Data Structures

Array Lists

CS 225

September 6, 2023

Brad Solomon & G Carl Evans

UNIVERSITY OF ILLINOIS URBANA-CHAMPAIGN

Department of Computer Science

Exam 1 Practice Exam Available

Practice exams give a rough idea of the format and style of questions

They are not exhaustive nor meaningfully repeatable

Learning Objectives

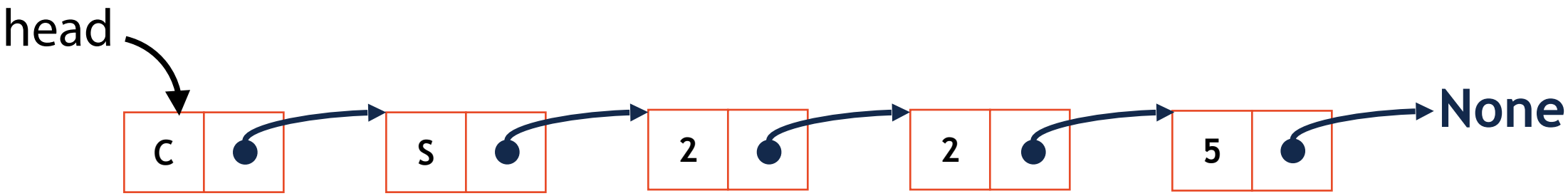
Review array list implementation

Discuss array resizing

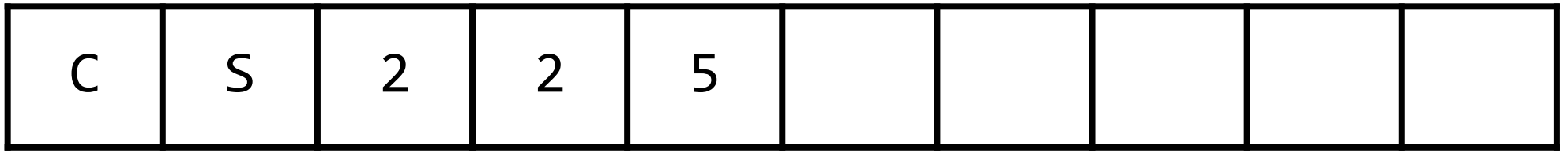
Consider extensions to lists

List Implementations

1. Linked List

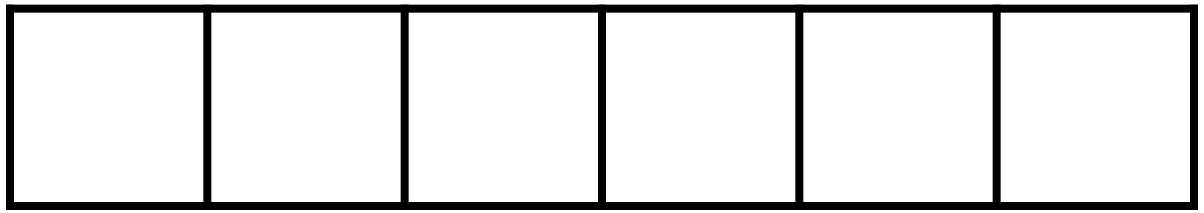


2. Array List

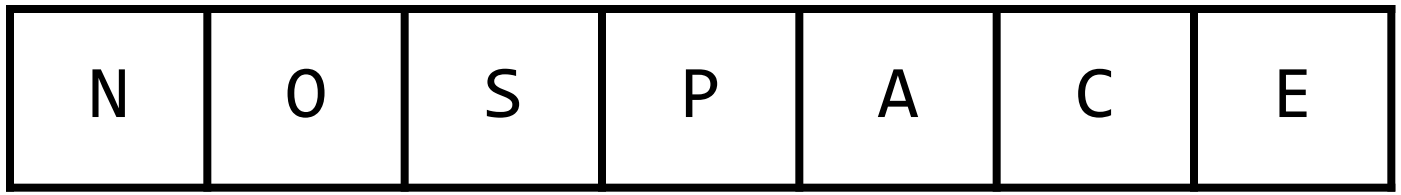


List.h

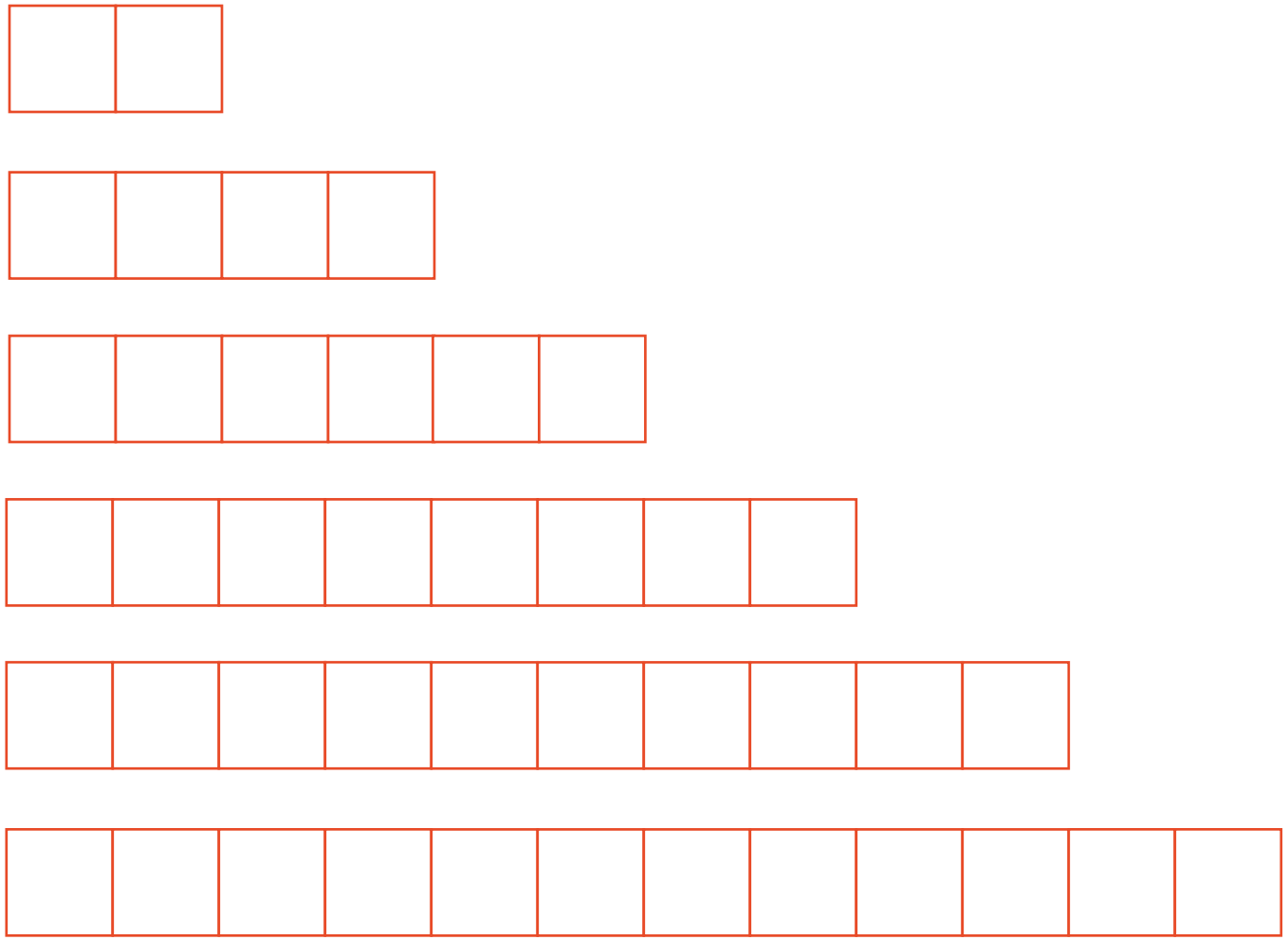
```
 1  #pragma once
 2
 3  template <typename T>
 4  class List {
 5  public:
…       /* --- */
25  private:
26    T *data_;
27
28    T *size;
29
30    T *capacity;
…
        /* --- */
    };
```



Array List: _addspace()

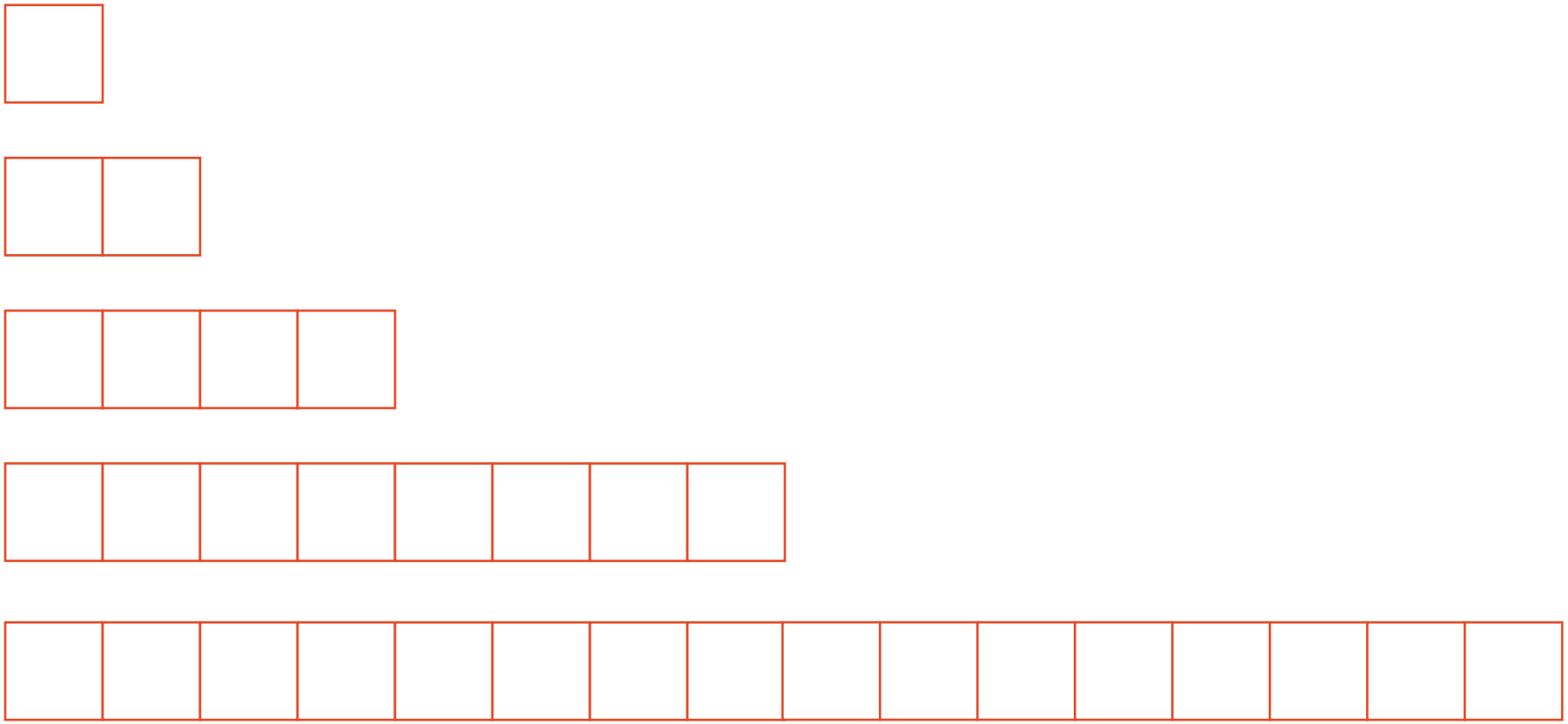


Resize Strategy: +2 elements every time



Resize Strategy: +2 elements every time

Resize Strategy: x2 elements every time



Resize Strategy: x2 elements every time

Array Implementation

	Singly Linked List	Array
Look up **arbitrary** location		
Insert after **given** element		
Remove after **given** element		
Insert at **arbitrary** location		
Remove at **arbitrary** location		
Search for an input **value**		

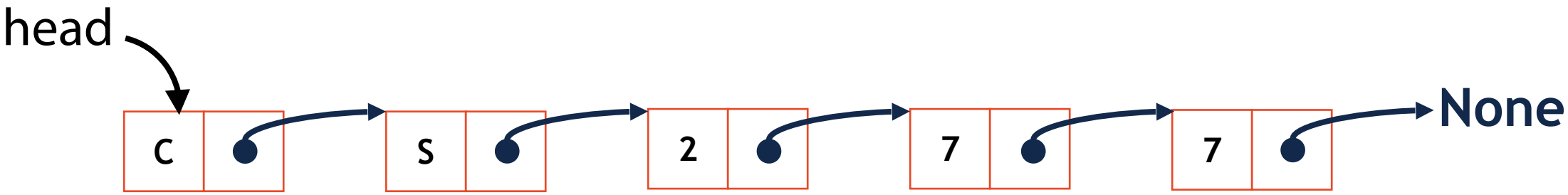
Thinking critically about lists: tradeoffs

The implementations shown are foundational.

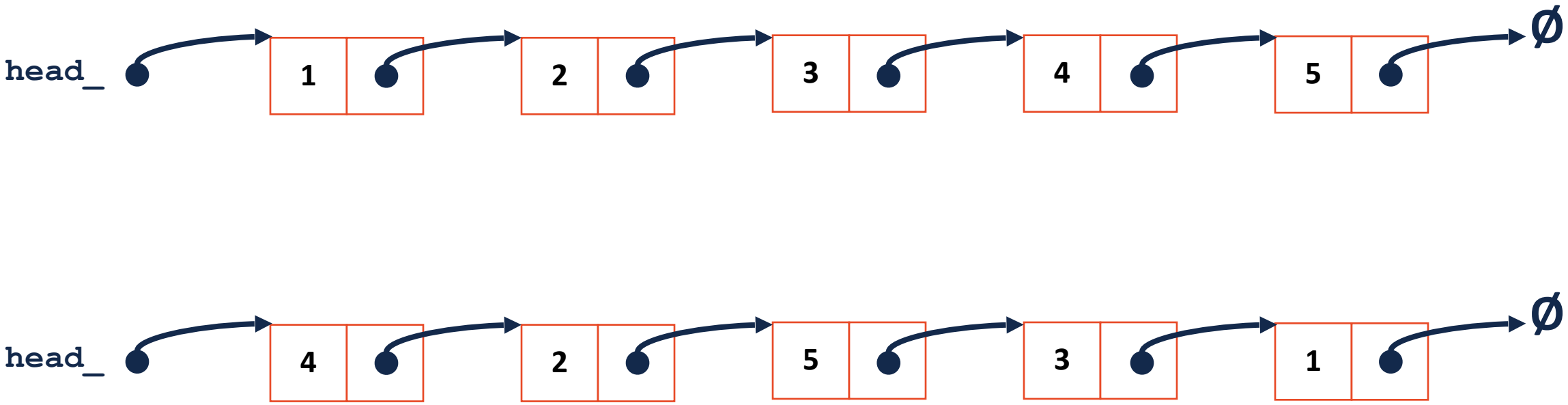
Can we make our lists better at some things? What is the cost?

Thinking critically about lists: tradeoffs

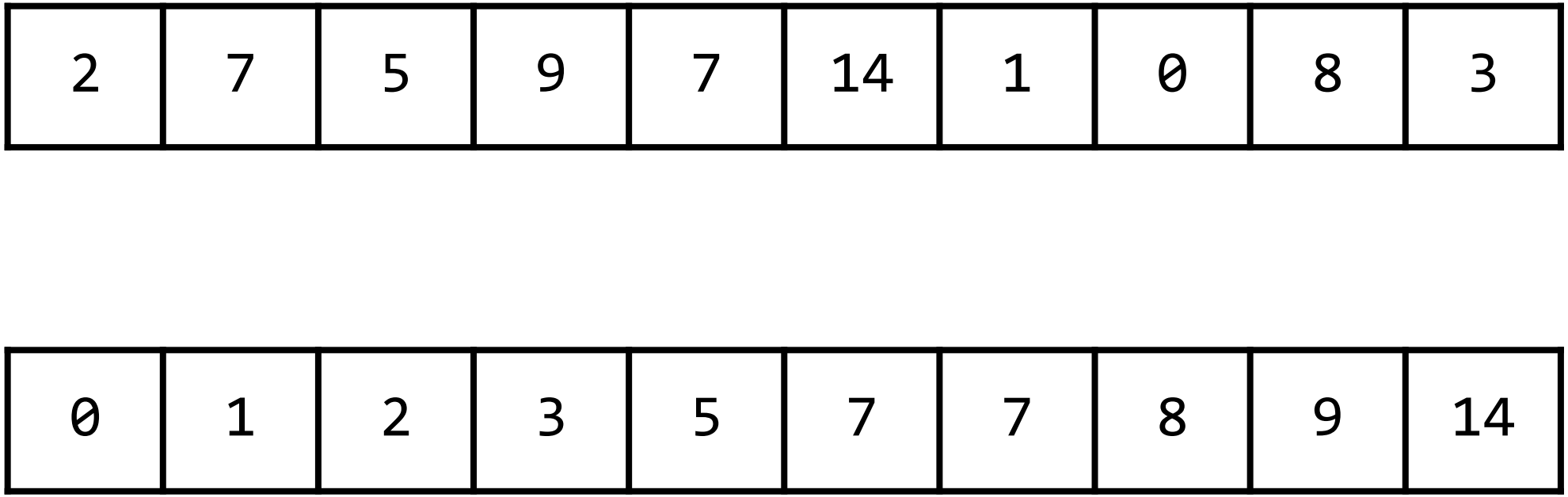
Getting the size of a linked list has a Big O of:



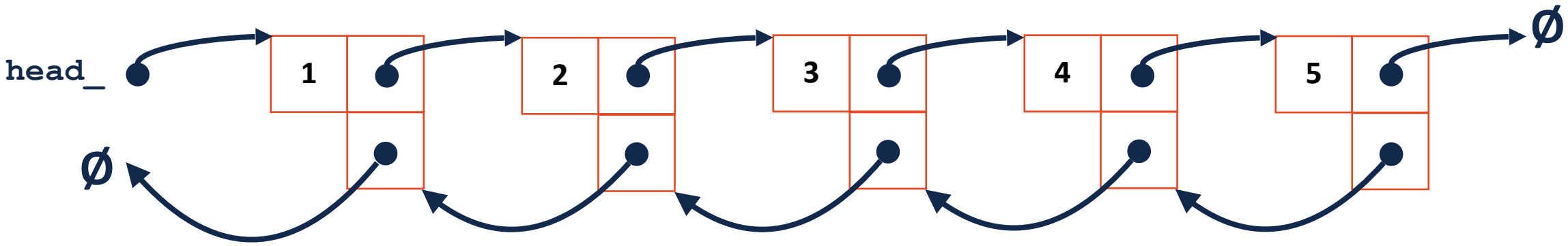
Thinking critically about lists: tradeoffs



Thinking critically about lists: tradeoffs



Thinking critically about lists: tradeoffs



Thinking critically about lists: tradeoffs

When we discuss data structures, consider how they can be modified or improved!

Can we make a ‘list’ that is O(1) to insert and remove? What is our tradeoff in doing so?

Stack Data Structure

A **stack** stores an ordered collection of objects (like a list)

However you can only do two operations:

Push: Put an item on top of the stack

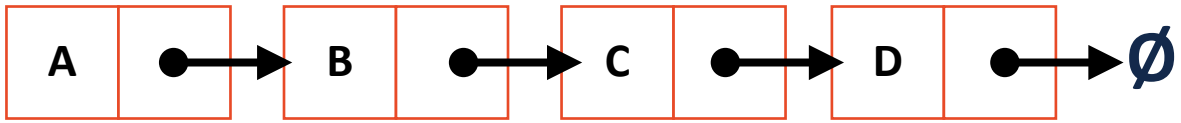
Pop: Remove the top item of the stack (and return it)


```
push(3); push(5); pop(); push(2)
```

Stack Data Structure

The **call stack** is a key concept for understanding recursion

```
63  template <typename T>
64  typename List<T>::ListNode *& List<T>::_index(unsigned index, ListNode *& root){
65
66      if (index == 0){ return root; }
67      if (root == nullptr){ return root; }
68
69      return _index(index - 1, root -> next);
70
71  }
```

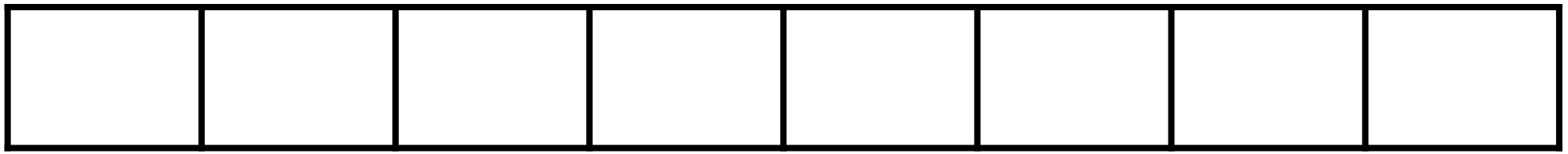


Stack Data Structure

C++ has a built-in stack

Underlying implementation is vector or deque

```
1  #include <stack>
2  int main() {
3    stack<int> stack;
4    stack.push(3);
5    stack.push(8);
6    stack.push(4);
7    stack.pop();
8    stack.push(7);
9    stack.pop();
10   stack.pop();
11   stack.push(2);
12   stack.push(1);
13   stack.push(3);
14   stack.push(5);
15   stack.pop();
16   stack.push(9);
17 }
18
19
20
```



Stack ADT

- [Order]:

- [Implementation]:

- [Runtime]:

Queue Data Structure

A **queue** stores an ordered collection of objects (like a list)

However you can only do two operations:

Enqueue: Put an item at the back of the queue

Dequeue: Remove and return the front item of the queue


```
enqueue(3); enqueue(5); dequeue(); enqueue(2)
```