QUICK REVIEW

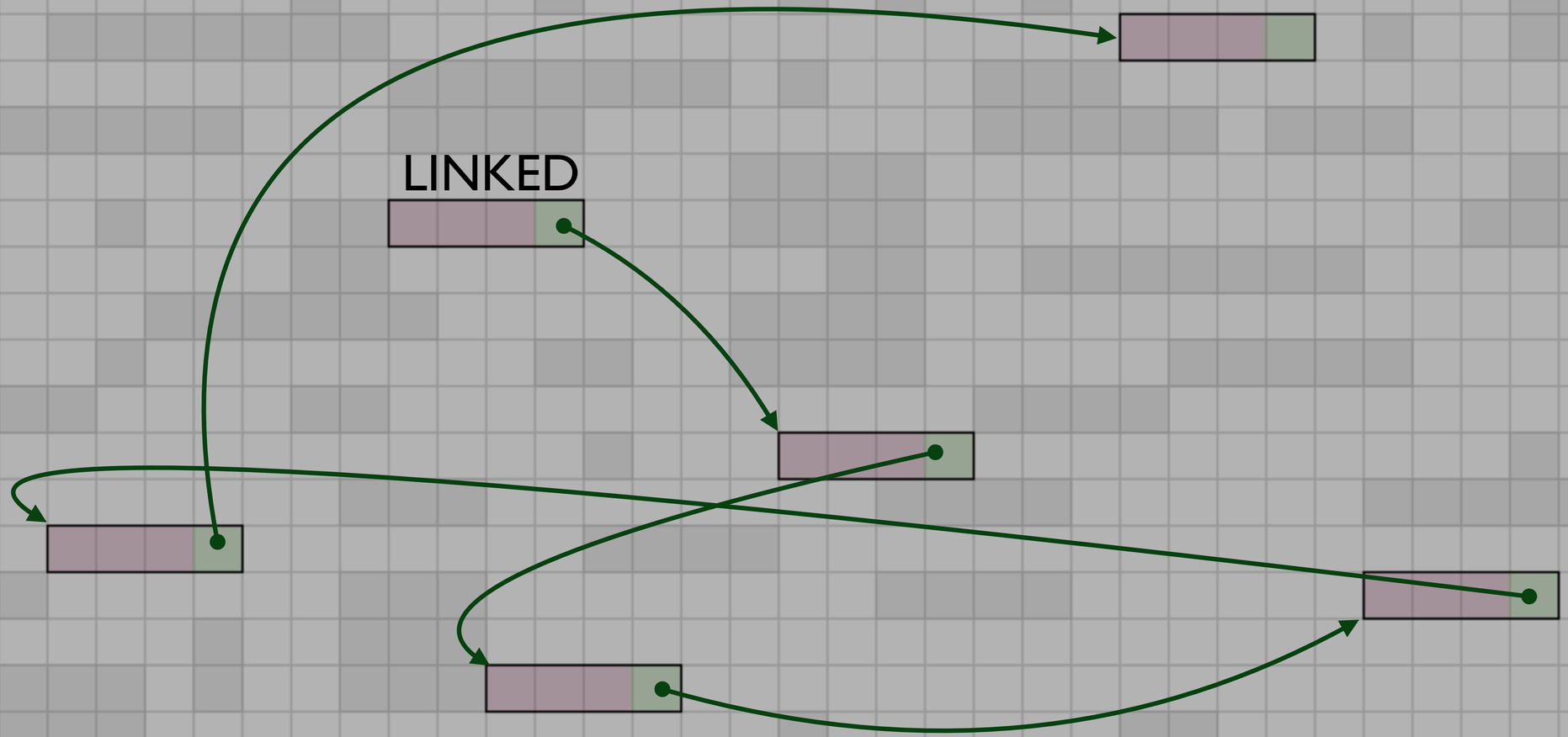
- **Primitive Data Types:**
  - Boolean
  - Number
  - Undefined
  - Null
  - String*
  - Symbol*
- **Complex Data Types:**
  - Objects: Arrays, RegExp, Date, Functions…and more!

QUICK QUIZ

- So how many BITS of memory does it take to represent a boolean value?
  - 1
- So how many BITS of memory does it take to represent an integer value?
  - 64
- So how many BITS of memory does it take to represent an object?
  - UNKNOWN :O

MEMORY

CONTIGUOUS



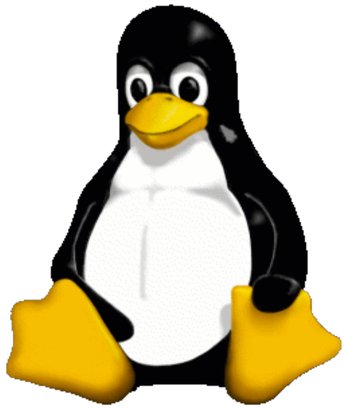
TRAJECTORY

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

MEMORY

Your application

Operating System

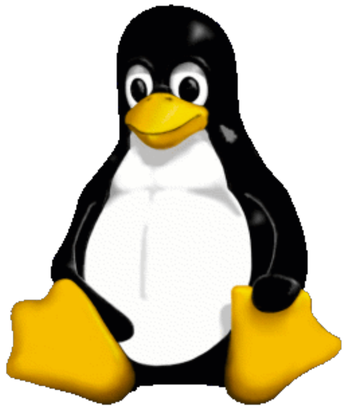


Memory

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

Your application

Operating System



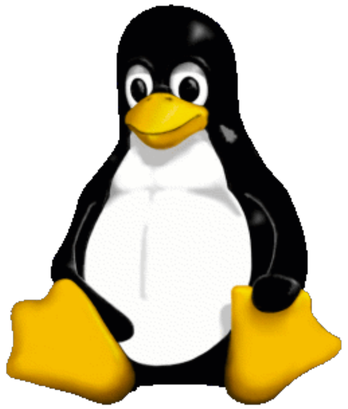
Memory

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



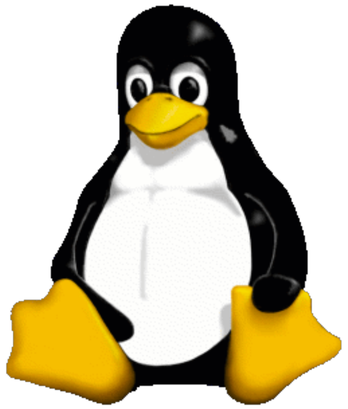
I'll see what we have available!

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


CONTIGUOUS ARRAY


0x40
0x41
0x42
0x43

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

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

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

```
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[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)?
```


STACKS

Stacks

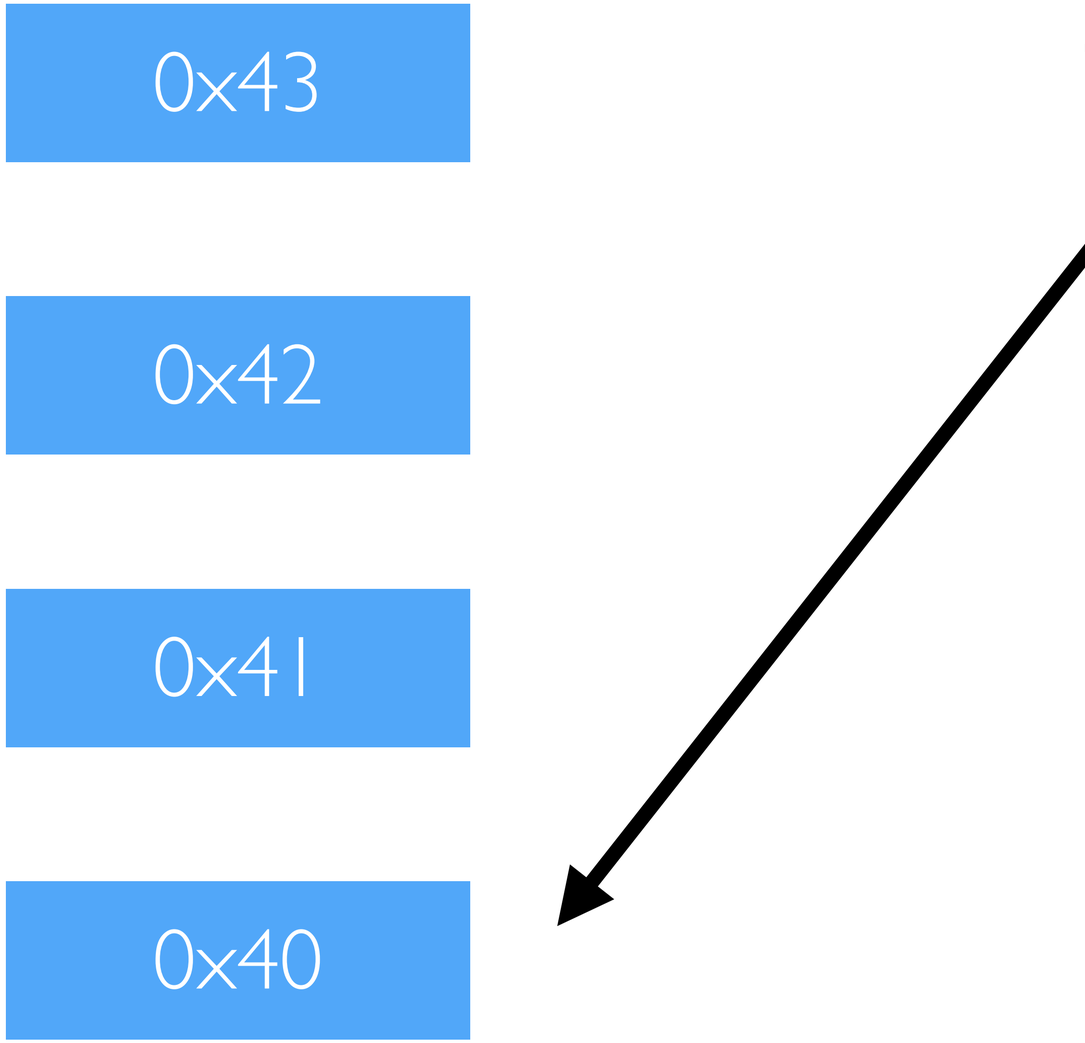
- **Collection of elements**
- **Ordered**
- **Elements can repeat (not a set)**
- **Some example operations:**
  - Make a new stack
  - Add an element to the stack ("push")
  - Retrieve an element from the list ("pop") - *must be LIFO (Last In, First Out)*
  - Check if stack is empty
  - Look at the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

0x43

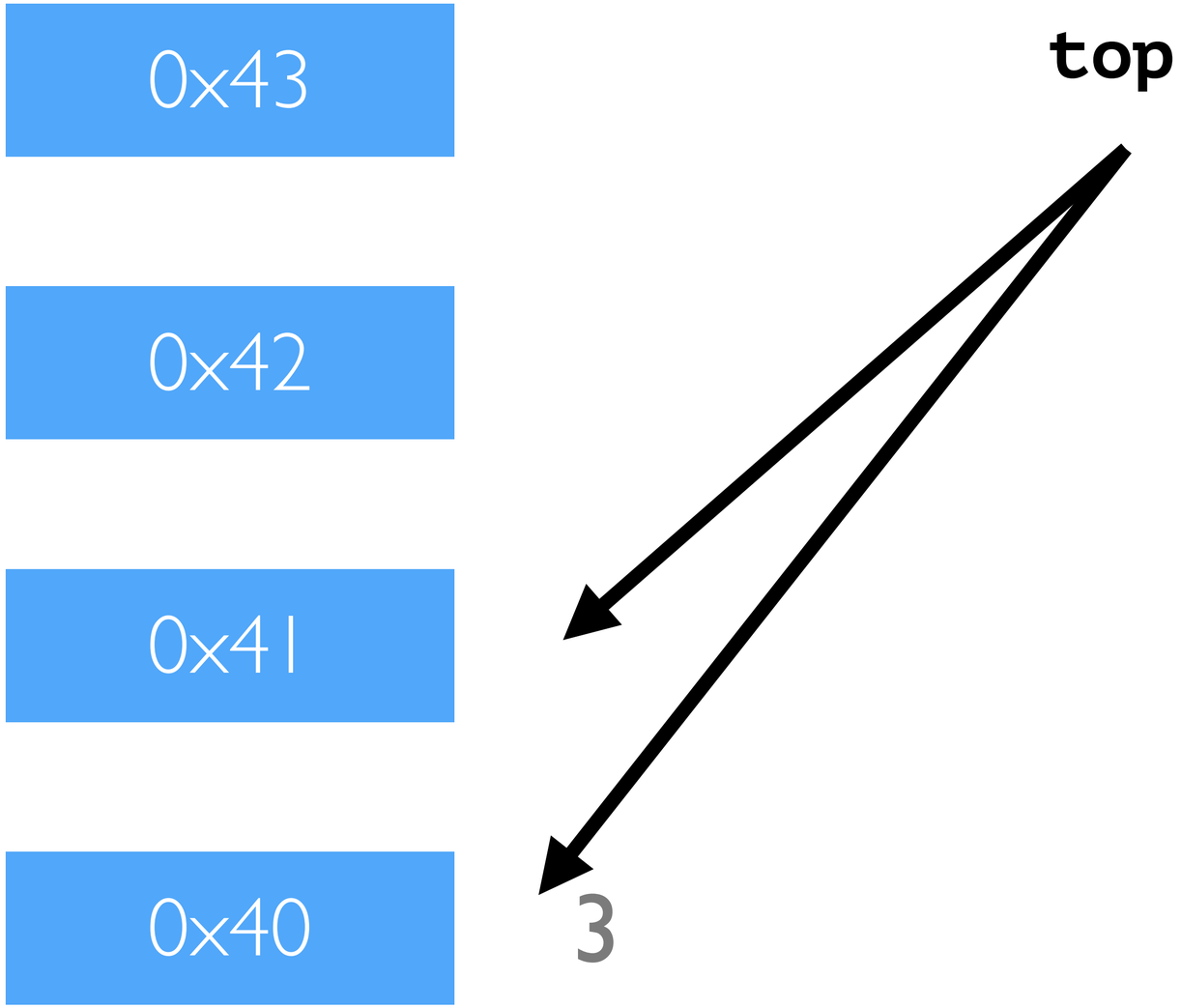
0x42

0x41

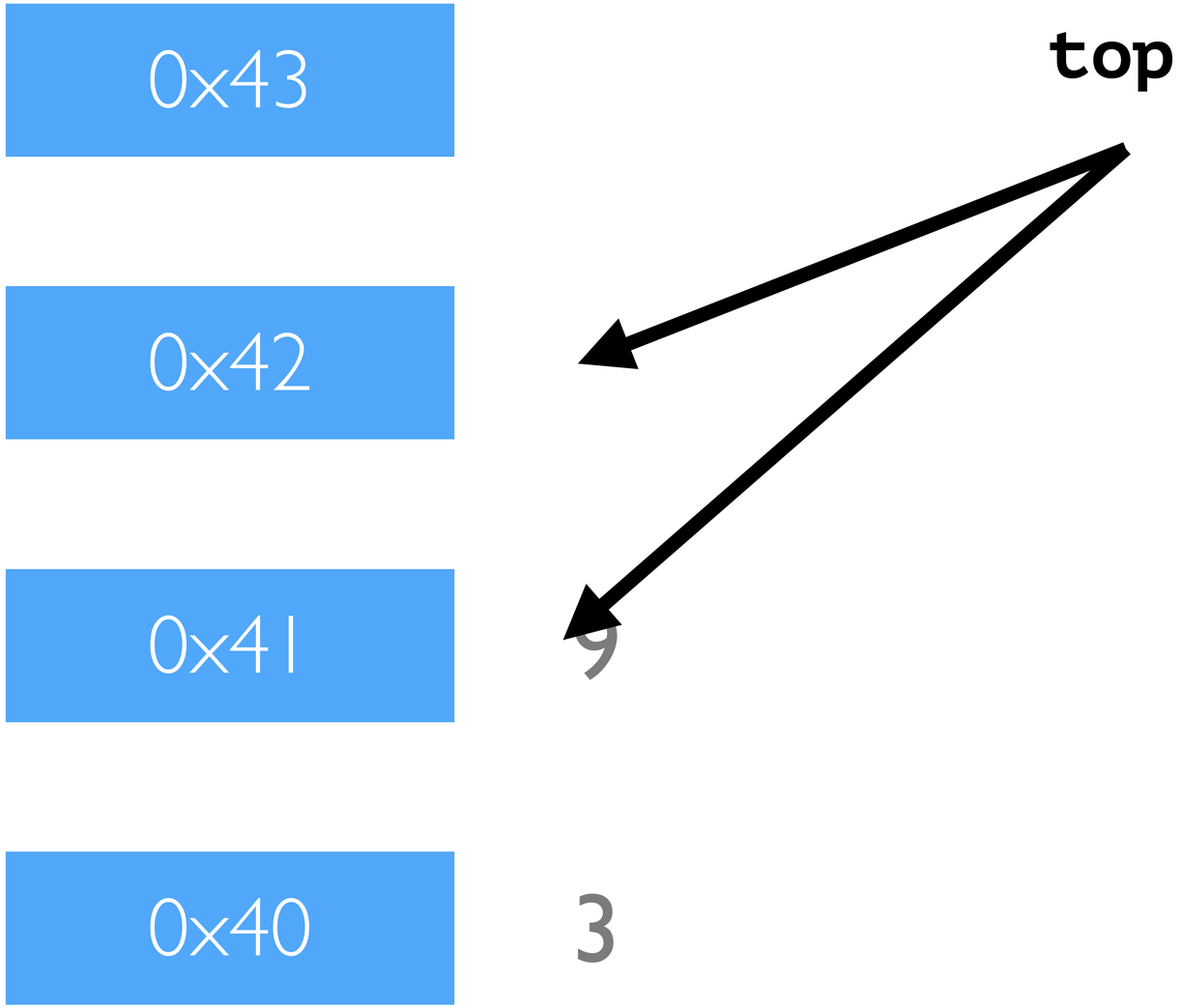
0x40



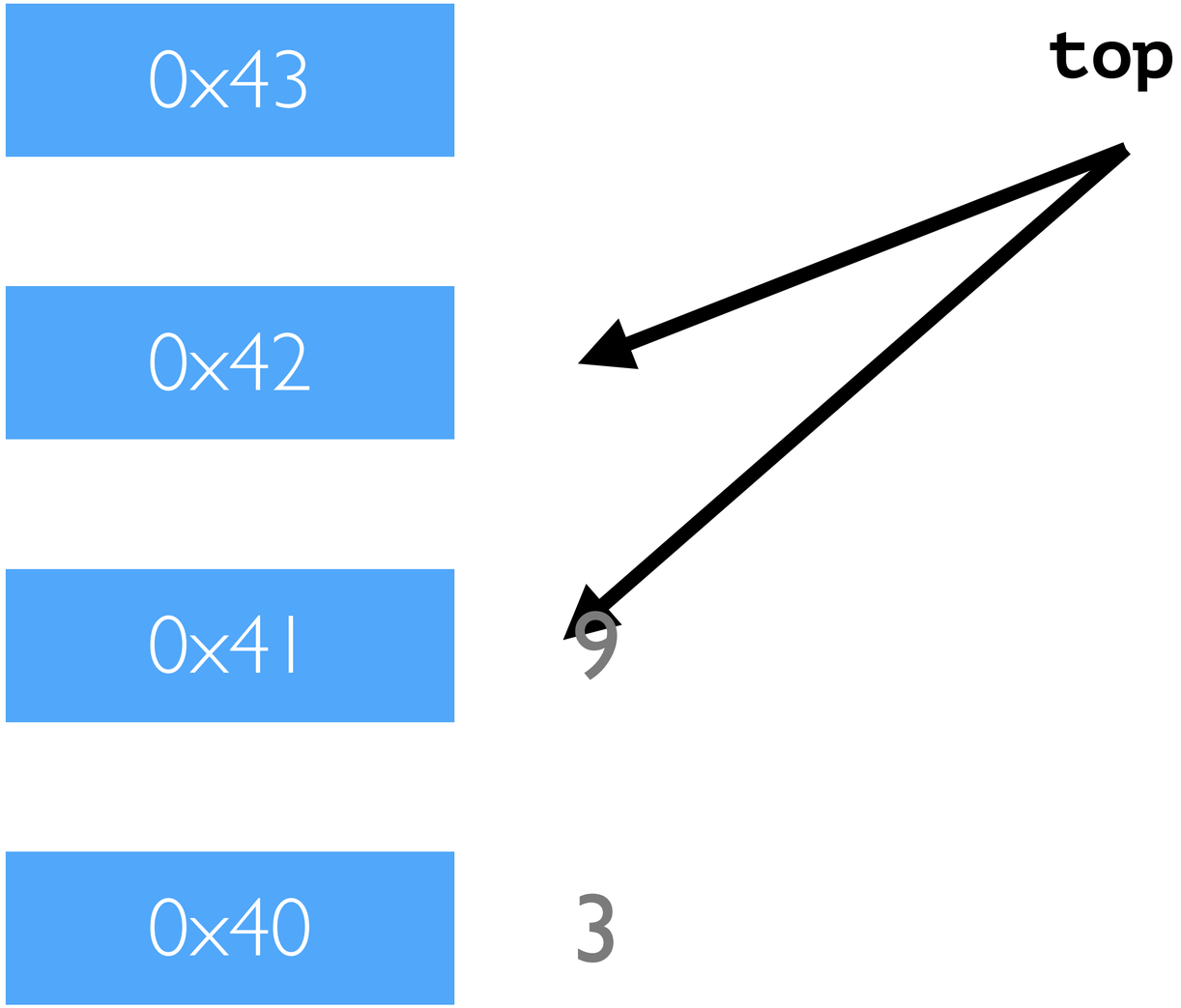

```
stack = new Stack()
```




```
stack = new Stack()

stack.push(3)
```




```
stack = new Stack()

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




```
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

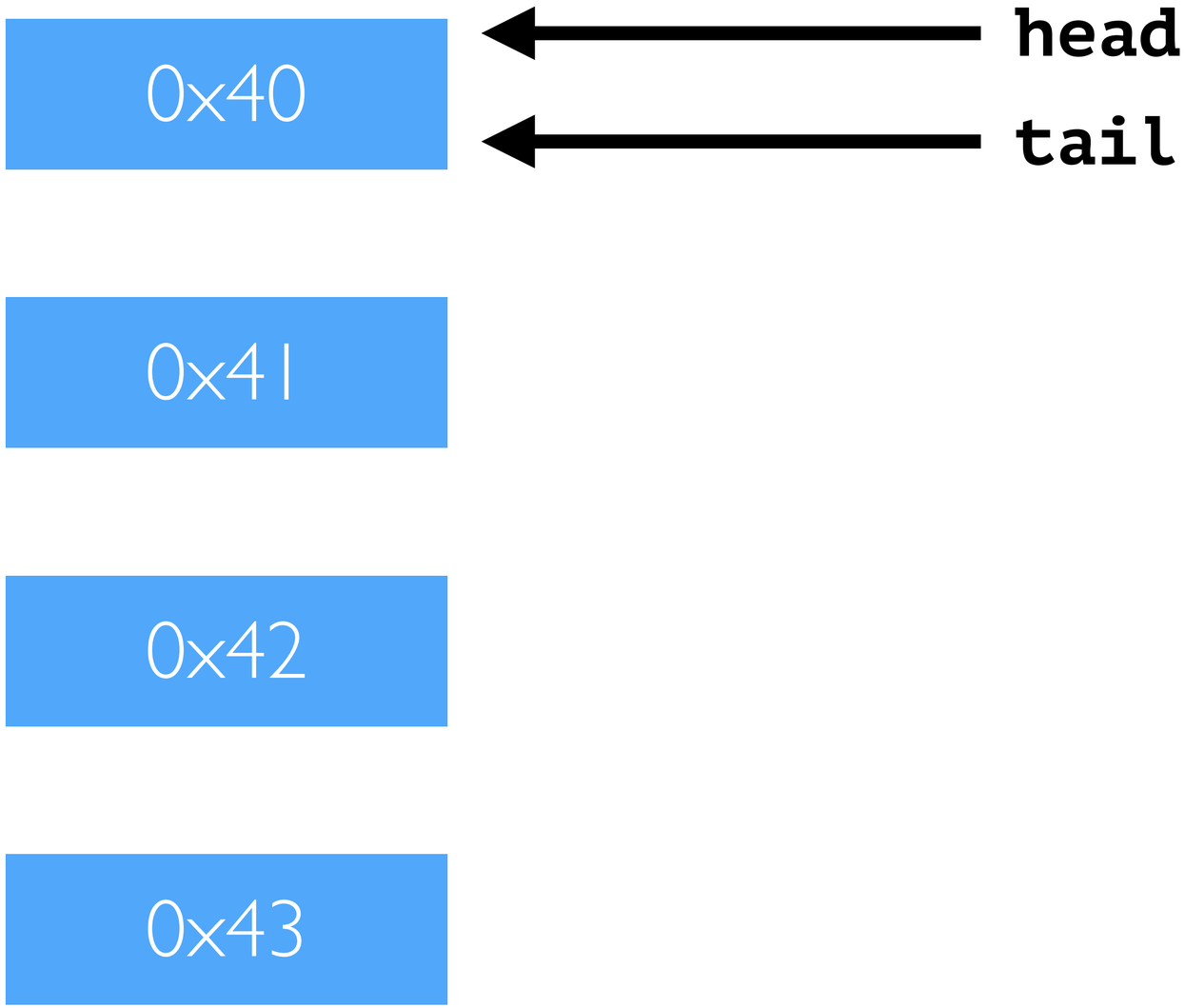
ADTs vs DSs

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

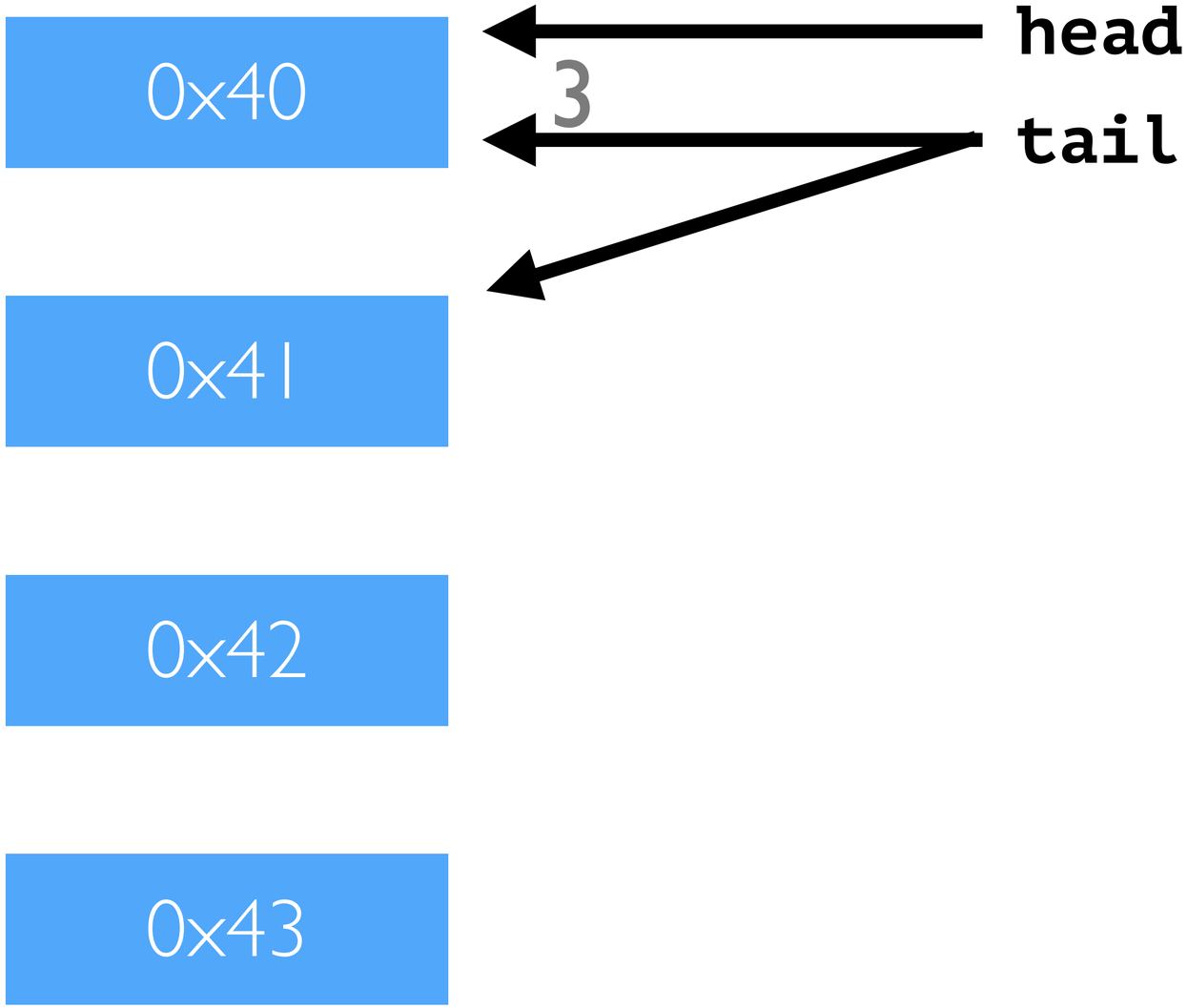
QUEUES

The Queue ADT

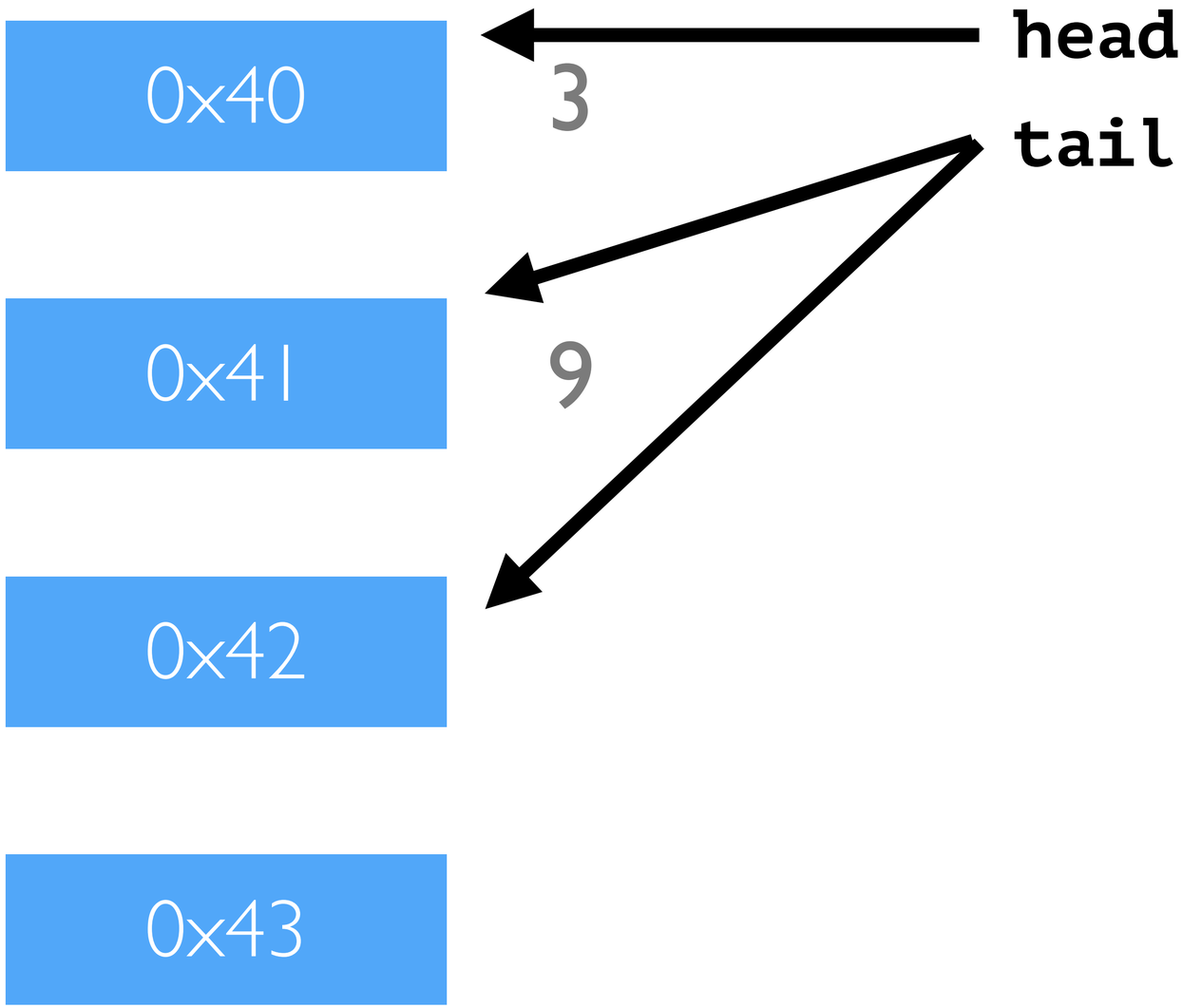
- Like Stack, except FIFO (First In, First Out)
- Collection of elements
- Ordered / sequential
- Operations:
  - Enqueue (add)
  - Dequeue (remove)
  - Peek
  - Clear
  - IsEmpty… etc.



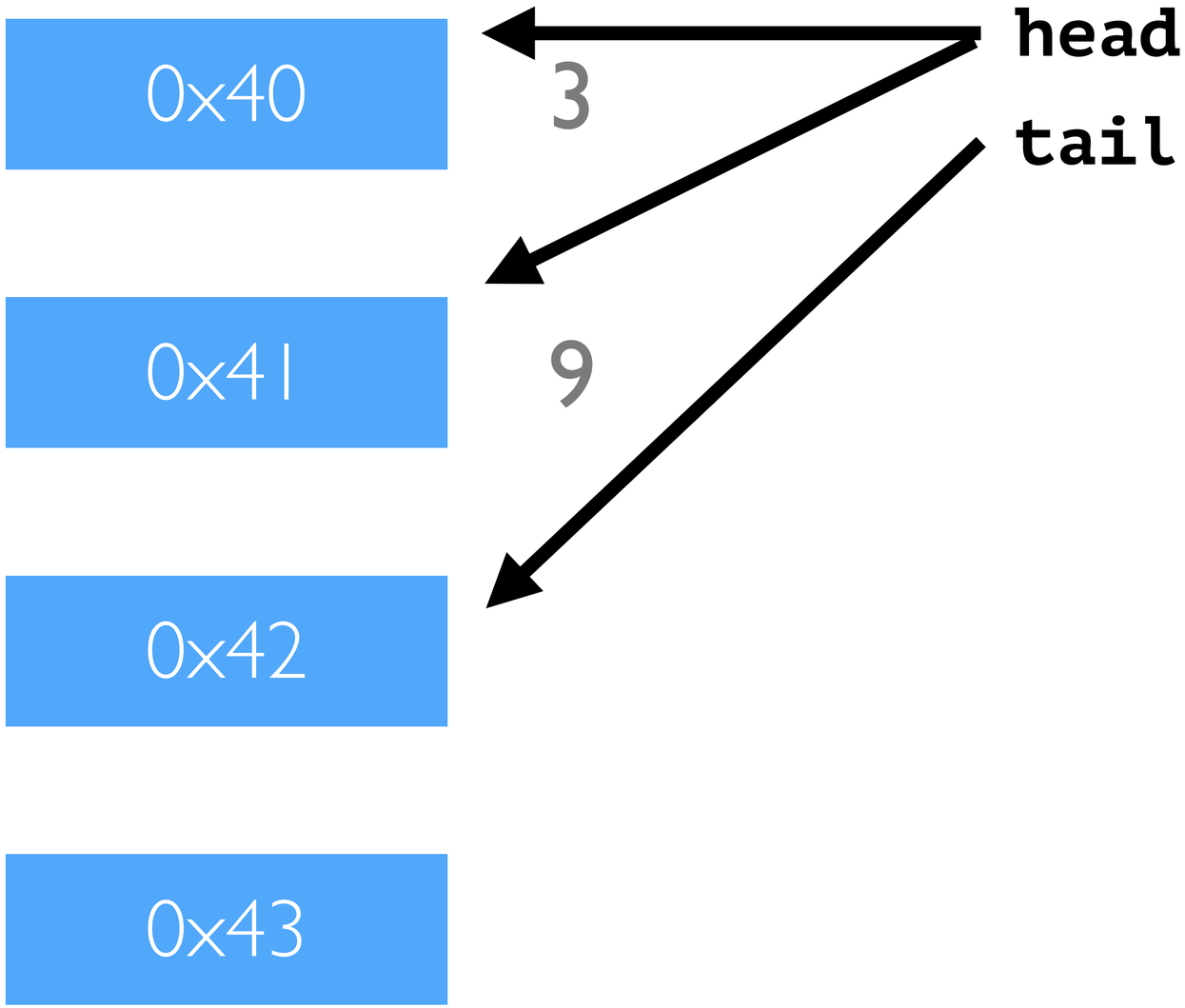

```
queue = new Queue()
```



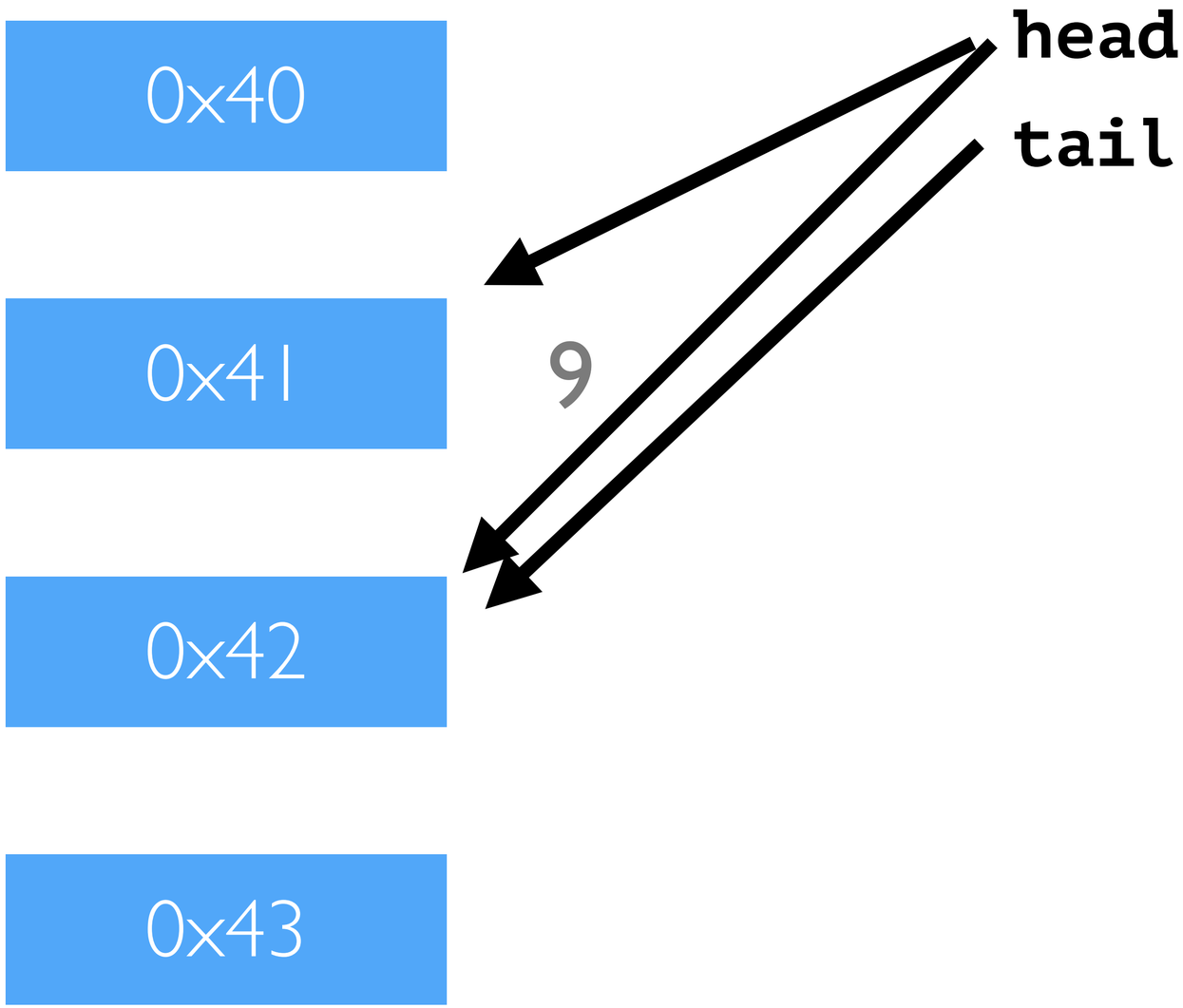

```
queue = new Queue()

queue.enqueue(3)
```




```
queue = new Queue()

queue.enqueue(3)
queue.enqueue(9)
```




```
queue = new Queue()

queue.enqueue(3)
queue.enqueue(9)

queue.dequeue() // 3
```




```
queue = new Queue()

queue.enqueue(3)
queue.enqueue(9)

queue.dequeue() // 3
queue.dequeue() // 9
```

WHAT ABOUT JS?

In JavaScript

- **Arrays are NOT (necessarily) contiguous arrays**
  - Their implementation is left up to the JavaScript engine
  - ES6 - you can create typed arrays (ex. Uint8Array)

- **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!**