

Stacks and Queues

TRAJECTORY

- **Memory allocation and Contiguous Arrays**
- **Stacks**
- **Data Structures vs Abstract Data Types**
- **Queues**

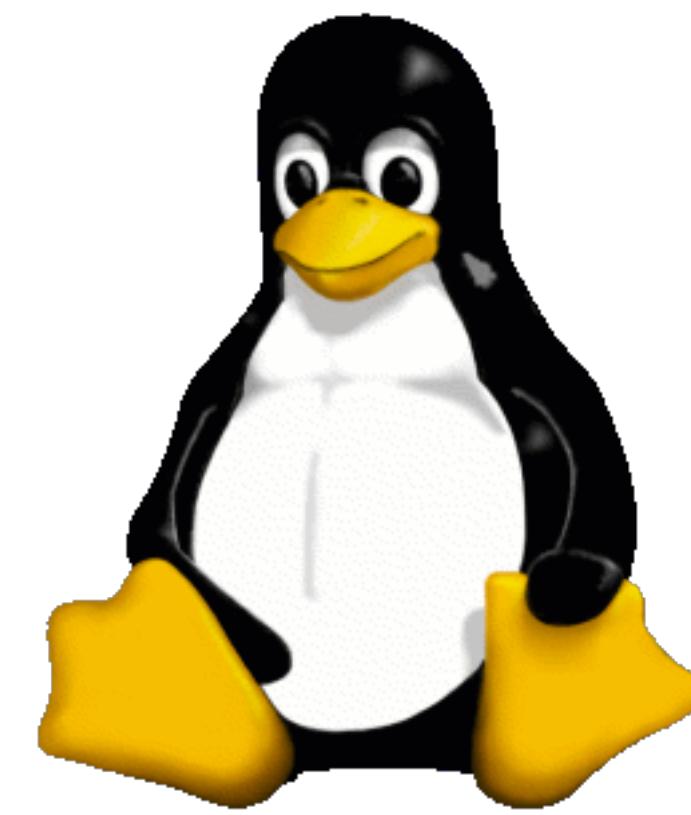
MEMORY



Your application

Operating System

Memory

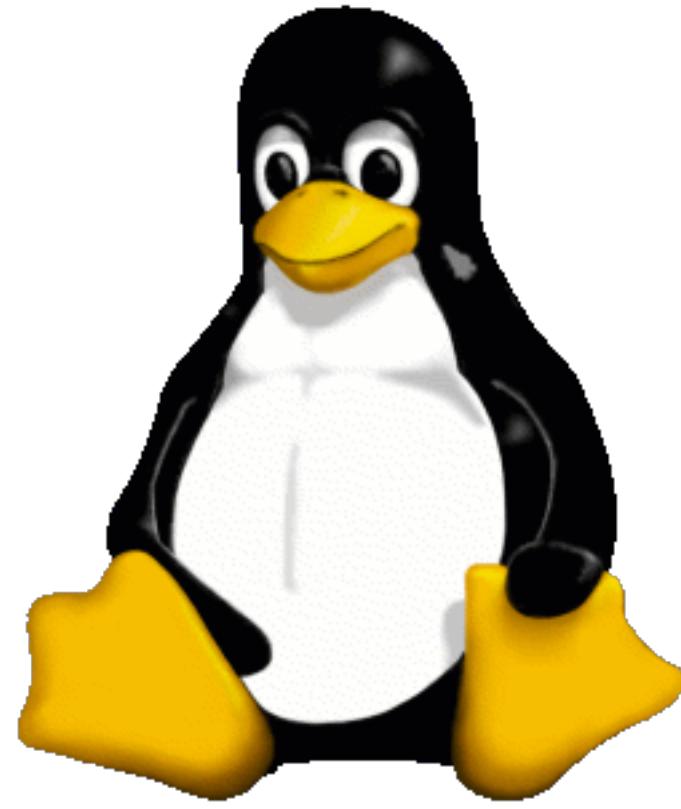


1	2	3	4
5	6	7	8
9	10	11	12

Your application

Operating System

Memory



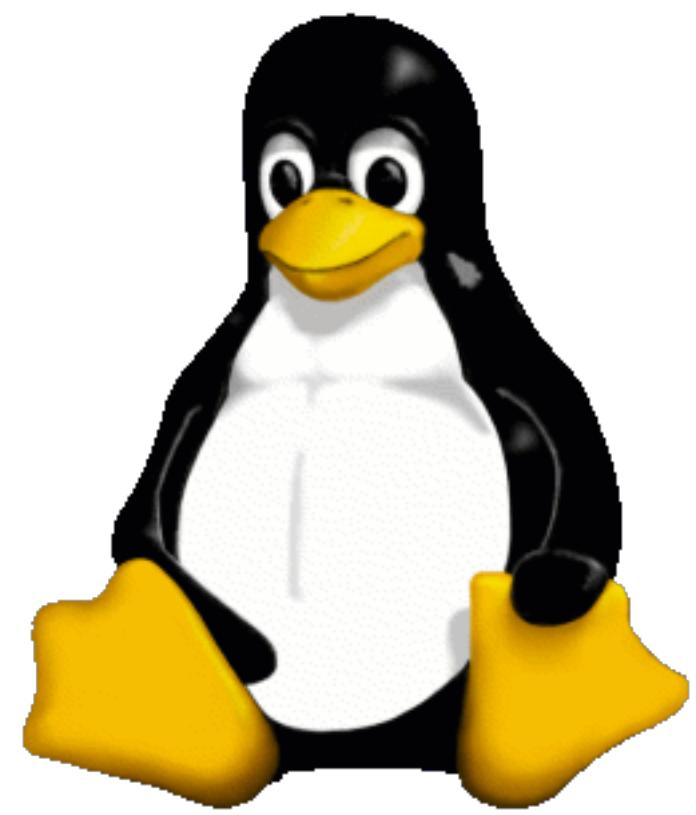
1	2	3	4
5	6	7	8
9	10	11	12

**int[4]; ...I mean,
I'd like to reserve
4 blocks of memory,
please!**

Your application



Operating System



Memory

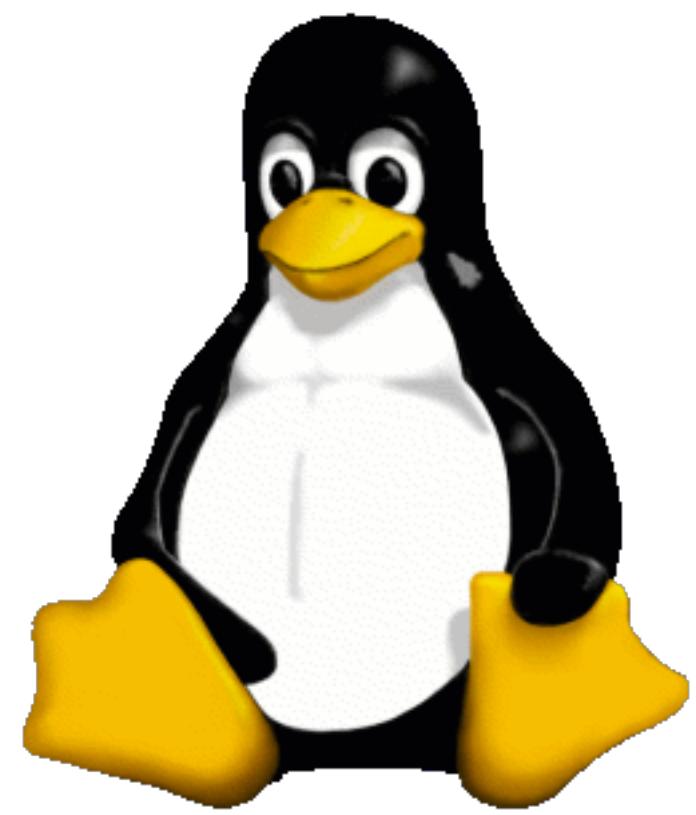


I'll see what we
have available!

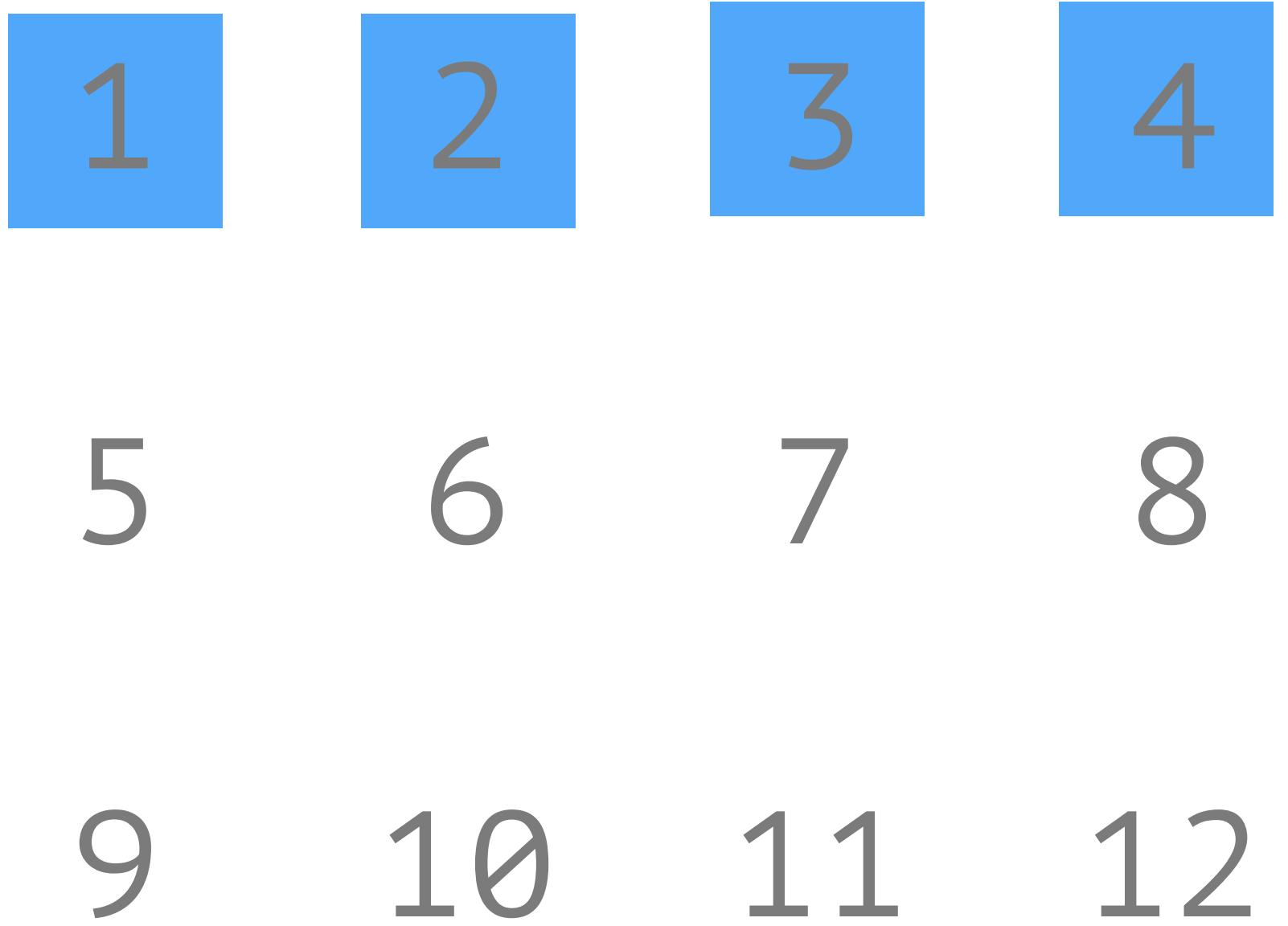
Your application



Operating System



Memory





CONTIGUOUS ARRAY

```
int myArr[4];
```

```
int myArr[4]; //myArr is 4 bytes starting at 0x40
```

0x40

0x41

0x42

0x43

```
int myArr[4]; //myArr is 4 bytes starting at 0x40
```

```
myArr[0] = 9; // put `9` at 0x40 + 0
```

9

0x40

0x41

0x42

0x43

```
int myArr[4]; //myArr is 4 bytes starting at 0x40
```

```
myArr[0] = 9; // put `9` at 0x40 + 0
```

```
myArr[2] = 3; // put `3` at 0x40 + 2 (0x42)
```

9

0x40

0x41

3

0x42

0x43

```
int myArr[4]; //myArr is 4 bytes starting at 0x40
```

```
myArr[0] = 9; // put `9` at 0x40 + 0
```

```
myArr[2] = 3; // put `3` at 0x40 + 2 (0x42)
```

```
myArr[2] // what is at 0x40 + 2 (0x42)?
```

9

0x40

0x41

3

0x42

0x43

STACKS



Stacks

- Ordered Collection of elements
 - **must be LIFO (Last In, First Out)**
- Elements can repeat (not a set)
- Some example operations:
 - Make a new stack
 - Add an element to the stack ("push")
 - Retrieve an element from the list ("pop")
 - Check if stack is empty
 - Look at the top element without removing it ("peek")
 - Clear the stack



0x43

0x42

0x41

0x40

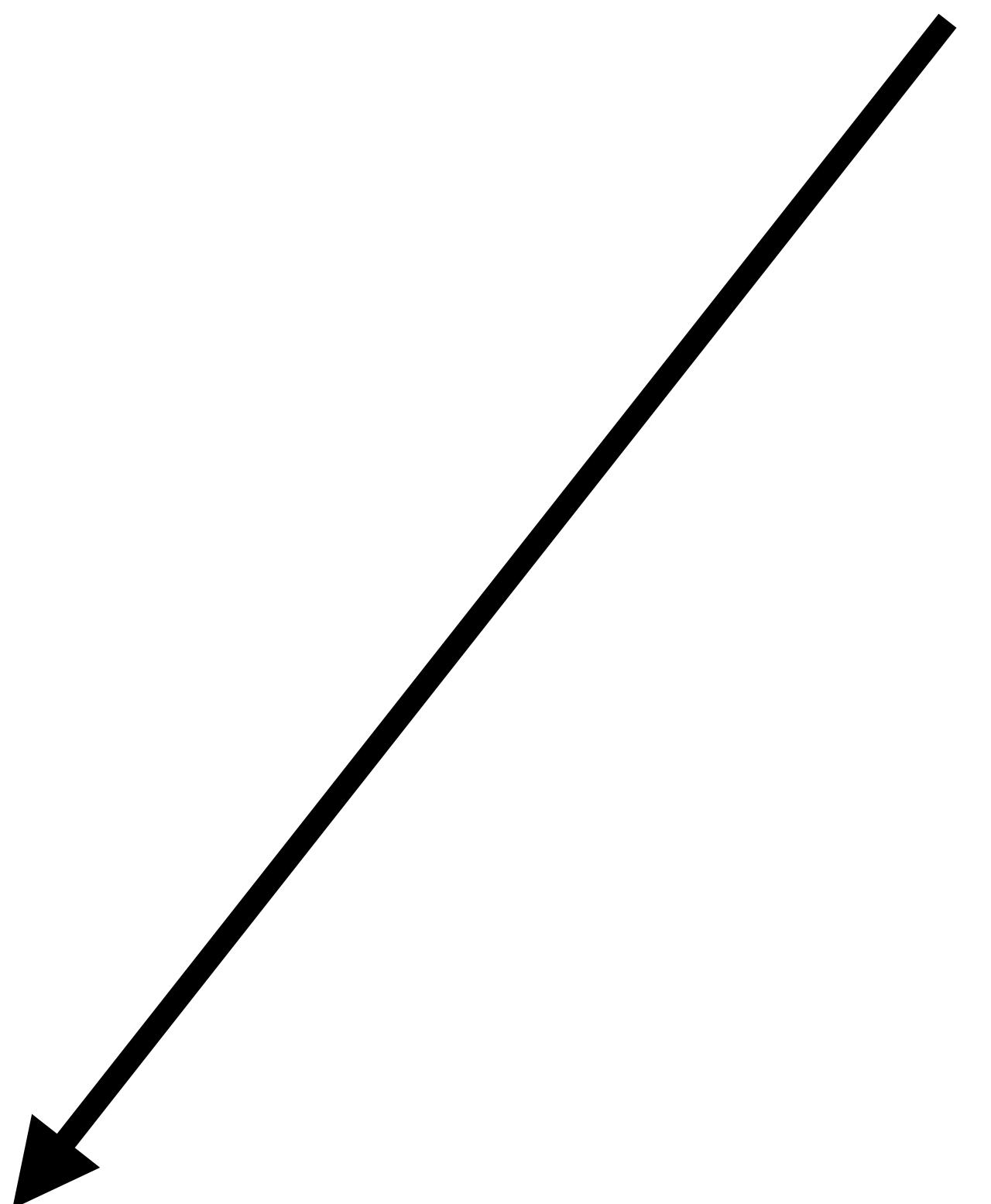
0x43

0x42

0x41

0x40

top



stack = new Stack()

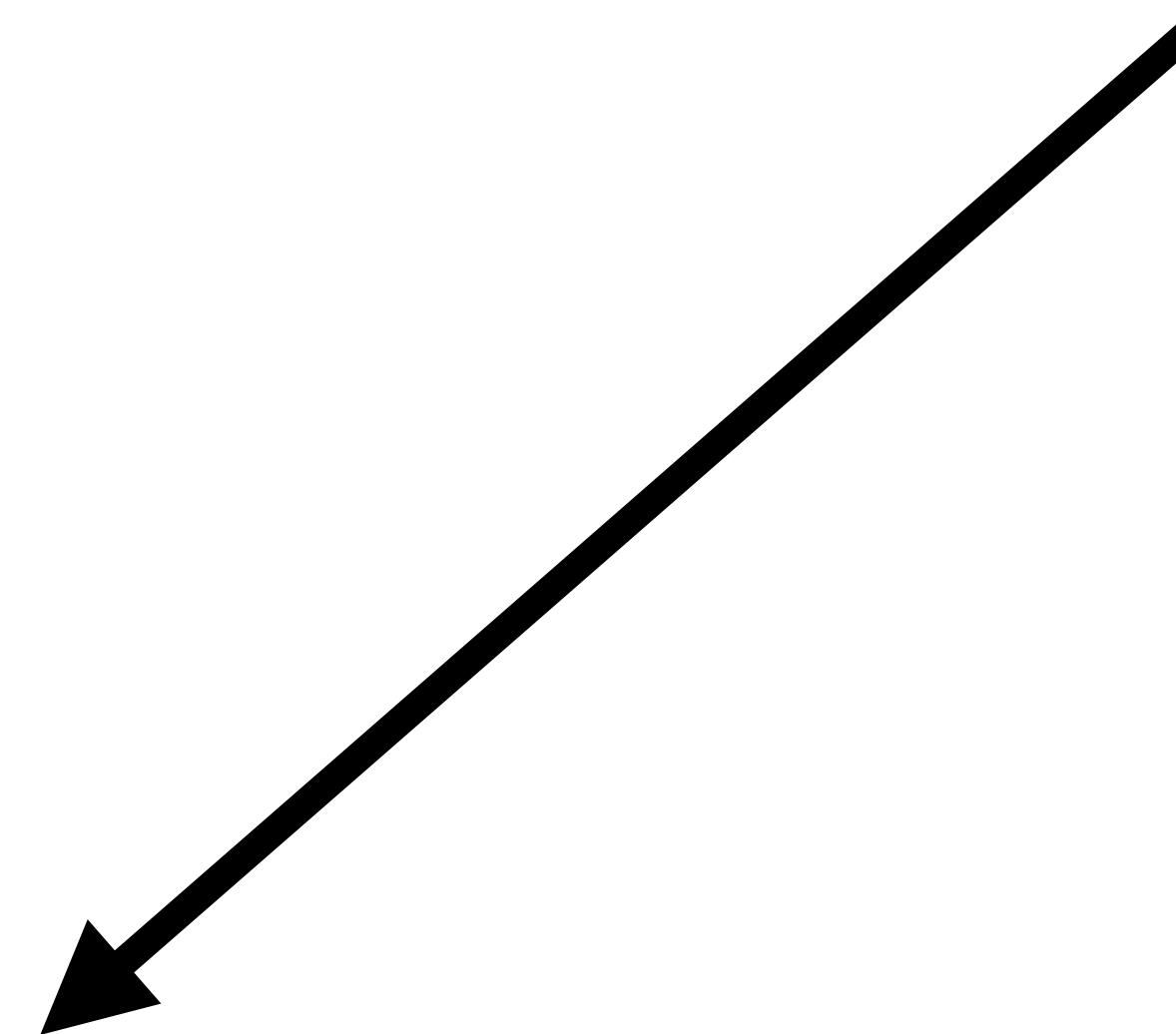
0x43

0x42

0x41

0x40

top



stack = new Stack()

stack.push(3)

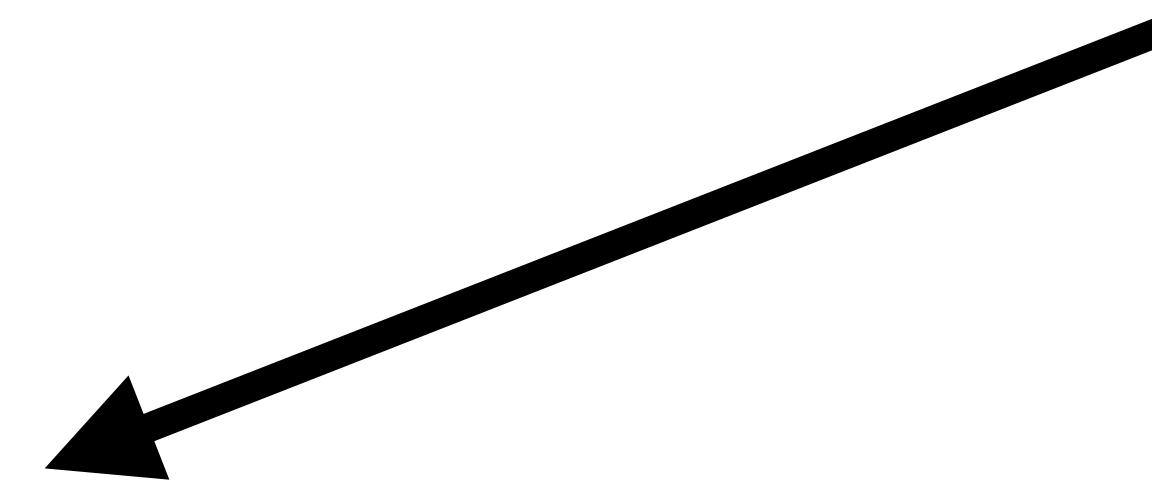
0x43

0x42

0x41

0x40

top



stack = new Stack()

stack.push(3)
stack.push(9)

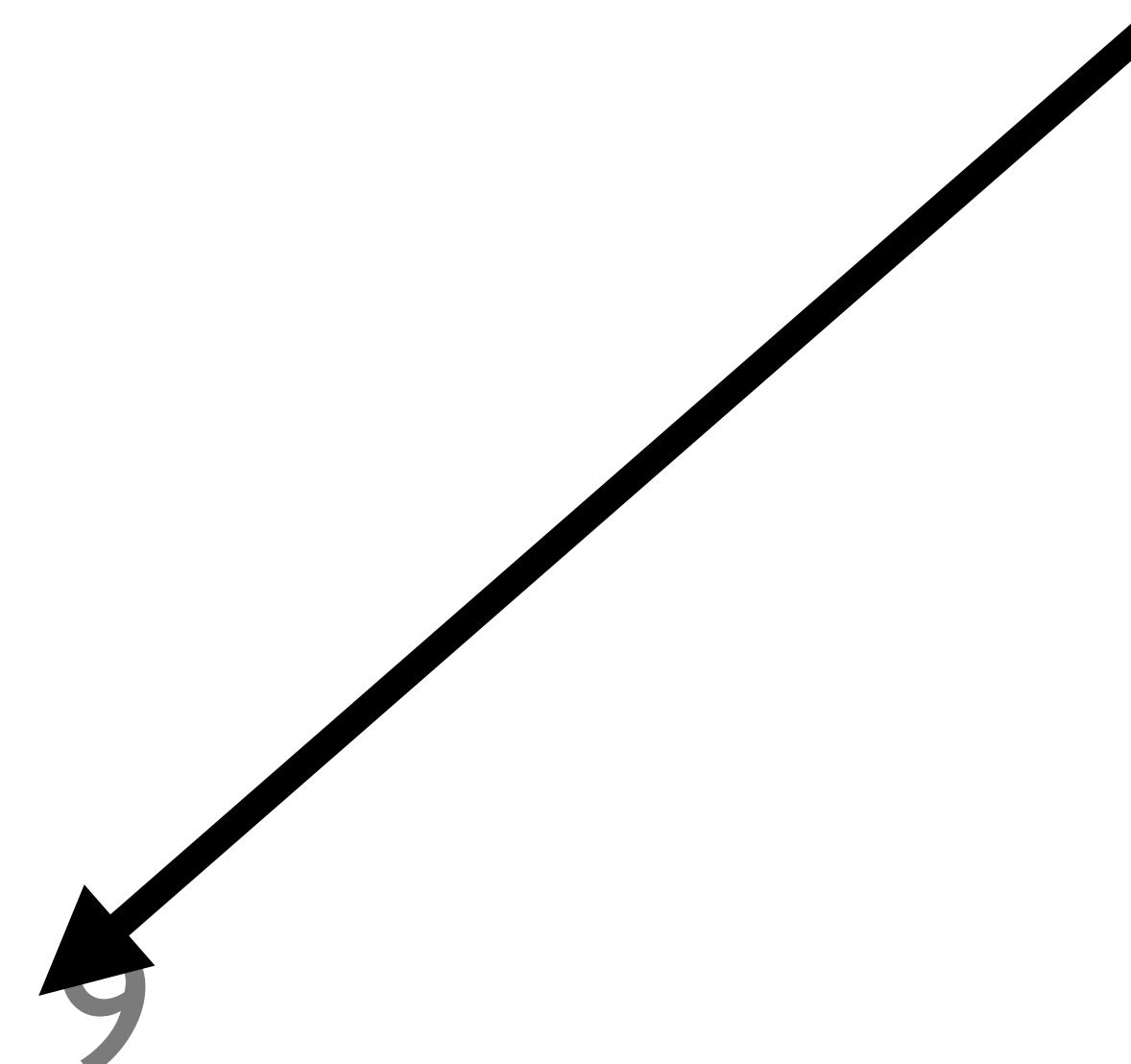
0x43

0x42

0x41

0x40

top



```
stack = new Stack()  
  
stack.push(3)  
stack.push(9)  
stack.pop() // 9
```

Data Structures vs Abstract Data Types

Abstract Data Types

- Describes *how information is related*
 - Ex. is the information ordered in some way? Are elements connected together?
- Describes operations we can perform on that information
 - Ex. add, remove, find

Data Structures

- Concrete, programmatic *implementations* of an ADT
- Describe how information is actually stored in memory
- Determines how performant operations are

The Stack ADT & Array DS

Stack Feature / Operation	Array Implementation with `top`
Collection of elements	Store values at memory addresses
Ordered	Sequential addresses maintain ordering
Create new stack	Initialize a new array
Push onto stack	Insert value at `top` var (next index to use)
Pop off of stack (LIFO)	Use `top` index to return latest value
Check if stack is empty	Return whether `top` variable is 0

A wide-angle photograph of a large crowd of people waiting in a queue at what appears to be a service counter or ticketing area. The queue is long and extends from the foreground down towards the bottom left. The people are dressed in various casual and semi-formal attire, suggesting a public event or a day-to-day service point. The setting is an indoor space with modern architectural elements, including a large glass window on the left and a white tiled wall in the background.

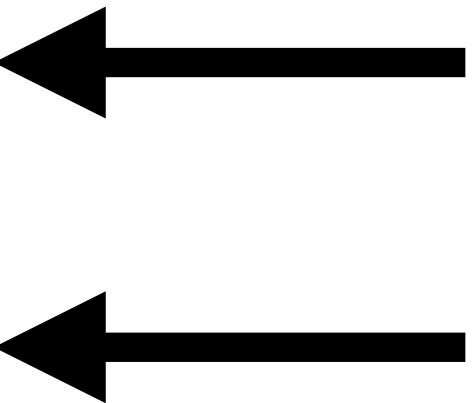
QUEUES

The Queue ADT

- Like Stack, except **FIFO (First In, First Out)**
- Collection of elements
- Ordered / sequential
- Operations:
 - Enqueue (add)
 - Dequeue (remove)
 - Peek
 - Clear
 - IsEmpty... etc.



0x40



head
tail

0x41

0x42

0x43

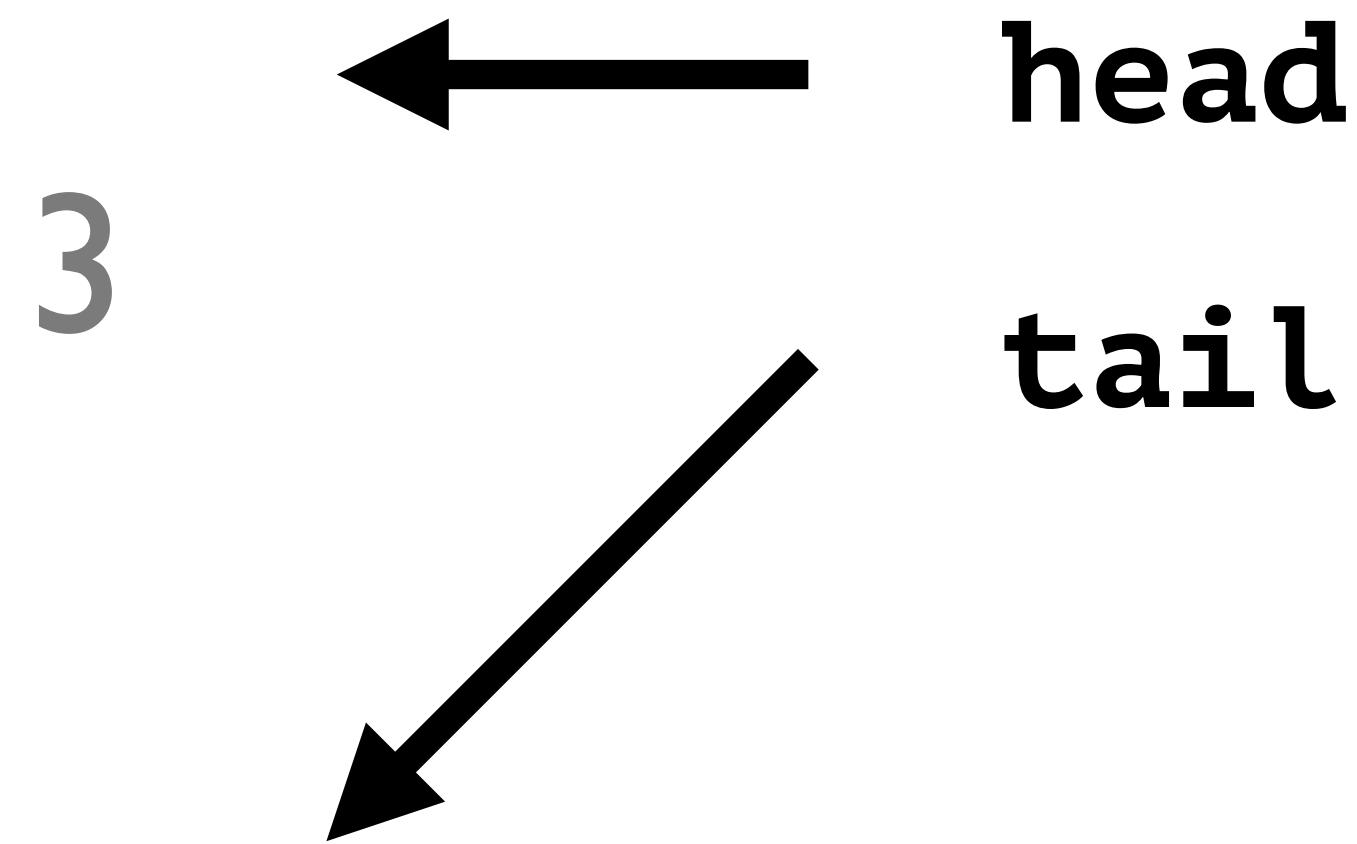
```
queue = new Queue()
```

0x40

0x41

0x42

0x43



```
queue = new Queue()
```

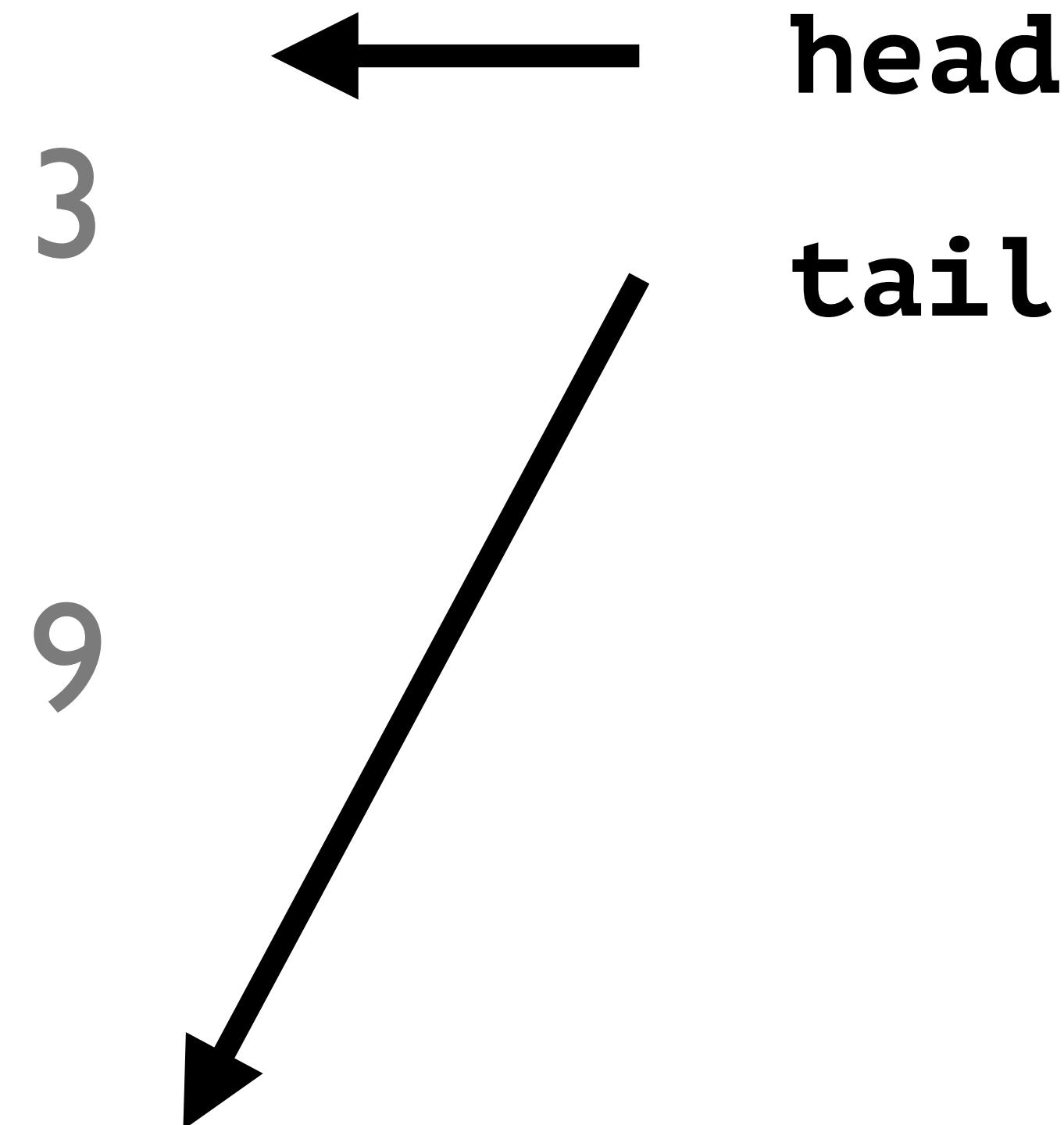
```
queue.enqueue(3)
```

0x40

0x41

0x42

0x43



`queue = new Queue()`

`queue.enqueue(3)`

`queue.enqueue(9)`

0x40

0x41

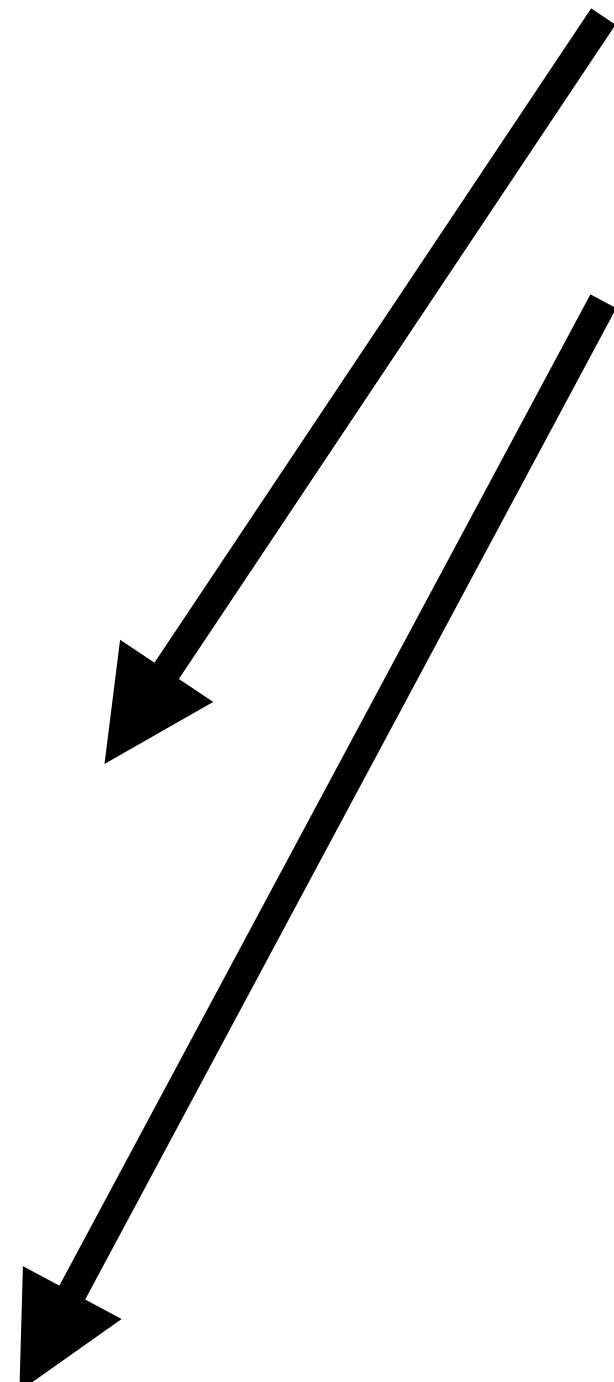
0x42

0x43

3

9

head
tail



```
queue = new Queue()
```

```
queue.enqueue(3)
```

```
queue.enqueue(9)
```

```
queue.dequeue() // 3
```

0x40

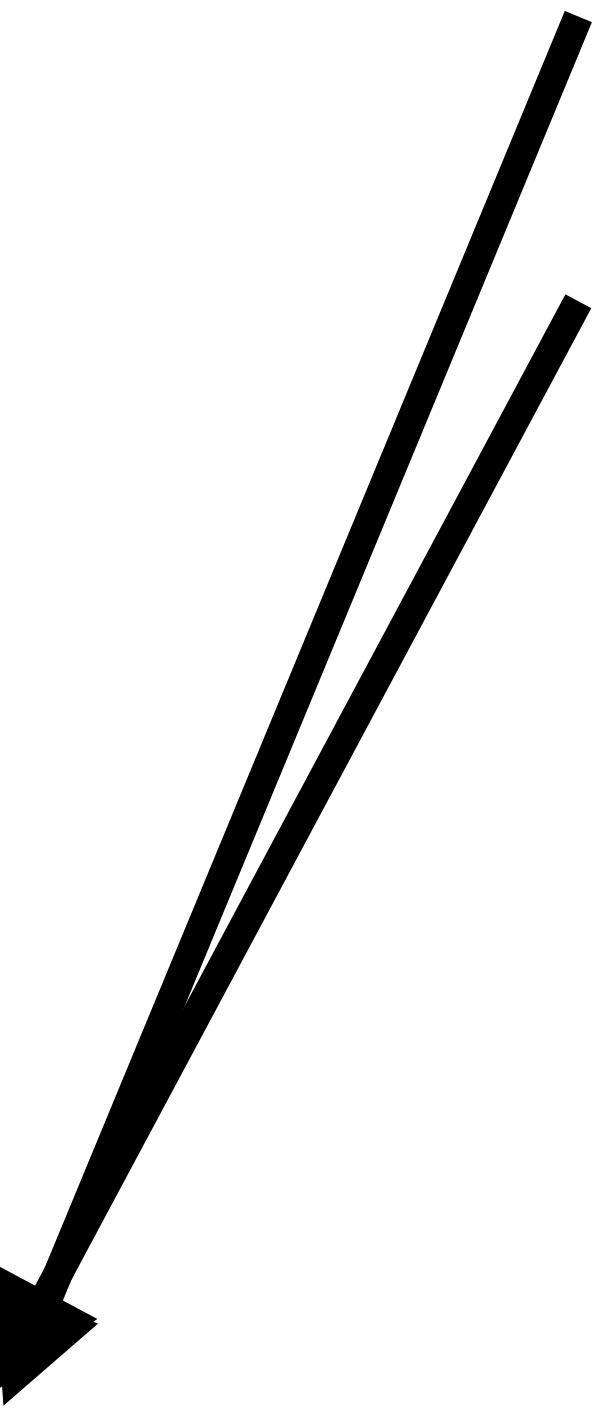
0x41

0x42

0x43

9

head
tail



```
queue = new Queue()
```

```
queue.enqueue(3)
```

```
queue.enqueue(9)
```

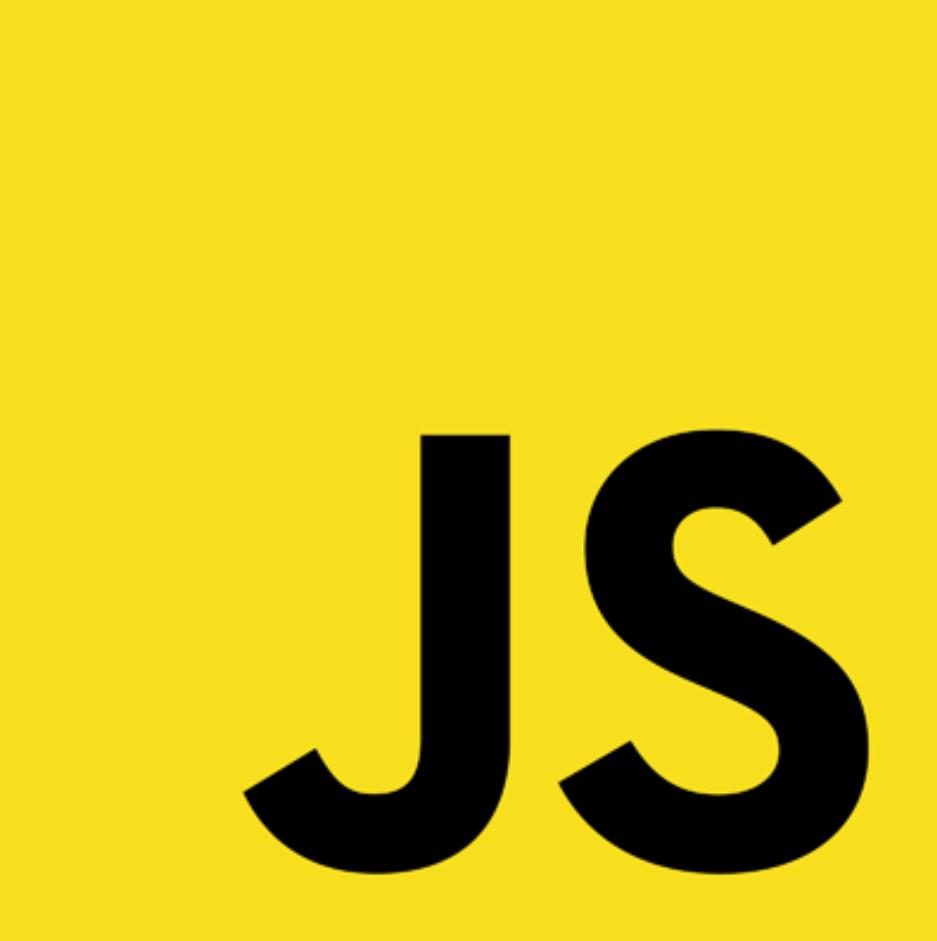
```
queue.dequeue() // 3
```

```
queue.dequeue() // 9
```

WHAT ABOUT JS ?

In JavaScript

- **Arrays are NOT contiguous arrays**
 - Their implementation is left up to the JavaScript engine
- **Simulate a stack**
 - push
 - pop
- **Simulate a queue**
 - `push` to enqueue
 - `shift` to dequeue
- **Data Structures take-home: implement a stack/queue *without* using these methods!**



JS

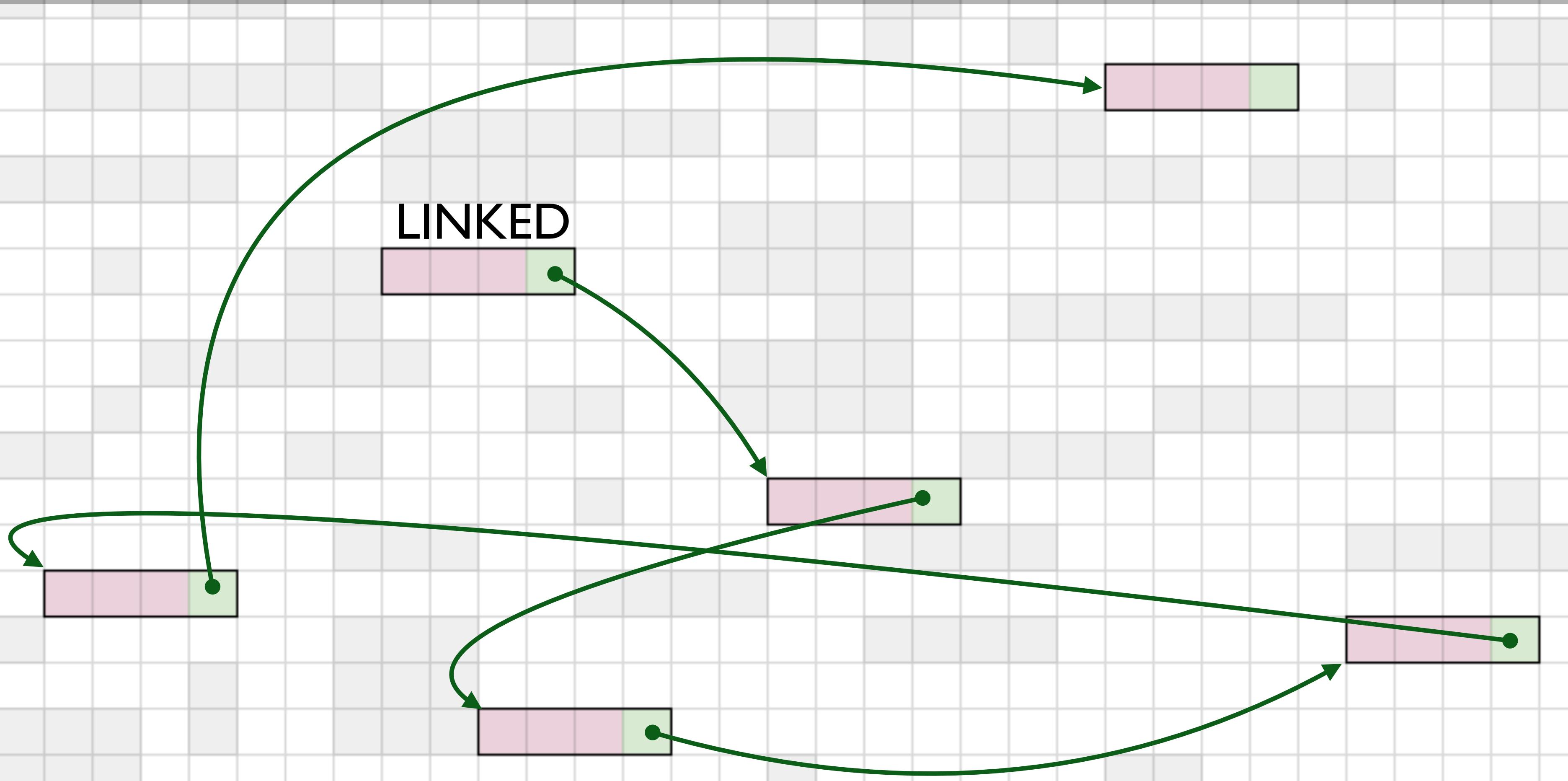
BREAK

Linked Lists

CONTIGUOUS



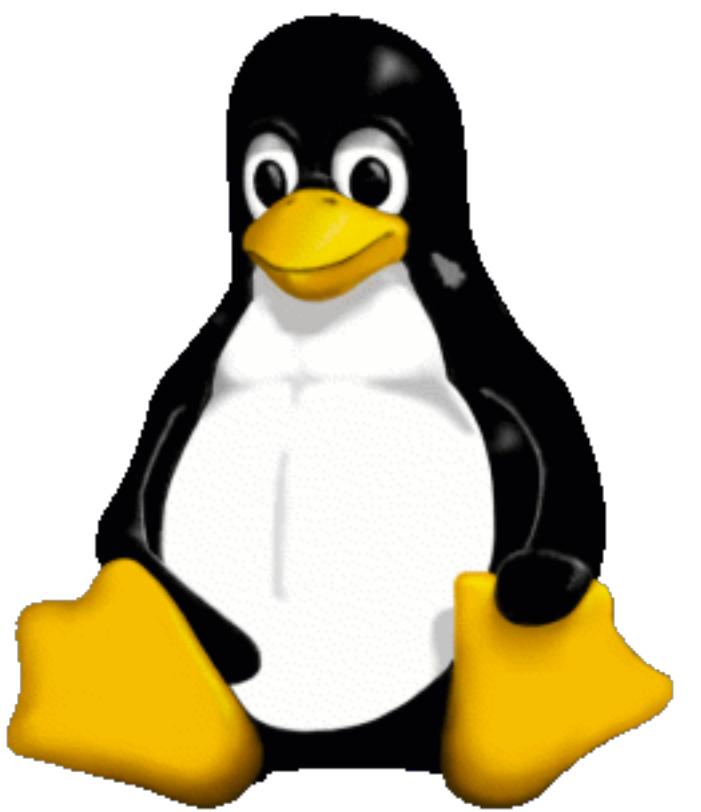
MEMORY



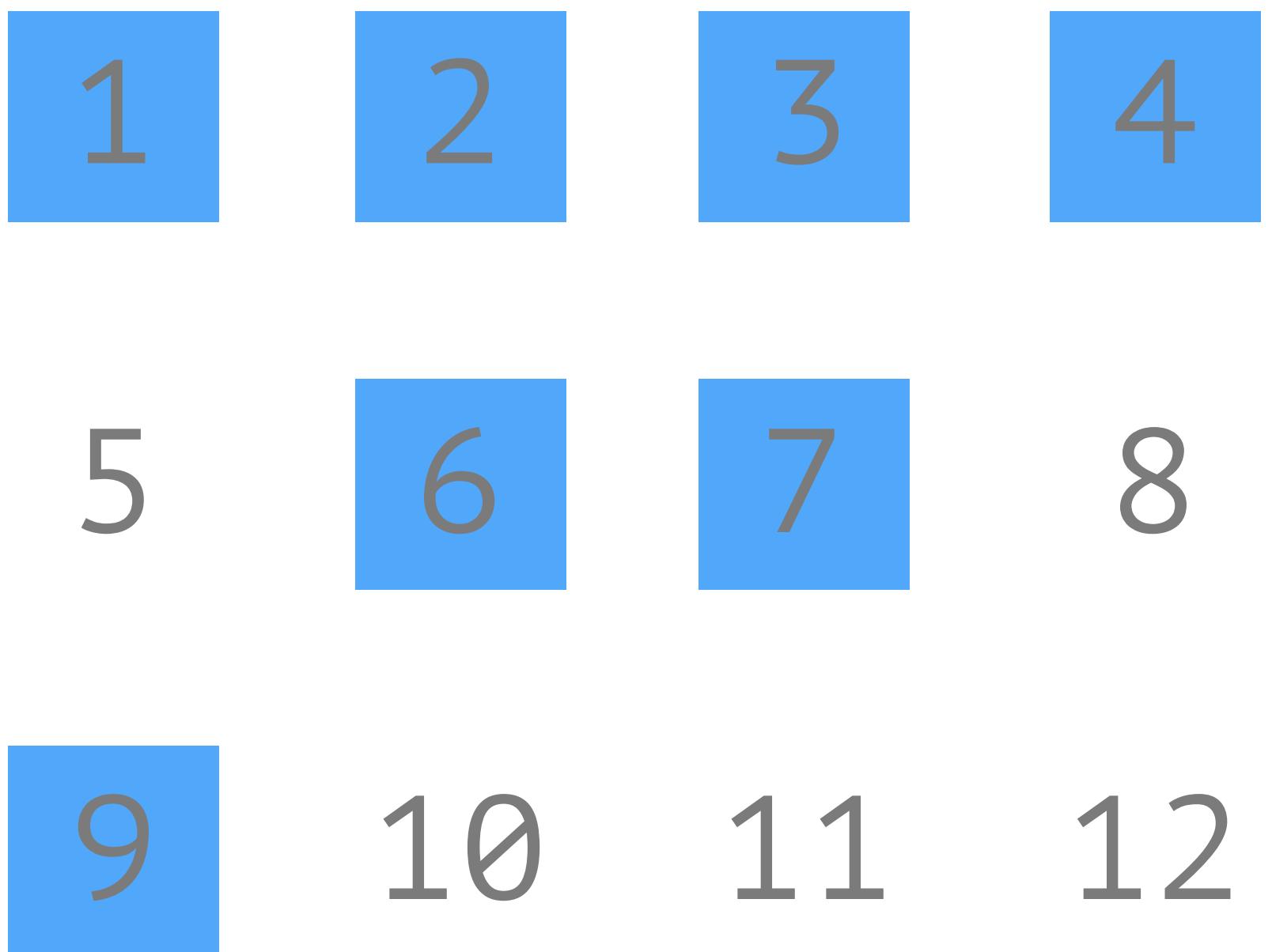
Your application



Operating System



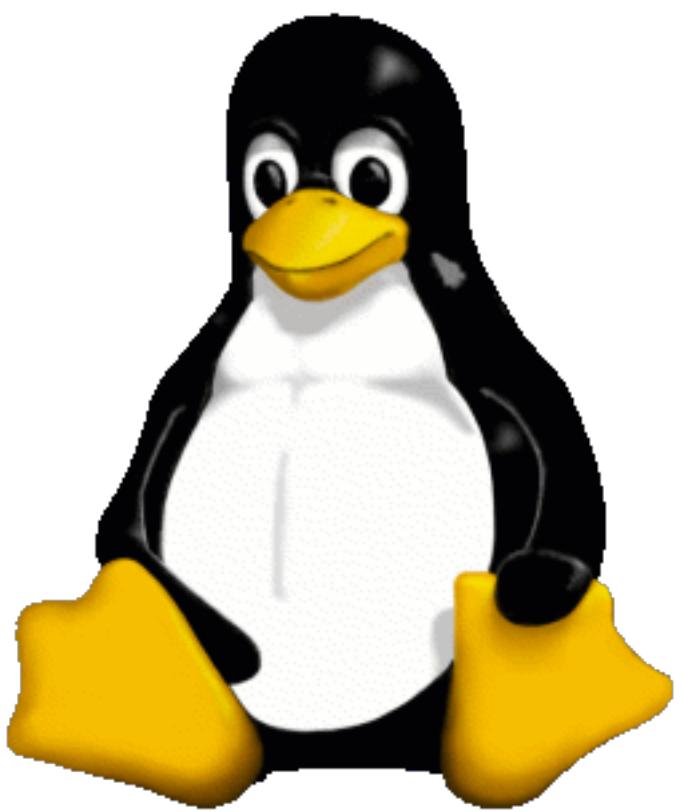
Memory



Your application

Operating System

Memory



1 2 3 4

5 6 7 8

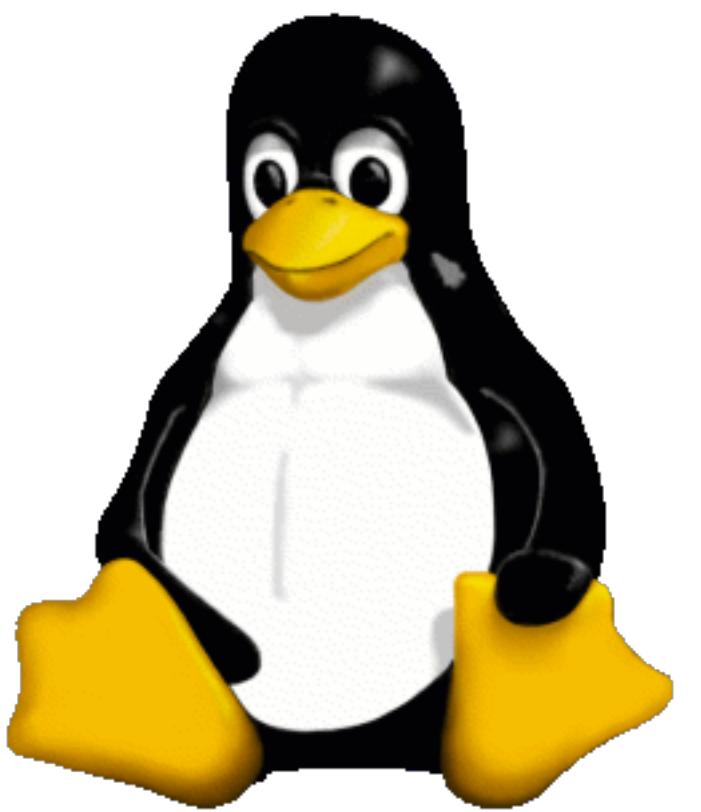
9 10 11 12

I just need one
block of memory for
now...doesn't matter
where!

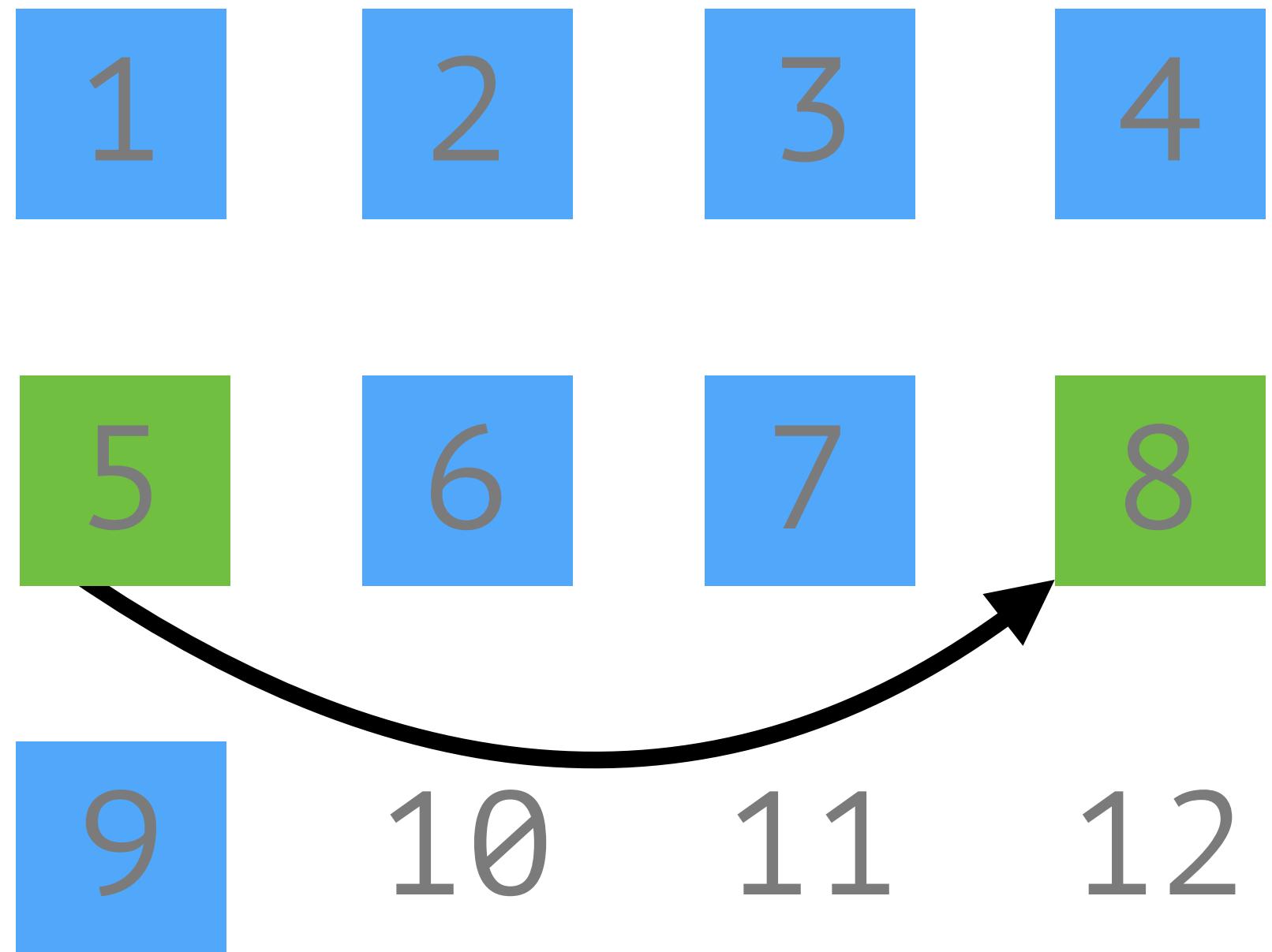
Your application



Operating System



Memory



Okay, I'm adding a new item. I'll use the previous cell you gave me to point to the next

LINKED LISTS



The Linked List DS

- Data structure used for *list, stack, queue, deque* ADTs etc.
- Uses **nodes** which encapsulate a **value** and pointer(s)
- Main entity holds reference(s) to just a head and/or tail node
 - the "**handle(s)**"
- Each node then *points to the next and/or previous node*
 - "**singly-linked**" (unidirectional) vs. "**doubly-linked**" (bidirectional)



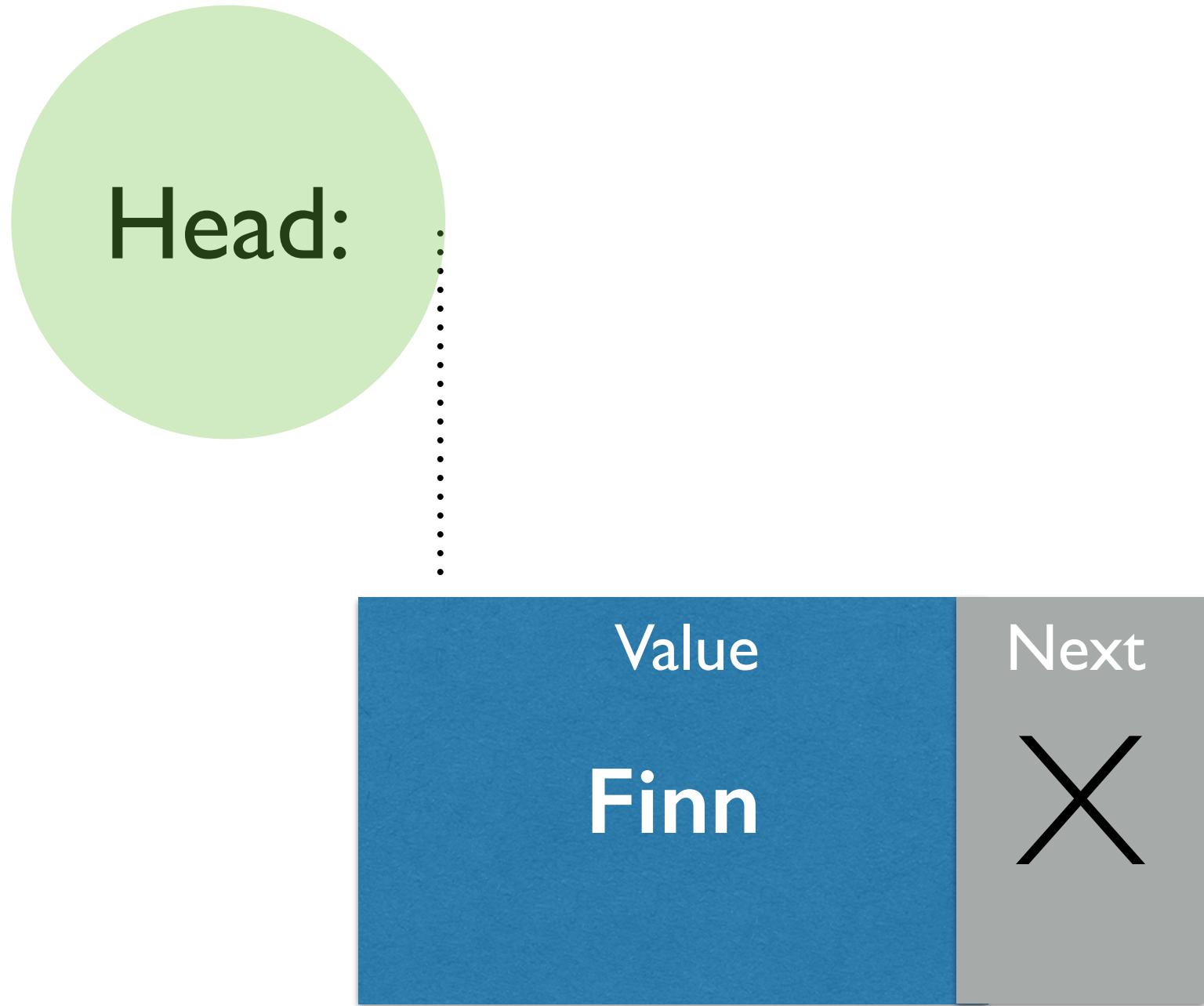


Linked List

Head:



Linked List



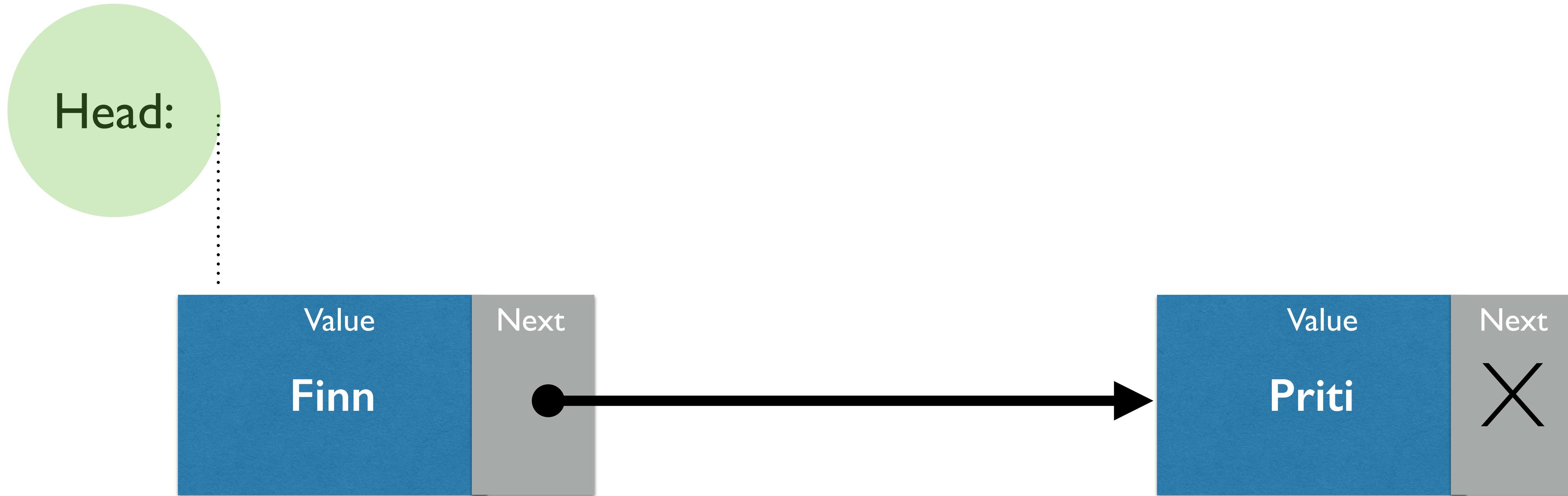


Linked List



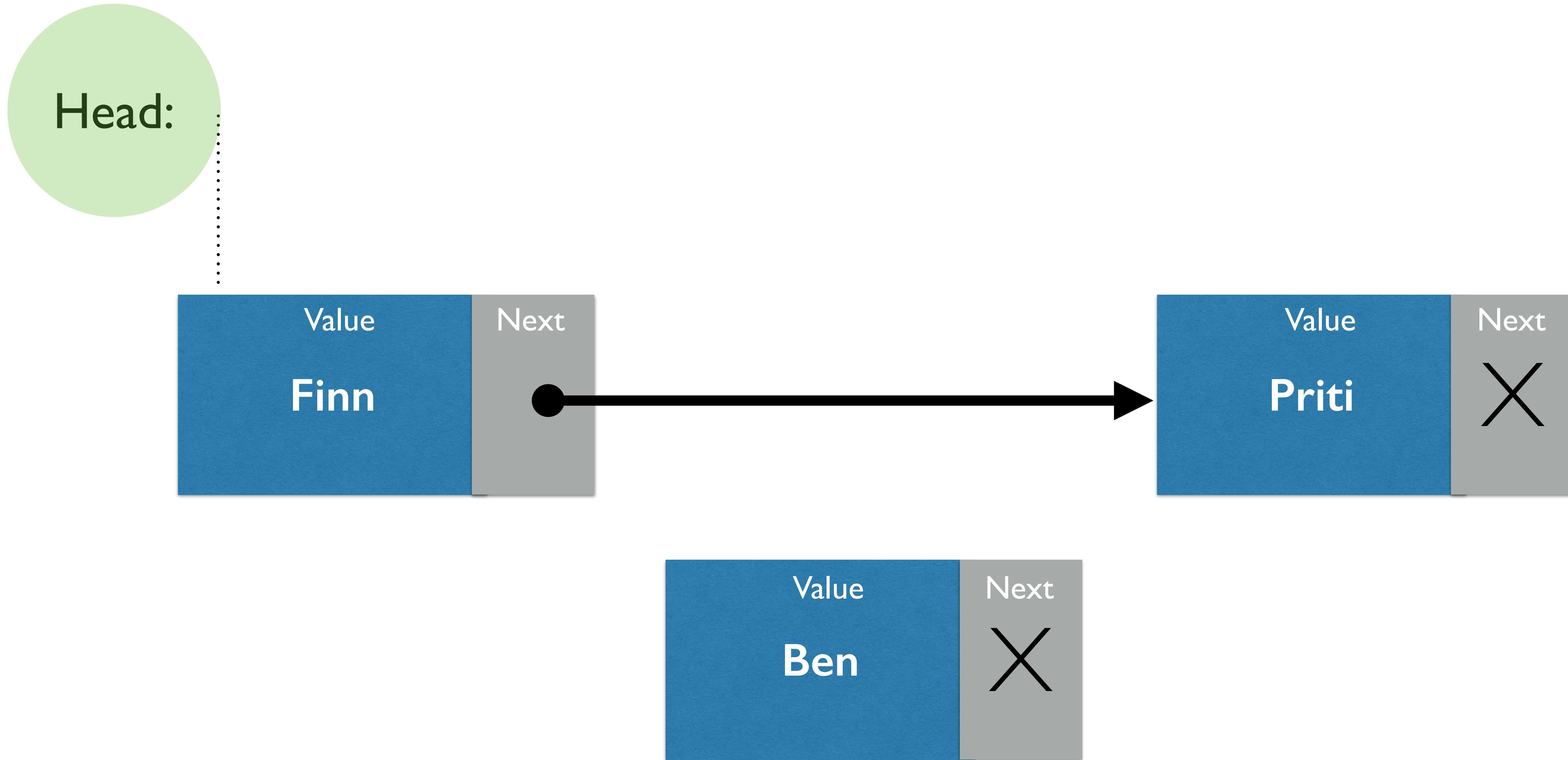


Linked List



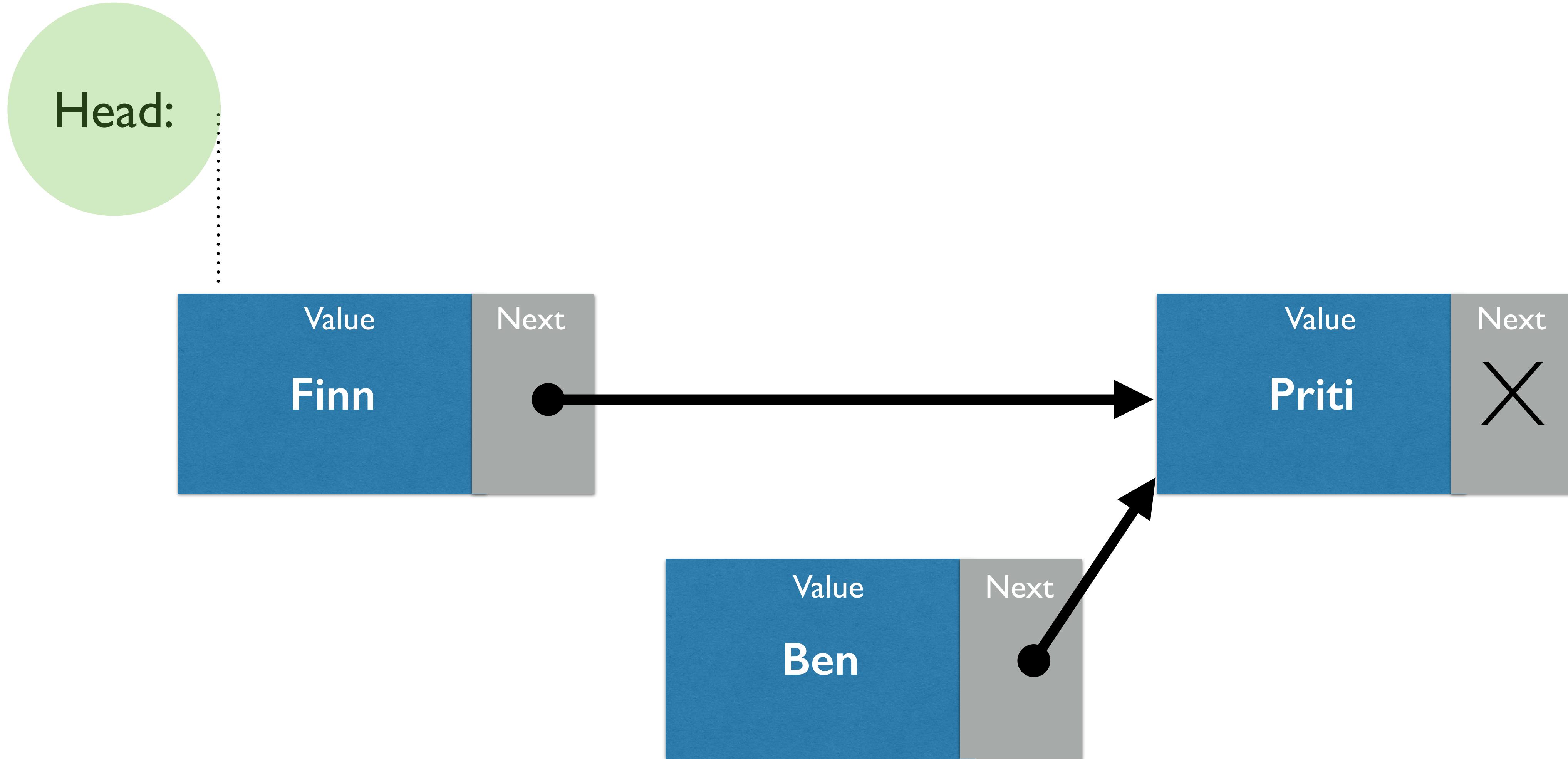


Linked List



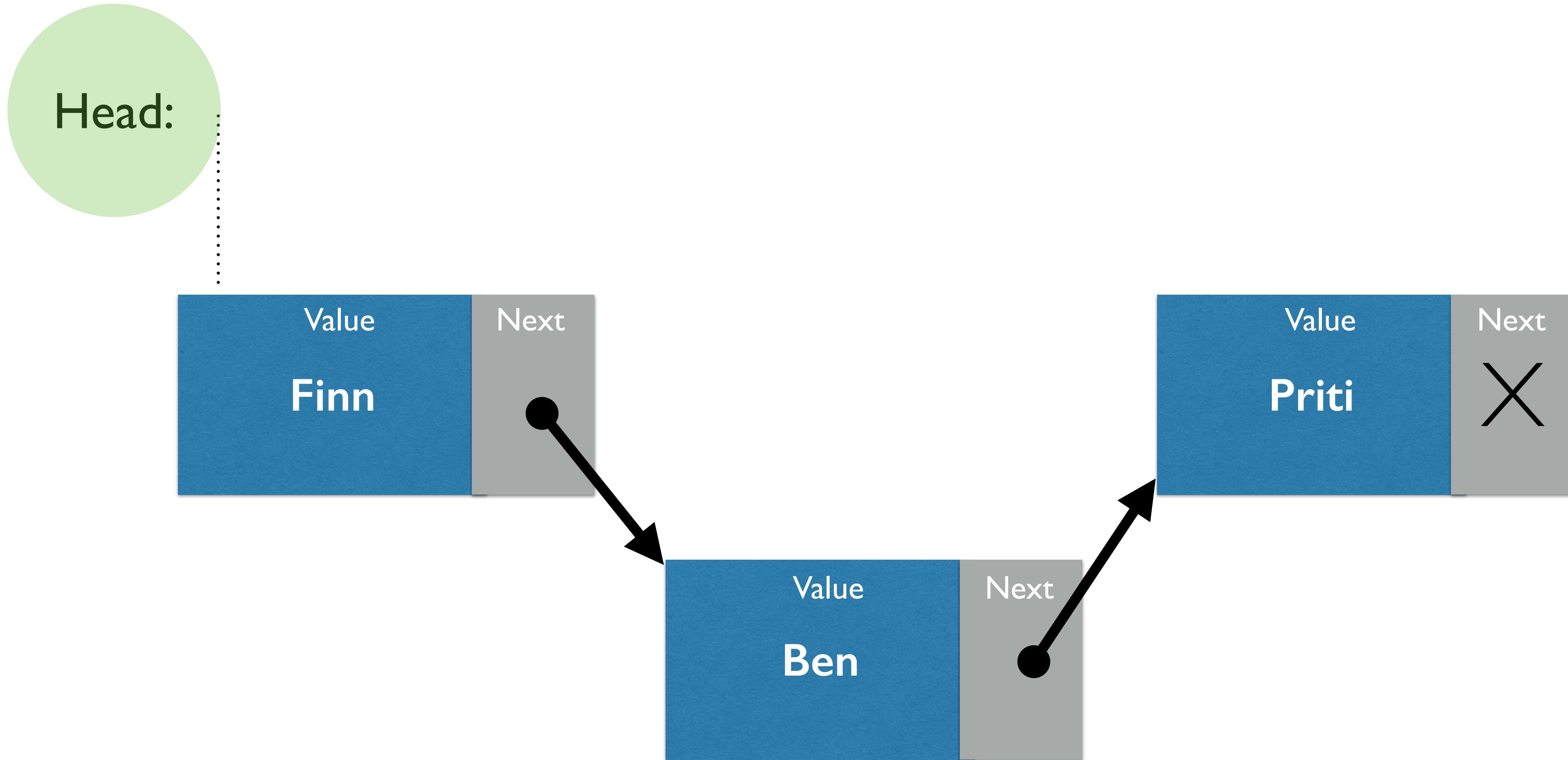


Linked List



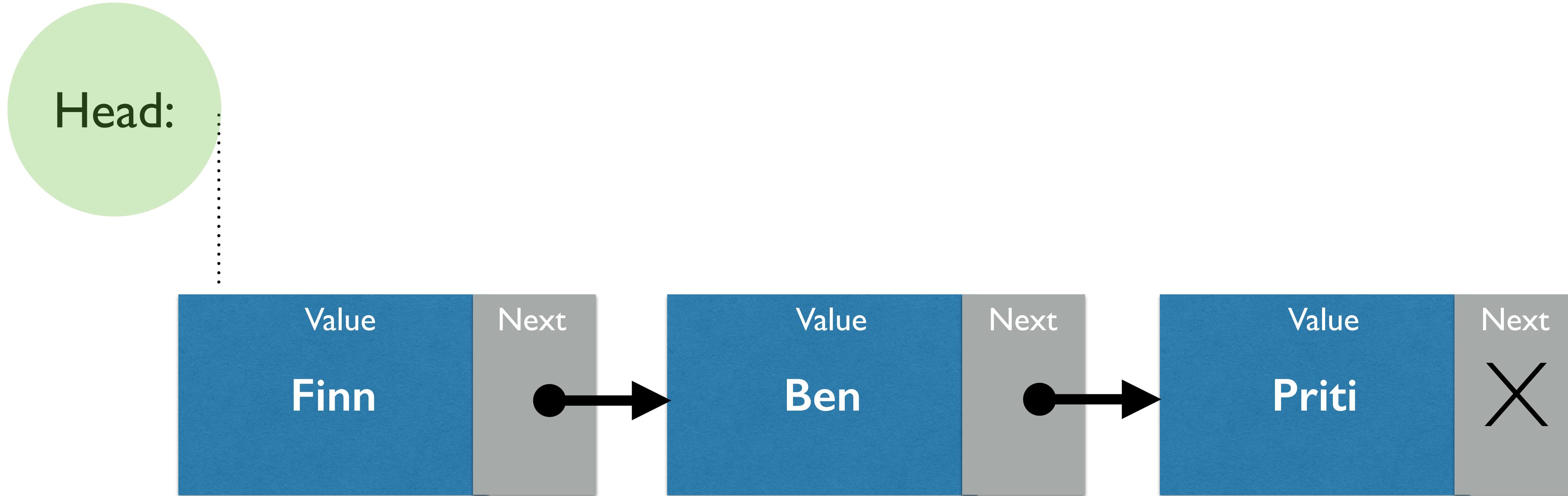


Linked List



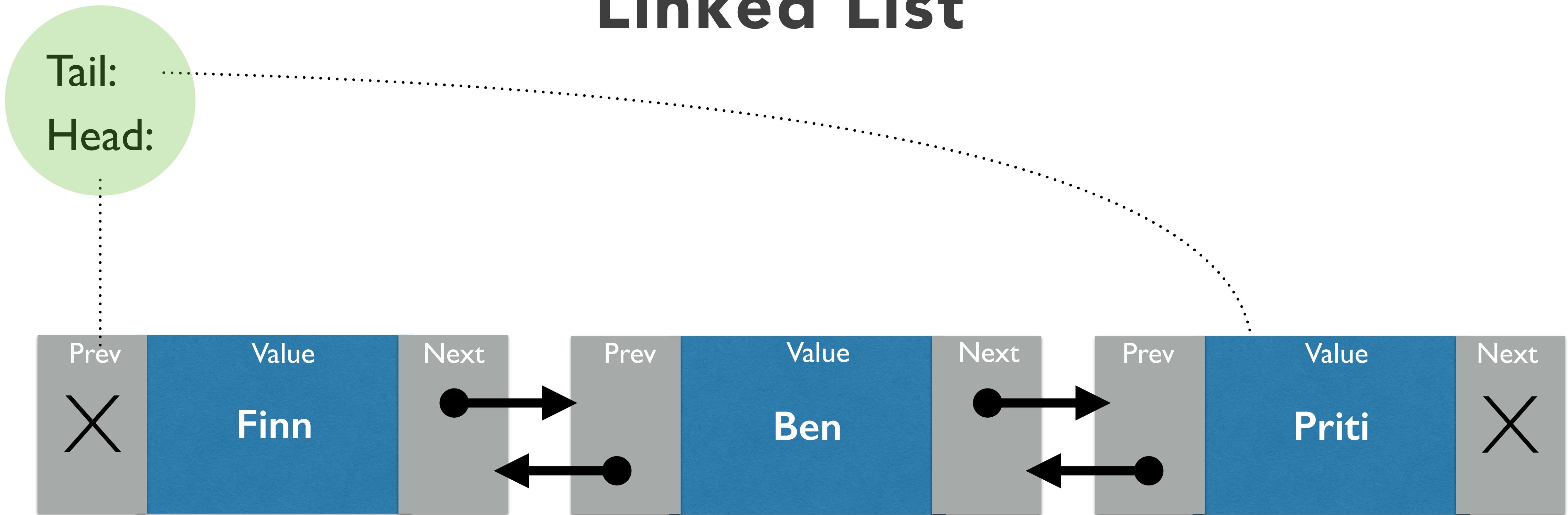


Linked List





Linked List





(some) pros/cons: Linked Lists vs Arrays

Operation	Linked List	Typed Array
Reach element in middle	Must crawl though nodes	Constant time
Insert in middle or start	Constant time (if we have ref). With handle, constant time	Must move all following elements
Add element to end	With handle, constant time	Constant time
Space per element	Container + element + pointer(s)	Just element!
Total space	Grows as needed	Pre-reserved & limited*
Physical locality	Not likely	Best possible

ADTs vs DSs

Common Abstract Data Types	(Some) Data Structures
Set	List, Linked Tree, Trie, Hash Table, etc.
List	Linked List, Array
Stack	Array, Linked List
Queue	Linked List, Deque
Map (Associative Array / Dictionary / etc.)	Hash Table, Association List, Red-Black Tree
Graph	Adjacency List, Adjacency Matrix
Tree	Linked Tree, Heap

WORKSHOP

