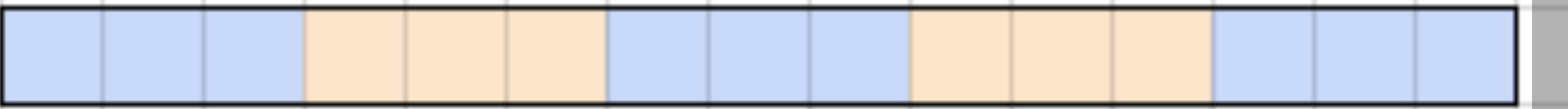
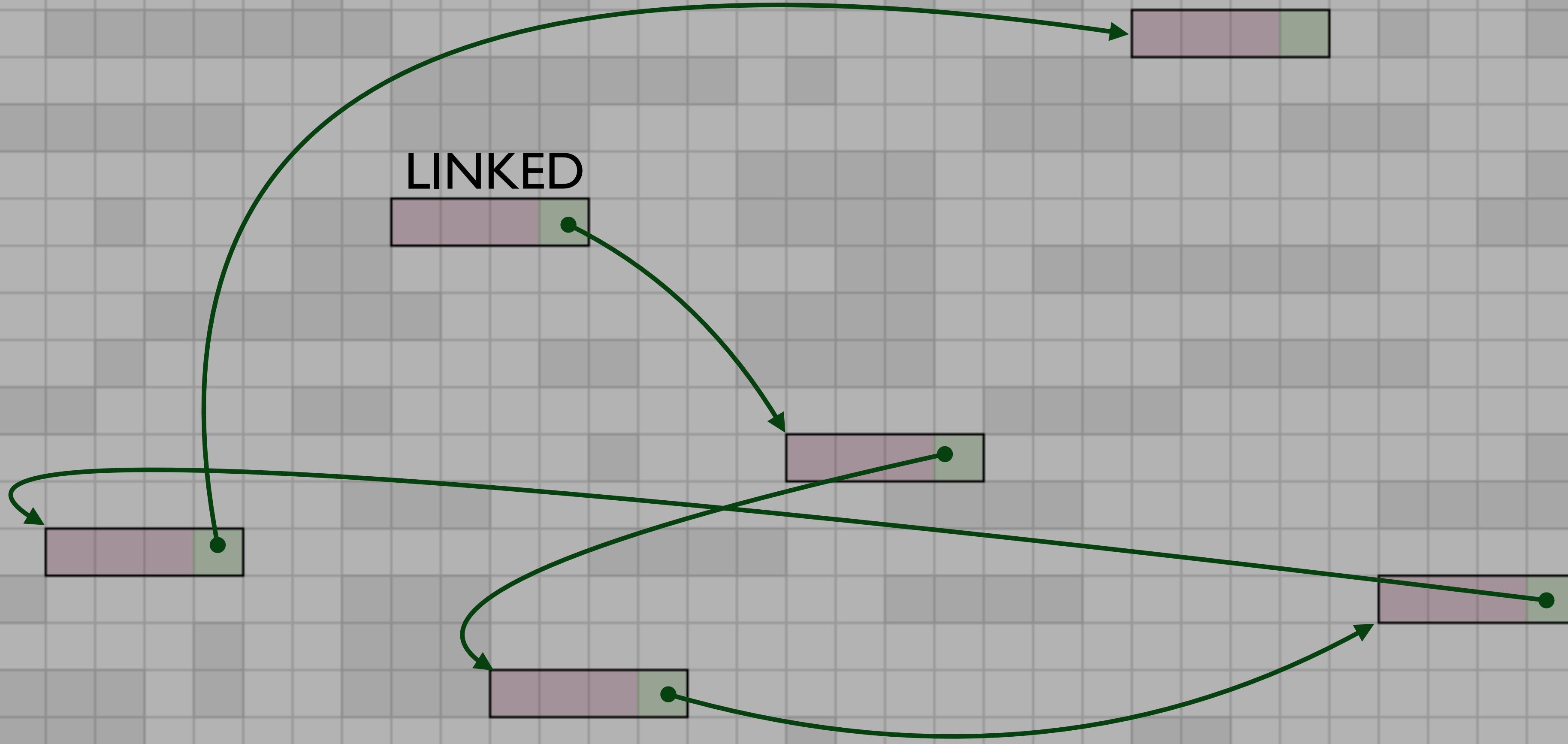


# CONTIGUOUS



# MEMORY

## LINKED



# TRAJECTORY

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

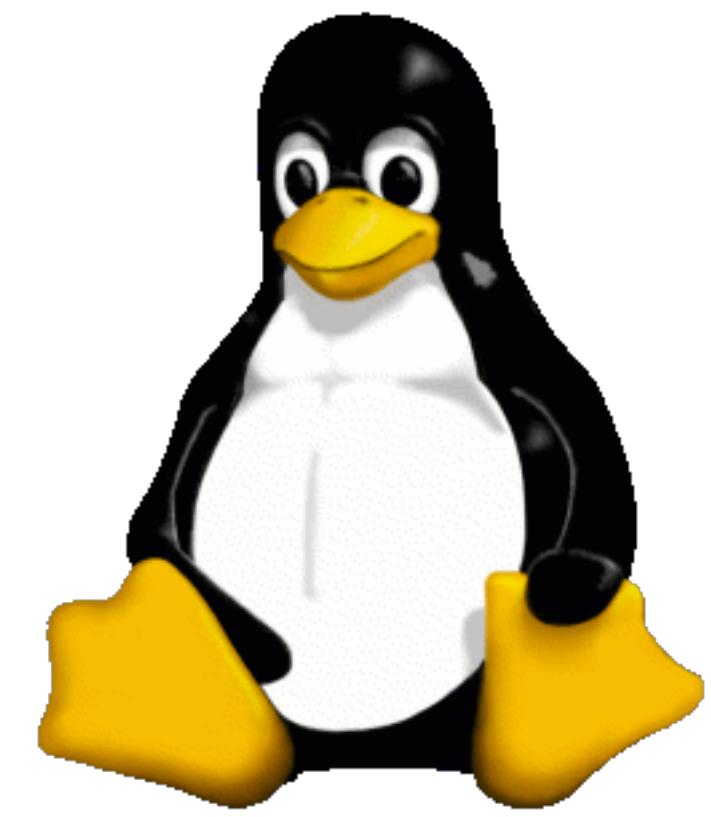
**MEMORY**



Your application



Operating System



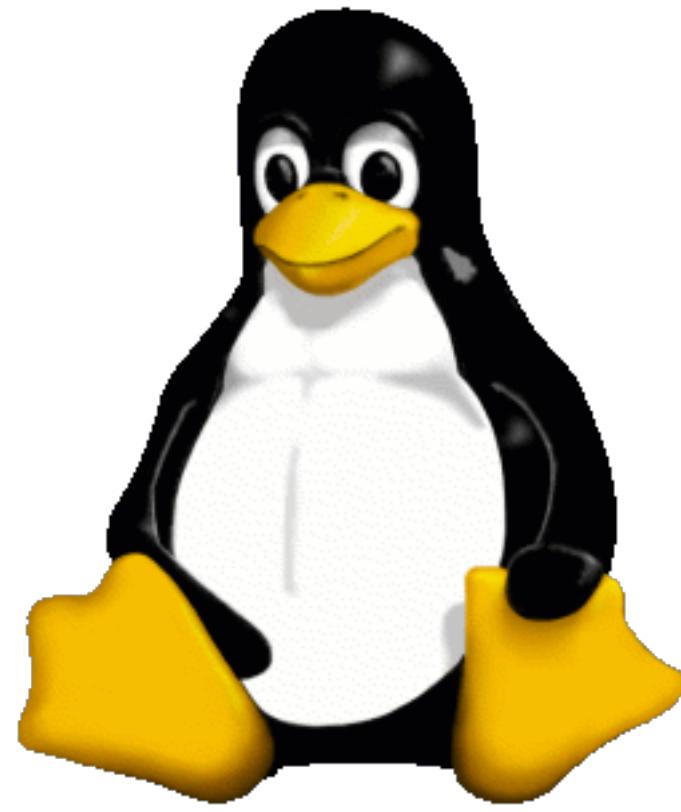
Memory



Your application

Operating System

Memory



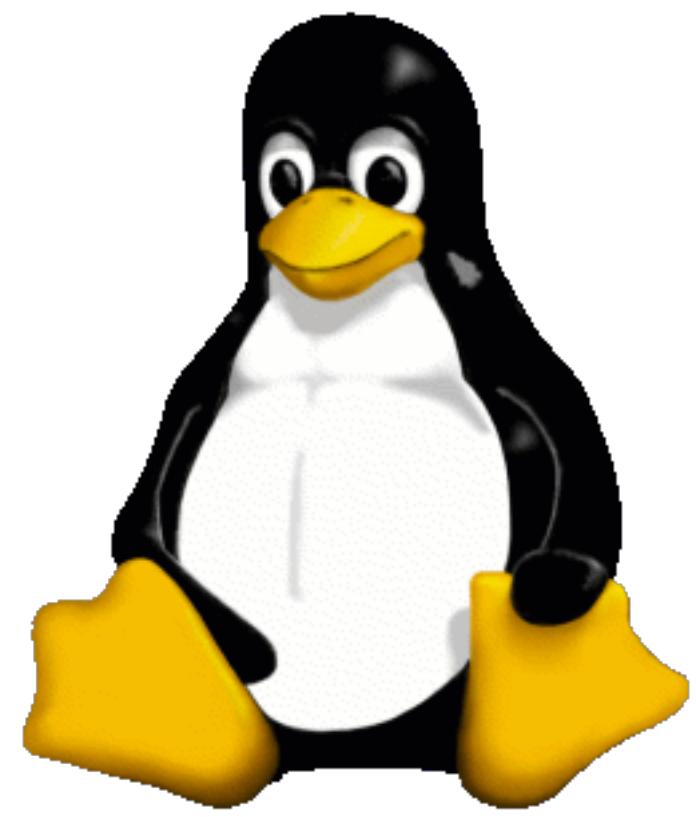
1	2	3	4
5	6	7	8
9	10	11	12

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

Your application



Operating System



Memory

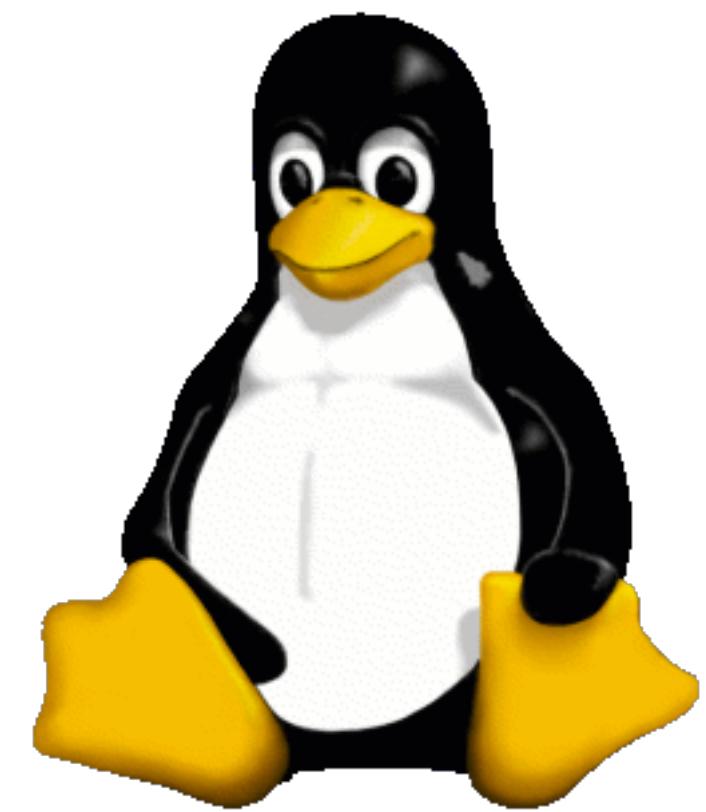


I'll see what we  
have available!

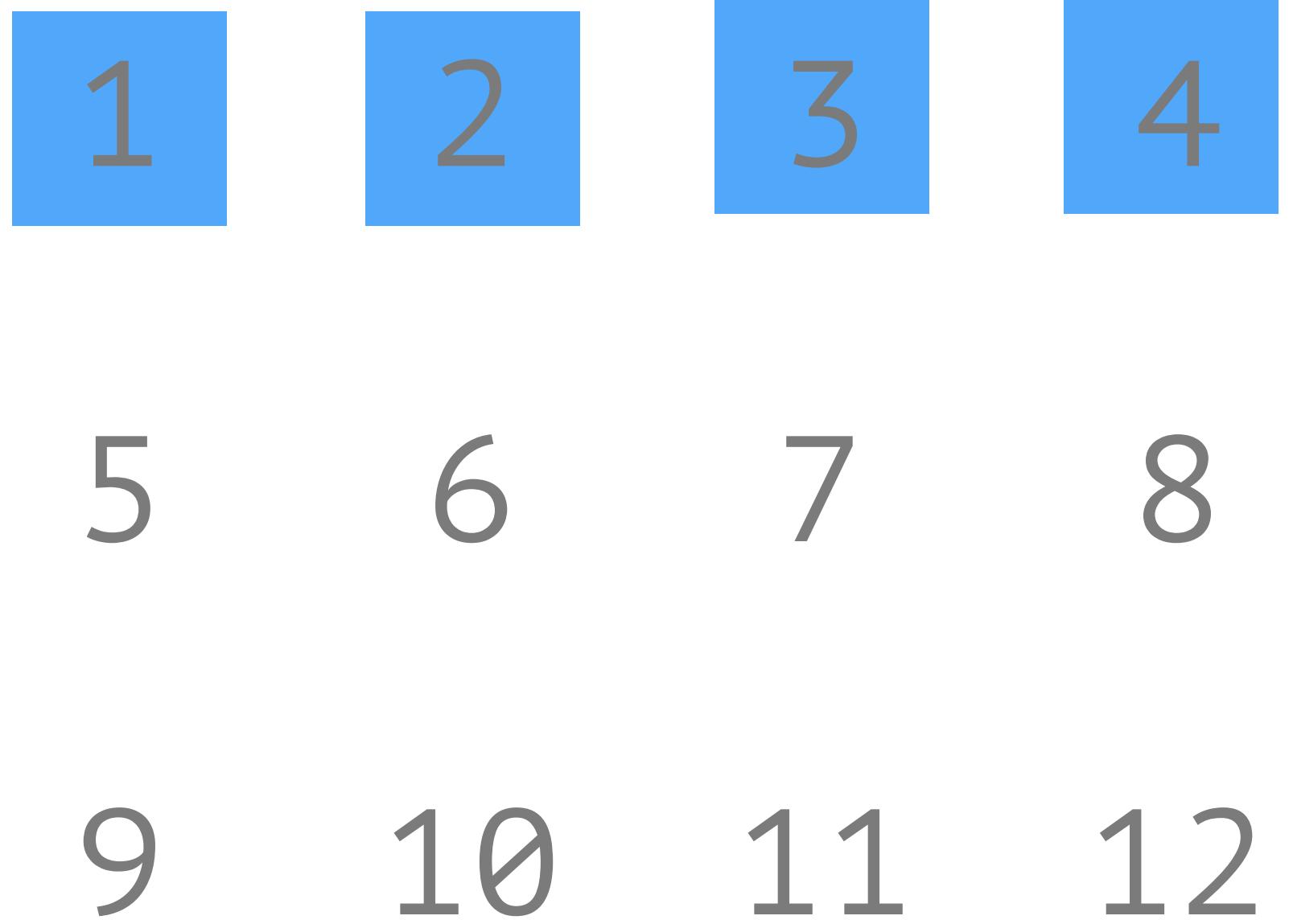
Your application



Operating System



Memory



**CONTIGUOUS ARRAY**

0x40

0x41

0x42

0x43

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

0x40

0x41

0x42

0x43

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

0x40

0x41

0x42

0x43

```
int myArr[4]; //myArr is 4 bytes starting at 0x40  
int myArr[0] = 9; // put `9` at 0x40 + 0
```

9

0x40

0x41

0x42

0x43

```
int myArr[4]; //myArr is 4 bytes starting at 0x40
int myArr[0] = 9; // put `9` at 0x40 + 0
int myArr[2] = 3; // put `3` at 0x40 + 2 (0x42)
```

9

0x40

3

0x42

0x43

```
int myArr[4]; //myArr is 4 bytes starting at 0x40  
int myArr[0] = 9; // put `9` at 0x40 + 0  
int myArr[2] = 3; // put `3` at 0x40 + 2 (0x42)
```

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

9

0x40

0x41

3

0x42

0x43

STACKS



# Stacks

- Collection of elements
- Ordered
- 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") - *must be LIFO (Last In, First Out)*
  - Check if stack is empty
  - Look at the top element without removing it ("peek")
  - Clear the stack



0x43

0x42

0x41

0x40

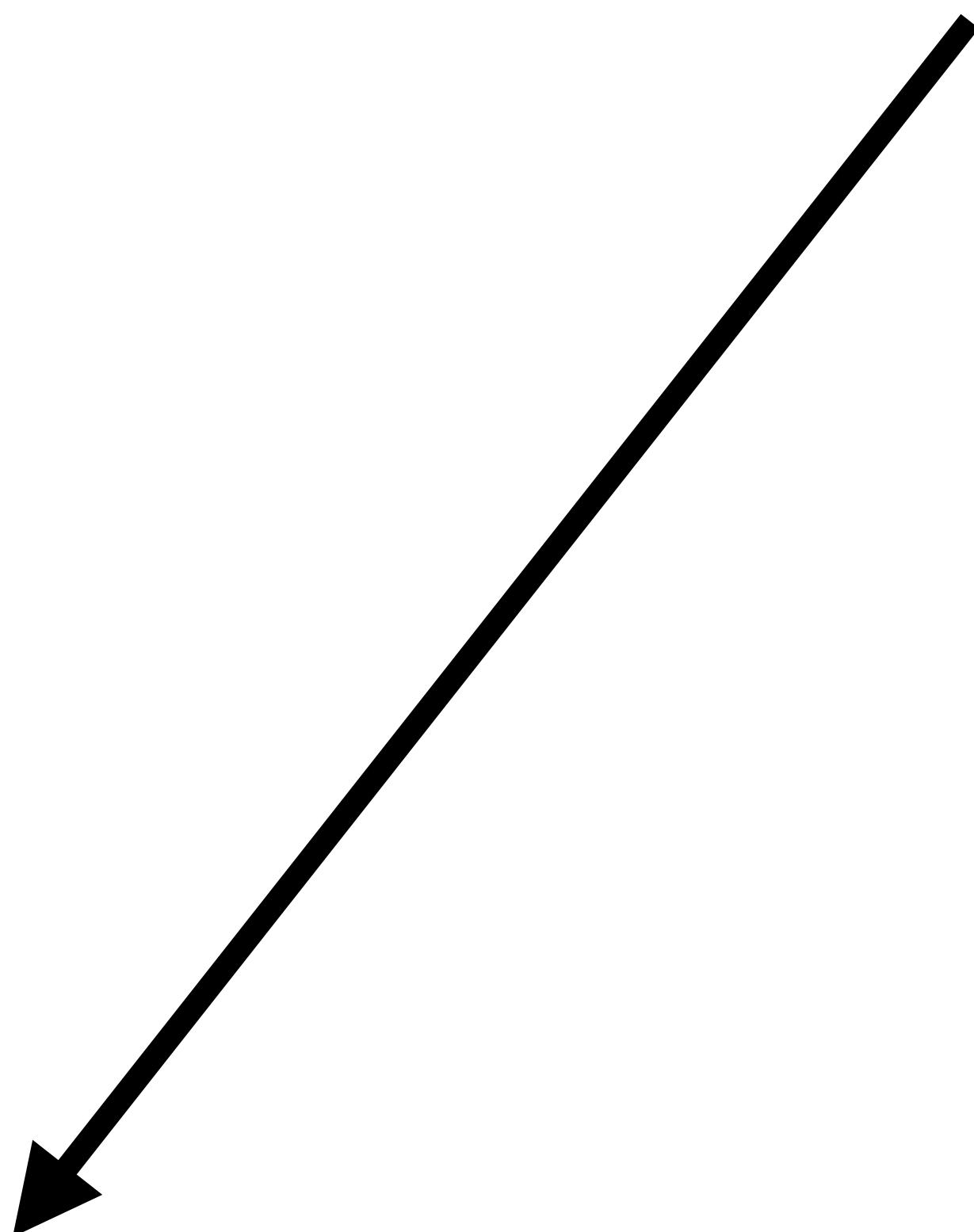
0x43

0x42

0x41

0x40

**head**



**stack = new Stack()**

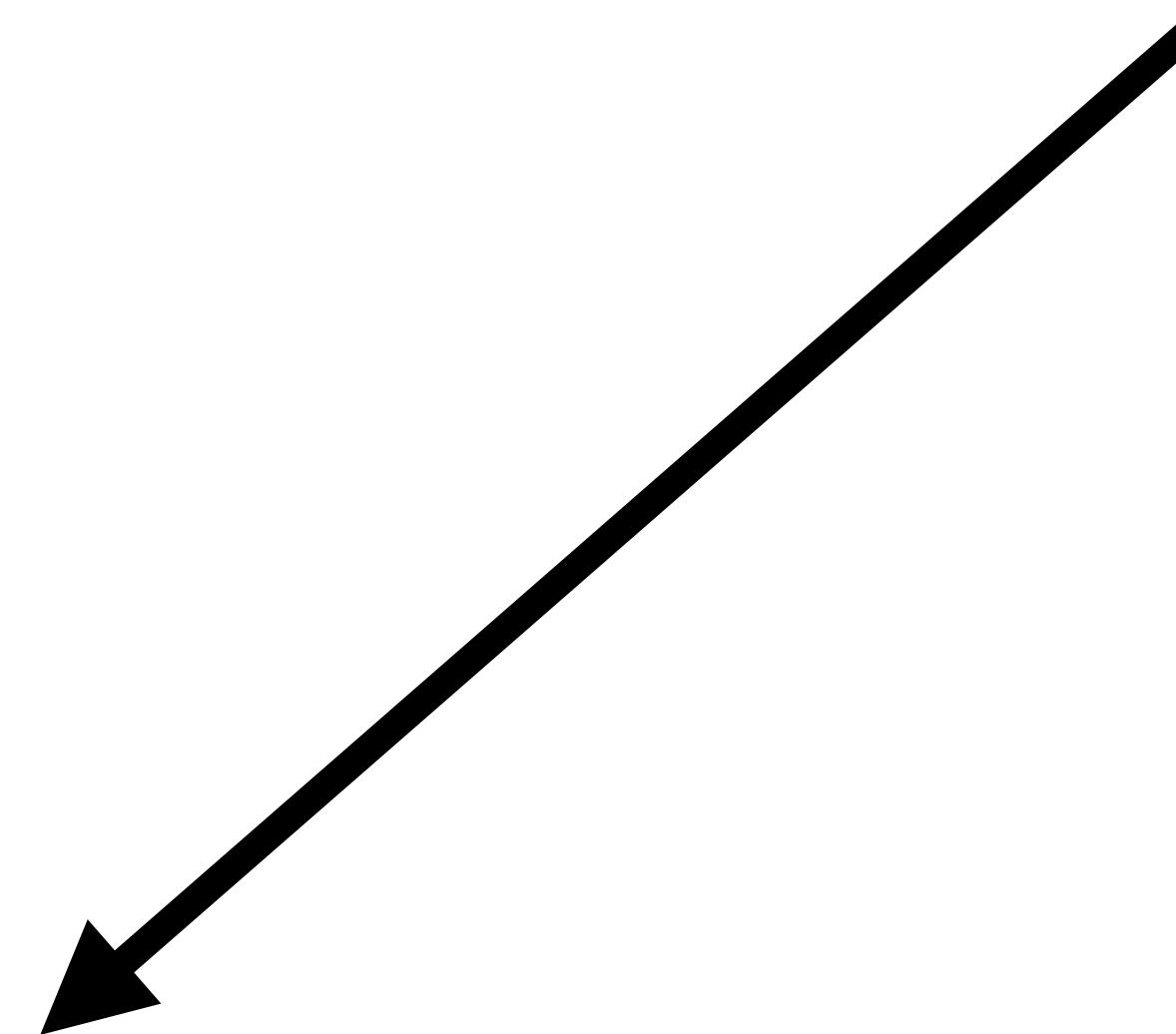
0x43

0x42

0x41

0x40

**head**



**stack = new Stack()**

**stack.push(3)**

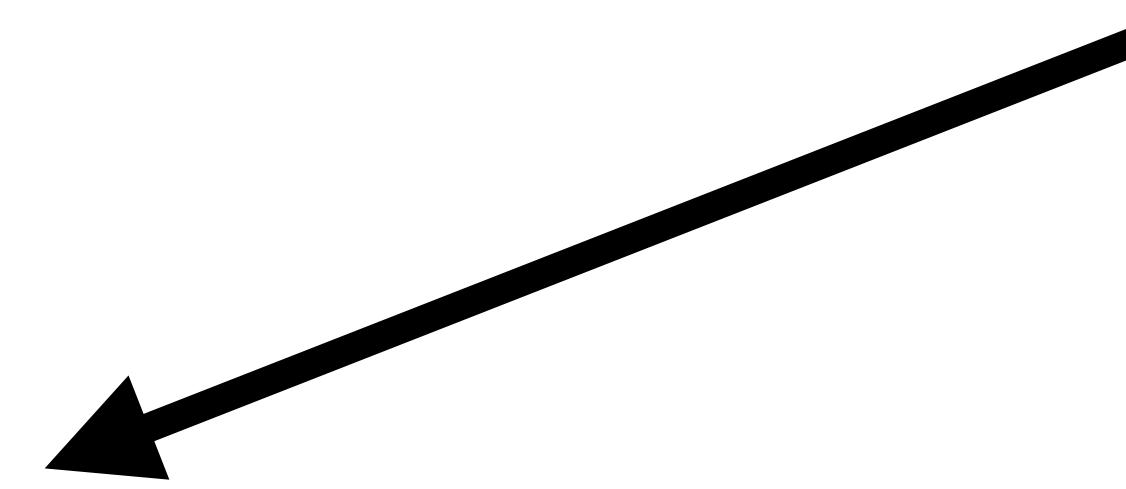
0x43

0x42

0x41

0x40

**head**



**stack = new Stack()**

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

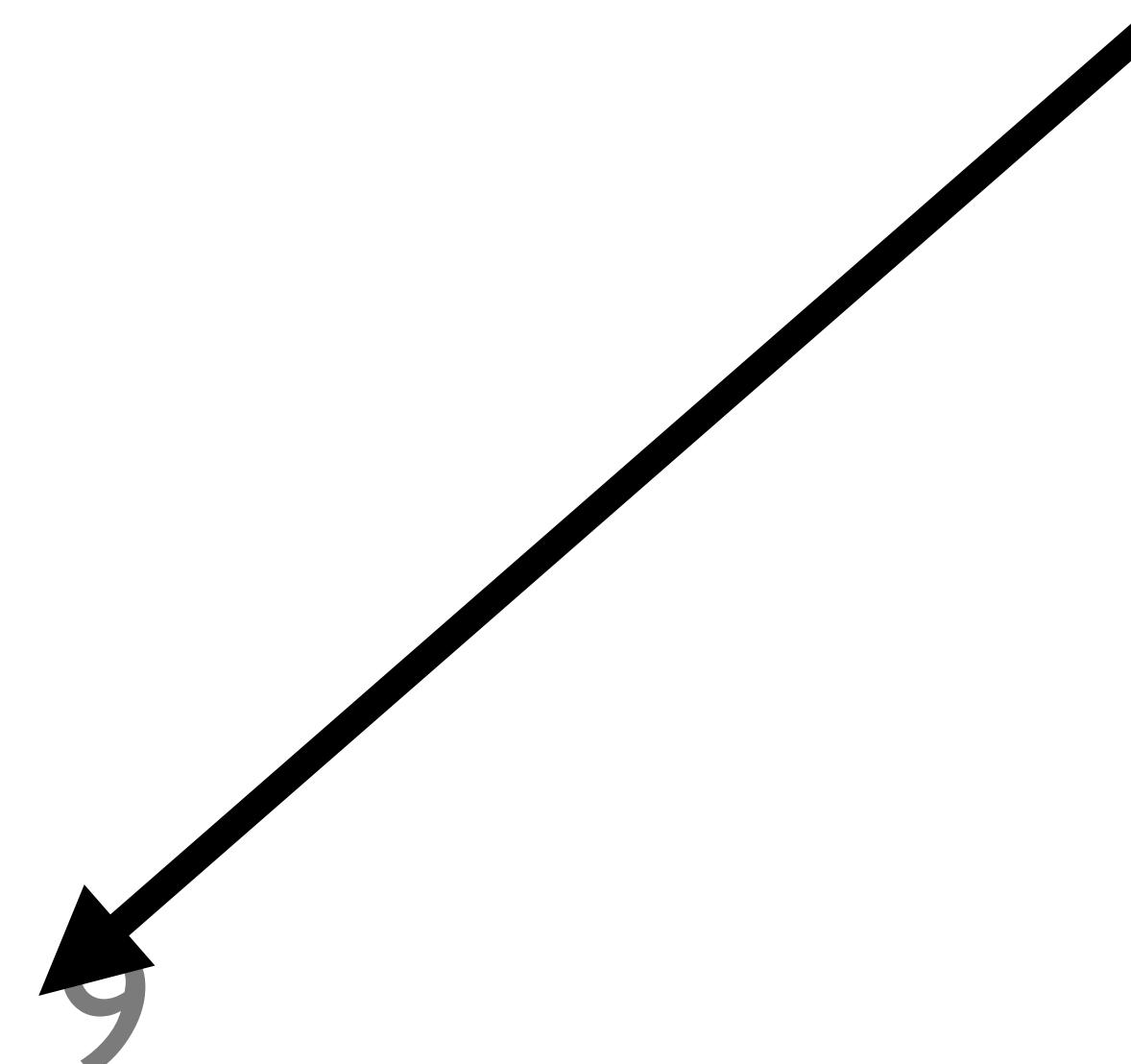
0x43

0x42

0x41

0x40

**head**



**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

# 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

A wide-angle photograph of a busy Apple Store interior. A massive crowd of people is gathered on a long, curved staircase and the floor level, all facing towards the left side of the frame where an Apple Store counter is located. The store has a modern design with white walls and large glass windows. The word "QUEUES" is overlaid in large, bold, black capital letters across the center-left of the image.

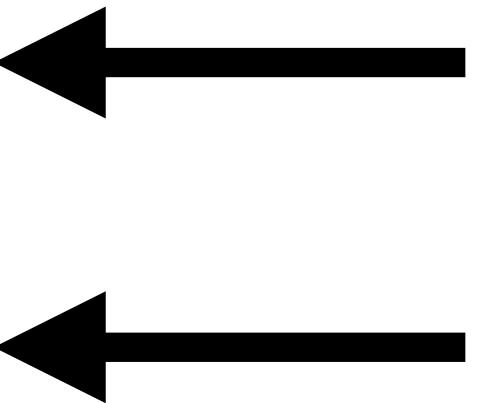
QUEUES

# 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

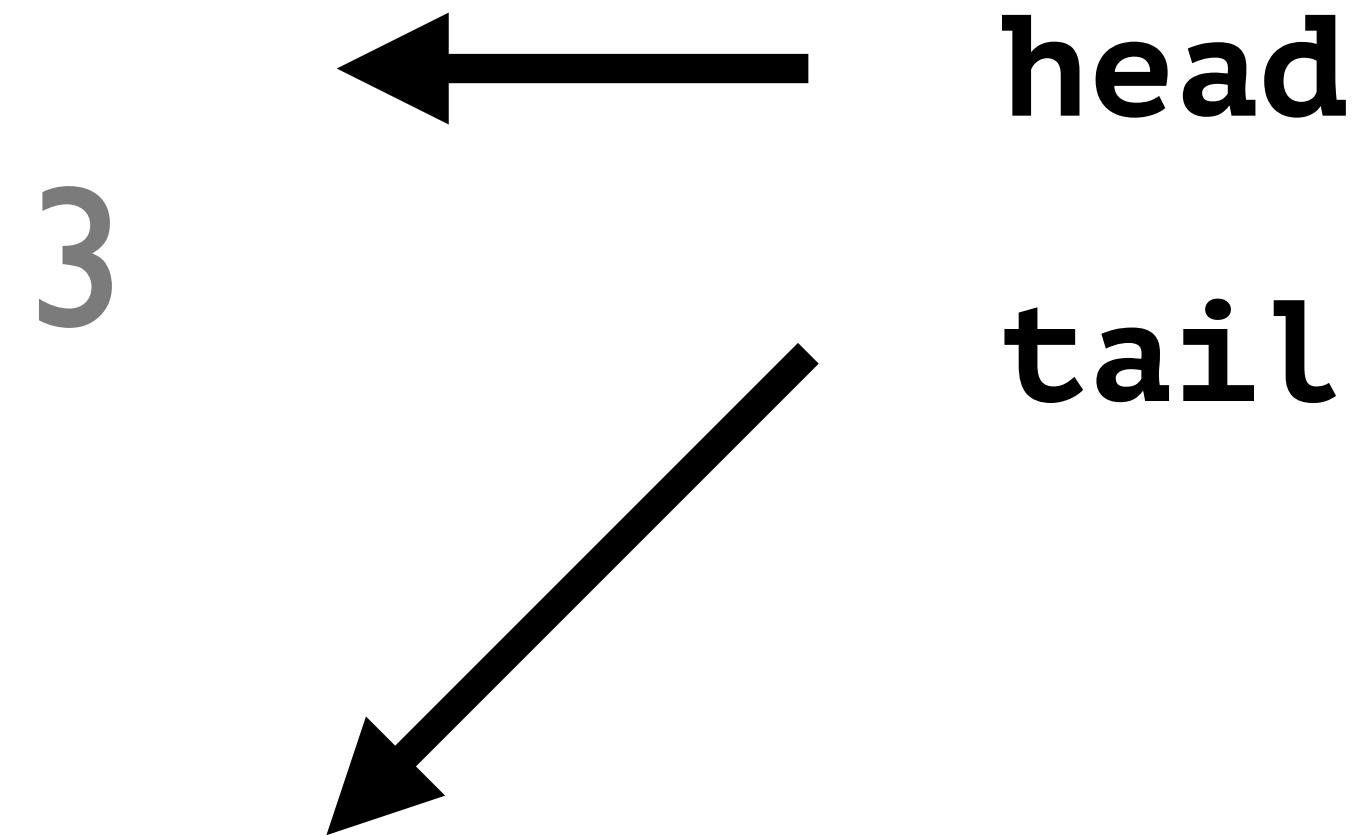
```
queue = new Queue()
```

0x40

0x41

0x42

0x43



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

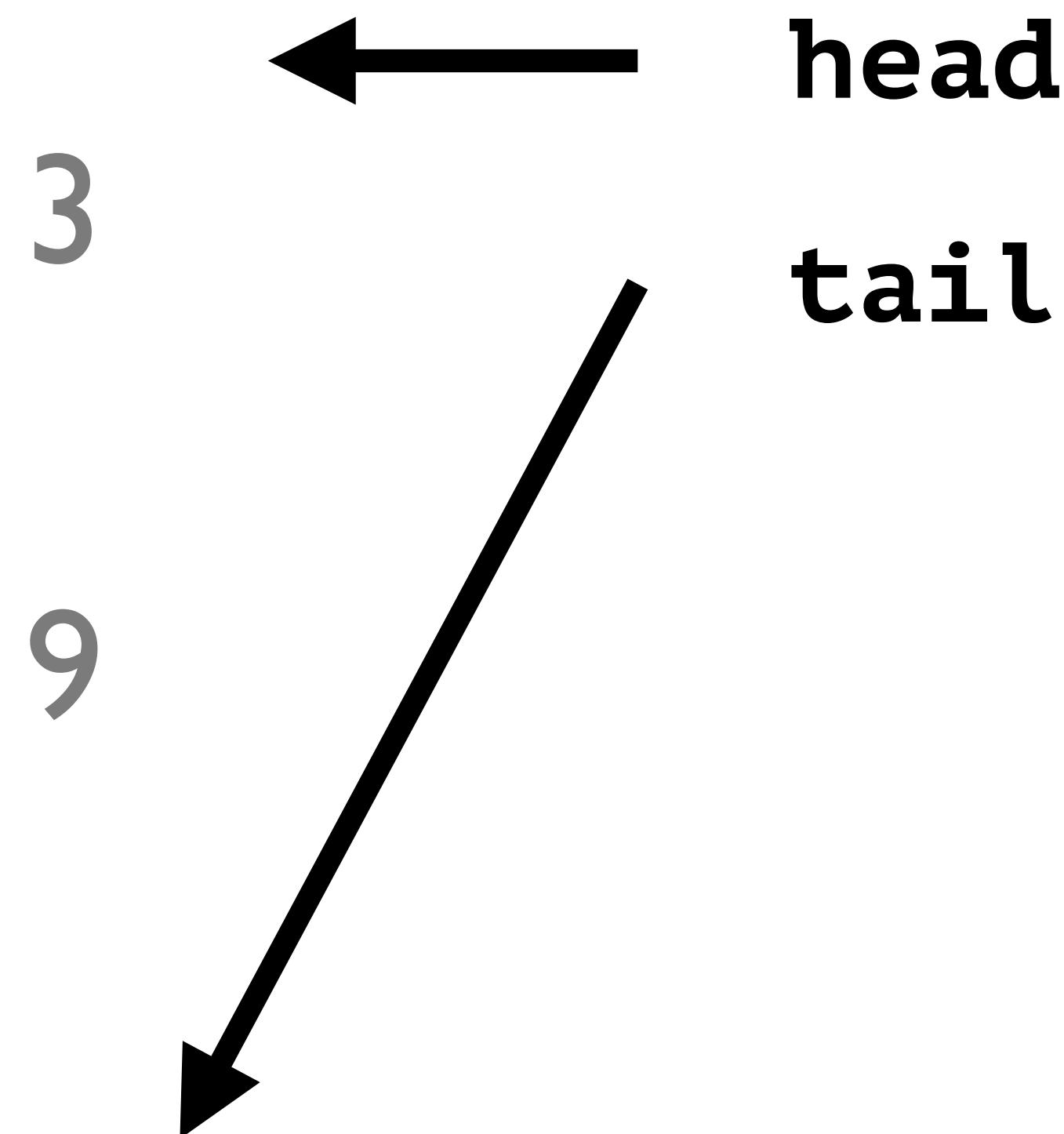
```
queue.enqueue(3)
```

0x40

0x41

0x42

0x43



`queue = new Queue()`

`queue.enqueue(3)`

`queue.enqueue(9)`

0x40

0x41

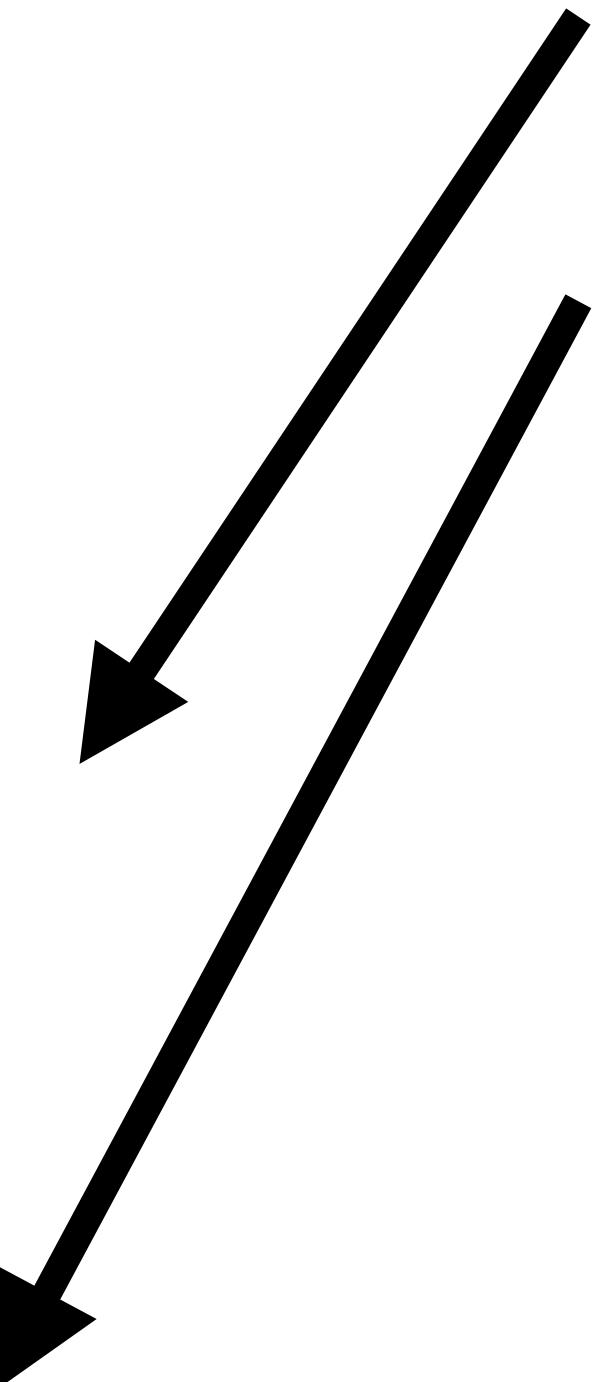
0x42

0x43

3

9

head  
tail



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

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

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

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

0x40

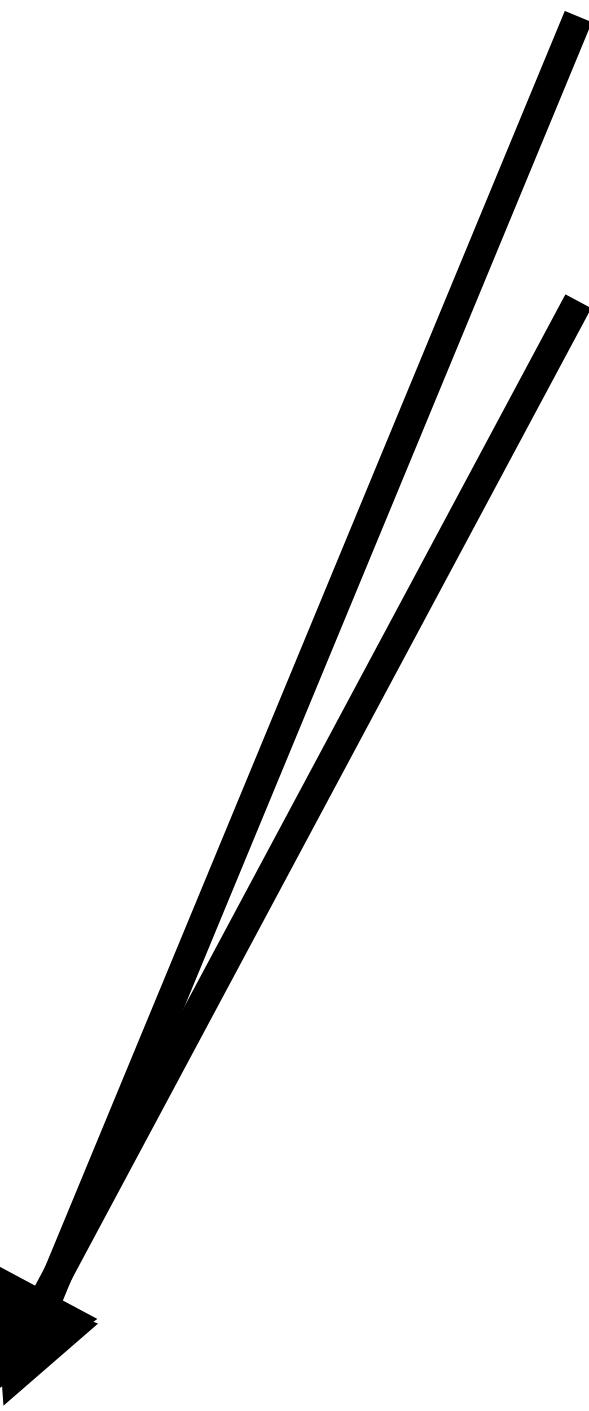
0x41

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

- ➊ **Arrays are NOT contiguous arrays**
  - Their implementation is left up to the JavaScript engine
  - ES6 - you can create typed arrays (ex. Uint8Array)
- ➋ **Simulate a stack**
  - push
  - pop
- ➌ **Simulate a queue**
  - `push` to enqueue
  - `shift` to dequeue
- ➍ **Data Structures take-home: implement a stack/queue *without* using these methods!**

