THE UNIVERSITY OF CHICAGO

ADVANCED iOS APPLICATION DEVELOPMENT

MPCS 51032 • SPRING 2020 • SESSION 1A

VIEW CONTROLLER EVOLUTION

VIEW CONTROLLER EVOLUTION

THE VIEW CONTROLLER

THE VIEW CONTROLLER

IN THE BEGINNING...

- UIViewControllers jobs
  - Manages “screenful of objects”
  - Manage view hierarchy
  - Applications flow between view controllers

THE VIEW CONTROLLER

CUSTOM VIEW CONTROLLERS IMPLEMENTATION

- Subclass `UIViewController`
- Associate the view controller with a view hierarchy
- Override select APIs to suit your needs
- Add your application logic

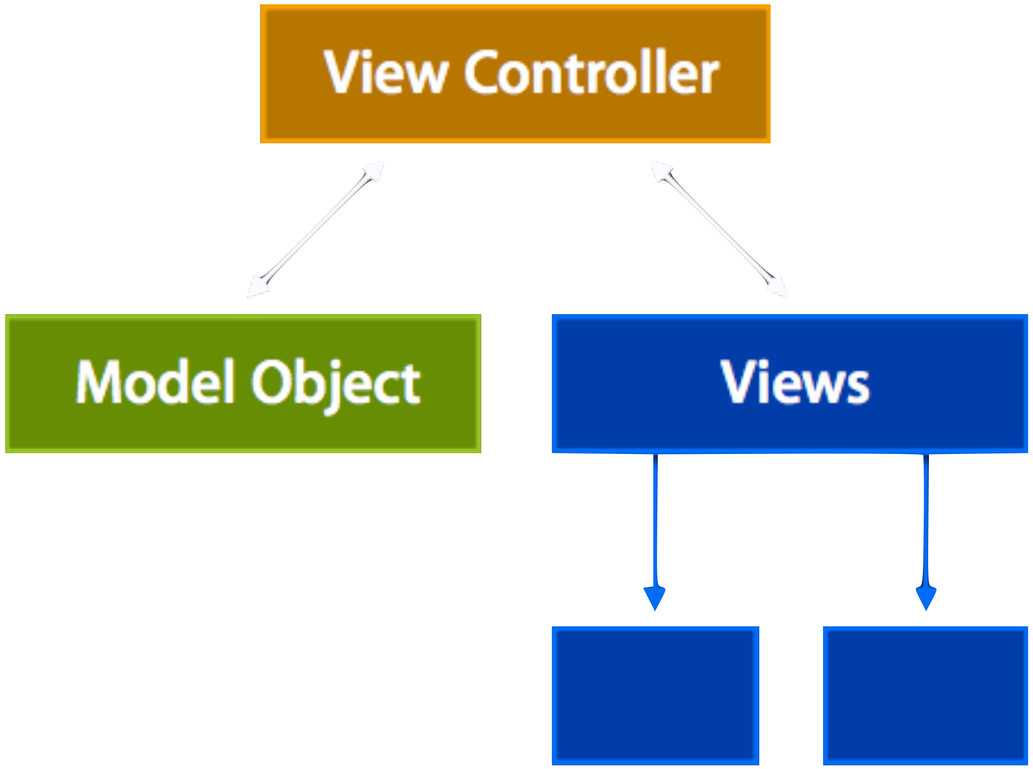


Image via Apple

THE VIEW CONTROLLER

CUSTOM VIEW CONTROLLERS IMPLEMENTATION

- Commonly overridden methods
  - Loading callbacks
    - `viewDidLoad()`
    - `viewDidUnload()`
  - Appearance callbacks
    - `viewWillAppear()`
    - `viewDidAppear()`
    - `viewWillDisappear()`
    - `viewDidDisappear()`

```
//
//  ViewController.swift
//  Session1-PageViewController
//
//  Created by T. Andrew Binkowski on 4/2/17.
//  Copyright © 2017 T. Andrew Binkowski. All rights reserved.
//

import UIKit

class ViewController: UIViewController {

    override func viewDidLoad() {
        super.viewDidLoad()

        // Do any additional setup after loading the view.
    }

    override func didReceiveMemoryWarning() {
        super.didReceiveMemoryWarning()
        // Dispose of any resources that can be recreated.
    }


    /*
    // MARK: - Navigation

    // In a storyboard-based application, you will often want to
    override func prepare(for segue: UIStoryboardSegue, sender:
        // Get the new view controller using segue.destinationVi
        // Pass the selected object to the new view controller.
    }
    */

}
```

THE VIEW CONTROLLER

CUSTOM VIEW CONTROLLERS IMPLEMENTATION

Other than segues

- How do view controllers get on/off the screen?
  - By container controllers
  - By presentation

```
presentViewController(gvc, animated: true, completion: nil)
```

  - By dismissal

```
presentingViewController?.dismissViewControllerAnimated(true, completion: nil)
```

  - By direct view manipulation

```
window.rootViewController = rootViewController;
```

THE VIEW CONTROLLER

SUBTITLE

- `UIViewController` advantages
  - Consistent with Apple apps
  - Easily reusable
  - Simplify many of the tasks for managing views and models

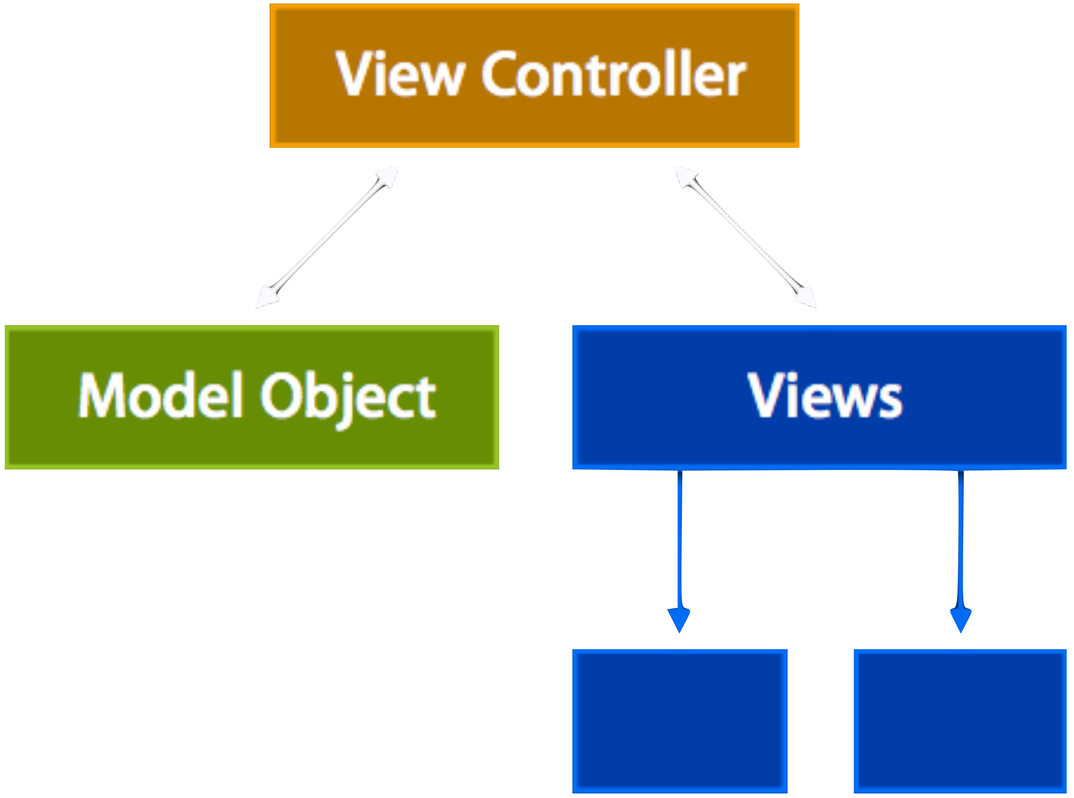
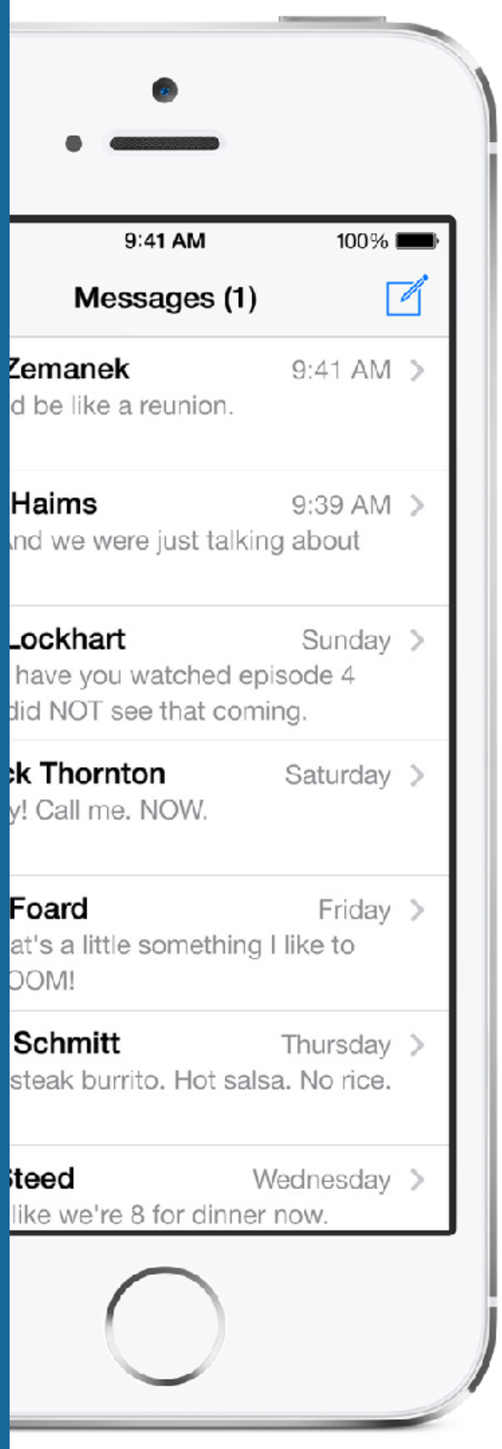
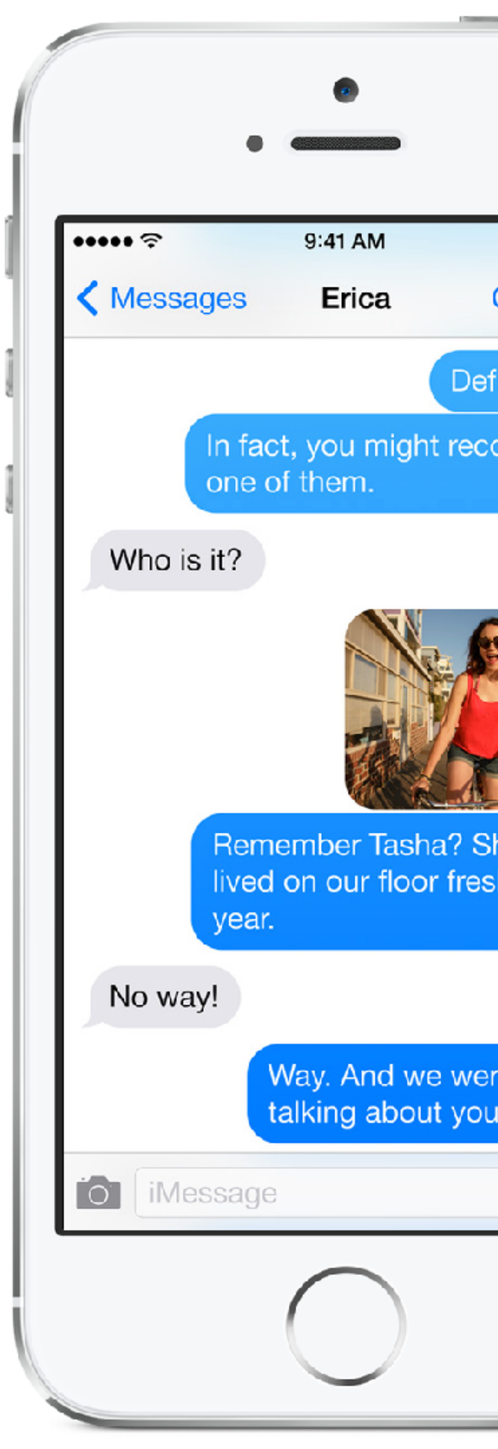


Image via Apple

VIEW CONTROLLERS

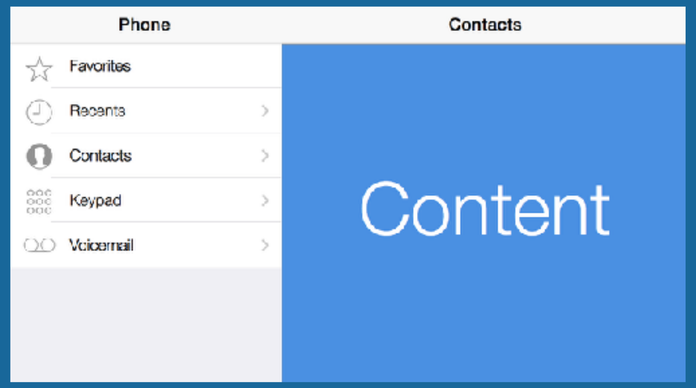
- For many interfaces it "makes sense"
- Screenful of content

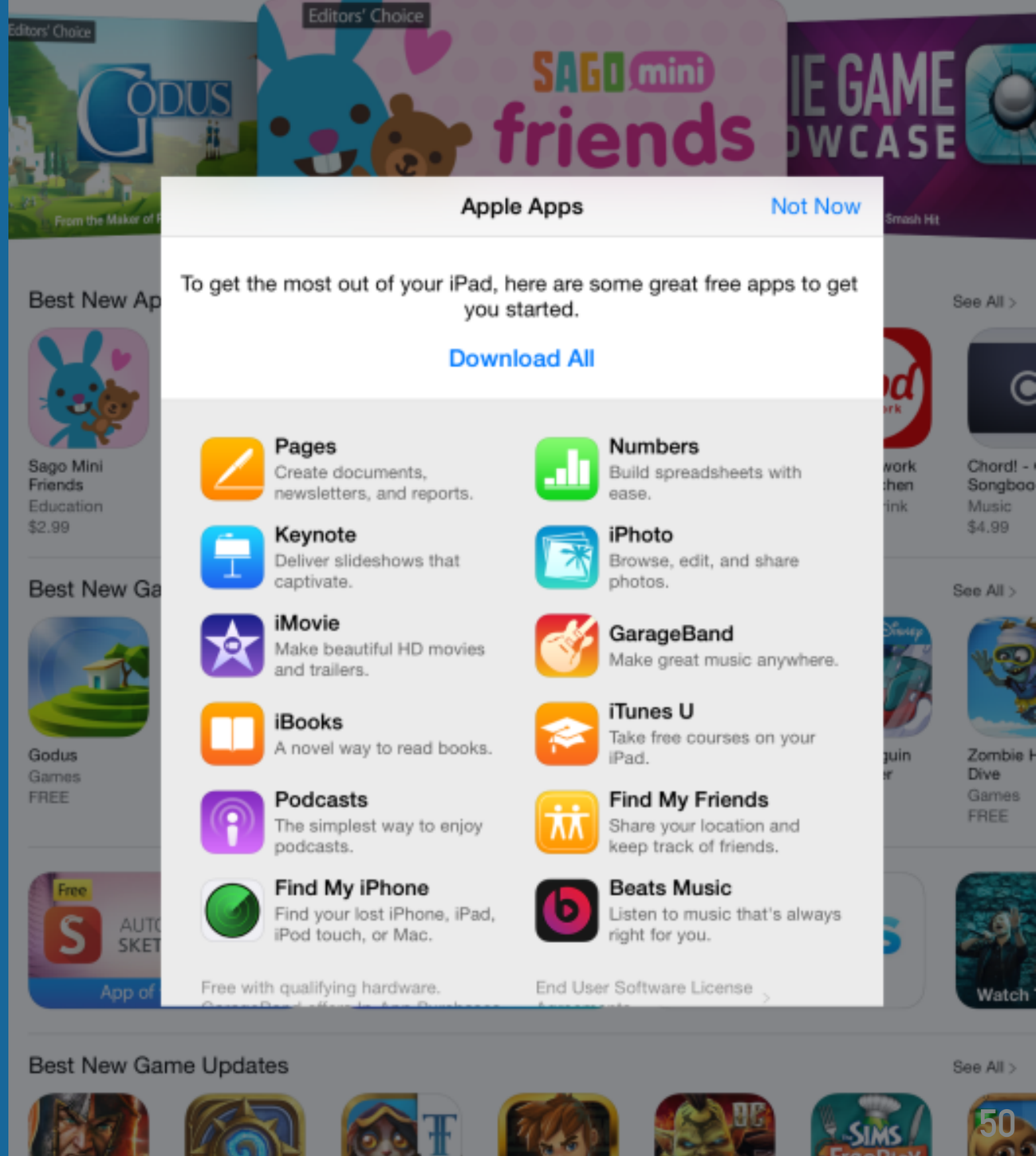




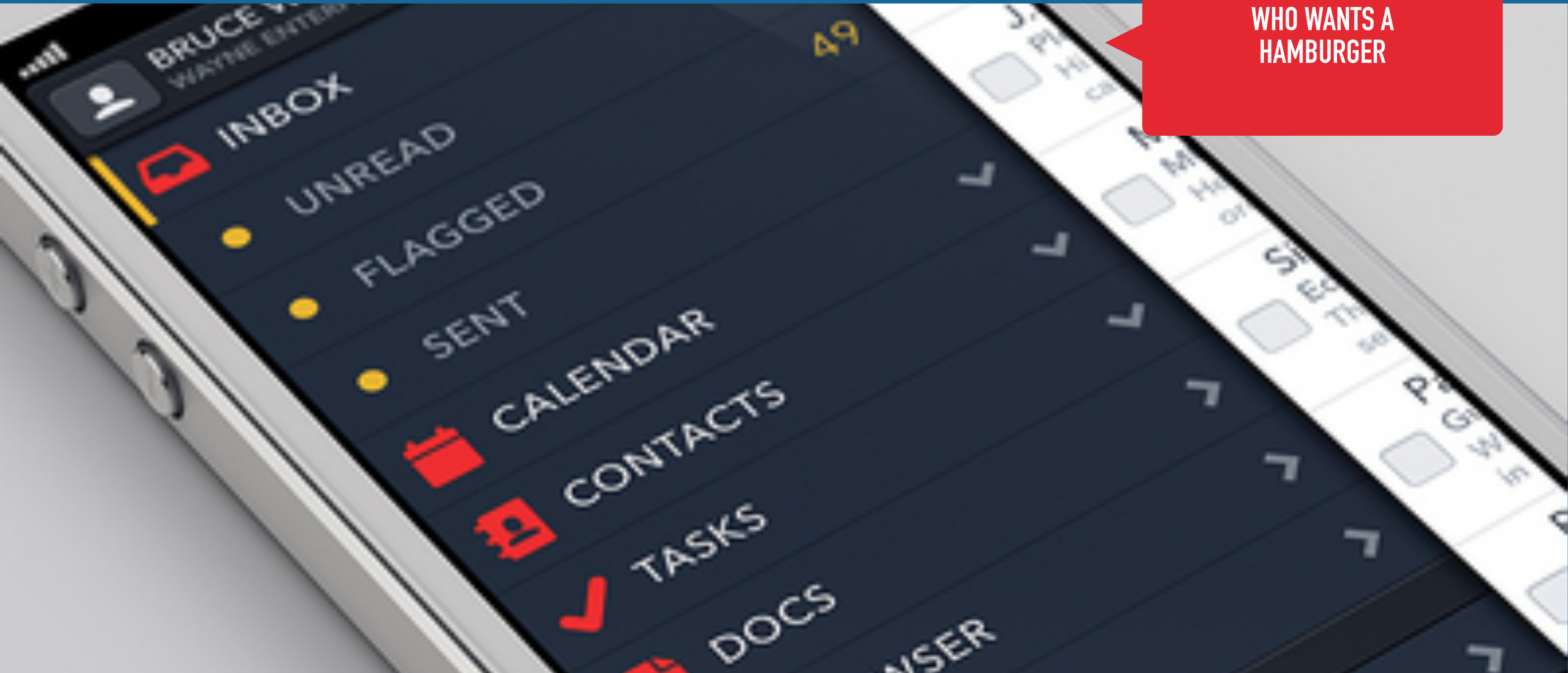
VIEW CONTROLLERS

- “Screenful of content” getting harder to define



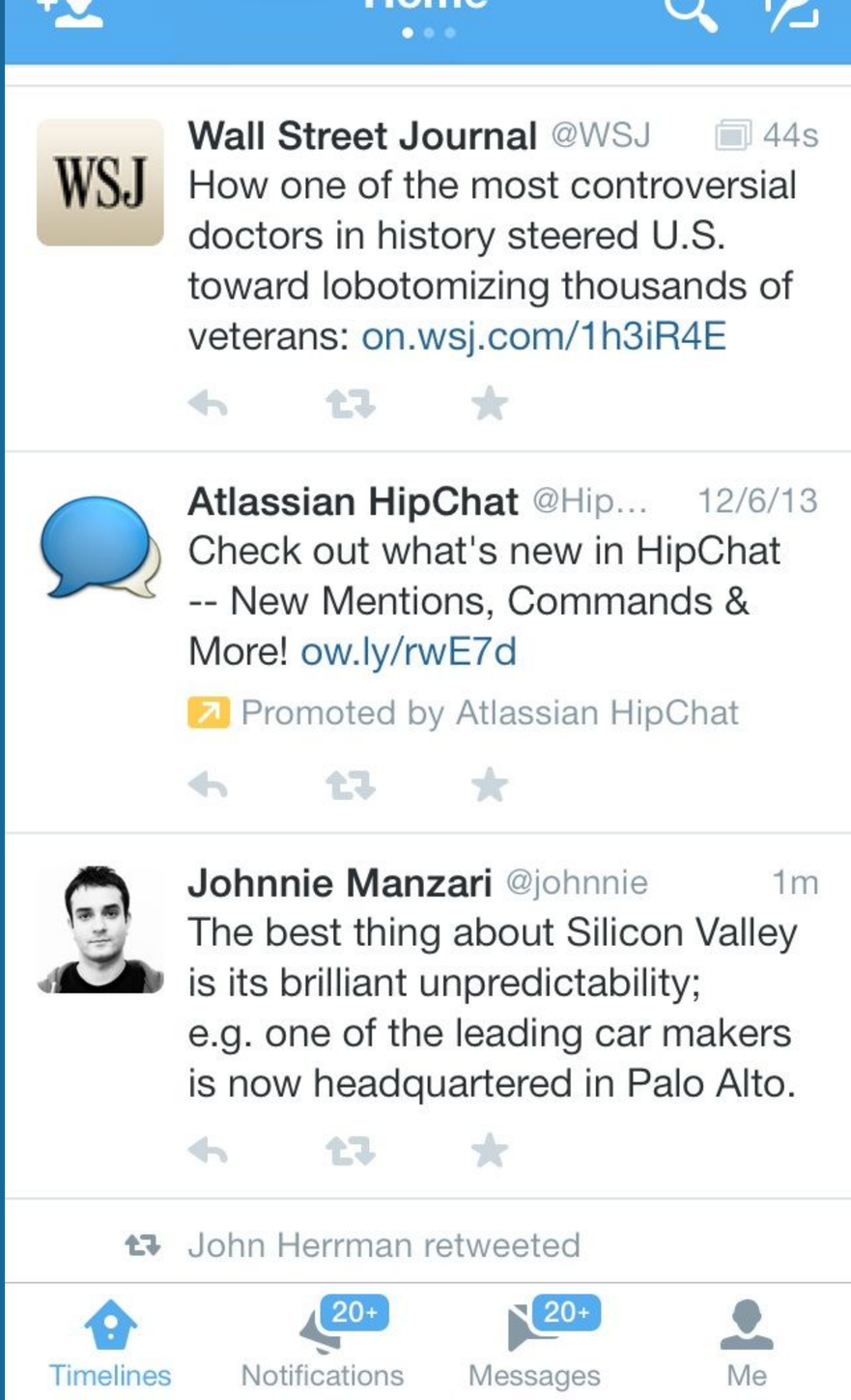


VIEW CONTROLLERS



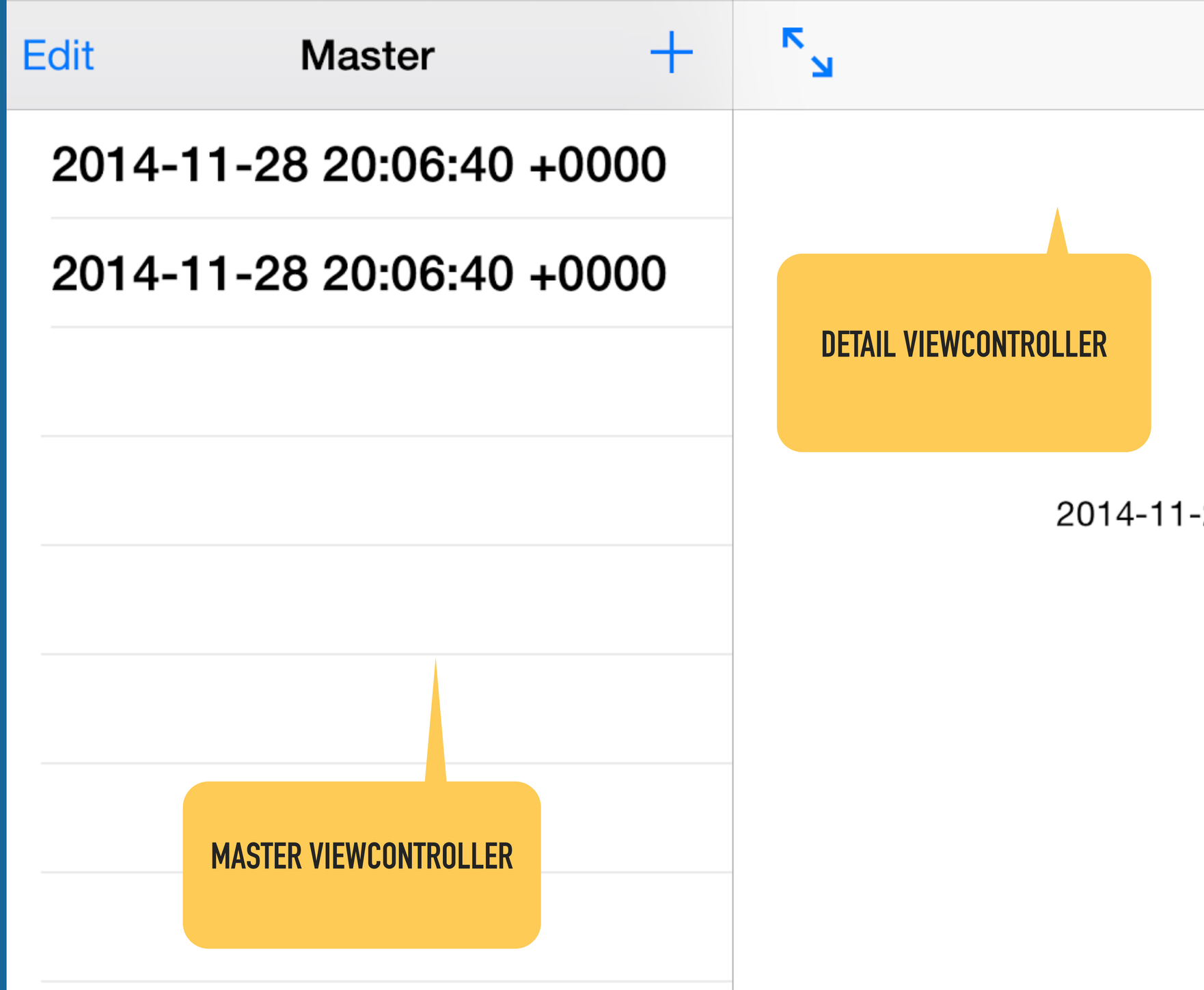
VIEW CONTROLLERS

- Special controllers manage “unit of content” not “pageful of content”
  - UISplitViewController
  - UITabBarController



AND NOW…

- The root view controller does manage a “screenful of content”
- window.rootView Controller = aViewController



VIEW CONTROLLERS

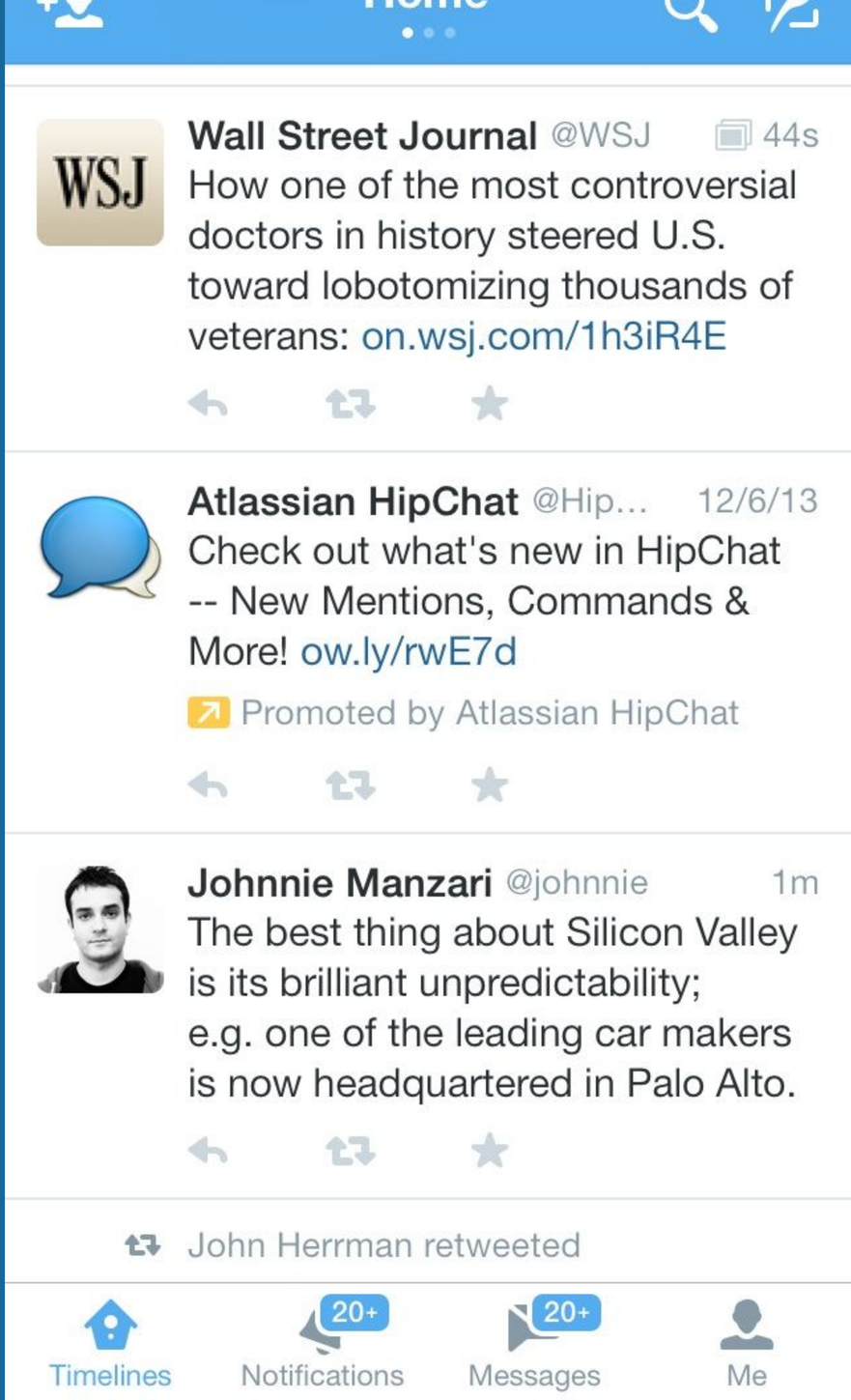
SUMMARY

- View controllers are just controllers—the “C” in MVC
- View controllers manage a view hierarchy—not a single view!
- View controllers are typically self contained (reusable)
- View controllers connect and support common iOS application flows
- More complex aesthetics or application flows need more flexibility

CONTENT VS. CONTAINER VIEW CONTROLLERS

VIEW CONTROLLER CONTAINERS

- And now...
  - Content view controller
    - Your custom view
  - Container view controllers
    - Split view controller
    - Tab bar controllers



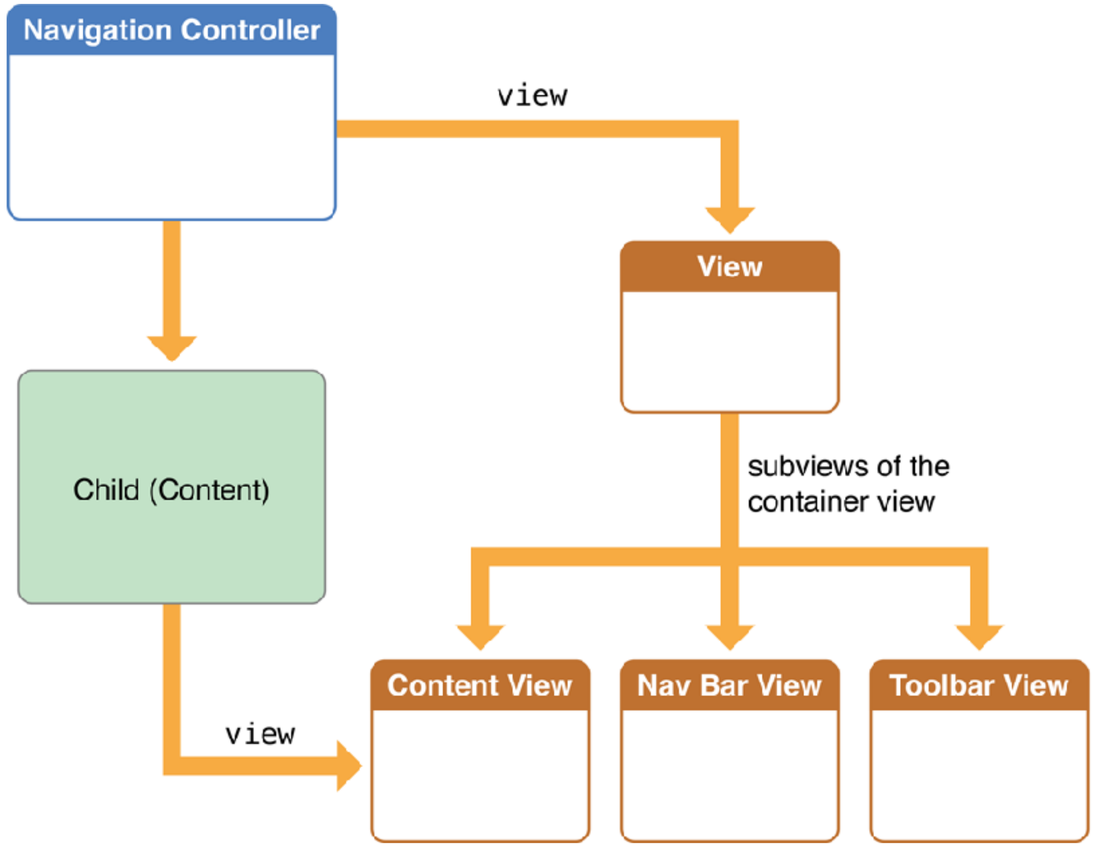
VIEW CONTROLLER CONTAINERS

- Container Controllers
  - Manage a hierarchy of child views
    - UITabBarController
    - UINavigationController
    - UISplitViewController

- Content Controllers
  - Manage individual screens
  - Can be more than one view controller per screen (as seen in a UISplitViewController)

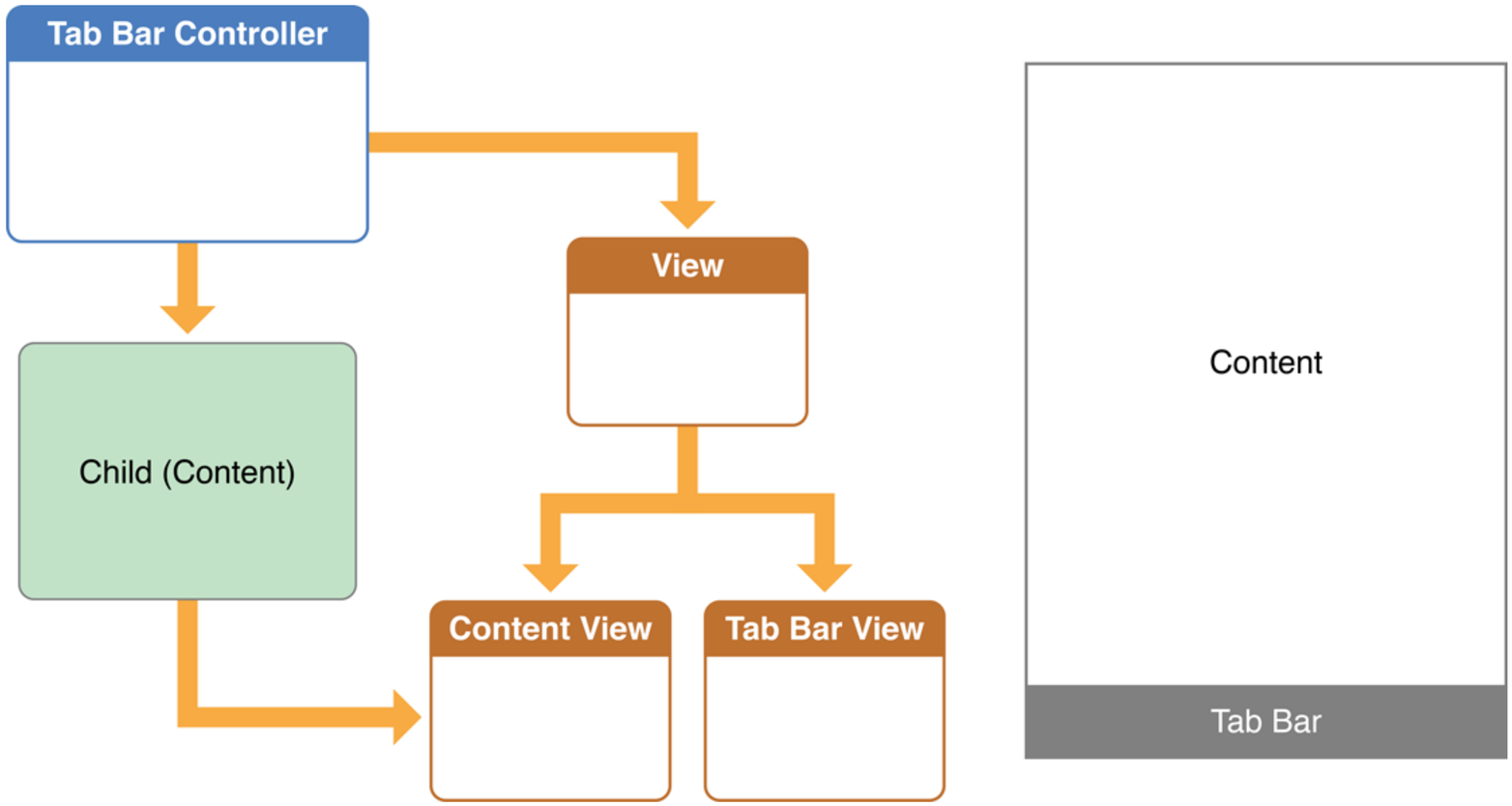
VIEW CONTROLLER CONTAINERS

- UINavigationController
  - View controllers on a stack that you push/pop them off to view
  - Transition animation where you may see multiple view controllers
  - Contain additional content views
    - Navigation bar
    - Tool bar

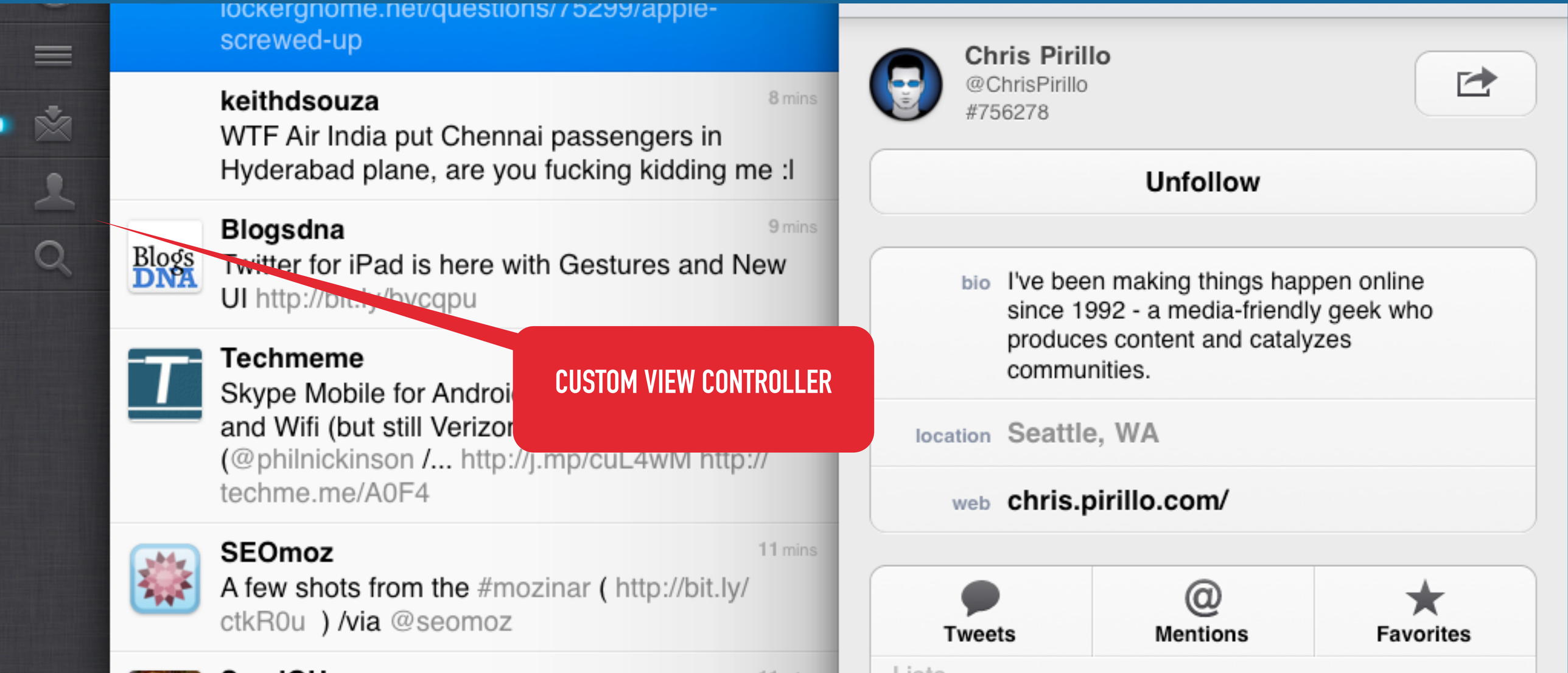


VIEW CONTROLLER CONTAINERS

- UITabBarController
  - Array of view controllers
  - Only one shown at a time

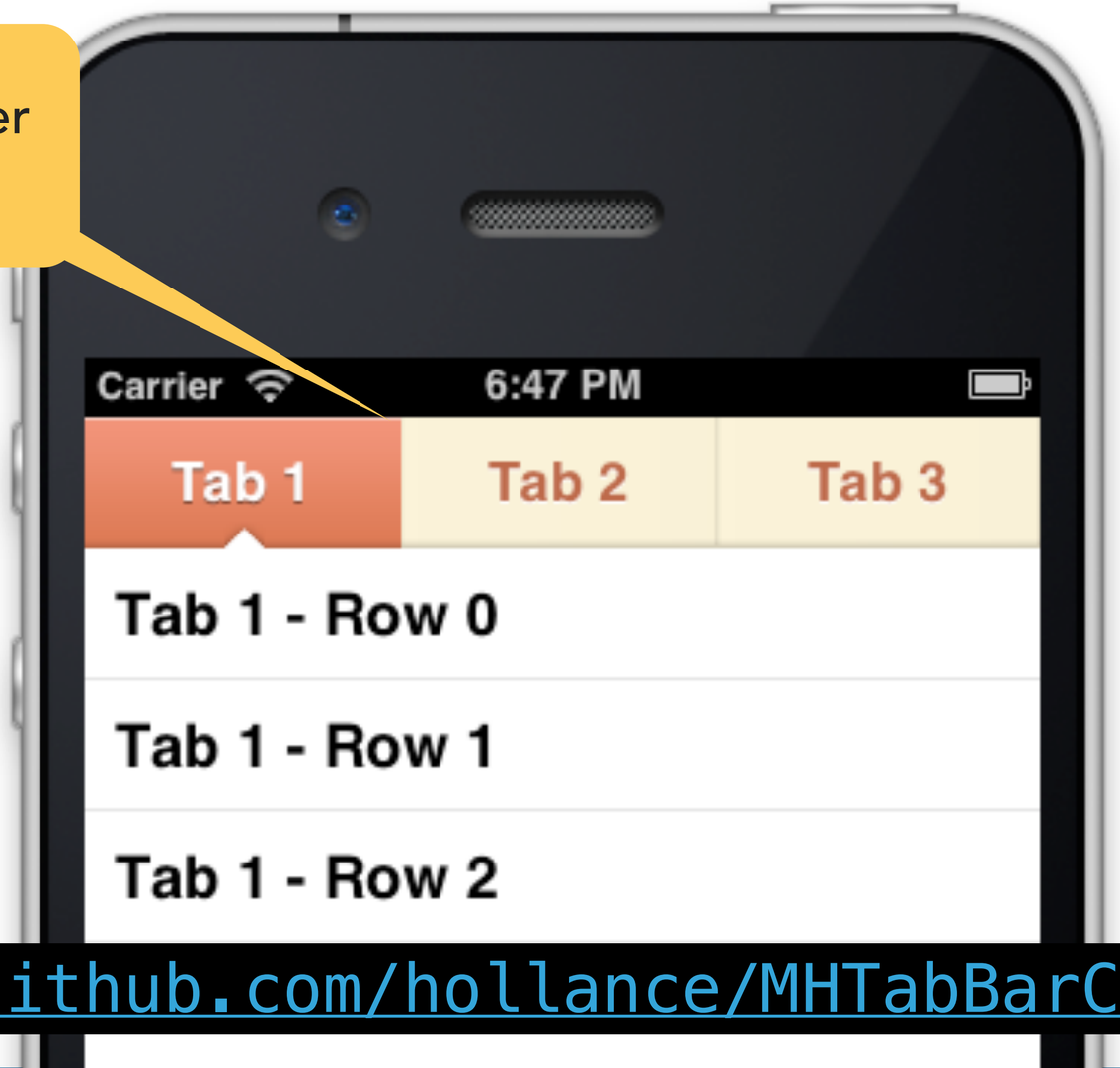


VIEW CONTROLLER CONTAINERS



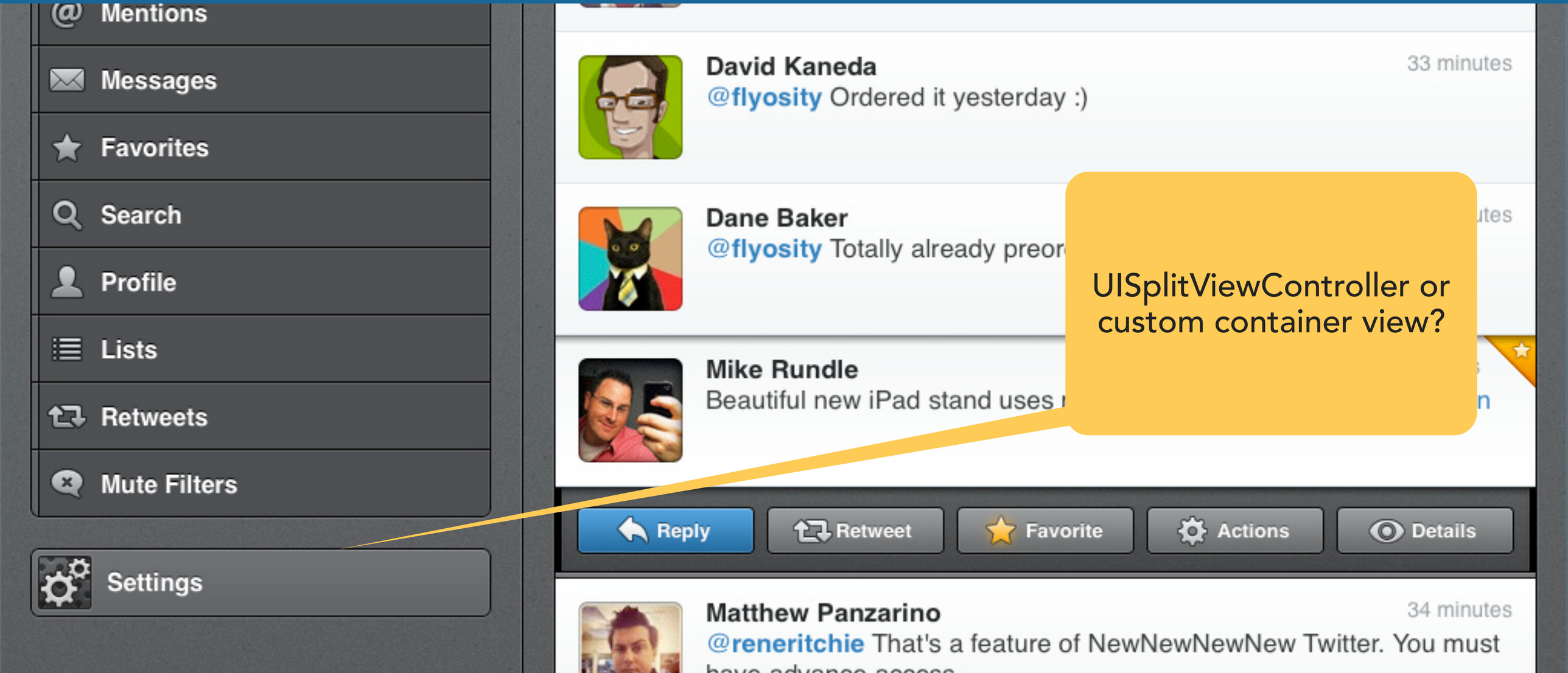
VIEW CONTROLLER CONTAINERS

custom view controller container



https://github.com/hollance/MHTabBarController

VIEW CONTROLLER CONTAINERS



VIEW CONTROLLER CONTAINMENT

VIEW CONTROLLER CONTAINERS

Blue
View color

Blue Arrow
Subview relationship

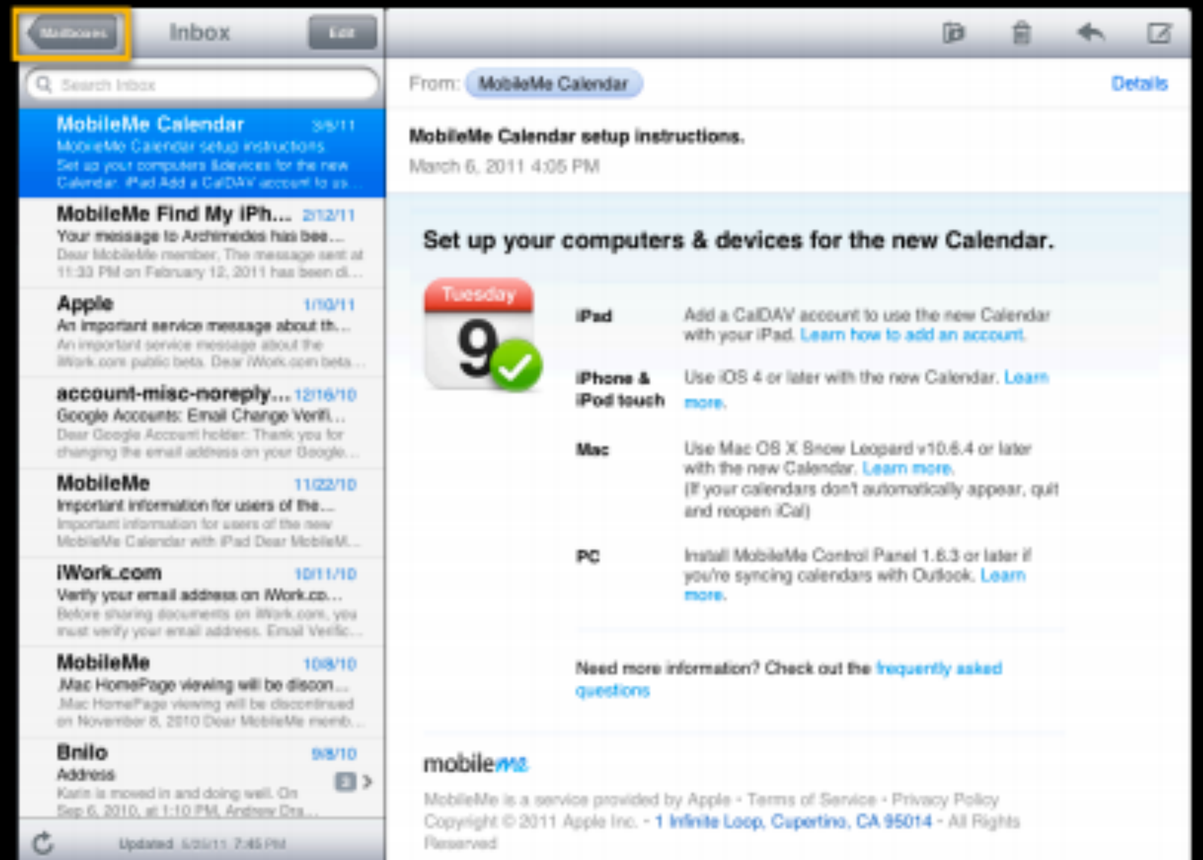
Gold
View controller color

Split View Controller

Gray Arrow
Parent view controller relationship

Image via Apple

VIEW CONTROLLER CONTAINERS



CONTENT CONTROLLER

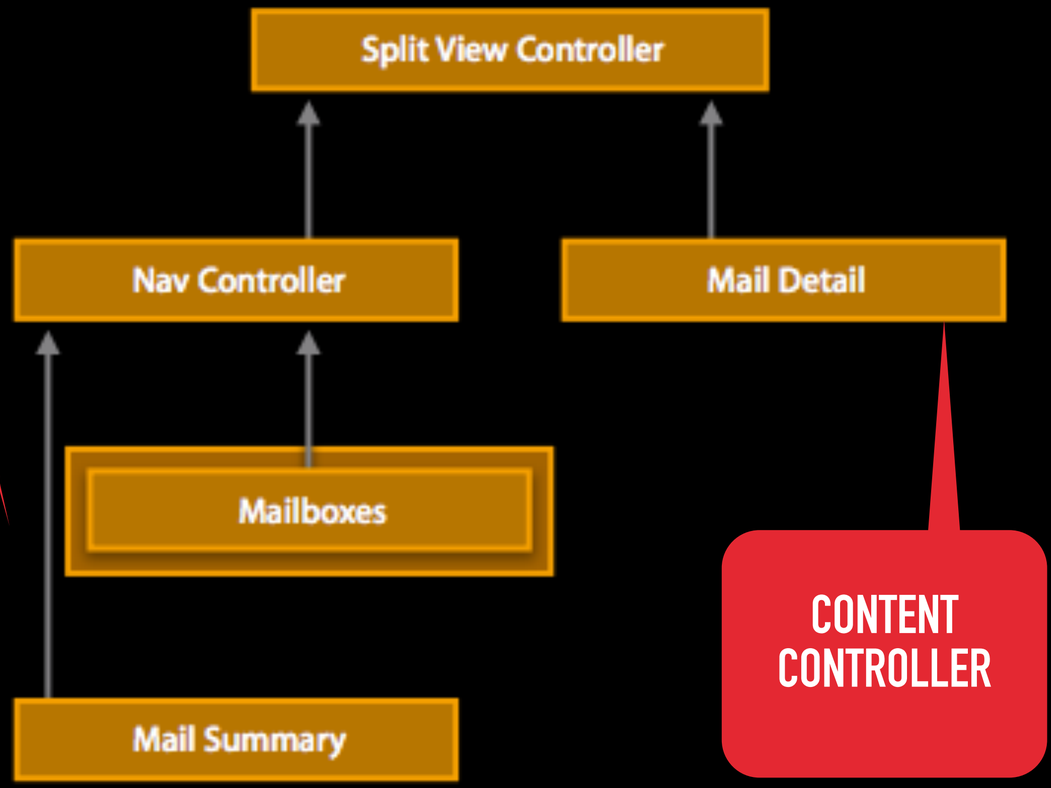


Image via Apple

VIEW CONTROLLER CONTAINERS

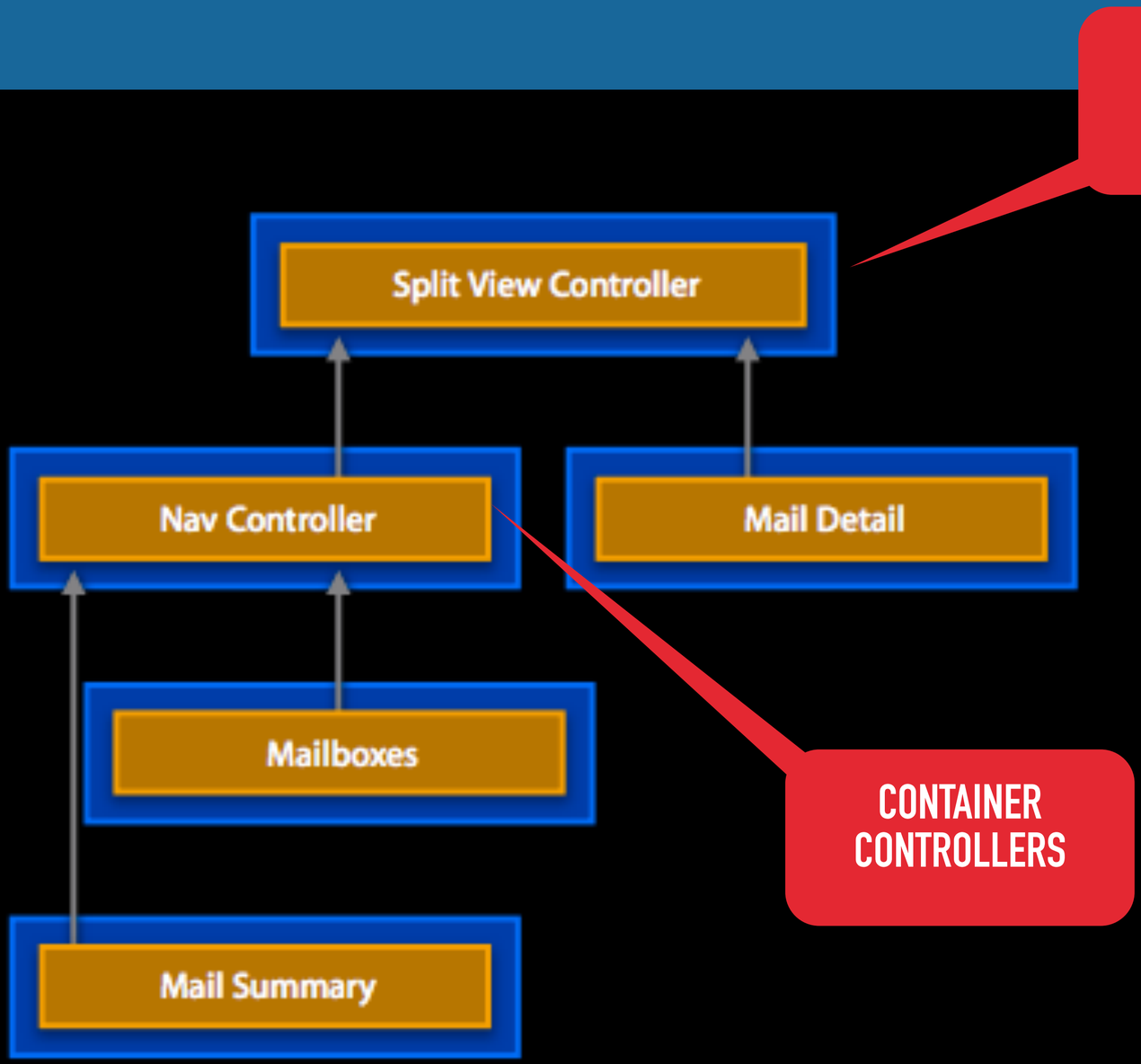
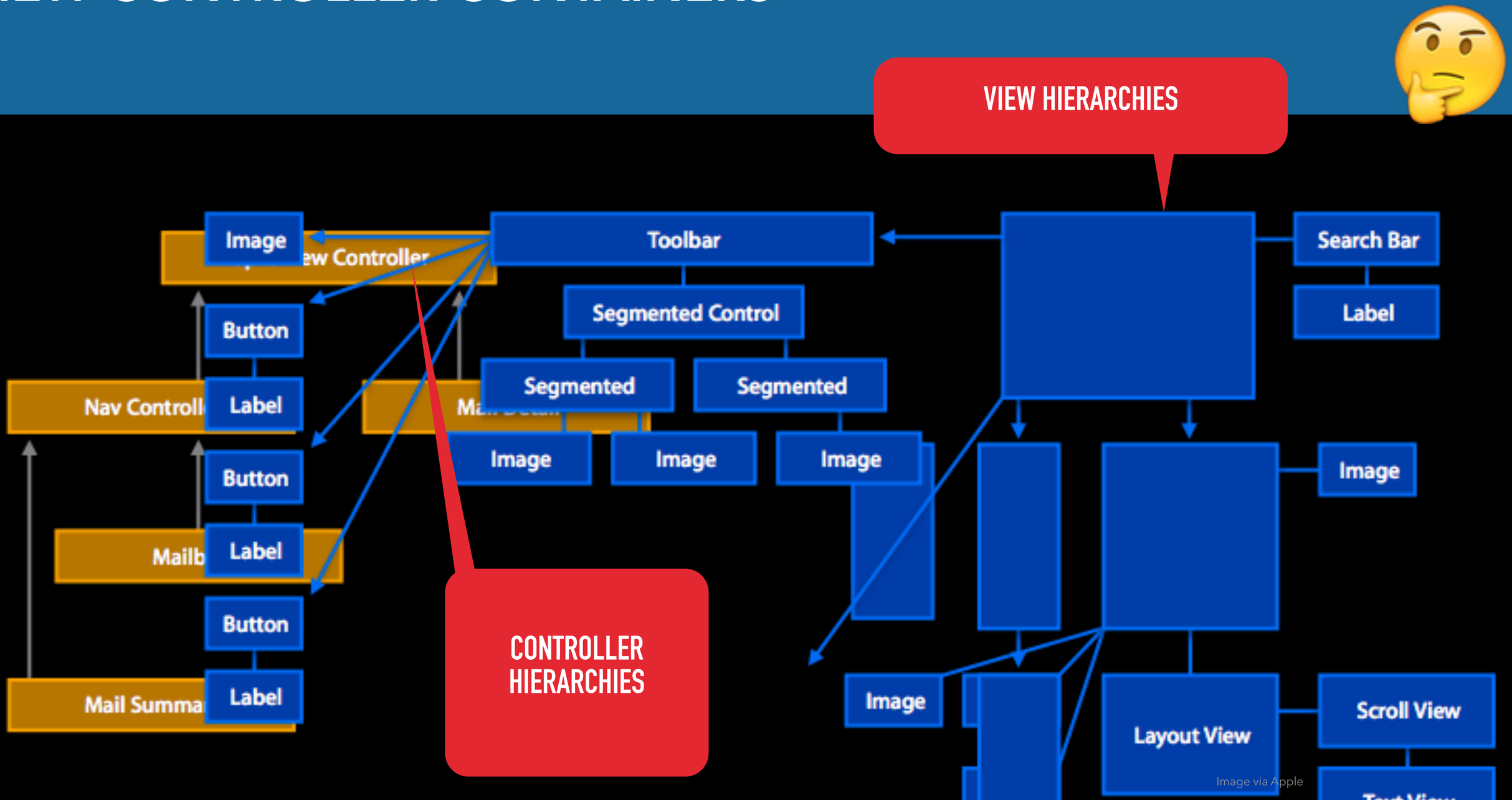


Image via Apple

VIEW CONTROLLER CONTAINERS



VIEW CONTROLLER CONTAINERS

- Container views allow a view to have child views which are in turn managed by their own view controllers

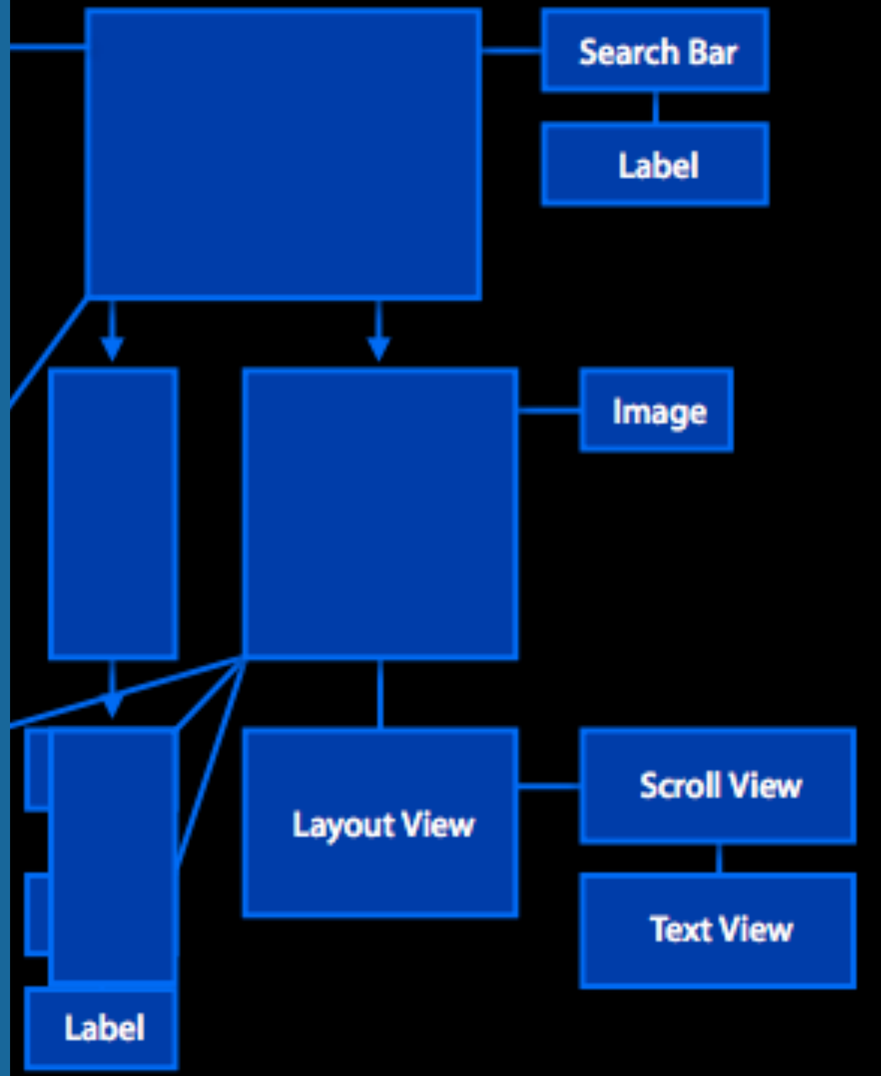


Image via Apple

VIEW CONTROLLER CONTAINERS

- Container Controllers are responsible for child/ parent relationships between view controllers
  - Pushing view controllers
  - Remove view controllers

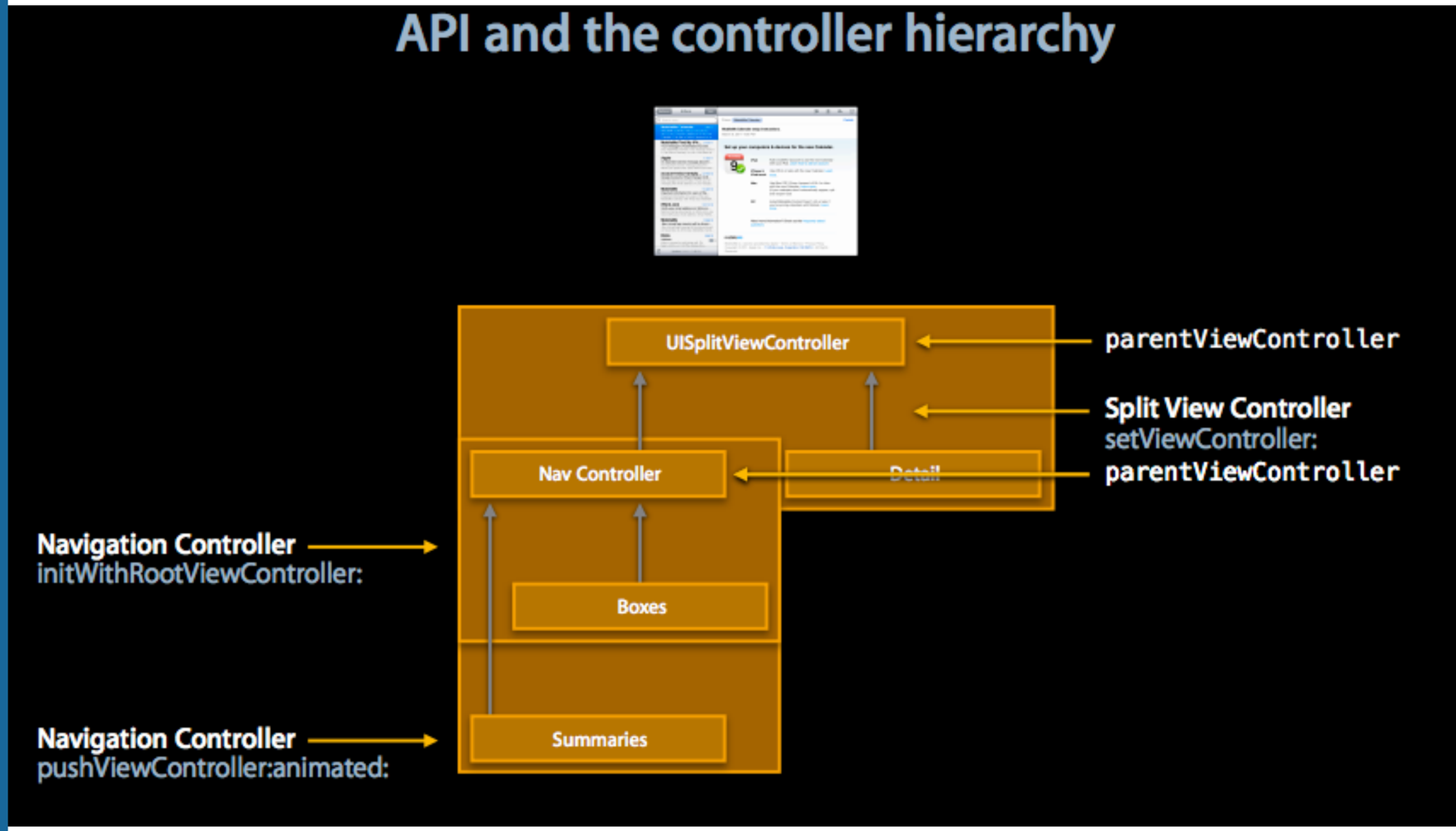


Image via Apple

VIEW CONTROLLER CONTAINERS

Tasks

- Configuring a View Controller Using Nib Files
- Interacting with Storyboards and Segues
- Managing the View
- Presenting View Controllers
- Supporting Custom Transitions and Presentations
- Responding to View Events
- Configuring the View's Layout Behavior
- Testing for Specific Kinds of View Transitions
- Configuring the View Rotation Settings
- Adapting to Environment Changes
- **Managing Child View Controllers in a Custom Container**

Language: Objective-C **Swift** Both On This Page Options

Managing Child View Controllers in a Custom Container

```
childViewControllers
addChildViewController(_:)
removeFromParentViewController()
transitionFromViewController(_:toViewController:duration:options:animations:completion:)
shouldAutomaticallyForwardAppearanceMethods()
beginAppearanceTransition(_:animated:)
endAppearanceTransition()
setOverrideTraitCollection(_:forChildViewController:)
overrideTraitCollectionForChildViewController(_:)
```

Responding to Containment Events

```
willMoveToParentViewController(_:)
```

CONNECTING VIEW CONTROLLERS

SUBTITLE

- Add a view controllers view to containers view hierarchy
  - 1. Calls the container's addChildViewController: method to add the child .
    - This calls the child's `willMoveToParentViewController:` method automatically
  - 2. It accesses the child's view property to retrieve the view and adds it to its own view hierarchy.
    - Container sets the child's size and position before adding the view; order matters
  - 3. Call the child's didMoveToParentViewController: method to signal that the operation is complete

```
// Create instance of view controller and color it.
let gvc = ColorViewController()
gvc.view.backgroundColor = UIColor.redColor()

// 1.
addChildViewController(gvc)

// 2.
gvc.view.frame = CGRectMake(50, 50, 200, 200)
view.addSubview(gvc.view)

// 3.
gvc.didMoveToParentViewController(self)
```

CONNECTING VIEW CONTROLLERS

SUBTITLE

- Add a view controllers view to containers view hierarchy
  - 1. Calls the container's addChildViewController: method to add the child .
    - This calls the child's `willMoveToParentViewController:` method automatically
  - 2. It accesses the child's view property to retrieve the view and adds it to its own view hierarchy.
    - Container sets the child's size and position before adding the view; order matters
  - 3. Call the child's didMoveToParentViewController: method to signal that the operation is complete

```
// Create instance of view controller and color it.
let gvc = ColorViewController()
gvc.view.backgroundColor = UIColor.redColor()

// 1.
addChildViewController(gvc)

// 2.
gvc.view.frame = CGRectMake(50, 50, 200, 200)
view.addSubview(gvc.view)

// 3.
gvc.didMoveToParentViewController(self)
```

CONNECTING VIEW CONTROLLERS

SUBTITLE

- Add a view controllers view to containers view hierarchy
  - 1. Calls the container's addChildViewController: method to add the child .
    - This calls the child's `willMoveToParentViewController:` method automatically
  - 2. It accesses the child's view property to retrieve the view and adds it to its own view hierarchy.
    - Container sets the child's size and position before adding the view; order matters

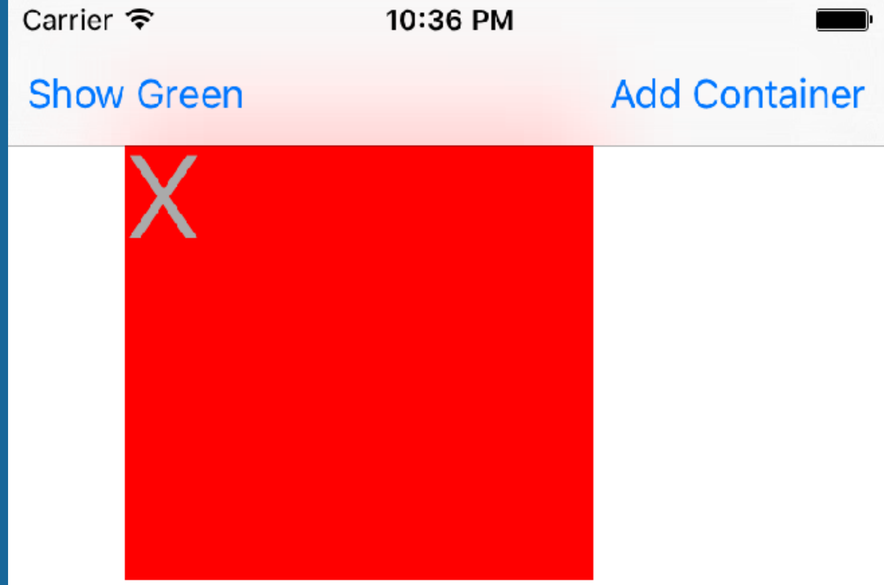
3. Call the child's didMoveToParentViewController: method to signal that the operation is complete

```
// Create instance of view controller and color it.
let gvc = ColorViewController()
gvc.view.backgroundColor = UIColor.redColor()

// 1.
addChildViewController(gvc)

// 2.
gvc.view.frame = CGRectMake(50, 50, 200, 200)
view.addSubview(gvc.view)

// 3.
gvc.didMoveToParentViewController(self)
```

CONNECTING VIEW CONTROLLERS

ADD A CHILD VIEW CONTROLLER

```
// Create instance of view controller and color it.
let gvc = ColorViewController()
gvc.view.backgroundColor = UIColor.redColor()

// 1.
addChildViewController(gvc)

// 2.
gvc.view.frame = CGRectMake(50, 50, 200, 200)
view.addSubview(gvc.view)

// 3.
gvc.didMoveToParentViewController(self)
```



CONNECTING VIEW CONTROLLERS

REMOVING A VIEW CONTROLLER FROM A CONTAINER

```
// 1. Calls the child's willMoveToParentViewController: method with a
// parameter of nil to tell the child that it is being removed.
sender.willMoveToParentViewController(nil)

// 2. Clean up the view hierarchy
sender.view.removeFromSuperview()

// 3. Calls the child's removeFromParentViewController method to remove it
// from the container.
sender.removeFromParentViewController()
```

CONNECTING VIEW CONTROLLERS

REMOVING A VIEW CONTROLLER FROM A CONTAINER

- Remove a view controllers view to containers view hierarchy
  - 1. Calls the child's `willMoveToParentViewController:` method with a parameter of nil to tell the child that it is being removed.
  - 3. Calling the `removeFromParentViewController` method automatically calls the child's `didMoveToParentViewController:` method

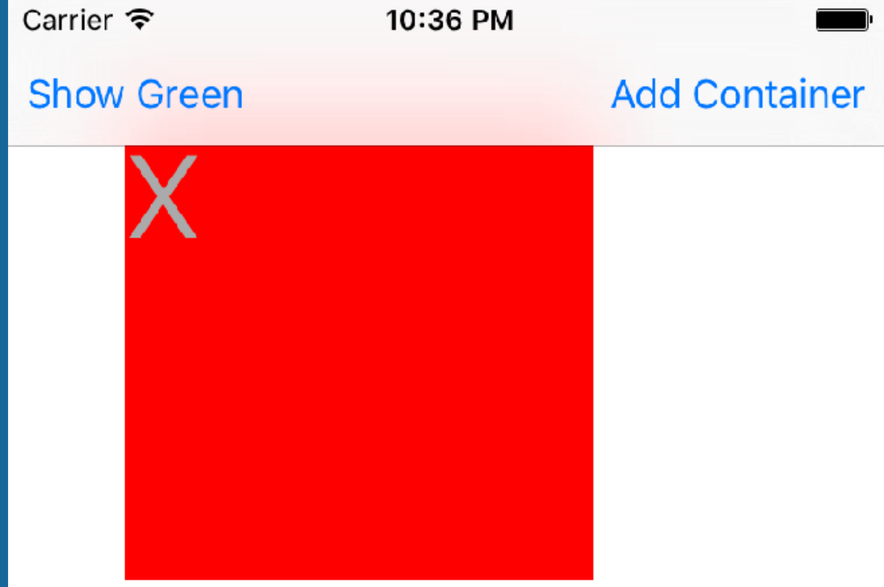
```
// 1. Calls the child's willMoveToParentViewController: method
// parameter of nil to tell the child that it is being removed
sender.willMoveToParentViewController(nil)

// 2. Clean up the view hierarchy
sender.view.removeFromSuperview()

// 3. Calls the child's removeFromParentViewController method
// from the container.
sender.removeFromParentViewController()
```

CONNECTING VIEW CONTROLLERS

SUBTITLE

- But do they know about each other?
  - Not if you’re practicing good MVC

- Possible ways to pass messages
  - NSNotifications
  - KVO
  - Delegation



CONNECTING VIEW CONTROLLERS

- Create a protocol to allow communication between parent and child

child view controller

```
import UIKit

///
/// Protocol that allows the remove child view controller cycle to be called
/// by the parent view controller
///
protocol ColorViewControllerDelegate: class {
  func removeFromContainerViewController(sender: ColorViewController)
}


///
/// View controller that can either dismiss itself or tell a parent view
/// controller to remove it
///
class ColorViewController: UIViewController {

  /// Keep a reference to the parent view controller
  weak var delegate: ColorViewControllerDelegate?


  //
  // MARK: - IBActions
  //

  @IBAction func tapCloseButton(sender: UIButton) {

    if parentViewController != nil {
      delegate?.removeFromContainerViewController(self)
    }



  }

  // MARK: - Life
  //
```

CONNECTING VIEW CONTROLLERS

- Tapping on the button in the child will remove it from the parent

```
protocol ColorViewControllerDelegate: class {...}


///
/// View controller that can either dismiss itself or tell a parent view
/// controller to remove it
///
class ColorViewController: UIViewController {

  /// Keep a reference to the parent view controller
  weak var delegate: ColorViewControllerDelegate?

  //
  // MARK: - IBActions
  //

  @IBAction func tapCloseButton(sender: UIButton) {

    if parentViewController != nil {
      delegate?.removeFromContainerViewController(self)
    }



  }

  //
  // MARK: - Life
  //

  override func viewDidLoad() {...}



}
```

CONNECTING VIEW CONTROLLERS

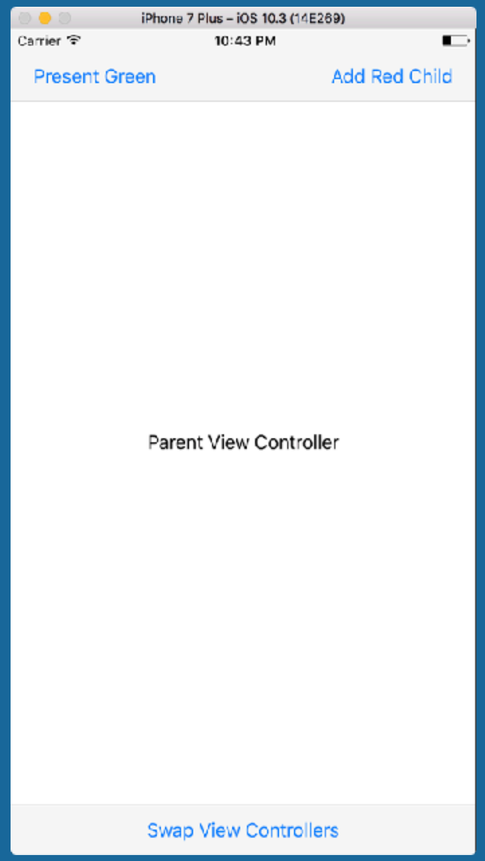
- More versatile implementation
- Determine if under containment or content view controller

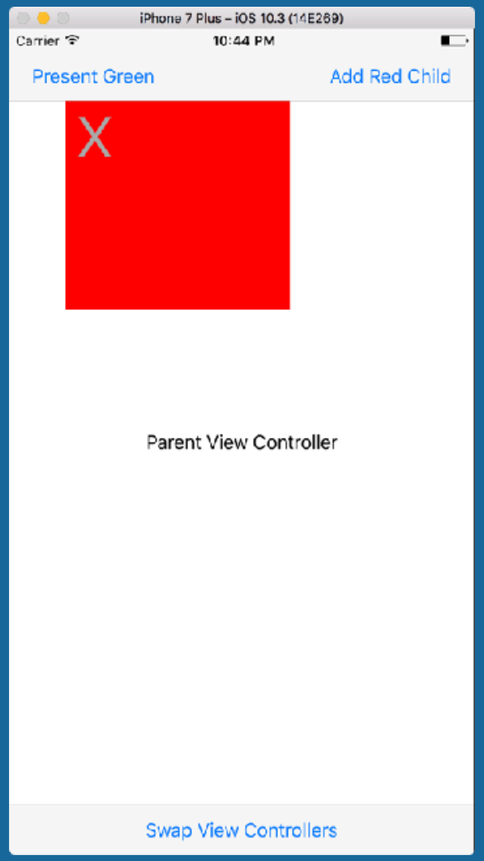
```
///
protocol ColorViewControllerDelegate: class {...}


///
/// View controller that can either dismiss itself or tell a parent view
/// controller to remove it
///
class ColorViewController: UIViewController {

  /// Keep a reference to the parent view controller
  weak var delegate: ColorViewControllerDelegate?

  //
  // MARK: - IBActions
  //

  @IBAction func tapCloseButton(sender: UIButton) {

    if parentViewController != nil {
      delegate?.removeFromContainerViewController(self)
    } else {
      presentingViewController?.dismissViewControllerAnimated(true, completion: nil)
    }

  }

  //
  // MARK: - Life
  //

  override func viewDidLoad() {...}



}
```

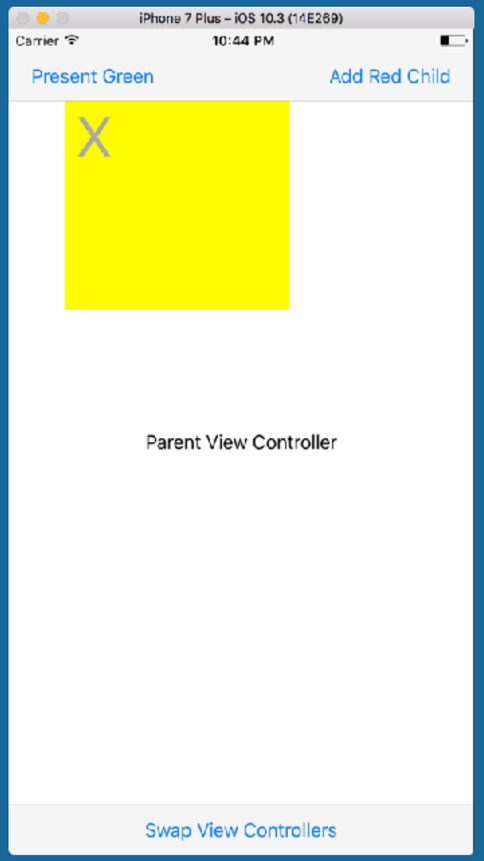
CHILD VIEW CONTROLLER VIEW TRANSITIONS

VIEW CONTROLLER CONTAINERS

- Transitioning between children view controllers

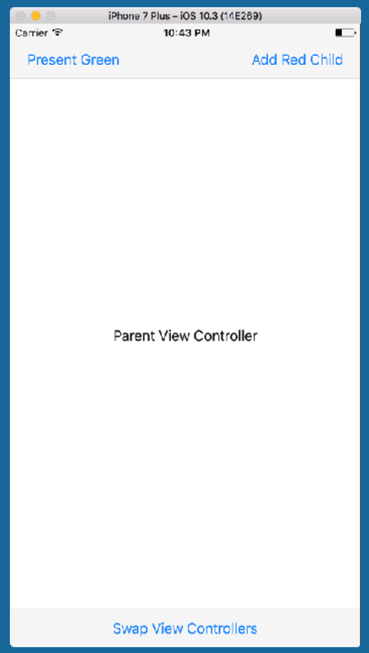






VIEW CONTROLLER CONTAINERS

- Transitioning between children view controllers

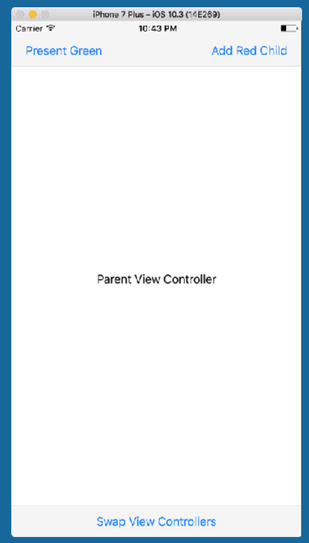


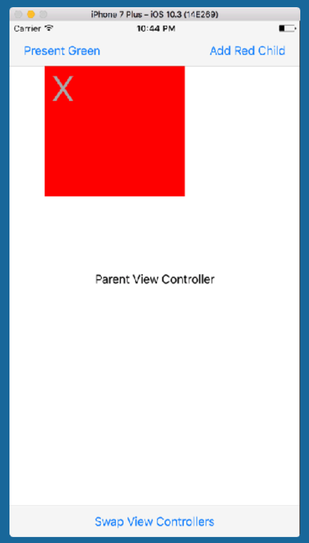

```
@IBAction func tapSwapViewControllers(_ sender: UIBarButtonItem) {
  // Create a new view controller (yellow color)
  let gvc = ColorViewController(nibName: "GreenViewController", bundle: nil)
  gvc.delegate = self
  gvc.view.backgroundColor = UIColor.yellow
  gvc.view.frame = CGRect(x: 50, y: 50, width: 200, height: 200)
  gvc.view.alpha = 0.0

  // The view controllers have to have the same parent...but
  // do not add it to the view hierarchy
  addChildViewController(gvc)

  // transition between them; fade out/ fade in
  self.transition(from: gvc2!, to: gvc, duration: 2.0,
                  options: UIViewAnimationOptions.curveEaseInOut,
                  animations: {
                    self.gvc2?.view.alpha = 0.0
                    gvc.view.alpha = 1.0

  }) { (animated) in
    print("done")
  }
}
```

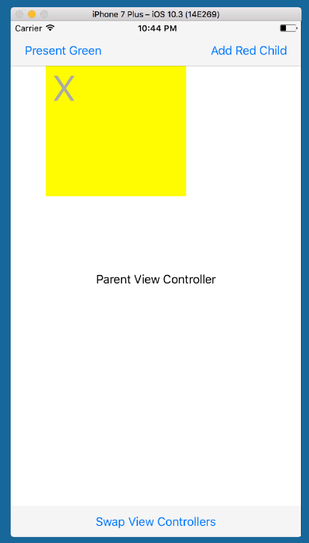
VIEW CONTROLLER CONTAINERS

CONTAINER VIEW

- Make you own tab bar







-

BUTTONS

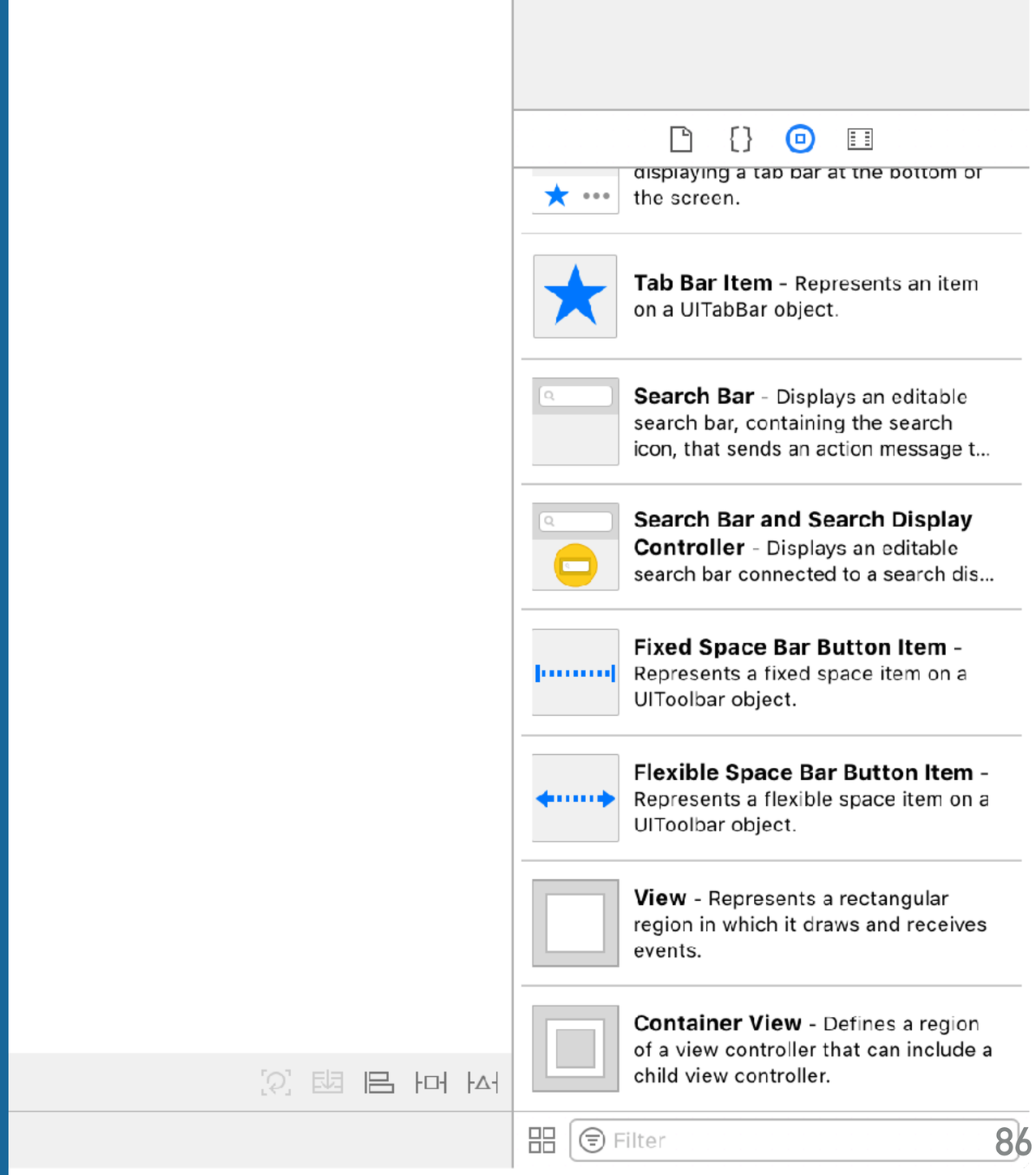
HOME. USERS. SETTINGS.

VIEW CONTROLLER CONTAINMENT USING INTERFACE BUILDER

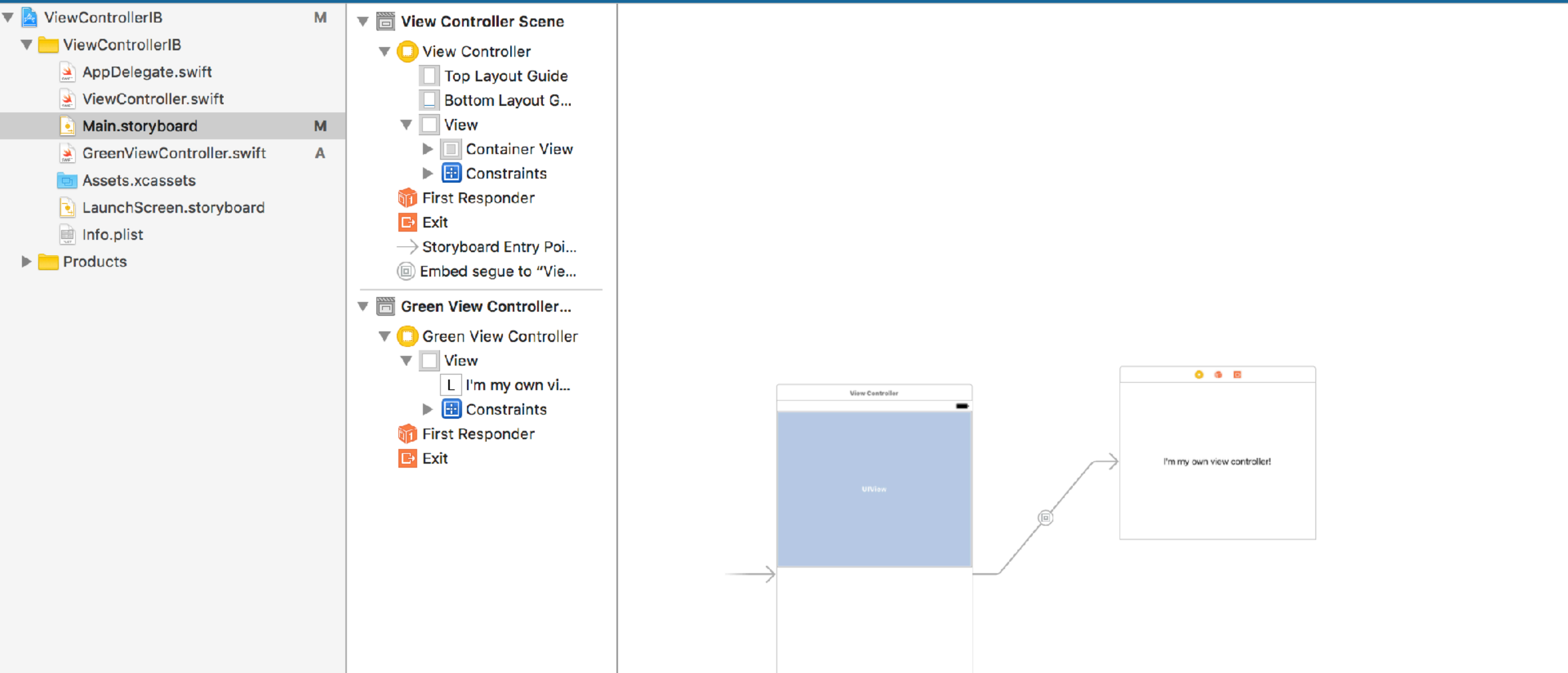
VIEW CONTROLLER CONTAINMENT USING INTERFACE BUILDER

SUBTITLE

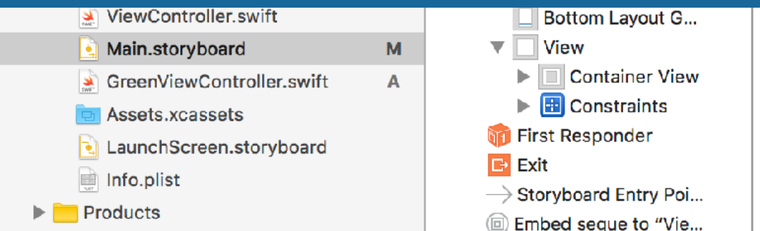
- Add Container Views to a View Controller in Interface Builder
- Automatically performs the parent-child dance for you

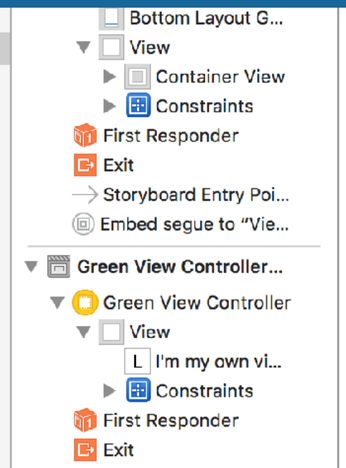


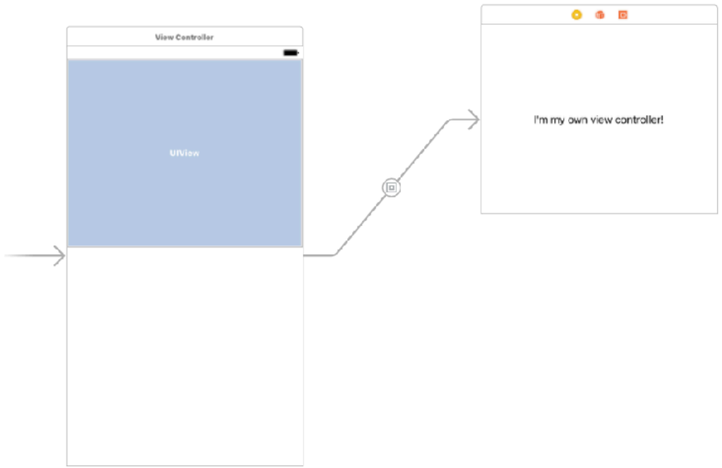
VIEW CONTROLLER CONTAINMENT USING INTERFACE BUILDER

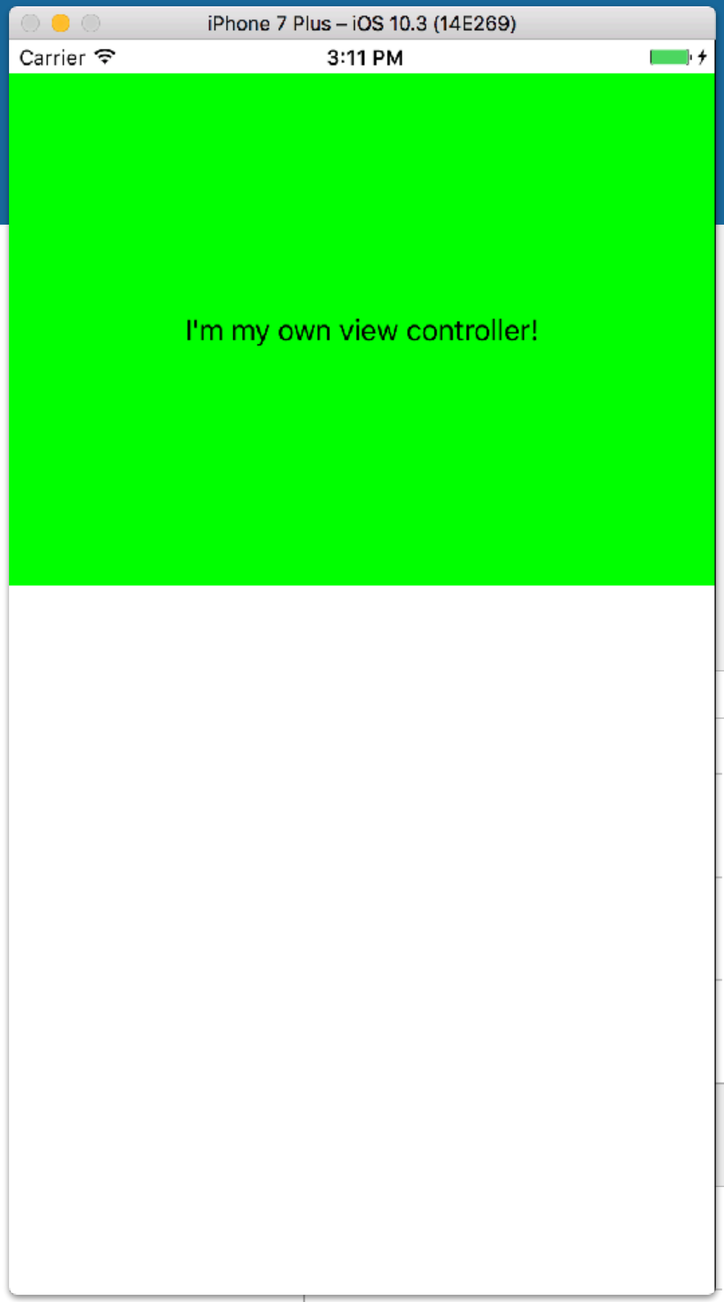


VIEW CONTROLLER CONTAINMENT USING INTERFACE BUILDER









SOME IMPORTANT THINGS TO REMEMBER ABOUT CONTAINMENT

WHAT YOU NEED TO KNOW ABOUT HIERARCHIES

- Container controllers are responsible for child/parent relationships
- There are consistent and inconsistent hierarchies
  - `UIViewControllerHierarchyInconsistencyException`
    - The view associated with a view controller was added directly to a view hierarchy
    - Pointing to view controller that are not in that view hierarchy
- Appearance callbacks are different than content view controllers

WHAT YOU NEED TO KNOW ABOUT HIERARCHIES

- Appearance callbacks
  - `addChildViewController` does not trigger appearance callbacks (e.g. viewDid/WillAppear)
  - This view controller is a child (nothing to do with appearance)
- Appearance callbacks called when they move in/out of the window hierarchy
  - Not necessarily when you “see” it
  - Add a view “behind” a view and you do not see
  - Think of it as “Made it into the window’s view hierarchy”

WHAT YOU NEED TO KNOW ABOUT HIERARCHIES

APPEARANCE CALLBACKS

- viewWillAppear:
  - ▪ Called before the view is added to the windows' view hierarchy
  - ▪ Called before `view.layoutSubviews()` (if necessary)
- viewDidAppear:
  - Called after the view is added to the view hierarchy
  - Called after view.layoutSubviews() (if necessary)

- viewWillDisappear:
  - Called before the view is removed from the windows' view hierarchy
- viewDidDisappear:
  - Called after the view is removed from the windows' view hierarchy

WHAT YOU NEED TO KNOW ABOUT HIERARCHIES

- When should you create a custom view controller?
  - Never, use Apple’s built in containers
  - Design, you want custom aesthetic
  - Function, new application flow

WHAT YOU NEED TO KNOW ABOUT HIERARCHIES

- Apple is updating container controllers
  - iOS5 UISplitViewController always shows master view controller
  - iOS7 Custom ViewController transitions (modal vc on iPhone)
  - iOS8 Custom presentation controllers
  - ...
  - iOS14 UIHamburger??
  - Not is SWIFTUI (yet)

WHAT YOU NEED TO KNOW ABOUT HIERARCHIES

- Practical suggestions for building a container view controller (from Apple)
  - Design as content view controller first
  - Never access any view other than the top-level view of the child view controller (i.e. don’t expose unnecessary details to children)
  - The container should use protocols to declare methods and properties

THE UNIVERSITY OF CHICAGO

ADVANCED iOS APPLICATION DEVELOPMENT

MPCS 51032 • SPRING 2020 • SESSION 1A