Data Structures and Algorithms
Random Algorithms and SkipList

CS 225
Brad Solomon

November 9, 2022

UNIVERSITY OF ILLINOIS URBANA-CHAMPAIGN

Department of Computer Science

Learning Objectives

Distinguish the three main types of 'random' in computer science

Motivate and introduce the skip list ADT

Conceptualize and code core functions

Randomized Algorithms

A **randomized algorithm** is one which uses a source of randomness somewhere in its implementation.

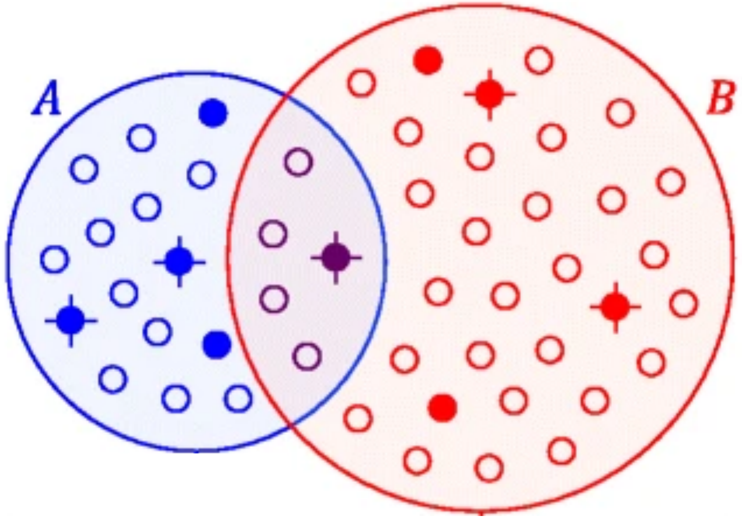
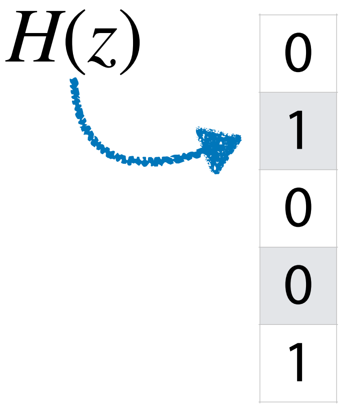
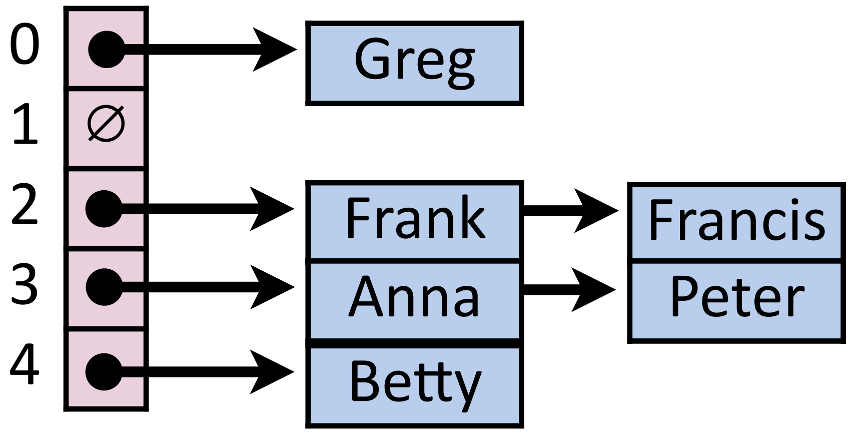


Figure from Ondov et al 2016





$H(x)$	0	2	1	0	0	4	0	2	0	6
$H(y)$	1	0	2	3	1	0	3	4	0	1
$H(z)$	2	1	0	2	0	1	0	0	7	2

Randomization in Algorithms

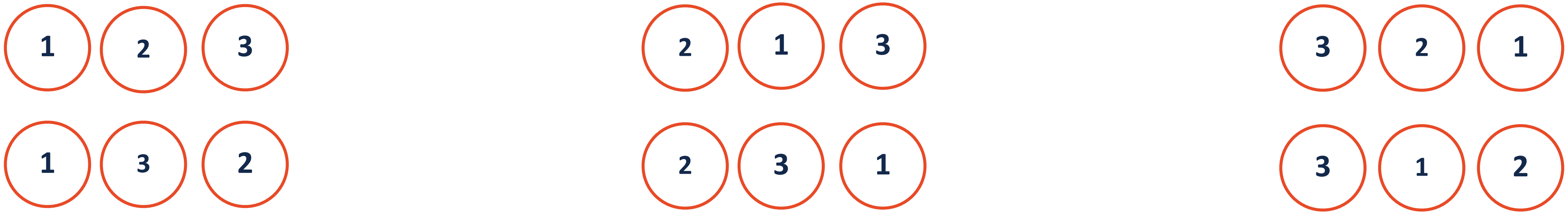
1. Assume input data is random to estimate average-case performance

2. Use randomness inside algorithm to estimate expected running time

3. Use randomness inside algorithm to approximate solution in fixed time

Randomization in Algorithms

1. Assume **input data is random** to estimate average-case performance



1 2 3

1 3 2

2 1 3

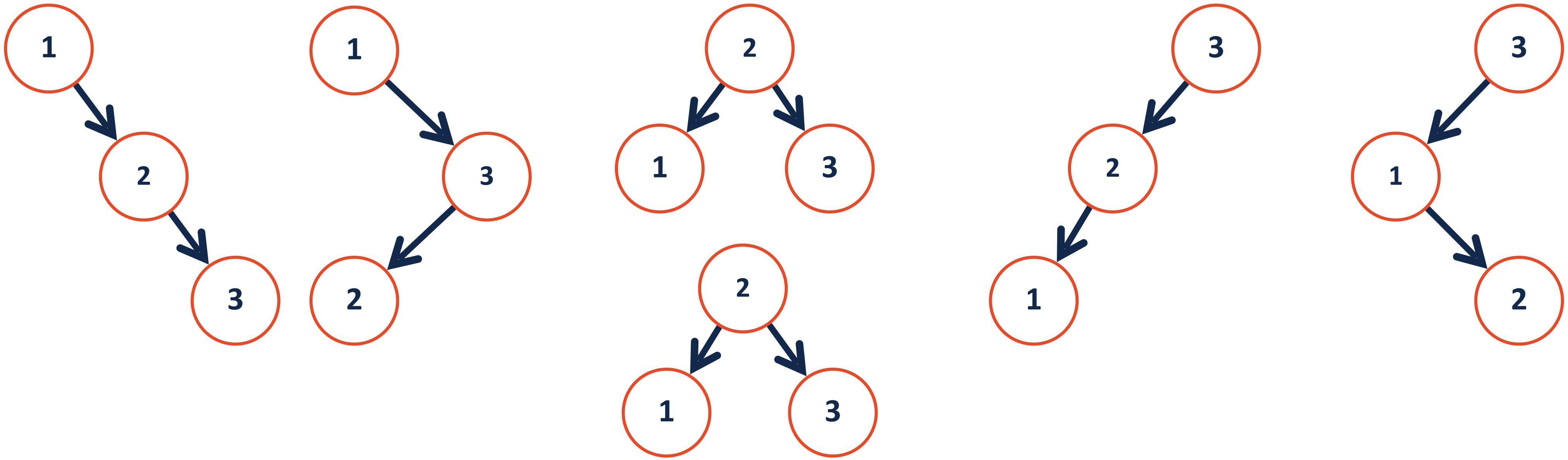
2 3 1

3 2 1

3 1 2

Randomization in Algorithms

1. Assume **input data is random** to estimate average-case performance



$$S(n) = (n-1) + \frac{1}{n}\sum_{i=0}^{n-1} S(i) + S(n-i-1)$$

Randomization in Algorithms

2. Use **randomness inside algorithm** to estimate expected running time

In **randomized quicksort**, the selection of the pivot is random.

Claim: The expected time is $O(n\ log\ n)$ ***for any input!***

1	2	3	4	5	6	7	8

8	7	6	5	4	3	2	1

6	2	1	3	7	8	5	4

Randomization in Algorithms

2. Use **randomness inside algorithm** to estimate expected running time

In **randomized quicksort**, the selection of the pivot is random.

Claim: The expected time is $O(n\ log\ n)$ ***for any input!***

Let X be the total comparisons and X_{ij} be an **indicator variable**:

$$X_{ij} = \begin{cases} 1 & \text{if } i\text{th object compared to } j\text{th} \\ 0 & \text{if } i\text{th object not compared to } j\text{th} \end{cases}$$

Then… $X = \sum_{i=1}^{n} \sum_{j=i+1}^{n} x_{ij}$

Expectation Analysis: Randomized Quicksort

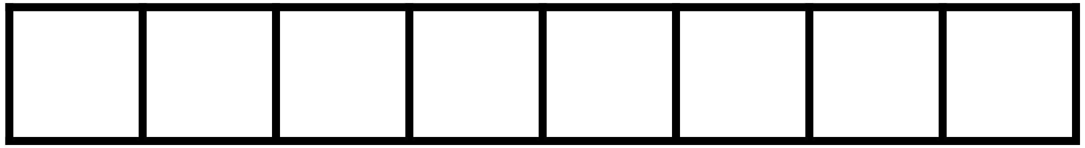
Claim: $E[X_{i,j}] = \dfrac{2}{j - i + 1}.$

Base Case: (N=2)

Expectation Analysis: Randomized Quicksort

Claim: $E[X_{i,j}] = \dfrac{2}{j - i + 1}$

Induction: Assume true for all inputs of $< n$



Expectation Analysis: Randomized Quicksort

$$E[X] = \sum_{i=1}^{n} \sum_{j=i+1}^{n} E[X_{ij}] \qquad E[X_{ij}] = \frac{2}{j - i + 1}$$

Expectation Analysis: Randomized Quicksort

$$E[X] = \sum_{i=1}^{n} \sum_{j=i+1}^{n} E[X_{ij}] \qquad E[X_{ij}] = \frac{2}{j - i + 1}$$

$$E[X] = \sum_{i=1}^{n} 2\left(\frac{1}{2} + \frac{1}{3} + \ldots + \frac{1}{n - i + 1}\right)$$

$$E[X] = \sum_{i=1}^{n} 2(H_{n-1} - 1) \leq 2n \cdot H_n \leq 2n \; ln \; n$$

Expectation Analysis: Randomized Quicksort

Summary: Randomized quick sort is $O(n\ log\ n)$ regardless of input

Randomness:

Assumptions:

Probabilistic Accuracy: Fermat primality test

Pick a random a in the range $[2,\ p-2]$

If p is prime and a is not divisible by p, then $a^{p-1} \equiv 1 \pmod{p}$

But… ***sometimes*** if n is composite and $a^{n-1} \equiv 1 \pmod{n}$

Probabilistic Accuracy: Fermat primality test

	$a^{p-1} \equiv 1 \pmod{p}$	$a^{p-1} \not\equiv 1 \pmod{p}$
p is prime		
p is not prime		

Probabilistic Accuracy: Fermat primality test

Let’s assume $\alpha = .5$

First trial: $a = a_0$ and prime test returns ‘prime!’

Second trial: $a = a_1$ and prime test returns ‘prime!’

Third trial: $a = a_2$ and prime test returns ‘not prime!’

Is our number prime?

What is our **false positive** probability? Our **false negative** probability?

Probabilistic Accuracy: Fermat primality test

Summary: Randomized algorithms can also have fixed (or bounded) runtimes at the cost of probabilistic accuracy.

Randomness:

Assumptions:

Types of randomized algorithms

A **Las Vegas** algorithm is a randomized algorithm which will always give correct answer if run enough times but has no fixed runtime.

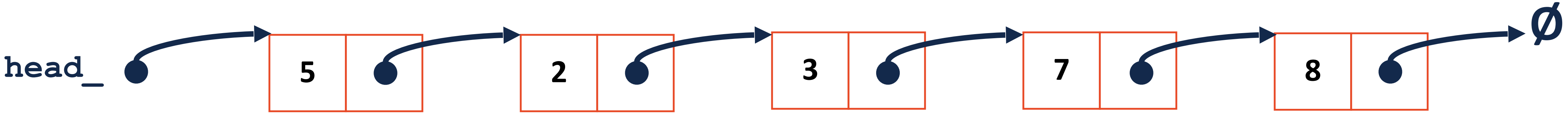
A **Monte Carlo** algorithm is a randomized algorithm which will run a fixed number of iterations and may give the correct answer.

Randomized Data Structures

Sometimes a data structure can be **too ordered / too structured**

Randomized data structures rely on **expected** performance

Linked List

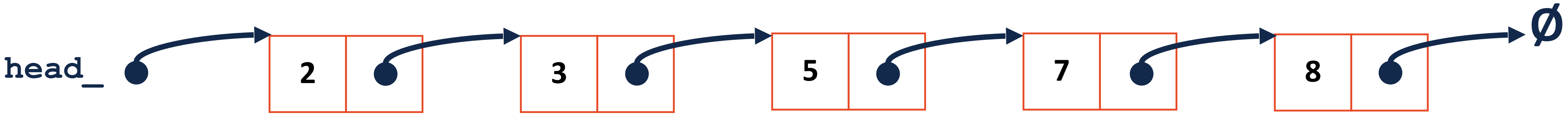


Pros:

Cons:

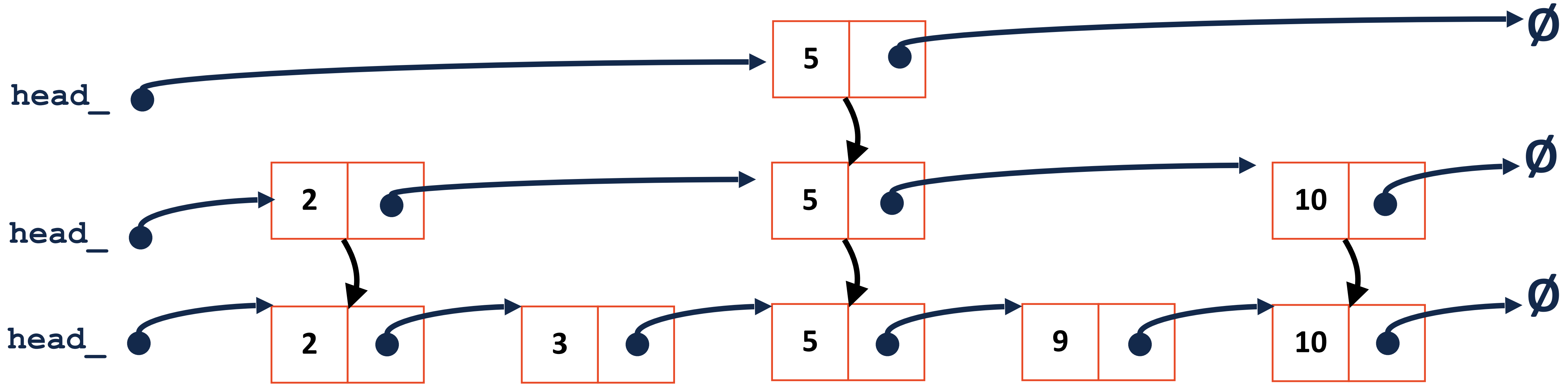
Linked List with ‘Checkpoints’

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



Linked List with Perfect Checkpoints

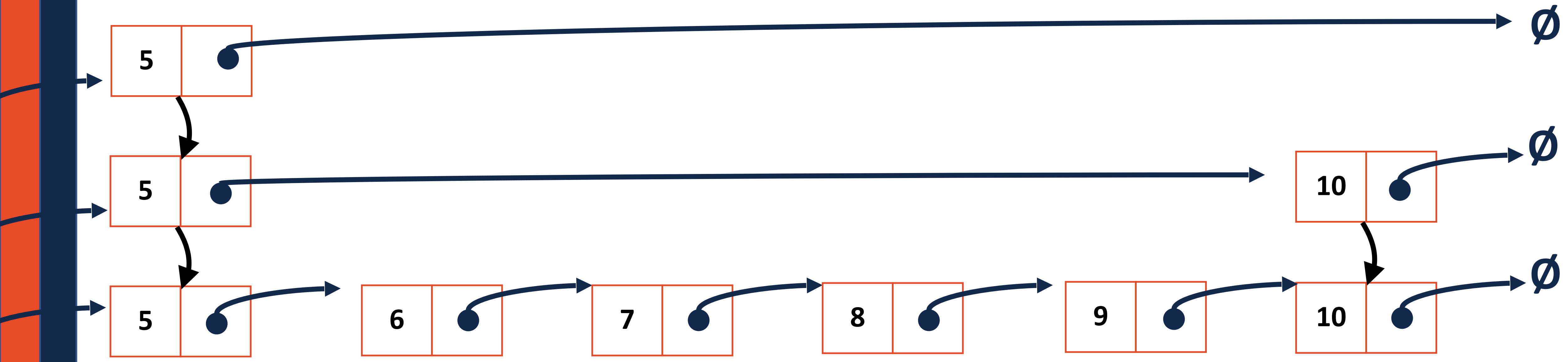
For optimal checkpoints, we want half the number of items at each level.



Linked List with Perfect Checkpoints

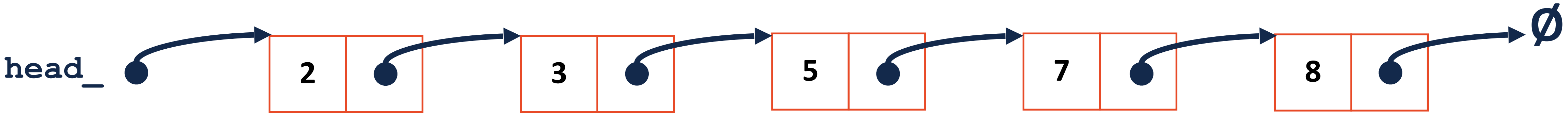
For optimal checkpoints, we want half the number of items at each level.

Maintaining this while inserting and deleting is too costly!



Linked List with Random Checkpoints

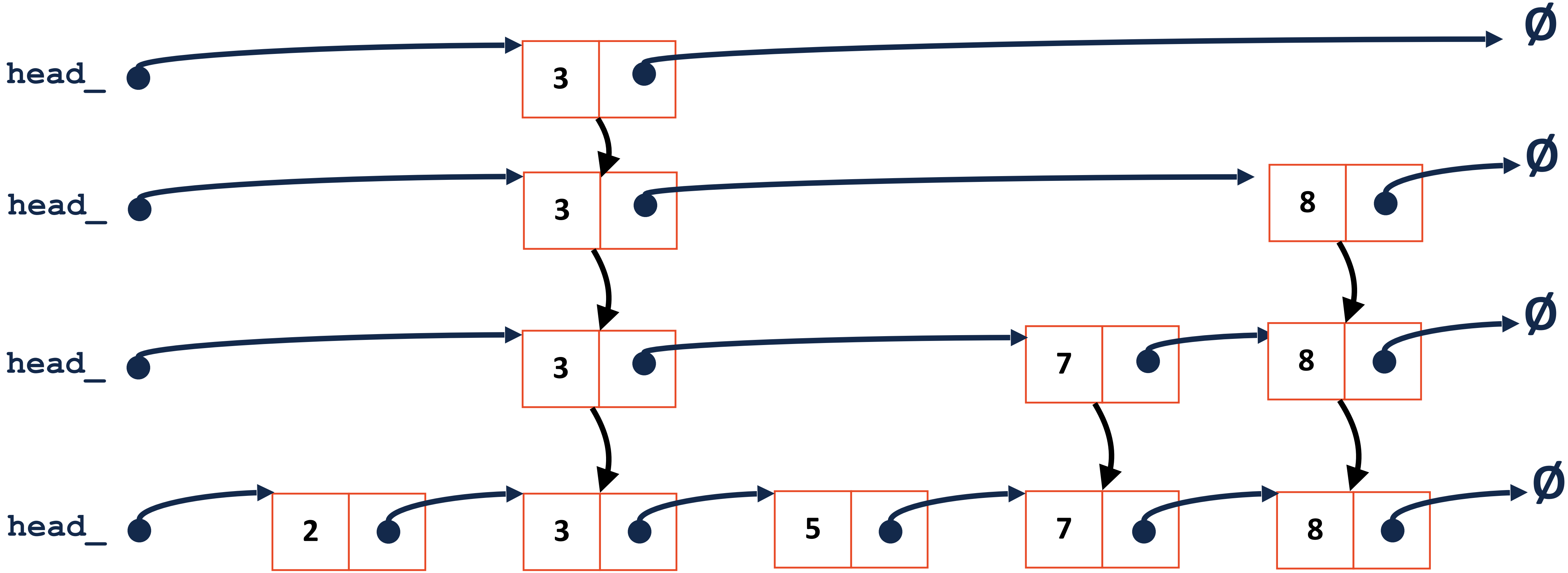
Instead of having **exactly** half each level, let's have **approximately** half!



Linked List with Random Checkpoints

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

Worst Case Performance:



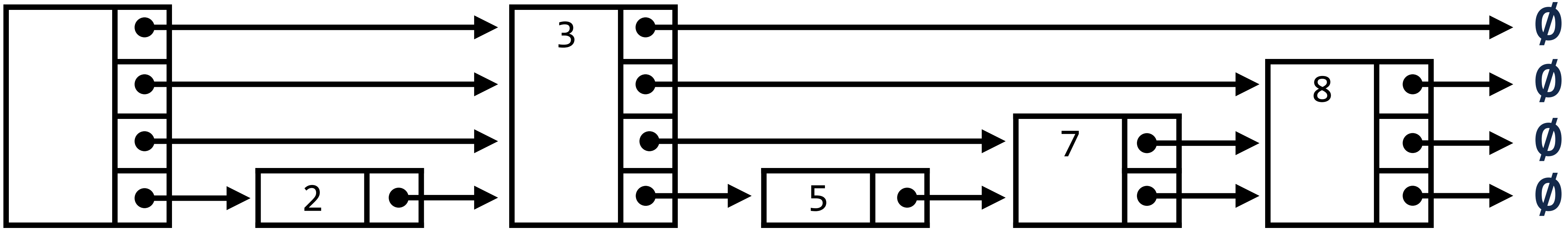
The Skip List

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 $O(log\ n)$



Skip List

```
template <class T>
class SkipList{
  public:
    class SkipNode{
      public:
        SkipNode(){
          next.push_back(nullptr);
        }

        SkipNode(int h, T & d){
          data = d;
          for(int i = 0; i <= h; i++){
            next.push_back(nullptr);
          }
        }
        T data;
        std::vector<SkipNode*> next;
    };

    int max; // max height
    float c; //update constant
    SkipNode* head;
    ...

```