

# Data Structures and Algorithms

## Skip List

CS 225  
Brad Solomon

December 5, 2025



Not on exam



UNIVERSITY OF  
**ILLINOIS**  
URBANA - CHAMPAIGN



Department of Computer Science



# Prepare questions for next week's review lecture

Monday will be a review lecture

Wednesday will be 'open office hours' (no lecture)

Final exam starts on Friday



# Learning Objectives

Capstone probability lectures with a literature example — the Skip List!

Review fundamentals of probabilistic data structures with the skip list

Conceptualize Skip List ADT functions

Analyze efficiency of skip list while reviewing fundamentals of probability

**The skip list is not on the final exam!**



# Where it all began... A faulty list

Imagine you have a list ADT implementation ***except***...

Every time you called **insert**, it would fail 50% of the time.

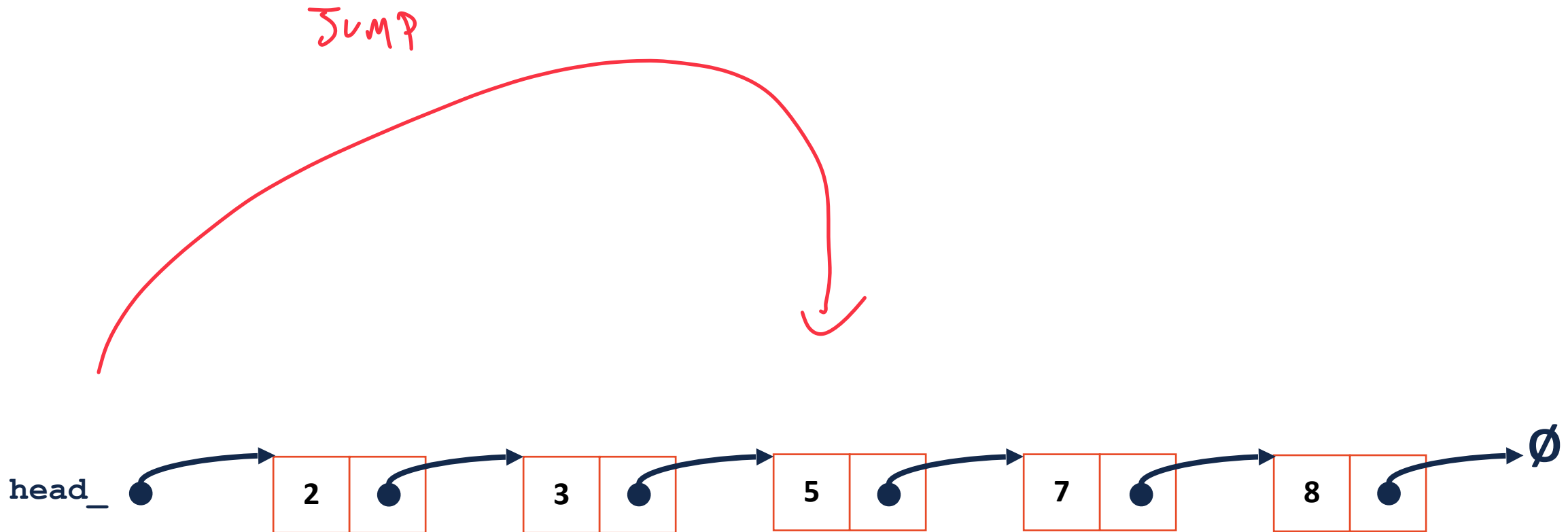
It turns out this system is also useful as an alternative linked list! **How?**



# An alternative linked list

**Goal:** Visit nodes in my linked list in, **on average**,  $\log n$  steps

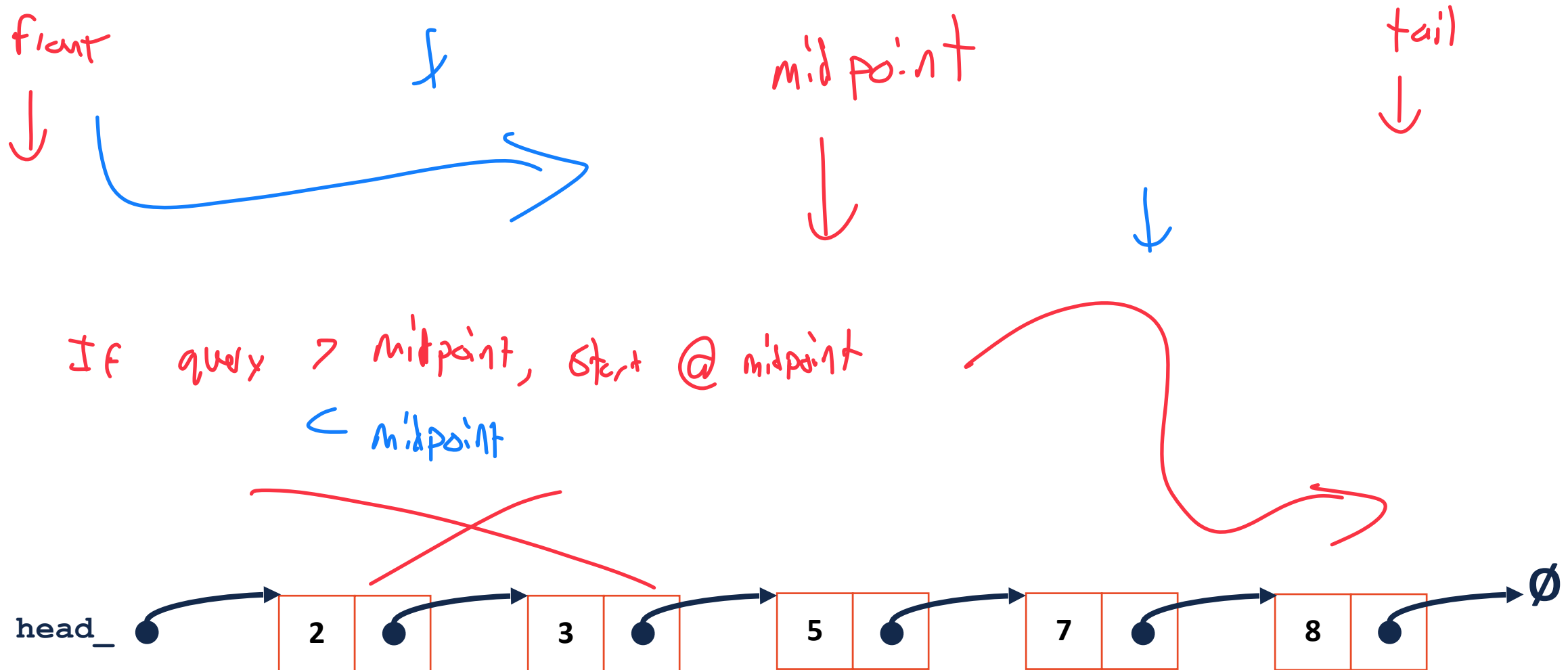
**Big Picture:** I need a way to access nodes **X** positions past the head





# Linked List with 'Checkpoints'

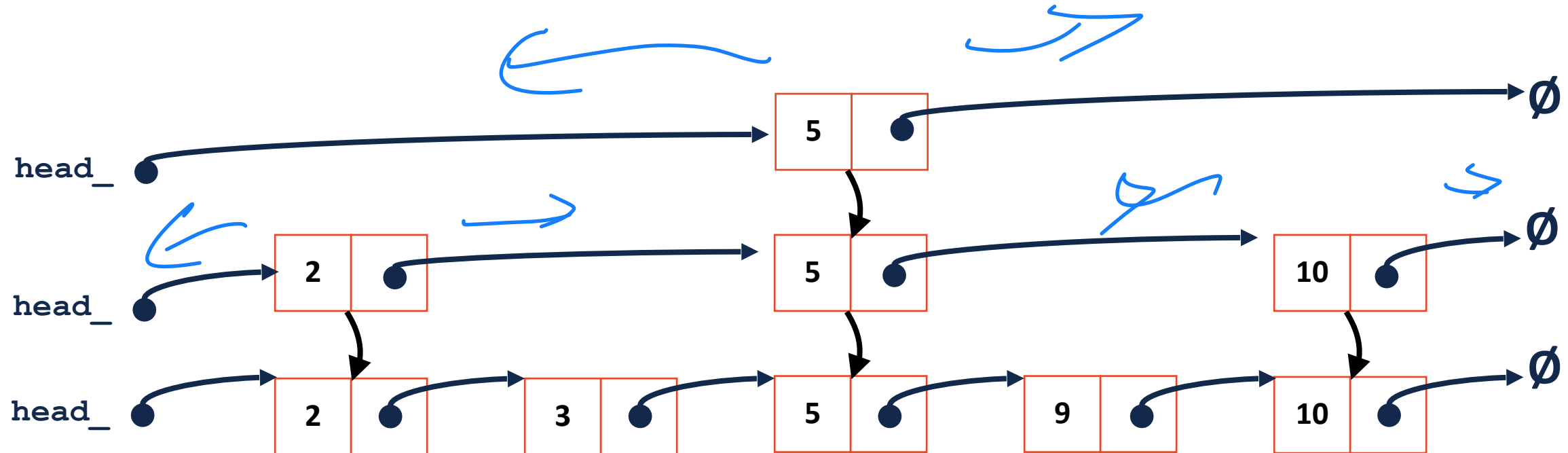
With some small overhead costs, we can store **checkpoints**.





# Linked List with Perfect Checkpoints

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



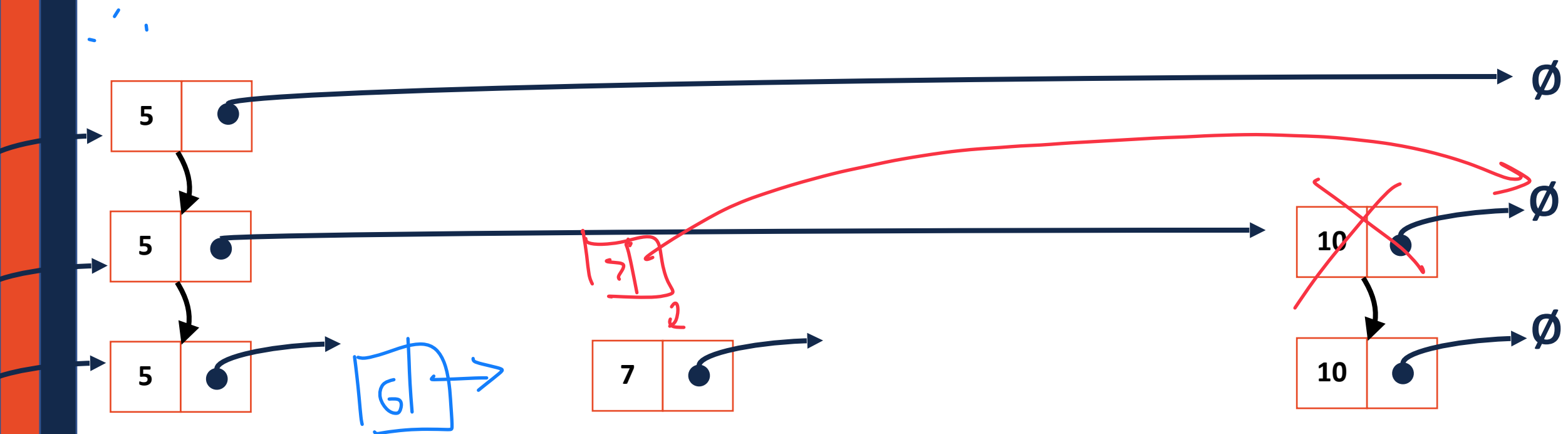


# Linked List with Perfect Checkpoints

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

Maintaining this while inserting and deleting is too costly!

After insert 6, change checkpoint

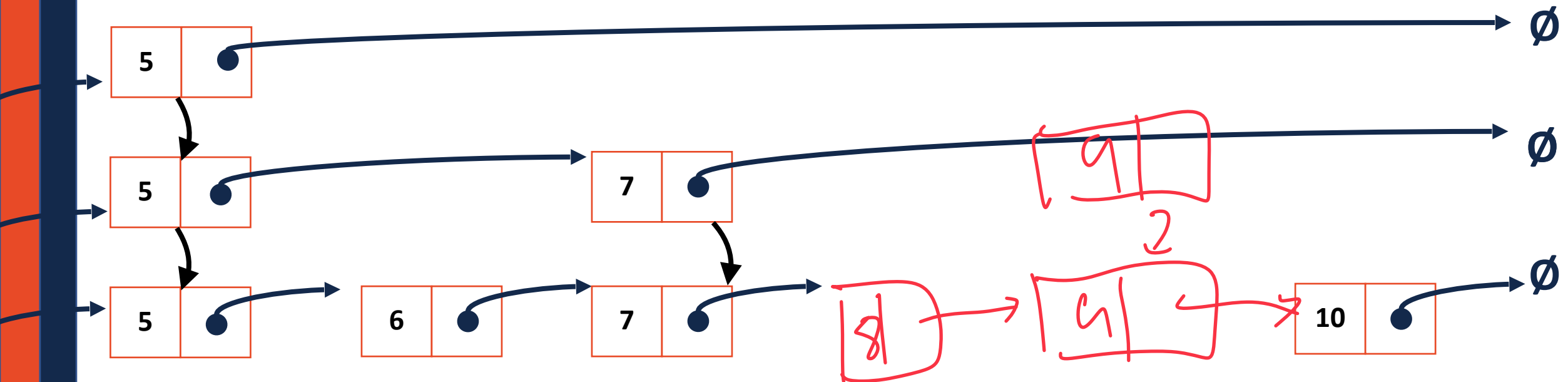




# Linked List with Perfect Checkpoints

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

Maintaining this while inserting and deleting is too costly!

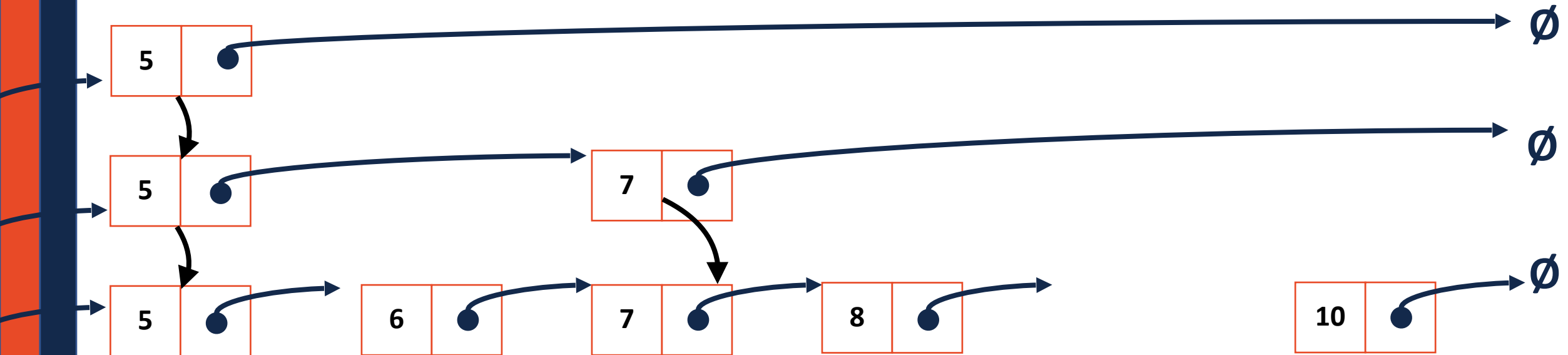




# Linked List with Perfect Checkpoints

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

Maintaining this while inserting and deleting is too costly!

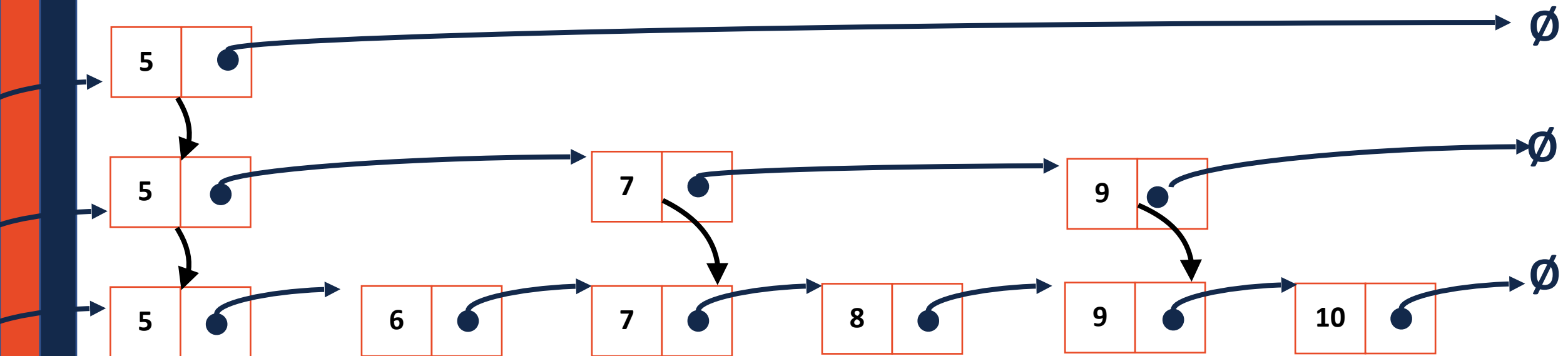




# Linked List with Perfect Checkpoints

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

**Problem:** Having an optimal set of checkpoints is costly to maintain

**Solution:** Every node sets random height



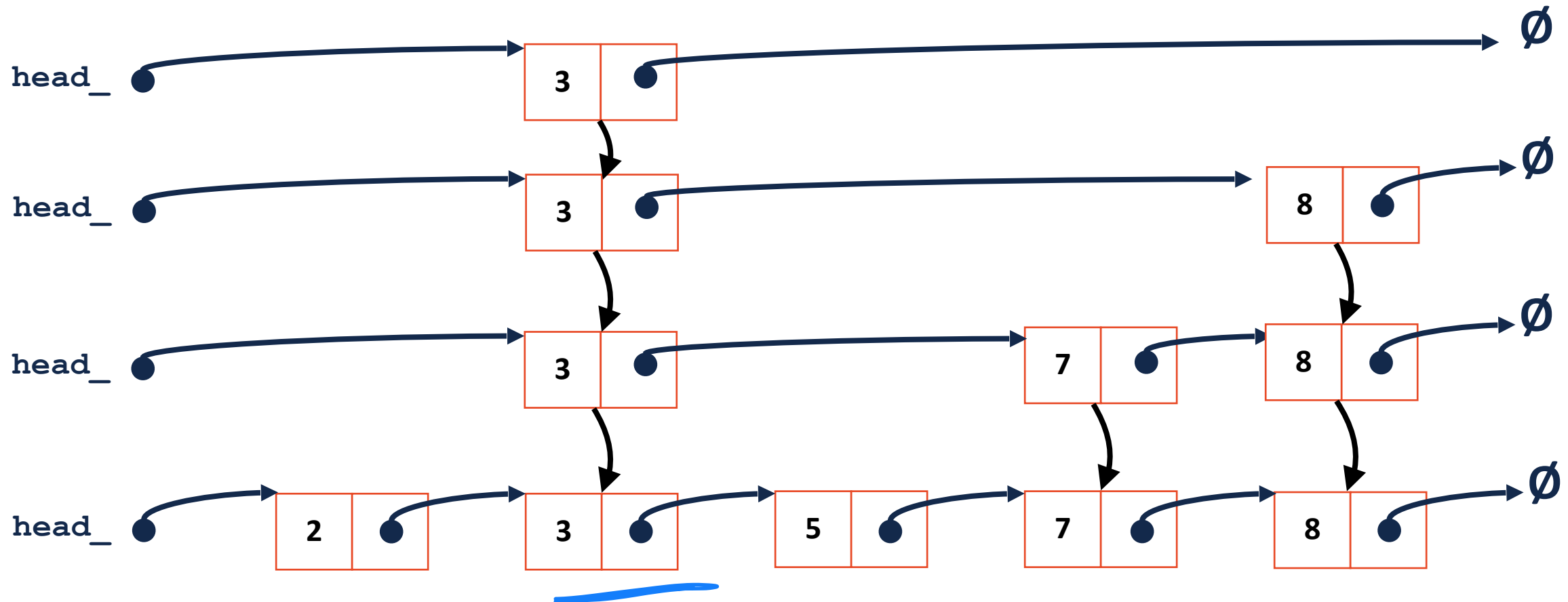




# Linked List with Random Checkpoints

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

To analyze efficiency we should use: Expectation runtime  
↳ Average





# The Skip List

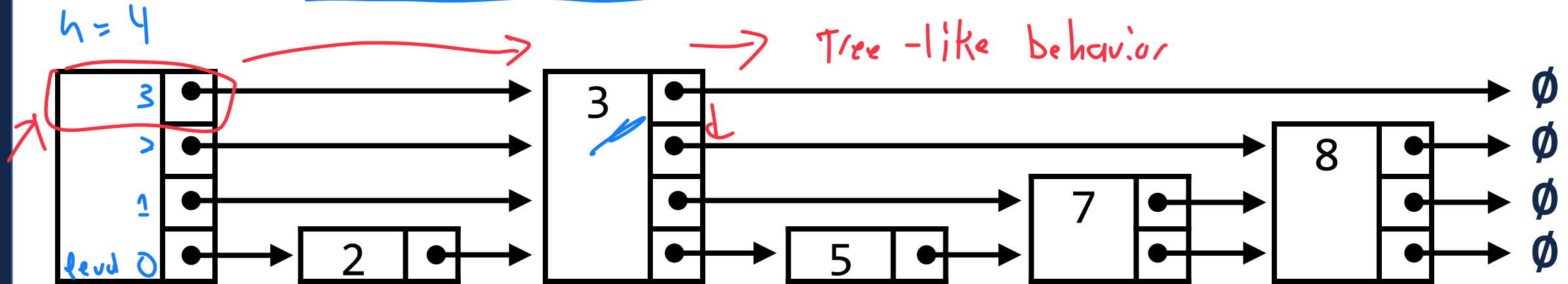


An ordered linked list where each node has variable size

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

The decision for height is **randomized** (Tradeoff  $\rightarrow$  NO rebalancing or adjusting other nodes)

**Claim:** The **average** time to find, insert, or remove is  $\log n$

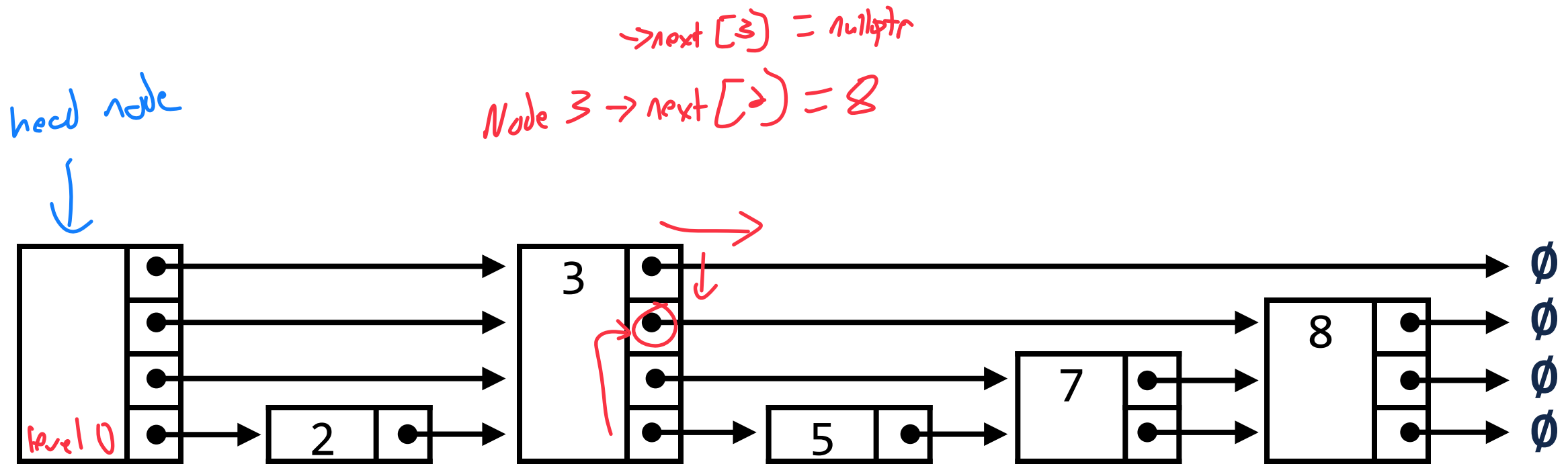




# The Skip List

What would a SkipNode class look like? How about the SkipList class?

- ↳ Key (T)
- ↳ Pointers (Vector)
- ↳ Height (unsigned)



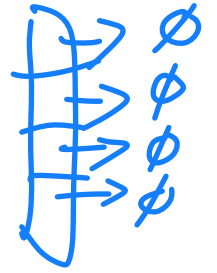


# Skip List

```
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
```

The Skip List head node has no data

null case has next



height of node is implicit in vector





# Skip List ADT

**Find**

**Insert**

**Remove**

**Constructor**



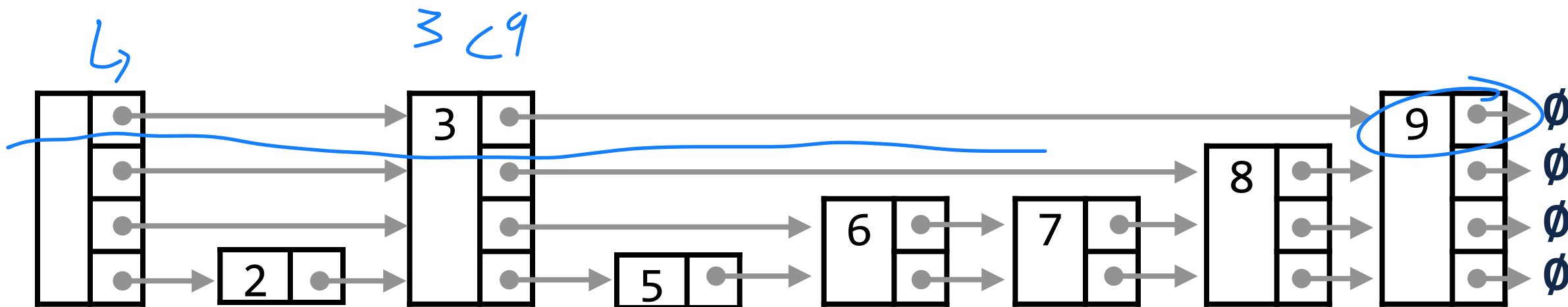
# Skip List Find

## Find (9)

If  $curr \rightarrow data == q$ , done!

If  $curr \rightarrow data == q_v$ ,  $curr = curr \rightarrow next$  (at same level)

$curr \rightarrow data > q$ , go down one level in height





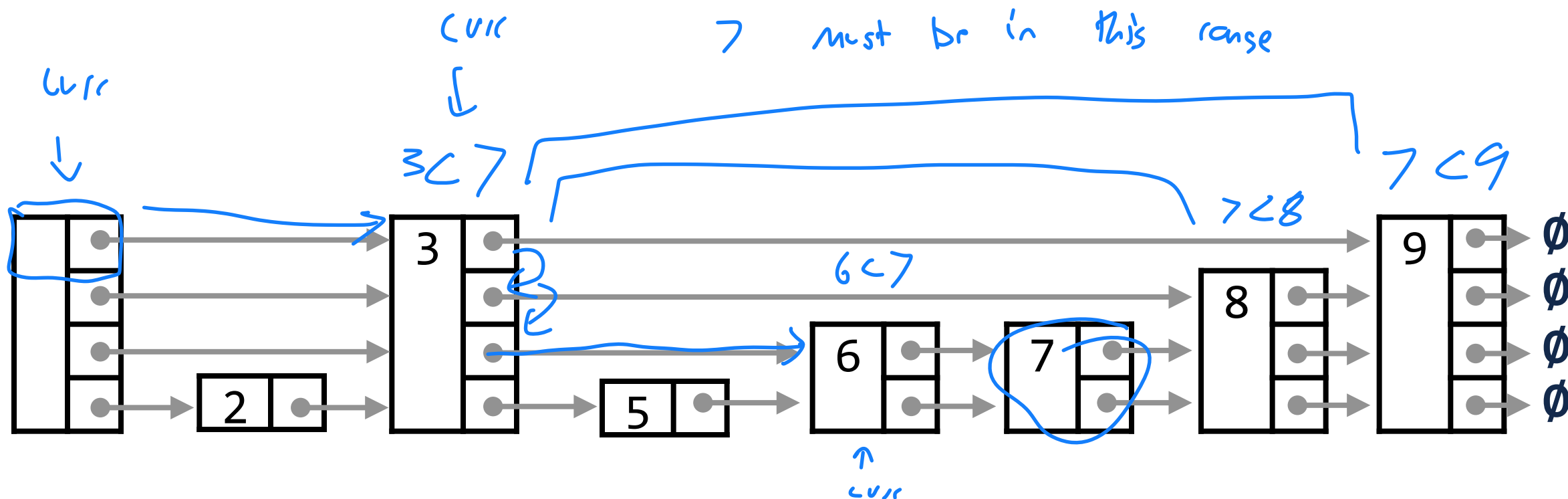
# Skip List Find

# Find (7)

Starting at top level... if next node's key matches, done!

If key smaller than next node's key, **move down a level**

If key larger than next node's key, **go to next node at current level**





# Skip List Find

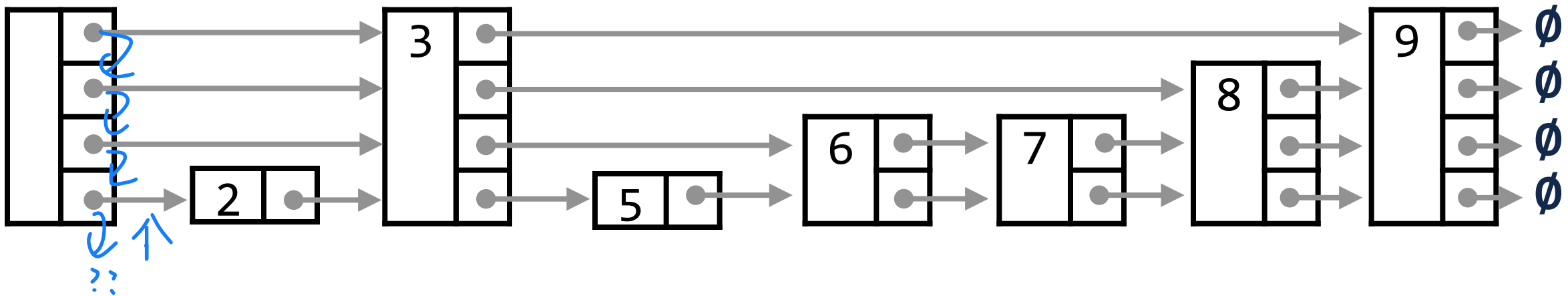
## Find (1)

Starting at top level... if next node's key matches, done!

If key smaller than next node's key, **move down a level**

If key larger than next node's key, **go to next node at current level**

Stop when height is  $-1$

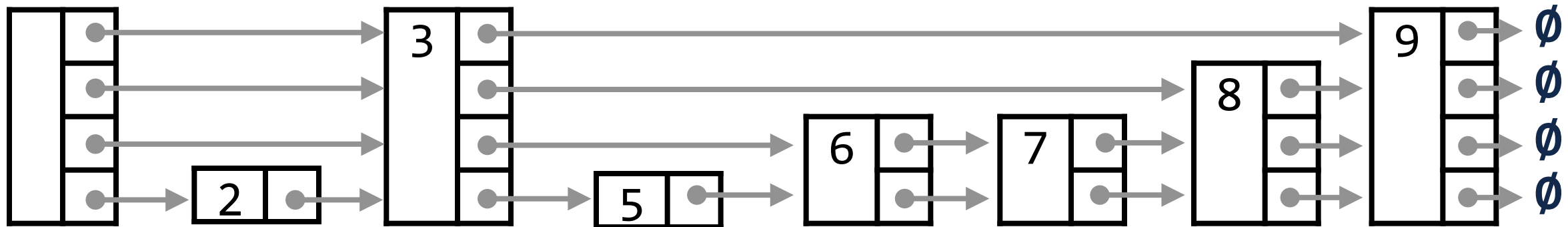




# Skip List Find



Could you code up Skip List Find?





# Skip List Find



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

initialize to head

outer loop decreases height

inner loop is linked list

one trick → look ahead

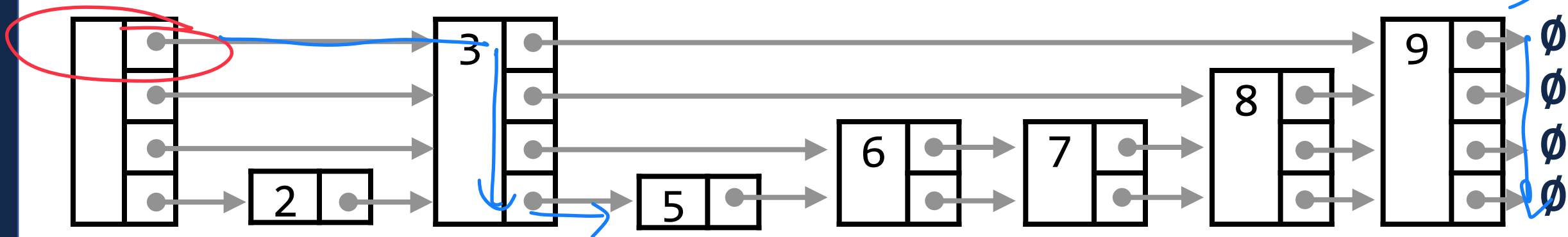
(at curr we compare next)

find match

or

return false

5

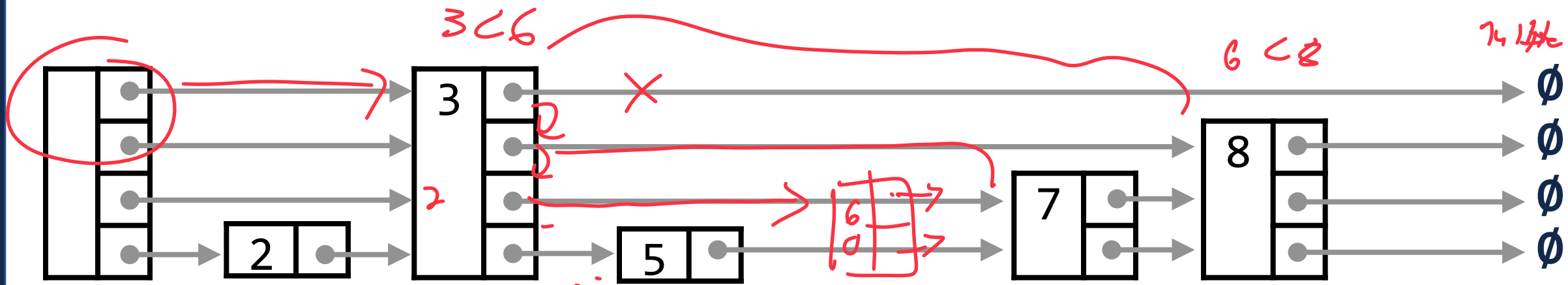




# Skip List Insert

- 1) Random height for new node  $\rightarrow$  height (2)
- 2) Find position to insert at highest level  $\nearrow$

Insert (6)

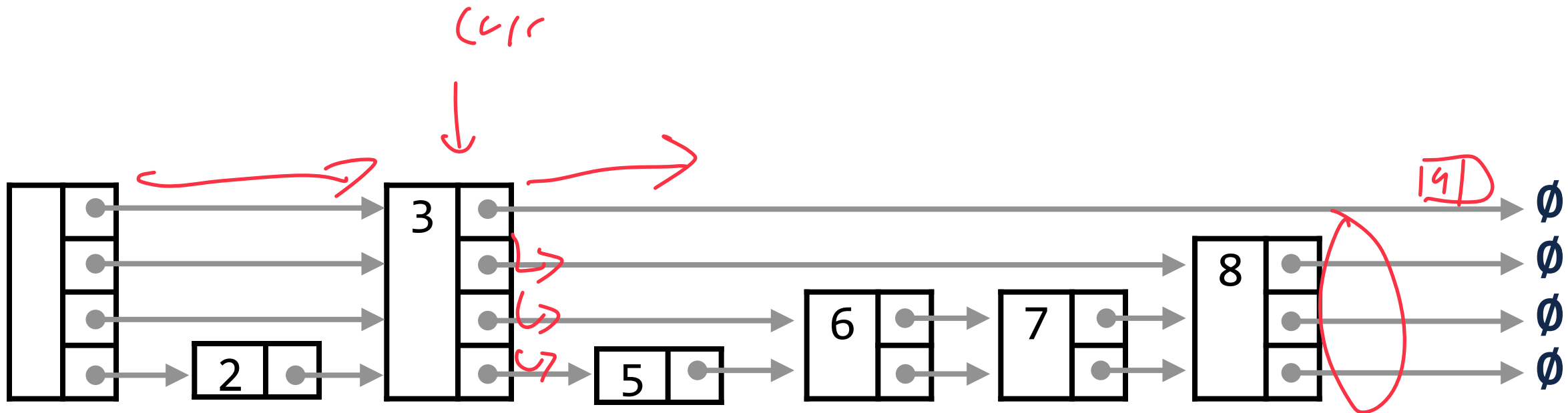




## Insert (9)

→ 5

↳ Once insert at highest level, go through every lower level

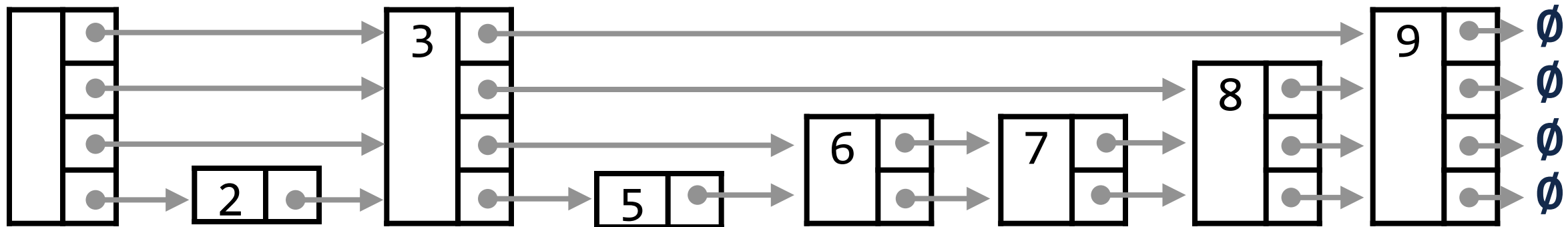




# Skip List Insert

Randomly generate height for insert

Use Find() logic but insert at every list with height  $\geq$  random

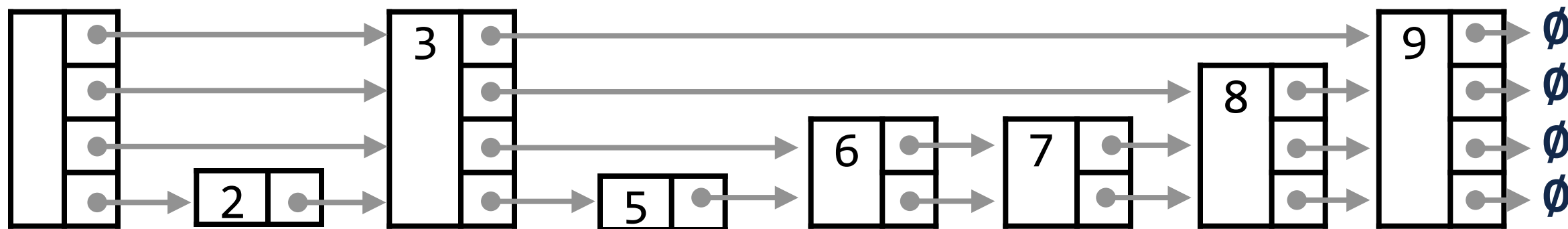




# Skip List Insert



Could you code up 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      tmp = curr->next[i];
12      curr->next[i] = n;
13      n->next[i] = tmp;
14    }
15  }
16  if (h > max){
17    int diff = h - max;
18    for(int i = 0; i < diff; i++){
19      (head->next).push_back(n);
20    }
21    max = h;
22  }
```

initialize

find

if height is equal to or less than initialized height of node

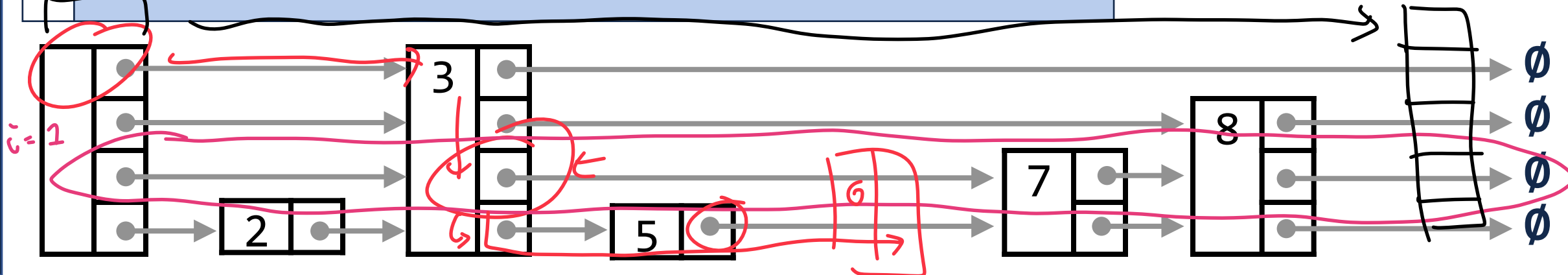
Add node to linked list

If random height exceeds height max

Add height to head node

$h = 2$   
height 2

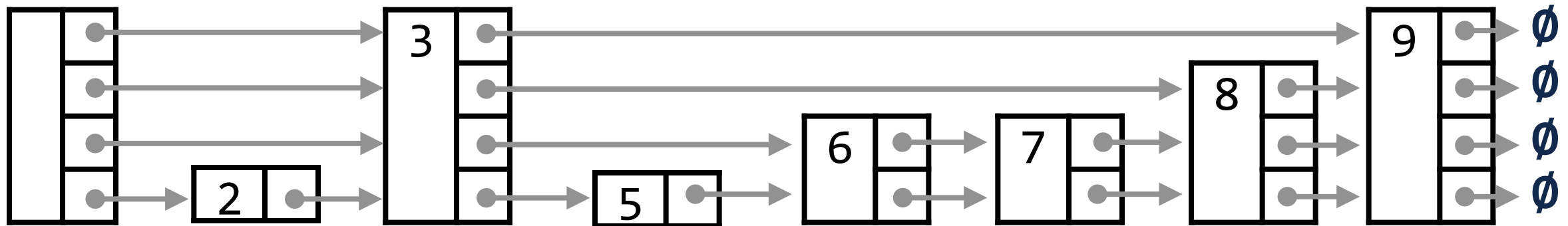
$[0] \rightarrow$





# Skip List Remove

Remove (9)



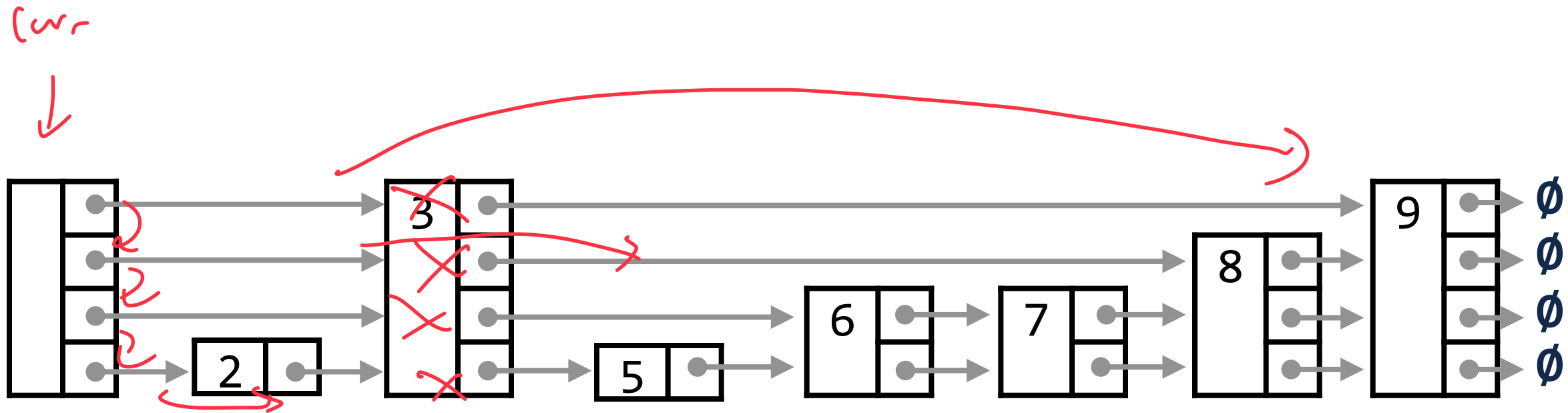


# Skip List Remove

Remove (3)

Use Find() logic but remove before descending **the previous node**

The remove is a standard Linked List Remove (but at each level)



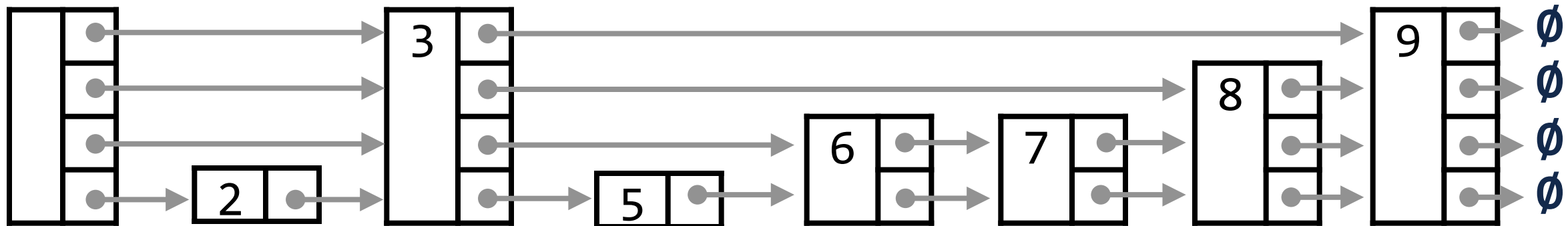


# Skip List Remove

Remove (5)

Use Find() logic but remove before descending **the previous node**

The remove is a standard Linked List Remove (but at each level)

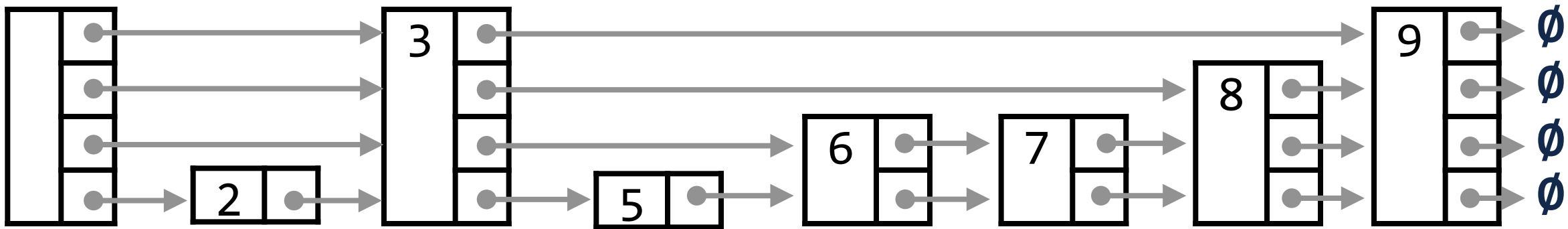




# Skip List Efficiency

We've seen the full ADT but haven't explored the runtime

What is the Big O for Find()?





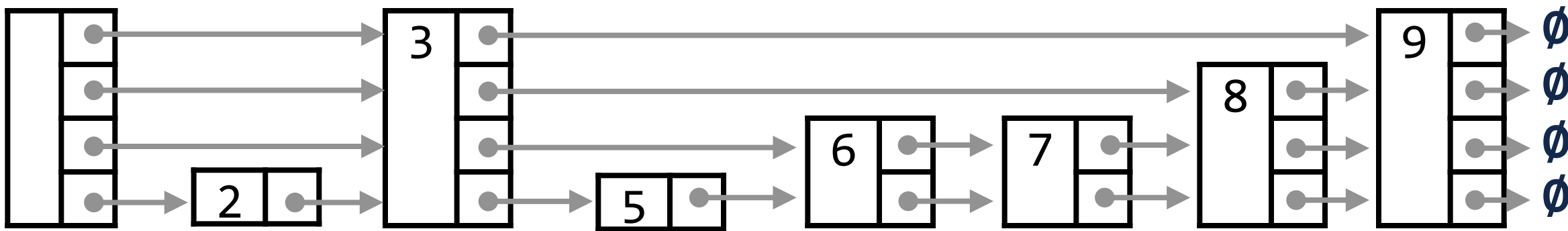
# Skip List Efficiency

# We've seen the full ADT but haven't explored the runtime

What is the Big O for Find()?  $O(n)$  for  $n$  nodes (keys)

## Using probability, how can we show skip list is better than $O(n)$ ?

$D \rightarrow D \rightarrow D \rightarrow D \rightarrow D \rightarrow \emptyset$  worst case world





# Skip List Efficiency

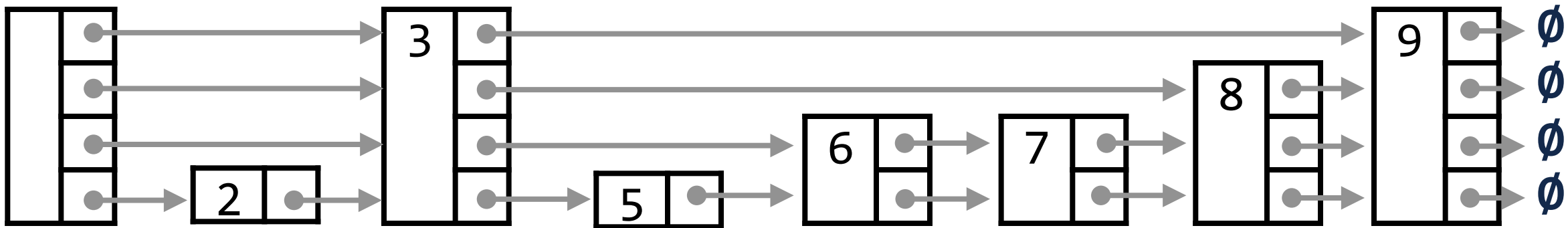


We've seen the full ADT but haven't explored the runtime

What is the Big O for Find()?  $O(n)$  for  $n$  nodes (keys)

**Using probability, how can we show skip list is better than  $O(n)$ ?**

- 1) Formalize the probability of SkipList reaching height  $h > \log n$
- 2) Define a recurrence relationship for search path
- 3) Use (1) and (2) to show that our **average** search time is  $\log n$





# Skip List Random Height

We want half the number of items (approximately) at each level.

How can we do this?

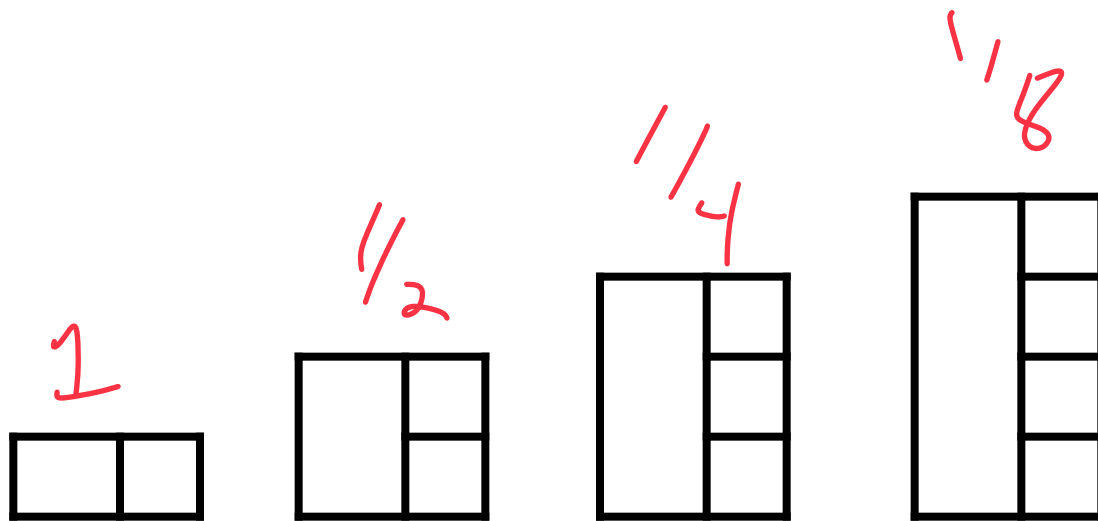


# Skip List Random Height

By definition, each increase in height occurs with probability  $c$ .

If  $c = 0.5$  (a coin flip), to reach level  $l$ , we must flip  $l$  heads in a row

**By definition the probability a node reaching level  $l$  is  $c^l$**



```
1  template <class T>
2  int SkipList<T>::randHeight() {
3      float frac = rand() / (float) RAND_MAX;
4      int h = 0;
5      while (frac < c) {
6          h++;
7          frac = rand() / (float) RAND_MAX;
8      }
9      return h;
}
```



# Skip List Expectation

We want to bound the height of a SkipList of  $n$  nodes but this is deceptively hard to prove **in expectation**:

$$E[h] = \sum_{l=0}^{\infty} E[I_l] \quad I_l = \begin{cases} 1 & \text{if } l\text{th level contains a node} \\ 0 & \text{if } l\text{th level contains no nodes} \end{cases}$$

*Handwritten notes:* A red arrow points to  $E[h]$ . A red arrow points to  $E[I_l]$  with the note "fun trick". A red underline is under the first case of the definition of  $I_l$ .

$$E[h] = \sum_{l=0}^{\lceil \log n \rceil} E[I_l] + \sum_{l=\lceil \log n \rceil+1}^{\infty} E[I_l] \leq \sum_{l=0}^{\lceil \log n \rceil} 1 + \sum_{l=\lceil \log n \rceil+1}^{\infty} \frac{n}{2^l}$$

*Handwritten notes:*

- An arrow points to the first sum with the note "Ignore this".
- An arrow points to the second sum with the note "if we can show this goes to 0".
- An arrow points to the "1" in the second sum with the note "Assume all height  $< \log n$ ".
- An arrow points to the  $\frac{n}{2^l}$  term with the note " $\sim \frac{n}{\log n}$ ".



# Skip List Average-Case Performance

Instead we will define an equation for the likelihood of SkipList of  $n$  nodes having a height larger than  $\log n$  and claim that the probability is small.

Let  $c$  be the probability of increasing a node's height by 1.

Probability a single node reaches level  $l$ :  $c^l$



# Skip List Average-Case Performance

Instead we will define an equation for the likelihood of SkipList of  $n$  nodes having a height larger than  $\log n$  and claim that the probability is small.

Let  $c$  be the probability of increasing a node's height by 1.

Probability a single node reaches level  $l$ :  $c^l$

Probability a single node does not reach level  $l$ :  $1 - c^l$



# Skip List Average-Case Performance

Instead we will define an equation for the likelihood of SkipList of  $n$  nodes having a height larger than  $\log n$  and claim that the probability is small.

Let  $c$  be the probability of increasing a node's height by 1.

Probability a single node reaches level  $l$ :  $c^l$

Probability a single node does not reach level  $l$ :  $1 - c^l$

Probability  $n$  nodes do not reach level  $l$ :  $(1 - c^l)^n$



# Skip List Average-Case Performance

Instead we will define an equation for the likelihood of SkipList of  $n$  nodes having a height larger than  $\log n$  and claim that the probability is small.

Let  $c$  be the probability of increasing a node's height by 1.

Probability a single node reaches level  $l$ :  $c^l$

Probability a single node does not reach level  $l$ :  $1 - c^l$

Probability  $n$  nodes do not reach level  $l$ :  $(1 - c^l)^n$



# Skip List Average-Case Performance

Instead we will define an equation for the likelihood of SkipList of  $n$  nodes having a height larger than  $\log n$  and claim that the probability is small.

Probability  $n$  nodes do not reach level  $l$ :  $(1 - c^l)^n$

Probability at least one node reaches level  $l$ :  $1 - (1 - c^l)^n$



# Skip List Average-Case Performance

Instead we will define an equation for the likelihood of SkipList of  $n$  nodes having a height larger than  $\log n$  and claim that the probability is small.

Probability  $n$  nodes do not reach level  $l$ :  $(1 - c^l)^n$

Probability at least one node reaches level  $l$ :  $1 - (1 - c^l)^n$

Using this equation, the probability of exceeding height  $h$  is:  $nc^h$

Still bad :-

**Skip List height is unbounded, but we control probability!**



# Skip List Average-Case Performance



To quote the original 1990 skipList paper:

*"Don't worry, be happy. Simply start a search at the highest level present in the list. As we will see in our analysis, the probability that the maximum level in a list of  $n$  elements is significantly larger than  $L(n)$  is very small."*

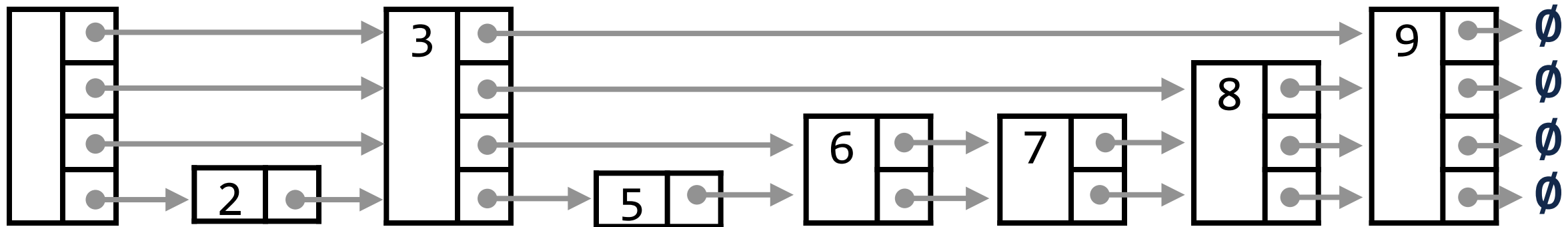
The authors use this logic to state  $L(n) = \log_{1/c} n$  as the optimal (or expected) max height.



# Skip List Expectation

**Claim:** Expected length of search of skip list is the height  $\approx (\log n)$

**Proof:** Direct with recurrence equation working backwards





# Skip List Expectation

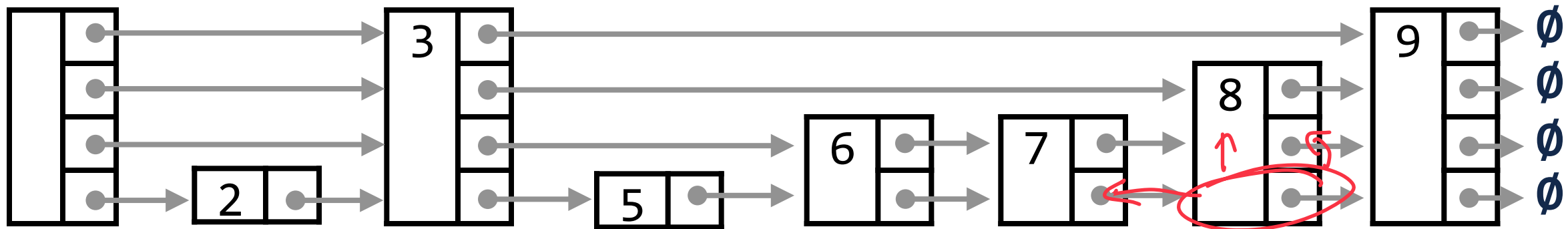
**Claim:** Expected length of search of skip list is the height  $\approx (\log n)$

**Proof:** Direct with recurrence equation working backwards

Let  $H(k)$  be the expected cost to search a path of  $k$  levels

Then  $H(k) = 1 + (1 - c) * H(k) + c * H(k - 1)$

*Handwritten notes: "go up a level" above the equation, and red arrows pointing to the terms  $(1 - c) * H(k)$  and  $c * H(k - 1)$ .*





# Skip List Expectation

**Claim:** Expected length of search of skip list is the height  $\approx (\log n)$

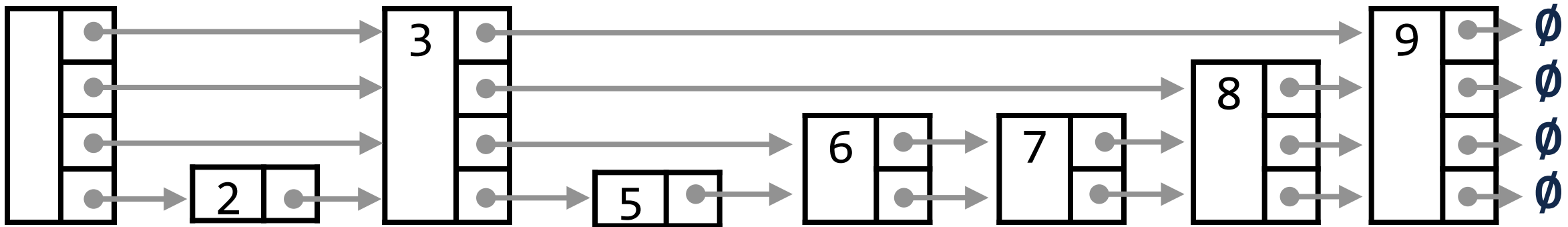
**Proof:** Direct with recurrence equation working backwards

Let  $H(k)$  be the expected cost to search a path of  $k$  levels

Then  $H(k) = 1 + (1 - c) * H(k) + c * H(k - 1)$

Rewrite:  $H(k) - (1 - c) * H(k) = 1 + c * H(k - 1)$

$1 - (1 - c) = c$





# Skip List Expectation

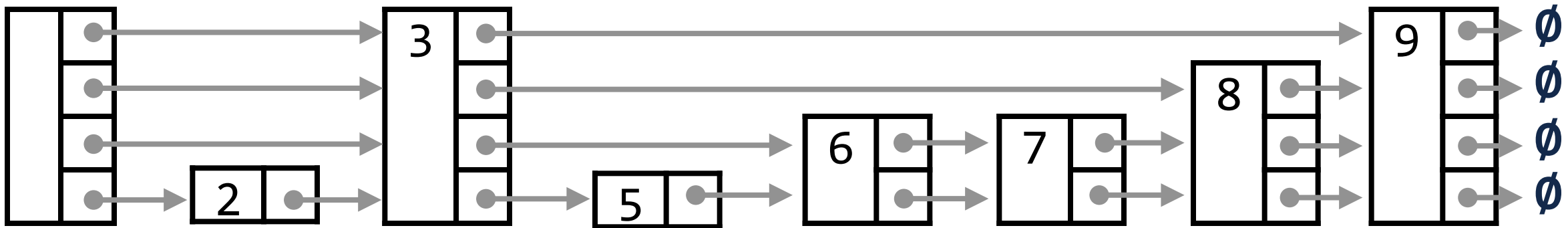
**Claim:** Expected length of search of skip list is the height  $\approx (\log n)$

**Proof:** Direct with recurrence equation working backwards

Let  $H(k)$  be the expected cost to search a path of  $k$  levels

Rewrite:  $H(k) - (1 - c) * H(k) = 1 + c * H(k - 1)$

Rewrite:





# Skip List Expectation

**Claim:** Expected length of search of skip list is the height  $\approx \log n$

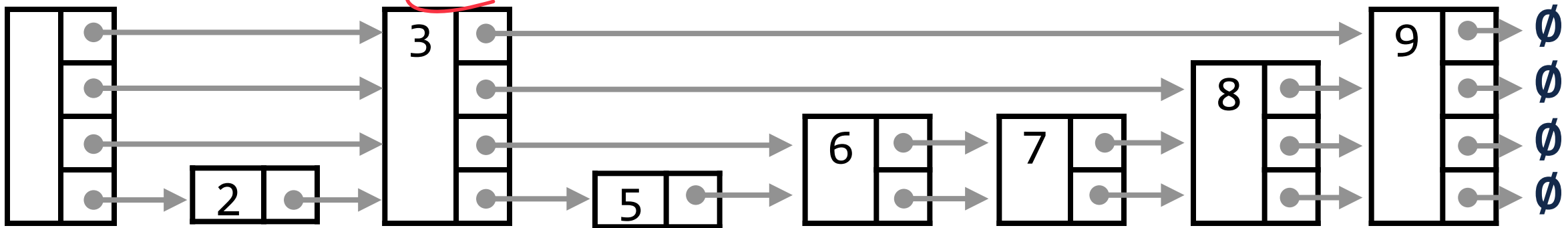
**Proof:** Direct with recurrence equation working backwards

Let  $H(k)$  be the expected cost to search a path of  $k$  levels

Rewrite:  $H(k) - (1 - c) * H(k) = 1 + c * H(k - 1)$

Rewrite:  $c * H(k) = 1 + c * H(k - 1) \Rightarrow H(k) = 1/c + H(k - 1)$

Trivial Soln:  $k/c$







# Skip List Efficiency

We've seen the full ADT but haven't explored the runtime

What is the Big O for Find()?  $O(n)$  for  $n$  nodes (keys)

**Skip List mimics behavior of AVL Tree, despite being linked list**

- 1) Our height is (on average)  $\log n$
- 2) The expected cost to traverse is height bounded!
- 3) So our average search time is  $\log n$  (on order of height)

TABLE II. Timings of Implementations of Different Algorithms

Implementation	Search Time	Insertion Time	Deletion Time
<i>Skip lists</i>	0.051 msec(1.0)	0.065 msec(1.0)	0.059 msec(1.0)
<i>non-recursive AVL trees</i>	0.046 msec(0.91)	0.10 msec (1.55)	0.085 msec(1.46)
<i>recursive 2-3 trees</i>	0.054 msec(1.05)	0.21 msec (3.2)	0.21 msec (3.65)
<i>Self-adjusting trees:</i>			
<i>top-down splaying</i>	0.15 msec (3.0)	0.16 msec (2.5)	0.18 msec (3.1)
<i>bottom-up splaying</i>	0.49 msec (9.6)	0.51 msec (7.8)	0.53 msec (9.0)

prove empirically



# In Conclusion

**If interested, read the original publication!**

William Pugh. 1990. Skip lists: a probabilistic alternative to balanced trees. Commun. ACM 33, 6 (June 1990), 668–676.

<https://doi.org/10.1145/78973.78977>

**If not, hopefully you learned a few things about probability in CS!**

