

Håkan Sundell, Philippas Tsigas

Fast and lock-free concurrent priority queue for multi-thread systems

Presented by Andreas Haas
University of Salzburg
Concurrency and Memory Management Seminar
2010-12-16

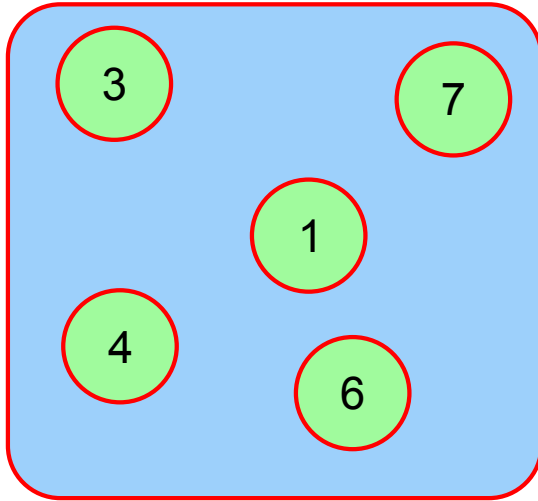
Overview

- ★ Introduction
- ★ Priority queue
- ★ Skip list
- ★ Non-blocking algorithm
- ★ Correctness

Non-blocking

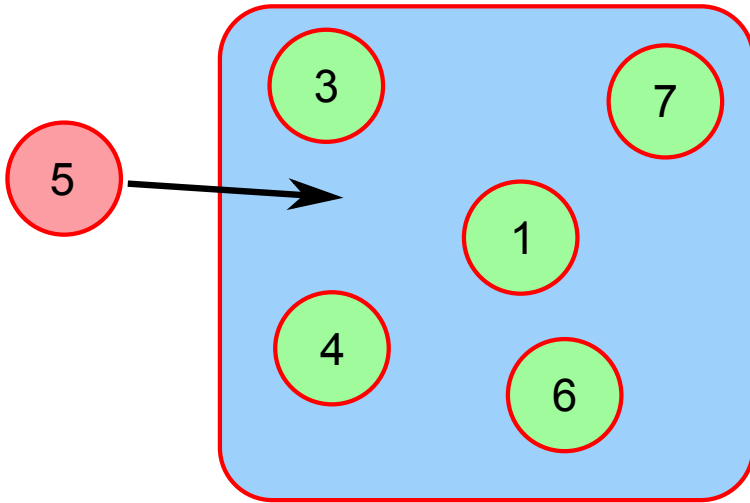
- ★ No mutual exclusion
- ★ Guaranteed progress of at least one operation
- ★ Atomic operations:
 - 🌀 Test-And-Set (TAS)
 - 🌀 Fetch-And-Add (FAA)
 - 🌀 Compare-And-Swap (CAS)

Priority queue



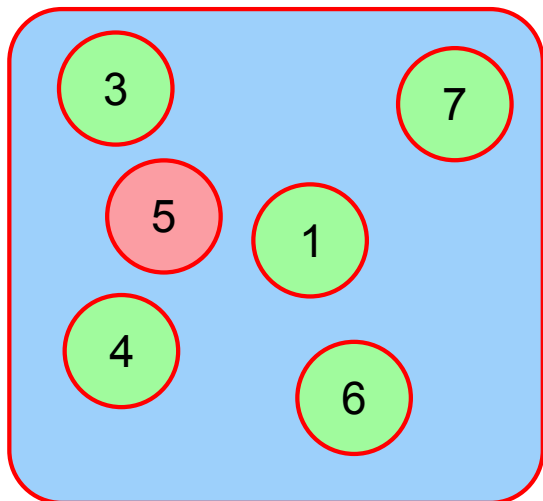
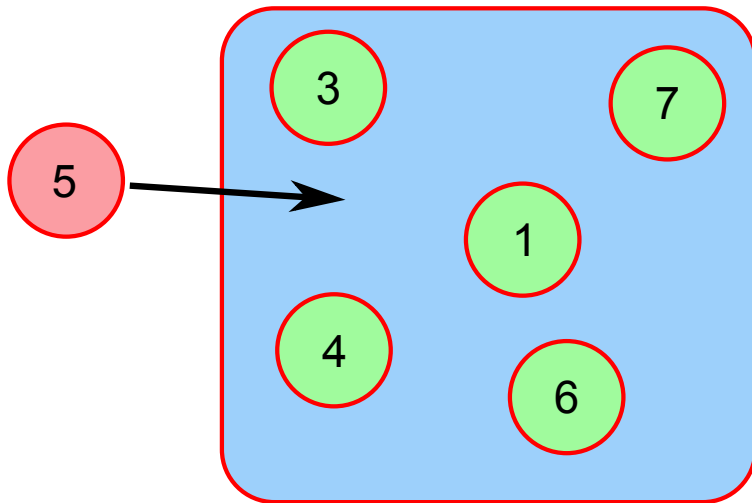
Priority queue

Insert



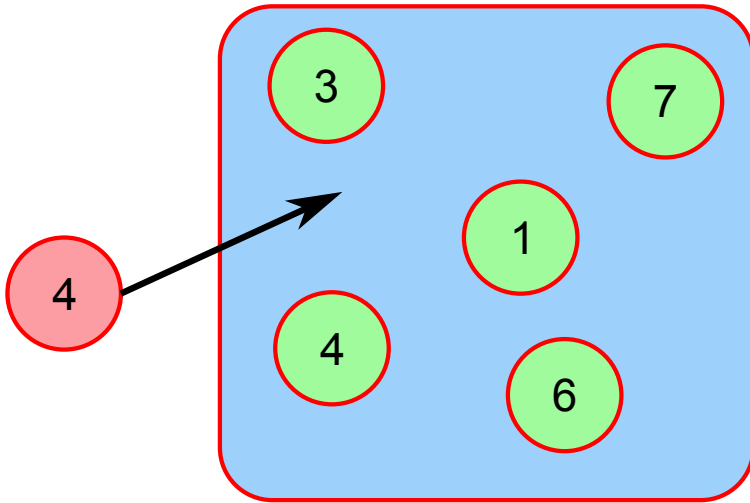
Priority queue

Insert



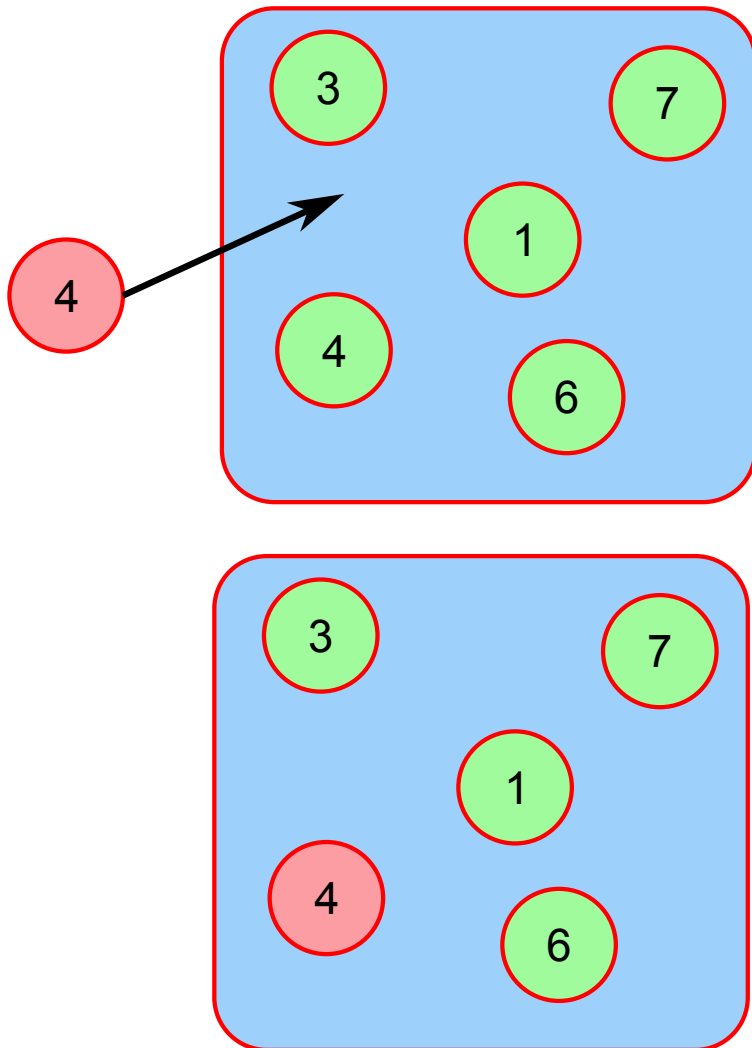
Priority queue

Insert



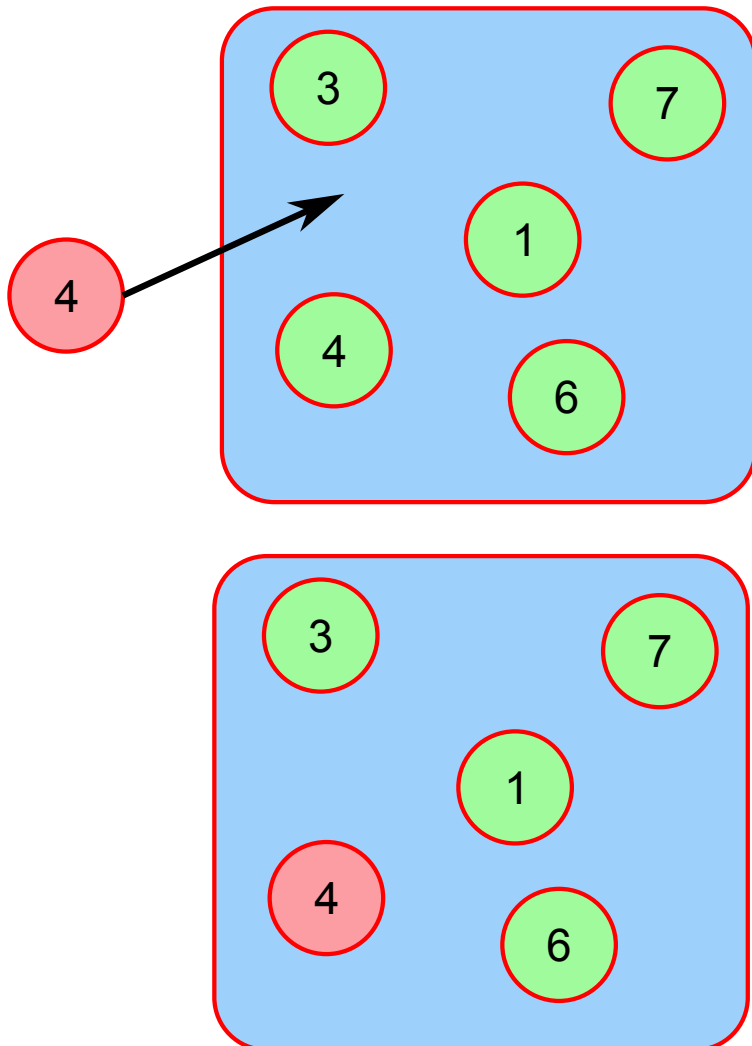
Priority queue

Insert

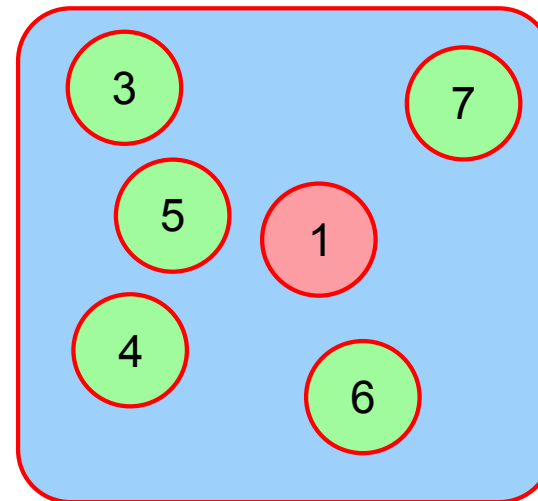


Priority queue

Insert

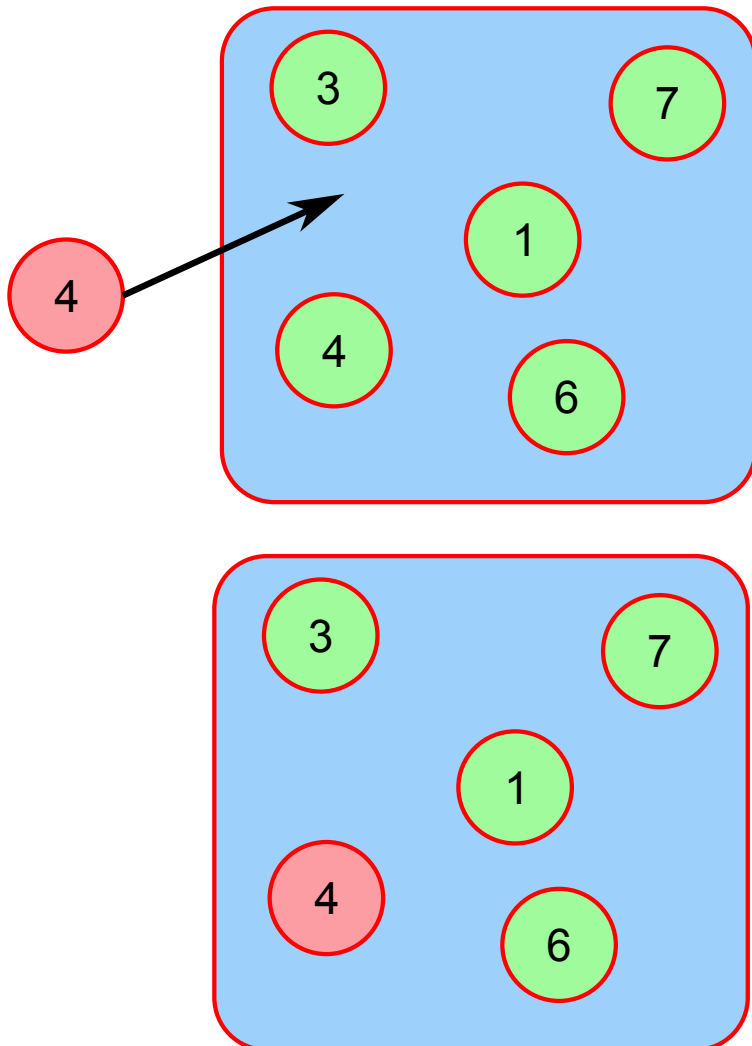


DeleteMin

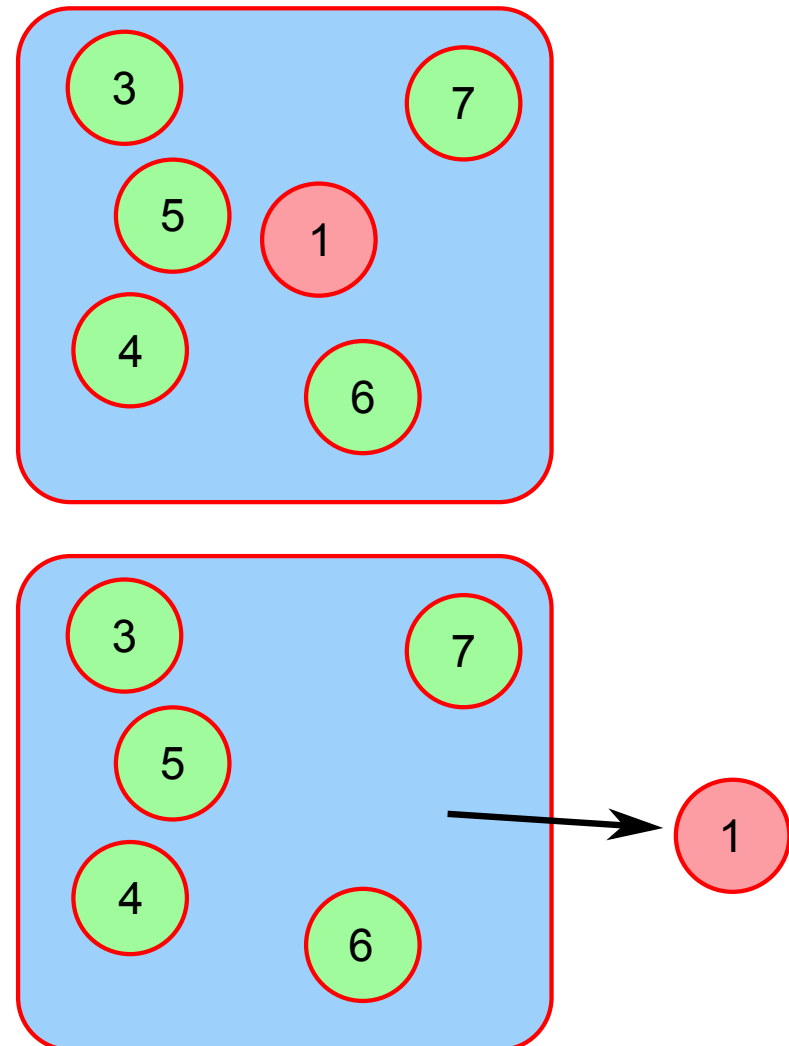


Priority queue

Insert



DeleteMin

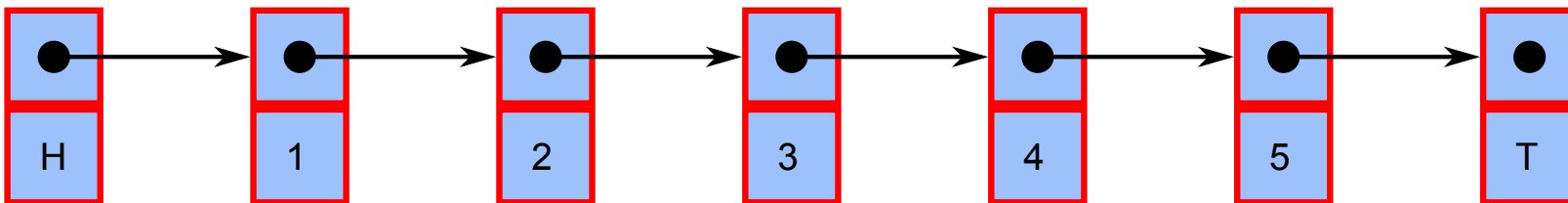


Priority Queue

- ★ Applications
 - 🌀 Scheduling
 - 🌀 Discrete Simulation
 - 🌀 Dijkstra's algorithm
 - 🌀 ...
- ★ Common implementations
 - 🌀 Sorted list
 - 🌀 Unsorted list
 - 🌀 Heap
 - 🌀 Binary search tree

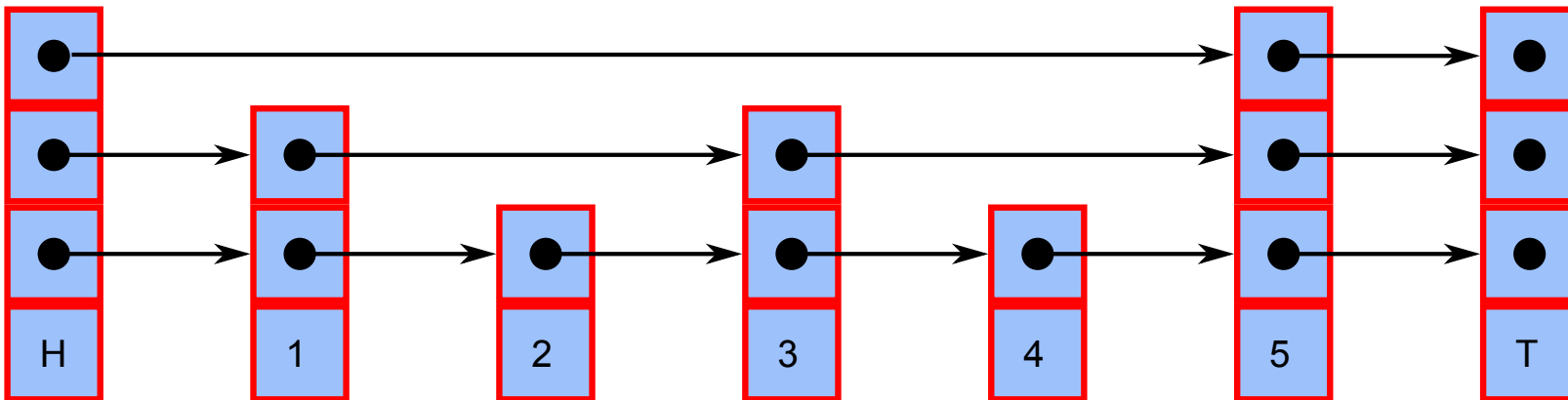
Skip list

★ Singly-linked list



Skip list

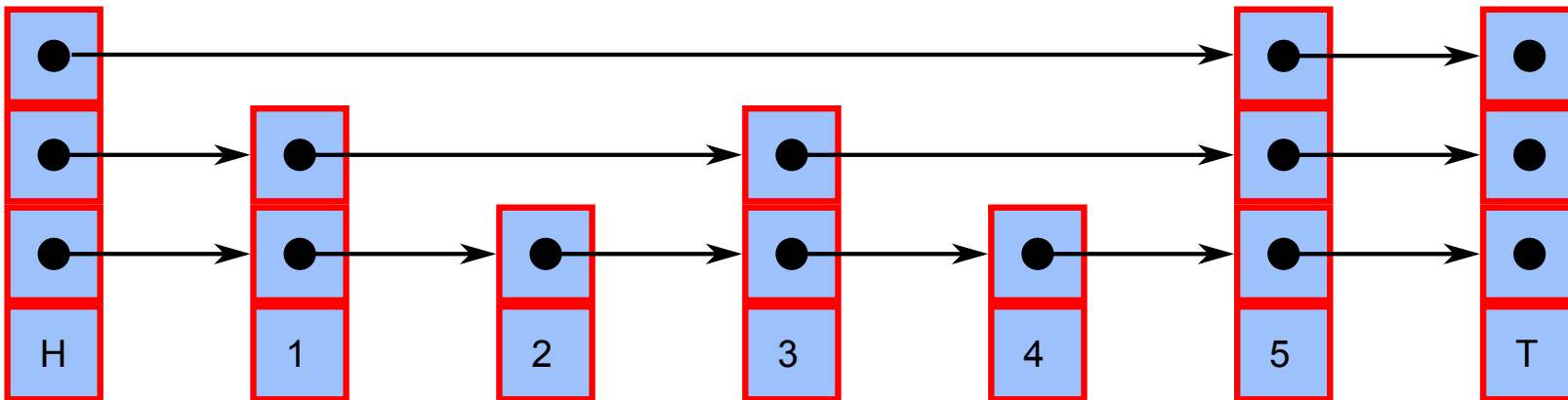
- ★ Singly-linked list
- ★ With shortcuts



Find node

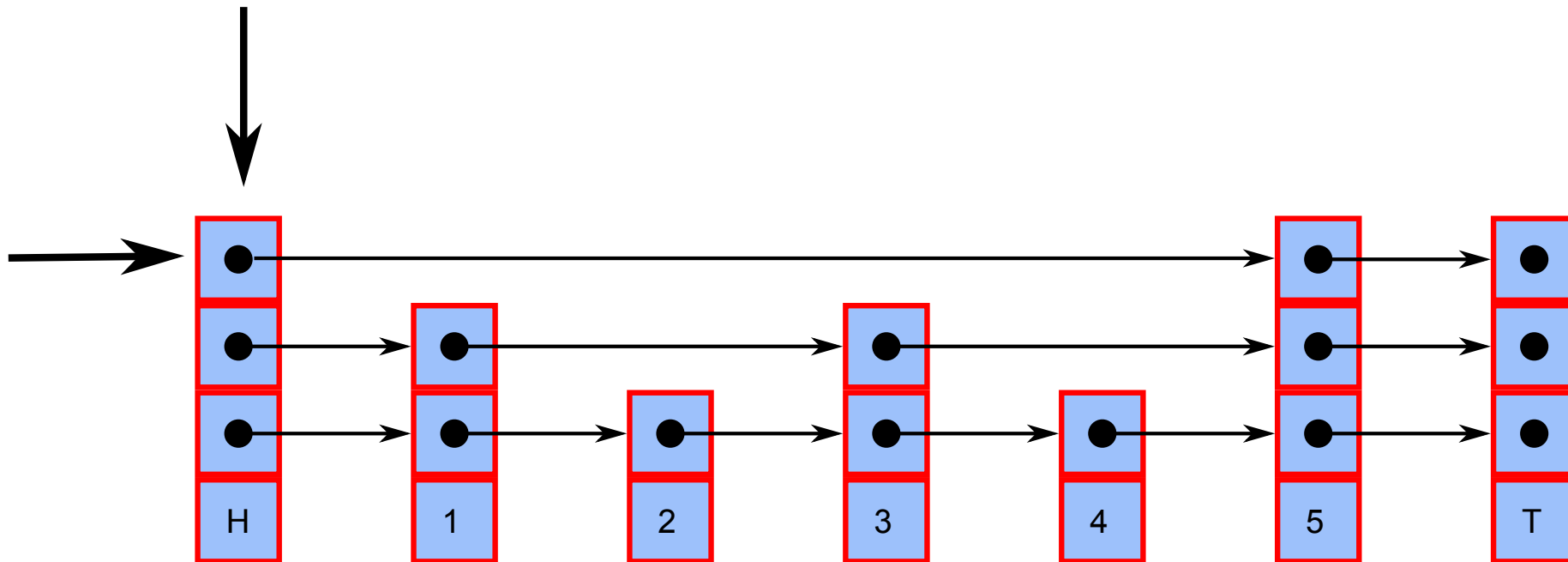
Find node

★ Find the node with key 2



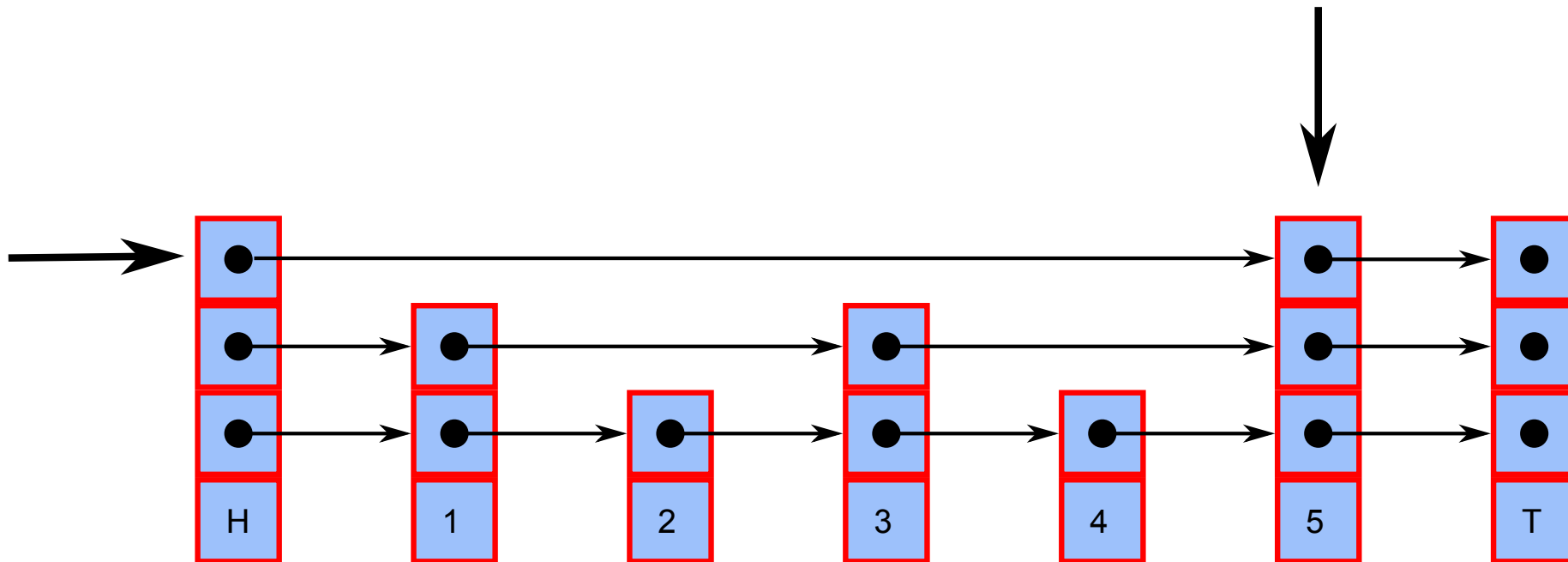
Find node

★ Find the node with key 2



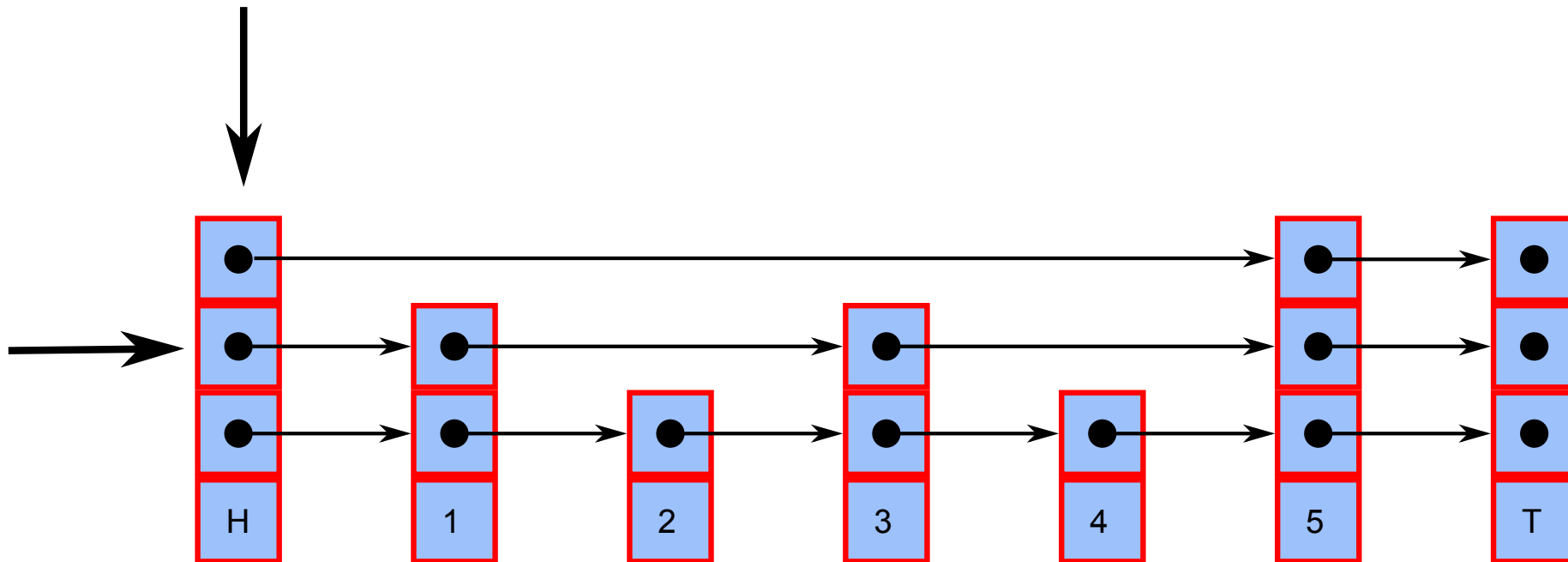
Find node

★ Find the node with key 2



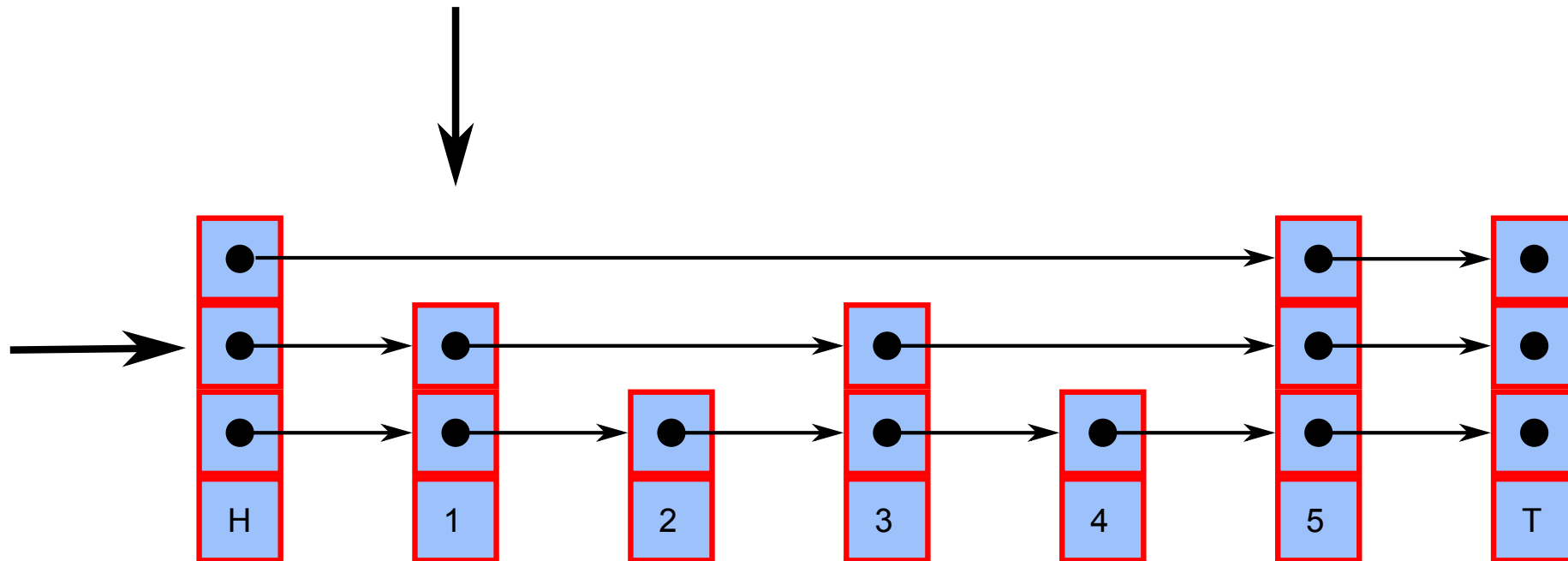
Find node

★ Find the node with key 2



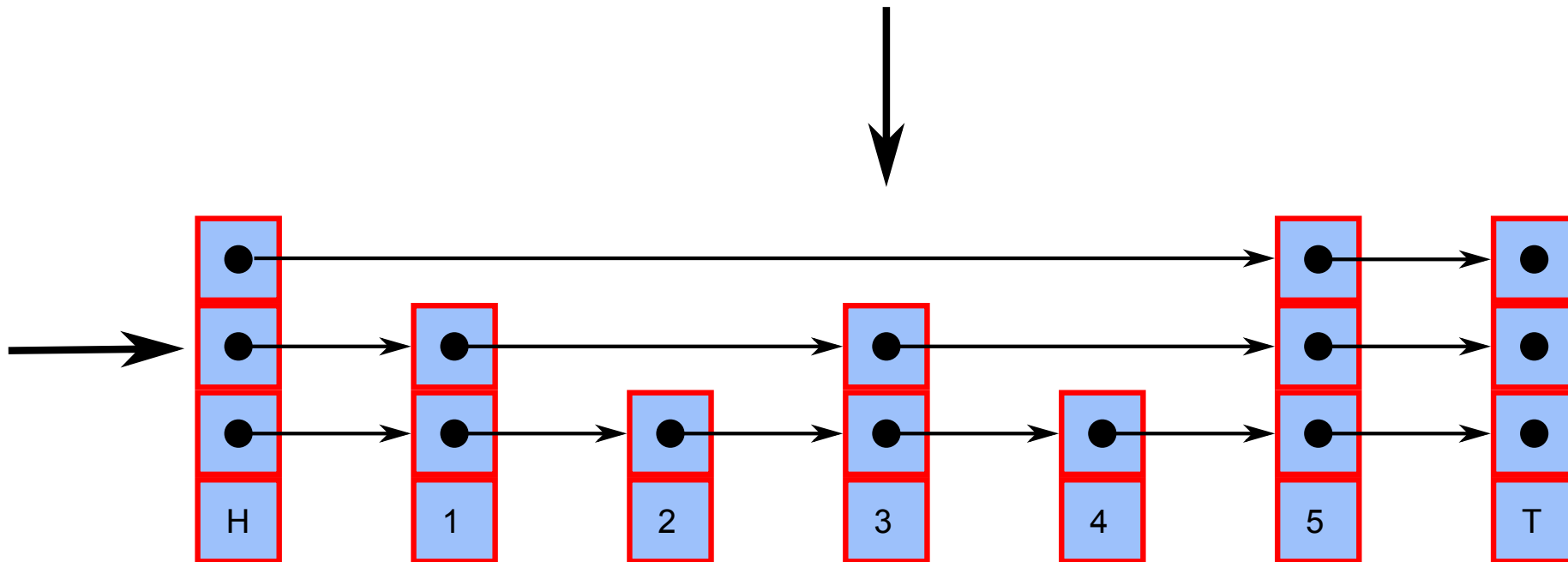
Find node

★ Find the node with key 2



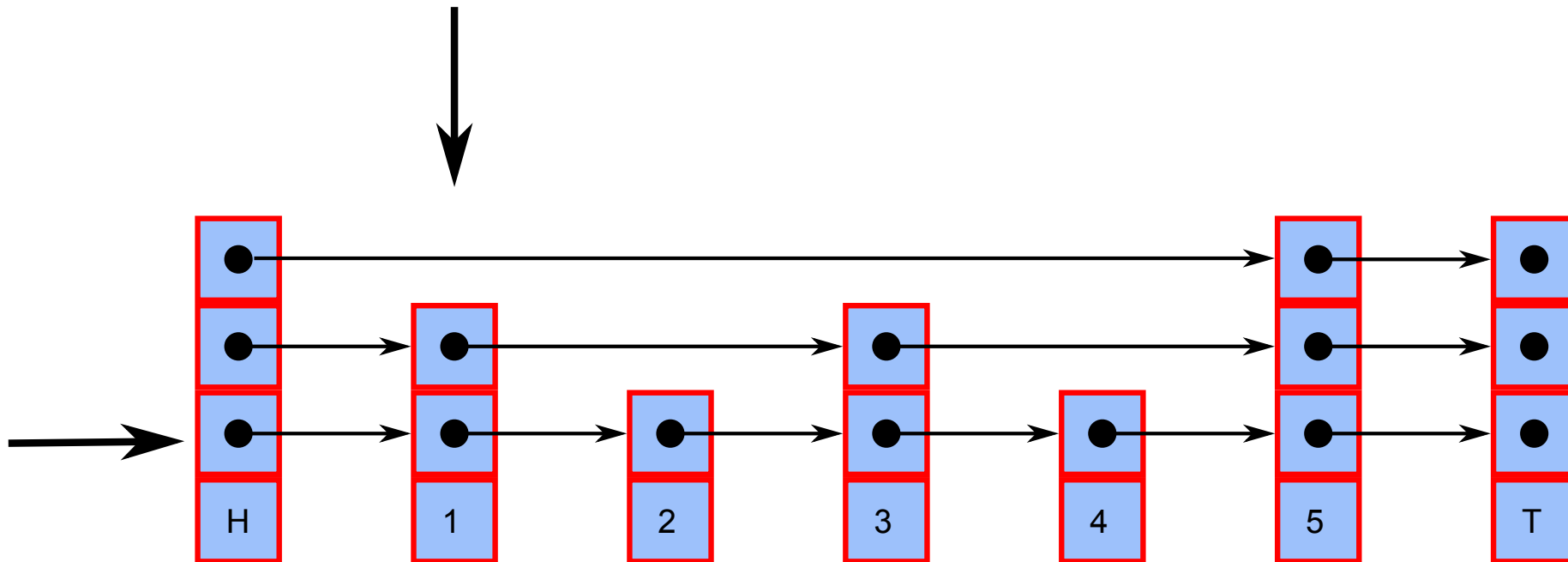
Find node

★ Find the node with key 2



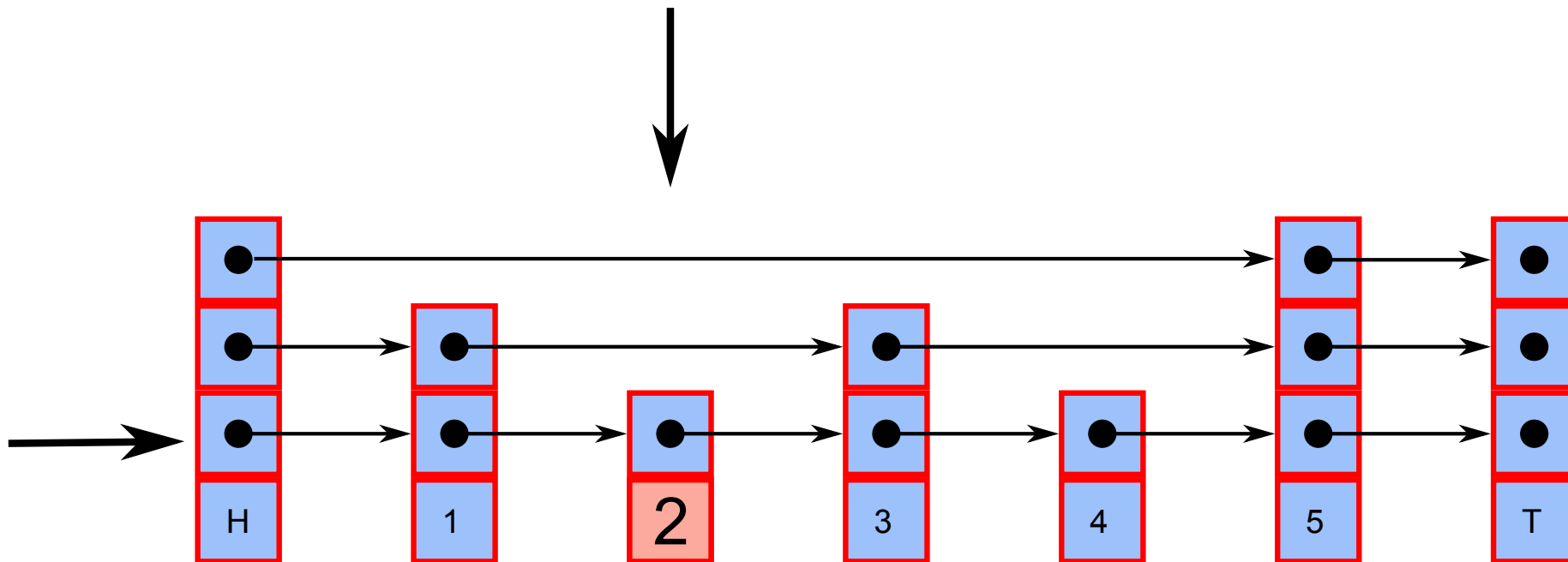
Find node

★ Find the node with key 2



Find node

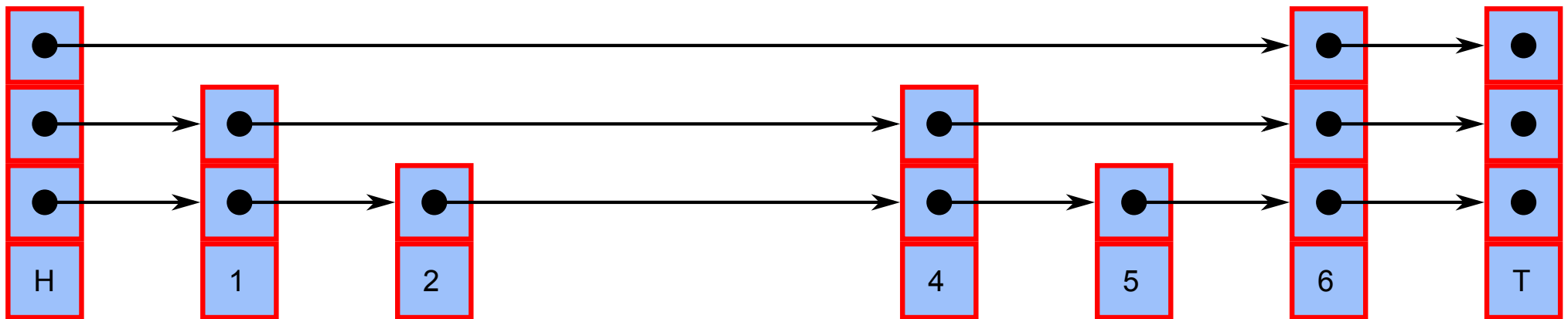
★ Find the node with key 2



Insert node

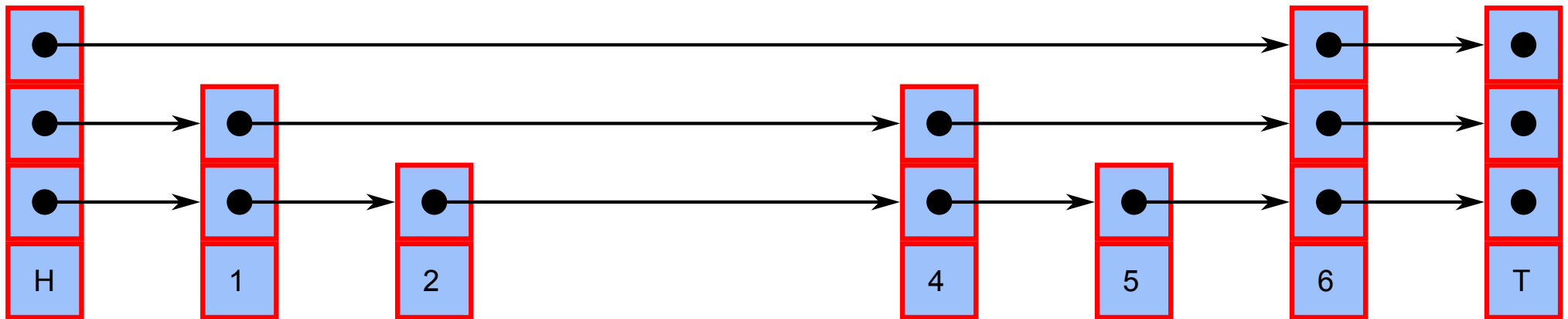
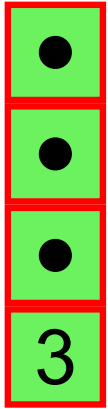
Insert node

3



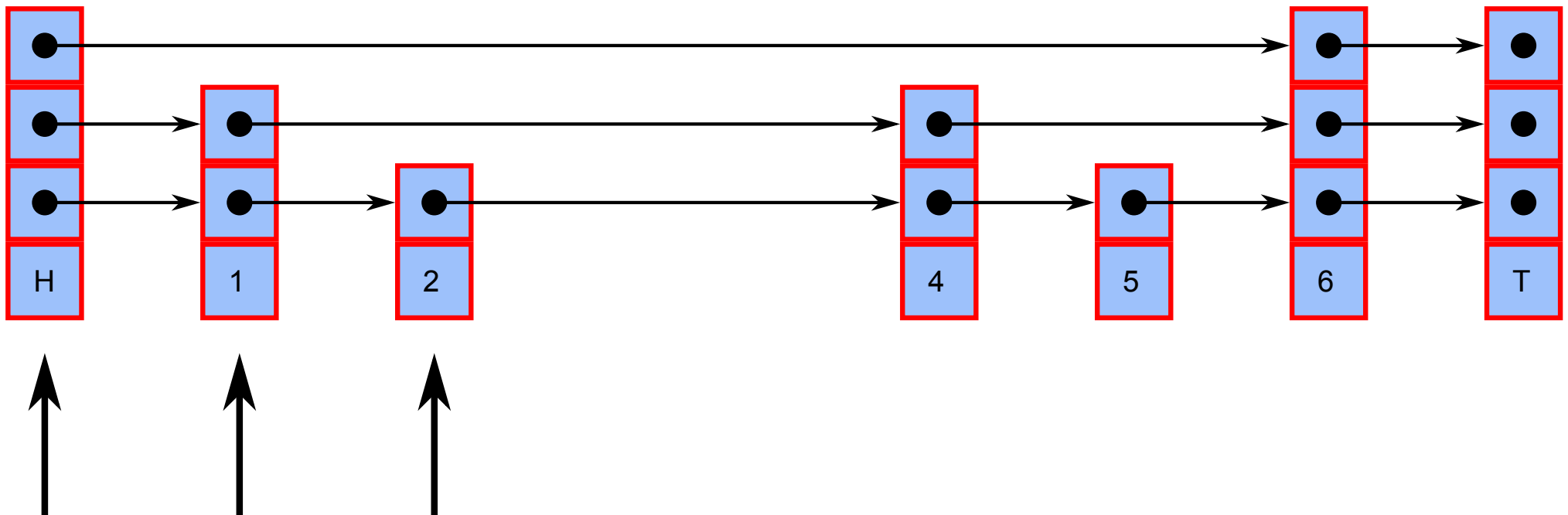
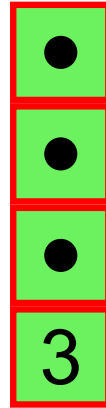
Insert node

★ Choose the level

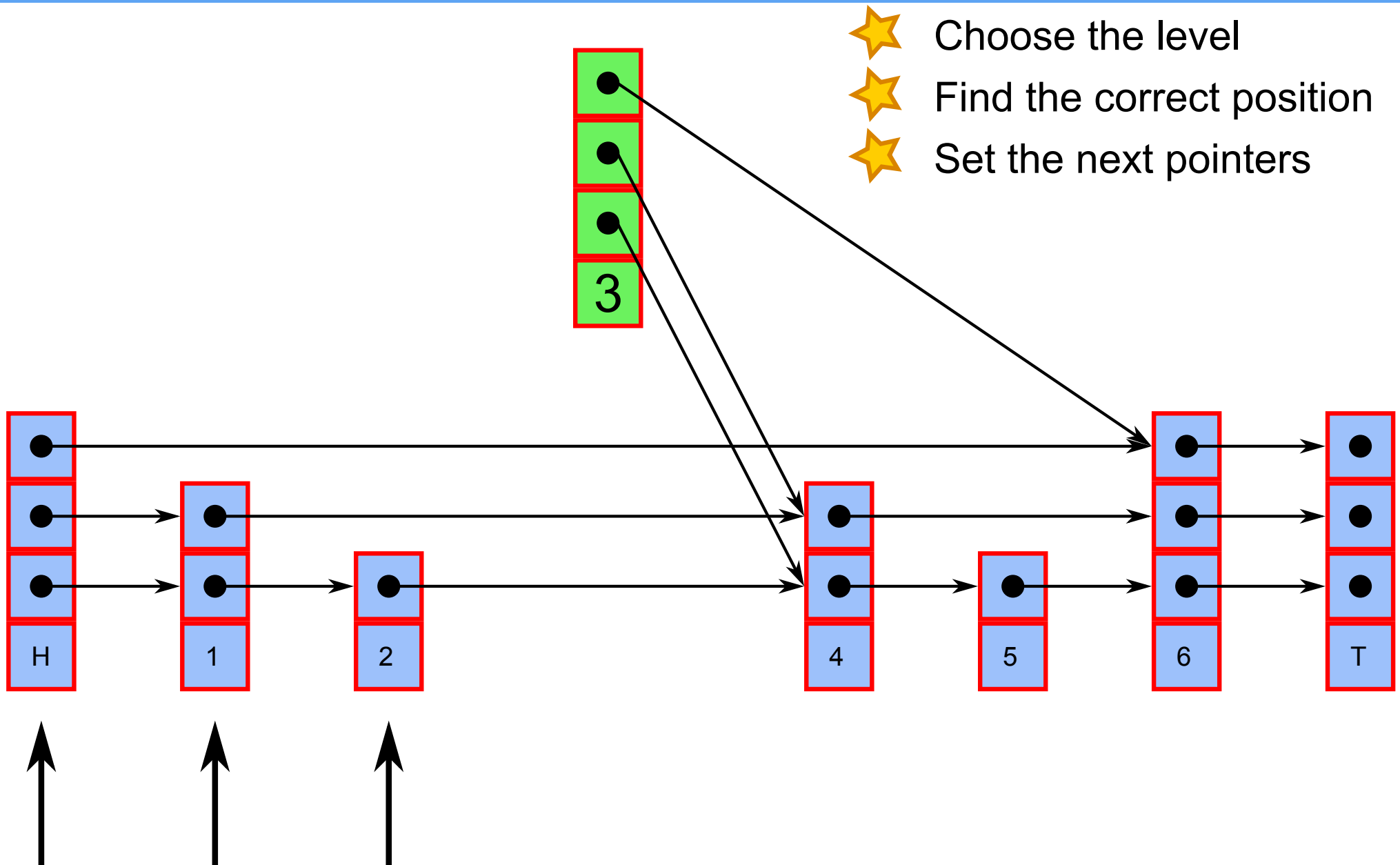


Insert node

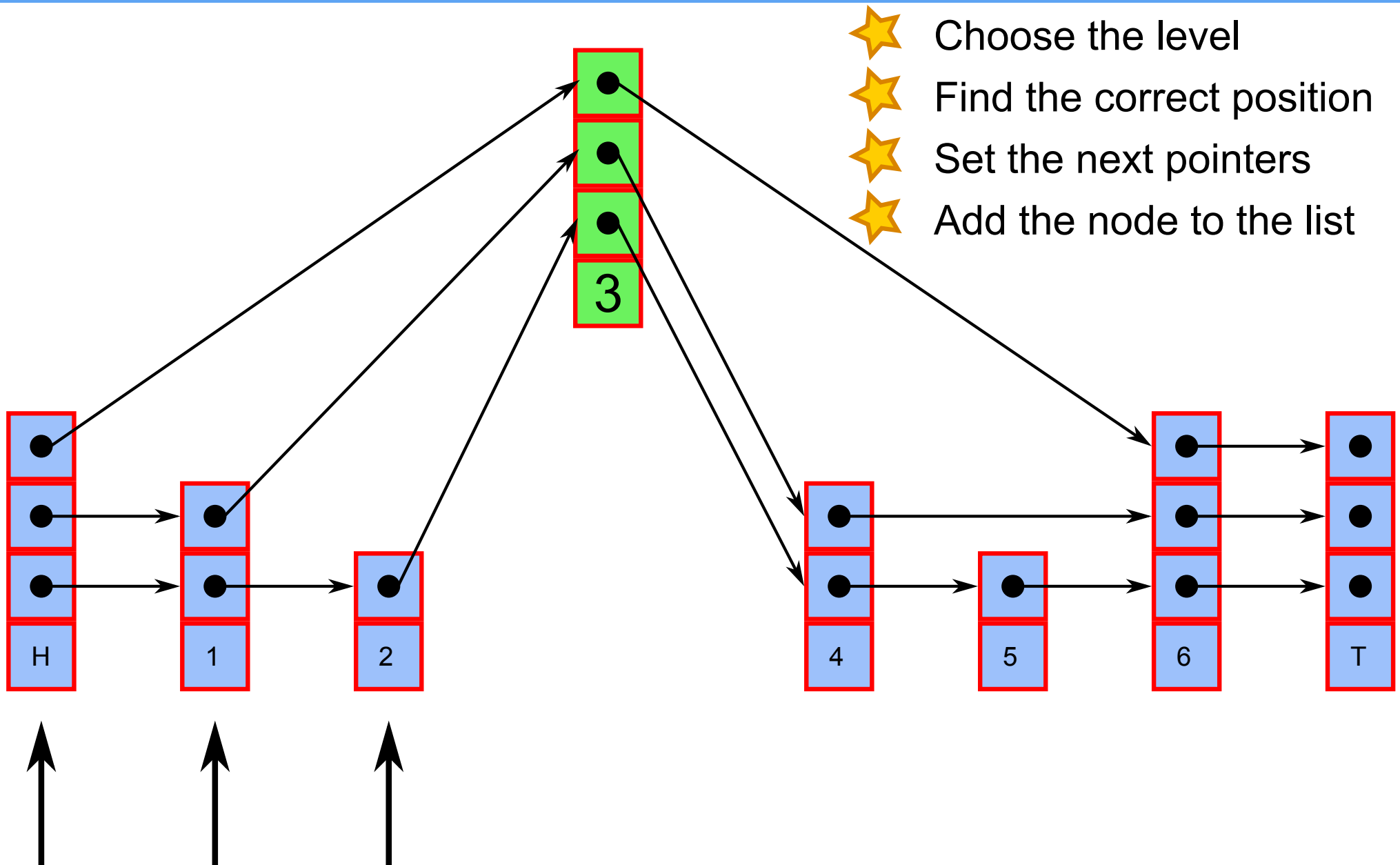
- ★ Choose the level
- ★ Find the correct position



Insert node

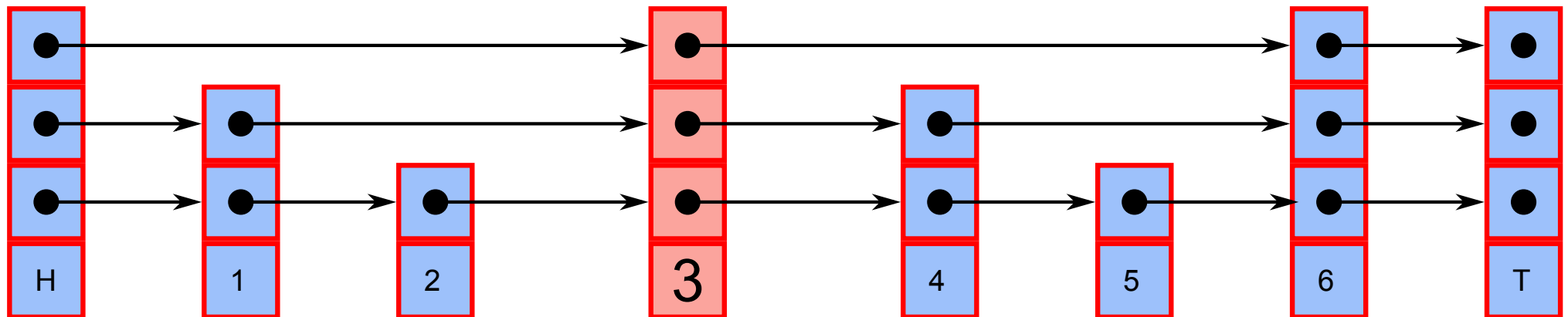


Insert node



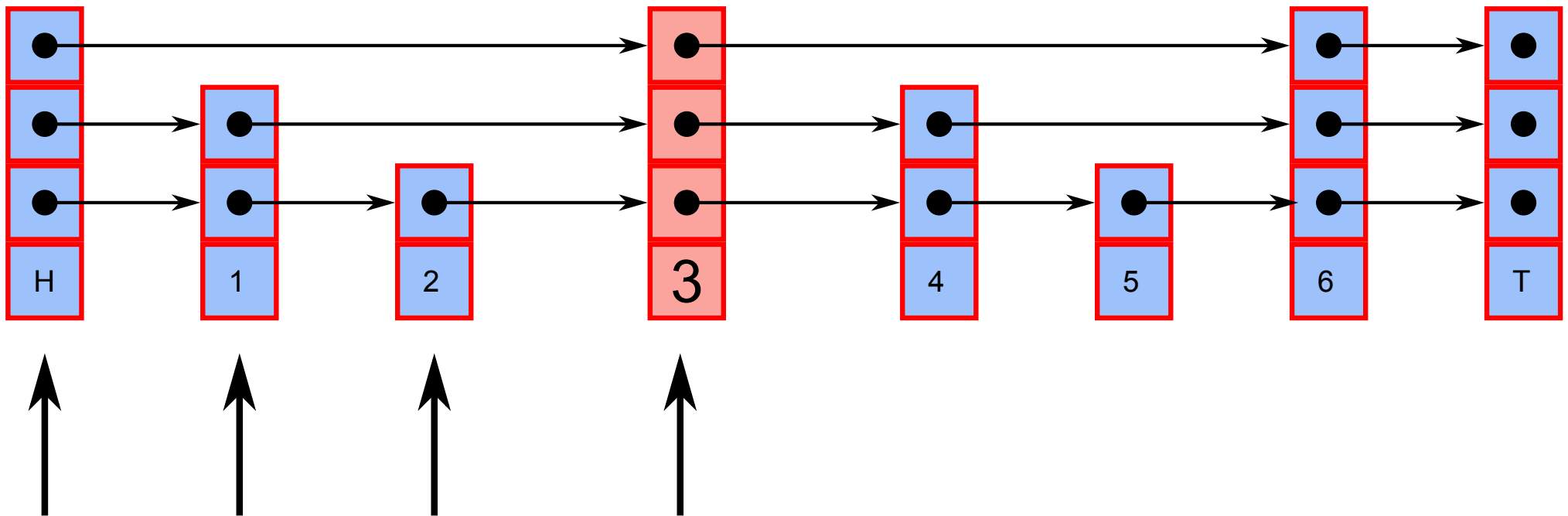
Remove node

Remove node

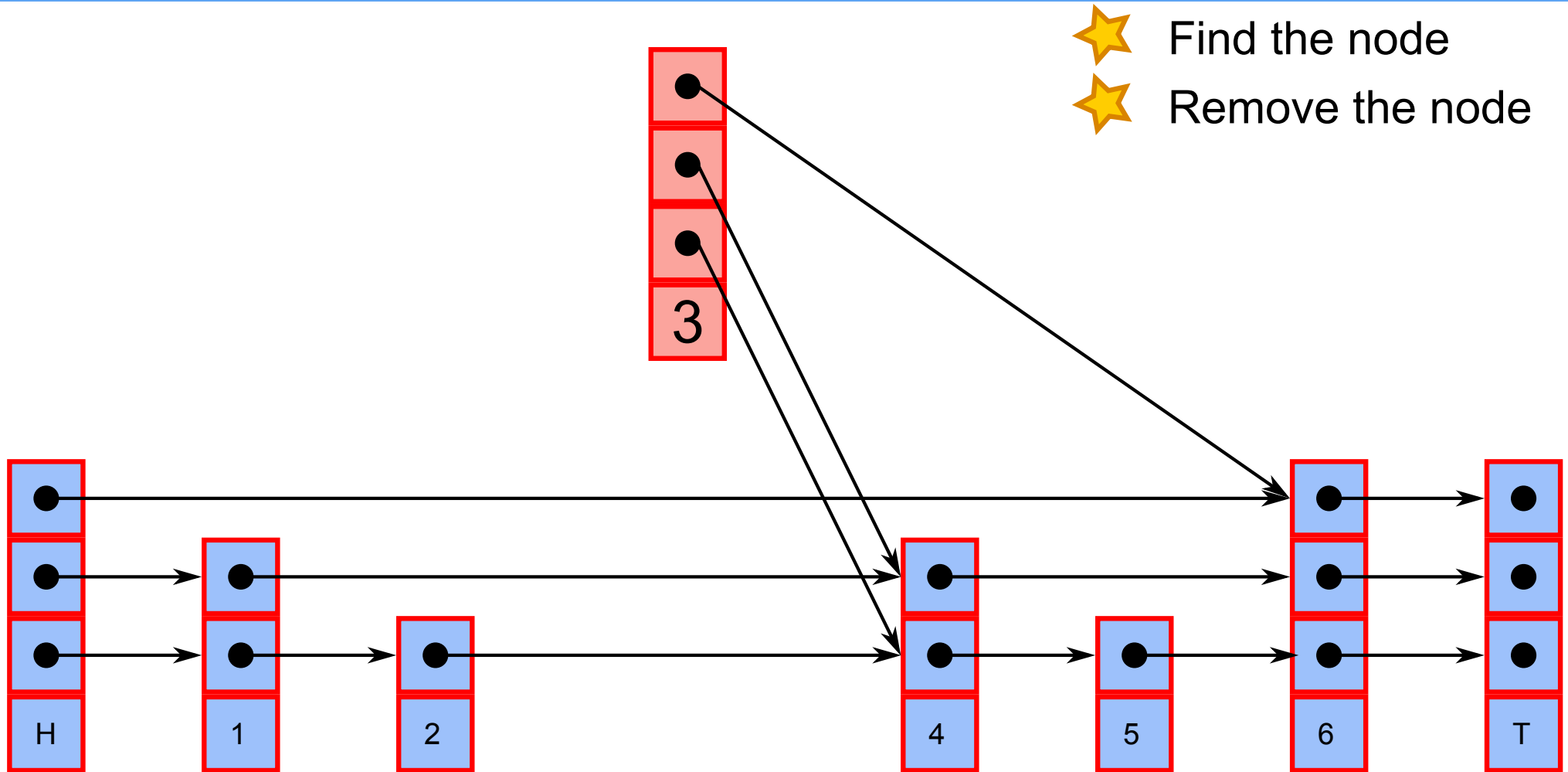


Remove node

★ Find the node



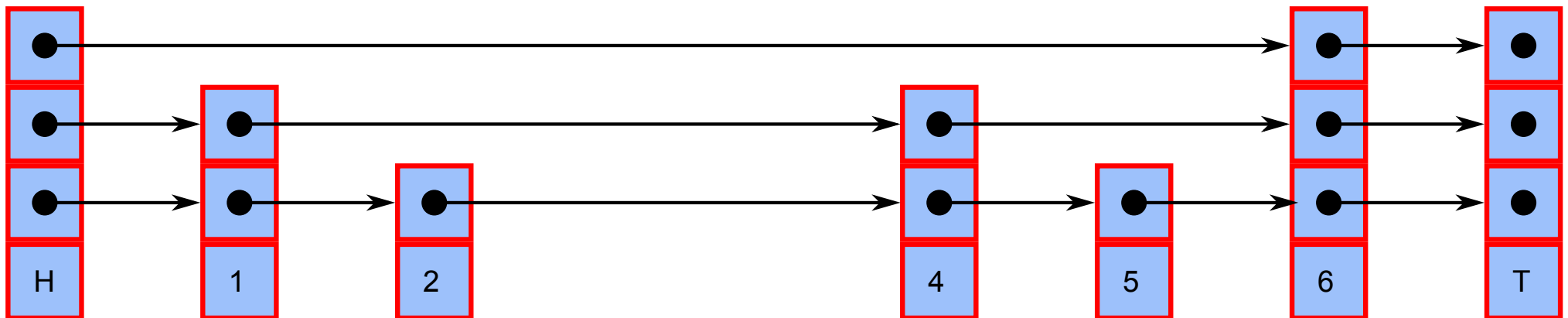
Remove node



Remove node

- ★ Find the node
- ★ Remove the node
- ★ Delete the node

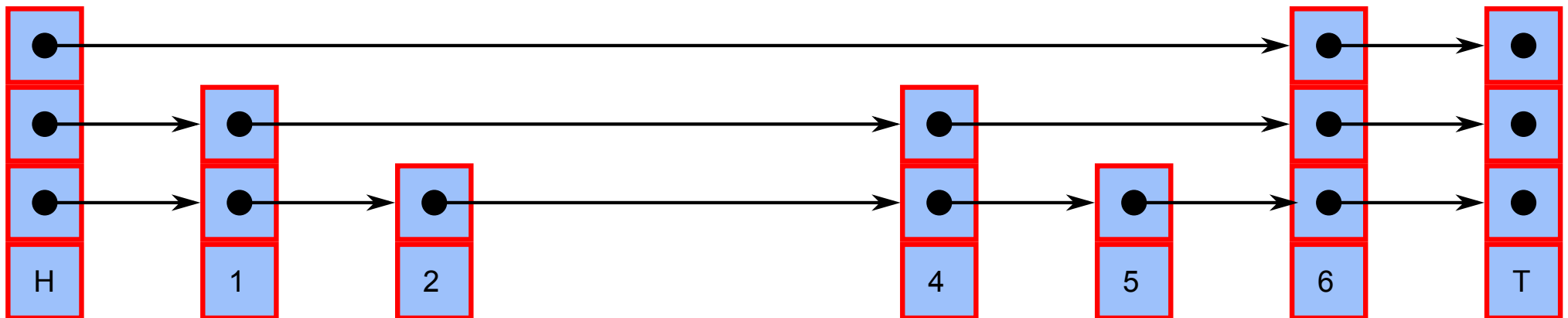
3



Remove node

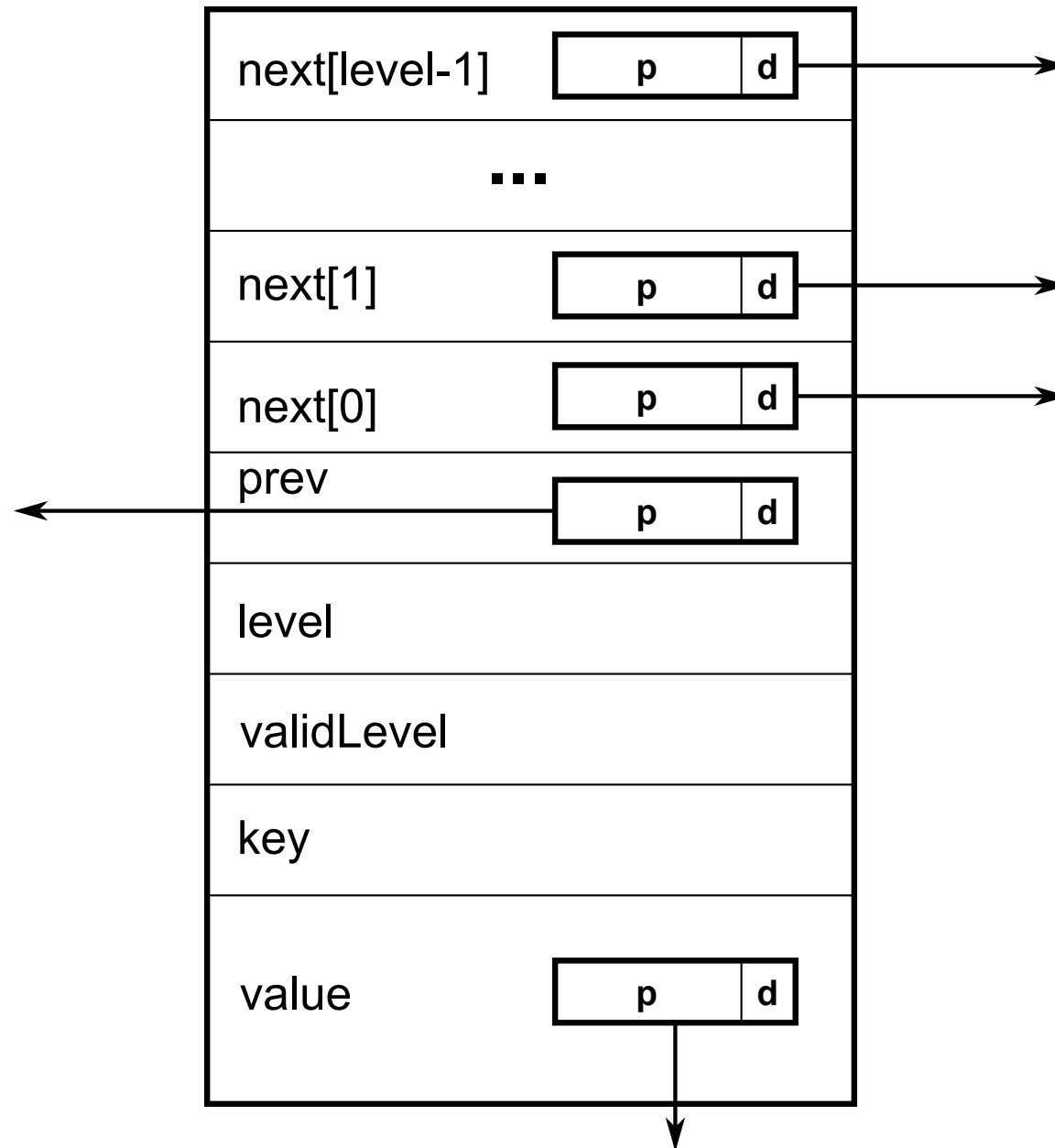
3

- ★ Find the node
- ★ Remove the node
- ★ Delete the node
- ★ Return the value



Non-blocking algorithm

Node Structure



Memory management

- ★ No node should be reclaimed and then later re-allocated while some other process is traversing that node
- 🌀 ABA Problem
- 🌀 Reference Counting is used

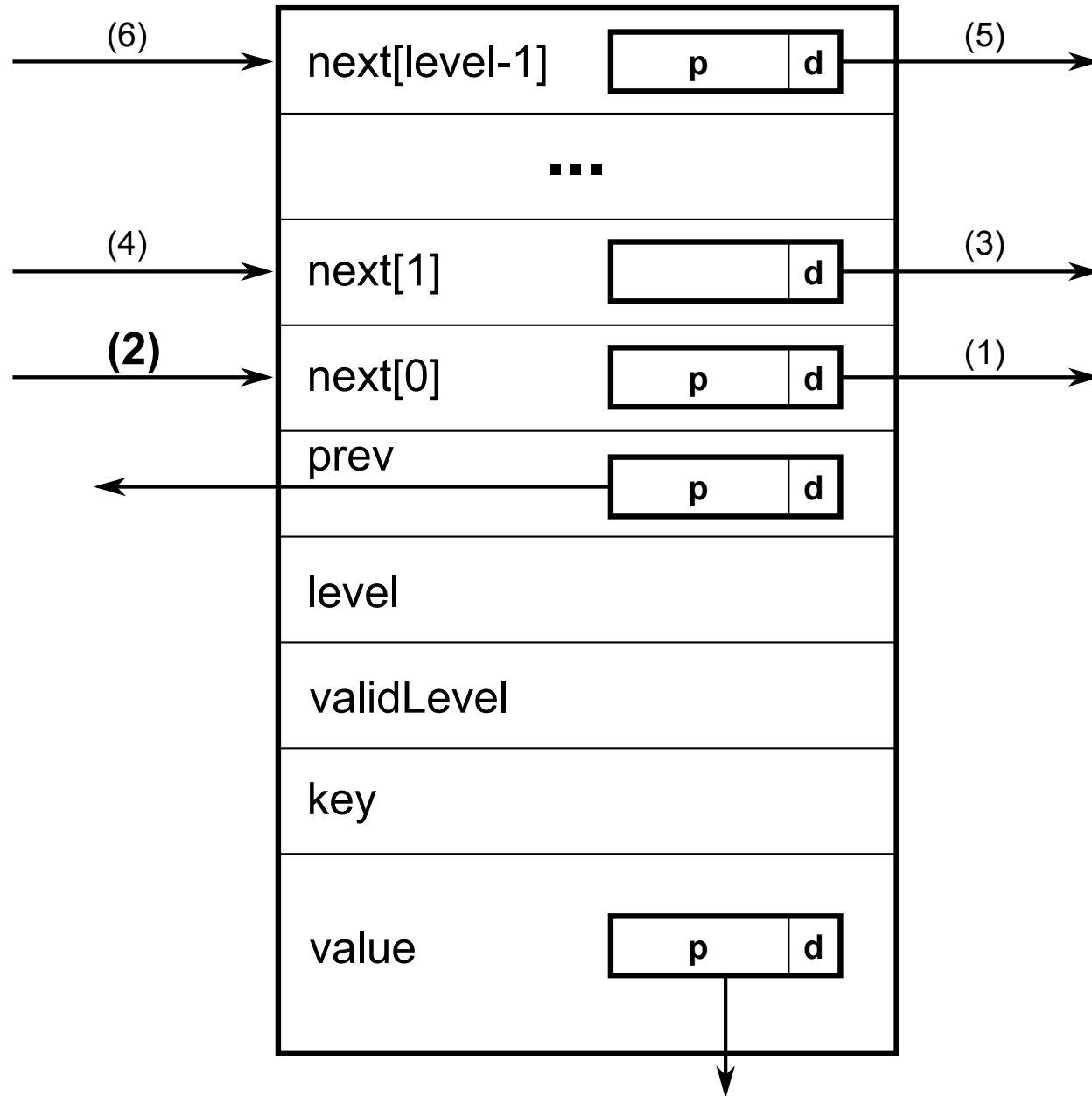
Key insight

- ★ Next pointers have to be changed consistently, but not necessarily all at once (atomically)
 - 🌀 A node is interpreted to be inserted when inserted at the lowest level
- ★ Use a deletion mark to mark a node that is about to be deleted
 - 🌀 A marked node is interpreted as deleted
 - 🌀 Every thread removes all marked nodes it encounters
 - ★ Helping strategy

Non-blocking find node

- ★ While traversing the skip list, all nodes are deleted which are flagged with the deletion mark.
- 🌀 **Helping to achieve non-blocking**

Non-blocking insert



Non-blocking Insert

Create a new node

```
I2....Choose level randomly according to the skip list
.....distribution
I3....newNode:=CreateNode(level,key,value);
I4....COPY_NODE(newNode);
```

Search insert position

```
I5....node1:=COPY_NODE(head);
I6....for i:=maxLevel-1 to 1 step -1 do
I7.....node2:=ScanKey(&node1, i, key);
I8.....RELEASE_NODE(node2);
I9.....if i < level then
I10.....savedNodes[i]:= COPY_NODE(node1);
I11...while true do
I12.....node2:=ScanKey(&node1, 0, key);
```

function Insert(key:integer, value:pointer to Value):
boolean

Change existing node

```
I12.....<value2,d> := node2.value;
I13.....if d=false and node2.key=key then
I14.....if CAS(&node2.value, <value2,false>,
.....<value, false>)
.....then
I15.....RELEASE_NODE(node1);
I16.....RELEASE_NODE(node2);
I17.....for i:=1 to level-1 do
I18.....RELEASE_NODE(savedNodes[i]);
I19.....RELEASE_NODE(newNode);
I20.....RELEASE_NODE(newNode);
I21.....return true2;
I22.....else
I23.....RELEASE_NODE(node2);
I24.....continue;
```

Insert new node

```
I25.....newNode.next[0]:= <node2, false>; (1)
I26.....RELEASE_NODE(node2); (2)
I27.....if CAS(&node1.next[0],<node2,false>,
.....<newNode,false>)
.....then
I28.....RELEASE_NODE(node1);
I29.....break;
I30.....Back-Off
I31..for i:=1 to level-1 do (3)-(6)
I32.....newNode.validLevel:=i;
I33.....node1:=savedNodes[i];
I34.....while true do
I35.....node2:=ScanKey(&node1,i,key);
I36.....newNode.next[i]:= node2,false ;
I37.....RELEASE_NODE(node2);
I38.....if newNode.value.d=true or
.....CAS(&node1.next[i],node2,newNode) then
I39.....RELEASE_NODE(node1);
I40.....break;
I41.....Back-Off
I42..newNode.validLevel:=level;
```

Delete node

```
I38.....if newNode.value.d=true or
.....CAS(&node1.next[i],node2,newNode) then
I39.....RELEASE_NODE(node1);
I40.....break;
I41.....Back-Off
I42..newNode.validLevel:=level;

I44..if newNode.value.d=true then
I45.....newNode:= HelpDelete(newNode,0);
I46..RELEASE_NODE(newNode);
I47..return true;
```

Create a new node

12....Choose level randomly according to the skip list

.....distribution

13....newNode:=CreateNode(level,key,value);

14....COPY_NODE(newNode);

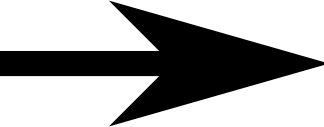
Search insert position

```
l5....node1:=COPY_NODE(head);
l6....for i:=maxLevel-1 to 1 step -1 do
l7.....node2:=ScanKey(&node1, i, key);
l8.....RELEASE_NODE(node2);
l9.....if i < level then
l10.....savedNodes[i]:= COPY_NODE(node1);
l11..while true do
l12.....node2:=ScanKey(&node1, 0, key);
```

Change existing node

```
I12.....<value2,d> := node2.value;
I13.....if d=false and node2.key=key then
I14.....if CAS(&node2.value, <value2,false>,
.....<value, false>)
.....then
I15.....RELEASE_NODE(node1);
I16.....RELEASE_NODE(node2);
I17.....for i:=1 to level-1 do
I18.....RELEASE_NODE(savedNodes[i]);
I19.....RELEASE_NODE(newNode);
I20.....RELEASE_NODE(newNode);
I21.....return true2;
I22.....else
I23.....RELEASE_NODE(node2);
I24.....continue;
```

Insert new node



```
l25.....newNode.next[0]:= <node2, false>;           (1)
l26.....RELEASE_NODE(node2);
l27.....if CAS(&node1.next[0],<node2,false>,           (2)
.....<newNode,false>)
.....then
l28.....RELEASE_NODE(node1);
l29.....break;
l30.....Back-Off
l31..for i:=1 to level-1 do                           (3)-(6)
l32.....newNode.validLevel:=i;
l33.....node1:=savedNodes[i];
l34.....while true do
l35.....node2:=ScanKey(&node1,i,key);
l36.....newNode.next[i]:= node2,false ;
l37.....RELEASE_NODE(node2);
l38.....if newNode.value.d=true or
.....CAS(&node1.next[i],node2,newNode) then
l39.....RELEASE_NODE(node1);
l40.....break;
l41.....Back-Off
l42..newNode.validLevel:=level;
```

Delete node

```
l38.....if newNode.value.d=true or
.....CAS(&node1.next[i],node2,newNode) then
l39.....RELEASE_NODE(node1);
l40.....break;
l41.....Back-Off
l42..newNode.validLevel:=level;

l44..if newNode.value.d=true then
l45.....newNode:= HelpDelete(newNode,0);
l46..RELEASE_NODE(newNode);
l47..return true;
```

Non-blocking Insert

Create a new node

```
I2....Choose level randomly according to the skip list
.....distribution
I3....newNode:=CreateNode(level,key,value);
I4....COPY_NODE(newNode);
```

Search insert position

```
I5....node1:=COPY_NODE(head);
I6....for i:=maxLevel-1 to 1 step -1 do
I7.....node2:=ScanKey(&node1, i, key);
I8.....RELEASE_NODE(node2);
I9.....if i < level then
I10.....savedNodes[i]:= COPY_NODE(node1);
I11...while true do
I12.....node2:=ScanKey(&node1, 0, key);
```

function Insert(key:integer, value:pointer to Value):
boolean

Change existing node

```
I12.....<value2,d> := node2.value;
I13.....if d=false and node2.key=key then
I14.....if CAS(&node2.value, <value2,false>,
.....<value, false>)
.....then
I15.....RELEASE_NODE(node1);
I16.....RELEASE_NODE(node2);
I17.....for i:=1 to level-1 do
I18.....RELEASE_NODE(savedNodes[i]);
I19.....RELEASE_NODE(newNode);
I20.....RELEASE_NODE(newNode);
I21.....return true2;
I22.....else
I23.....RELEASE_NODE(node2);
I24.....continue;
```

Insert new node

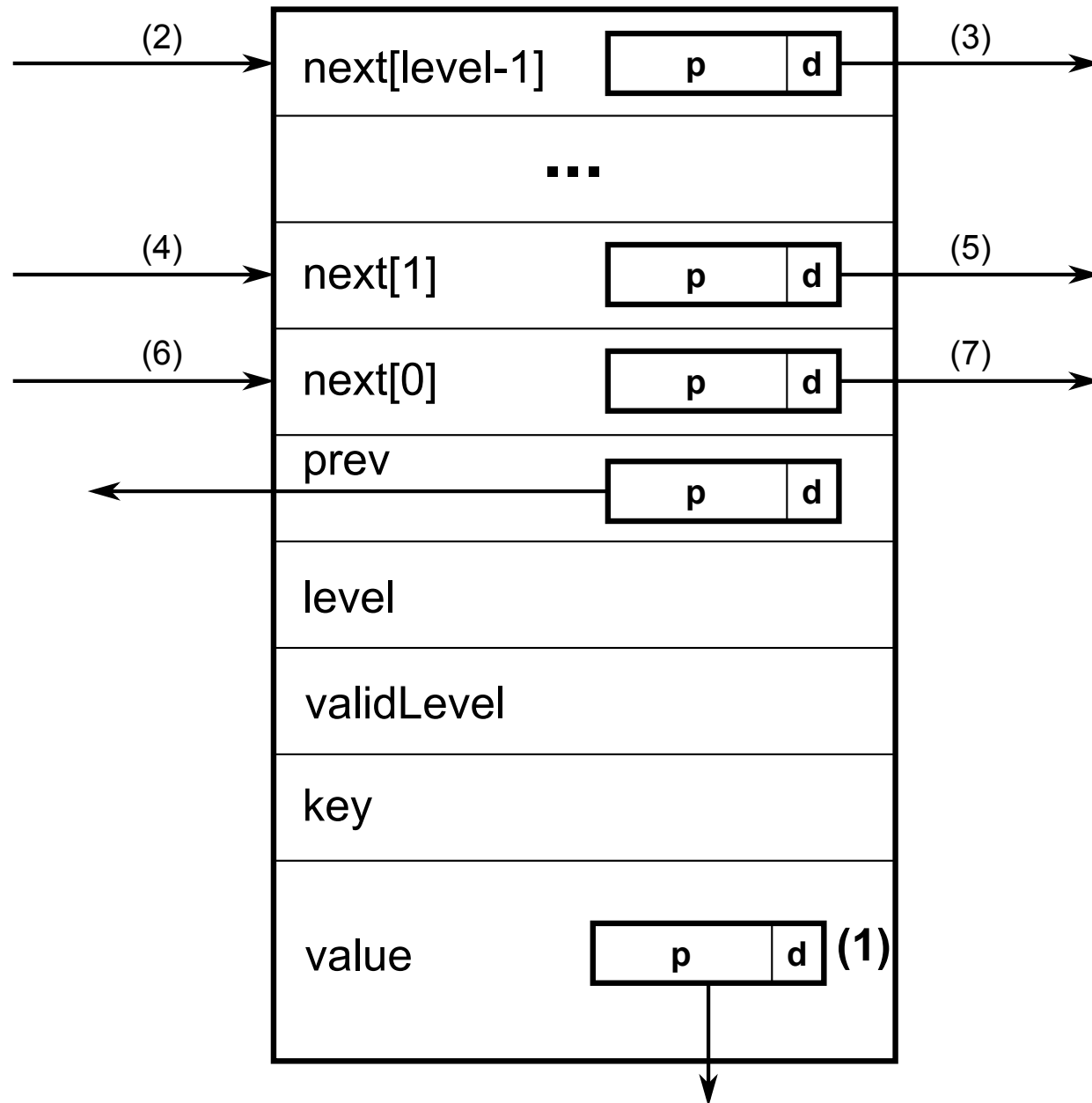
```
I25.....newNode.next[0]:= <node2, false>; (1)
I26.....RELEASE_NODE(node2); (2)
I27.....if CAS(&node1.next[0],<node2,false>,
.....<newNode,false>)
.....then
I28.....RELEASE_NODE(node1);
I29.....break;
I30.....Back-Off
I31..for i:=1 to level-1 do (3)-(6)
I32.....newNode.validLevel:=i;
I33.....node1:=savedNodes[i];
I34.....while true do
I35.....node2:=ScanKey(&node1,i,key);
I36.....newNode.next[i]:= node2,false ;
I37.....RELEASE_NODE(node2);
I38.....if newNode.value.d=true or
.....CAS(&node1.next[i],node2,newNode) then
I39.....RELEASE_NODE(node1);
I40.....break;
I41.....Back-Off
I42..newNode.validLevel:=level;
```

Delete node

```
I38.....if newNode.value.d=true or
.....CAS(&node1.next[i],node2,newNode) then
I39.....RELEASE_NODE(node1);
I40.....break;
I41.....Back-Off
I42..newNode.validLevel:=level;

I44..if newNode.value.d=true then
I45.....newNode:= HelpDelete(newNode,0);
I46..RELEASE_NODE(newNode);
I47..return true;
```

Non-blocking DeleteMin



Non-blocking DeleteMin

Find the first node

```
D3...prev:=COPY_NODE(head);
D4...while true do
D5.....node1:=ReadNext(&prev,0);
D6.....if node1=tail then
D7.....RELEASE_NODE(prev);
D8.....RELEASE_NODE(node1);
D9.....return NULL;
.....retry:
D10..... value,d :=node1.value;
D11.....if node1 != prev.next[0].p then
D12.....RELEASE_NODE(node1);
D13.....continue;
D14.....if d=false then
D15.....if CAS(&node1.value,<value,false>, <value,true>) then (1)
D16.....node1.prev:=prev;
D17.....break;
D18.....else goto retry;
D19.....else d=true then
D20.....node1:=HelpDelete(node1,0);
D21.....RELEASE_NODE(prev);
D22.....prev:=node1;
```

function DeleteMin():
pointer to Node

Set the deletion mark

```
D15.....if CAS(&node1.value, <value,false>, <value,true>) then (1)
D16.....node1.prev:=prev;
D17.....break;
D18.....else goto retry;
```

Return value

D33..return value;

Remove node

```
D23..for i:=0 to node1.level-1 do
D24.....repeat
D25..... node2,d :=node1.next[i];
D26.....until d= true or CAS(&node1.next[i], node2,false ,
..... node2,true );
```


Delete node

```
D27..prev:=COPY_NODE(head);
D28..for i:=node1.level-1 to 0 step -1 do
D29.....RemoveNode(node1,&prev,i);
D30..RELEASE_NODE(prev);
D31..RELEASE_NODE(node1);
D32..RELEASE_NODE(node1); /* Delete the node */
```

(2)-(7)

Find the first node

```
D3....prev:=COPY_ NODE(head);
D4....while true do
D5.....node1:=ReadNext(&prev,0);
D6.....if node1=tail then
D7.....RELEASE_ NODE(prev);
D8.....RELEASE_ NODE(node1);
D9.....return NULL;
.....retry:
D10..... value,d :=node1.value;
D11.....if node1 != prev.next[0].p then
D12.....RELEASE_ NODE(node1);
D13.....continue;
D14.....if d=false then
D15.....if CAS(&node1.value,<value,false>, <value,true>) then (1)
D16.....node1.prev:=prev;
D17.....break;
D18.....else goto retry;
D19.....else d=true then
D20.....node1:=HelpDelete(node1,0);
D21.....RELEASE_ NODE(prev);
D22.....prev:=node1;
```



Set the deletion mark

```
D15.....if CAS(&node1.value, <value,false>, <value,true>) then (1)
D16.....node1.prev:=prev;
D17.....break;
D18.....else goto retry;
```

Remove node

```
D23..for i:=0 to node1.level-1 do
D24.....repeat
D25.....    node2,d :=node1.next[i];
D26.....until d= true or CAS(&node1.next[i],    node2,false  ,
.....                               node2,true  );
```

Delete node

```
D27..prev:=COPY_NODE(head);  
D28..for i:=node1.level-1 to 0 step -1 do (2)-(7)  
D29.....RemoveNode(node1,&prev,i);  
D30..RELEASE_NODE(prev);  
D31..RELEASE_NODE(node1);  
D32..RELEASE_NODE(node1); /* Delete the node */
```

Return value

D33..return value;



Non-blocking DeleteMin

Find the first node

```
D3...prev:=COPY_NODE(head);
D4...while true do
D5.....node1:=ReadNext(&prev,0);
D6.....if node1=tail then
D7.....RELEASE_NODE(prev);
D8.....RELEASE_NODE(node1);
D9.....return NULL;
.....retry:
D10..... value,d :=node1.value;
D11.....if node1 != prev.next[0].p then
D12.....RELEASE_NODE(node1);
D13.....continue;
D14.....if d=false then
D15.....if CAS(&node1.value,<value,false>, <value,true>) then (1)
D16.....node1.prev:=prev;
D17.....break;
D18.....else goto retry;
D19.....else d=true then
D20.....node1:=HelpDelete(node1,0);
D21.....RELEASE_NODE(prev);
D22.....prev:=node1;
```

function DeleteMin():
pointer to Node

Set the deletion mark

```
D15.....if CAS(&node1.value, <value,false>, <value,true>) then (1)
D16.....node1.prev:=prev;
D17.....break;
D18.....else goto retry;
```

Return value

D33..return value;

Remove node

```
D23..for i:=0 to node1.level-1 do
D24.....repeat
D25..... node2,d :=node1.next[i];
D26.....until d= true or CAS(&node1.next[i], node2,false ,
..... node2,true );
```

Delete node

```
D27..prev:=COPY_NODE(head);
D28..for i:=node1.level-1 to 0 step -1 do
D29.....RemoveNode(node1,&prev,i);
D30..RELEASE_NODE(prev);
D31..RELEASE_NODE(node1);
D32..RELEASE_NODE(node1); /* Delete the node */
```

(2)-(7)

Correctness

- ★ Changes to the skip list are done in a single CAS instruction
- 🌀 Insert: The next pointer pointing to the new node
- 🌀 DeleteMin: The deletion mark of the node with the minimal key is set
- ★ The data structure is always in a consistent state
- ★ Linearizable since changes occur atomically

Conclusion

- ★ Priority Queue based on a sorted linked list
 - 🌀 Use shortcuts (skip list) as optimization
 - ★ The shortcuts are not part of the consistent state of the priority queue
- ★ Delayed deletion using deletion marks
- ★ Helping for non-blocking
- ★ Correctness is achieved by changing the consistent state of all threads by single CAS operations

Thank you for your attention

★ Questions?

- ★ Håkan Sundell, Philippas Tsigas: Fast and lock-free concurrent priority queue for multi-thread systems, JPDC 65, 2005