CS2048: Introduction to iPhone Development

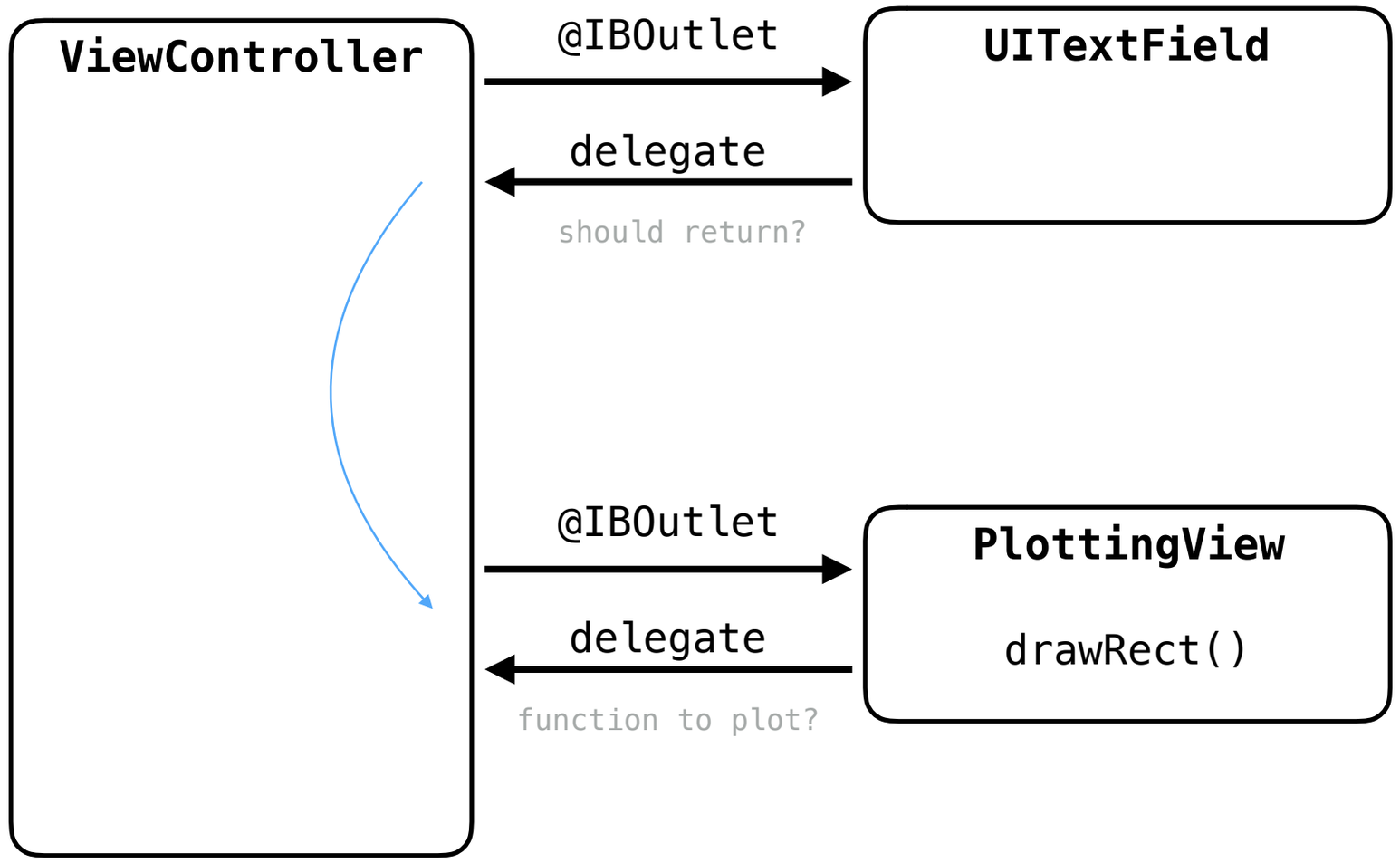
Lecture 3

Instructor: ~~Daniel Hauagge~~ Craig Frey

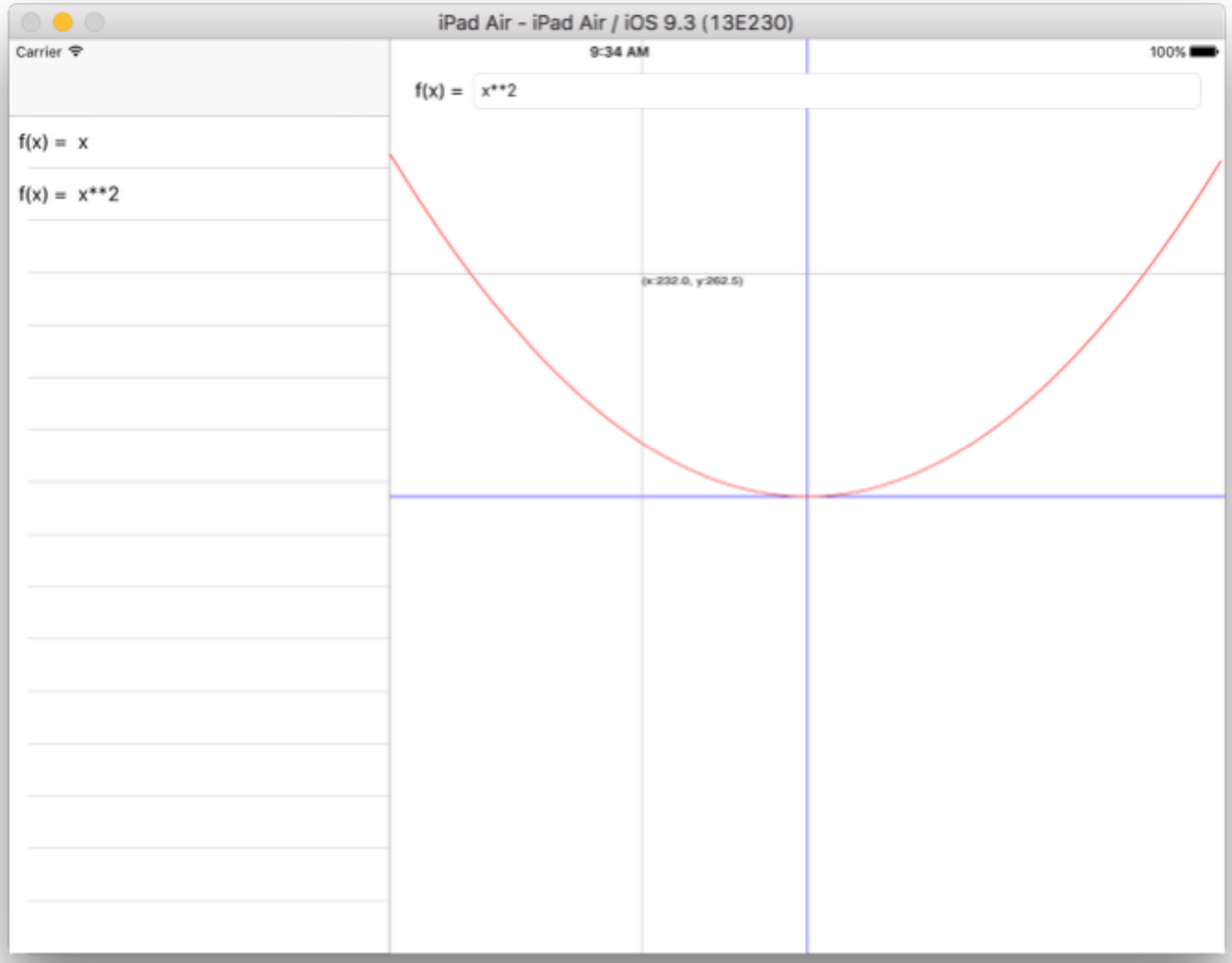
Today

- Continuation of MVC
  - Models
- Multiple view controllers
- UITableViews
- Segues

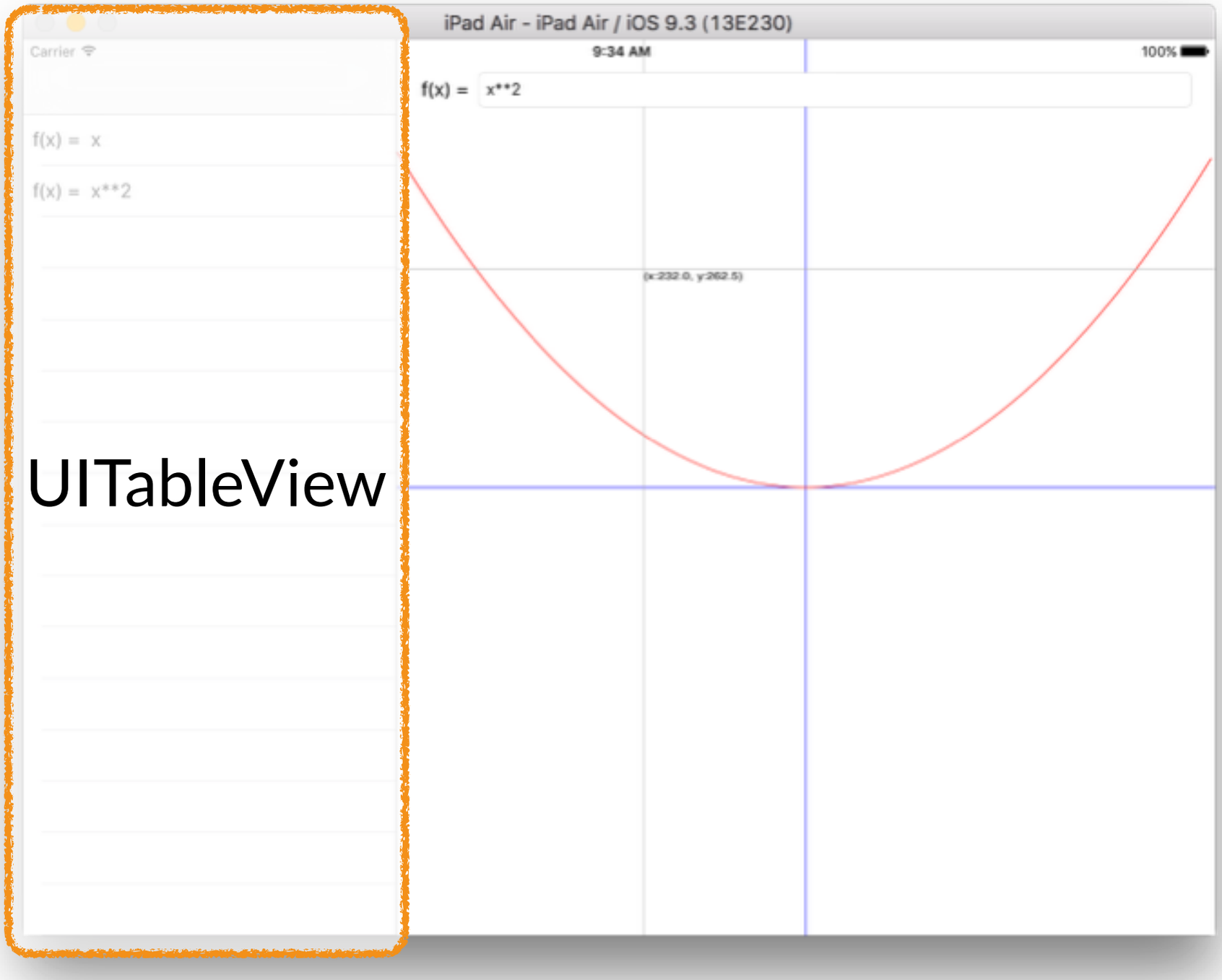
Recap of last lecture



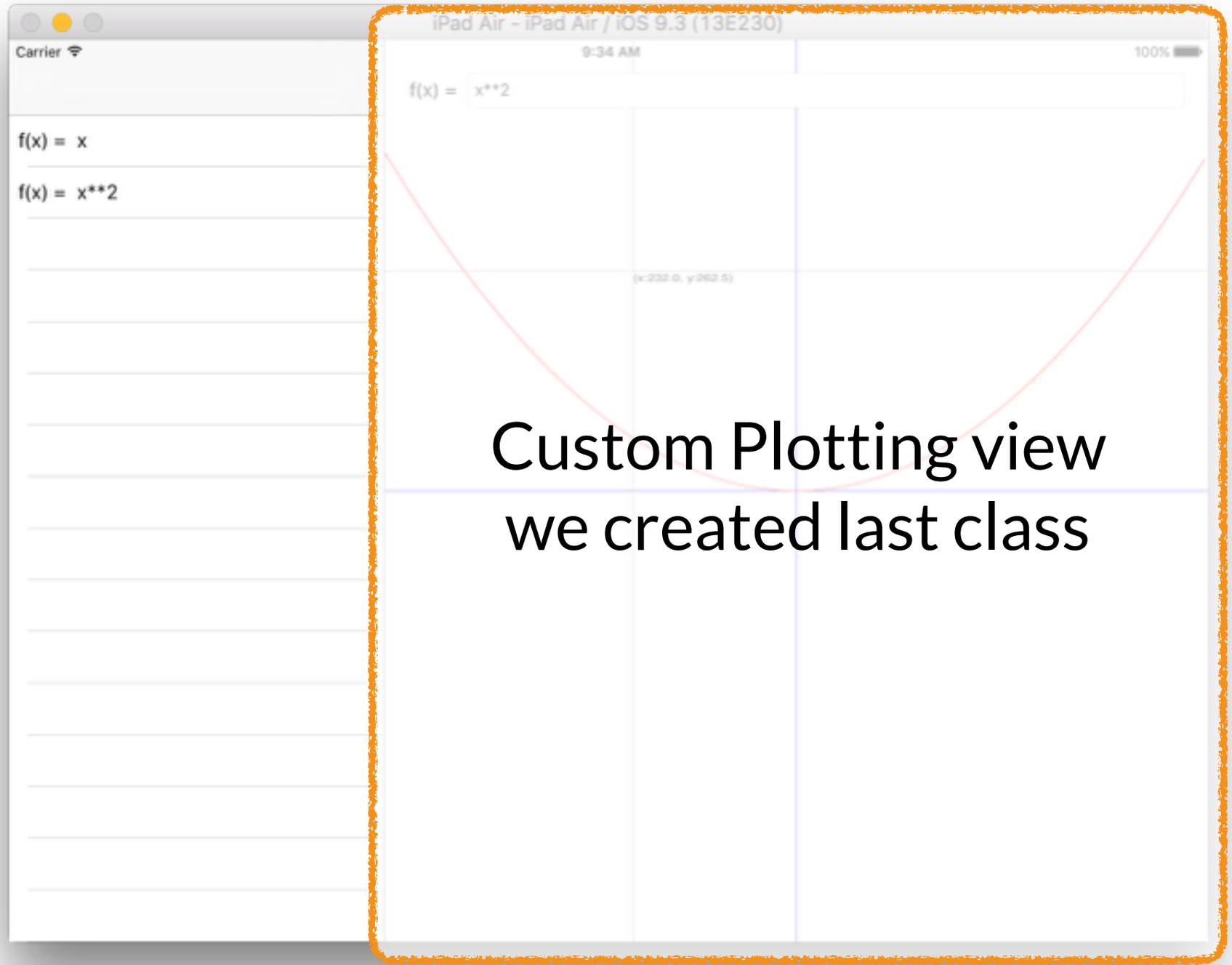
App


iPad Air - iPad Air / iOS 9.3 (13E230)
Carrier
9:34 AM
100%
f(x) = x**2
f(x) = x
f(x) = x**2
(x:232.0, y:262.5)

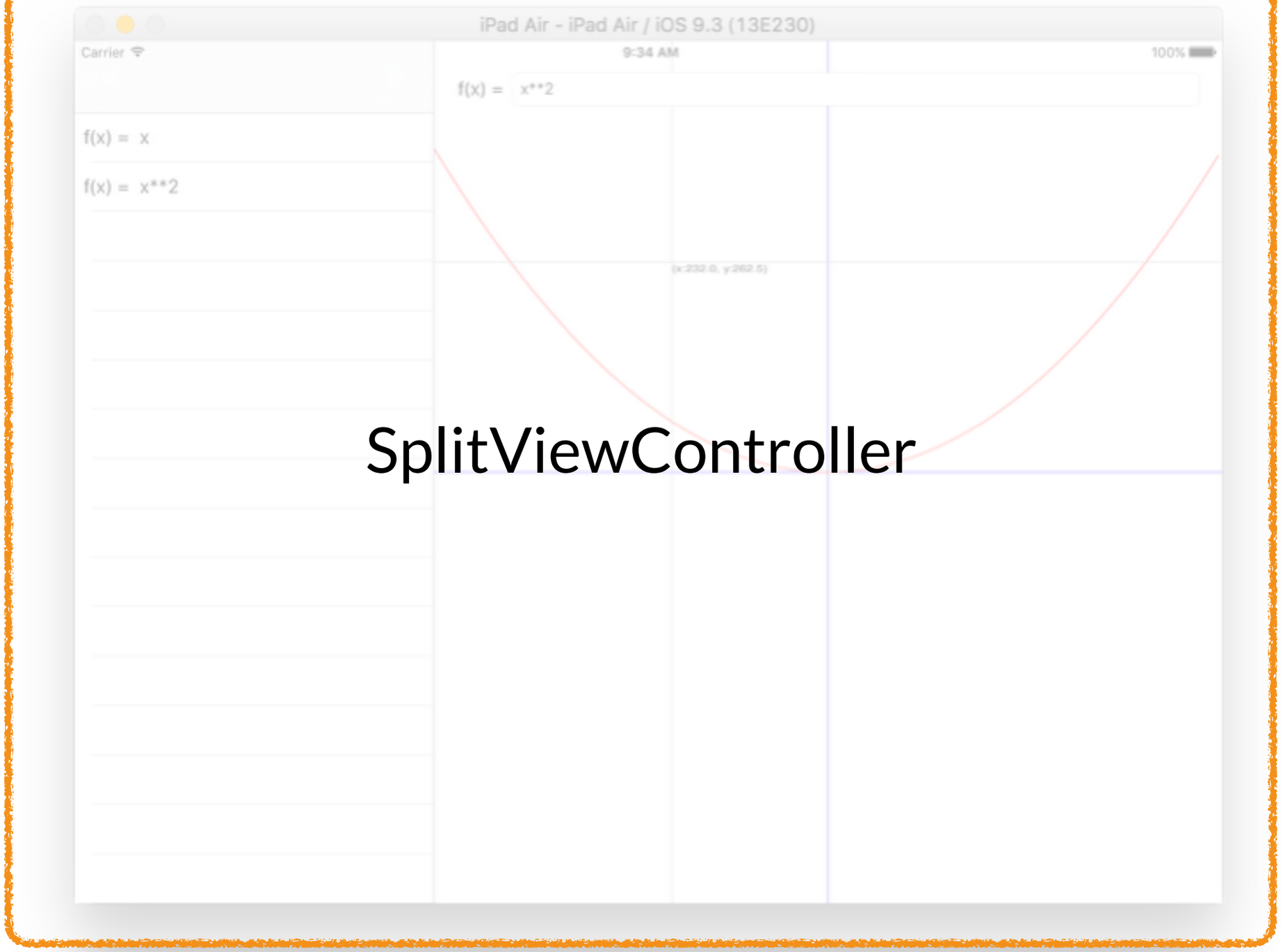
App


iPad Air - iPad Air / iOS 9.3 (13E230)
Carrier
9:34 AM
100%
f(x) = x**2
f(x) = x
f(x) = x**2
(x:232.0, y:262.5)
UITableView

App

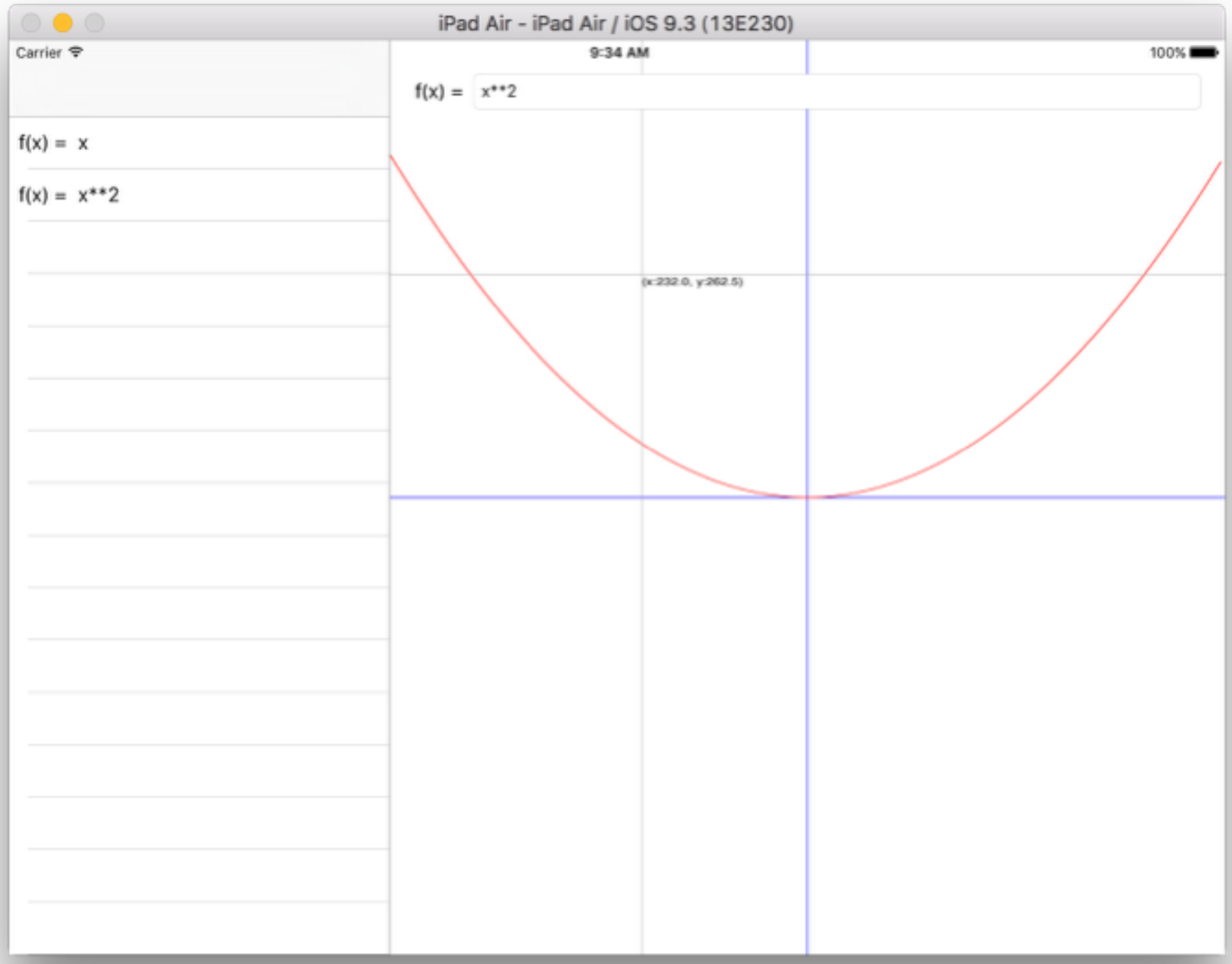

Carrier
f(x) = x
f(x) = x**2
iPad Air - iPad Air / iOS 9.3 (13E230)
9:34 AM
100%
f(x) = x**2
Custom Plotting view
we created last class

App

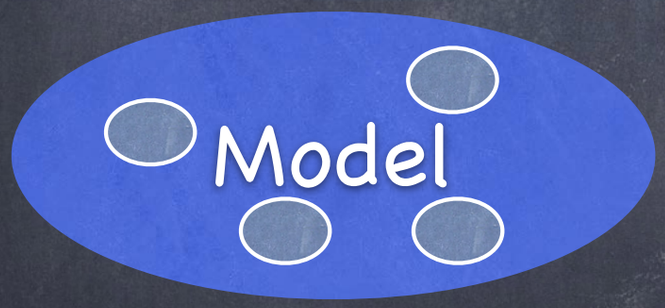

iPad Air - iPad Air / iOS 9.3 (13E230)
Carrier
9:34 AM
100%
f(x) = x**2
f(x) = x
f(x) = x**2
(x:232.0, y:262.5)
SplitViewController

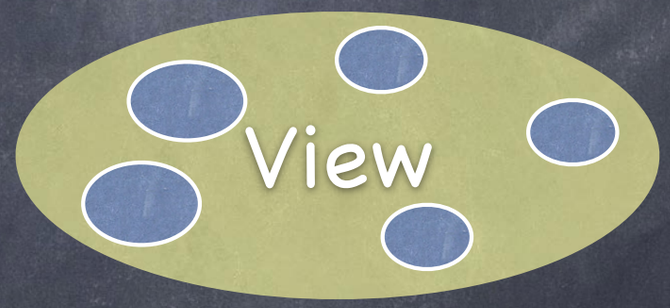
App

talk about split view controller



MVC



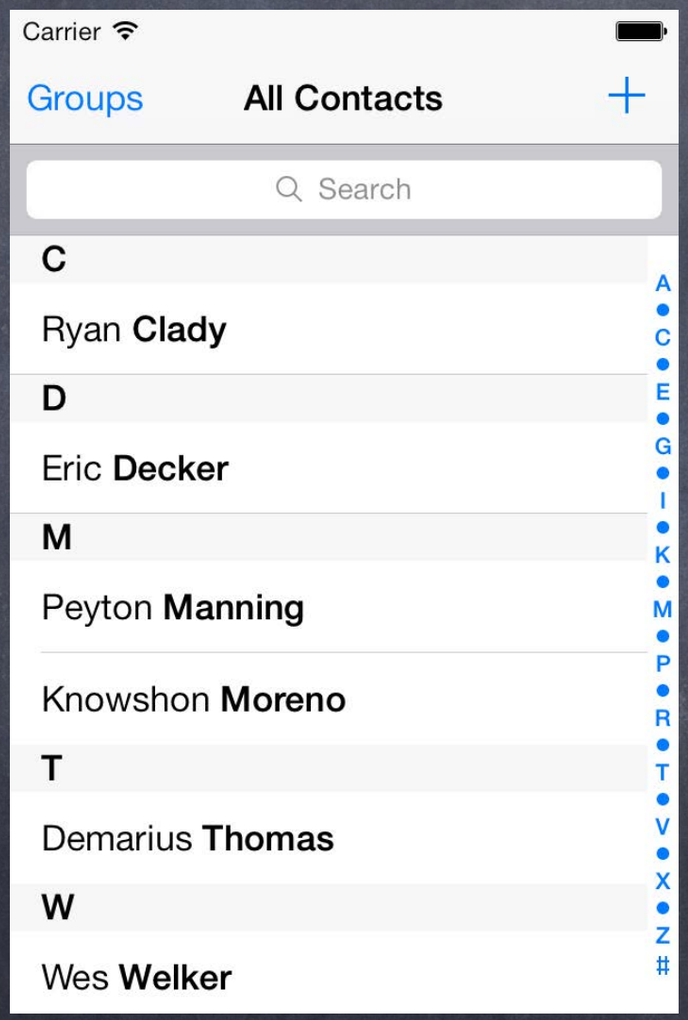


Divide objects in your program into 3 "camps."

UITableView

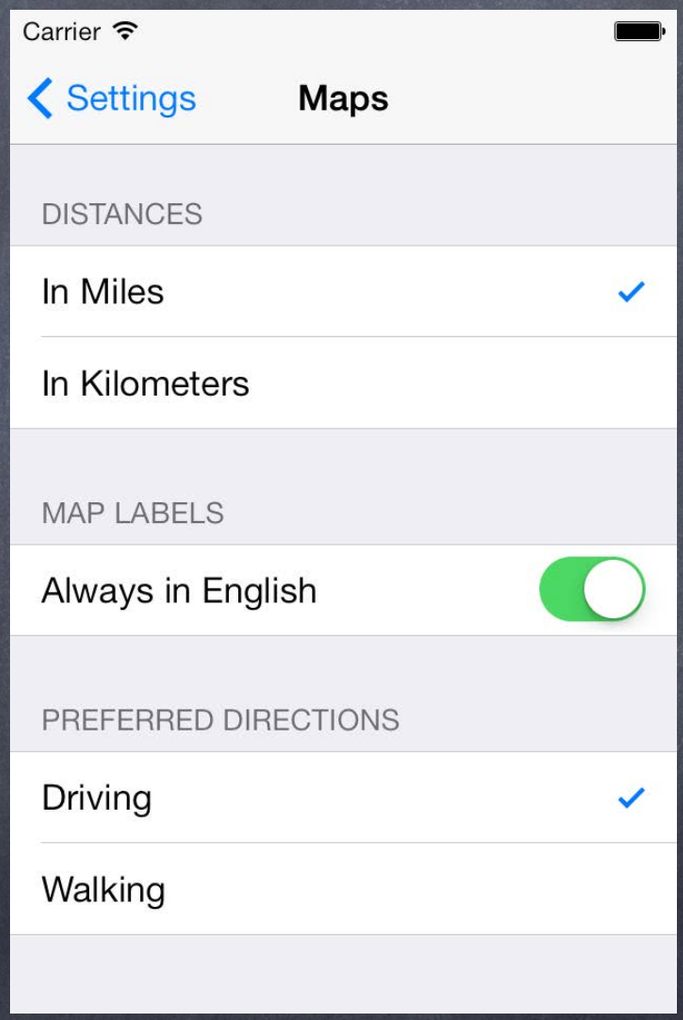
UITableView

UITableViewStyle.Plain



Dynamic (List) & Plain (ungrouped)

.Grouped



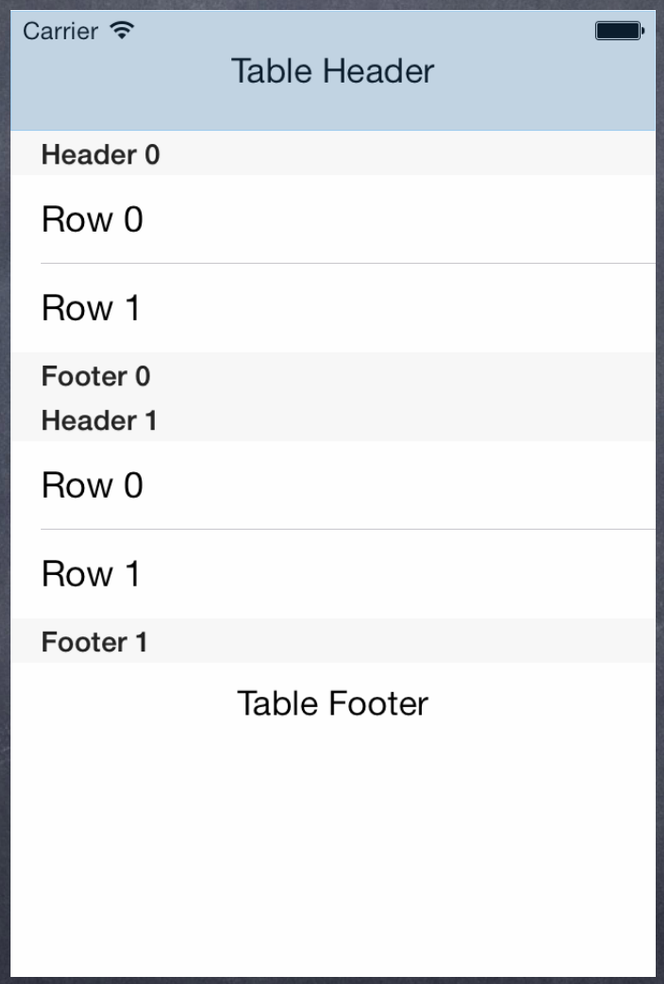
Static & Grouped

CS193p
Spring 2016

UITableView

Plain Style

Table Header



`var tableHeaderView: UIView`

CS193p
Spring 2016

UITableView

Plain Style

Table Header

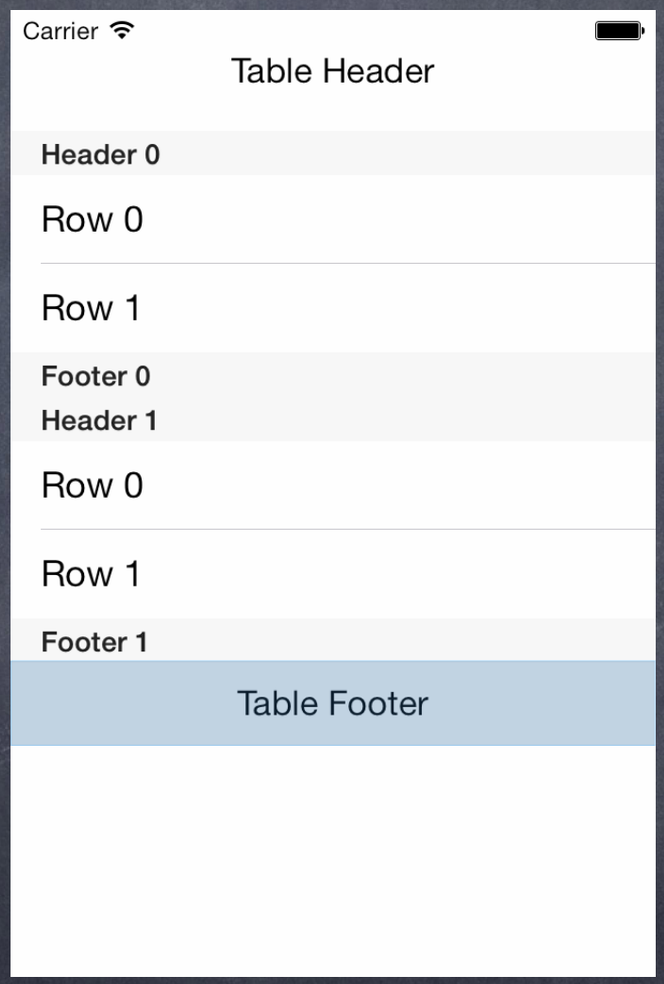


Table Footer

`var tableFooterView: UIView`

CS193p
Spring 2016

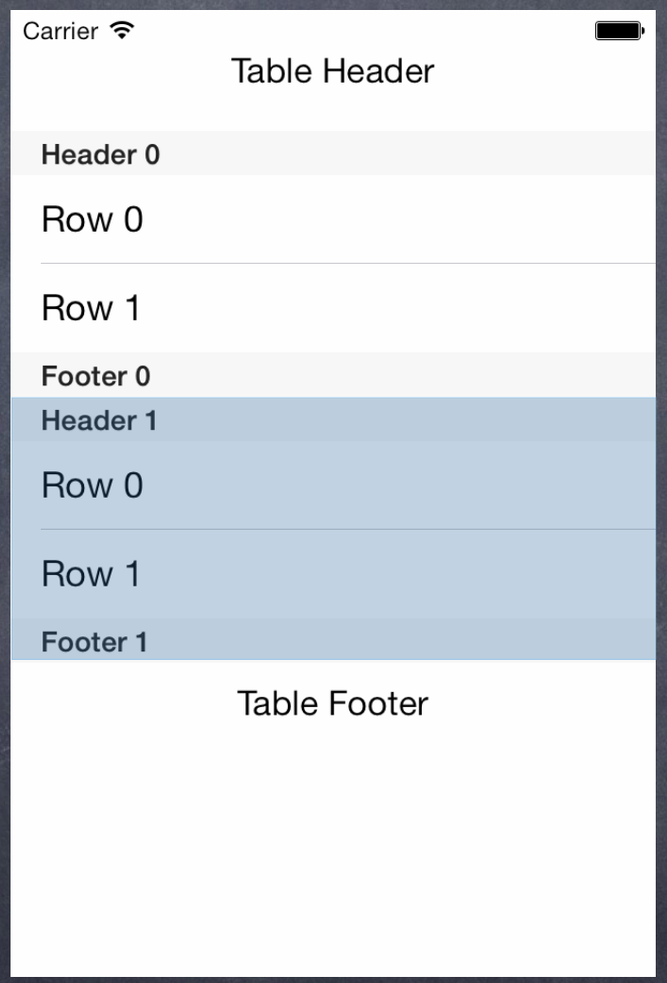
UITableView

Plain Style

Table Header

Section

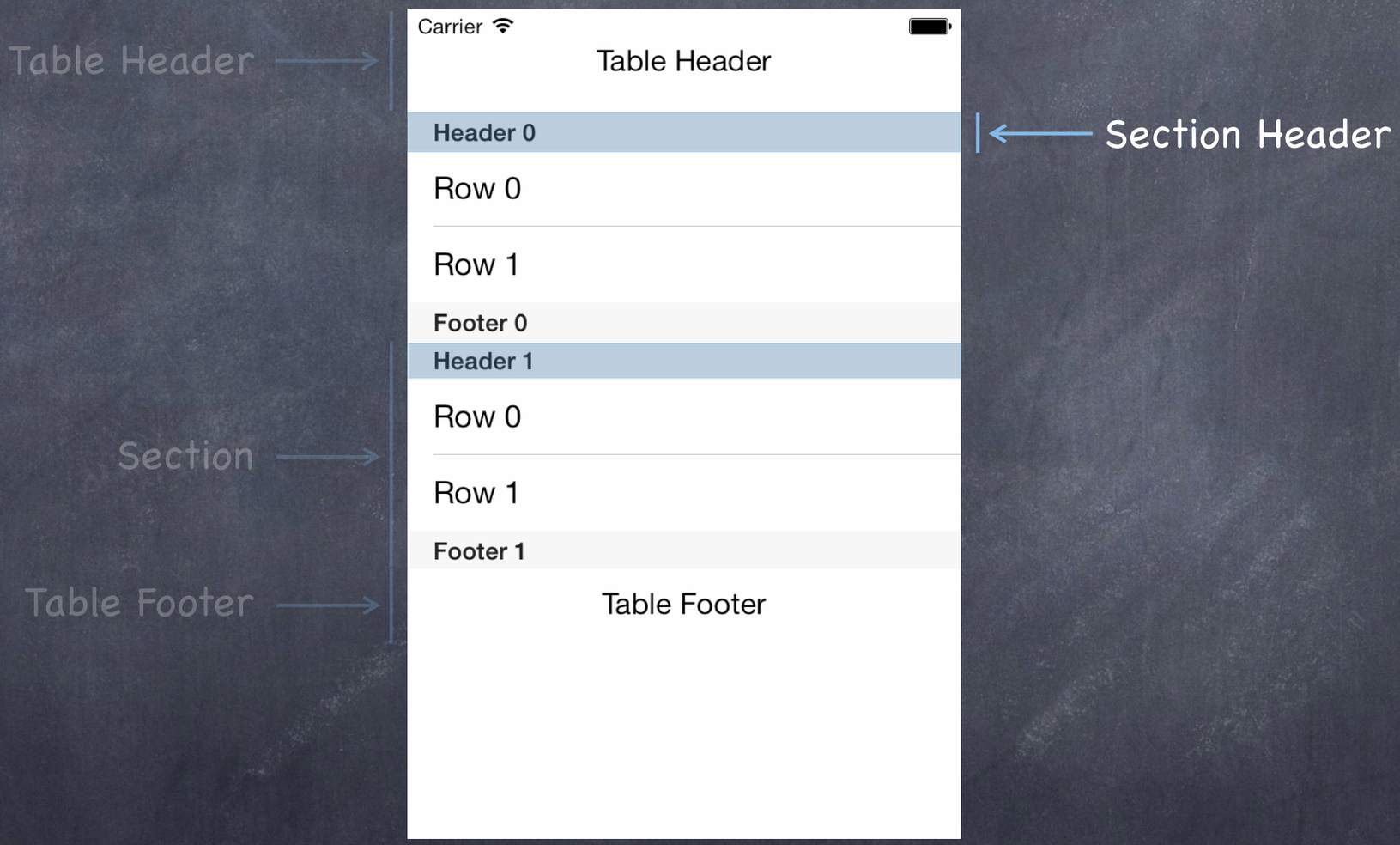
Table Footer



CS193p
Spring 2016

UITableView

Plain Style

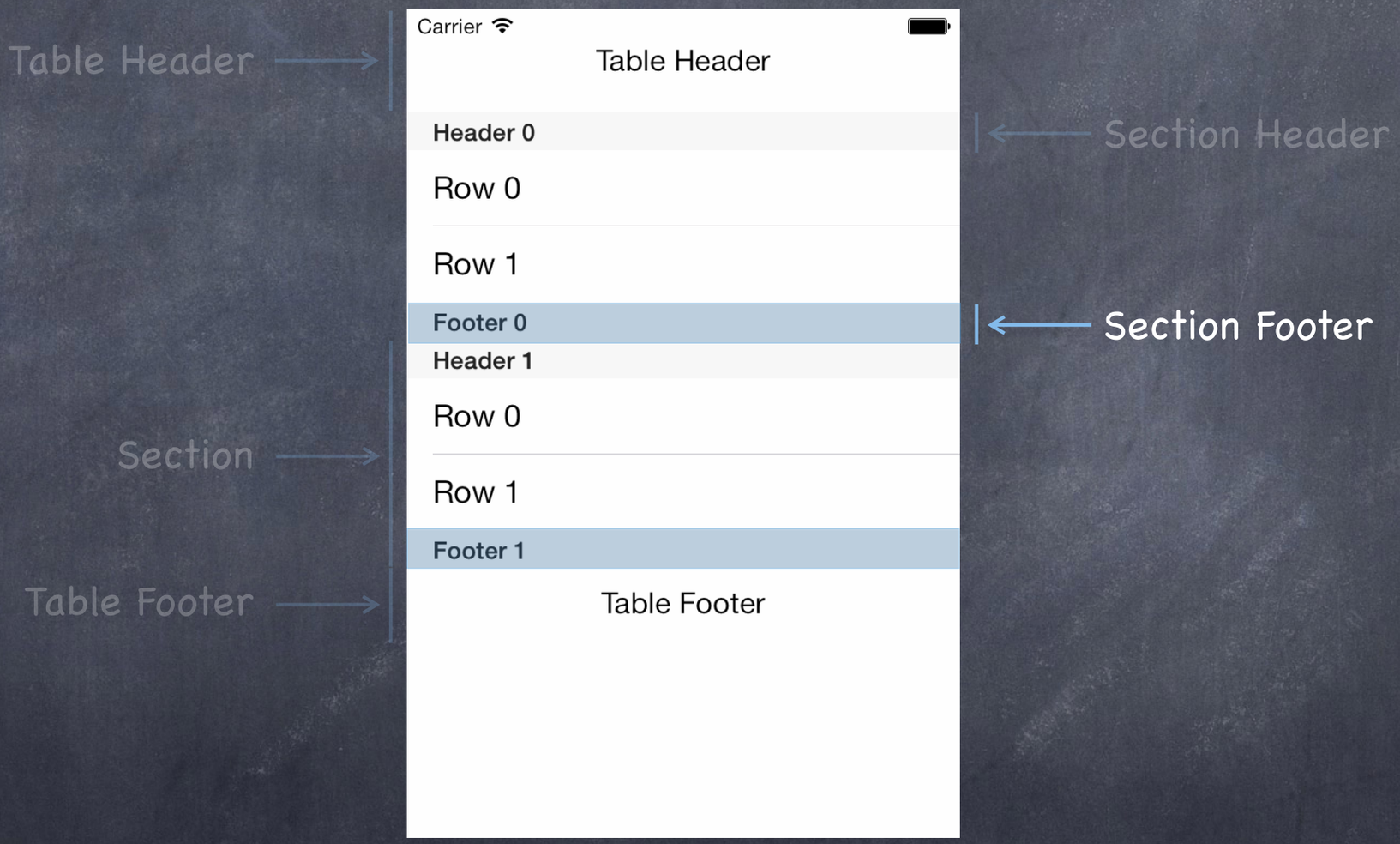


`UITableViewDataSource's tableView(UITableView, titleForHeaderInSection: Int)`

CS193p
Spring 2016

UITableView

Plain Style

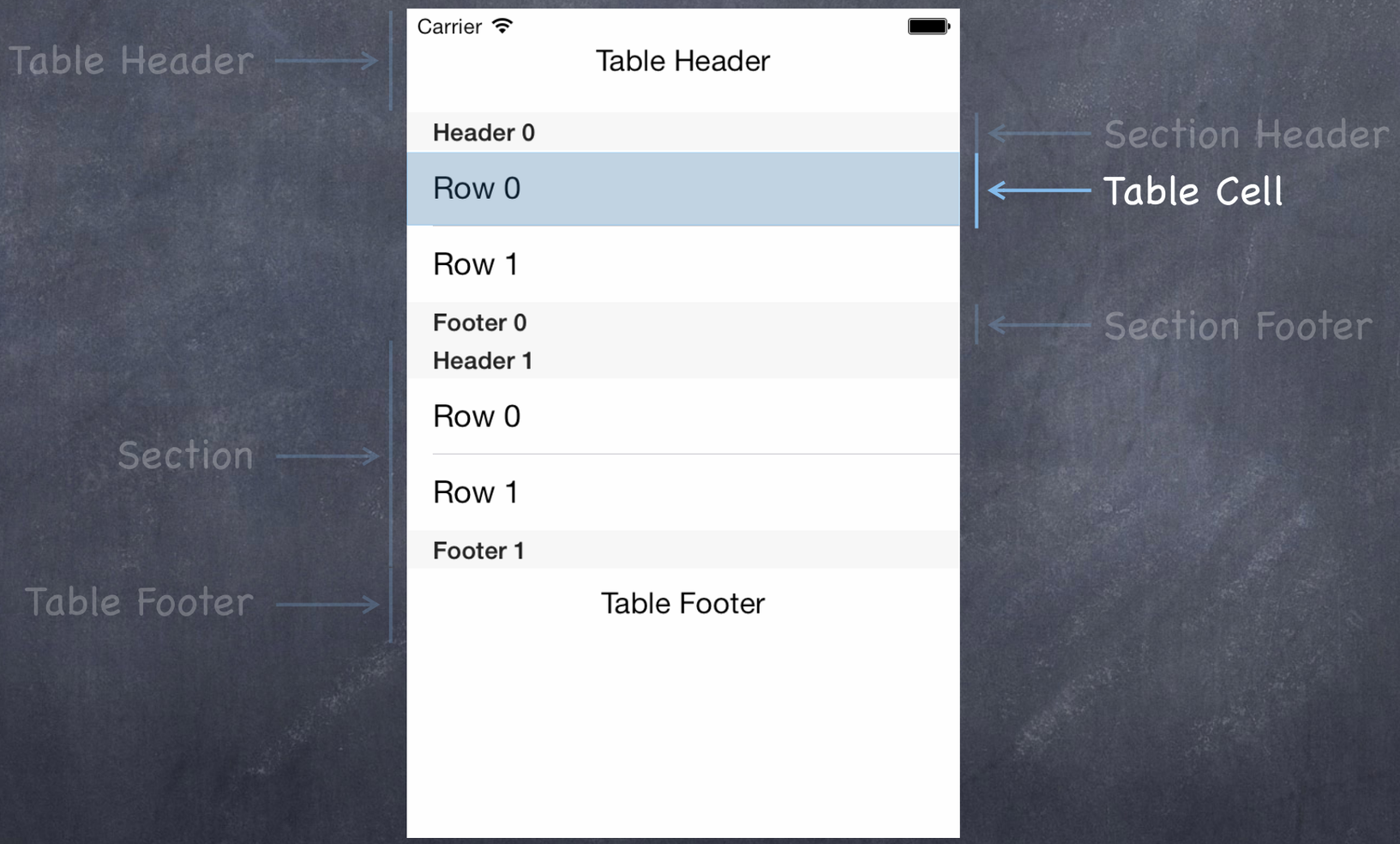


`UITableViewDataSource's tableView(UITableView, titleForFooterInSection: Int)`

CS193p
Spring 2016

UITableView

Plain Style

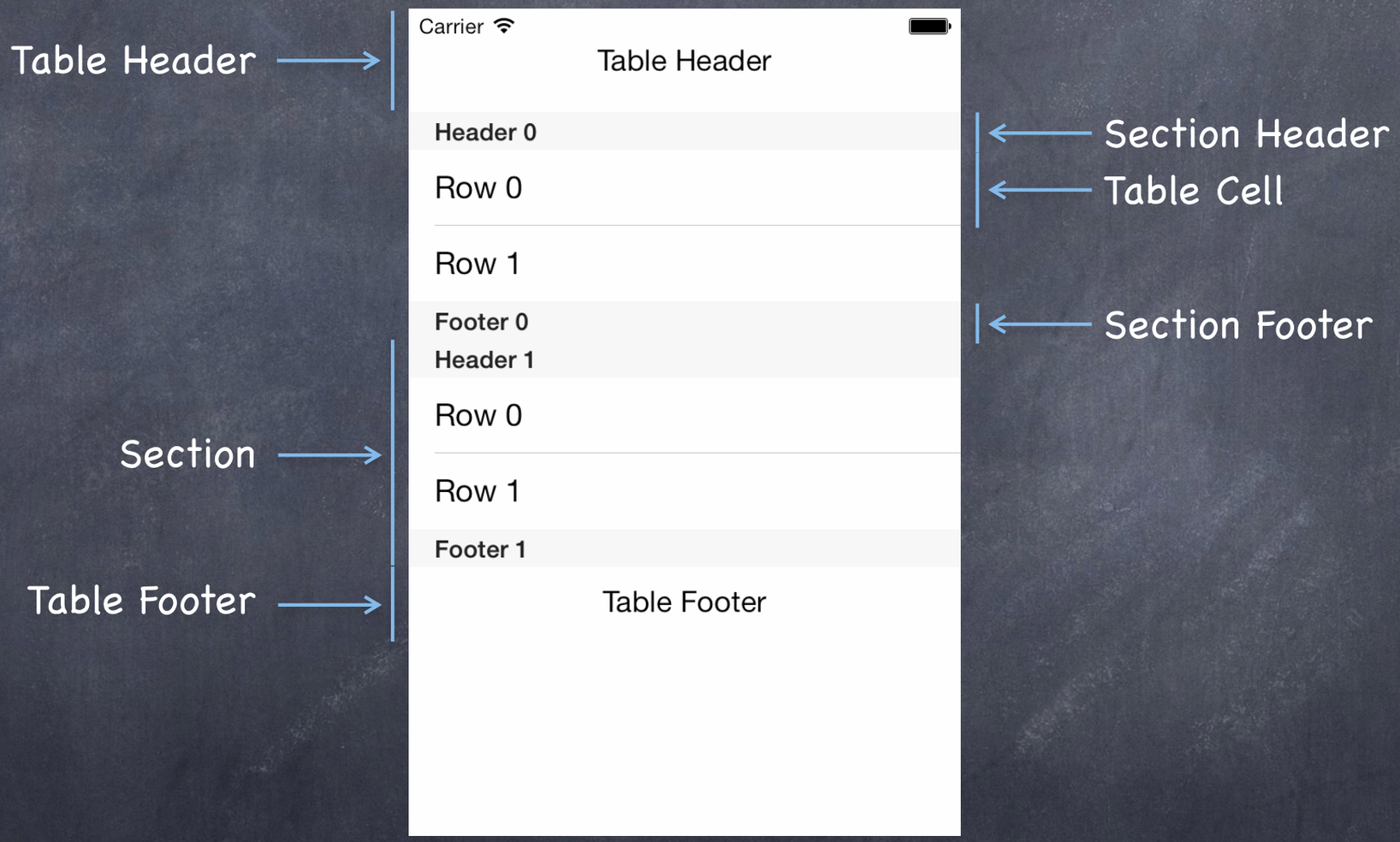


`UITableViewDataSource's tableView(UITableView, cellForRowAtIndexPath: NSIndexPath)`

CS193p
Spring 2016

UITableView

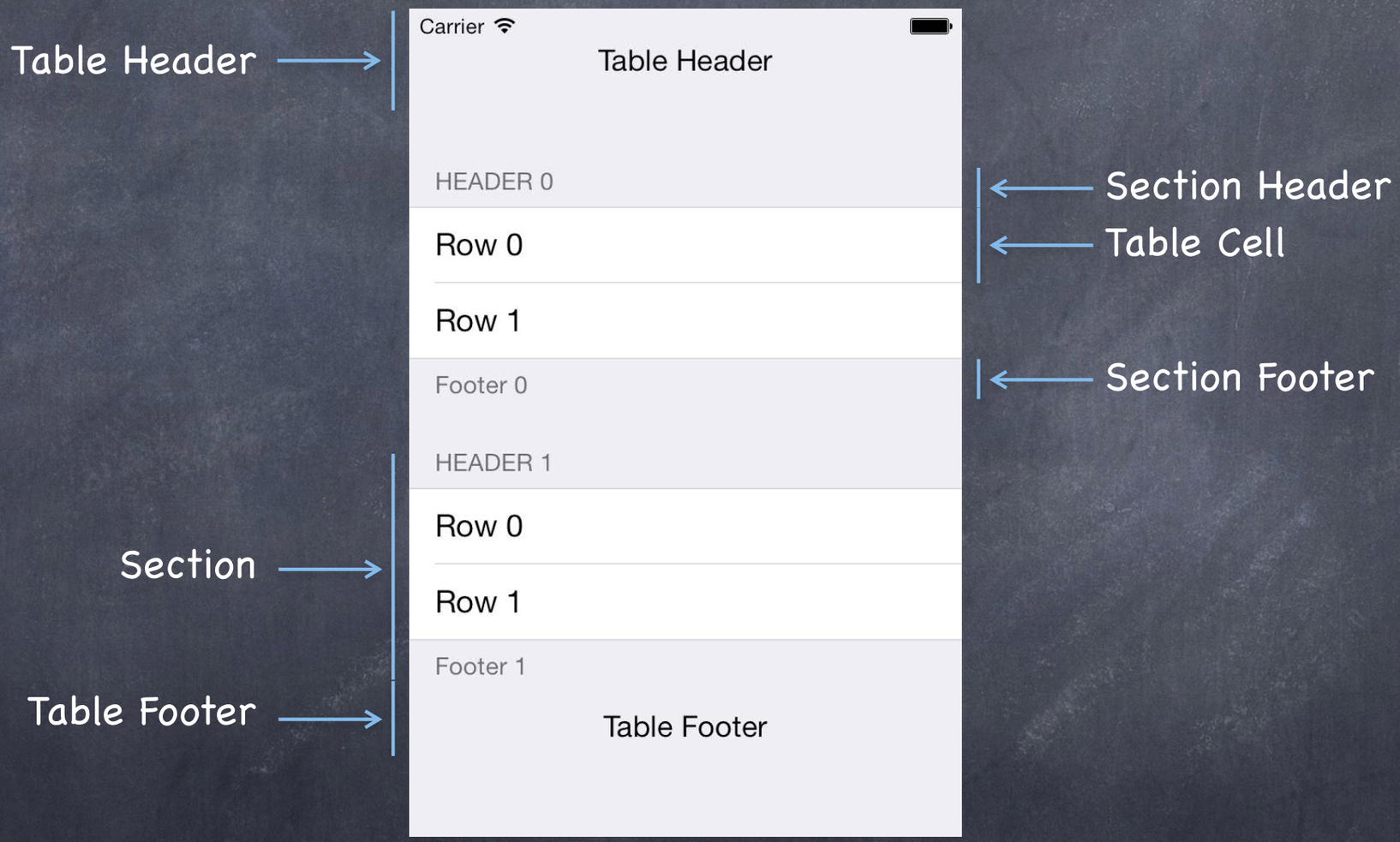
Plain Style



CS193p
Spring 2016

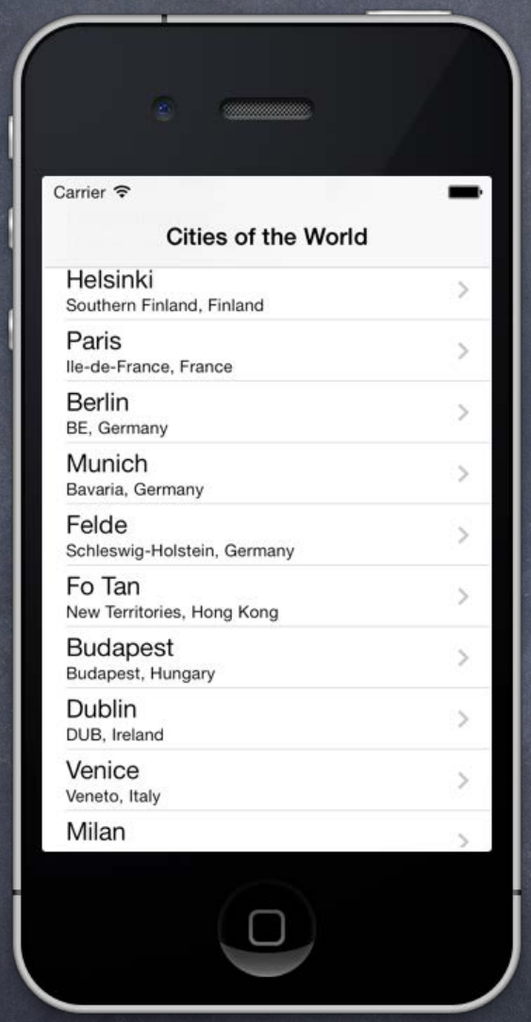
UITableView

Grouped Style

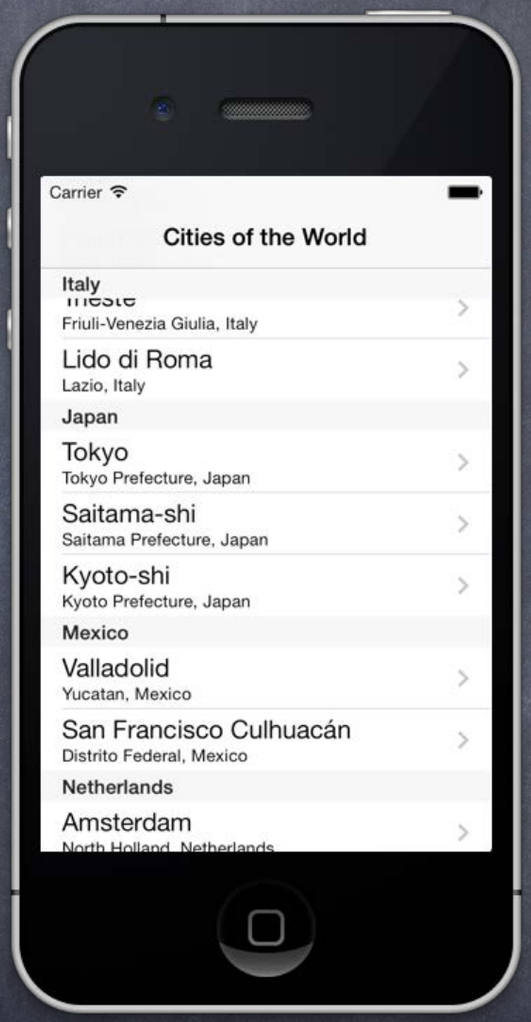


CS193p
Spring 2016

Sections or Not



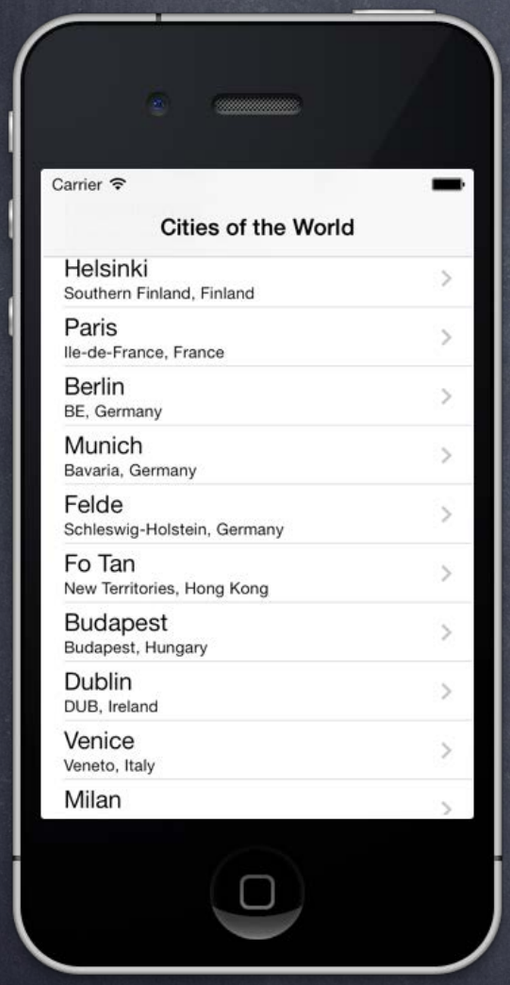
No Sections



Sections

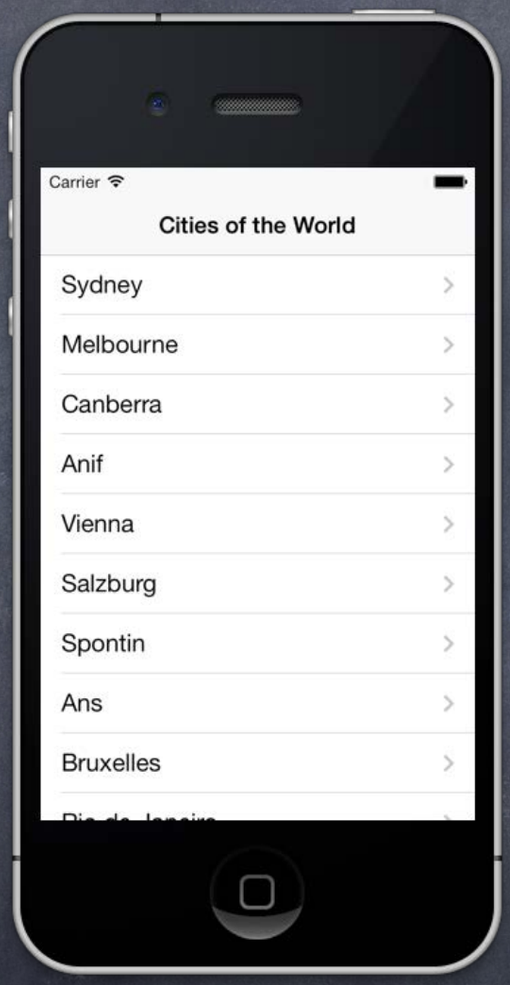
CS193p
Spring 2016

Cell Type



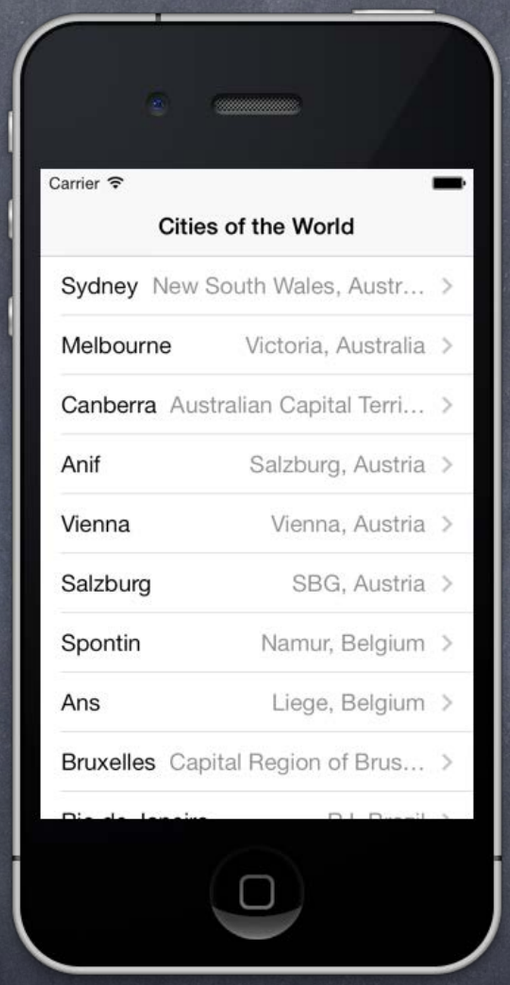
Subtitle

`UITableViewCellStyle.Subtitle`



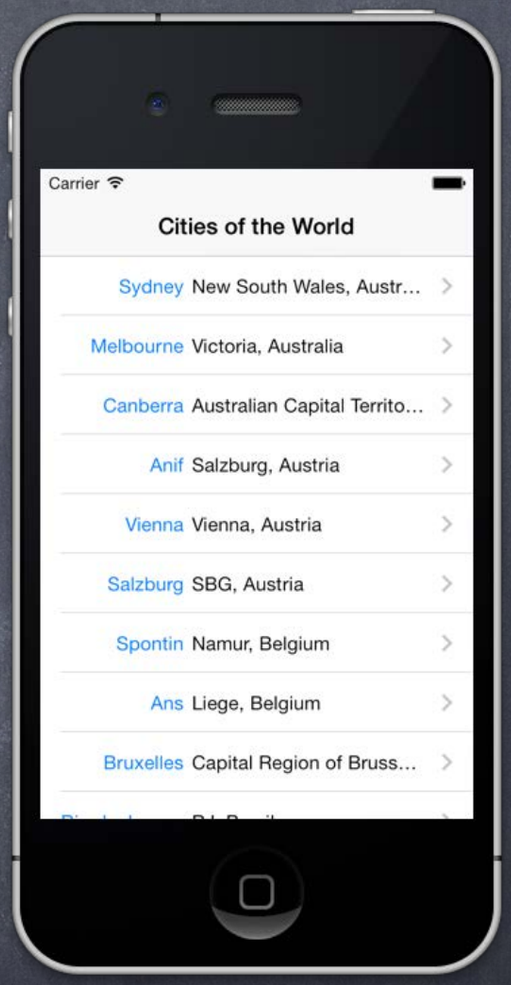
Basic

`.Default`



Right Detail

`.Value1`



Left Detail

`.Value2`

{ Coding Session }

UITableView

UITableView Protocols

- How to connect all this stuff up in code?

  Connections to code are made using the UITableView's dataSource and delegate

  The delegate is used to control how the table is displayed (it's look and feel)

  The dataSource provides the data that is displayed inside the cells

  UITableViewController automatically sets itself as the UITableView's delegate & dataSource

  Your UITableViewController subclass will also have a property pointing to the UITableView …

```
var tableView: UITableView // self.view in UITableViewController
```

- When do we need to implement the dataSource?

  Whenever the data in the table is dynamic (i.e. not static cells)

  There are three important methods in this protocol …

  How many sections in the table?

  How many rows in each section?

  Give me a view to use to draw each cell at a given row in a given section.

  Let's cover the last one first (since the first two are very straightforward) …

UITableViewDataSource

- How does a dynamic table know how many rows there are?
  - And how many sections, too, of course?
  - Via these `UITableViewDataSource` protocol methods …
  - `func numberOfSectionsInTableView(sender: UITableView) -> Int`
  - `func tableView(sender: UITableView, numberOfRowsInSection: Int) -> Int`
- Number of sections is 1 by default
  - In other words, if you don't implement `numberOfSectionsInTableView`, it will be 1
- No default for `numberOfRowsInSection`
  - This is a required method in this protocol (as is `cellForRowAtIndexPath`)
- What about a static table?
  - Do not implement these `dataSource` methods for a static table
  - `UITableViewController` will take care of that for you
  - You edit the data directly in the storyboard

UITableViewDataSource

- Summary

  Loading your table view with data is simple …

  1. set the table view's `dataSource` to your Controller (automatic with `UITableViewController`)
  2. implement `numberOfSectionsInTableView` and `numberOfRowsInSection`
  3. implement `cellForRowAtIndexPath` to return loaded-up `UITableViewCells`

- Section titles are also considered part of the table's "data"

  So you return this information via `UITableViewDataSource` methods …

  `func tableView(UITableView, titleFor{Header,Footer}InSection: Int) -> String`

  If a `String` is not sufficient, the `UITableView`'s `delegate` can provide a `UIView`

- There are a number of other methods in this protocol

  But we're not going to cover them in lecture

  They are mostly about dealing with editing the table by deleting/moving/inserting rows

  That's because when rows are deleted, inserted or moved, it would likely modify the Model

  (and we're talking about the `UITableViewDataSource` protocol here)

CS193p
Spring 2016

UITableViewDelegate

- So far we've only talked about the UITableView's dataSource
  - But UITableView has another protocol-driven delegate called its delegate
- The delegate controls how the UITableView is displayed
  - Not the data it displays (that's the dataSource's job), how it is displayed
- Common for dataSource and delegate to be the same object
  - Usually the Controller of the MVC containing the UITableView
  - Again, this is set up automatically for you if you use UITableViewController
- The delegate also lets you observe what the table view is doing
  - Especially responding to when the user selects a row
  - Usually you will just segue when this happens, but if you want to track it directly …

UITableView "Target/Action"

- UITableViewDelegate method sent when row is selected

  This is sort of like "table view target/action" (only needed if you're not segueing, of course)

  Example: if the master in a split view wants to update the detail without segueing to a new one

```
func tableView(sender: UITableView, didSelectRowAtIndexPath indexPath: NSIndexPath) {
    // go do something based on information about my Model
    // corresponding to indexPath.row in indexPath.section
    // maybe directly update the Detail if I'm the Master in a split view?
}
```

- Delegate method sent when Detail Disclosure button is touched 

```
func tableView(UITableView, accessoryButtonTappedForRowWithIndexPath: NSIndexPath)
```

  Again, you can just segue from that Detail Disclosure button if you prefer

UITableViewDelegate

- Lots and lots of other `delegate` methods
  - `will`/`did` methods for both selecting and deselecting rows
  - Providing `UIView` objects to draw section headers and footers
  - Handling editing rows (moving them around with touch gestures)
  - `willBegin`/`didEnd` notifications for editing
  - Copying/pasting rows

UITableView

- What if your Model changes?

  `func reloadData()`

  Causes the `UITableView` to call `numberOfSectionsInTableView` and `numberOfRowsInSection` all over again and then `cellForRowAtIndexPath` on each visible row

  Relatively heavyweight, but if your entire data structure changes, that's what you need

  If only part of your Model changes, there are lighter-weight reloaders, for example …

  ```
  func reloadRowsAtIndexPaths(indexPaths: [NSIndexPath],
                              withRowAnimation: UITableViewRowAnimation)
  ```

UITableView

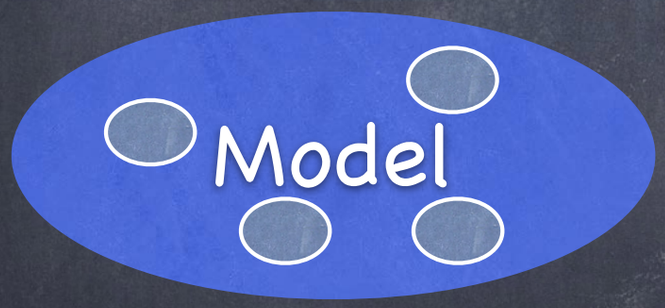
- Controlling the height of rows
  - Row height can be fixed (`UITableView`'s `var rowHeight: CGFloat`)
  - Or it can be determined using autolayout (`rowHeight = UITableViewAutomaticDimension`)
  - If you do automatic, help the table view out by setting `estimatedRowHeight` to something
  - The `UITableView`'s `delegate` can also control row heights …
  - `func tableView(UITableView, {estimated}heightForRowAtIndexPath: NSIndexPath) -> CGFloat`
  - Beware: the non-estimated version of this could get called A LOT if you have a big table

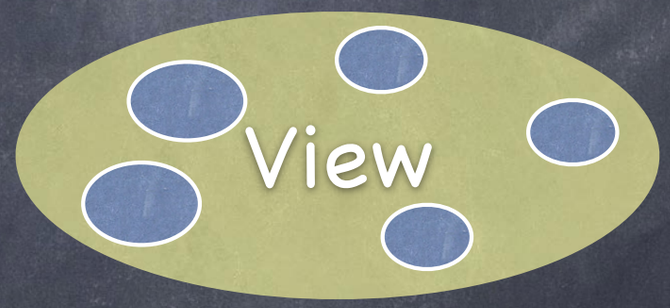
UITableView

- There are dozens of other methods in `UITableView` itself
  - Setting headers and footers for the entire table.
  - Controlling the look (separator style and color, default row height, etc.).
  - Getting cell information (cell for index path, index path for cell, visible cells, etc.).
  - Scrolling to a row.
  - Selection management (allows multiple selection, getting the selected row, etc.).
  - Moving, inserting and deleting rows, etc.
  - As always, part of learning the material in this course is studying the documentation

Multiple MVC's

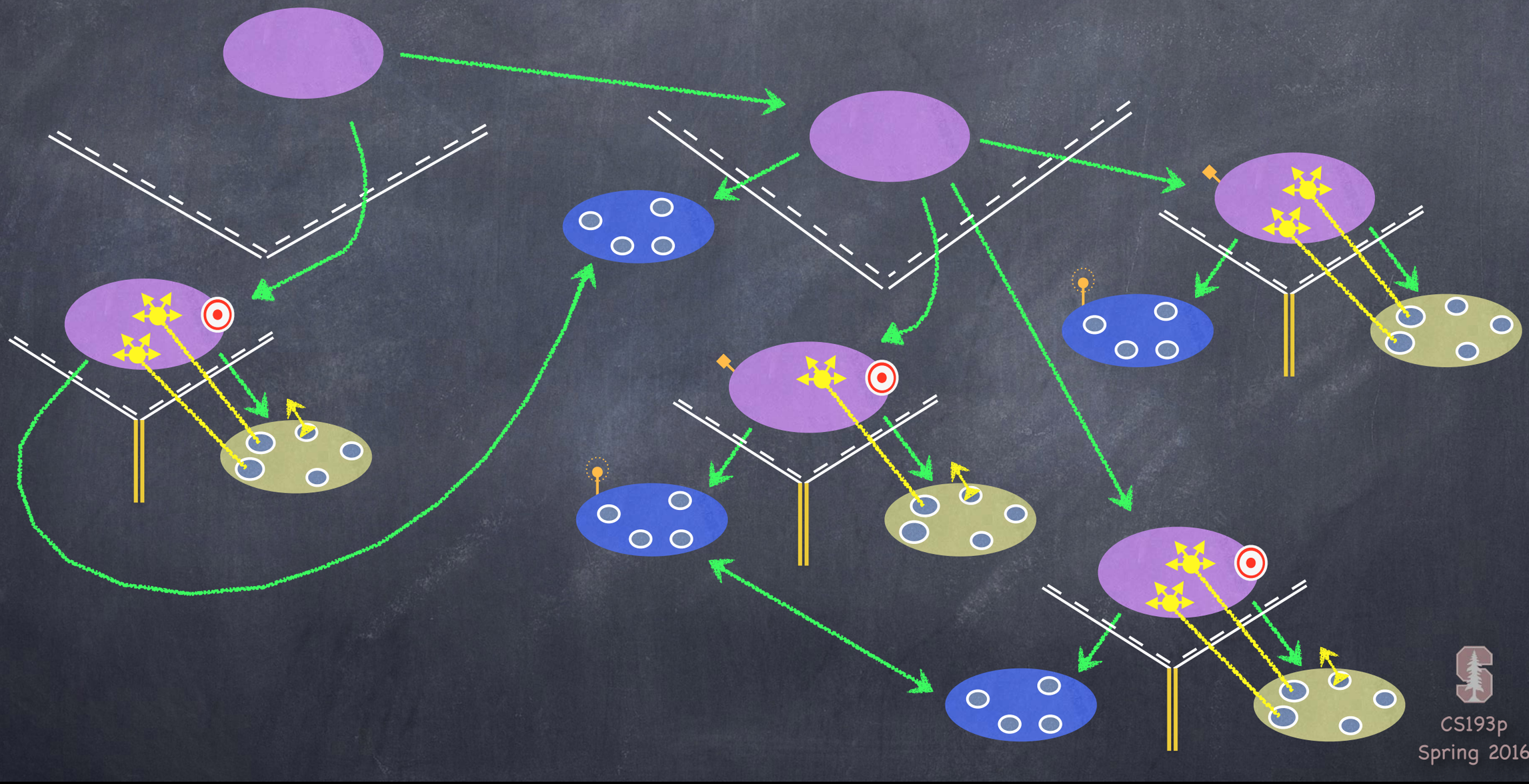
MVC





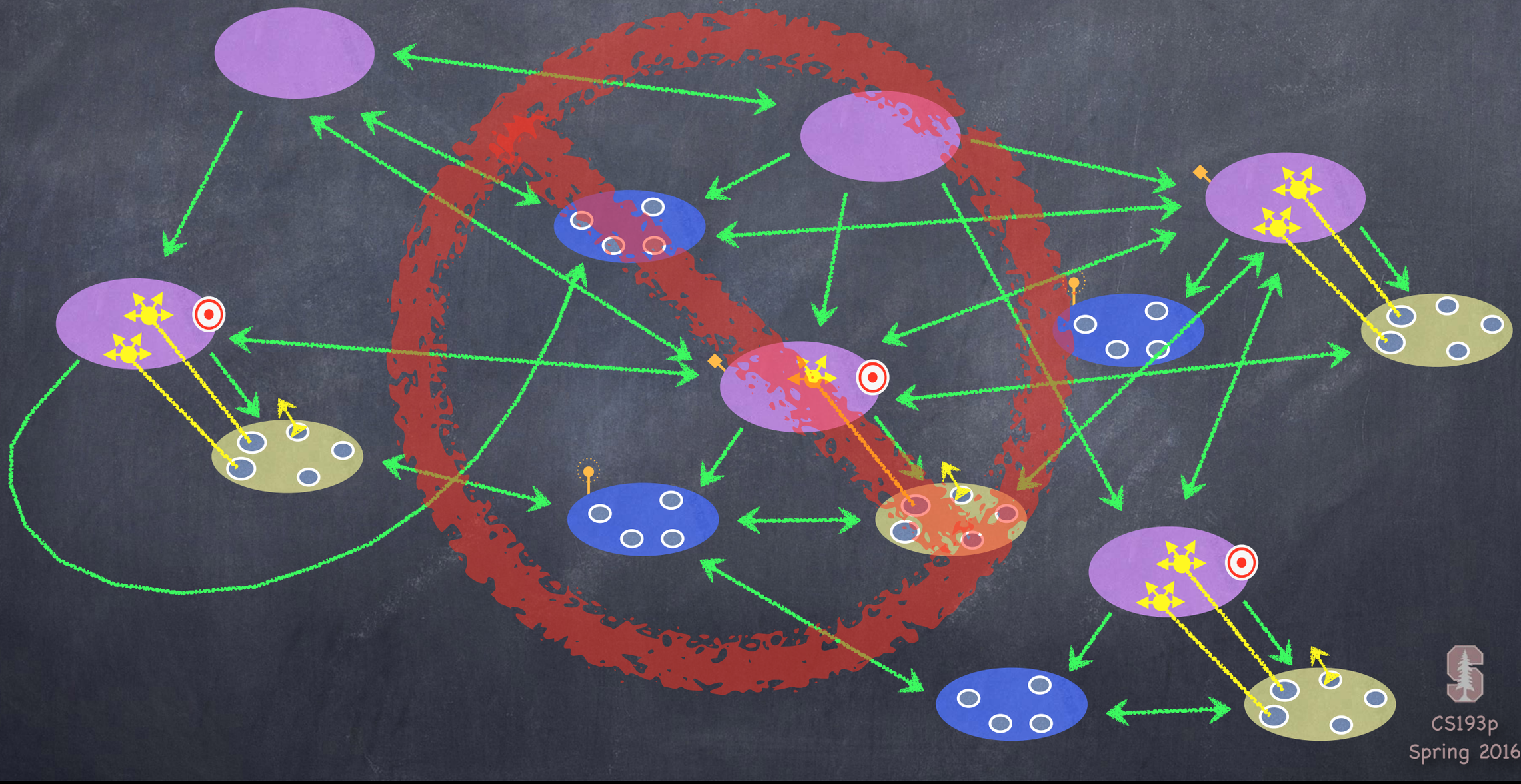
Divide objects in your program into 3 "camps."

MVCs working together



CS193p
Spring 2016

MVCs not working together



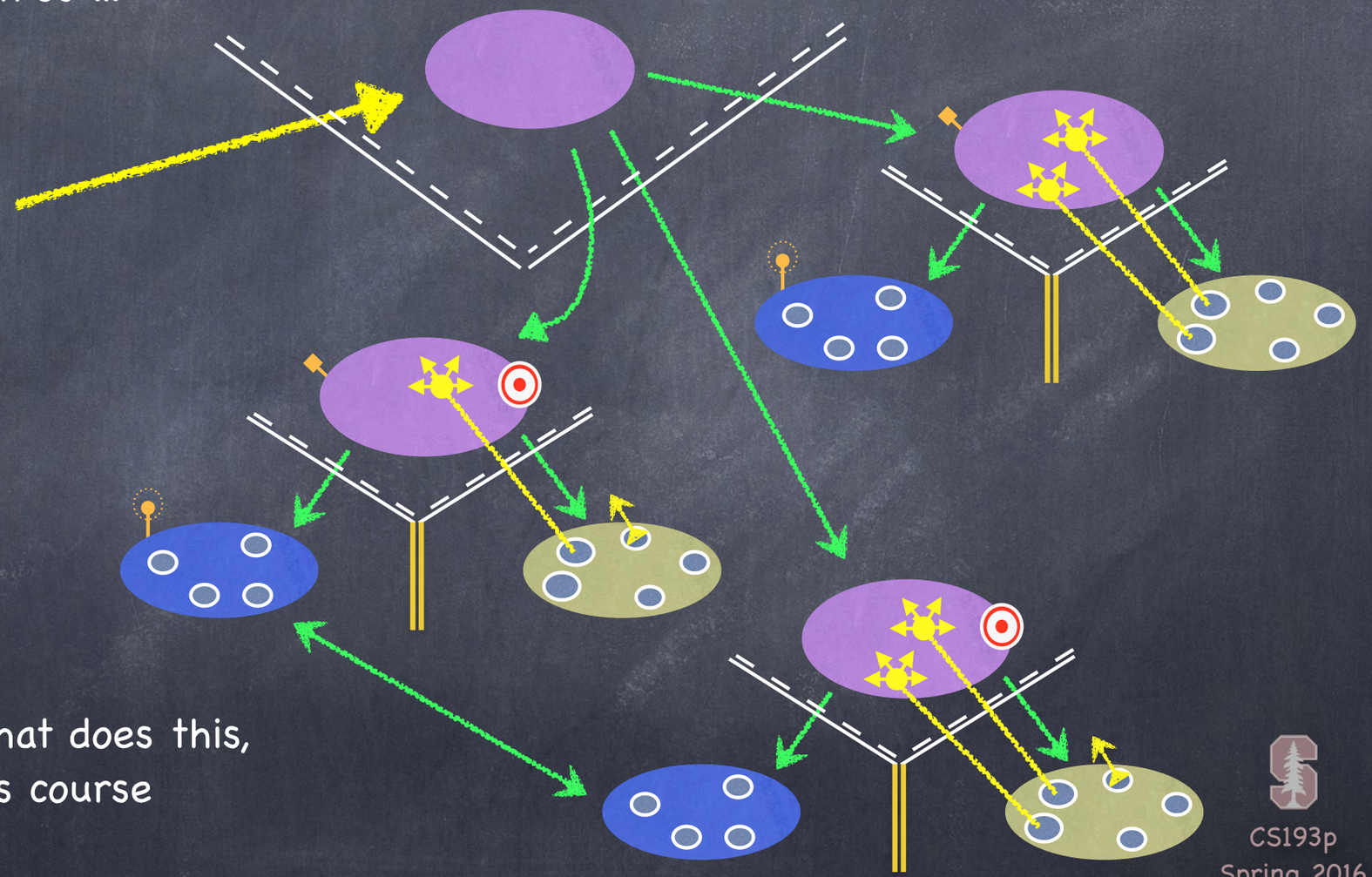
CS193p
Spring 2016

Multiple MVCs

- Time to build more powerful applications

  To do this, we must combine MVCs …

iOS provides some Controllers
whose View is "other MVCs" *



* you could build your own Controller that does this,
but we're not going to cover that in this course

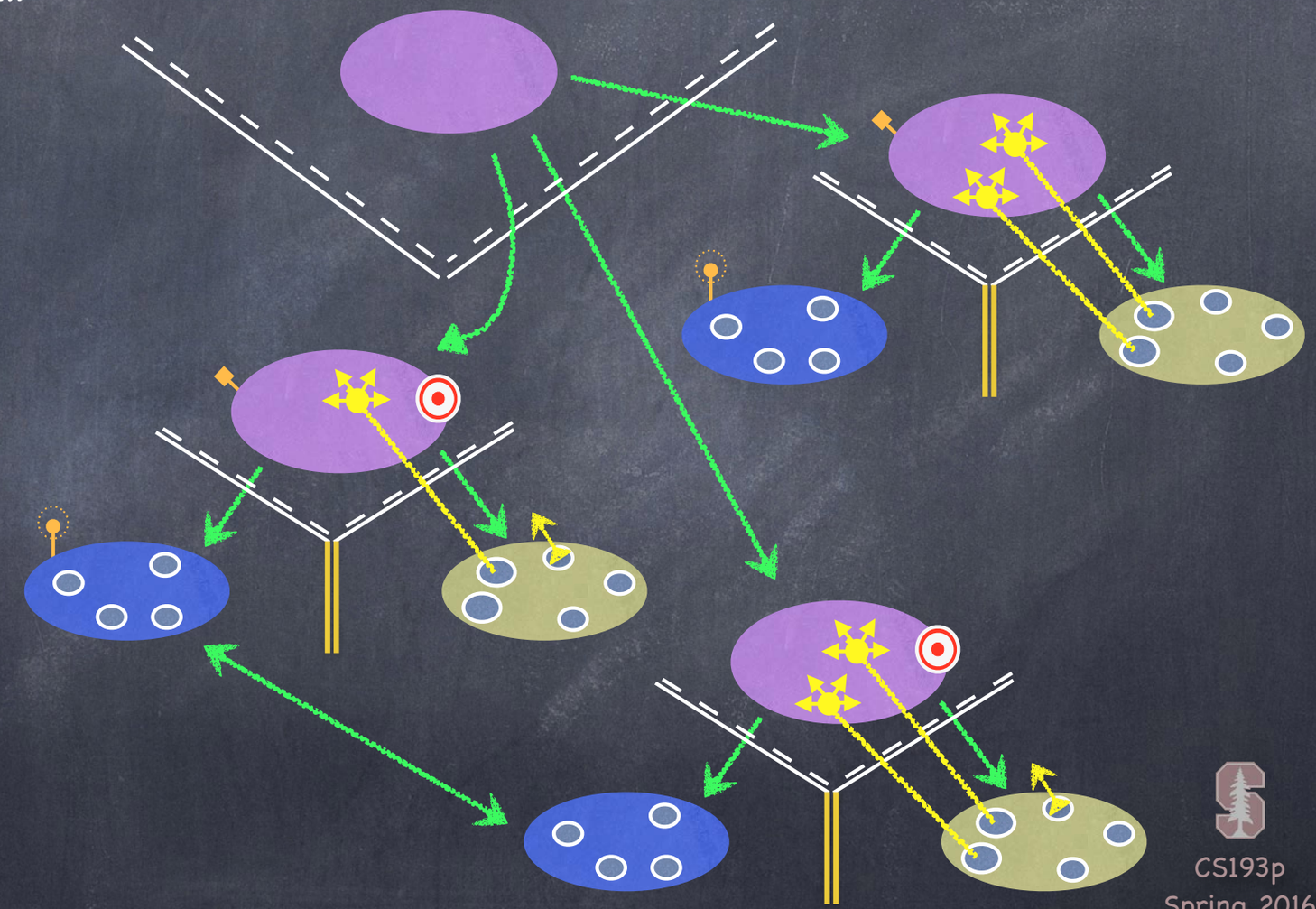
CS193p
Spring 2016

Multiple MVCs

- Time to build more powerful applications

  To do this, we must combine MVCs ...

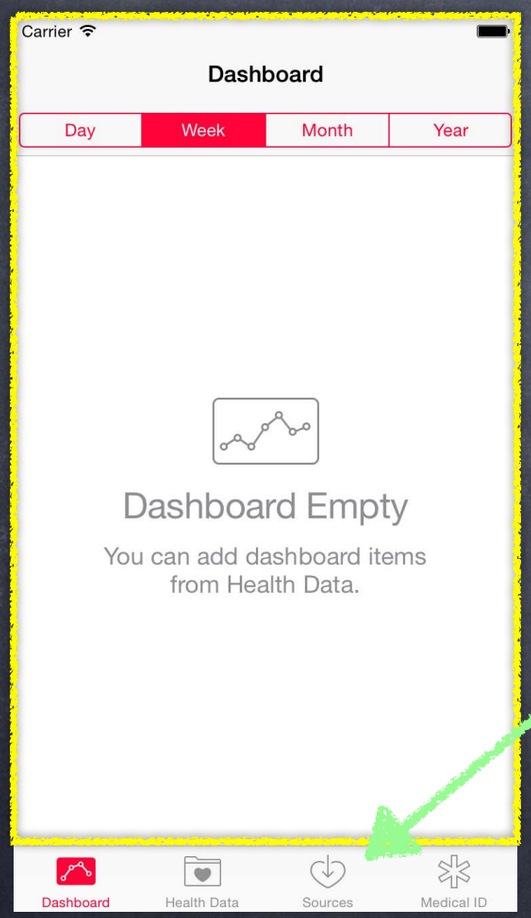
iOS provides some Controllers whose View is "other MVCs"
Examples:
UITabBarController
UISplitViewController
UINavigationController



CS193p
Spring 2016

UITabBarController

- It lets the user choose between different MVCs ...



A "Dashboard" MVC

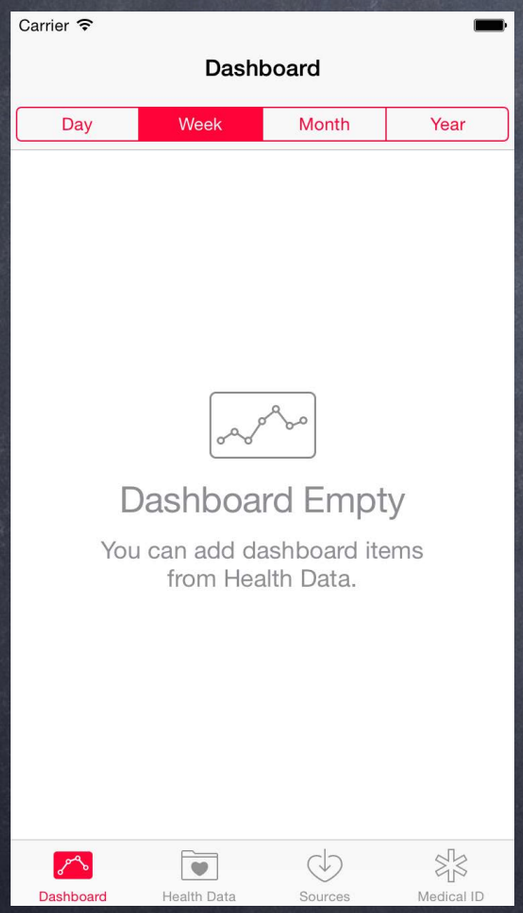
The icon, title and even a "badge value" on these is determined by the MVCs themselves via their property:

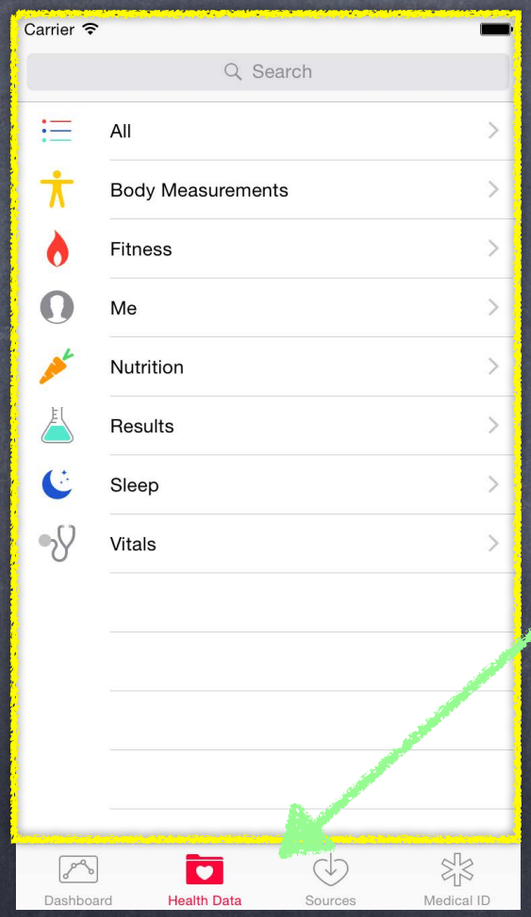
```
var tabBarItem: UITabBarItem!
```

But usually you just set them in your storyboard.

UITabBarController

- It lets the user choose between different MVCs ...





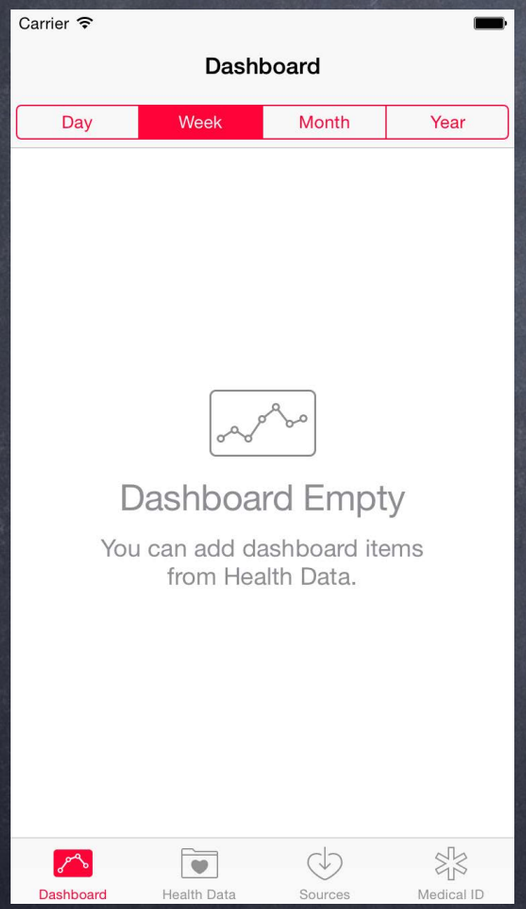
A "Health Data" MVC

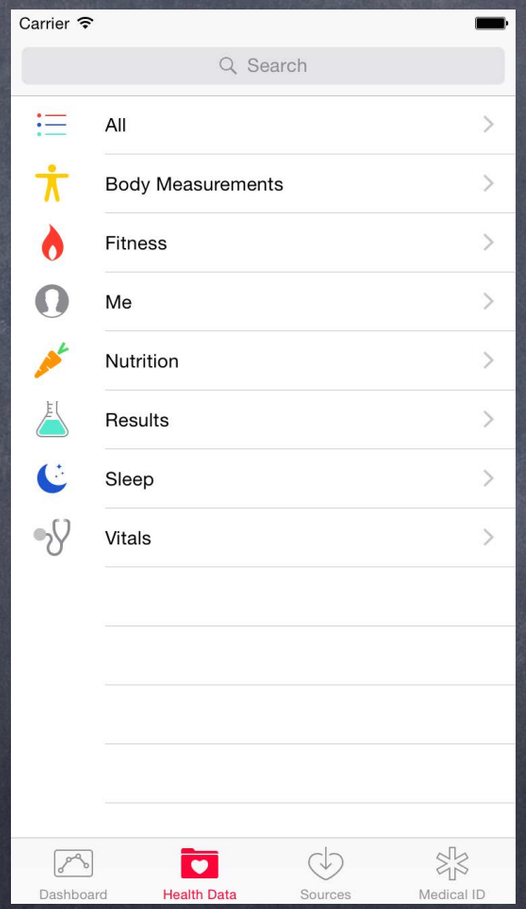
If there are too many tabs to fit here, the UITabBarController will automatically present a UI for the user to manage the overflow!

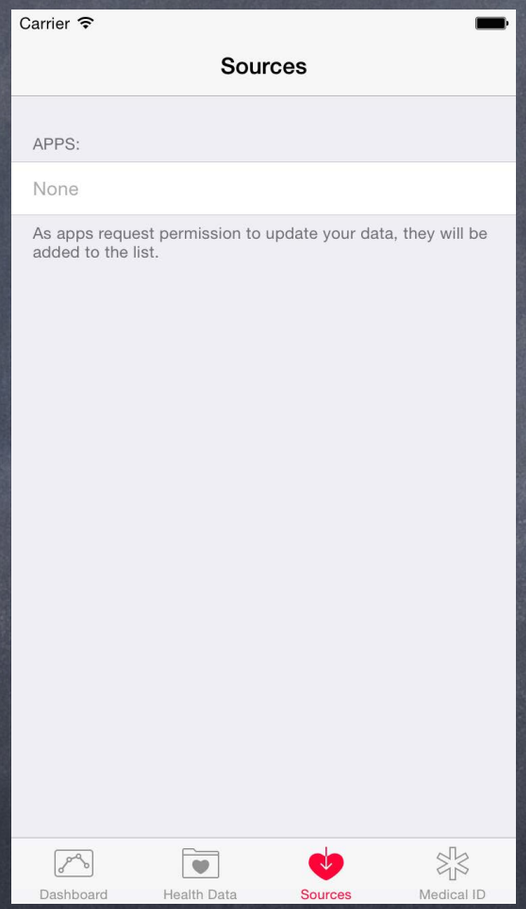
CS193p
Spring 2016

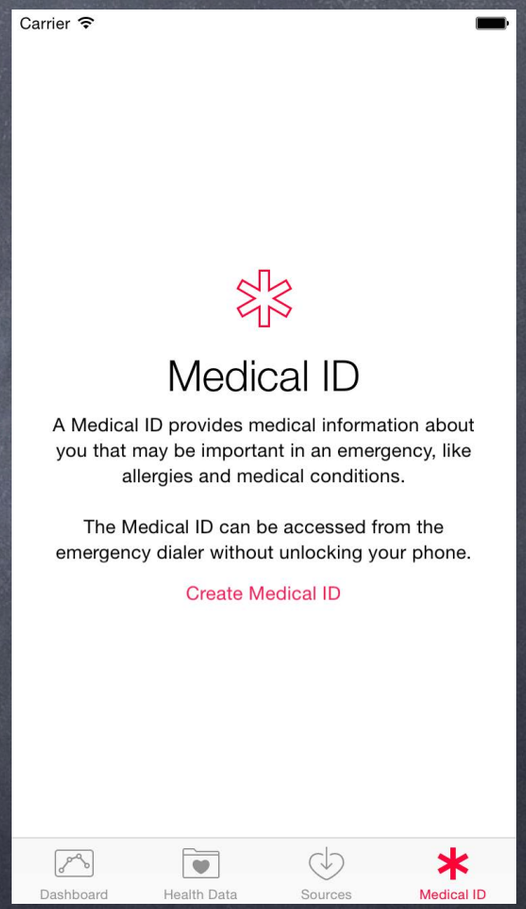
UITabBarController

- It lets the user choose between different MVCs ...









CS193p
Spring 2016

{ Coding Session }

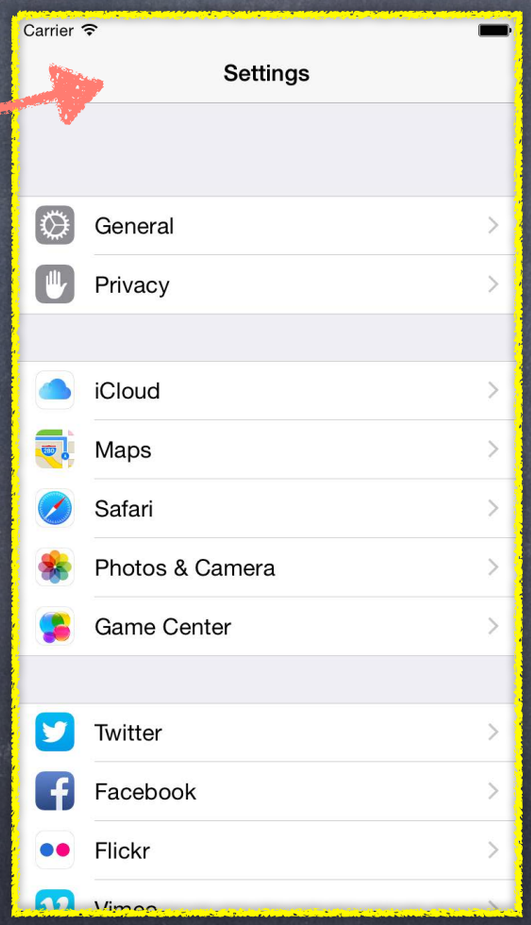
UINavigationController

- Pushes and pops MVCs off of a stack (like a stack of cards) …

This top area is drawn by the `UINavigationController`

But the contents of the top area (like the title or any buttons on the right) are determined by the MVC currently showing (in this case, the "All Settings" MVC)

Each MVC communicates these contents via its `UIViewController`'s `navigationItem` property

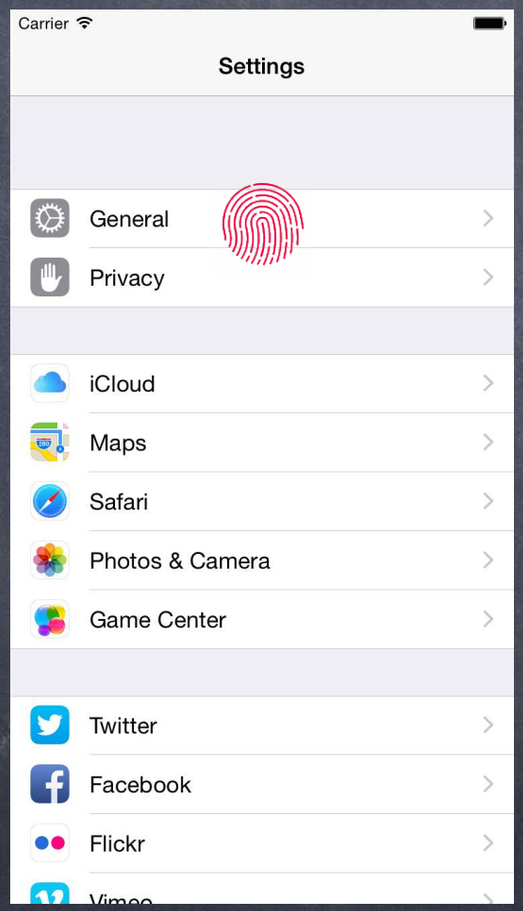


An "All Settings" MVC

CS193p
Spring 2016

UINavigationController

- Pushes and pops MVCs off of a stack (like a stack of cards) ...



CS193p
Spring 2016

UINavigationController

- Pushes and pops MVCs off of a stack (like a stack of cards) …

A "General Settings" MVC

It's possible to add MVC-specific buttons here too via the `UIViewController`'s `toolbarItems` property

CS193p
Spring 2016

UINavigationController

- Pushes and pops MVCs off of a stack (like a stack of cards) ...

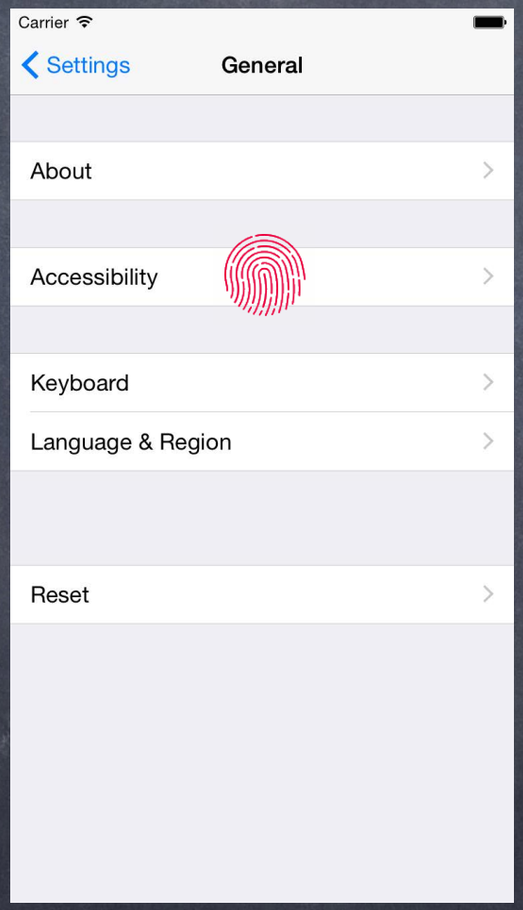
Notice this "back" button has appeared. This is placed here automatically by the `UINavigationController`.

A "General Settings" MVC

CS193p
Spring 2016

UINavigationController

- Pushes and pops MVCs off of a stack (like a stack of cards) …



CS193p
Spring 2016

UINavigationController

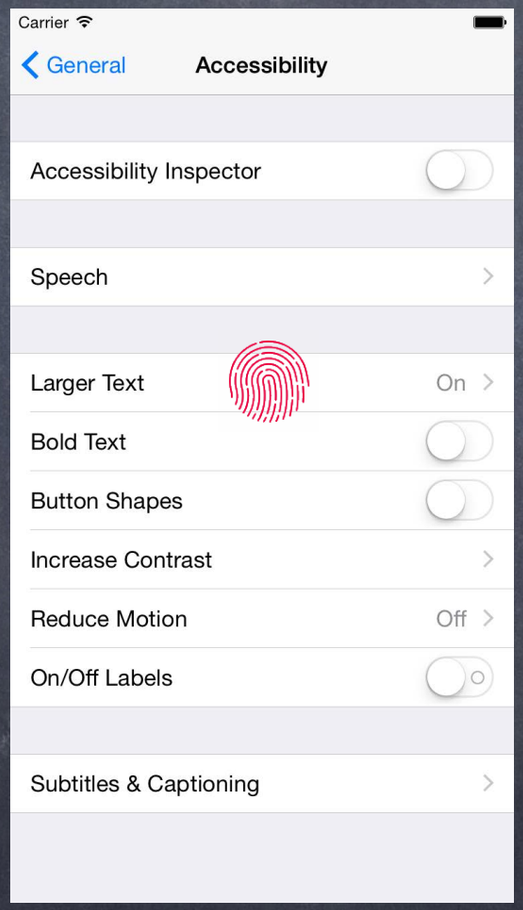
- Pushes and pops MVCs off of a stack (like a stack of cards) ...

An "Accessibility" MVC

CS193p
Spring 2016

UINavigationController

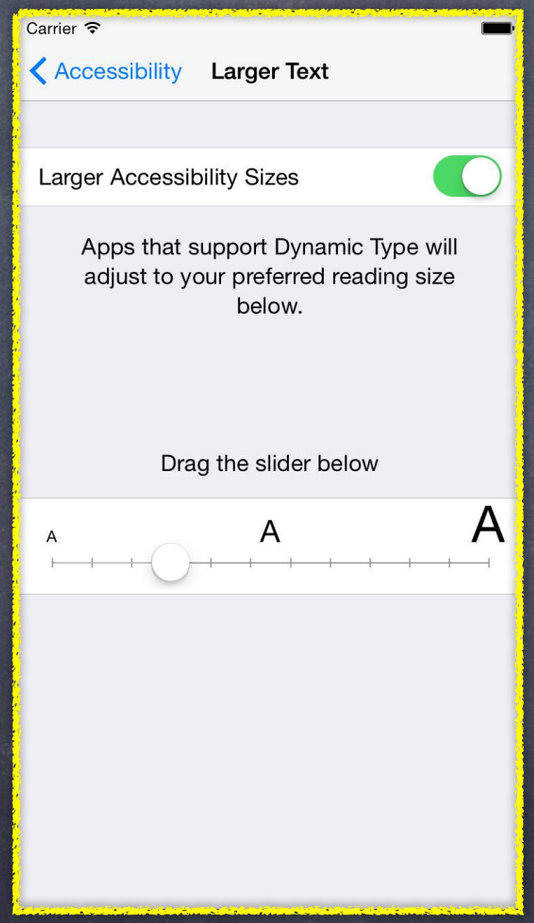
- Pushes and pops MVCs off of a stack (like a stack of cards) …



CS193p
Spring 2016

UINavigationController

- Pushes and pops MVCs off of a stack (like a stack of cards) ...

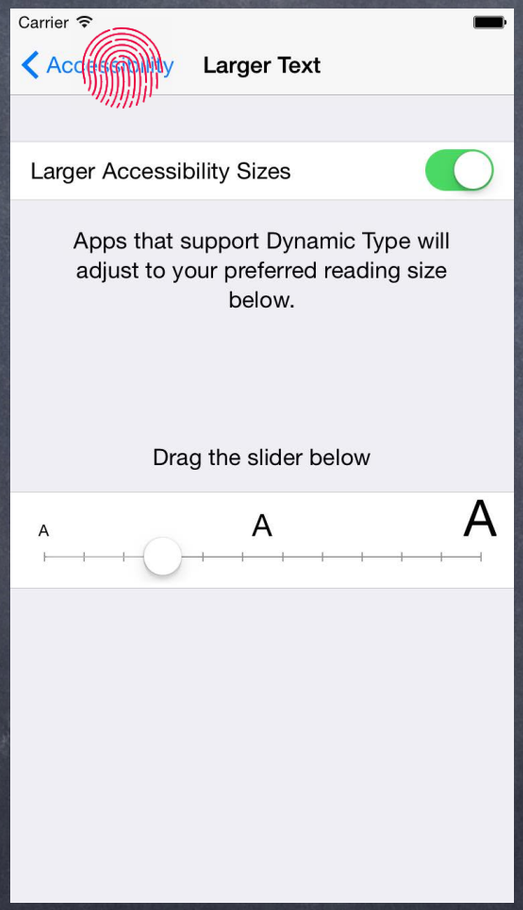


A "Larger Text" MVC

CS193p
Spring 2016

UINavigationController

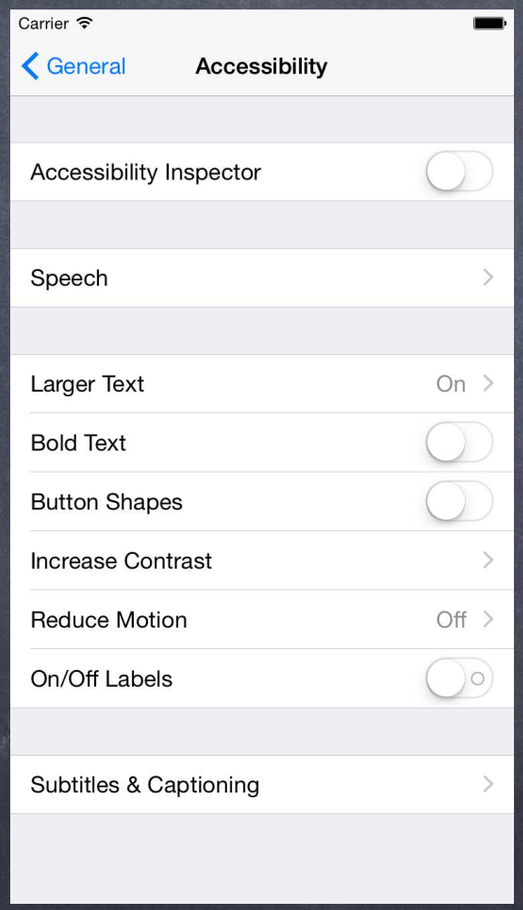
- Pushes and pops MVCs off of a stack (like a stack of cards) …



CS193p
Spring 2016

UINavigationController

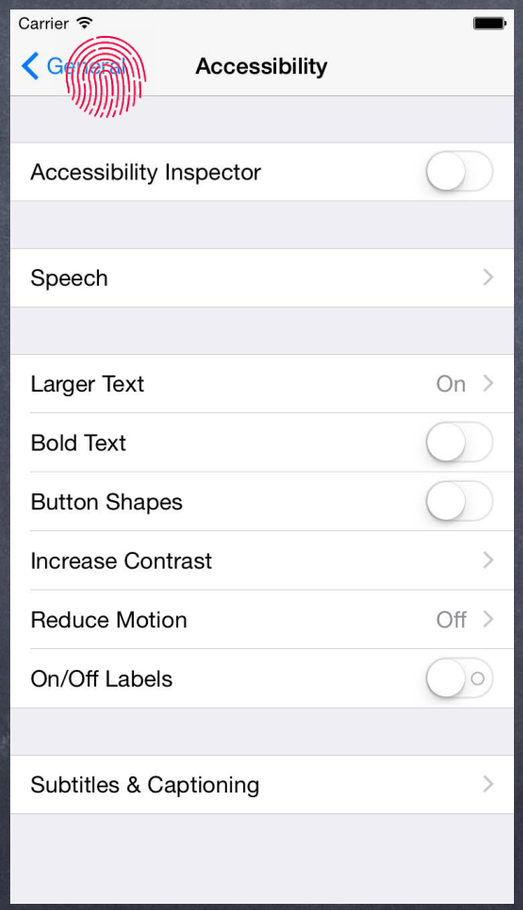
- Pushes and pops MVCs off of a stack (like a stack of cards) ...



CS193p
Spring 2016

UINavigationController

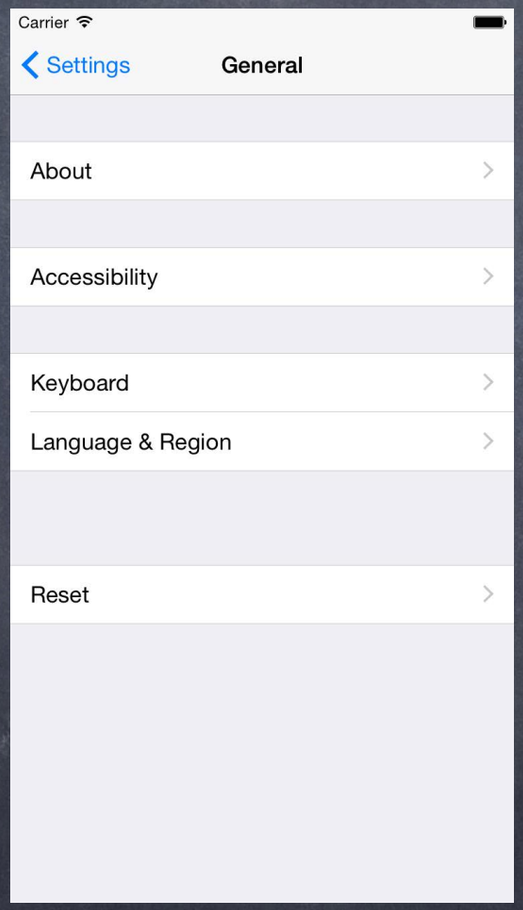
- Pushes and pops MVCs off of a stack (like a stack of cards) ...



CS193p
Spring 2016

UINavigationController

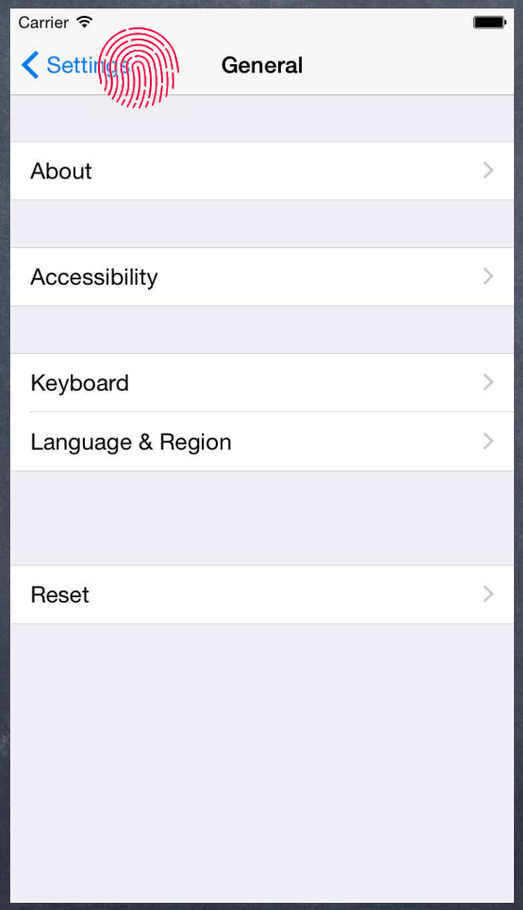
- Pushes and pops MVCs off of a stack (like a stack of cards) ...



CS193p
Spring 2016

UINavigationController

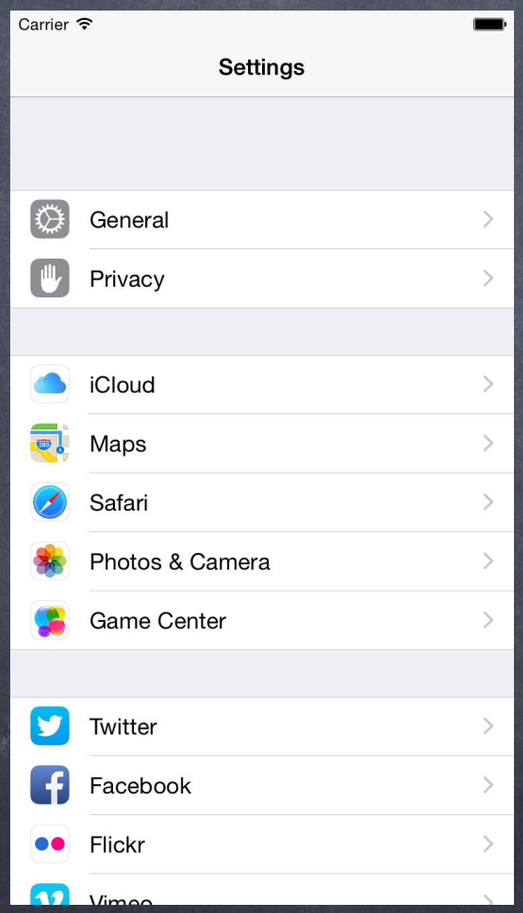
- Pushes and pops MVCs off of a stack (like a stack of cards) …



CS193p
Spring 2016

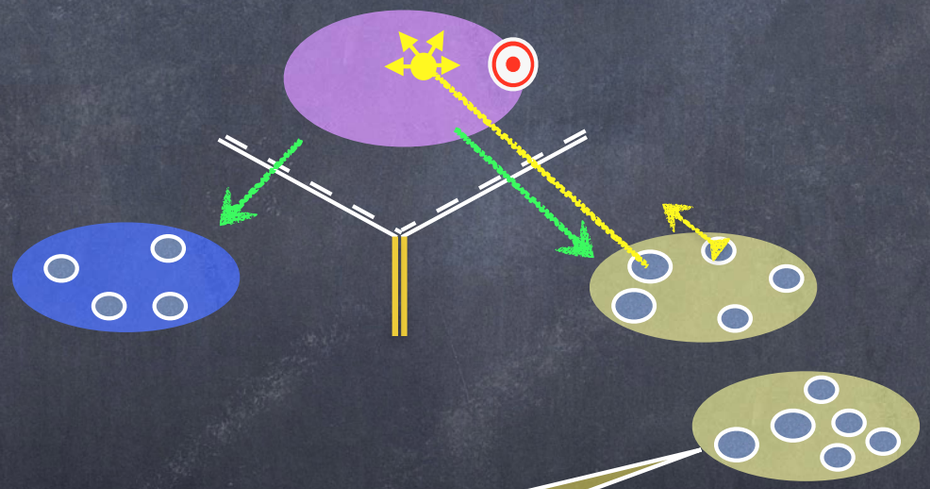
UINavigationController

- Pushes and pops MVCs off of a stack (like a stack of cards) …



CS193p
Spring 2016

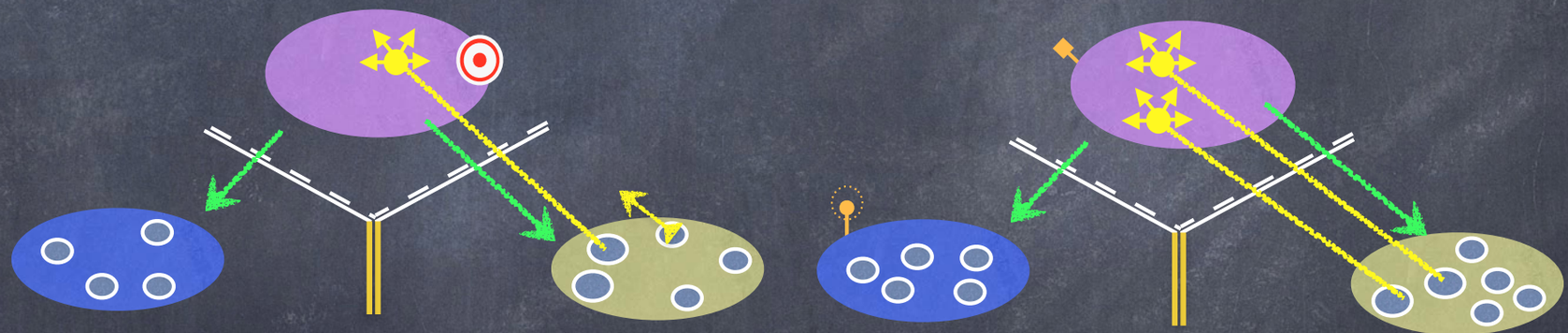
UINavigationController



I want more features, but it doesn't make sense to put them all in one MVC!

CS193p
Spring 2016

UINavigationController

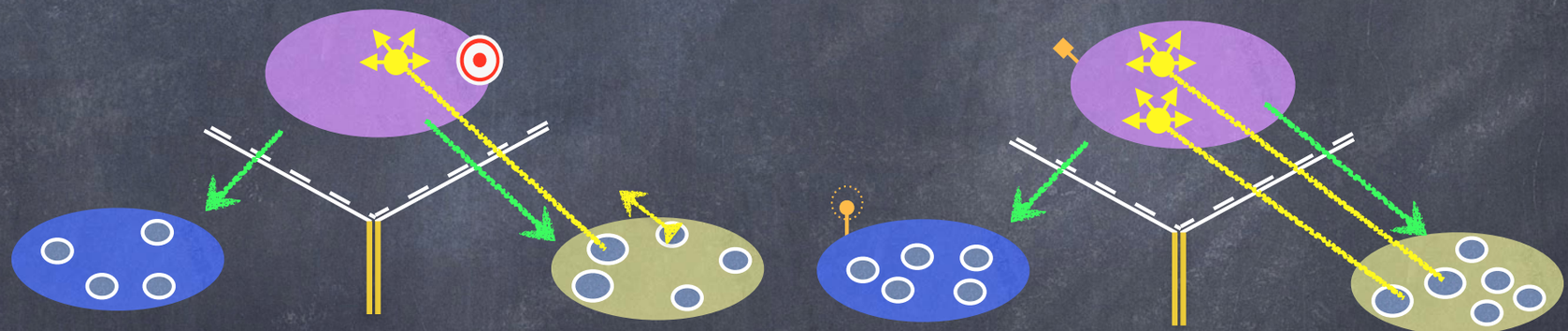


So I create a new MVC to encapsulate that functionality.

CS193p
Spring 2016

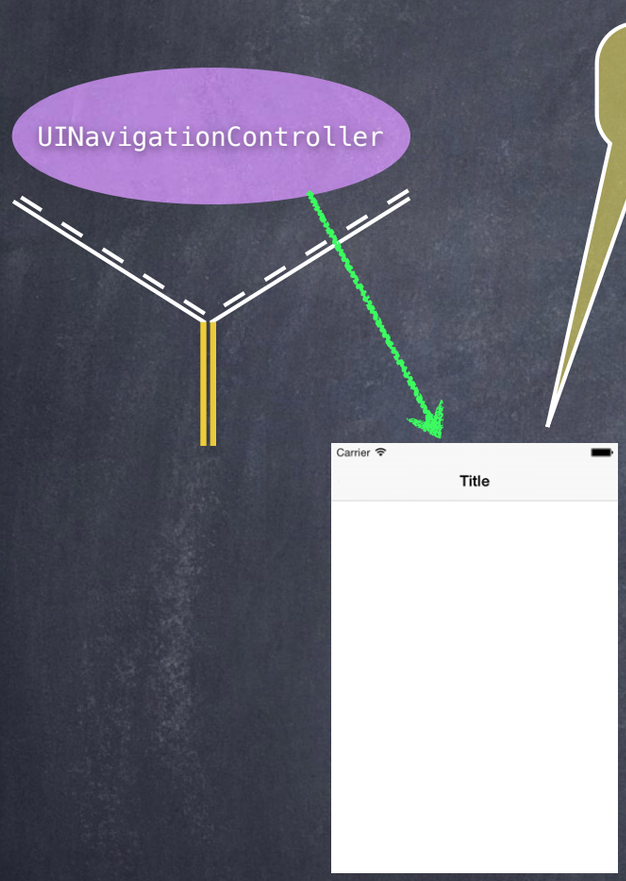
UINavigationController

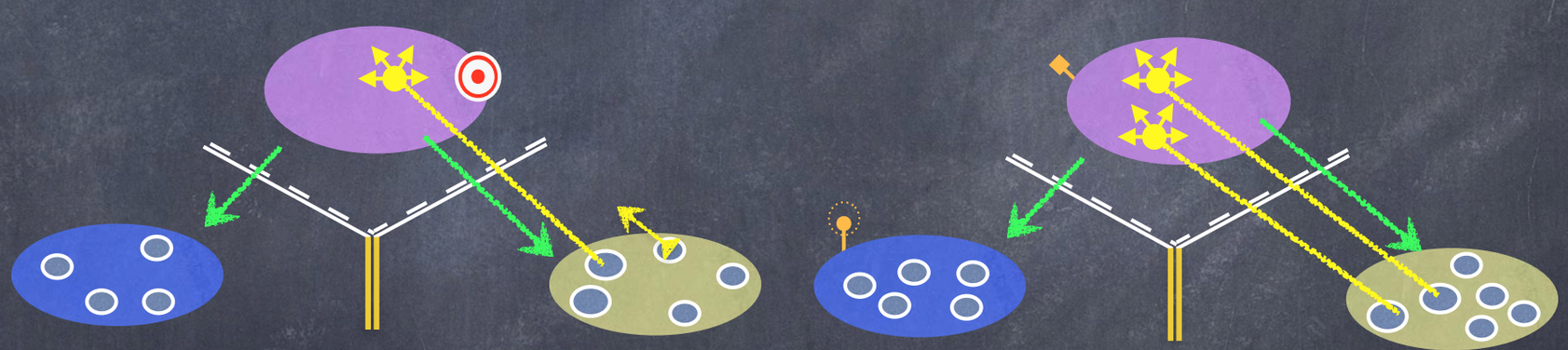
We can use a UINavigationController to let them share the screen.



CS193p
Spring 2016

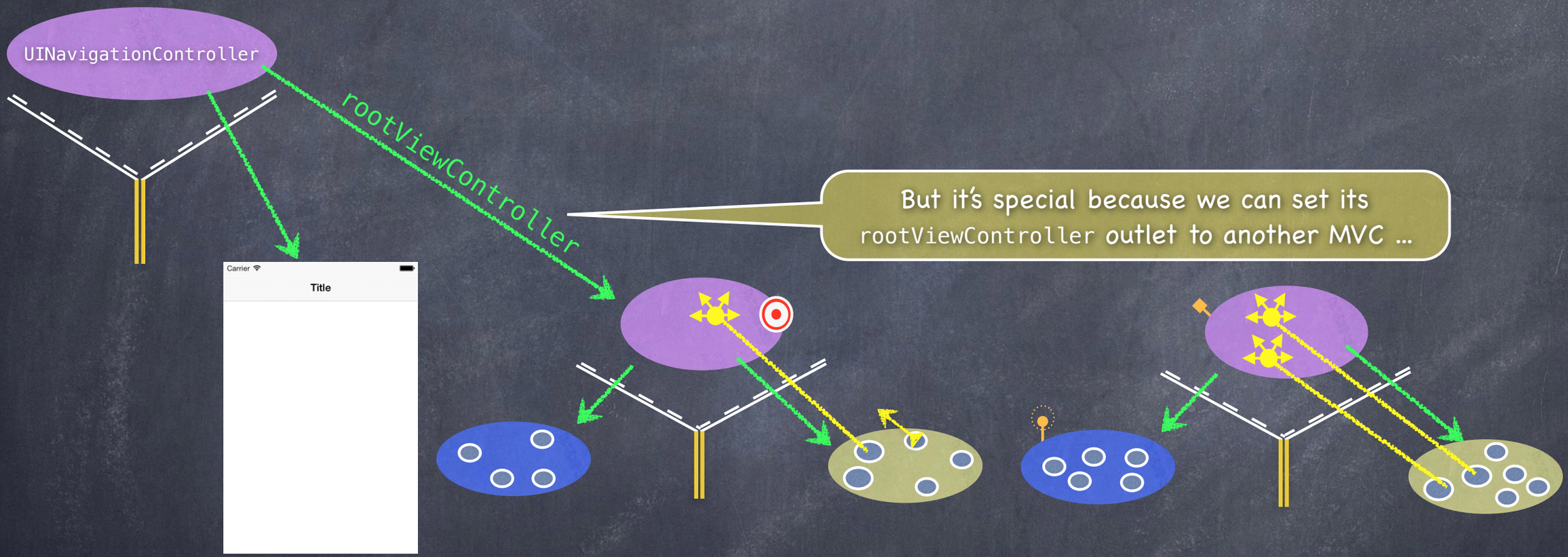
UINavigationController



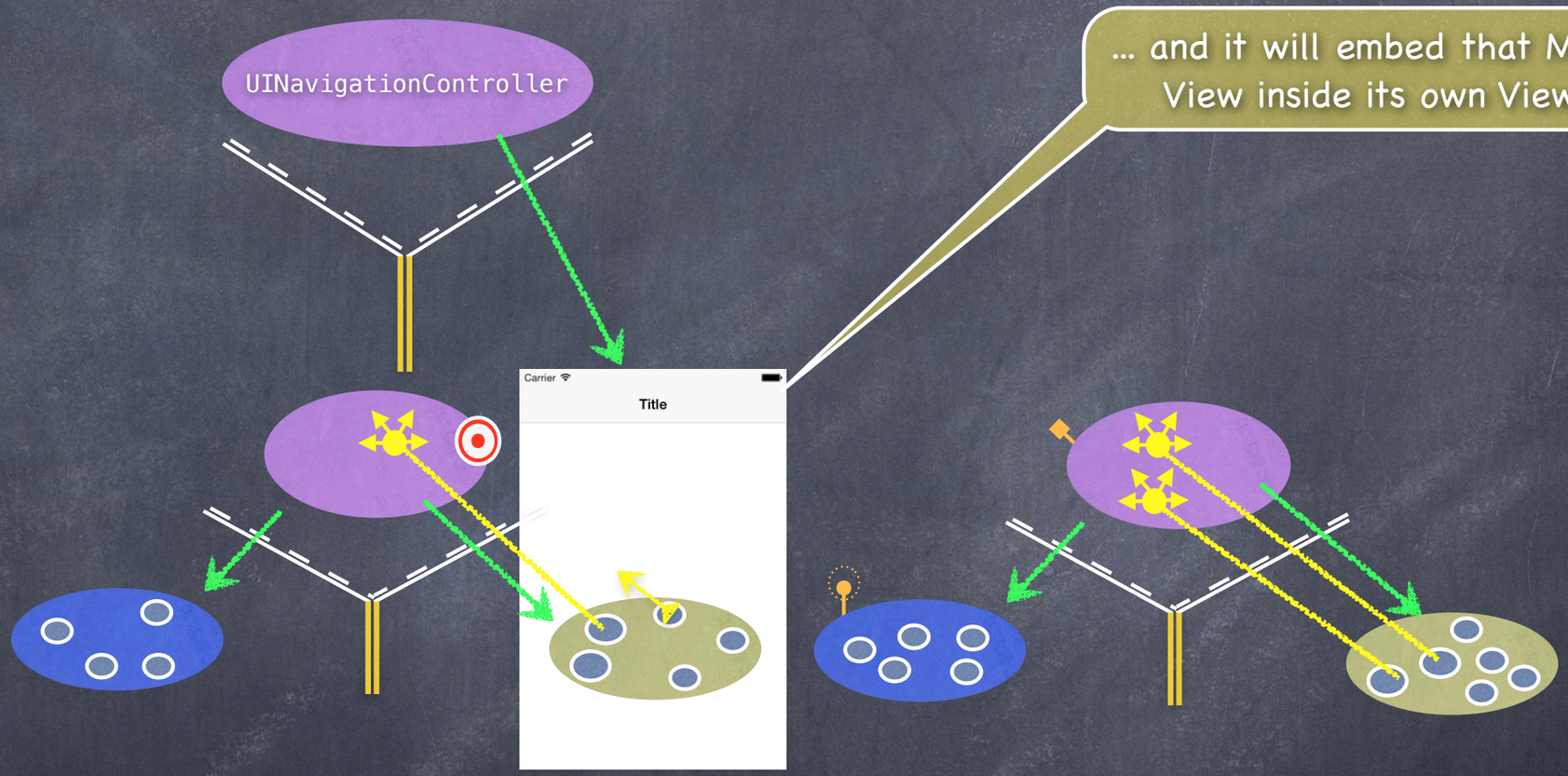


CS193p
Spring 2016

UINavigationController

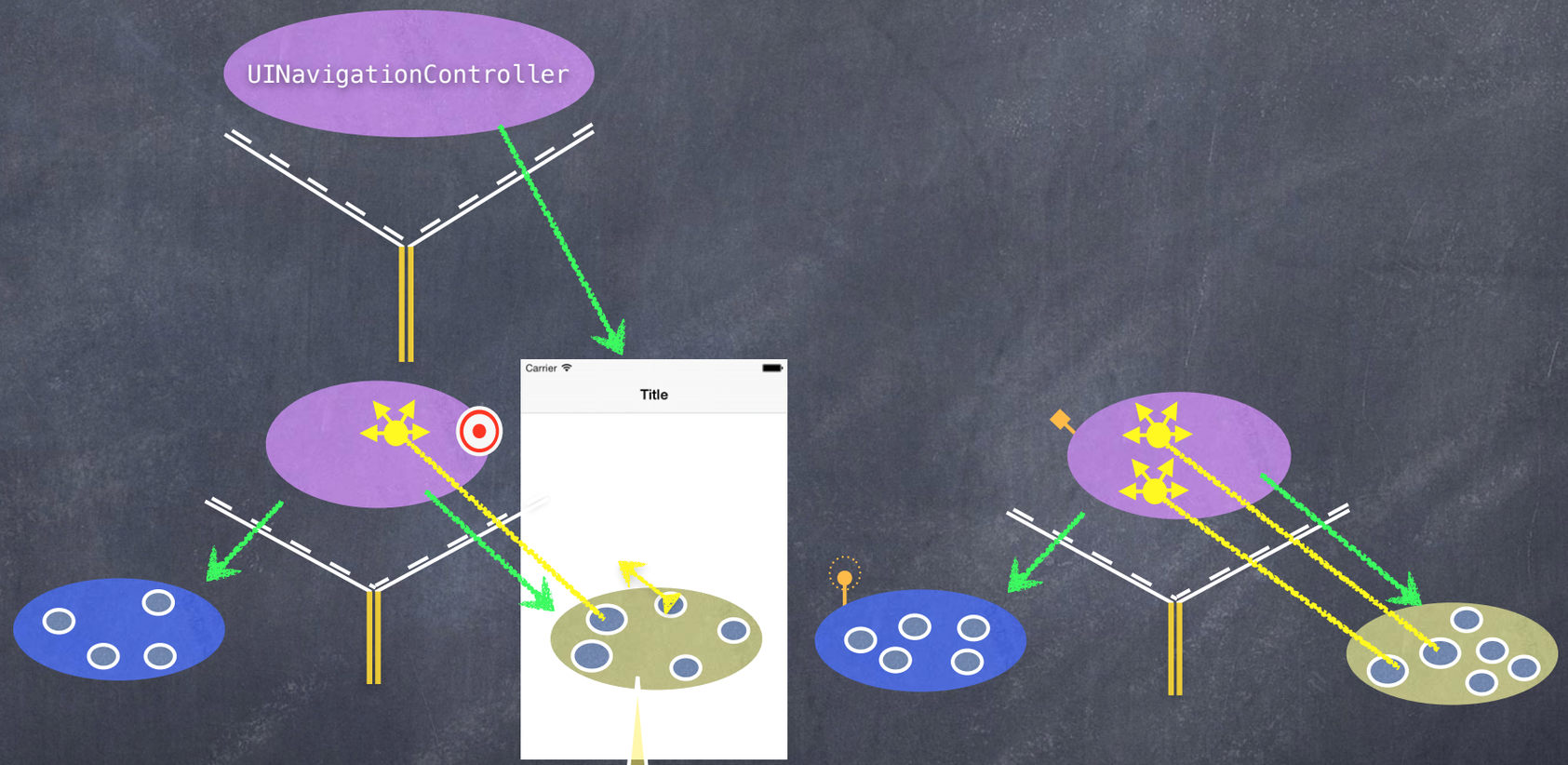


UINavigationController



CS193p
Spring 2016

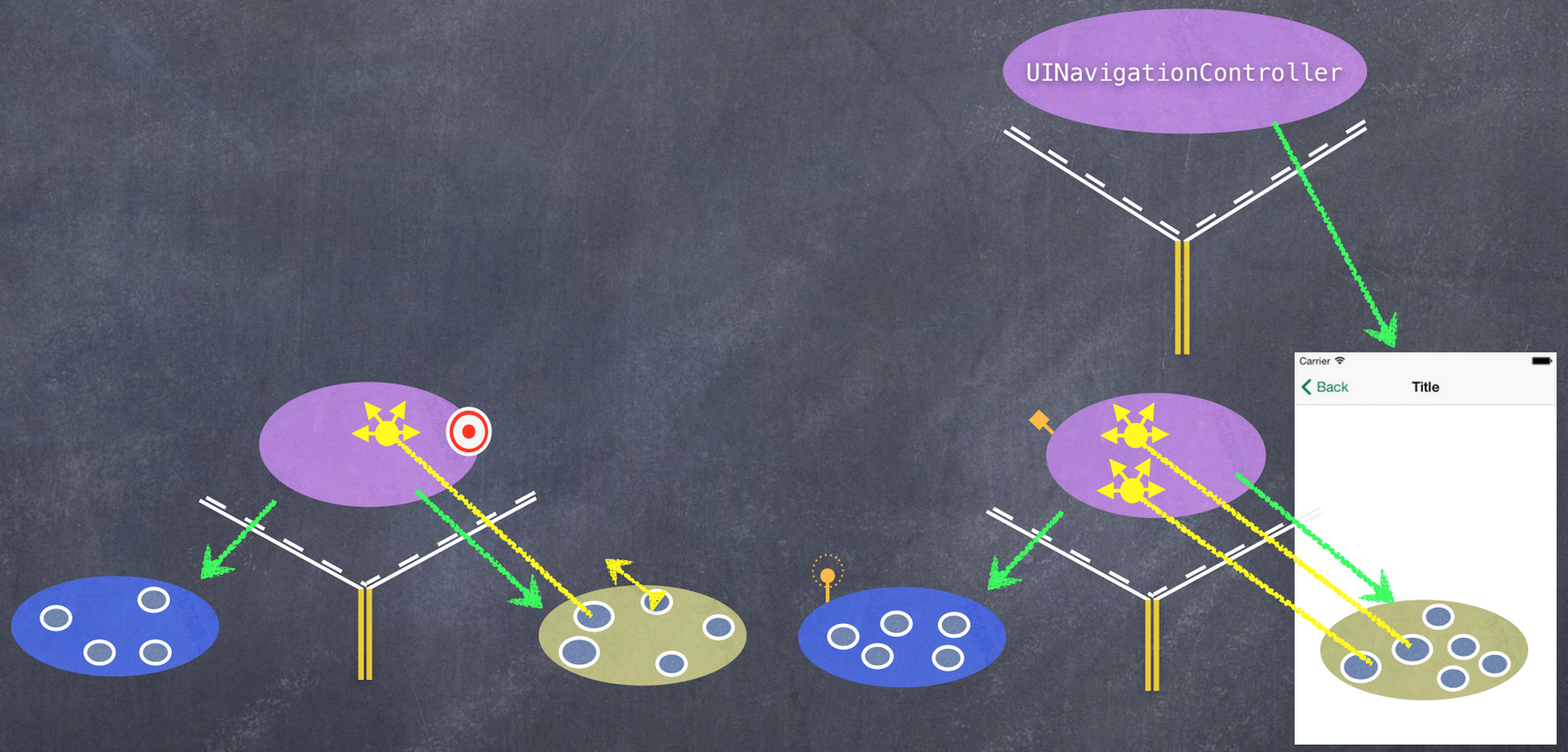
UINavigationController



Then a UI element in this View (e.g. a `UIButton`) can segue to the other MVC and its View will now appear in the `UINavigationController` instead.

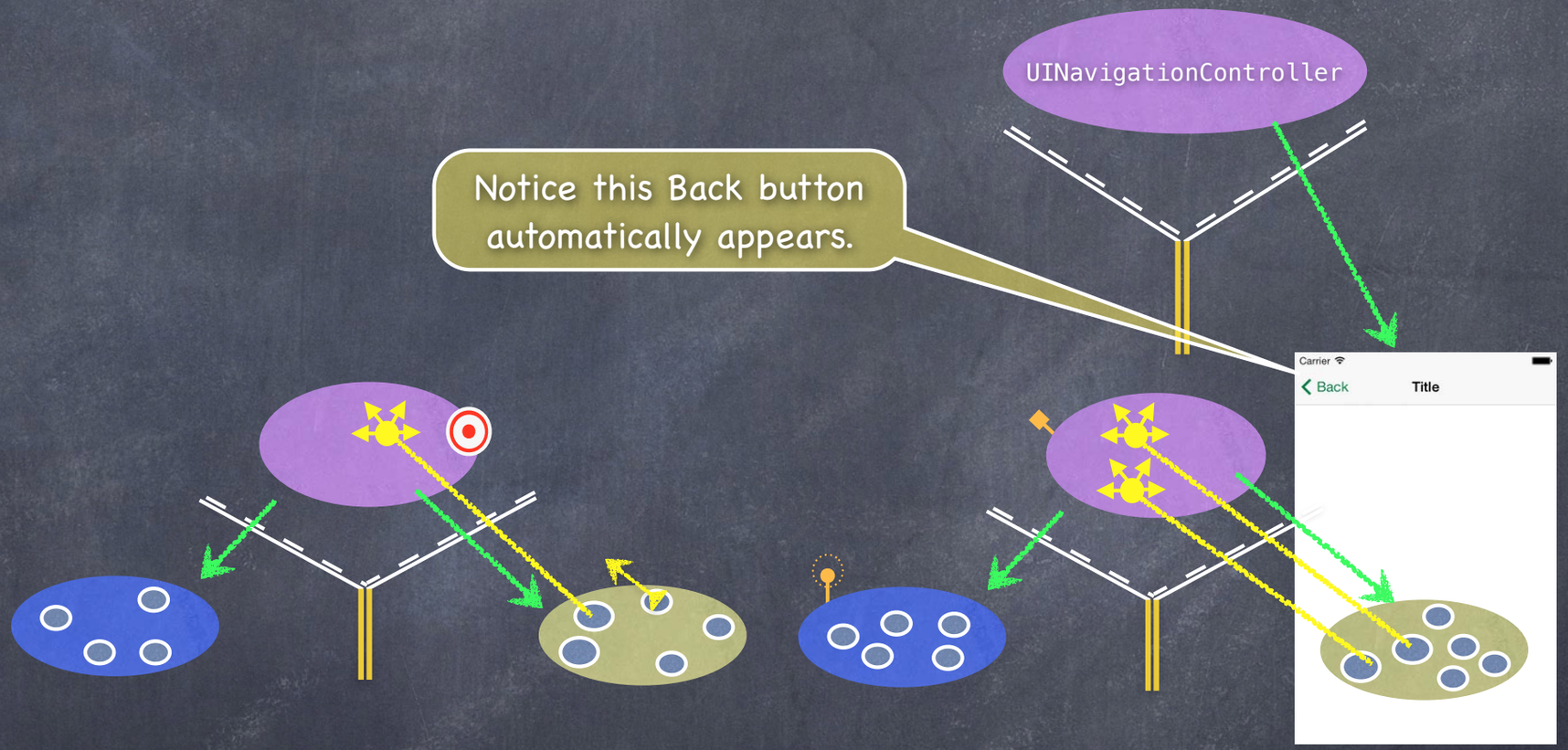
CS193p
Spring 2016

UINavigationController



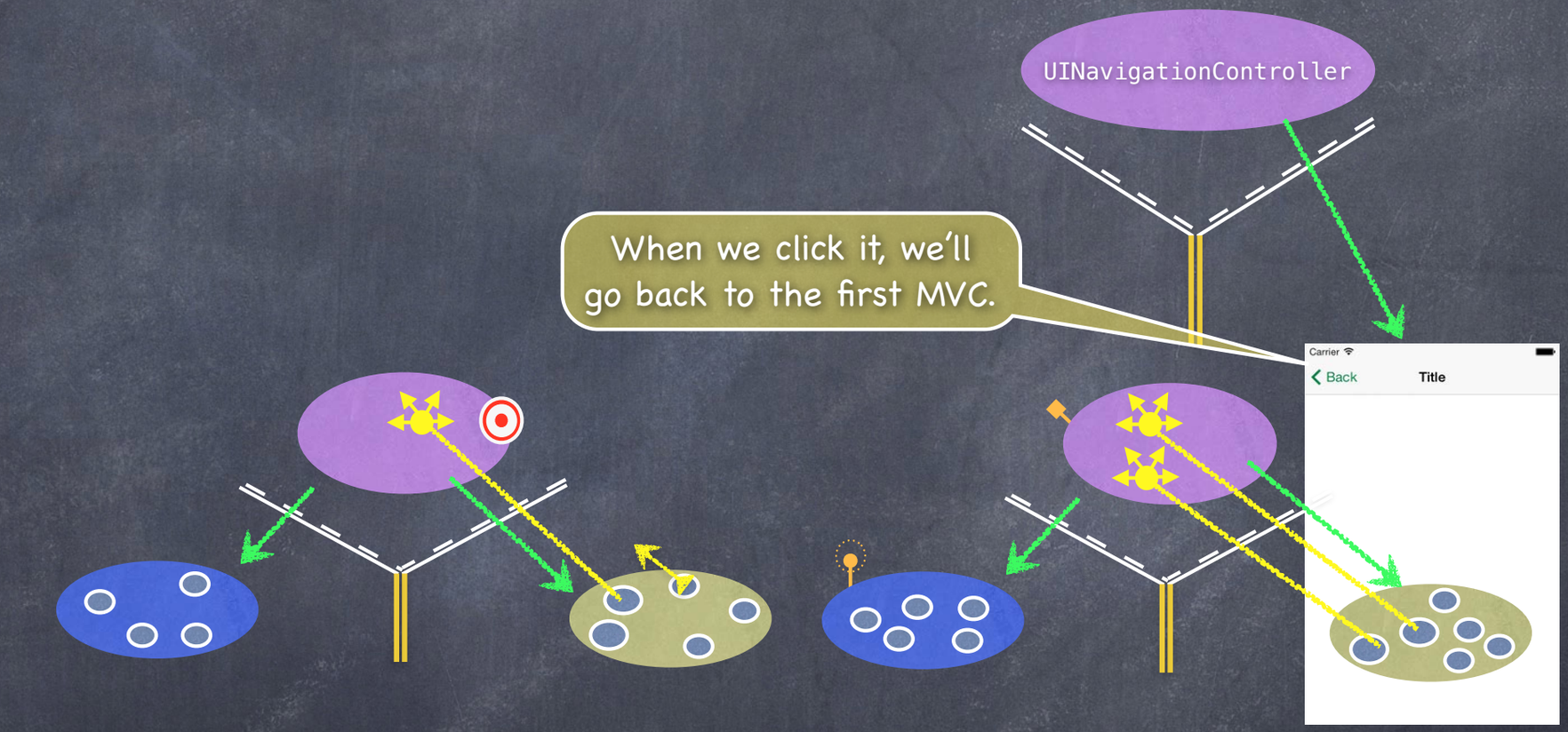
We call this kind of segue a "Show (push) segue".

UINavigationController



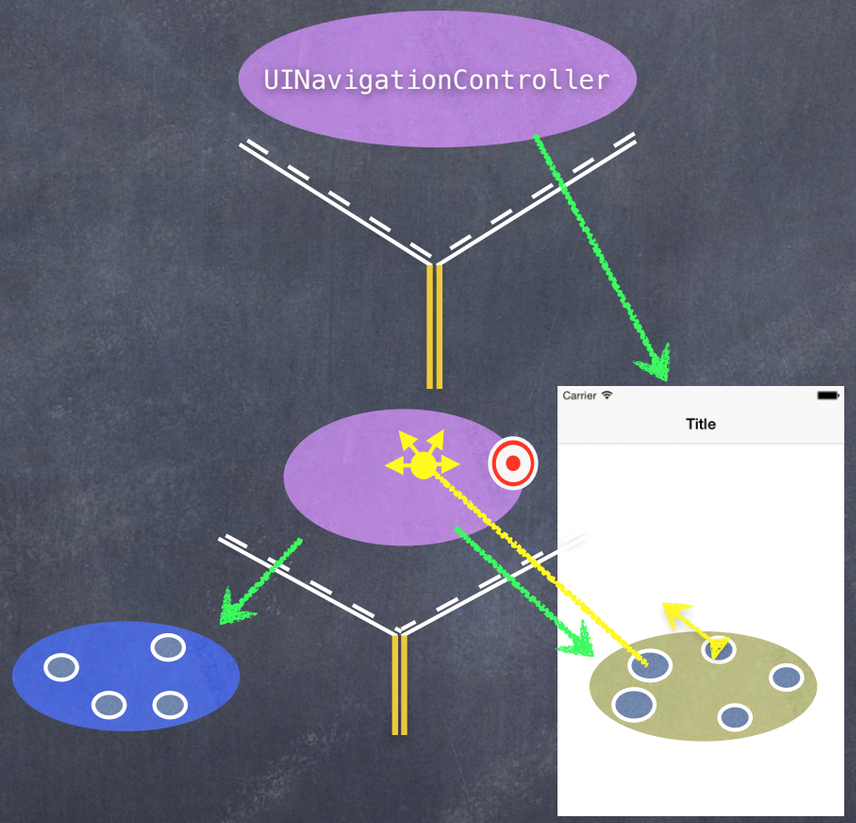
CS193p
Spring 2016

UINavigationController



CS193p
Spring 2016

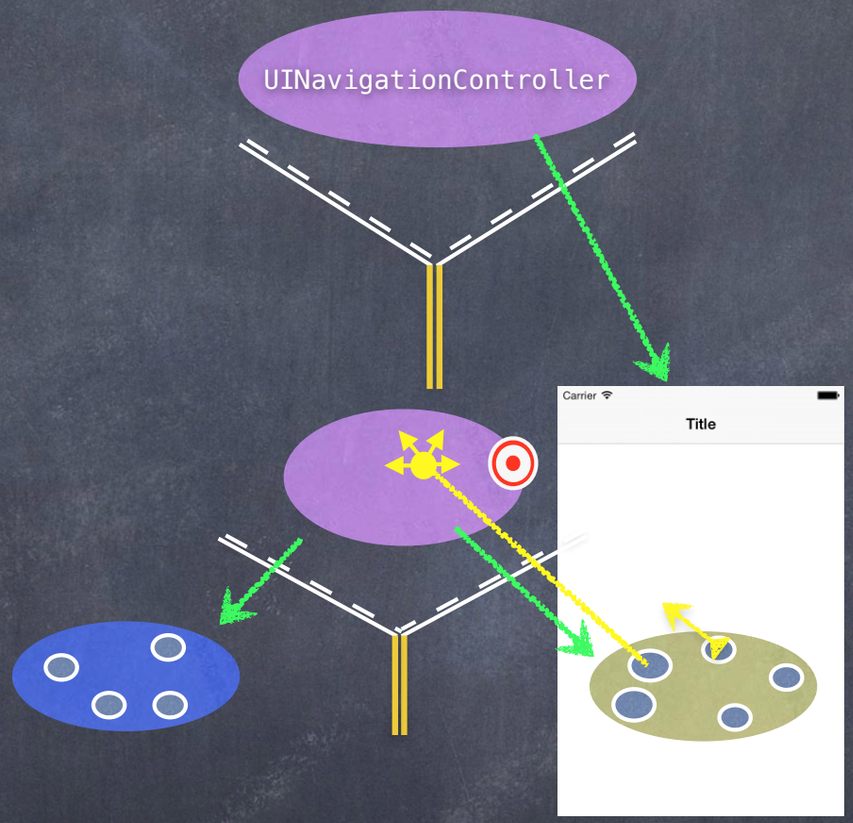
UINavigationController



Notice that after we back out of an MVC,
it disappears (it is deallocated from the heap, in fact).

CS193p
Spring 2016

UINavigationController



CS193p
Spring 2016

Accessing the sub-MVCs

- You can get the sub-MVCs via the `viewControllers` property

```
var viewControllers: [UIViewController]? { get set } // can be optional (e.g. for tab bar)
// for a tab bar, they are in order, left to right, in the array
// for a split view, [0] is the master and [1] is the detail
// for a navigation controller, [0] is the root and the rest are in order on the stack
// even though this is settable, usually setting happens via storyboard, segues, or other
// for example, navigation controller's push and pop methods
```

- But how do you get ahold of the SVC, TBC or NC itself?

Every `UIViewController` knows the Split View, Tab Bar or Navigation Controller it is currently in

These are `UIViewController` properties …

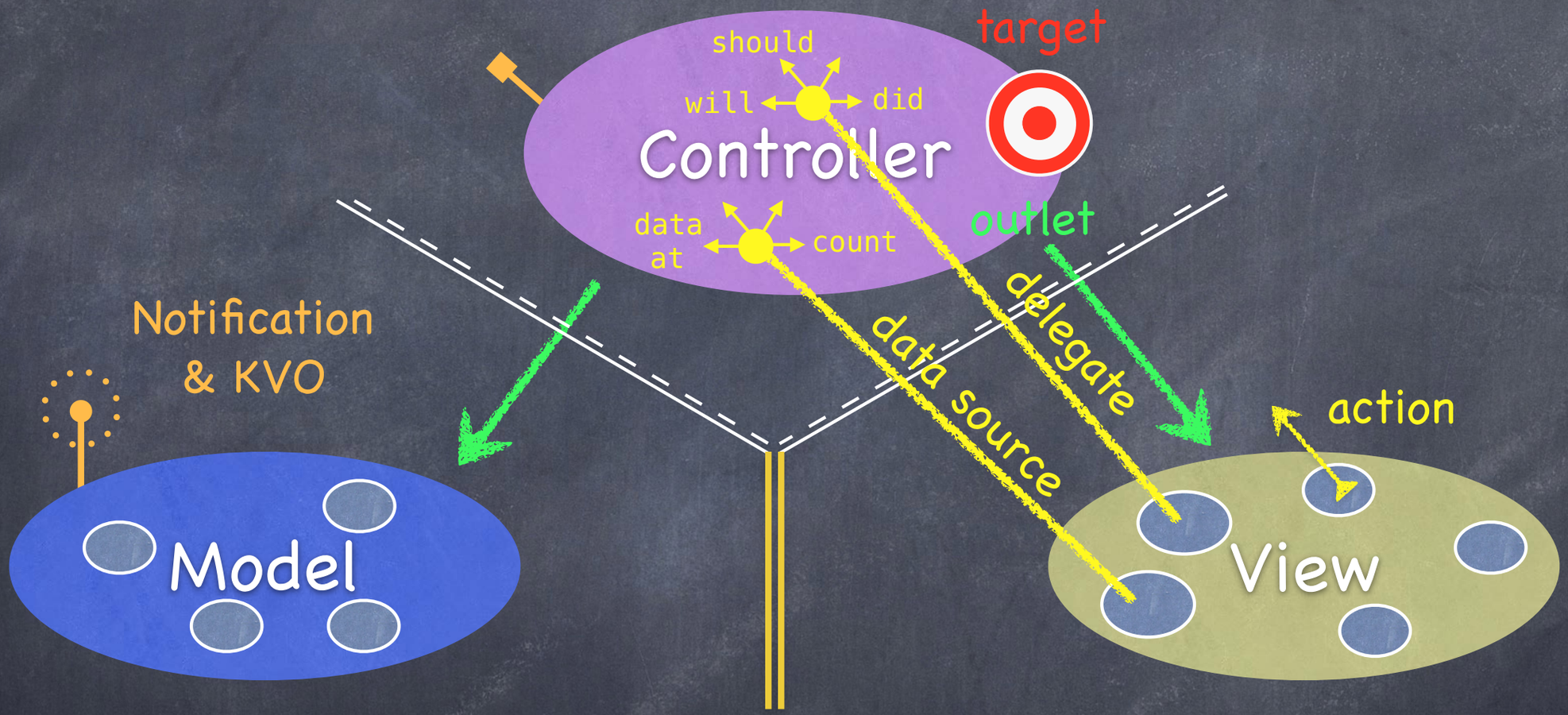
```
var tabBarController: UITabBarController? { get }
var splitViewController: UISplitViewController? { get }
var navigationController: UINavigationController? { get }
```

So, for example, to get the detail of the split view controller you are in …

```
if let detailVC: UIViewController = splitViewController?.viewControllers[1] { … }
```

Notifications

MVC



Now combine MVC groups to make complicated programs ...

CS193p
Spring 2016

Notifications

- Main class is NSNotificationCenter
- Use it for sending notifications