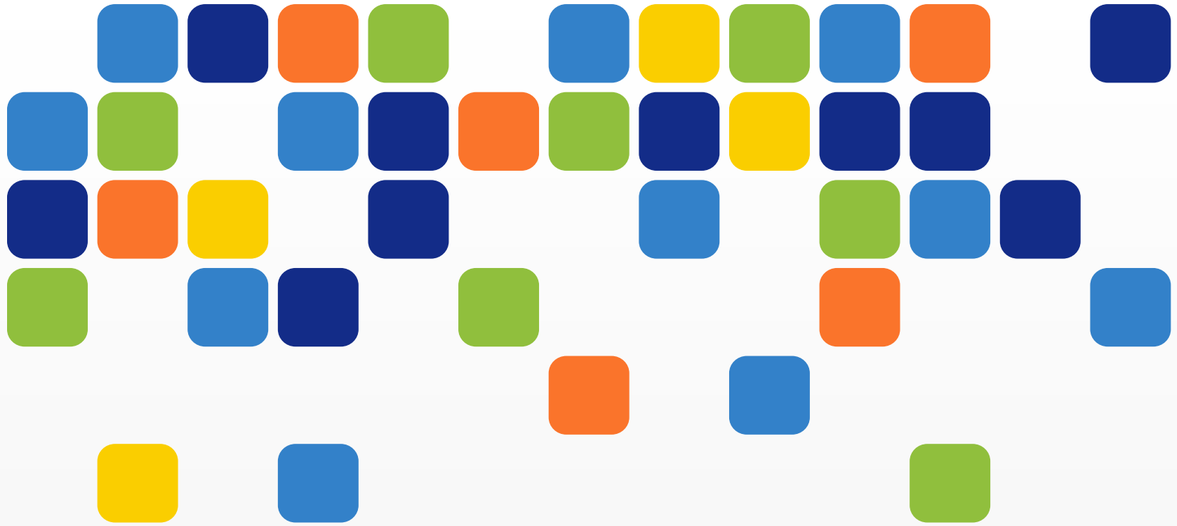


Winning with IB and Xcode 6

Wonderful Cool Things That Help You

CapTech®

Agenda

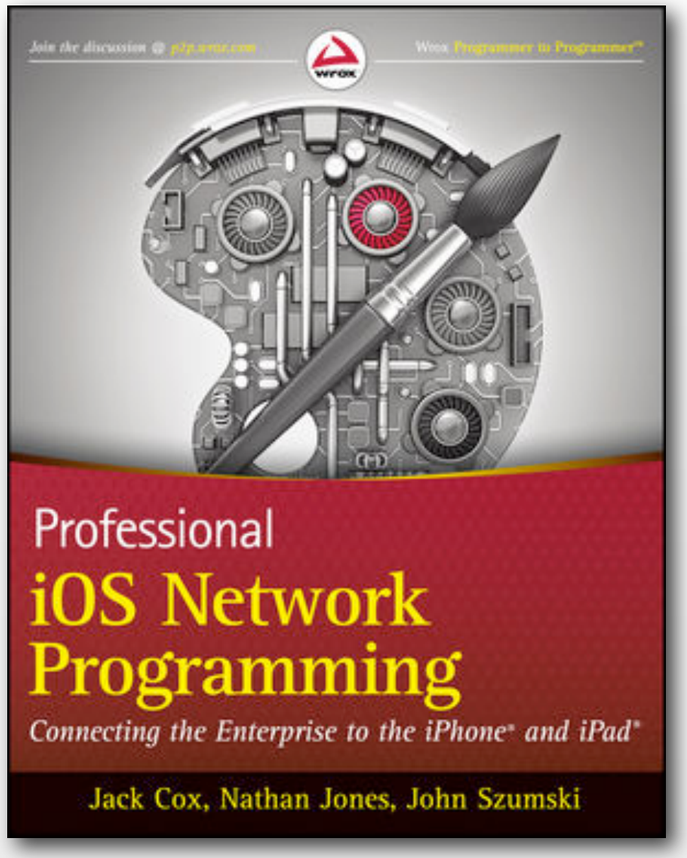
- Introduction
- Quick History of IB
- New Stuff
  - IBDesignable/IBInspectable
  - Size Classes

Introduction

- self = [
  - Jack Cox
  - Managing Director, Mobile Technologies at CapTech Consulting
  - Author: Professional iOS Network Programming
  - Husband, Father, Christian];
- Contact Info

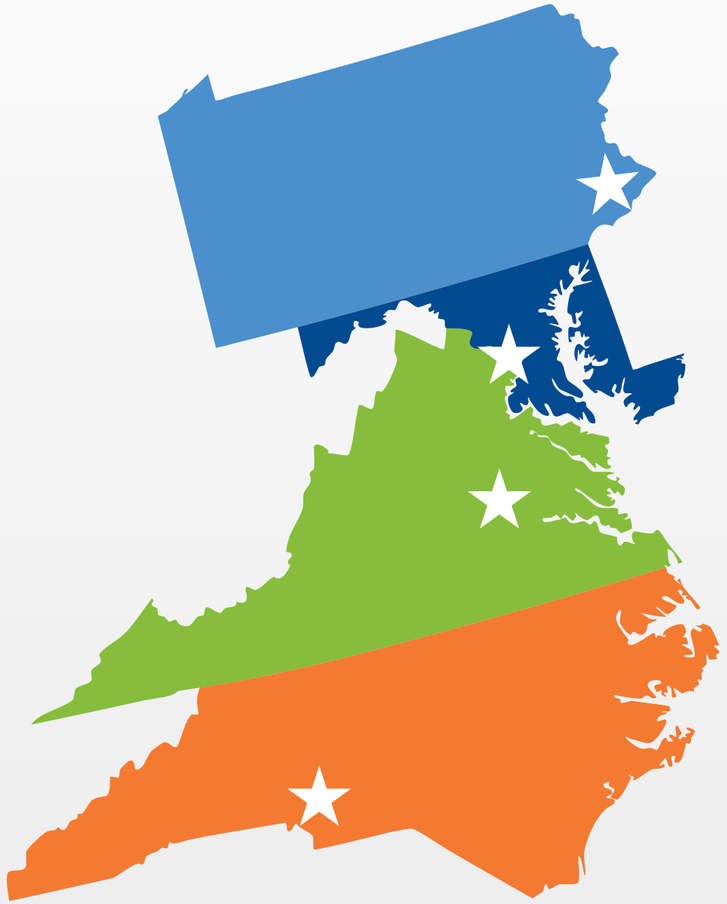
  jcox@captechconsulting.com

  @jcox_mobile



Who is CapTech

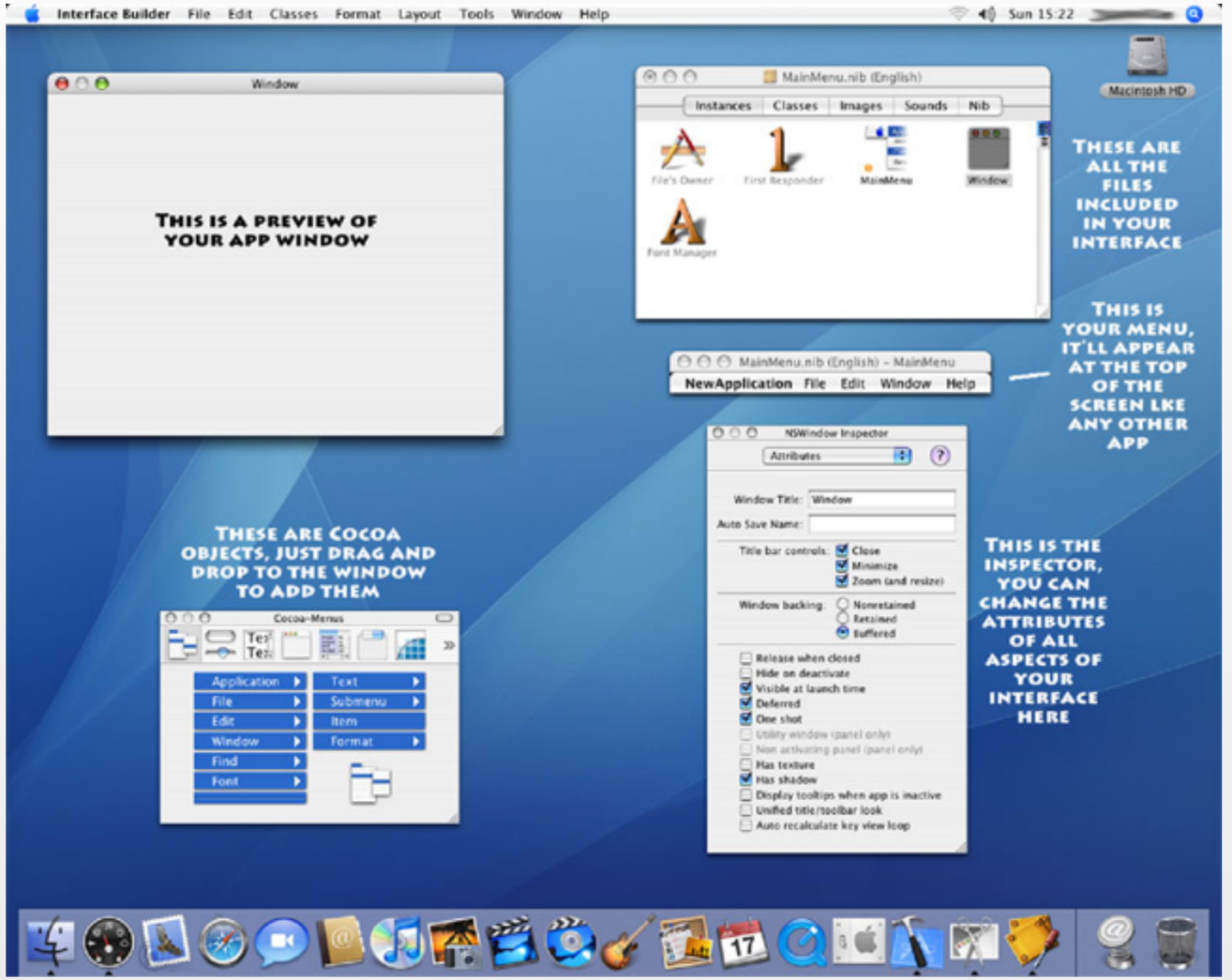
- Mid sized consulting firm
- Based in the Mid-Atlantic region
- ~500 Consultants
- ~90 involved in mobile projects

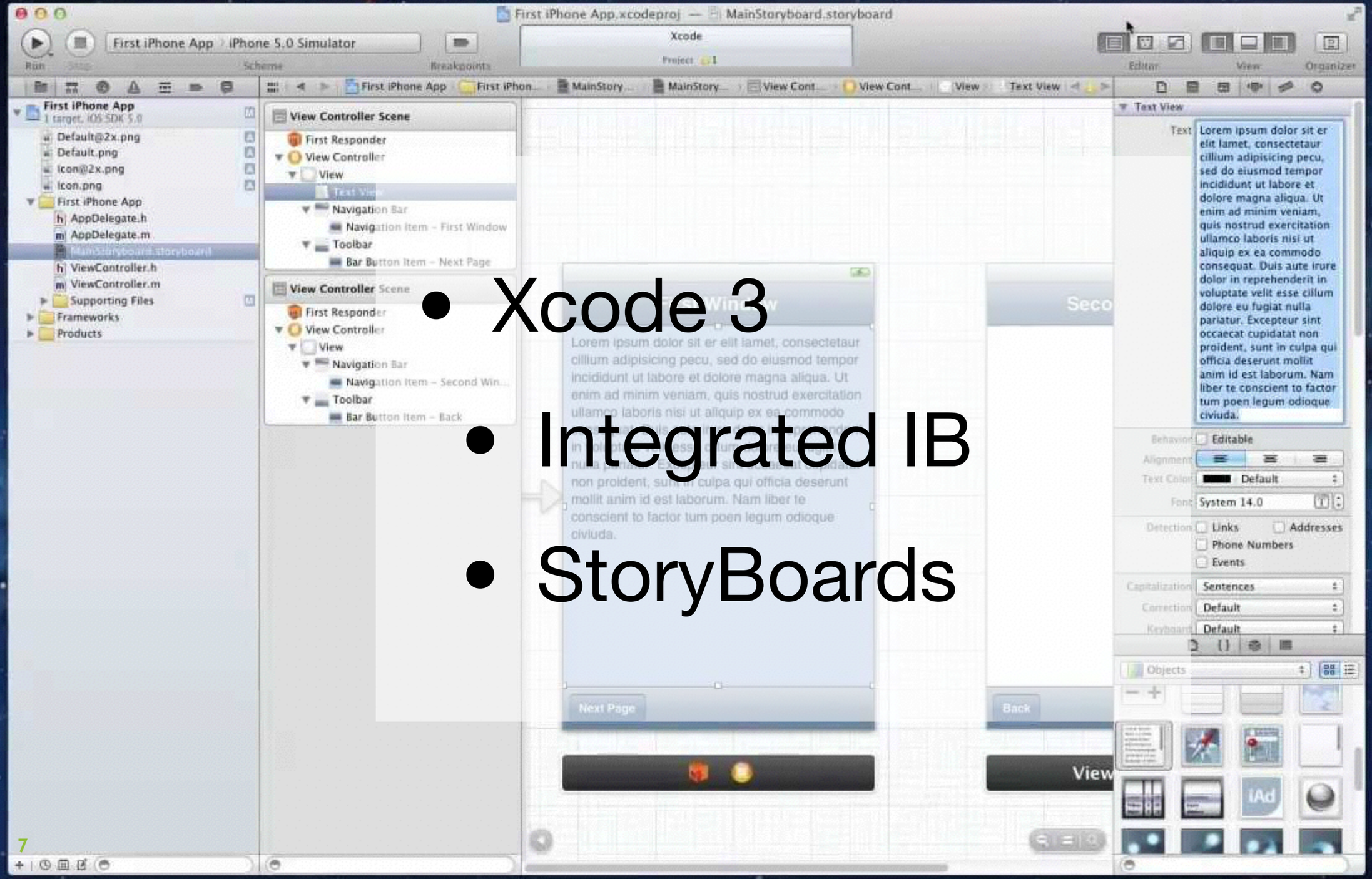


History of Interface Builder

History of IB and Xcode

- Xcode 2
  - IB Separate From Xcode
  - Almost like an after thought

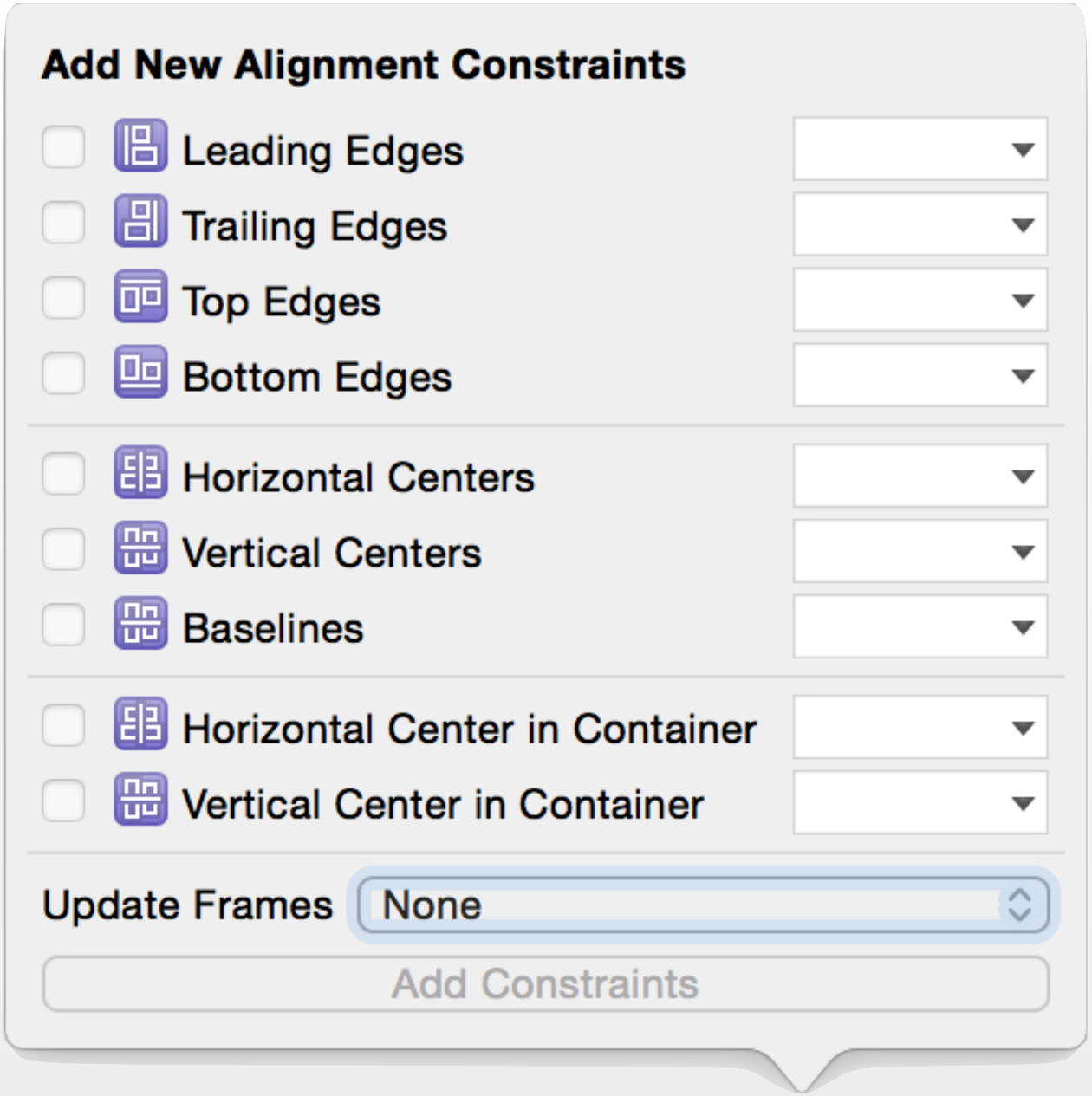




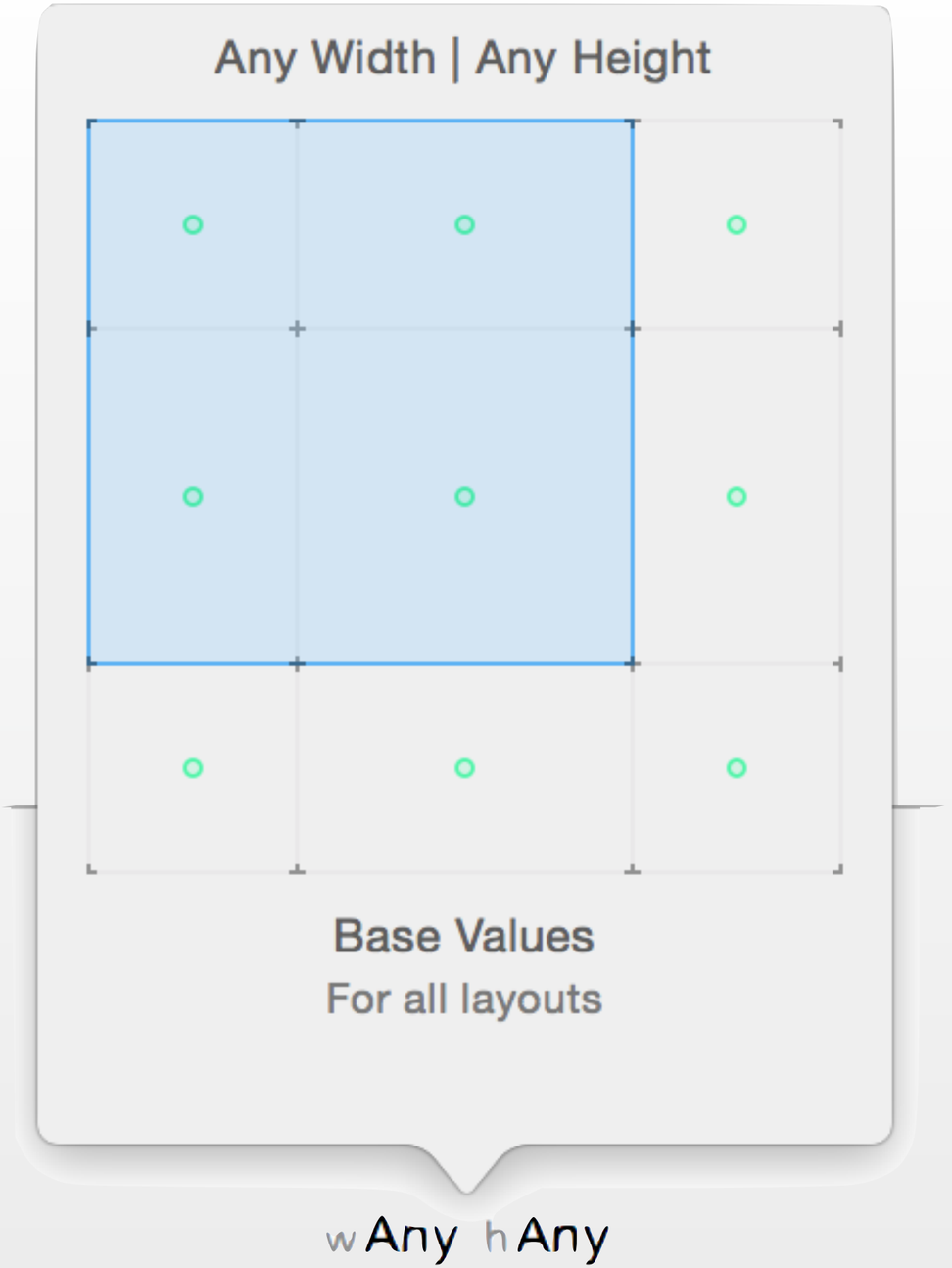
- Xcode 3
- Integrated IB
- StoryBoards

- Xcode 4
  - Auto Layout
  - Enough Said

- Xcode 5
  - Usable Auto Layout!



- Xcode 6
  - Size Classes
  - IBDesignable/ IBInspectable



Good Things of Xcode 6

- IBDesignable
- IBInspectable
- Size Classes

IBDesignables and IBInspectables

- Cool
- Easy
- Surprisingly Useful

IBDesignables - What it does

- Allows live preview of UIView subclasses in IB
- Easy Peasy:

```
IB_DESIGNABLE
@interface EllipseView : UIView
```

```
@IBDesignable class StarView : UIView {
```

IBDesignables - What It Does

- Demo

How Does It Work?

- IB Compiles UIView designable subclasses
- Loads dynamically into Xcode runtime
- Each time property changes or designable class changes
  - New class reloaded and redisplayed in IB

Characteristics

- No change to normal drawRect:

- IBInspectables not required

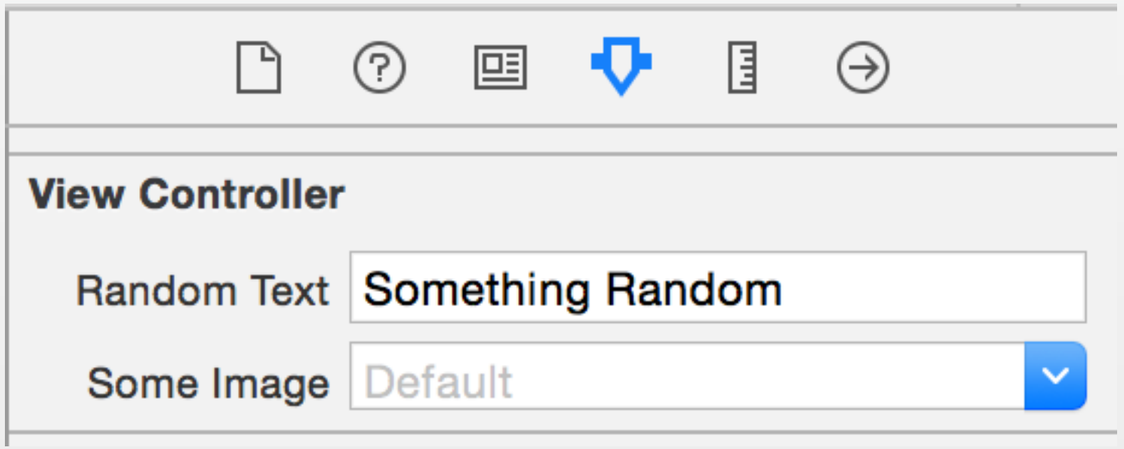
- Must be auto layout based

- Best used with Frameworks

- Beware of 3rd party designables

IBInspectable - What It Does:

```
@property (nonatomic, strong)
          IBInspectable NSString *randomText;
@property (nonatomic, strong)
          IBInspectable UIImage  *someImage;
```



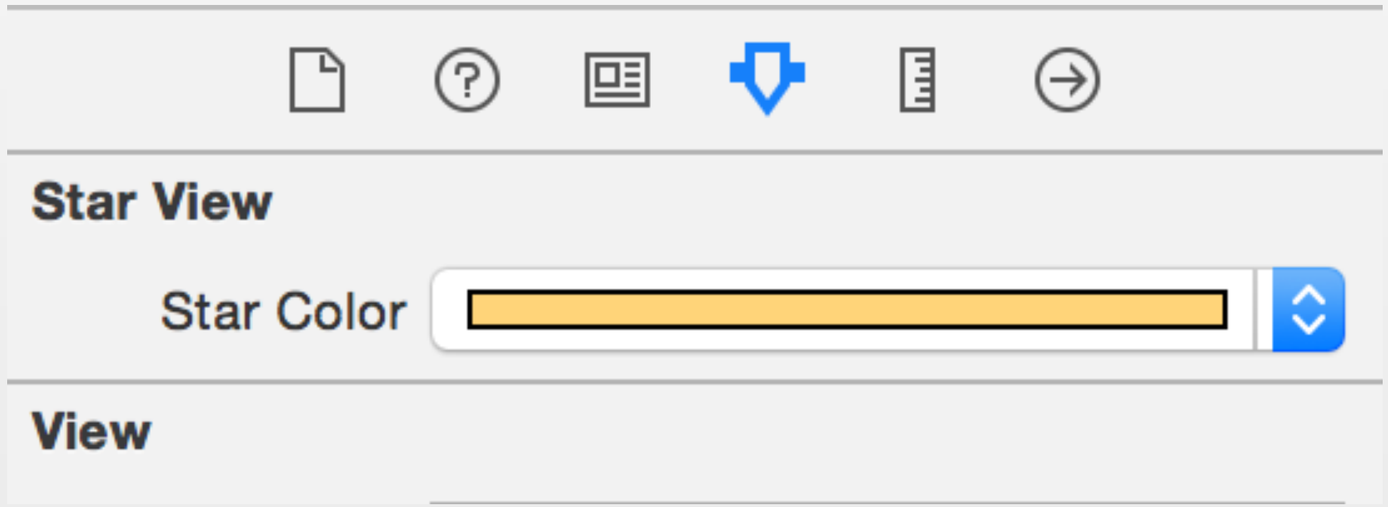
IBInspectable - What It Does

- Demo

- Enables setting object properties from IB
- Also Easy Peasy:

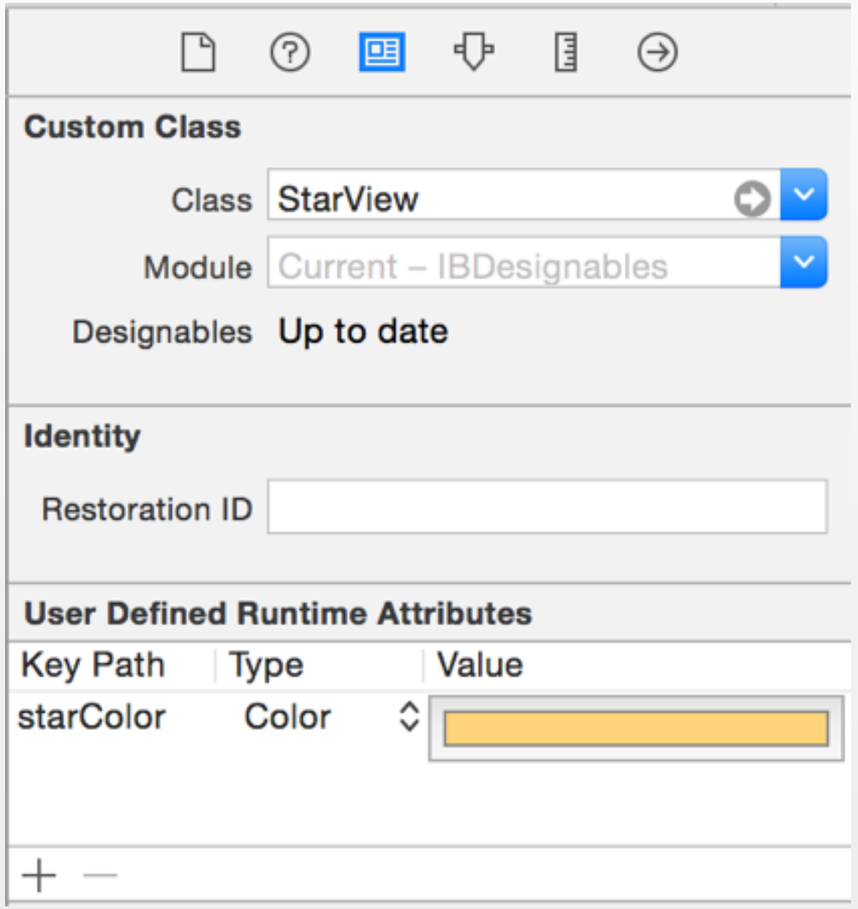
```
@property (nonatomic, retain)
            IBInspectable UIColor *fillColor;

@IBInspectable var starColor:UIColor
```



How does it do it

- Reads annotations on classes
- Wraps existing runtime attribute functionality
- Eliminates the need to remember property names



Characteristics

- Supports many property types:
  - BOOL
  - Number
  - String
- Point
- Rect
- Range
- Color
- Image

- Works against any object you can instantiate with IB

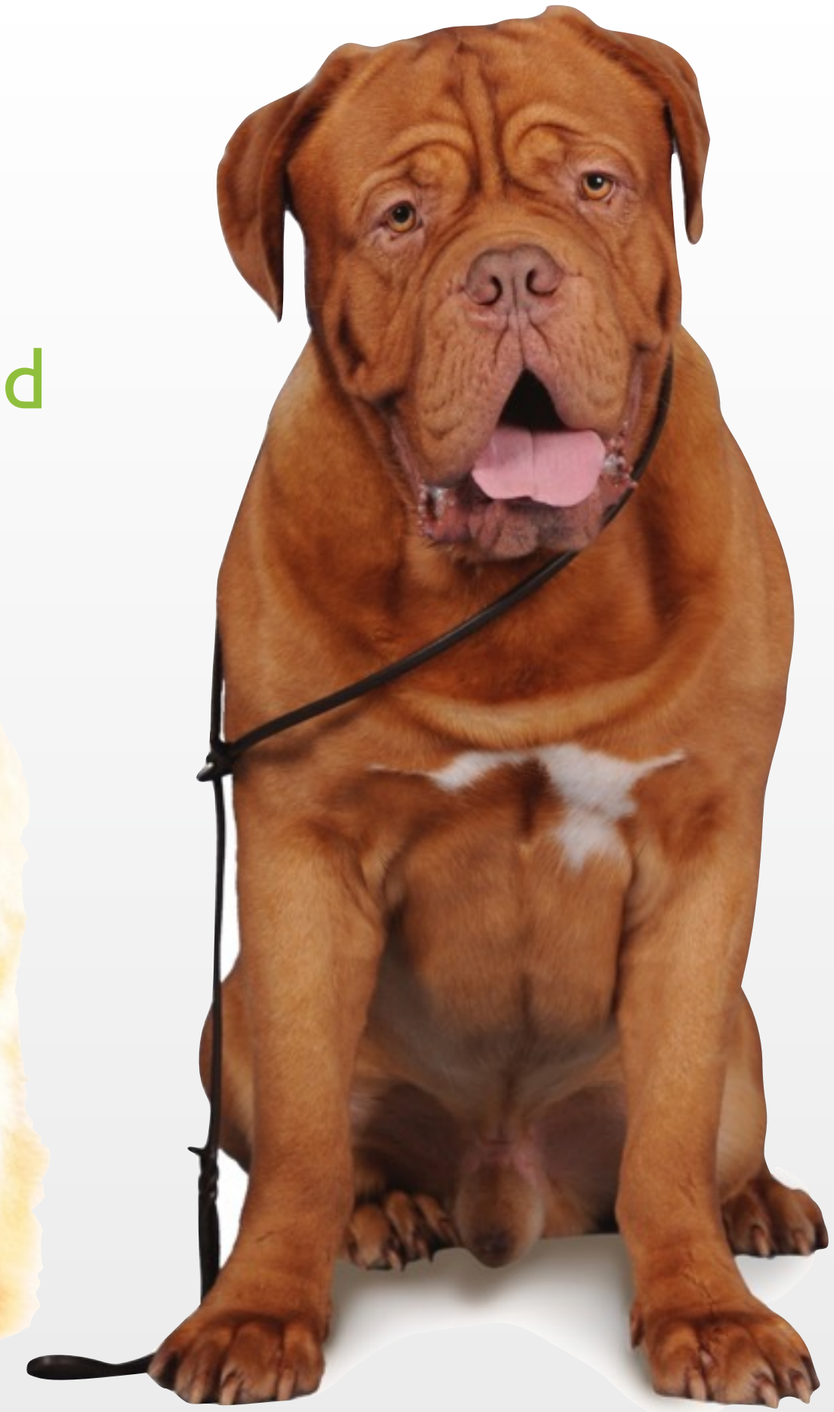
- Works with Obj-C categories

Extending UIView

- Demo

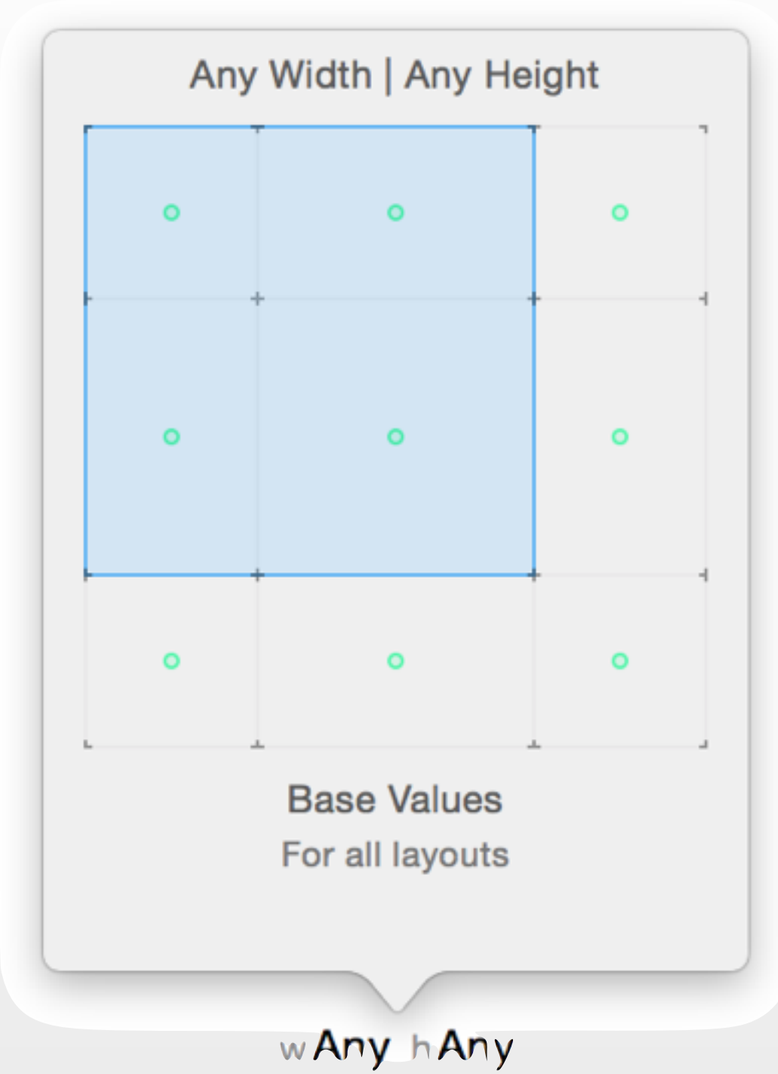
Size Classes

The solution we didn't know we needed

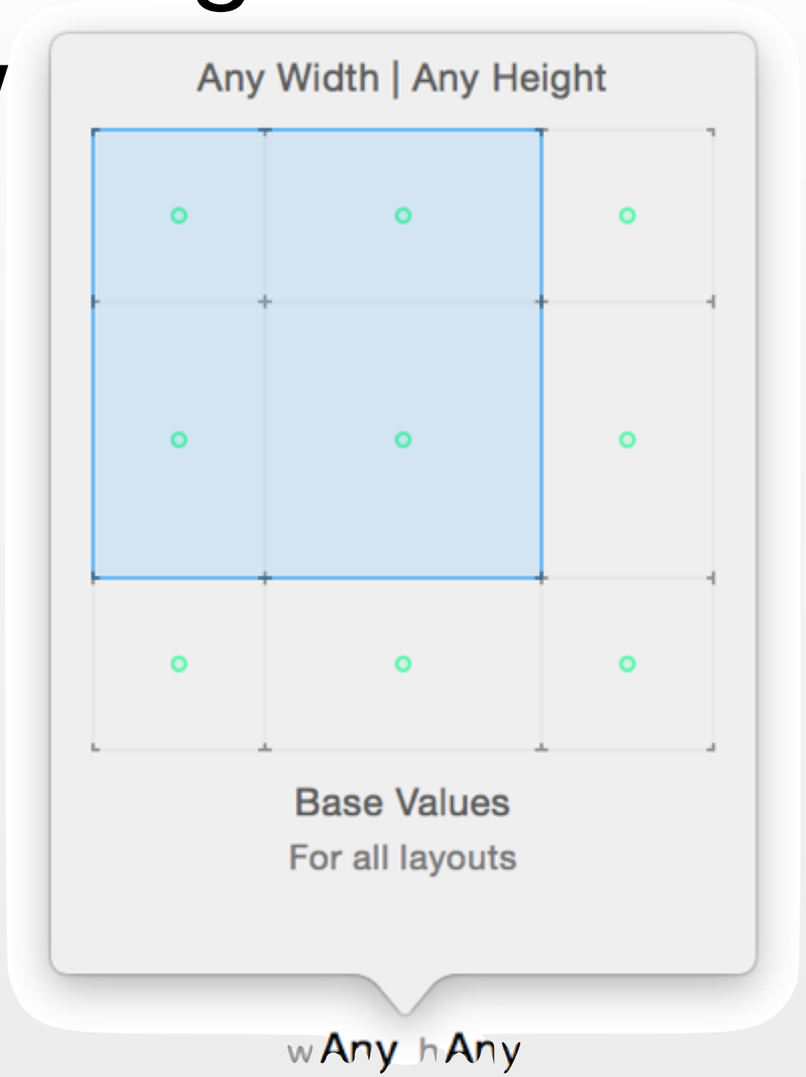


What Does It Do

- Define multiple layouts in a single XIB/Storyboard
- Size of device drives layout chosen, with caveats



- Size classes describe the height and width allowed to the view
  - Regular Height/Width
  - Compact Height/Width
  - Any Height/Width



Non-Plus Phone Classes

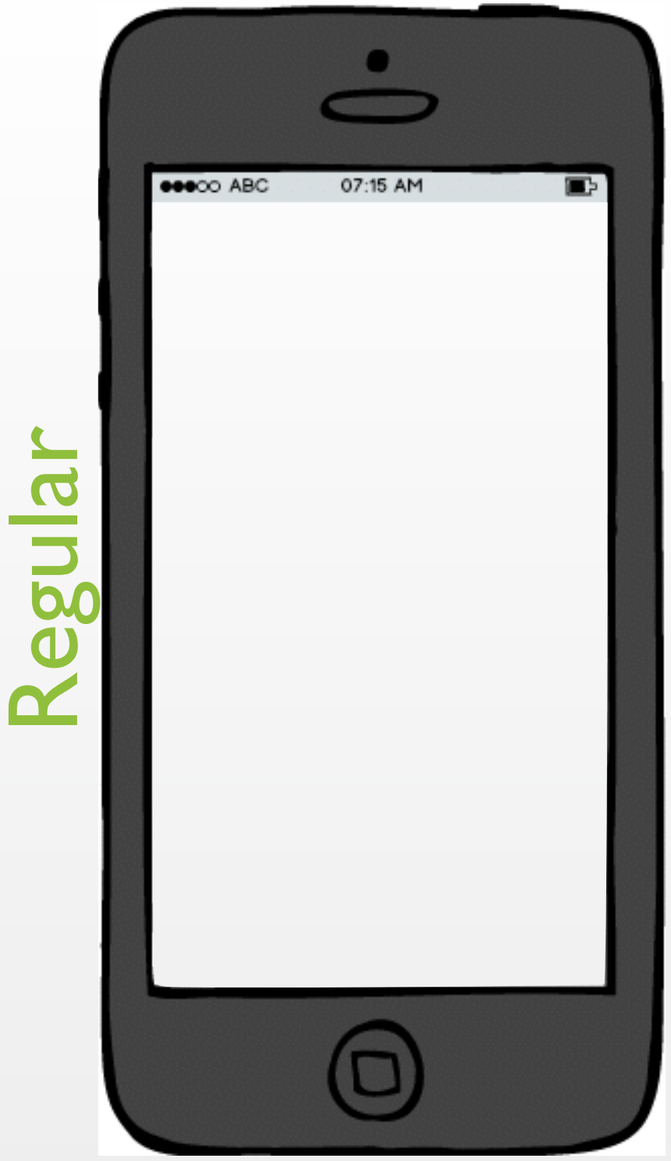
Regular

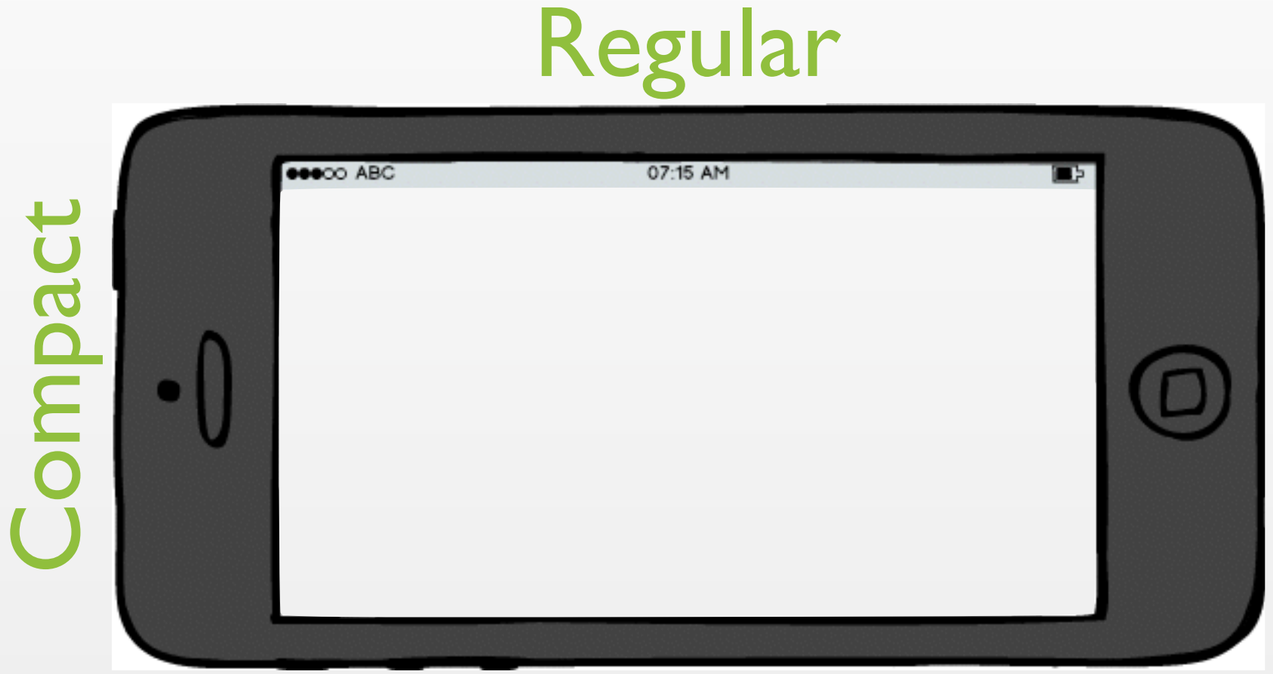
Compact

Compact

Compact

Plus Phone Classes





iPad Size Classes

Regular

Regular

Regular

What You Do

- Define a layout with all possible size classes fully specified
  - Constraints and views can be present or absent in a size class
- Use the Any-Any class as the catch-all

What It Does

1. When device size changes or XIB loads
    1. Rotation
    2. Some mysterious other times
2. XIB Loader activates objects based on your specifications
    1. Views
    2. Constraints
    3. Images

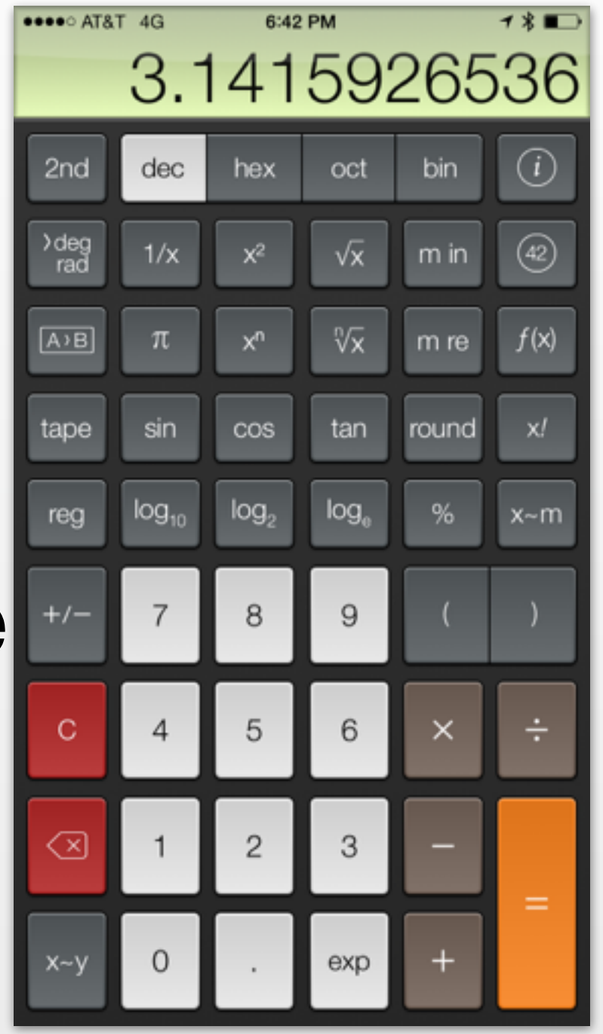
What it lets you do

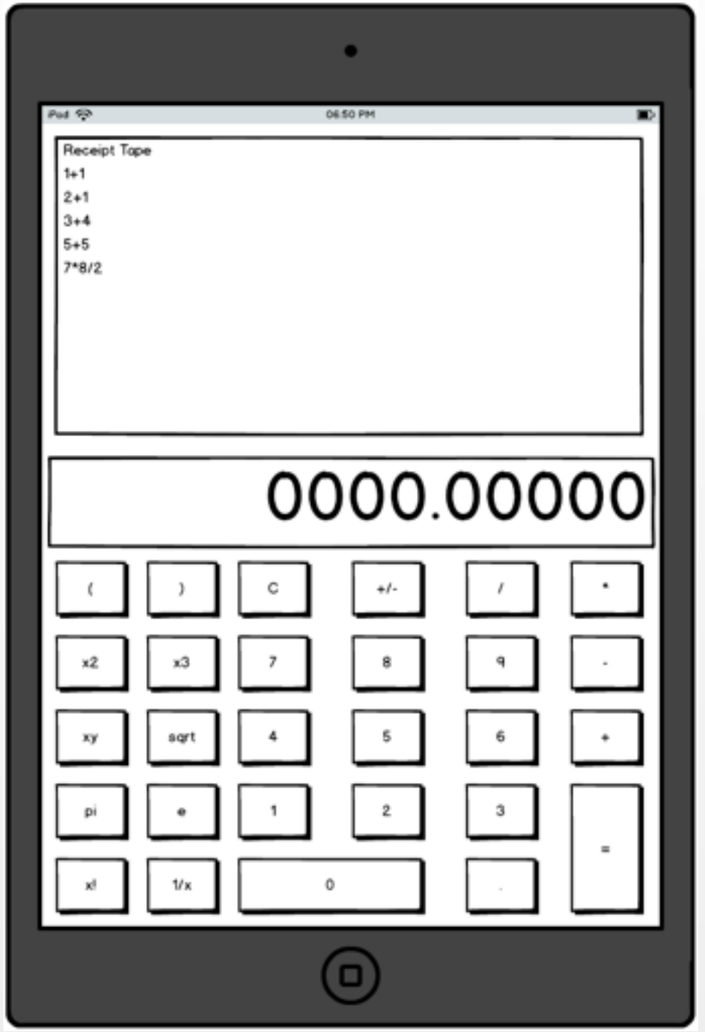
- Build a single view for all devices
- Radically change the layout based on device size

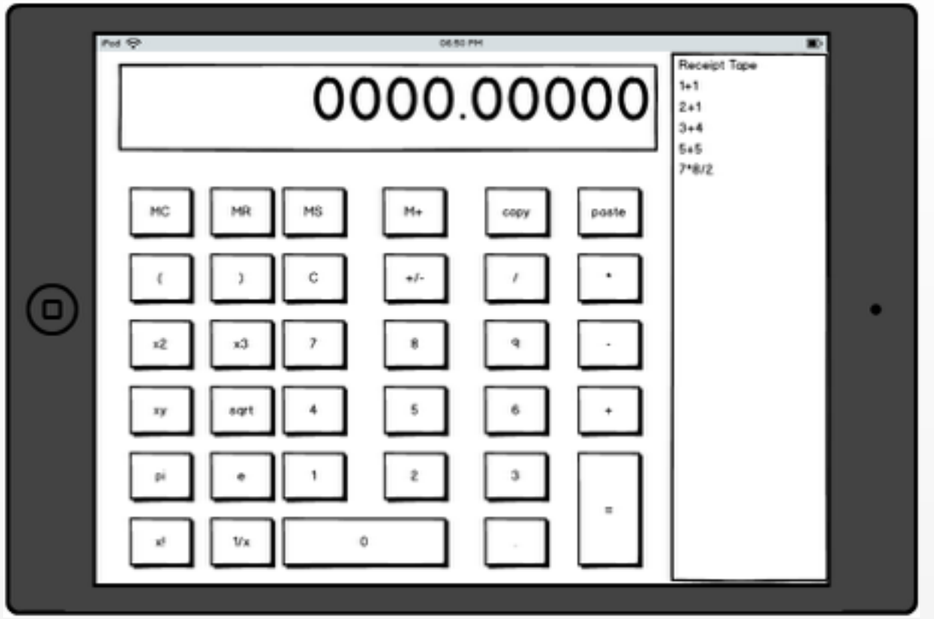
Enough Talk Already…

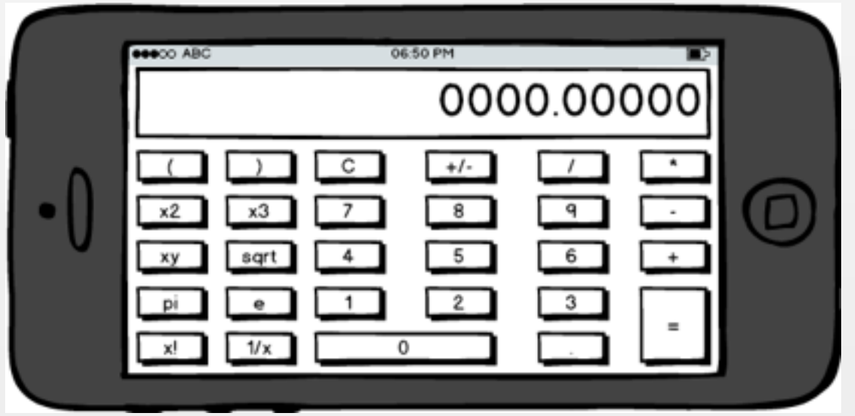
Case Study

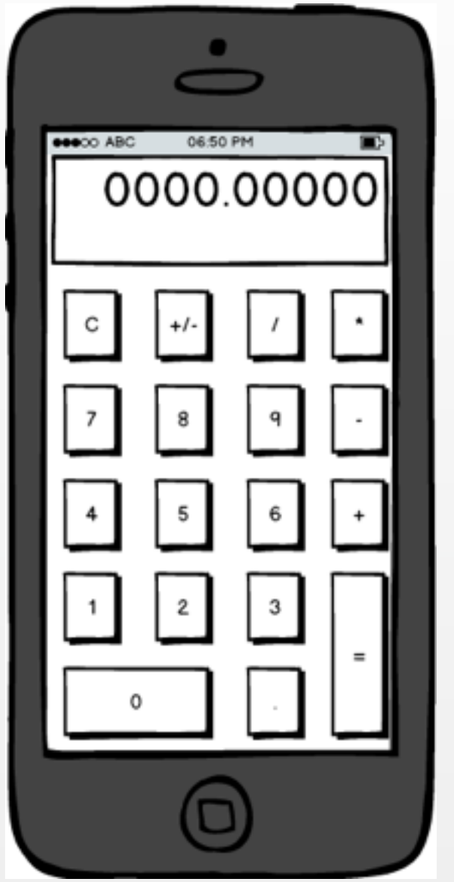
- Client wants a calculator app
  - Universal app
  - Very different layouts per device
  - All orientations supported




iPad
06:50 PM
Receipt Tape
1+1
2+1
3+4
5+5
7*8/2
0000.00000
( ) C +/- / *
x2 x3 7 8 9 -
xy sqrt 4 5 6 +
pi e 1 2 3 =
x! 1/x 0 .


iPad
06:50 PM
0000.00000
MC MR MS M+ copy paste
( ) C +/- / *
x2 x3 7 8 9 -
xy sqrt 4 5 6 +
pi e 1 2 3 =
x! 1/x 0 .
Receipt Tape
1+1
2+1
3+4
5+5
7*8/2

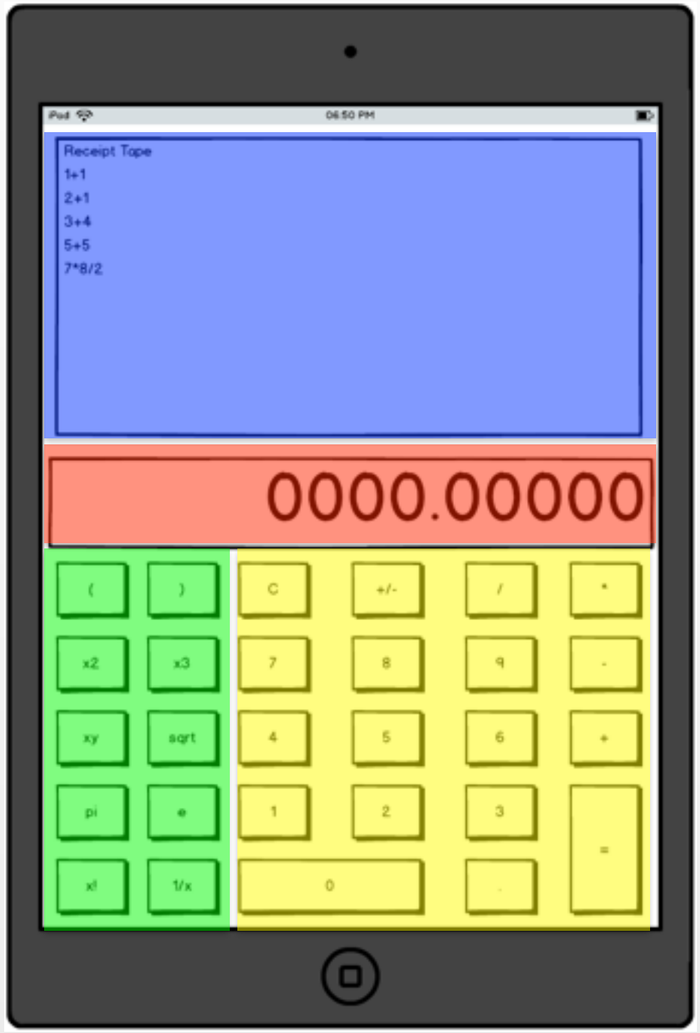

ABC
06:50 PM
0000.00000
( ) C +/- / *
x2 x3 7 8 9 -
xy sqrt 4 5 6 +
pi e 1 2 3 =
x! 1/x 0 .

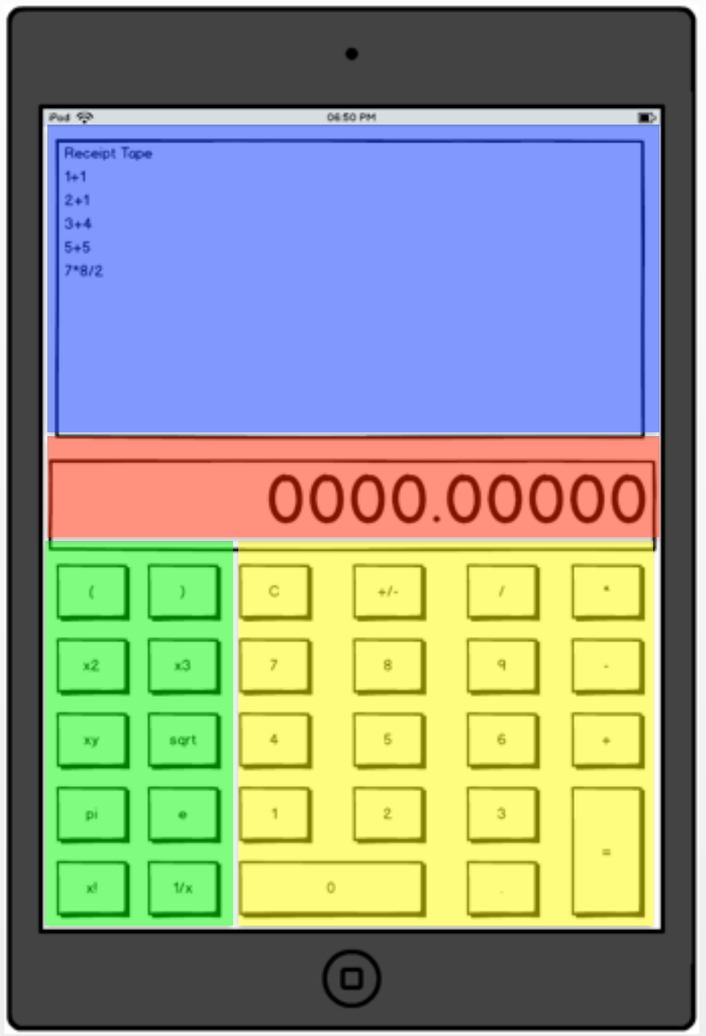

ABC
06:50 PM
0000.00000
C +/- / *
7 8 9 -
4 5 6 +
1 2 3 =
0 .

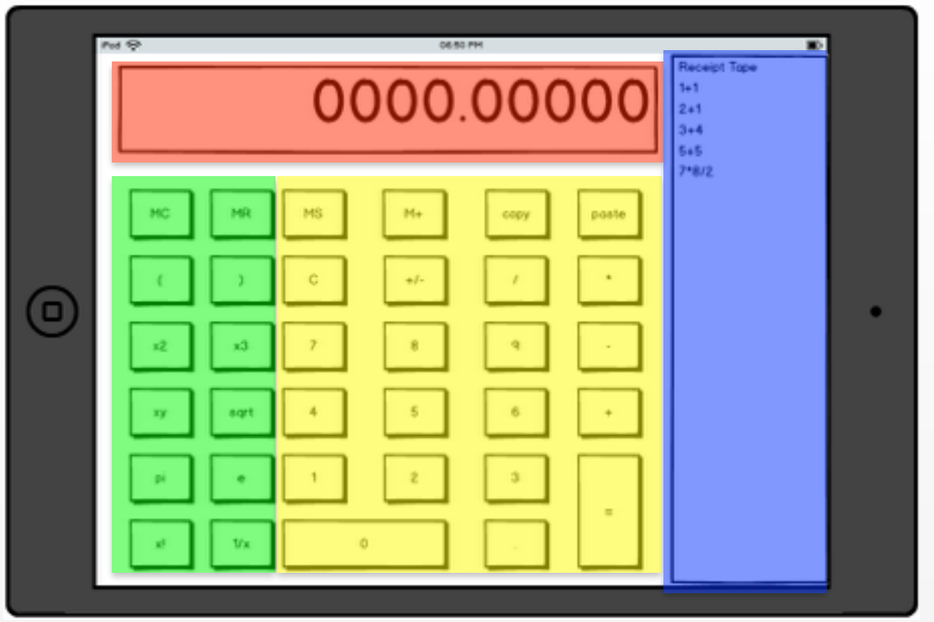
So…How are we going to build this?

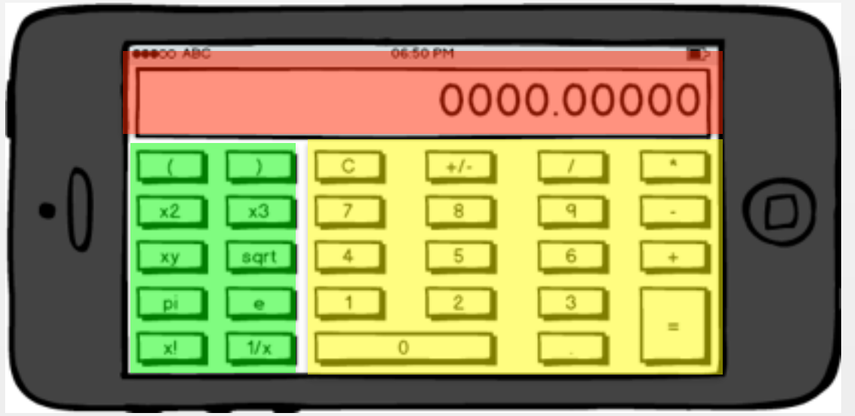
Simplify The Problem

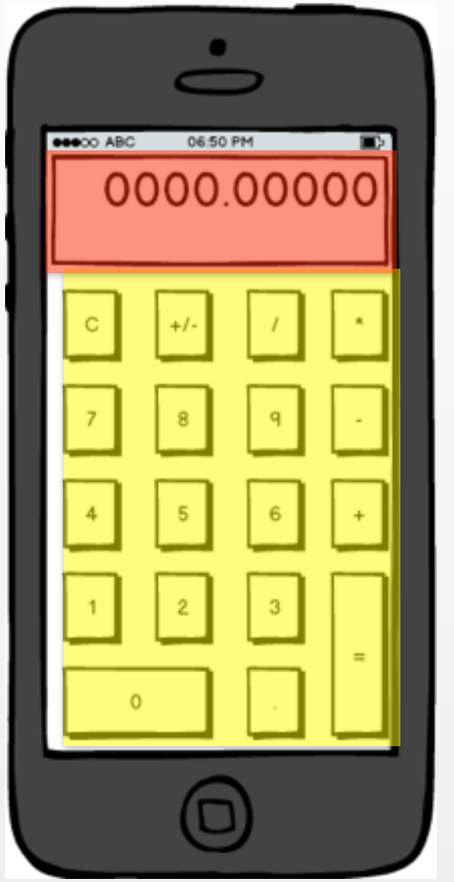
- 4 Major sections of the display
- Ignore the internals of each major section



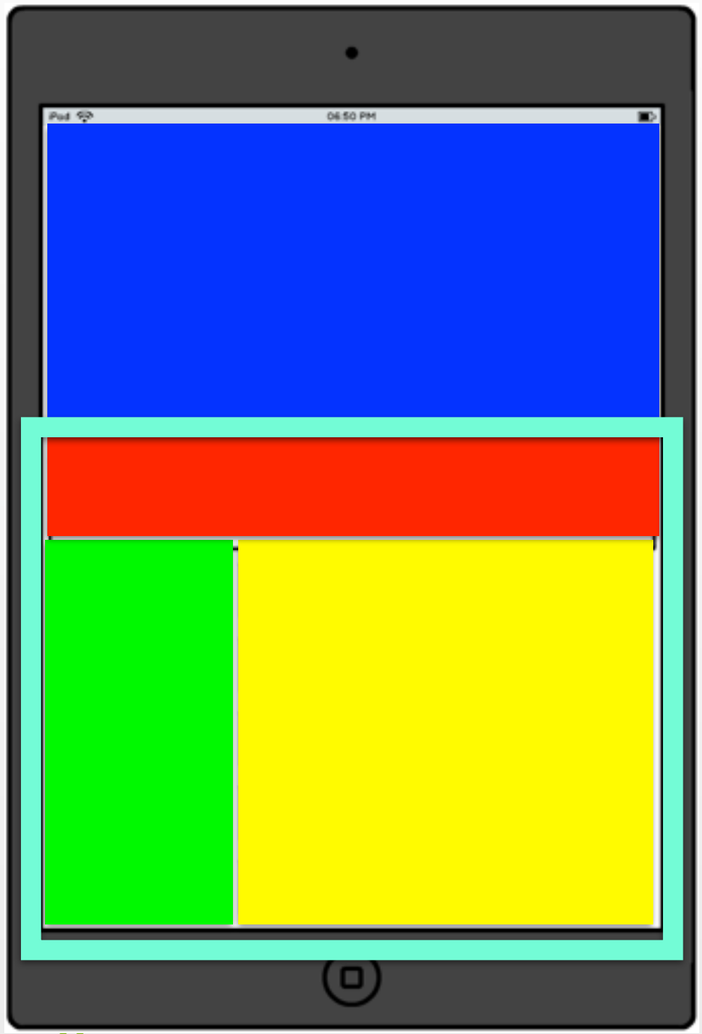

Receipt Tape
1+1
2+1
3+4
5+5
7*8/2
0000.00000
( ) C +/- / *
x2 x3 7 8 9 -
xy sqrt 4 5 6 +
pi e 1 2 3 =
x! 1/x 0 .

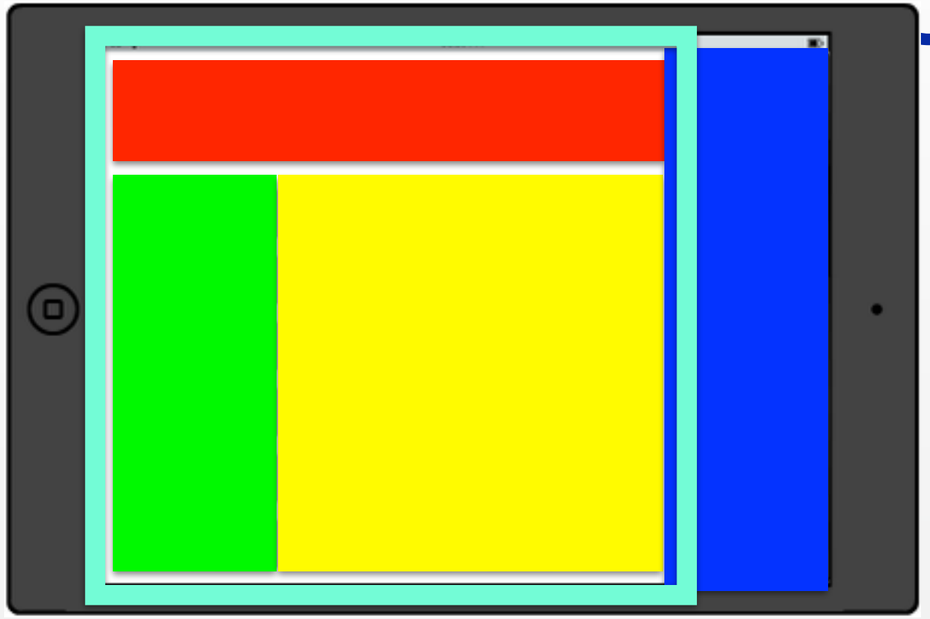

0000.00000
MC MR MS M+ copy paste
( ) C +/- / *
x2 x3 7 8 9 -
xy sqrt 4 5 6 +
pi e 1 2 3 =
x! 1/x 0 .
Receipt Tape
1+1
2+1
3+4
5+5
7*8/2

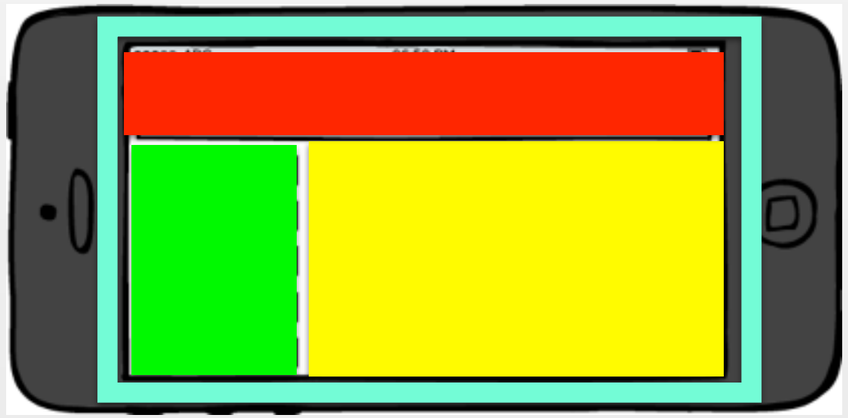

ABC 06:50 PM
0000.00000
( ) C +/- / *
x2 x3 7 8 9 -
xy sqrt 4 5 6 +
pi e 1 2 3 =
x! 1/x 0 .

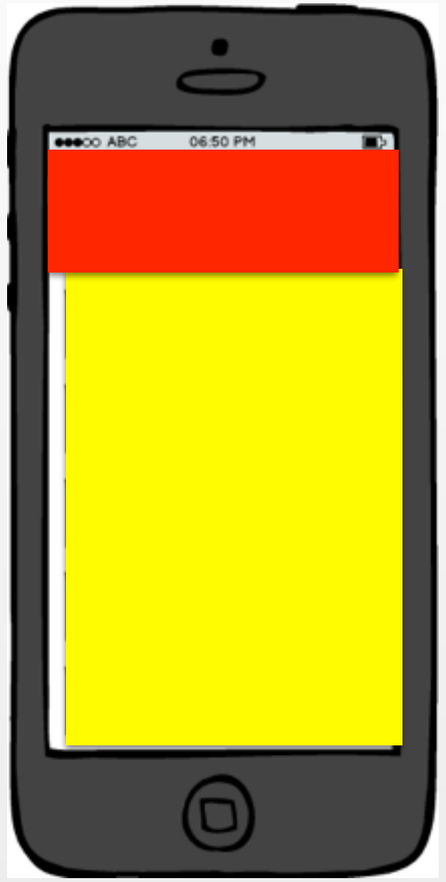

ABC 06:50 PM
0000.00000
C +/- / *
7 8 9 -
4 5 6 +
1 2 3 =
0 .

Find The Most Common Structure



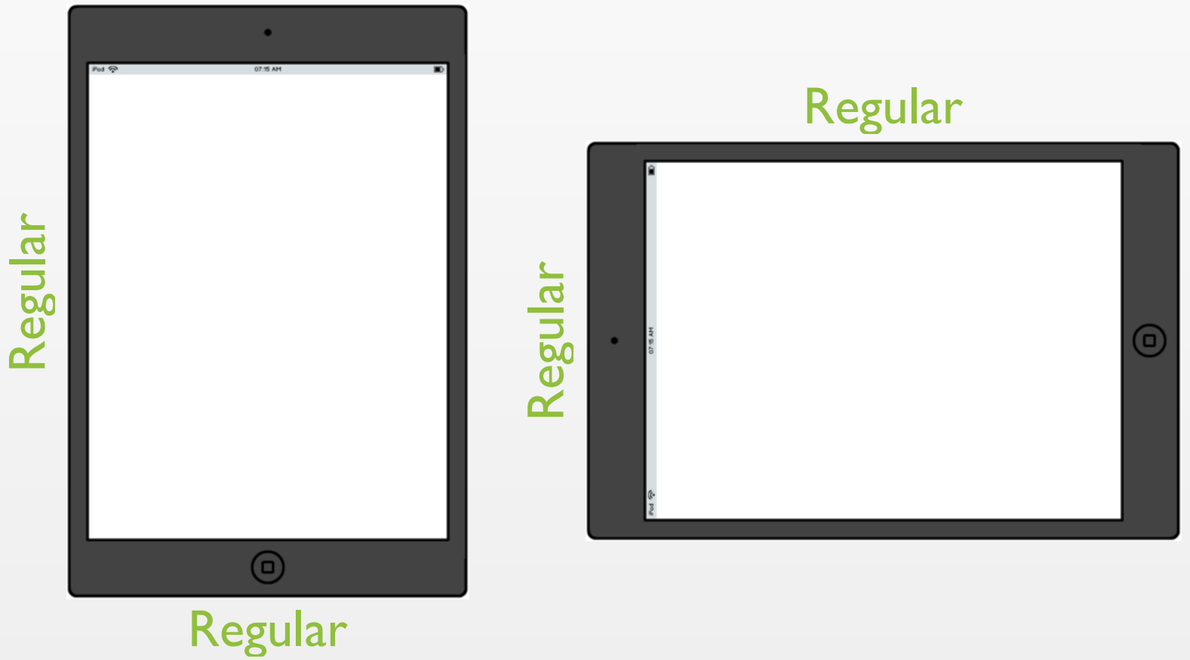






What About the iPad?

- Both orientations have the same size classes
- Oh No!



Trait Override View Controllers

- You can tell a child View Controller what size class to use
- Feels like a hack
  - Apple provides one

How To Use Trait Overrides?

Define 4 distinct layouts to match match devices and orientations

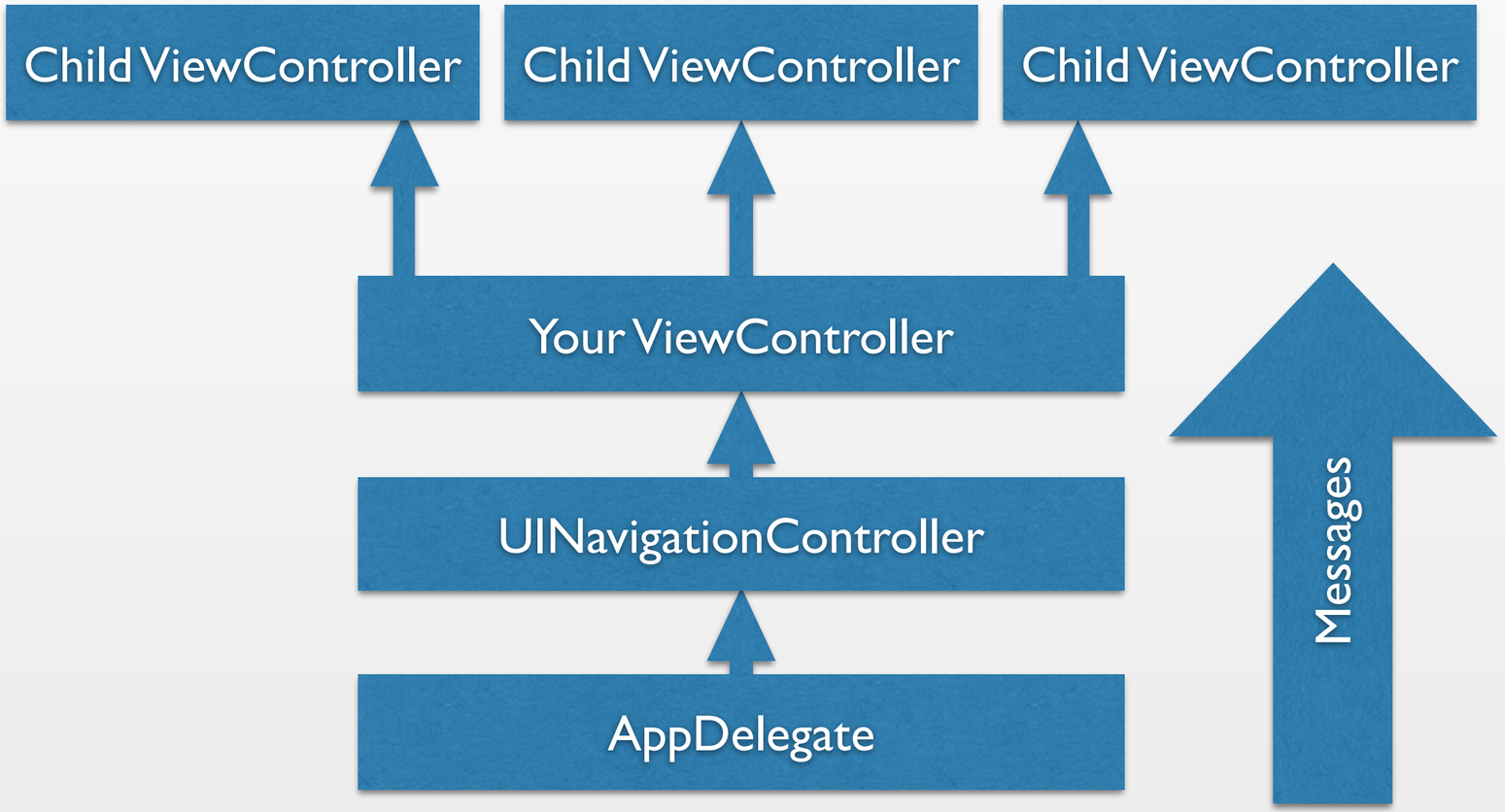

Regular
Compact


Regular
Regular

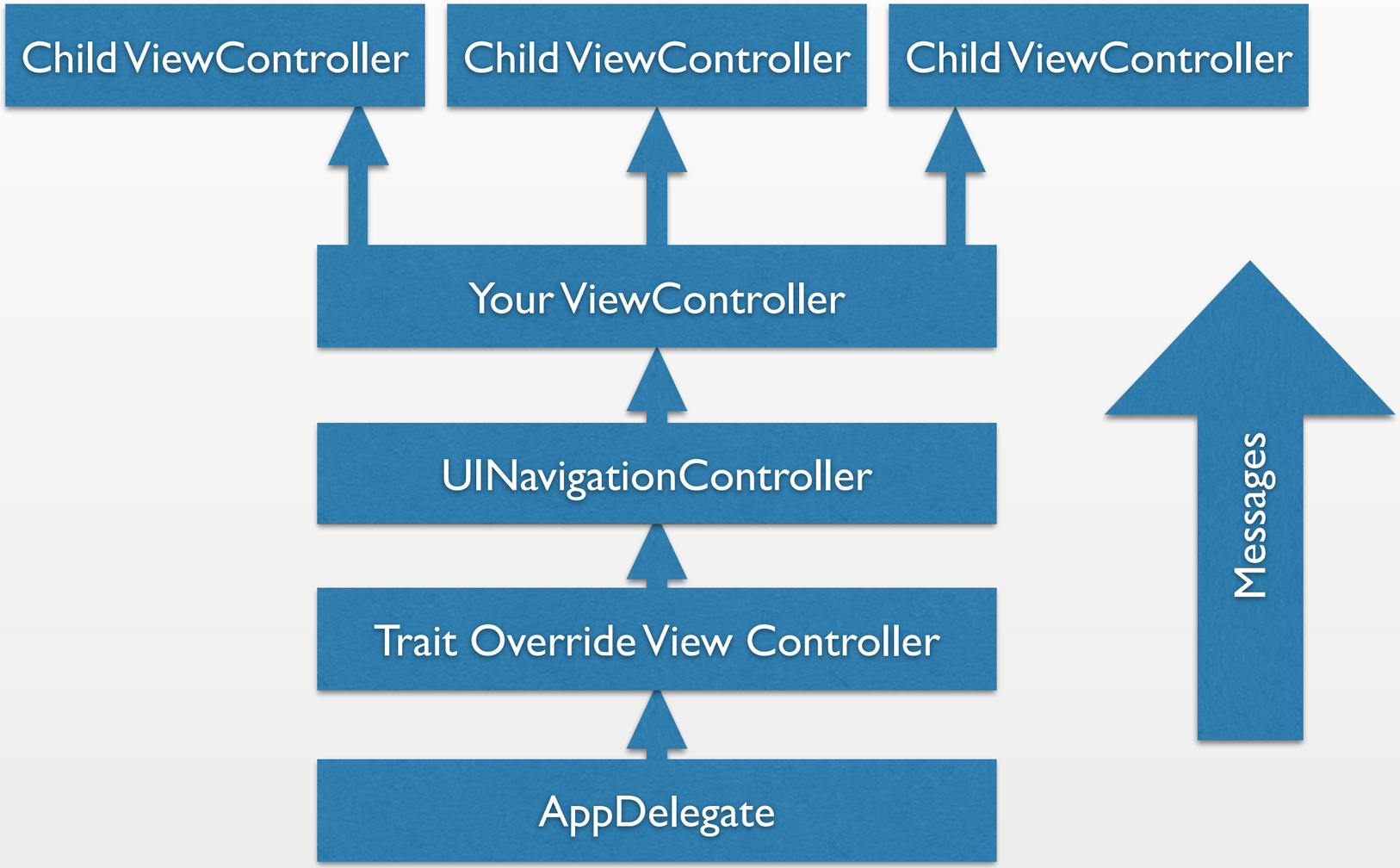

Compact
Regular


Compact
Compact

View Controller Containment



View Controller Containment



Forcing Size Classes

SWIFT

```swift
func setOverrideTraitCollection(_ collection: UITraitCollection?,
         forChildViewController childViewController: UIViewController)
```

OBJECTIVE-C

```objc
- (void)setOverrideTraitCollection:(UITraitCollection *)collection
            forChildViewController:(UIViewController *)childViewController
```

Overridable Traits

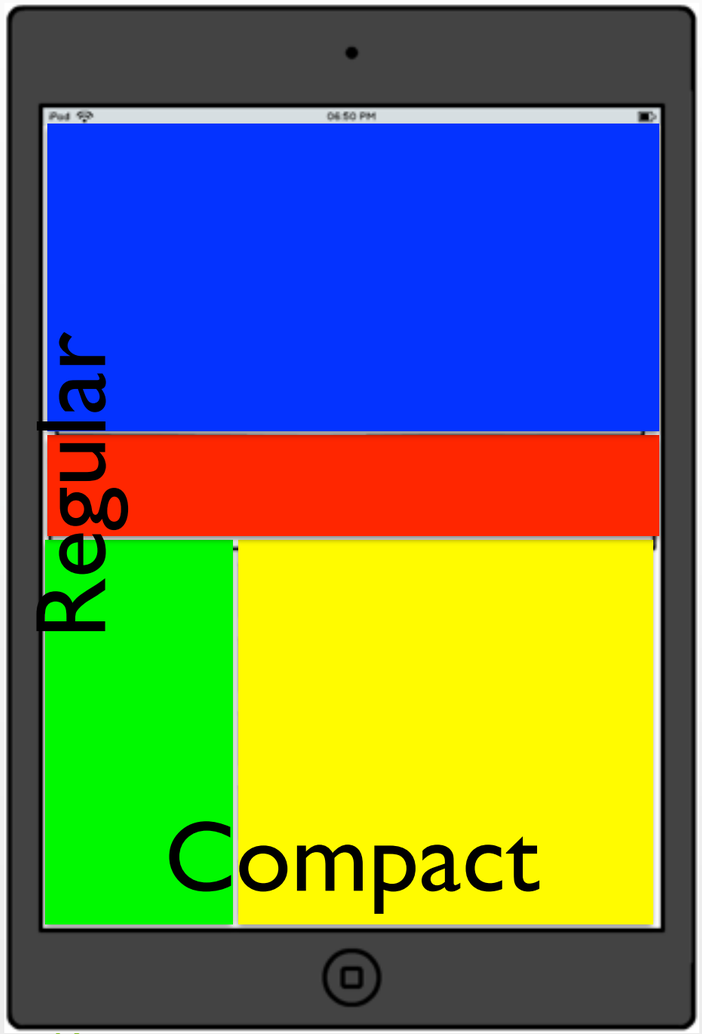
- Horizontal Size Class
- Vertical Size Class
- Interface Idiom
- Display Scale

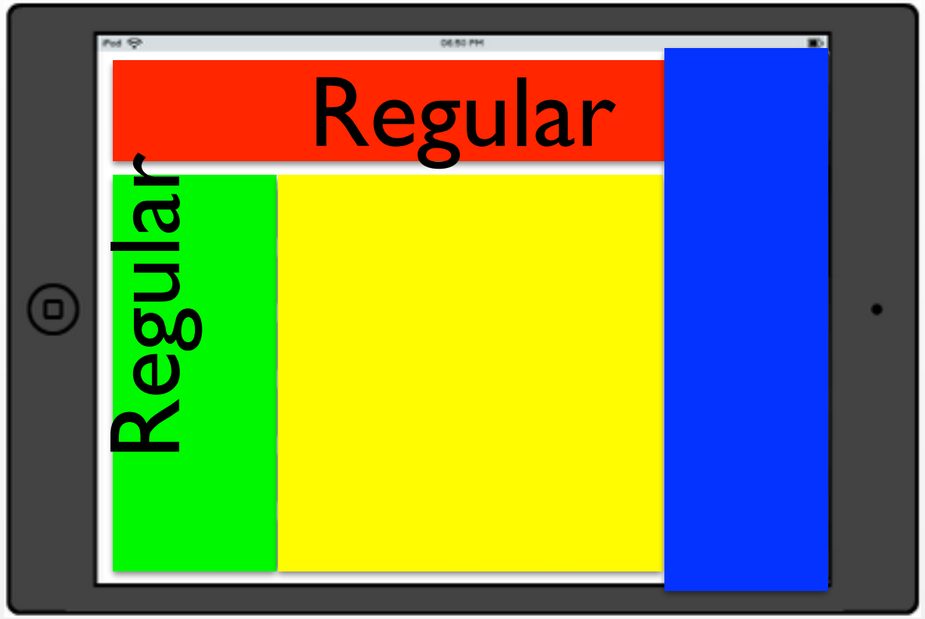
Code!

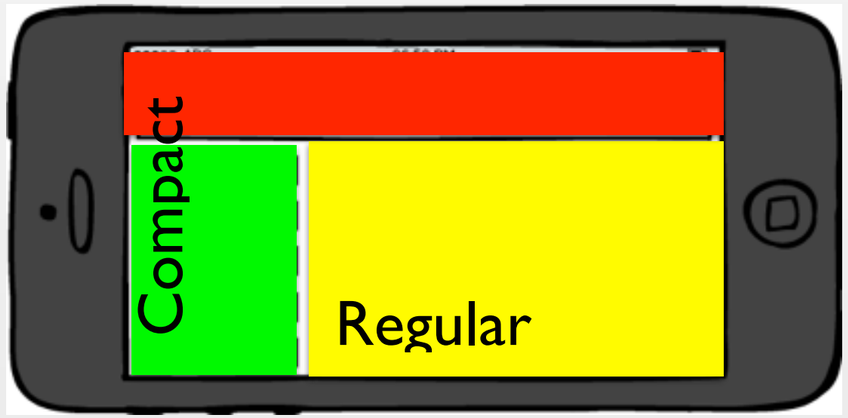
Now We Have Responsive App Design

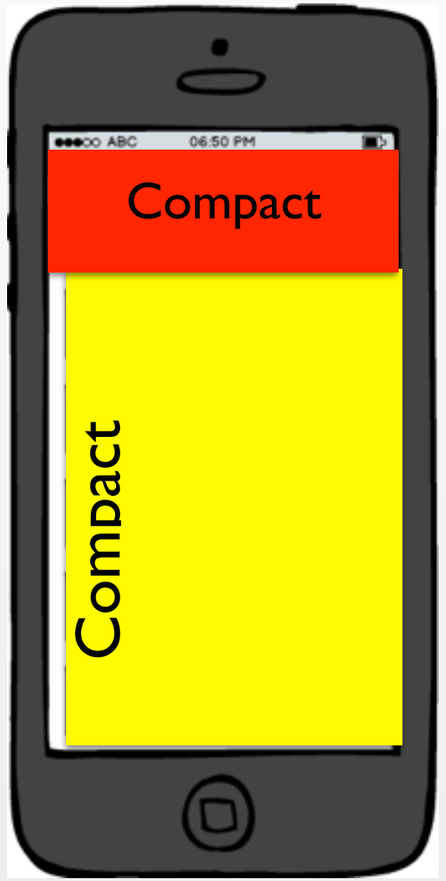
What's the best strategy for building an adaptive UI?

1 - Identify and simplify the layouts

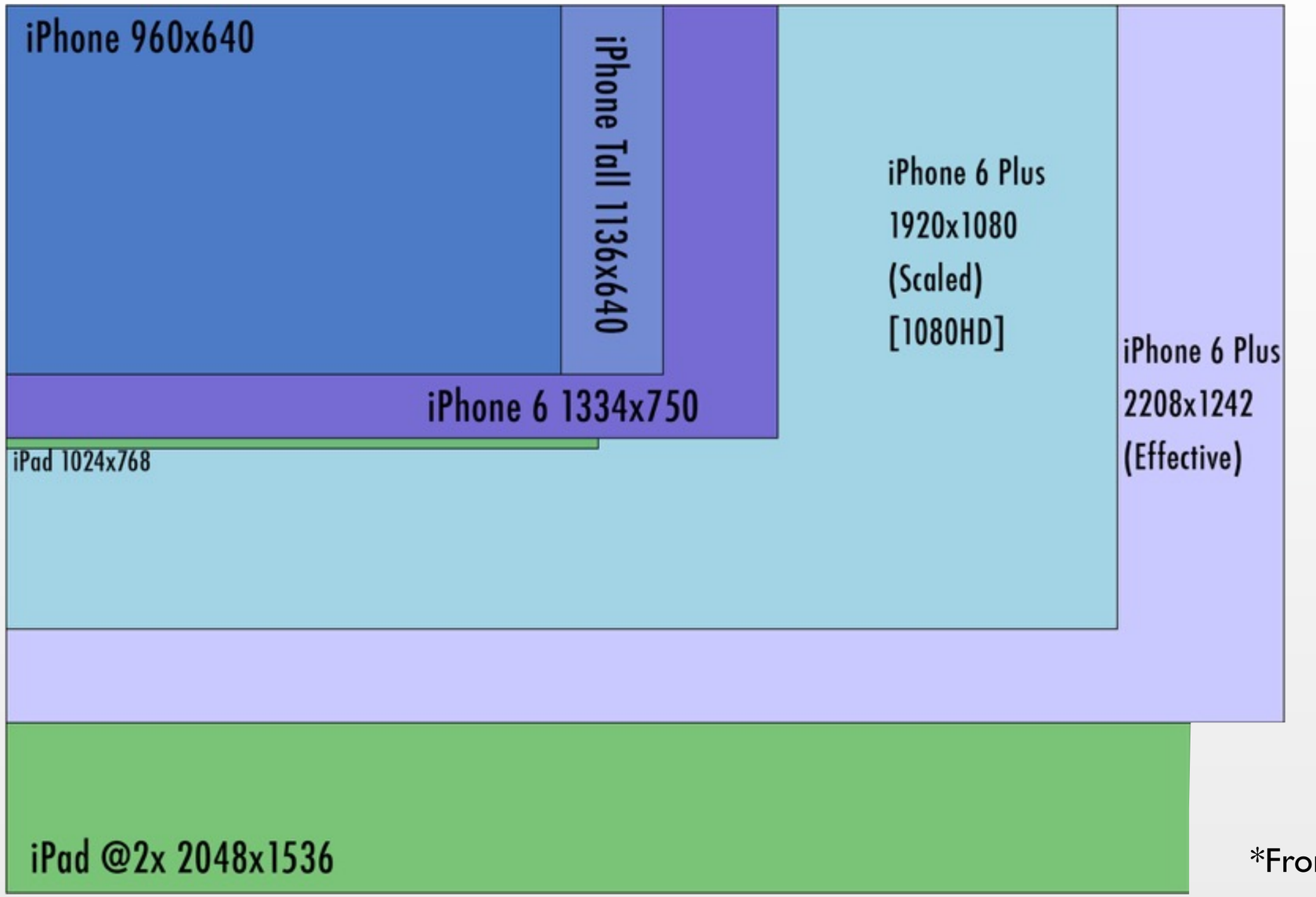




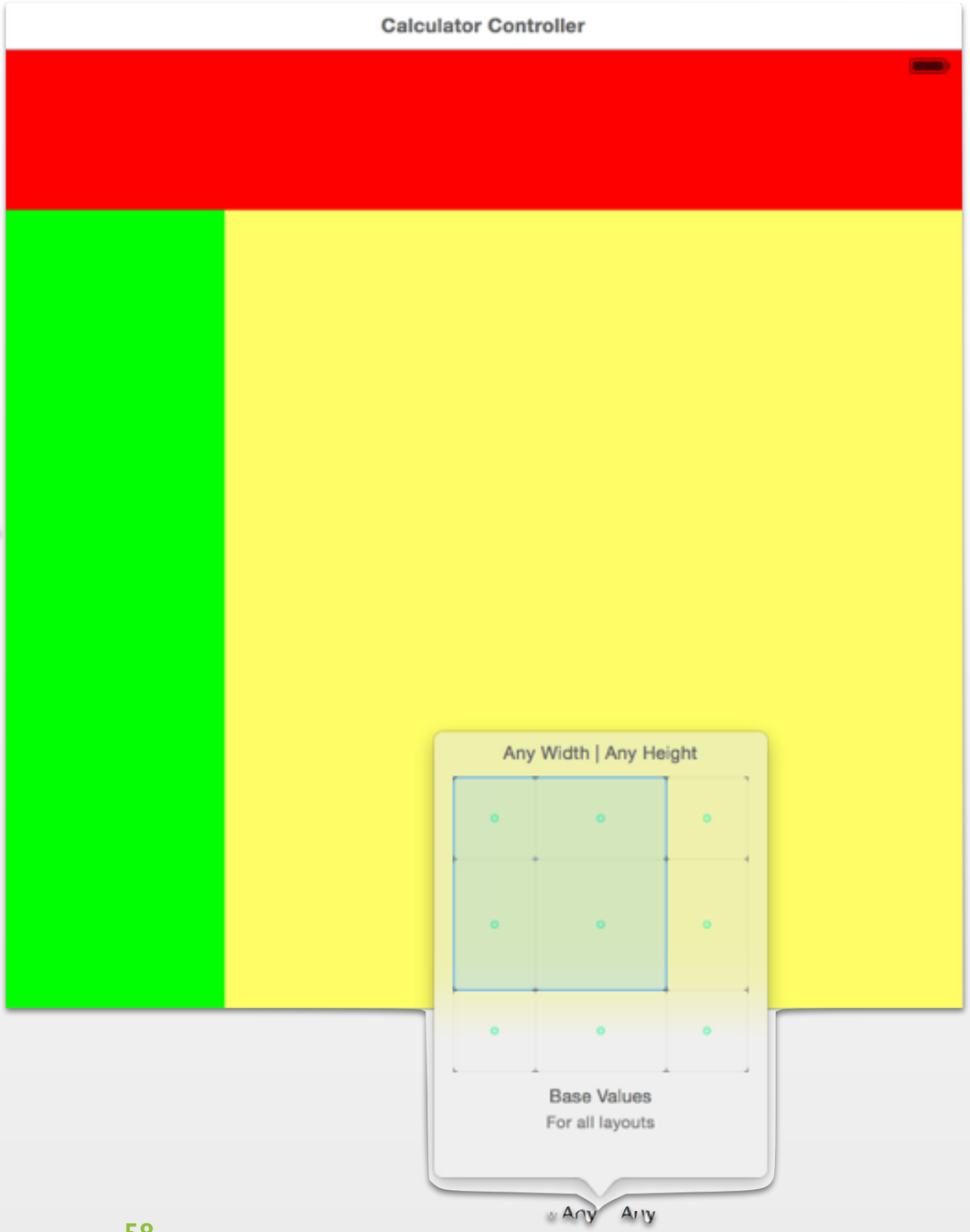




2 - Identify the breakpoints

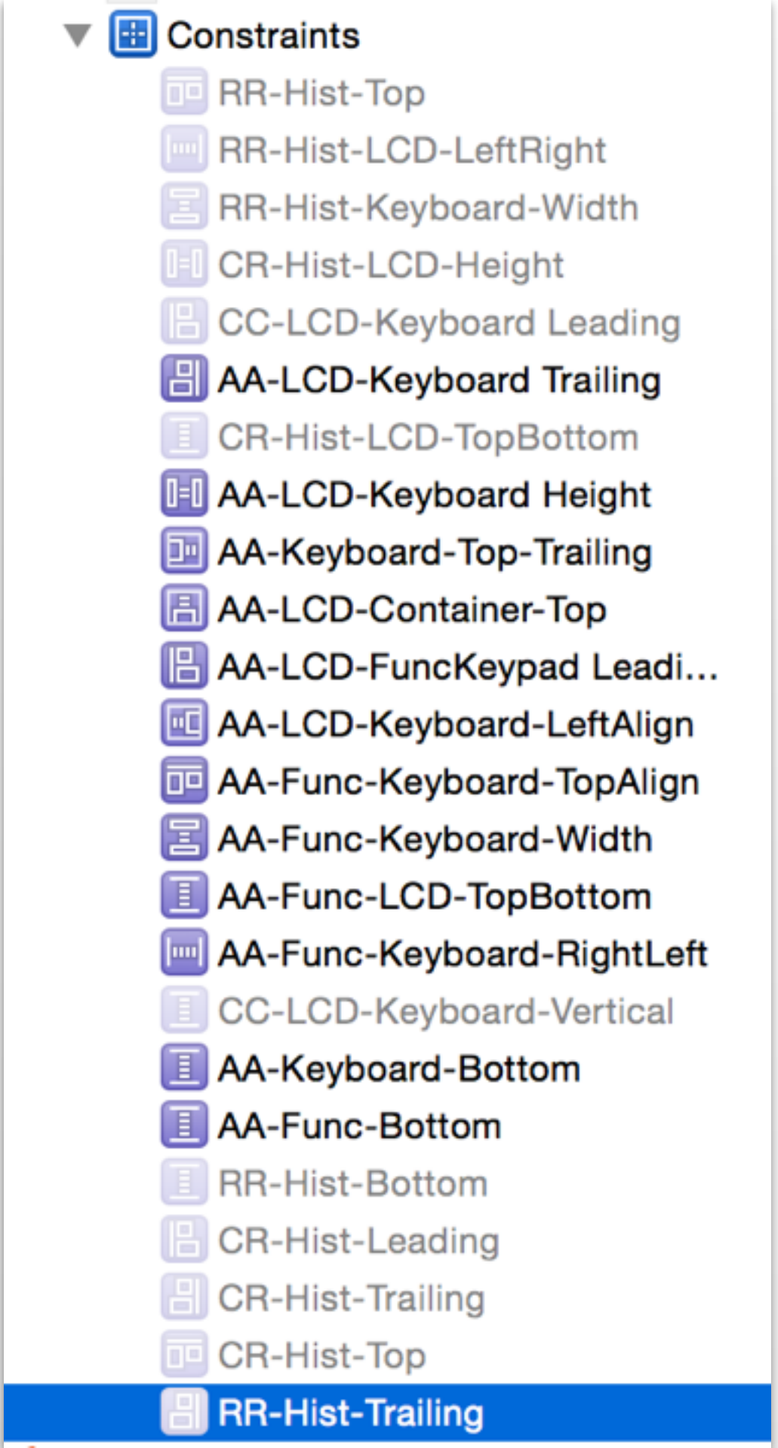


*From David Smith - @_DavidSmith



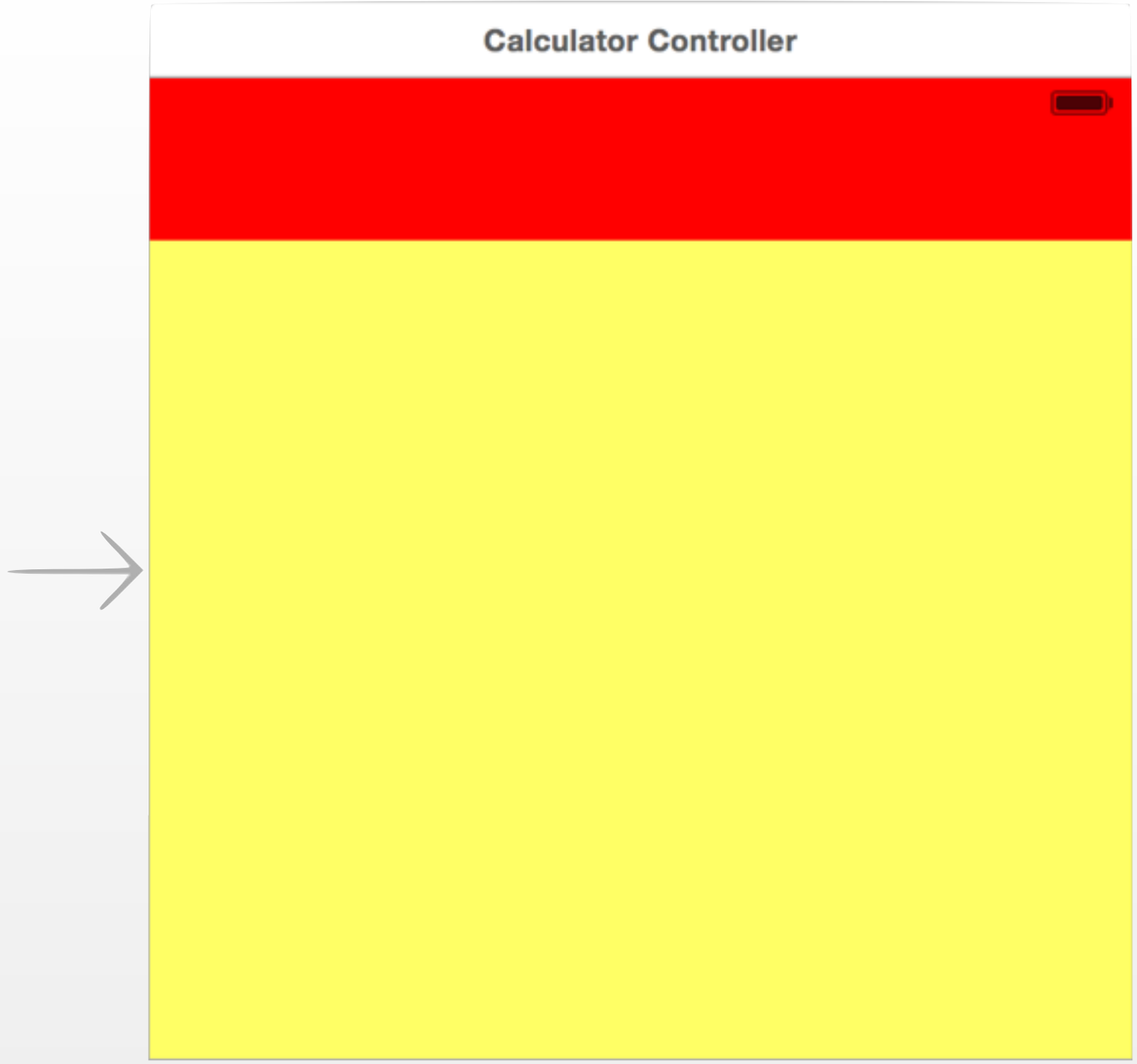
3 - Layout Any-Any size class

- Identify ‘Core’ object
- Add only any-any views
- Add constraints
- Commit!



4 - Name your constraints

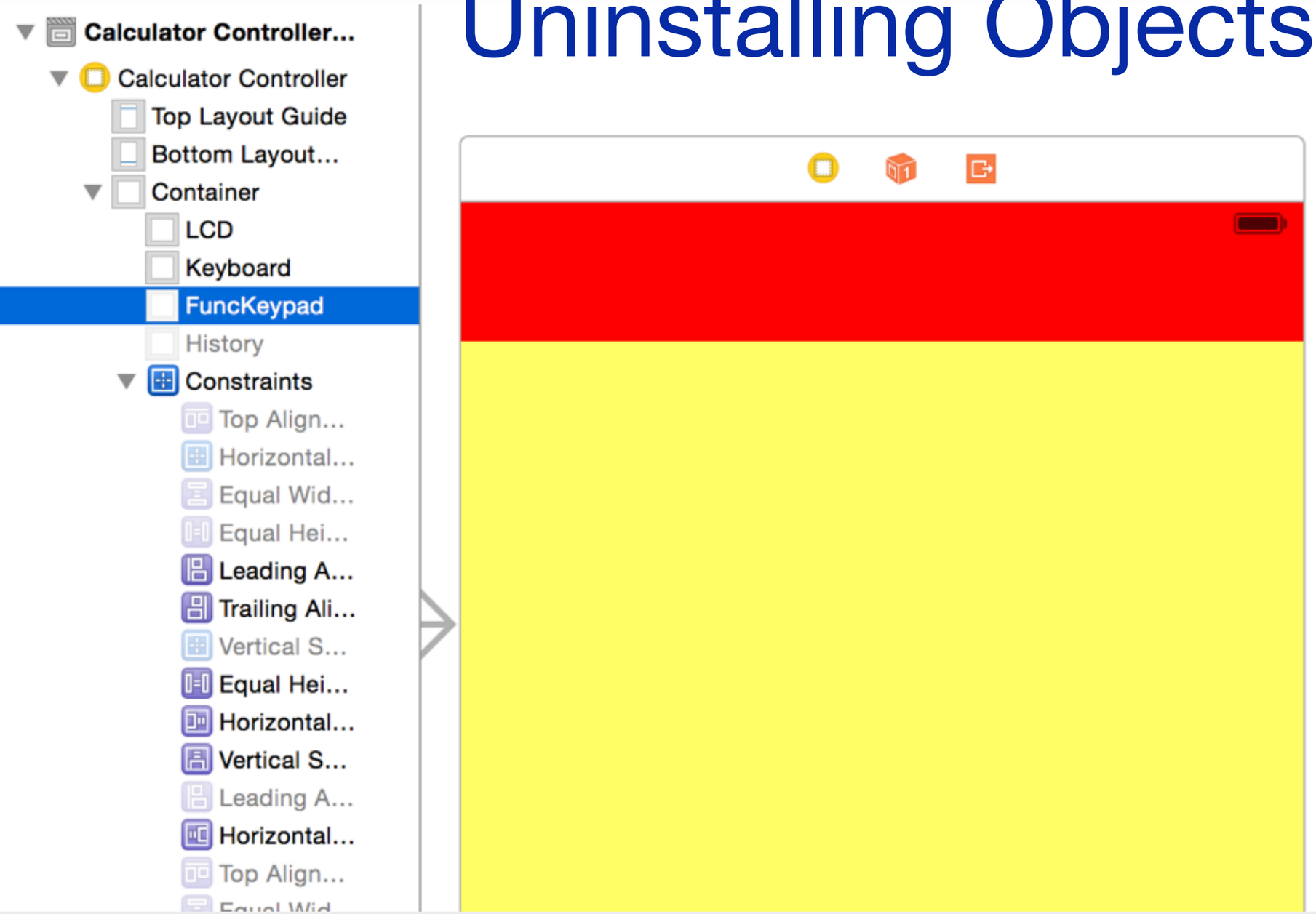
- Override the default constraint
- Helps identify the many constraints that will be created



5 - Layout other size class

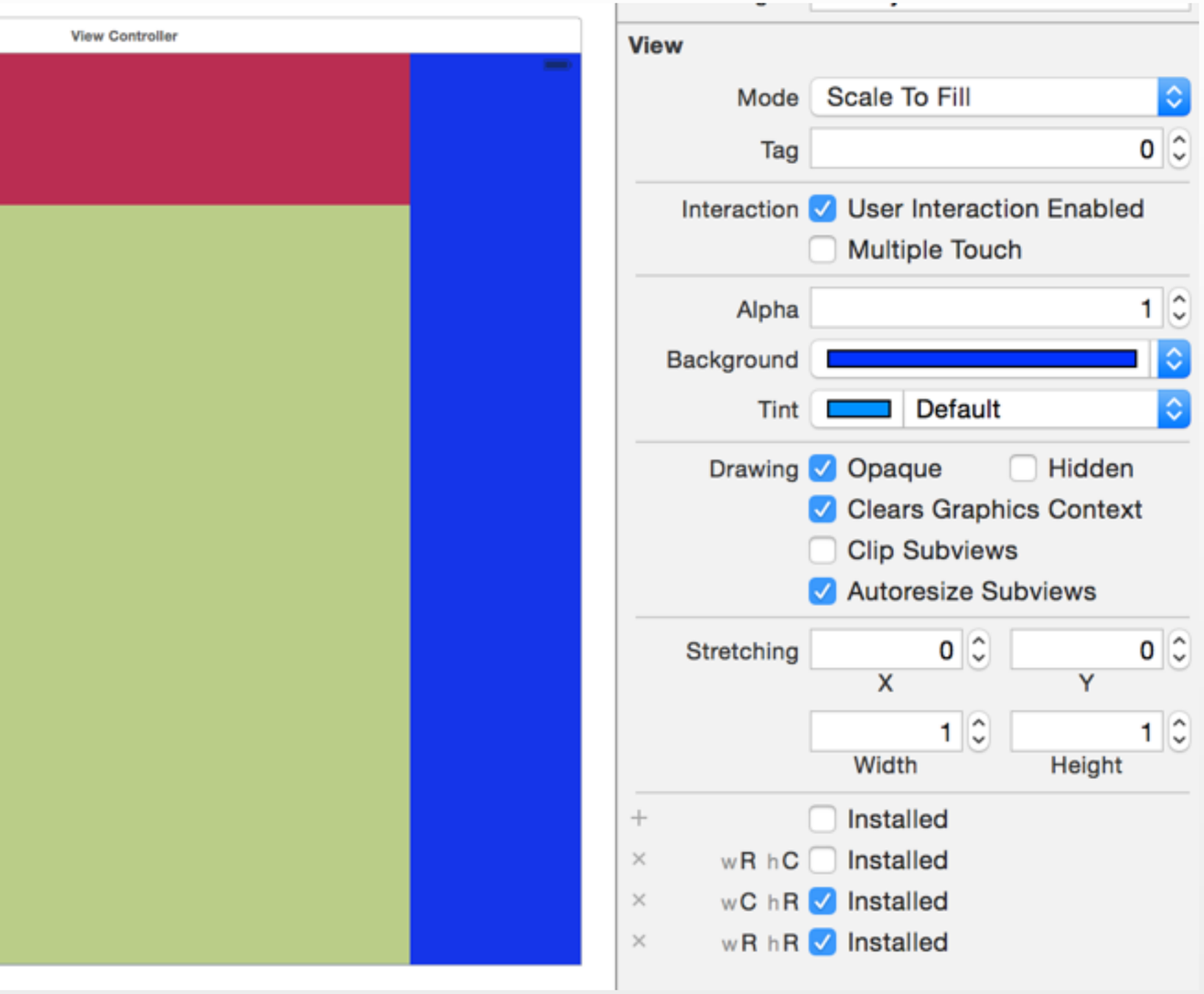
- ‘Uninstall’ objects
- Uninstall Constraints
- Add objects
- Add constraints
- Commit!

Uninstalling Objects



Demo

6 - Layout other size class



- Wash
- Rinse
- Commit!
- Repeat

Size-class Notes

- It Just Works™

- Encourages reuse

- Constraint ambiguity and conflicts limited to a size class

- Works in cells (TableView and CollectionView)
  - Some initial load janky-ness

- Must satisfy any-any

- Some Apple supplied containers don’t relay size class info

- Modals bypass your trait overrides

- UISplitViewController must be root.

Conclusion?

Questions?

Source:

github.com/jack-cox-captech/IBDesignables

github.com/jack-cox-captech/WinningWithIB

Contact Info

jcox@captechconsulting.com

@jcox_mobile