iOS Dev Accelerator
Week1 Day3

- AutoLayout
- SizeClasses
- Network Controller
- UINavigationController

Homework Review

Autolayout

- A constraint-based layout system for making user interfaces.
- AutoLayout needs to know 2 things about every view in your interface:
  1. How big the view is going to be
  2. Where the object is going to be located
- You accomplish this using constraints.

Constraints

- Constraints are the fundamental building block of autolayout.
- Constraints contain rules for the layout of your interface's elements.
- You could give a 50 point height constraint to an imageView, which constrains that view to always have a 50 point height. Or you give it a constraint to always be 20 points from the bottom of its superview.
- Constraints work together, but sometimes they conflict with other constraints.
- At runtime Autolayout considers all constraints, and then calculates the positions and sizes that bests satisfies all the constraints.

Attributes

- When you attach constraints to a view, you attach them using attributes.
- The attributes are: **left/leading, right/trailing, top, bottom, width, height, centerX, centerY.**
- **The attribute tells the constraint which side, center, or height/width to attach to.**
- So if you attach a constraint of 50 points from a button's left attribute to its container's left attribute, thats saying "I want this button to be 50 points over from its super view's left side"

Constraints + Attributes = Math time

- “You can think of a constraint as a mathematical representation of a human expressible statement”
- So if you say “the left edge should be 20 points from the left edge of its containing view”
- This translates to button.left = container.left x 1.0 + 20
- which is in the form of y = mx+b
- first attribute = second attribute * multiplier + constant
- In the case of an absolute value, like pinning height, width, centerX, or centerY, the second attribute is nil.
- You can change the constants in code as an easy way to programmatically adjust your interface.

Storyboard and Autolayout

- Storyboard makes setting up autolayout pretty intuitive and painless, and even though you can setup autolayout completely in code, Apple strongly recommends doing it in storyboard.
- Xcode will let you build your app even if you have constraints that are conflicting and incorrect, but Apple says you should never ship an app like that.
- When you drag a object onto your interface, it starts out with no constraints.

Intrinsic Content Size

- Intrinsic content size is the minimum size a view needs to display its content.
- Its available for certain UIView subclasses:
  - UIButton & UILabel: these views are as large as they need to display their full text
  - UIImageView: image views have a size big enough to display their entire image. This can change with its content mode.
- You will know a view has an intrinsic content size if autolayout doesn't require its size to be described with constraints.

Demo

Size Classes

Size Classes

- “Size classes are traits assigned to a user interface element, like a screen or a view”
- There are only two types of size classes, Regular and Compact.
- Size classes, together with displayScale and userInterfaceIdiom (iPhone or iPad) make up a trait collection.
- Everything on screen has a trait collection, including the screen itself, and view controllers as well.
- The storyboard uses a view controller’s trait collection to figure out which layout should be currently displayed to the user.

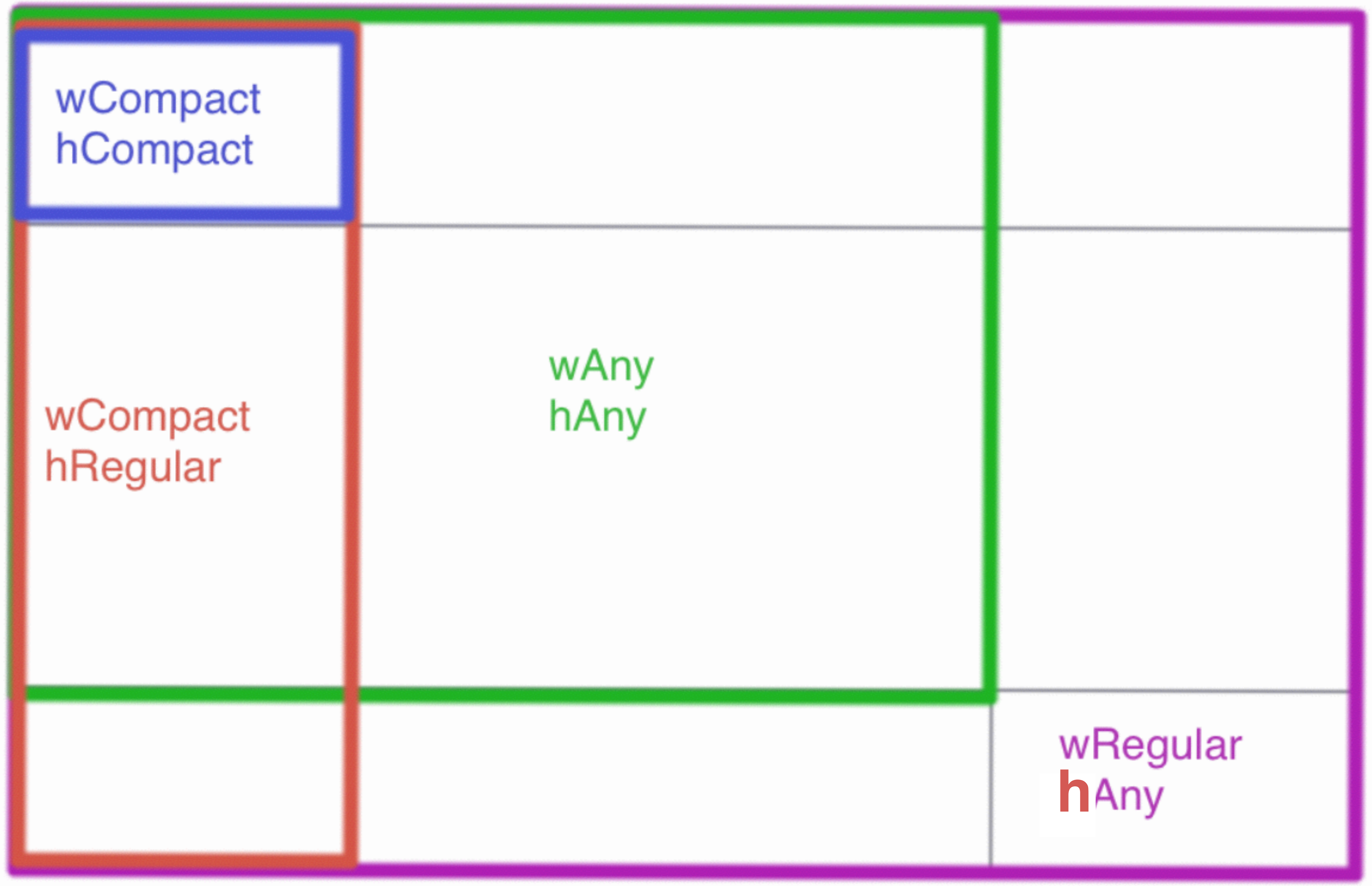
Size Classes and Storyboard

- Size classes allow you to have different constraints and layouts for each configuration on the storyboard.
- By default, every size class configuration will pull from the base configuration, which is wAny hAny.
- If you change your storyboard’s configuration, certain changes you make will only apply when your app is running in that specific size class.

Size Classes

- Specifically, there are 4 things you can change in each configuration on your storyboard:
    1. constraint constants.
    2. font and font sizes
    3. turning constraints off
    4. turning view on and off

Size Classes



- iPad Landscape and Portrait
- Base configuration
- iPhone Portrait
- iPhone Landscape

Demo

Network Controller

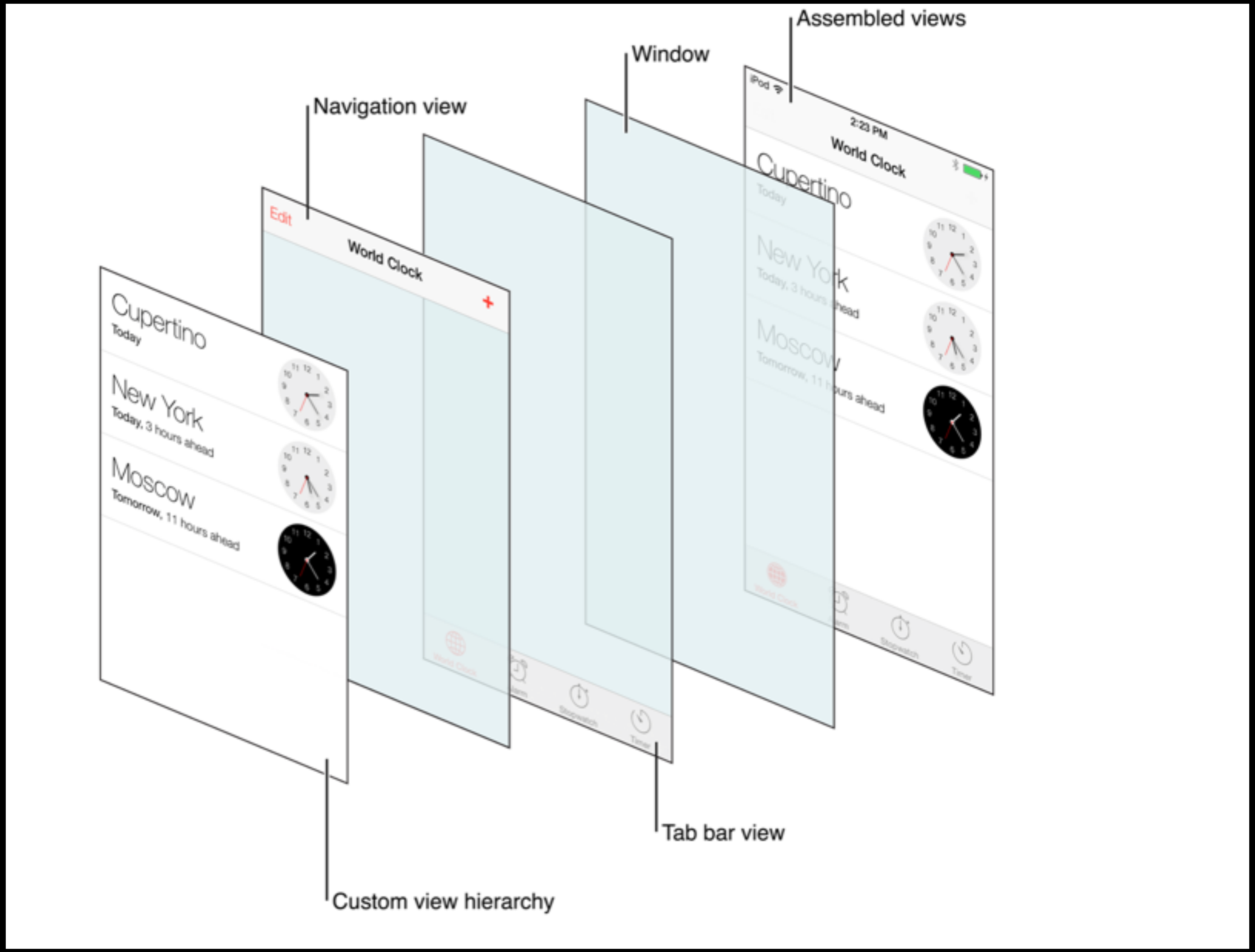
Network code

- Right now we have our network code inside of our view controllers.
- This works, but what if a bunch of different view controllers are making the same network calls? Thats not efficient.
- Anytime you have duplicate code anywhere, that is a ‘code smell’ or a hint that you need to do some refactoring.

Network code best practices

- There are two spots that experienced iOS developers like to put their network code:
  - In the models classes themselves.
  - In a consolidated network class.
- I greatly prefer the consolidated network class, because it makes our model classes a lot less complicated, and I like only having to go to one place for network code issues/debugging.
- Theres a few name patterns for a class like this:
  - <Name of the web api>Service (ex: TwitterService)
  - <Name of the web api>API (ex: TwitterAPI)
  - NetworkController
  - NetworkManager

Demo



Navigation Controllers

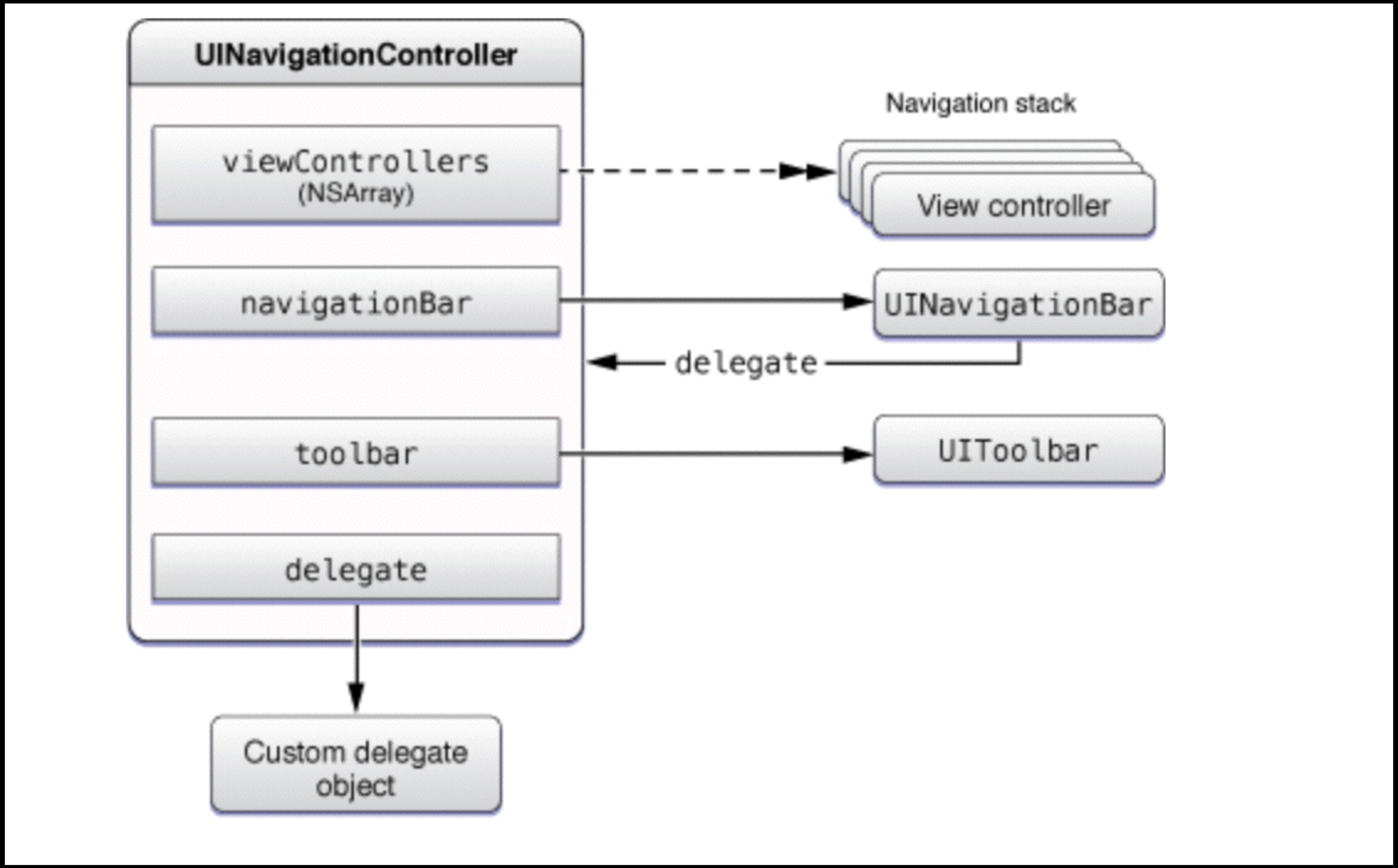
2 types of View Controllers

- Conceptually, there are two flavors of view controllers:
  - Content View Controllers: Present your app's content. Used to populate views with data from the model and respond to user actions.
  - Container view controllers: Used to manage content view controllers.
- **Container view controllers are the parents, and content view controllers are the children.**

Navigation Controller

- A navigation controller is an example of a container view controller
- “A navigation controller manages a stack of view controllers to provide a drill down interface for hierarchal content”

Navigation Controller Anatomy



Creating a Navigation Controller

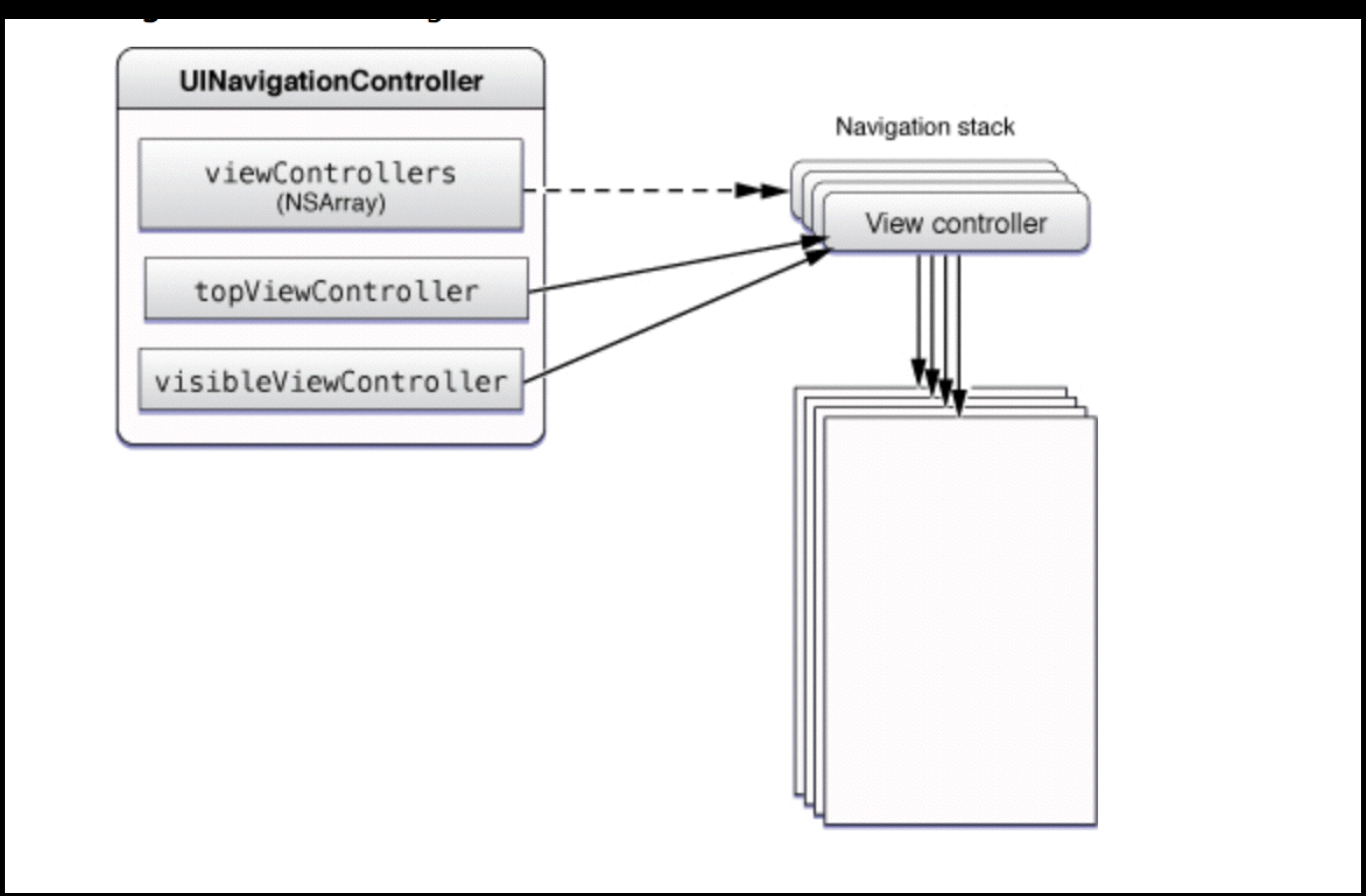
- 2 simple ways to get a navigation controller into your app:
    - Instantiate it in code. UINavigationController has 2 inits, one that takes in a view controller as its root view controller, and another that takes custom navigation bar and tool bar subclasses.
    - Embed it via storyboard.

Demo

Pushing and Popping

- A navigation controller uses a stack data structure to manage all of its children content view controllers.
- A stack is a pretty simple data structure. To add something to the stack, we push onto it. To take something off the stack, we pop.
- So to get a view controller on to the top of our navigation controller's stack, aka on screen, we can simply call pushViewController() on our navigation controller, and pass in the VC we want to push.
- And for taking a view controller off these stack, basically like pressing the back button, we call popViewController().
- There are also methods for popping to the root and popping to specific view controller in the stack.

Pushing and Popping



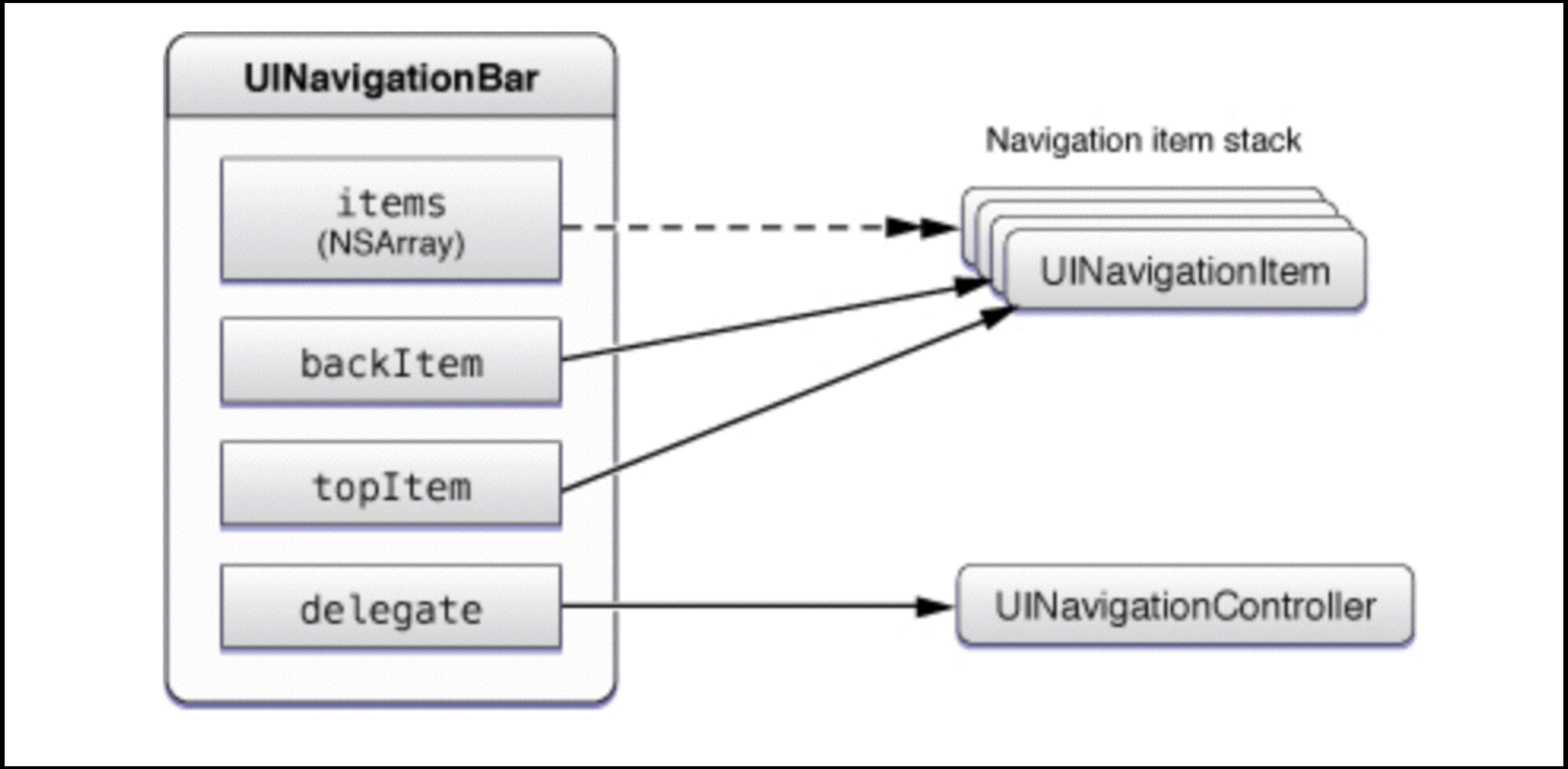
Demo

Navigation Bar

- The navigation bar of the navigation controller manages the controls of the navigation interface.
- The navigation controller takes most of the responsibility in creating and maintaining the navigation bar.
- You can also create UINavigationBar’s as standalone view’s and use them in your apps without using a navigation controller (rare)

Navigation Bar Anatomy

- A navigation bar has pretty similar setup as the navigation controller:



UINavigationItem

- UINavigationItem provides the content that the navigation bar displays. It is a wrapper object that manages the buttons and views to display in a navigation bar.
- The managing navigation controller uses the navigation items of the topmost two view controllers to populate the navigation bar with content.
- The navigation bar keeps a stack of all the items, in the exact same order as the navigation controller keeps track of its child content view controllers.
- Each View controller has a property that points to its corresponding navigation item
- The navigation bar has 3 positions for items: left, center, and right.

UINavigationItem positions

- Left: usually reserved for the back button, but you can replace it with whatever view you want by setting the navigation bar's leftBarButtonItem property.
- Center: Displays the title of the currently displayed view controller.
- Right: Empty by default, is typically used to place buttons that fire off actions.

Altering the Nav Bar

- The Navigation Controller owns the navigation bar and is very protective of it.
- You can't modify its bounds,frame, or alpha values directly.
- The properties you can modify are barStyle, translucent, and tintColor.
- To hide the navigation bar, call the method setNavigationBarHidden(animated:)

More hiding properties on the nav controller

- hideBarsOnTap
- hideBarsOnSwipe
- hidesBarsWhenVerticallyCompact
- hidesBarsWhenKeyboardAppears
- barHideOnTapGestureRecognizer
- barHideOnSwipeGestureRecognizer

Demo