App Structure and UIKit

March 17, ‘16

Working with iOS SDK

- Most iOS SDK are still written in Objective-C and C.
  Also for SDKs of OS X, tvOS and watchOS.

- Both Swift and Objective-C are able to work with iOS SDK.
  This relationship is like the one between JavaScript and jQuery or the one between Python and Django.

- Objective-C classes are bridged as Swift classes. And C structs are bridged as Swift structs.
  Most types in iOS SDK are actually classes. `@objc` may be used necessary.

Working with iOS SDK

```
@objc protocol SomeProtocol: NSObjectProtocol {
    func someRequired()
    optional func someOptional()
}
```

- Types conforming to a Swift protocol must implement all methods declared in the protocol. *(Like Java's Interface)*
- Objective-C protocols include optional methods, which are not required to be implemented by conforming types.

Working with iOS SDK

- Numbers *(Integers, Floats, and Boolean)*, String, Array, Set, and Dictionary are bridged between 2 languages.
  As NSNumber, NSString, NSArray, NSSet, and NSDictionary.

- Objective-C doesn't use *namespace*. Instead, it uses prefixes before class names.
  "NS" means for the "NeXTSTEP" where the OS X is derived from. "UI" means UIKit (the iOS SDK).

- Check `ObjC-Bridge.playground` in the *Swift-Introduction* git repo.

Launch iOS app

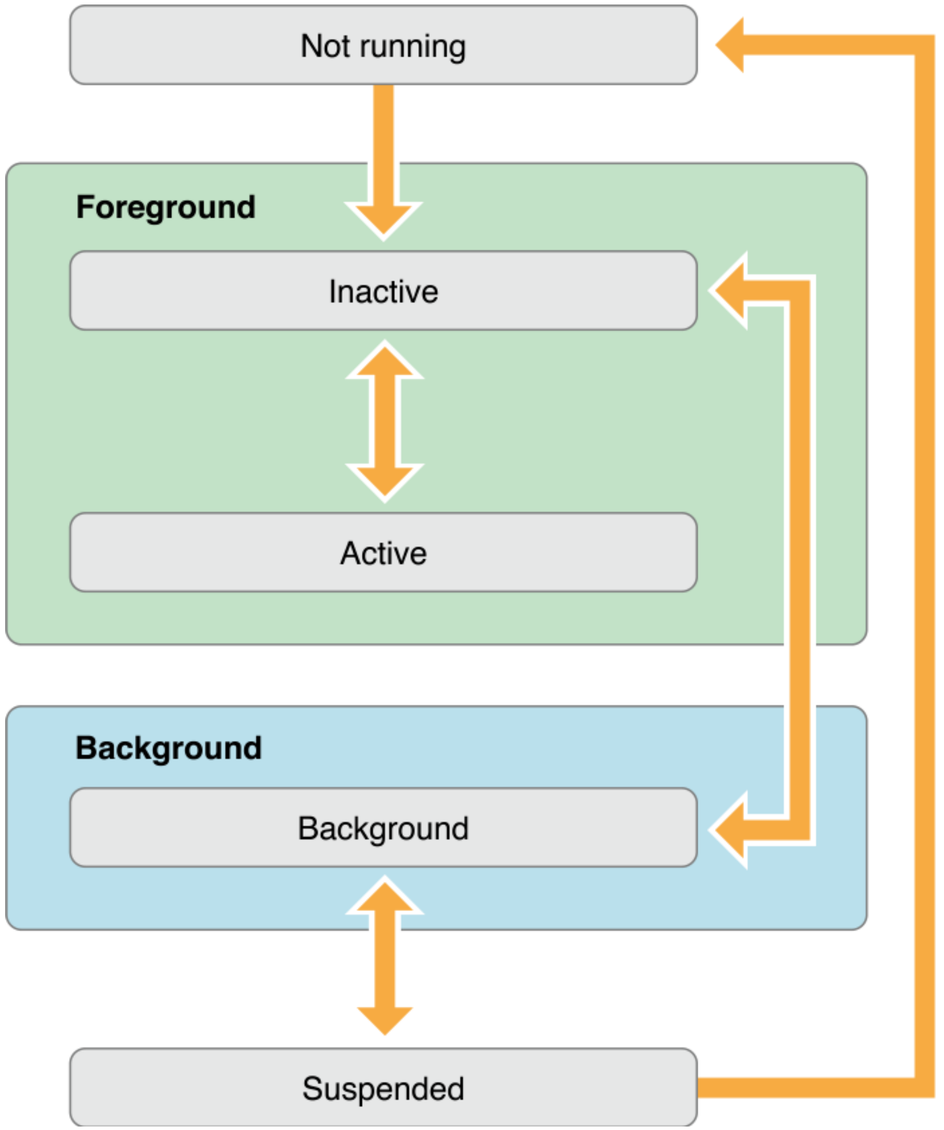
Launch iOS app

- The runtime would find for the class which annotated with the `@UIApplicationMain` attribute. This class would be the main entry point of your app.
  It's usually annotated on `AppDelegate` class in `AppDelegate.swift`

- The annotated class must implement `UIApplicationDelegate` protocol. And it *usually* inherits from `UIResponder` class.
  UIApplicationDelegate handles events related to app's lifecycle. UIResponder responds to global events of the app.

App Lifecycle

- Implement methods of `UIApplicationDelegate` to handle app lifecycle events.
- Observe `NSNotification` emitted by `UIApplication` to handle app lifecycle events.
  Notification pattern would be mentioned in future classes.

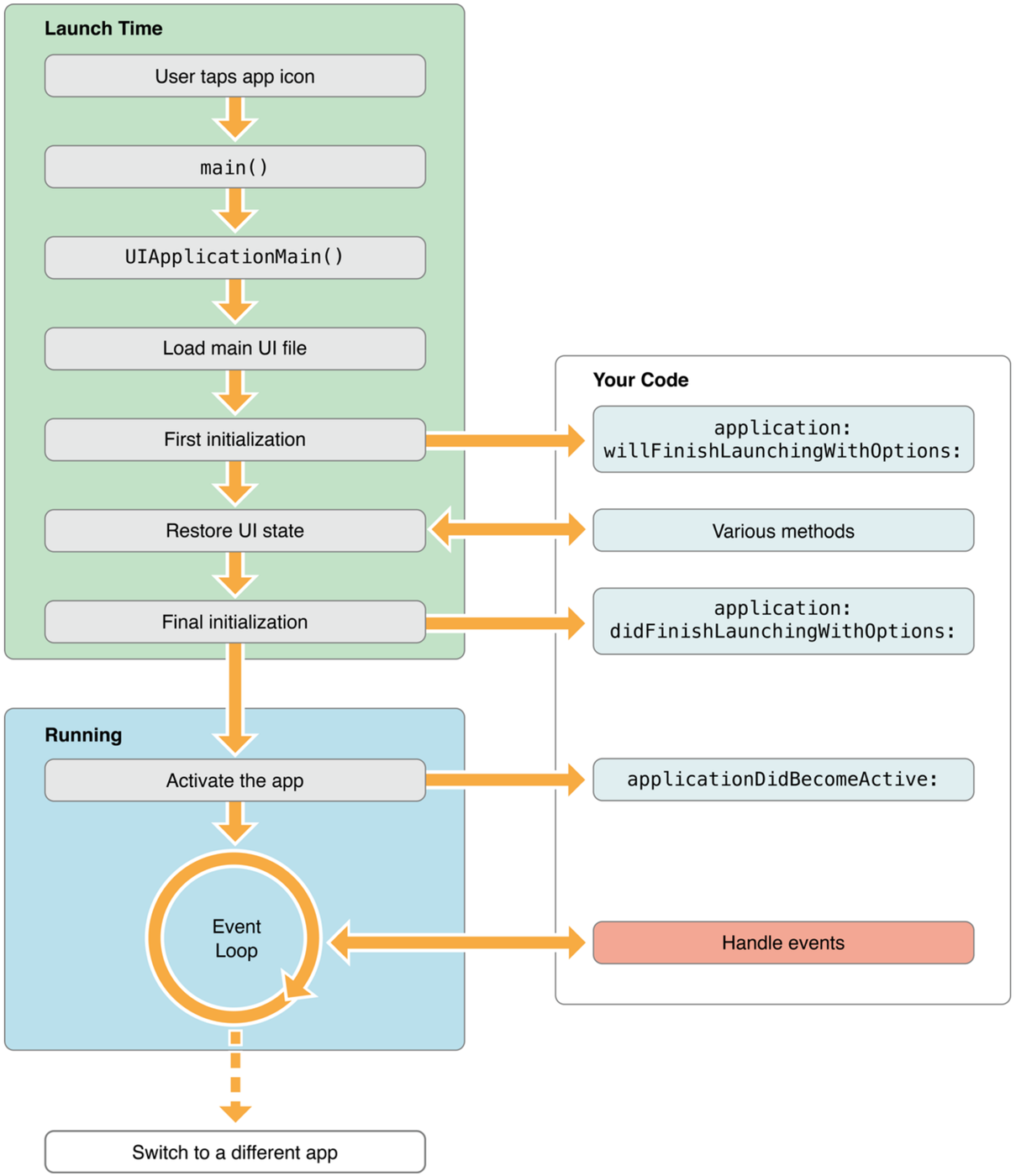


App Lifecycle - Event Handling

- Implementing method in `UIApplicationDelegate` is like a ground control center of your app elements, which is suited for app-level resources.
- Observing `NSNotification` emitted by `UIApplication` is distributed in each elements and better for local resources.

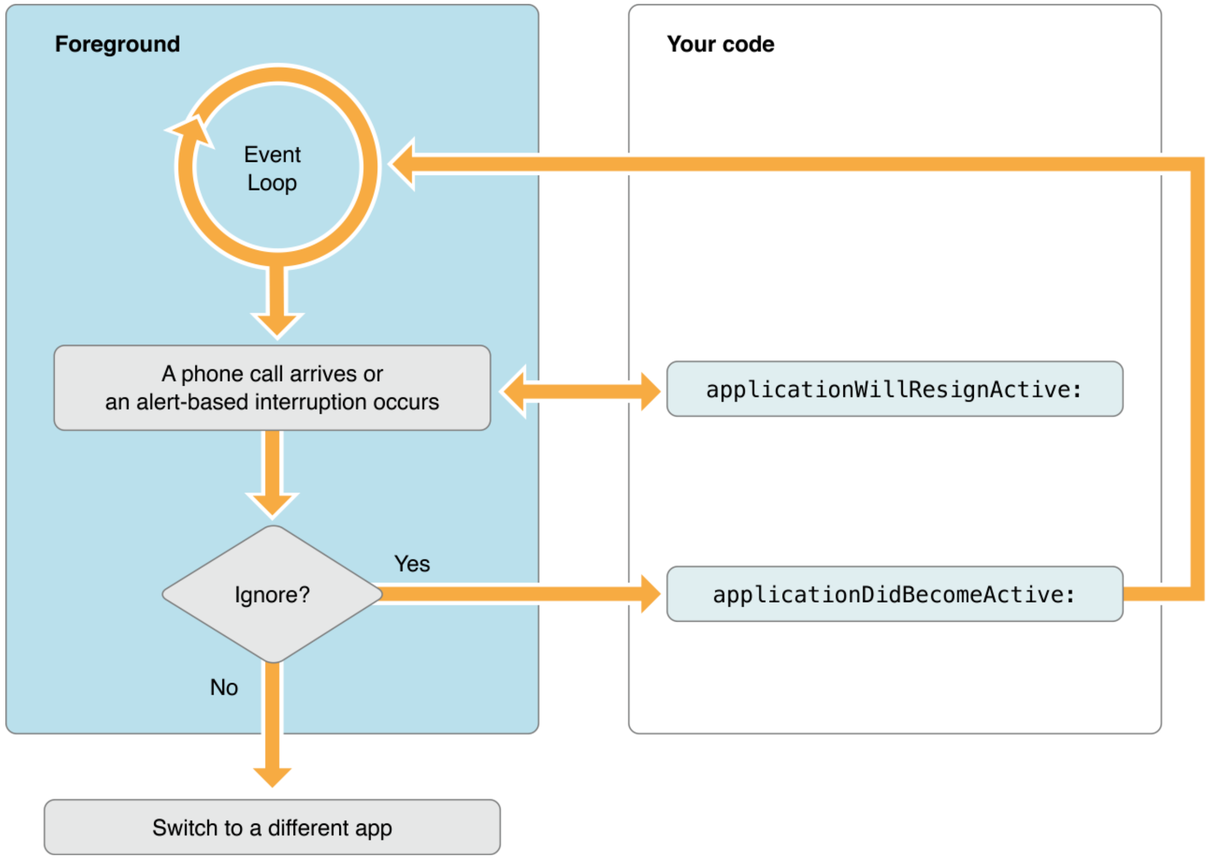
App Lifecycle

- We usually prepare app level resources when “finish initialization” state.
- The event loop dispatches events to the responder chain.
  Your UI elements are part of the responder chain.



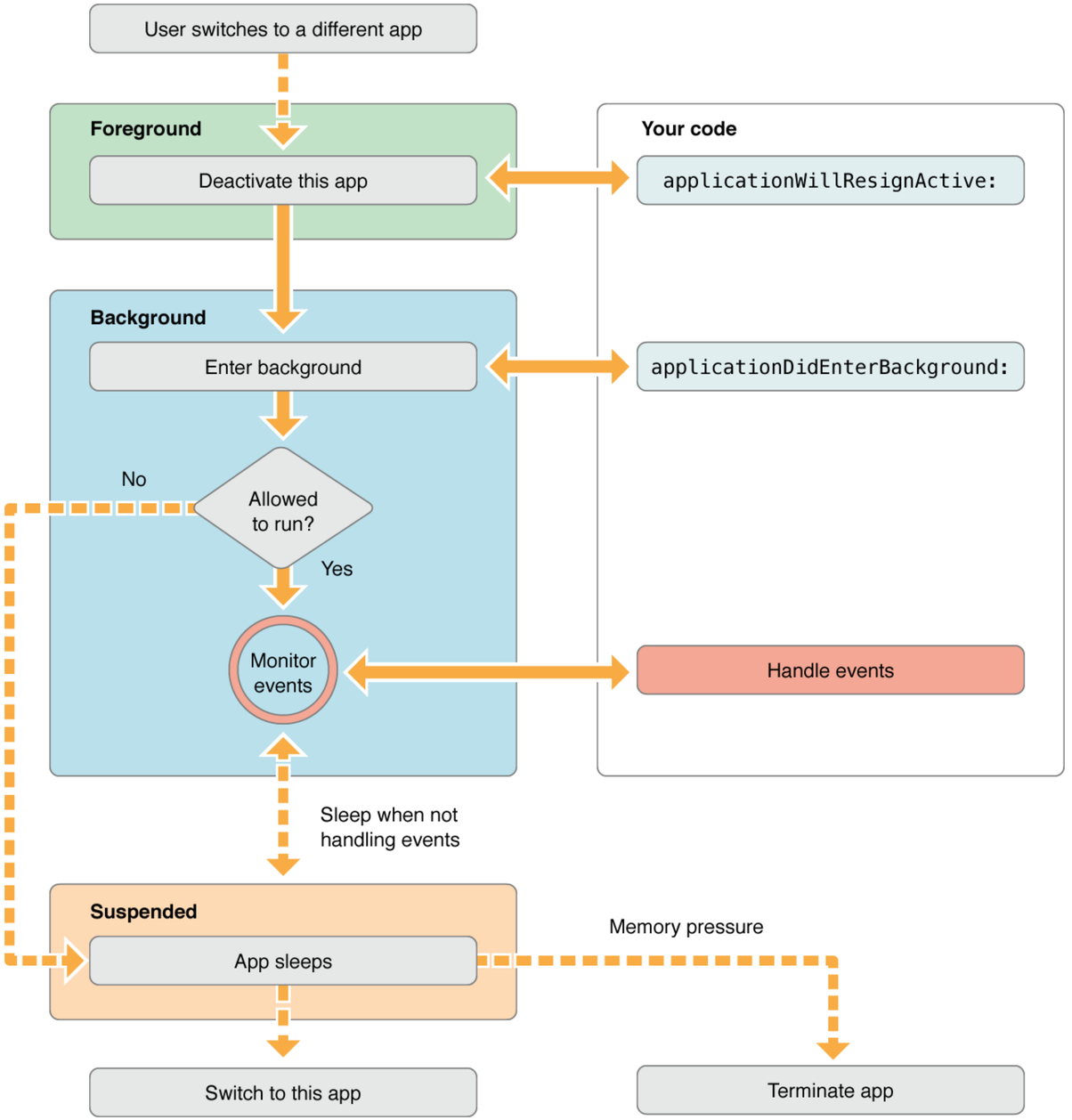
App Lifecycle

- Interruptions result in a temporary loss of control by your app.
  Including user double-clicks the home button



App Lifecycle

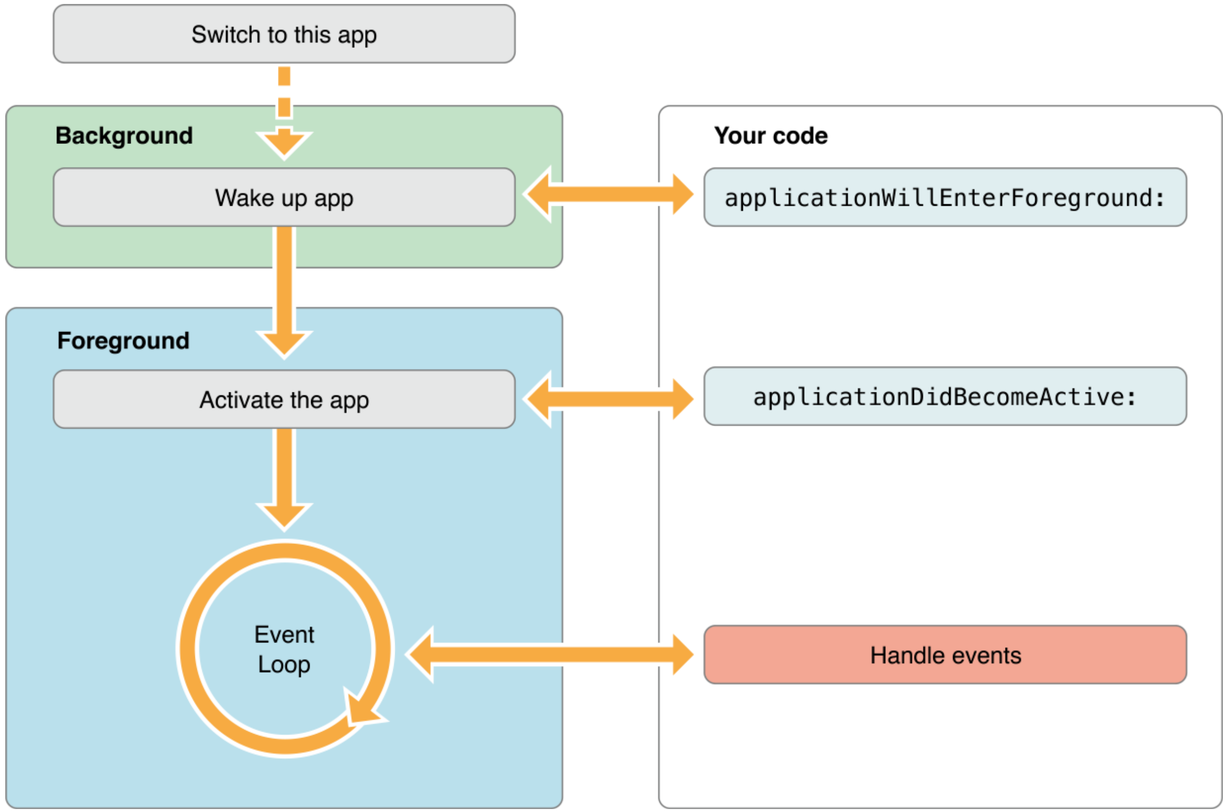
- Background modes are advanced topics.
  Check Apple's references
- Save data and app states when the app is being deactivated.



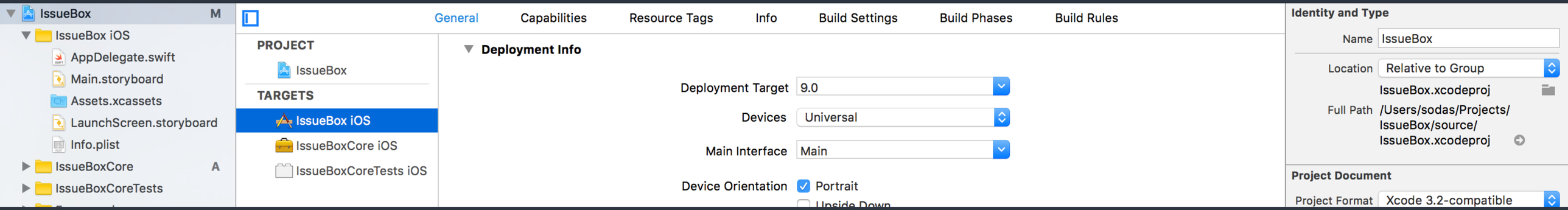
Launch iOS app > App Lifecycle

App Lifecycle

- When coming back to the foreground, remember to check system changes.
  Like locale change, file system change, and etc.



Launch iOS app - Load Storyboard



- By default, the runtime would find the Main Interface settings of your project to load the Storyboard and instantiate *the initial view controller* as app's root view controller.

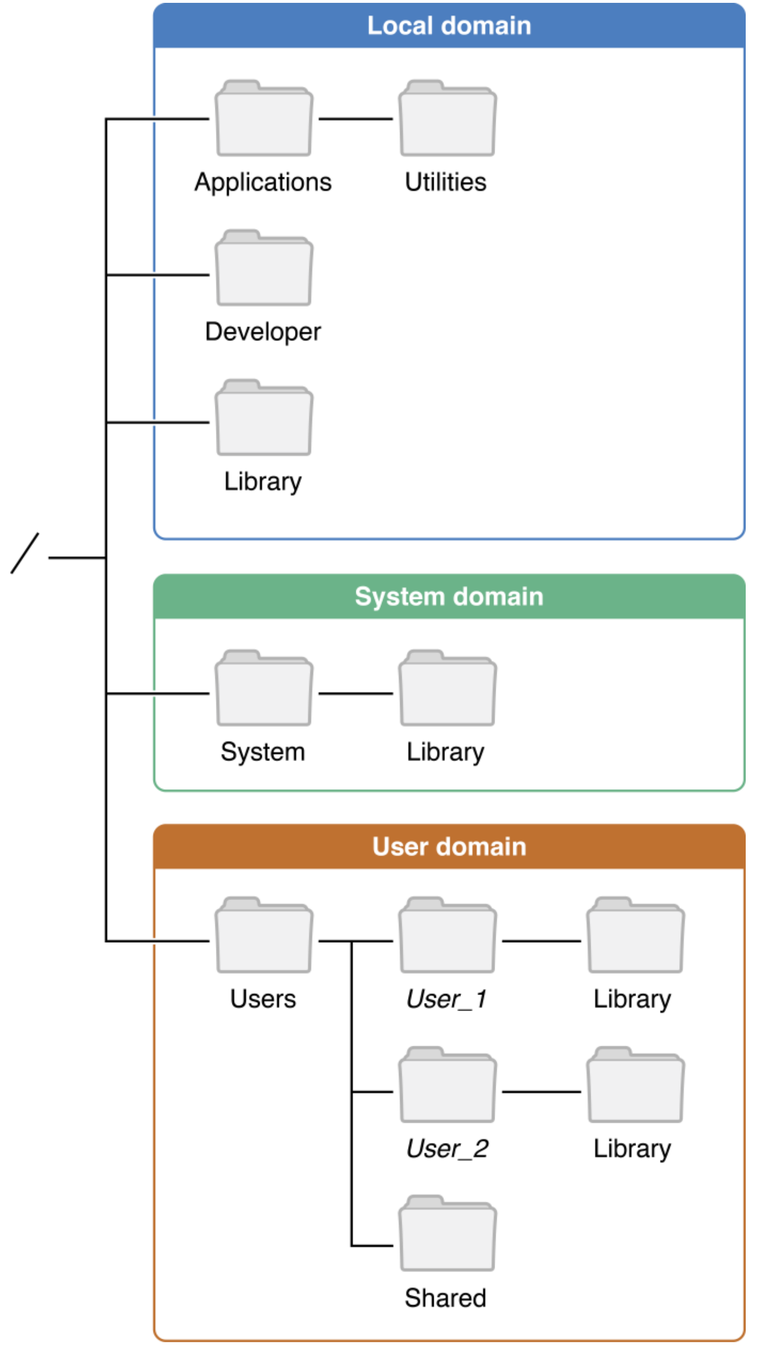
  This setting is actually stored in "Info.plist" of your app

App Structure and File System

File System in OS X

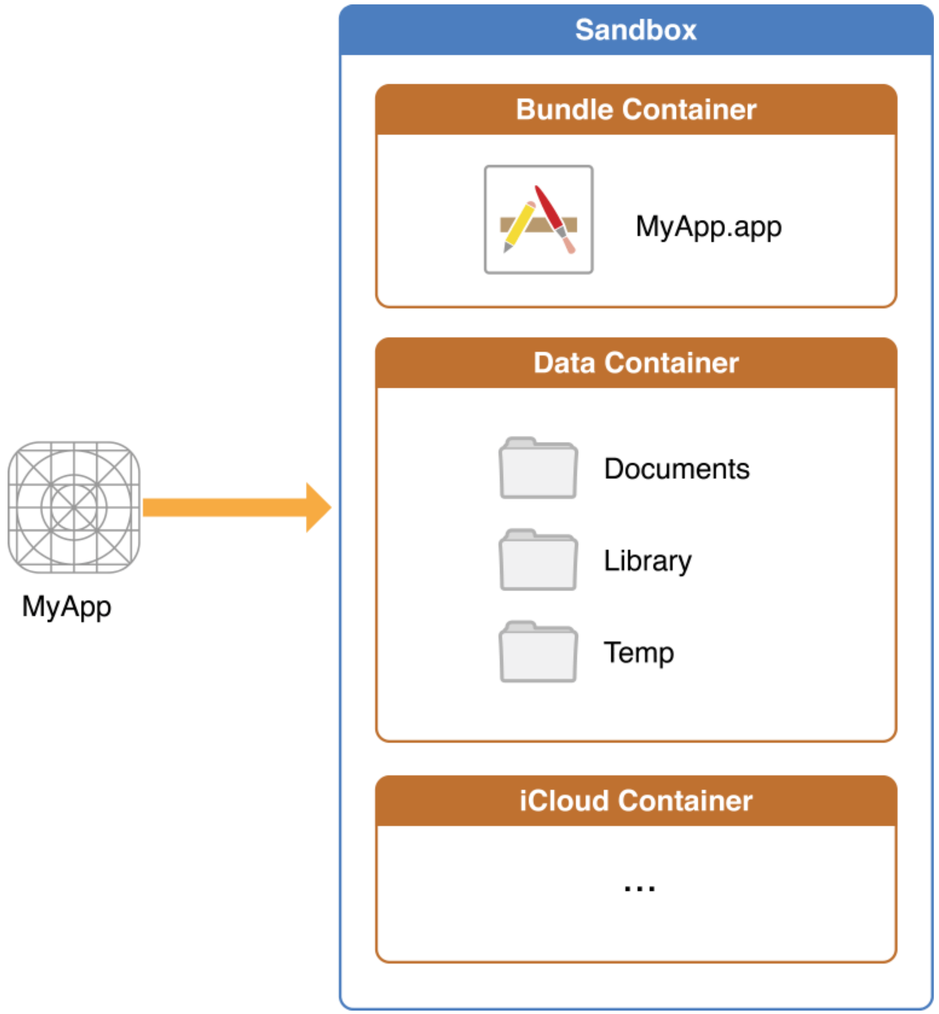
- the OS X is a Unix-based system.
  Derived from BSD and NeXTSTEP. The root of FS is “/”. But Apple changed FS naming convention of UNIX-like system.

- Applications may be able to walk through the whole file system.
  Applications may be also sandboxed which are only able to access its own directory.



Sandboxed iOS app

- Each app has its own sandbox, like an isolated island.
  Each apps are only able to access its own files and directories in its sandbox.
- Use API call to get paths.
- In the sandbox, there are only User Domain folders.
  No folders in local domain and system domain are available to iOS apps.



- `<Application_Home>/AppName.app`
  This is the bundle directory containing the app itself. Readonly.

- `<Application_Home>/Documents/`
  Store user documents and data files, which are not re-generable.
  It will be backed up.

- `<Application_Home>/tmp/`
  Temporary files that do not need to persist between launches of your app.
  It won't be backed up.

- `<Application_Home>/Library/`
  Top-level directory for files that are not user data files (regenerable by app).
  Content are usually grouped by your bundle identifier.
  It will be backed up, except the “Caches” folder.

- `<Application_Home>/Library/Caches`
  Used to store cached files. It won’t be backed up.

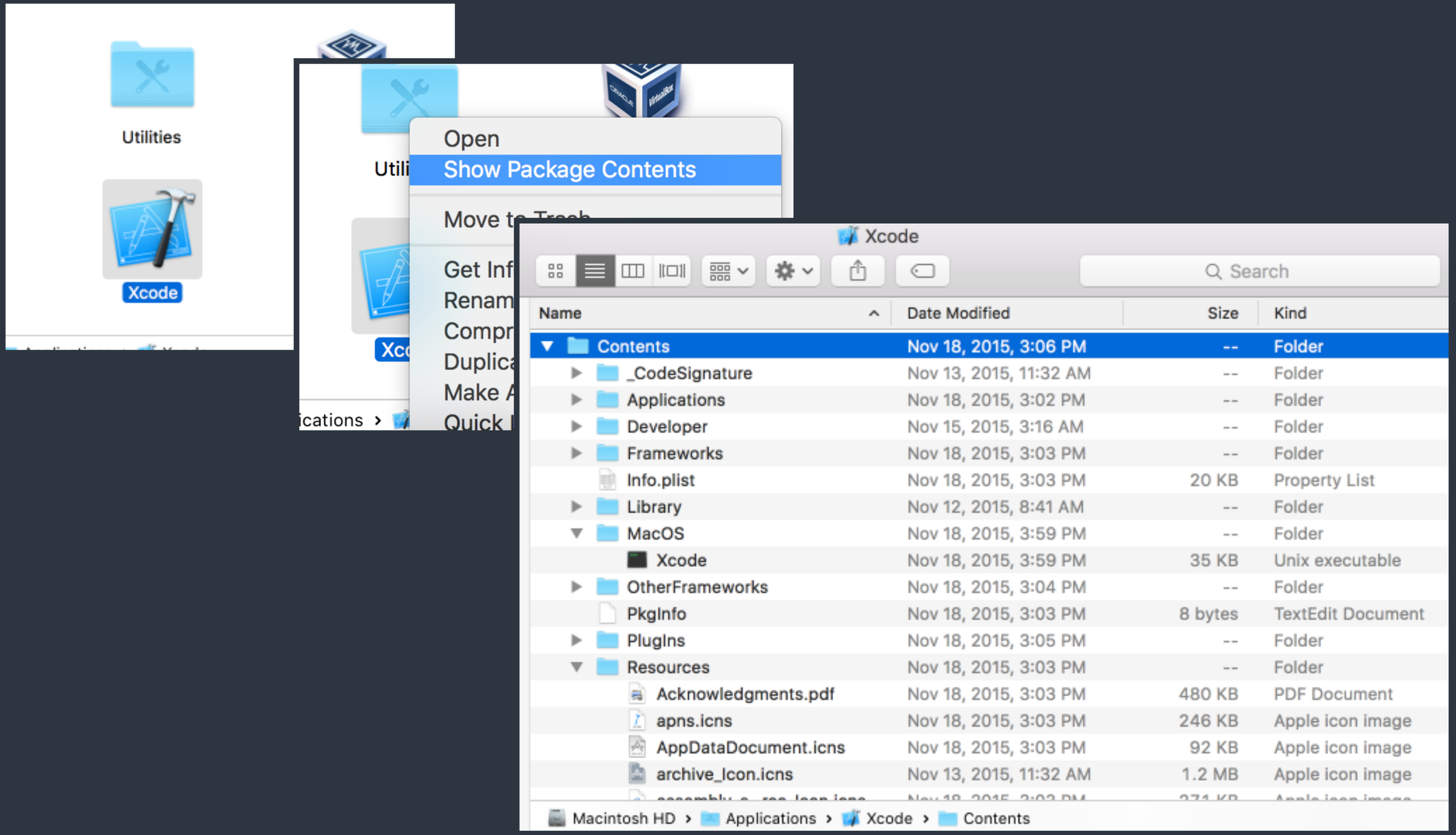
- `<Application_Home>/Library/Application Support`
  In general, this directory includes files that the app uses to run but that should remain hidden from the user.

Application Bundle

- In OS X and its derived platforms, applications are bundles.
  Bundles are a fundamental technology in OS X that are used to encapsulate code and resources.

- A package is any directory that the Finder presents to the user as if it were a single file.

- A bundle is a package with a standardized hierarchical structure that holds executable code and the resources used by that code.


Utilities
Xcode
Open
Show Package Contents
Move to Trash
Get Info
Rename
Compress
Duplicate
Make Alias
Quick Look
Xcode
Search
Name
Date Modified
Size
Kind
Contents Nov 18, 2015, 3:06 PM -- Folder
_CodeSignature Nov 13, 2015, 11:32 AM -- Folder
Applications Nov 18, 2015, 3:02 PM -- Folder
Developer Nov 15, 2015, 3:16 AM -- Folder
Frameworks Nov 18, 2015, 3:03 PM -- Folder
Info.plist Nov 18, 2015, 3:03 PM 20 KB Property List
Library Nov 12, 2015, 8:41 AM -- Folder
MacOS Nov 18, 2015, 3:59 PM -- Folder
Xcode Nov 18, 2015, 3:59 PM 35 KB Unix executable
OtherFrameworks Nov 18, 2015, 3:04 PM -- Folder
PkgInfo Nov 18, 2015, 3:03 PM 8 bytes TextEdit Document
PlugIns Nov 18, 2015, 3:05 PM -- Folder
Resources Nov 18, 2015, 3:03 PM -- Folder
Acknowledgments.pdf Nov 18, 2015, 3:03 PM 480 KB PDF Document
apns.icns Nov 18, 2015, 3:03 PM 246 KB Apple icon image
AppDataDocument.icns Nov 18, 2015, 3:03 PM 92 KB Apple icon image
archive_Icon.icns Nov 13, 2015, 11:32 AM 1.2 MB Apple icon image
Macintosh HD › Applications › Xcode › Contents

Content of iOS App Bundle

- `<AppBundle>/<AppName>`
  The main Unix executable file

- `<AppBundle>/Info.plist`
  Configuration information for the application, including app display name, identifier, and main storyboard file. The system relies on the presence of this file to identify relevant information about your app and any related files.

Content of iOS App Bundle

- `<AppBundle>/*.lproj`
  Localized resources
- `<AppBundle>/Frameworks`
  Embedded frameworks (dynamically linked components)
- `<AppBundle>/*.*`
  General resources and assets

NSBundle

```
let mainBundle = NSBundle.mainBundle()
let pathOfContentTxt: String? = mainBundle.
    pathForResource("content", ofType: "txt")
let infoDict = mainBundle.infoDictionary
```

- Use `NSBundle.mainBundle()` to access the app's bundle.
- Use `pathForResource(_:ofType:)` method to find assets and resources in a given bundle.
- Use `infoDictionary` to access the Info.plist content of a bundle.

NSFileManager

```
let fileManager = NSFileManager.defaultManager()
let documentsURLs = fileManager.URLsForDirectory(.DocumentDirectory,
    inDomains: .UserDomainMask)
let libraryURLs = fileManager.URLsForDirectory(.LibraryDirectory,
    inDomains: .UserDomainMask)
```

- Use `NSFileManager` to find paths of sandbox directories. It also provides methods for manipulating file systems.
- Path manipulation APIs are provided by `NSString`.

Common Resources in an App

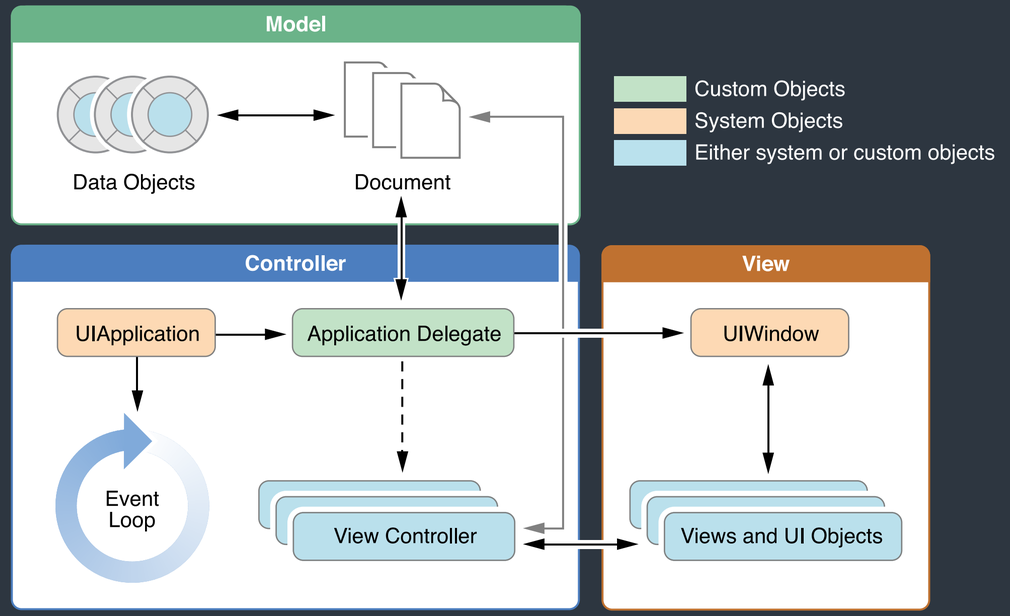
- Storyboard
- Asset catalogs
  Used to simplify management of images that are used by your app as part of its user interface.
- Launch files
  Provides a simple placeholder image that iOS displays when your app starts up.

Demo for Asset catalog and Launch Screen

View Controller and MVC Pattern

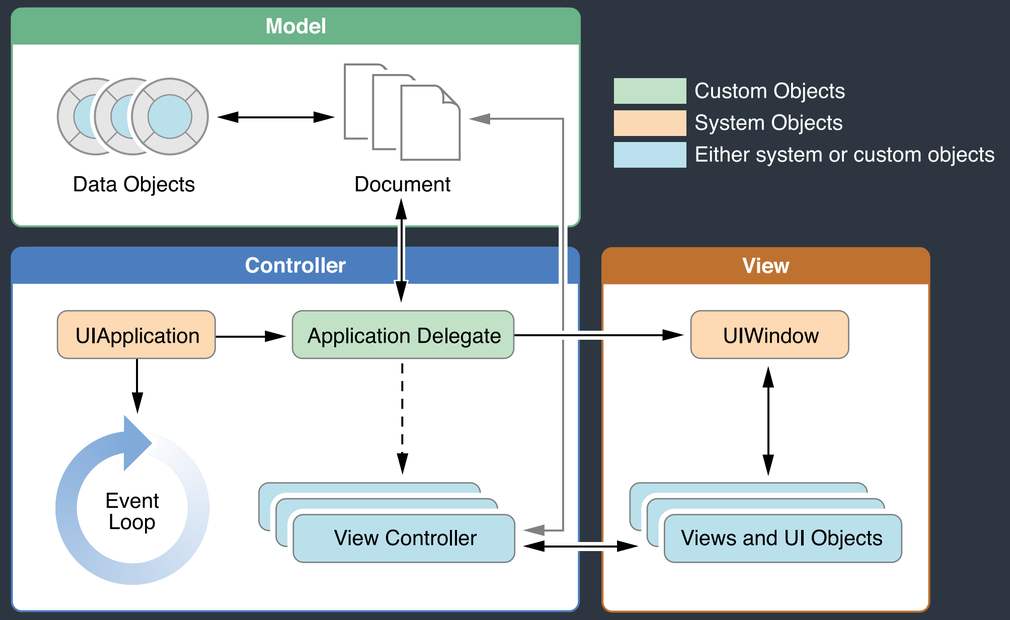
MVC Pattern

- Usually used in GUI application development
- Becomes popular for web applications too



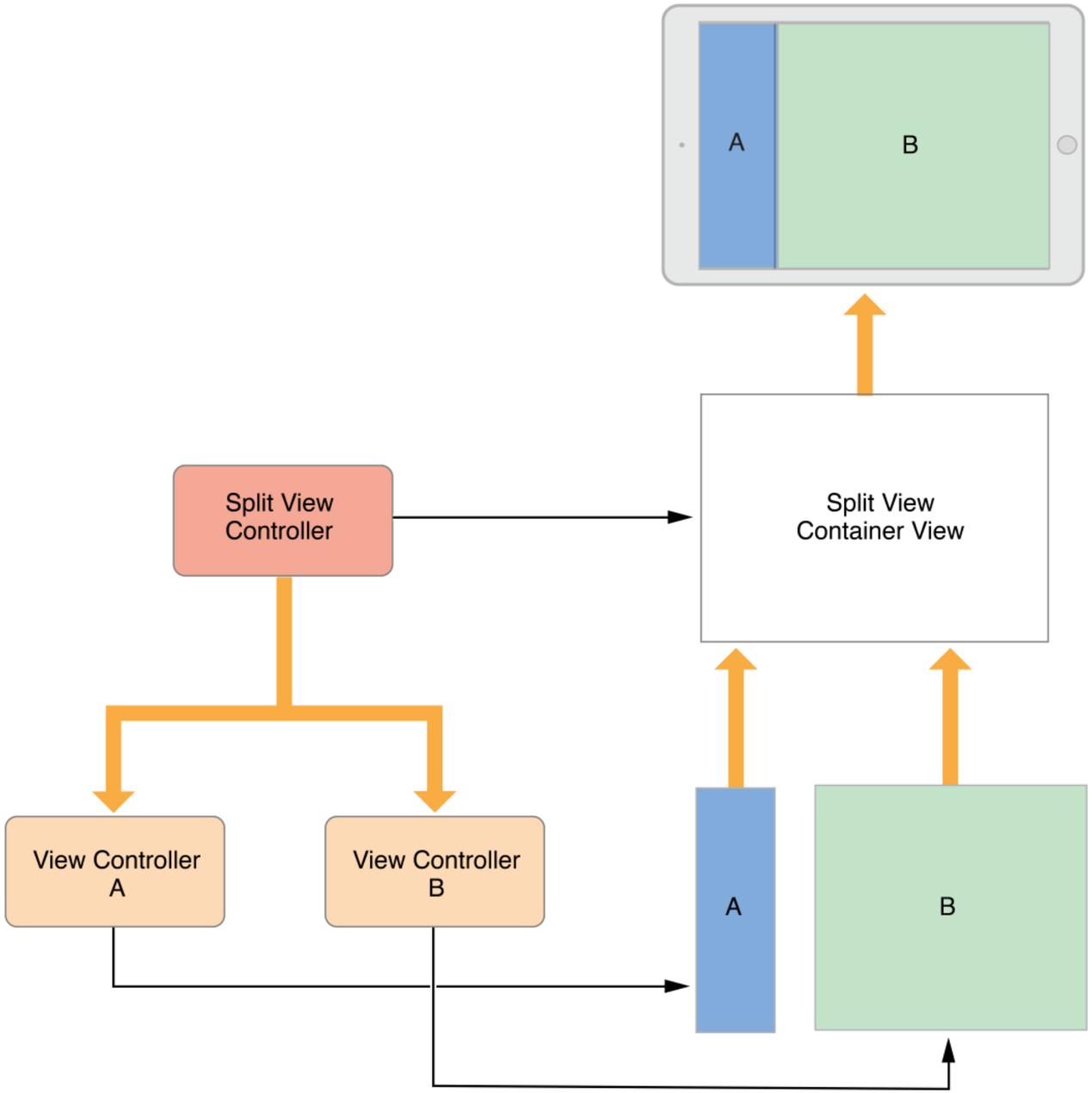
MVC Pattern

- The model directly manages the data, logic and rules of the application
- A view can be any output representation of information
- The controller accepts input and converts it to commands for the model or view



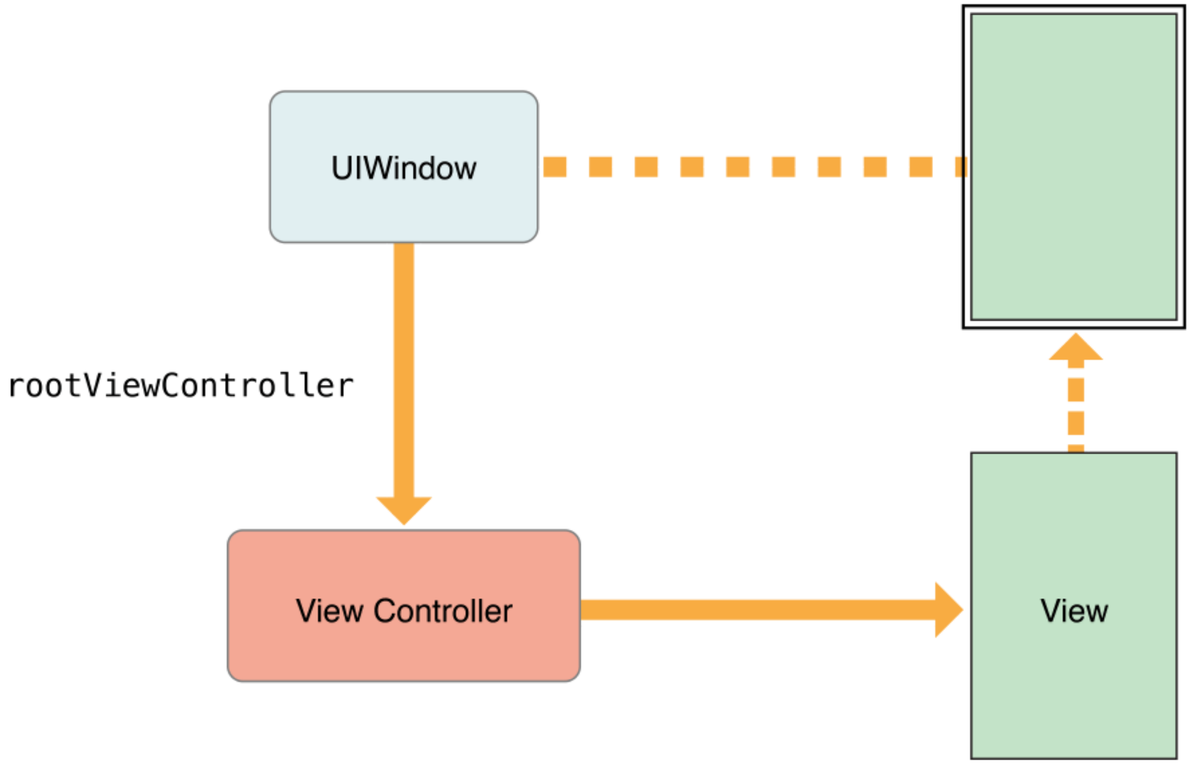
View Controller

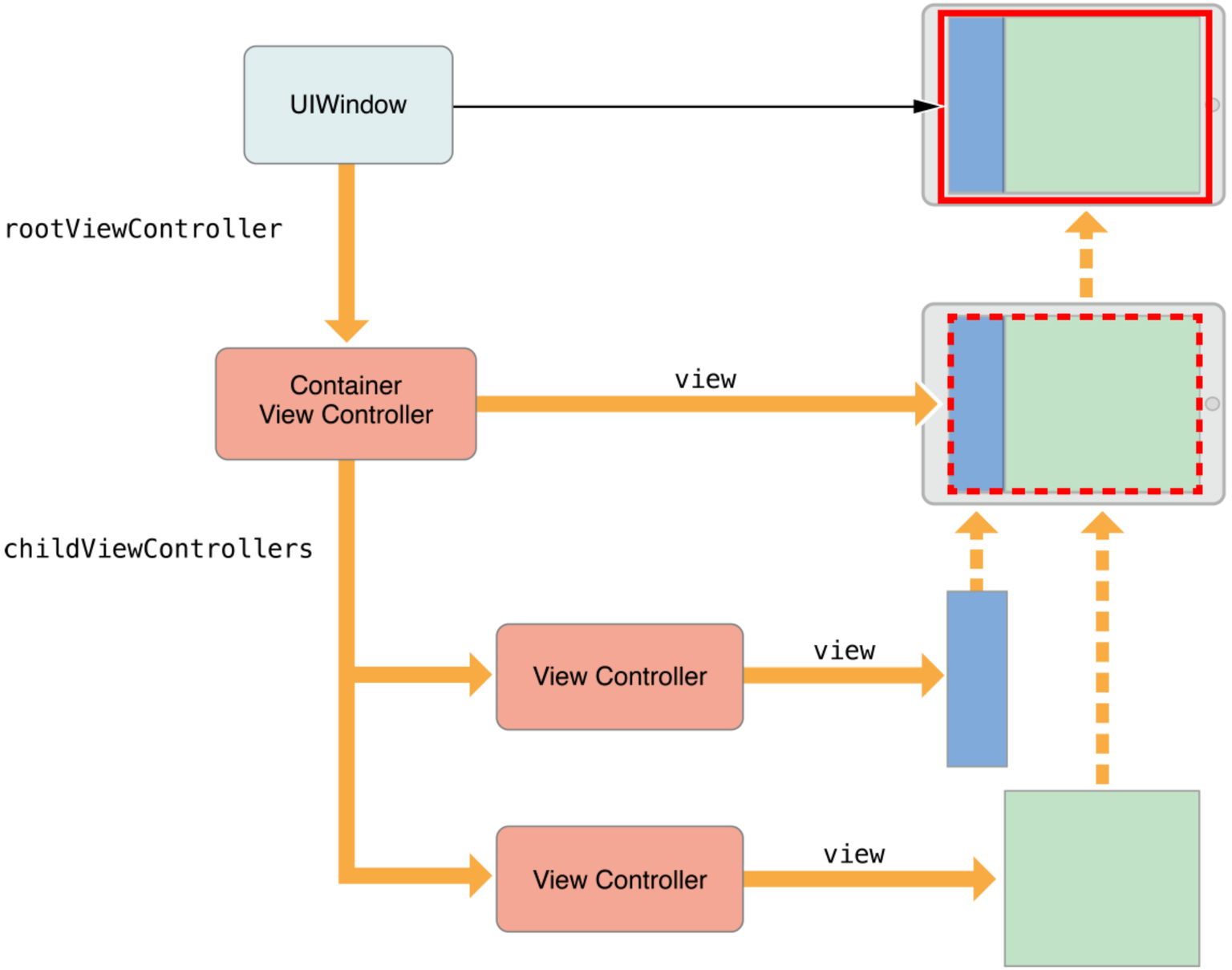
- View Management
  Including Layout and Adaptivity of a tree of views.
- User Interactions
- Data Marshaling
- Resource Management

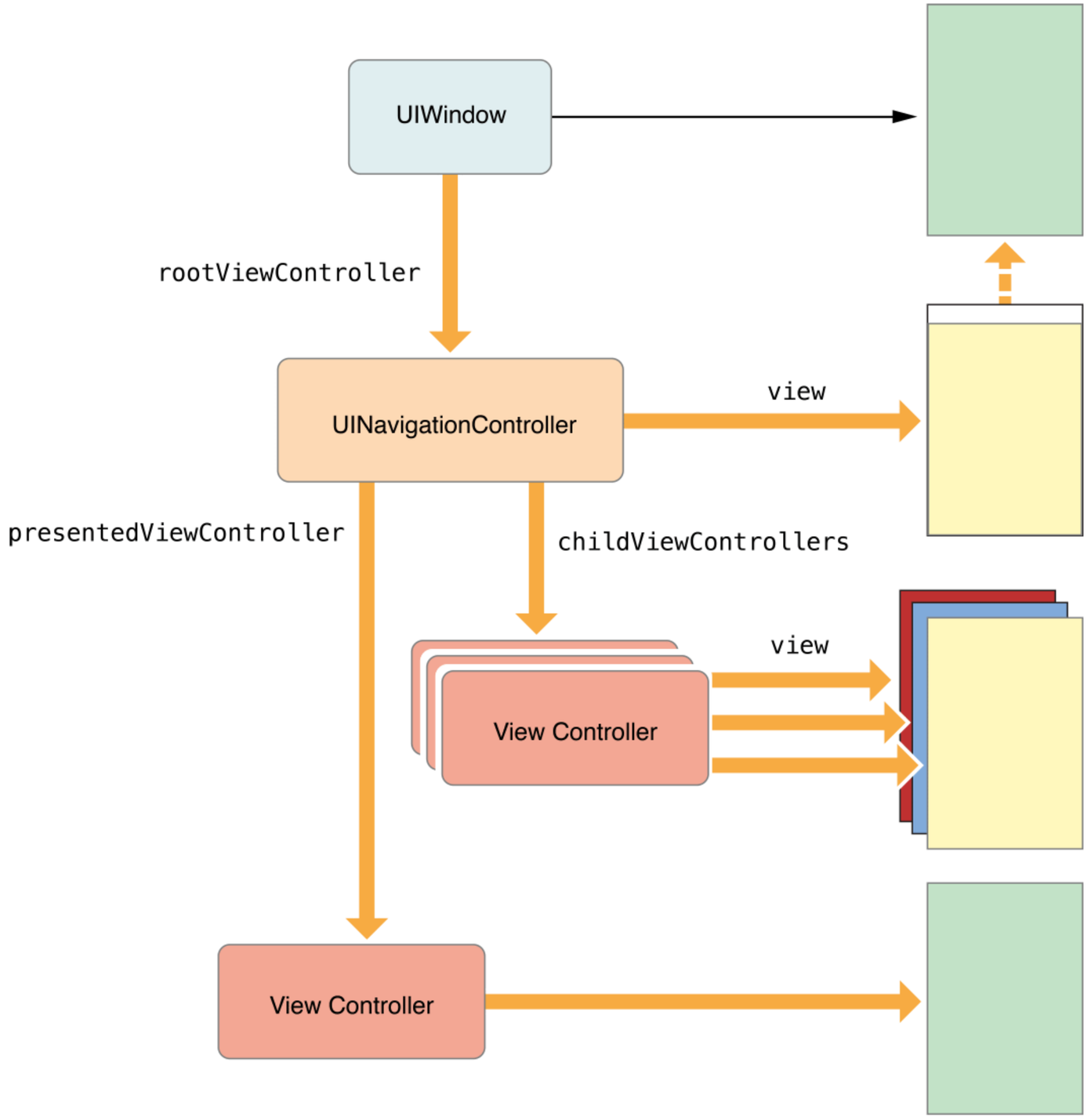


View Controller

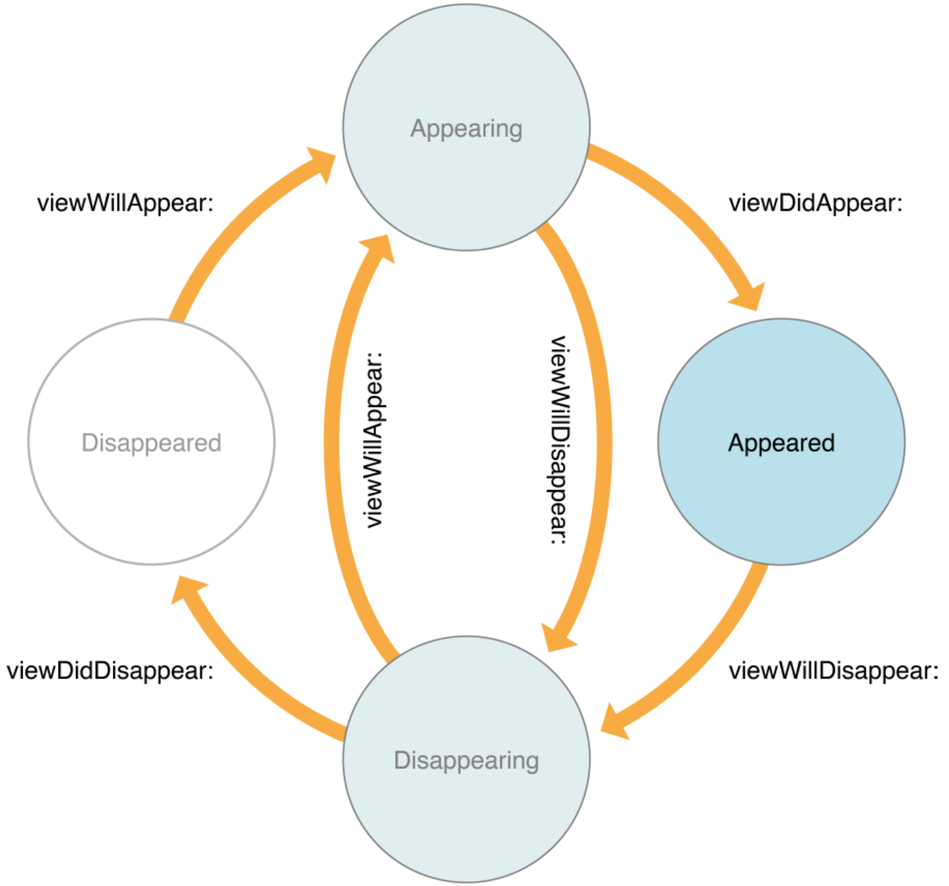
- Each app has a root view controller which attached to its window.
  It's usually the first view controller in your storyboard.




UIWindow
rootViewController
Container
View Controller
view
childViewControllers
View Controller
view
View Controller
view


UIWindow
rootViewController
UINavigationController
view
presentedViewController
childViewControllers
view
View Controller
View Controller

View Controller States



View Controller Memory Management

Methods	Usage / Task
`init` *(initializers)*	Allocate critical data structures required by your view controller.
`viewDidLoad`	Allocate or load data to be displayed in your views. Custom setup of your views.
`didReceiveMemoryWarning`	Respond to low-memory notifications.
`deinit`	Release resources if necessary.

Communication Patterns - I

Communication Patterns - I

- Notifications
- Key-Value Observation KVO
- Callback blocks/closures
- Delegation
- Target-Action

Target-Action Pattern

- Target-Action is the typical pattern used to send messages in response to UI events.
- Target-Action establishes is loose coupling and easy to setup between the event sender and the receiver.
  Storyboard connections and UIControl uses this pattern.
- Compilers could not check and validate for developers, like duck-typing.

Delegate Pattern

- Delegation is a widespread pattern throughout Apple's frameworks.
  It allows us to customize an object's behavior and to be notified about certain events.

- Delegation uses protocols to make sure the receiver understands curtain methods.
  It's still loose-coupled, but compilers know how to check the relationship between event sender and receiver.

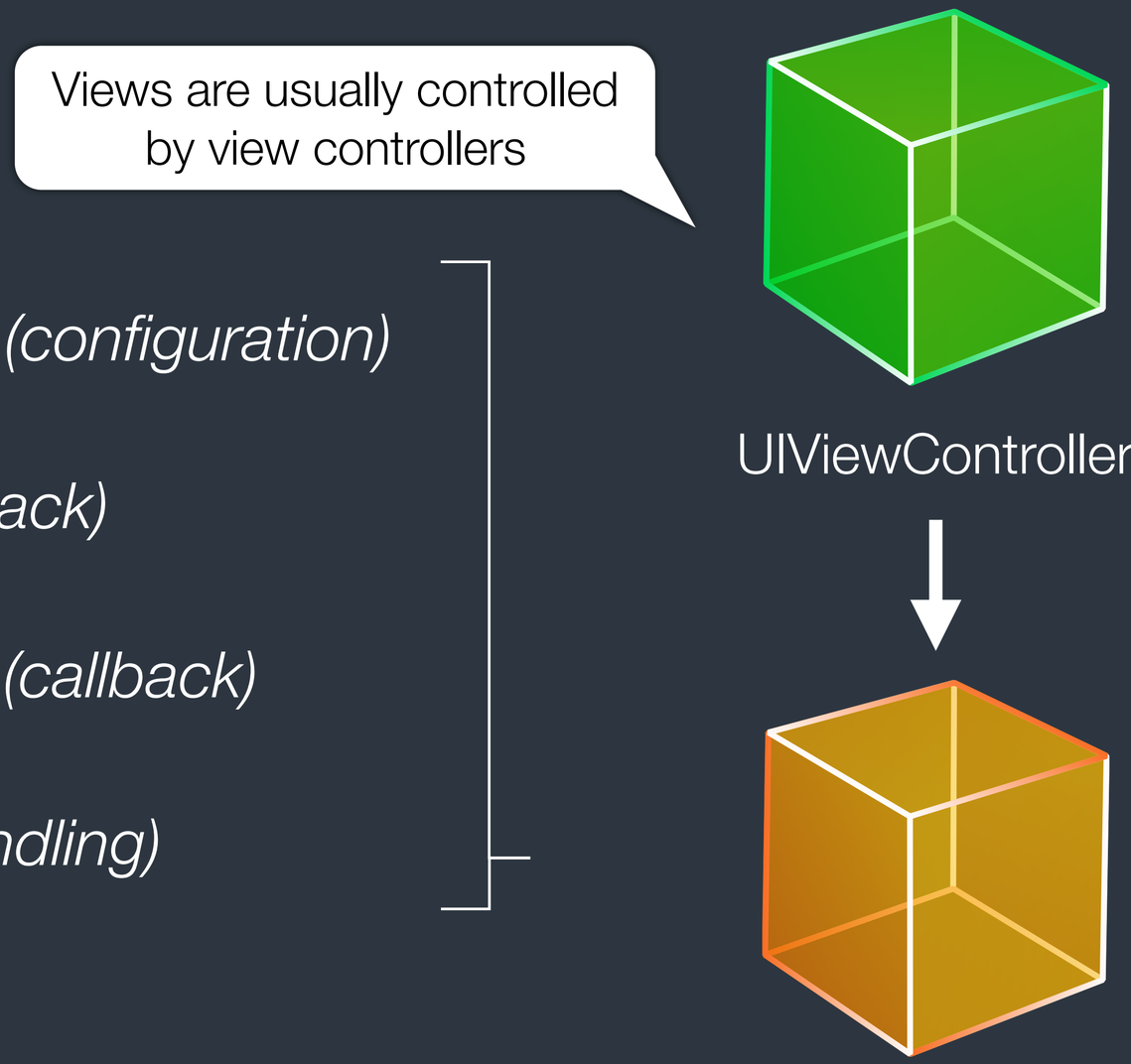
Delegate Pattern

- Should I start loading this URL? *(configuration)*
- I did start loading the URL. *(callback)*
- I have finished loading the URL. *(callback)*
- I failed to load the URL. *(error handling)*

UIWebView

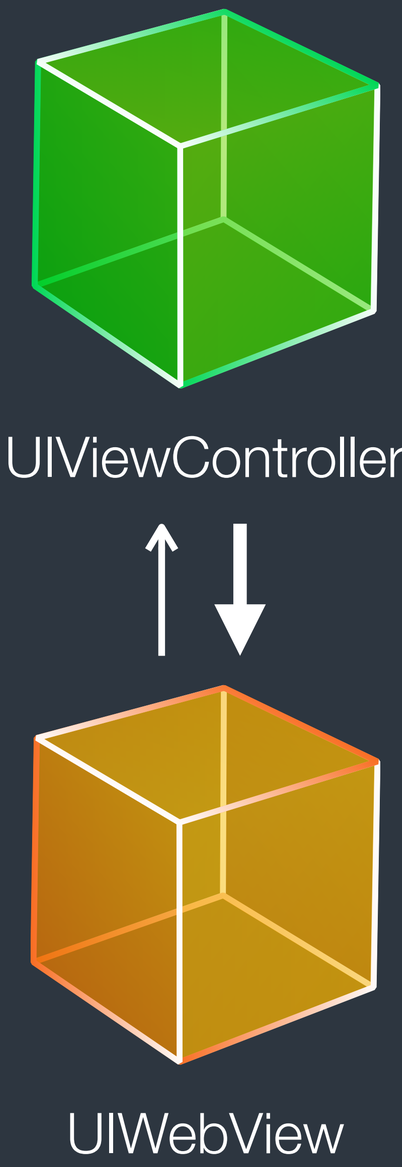
Delegate Pattern

- Should I start loading this URL? *(configuration)*
- I did start loading the URL. *(callback)*
- I have finished loading the URL. *(callback)*
- I failed to load the URL. *(error handling)*



Delegate Pattern

- Should I start loading this URL? *(configuration)*
- I did start loading the URL. *(callback)*
- I have finished loading the URL. *(callback)*
- I failed to load the URL. *(error handling)*



Delegate Pattern

Protocols could make compilers check conformity

Use protocols as a more general and abstract type

- Should I start loading this URL? *(configuration)*
- I did start loading the URL. *(callback)*
- I have finished loading the URL. *(callback)*
- I failed to load the URL. *(error handling)*

UIWebViewDelegate

UIWebView

Delegate Pattern - Declaration

```
@objc protocol MYWebViewDelegate: NSObjectProtocol {
    optional func webView(webView: MYWebView, shouldLoadURL url: NSURL) -> Bool
    optional func webView(webView: MYWebView, didStartLoadingURL url: NSURL)
    optional func webView(webView: MYWebView, finishedLoadingURL url: NSURL)
    optional func webView(webView: MYWebView, failedToLoadURL url: NSURL,
        withError error: NSError?)
}
```

- We usually pass the instance into delegate methods for identification since the delegatee may be shared by multiple delegators.

Delegate Pattern - Implementation

```
class MYWebView: UIView {
    weak var delegate: MYWebViewDelegate?

    func loadURL(url: NSURL) {
        if let shouldLoad = self.delegate?.webView?(self, shouldLoadURL: url) {
            if !shouldLoad { return }
        }
        self.delegate?.webView?(self, didStartLoadingURL: url)
        // Load ...
        let success = true
        // Done
        if success {
            self.delegate?.webView?(self, finishedLoadingURL: url)
        } else {
            self.delegate?.webView?(self, failedToLoadURL: url, withError: nil)
        }
    }
}
```

Delegate Pattern - Adoption

```
class MYViewController: UIViewController, MYWebViewDelegate {

    var webView: MYWebView!

    override func viewDidLoad() {
        super.viewDidLoad()
        self.webView = MYWebView()
        self.webView.delegate = self
    }

    func webView(webView: MYWebView, didStartLoadingURL url: NSURL) {
        print("Start loading url: \(url)")
    }
}
```

Demo: TextFieldDelegate *(uikit-intro repo)*

References

- Using Swift with Cocoa and Objective-C
- Strategies for Handling App State Transitions
  App Programming Guide for iOS
- File System Programming Guide
- About Asset Catalogs
- View Controller Programming Guide for iOS
- Communication Patterns @ objc.io
- Target-Action
  Concepts in Objective-C Programming
- Delegates and Data Sources
  Concepts in Objective-C Programming