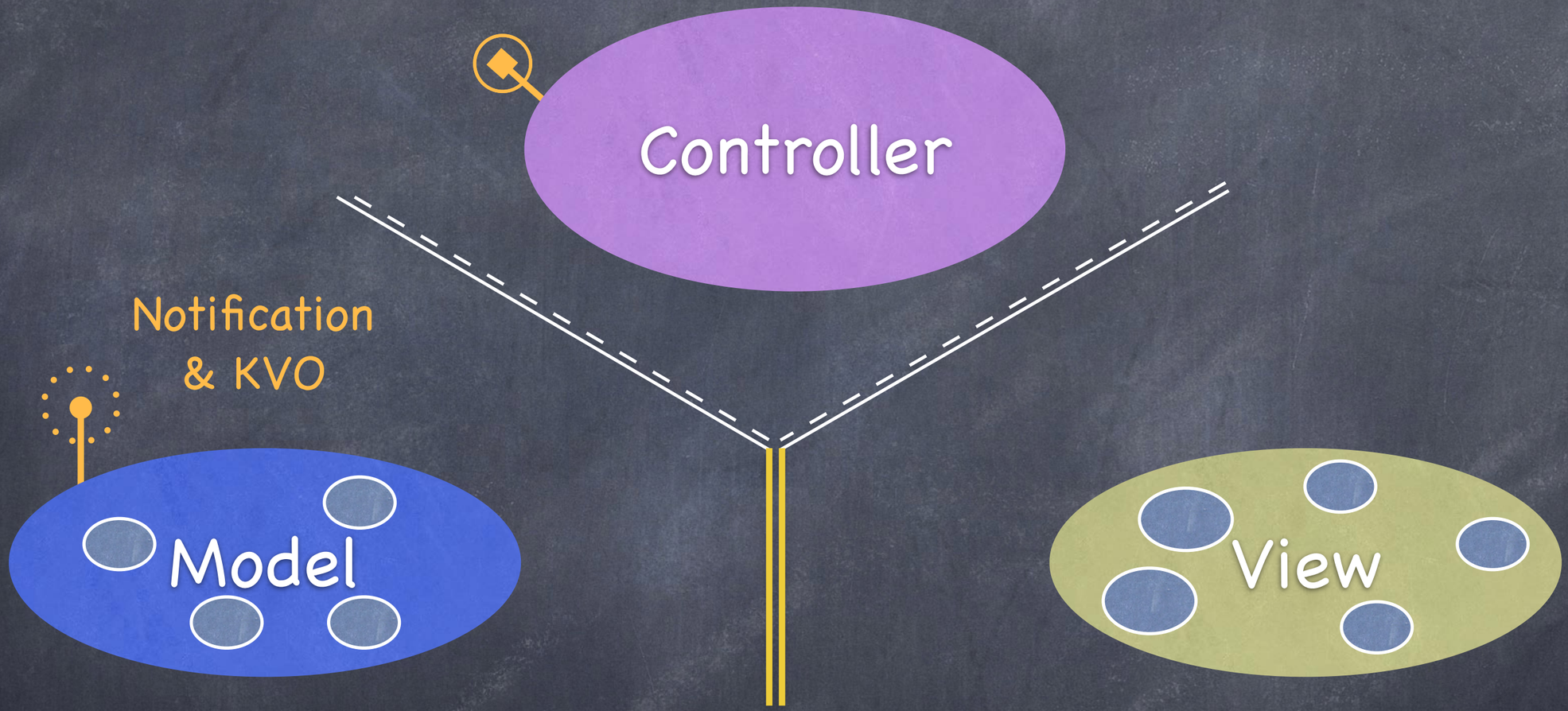
# Stanford CS193p

## Developing Applications for iOS
## Spring 2016

# Today

- Application Lifecycle
  - Notifications
  - AppDelegate
  - Info.plist
  - Capabilities
- Alerts
  - Informing the user of some notable happening
- Cloud Kit
  - Sort of like a (very) simplified Core Data but on the network
  - (Time permitting)

# MVC




Radio Station Communication

# NSNotification

- Notifications

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

- NSNotificationCenter

  Get the default "notification center" via `NSNotificationCenter.defaultCenter()`

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

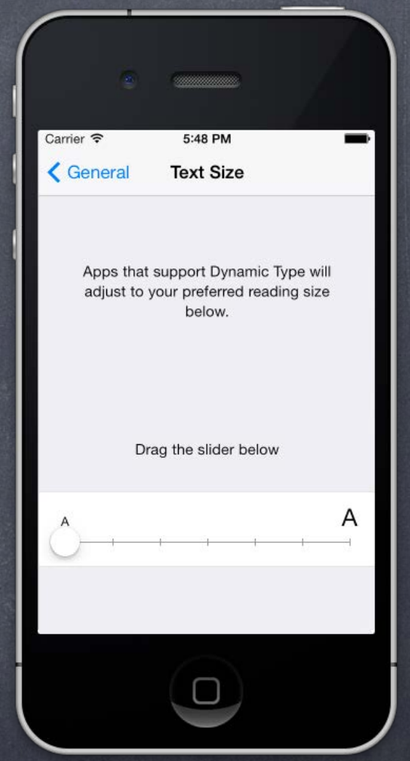
```
var observer: NSObjectProtocol? // a cookie to remove with
observer = addObserverForName(String, // the name of the radio station
                  object: AnyObject?, // the broadcaster (or nil for "anyone")
                   queue: NSOperationQueue?) // the queue to execute the closure on
{ (notification: NSNotification) -> Void in
    let info: [NSObject:AnyObject]? = notification.userInfo
    // info is a dictionary of notification-specific information
}
```

# NSNotification

- Example

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

```
let center = NSNotificationCenter.defaultCenter()

var observer =
center.addObserverForName(UIContentSizeCategoryDidChangeNotification
                  object: UIApplication.sharedApplication(),
                   queue: NSOperationQueue.mainQueue())
{ 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
```




# NSNotification

- Posting an NSNotification

Create an NSNotification …

```
let notification = NSNotification(
      name: String                  // name of the "radio station"
    object: AnyObject?,             // who is sending this notification (usually self)
  userInfo: Dictionary              // any info you want to pass to station listeners
)
```

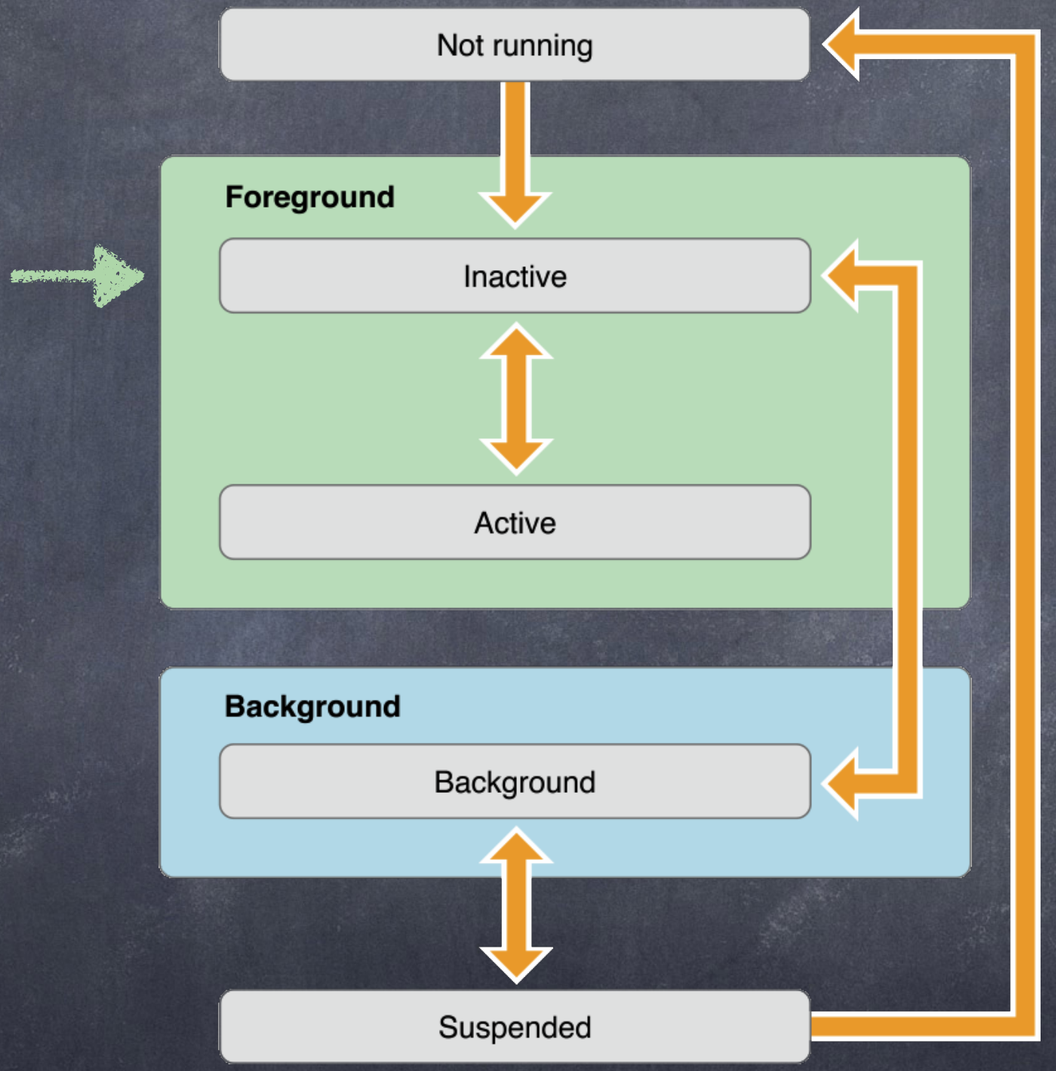
… then post the NSNotification …

```
NSNotificationCenter.defaultCenter().postNotification(notification)
```

Any blocks added with `addObserverForName` will be executed.
Either immediately on the same queue as `postNotification` (if `queue` was nil).
Or asynchronously by posting the block onto the `queue` specified with `addObserverForName`.

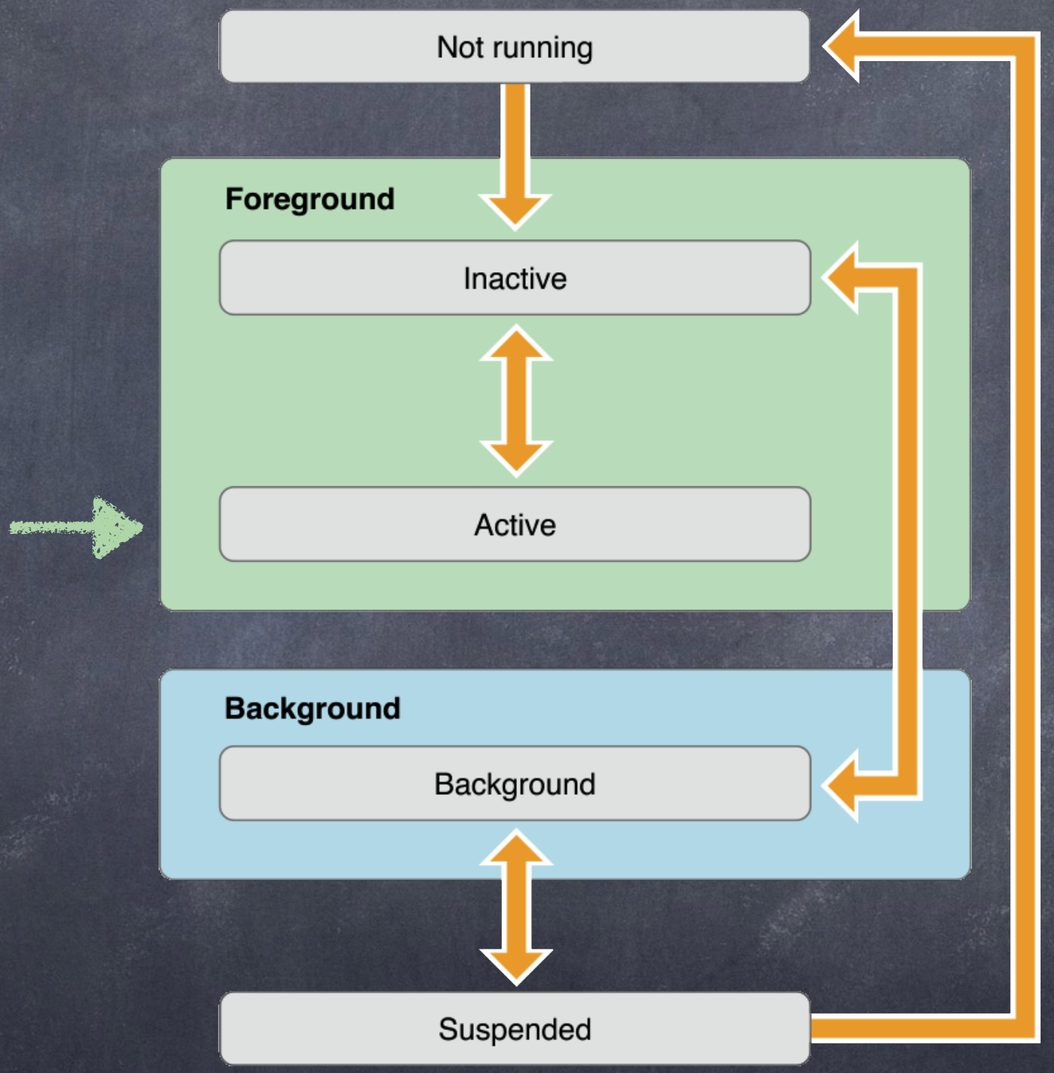
# Application Lifecycle

Running your code, but no UI events.

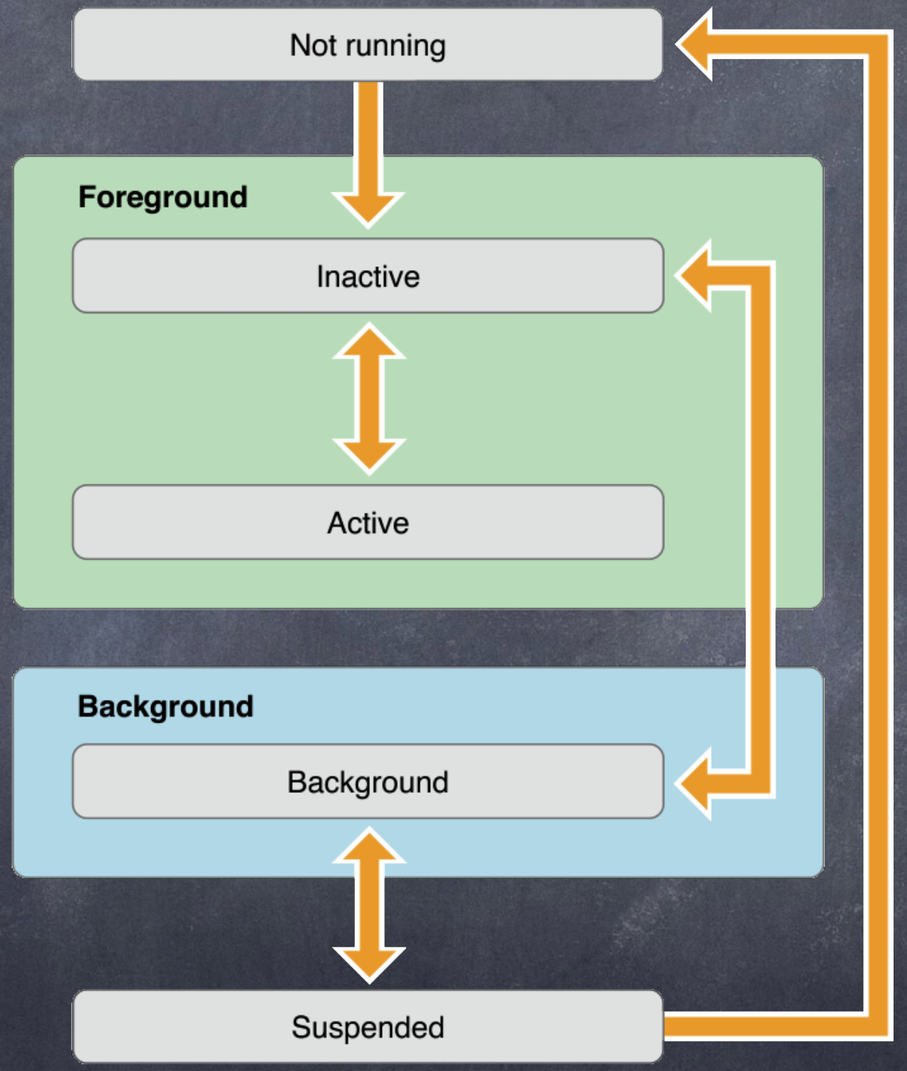



# Application Lifecycle

Running your code, receiving and processing UI events.

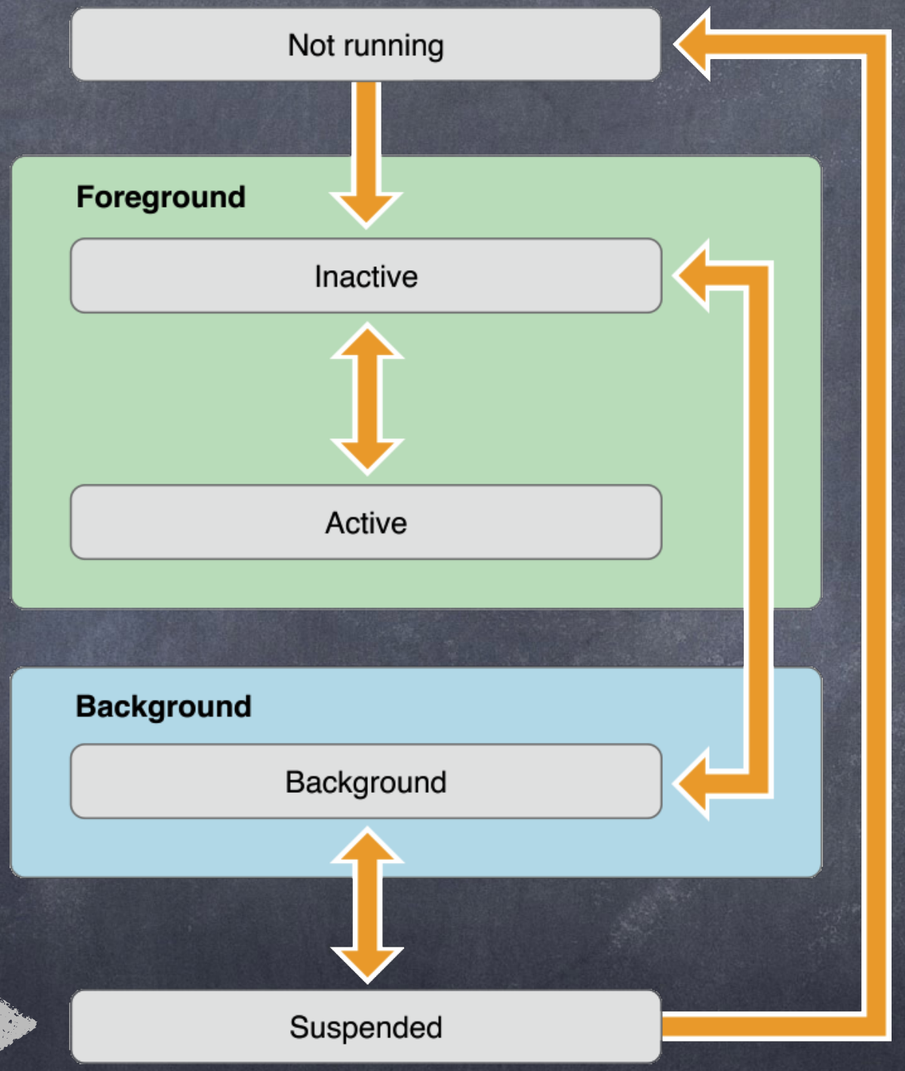



# Application Lifecycle



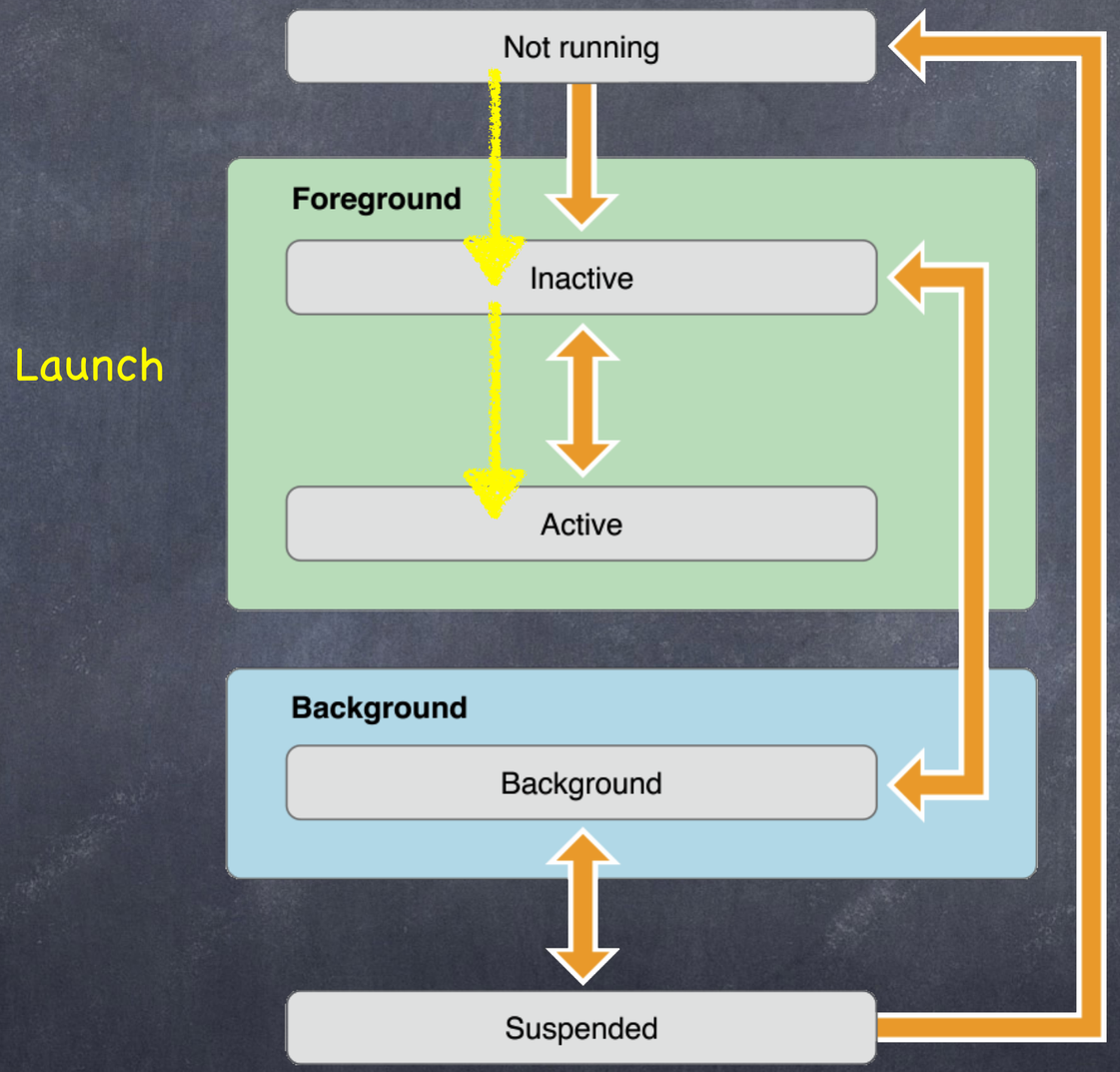

Running your code for a limited time, no UI events.

# Application Lifecycle



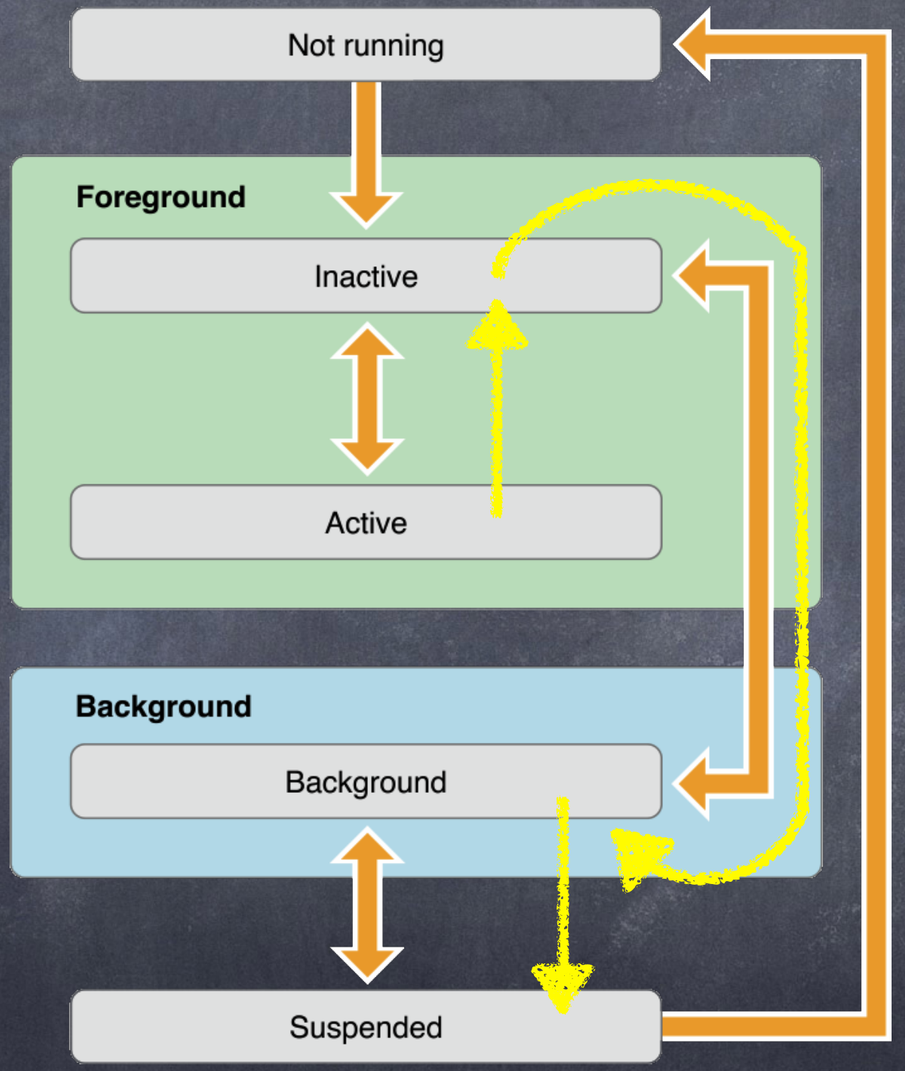

Your code not running.
You could be killed.

# Application Lifecycle

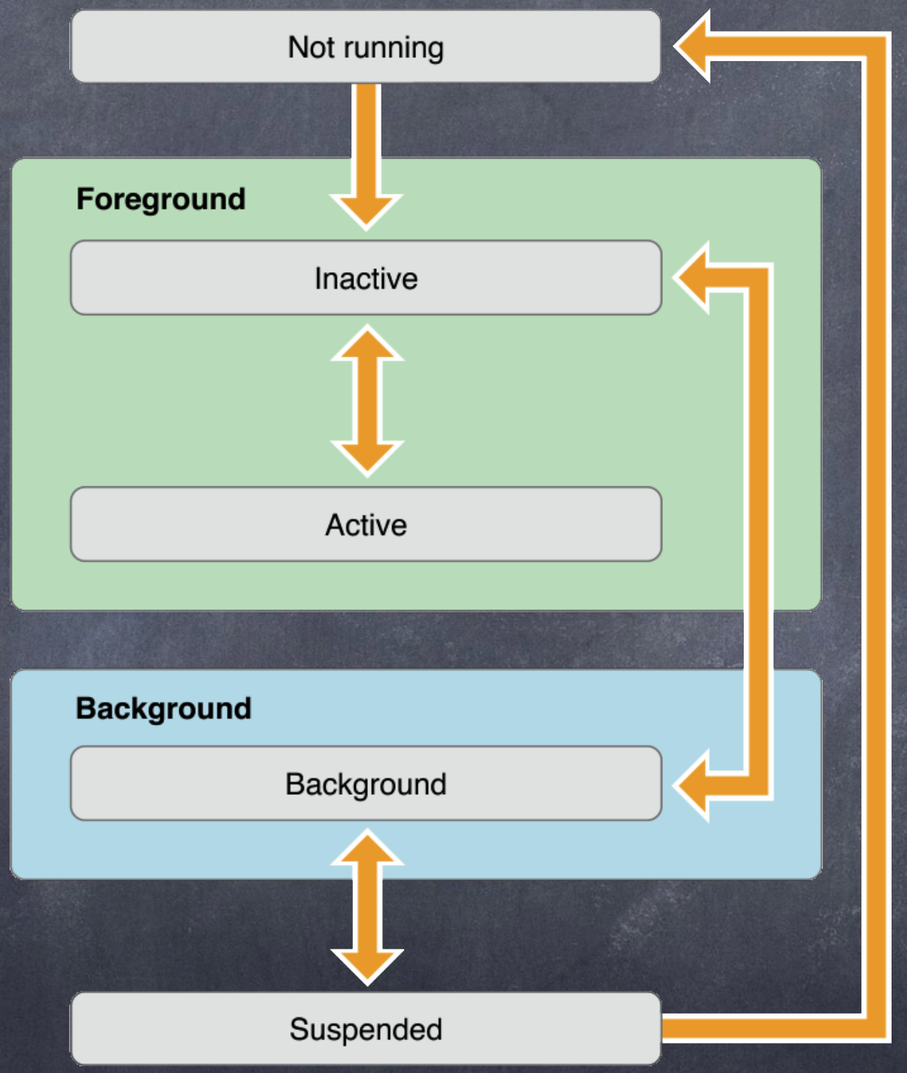


# Application Lifecycle

Switch to another application




# Application Lifecycle




Killed
(notice no code runs between suspended and killed)

# Application Lifecycle

Your AppDelegate will receive …

```
func application(UIApplication,
    didFinishLaunchingWithOptions: [NSObject:AnyObject])
```

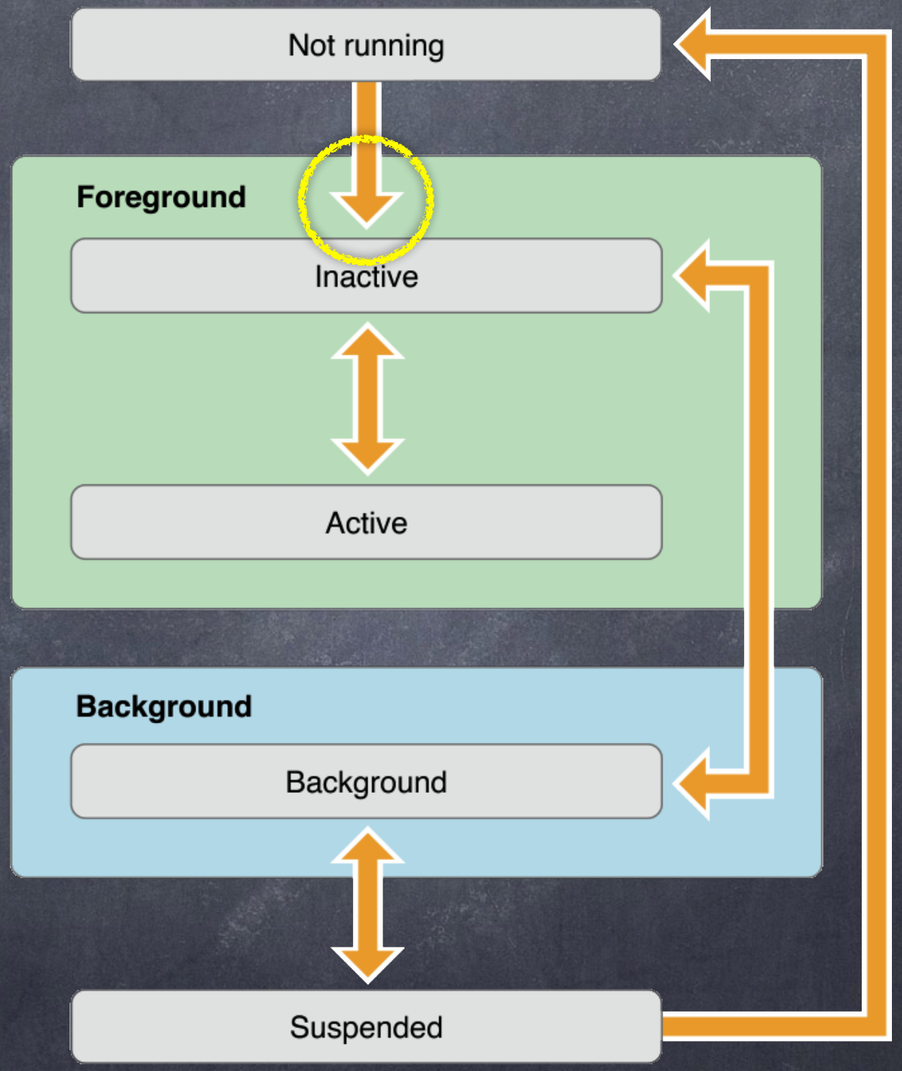
… and you can observe …

```
UIApplicationDidFinishLaunchingNotification
```

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

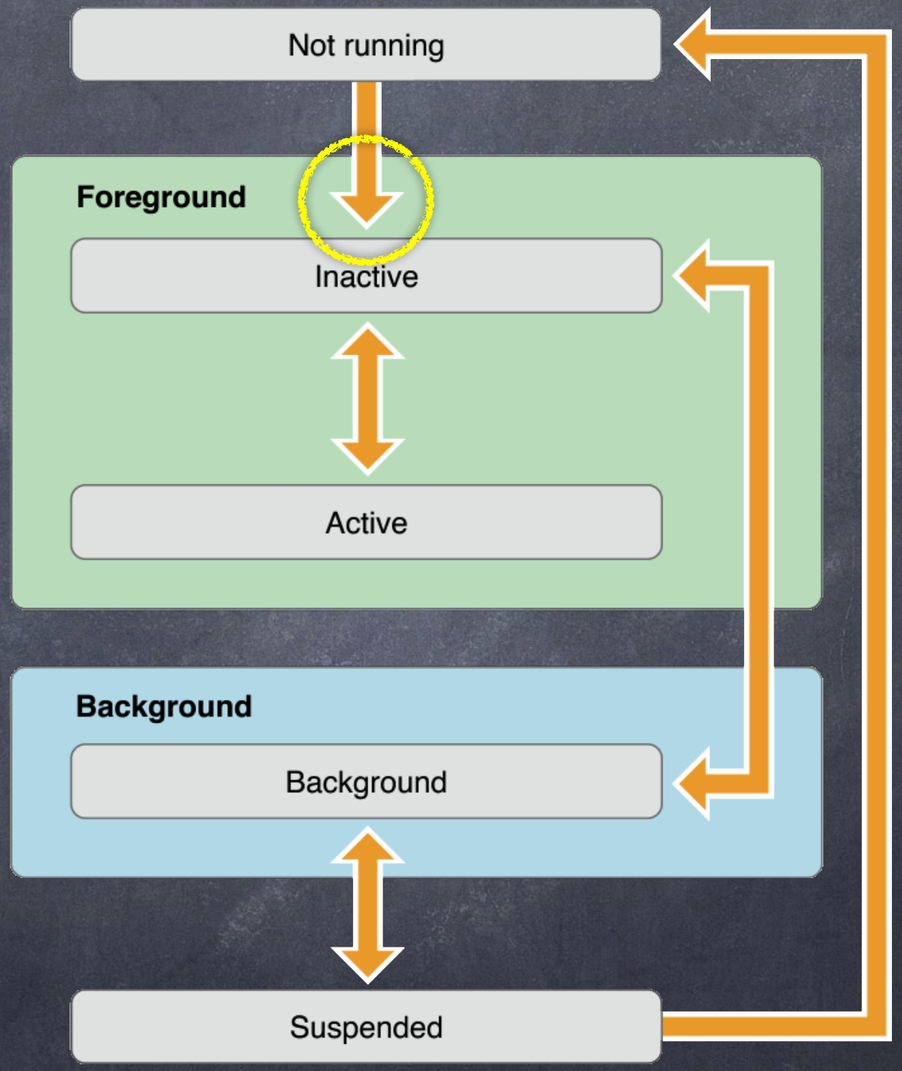
Your AppDelegate will receive …

```
func application(UIApplication,
    didFinishLaunchingWithOptions: [NSObject:AnyObject])
```

… and you can observe …

```
UIApplicationDidFinishLaunchingNotification
```

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

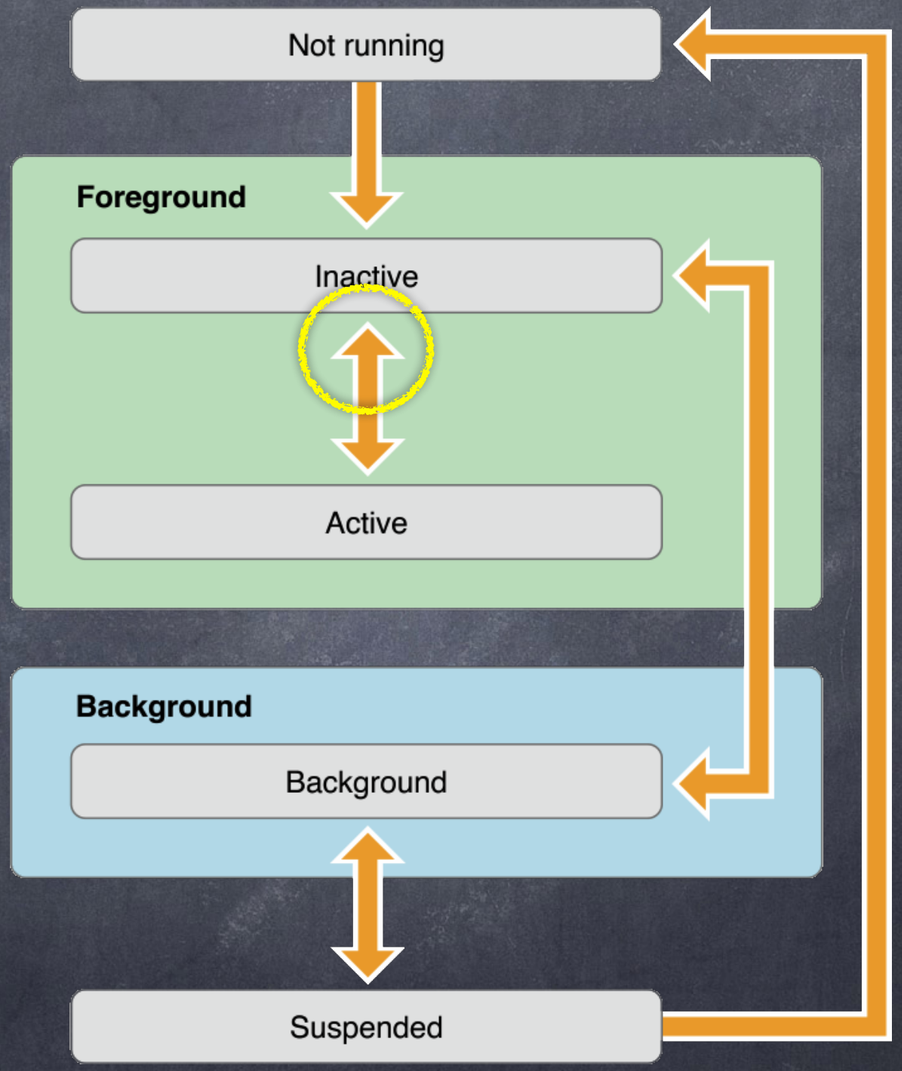
Your AppDelegate will receive …

```
func applicationWillResignActive(UIApplication)
```

… and you can observe …

```
UIApplicationWillResignActiveNotification
```

You will want to "pause" your UI here.
For example, Breakout would want to stop the bouncing ball.
This might happen because a phone call comes in.
Or you might be on your way to the background.




# Application Lifecycle

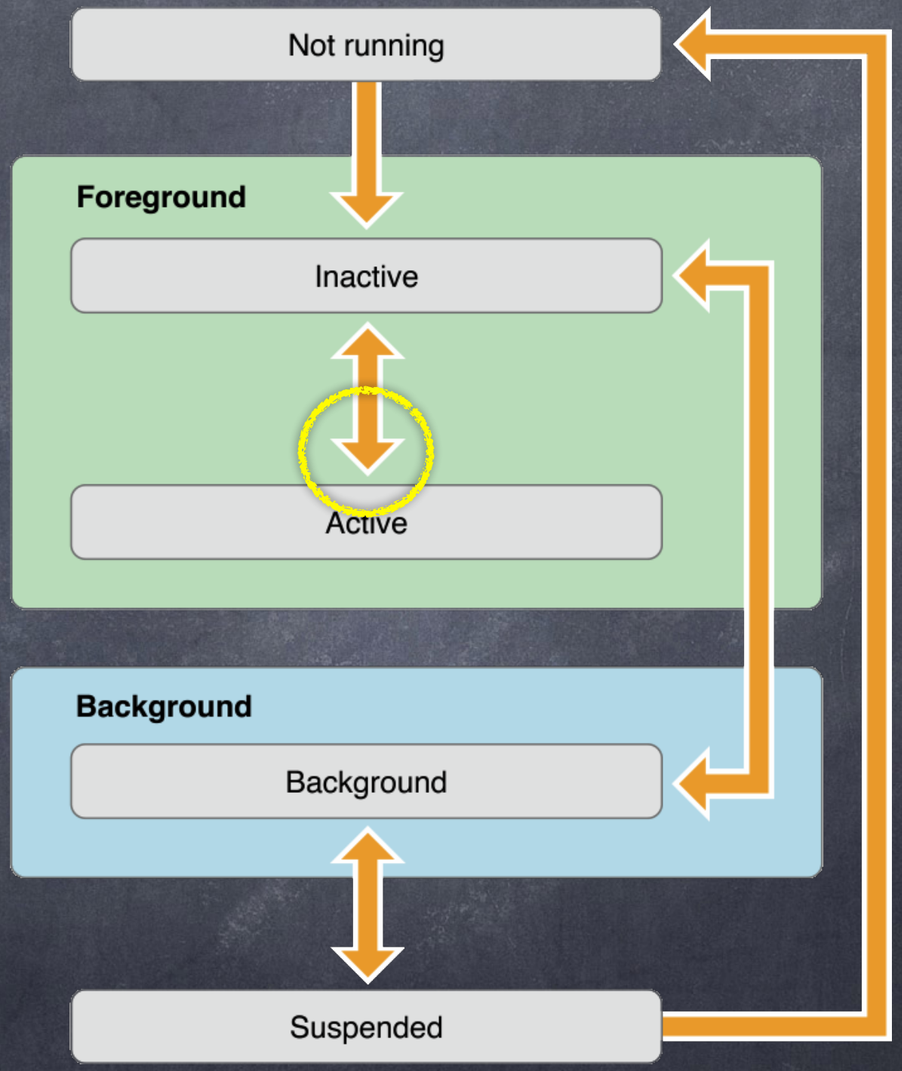
Your AppDelegate will receive …

```
func applicationDidBecomeActive(UIApplication)
```

… and you can observe …

```
UIApplicationDidBecomeActiveNotification
```

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



# Application Lifecycle

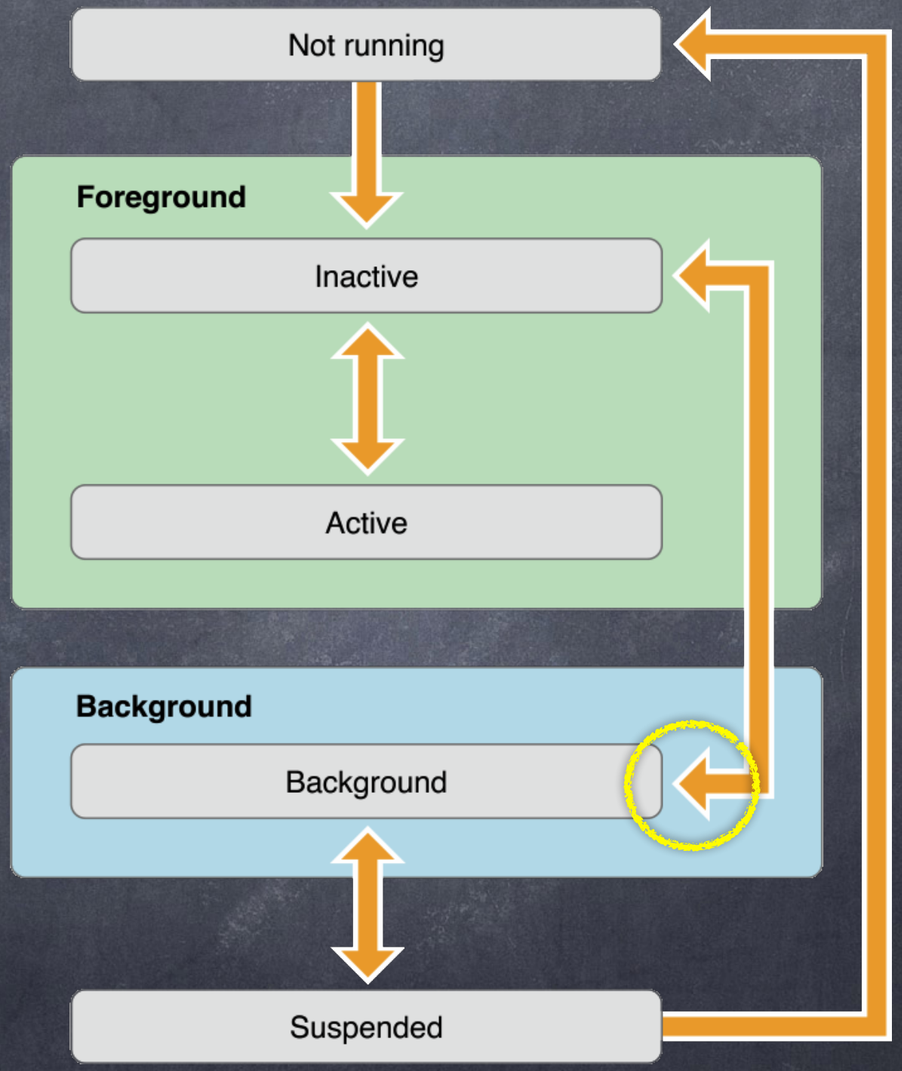
Your AppDelegate will receive …

```
func applicationDidEnterBackground(UIApplication)
```

… and you can observe …

```
UIApplicationDidEnterBackgroundNotification
```

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

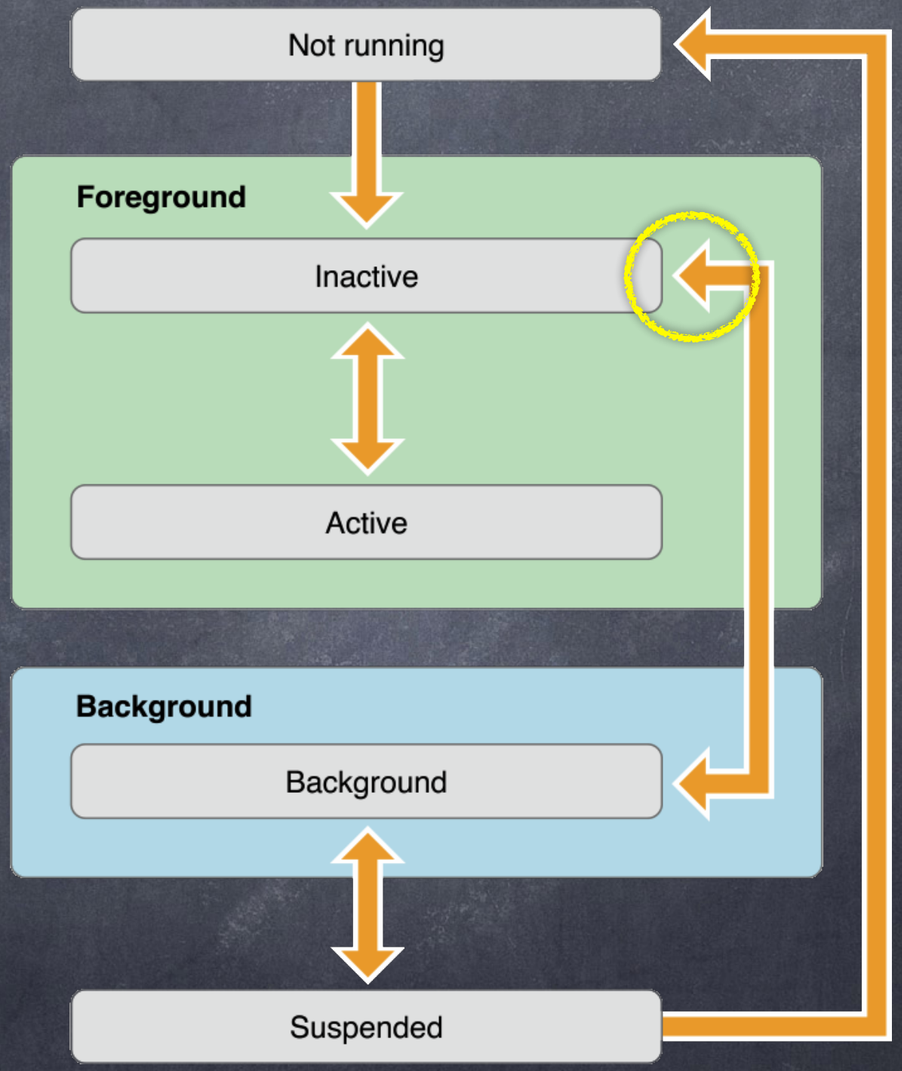
Your `AppDelegate` will receive …

```
func applicationWillEnterForeground(UIApplication)
```

… and you can observe …

```
UIApplicationWillEnterForegroundNotification
```

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 …
  - Local Notifications (set timers to go off at certain times … will wake your application if needed).
  - Remote (Push) Notifications (information coming in from data servers).
  - 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

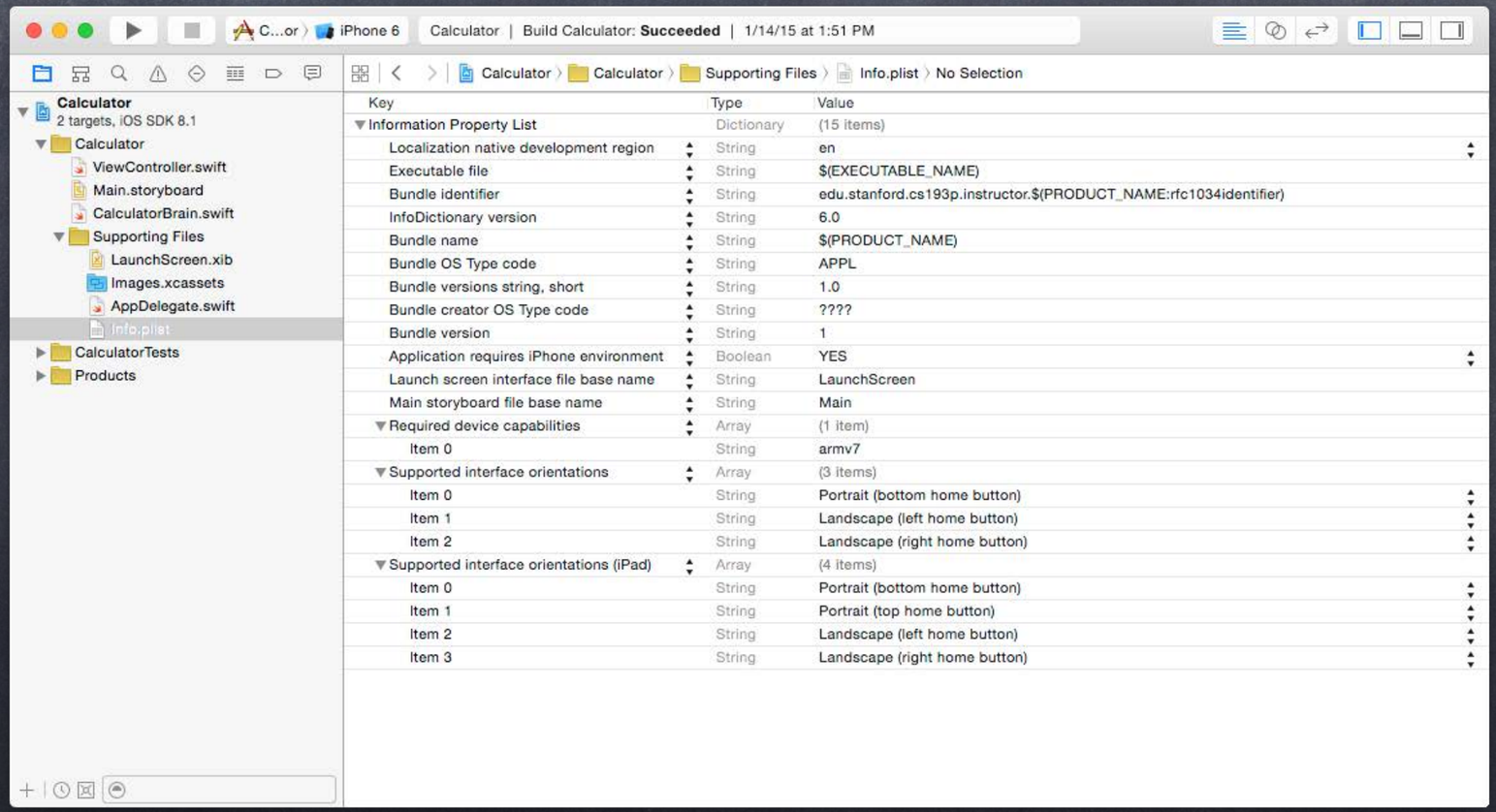
- Shared instance
  - There is a single `UIApplication` instance in your application
  - `let myApp = UIApplication.sharedApplication()`
  - It manages all global behavior
  - You never need to subclass it
  - It delegates everything you need to be involved in to its `UIApplicationDelegate`
  - However, it does have some useful functionality …
- Opening a URL in another application
  - `func openURL(NSURL)`
  - `func canOpenURL(NSURL) -> Bool`
- Registering or Scheduling Notifications (Push or Local)
  - `func (un)registerForRemoteNotifications()`
  - `func scheduleLocalNotification(UILocalNotification)`
  - `func registerUserNotificationSettings(UIUserNotificationSettings)` // permit for badges, etc.

# UIApplication

- Setting the fetch interval for background fetching
  - You must set this if you want background fetching to work …
  - `func setMinimumBackgroundFetchInterval(NSTimeInterval)`
  - Usually you will set this to `UIApplicationBackgroundFetchIntervalMinimum`
- Asking for more time when backgrounded
  - `func backgroundTaskWithExpirationHandler(handler: () -> 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 networkActivityIndicatorVisible: Bool` // unfortunately just a `Bool`, be careful
- Finding out about things
  - `var backgroundTimeRemaining: NSTimeInterval { get }` // until you are suspended
  - `var preferredContentSizeCategory: String { get }` // big fonts or small fonts
  - `var applicationState: UIApplicationState { get }` // foreground, background, active

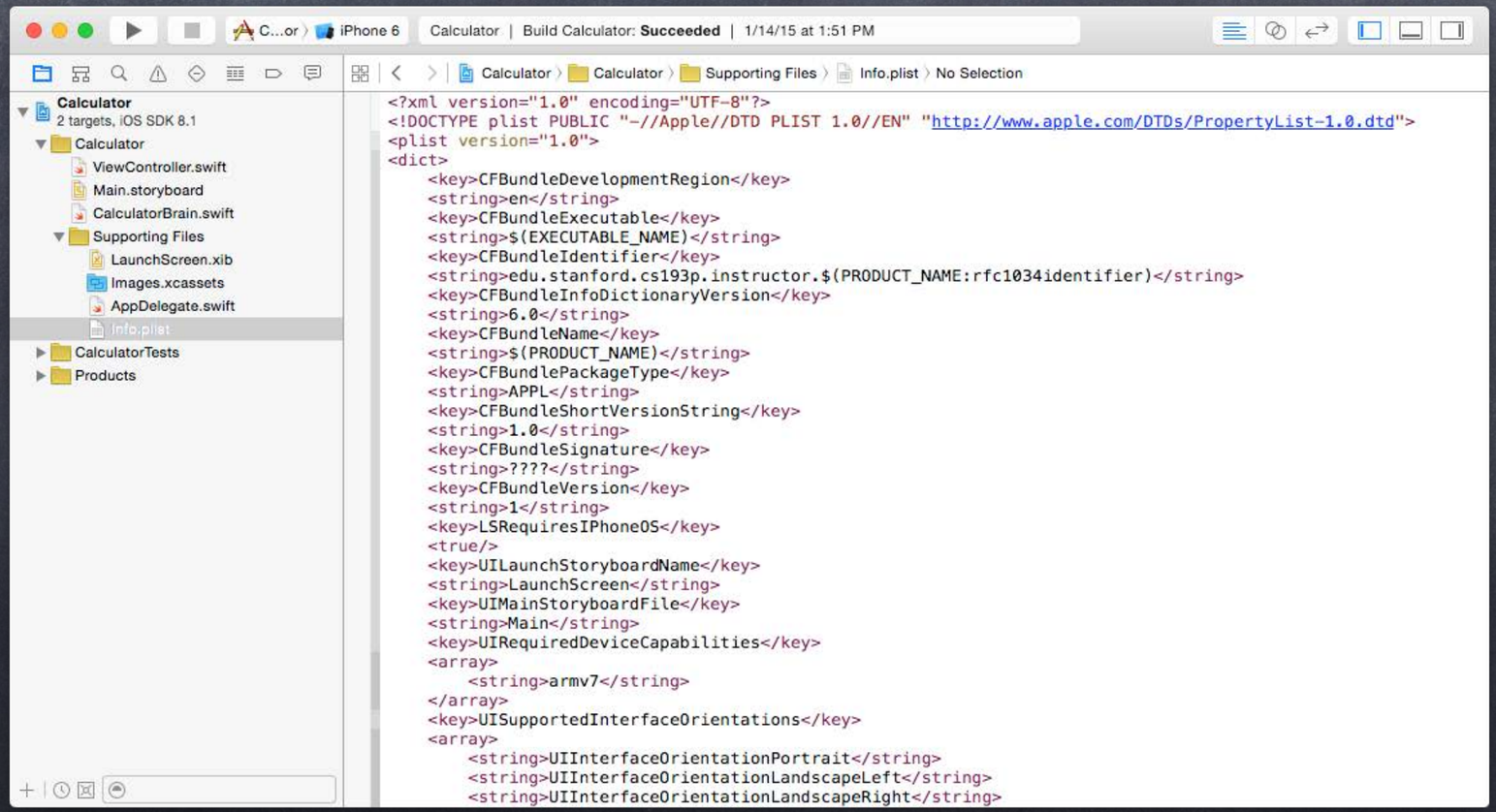
# Info.plist

- 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



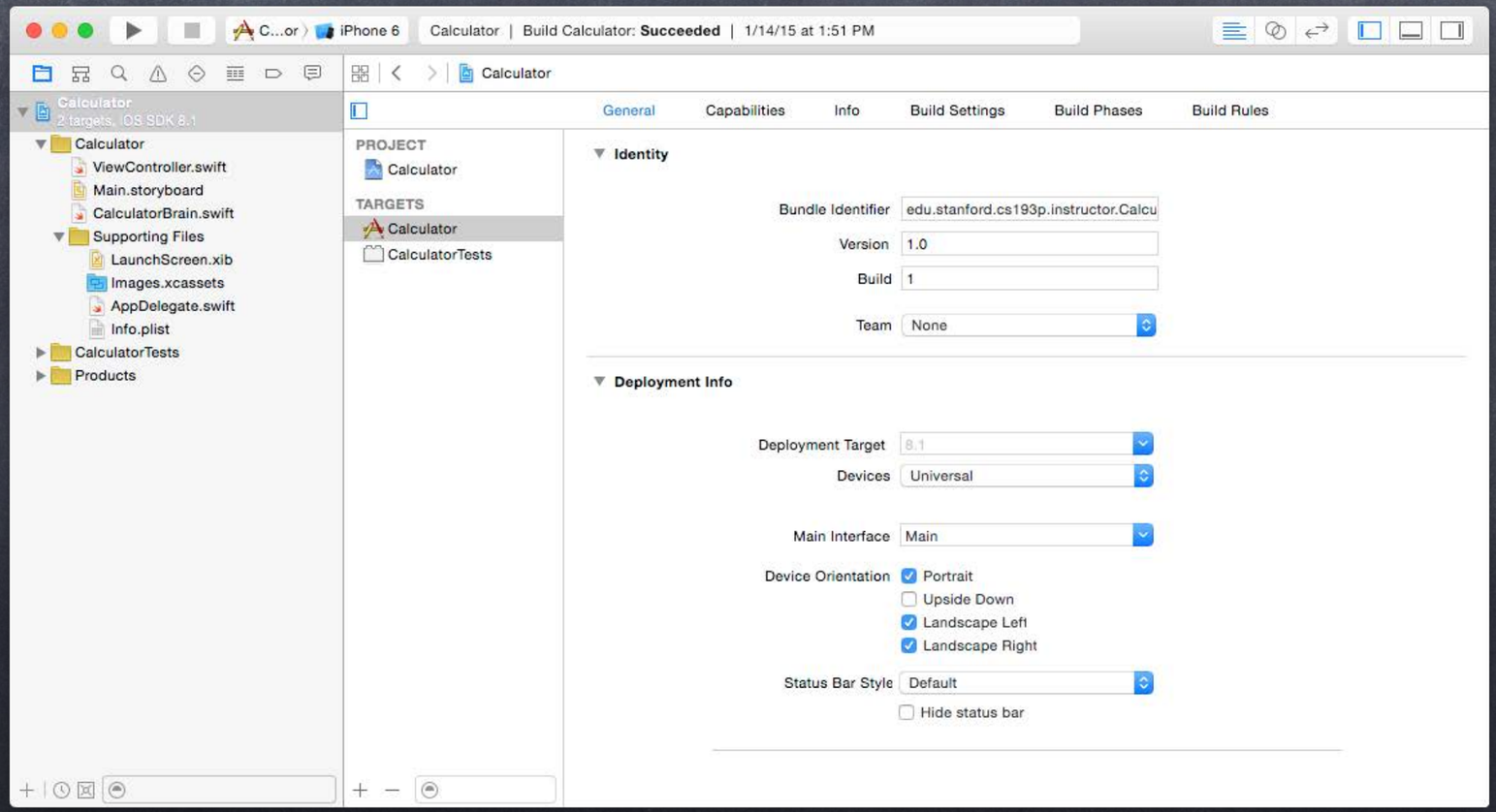

# Info.plist

- 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!



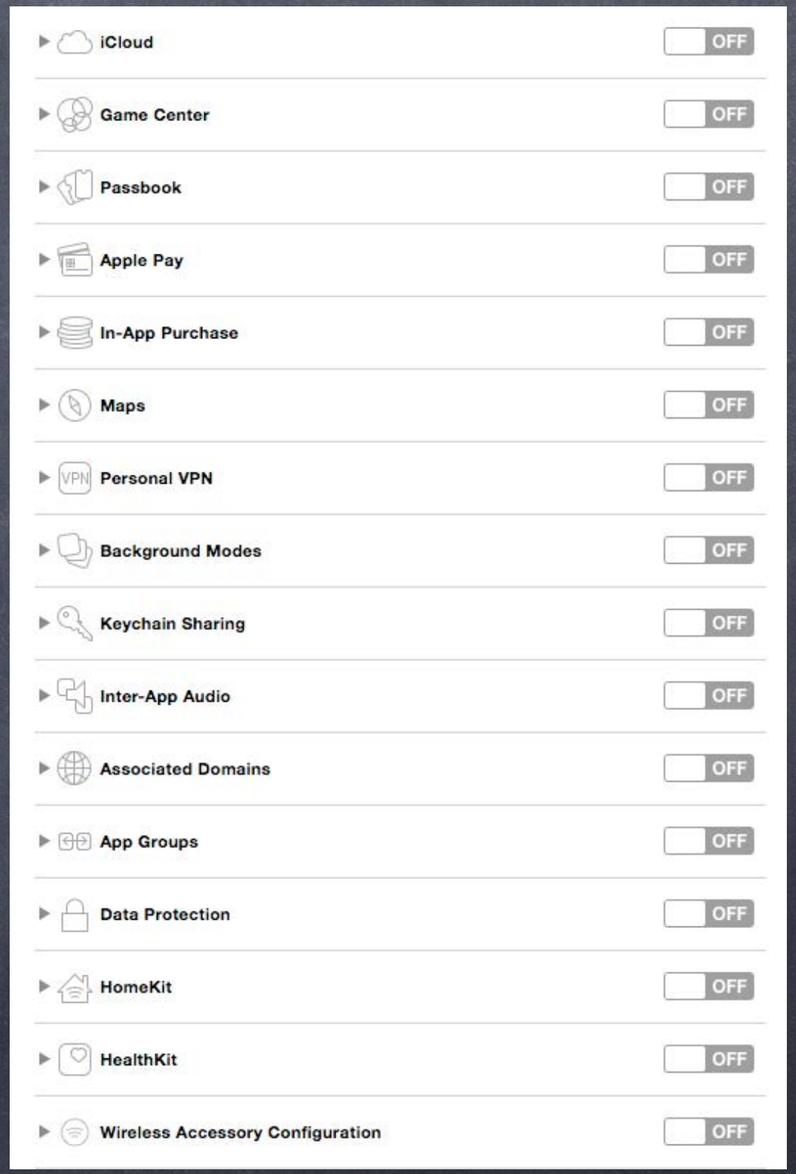

# Info.plist

- 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

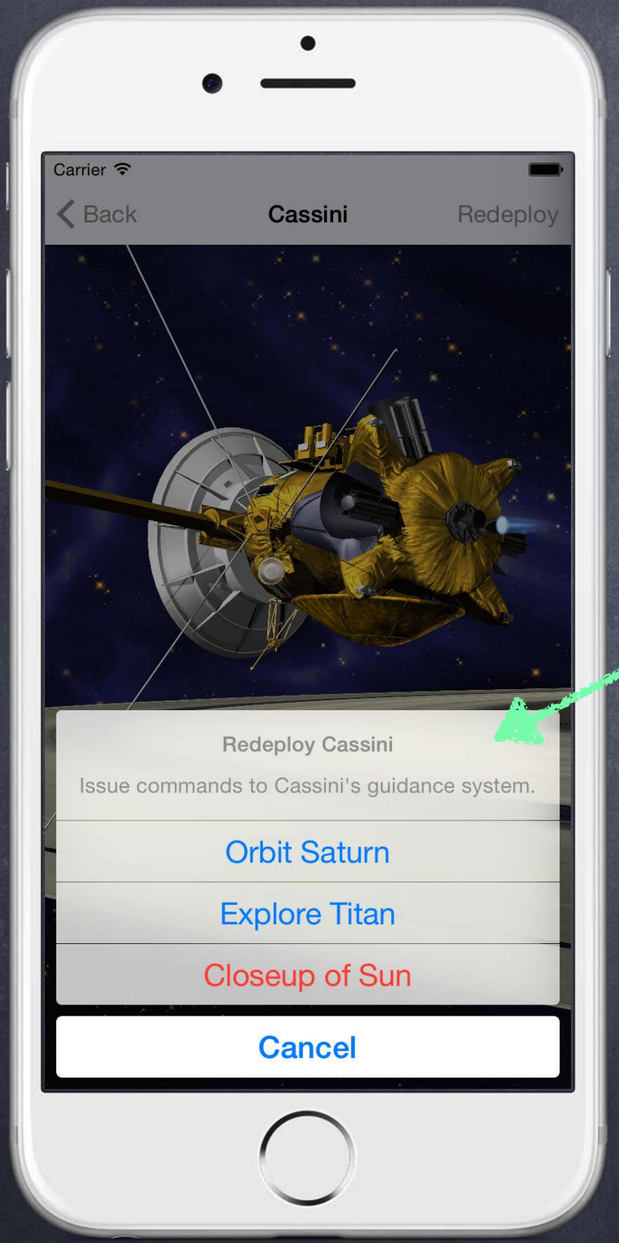
- Some features require enabling
  - These are server and interoperability features
  - Like iCloud, Game Center, etc.
- Switch on in Capabilities tab
  - Inside your Project Settings
- Not enough time to cover these!
  - But check them out!
  - Many require full Developer Program membership
  - Familiarize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




# Alerts and Action Sheets

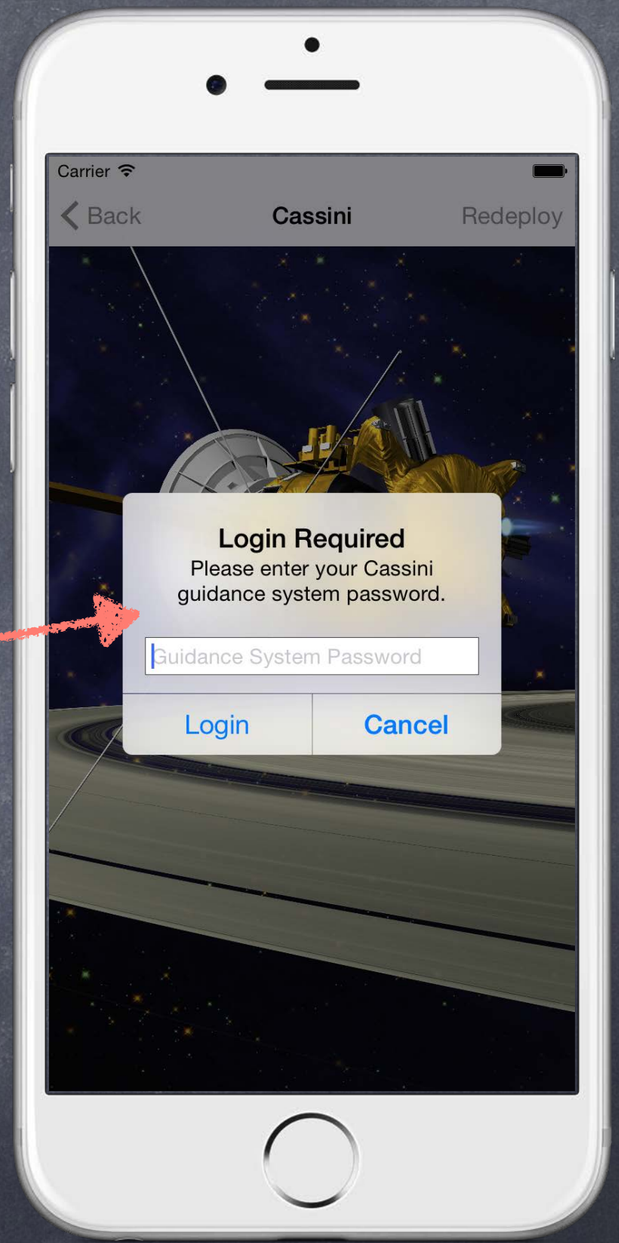
- Two kinds of "pop up and ask the user something" mechanisms
  - Alerts
  - Action Sheets
- Alerts
  - Pop up in the middle of the screen.
  - Usually ask questions with only two (or one) answers (e.g. OK/Cancel, Yes/No, etc.).
  - Can be disruptive to your user-interface, so use carefully.
  - Often 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

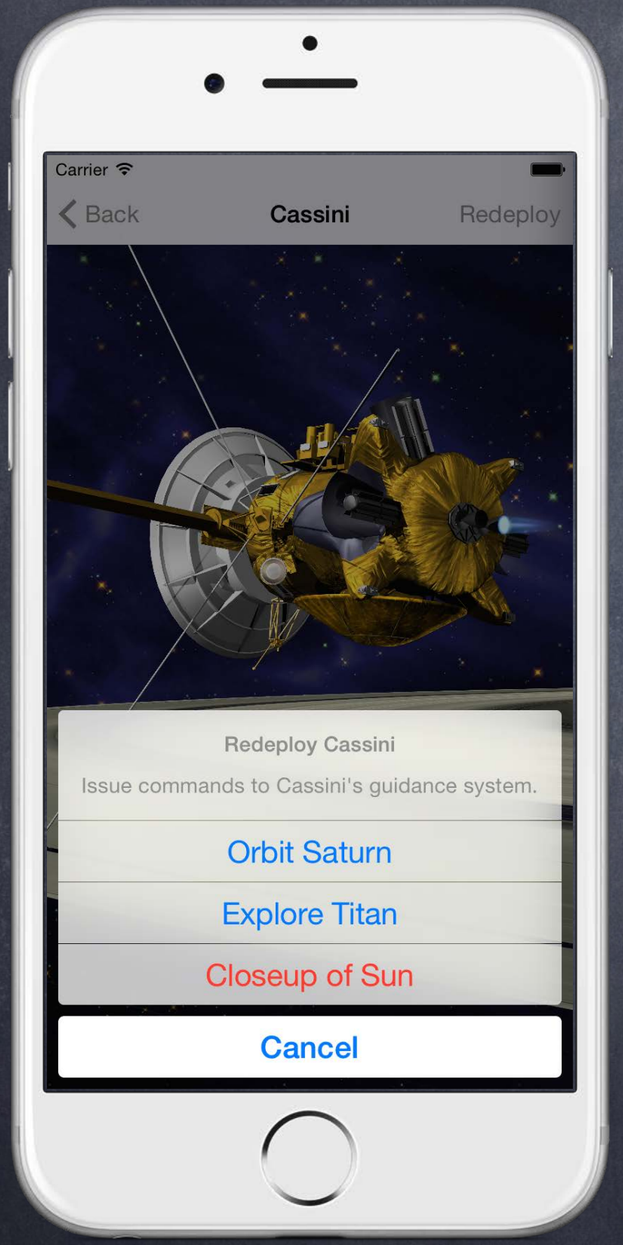



Action Sheet

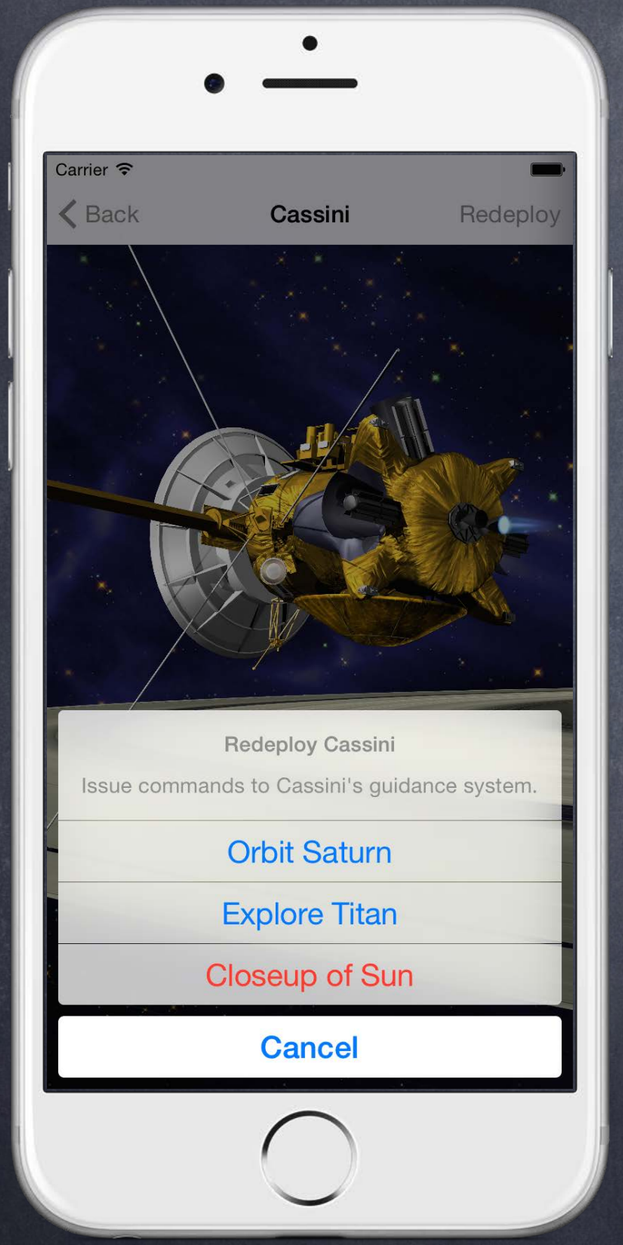
Alert




CS193p
Spring 2016

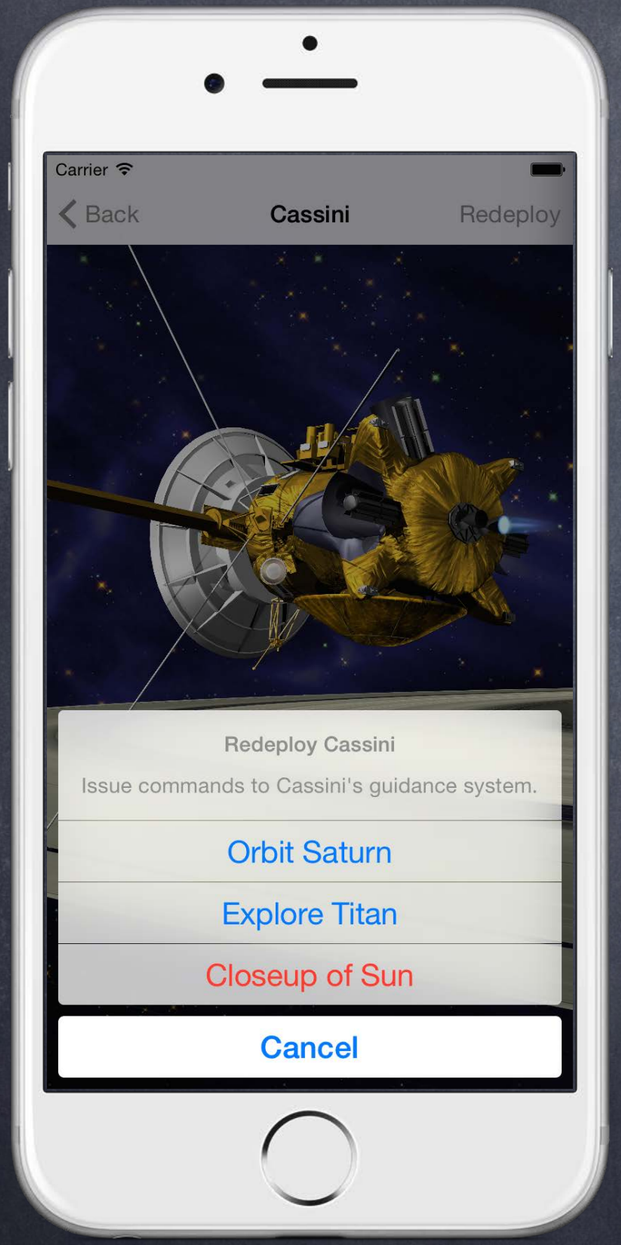



```
var alert = UIAlertController(
    title: “Redeploy Cassini”,
    message: “Issue commands to Cassini’s guidance system.”,
    preferredStyle: UIAlertControllerStyle.ActionSheet
)
```

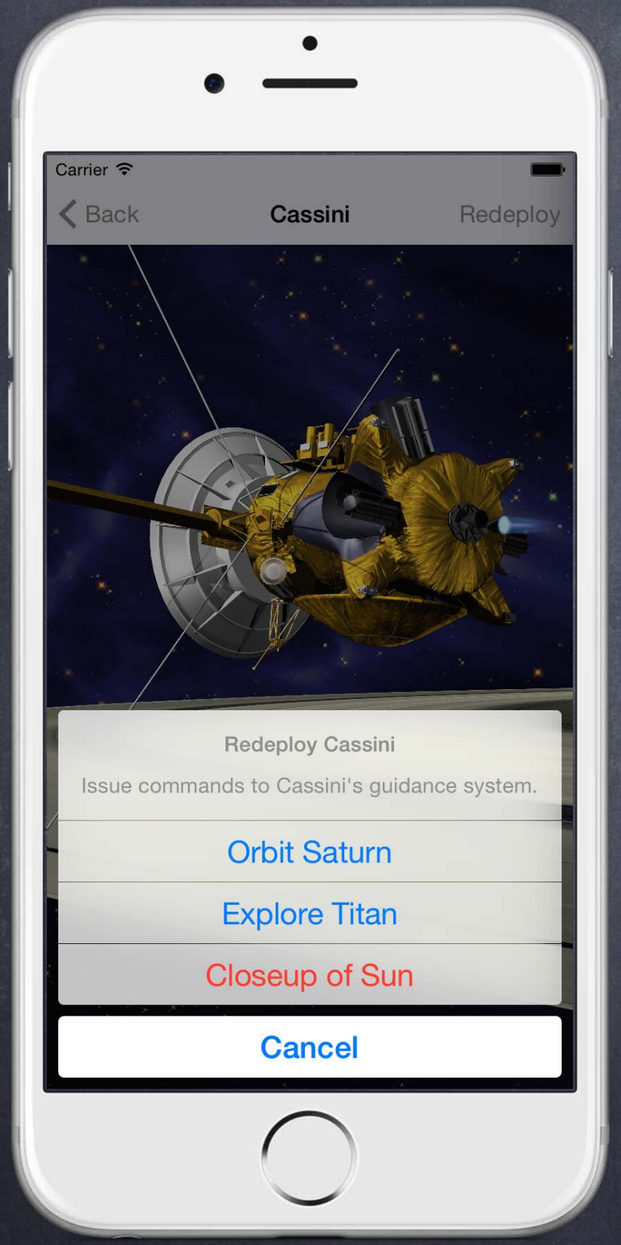



```
var alert = UIAlertController(
    title: “Redeploy Cassini”,
    message: “Issue commands to Cassini’s guidance system.”,
    preferredStyle: UIAlertControllerStyle.ActionSheet
)
```

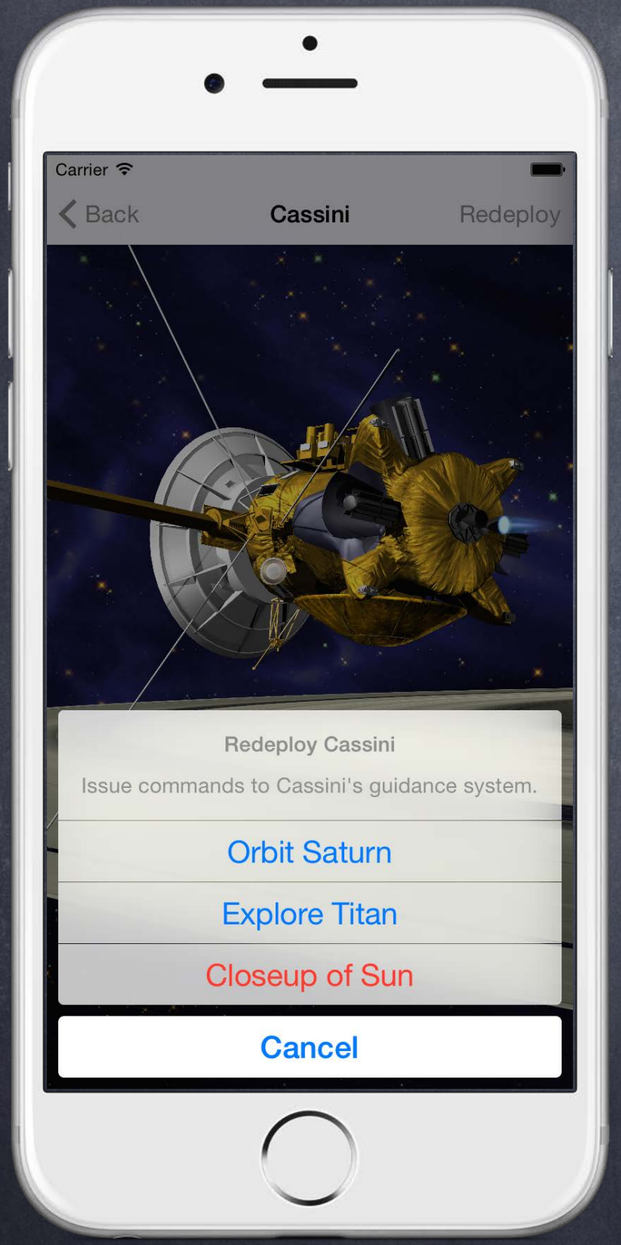
CS193p
Spring 2016



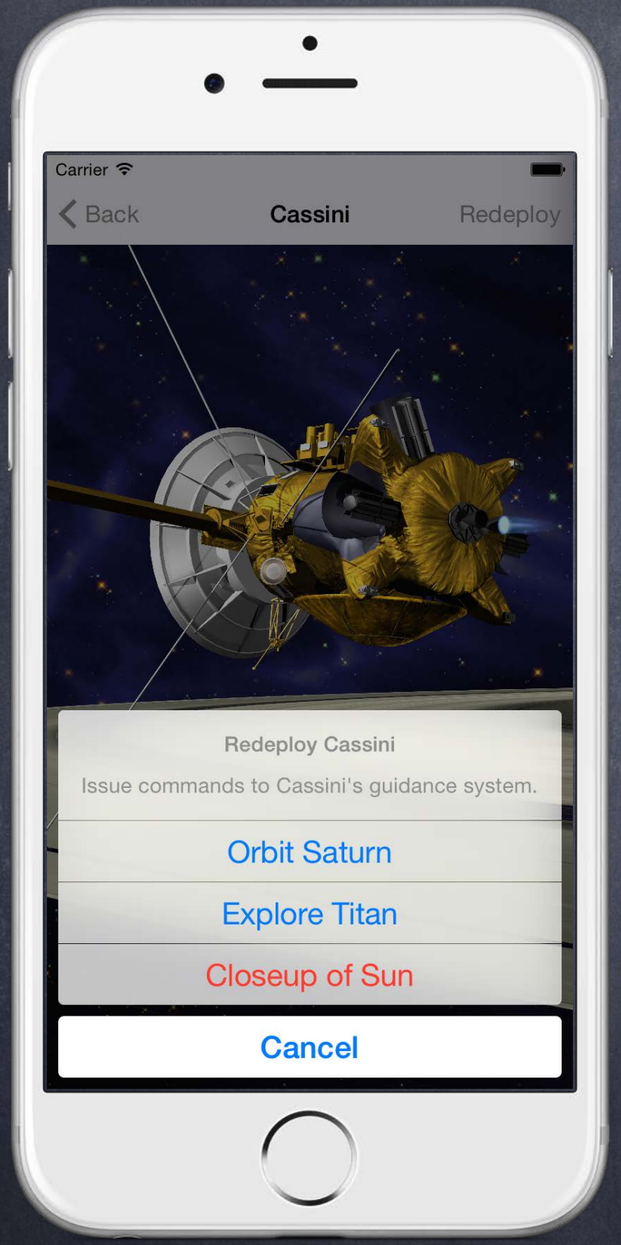

```
var alert = UIAlertController(
    title: “Redeploy Cassini”,
    message: “Issue commands to Cassini’s guidance system.”,
    preferredStyle: UIAlertControllerStyle.ActionSheet
)

alert.addAction(...)
```



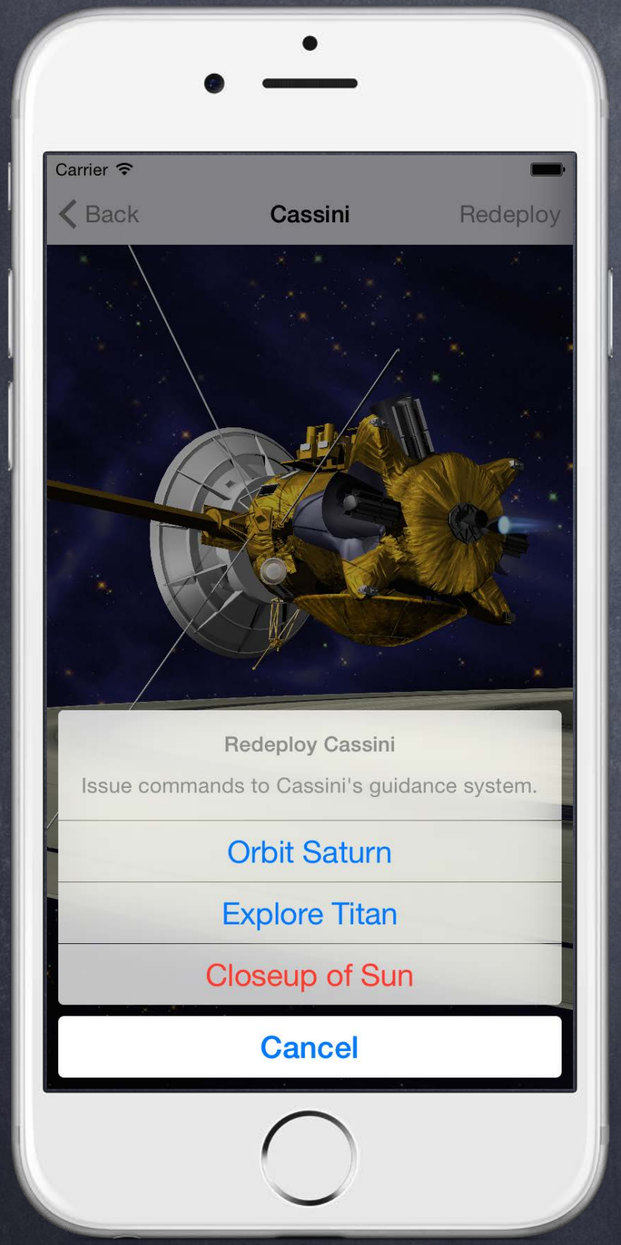

```
var alert = UIAlertController(
    title: “Redeploy Cassini”,
    message: “Issue commands to Cassini’s guidance system.”,
    preferredStyle: UIAlertControllerStyle.ActionSheet
)

alert.addAction(UIAlertAction(...))
```



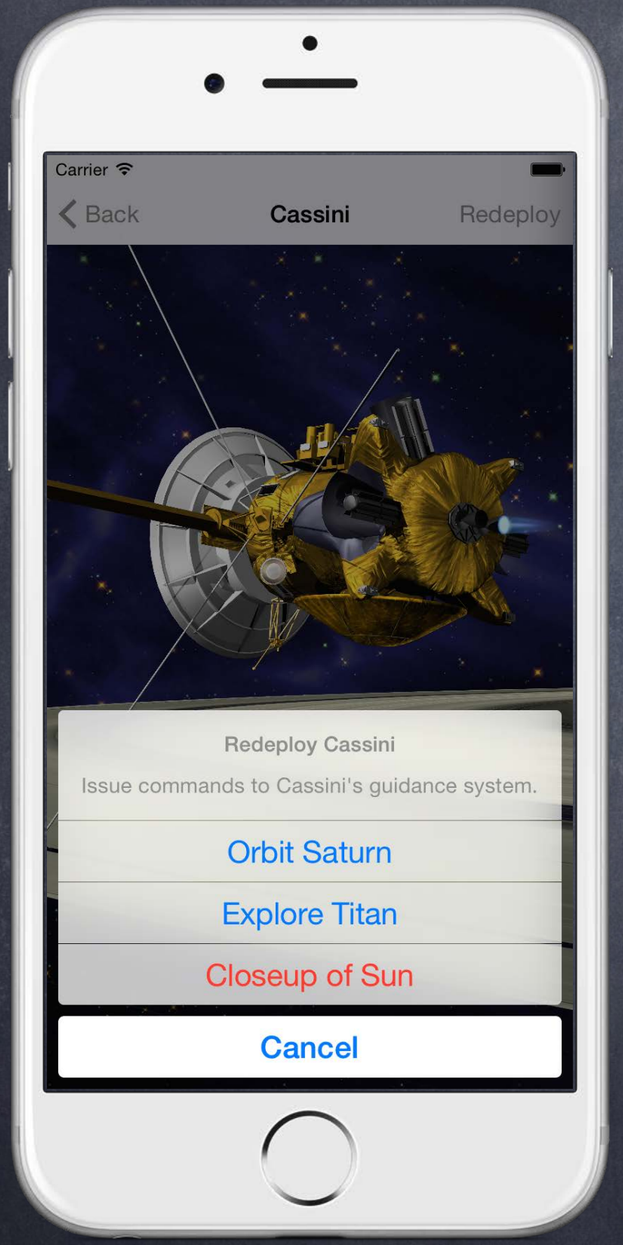

```
var alert = UIAlertController(
    title: “Redeploy Cassini”,
    message: “Issue commands to Cassini’s guidance system.”,
    preferredStyle: UIAlertControllerStyle.ActionSheet
)

alert.addAction(UIAlertAction(
    title: String,
    style: UIAlertActionStyle,
    handler: (action: UIAlertAction) -> Void
))
```



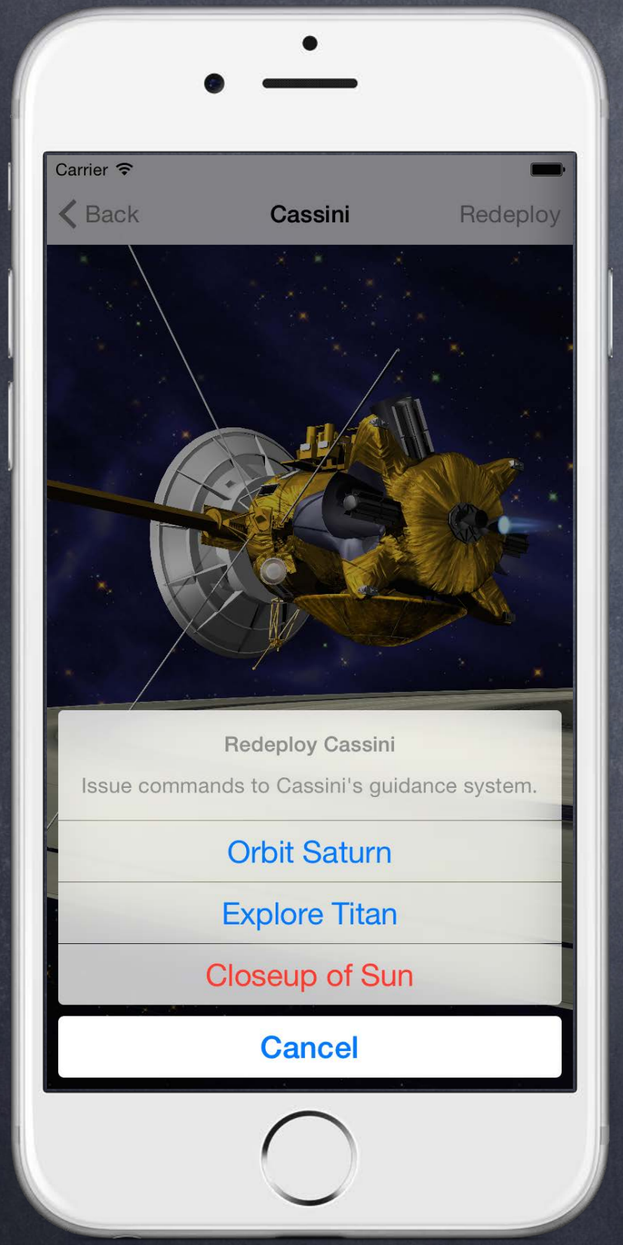

```
var alert = UIAlertController(
    title: “Redeploy Cassini”,
    message: “Issue commands to Cassini’s guidance system.”,
    preferredStyle: UIAlertControllerStyle.ActionSheet
)

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



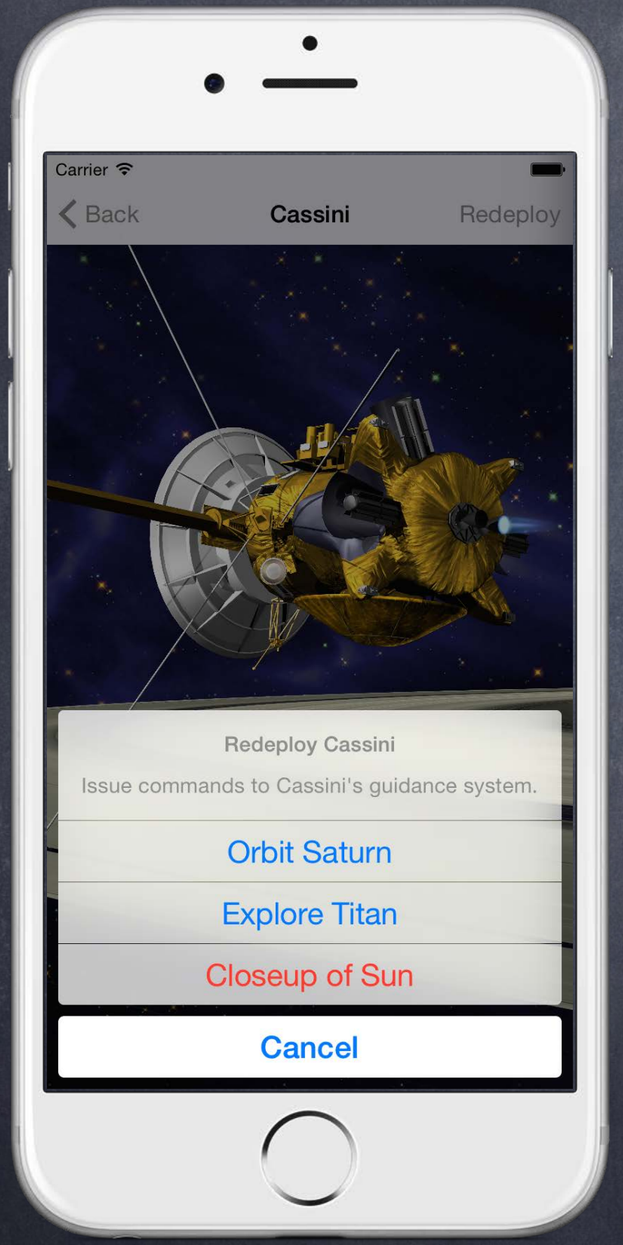

```
var alert = UIAlertController(
    title: “Redeploy Cassini”,
    message: “Issue commands to Cassini’s guidance system.”,
    preferredStyle: UIAlertControllerStyle.ActionSheet
)

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: UIAlertControllerStyle.ActionSheet
)

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

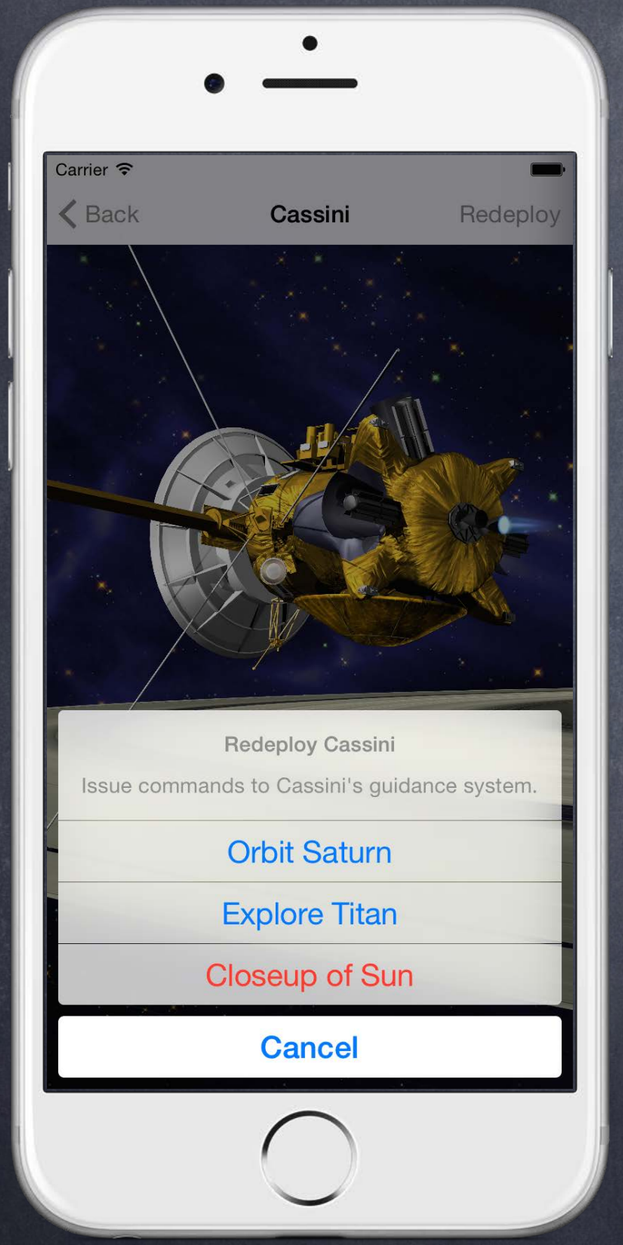



```
var alert = UIAlertController(
    title: “Redeploy Cassini”,
    message: “Issue commands to Cassini’s guidance system.”,
    preferredStyle: UIAlertControllerStyle.ActionSheet
)

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
    }
)
```

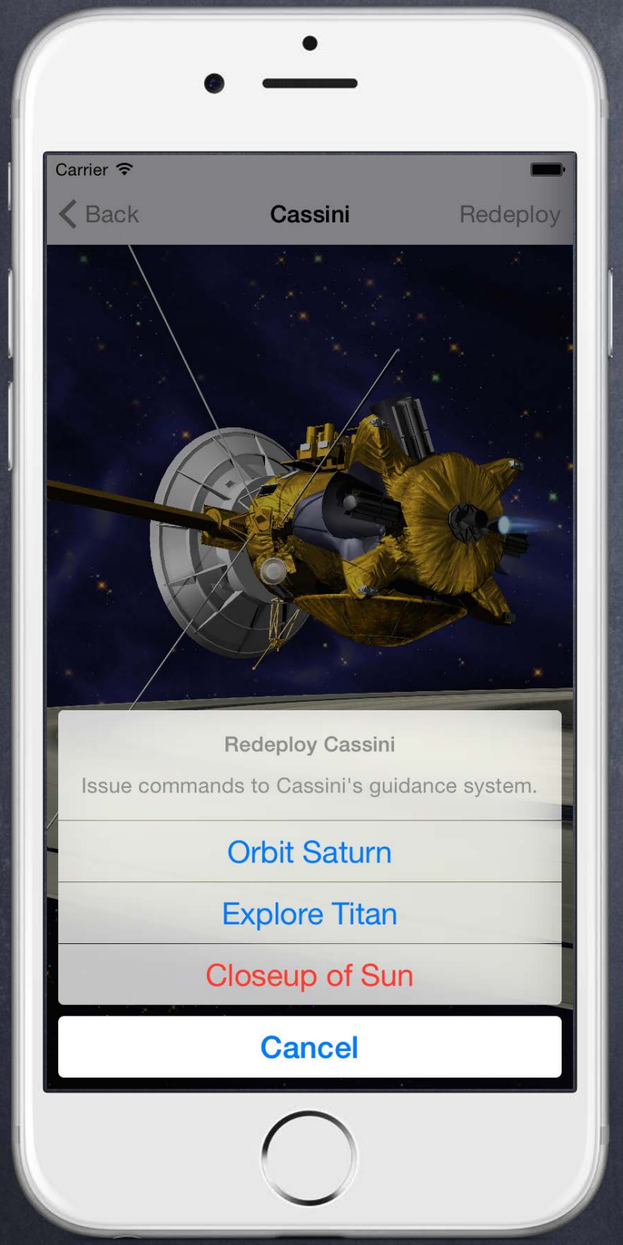



```
var alert = UIAlertController(
    title: “Redeploy Cassini”,
    message: “Issue commands to Cassini’s guidance system.”,
    preferredStyle: UIAlertControllerStyle.ActionSheet
)

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
    }
)
```

CS193p
Spring 2016

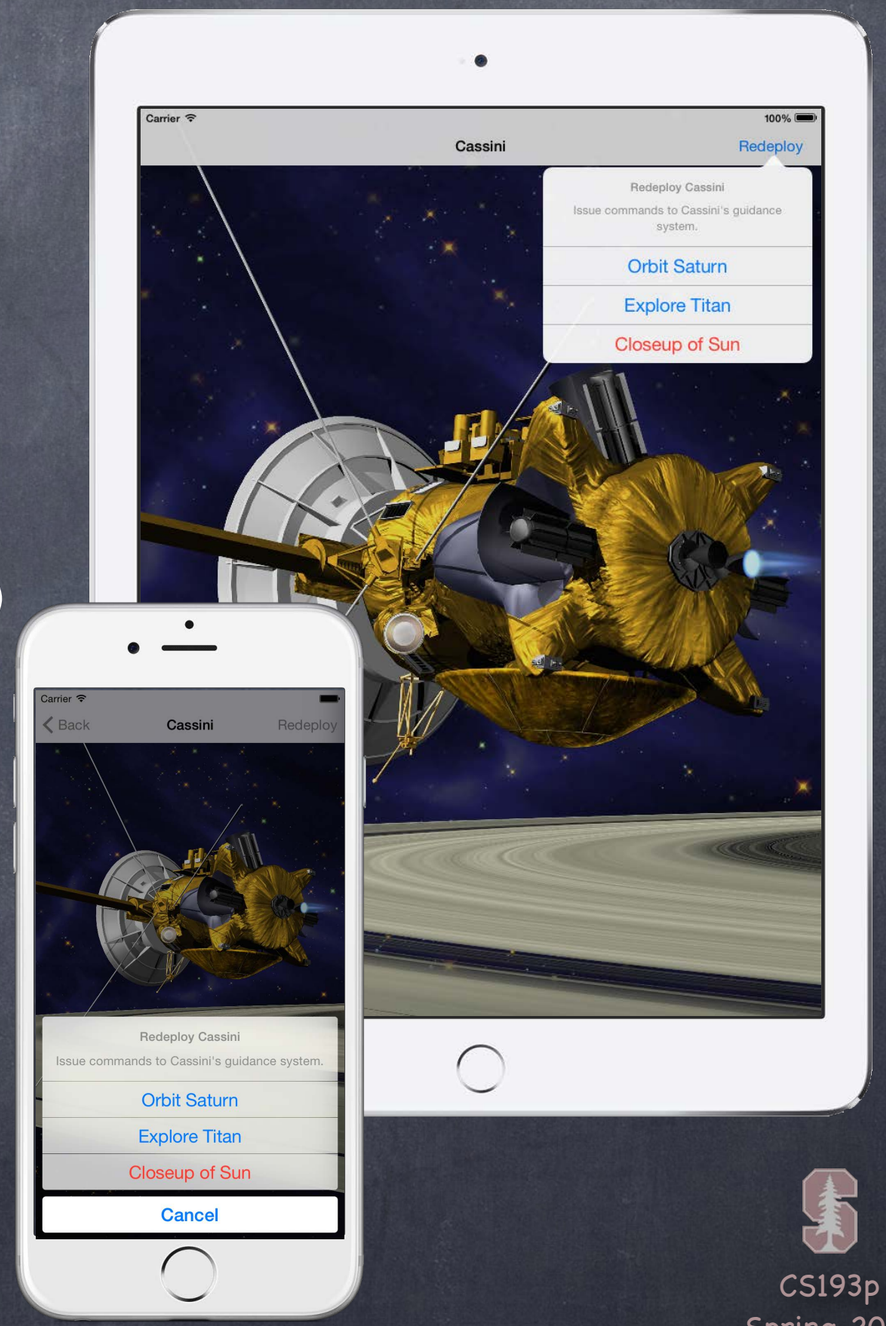



```
var alert = UIAlertController(
    title: “Redeploy Cassini”,
    message: “Issue commands to Cassini’s guidance system.”,
    preferredStyle: UIAlertControllerStyle.ActionSheet
)

alert.addAction(/* orbit saturn action */)
alert.addAction(/* explore titan action */)
alert.addAction(/* destroy with closeup of sun action */)
alert.addAction(/* do nothing cancel action */)
```




```
var alert = UIAlertController(
    title: “Redeploy Cassini”,
    message: “Issue commands to Cassini’s guidance system.”,
    preferredStyle: UIAlertControllerStyle.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 */)

presentViewController(alert, animated: true, completion: nil)
```

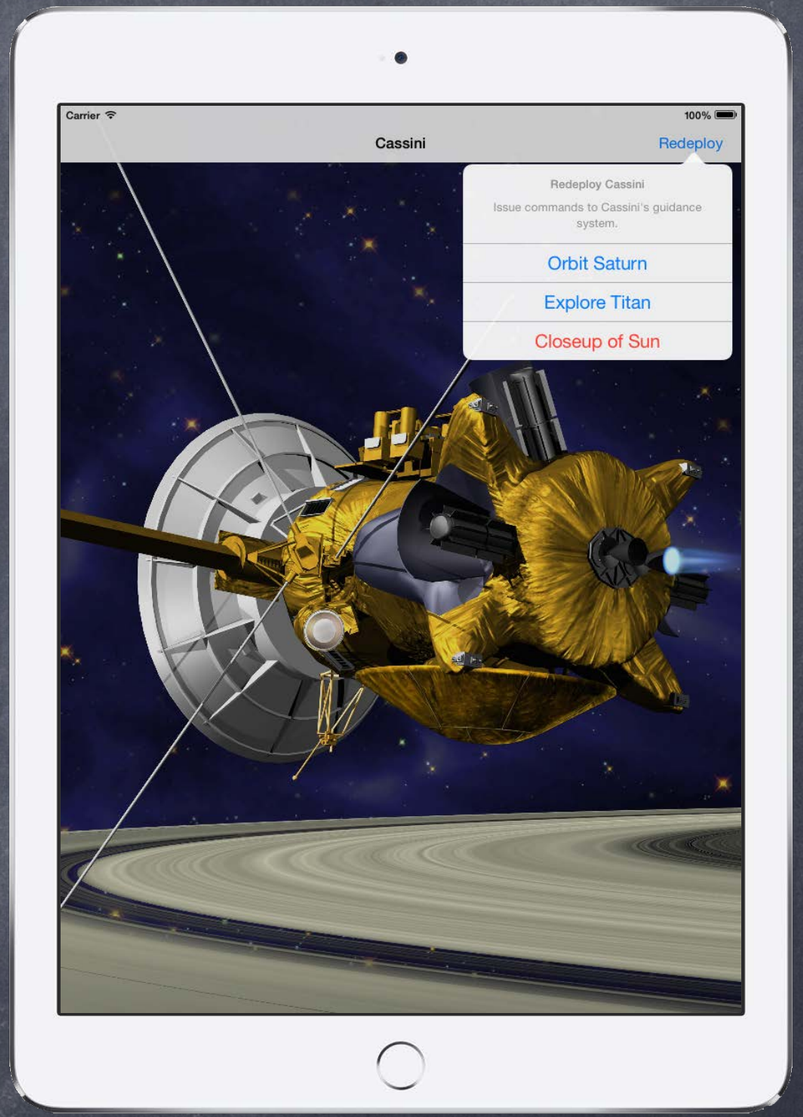
```
var alert = UIAlertController(
    title: “Redeploy Cassini”,
    message: “Issue commands to Cassini’s guidance system.”,
    preferredStyle: UIAlertControllerStyle.ActionSheet
)

alert.addAction(/* orbit saturn action */)
alert.addAction(/* explore titan action */)
alert.addAction(/* destroy with closeup of sun action */)
alert.addAction(/* do nothing cancel action */)

presentViewController(alert, animated: true, completion: nil)
```



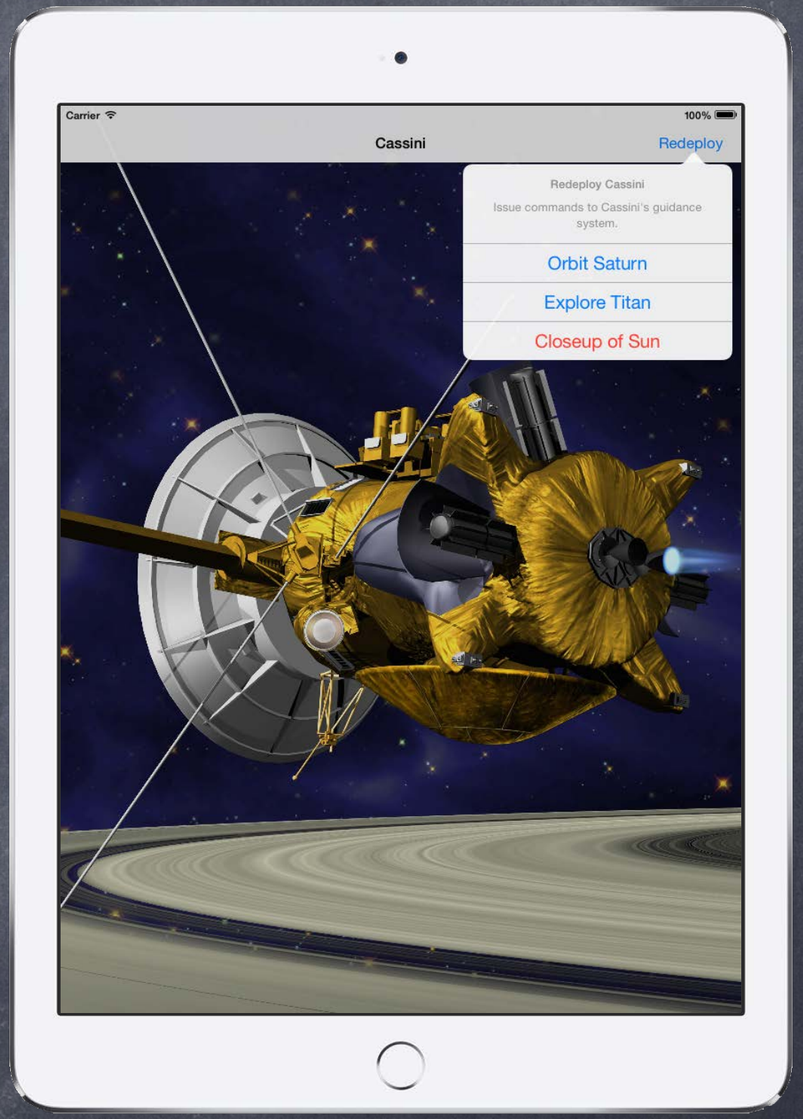



CS193p
Spring 2016

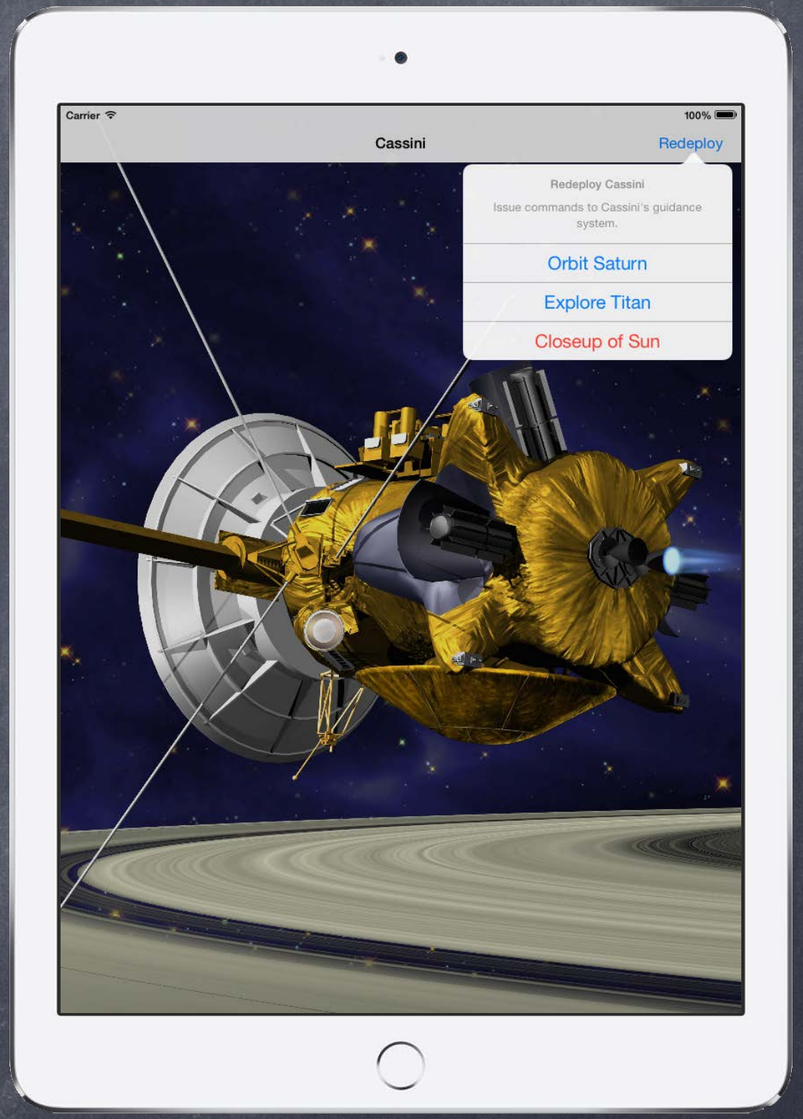
```
var alert = UIAlertController(
    title: “Redeploy Cassini”,
    message: “Issue commands to Cassini’s guidance system.”,
    preferredStyle: UIAlertControllerStyle.ActionSheet
)

alert.addAction(/* orbit saturn action */)
alert.addAction(/* explore titan action */)
alert.addAction(/* destroy with closeup of sun action */)
alert.addAction(/* do nothing cancel action */)



presentViewController(alert, animated: true, completion: nil)
```



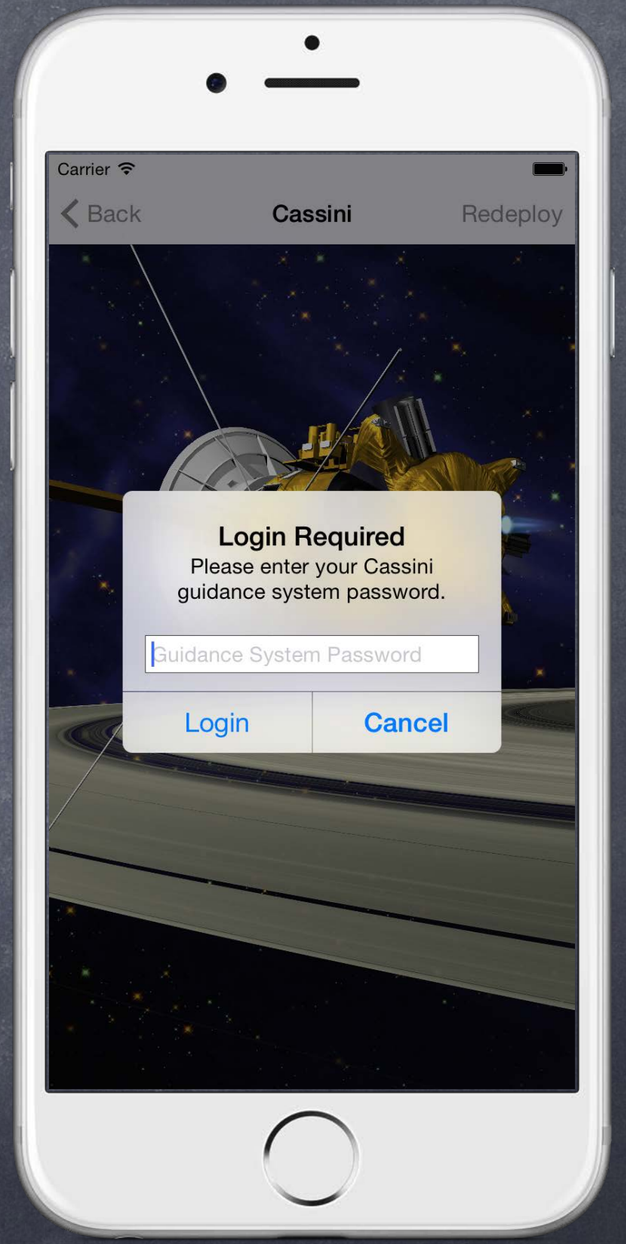

```
var alert = UIAlertController(
    title: “Redeploy Cassini”,
    message: “Issue commands to Cassini’s guidance system.”,
    preferredStyle: UIAlertControllerStyle.ActionSheet
)

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


presentViewController(alert, animated: true, completion: nil)
```



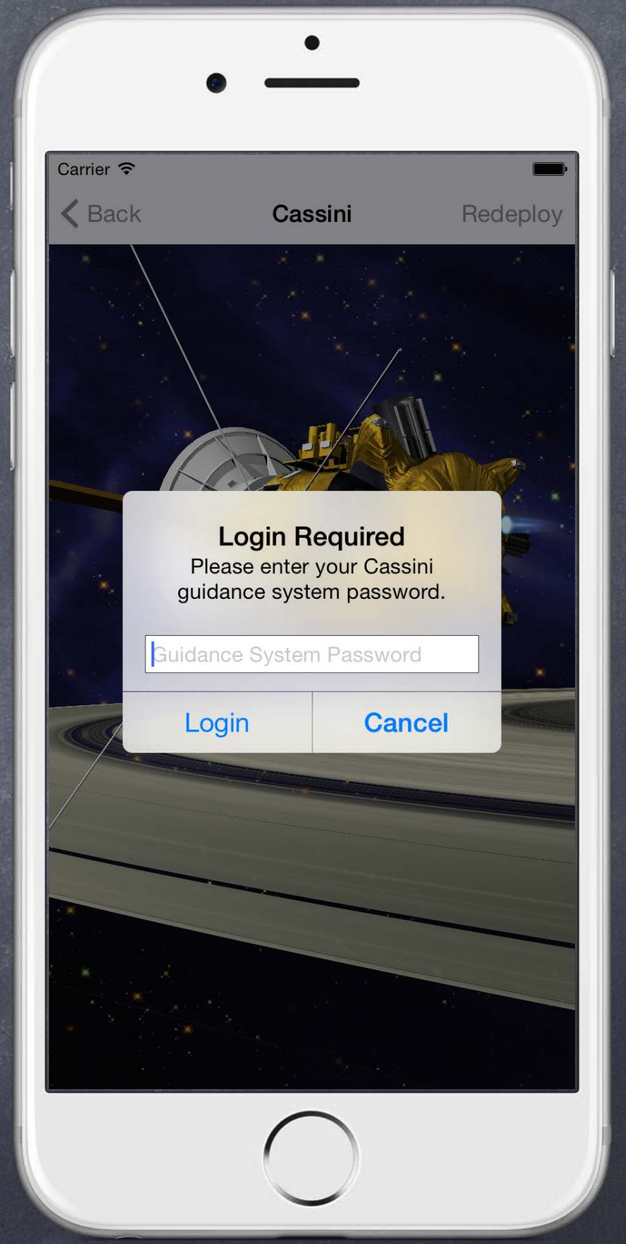

```
var alert = UIAlertController(
    title: “Redeploy Cassini”,
    message: “Issue commands to Cassini’s guidance system.”,
    preferredStyle: UIAlertControllerStyle.ActionSheet
)

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

presentViewController(alert, animated: true, completion: nil)
```



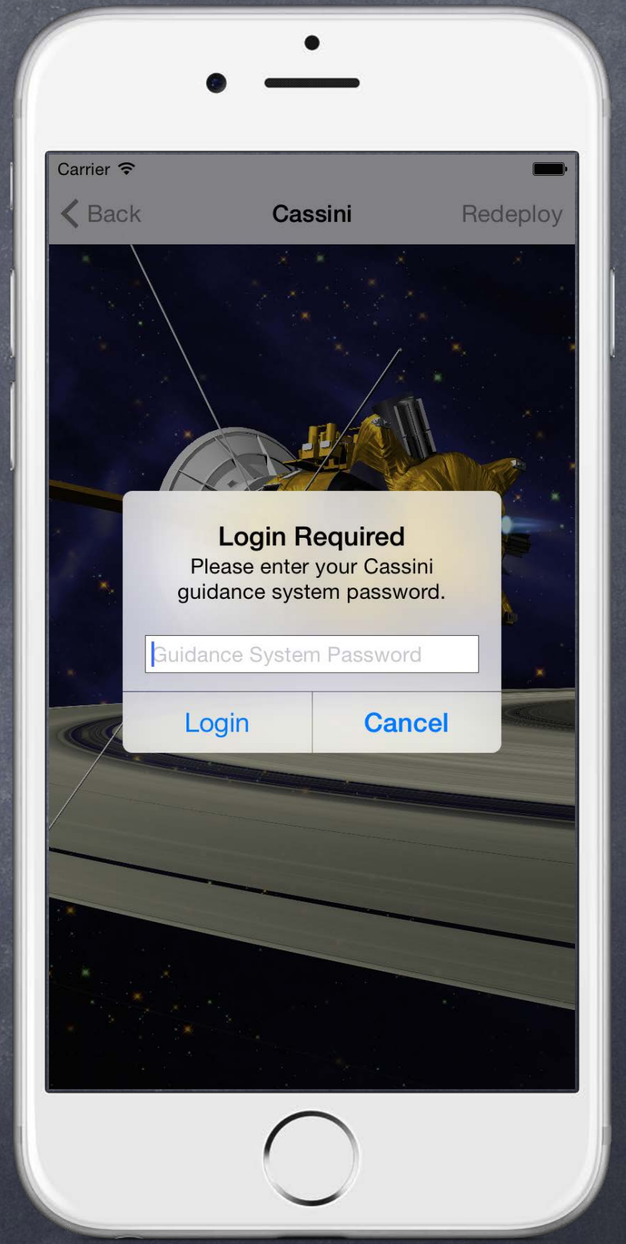

```
var alert = UIAlertController(
    title: “Login Required”,
    message: “Please enter your Cassini guidance system...”,
    preferredStyle: UIAlertControllerStyle.Alert
)
```



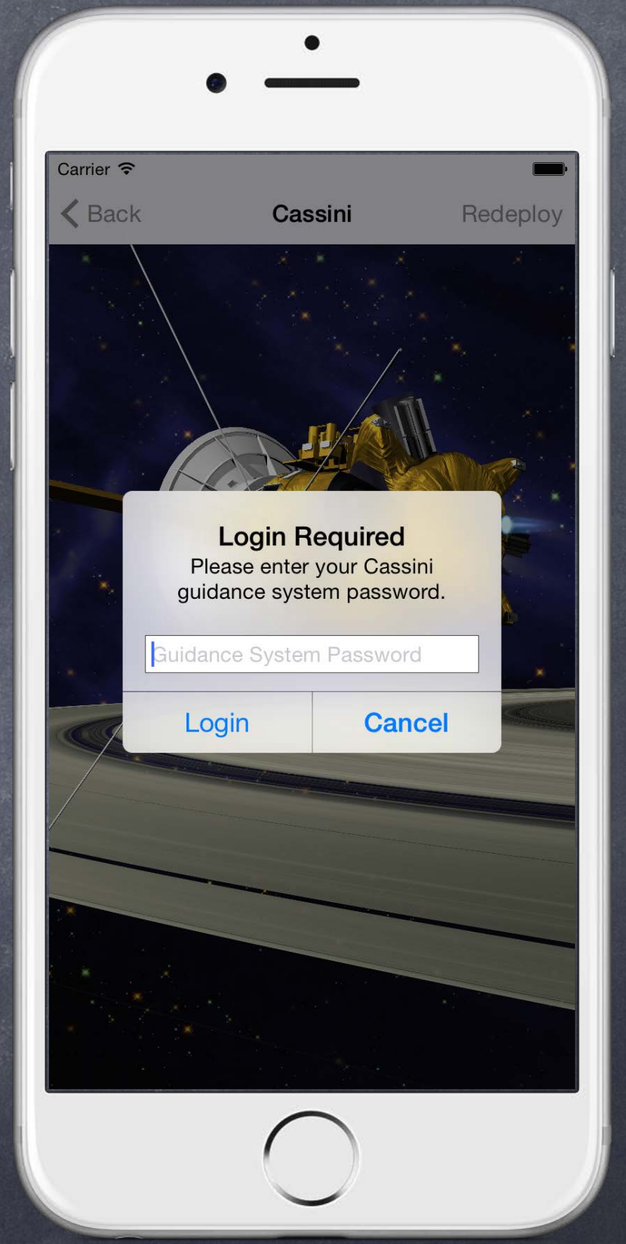
```
var alert = UIAlertController(
    title: “Login Required”,
    message: “Please enter your Cassini guidance system...”,
    preferredStyle: UIAlertControllerStyle.Alert
)

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



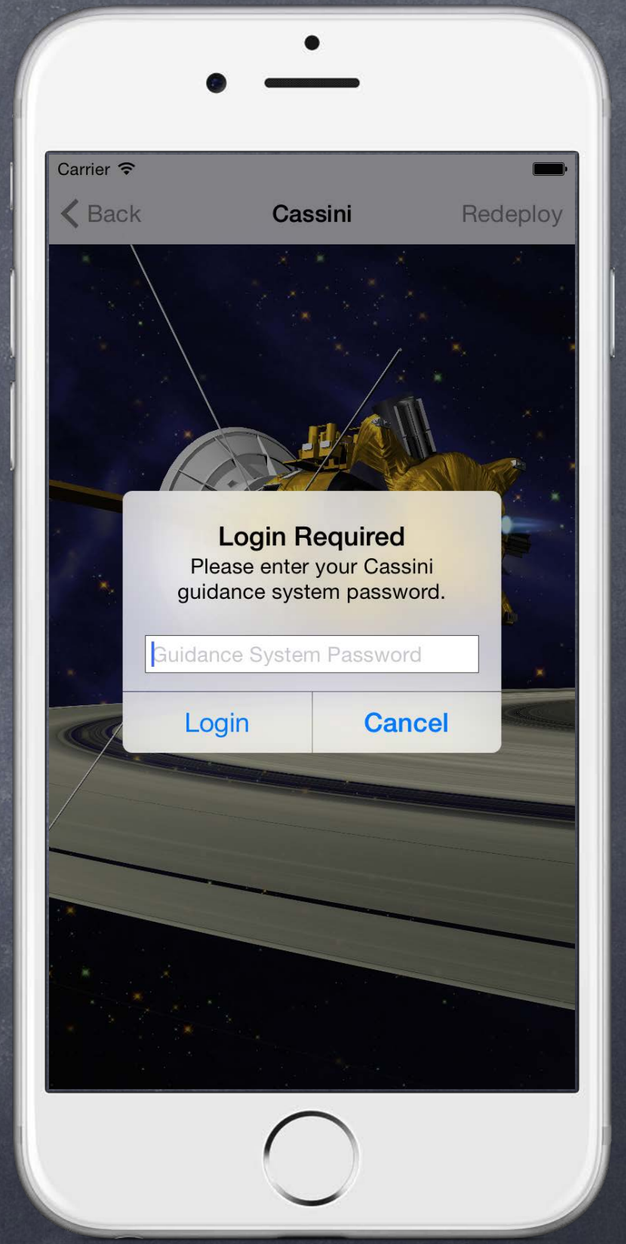
```
var alert = UIAlertController(
    title: “Login Required”,
    message: “Please enter your Cassini guidance system...”,
    preferredStyle: UIAlertControllerStyle.Alert
)

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



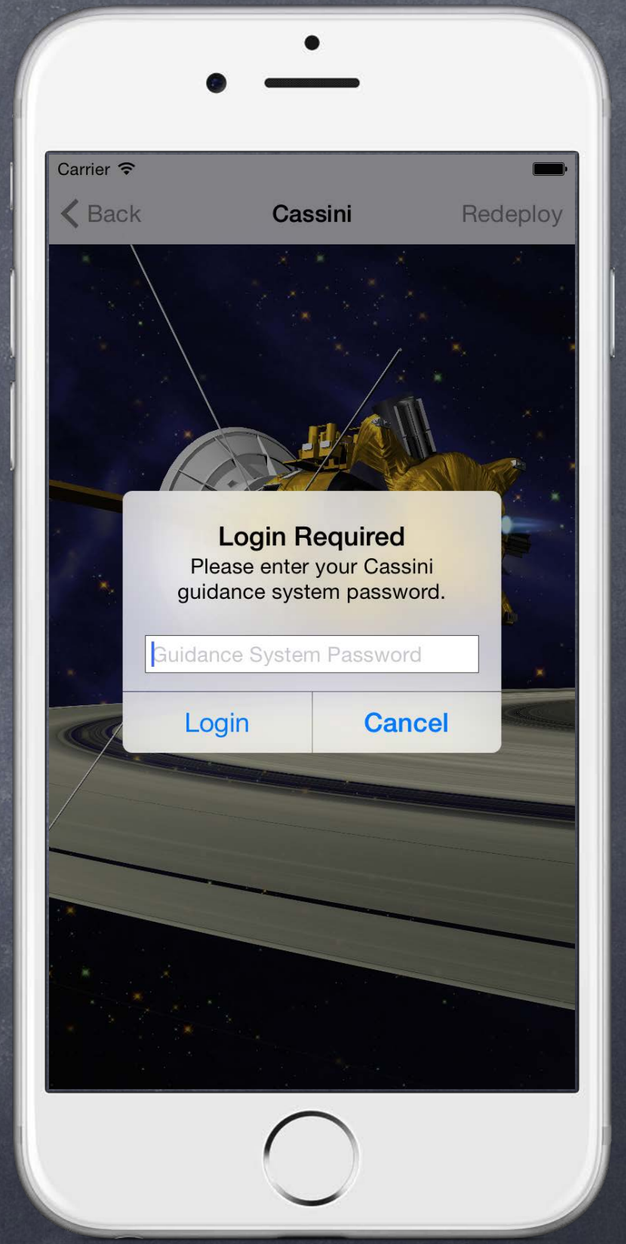
```
var alert = UIAlertController(
    title: “Login Required”,
    message: “Please enter your Cassini guidance system...”,
    preferredStyle: UIAlertControllerStyle.Alert
)

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

alert.addAction(UIAlertAction(
    title: “Login”,
    style: .Default)
    { (action: UIAlertAction) -> Void in
        // get password and log in


    }
)
```



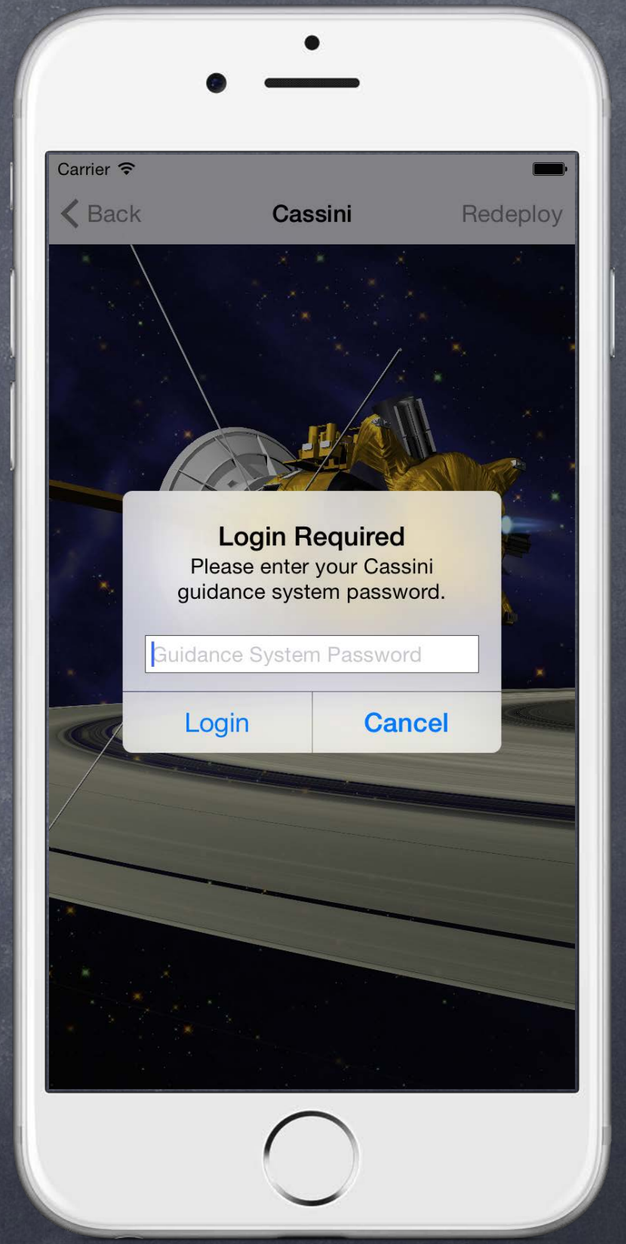
```
var alert = UIAlertController(
    title: “Login Required”,
    message: “Please enter your Cassini guidance system...”,
    preferredStyle: UIAlertControllerStyle.Alert
)

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

alert.addAction(UIAlertAction(
    title: “Login”,
    style: .Default)
    { (action: UIAlertAction) -> Void in
        // get password and log in


    }
)


alert.addTextFieldWithConfigurationHandler { (textField) in
    textField.placeholder = “Guidance System Password”
}
```

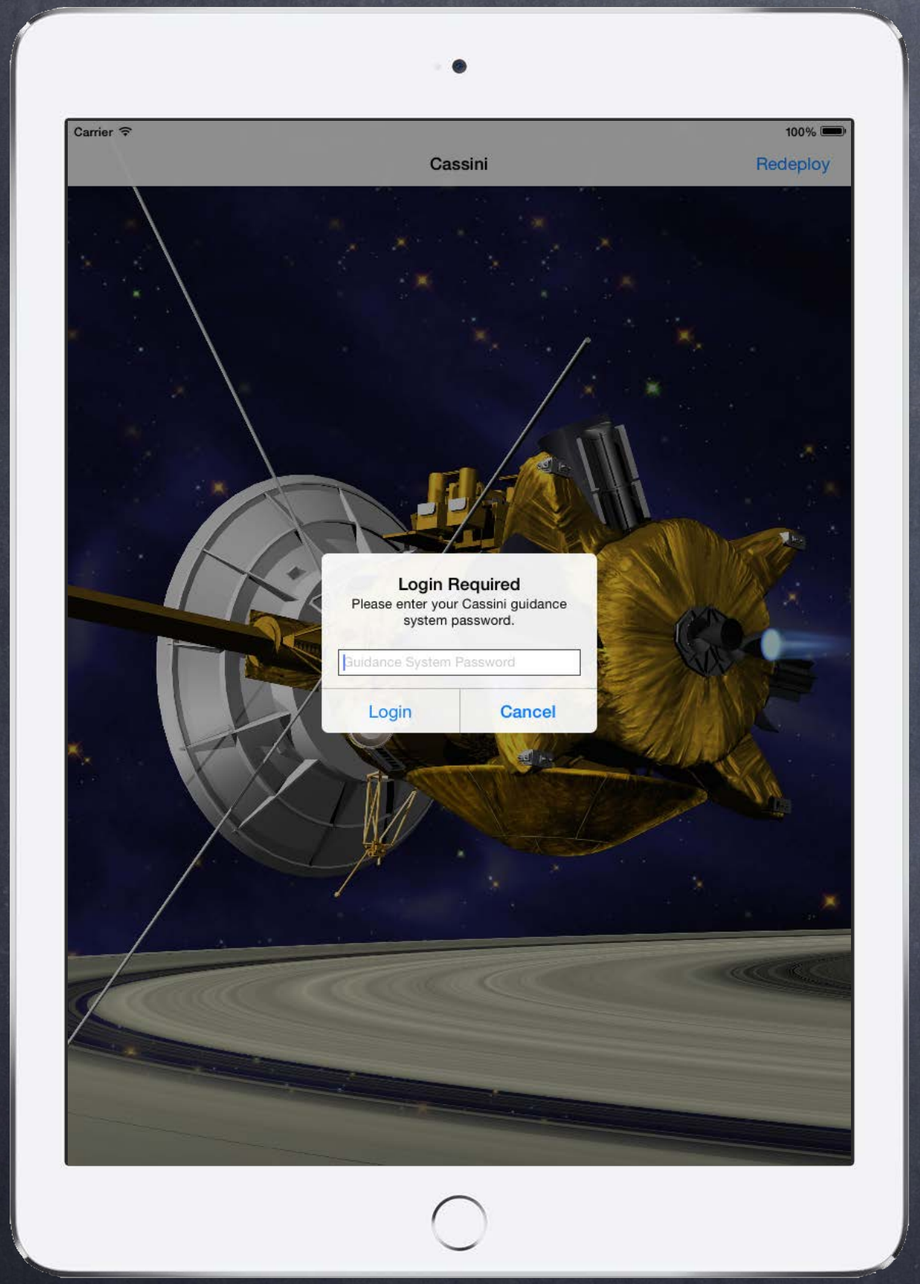


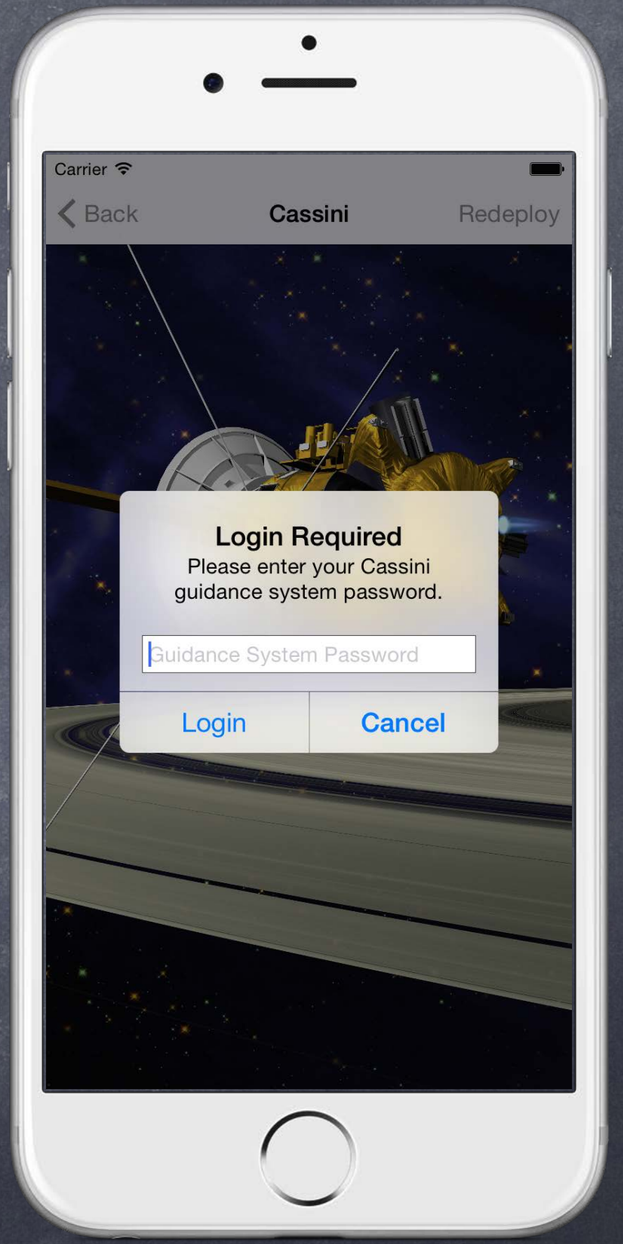

```
var alert = UIAlertController(
    title: “Login Required”,
    message: “Please enter your Cassini guidance system...”,
    preferredStyle: UIAlertControllerStyle.Alert
)

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.addTextFieldWithConfigurationHandler { (textField) in
    textField.placeholder = “Guidance System Password”
}
```




```
var alert = UIAlertController(
    title: “Login Required”,
    message: “Please enter your Cassini guidance system...”,
    preferredStyle: UIAlertControllerStyle.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.addTextFieldWithConfigurationHandler { (textField) in
    textField.placeholder = “Guidance System Password”
}

presentViewController(alert, animated: true, completion: nil)
```



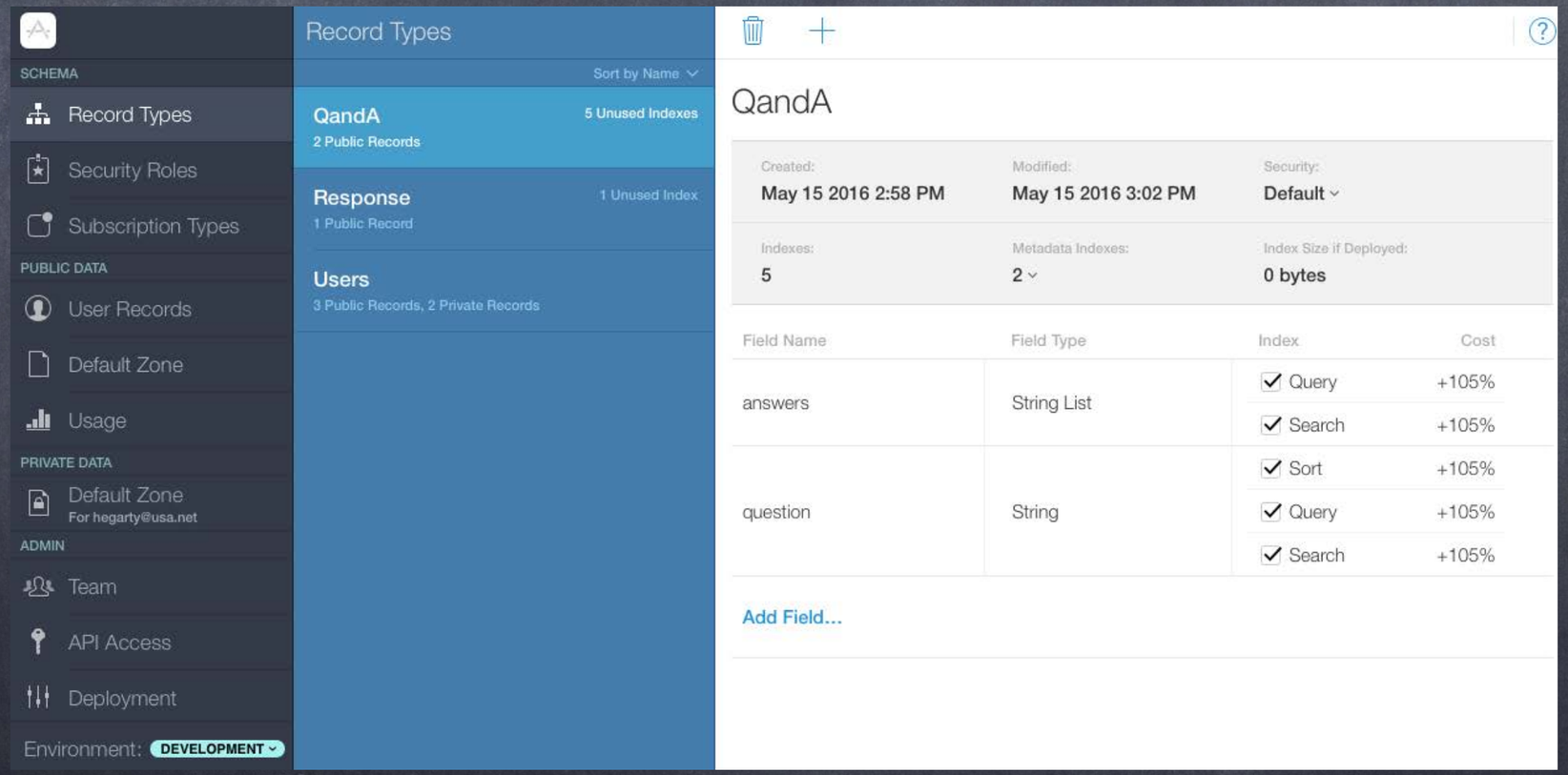

CS193p
Spring 2016

# Cloud Kit

- Cloud Kit
  - A database in the cloud.
  - Simple to use, but with very basic "database" operations.
  - Since it's on the network, accessing the database could be slow or even impossible.
  - This requires some thoughtful programming.
- Important Concepts
  - Record Type - like an Entity in Core Data
  - Fields - like Attributes in Core Data
  - Record - an "instance" of a Record Type
  - Reference - like a Relationship in Core Data
  - Database - a place where Records are stored
  - Zone - a sub-area of a Database
  - Container - collection of Databases
  - Query - an NSPredicate-based database search
  - Subscription - a "standing Query" which sends push notifications when changes occur

# Cloud Kit

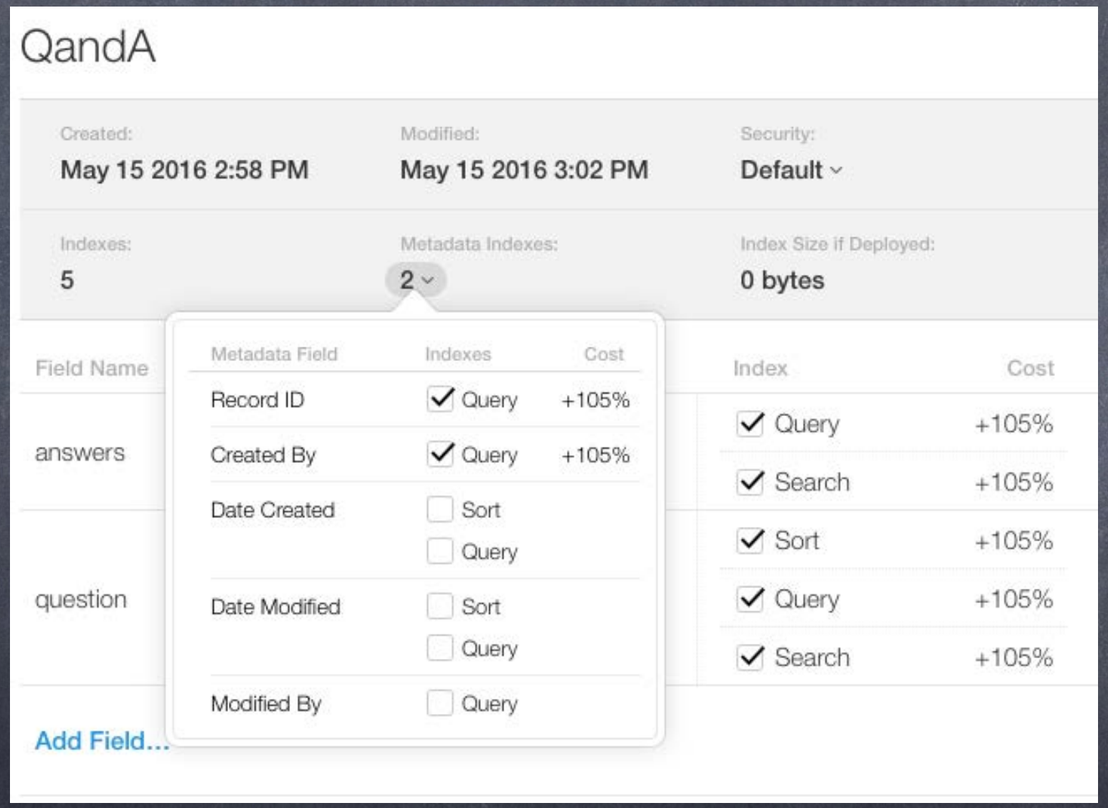
- Cloud Kit Dashboard
  - A web-based UI to look at everything you are storing.
  - Shows you all your Record Types and Fields as well as the data in Records.
  - You can add new Record Types and Fields and also turn on/off indexes for various Fields.




# Cloud Kit

- Cloud Kit Dashboard

  Note that you can turn on indexes for meta data too (like who created the Record or when).

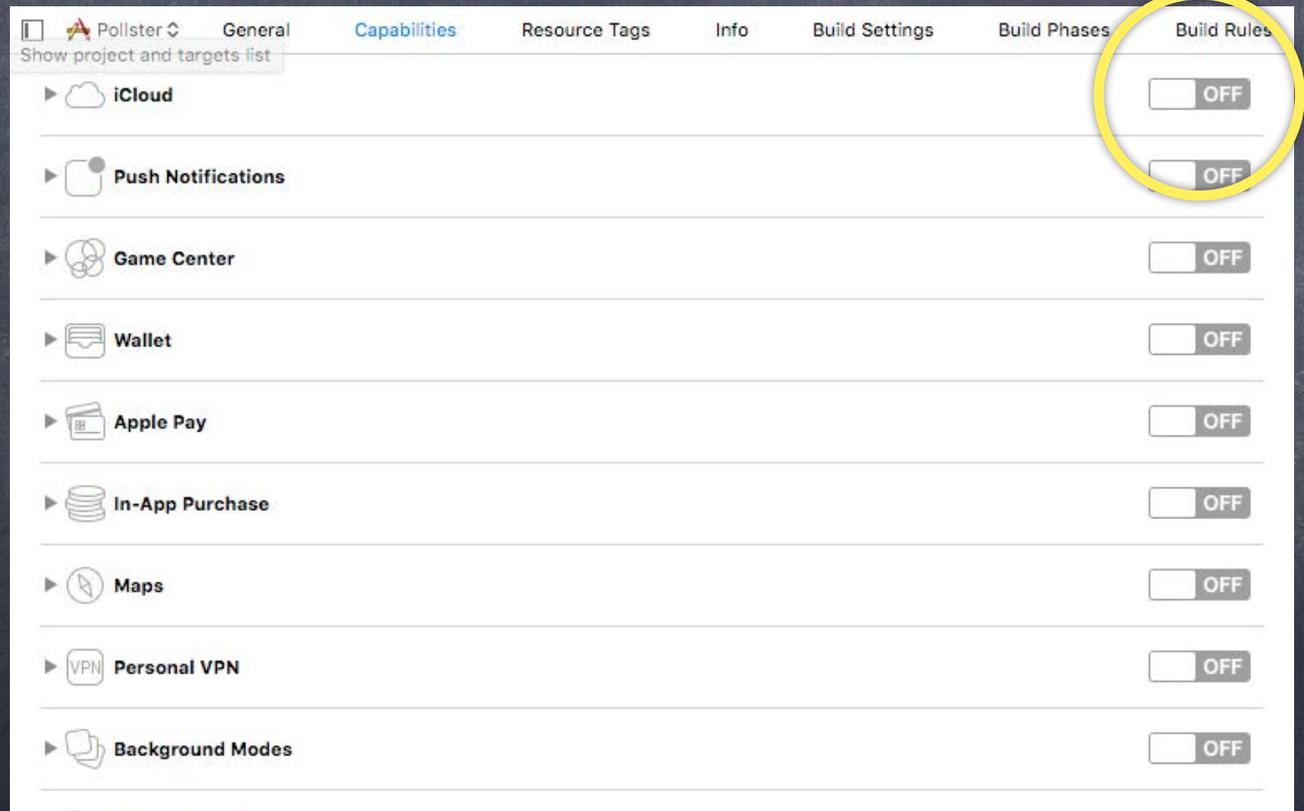



# Cloud Kit

- Dynamic Schema Creation
  - But you don't have to create your schema in the Dashboard.
  - You can create it "organically" by simply creating and storing things in the database.
  - When you store a record with a new, never-before-seen Record Type, it will create that type.
  - Or if you add a Field to a Record, it will automatically create a Field for it in the database.
  - This only works during Development, not once you deploy to your users.

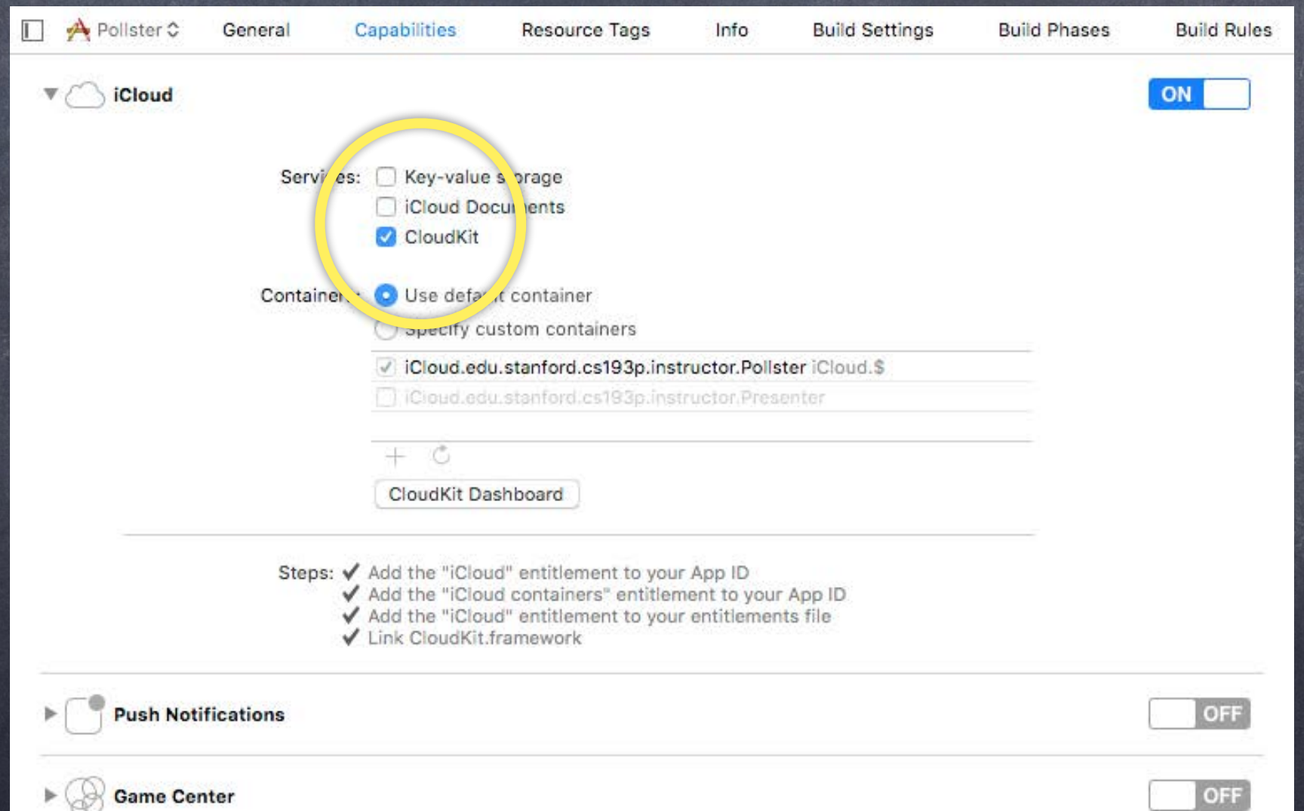
# Cloud Kit

- Nothing will work until you enable iCloud in your Project
  - Go to your Project Settings and, under Capabilities, turn on iCloud.



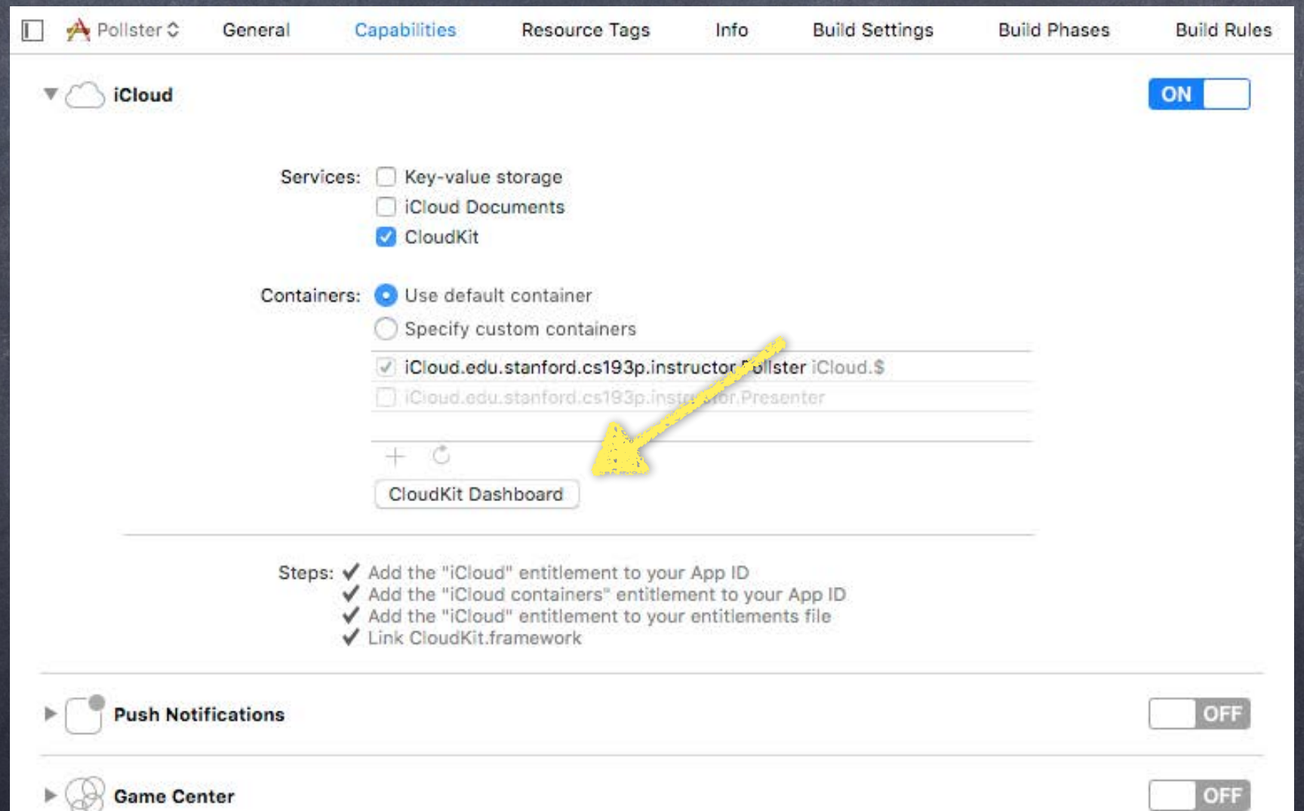

# Cloud Kit

- Nothing will work until you enable iCloud in your Project
  - Go to your Project Settings and, under Capabilities, turn on iCloud.
  - Then, choose CloudKit from the iCloud Services.



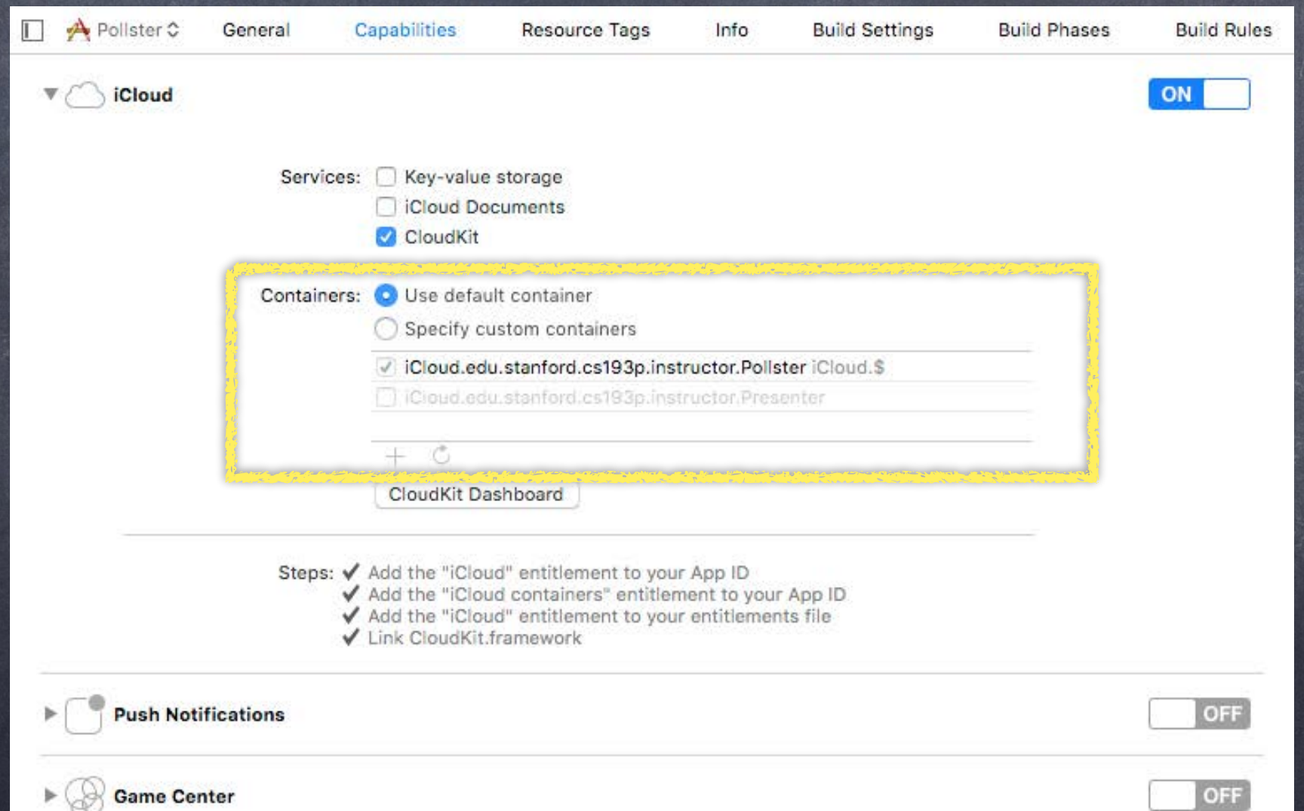

# Cloud Kit

- Nothing will work until you enable iCloud in your Project
  - Go to your Project Settings and, under Capabilities, turn on iCloud.
  - Then, choose CloudKit from the iCloud Services.
  - You'll also see a CloudKit Dashboard button which will take you to the Dashboard.



# Cloud Kit

- Nothing will work until you enable iCloud in your Project
  - Go to your Project Settings and, under Capabilities, turn on iCloud.
  - Then, choose CloudKit from the iCloud Services.
  - You'll also see a CloudKit Dashboard button which will take you to the Dashboard.
  - We're only going to cover the default Container (1 public and 1 private database).




# Cloud Kit

- How to create Records in the Database

  First you need to get a Database.

  You find them in a Container.

  We're only going to take a look at how to use the default Container.

  Inside this Container are two Databases we can use: a public one and a private one.

  The data in the public one can be seen by all people running your application.

  The data in the private one can only be seen by the logged-in iCloud user at the time.

```
let publicDatabase = CKContainer.defaultContainer().publicCloudDatabase
let privateDatabase = CKContainer.defaultContainer().privateCloudDatabase
```

# Cloud Kit

- How to create Records in the Database

  Next you create a Record using the `CKRecord` class.

  ```
  let record = CKRecord(recordType: String)
  ```

  The `recordType` is the name of the thing you want to create (like an Entity name in CoreData).
  Again, you do not have to go to your Dashboard first to create this Record Type.
  It will get created for you on the fly.

# Cloud Kit

- How to create Records in the Database

  Now add Attribute-like things to it using a `Dictionary`-like syntax (created on the fly too) …

```
let record = CKRecord(recordType: String)
record[“text”] = …
record[“created”] = …
record[“items”] = [ …, …, … ]
```

  So what can the … values be? Things in iOS that conform to the `CKRecordValue` protocol.
  Namely … `NSString`, `NSNumber`, `NSArray` and `NSDate` (and bridged Swift types)
  Also … `CKReference`, `CKAsset`, `CLLocation`

  `CKReference` is a reference to another record (sort of like a relationship in Core Data).
  `CKAsset` is for large files (an image or sound or video).
  `CLLocation` is a gps coordinate (we'll talk more about those next week).
  Arrays can be arrays of any of the other `CKRecordValue` types.

# Cloud Kit

- How to create Records in the Database

  You also get the values using Dictionary-like syntax.

  But note that the type returned is `CKRecordValue`, so you'll have to cast it with `as?`.

```
let theText = record["text"] as? String
let createdDate = record["created"] = as? NSDate
let items = record["items"] as? [String]
let amount = record["cost"] as? Double
```

  All of the above `lets` would of course be Optionals since we're using `as?` to cast them.

# Cloud Kit

- How to create Records in the Database

  Now you can store the Record in a Database (public or private).
  This is done using an `NSOperation` (the object-oriented GCD dispatching API).
  Specifically, a subclass called `CKModifyRecordsOperation`.
  You just create one (the initializer takes an array of records to save or delete),
  Then put it on an `NSOperationQueue` to start it up.
  But if you just want to do a simple save, there's a convenience method for it in `CKDatabase`:

```
func saveRecord(record: CKRecord, completionHandler: (CKRecord?, NSError?) -> Void)
```

  There are some things this convenience method can't do that `CKModifyRecordsOperation` can
  (like overwriting newer versions of the record, writing multiple records at once, etc.),
  but this convenience method works for the vast majority of saves.

# Cloud Kit

- How to create Records in the Database

  All this talk of `NSOperation` and seeing `completionHandler` is a dead giveaway that ...
  Saving Records in the Database is done asychronously (obviously, since it's over the network).
  In fact, all interactions with the Database are asynchronous.

  Architect your code appropriately!

  Most notably, the closures you give these methods are NOT executed on the main queue!
  So be sure to `dispatch_async(dispatch_get_main_queue()) { }` to do any UI work.

# Cloud Kit

- How to create Records in the Database

  Example …

```
let tweet = CKRecord("Tweet")
tweet["text"] = "140 characters of pure joy"
let db = CKContainer.defaultContainer().publicCloudDatabase
db.saveRecord(tweet) { (savedRecord: CKRecord?, error: NSError?) -> Void in
    if error == nil {
        // hooray!
    } else if error?.errorCode == CKErrorCode.NotAuthenticated.rawValue {
        // tell user he or she has to be logged in to iCloud for this to work!
    } else {
        // report other errors (there are 29 different CKErrorCodes!)
    }
}
```

  This will automatically create the `Tweet` "Entity" and `text` "Attribute" in the database schema.

# Cloud Kit

- How to create Records in the Database

  Some errors that come back also have a "retry" interval specified.

  Likely you will want to set an `NSTimer` to try again after this interval.

```
db.saveRecord(record) { (savedRecord: CKRecord?, error: NSError?) -> Void in
    if let retry = error?.userInfo[CKErrorRetryAfterKey] as? NSTimeInterval {
        dispatch_async(dispatch_get_main_queue()) {
            NSTimer.scheduledTimerWithTimeInterval(
                retry, target: …, selector: …, userInfo: …, repeats: …
            )
        }
    }
}
```

  Don't forget that you need to start `NSTimer` on the main queue.

# Cloud Kit

- Retrieving Records from the Database

  Fetching Records also is done via NSOperation (CKFetchRecordsOperation).
  But there is a convenience method in CKDatabase for fetching as well …

```
func performQuery(
    query: CKQuery,
    inZoneWithID: String,
    completionHandler: (records: [CKRecord]?, error: NSError?) -> Void
)
```

  The CKQuery is created with its init(recordType: String, predicate: NSPredicate)

  Ah, there's our old friend NSPredicate!
  Of course, since this database is a lot simpler than Core Data, the predicate must be too.
  See the CKQuery documentation for a full list.
  If you use the special "Field name" self in your predicate, it will search all indexed Fields.

# Cloud Kit

- Retrieving Records from the Database

  Example …

```
let db = CKContainer.defaultContainer().publicCloudDatabase
let predicate = NSPredicate(format: “text contains %@“, searchString)
let query = CKQuery(recordType: “Tweet”, predicate: predicate)
db.performQuery(query) { (records: [CKRecord]?, error: NSError?) in
    if error == nil {
        // records will be an array of matching CKRecords
    } else if error?.errorCode == CKErrorCode.NotAuthenticated.rawValue {
        // tell user he or she has to be logged in to iCloud for this to work!
    } else {
        // report other errors (there are 29 different CKErrorCodes!)
    }
}
```

# Cloud Kit

- Retrieving Records from the Database

  A special kind of `CKRecord` is for the user who is currently logged in to iCloud.
  You get that `CKRecord` using this method in `CKContainer` …

```
func fetchUserRecordIDWithCompletionHandler(
    completionHandler: (recordID: CKRecordID?, error: NSError?) -> Void
)
```

  This is useful mostly as a reference to the user to be stored in Fields in the database.
  The `CKRecordID.recordName` can function as sort of a "blind login name" for the user.

# Cloud Kit

- Retrieving Records from the Database

  And all Records have a Field called `creatorUserRecordID`.
  It is the `CKRecordID` of the user who created the Record.

  For example, to get all the Tweets created by the currently-logged in iCloud user …

```
CKContainer.defaultContainer().fetchUserRecordIDWithCompletionHandler
{ (userRecordID: CKRecordID?, error: NSError?) in
    let predicate = NSPredicate(format: “creatorUserRecordID = %@“, userRecordID)
    let query = CKQuery(recordType: “Tweet”, predicate: predicate)
    let database = CKContainer.defaultContainer().publicCloudDatabase
    database.performQuery(query, inZoneWithID: nil) { (records, error) in
        // records would contain all Tweets created by the currently logged in iCloud user
    }
}
```

  For this to work, you must turn on the “Created By” Metadata Index in the Dashboard.

# Cloud Kit

- Retrieving Records from the Database

  Sometimes you end up with a CKRecordID when what you want is a CKRecord.
  You can turn a CKRecordID into a CKRecord with this CKDatabase method …

```
func fetchRecordWithID(CKRecordID, completionHandler: (CKRecord?, NSError?) -> Void)
```

  You can get a CKRecordID from a CKRecord with CKRecord's recordID var.
  You can also find out the Record Type (e.g. Entity) of a CKRecord with its recordType var.
  Ditto its creator, creation date, last modification date, etc.

# Cloud Kit

- Deleting Records from the Database

  To delete a CKRecord from the database, you use its CKRecordID in this CKDatabase method ...

```
func deleteRecordWithID(CKRecordID, completionHandler: (CKRecordID?, NSError?) -> Void)
```

# Cloud Kit

- Storing a reference to another Record

  To have a Field which points to another Record, you cannot just say …

```
let twitterUser = CKRecord(recordType: “TwitterUser”)
let tweet = CKRecord(recordType: “Tweet”)
tweet[“tweeter”] = twitterUser
```

  To store a relationship between Records like this, you must use a `CKReference` …

  `tweet[“tweeter”] = CKReference(record: twitterUser, action: .DeleteSelf or .None)`

  `.DeleteSelf` means if `twitterUser` is deleted from the database, delete this `tweet` too.
  For this cascading deleting to work the user must have write access to the other Record Types.
  You set up write permissions for Record Types in the Dashboard.

  When creating an `NSPredicate`, you do NOT have to create a `CKReference` explicitly.

  `let predicate = NSPredicate(format: “tweeter = %@“, twitterUser)` // is okay

  There is no CoreData `NSSet` equivalent, you just `performQuery` and get an array back.

# Cloud Kit

- Standing Queries (aka Subscriptions)

  Sometimes it'd be nice for iCloud to just let you know when something changes
  (rather than having to be `performQuery()`'ing all the time).

  You can set up a query that, when the results change, causes a Push Notification to be sent.

  Example: Get notified when any Tweet is created or deleted …

```
let predicate = NSPredicate(format: "TRUEPREDICATE")
let subscription = CKSubscription(
    recordType: "Tweet",
    predicate: predicate,
    subscriptionID: "All Tweet Creation and Deletion" // must be unique
    options: [.FiresOnRecordCreation,.FiresOnRecordDeletion]
) // other options are .FiresOnRecordUpdate and .FiresOnce (deletes subscription after firing)
subscription.notificationInfo = … // more on this in a moment
database.saveSubscription(subscription) { (subscription, error) in
    // common error here is ServerRejectedRequest (because already subscribed)
}
```

# Cloud Kit

- Push Notifications

  While this is fantastically cool, we have to learn now how to receive a Push Notification.
  Push Notifications are also known as Remote Notifications.
  Remote Notifications are handled through the `AppDelegate`.
  Remote Notifications cannot be received in the Simulator.

  First we have to let the `UIApplication` know that we're willing to receive Push Notifications ...

```
let application = UIApplication.sharedApplication()
let types = .None // can also be .Alert, .Badge or .Sound notifications
let settings = UIUserNotificationSettings(forTypes: types, categories: nil)
application.registerUserNotificationSettings(settings)
application.registerForRemoteNotifications()
```

  This probably wants to be in `application(didFinishLaunchingWithOptions:)`.
  If you want the push notification from iCloud to put up an alert or badge or sound,
  configure the `notificationInfo` var of the subscription.

# Cloud Kit

- Push Notifications

  Next we must implement the Application Lifecycle method below.
  Inside, we have to convert the `userInfo` passed into a CloudKit notification object.

```
func application(
    application: UIApplication,
    didReceiveRemoteNotification userInfo: [NSObject:AnyObject]
) {
    let dict = userInfo as! [String:NSObject]
    let ckn = CKNotification(fromRemoteNotificationDictionary: dict)
}
```

  This `ckn` can either be a `CKQueryNotification` or something else (we won't talk about).
  If it's a `CKQueryNotification`, it has the changed `recordID` & why (`queryNotificationReason`).
  You can configure the subscription's `notificationInfo` to prefetch fields (`recordFields`).

# Cloud Kit

- Push Notifications

  So we have this great notification, but how do we pass it on to whoever in our app needs it? One very simple way is to use an NSNotification (radio station).

```
func application(
    application: UIApplication,
    didReceiveRemoteNotification userInfo: [NSObject:AnyObject]
) {
    let dict = userInfo as! [String:NSObject]
    let ckn = CKNotification(fromRemoteNotificationDictionary: dict)
    let localNotification = NSNotification(
        name: "MyCloudKitNotificationName", // should be a global constant
        object: self, // the AppDelegate is posting this notification
        userInfo: ["CKNKey":ckn] // should be a global constant
    )
    NSNotificationCenter.defaultCenter().postNotification(localNotification)
}
```

# Cloud Kit

- Push Notifications

  Now anyone, anywhere in our application can get these Push Notifications forwarded to them by listening for this NSLocalNotification …

```
var ckObserver = NSNotificationCenter.defaultCenter().addObserverForName(
    “MyCloudKitNotificationName”, // should use that global constant here
    object: nil, // accept from anyone, or could put AppDelegate here
    queue: nil,  // the queue the poster posts on (will be main queue)
    usingBlock: { notification in  // use global constant for “CKNKey” …
        if let ckqn = notification.userInfo?[“CKNKey”] as? CKQueryNotification {
            if let ckqnSubID = ckqn.subscriptionID where mySubID == ckqnSubID {
                // do something with the CKQueryNotification
            }
        }
    }
}
```

  Later, stop listening with `NSNotificationCenter.defaultCenter().removeObserver(ckObserver)`