GENERAL ASSEMBLY

ARC & THREADING

Zeke Abuhoff

Lead iOS Instructor, General Assembly

LEARNING OBJECTIVES

+ Explain the basic mechanics of automatic reference counting
+ Describe the difference in memory management between classes and structs
+ Explain the basic mechanics of using multiple threads in an iOS app
+ Debug a threading deadlock

MEMORY

We know that data you don't save to disk is only stored in memory.

But how does memory work?

MEMORY

How does memory work?

Theory 1	All data stays in memory forever.

```
let firstString = “Case Keenum”
let secondString = “Jared Koff”
let thirdString = “Sean Mannion”
// etc...
```

MEMORY

How does memory work?

Theory 1	All data stays in memory forever.

```
let firstString = “Case Keenum”
let secondString = “Jared Koff”
let thirdString = “Sean Mannion”
// etc...
```

ERROR
ran out of memory

MEMORY

How does memory work?

Theory 2	At the close of any scope, all data referenced is deleted.

```
func newUser() -> String {
    let user = User()
    return user
}
```

MEMORY

How does memory work?

Theory 2	At the close of any scope, all data referenced is deleted.

```
func newUser() -> String {
    let user = User()
    return user
}
```

ERROR
bad access

MEMORY

How does memory work?

Theory 3	The system tracks references. If something has 0 references, it's deleted.

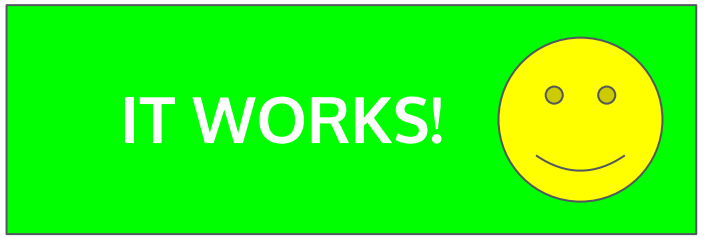
```
func printName(user: User) {
    let userName = user.name
    print(userName)
}
```

MEMORY

How does memory work?

Theory 3	The system tracks references. If something has 0 references, it's deleted.

```
func printName(user: User) {
    let userName = user.name
    print(userName)
}
```

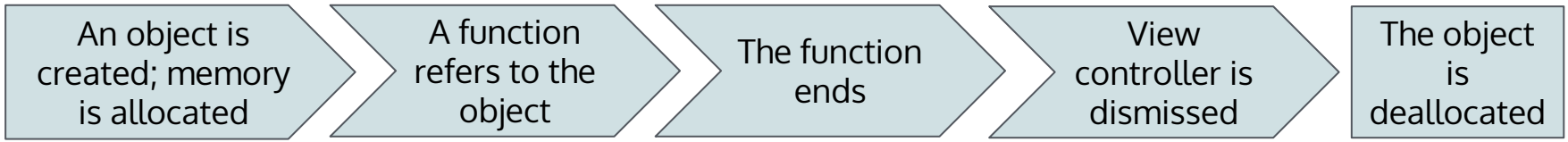


ARC

iOS apps manage memory with **Automatic Reference Counting**.

ARC is a system that keeps track of how many places each object is referenced at runtime.

Once no currently running code has a reference to an object, its memory gets deallocated - it's gone.



ARC

There are two kinds of references in ARC:

`weak` and `strong`

Weak references DO NOT increment the reference count.

Strong references DO increment the reference count.

ARC

Food for thought:

1) Class instances use ARC but struct instances don't. Why is that?
2) Why do we have both strong and weak references?
3) IBOutlets are weak references. Why not strong?

THREADING

We know from our work with HTTP requests that an iOS app can multitask.

```
DispatchQueue.main.async { }

DispatchQueue(label: “background”).sync { }
```

So far, we’ve used threading to move operations to the main queue, because we can’t update the UI on background threads.

But what else can go wrong?

THREADING

For one thing, mismanaged threads can lead to **race conditions**. A race condition is a bug where the app's behavior changes significantly because of small changes in timing.

Example

- Retrieving an image from a url takes about a second.
- The table view that will display this image displays nothing at first, waits 1 second, then reloads its data, expecting an image to have been retrieved.
- If the image is retrieved quickly, the app works fine. If the image takes longer than 1 second to load, the user never sees the image. To the user, the app seems broken half the time for no reason.

THREADING

Mismanaged threads can also end up deadlocked. A **deadlock** occurs when two threads are both waiting for each other to finish an operation before continuing. This part of the app becomes frozen.

Example

- Background thread 1 needs to update the UI. It synchronously executes a block on the main thread.
- One of the lines in the block calls a particular function. That function happens to synchronously execute on background thread 1.
- Background thread 1 won't perform an operation until the main thread finishes the block. The block won't finish until background thread 1 performs an operation.

THREADING

Food for thought:

1) Why isn't everything on the main thread?
2) Why isn't everything on a background thread?
3) How many threads is too many?