

# CS 329E

# Elements of Mobile Computing

Fall 2017

University of Texas at Austin

Lecture 5

# Agenda

- Project
- Brief iOS UI layout history
- Auto Layout
- Constraints
- Storyboards
- Segues

Project

# Project

- Project teams must be defined by no later than Monday, 9/18.
- Teams of 3.
- Each team should send me an email stating they are a team - 1 email per team - with all members CC'd.
- The first deliverable is the Application Idea paper.

# Brief iOS UI Layout History

# Brief iOS UI Layout History

Some history about user interface layout:

- In the beginning, there was just *manually positioning* all the user interface elements - using frames.
  - Worked fine because there was only 1 iPhone size.
- Then came *Springs and Struts* - similar in concept to AutoLayout, but much less capable.
- iOS 6 introduced *Auto Layout*, which also introduced the concept of *constraints*.
  - We'll be focusing on auto layout and constraints because they are foundational to UI design all by themselves, and are used with Size Classes.
- iOS 8 introduced *Adaptive Layout* with *Size Classes*.
  - This is enabled by default when creating a project.
  - By default, the UI you design is used for all device sizes.
- Xcode 8/iOS 10 introduced *Trait Variations* to work with Size Classes.

# Auto Layout

# Auto Layout

What is Auto Layout?

- A mechanism that allows developers to create *adaptive* interfaces that respond appropriately to changes in screen size and device orientation (portrait/landscape).
- Which means that, in the end, you have a way to properly display a user interface on N different sized devices, with no additional code.

That said: Auto Layout *can* be very frustrating to work with until you get enough (painful) experience under your belt. But it is well worth it.



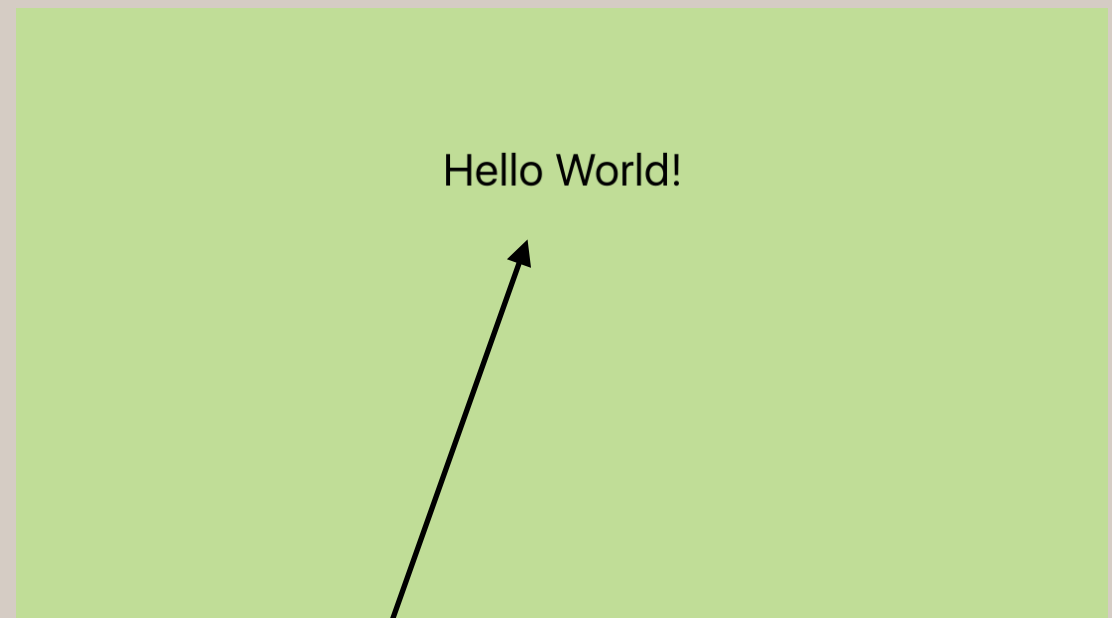
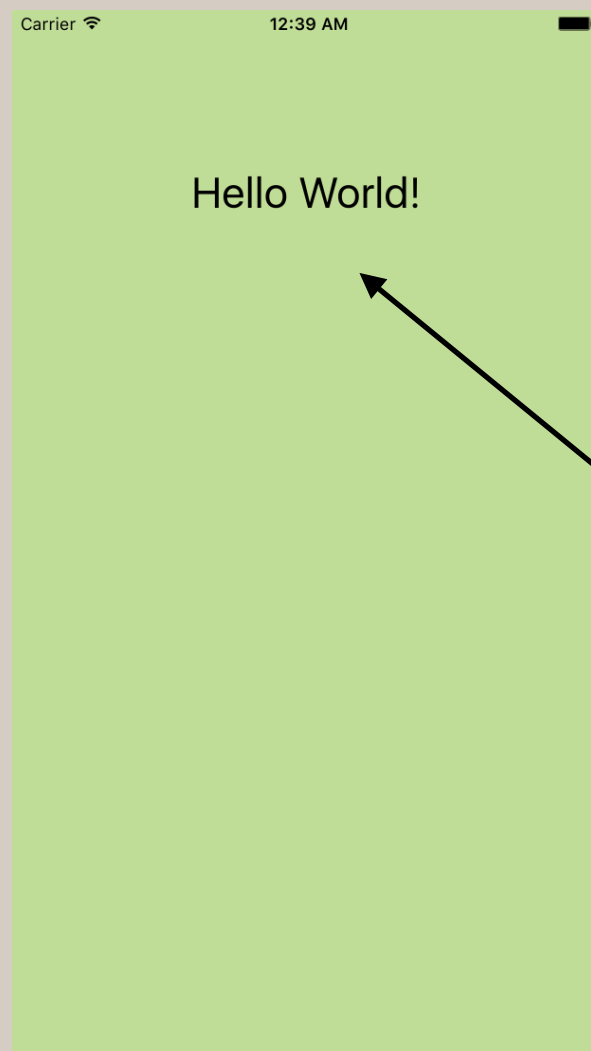
# Auto Layout

Who cares about *adaptive* interfaces? Everyone - should.

Screen orientations

Portrait

Landscape



Centered regardless of orientation

# Auto Layout

Auto Layout involves:

- Constraints
- Priorities
- Conflict resolution
- Internal and external changes
- Intrinsic content size

# Auto Layout

Auto Layout is all about *constraints*:

- Can be set in Interface Builder or programmatically.
- You can define constraints in IB using the *Add new constraints* and *Align* tools - found in the storyboard, bottom right.
- You can fix constraint issues with the *Resolve* tool.
- You can fix frames with the *Update frames* tool.



# Auto Layout

What are constraints?

- A constraint is a relationship between two user interface elements
- Example constraints:
  - A constraint that says there is a fixed distance between two user interface elements
  - A constraint that says a user interface element should be positioned in the horizontal middle

# Auto Layout

Constraint example 1:

Position the top of a label (label2) 20 points from the bottom of another label (label1).

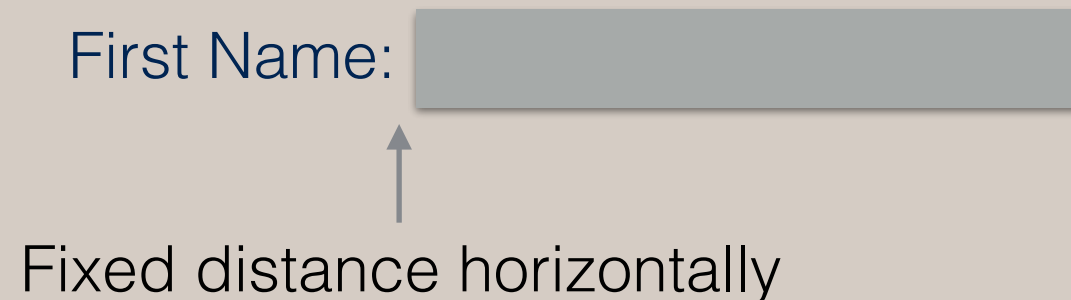


Constraint:  $\text{label-2 top} = \text{label-1 bottom} + 20 \text{ points}$

# Auto Layout

Constraint example 2:

Position the left of a text field a certain distance from the right of a label.



# Auto Layout

Some kinds of constraints you can define:

- Horizontal and/or vertical distance between elements, or edge of a containing view.
- Alignment of the leading, trailing, bottom and/or top edge of two elements.
- Horizontal and/or vertical centering of an element to its containing view.
- Specific width and/or height of an element.
- Content hugging priority.
  - To prevent an element from getting larger.
- Content compression resistance priority.
  - To prevent an element from getting smaller.

# Auto Layout

Constraint priority:

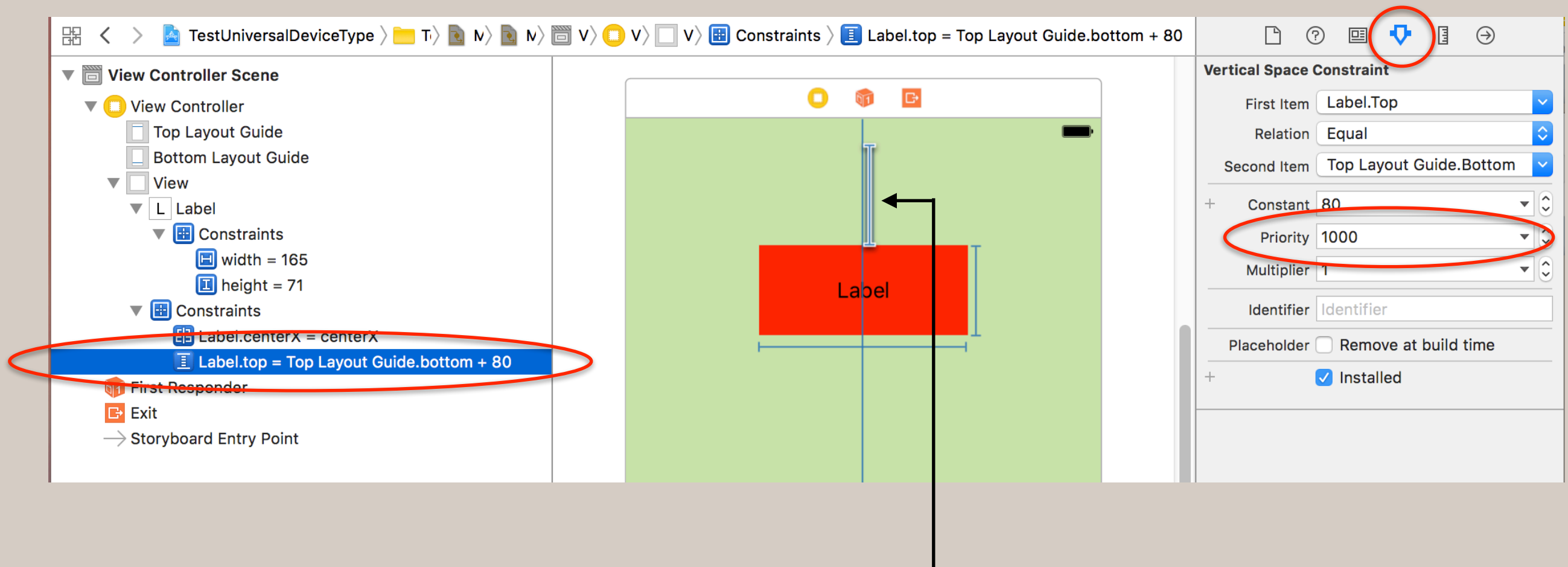
- Each constraint has a priority associated with it.
- Constraint conflict is resolved by the *auto layout engine* choosing the constraint with the higher priority.

That said: Most UIs don't mess much with constraint priorities.



# Auto Layout

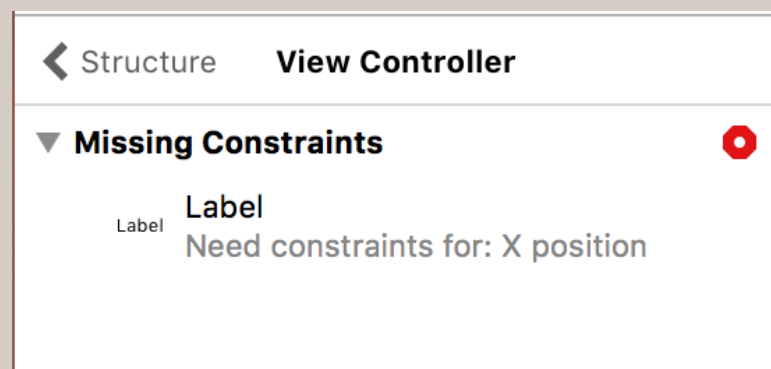
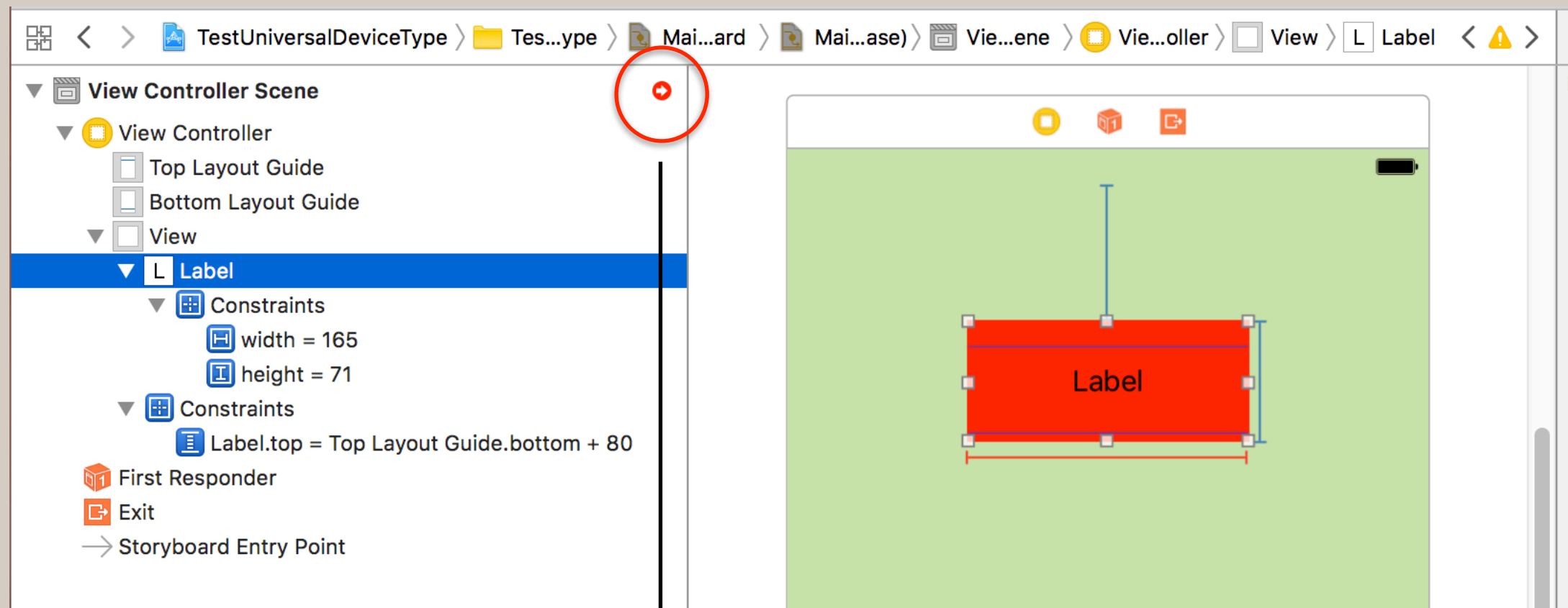
Constraint priority:



The selected constraint's line is thicker than the others.

# Auto Layout

Conflicting or missing constraint:



# Auto Layout

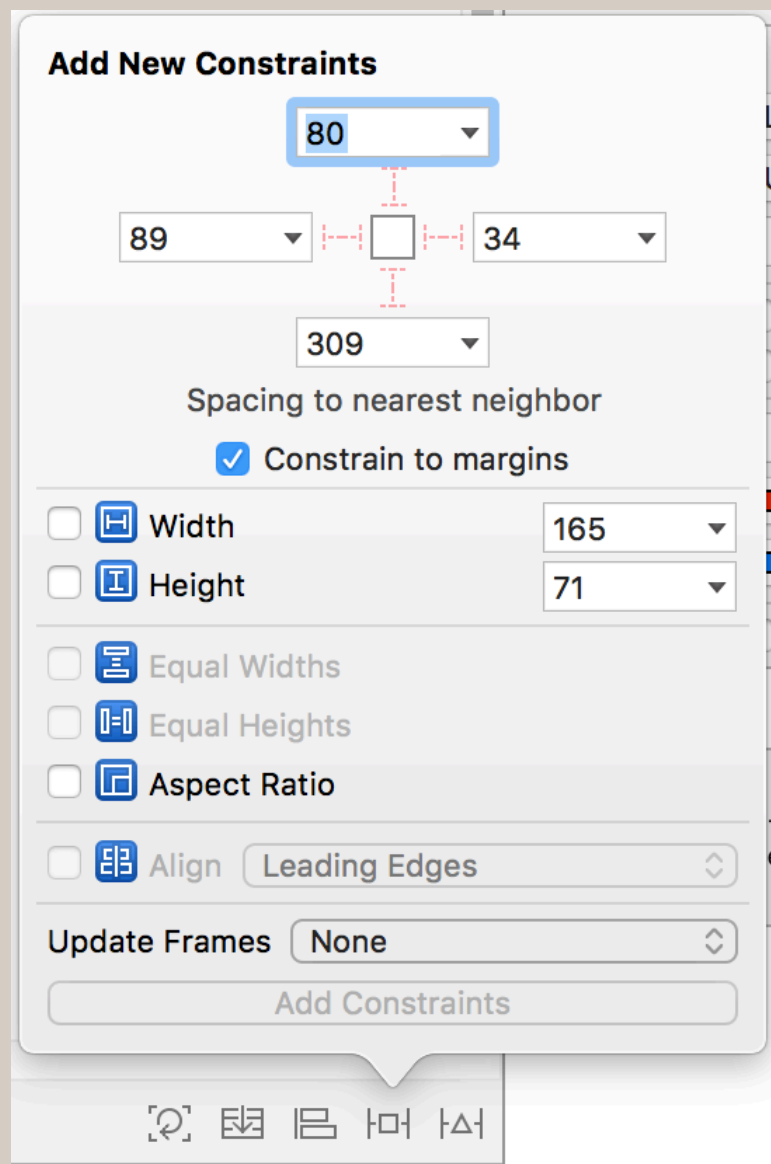
When to use the tools:

- Adding new constraints
  - Add new constraints
  - Align
- Clearing/updating constraints
  - Resolve auto layout issues
- Updating frames not in alignment with constraints
  - Update frames

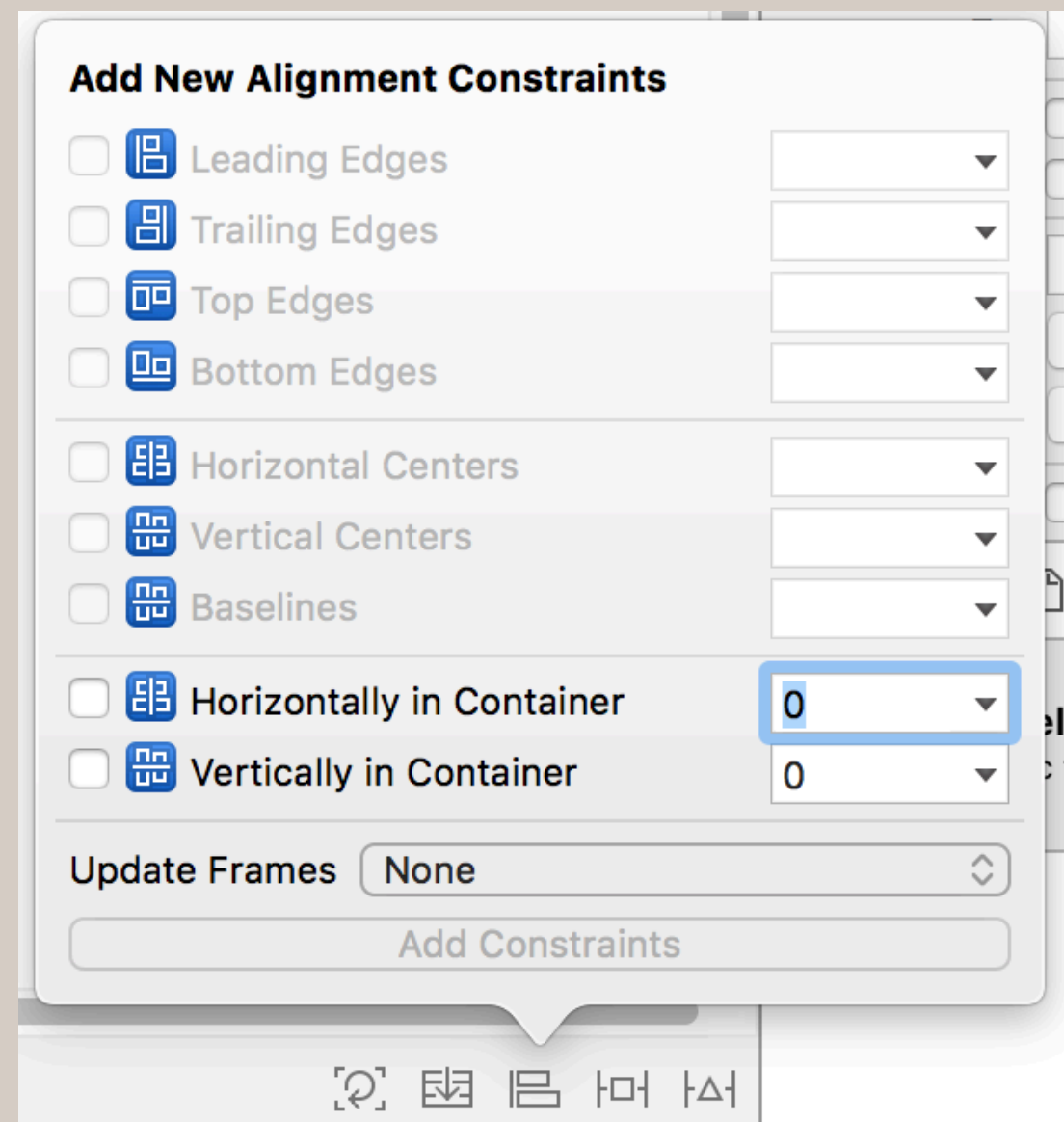


# Auto Layout

Constraints:



Add new constraints tool



Align tool

# Auto Layout

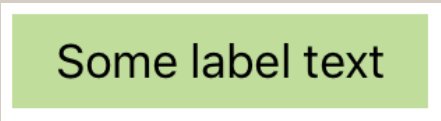
How does Auto Layout work?

- Auto Layout dynamically calculates the size and position of all the views in your view hierarchy, based on *constraints* placed on those views.
- This constraint-based approach to design allows you to build user interfaces that dynamically respond to both internal and external (device rotation) changes.

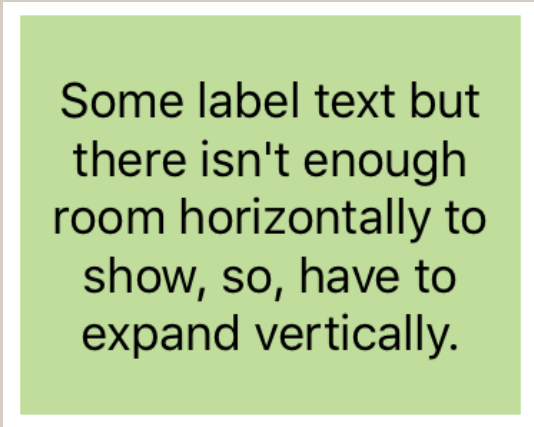
# Auto Layout

Internal changes:

- Internal changes occur when the size of the views or controls in your user interface change.
- A common source of internal change is the content displayed by the application changes, perhaps due to the length of some text in a label.



Some label text



Some label text but  
there isn't enough  
room horizontally to  
show, so, have to  
expand vertically.

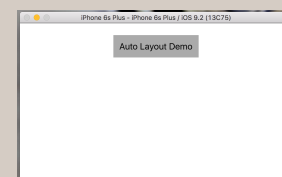
# Auto Layout

External changes:

- External changes occur when the size or shape of your *superview* changes. With each change, you must update the layout of your view hierarchy to best use the available space.
- A common source of external change is a device rotation (orientation) - portrait to landscape or landscape to portrait.



Portrait



Landscape

# Auto Layout

Intrinsic Content Size:

- The ability of a user interface element to determine the size it needs to be - based on its content.

Auto Layout asks the user interface elements how big they need to be and lays out the screen based on that information.

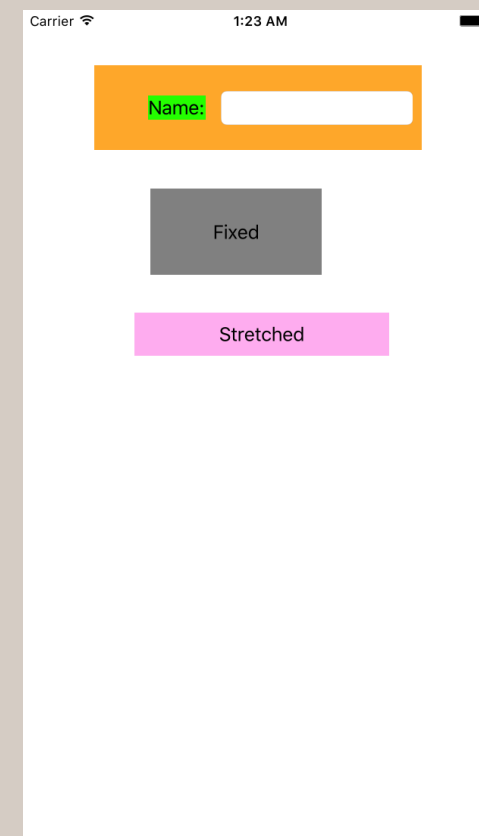
There are times when you want to control an elements size - for example, when displaying an image in an image view. The image you're trying to display could be quite large. You therefore want it scaled to stay within a given size.



# Auto Layout

Demo:

- TestConstraints1
- Demonstrate movement/sizing of views when moving between portrait and landscape orientation.
- NO CODE - all done with constraints in the view controller on the storyboard.



# In-Class Exercise

# In-Class Exercise

- Create a user interface and define auto layout rules to handle device rotation and sizing.
- Rotate the device to prove the correct constraints are defined to handle orientation change.
  - Rotating a simulator:
    - Rotate Left - Command+Left-arrow
    - Rotate Right - Command+Right-arrow
- Run the app on different simulators - 4s (3.5"), 5s (4"), 6s (4.7"), 6s Plus (5.5"), iPad - and rotate.
- Assistant Editor - Preview

<http://www.paintcodeapp.com/news/ultimate-guide-to-iphone-resolutions>

# Storyboards

# Storyboards

What is a storyboard?

A Storyboard is a visual representation of the appearance and flow of your application.

A new project will include a storyboard - Main.storyboard.

You can have any number of storyboards in an application.

# Storyboards

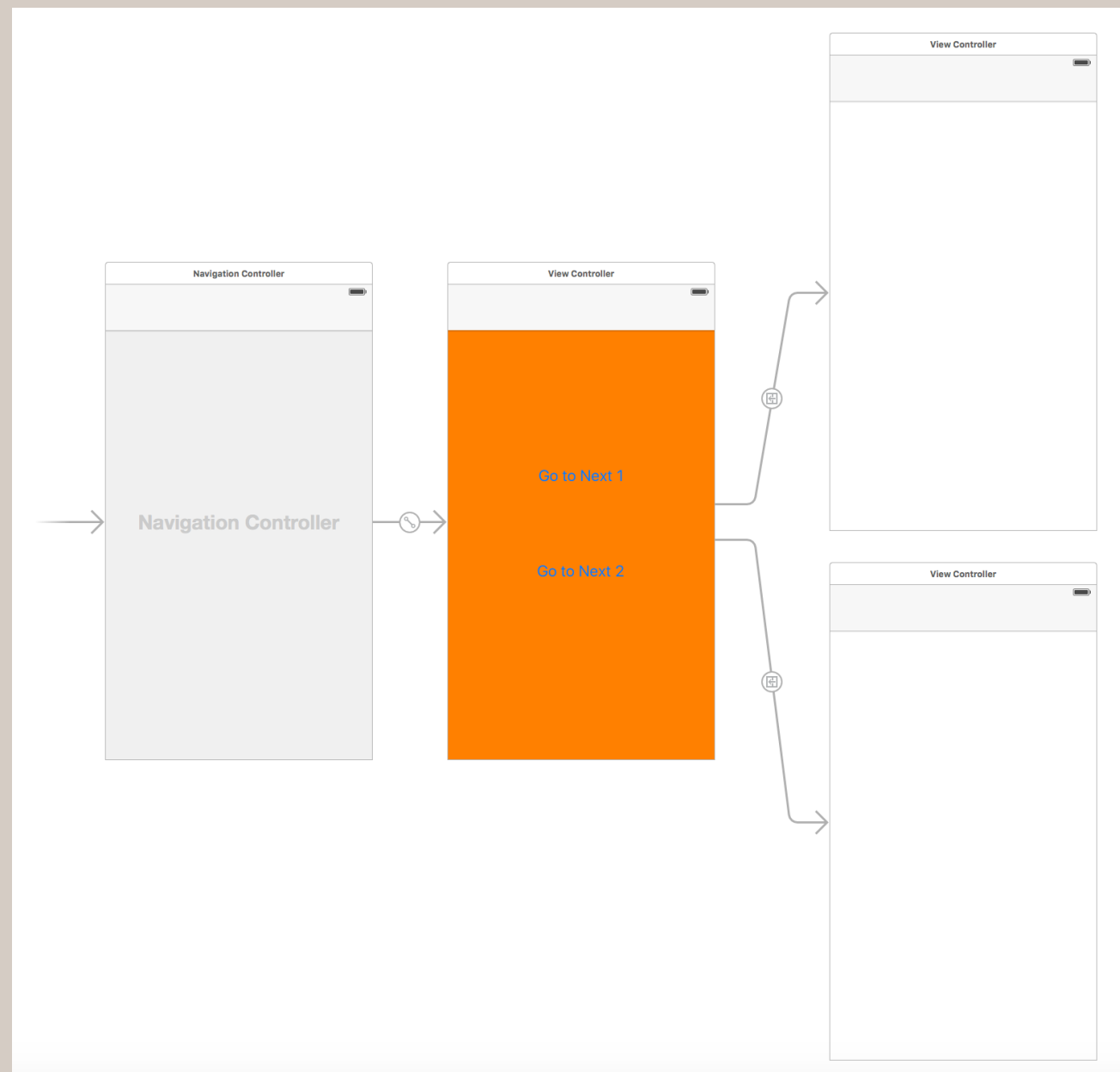
Each view controller in a storyboard is called a scene.

You can have as many scenes in a storyboard as you want.

You can establish connections between scenes using segues.

# Storyboards

Example storyboard with 4 scenes



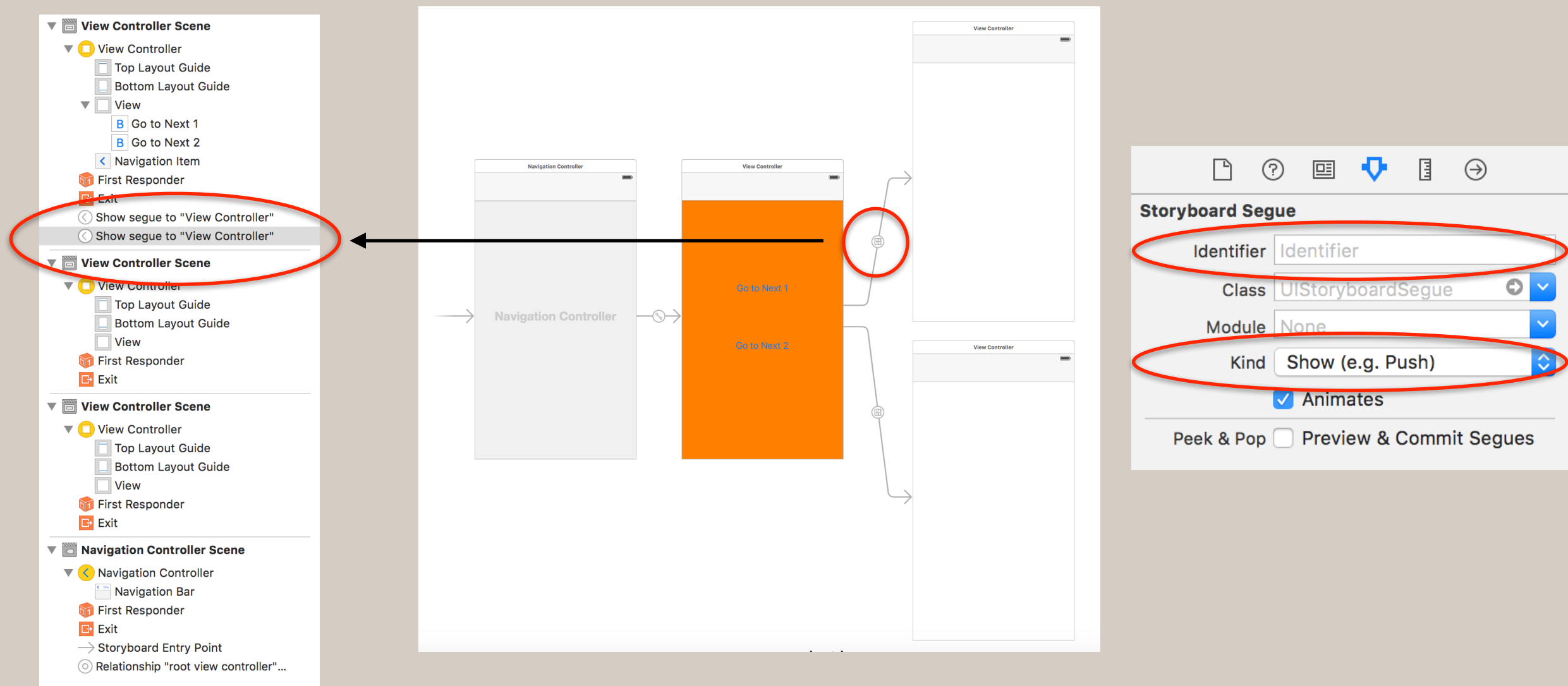
# Segues



# Segues

What are segues?

- Connections between *scenes* in a storyboard.



# Segues

When a segue is invoked, a method is called that allows you to set state in the target view controller.

This is an optional method you can implement.

```
override func prepare(for segue: UIStoryboardSegue, sender: Any?) {  
    if segue.identifier == "Detail1" {  
        if let destinationVC = segue.destinationViewController as? ViewController2 {  
            destinationVC.message = "Hello from View Controller 1"  
        }  
    }  
}
```

See *TestNavigationControllerSegue* sample app in Canvas.

# In-Class Exercise

# In-Class Exercise

Segues:

- Create a user interface with three view controllers.
  - Each one having a local property.
- Embed the first view controller in a navigation controller.
- Set the background color for each view controller to a different color.
- Add two buttons to the first view controller.
- Add constraints to fix the size of the buttons and have them centered horizontally.
- Create a segue from each button to one of the other view controllers.