CS 329E
Elements of Mobile Computing

Spring 2018
University of Texas at Austin

Lecture 4

Agenda

- Event-Driven programming
- MVC (Model, View, Controller) pattern
- View Controllers
- Views
- App Startup
- First iOS Application
- Homework 2

Event-Driven Programming

MVC

Prior to event-driven programs, programs were primarily procedural.

That is, what a user could do and in what order was totally controlled by the program.

An event-driven program meant the user had a lot of control on what was going to happen next.

MVC

Main programming paradigms:

- Imperative programming – defines computation as statements that change a program state
- Procedural programming, structured programming – specifies the steps the program must take to reach the desired state
- Declarative programming – defines computation logic without defining its control flow
- Functional programming – treats computation as the evaluation of mathematical functions and avoids state and mutable data
- Object-oriented programming (OOP) – organizes programs as objects: data structures consisting of data fields and methods together with their interactions
- ***Event-driven programming*** – the flow of the program is determined by events, such as sensor outputs or user actions (mouse clicks, key presses) or messages from other programs or threads
- Automata-based programming – a program, or part, is treated as a model of a finite state machine or any other formal automata

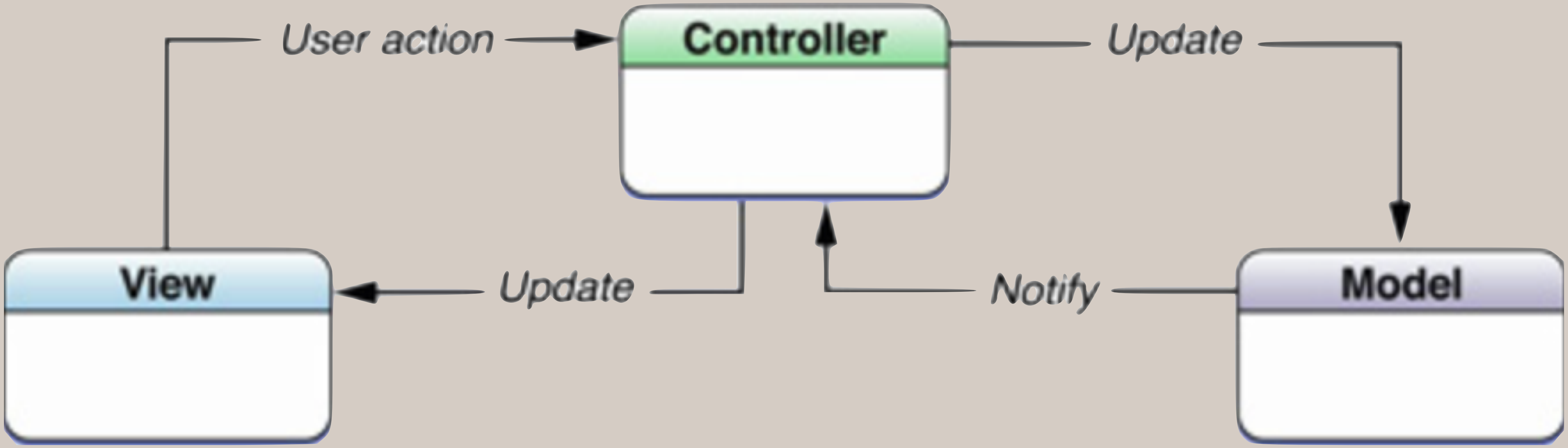
MVC

GUI applications are *event-driven:*

- Event-driven programming is a programming paradigm in which the *flow of the program is determined by events* such as user actions (mouse clicks, key presses), sensor outputs, or messages from other programs/threads
    - The notion is that the application sits, waiting for input from the user - which can come from many directions
- Event-driven programming is *the dominant paradigm* used in graphical user interfaces and other applications that are centered on performing certain actions in response to user input

MVC (Model, View, Controller) pattern

MVC



MVC

The MVC Pattern:

- Assigns objects in an application to one of three roles: model, view, or controller
- Defines the way objects communicate with each other
- Each of the three types of objects is separated from the others by abstract boundaries and communicates with objects of the other types across those boundaries
- Is central to a good design for an iOS app

MVC

The benefits of adopting this pattern:

- Many objects in these applications tend to be more reusable, and their interfaces tend to be better defined
- Tend to be more easily extensible than other applications
- Many Cocoa technologies and architectures are based on MVC and require that your custom objects play one of the MVC roles
- A common pattern for interactive applications with a Graphical User Interface (GUI)

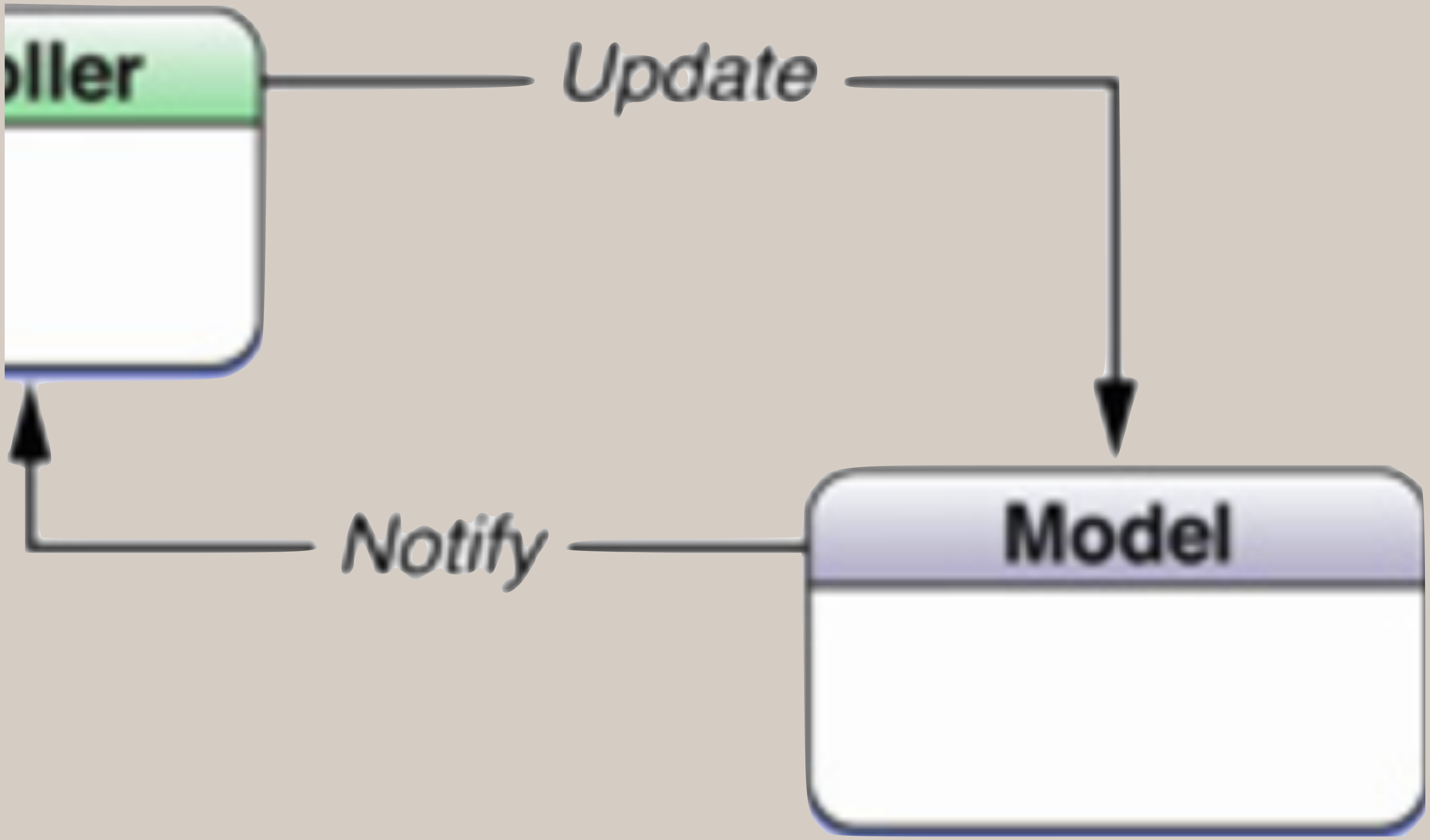
MVC

This kind of breakdown of responsibilities in an iOS app is standard for *event-driven* types of programs.

Event-driven programs came to be when GUIs emerged as the dominant interaction mechanism for computers.

MVC

Model Layer



MVC

Model Layer:

- Model objects encapsulate the data specific to an application and define the logic and computation that manipulate and process that data
- Much of the data that is part of the persistent state of the application should reside in the model objects after the data is loaded into the application
- Ideally, a model object should have no explicit connection to the view objects that present its data and allow users to edit that data—it should not be concerned with user-interface and presentation issues

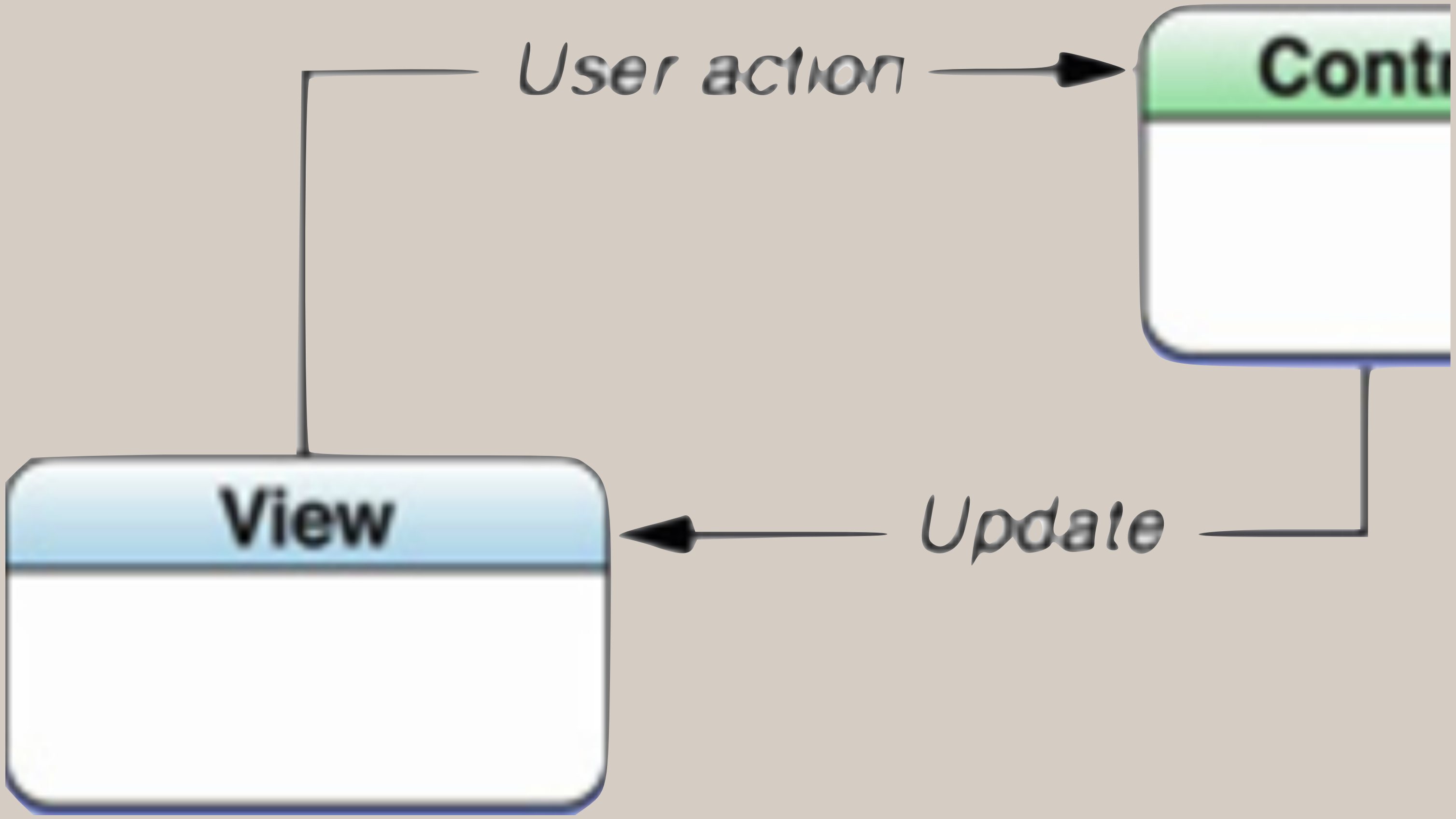
MVC

Model Layer - Communication:

- User actions in the view layer that create or modify data are communicated through a controller object and result in the creation or updating of a model object
- When a model object changes (for example, new data is received over a network connection), it notifies a controller object, which updates the appropriate view objects

MVC

View Layer



MVC

View Layer:

- A view object is an object in an application that users can see
- A view object knows how to draw itself and can respond to user actions
- A major purpose of view objects is to display data from the application's model objects and to enable the editing of that data
- Despite this, view objects are typically decoupled from model objects in an MVC application
- Because you typically reuse and reconfigure them, view objects provide consistency between applications
- Both the UIKit and AppKit frameworks provide collections of view classes, and Interface Builder offers dozens of view objects in its Library

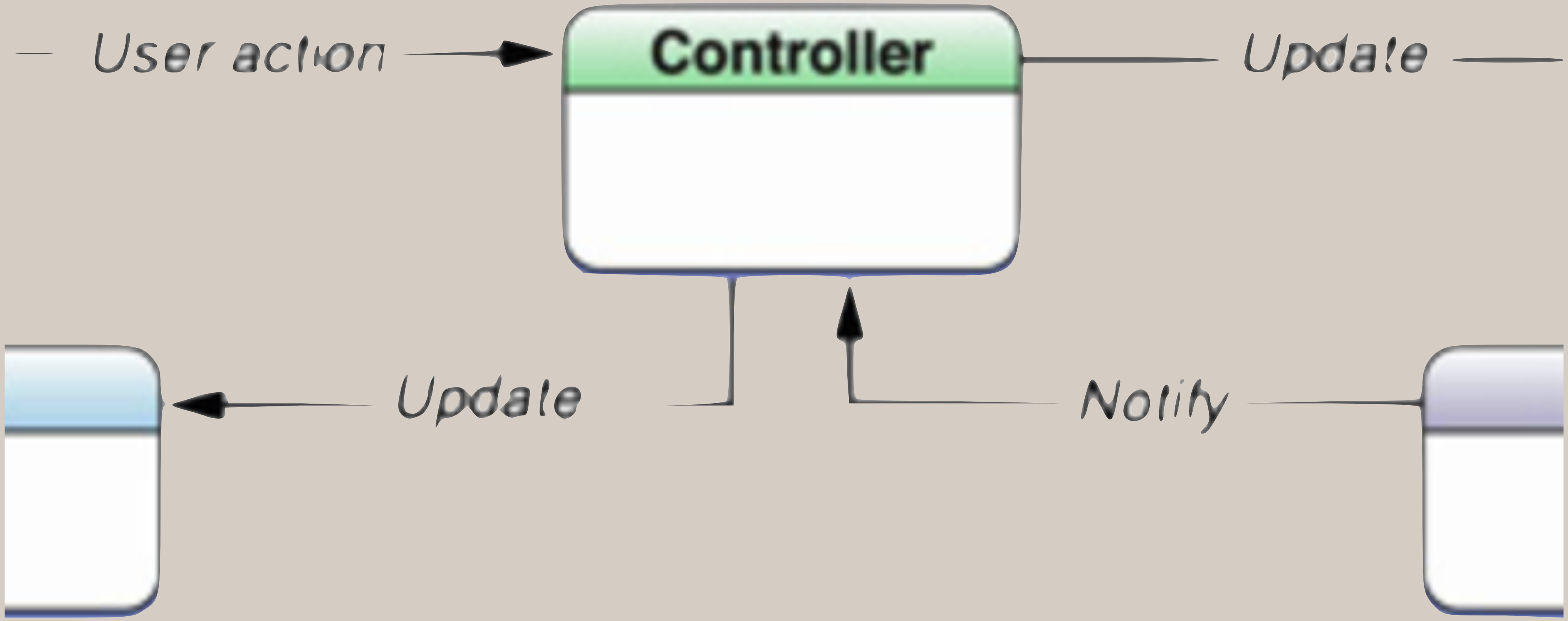
MVC

View Layer - Communication:

- View objects learn about changes in model data through the application’s controller objects and communicate user-initiated changes—for example, text entered in a text field—through controller objects to an application’s model objects

MVC

Controller Layer



MVC

Controller Layer:

- A controller object acts as an intermediary between one or more of an application’s view objects and one or more of its model objects
- Controller objects are thus a conduit through which view objects learn about changes in model objects and vice versa
- Controller objects can also perform setup and coordinating tasks for an application and manage the life cycles of other objects

MVC

Controller Layer - Communication:

- A controller object interprets user actions made in view objects and communicates new or changed data to the model layer
- When model objects change, a controller object communicates that new model data to the view objects so that they can display it

MVC

Where do the MVC components come from:

- iOS frameworks and classes provide 2 out of the 3 components of MVC - View Controllers and Views
    - Although, you generally customize View Controllers and Views
- You custom define the third component - Model

View Controllers

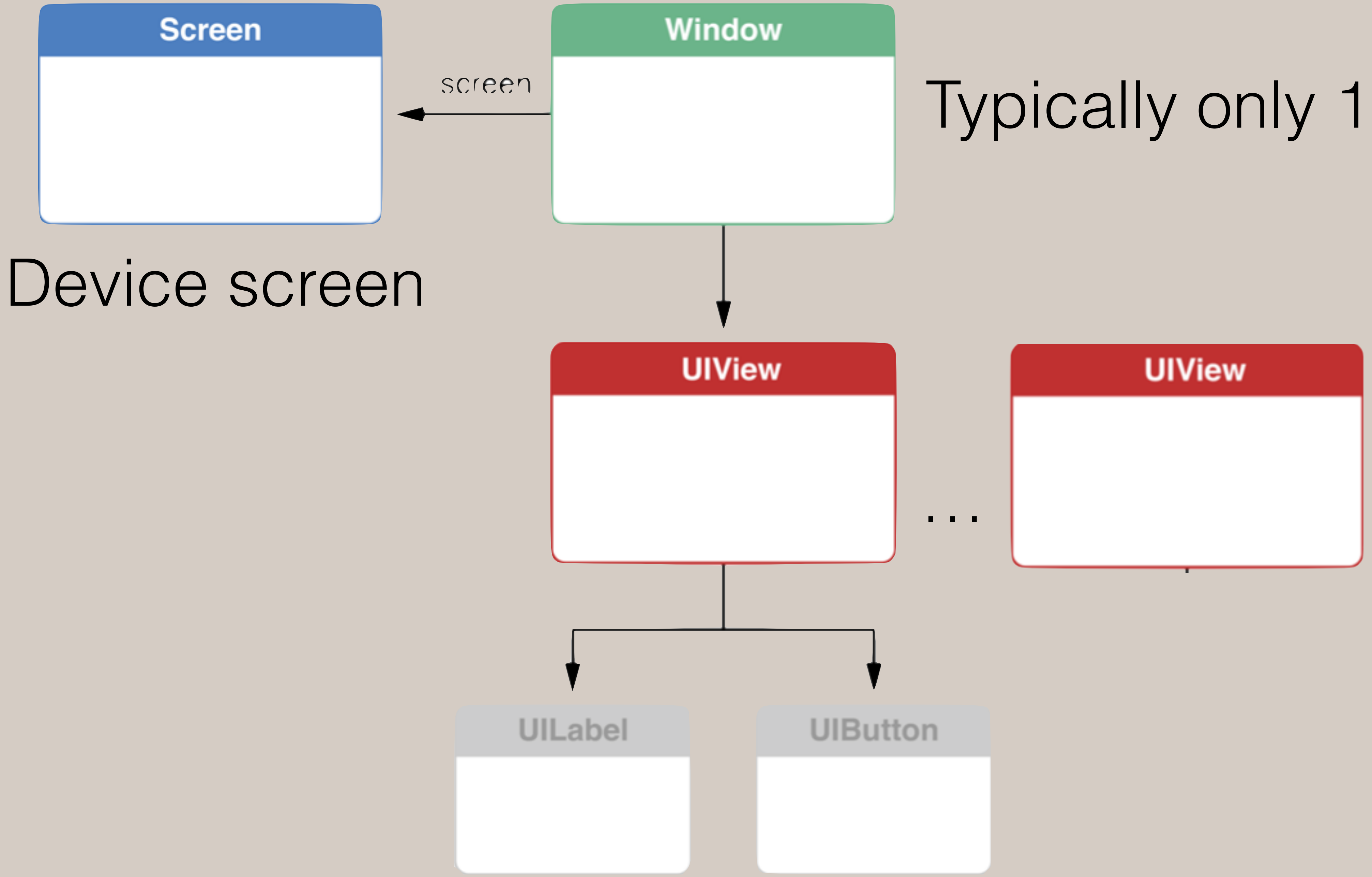
View Controllers

What are View Controllers?

- They are the objects in your iOS application that contain the coordinating code between the data and view components
- All view controllers derive from the *UIViewController* class
- All iOS applications have *at least one* view controller
    - and at least one, and typically only one, window

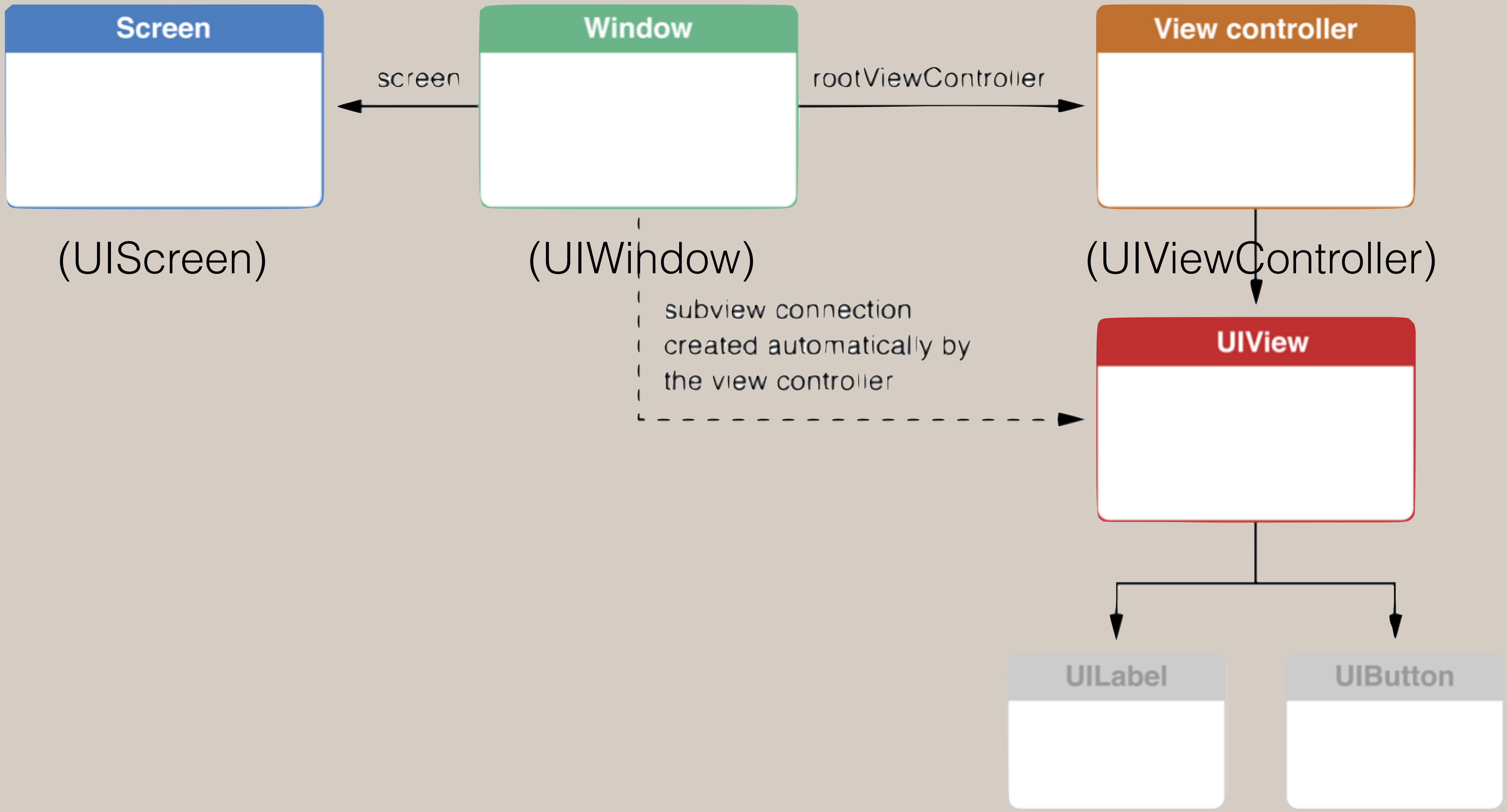
View Controllers

A window with its target screen and content views



View Controllers

A view controller attached to a window automatically adds it’s views as a subview of the window



View Controllers

How does having only one window work for complicated applications?

- For iPhone applications the screen real estate is limited enough that the user interface is broken into small chunks (views) that are managed by view controllers
  - Only one chunk is displayed at any given time
    - Although, with the larger phones (6Plus) this is changing
- iPad applications can more easily make use of multiple windows

View Controllers

View controllers can be divided into two general categories:

- *Content* view controllers - example: UIViewController
  - A content view controller presents content on the screen using a view or a group of views organized into a *view hierarchy*
- *Container* view controllers - example: UINavigationController
  - A container view controller contains content owned by other view controllers
  - These other view controllers are explicitly assigned to the container view controller as its children
  - A container controller can be both a parent to other controllers and a child of another container
  - Ultimately, this combination of controllers establishes a view *controller hierarchy*

View Controllers

Survey of some view controllers:

- Basic view controller - UIViewController
- Table view controller - UITableViewController
- Navigation view controller - UINavigationController
- Page view controller - UIPageViewController
- Tab view controller - UITabBarController

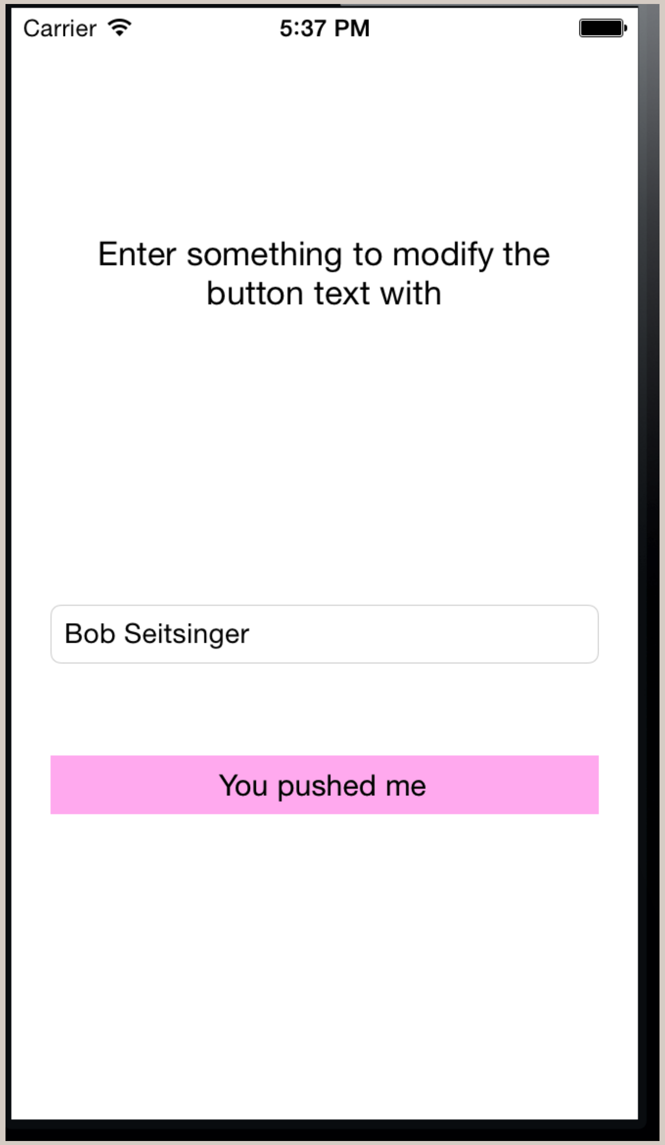
View Controllers

Basic view controller:

- Used to display any combination of views (widgets) - labels, buttons, text fields, etc.

View Controllers

Basic view controller (continued):



Enter something to modify the button text with

Bob Seitsinger

View Controllers

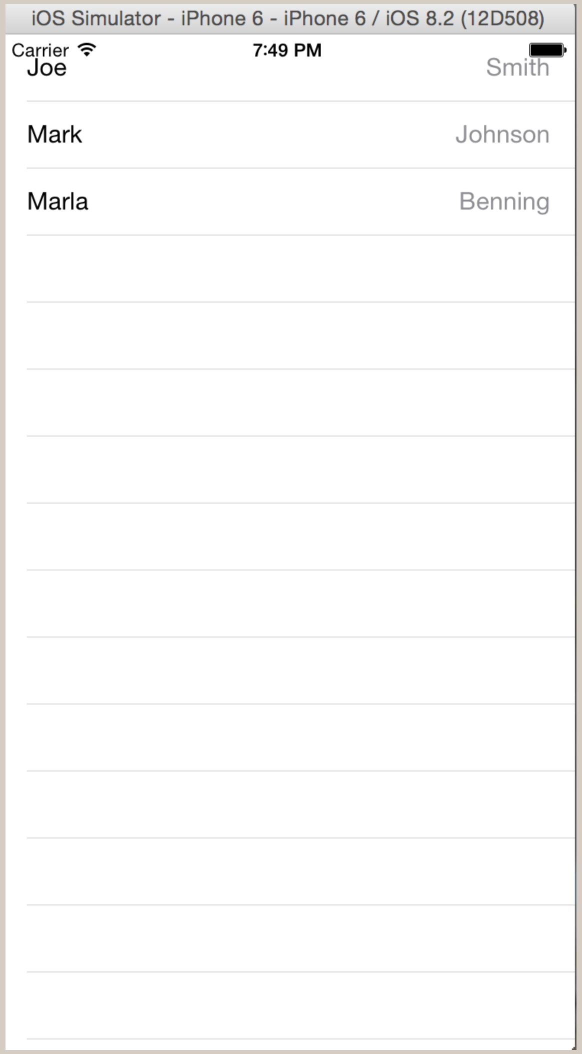
Table view controller:

- Used to display a list of things, in tabular form
- Each item in the list is termed a *table cell*
- By default, all cells have the same layout
  - However, you can programmatically define a custom layout for any given cell
- You can divide it into N *sections*
- Each section can have a title
- Auto scrolls

View Controllers

Table view controller (continued):

- Simple table view cell - using one of the pre-defined table cell types

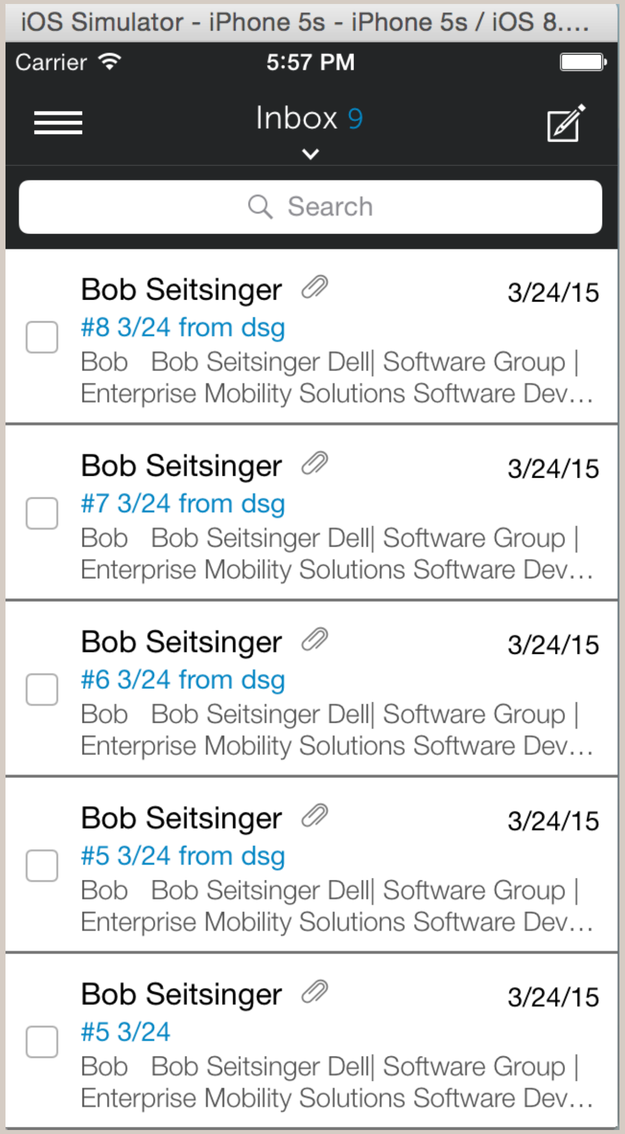


Joe	Smith
Mark	Johnson
Marla	Benning

View Controllers

Table view controller (continued):

- Custom table view cell



□ Bob Seitsinger 3/24/15
#8 3/24 from dsg
Bob Bob Seitsinger Dell| Software Group | Enterprise Mobility Solutions Software Dev…

□ Bob Seitsinger 3/24/15
#7 3/24 from dsg
Bob Bob Seitsinger Dell| Software Group | Enterprise Mobility Solutions Software Dev…

□ Bob Seitsinger 3/24/15
#6 3/24 from dsg
Bob Bob Seitsinger Dell| Software Group | Enterprise Mobility Solutions Software Dev…

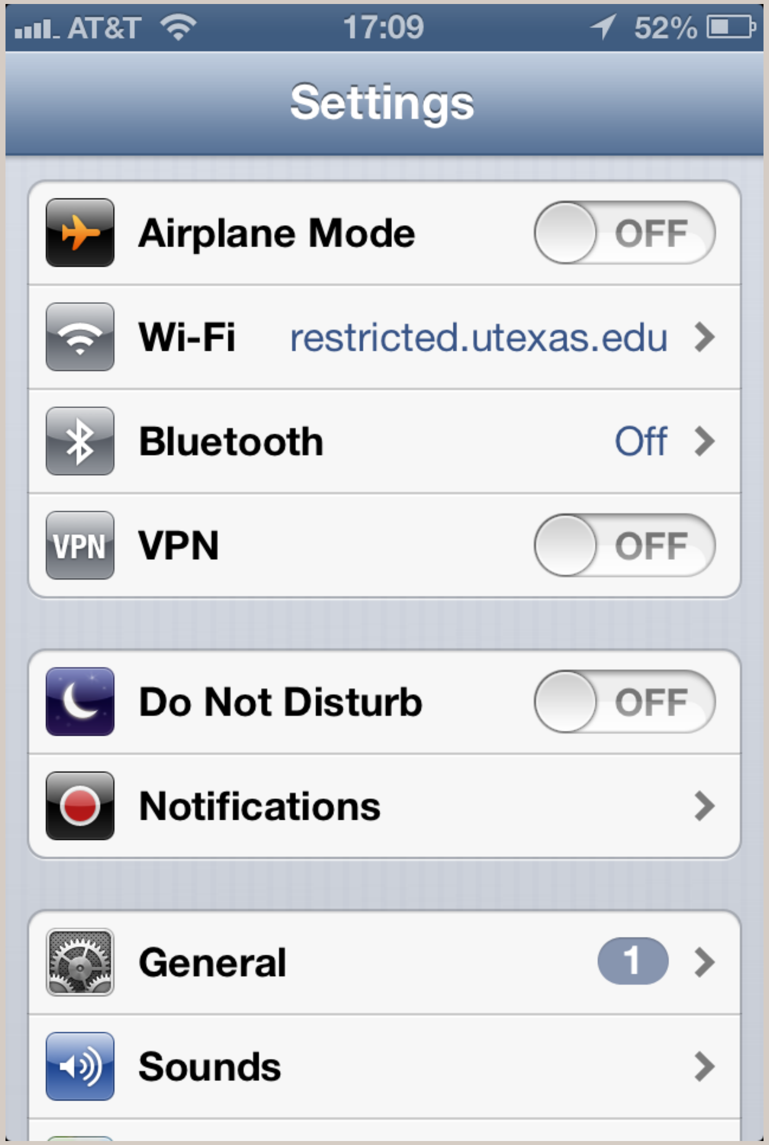
□ Bob Seitsinger 3/24/15
#5 3/24 from dsg
Bob Bob Seitsinger Dell| Software Group | Enterprise Mobility Solutions Software Dev…

□ Bob Seitsinger 3/24/15
#5 3/24
Bob Bob Seitsinger Dell| Software Group | Enterprise Mobility Solutions Software Dev…

View Controllers

Table view controller (continued):

- With multiple sections



AT&T 17:09 52%

Settings

Airplane Mode OFF

Wi-Fi restricted.utexas.edu

Bluetooth Off

VPN OFF

Do Not Disturb OFF

Notifications

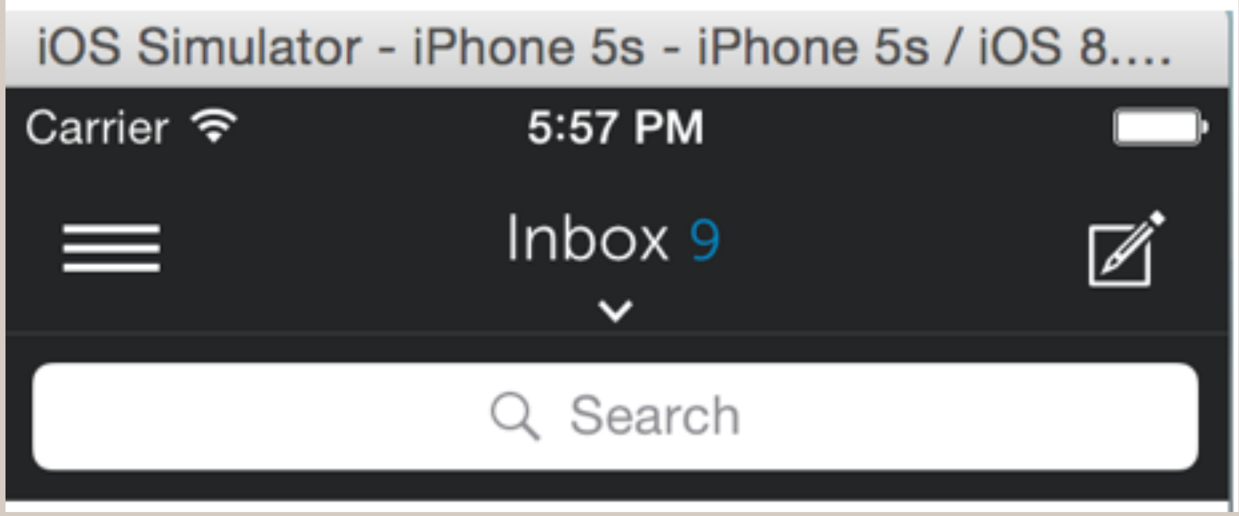
General 1

Sounds

View Controllers

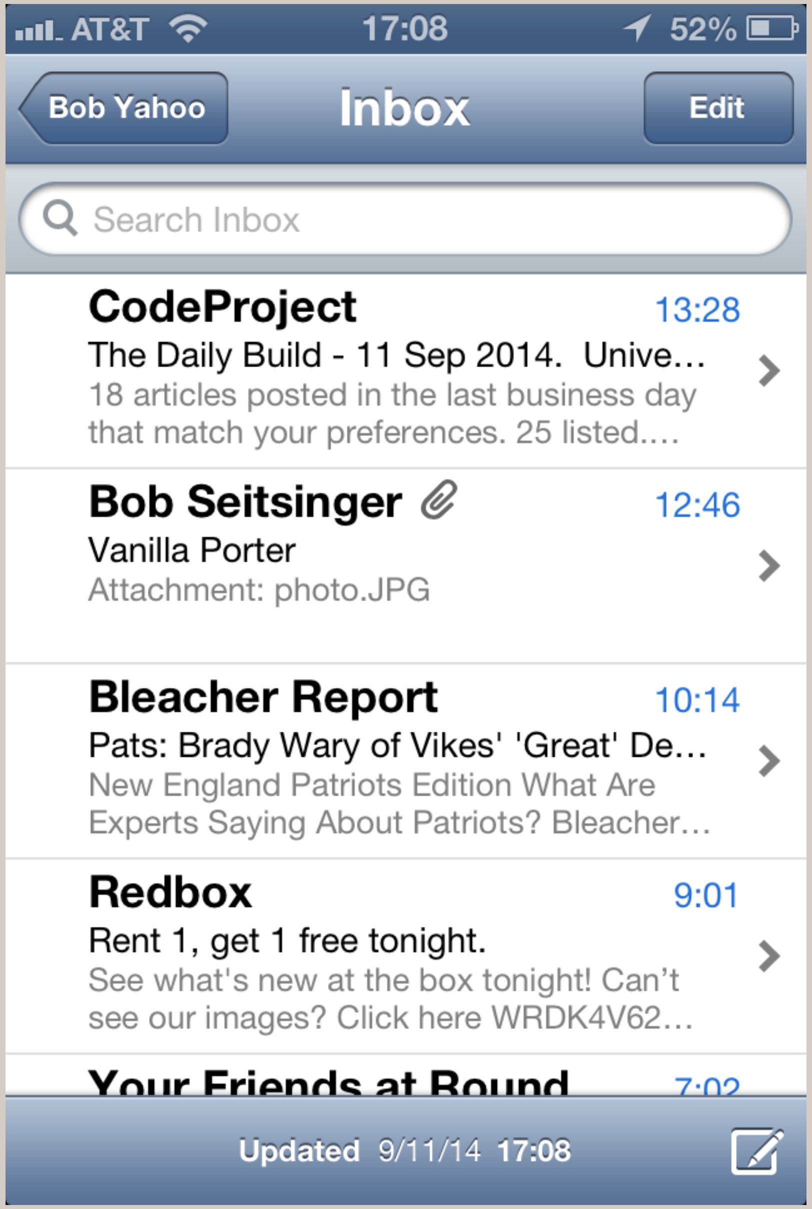
Navigation controller:

- Used to contain and coordinate navigating between view controllers
- Includes a portion of the top of the screen where it provides an area for an optional two buttons (far left, far right) and some text in the middle - for actions



View Controllers

Navigation controller (continued):



CodeProject 13:28
The Daily Build - 11 Sep 2014. Unive…
18 articles posted in the last business day that match your preferences. 25 listed.…

Bob Seitsinger 12:46
Vanilla Porter
Attachment: photo.JPG

Bleacher Report 10:14
Pats: Brady Wary of Vikes' 'Great' De…
New England Patriots Edition What Are Experts Saying About Patriots? Bleacher…

Redbox 9:01
Rent 1, get 1 free tonight.
See what's new at the box tonight! Can't see our images? Click here WRDK4V62…

Your Friends at Round 7:02

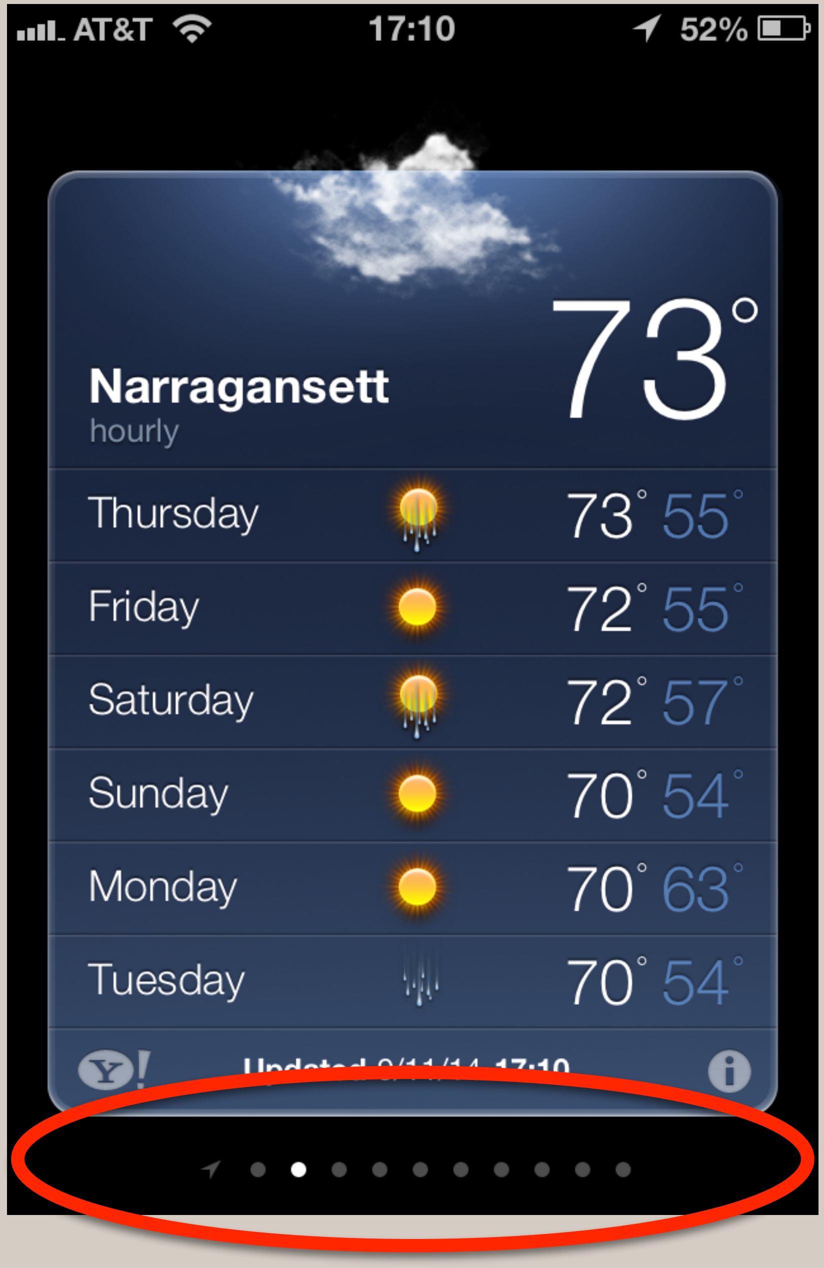
View Controllers

Page view controller:

- Provides an interface to simulate the notion of flipping through pages
- There is a horizontal line of dots that represent *pages* the user can navigate to
- Navigation is in the form of left or right swipes

View Controllers

Page view controller (continued):



Narragansett
hourly

73°

Thursday	73° 55°
Friday	72° 55°
Saturday	72° 57°
Sunday	70° 54°
Monday	70° 63°
Tuesday	70° 54°

Updated 9/11/14 17:10

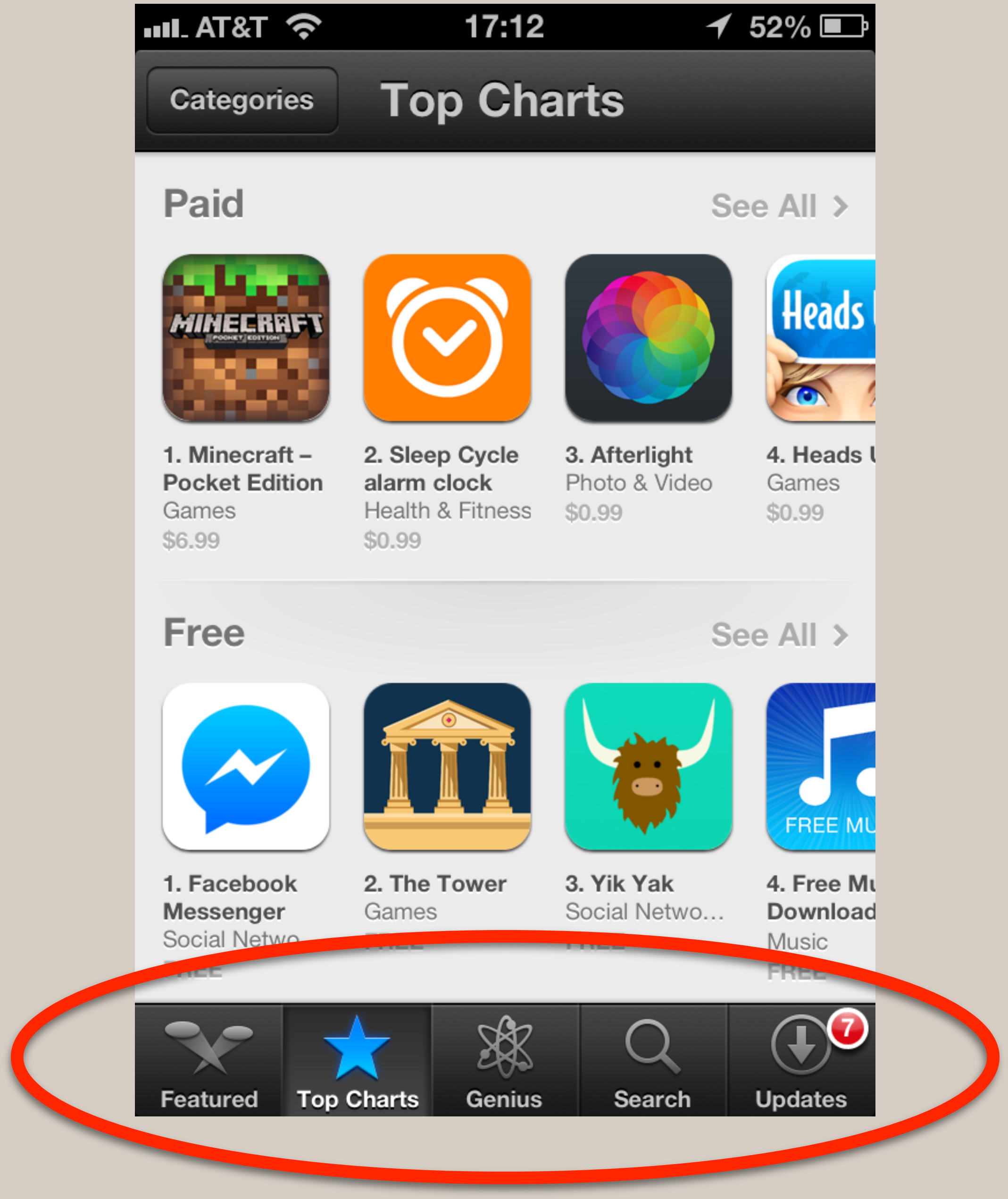
View Controllers

Tab Bar controller:

- Provides a row of tabs at the bottom of the screen, to be able to jump between features/functions
- Each button navigates to a different view controller

View Controllers

Tab Bar controller (continued):



Views

Views

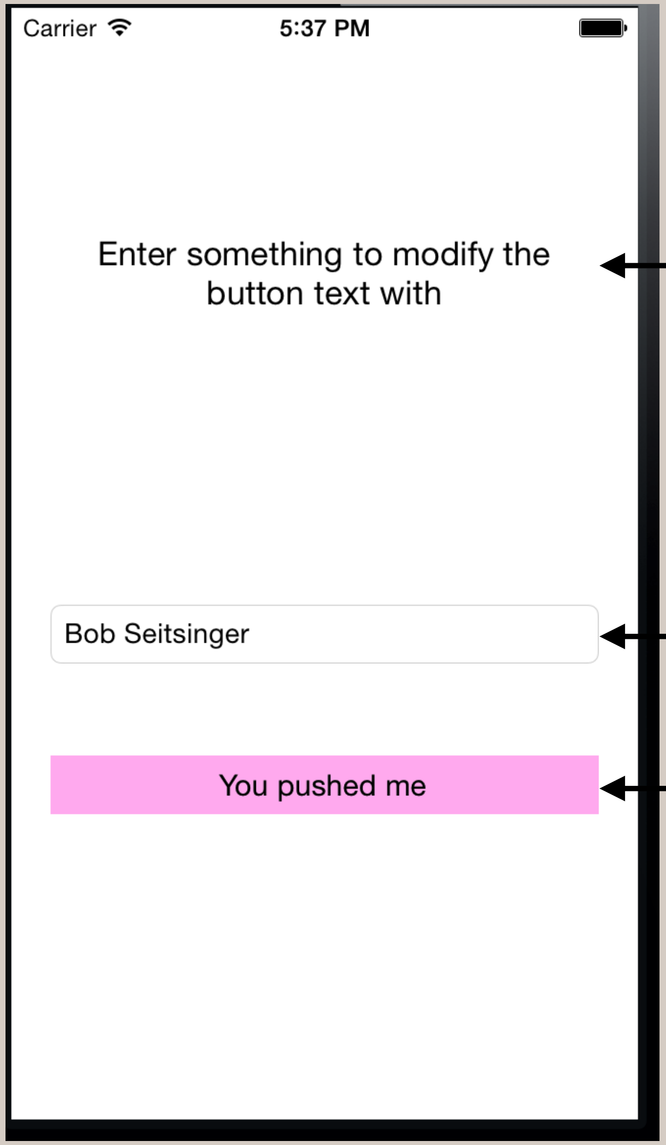
What are views?

- Views are basically all the individual elements in your user interface
- Examples would be - buttons, labels, text fields (input), etc.
- Views can contain other views
- Every view controller has a *base view* where all other views (buttons, labels, etc) are added as children views, thus establishing a *view hierarchy*
- Every view has numerous properties - such as isHidden
  - If a given view is hidden, all it’s subviews are hidden

Views

This simple application has 3 views:

- Label - to display text
- Text Field - for input
- Button - to initiate an action



Enter something to modify the button text with

Bob Seitsinger

Base view

Label

Text Field

Button

View hierarchy:
Base view
    Label
    Text field
    Button

App Startup

App Startup

Basic app startup sequence:

- AppDelegate: didFinishLaunchingWithOptions
- ViewController: viewDidLoad
- ViewController: viewWillAppear
- AppDelegate: applicationDidBecomeActive

You have to decide what method to include your app-specific startup code in.

You don’t want to stall app startup too much.

In-Class Exercise

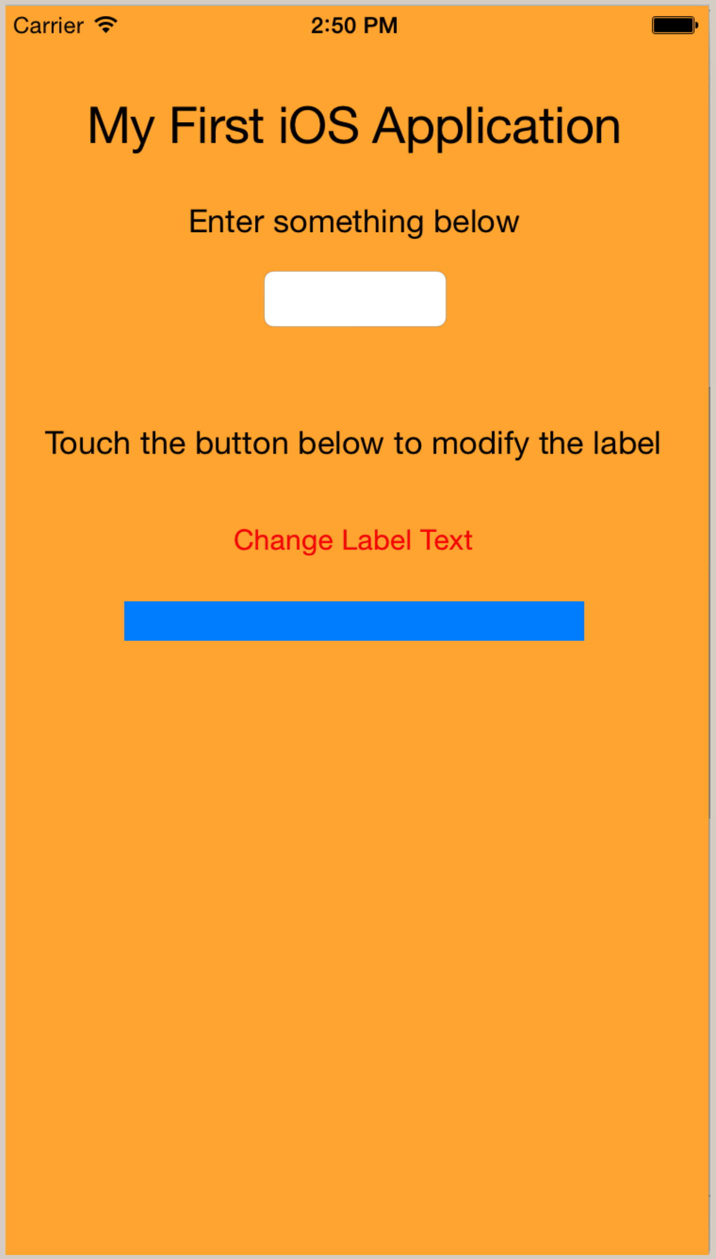
In-Class Exercise

First iOS Application:

- Define a Single View UI and hook up UI elements to code
- UI to include:
  - Label - to display some information
    - How to set label text
  - Button - to invoke an action
    - How to set button text
    - How to define a button handler
  - Text field - for input
    - How to get input
    - How to dismiss the keyboard

First iOS Application

The UI we will create:



My First iOS Application

Enter something below

Touch the button below to modify the label

First iOS Application

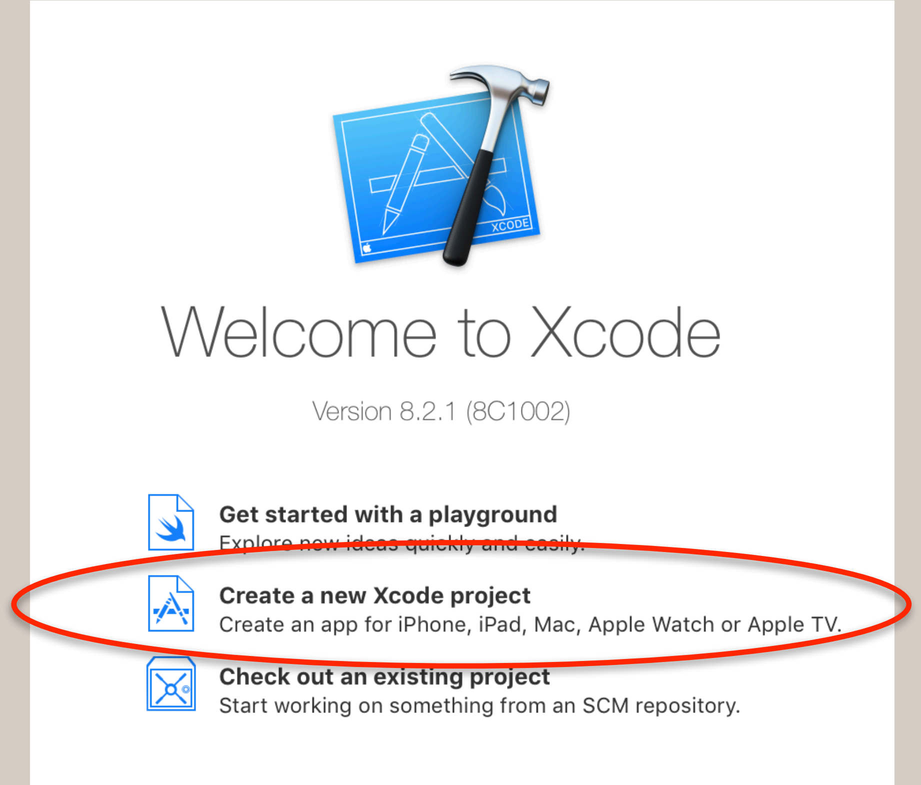
How to dismiss the keyboard:

- Add *UITextFieldDelegate* to your class definition
- Set the text field *delegate*, usually in *viewDidLoad*
- Implement the *textFieldShouldReturn* method
  - Makes the keyboard go away when you touch the Return key on the keyboard
- Optionally, implement the *touchesBegan* method
  - Makes the keyboard go away when you touch anywhere outside the text field or keyboard

Creating the First iOS Application

Creating the First iOS Application

A. Xcode starting up OR B. Xcode already running



Welcome to Xcode

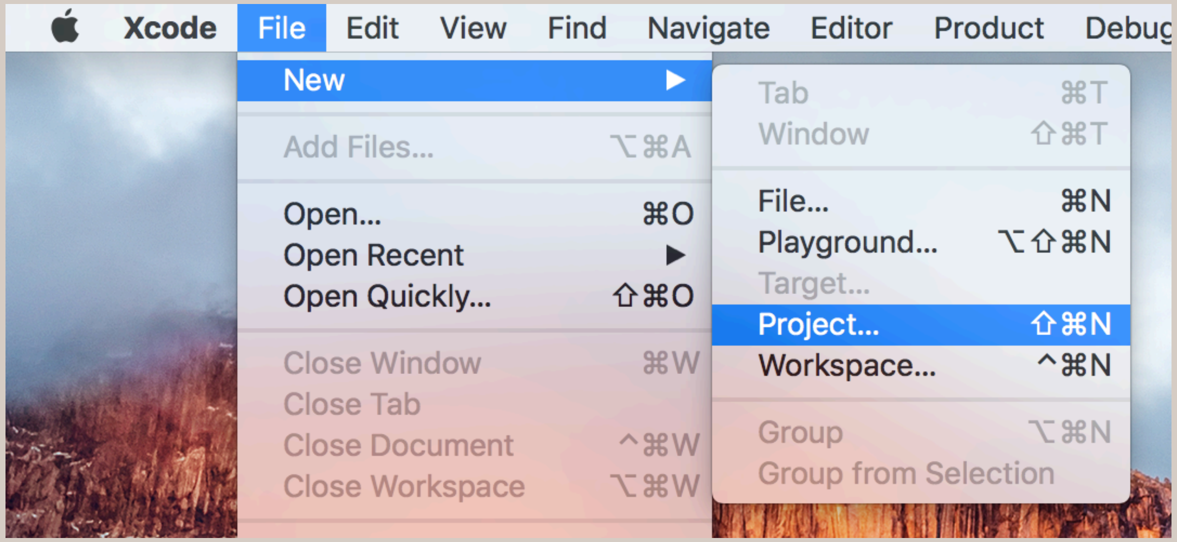
Version 8.2.1 (8C1002)

Get started with a playground
Explore new ideas quickly and easily.

Create a new Xcode project
Create an app for iPhone, iPad, Mac, Apple Watch or Apple TV.

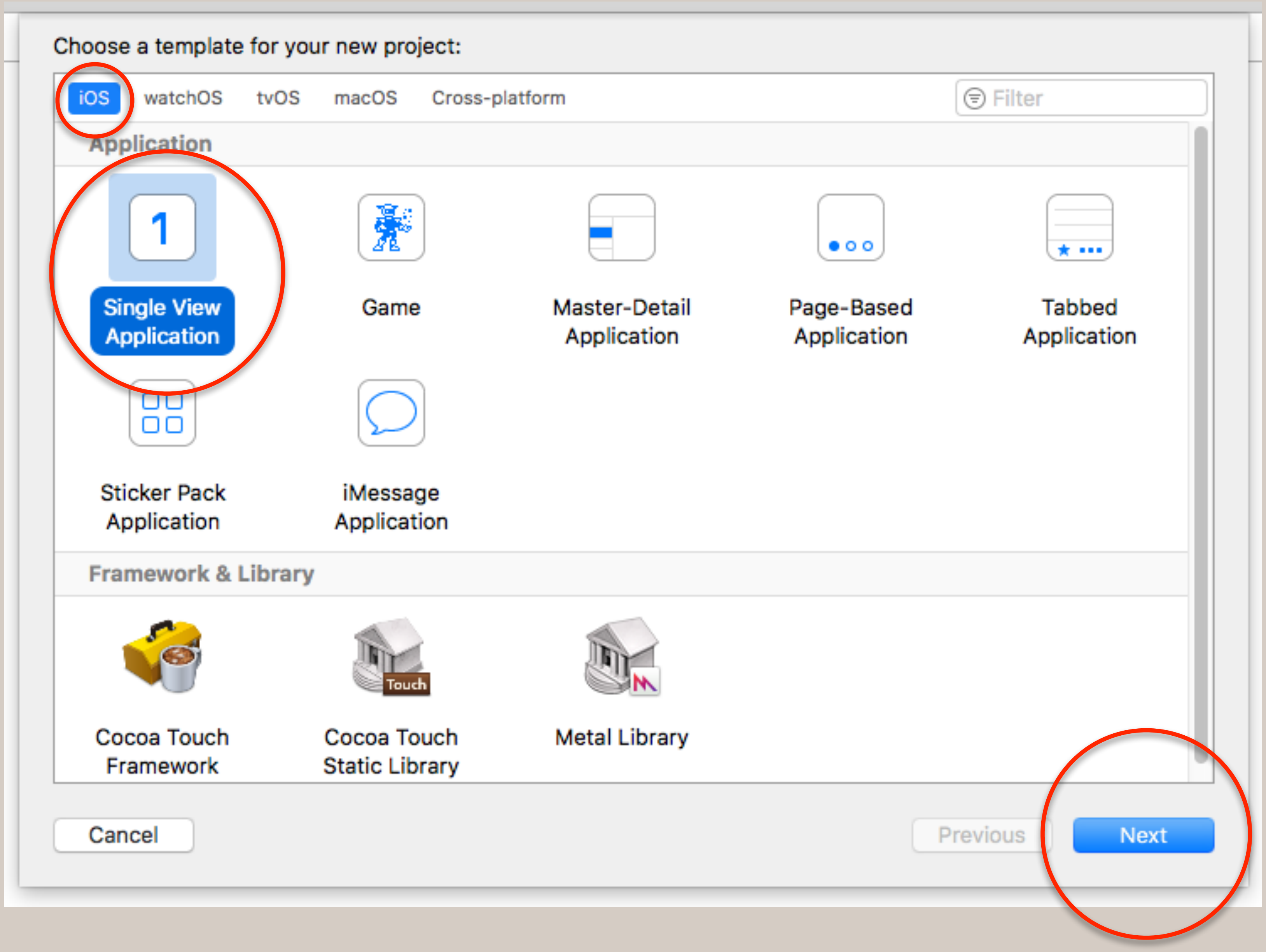
Check out an existing project
Start working on something from an SCM repository.

Xcode: File -> New -> Project



Creating the First iOS Application

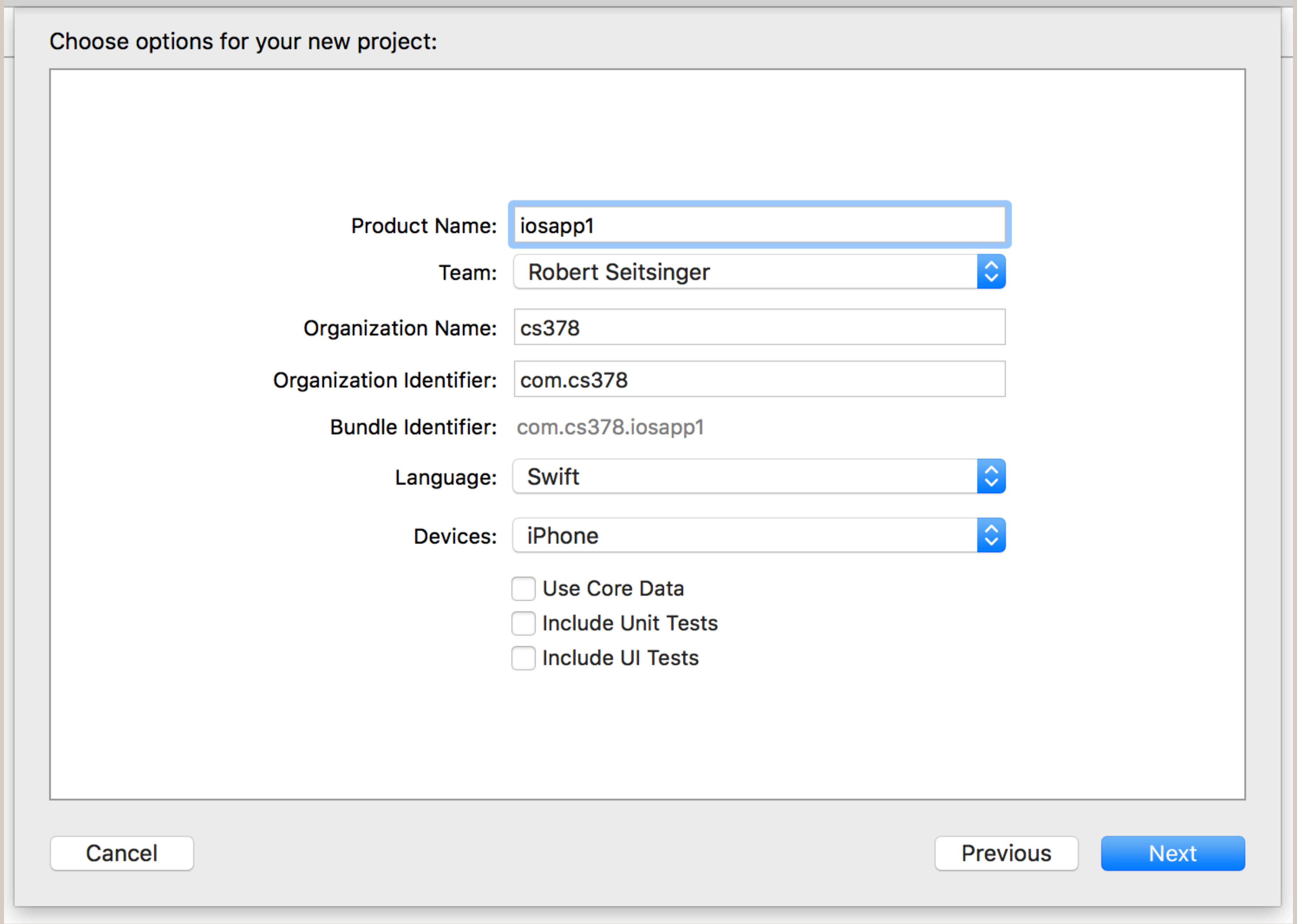
Select Single View iOS application:



Choose a template for your new project:

Creating the First iOS Application

Enter/Select project options:



Choose options for your new project:

Product Name: iosapp1

Team: Robert Seitsinger

Organization Name: cs378

Organization Identifier: com.cs378

Bundle Identifier: com.cs378.iosapp1

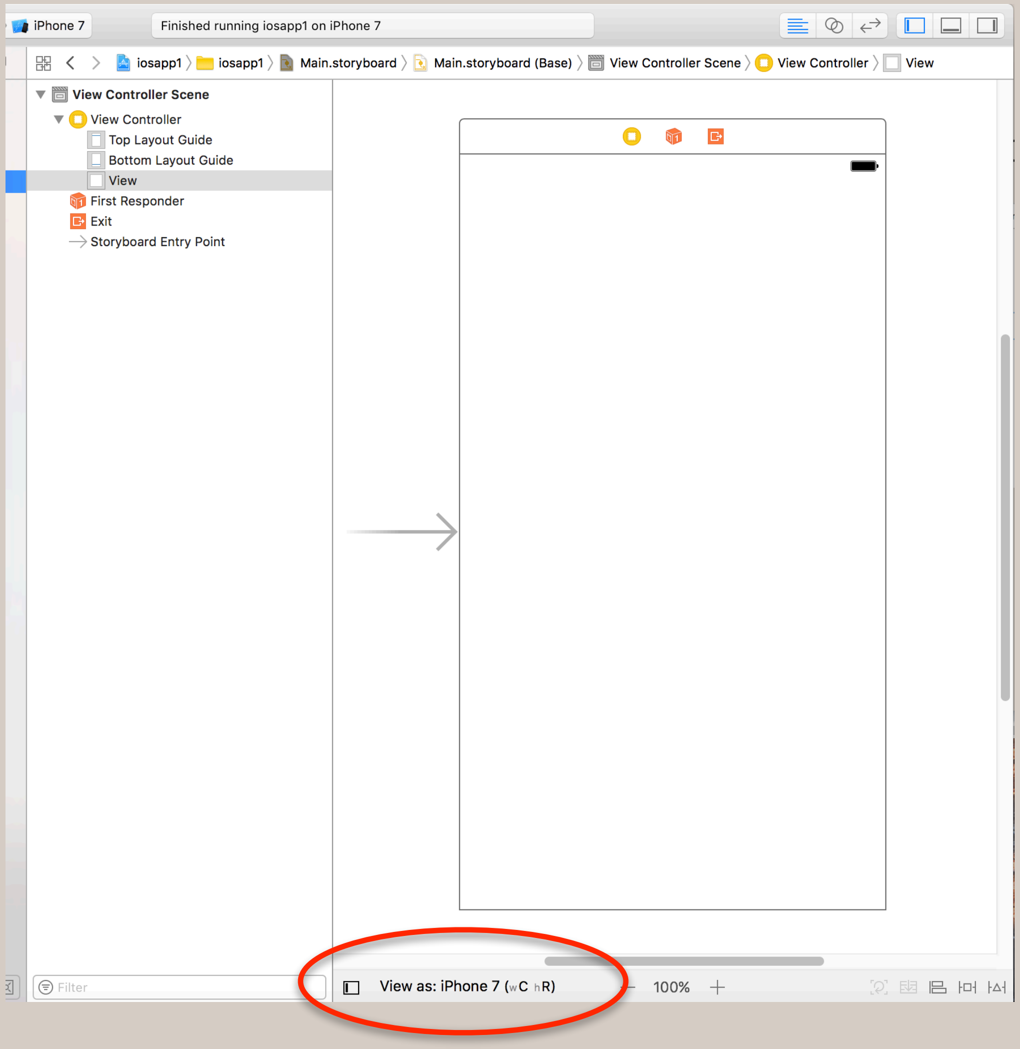
Language: Swift

Devices: iPhone

□ Use Core Data
□ Include Unit Tests
□ Include UI Tests

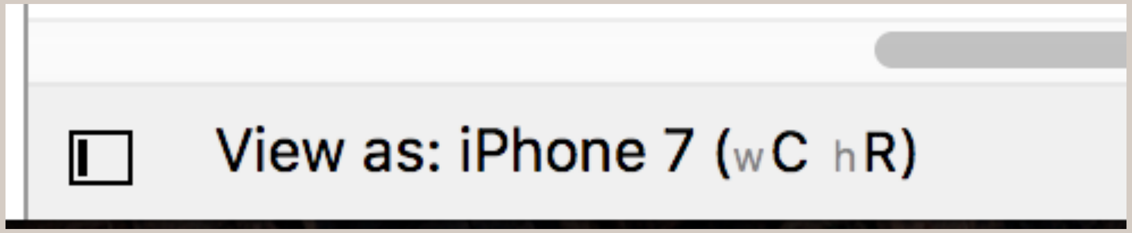
Creating the First iOS Application

Setting device size in storyboard:


iPhone 7
Finished running iosapp1 on iPhone 7
iosapp1 〉 iosapp1 〉 Main.storyboard 〉 Main.storyboard (Base) 〉 View Controller Scene 〉 View Controller 〉 View
View Controller Scene
View Controller
Top Layout Guide
Bottom Layout Guide
View
First Responder
Exit
Storyboard Entry Point
Filter
100%

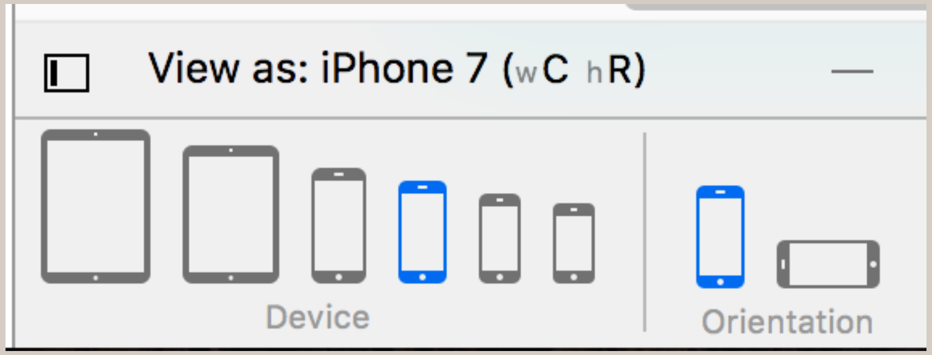
View as: iPhone 7 (wC hR)

Collapsed



View as: iPhone 7 (wC hR)

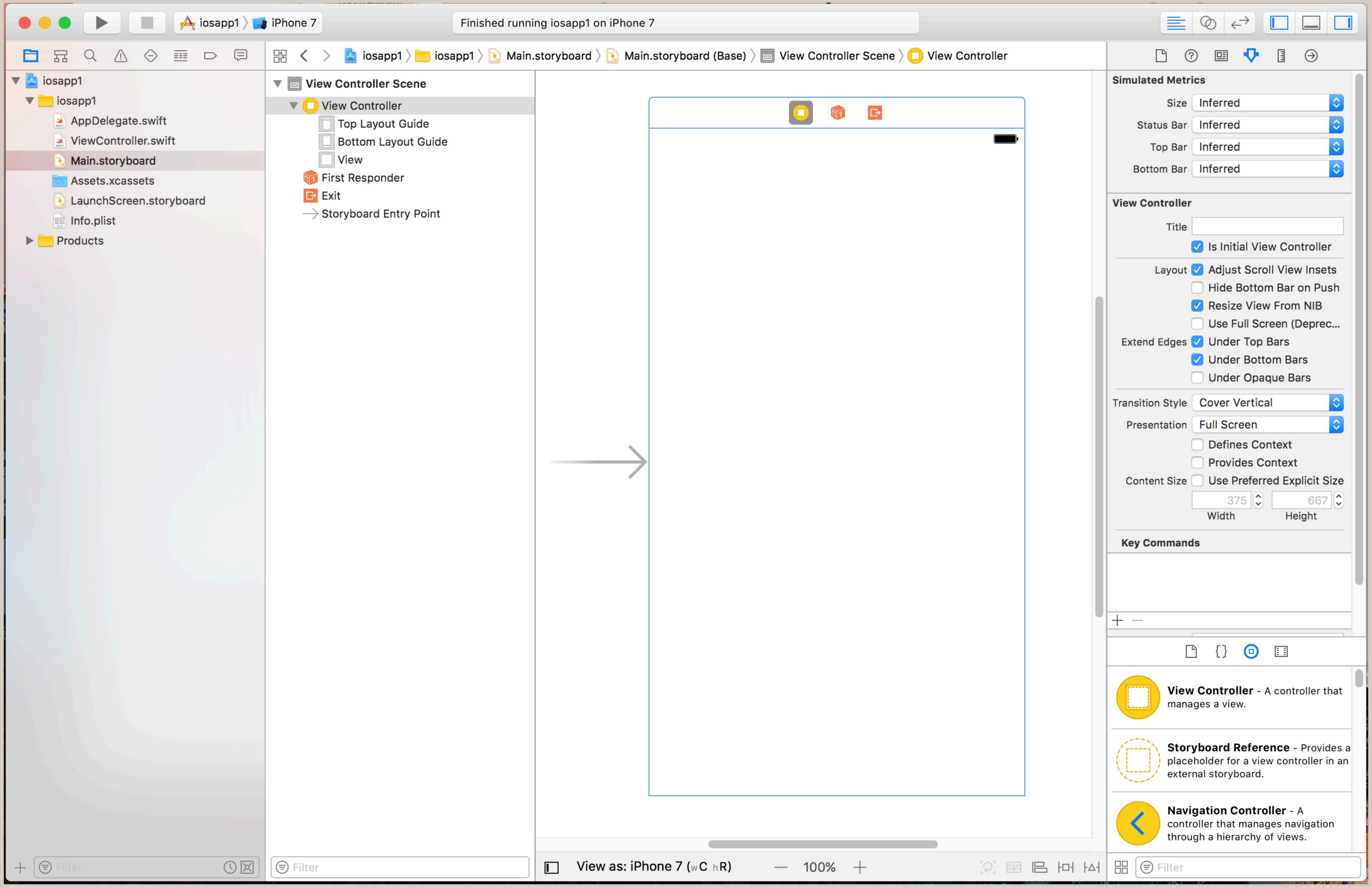
Expanded



View as: iPhone 7 (wC hR)

Creating the First iOS Application

Resulting project:



Finished running iosapp1 on iPhone 7

Simulated Metrics

Size Inferred

Status Bar Inferred

Top Bar Inferred

Bottom Bar Inferred

View Controller

Title

☑ Is Initial View Controller

Layout ☑ Adjust Scroll View Insets

□ Hide Bottom Bar on Push

☑ Resize View From NIB

□ Use Full Screen (Deprec...

Extend Edges ☑ Under Top Bars

☑ Under Bottom Bars

□ Under Opaque Bars

Transition Style Cover Vertical

Presentation Full Screen

□ Defines Context

□ Provides Context

Content Size □ Use Preferred Explicit Size

375 Width 667 Height

Key Commands

View Controller - A controller that manages a view.

Storyboard Reference - Provides a placeholder for a view controller in an external storyboard.

Navigation Controller - A controller that manages navigation through a hierarchy of views.

Homework

Homework 2

- Write a Single View iOS application in Swift
- Purpose is to get familiar with developing basic iOS applications using Swift
- Posted to Canvas
- Due in 1 week