Stanford CS193p

Developing Applications for iOS
Winter 2015

Today

- Multiple MVCs
  - Split View Controllers & Navigation Controllers & Tab Bar Controllers
  - Segues
  - Demo: Psychologist
  - Popovers (time permitting)

Multiple MVCs

- Time to build more powerful applications

  To do this, we must combine MVCs …

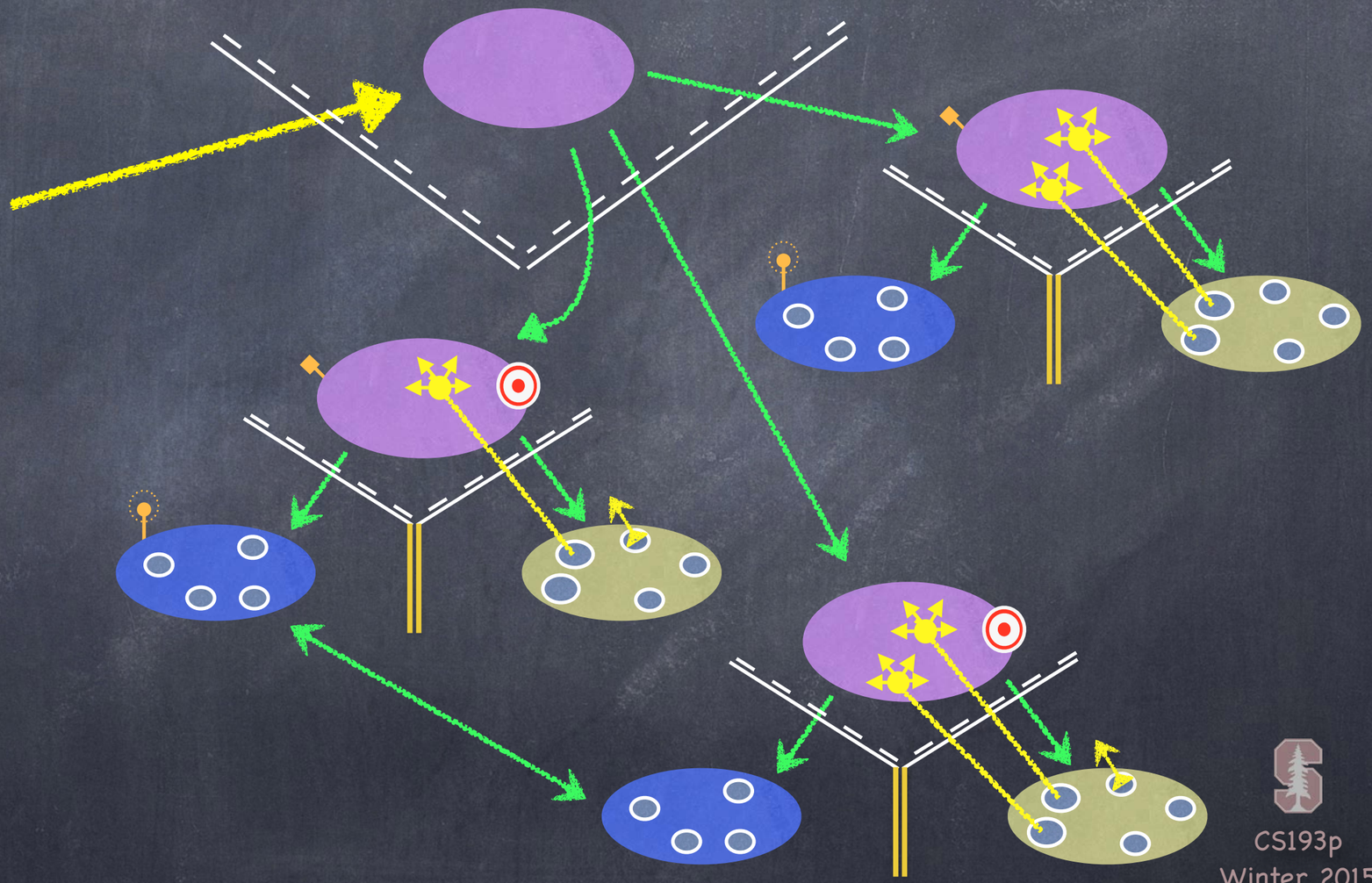
iOS provides some Controllers whose View is "other MVCs"

Examples:

`UITabBarController`

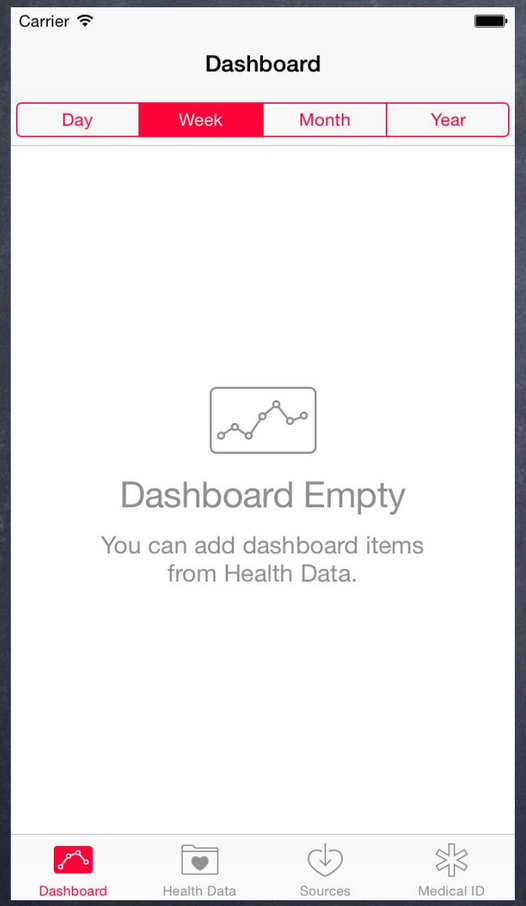
`UISplitViewController`

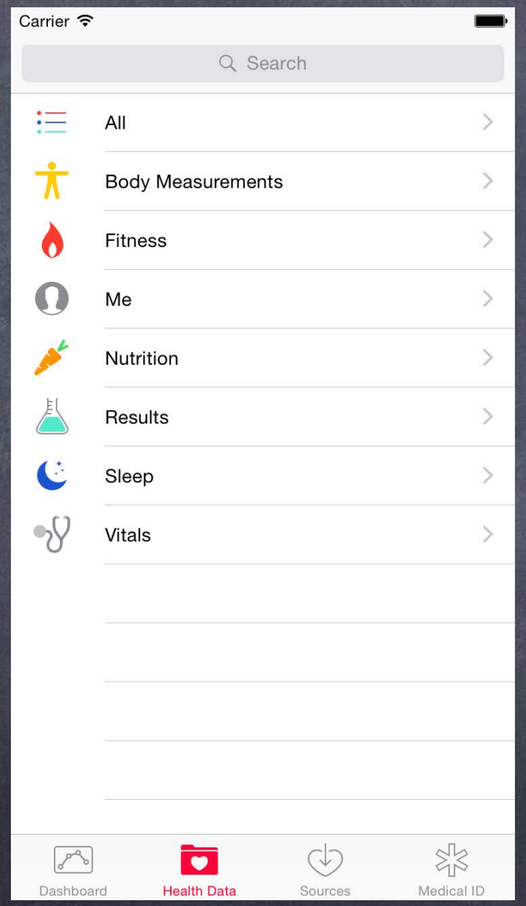
`UINavigationController`

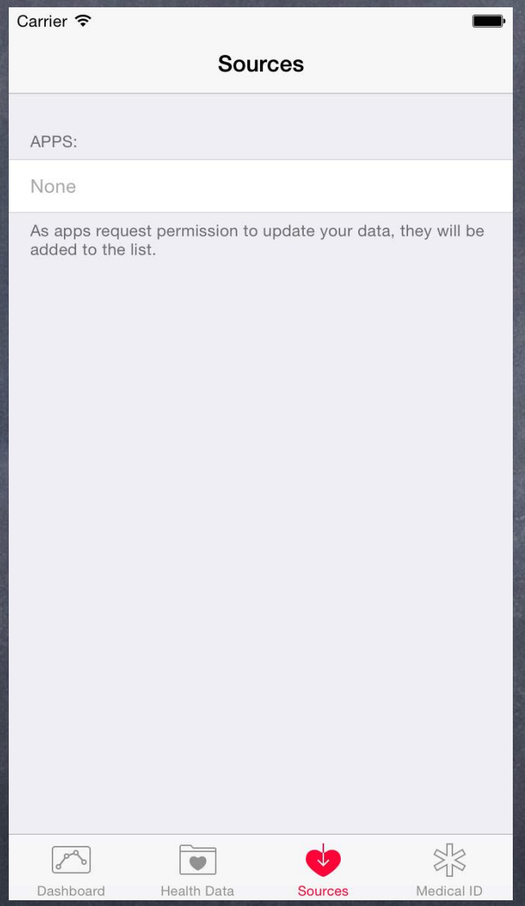


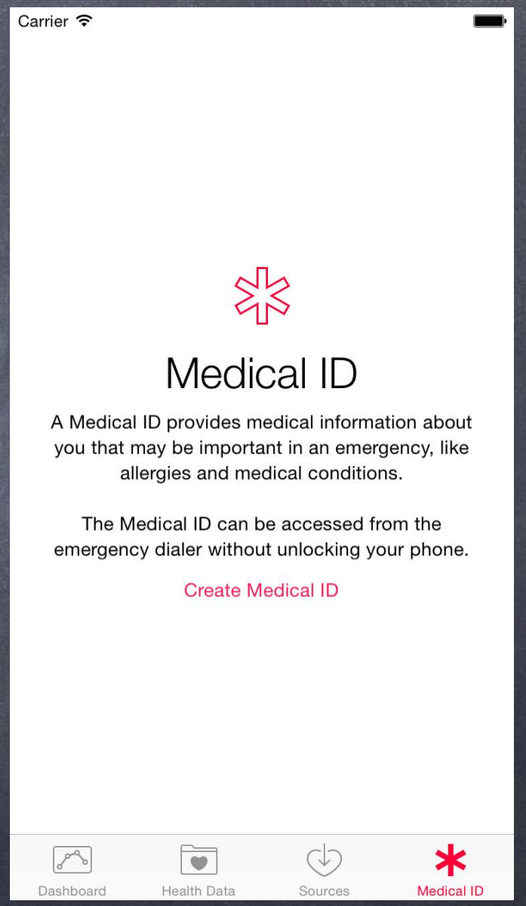
UITabBarController

- It lets the user choose between different MVCs …







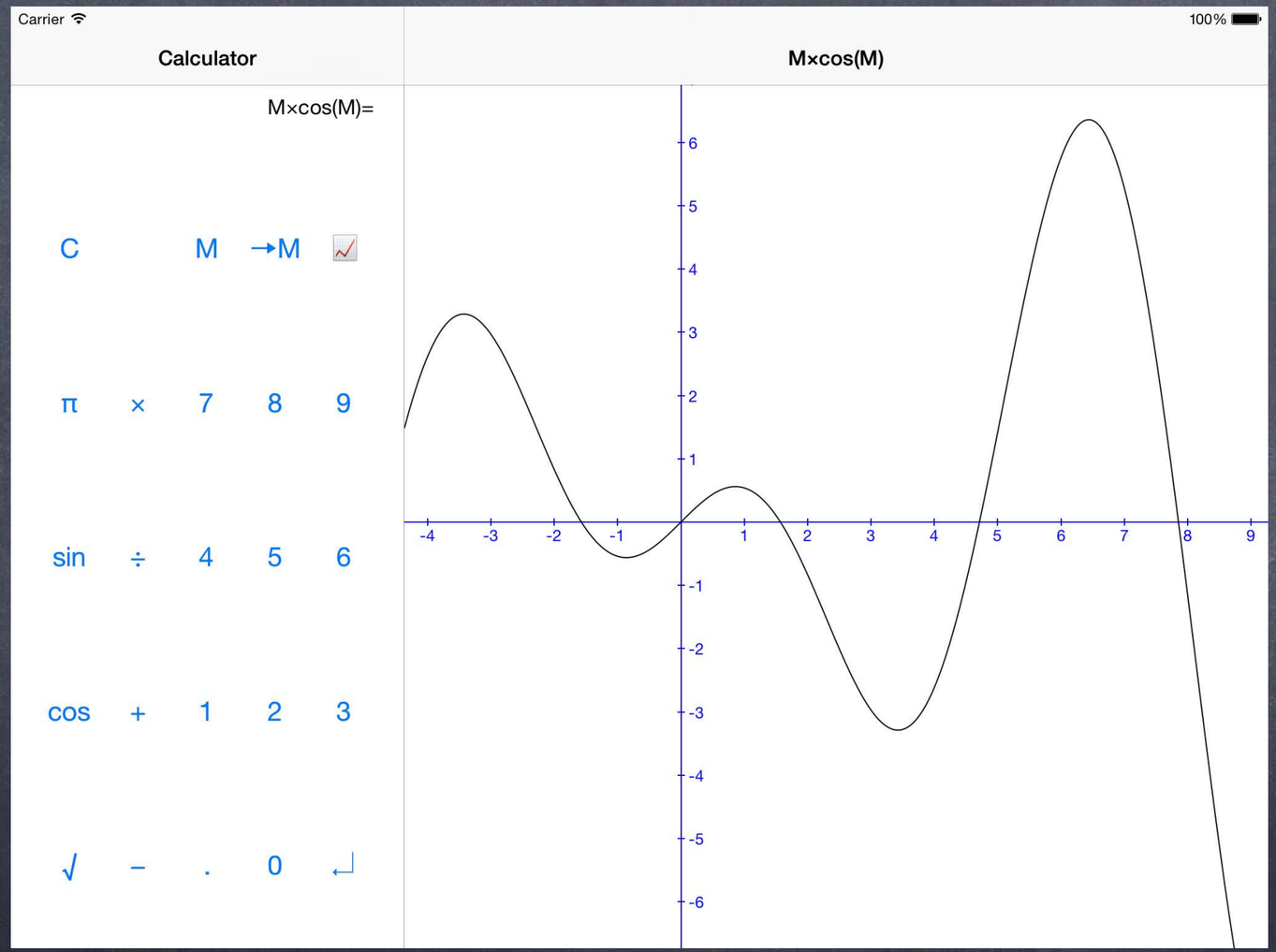


CS193p
Winter 2015

UISplitViewController

- Puts two MVCs side-by-side …

Master



Detail

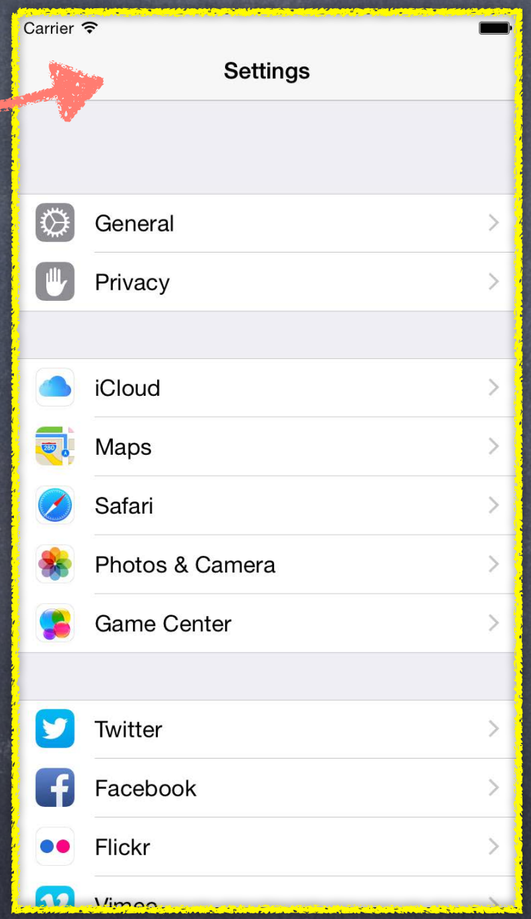
UINavigationController

- 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)

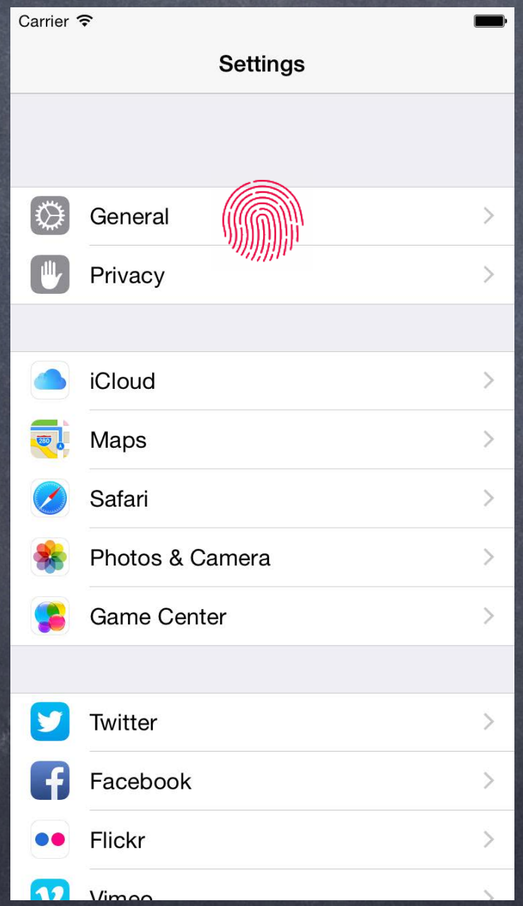
Each MVC communicates these contents via its `UIViewController`'s `navigationItem` property



An "All Settings" MVC

UINavigationController

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



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

UINavigationController

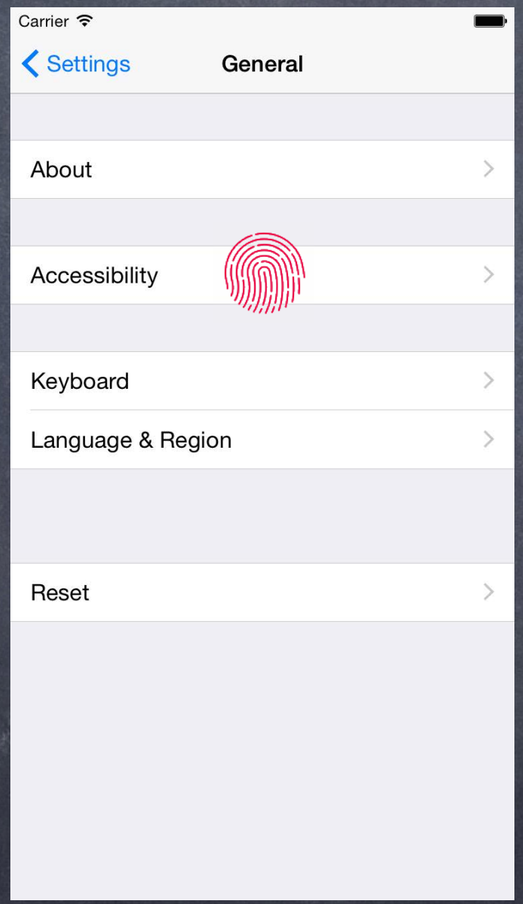
- Pushes and pops MVCs off of a stack (like a stack of cards) …

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

A "General Settings" MVC

UINavigationController

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



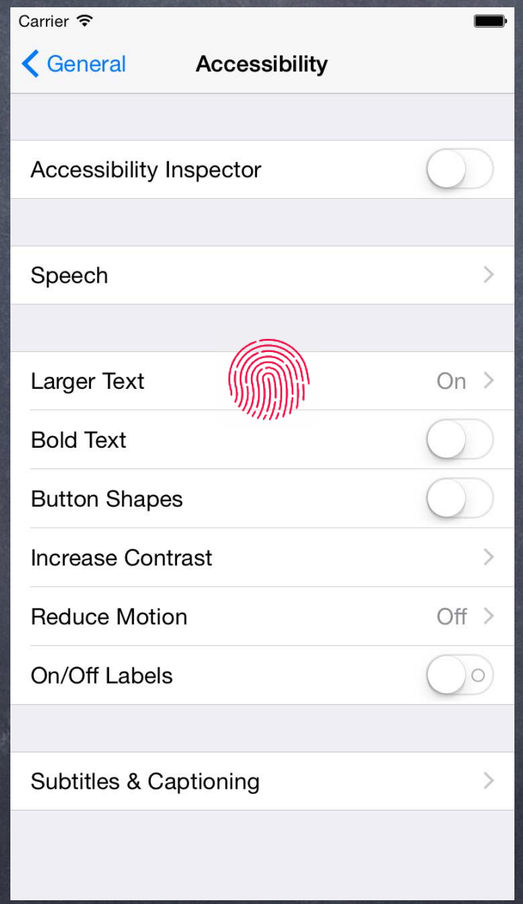
UINavigationController

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

An "Accessibility" MVC

UINavigationController

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



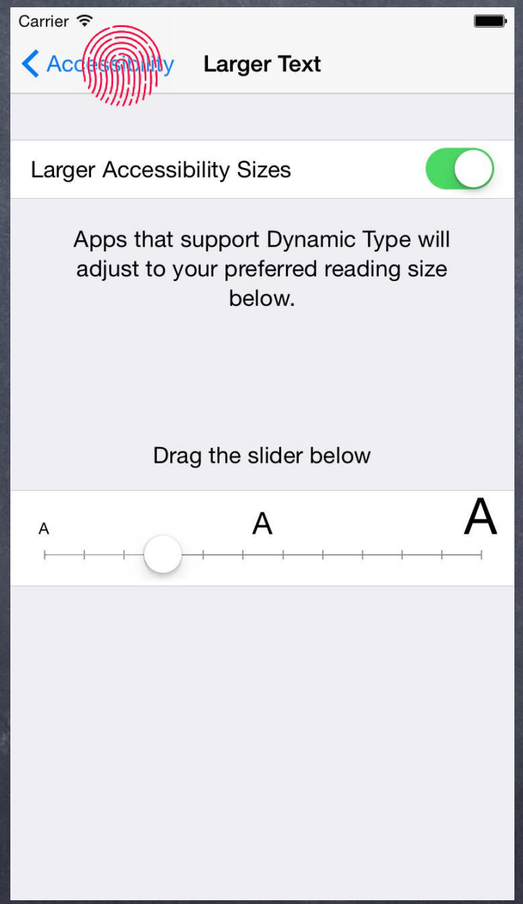
UINavigationController

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

A "Larger Text" MVC

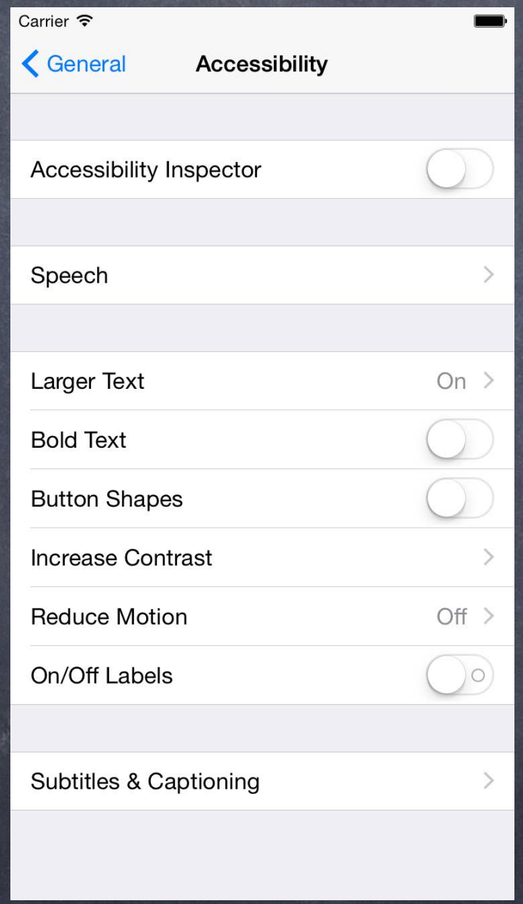
UINavigationController

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



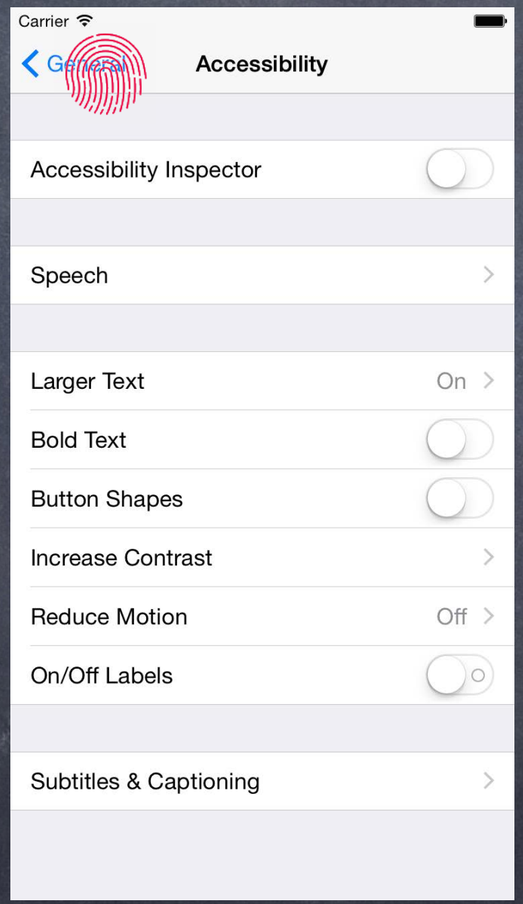
UINavigationController

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



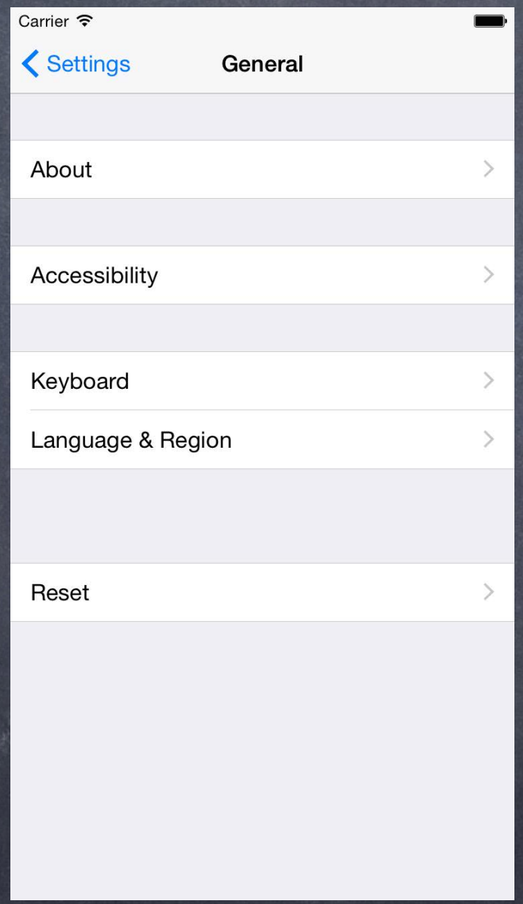
UINavigationController

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



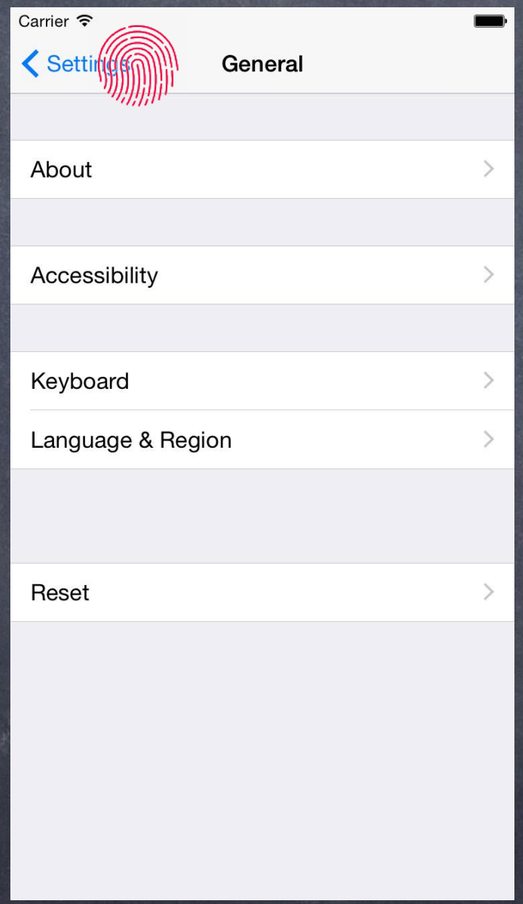
UINavigationController

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



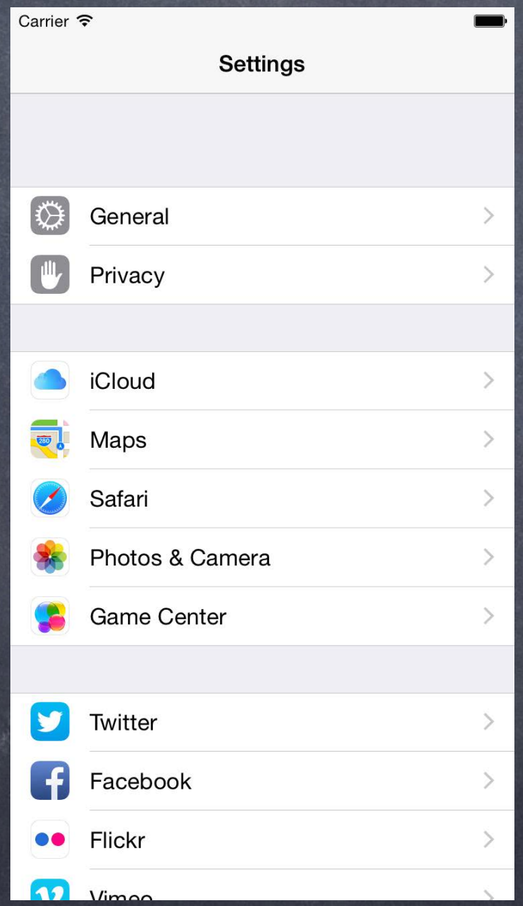
UINavigationController

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

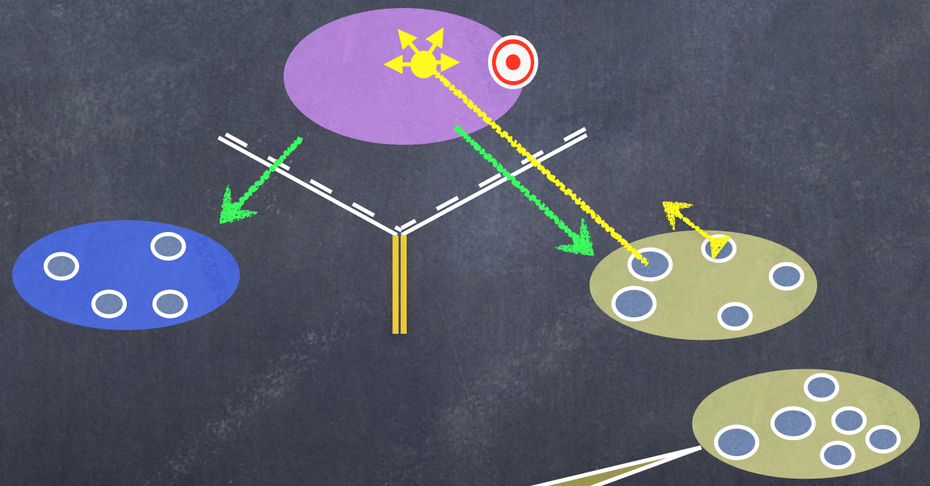


UINavigationController

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

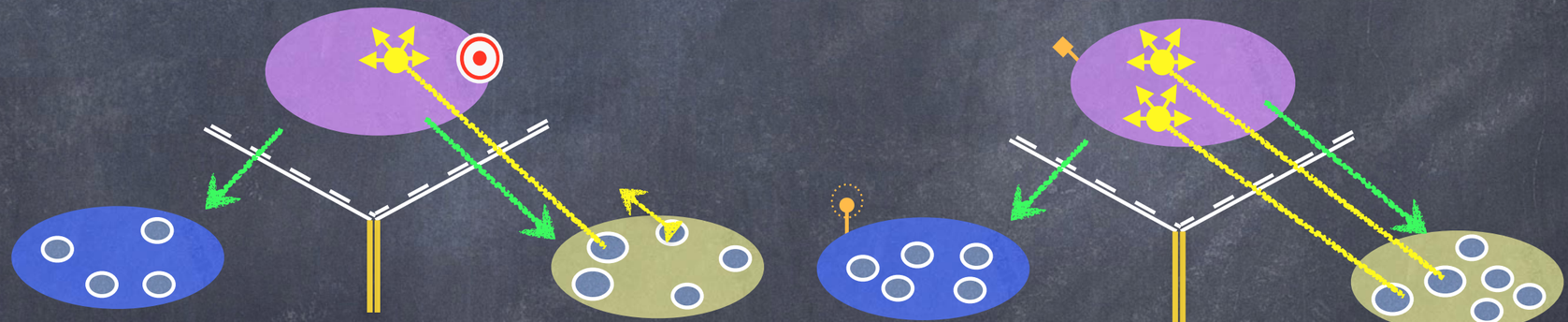


UINavigationController



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

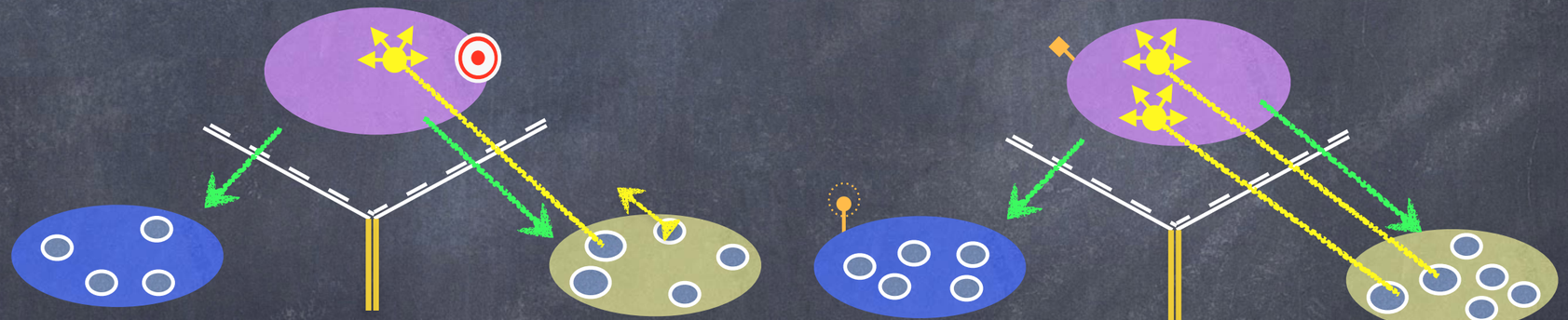
UINavigationController



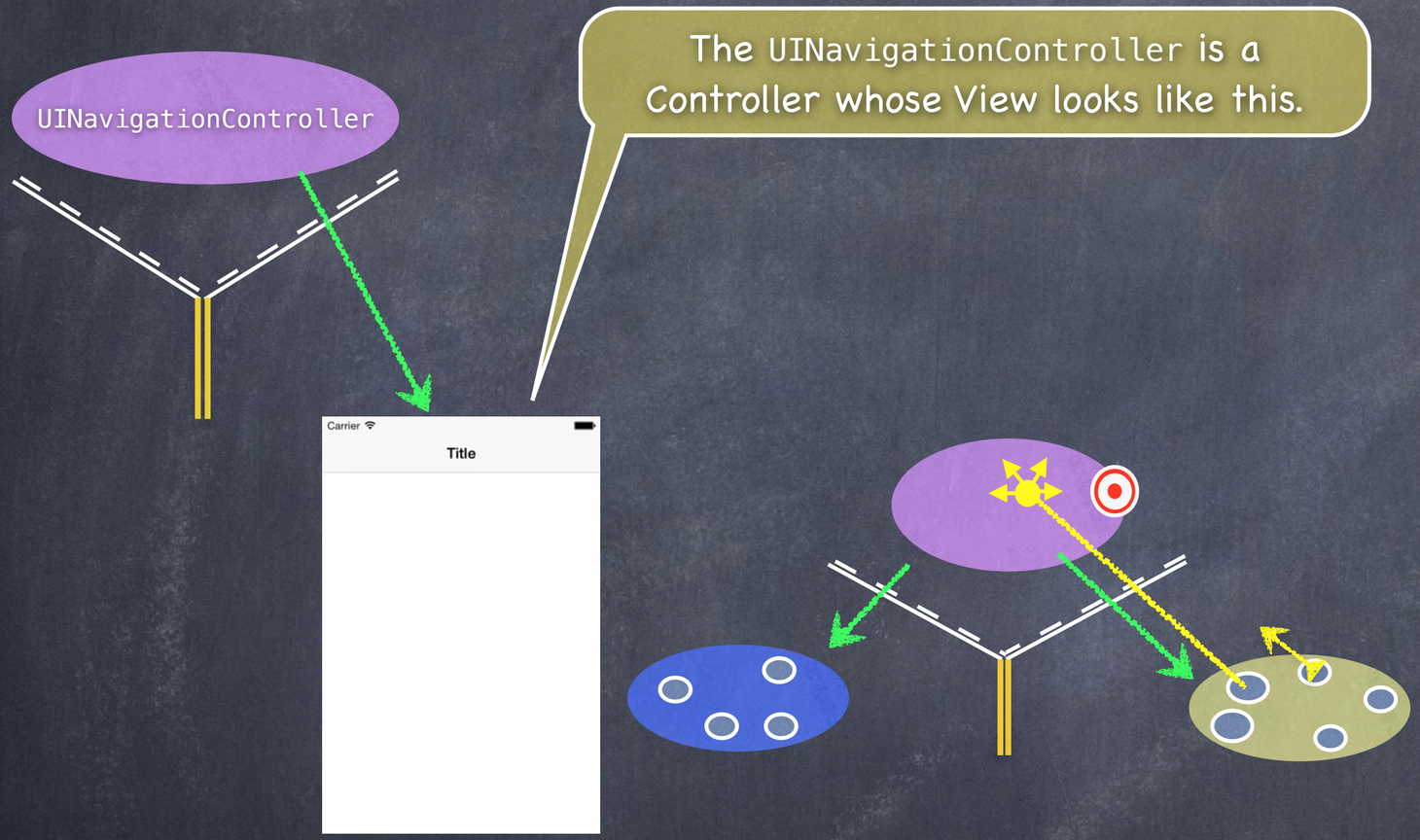
So I design a new MVC to encapsulate that functionality.

UINavigationController

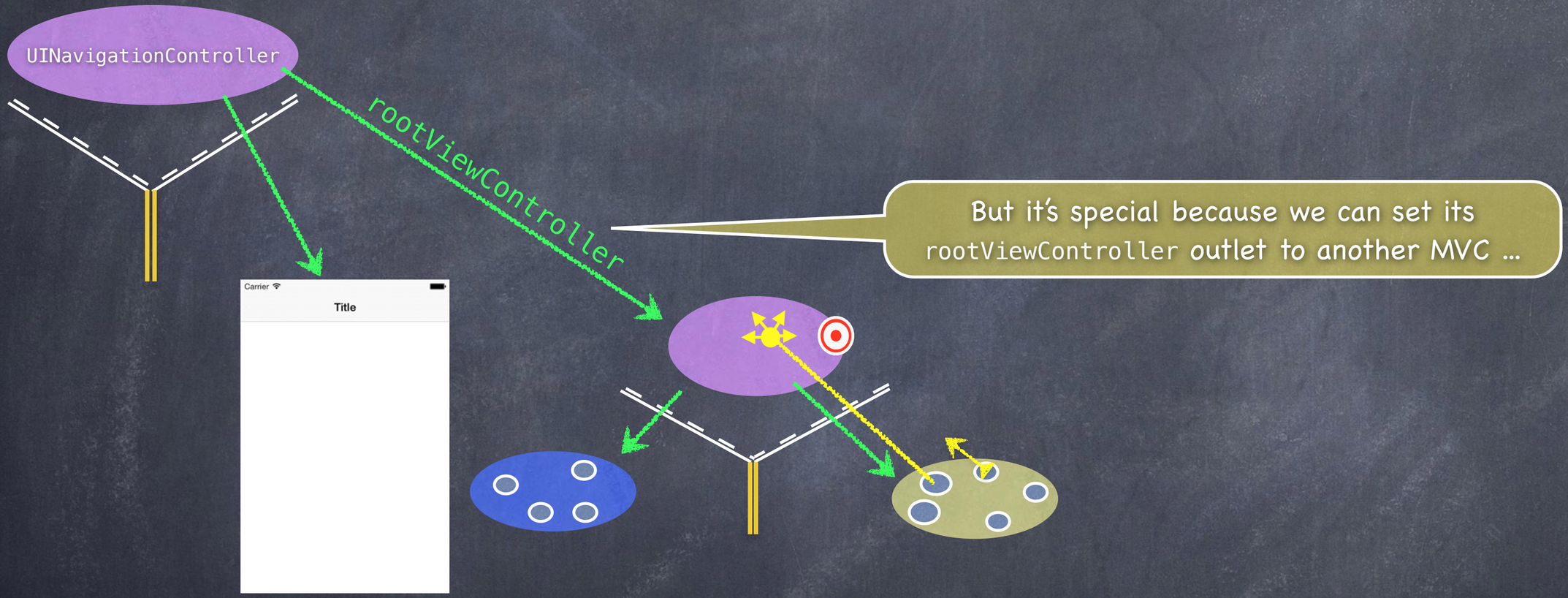
We can use a UINavigationController to let them share the screen.



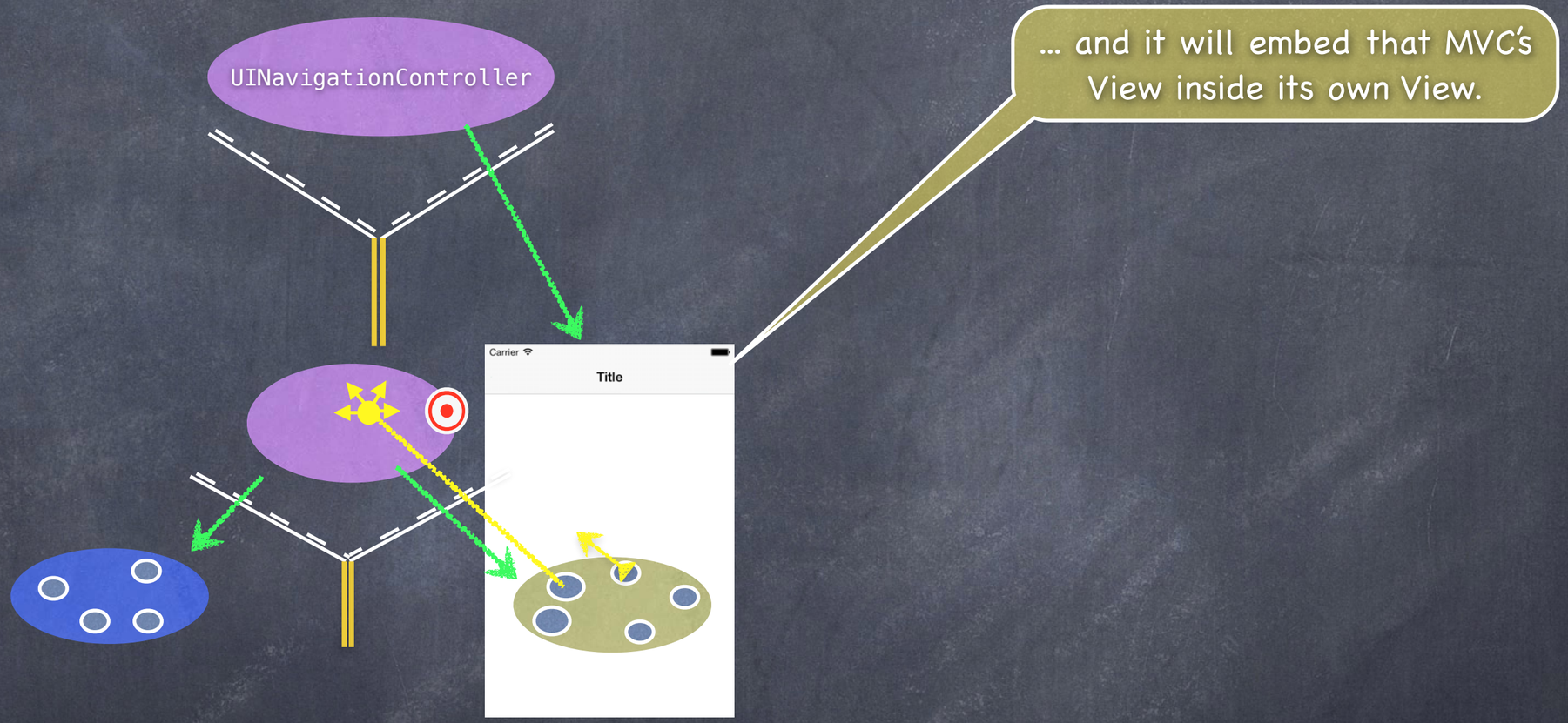
UINavigationController



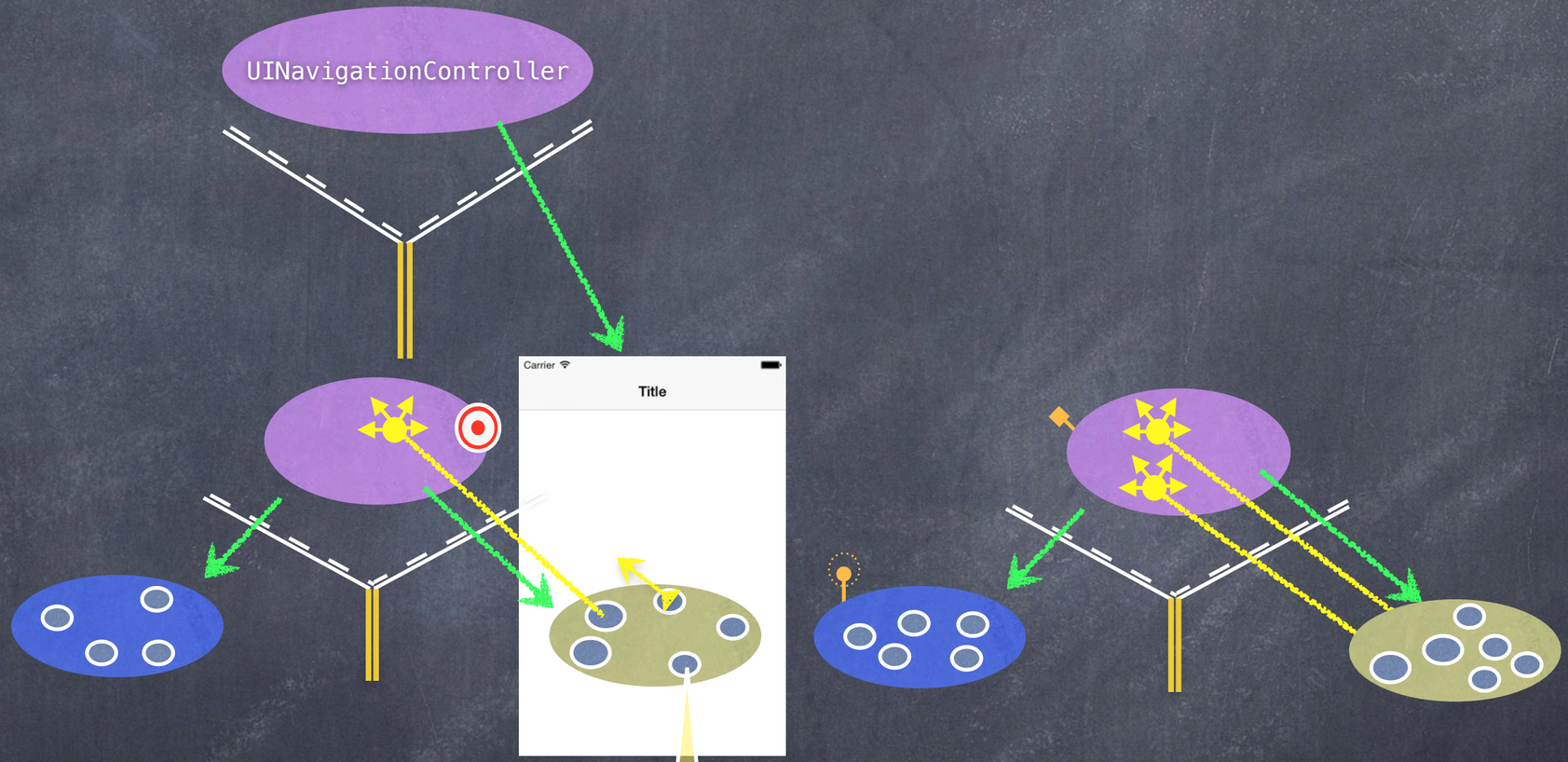
UINavigationController



UINavigationController

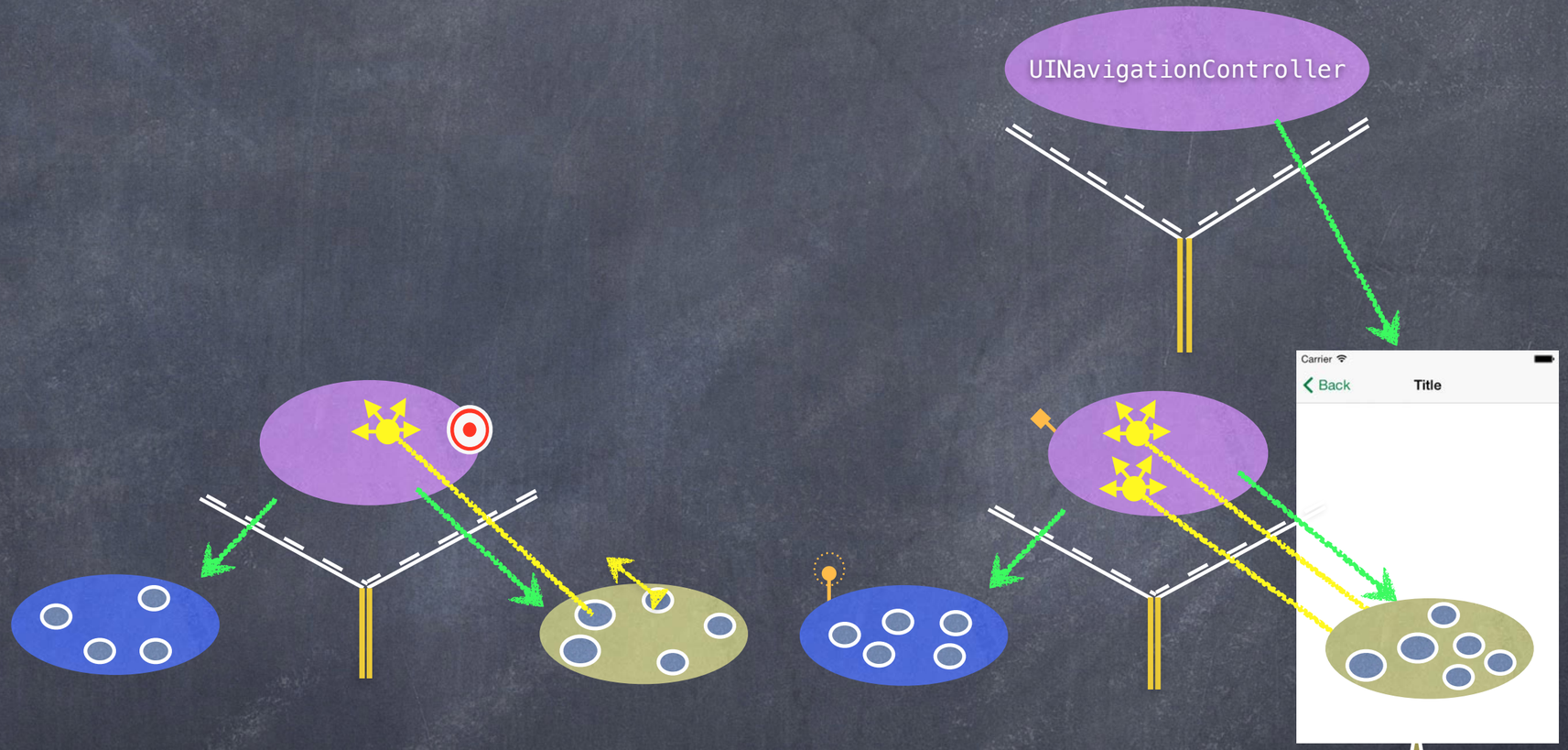


UINavigationController



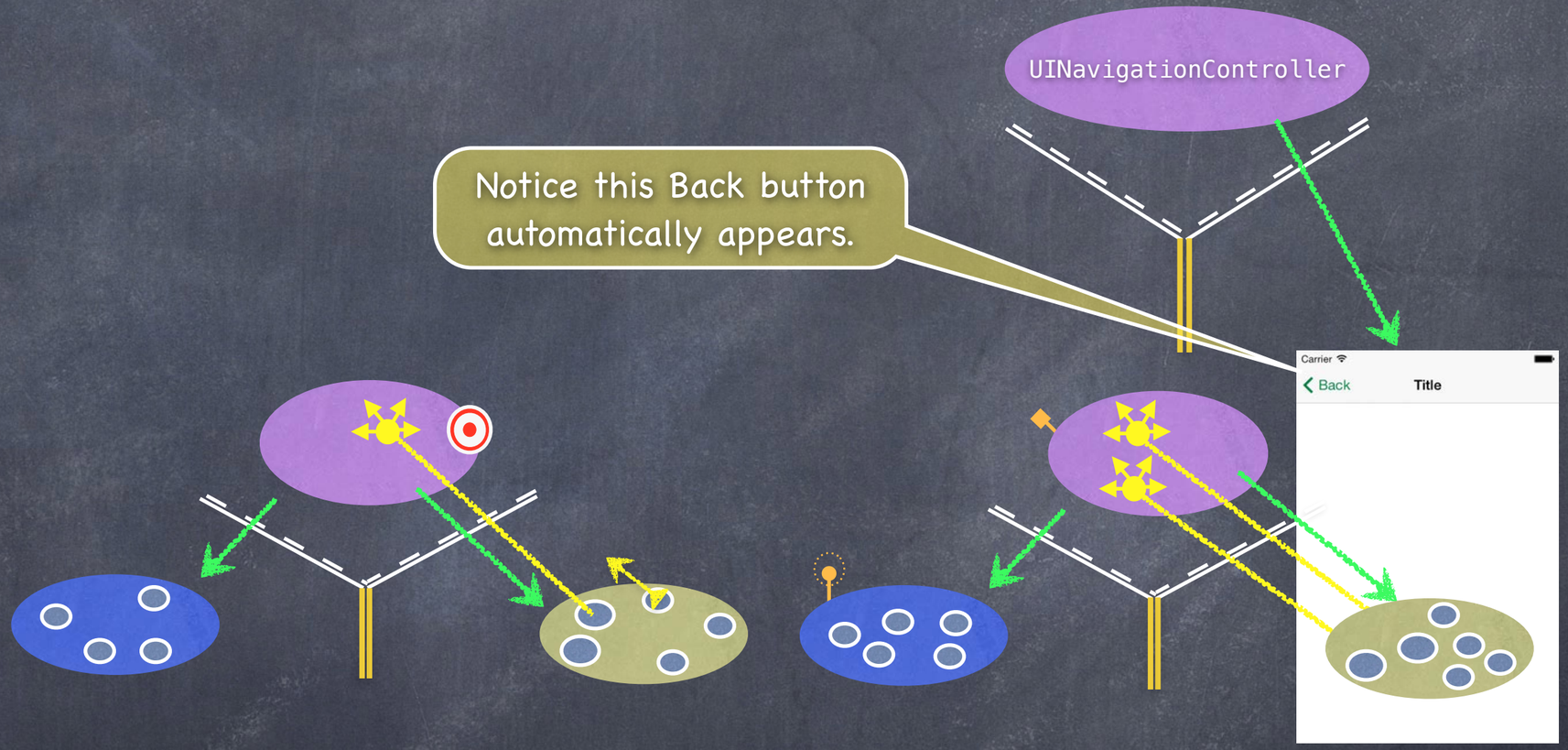
When a UI element in this View (e.g. a `UIButton`) is activated, it will segue to create a new MVC ...

UINavigationController

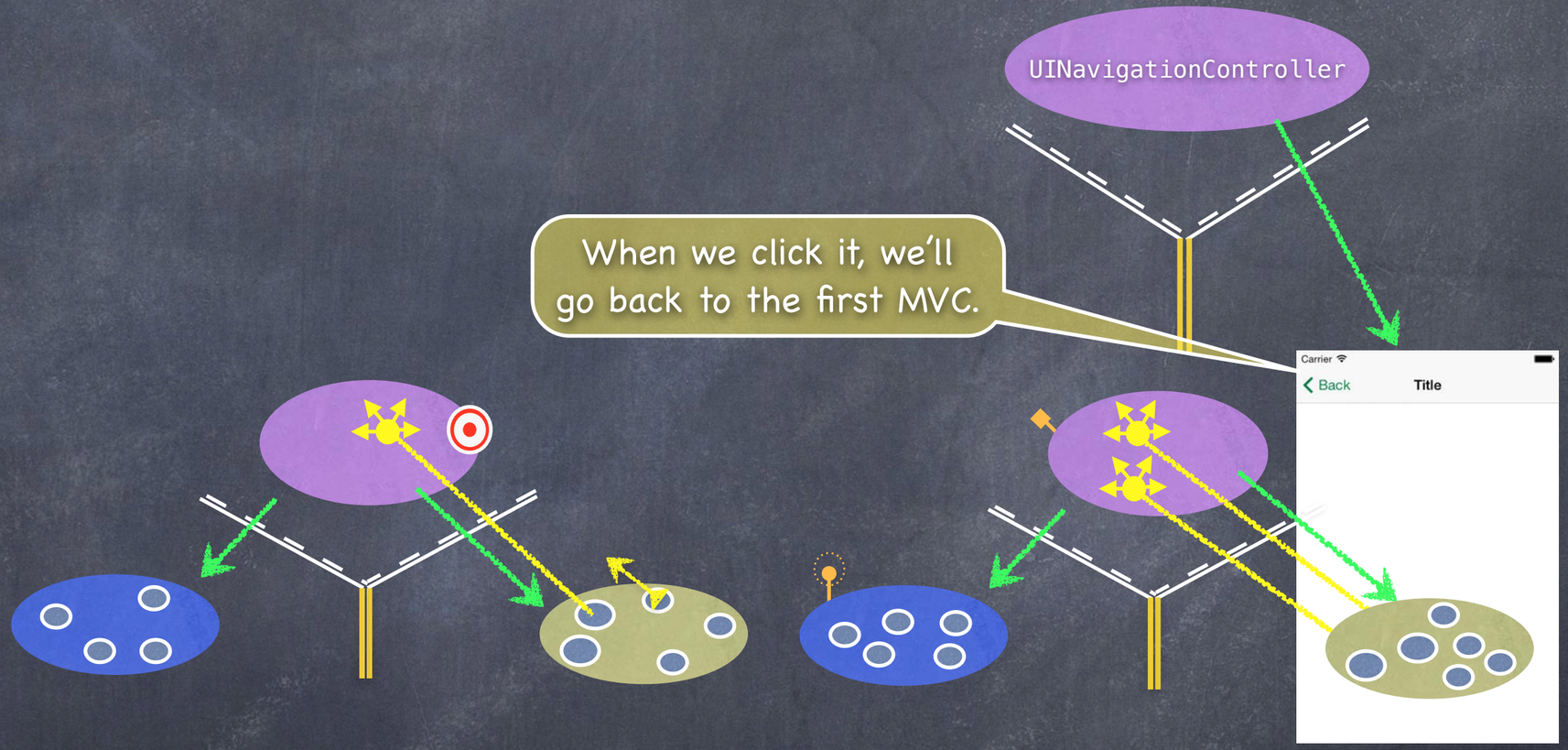


And replace the Navigation Controller's View with that new MVC's View.
We call this kind of segue a "Show (push) segue".

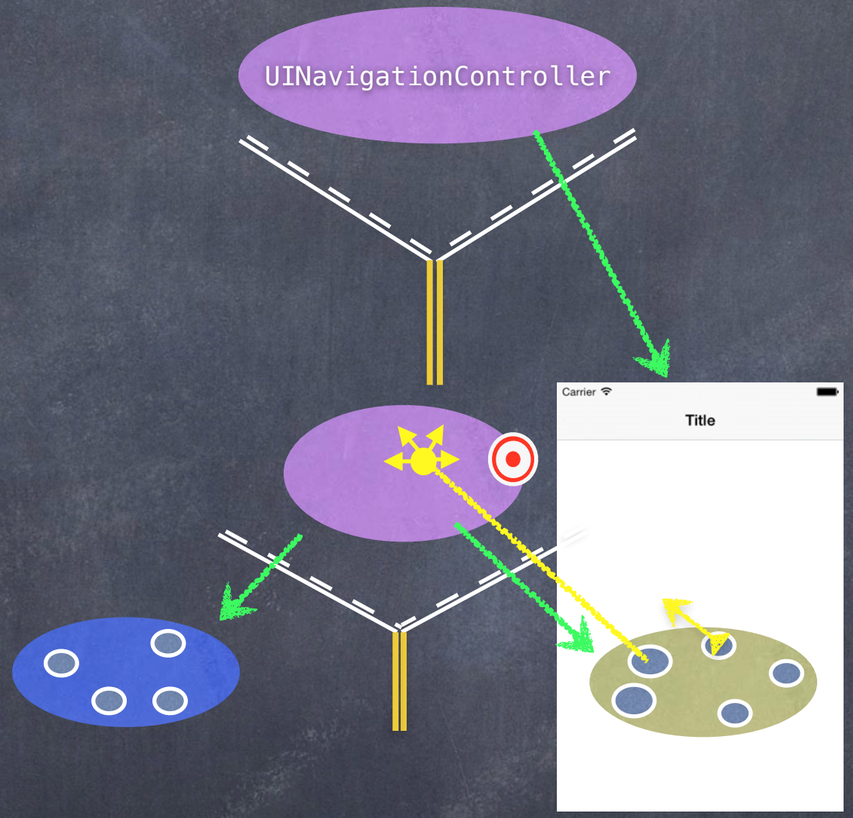
UINavigationController



UINavigationController



UINavigationController



Notice that the other MVC is completely gone.

Accessing the sub-MVCs

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

```
var viewControllers: [UIViewController] { get set } // possibly an optional
// 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 …

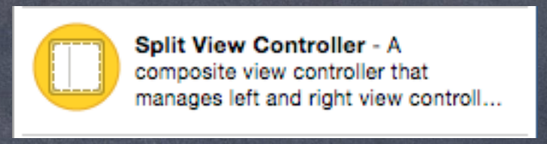
```
if let detailVC: UIViewController = splitViewController?.viewControllers[1] { … }
```

Wiring up MVCs

- How do we wire all this stuff up?

  Let's say we have a Calculator MVC and a Calculator Graphing MVC

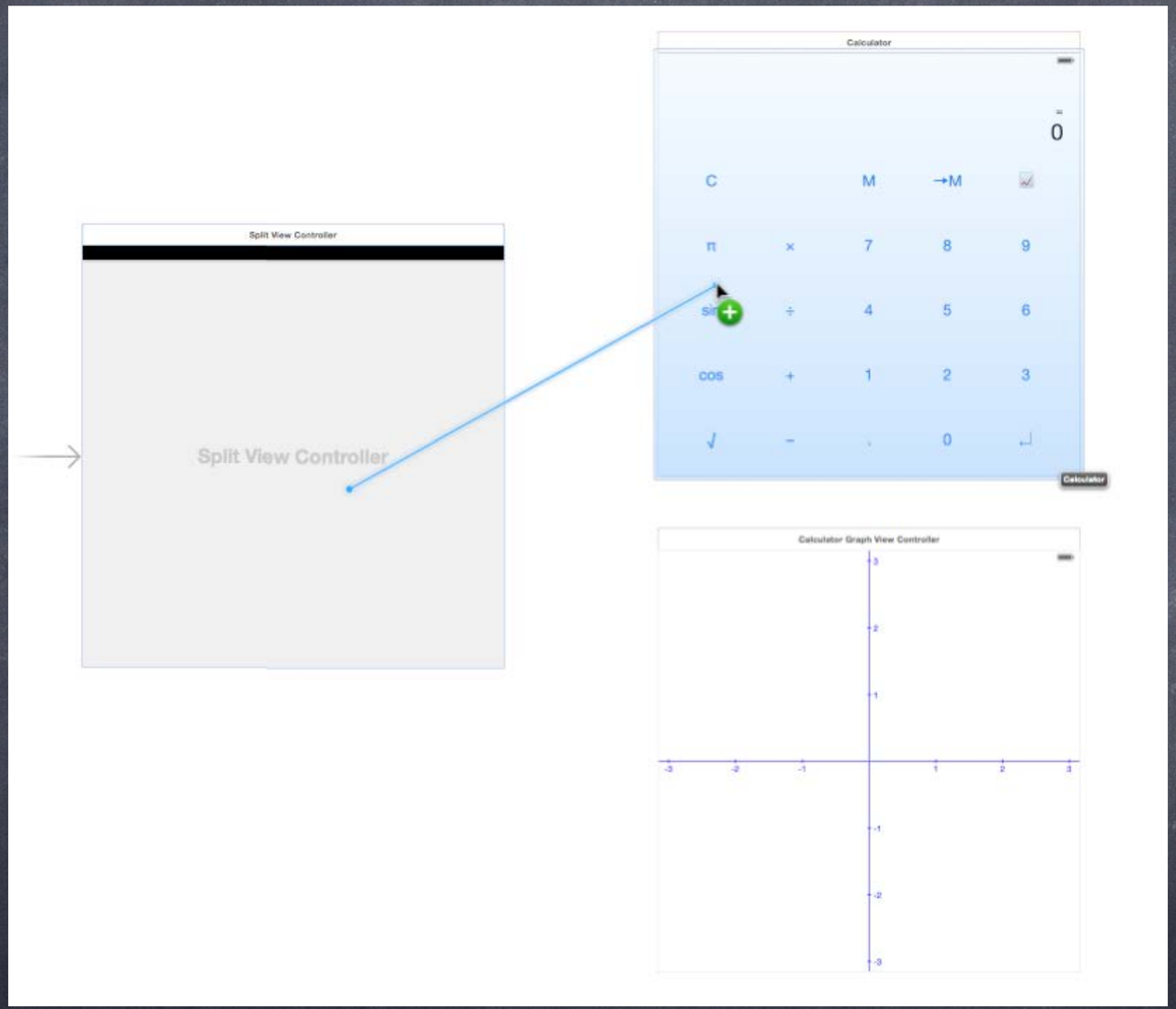
  How do we hook them up to be the two sides of a Split View?

  Just drag out a 

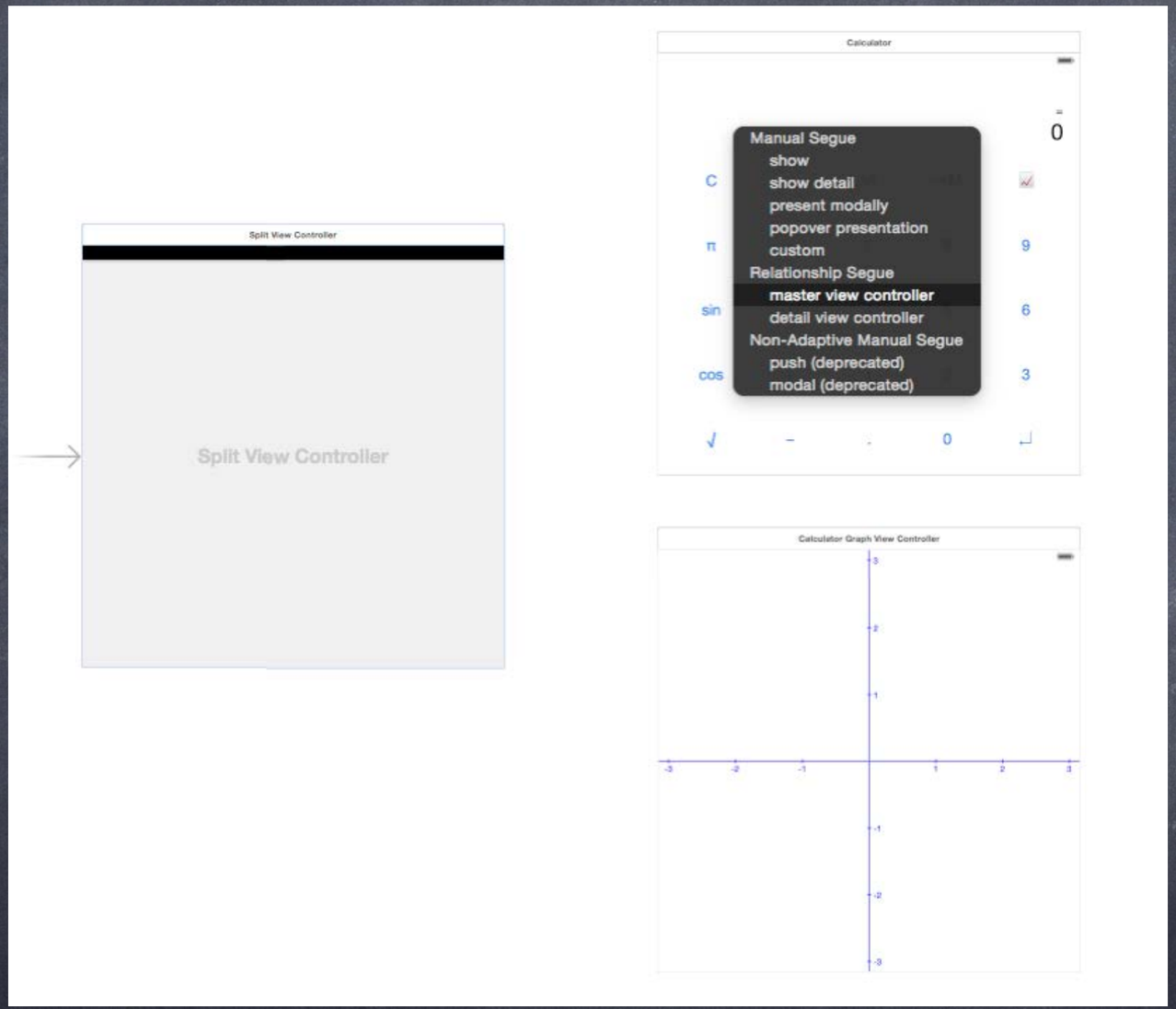
  (and delete all the extra VCs it brings with it)

  Then ctrl-drag from the `UISplitViewController` to the master and detail MVCs ...

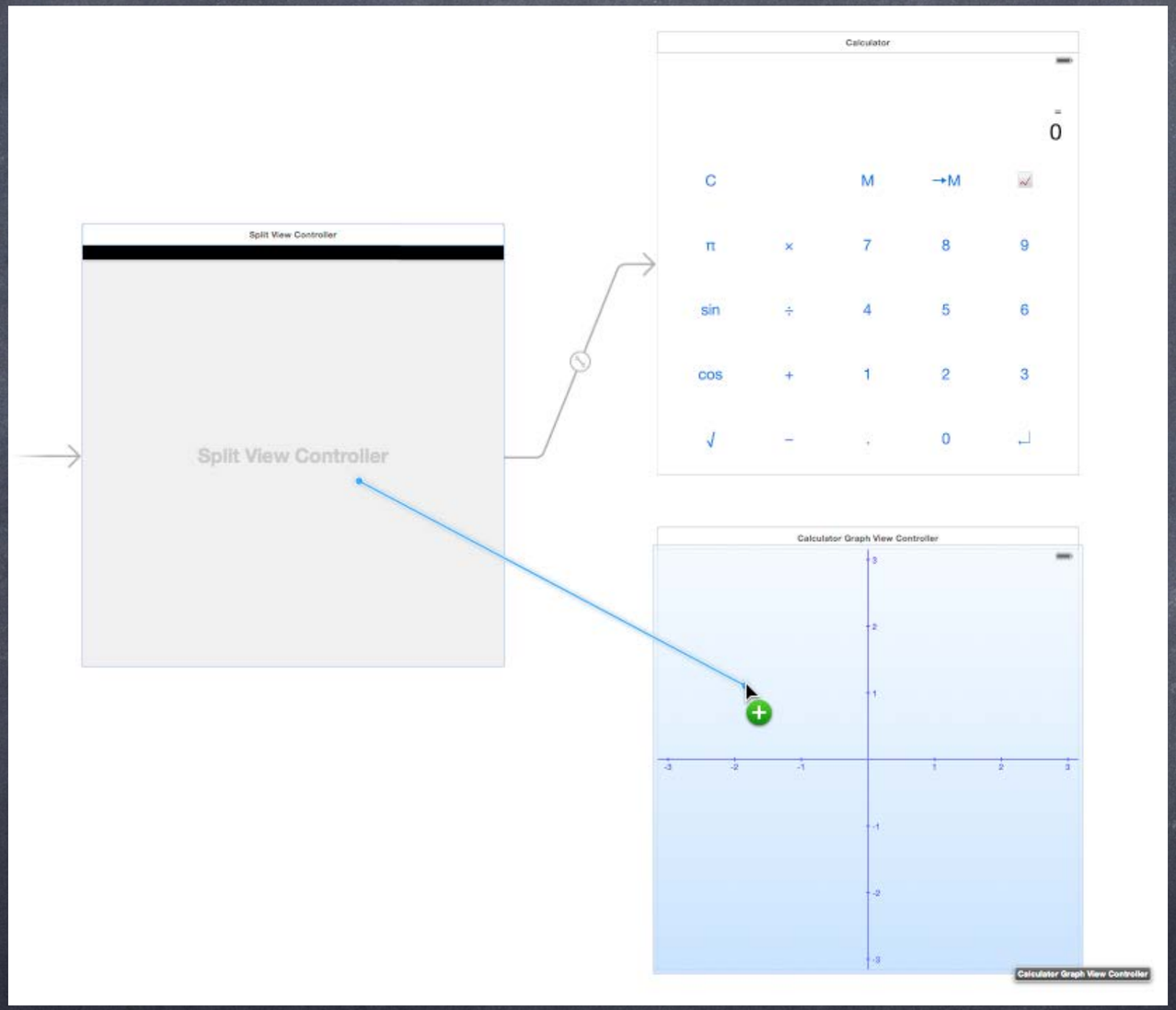
Wiring up MVCs



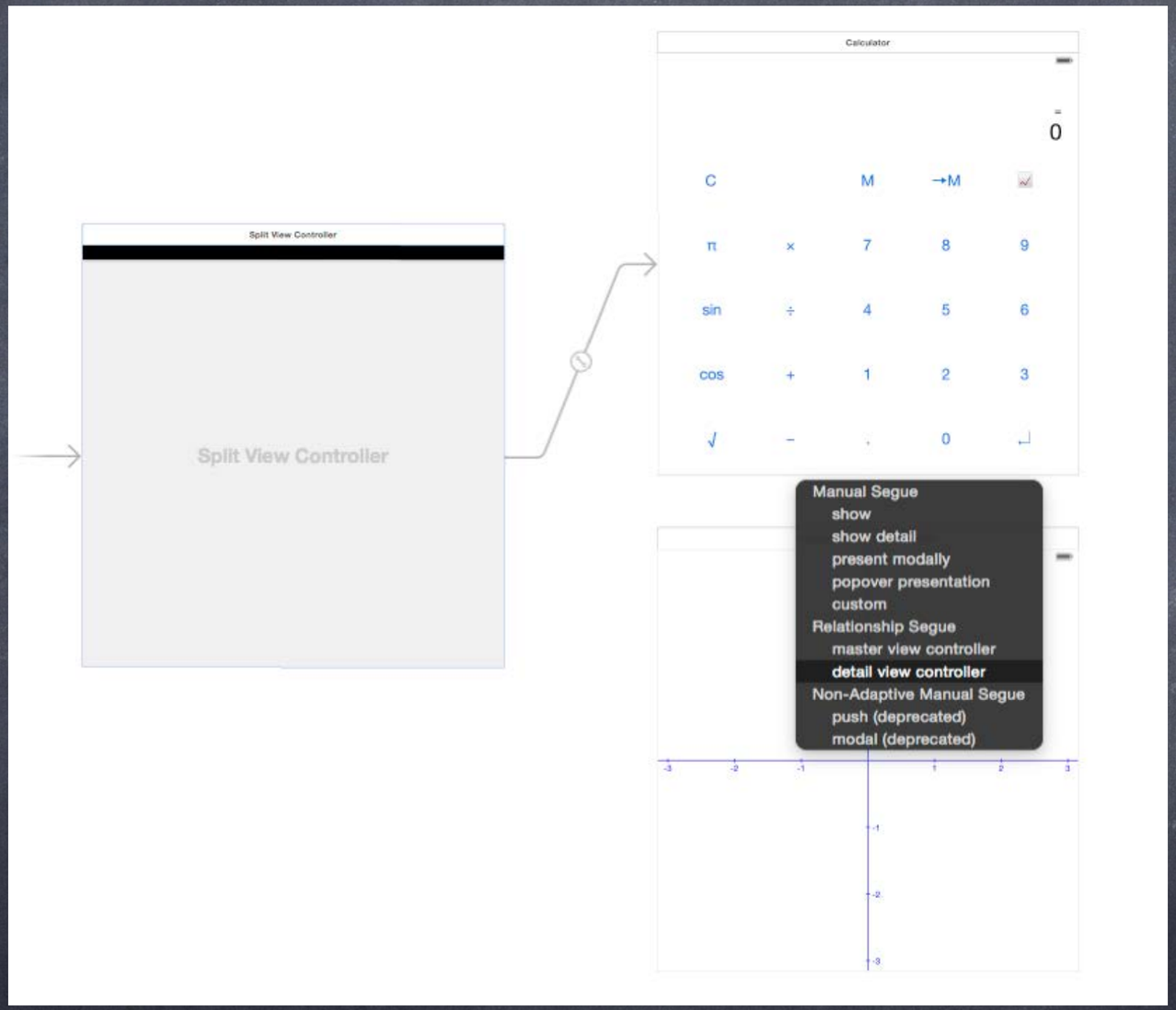
Wiring up MVCs



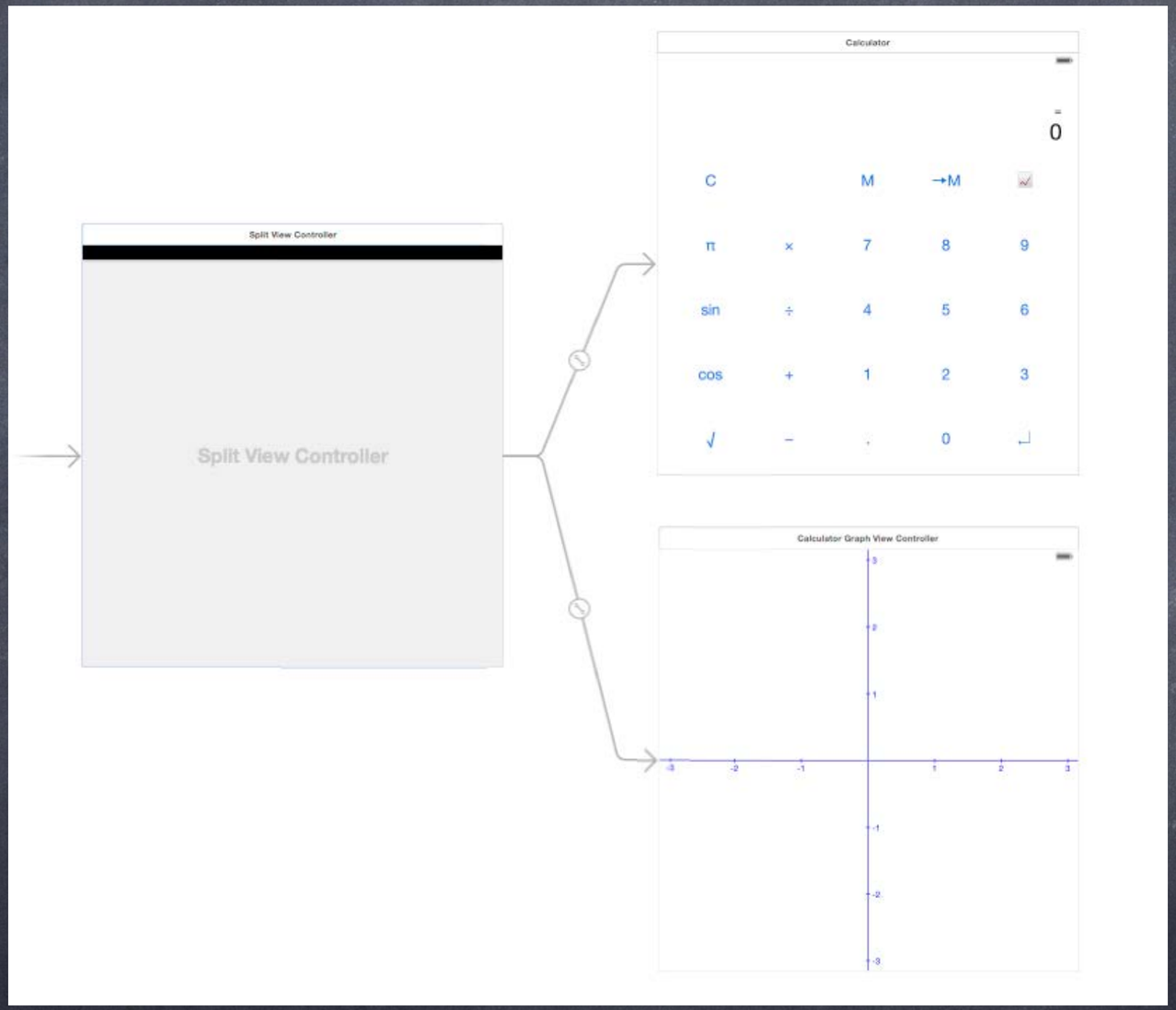
Wiring up MVCs



Wiring up MVCs



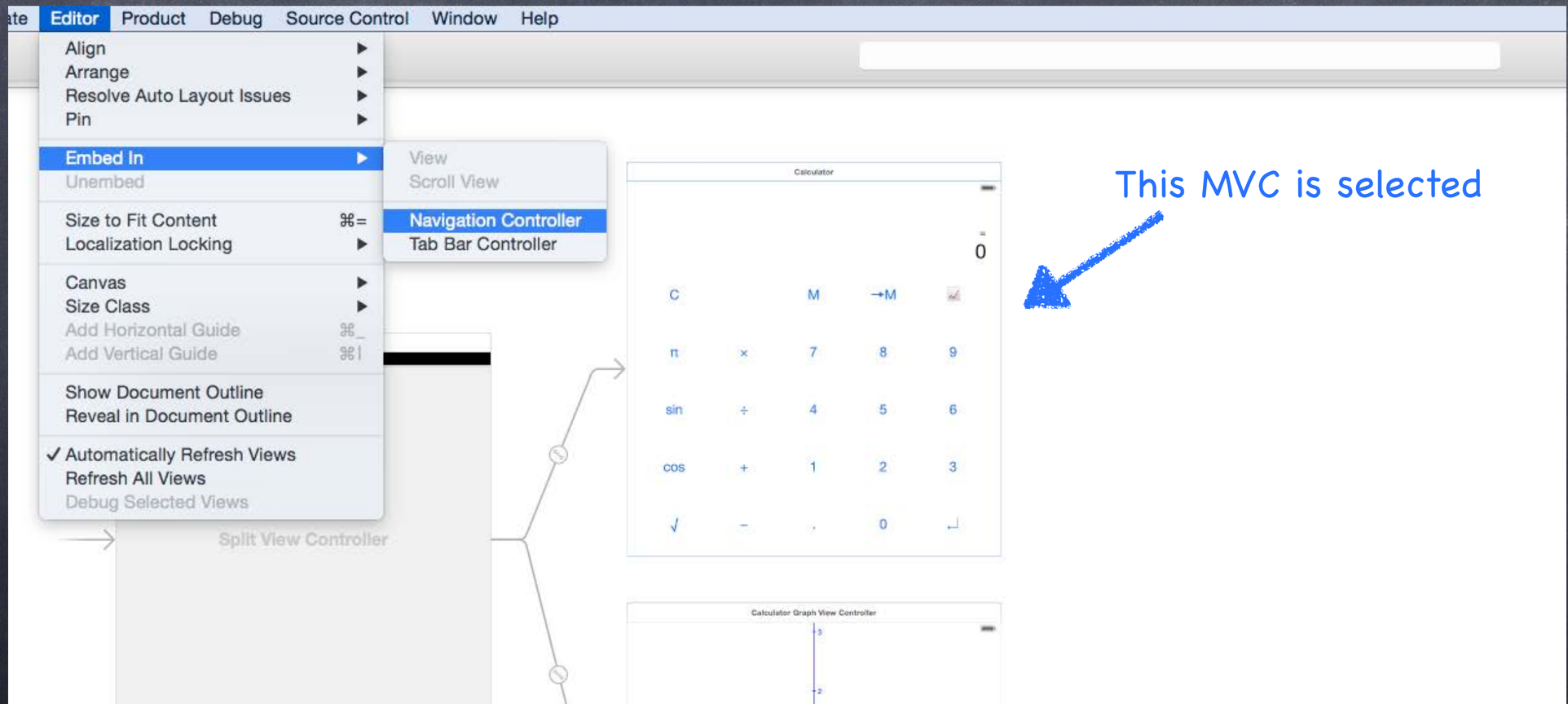
Wiring up MVCs



CS193p
Winter 2015

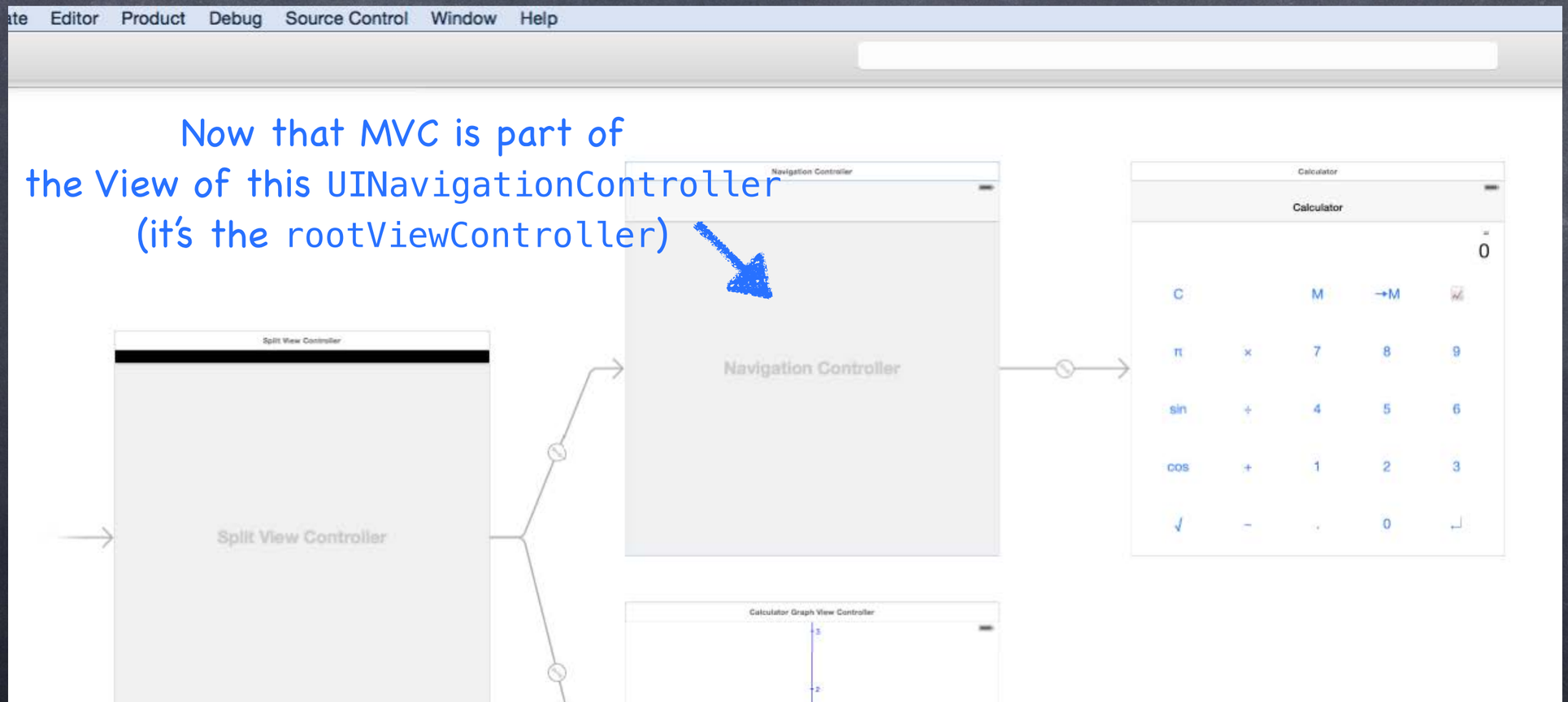
Wiring up MVCs

- But split view can only do its thing properly on iPad
  - So we need to put some Navigation Controllers in there so it will work on iPhone
  - The Navigation Controllers will be good for iPad too because the MVCs will get titles
  - The simplest way to wrap a Navigation Controller around an MVC is with Editor->Embed In



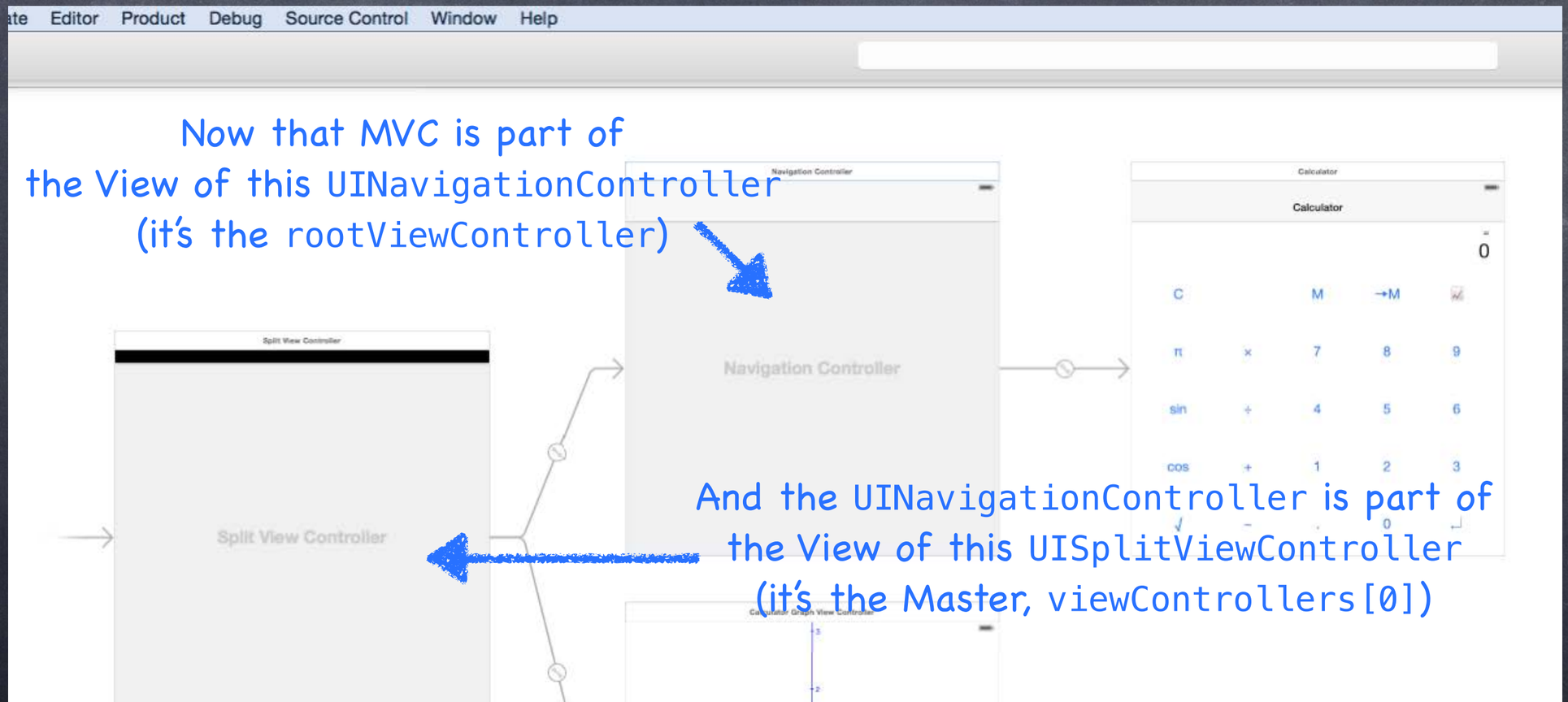
Wiring up MVCs

- But split view can only do its thing properly on iPad
  - So we need to put some Navigation Controllers in there so it will work on iPhone
  - The Navigation Controllers will be good for iPad too because the MVCs will get titles
  - The simplest way to wrap a Navigation Controller around an MVC is with Editor->Embed In



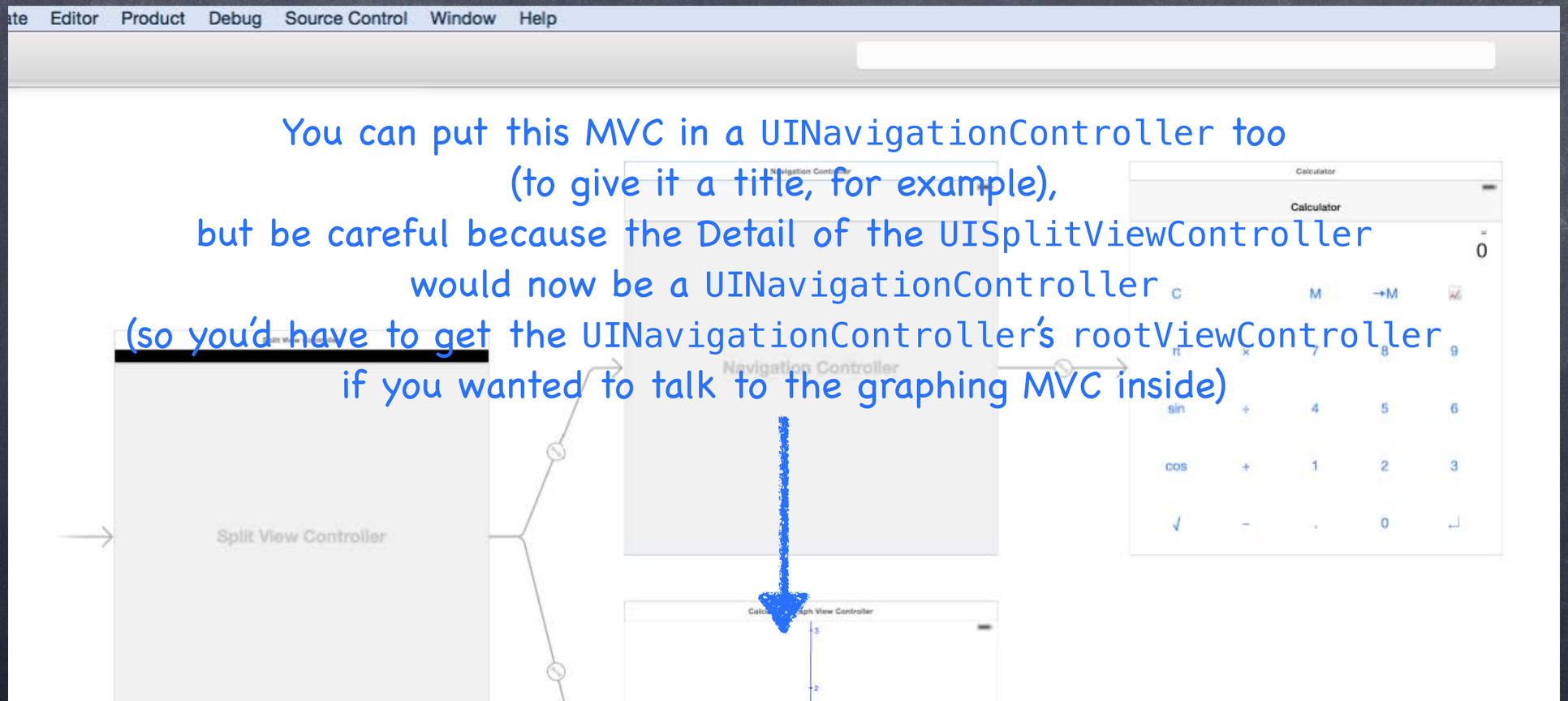
Wiring up MVCs

- But split view can only do its thing properly on iPad
  - So we need to put some Navigation Controllers in there so it will work on iPhone
  - The Navigation Controllers will be good for iPad too because the MVCs will get titles
  - The simplest way to wrap a Navigation Controller around an MVC is with Editor->Embed In



Wiring up MVCs

- But split view can only do its thing properly on iPad
  - So we need to put some Navigation Controllers in there so it will work on iPhone
  - The Navigation Controllers will be good for iPad too because the MVCs will get titles
  - The simplest way to wrap a Navigation Controller around an MVC is with Editor->Embed In



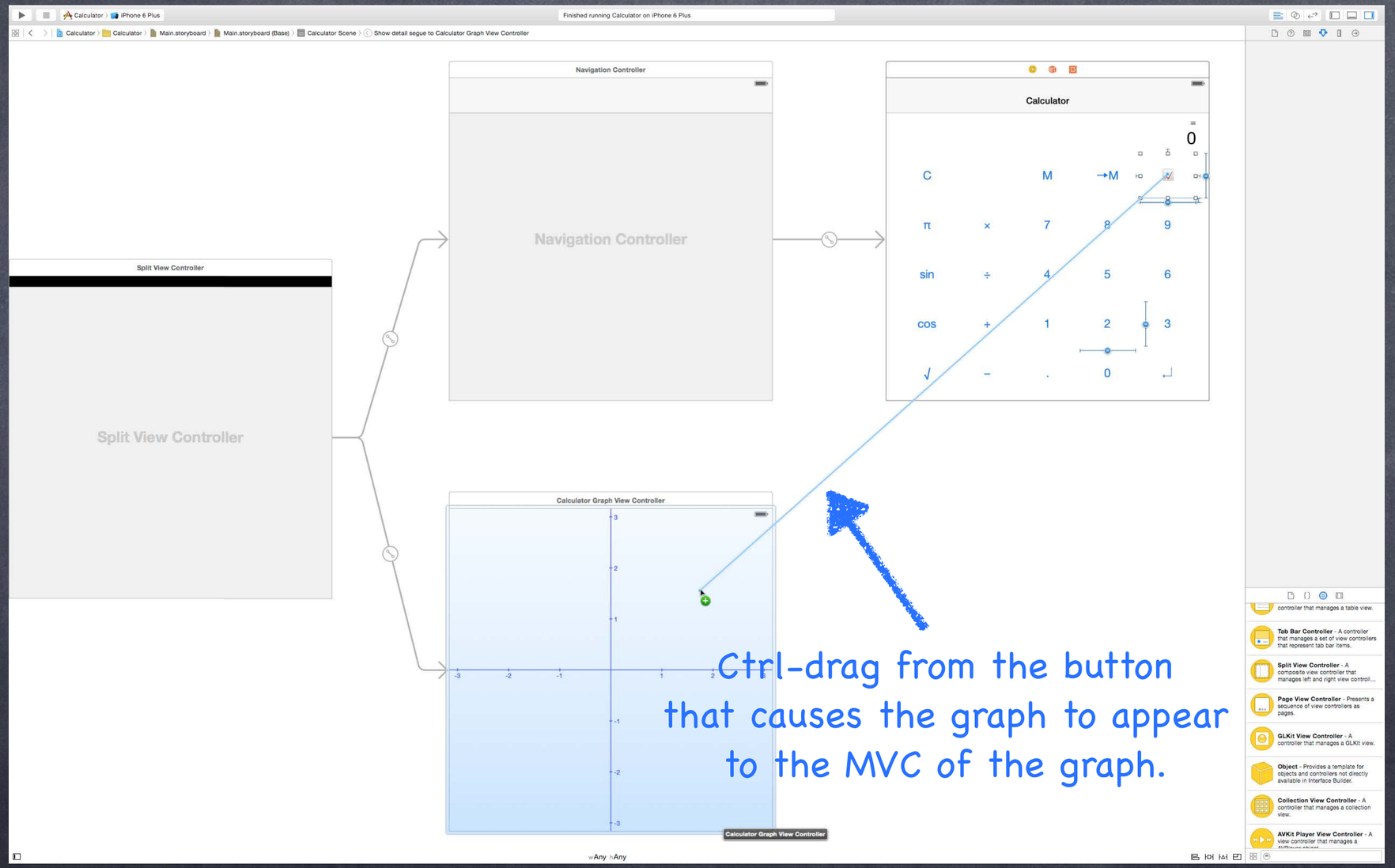
Segues

- We've built up our Controllers of Controllers, now what?
  - Now we need to make it so that one MVC can cause another to appear
  - We call that a "segue"
- Kinds of segues (they will adapt to their environment)
  - Show Segue (will push in a Navigation Controller, else Modal)
  - Show Detail Segue (will show in Detail of a Split View or will push in a Navigation Controller)
  - Modal Segue (take over the entire screen while the MVC is up)
  - Popover Segue (make the MVC appear in a little popover window)
- Segues always create a new instance of an MVC
  - This is important to understand
  - The Detail of a Split View will get replaced with a new instance of that MVC
  - When you segue in a Navigation Controller it will not segue to some old instance, it'll be new

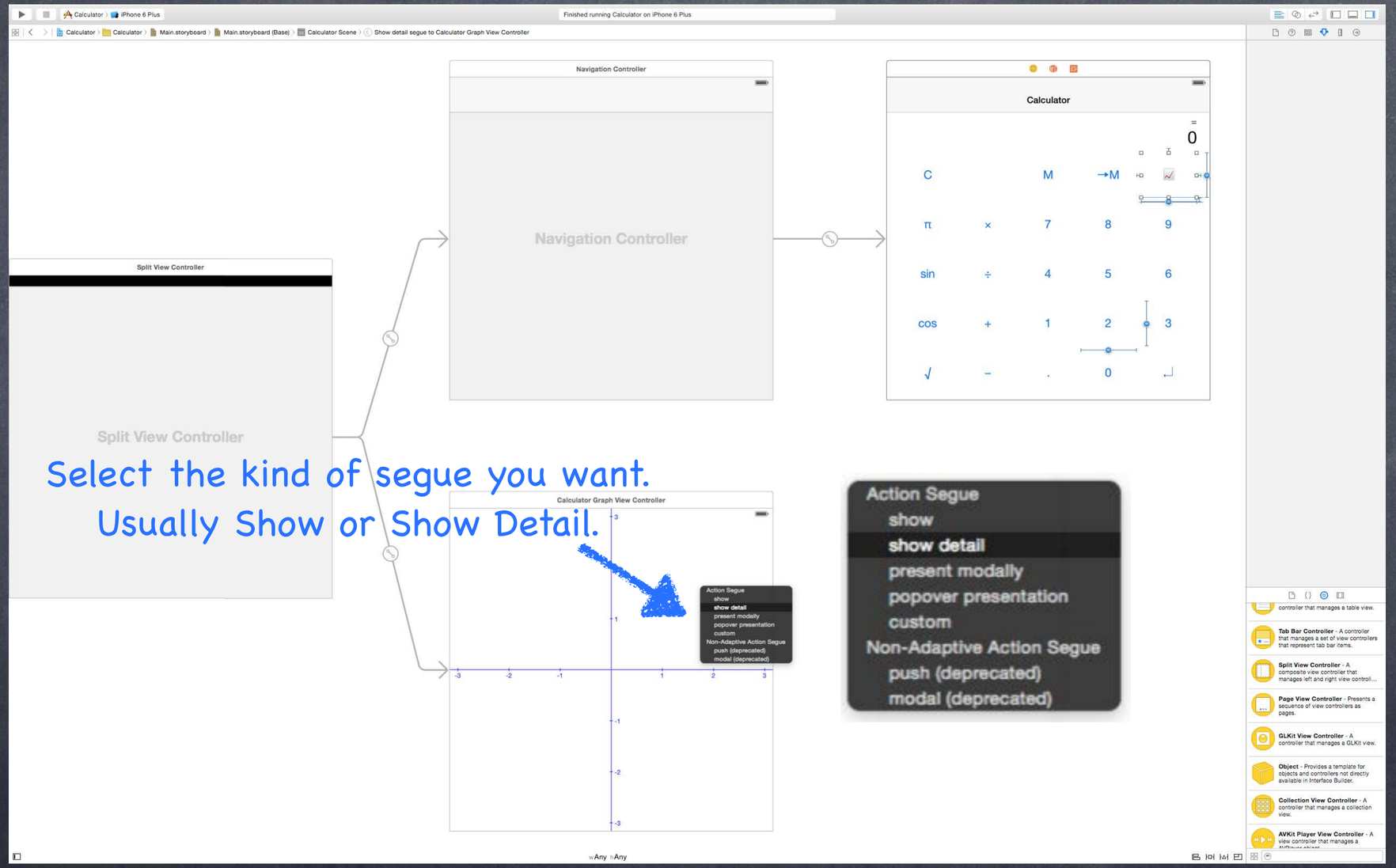
Segues

- How do we make these segues happen?
  - Ctrl-drag in a storyboard from an instigator (like a button) to the MVC to segue to
  - Can be done in code as well

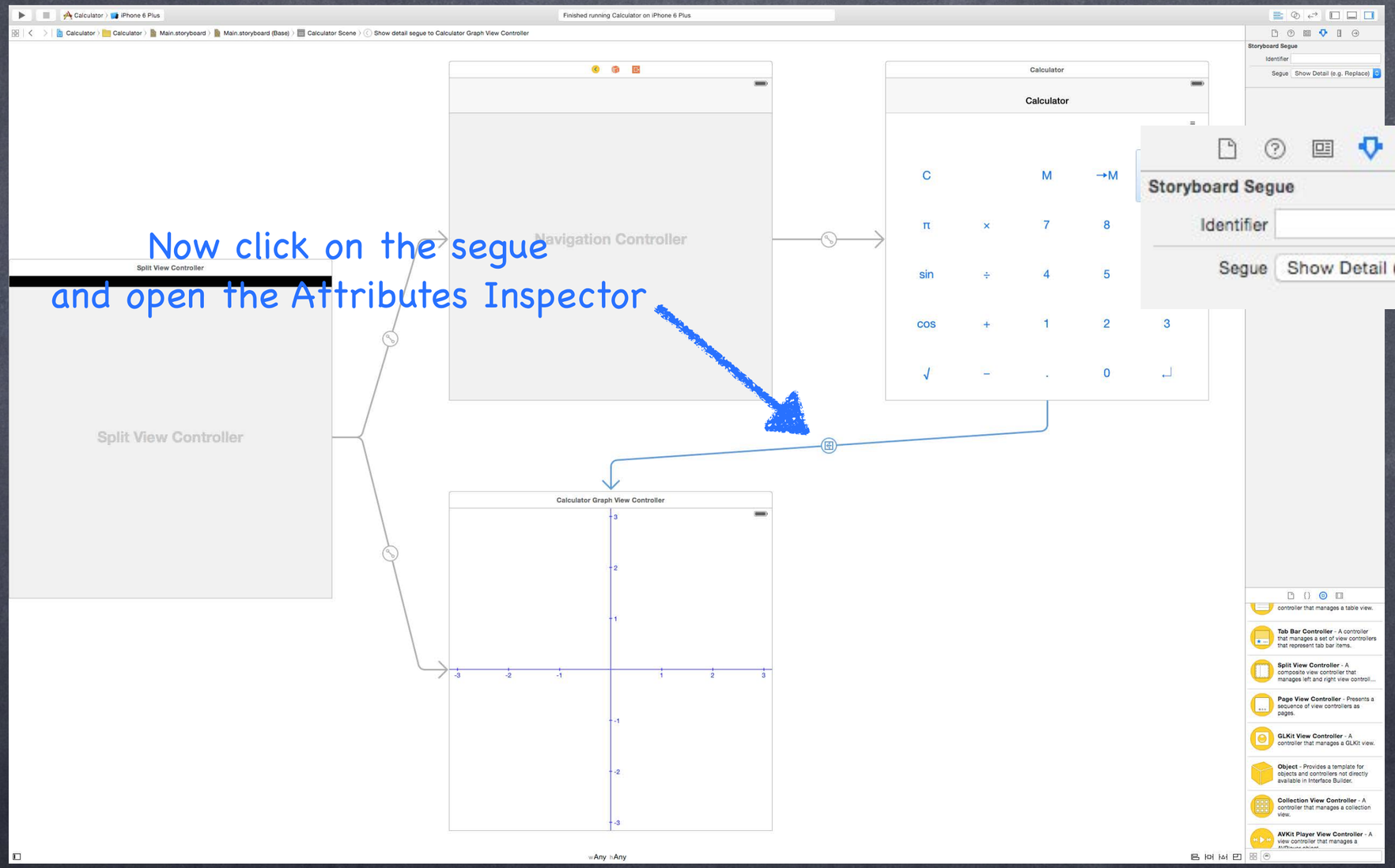
Segues


Navigation Controller
Navigation Controller
Calculator
Split View Controller
Split View Controller
Calculator Graph View Controller
Calculator Graph View Controller
Ctrl-drag from the button
that causes the graph to appear
to the MVC of the graph.

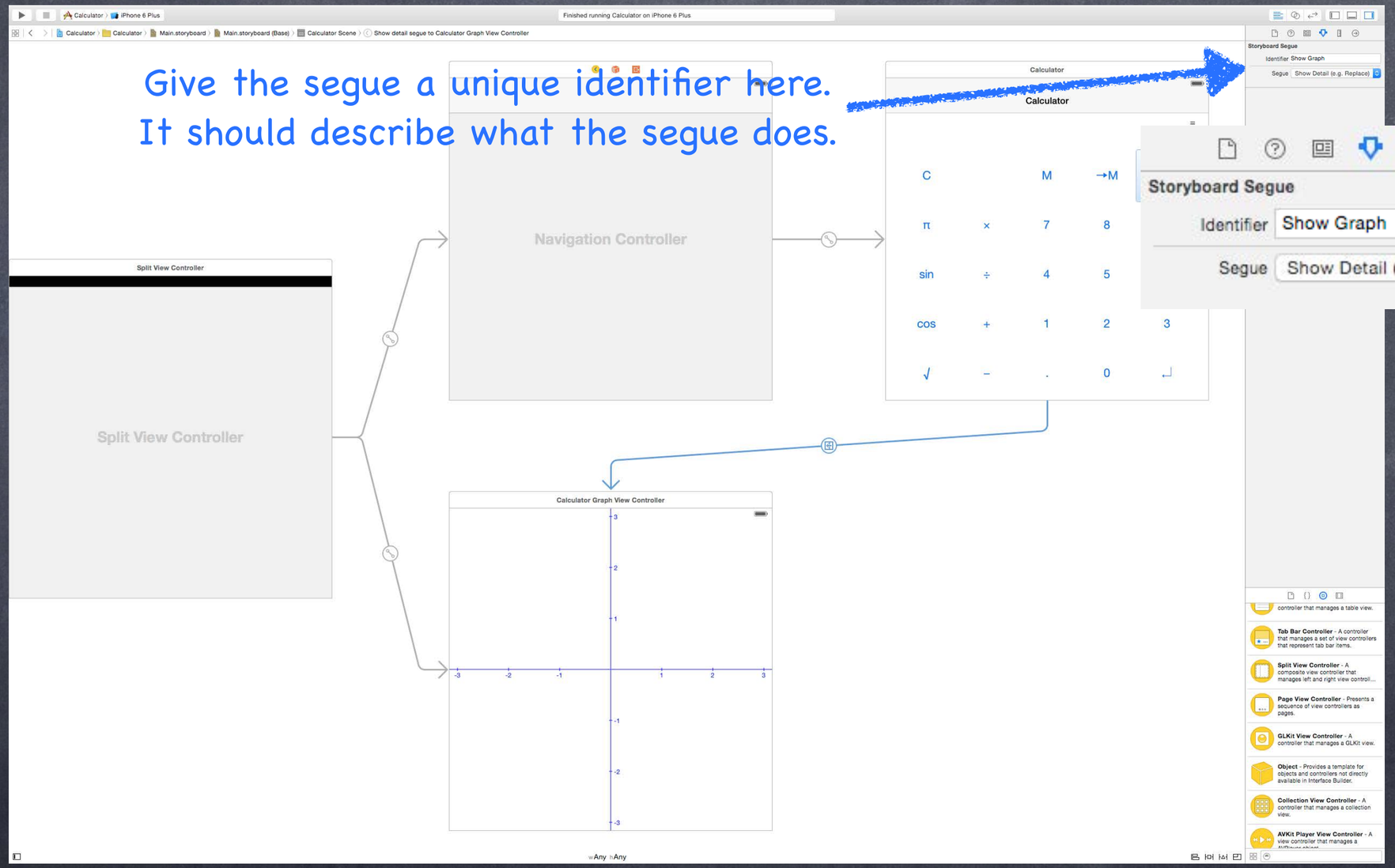
Segues


Navigation Controller
Navigation Controller
Calculator
Split View Controller
Split View Controller
Select the kind of segue you want.
Usually Show or Show Detail.
Calculator Graph View Controller
Action Segue
show
show detail
present modally
popover presentation
custom
Non-Adaptive Action Segue
push (deprecated)
modal (deprecated)

Segues


Now click on the segue
and open the Attributes Inspector
Split View Controller
Navigation Controller
Calculator
Calculator Graph View Controller
Storyboard Segue
Identifier
Segue
Show Detail (e.g. Replace)

Segues


Give the segue a unique identifier here.
It should describe what the segue does.
Split View Controller
Navigation Controller
Calculator
Calculator Graph View Controller
Storyboard Segue
Identifier Show Graph
Segue Show Detail (e.g. Replace)

Segues

- What's that identifier all about?
  - You would need it to invoke this segue from code using this `UIViewController` method
  - `func performSegueWithIdentifier(identifier: String, sender: AnyObject?)`
  - (but we almost never do this because we set usually ctrl-drag from the instigator)
  - The `sender` can be whatever you want (you'll see where it shows up in a moment)
  - You can ctrl-drag from the Controller itself to another Controller if you're segueing via code
  - (because in that case, you'll be specifying the `sender` above)
- More important use of the identifier: preparing for a segue
  - When a segue happens, the View Controller containing the instigator gets a chance to prepare the destination View Controller to be segued to
  - Usually this means setting up the segued-to MVC's Model and display characteristics
  - Remember that the MVC segued to is always a fresh instance (never a reused one)

Preparing for a Segue

- The method that is called in the instigator's Controller

```
func prepareForSegue(segue: UIStoryboardSegue, sender: AnyObject?) {
    if let identifier = segue.identifier {
        switch identifier {
            case “Show Graph”:
                if let vc = segue.destinationViewController as? MyController {
                    vc.property1 = …
                    vc.callMethodToSetItUp(…)
                }
            default: break
        }
    }
}
```

Preparing for a Segue

- The method that is called in the instigator's Controller

```
func prepareForSegue(segue: UIStoryboardSegue, sender: AnyObject?) {
    if let identifier = segue.identifier {
        switch identifier {
            case “Show Graph”:
                if let vc = segue.destinationViewController as? MyController {
                    vc.property1 = …
                    vc.callMethodToSetItUp(…)
                }
            default: break
        }
    }
}
```

The segue passed in contains important information about this segue:

1. the identifier from the storyboard
2. the Controller of the MVC you are segueing to (which was just created for you)

Preparing for a Segue

- The method that is called in the instigator's Controller

```
func prepareForSegue(segue: UIStoryboardSegue, sender: AnyObject?) {
    if let identifier = segue.identifier {
        switch identifier {
            case “Show Graph”:
                if let vc = segue.destinationViewController as? MyController {
                    vc.property1 = …
                    vc.callMethodToSetItUp(…)
                }
            default: break
        }
    }
}
```

The sender is either the instigating object from a storyboard (e.g. a UIButton)
or the sender you provided (see last slide) if you invoked the segue manually in code

Preparing for a Segue

- The method that is called in the instigator's Controller

```
func prepareForSegue(segue: UIStoryboardSegue, sender: AnyObject?) {
    if let identifier = segue.identifier {
        switch identifier {
            case “Show Graph”:
                if let vc = segue.destinationViewController as? MyController {
                    vc.property1 = …
                    vc.callMethodToSetItUp(…)
                }
            default: break
        }
    }
}
```

Here is the identifier from the storyboard (it can be `nil`, so be sure to check for that case)

Your Controller might support preparing for lots of different segues from different instigators so this identifier is how you'll know which one you're preparing for

Preparing for a Segue

- The method that is called in the instigator's Controller

```
func prepareForSegue(segue: UIStoryboardSegue, sender: AnyObject?) {
    if let identifier = segue.identifier {
        switch identifier {
            case “Show Graph”:
                if let vc = segue.destinationViewController as? CalcGraphController {
                    vc.property1 = …
                    vc.callMethodToSetItUp(…)
                }
            default: break
        }
    }
}
```

For this example, we'll assume we entered "Show Graph" in the Attributes Inspector when we had the segue selected in the storyboard

Preparing for a Segue

- The method that is called in the instigator's Controller

```
func prepareForSegue(segue: UIStoryboardSegue, sender: AnyObject?) {
    if let identifier = segue.identifier {
        switch identifier {
            case “Show Graph”:
                if let vc = segue.destinationViewController as? CalcGraphController {
                    vc.property1 = …
                    vc.callMethodToSetItUp(…)
                }
            default: break
        }
    }
}
```

Here we are looking at the Controller of the MVC we're segueing to
It is `AnyObject`, so we must cast it to the Controller we (should) know it to be

Preparing for a Segue

- The method that is called in the instigator's Controller

```
func prepareForSegue(segue: UIStoryboardSegue, sender: AnyObject?) {
    if let identifier = segue.identifier {
        switch identifier {
            case “Show Graph”:
                if let vc = segue.destinationViewController as? CalcGraphController {
                    vc.property1 = …
                    vc.callMethodToSetItUp(…)
                }
            default: break
        }
    }
}
```

This is where the actual preparation of the segued-to MVC occurs
Hopefully the MVC has a clear public API that it wants you to use to prepare it
Once the MVC is prepared, it should run on its own power (only using delegation to talk back)

Preparing for a Segue

- The method that is called in the instigator's Controller

```
func prepareForSegue(segue: UIStoryboardSegue, sender: AnyObject?) {
    if let identifier = segue.identifier {
        switch identifier {
            case “Show Graph”:
                if let vc = segue.destinationViewController as? CalcGraphController {
                    vc.property1 = …
                    vc.callMethodToSetItUp(…)
                }
            default: break
        }
    }
}
```

It is crucial to understand that this preparation is happening BEFORE outlets get set!

It is a very common bug to prepare an MVC thinking its outlets are set.

Preventing Segues

- You can prevent a segue from happening too
  - Just implement this in your `UIViewController` …

```
func shouldPerformSegueWithIdentifier(identifier: String?, sender: AnyObject?) -> Bool
```

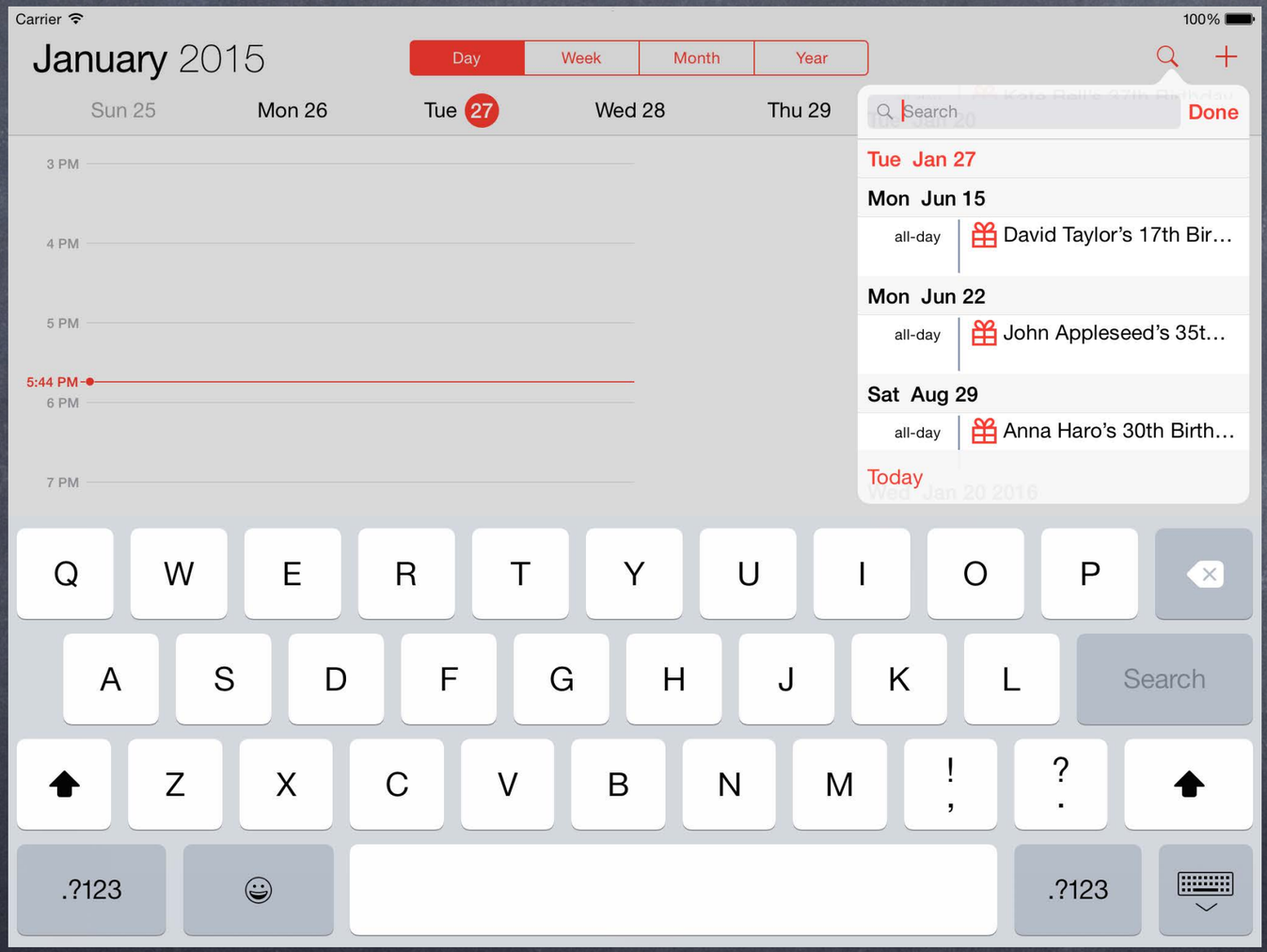
  - The `identifier` is the one in the storyboard.
  - The `sender` is the instigating object (e.g. the button that is causing the segue).

Demo

- Psychologist
  - This is all best understood via demonstration
  - We will create a new Psychologist MVC
  - The Psychologist will reveal his diagnosis by segueing to the Happiness MVC
  - We'll put the MVCs into navigation controllers inside split view controllers
  - That way, it will work on both iPad and iPhone devices

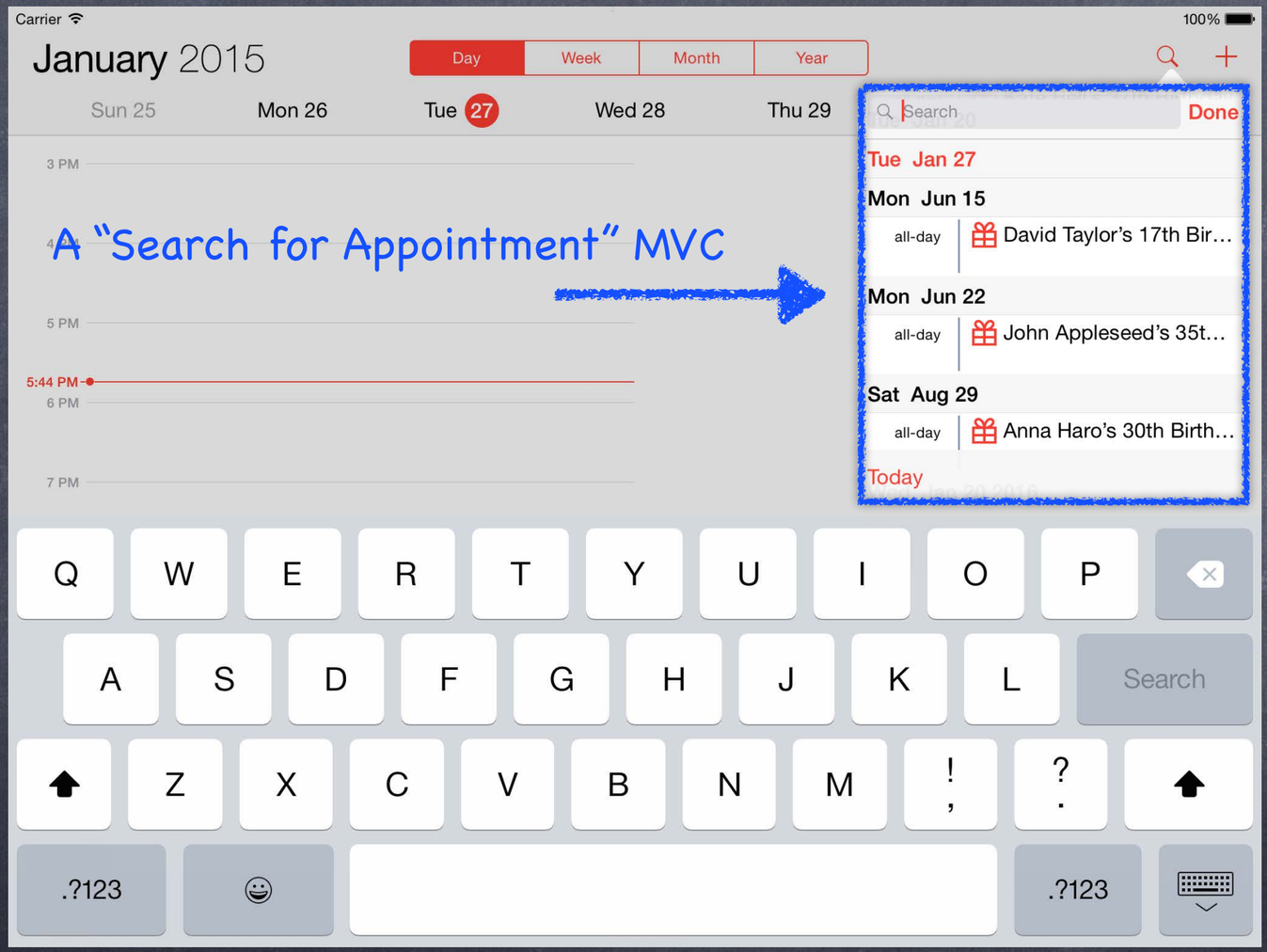
Popover

- Popovers pop an entire MVC over the rest of the screen



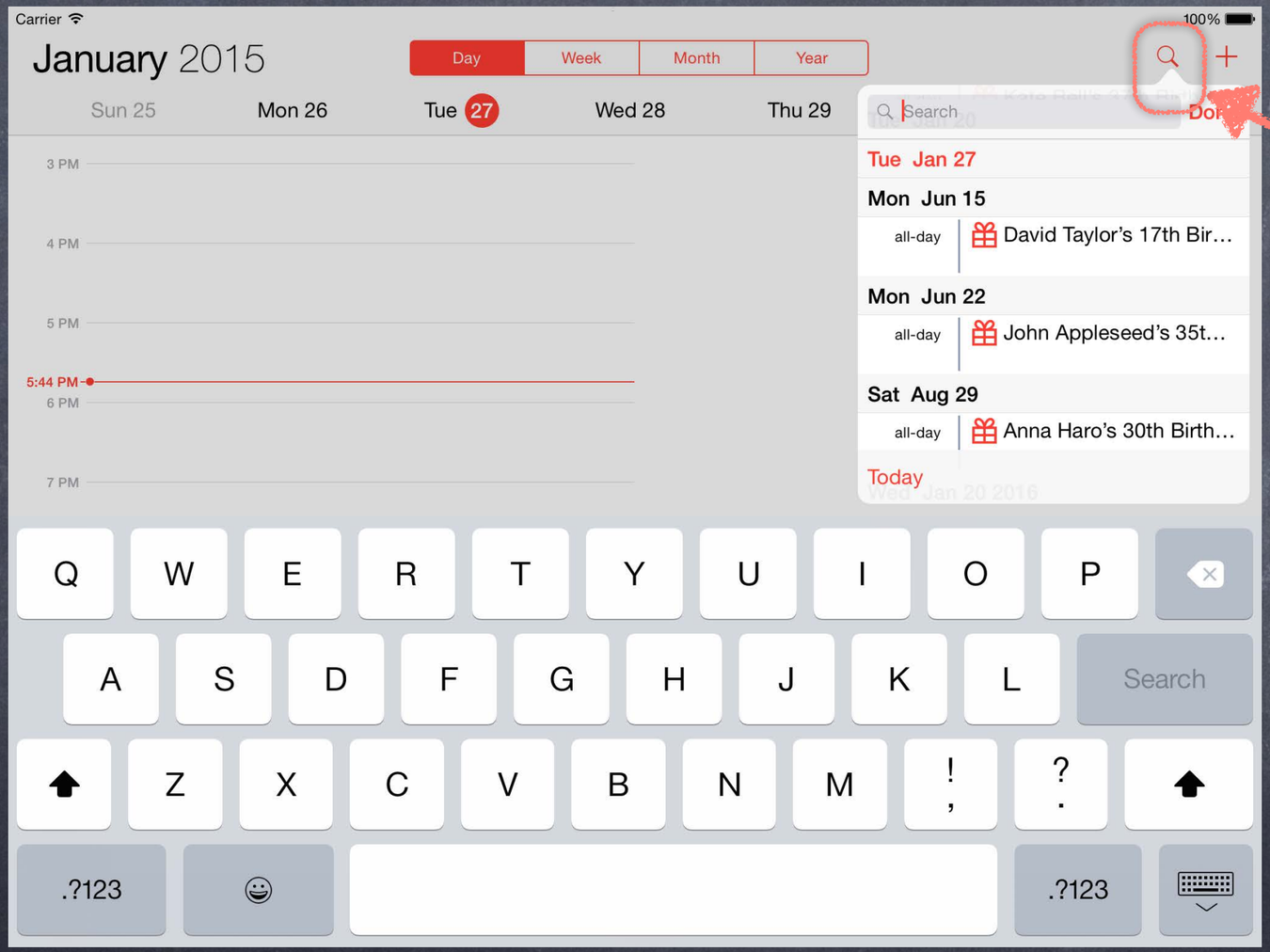
Popover

- Popovers pop an entire MVC over the rest of the screen



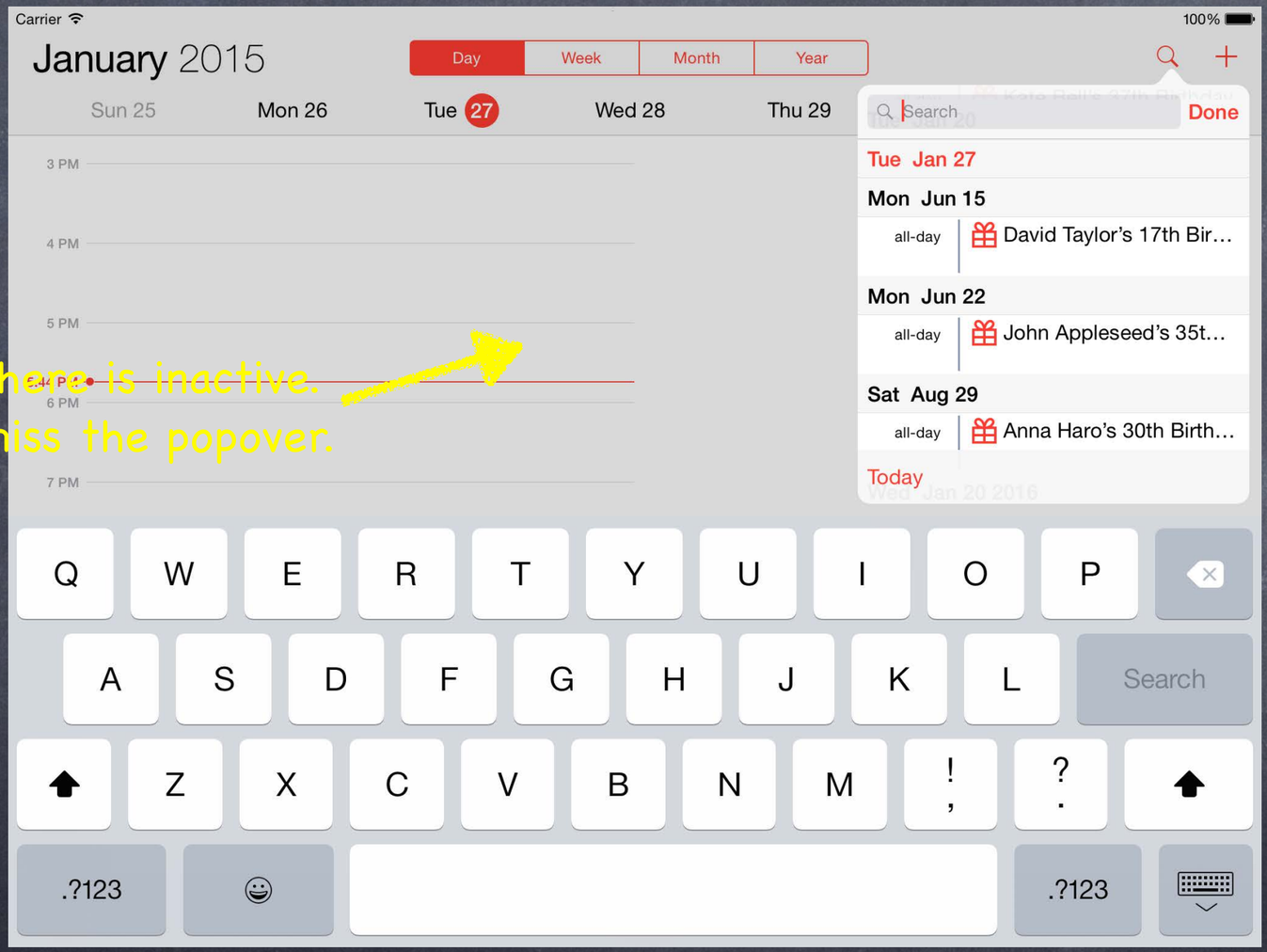
Popover

- Popovers pop an entire MVC over the rest of the screen



Popover

- Popovers pop an entire MVC over the rest of the screen



The grayed out area here is inactive.
Touching in it will dismiss the popover.

Popover

- Popovers are not quite the same as these other combiners
  - Tab Bar, Split View and Navigation Controllers are `UIViewControllers`, popovers are not
- Seguing to a popover works almost exactly the same though
  - You still ctrl-drag, you still have an identifier, you still get to prepare
- Things to note when preparing for a popover segue
  - All segues are managed via a `UIPresentationController` (but we're not going to cover that)
  - But we are going to talk about a popover's `UIPresentationController`
  - It can tell you what caused the popover to appear (a bar button item or just a rectangle)
  - And it can let you control how the popover is presented
  - For example, you can control what direction the popover's arrow is allowed to point
  - Or you can control how a popover adapts to different sizes classes (e.g. iPad vs iPhone)

Popover Prepare

- Here's a prepareForSegue that prepares for a Popover segue

```
func prepareForSegue(segue: UIStoryboardSegue, sender: AnyObject?) {
    if let identifier = segue.identifier {
        switch identifier {
            case "Do Something in a Popover Segue":
                if let vc = segue.destinationViewController as? MyController {
                    if let ppc = vc.popoverPresentationController {
                        ppc.permittedArrowDirections = UIPopoverArrowDirection.Any
                        ppc.delegate = self
                    }
                    // more preparation here
                }
            default: break
        }
    }
}
```

One thing that is different is that we are retrieving the popover's presentation controller

Popover Prepare

- Here's a prepareForSegue that prepares for a Popover segue

```
func prepareForSegue(segue: UIStoryboardSegue, sender: AnyObject?) {
    if let identifier = segue.identifier {
        switch identifier {
            case "Do Something in a Popover Segue":
                if let vc = segue.destinationViewController as? MyController {
                    if let ppc = vc.popoverPresentationController {
                        ppc.permittedArrowDirections = UIPopoverArrowDirection.Any
                        ppc.delegate = self
                    }
                    // more preparation here
                }
            default: break
        }
    }
}
```

We can use it to set some properties that will control how the popover pops up

Popover Prepare

- Here's a prepareForSegue that prepares for a Popover segue

```
func prepareForSegue(segue: UIStoryboardSegue, sender: AnyObject?) {
    if let identifier = segue.identifier {
        switch identifier {
            case "Do Something in a Popover Segue":
                if let vc = segue.destinationViewController as? MyController {
                    if let ppc = vc.popoverPresentationController {
                        ppc.permittedArrowDirections = UIPopoverArrowDirection.Any
                        ppc.delegate = self
                    }
                    // more preparation here
                }
            default: break
        }
    }
}
```

And we can control the presentation by setting ourself (the Controller) as the delegate

Popover Presentation Controller

- What can we control as the presentation controller's delegate?

  One very interesting thing is how a popover "adapts" to different sizes

  By default, it will present on compact sizes Modally (i.e. take over the whole screen)

  But the delegate can control this "adaptation" behavior, either by preventing it …

```
func adaptivePresentationStyleForPresentationController(UIPresentationController)
    -> UIModalPresentationStyle
{
    return UIModalPresentationStyle.None // don't adapt (default is .FullScreen)
}
```

  … or by allowing the full screen presentation to happen, but modifying the MVC that is put up …

```
func presentationController(UIPresentationController,
   viewControllerForAdaptivePresentationStyle: UIModalPresentatinoStyle)
    -> UIViewController?
{
    // return a UIViewController to use (e.g. wrap a Navigation Controller around your MVC)
}
```

Popover Size

- Important Popover Issue: Size
  - A popover will be made pretty large unless someone tells it otherwise.
  - The MVC being presented knows best what it's "preferred" size inside a popover would be.
  - It expresses that via this property in itself (i.e. in the Controller of the MVC being presented) ...
  - `var preferredContentSize: CGSize`
  - The MVC is not guaranteed to be that size, but the system will try its best.

Demo

- Popover in Psychologist

  Add a popover that shows the patient's diagnostic history