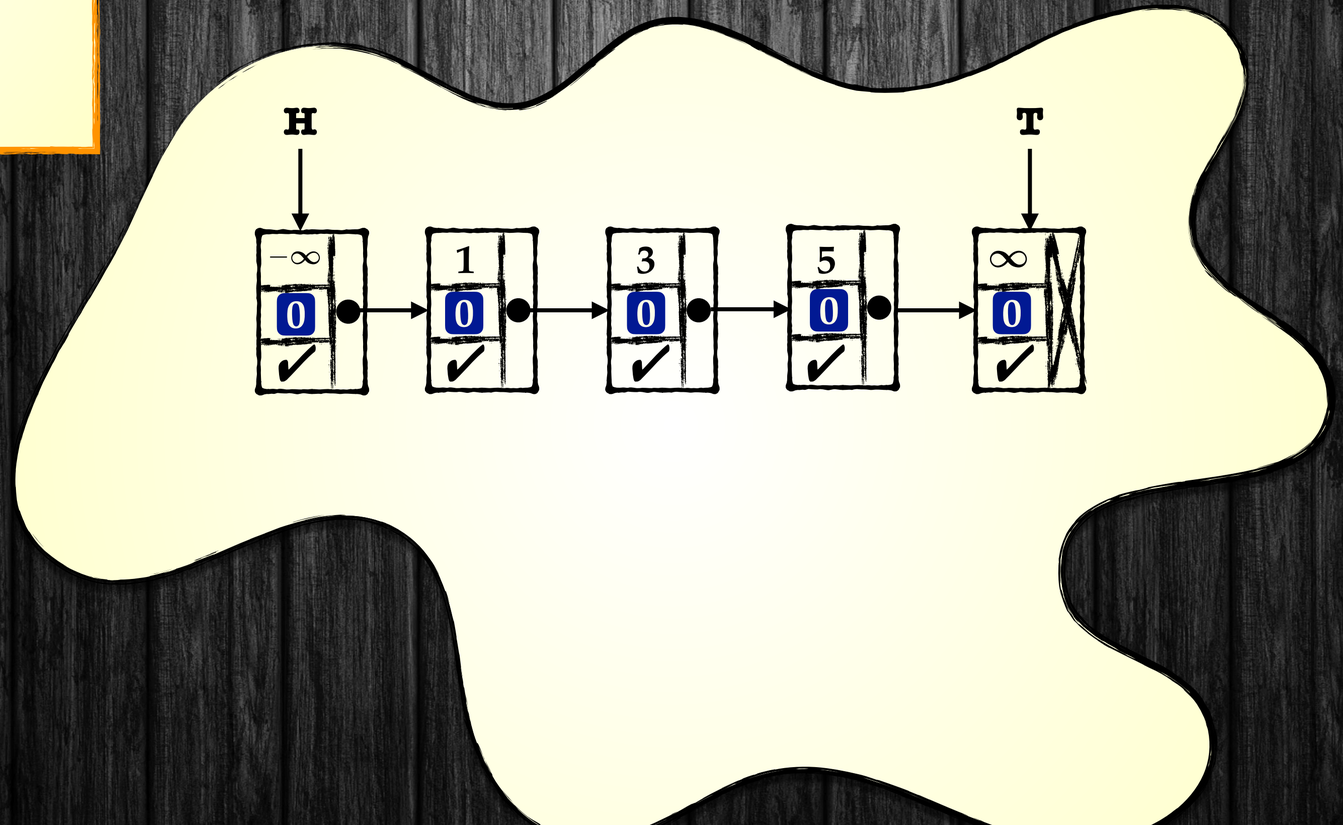
2. node absent

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)

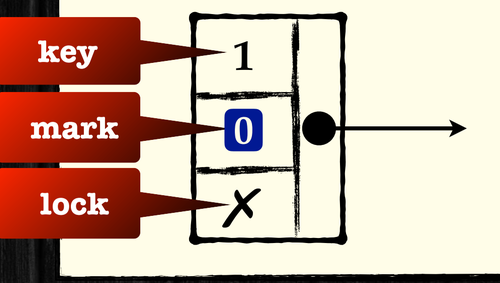


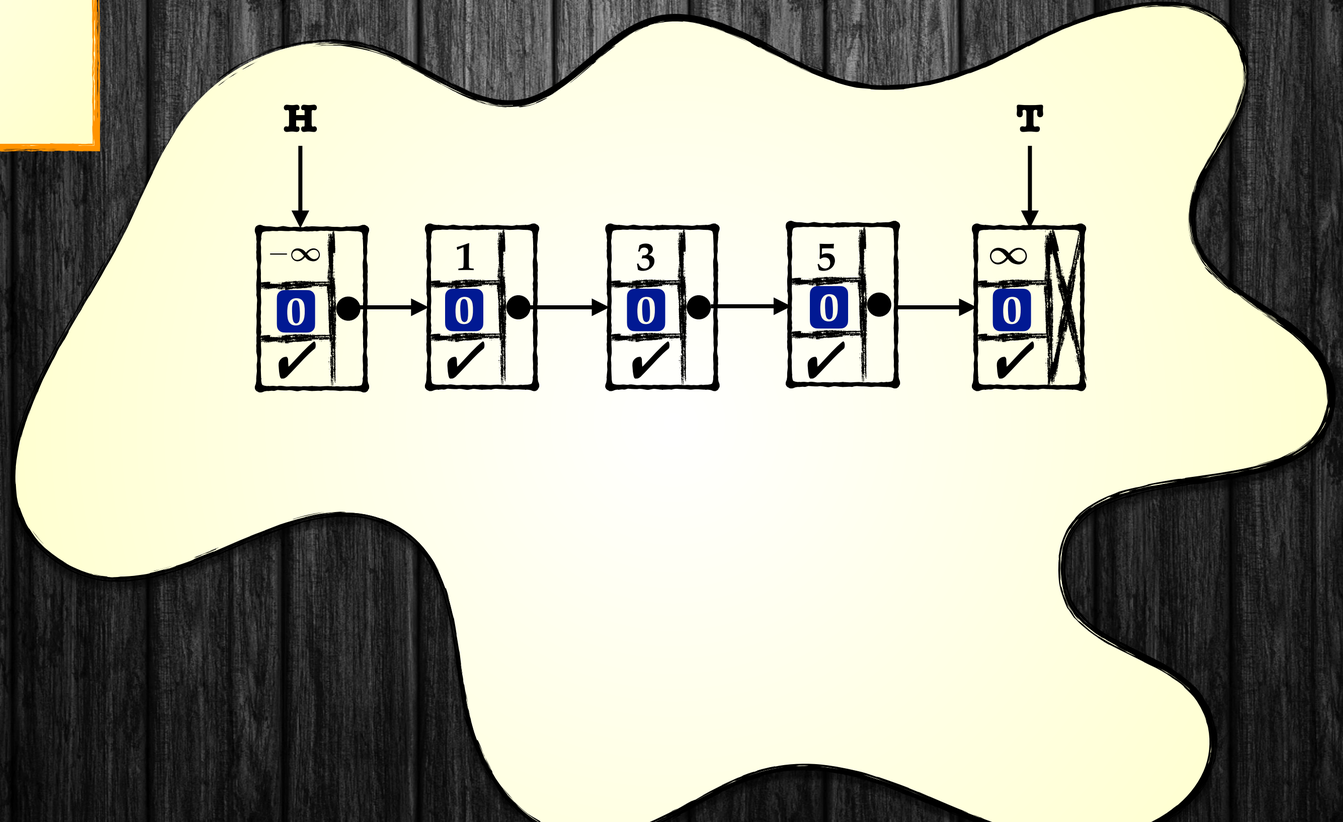


Sets | Lazy Synchronization

```
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)

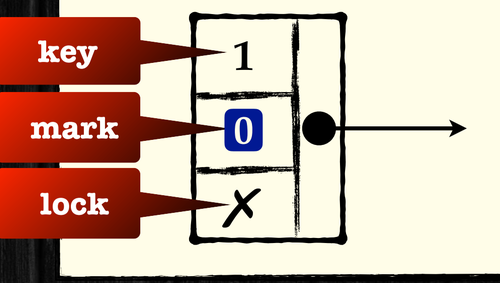


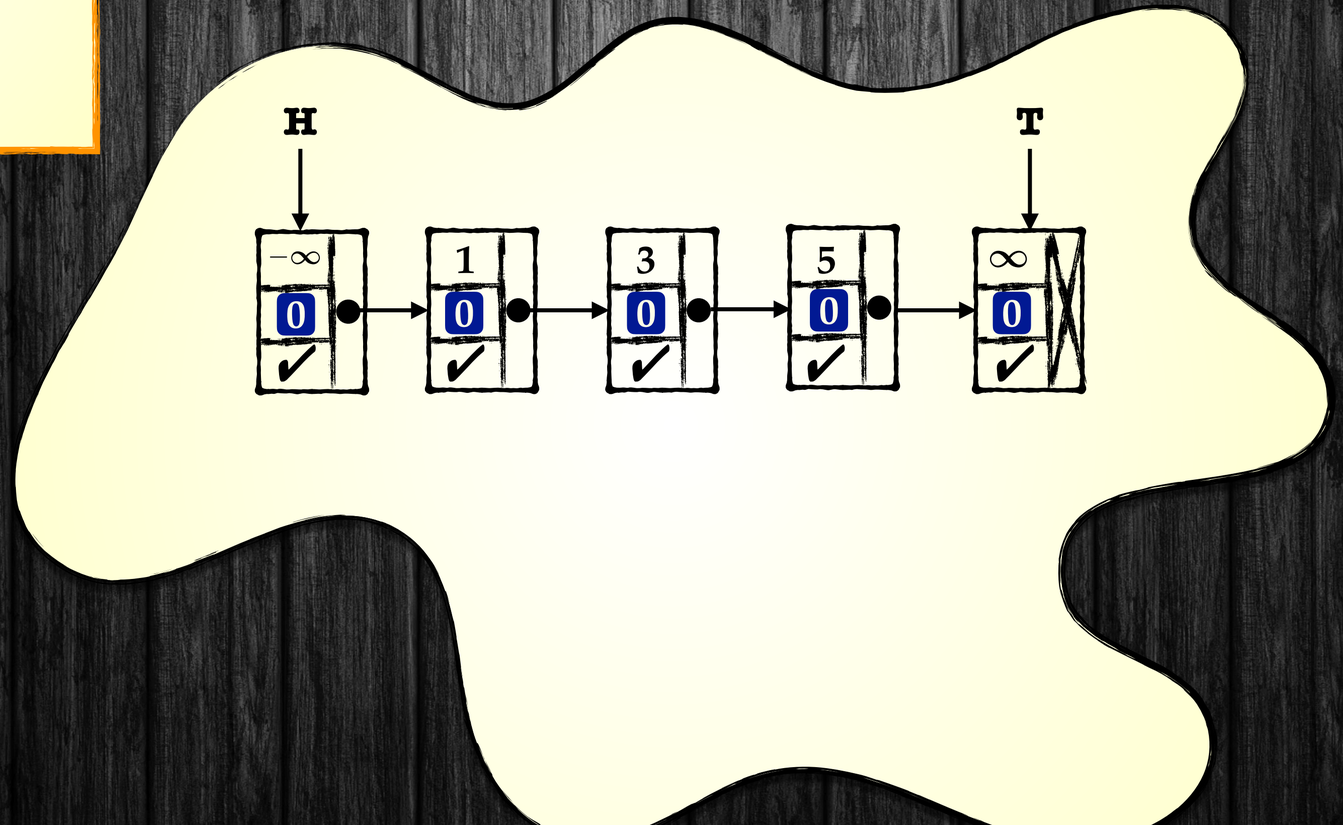


Sets | Lazy Synchronization

```
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)

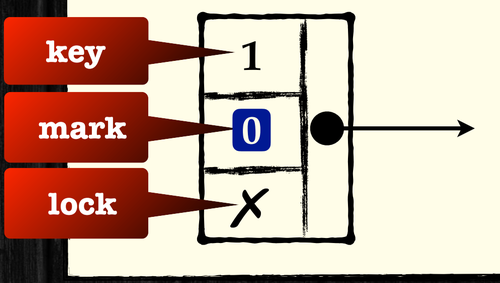


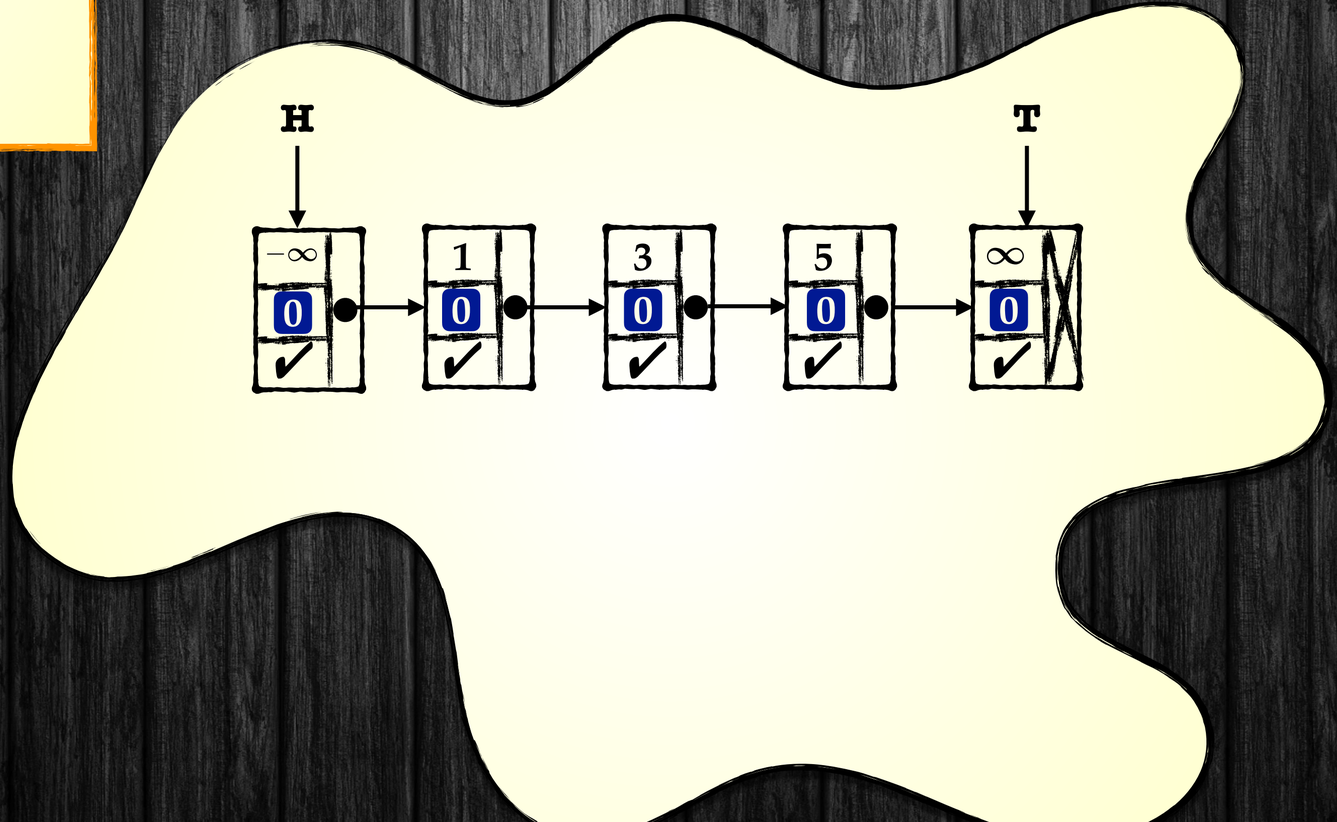


Sets | Lazy Synchronization

```
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);
```



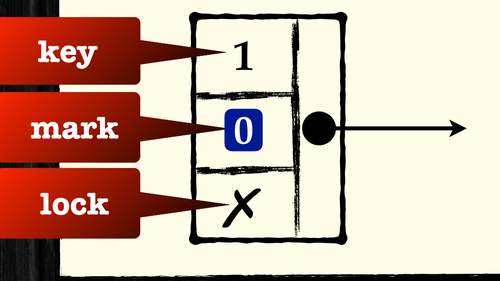


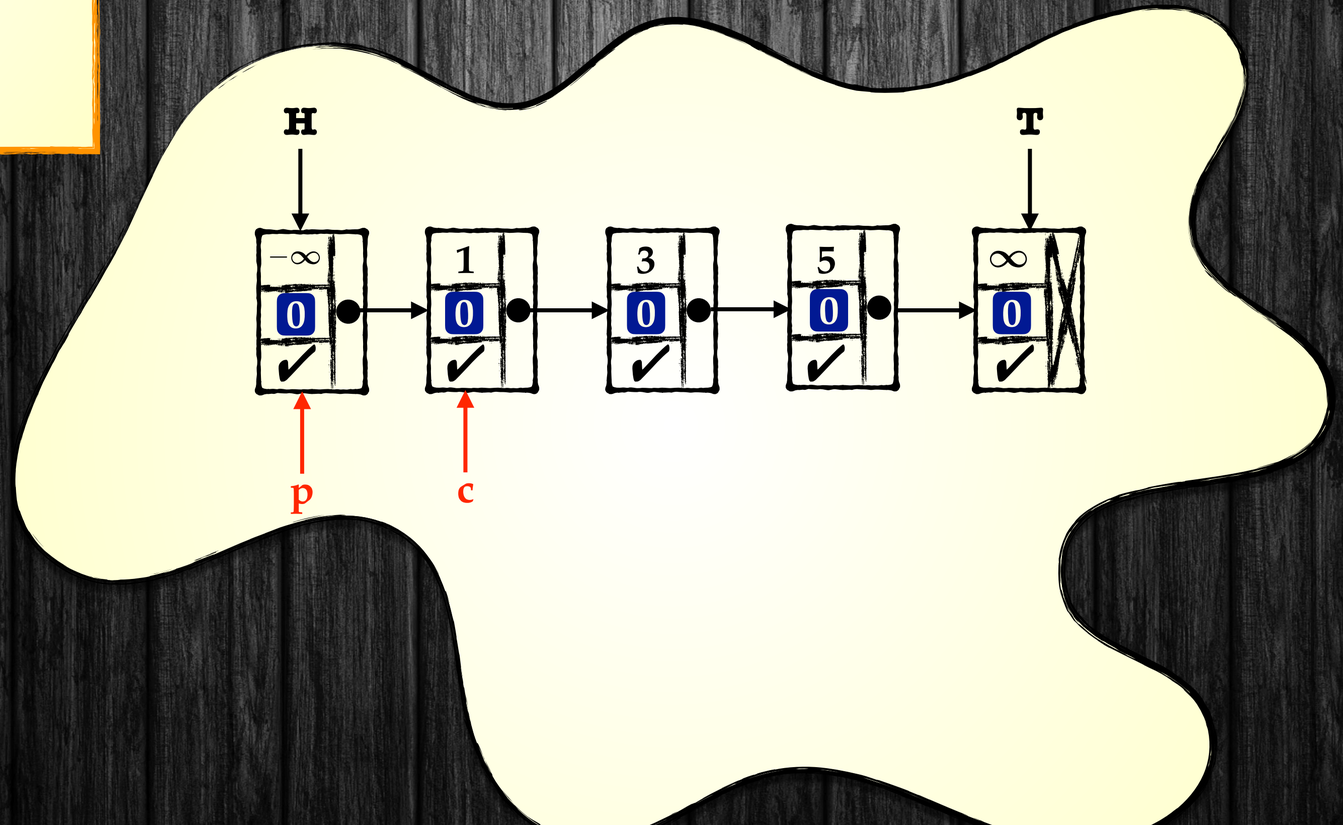


Sets | Lazy Synchronization

```
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)

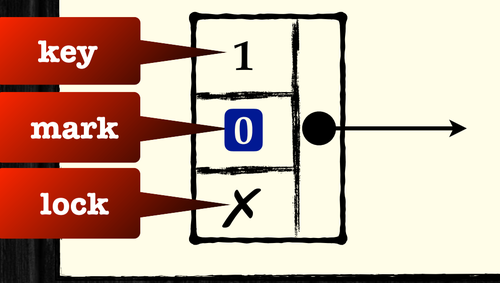


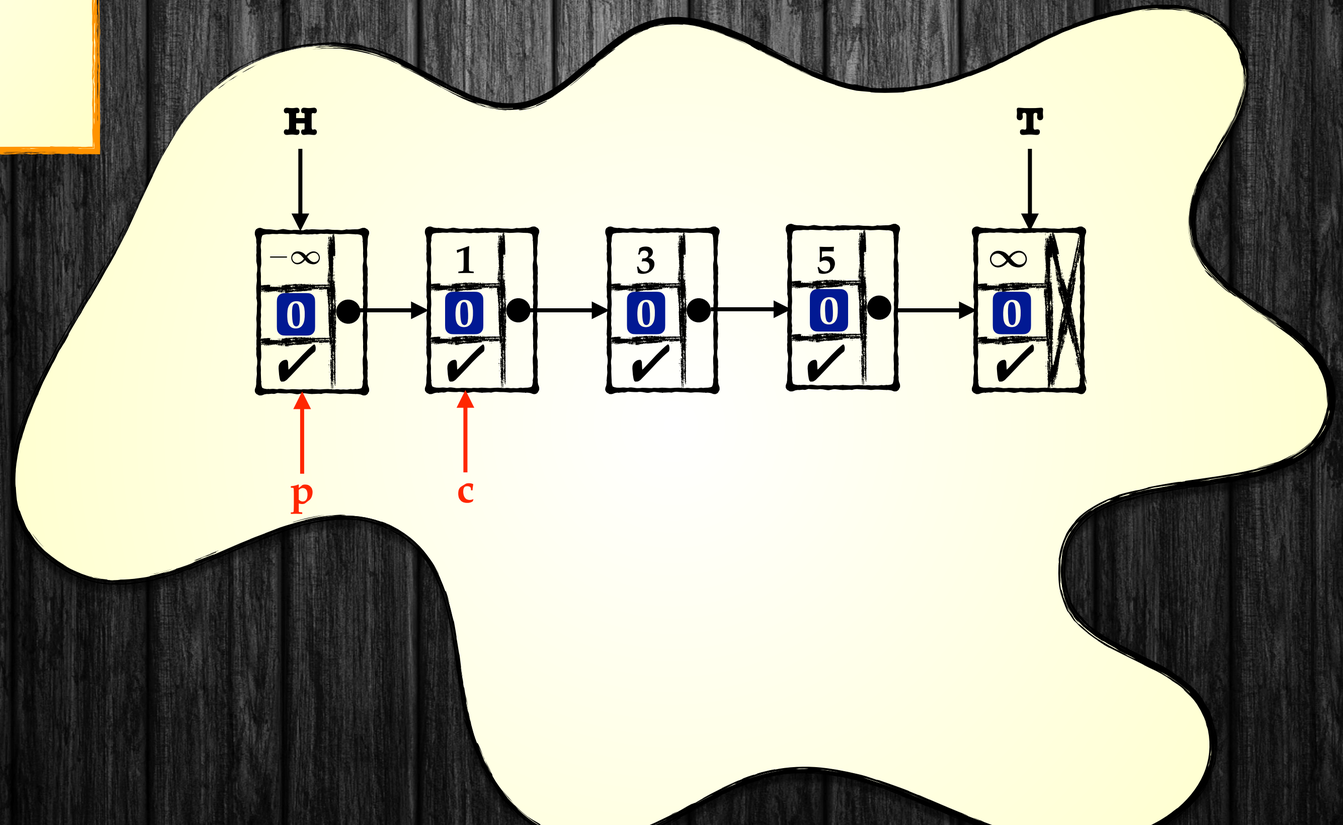


Sets | Lazy Synchronization

```
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)

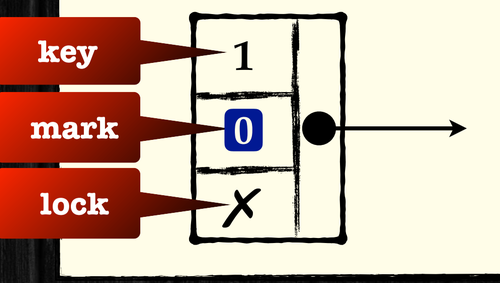


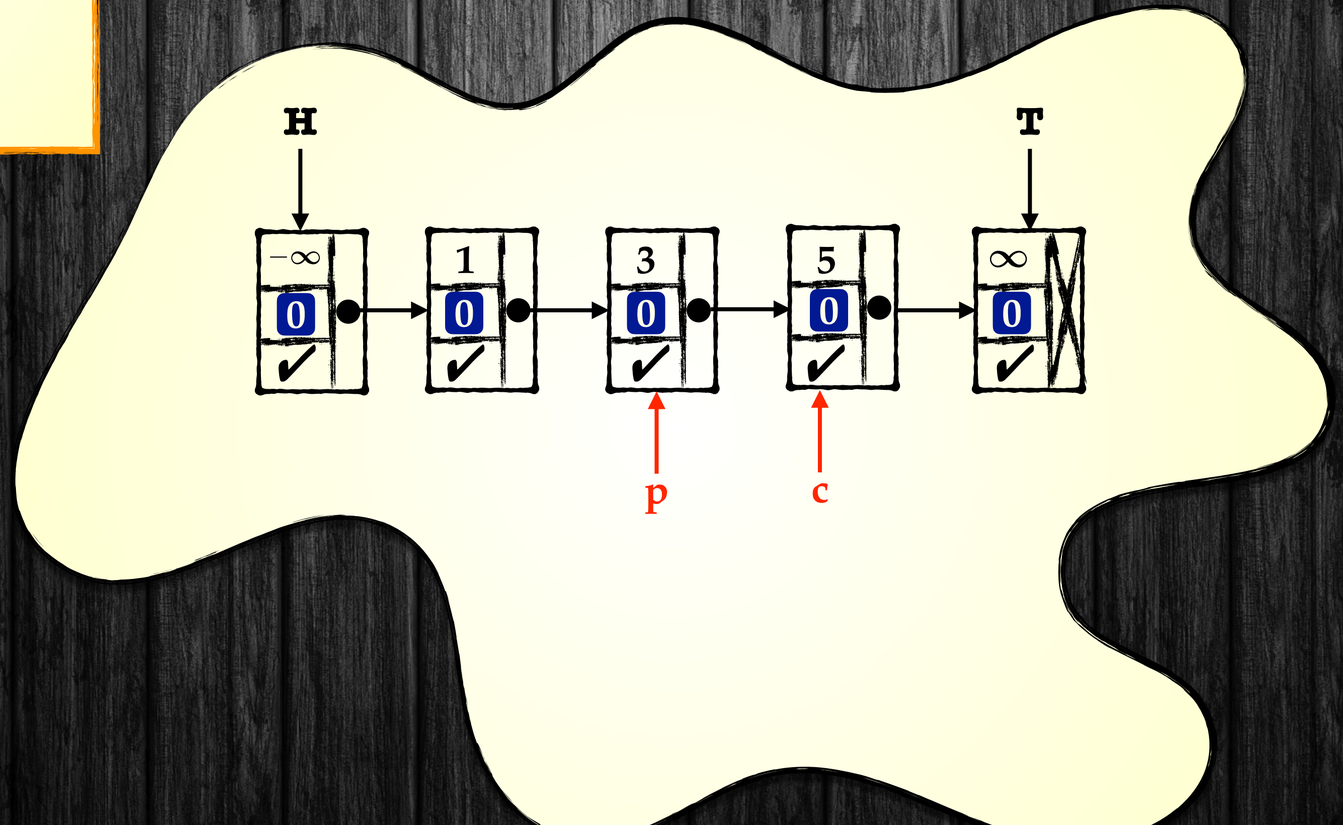


Sets | Lazy Synchronization

```
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)

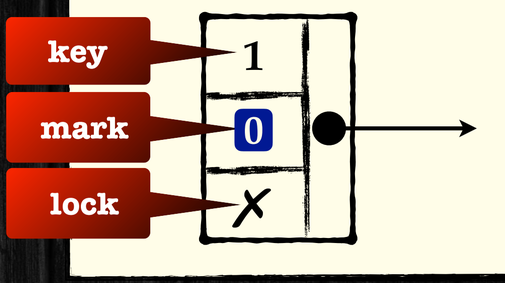


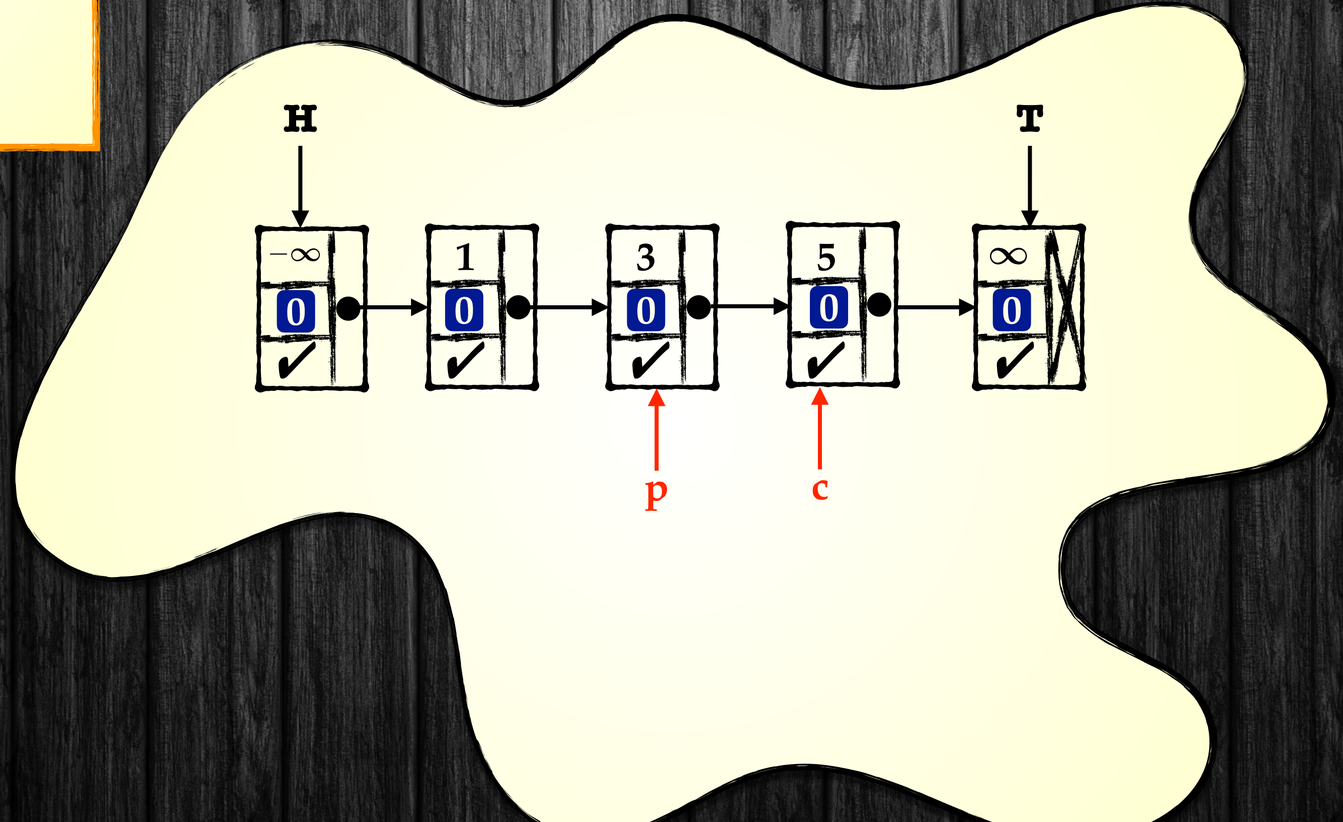


Sets | Lazy Synchronization

```
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);
```



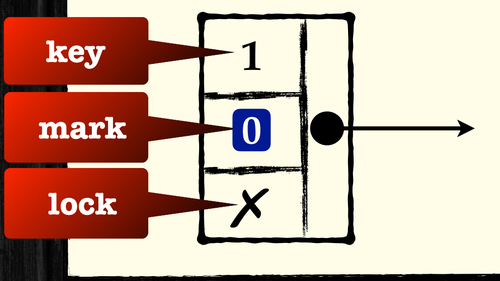


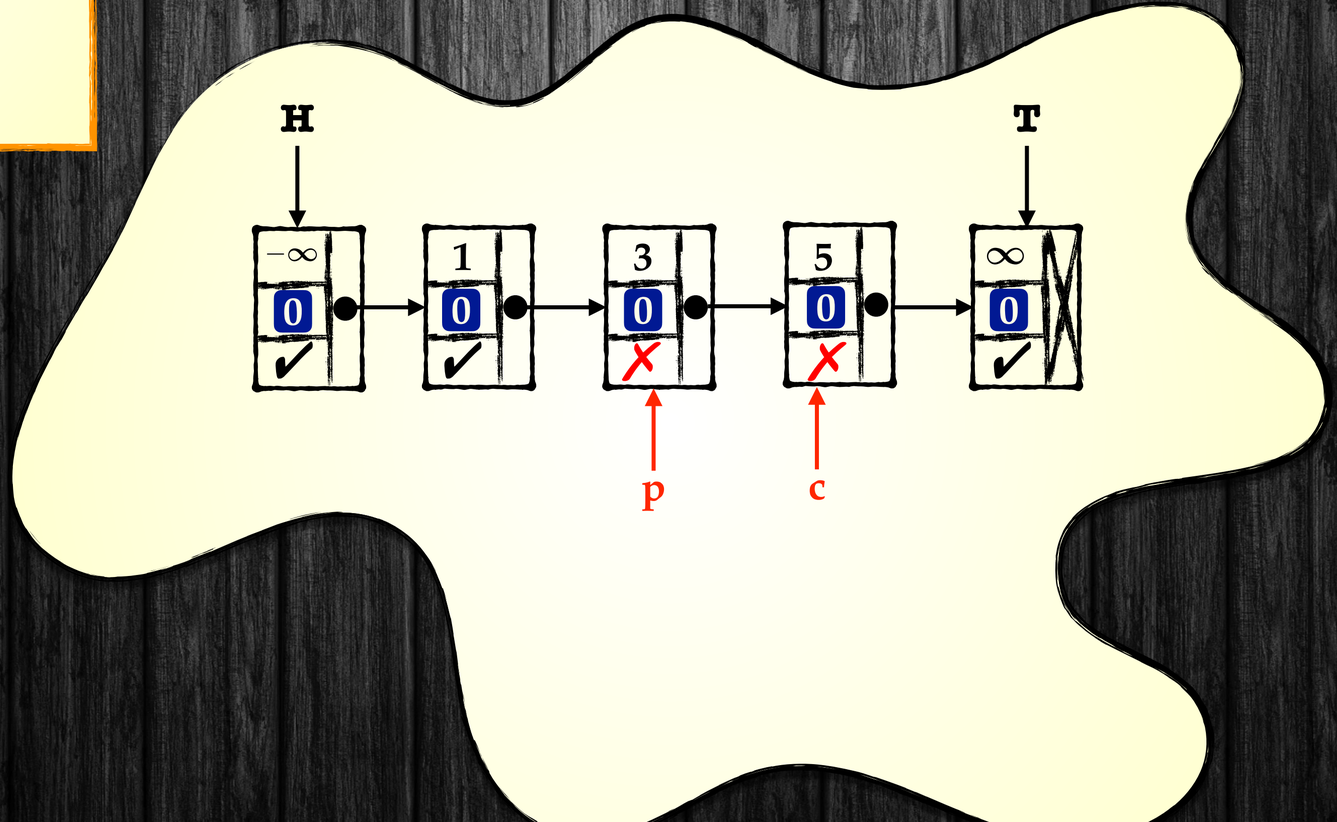


Sets | Lazy Synchronization

```
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);
```



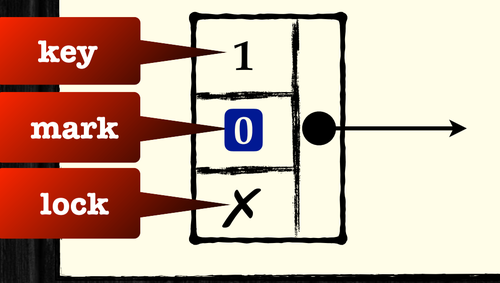


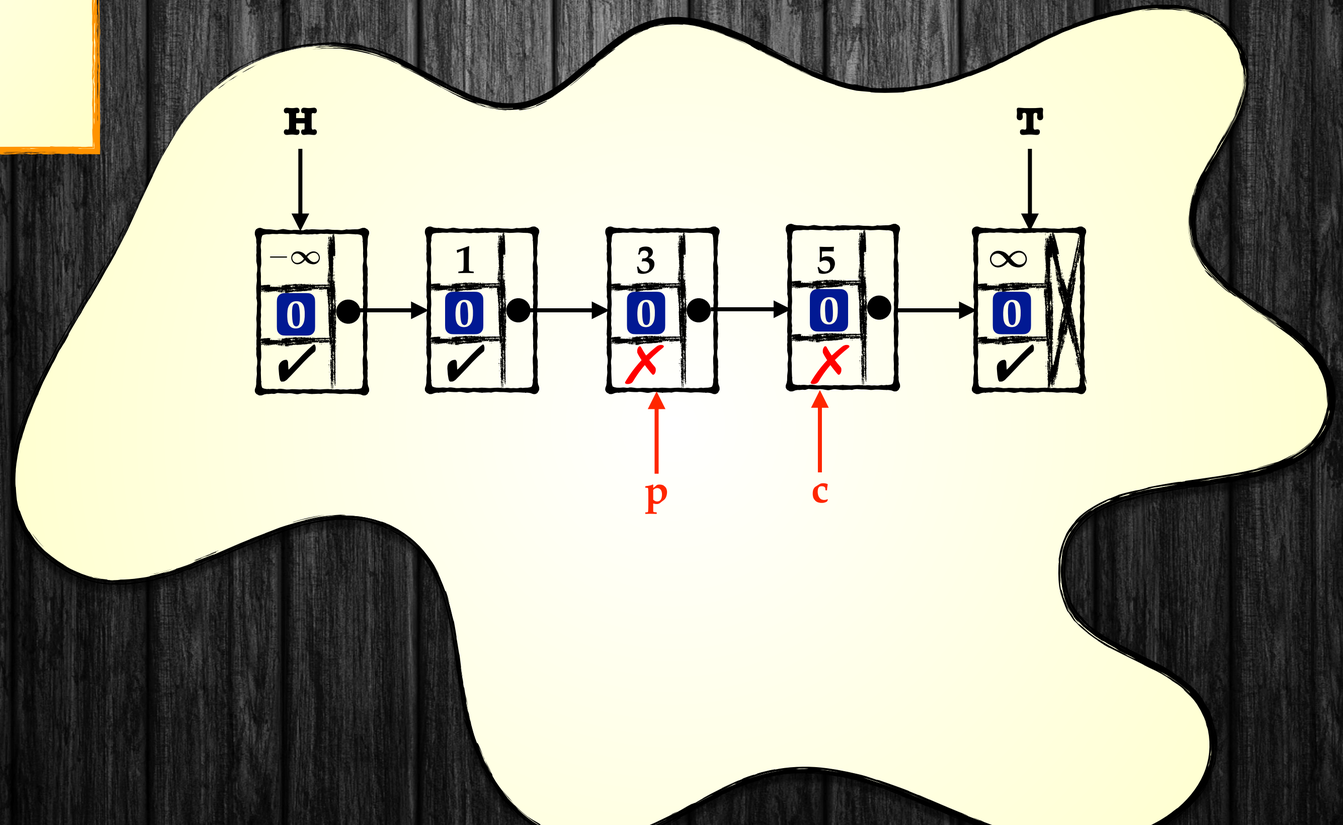


Sets | Lazy Synchronization

```
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)

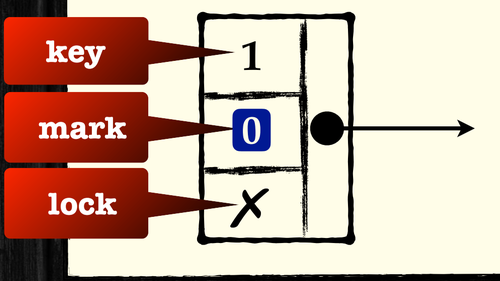


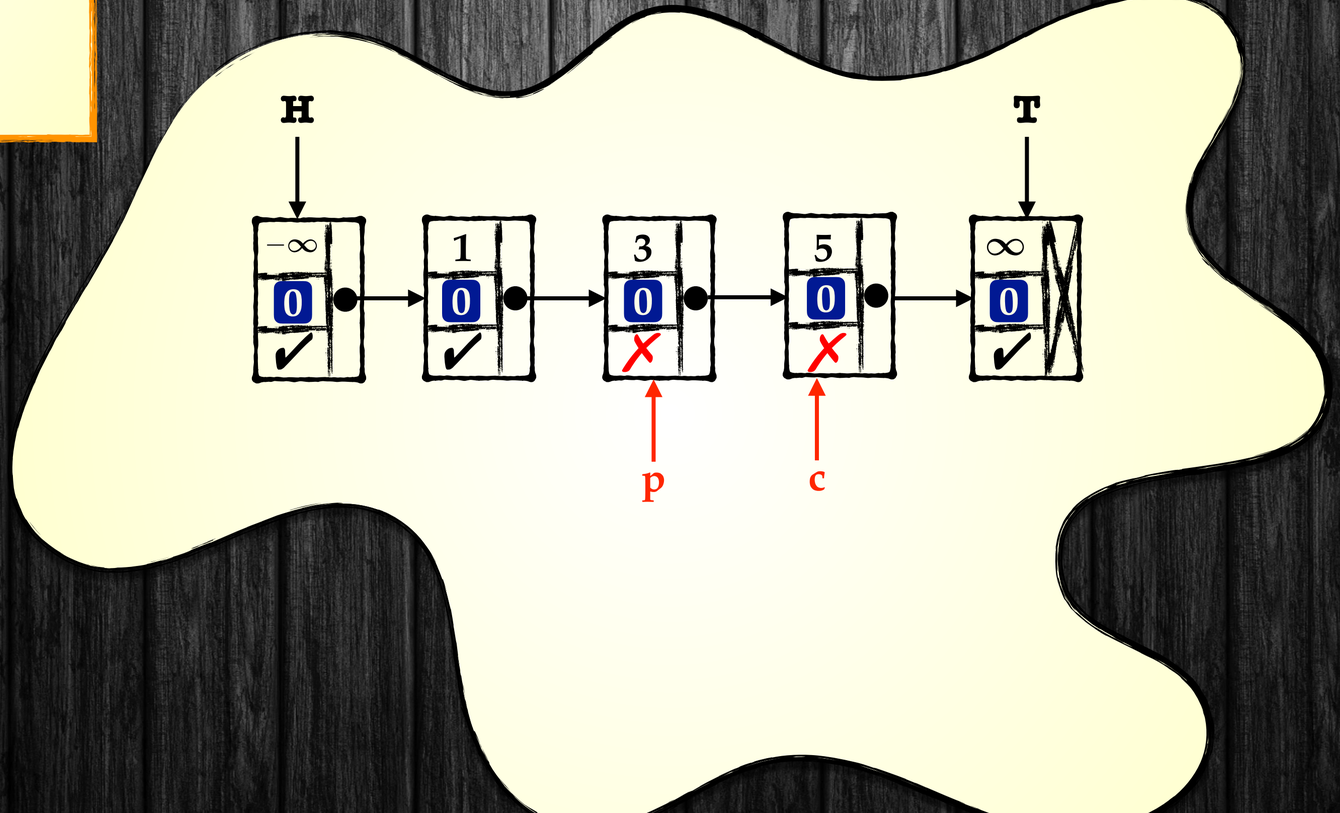


Sets | Lazy Synchronization

```
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);
```



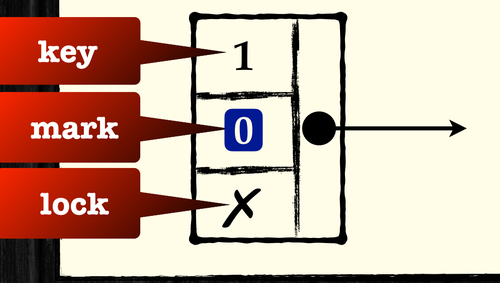


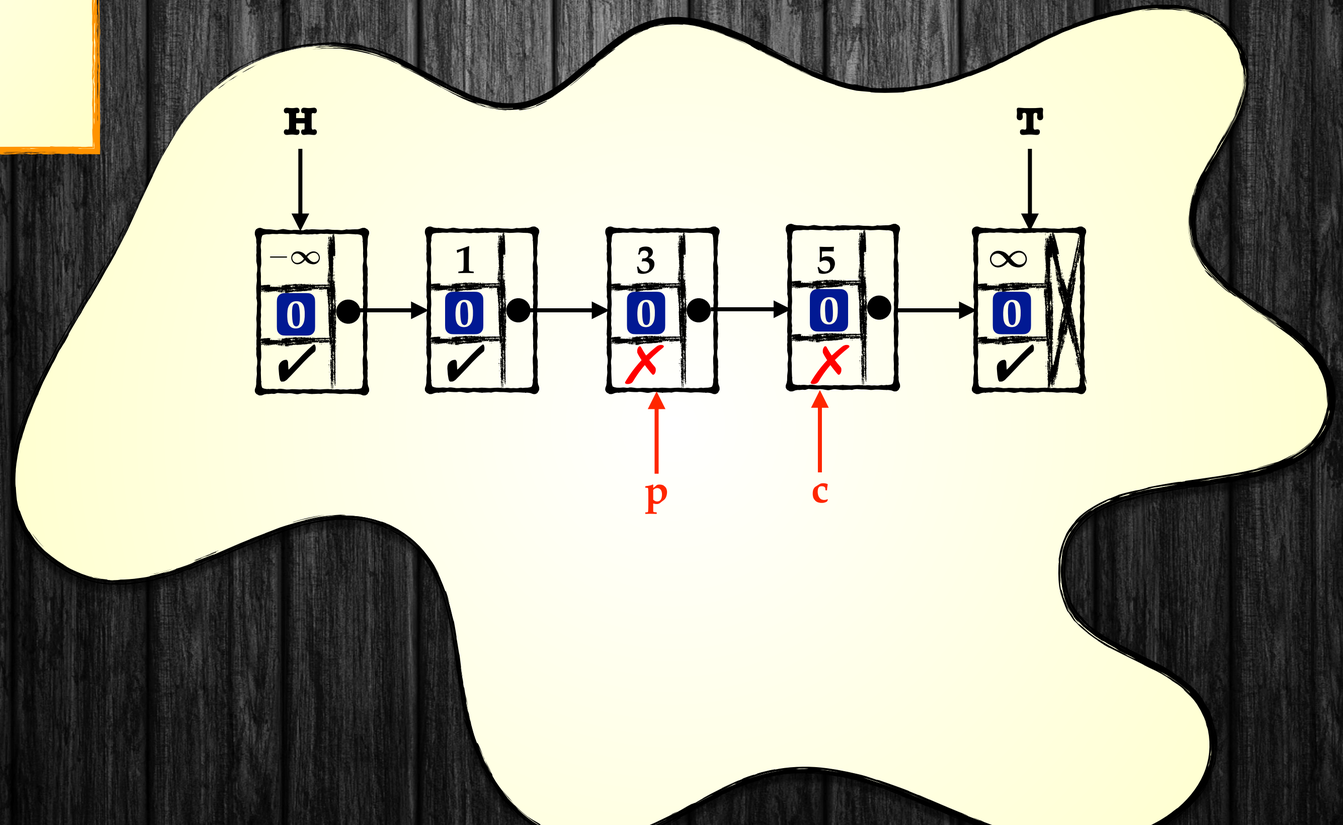


Sets | Lazy Synchronization

```
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)

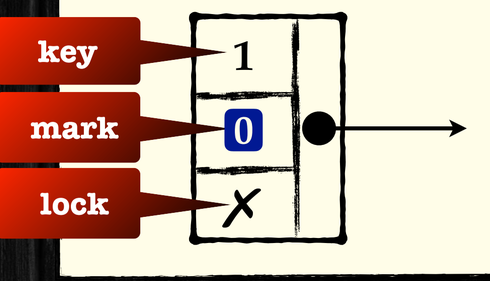


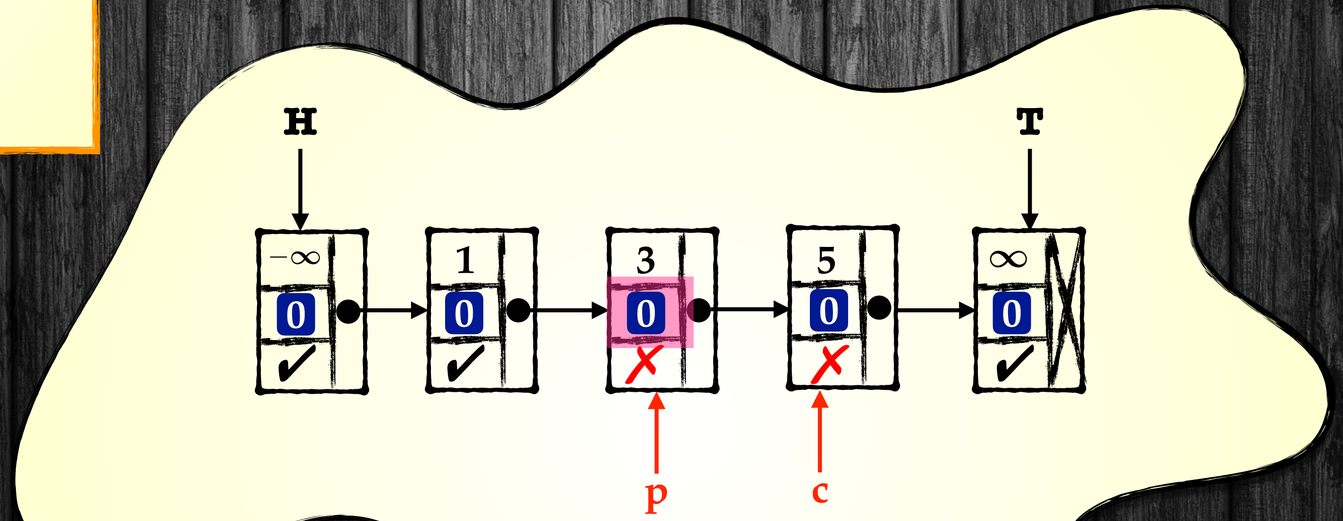


Sets | Lazy Synchronization

```
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)

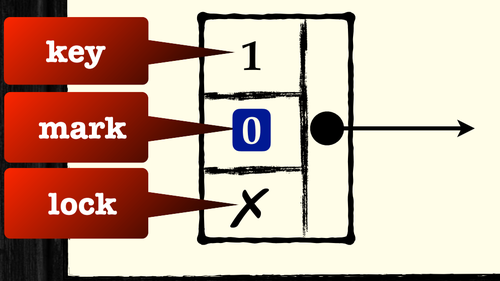


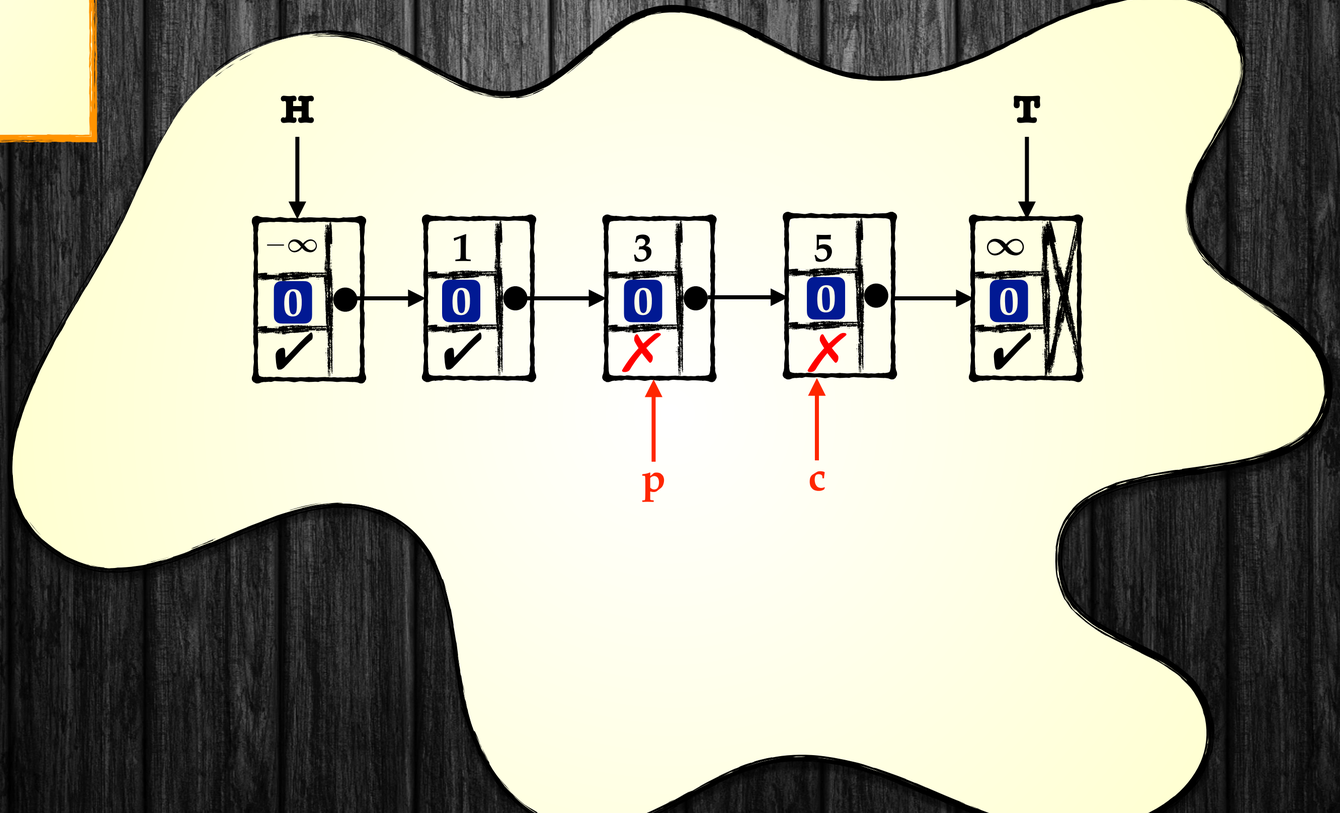


Sets | Lazy Synchronization

```
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);
```





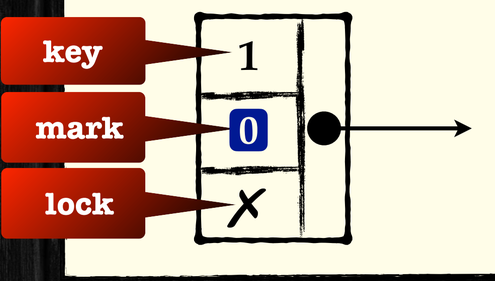


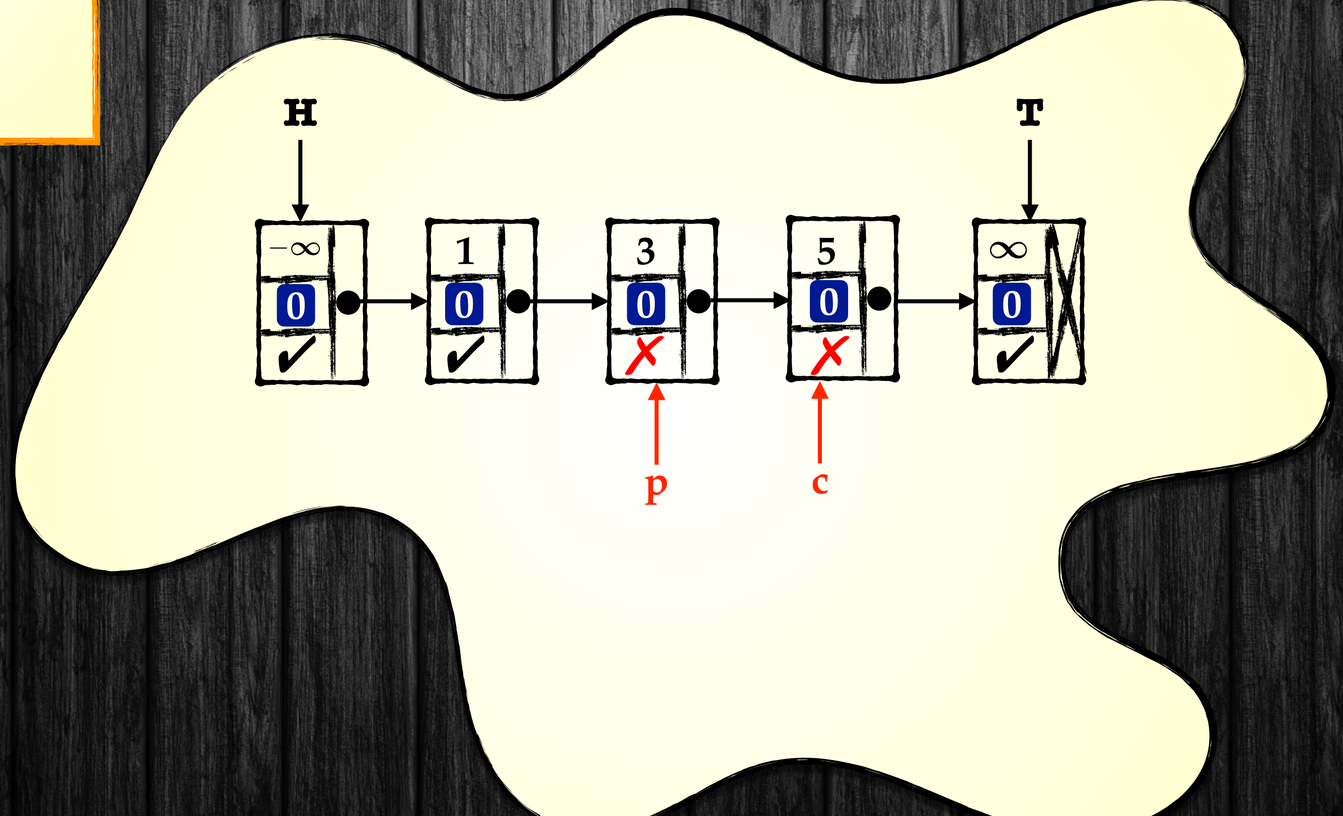
Sets | Lazy Synchronization

```
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)

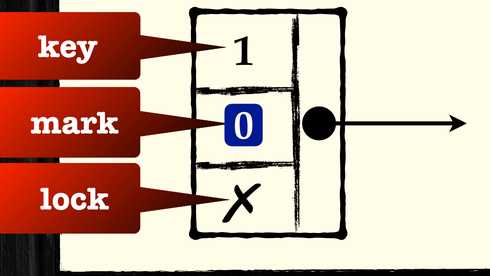


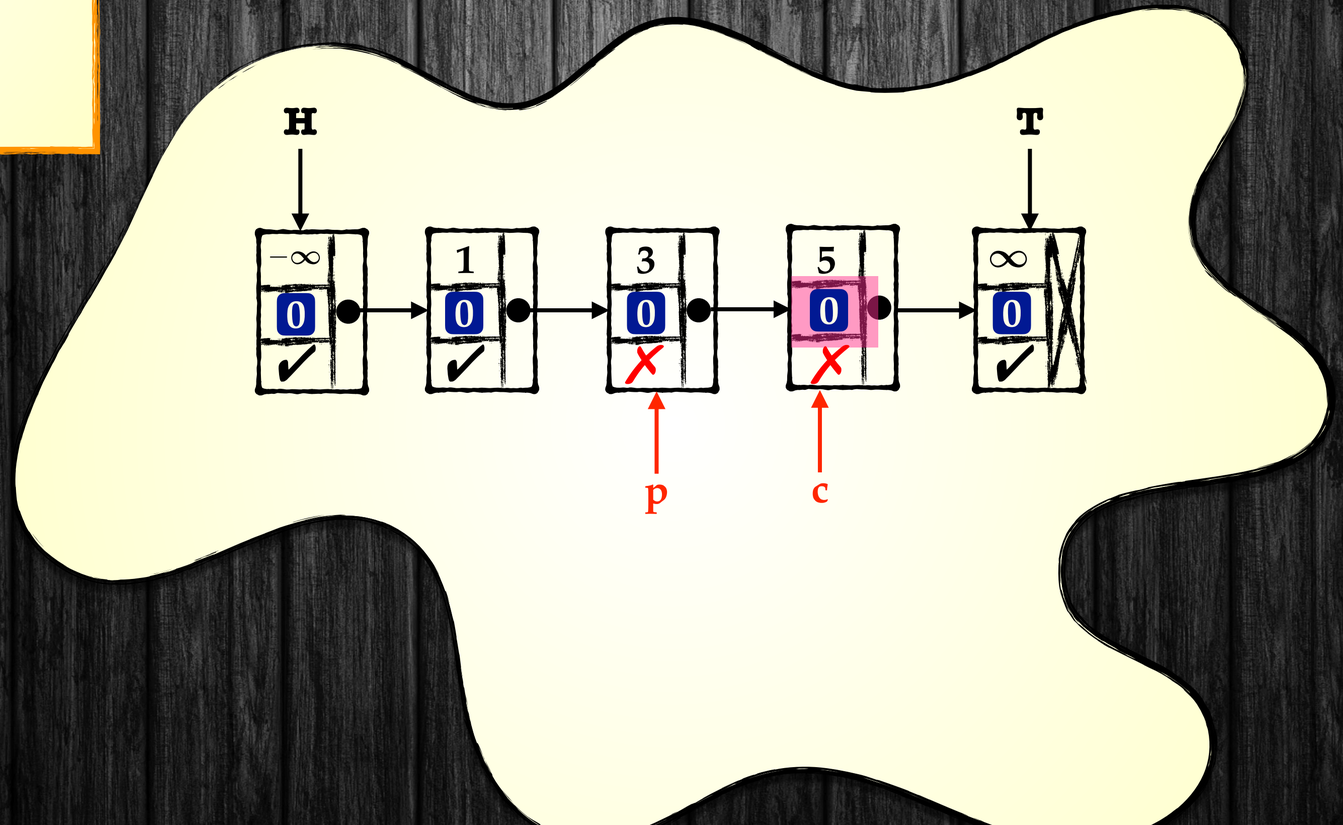


Sets | Lazy Synchronization

```
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)

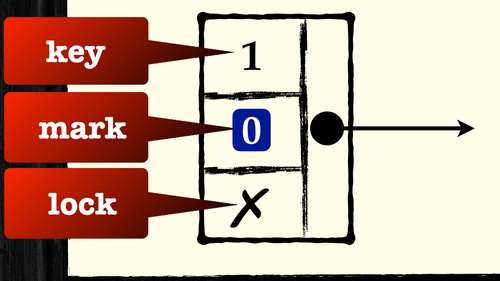


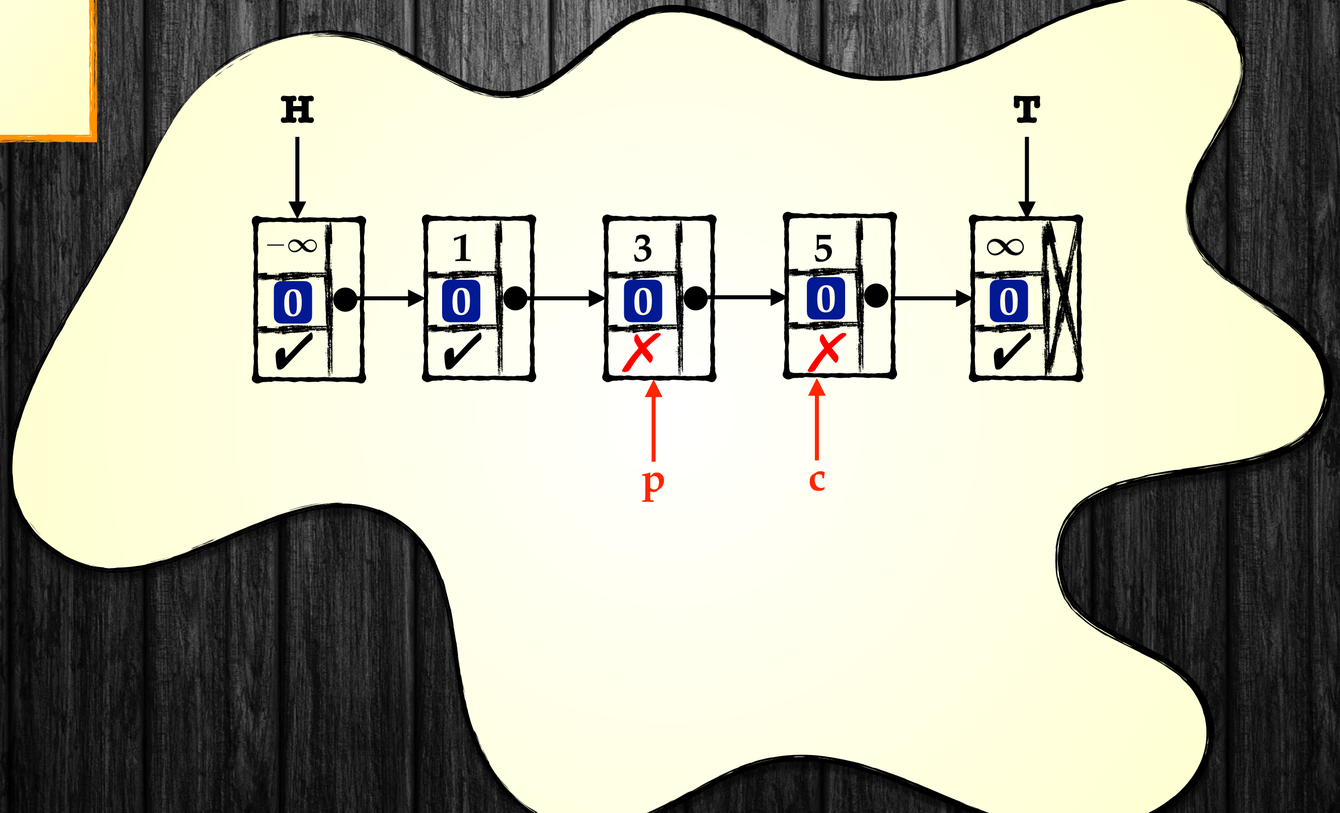


Sets | Lazy Synchronization

```
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);
```



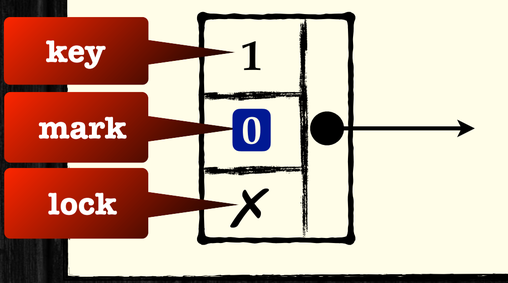


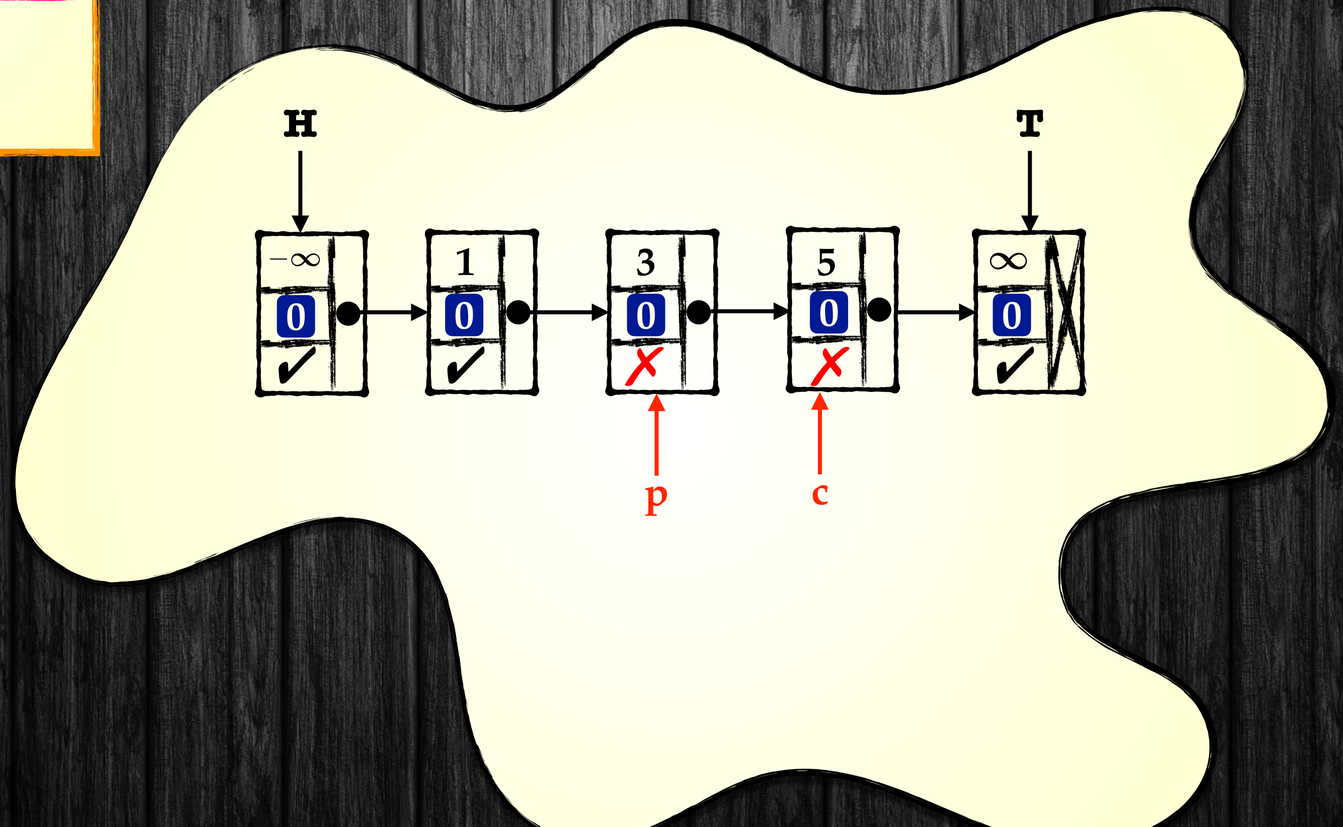


Sets | Lazy Synchronization

```
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)

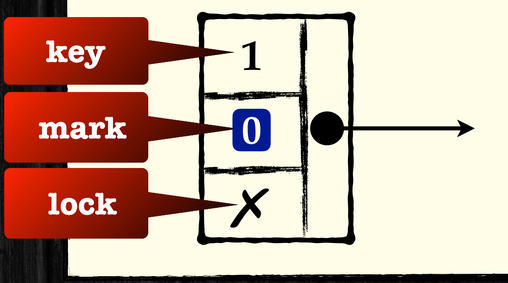


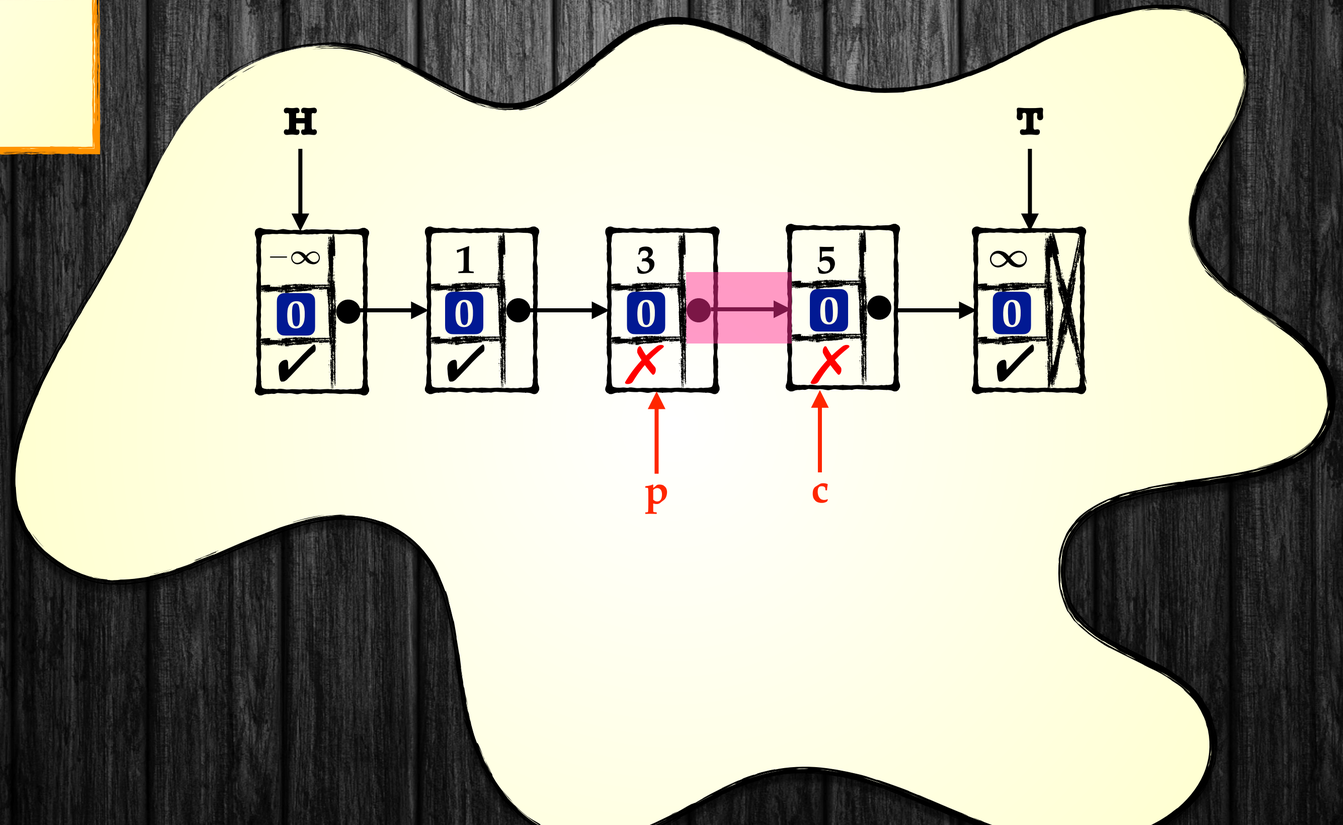


Sets | Lazy Synchronization

```
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)

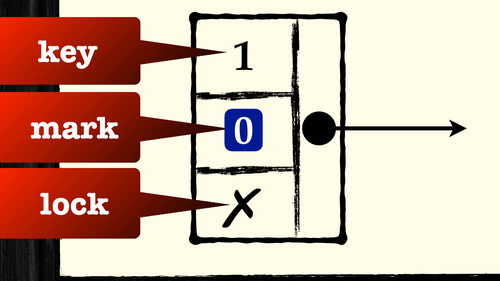


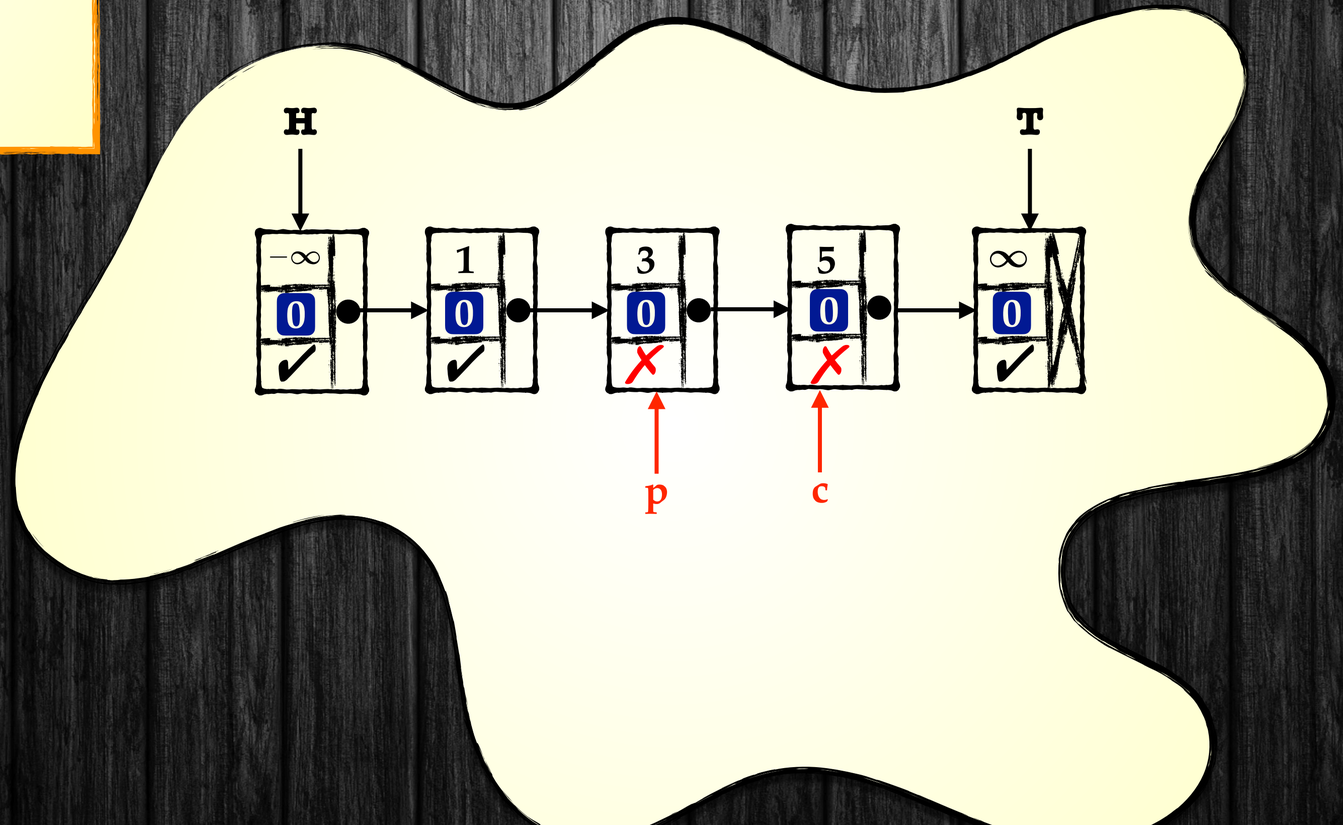


Sets | Lazy Synchronization

```
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)

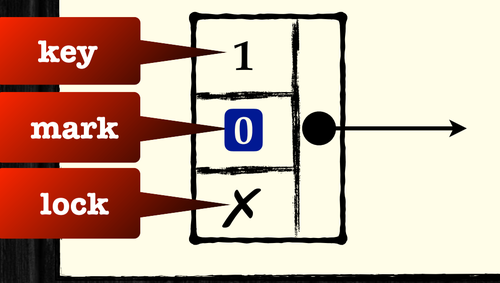


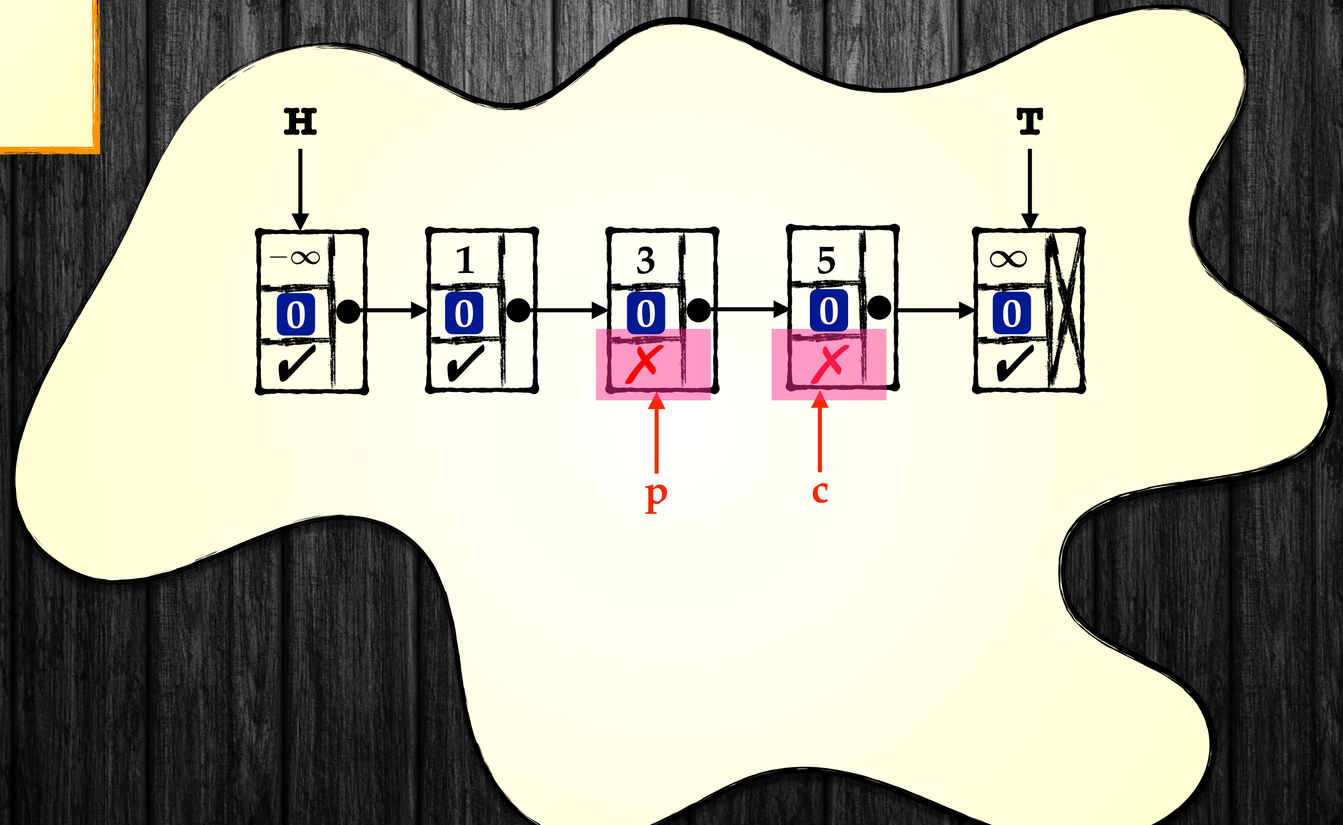


Sets | Lazy Synchronization

```
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)

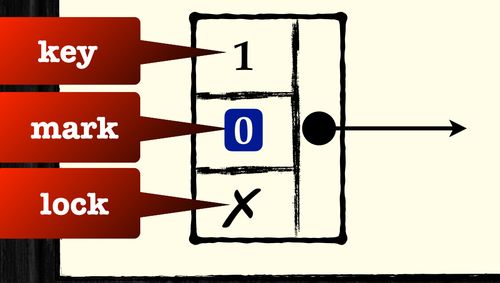


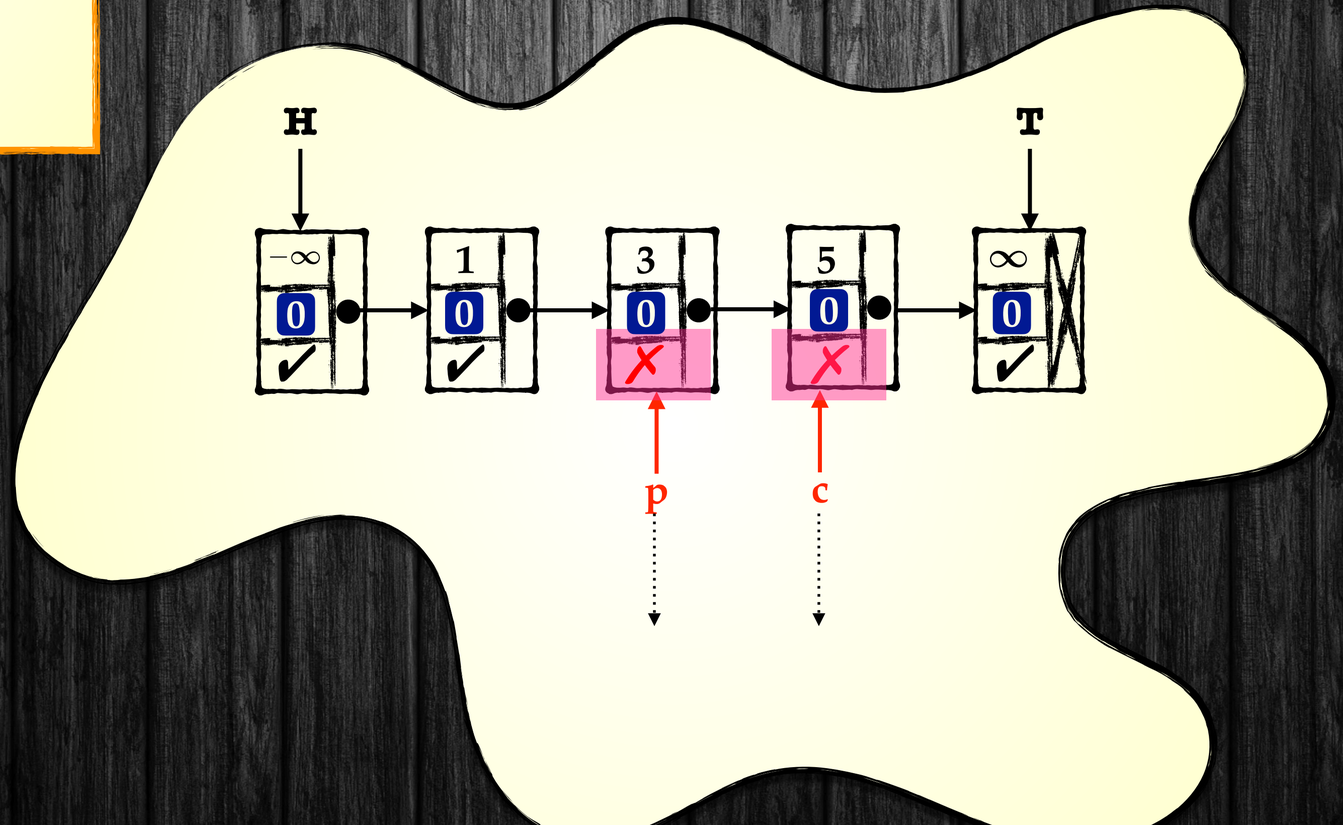


Sets | Lazy Synchronization

```
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)



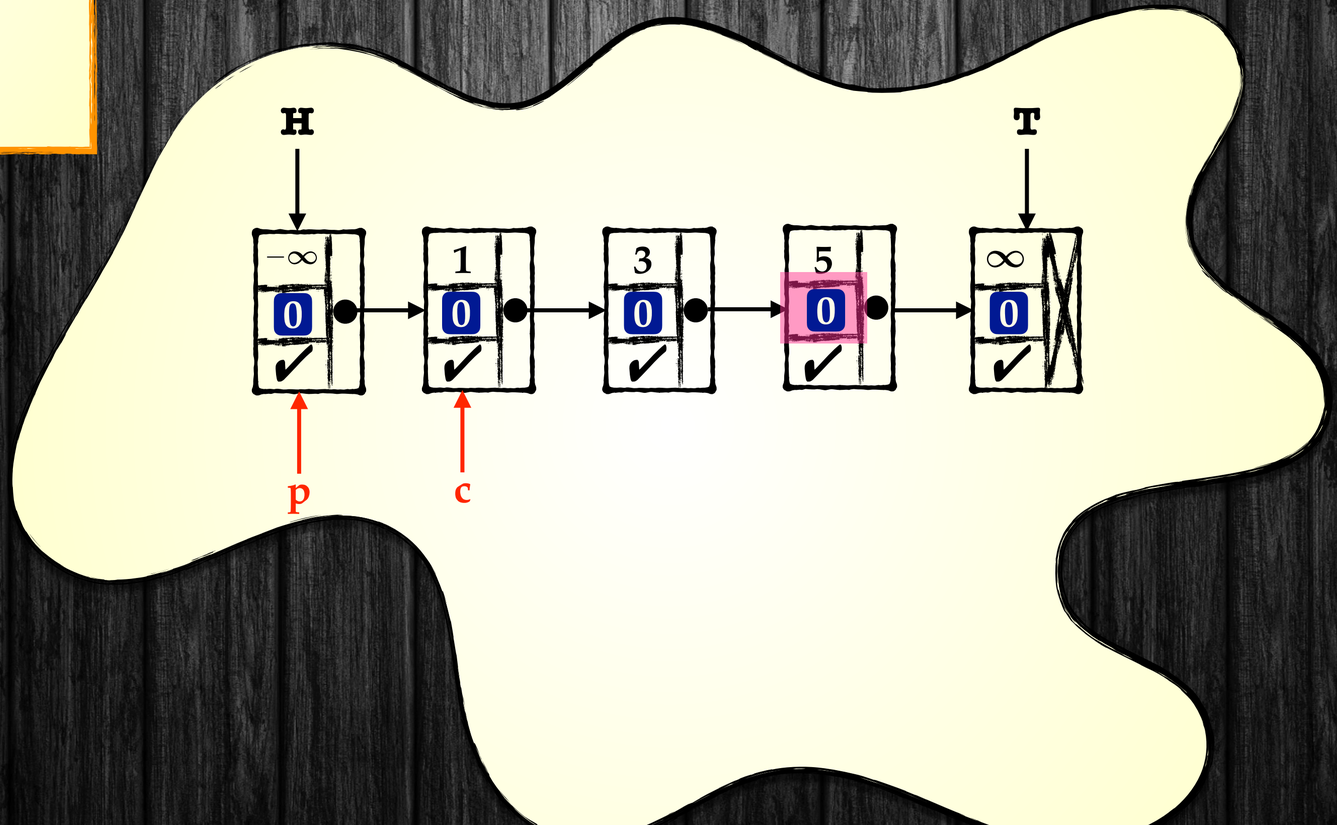


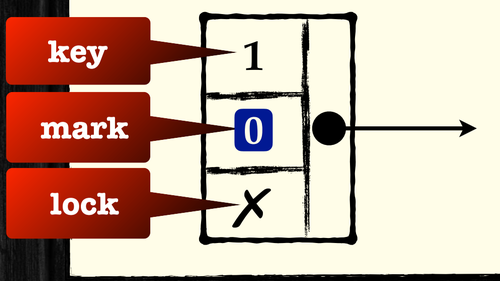
Sets | Lazy Synchronization

```
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



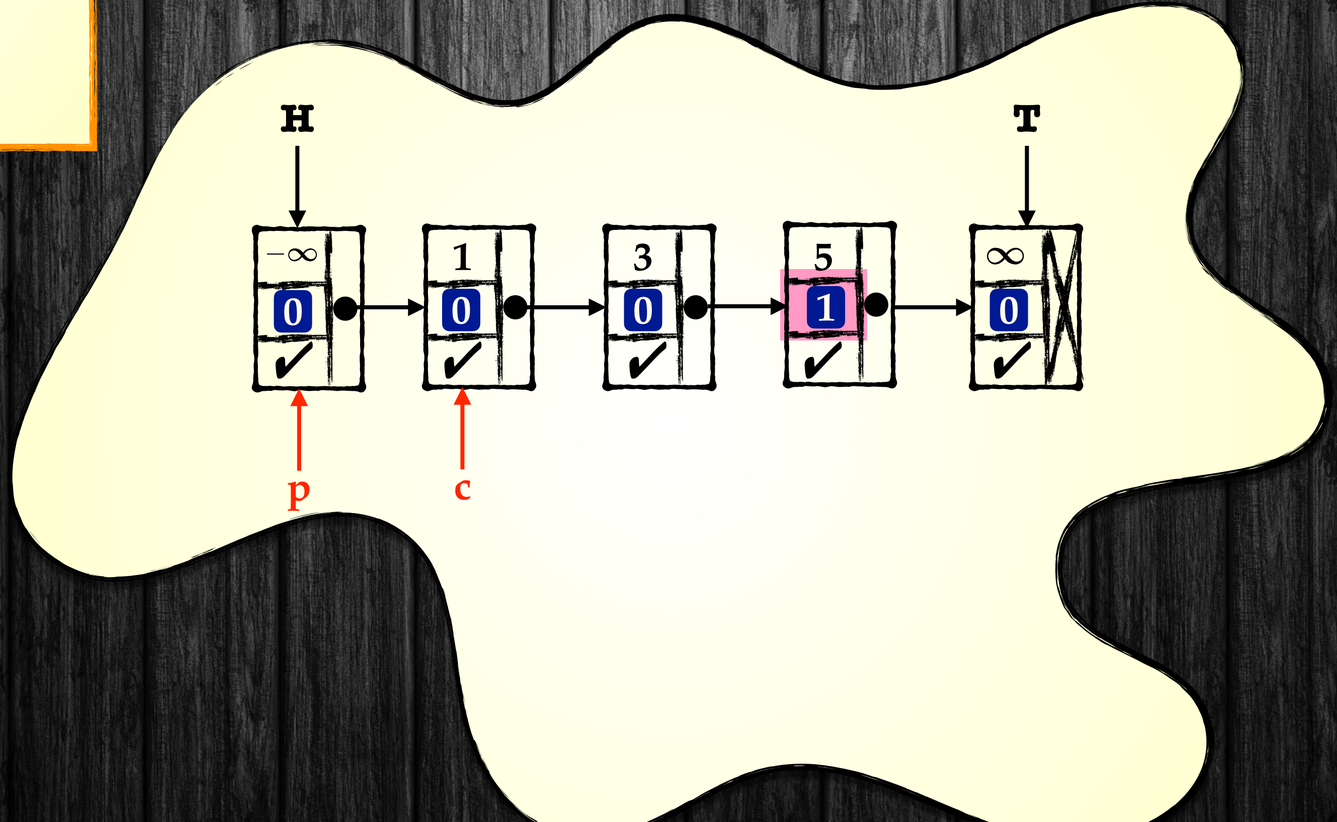


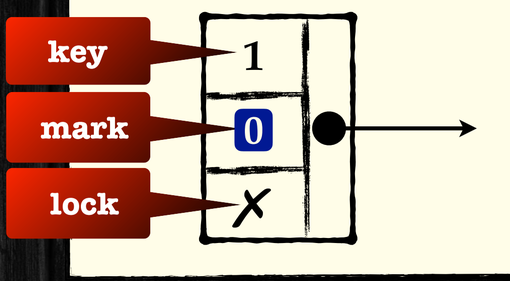
Sets | Lazy Synchronization

```
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



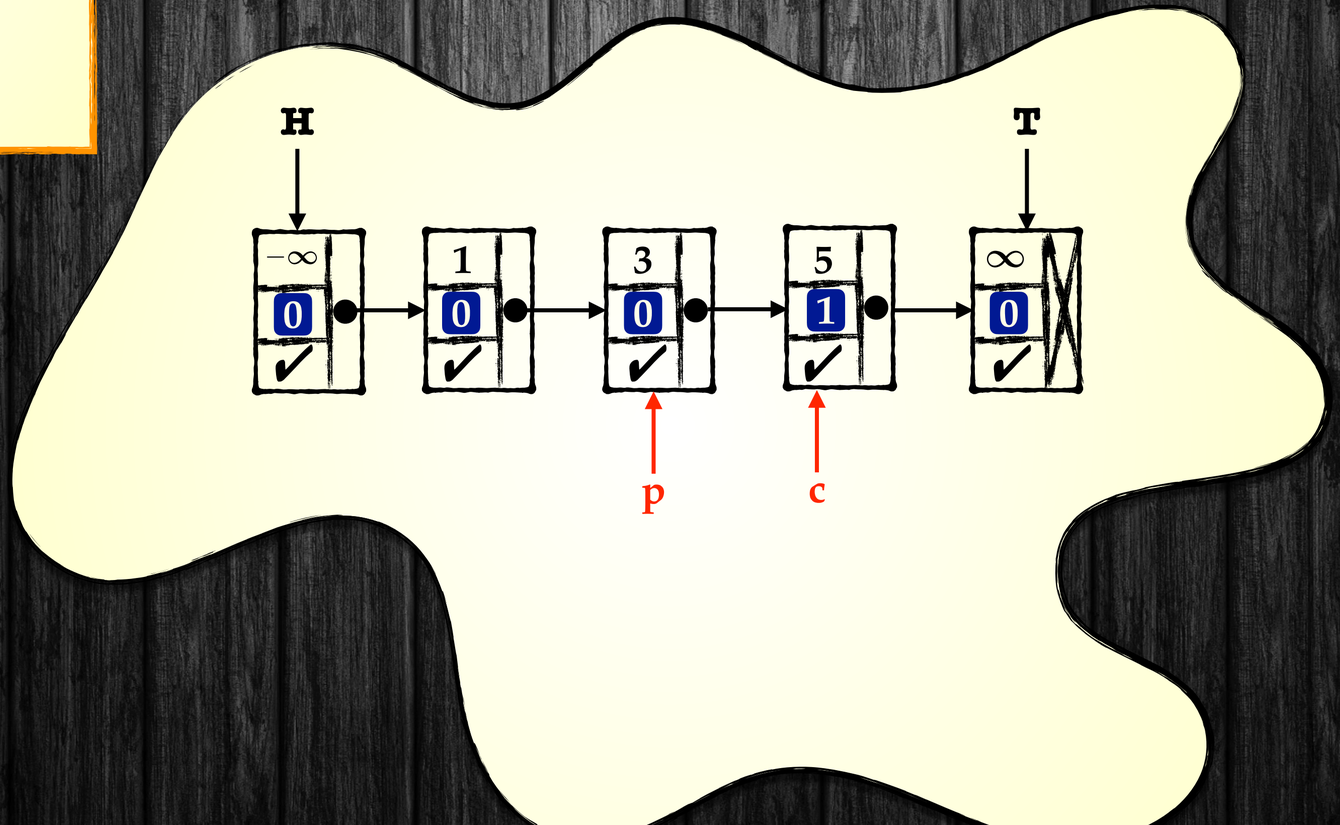


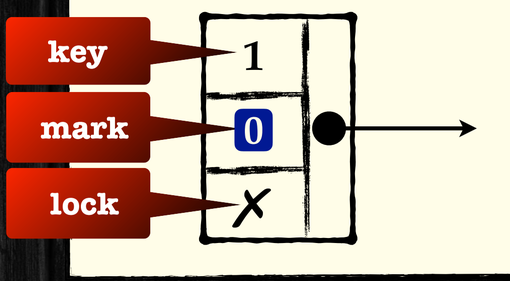
Sets | Lazy Synchronization

```
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);
```

locate(4)

different scenario



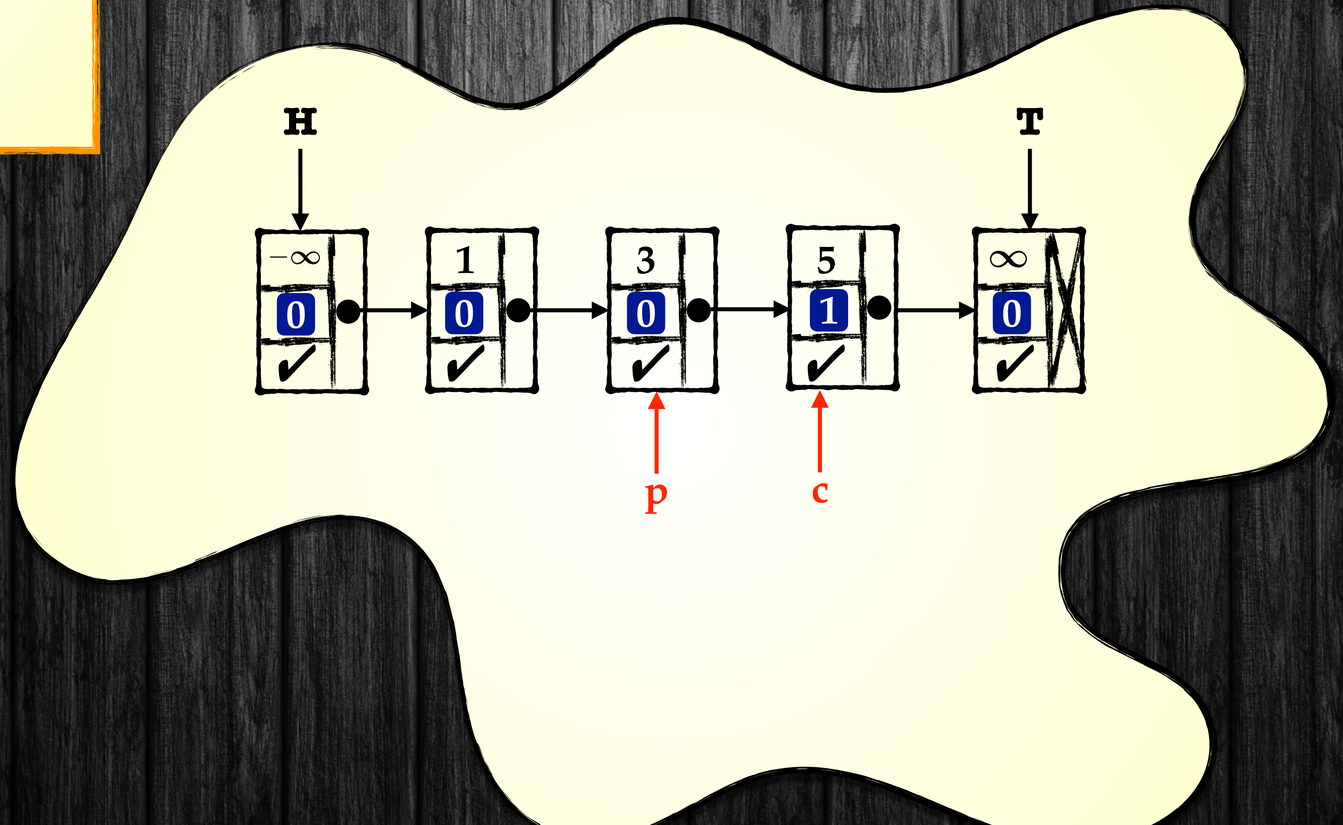


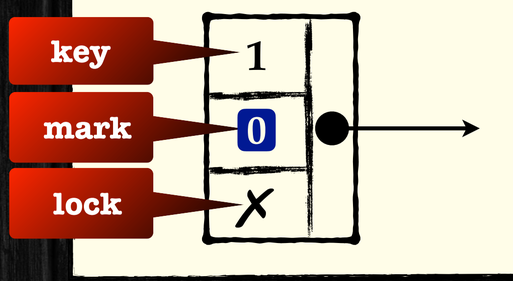
Sets | Lazy Synchronization

```
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



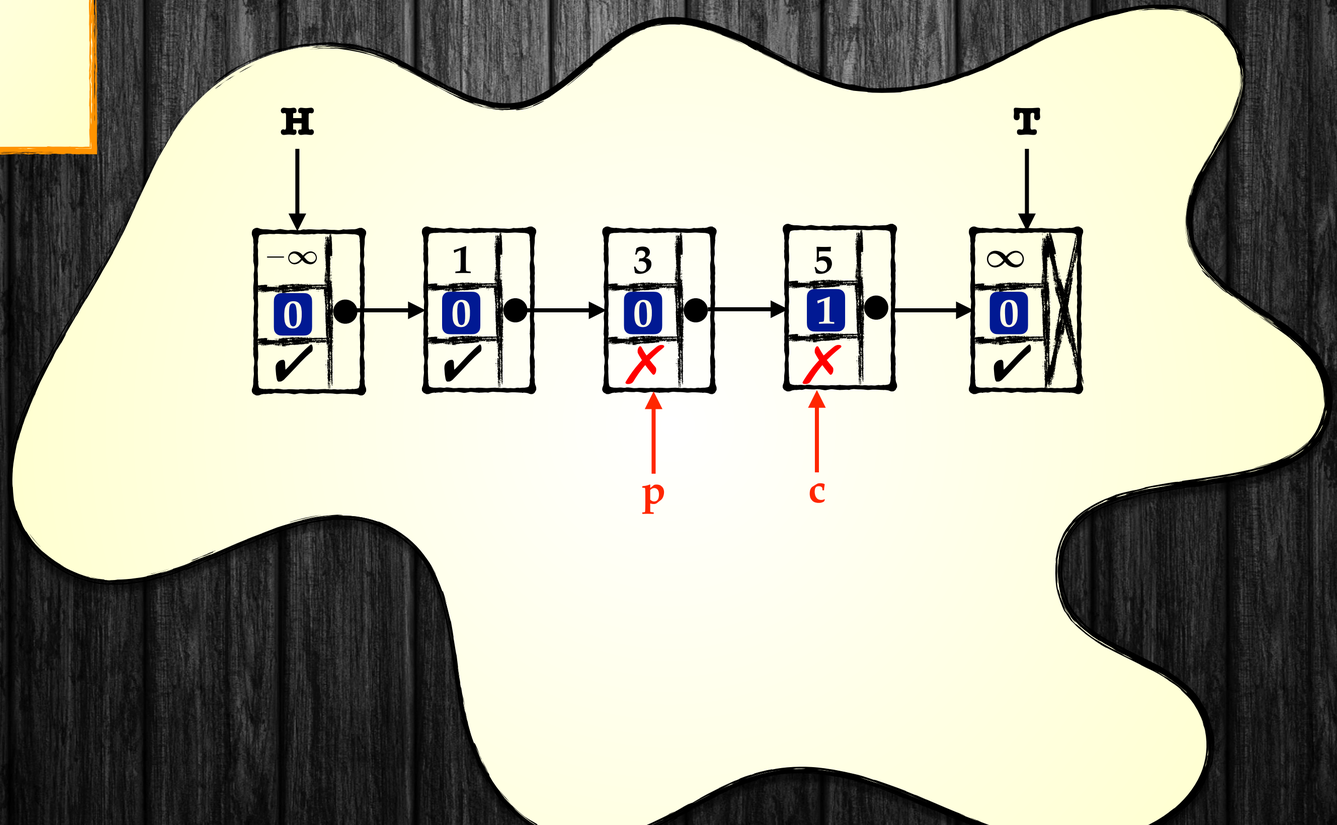


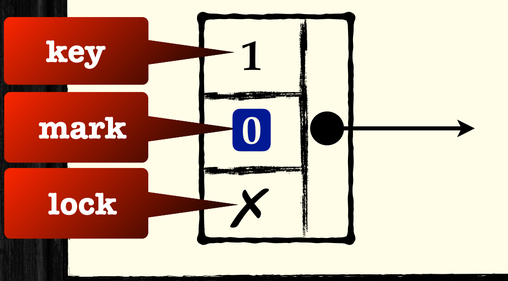
Sets | Lazy Synchronization

```
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);
```

locate(4)

different scenario



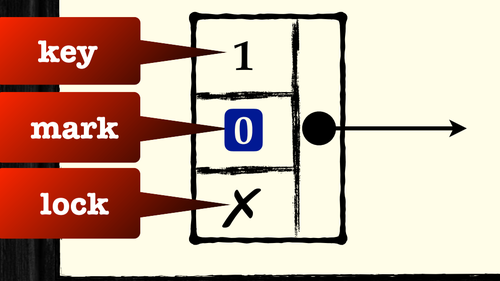


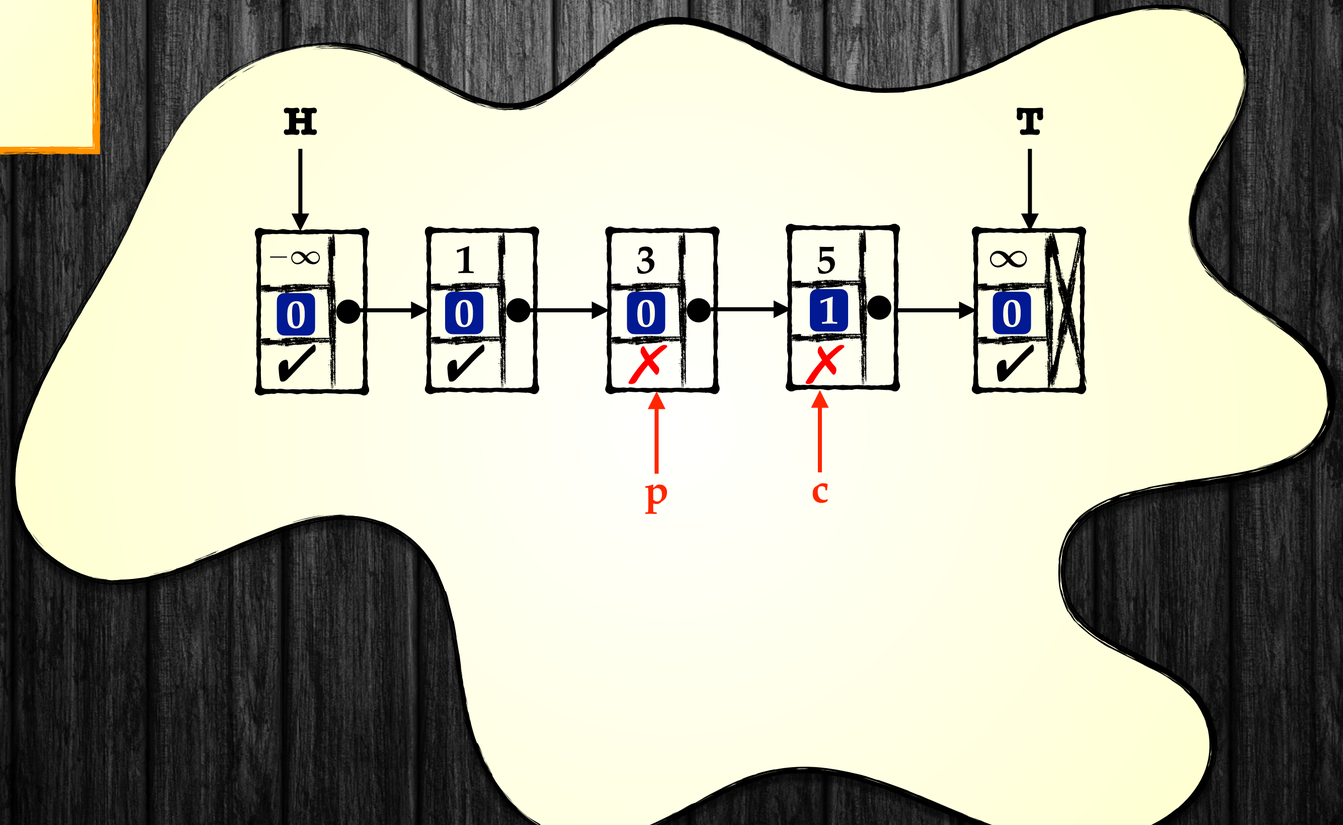
Sets | Lazy Synchronization

```
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



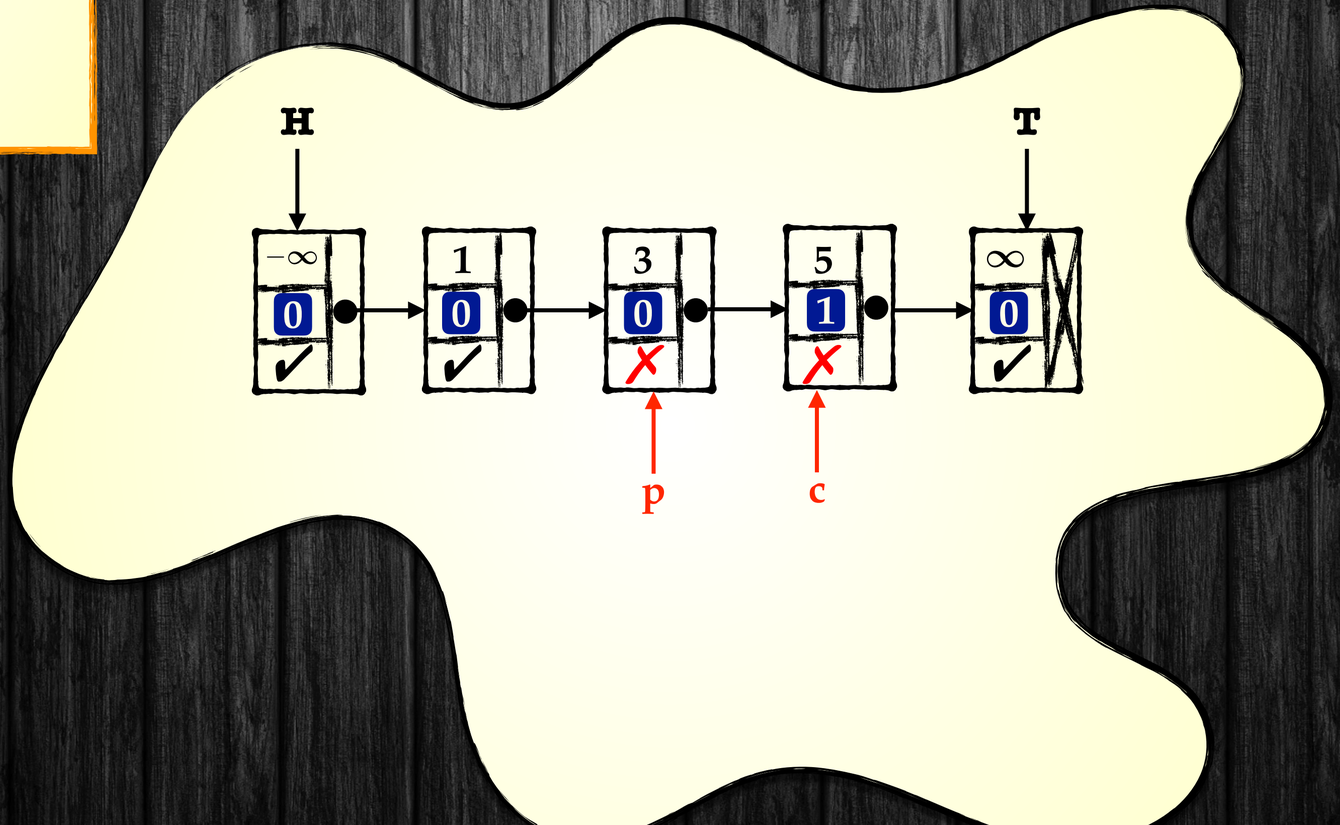


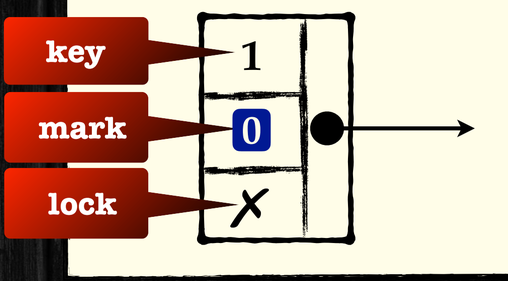
Sets | Lazy Synchronization

```
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);
```

locate(4)

different scenario



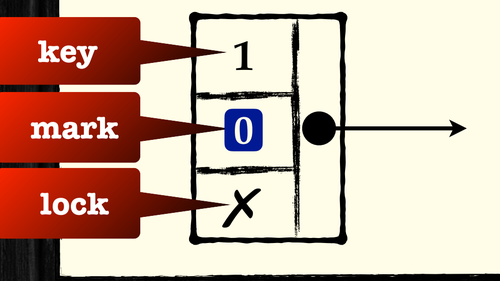


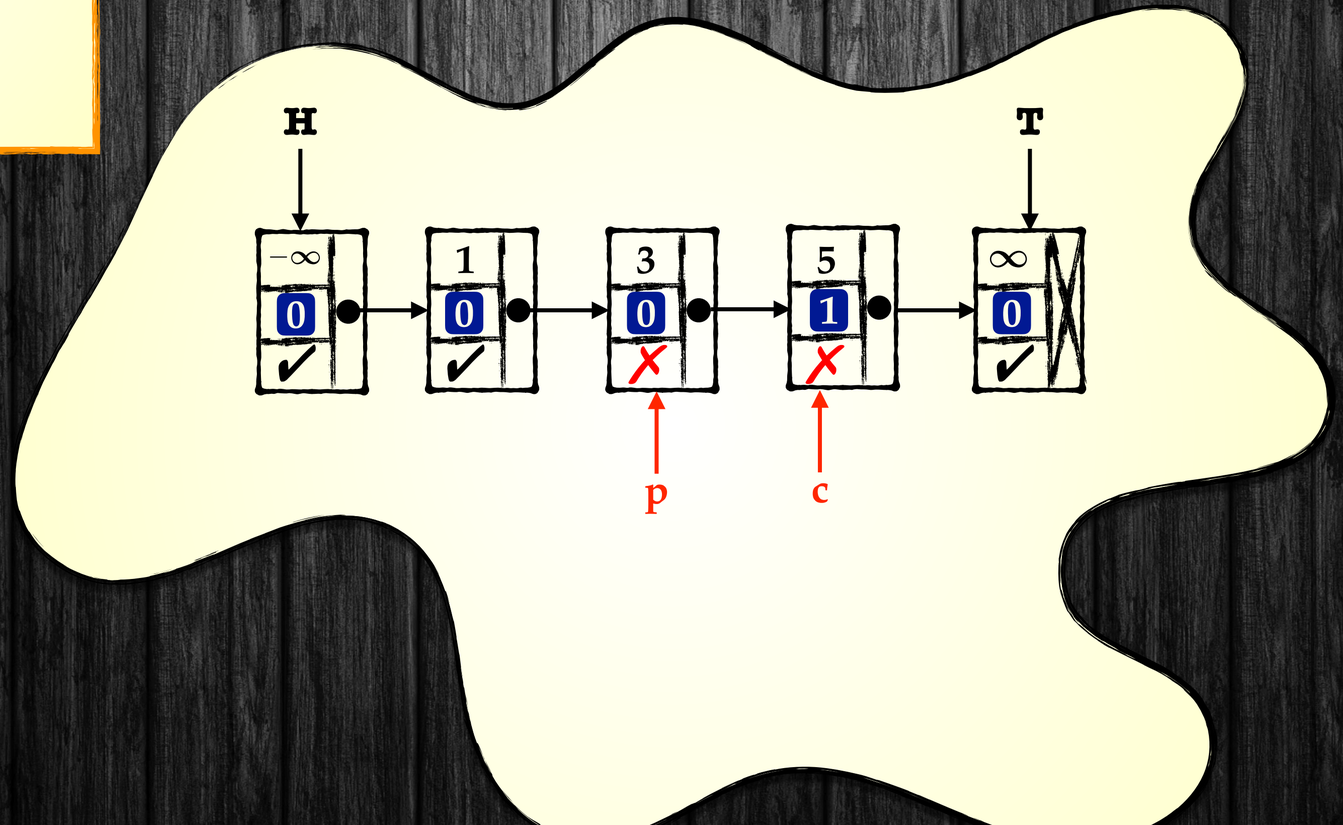
Sets | Lazy Synchronization

```
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



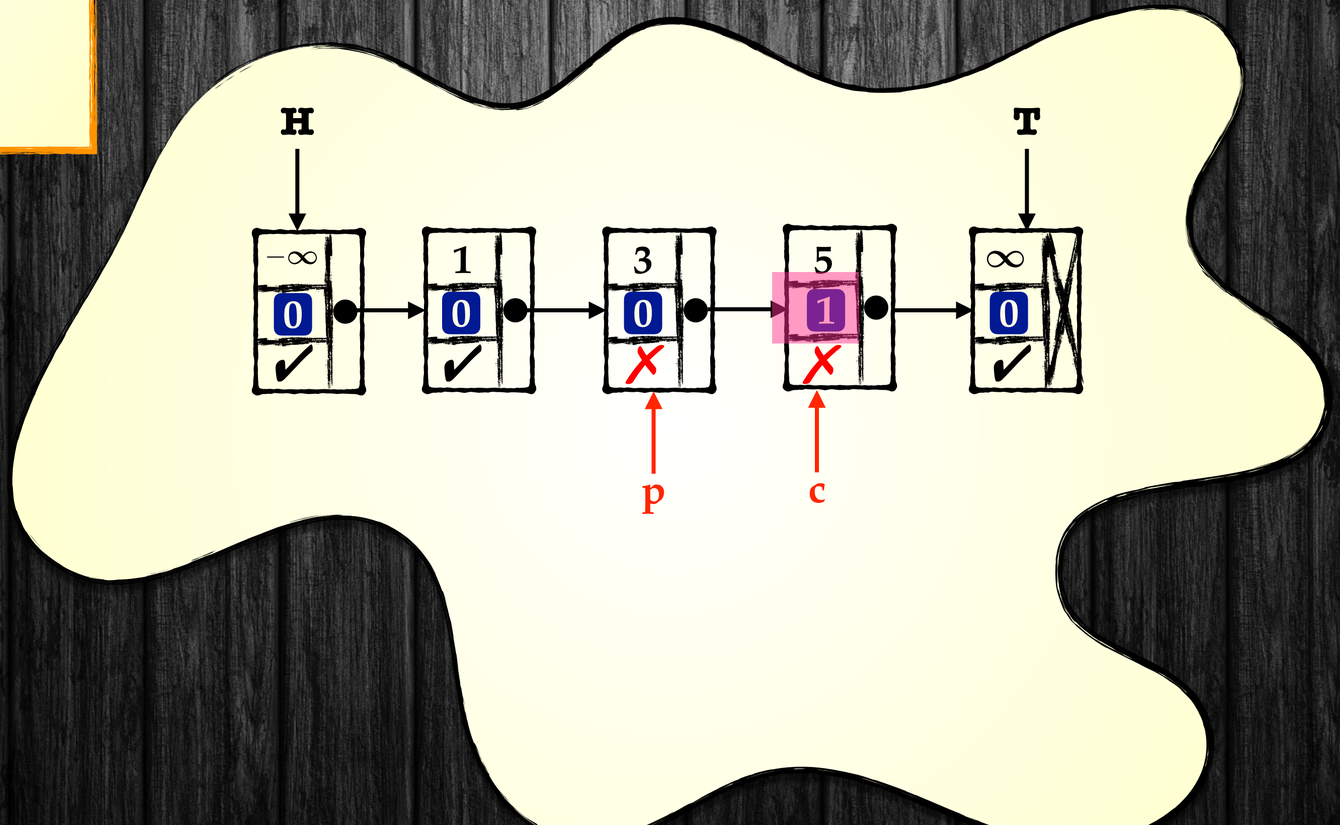


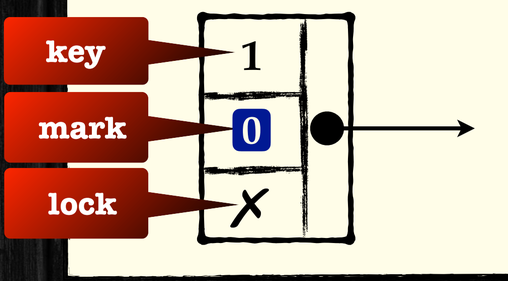
Sets | Lazy Synchronization

```
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);
```

locate(4)

different scenario



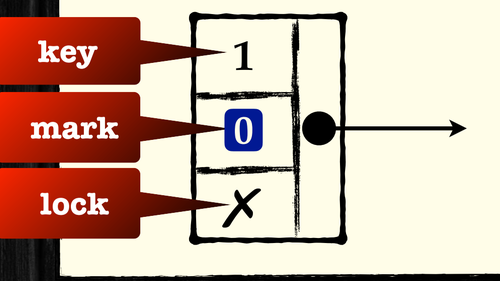


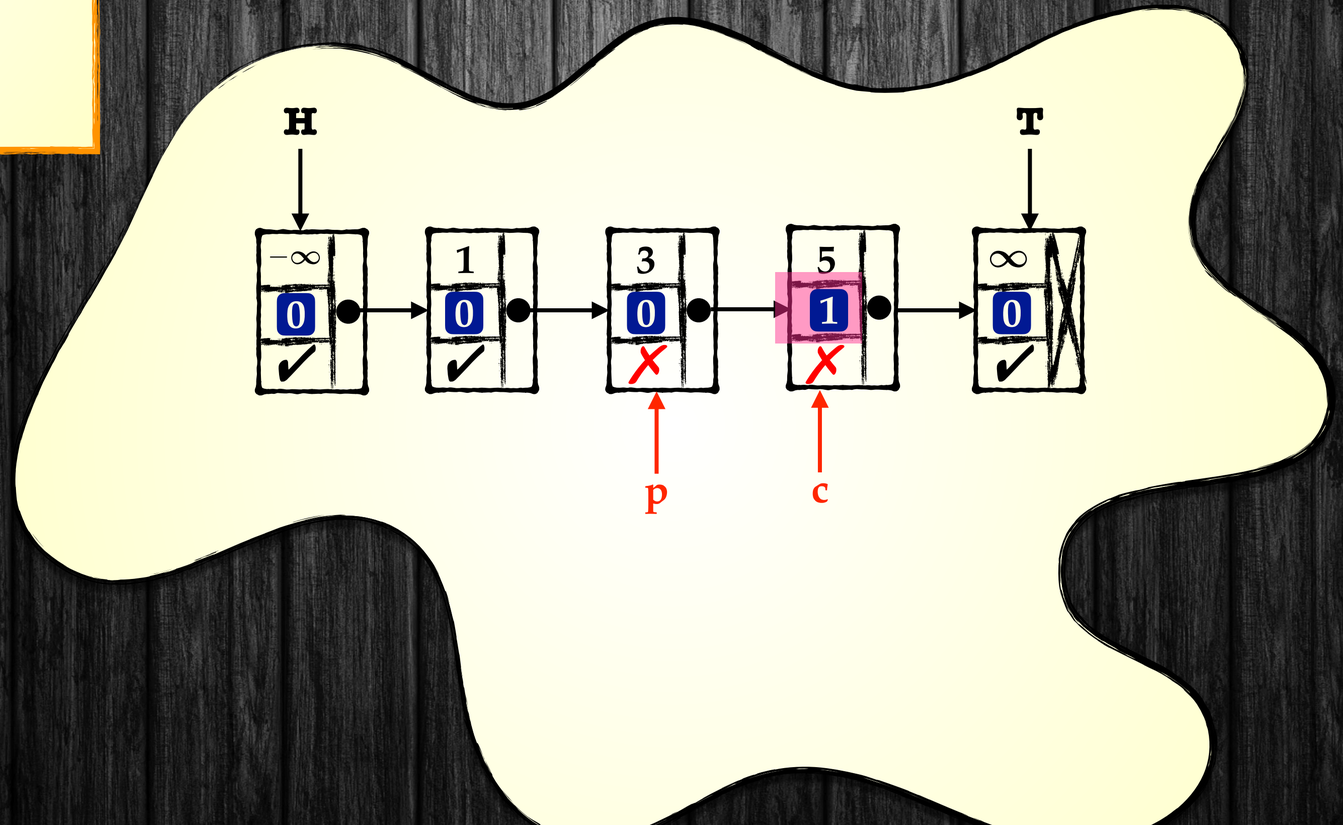
Sets | Lazy Synchronization

```
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



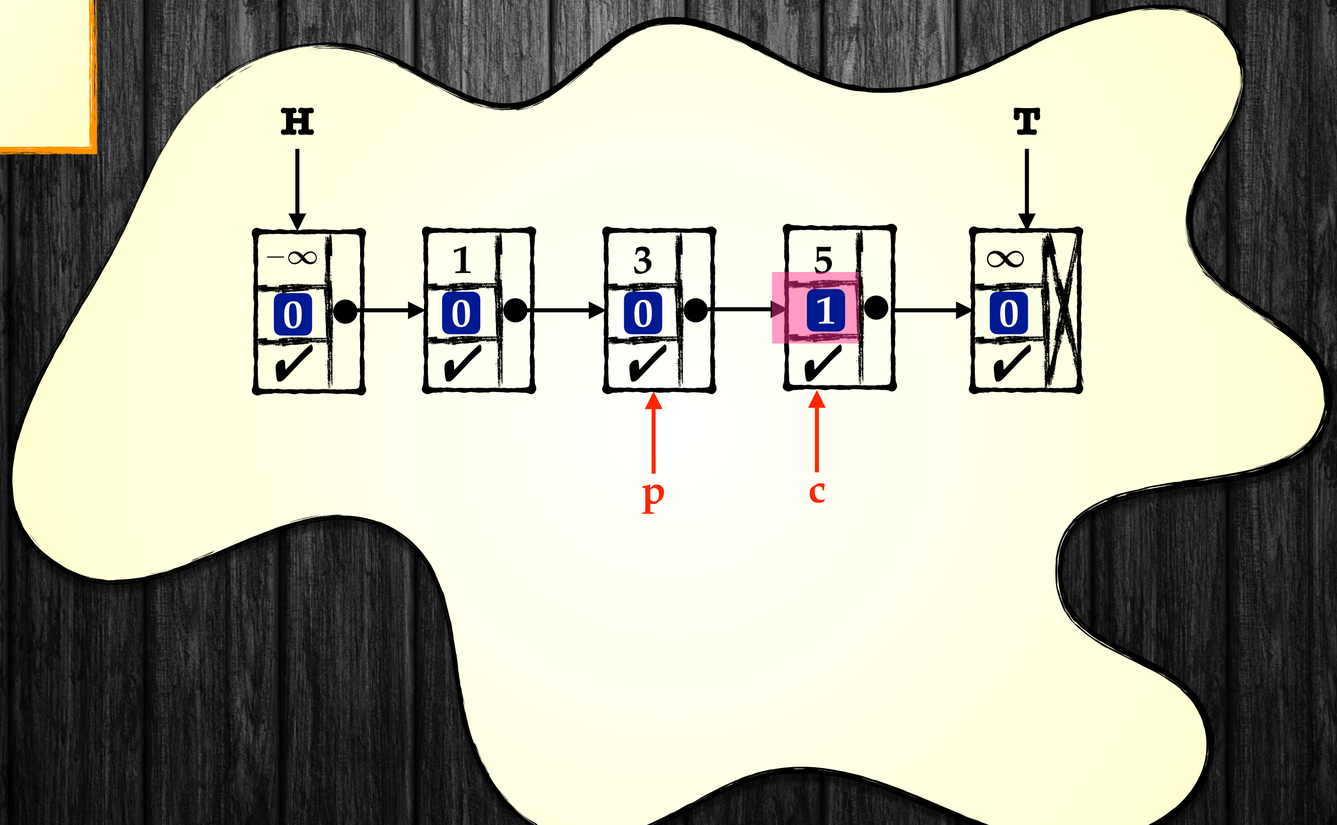


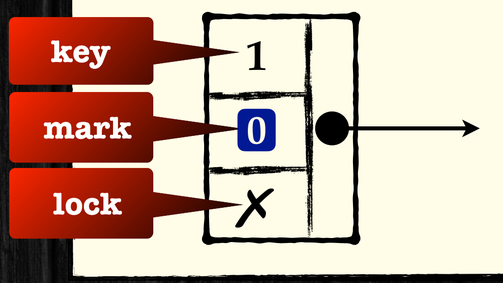
Sets | Lazy Synchronization

```
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);
```

locate(4)

different scenario



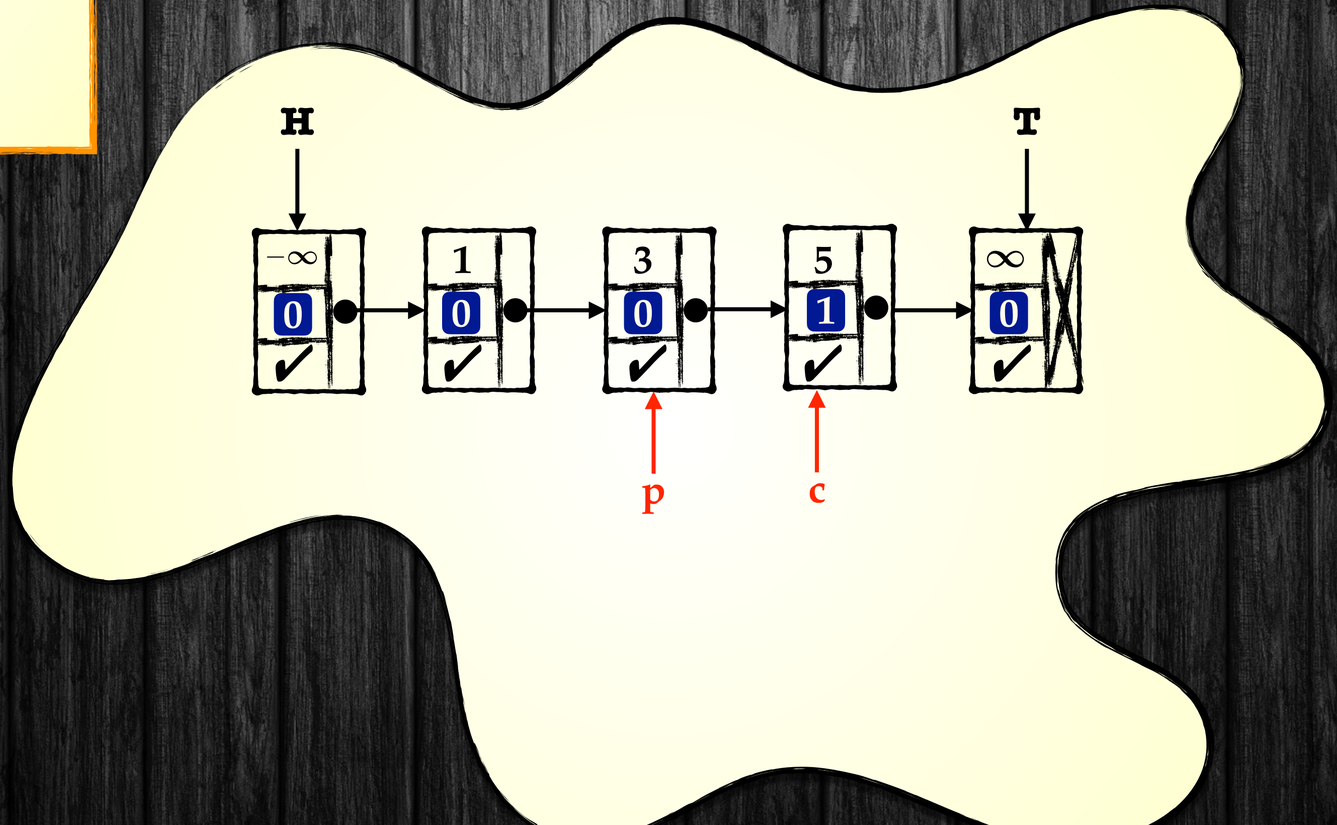


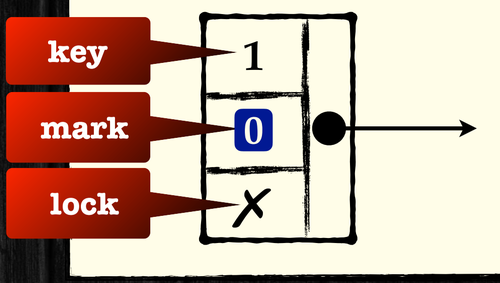
Sets | Lazy Synchronization

```
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



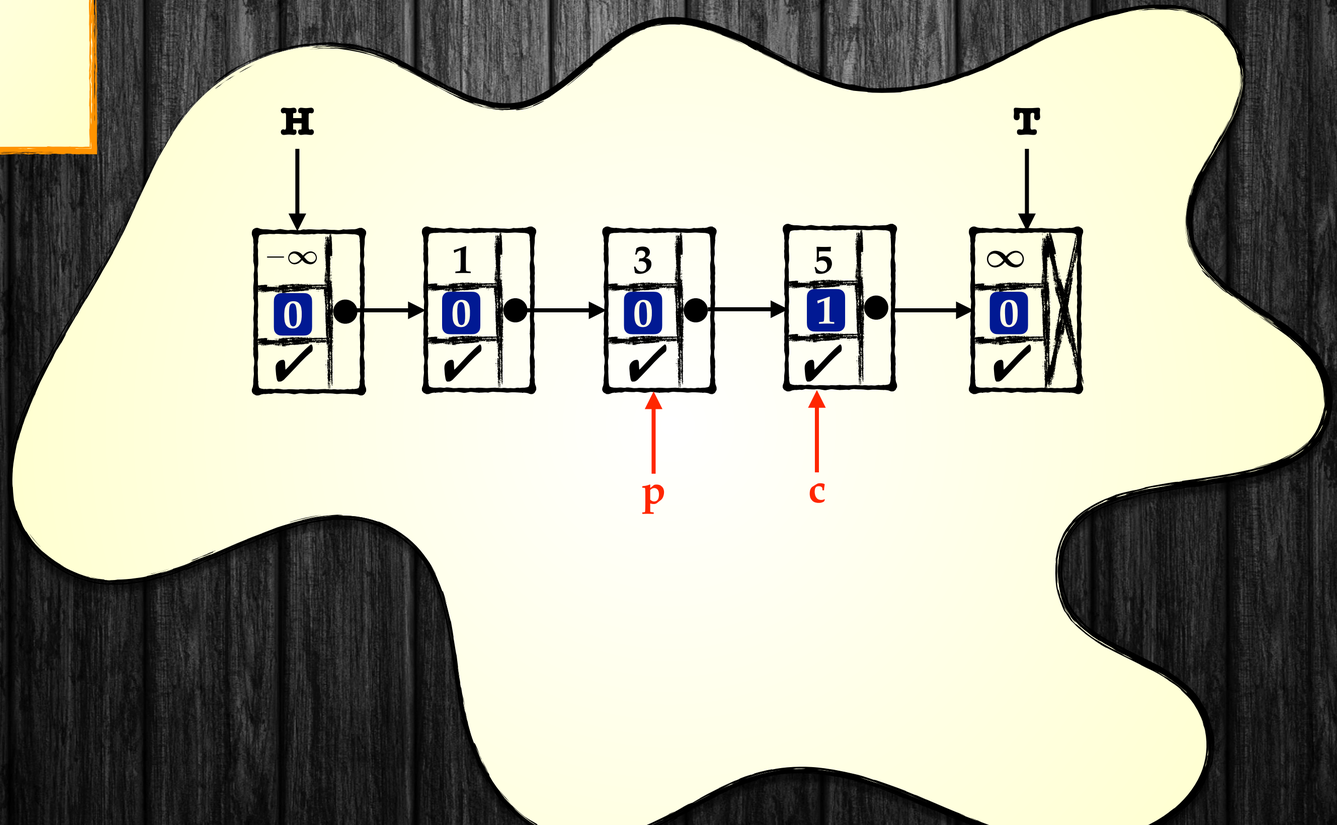


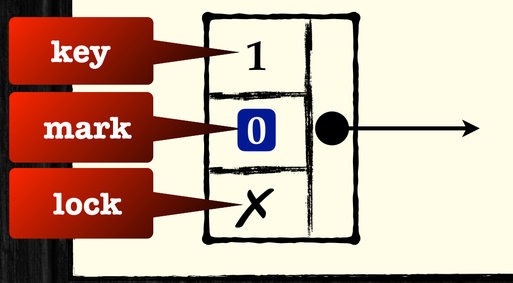
Sets | Lazy Synchronization

```
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);
```

locate(4)

different scenario





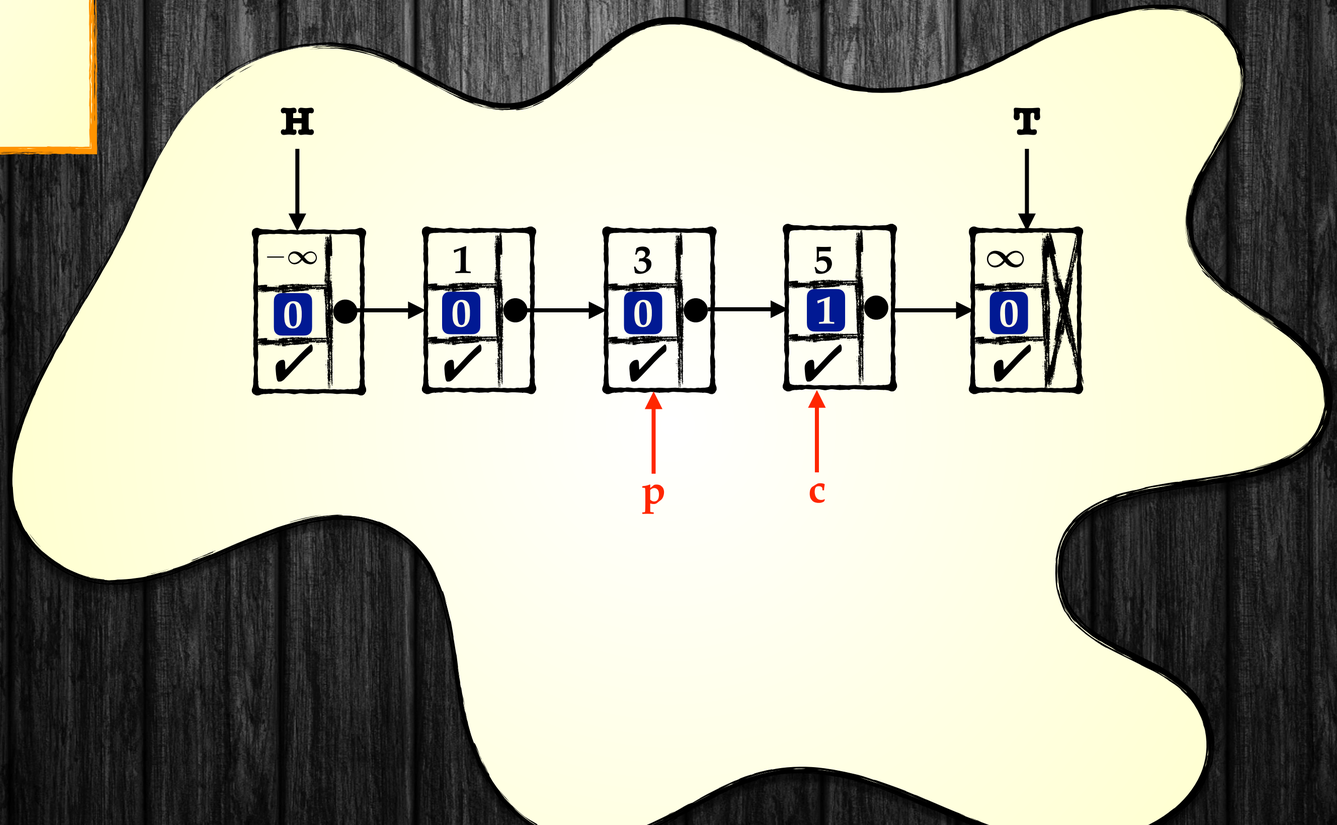
Sets | Lazy Synchronization

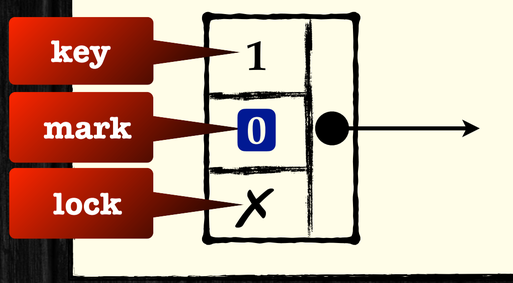
```
locate(e):
local p, c
1  while (true)
2     p := head;
3     c := p.next;
4     while (c.val < e)
5        p := c;
6        c := c.next
7     lock(p); lock(c);
8     if (!p.mark && !c.mark && p.next=c)
9        return(p,c);
10    else
11       unlock(p);
12       unlock(c);
```

try again

locate(4)

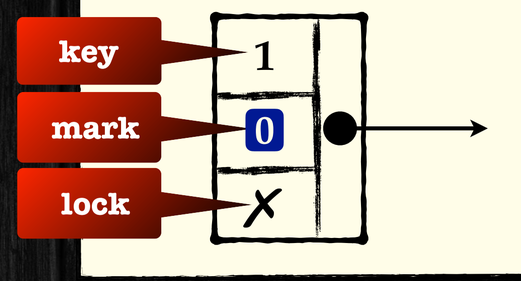
different scenario

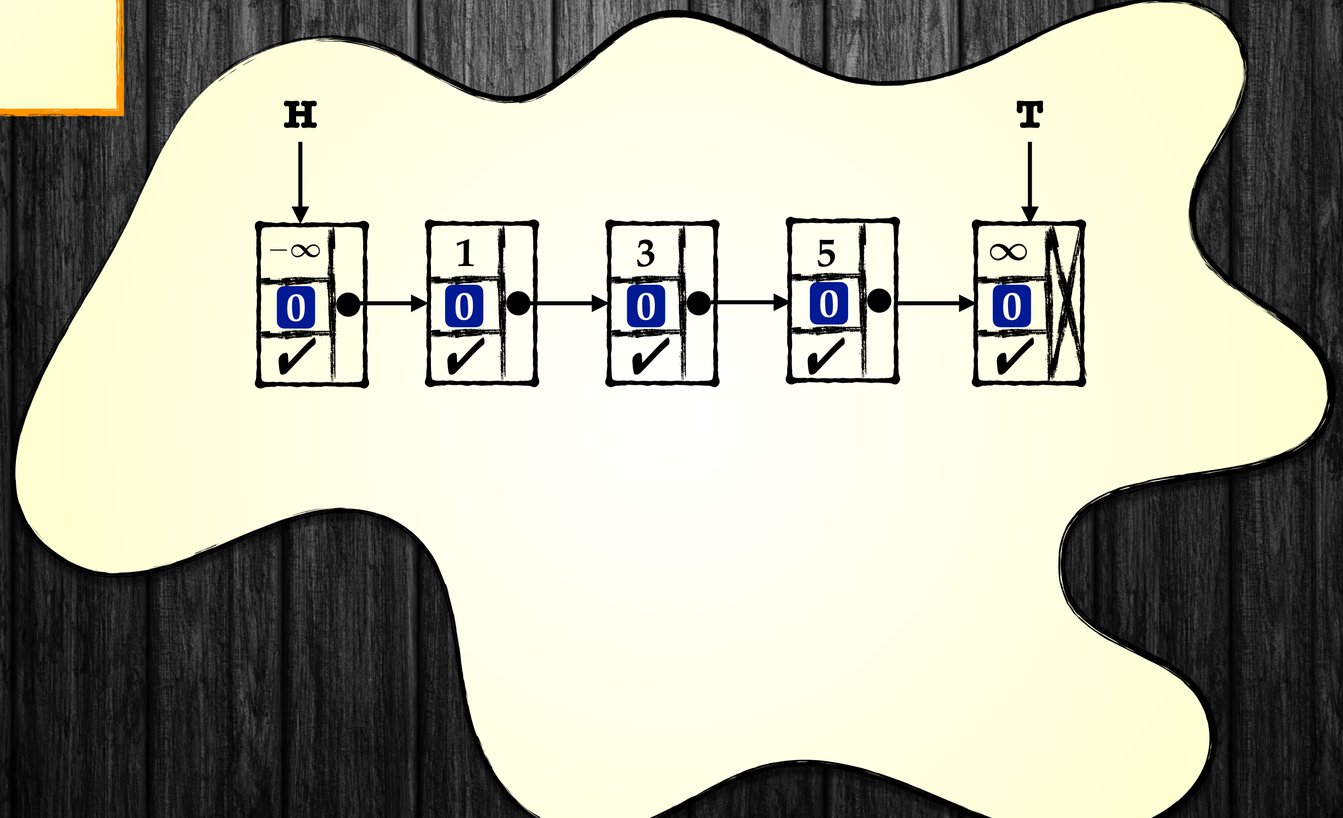




Sets | Lazy Synchronization

```
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;
```

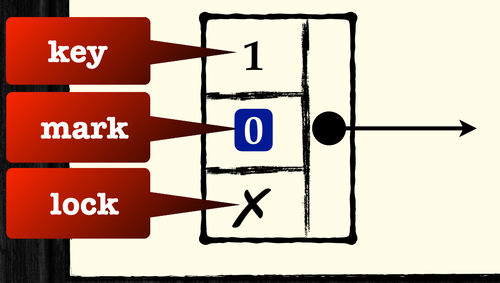


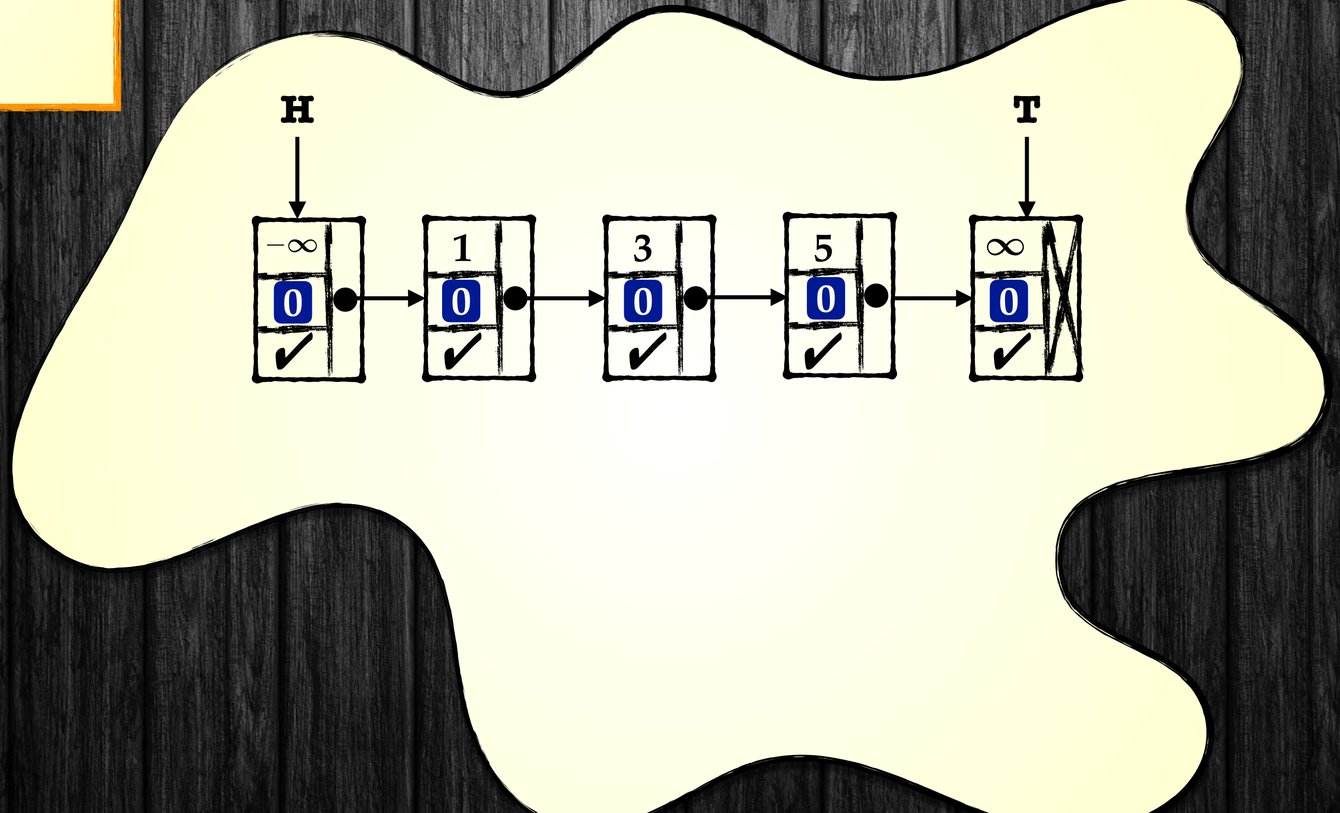


Sets | Lazy Synchronization

```
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;
```



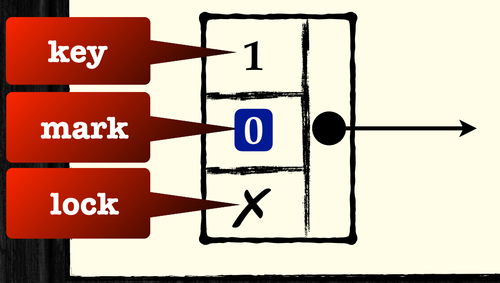


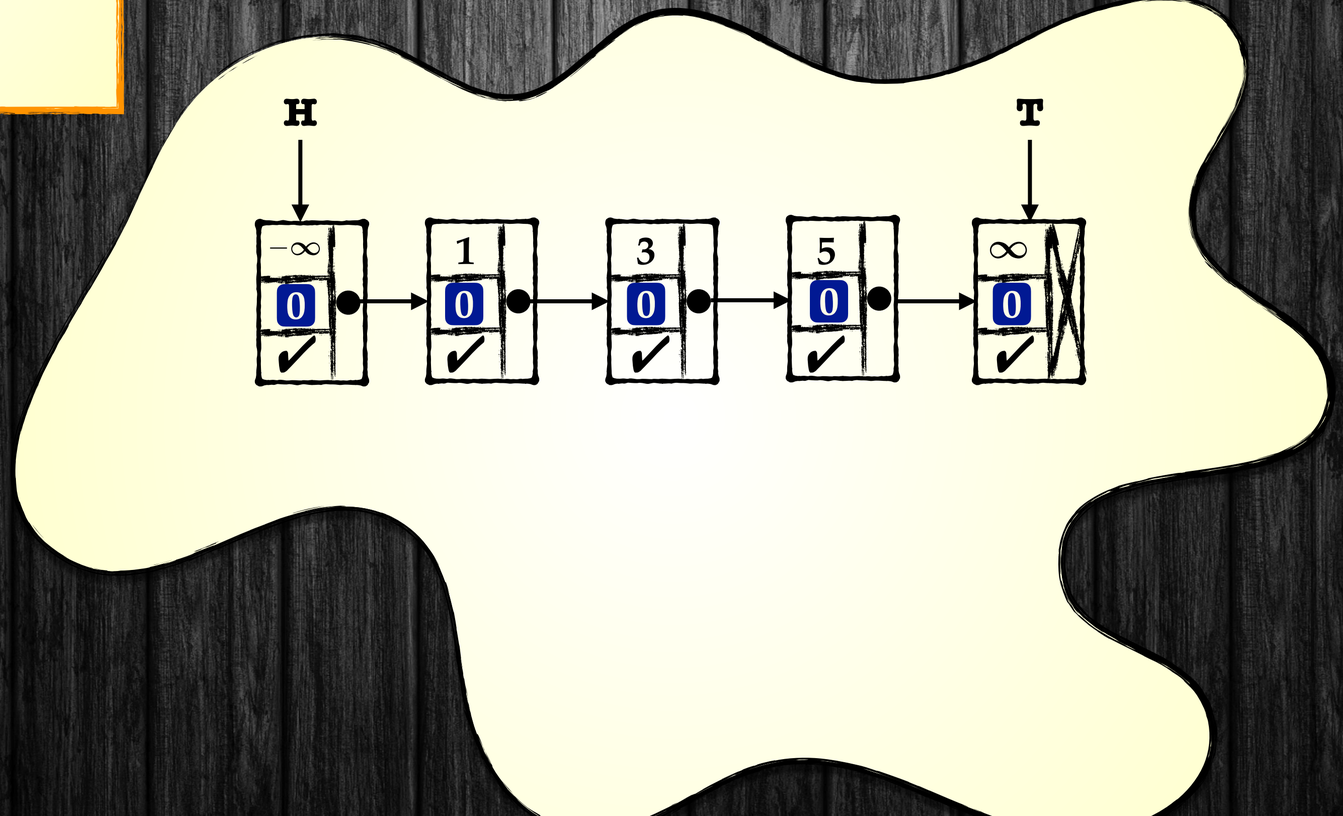


Sets | Lazy Synchronization

```
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;
```







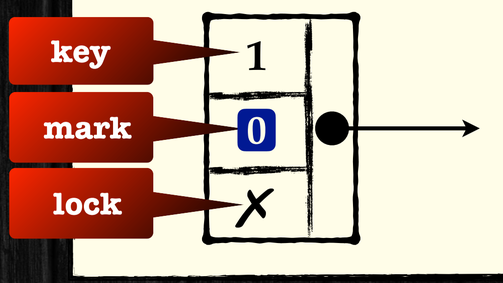
Sets | Lazy Synchronization

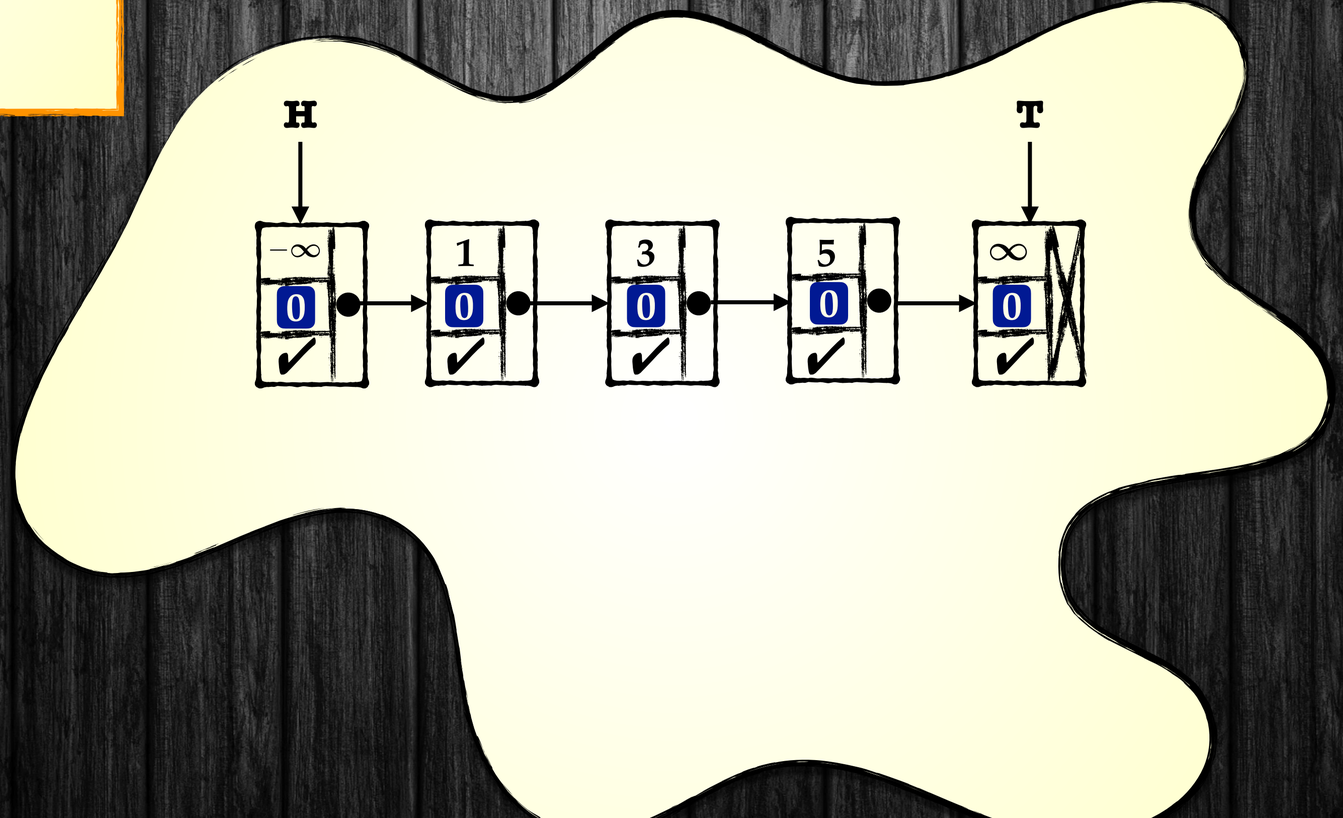
```
add(e):
   local p, c, n, r
   1  (p,c) := locate(e);
   2  if (c.val <> e)
   3     n := new Node(0,e,c,false);
   4     p.next := n;
   5     r := true;
   6  else r := false;
   7  unlock(p);
   8  unlock(c);
   9  return r;
```

result — Boolean



add(4)





Sets | Lazy Synchronization

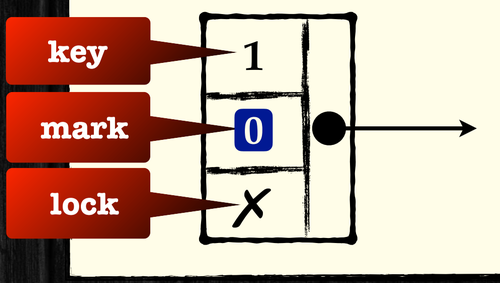
```
add(e):
   local p, c, n, r
   1  (p,c) := locate(e);
   2  if (c.val <> e)
   3     n := new Node(0,e,c,false);
   4     p.next := n;
   5     r := true;
   6  else r := false;
   7  unlock(p);
   8  unlock(c);
   9  return r;
```

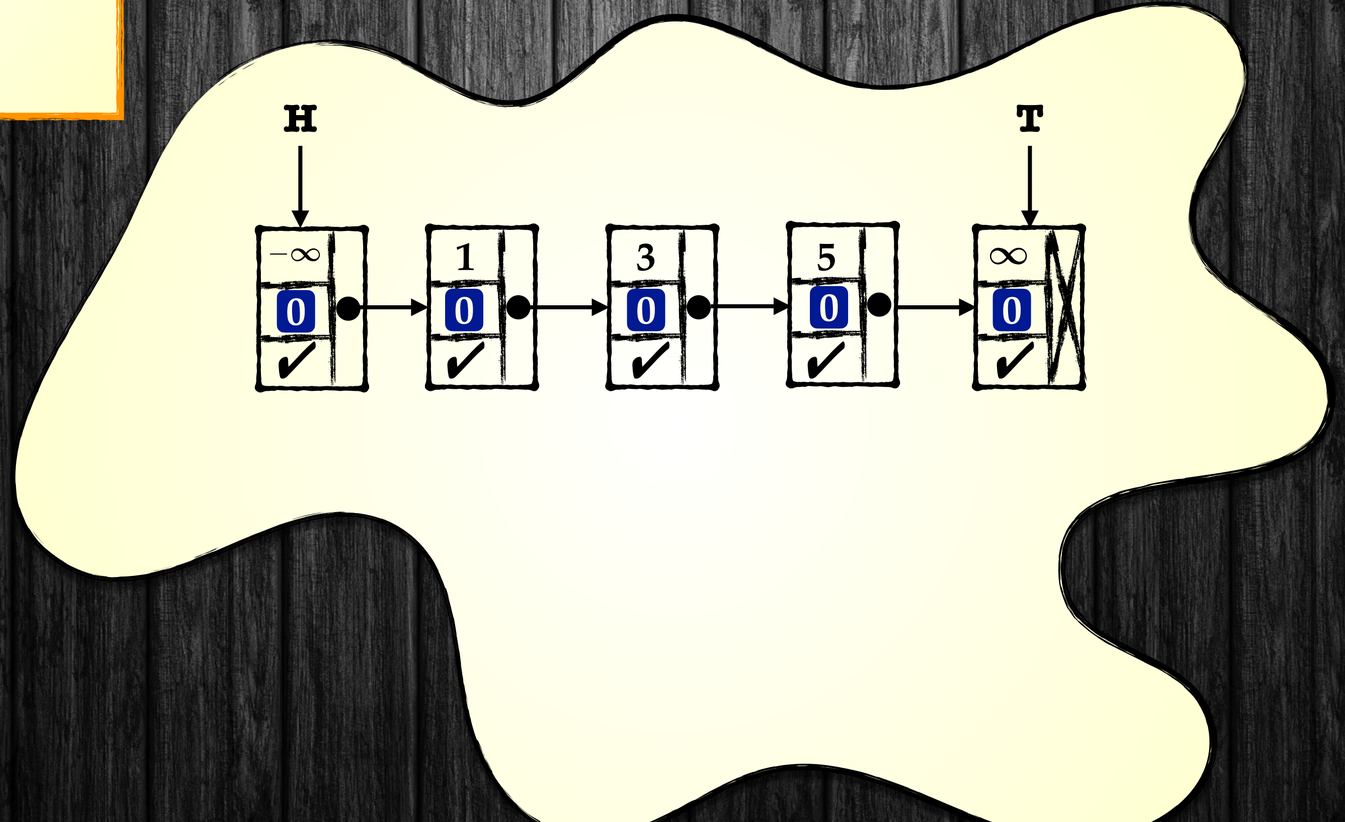
result

Boolean



add(4)

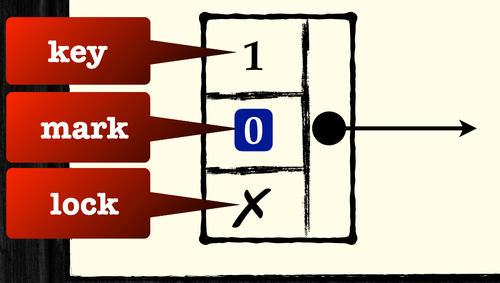


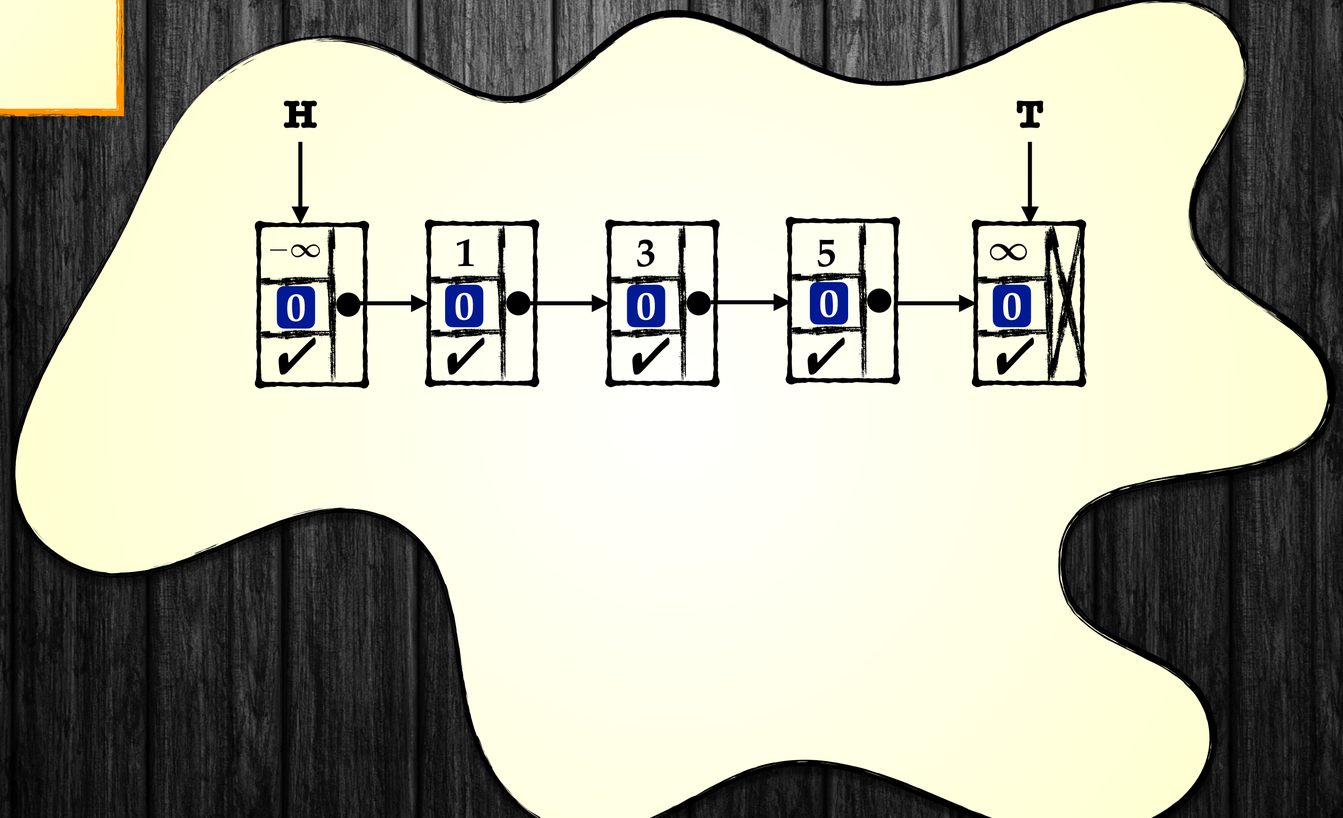


Sets | Lazy Synchronization

```
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;
```



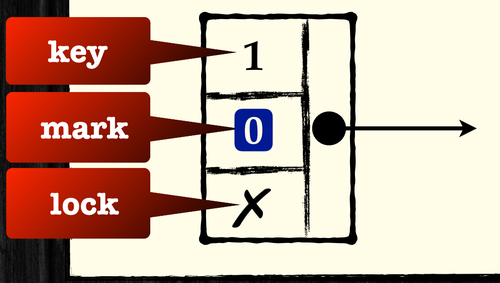


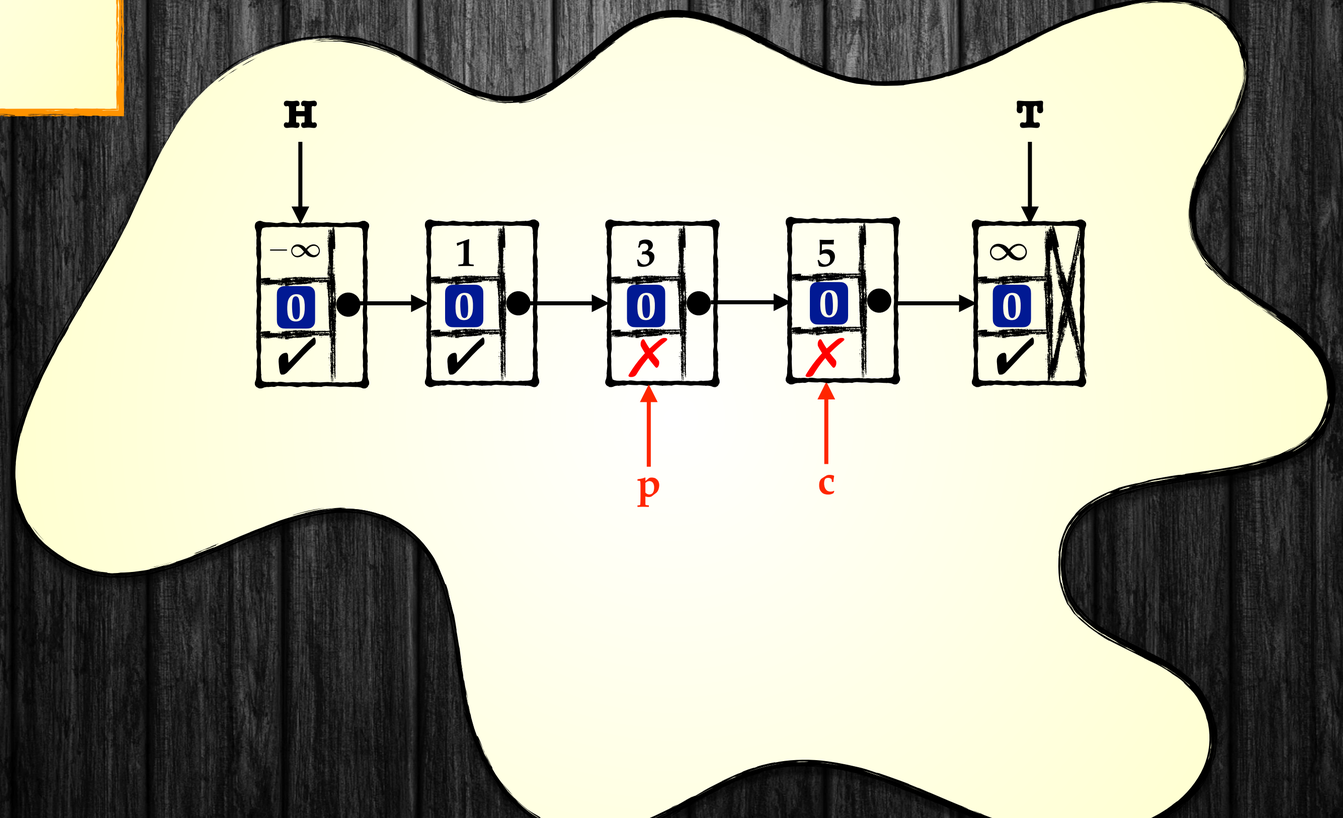


Sets | Lazy Synchronization

```
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;
```



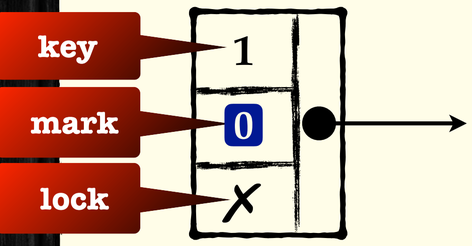


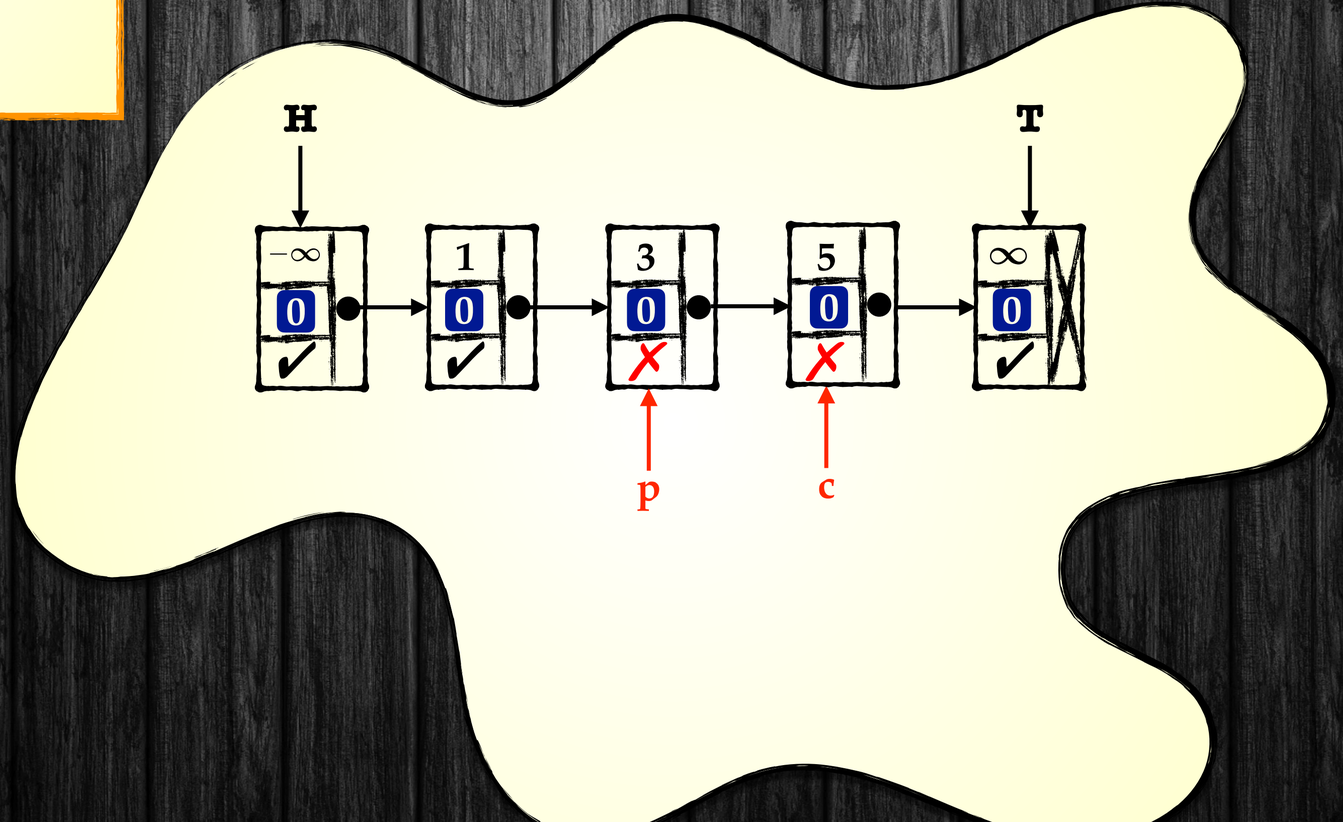


Sets | Lazy Synchronization

```
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)

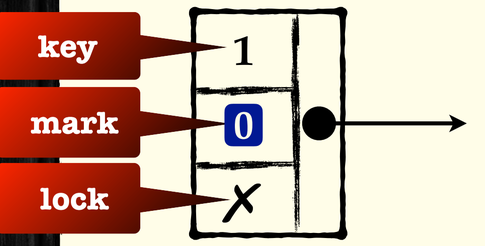


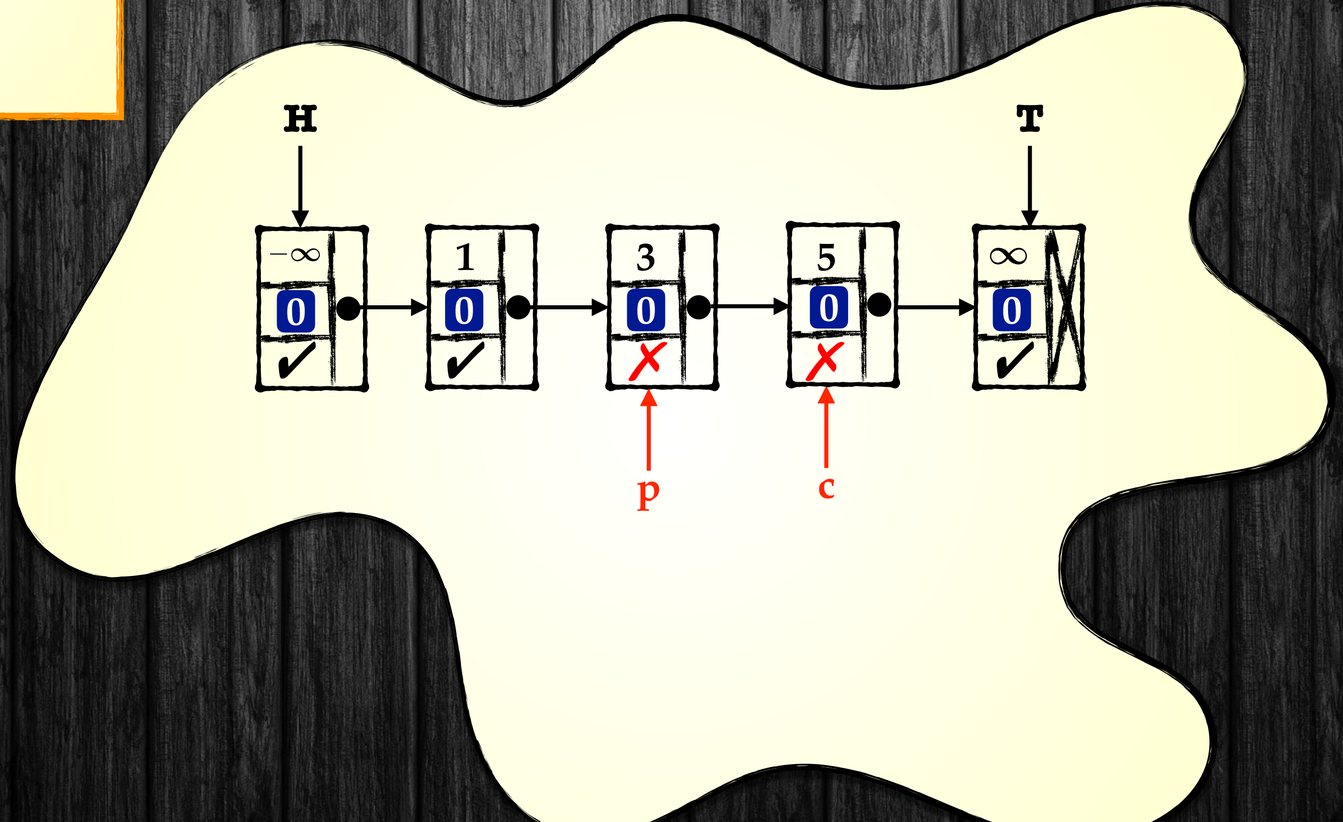


Sets | Lazy Synchronization

```
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)

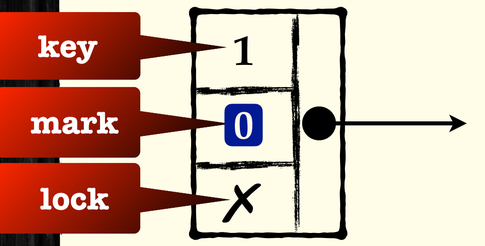


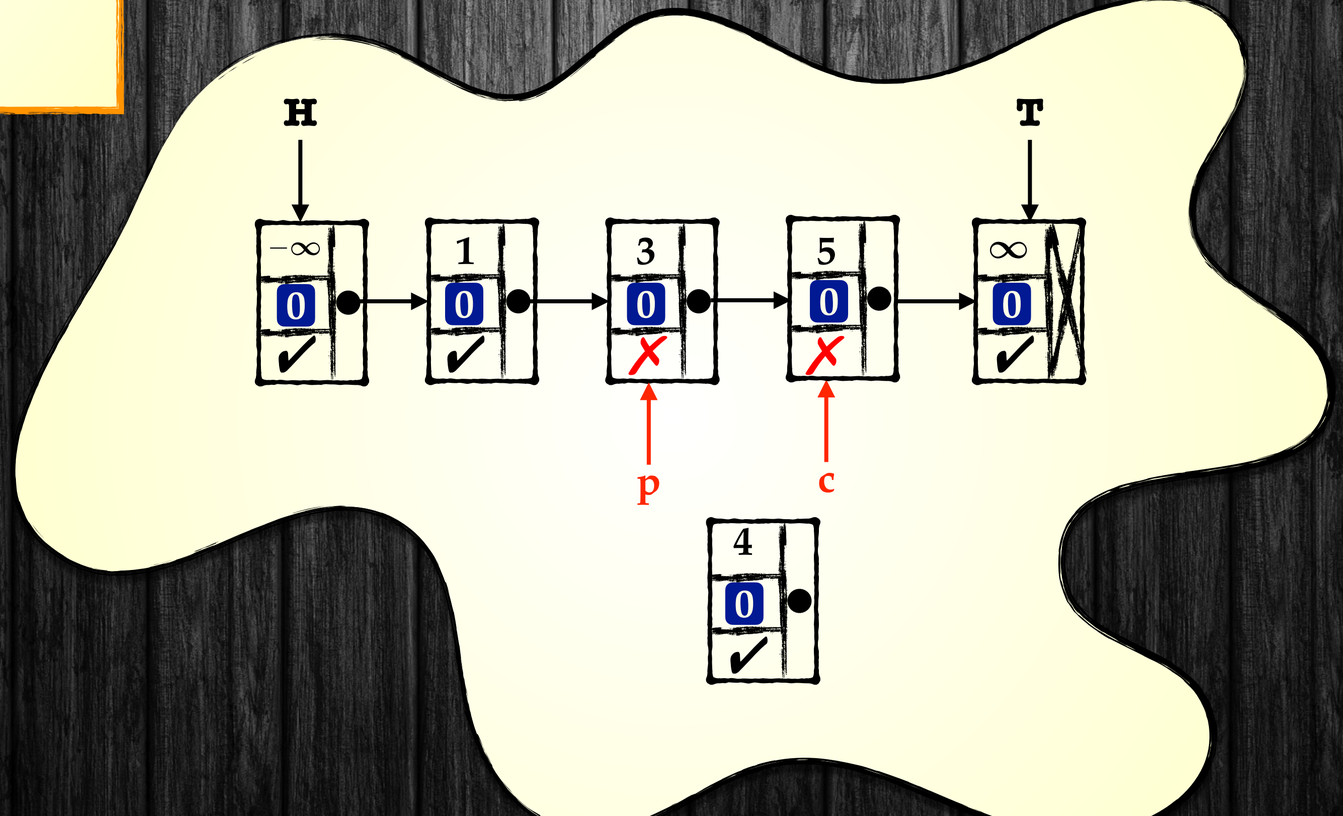


Sets | Lazy Synchronization

```
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)

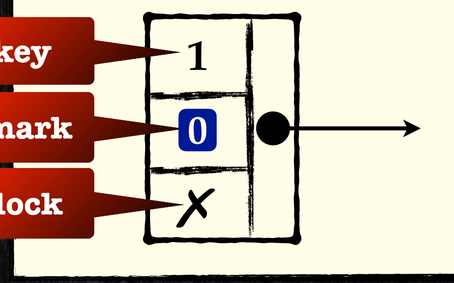


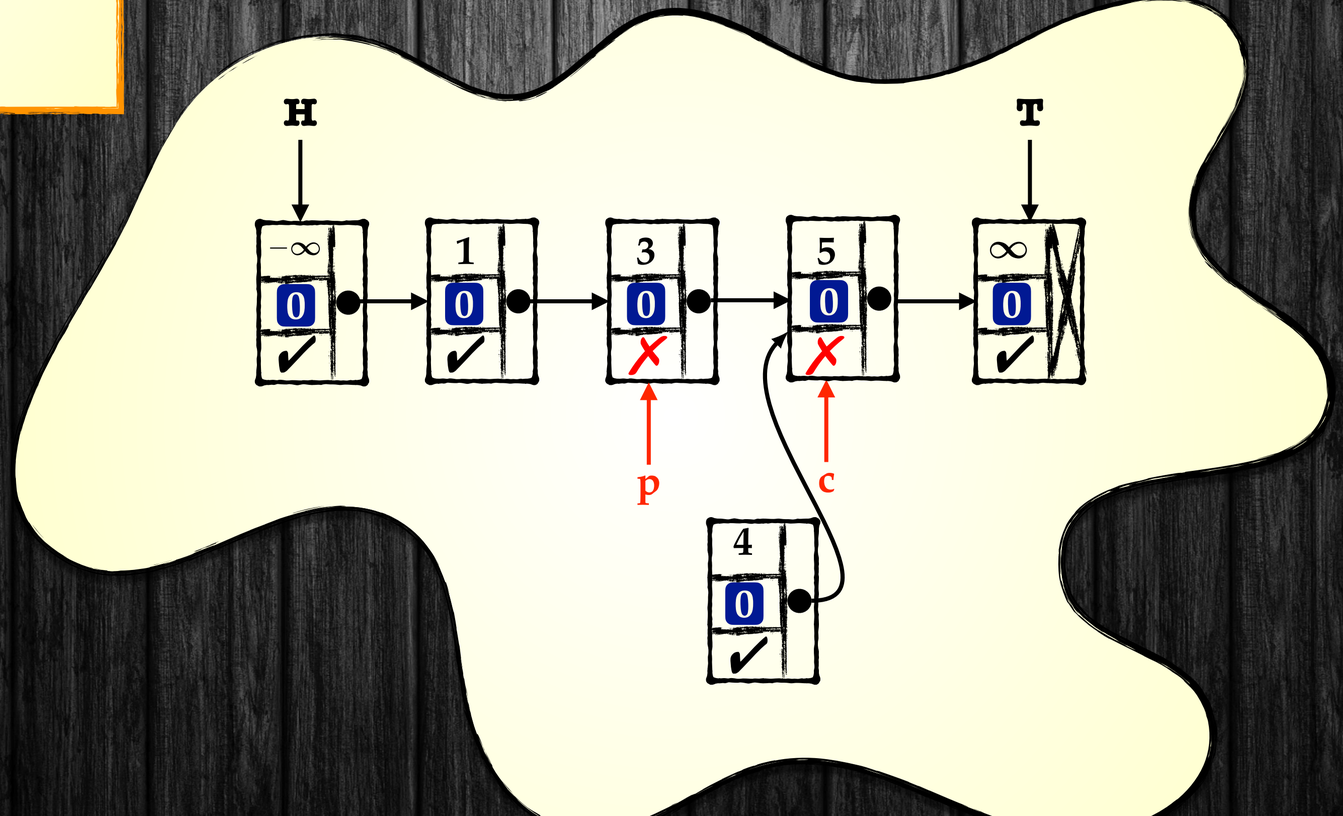


Sets | Lazy Synchronization

```
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)

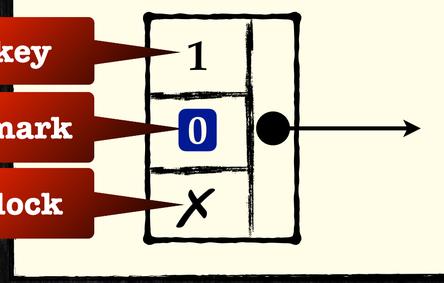


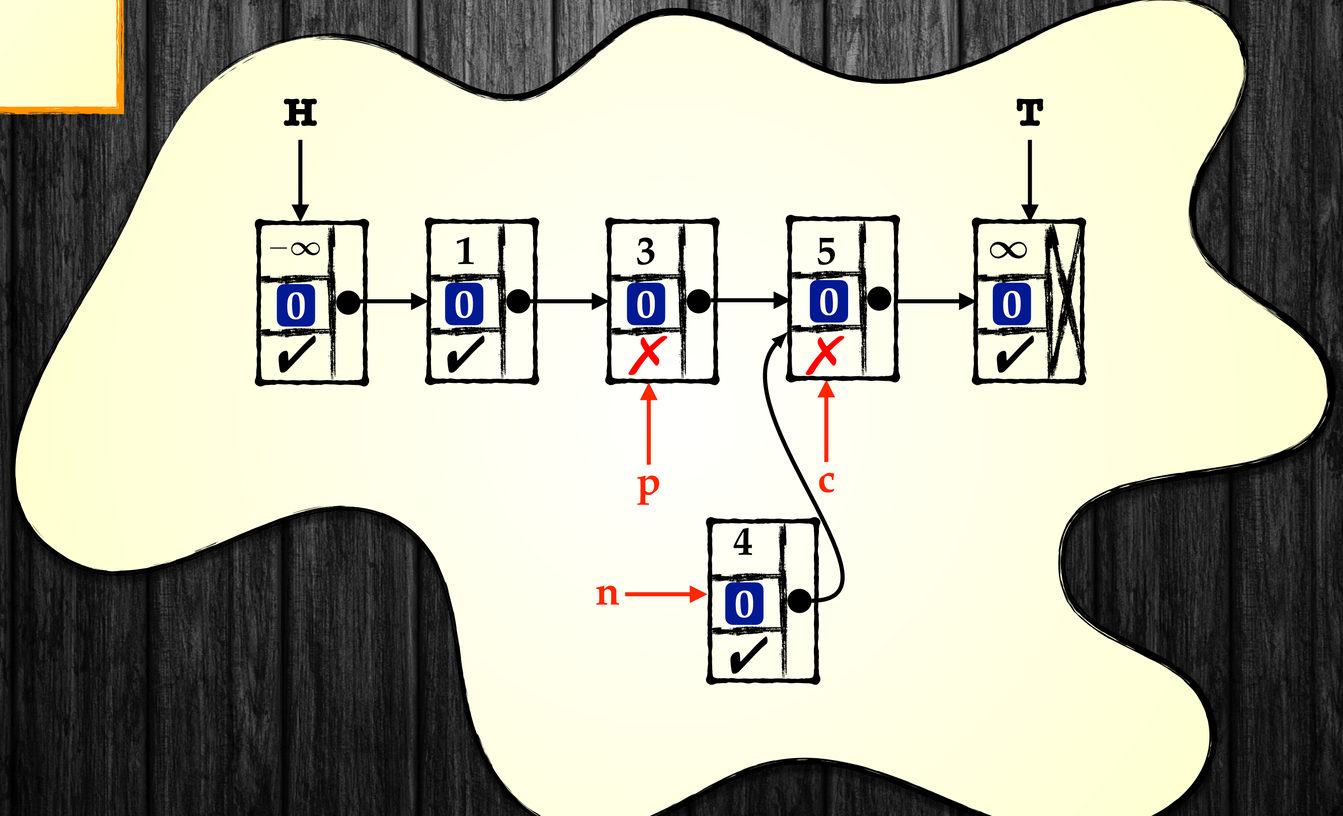


Sets | Lazy Synchronization

```
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)

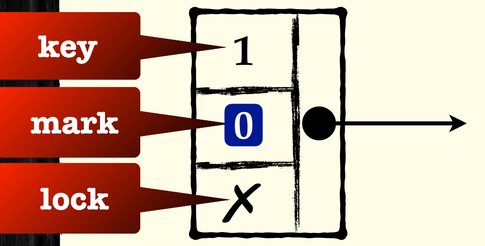


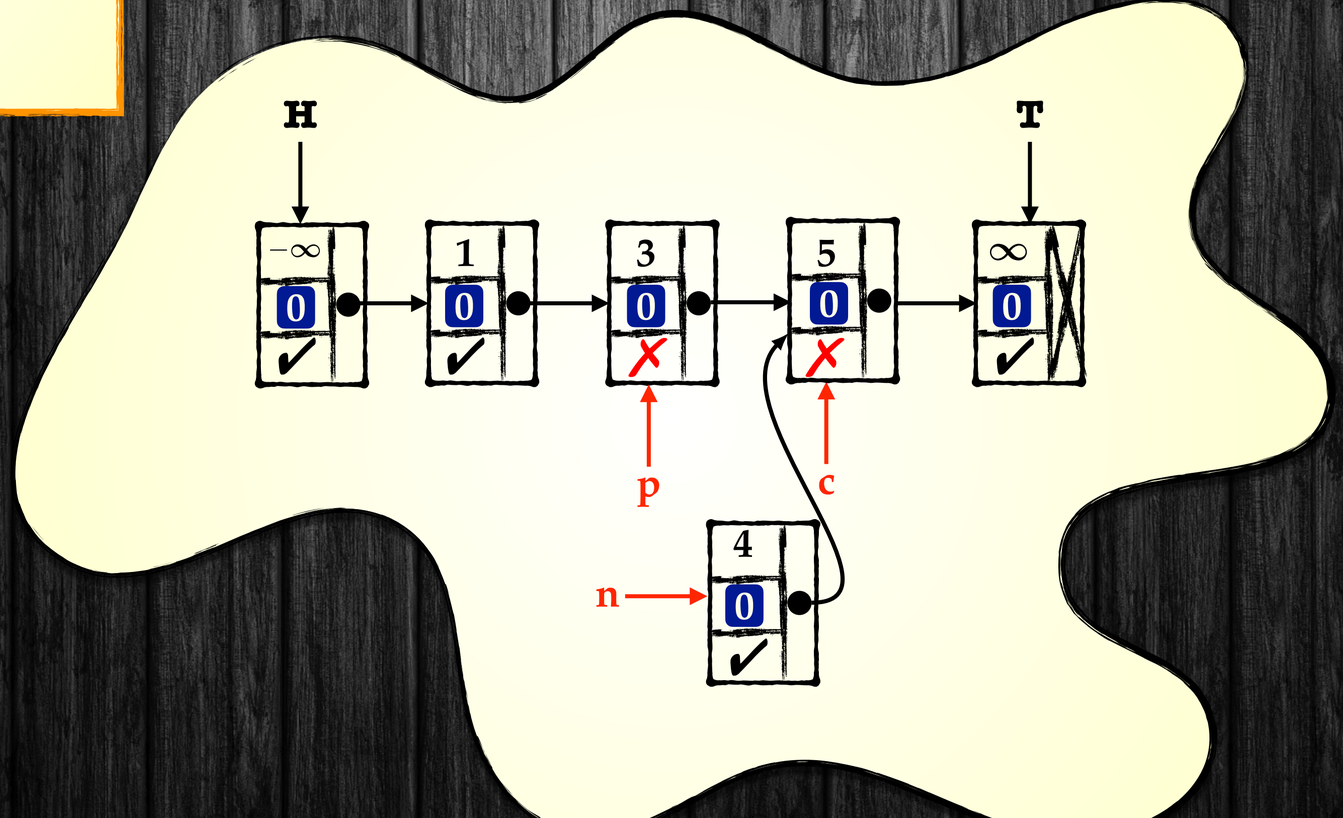


Sets | Lazy Synchronization

```
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)

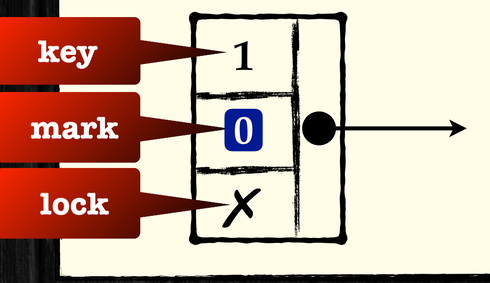


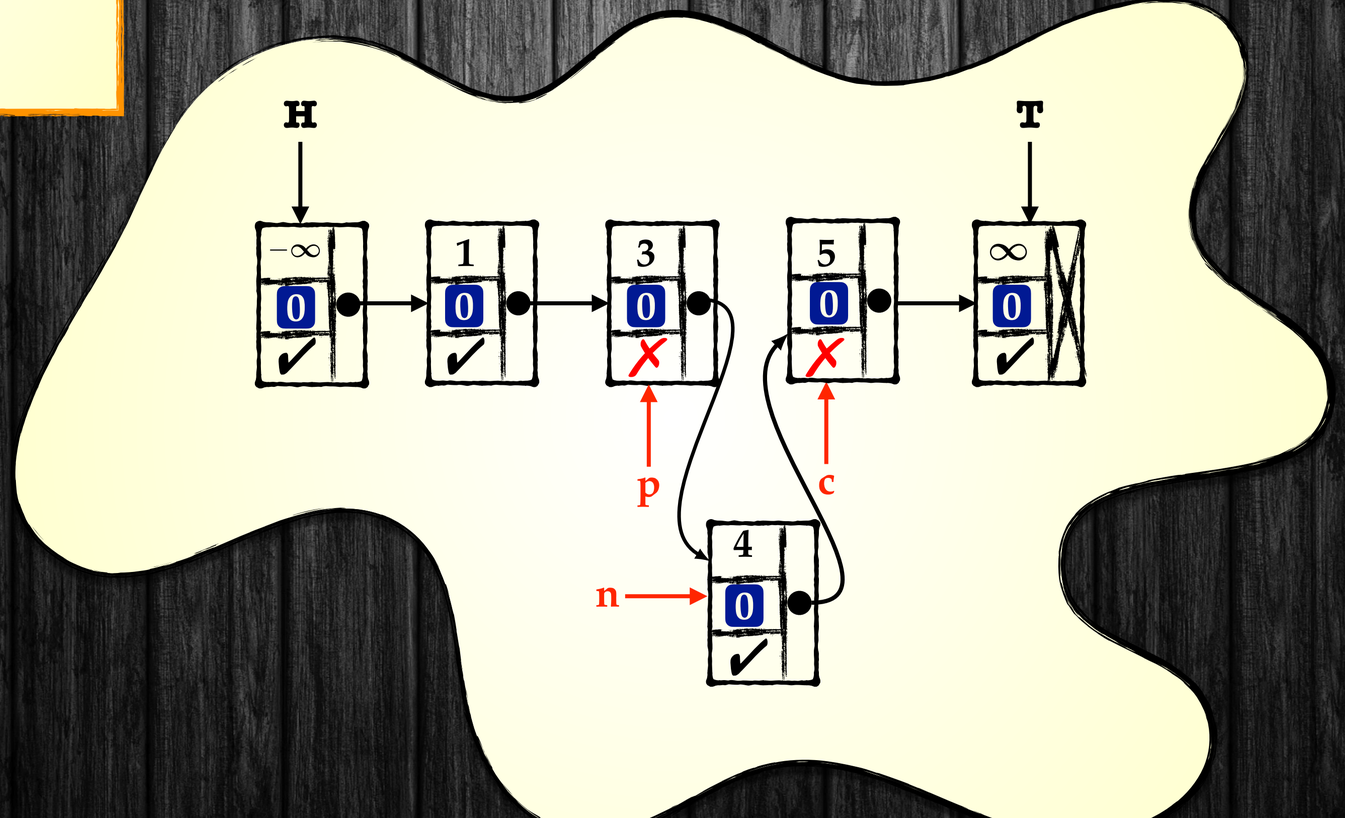


Sets | Lazy Synchronization

```
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)

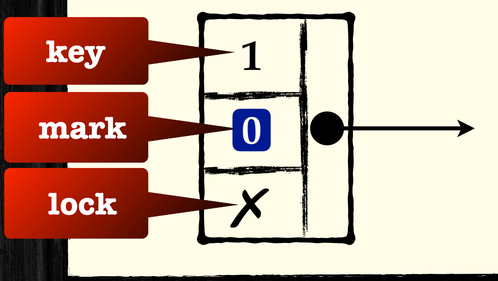




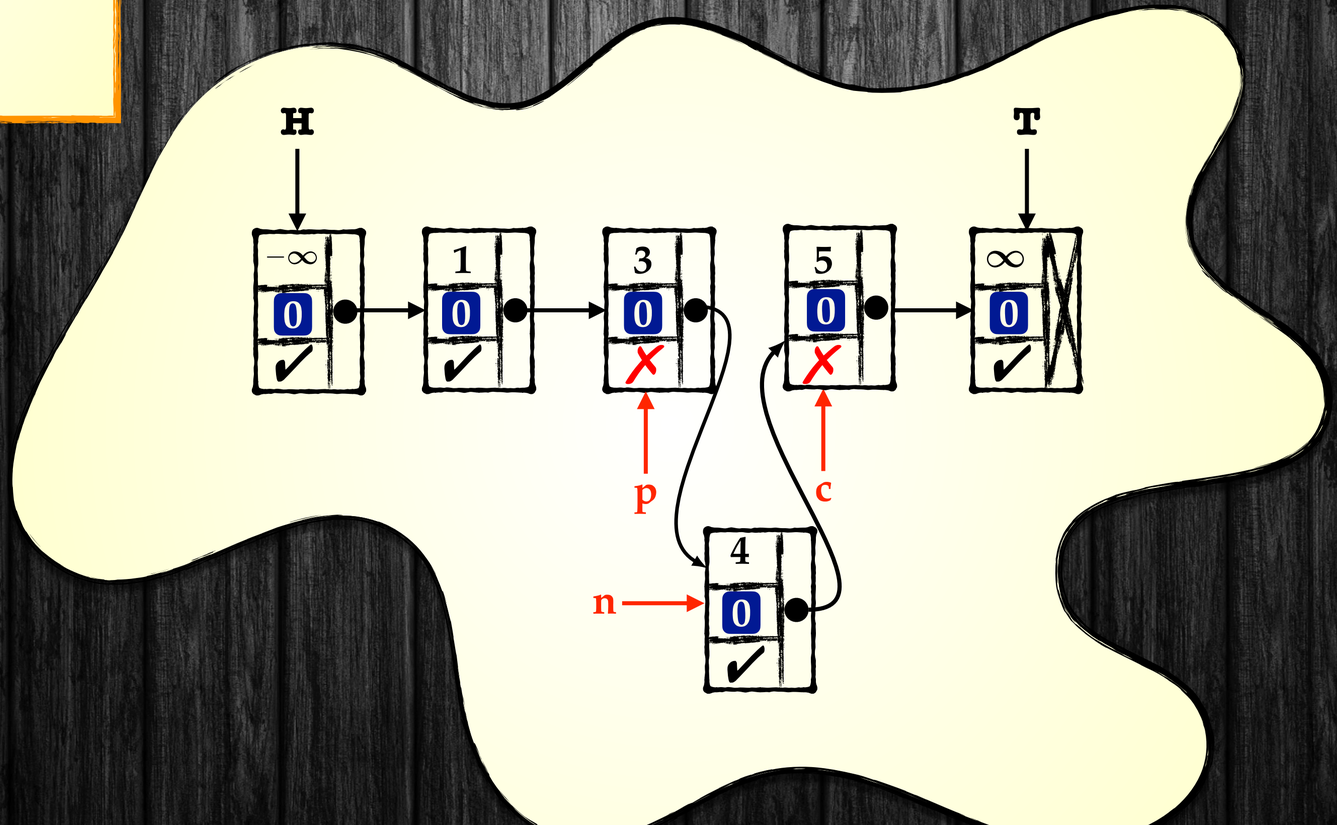
Sets | Lazy Synchronization

```
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;
```





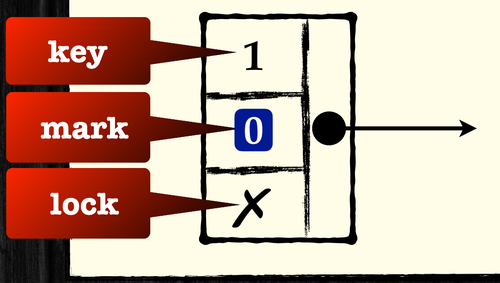
linearization point



Sets | Lazy Synchronization

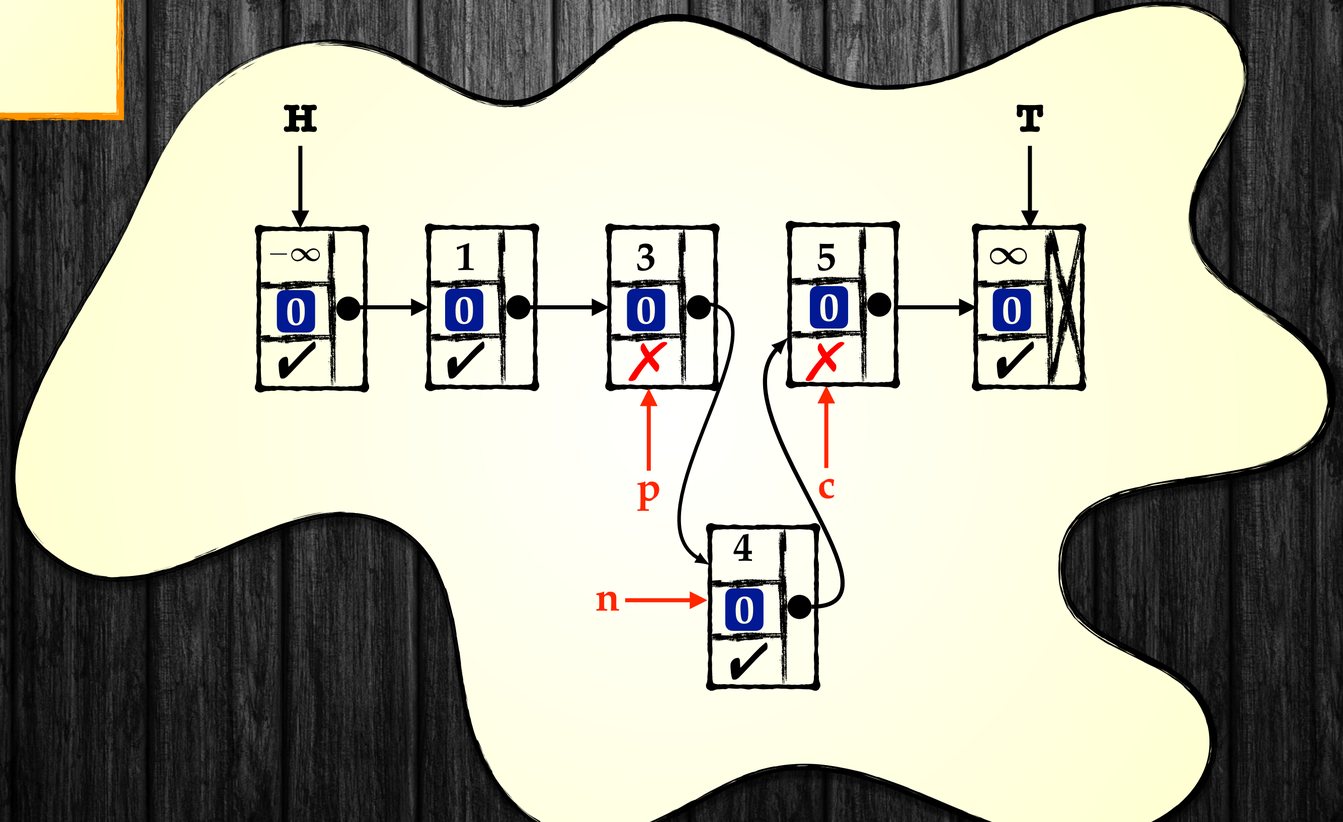
```
add(e):
   local p, c, n, r
1  (p,c) := locate(e);
2  if (c.val <> e)
3     n := new Node(0,e,c,false);
4     p.next := n;
5     r := true;
6  else r := false;
7  unlock(p);
8  unlock(c);
9  return r;
```





successful add

linearization point



Sets | Lazy Synchronization

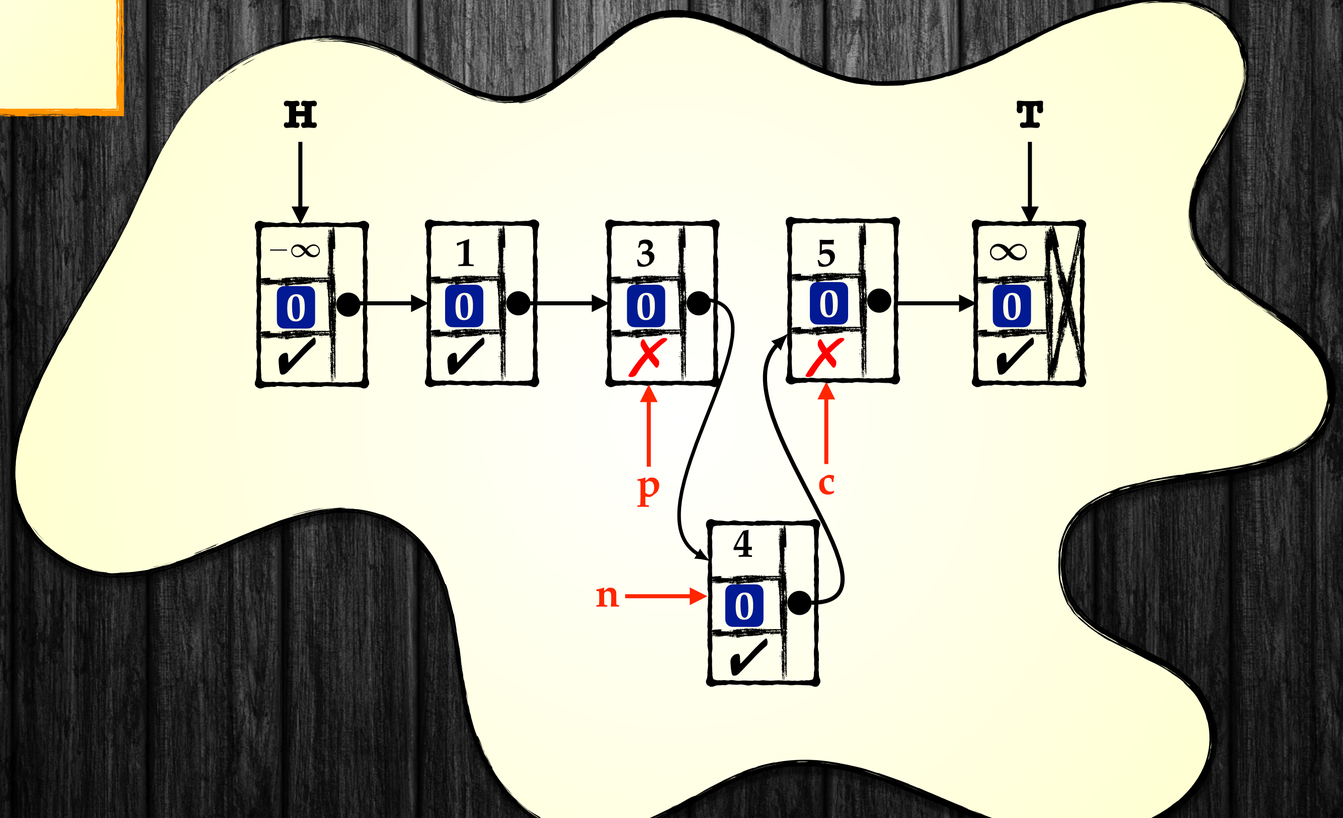
```
add(e):
   local p, c, n, r
1  (p,c) := locate(e);
2  if (c.val <> e)
3     n := new Node(0,e,c,false);
4     p.next := n;
5     r := true;
6  else r := false;
7  unlock(p);
8  unlock(c);
9  return r;
```

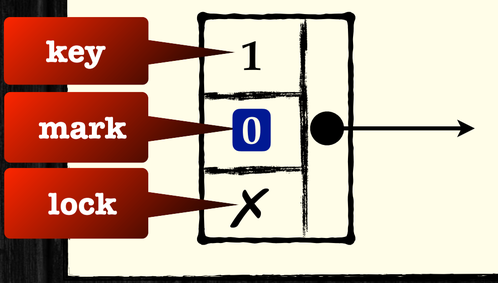
add(4)

"when p.next is set"

successful add

linearization point





Sets | Lazy Synchronization

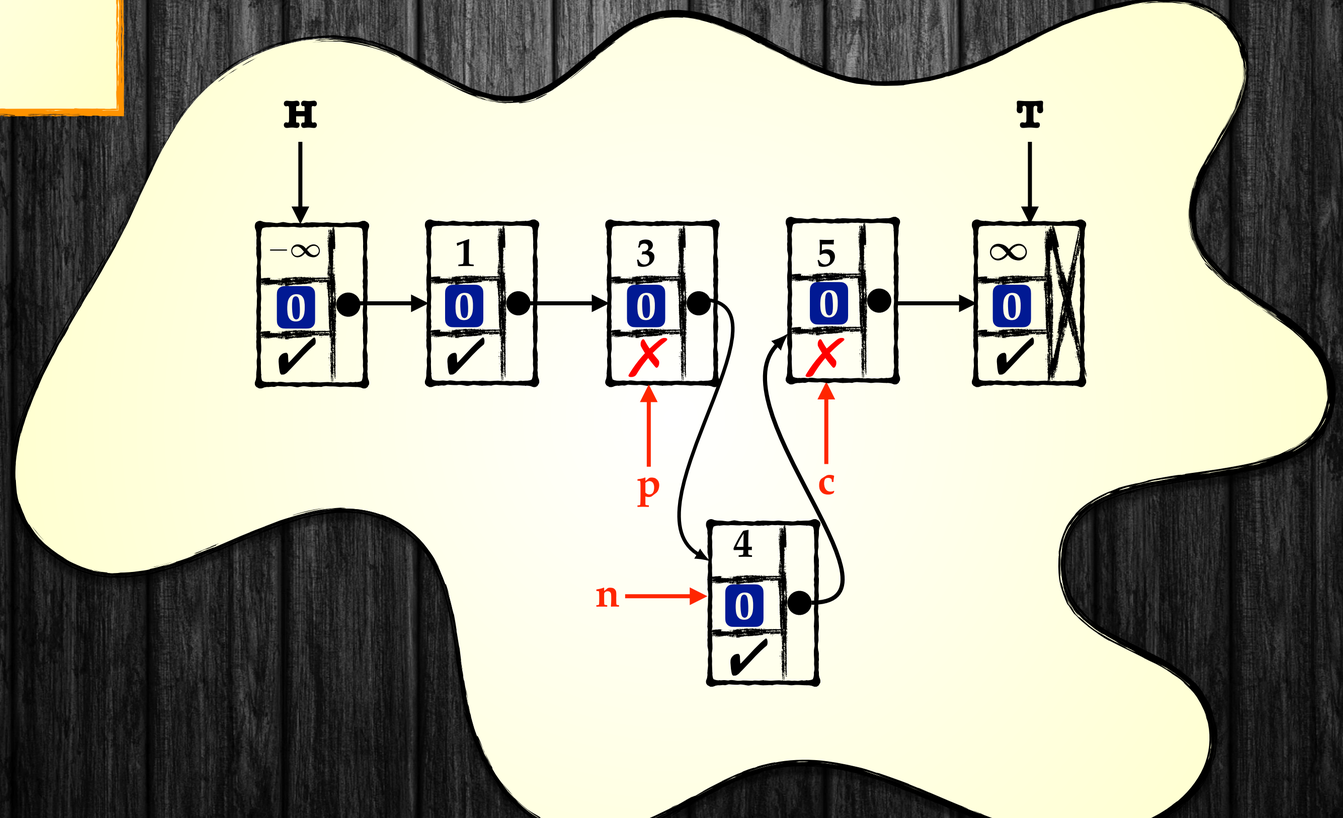
```
add(e):
   local p, c, n, r
1  (p,c) := locate(e);
2  if (c.val <> e)
3     n := new Node(0,e,c,false);
4     p.next := n;
5     r := true;
6  else r := false;
7  unlock(p);
8  unlock(c);
9  return r;
```

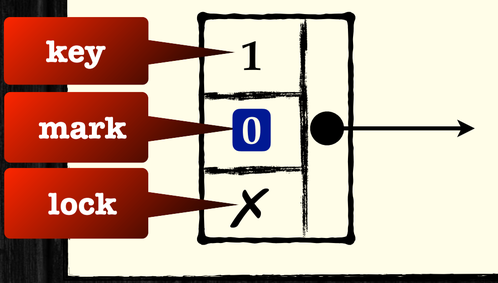
add(4)

"when p.next is set"

successful add

linearization point

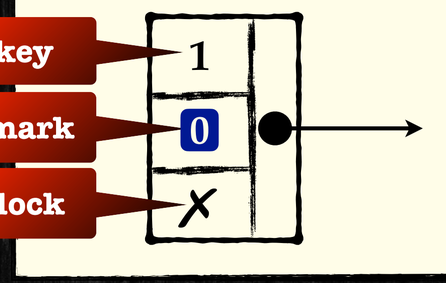


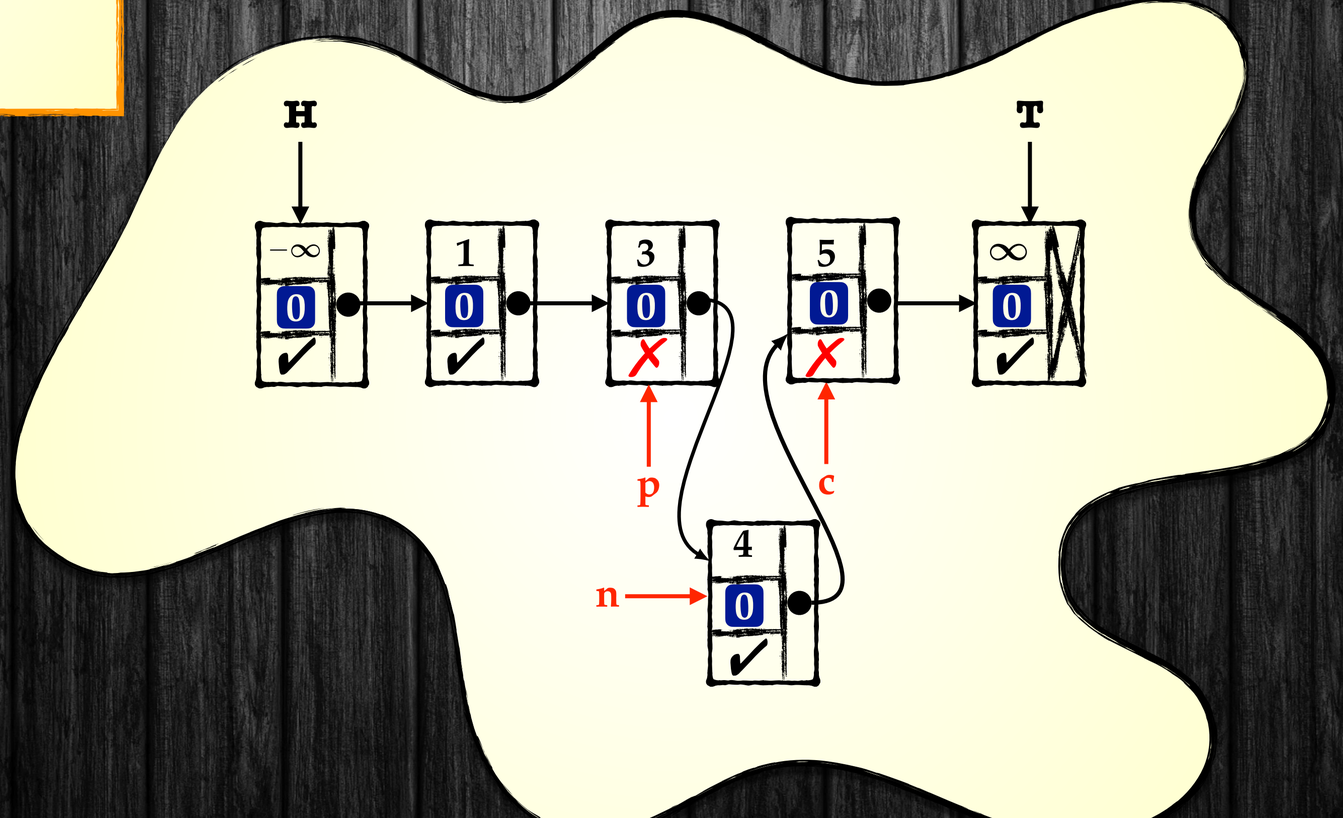


Sets | Lazy Synchronization

```
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)

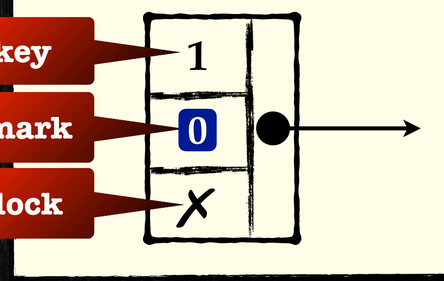


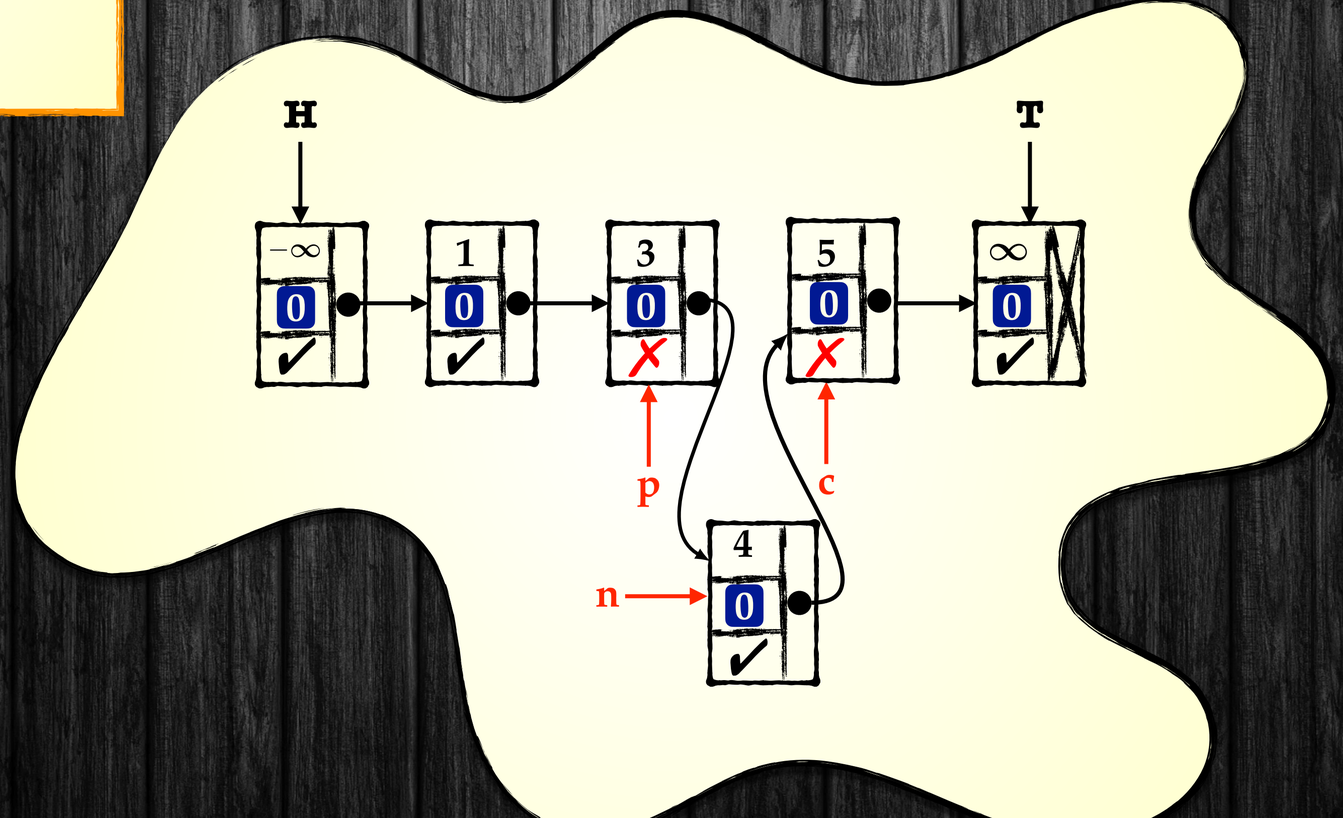


Sets | Lazy Synchronization

```
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)

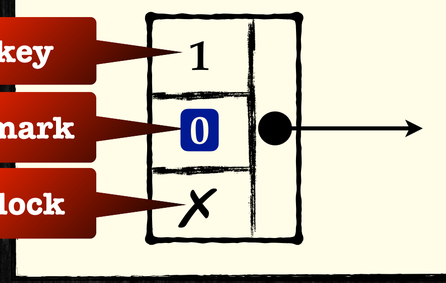


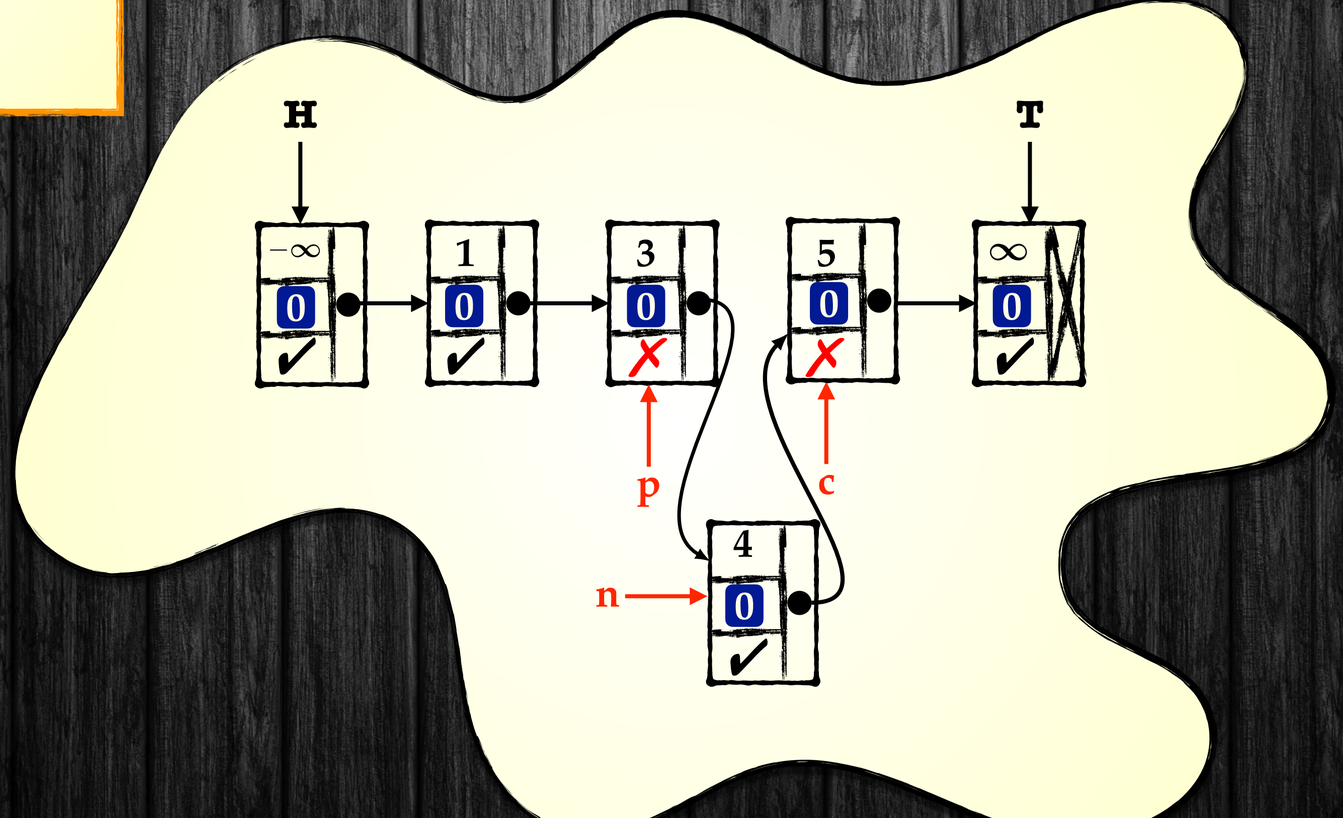


Sets | Lazy Synchronization

```
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)

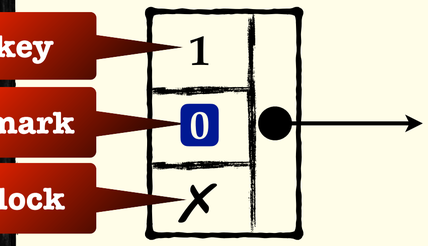


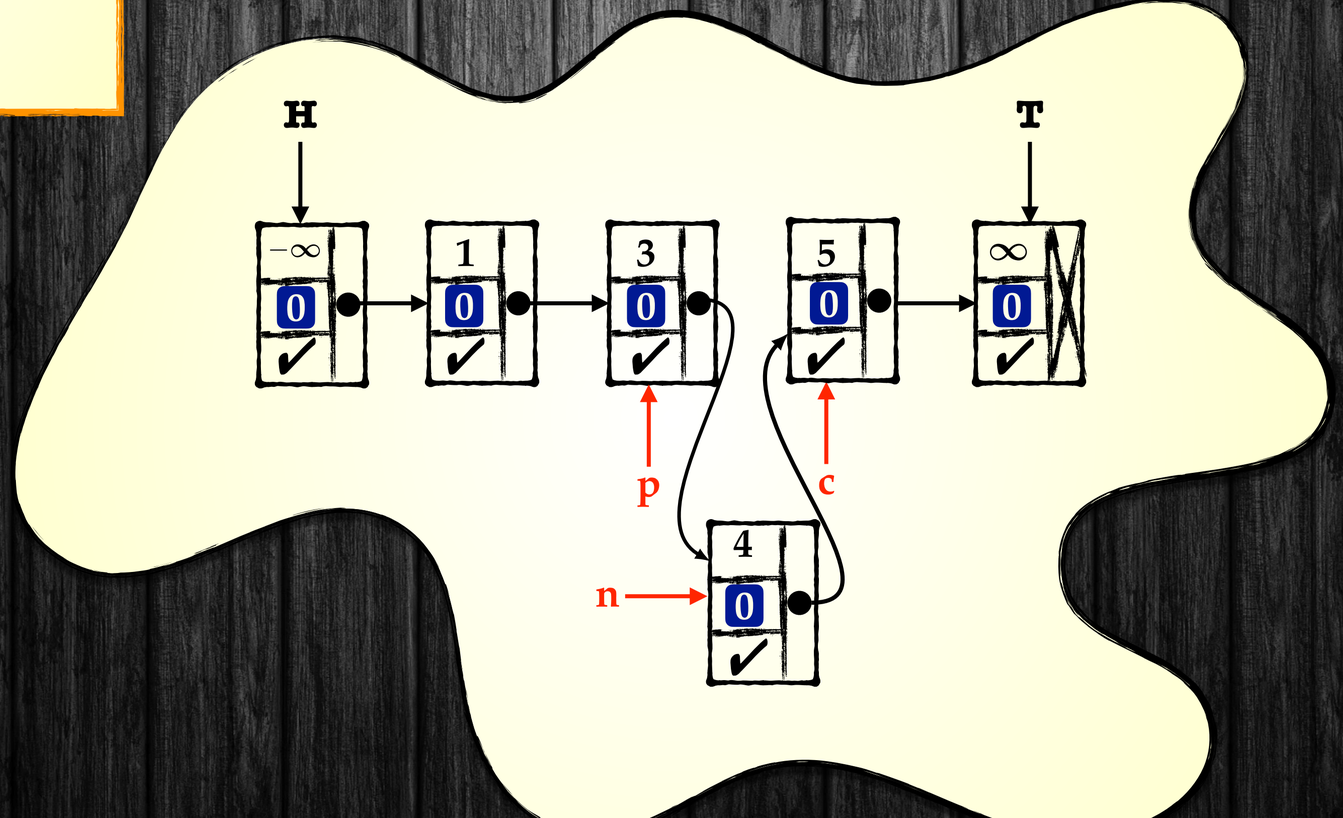


Sets | Lazy Synchronization

```
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)

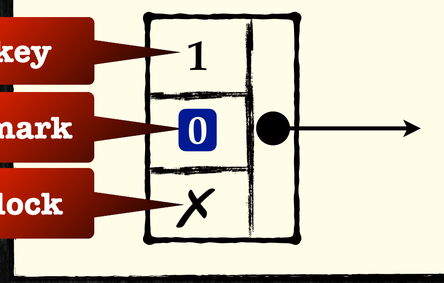


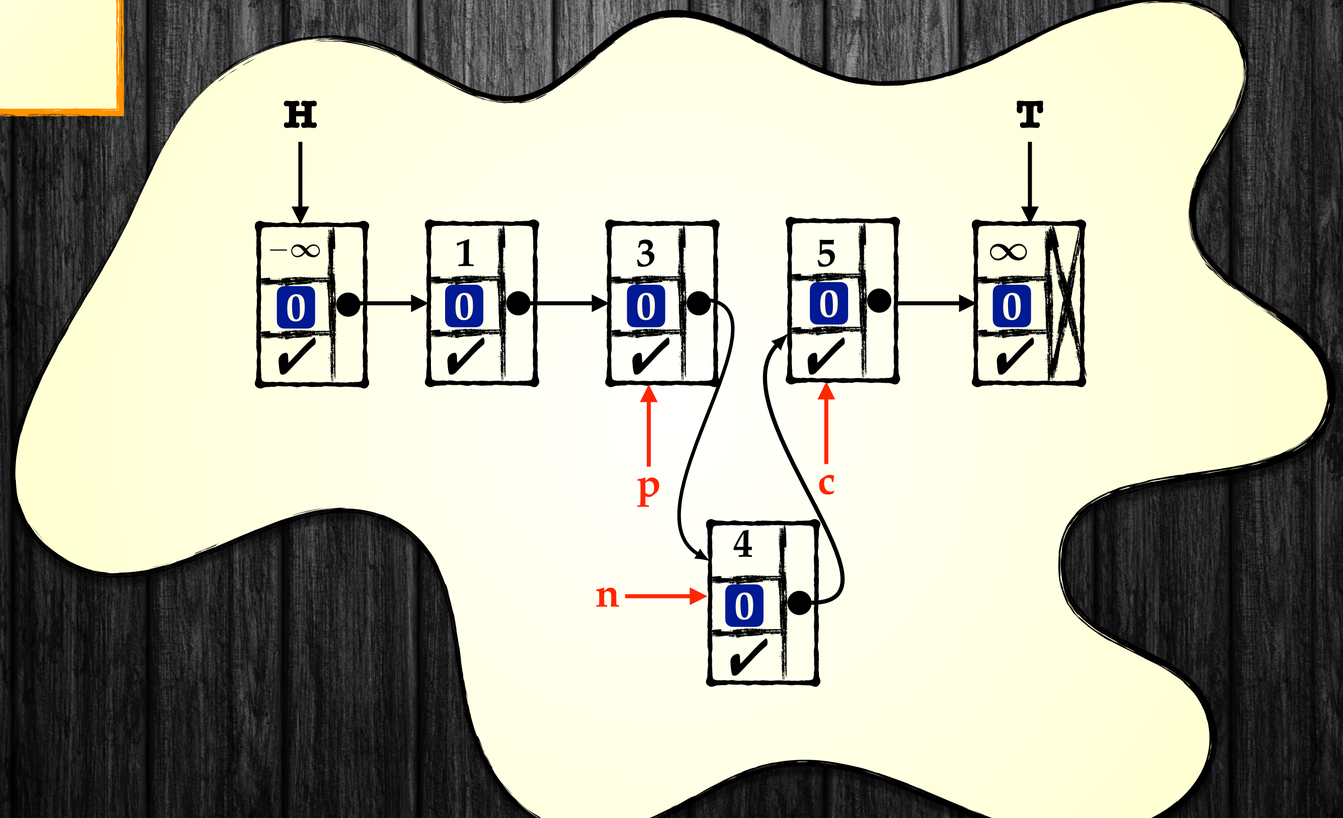


Sets | Lazy Synchronization

```
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)

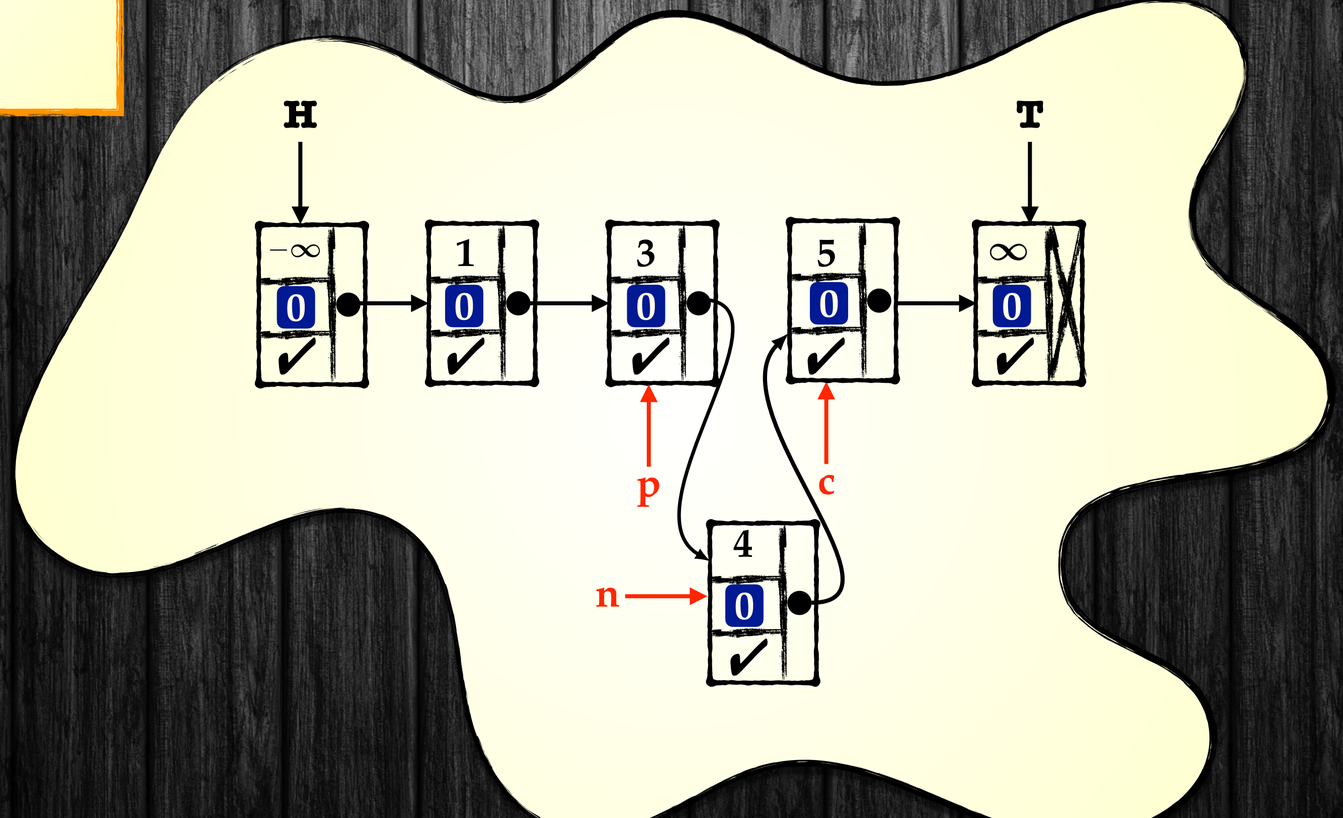


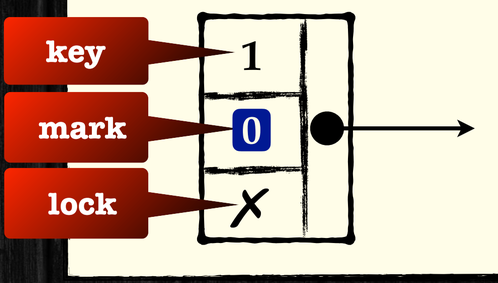


Sets | Lazy Synchronization

```
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)

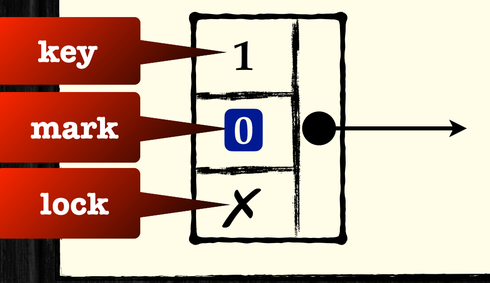


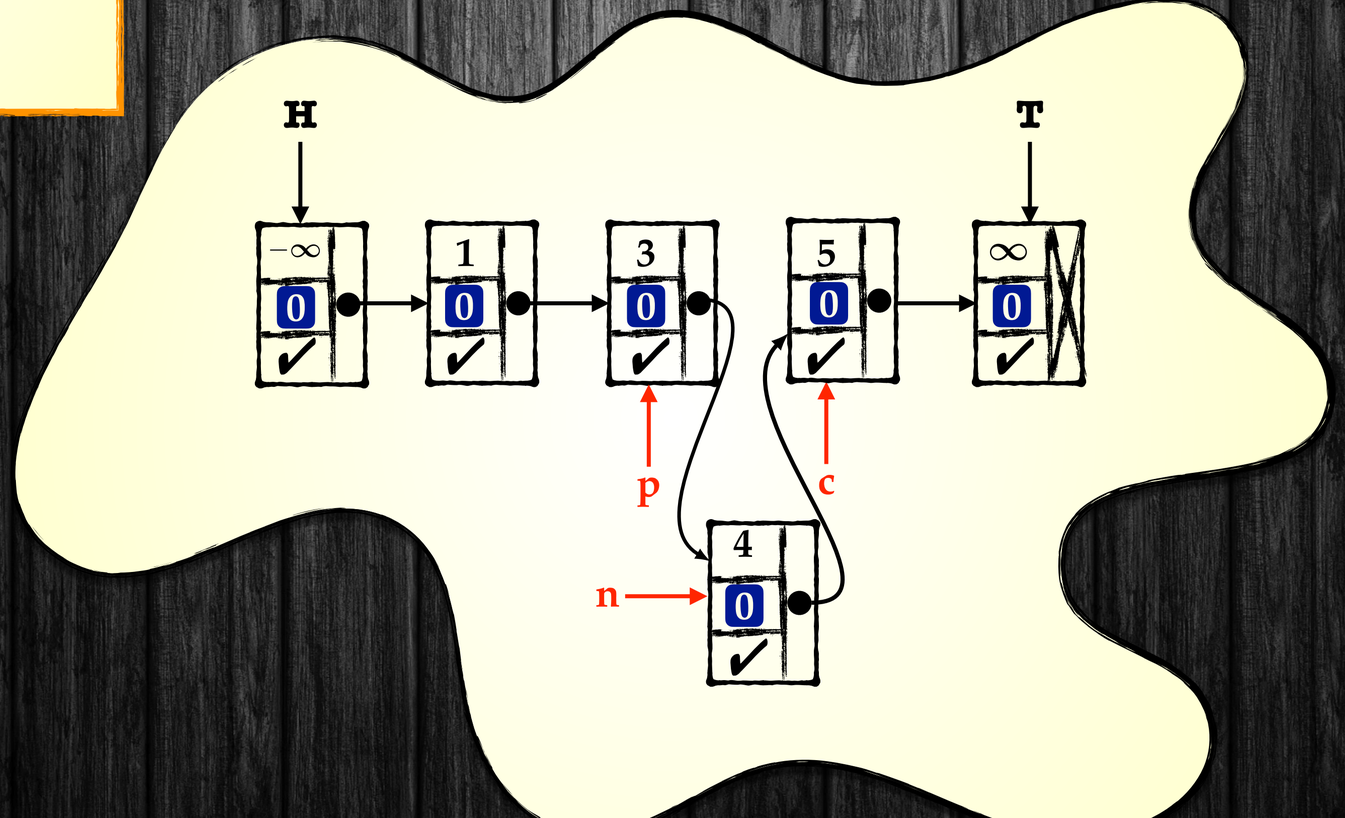


Sets | Lazy Synchronization

```
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)

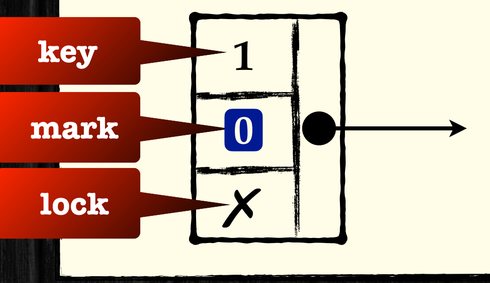


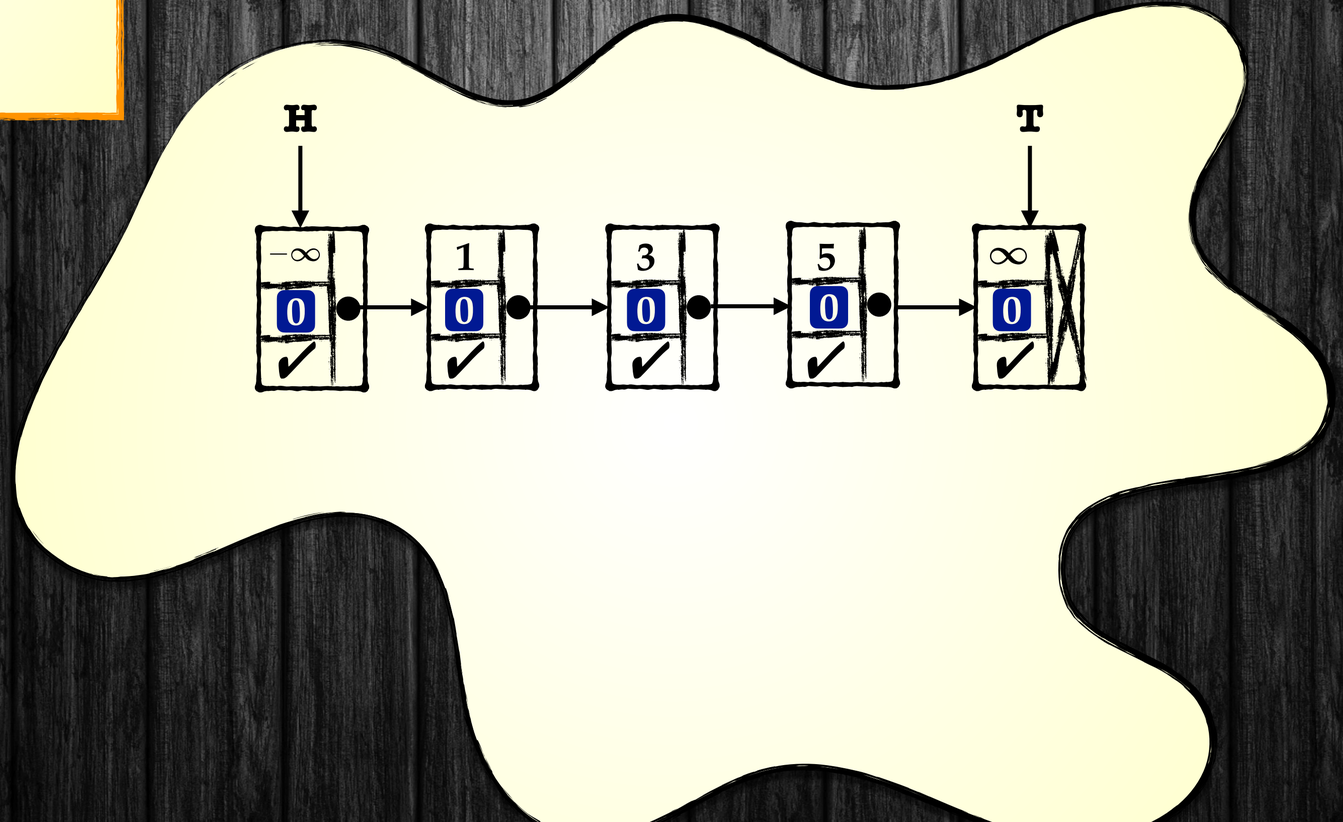


Sets | Lazy Synchronization

```
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)

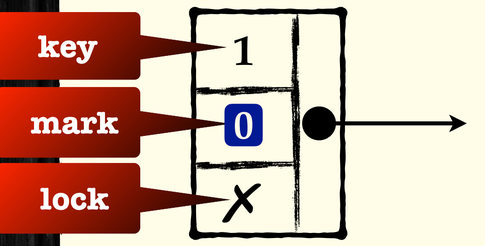


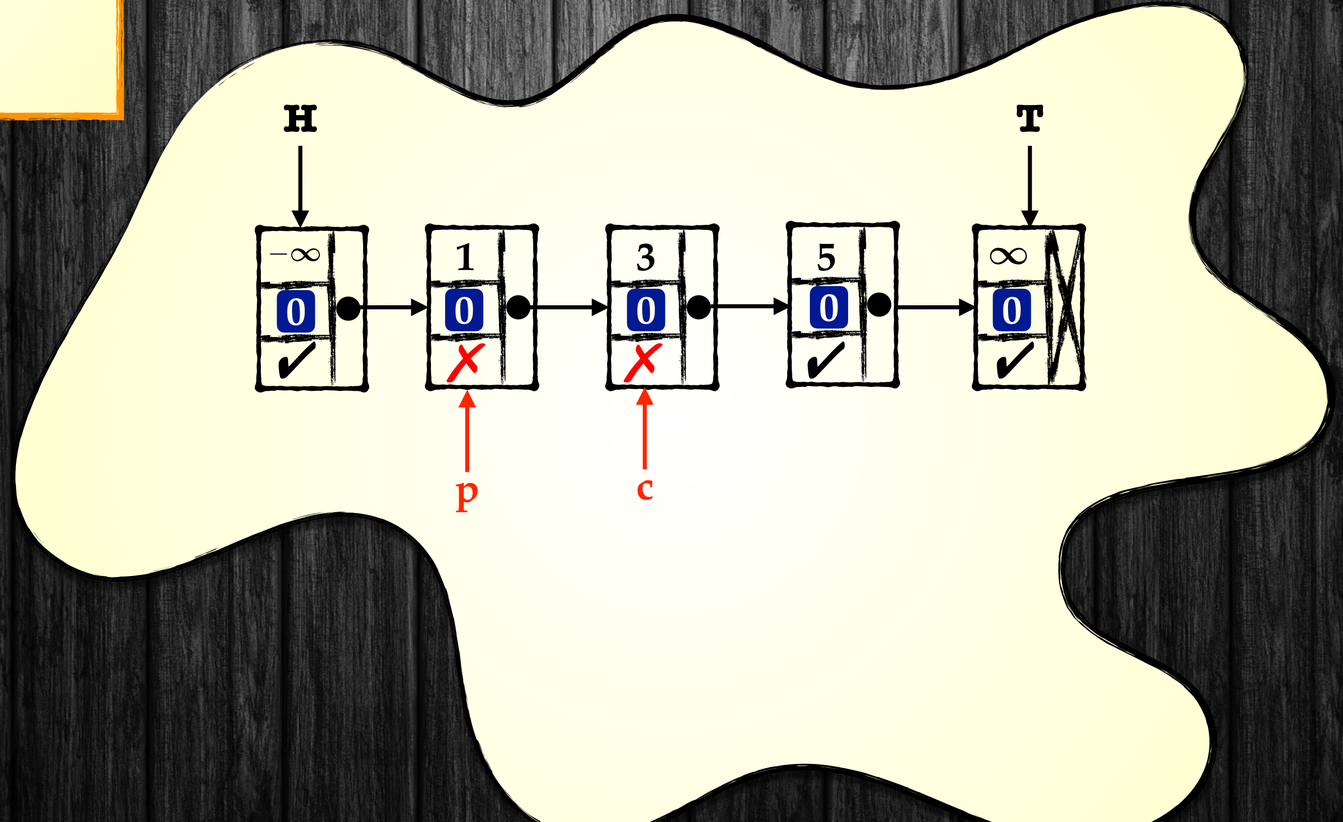


Sets | Lazy Synchronization

```
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)

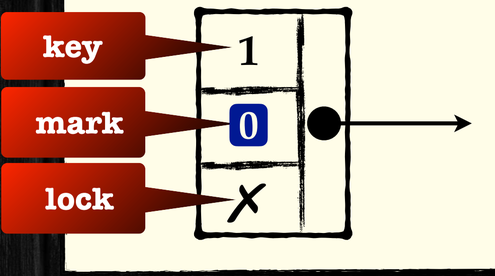


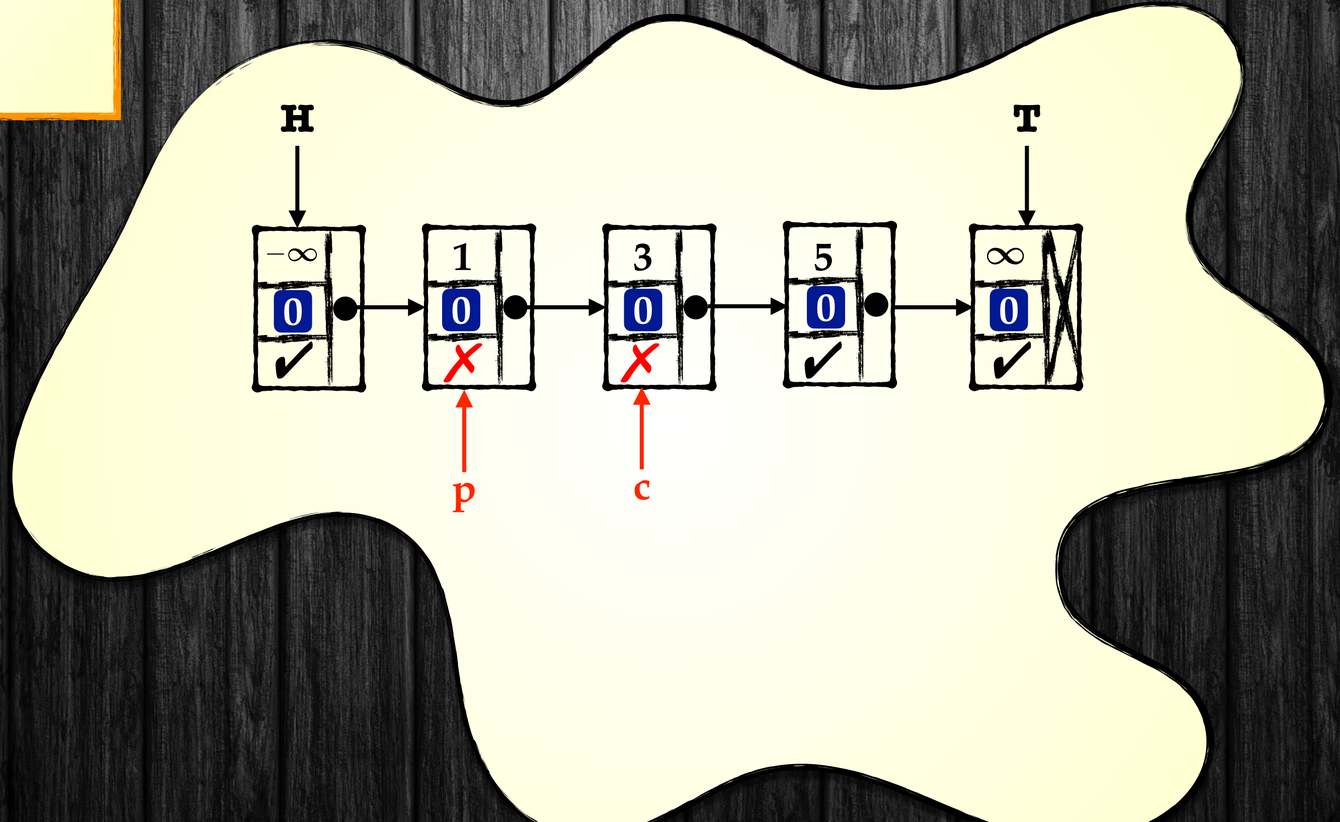


Sets | Lazy Synchronization

```
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;
```





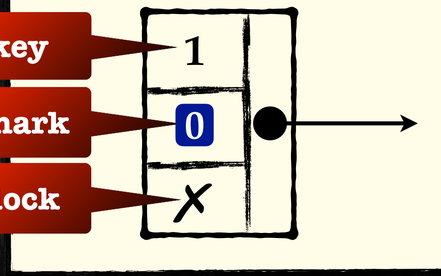


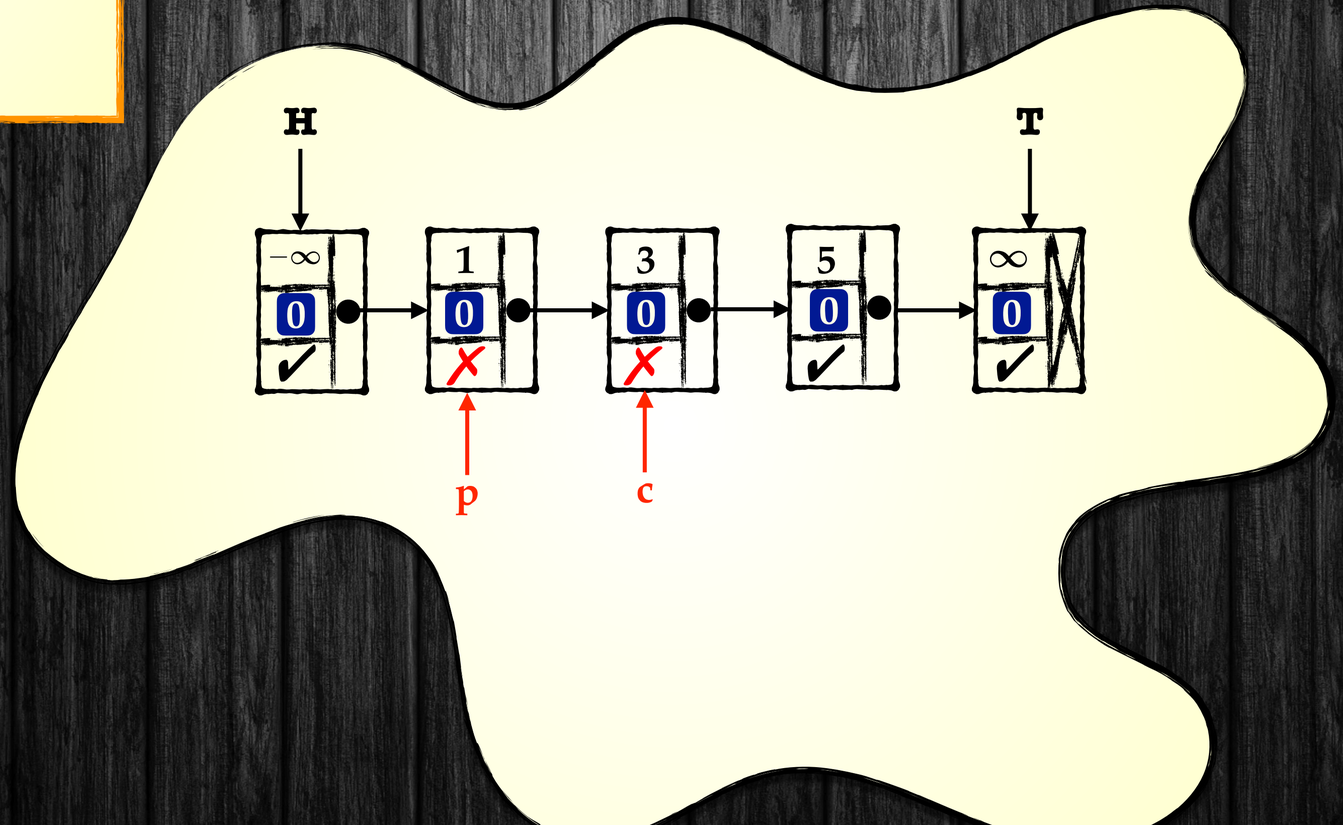
Sets | Lazy Synchronization

```
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





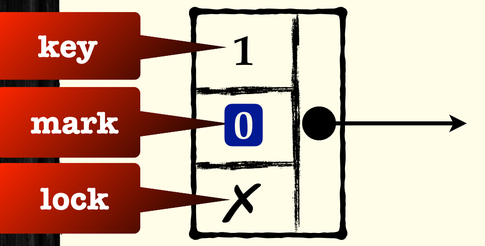
Sets | Lazy Synchronization

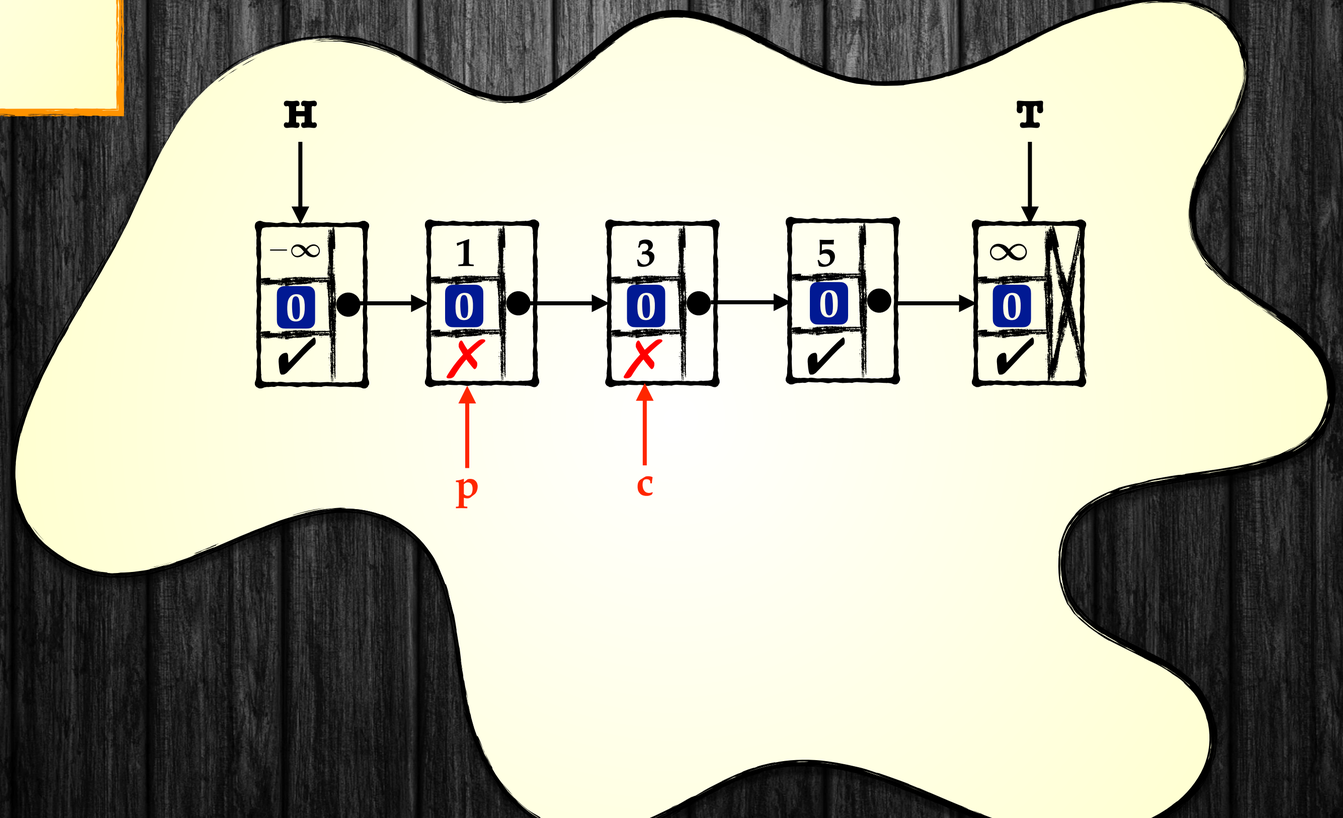
```
add(e):
   local p, c, n, r
1  (p,c) := locate(e);
2  if (c.val <> e)
3     n := new Node(0,e,c,false);
4     p.next := n;
5     r := true;
6  else r := false;
7  unlock(p);
8  unlock(c);
9  return r;
```

add(3)

unsuccessful add

linearization point





Sets | Lazy Synchronization

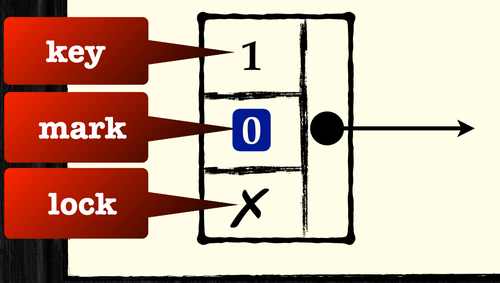
```
add(e):
   local p, c, n, r
1  (p,c) := locate(e);
2  if (c.val <> e)
3     n := new Node(0,e,c,false)
4     p.next := n;
5     r := true;
6  else r := false;
7  unlock(p);
8  unlock(c);
9  return r;
```

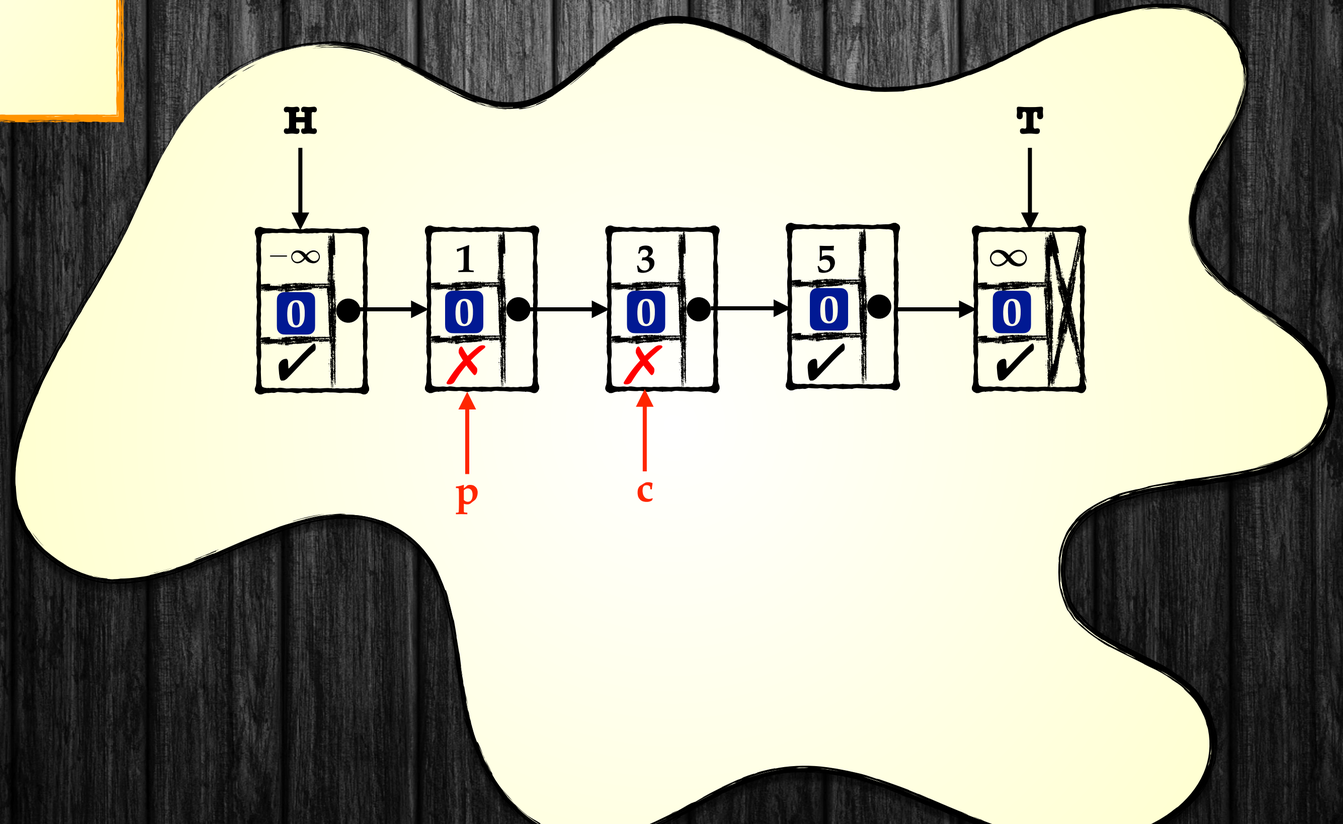
"when unmarked entry is observed"

unsuccessful add

linearization point







Sets | Lazy Synchronization

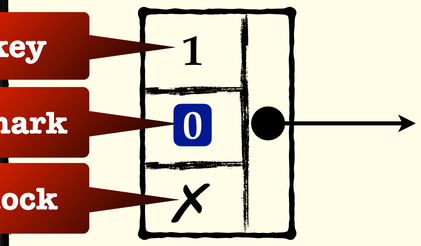
```
add(e):
   local p, c, n, r
1  (p,c) := locate(e);
2  if (c.val <> e)
3     n := new Node(0,e,c,false)
4     p.next := n;
5     r := true;
6  else r := false;
7  unlock(p);
8  unlock(c);
9  return r;
```

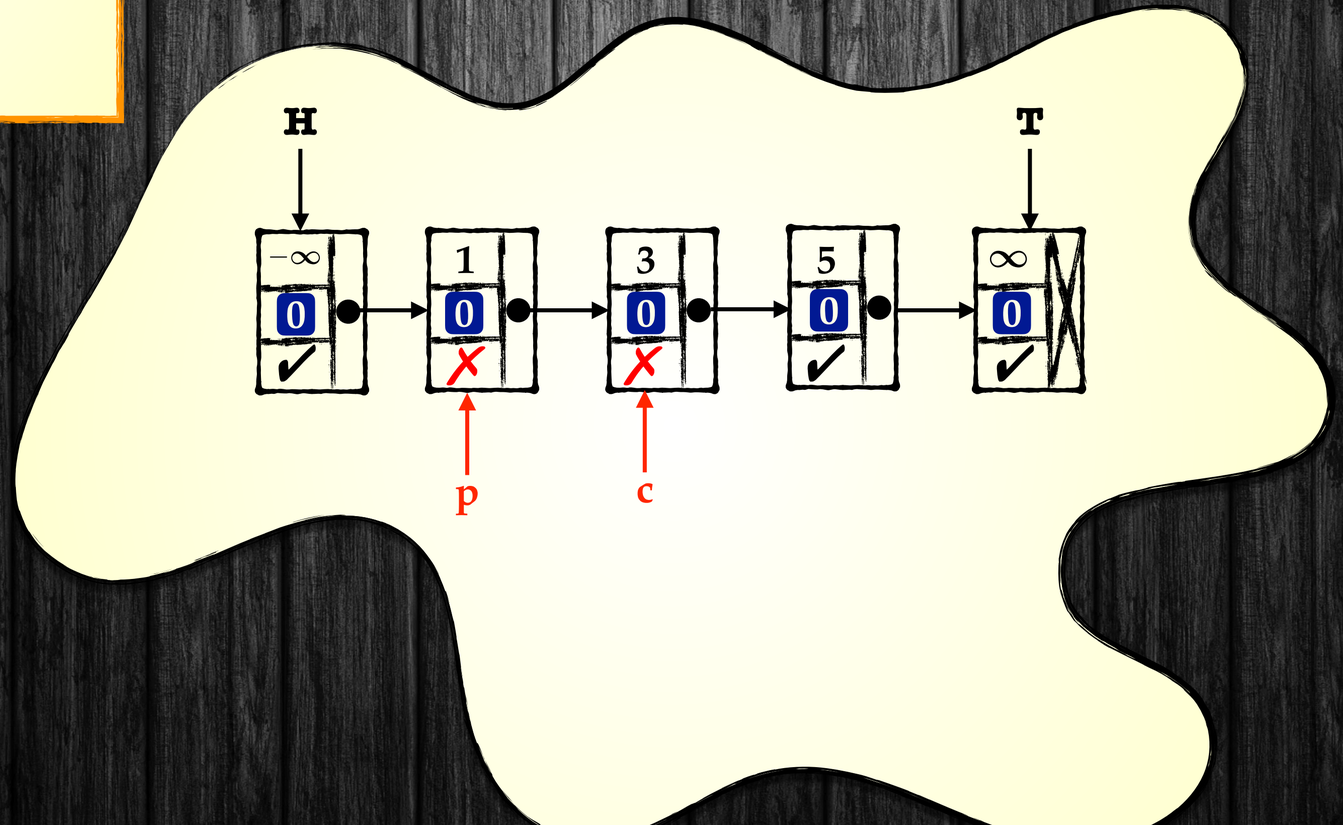
"when unmarked entry is observed"

unsuccessful add

linearization point

add(3)

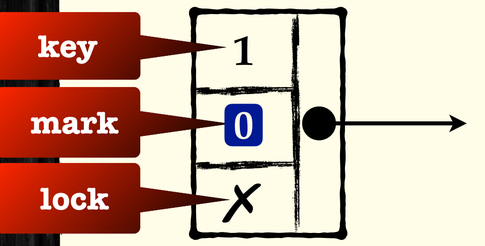


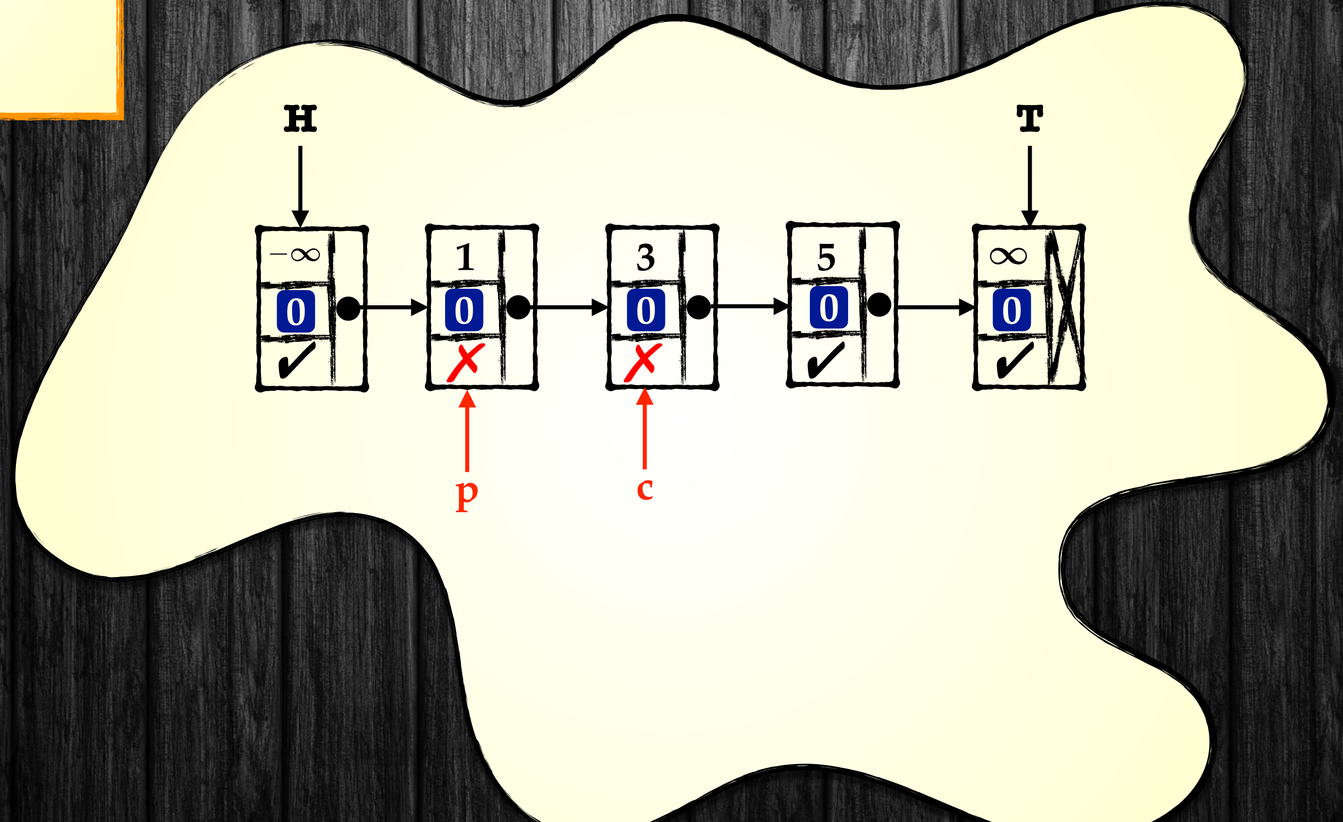


Sets | Lazy Synchronization

```
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)

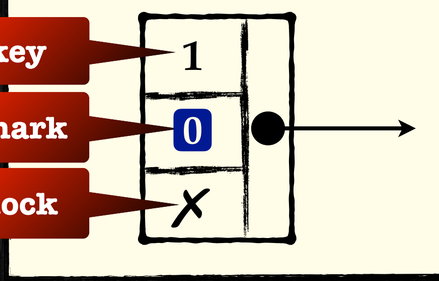


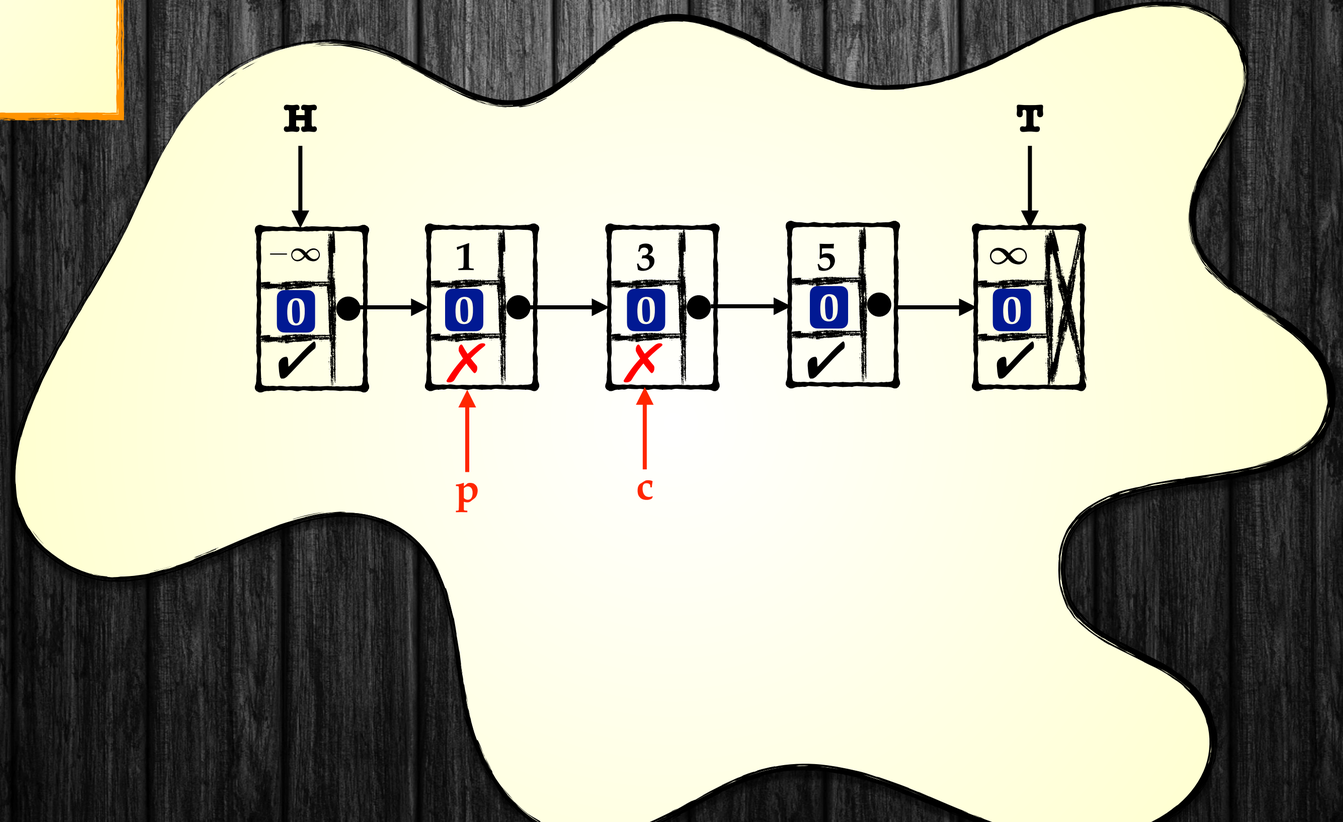


Sets | Lazy Synchronization

```
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)

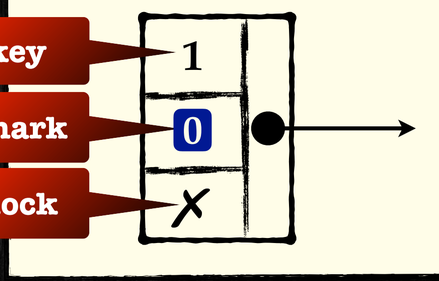


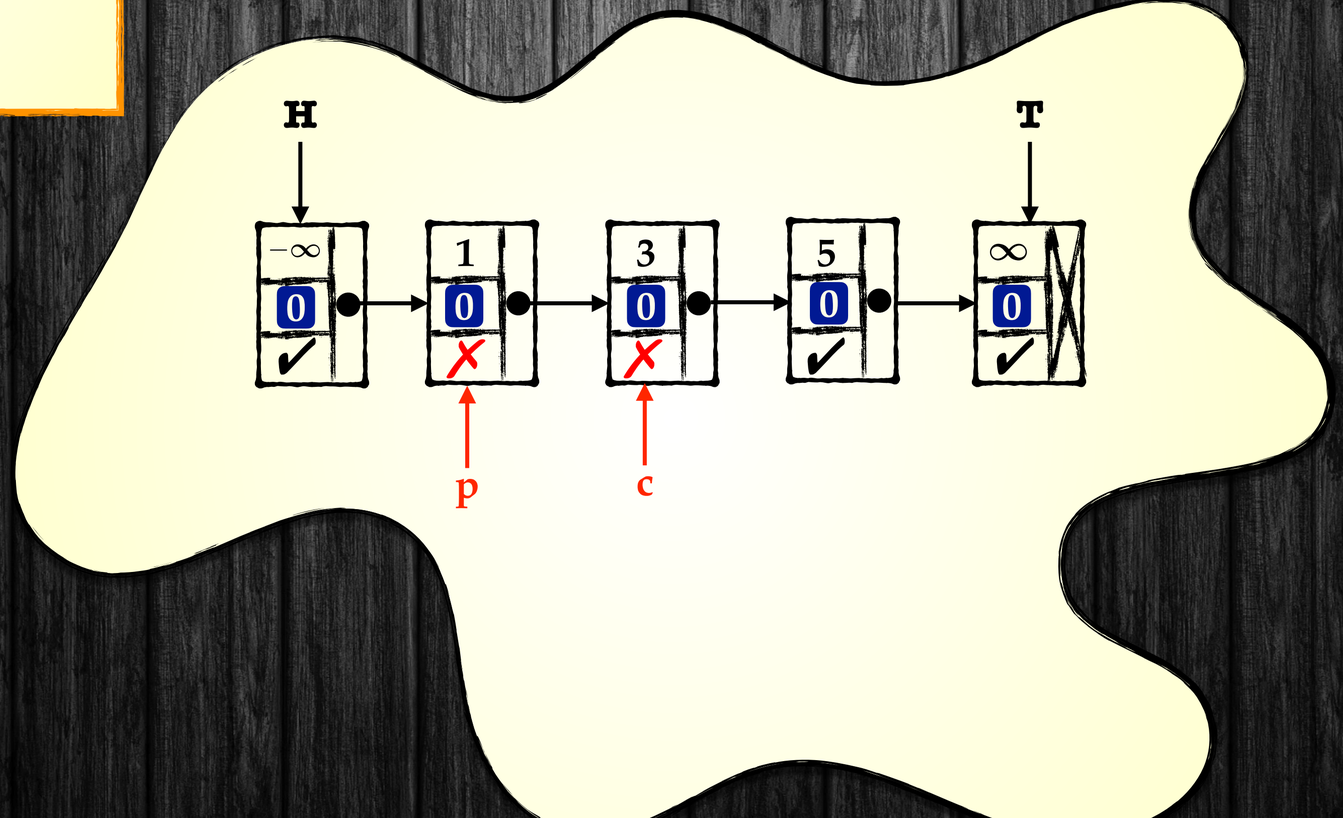


Sets | Lazy Synchronization

```
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)

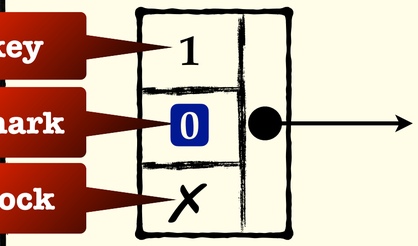


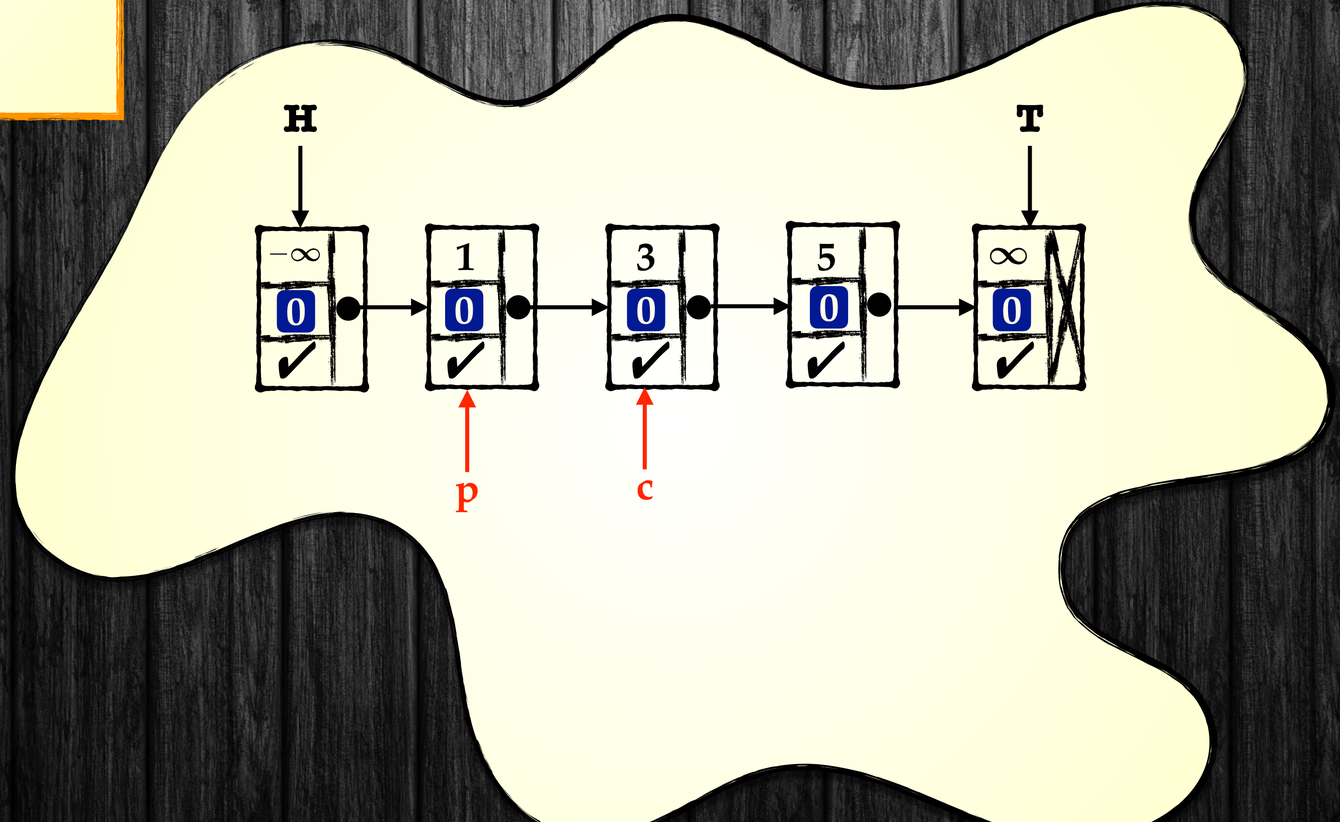


Sets | Lazy Synchronization

```
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)

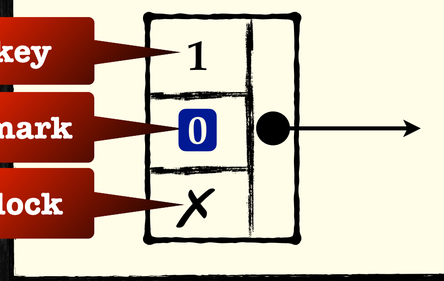


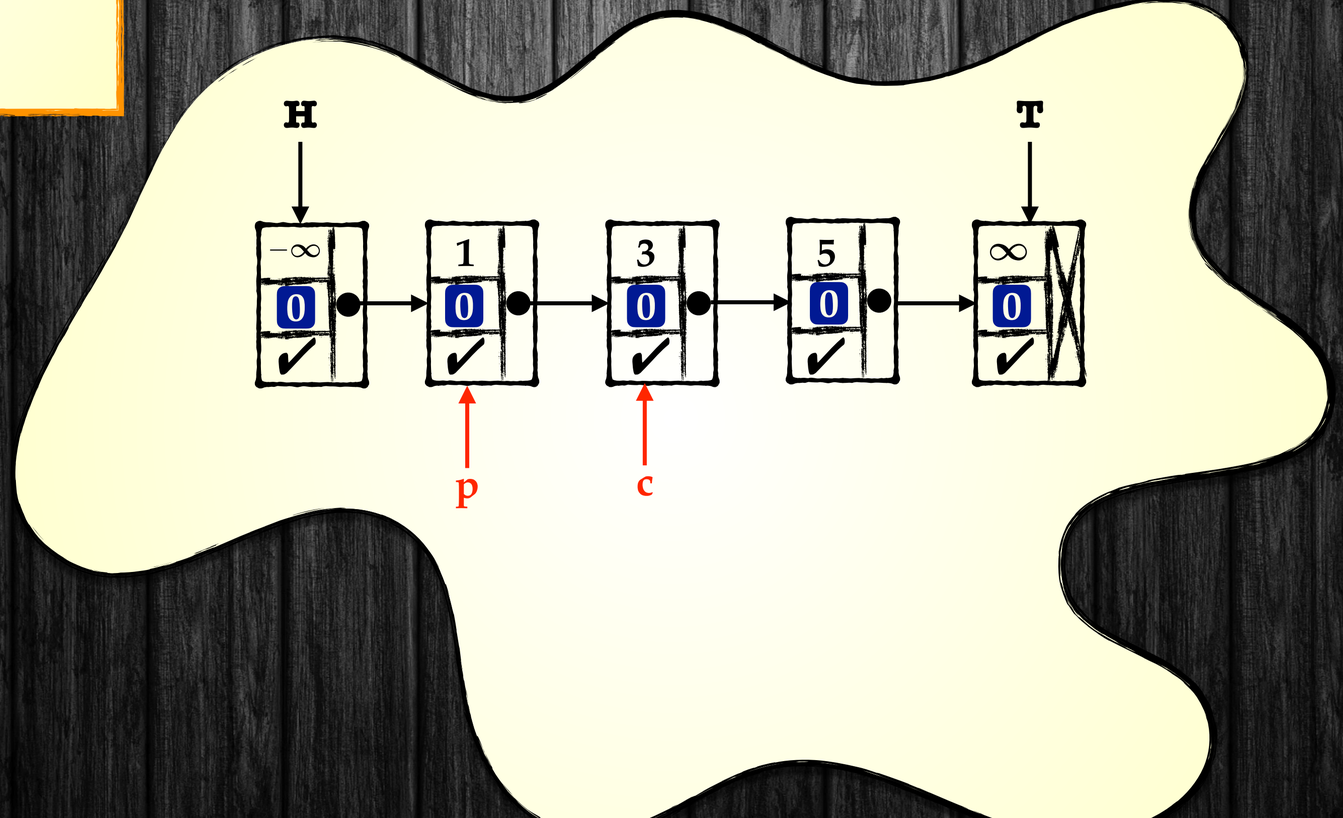


Sets | Lazy Synchronization

```
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)

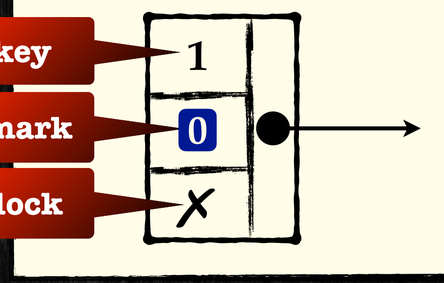


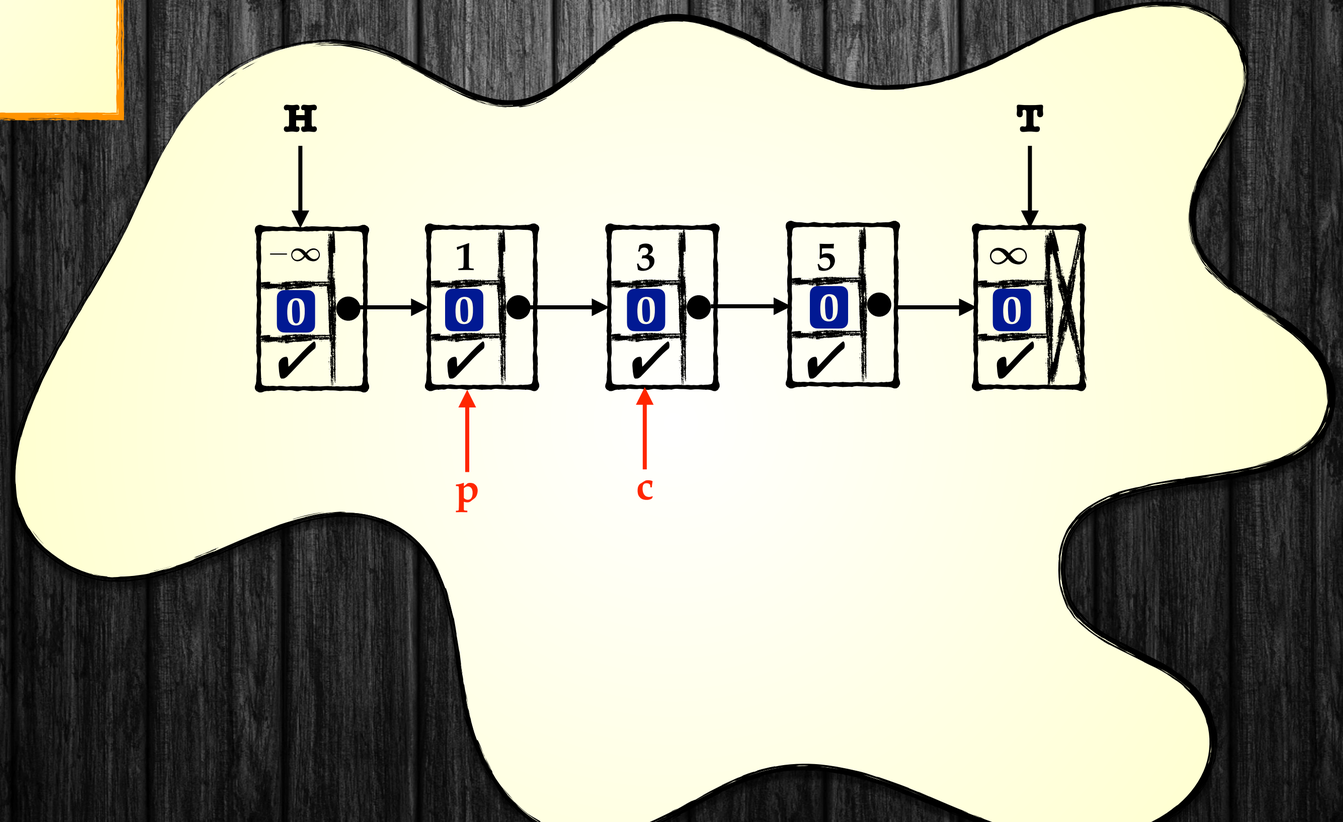


Sets | Lazy Synchronization

```
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;
```

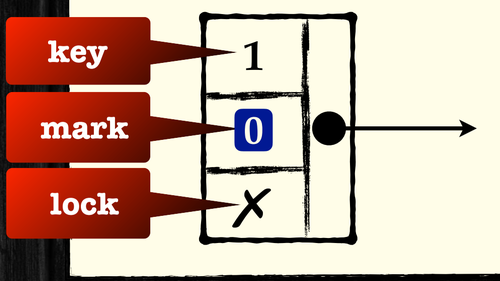
add(3)

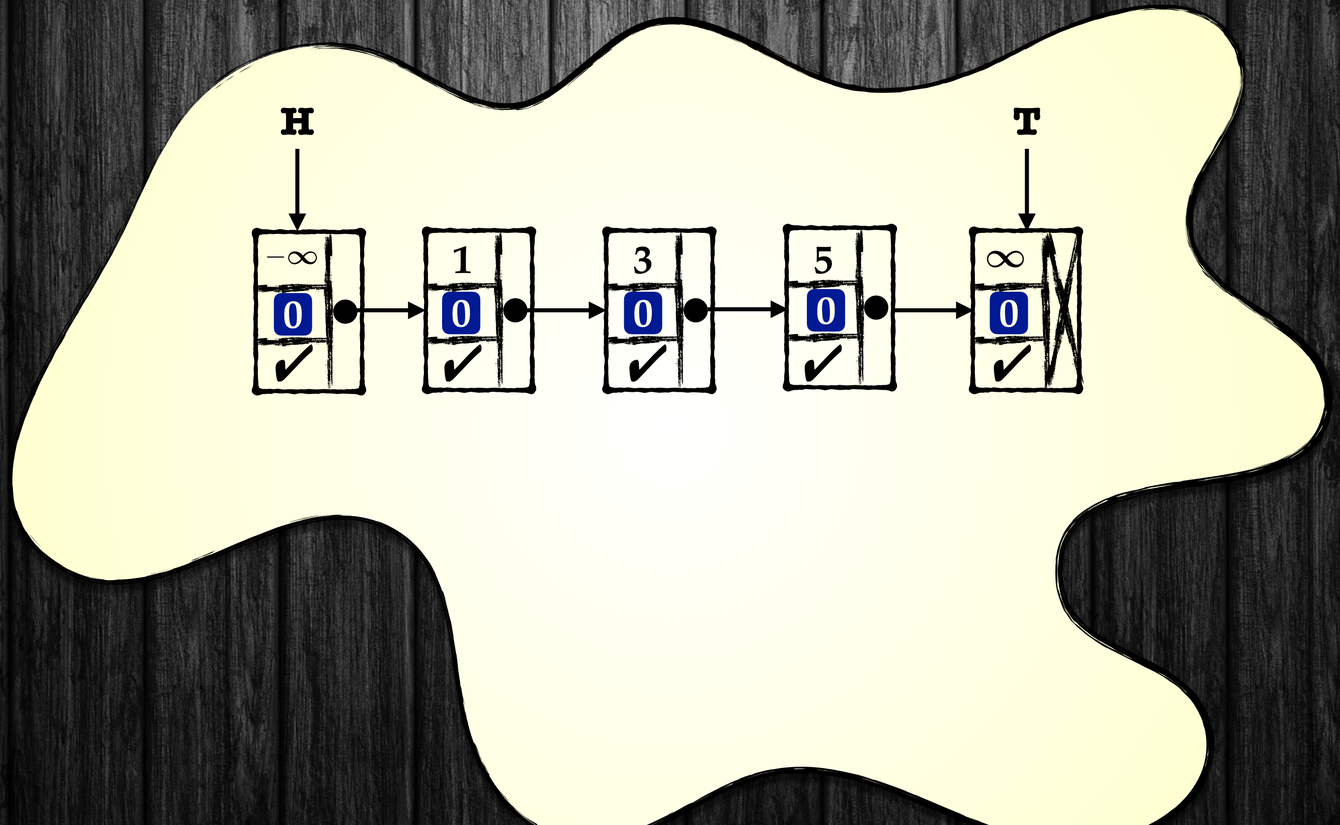




```
rmv(e):
local p, c, n, r
1  (p,c) := locate(e);
2  if (c.val = e)
3     c.mark := true
4     n := c.next;
5     p.next := n;
6     r := true;
7  else r := false;
8  unlock(p);
9  unlock(c);
10 return r;
```

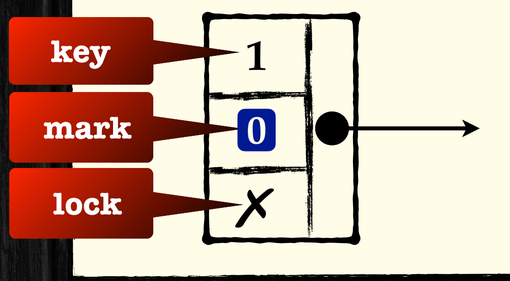


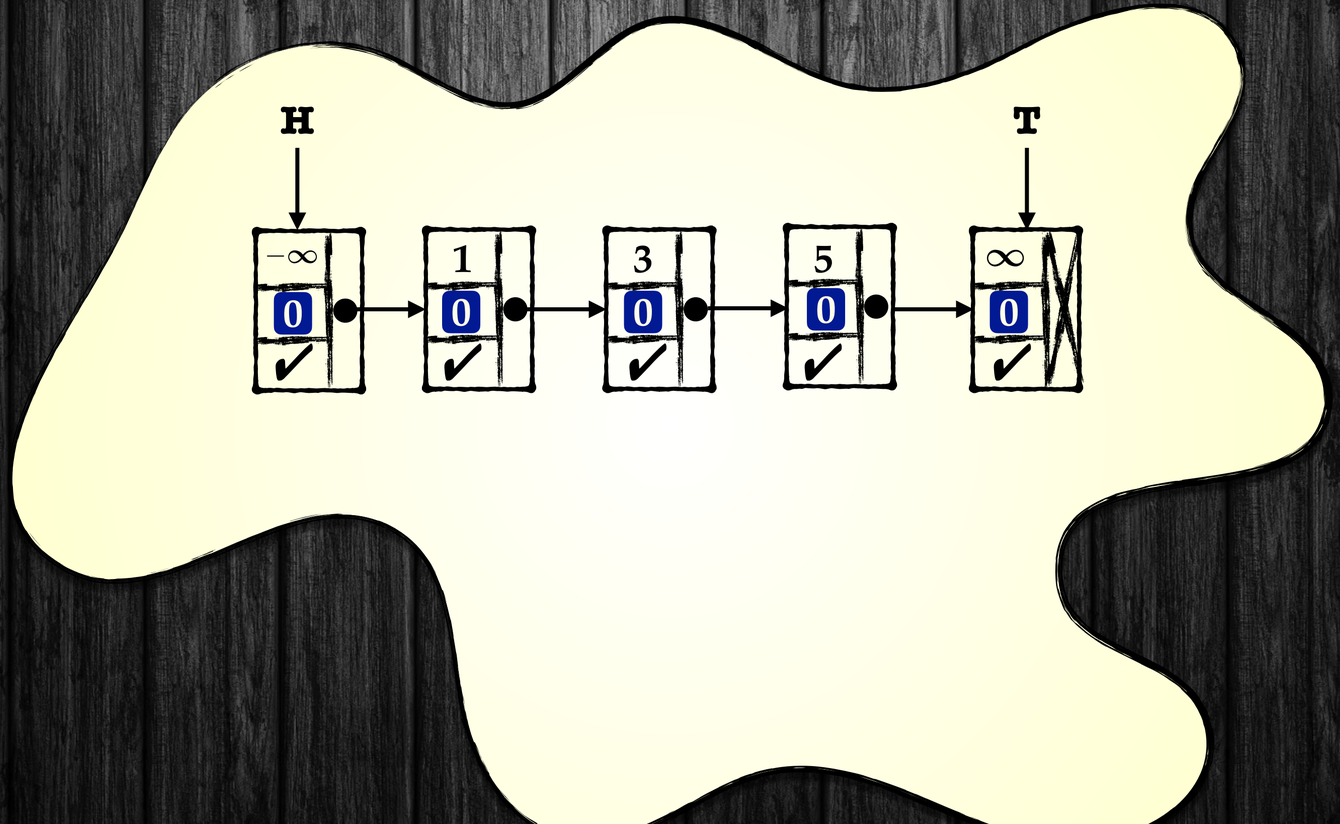




```
rmv(e):
local p, c, n, r
1  (p,c) := locate(e);
2  if (c.val = e)
3     c.mark := true
4     n := c.next;
5     p.next := n;
6     r := true;
7  else r := false;
8  unlock(p);
9  unlock(c);
10 return r;
```

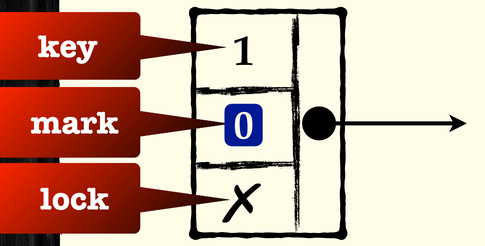


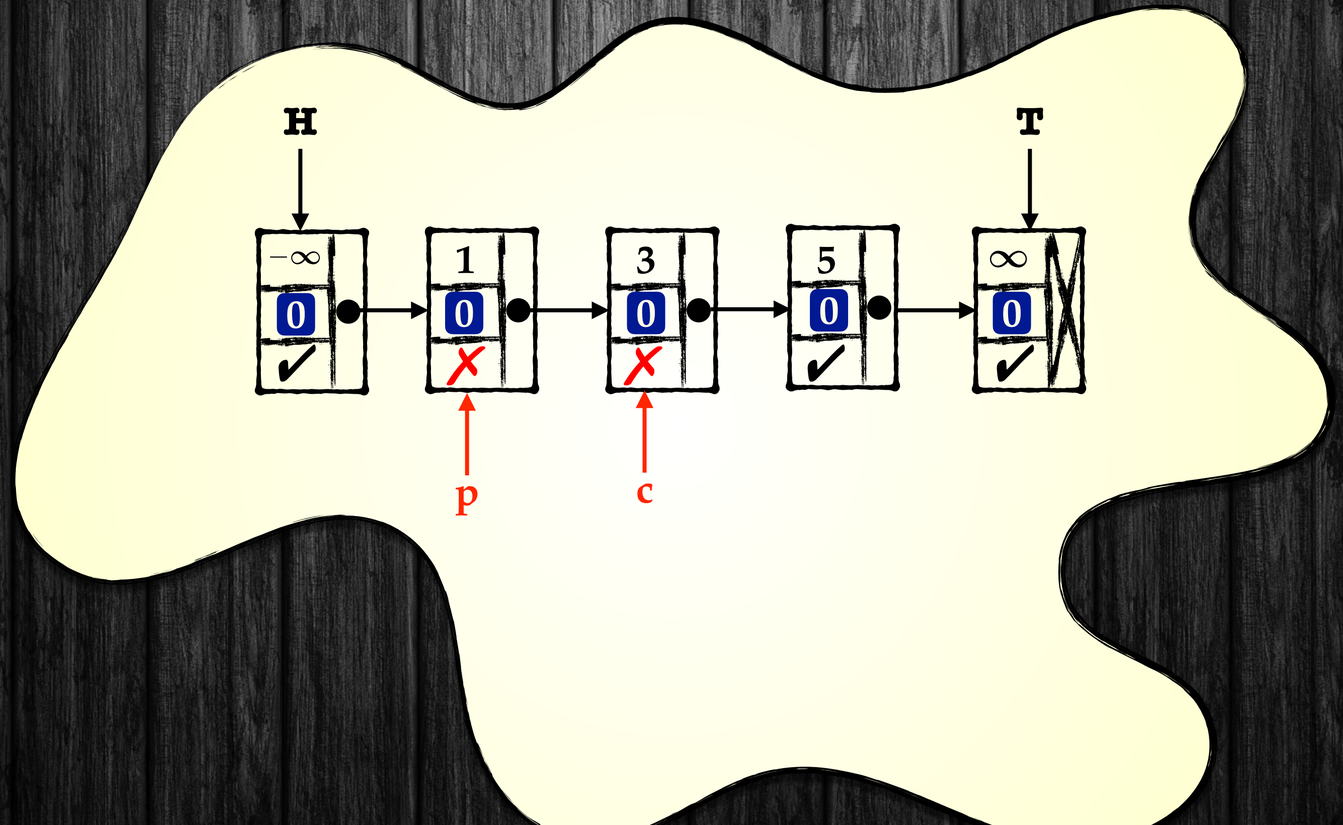




```
rmv(e):
local p, c, n, r
1  (p,c) := locate(e);
2  if (c.val = e)
3     c.mark := true
4     n := c.next;
5     p.next := n;
6     r := true;
7  else r := false;
8  unlock(p);
9  unlock(c);
10 return r;
```

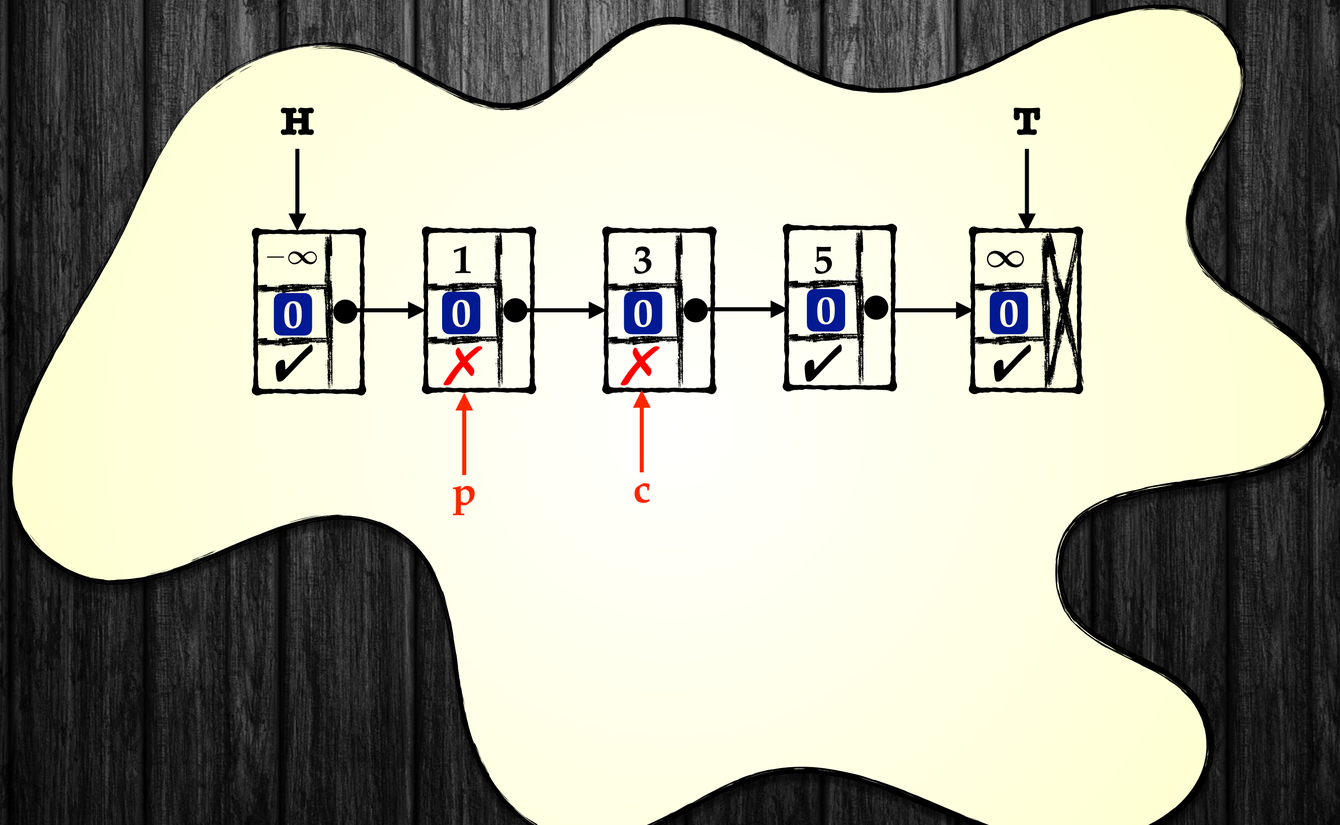
rmv(3)

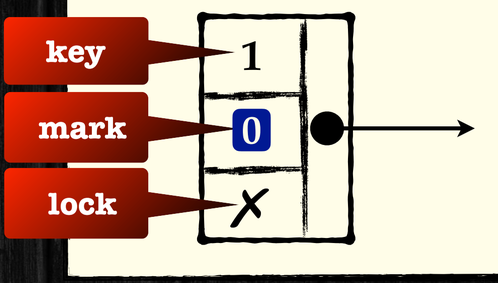




```
rmv(e):
local p, c, n, r
1  (p,c) := locate(e);
2  if (c.val = e)
3     c.mark := true
4     n := c.next;
5     p.next := n;
6     r := true;
7  else r := false;
8  unlock(p);
9  unlock(c);
10 return r;
```

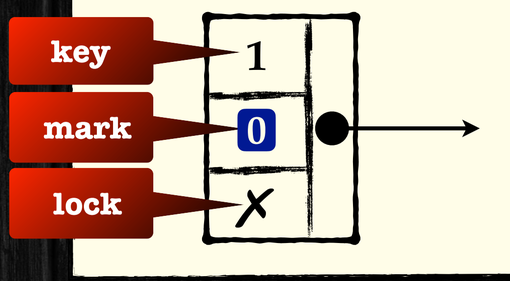
rmv(3)

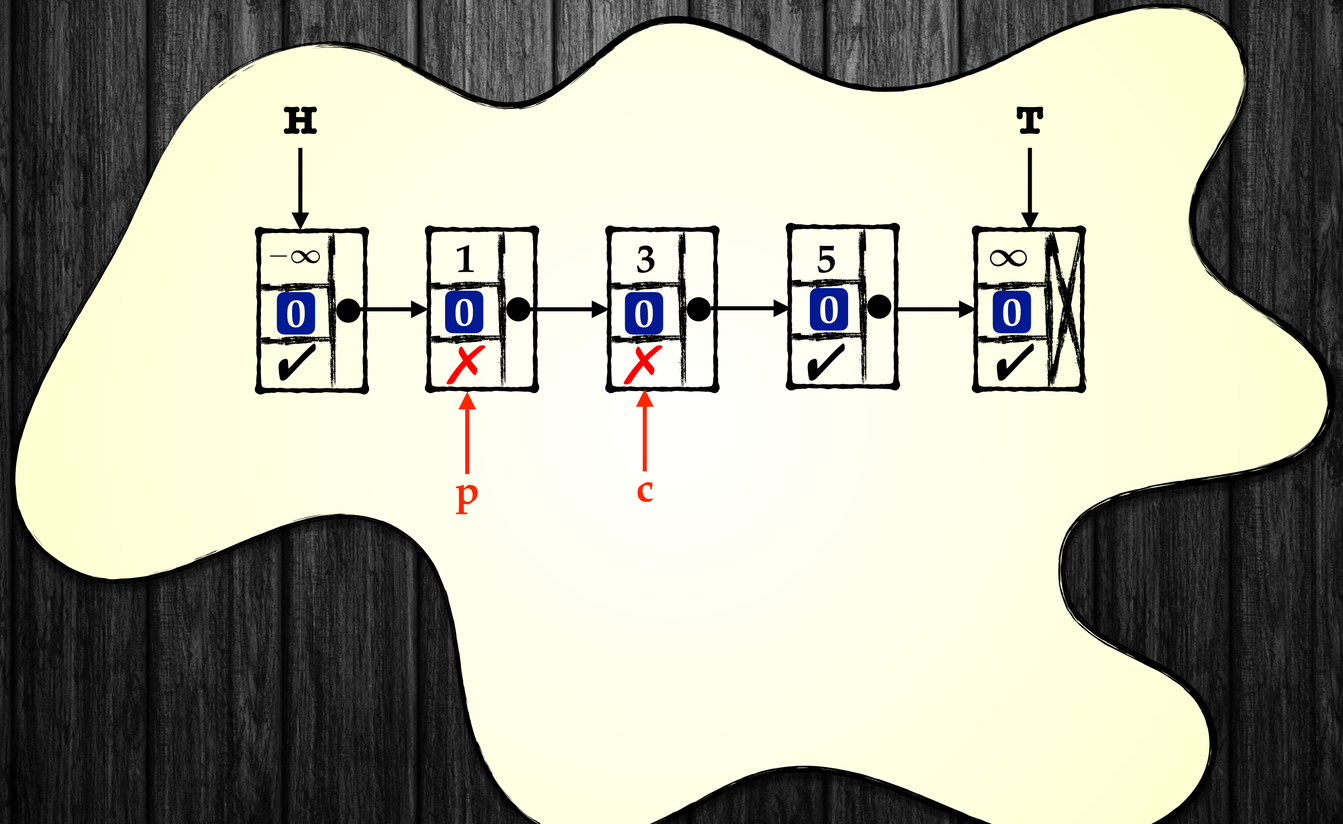




```
rmv(e):
local p, c, n, r
1  (p,c) := locate(e);
2  if (c.val = e)
3     c.mark := true
4     n := c.next;
5     p.next := n;
6     r := true;
7  else r := false;
8  unlock(p);
9  unlock(c);
10 return r;
```



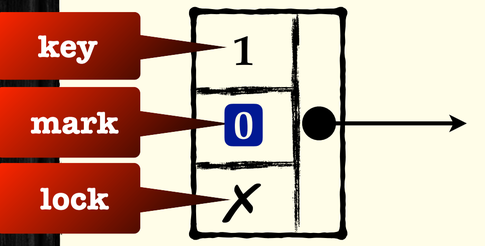


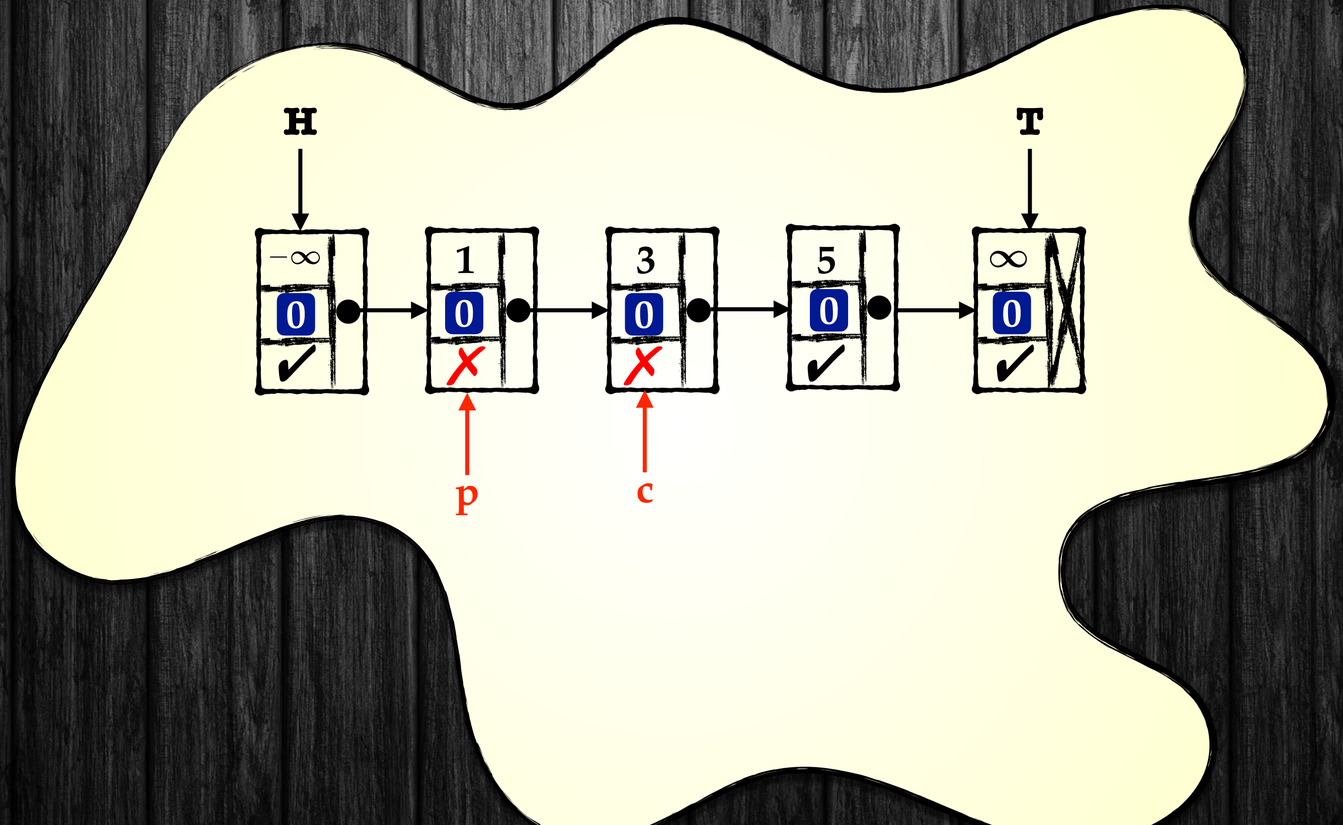


```
rmv(e):
local p, c, n, r
1  (p,c) := locate(e);
2  if (c.val = e)
3     c.mark := true
4     n := c.next;
5     p.next := n;
6     r := true;
7  else r := false;
8  unlock(p);
9  unlock(c);
10 return r;
```

rmv(3)

logical deletion

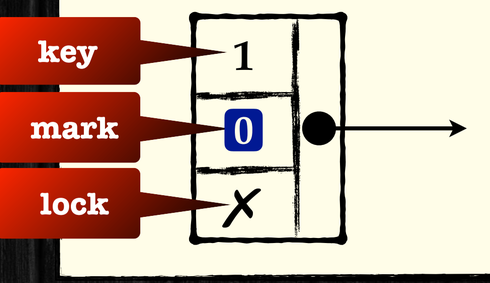


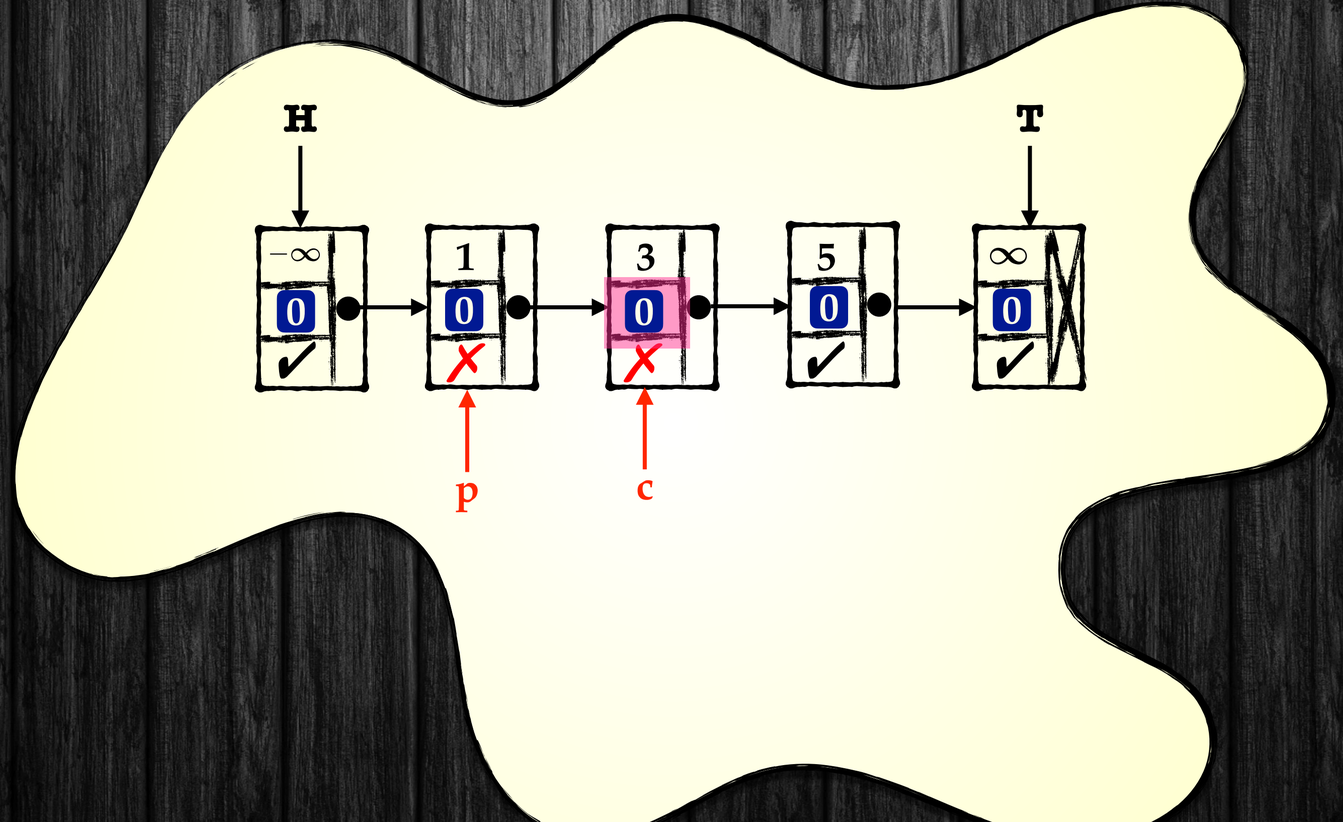


```
rmv(e):
local p, c, n, r
1  (p,c) := locate(e);
2  if (c.val = e)
3     c.mark := true
4     n := c.next;
5     p.next := n;
6     r := true;
7  else r := false;
8  unlock(p);
9  unlock(c);
10 return r;
```

rmv(3)

logical deletion

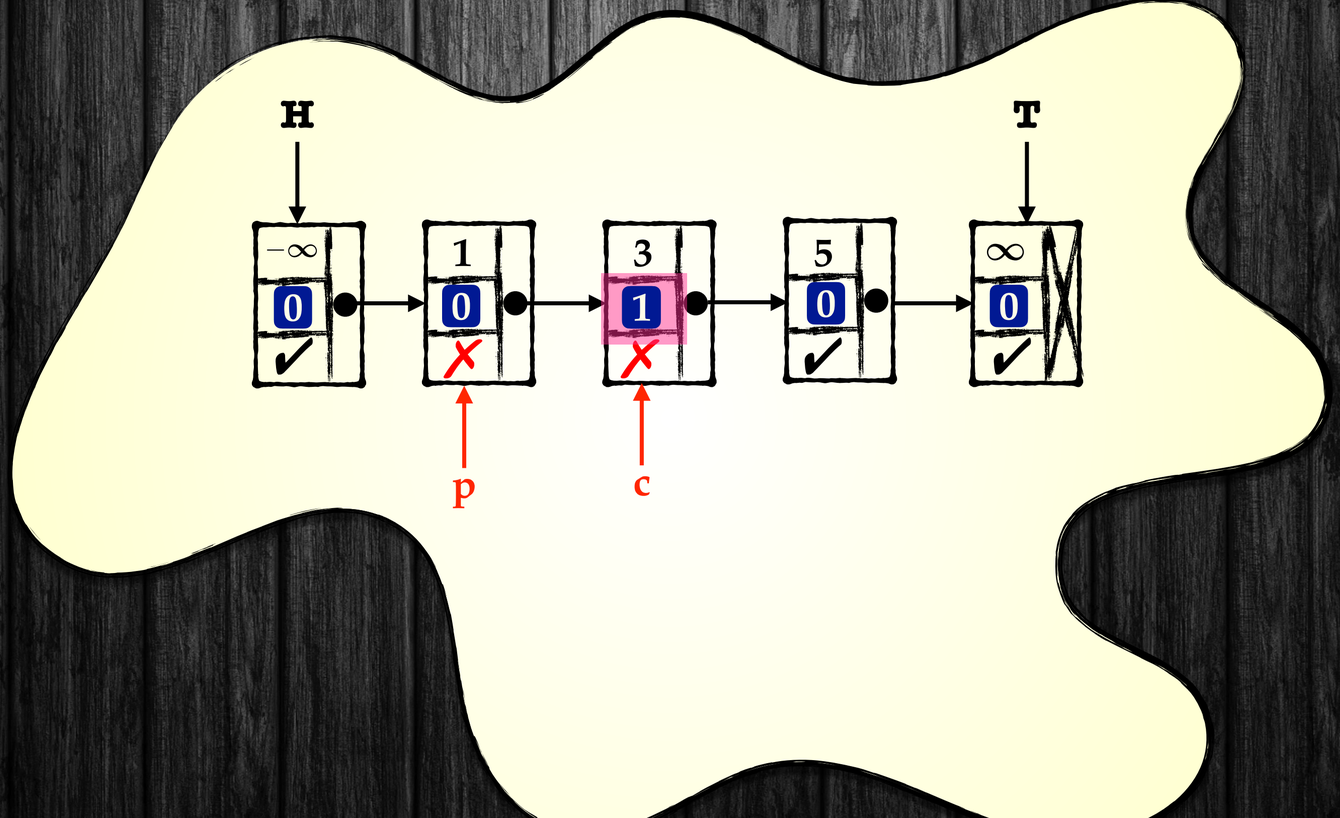


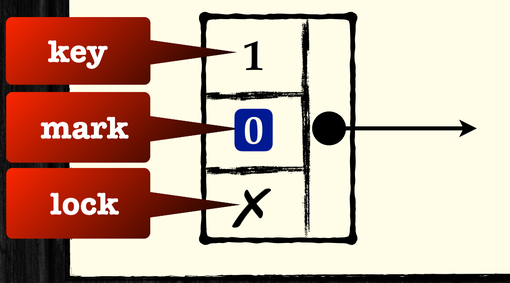


```
rmv(e):
local p, c, n, r
1  (p,c) := locate(e);
2  if (c.val = e)
3     c.mark := true
4     n := c.next;
5     p.next := n;
6     r := true;
7  else r := false;
8  unlock(p);
9  unlock(c);
10 return r;
```

rmv(3)

logical deletion





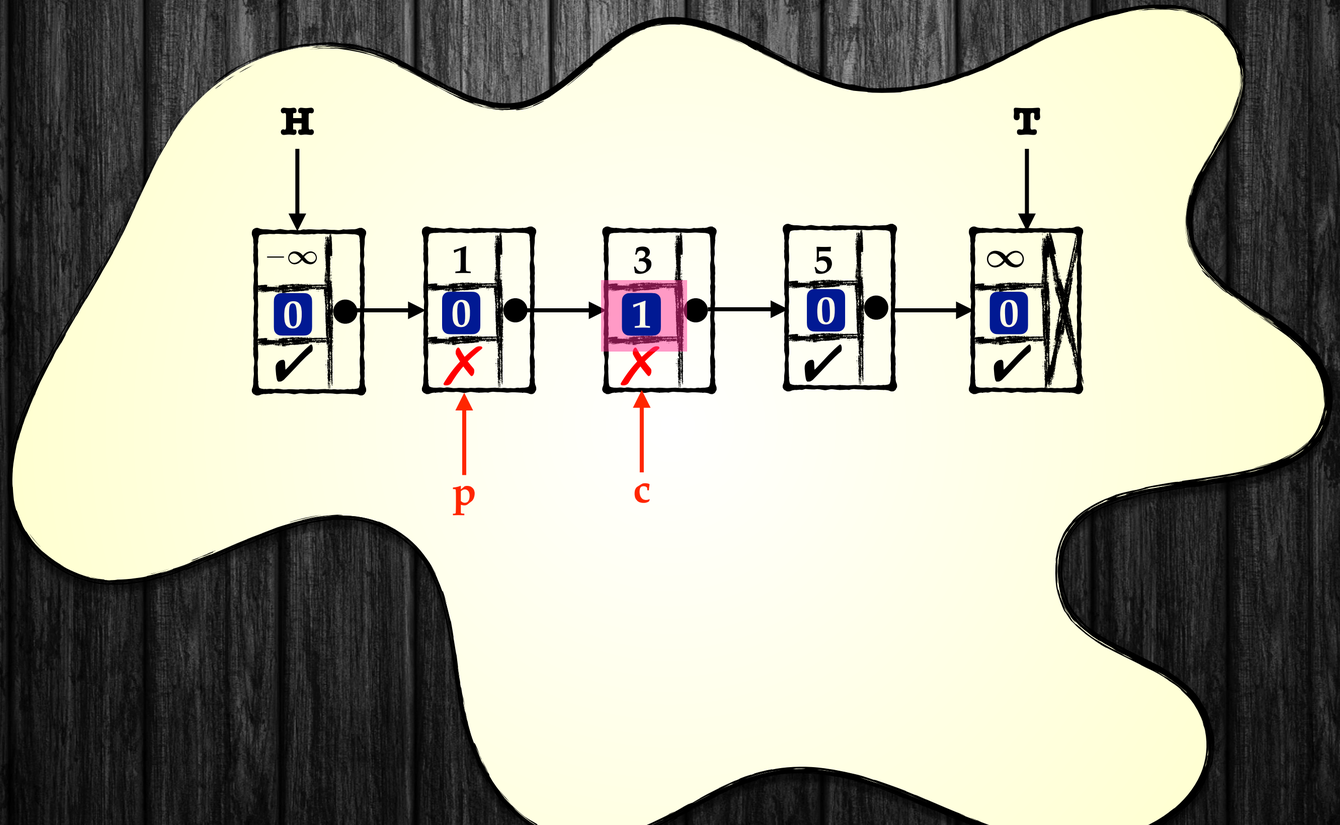
Sets | Lazy Synchronization

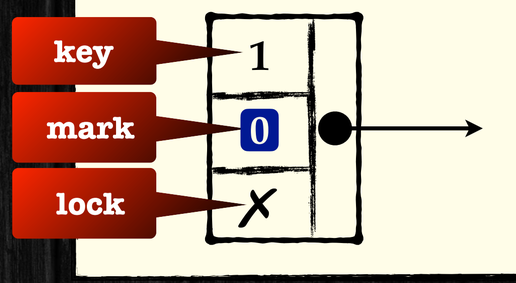
```
rmv(e):
local p, c, n, r
1  (p,c) := locate(e);
2  if (c.val = e)
3     c.mark := true
4     n := c.next;
5     p.next := n;
6     r := true;
7  else r := false;
8  unlock(p);
9  unlock(c);
10 return r;
```

rmv(3)

logical deletion

linearization point





Sets | Lazy Synchronization

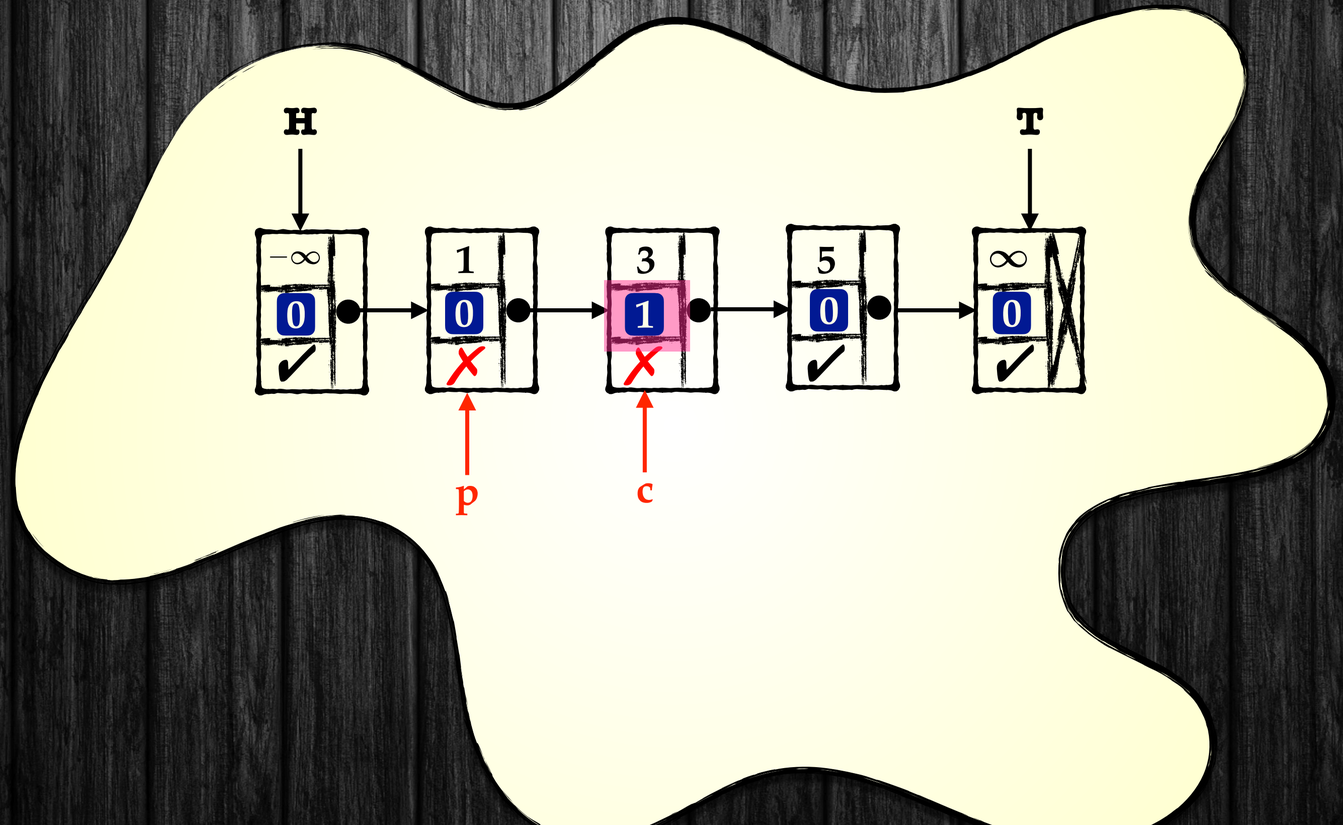
```
rmv(e):
local p, c, n, r
1  (p,c) := locate(e);
2  if (c.val = e)
3     c.mark := true
4     n := c.next;
5     p.next := n;
6     r := true;
7  else r := false;
8  unlock(p);
9  unlock(c);
10 return r;
```

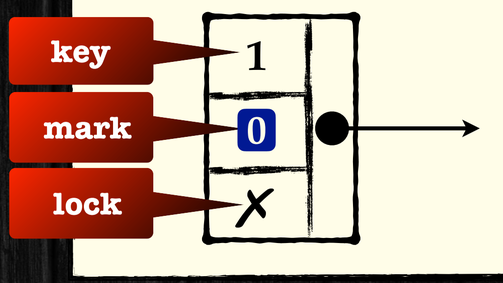
rmv(3)

logical deletion

successful remove

linearization point





Sets | Lazy Synchronization

```
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;
```

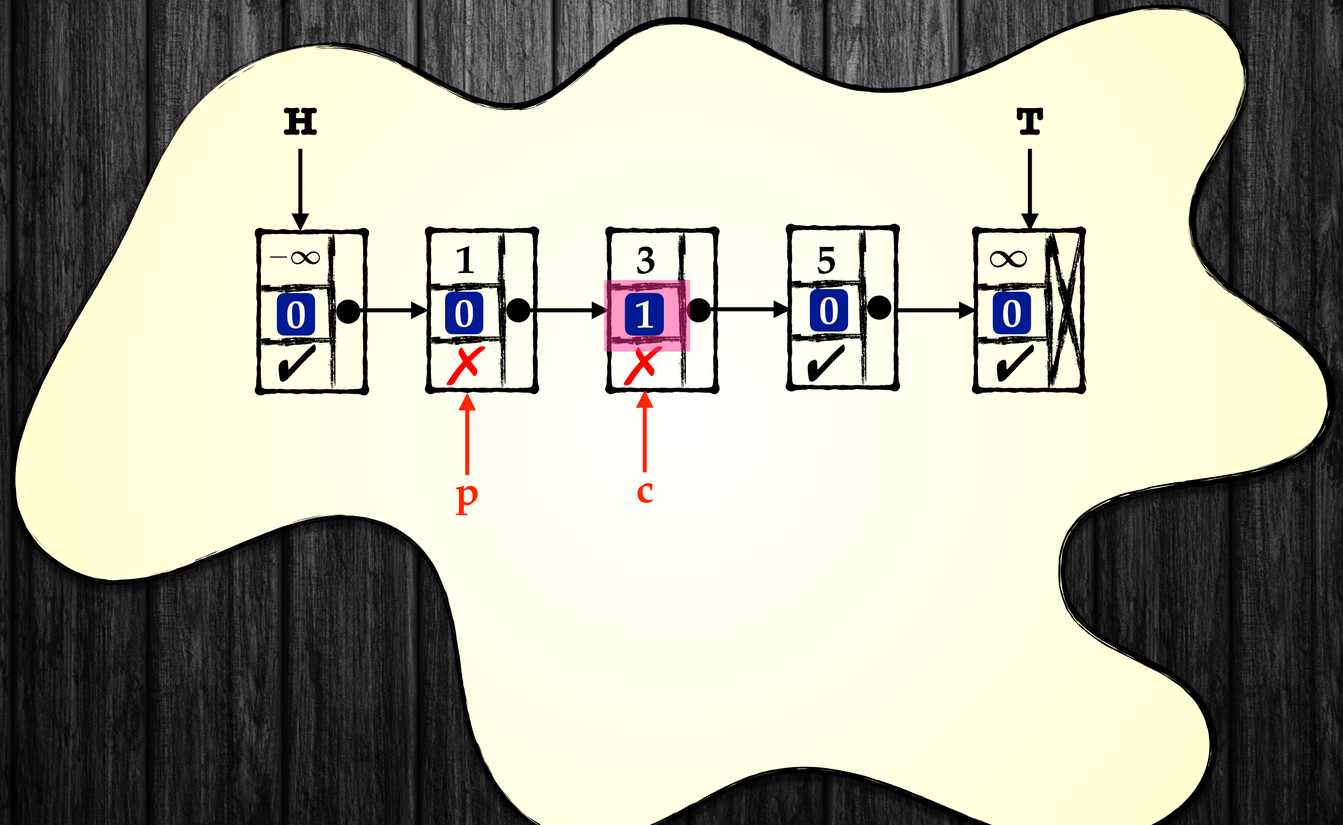
rmv(3)

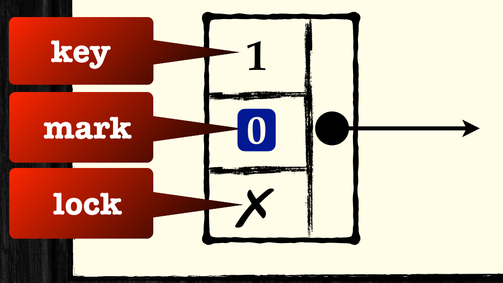
logical deletion

"when the cell is marked"

successful remove

linearization point





Sets | Lazy Synchronization

```
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;
```

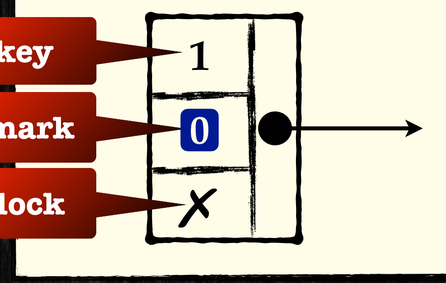
rmv(3)

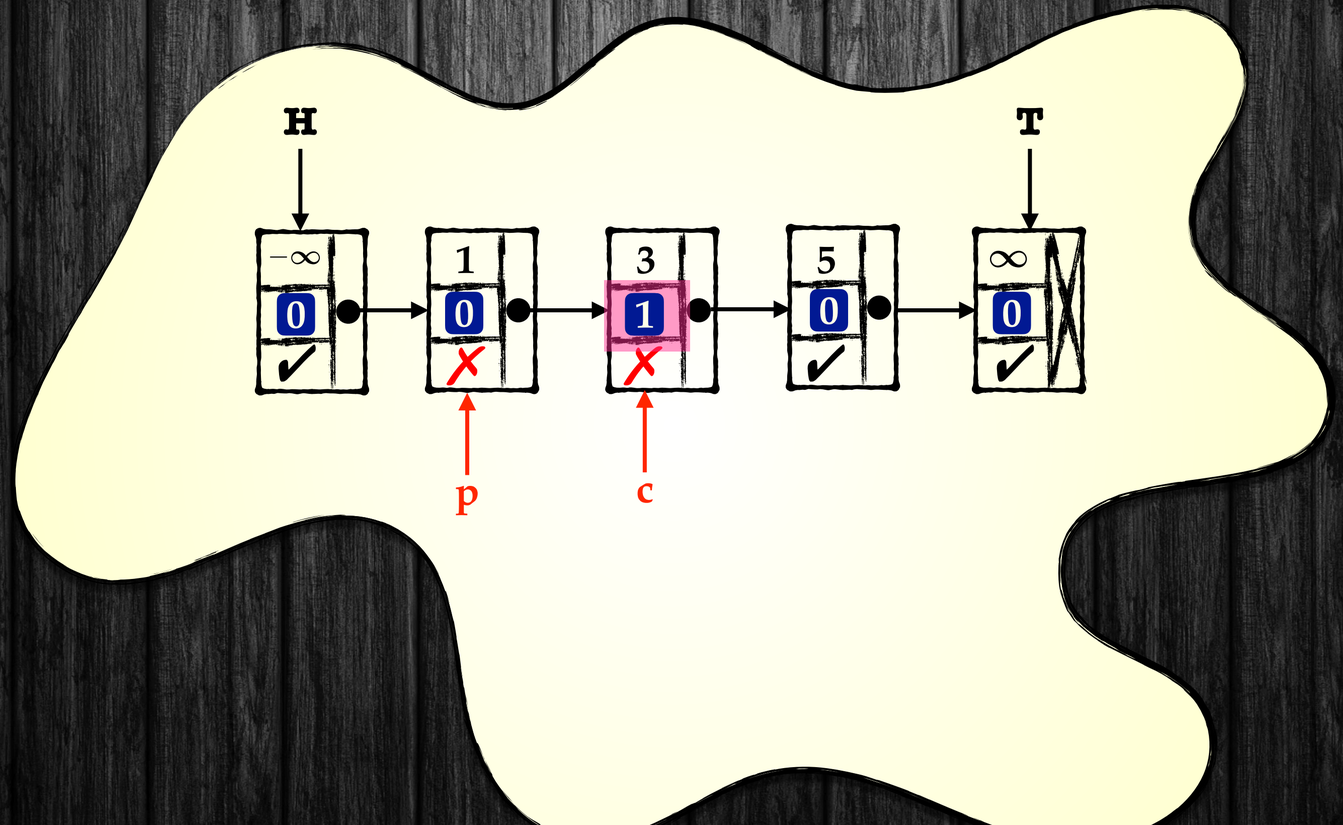
logical deletion

"when the cell is marked"

successful remove

linearization point





Sets | Lazy Synchronization

```
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 logically deleted"

"when the cell is marked"

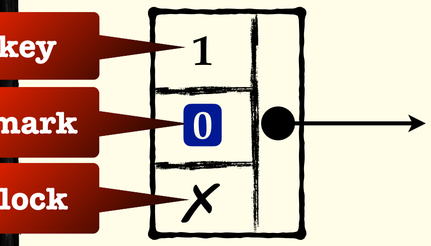
successful remove

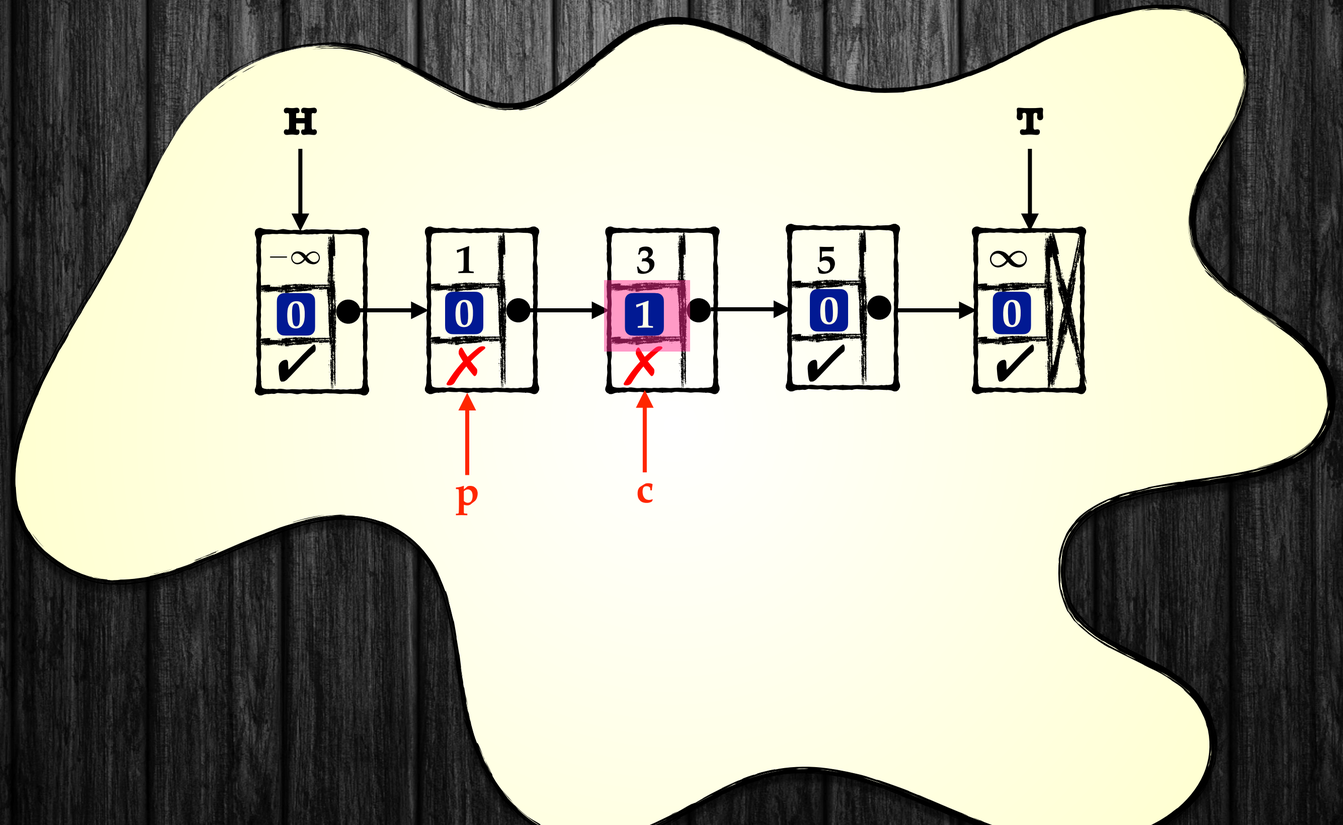
rmv(3)

logical deletion

linearization point

key
mark
lock

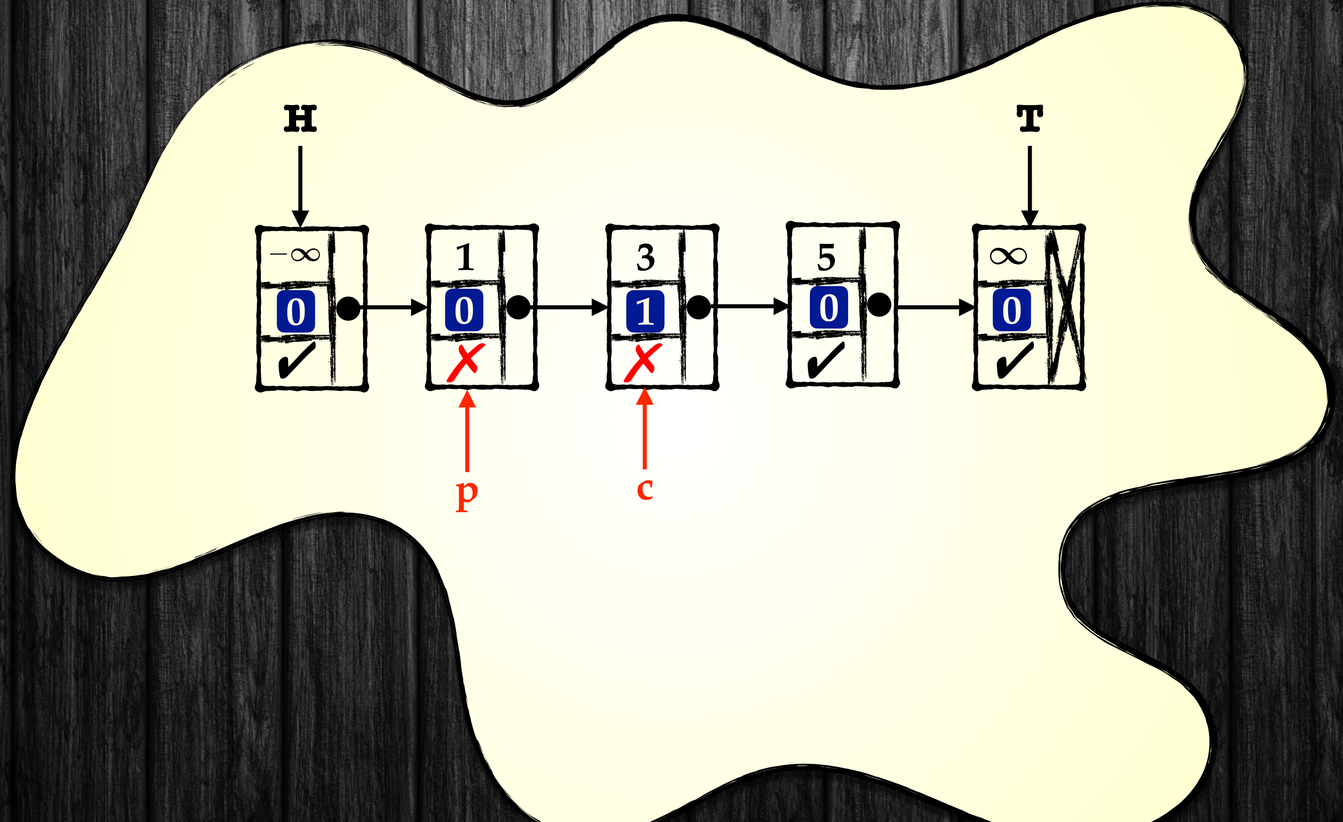


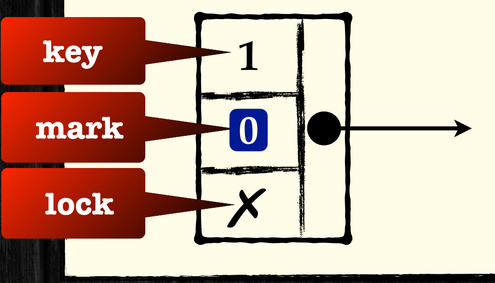


Sets | Lazy Synchronization

```
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;
```

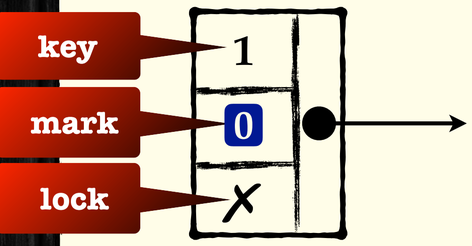
rmv(3)

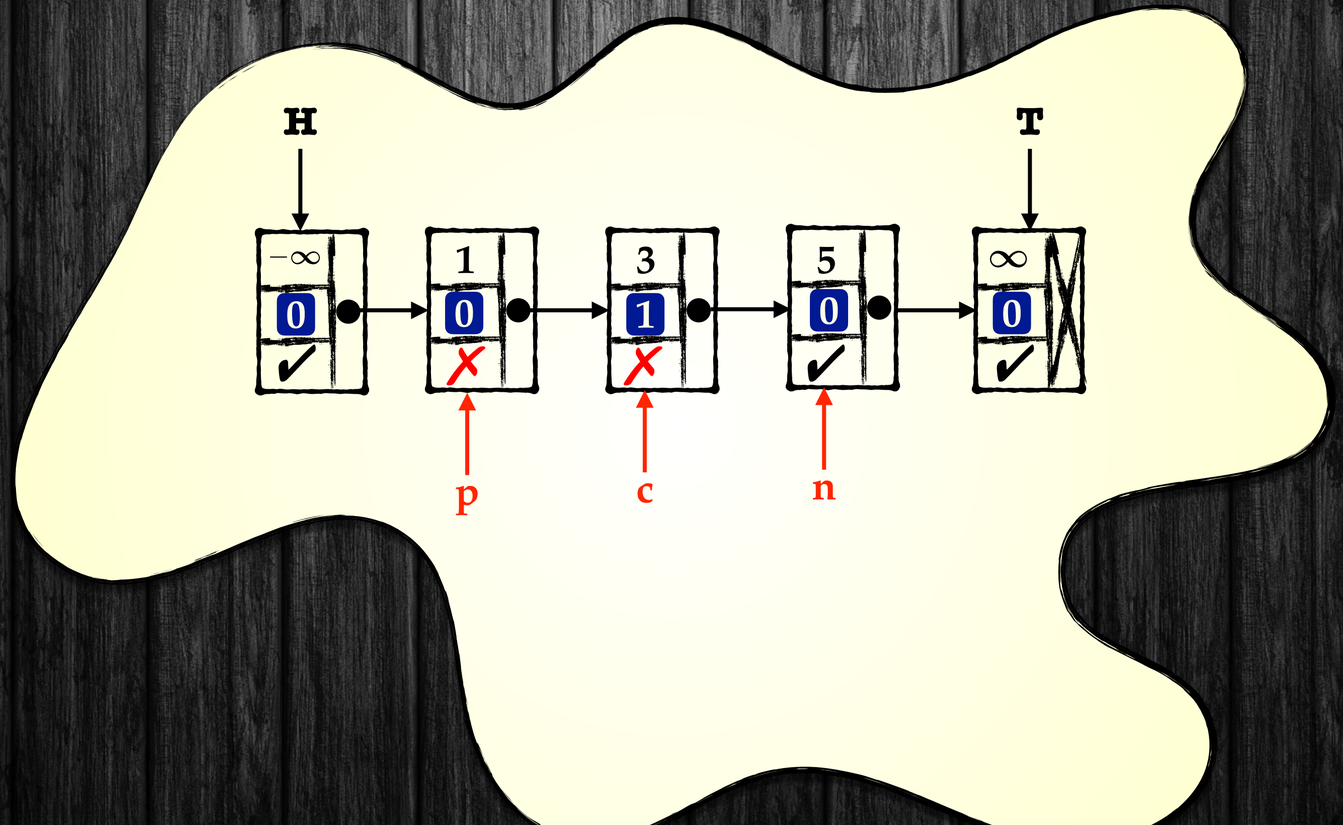




```
rmv(e):
local p, c, n, r
1  (p,c) := locate(e);
2  if (c.val = e)
3     c.mark := true
4     n := c.next;
5     p.next := n;
6     r := true;
7  else r := false;
8  unlock(p);
9  unlock(c);
10 return r;
```

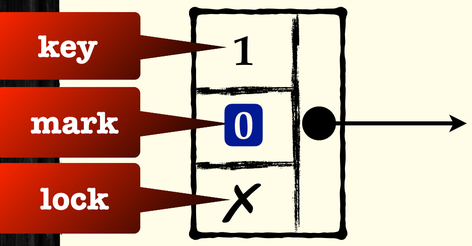
rmv(3)

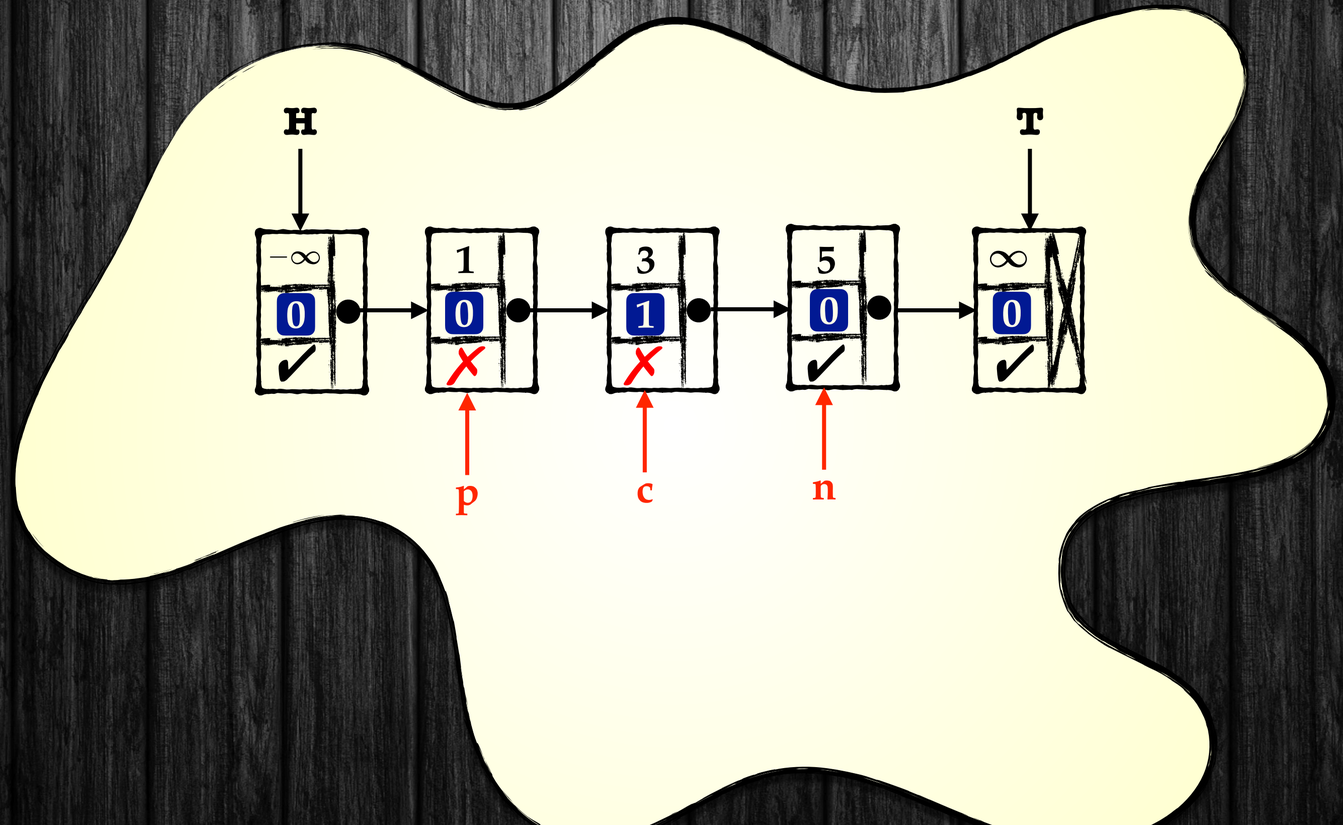




```
rmv(e):
local p, c, n, r
1  (p,c) := locate(e);
2  if (c.val = e)
3     c.mark := true
4     n := c.next;
5     p.next := n;
6     r := true;
7  else r := false;
8  unlock(p);
9  unlock(c);
10 return r;
```

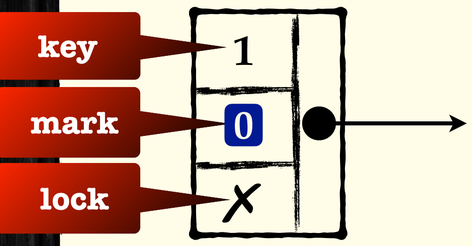
rmv(3)

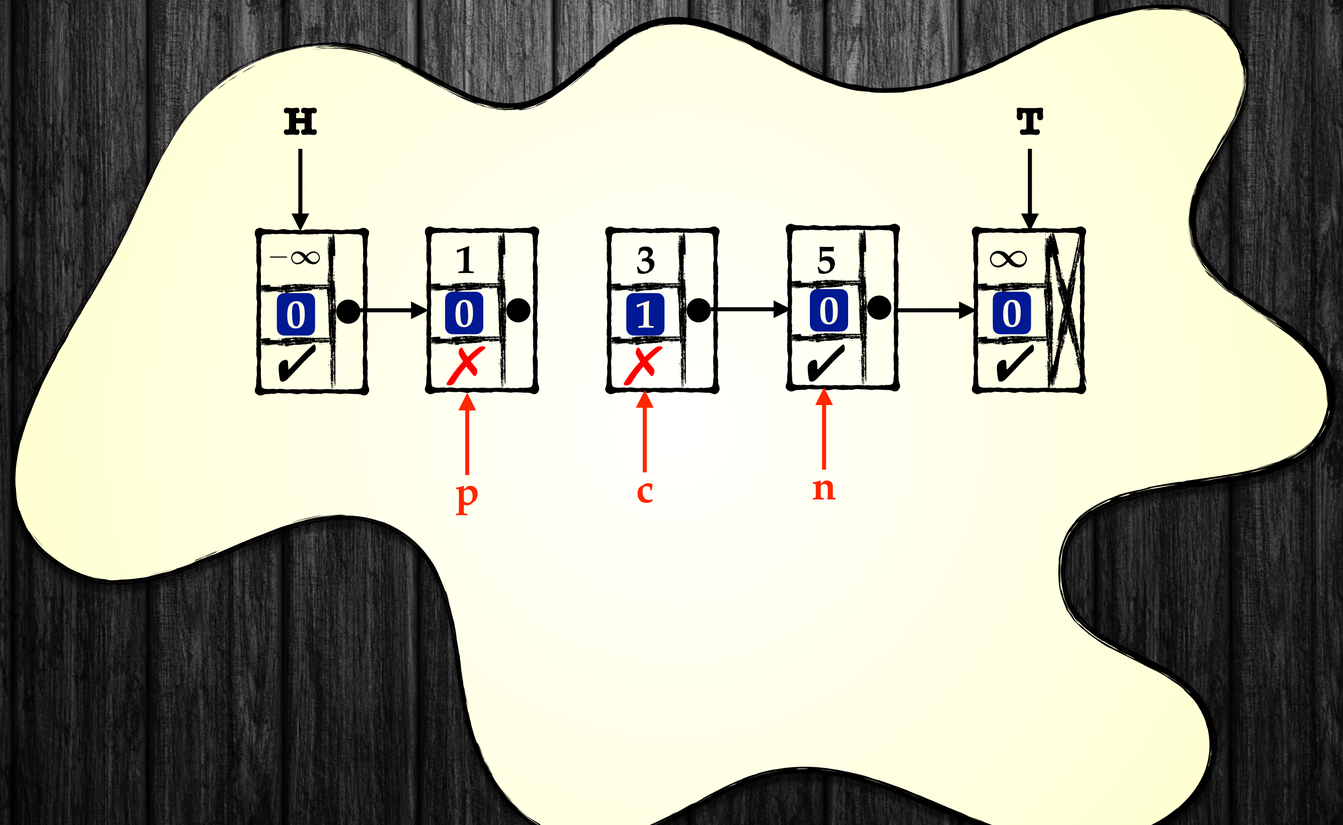




```
rmv(e):
local p, c, n, r
1  (p,c) := locate(e);
2  if (c.val = e)
3     c.mark := true
4     n := c.next;
5     p.next := n;
6     r := true;
7  else r := false;
8  unlock(p);
9  unlock(c);
10 return r;
```

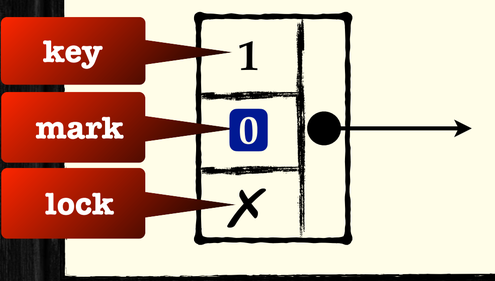
rmv(3)

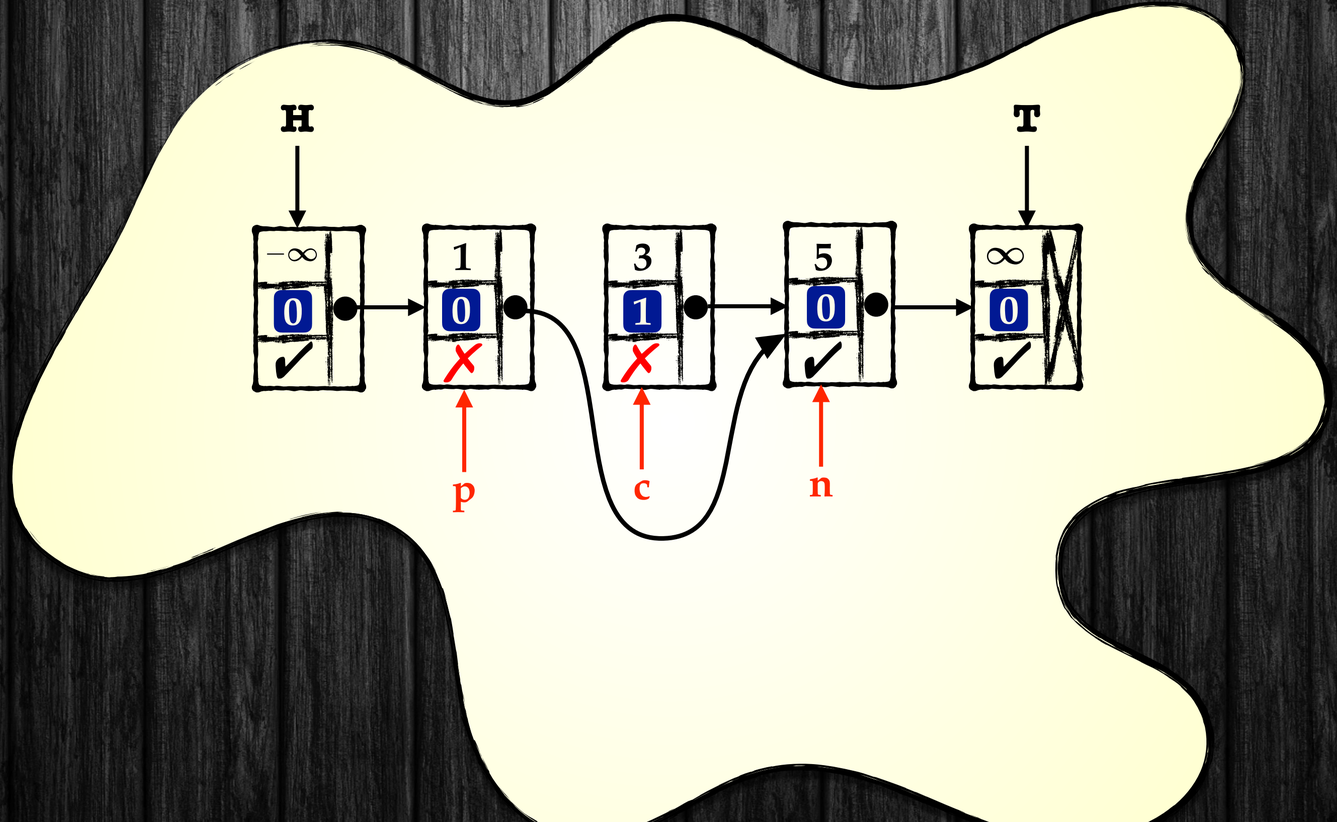




```
rmv(e):
local p, c, n, r
1  (p,c) := locate(e);
2  if (c.val = e)
3     c.mark := true
4     n := c.next;
5     p.next := n;
6     r := true;
7  else r := false;
8  unlock(p);
9  unlock(c);
10 return r;
```



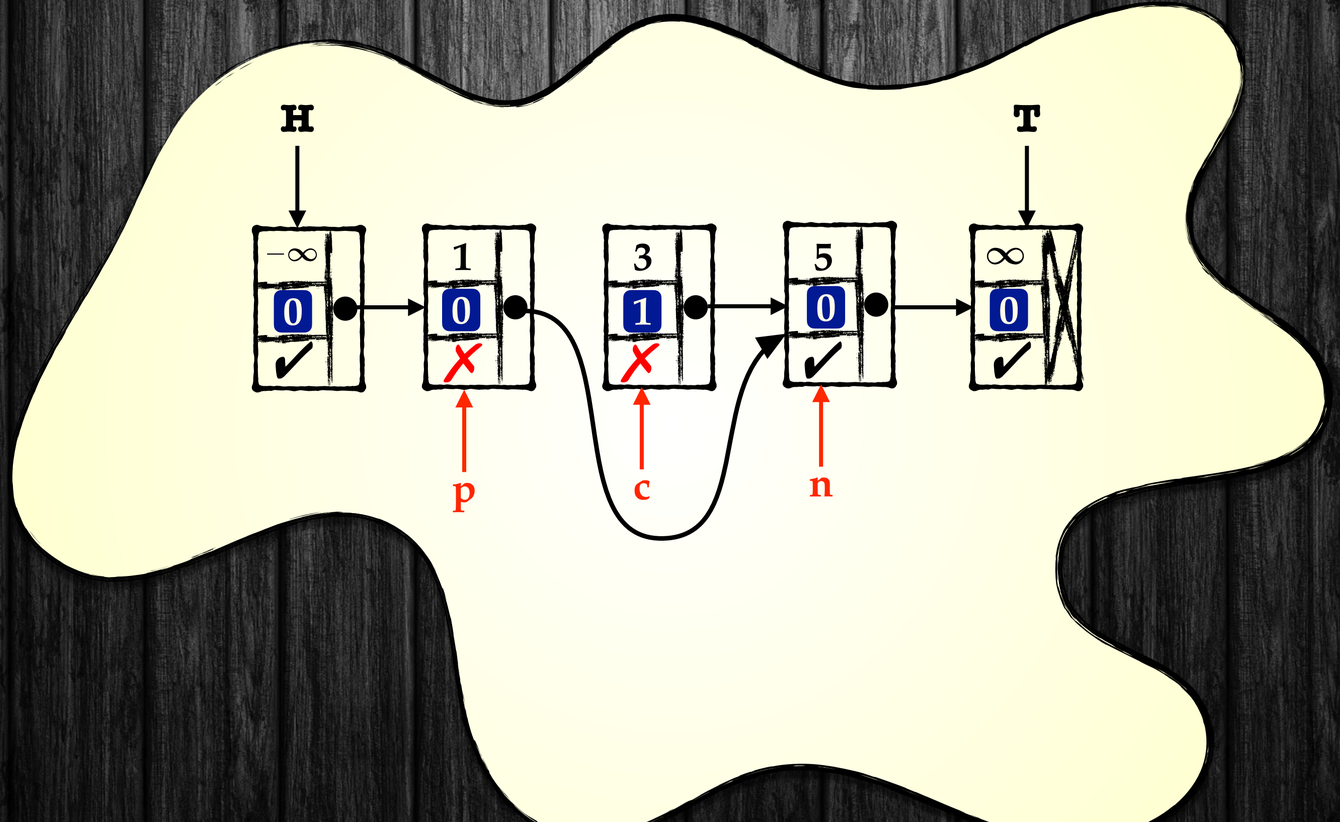


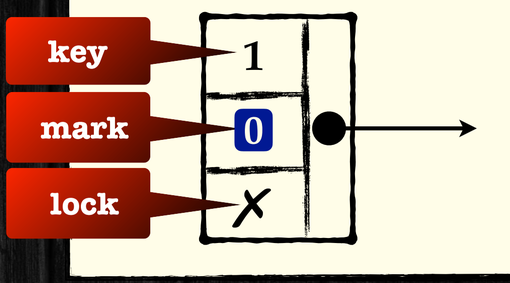


```
rmv(e):
local p, c, n, r
1  (p,c) := locate(e);
2  if (c.val = e)
3     c.mark := true
4     n := c.next;
5     p.next := n;
6     r := true;
7  else r := false;
8  unlock(p);
9  unlock(c);
10 return r;
```

rmv(3)

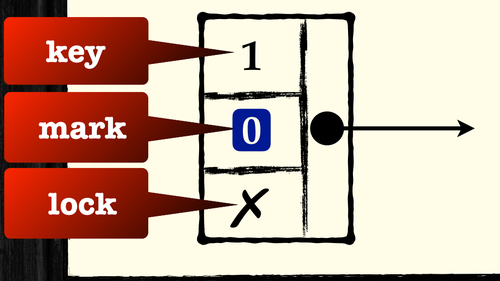
physical deletion

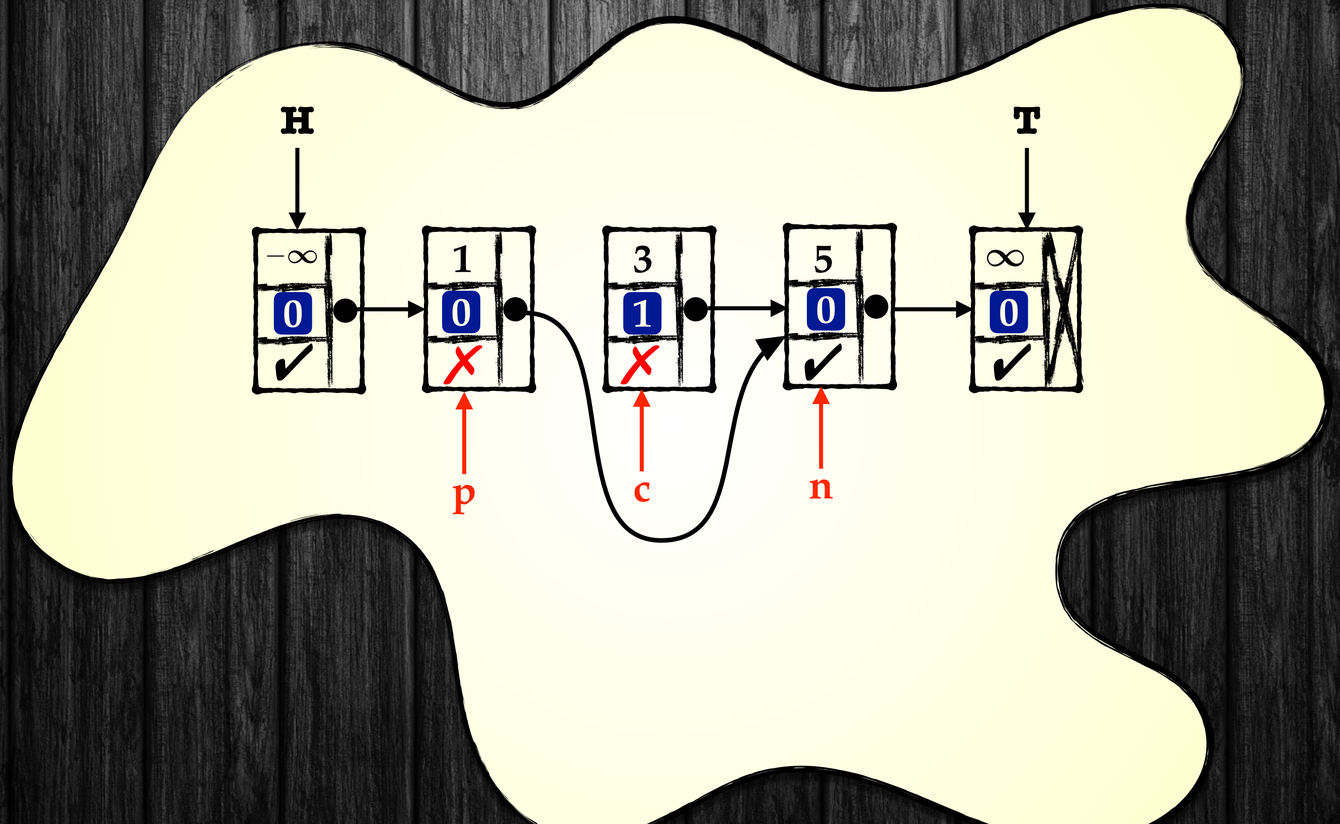




```
rmv(e):
local p, c, n, r
1  (p,c) := locate(e);
2  if (c.val = e)
3     c.mark := true
4     n := c.next;
5     p.next := n;
6     r := true;
7  else r := false;
8  unlock(p);
9  unlock(c);
10 return r;
```

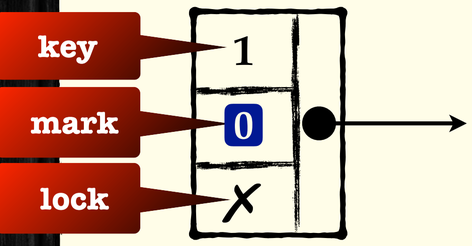


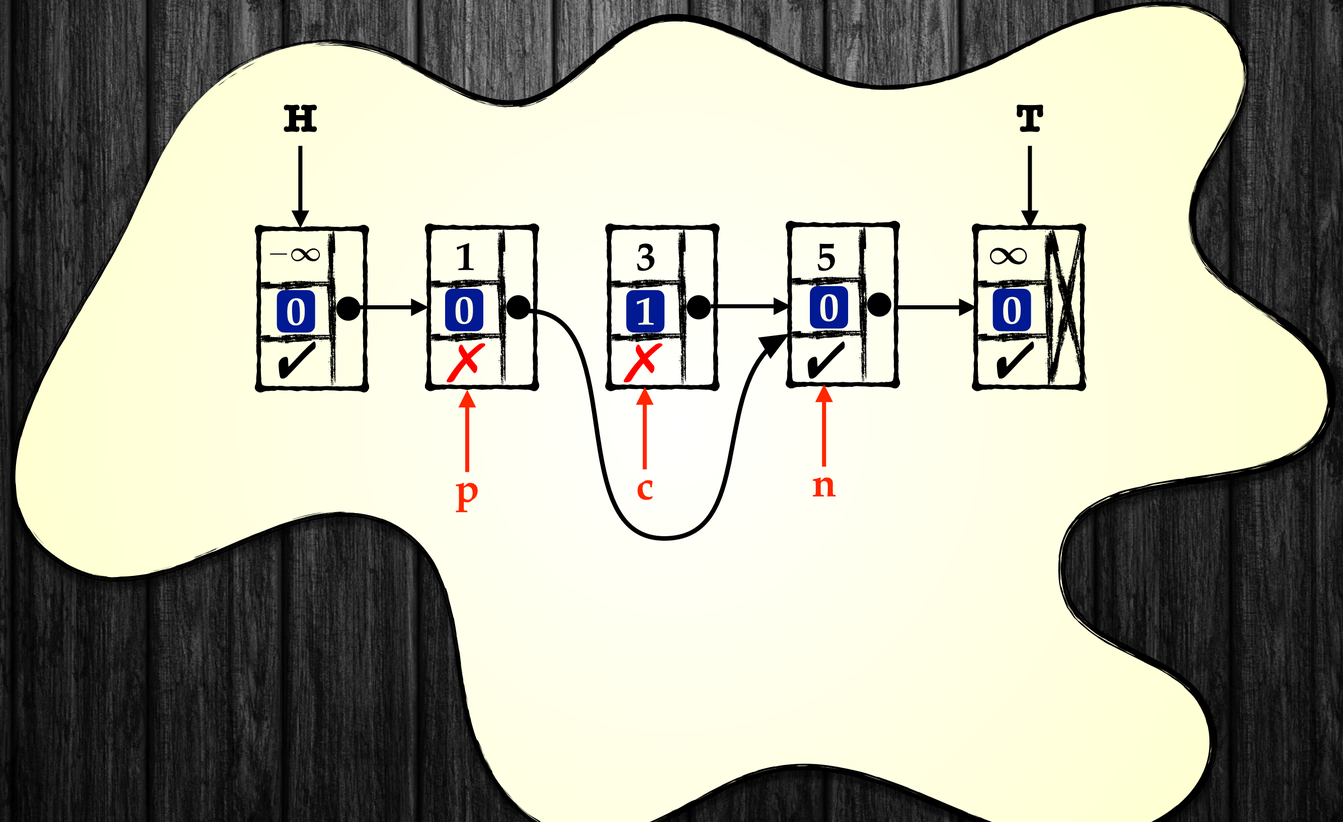




```
rmv(e):
local p, c, n, r
1  (p,c) := locate(e);
2  if (c.val = e)
3     c.mark := true
4     n := c.next;
5     p.next := n;
6     r := true;
7  else r := false;
8  unlock(p);
9  unlock(c);
10 return r;
```

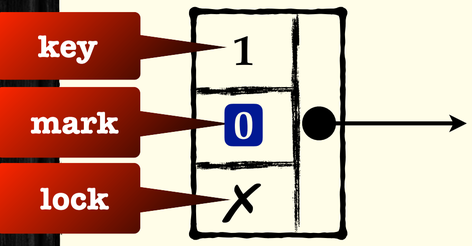
rmv(3)

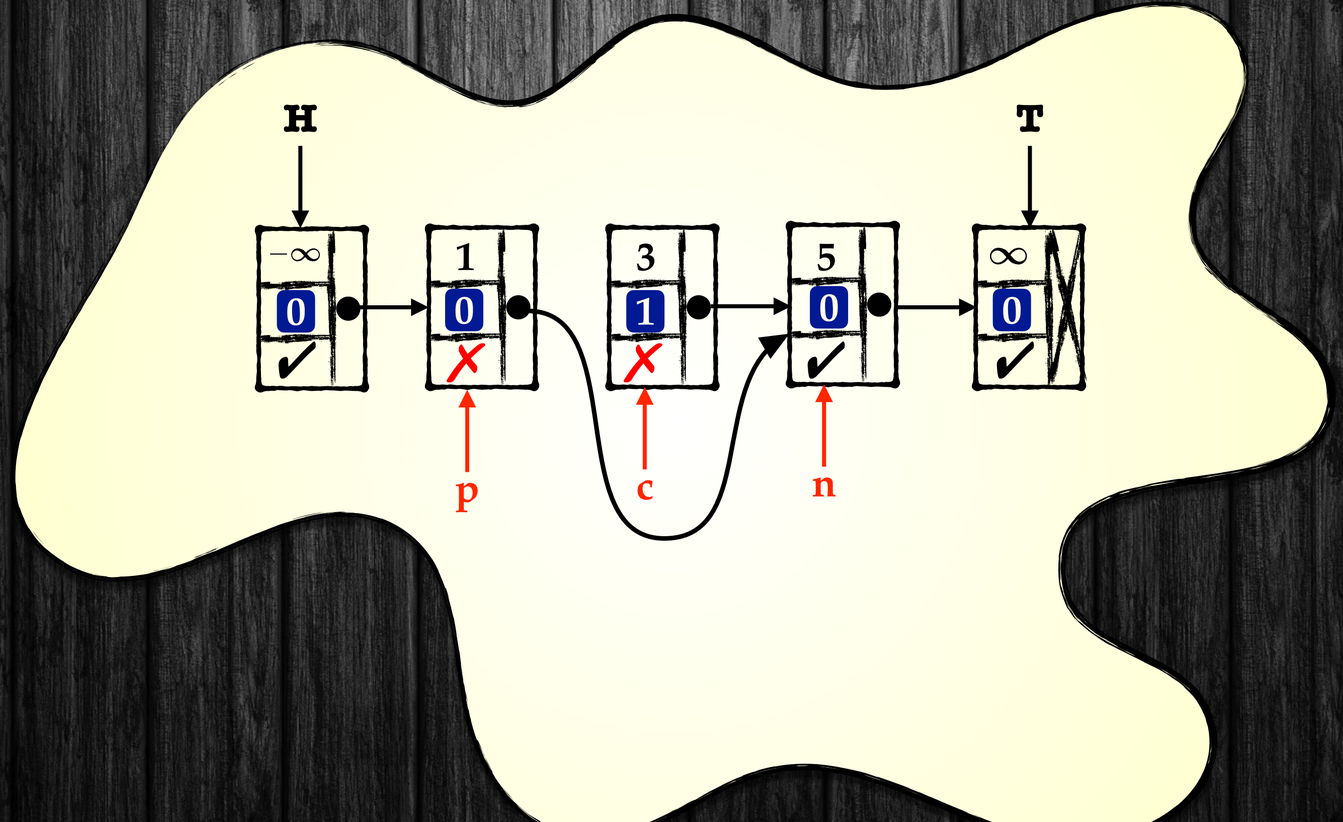




```
rmv(e):
local p, c, n, r
1  (p,c) := locate(e);
2  if (c.val = e)
3     c.mark := true
4     n := c.next;
5     p.next := n;
6     r := true;
7  else r := false;
8  unlock(p);
9  unlock(c);
10 return r;
```

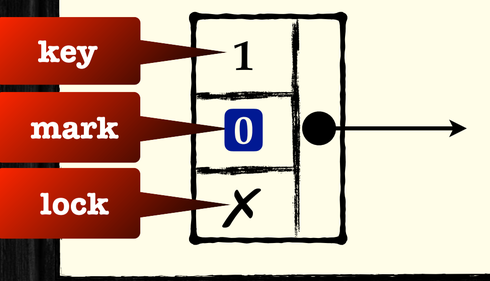
rmv(3)

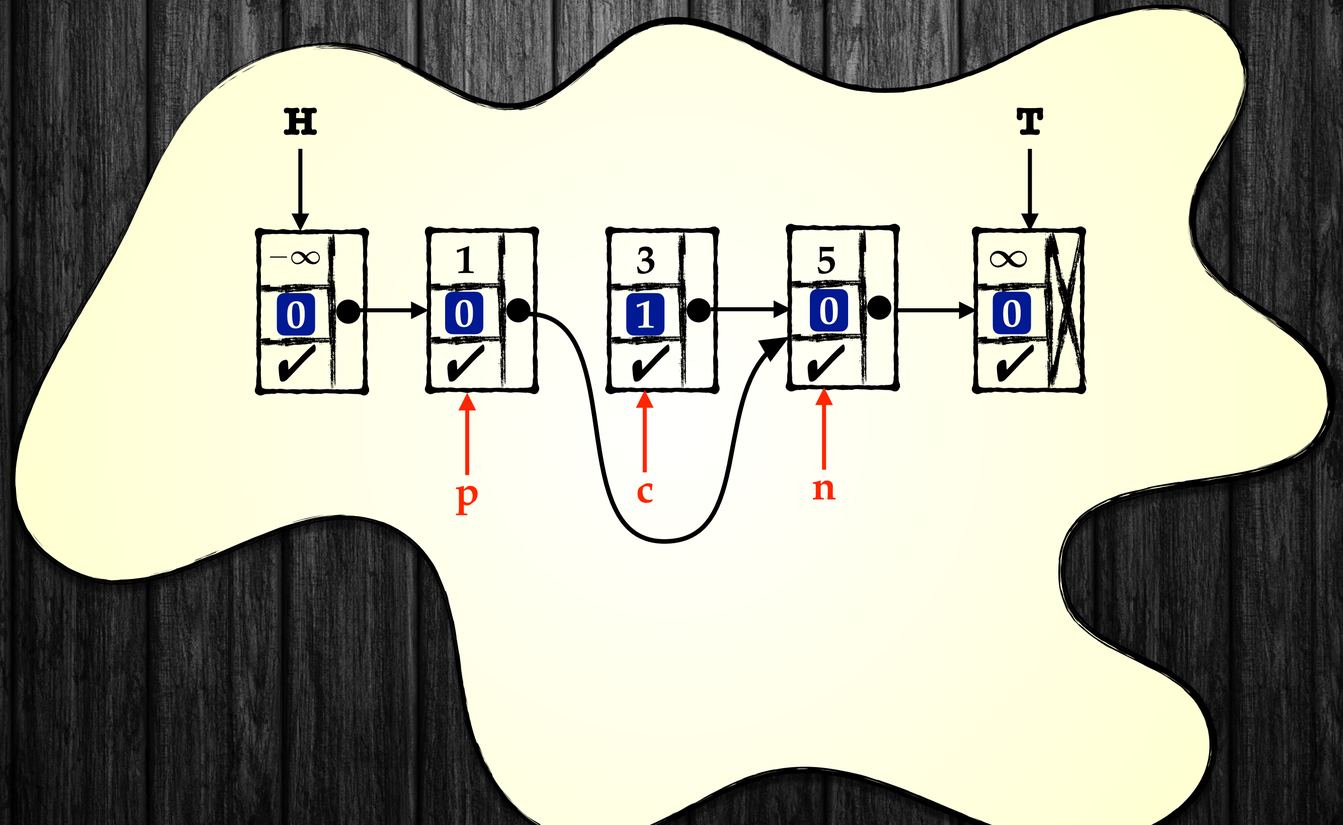




```
rmv(e):
local p, c, n, r
1  (p,c) := locate(e);
2  if (c.val = e)
3     c.mark := true
4     n := c.next;
5     p.next := n;
6     r := true;
7  else r := false;
8  unlock(p);
9  unlock(c);
10 return r;
```

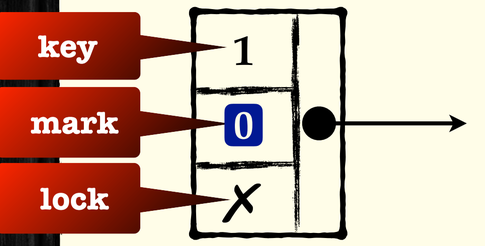
rmv(3)

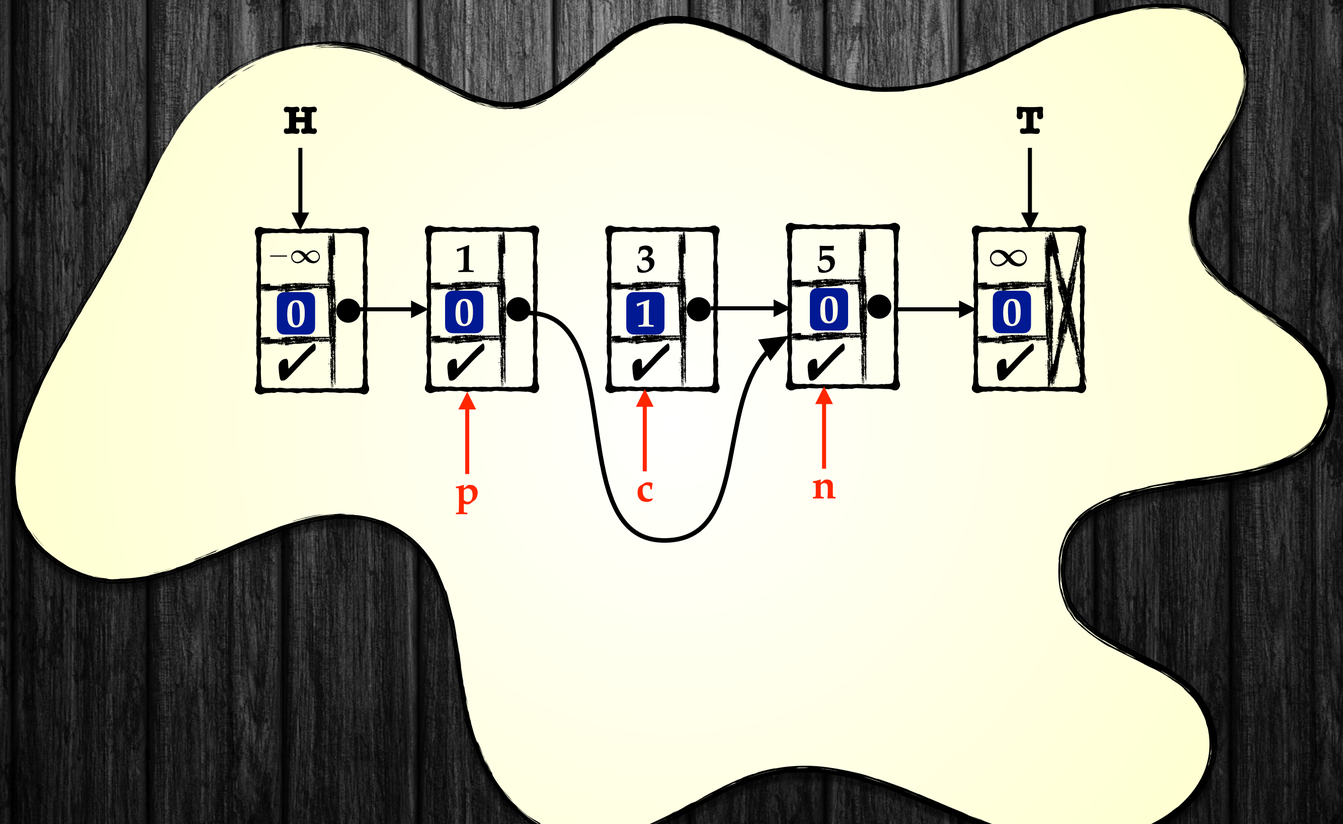




```
rmv(e):
local p, c, n, r
1  (p,c) := locate(e);
2  if (c.val = e)
3     c.mark := true
4     n := c.next;
5     p.next := n;
6     r := true;
7  else r := false;
8  unlock(p);
9  unlock(c);
10 return r;
```

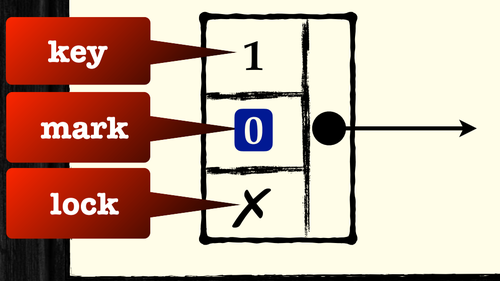
rmv(3)

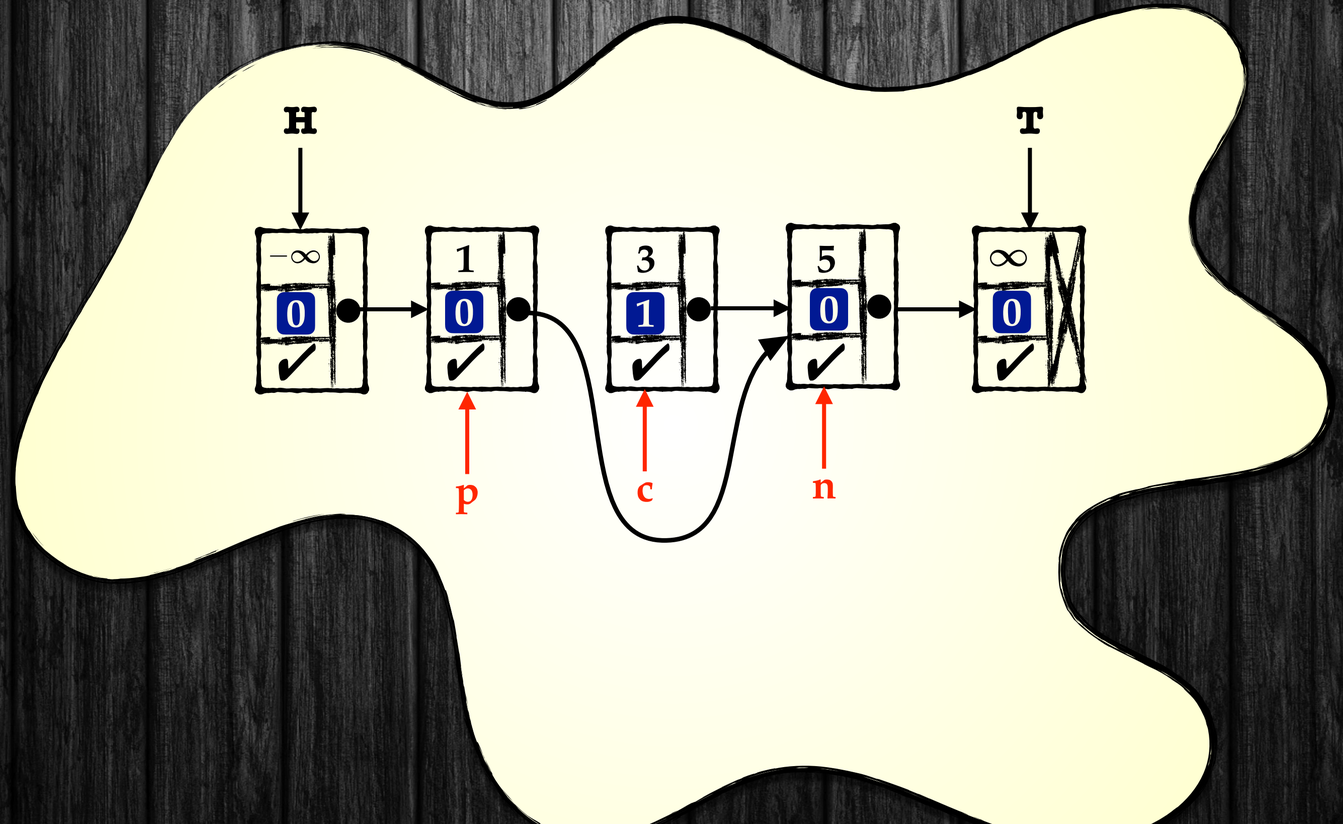




```
rmv(e):
local p, c, n, r
1  (p,c) := locate(e);
2  if (c.val = e)
3     c.mark := true
4     n := c.next;
5     p.next := n;
6     r := true;
7  else r := false;
8  unlock(p);
9  unlock(c);
10 return r;
```

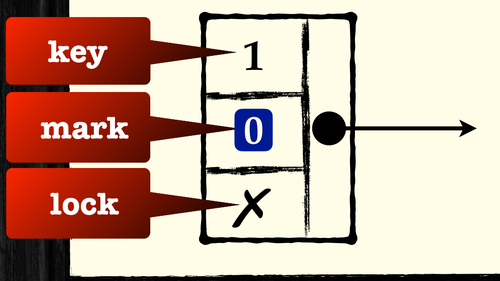


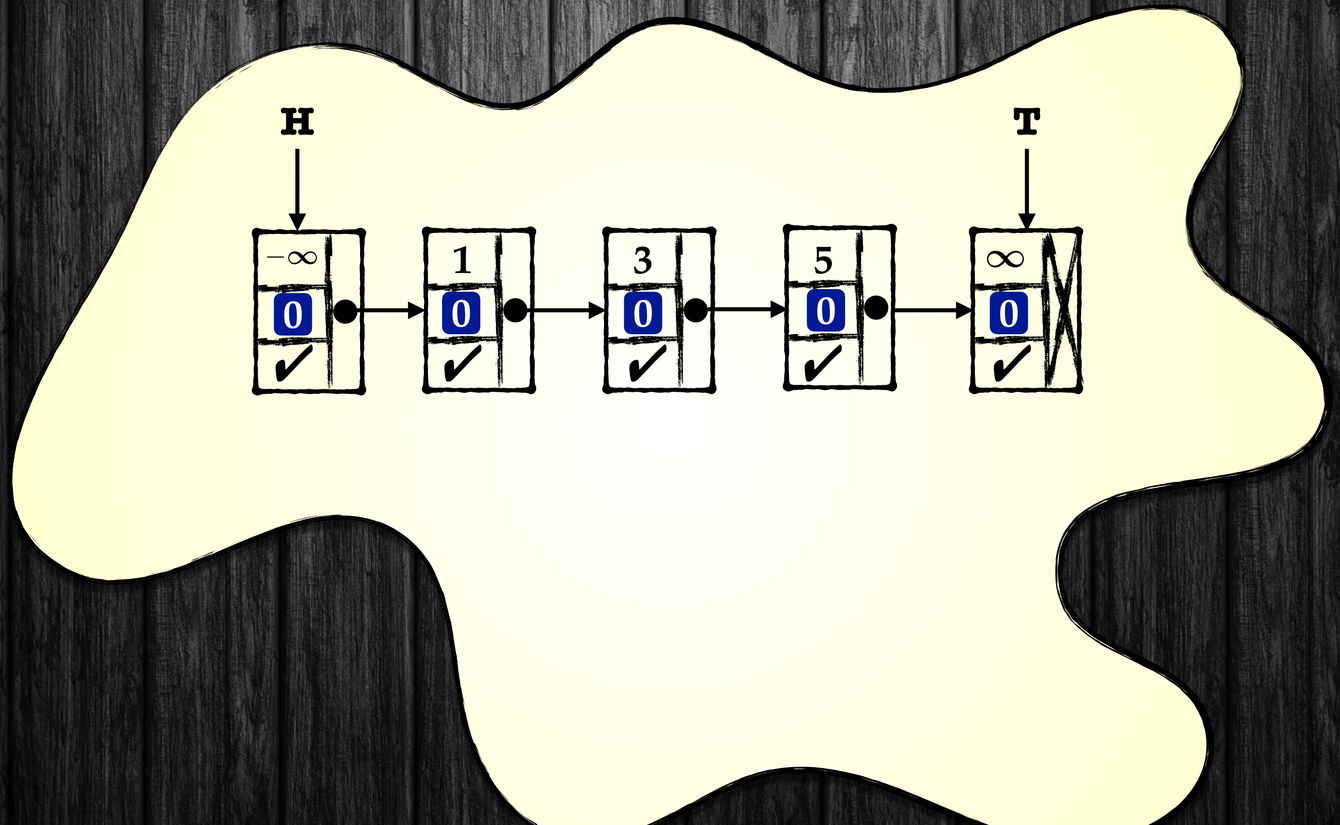




```
rmv(e):
local p, c, n, r
1  (p,c) := locate(e);
2  if (c.val = e)
3     c.mark := true
4     n := c.next;
5     p.next := n;
6     r := true;
7  else r := false;
8  unlock(p);
9  unlock(c);
10 return r;
```

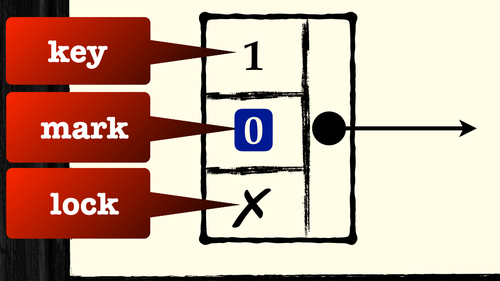


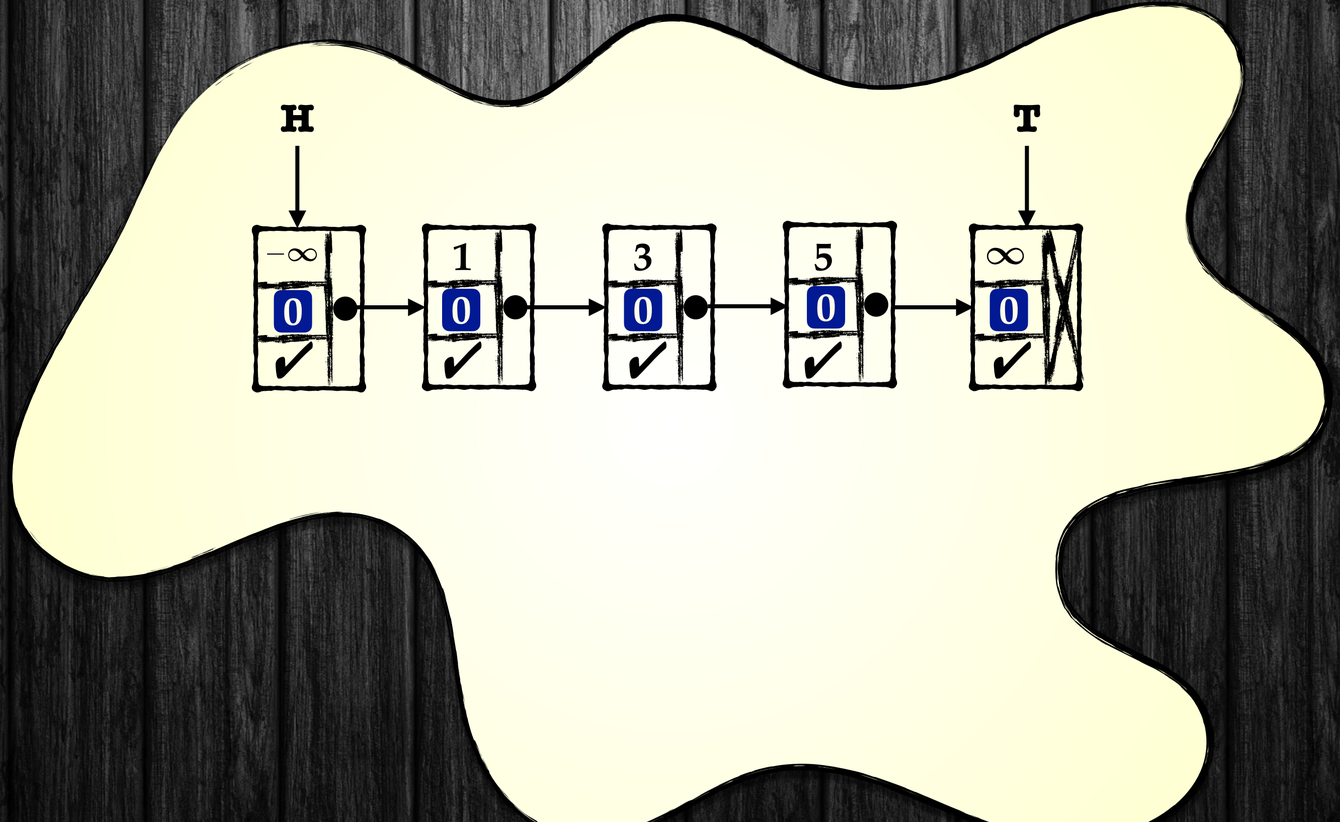




```
rmv(e):
local p, c, n, r
1  (p,c) := locate(e);
2  if (c.val = e)
3     c.mark := true
4     n := c.next;
5     p.next := n;
6     r := true;
7  else r := false;
8  unlock(p);
9  unlock(c);
10 return r;
```

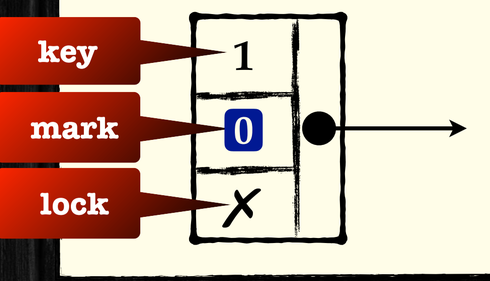


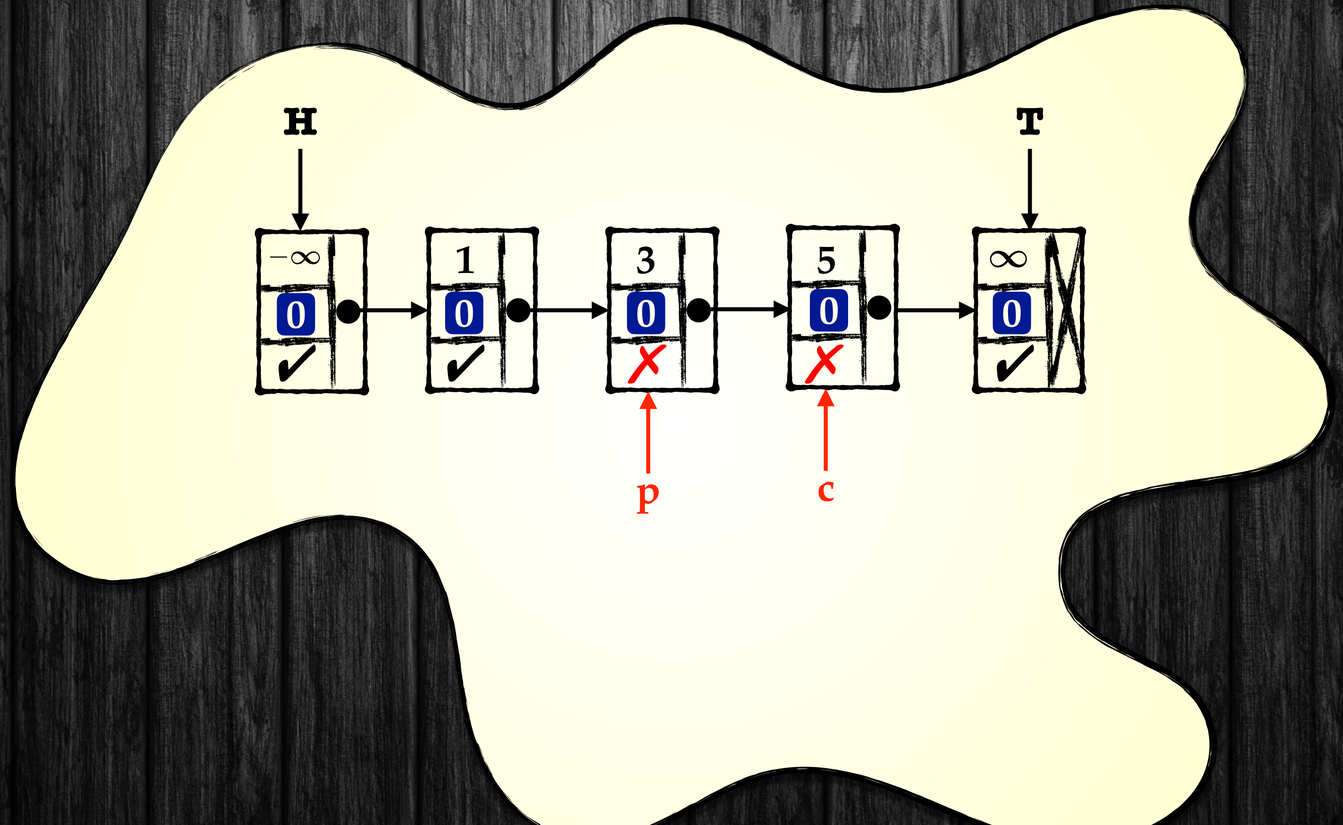




```
rmv(e):
local p, c, n, r
1  (p,c) := locate(e);
2  if (c.val = e)
3     c.mark := true
4     n := c.next;
5     p.next := n;
6     r := true;
7  else r := false;
8  unlock(p);
9  unlock(c);
10 return r;
```

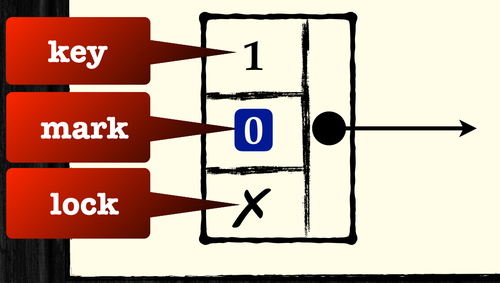
rmv(4)

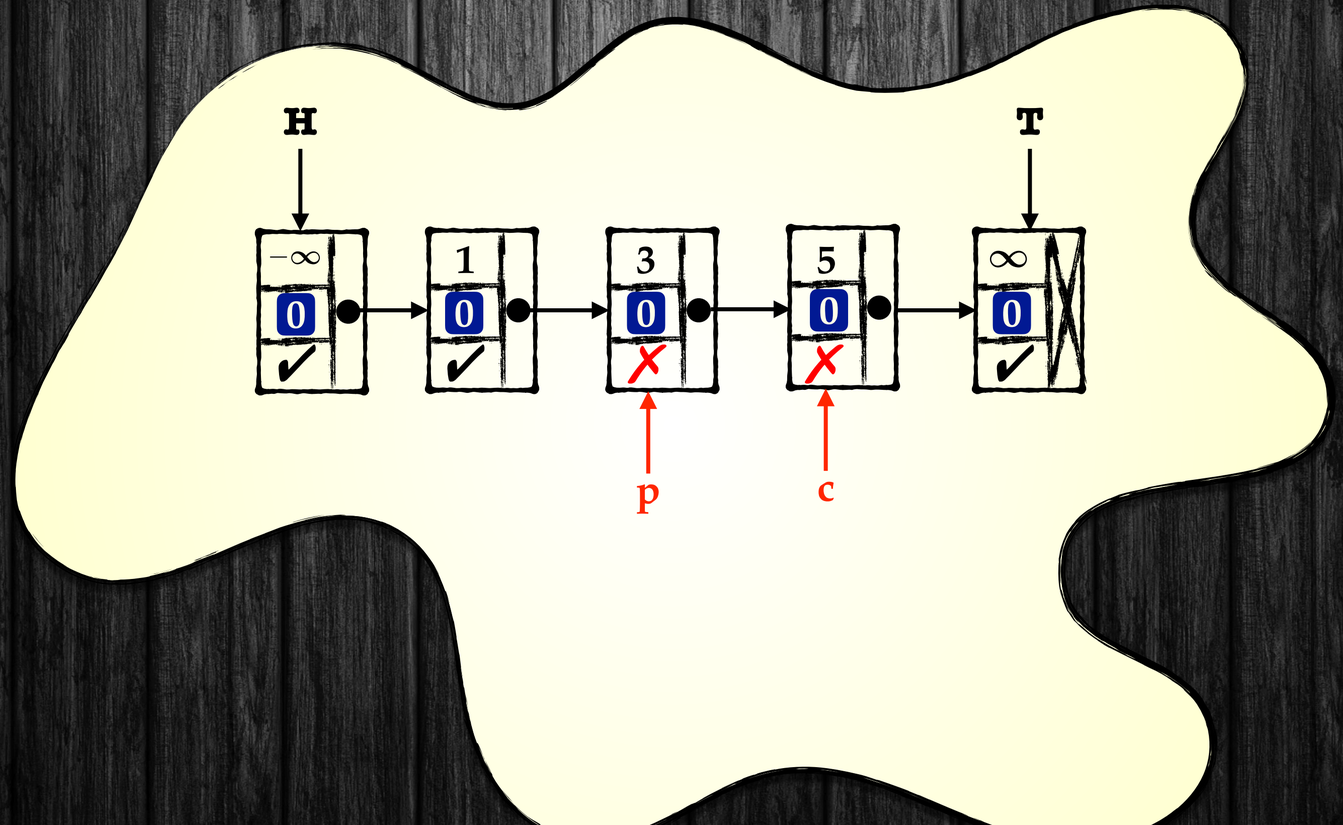




```
rmv(e):
local p, c, n, r
1  (p,c) := locate(e);
2  if (c.val = e)
3     c.mark := true
4     n := c.next;
5     p.next := n;
6     r := true;
7  else r := false;
8  unlock(p);
9  unlock(c);
10 return r;
```





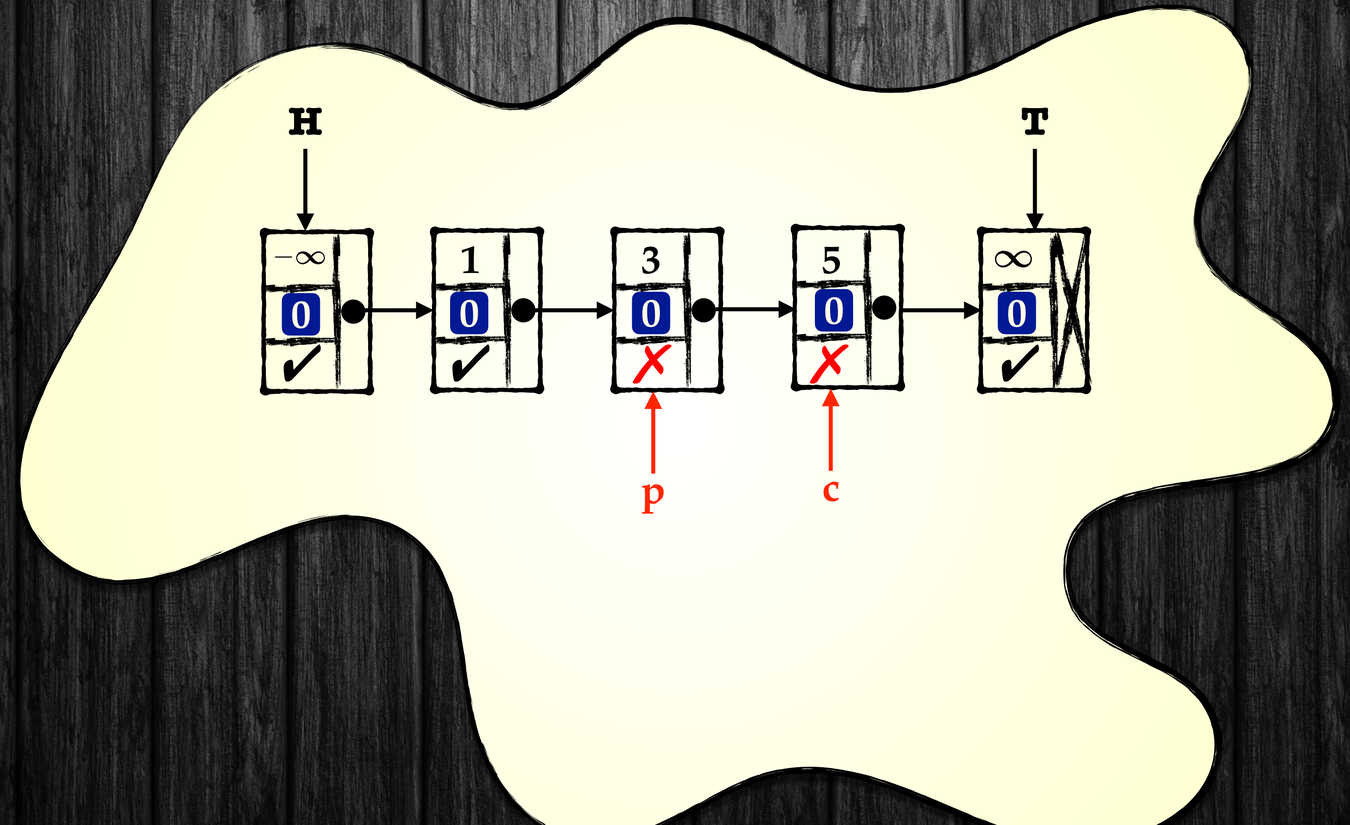


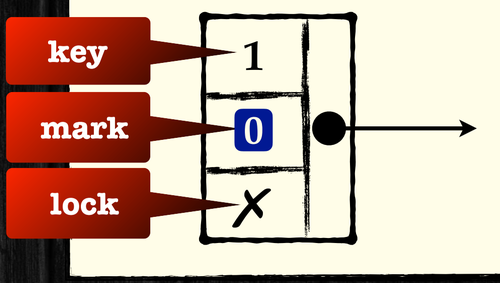
Sets | Lazy Synchronization

```
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;
```



linearization point



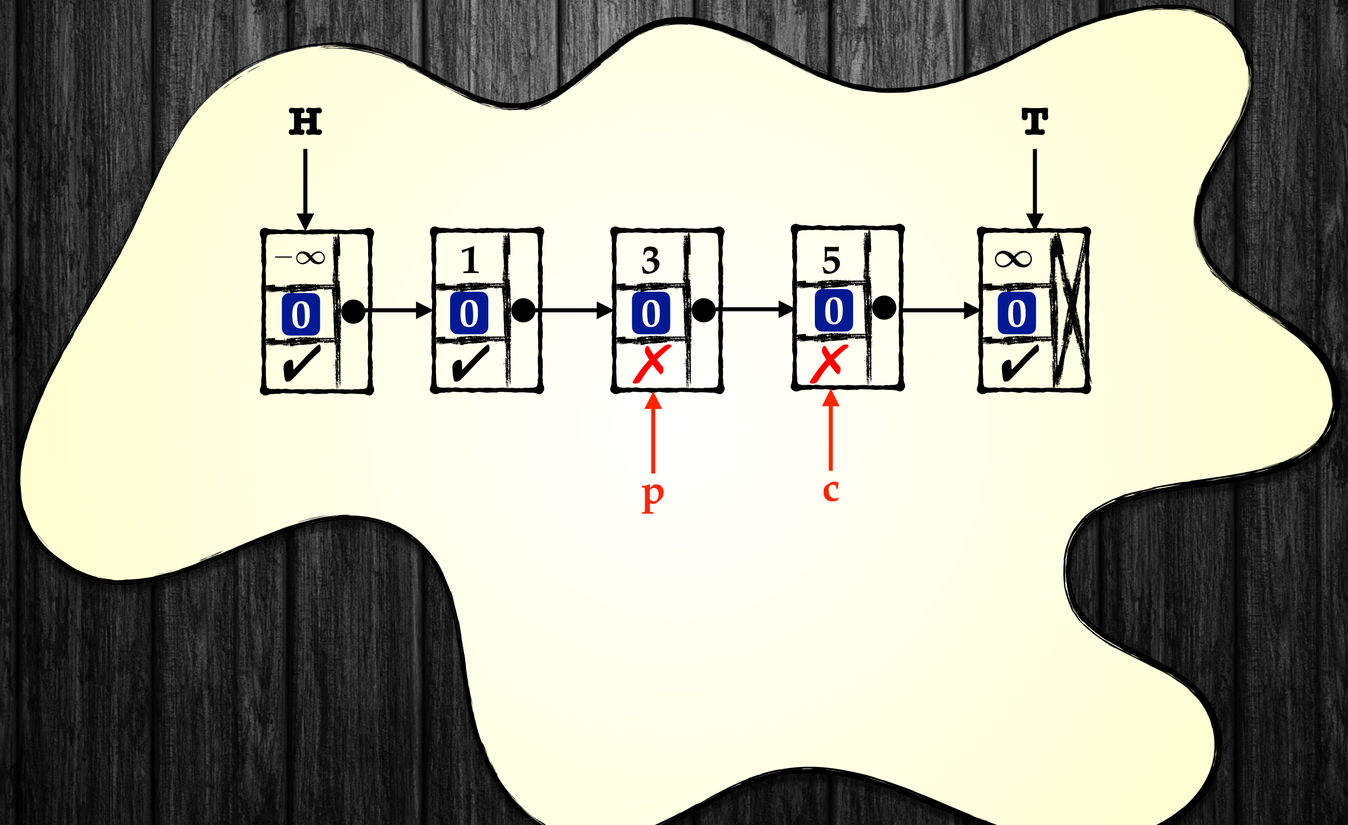


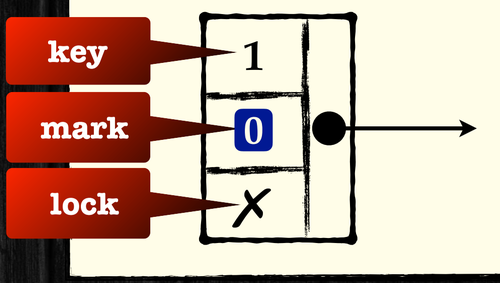
```
rmv(e):
local p, c, n, r
1  (p,c) := locate(e);
2  if (c.val = e)
3     c.mark := true
4     n := c.next;
5     p.next := n;
6     r := true;
7  else r := false;
8  unlock(p);
9  unlock(c);
10 return r;
```



unsuccessful remove

linearization point





Sets | Lazy Synchronization

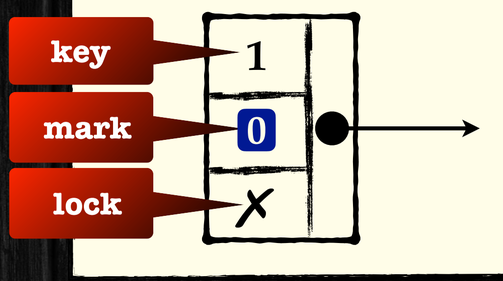
```
rmv(e):
local p, c, n, r
1  (p,c) := locate(e);
2  if (c.val = e)
3     c.mark := true
4     n := c.next;
5     p.next := n;
6     r := true;
7  else r := false;
8  unlock(p);
9  unlock(c);
10 return r;
```

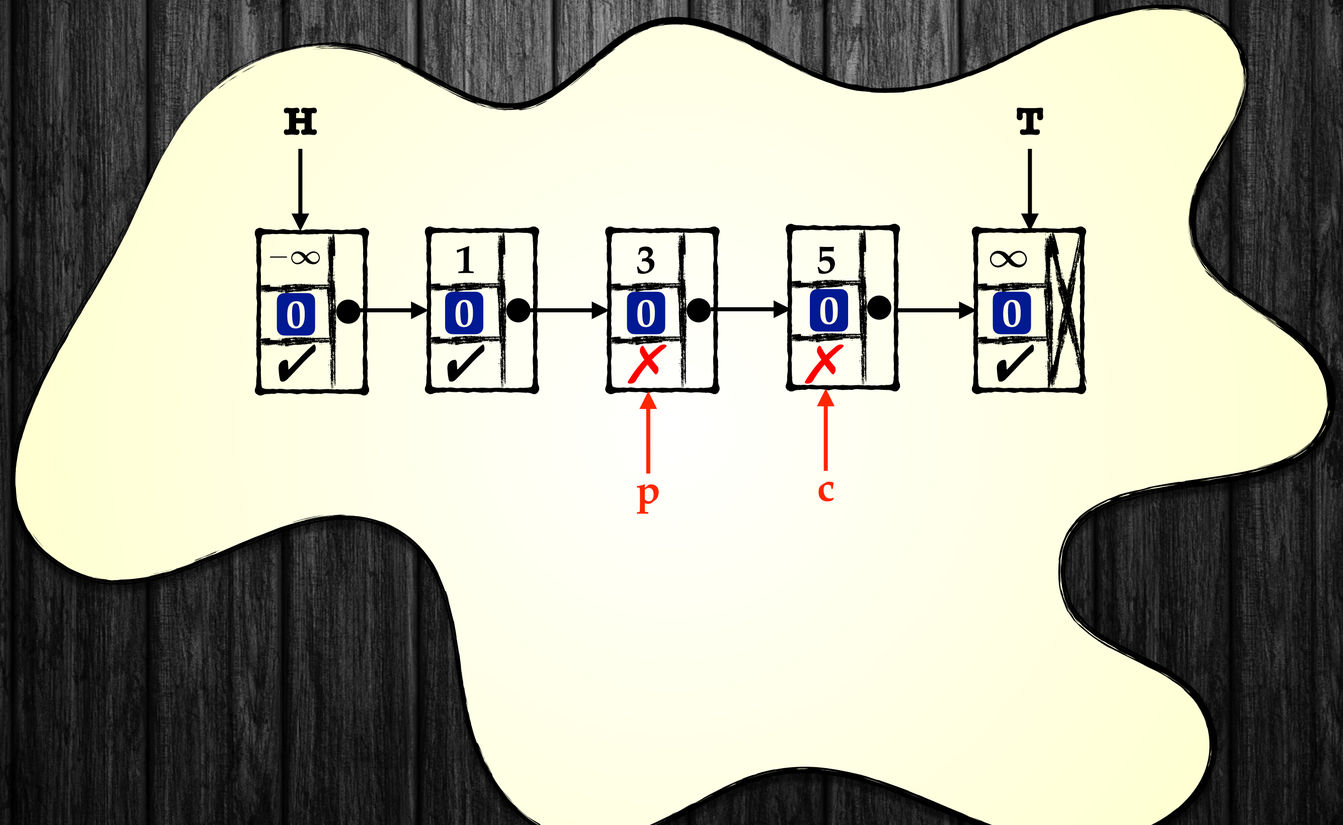
"first unmarked entry with larger key"

unsuccessful remove

linearization point







Sets | Lazy Synchronization

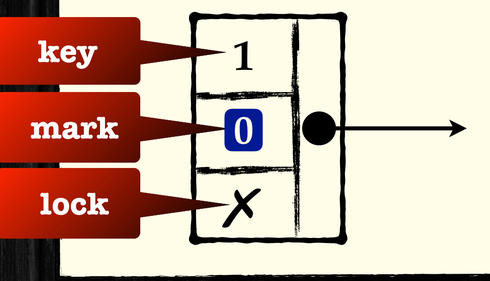
```
rmv(e):
local p, c, n, r
1  (p,c) := locate(e);
2  if (c.val = e)
3     c.mark := true
4     n := c.next;
5     p.next := n;
6     r := true;
7  else r := false;
8  unlock(p);
9  unlock(c);
10 return r;
```

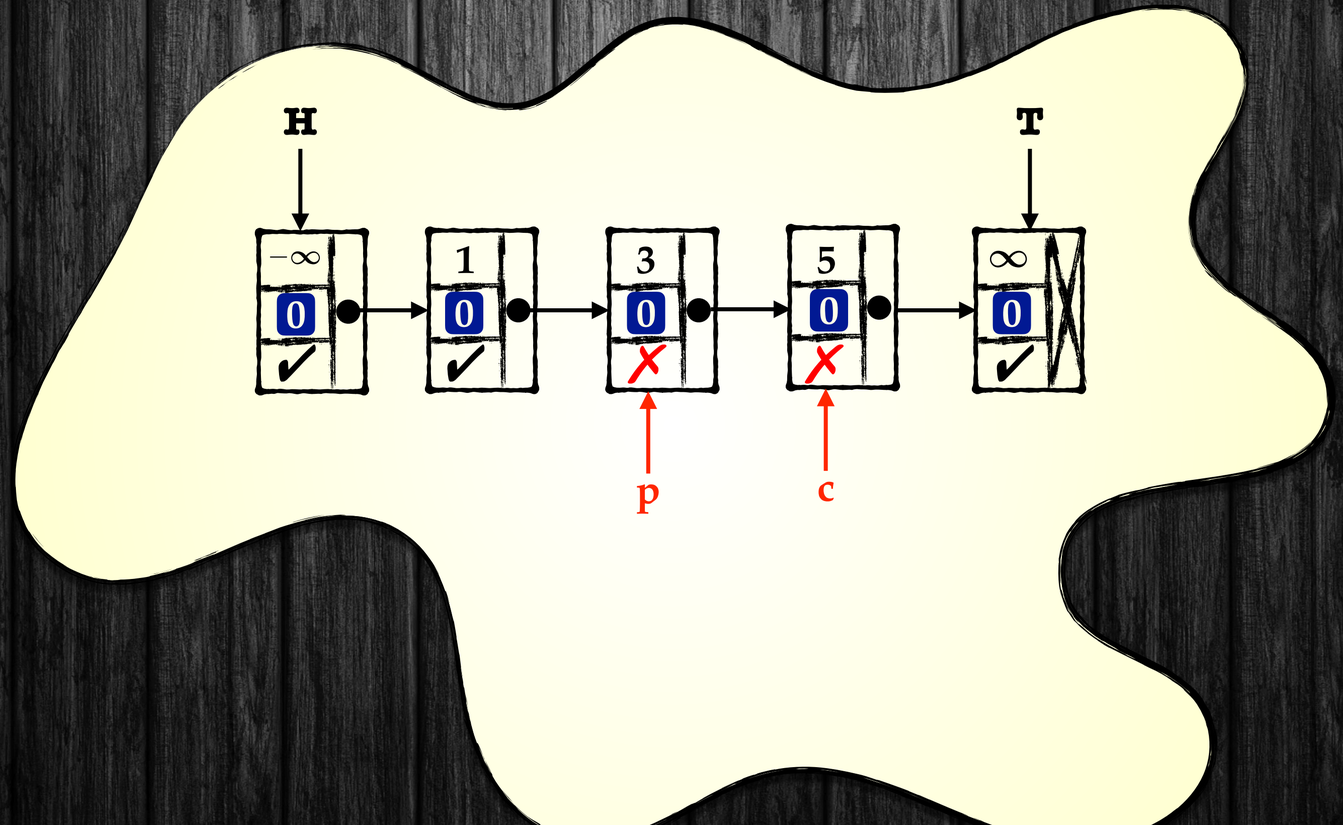
"first unmarked entry with larger key"

unsuccessful remove

linearization point

rmv(4)





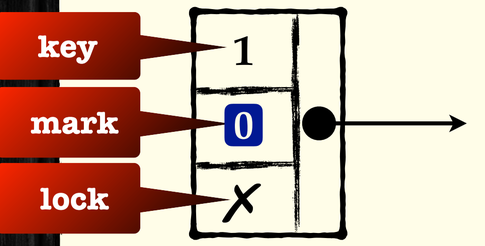
```
rmv(e):
local p, c, n, r
1  (p,c) := locate(e);
2  if (c.val = e)
3     c.mark := true
4     n := c.next;
5     p.next := n;
6     r := true;
7  else r := false;
8  unlock(p);
9  unlock(c);
10 return r;
```

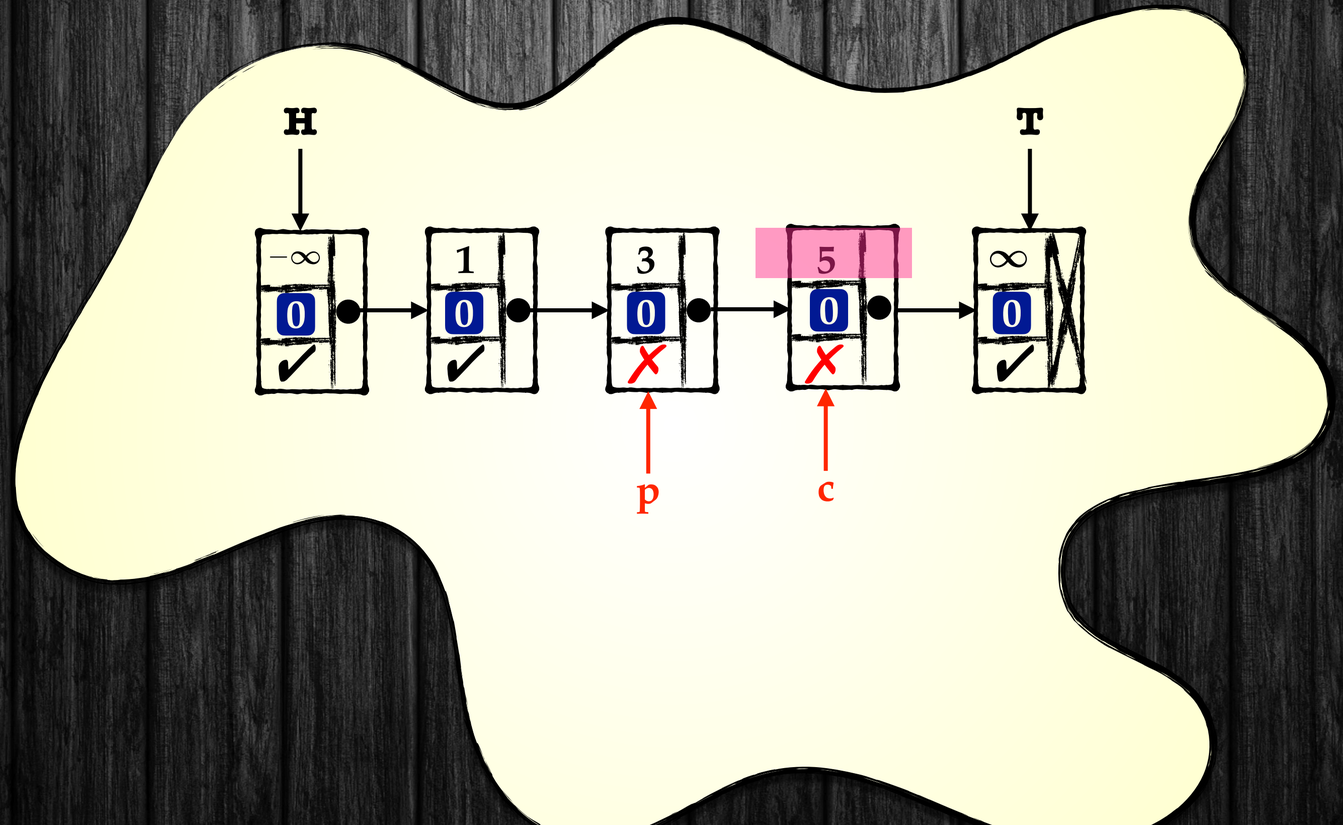
rmv(4)

"first unmarked entry with larger key"

unsuccessful remove

linearization point





Sets | Lazy Synchronization

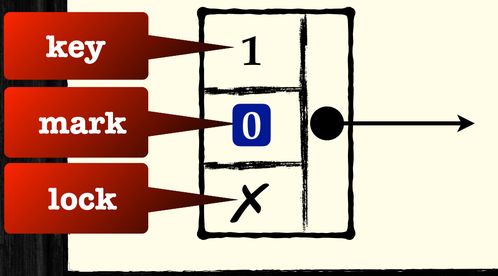
```
rmv(e):
local p, c, n, r
1  (p,c) := locate(e);
2  if (c.val = e)
3     c.mark := true
4     n := c.next;
5     p.next := n;
6     r := true;
7  else r := false;
8  unlock(p);
9  unlock(c);
10 return r;
```

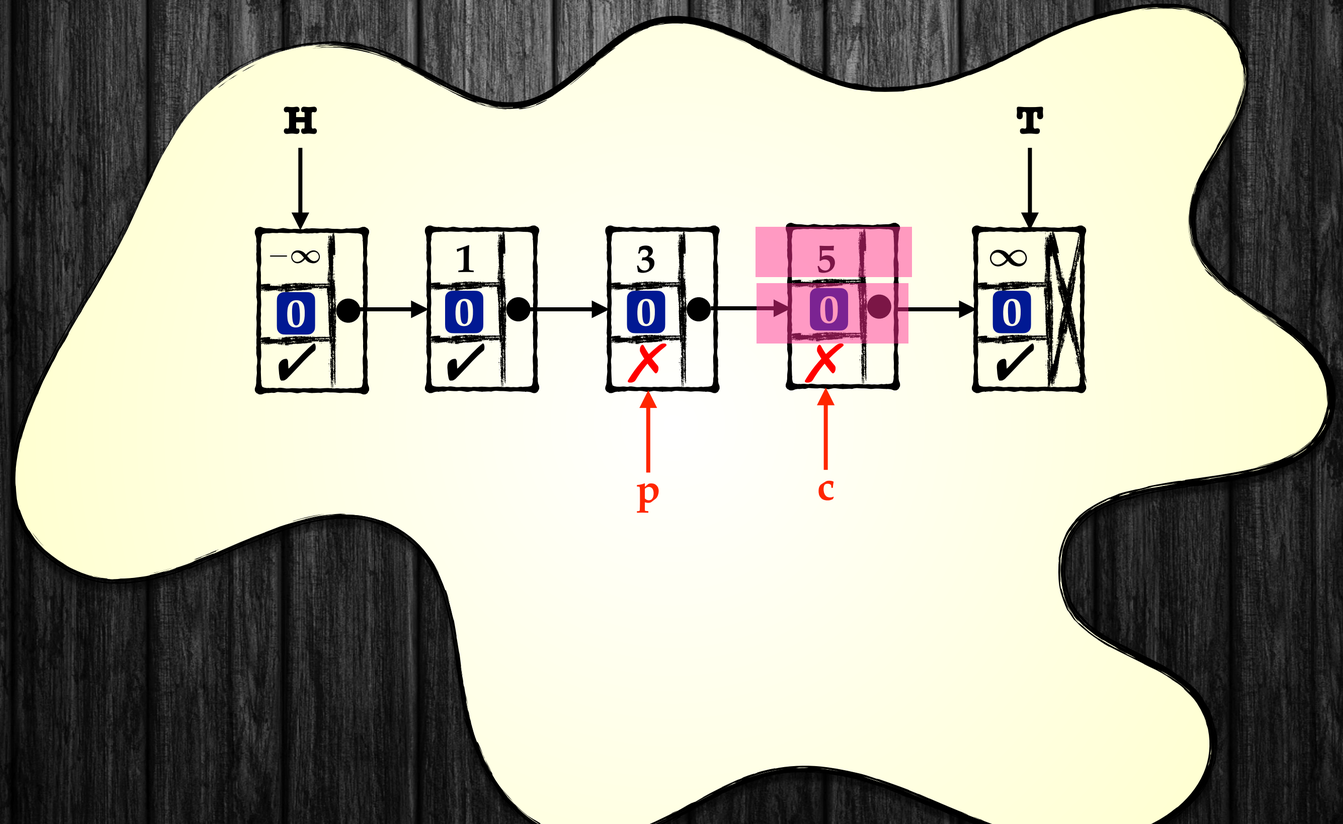
"first unmarked entry with larger key"

unsuccessful remove

linearization point







Sets | Lazy Synchronization

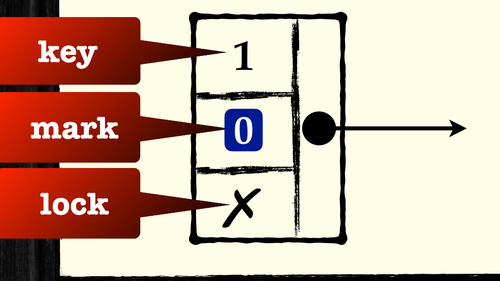
```
rmv(e):
local p, c, n, r
1  (p,c) := locate(e);
2  if (c.val = e)
3     c.mark := true
4     n := c.next;
5     p.next := n;
6     r := true;
7  else r := false;
8  unlock(p);
9  unlock(c);
10 return r;
```

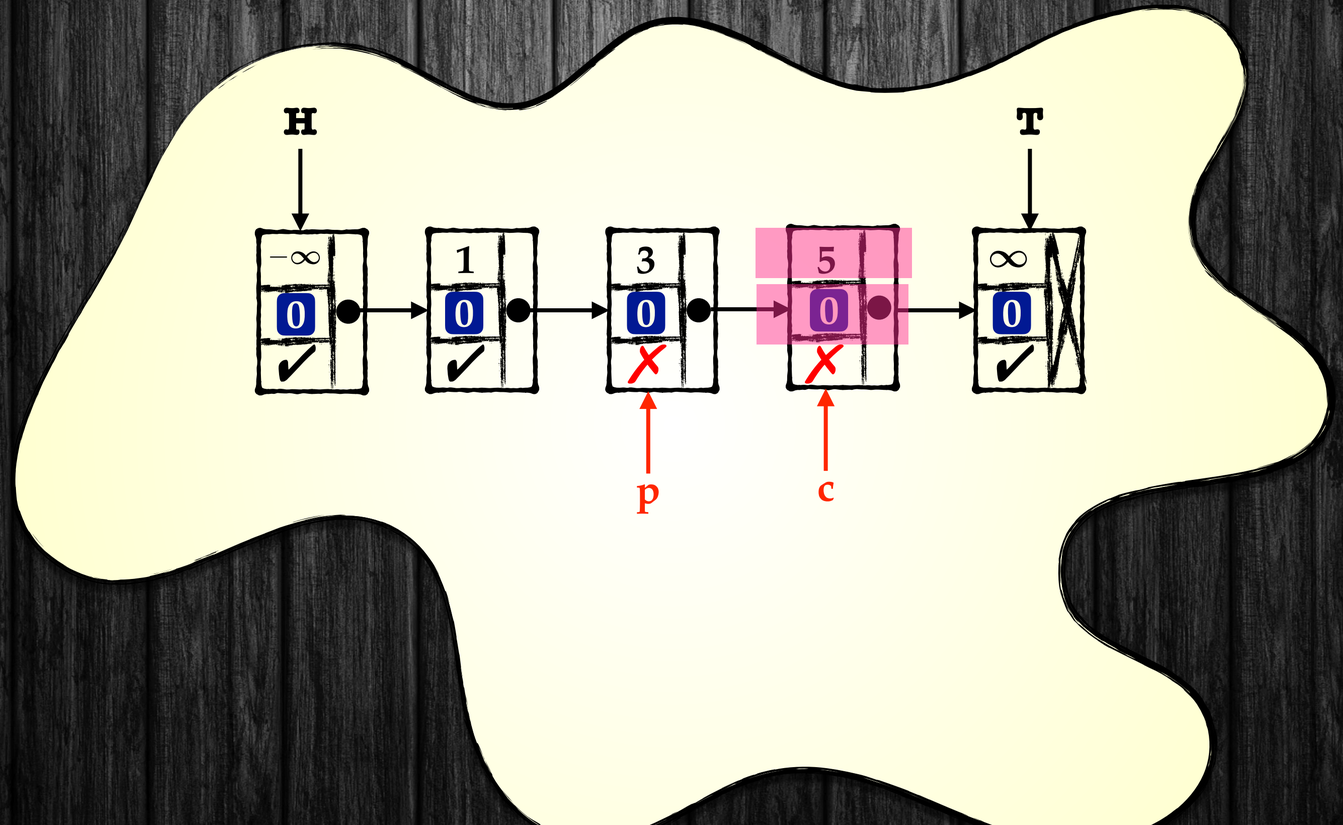
"first unmarked entry with larger key"

unsuccessful remove

linearization point

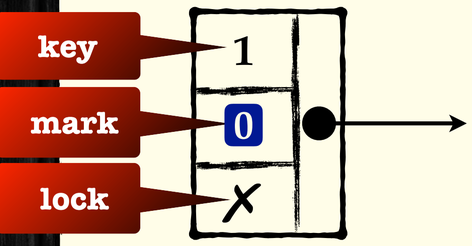
rmv(4)

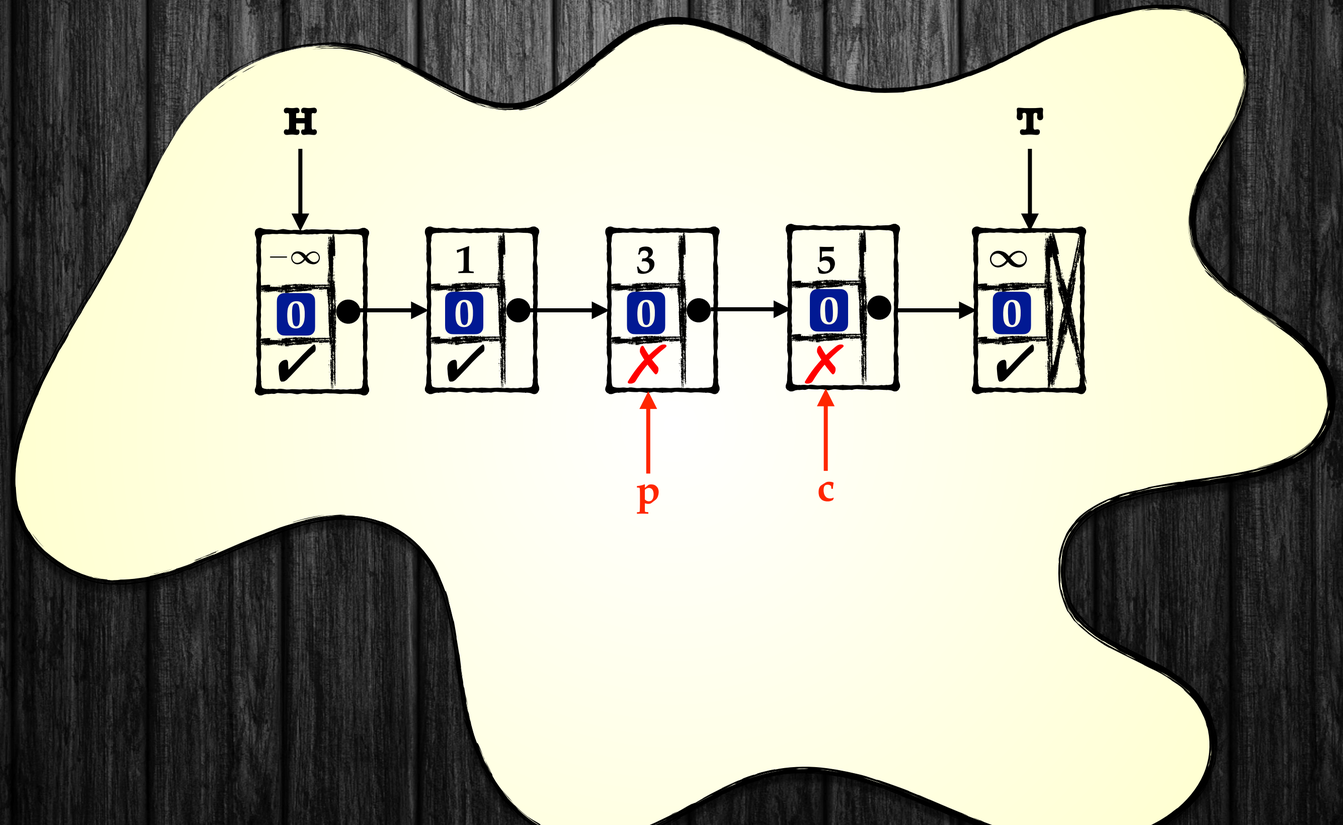




```
rmv(e):
local p, c, n, r
1  (p,c) := locate(e);
2  if (c.val = e)
3     c.mark := true
4     n := c.next;
5     p.next := n;
6     r := true;
7  else r := false;
8  unlock(p);
9  unlock(c);
10 return r;
```

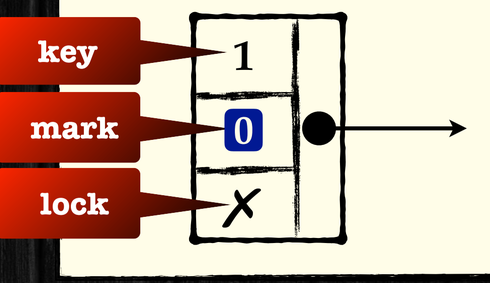
rmv(4)

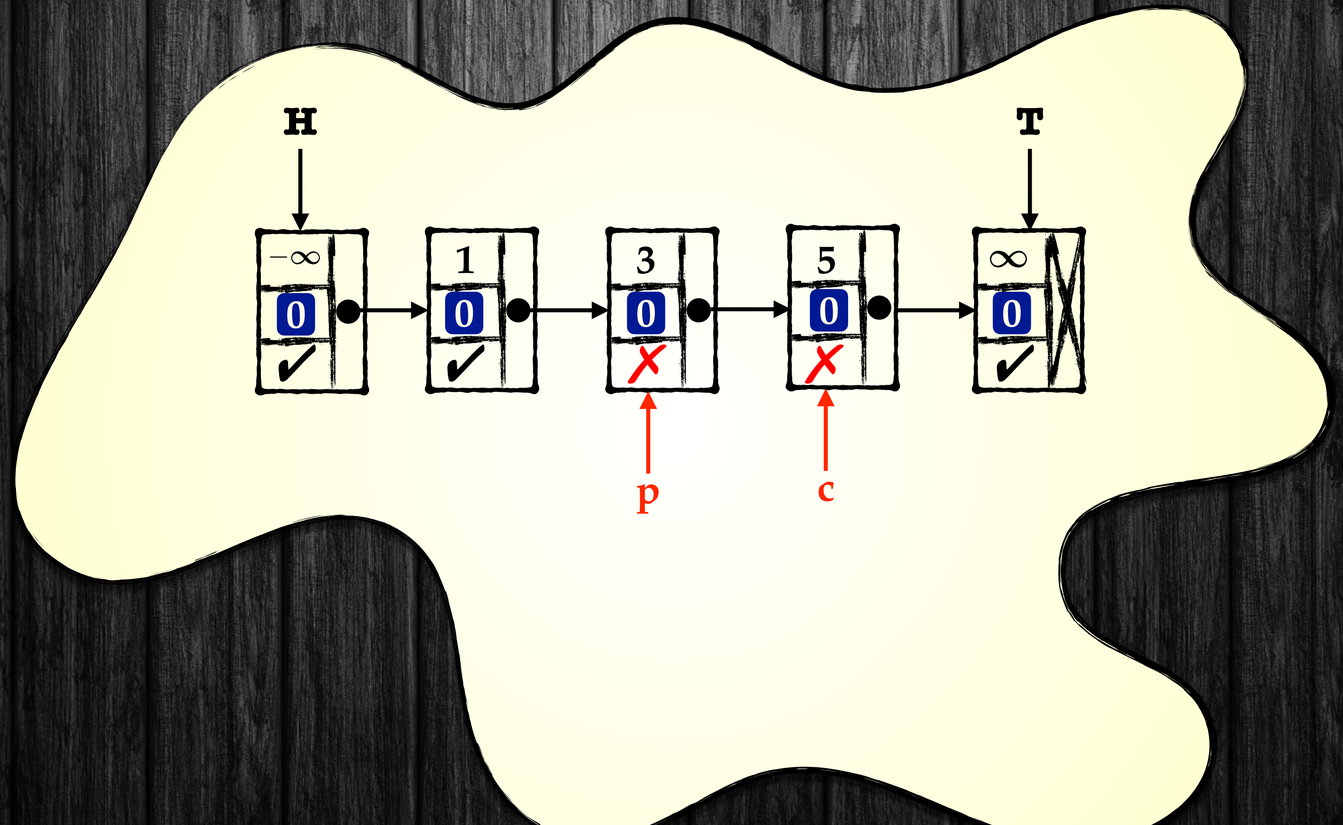




```
rmv(e):
local p, c, n, r
1  (p,c) := locate(e);
2  if (c.val = e)
3     c.mark := true
4     n := c.next;
5     p.next := n;
6     r := true;
7  else r := false;
8  unlock(p);
9  unlock(c);
10 return r;
```

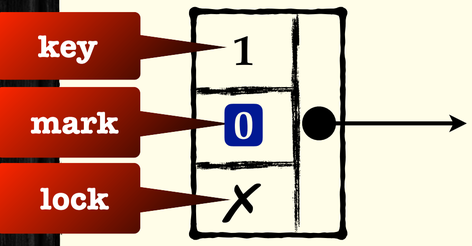
rmv(4)

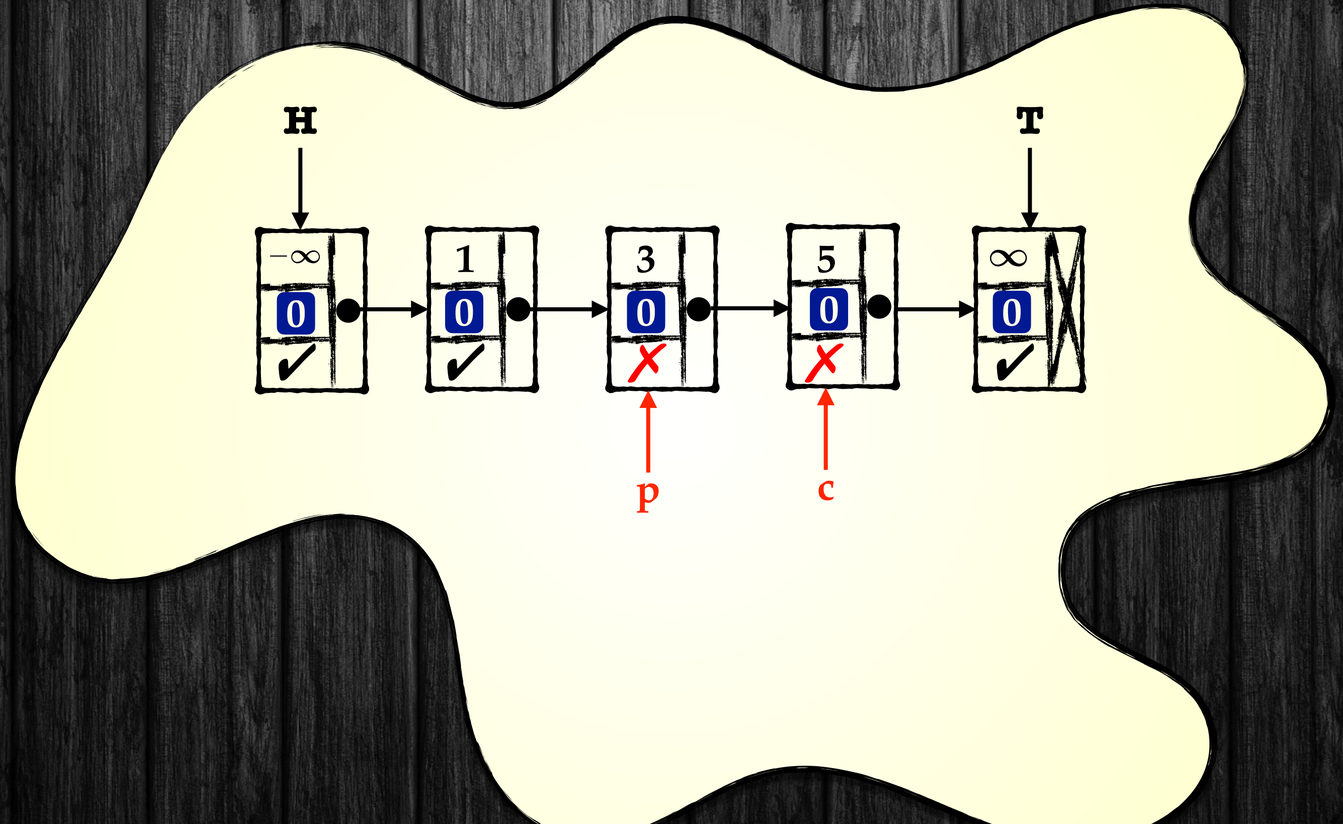




```
rmv(e):
local p, c, n, r
1  (p,c) := locate(e);
2  if (c.val = e)
3     c.mark := true
4     n := c.next;
5     p.next := n;
6     r := true;
7  else r := false;
8  unlock(p);
9  unlock(c);
10 return r;
```

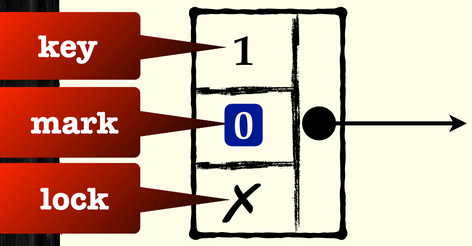
rmv(4)

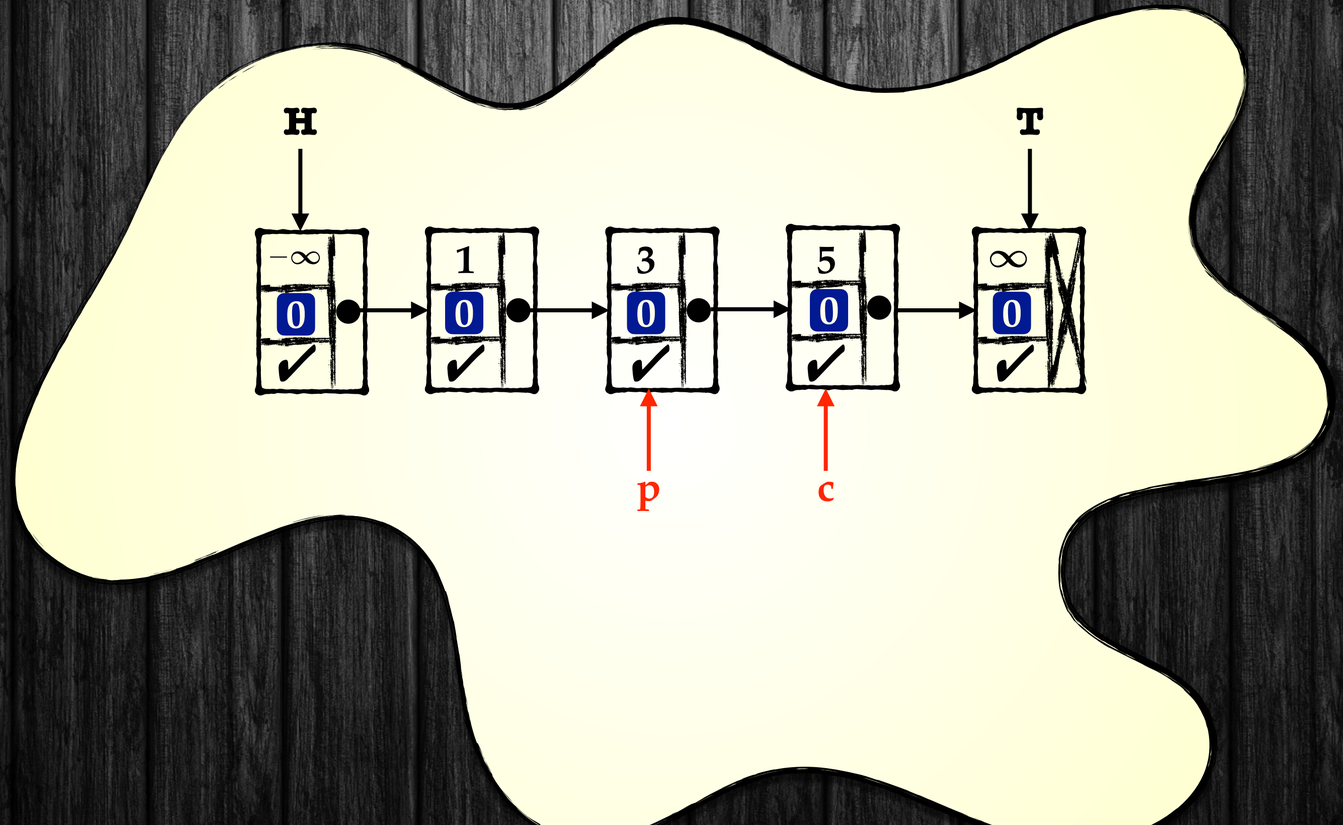




```
rmv(e):
local p, c, n, r
1  (p,c) := locate(e);
2  if (c.val = e)
3     c.mark := true
4     n := c.next;
5     p.next := n;
6     r := true;
7  else r := false;
8  unlock(p);
9  unlock(c);
10 return r;
```

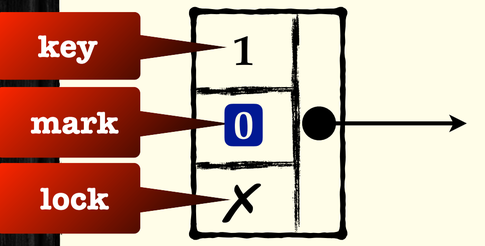
rmv(4)

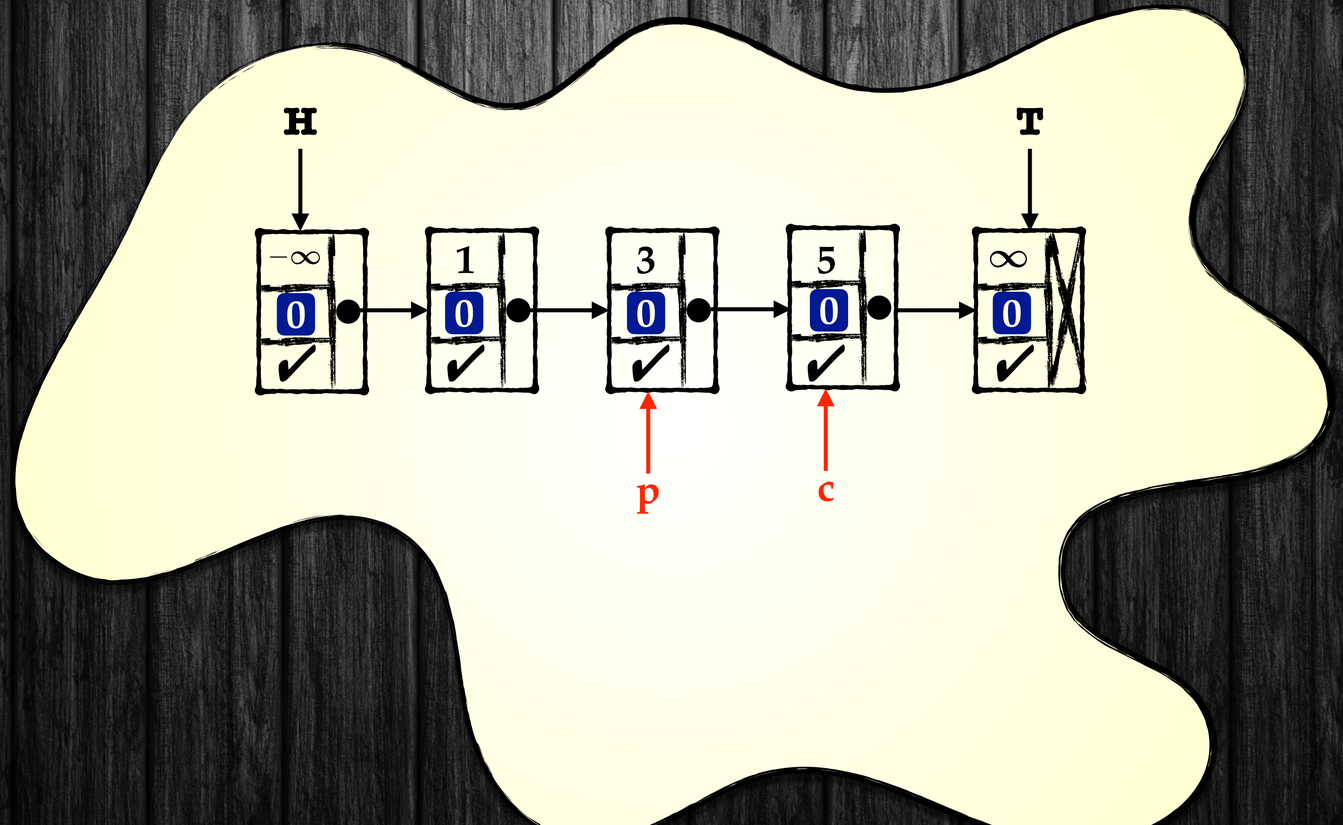




```
rmv(e):
local p, c, n, r
1  (p,c) := locate(e);
2  if (c.val = e)
3     c.mark := true
4     n := c.next;
5     p.next := n;
6     r := true;
7  else r := false;
8  unlock(p);
9  unlock(c);
10 return r;
```

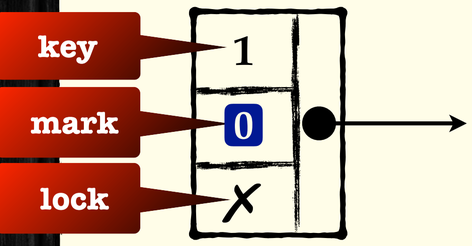
rmv(4)

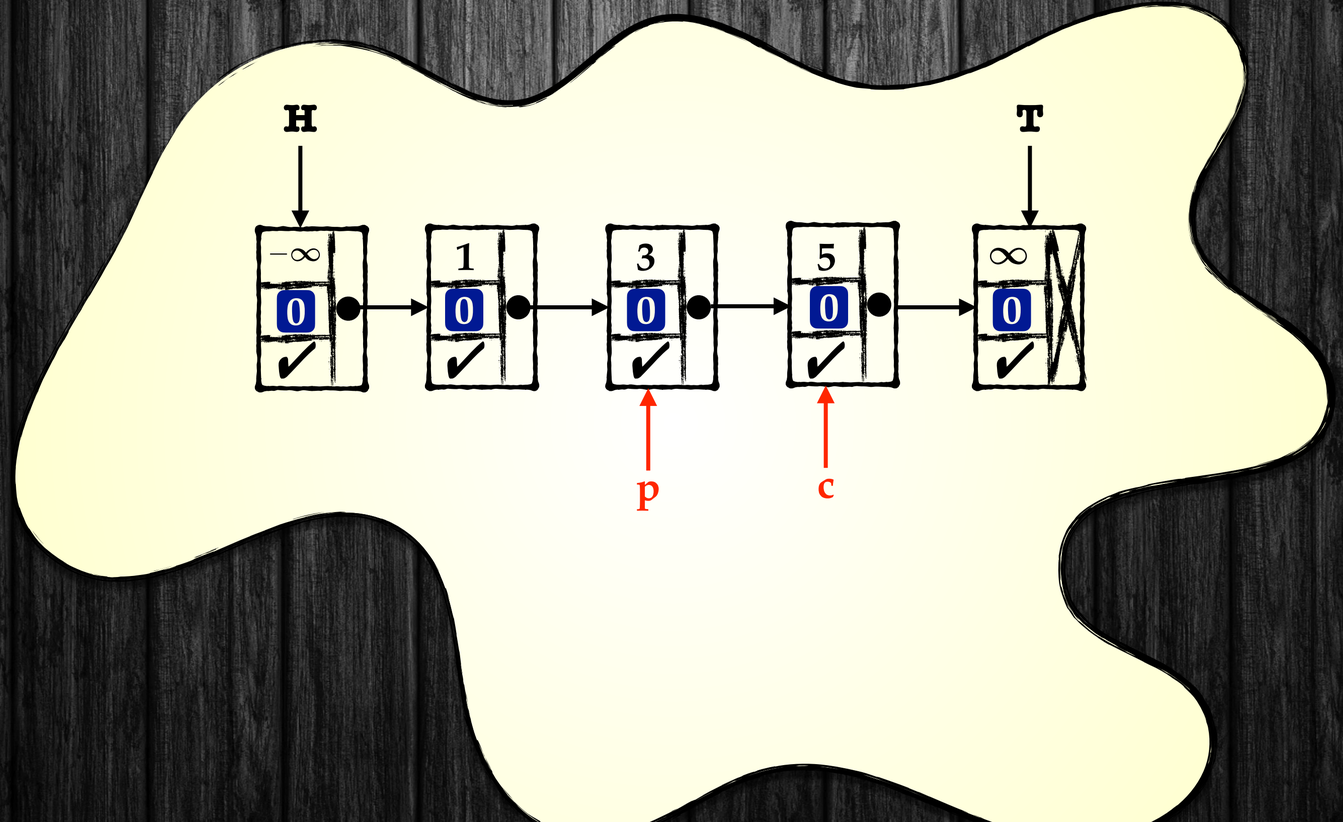




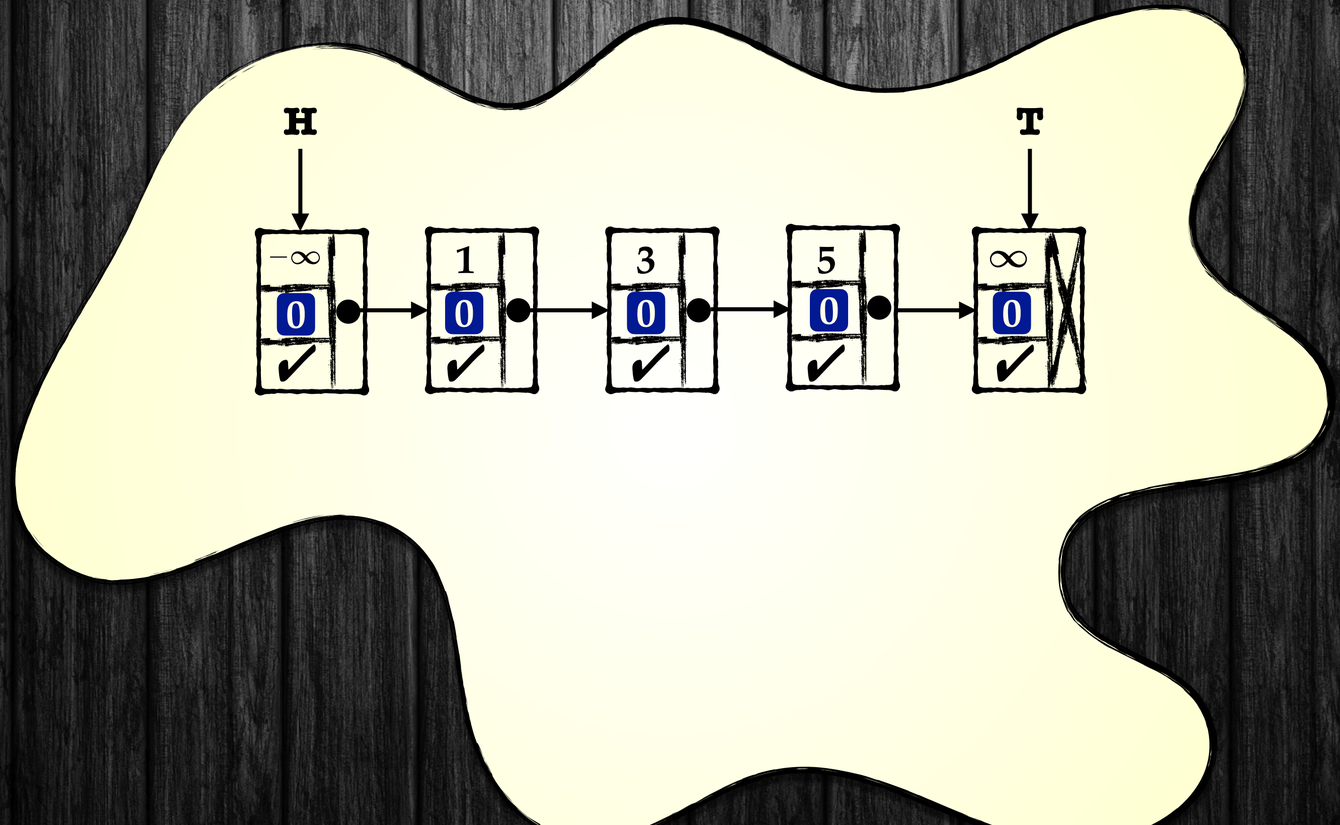
```
rmv(e):
local p, c, n, r
1  (p,c) := locate(e);
2  if (c.val = e)
3     c.mark := true
4     n := c.next;
5     p.next := n;
6     r := true;
7  else r := false;
8  unlock(p);
9  unlock(c);
10 return r;
```

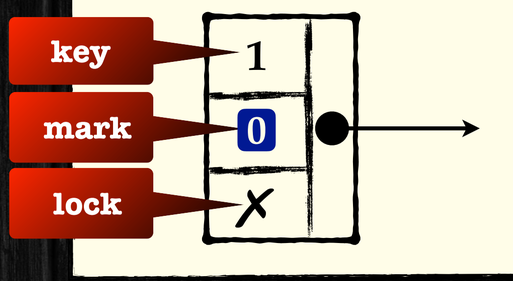
rmv(4)



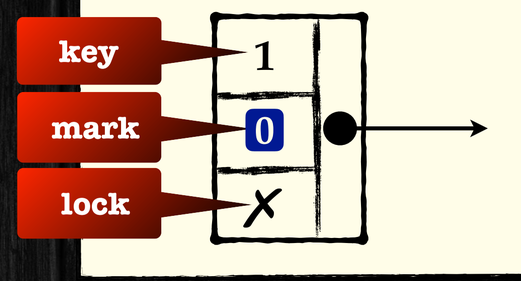


```
ctn(e):
local c, b
1  c := Head;
2  while (c.val < e)
3     c := c.next
4  b := c.mark
5  if (!b && c.val = e)
6     return true;
7  else return false;
```

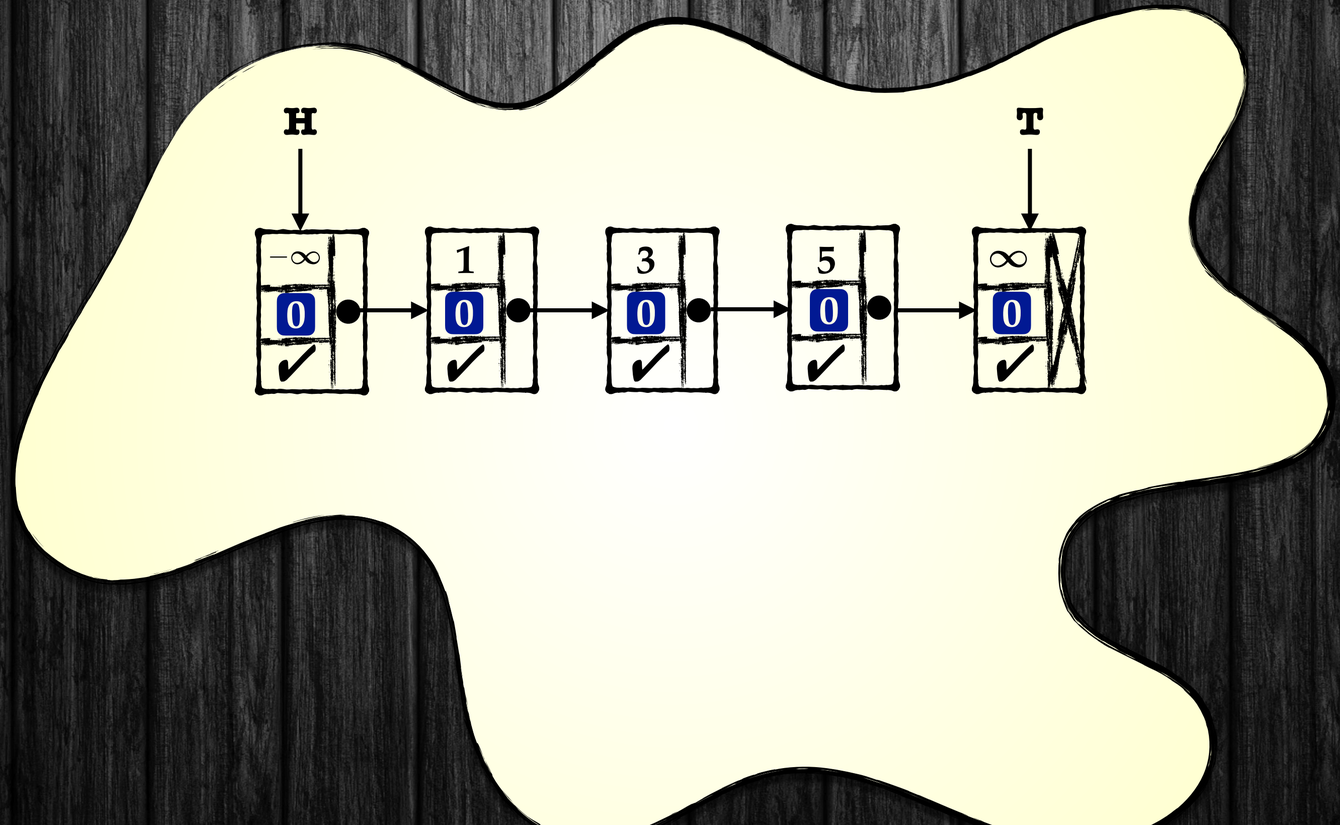




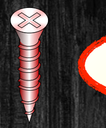
```
ctn(e):
local c, b
1  c := Head;
2  while (c.val < e)
3     c := c.next
4  b := c.mark
5  if (!b && c.val = e)
6     return true;
7  else return false;
```



no locks acquired

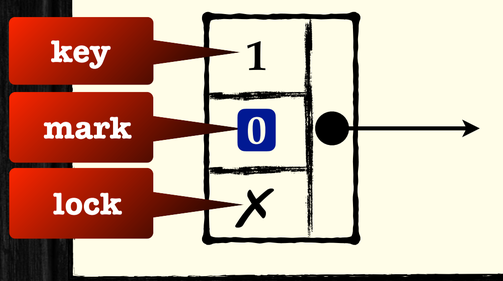


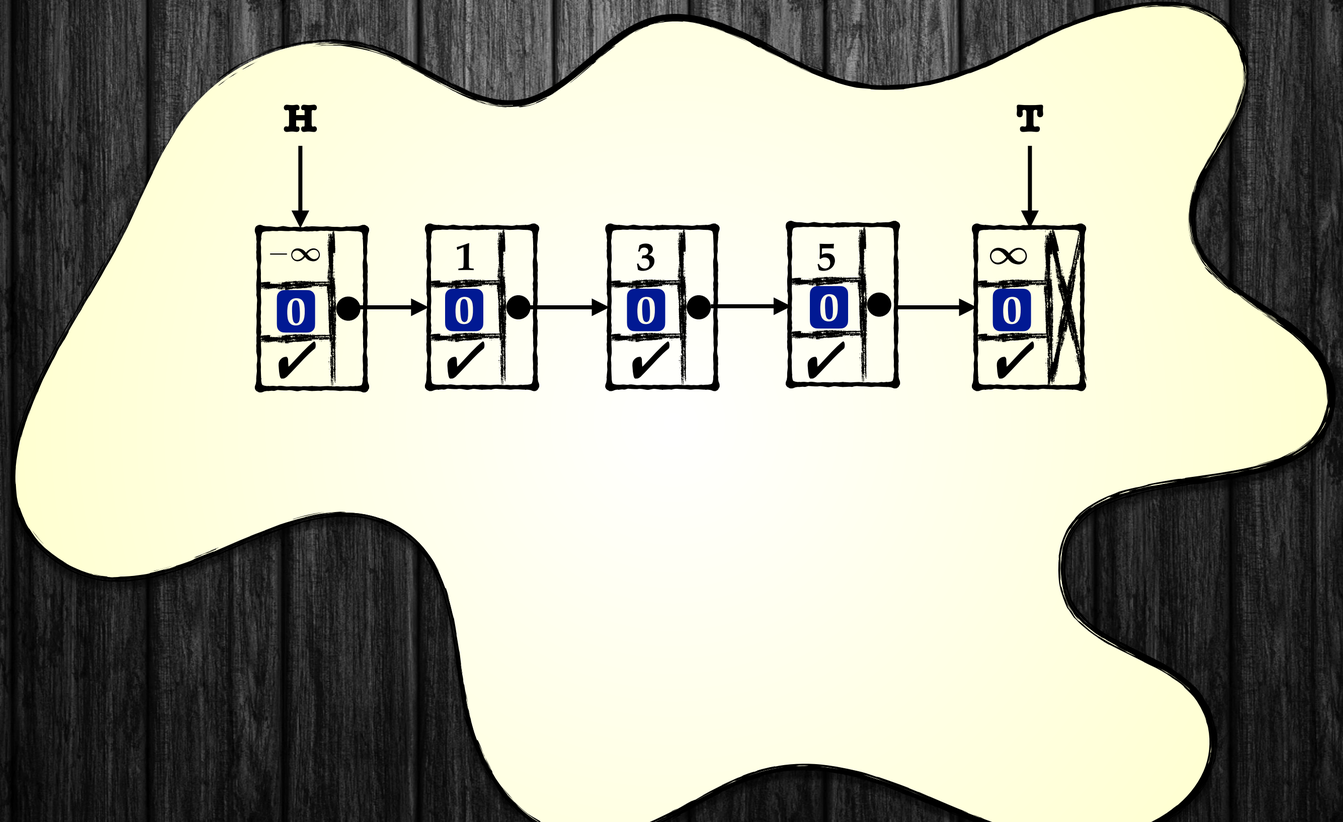
```
ctn(e):
local c, b
1  c := Head;
2  while (c.val < e)
3     c := c.next
4  b := c.mark
5  if (!b && c.val = e)
6     return true;
7  else return false;
```



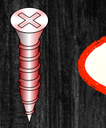
ctn(3)

no locks acquired



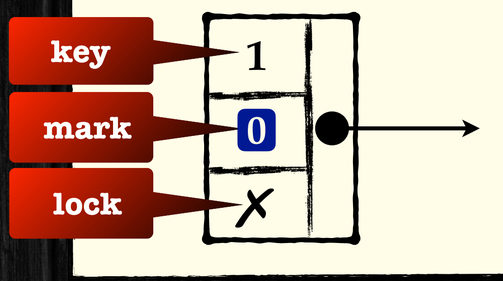


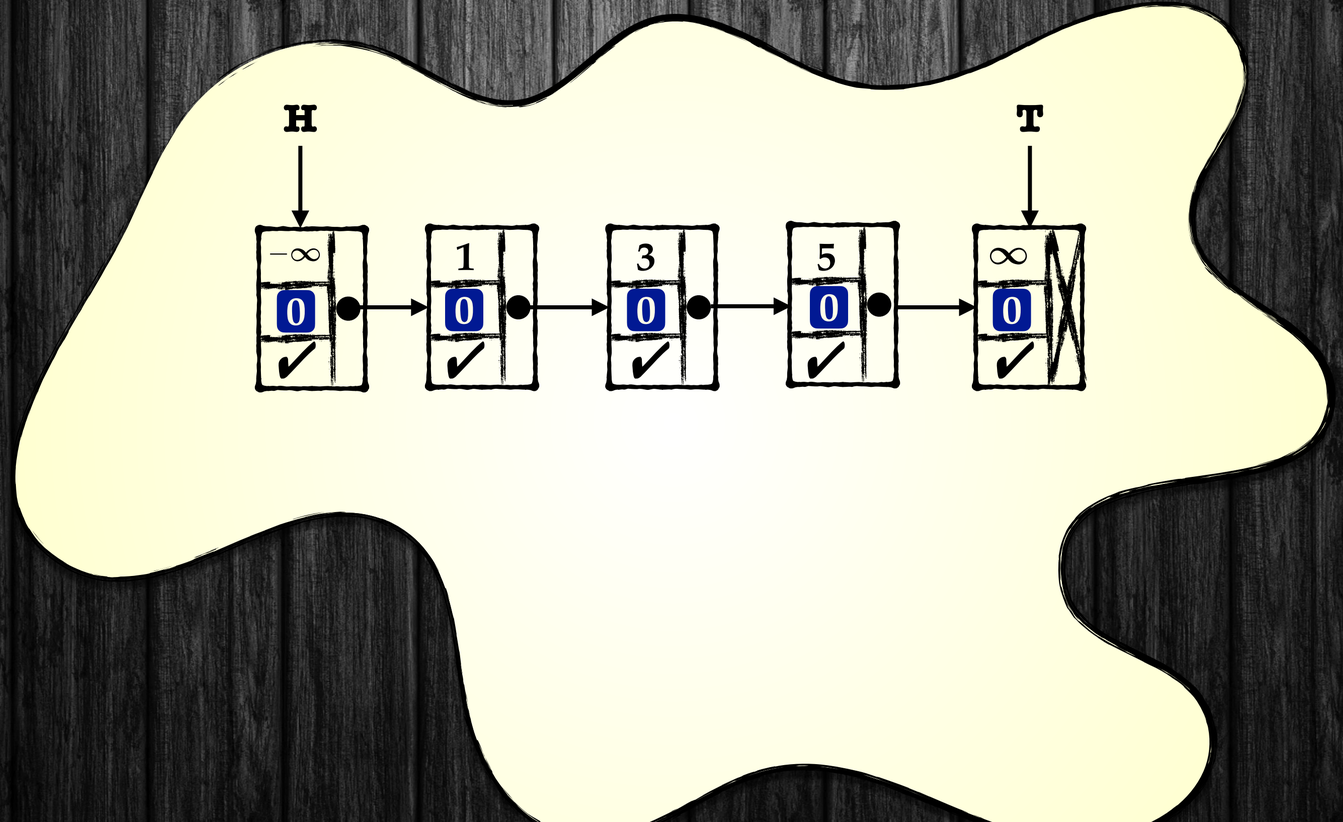
```
ctn(e):
local c, b
1  c := Head;
2  while (c.val < e)
3     c := c.next
4  b := c.mark
5  if (!b && c.val = e)
6     return true;
7  else return false;
```



ctn(3)

no locks acquired

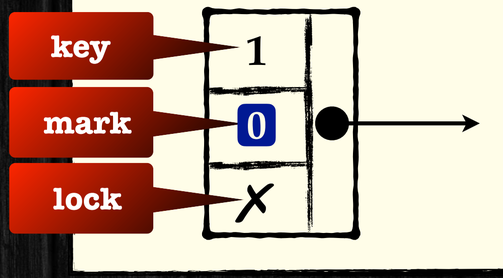


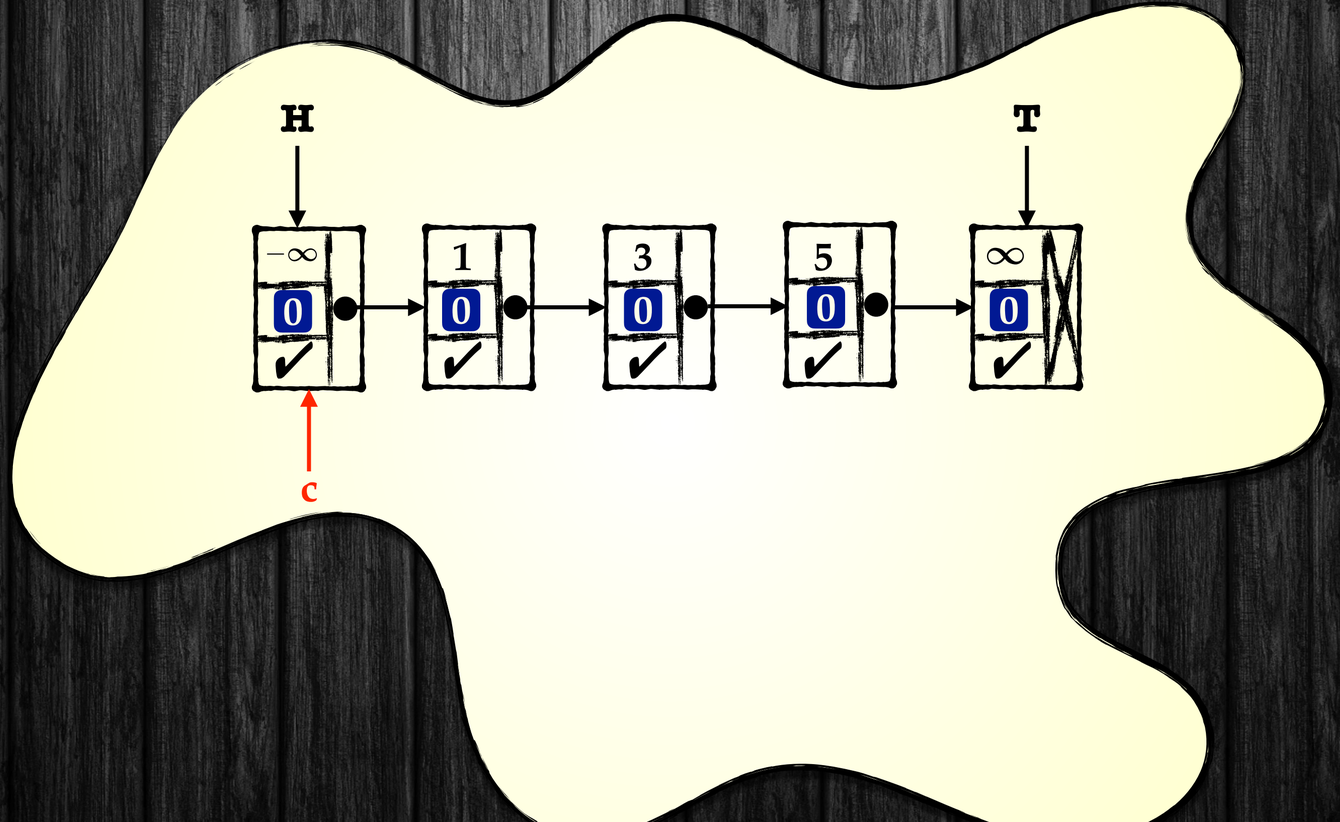


Sets | Lazy Synchronization

```
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;
```

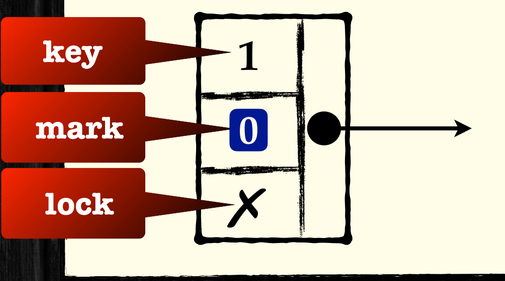


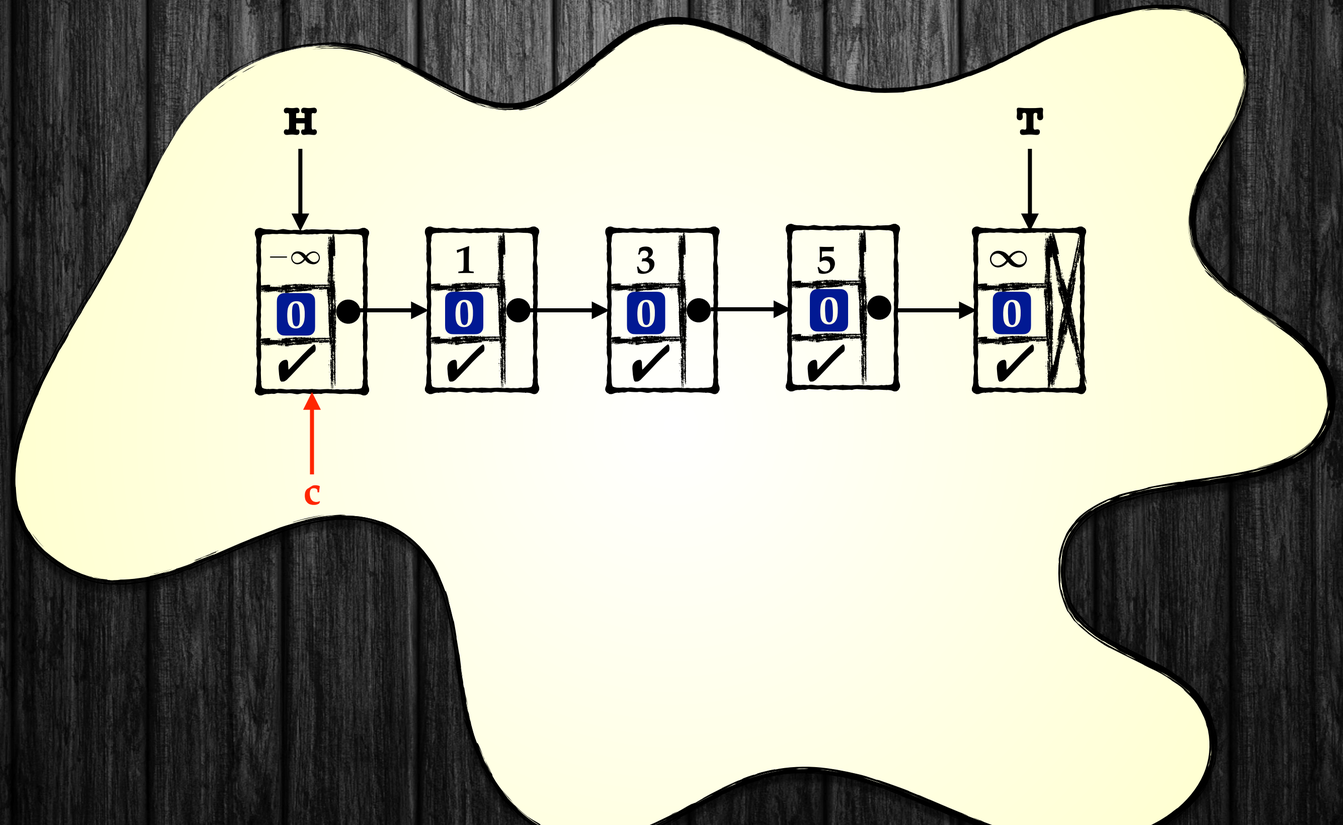




```
ctn(e):
local c, b
1  c := Head;
2  while (c.val < e)
3     c := c.next
4  b := c.mark
5  if (!b && c.val = e)
6     return true;
7  else return false;
```

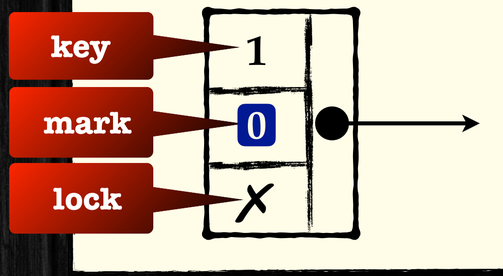
ctn(3)

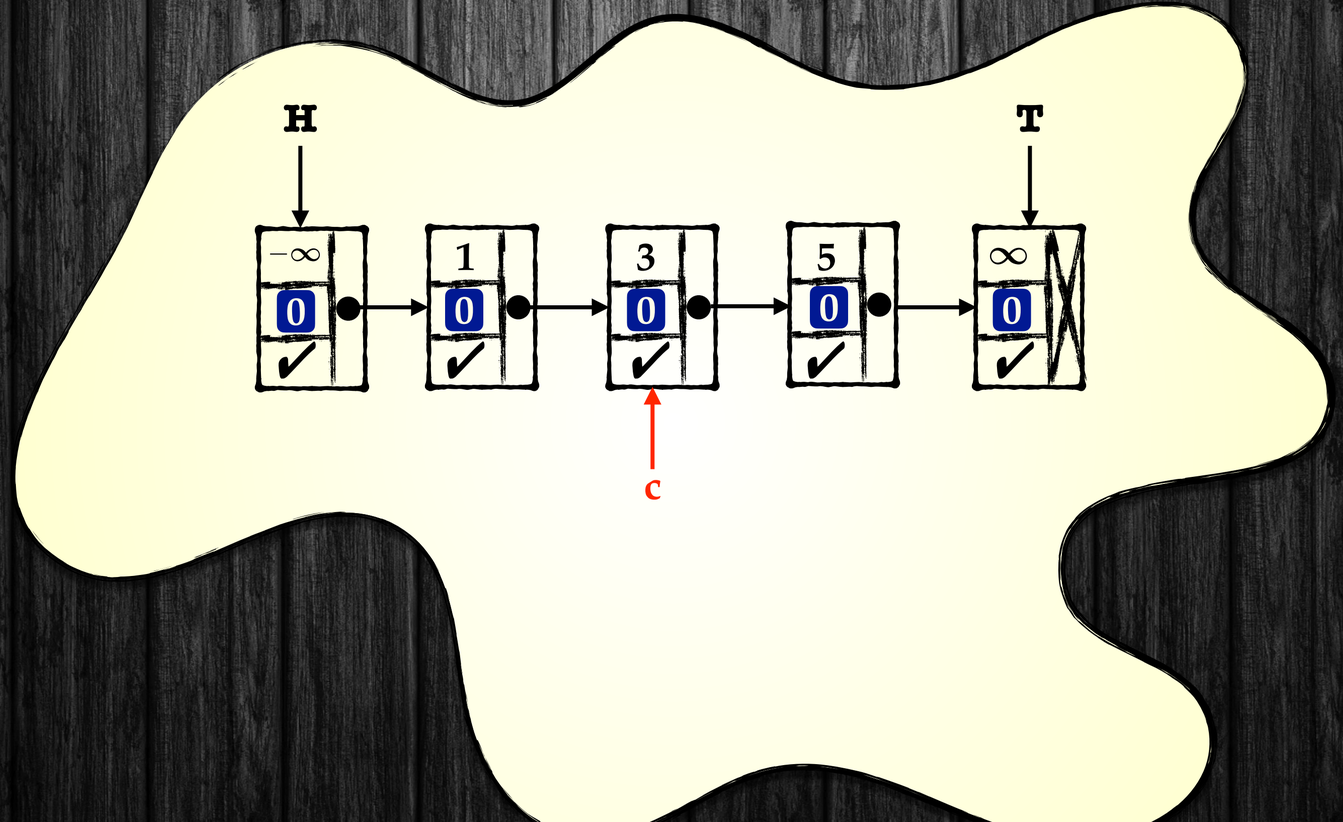




```
ctn(e):
local c, b
1  c := Head;
2  while (c.val < e)
3     c := c.next
4  b := c.mark
5  if (!b && c.val = e)
6     return true;
7  else return false;
```

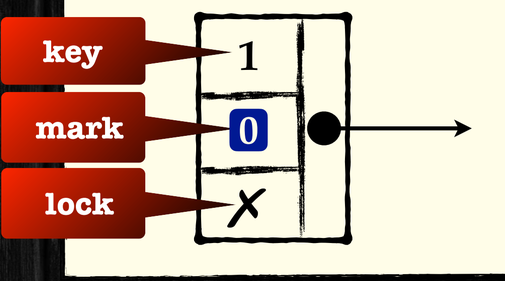


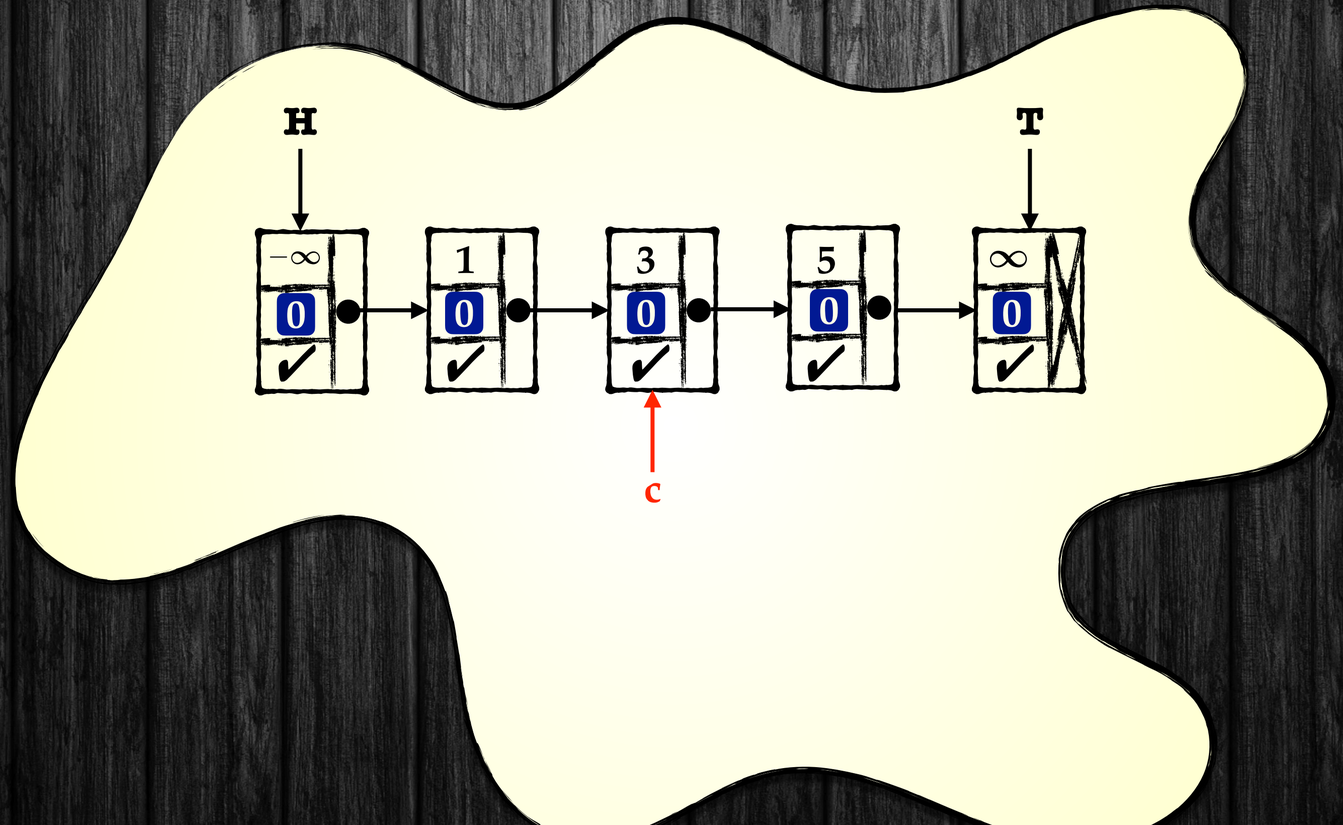




```
ctn(e):
local c, b
1  c := Head;
2  while (c.val < e)
3     c := c.next
4  b := c.mark
5  if (!b && c.val = e)
6     return true;
7  else return false;
```

ctn(3)

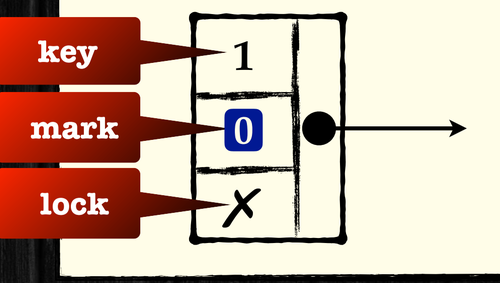


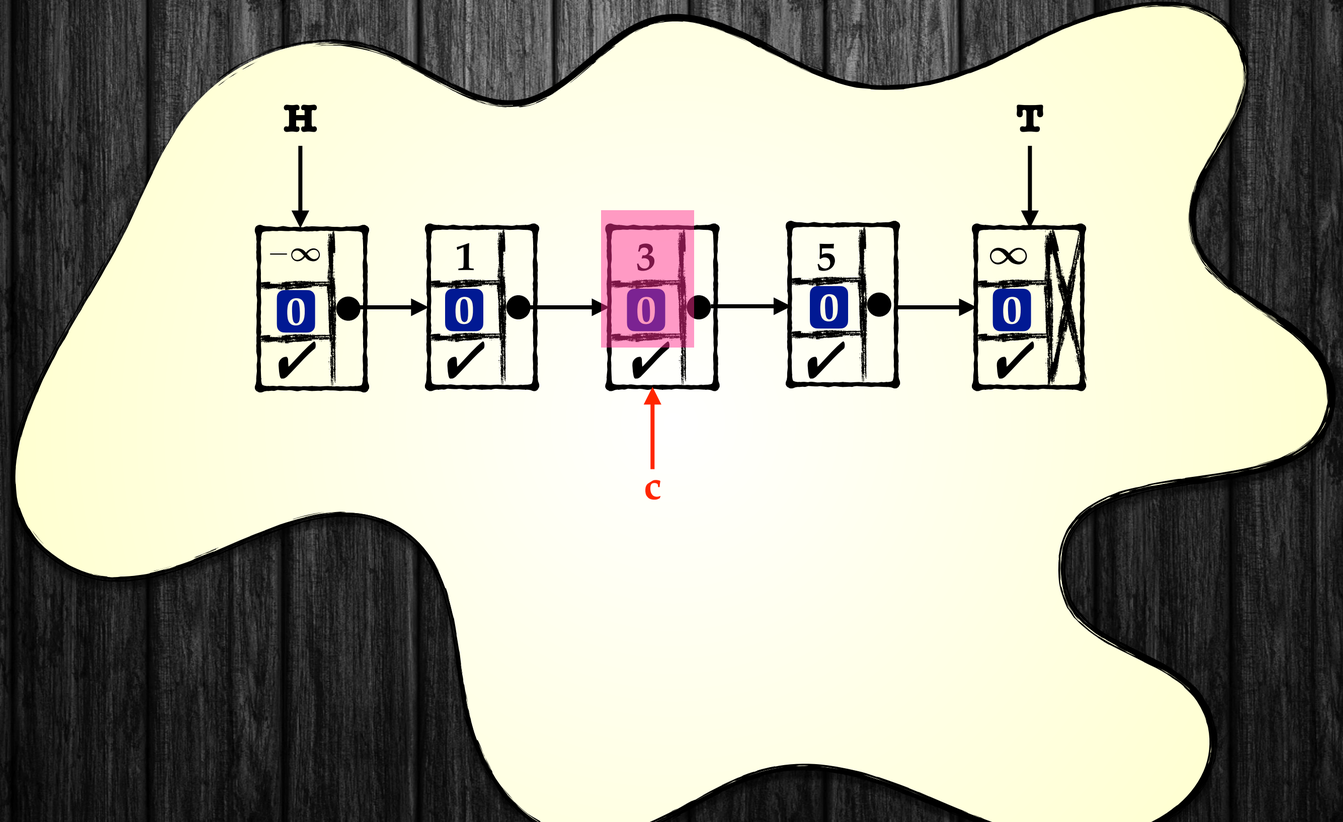


Sets | Lazy Synchronization

```
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)



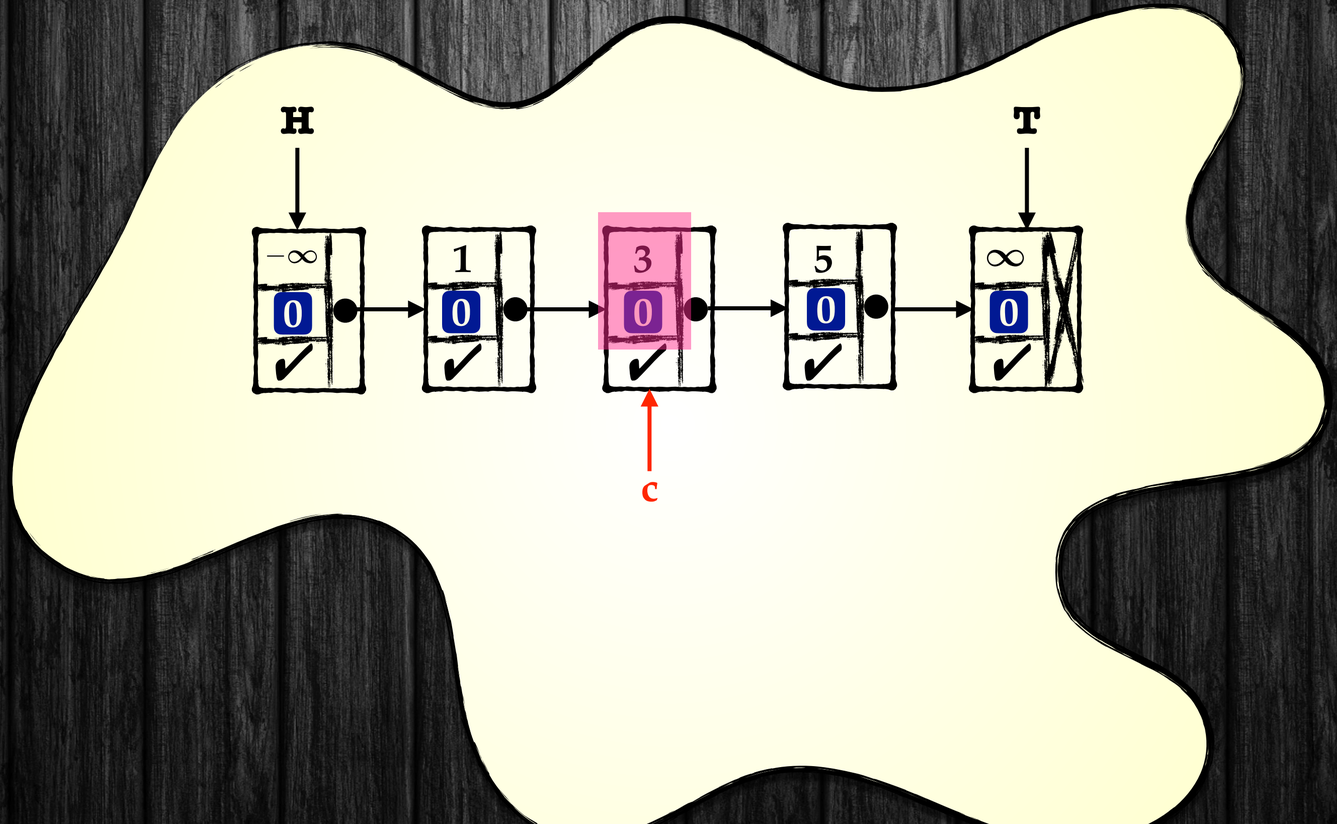


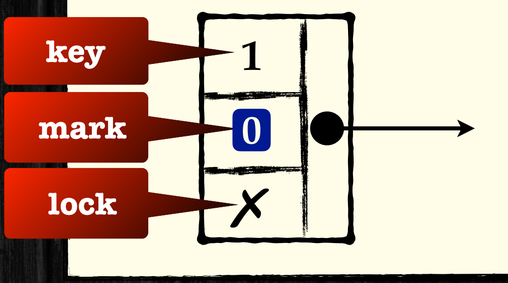
Sets | Lazy Synchronization

```
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

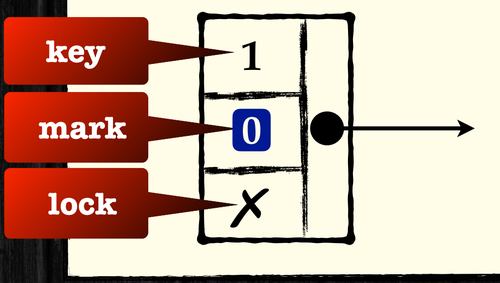




Sets | Lazy Synchronization

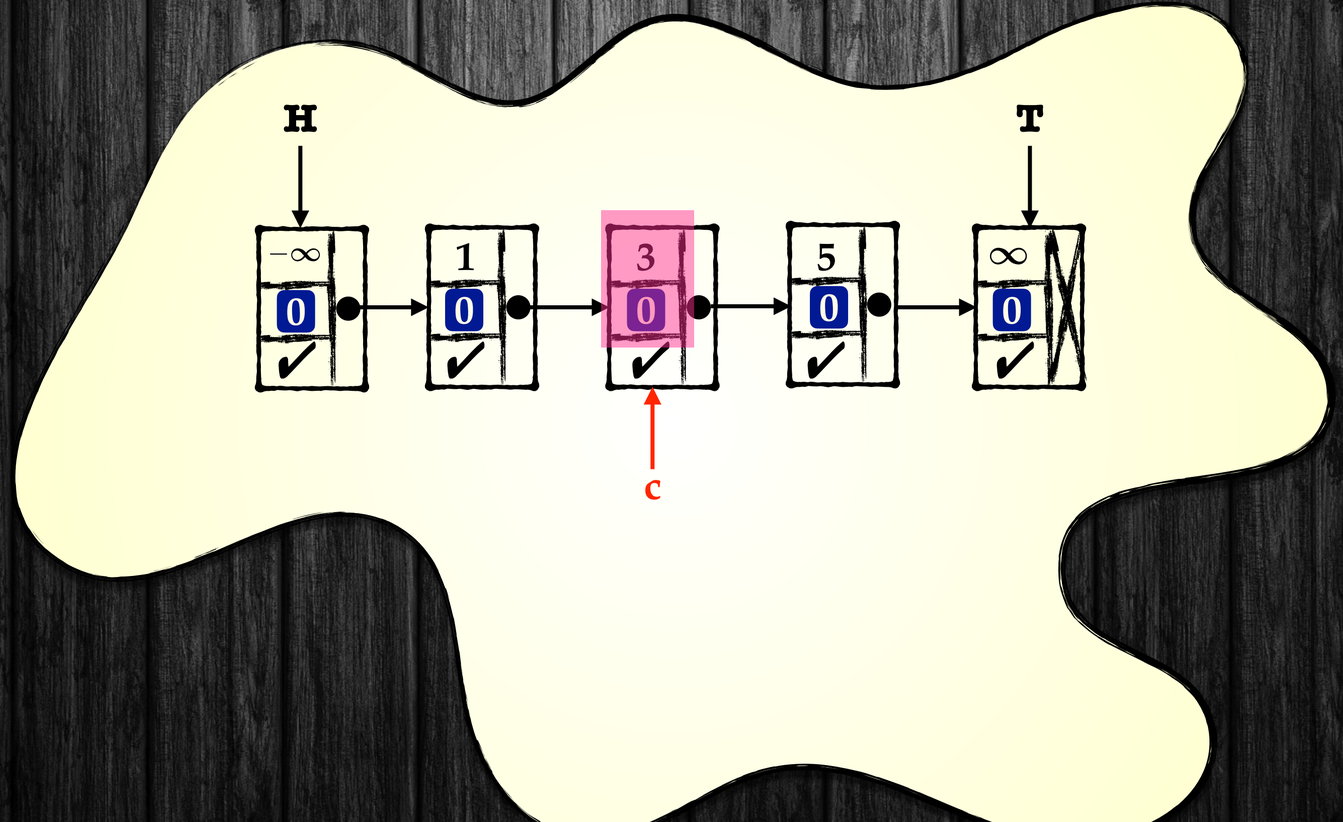
```
ctn(e):
local c, b
1  c := Head;
2  while (c.val < e)
3     c := c.next
4  b := c.mark
5  if (!b && c.val = e)
6     return true;
7  else return false;
```





successful contains

linearization point



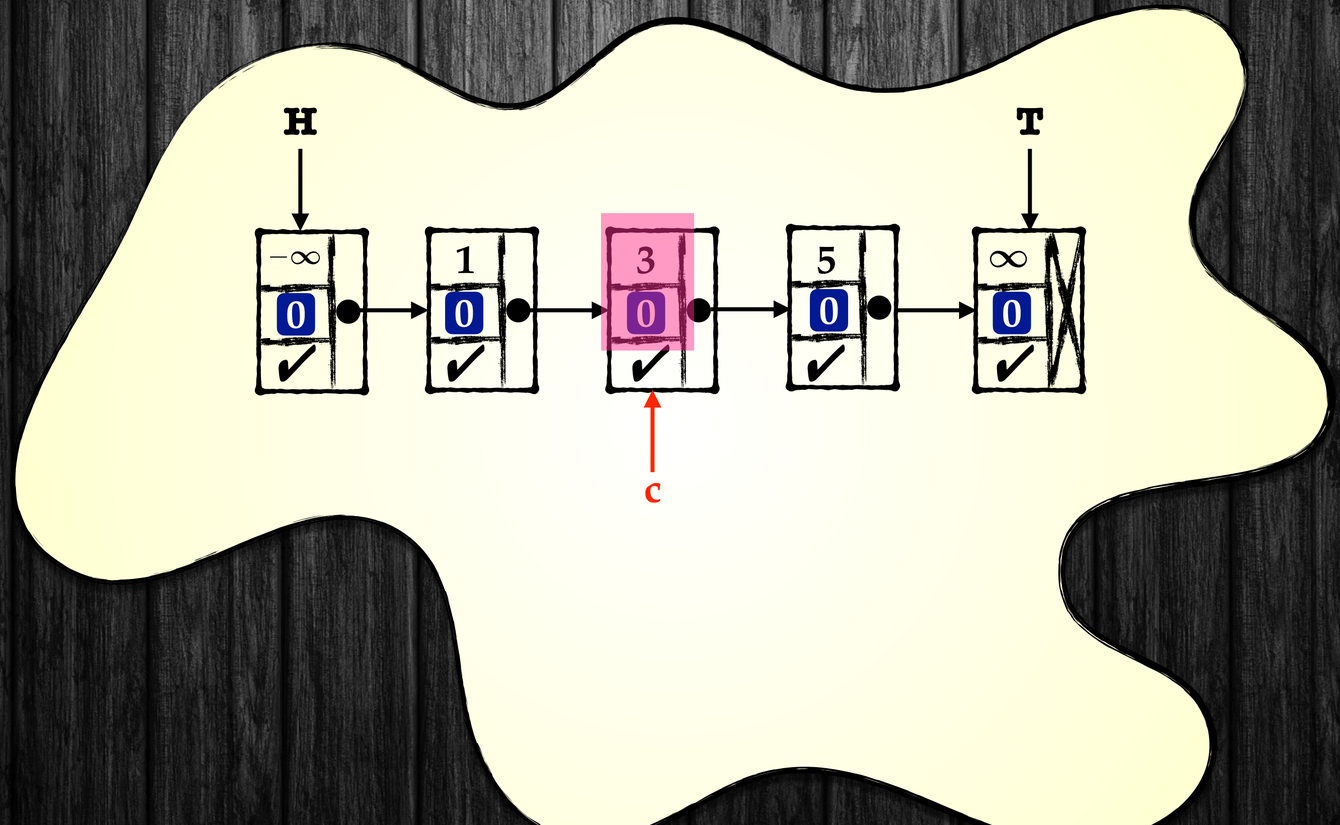
Sets | Lazy Synchronization

```
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;
```

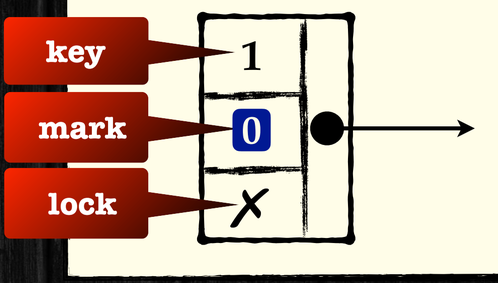
"unmarked entry with input key"

successful contains

linearization point







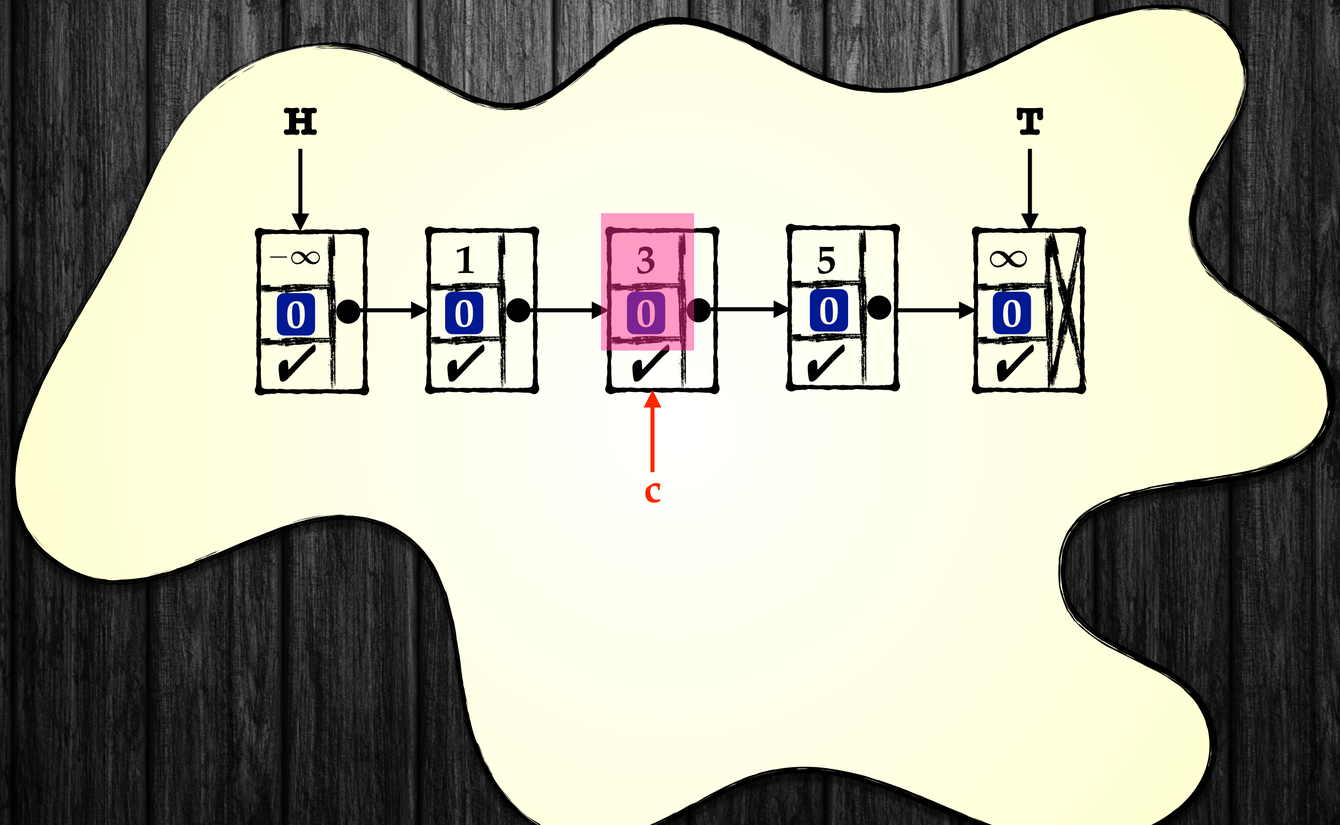
Sets | Lazy Synchronization

```
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;
```

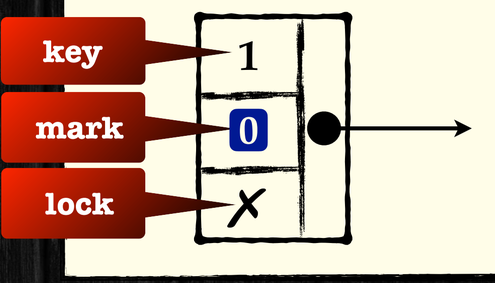
"unmarked entry with input key"

successful contains

linearization point



ctn(3)



Sets | Lazy Synchronization

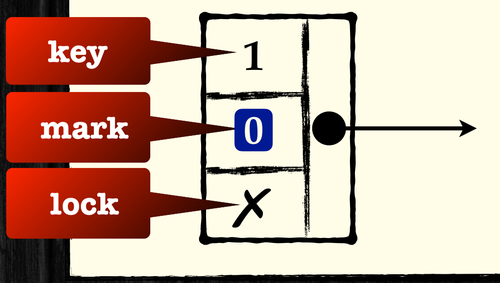
```
ctn(e):
local c, b
1  c := Head;
2  while (c.val < e)
3     c := c.next
4  b := c.mark
5  if (!b && c.val = e)
6     return true;
7  else return false;
```

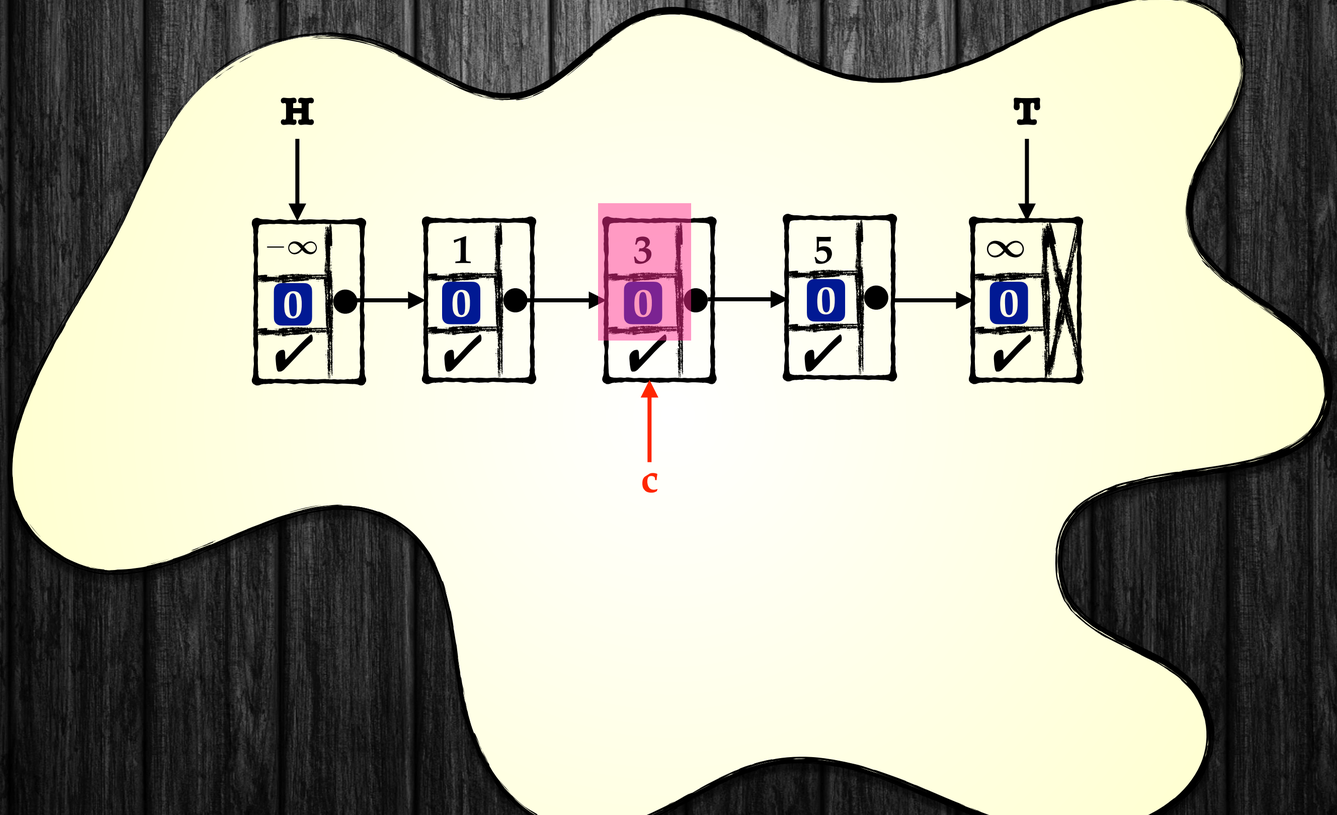
"unmarked entry with input key"

successful contains

linearization point



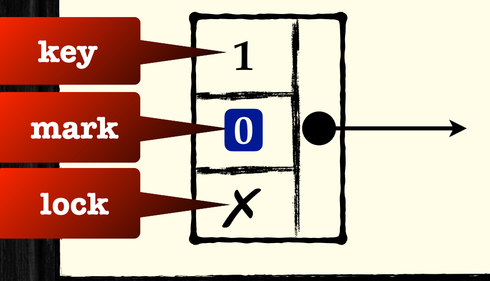


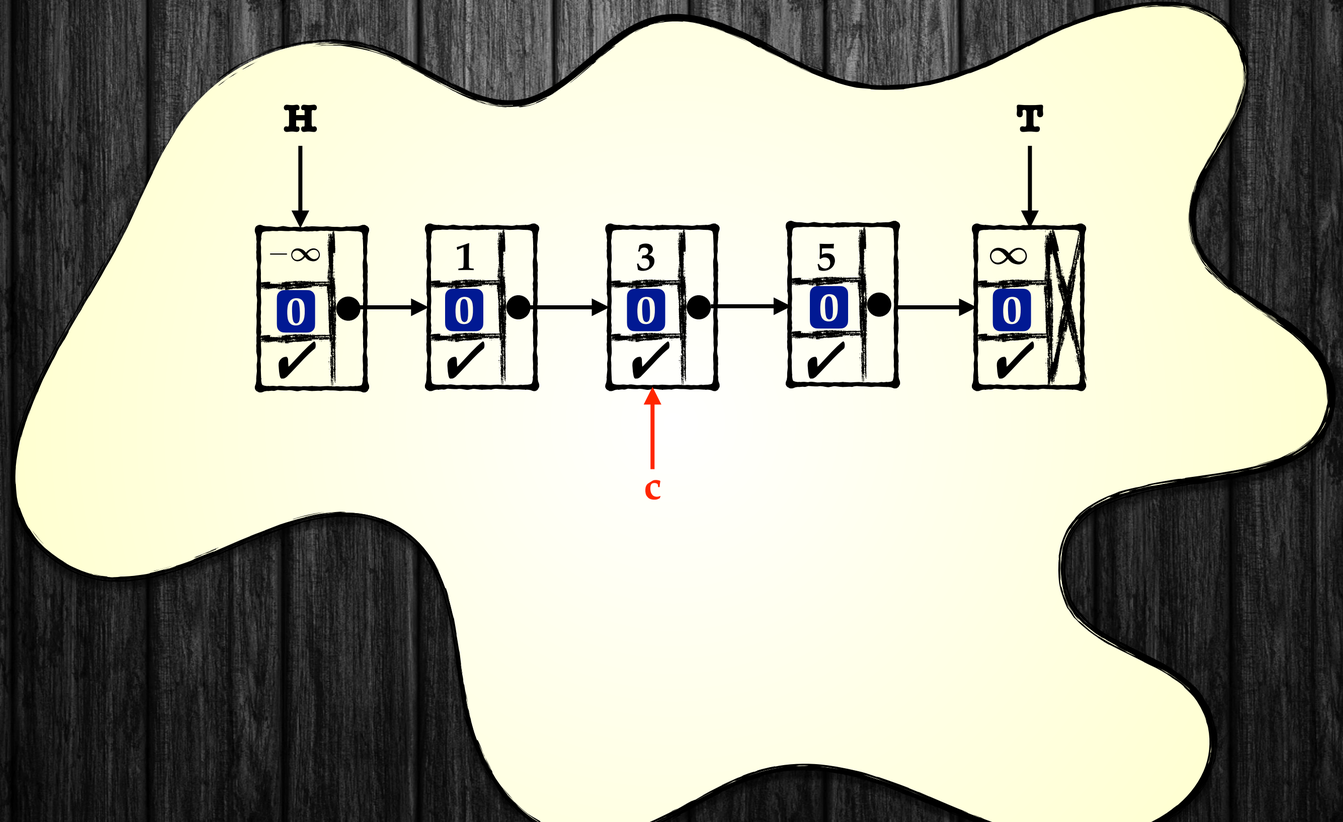


Sets | Lazy Synchronization

```
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)

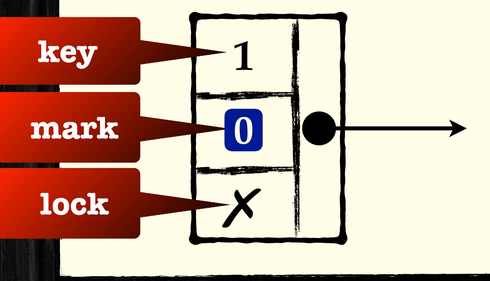


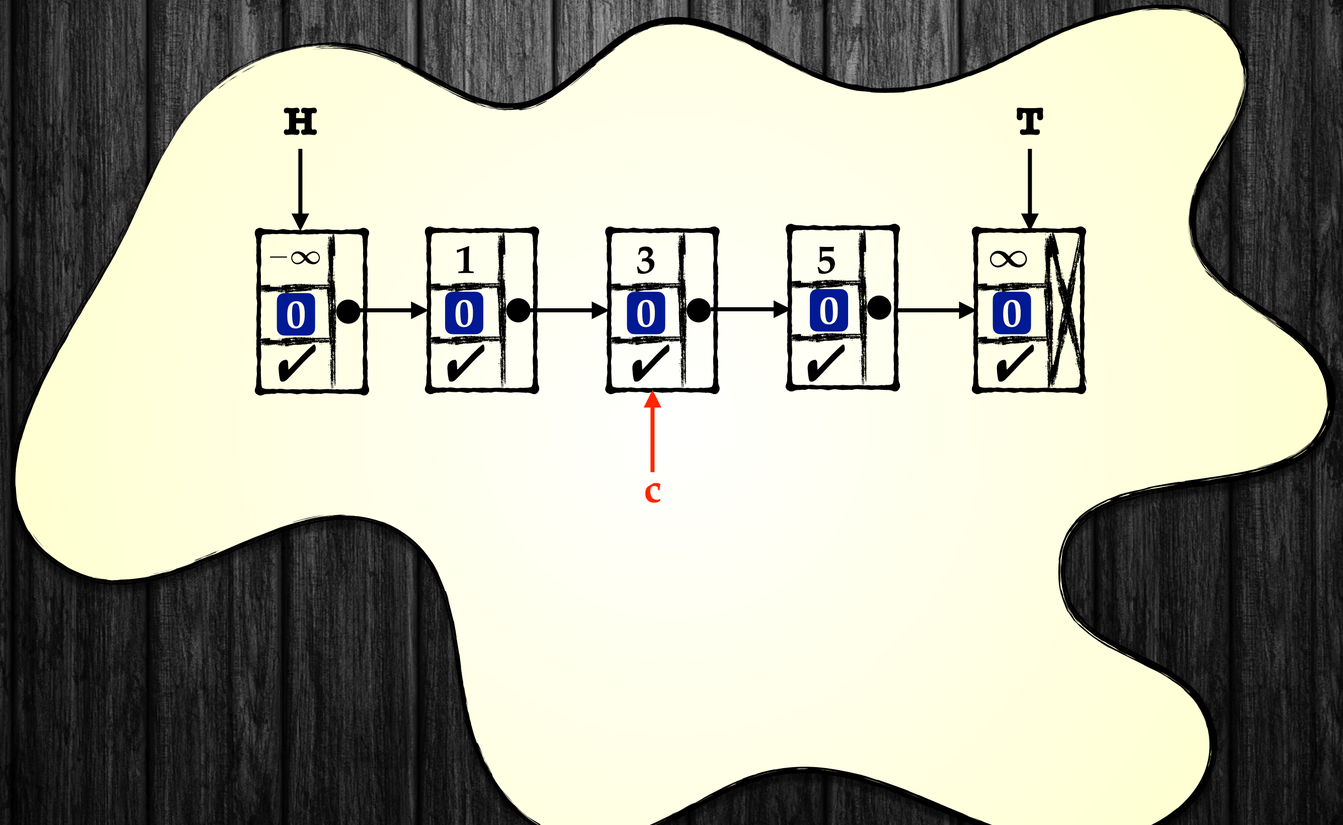


Sets | Lazy Synchronization

```
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;
```

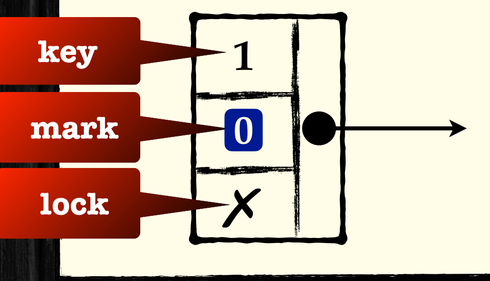
ctn(3)

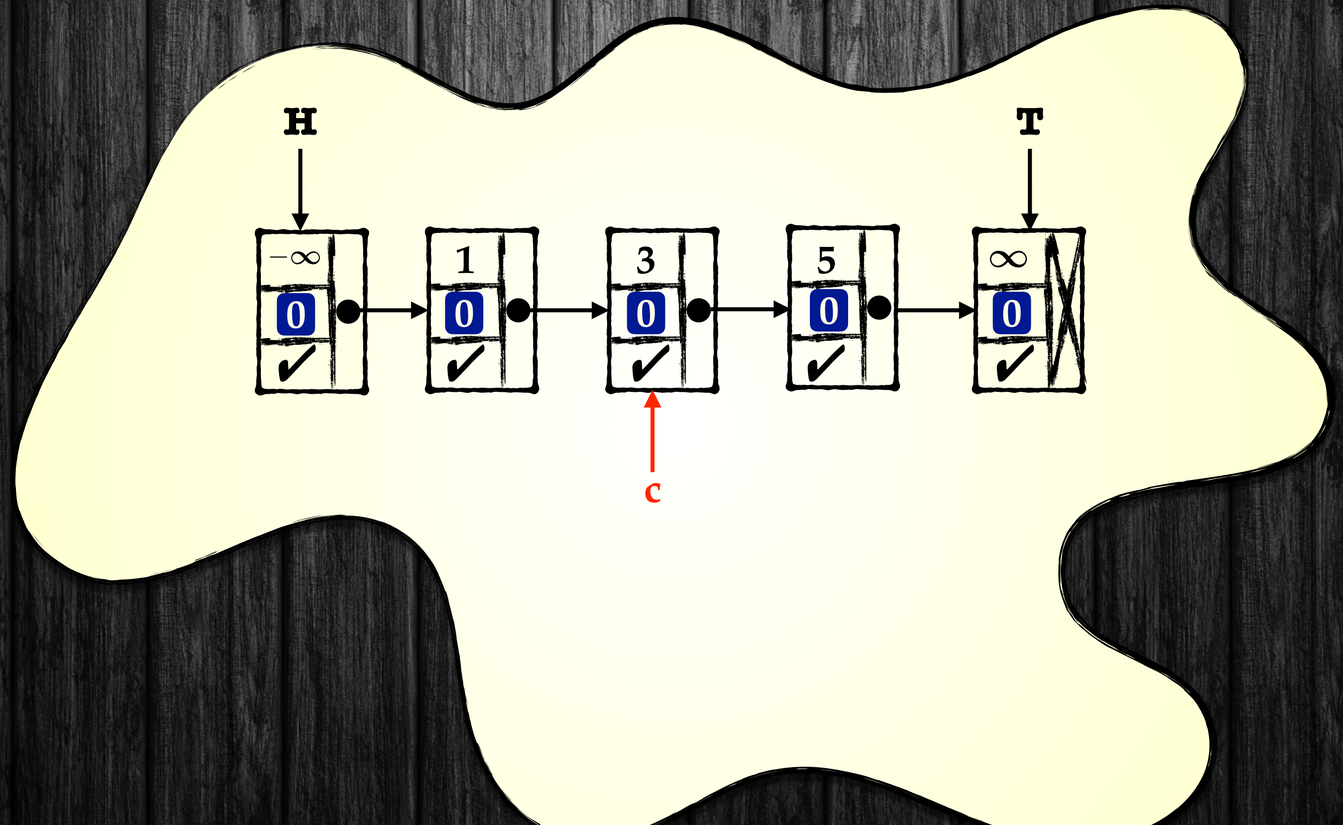




```
ctn(e):
local c, b
1  c := Head;
2  while (c.val < e)
3     c := c.next
4  b := c.mark
5  if (!b && c.val = e)
6     return true;
7  else return false;
```

ctn(3)

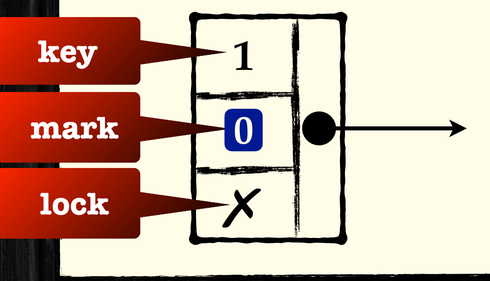


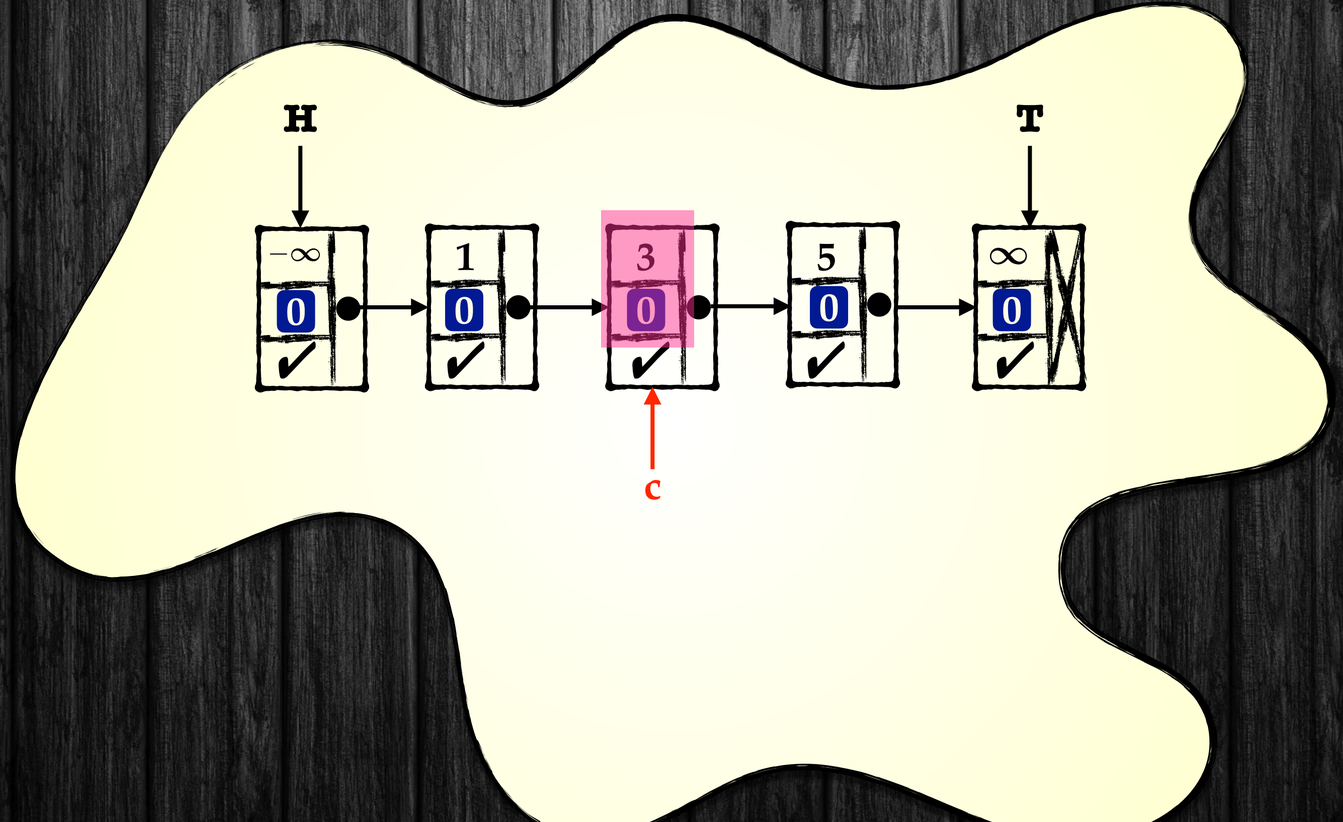


Sets | Lazy Synchronization

```
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;
```

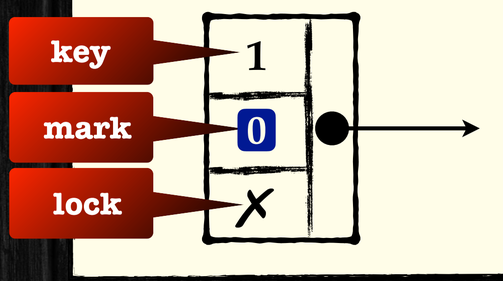
ctn(3)

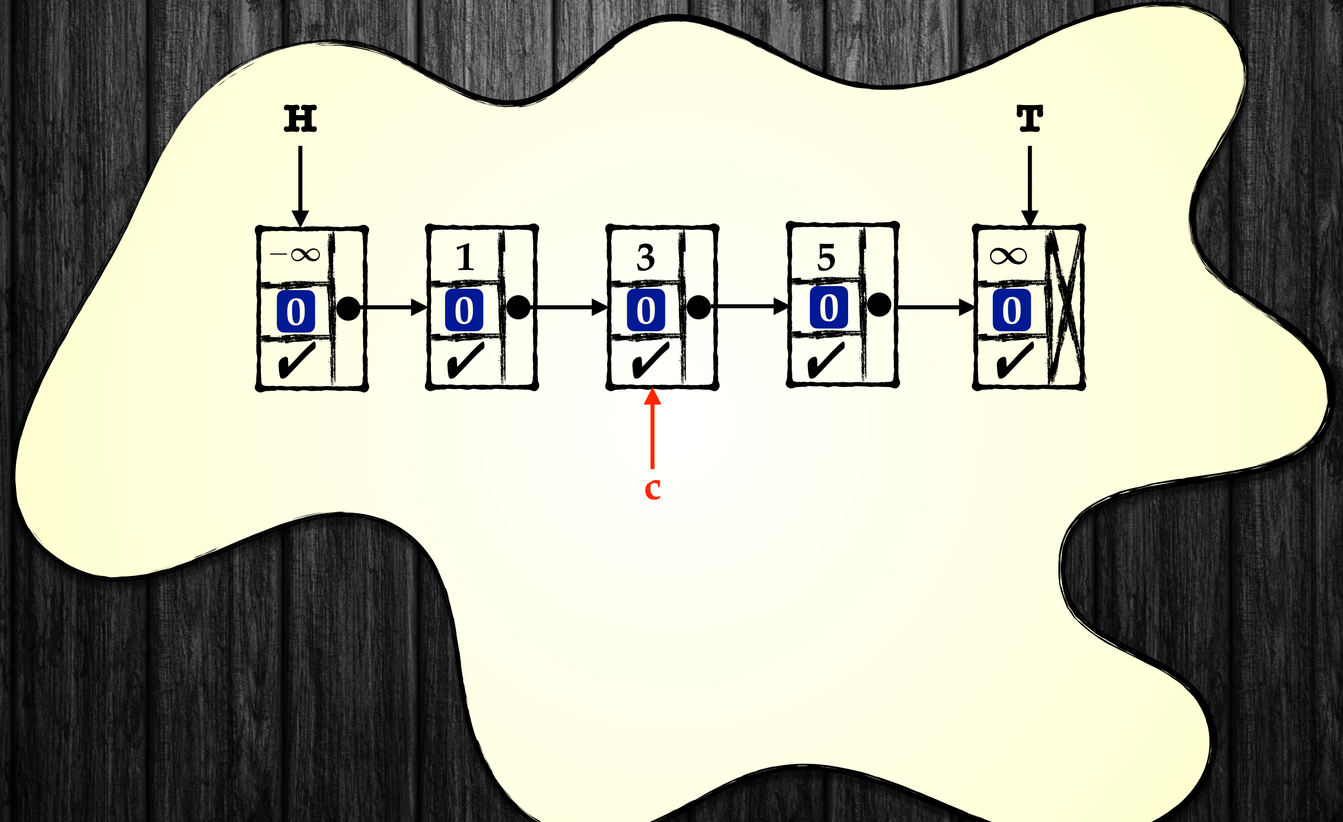




```
ctn(e):
local c, b
1  c := Head;
2  while (c.val < e)
3     c := c.next
4  b := c.mark
5  if (!b && c.val = e)
6     return true;
7  else return false;
```

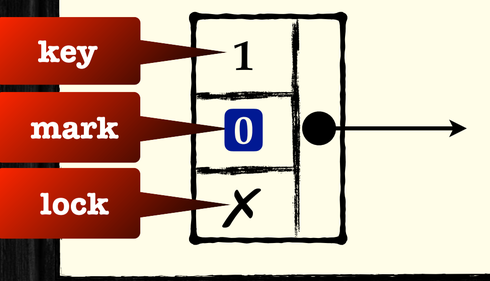


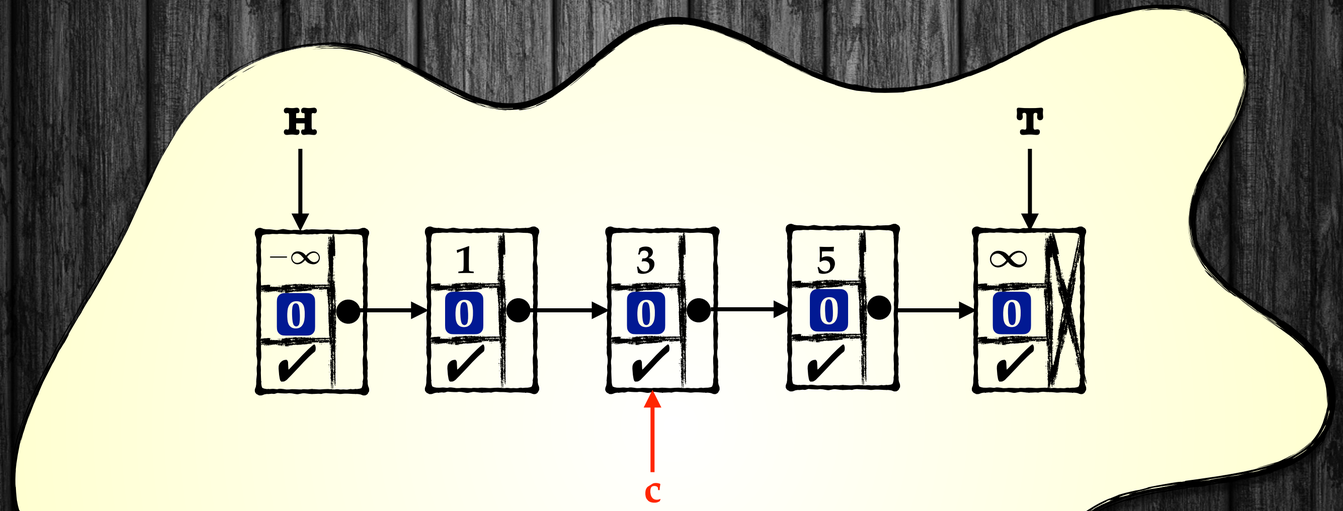




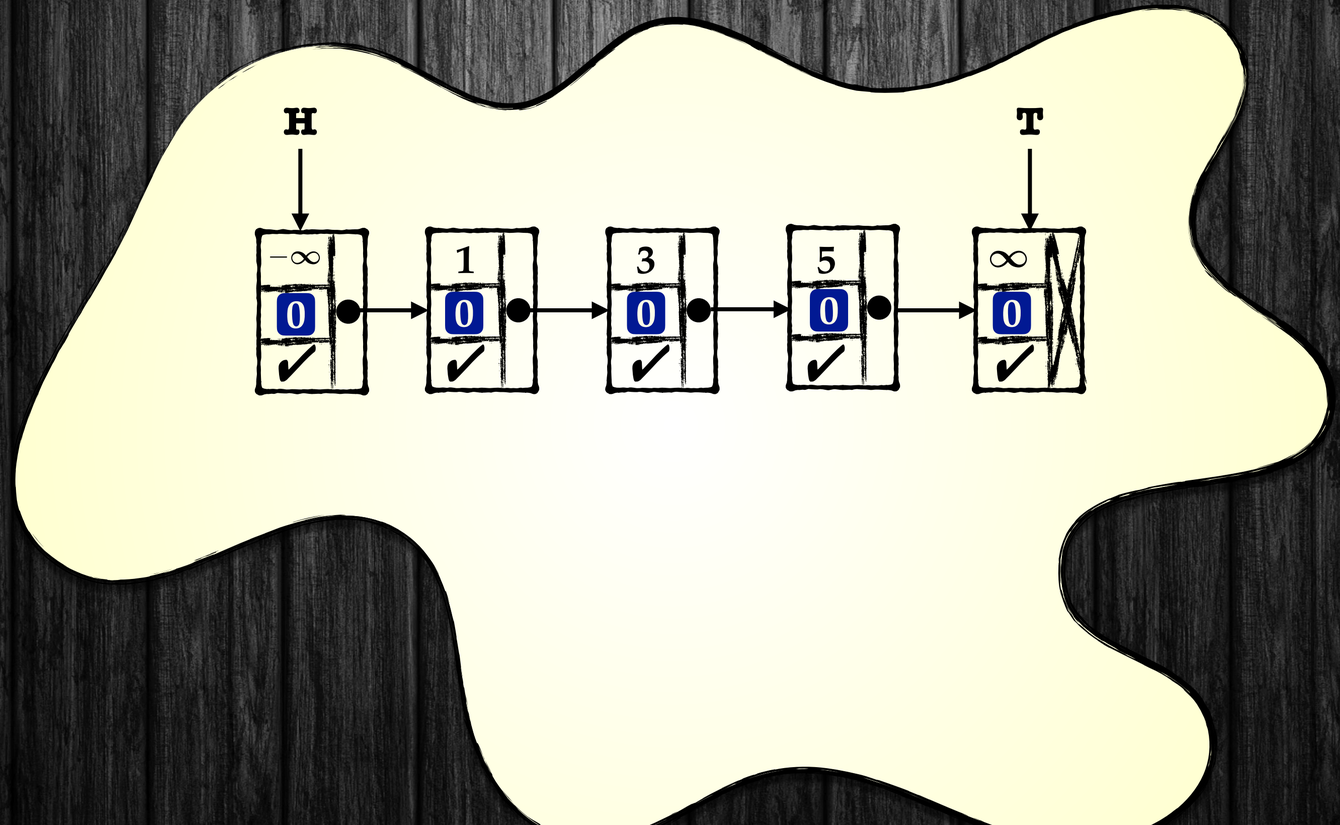
```
ctn(e):
local c, b
1  c := Head;
2  while (c.val < e)
3     c := c.next
4  b := c.mark
5  if (!b && c.val = e)
6     return true;
7  else return false;
```

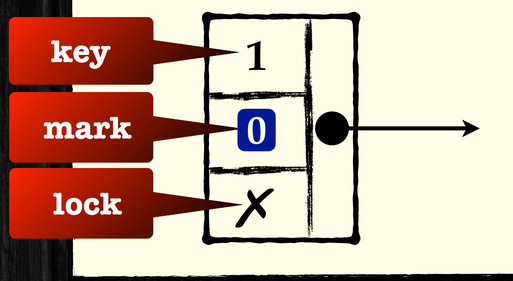
ctn(3)





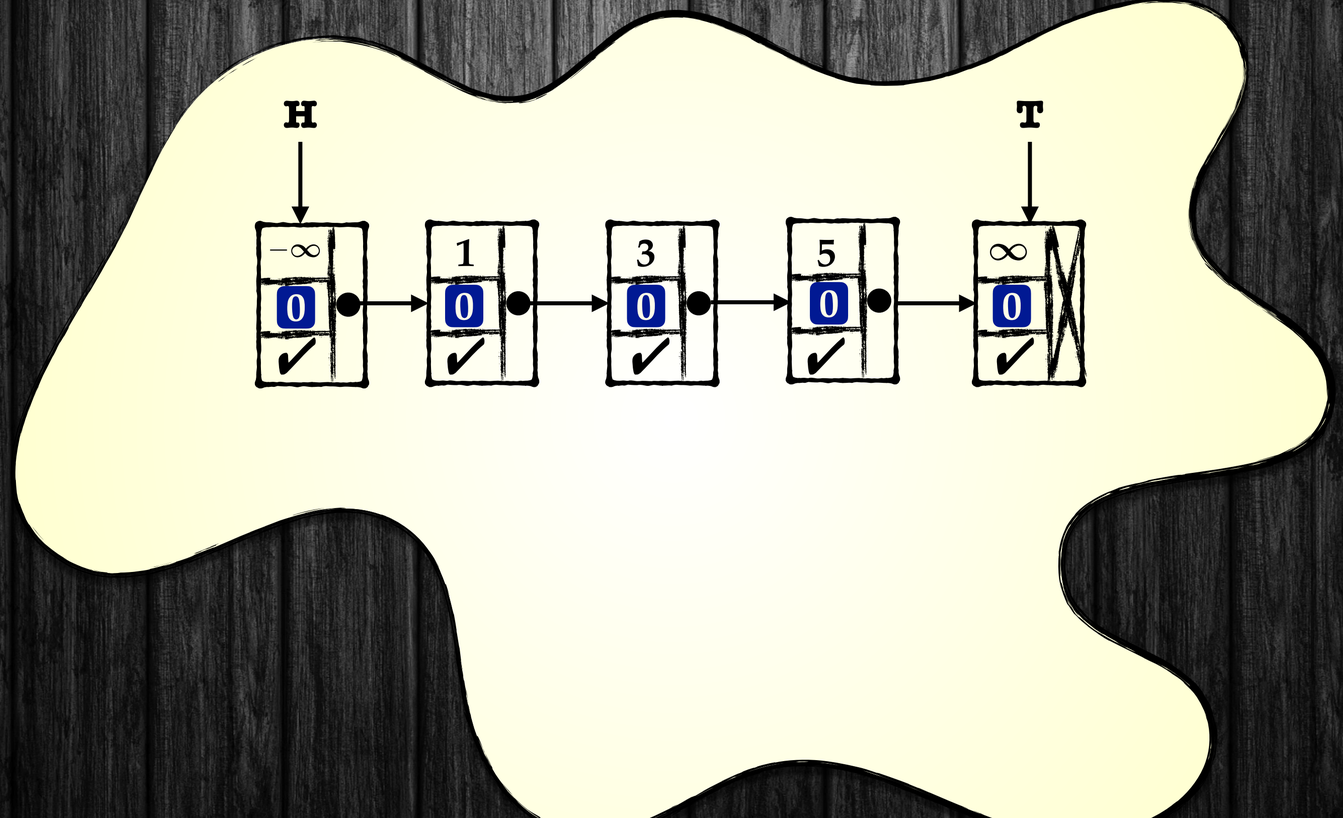
```
ctn(e):
local c, b
1  c := Head;
2  while (c.val < e)
3     c := c.next
4  b := c.mark
5  if (!b && c.val = e)
6     return true;
7  else return false;
```

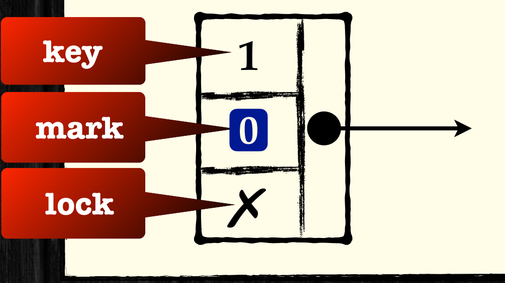




```
ctn(e):
local c, b
1  c := Head;
2  while (c.val < e)
3     c := c.next
4  b := c.mark
5  if (!b && c.val = e)
6     return true;
7  else return false;
```

unsuccessful contains

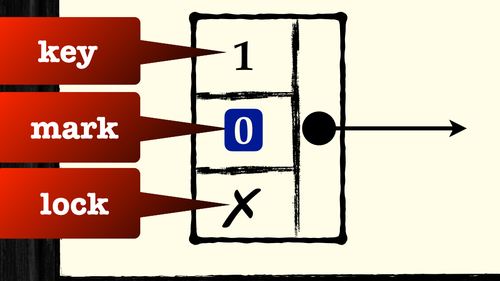


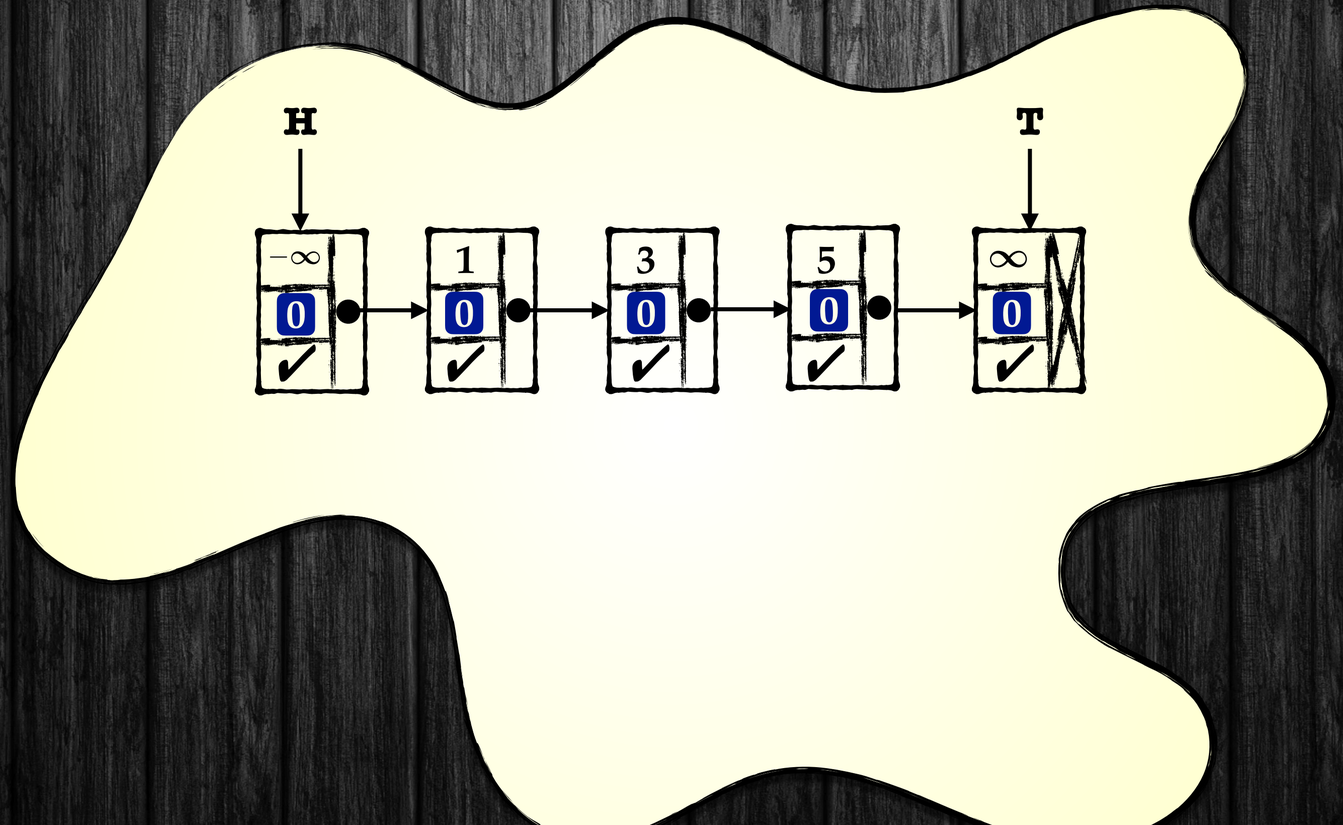


```
ctn(e):
local c, b
1  c := Head;
2  while (c.val < e)
3     c := c.next
4  b := c.mark
5  if (!b && c.val = e)
6     return true;
7  else return false;
```

unsuccessful contains

"complicated linearization policy"



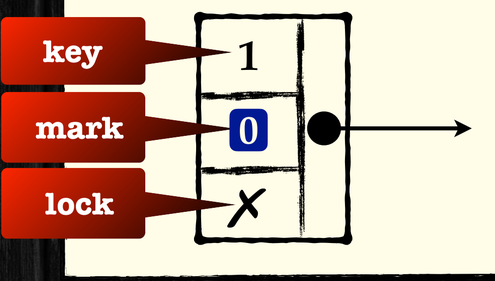


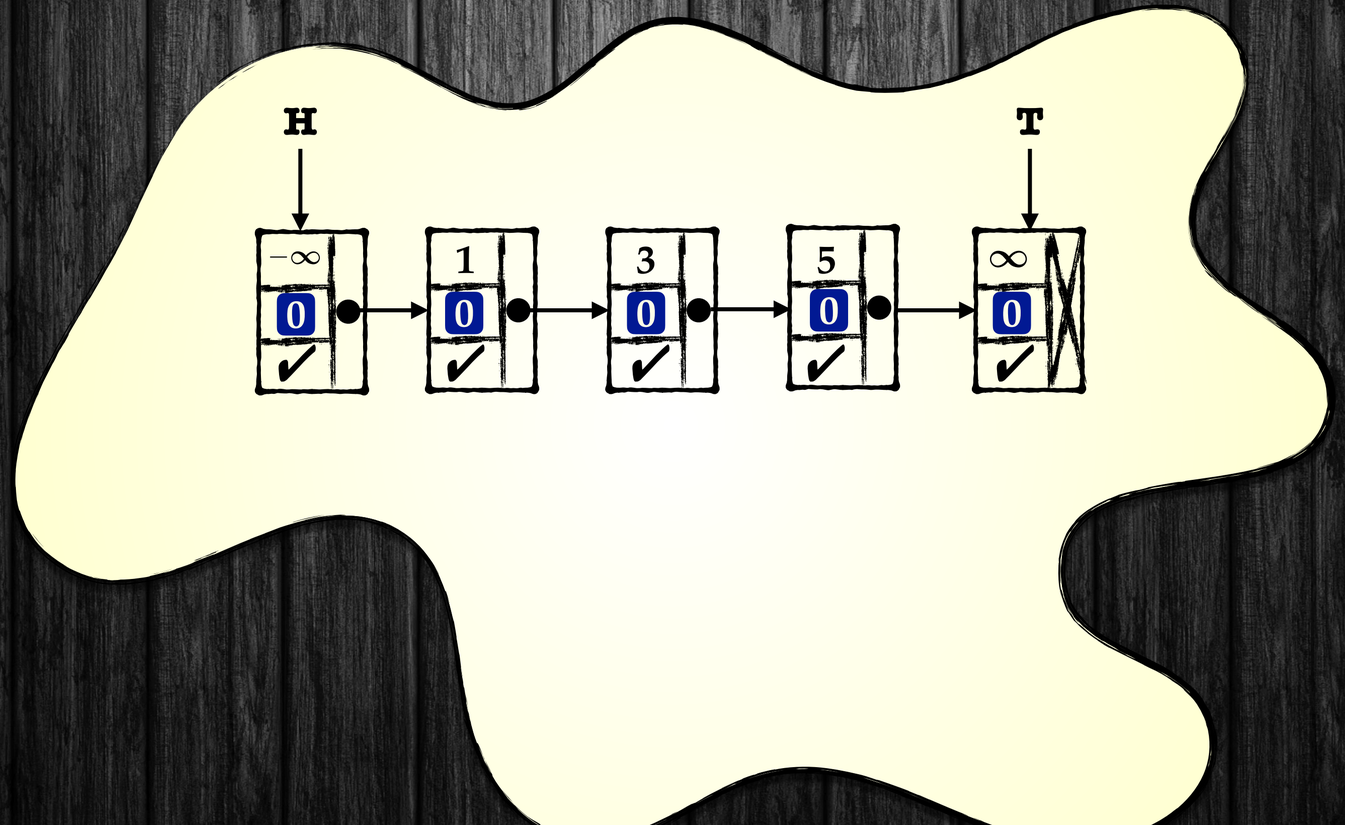
```
ctn(e):
local c, b
1  c := Head;
2  while (c.val < e)
3     c := c.next
4  b := c.mark
5  if (!b && c.val = e)
6     return true;
7  else return false;
```

unsuccessful contains

unsuccessful contains

“complicated linearization policy”





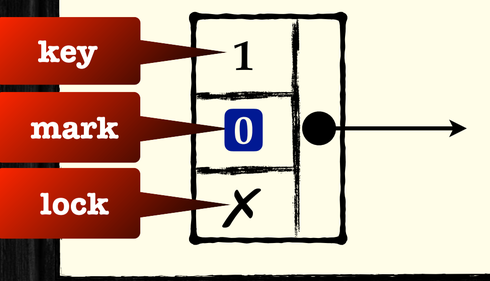
```
ctn(e):
local c, b
1  c := Head;
2  while (c.val < e)
3     c := c.next
4  b := c.mark
5  if (!b && c.val = e)
6     return true;
7  else return false;
```

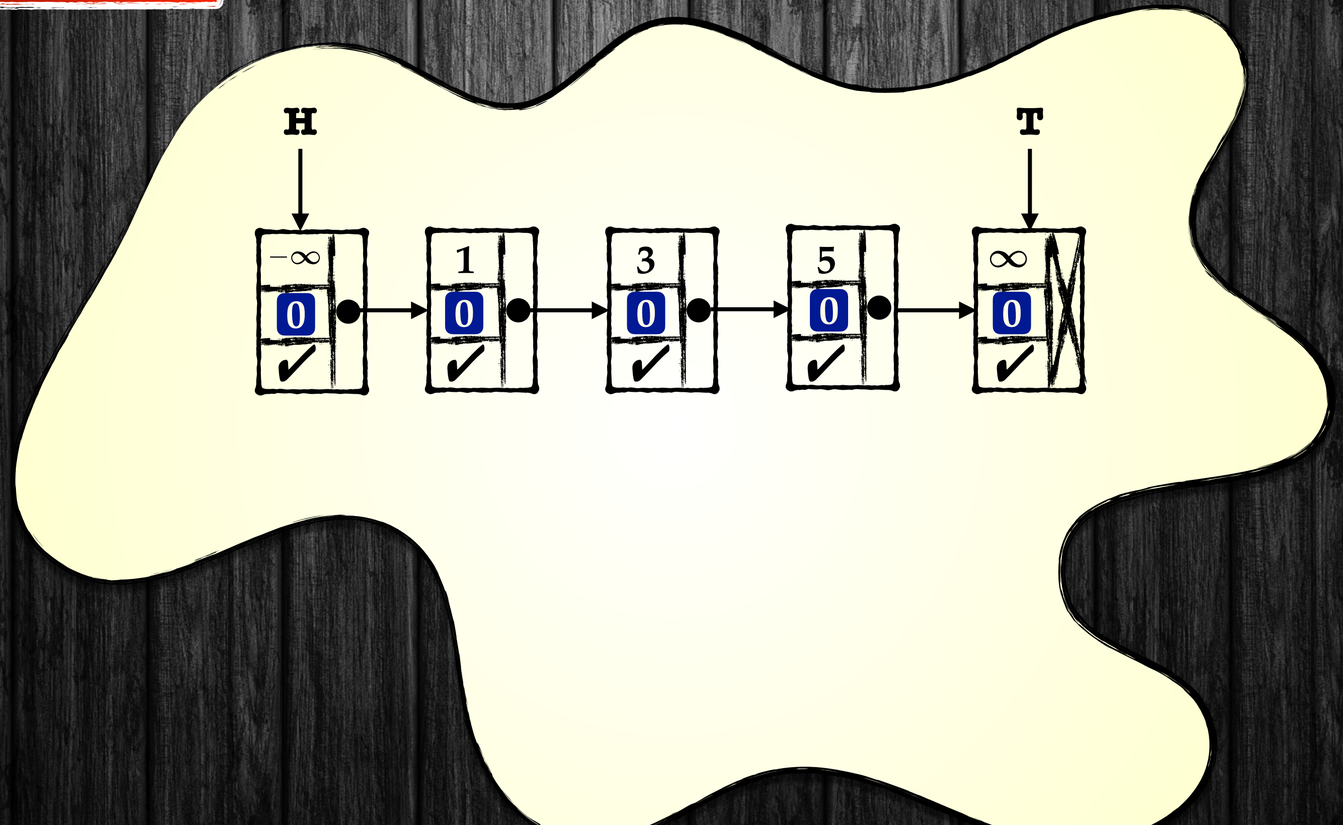
"when finding the cell with the input key marked"

unsuccessful contains

unsuccessful contains

"complicated linearization policy"





```
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;
```

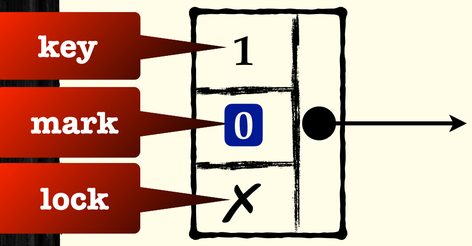
"when finding the cell with the input key marked"

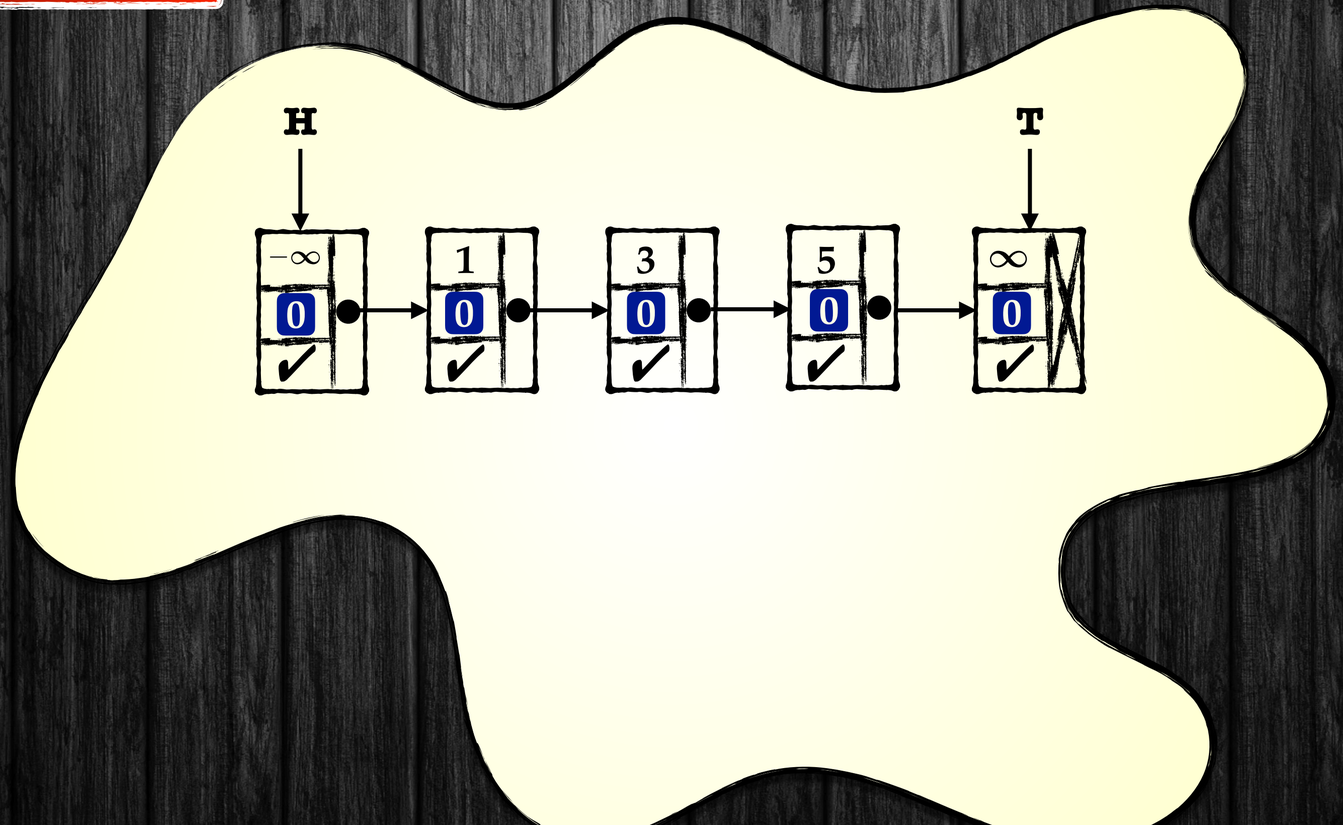
wrong linearization policy

unsuccessful contains

unsuccessful contains

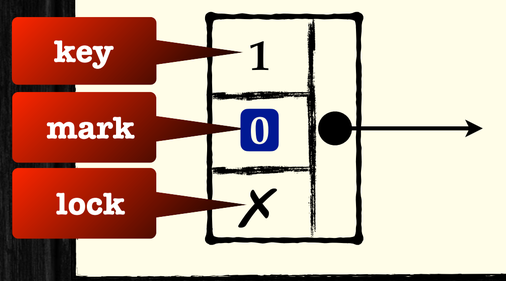
"complicated linearization policy"

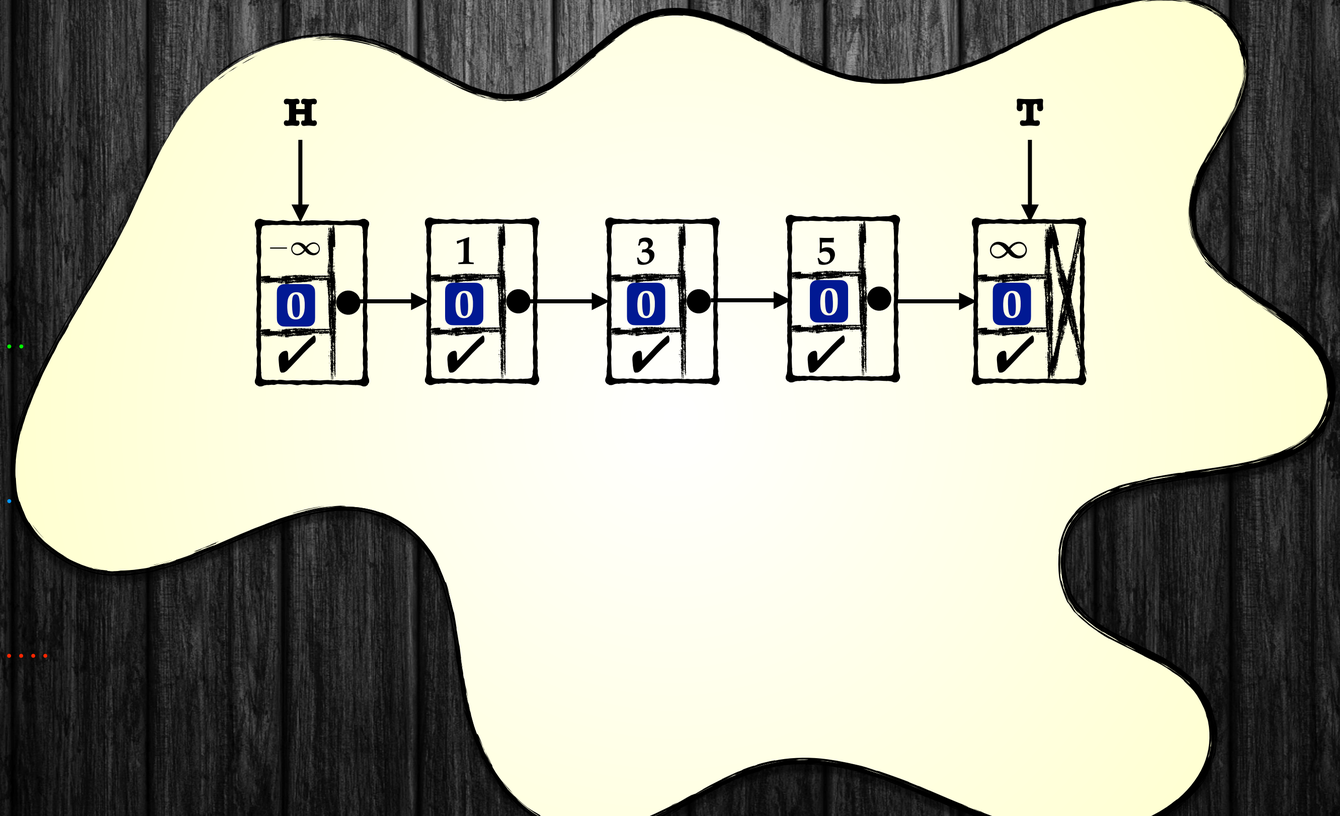




Sets | Lazy Synchronization

```
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;
```

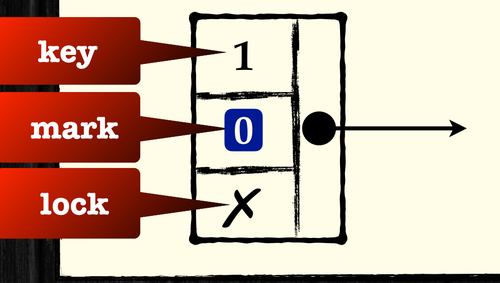


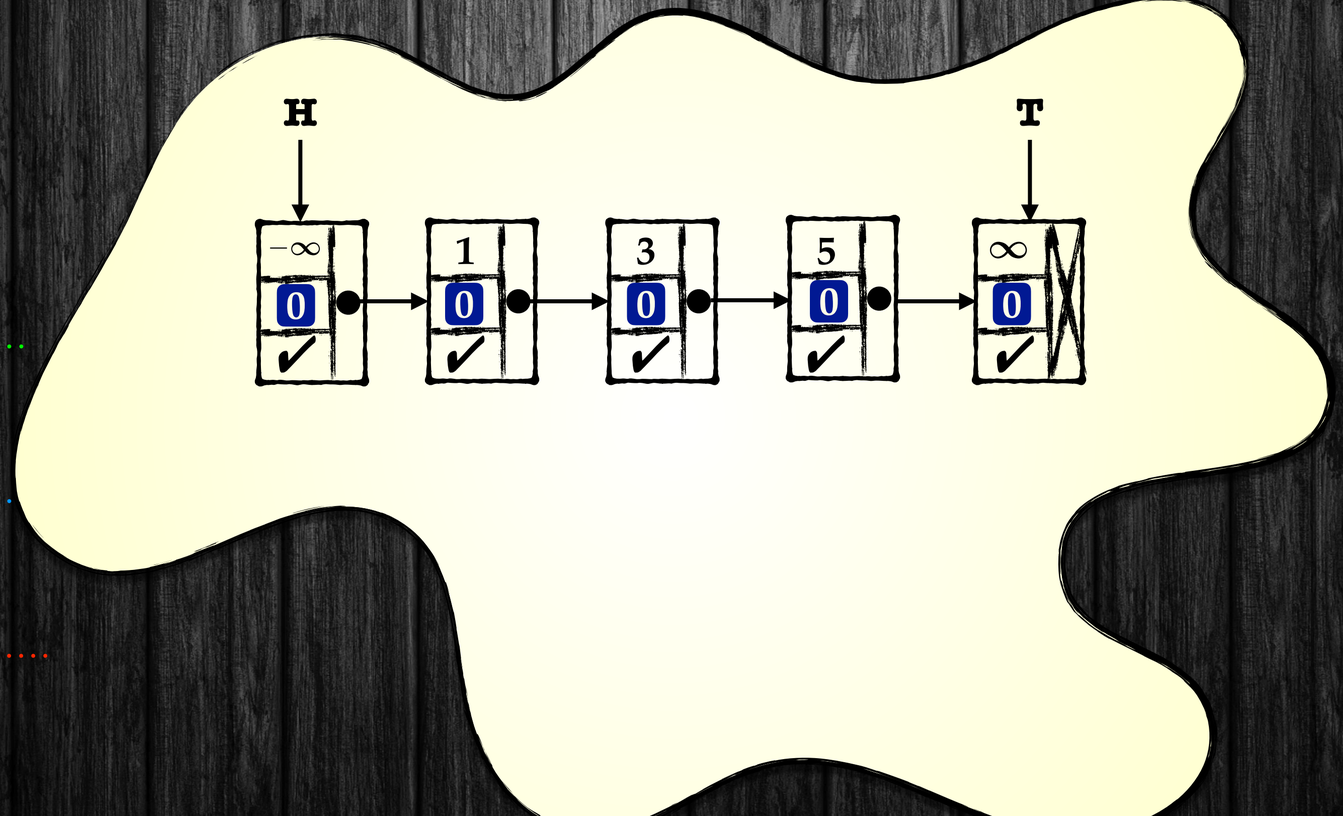


Sets | Lazy Synchronization

```
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)

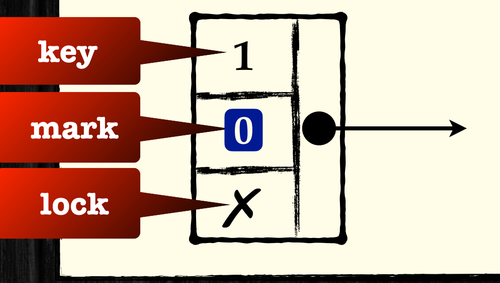


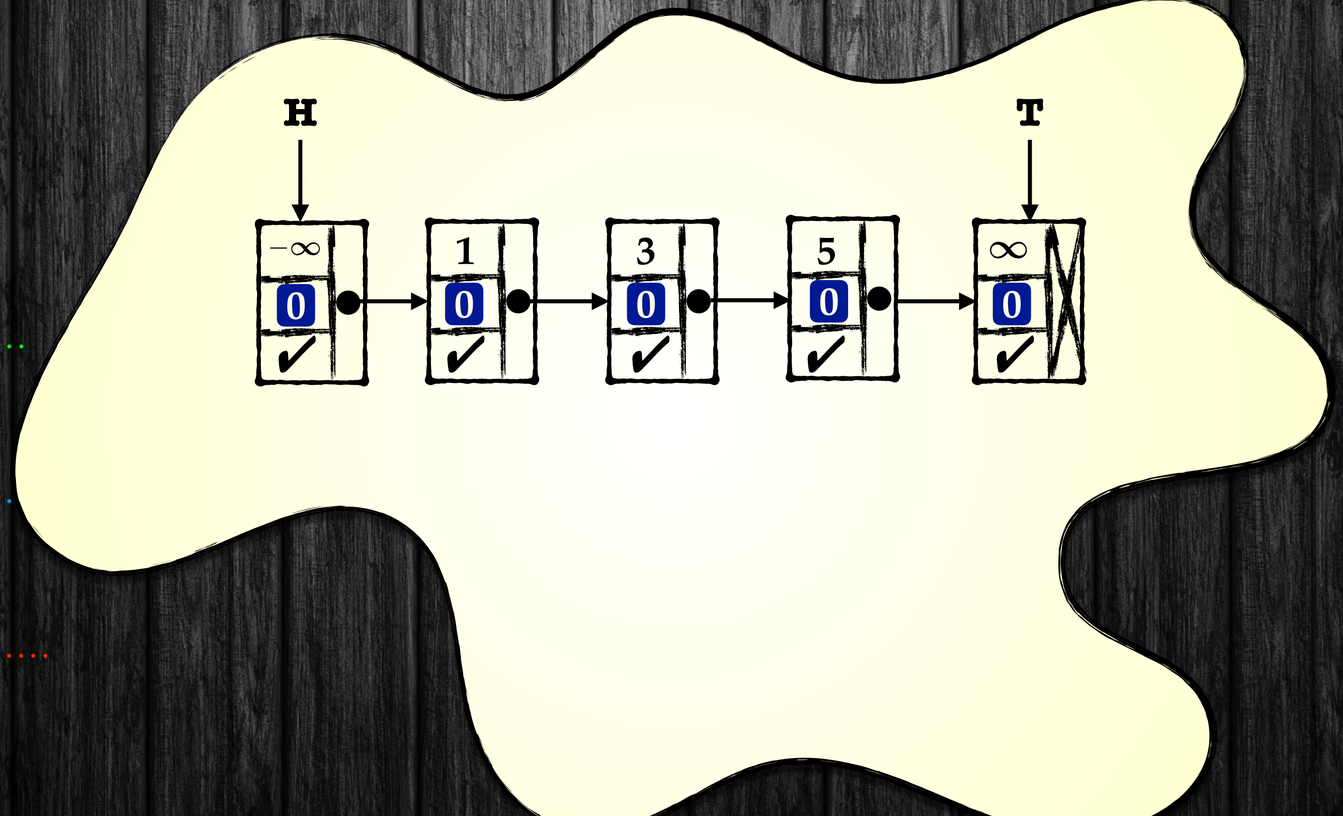


Sets | Lazy Synchronization

```
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)

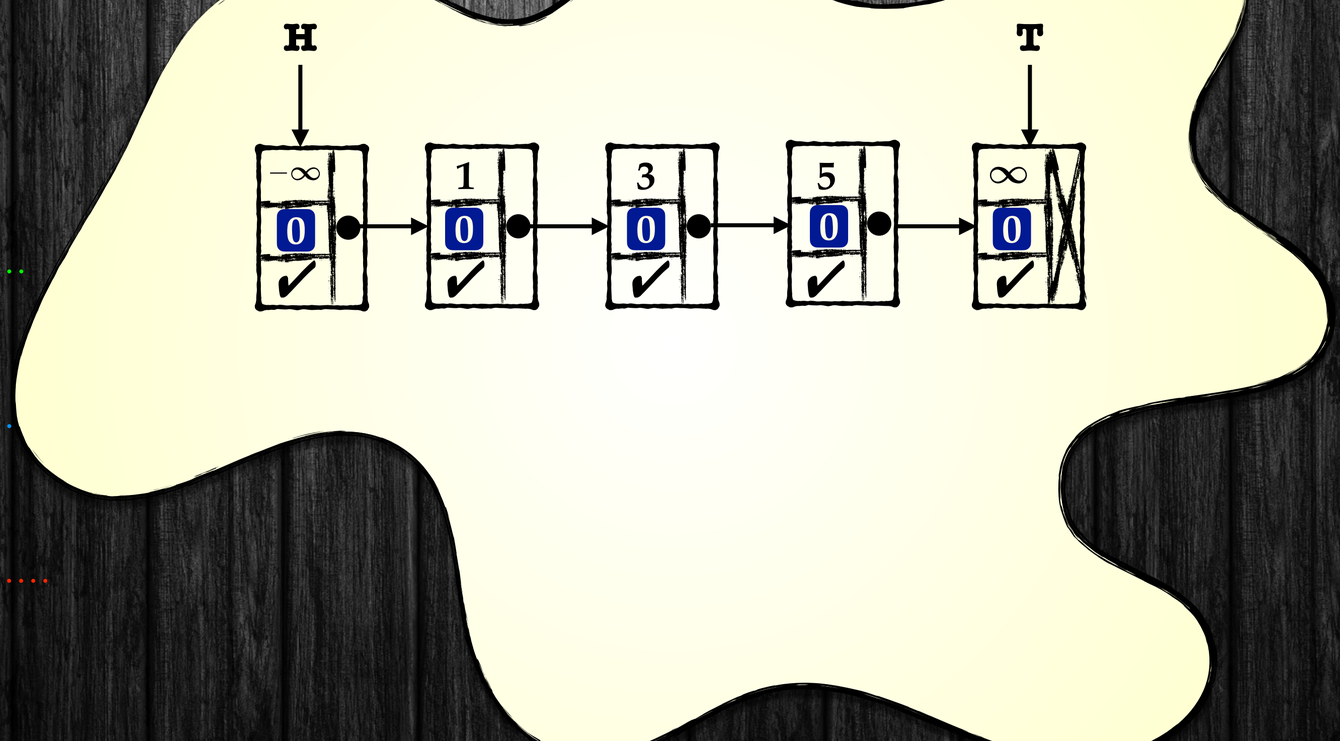


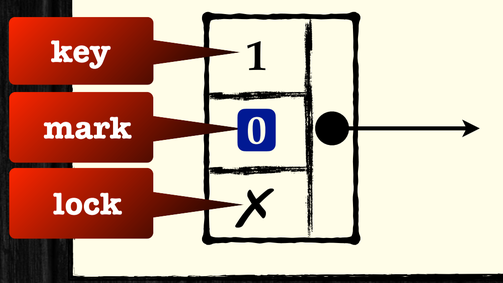


Sets | Lazy Synchronization

```
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)



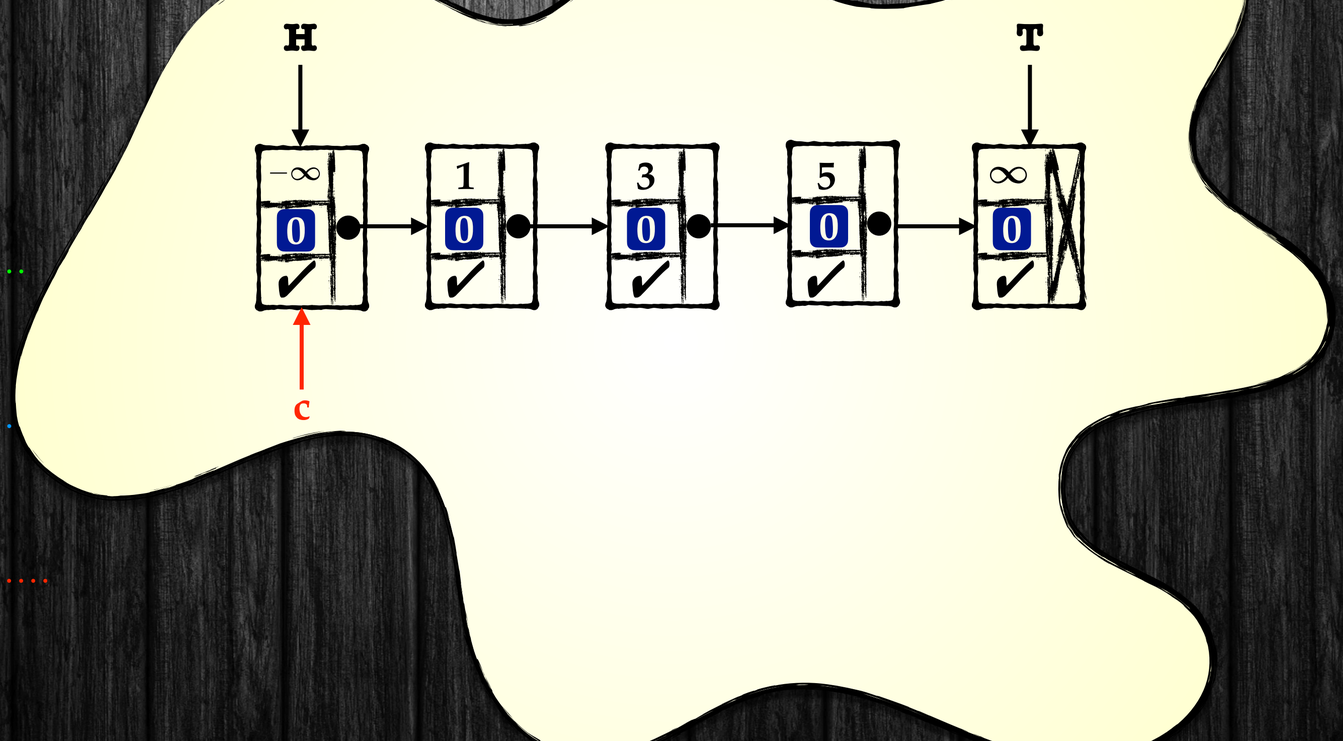


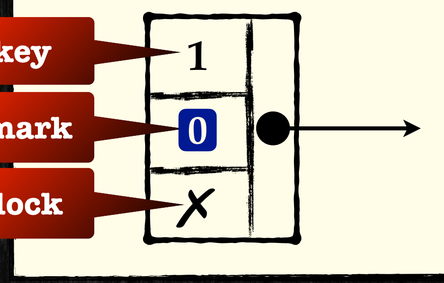
ctn(3)

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)



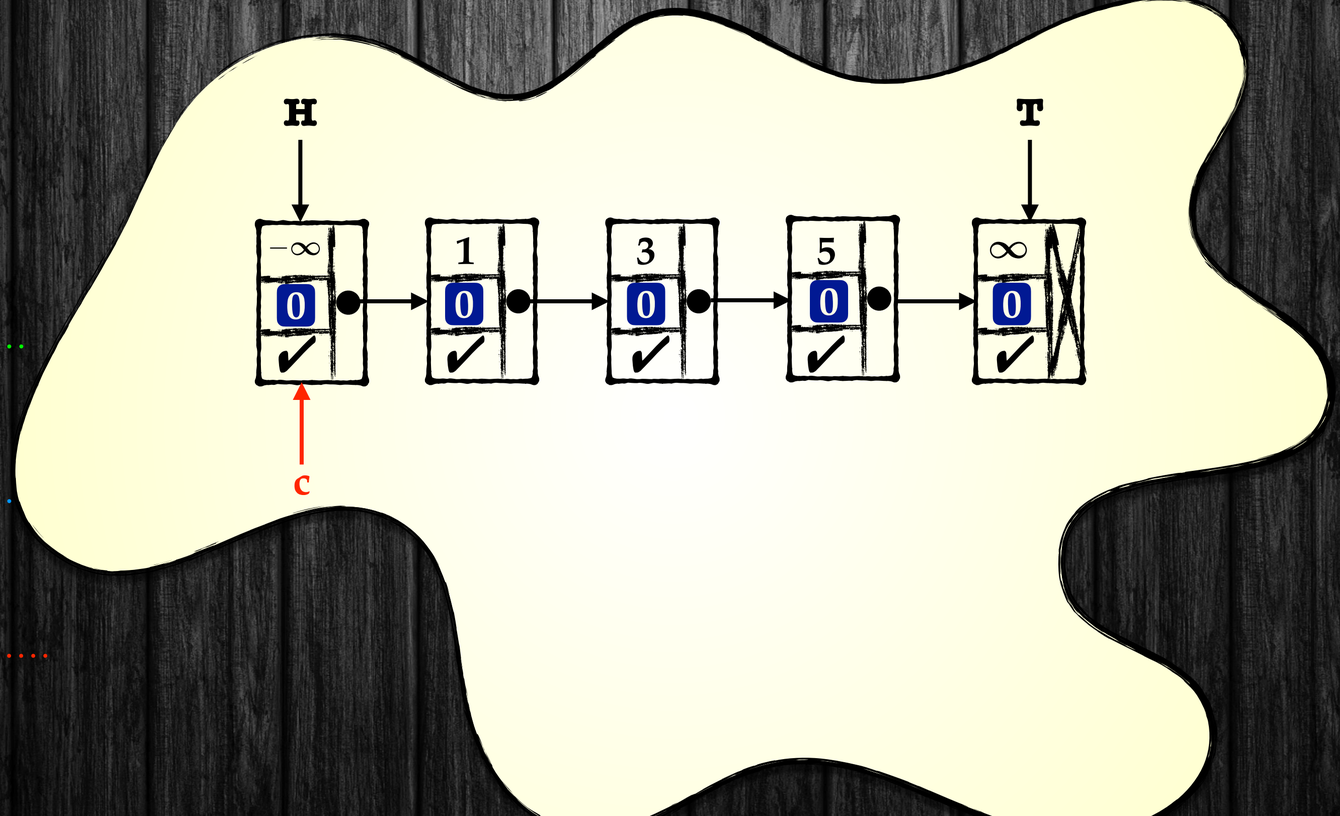


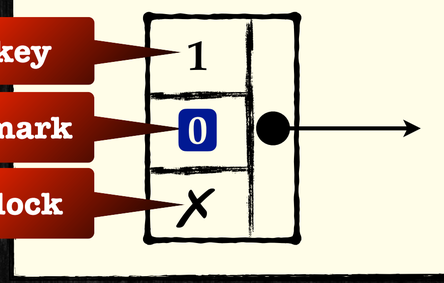
ctn(3)

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)



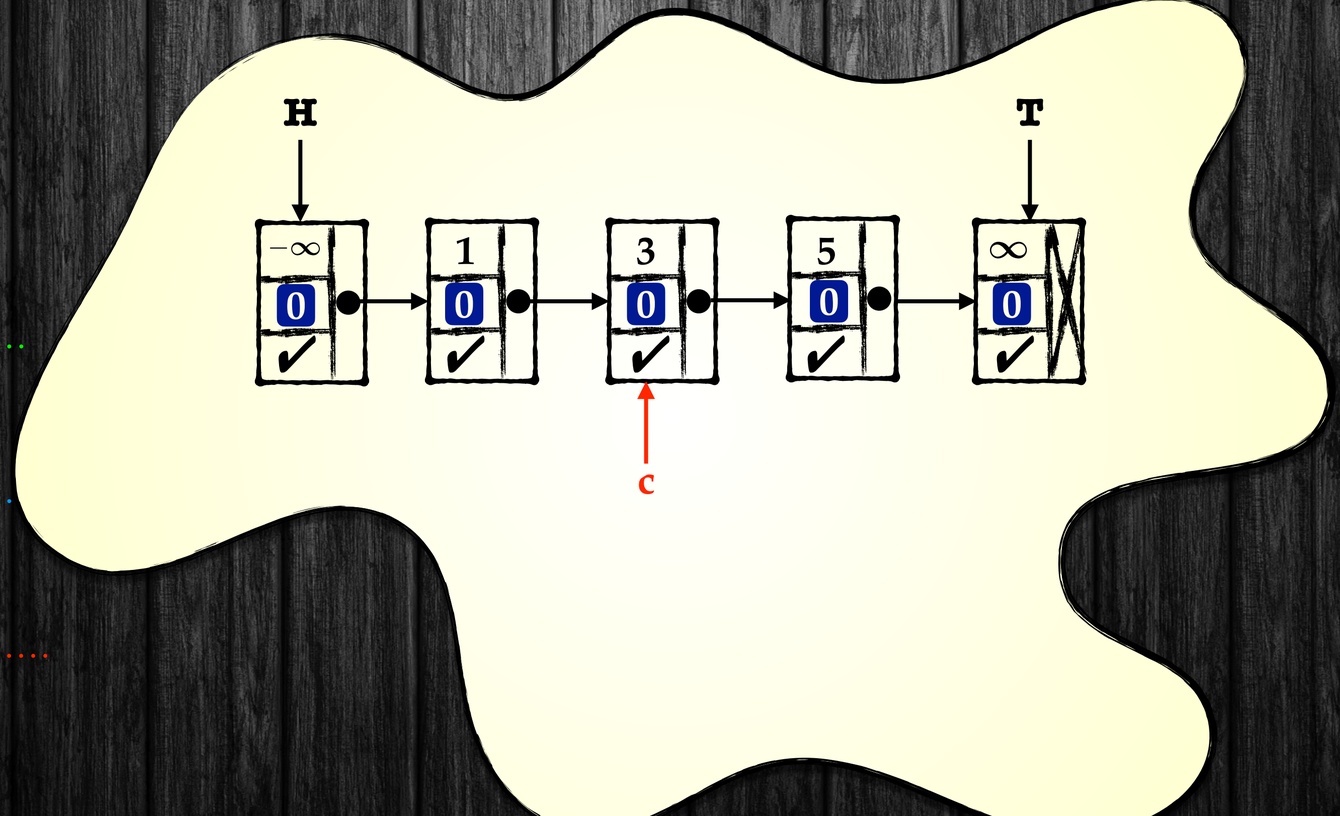


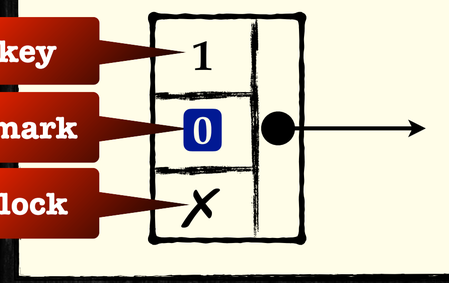
ctn(3)

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)



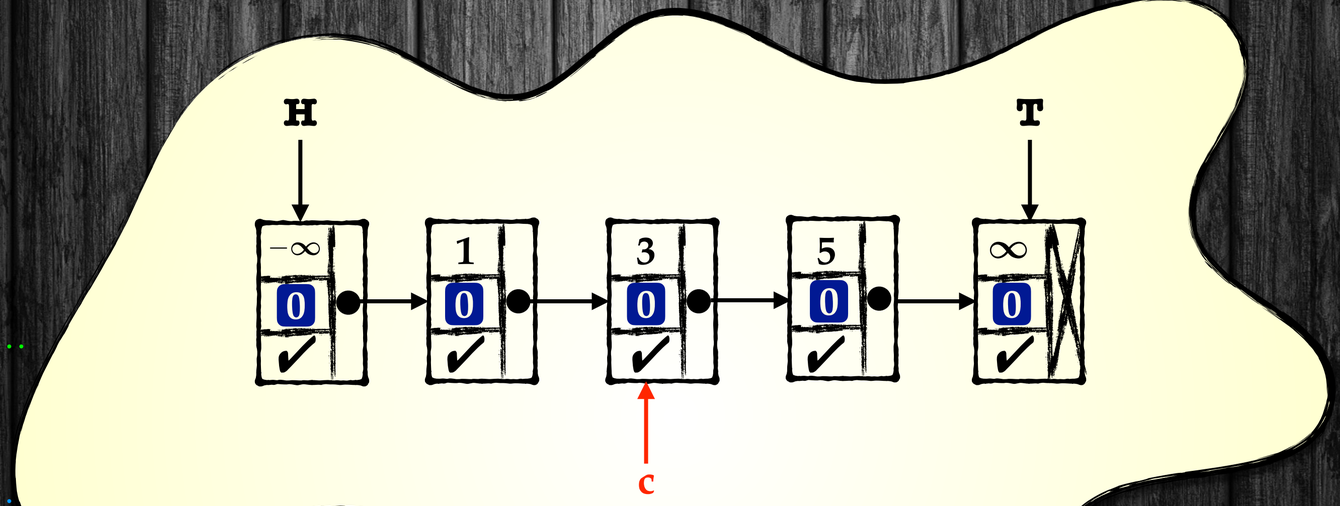


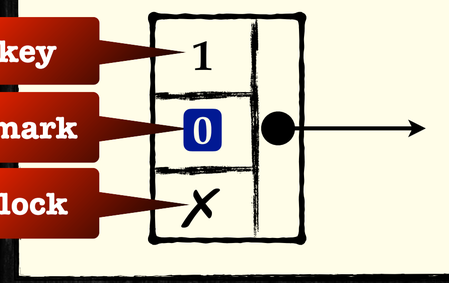
ctn(3)

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)



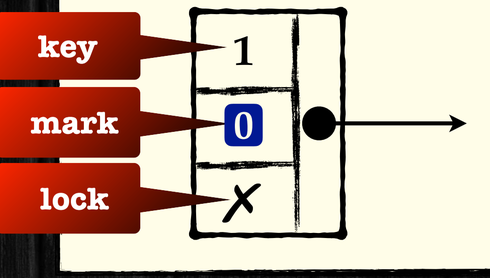


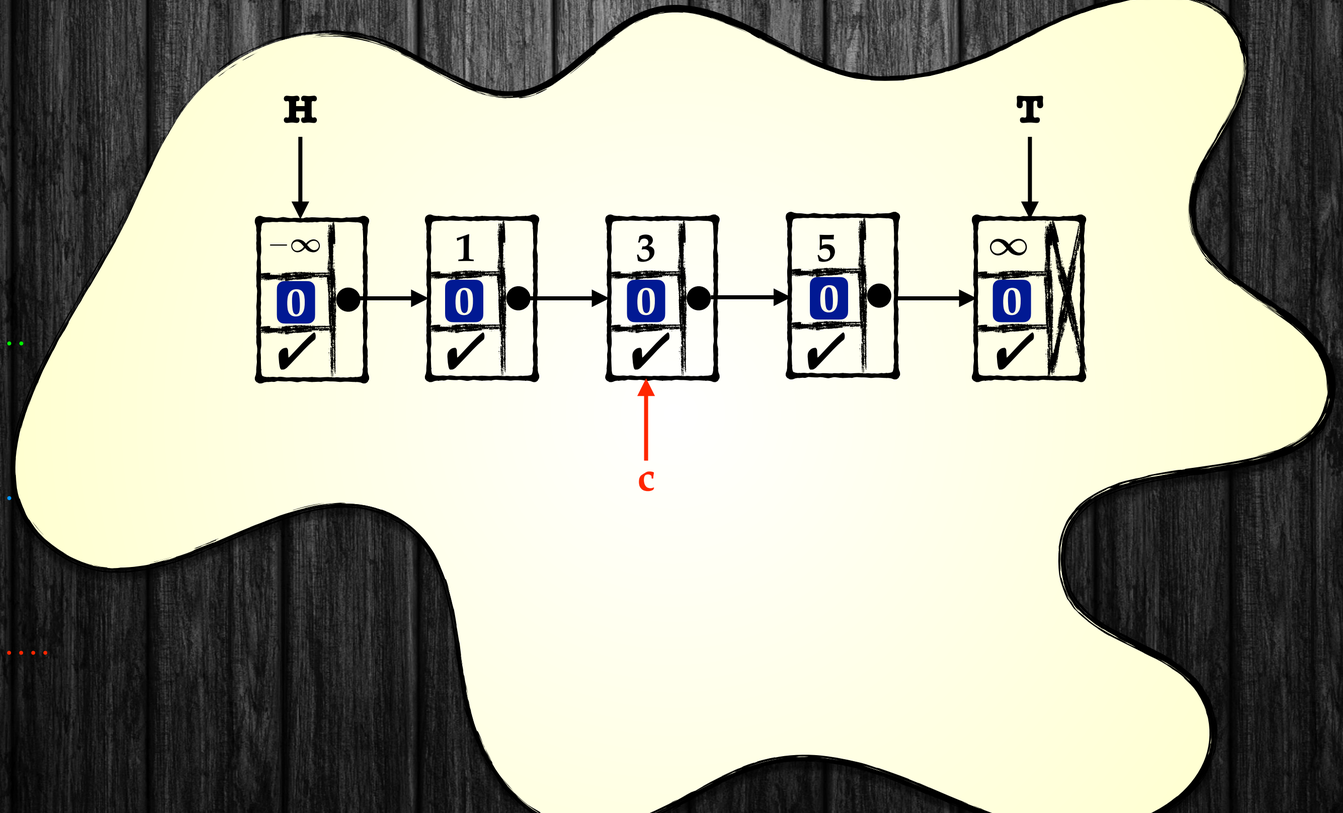
ctn(3)

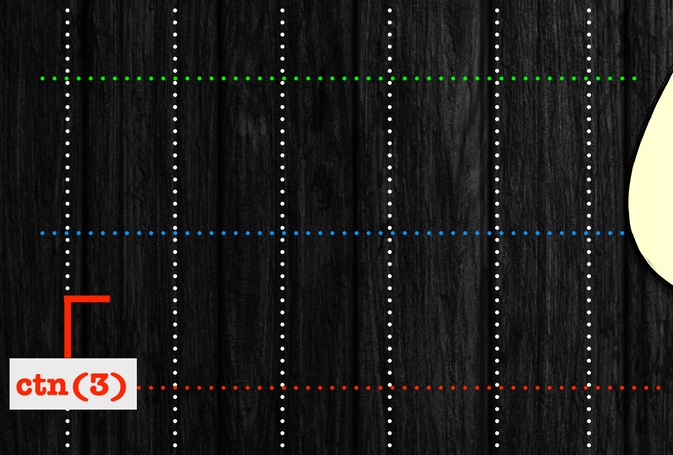
```
ctn(e):
local c, b
1  c := Head;
2  while (c.val < e)
3     c := c.next
4  b := c.mark
5  if (!b && c.val = e)
6     return true;
7  else return false;
```

ctn(3)

not executed yet





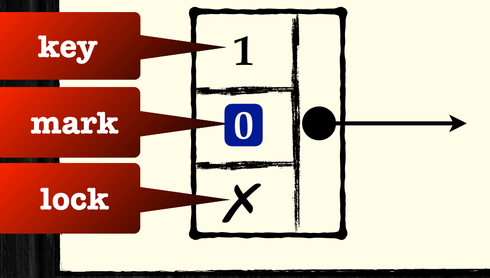


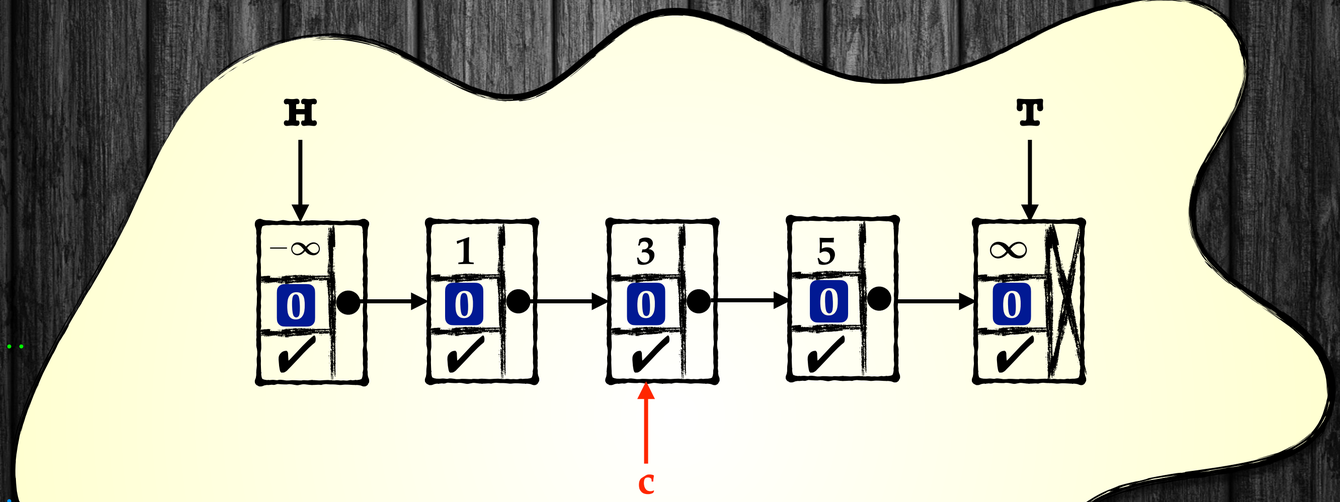
Sets | Lazy Synchronization

```
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





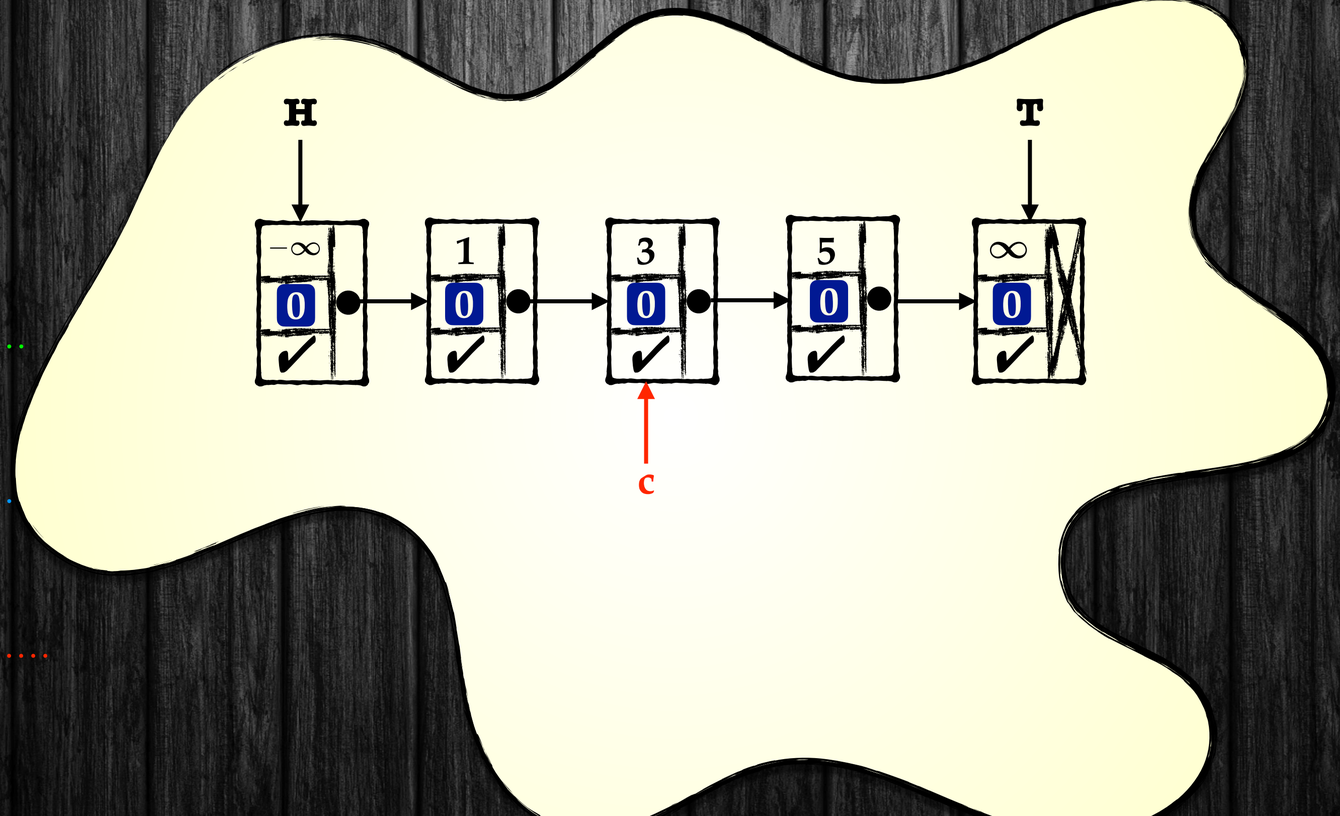
ctn(3)

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



key 1
mark 0
lock ✗

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;
```

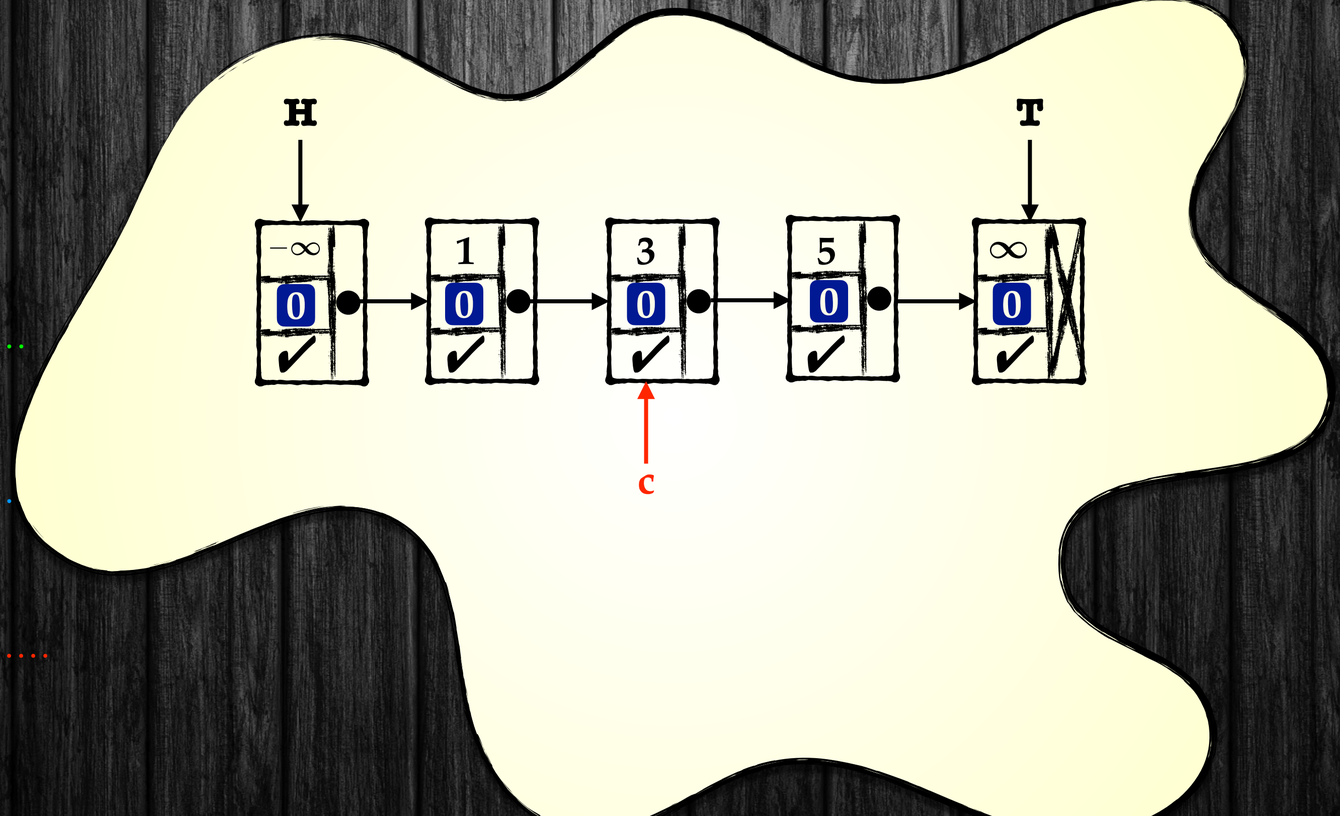
ctn(3)

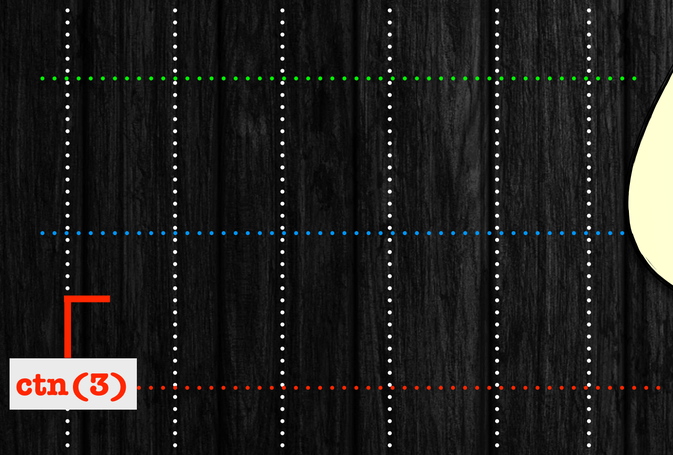
Sets | Lazy Synchronization

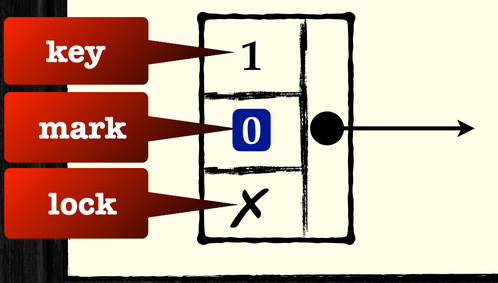
```
ctn(e):
local c, b
1  c := Head;
2  while (c.val < e)
3     c := c.next
4  b := c.mark
5  if (!b && c.val = e)
6     return true;
7  else return false;
```

ctn(3)

not executed yet







rmv(3)

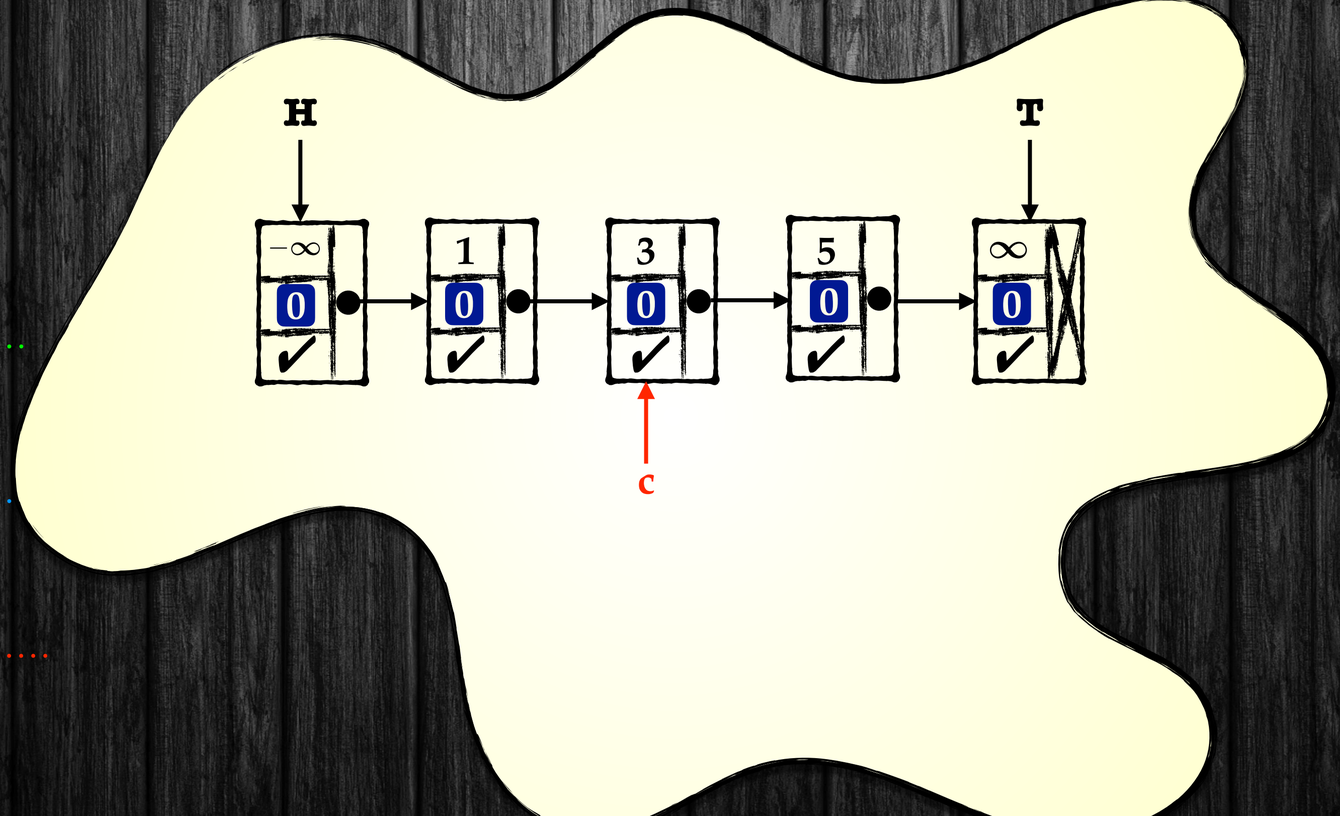
```
rmv(e):
local p, c, n, r
1  (p,c) := locate(e);
2  if (c.val = e)
3     c.mark := true
4     n := c.next;
5     p.next := n;
6     r := true;
7  else r := false;
8  unlock(p);
9  unlock(c);
10 return r;
```

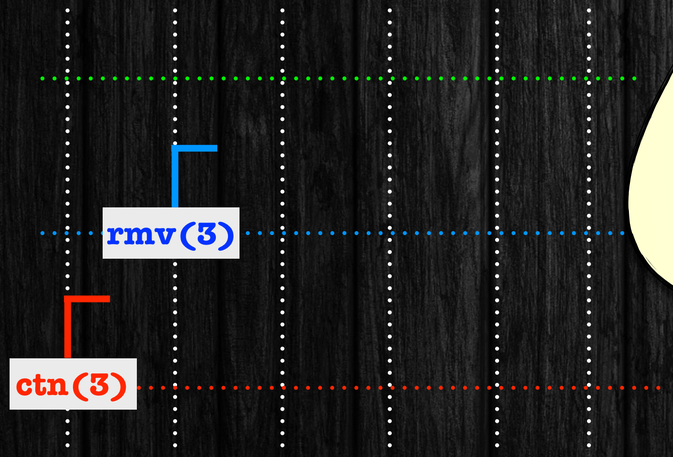
Sets | Lazy Synchronization

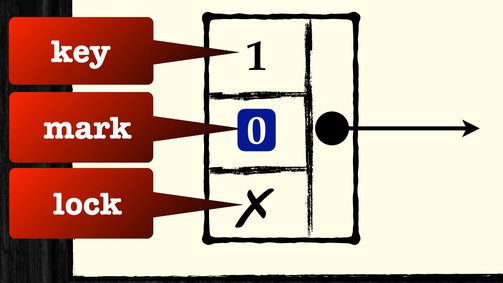
```
ctn(e):
local c, b
1  c := Head;
2  while (c.val < e)
3     c := c.next
4  b := c.mark
5  if (!b && c.val = e)
6     return true;
7  else return false;
```

ctn(3)

not executed yet







rmv(3)

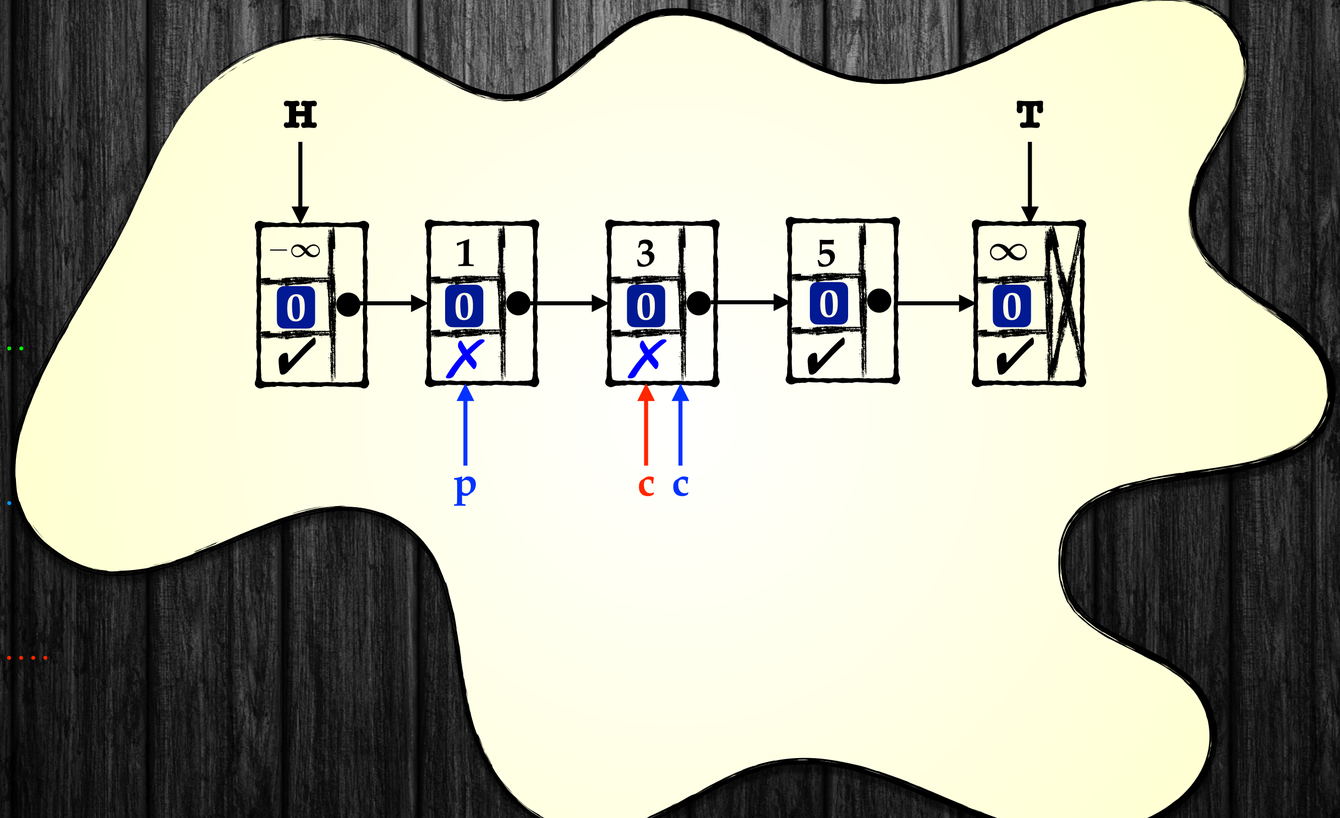
```
rmv(e):
local p, c, n, r
1  (p,c) := locate(e);
2  if (c.val = e)
3     c.mark := true
4     n := c.next;
5     p.next := n;
6     r := true;
7  else r := false;
8  unlock(p);
9  unlock(c);
10 return r;
```

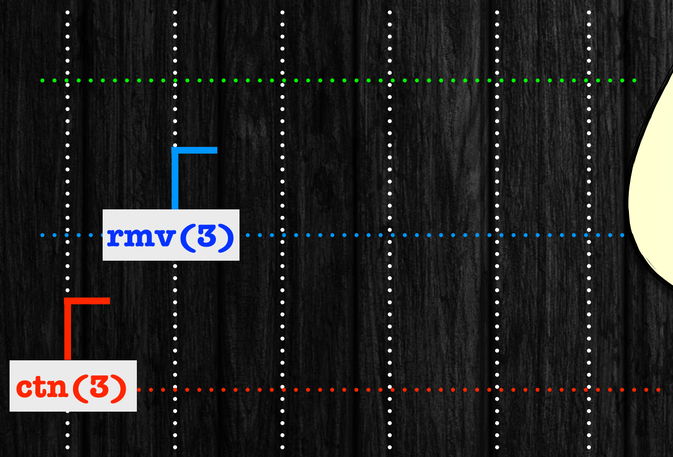
Sets | Lazy Synchronization

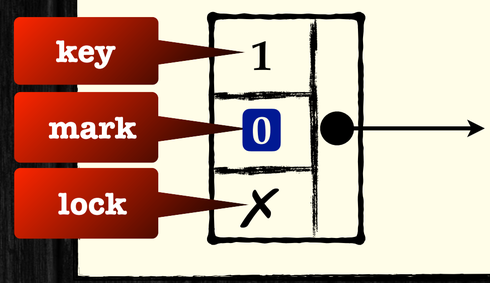
```
ctn(e):
local c, b
1  c := Head;
2  while (c.val < e)
3     c := c.next
4  b := c.mark
5  if (!b && c.val = e)
6     return true;
7  else return false;
```

ctn(3)

not executed yet







rmv(3)

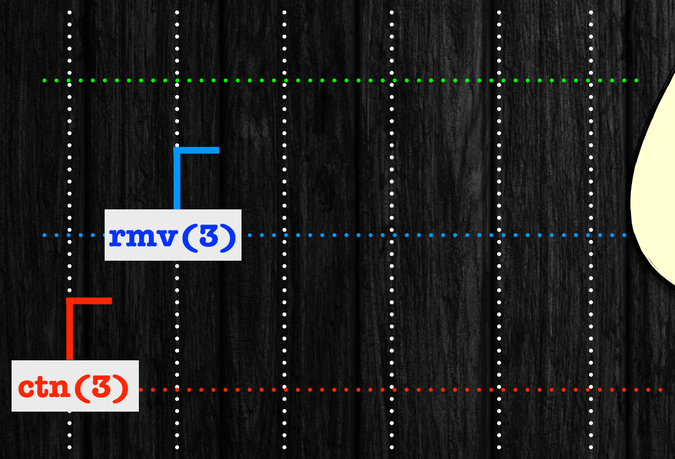
```
rmv(e):
local p, c, n, r
1  (p,c) := locate(e);
2  if (c.val = e)
3     c.mark := true
4     n := c.next;
5     p.next := n;
6     r := true;
7  else r := false;
8  unlock(p);
9  unlock(c);
10 return r;
```

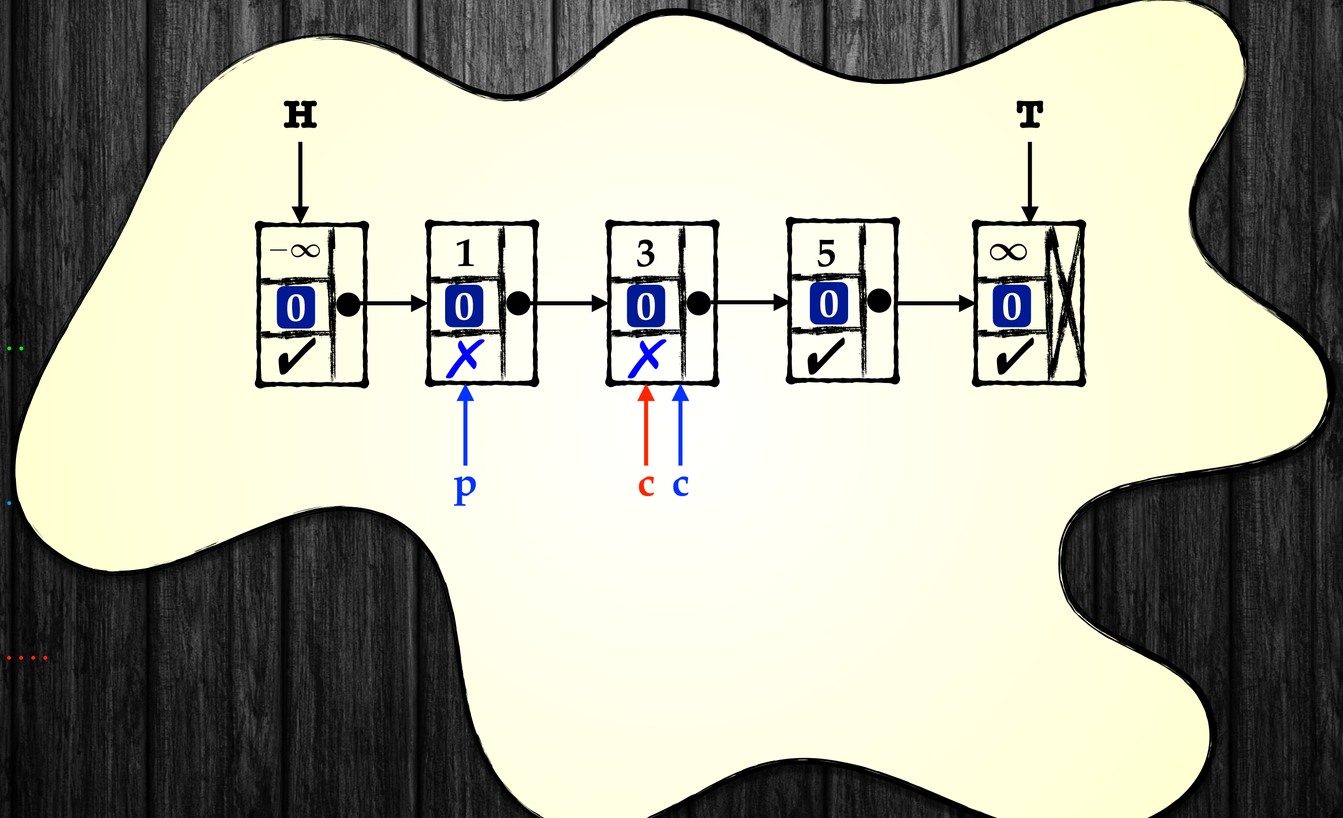
Sets | Lazy Synchronization

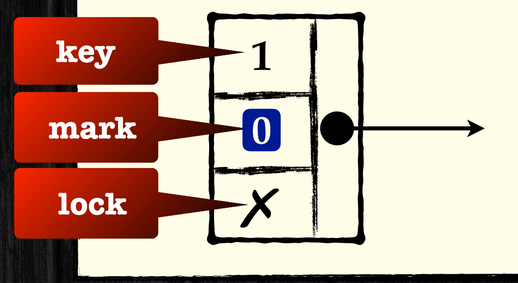
```
ctn(e):
local c, b
1  c := Head;
2  while (c.val < e)
3     c := c.next
4  b := c.mark
5  if (!b && c.val = e)
6     return true;
7  else return false;
```

ctn(3)

not executed yet







rmv(3)

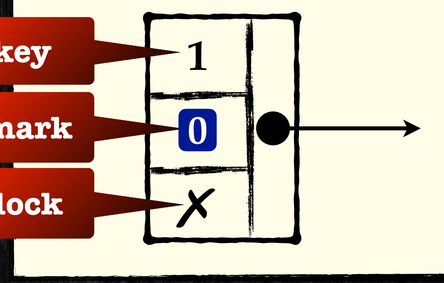
```
rmv(e):
local p, c, n, r
1  (p,c) := locate(e);
2  if (c.val = e)
3     c.mark := true
4     n := c.next;
5     p.next := n;
6     r := true;
7  else r := false;
8  unlock(p);
9  unlock(c);
10 return r;
```

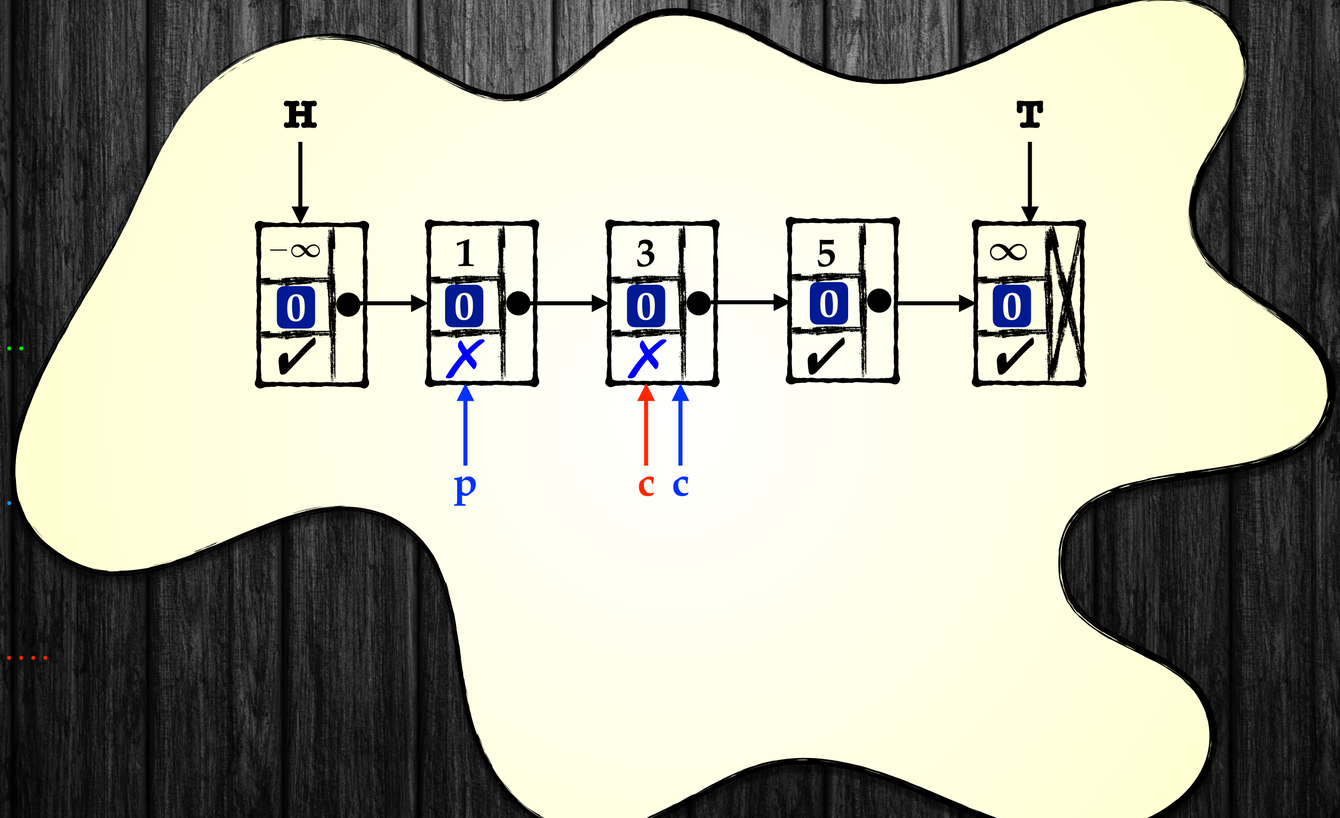
Sets | Lazy Synchronization

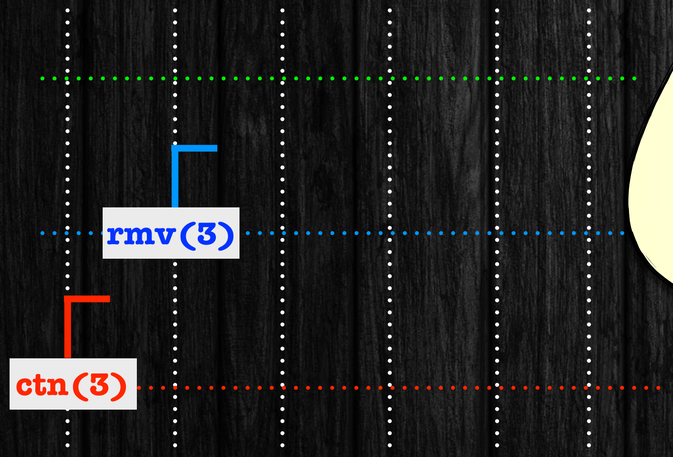
```
ctn(e):
local c, b
1  c := Head;
2  while (c.val < e)
3     c := c.next
4  b := c.mark
5  if (!b && c.val = e)
6     return true;
7  else return false;
```

ctn(3)

not executed yet







rmv(3)

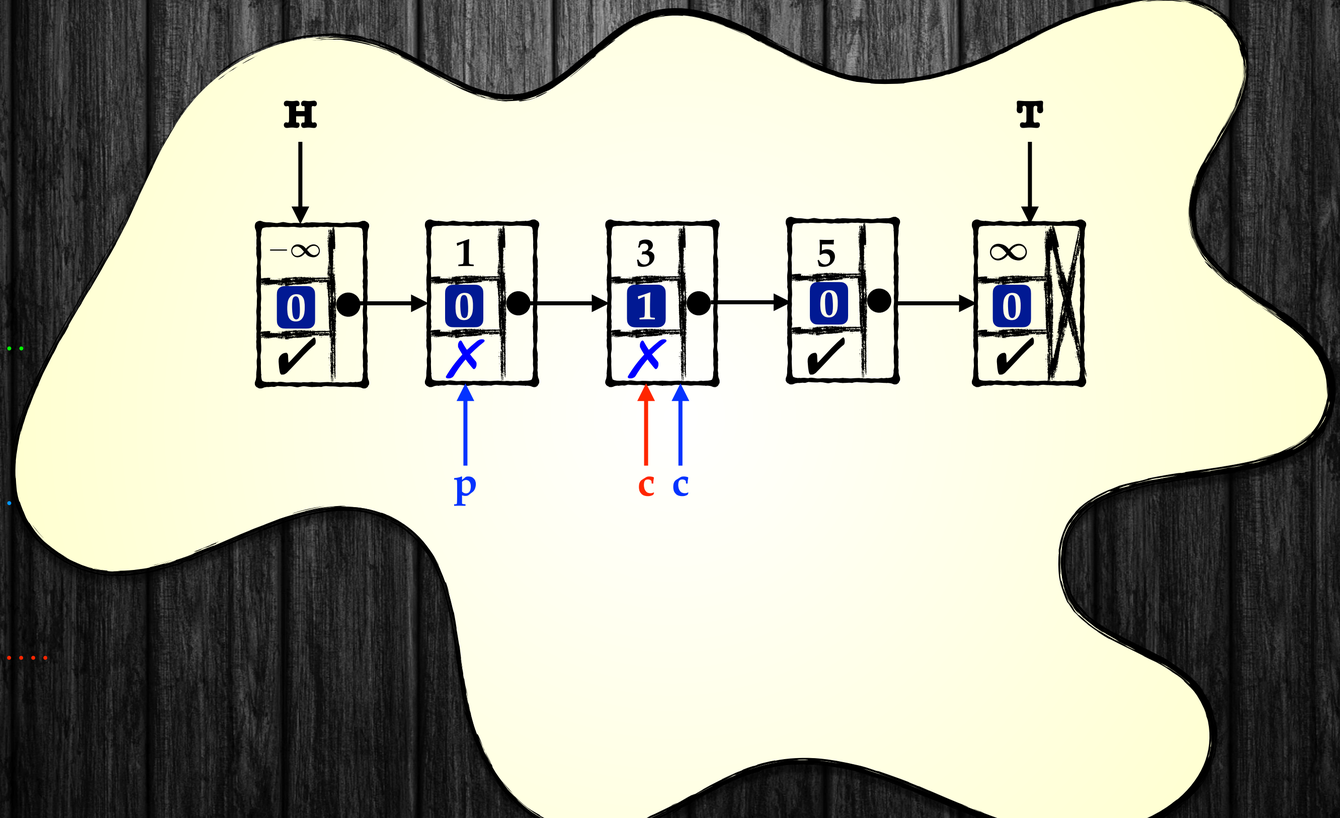
```
rmv(e):
local p, c, n, r
1  (p,c) := locate(e);
2  if (c.val = e)
3     c.mark := true
4     n := c.next;
5     p.next := n;
6     r := true;
7  else r := false;
8  unlock(p);
9  unlock(c);
10 return r;
```

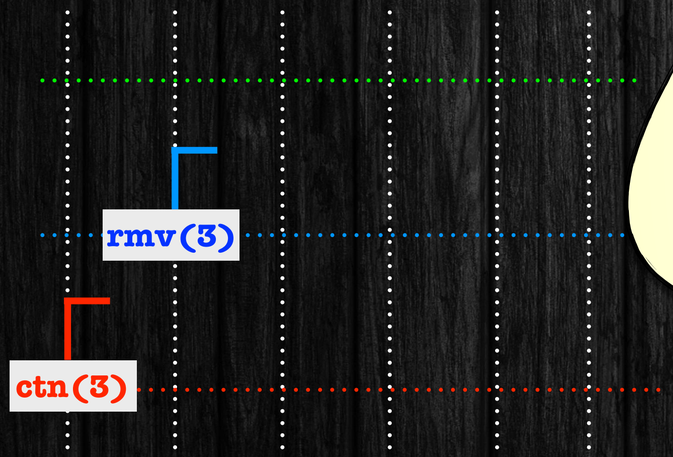
Sets | Lazy Synchronization

```
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)

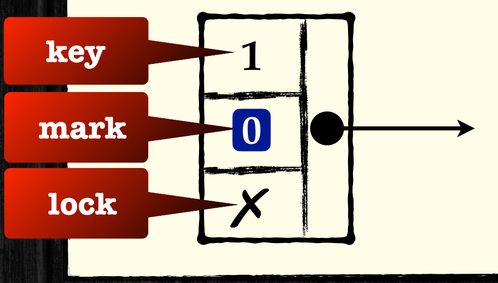
not executed yet






```
rmv(e):
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)

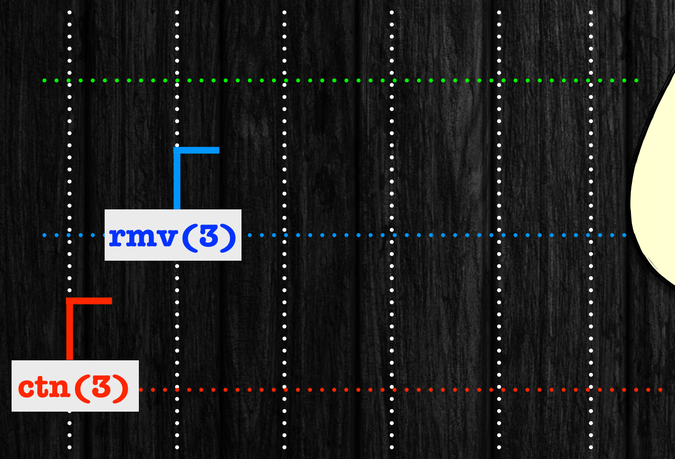


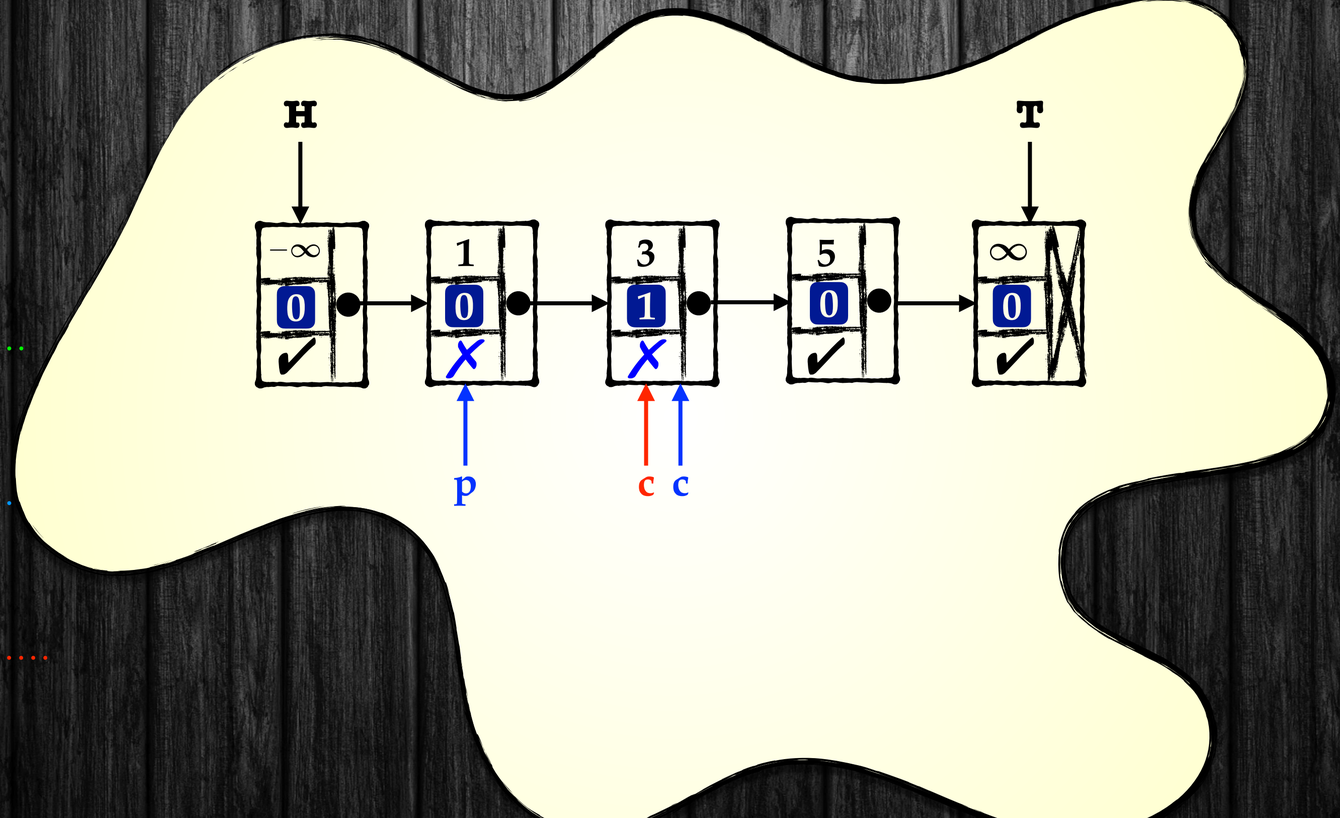
Sets | Lazy Synchronization

```
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;
```

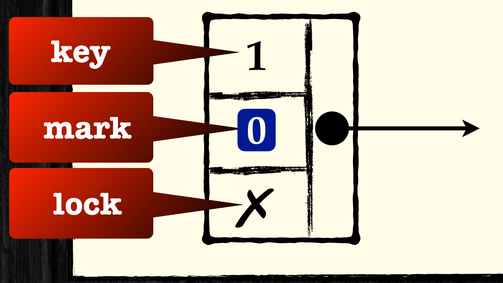
ctn(3)

not executed yet





linearization point



rmv(3)

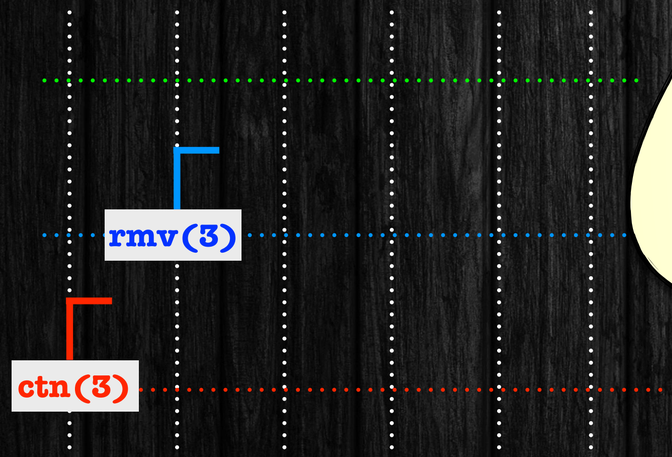
```
rmv(e):
local p, c, n, r
1  (p,c) := locate(e);
2  if (c.val = e)
3     c.mark := true
4     n := c.next;
5     p.next := n;
6     r := true;
7  else r := false;
8  unlock(p);
9  unlock(c);
10 return r;
```

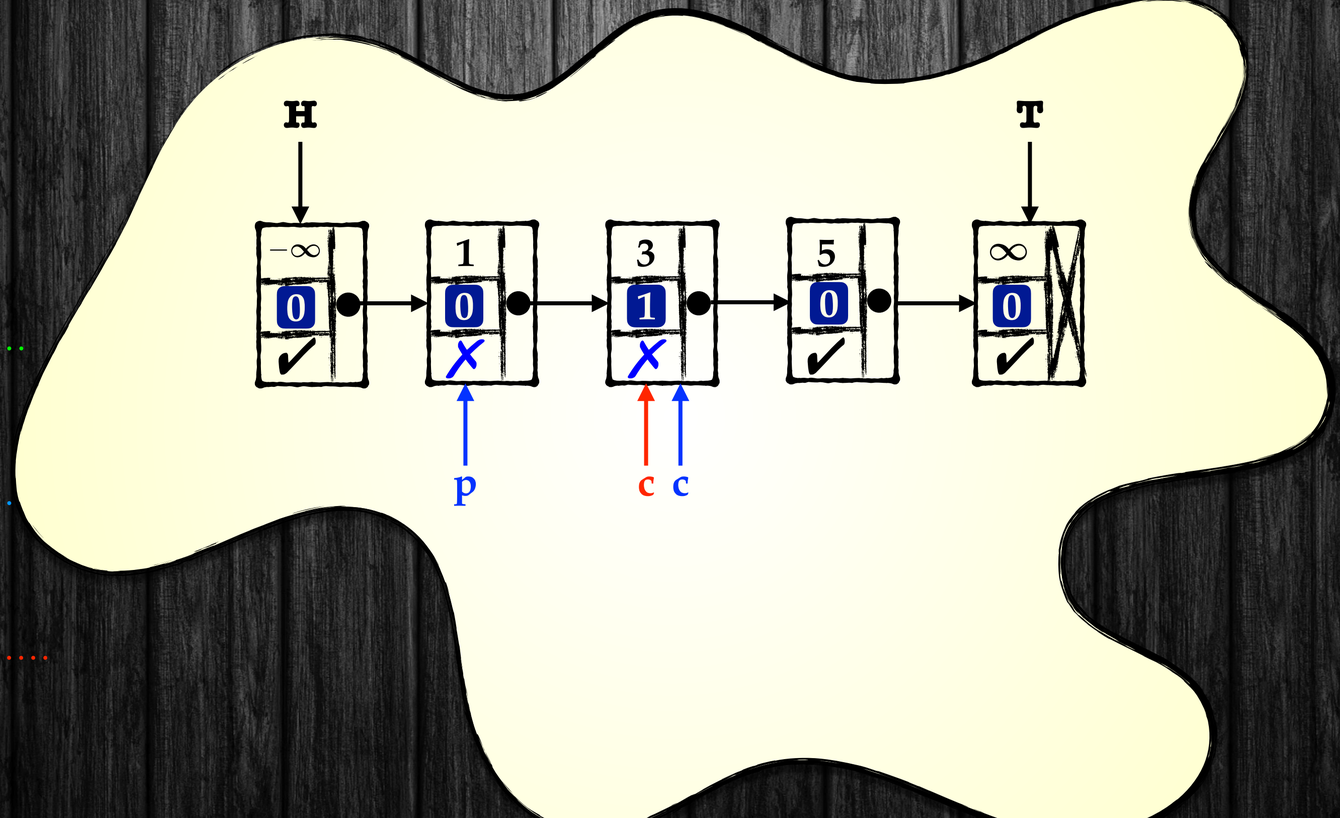
Sets | Lazy Synchronization

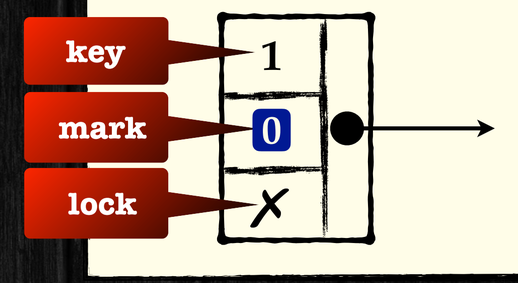
```
ctn(e):
local c, b
1  c := Head;
2  while (c.val < e)
3     c := c.next
4  b := c.mark
5  if (!b && c.val = e)
6     return true;
7  else return false;
```

ctn(3)

not executed yet







linearization point

successful remove

```
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;
```

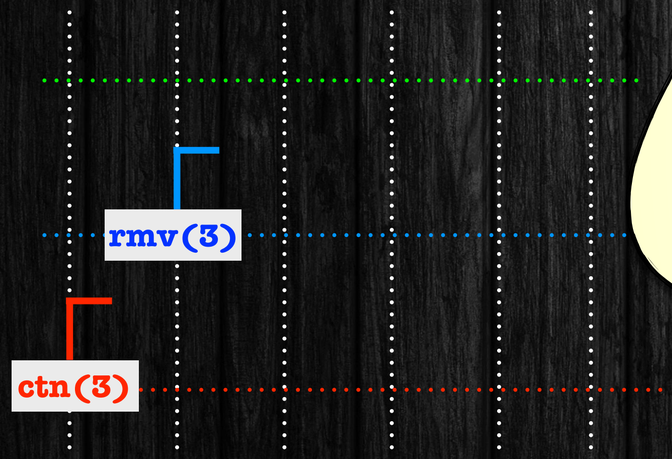
rmv(3)

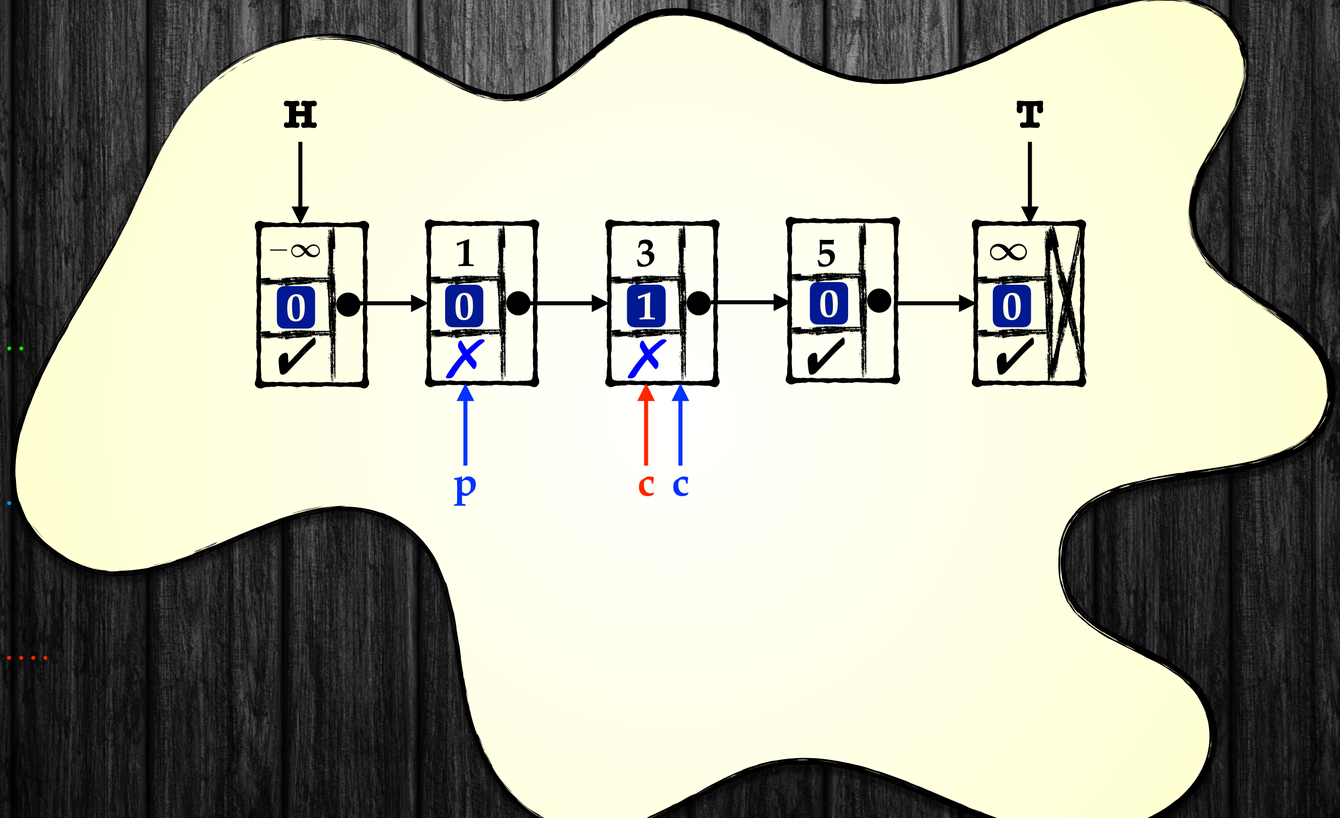
Sets | Lazy Synchronization

```
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;
```

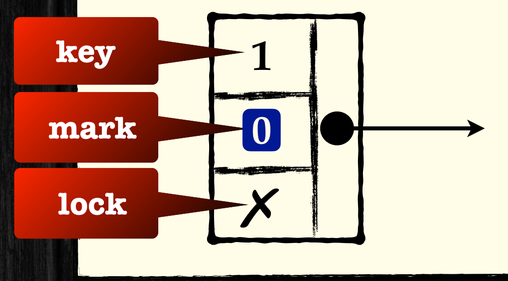
ctn(3)

not executed yet





linearization point



successful remove

"when the cell is marked"

rmv(3)

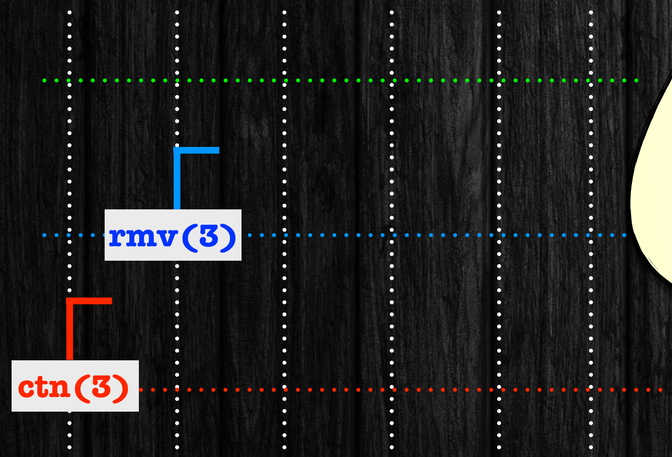
```
rmv(e):
local p, c, n, r
1  (p,c) := locate(e);
2  if (c.val = e)
3     c.mark := true
4     n := c.next;
5     p.next := n;
6     r := true;
7  else r := false;
8  unlock(p);
9  unlock(c);
10 return r;
```

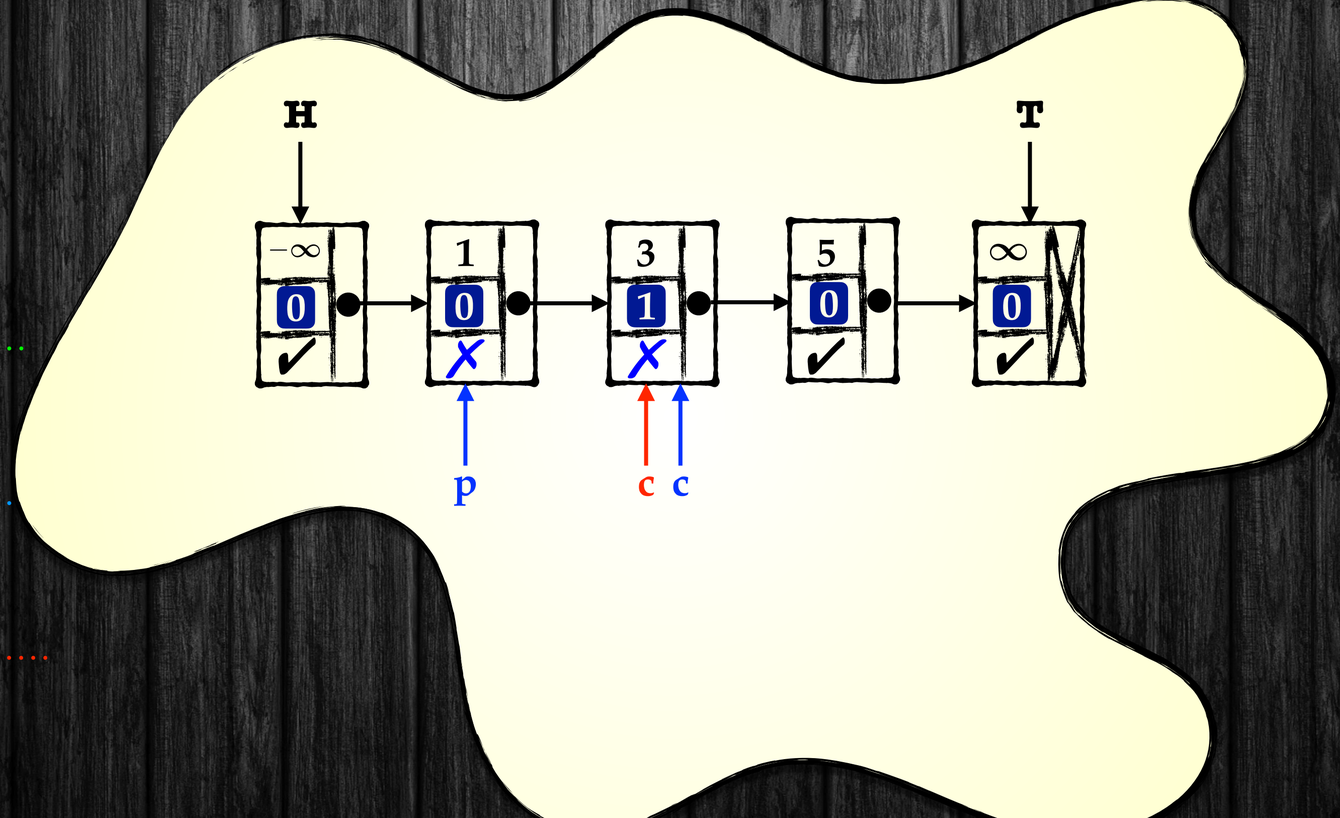
Sets | Lazy Synchronization

```
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;
```

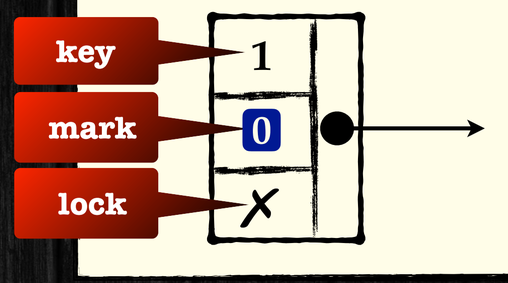
ctn(3)

not executed yet





linearization point



successful remove

"when the cell is marked"

rmv(3)

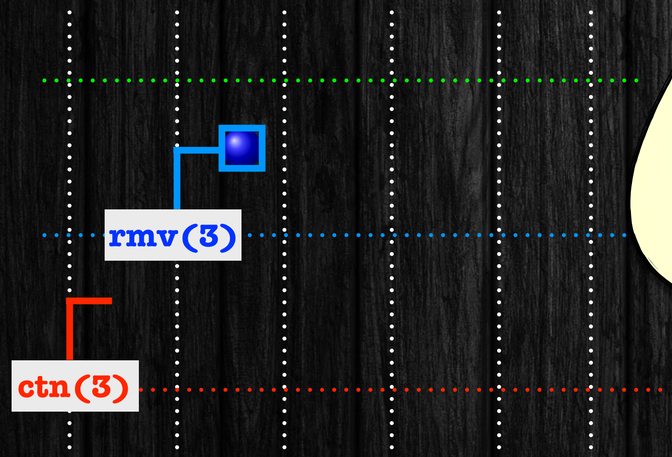
```
rmv(e):
local p, c, n, r
1  (p,c) := locate(e);
2  if (c.val = e)
3     c.mark := true
4     n := c.next;
5     p.next := n;
6     r := true;
7  else r := false;
8  unlock(p);
9  unlock(c);
10 return r;
```

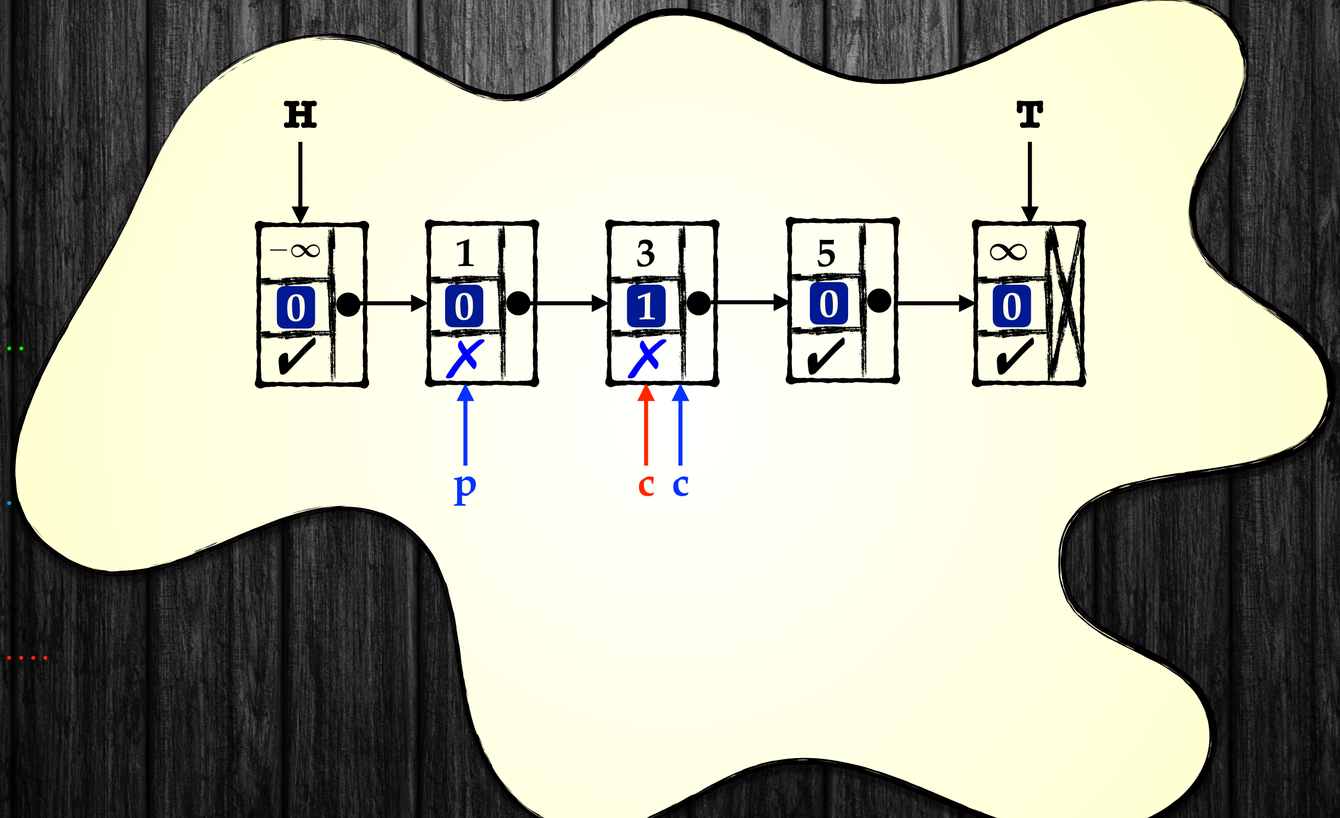
Sets | Lazy Synchronization

```
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;
```

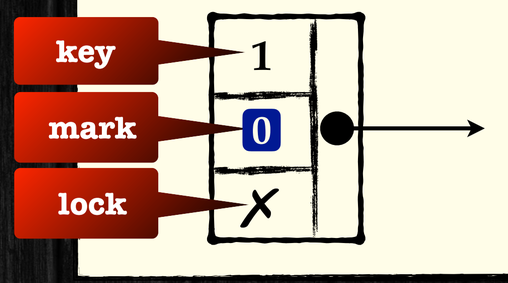
ctn(3)

not executed yet





linearization point



successful remove

"when the cell is marked"

rmv(3)

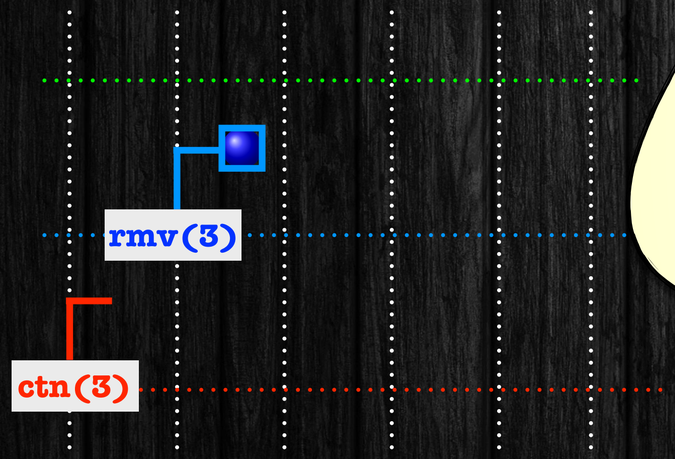
```
rmv(e):
local p, c, n, r
1  (p,c) := locate(e);
2  if (c.val = e)
3     c.mark := true
4     n := c.next;
5     p.next := n;
6     r := true;
7  else r := false;
8  unlock(p);
9  unlock(c);
10 return r;
```

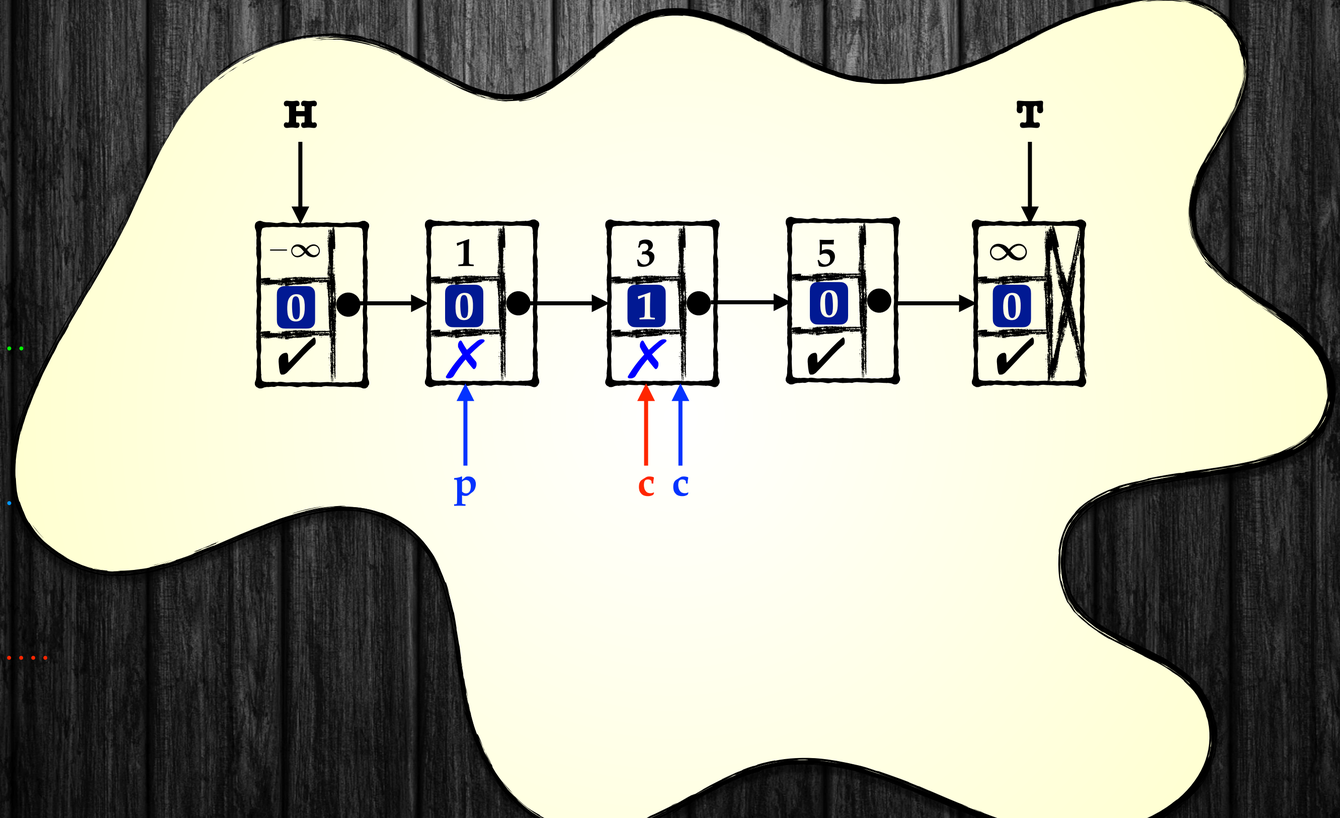
Sets | Lazy Synchronization

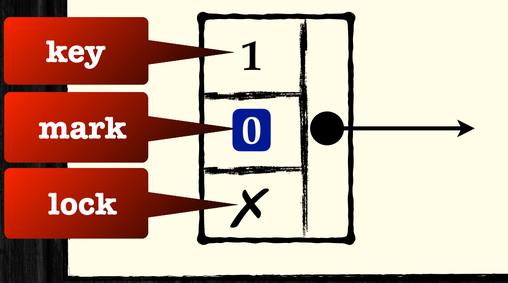
```
ctn(e):
local c, b
1  c := Head;
2  while (c.val < e)
3     c := c.next
4  b := c.mark
5  if (!b && c.val = e)
6     return true;
7  else return false;
```

ctn(3)

not executed yet







rmv(3)

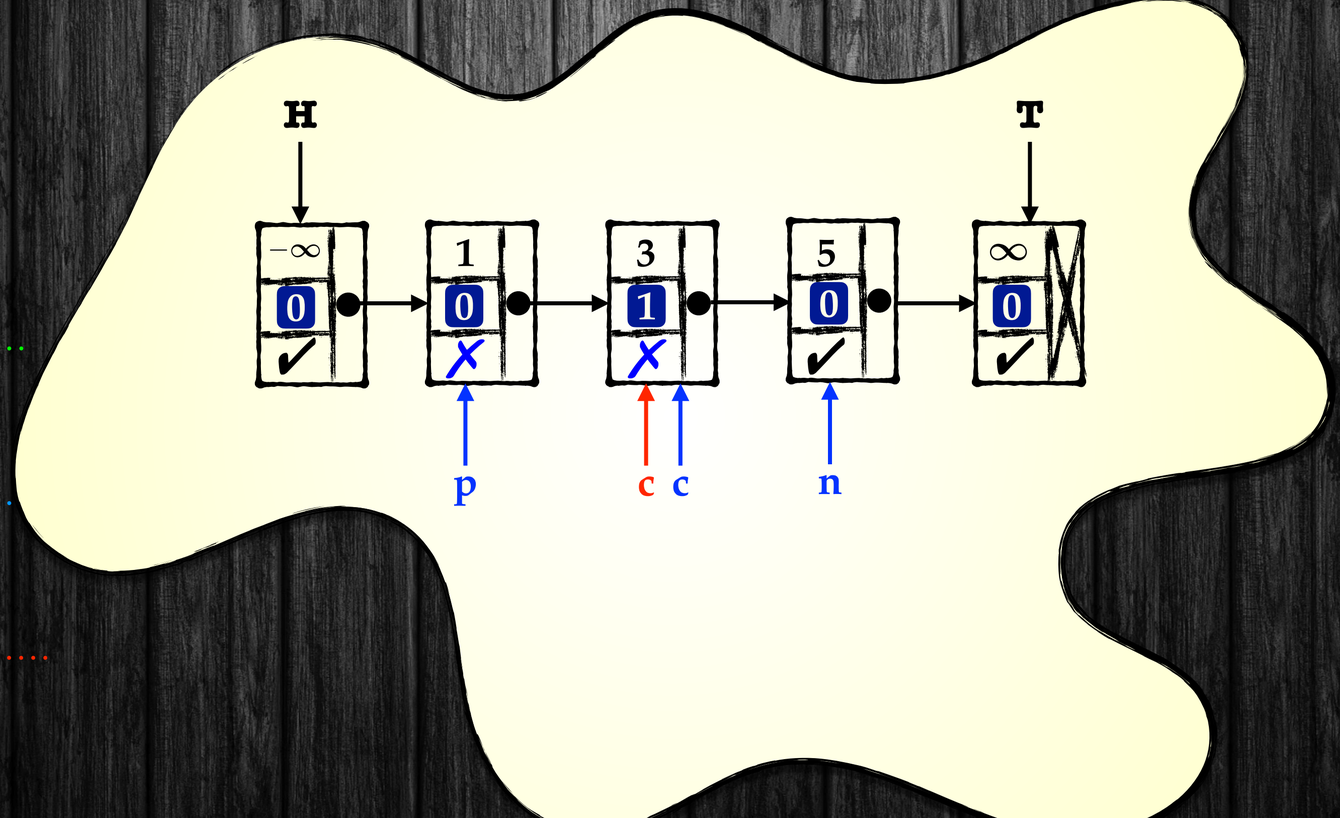
```
rmv(e):
local p, c, n, r
1  (p,c) := locate(e);
2  if (c.val = e)
3     c.mark := true
4     n := c.next;
5     p.next := n;
6     r := true;
7  else r := false;
8  unlock(p);
9  unlock(c);
10 return r;
```

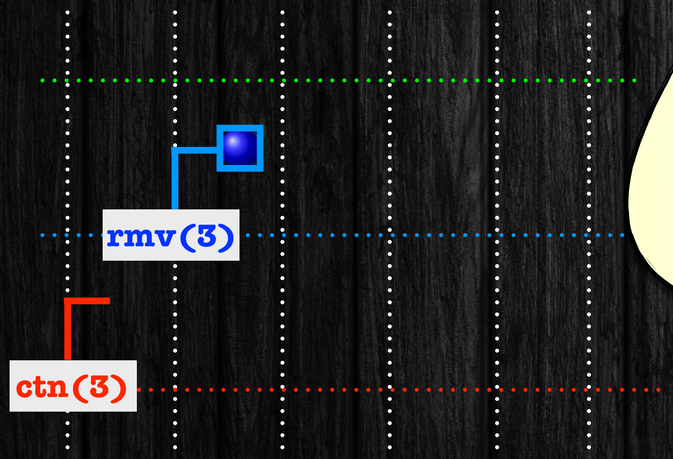
Sets | Lazy Synchronization

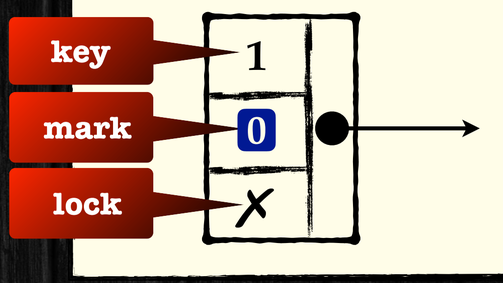
```
ctn(e):
local c, b
1  c := Head;
2  while (c.val < e)
3     c := c.next
4  b := c.mark
5  if (!b && c.val = e)
6     return true;
7  else return false;
```

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;
```

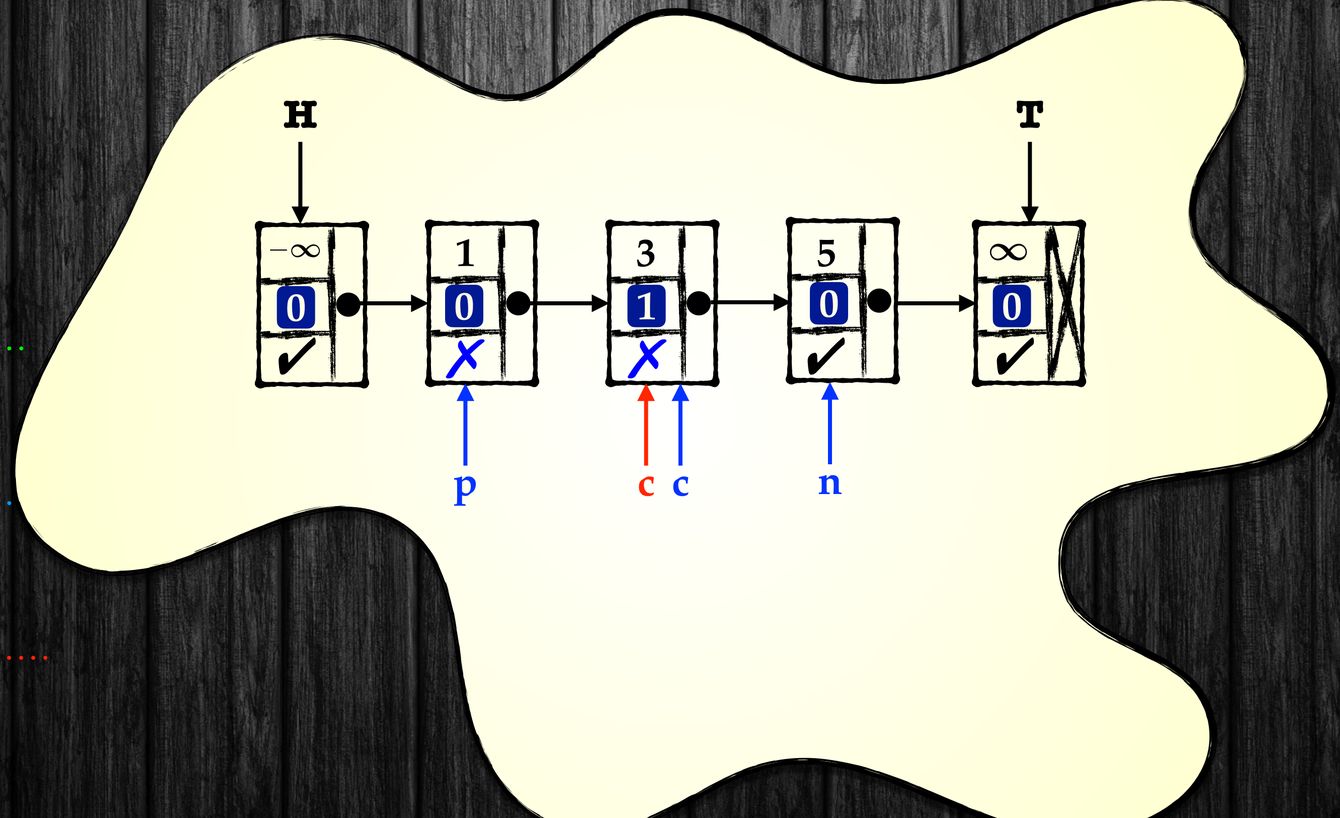
Sets | Lazy Synchronization

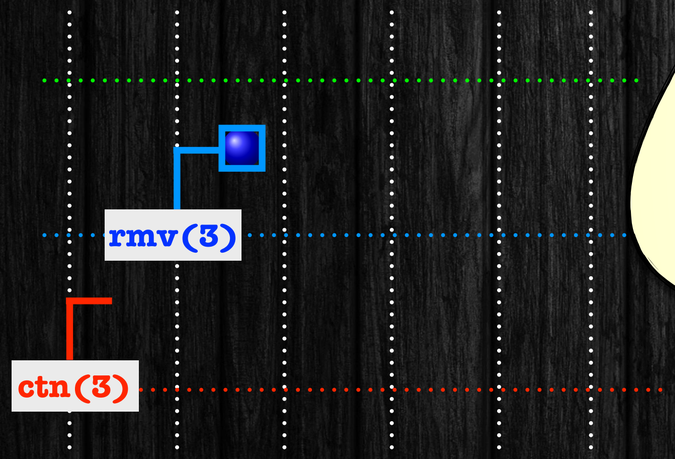
```
ctn(e):
local c, b
1  c := Head;
2  while (c.val < e)
3     c := c.next
4  b := c.mark
5  if (!b && c.val = e)
6     return true;
7  else return false;
```

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;
```

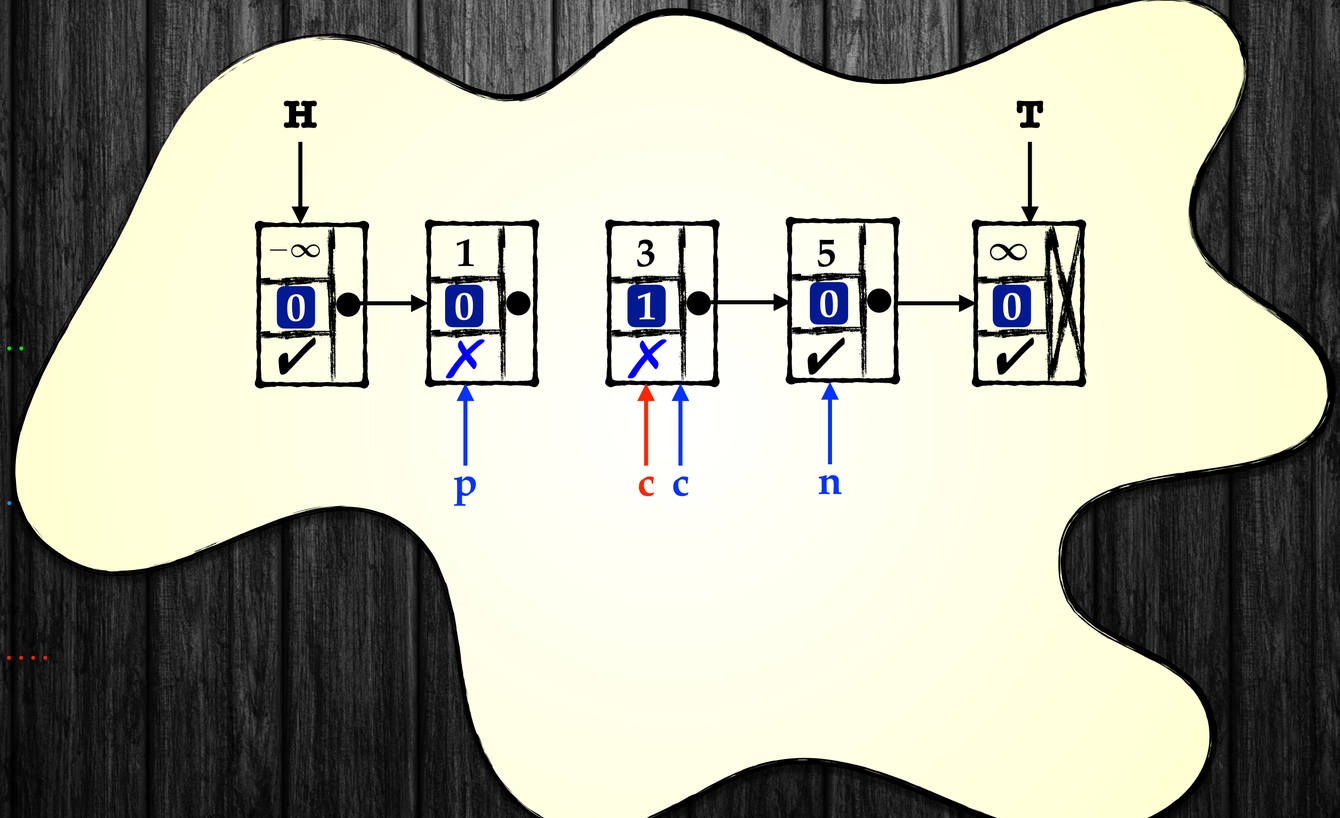
Sets | Lazy Synchronization

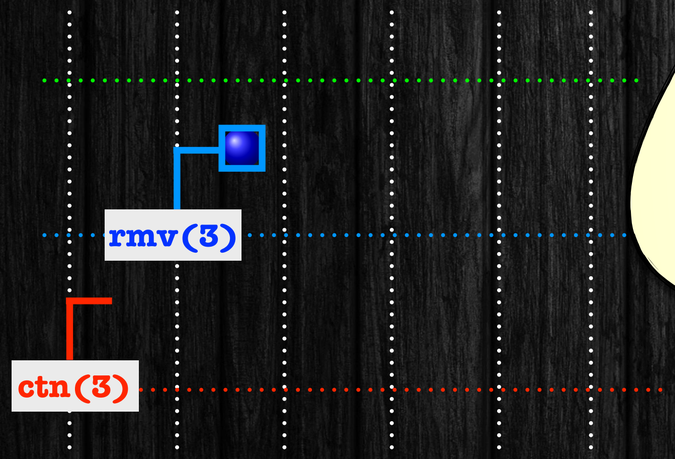
```
ctn(e):
local c, b
1  c := Head;
2  while (c.val < e)
3     c := c.next
4  b := c.mark
5  if (!b && c.val = e)
6     return true;
7  else return false;
```

ctn(3)

not executed yet







rmv(3)

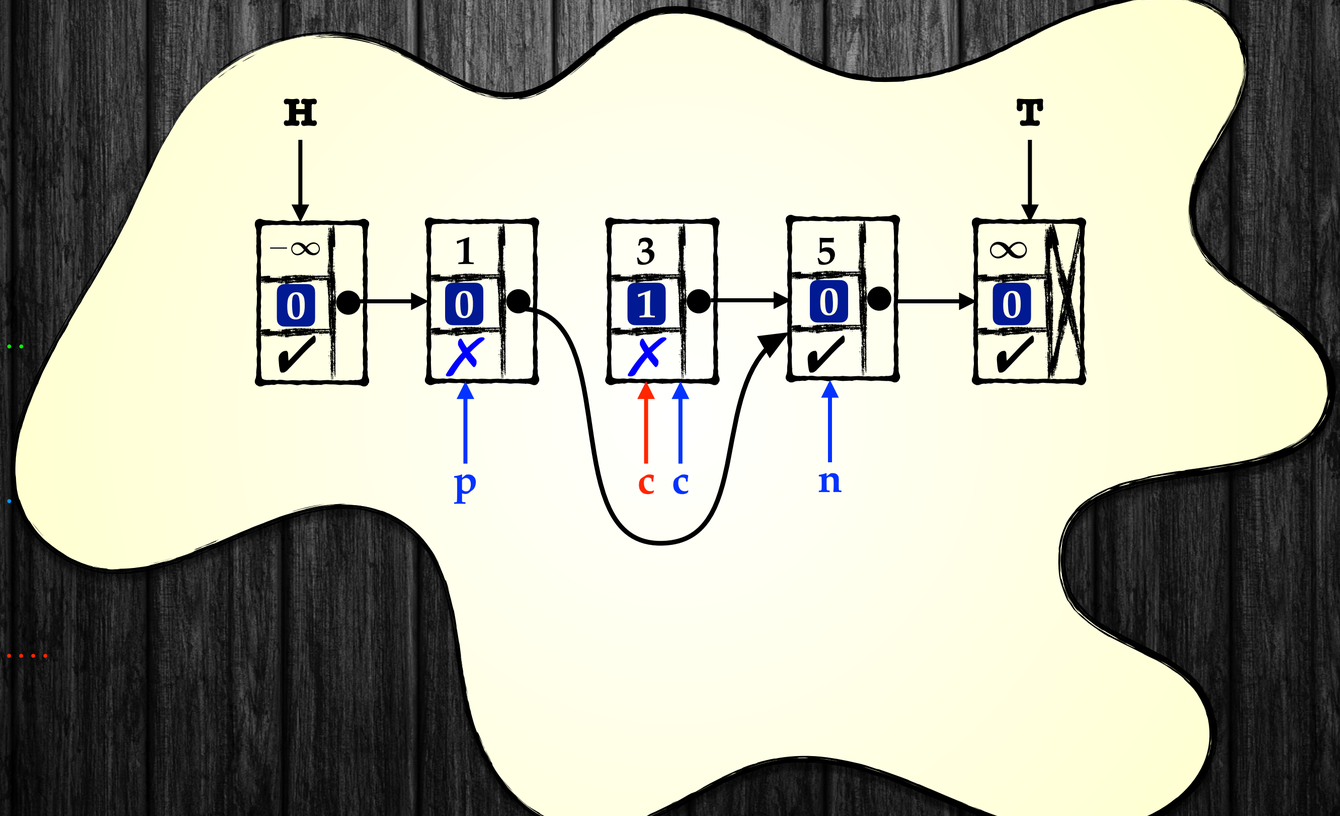
```
rmv(e):
local p, c, n, r
1  (p,c) := locate(e);
2  if (c.val = e)
3     c.mark := true
4     n := c.next;
5     p.next := n;
6     r := true;
7  else r := false;
8  unlock(p);
9  unlock(c);
10 return r;
```

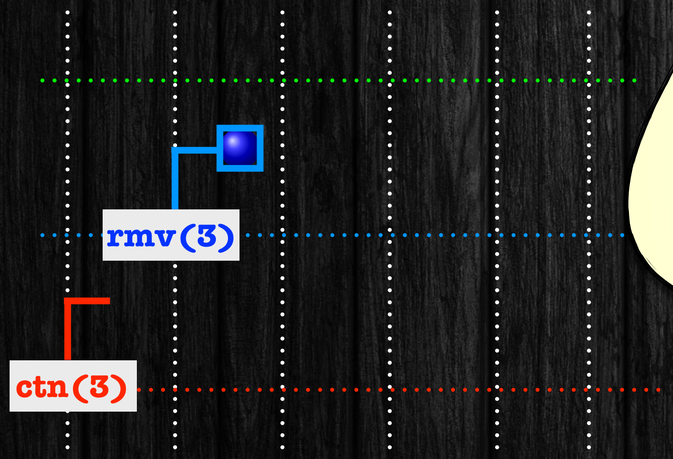
Sets | Lazy Synchronization

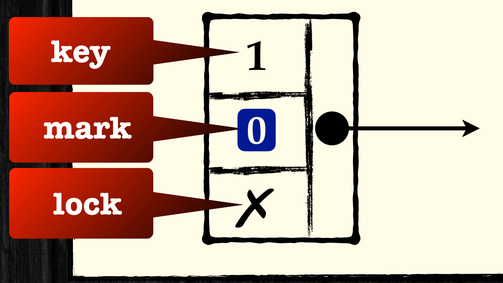
```
ctn(e):
local c, b
1  c := Head;
2  while (c.val < e)
3     c := c.next
4  b := c.mark
5  if (!b && c.val = e)
6     return true;
7  else return false;
```

ctn(3)

not executed yet







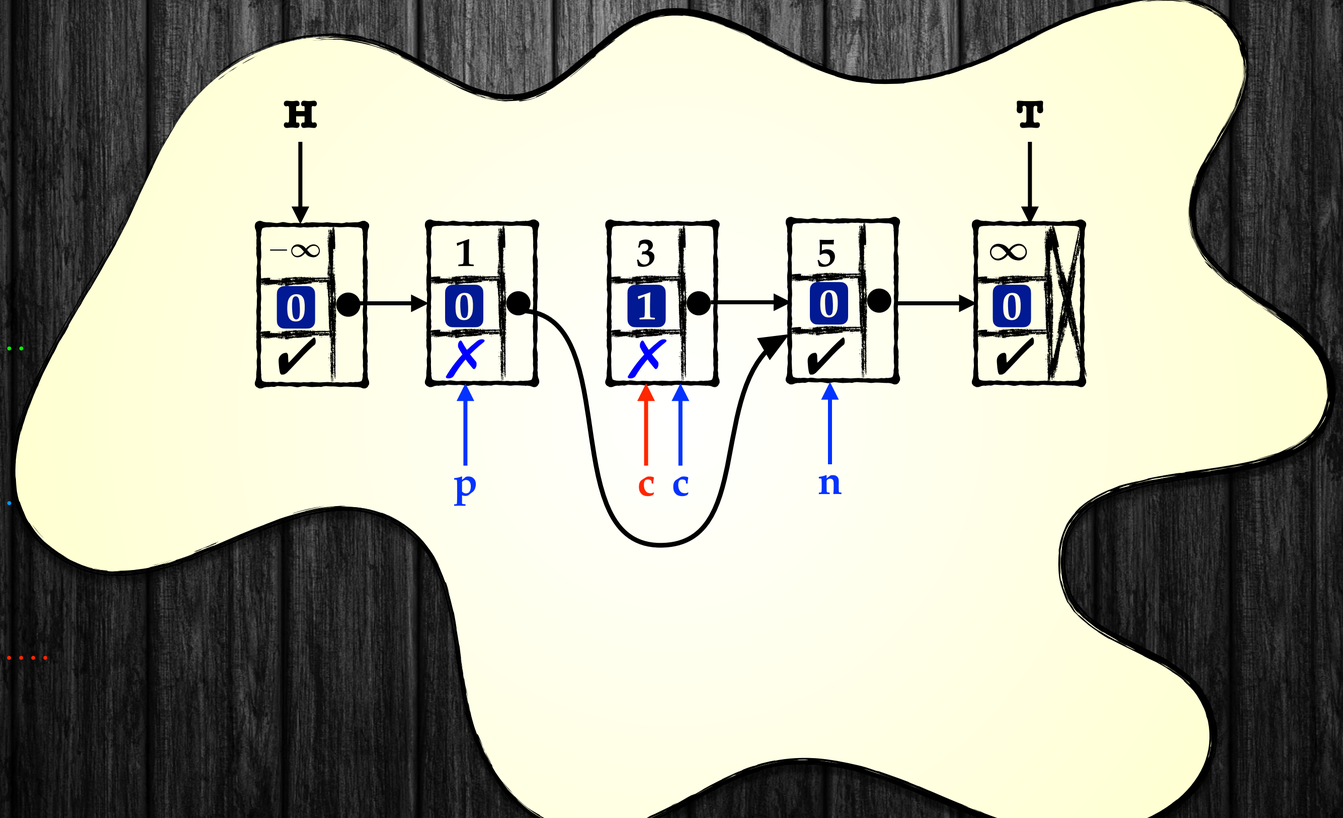
rmv(3)

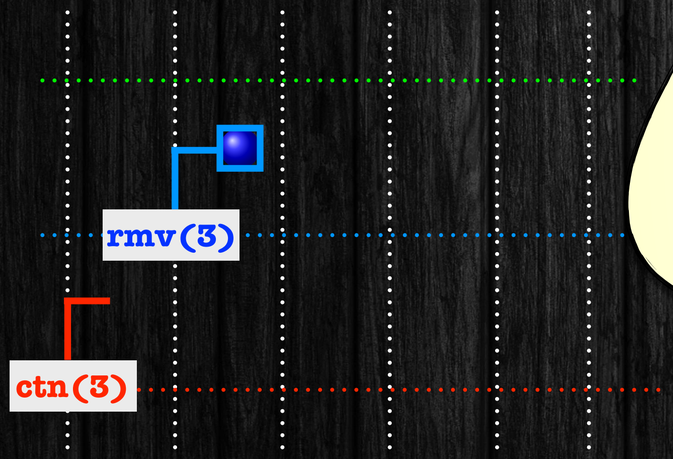
```
rmv(e):
local p, c, n, r
1  (p,c) := locate(e);
2  if (c.val = e)
3     c.mark := true
4     n := c.next;
5     p.next := n;
6     r := true;
7  else r := false;
8  unlock(p);
9  unlock(c);
10 return r;
```

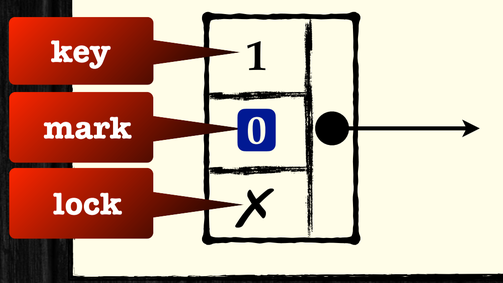
```
ctn(e):
local c, b
1  c := Head;
2  while (c.val < e)
3     c := c.next
4  b := c.mark
5  if (!b && c.val = e)
6     return true;
7  else return false;
```

ctn(3)

not executed yet







rmv(3)

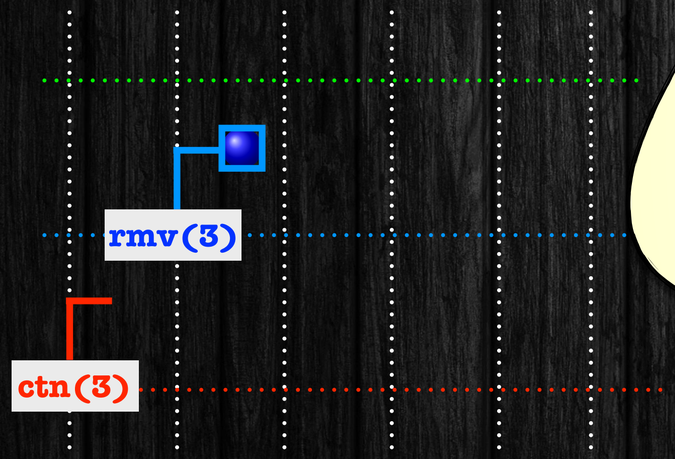
```
rmv(e):
local p, c, n, r
1  (p,c) := locate(e);
2  if (c.val = e)
3     c.mark := true
4     n := c.next;
5     p.next := n;
6     r := true;
7  else r := false;
8  unlock(p);
9  unlock(c);
10 return r;
```

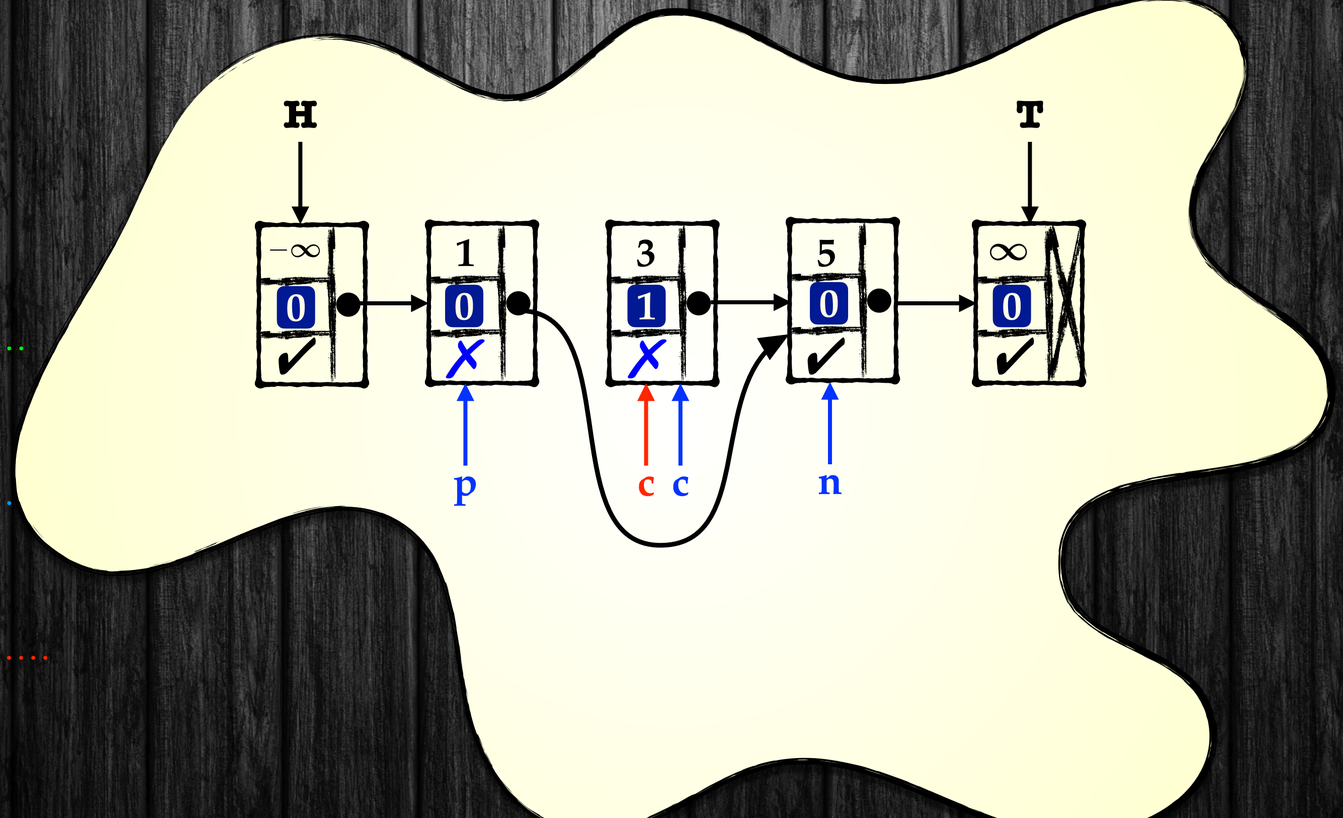
Sets | Lazy Synchronization

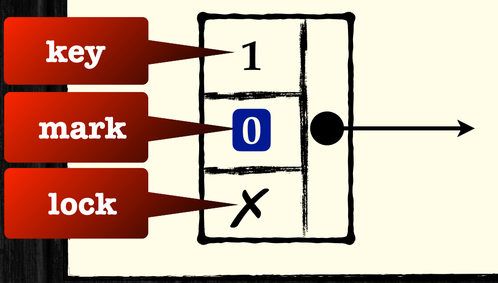
```
ctn(e):
local c, b
1  c := Head;
2  while (c.val < e)
3     c := c.next
4  b := c.mark
5  if (!b && c.val = e)
6     return true;
7  else return false;
```

ctn(3)

not executed yet







rmv(3)

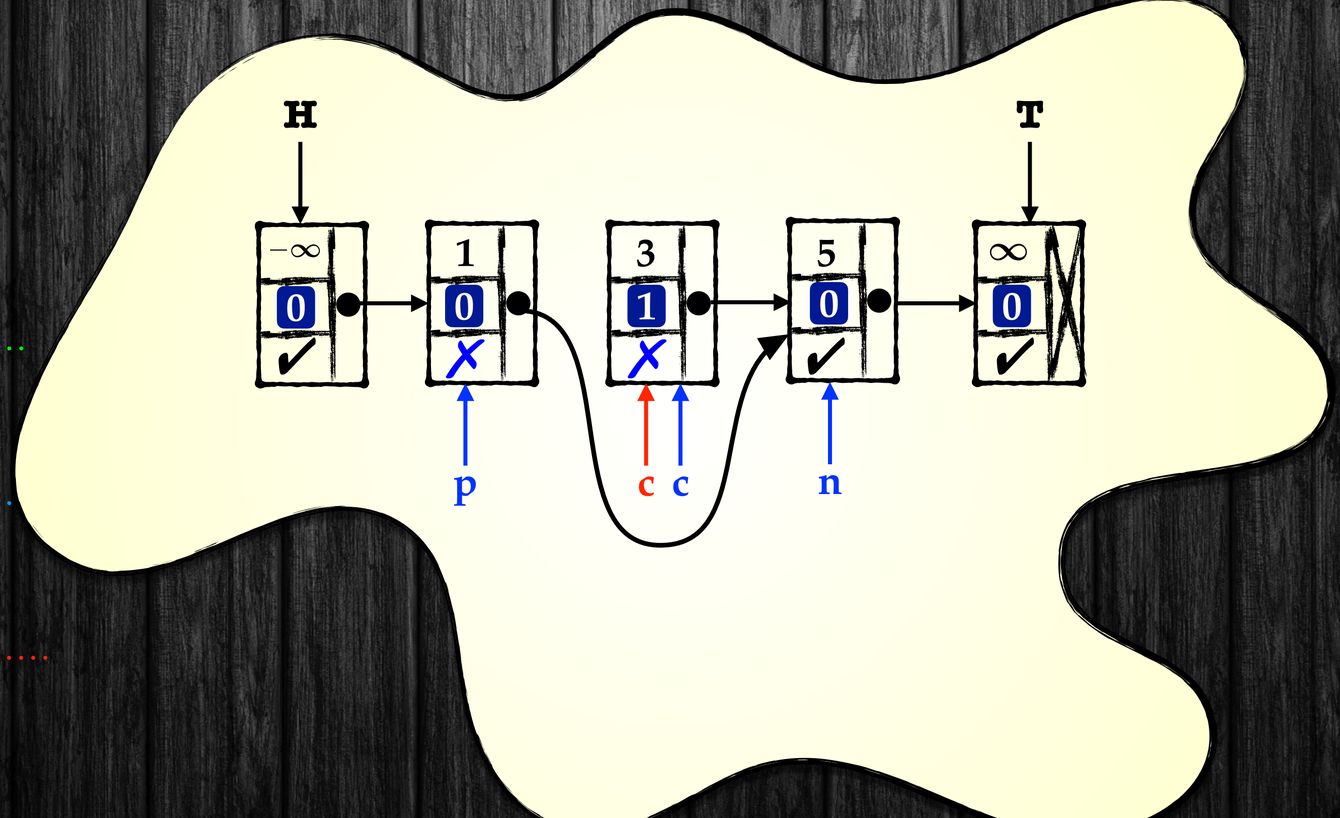
```
rmv(e):
local p, c, n, r
1  (p,c) := locate(e);
2  if (c.val = e)
3     c.mark := true
4     n := c.next;
5     p.next := n;
6     r := true;
7  else r := false;
8  unlock(p);
9  unlock(c);
10 return r;
```

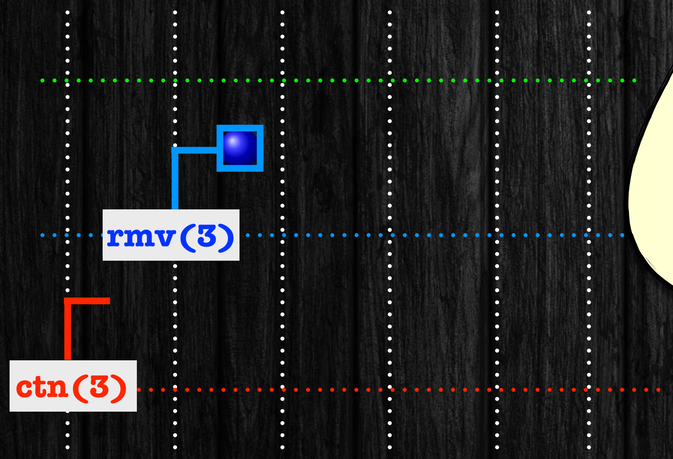
Sets | Lazy Synchronization

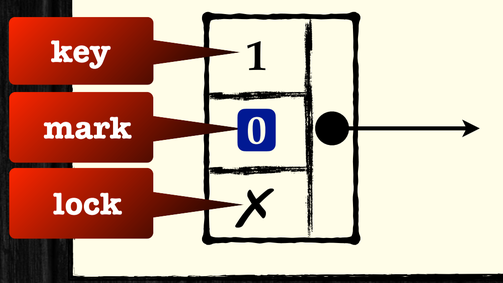
```
ctn(e):
local c, b
1  c := Head;
2  while (c.val < e)
3     c := c.next
4  b := c.mark
5  if (!b && c.val = e)
6     return true;
7  else return false;
```

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;
```

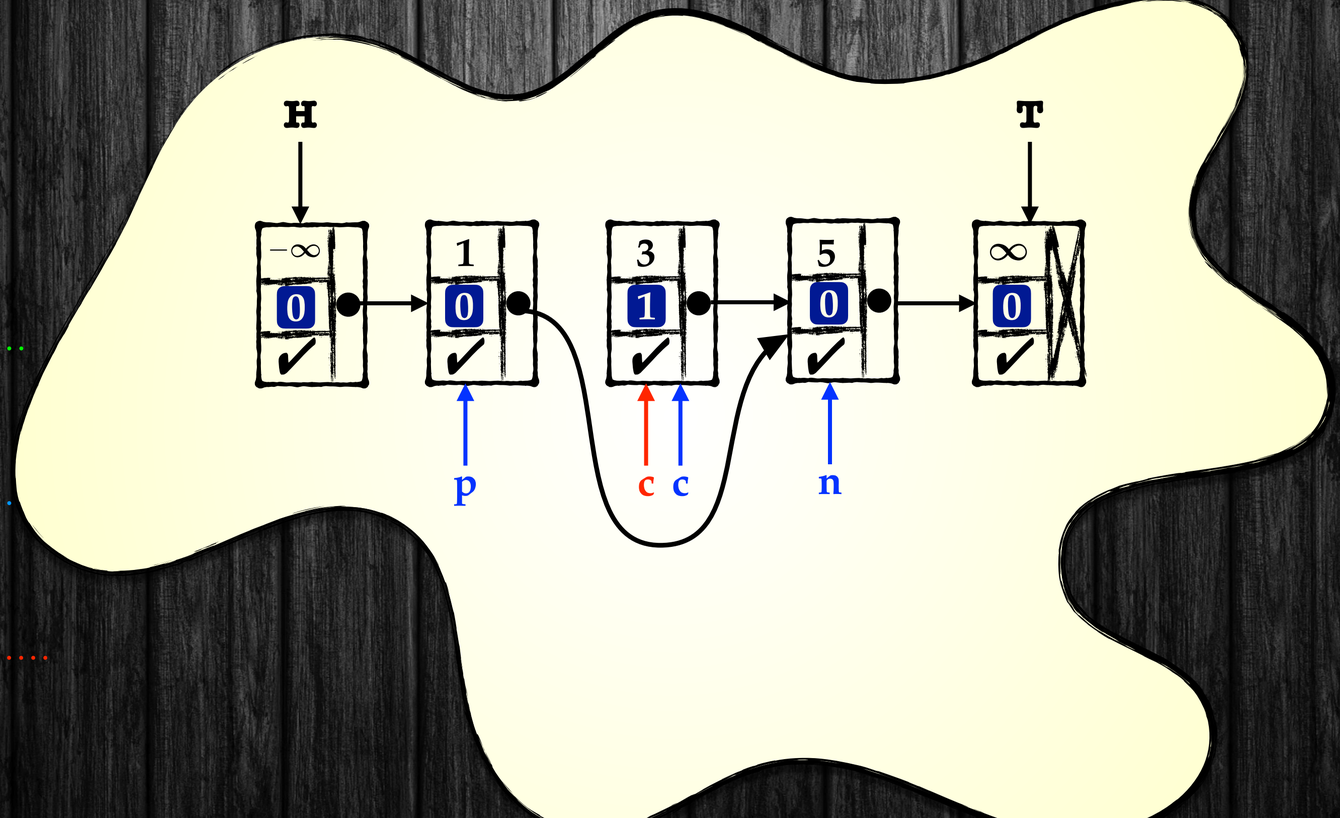
Sets | Lazy Synchronization

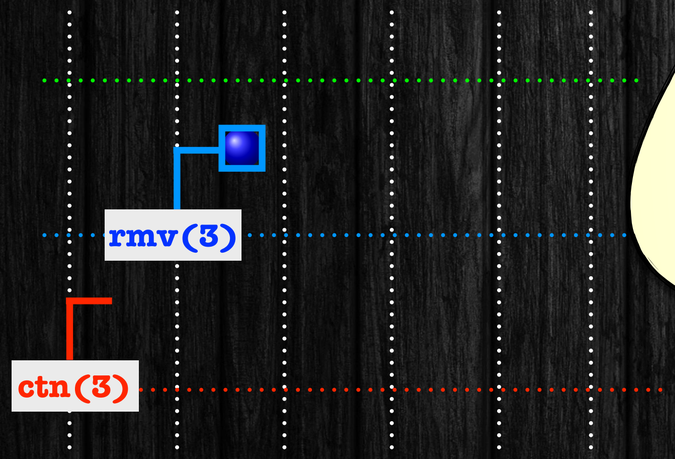
```
ctn(e):
local c, b
1  c := Head;
2  while (c.val < e)
3     c := c.next
4  b := c.mark
5  if (!b && c.val = e)
6     return true;
7  else return false;
```

ctn(3)

not executed yet







rmv(3)

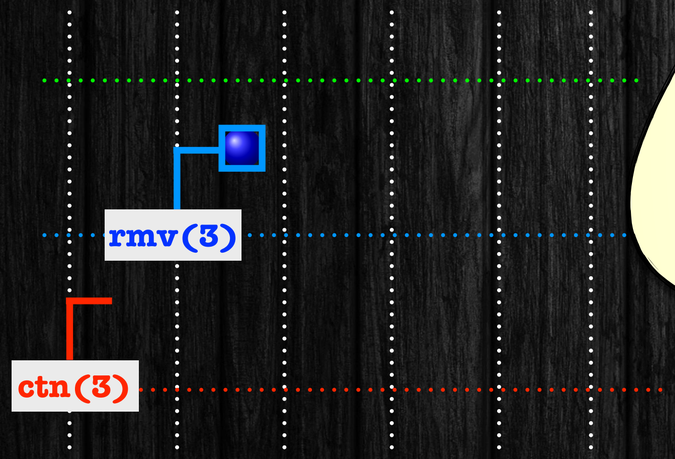
```
rmv(e):
local p, c, n, r
1  (p,c) := locate(e);
2  if (c.val = e)
3     c.mark := true
4     n := c.next;
5     p.next := n;
6     r := true;
7  else r := false;
8  unlock(p);
9  unlock(c);
10 return r;
```

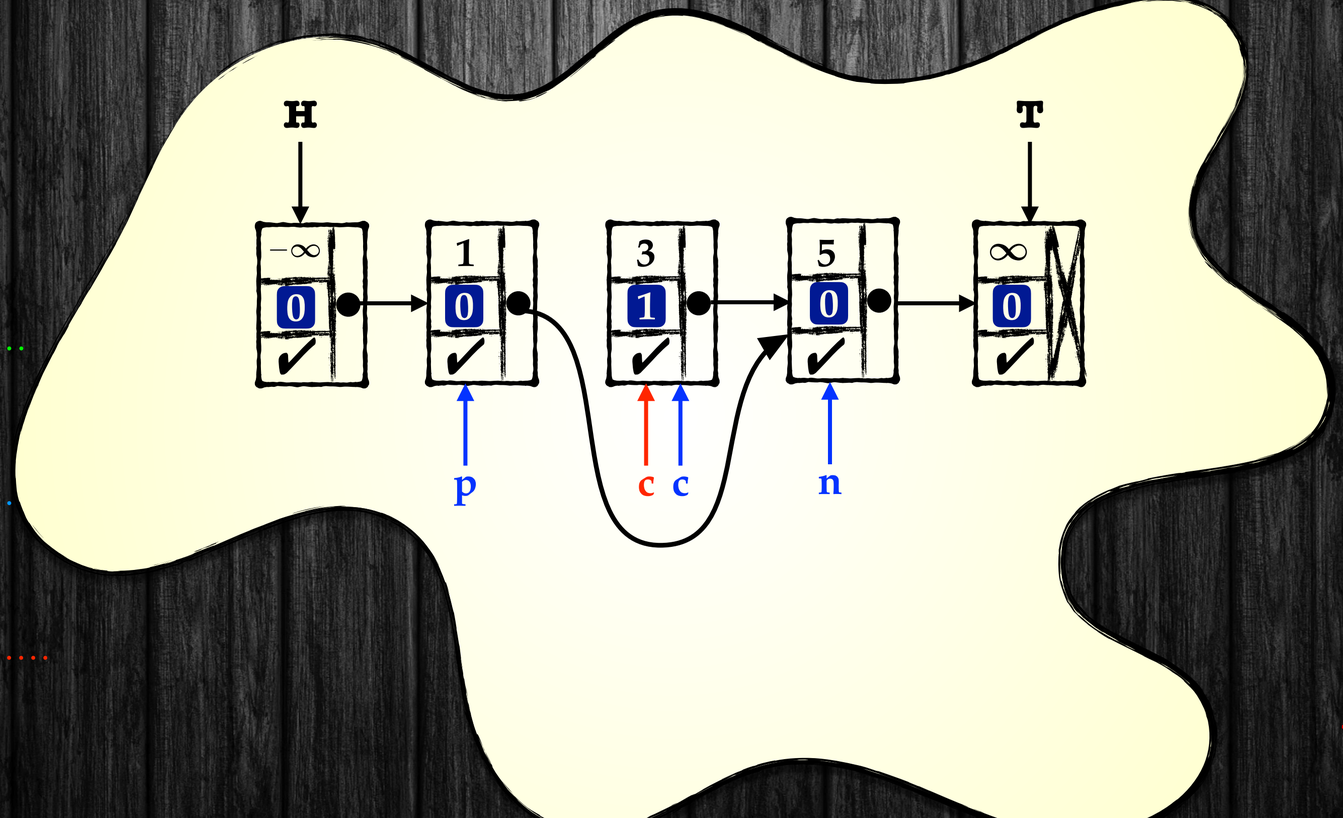
Sets | Lazy Synchronization

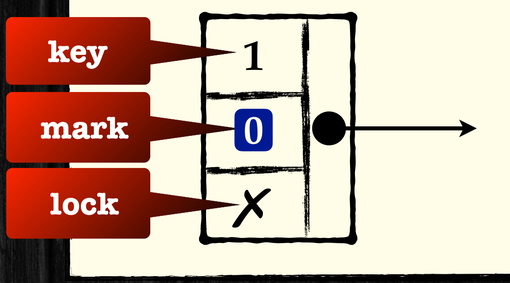
```
ctn(e):
local c, b
1  c := Head;
2  while (c.val < e)
3     c := c.next
4  b := c.mark
5  if (!b && c.val = e)
6     return true;
7  else return false;
```

ctn(3)

not executed yet







rmv(3)

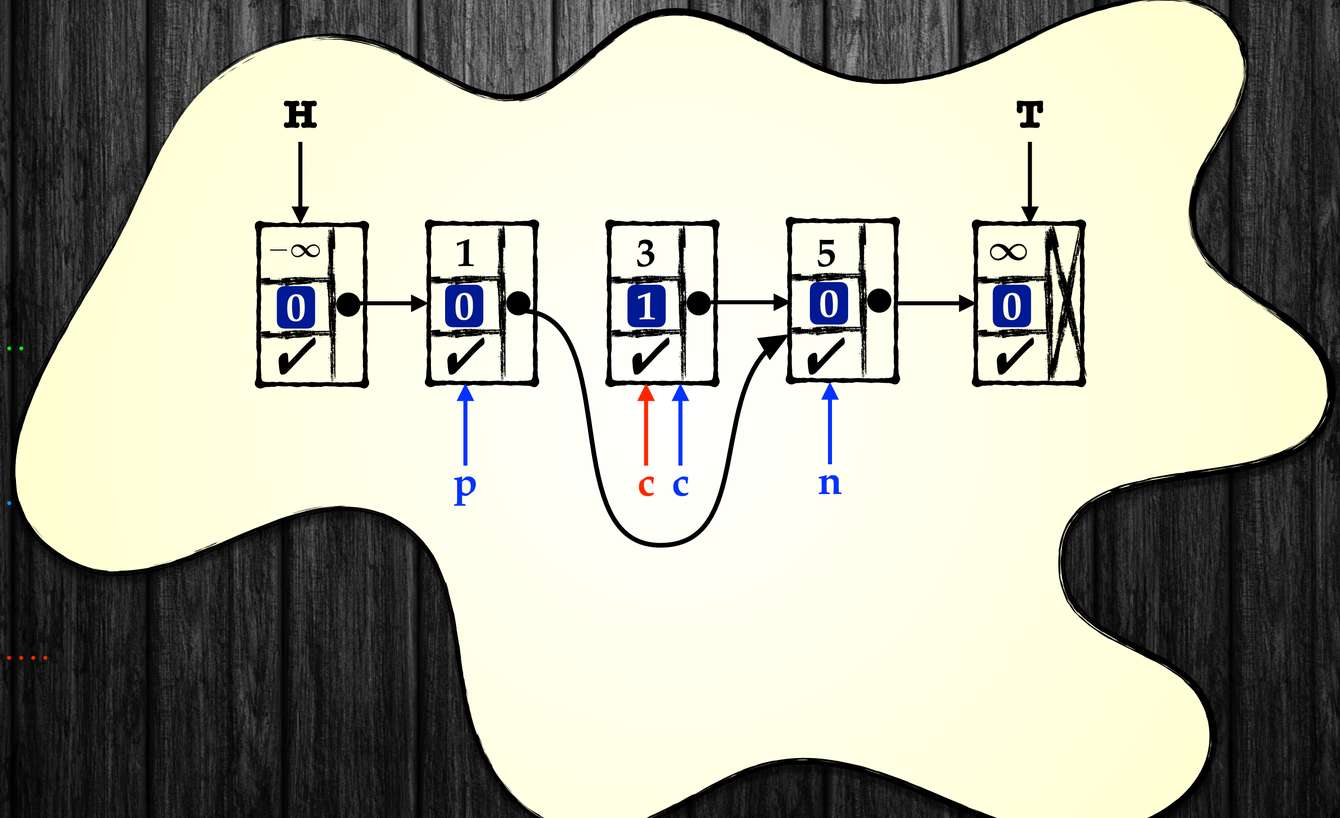
```
rmv(e):
local p, c, n, r
1  (p,c) := locate(e);
2  if (c.val = e)
3     c.mark := true
4     n := c.next;
5     p.next := n;
6     r := true;
7  else r := false;
8  unlock(p);
9  unlock(c);
10 return r;
```

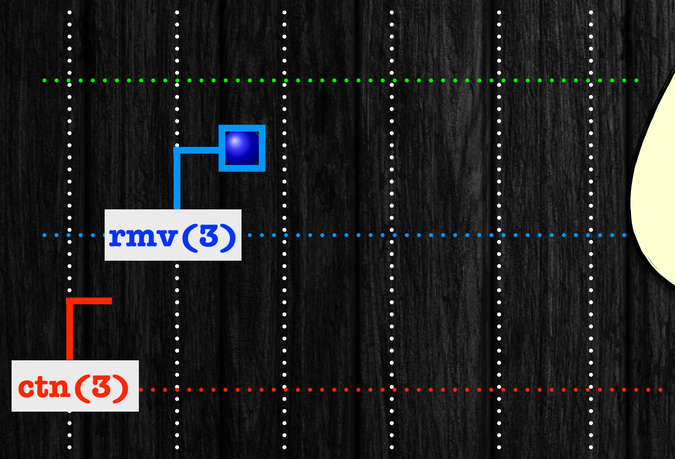
Sets | Lazy Synchronization

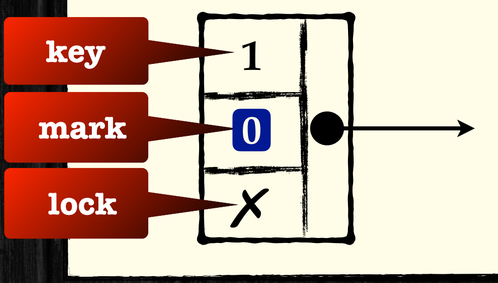
```
ctn(e):
local c, b
1  c := Head;
2  while (c.val < e)
3     c := c.next
4  b := c.mark
5  if (!b && c.val = e)
6     return true;
7  else return false;
```

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;
```

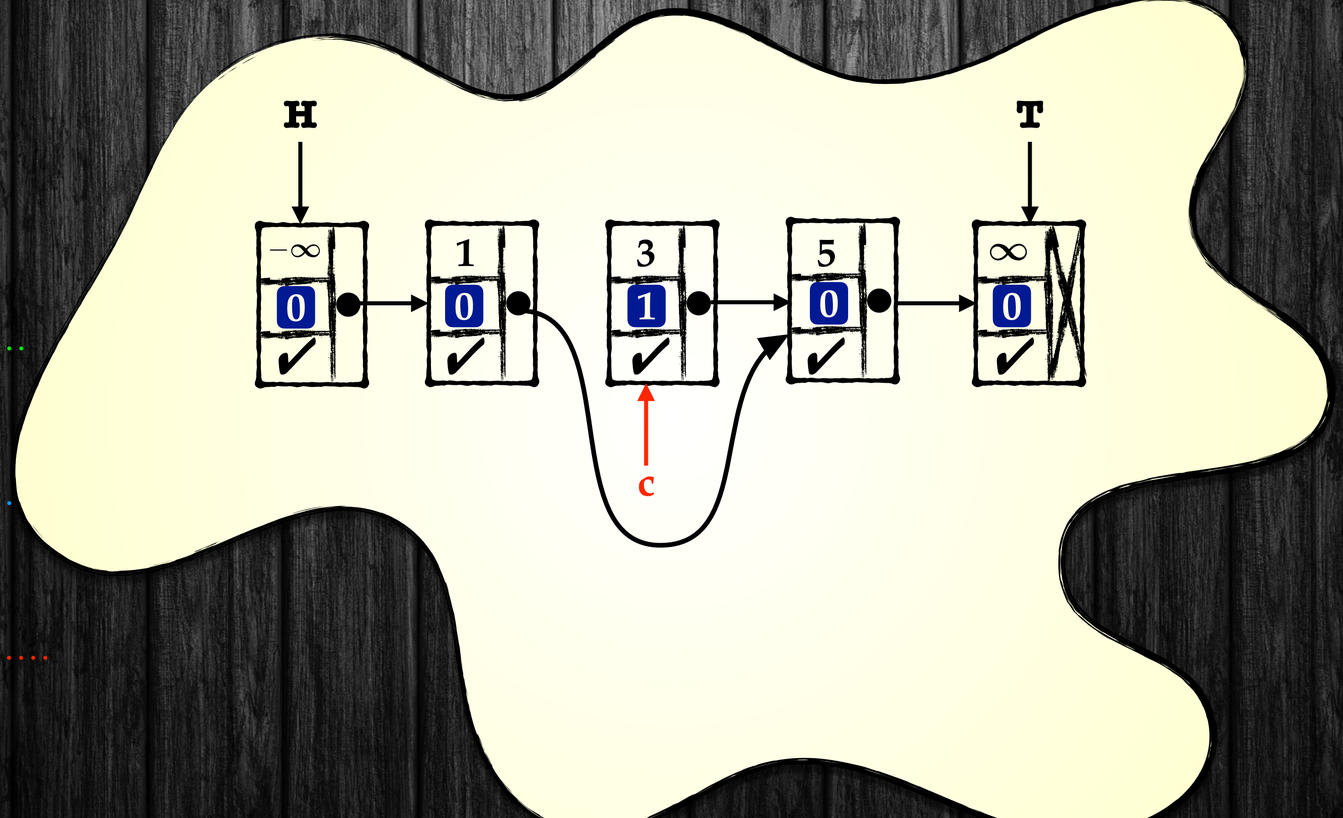
Sets | Lazy Synchronization

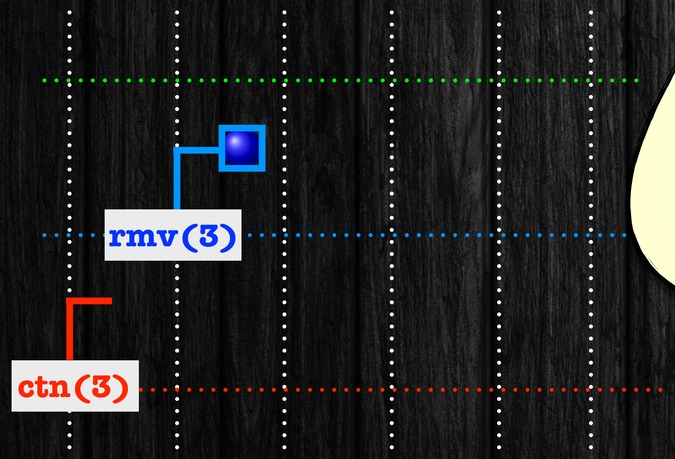
```
ctn(e):
local c, b
1  c := Head;
2  while (c.val < e)
3     c := c.next
4  b := c.mark
5  if (!b && c.val = e)
6     return true;
7  else return false;
```

ctn(3)

not executed yet







rmv(3)

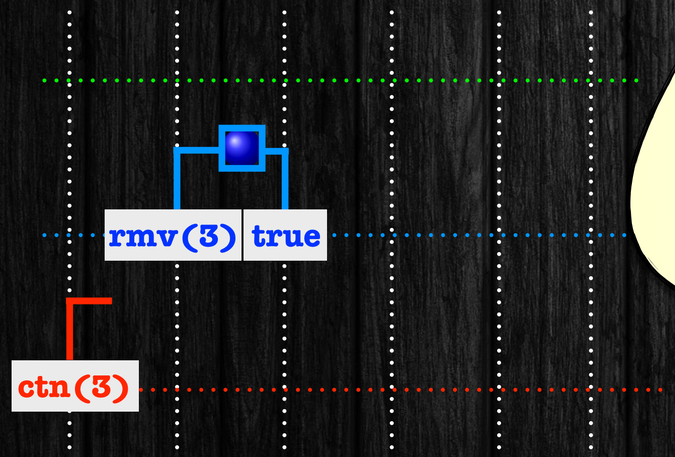
```
rmv(e):
local p, c, n, r
1  (p,c) := locate(e);
2  if (c.val = e)
3     c.mark := true
4     n := c.next;
5     p.next := n;
6     r := true;
7  else r := false;
8  unlock(p);
9  unlock(c);
10 return r;
```

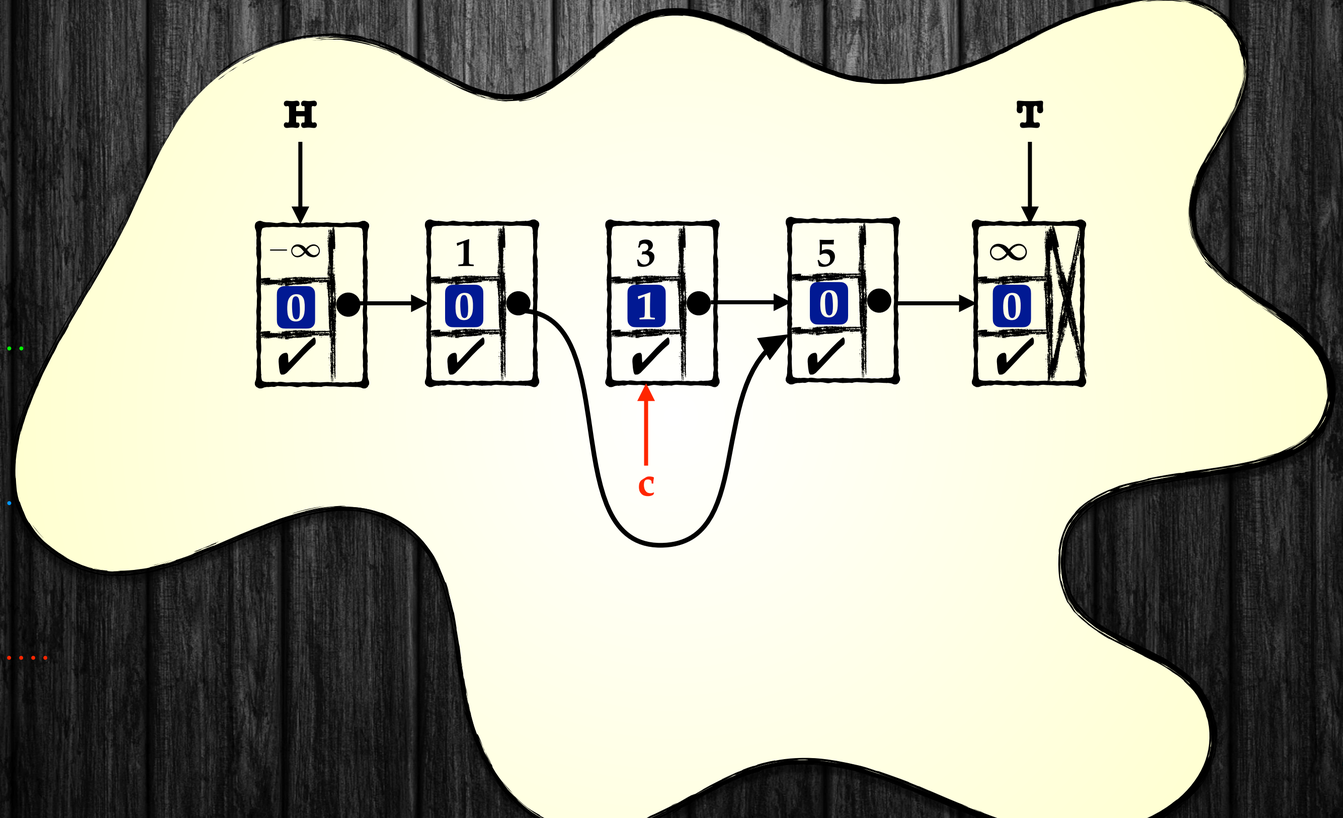
Sets | Lazy Synchronization

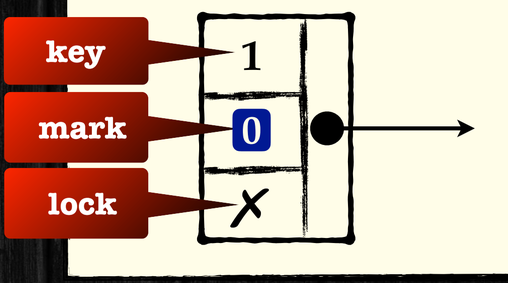
```
ctn(e):
local c, b
1  c := Head;
2  while (c.val < e)
3     c := c.next
4  b := c.mark
5  if (!b && c.val = e)
6     return true;
7  else return false;
```

ctn(3)

not executed yet







rmv(3)

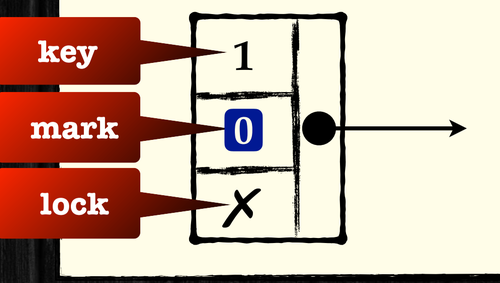
```
rmv(e):
local p, c, n, r
1  (p,c) := locate(e);
2  if (c.val = e)
3     c.mark := true
4     n := c.next;
5     p.next := n;
6     r := true;
7  else r := false;
8  unlock(p);
9  unlock(c);
10 return r;
```

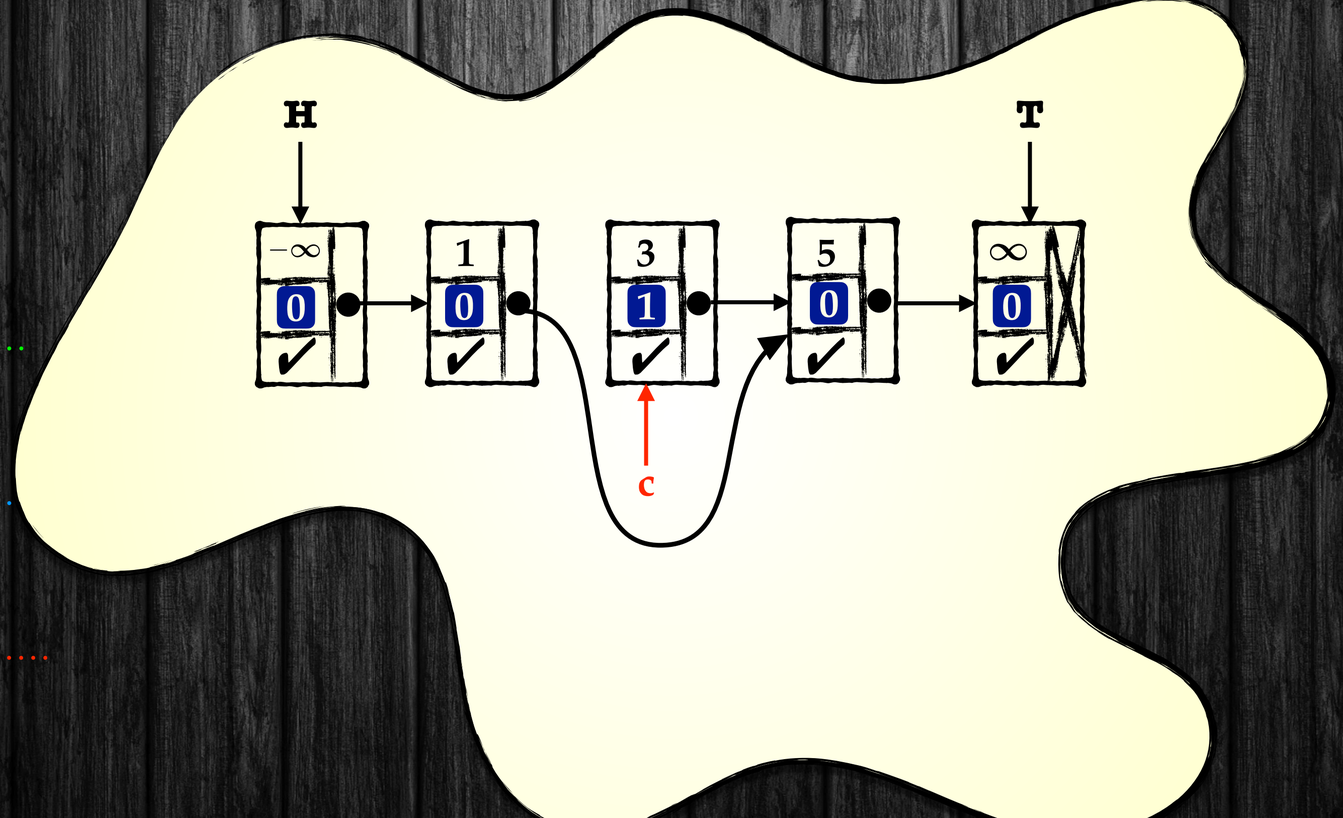
Sets | Lazy Synchronization

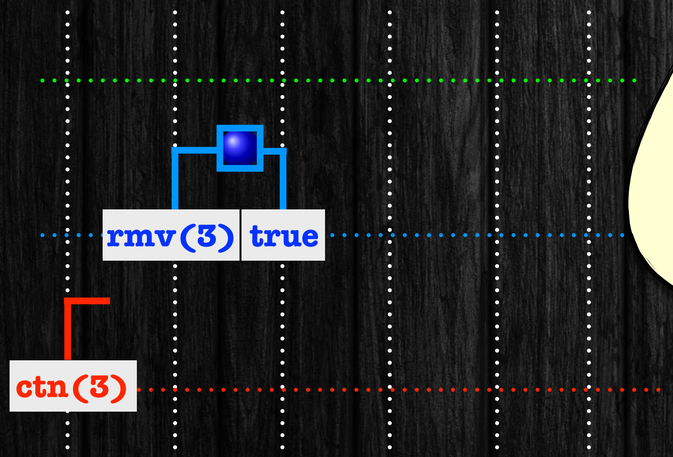
```
ctn(e):
local c, b
1  c := Head;
2  while (c.val < e)
3     c := c.next
4  b := c.mark
5  if (!b && c.val = e)
6     return true;
7  else return false;
```

ctn(3)

not executed yet



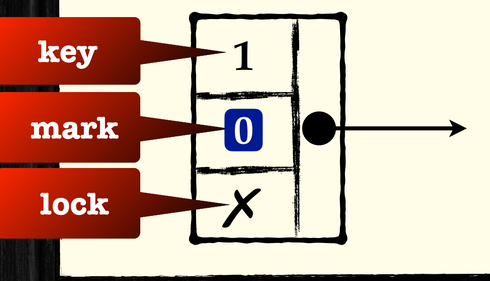


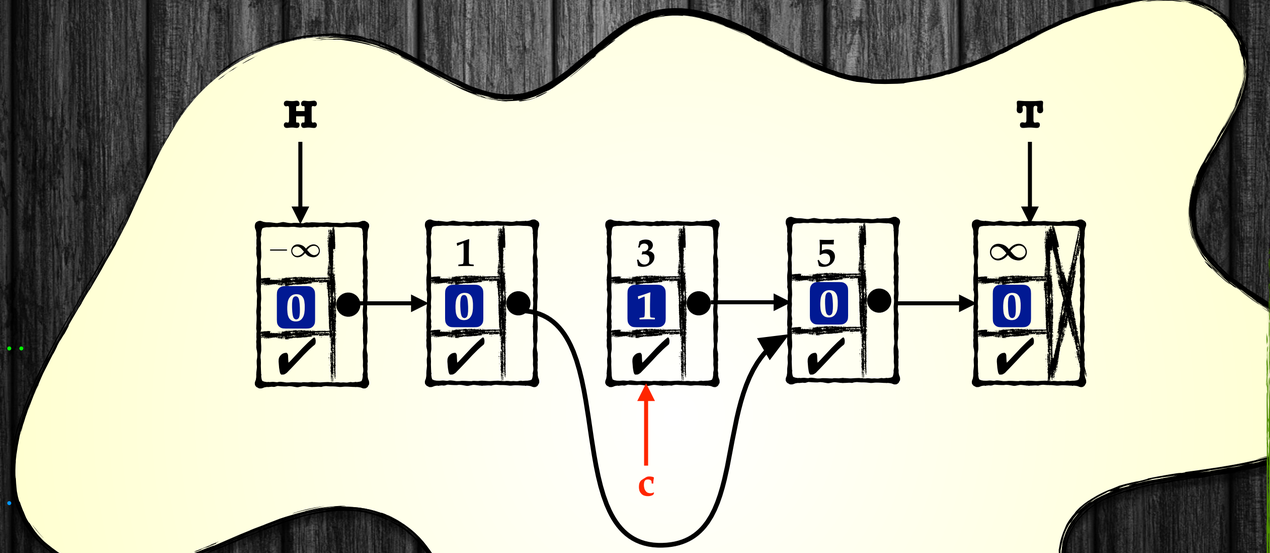


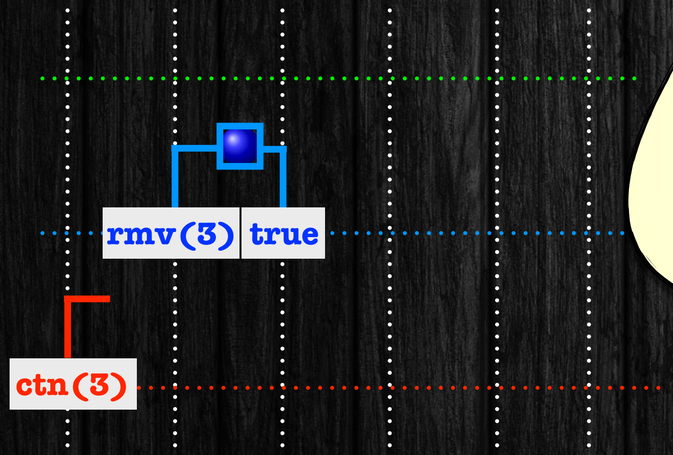
```
ctn(e):
local c, b
1  c := Head;
2  while (c.val < e)
3     c := c.next
4  b := c.mark
5  if (!b && c.val = e)
6     return true;
7  else return false;
```

ctn(3)

not executed yet







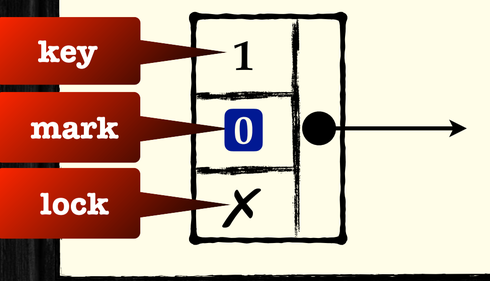
add(3)

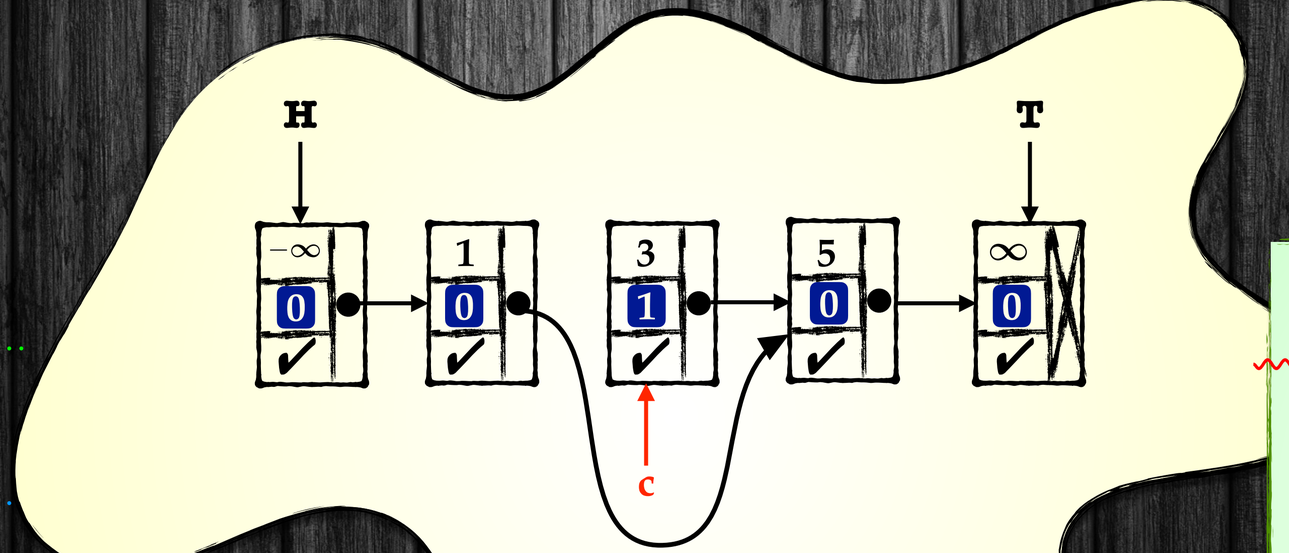
```
add(e):
   local p, c, n, r
   1  (p,c) := locate(e);
   2  if (c.val <> e)
   3     n := new Node(0,e,c,false);
   4     p.next := n;
   5     r := true;
   6  else r := false;
   7  unlock(p);
   8  unlock(c);
   9  return r;
```

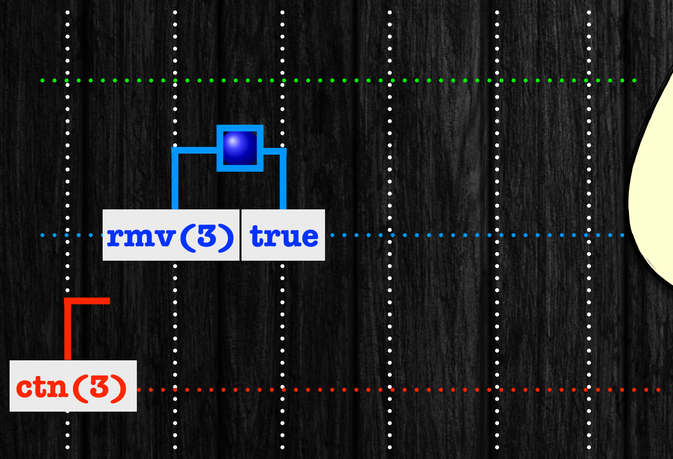
```
ctn(e):
local c, b
1  c := Head;
2  while (c.val < e)
3     c := c.next
4  b := c.mark
5  if (!b && c.val = e)
6     return true;
7  else return false;
```

ctn(3)

not executed yet







add(3)

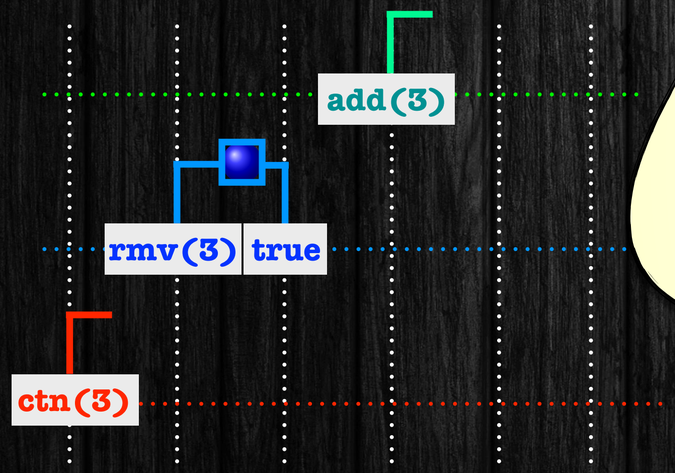
```
add(e):
   local p, c, n, r
1  (p,c) := locate(e);
2  if (c.val <> e)
3     n := new Node(0,e,c,false);
4     p.next := n;
5     r := true;
6  else r := false;
7  unlock(p);
8  unlock(c);
9  return r;
```

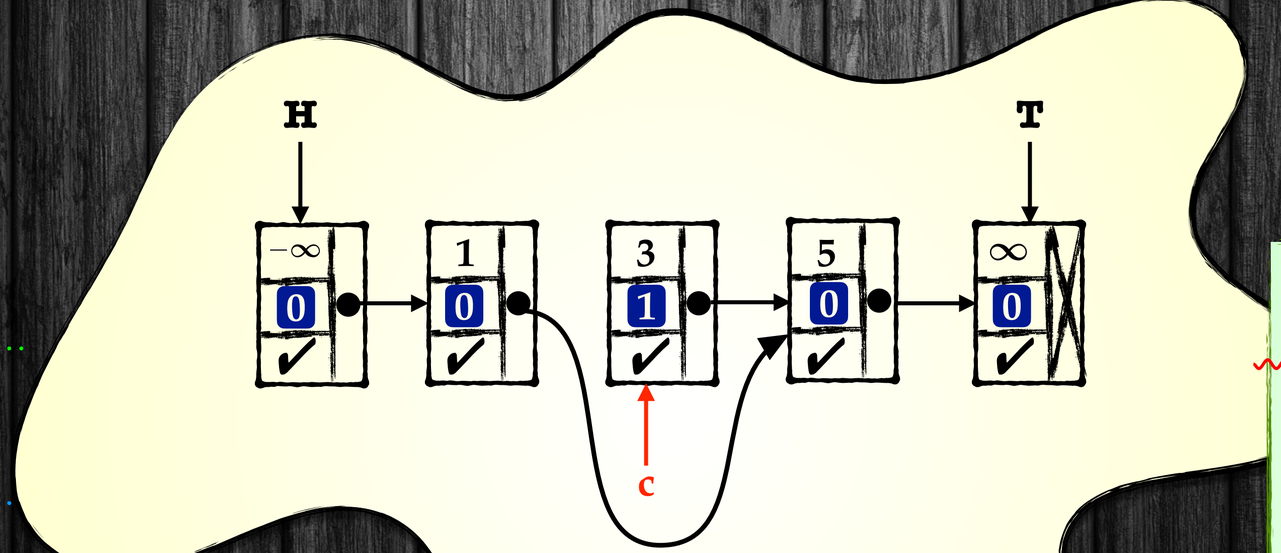
Sets | Lazy Synchronization

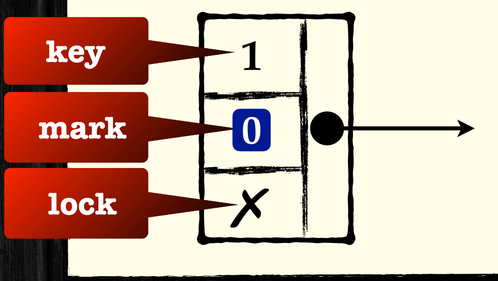
```
ctn(e):
local c, b
1  c := Head;
2  while (c.val < e)
3     c := c.next
4  b := c.mark
5  if (!b && c.val = e)
6     return true;
7  else return false;
```

ctn(3)

not executed yet







add(3)

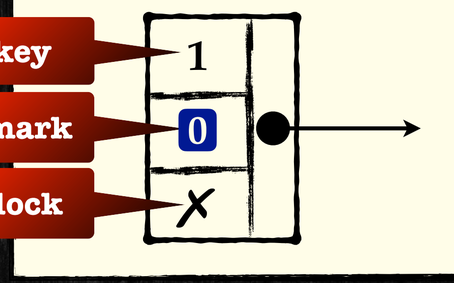
```
add(e):
   local p, c, n, r
1  (p,c) := locate(e);
2  if (c.val <> e)
3     n := new Node(0,e,c,false);
4     p.next := n;
5     r := true;
6  else r := false;
7  unlock(p);
8  unlock(c);
9  return r;
```

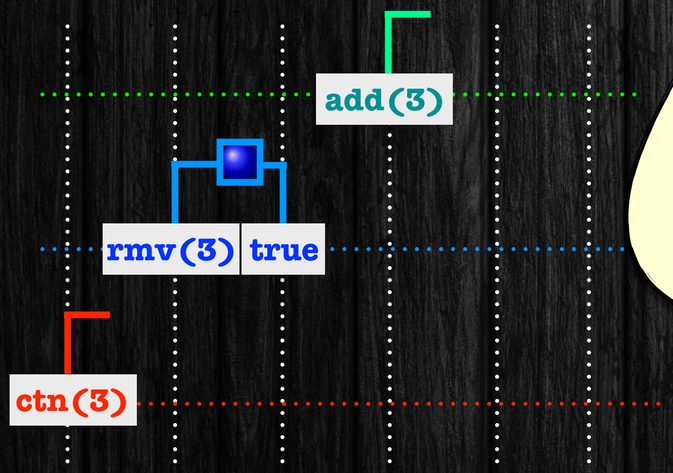
Sets | Lazy Synchronization

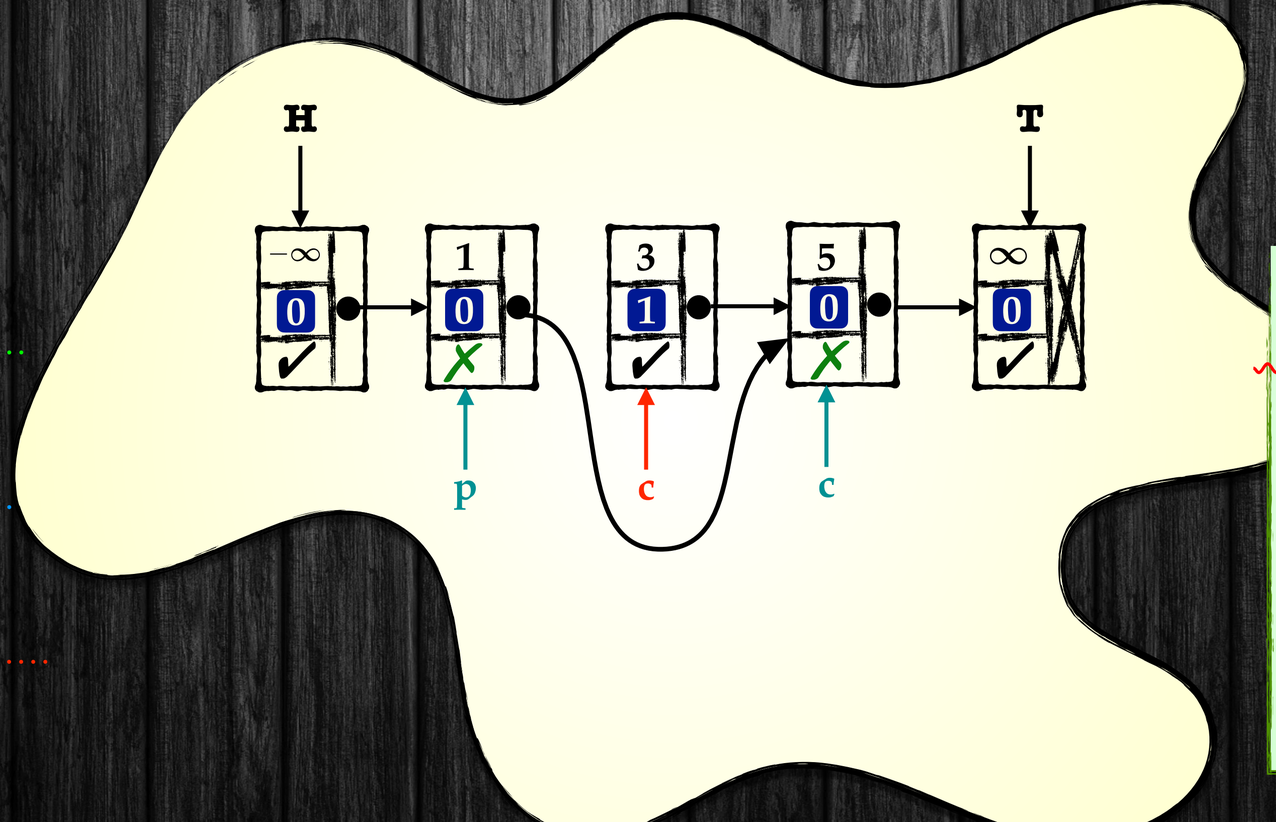
```
ctn(e):
local c, b
1  c := Head;
2  while (c.val < e)
3     c := c.next
4  b := c.mark
5  if (!b && c.val = e)
6     return true;
7  else return false;
```

ctn(3)

not executed yet







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;
```

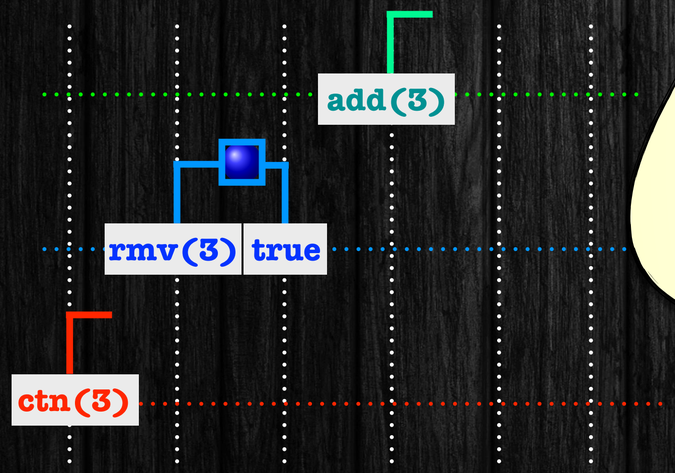
Sets | Lazy Synchronization

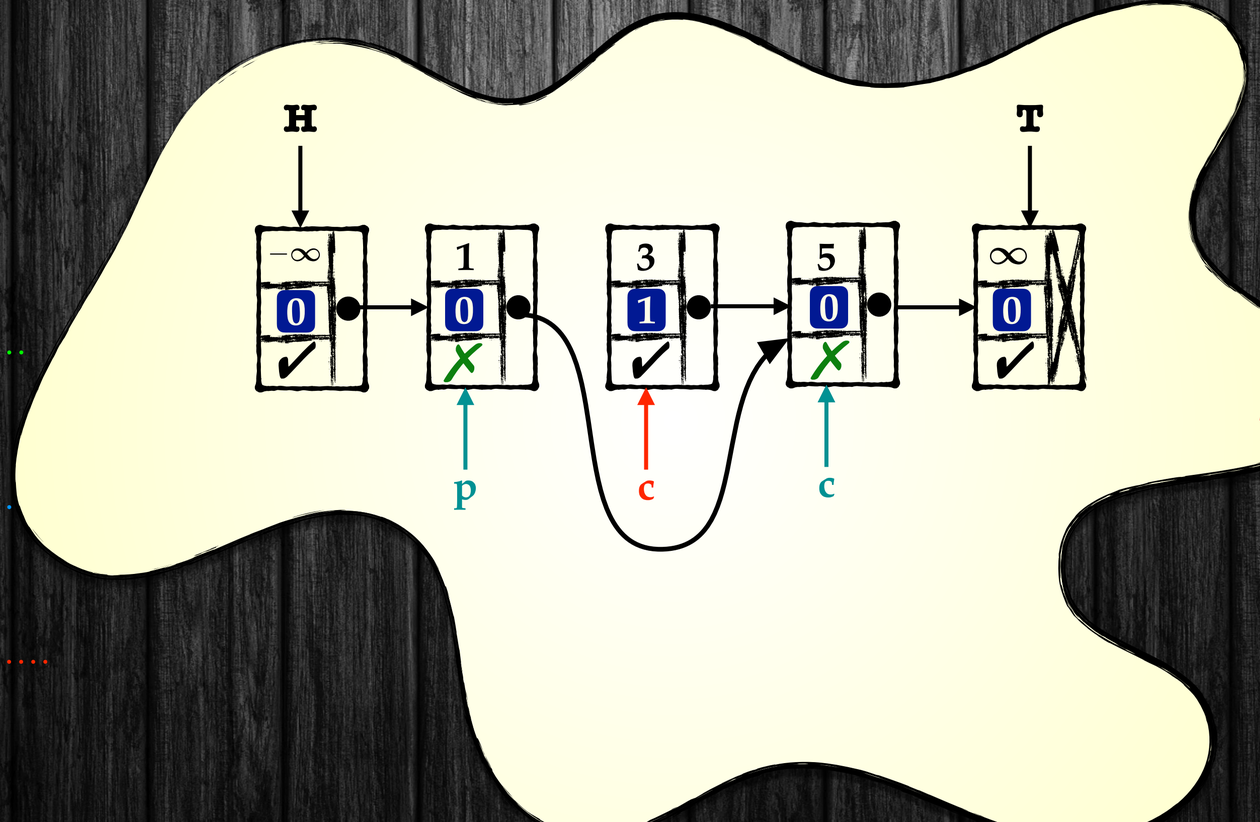
```
ctn(e):
local c, b
1  c := Head;
2  while (c.val < e)
3     c := c.next
4  b := c.mark
5  if (!b && c.val = e)
6     return true;
7  else return false;
```

ctn(3)

not executed yet







add(3)

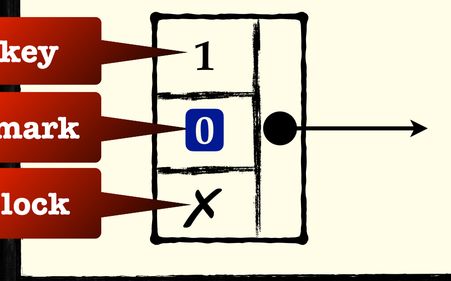
```
add(e):
   local p, c, n, r
   1  (p,c) := locate(e);
   2  if (c.val <> e)
   3     n := new Node(0,e,c,false);
   4     p.next := n;
   5     r := true;
   6  else r := false;
   7  unlock(p);
   8  unlock(c);
   9  return r;
```

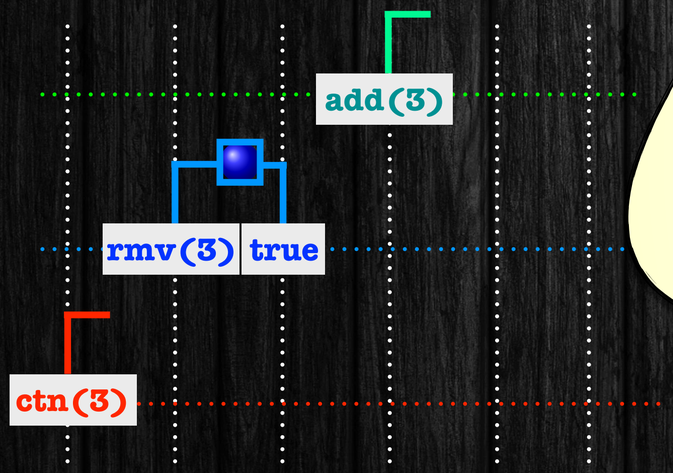
Sets | Lazy Synchronization

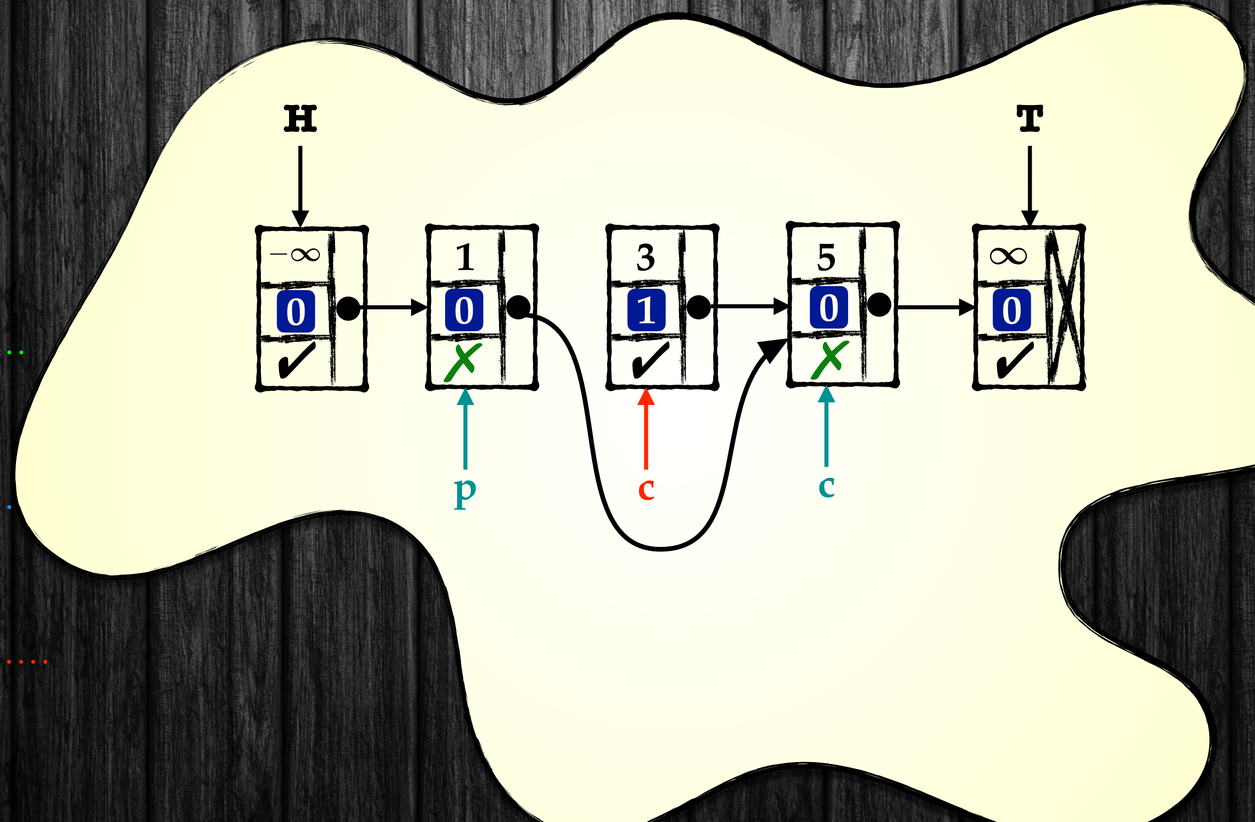
```
ctn(e):
local c, b
1  c := Head;
2  while (c.val < e)
3     c := c.next
4  b := c.mark
5  if (!b && c.val = e)
6     return true;
7  else return false;
```

ctn(3)

not executed yet







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;
```

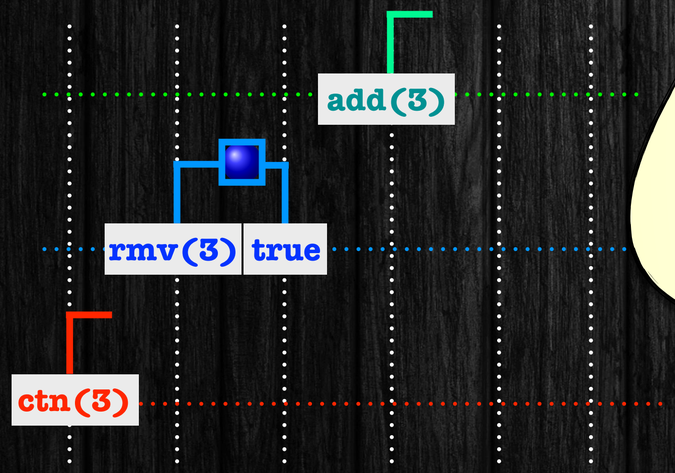
Sets | Lazy Synchronization

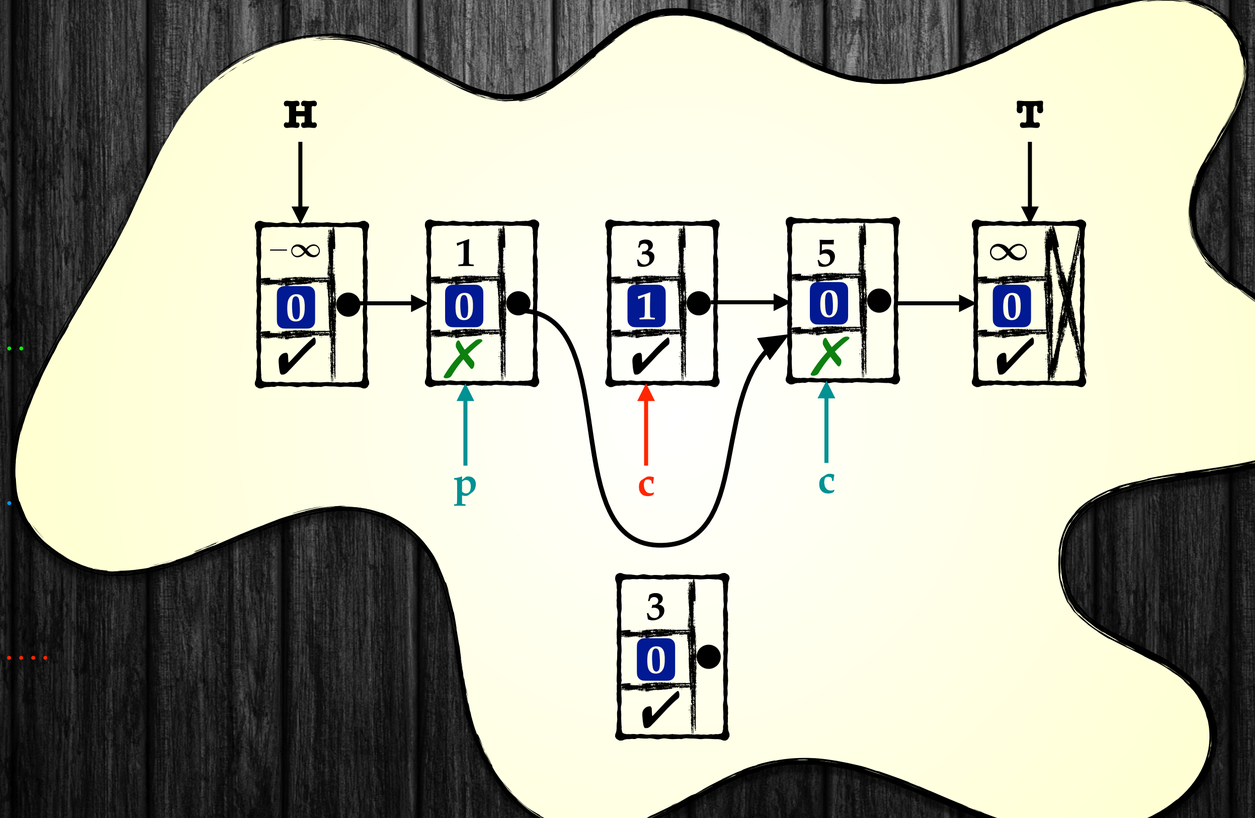
```
ctn(e):
local c, b
1  c := Head;
2  while (c.val < e)
3     c := c.next
4  b := c.mark
5  if (!b && c.val = e)
6     return true;
7  else return false;
```

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;
```

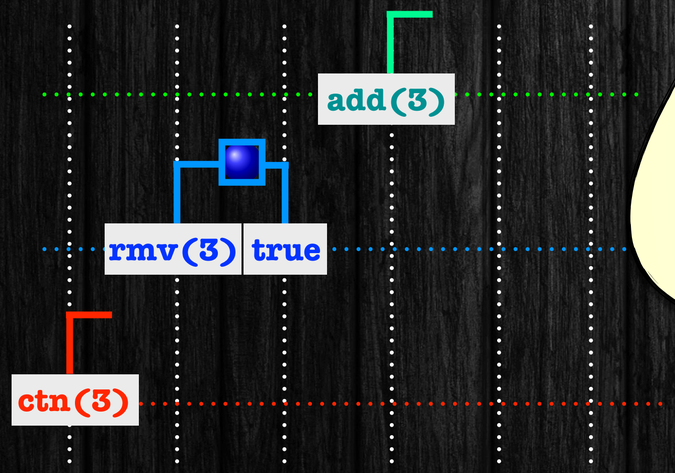
Sets | Lazy Synchronization

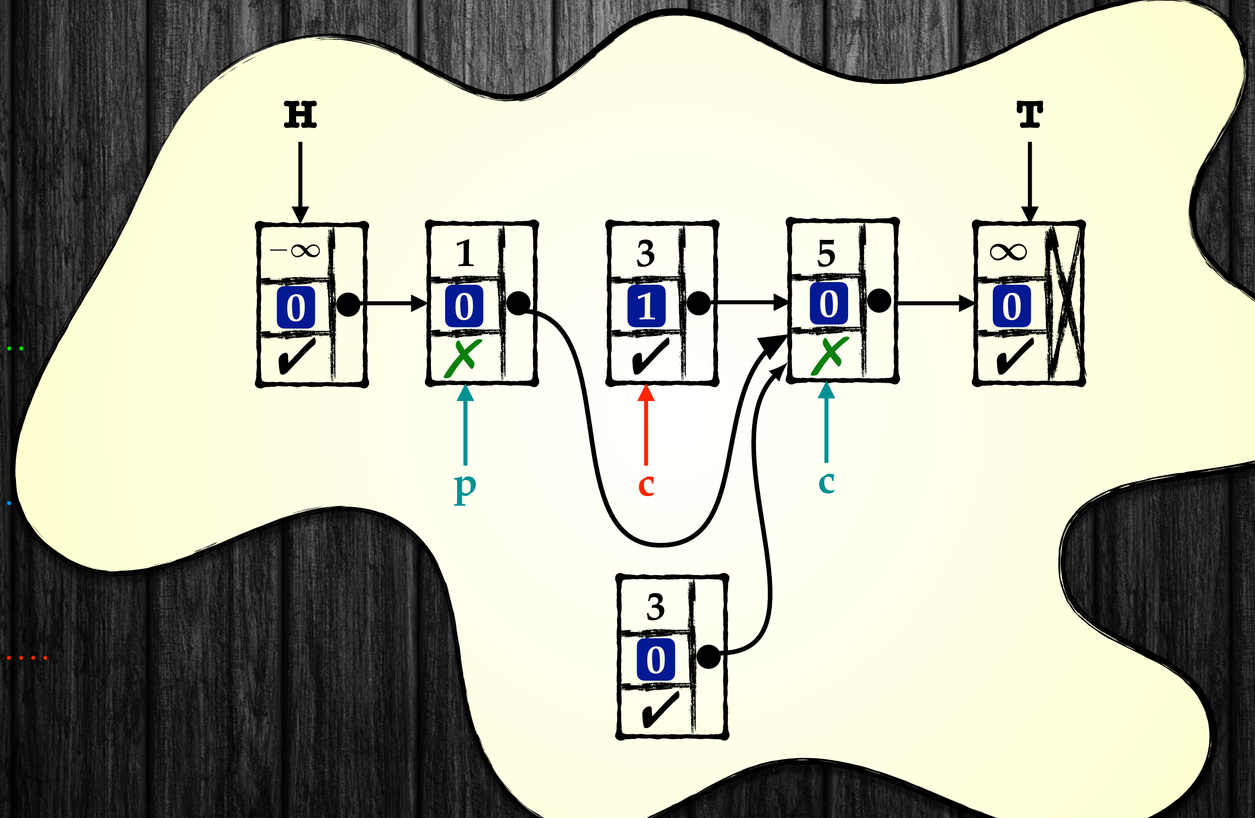
```
ctn(e):
local c, b
1  c := Head;
2  while (c.val < e)
3     c := c.next
4  b := c.mark
5  if (!b && c.val = e)
6     return true;
7  else return false;
```

ctn(3)

not executed yet







add(3)

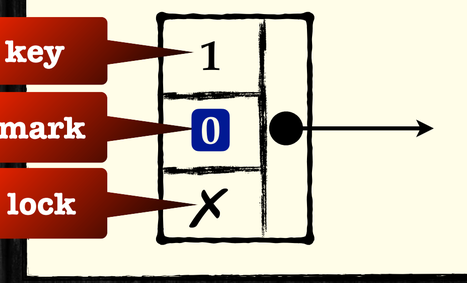
```
add(e):
   local p, c, n, r
   1  (p,c) := locate(e);
   2  if (c.val <> e)
   3     n := new Node(0,e,c,false);
   4     p.next := n;
   5     r := true;
   6  else r := false;
   7  unlock(p);
   8  unlock(c);
   9  return r;
```

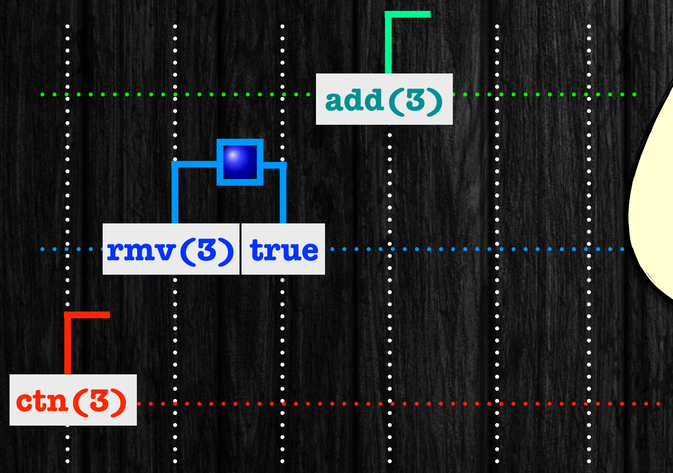
Sets | Lazy Synchronization

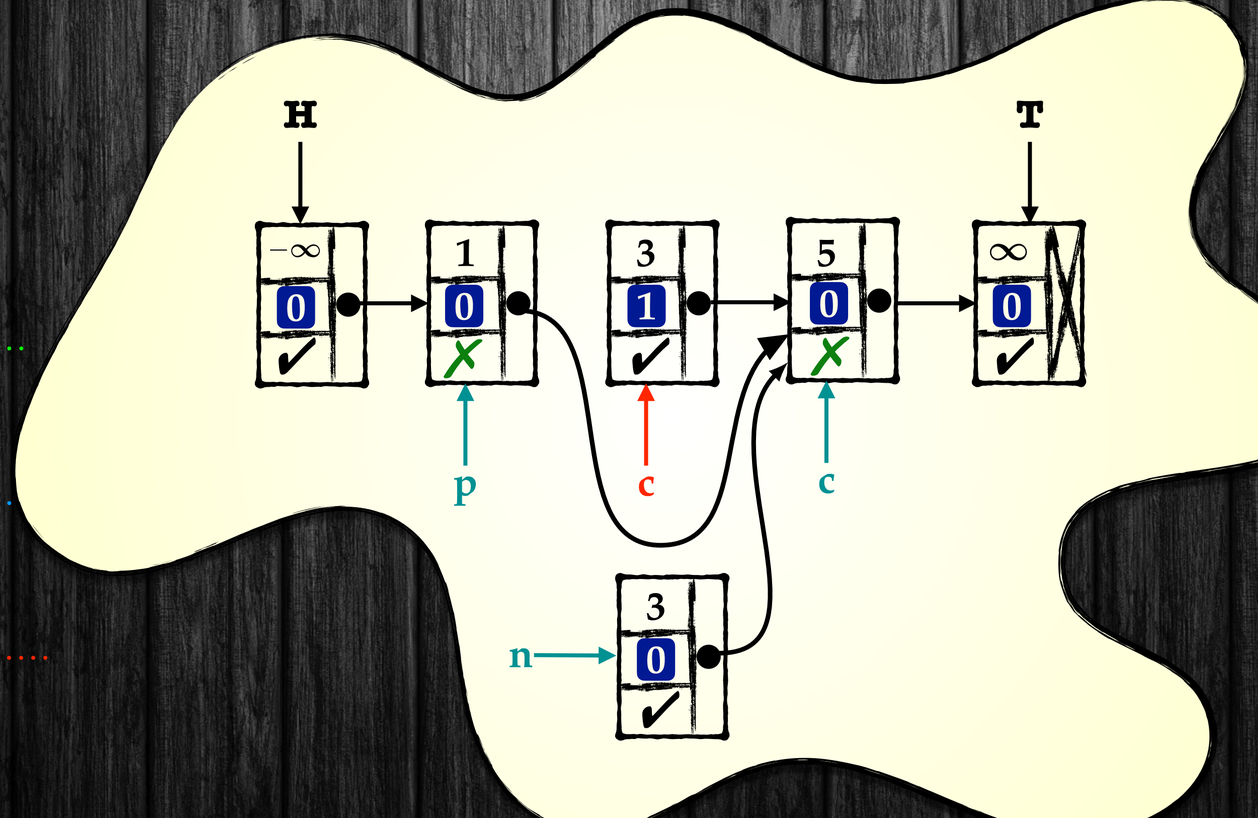
```
ctn(e):
local c, b
1  c := Head;
2  while (c.val < e)
3     c := c.next
4  b := c.mark
5  if (!b && c.val = e)
6     return true;
7  else return false;
```

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

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;
```

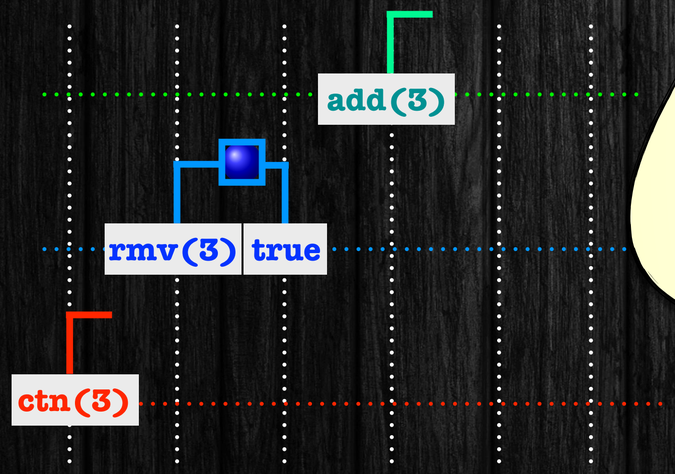
Sets | Lazy Synchronization

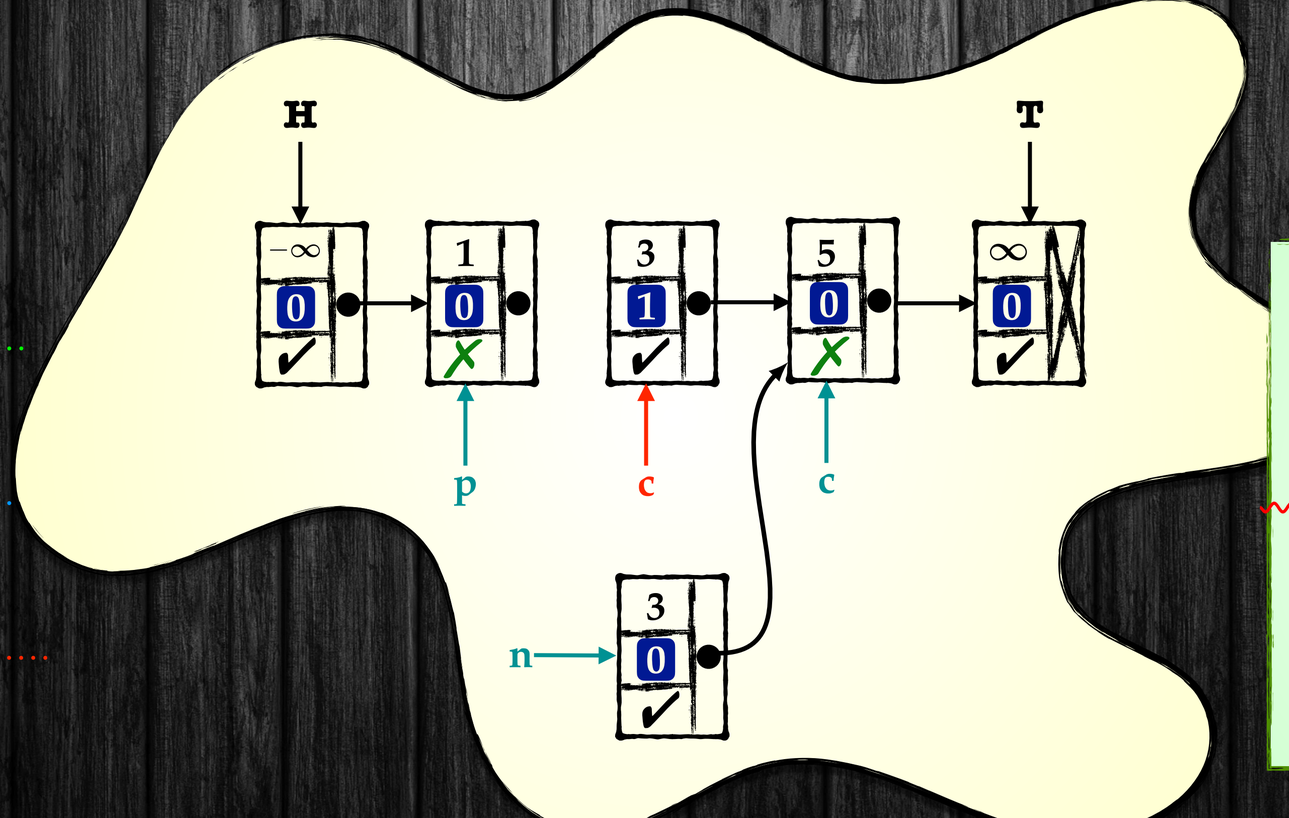
```
ctn(e):
local c, b
1  c := Head;
2  while (c.val < e)
3     c := c.next
4  b := c.mark
5  if (!b && c.val = e)
6     return true;
7  else return false;
```

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;
```

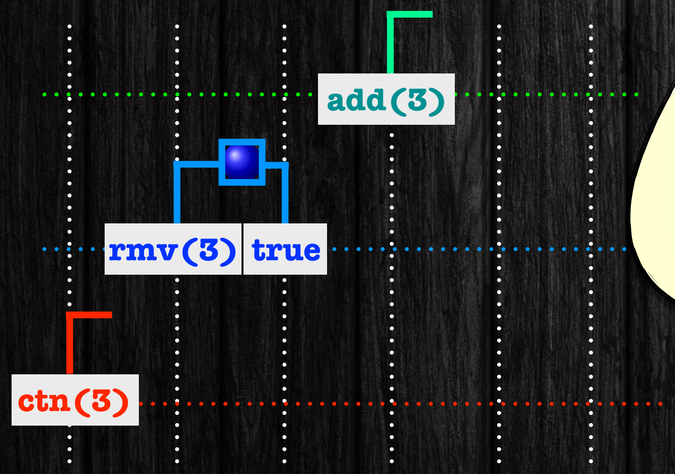
Sets | Lazy Synchronization

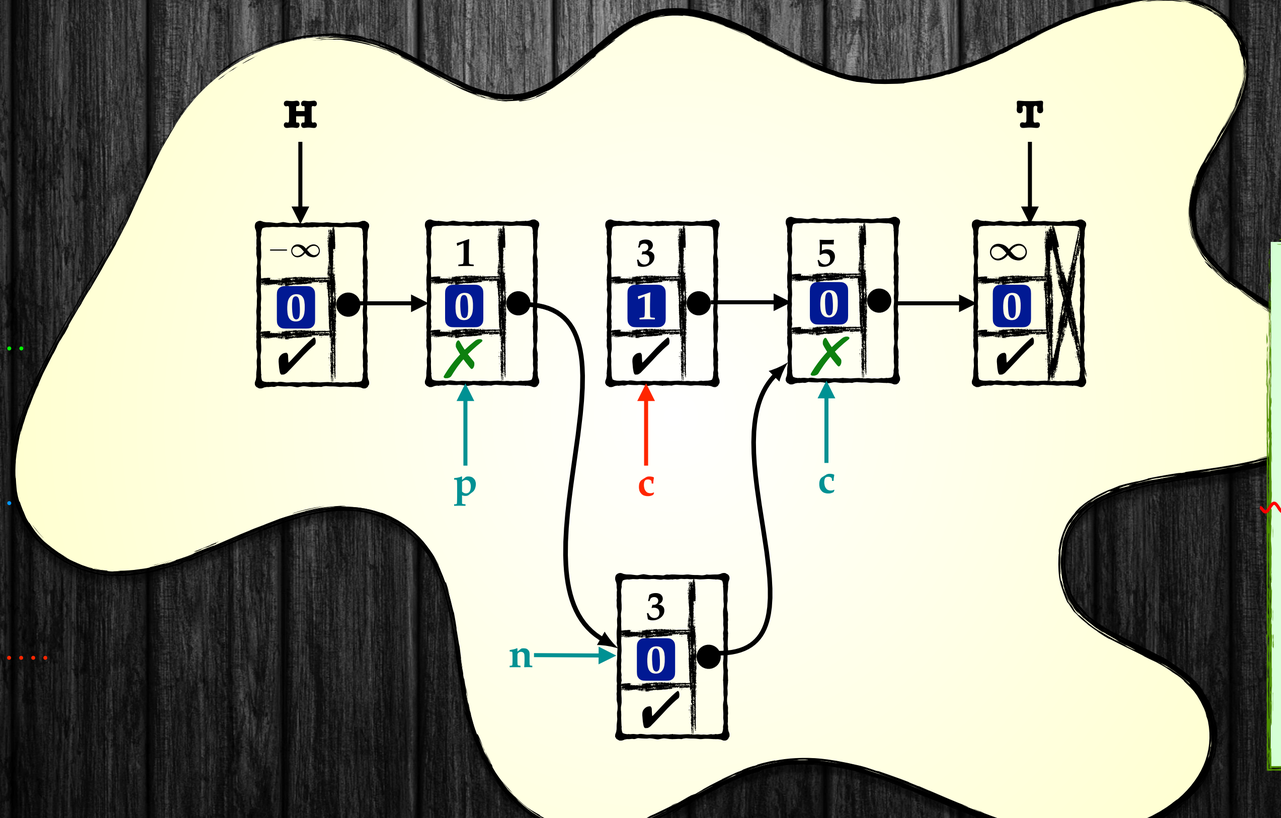
```
ctn(e):
local c, b
1  c := Head;
2  while (c.val < e)
3     c := c.next
4  b := c.mark
5  if (!b && c.val = e)
6     return true;
7  else return false;
```

ctn(3)

not executed yet







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;
```

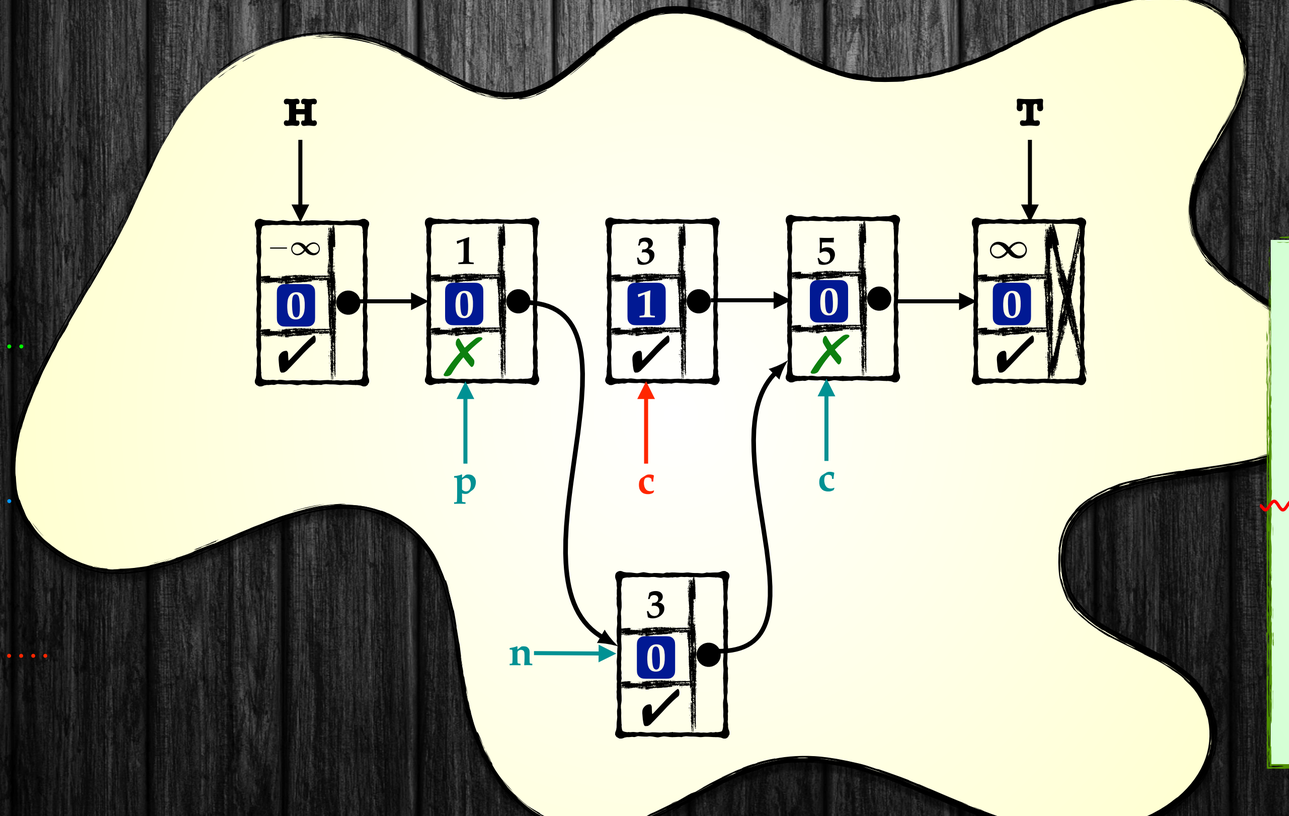
Sets | Lazy Synchronization

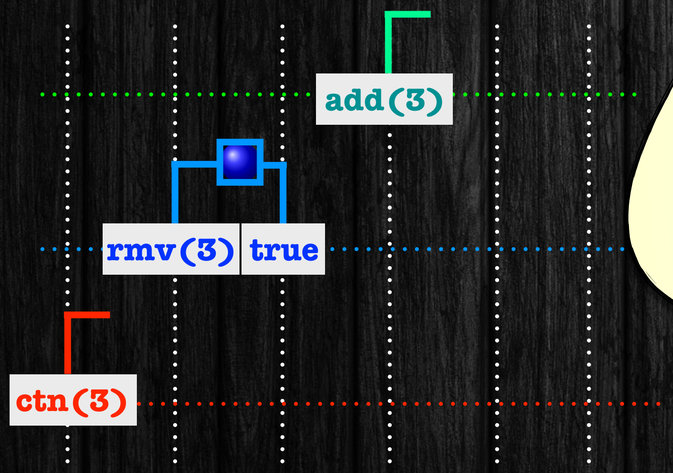
```
ctn(e):
local c, b
1  c := Head;
2  while (c.val < e)
3     c := c.next
4  b := c.mark
5  if (!b && c.val = e)
6     return true;
7  else return false;
```

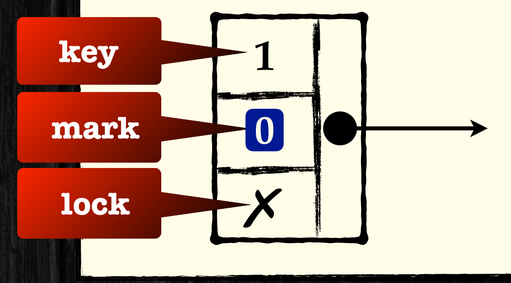
ctn(3)

not executed yet

linearization point







add(3)

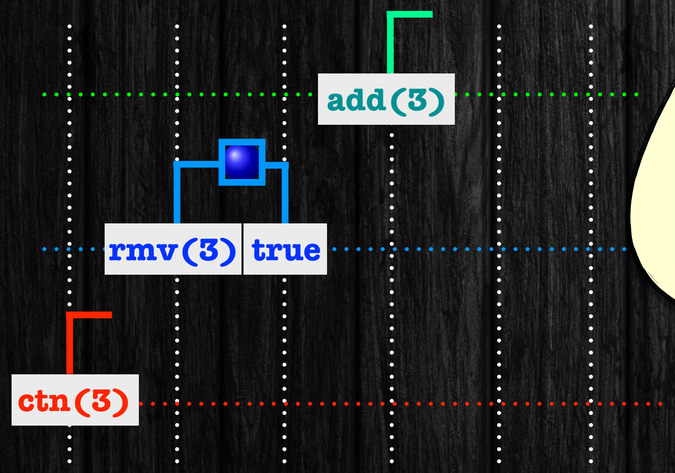
```
add(e):
   local p, c, n, r
   1  (p,c) := locate(e);
   2  if (c.val <> e)
   3     n := new Node(0,e,c,false);
   4     p.next := n;
   5     r := true;
   6  else r := false;
   7  unlock(p);
   8  unlock(c);
   9  return r;
```

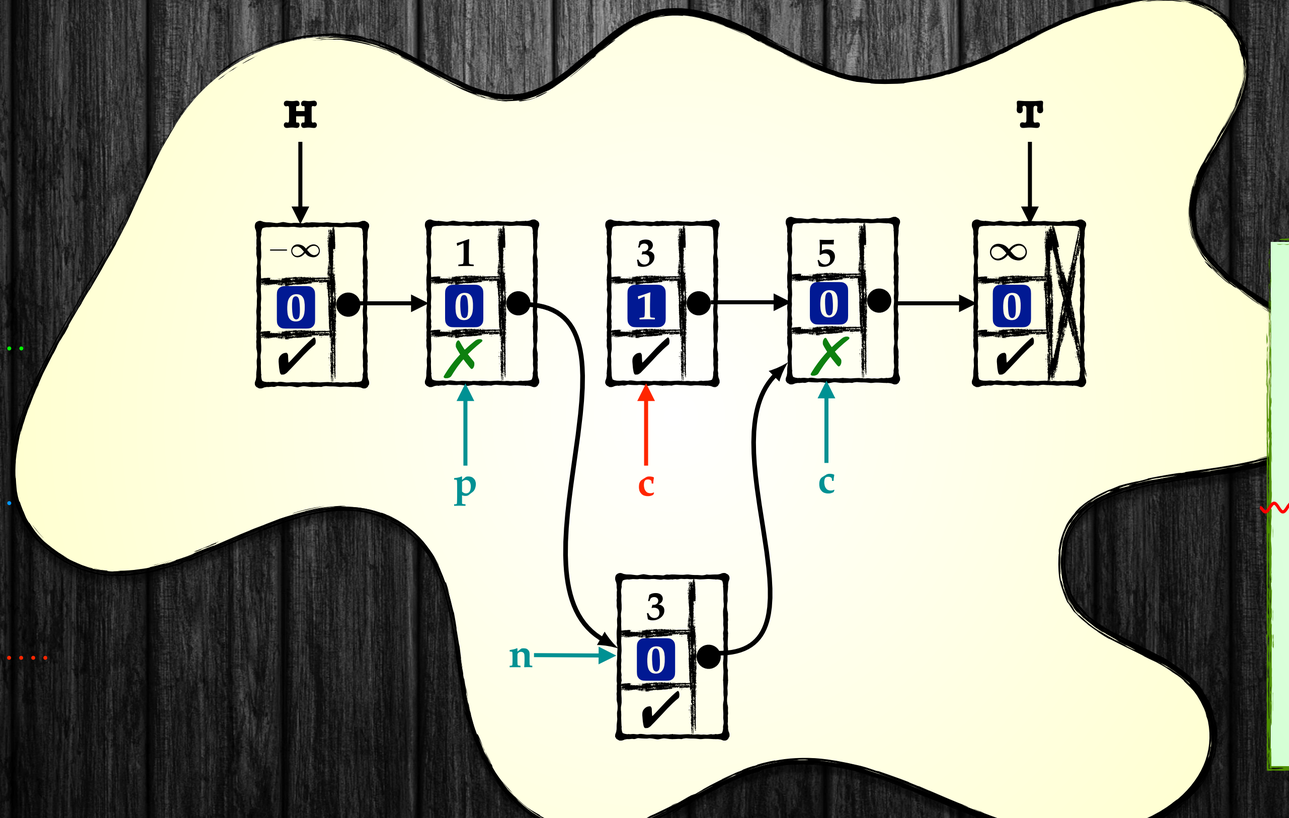
Sets | Lazy Synchronization

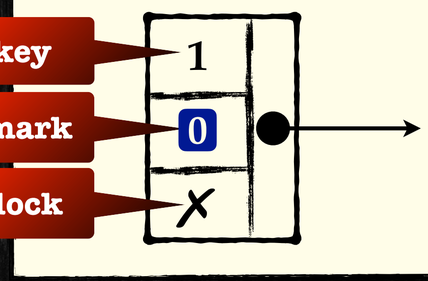
```
ctn(e):
local c, b
1  c := Head;
2  while (c.val < e)
3     c := c.next
4  b := c.mark
5  if (!b && c.val = e)
6     return true;
7  else return false;
```

ctn(3)

not executed yet







linearization point

successful add

add(3)

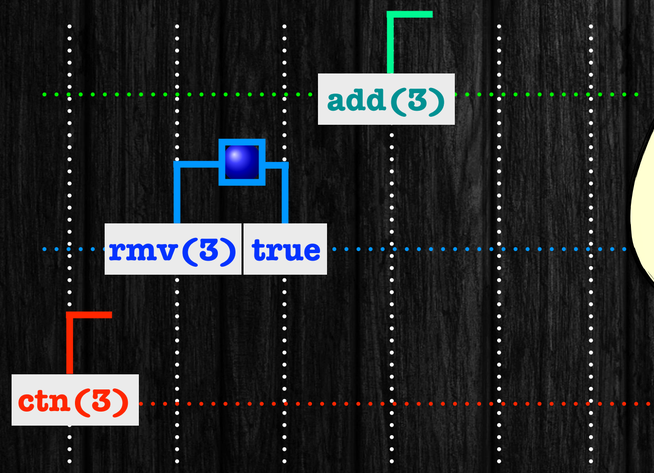
```
add(e):
   local p, c, n, r
   1  (p,c) := locate(e);
   2  if (c.val <> e)
   3     n := new Node(0,e,c,false);
   4     p.next := n;
   5     r := true;
   6  else r := false;
   7  unlock(p);
   8  unlock(c);
   9  return r;
```

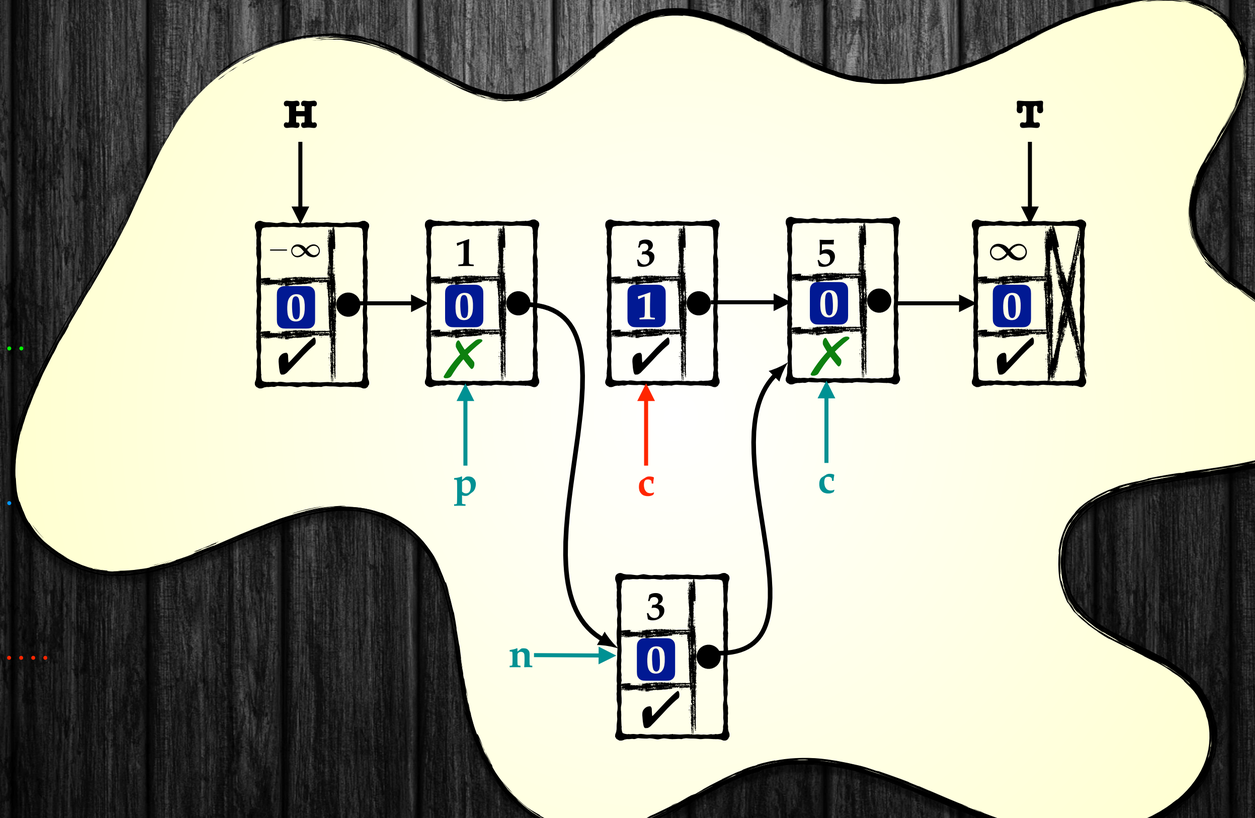
Sets | Lazy Synchronization

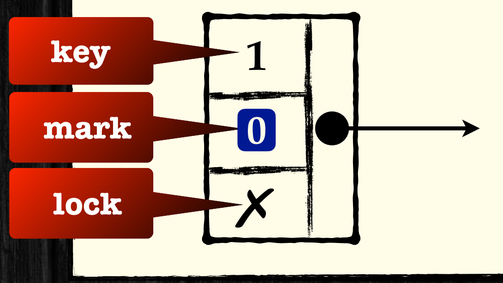
```
ctn(e):
local c, b
1  c := Head;
2  while (c.val < e)
3     c := c.next
4  b := c.mark
5  if (!b && c.val = e)
6     return true;
7  else return false;
```

ctn(3)

not executed yet







linearization point

successful add

"when p.next is set"

add(3)

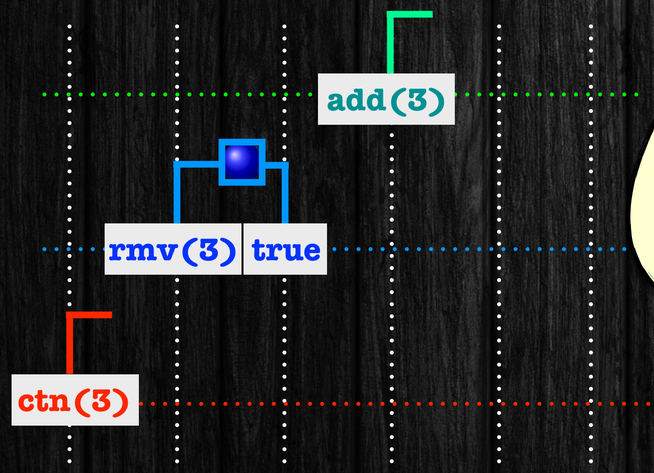
```
add(e):
   local p, c, n, r
   1  (p,c) := locate(e);
   2  if (c.val <> e)
   3     n := new Node(0,e,c,false);
   4     p.next := n;
   5     r := true;
   6  else r := false;
   7  unlock(p);
   8  unlock(c);
   9  return r;
```

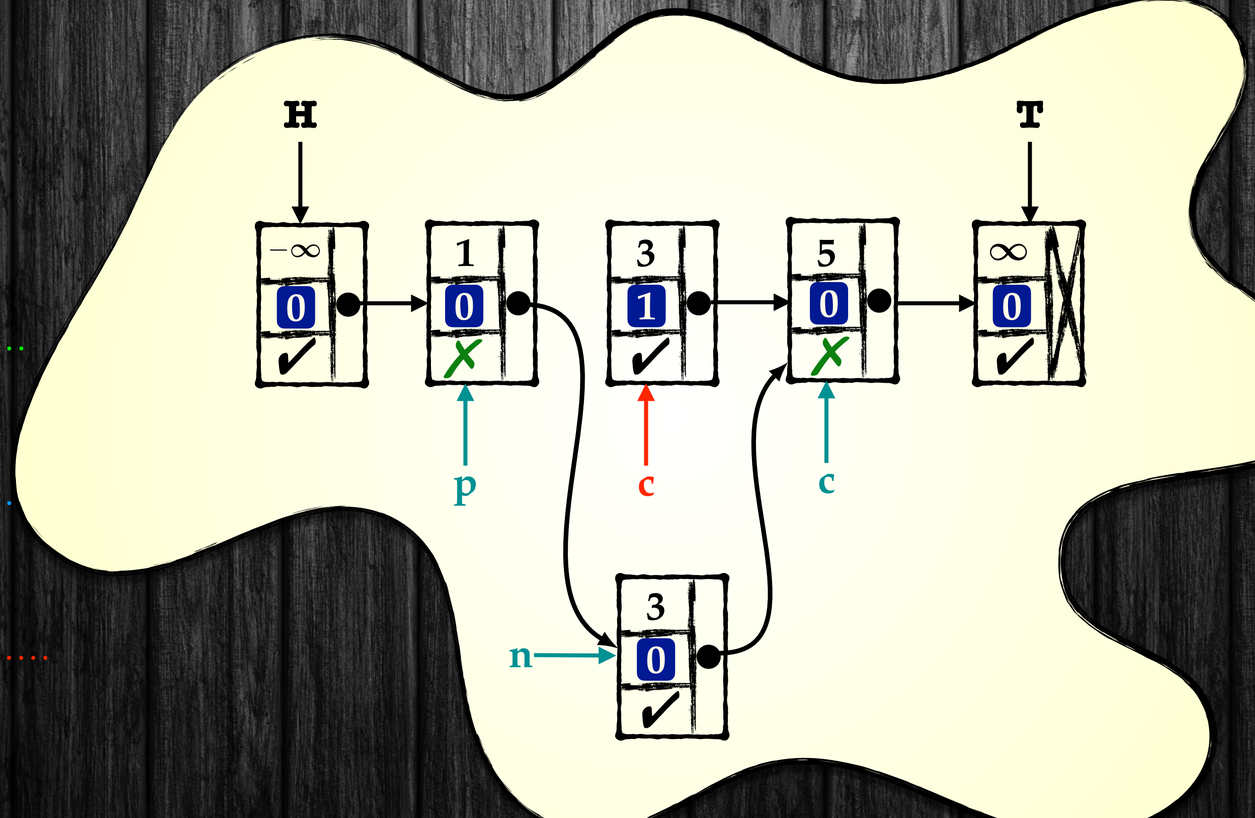
Sets | Lazy Synchronization

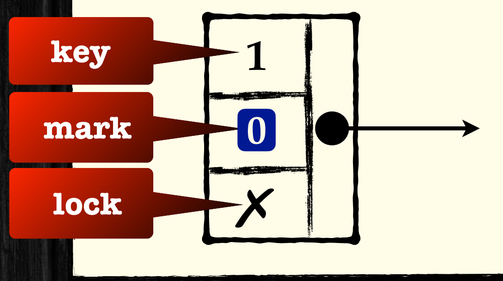
```
ctn(e):
local c, b
1  c := Head;
2  while (c.val < e)
3     c := c.next
4  b := c.mark
5  if (!b && c.val = e)
6     return true;
7  else return false;
```

ctn(3)

not executed yet







linearization point

successful add

"when p.next is set"

add(3)

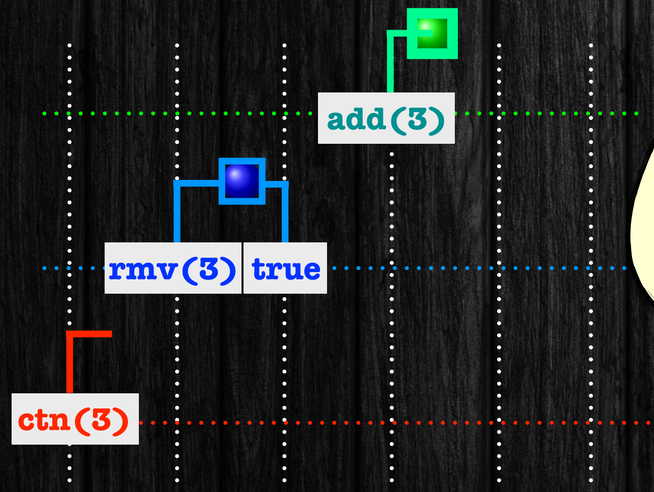
```
add(e):
   local p, c, n, r
   1  (p,c) := locate(e);
   2  if (c.val <> e)
   3     n := new Node(0,e,c,false);
   4     p.next := n;
   5     r := true;
   6  else r := false;
   7  unlock(p);
   8  unlock(c);
   9  return r;
```

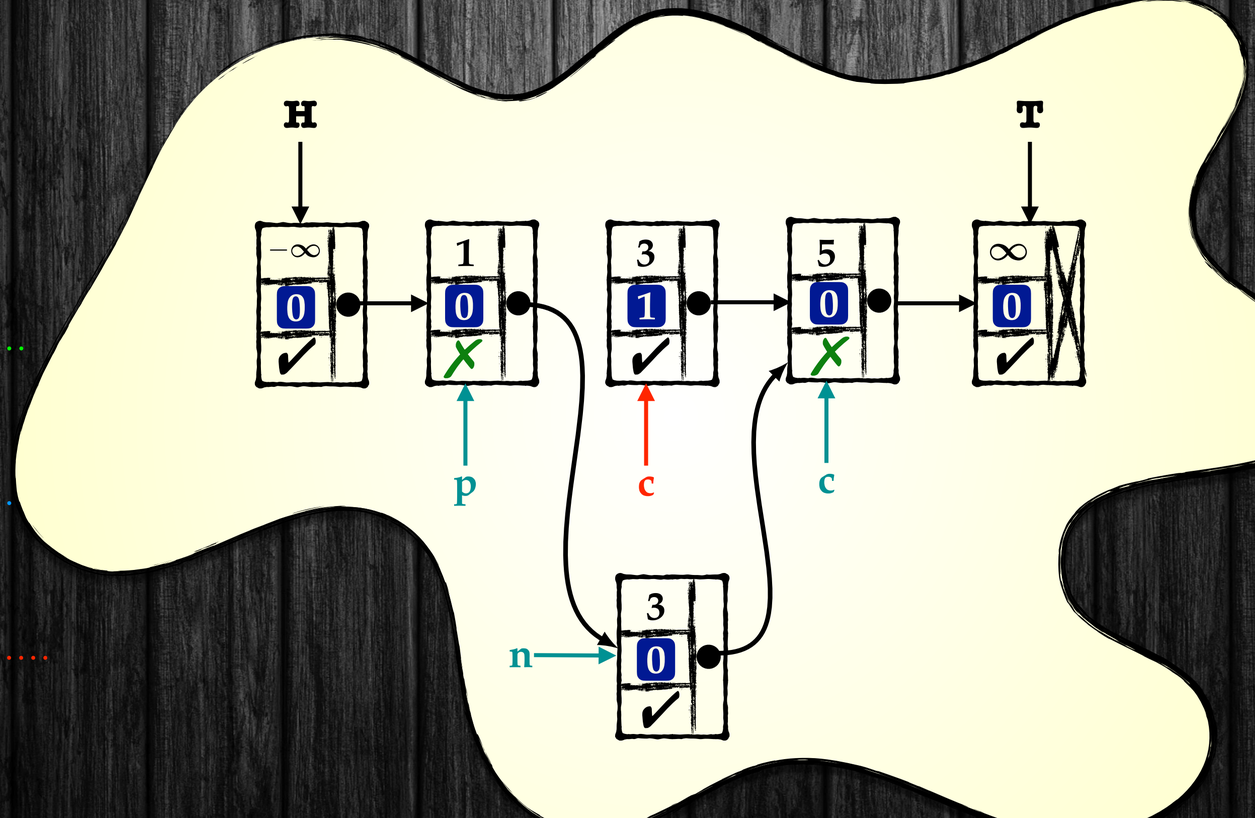
Sets | Lazy Synchronization

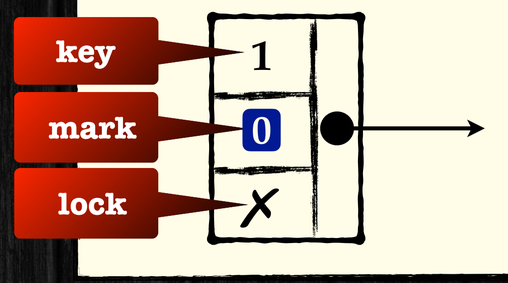
```
ctn(e):
local c, b
1  c := Head;
2  while (c.val < e)
3     c := c.next
4  b := c.mark
5  if (!b && c.val = e)
6     return true;
7  else return false;
```

ctn(3)

not executed yet







linearization point

successful add

"when p.next is set"

add(3)

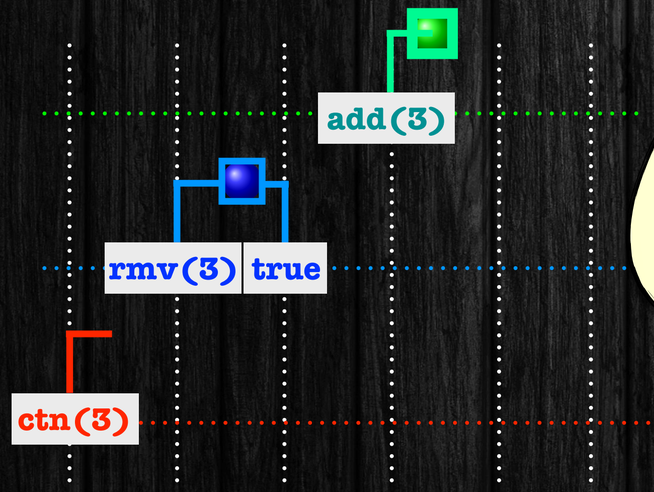
```
add(e):
   local p, c, n, r
   1  (p,c) := locate(e);
   2  if (c.val <> e)
   3     n := new Node(0,e,c,false);
   4     p.next := n;
   5     r := true;
   6  else r := false;
   7  unlock(p);
   8  unlock(c);
   9  return r;
```

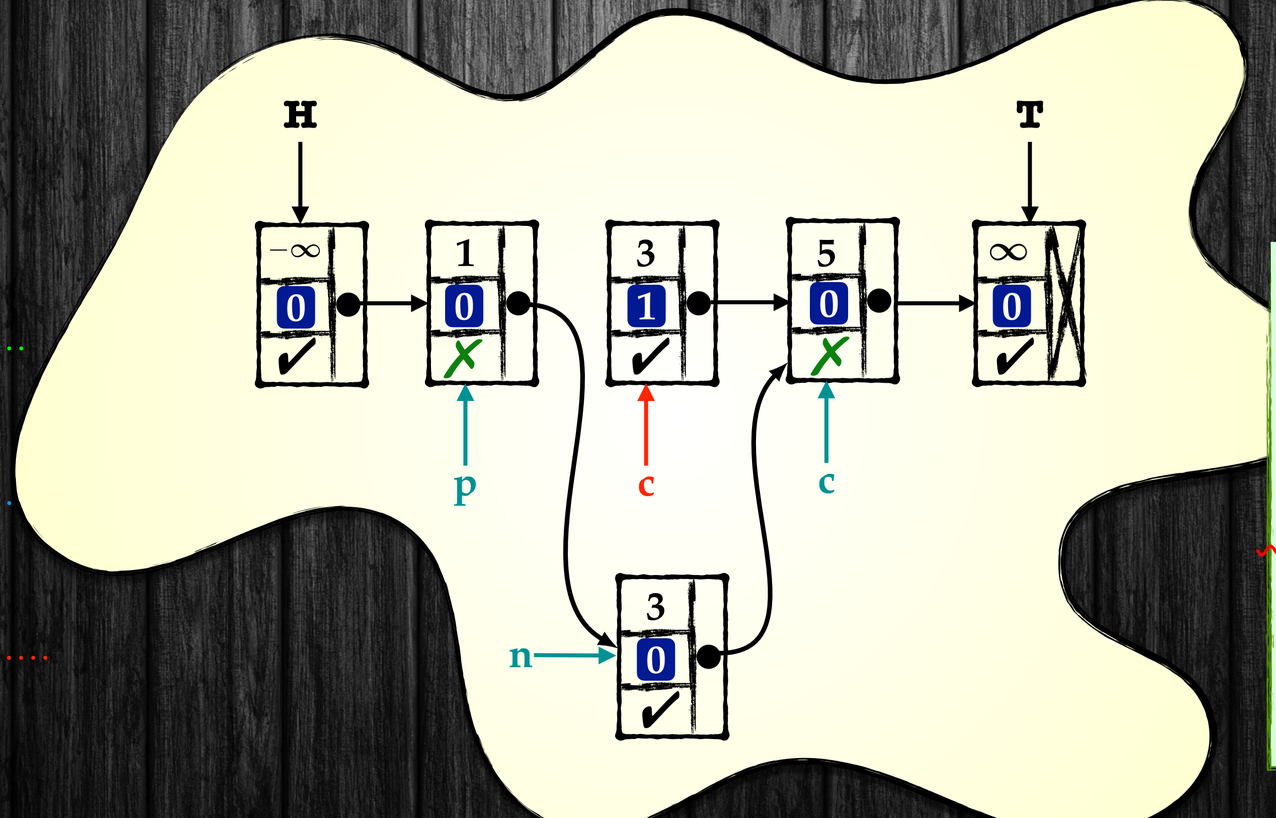
Sets | Lazy Synchronization

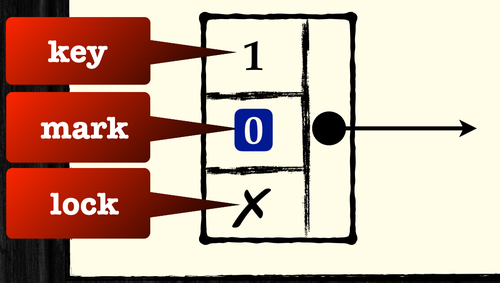
```
ctn(e):
local c, b
1  c := Head;
2  while (c.val < e)
3     c := c.next
4  b := c.mark
5  if (!b && c.val = e)
6     return true;
7  else return false;
```

ctn(3)

not executed yet







add(3)

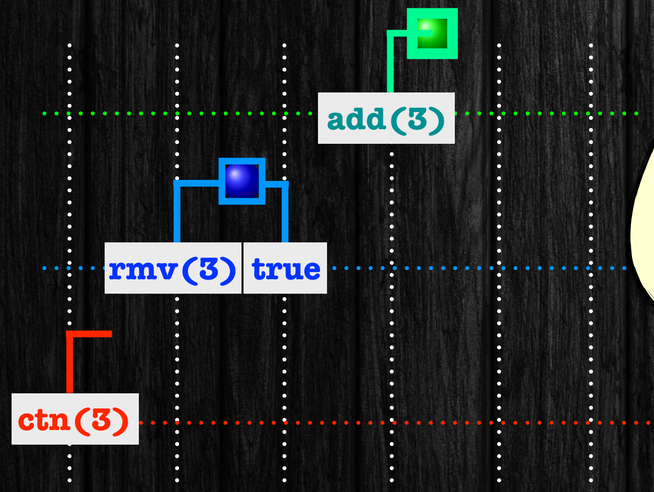
```
add(e):
   local p, c, n, r
   1  (p,c) := locate(e);
   2  if (c.val <> e)
   3     n := new Node(0,e,c,false);
   4     p.next := n;
   5     r := true;
   6  else r := false;
   7  unlock(p);
   8  unlock(c);
   9  return r;
```

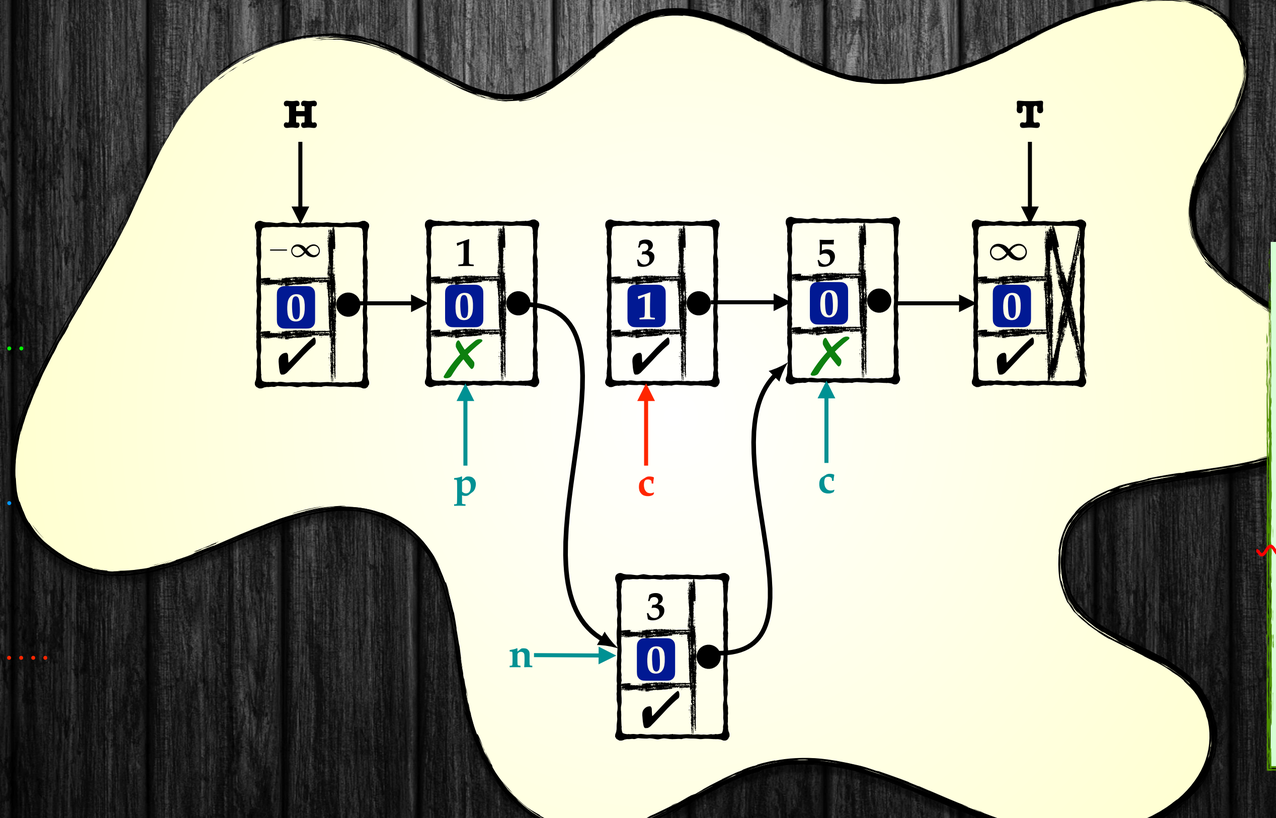
Sets | Lazy Synchronization

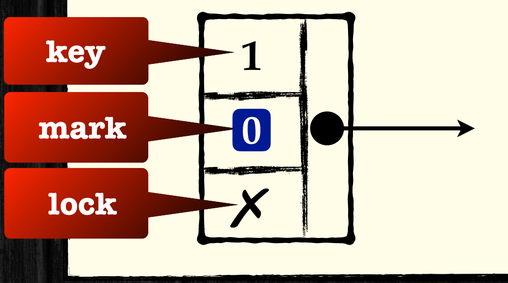
```
ctn(e):
local c, b
1  c := Head;
2  while (c.val < e)
3     c := c.next
4  b := c.mark
5  if (!b && c.val = e)
6     return true;
7  else return false;
```

ctn(3)

not executed yet







add(3)

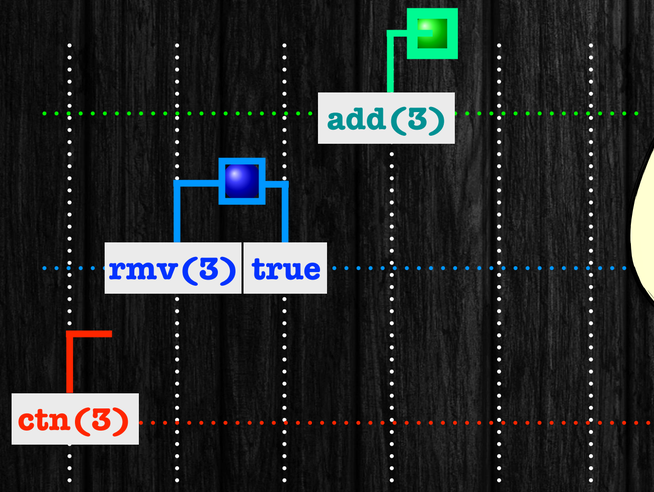
```
add(e):
   local p, c, n, r
1  (p,c) := locate(e);
2  if (c.val <> e)
3     n := new Node(0,e,c,false);
4     p.next := n;
5     r := true;
6  else r := false;
7  unlock(p);
8  unlock(c);
9  return r;
```

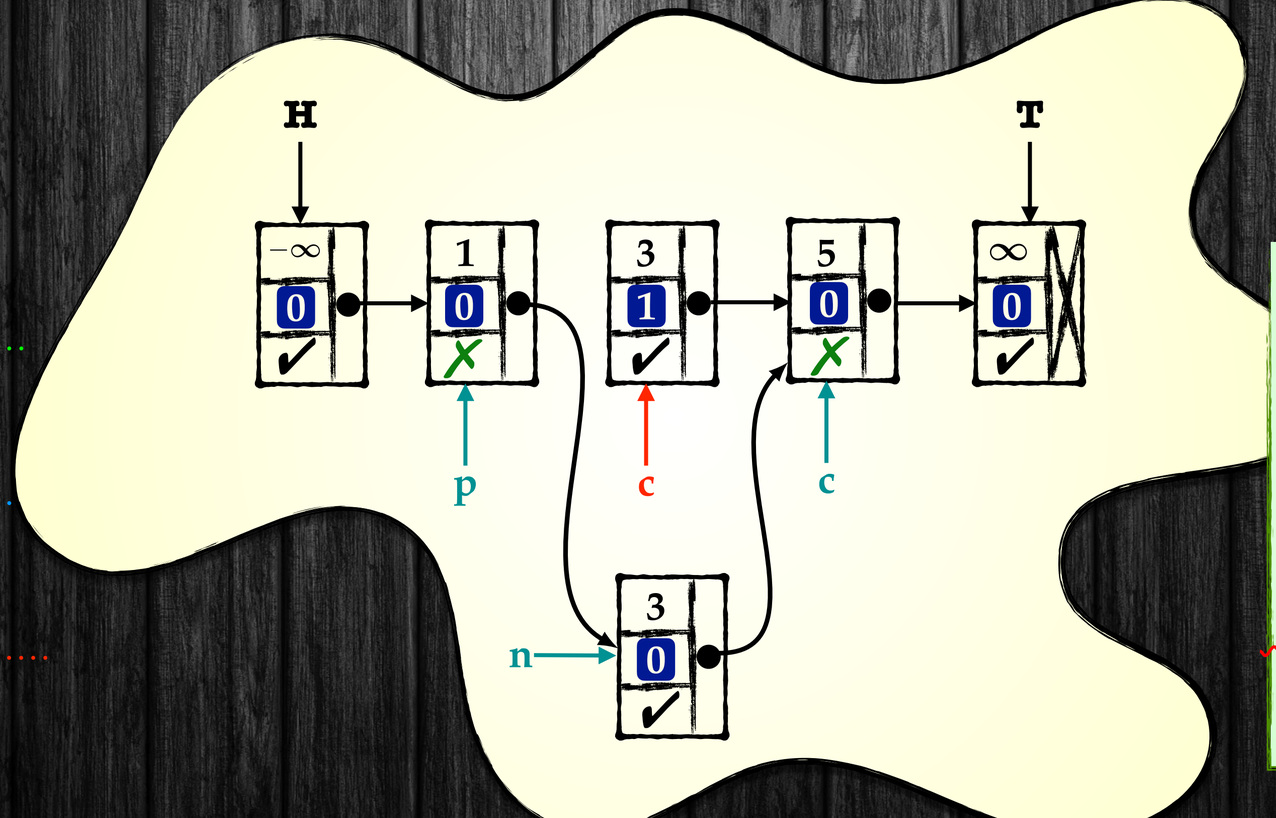
Sets | Lazy Synchronization

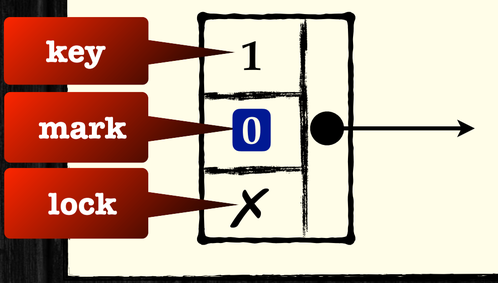
```
ctn(e):
local c, b
1  c := Head;
2  while (c.val < e)
3     c := c.next
4  b := c.mark
5  if (!b && c.val = e)
6     return true;
7  else return false;
```

ctn(3)

not executed yet







add(3)

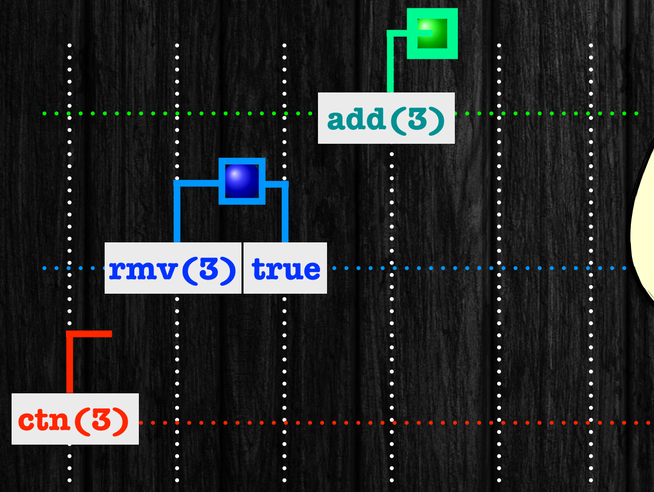
```
add(e):
   local p, c, n, r
   1  (p,c) := locate(e);
   2  if (c.val <> e)
   3     n := new Node(0,e,c,false);
   4     p.next := n;
   5     r := true;
   6  else r := false;
   7  unlock(p);
   8  unlock(c);
   9  return r;
```

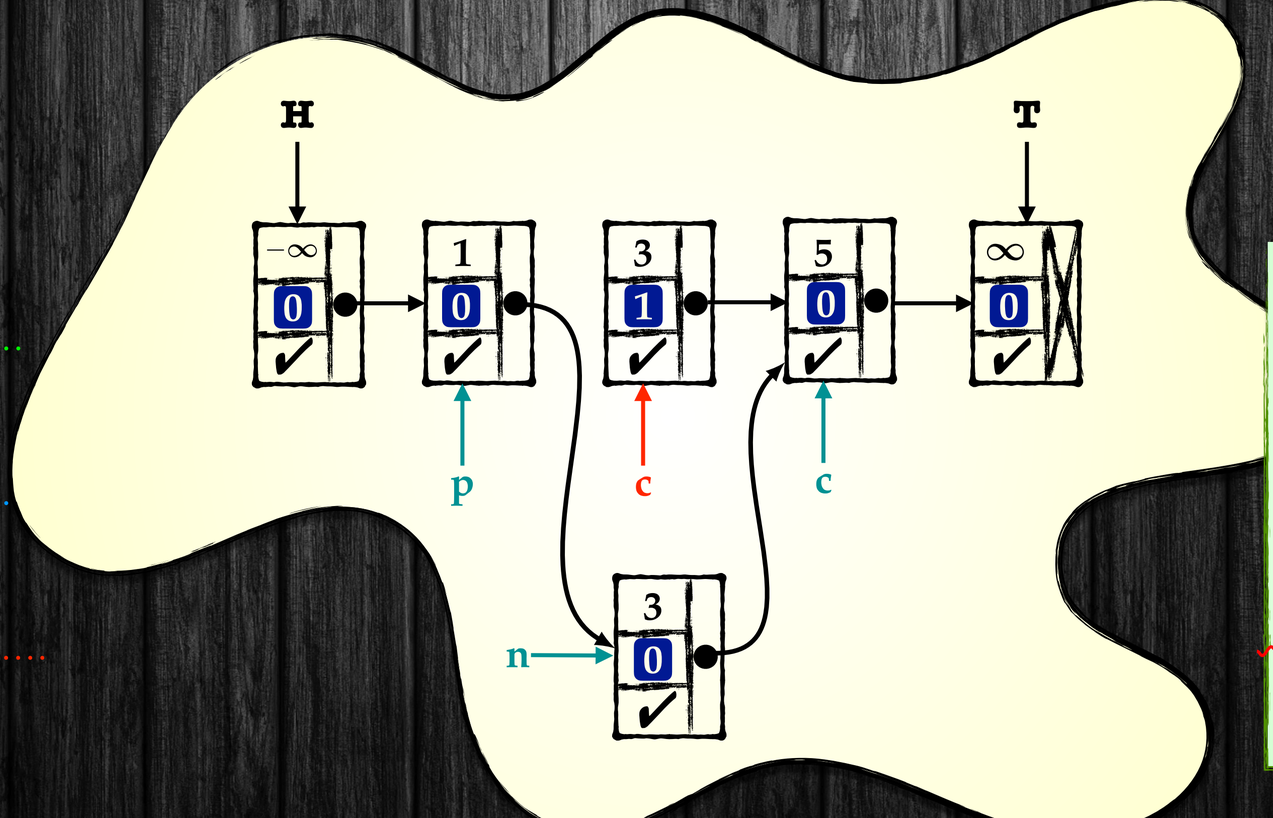
Sets | Lazy Synchronization

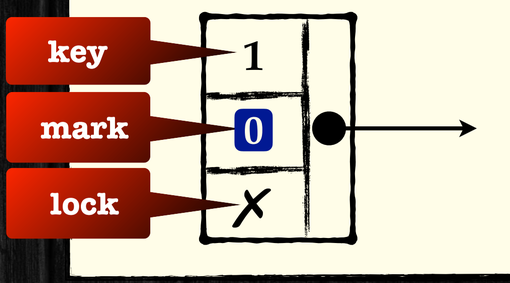
```
ctn(e):
local c, b
1  c := Head;
2  while (c.val < e)
3     c := c.next
4  b := c.mark
5  if (!b && c.val = e)
6     return true;
7  else return false;
```

ctn(3)

not executed yet







add(3)

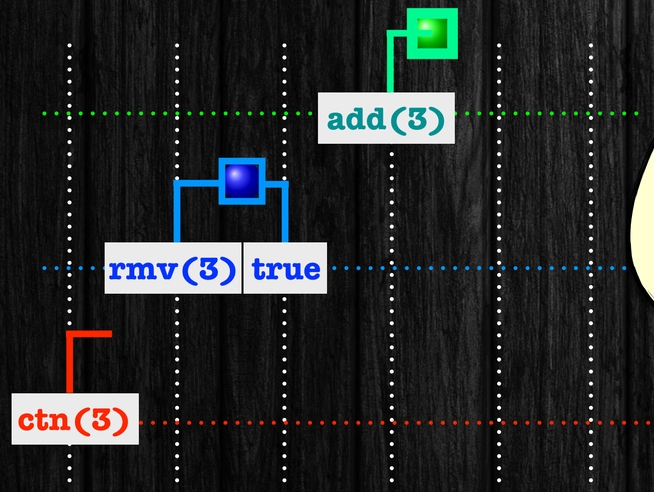
```
add(e):
   local p, c, n, r
1  (p,c) := locate(e);
2  if (c.val <> e)
3     n := new Node(0,e,c,false);
4     p.next := n;
5     r := true;
6  else r := false;
7  unlock(p);
8  unlock(c);
9  return r;
```

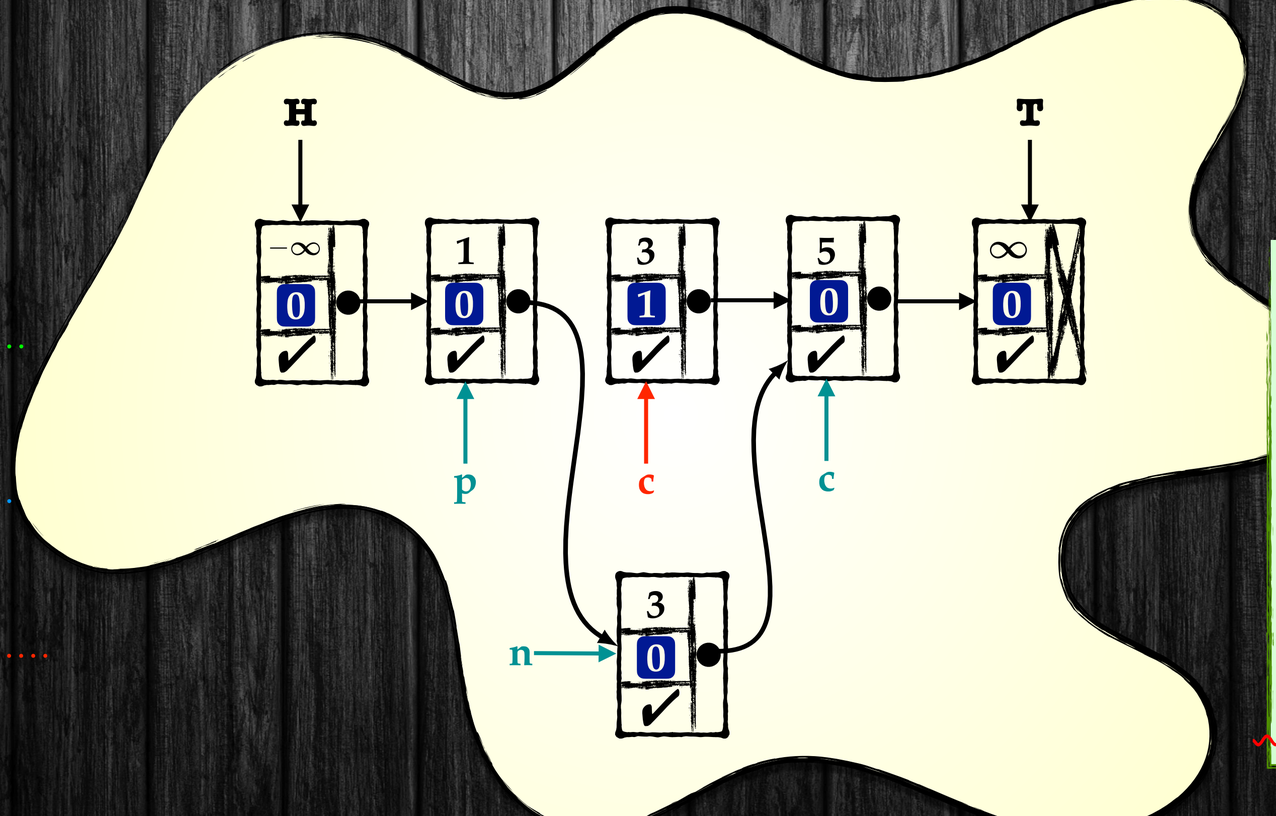
Sets | Lazy Synchronization

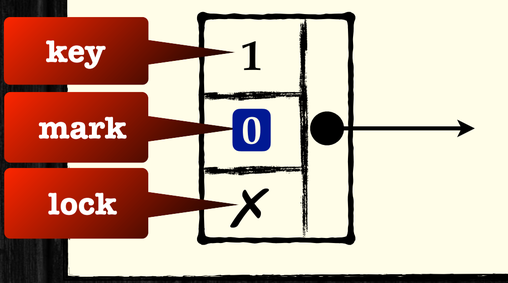
```
ctn(e):
local c, b
1  c := Head;
2  while (c.val < e)
3     c := c.next
4  b := c.mark
5  if (!b && c.val = e)
6     return true;
7  else return false;
```

ctn(3)

not executed yet







add(3)

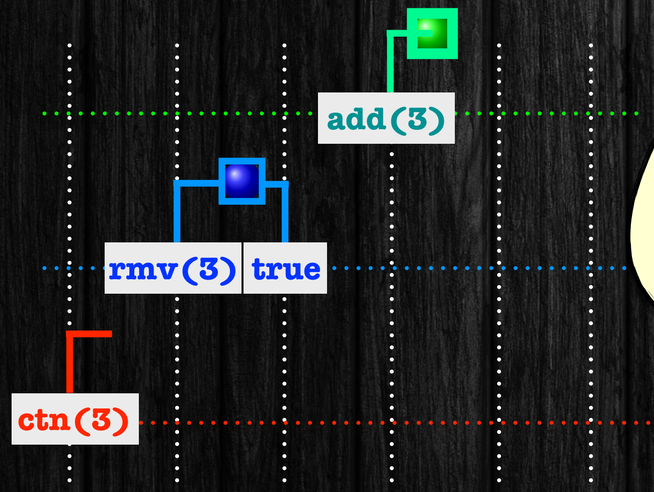
```
add(e):
   local p, c, n, r
1  (p,c) := locate(e);
2  if (c.val <> e)
3     n := new Node(0,e,c,false);
4     p.next := n;
5     r := true;
6  else r := false;
7  unlock(p);
8  unlock(c);
9  return r;
```

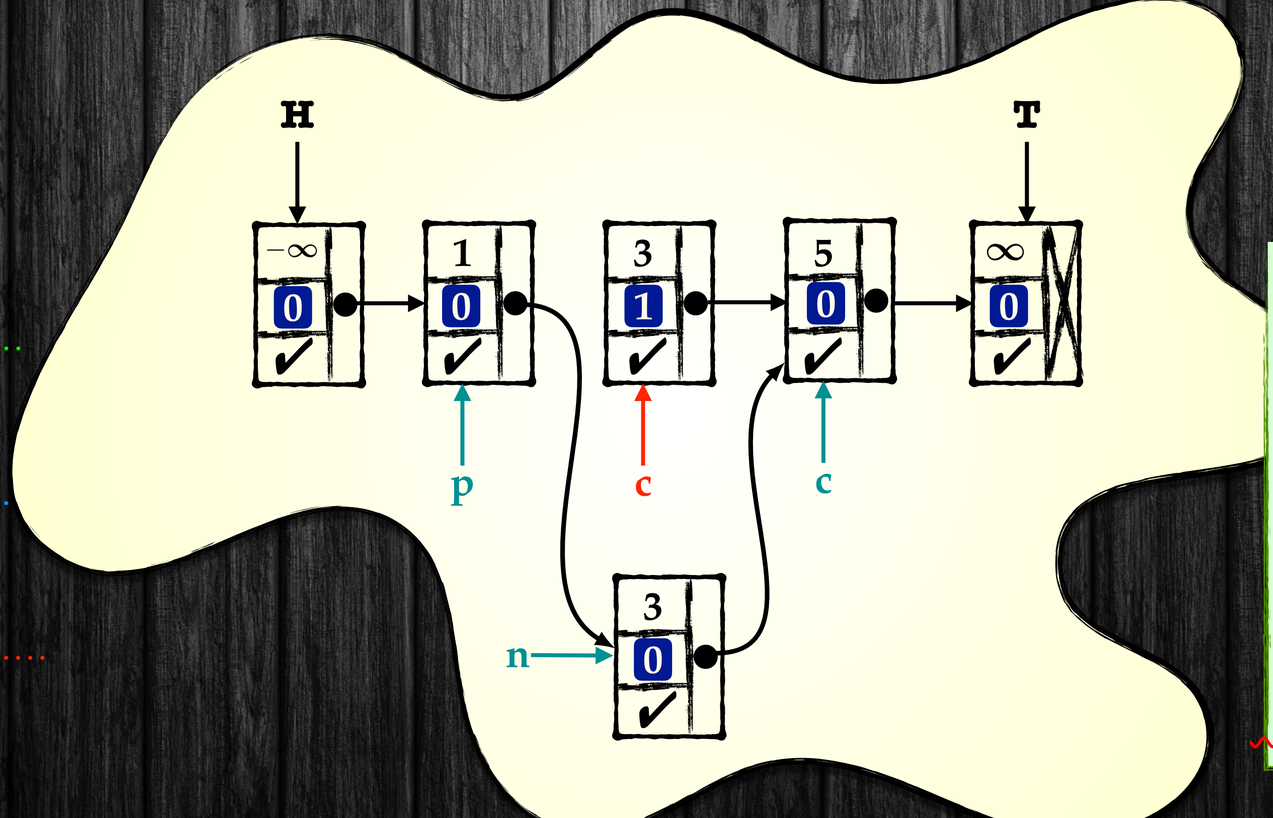
Sets | Lazy Synchronization

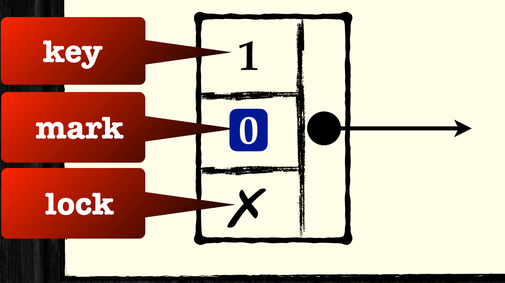
```
ctn(e):
local c, b
1  c := Head;
2  while (c.val < e)
3     c := c.next
4  b := c.mark
5  if (!b && c.val = e)
6     return true;
7  else return false;
```

ctn(3)

not executed yet







add(3)

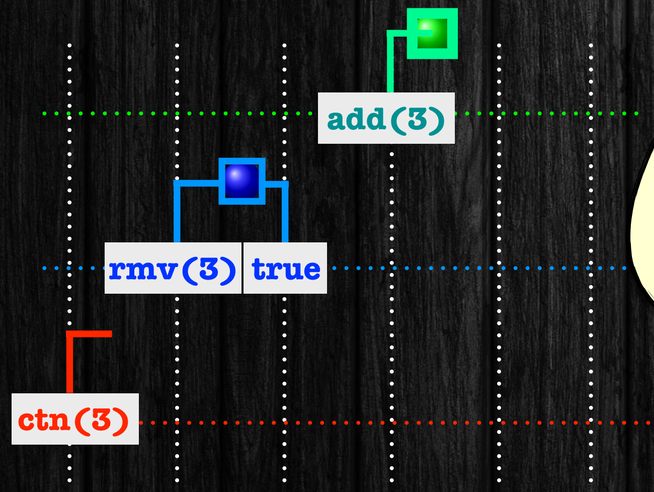
```
add(e):
   local p, c, n, r
1  (p,c) := locate(e);
2  if (c.val <> e)
3     n := new Node(0,e,c,false);
4     p.next := n;
5     r := true;
6  else r := false;
7  unlock(p);
8  unlock(c);
9  return r;
```

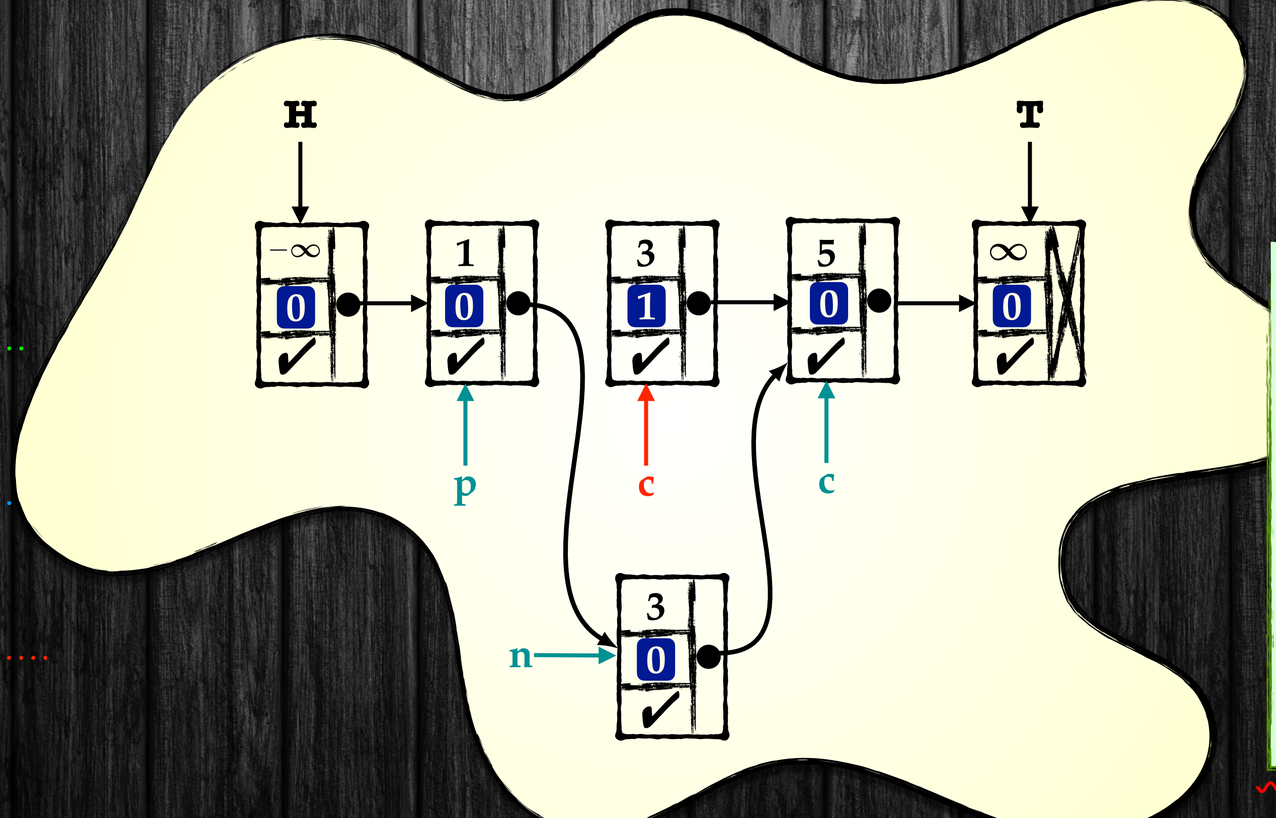
Sets | Lazy Synchronization

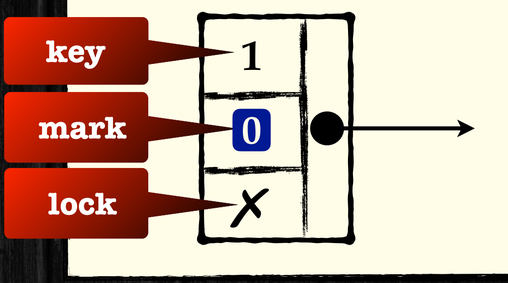
```
ctn(e):
local c, b
1  c := Head;
2  while (c.val < e)
3     c := c.next
4  b := c.mark
5  if (!b && c.val = e)
6     return true;
7  else return false;
```

ctn(3)

not executed yet







add(3)

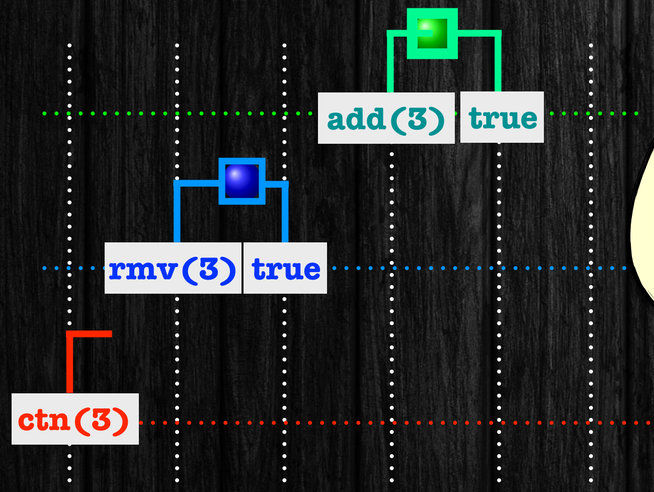
```
add(e):
   local p, c, n, r
1  (p,c) := locate(e);
2  if (c.val <> e)
3     n := new Node(0,e,c,false);
4     p.next := n;
5     r := true;
6  else r := false;
7  unlock(p);
8  unlock(c);
9  return r;
```

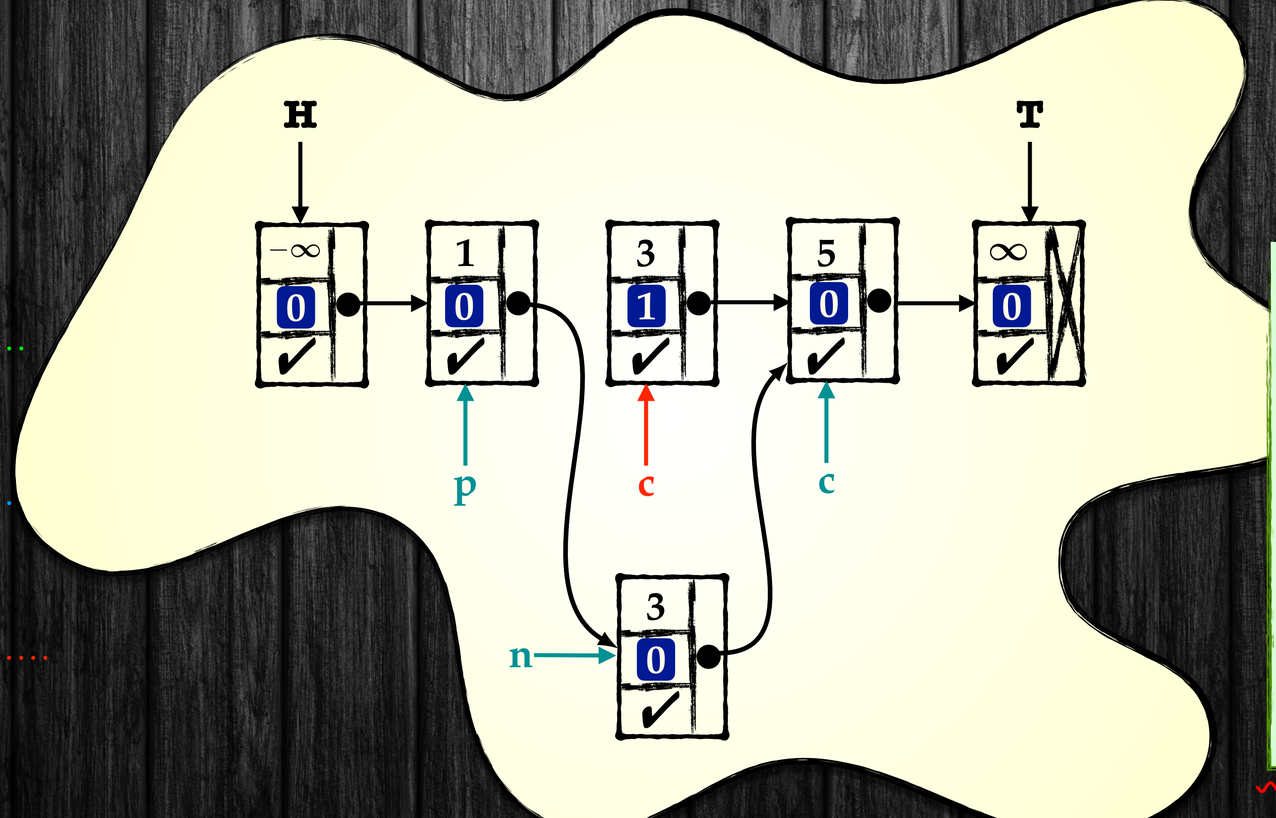
Sets | Lazy Synchronization

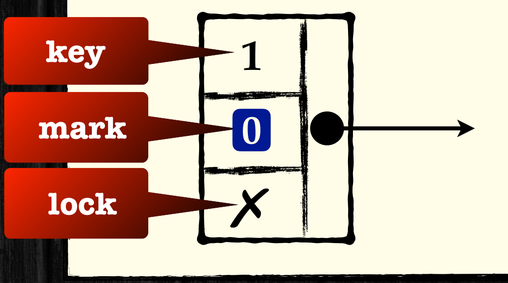
```
ctn(e):
local c, b
1  c := Head;
2  while (c.val < e)
3     c := c.next
4  b := c.mark
5  if (!b && c.val = e)
6     return true;
7  else return false;
```

ctn(3)

not executed yet







add(3)

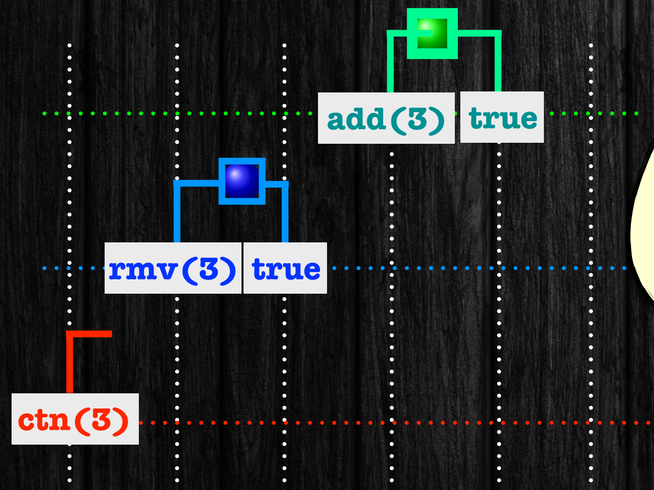
```
add(e):
   local p, c, n, r
   1  (p,c) := locate(e);
   2  if (c.val <> e)
   3     n := new Node(0,e,c,false);
   4     p.next := n;
   5     r := true;
   6  else r := false;
   7  unlock(p);
   8  unlock(c);
   9  return r;
```

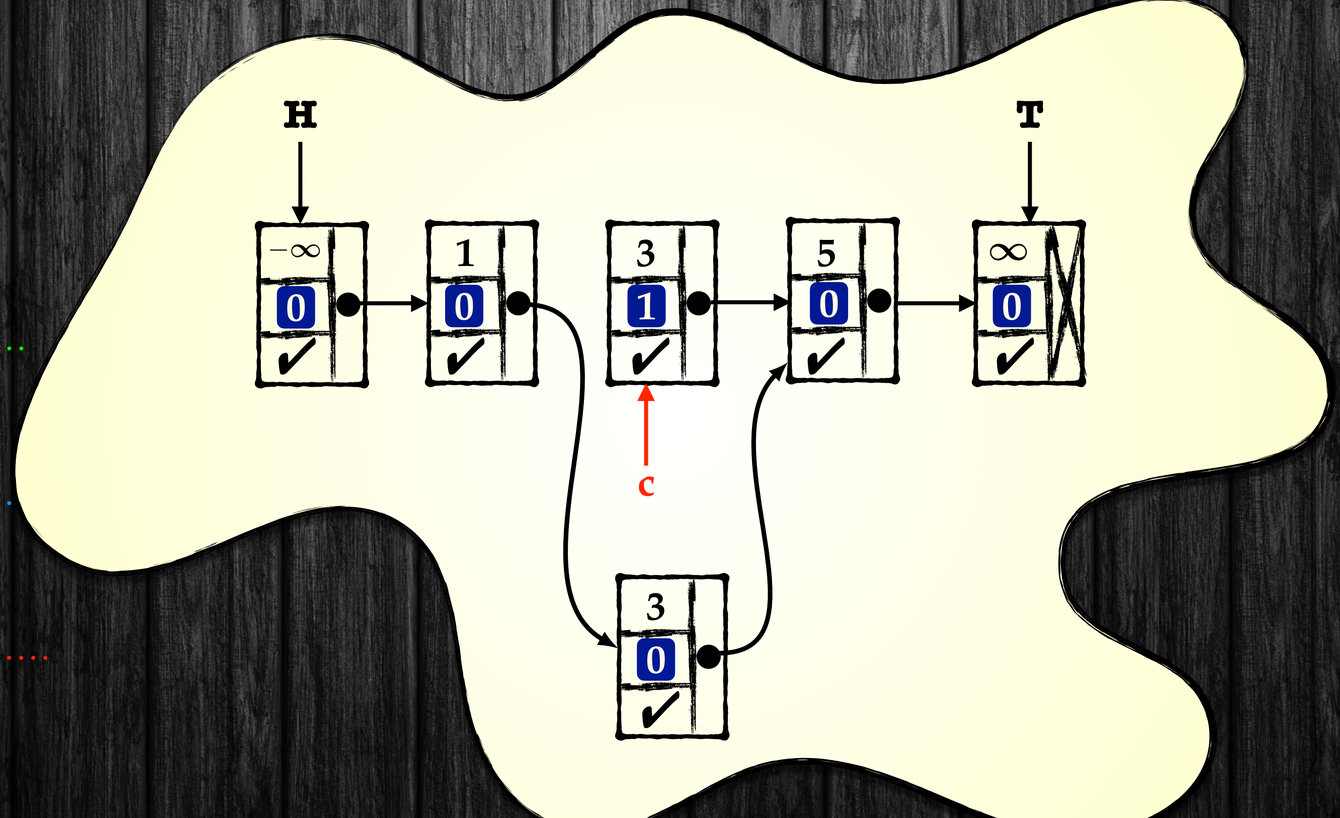
Sets | Lazy Synchronization

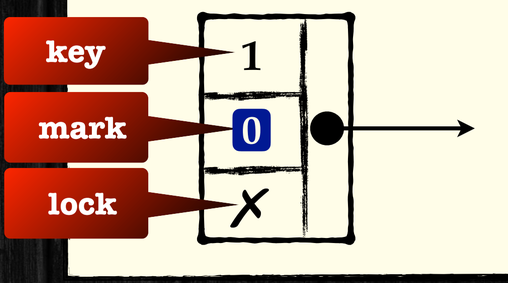
```
ctn(e):
local c, b
1  c := Head;
2  while (c.val < e)
3     c := c.next
4  b := c.mark
5  if (!b && c.val = e)
6     return true;
7  else return false;
```

ctn(3)

not executed yet







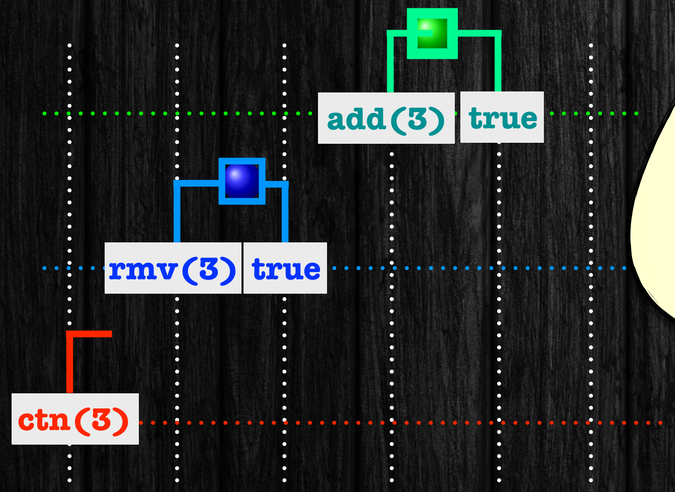
Sets | Lazy Synchronization

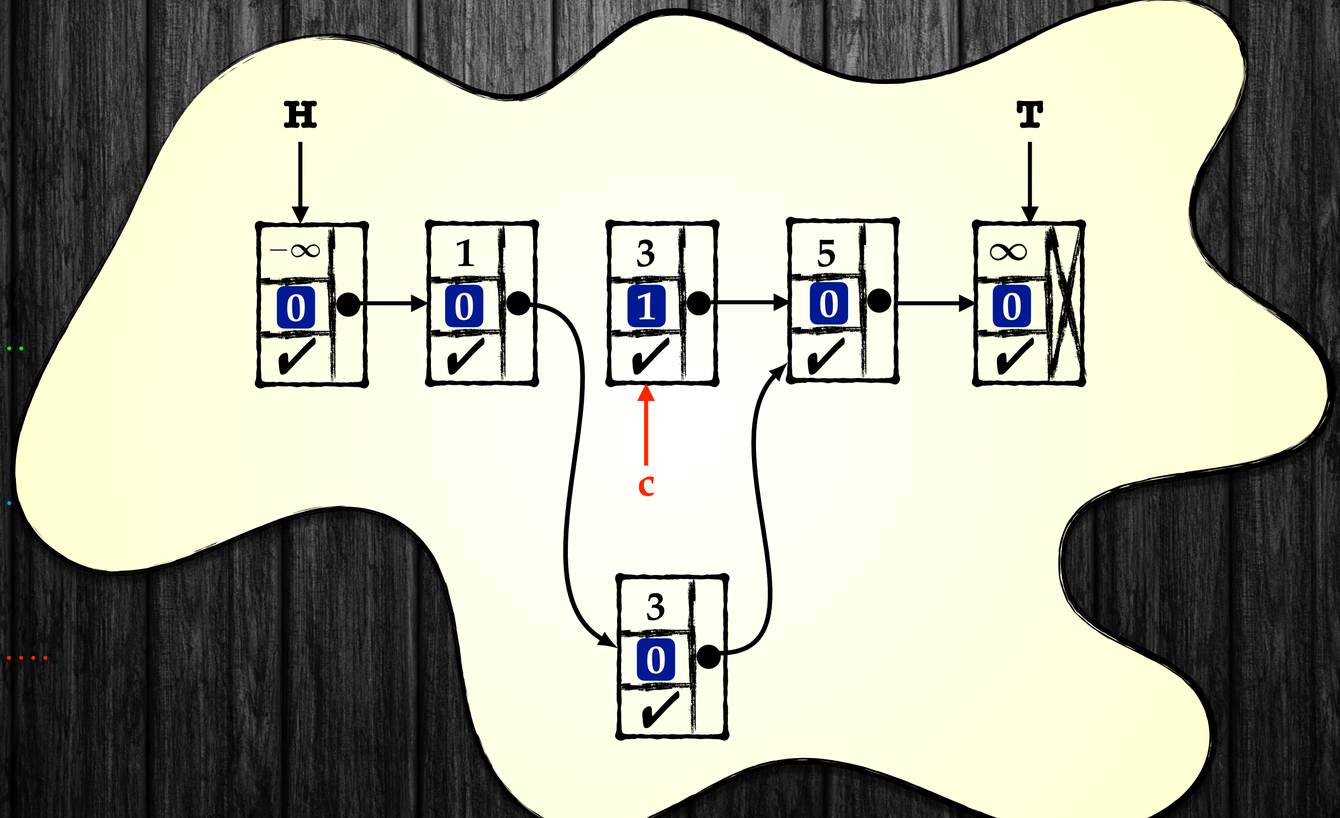
```
ctn(e):
local c, b
1  c := Head;
2  while (c.val < e)
3     c := c.next
4  b := c.mark
5  if (!b && c.val = e)
6     return true;
7  else return false;
```

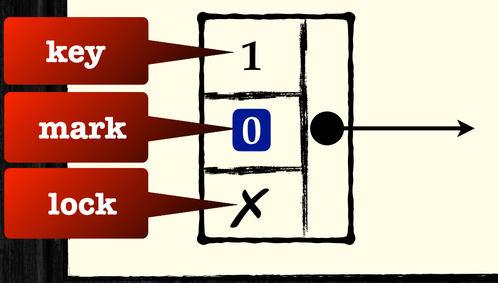
ctn(3)

not executed yet

becomes active again



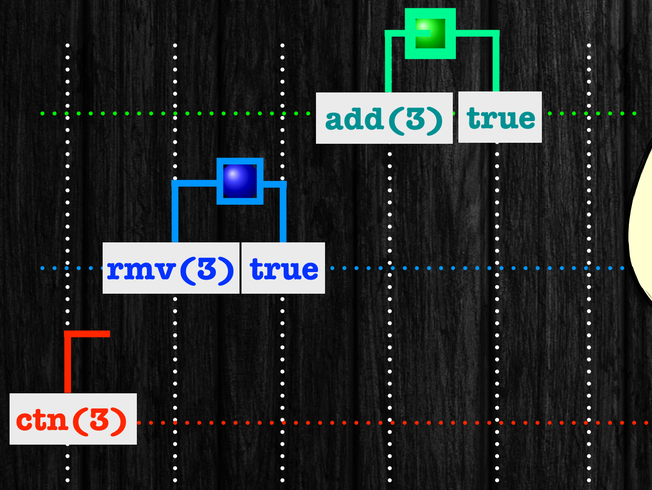


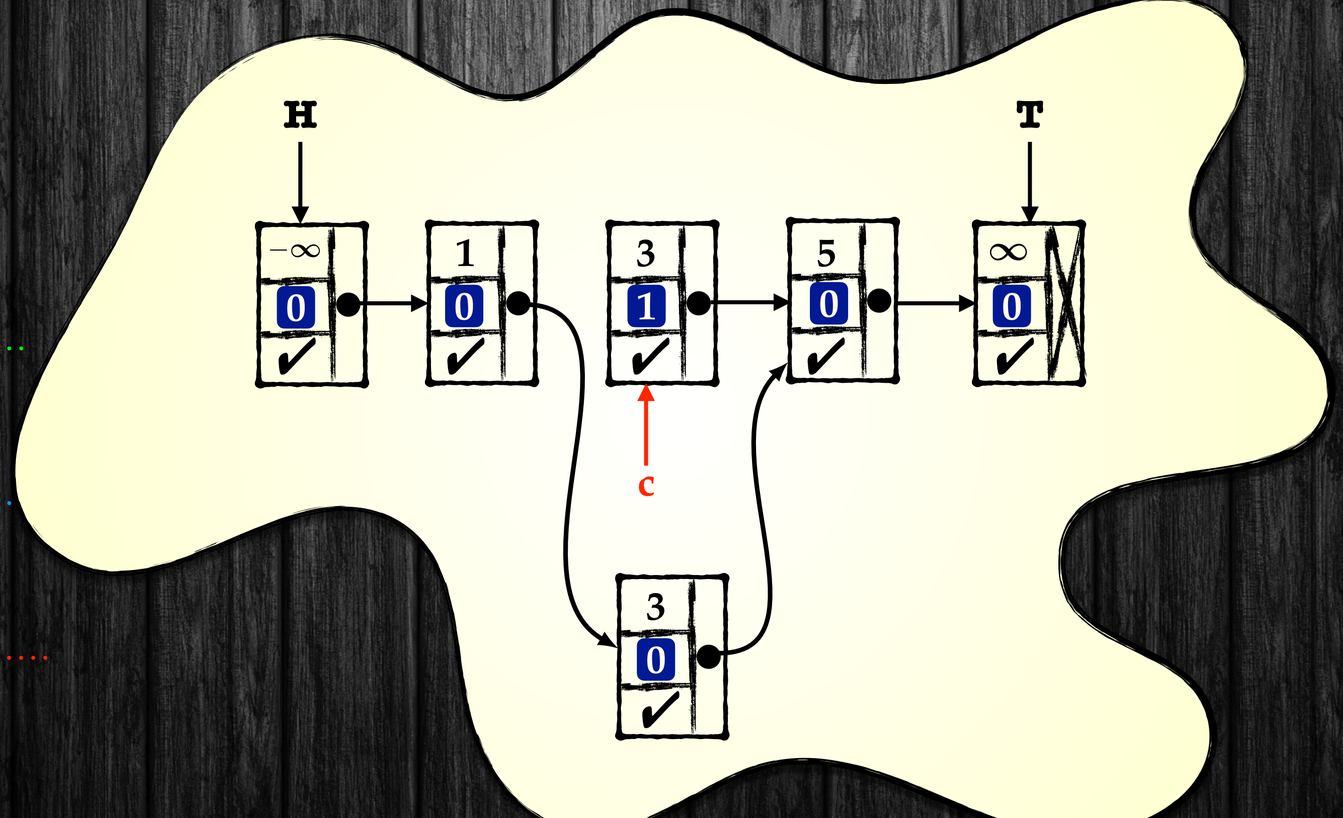


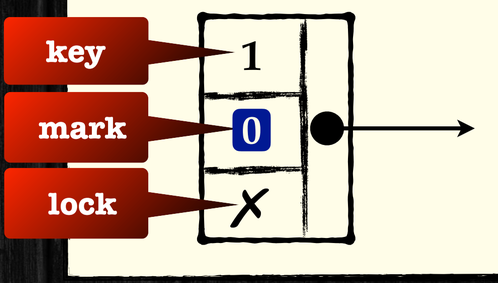
Sets | Lazy Synchronization

```
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)



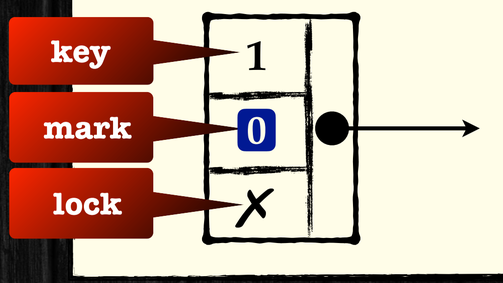


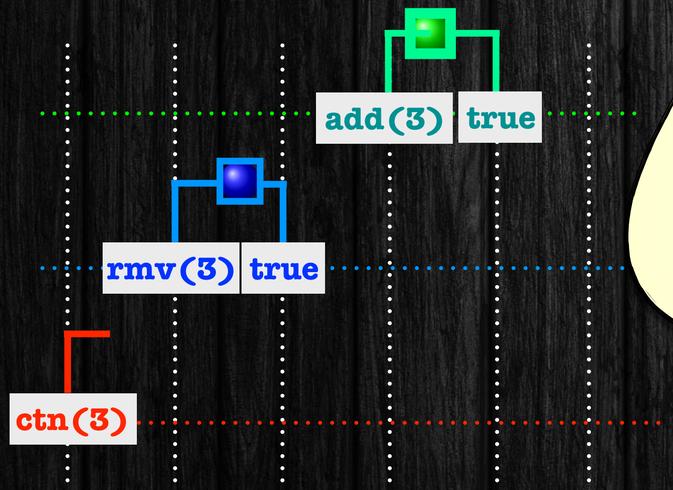


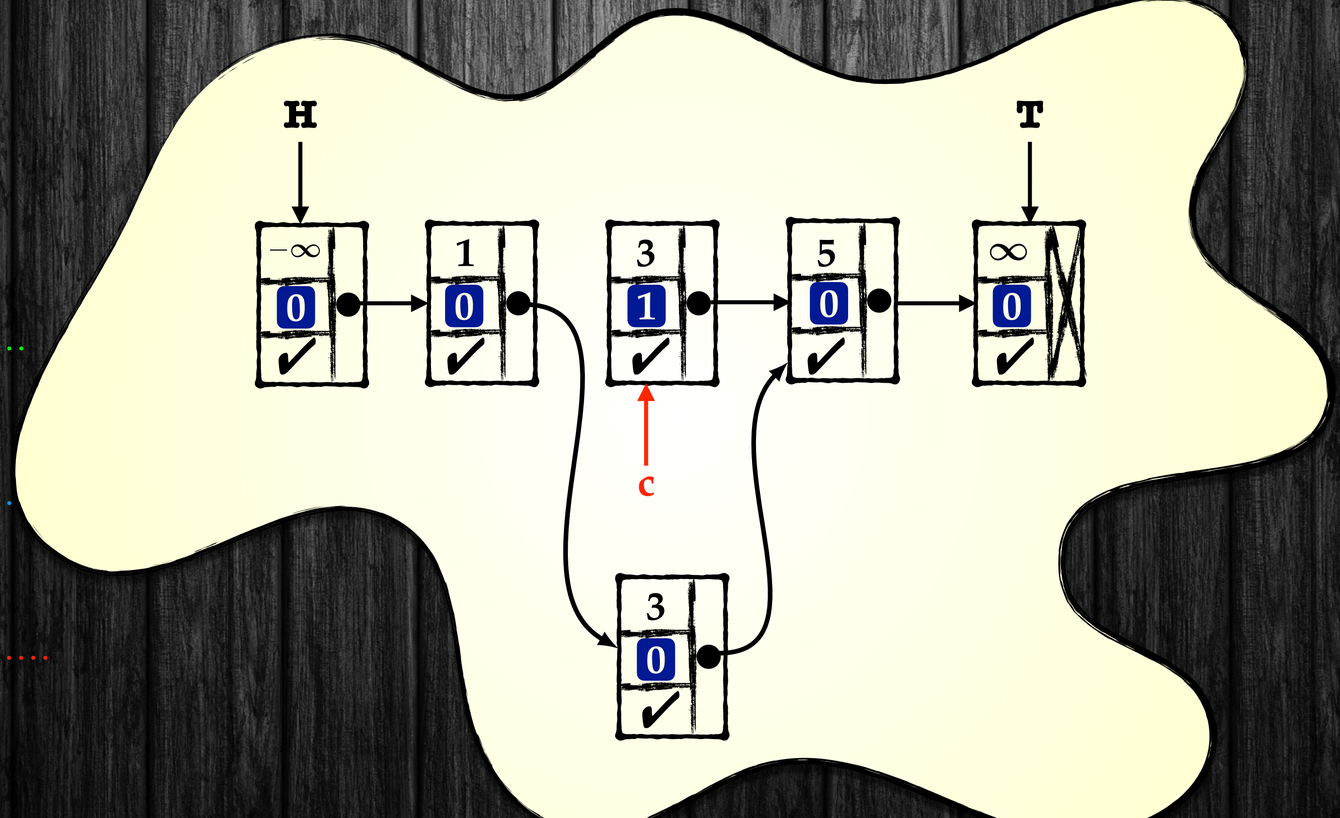
Sets | Lazy Synchronization

```
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;
```





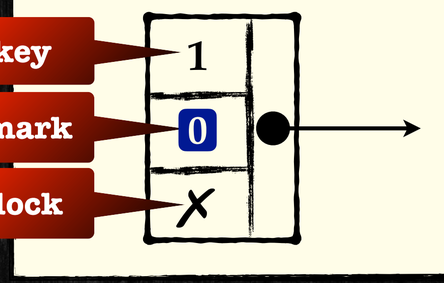


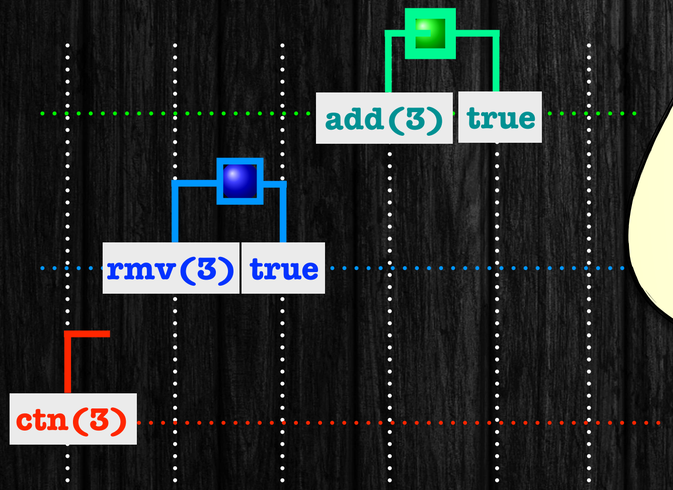


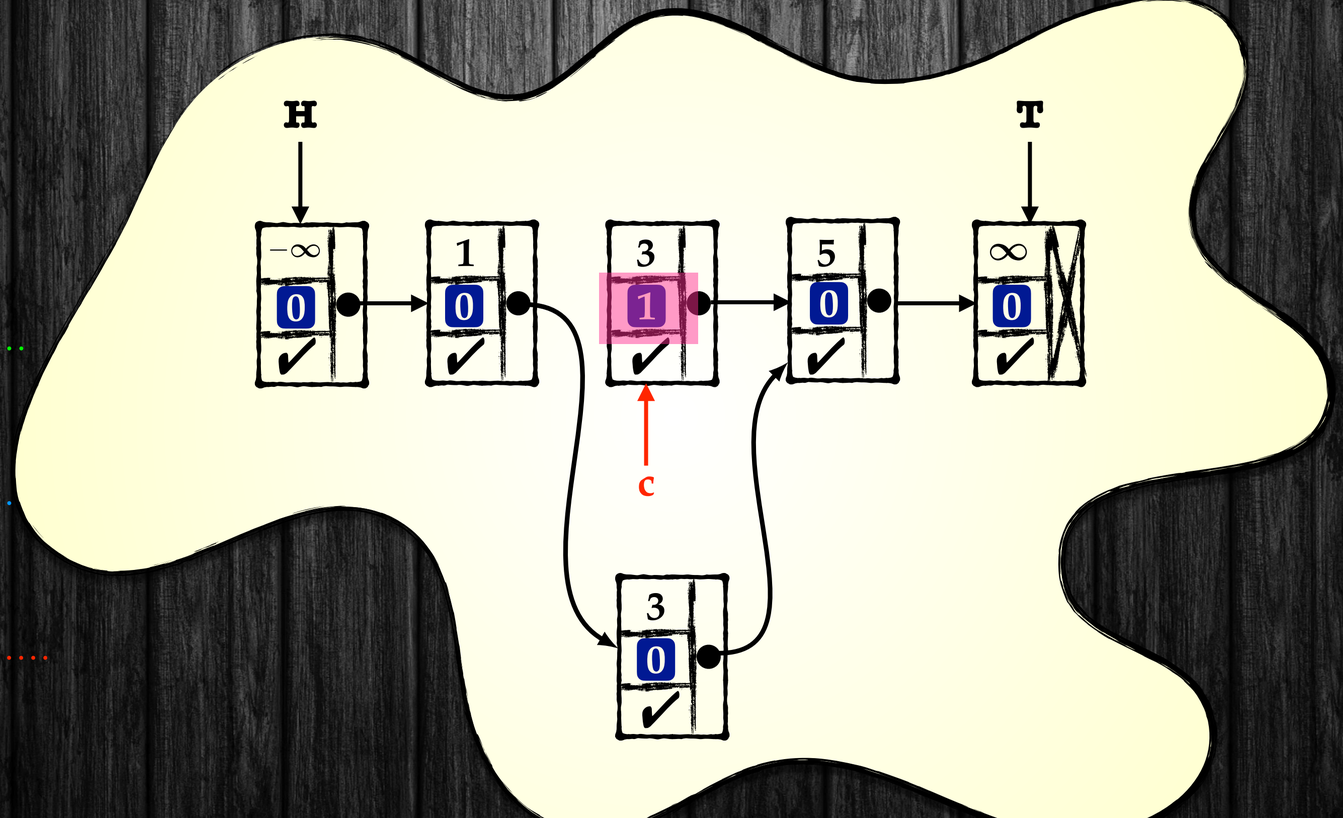
Sets | Lazy Synchronization

```
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)



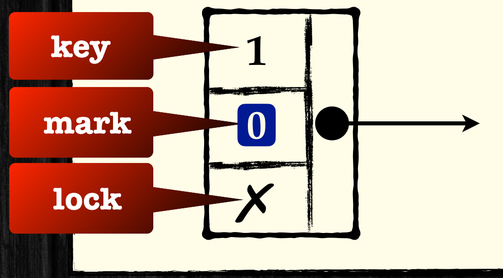




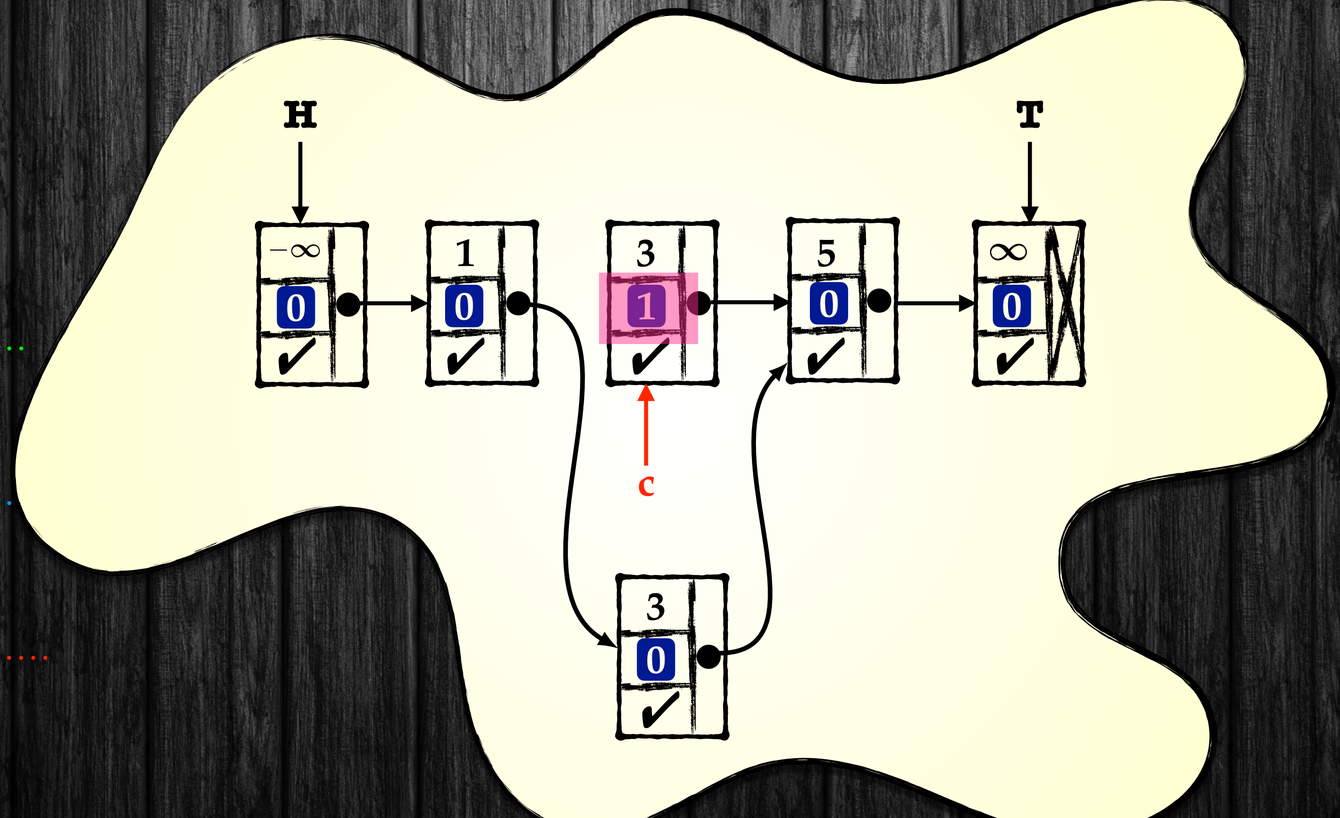
Sets | Lazy Synchronization

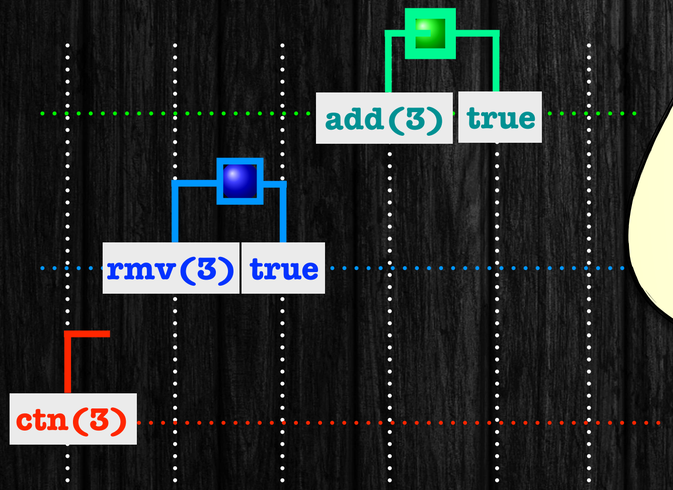
```
ctn(e):
local c, b
1  c := Head;
2  while (c.val < e)
3     c := c.next
4  b := c.mark
5  if (!b && c.val = e)
6     return true;
7  else return false;
```





linearization point





Sets | Lazy Synchronization

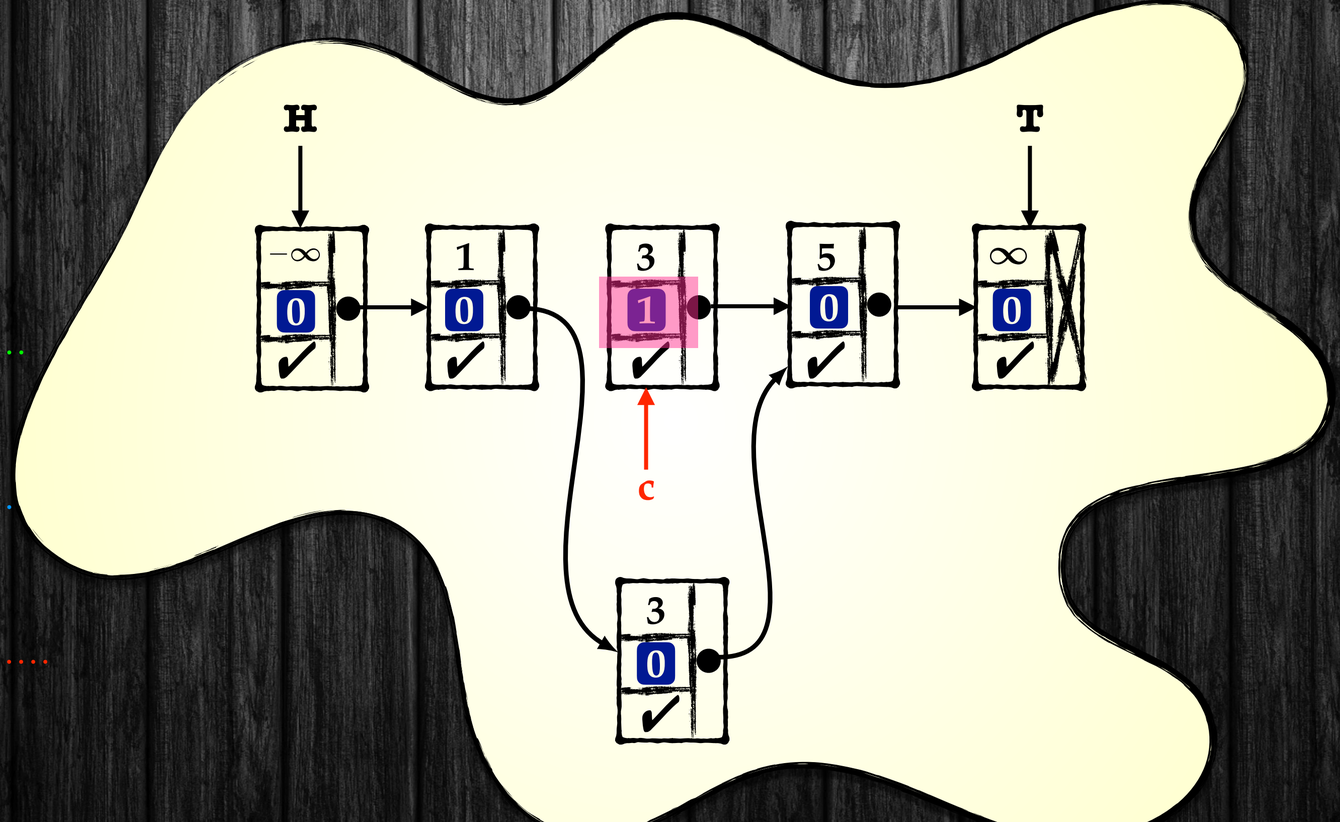
```
ctn(e):
local c, b
1  c := Head;
2  while (c.val < e)
3     c := c.next
4  b := c.mark
5  if (!b && c.val = e)
6     return true;
7  else return false;
```

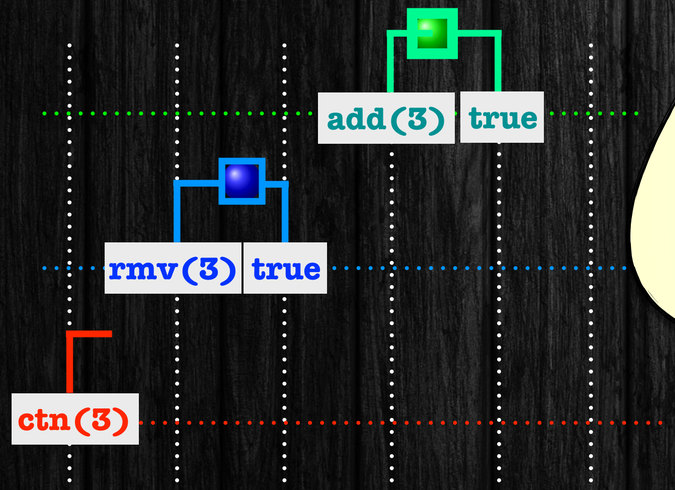
ctn(3)

key 1
mark 0
lock ✗

unsuccessful contains

linearization point





Sets | Lazy Synchronization

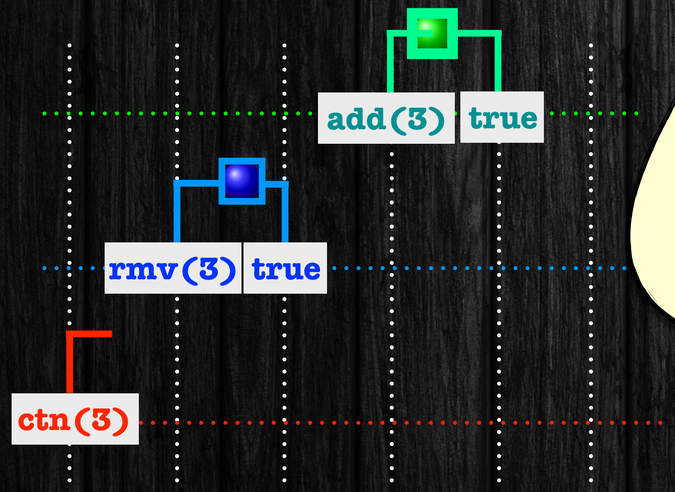
```
ctn(e):
local c, b
1  c := Head;
2  while (c.val < e)
3     c := c.next
4  b := c.mark
5  if (!b && c.val = e)
6     return true;
7  else return false;
```

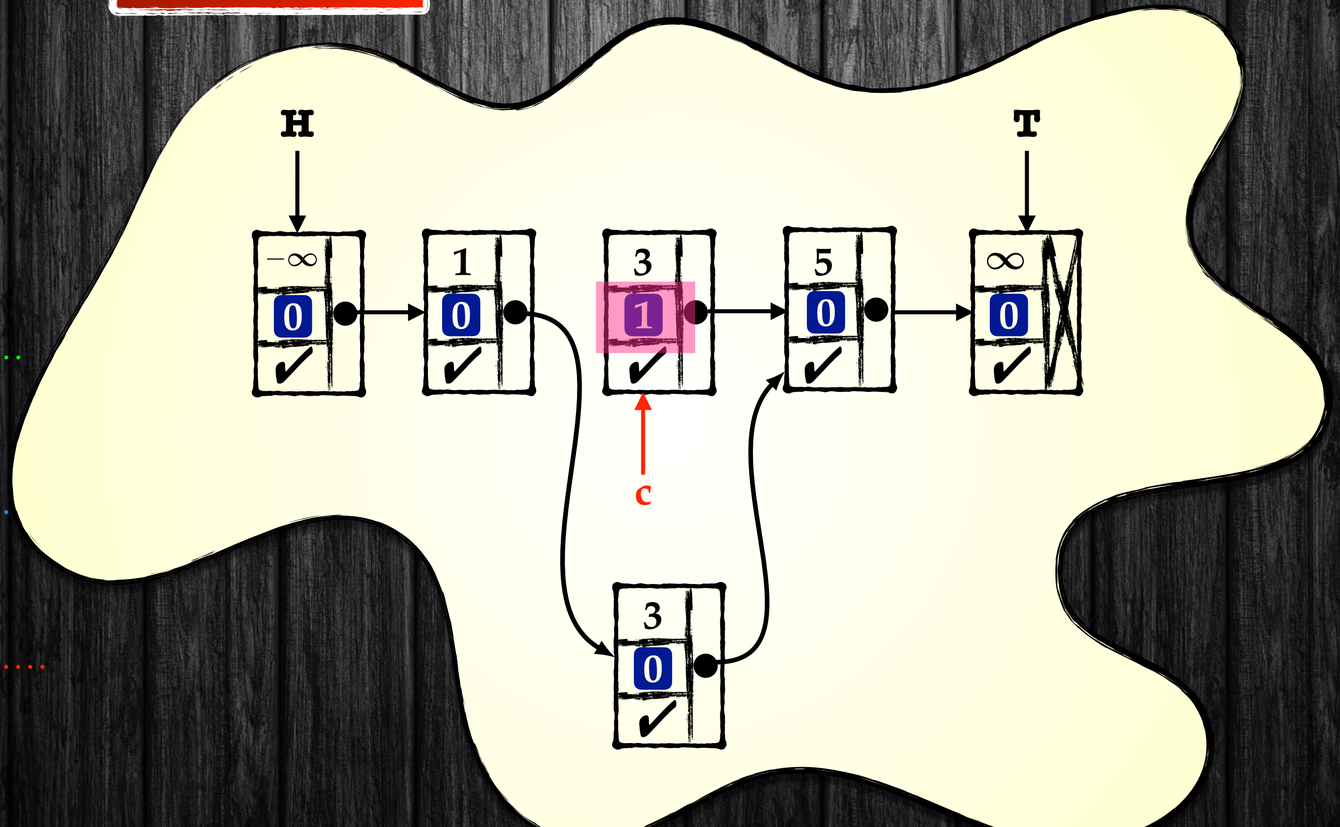
ctn(3)

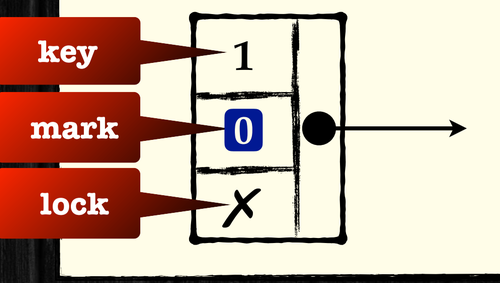
"when finding the cell with the input key marked"

unsuccessful contains

linearization point







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;
```

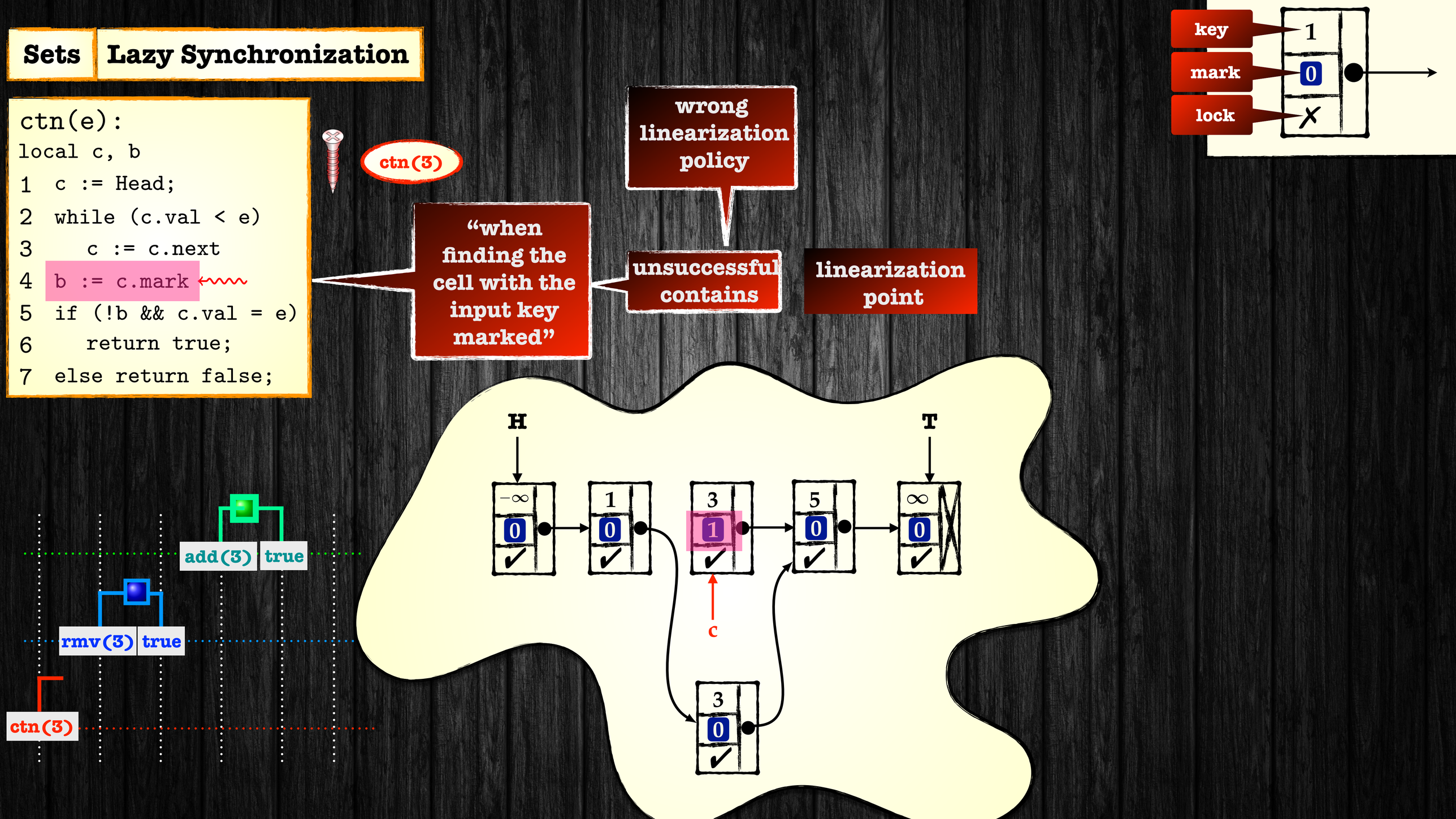
ctn(3)

"when finding the cell with the input key marked"

unsuccessful contains

wrong linearization policy

linearization point



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;
```

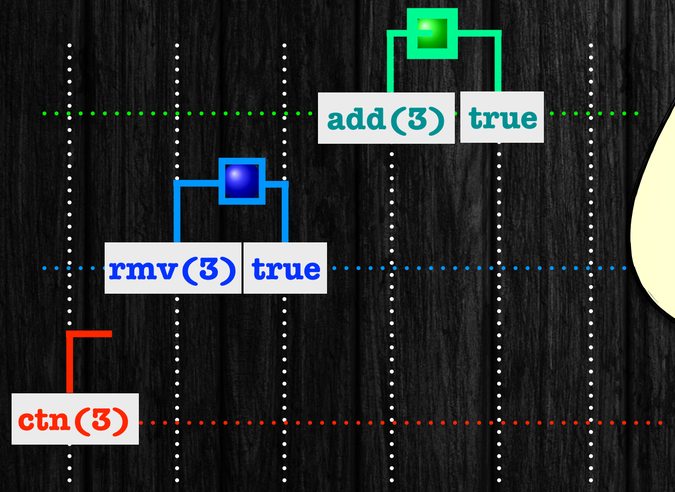
ctn(3)

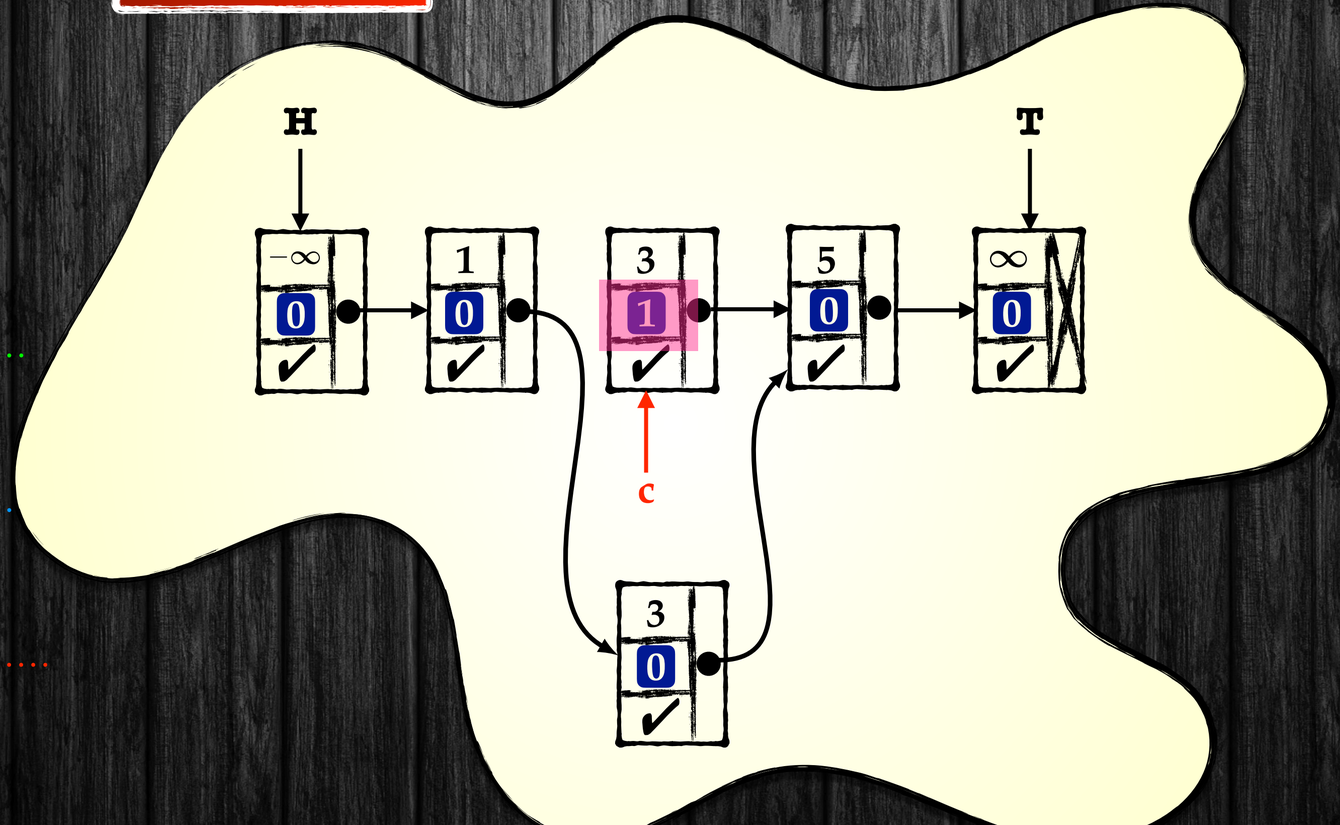
"when finding the cell with the input key marked"

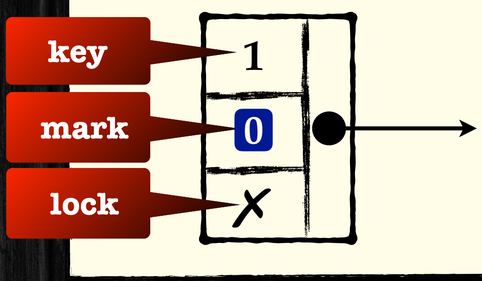
unsuccessful contains

wrong linearization policy

linearization point







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;
```

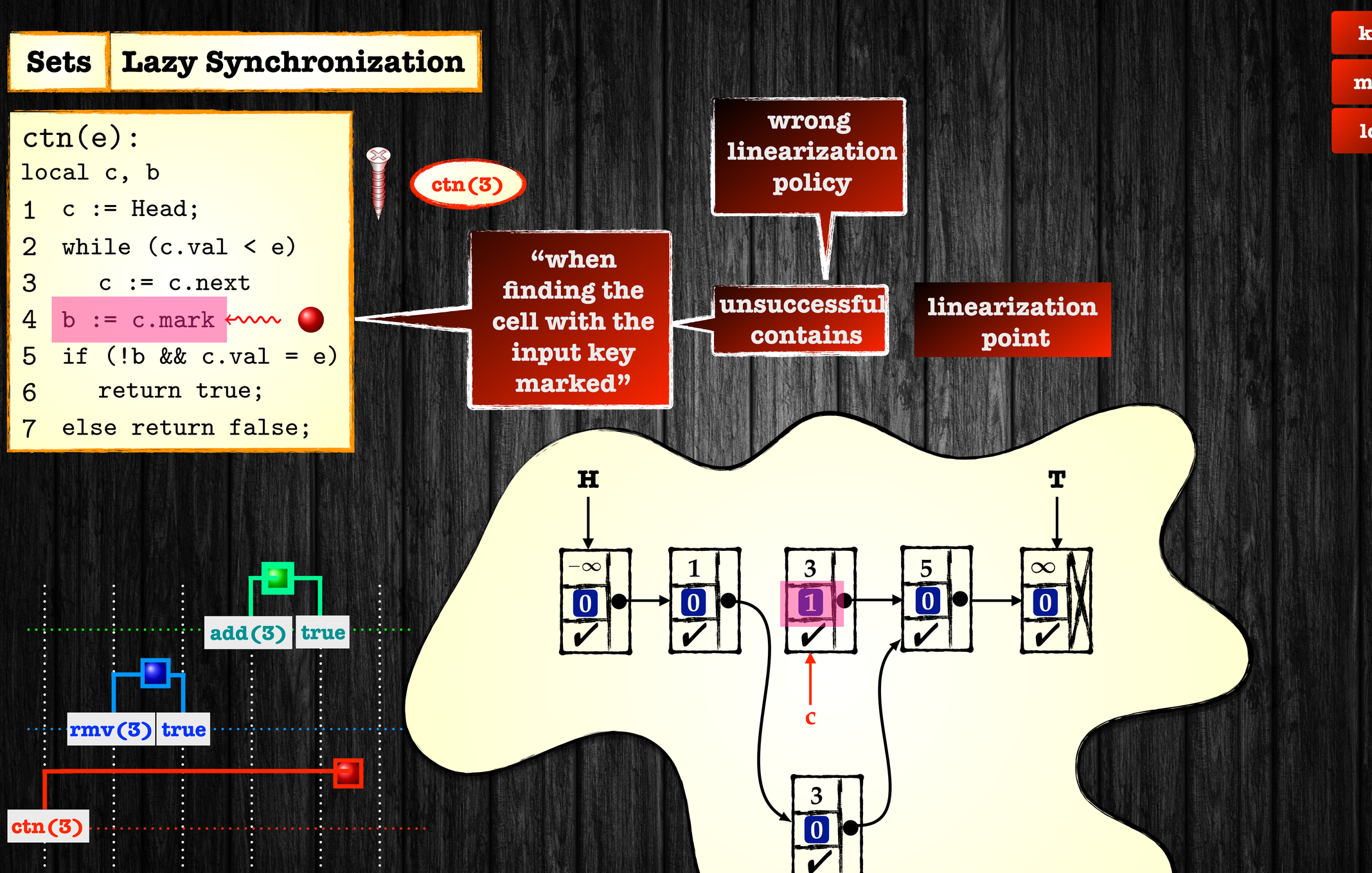
ctn(3)

"when finding the cell with the input key marked"

unsuccessful contains

wrong linearization policy

linearization point



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;
```

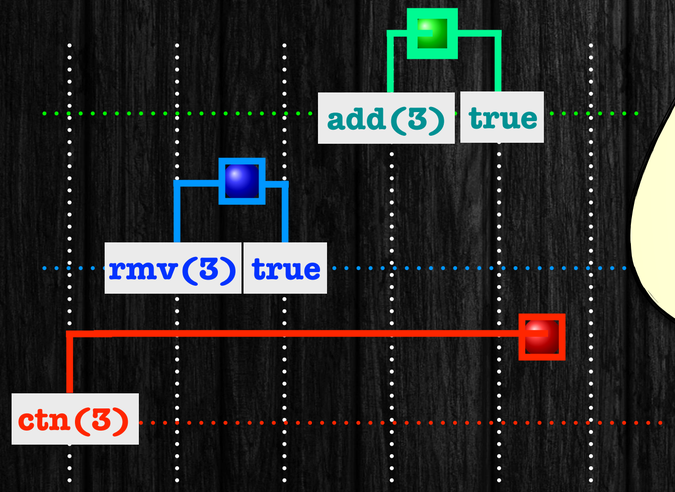
ctn(3)

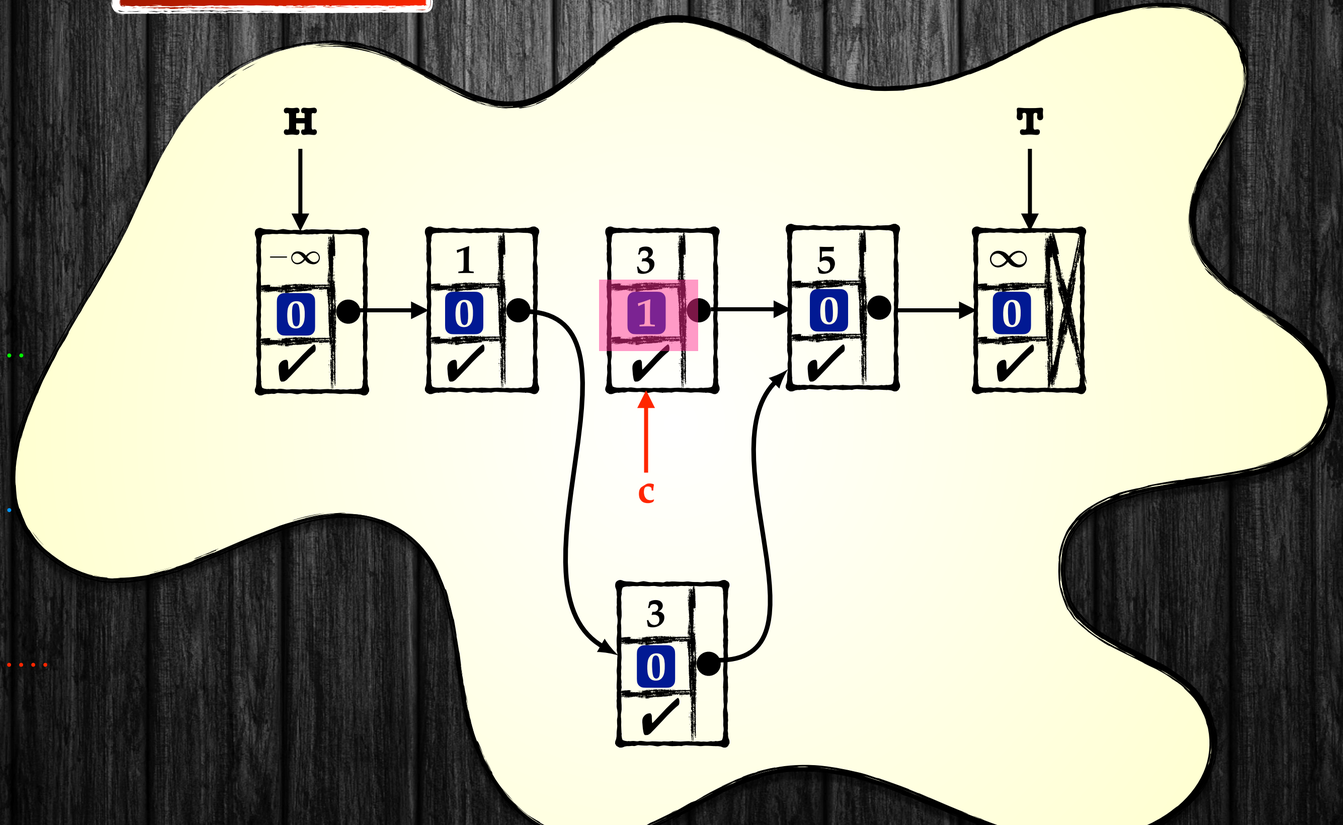
wrong linearization policy

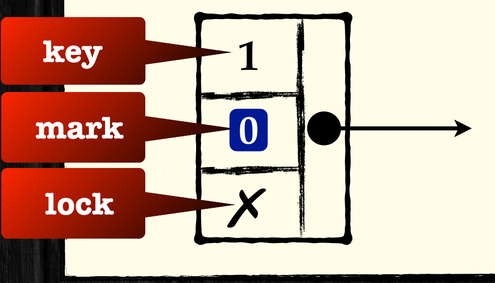
"when finding the cell with the input key marked"

unsuccessful contains

linearization point







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;
```

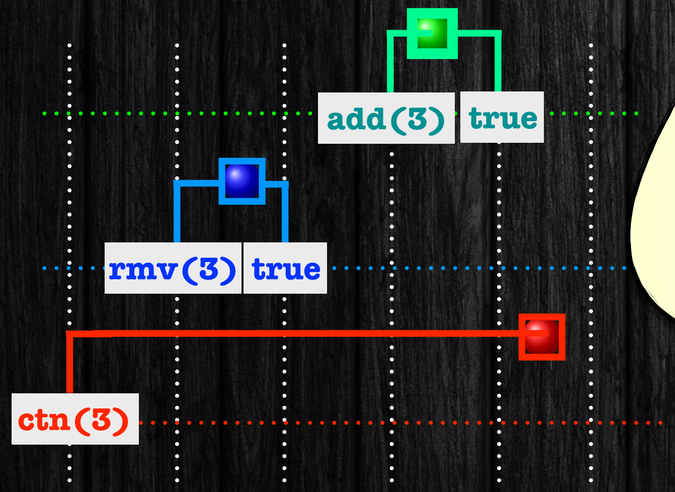
ctn(3)

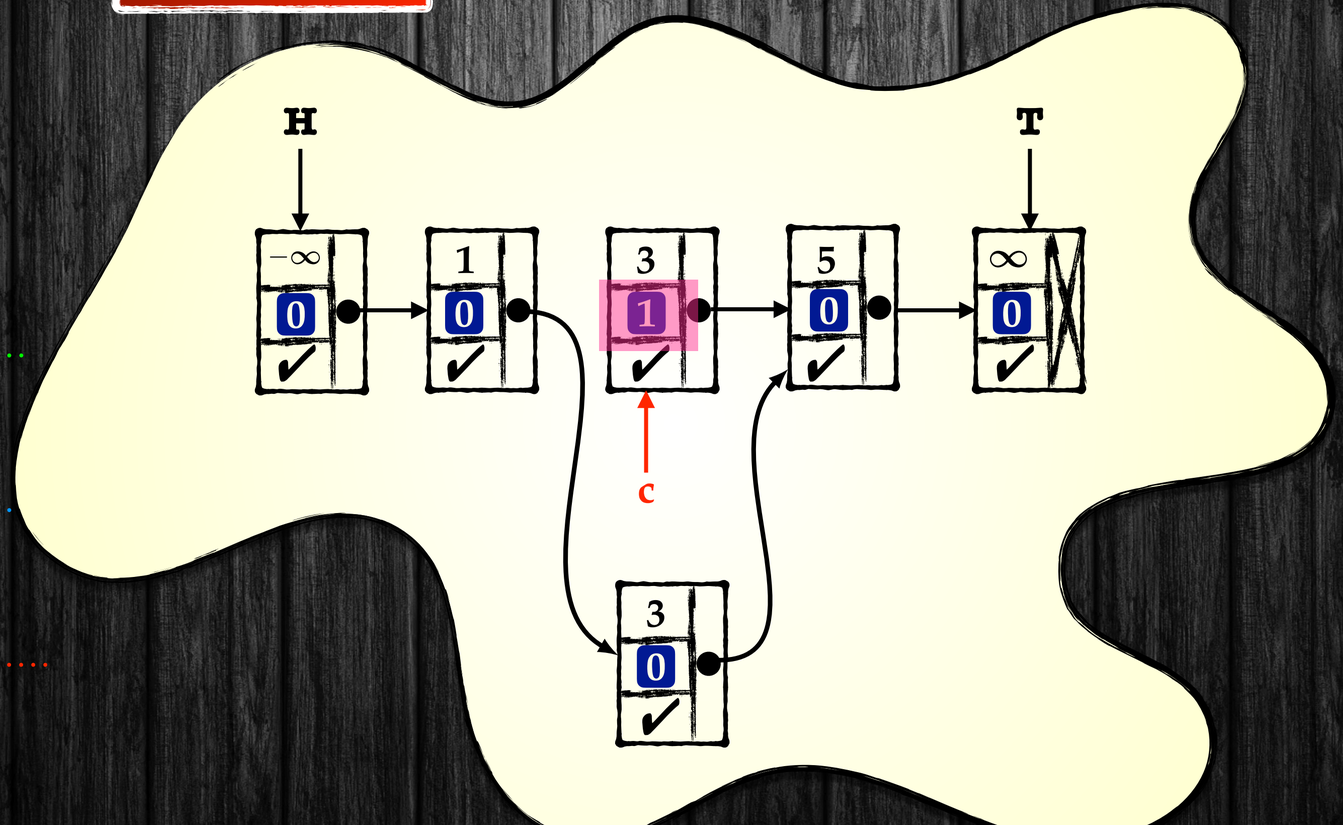
wrong linearization policy

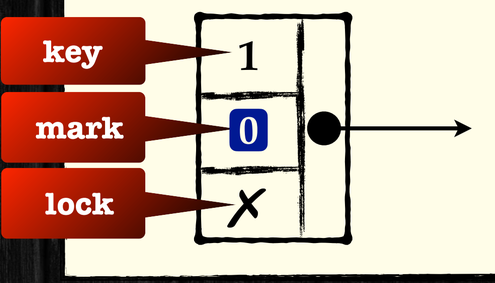
"when finding the cell with the input key marked"

unsuccessful contains

linearization point







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;
```

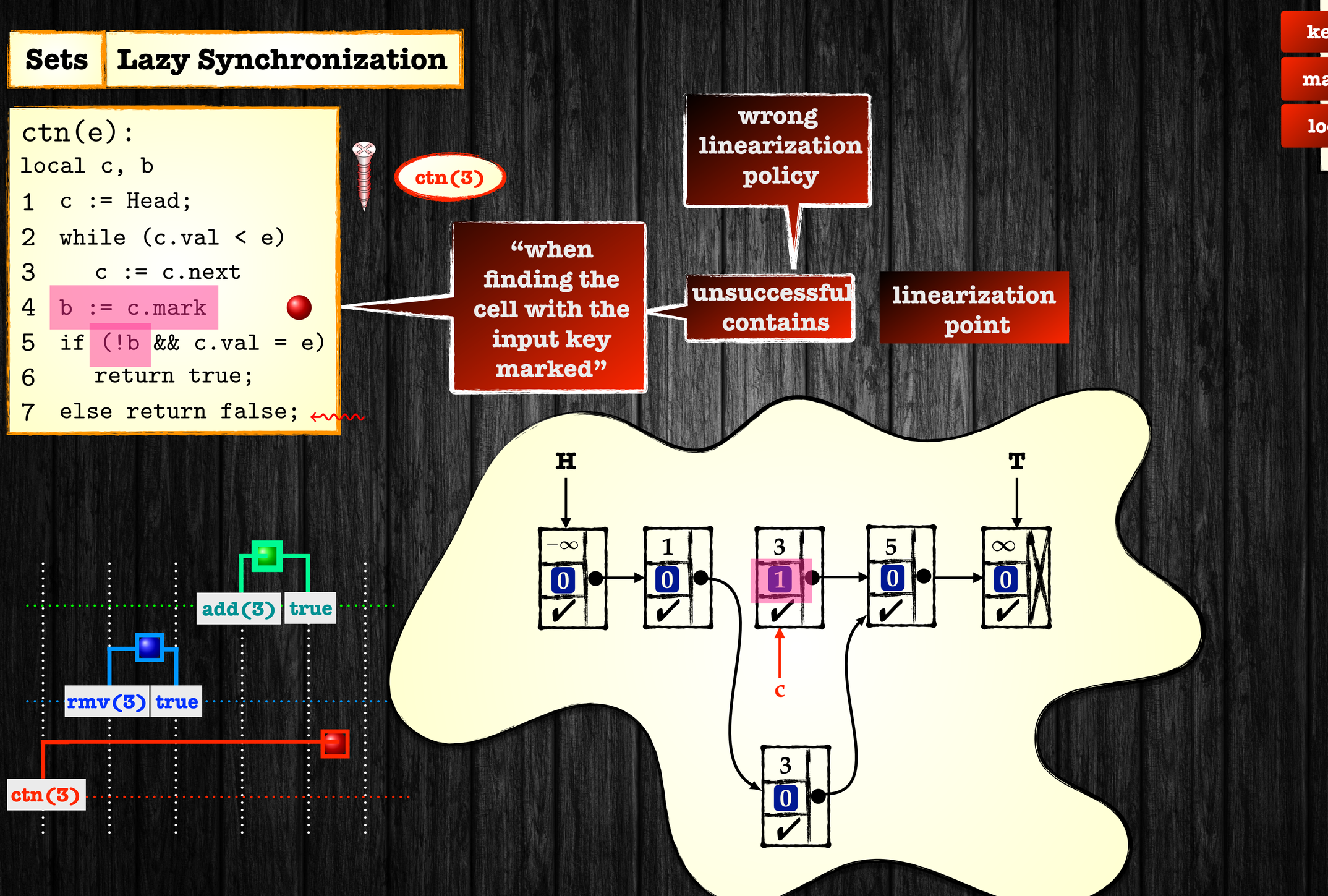
ctn(3)

"when finding the cell with the input key marked"

unsuccessful contains

wrong linearization policy

linearization point



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)

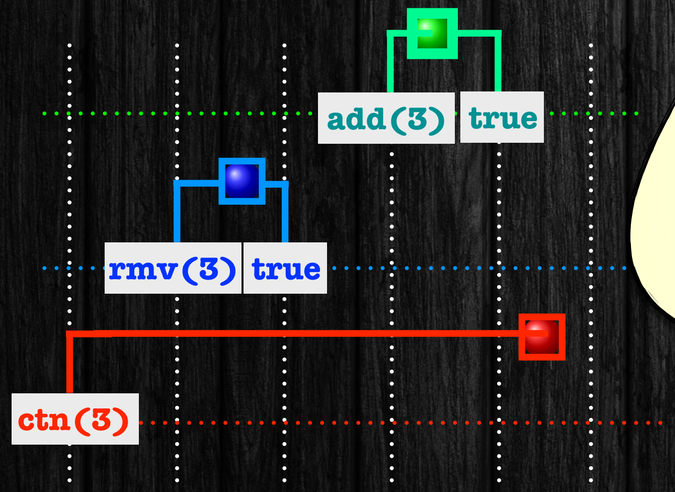
"when finding the cell with the input key marked"

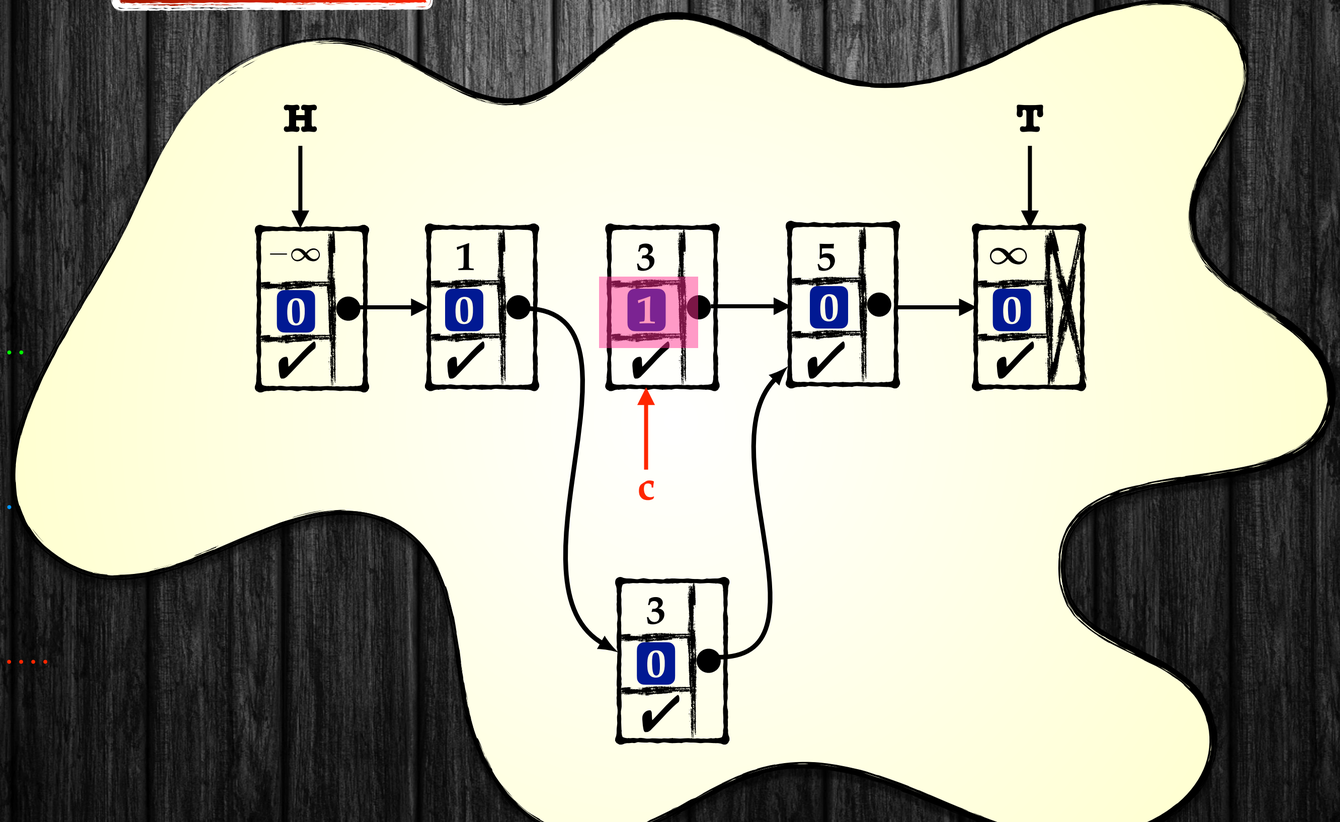
unsuccessful contains

wrong linearization policy

linearization point







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;
```

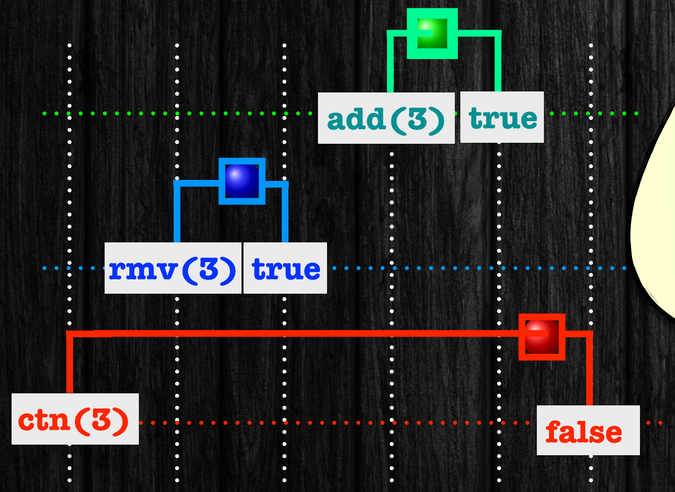
ctn(3)

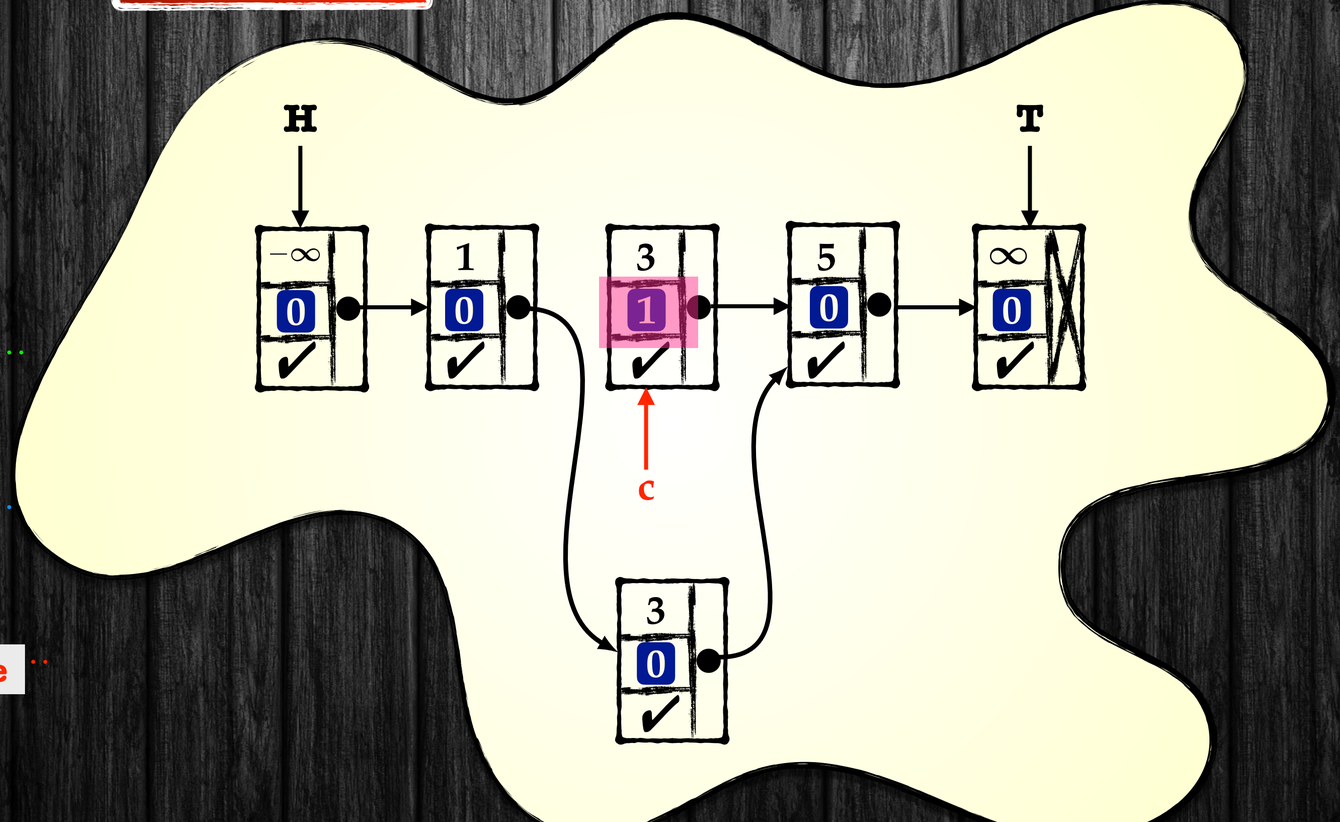
"when finding the cell with the input key marked"

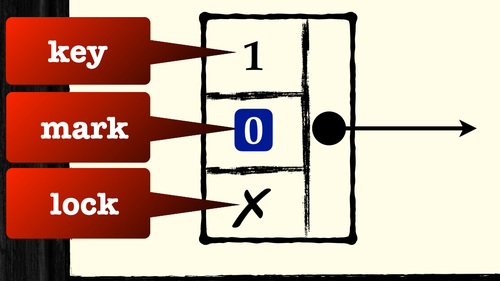
unsuccessful contains

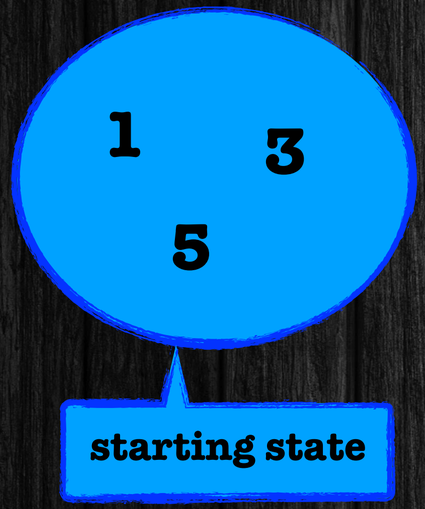
wrong linearization policy

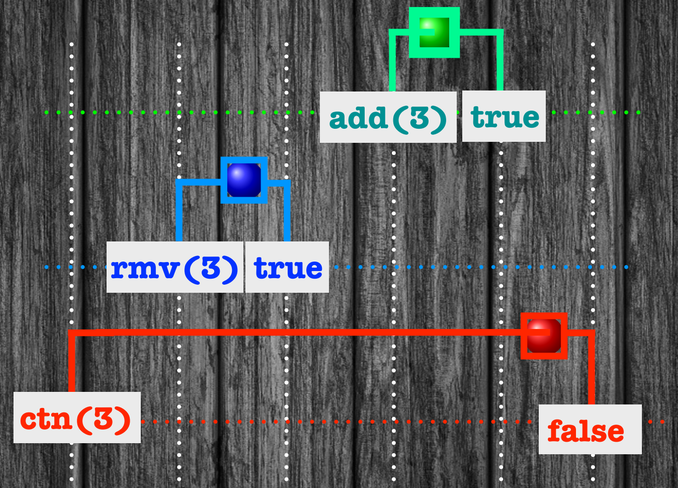
linearization point

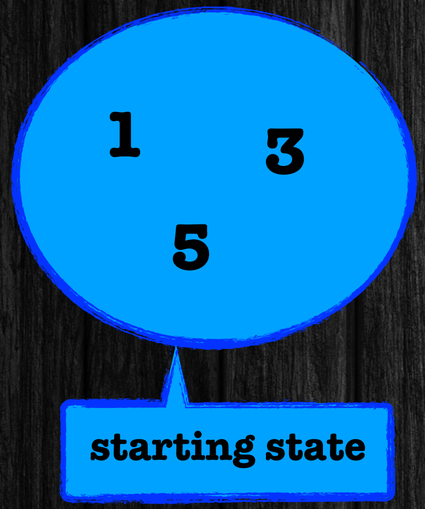


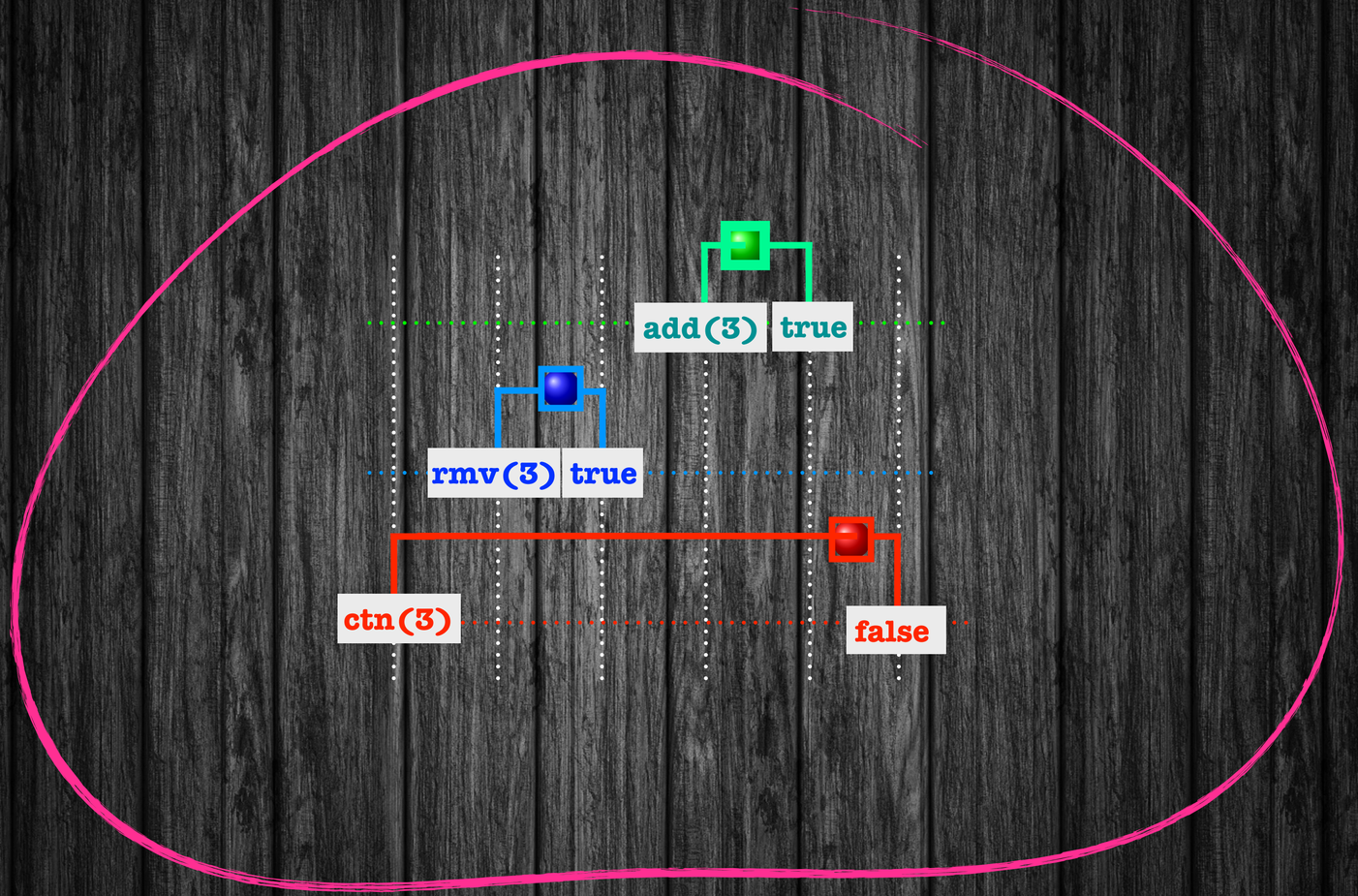


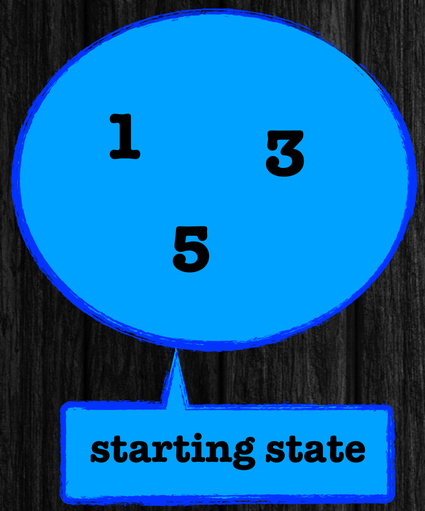


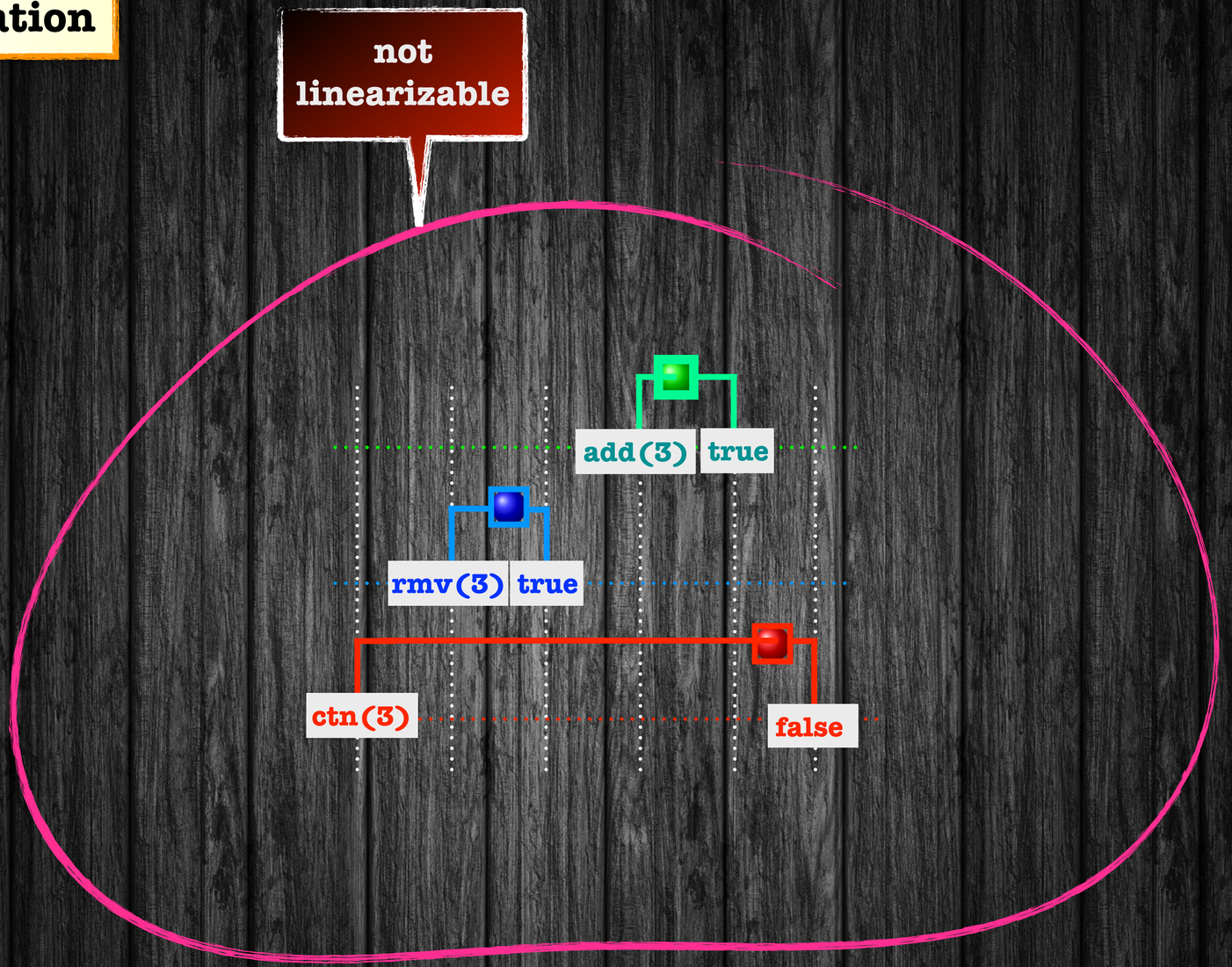

1
3
5
starting state


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

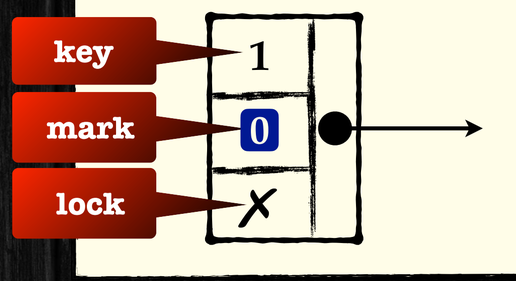

1
3
5
starting state

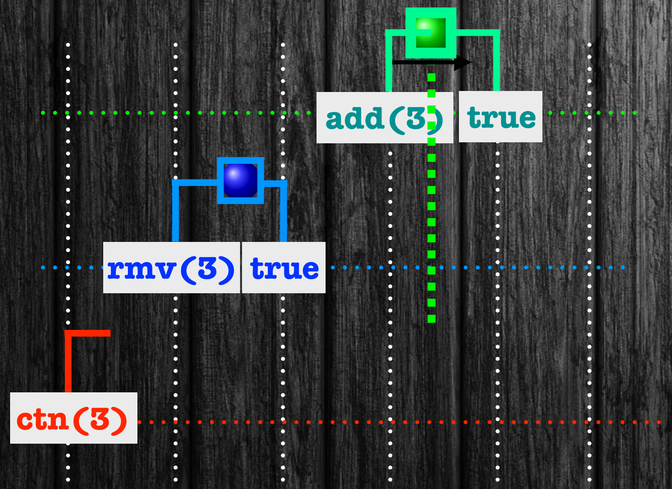

add(3)
true
rmv(3)
true
ctn(3)
false


1
3
5
starting state

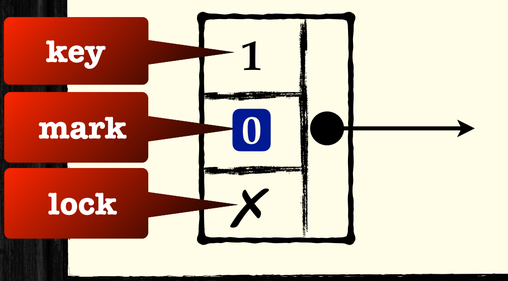

not linearizable
add(3)
true
rmv(3)
true
ctn(3)
false

Sets | Lazy Synchronization

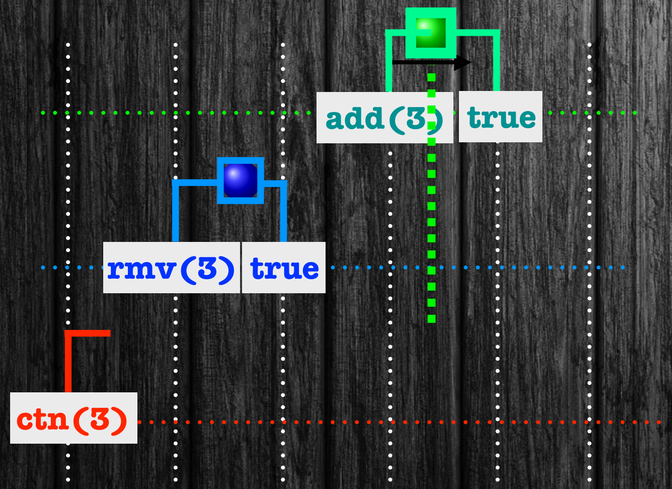


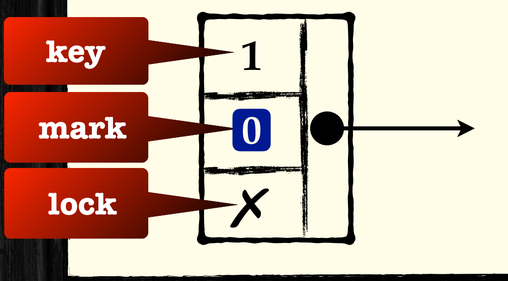


Sets | Lazy Synchronization



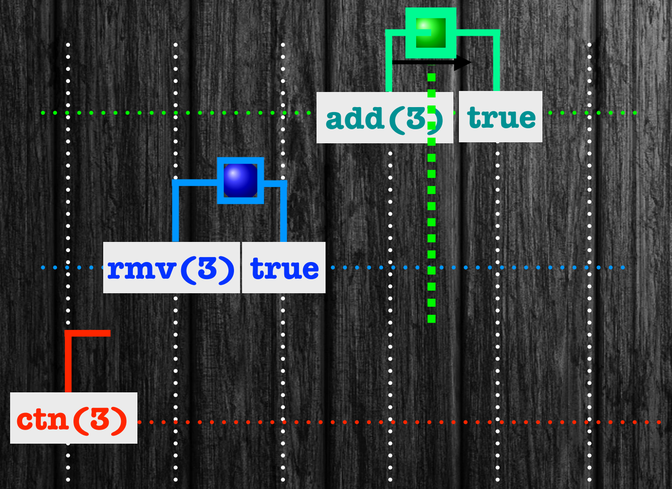
linearization point





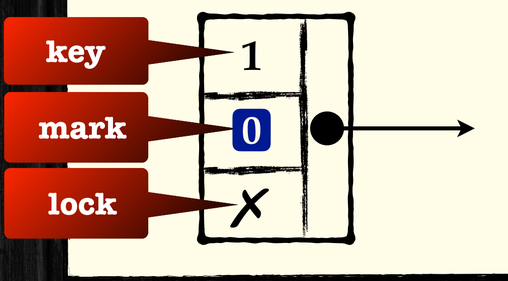
correct linearization policy

linearization point

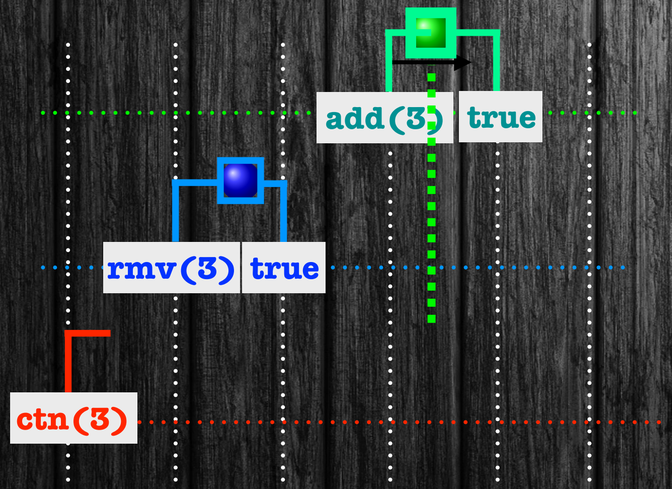


unsuccessful contains

correct linearization policy



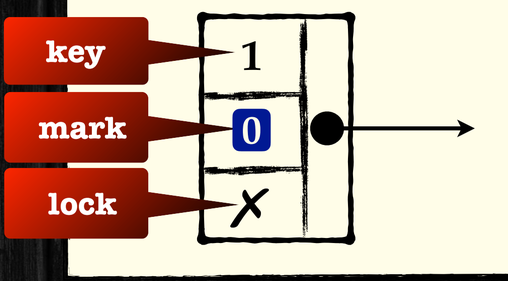
linearization point



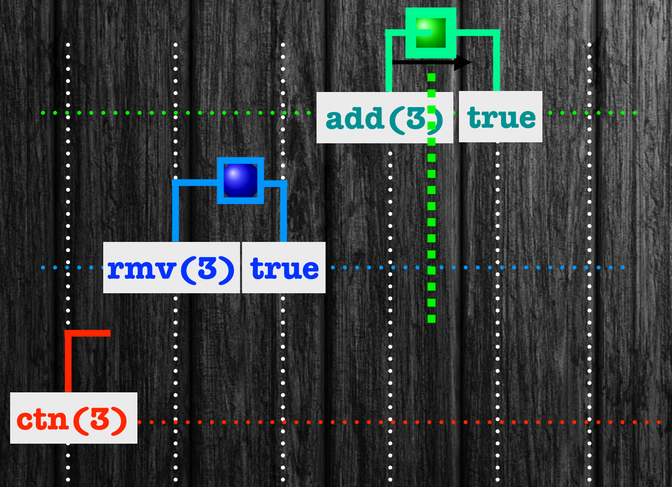
the earliest time point where

unsuccessful contains

correct linearization policy



linearization point



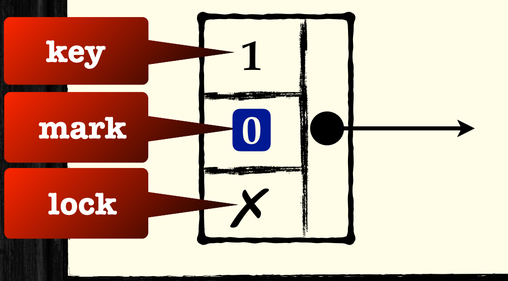
Sets | Lazy Synchronization

the earliest time point where

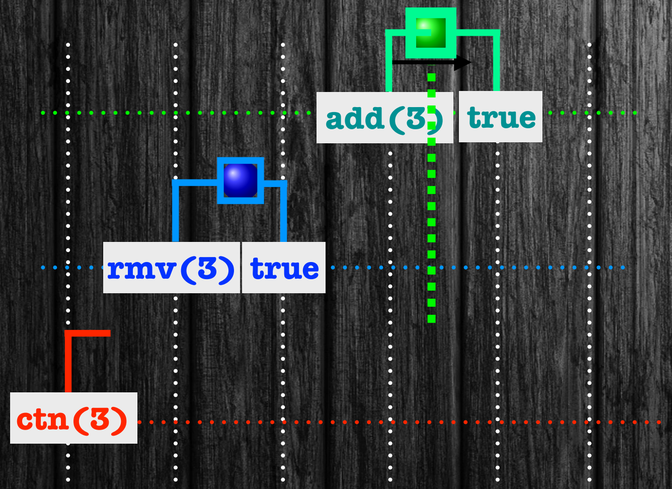
unsuccessful contains

1 we find node with key > input key

correct linearization policy



linearization point



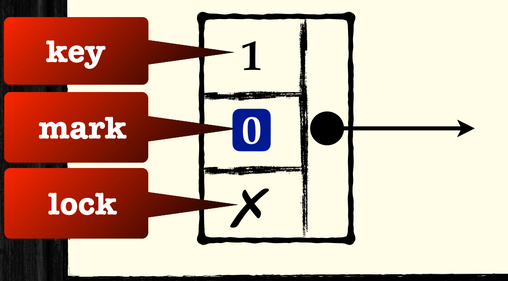
Sets | Lazy Synchronization

the earliest time point where

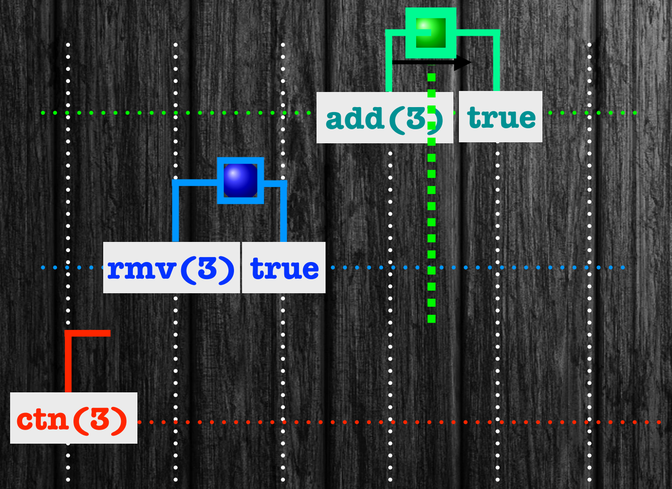
unsuccessful contains

1. we find node with key > input key
2. find marked node with key = input key

correct linearization policy



linearization point



Sets | Lazy Synchronization

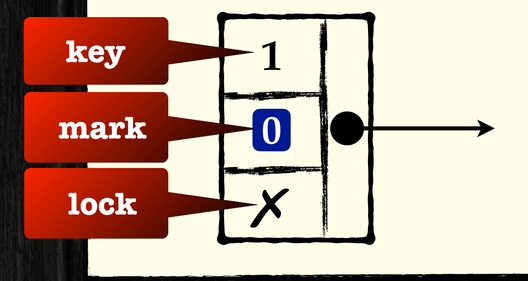
the earliest time point where

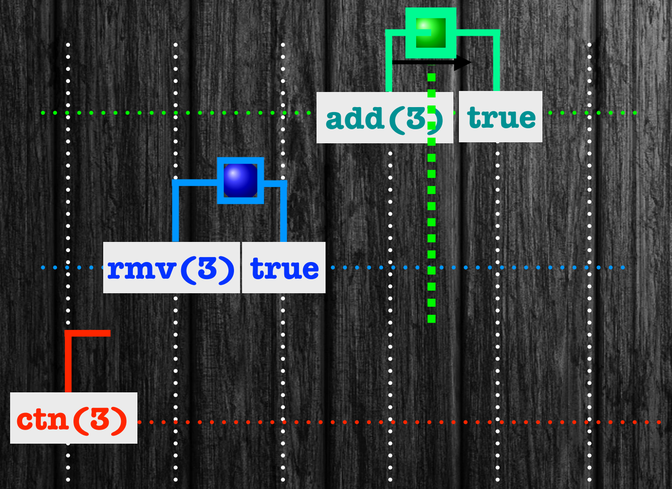
unsuccessful contains

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

linearization point





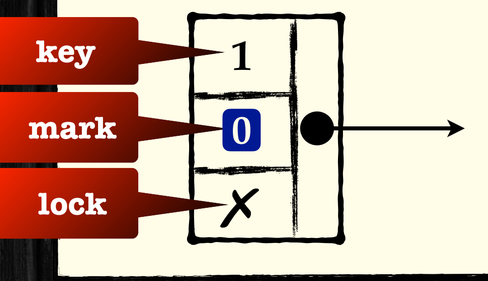
the earliest time point where

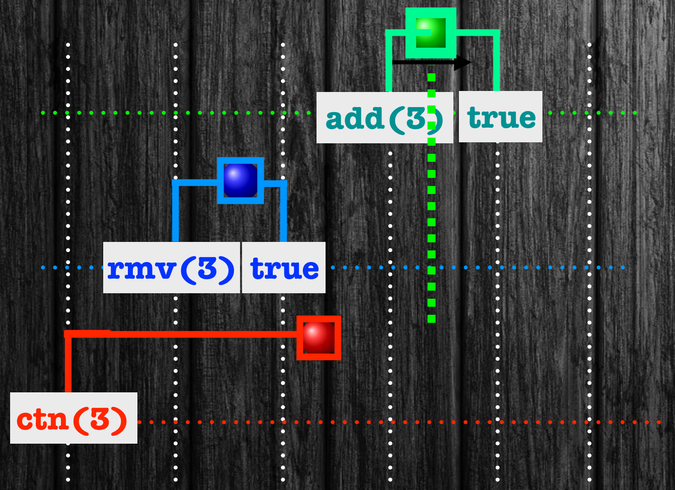
unsuccessful contains

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

linearization point



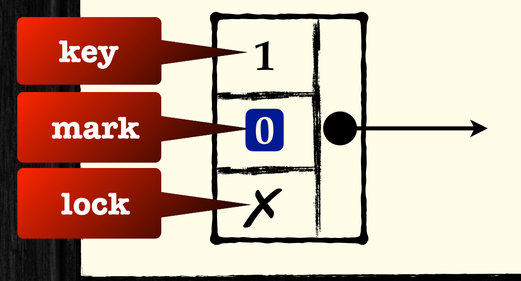


the earliest time point where

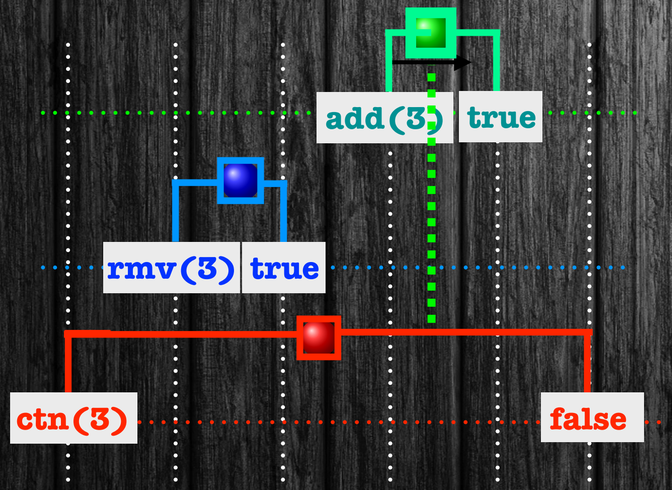
unsuccessful contains

correct linearization policy

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



linearization point

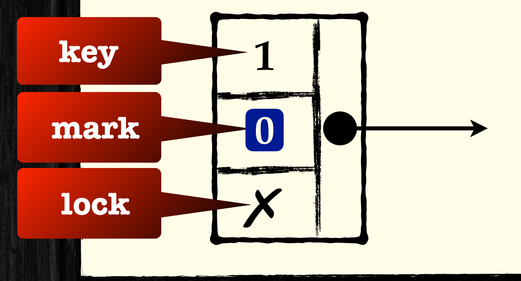


the earliest time point where

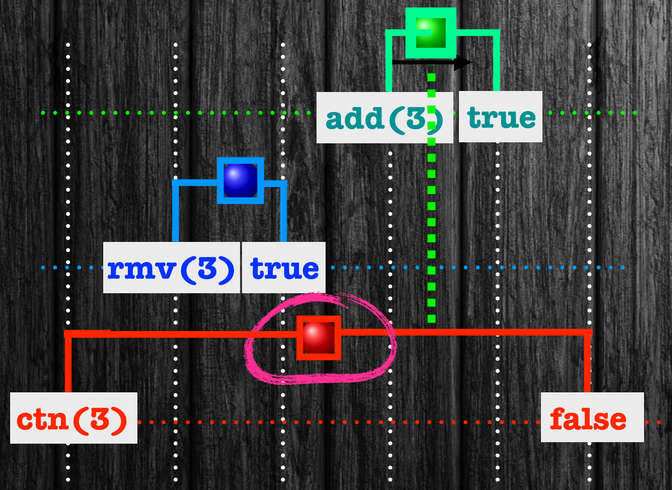
unsuccessful contains

correct linearization policy

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



linearization point

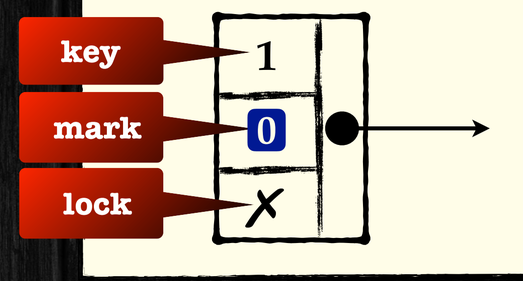


the earliest time point where

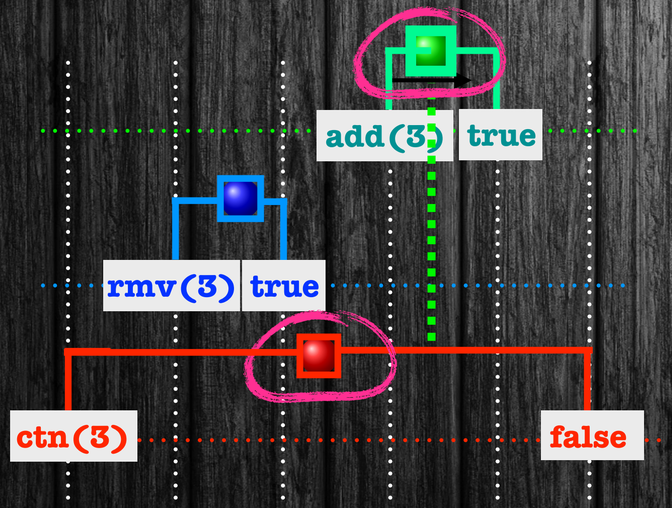
unsuccessful contains

correct linearization policy

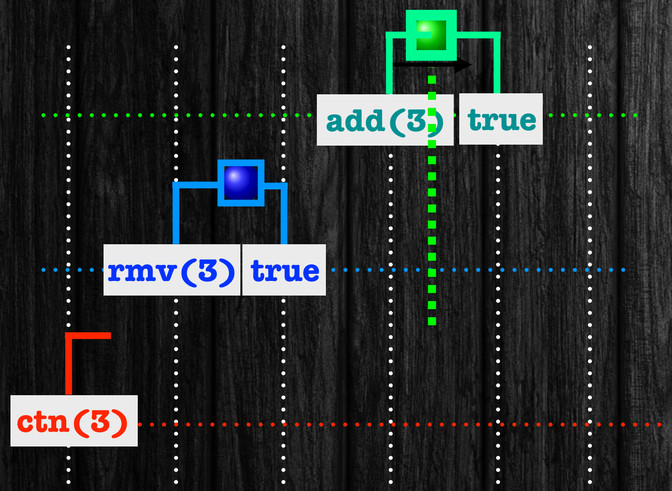
1. we find node with key > input key
2. find marked node with key = input key
3. just before a new node added with key = input key



linearization point



Sets | Lazy Synchronization



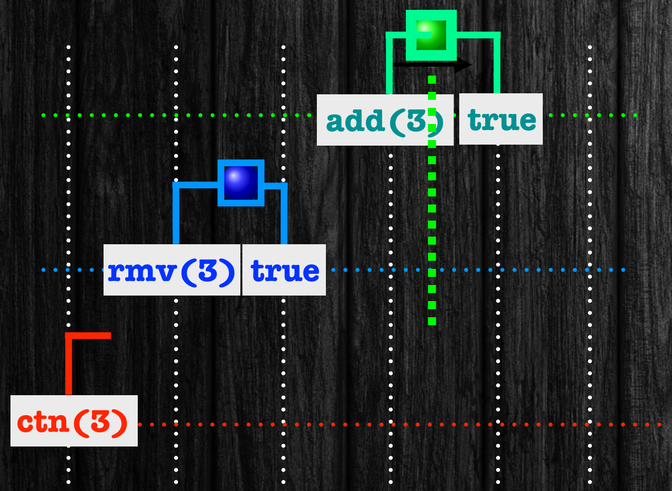
linearization point

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

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



linearization point

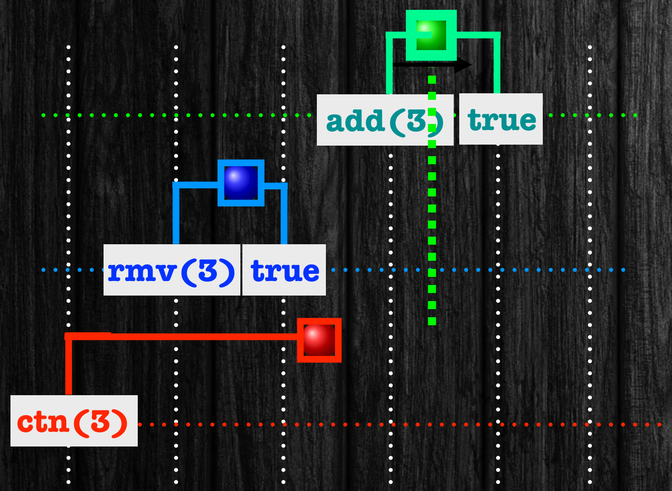
the earliest time point where

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

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	?

correct linearization policy

Sets | Lazy Synchronization



linearization point

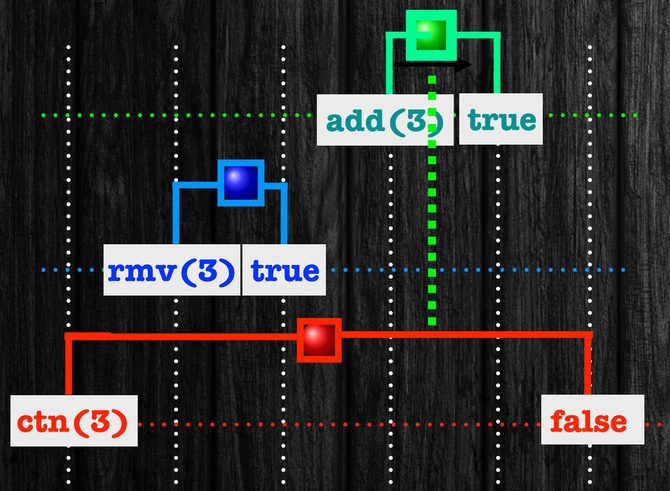
the earliest time point where

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

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	?

correct linearization policy

Sets | Lazy Synchronization



linearization point

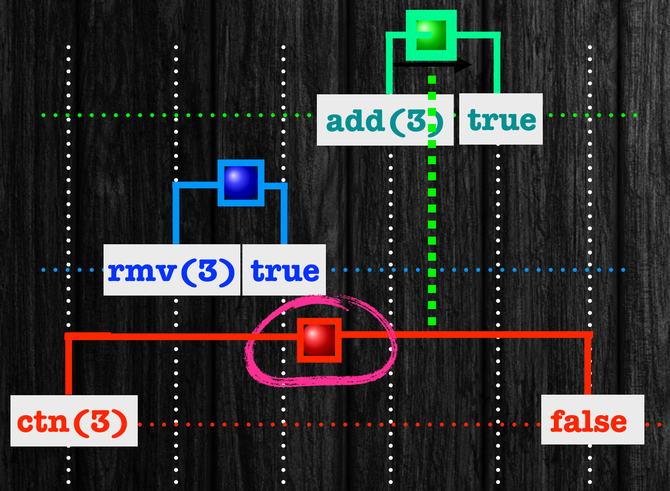
the earliest time point where

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

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	?

correct linearization policy

Sets | Lazy Synchronization



linearization point

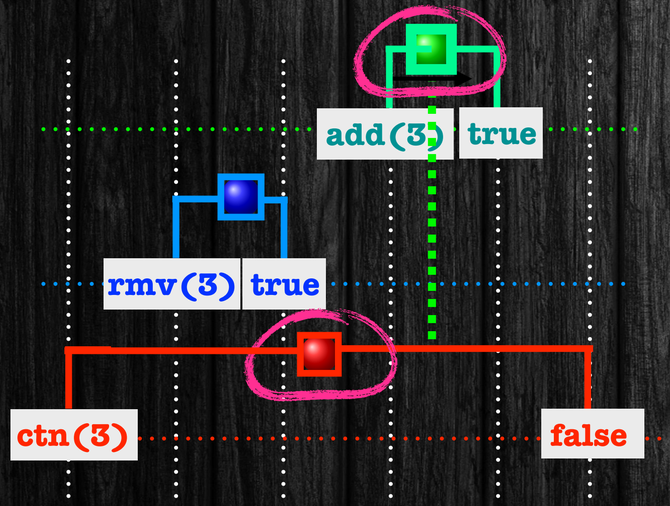
the earliest time point where

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

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	?

correct linearization policy

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

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	?

correct linearization policy