iOS Dev Accelerator
Week 5 Day 3

- App States
- iCloud Sync!
- NSFetchedResultsController
- Web Tools Wednesday: MongoDB

App States

App state behaviors

- Apps behave different in the background and foreground.
- The OS purposely limits what your app can do while its in the background, because it always gives resource priority to the foreground app.
- Your app is always notified when it transitions from background to foreground and vice versa.

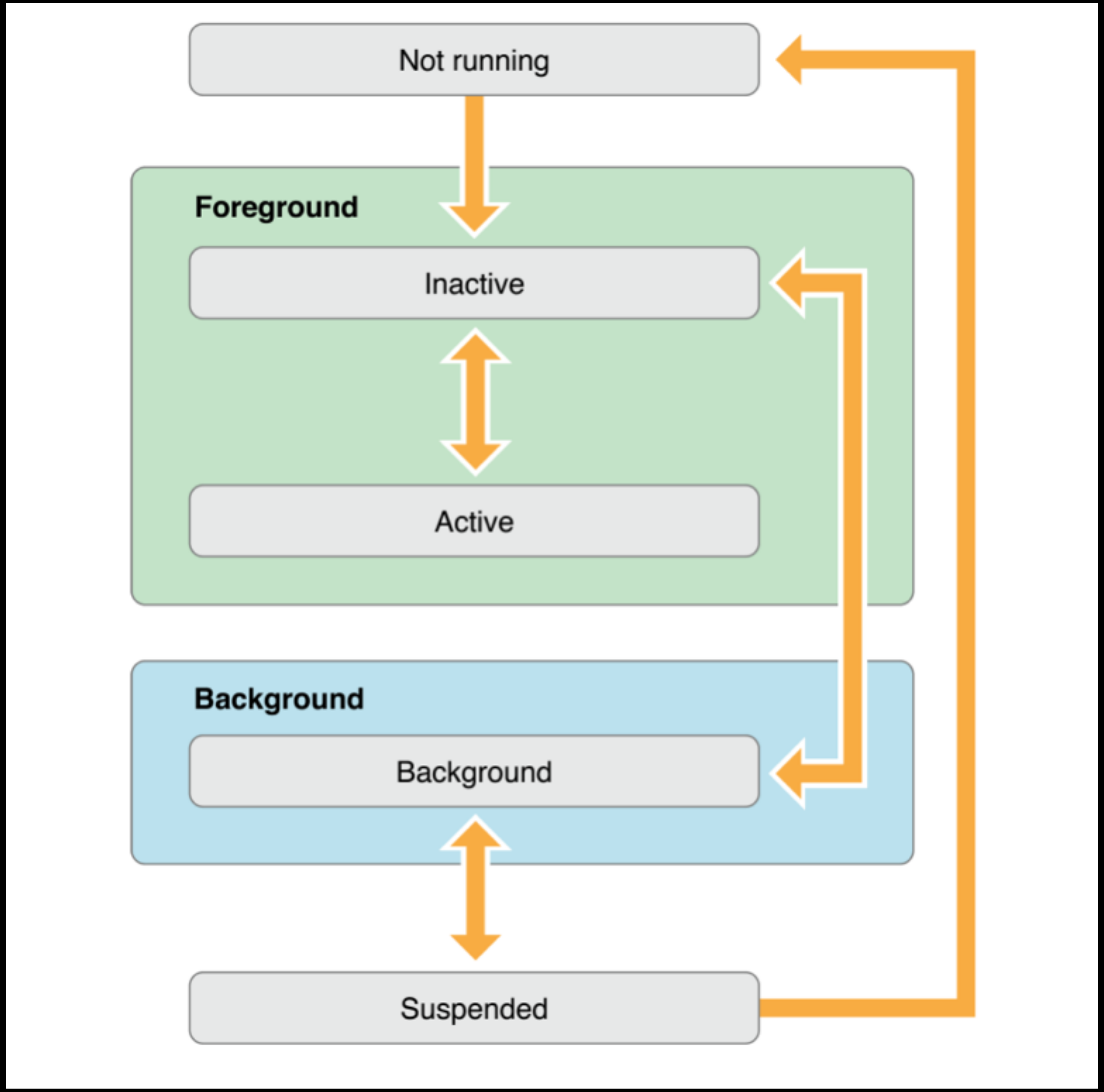
Apples App State Guidelines

1. Required - Respond appropriately to state transitions in your app.
2. Required - When moving to background specifically, make sure your app adjusts its behavior accordingly.
3. Recommended - Register for notifications that report system changes that your app needs. If your app is suspended, the system queues up these changes and then gives them to you when your app resumes.
4. Optional - If your app actually does some kind of work in the background, you need to ask the system for the appropriate permissions to do that work.

App states

- Any at given time, your app is in one of 5 states:
  - **Not Running:** the app has not been launched, or was running and was terminated by the system.
  - **Inactive:** The app is running in the foreground but it is currently not receiving events. This is usually brief and during transitions or phonecalls.
  - **Active:** The app is running in the foreground and receiving events. This is the normal mode for apps.
  - **Background:** The app is in the background and executing code. Most apps briefly enter this mode on their way to being suspended. However, an app that requests extra executing time may remain in this state for an extra period of time. Also, an app being launched directly into the background enters this state instead of inactive.
  - **Suspended:** The app is in the background but not executing code. The system moves apps to this state automatically and does not notify them before doing so. While suspended, an app remains in memory but does not execute code. When a low-memory condition occurs, apps taking up large amounts of memory are terminated.

App states diagram



App state transitions

- Most state transitions are accompanied by a corresponding call to the methods of your app delegate object:
- application:willFinishLaunchingWithOptions: - This method is your apps first chance to execute code at launch time.
- application:DidFinishLaunchingWithOptions: - This method allows you to perform any final initialization before your app is displayed to the user.
- applicationDidBecomeActive: - Lets your app know that it is about to become the foreground app.

App state transitions cont.

- applicationWillResignActive: - Lets you know your app is transitioning away from being the foreground app.
- applicationDidEnterBackground: - Lets you know your app is now running in the background and may be suspended at any time.
- applicationWillEnterForeground: - Lets you know your app is moving out of the background and back into the foreground, but it is not yet active.
- applicationWillTerminate: - Lets you know your app is being terminated. This method is not called if your app is already suspended.

Things to do at Launch time

- Check contents of the launch options dictionary about why the app was launched, and respond appropriately. If its a regular launch from the user you usually don't care, but if the app was launched by the system for some background task, you definitely care.
- Initialize the app’s critical data structures.
- Prepare your app for display if its going to be displayed.
- Remember to keep the willFinishLaunching and didFinishLaunching methods minimal to not slow down launch speeds.
- The system kills any app that takes longer than 5 seconds to launch.
- When launching straight into the background, there shouldn't be much to do except respond to the event that caused the launch.

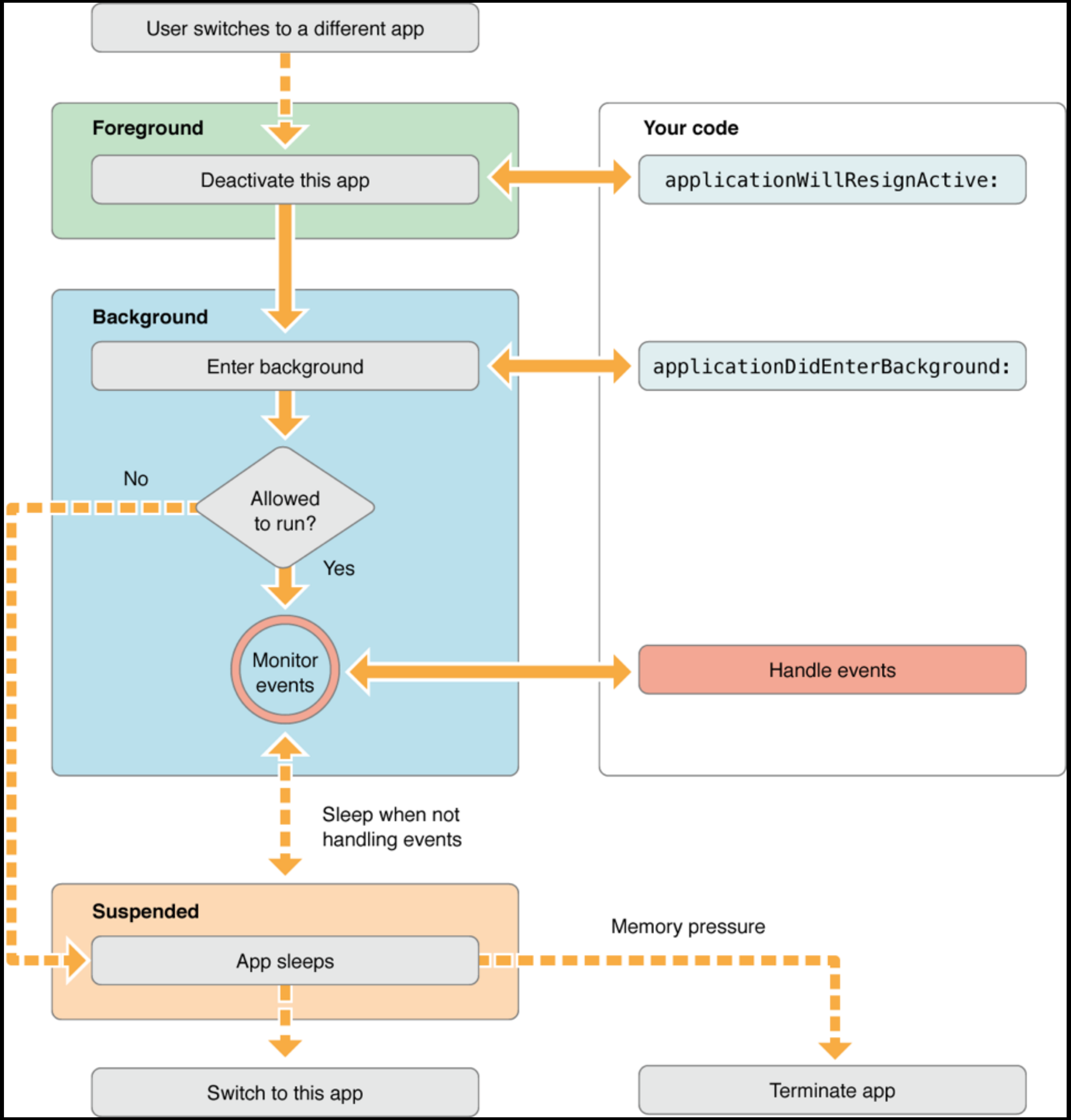
Responding to interruptions

- When an alert based interruption occurs (phone call,text message, system message) your app temporarily moves into the inactive state so the system can send a prompt to a user.
- The app remains in this state until the user dismisses the prompt.
- After the prompt dismisses, your app is either returned to active or moved to background state.
- In response to this workflow, your app should do the following things in applicationWillResignActive:
    - Stop timers or periodic tasks
    - Stop running queries
    - Don't initiate any new tasks.
    - Pause movie playback (except when over airplay)
    - Pause the game
    - suspend any non critical operation or dispatch queues.

Moving to the Background

- When user presses home button, sleep/wake button, or launches another app, the foreground app is transitioned to the inactive state and then background state.
- first applicationWillResignActive is called, and then applicationDidEnterBackground.
- Most apps move to suspended state after applicationDidEnterBackground returns or shortly there after.
- Apps that request specific background tasks may continue to run for a while longer.
- New in iOS8 : Location apps are automatically relaunched after the user kills it!

Moving to the Background



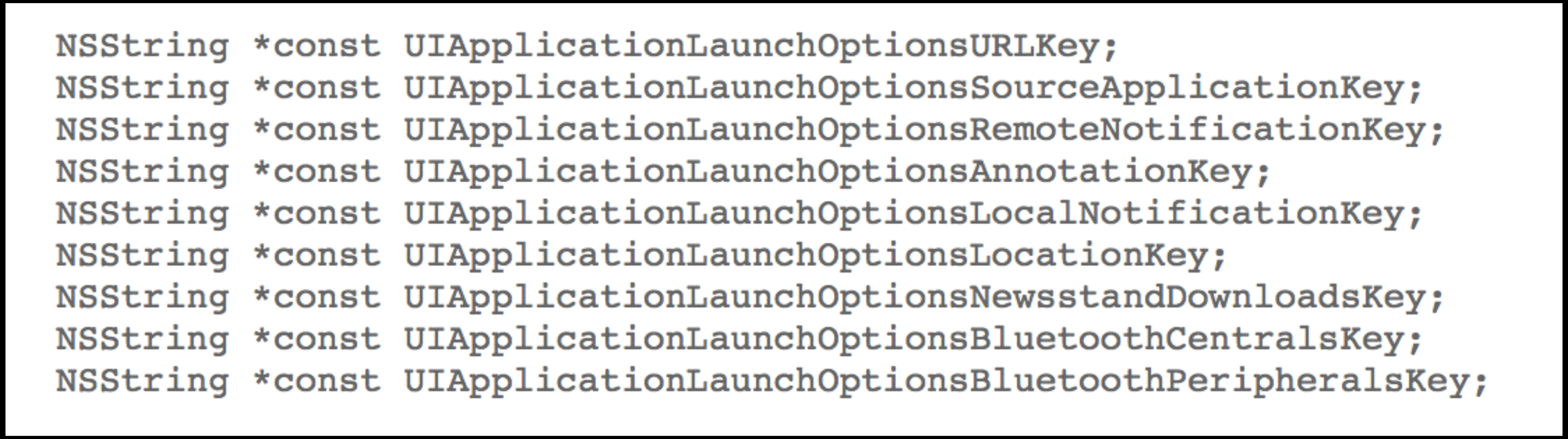
What to do when moving to background

- Say cheese: When the applicationDidEnterBackground method returns, the system takes a picture of your app’s user interface and uses the image for transitions. So clear out any sensitive data.
- Save User Data: all your unsaved data should be saved to disk, since your app could be quietly killed while in the suspended state.
- Free up memory: apps are killed based on how much memory they are taking up, so release as much memory as possible! (big images especially)

DidFinishLaunching Dictionary

- This dictionary will usually be empty if the user launches the app on their own.

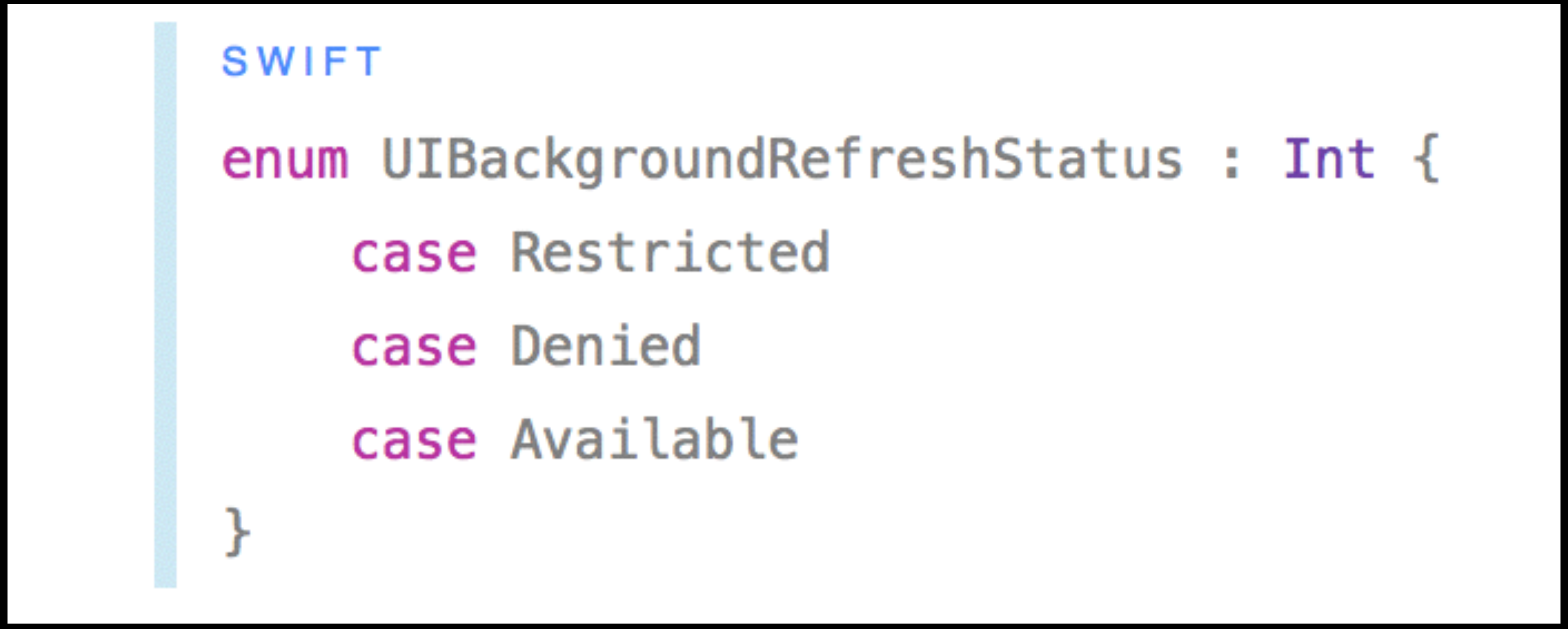
Possible Keys:



```
NSString *const UIApplicationLaunchOptionsURLKey;
NSString *const UIApplicationLaunchOptionsSourceApplicationKey;
NSString *const UIApplicationLaunchOptionsRemoteNotificationKey;
NSString *const UIApplicationLaunchOptionsAnnotationKey;
NSString *const UIApplicationLaunchOptionsLocalNotificationKey;
NSString *const UIApplicationLaunchOptionsLocationKey;
NSString *const UIApplicationLaunchOptionsNewsstandDownloadsKey;
NSString *const UIApplicationLaunchOptionsBluetoothCentralsKey;
NSString *const UIApplicationLaunchOptionsBluetoothPeripheralsKey;
```

`UIApplication.sharedApplication().backgroundRefreshStatus`

- Every app either has the ability to be launched into the background so it can perform background tasks or it doesn't.
- Check this this property to see if this is available, and warn the user if it isn't but your app replies on this.
- Users can disable background refresh on specific apps in Settings>General>Background Refresh



SWIFT

```swift
enum UIBackgroundRefreshStatus : Int {
    case Restricted
    case Denied
    case Available
}
```

App state behaviors

- Apps behave different in the background and foreground.
- The OS purposely limits what your app can do while its in the background, because it always gives resource priority to the foreground app.
- Your app is always notified when it transitions from background to foreground and vice versa.

iCloud

iCloud

- "iCloud is a cloud service that gives your users a consistent and seamless experience across all of their iCloud-enabled devices"
- iCloud works with things called '**ubiquity** containers', which are special folders that your app uses to store data in the cloud.
- Whenever you make a change in your ubiquity container, the system uploads the changes to the cloud.
- iCloud is closely integrated with CoreData to help persist your manage objects to the cloud. This is another incentive to use core data to manage your model layer!

Enabling iCloud in your app

- To use iCloud in your app, you have enable iCloud support in your app and in your project.
- Navigate to your app’s target in the project settings in Xcode, and select the capabilities pane. Switch the iCloud option on and then follow the configuration prompts.

Enabling iCloud in your Core Data Stack

- iOS8 makes enabling iCloud in your core stack is as simple as setting up your persistent store with a dictionary containing a few special options:
    - NSPersistentStoreUbiquitousContentNameKey - option to specify that a persistent store has a given name in ubiquity
    - NSMigratePersistentStoresAutomaticallyOption - key to automatically attempt to migrate versioned stores
    - NSInterMappingModelAutomaticallyOption - key to attempt to create the mapping model automatically

Listening for changes in the cloud

- You can observe a specific notification that comes from your cloud enabled store coordinator to know when changes have happened in the cloud.
- The name of the notification is NSPersistentStoreDidImportUbiquitousContentChangesNotification
- Inside the selector you designate to be fired when the notification is observed, you do 2 things:
  - call performBlock{} on your context
  - Inside that block you call mergeChangesFromContextDidSaveNotification on your context and simply pass in the notification object you received.

Demo

NSFetchedResultsController

Fetched Results Controller

- "You use a fetched results controller to efficiently manage the results returned from a Core Data fetch request to provide data for a UITableView object"
- A fetched Results controller has 3 modes:
  - No tracking: delegate is set to nil. the controller just shows the data from the original fetch.
  - Memory-only tacking: delegate is non-nil and the file cache name is set to nil. Controller monitors objects in its result set and updates in response to changes.
  - Full persistent tracking: delegate and file cache name are non-nil. Controller tracks and displays changes and maintains a persistent cache of the results.

NSFetchedResultsController

- You typically have an NSFetchedResultsController property on the view controller that is also managed the table view.
- You initialized the fetched results controller with 4 parameters:
  - a fetch request. must contain at least one sort descriptor for order.
  - a managed object context.
  - optional: a key path that returns the section names. the controller spits the results based on this designated attribute.
  - optional: the name of the cache file the controller should use.

NSFetchedResultsControllerDelegate

- the fetched results controller notifies its delegate that the controller's fetched results have been changed due to an add, remove, move, or update operations.
    - controllerWillChangeContent: - tell your tableview to beginUpdates.
    - controllerDidChangeSection:atIndex:forChangeType: - insert new sections or delete old sections in your tableview
    - controllerDidChangeObject:atIndexPath:ForChangeType: - insert/change/delete/move rows in your tableview
    - controllerDidChangeContent: - tell your tableview to endUpdates.

Demo

Web tools wednesday: MongoDB and JSON

MongoDB

- MongoDB is a database that stores JSON objects.
- Unlike SQL, mongo doesn't support relationships.
- A relational database stores data in tables and rows, while MongoDB stores things in collections of JSON objects.
- MongoDB doesn't store the data in row JSON form. Instead it stores it in something called BSON — Binary JSON.

MongoDB

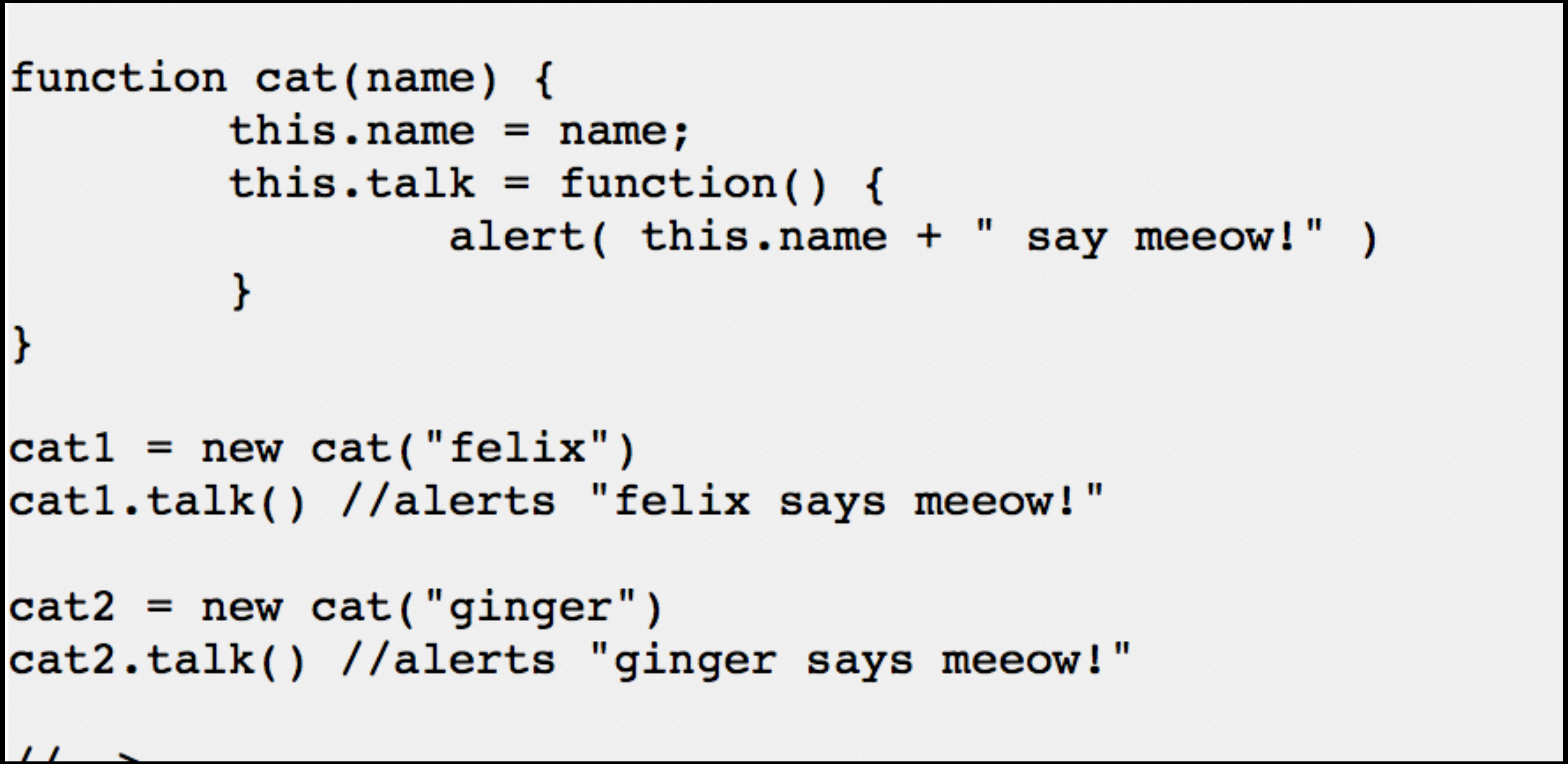
- MongoDB is a native application, and is interfaced with through something called a driver.
- There are mongoDB drivers for almost any environment, including NodeJS!
- So you will be running mongoDB on a separate port on your computer, side by side with your node app.
- To install the mongodb module into your node app, add it your dependencies in your package.json file

Setting up your Driver

- We will setup a constructor in a separate js file for our ‘collection driver’.
- We will then add functionality to the constructor by adding things to its prototype. So any objects we make using the constructor will get all the functionality we added to its prototype.
- We will then expose that constructor, which allows other node js files to access that constructor, like our server.js file.

Javascript Constructor

- A object constructor is just a javascript function
- Its different from regular functions because it is called via the new operator.
- Cat Example from javascriptkit.com:



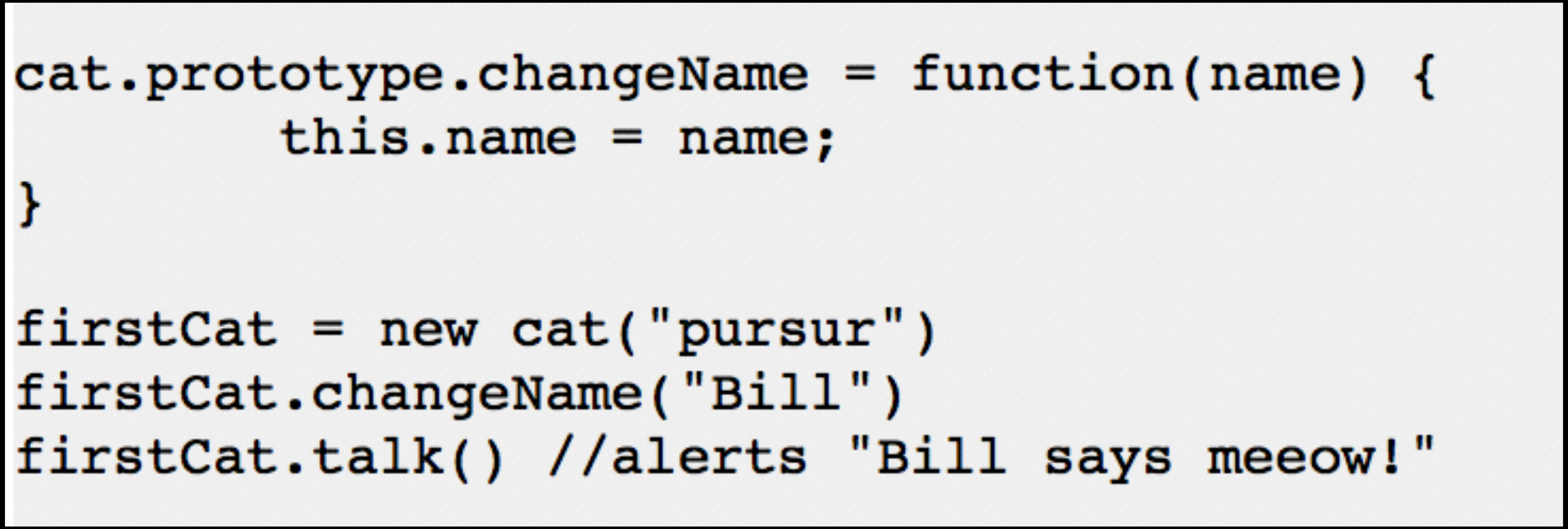
```
function cat(name) {
        this.name = name;
        this.talk = function() {
                alert( this.name + " say meeow!" )
        }
}

cat1 = new cat("felix")
cat1.talk() //alerts "felix says meeow!"

cat2 = new cat("ginger")
cat2.talk() //alerts "ginger says meeow!"
```

Constructor and prototype

- Prototype is a type of inheritance in Javascript.
- We can use it whenever we want an object to inherit a method after it has been defined
- You can think of it like attaching a method to an object after it has been defined. All instances that used the constructor will then gain that functionality.



```
cat.prototype.changeName = function(name) {
        this.name = name;
}

firstCat = new cat("pursur")
firstCat.changeName("Bill")
firstCat.talk() //alerts "Bill says meeow!"
```

Demo