



Stanford CS193p

Developing Applications for iOS
Fall 2017-18



CS193p
Fall 2017-18

Today

- 🕒 Multiple MVCs

- Tab Bar, Navigation and Split View Controllers

- Demo: Theme Chooser in Concentration

- 🕒 Timer

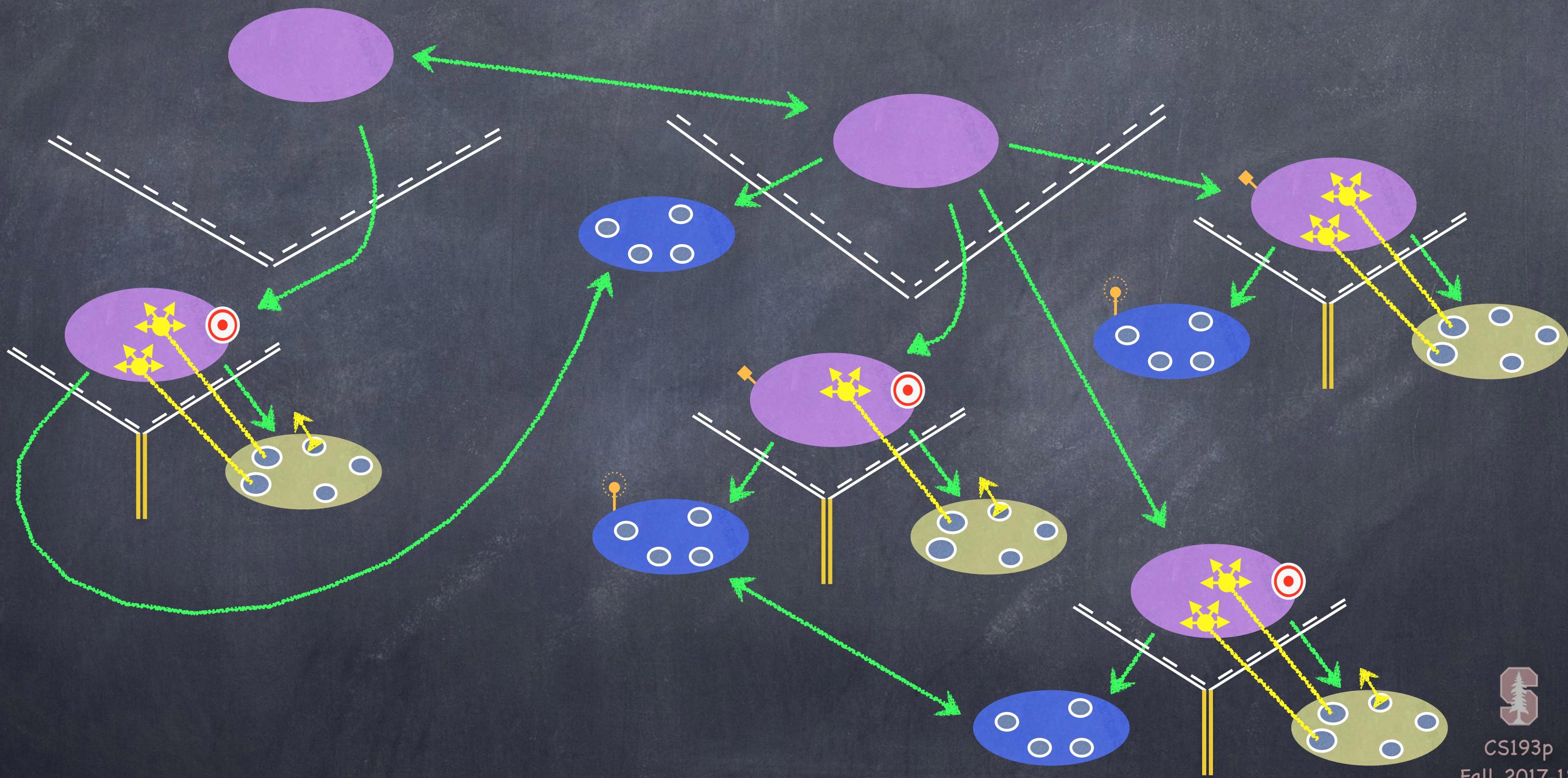
- 🕒 Animation

- UIViewControllerAnimated

- Transitions



MVCs working together

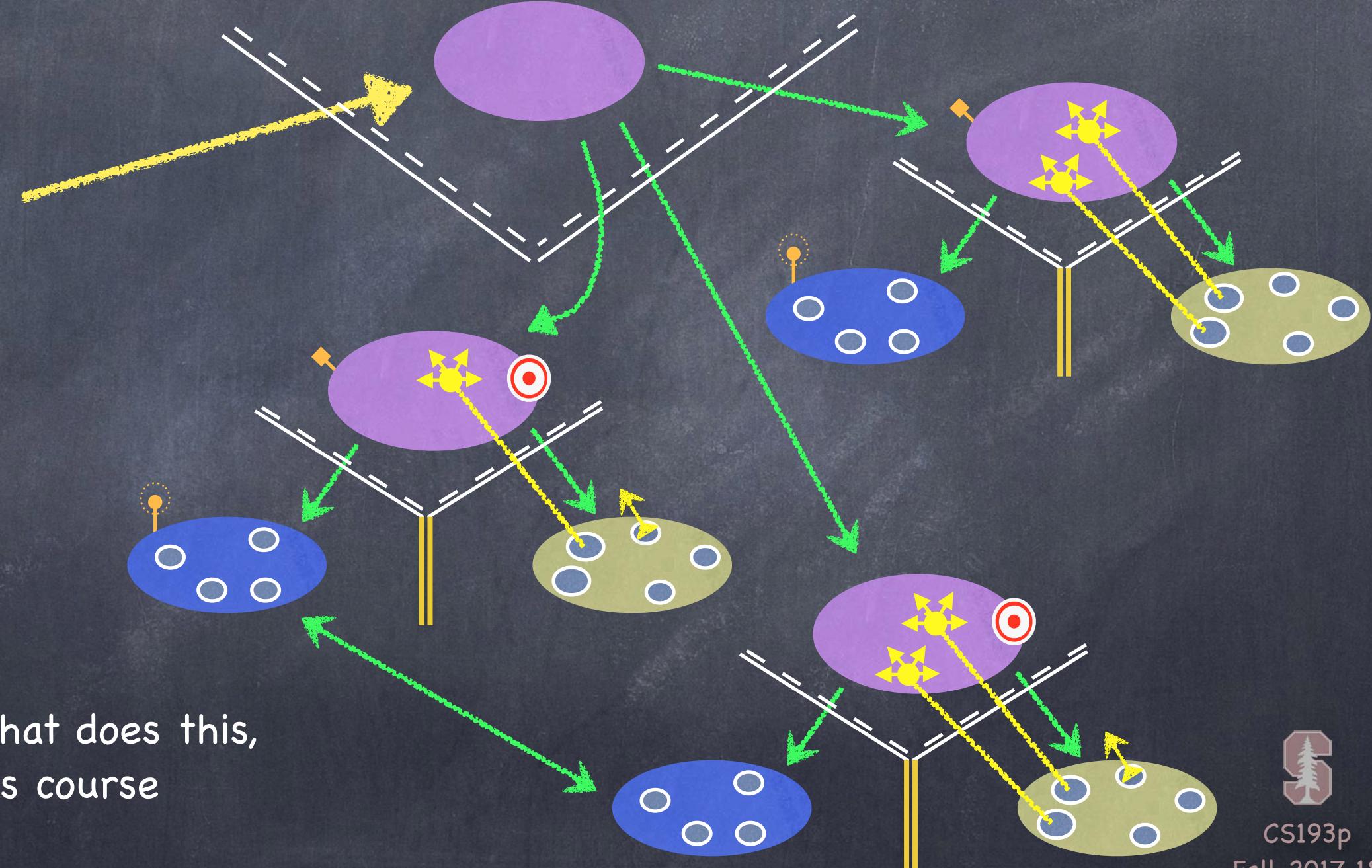


Multiple MVCS

- 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



Multiple MVCS

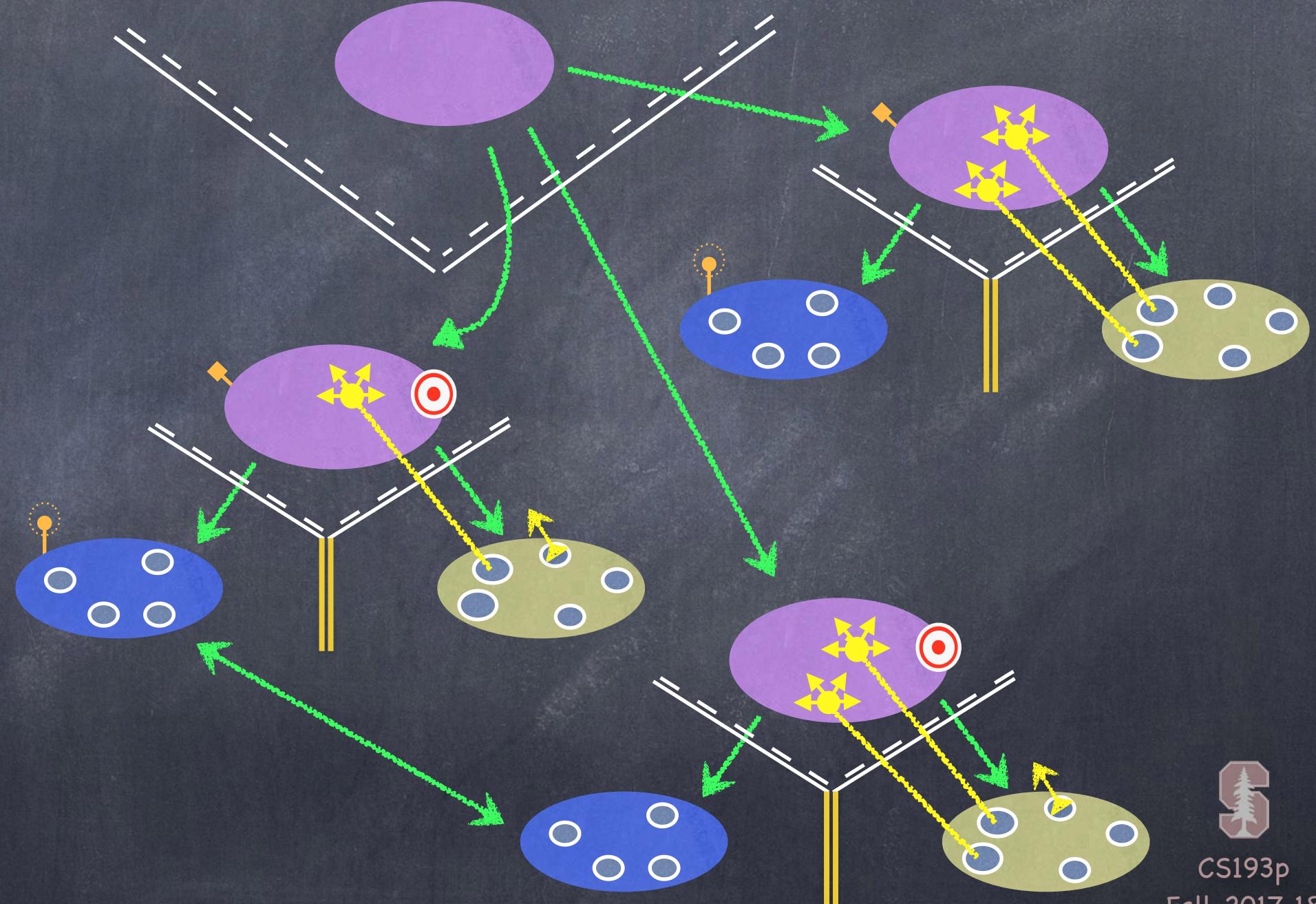
- Time to build more powerful applications

To do this, we must combine MVCS ...

iOS provides some Controllers
whose View is “other MVCS”

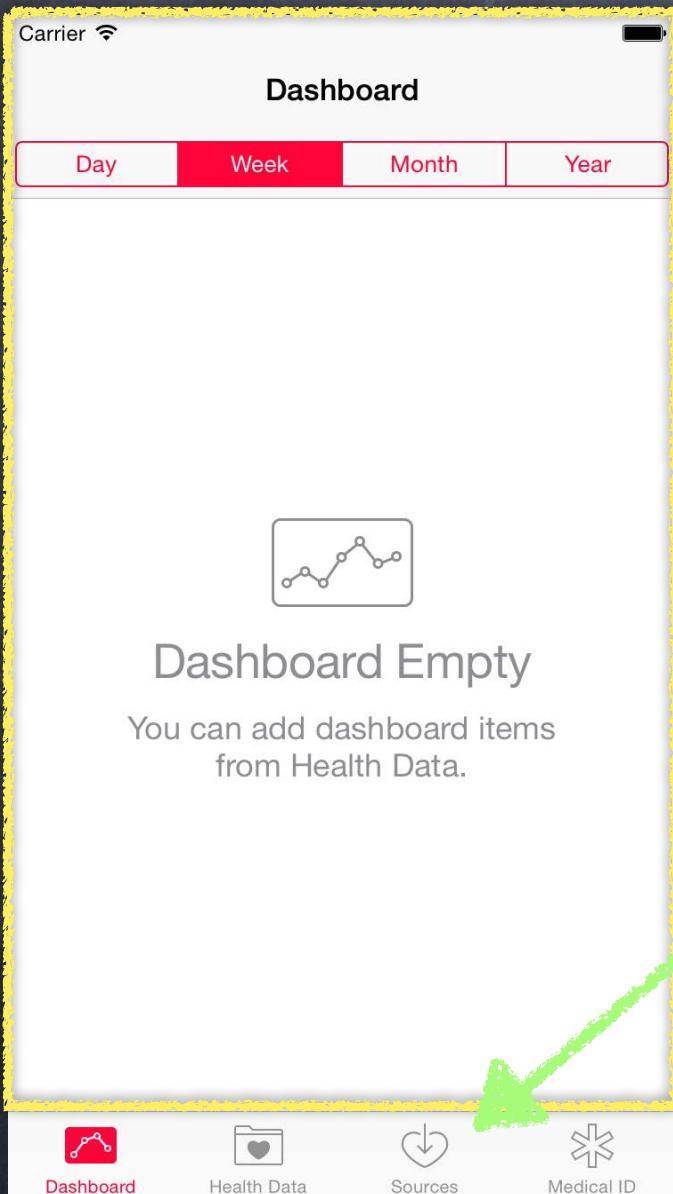
Examples:

`UITabBarController`
`UISplitViewController`
`UINavigationController`



UITabBarController

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



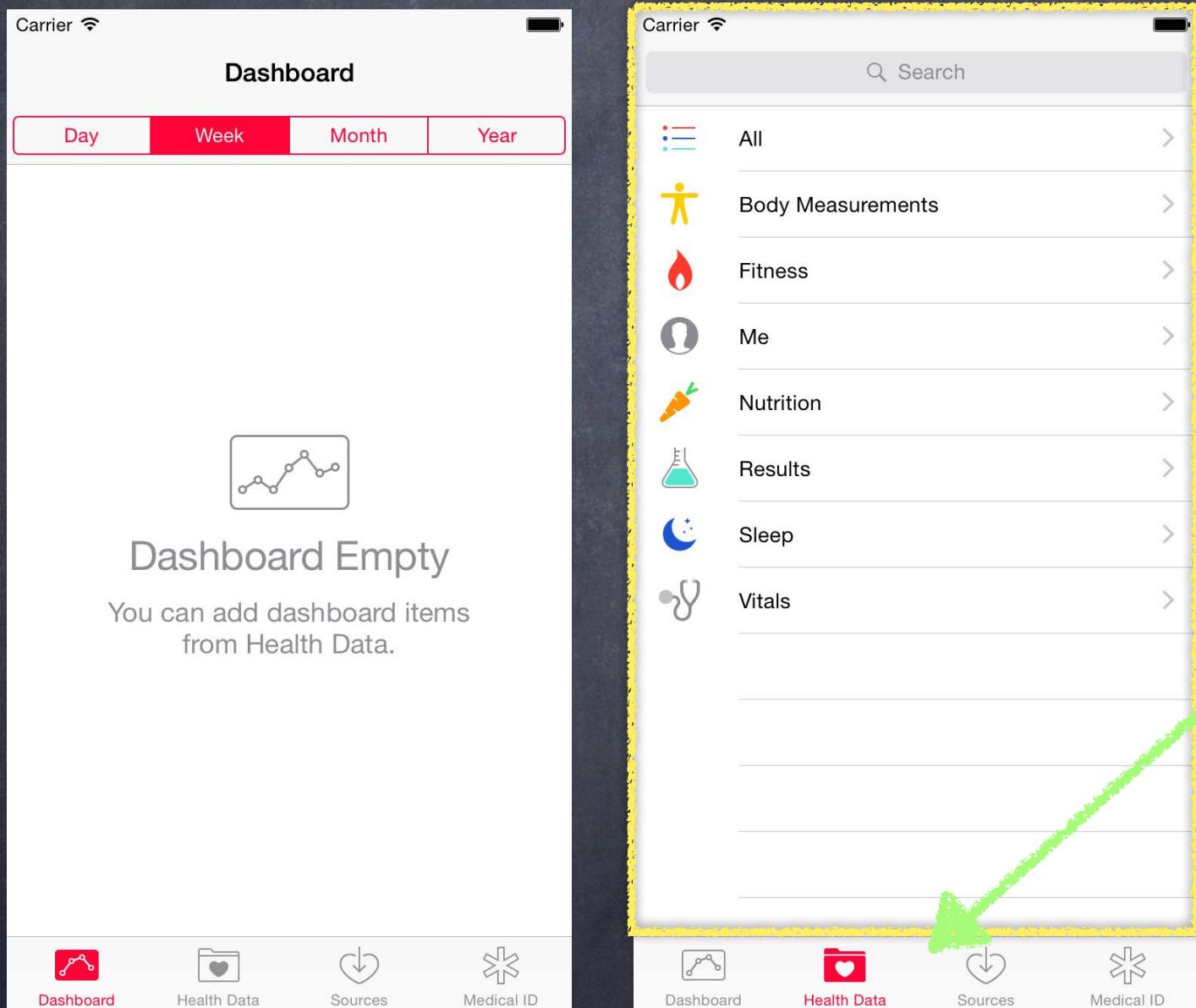
A "Dashboard" MVC

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 ...



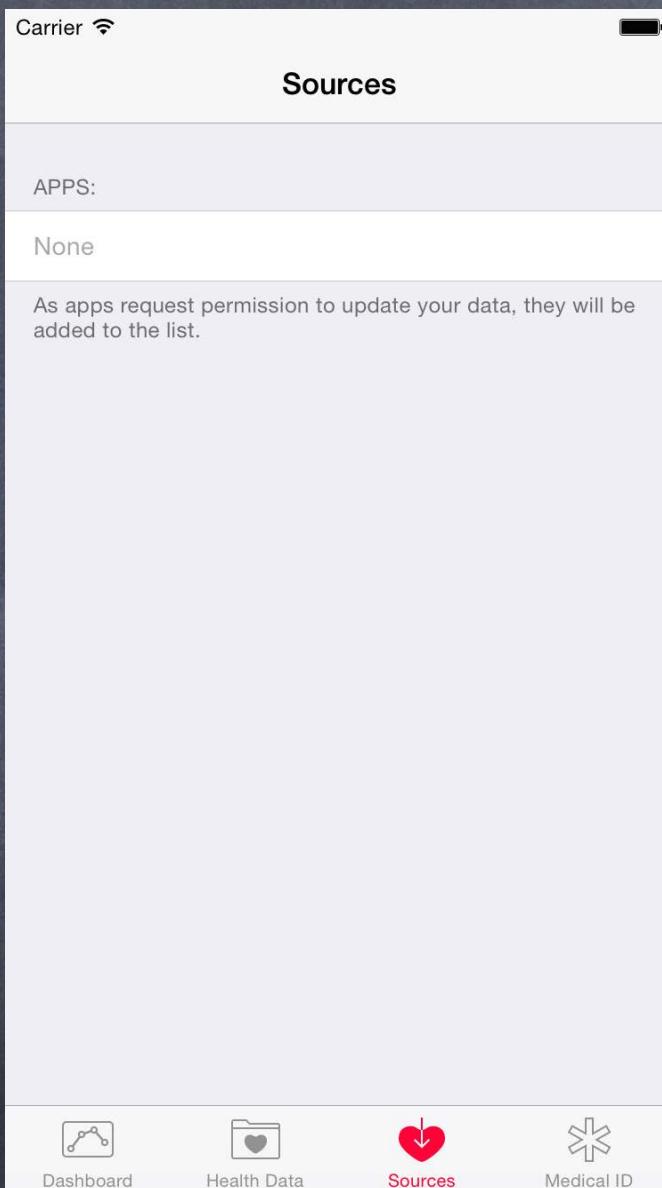
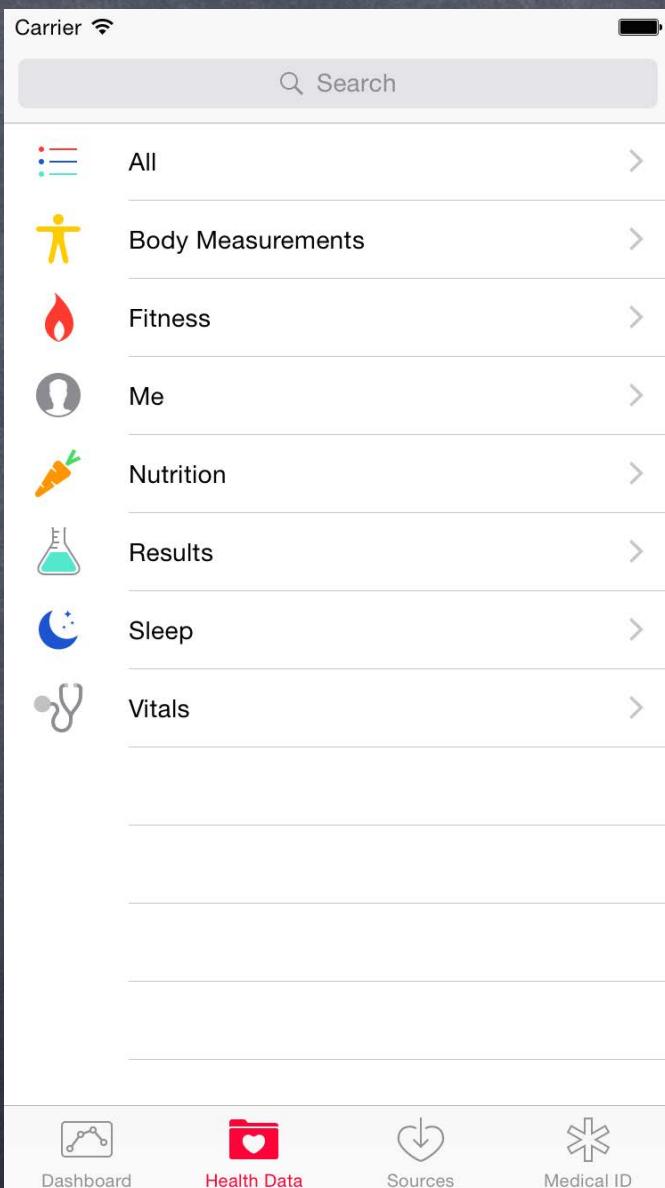
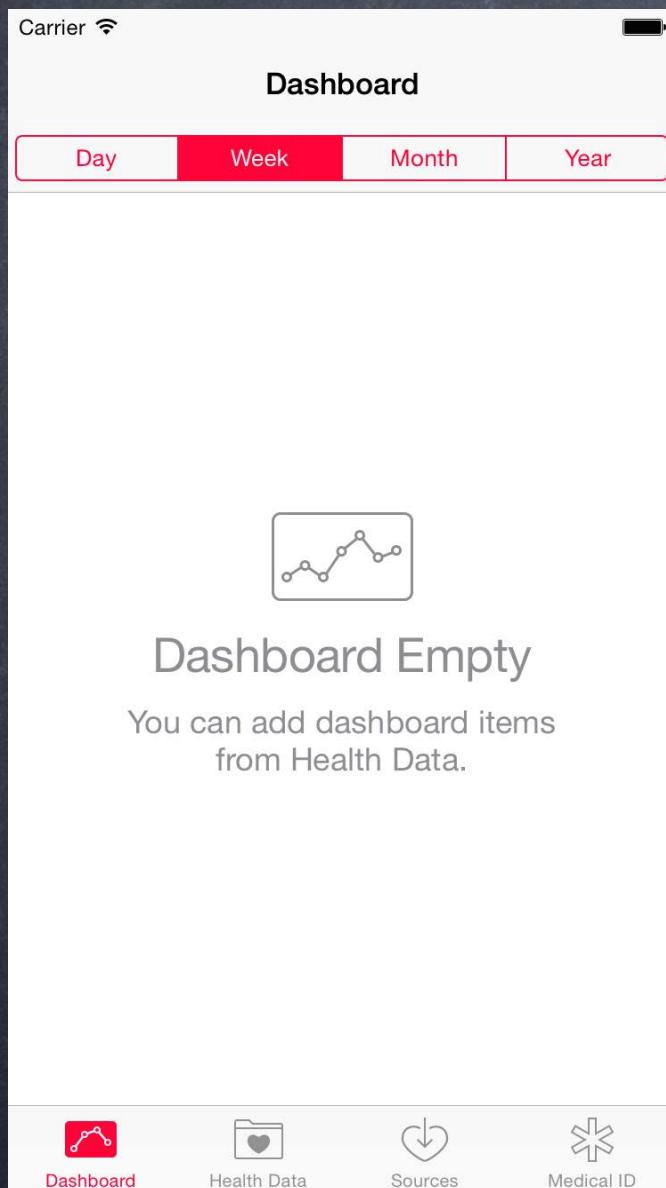
A “Health Data” MVC

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



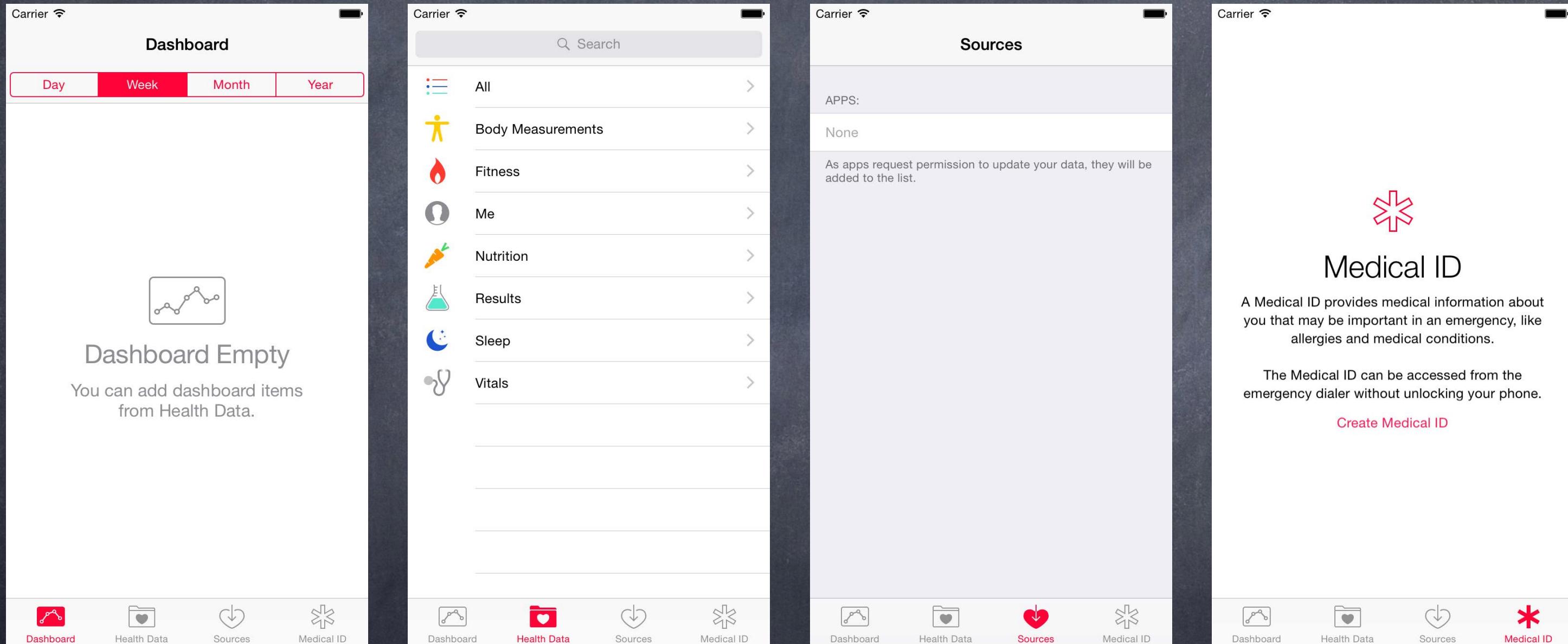
UITabBarController

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



UITabBarController

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



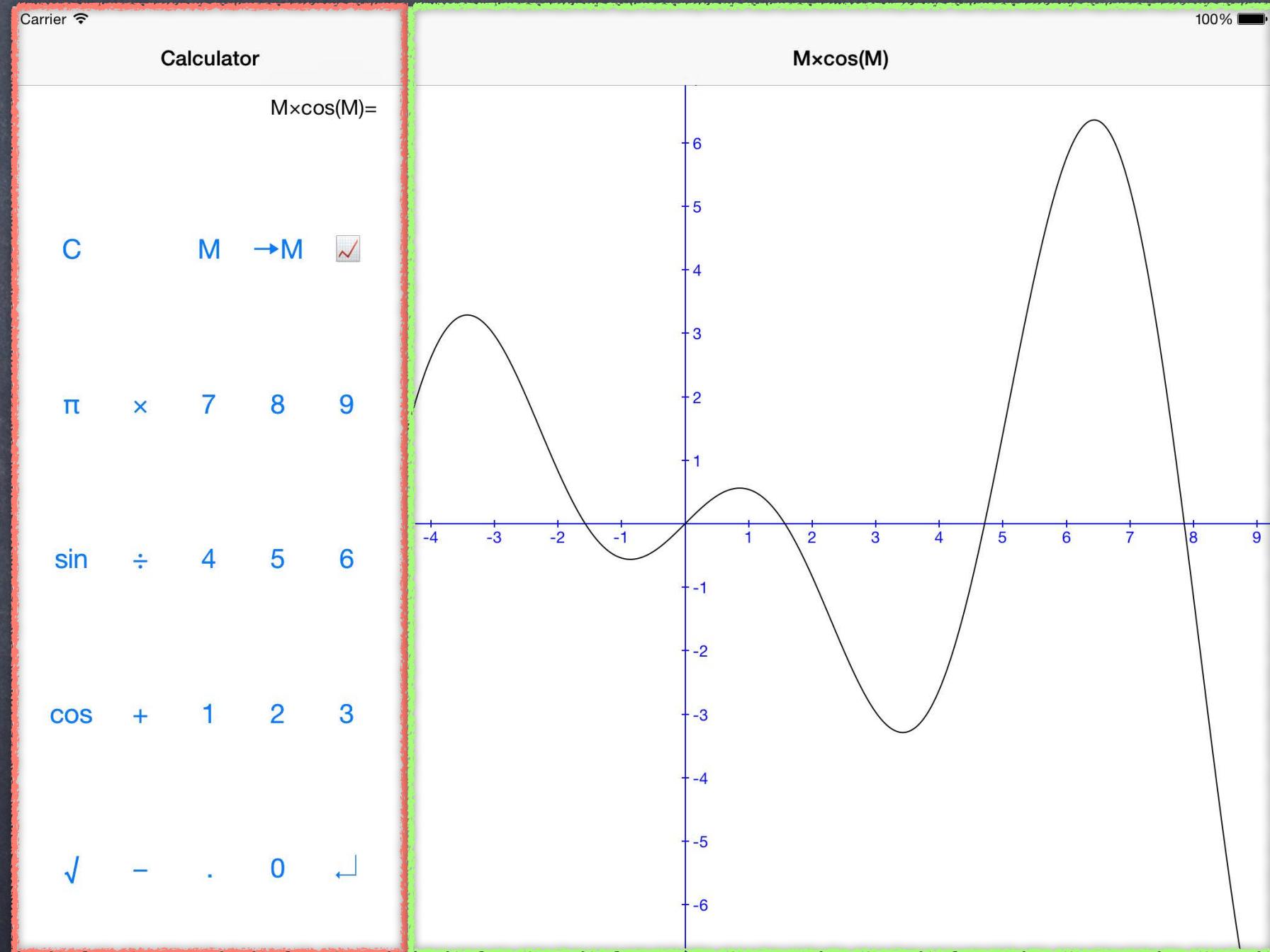
UISplitViewController

- Puts two MVCs side-by-side ...

A
Calculator
MVC



Master



A
Calculator Graph
MVC



Detail



UISplitViewController

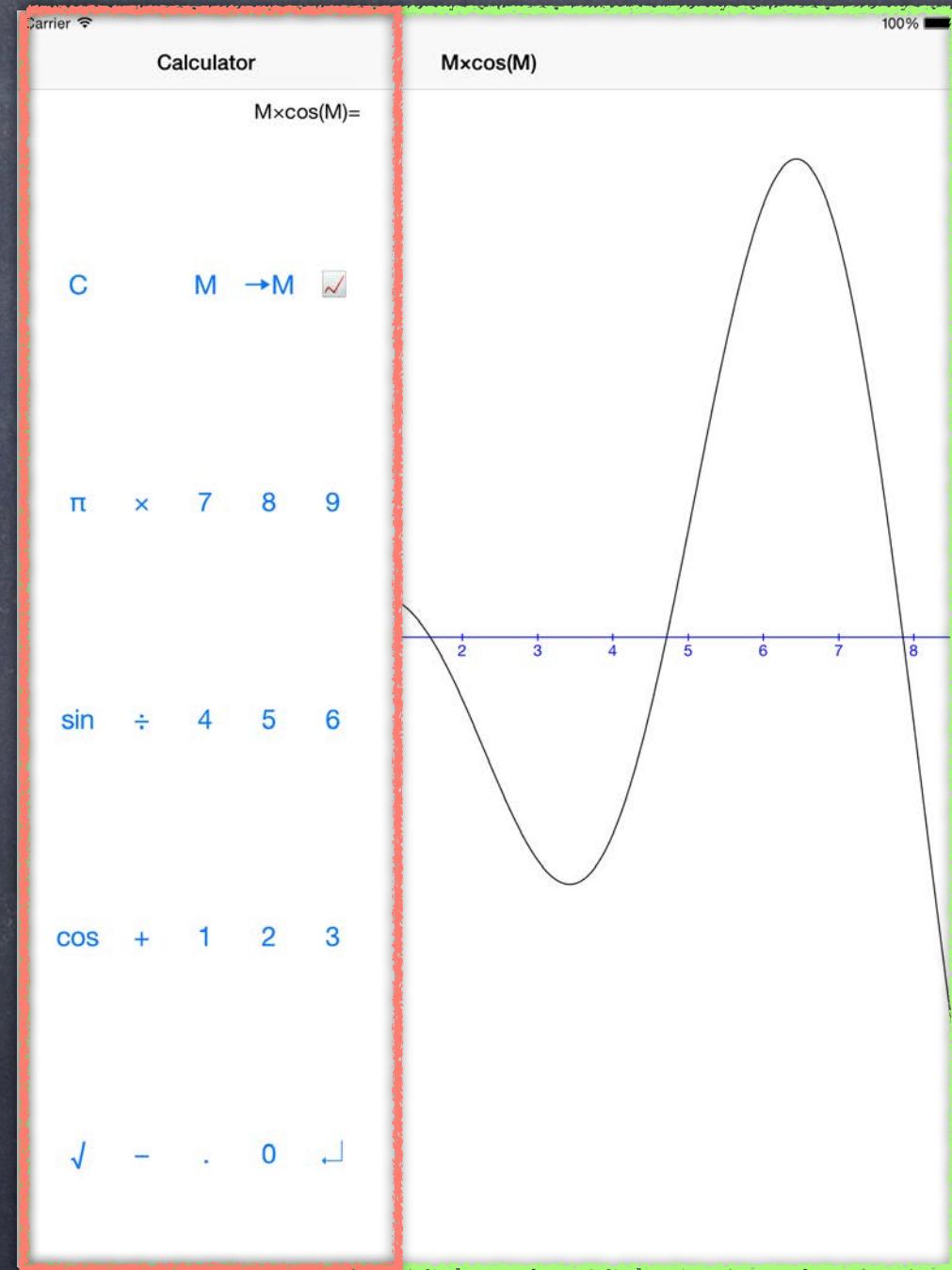
- Puts two MVCs side-by-side ...

A
Calculator
MVC

Master

A
Calculator Graph
MVC

Detail



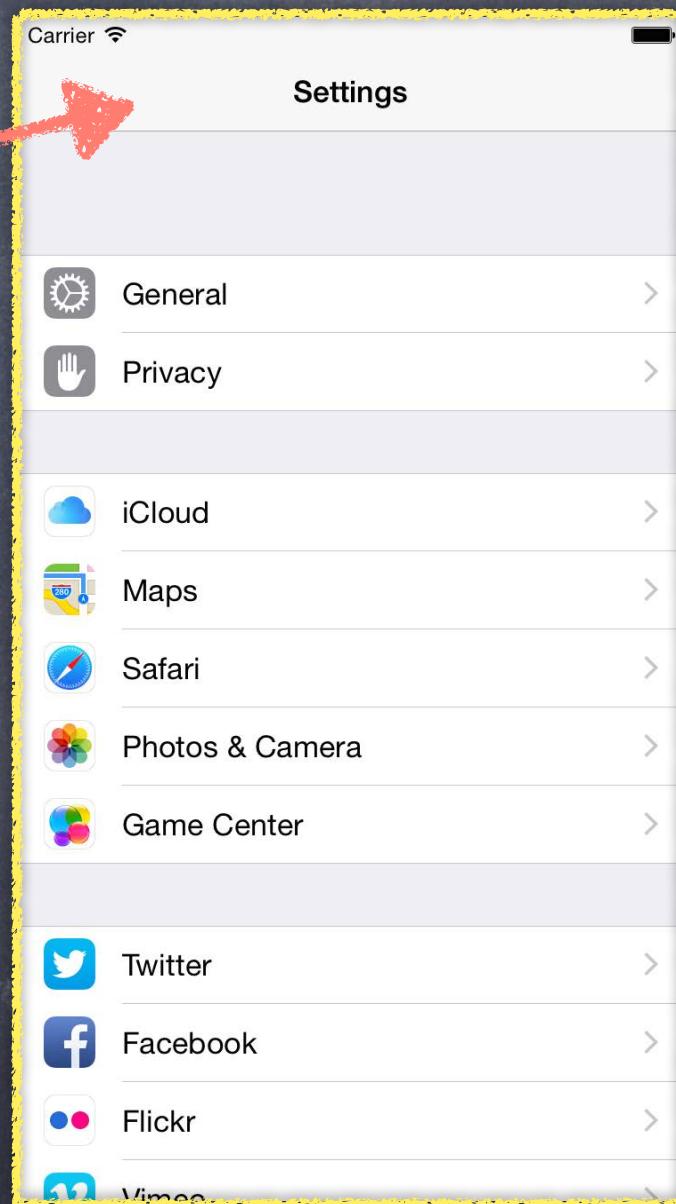
UINavigationController

- 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



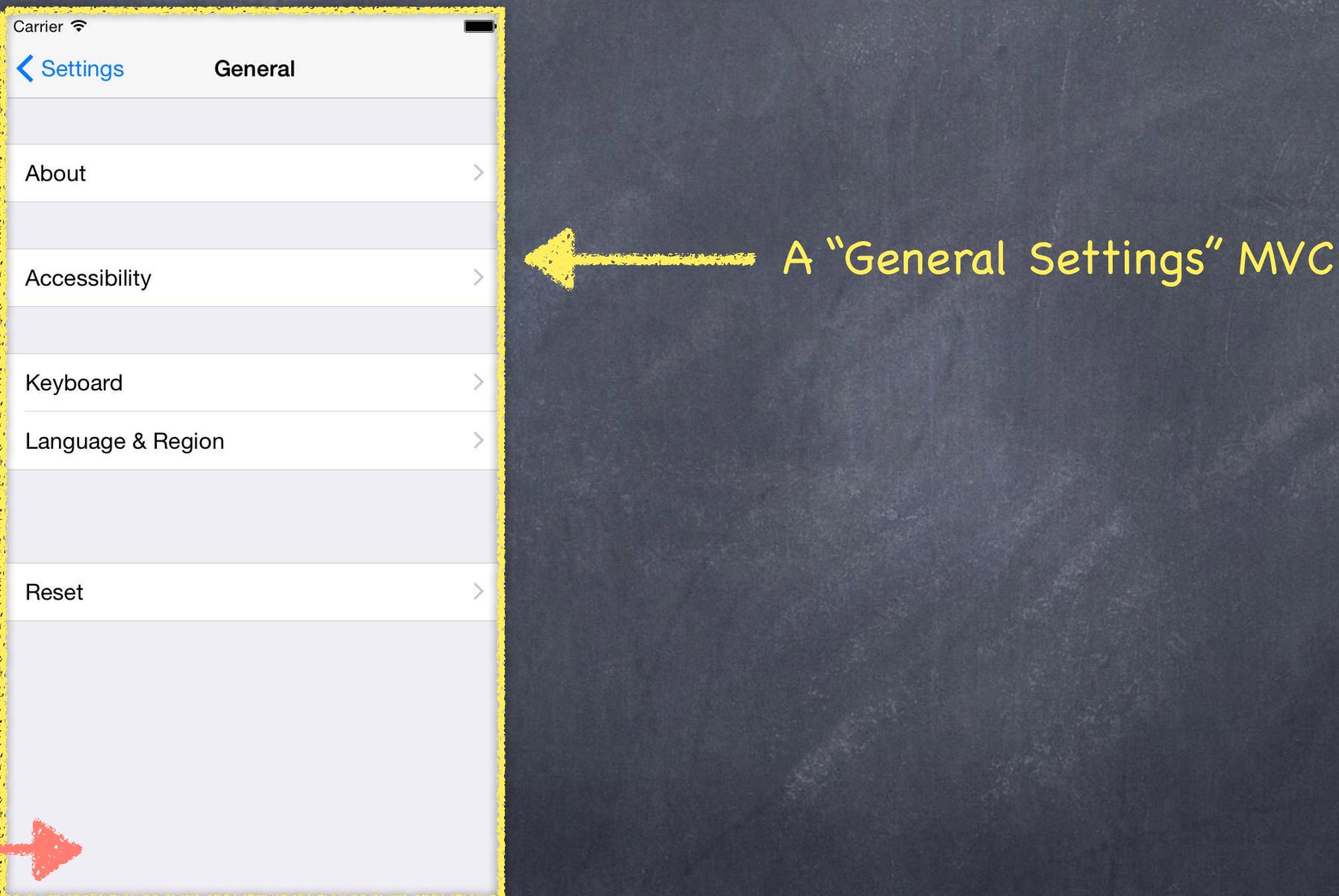
UINavigationController

- 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) ...

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



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) ...

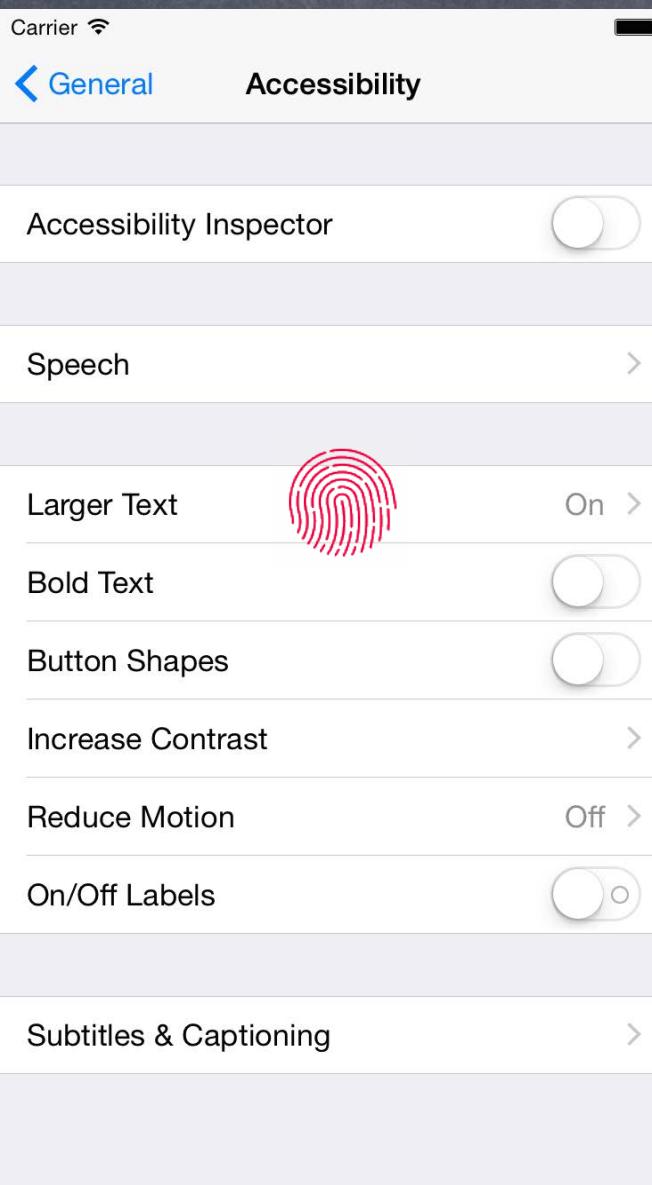


An “Accessibility” MVC



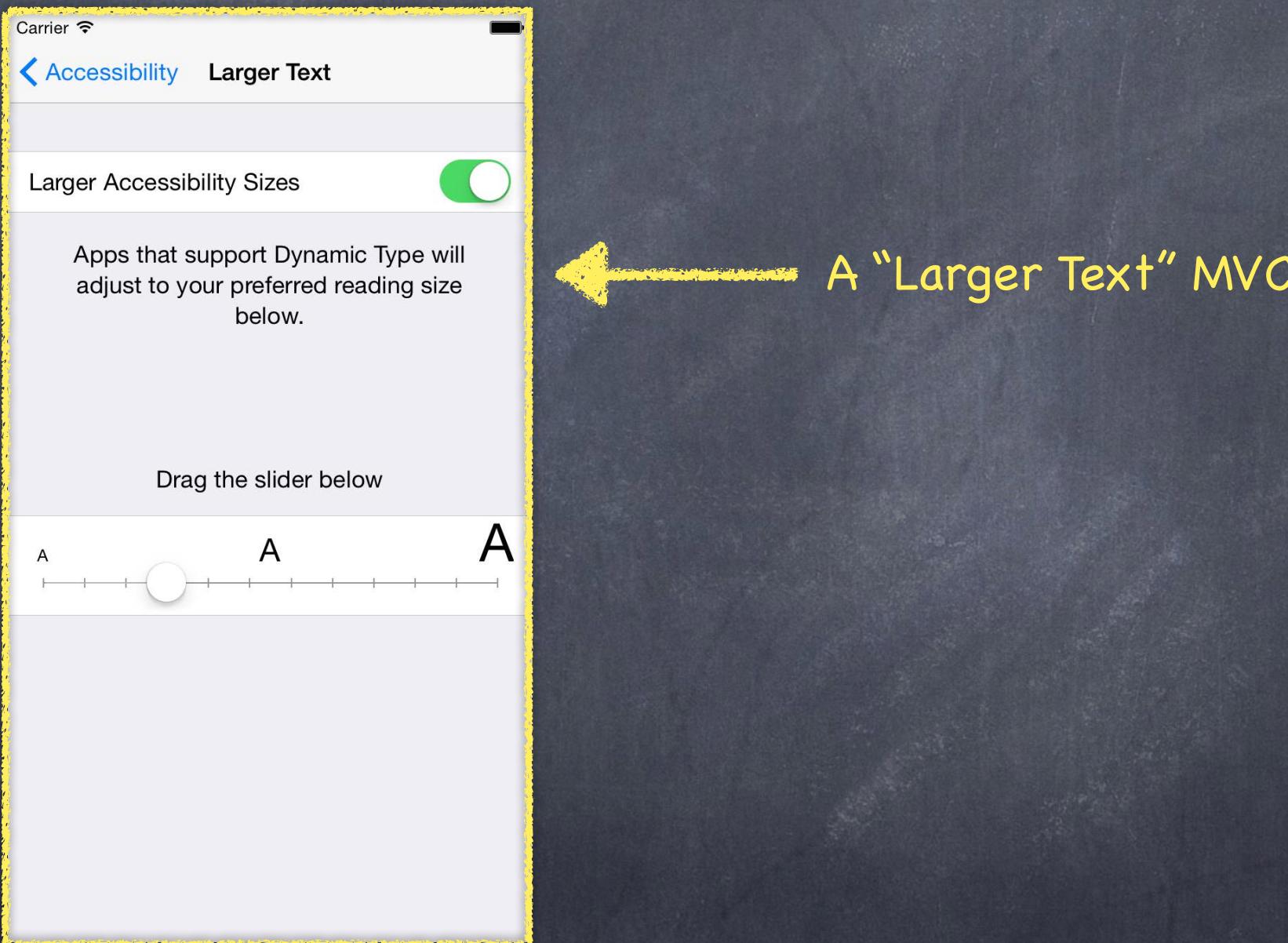
UINavigationController

- 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) ...



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) ...

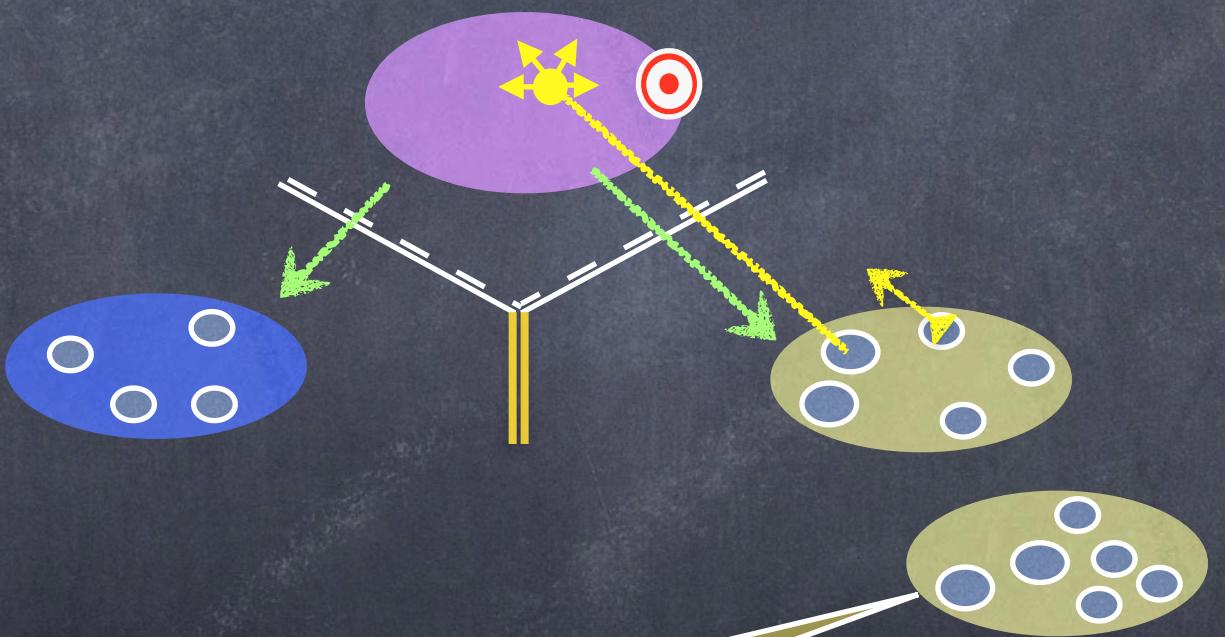


UINavigationController

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



UINavigationController



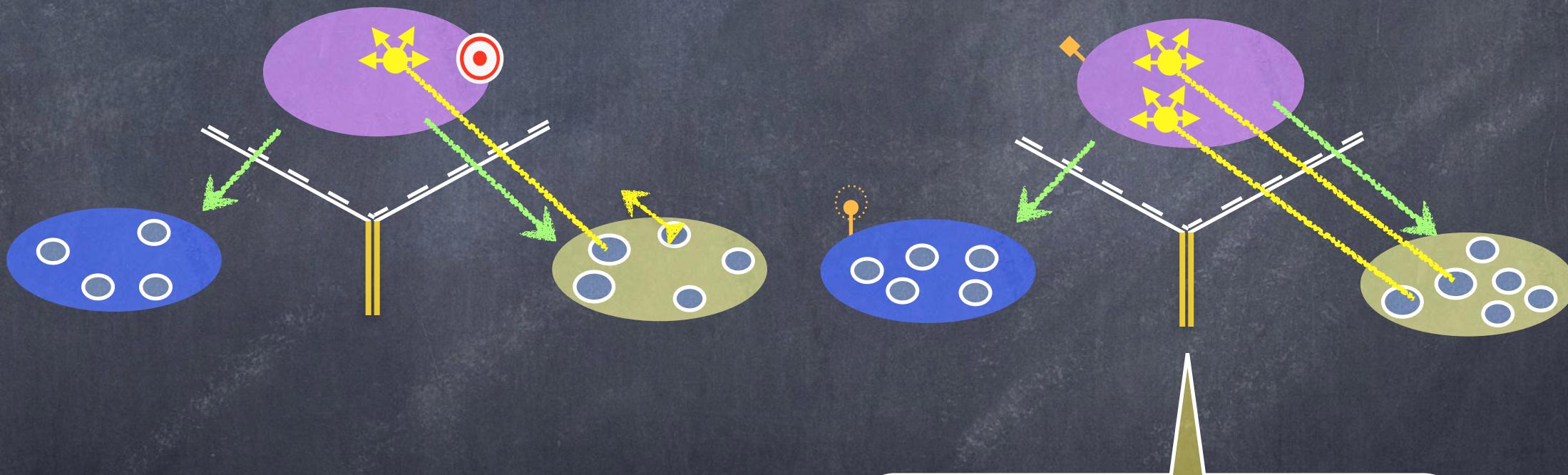
I want more features, but it doesn't make sense to put them all in one MVC!



CS193p

Fall 2017-18

UINavigationController

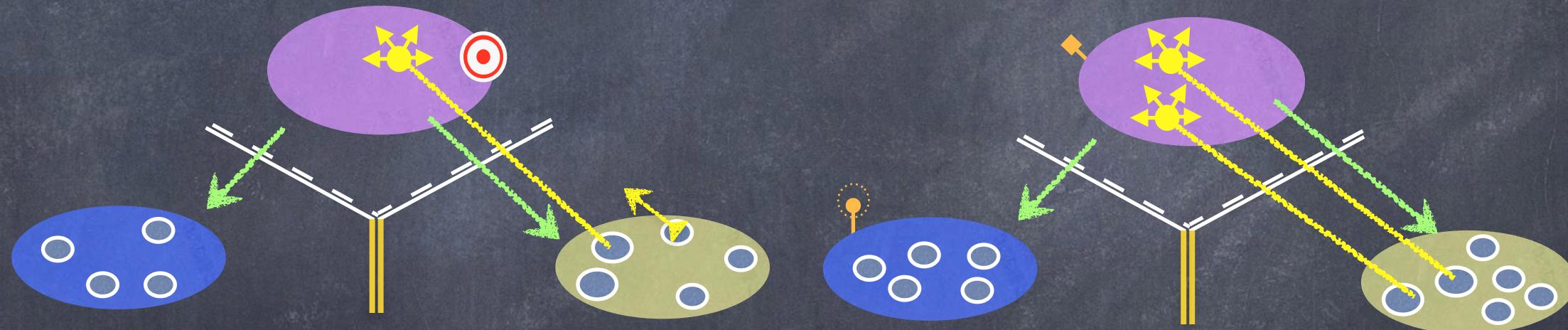


So I create a new MVC to
encapsulate that functionality.

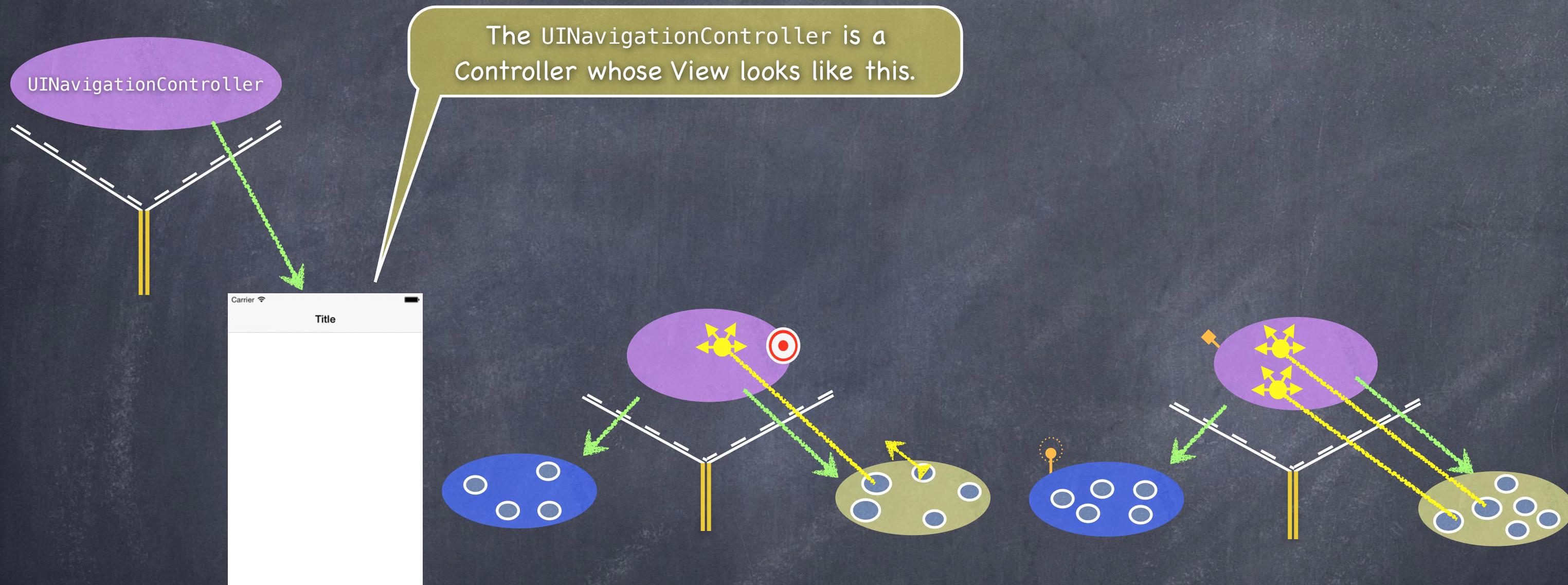


UINavigationController

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



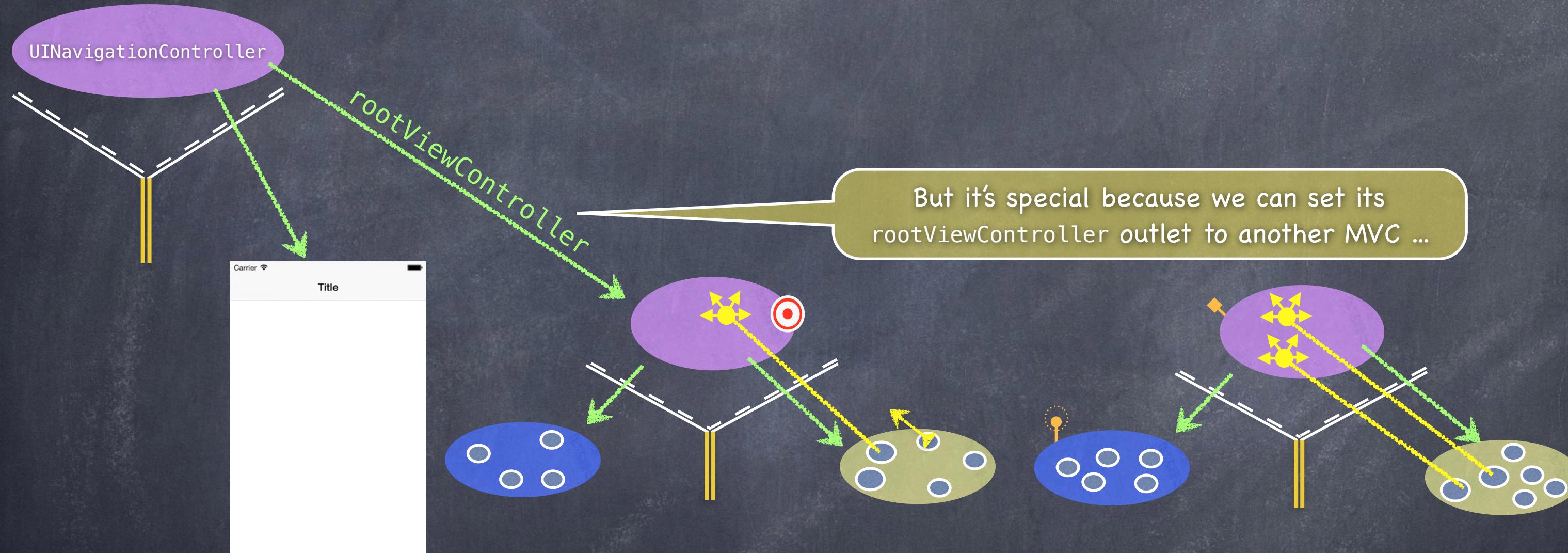
UINavigationController



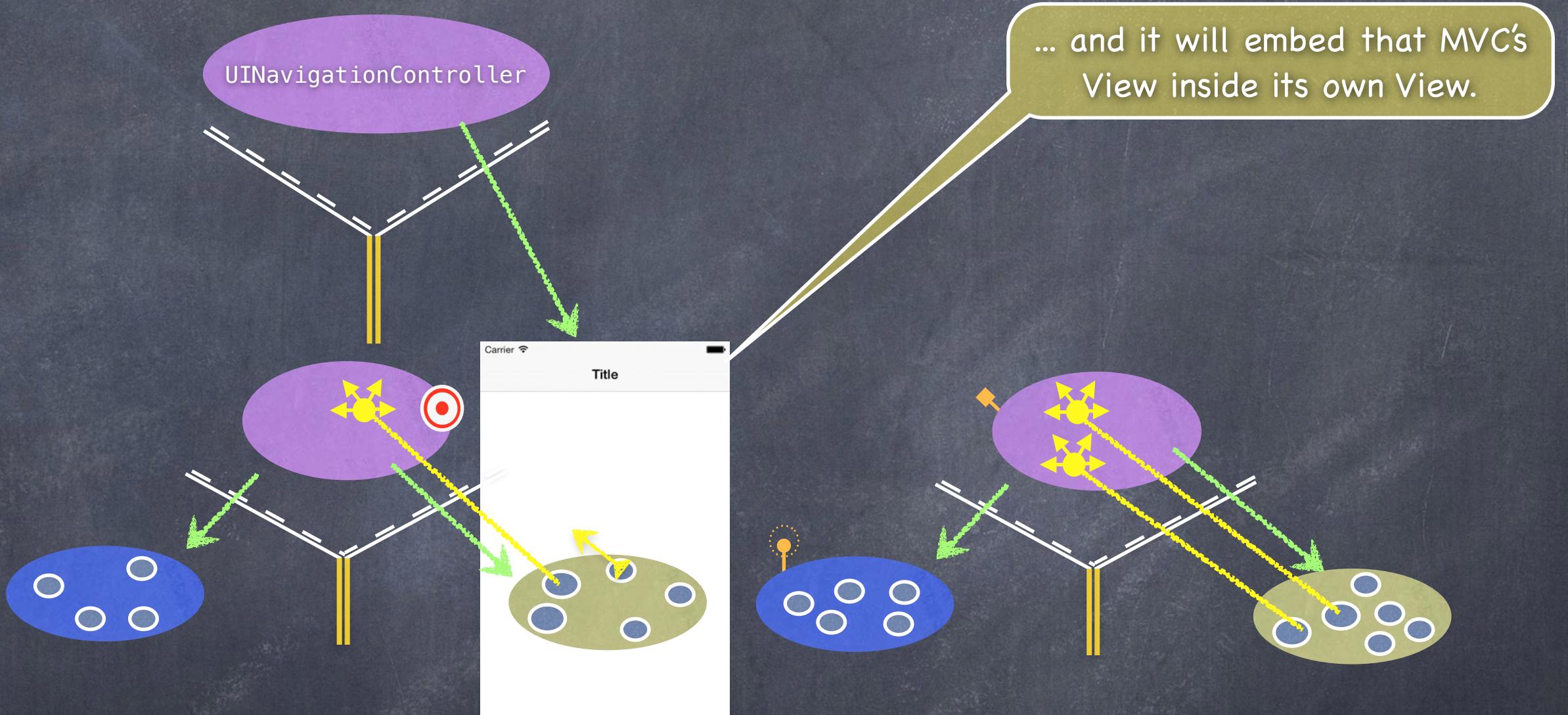
CS193p

Fall 2017-18

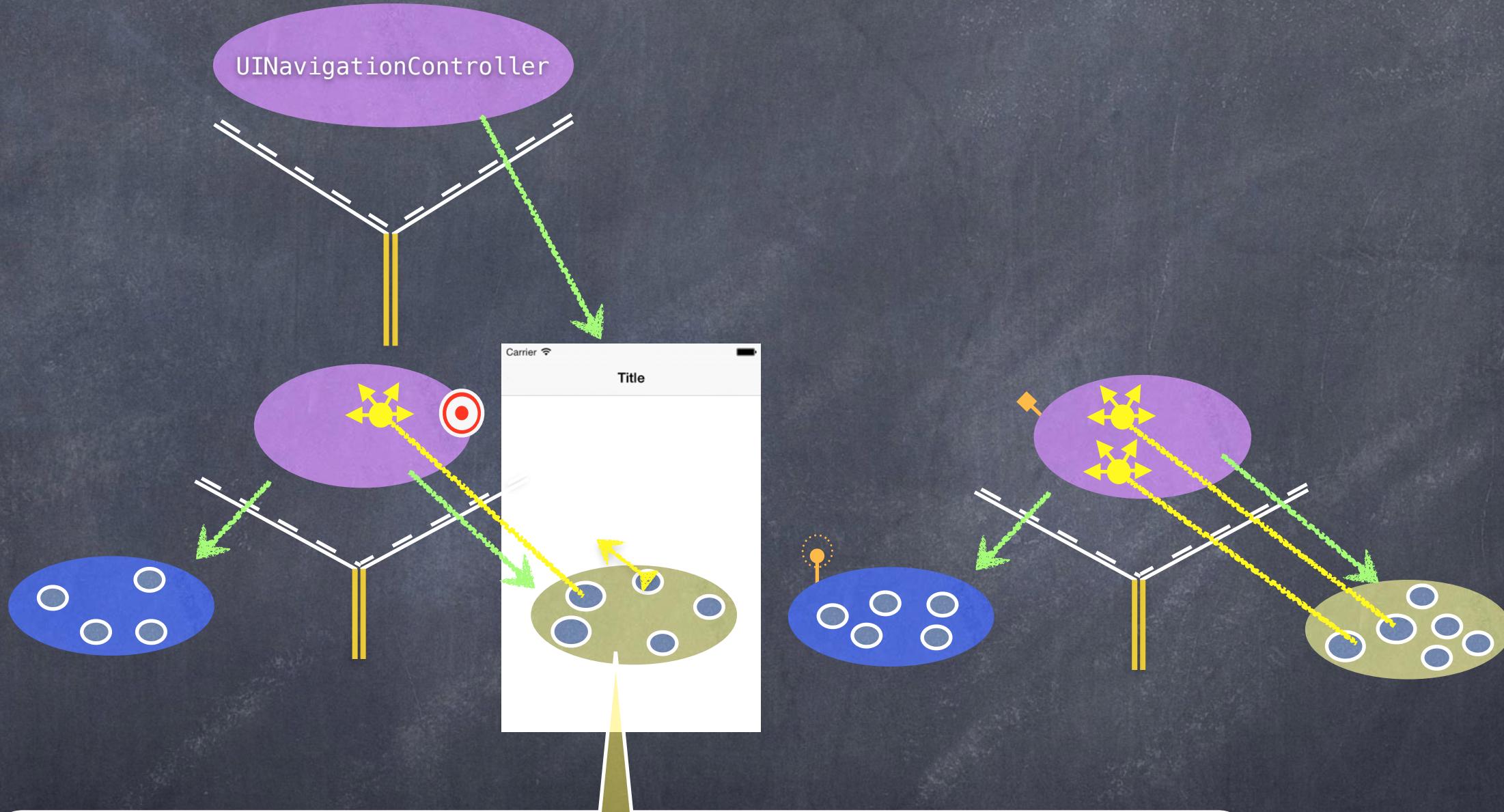
UINavigationController



UINavigationController



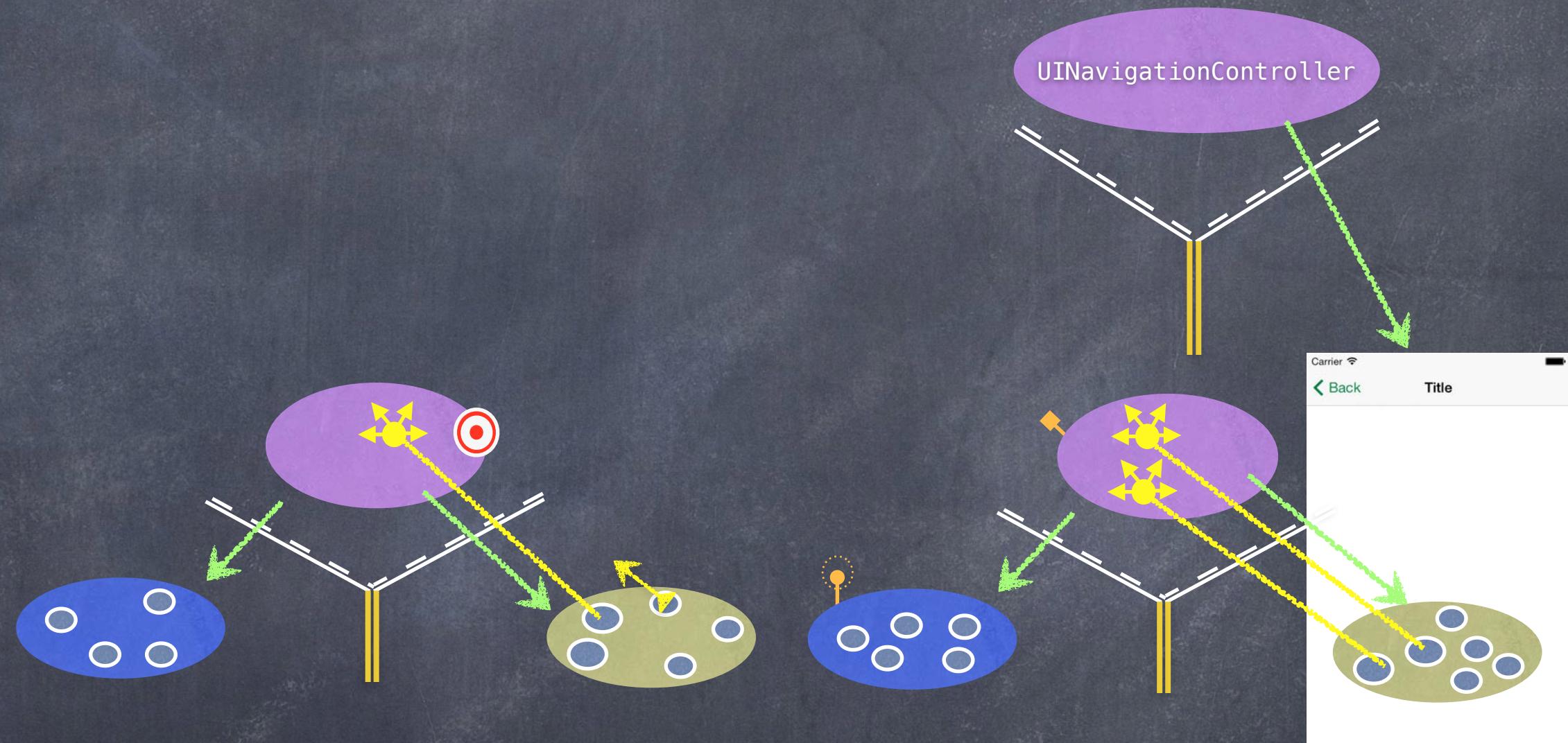
UINavigationController



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.



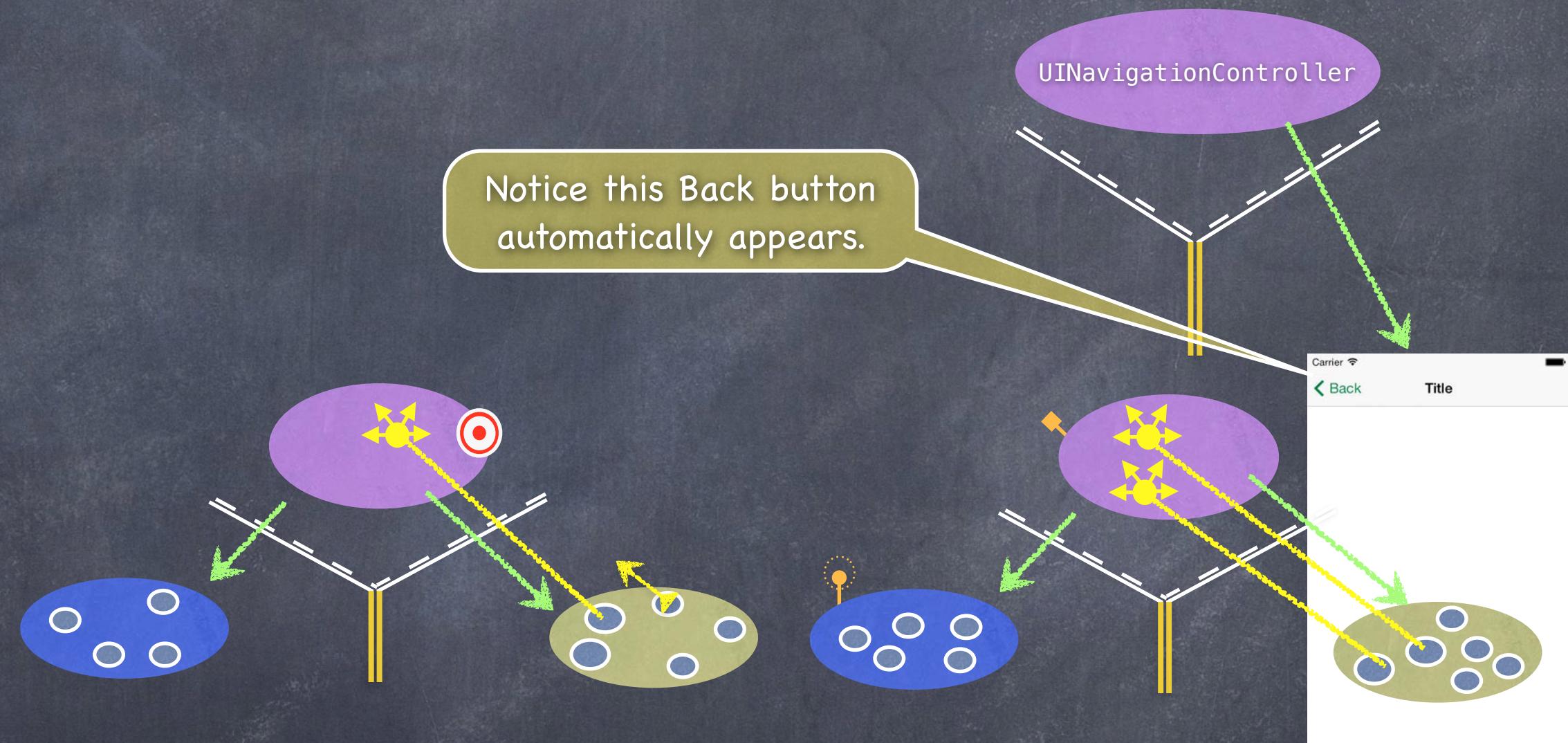
UINavigationController



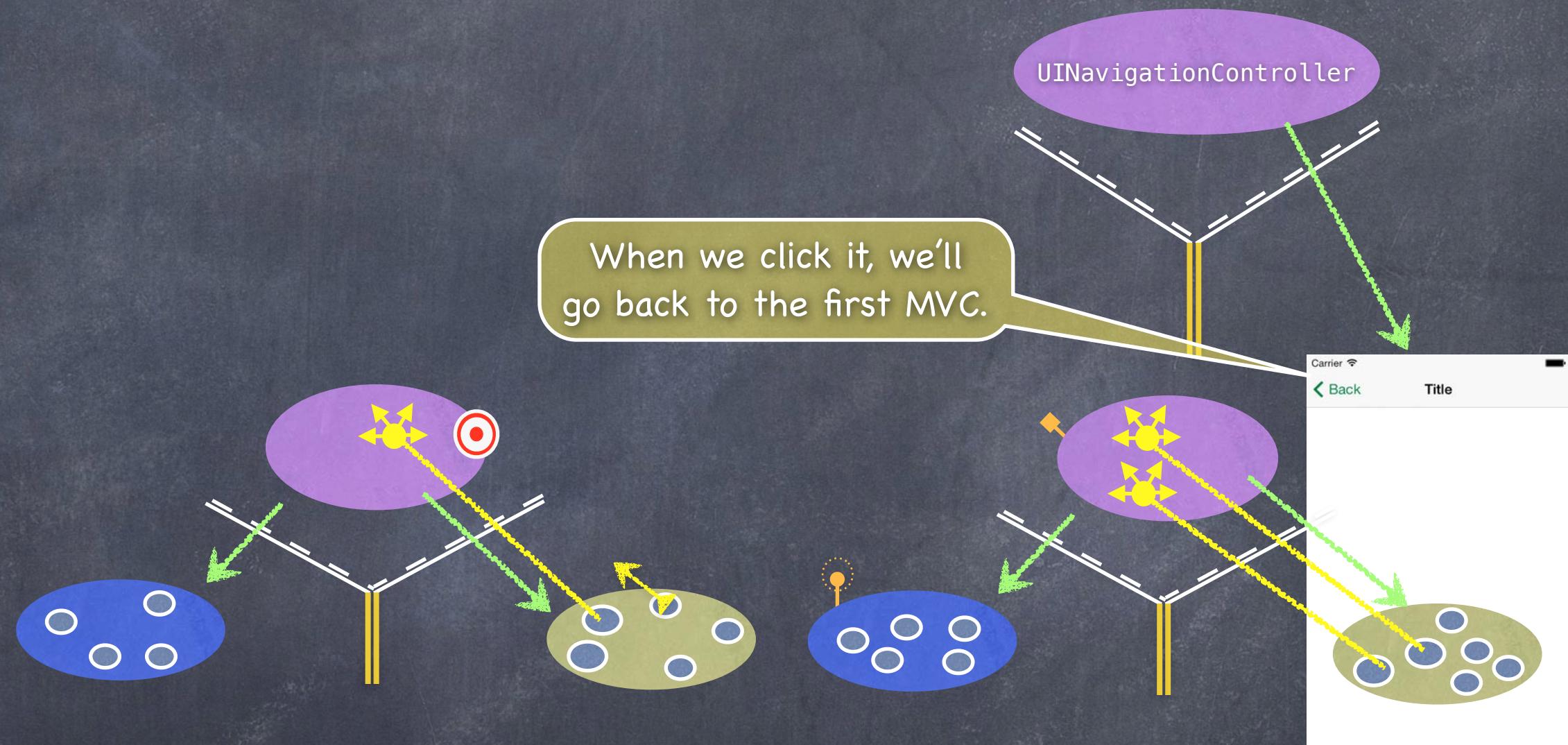
We call this kind of segue a
“Show (push) segue”.



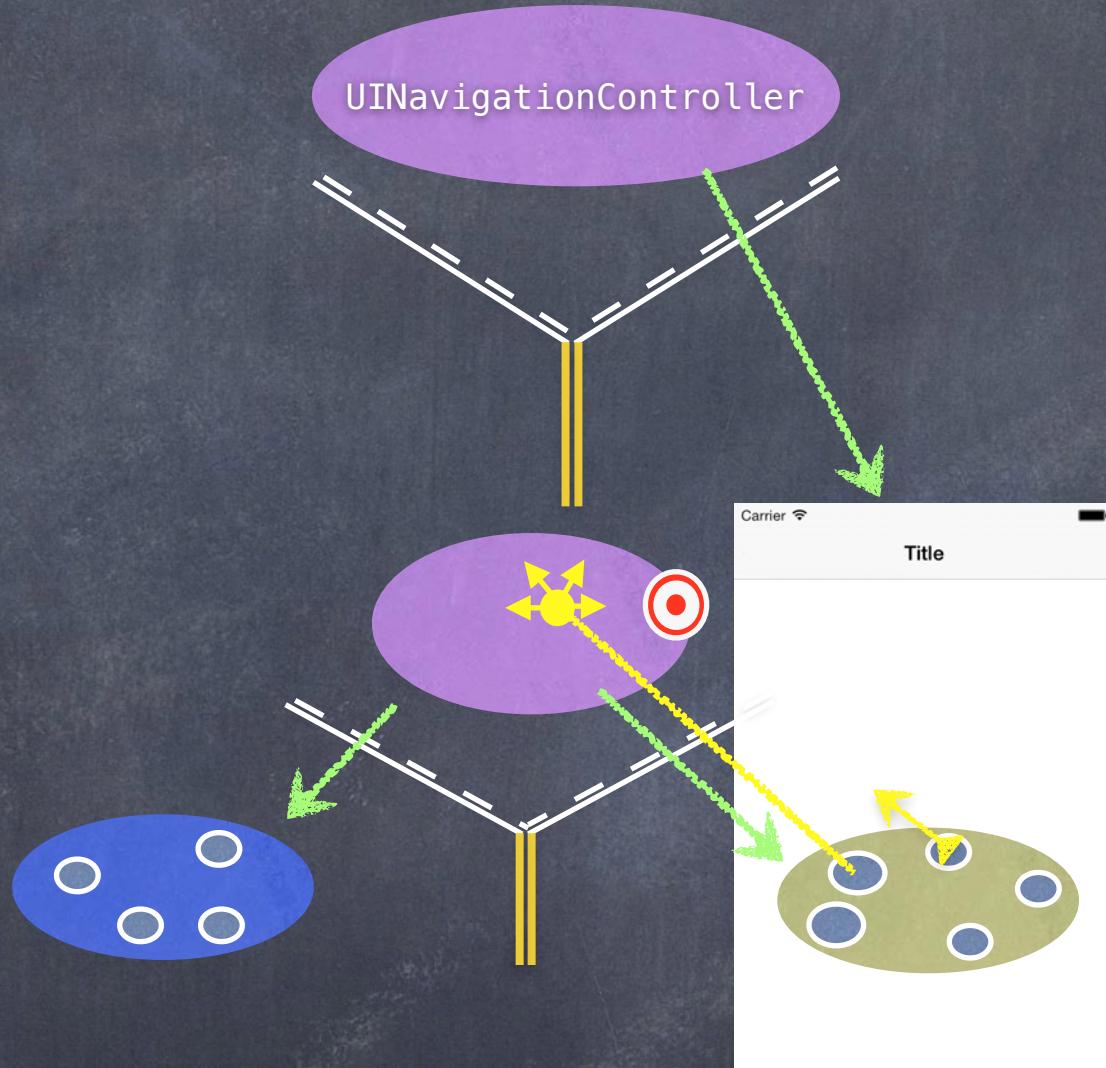
UINavigationController



UINavigationController



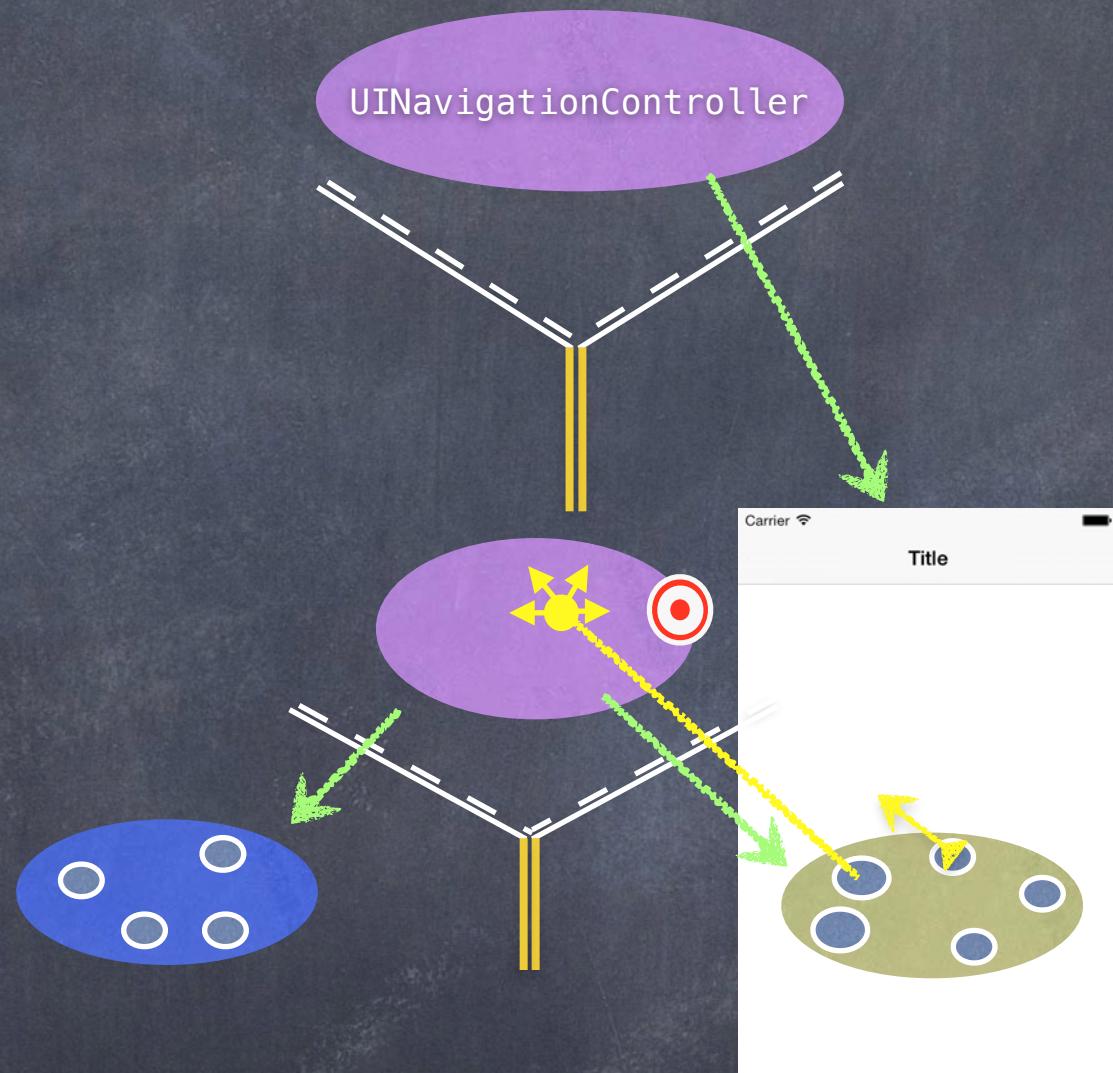
UINavigationController



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



UINavigationController



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 (right side) of the split view controller you are in ...

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



Pushing/Popping

- ⌚ Adding (or removing) MVCs from a UINavigationController

```
func pushViewController(_ vc: UIViewController, animated: Bool)
```

```
func popViewController(animated: Bool)
```

But we usually don't do this. Instead we use Segues. More on this in a moment.



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

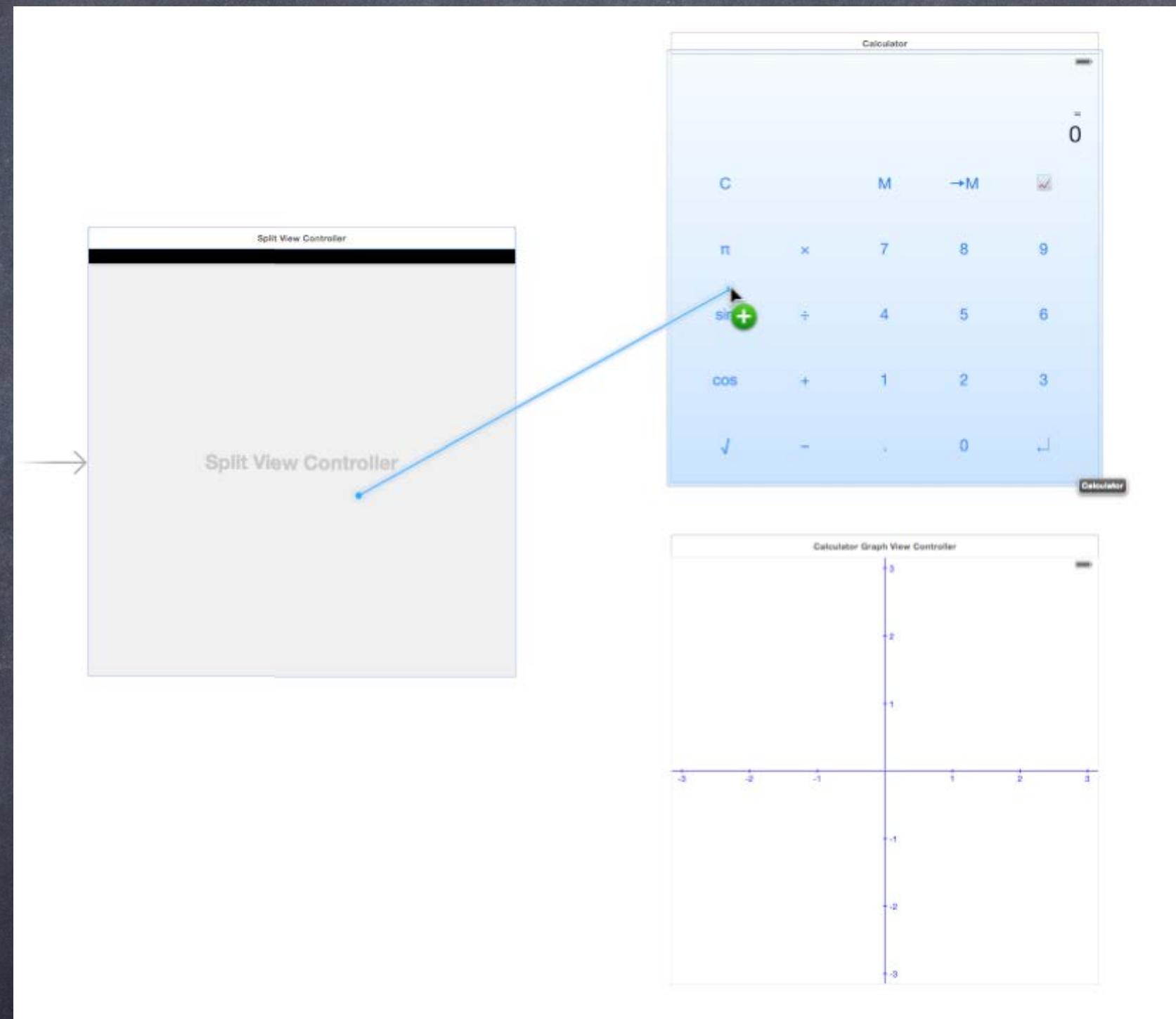


(and delete all the extra VCs it brings with it)

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



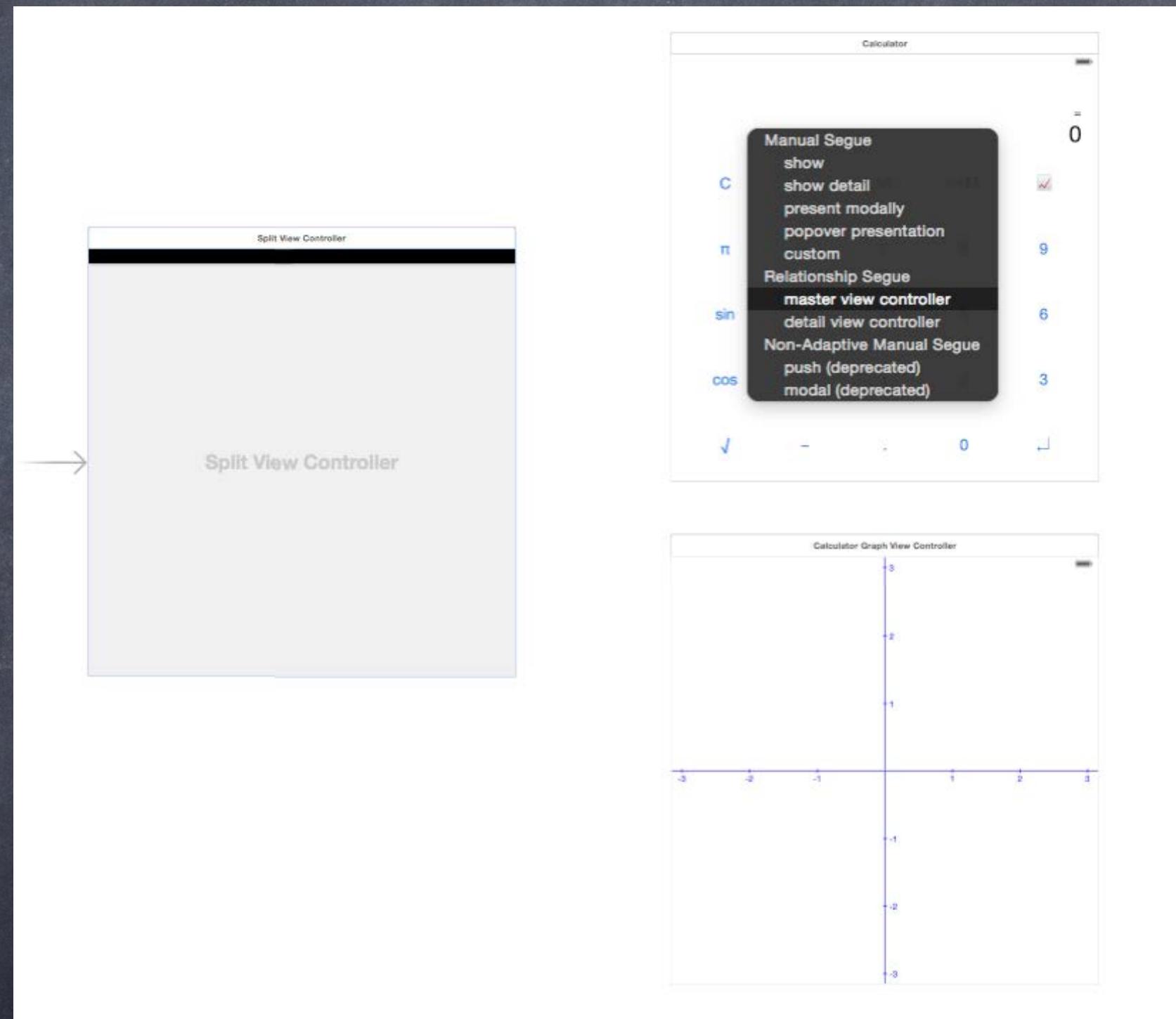
Wiring up MVCS



CS193p

Fall 2017-18

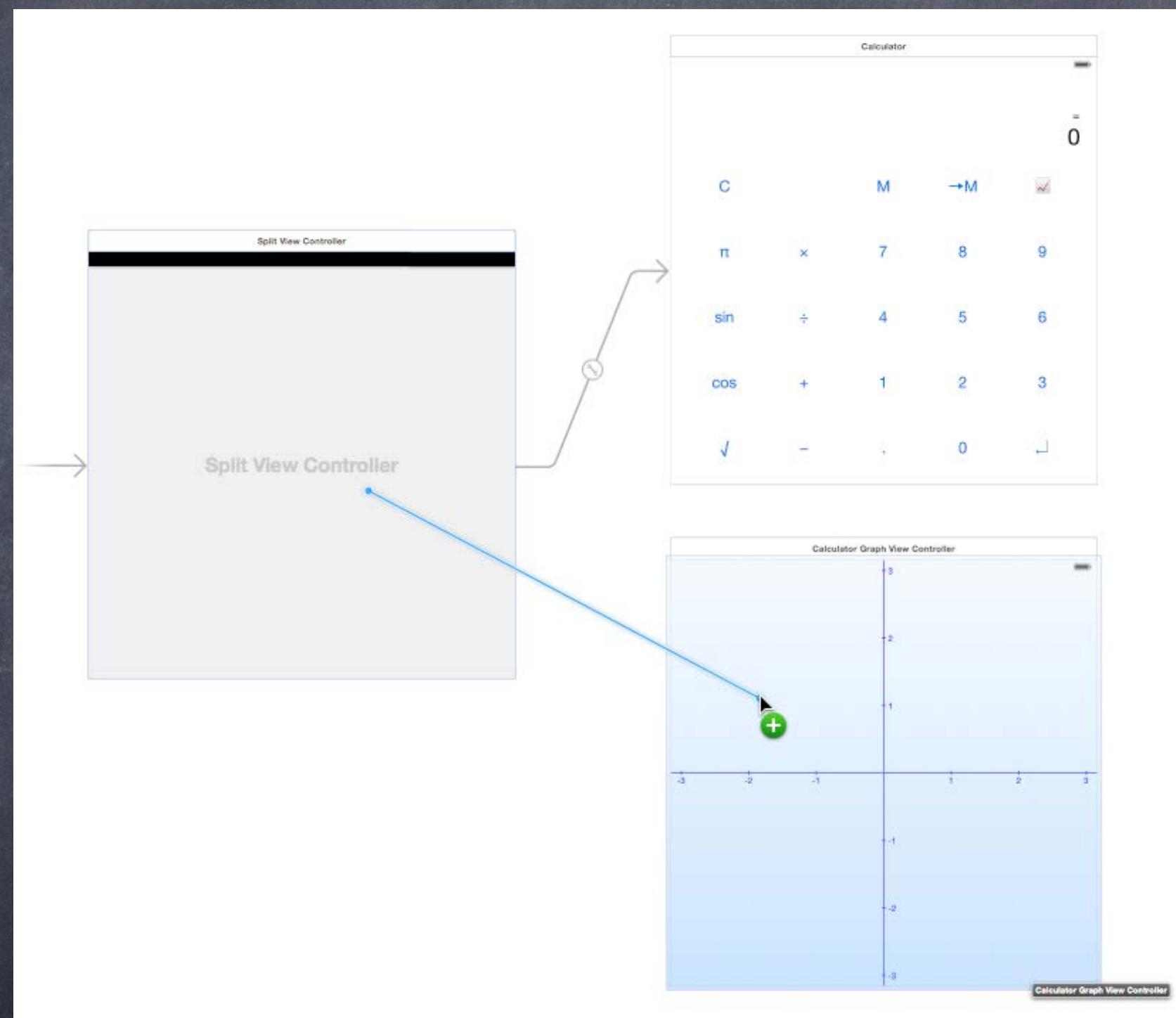
Wiring up MVCS



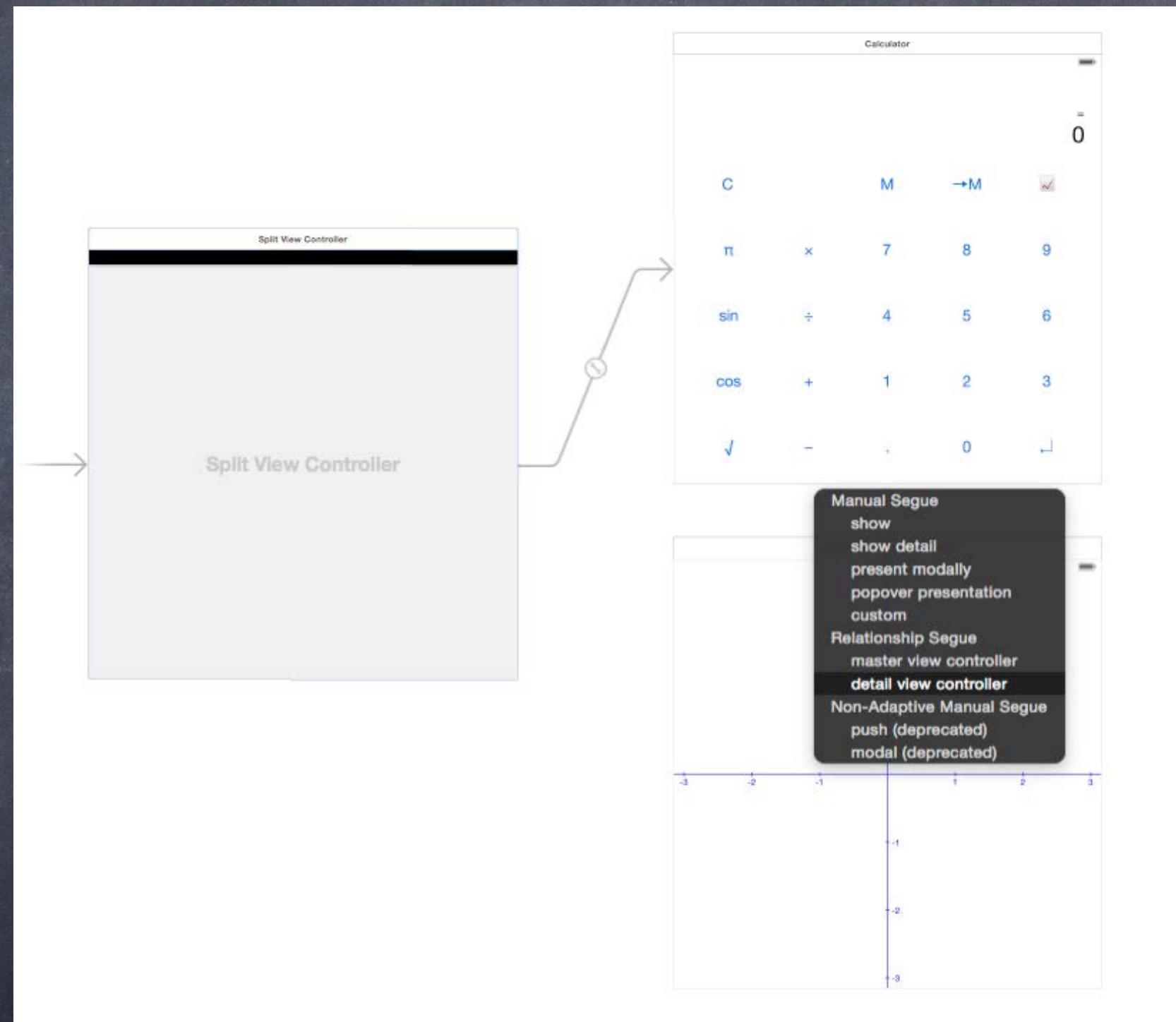
CS193p

Fall 2017-18

Wiring up MVCS



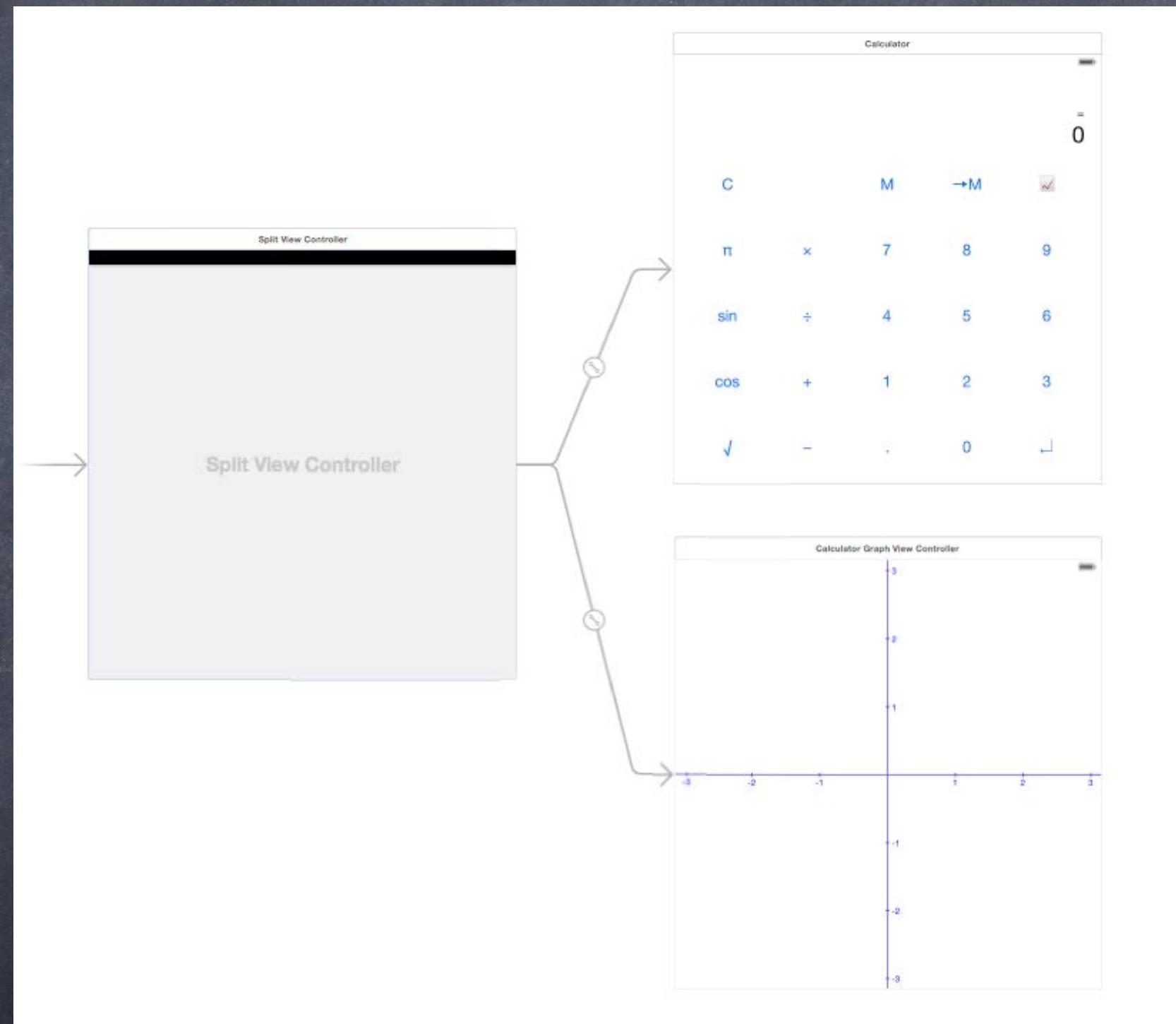
Wiring up MVCS



CS193p

Fall 2017-18

Wiring up MVCS



CS193p

Fall 2017-18

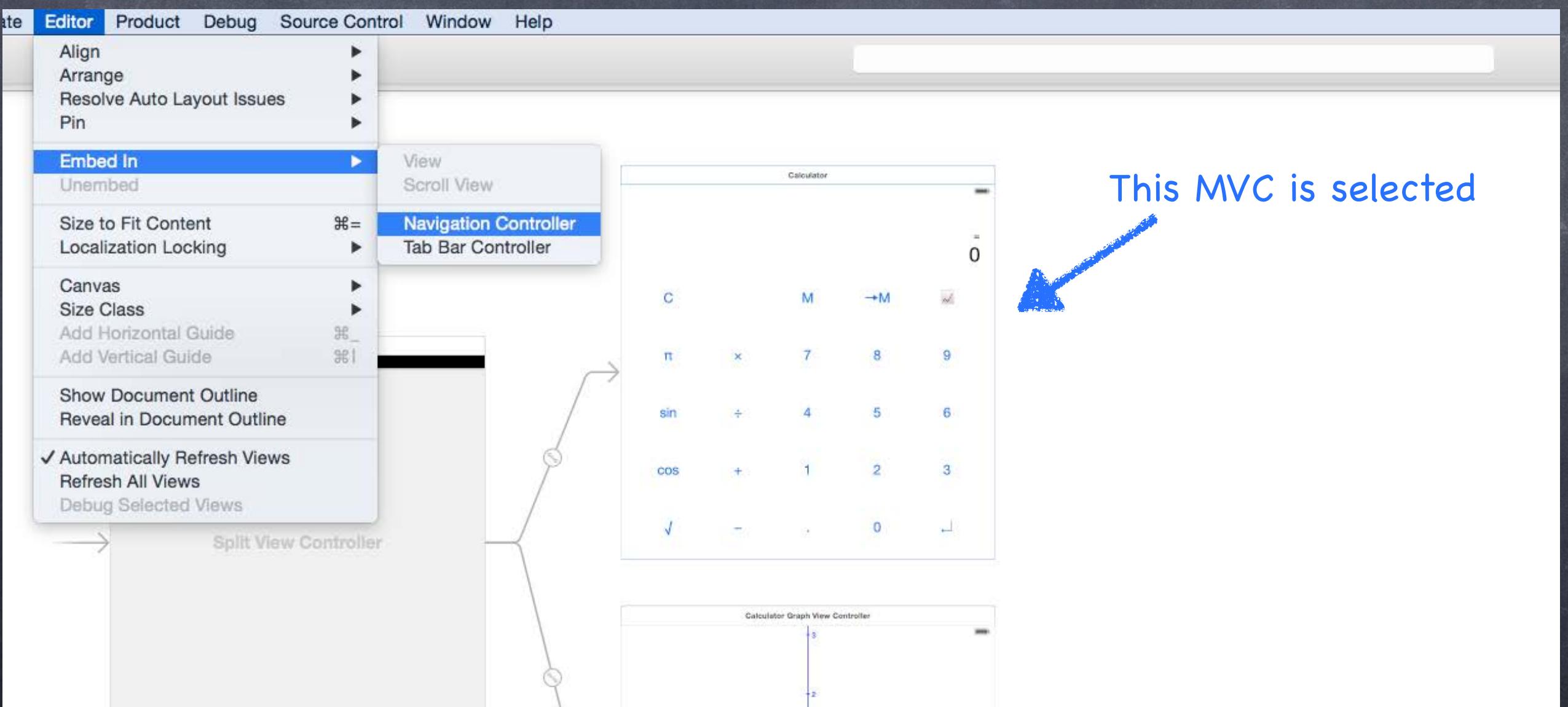
Wiring up MVCS

- But split view can only do its thing properly on iPad/iPhone+

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



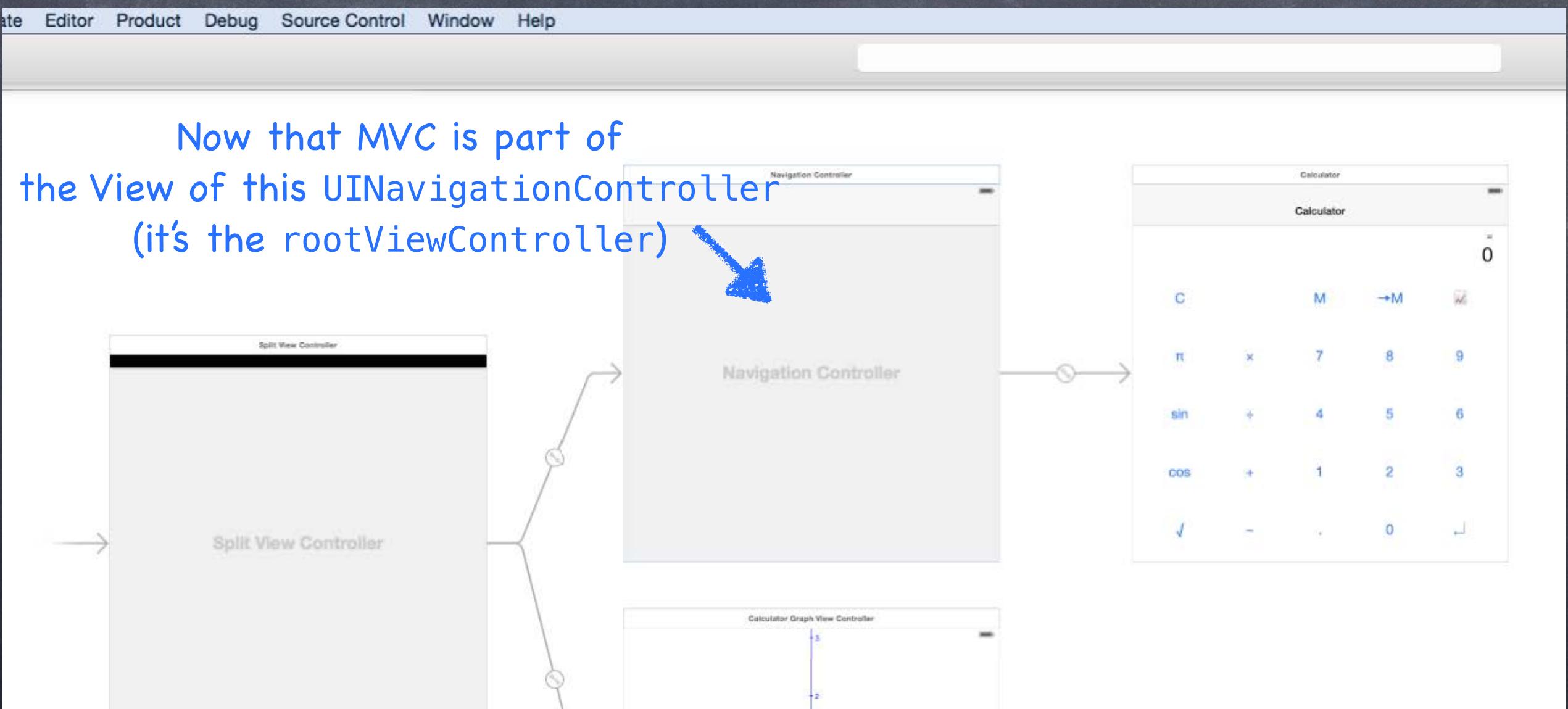
Wiring up MVCS

- But split view can only do its thing properly on iPad/iPhone+

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



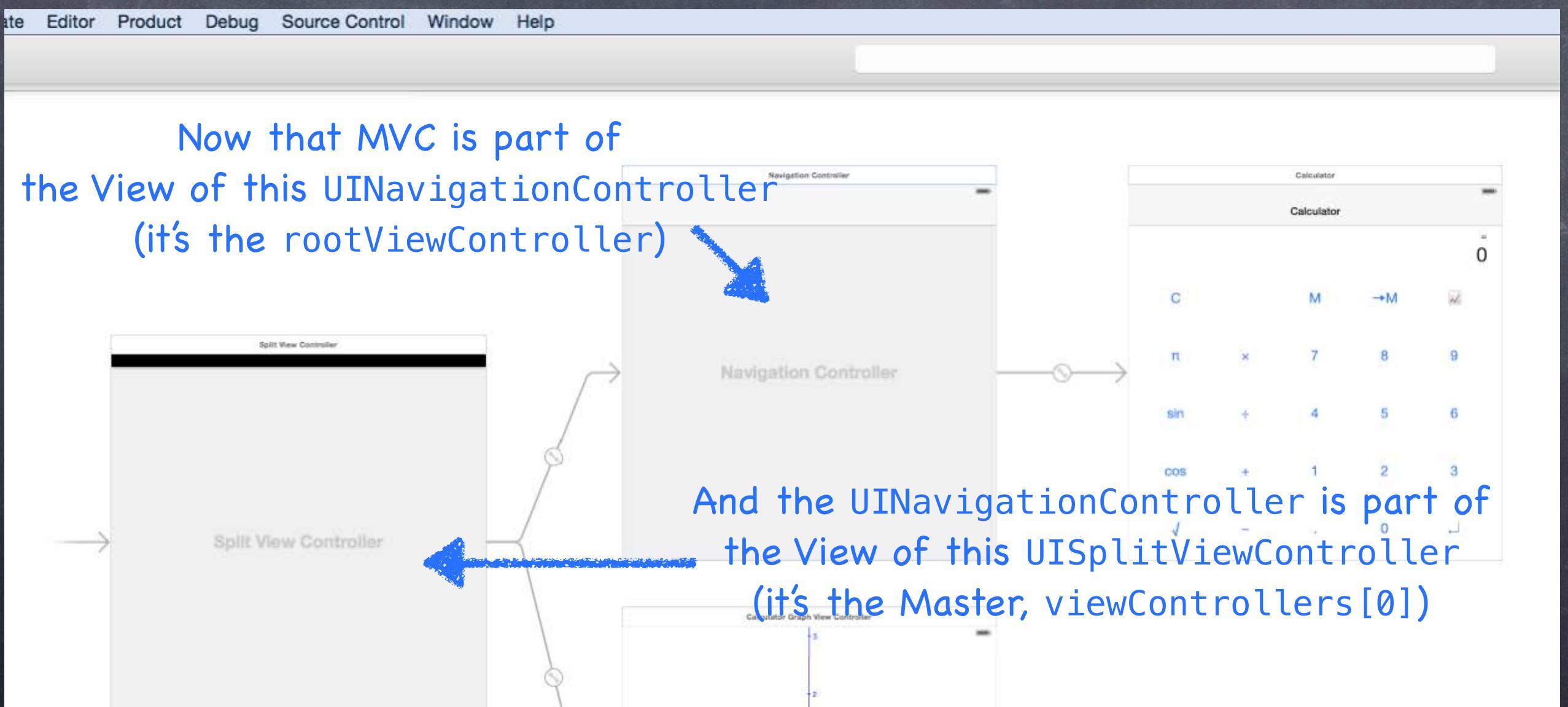
Wiring up MVCS

- But split view can only do its thing properly on iPad/iPhone+

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



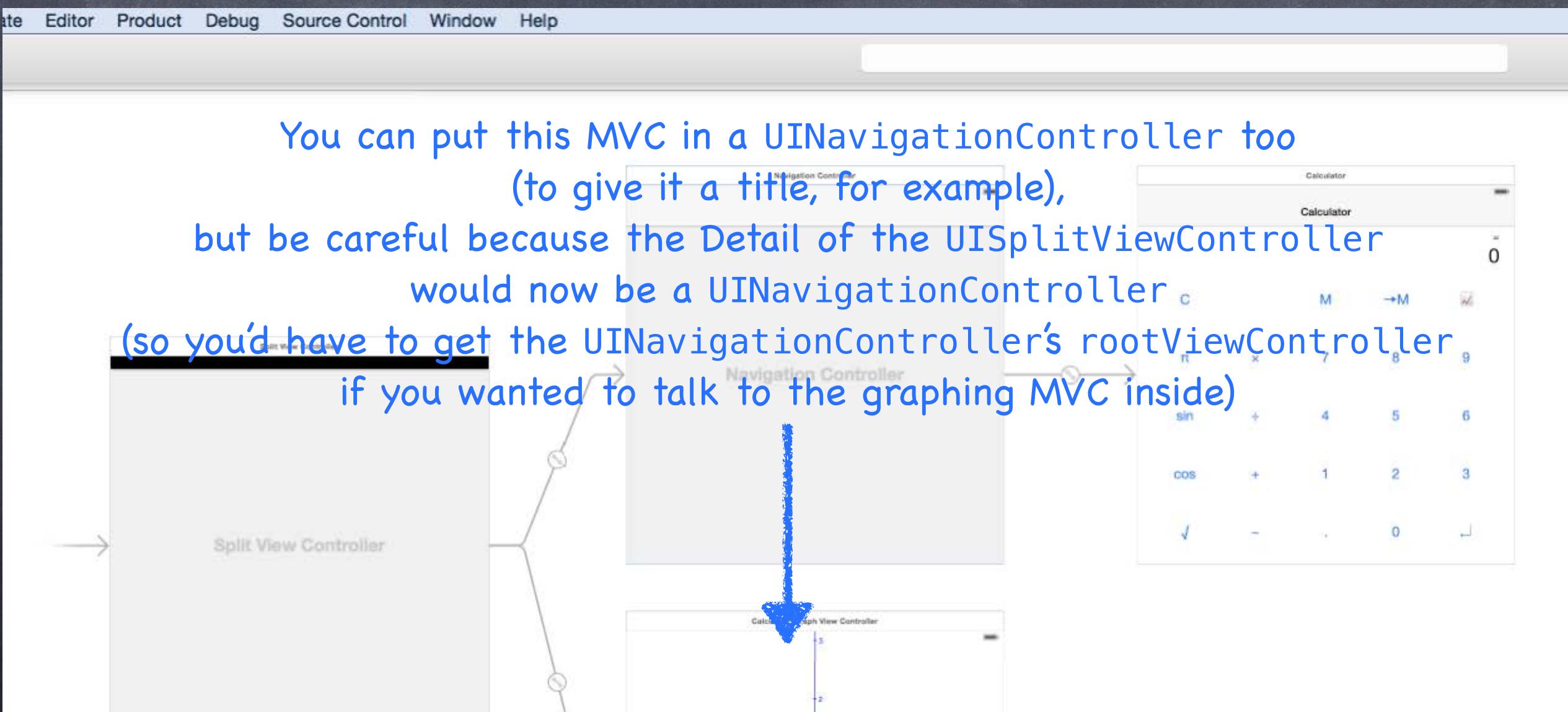
Wiring up MVCS

- But split view can only do its thing properly on iPad/iPhone+

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

Even 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

Going "back" in a Navigation Controller is NOT a segue though (so no new instance there)



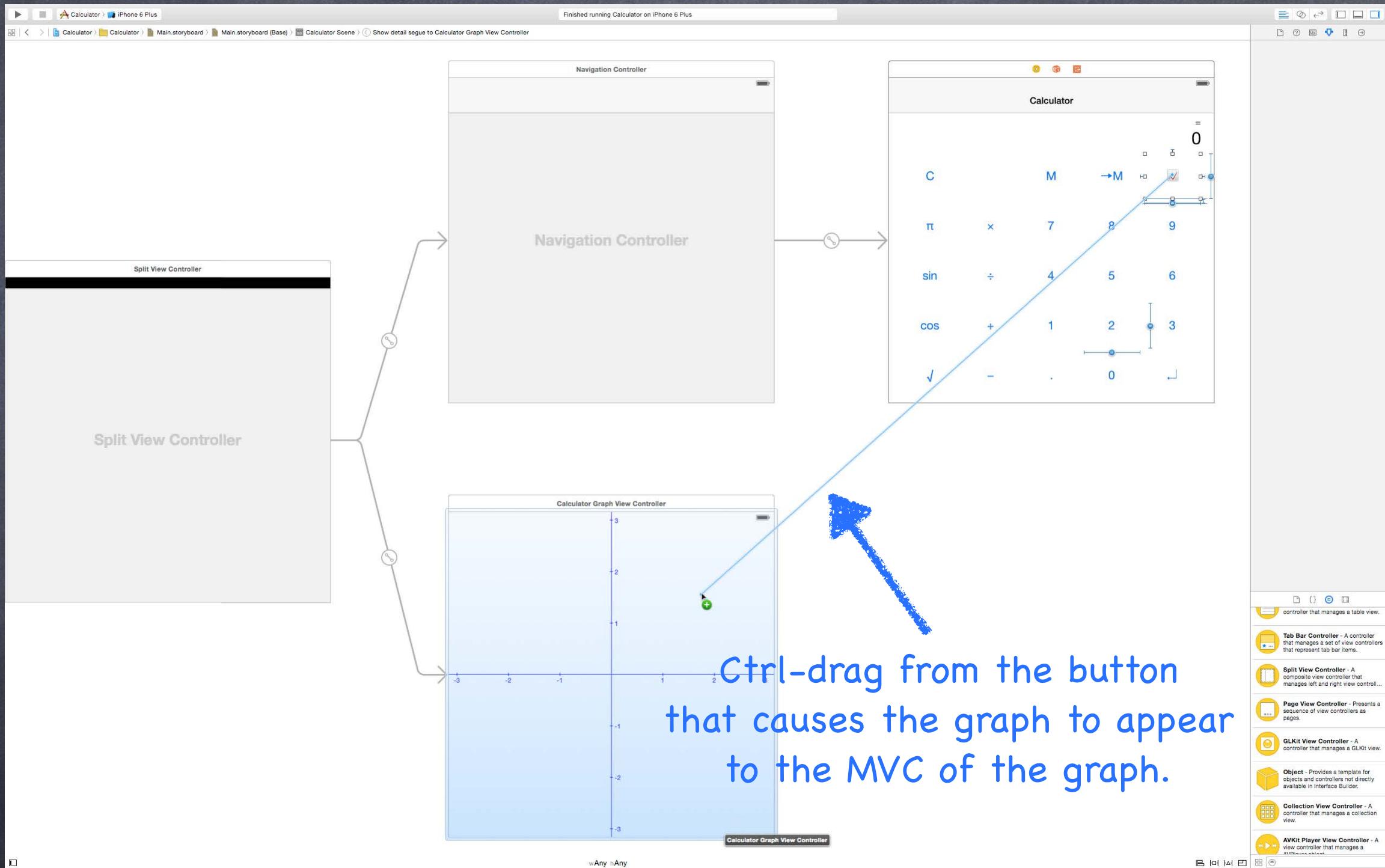
Segues

- How do we make these segues happen?

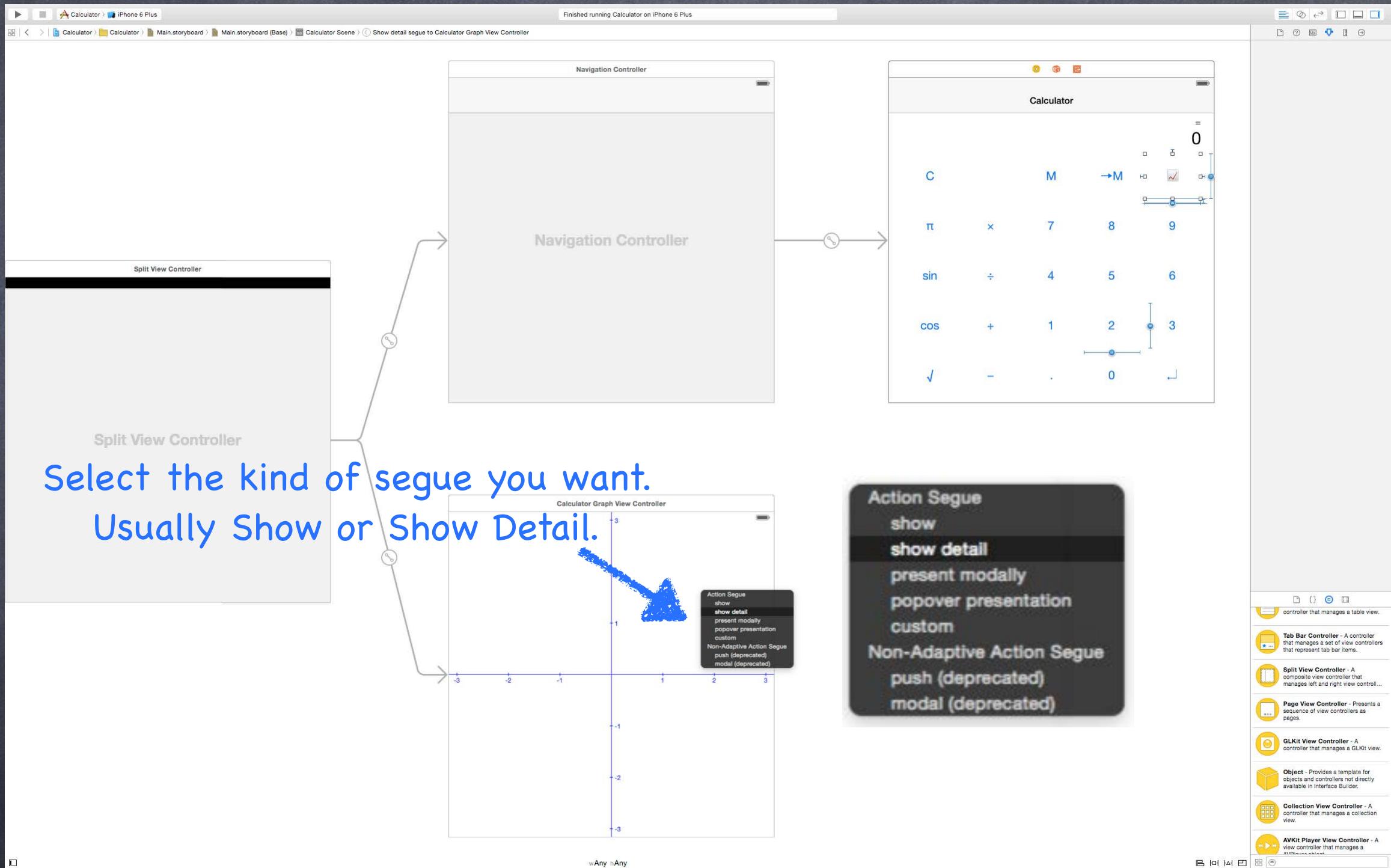
Ctrl-drag in a storyboard from an instigator (like a button) to the MVC to segue to
Can be done in code as well



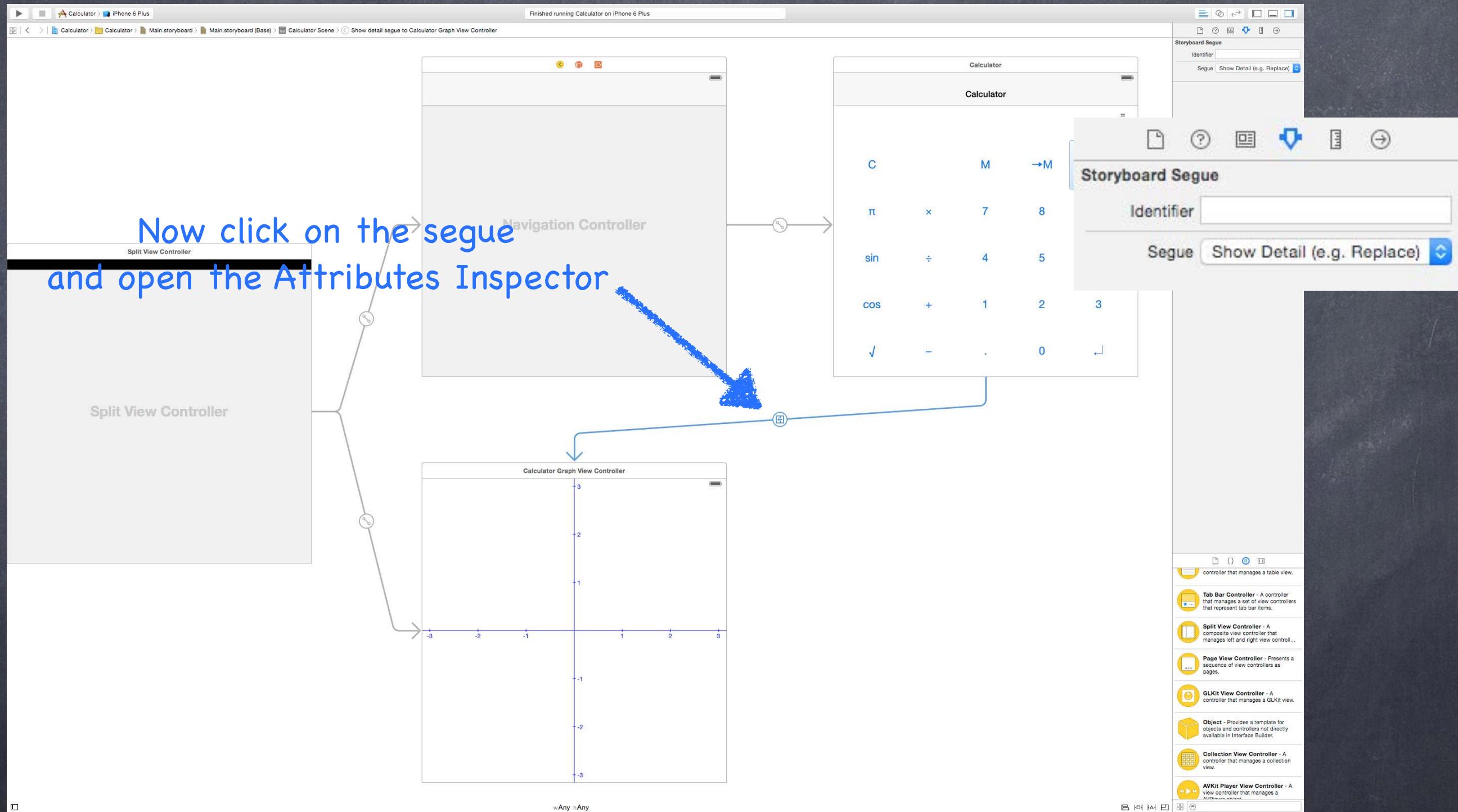
Segues



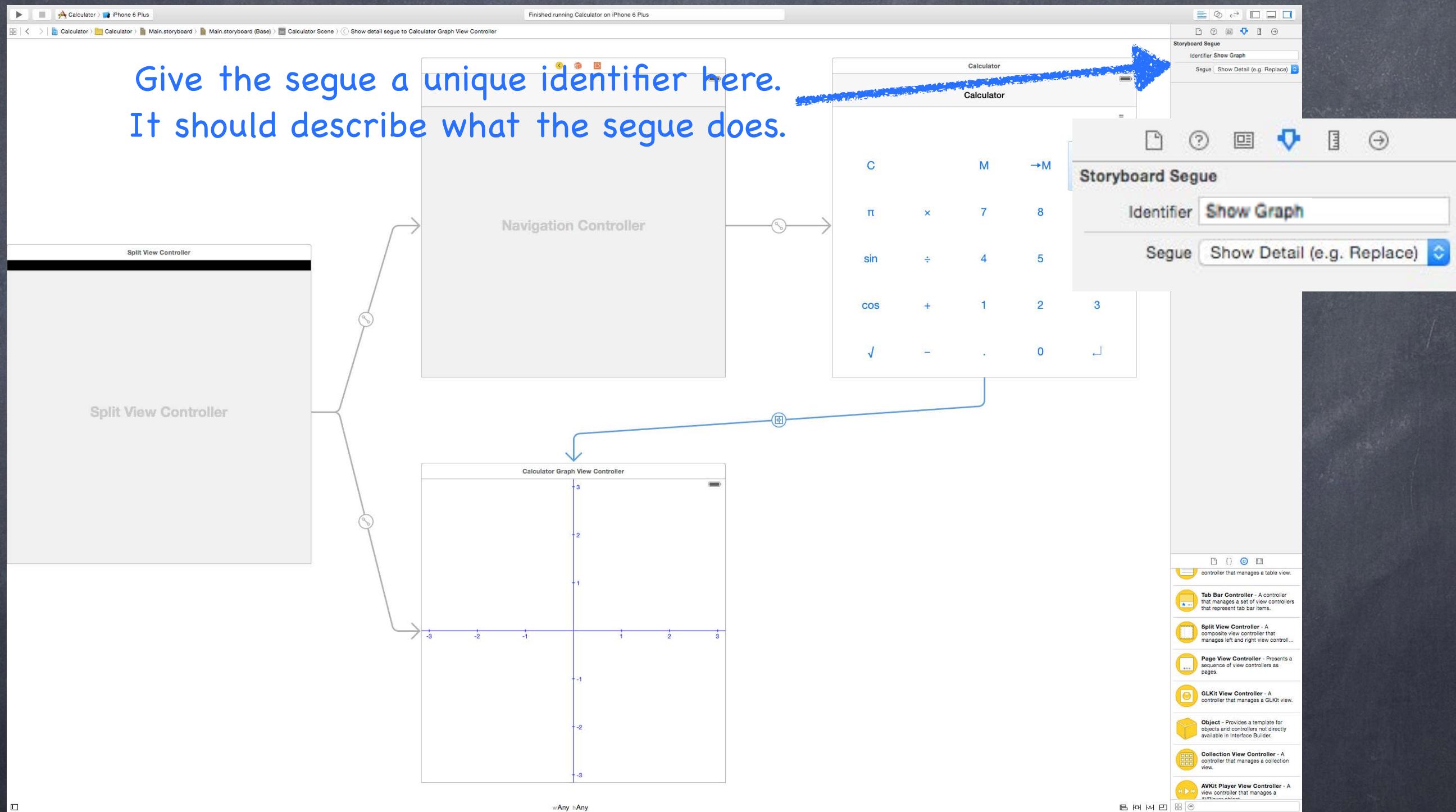
Segues



Segues



Segues



Segues

• What's that identifier all about?

You would need it to invoke this segue from code using this UIViewController method

```
func performSegue(withIdentifier: String, sender: Any?)
```

(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 prepare(for segue: UIStoryboardSegue, sender: Any?) {  
    if let identifier = segue.identifier {  
        switch identifier {  
            case "Show Graph":  
                if let vc = segue.destination as? GraphController {  
                    vc.property1 = ...  
                    vc.callMethodToSetUp(...)  
                }  
            default: break  
        }  
    }  
}
```



Preparing for a Segue

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

```
func prepare(for segue: UIStoryboardSegue, sender: Any?) {  
    if let identifier = segue.identifier {  
        switch identifier {  
            case "Show Graph":  
                if let vc = segue.destination as? GraphController {  
                    vc.property1 = ...  
                    vc.callMethodToSetUp(...)  
                }  
            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 prepare(for segue: UIStoryboardSegue, sender: Any?) {  
    if let identifier = segue.identifier {  
        switch identifier {  
            case "Show Graph":  
                if let vc = segue.destination as? GraphController {  
                    vc.property1 = ...  
                    vc.callMethodToSetUp(...)  
                }  
            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 prepare(for segue: UIStoryboardSegue, sender: Any?) {  
    if let identifier = segue.identifier {  
        switch identifier {  
            case "Show Graph":  
                if let vc = segue.destination as? GraphController {  
                    vc.property1 = ...  
                    vc.callMethodToSetUp(...)  
                }  
            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 prepare(for segue: UIStoryboardSegue, sender: Any?) {  
    if let identifier = segue.identifier {  
        switch identifier {  
            case "Show Graph":  
                if let vc = segue.destination as? GraphController {  
                    vc.property1 = ...  
                    vc.callMethodToSetUp(...)  
                }  
            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 prepare(for segue: UIStoryboardSegue, sender: Any?) {  
    if let identifier = segue.identifier {  
        switch identifier {  
            case "Show Graph":  
                if let vc = segue.destination as? GraphController {  
                    vc.property1 = ...  
                    vc.callMethodToSetUp(...)  
                }  
            default: break  
        }  
    }  
}
```

Here we are looking at the Controller of the MVC we're segueing to
It is Any 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 prepare(for segue: UIStoryboardSegue, sender: Any?) {  
    if let identifier = segue.identifier {  
        switch identifier {  
            case "Show Graph":  
                if let vc = segue.destination as? GraphController {  
                    vc.property1 = ...  
                    vc.callMethodToSetUp(...)  
                }  
            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 prepare(for segue: UIStoryboardSegue, sender: Any?) {  
    if let identifier = segue.identifier {  
        switch identifier {  
            case "Show Graph":  
                if let vc = segue.destination as? GraphController {  
                    vc.property1 = ...  
                    vc.callMethodToSetUp(...)  
                }  
            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 return false from this method your UIViewController ...

```
func shouldPerformSegue(withIdentifier identifier: String?, sender: Any?) -> 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

⌚ Concentration Theme Chooser

This is all best understood via demonstration

We'll put the MVCS into navigation controllers inside split view controllers

That way, it will work on both iPad and iPhone devices



Timer

- ⌚ Used to execute code periodically

- You can set it up to go off once at some time in the future, or to repeatedly go off

- If repeatedly, the system will not guarantee exactly when it goes off, so this is not “real-time”

- But for most UI “order of magnitude” activities, it’s perfectly fine

- We don’t generally use it for “animation” (more on that later)

- It’s more for larger-grained activities



Timer

- ⌚ Fire one off with this method ...

```
class func scheduledTimer(  
    withTimeInterval: TimeInterval,  
    repeats: Bool,  
    block: (Timer) -> Void  
) -> Timer
```

- ⌚ Example

```
private weak var timer: Timer?  
  
timer = Timer.scheduledTimer(withTimeInterval: 2.0, repeats: true) { timer in  
    // your code here  
}
```

Every 2 seconds (approximately), the closure will be executed.

Note that the var we stored the timer in is **weak**.

That's okay because the run loop will keep a strong pointer to this as long as it's scheduled.



Timer

⌚ Stopping a repeating timer

We need to be a bit careful with repeating timers ... you don't want them running forever.

You stop them by calling `invalidate()` on them ...

`timer.invalidate()`

This tells the run loop to stop scheduling the timer.

The run loop will thus give up its strong pointer to this timer.

If your pointer to the timer is `weak`, it will be set to nil at this point.

This is nice because an invalidated timer like this is no longer of any use to you.

⌚ Tolerance

It might help system performance to set a tolerance for "late firing".

For example, if you have timer that goes off once a minute, a tolerance of 10s might be fine.

`myOneMinuteTimer.tolerance = 10 // in seconds`

The firing time is relative to the start of the timer (not the last time it fired), i.e. no "drift".



Kinds of Animation

- ⦿ Animating UIView properties

Changing things like the frame or transparency.

- ⦿ Animating Controller transitions (as in a UINavigationController)

Beyond the scope of this course, but fundamental principles are the same.

- ⦿ Core Animation

Underlying powerful animation framework (also beyond the scope of this course).

- ⦿ OpenGL and Metal

3D

- ⦿ SpriteKit

“2.5D” animation (overlapping images moving around over each other, etc.)

- ⦿ Dynamic Animation

“Physics”-based animation.



UIView Animation

- Changes to certain UIView properties can be animated over time
 - frame/center
 - bounds (transient size, does not conflict with animating center)
 - transform (translation, rotation and scale)
 - alpha (opacity)
 - backgroundColor

