Data Structures and Algorithms

Skip List

CS 225
G Carl Evans

April 26, 2024

UNIVERSITY OF ILLINOIS URBANA-CHAMPAIGN

Department of Computer Science

Slides by Brad Solomon

Learning Objectives

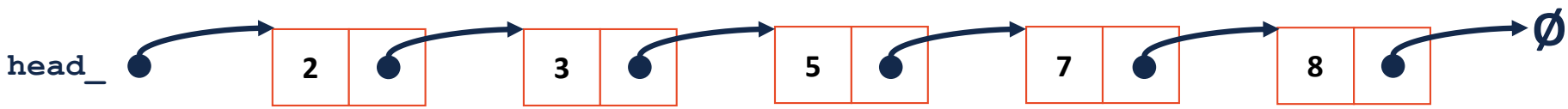
Motivate and introduce the skip list ADT

Conceptualize and code core functions

Discuss efficiency of skip list

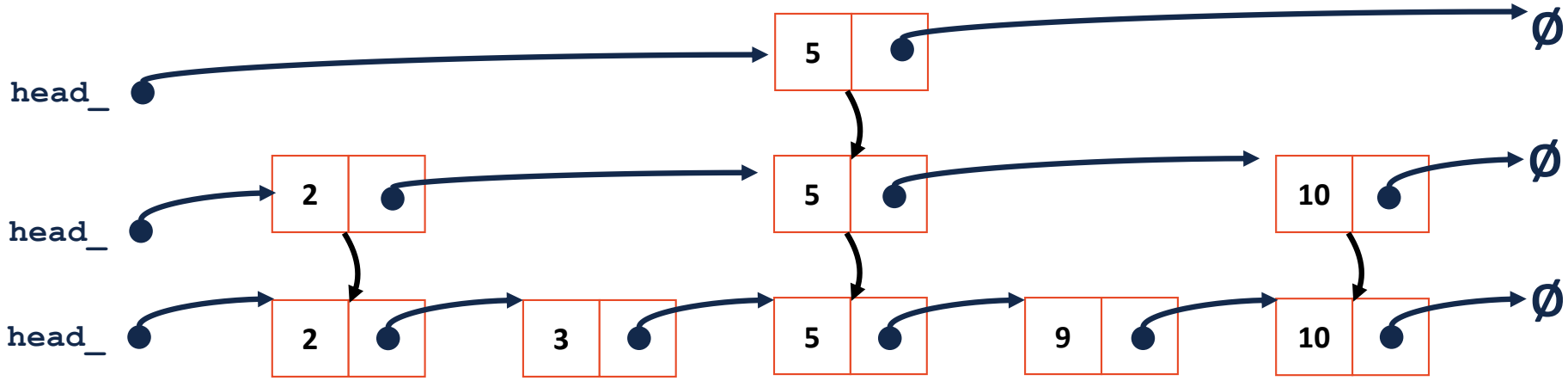
Linked List with ‘Checkpoints’

With some small overhead costs, we can store **checkpoints**.



Linked List with Perfect Checkpoints

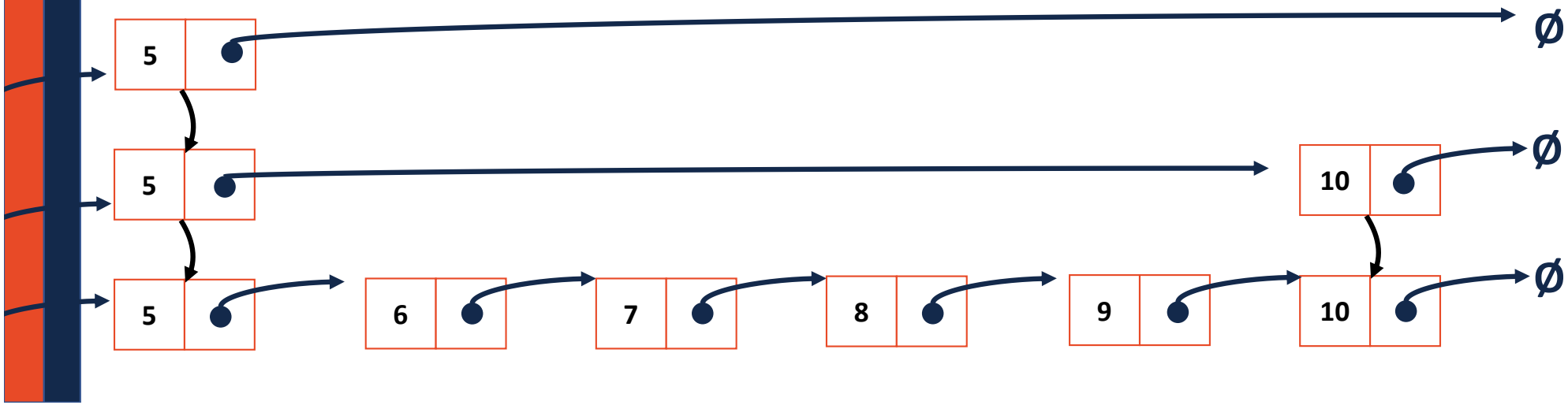
For optimal checkpoints, we want half the number of items at each level



Linked List with Perfect Checkpoints

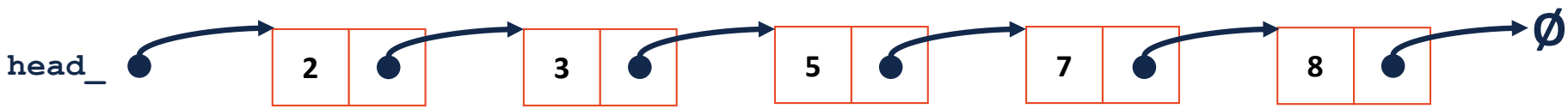
For optimal checkpoints, we want half the number of items at each level

Maintaining this while inserting and deleting is too costly!



Linked List with Random Checkpoints

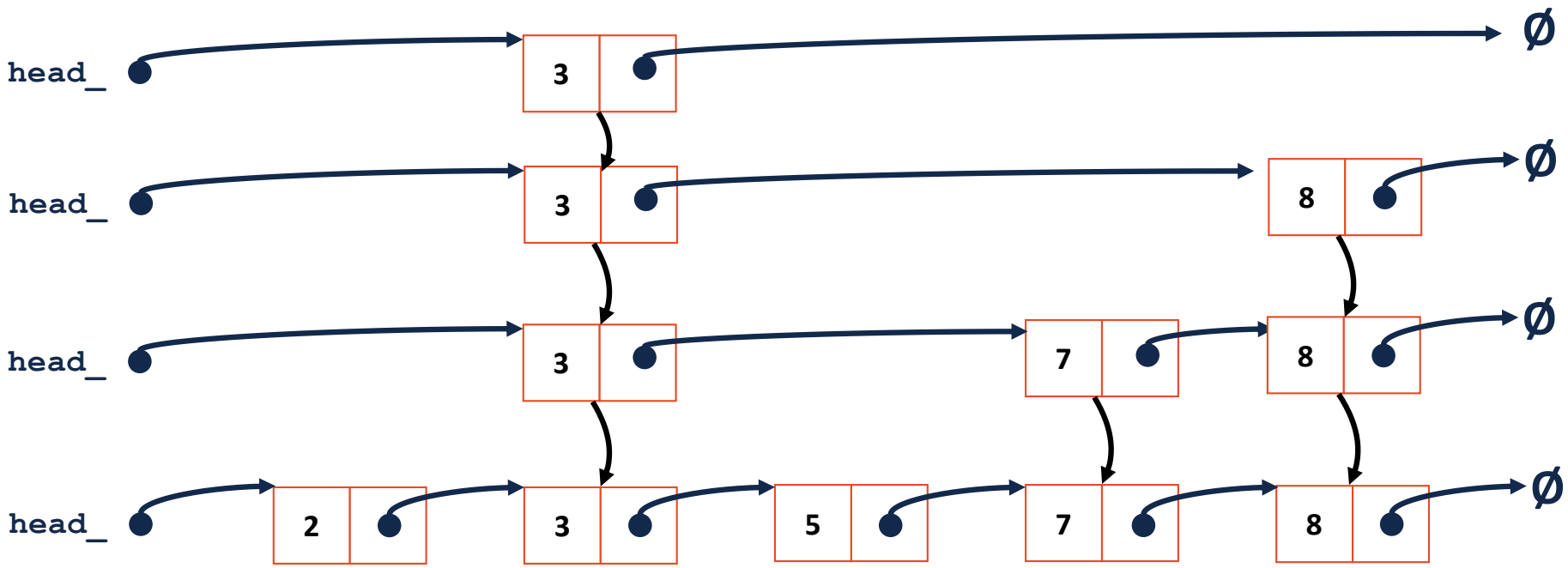
Instead of having **exactly** half each level, let's have **approximately** half



Linked List with Random Checkpoints

Instead of having **exactly** half each level, let's have **approximately** half

To analyze runtimes we use: _____



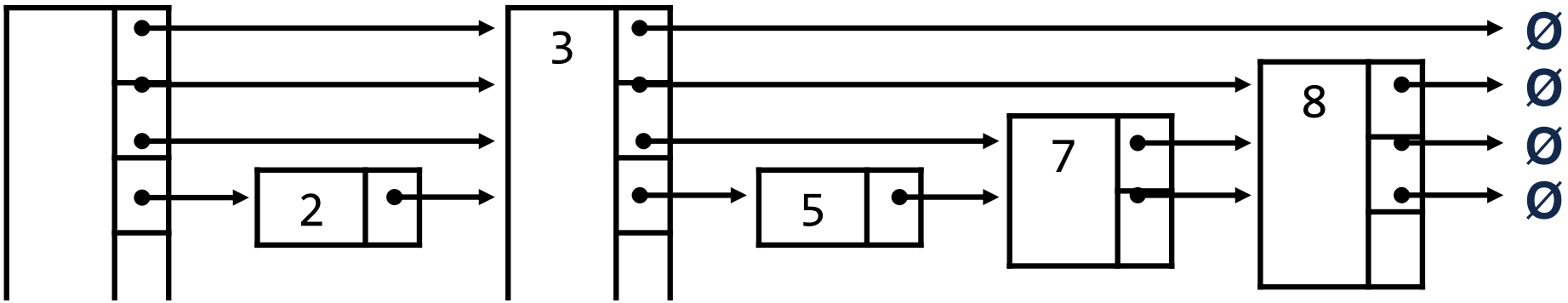
The Skip List

An ordered linked list where each node has variable size

Each node has at most one key but an arbitrary number of pointers

The decision for height is **randomized**

Claim: The **expected** time to insert, search, or delete is $O(logn)$

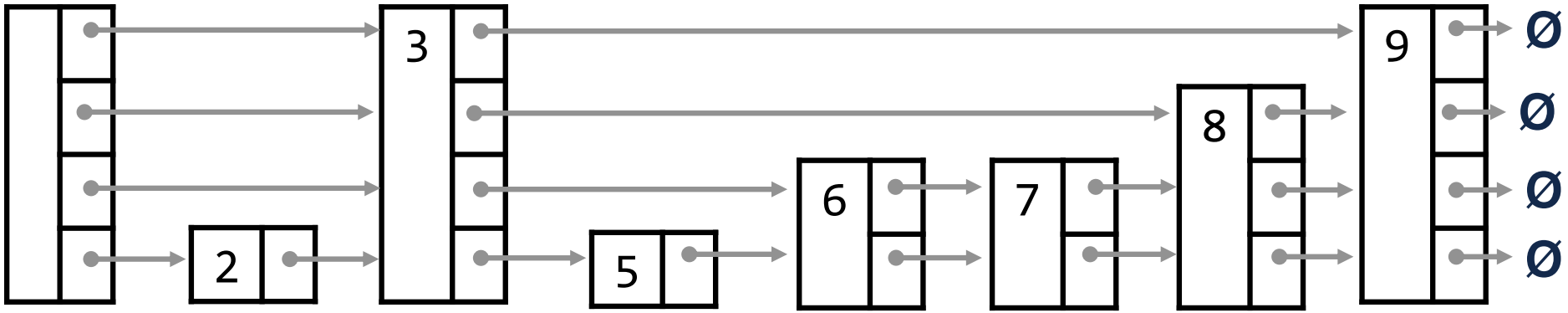


Skip List

```
 1 template <class T>
 2 class SkipList{
 3   public:
 4     class SkipNode{
 5       public:
 6         SkipNode(){
 7           next.push_back(nullptr);
 8         }
 9
10         SkipNode(int h, T & d){
11           data = d;
12           for(int i = 0; i <= h; i++){
13             next.push_back(nullptr);
14           }
15         }
16         T data;
17         std::vector<SkipNode*> next;
18     };
19
20     int max; // max height
21     float c; //update constant
22     SkipNode* head;
23     ...
24
```

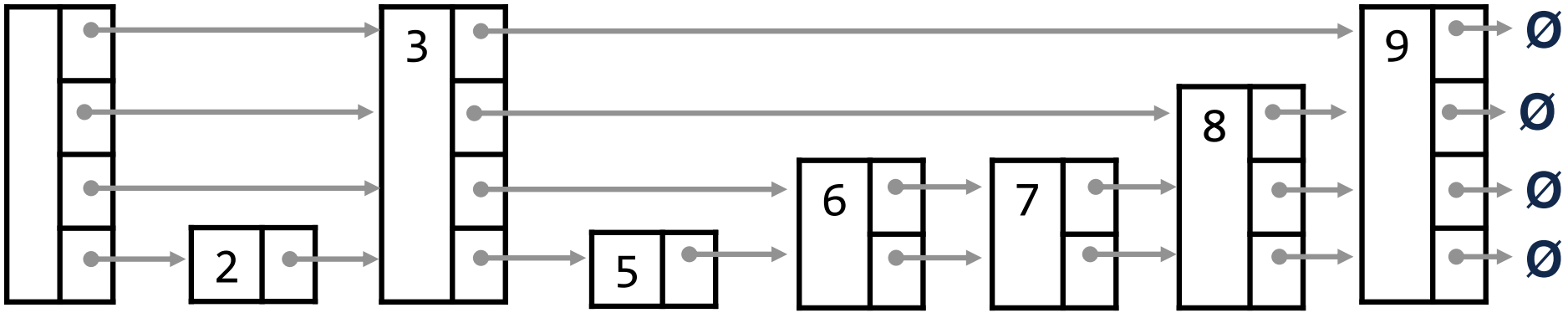
Skip List Find

`Find(9)`



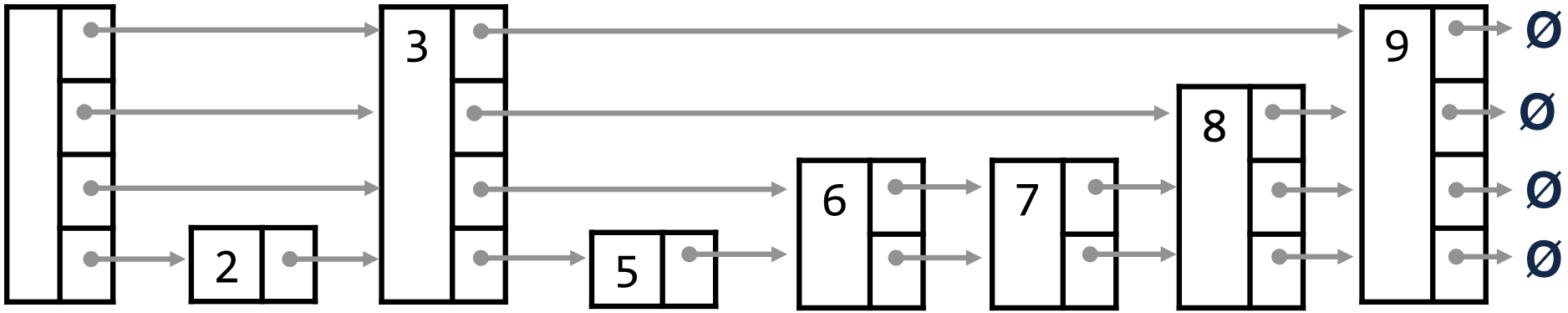
Skip List Find

`Find(7)`



Skip List Find

`Find(1)`

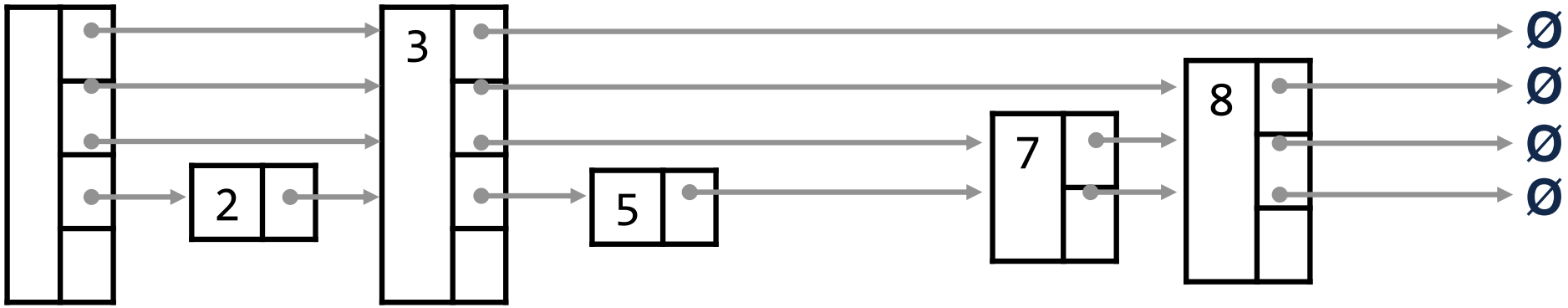


Skip List Find

```
 1 template <class T>
 2 bool SkipList<T>::find(T data){
 3
 4   SkipNode* curr = head;
 5
 6   for(int i = max; i >= 0; i--){
 7     while(curr->next[i]!= nullptr && curr->next[i]->data < data ){
 8       curr = curr->next[i];
 9     }
10   }
11
12   curr = curr->next[0];
13
14   if (curr != nullptr && curr->data == data){
15     return true;
16   }
17   return false;
18 }
19
```

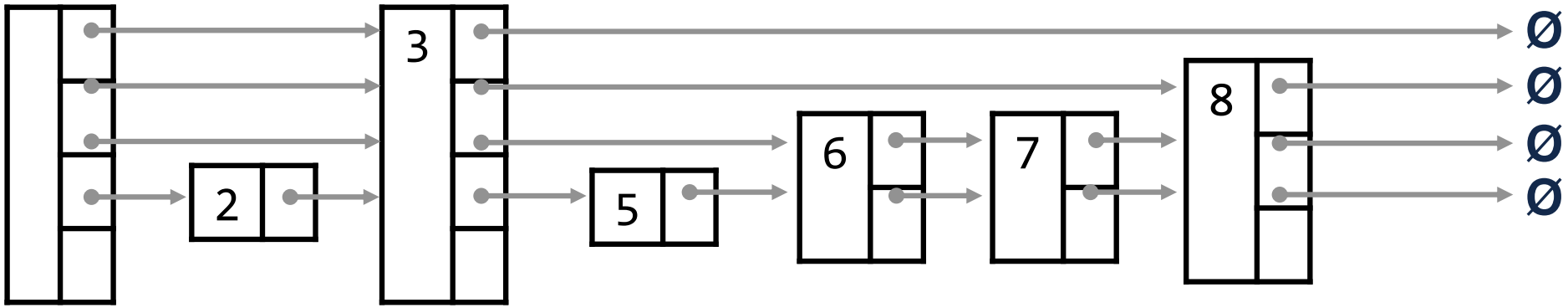
Skip List Insert

`Insert(6)`

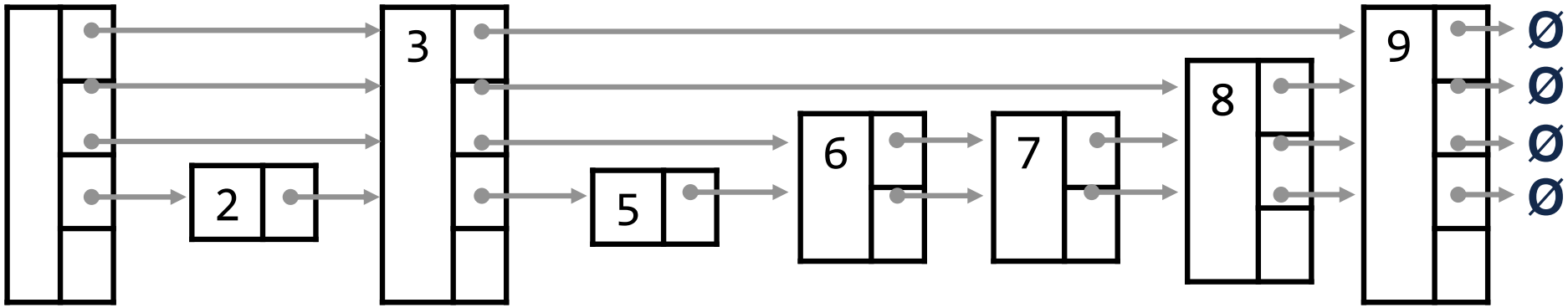


Skip List Insert

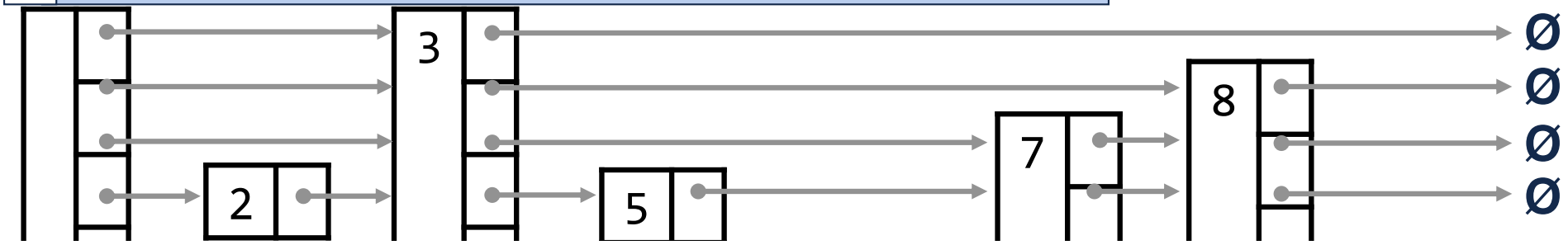
`Insert(9)`



Skip List Insert

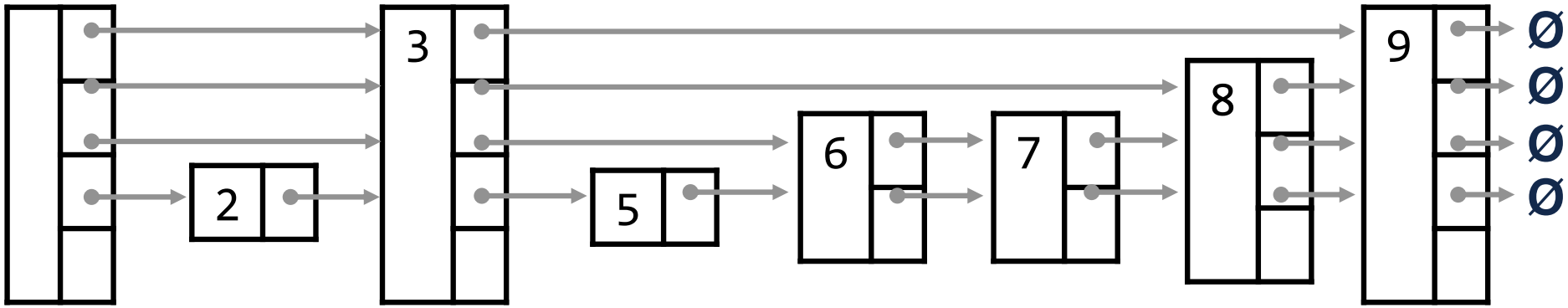


```
1  void SkipList<T>::insert(T data){
2    int h = randHeight();
3    SkipNode* n = new SkipNode(h, data);
4    SkipNode* curr = head;
5
6    for(int i = max; i >= 0; i--){
7      while(curr->next[i]!= nullptr && curr->next[i]->data < data){
8        curr = curr->next[i];
9      }
10     if (h >= i){
11       curr->next[i]=n;
12       n->next[i]=nextNode;
13     }
14   }
15   if (h > max){
16     int diff = h-max;
17     for(int i = 0; i < diff; i++){
18       (head->next).push_back(n);
19     }
20     max = h;
21   }
22 }
```



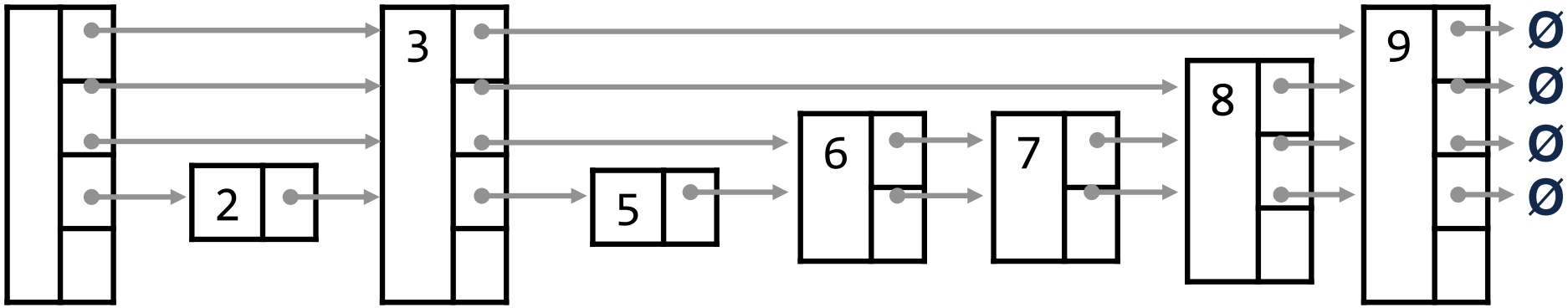
Skip List Remove

Remove(9)



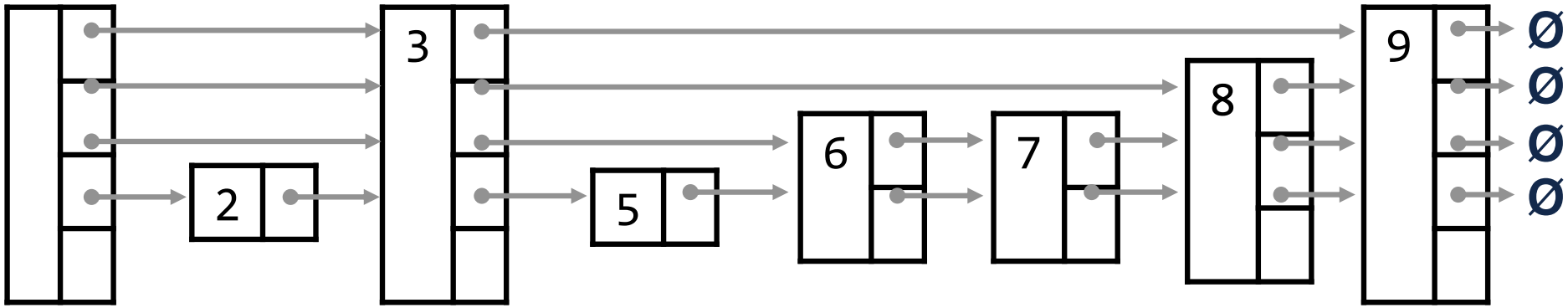
Skip List Remove

Remove(3)



Skip List Remove

Remove(5)



Skip List Expectation

Lets assume our skip list uses a coin flip for randomness (`c=0.5`)

Claim: Expected size of a node is 2.

Skip List Expectation

Lets assume our skip list uses a coin flip for randomness (`c=0.5`)

Claim: Expected size of skip list is 2n.

Skip List Expectation

Claim: Expected height of skip list is $O(logn)$

Skip List Expectation

Claim: Expected height of skip list is $O(logn)$

$$E[h] = \sum_{l=0}^{\lceil logn \rceil} E[I_l] + \sum_{l=\lceil logn \rceil+1}^{\infty} E[I_l] \quad E[I_l] = \begin{cases} 1 & l\,th\ level\ contains\ a\ no \\ 0 & l\,th\ level\ empty \end{cases}$$

Skip List Expectation

Claim: Expected length of search of skip list is $O(logn)$