



# Stanford CS193p

Developing Applications for iOS  
Fall 2017-18



CS193p  
Fall 2017-18



# Today

## 👁️ Alert and Action Sheet

Presenting a view controller to notify user of extraordinary event

Or to make a “branching decision” in the UI

Demo: Report bad dropped background images to the user in EmojiArt

## 👁️ Notifications & KVO

Finding out what's going on

e.g. Keyboard appeared or Document State changed or User Font Size changed

Demo: Use KVO and Notifications instead of Delegation for EmojiArtView change tracking

## 👁️ Application Lifecycle

What the heck are all those methods in AppDelegate.swift?





# Alerts and Action Sheets

- Two kinds of “pop up and ask the user something” mechanisms

- Alerts

- Action Sheets

- Alerts

- Pop up in the middle of the screen.

- Usually ask questions with only two answers (e.g. OK/Cancel, Yes/No, etc.).

- Can be disruptive to your user-interface, so use carefully.

- Often used for “asynchronous” problems (“connection reset” or “network fetch failed”).

- Can have a text field to get a quick answer (e.g. password)

- Action Sheets

- Usually slides in from the bottom of the screen on iPhone/iPod Touch, and in a popover on iPad.

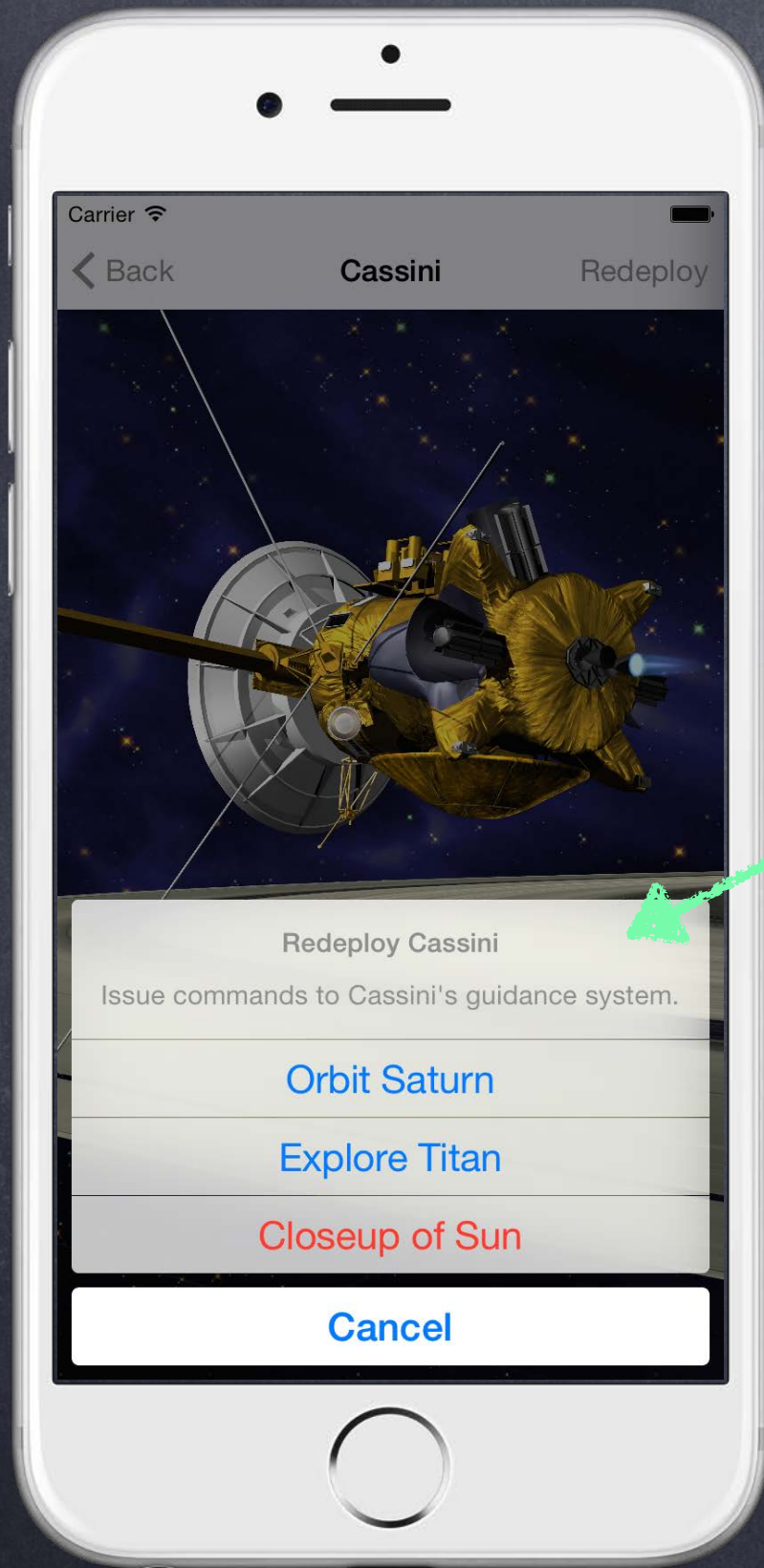
- Can be displayed from bar button item or from any rectangular area in a view.

- Generally asks questions that have more than two answers.

- Think of action sheets as presenting “branching decisions” to the user (i.e. what next?).

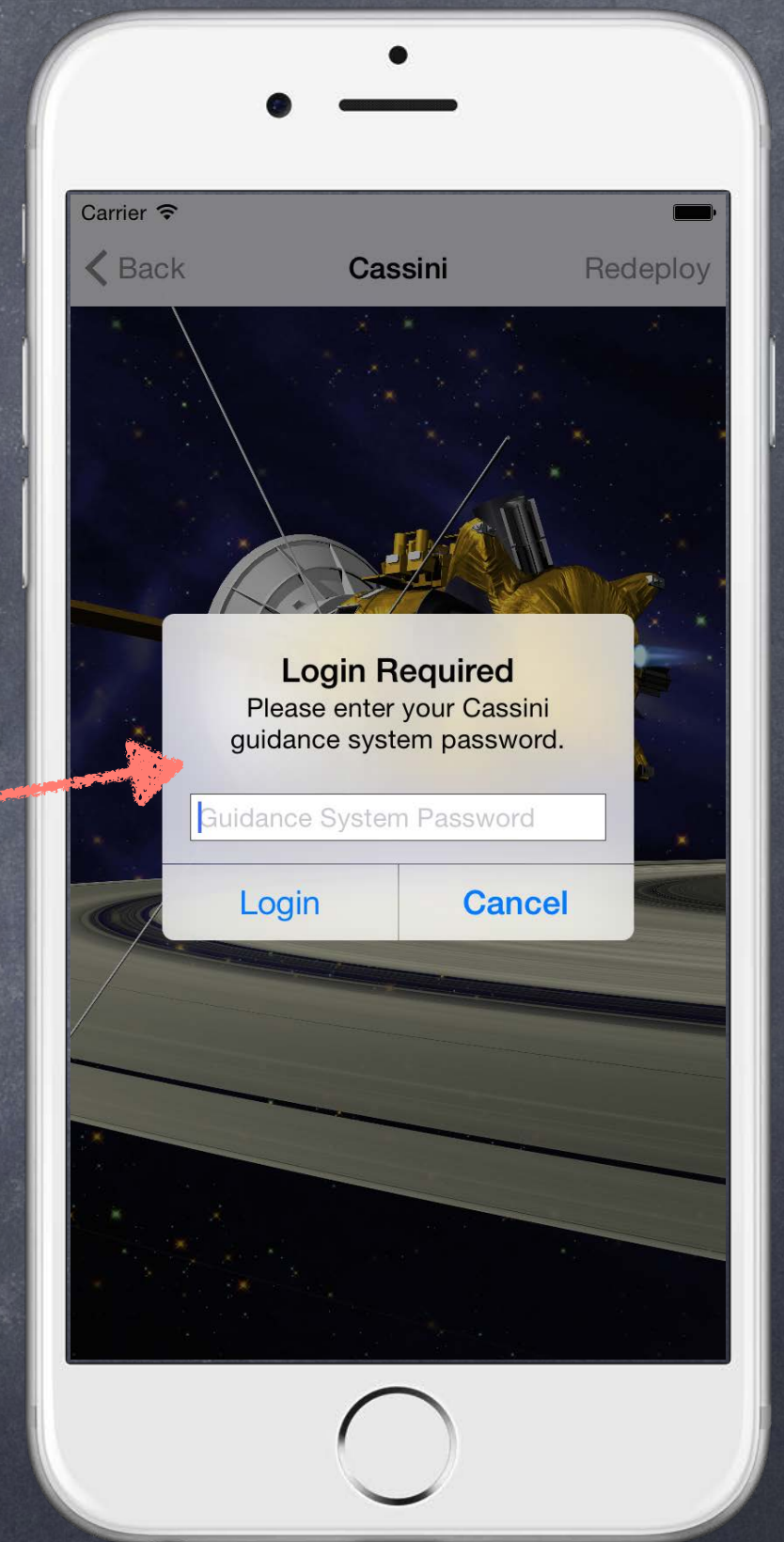




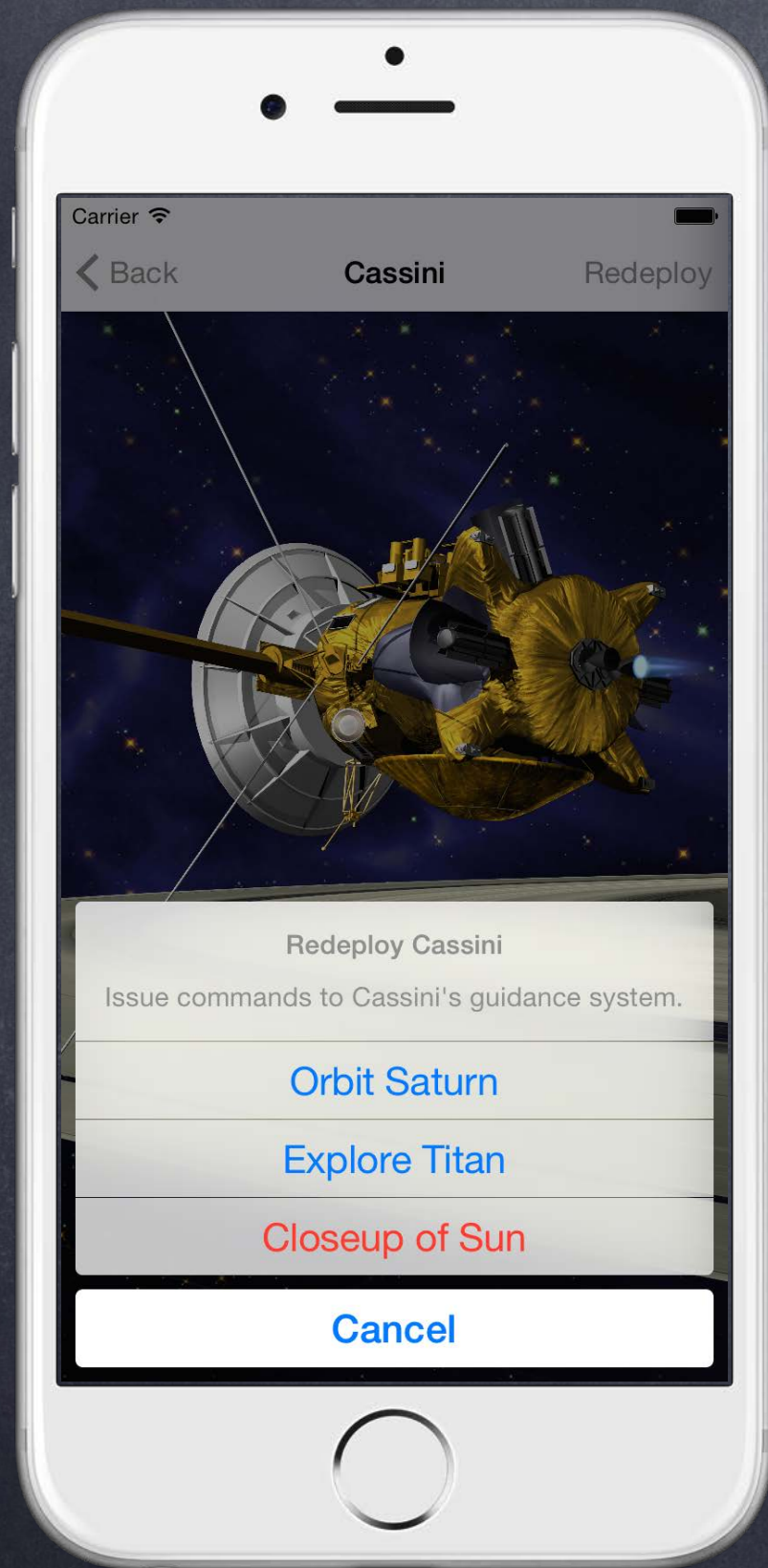


Action Sheet

Alert



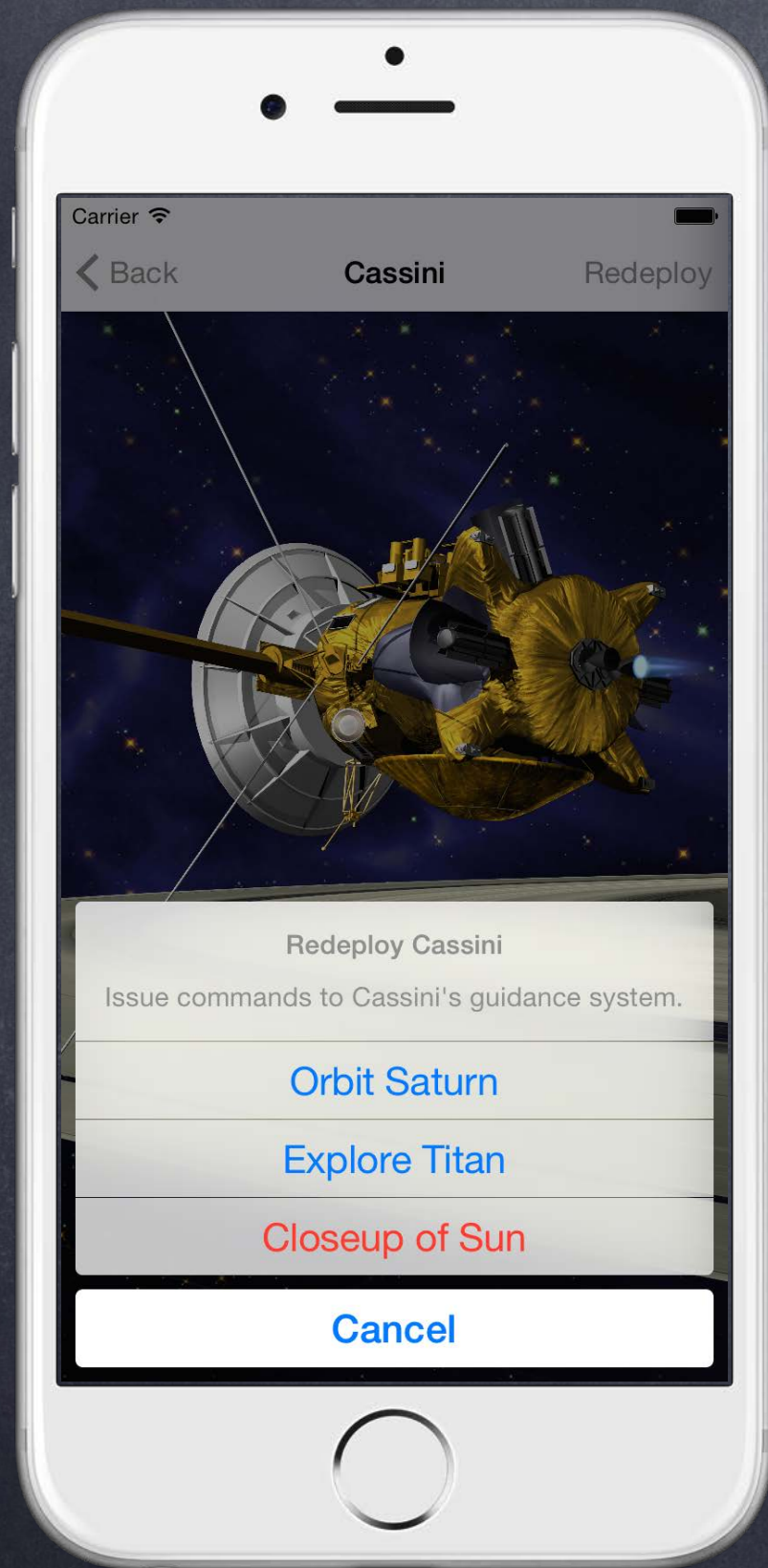




```
var alert = UIAlertController(  
  title: "Redeploy Cassini",  
  message: "Issue commands to Cassini's guidance system.",  
  preferredStyle: .actionSheet  
)
```



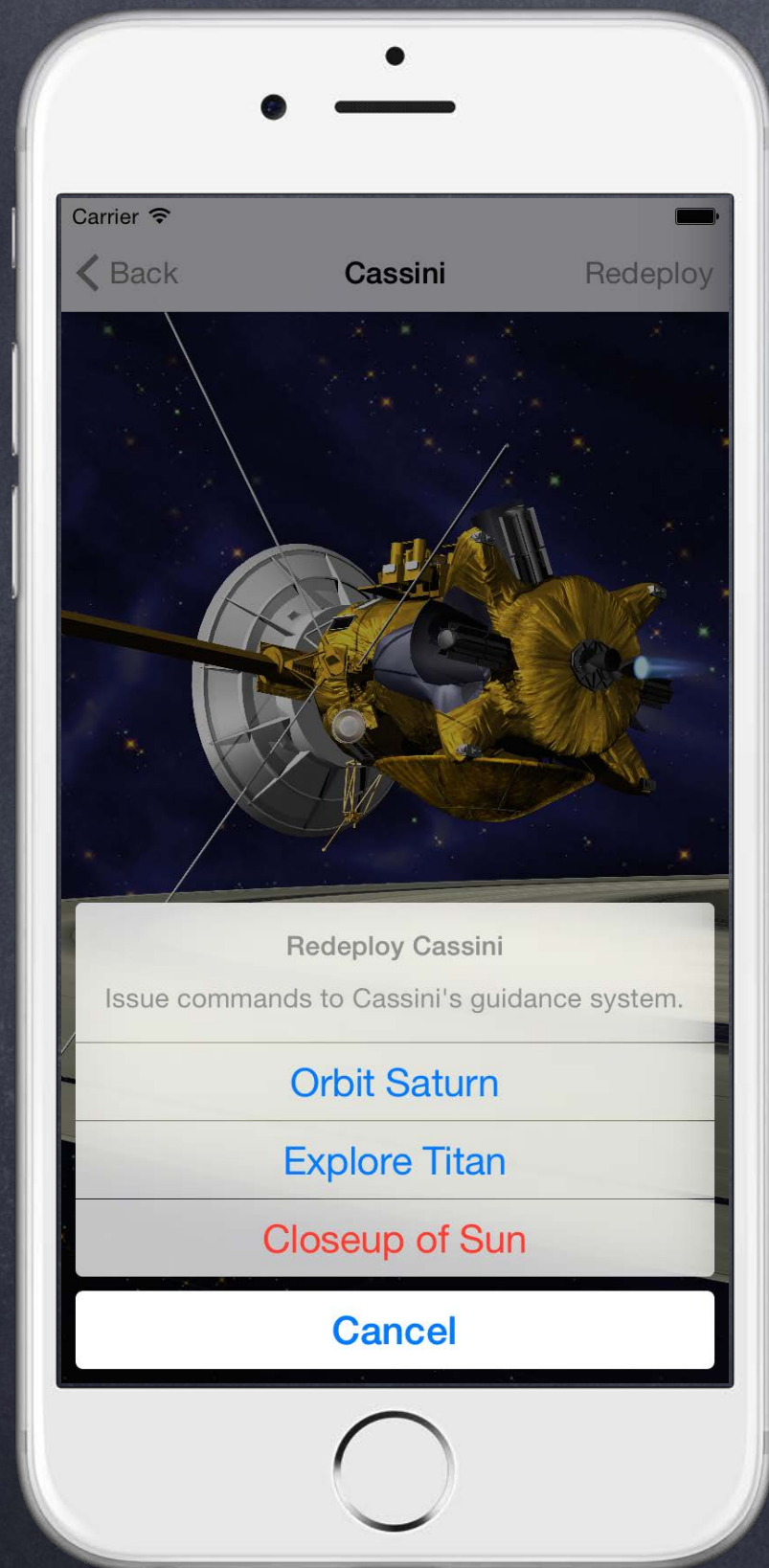




```
var alert = UIAlertController(  
  title: "Redeploy Cassini",  
  message: "Issue commands to Cassini's guidance system.",  
  preferredStyle: .actionSheet  
)
```



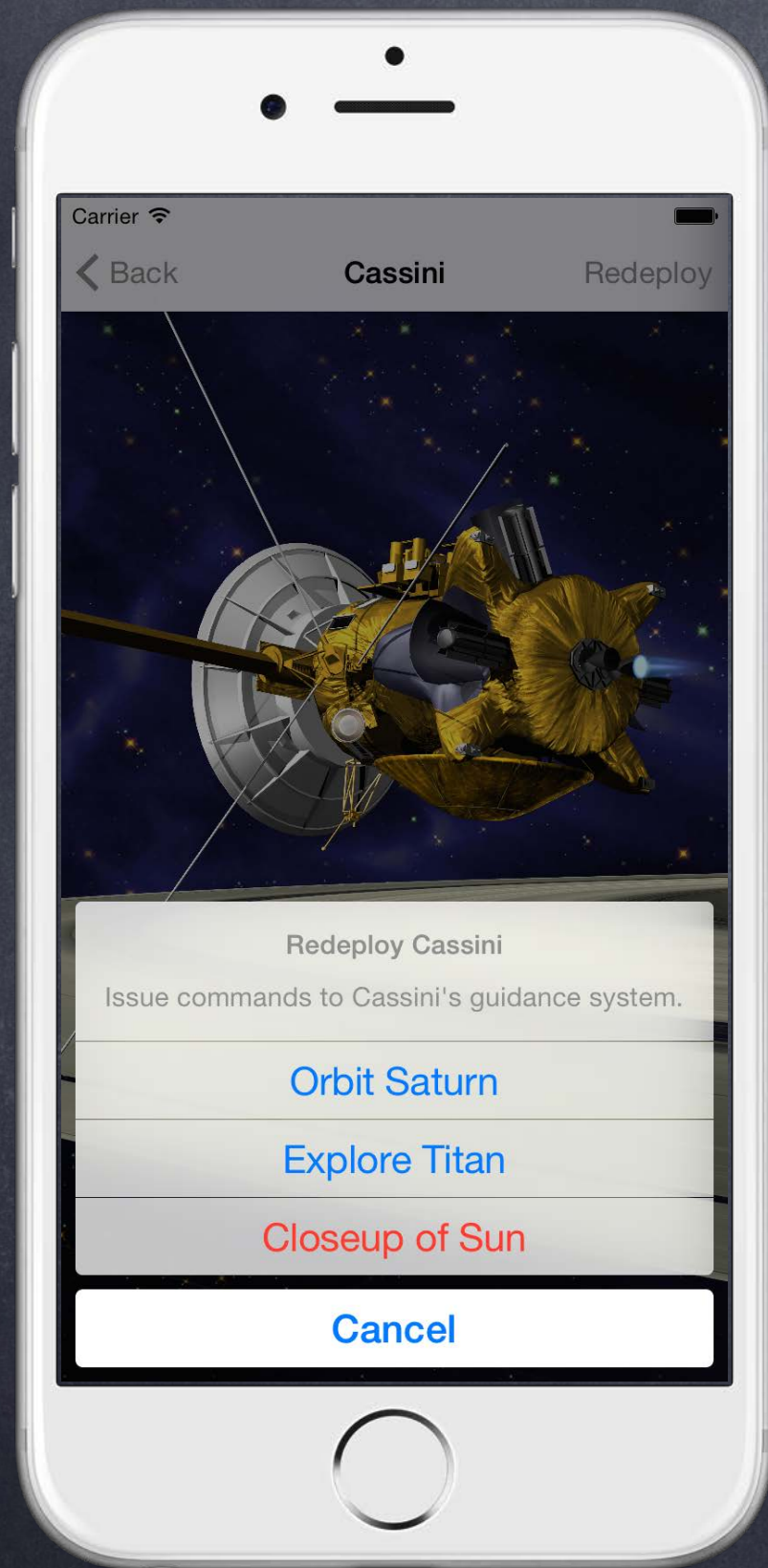




```
var alert = UIAlertController(  
    title: "Redeploy Cassini",  
    message: "Issue commands to Cassini's guidance system.",  
    preferredStyle: .actionSheet  
)  
  
alert.addAction(...)
```



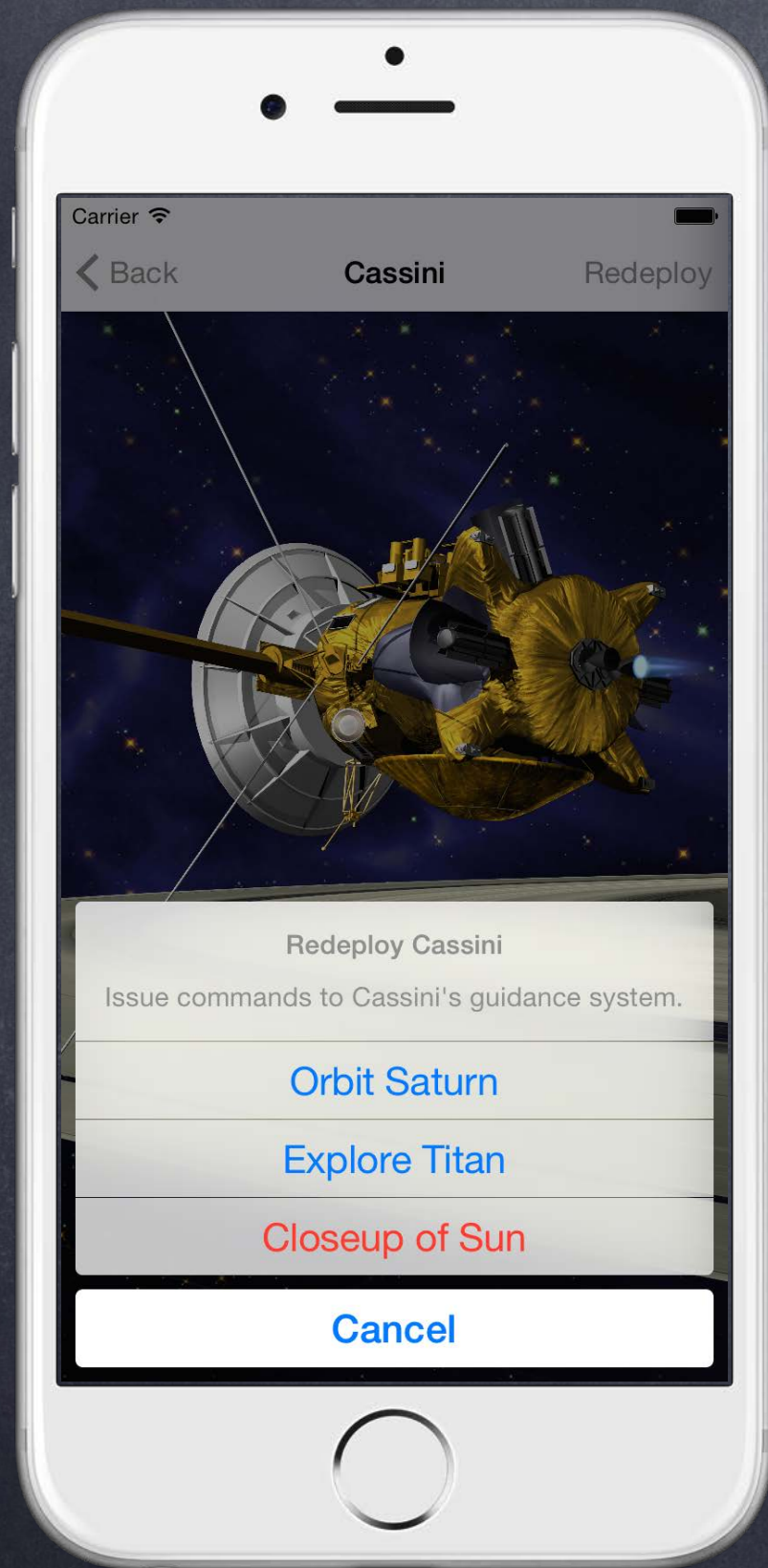




```
var alert = UIAlertController(  
    title: "Redeploy Cassini",  
    message: "Issue commands to Cassini's guidance system.",  
    preferredStyle: .actionSheet  
)  
  
alert.addAction(UIAlertAction(...))
```



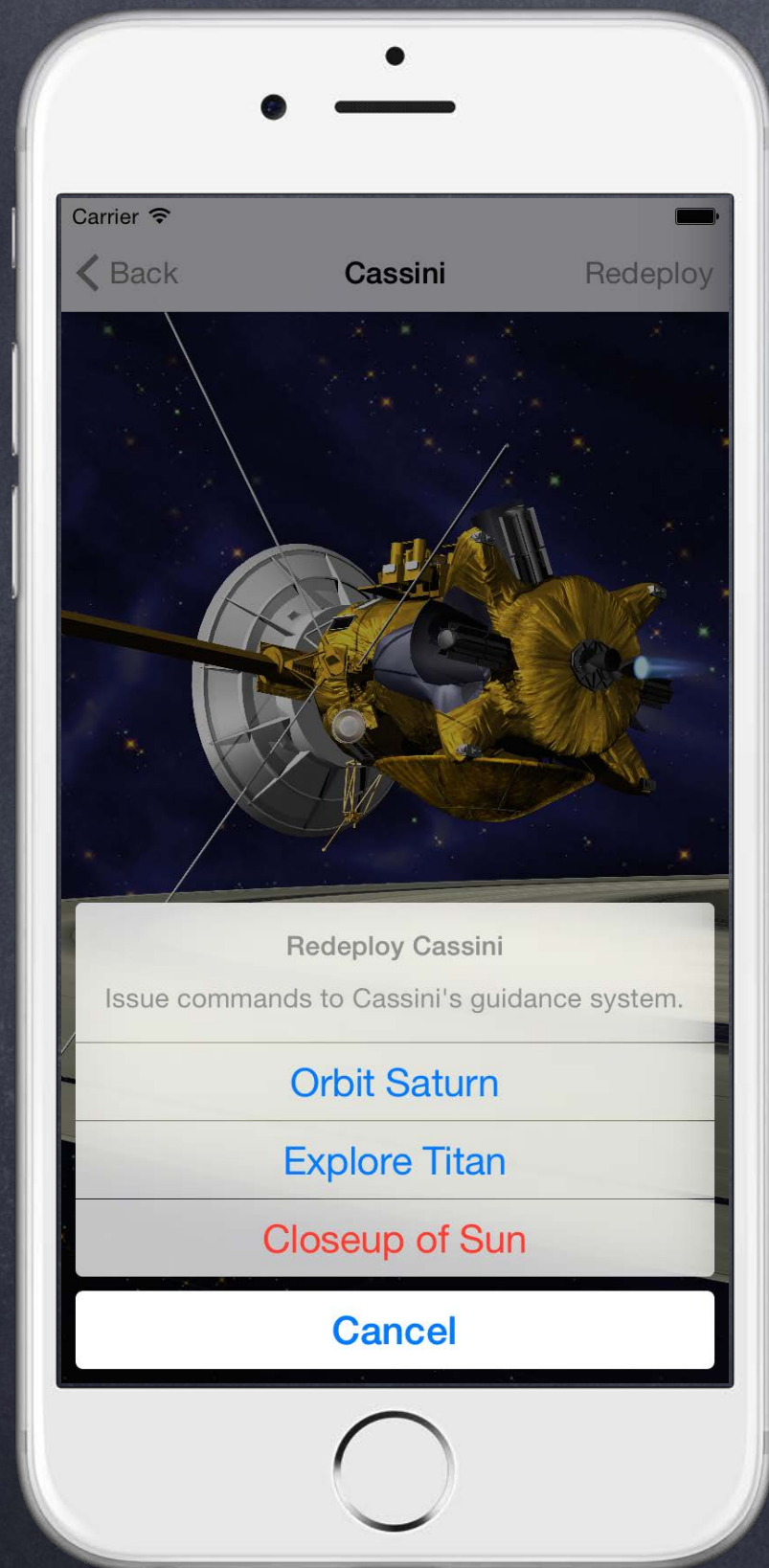




```
var alert = UIAlertController(  
    title: "Redeploy Cassini",  
    message: "Issue commands to Cassini's guidance system.",  
    preferredStyle: .actionSheet  
)  
  
alert.addAction(UIAlertAction(  
    title: String,  
    style: UIAlertActionStyle,  
    handler: (action: UIAlertAction) -> Void  
))
```



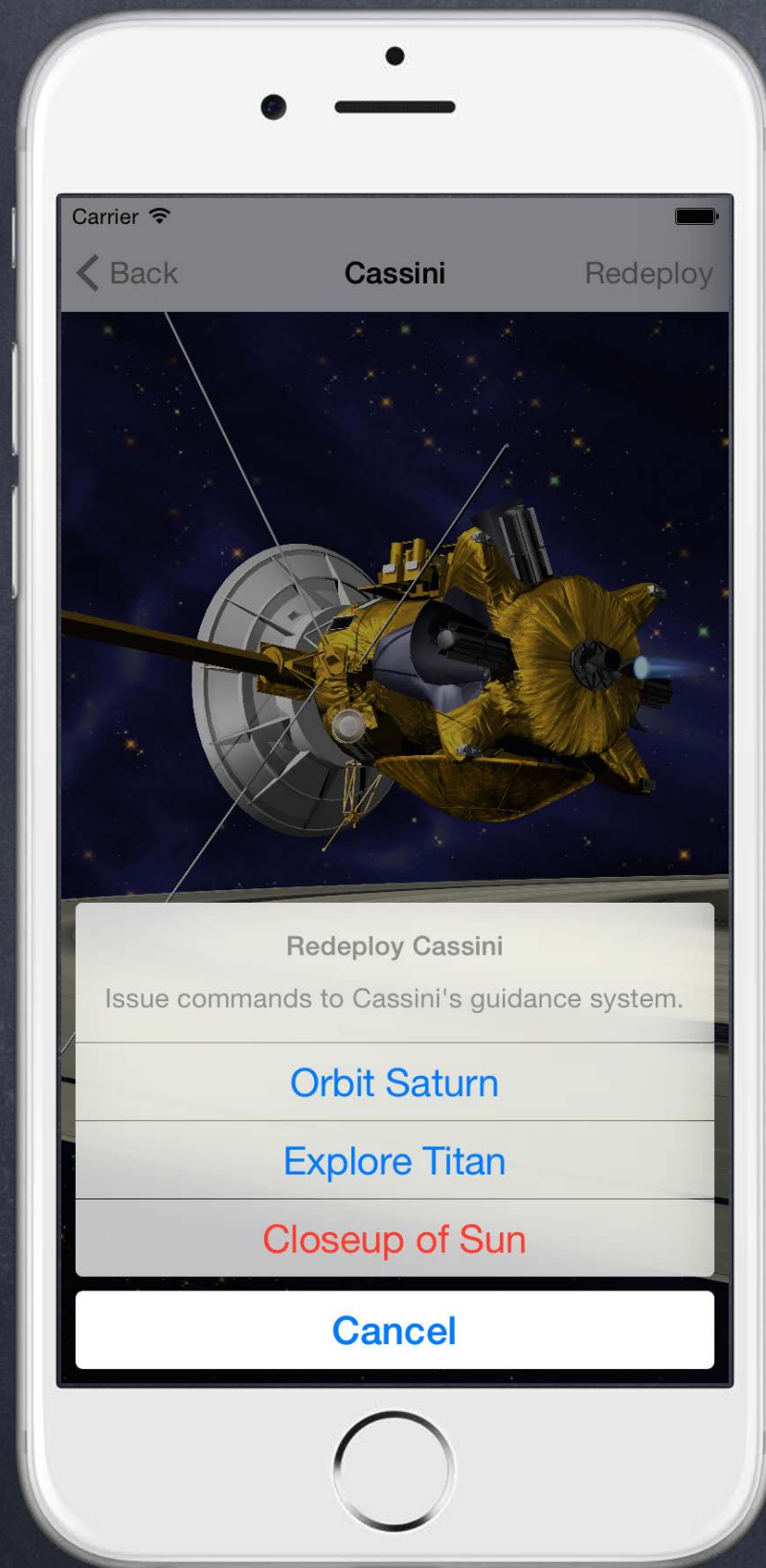




```
var alert = UIAlertController(  
    title: "Redeploy Cassini",  
    message: "Issue commands to Cassini's guidance system.",  
    preferredStyle: .actionSheet  
)  
  
alert.addAction(UIAlertAction(  
    title: "Orbit Saturn",  
    style: UIAlertActionStyle.default)  
    { (action: UIAlertAction) -> Void in  
        // go into orbit around saturn  
    }  
)  
)
```







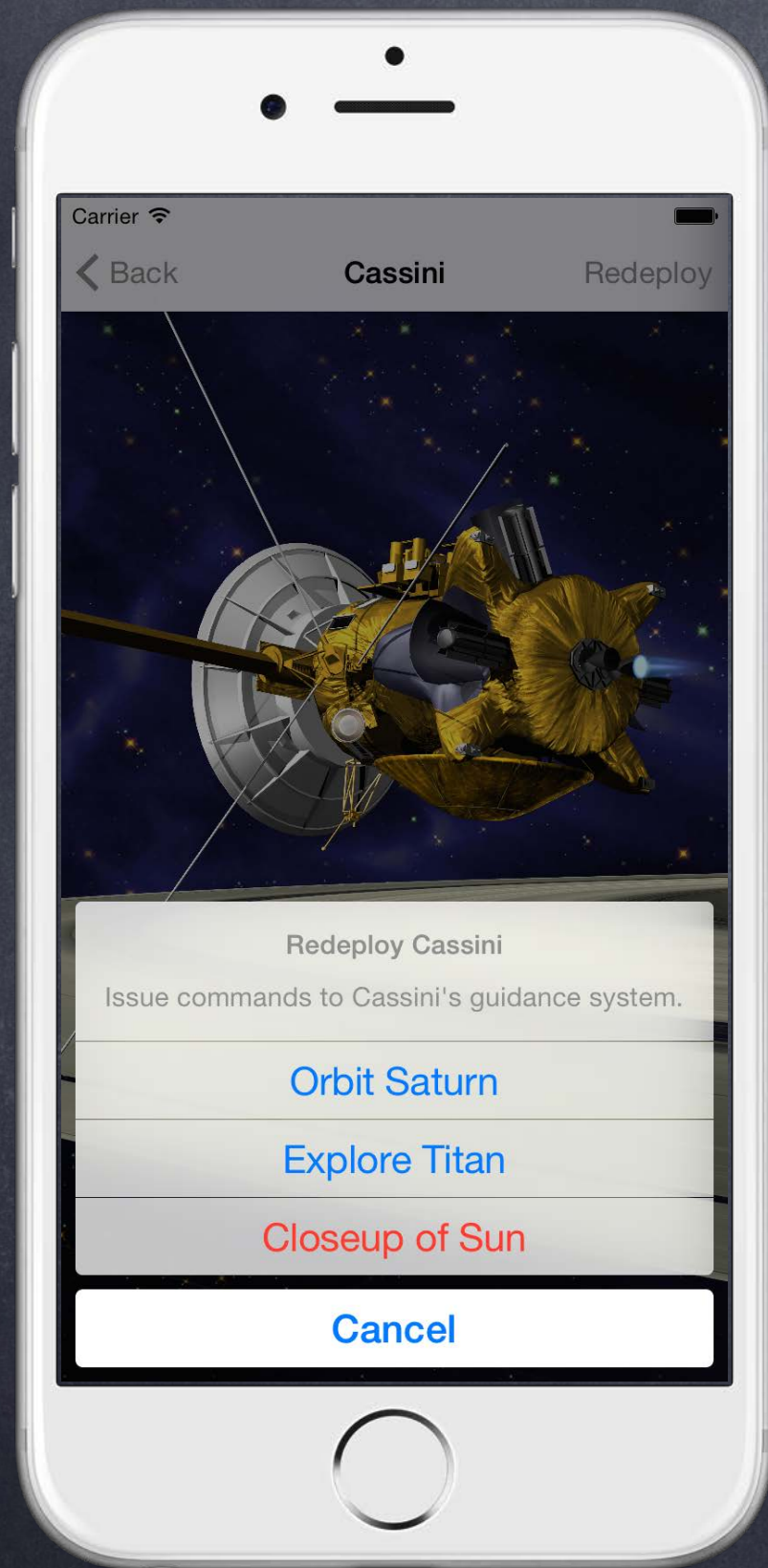
```
var alert = UIAlertController(
    title: "Redeploy Cassini",
    message: "Issue commands to Cassini's guidance system.",
    preferredStyle: .actionSheet
)

alert.addAction(UIAlertAction(
    title: "Orbit Saturn",
    style: UIAlertActionStyle.default)
{ (action: UIAlertAction) -> Void in
    // go into orbit around saturn
}
)

alert.addAction(UIAlertAction(
    title: "Explore Titan",
    style: .default)
{ (action: UIAlertAction) -> Void in
    if !self.loggedIn { self.login() }
    // if loggedIn go to titan
}
)
)
```





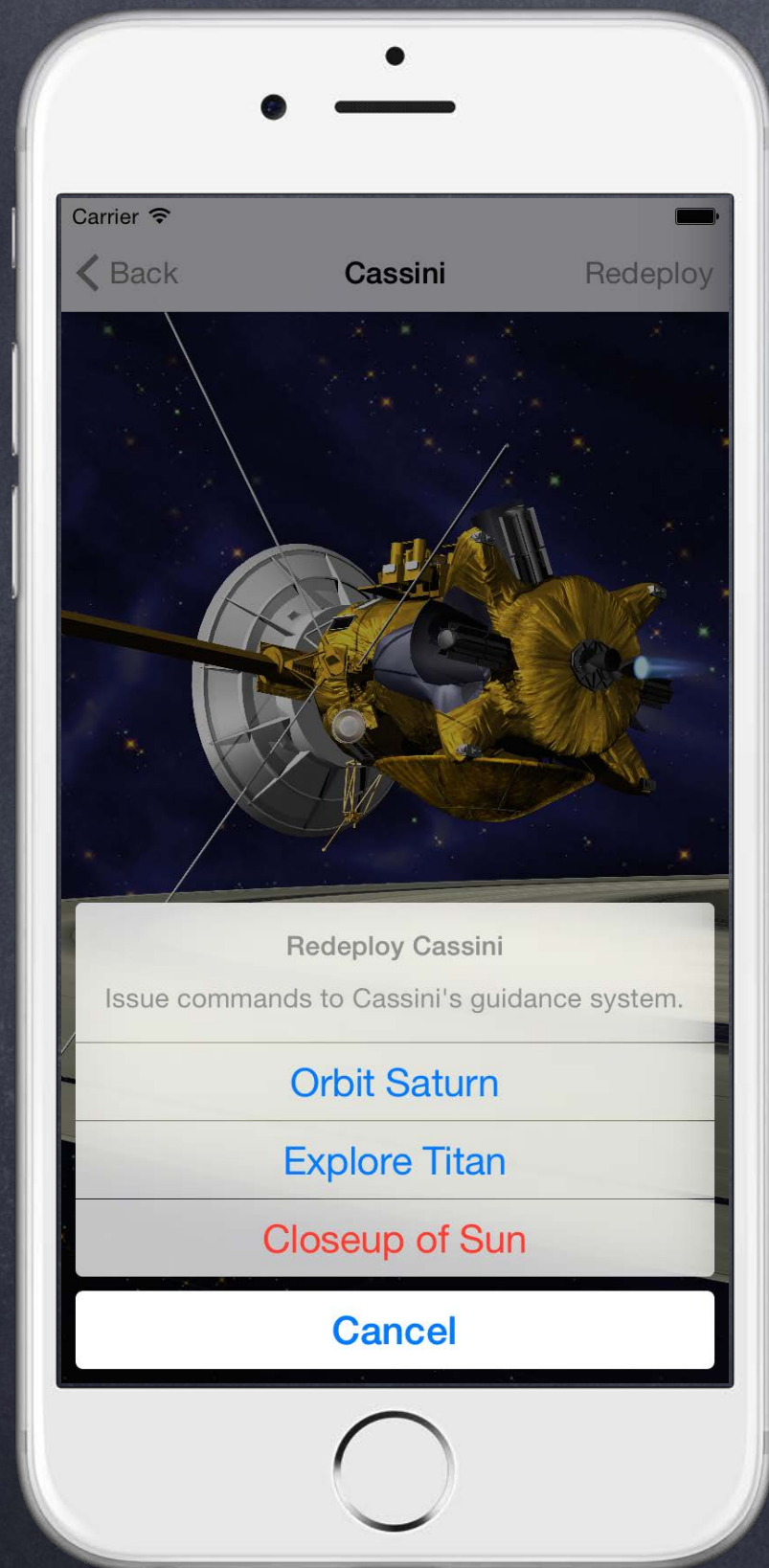


```
var alert = UIAlertController(  
    title: "Redeploy Cassini",  
    message: "Issue commands to Cassini's guidance system.",  
    preferredStyle: .actionSheet  
)
```

```
alert.addAction(/* orbit saturn action */)   
alert.addAction(/* explore titan action */)
```







```
var alert = UIAlertController(
    title: "Redeploy Cassini",
    message: "Issue commands to Cassini's guidance system.",
    preferredStyle: .actionSheet
)

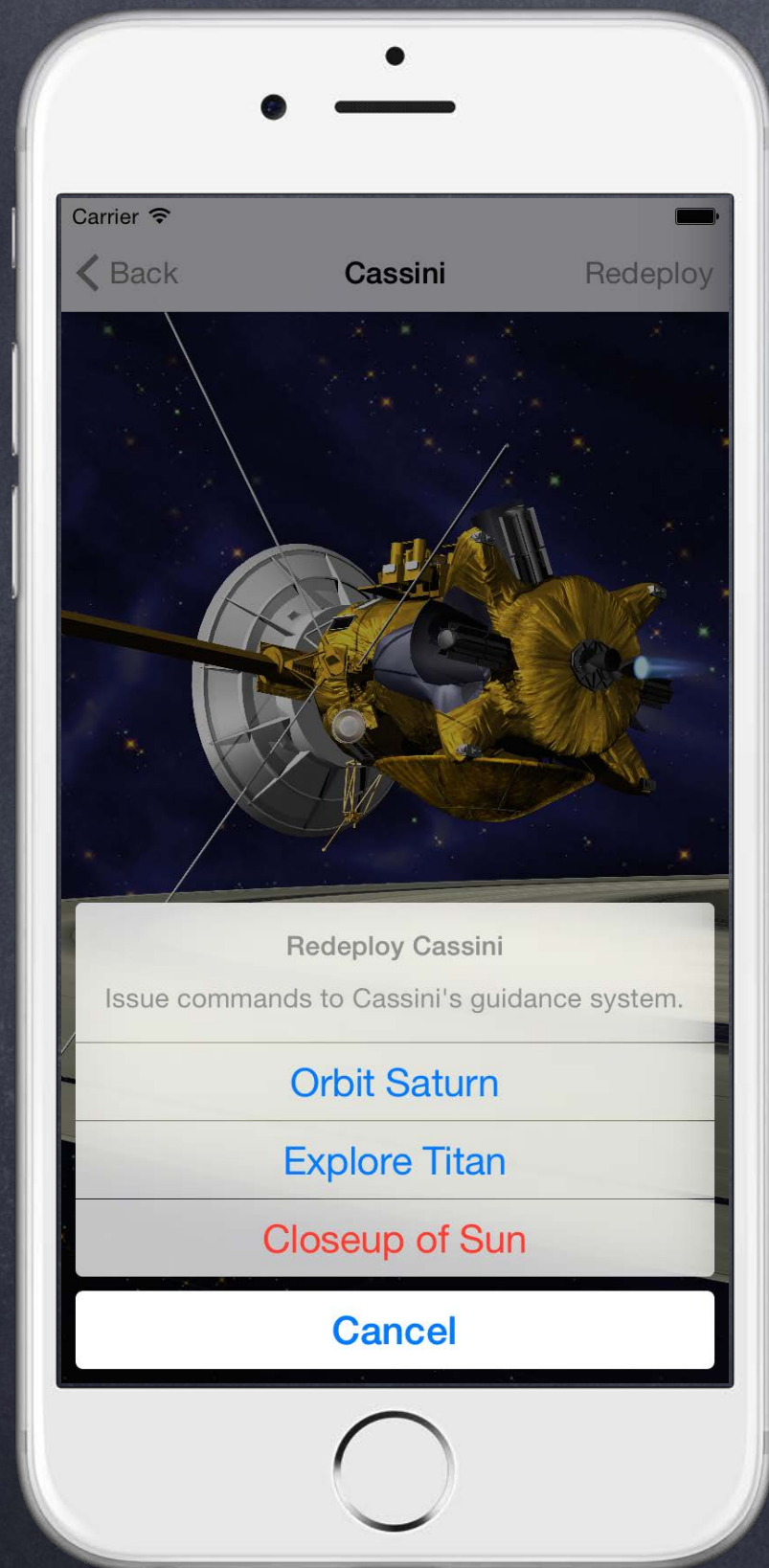
alert.addAction(/* orbit saturn action */)
alert.addAction(/* explore titan action */)

alert.addAction(UIAlertAction(
    title: "Closeup of Sun",
    style: .destructive)
    { (action: UIAlertAction) -> Void in
        if !loggedIn { self.login() }
        // if loggedIn destroy Cassini by going to Sun
    }
)

alert.addAction(UIAlertAction(
    title: "Cancel",
    style: .cancel)
    { (action: UIAlertAction) -> Void in
        // do nothing
    }
)
)
```





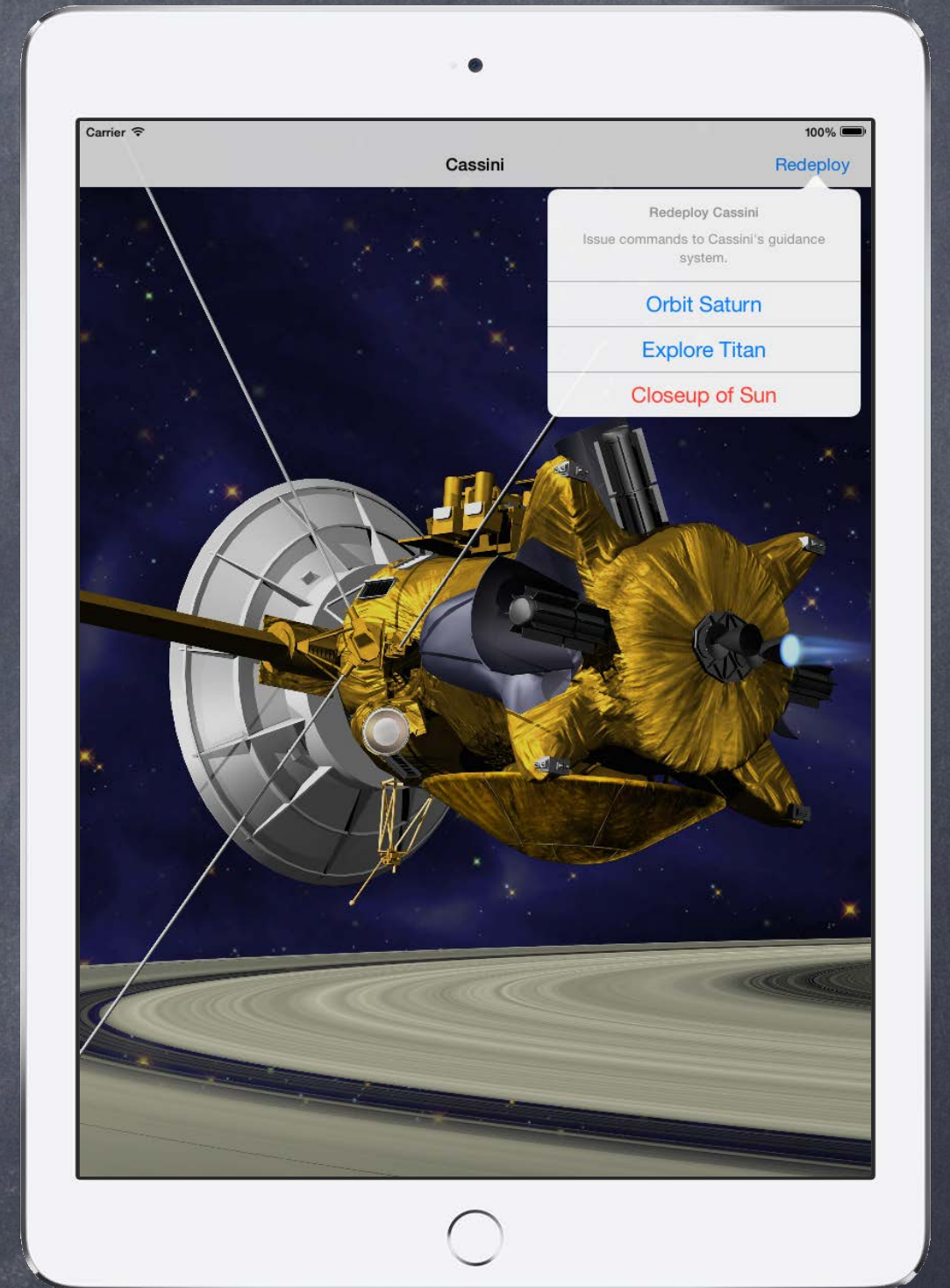


```
var alert = UIAlertController(  
    title: "Redeploy Cassini",  
    message: "Issue commands to Cassini's guidance system.",  
    preferredStyle: .actionSheet  
)  
  
alert.addAction(/* orbit saturn action */)   
alert.addAction(/* explore titan action */)   
alert.addAction(/* destroy with closeup of sun action */)   
alert.addAction(/* do nothing cancel action */)   
  
present(alert, animated: true, completion: nil)
```



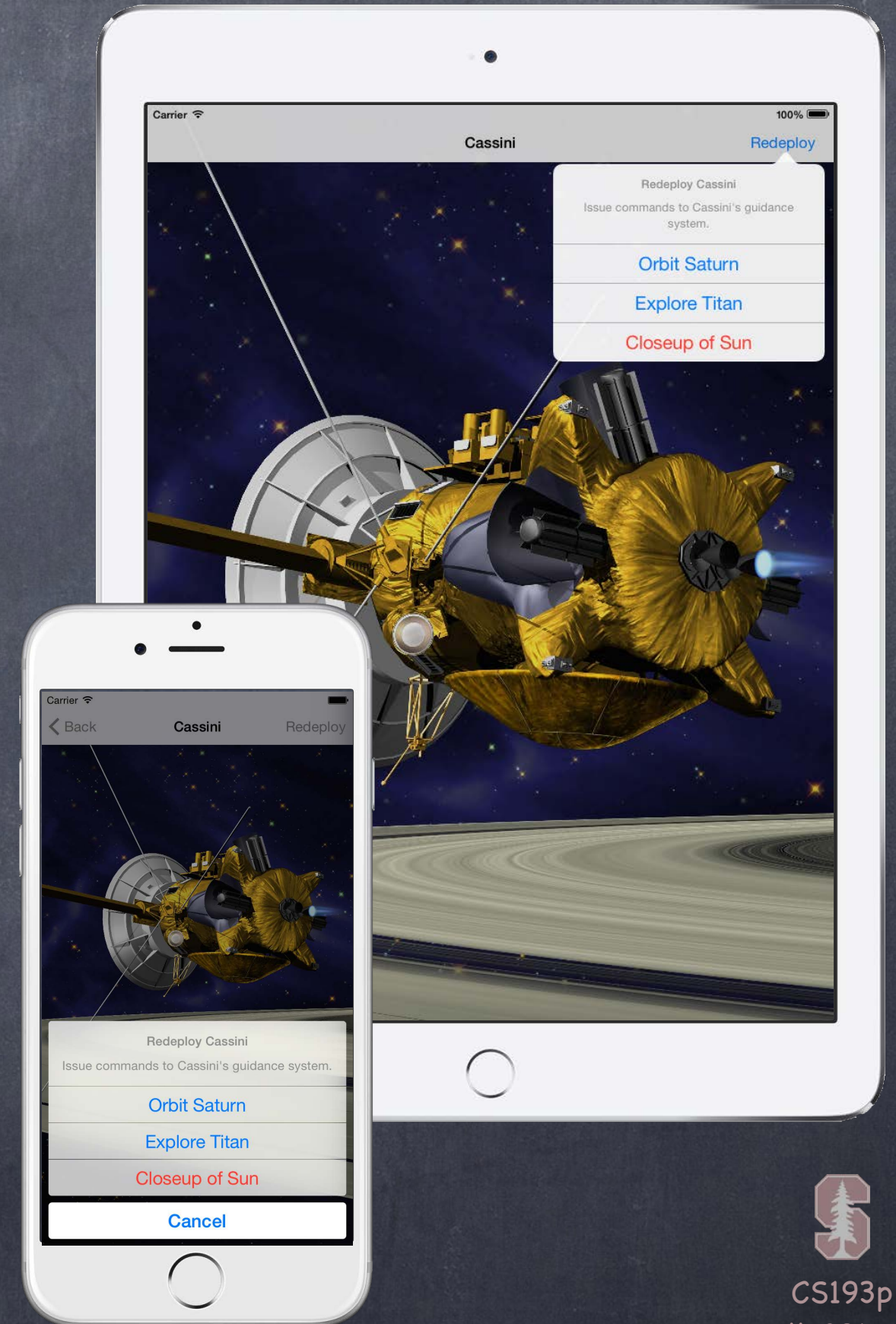


```
var alert = UIAlertController(  
    title: "Redeploy Cassini",  
    message: "Issue commands to Cassini's guidance system.",  
    preferredStyle: .actionSheet  
)  
  
alert.addAction(/* orbit saturn action */)   
alert.addAction(/* explore titan action */)   
alert.addAction(/* destroy with closeup of sun action */)   
alert.addAction(/* do nothing cancel action */)   
  
alert.modalPresentationStyle = .popover  
let ppc = alert.popoverPresentationController  
ppc?.barButtonItem = redeployBarButtonItem  
  
present(alert, animated: true, completion: nil)
```



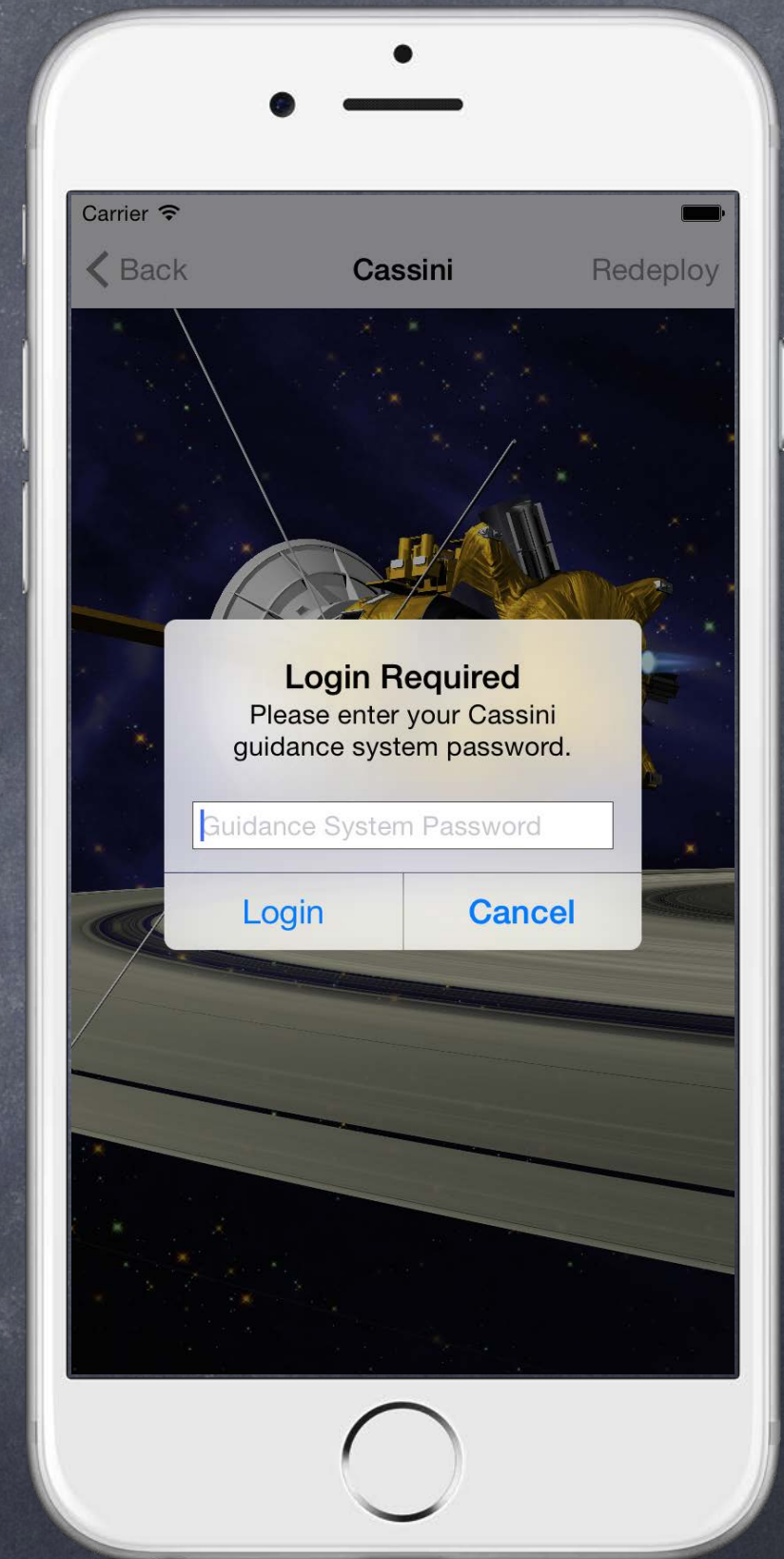


```
var alert = UIAlertController(  
    title: "Redeploy Cassini",  
    message: "Issue commands to Cassini's guidance system.",  
    preferredStyle: .actionSheet  
)  
  
alert.addAction(/* orbit saturn action */)   
alert.addAction(/* explore titan action */)   
alert.addAction(/* destroy with closeup of sun action */)   
alert.addAction(/* do nothing cancel action */)   
  
alert.modalPresentationStyle = .popover  
let ppc = alert.popoverPresentationController  
ppc?.barButtonItem = redeployBarButtonItem  
  
present(alert, animated: true, completion: nil)
```



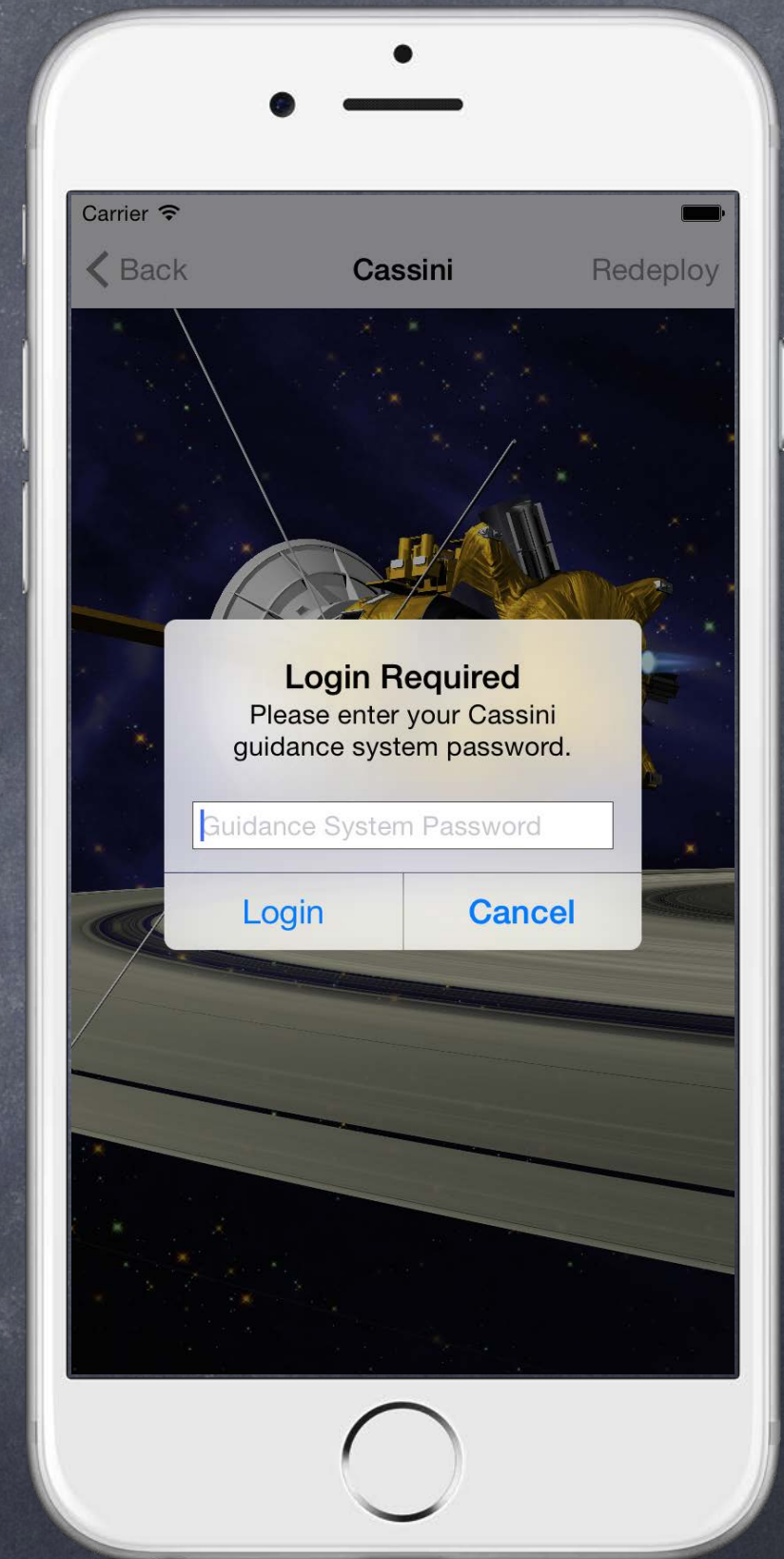


```
var alert = UIAlertController(  
    title: "Login Required",  
    message: "Please enter your Cassini guidance system...",  
    preferredStyle: .alert  
)  
  
alert.addAction(UIAlertAction(  
    title: "Cancel",  
    style: .cancel)  
    { (action: UIAlertAction) -> Void in  
        // do nothing  
    }  
)  
)
```





```
var alert = UIAlertController(  
    title: "Login Required",  
    message: "Please enter your Cassini guidance system...",  
    preferredStyle: .alert  
)  
  
alert.addAction(/* cancel button action */)  
  
alert.addAction(UIAlertAction(  
    title: "Login",  
    style: .default)  
    { (action: UIAlertAction) -> Void in  
        // get password and log in  
    }  
)  
}
```





```

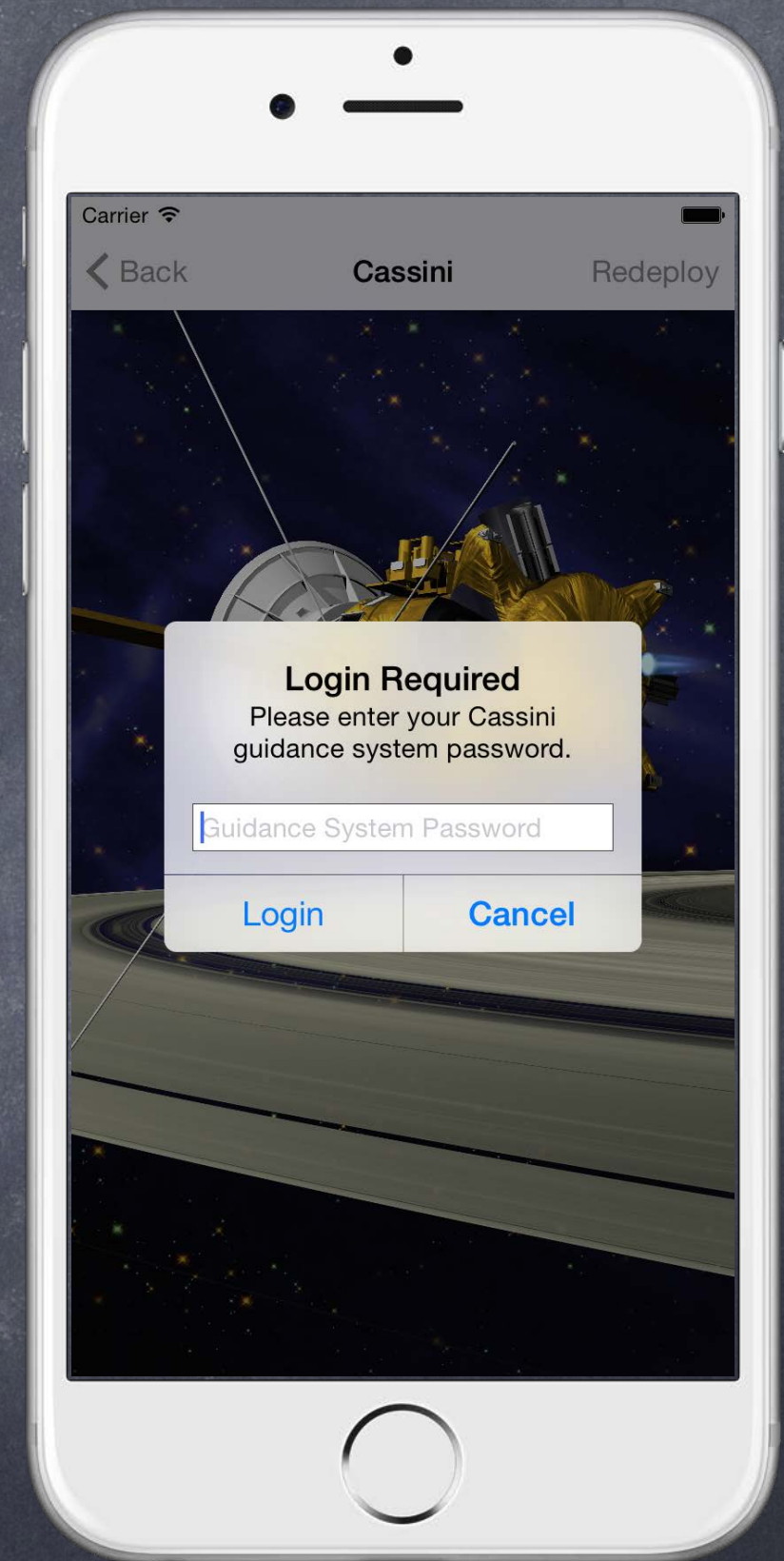
var alert = UIAlertController(
    title: "Login Required",
    message: "Please enter your Cassini guidance system...",
    preferredStyle: .alert
)

alert.addAction(/* cancel button action */)

alert.addAction(UIAlertAction(
    title: "Login",
    style: .default)
    { (action: UIAlertAction) -> Void in
        // get password and log in
    }
)

alert.addTextField(configurationHandler: { textField in
    textField.placeholder = "Guidance System Password"
    textField.isSecureTextEntry = true
})

```





```

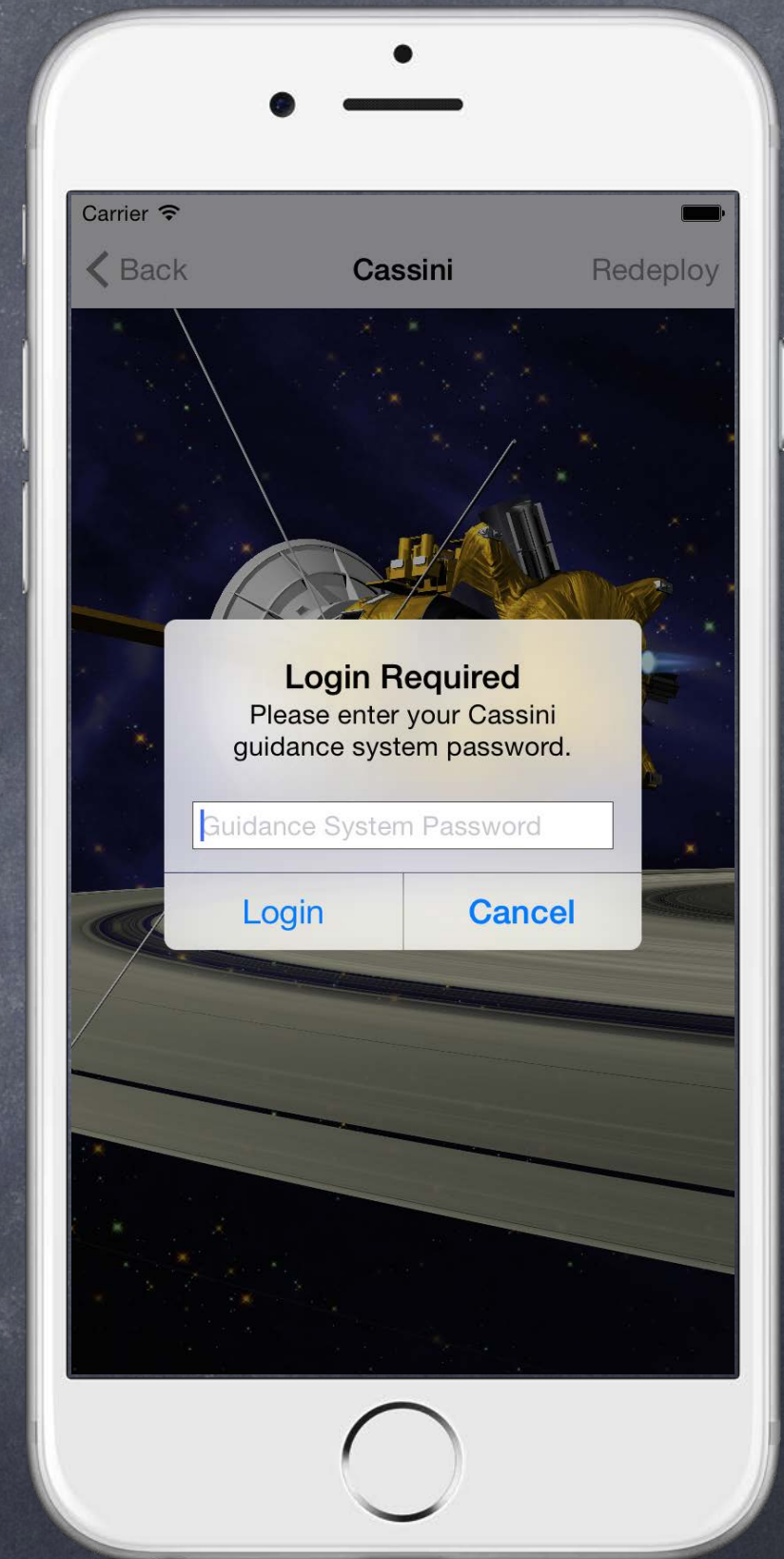
var alert = UIAlertController(
    title: "Login Required",
    message: "Please enter your Cassini guidance system...",
    preferredStyle: .alert
)

alert.addAction(/* cancel button action */)

alert.addAction(UIAlertAction(
    title: "Login",
    style: .default)
{ (action: UIAlertAction) -> Void in
    // get password and log in
    if let tf = self.alert.textFields?.first {
        self.loginWithPassword(tf.text)
    }
}
)

alert.addTextField(configurationHandler: { textField in
    textField.placeholder = "Guidance System Password"
    textField.isSecureTextEntry = true
})

```





```

var alert = UIAlertController(
    title: "Login Required",
    message: "Please enter your Cassini guidance system...",
    preferredStyle: .alert
)

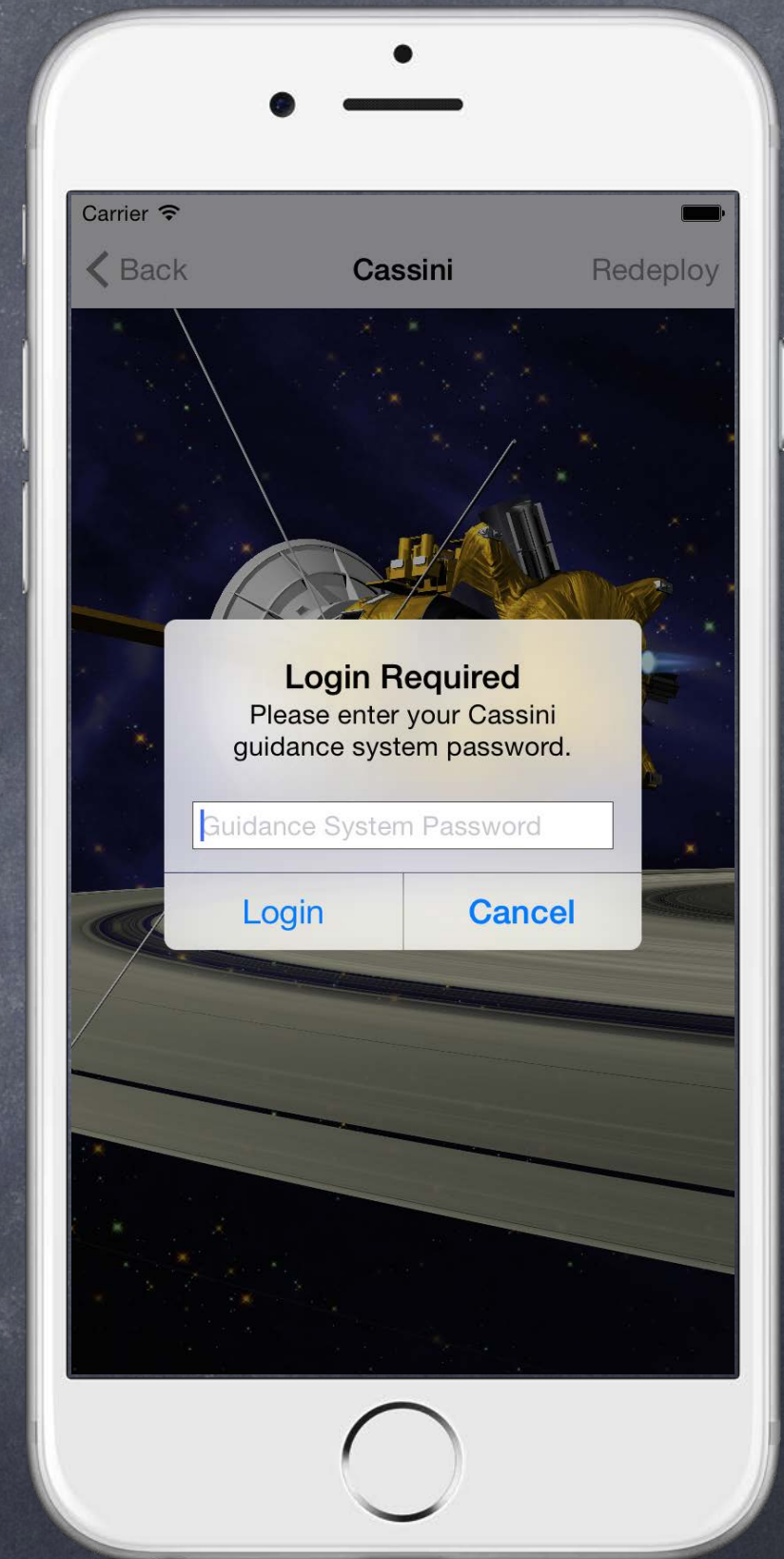
alert.addAction(/* cancel button action */)

alert.addAction(UIAlertAction(
    title: "Login",
    style: .default)
    { (action: UIAlertAction) -> Void in
        // get password and log in
        if let tf = self.alert.textFields?.first {
            self.loginWithPassword(tf.text)
        }
    }
)

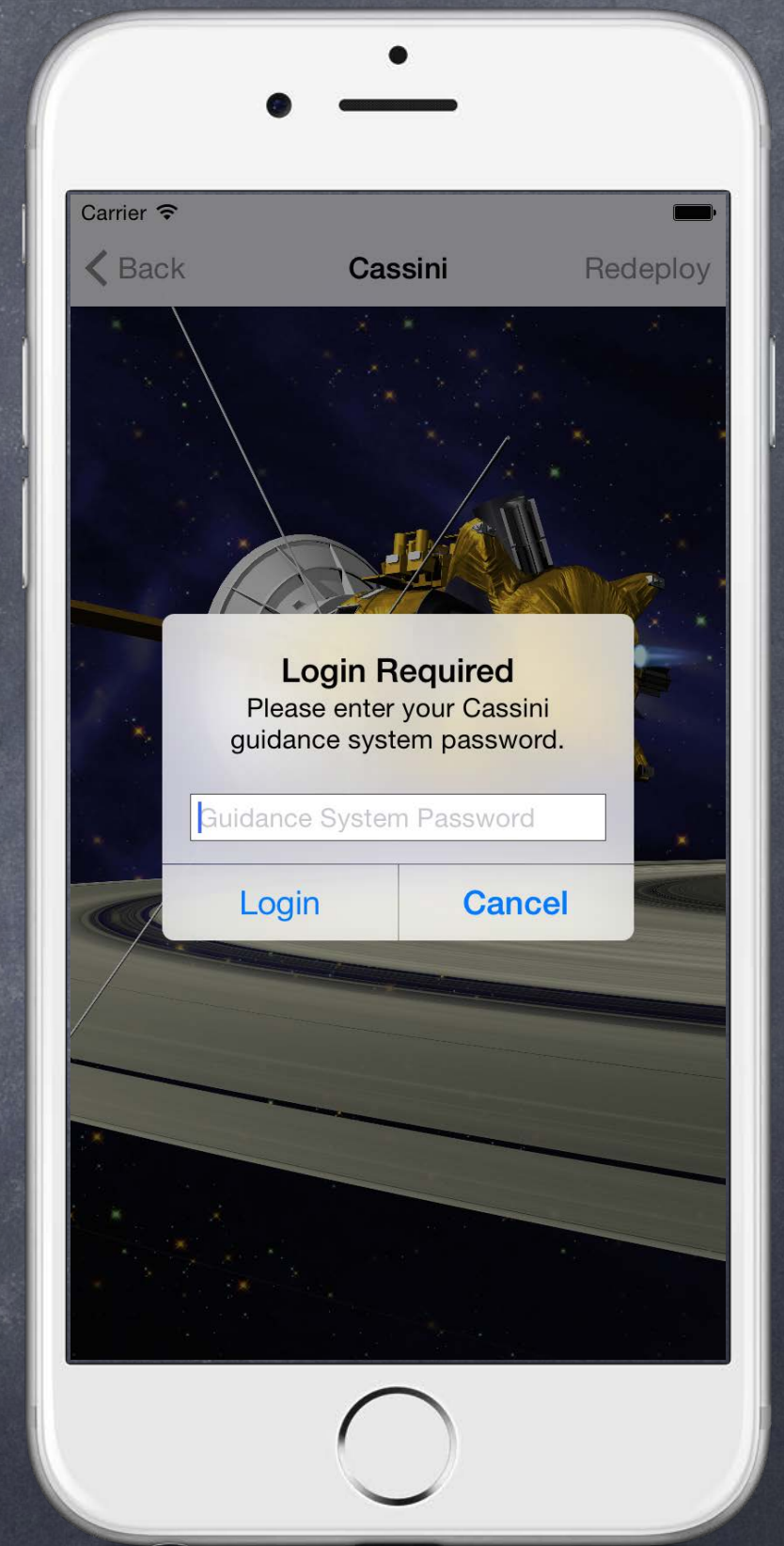
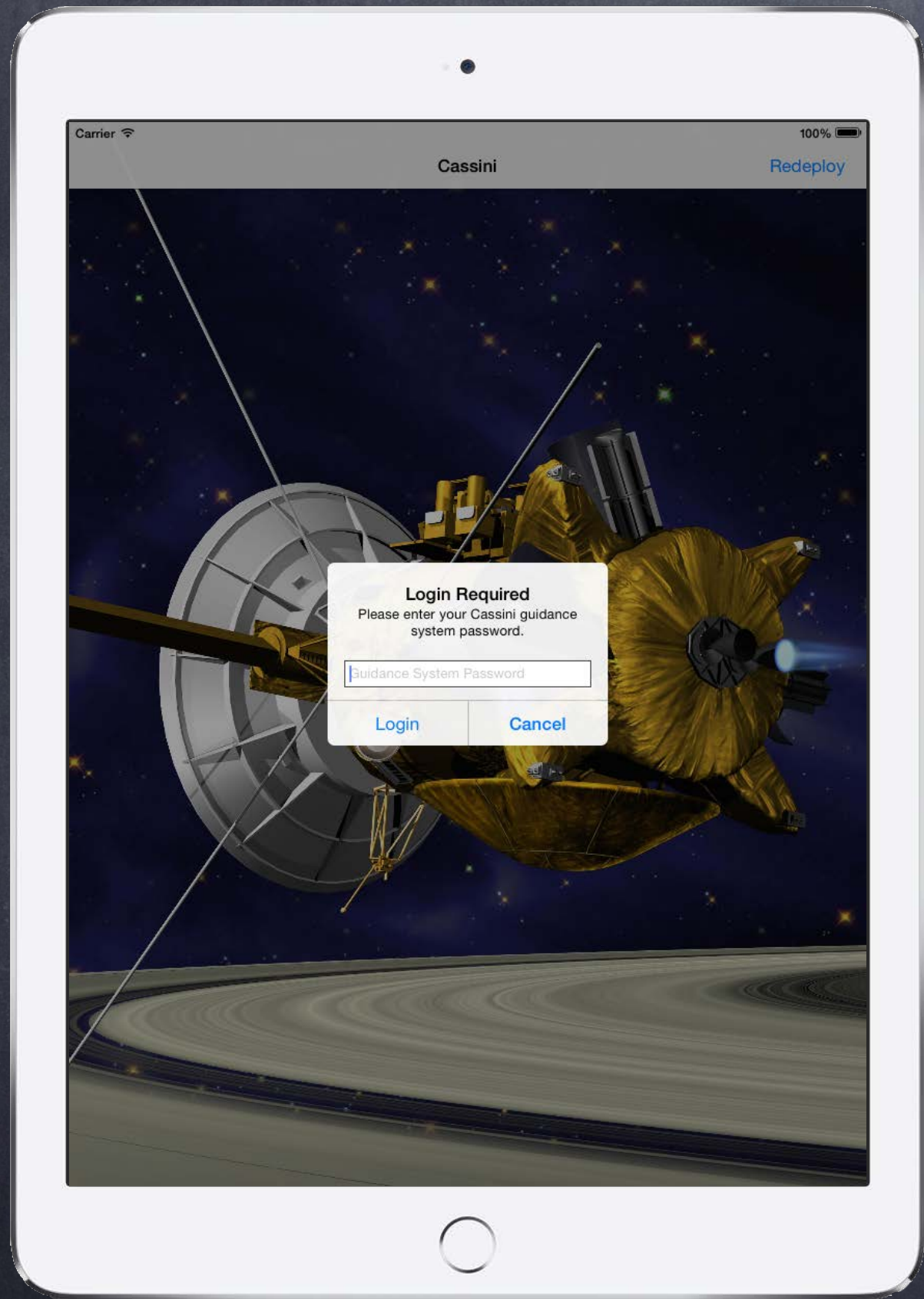
alert.addTextField(configurationHandler: { textField in
    textField.placeholder = "Guidance System Password"
    textField.isSecureTextEntry = true
})

present(alert, animated: true, completion: nil)

```









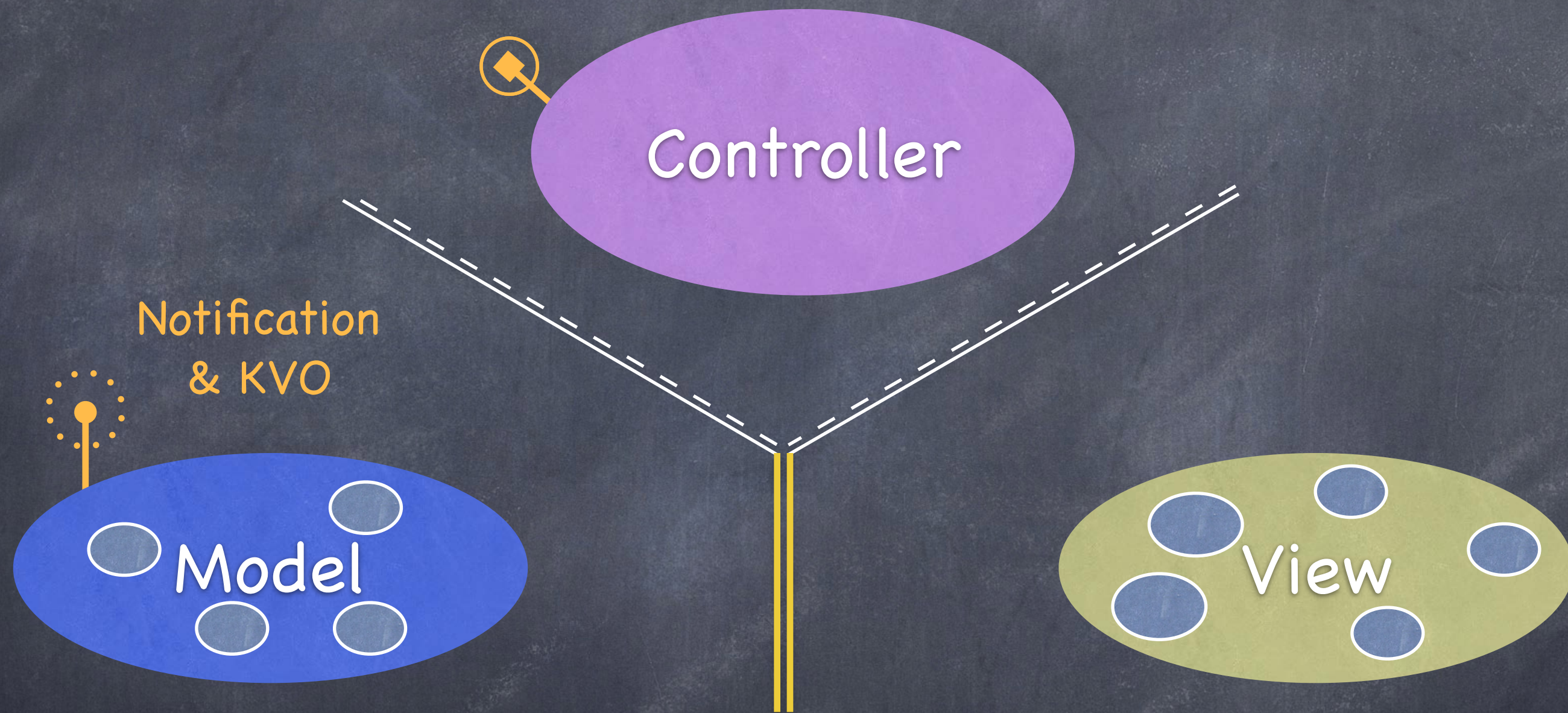
# Demo

## 👁 Alerts

Report bad dropped background images to the user in EmojiArt







## Radio Station Communication





# Notification

## • Notifications

The “radio station” from the MVC slides. For Model (or global) to Controller communication.

## • NotificationCenter

Get the default “notification center” via `NotificationCenter.default`

Then send it the following message if you want to “listen to a radio station” ...

```
var observer: NSObjectProtocol? // a cookie to later “stop listening” with
observer = NotificationCenter.default.addObserver(
    forName: Notification.Name, // the name of the radio station
    object: Any?, // the broadcaster (or nil for “anyone”)
    queue: OperationQueue? // the queue on which to dispatch the closure below
) { (notification: Notification) -> Void in // closure executed when broadcasts occur
    let info: Any? = notification.userInfo
    // info is usually a dictionary of notification-specific information
}
```





# Notification

## • What is `Notification.Name`?

Look this up in the documentation to see what iOS system radio stations you can listen to. There are a lot.

You will see them as static vars on `Notification.Name`.

You can make your own radio station name with `Notification.Name(String)`.

More on broadcasting on your own station in a couple of slides ...



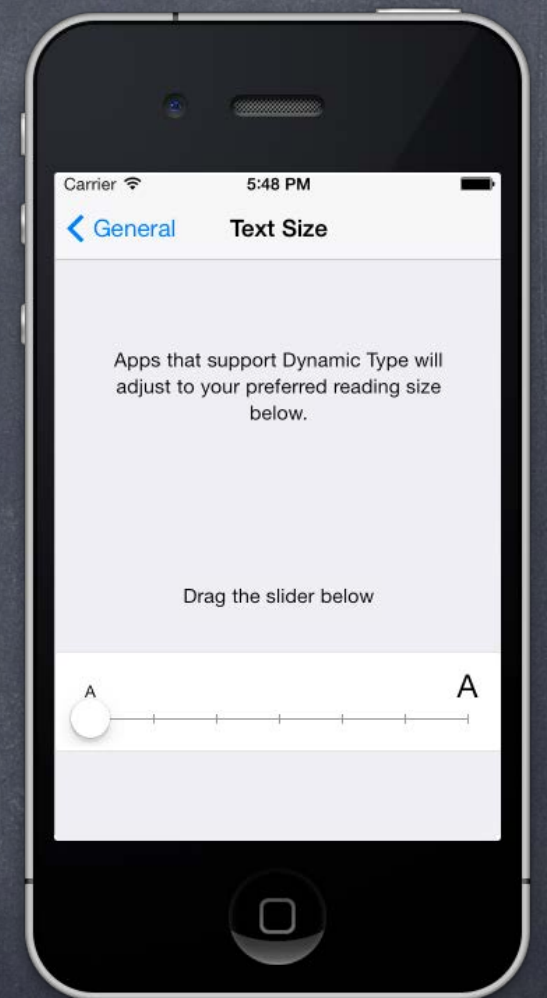


# Notification

## • Example of listening to “radio station broadcasts”

Watching for changes in the size of preferred fonts (user can change this in Settings) ...

```
let center = NotificationCenter.default
var observer = center.addObserver(
    forName: Notification.Name.UIContentSizeCategoryDidChange
    object: UIApplication.shared, // or nil
    queue: OperationQueue.main // or nil
) { notification in
    // re-set the fonts of objects using preferred fonts
    // or look at the size category and do something with it ...
    let c = notification.userInfo?[UIContentSizeCategoryNewValueKey]
    // c might be UIContentSizeCategorySmall, for example
}
center.removeObserver(observer) // when you're done listening
```





# Notification

## • Posting a Notification

```
NotificationCenter.default.post(  
    name: Notification.Name,           // name of the "radio station"  
    object: Any?,                     // who is sending this notification (usually self)  
    userInfo: [AnyHashable:Any]? = nil // any info you want to pass to station listeners  
)
```

Any closures added with addObserver will be executed.

Either immediately on the same queue as post (if queue was nil).

Or asynchronously by posting the block onto the queue specified with addObserver.





# KVO

## • Watching the properties of NSObject subclasses

The basic idea of KVO is to register a closure to invoke when a property value changes

There is some “mechanism” required to make this work

We’re not going to talk about that, but NSObject implements this mechanism

Thus objects that inherit from NSObject can participate

## • What’s it good for?

Usually used by a Controller to observe either its Model or its View

Not every property works with KVO

A property has to be Key Value Coding-compliant to work

There are a few properties scattered throughout the iOS frameworks that are compliant

For example, UIView’s frame and center work with KVO

So does most of CALayer underneath UIView

Of course, you can make properties in your own NSObject subclasses compliant

(though we don’t have time to talk about how to do any of that right now)

You’re unlikely to use KVO much, but it’s something that’s good to know it exists





# KVO

## • How does it work?

```
var observation = observed.observe(keyPath: KeyPath) { (observed, change) in
    // code to execute when the property described by keyPath changes
}
```

As long as the `observation` remains in the heap, the closure will stay active.

The `change` argument to the closure is an `NSKeyValueObservedChange`.

`NSKeyValueObservedChange` has the old value and the new value in it.

The syntax for a `KeyPath` is `\Type.property` or even `\Type.prop1.prop2.prop3`.

Swift can infer the `Type` (since that `Type` has to make sense for `observed`).





# Demo

## 👁 Notifications and KVO in EmojiArt

Track changes in our document state.

After lecture, I added code to `EmojiArtView` to let its Controller know about changes that was done using Delegation (i.e. I added an `EmojiArtViewDelegate`)

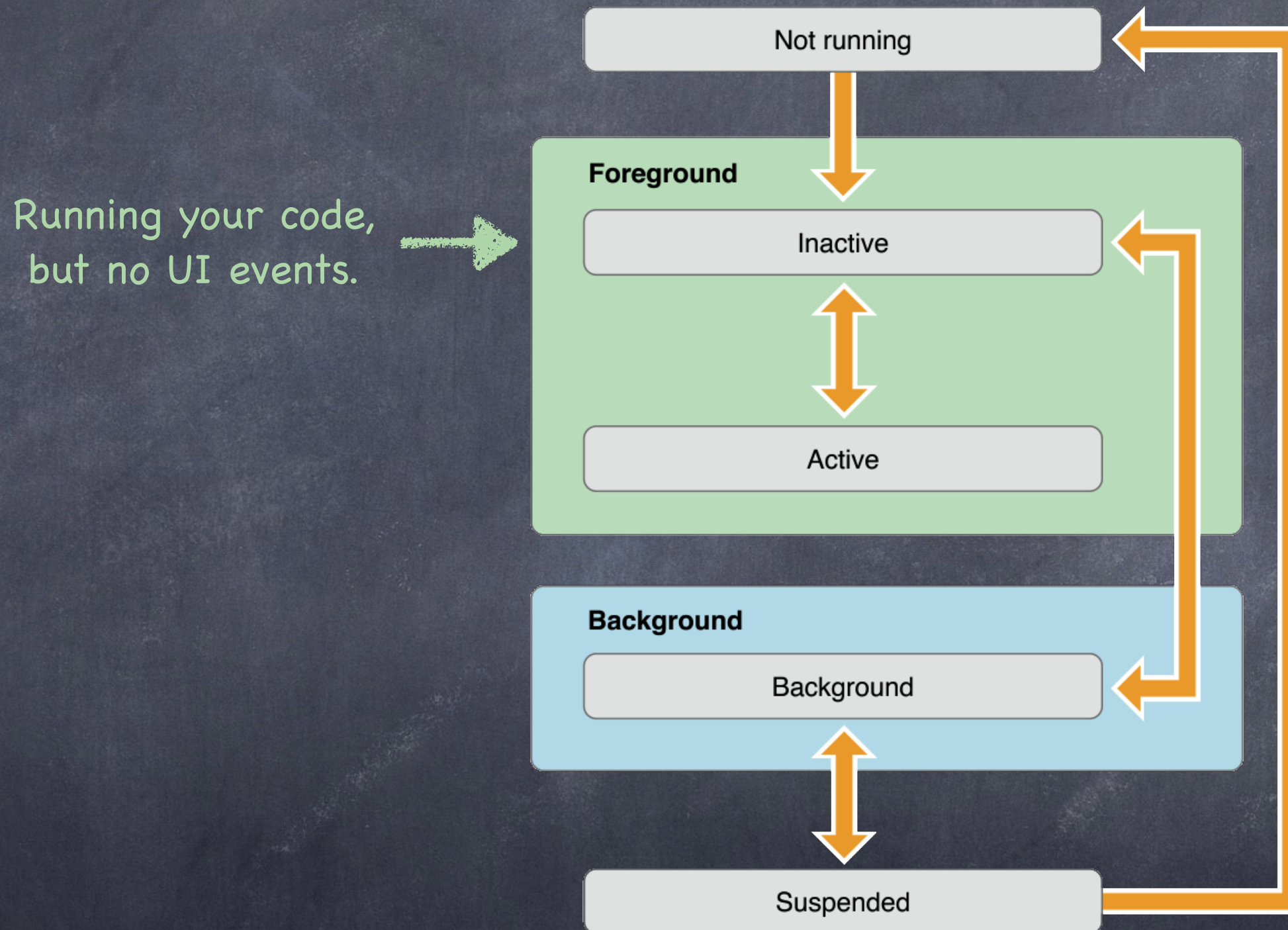
This required some code in the gesture-recognizing code to notify delegate of changes

1. We can remove that change-tracking code by using KVO (on the emojis' positions)
2. Then we can use `NotificationCenter` as a replacement for Delegation entirely



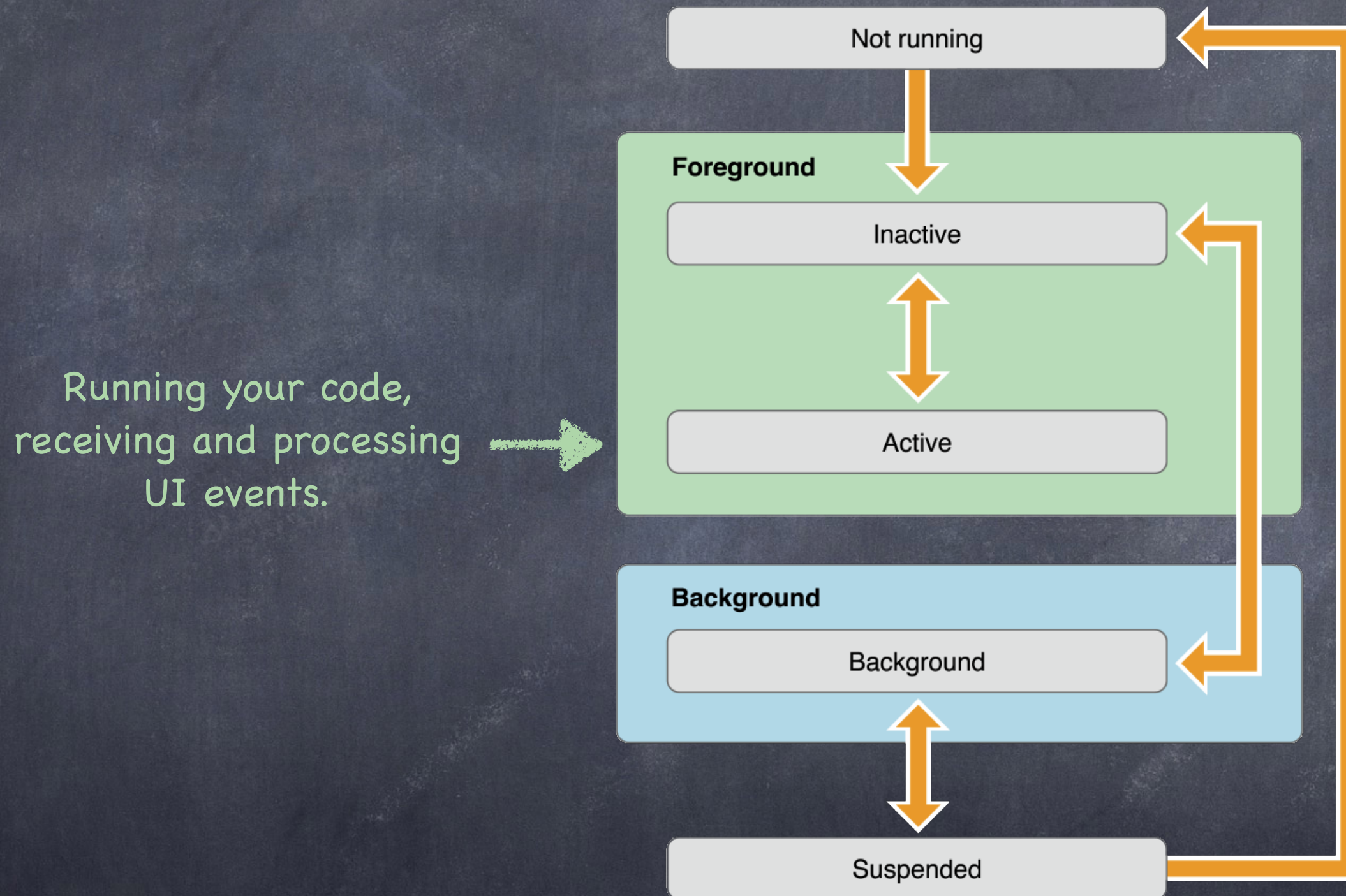


# Application Lifecycle



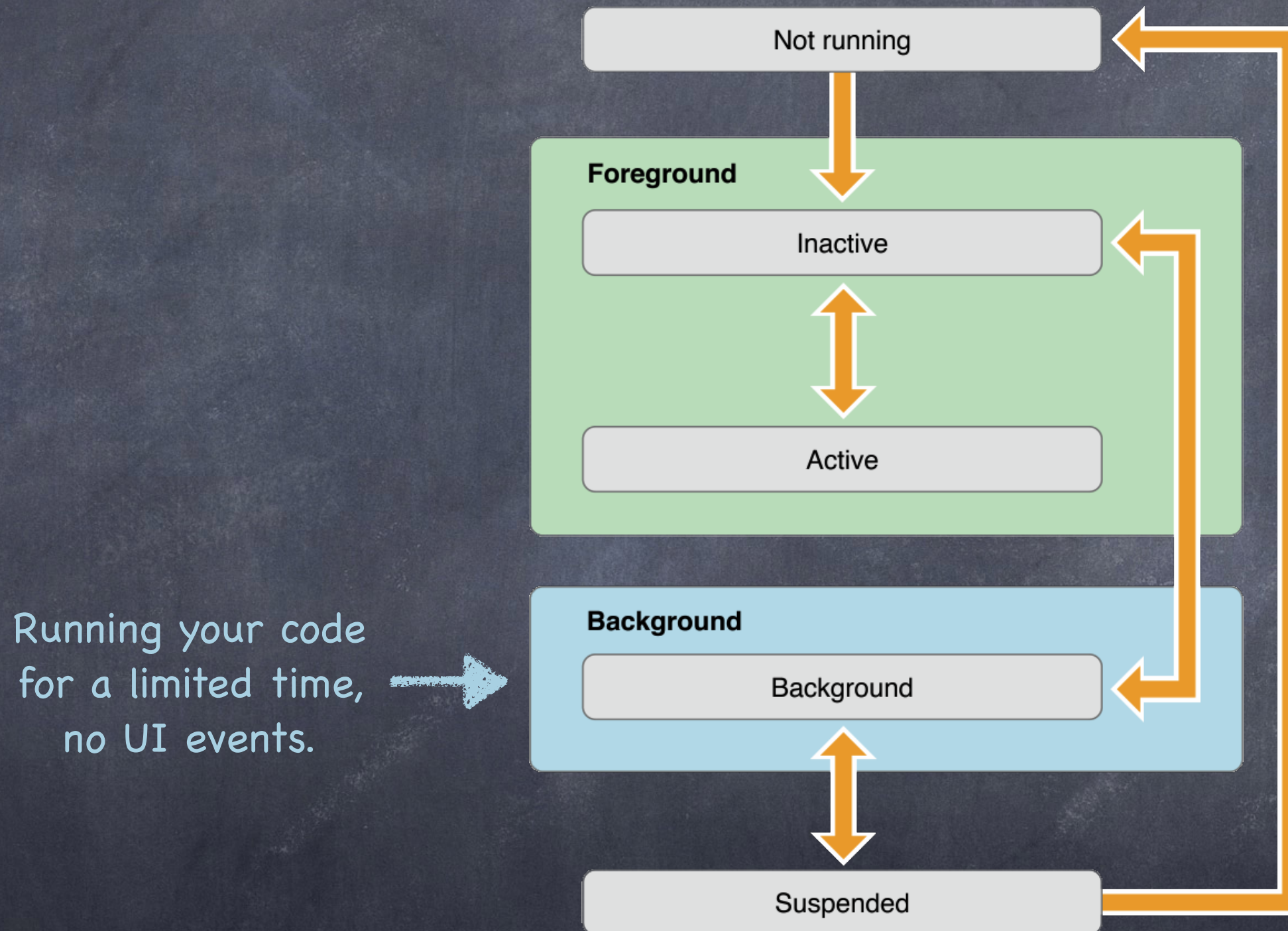


# Application Lifecycle



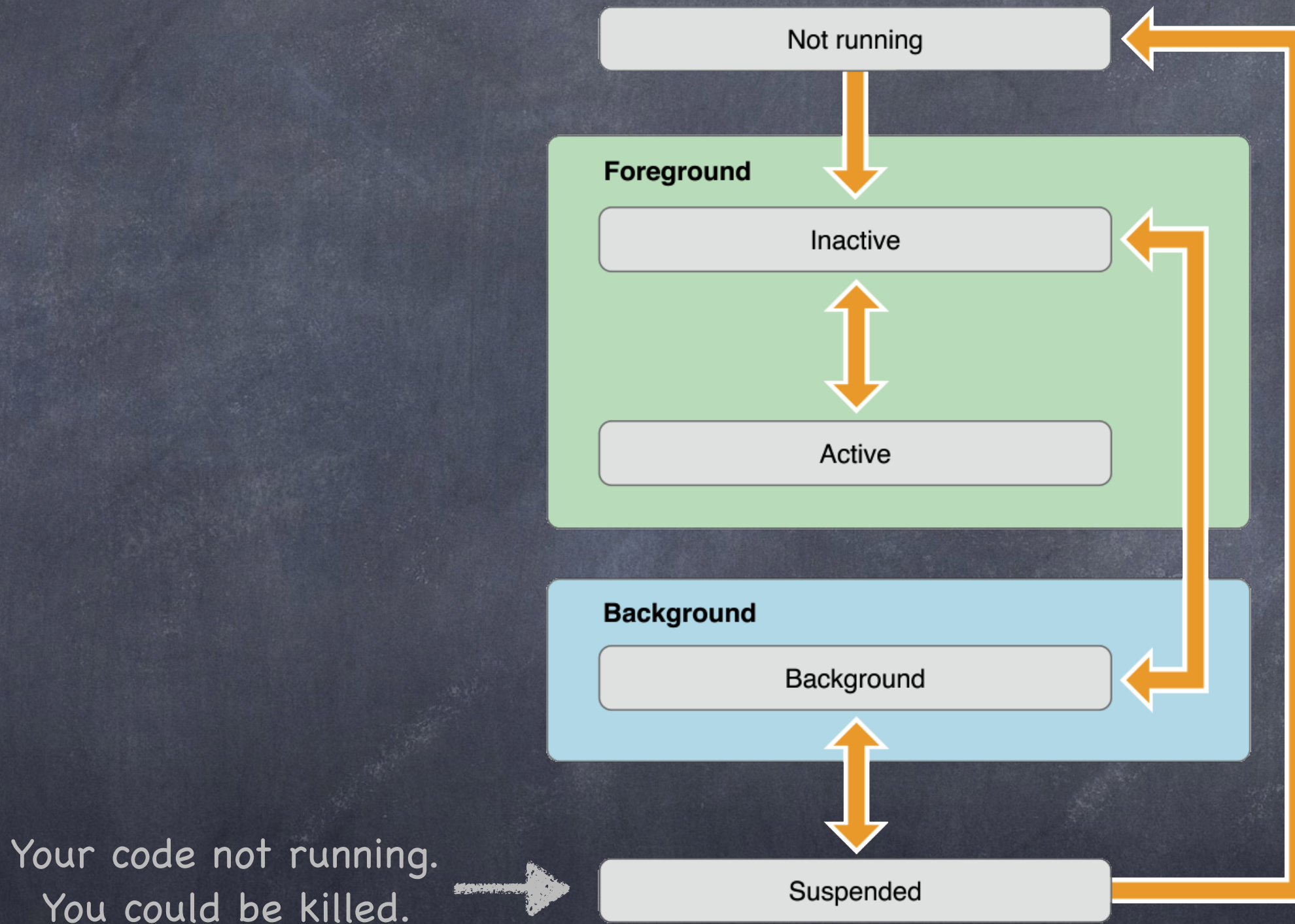


# Application Lifecycle



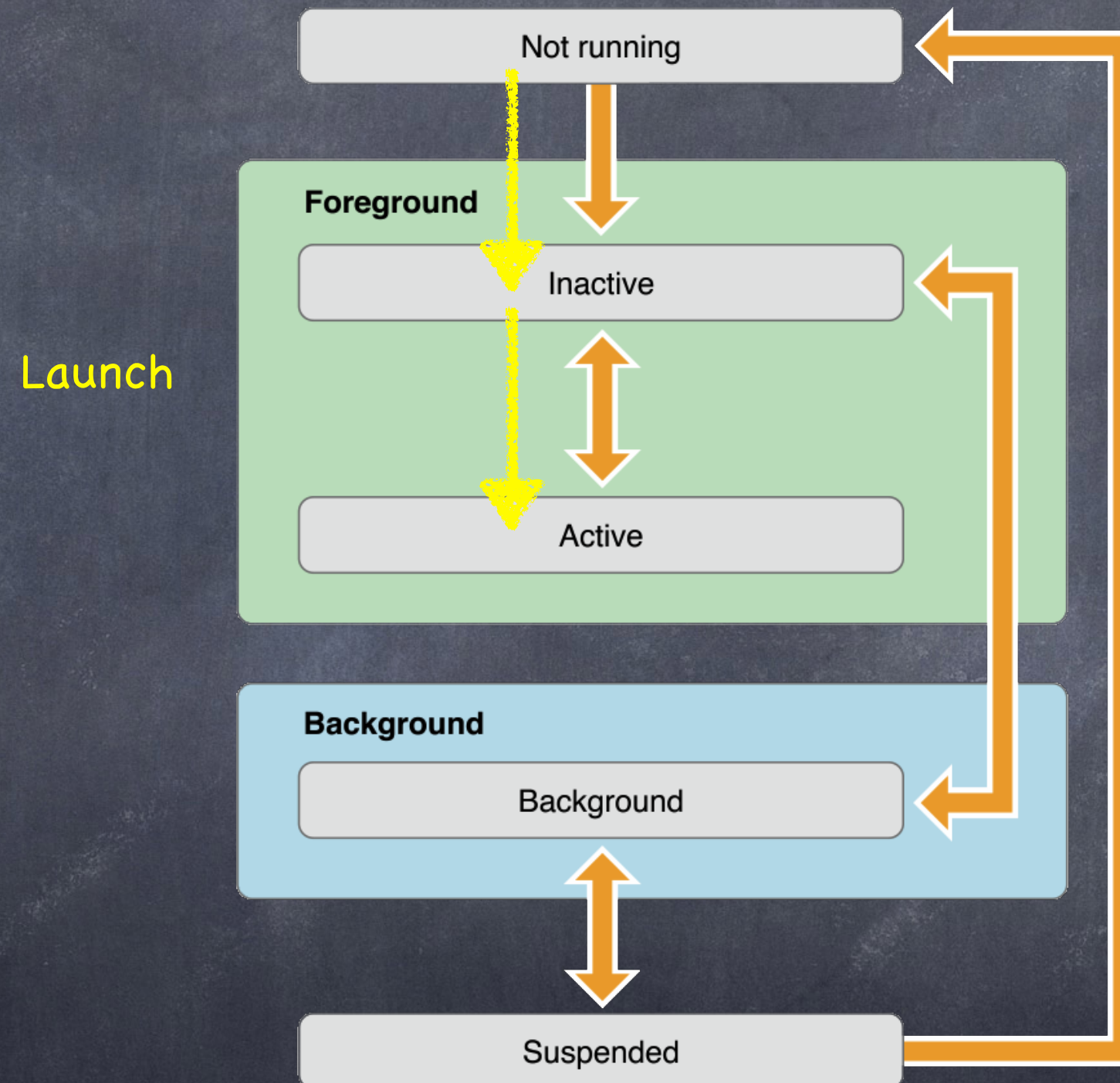


# Application Lifecycle



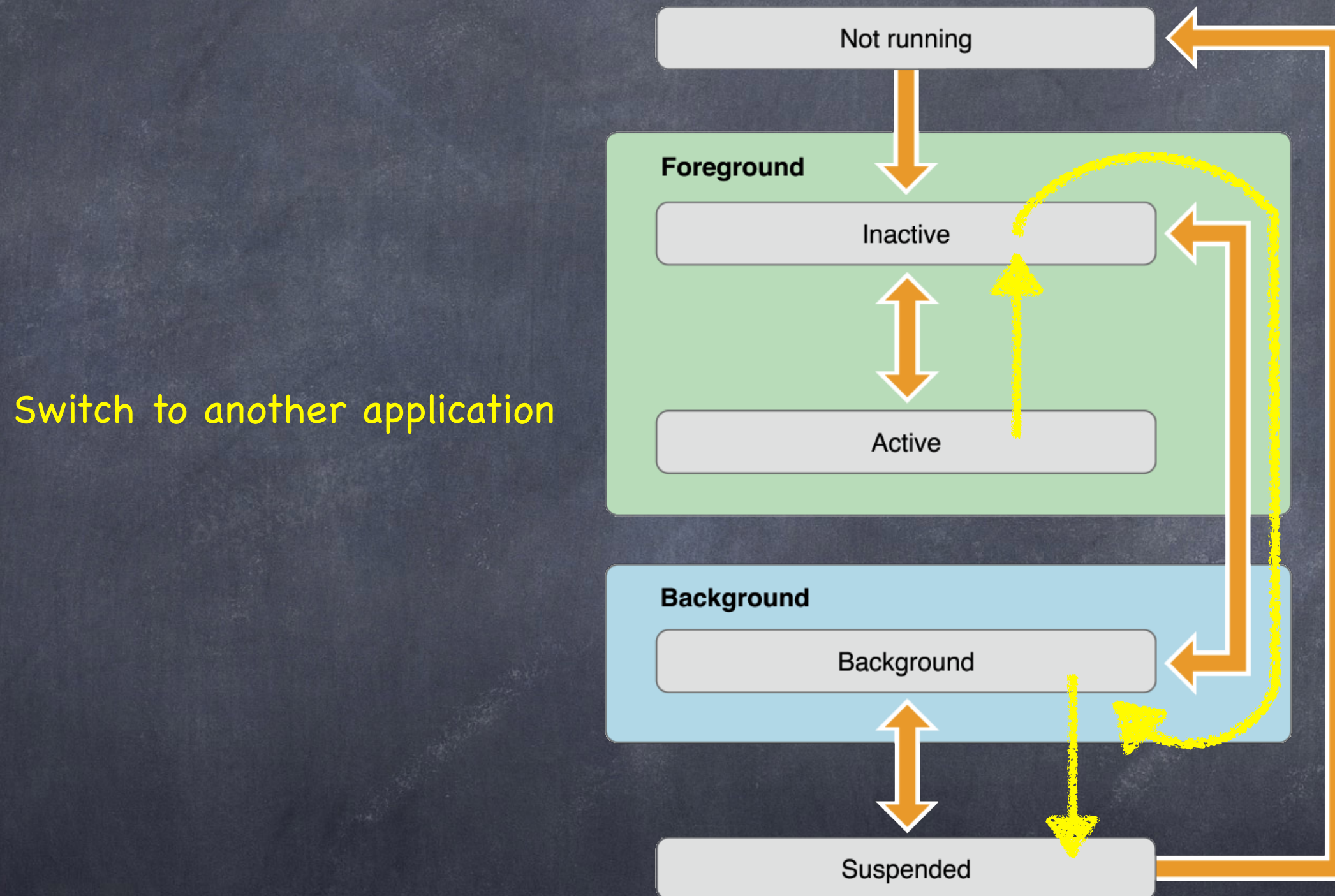


# Application Lifecycle



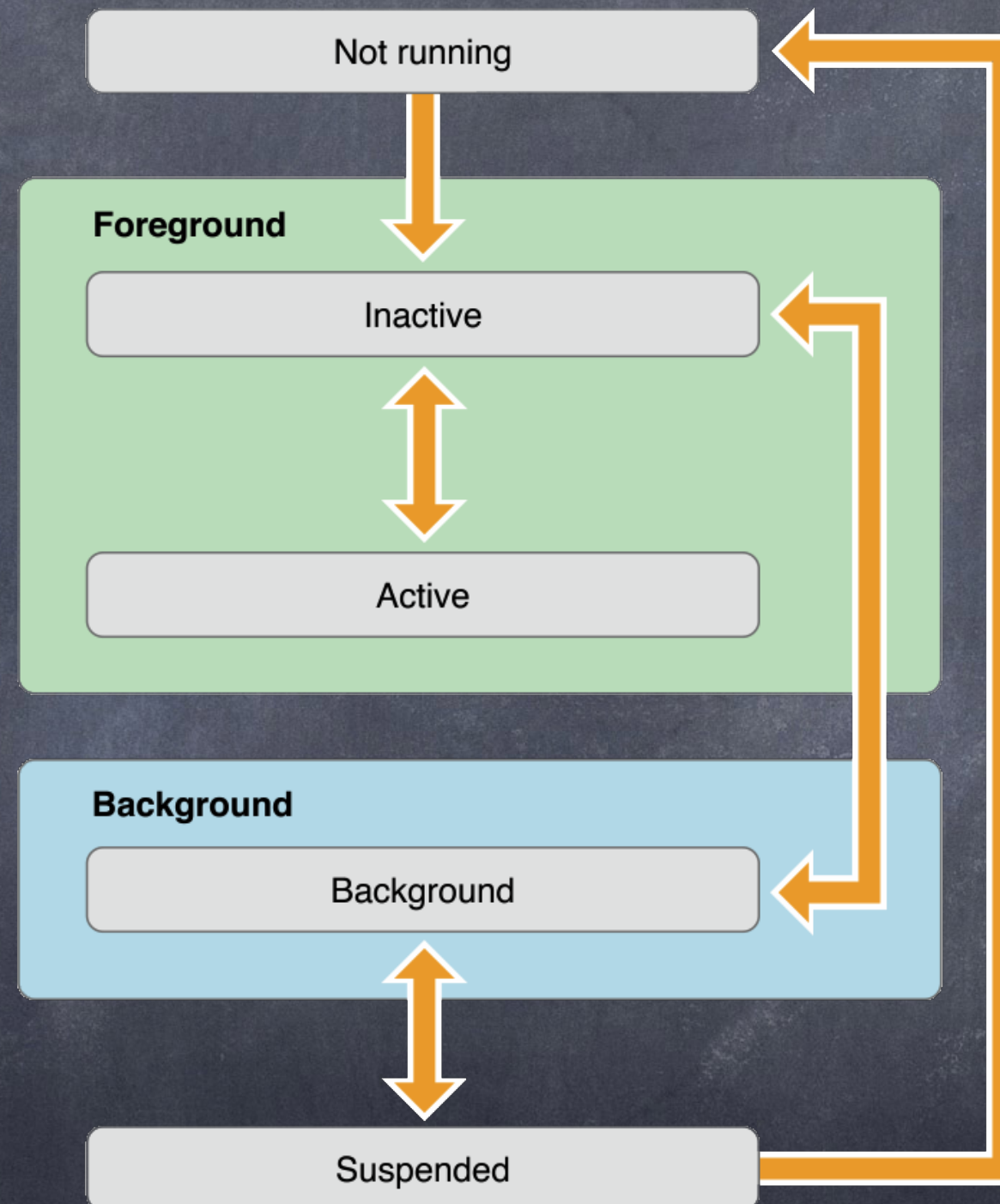


# Application Lifecycle





# Application Lifecycle



↑  
**Killed**  
(notice no code runs  
between suspended  
and killed)





# Application Lifecycle

Your AppDelegate will receive ...

```
func application(UIApplication,  
    will/didFinishLaunchingWithOptions:  
    [UIApplicationLaunchOptionsKey:Any]? = nil)
```

... and you can observe ...

`UIApplicationDidFinishLaunching`

The passed dictionary (also in `notification.userInfo`) tells you why your application was launched.

Some examples ...

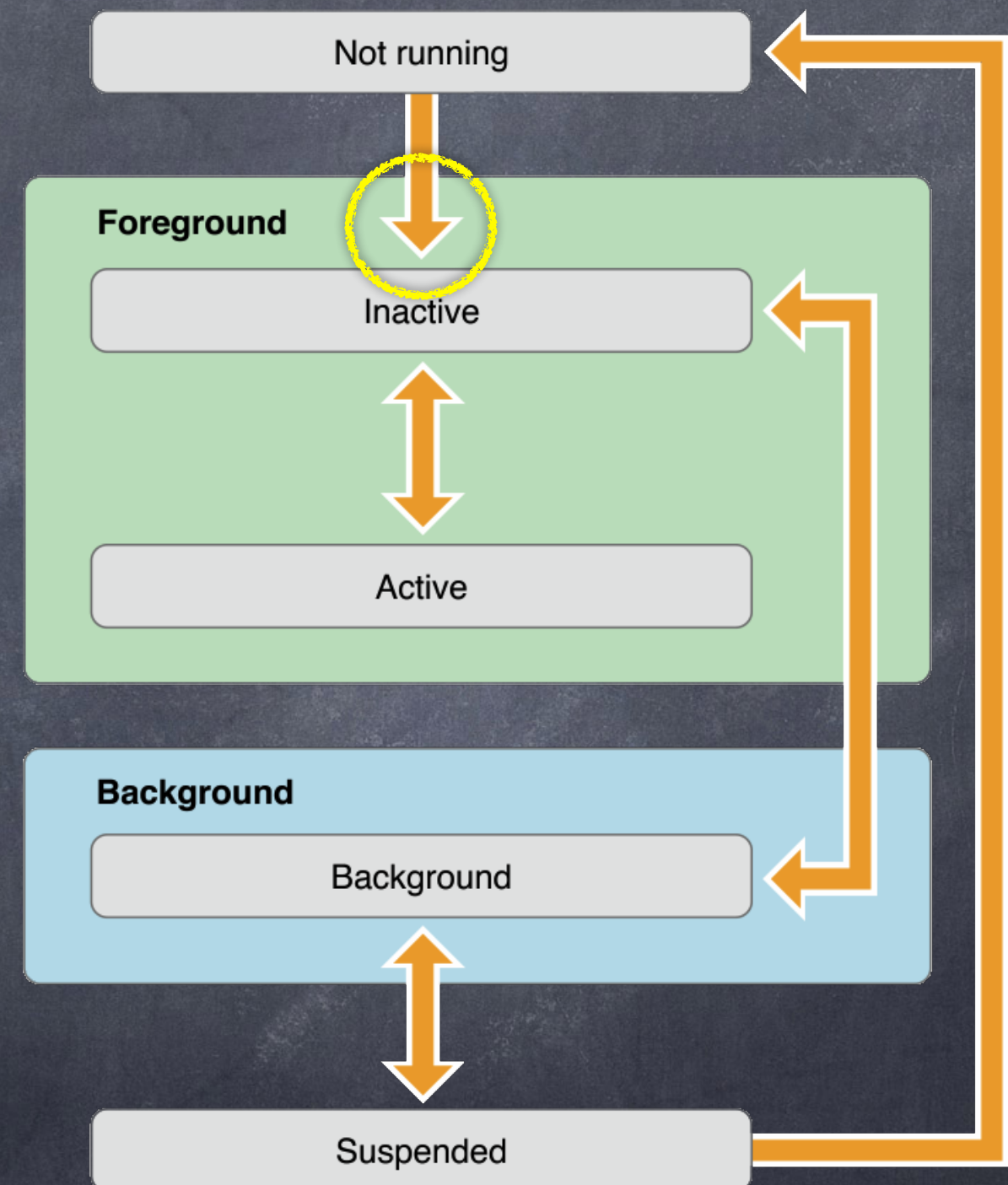
Someone wants you to open a URL

You entered a certain place in the world

You are continuing an activity started on another device

A notification arrived for you (push or local)

Bluetooth attached device wants to interact with you





# Application Lifecycle

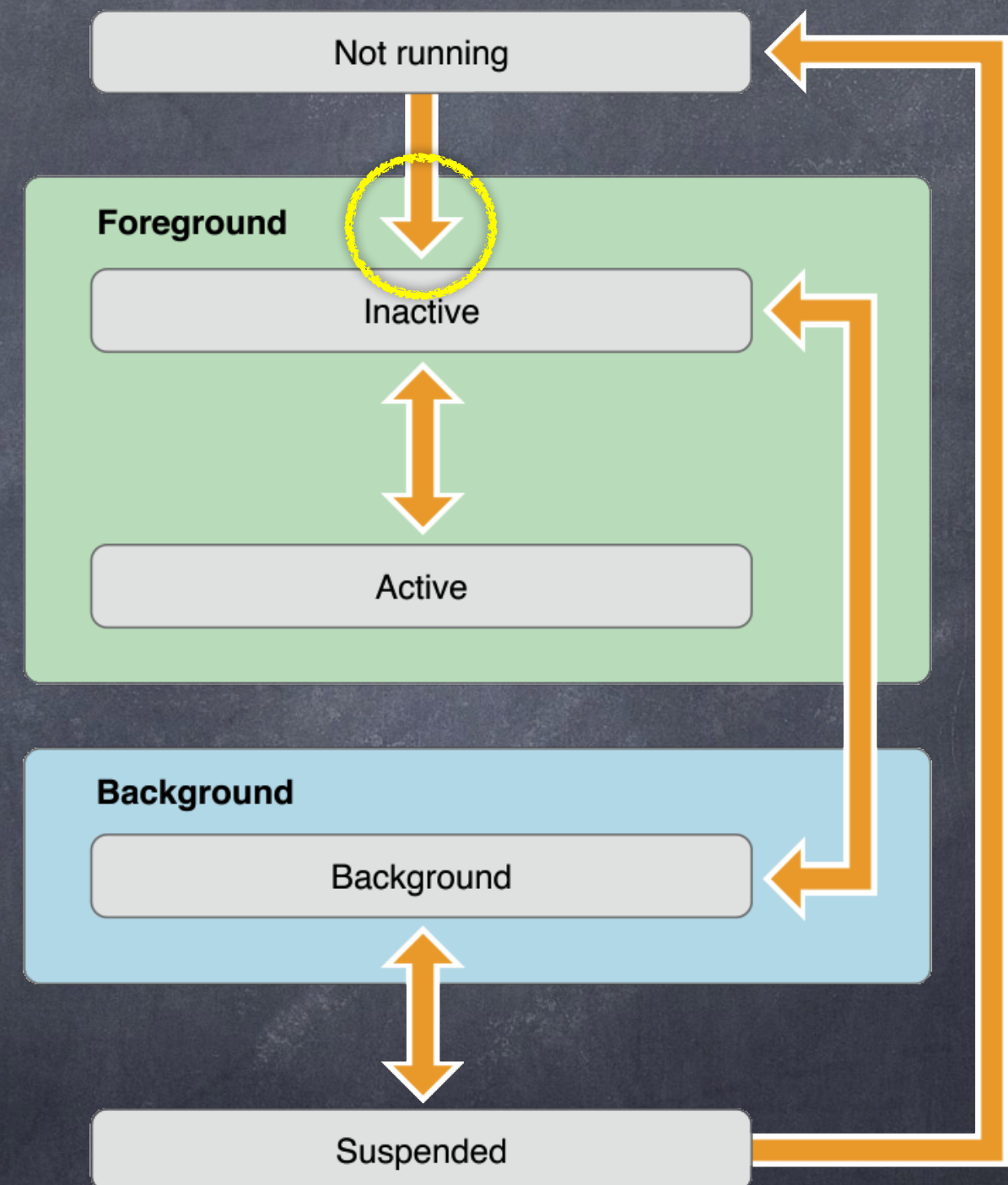
Your AppDelegate will receive ...

```
func application(UIApplication,  
    will/didFinishLaunchingWithOptions:  
    [UIApplicationLaunchOptionsKey:Any]? = nil)
```

... and you can observe ...

`UIApplicationDidFinishLaunching`

It used to be that you would build your UI here.  
For example, you'd instantiate a split view controller  
and put a navigation controller inside, then push  
your actual content view controller.  
But nowadays we use storyboards for all that.  
So often you do not implement this method at all.





# Application Lifecycle

Your AppDelegate will receive ...

```
func applicationWillResignActive(UIApplication)
```

... and you can observe ...

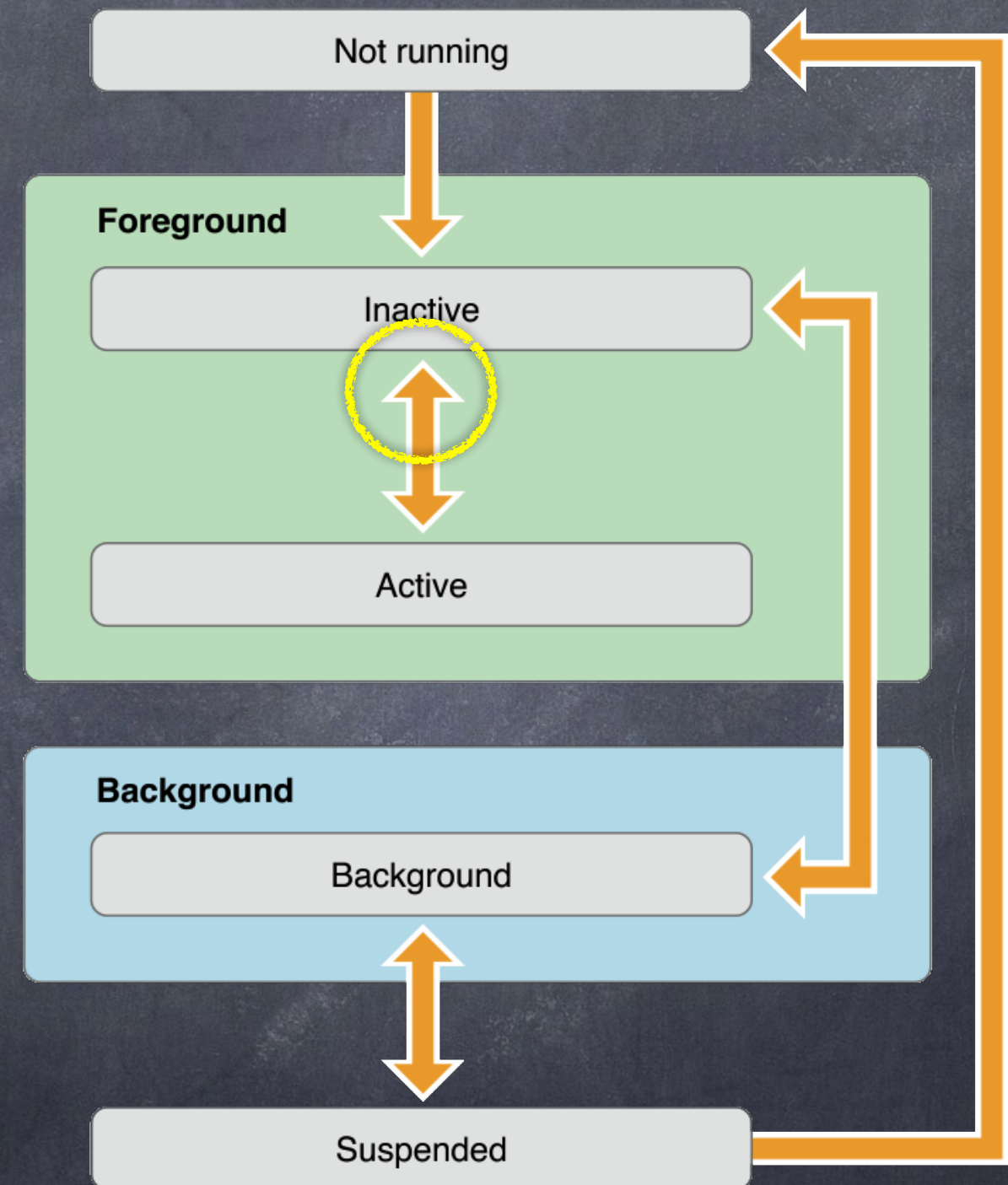
```
UIApplicationWillResignActive
```

You will want to “pause” your UI here.

For example, Asteroids would want to pause the asteroids.

This might happen because a phone call comes in.

Or you might be on your way to the background.





# Application Lifecycle

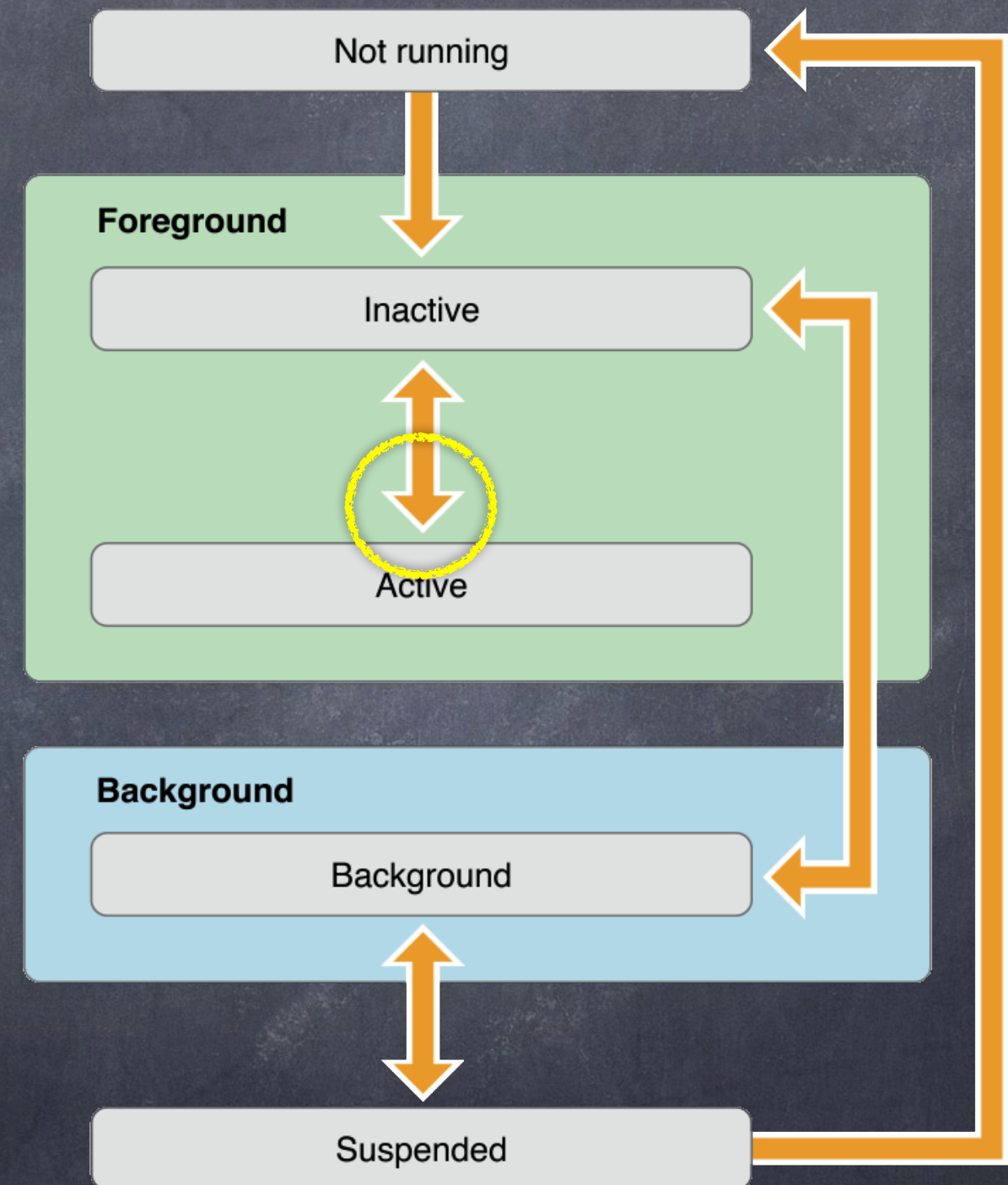
Your AppDelegate will receive ...

```
func applicationDidBecomeActive(UINavigationController)
```

... and you can observe ...

```
UIApplicationDidBecomeActive
```

If you have "paused" your UI previously  
here's where you would reactivate things.





# Application Lifecycle

Your AppDelegate will receive ...

```
func applicationDidEnterBackground(UIApplication)
```

... and you can observe ...

```
UIApplicationDidEnterBackground
```

Here you want to (quickly) batten down the hatches.

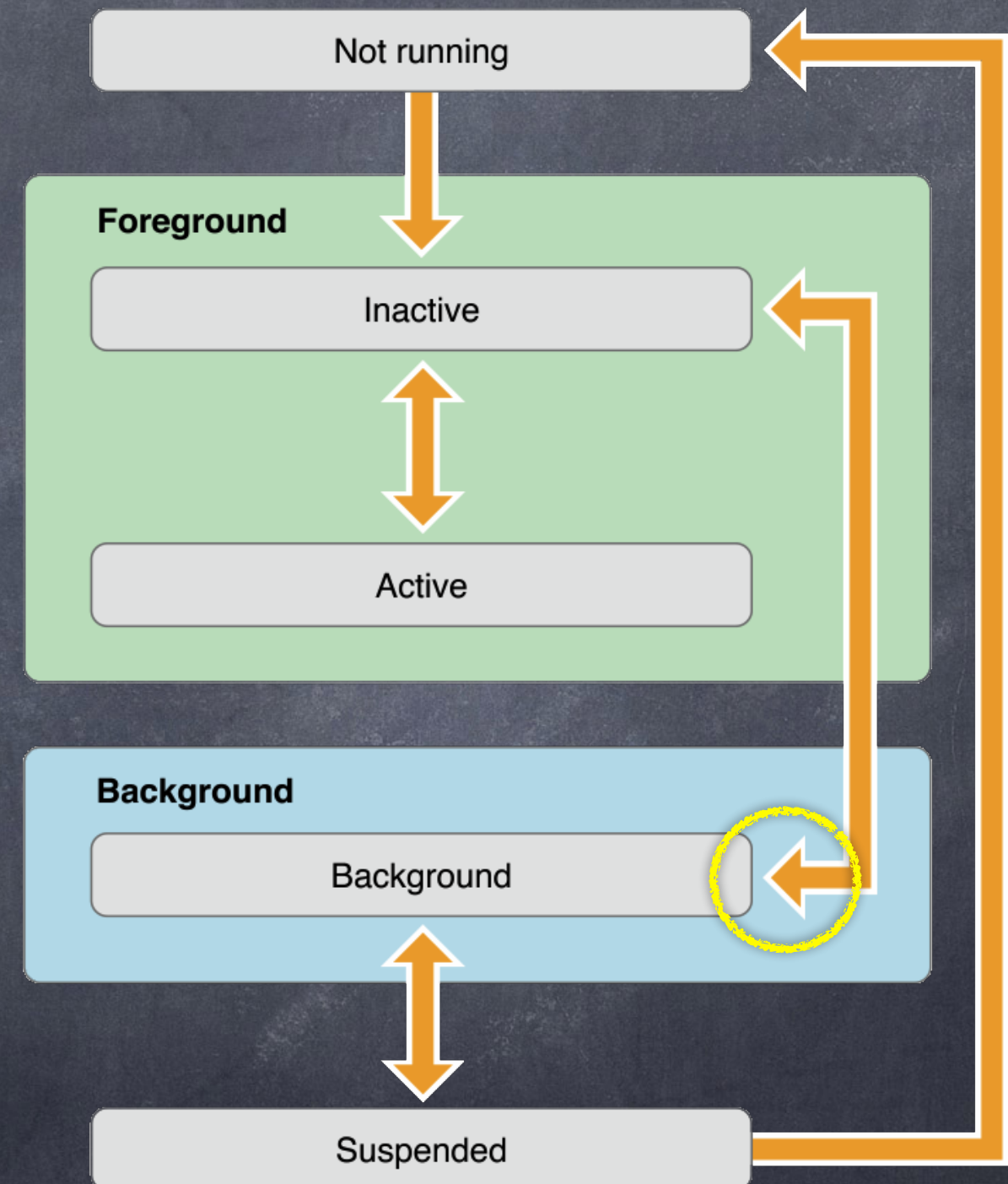
You only get to run for 30s or so.

You can request more time, but don't abuse this

(or the system will start killing you instead).

Prepare yourself to be eventually killed here

(probably won't happen, but be ready anyway).





# Application Lifecycle

Your AppDelegate will receive ...

```
func applicationWillEnterForeground(UINavigationController)
```

... and you can observe ...

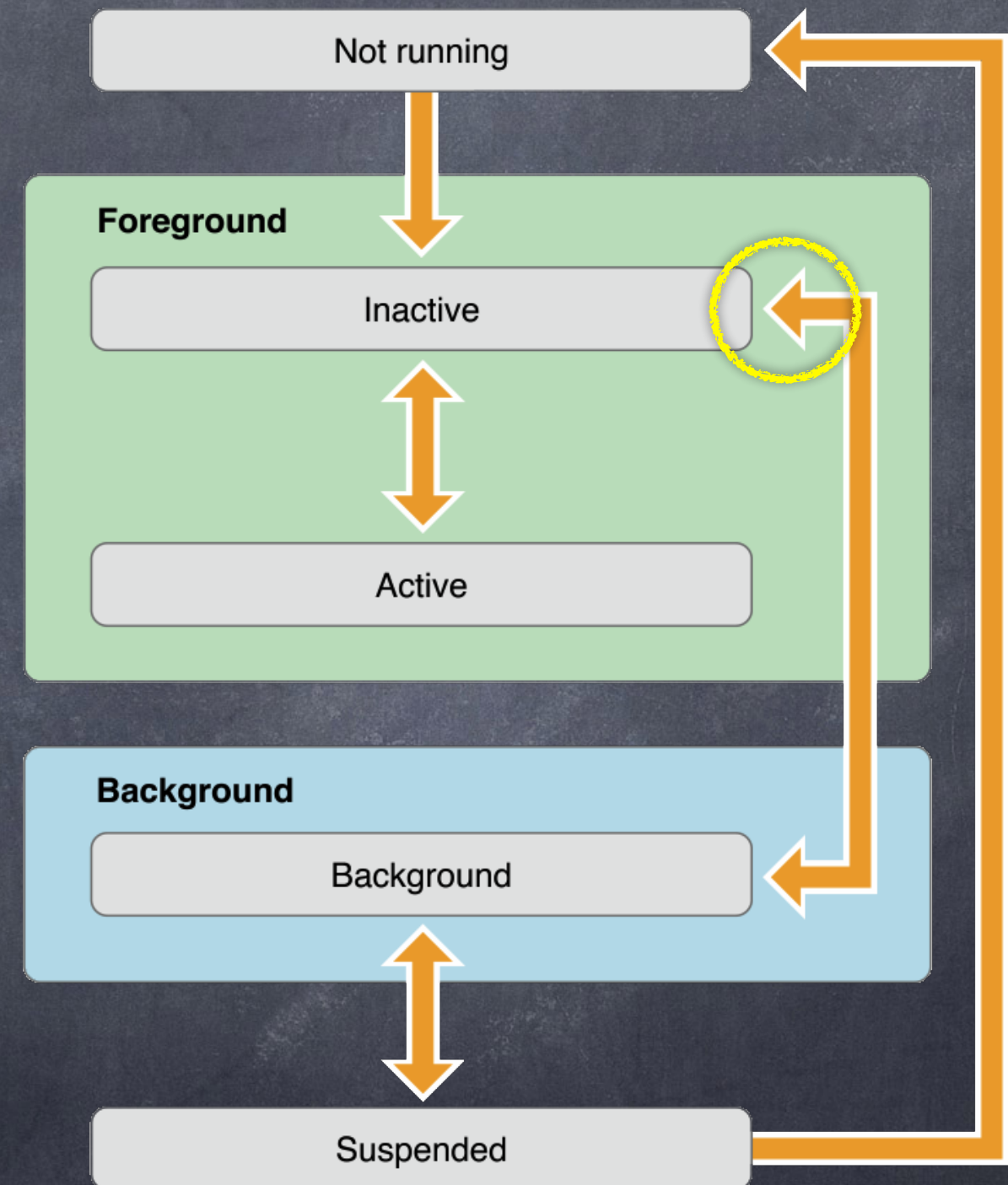
```
UIApplicationWillEnterForeground
```

Whew! You were not killed from background state!

Time to un-batten the hatches.

Maybe undo what you did in `DidEnterBackground`.

You will likely soon be made Active.





# UIApplicationDelegate

## • Other AppDelegate items of interest ...

State Restoration (saving the state of your UI so that you can restore it even if you are killed).

Data Protection (files can be set to be protected when a user's device's screen is locked).

Open URL (in Xcode's Info tab of Project Settings, you can register for certain URLs).

Background Fetching (you can fetch and receive results while in the background).





# UIApplication

## • Shared instance

There is a single `UIApplication` instance in your application

```
let myApp = UIApplication.shared
```

It manages all global behavior

You never need to subclass it

It delegates everything you need to be involved in to its `UIApplicationDelegate`

However, it does have some useful functionality ...

## • Opening a URL in another application

```
func open(URL)
```

```
func canOpenURL(URL) -> Bool
```

## • Registering to receive Push Notifications

```
func (un)registerForRemoteNotifications()
```

Notifications, both local and push, are handled by the `UNNotification` framework.





# UIApplication

- Setting the fetch interval for background fetching

You must set this if you want background fetching to work ...

```
func setMinimumBackgroundFetchInterval(TimeInterval)
```

Usually you will set this to `UIApplicationBackgroundFetchIntervalMinimum`

- Asking for more time when backgrounded

```
func beginBackgroundTask(withExpirationHandler: (() -> Void)?) -> UIBackgroundTaskIdentifier
```

Do NOT forget to call `endBackgroundTask(UIBackgroundTaskIdentifier)` when you're done!

- Turning on the "network in use" spinner (status bar upper left)

```
var isNetworkActivityIndicatorVisible: Bool // unfortunately just a Bool, be careful
```

- Finding out about things

```
var backgroundTimeRemaining: TimeInterval { get } // until you are suspended
```

```
var preferredContentSizeCategory: UIContentSizeCategory { get } // big fonts or small fonts
```

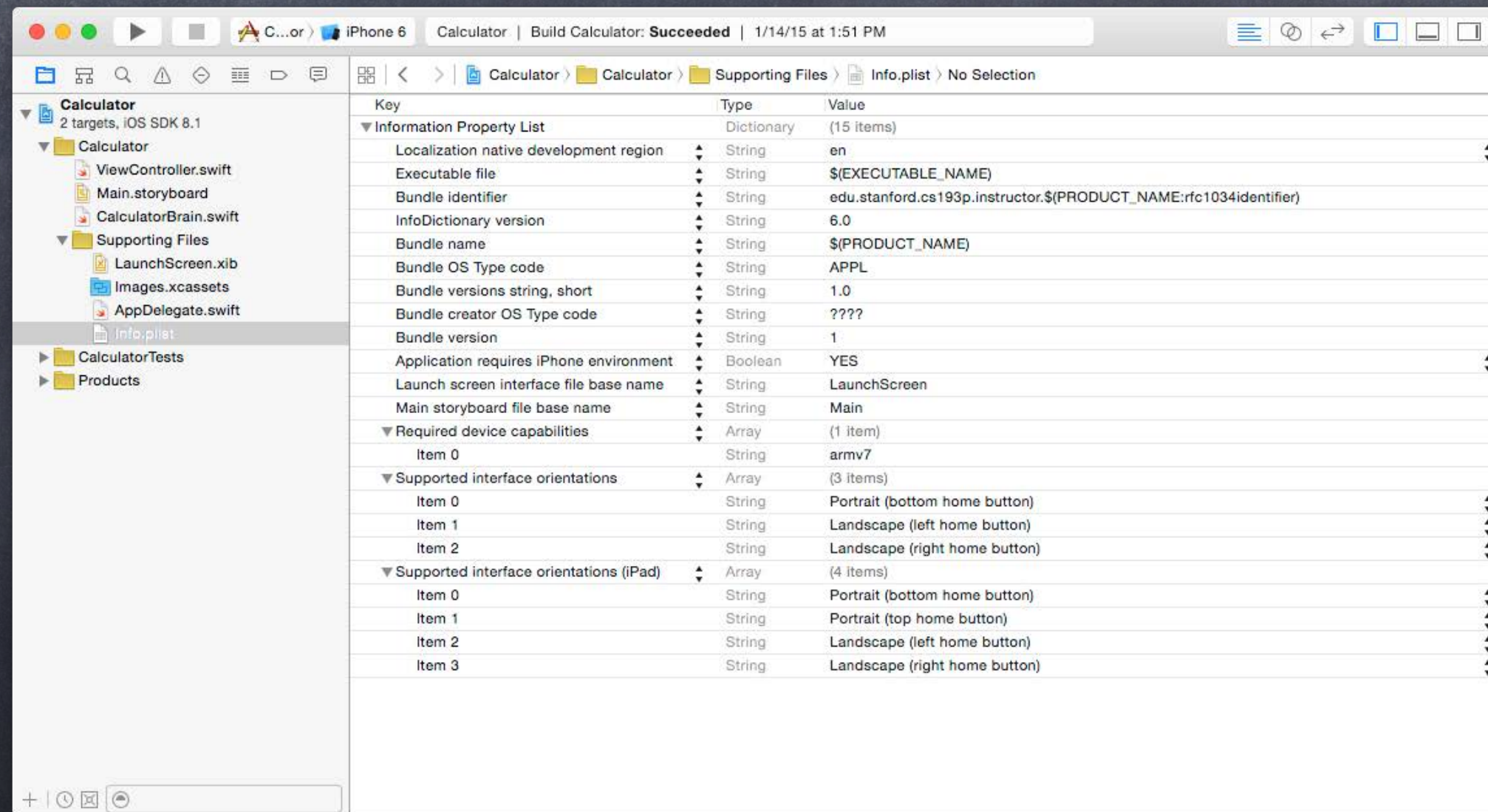
```
var applicationState: UIApplicationState { get } // foreground, background, active
```





# Info.plist

- Many of your application's settings are in Info.plist  
You can edit this file (in Xcode's property list editor) by clicking on Info.plist



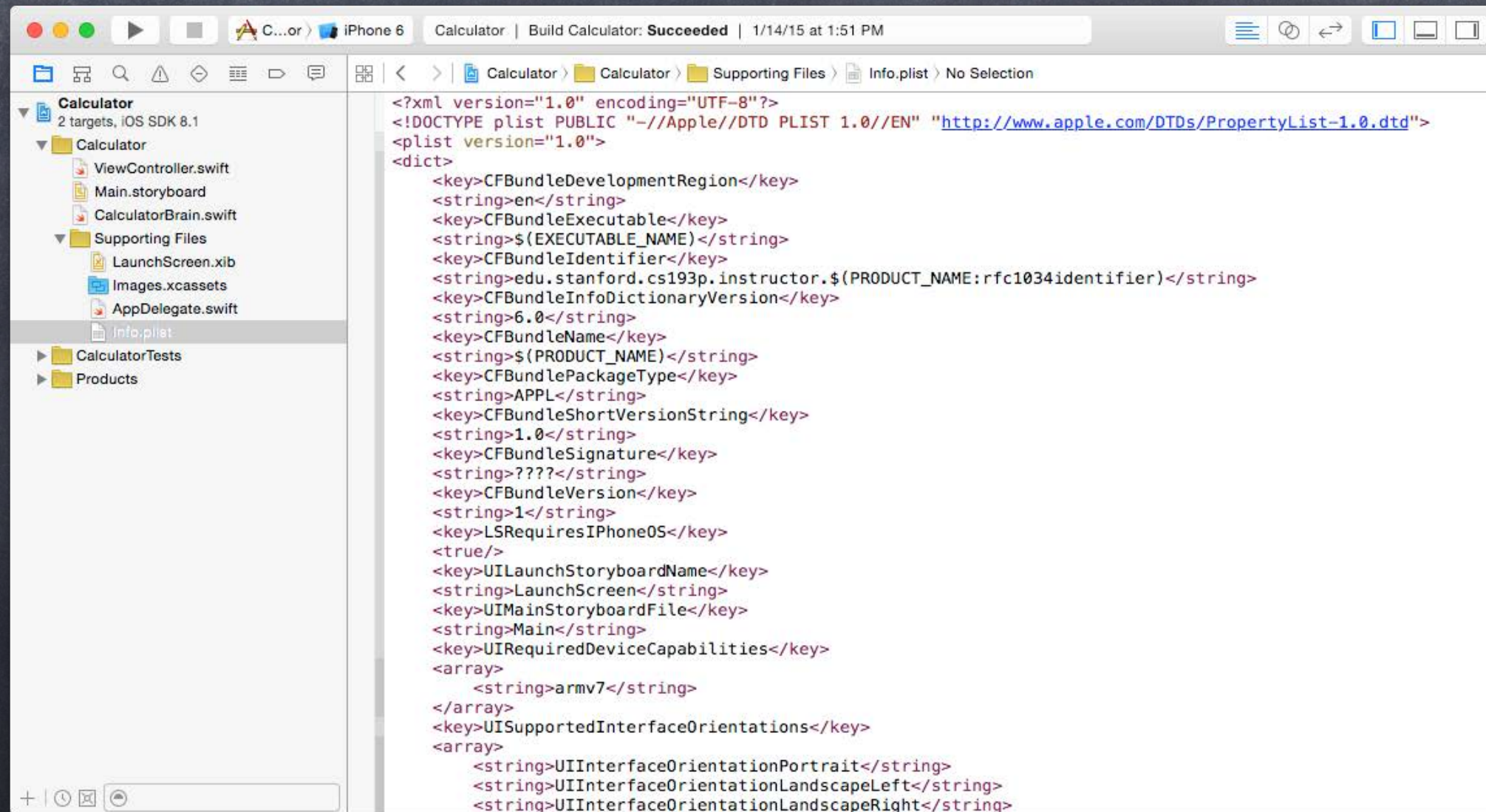


# Info.plist

## Many of your application's settings are in Info.plist

You can edit this file (in Xcode's property list editor) by clicking on Info.plist

Or you can even edit it as raw XML!

A screenshot of the Xcode IDE showing the Info.plist file for a project named 'Calculator'. The left sidebar shows the project structure with 'Info.plist' selected under 'Supporting Files'. The main editor displays the raw XML content of the Info.plist file. The XML includes keys for development region, executable name, bundle identifier, version, name, package type, short version string, signature, version, LSRequiresIPhoneOS, launch storyboard name, main storyboard file, required device capabilities, and supported interface orientations.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plist PUBLIC "-//Apple//DTD PLIST 1.0//EN" "http://www.apple.com/DTDs/PropertyList-1.0.dtd">
<plist version="1.0">
<dict>
  <key>CFBundleDevelopmentRegion</key>
  <string>en</string>
  <key>CFBundleExecutable</key>
  <string>$(EXECUTABLE_NAME)</string>
  <key>CFBundleIdentifier</key>
  <string>edu.stanford.cs193p.instructor.$(PRODUCT_NAME:rfc1034identifier)</string>
  <key>CFBundleInfoDictionaryVersion</key>
  <string>6.0</string>
  <key>CFBundleName</key>
  <string>$(PRODUCT_NAME)</string>
  <key>CFBundlePackageType</key>
  <string>APPL</string>
  <key>CFBundleShortVersionString</key>
  <string>1.0</string>
  <key>CFBundleSignature</key>
  <string>????</string>
  <key>CFBundleVersion</key>
  <string>1</string>
  <key>LSRequiresIPhoneOS</key>
  <true/>
  <key>UILaunchStoryboardName</key>
  <string>LaunchScreen</string>
  <key>UIMainStoryboardFile</key>
  <string>Main</string>
  <key>UIRequiredDeviceCapabilities</key>
  <array>
    <string>armv7</string>
  </array>
  <key>UISupportedInterfaceOrientations</key>
  <array>
    <string>UIInterfaceOrientationPortrait</string>
    <string>UIInterfaceOrientationLandscapeLeft</string>
    <string>UIInterfaceOrientationLandscapeRight</string>
  </array>
</dict>
</plist>
```





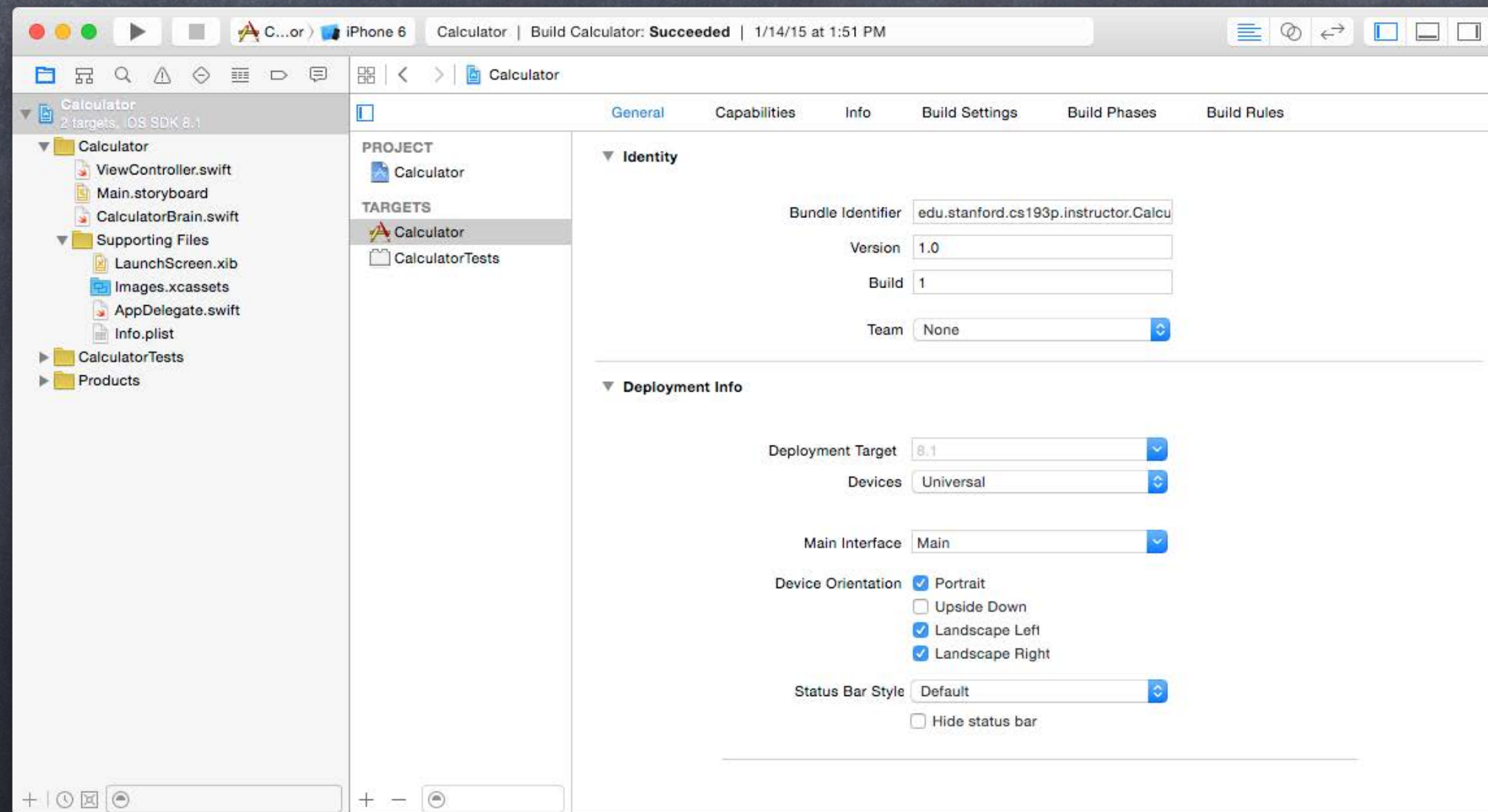
# Info.plist

## • Many of your application's settings are in Info.plist

You can edit this file (in Xcode's property list editor) by clicking on Info.plist

Or you can even edit it as raw XML!

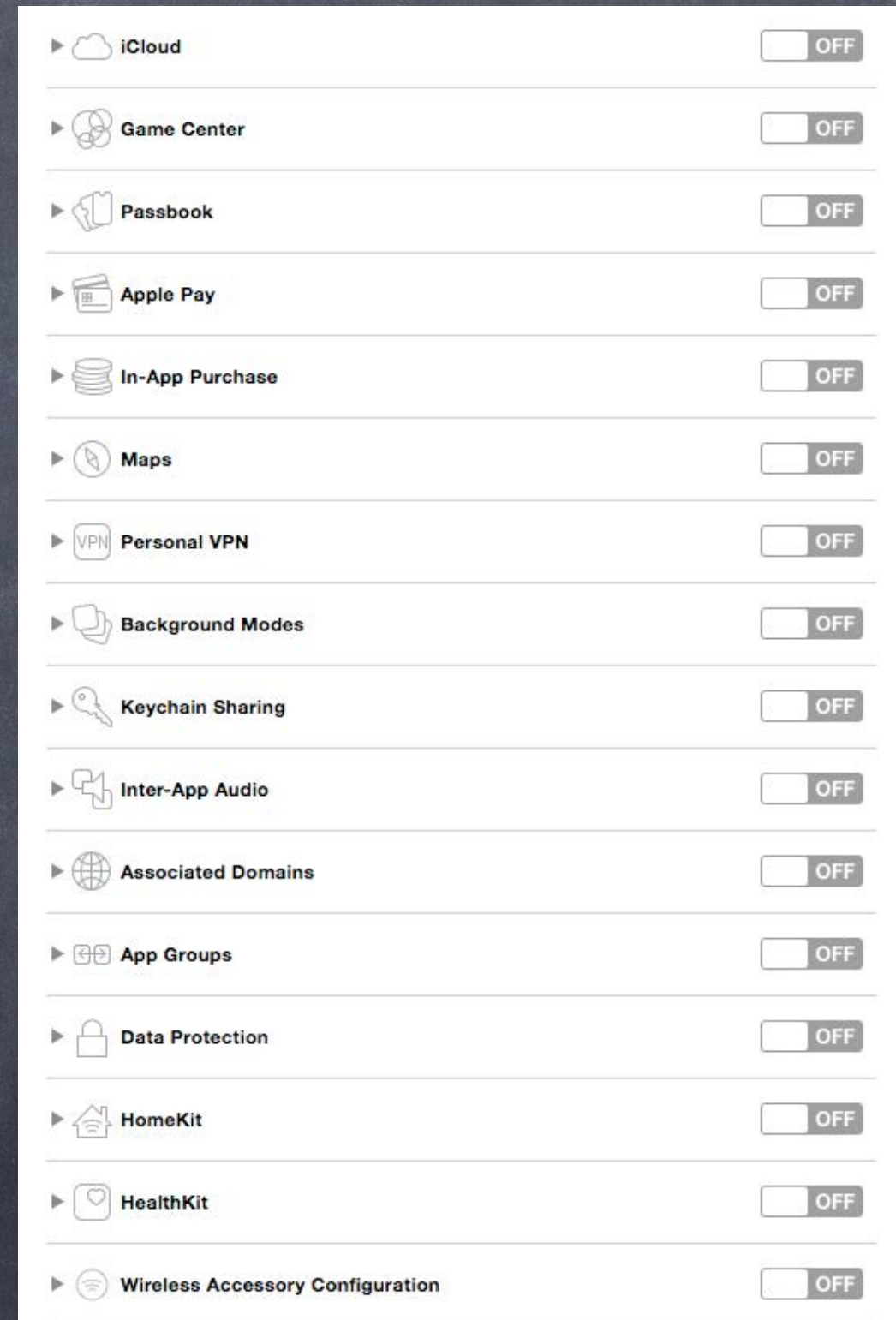
But usually you edit Info.plist settings by clicking on your project in the Navigator ...





# Capabilities

- Some features require enabling  
These are server and interoperability features  
Like iCloud, Game Center, etc.
- Switch on in Capabilities tab  
Inside your Project Settings
- Not enough time to cover these!  
But check them out!  
Many require full Developer Program membership  
Familiarize yourself with their existence





# Demo Code

Download the [demo code](#) from today's lecture.

