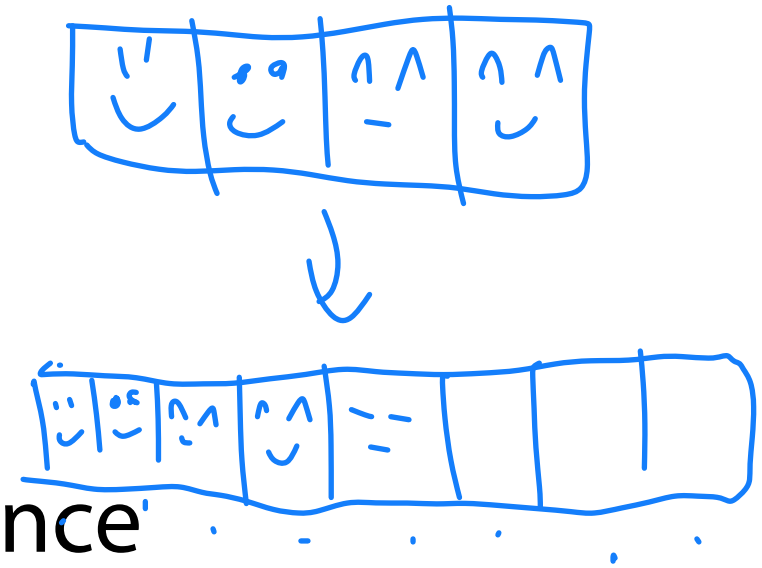
Data Structures

Array Lists

CS 225

Brad Solomon

September 9, 2024



UNIVERSITY OF ILLINOIS URBANA-CHAMPAIGN

Department of Computer Science

Post your art on #mp-art!

Exam 1 (9/18 — 9/20)

Cover up to today! 9/9

Autograded MC and one coding question

Manually graded short answer prompt

Practice exam will be released on PL

Topics covered can be found on website

Registration started August 22

https://courses.engr.illinois.edu/cs225/fa2024/exams/

Learning Objectives

Discuss data variables for implementing array lists

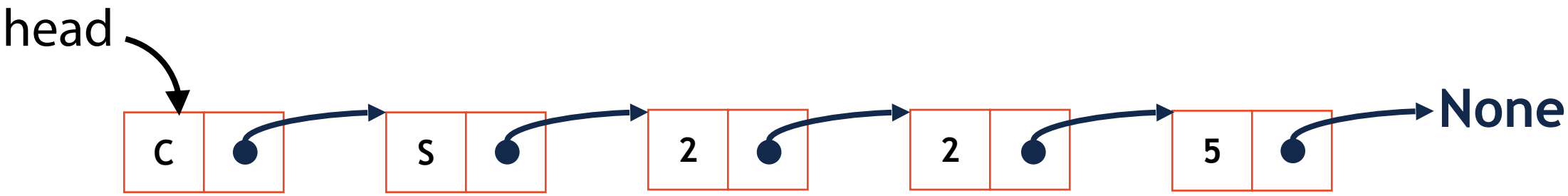
review

~~Introduce~~ array list implementations

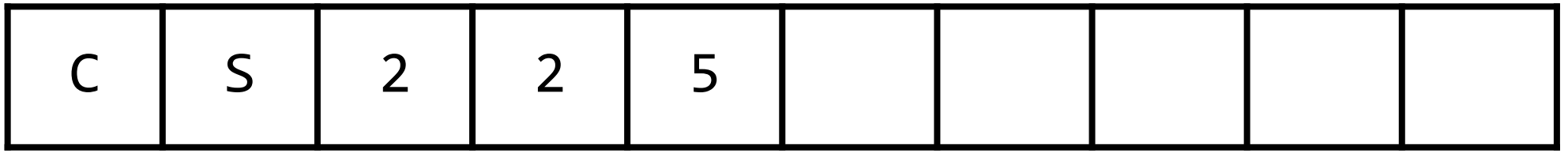
Consider extensions to lists

List Implementations

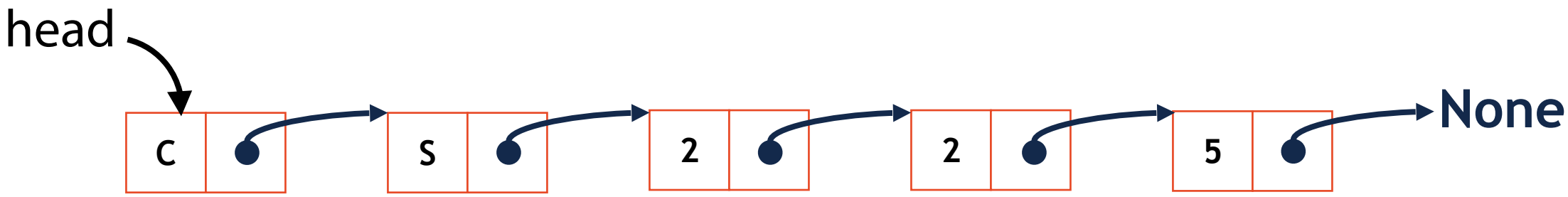
1. Linked List



2. Array List



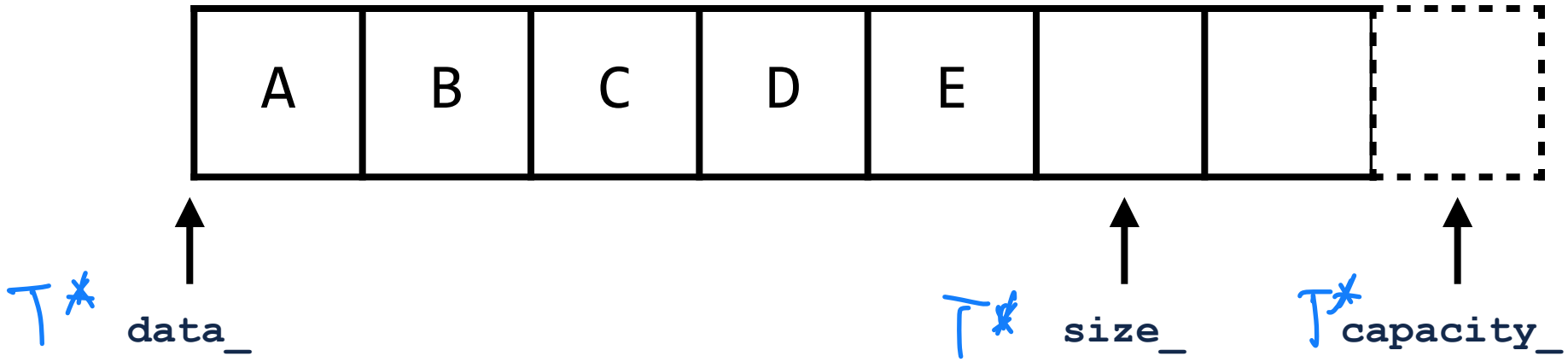
Linked List Runtimes



	@Front	@RefPointer	@Index
Insert	O(1)	O(1)	O(n)
Delete	O(1)	O(1)	O(n)

↳ Find O(n)
↳ Modification O(1)

Array List



In C++, vector is implemented as:

1) **Data:** Stored as a pointer to array start

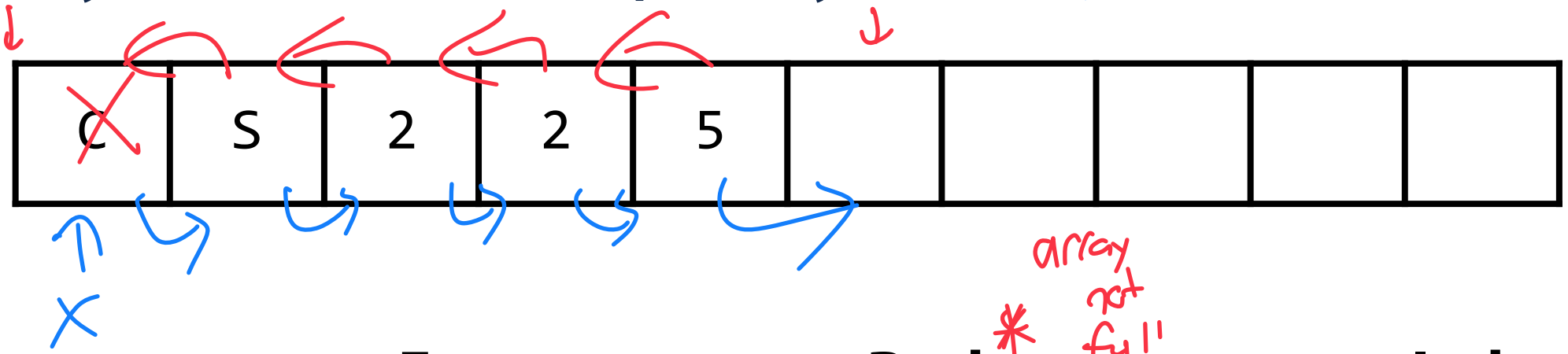
2) **Size:** Stored as a pointer to the next available space

3) **Capacity:** Stored as a pointer past the end of the array

Array List: Not at capacity

insert(x)

capacity

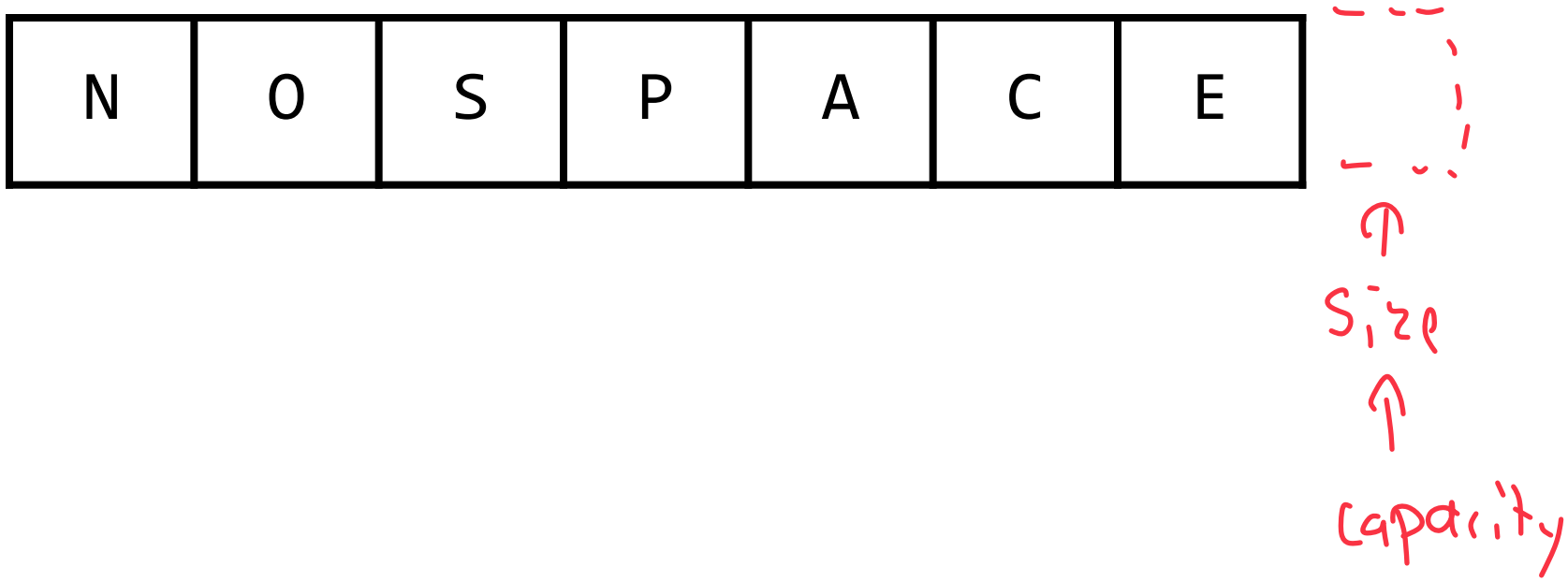


array not full

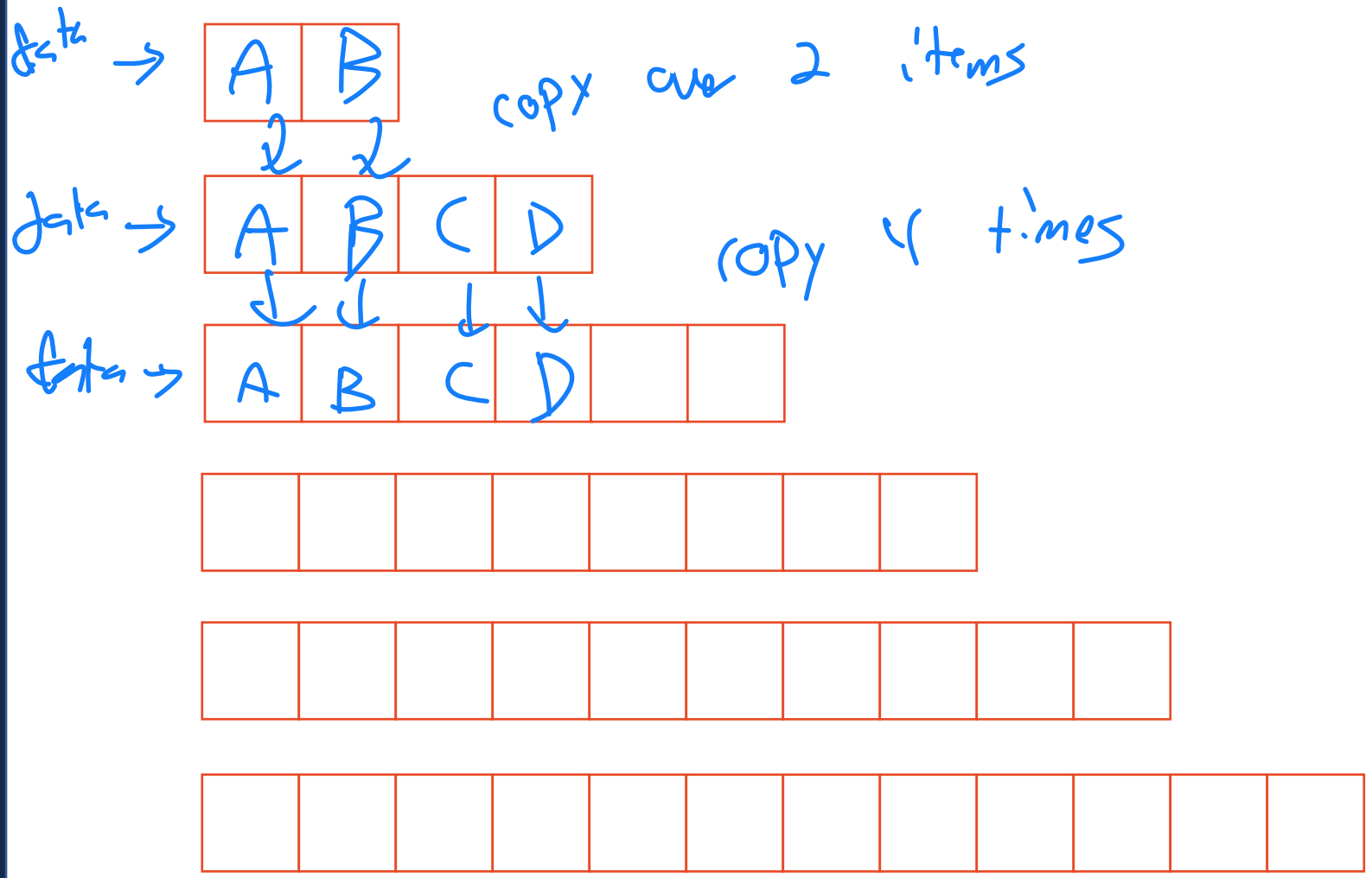
*

	@Front	@Back	@Index
Insert	O(n)	O(1)	O(n)
Delete	O(n)	O(1)	O(n)

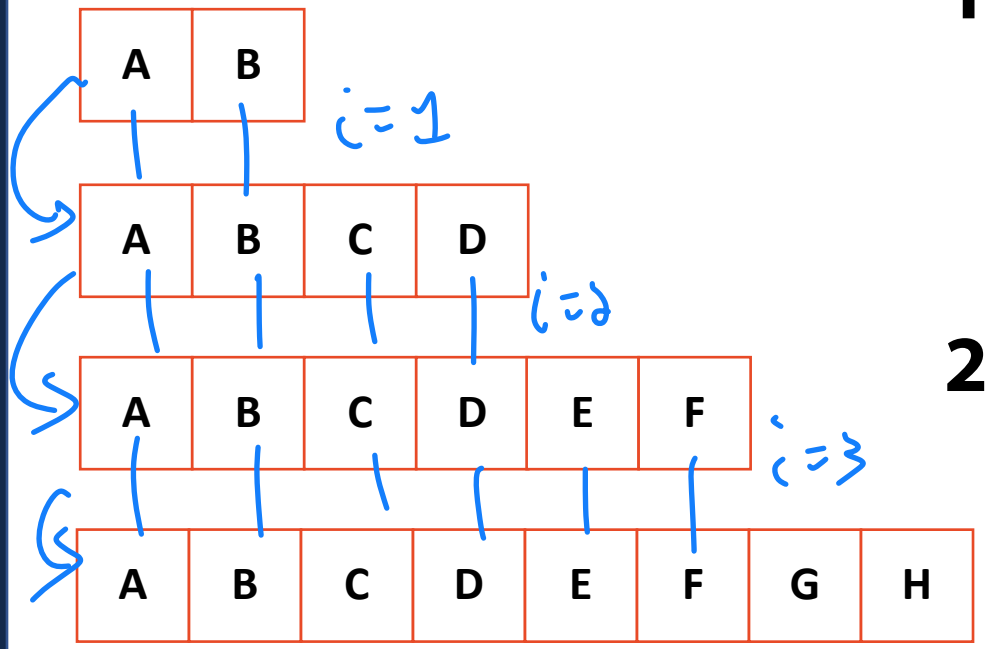
Array List: addspace(data)



Resize Strategy: +2 elements every time



Resize Strategy: +2 elements every time



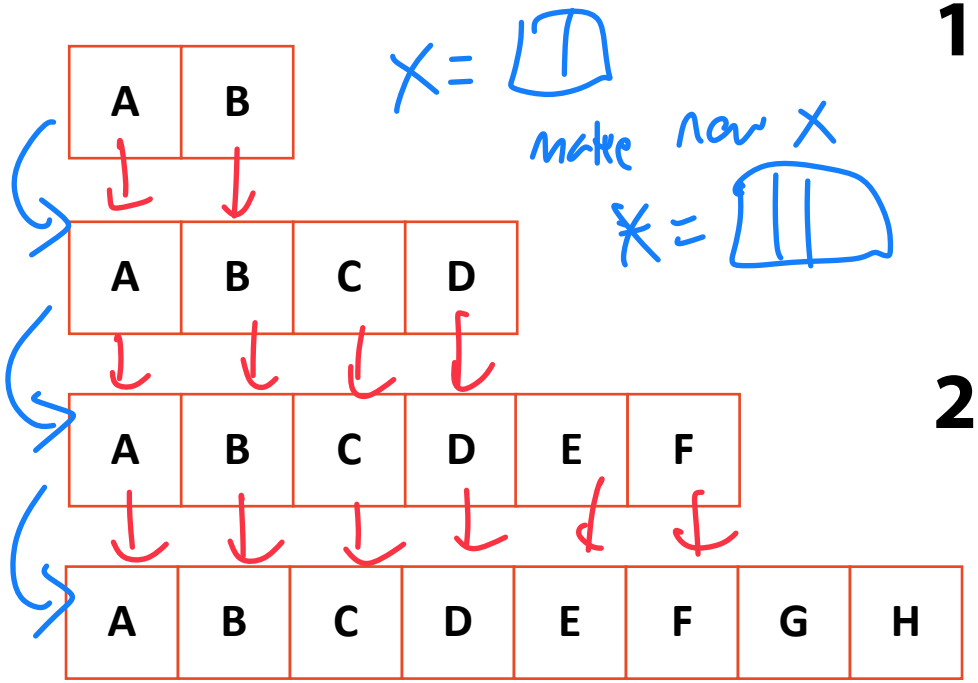
1) How many copy calls per reallocation?

At iteration i, $2i$ copies

2) Total reallocations for N objects?

$K = N/2$

Resize Strategy: +2 elements every time



1) How many copy calls per reallocation?

For reallocation i, 2i copy calls are made

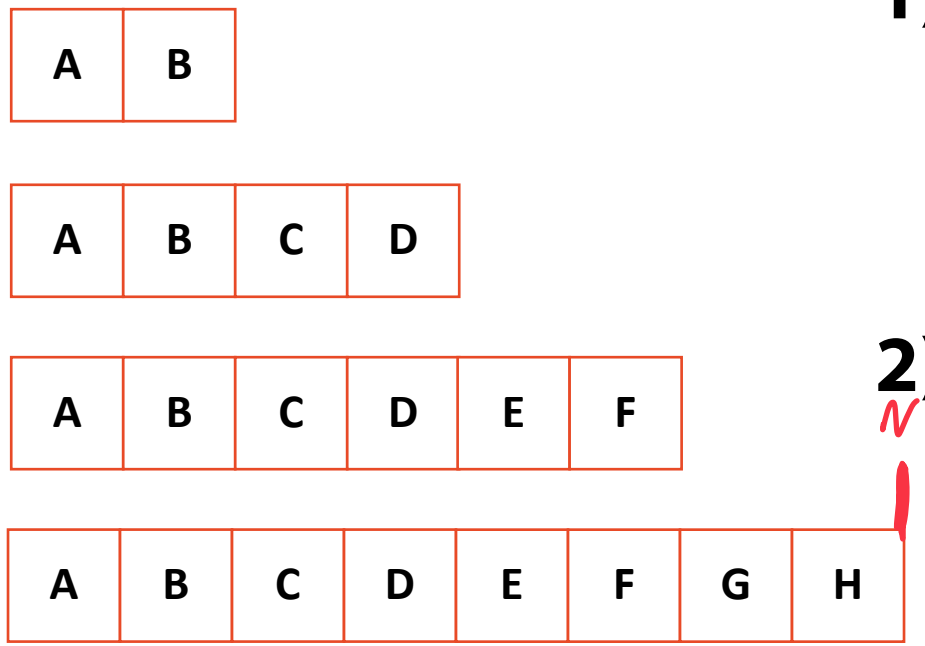
2) Total reallocations for N objects?

Let k be the number of reallocs, $k = \frac{N}{2}$

Total number of copy calls:

$$\sum_{i=1}^{k} 2i = k(k+1) = k^2 + k$$

Resize Strategy: +2 elements every time



1) How many copy calls per reallocation?

For reallocation i, 2i copy calls are made

2) Total reallocations for N objects?

Let k be the number of reallocs, $k = \frac{N}{2}$

Total number of copy calls:

$$\sum_{i=1}^{k} 2i = k(k+1) = k^2 + k$$

$k = N/2 \quad \Rightarrow \quad \frac{N^2}{4} + \frac{N}{2}$

… For N objects: $\frac{N^2 + 2N}{4}$

Resize Strategy: +2 elements every time

Aug case is bad

↳ is not amortized!

Total copies for N inserts: $\frac{N^2 + 2N}{4}$

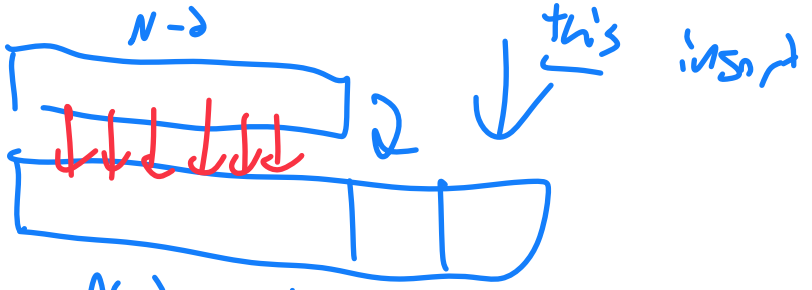
Amortized: Total work over N inserts

↳ $\frac{N^2 + 2N}{4N} \Rightarrow O(N)$

Divide by N to get amortized for one insert

Estimate work for one insert, compare & contrast

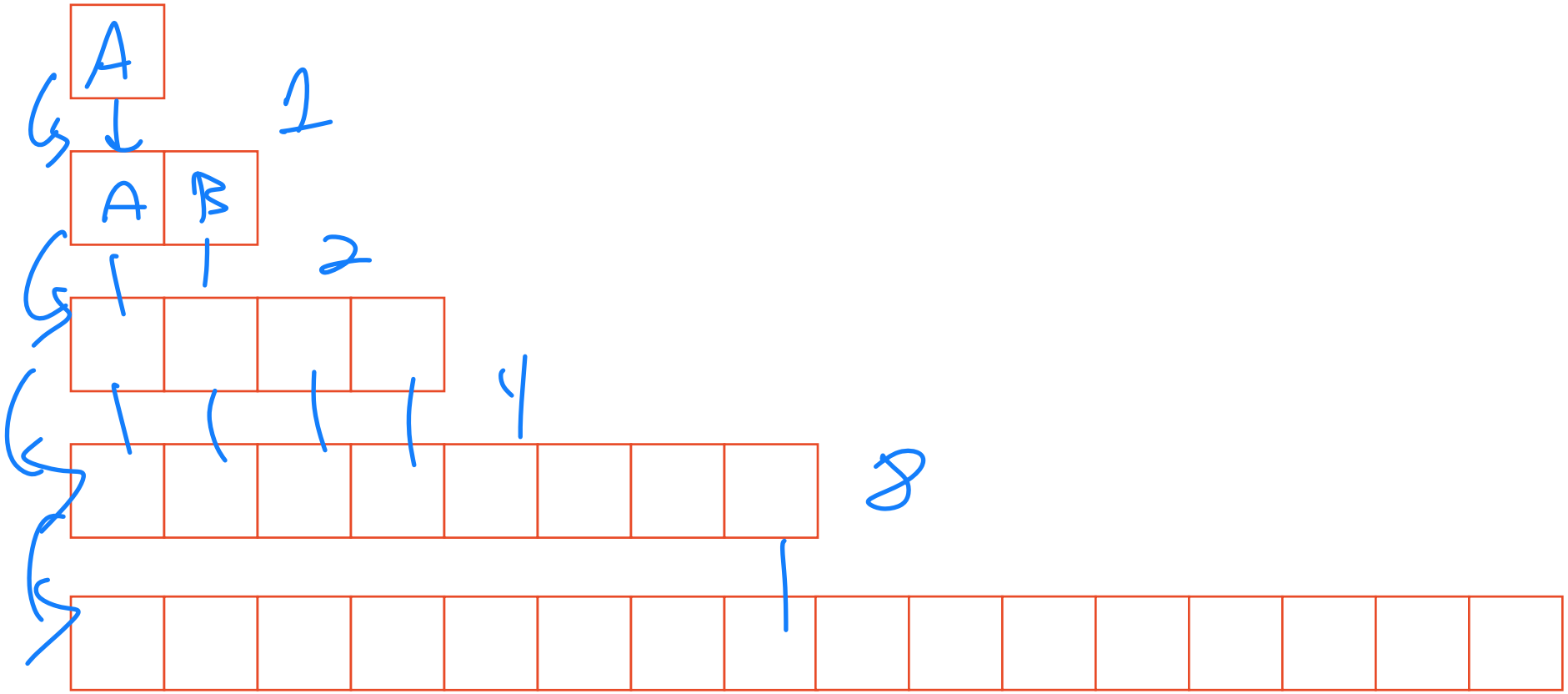
Big O: $O(N)$



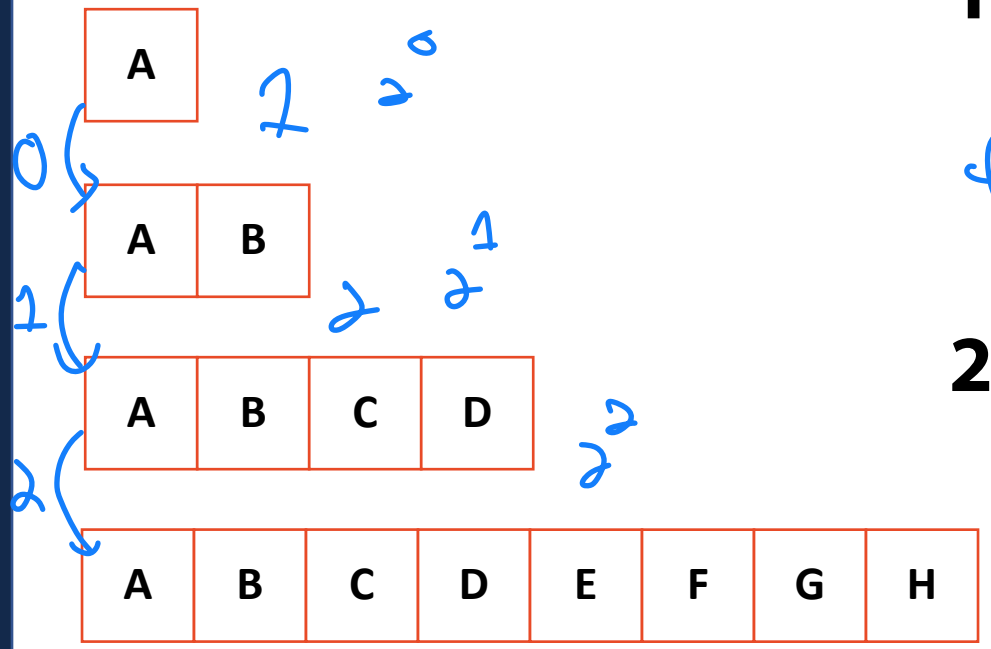
$O(1)$ for this insert

N-1 th insert $O(N)$

Resize Strategy: x2 elements every time



Resize Strategy: x2 elements every time



1) How many copy calls per reallocation?

for realloc i, 2^i copies

More copies

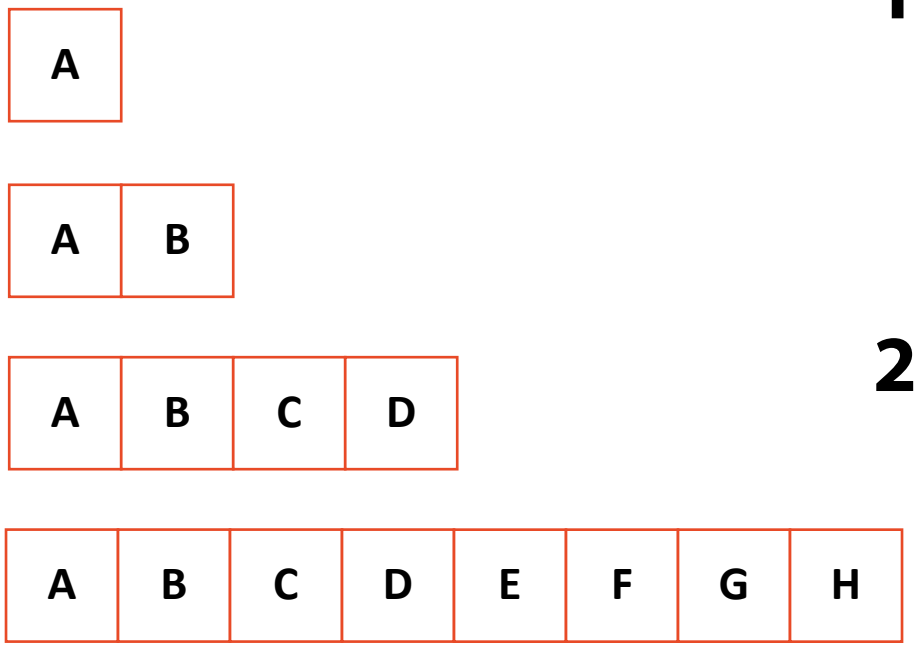
2) Total reallocations for N objects?

$N \leq 2^K \rightarrow K = \lceil \log_2 N \rceil$

Less realloc

is larger or equal to N

Resize Strategy: x2 elements every time



1) How many copy calls per reallocation?

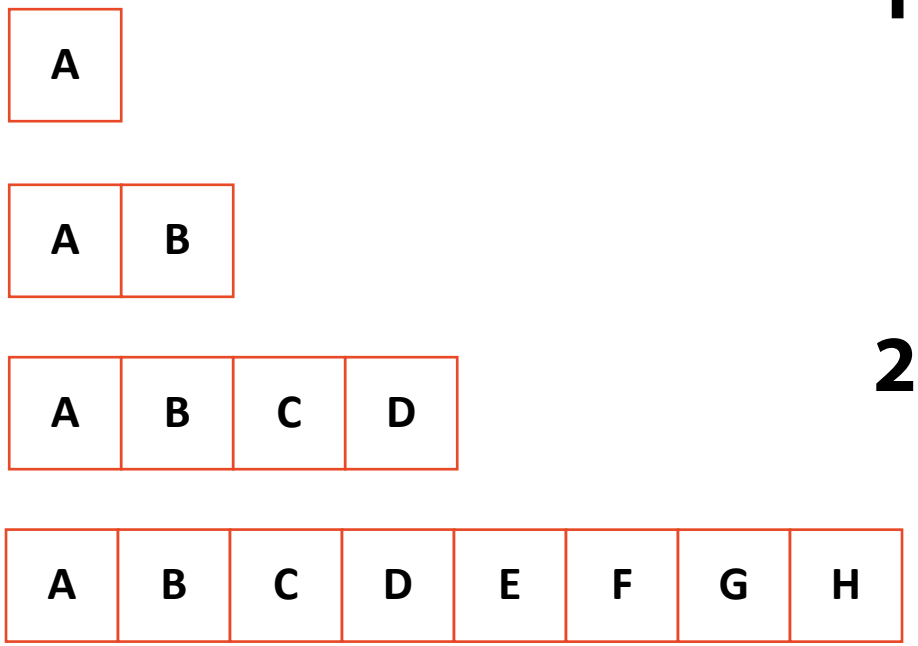
For reallocation i, 2^i copy calls are made

2) Total reallocations for N objects?

k = final realloc needed = $\lceil log_2 n \rceil$

Total number of copy calls: $\sum_{i=0}^{k} 2^i = 2^{k+1} - 1$

Resize Strategy: x2 elements every time



1) How many copy calls per reallocation?

For reallocation i, 2^i copy calls are made

2) Total reallocations for N objects?

k = final realloc needed = $\lceil log_2 n \rceil$

Total number of copy calls:

$$\sum_{i=0}^{k} 2^i = 2^{k+1} - 1$$

$k = \log_2 n$

$2^{\log_2 n + 1} - 1$

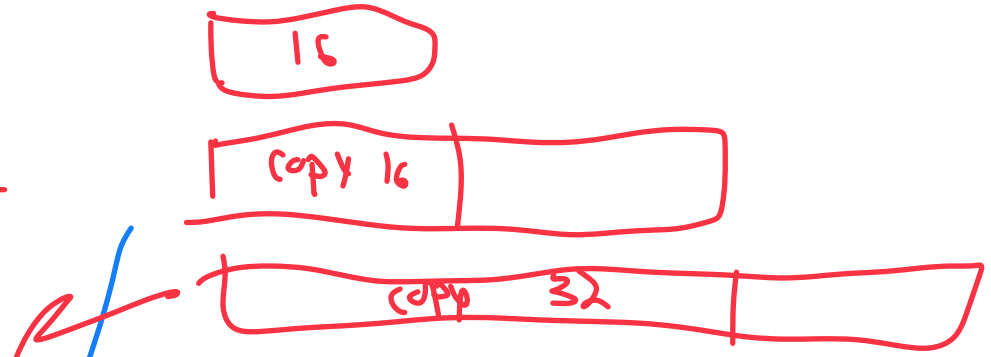
$2 \cdot 2^{\log_2 n} - 1$

n

… For N objects: $2n - 1$

Resize Strategy: x2 elements every time

Total copies for n inserts: $2n - 1$



Amortized:

To do one insert

$\frac{2n-1}{n}$

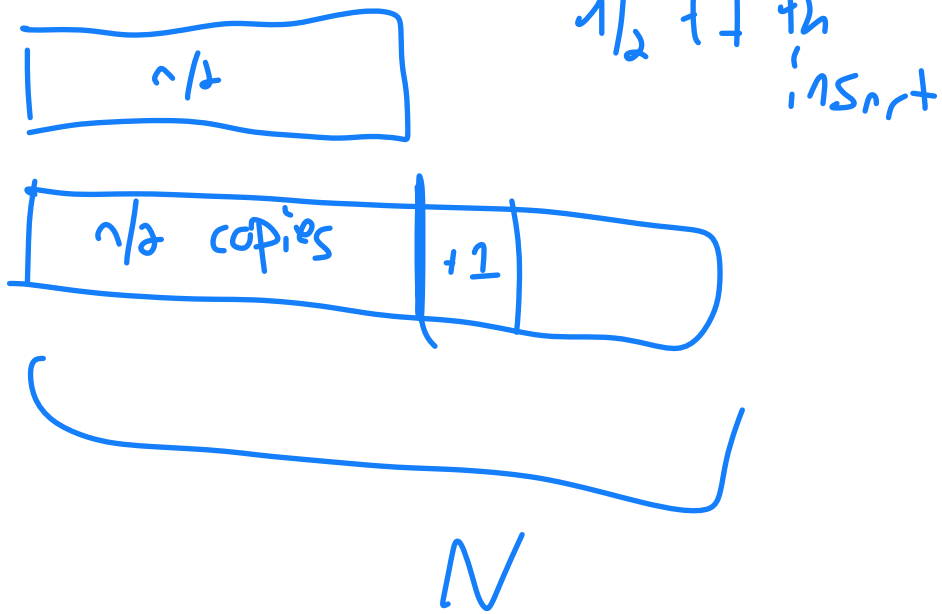
is $O(1)^*$

Diff btwn amortized / average?

- Assumptions made

(No prob distribution here!)

Big O: $O(n)$



1/2 + 1 th insert

List Implementation

	Singly Linked List	Array
Look up **arbitrary** location ↳ index	O(n)	O(1) :)
Insert after **given** element ↳ ref pointer	O(1) :)	O(n)
Remove after **given** element	O(1) :)	O(n)
Insert at **arbitrary** location	Find O(n) change O(1) O(n)	Find O(1) change O(n) O(n)
Remove at **arbitrary** location	O(n)	O(n)
Search for an input **value**	O(n)	O(n)

Special cases

insert/remove front O(1) always

if array not full insert/remove back is O(1) amortized*

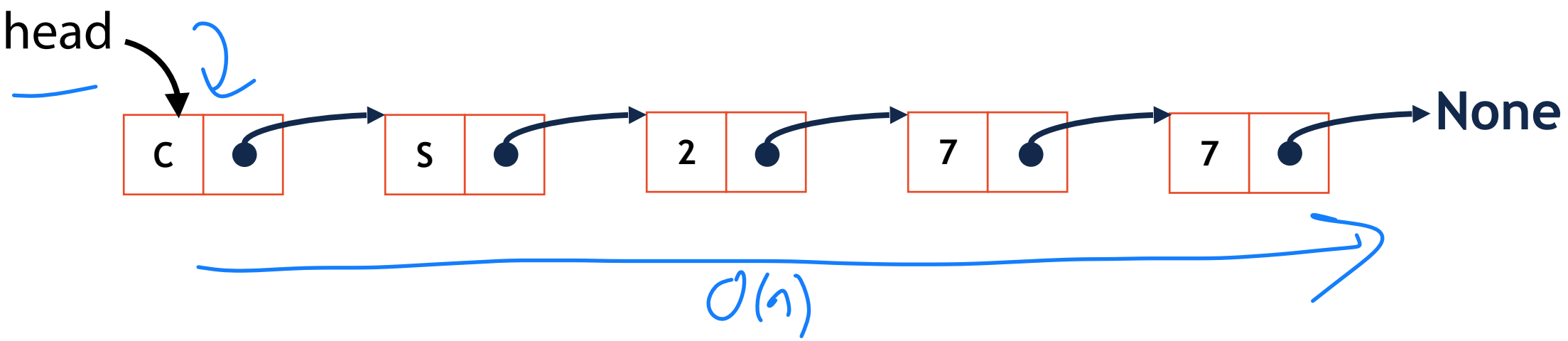
Thinking critically about lists: tradeoffs

The implementations shown are foundational (simple).

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: O(n)

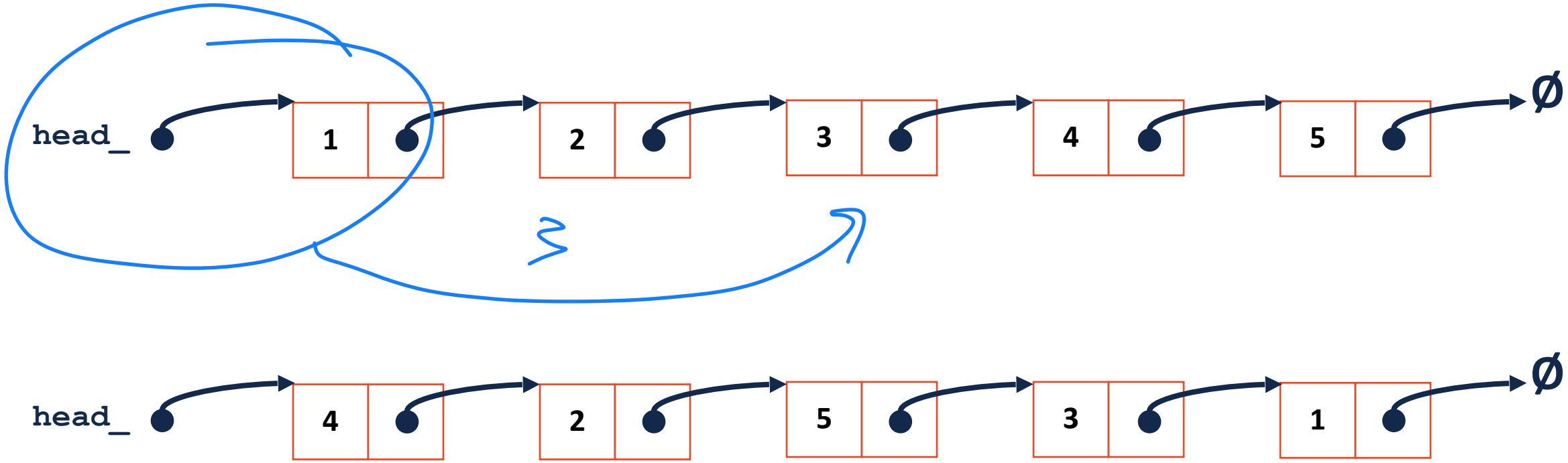


why not add size integer to List class?

+1 int in memory cost

+1 op to increment/dec on insert/remove

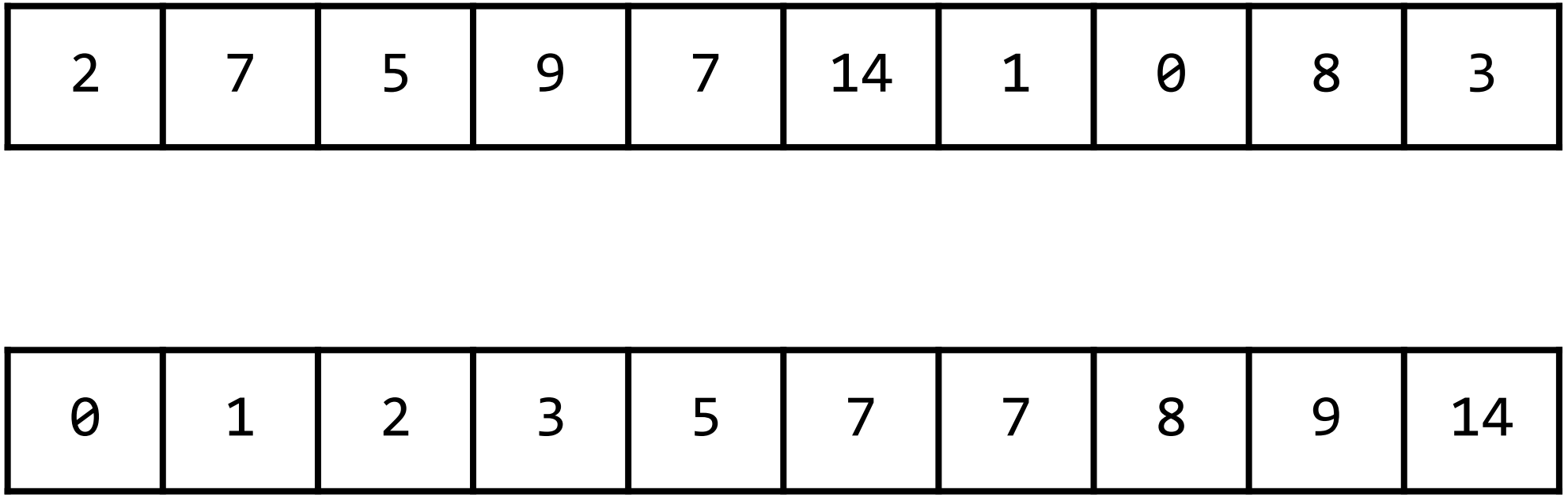
Thinking critically about lists: tradeoffs



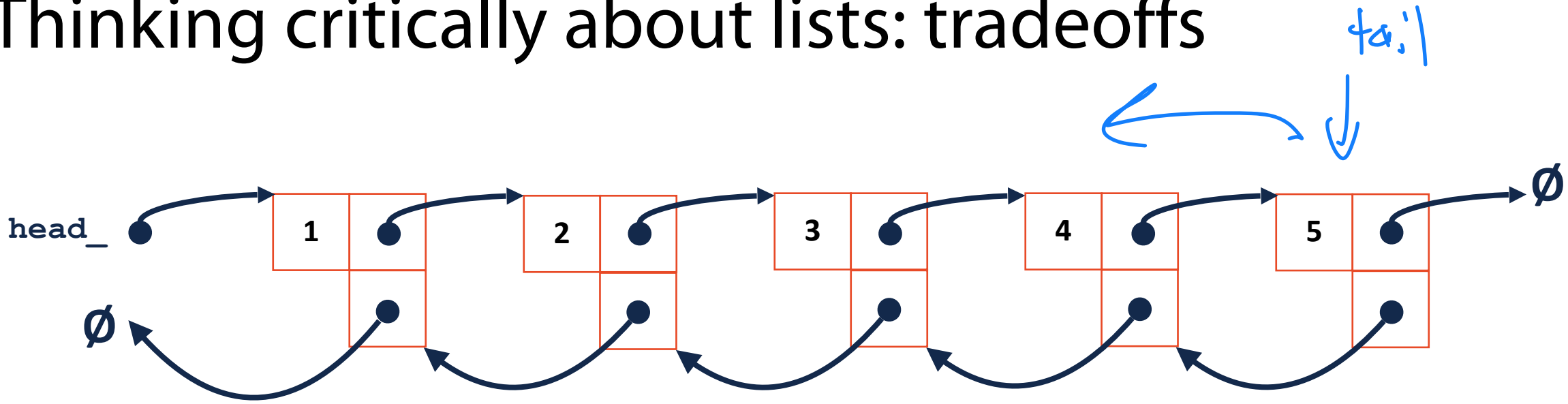
if I only want to touch the min val

is each min val in order

Thinking critically about lists: tradeoffs



Thinking critically about lists: tradeoffs



O(n) Memory increase

Size = #

Thinking critically about lists: tradeoffs

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

Next time: Can we make a 'list' that is O(1) to insert and remove? What is our tradeoff in doing so?