iOS Foundations II
Day 2

- OOP Review
- The story of an App
- View Controllers
- Design Patterns/MVC
- IBOutlet & IBAction
- Arrays

OOP review

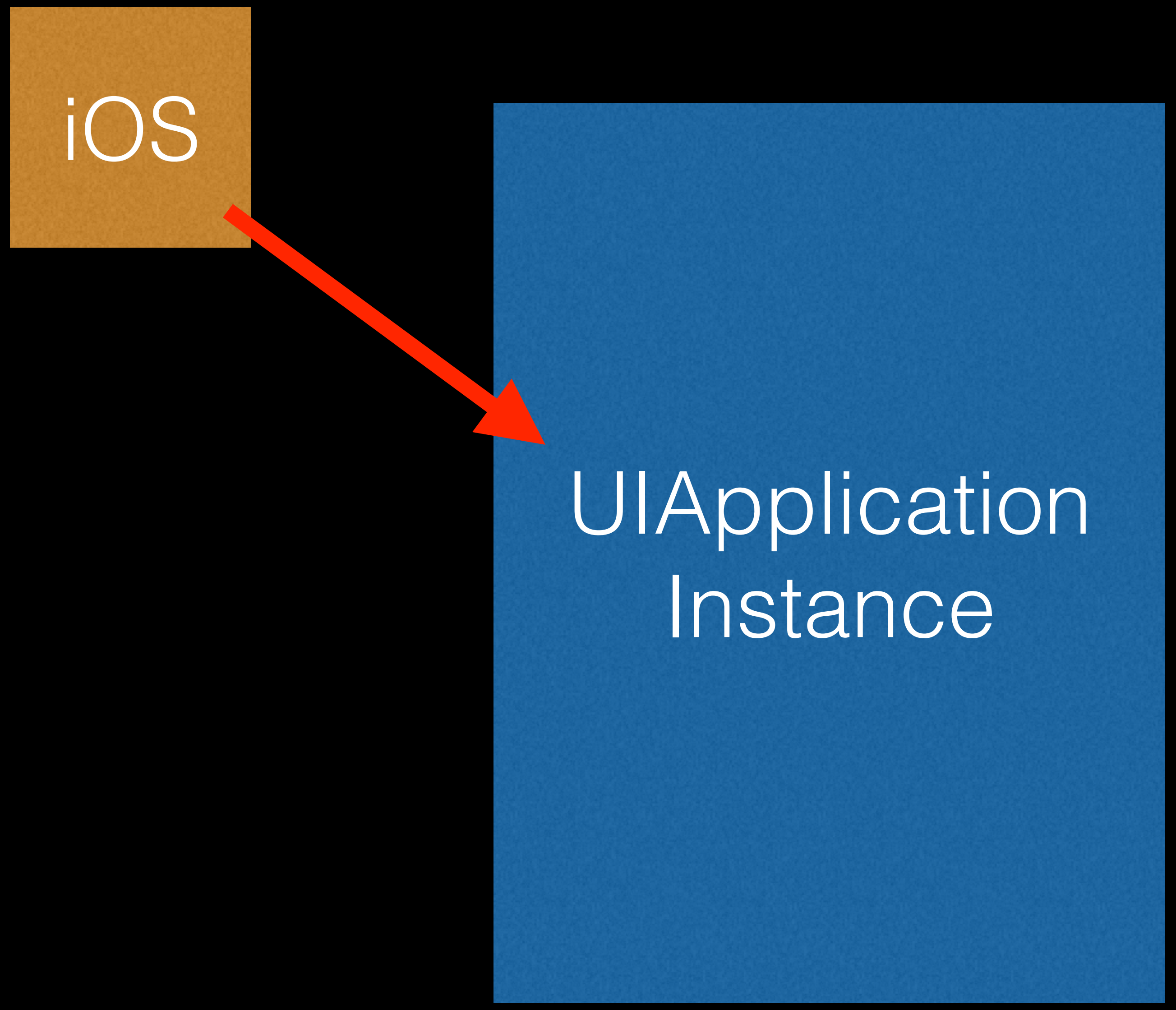
What happens when you run an app?

Course Adaptations

- Real-time visual feedback - thumbs up/flat/down
- Slide content: less detail, fewer bullets
- Feedback: It’s great to offer relevant/interesting info
- Shared teaching goal: each student will use AirPlay or whiteboard once per class
- Class project for extra credit. Discuss scope, design. (e.g., UI for castles?)

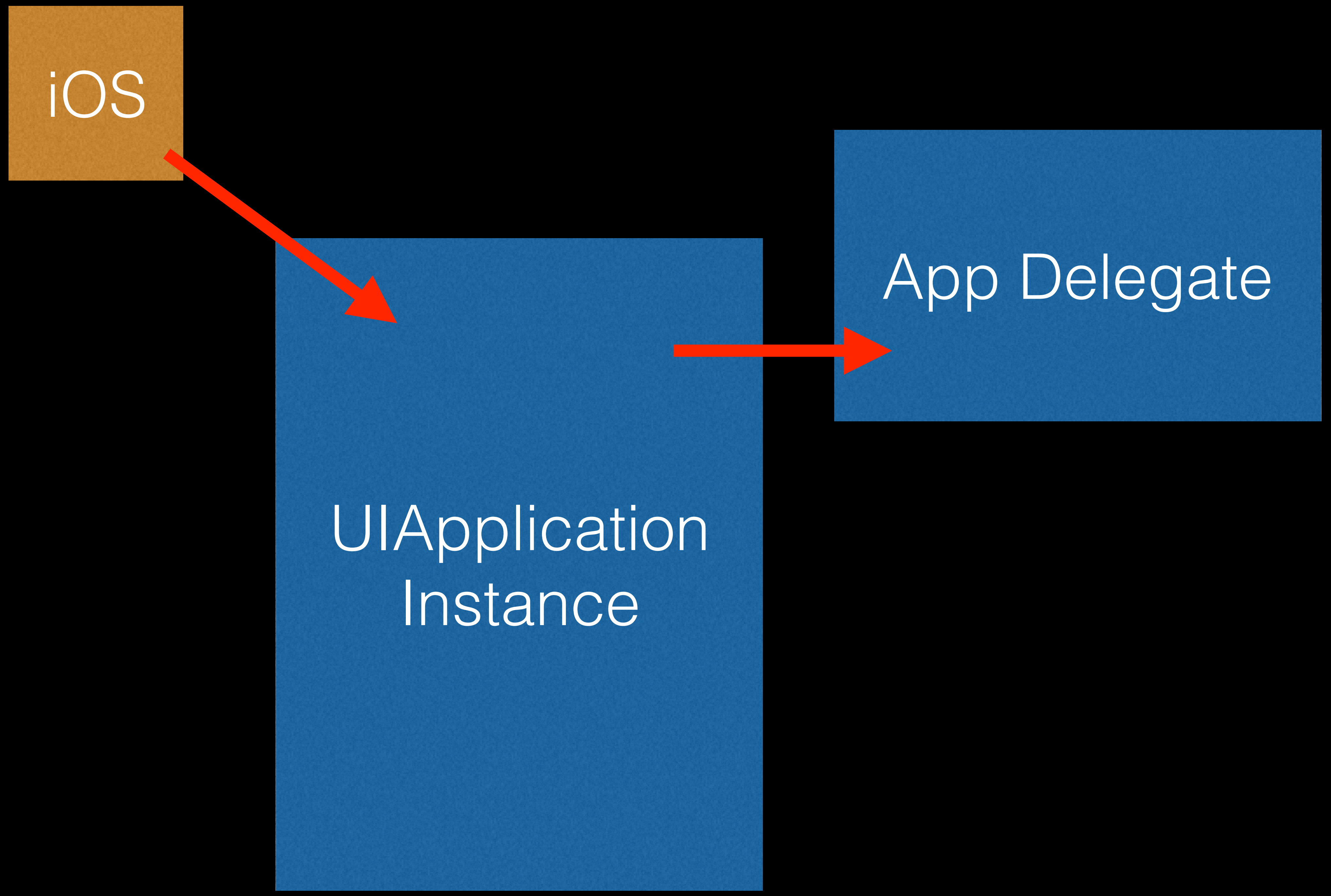
Step 1

- When your app is first launched, either from the home menu screen of the device or by hitting Play in Xcode:



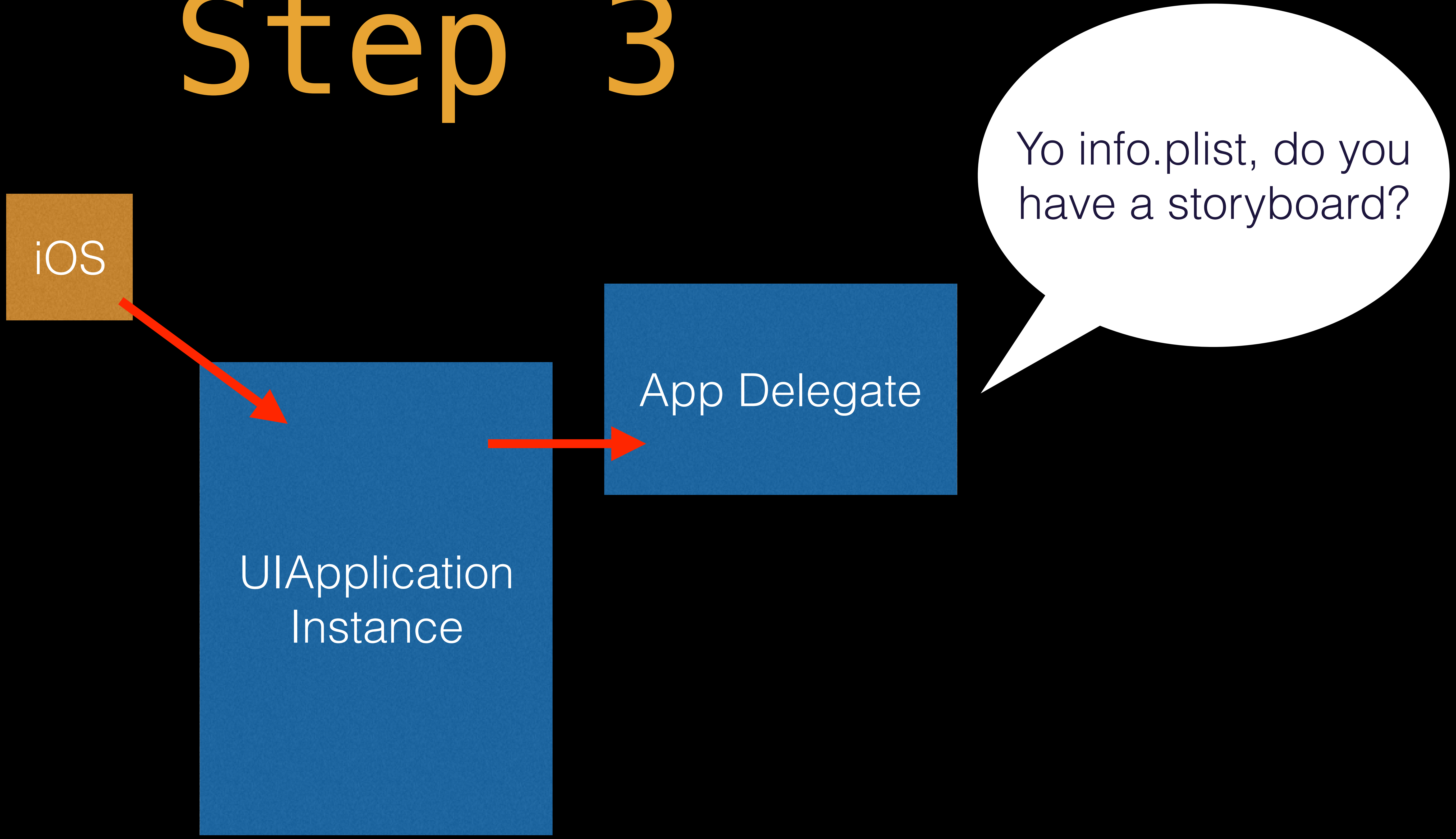
**An instance of the UIApplication class is created, representing your app. It is held
onto (owned) by the operating system (iOS).**

Step 2



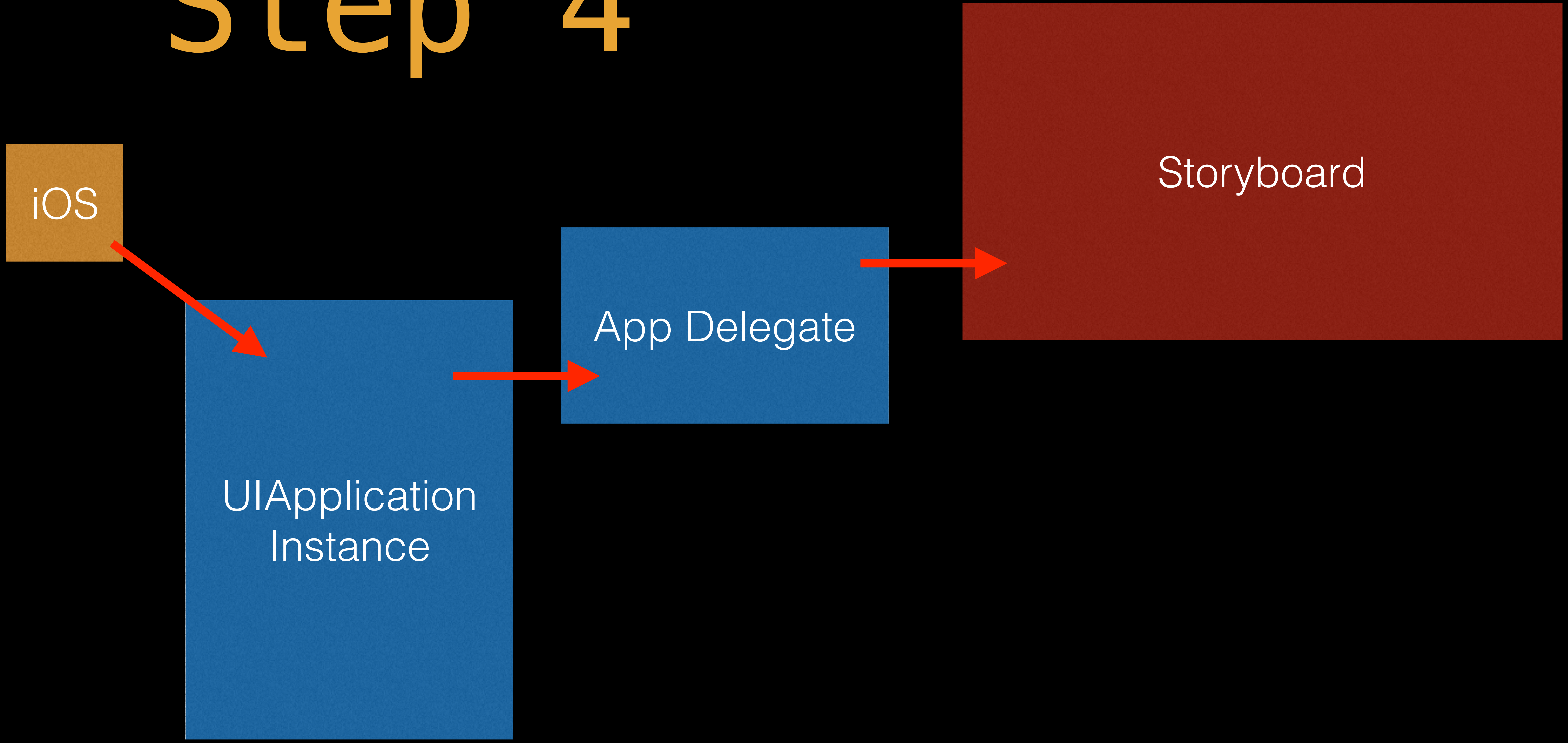
An instance of the UIApplicationDelegate class is created by the UIApplication instance, and is owned by the UIApplication Instance

Step 3



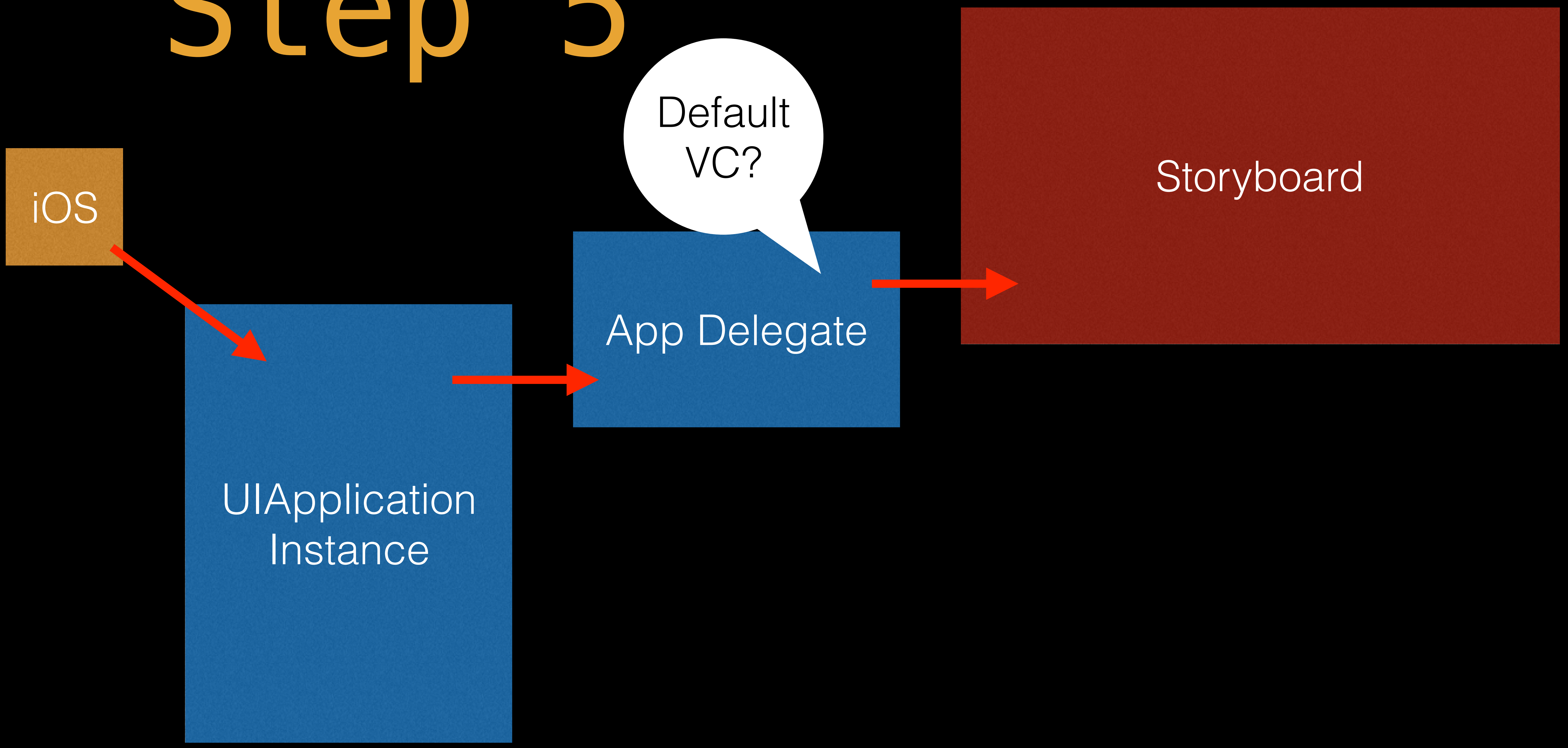
App Delegate looks at your app’s project settings (info.plist), and sees if you have an initial storyboard or not

Step 4



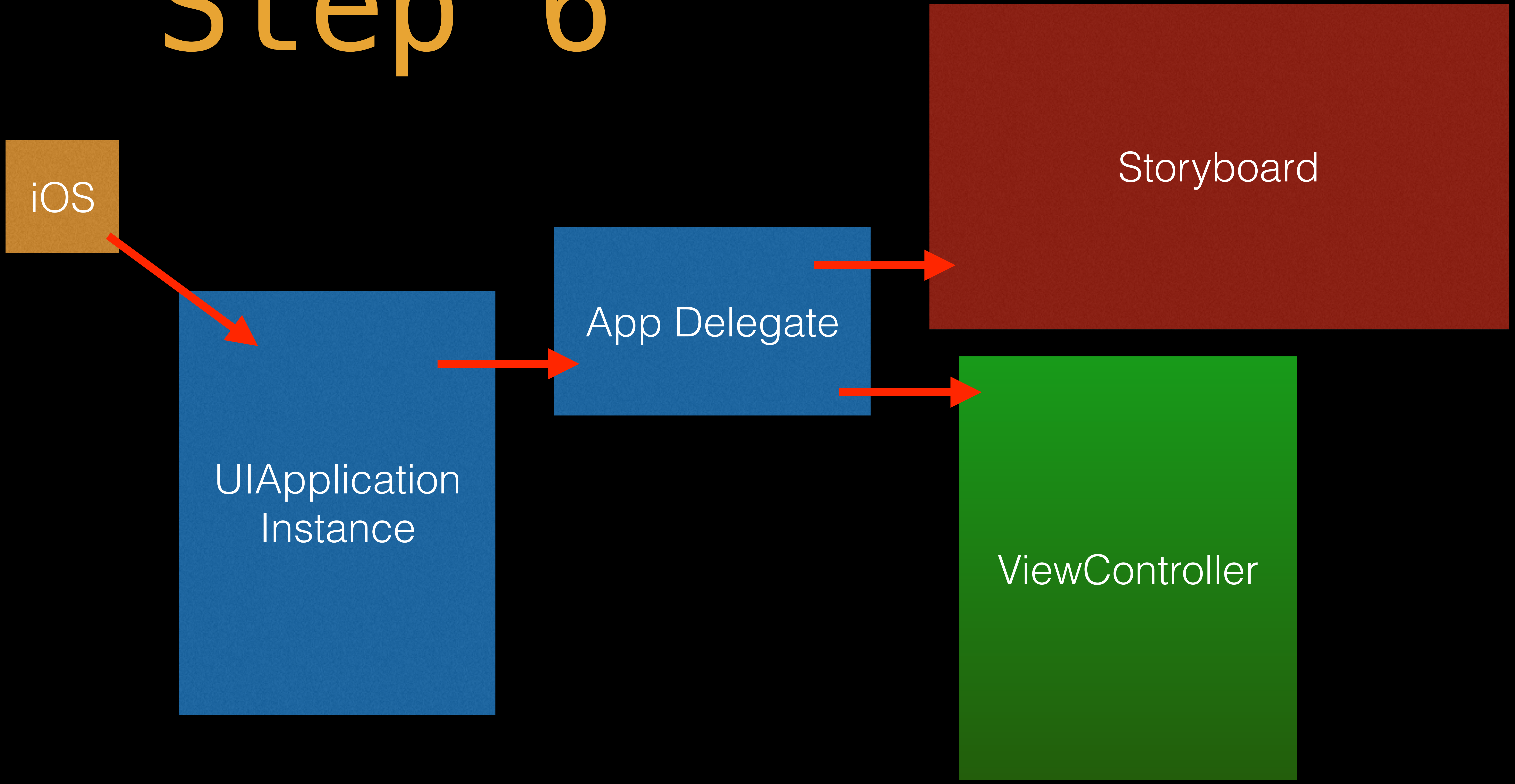
The answer was yes, so an instance of UIStoryboard is created from the storyboard you setup in Xcode

Step 5



App delegate asks Storyboard, do you have an initial view controller? If so, what is it?

Step 6



Storyboard finds the view controller/scene you marked as the initial view controller, instantiates an instance of it WITH ALL THE COOL STUFF YOU ADDED VIA STORYBOARD ADDED TO IT. App Delegate now owns this VC, and your app is now ready to be used.

Step 6

iOS
UIApplication Instance
App Delegate
Storyboard
ViewController
Detail ViewController
Favorites ViewController

As the user progresses through the app, this 'object graph' will continue to grow (and shrink!)

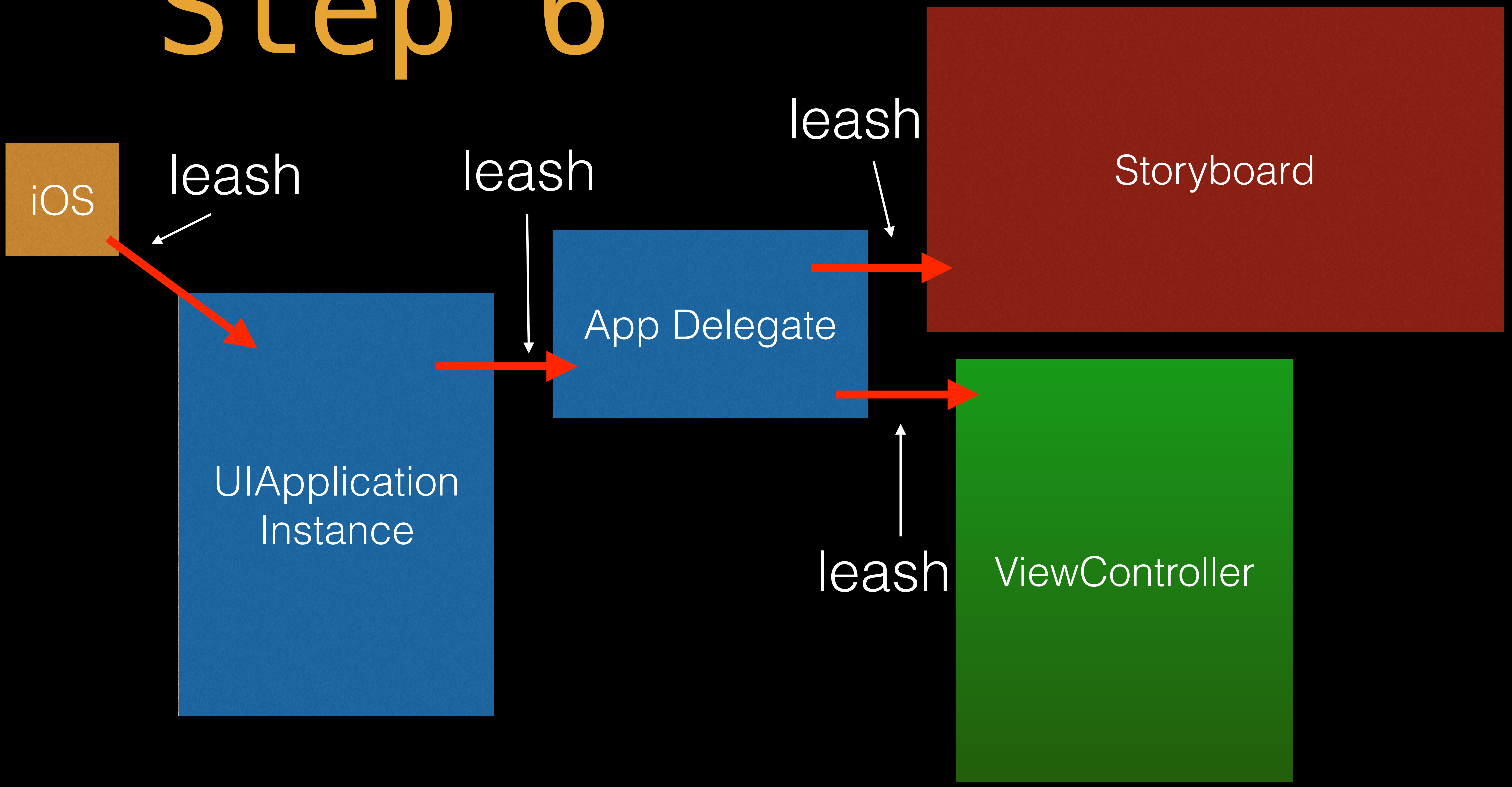
Moral of the story

- Moral of the story: An app is just a bunch of objects.
- Objects are just instances of classes.
- To build an app, design your own custom classes, use Apple’s classes, and maybe even use 3rd party classes.
- You saw the word “own” many times in these slides. This idea of ownership is related to memory management. Understanding how a language implements memory management is critical to mastering it.

Memory Metaphor

- A great metaphor to help understand how memory management works is a **dog on a leash**.
- Think of your objects as dogs.
- An owner of a dog keeps a leash on its dog because he/she wants the dog to stick around.
- As soon as a dog is unleashed, the poor thing gets destroyed (de-allocated)

Step 6



iOS owns the UIApplication Instance

UIApplication Instance owns App Delegate

App Delegate owns storyboard AND the initial View Controller

Ownership

- When you have a leash on an object, you **own** the object. Pwnage!
- This idea of **ownership** is the fundamental concept iOS memory management is built on. It is used in many other languages as well.
- Critical question: How do I become somethings owner?
  —> Use variables or properties

Strong Properties

- By default, a property is a “strong reference” (it’s a leash!)
- When you assign an object to a variable or attribute, it “owns” the object and keeps it alive.
- As long as an object has >=1 owners, it is kept alive by the system.

ARC

- All of this ownership stuff is happening in the background for you.
- This is all thanks to a system called ARC, or Automatic Reference Counting.
- Prior to iOS 5, you had to manually call retain and release on all of your objects to ensure proper memory management. Now ARC inserts this code for you at compile time! Hooray!

Demo 2.1:
ARC & Strong & Weak References

iOS Design Patterns

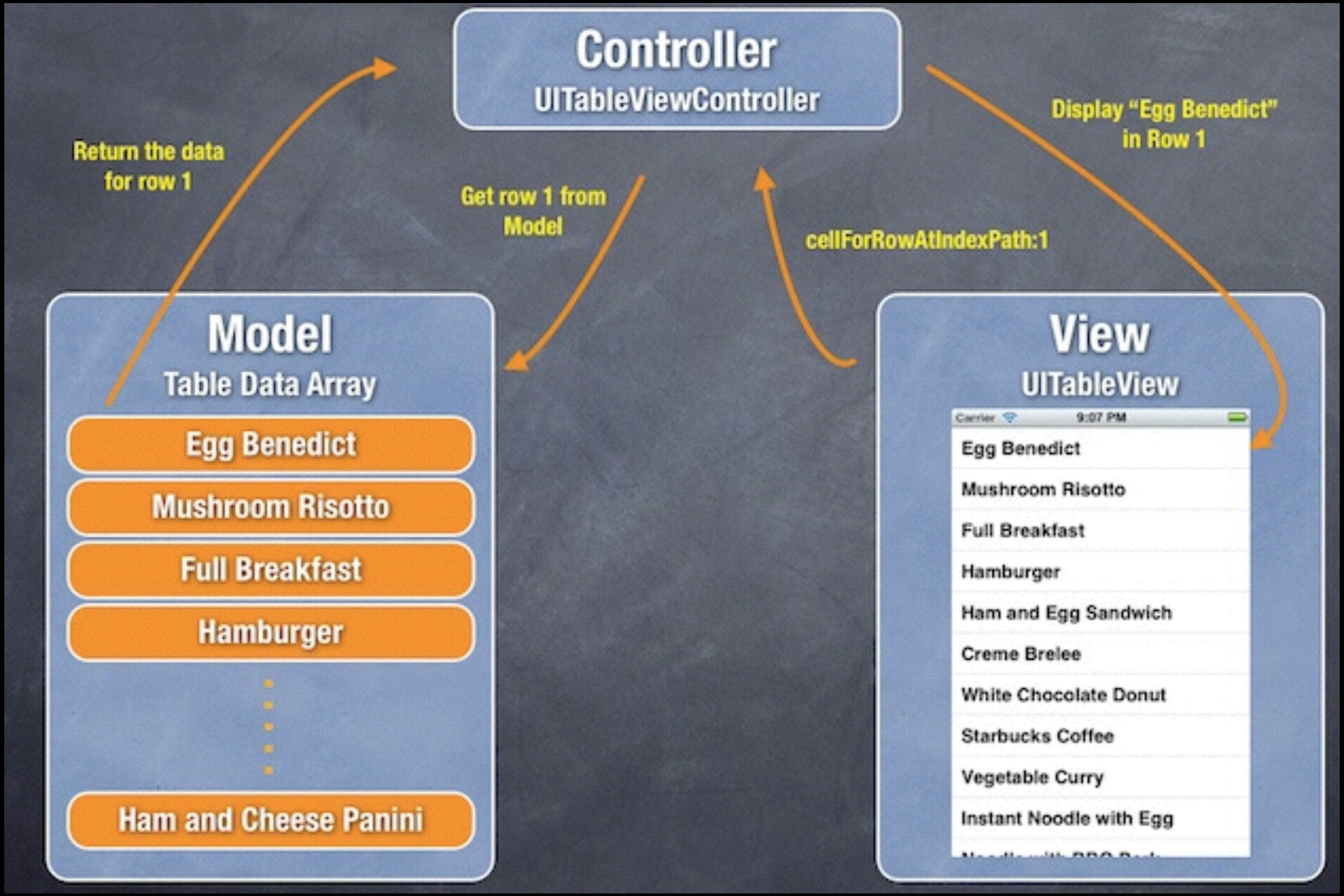
- “A software design pattern is a general reusable solution to commonly occurring problems within a given context in software design”
- No matter what kind of app it is, there are a few fundamental design pattern techniques that all apps use.

iOS Design Patterns

- The most important design patterns to know for iOS development:
    - **Model View Controller (MVC) governs the overall structure of your app. This is design pattern is championed by Apple (and other companies). We will focus on it today.**
    - *Delegation* facilitates transfer of data and responsibilities from one object to another.
    - *Target-Action* translates user interactions into code your app can execute.
    - *Closures/Blocks*: Use these for callbacks and asynchronous code.

Design Patterns Are Cool

- Benefit of universal use of design patterns is commonality between all apps besides just the language they are programmed in.
- If someone shows you their app's source code, you can ask things like "What kind of model classes are you using?" or "How many View Controllers do you have?" and sound like a guru.



MVC (Model–View–Controller)

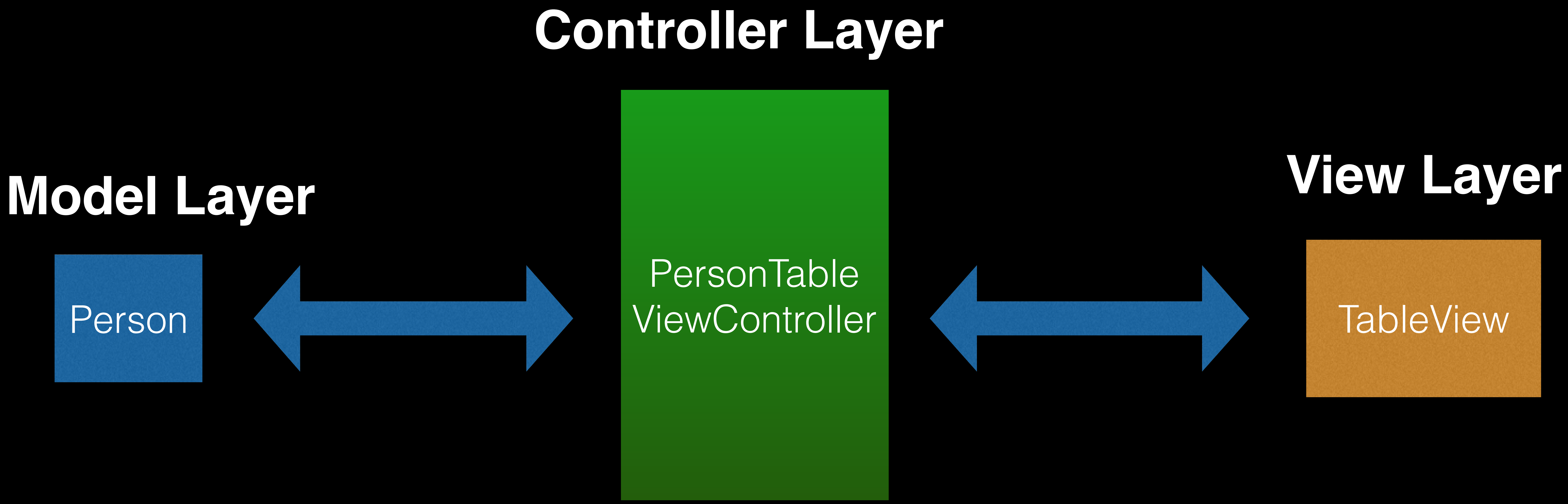
MVC Facts

- Introduced in the 70’s with the Smalltalk programming language.
- Didn’t become a popular concept until the late 80’s
- MVC is very popular with web applications. It’s not just for mobile or desktop.

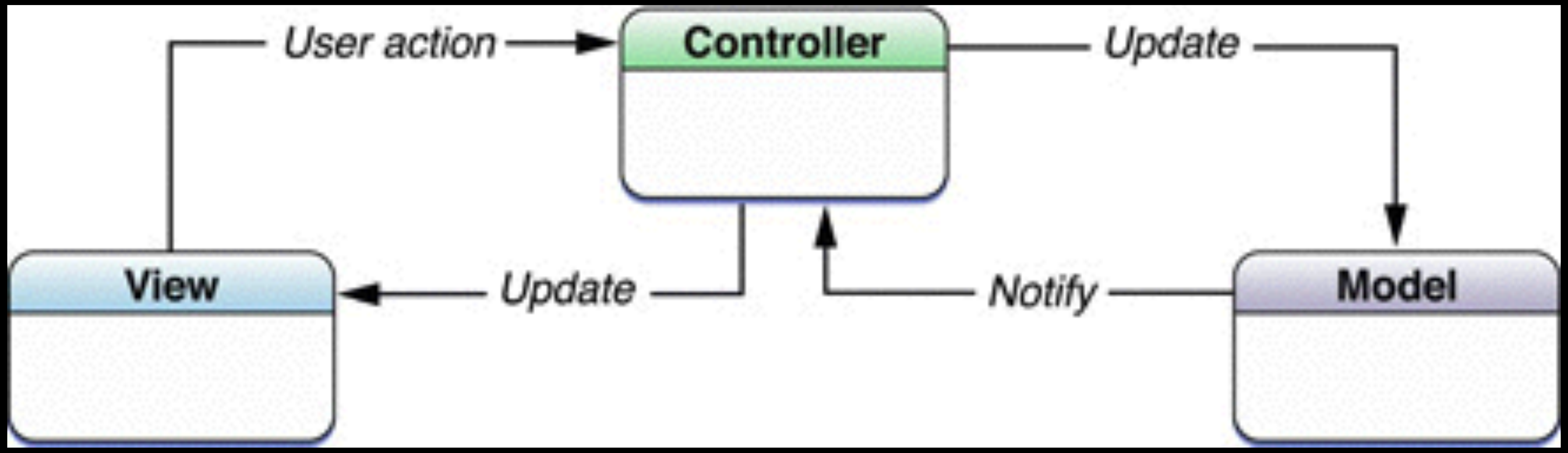
What is MVC?

- MVC is the separation of **M**odel, **V**iew, and **C**ontroller.
- It is a separation of concerns for your code. Being able to separate out these components makes your code easier to read, re-use, test, think about, and discuss.
- But MVC isn't just about **separation**, we will see its also about **communication**.
- Every object in your app is assigned to one of three roles: model, view, or controller.
- The **Model layer** is the data of your app, the **View layer** is anything the user sees and interacts with, and the **Controller layer** mediates between the two.

MVC and Communication



MVC or MCV LOL?



Some people joke its more like MCV, because the controller is the middle man so the C should go in the middle*

*Classic programming joke

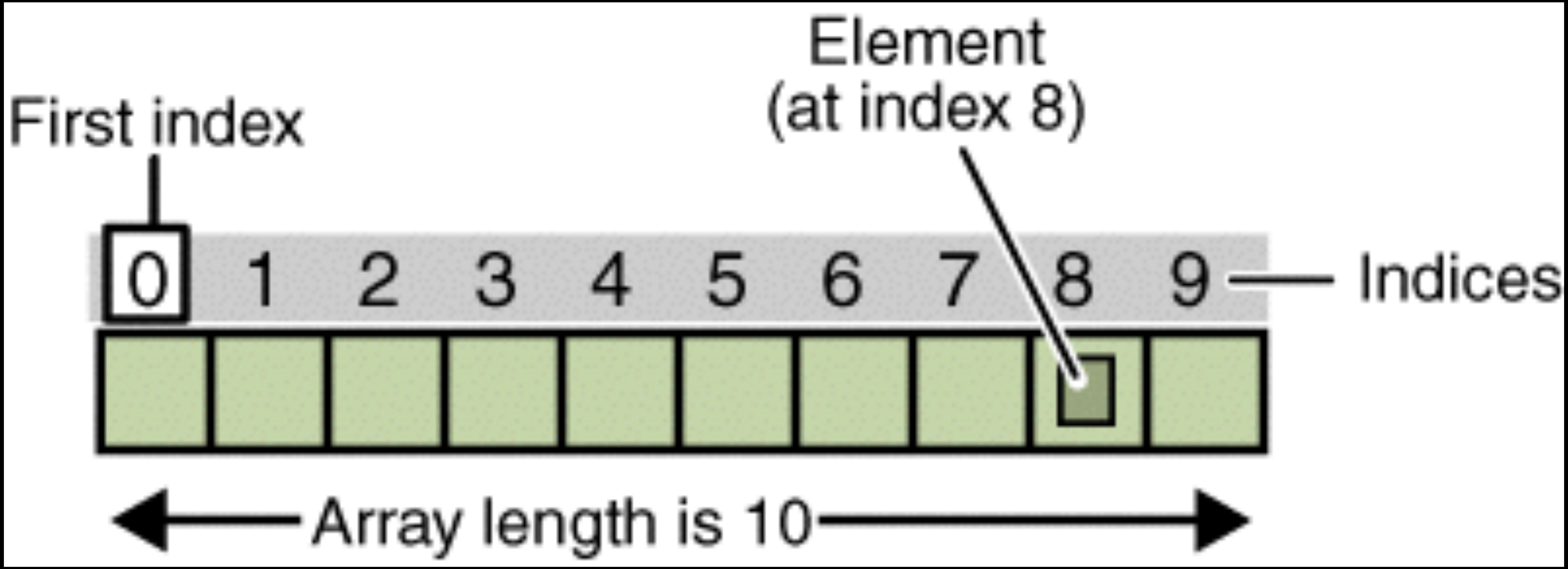
Model Layer

- Model objects encapsulate data and logic that are going to be used by your application.
- The Twitter App has a Tweet model class, a User model class, a Favorite model class, etc.

Model option: Use arrays

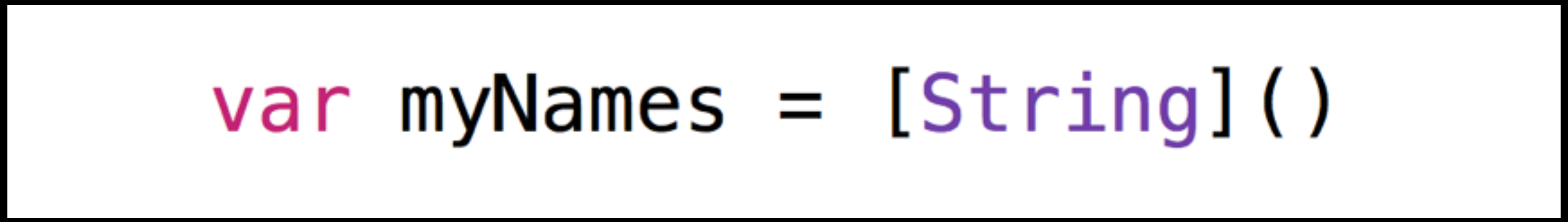
- Arrays are one way to store data in your **model**
- Arrays are very important in practically *any* programming language
- Arrays are used in virtually every iOS app ever made!
- You typically use arrays when you have a collection of similar objects or data and want to perform similar operations on them.
- Arrays are considered a “collection type”
  (Aside: **sets** are also a collection type in Swift.)
- Arrays are ordered <— important

Arrays


First index
Element
(at index 8)
0
1
2
3
4
5
6
7
8
9
Indices
Array length is 10

Creating an Array in Swift

- An array is considered a type, and the way to signify an array type in Xcode is just []
- But thats not quite complete, because inside the brackets you need to also state the type of objects you are going to be putting inside the array.
- So [String] is the type of an array that holds Strings. and [UIImage] is the type of an array that holds images.
- To actually instantiate an array, you use () after the closing square bracket to create the array:



```swift
var myNames = [String]()
```

Adding things to an array

- There are two ways an object can get inside an array in Swift:
    - Arrays have a method called append, which takes in one parameter, the object you want to add to the end of the array:

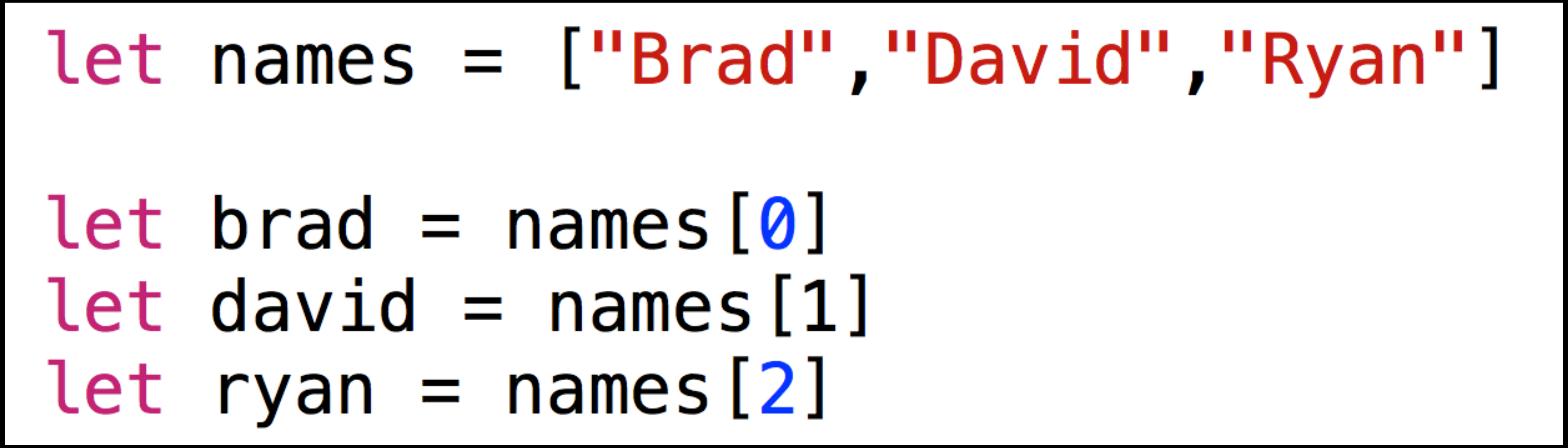
```swift
var myNames = [String]()
myNames.append("Brad")
```

- When you initial create an array, you can use a special shorthand syntax where you place all the objects in the brackets of the array you are creating:

```swift
let names = ["Brad","David","Ryan"]
```

Retrieving objects from an array

- Retrieving objects from an array uses a special syntax that also involves []
- You retrieve objects from arrays by their index number.
- Remember the index starts at 0, not 1! (forgetting this fact is pretty common, and leads to the classic 'off by one' error)



```
let names = ["Brad","David","Ryan"]

let brad = names[0]
let david = names[1]
let ryan = names[2]
```

Demo 2.2: Arrays

View Layer

- A **view** object is visible to the user, and can provide interaction.
- Can enables display and editing of app’s data.
- Communication between **view** and **model** layers is made possible by…..

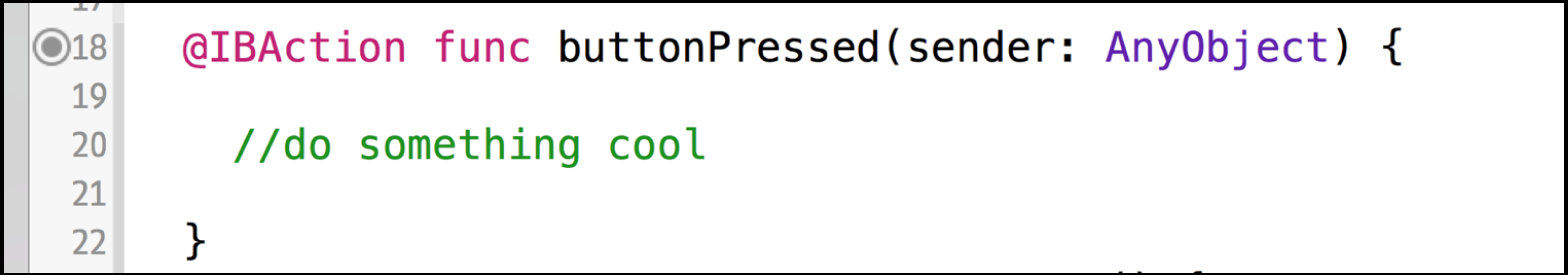
Controller Layer

- Act as intermediary between **model** layer and **view** layer.
- The most common form of a controller in iOS is a view controller.
- At first, your view controllers will contain a lot of code. Eventually you should strive to make them lighter so its easier to understand what they are doing at a glance.

C: “glues” M and V: IBOutlets & IBActions

IBAction

- InterfaceBuilderAction, or IBAction for short, are special methods triggered by a user interface object that was laid on storyboard.

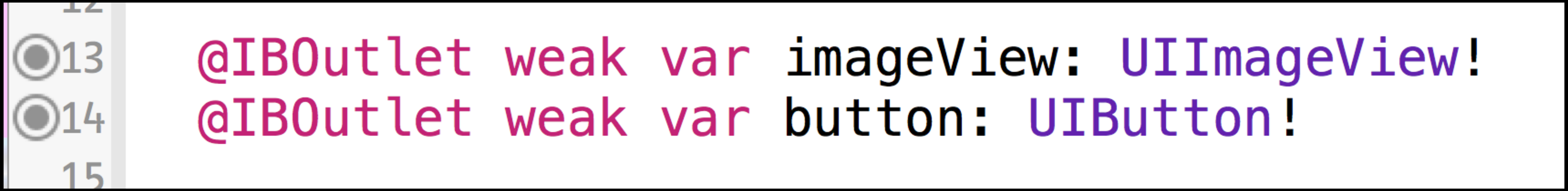


```
@IBAction func buttonPressed(sender: AnyObject) {

  //do something cool

}
```

- Multiple interface objects can be hooked up to the same IBAction method
- Whenever an IBAction has a parameter, like you see above, it is simply the interface object that triggered this action. Best practice is to name this parameter ‘sender’.
- If you have multiple buttons hooked up to the same IBAction, you can inspect sender to see which button actually triggered the action (could check the tag, or class of sender)

IBOutlet

- InterfaceBuilderOutlet, or IBOutlet for short, is a special type of property that creates a link between your code and your interface objects on your storyboard.
- So if you drag a UIImageView or UIButton onto your storyboard, you can create an outlet for these interface objects:



```
@IBOutlet weak var imageView: UIImageView!
@IBOutlet weak var button: UIButton!
```

- This allows you to access them in code just like any other regular property.
- The weak keyword here means this property is not a strong reference, its not a leash. we don't need the leash here because these object's super views already have ownership.

Demo 2.3:
Castle MVC
(Xcode UI, live, everyone)

View Controllers

- View controllers: ‘virtual link’ between app’s **data** and visual **appearance** of your app.
- Whenever app displays a UI, displayed content is managed by either one view controller or a group of them working together.
- Because of this, view controllers provide a foundational framework for building apps.

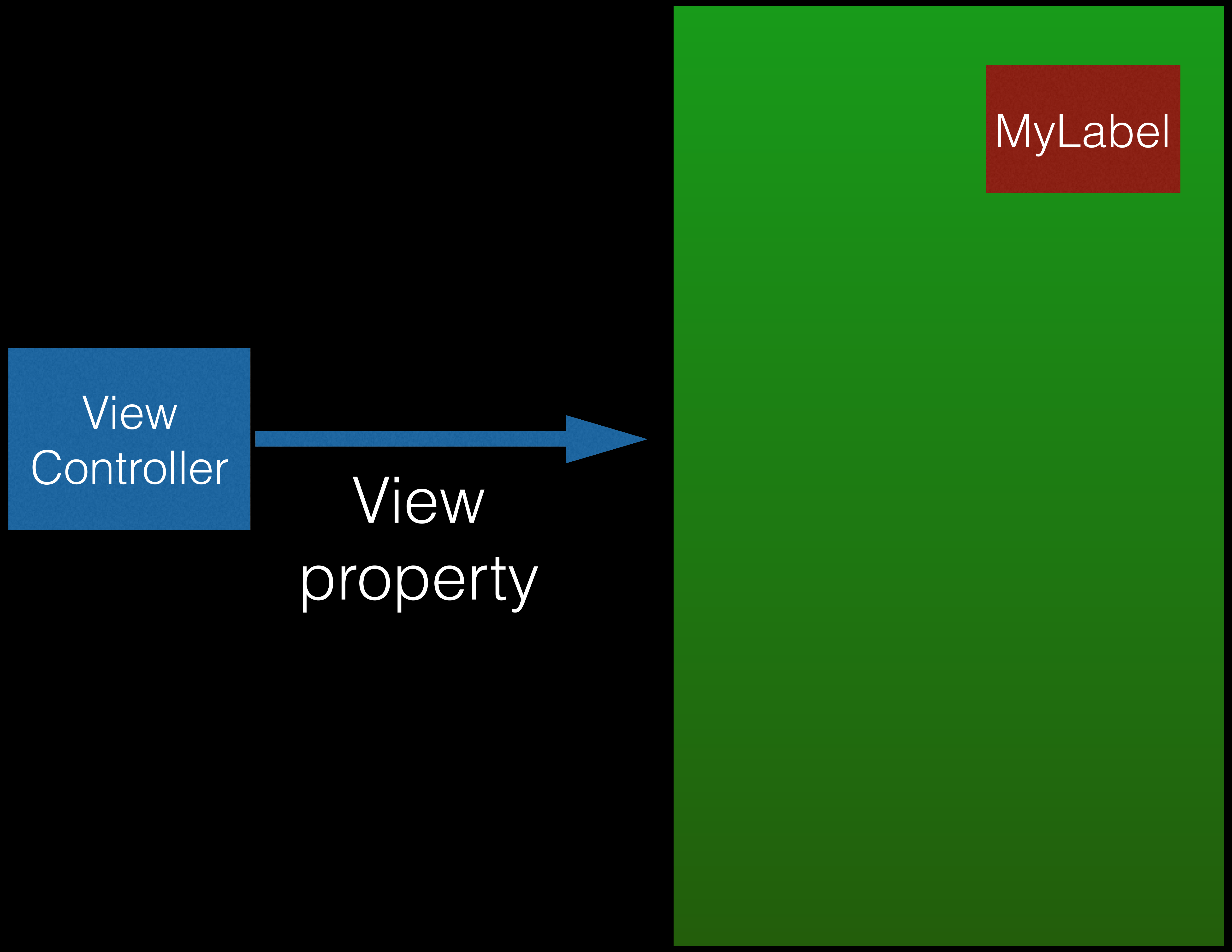
`UIViewController`

- Apple provides many built-in view controllers that are already set up to help you achieve many common interfaces (“widgets”)
- You will also write your own custom view controller classes to show custom data to the user.
- Apple provides a view controller base class. The name of the class is **UIViewController**. You will *subclass* UIViewController quite often.

View Controller and its View

- A View Controller has a UIView property thats simply called ‘view’.
- UIView is a class that defines a rectangular area on the screen and handles the rendering of any content in its area and any interactions with that content.
- If you have a UIView called ‘MyLabel’ and you then add ‘MyLabel’ to the View Controller view property, MyLabel is considered a subView of the View Controller’s view property. Or, the View Controller’s view property is now the super view of MyLabel.

View Controller and its View



You will often hear/read a view controller's view referred to as the root view

Demo 2.4:
Sparta panel
(Xcode UI, live, everyone)

View Controller Life Cycle

- A View Controller’s view property is so important, its entire ‘life cycle’ is designed around when the view is first loaded, and whenever it is shown or disappears.
- When we talk about the ‘life cycle’ of a view controller, we are talking about when its ‘life cycle’ methods are called.

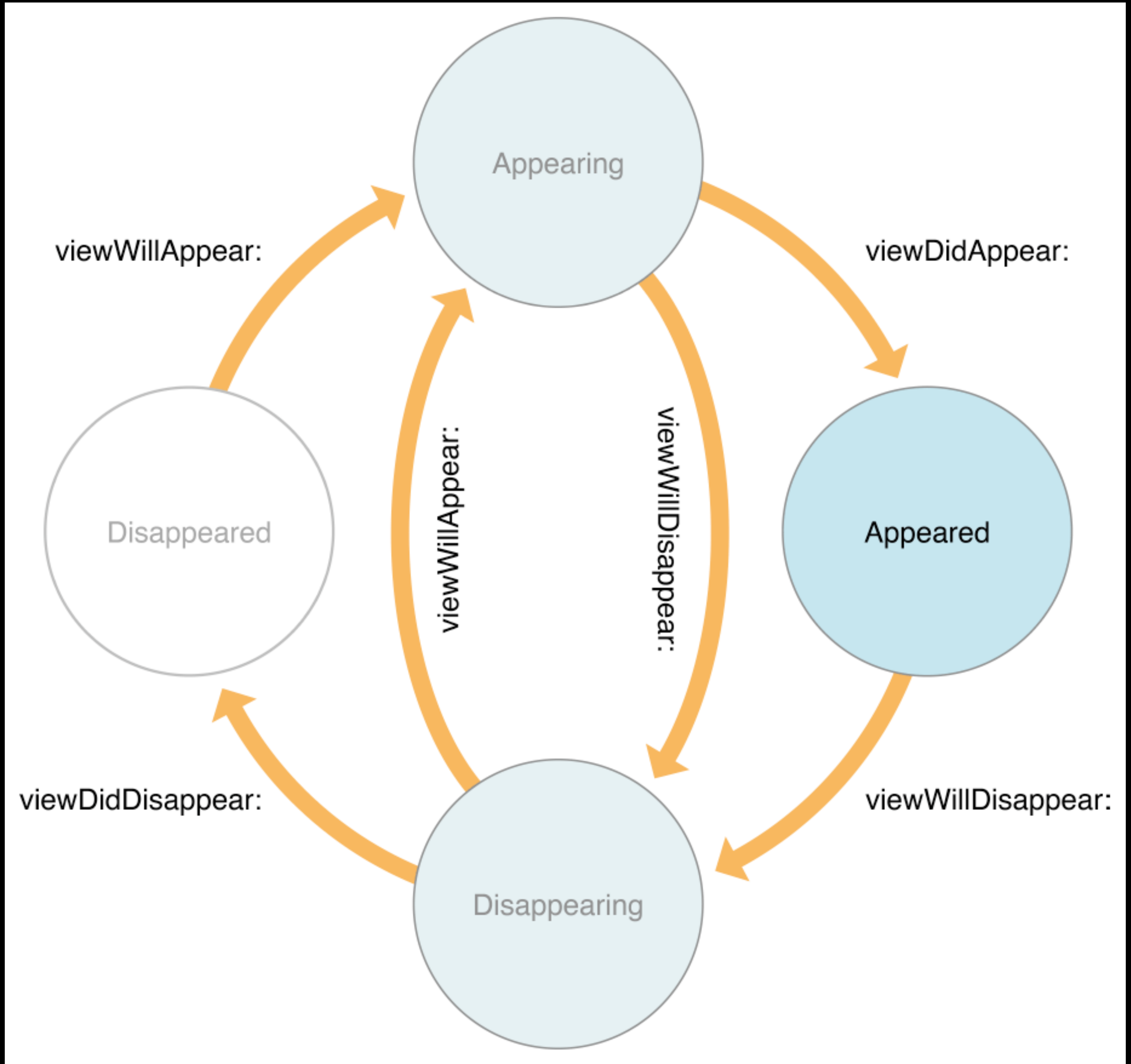
Life Cycle Methods

- Whenever some part of your app asks the view controller for it view object and that object is not currently in memory (alive and owned by something), two methods are called on the view controller:
  1. **loadView** - if the VC is coming from storyboard, it loads the interface you laid out in storyboard and **you must NOT use this method**. If no storyboard, you should programmatically create your interface here.
  2. **viewDidLoad** - this lets your VC perform any additional load-time tasks not related to creating the interface.

More Life Cycle methods

- In addition to the load-time methods, there are also life cycle methods that are called when the View Controller’s view is about to be shown or removed:
  - **viewWillAppear** - is called right before the view appears on screen
  - **viewDidAppear** - is called right after the view appears on screen
  - **viewWillDisappear** - is called right before the view is about to be removed
  - **viewDidDisappear** - is called right after the view has been removed

View Life Cycle



Demo 2.6:
View Life Cycle
(Xcode UI, live, everyone)

Planning Your View Controllers

- View Controllers are so important to iOS development, when you are designing and writing your app, you are often thinking “How many View Controller does this feature need?” or “Which View Controllers need this functionality?”
- Because of this, it is nice to have a structured way to plan out a specific View Controller….

Planning Your View Controllers in 6 steps

1. Are you using a storyboard to implement this VC?
2. When is it instantiated?
3. What data does it show?
4. What tasks does it perform?
5. How is its view displayed onscreen?
6. How does it collaborate with other view controllers?

Introduction to git

- Specify the text editor you'll use for commit messages

```
$ git config --global core.editor emacs
```

- Add workspace file to local repo:

```
$ git add [file] # "Staged" status; marked for repo,
  but not in yet
$ git commit # Copy file from workspace to local repo
  and update metadata
```

- Synchronize local repo with remote repo

```
$ git push
```

Introduction to git

- Remove file from remote repo:

```
$ git rm [file] # Deletes local file
$ git add -A # Removes file from local
  repo
$ git push # Removes file from remote
  repo
```

Introduction to git

- Edit `gitTest/student_names.txt` by adding your name to the file (anywhere in the file)

```
$ git commit
$ git status
```

- Push local (committed) changes to the same remote repo you cloned. On your **first** “git push”, git will prompt you for your github email & password. After first push, git should remember your login info.

```
$ git push
```