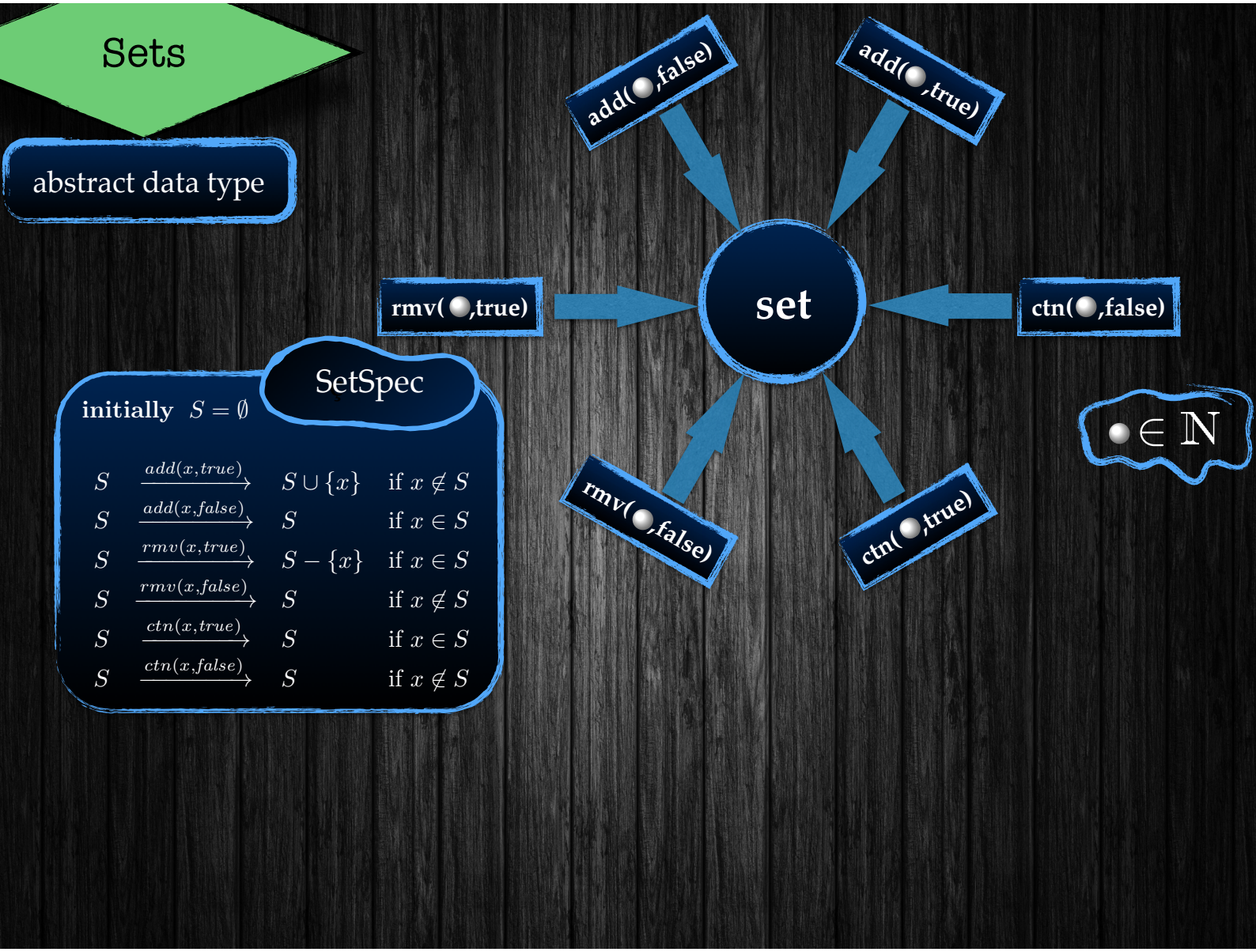
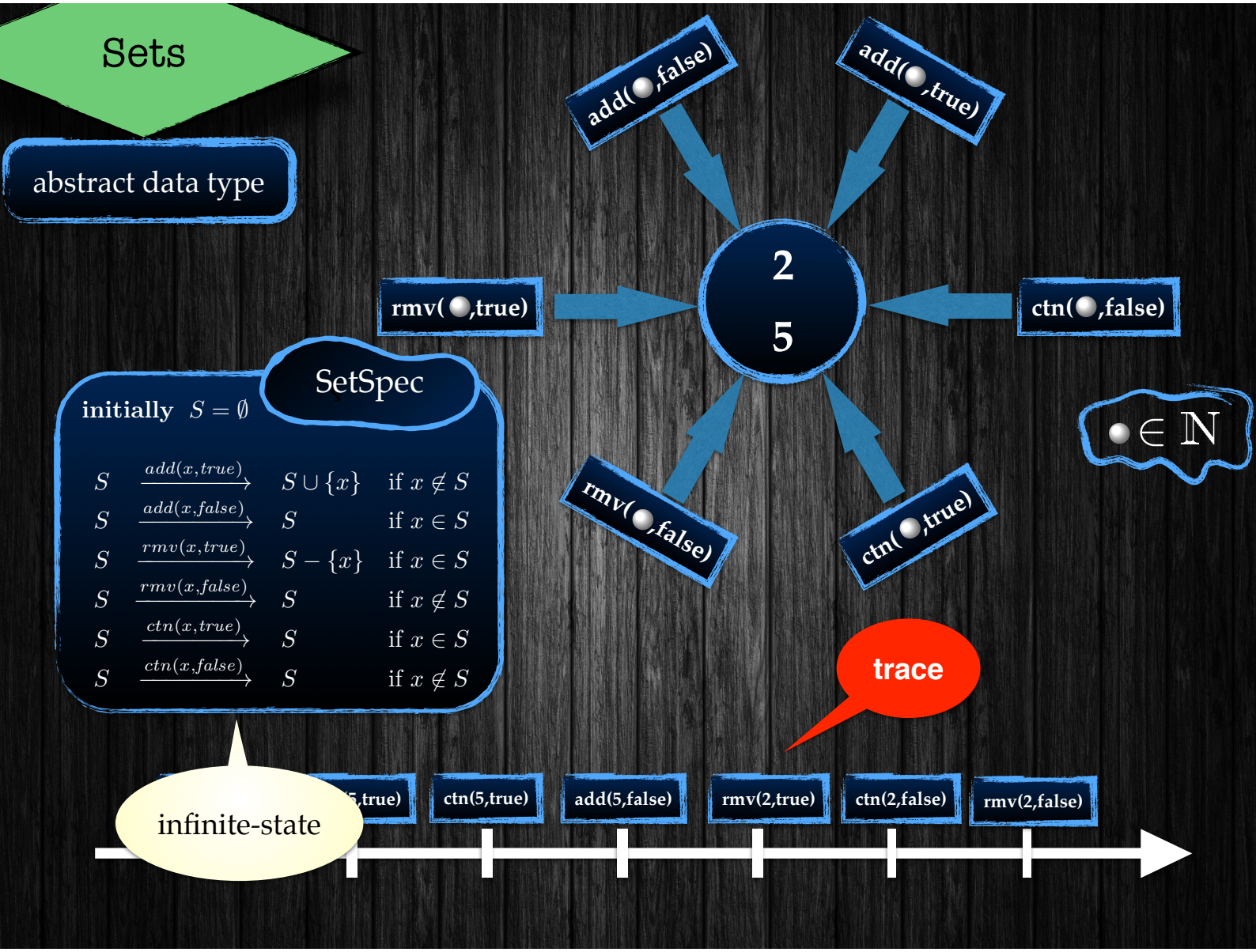
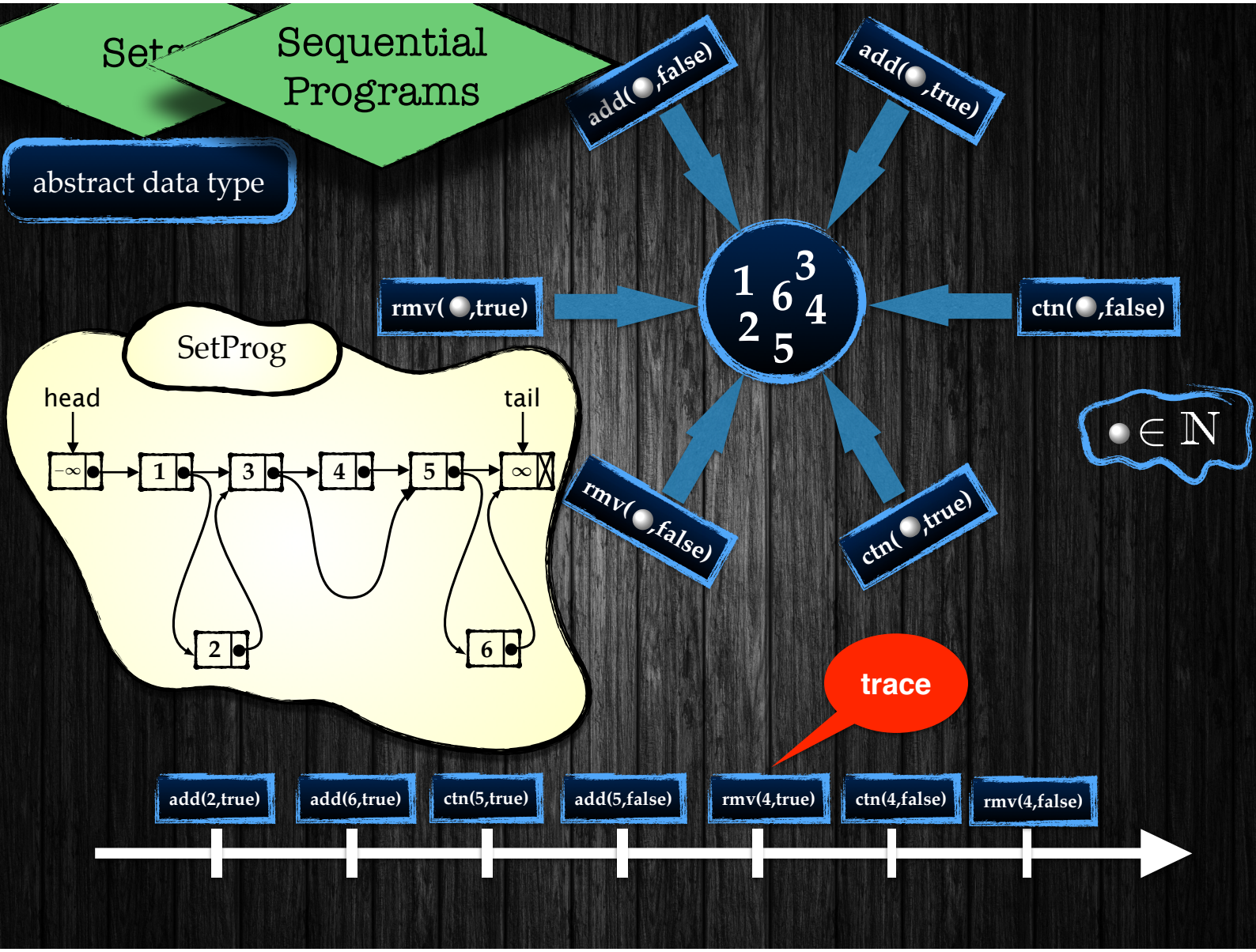
Verifying Linearizability

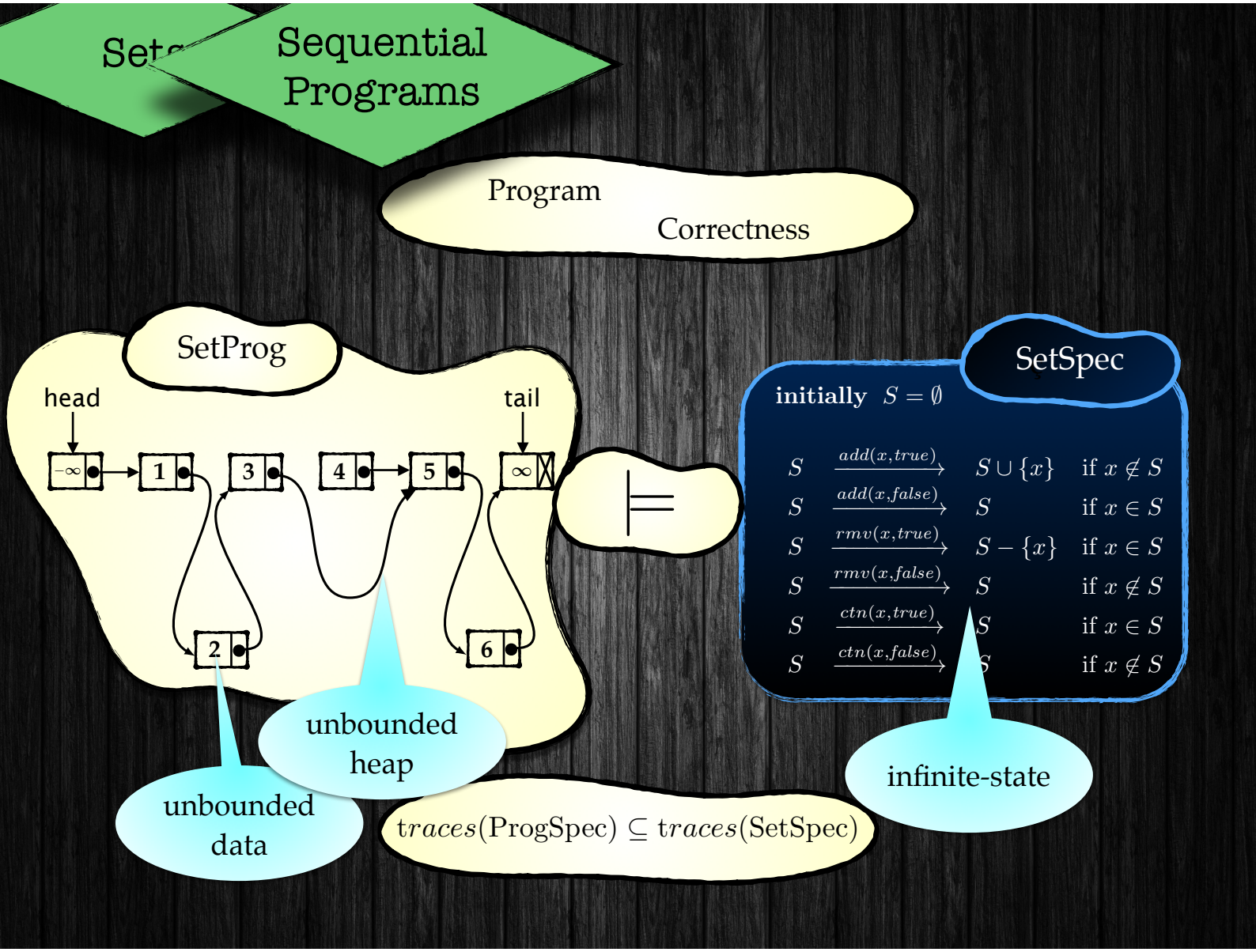
Parosh Aziz Abdulla, Bengt Jonsson, Cong Quy Trinh

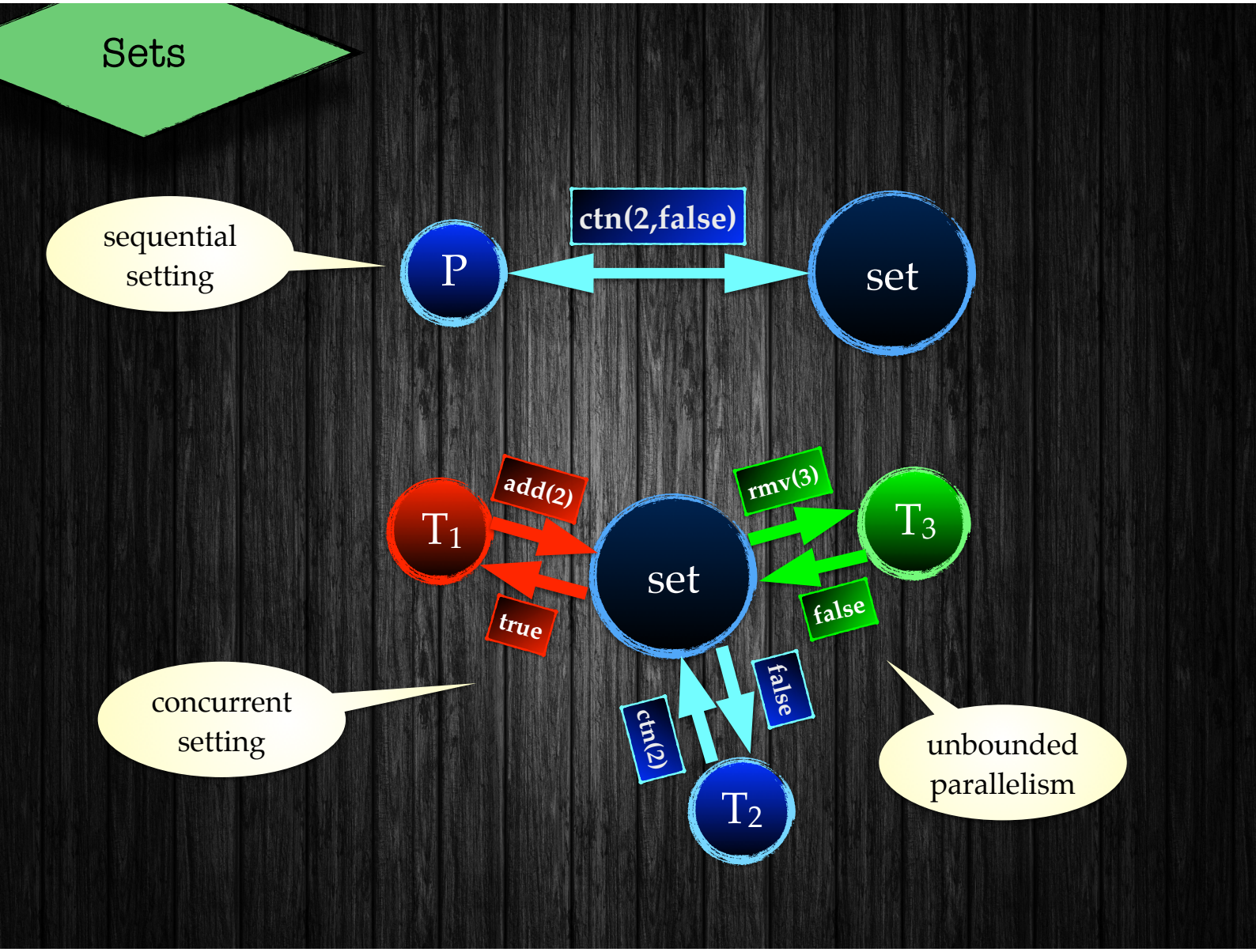
Uppsala University

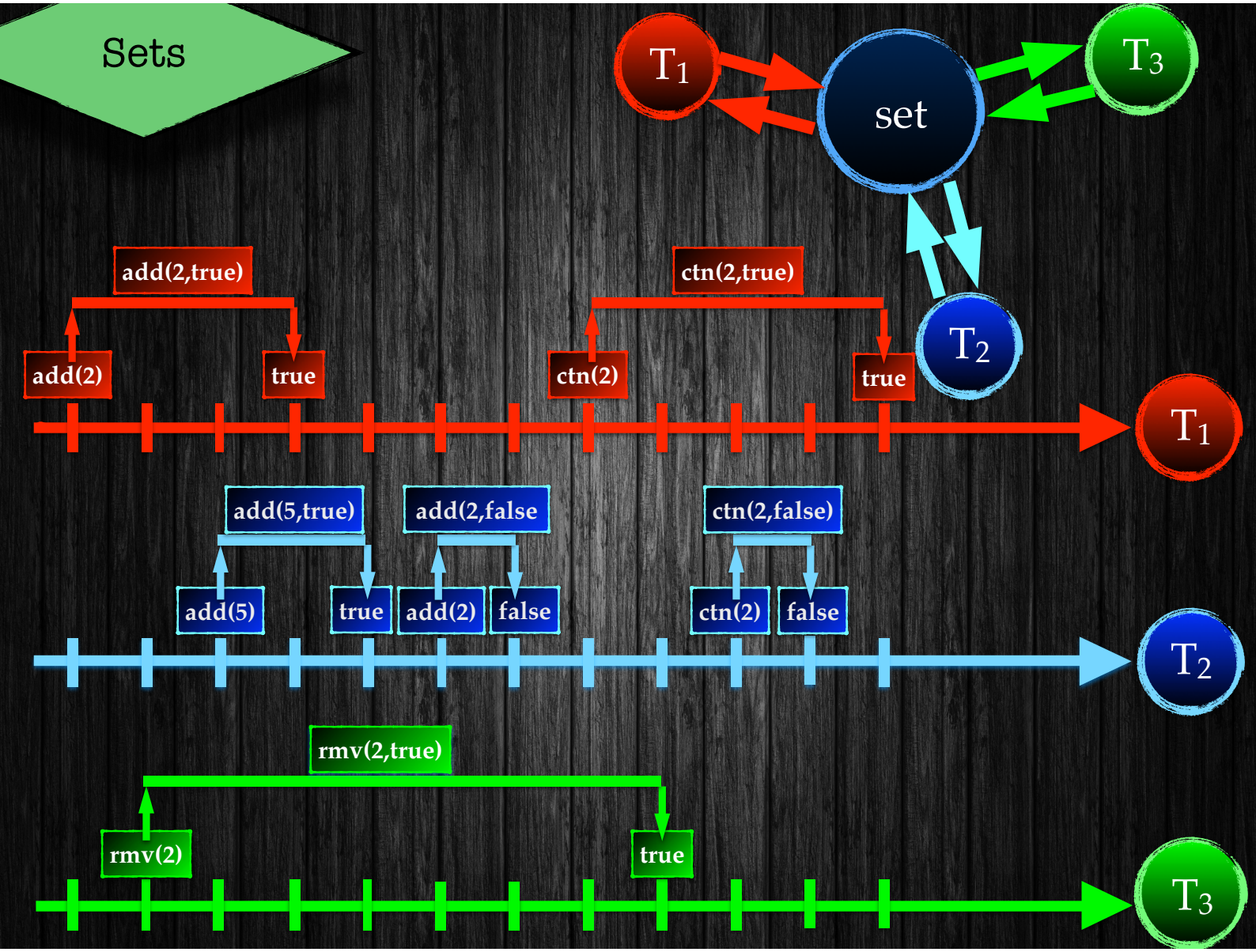

Sets
abstract data type
add(,false)
add(,true)
rmv(,true)
set
ctn(,false)
rmv(,false)
ctn(,true)
∈ ℕ
SetSpec
initially $S = \emptyset$
$S \xrightarrow{add(x,true)} S \cup \{x\}$ if $x \notin S$
$S \xrightarrow{add(x,false)} S$ if $x \in S$
$S \xrightarrow{rmv(x,true)} S - \{x\}$ if $x \in S$
$S \xrightarrow{rmv(x,false)} S$ if $x \notin S$
$S \xrightarrow{ctn(x,true)} S$ if $x \in S$
$S \xrightarrow{ctn(x,false)} S$ if $x \notin S$

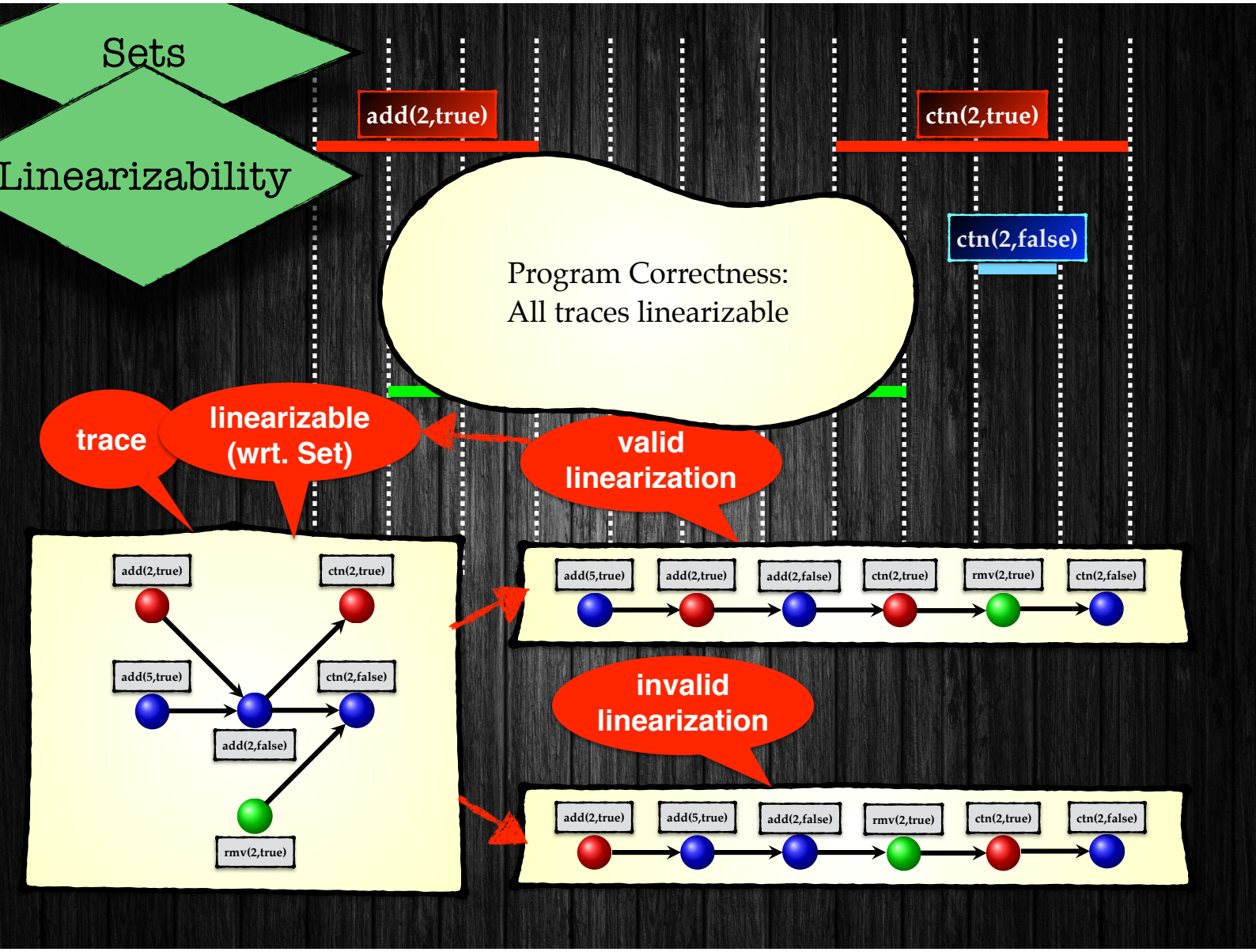

Sets
abstract data type
add(,false)
add(,true)
2
5
rmv(,true)
ctn(,false)
rmv(,false)
ctn(,true)
∈ ℕ
SetSpec
initially $S = \emptyset$
$S \xrightarrow{add(x,true)} S \cup \{x\}$ if $x \notin S$
$S \xrightarrow{add(x,false)} S$ if $x \in S$
$S \xrightarrow{rmv(x,true)} S - \{x\}$ if $x \in S$
$S \xrightarrow{rmv(x,false)} S$ if $x \notin S$
$S \xrightarrow{ctn(x,true)} S$ if $x \in S$
$S \xrightarrow{ctn(x,false)} S$ if $x \notin S$
trace
infinite-state
ctn(5,true)
add(5,false)
rmv(2,true)
ctn(2,false)
rmv(2,false)

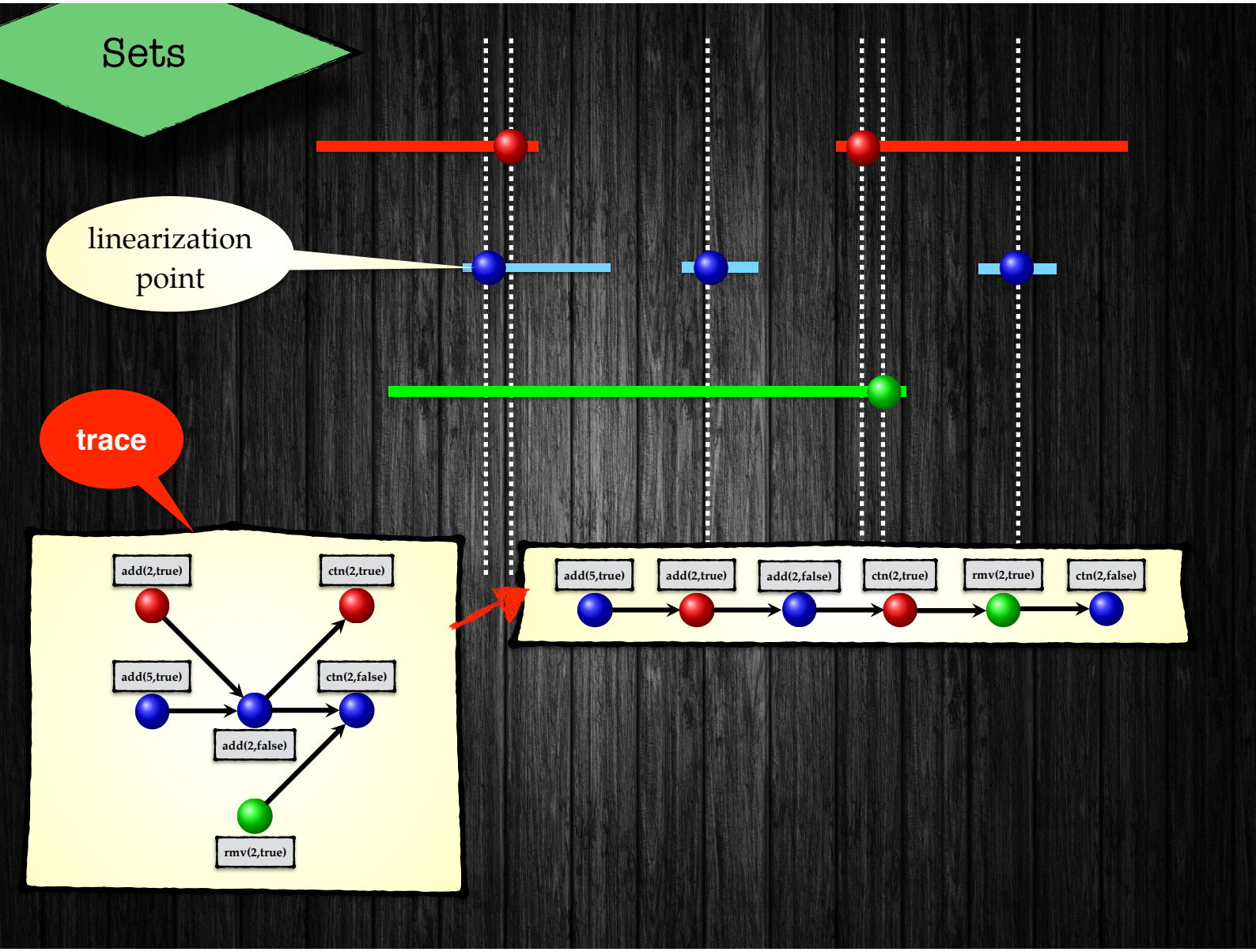

Set
Sequential Programs
abstract data type
add(,false)
add(,true)
rmv(,true)
1 3 6 4 2 5
ctn(,false)
∈ ℕ
rmv(,false)
ctn(,true)
SetProg
head
tail
−∞
1
3
4
5
∞
2
6
trace
add(2,true)
add(6,true)
ctn(5,true)
add(5,false)
rmv(4,true)
ctn(4,false)
rmv(4,false)


Sequential Programs
Program Correctness
SetProg
head
tail
−∞
1
3
4
5
∞
2
6
unbounded heap
unbounded data
SetSpec
initially $S = \emptyset$
$S \xrightarrow{add(x,true)} S \cup \{x\}$ if $x \notin S$
$S \xrightarrow{add(x,false)} S$ if $x \in S$
$S \xrightarrow{rmv(x,true)} S - \{x\}$ if $x \in S$
$S \xrightarrow{rmv(x,false)} S$ if $x \notin S$
$S \xrightarrow{ctn(x,true)} S$ if $x \in S$
$S \xrightarrow{ctn(x,false)} S$ if $x \notin S$
infinite-state
$traces(\text{ProgSpec}) \subseteq traces(\text{SetSpec})$


Sets
sequential setting
P
ctn(2,false)
set
T1
add(2)
true
set
rmv(3)
false
T3
concurrent setting
ctn(2)
false
T2
unbounded parallelism


Sets
T1
set
T3
T2
add(2,true)
add(2)
true
ctn(2,true)
ctn(2)
true
T1
add(5,true)
add(5)
true
add(2,false
add(2)
false
ctn(2,false)
ctn(2)
false
T2
rmv(2,true)
rmv(2)
true
T3


Sets
Linearizability
add(2,true)
ctn(2,true)
ctn(2,false)
Program Correctness:
All traces linearizable
trace
linearizable
(wrt. Set)
valid
linearization
add(2,true)
ctn(2,true)
add(5,true)
ctn(2,false)
add(2,false)
rmv(2,true)
add(5,true)
add(2,true)
add(2,false)
ctn(2,true)
rmv(2,true)
ctn(2,false)
invalid
linearization
add(2,true)
add(5,true)
add(2,false)
rmv(2,true)
ctn(2,true)
ctn(2,false)


Sets
linearization point
trace
add(2,true)
ctn(2,true)
add(5,true)
ctn(2,false)
add(2,false)
rmv(2,true)
add(5,true)
add(2,true)
add(2,false)
ctn(2,true)
rmv(2,true)
ctn(2,false)

Roadm

Goal

- Automatic Verification of Linearizability

Challenges

- Specifying data structures
- Specifying Linearisation Points
- Unbounded Data Domains
- Unbounded Parallelism
- Unbounded Heaps

Observers

Controllers

Data Abstraction

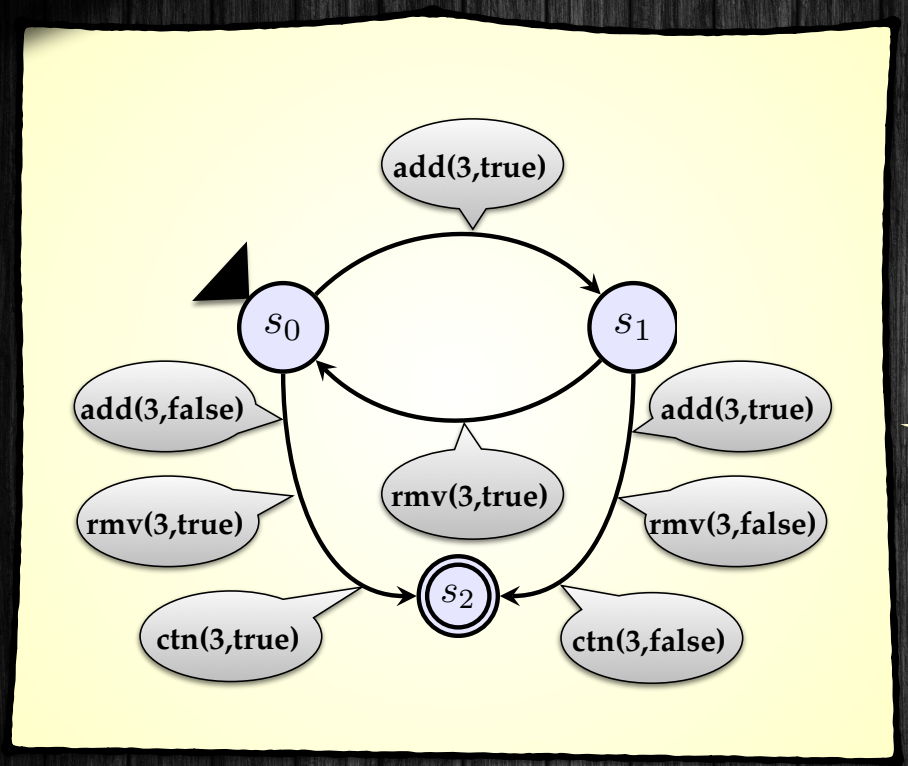
Thread Abstraction

Heap Abstraction

Experiments

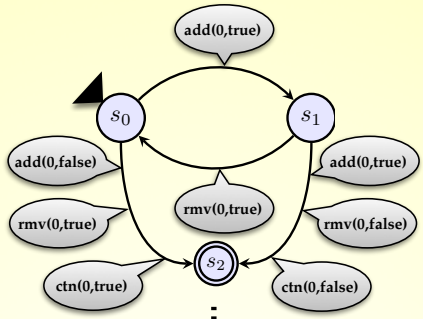
Algorithms	*Time (s)*	*Algorithms*	*Time (s)*
MS two-lock queue [24]	0.02	Treiber stack [31]	0.18
MS lock-free queue [24]	1.87	HSY stack [17]	39.85
Elimination queue [26] ✓	325.56	DGLM queue [11]	6.99
Vechev-CAS set [38] ✓	72.01	HW queue [20]	15.21
Vechev-DCAS set [38]	1.67	Optimistic set [19]	10.43
Harris lock-free set [13]	13.17	Lazy set [19] ✓	49.23
Michael lock-free set [25]	17.37		55.01
Pessimistic set [19]	7.91	O′Hearn set [27] ✓	65.01
Unordered set [40] ✓	80.01		65.16
HM lock-free set [19] ✓	78.22	CCAS [14] ✓	0.04
	79.01	RDCSS [14] ✓	0.26

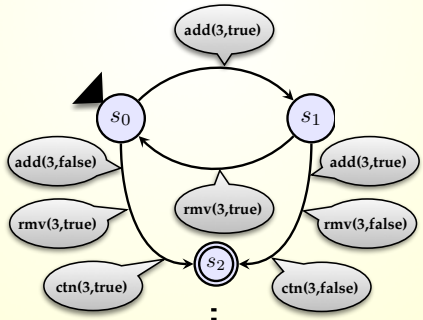
Observers



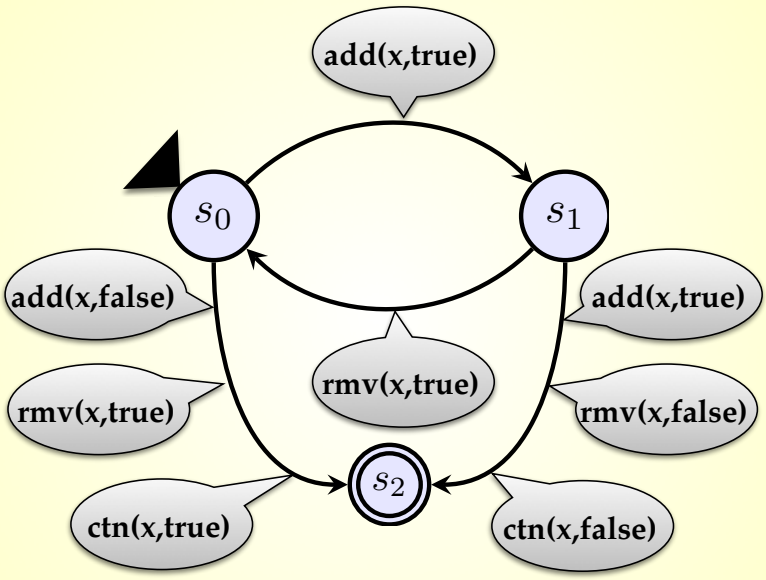
violation
of set behaviour
due to “3”

Observers

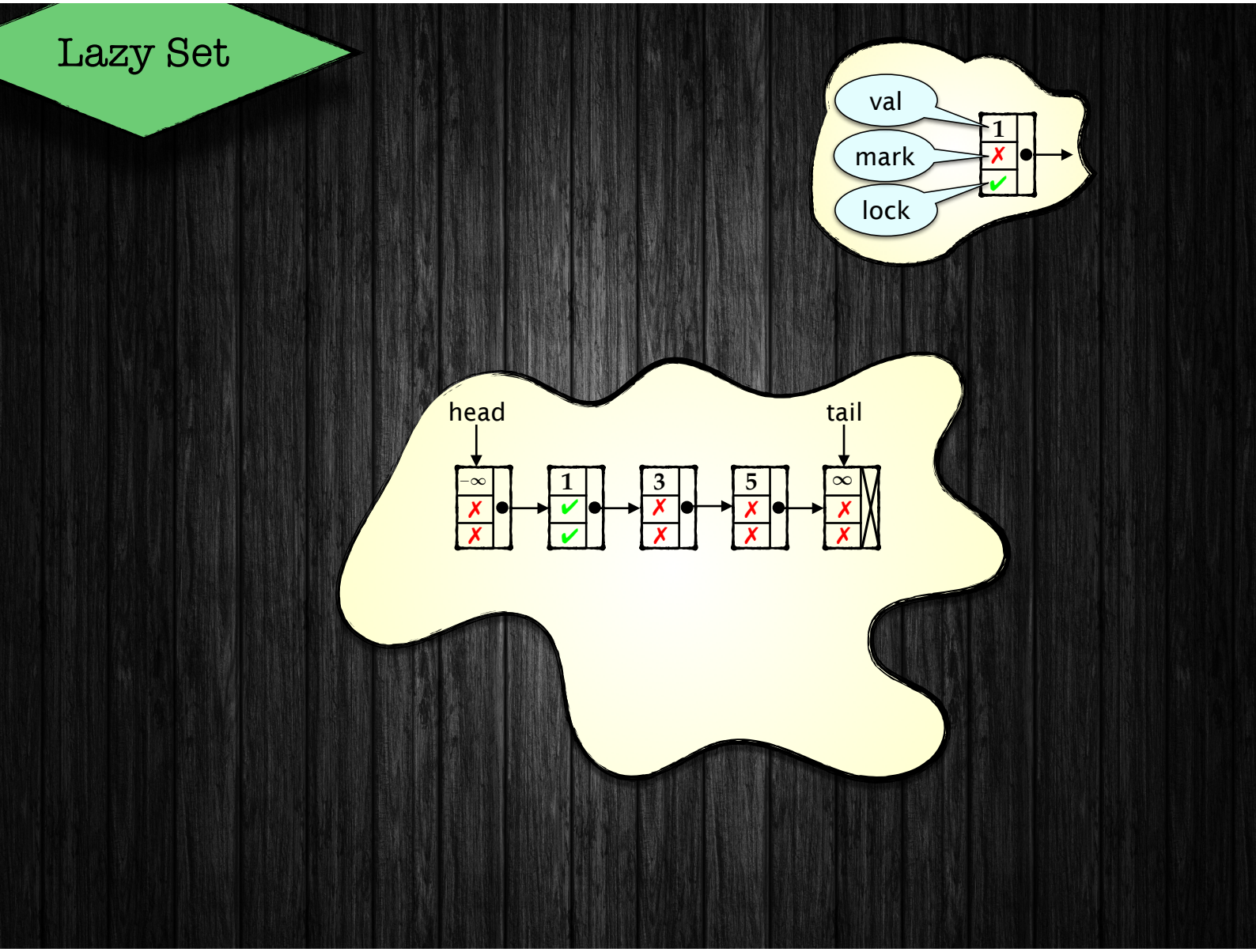




SET observer

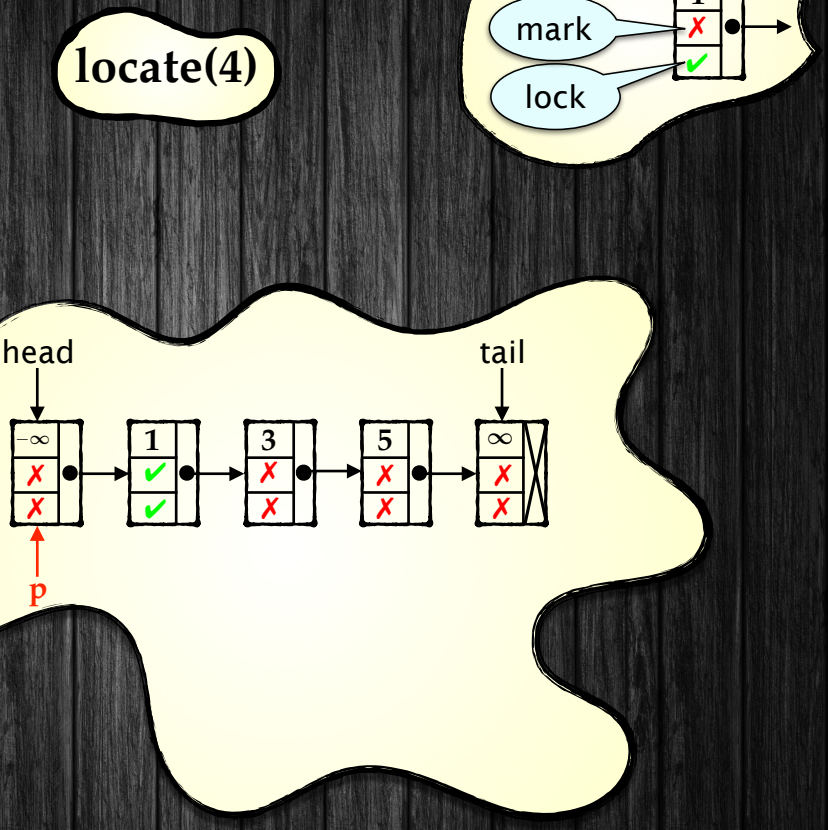


Lazy Set



Lazy Set

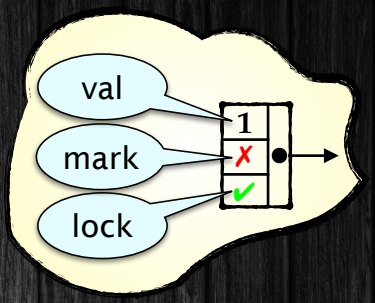
```
locate(e):
local p, c
1  while (true)
2    p := head;
3    c := p.next;
4    while (c.val < e)
5      p := c;
6      c := c.next
7    lock(p); lock(c);
8    if (!p.mark && !c.mark && p.next=c)
9      return(p,c);
10   else
11     unlock(p);
12     unlock(c);
```

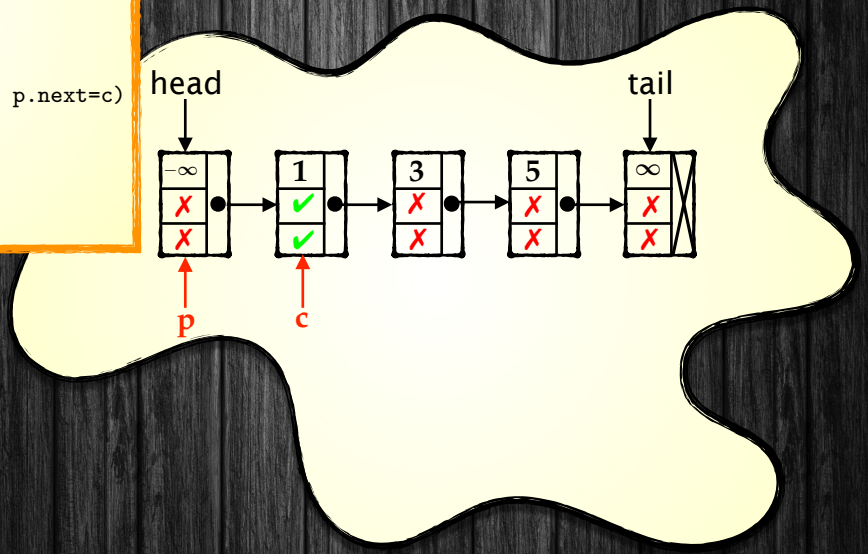


Lazy Set

```
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)

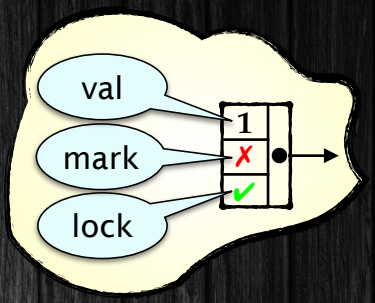


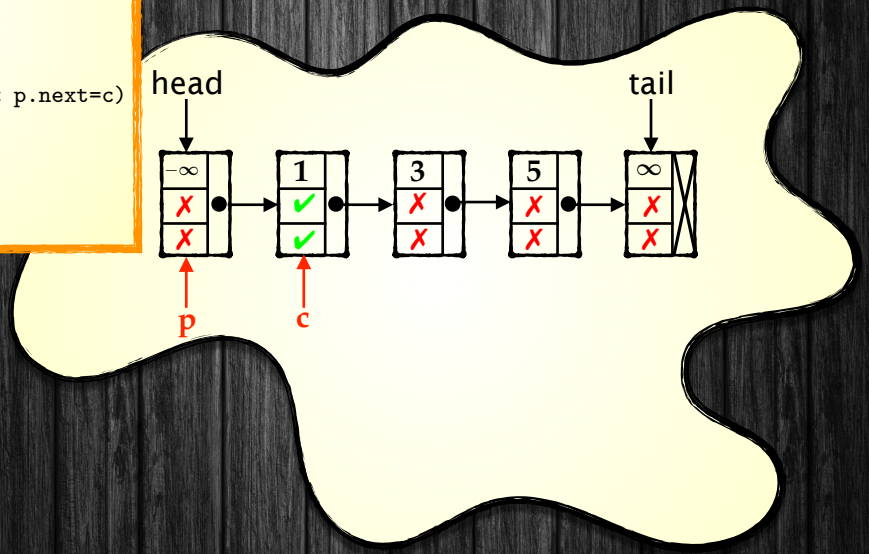


Lazy Set

```
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);
```

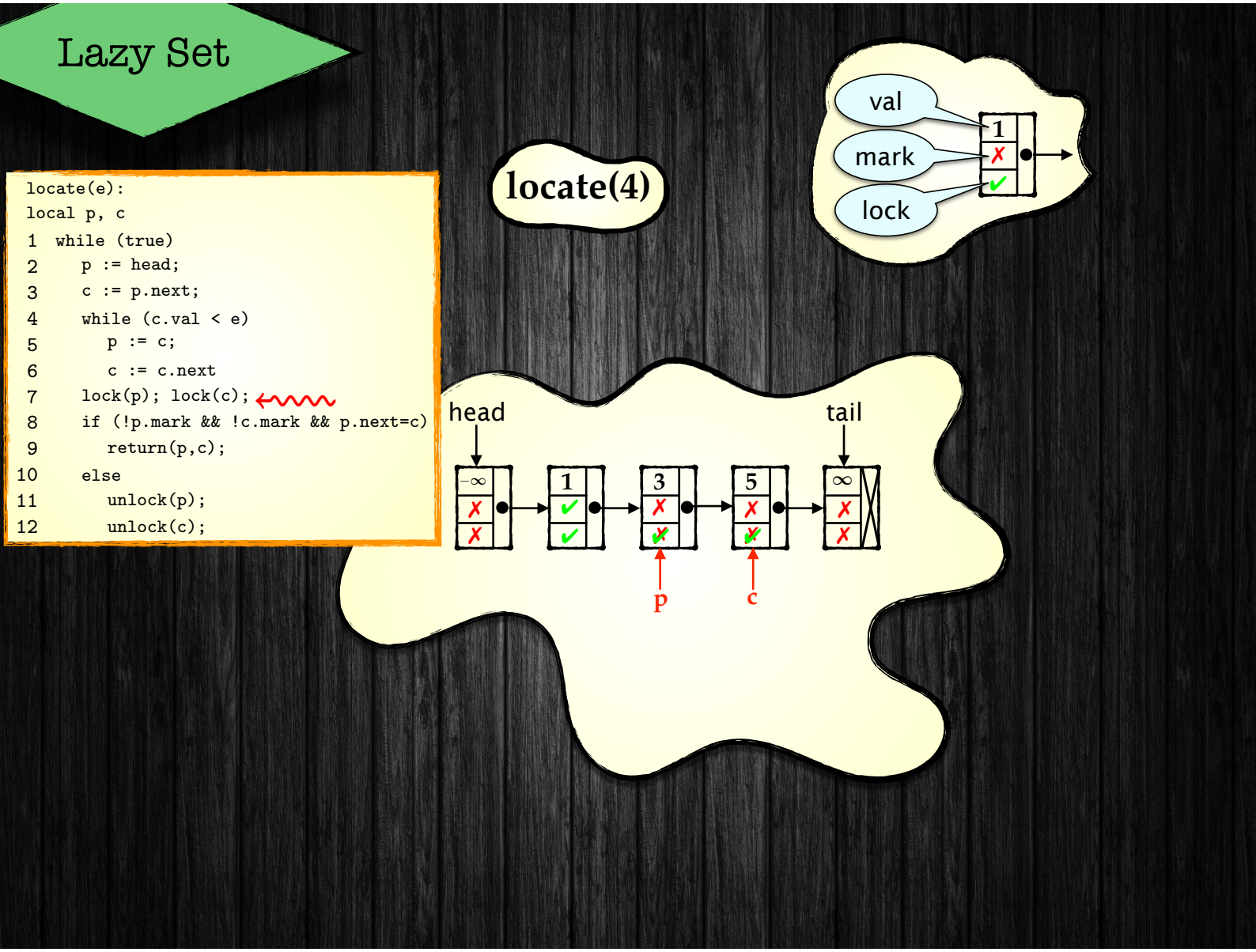
locate(4)





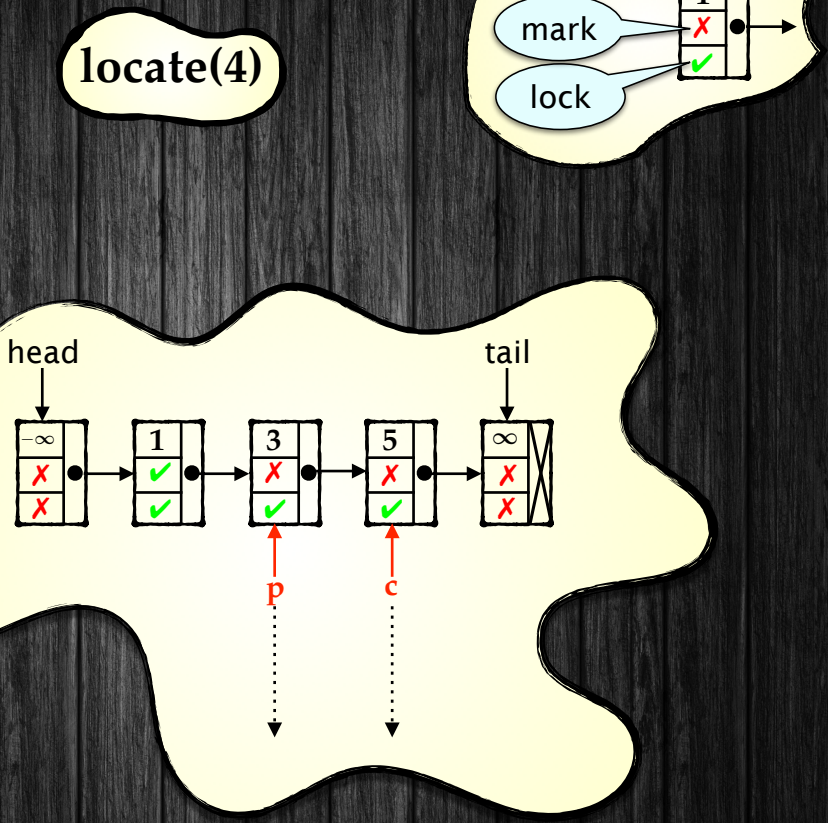
Lazy Set

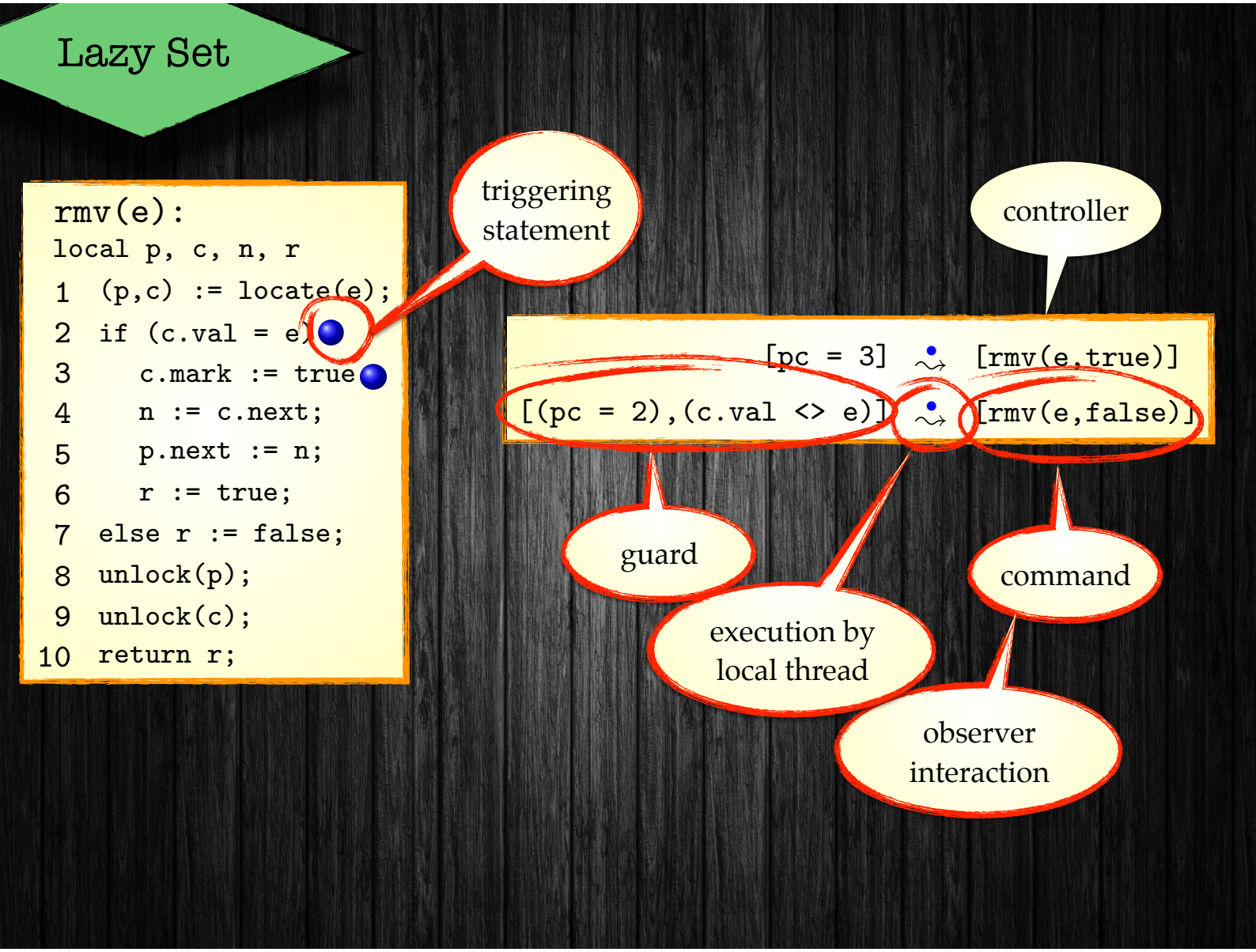
```
locate(e):
local p, c
1  while (true)
2     p := head;
3     c := p.next;
4     while (c.val < e)
5        p := c;
6        c := c.next
7     lock(p); lock(c);
8     if (!p.mark && !c.mark && p.next=c)
9        return(p,c);
10    else
11       unlock(p);
12       unlock(c);
```



Lazy Set

```
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);
```




Lazy Set
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;
triggering statement
controller
[pc = 3] [rmv(e,true)]
[(pc = 2),(c.val <> e)] [rmv(e,false)]
guard
execution by local thread
command
observer interaction

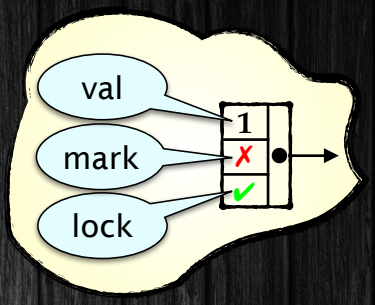
Lazy Set

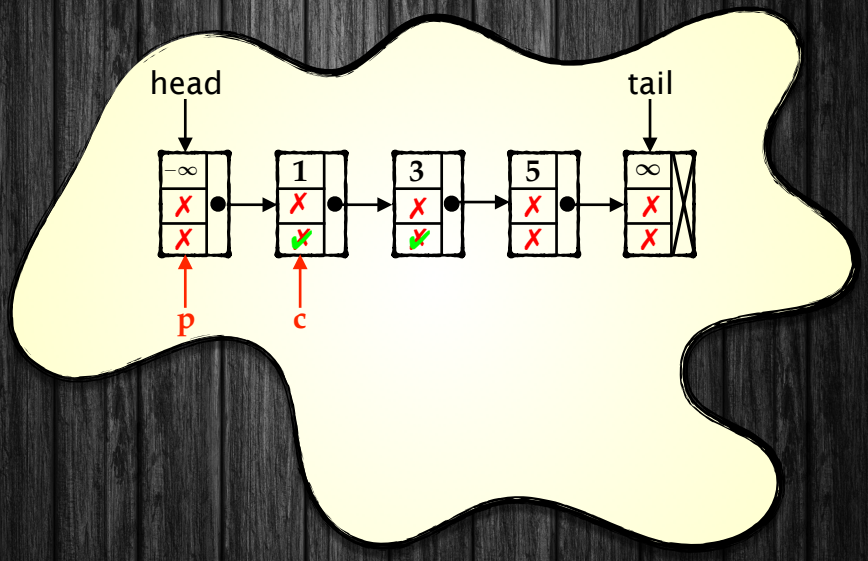
rmv(3)

```
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;
```

[pc = 3] ⇝ [rmv(e,true)]

[(pc = 2),(c.val <> e)] ⇝ [rmv(e,false)]



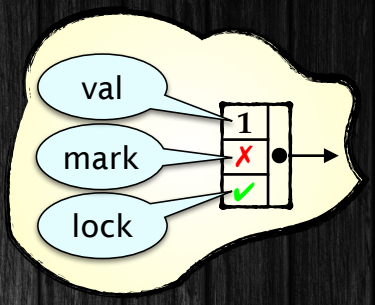


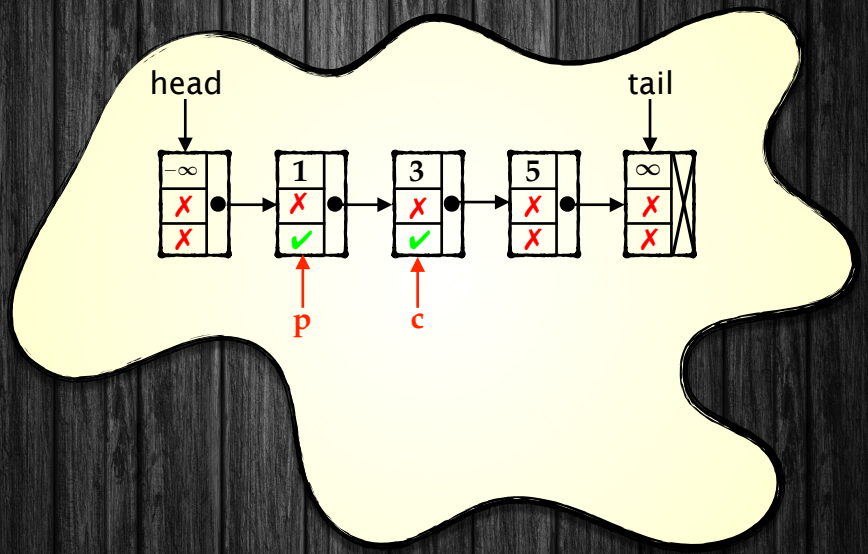
Lazy Set

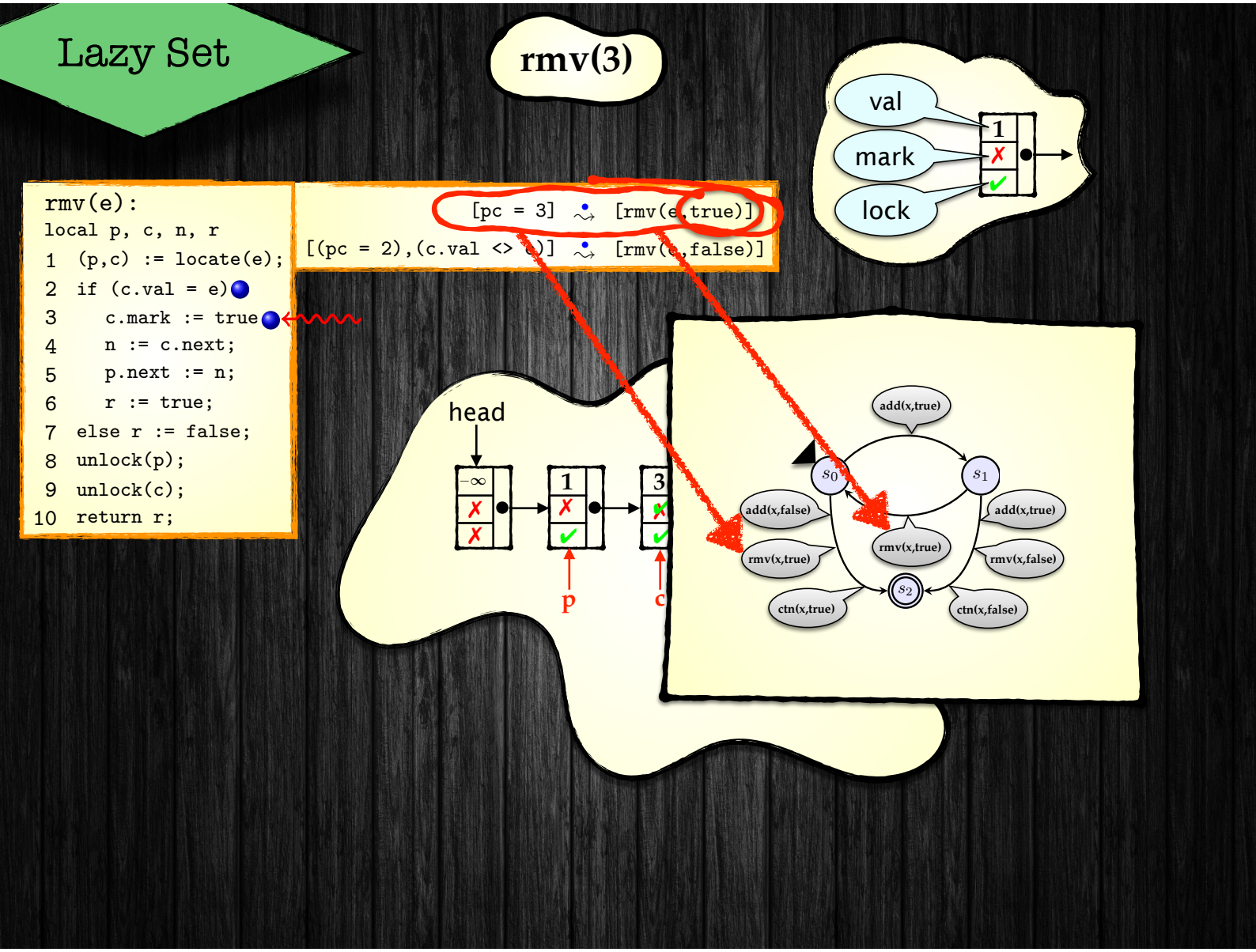
rmv(3)

```
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;
```

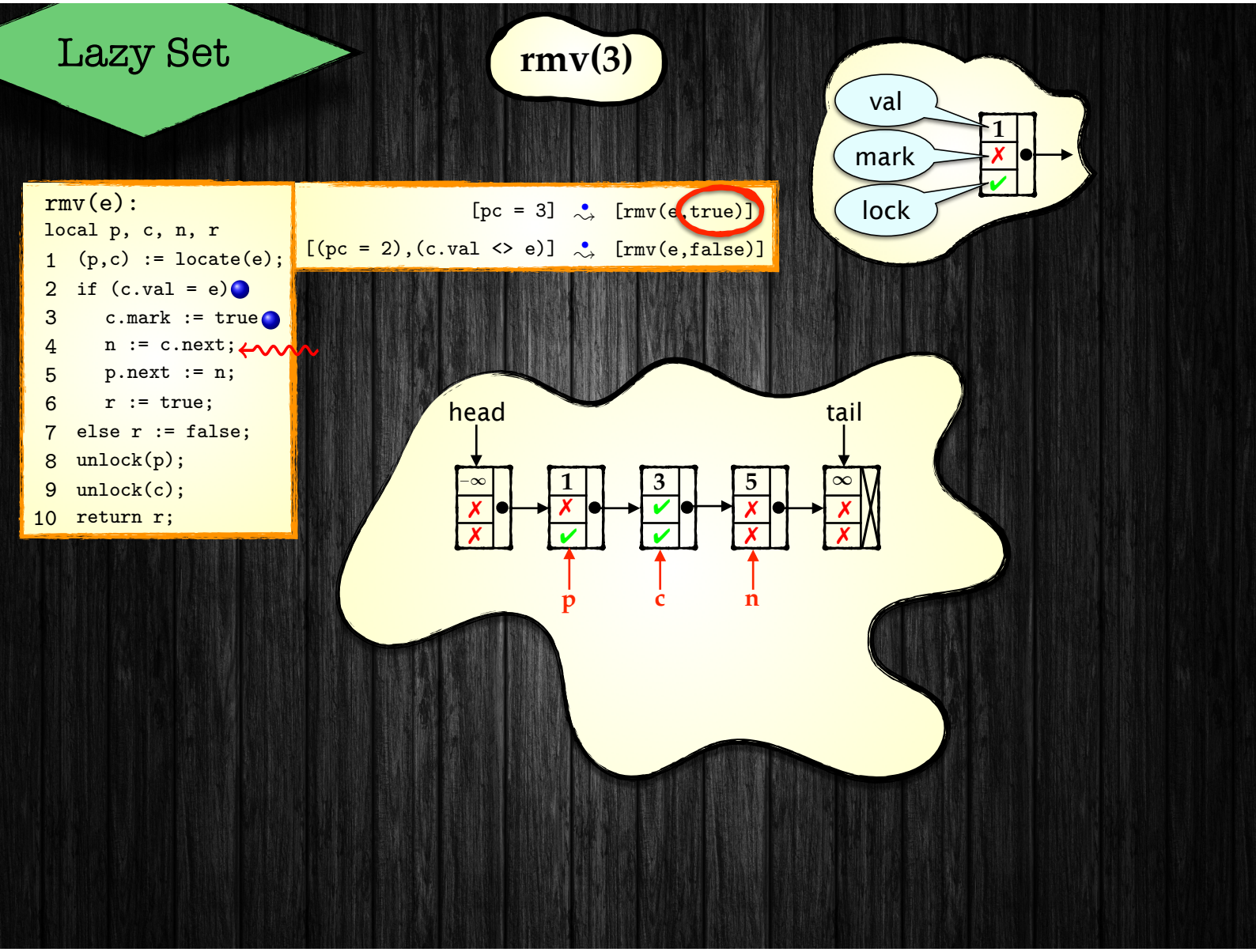
```
[pc = 3]  ⇝  [rmv(e,true)]
[(pc = 2),(c.val <> e)]  ⇝  [rmv(e,false)]
```






Lazy Set
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;
[pc = 3] [rmv(e,true)]
[(pc = 2),(c.val <> e)] [rmv(e,false)]
val
mark
lock
1
head
−∞
1
3
p
c
add(x,true)
add(x,false)
rmv(x,true)
rmv(x,false)
ctn(x,true)
ctn(x,false)
s_0
s_1
s_2

Lazy Set


rmv(3)
val
mark
lock
1
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;
[pc = 3] [rmv(e,true)]
[(pc = 2),(c.val <> e)] [rmv(e,false)]
head
tail
−∞
1
3
5
∞
p
c
n

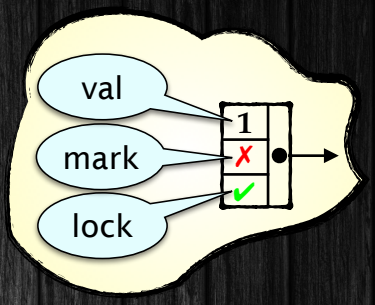
Lazy Set

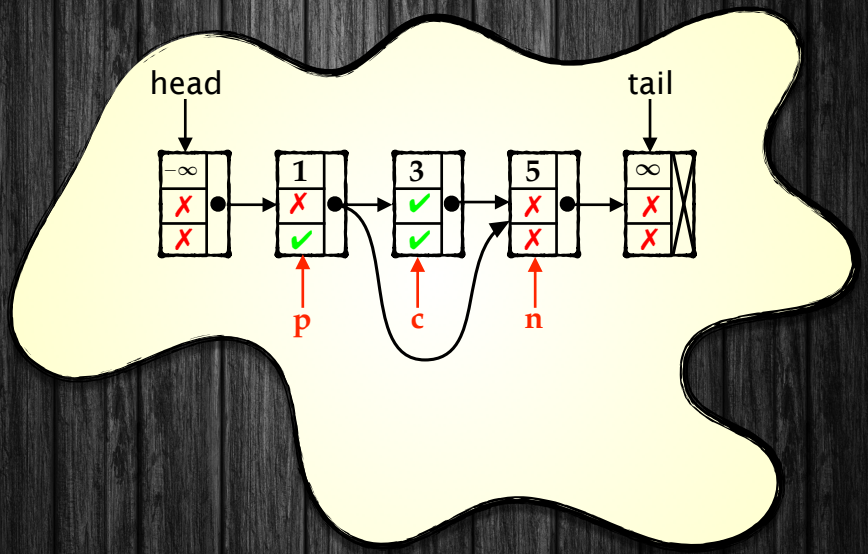
rmv(3)

```
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;
```

[pc = 3] ⇝ [rmv(e,true)]

[(pc = 2),(c.val <> e)] ⇝ [rmv(e,false)]





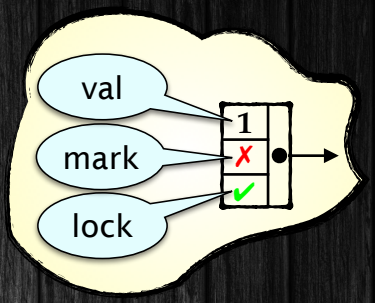
Lazy Set

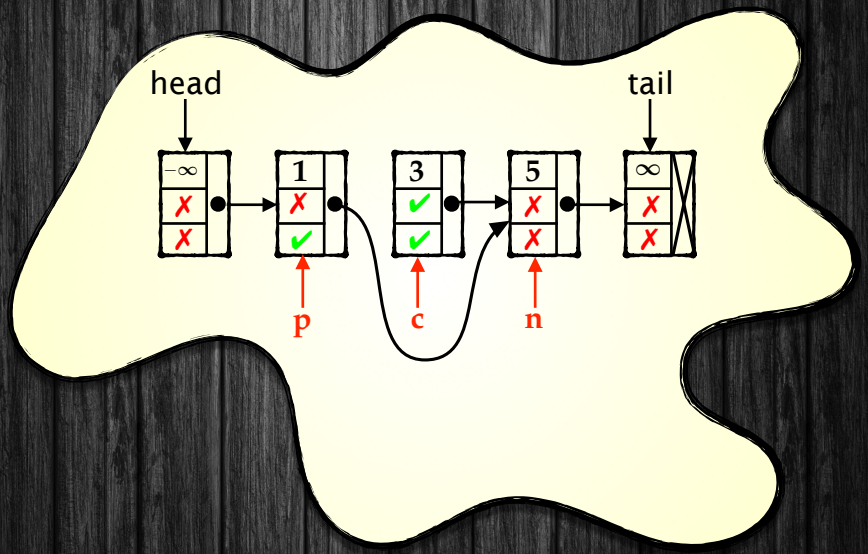
rmv(3)

```
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;
```

[pc = 3] ⇝ [rmv(e,true)]

[(pc = 2),(c.val <> e)] ⇝ [rmv(e,false)]





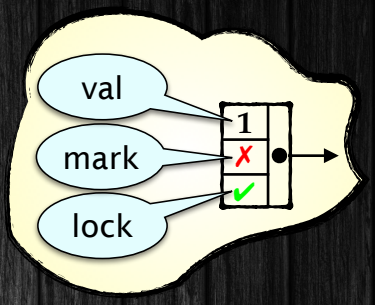
Lazy Set

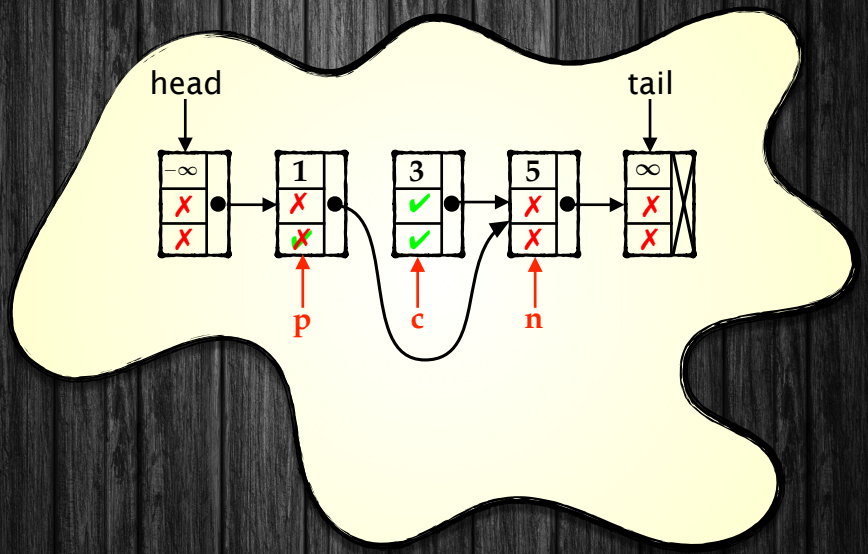
rmv(3)

```
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;
```

[pc = 3] ⇝ [rmv(e,true)]

[(pc = 2),(c.val <> e)] ⇝ [rmv(e,false)]





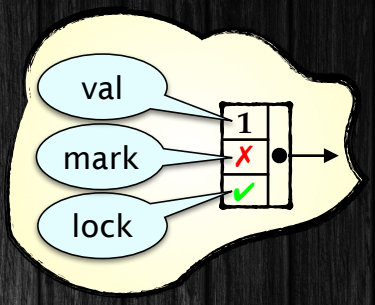
Lazy Set

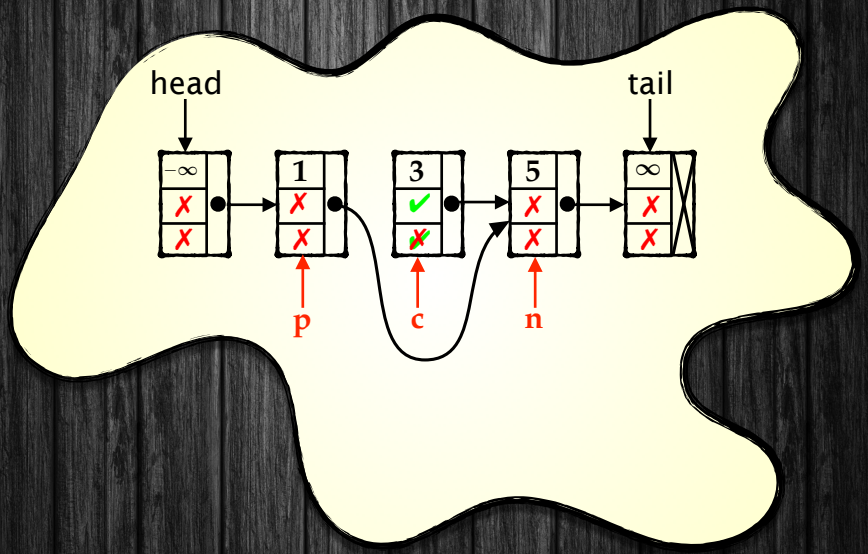
rmv(3)

```
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;
```

[pc = 3] ⇝ [rmv(e,true)]

[(pc = 2),(c.val <> e)] ⇝ [rmv(e,false)]



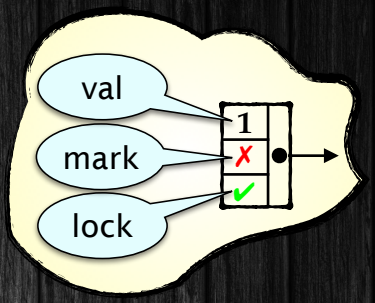


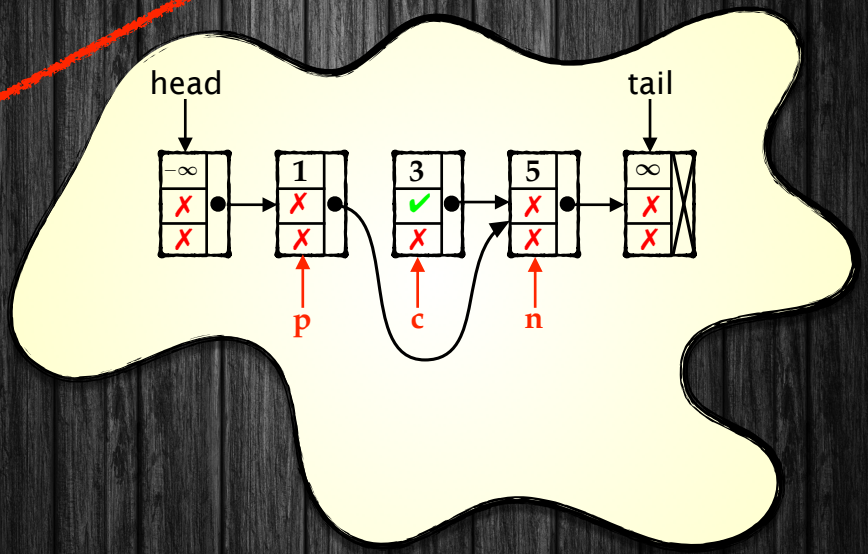
Lazy Set

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;
```

```
[pc = 3]                  [rmv(e,true)]
[(pc = 2),(c.val <> e)]   [rmv(e,false)]
```





Lazy Set

```
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;
```

observer interaction

[pc = 4] [add(e,true)|!add(e)]

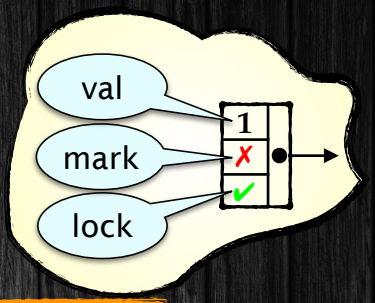
[(pc = 2),(c.val = e)] [add(e,false)]

broadcast: helping

Lazy Set

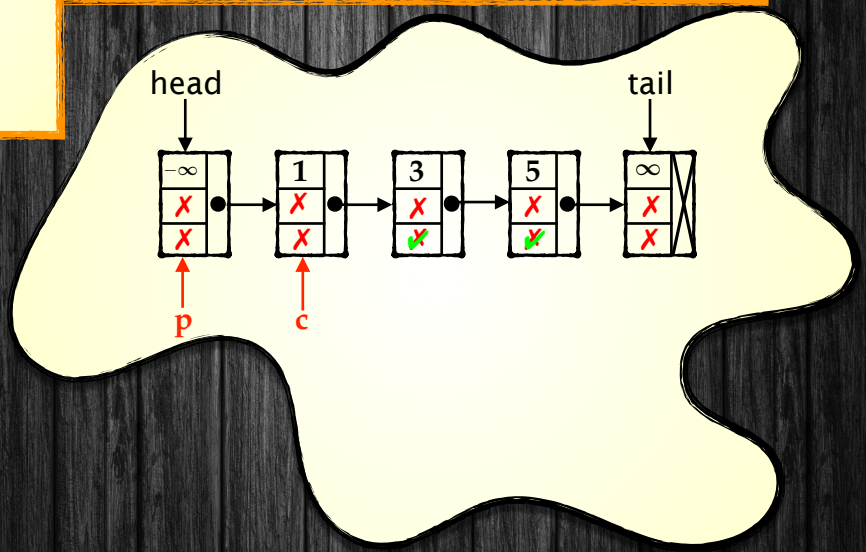
add(4)

```
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;
```

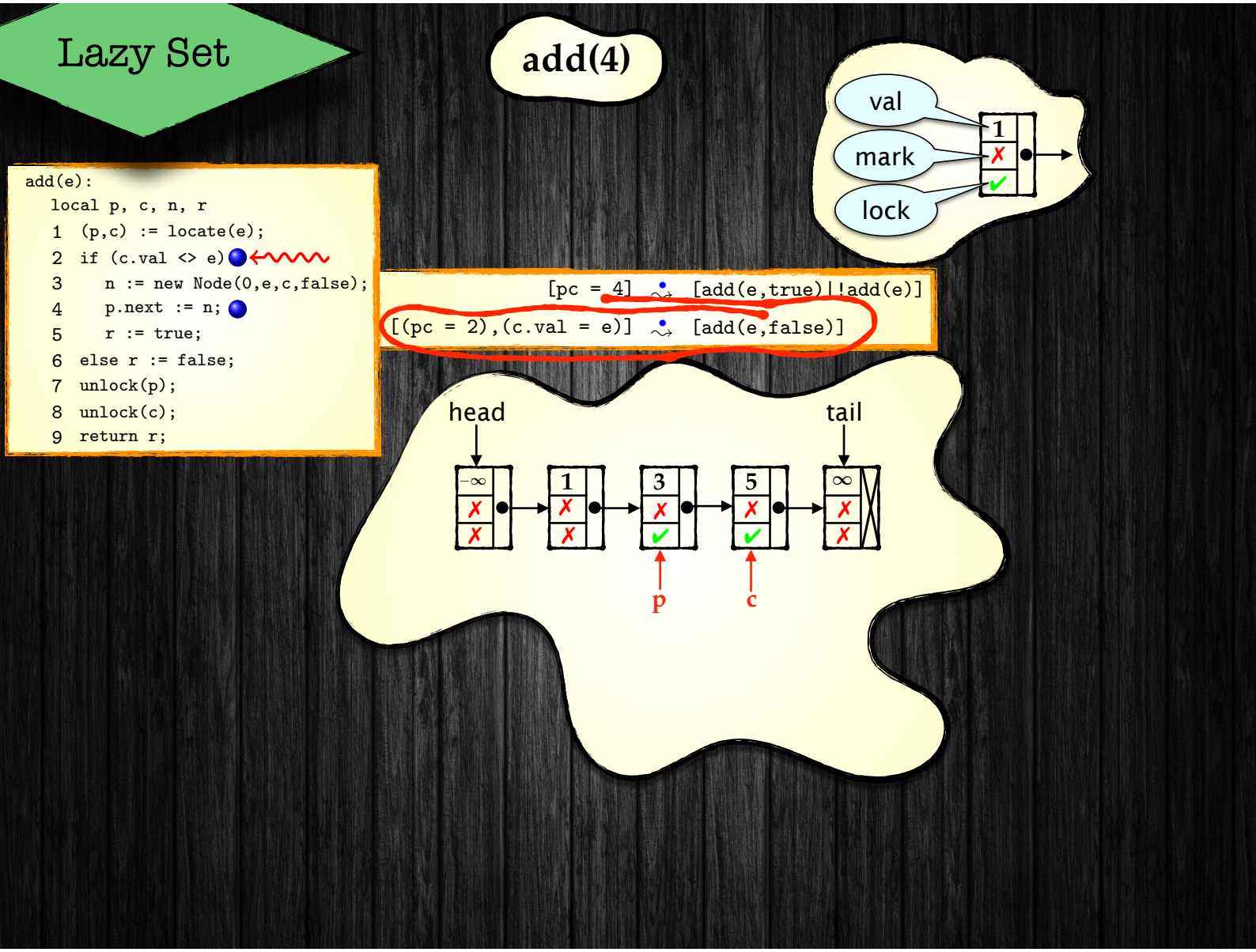


[pc = 4] ⇝ [add(e,true)|!add(e)]

[(pc = 2),(c.val = e)] ⇝ [add(e,false)]



Lazy Set


add(4)
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;
val
mark
lock
1
[pc = 4] [add(e,true)|!add(e)]
[(pc = 2),(c.val = e)] [add(e,false)]
head
tail
−∞
1
3
5
∞
p
c

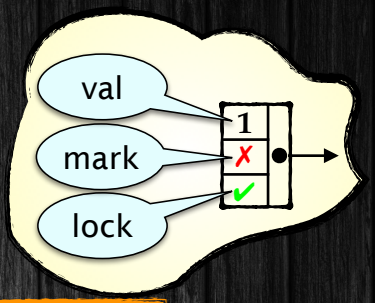
Lazy Set

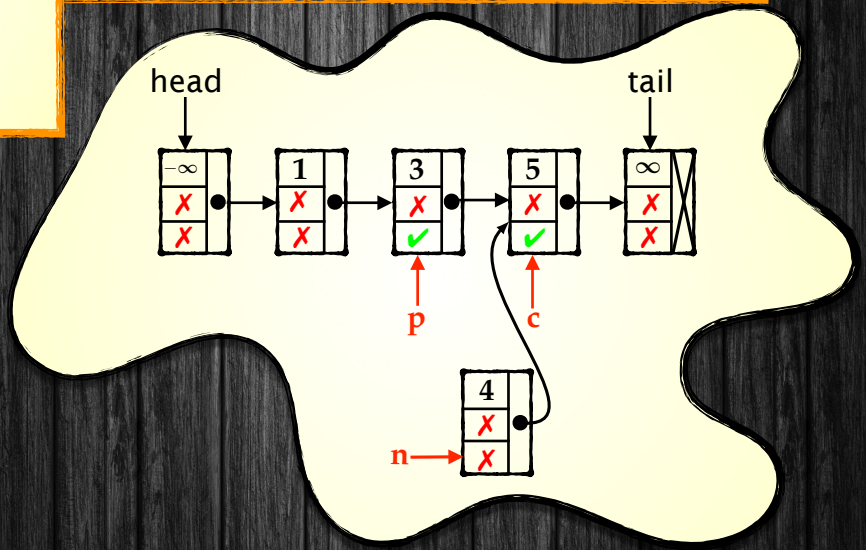
add(4)

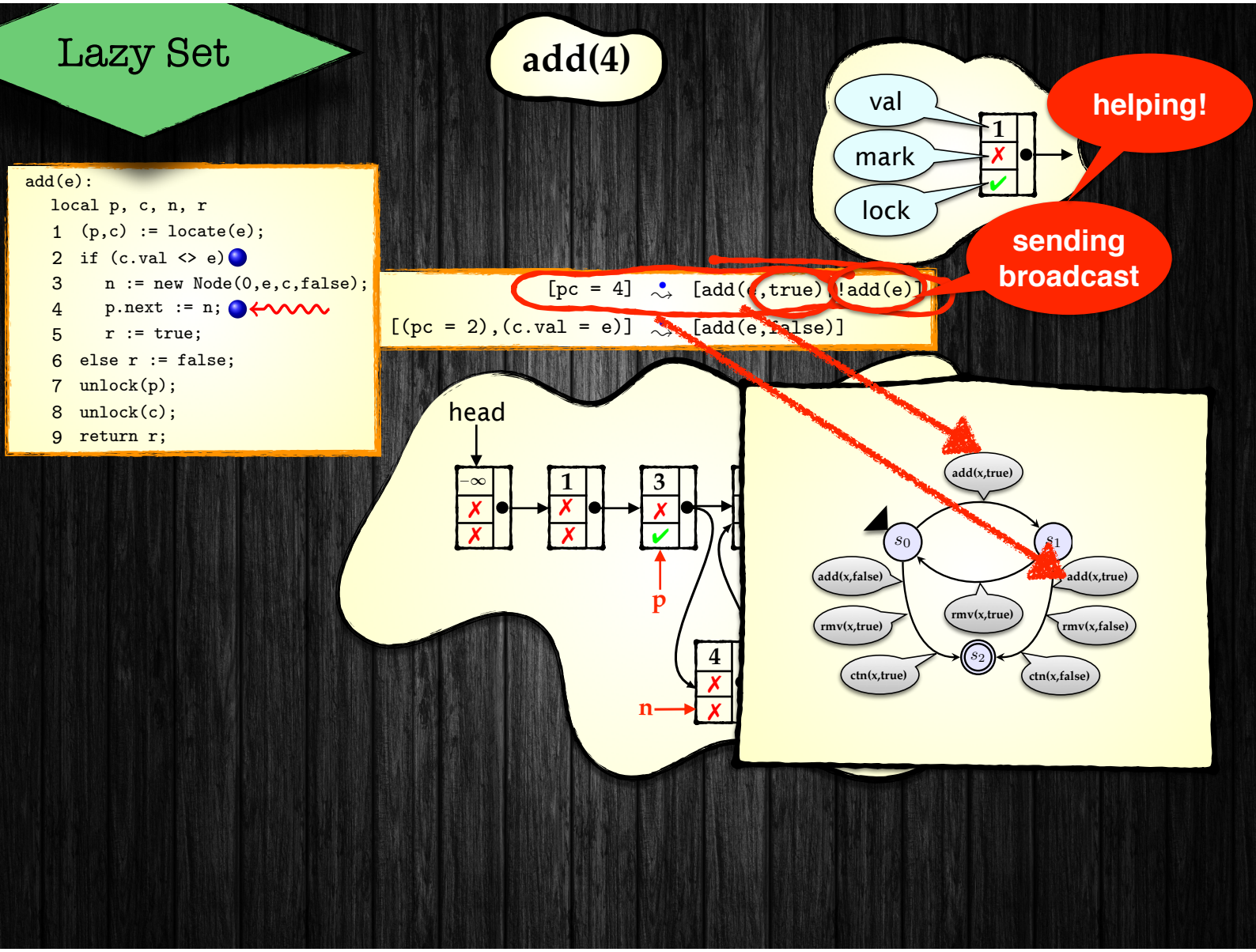
```
add(e):
  local p, c, n, r
1 (p,c) := locate(e);
2 if (c.val <> e)
3   n := new Node(0,e,c,false);
4   p.next := n;
5   r := true;
6 else r := false;
7 unlock(p);
8 unlock(c);
9 return r;
```

[pc = 4] ⇝ [add(e,true)|!add(e)]

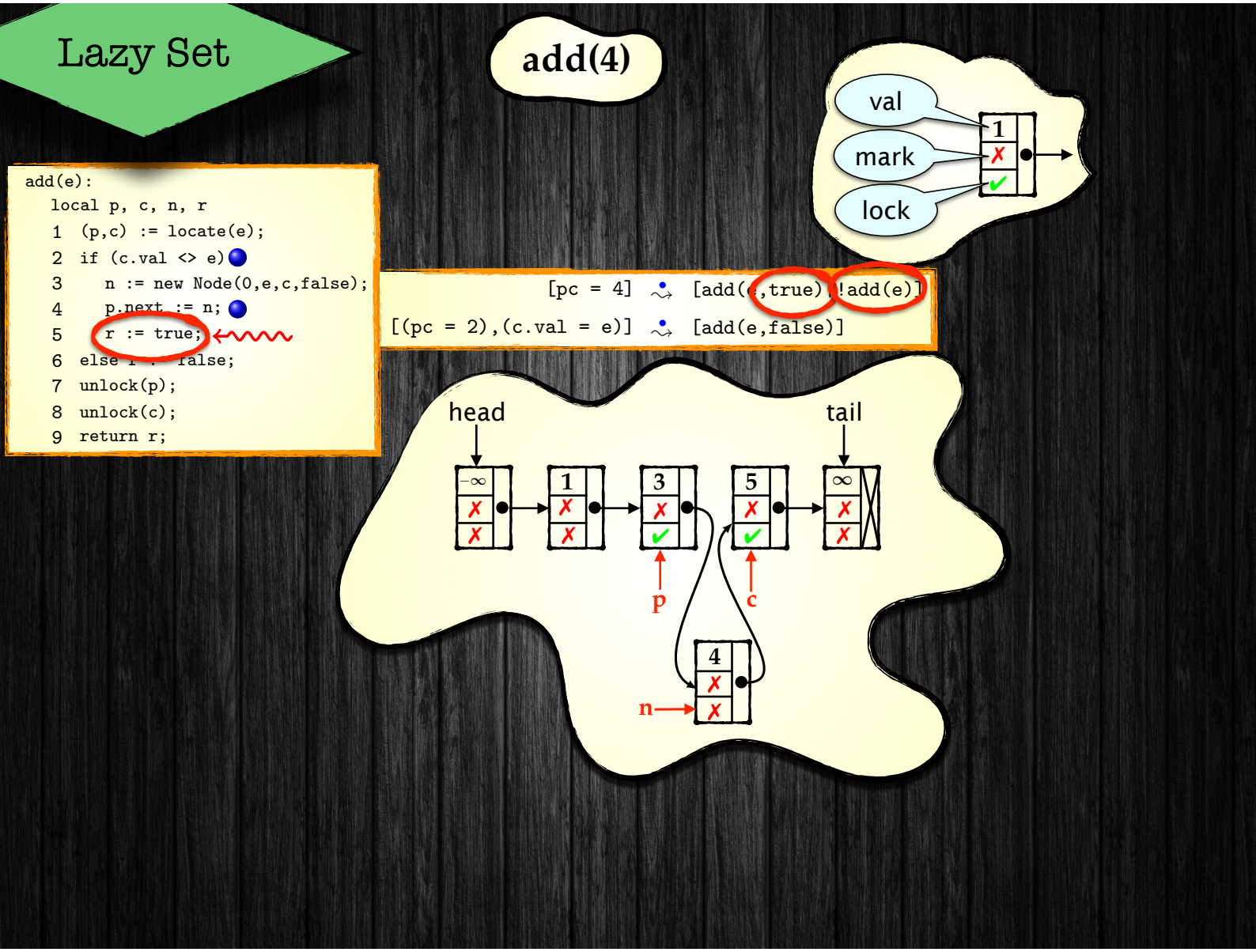
[(pc = 2),(c.val = e)] ⇝ [add(e,false)]






Lazy Set
add(4)
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;
val
mark
lock
1
helping!
sending broadcast
[pc = 4]
[add(e,true) !add(e)]
[(pc = 2),(c.val = e)]
[add(e,false)]
head
−∞
1
3
4
p
n
add(x,true)
add(x,false)
rmv(x,true)
rmv(x,false)
ctn(x,true)
ctn(x,false)
s_0
s_1
s_2

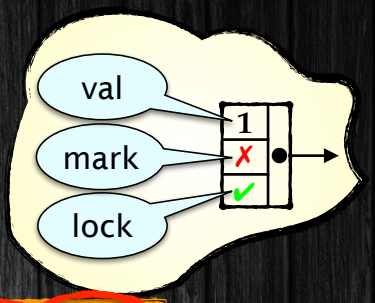
Lazy Set


add(4)
val
mark
lock
1
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;
[pc = 4] [add(e,true) !add(e)]
[(pc = 2),(c.val = e)] [add(e,false)]
head
tail
−∞
1
3
5
∞
4
p
c
n

Lazy Set

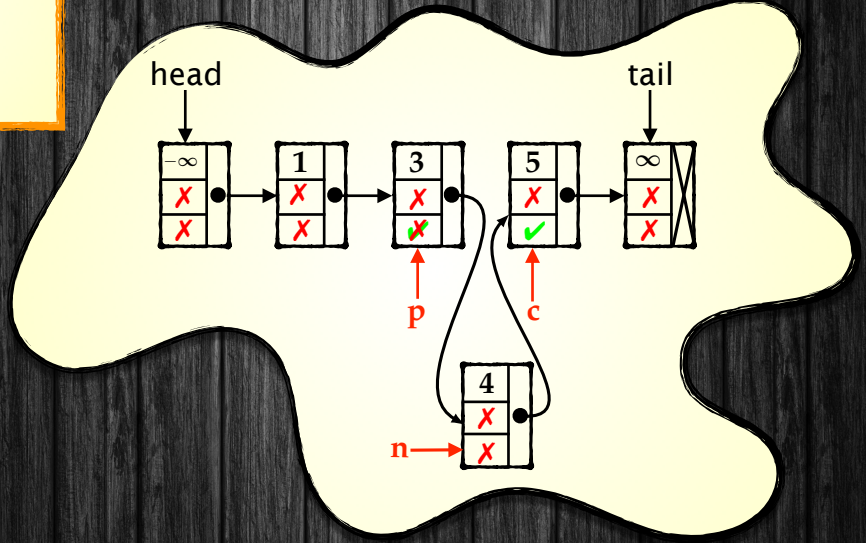
add(4)

```
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;
```



[pc = 4] ⇝ [add(e,true), !add(e)]

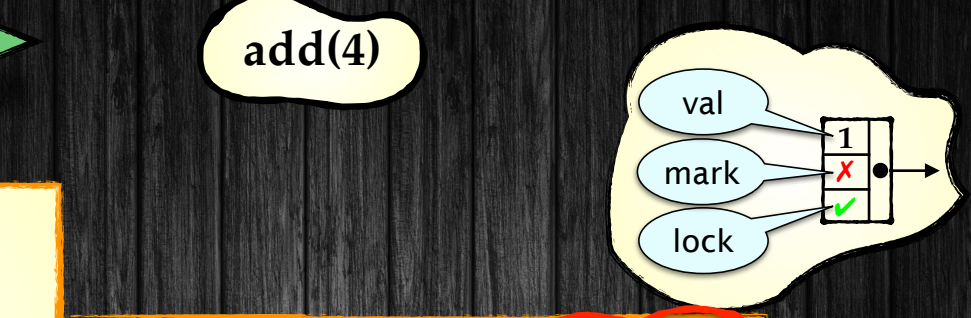
[(pc = 2),(c.val = e)] ⇝ [add(e,false)]



Lazy Set

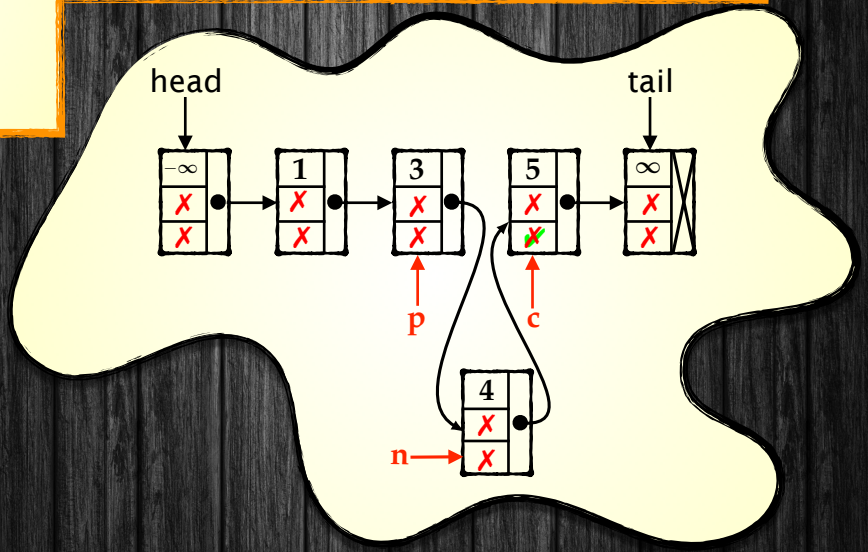
```
add(e):
   local p, c, n, r
1  (p,c) := locate(e);
2  if (c.val <> e)
3     n := new Node(0,e,c,false);
4     p.next := n;
5     r := true;
6  else r := false;
7  unlock(p);
8  unlock(c);
9  return r;
```

add(4)

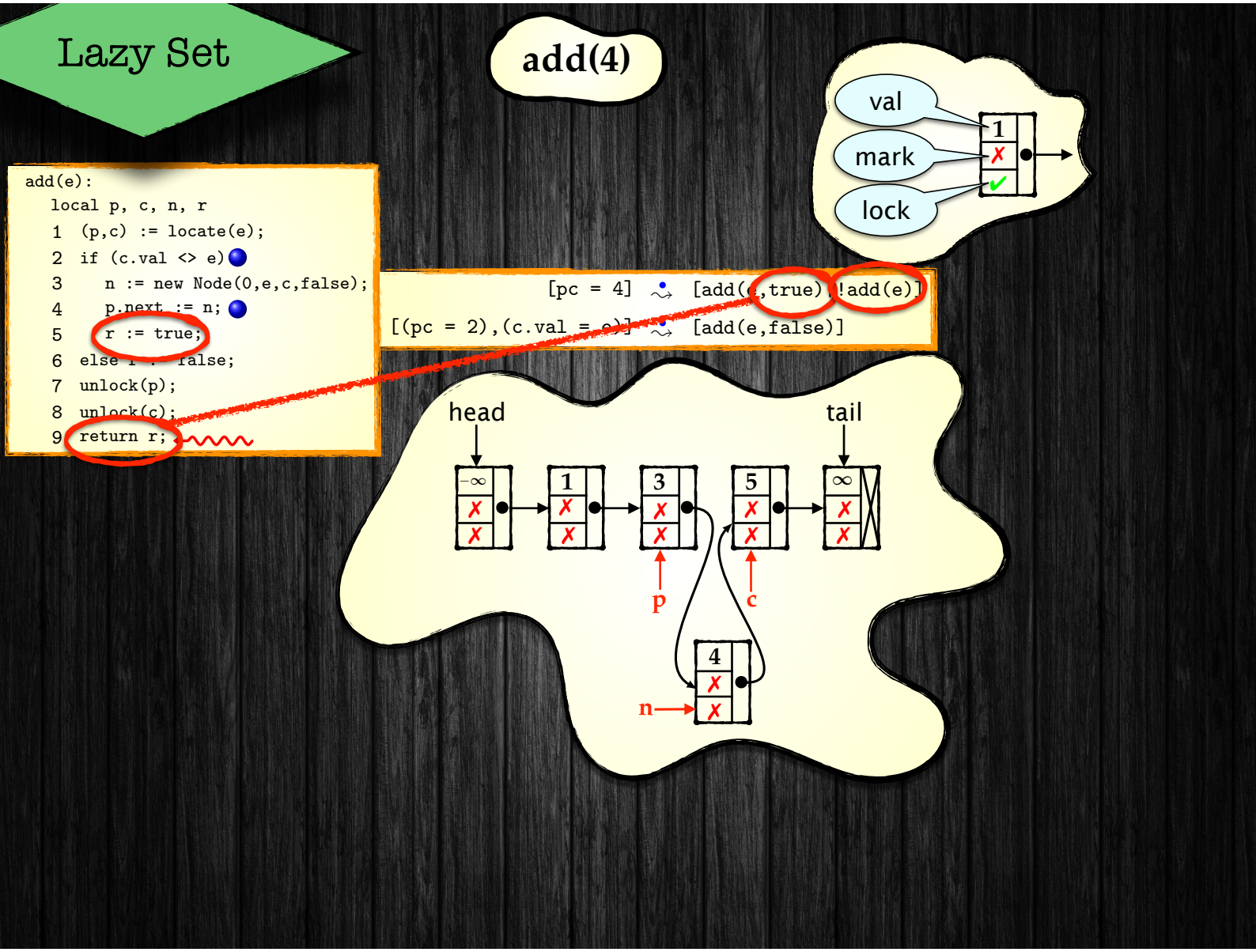


[pc = 4] ⇝ [add(e,true) ∨ !add(e)]

[(pc = 2),(c.val = e)] ⇝ [add(e,false)]

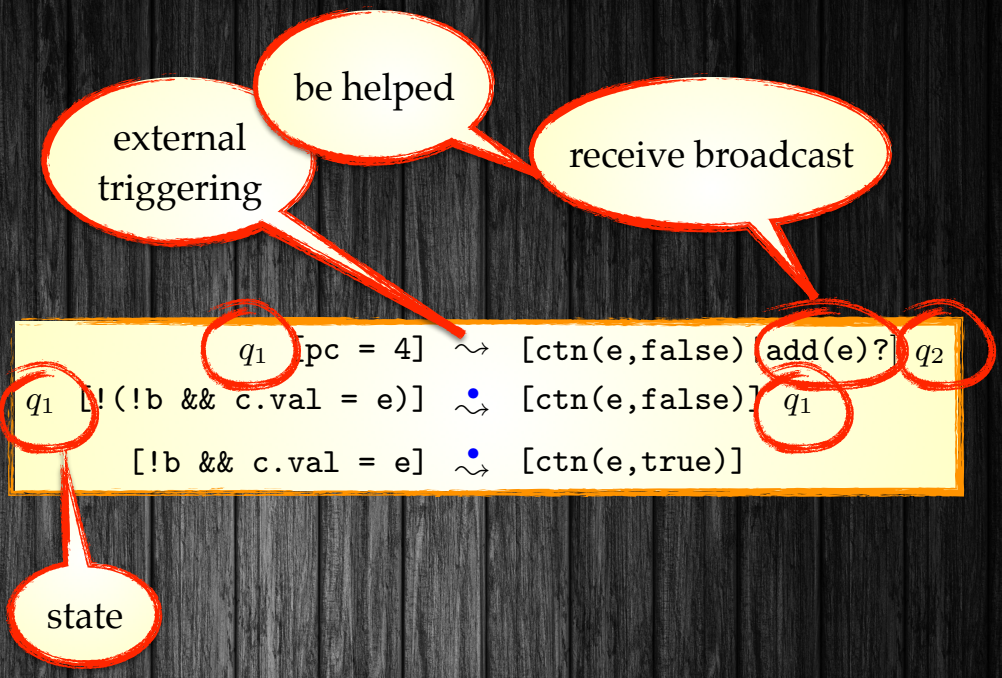


Lazy Set


add(4)
val
mark
lock
1
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;
[pc = 4] [add(e,true) !add(e)]
[(pc = 2),(c.val = e)] [add(e,false)]
head
tail
−∞
1
3
5
∞
p
c
4
n

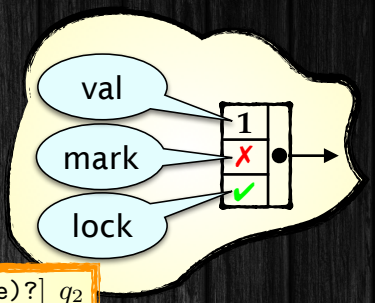
Lazy Set

```
ctn(e):
local c
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;
```



Lazy Set

ctn(3)

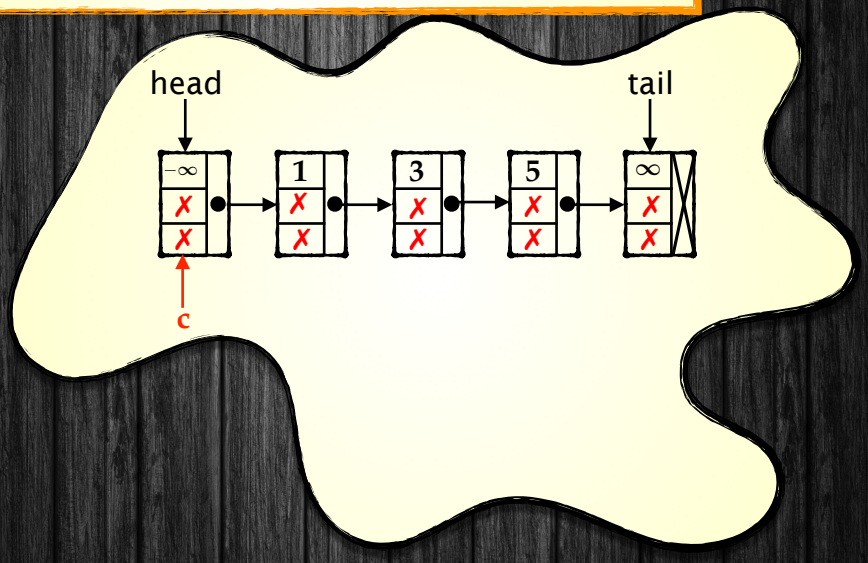



```
ctn(e):
local c
1  c := Head;
2  while (c.val < e)
3     c := c.next
4  b := c.mark
5  if (!b && c.val = e)
6     return true;
7  else return false;
```

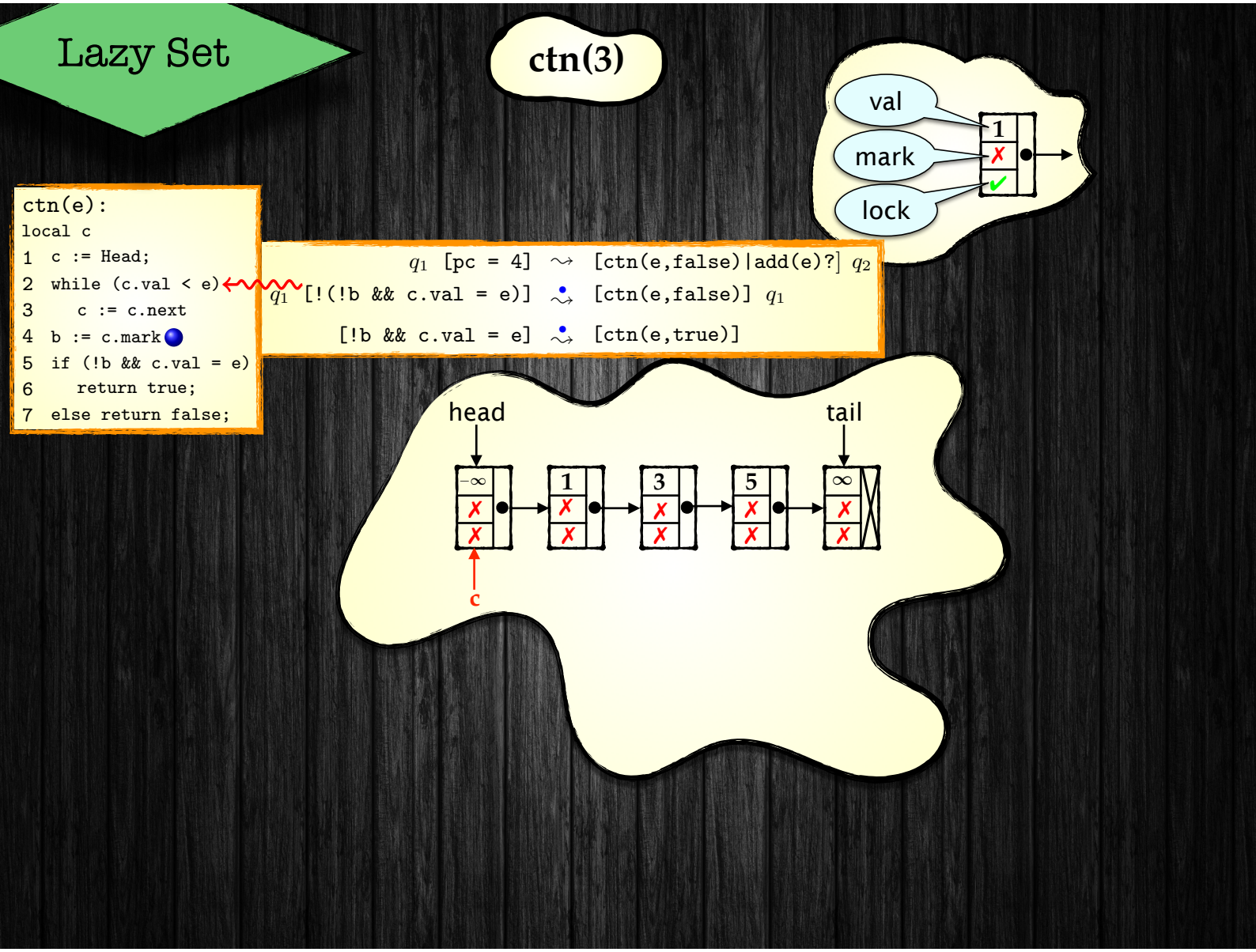
q_1 [pc = 4] $\rightsquigarrow$ [ctn(e,false)|add(e)?] q_2

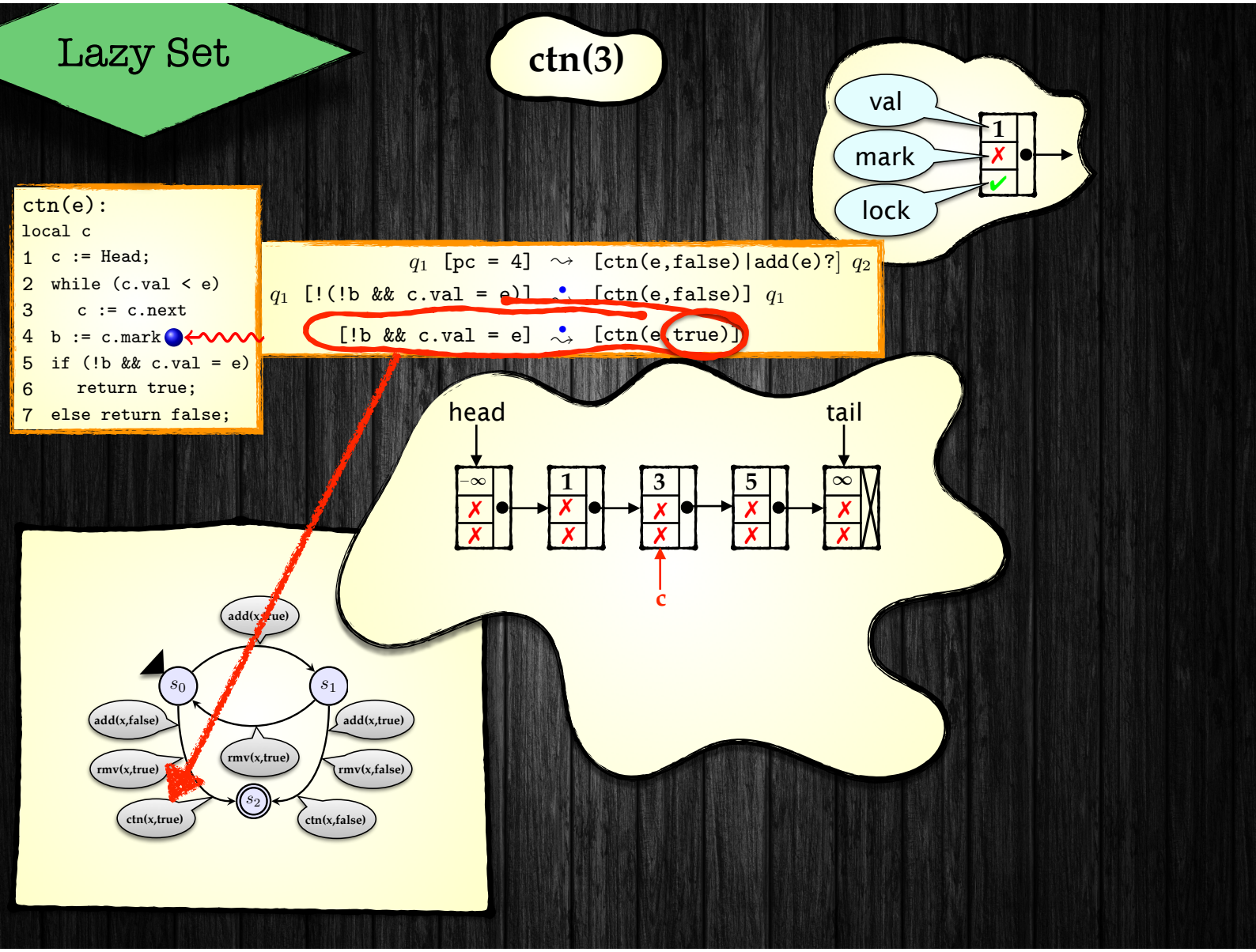
q_1 [!(!b && c.val = e)] $\rightsquigarrow$ [ctn(e,false)] q_1

[!b && c.val = e] $\rightsquigarrow$ [ctn(e,true)]



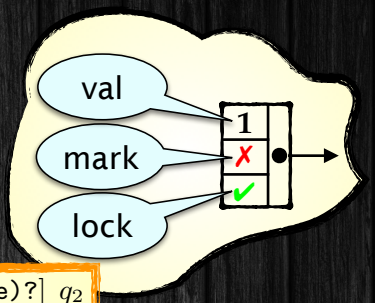
Lazy Set


ctn(3)
val
mark
lock
1
ctn(e):
local c
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;
q_1 [pc = 4] ⇝ [ctn(e,false)|add(e)?] q_2
q_1 [!(!b && c.val = e)] ⇝ [ctn(e,false)] q_1
[!b && c.val = e] ⇝ [ctn(e,true)]
head
tail
−∞
1
3
5
∞
c


Lazy Set
ctn(3)
val
mark
lock
1
ctn(e):
local c
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;
q_1 [pc = 4] ⇝ [ctn(e,false)|add(e)?] q_2
q_1 [!(!b && c.val = e)] ⇝ [ctn(e,false)] q_1
[!b && c.val = e] ⇝ [ctn(e,true)]
head
tail
−∞
1
3
5
∞
c
add(x,true)
s_0
s_1
add(x,false)
add(x,true)
rmv(x,true)
rmv(x,false)
s_2
ctn(x,true)
ctn(x,false)

Lazy Set

ctn(3)

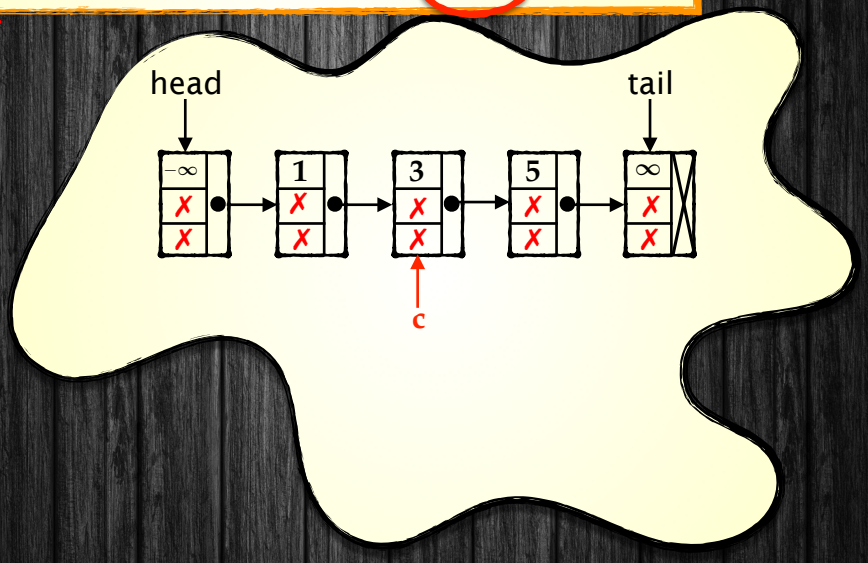



```
ctn(e):
local c
1  c := Head;
2  while (c.val < e)
3     c := c.next
4  b := c.mark
5  if (!b && c.val = e)
6     return true;
7  else return false;
```

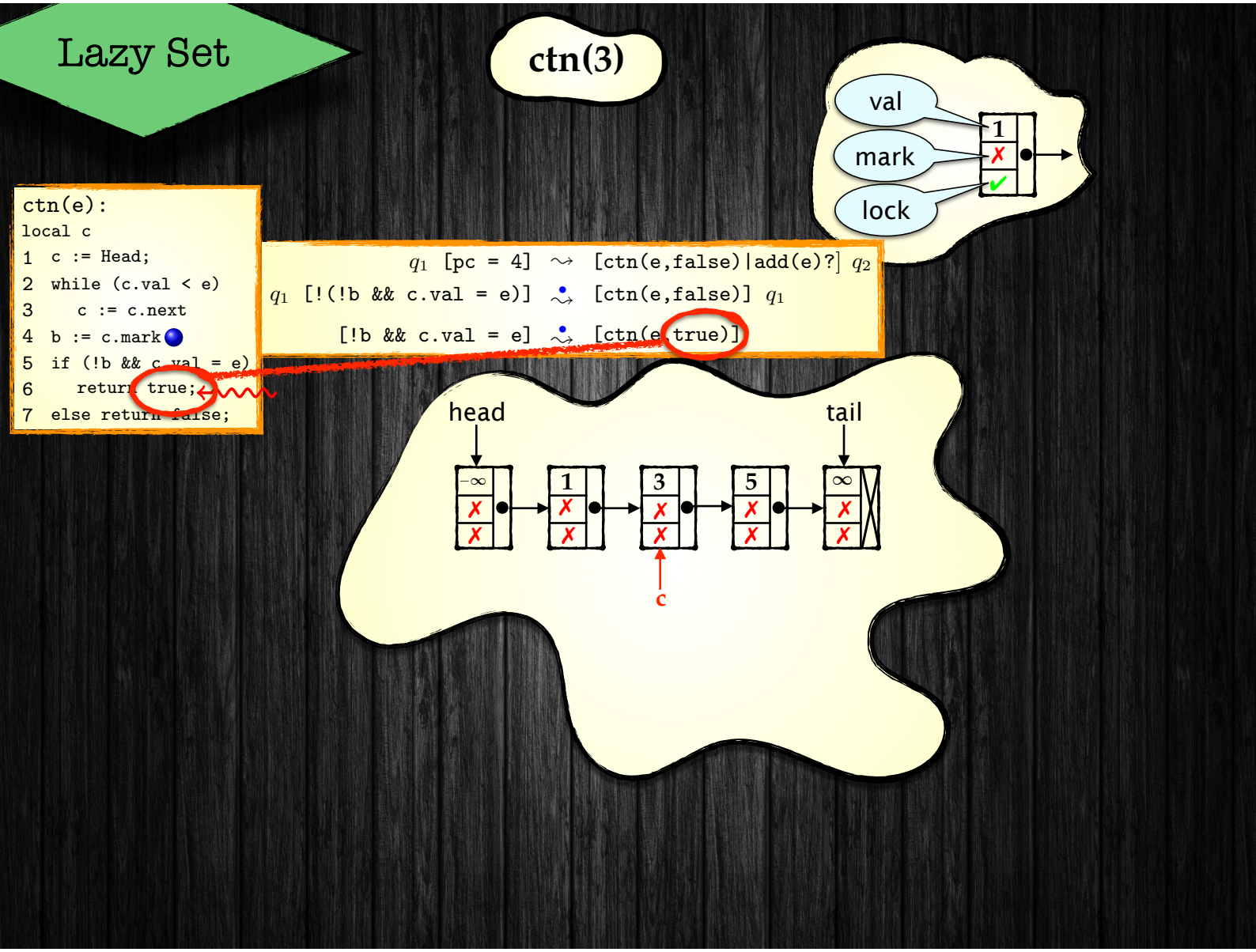
q_1 [pc = 4] $\rightsquigarrow$ [ctn(e,false)|add(e)?] q_2

q_1 [!(!b && c.val = e)] $\rightsquigarrow$ [ctn(e,false)] q_1

[!b && c.val = e] $\rightsquigarrow$ [ctn(e,true)]

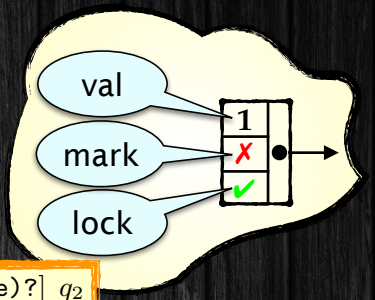


Lazy Set


ctn(3)
val
mark
lock
1
ctn(e):
local c
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;
q_1 [pc = 4] ⇝ [ctn(e,false)|add(e)?] q_2
q_1 [!(!b && c.val = e)] ⇝ [ctn(e,false)] q_1
[!b && c.val = e] ⇝ [ctn(e,true)]
head
tail
−∞
1
3
5
∞
c

Lazy Set

ctn(3)



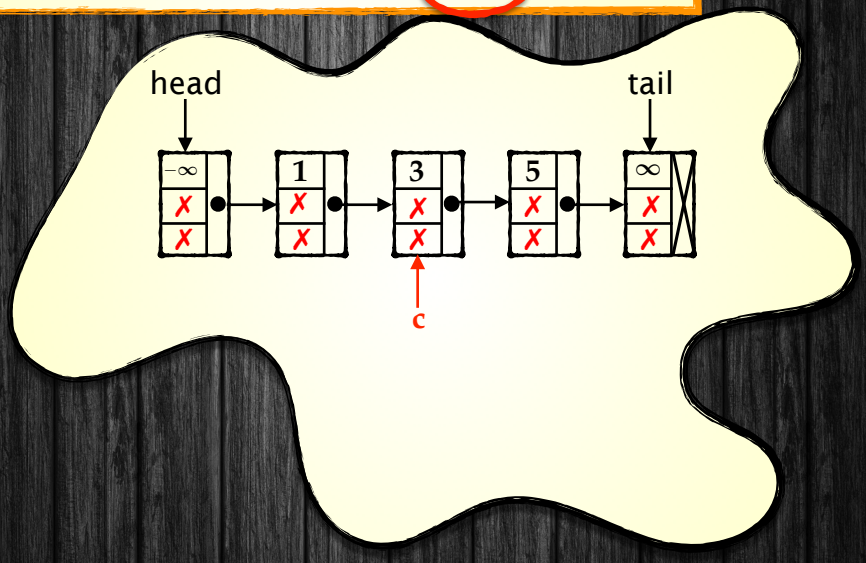

```
ctn(e):
local c
1  c := Head;
2  while (c.val < e)
3     c := c.next
4  b := c.mark
5  if (!b && c.val = e)
6     return true;
7  else return false;
```

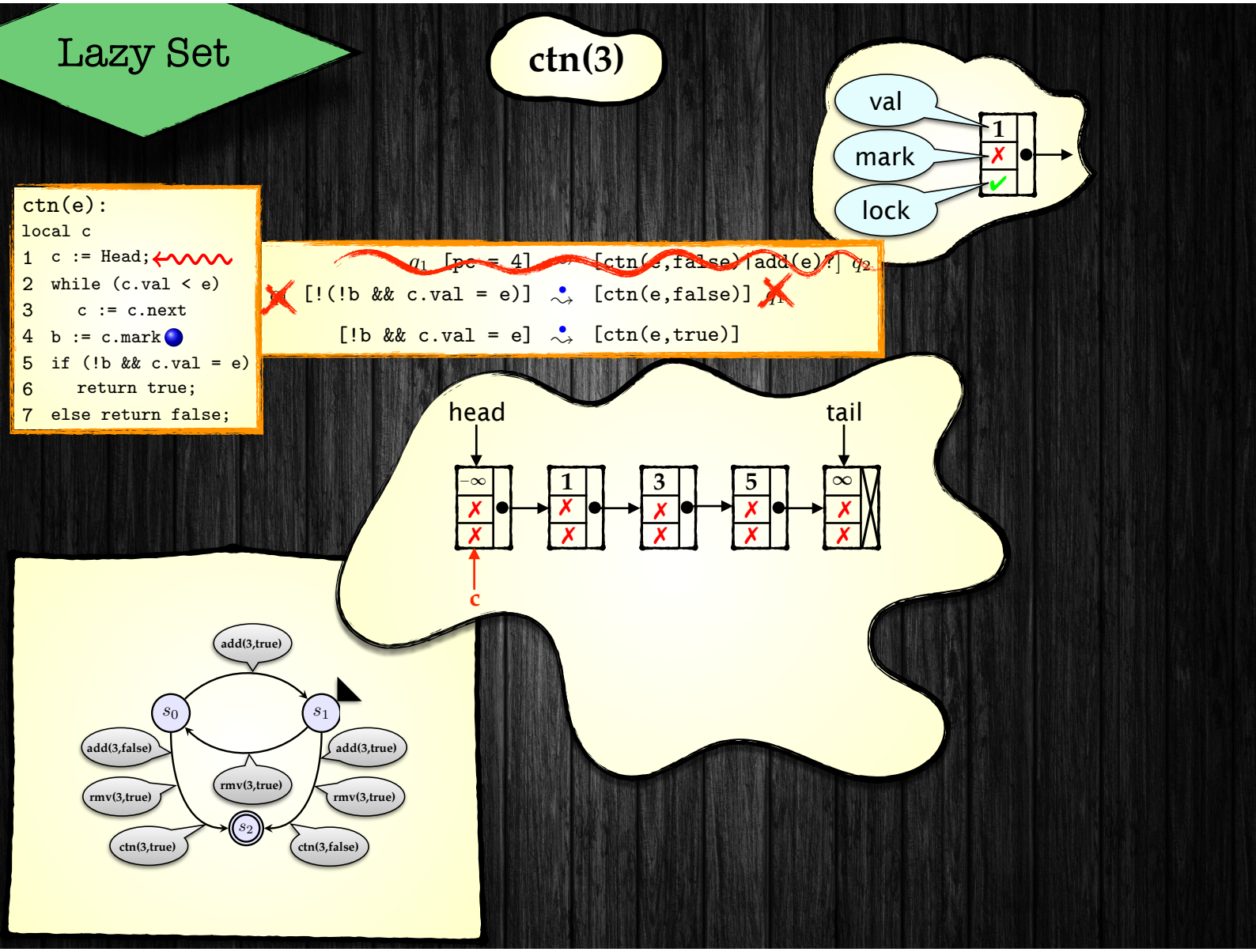
q_1 [pc = 4] $\rightsquigarrow$ [ctn(e,false)|add(e)?] q_2

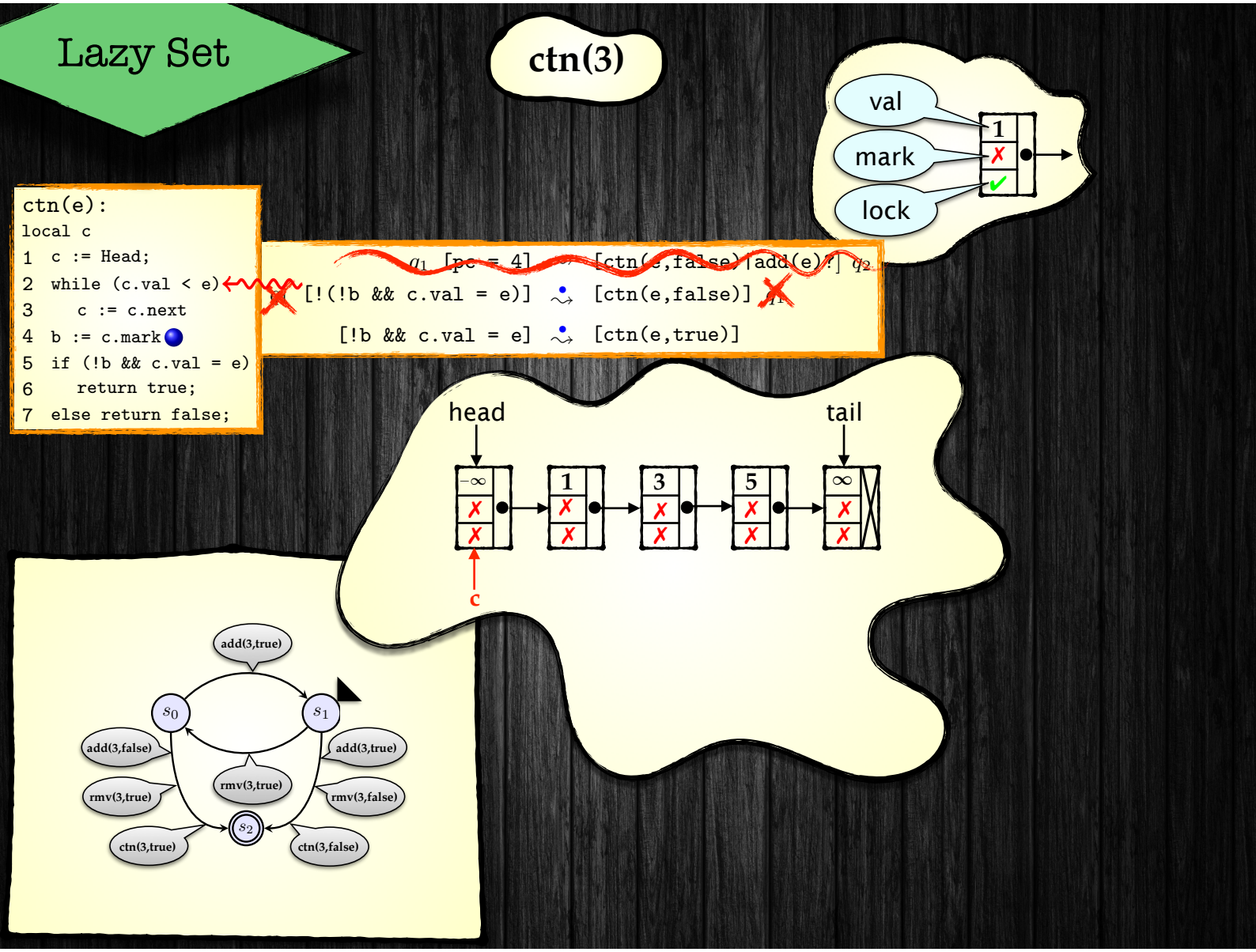
q_1 [!(!b && c.val = e)] $\rightsquigarrow$ [ctn(e,false)] q_1

[!b && c.val = e] $\rightsquigarrow$ [ctn(e,true)]

How about?

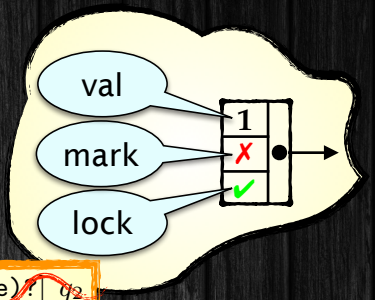



Lazy Set
ctn(3)
val
mark
lock
1
ctn(e):
local c
1 c := Head;
2 while (c.val < e)
3 c := c.next
4 b := c.mark
5 if (!b && c.val = e)
6 return true;
7 else return false;
q_1 [pc = 4] [ctn(e,false)|add(e)?] q_2
[!(!b && c.val = e)] [ctn(e,false)]
[!b && c.val = e] [ctn(e,true)]
head
tail
−∞
1
3
5
∞
c
add(3,true)
s_0
s_1
add(3,false)
add(3,true)
rmv(3,true)
rmv(3,true)
rmv(3,true)
s_2
ctn(3,true)
ctn(3,false)


Lazy Set
ctn(3)
val
mark
lock
1
ctn(e):
local c
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;
[!(!b && c.val = e)] [ctn(e,false)]
[!b && c.val = e] [ctn(e,true)]
head
tail
−∞
1
3
5
∞
c
add(3,true)
add(3,false)
add(3,true)
rmv(3,true)
rmv(3,true)
rmv(3,false)
ctn(3,true)
ctn(3,false)

Lazy Set

ctn(3)

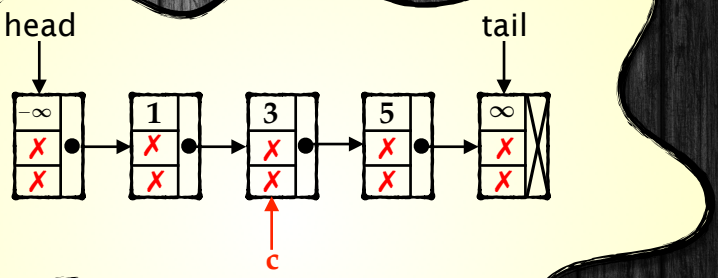


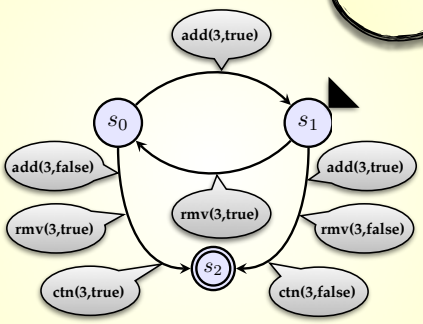

```
ctn(e):
local c
1  c := Head;
2  while (c.val < e)
3     c := c.next
4  b := c.mark
5  if (!b && c.val = e)
6     return true;
7  else return false;
```

q_1 [pc = 4] [ctn(e,false)|add(e)?] q_2

[!(!b && c.val = e)] [ctn(e,false)]

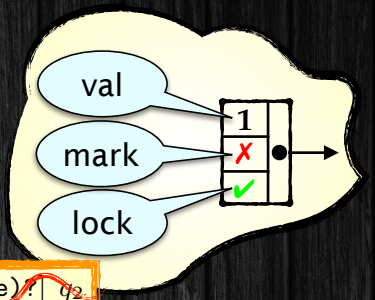
[!b && c.val = e] [ctn(e,true)]





Lazy Set

ctn(3)

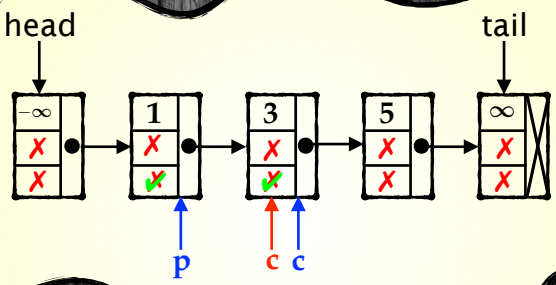


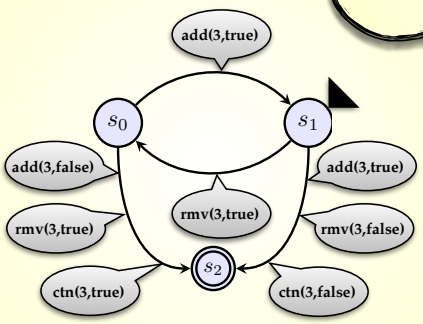

```
ctn(e):
local c
1  c := Head;
2  while (c.val < e)
3     c := c.next
4  b := c.mark
5  if (!b && c.val = e)
6     return true;
7  else return false;
```

q_1 [pc = 4] ⇝ [ctn(e,false)⌈add(e)?⌉] q_2

[!(!b && c.val = e)] ⇝ [ctn(e,false)]

[!b && c.val = e] ⇝ [ctn(e,true)]





rmv(3)

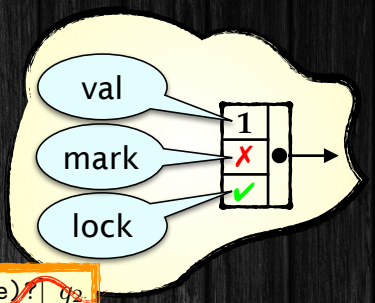
```
rmv(e):
local p, c, n, r
1  (p,c) := locate(e);
2  if (c.val = e)
3     c.mark := true
4     n := c.next;
5     p.next := n;
6     r := true;
7  else r := false;
8  unlock(p);
9  unlock(c);
10 return r;
```

[pc = 3] ⇝ [rmv(e,true)]

[(pc = 2),(c.val <> e)] ⇝ [rmv(e,false)]

Lazy Set

ctn(3)

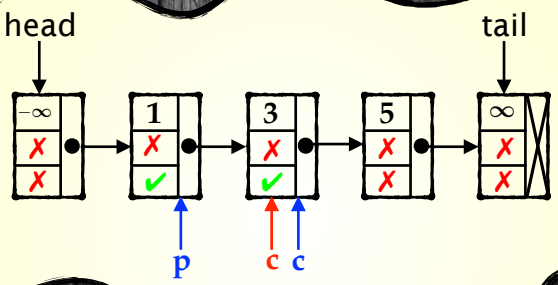


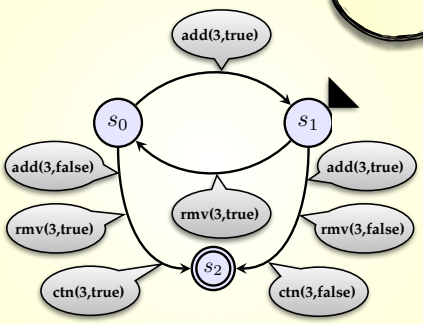

```
ctn(e):
local c
1  c := Head;
2  while (c.val < e)
3     c := c.next
4  b := c.mark
5  if (!b && c.val = e)
6     return true;
7  else return false;
```

q_1 [pc = 4] ⇝ [ctn(e,false)⌈add(e)?⌉] q_2

[!(!b && c.val = e)] ⇝ [ctn(e,false)] q_f

[!b && c.val = e] ⇝ [ctn(e,true)]



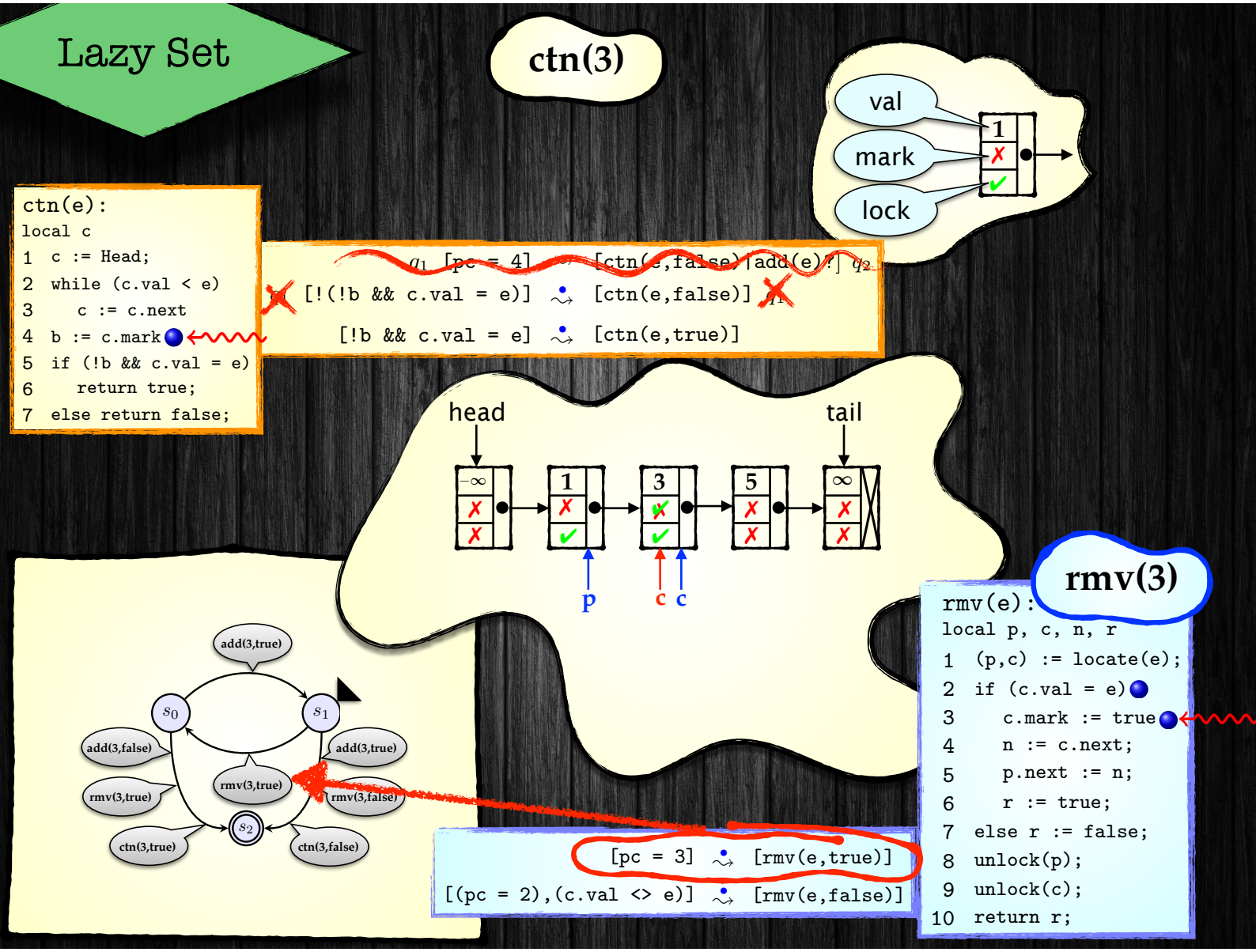


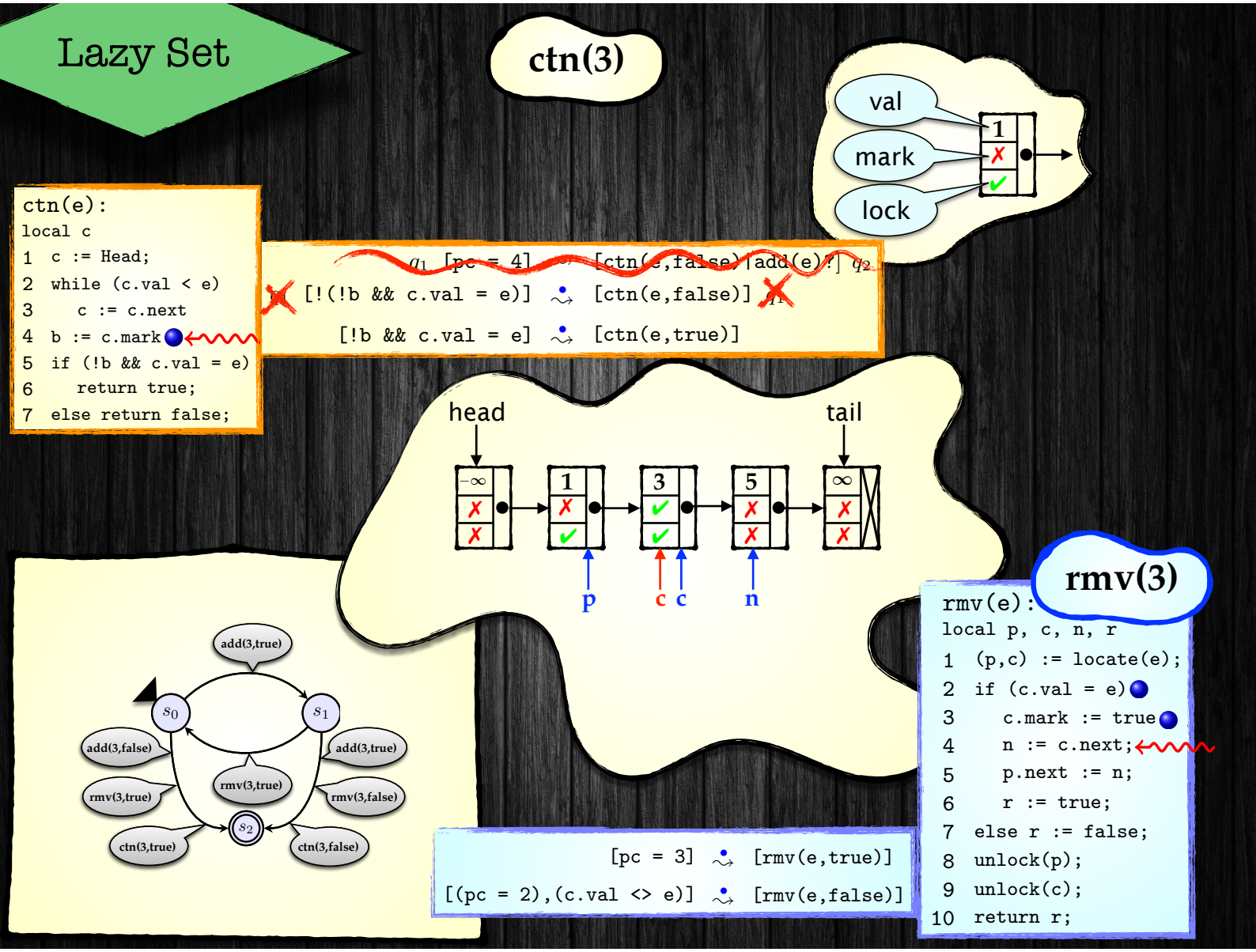
rmv(3)

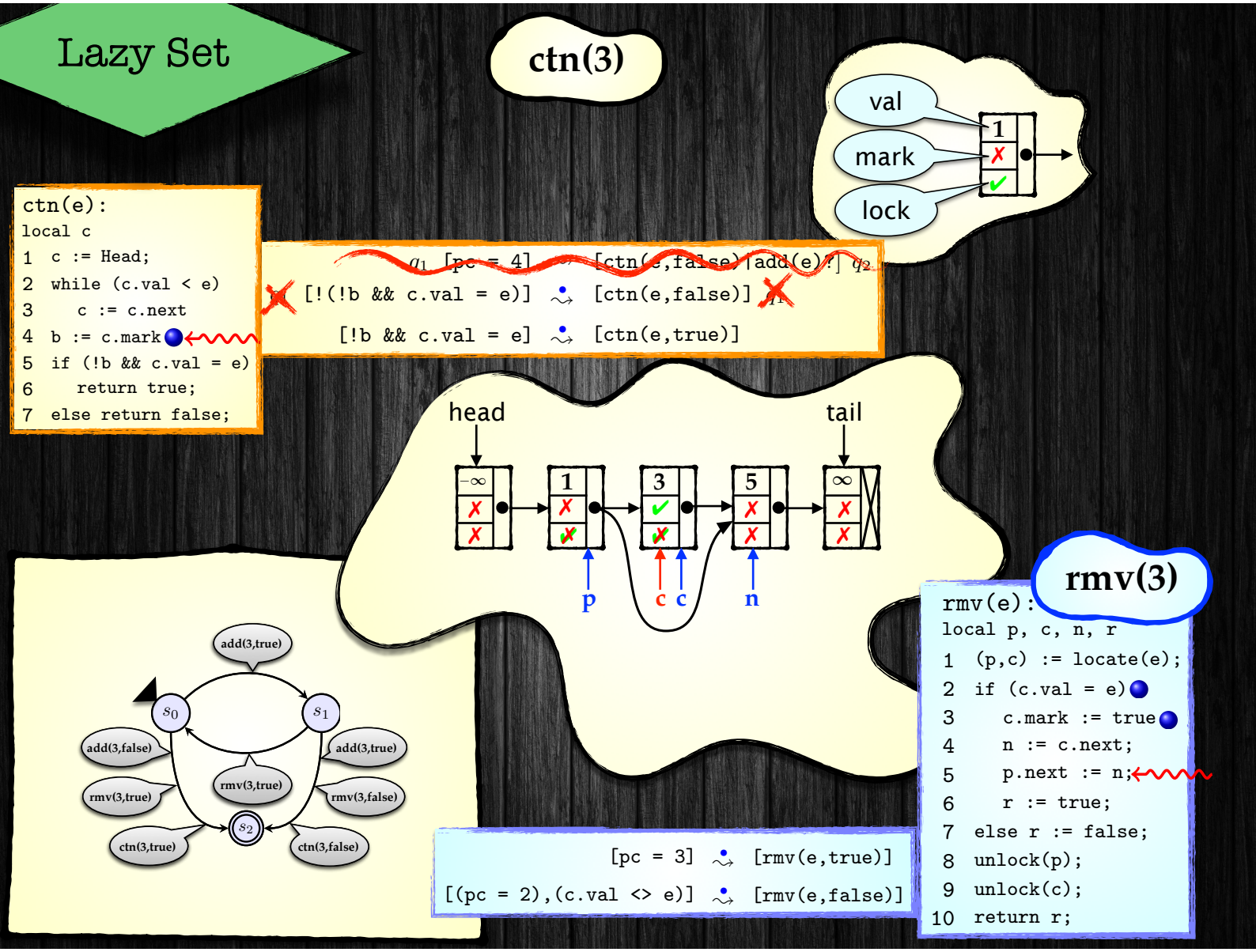
```
rmv(e):
local p, c, n, r
1  (p,c) := locate(e);
2  if (c.val = e)
3     c.mark := true
4     n := c.next;
5     p.next := n;
6     r := true;
7  else r := false;
8  unlock(p);
9  unlock(c);
10 return r;
```

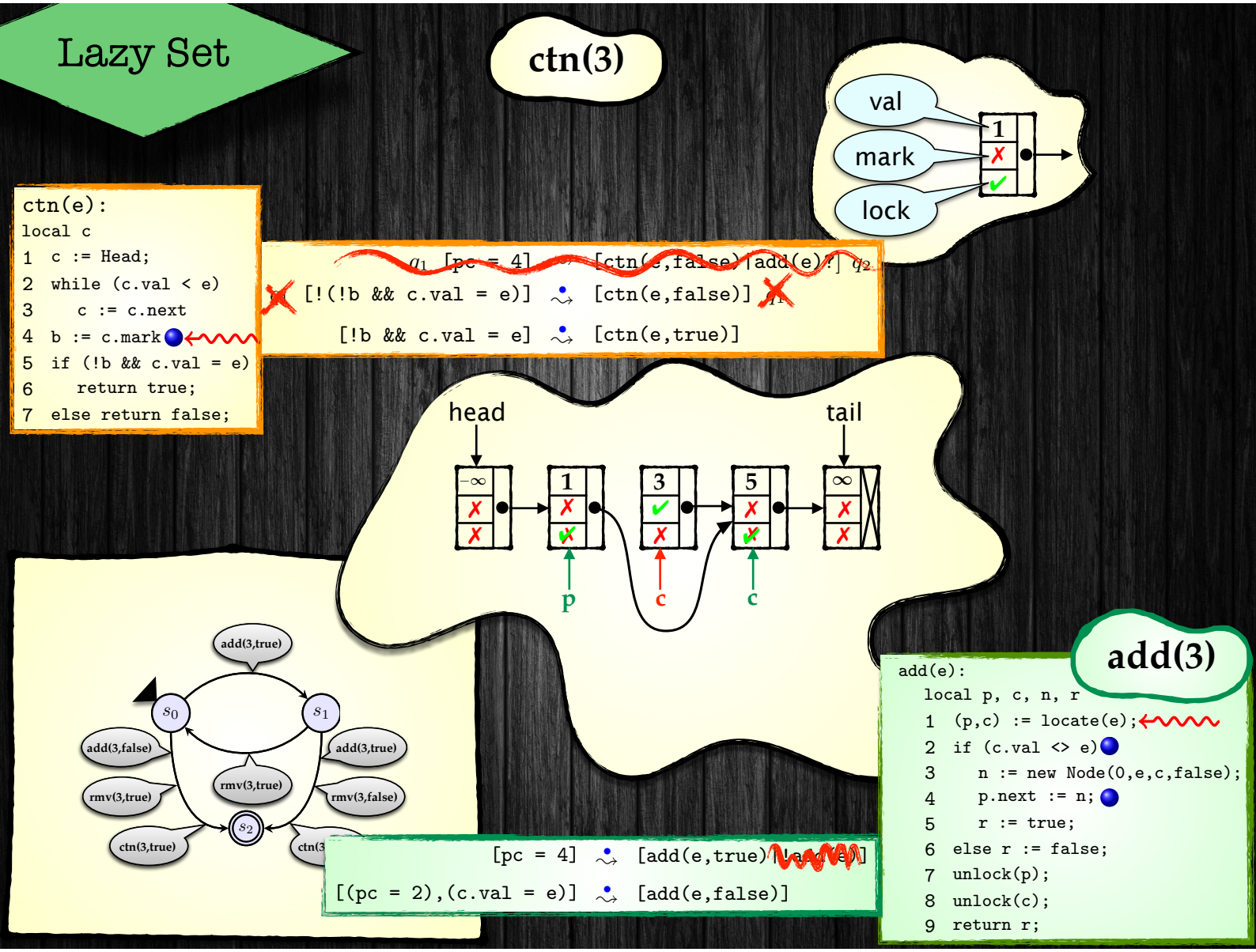
[pc = 3] ⇝ [rmv(e,true)]

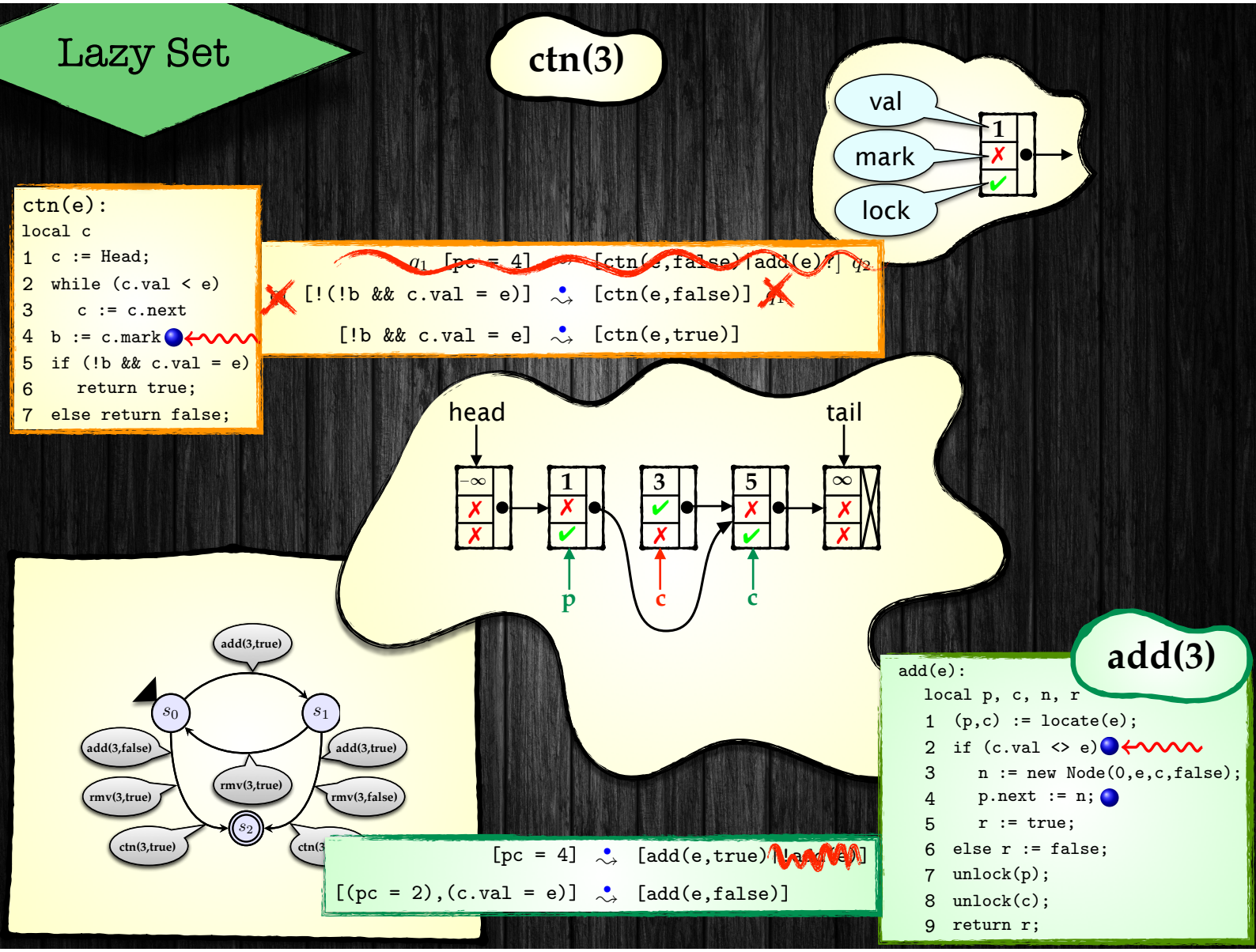
[(pc = 2),(c.val <> e)] ⇝ [rmv(e,false)]

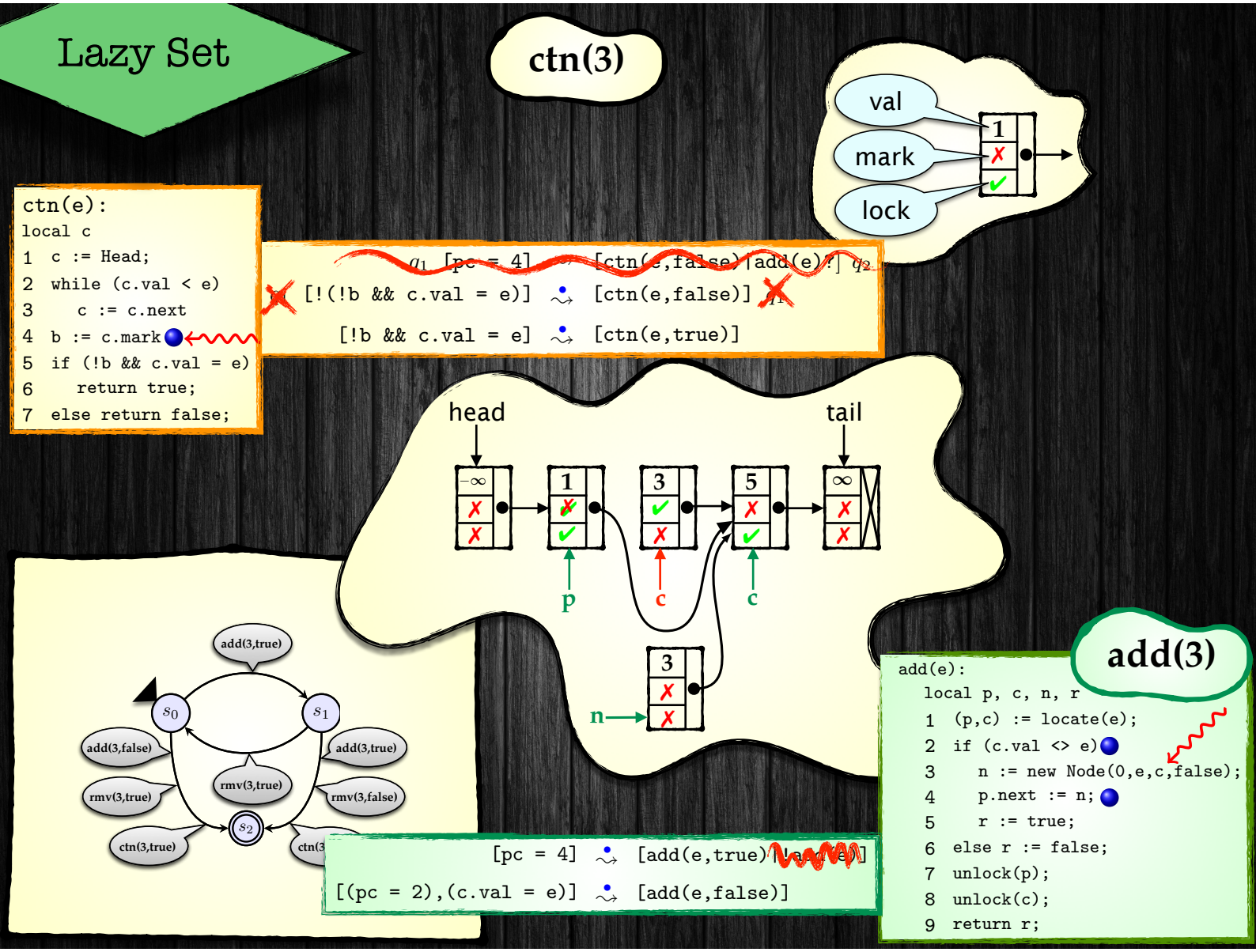

Lazy Set
ctn(3)
val
mark
lock
1
ctn(e):
local c
1 c := Head;
2 while (c.val < e)
3 c := c.next
4 b := c.mark
5 if (!b && c.val = e)
6 return true;
7 else return false;
[!(!b && c.val = e)]
[ctn(e,false)]
[!b && c.val = e]
[ctn(e,true)]
head
tail
−∞
1
3
5
∞
p
c
c
rmv(3)
rmv(e):
local p, c, n, r
1 (p,c) := 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;
[pc = 3]
[rmv(e,true)]
[(pc = 2),(c.val <> e)]
[rmv(e,false)]
add(3,true)
add(3,false)
add(3,true)
rmv(3,true)
rmv(3,true)
rmv(3,false)
ctn(3,true)
ctn(3,false)

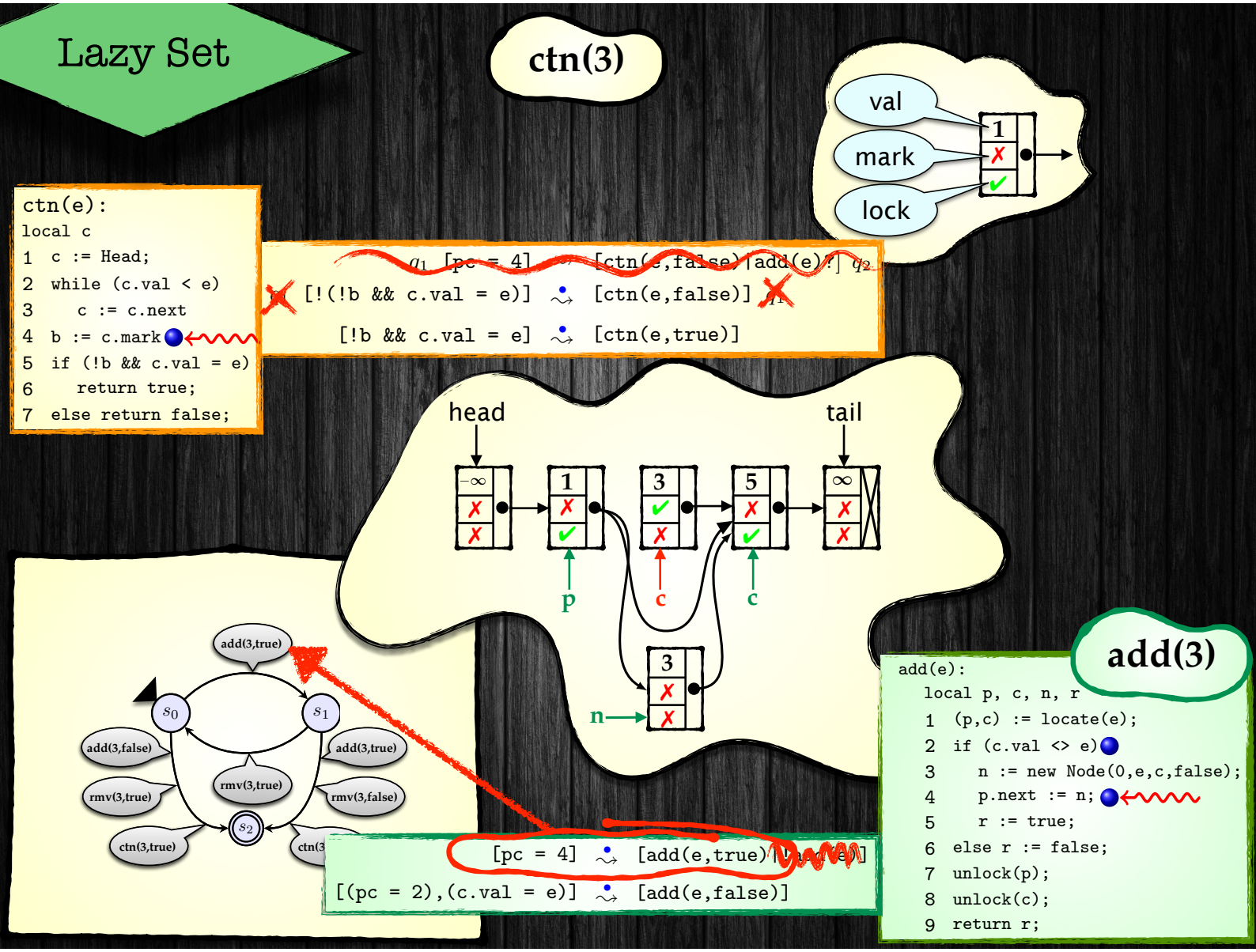

Lazy Set
ctn(3)
val
mark
lock
1
ctn(e):
local c
1 c := Head;
2 while (c.val < e)
3 c := c.next
4 b := c.mark
5 if (!b && c.val = e)
6 return true;
7 else return false;
[!(!b && c.val = e)]
[ctn(e,false)]
[!b && c.val = e]
[ctn(e,true)]
head
tail
−∞
1
3
5
∞
p
c
c
n
rmv(3)
rmv(e):
local p, c, n, r
1 (p,c) := 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;
[pc = 3]
[rmv(e,true)]
[(pc = 2),(c.val <> e)]
[rmv(e,false)]
add(3,true)
add(3,false)
add(3,true)
rmv(3,true)
rmv(3,true)
rmv(3,false)
ctn(3,true)
ctn(3,false)

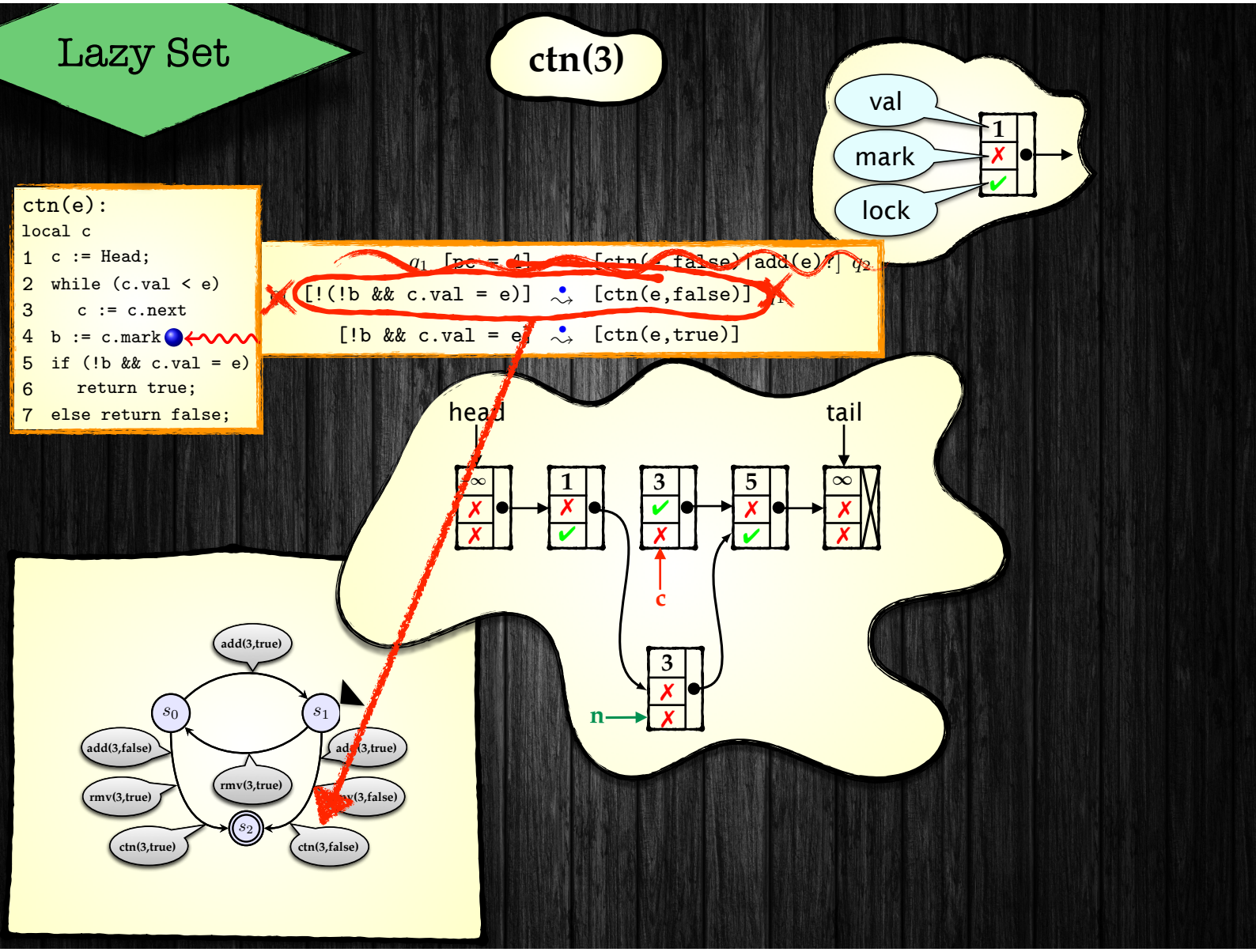

Lazy Set
ctn(3)
val
mark
lock
1
ctn(e):
local c
1 c := Head;
2 while (c.val < e)
3 c := c.next
4 b := c.mark
5 if (!b && c.val = e)
6 return true;
7 else return false;
[!(!b && c.val = e)]
[!b && c.val = e]
[ctn(e,true)]
head
tail
−∞
1
3
5
∞
p
c
c
n
rmv(3)
rmv(e):
local p, c, n, r
1 (p,c) := locate(e);
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;
[pc = 3]
[(pc = 2),(c.val <> e)]
[rmv(e,true)]
[rmv(e,false)]
add(3,true)
add(3,false)
rmv(3,true)
rmv(3,false)
ctn(3,true)
ctn(3,false)

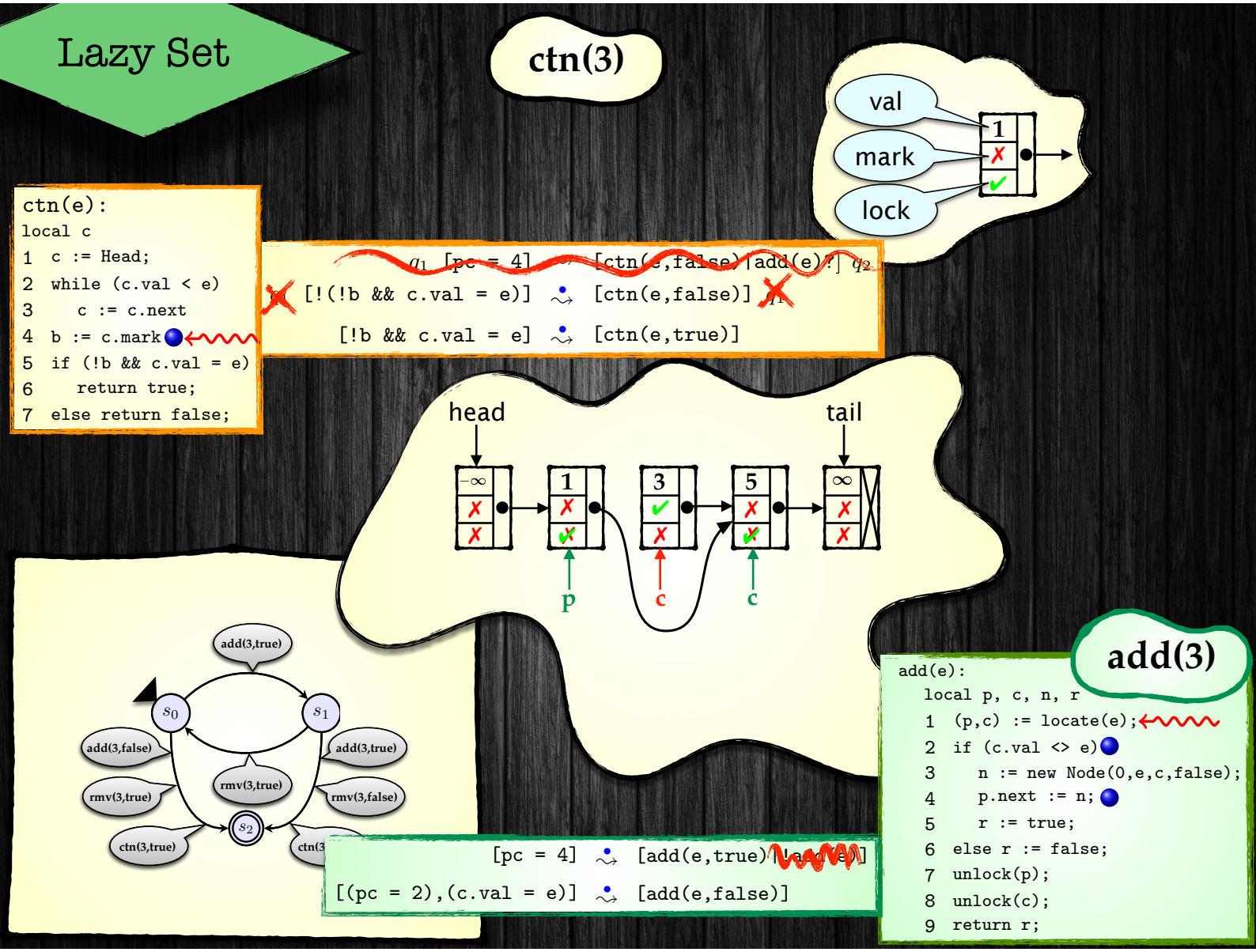

Lazy Set
ctn(3)
val
mark
lock
1
ctn(e):
local c
1 c := Head;
2 while (c.val < e)
3 c := c.next
4 b := c.mark
5 if (!b && c.val = e)
6 return true;
7 else return false;
[!(!b && c.val = e)]
[ctn(e,false)]
[!b && c.val = e]
[ctn(e,true)]
head
tail
−∞
1
3
5
∞
p
c
c
add(3,true)
add(3,false)
add(3,true)
rmv(3,true)
rmv(3,true)
rmv(3,false)
ctn(3,true)
[pc = 4]
[add(e,true)]
[(pc = 2),(c.val = e)]
[add(e,false)]
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;


Lazy Set
ctn(3)
val
mark
lock
1
ctn(e):
local c
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;
[!(!b && c.val = e)]
[ctn(e,false)]
[!b && c.val = e]
[ctn(e,true)]
head
tail
−∞
1
3
5
∞
p
c
c
add(3,true)
add(3,false)
add(3,true)
rmv(3,true)
rmv(3,true)
rmv(3,false)
ctn(3,true)
[pc = 4]
[add(e,true)]
[(pc = 2),(c.val = e)]
[add(e,false)]
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;


Lazy Set
ctn(3)
val
mark
lock
1
ctn(e):
local c
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;
[!(!b && c.val = e)]
[ctn(e,false)]
[!b && c.val = e]
[ctn(e,true)]
head
tail
−∞
1
3
5
∞
p
c
c
n
add(3,true)
add(3,false)
add(3,true)
rmv(3,true)
rmv(3,true)
rmv(3,false)
ctn(3,true)
[pc = 4]
[add(e,true)]
[(pc = 2),(c.val = e)]
[add(e,false)]
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;

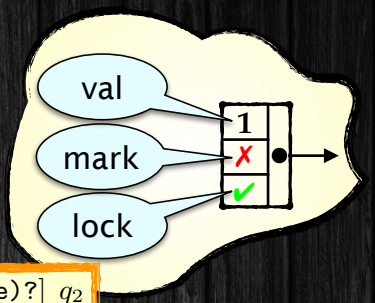

Lazy Set
ctn(3)
val
mark
lock
1
ctn(e):
local c
1 c := Head;
2 while (c.val < e)
3 c := c.next
4 b := c.mark
5 if (!b && c.val = e)
6 return true;
7 else return false;
[!(!b && c.val = e)]
[ctn(e,false)]
[!b && c.val = e]
[ctn(e,true)]
head
tail
−∞
1
3
5
∞
p
c
c
n
add(3,true)
add(3,false)
add(3,true)
rmv(3,true)
rmv(3,true)
rmv(3,false)
ctn(3,true)
[pc = 4]
[add(e,true)]
[(pc = 2),(c.val = e)]
[add(e,false)]
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;


Lazy Set
ctn(3)
val
mark
lock
1
ctn(e):
local c
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;
[!(!b && c.val = e)] [ctn(e,false)]
[!b && c.val = e] [ctn(e,true)]
head
tail
c
n
add(3,true)
add(3,false)
rmv(3,true)
rmv(3,false)
ctn(3,true)
ctn(3,false)


Lazy Set
ctn(3)
val
mark
lock
1
ctn(e):
local c
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;
[!(!b && c.val = e)]
[ctn(e,false)]
[!b && c.val = e]
[ctn(e,true)]
head
tail
−∞
1
3
5
∞
p
c
c
add(3,true)
add(3,false)
add(3,true)
rmv(3,true)
rmv(3,true)
rmv(3,false)
ctn(3,true)
[pc = 4]
[add(e,true)]
[(pc = 2),(c.val = e)]
[add(e,false)]
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;

Lazy Set

ctn(3)

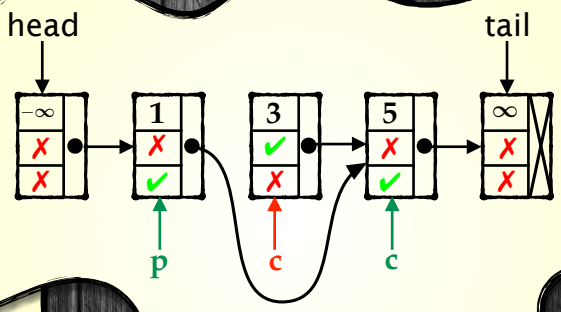


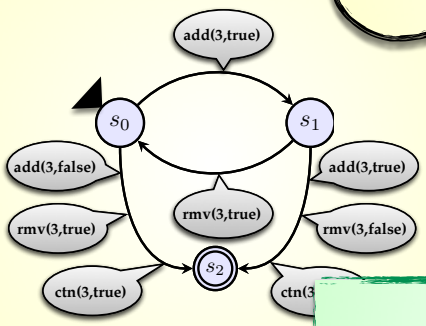

```
ctn(e):
local c
1  c := Head;
2  while (c.val < e)
3     c := c.next
4  b := c.mark
5  if (!b && c.val = e)
6     return true;
7  else return false;
```

q_1 [pc = 4] $\rightsquigarrow$ [ctn(e,false)|add(e)?] q_2

q_1 [!(!b && c.val = e)] $\rightsquigarrow$ [ctn(e,false)] q_1

[!b && c.val = e] $\rightsquigarrow$ [ctn(e,true)]





[pc = 4] $\rightsquigarrow$ [add(e,true)|!add(e)]

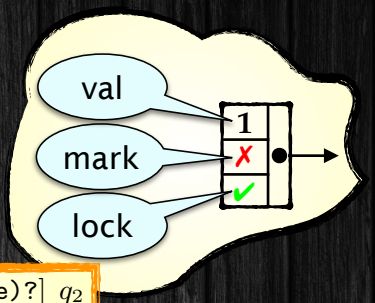
[(pc = 2),(c.val = e)] $\rightsquigarrow$ [add(e,false)]

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;
```

Lazy Set

ctn(3)

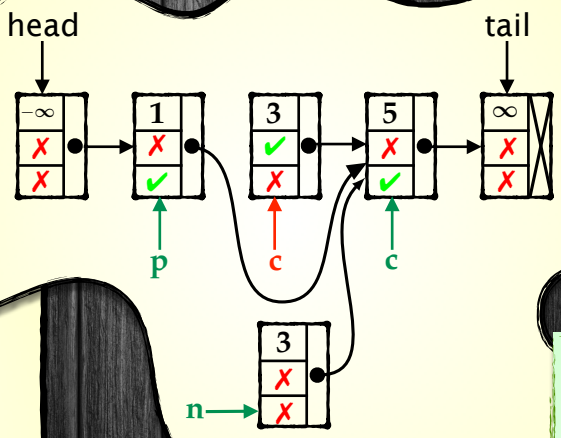


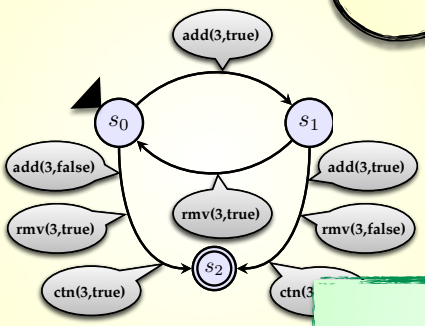

```
ctn(e):
local c
1  c := Head;
2  while (c.val < e)
3     c := c.next
4  b := c.mark
5  if (!b && c.val = e)
6     return true;
7  else return false;
```

q_1 [pc = 4] $\rightsquigarrow$ [ctn(e,false)|add(e)?] q_2

q_1 [!(!b && c.val = e)] $\rightsquigarrow$ [ctn(e,false)] q_1

[!b && c.val = e] $\rightsquigarrow$ [ctn(e,true)]



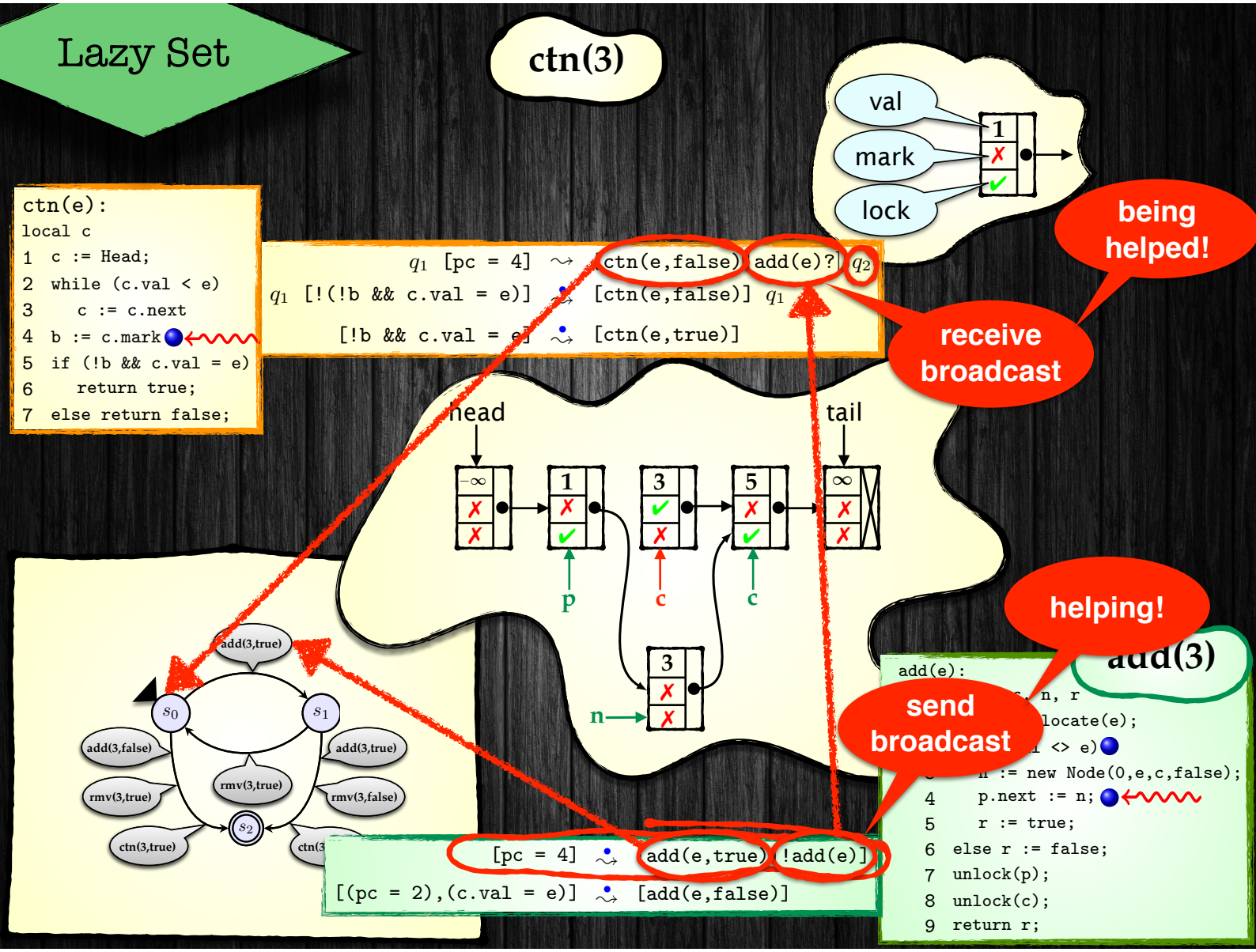


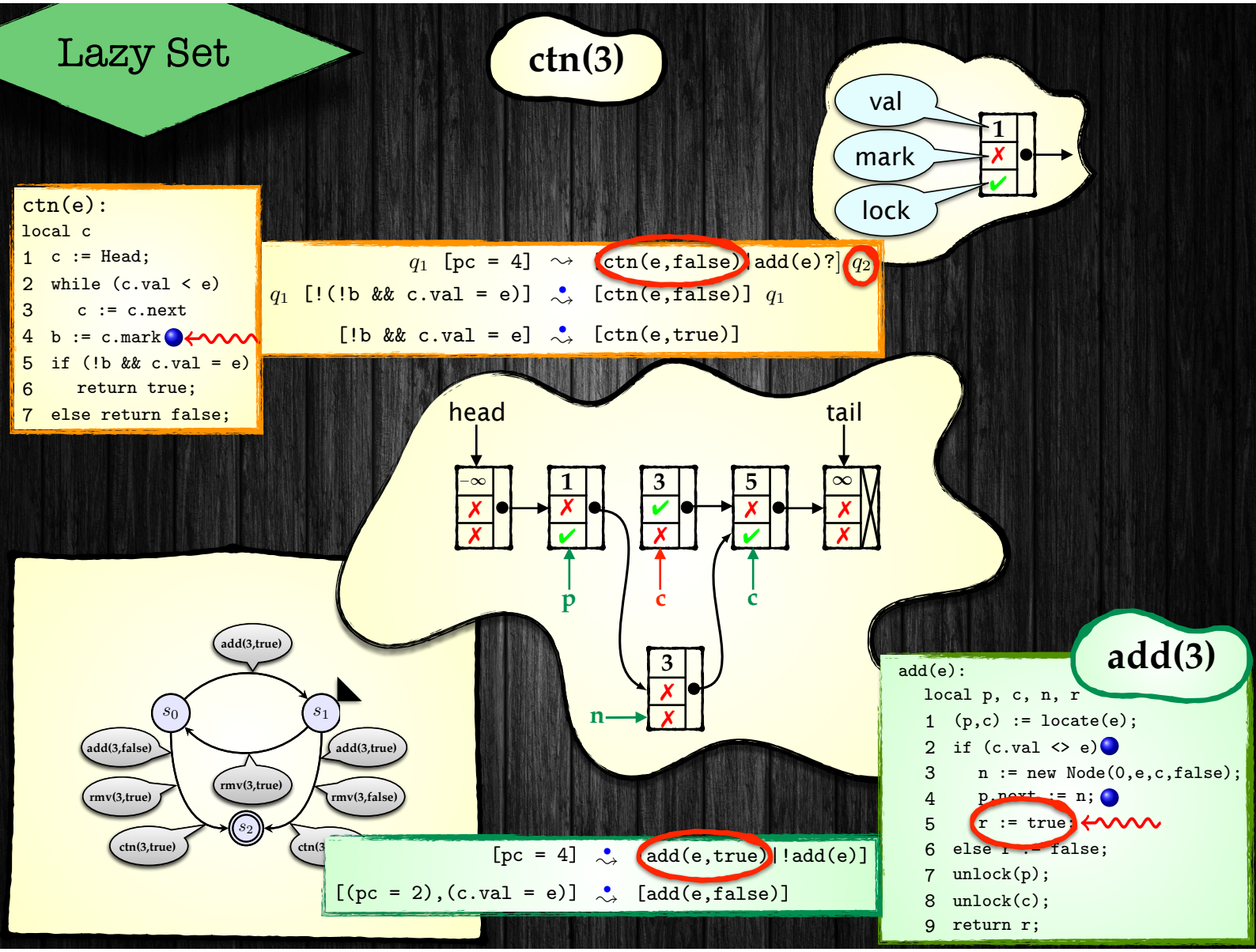
[pc = 4] $\rightsquigarrow$ [add(e,true)|!add(e)]

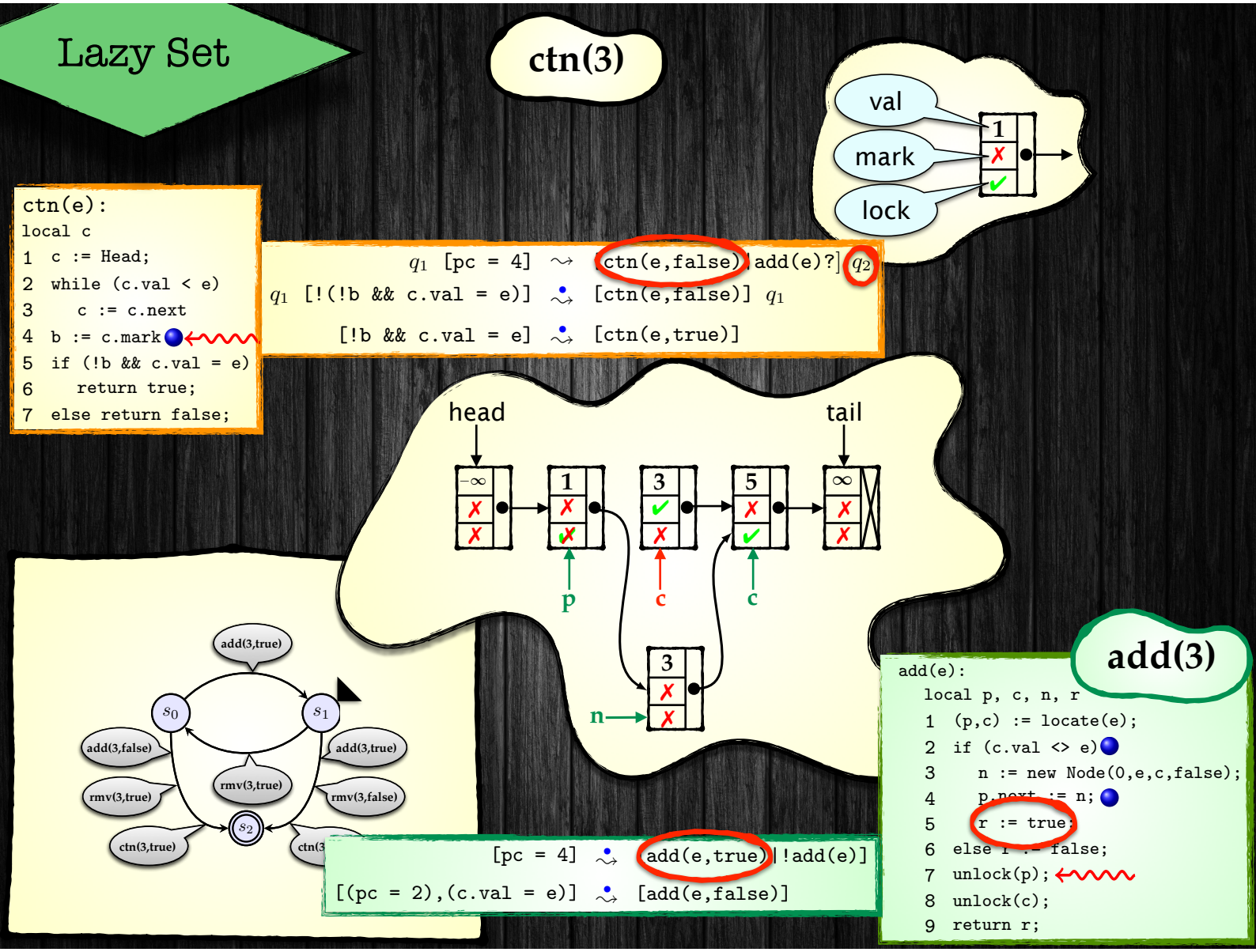
[(pc = 2),(c.val = e)] $\rightsquigarrow$ [add(e,false)]

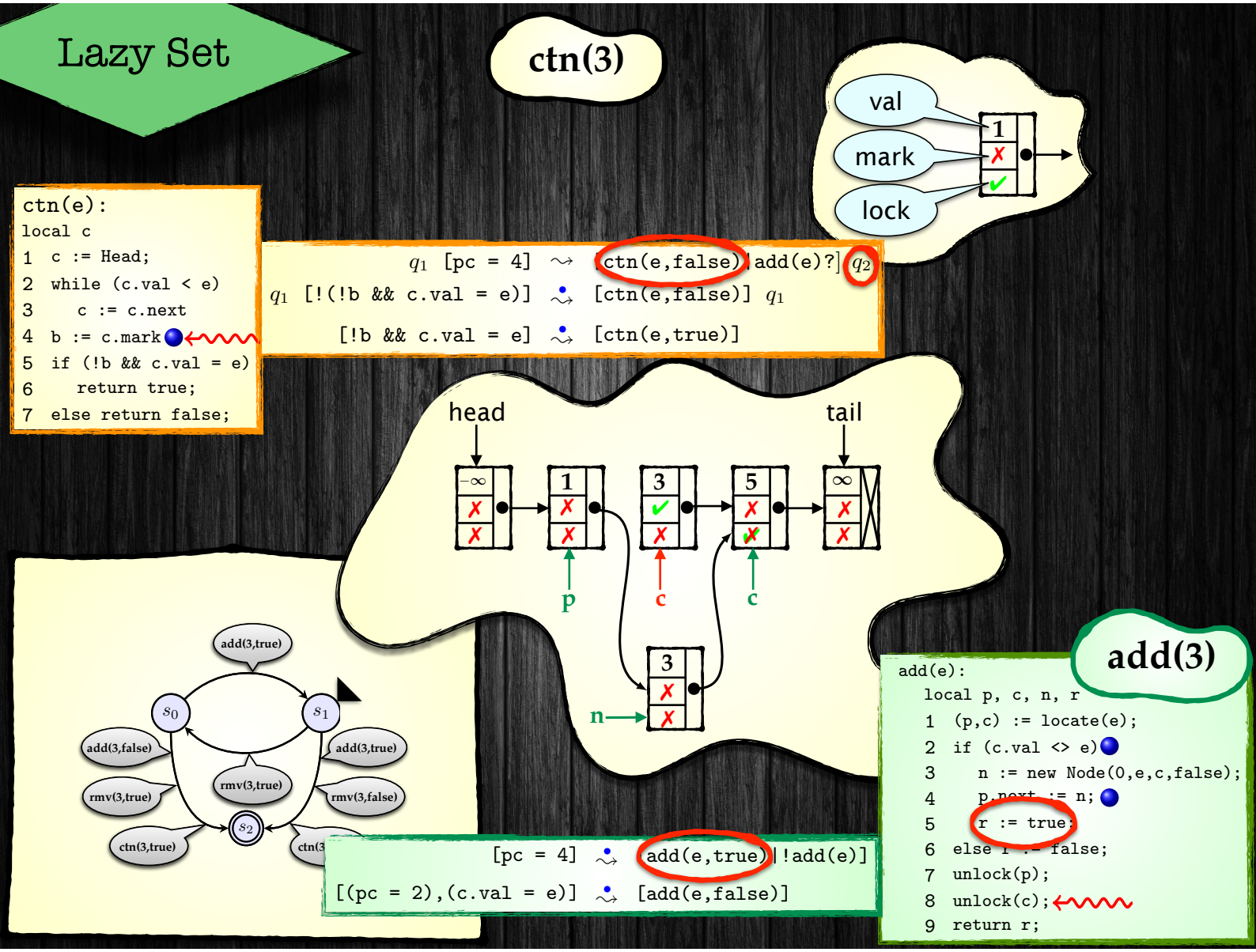
add(3)

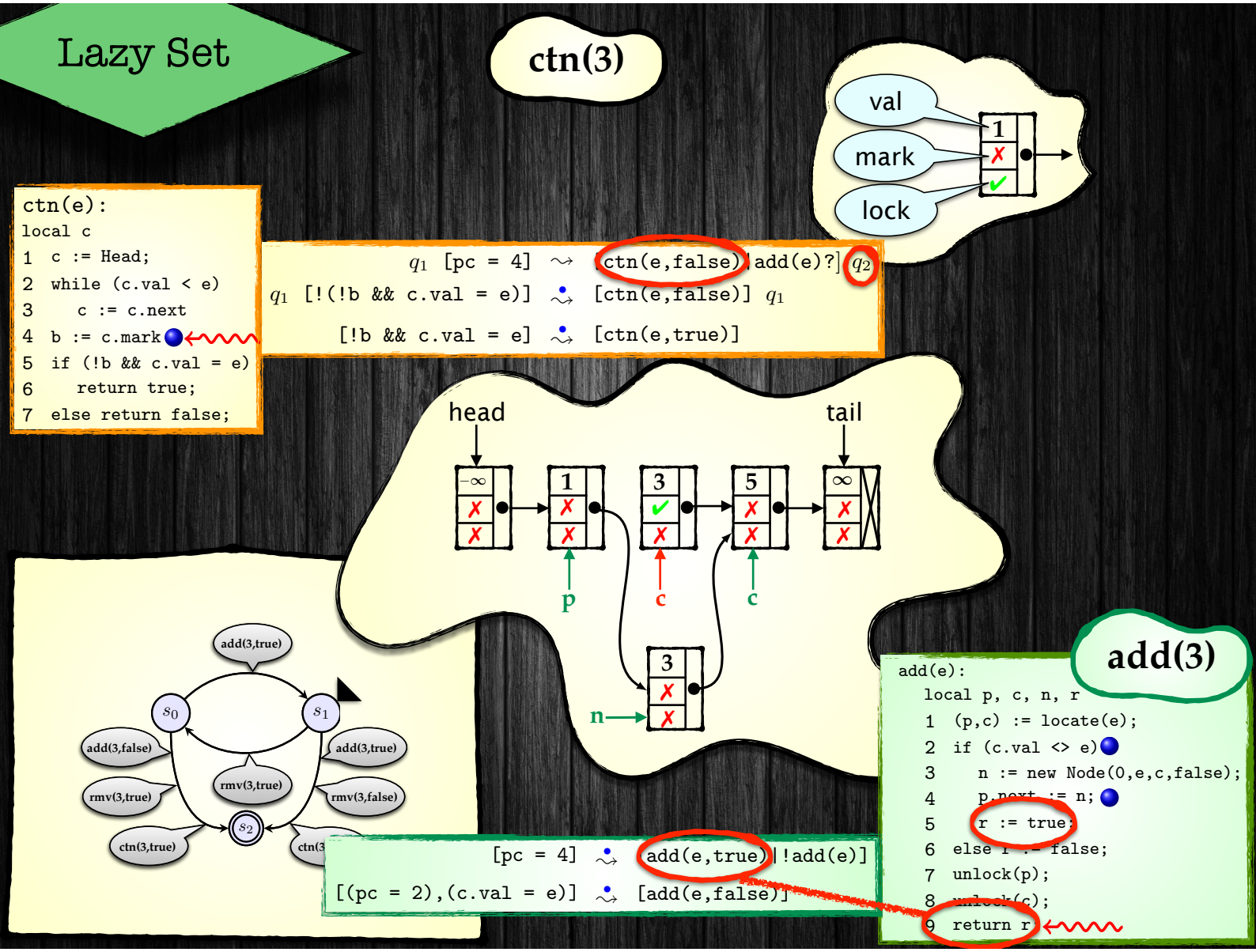
```
add(e):
   local p, c, n, r
   1  (p,c) := locate(e);
   2  if (c.val <> e)
   3     n := new Node(0,e,c,false);
   4     p.next := n;
   5     r := true;
   6  else r := false;
   7  unlock(p);
   8  unlock(c);
   9  return r;
```


Lazy Set
ctn(3)
val
mark
lock
1
being helped!
receive broadcast
ctn(e):
local c
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;
q1 [pc = 4] ctn(e,false) add(e)? q2
q1 [!(!b && c.val = e)] [ctn(e,false)] q1
[!b && c.val = e] [ctn(e,true)]
head
tail
−∞
1
3
5
∞
p
c
c
n
3
helping!
add(3)
send broadcast
add(e):
n, r
locate(e);
<> e)
:= 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,true)
s0
s1
add(3,false)
add(3,true)
rmv(3,true)
rmv(3,false)
s2
ctn(3,true)
[pc = 4] add(e,true) !add(e)]
[(pc = 2),(c.val = e)] [add(e,false)]


Lazy Set
ctn(3)
val
mark
lock
1
ctn(e):
local c
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;
q_1 [pc = 4] ⇝ [ctn(e,false) | add(e)?] q_2
q_1 [!(!b && c.val = e)] ⇝ [ctn(e,false)] q_1
[!b && c.val = e] ⇝ [ctn(e,true)]
head
tail
−∞
1
3
5
∞
p
c
c
3
n
add(3,true)
s_0
s_1
add(3,false)
add(3,true)
rmv(3,true)
rmv(3,true)
rmv(3,false)
s_2
ctn(3,true)
[pc = 4] ⇝ [add(e,true) | !add(e)]
[(pc = 2),(c.val = e)] ⇝ [add(e,false)]
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;

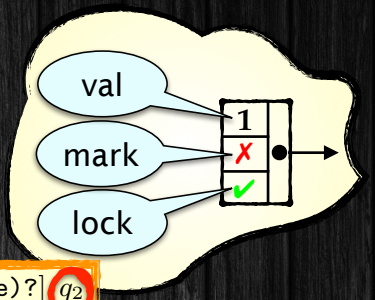

Lazy Set
ctn(3)
val
mark
lock
1
ctn(e):
local c
1 c := Head;
2 while (c.val < e)
3 c := c.next
4 b := c.mark
5 if (!b && c.val = e)
6 return true;
7 else return false;
q1 [pc = 4] ⇝ [ctn(e,false) | add(e)?] q2
q1 [!(!b && c.val = e)] ⇝ [ctn(e,false)] q1
[!b && c.val = e] ⇝ [ctn(e,true)]
head
tail
−∞
1
3
5
∞
p
c
c
n
add(3,true)
s0
s1
add(3,false)
add(3,true)
rmv(3,true)
rmv(3,true)
rmv(3,false)
s2
ctn(3,true)
[pc = 4] ⇝ [add(e,true) | !add(e)]
[(pc = 2),(c.val = e)] ⇝ [add(e,false)]
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;


Lazy Set
ctn(3)
val
mark
lock
1
ctn(e):
local c
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;
q_1 [pc = 4] ⇝ [ctn(e,false)|add(e)?] q_2
q_1 [!(!b && c.val = e)] ⇝ [ctn(e,false)] q_1
[!b && c.val = e] ⇝ [ctn(e,true)]
head
tail
−∞
1
3
5
∞
p
c
c
n
add(3,true)
add(3,false)
add(3,true)
rmv(3,true)
rmv(3,true)
rmv(3,false)
ctn(3,true)
s_0
s_1
s_2
[pc = 4] ⇝ [add(e,true)|!add(e)]
[(pc = 2),(c.val = e)] ⇝ [add(e,false)]
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;


Lazy Set
ctn(3)
val
mark
lock
1
ctn(e):
local c
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;
q1 [pc = 4] ⇝ [ctn(e,false)|add(e)?] q2
q1 [!(!b && c.val = e)] ⇝ [ctn(e,false)] q1
[!b && c.val = e] ⇝ [ctn(e,true)]
head
tail
−∞
1
3
5
∞
p
c
c
n
add(3,true)
s0
s1
add(3,false)
add(3,true)
rmv(3,true)
rmv(3,true)
rmv(3,false)
s2
ctn(3,true)
[pc = 4] ⇝ [add(e,true)|!add(e)]
[(pc = 2),(c.val = e)] ⇝ [add(e,false)]
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;

Lazy Set

ctn(3)

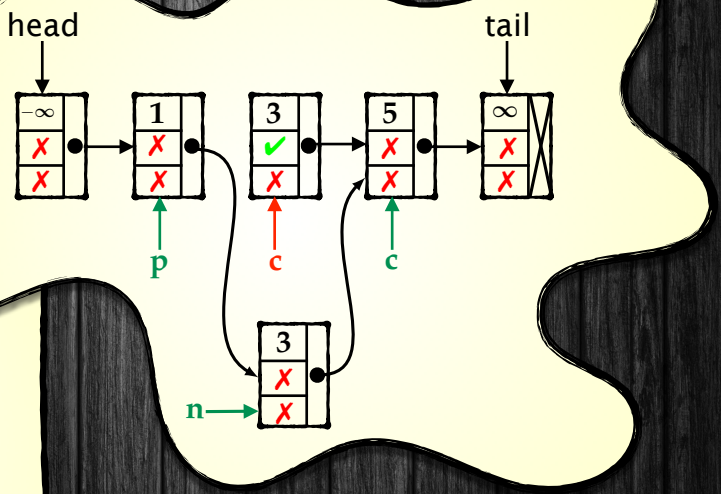


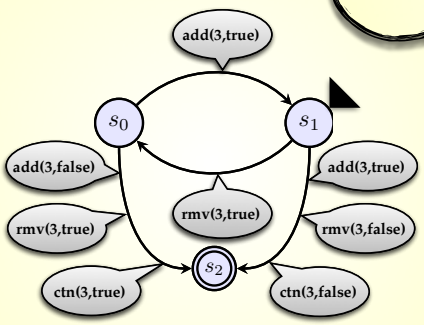

```
ctn(e):
local c
1  c := Head;
2  while (c.val < e)
3     c := c.next
4  b := c.mark
5  if (!b && c.val = e)
6     return true;
7  else return false;
```

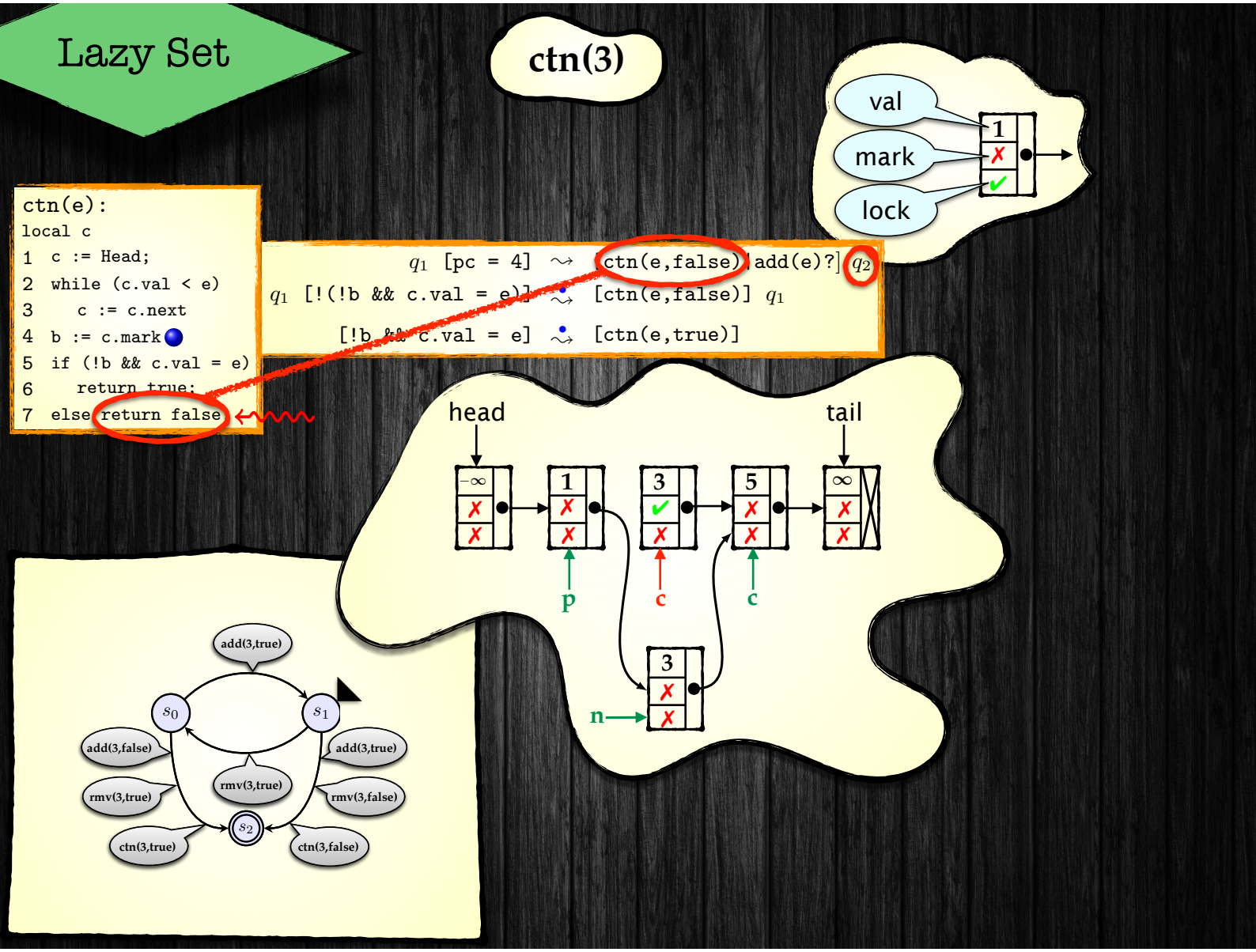
q_1 [pc = 4] ⇝ [ctn(e,false)|add(e)?] q_2

q_1 [!(!b && c.val = e)] ⇝ [ctn(e,false)] q_1

[!b && c.val = e] ⇝ [ctn(e,true)]






Lazy Set
ctn(3)
val
mark
lock
1
ctn(e):
local c
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
q_1 [pc = 4] ⇝ [ctn(e,false)|add(e)?] q_2
q_1 [!(!b && c.val = e)] ⇝ [ctn(e,false)] q_1
[!b && c.val = e] ⇝ [ctn(e,true)]
head
tail
−∞
1
3
5
∞
p
c
c
3
n
add(3,true)
s_0
s_1
add(3,false)
add(3,true)
rmv(3,true)
rmv(3,true)
rmv(3,false)
s_2
ctn(3,true)
ctn(3,false)