



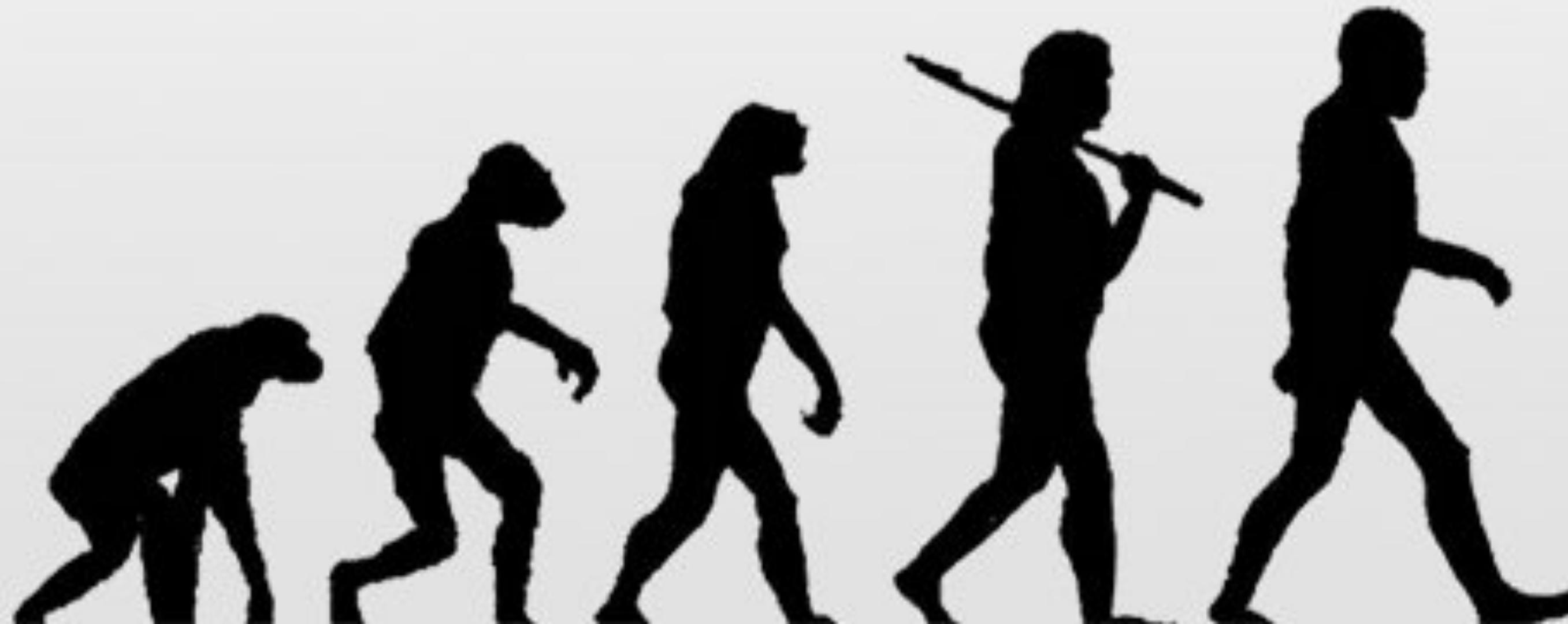
ADVANCED iOS APPLICATION DEVELOPMENT

MPCS 51032 • SPRING 2020 • SESSION 1A

VIEW CONTROLLER EVOLUTION

VIEW CONTROLLER EVOLUTION

iPhone
Evolution



THE VIEW CONTROLLER

THE VIEW CONTROLLER

IN THE BEGINNING...

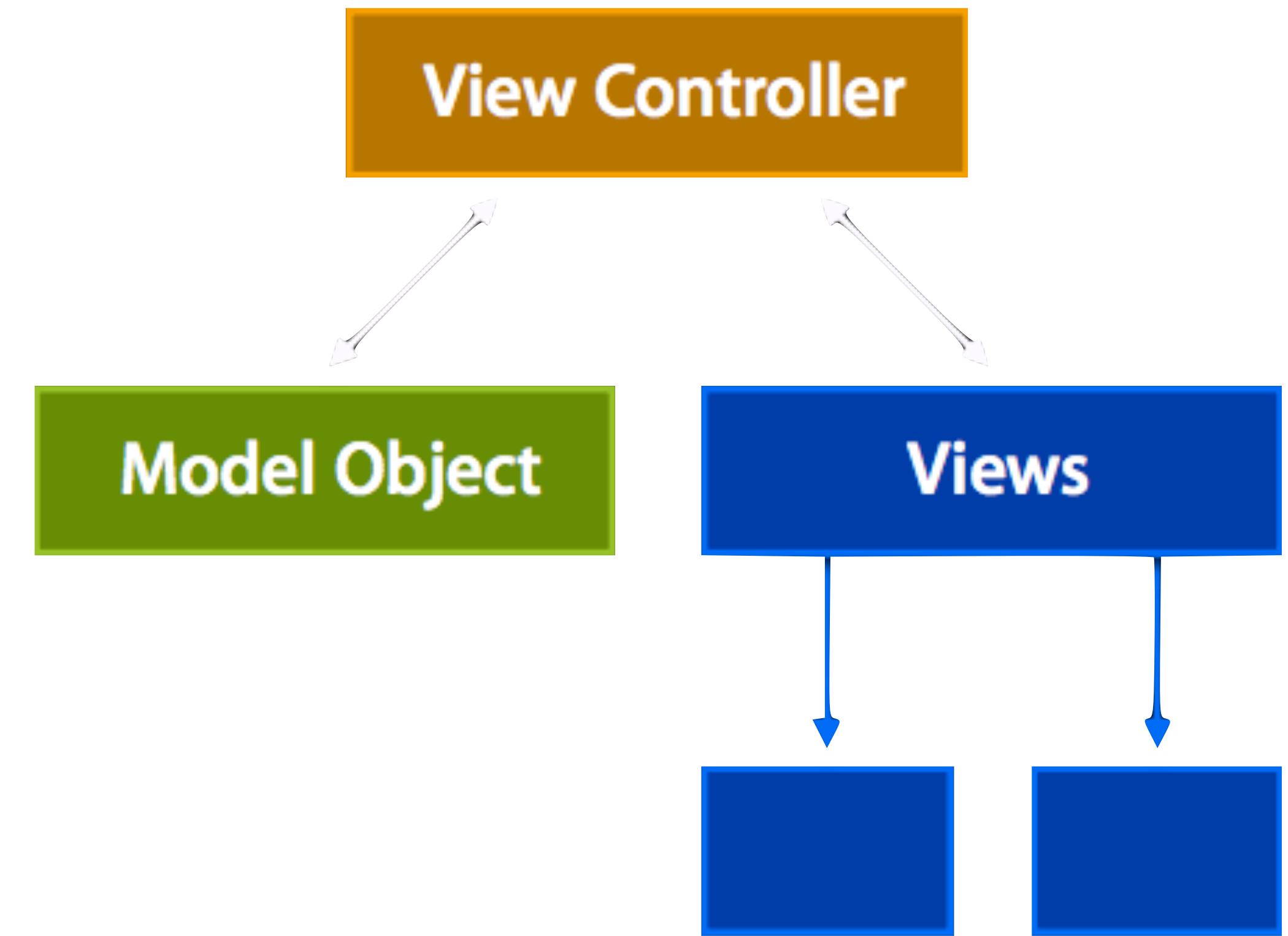
- `UIViewController`s jobs
 - Manages “screenful of objects”
 - Manage view hierarchy
 - Applications flow between view controllers



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



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)
```

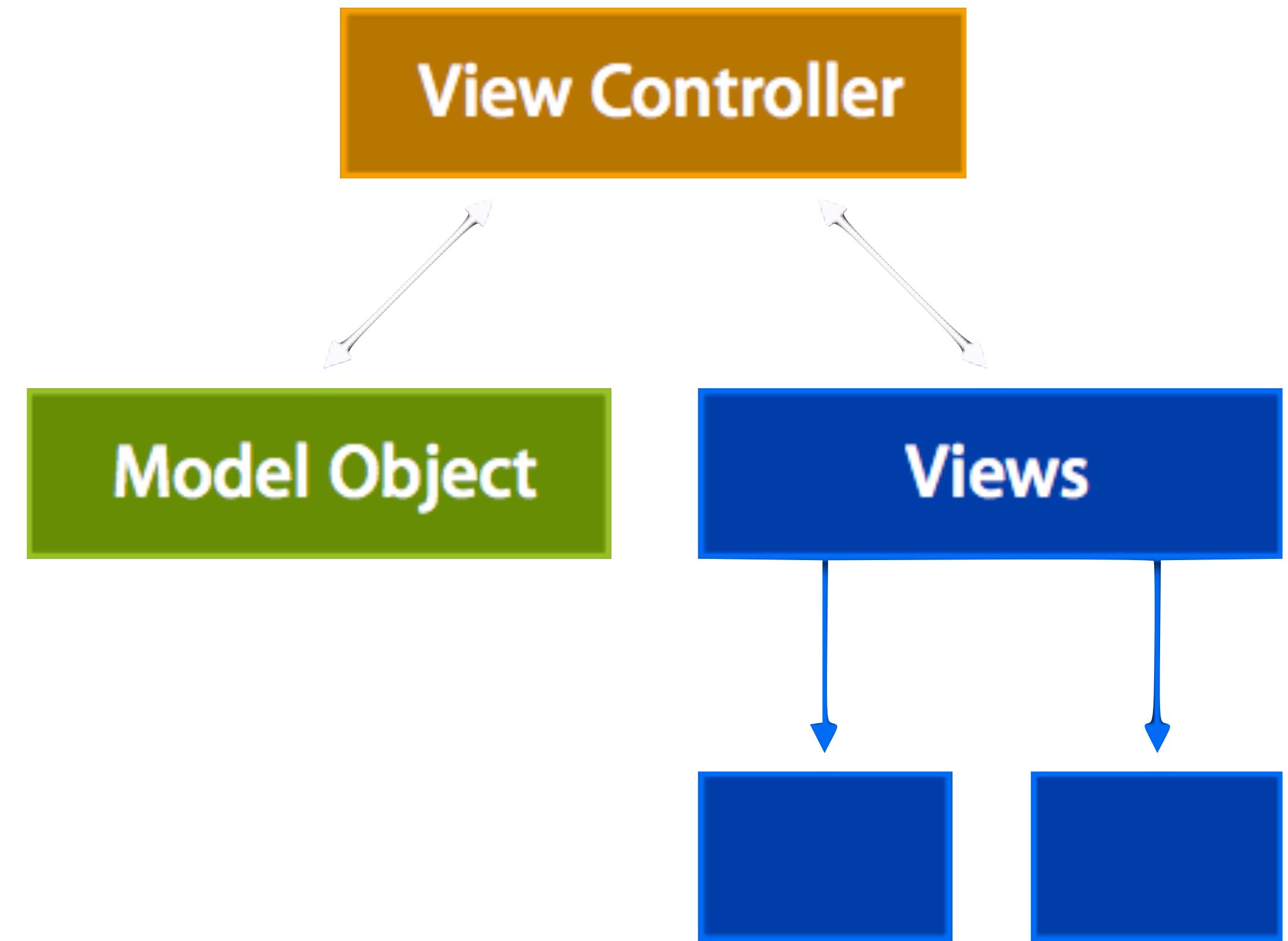
- By direct view manipulation

```
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



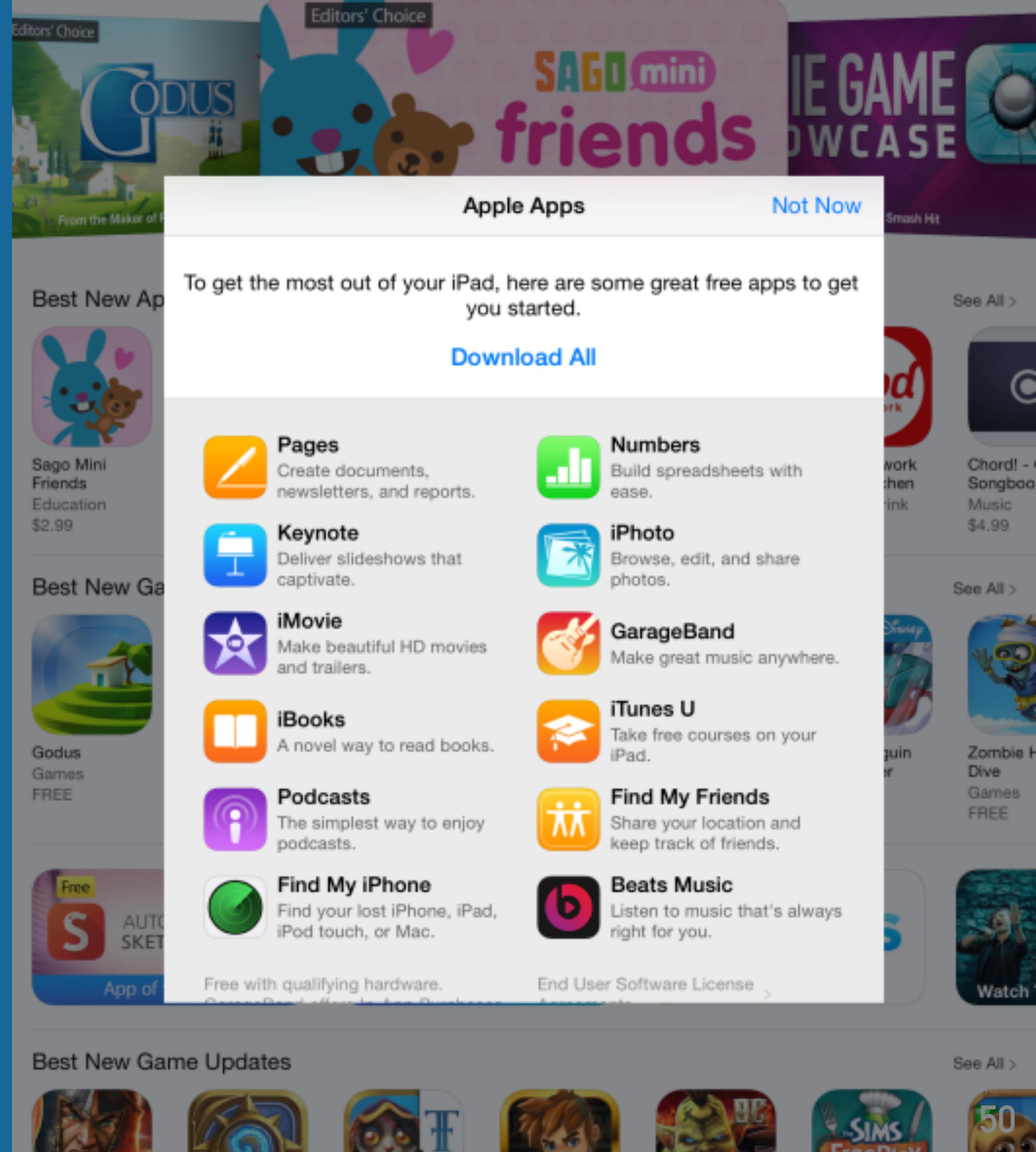
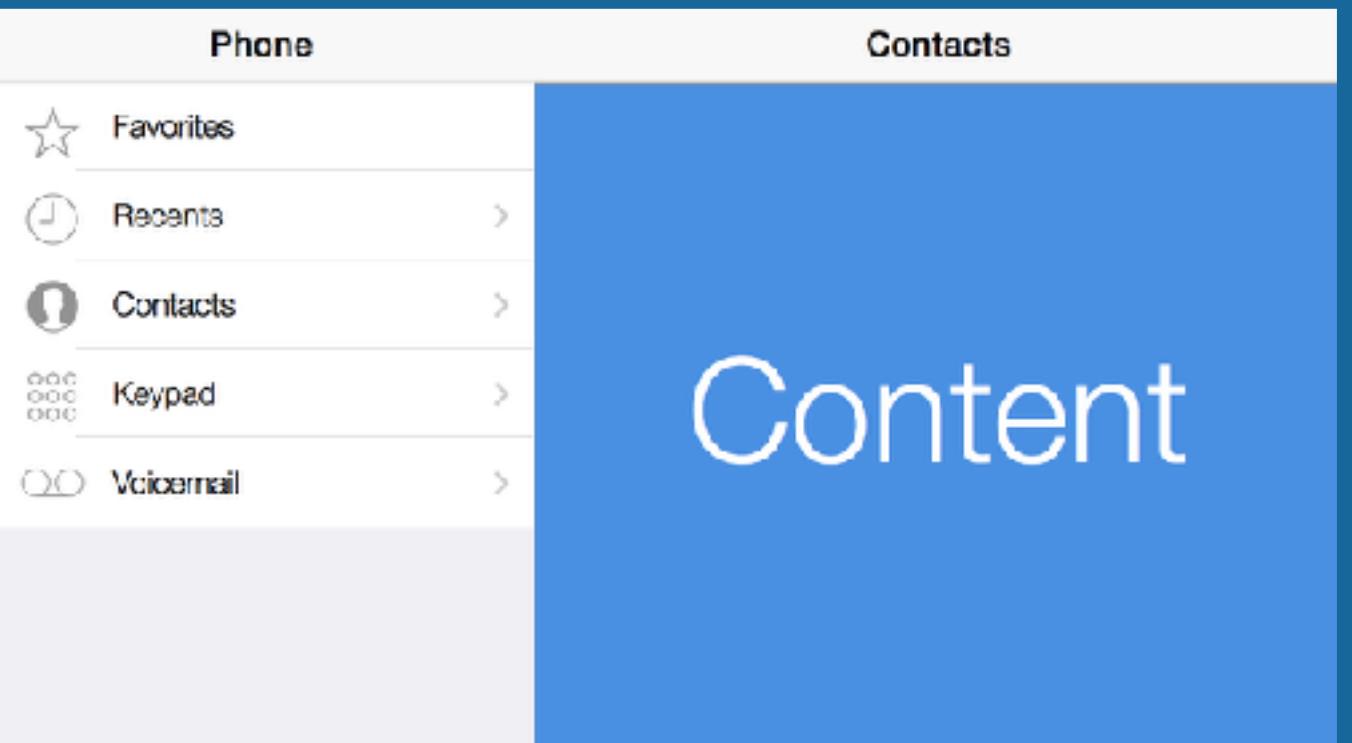
VIEW CONTROLLERS

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

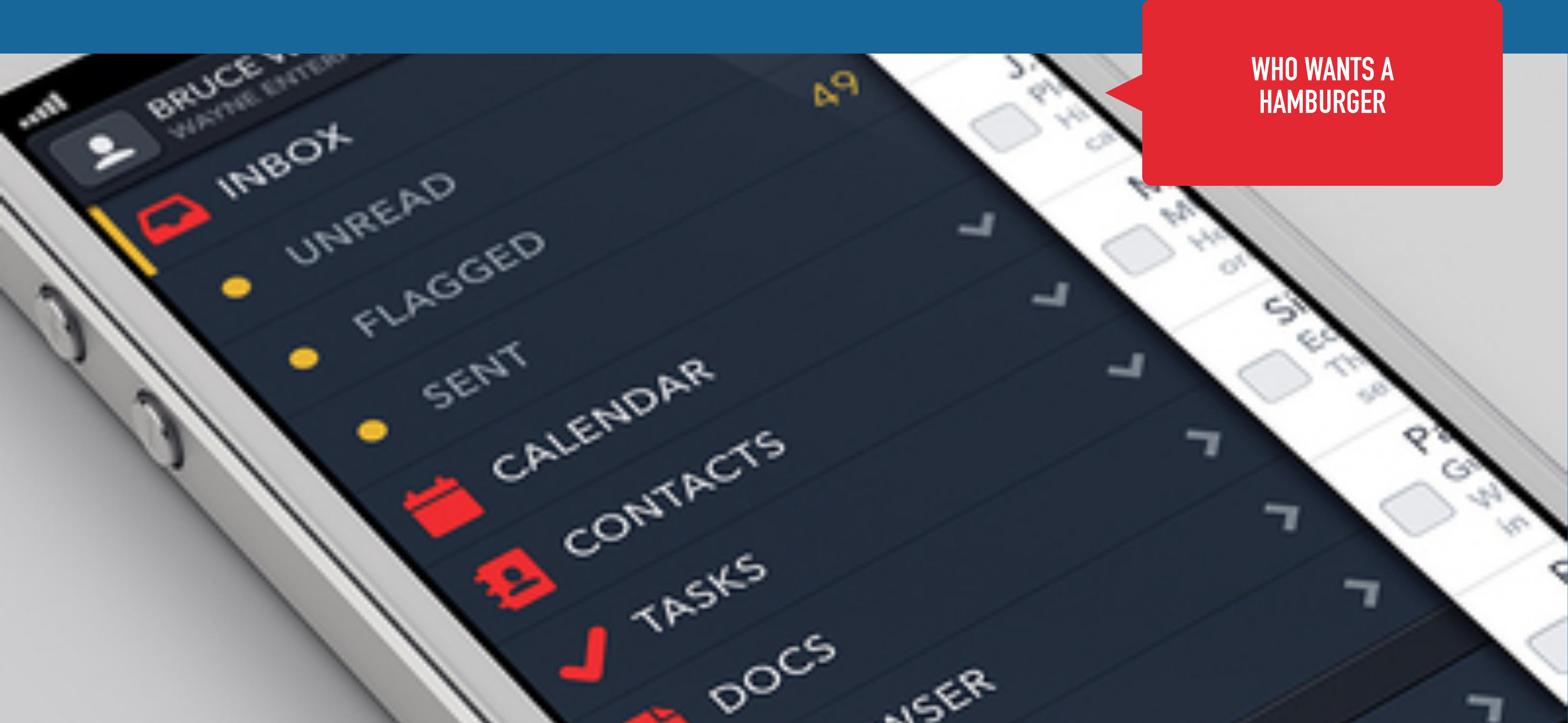


VIEW CONTROLLERS

- “Screenful of content” getting harder to define



VIEW CONTROLLERS



WHO WANTS A
HAMBURGER

VIEW CONTROLLERS

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

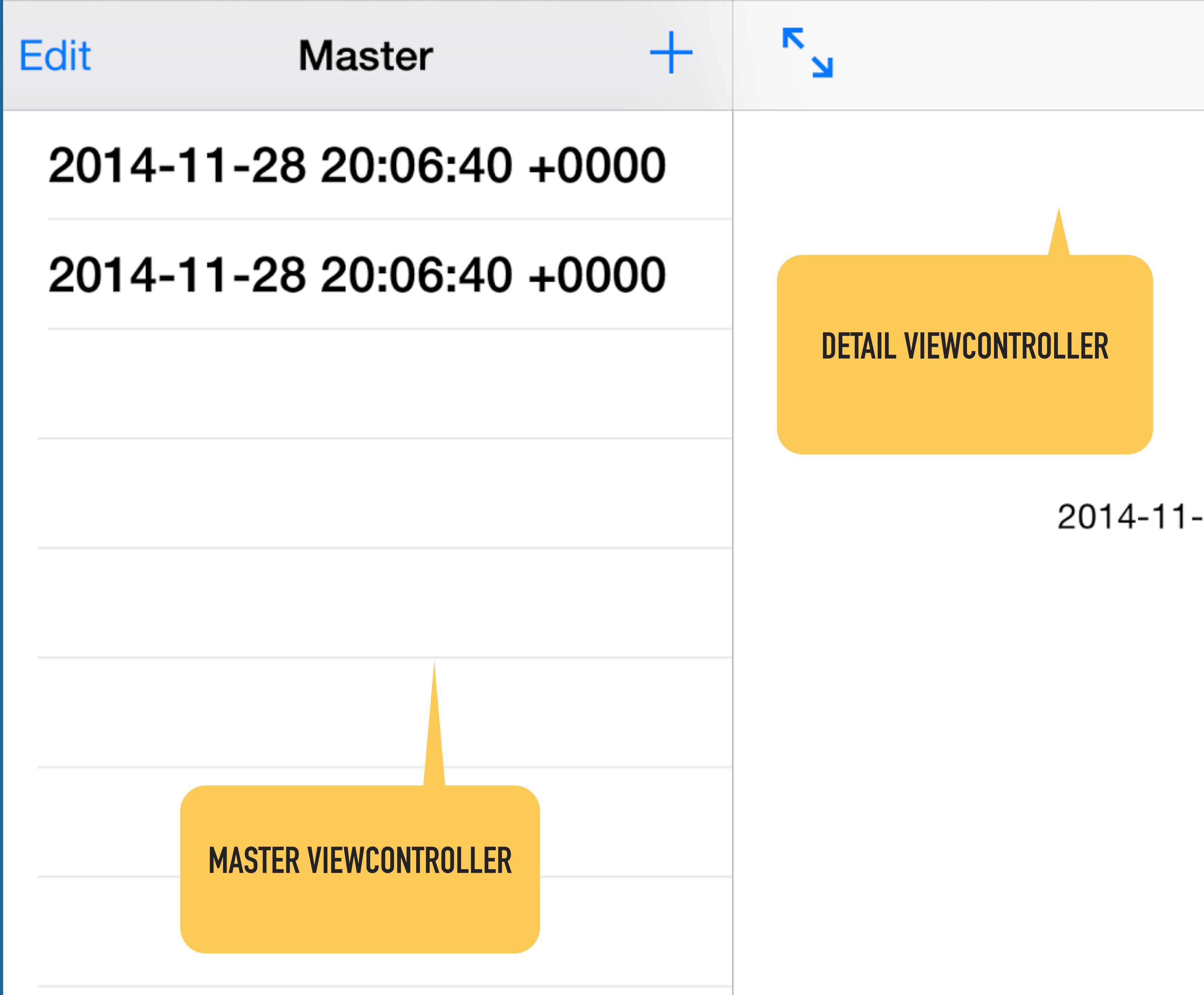
The image shows a Twitter feed with three visible tweets:

- Wall Street Journal @WSJ** 44s ago
How one of the most controversial doctors in history steered U.S. toward lobotomizing thousands of veterans: on.wsj.com/1h3iR4E
Retweet Retweet Star
- Atlassian HipChat @Hip...** 12/6/13
Check out what's new in HipChat -- New Mentions, Commands & More! ow.ly/rwE7d
Promoted by Atlassian HipChat Retweet Retweet Star
- Johnnie Manzari @johnnie** 1m ago
The best thing about Silicon Valley is its brilliant unpredictability; e.g. one of the leading car makers is now headquartered in Palo Alto.
Retweet Retweet Star

At the bottom, it says "John Herrman retweeted". The navigation bar at the bottom includes "Timelines", "Notifications" (with 20+ notifications), "Messages" (with 20+ messages), and "Me".

AND NOW...

- The root view controller does manage a “screenful of content”
- `window.rootViewController = 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

Wall Street Journal @WSJ 44s
How one of the most controversial doctors in history steered U.S. toward lobotomizing thousands of veterans: on.wsj.com/1h3iR4E

Atlassian HipChat @Hip... 12/6/13
Check out what's new in HipChat -- New Mentions, Commands & More! ow.ly/rwE7d

Promoted by Atlassian HipChat

Johnnie Manzari @johnnie 1m
The best thing about Silicon Valley is its brilliant unpredictability; e.g. one of the leading car makers is now headquartered in Palo Alto.

John Herrman retweeted

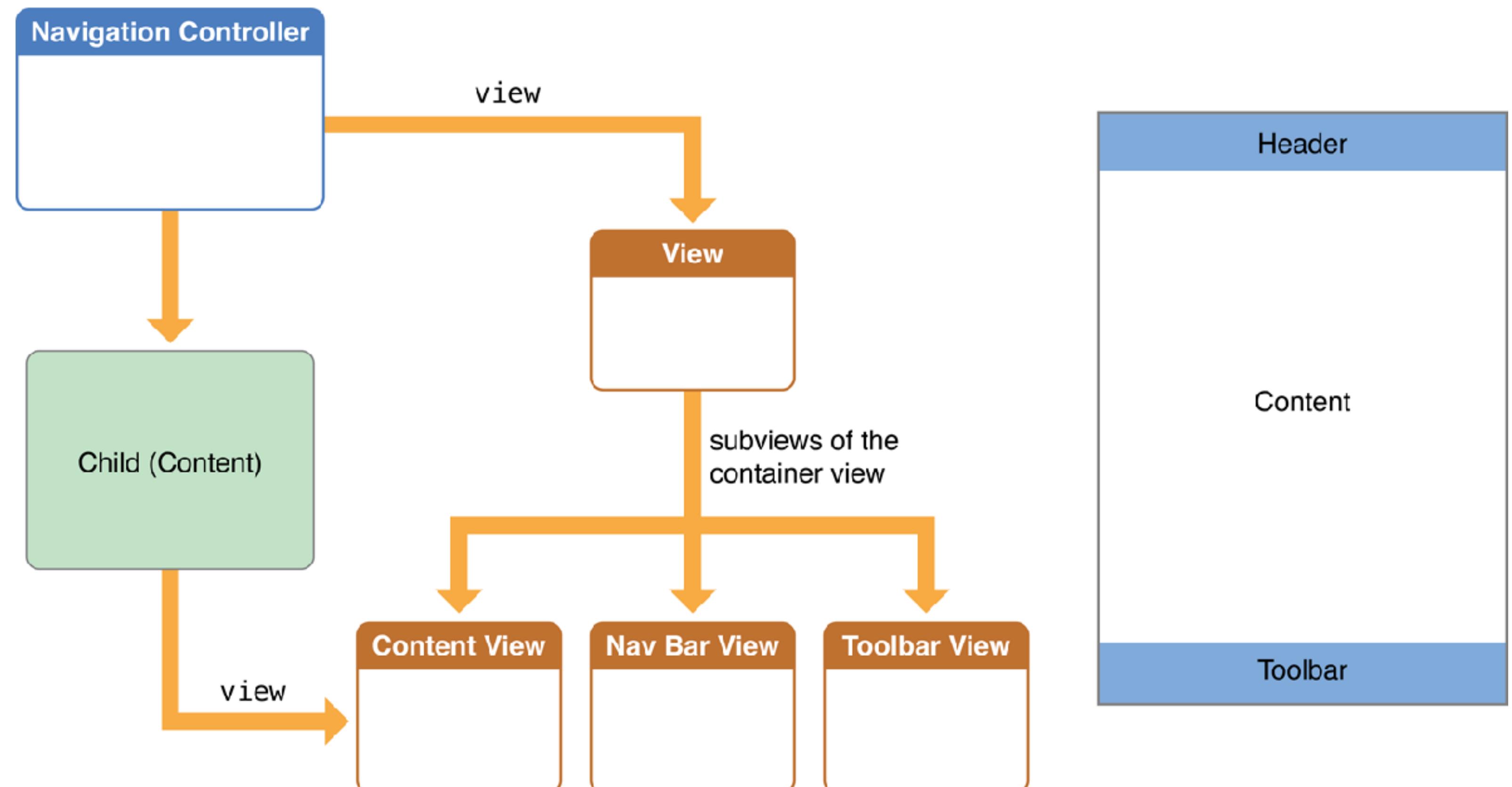
Timelines 20+ Notifications 20+ Messages Me

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)

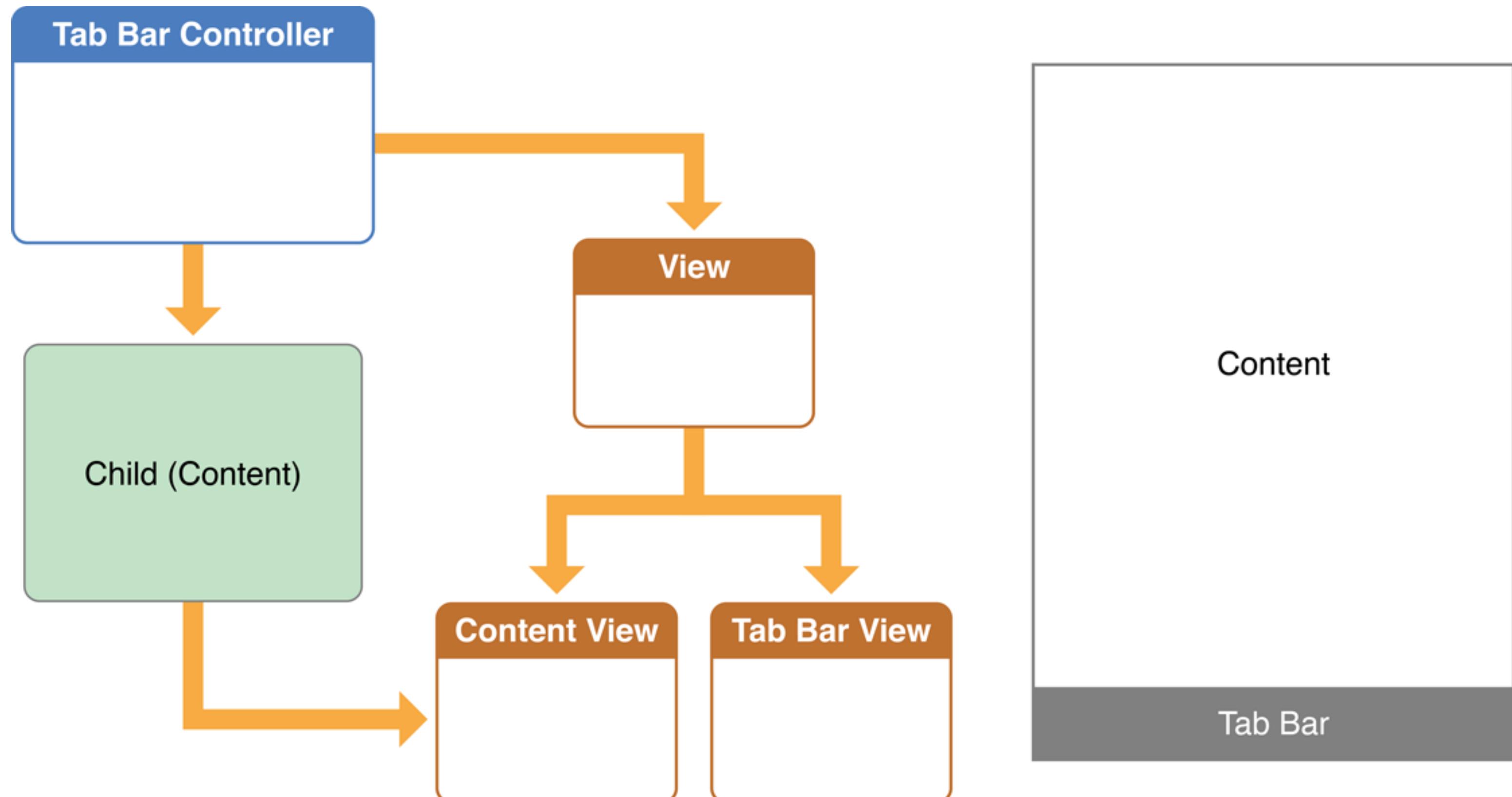
VIEW CONTROLLER CONTAINERS

- 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

lockergnome.net/questions//5299/apple-screwed-up

keithdsouza 8 mins
WTF Air India put Chennai passengers in Hyderabad plane, are you fucking kidding me :|

BlogsDNA 9 mins
Twitter for iPad is here with Gestures and New UI <http://bit.ly/bycqpu>

Techmeme
Skype Mobile for Android and Wifi (but still Verizon) (@philnickinson /... <http://j.mp/cuL4WM> <http://techme.me/A0F4>)

SEOMoz 11 mins
A few shots from the #mozinar (<http://bit.ly/ctkR0u>) /via @seomoz

A red arrow points from the "Techmeme" tweet to a red callout box containing the text "CUSTOM VIEW CONTROLLER".

Chris Pirillo
@ChrisPirillo
#756278

Unfollow

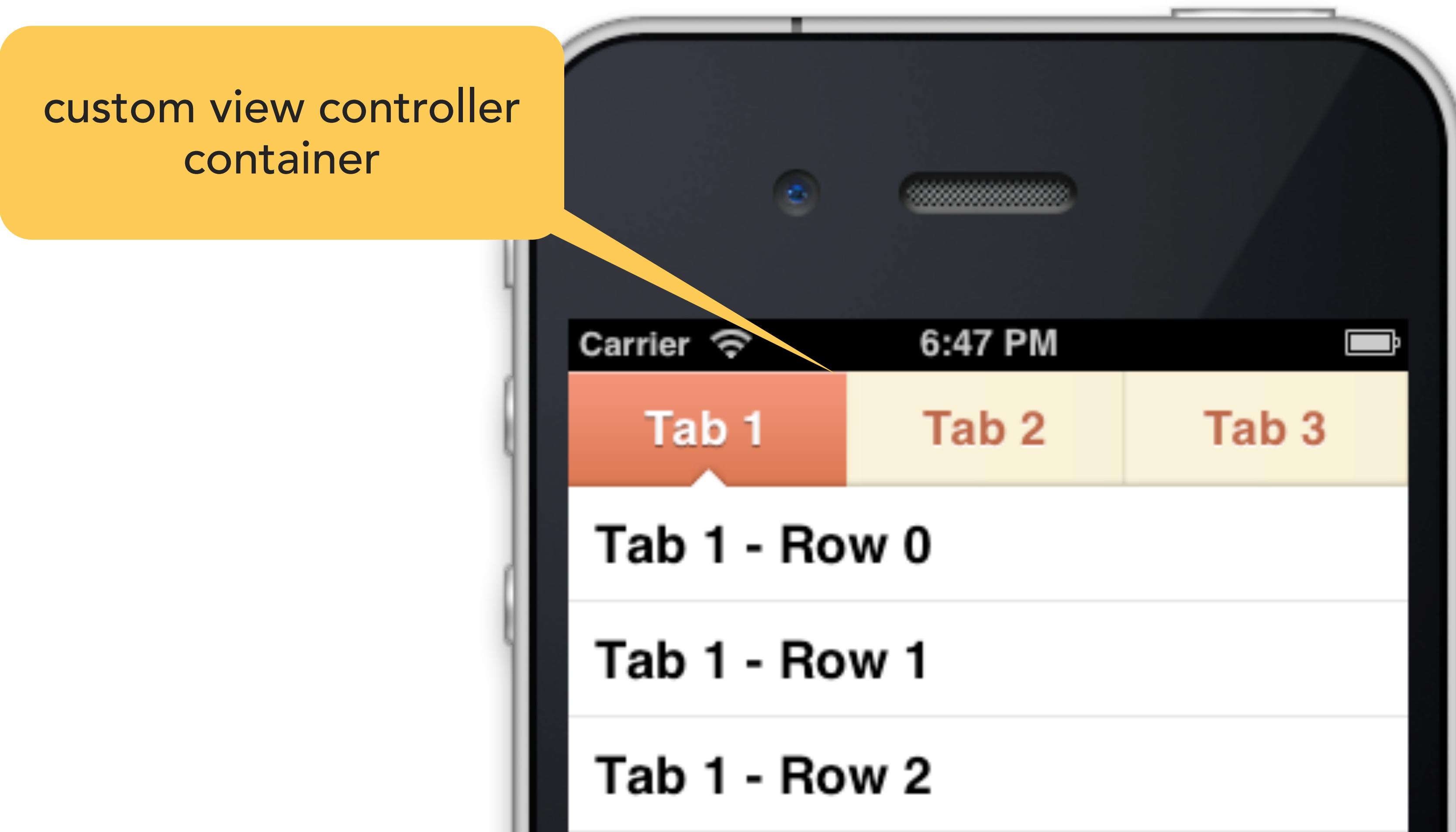
bio I've been making things happen online since 1992 - a media-friendly geek who produces content and catalyzes communities.

location Seattle, WA

web chris.pirillo.com/

Tweets **@Mentions** **Favorites**

VIEW CONTROLLER CONTAINERS



<https://github.com/hollance/MHTabBarController>

VIEW CONTROLLER CONTAINERS

The image shows a Twitter application interface. On the left is a dark grey sidebar with white text and icons for navigation: @ Mentions, Messages, Favorites, Search, Profile, Lists, Retweets, Mute Filters, and Settings. On the right is the main content area, which appears to be a custom container view or a UISplitViewController setup. It displays a timeline of tweets. The first tweet is from David Kaneda (@flyosity) with the text "Ordered it yesterday :)". The second tweet is from Dane Baker (@flyosity) with the text "Totally already preord". The third tweet is from Mike Rundle with the text "Beautiful new iPad stand uses". At the bottom of the main screen are five buttons: Reply, Retweet, Favorite (which is highlighted with a yellow star), Actions, and Details.

UISplitViewController or
custom container view?

VIEW CONTROLLER CONTAINMENT

VIEW CONTROLLER CONTAINERS

Blue
View color



Blue Arrow
Subview relationship



Gold
View controller color

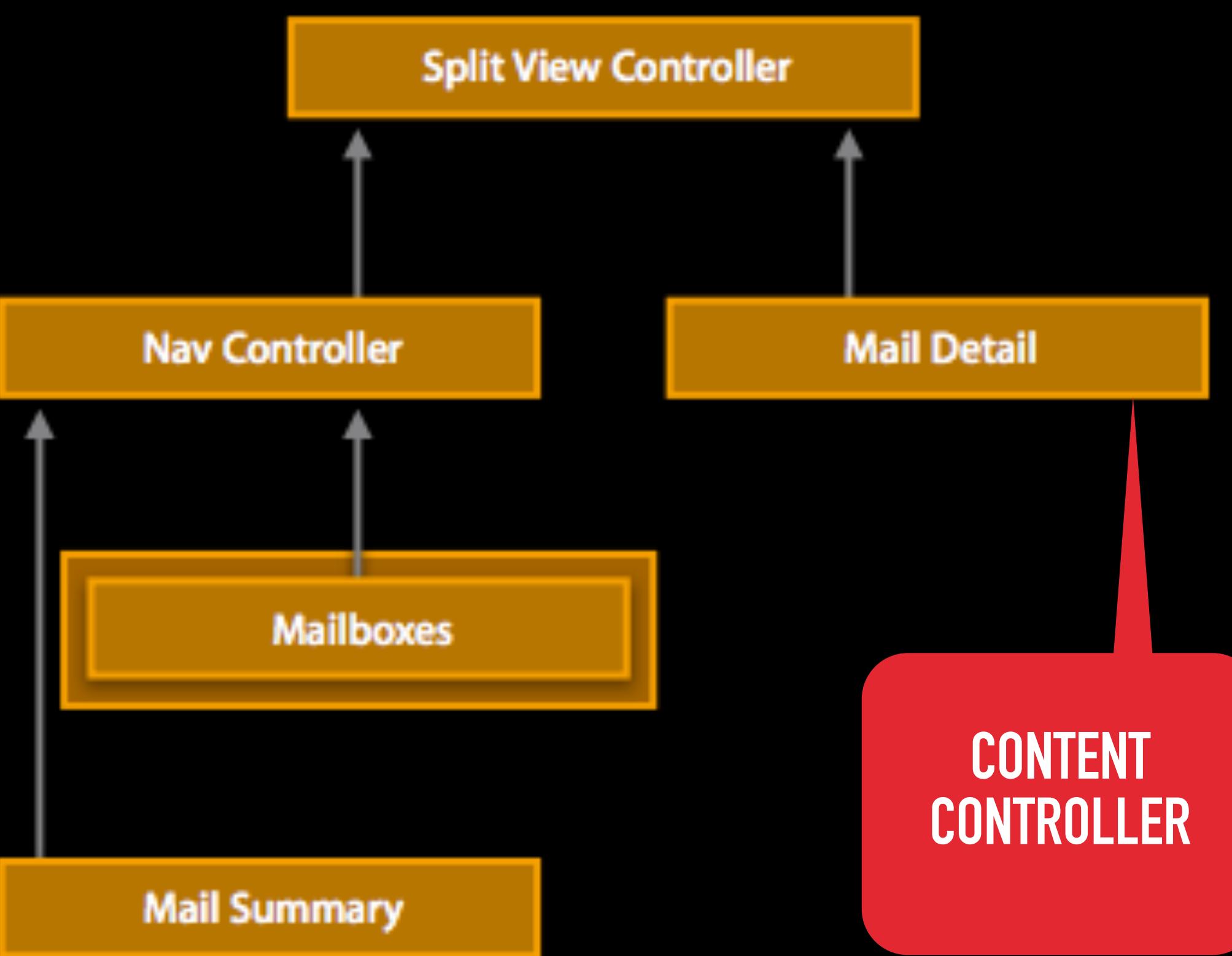
Split View Controller

Gray Arrow
Parent view controller
relationship



VIEW CONTROLLER CONTAINERS

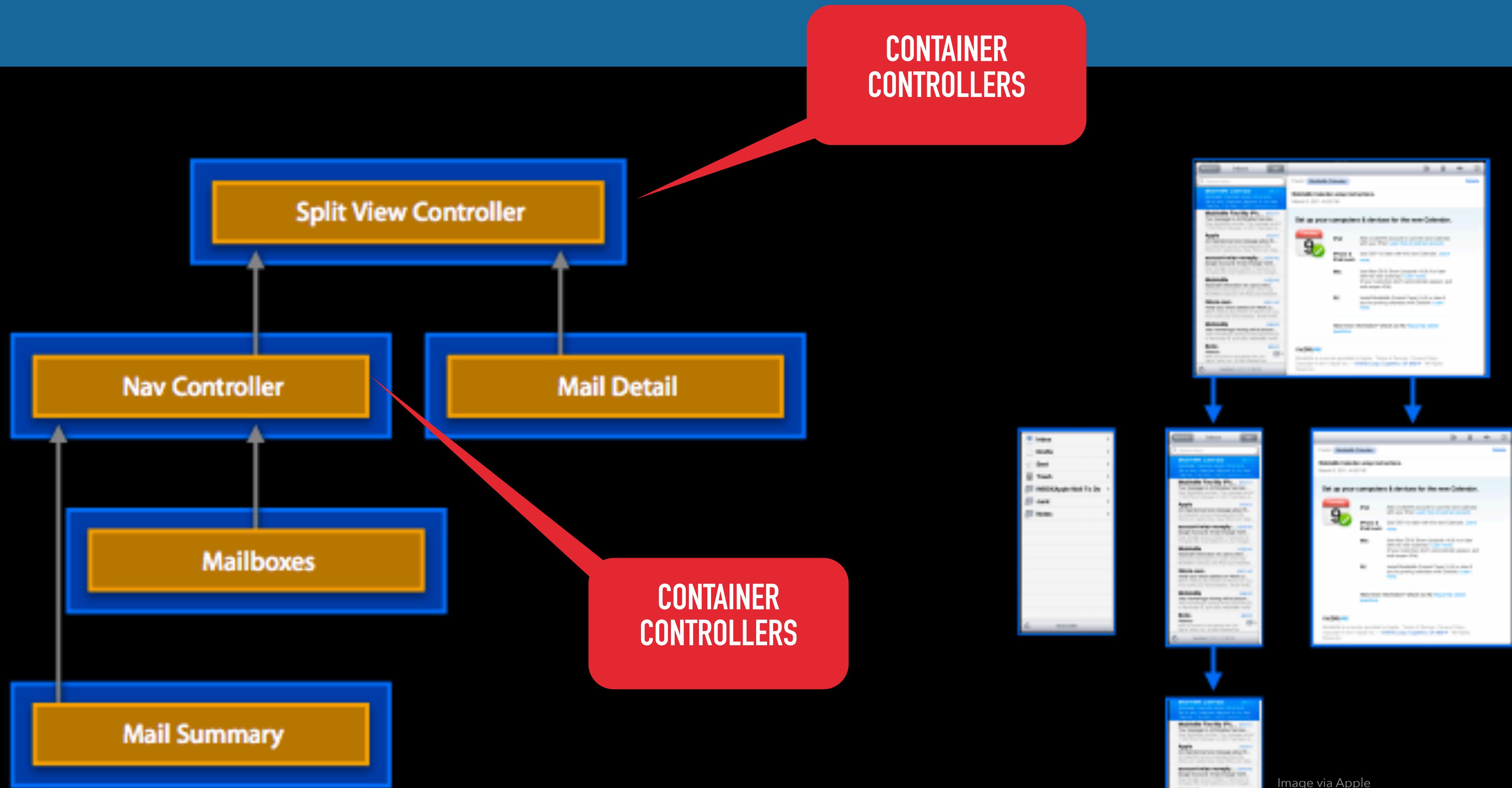
CONTENT CONTROLLER



CONTENT
CONTROLLER

Image via Apple

VIEW CONTROLLER CONTAINERS



VIEW CONTROLLER CONTAINERS



VIEW HIERARCHIES

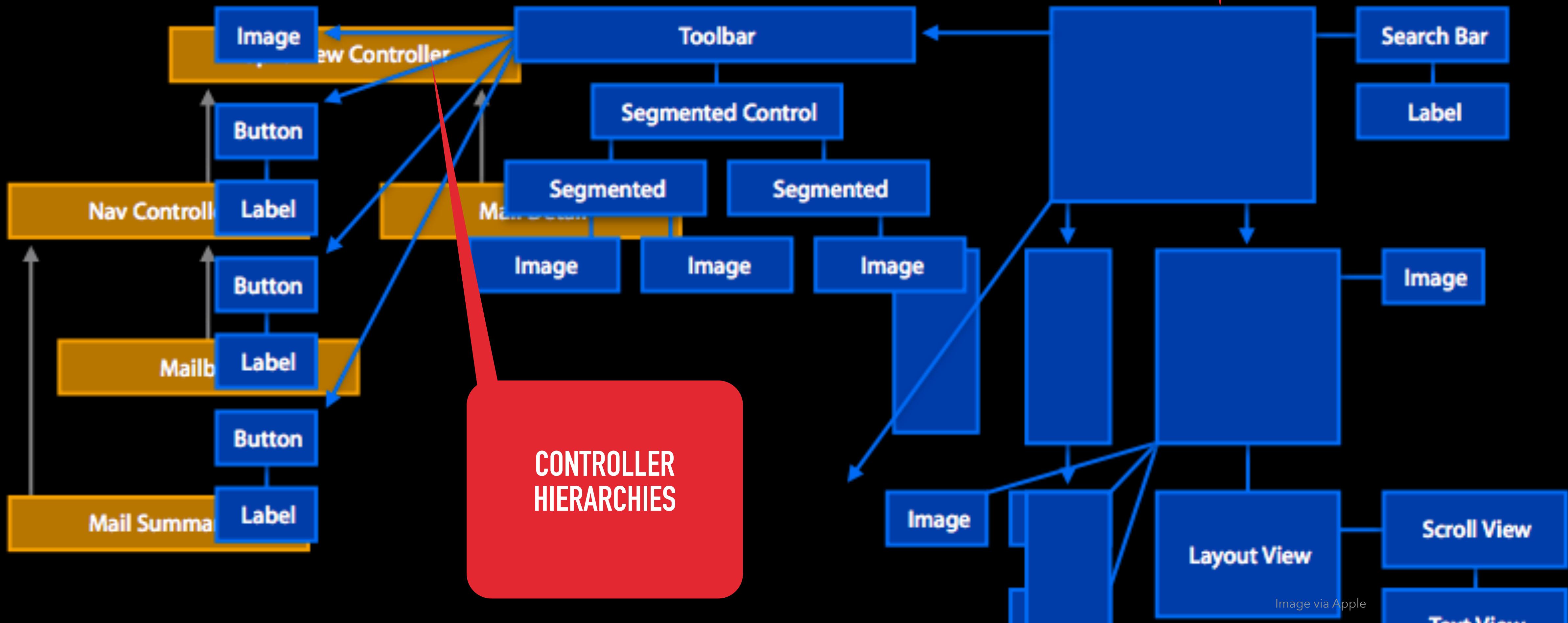
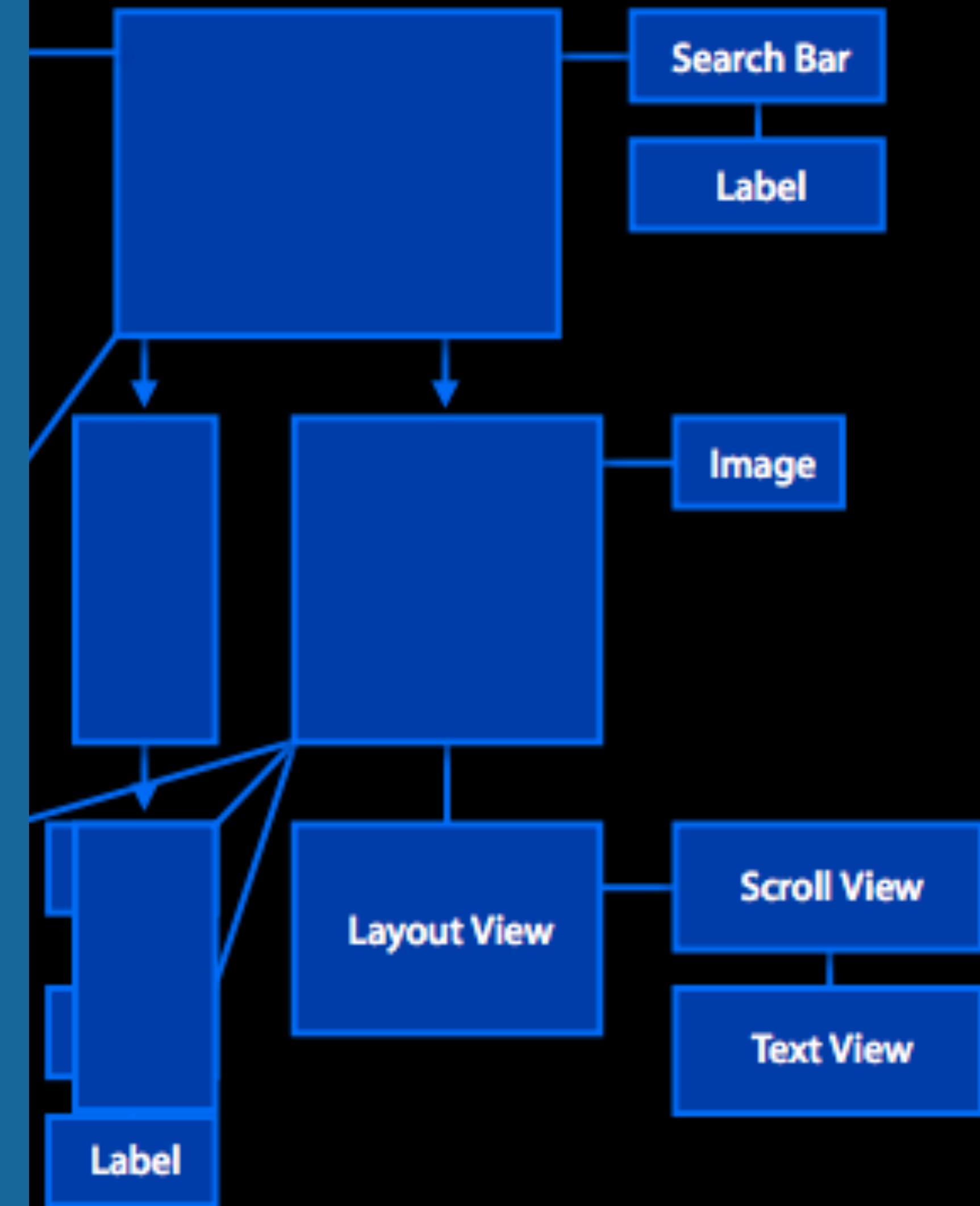


Image via Apple

Text View

VIEW CONTROLLER CONTAINERS

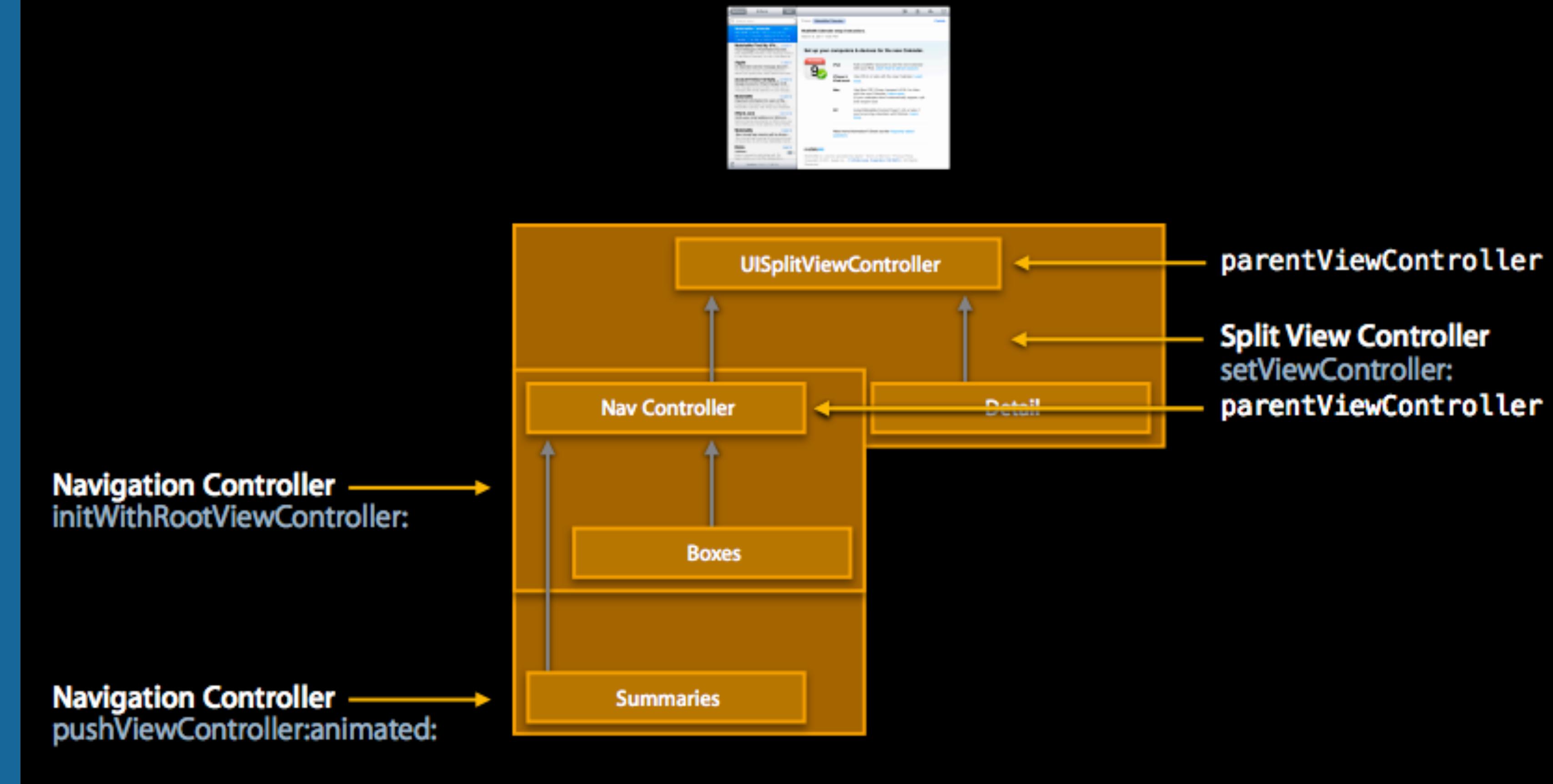
- Container views allow a view to have child views which are in turn managed by their own view controllers



VIEW CONTROLLER CONTAINERS

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

API and the controller hierarchy



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)
```

Show Green

Add Container



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()
```

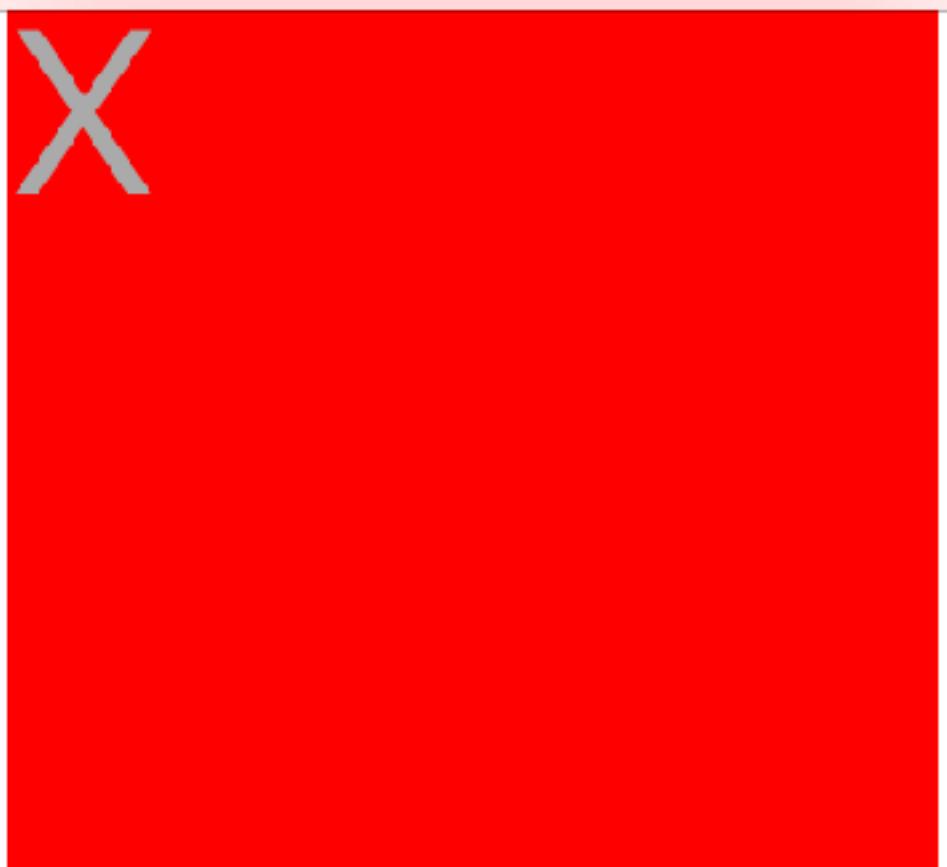
CONNECTING VIEW CONTROLLERS

SUBTITLE

- But do they know about each other?
 - Not if you're practicing good MVC
- Possible ways to pass messages
 - NSNotifications
 - KVO
 - Delegation

Show Green

Add Container



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: - Lite
//
```

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
    //

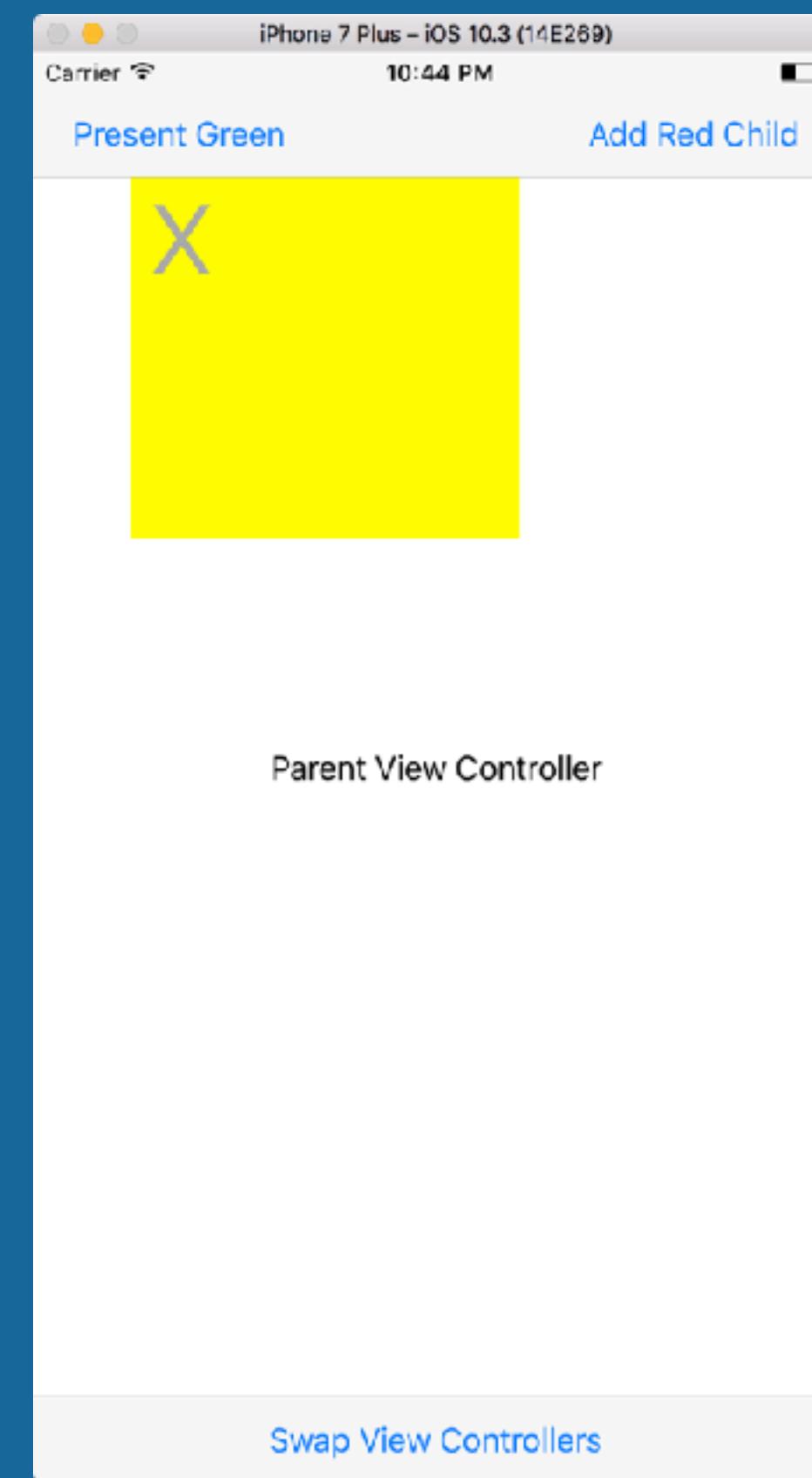
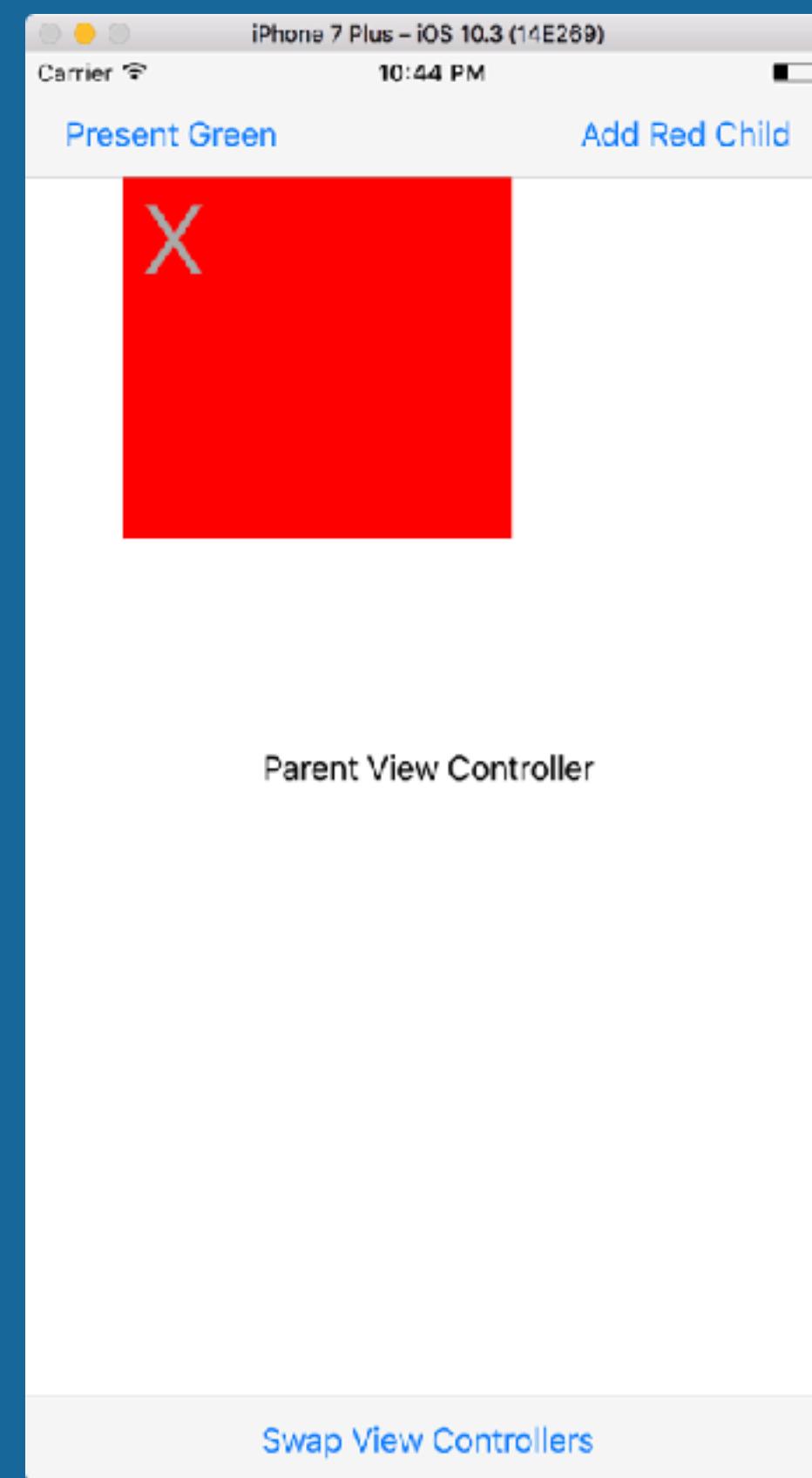
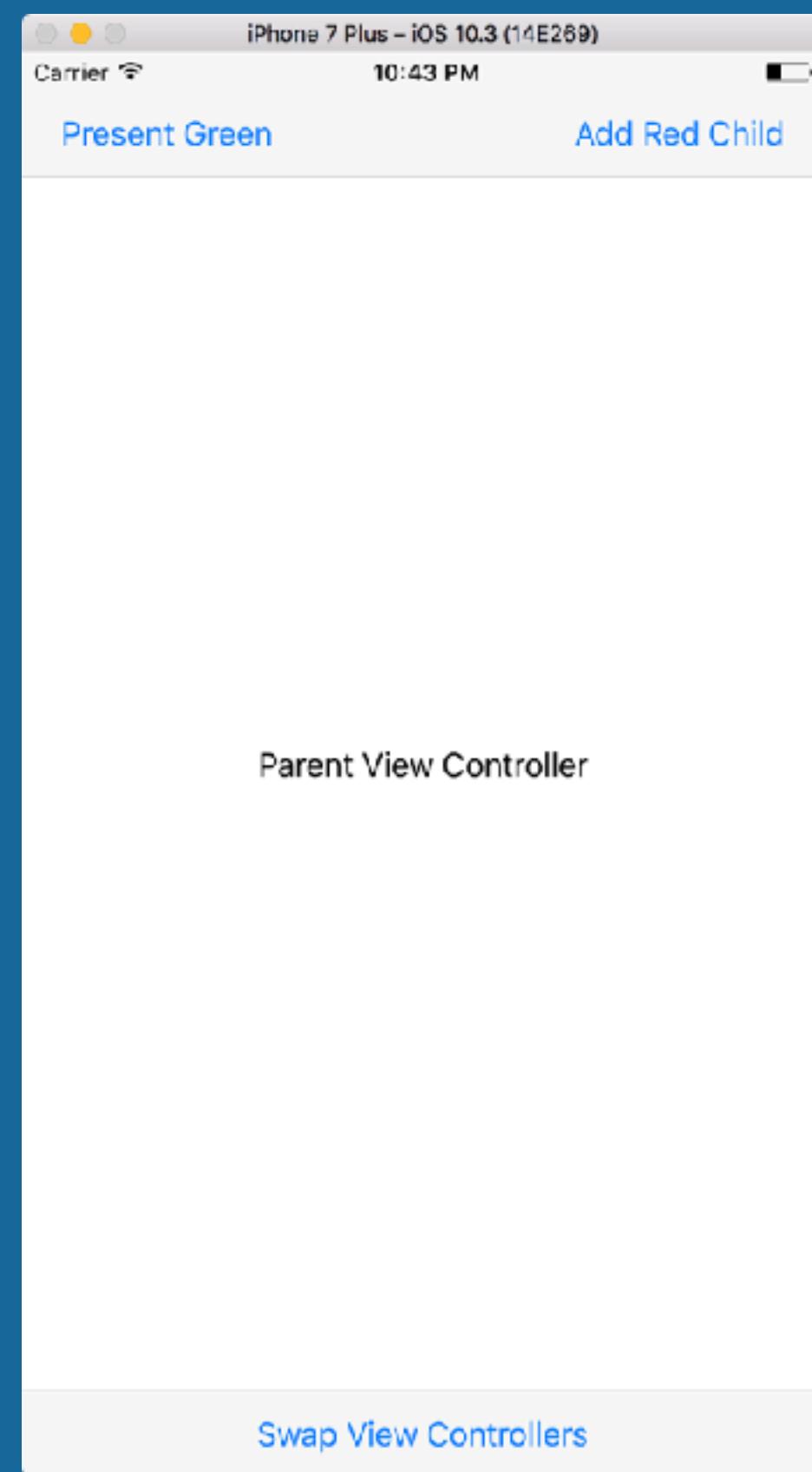
    override func viewDidLoad() { ... }

}
```

CHILD VIEW
CONTROLLER VIEW
TRANSITIONS

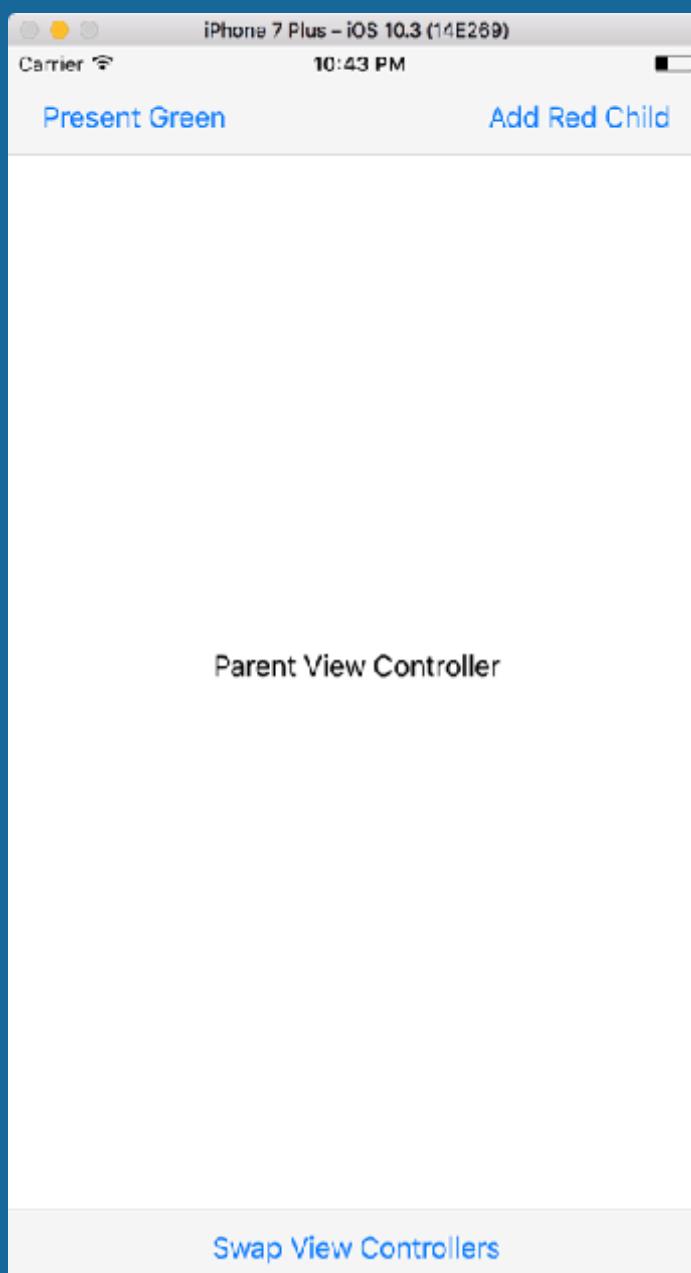
VIEW CONTROLLER CONTAINERS

- Transitioning between children view controllers



VIEW CONTROLLER CONTAINERS

- Transitioning between children view controllers

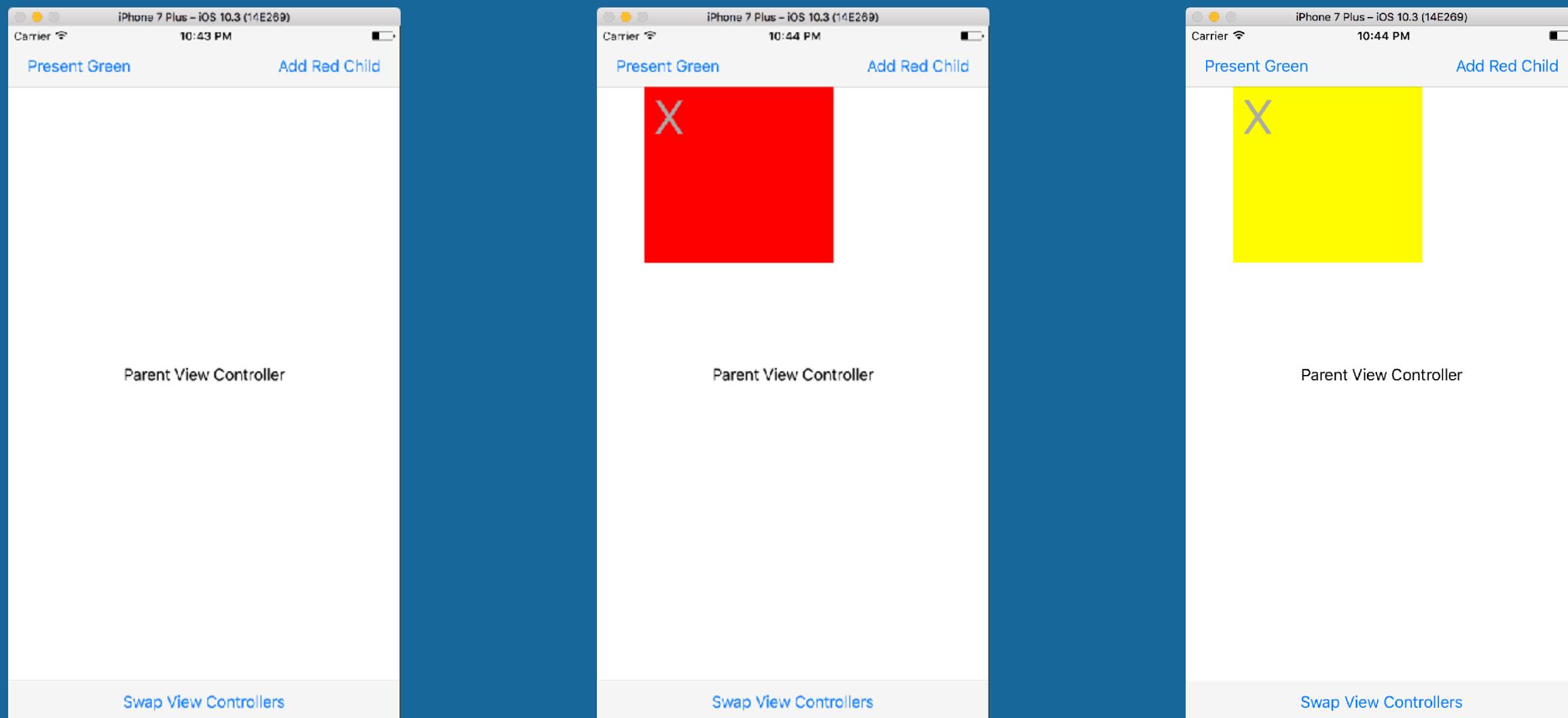


```
@IBAction func tapSwapViewControllers(_ sender: UIBarButtonItem) {  
    // Create a new view controller (yellow color)  
    let gvc = ColorViewController(nibName: "GreenViewController", bundle: nil)  
    gvc.delegate = self  
    gvc.view.backgroundColor = UIColor.yellow  
    gvc.view.frame = 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 your 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

The screenshot shows the Interface Builder library with several items listed:

- Tab Bar Item** - Represents an item on a UITabBar object.
- Search Bar** - Displays an editable search bar, containing the search icon, that sends an action message ...
- Search Bar and Search Display Controller** - Displays an editable search bar connected to a search dis...
- Fixed Space Bar Button Item** - Represents a fixed space item on a UIToolbar object.
- Flexible Space Bar Button Item** - Represents a flexible space item on a UIToolbar object.
- View** - Represents a rectangular region in which it draws and receives events.
- Container View** - Defines a region of a view controller that can include a child view controller.

At the bottom of the library, there are several small icons: a magnifying glass, a document, a square, a triangle, a circle, and a rectangle. To the right of these icons is a "Filter" button with a magnifying glass icon, and at the bottom right corner is the number "86".

VIEW CONTROLLER CONTAINMENT USING INTERFACE BUILDER

The screenshot shows the Xcode interface with the Project Navigator on the left and the Interface Builder editor on the right.

Project Navigator:

- ViewControllerIB (selected)
- ViewControllerIB
- AppDelegate.swift
- ViewController.swift
- Main.storyboard (selected)
- GreenViewController.swift
- Assets.xcassets
- LaunchScreen.storyboard
- Info.plist
- Products

Interface Builder (Main.storyboard):

The storyboard contains two view controllers:

- View Controller Scene:** Contains a **View Controller** object. This view controller has a **Container View** (represented by a blue rounded rectangle) and some **Constraints**. It also includes standard iOS objects like **First Responder**, **Exit**, and **Storyboard Entry Point**.
- Green View Controller...**: Contains a **Green View Controller** object. This view controller has a **View** object with a label **I'm my own vi...** and some **Constraints**. It also includes **First Responder** and **Exit**.

Preview Area:

The preview area shows two views side-by-side. The left view is labeled "View Controller" and contains a blue **UIView**. The right view is labeled "Green View Controller..." and contains the text "I'm my own view controller!".

A segue connection is visible between the two views, indicating the containment relationship where the green view controller is embedded within the main view controller's container view.

VIEW CONTROLLER CONTAINMENT USING INTERFACE BUILDER

The screenshot illustrates the setup and execution of View Controller Containment in Xcode.

Left Side (Interface Builder):

- Project Navigator:** Shows files like ViewController.swift, Main.storyboard, GreenViewController.swift, Assets.xcassets, LaunchScreen.storyboard, and Info.plist.
- File Inspector (M):** Shows the file is selected.
- Attributes Inspector (A):** Shows the file is selected.
- Document Outline:** Shows the structure:
 - Main.storyboard
 - Bottom Layout G...
 - View
 - Container View
 - Constraints
 - First Responder
 - Exit
 - Storyboard Entry Poi...
 - Embed segue to "Vie..."
 - Green View Controller...
 - Green View Controller
 - View
 - I'm my own vi...
 - Constraints
 - First Responder
 - Exit

Middle Section (Interface Builder Preview):

A diagram shows a main view controller's view (blue) containing a container view. A segue connects the container view to a secondary view controller's view (white), which displays the text "I'm my own view controller!".

Right Side (Simulator Preview):

The iPhone 7 Plus simulator window shows the final result. The main screen is green, and the secondary view controller's view displays the text "I'm my own view controller!".

**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 in 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



ADVANCED iOS APPLICATION DEVELOPMENT

MPCS 51032 • SPRING 2020 • SESSION 1A