iOS Dev Accelerator
Week 2 Day 1

- UIAlertController
- UIImagePickerController
- Core Image
- Size Classes
- iPad

The MVC layout of our Week 2 App

Model Layer

- Post
- FilterService
- ParseService

Controller Layer

- Photo ViewController
- Gallery ViewController
- Timeline ViewController
- Tab Bar Controller
- AlertController

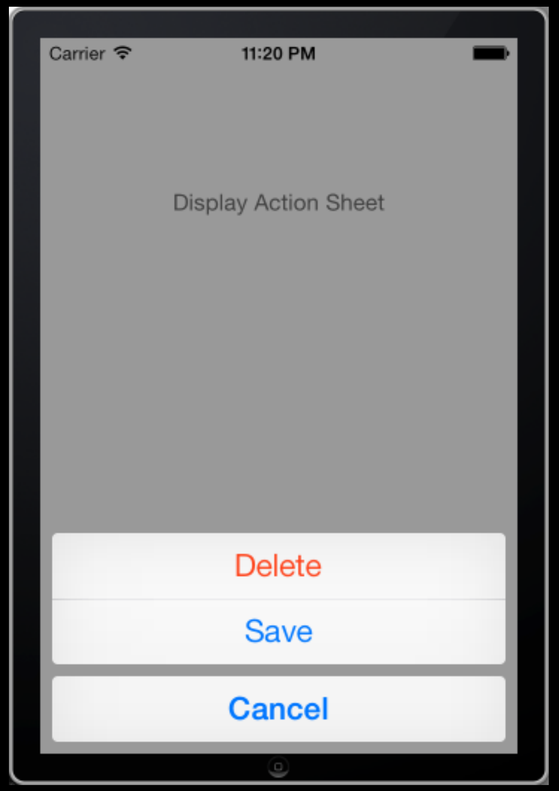
View Layer

- Storyboard
- FilterCell
- GalleryCell
- Buttons, Labels, Bars
- CollectionViews

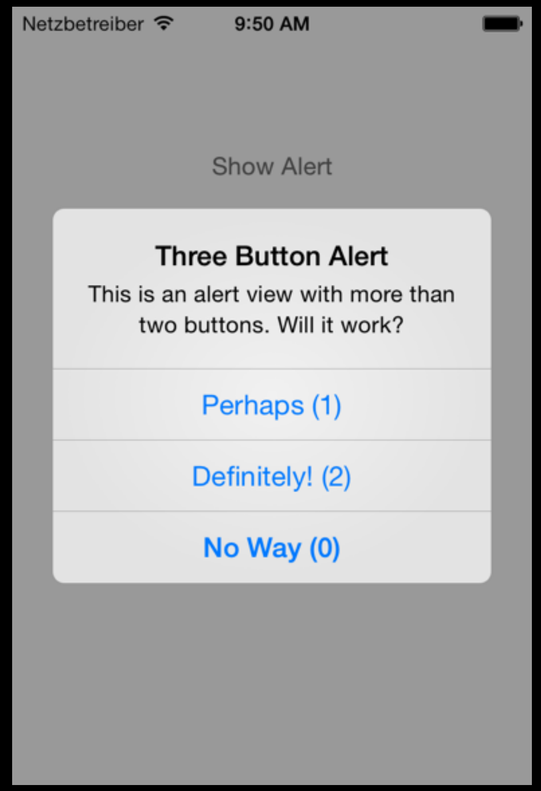
UIAlertController

UIAlertController

- “UIAlertController object displays an alert message to the user”
- Replaces both UIActionSheet and UIAlertView in iOS8
- After configuring the Alert Controller present it with presentViewController:animated: Completion:



ActionSheet



AlertView

UIAlertController Setup

`init(title:message:preferredStyle:)`

Creates and returns a view controller for displaying an alert to the user.

Declaration

SWIFT

```
convenience init(title title: String!,
                 message message: String!,
           preferredStyle preferredStyle: UIAlertControllerStyle)
```

Parameters

title	The title of the alert. Use this string to get the user's attention and communicate the reason for the alert.
message	Descriptive text that provides additional details about the reason for the alert.
preferredStyle	The style to use when presenting the alert controller. Use this parameter to configure the alert controller as an action sheet or as a modal alert.

UIAlertController Configuration

- In order to add buttons to your alert controller, you need to add actions.
- An action is a instance of the UIAlertAction class.
- “A UIAlertAction object represents an action that can be taken when tapping a button in an alert”
- Uses a closure expression (great!) to define the behavior of when the button is pressed. This is called the handler.

UIAlertAction Setup

`init(title:style:handler:)`

Create and return an action with the specified title and behavior.

Declaration

SWIFT

```
convenience init(title title: String!,
                 style style: UIAlertActionStyle,
               handler handler: ((UIAlertAction!) -> Void)!)
```

Parameters

title	The text to use for the button title. The value you specify should be localized for the user's current language. This parameter must not be `nil`.
style	Additional styling information to apply to the button. Use the style information to convey the type of action that is performed by the button. For a list of possible values, see the constants in UIAlertActionStyle.
handler	A block to execute when the user selects the action. This block has no return value and takes the selected action object as its only parameter.

Return Value

A new alert action object.

Feedback

Adding Actions

- Adding actions to the AlertController is as easy as calling addAction: on your AlertController and passing in the UIAlertAction(s)
- The order in which you add those actions determines their order in the resulting AlertController.

Presenting the alert controller

- To present the alert controller, you can call presentViewController:animated:completion: on the parent view controller
- This will work out of the box for iPhone, but on iPad it takes a bit more configuration
- On iPad you have to tell the alert controller where to present from, since its going to be a pop out menu.
- You can do this by setting the sourceView and sourceRect on the alert controller's popoverPresentationController.
- It's a great place to use optional binding because the popoverPresentationController will be nil on iPhone.

Demo

Camera Programming

- 2 ways for interfacing with the camera in your app:
  1. UIImagePickerController (easy mode)
  2. AVFoundation Framework (hard mode)

UIImagePickerController

- The workflow of using UIImagePickerController is 3 steps:
    1. Instantiate and modally present the UIImagePickerController
    2. ImagePicker manages the user's interaction with the camera or photo library
    3. The system invokes your image picker controller delegate methods to handle the user being done with the picker.

UIImagePickerController Setup

- The first thing you have to account for is checking if the device has a camera.
- If your app absolutely relies on a camera, add a UIRequiredDeviceCapabilities key in your info.plist
- Use the isSourceTypeAvailable class method on UIImagePickerController to check if camera is available.

UIImagePickerController Setup

- Next make sure something is setup to be the delegate of the picker. This is usually the view controller that is spawning the picker.
- The final step is to actually create the UIImagePicker with a sourceType of UIImagePickerControllerSourceTypeCamera.
- Media Types: Used to specify if the camera should be locked to photos, videos, or both.
- AllowsEditing property to set if the user is able to modify the photo in the picker after taking the photo.

UIImagePickerControllerDelegate

- The Delegate methods control what happens after the user is done using the picker. 2 big method:

  1. imagePickerControllerDidCancel:

  2. imagePickerController:didFinishPickingMediaWithInfo:

- In order to conform to the UIImagePickerControllerDelegate, you must also conform to the UINavigationControllerDelegate. Both have no required methods.

Info Dictionary

The info dictionary has a number of items related to the image that was taken:

```
NSString *const UIImagePickerControllerMediaType;
NSString *const UIImagePickerControllerOriginalImage;
NSString *const UIImagePickerControllerEditedImage;
NSString *const UIImagePickerControllerCropRect;
NSString *const UIImagePickerControllerMediaURL;
NSString *const UIImagePickerControllerReferenceURL;
NSString *const UIImagePickerControllerMediaMetadata;
```

MediaType is either kUTTypeImage or kUTTypeMovie

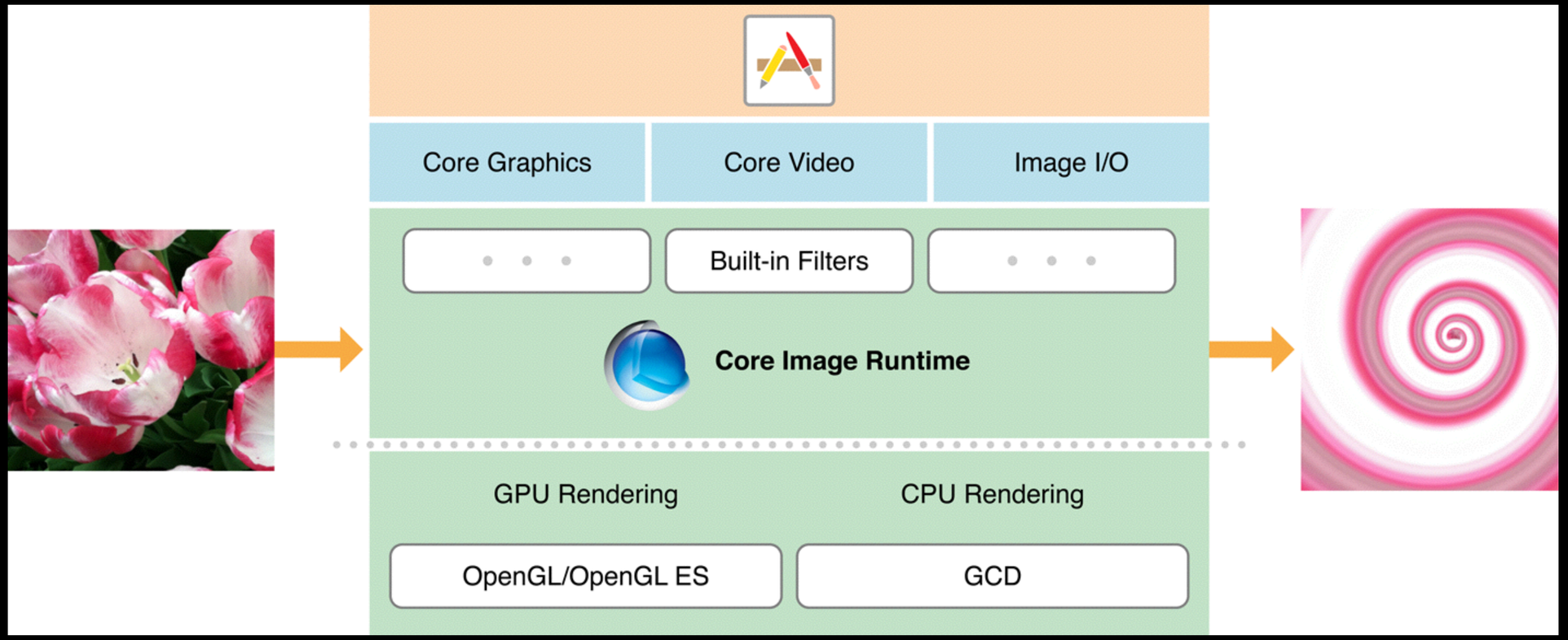
Demo

CoreImage

CoreImage

- “Core Image is an image processing and analysis technology designed to provide near real-time processing for still and video images”
- Can use either the GPU or CPU
- “Core Image hides the details of low-level graphic processing….You don’t need to know the details of OpenGL/ES to leverage the power of the GPU”

CoreImage


Core Graphics
Core Video
Image I/O
Built-in Filters
Core Image Runtime
GPU Rendering
CPU Rendering
OpenGL/OpenGL ES
GCD

CoreImage Offerings

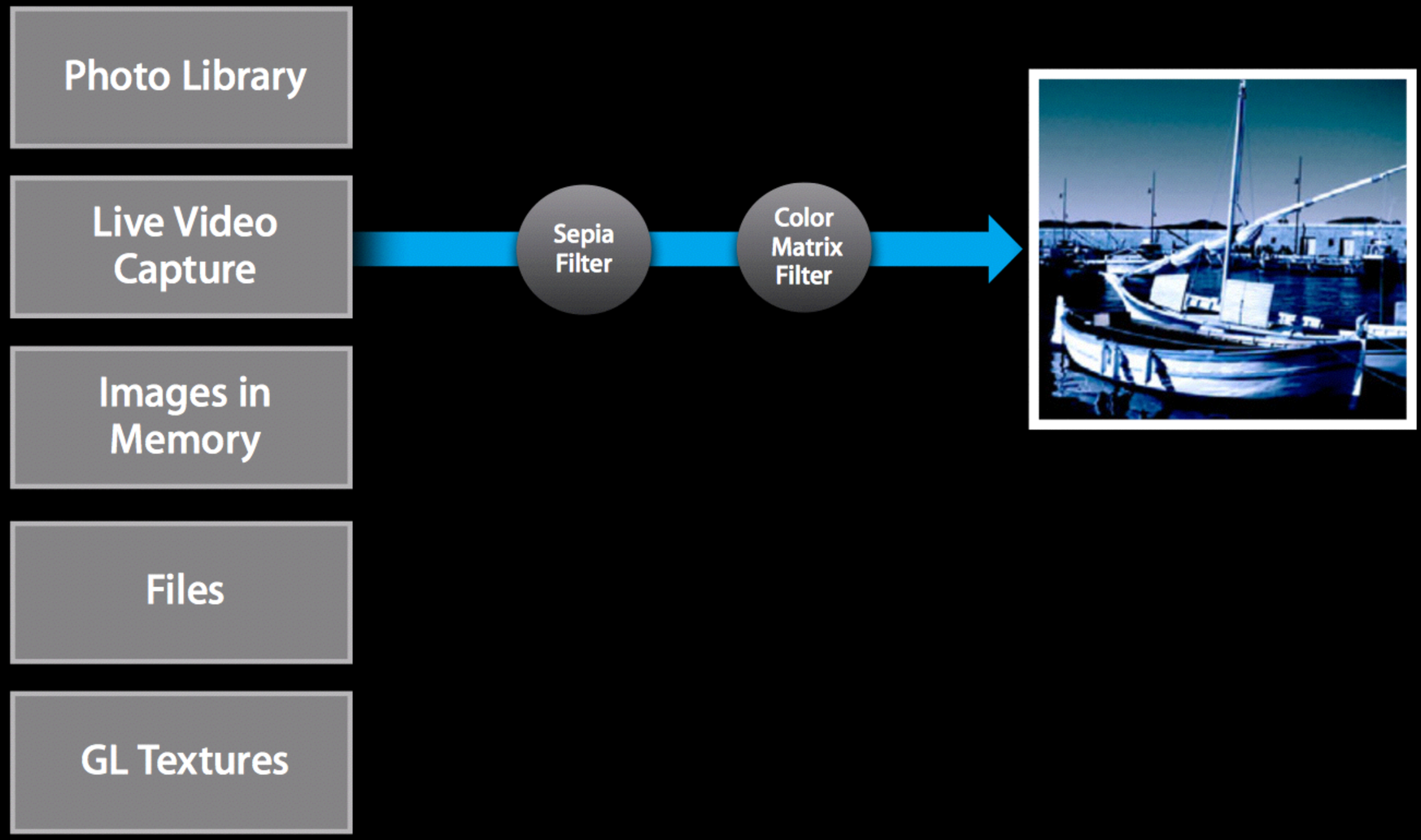
- Built-in image processing filters (90+ on iOS)
- Face and Feature detection capability
- Support for automatic image enhancement
- Ability to chain multiple filters together to create custom effects

guy using CoreImage

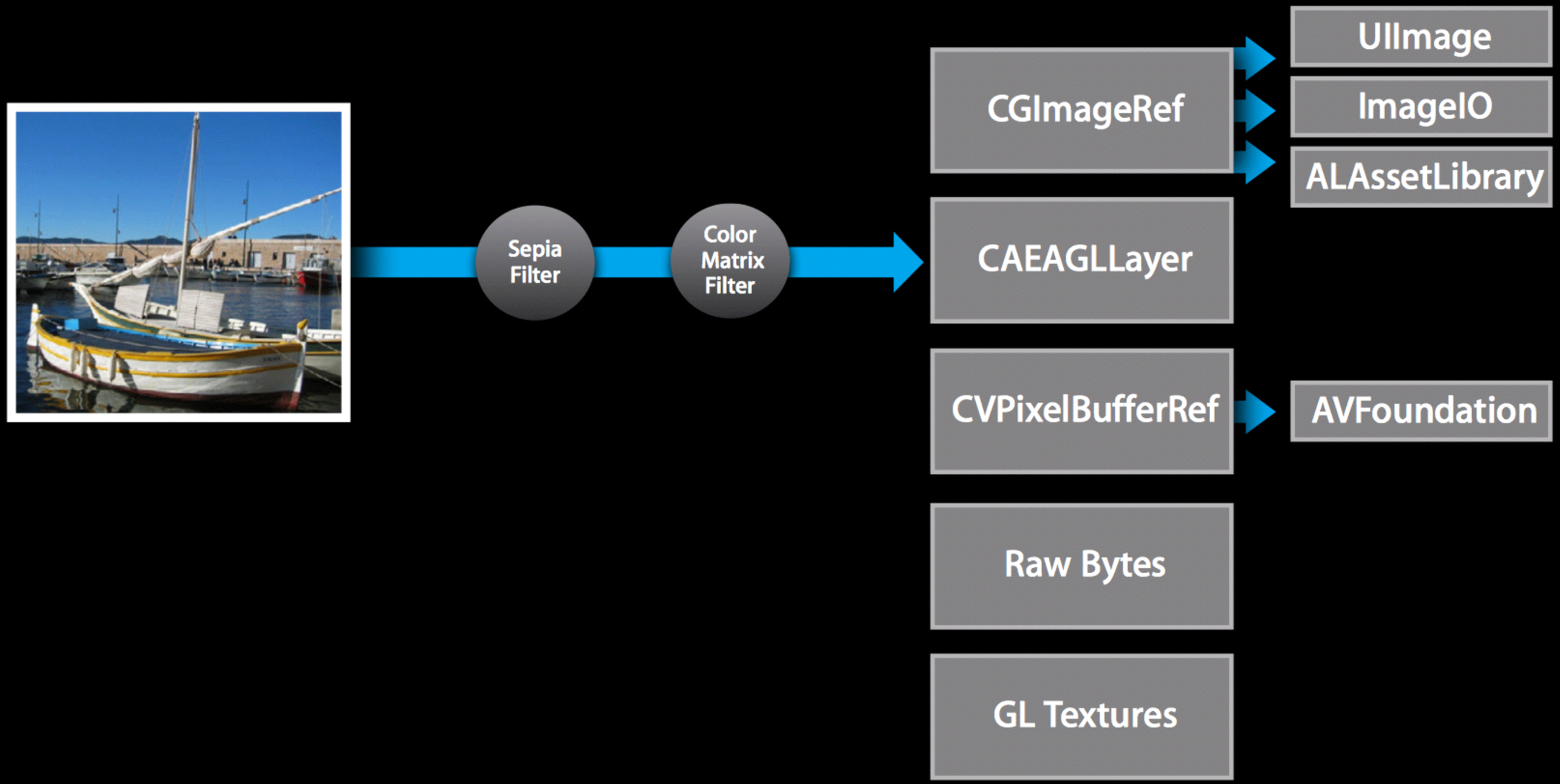
Filtering

- Filters applied on a per pixel basis
- Can be chained together

Filtering Inputs are Flexible



As are the Outputs



CIAdditionCompositing
CIAffineClamp
CIAffineTile
CIAffineTransform
CIBarsSwipeTransition
CIBlendWithMask
CIBloom
CICheckerboardGenerator
CICircleSplashDistortion
CICircularScreen
CIColorBlendMode
CIColorBurnBlendMode
CIColorControls
CIColorCube
CIColorDodgeBlendMode
CIColorInvert
CIColorMap
CIColorMatrix
CIColorMonochrome
CIColorPosterize
CIConstantColorGenerator
CICopyMachineTransition
CICrop
CIDarkenBlendMode
CIDifferenceBlendMode
CIDisintegrateWithMask
CIDissolveTransition
CIDotScreen
CIEightfoldReflectedTile
CIExclusionBlendMode
CIExposureAdjust
CIFalseColor
CIFlashTransition
CIFourfoldReflectedTile
CIFourfoldRotatedTile
CIFourfoldTranslatedTile
CIGammaAdjust
CIGaussianBlur
CIGaussianGradient
CIGlideReflectedTile
CIGloom
CIHardLightBlendMode
CIHatchedScreen
CIHighlightShadowAdjust
CIHoleDistortion
CIHueAdjust
CIHueBlendMode
CILanczosScaleTransform
CILightenBlendMode
CILightTunnel
CILinearGradient
CILineScreen
CILuminosityBlendMode
CIMaskToAlpha
CIMaximumComponent
CIMaximumCompositing
CIMinimumComponent
CIMinimumCompositing
CIModTransition
CIMultiplyBlendMode
CIMultiplyCompositing
CIOverlayBlendMode
CIPerspectiveTile
CIPerspectiveTransform
CIPinchDistortion
CIPixellate
CIRadialGradient
CIRandomGenerator
CISaturationBlendMode
CIScreenBlendMode
CISepiaTone
CISharpenLuminance
CISixfoldReflectedTile
CISixfoldRotatedTile
CISoftLightBlendMode
CISourceAtopCompositing
CISourceInCompositing
CISourceOutCompositing
CISourceOverCompositing
CIStarShineGenerator
CIStraightenFilter
CIStripesGenerator
CISwipeTransition
CITemperatureAndTint
CIToneCurve
CITriangleKaleidoscope
CITwelvefoldReflectedTile
CITwirlDistortion
CIUnsharpMask
CIVibrance
CIVignette
CIVortexDistortion
CIWhitePointAdjust

CIImage

- An Immutable object that represents the recipe for an Image
- Can represent a file from disk or the output of a CIFilter
- Multiple ways to create one:

```
var image  = CIImage(contentsOfURL: url)
```

```
var anotherImage = CIImage(image: UIImage())
```

Also has inits from Raw bytes,
NSData,CGImage,PixelBuffers,etc

CIFilter

- Mutable object that represents a filter (not thread safe since its mutable!)
- Produces an output image based on the input.
- Each filter has a different set of inputKey's you can modify to alter the effect of the filter:

```
var filter = CIFilter(name: "CISepiaTone")
filter.setValue(image, forKey: kCIInputImageKey)
filter.setValue(NSNumber(float: 0.8), forKey: @"inputIntensity")
```

- You can query for all the inputs of a filter with the .inputKeys property on an instance of CIFilter

CIContext

- An object through which Core Image draws results
- Can be based on CPU or GPU
- Always use GPU because the CPU performance sucks in comparison when dealing with graphical computations. All iOS 8 supporting devices support GPU context

```
self.context = CIContext(options: nil)
```
← CPU context

```
var options = [kCIContextWorkingColorSpace : NSNull()]
var myEAGLContext = EAGLContext(API: EAGLRenderingAPI.OpenGLES2)
self.gpuContext = CIContext(EAGLContext: myEAGLContext, options: options)
```
← GPU context

Demo

Size Classes

Size Classes

- “Size classes are traits assigned to a user interface element, like a screen or a view”
- There are three types of size classes, Regular, Compact, and Any.
- 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. Most often you only care about the view controllers trait collection.
- 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.

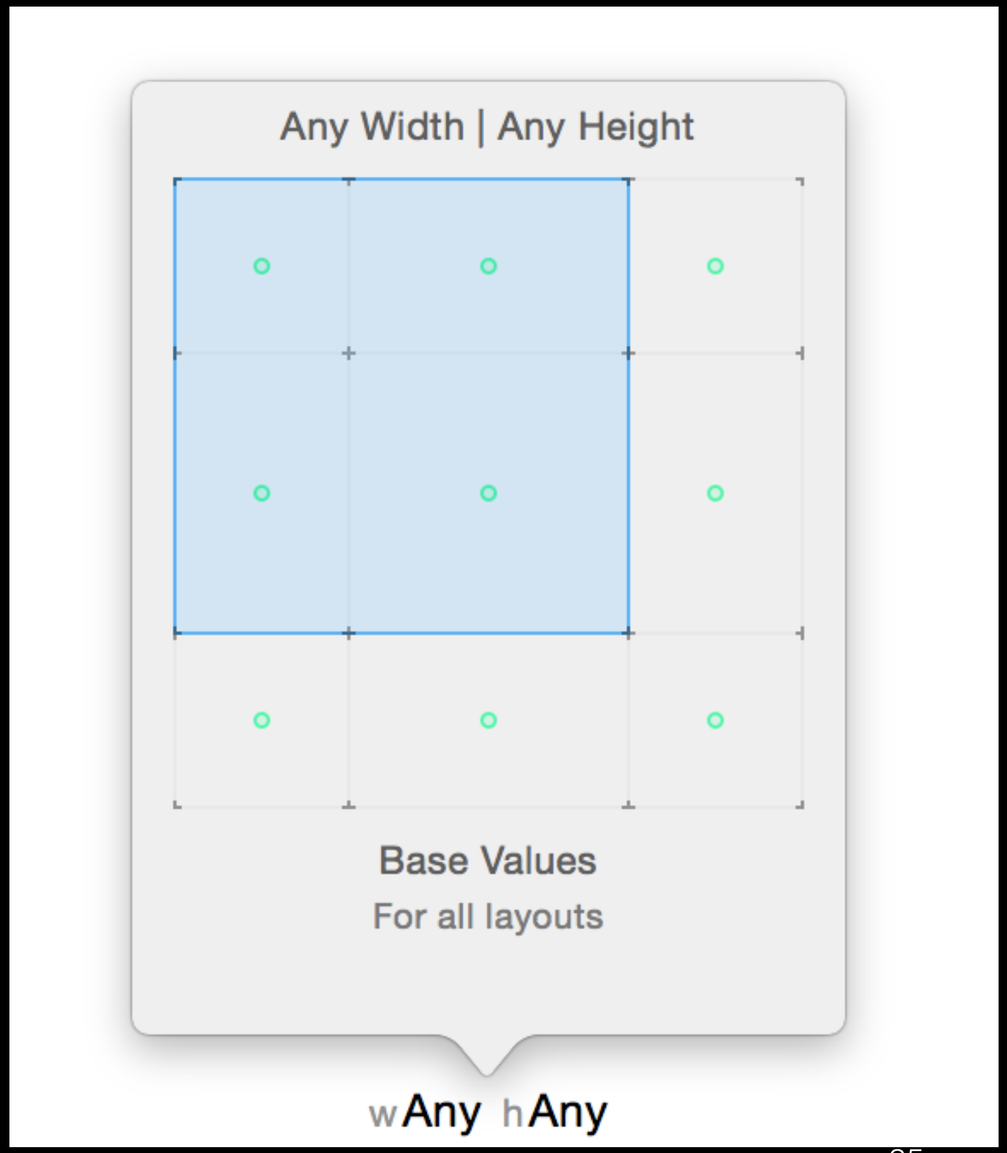
Size Classes

- 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 on and off
  4. turning view on and off

Demo

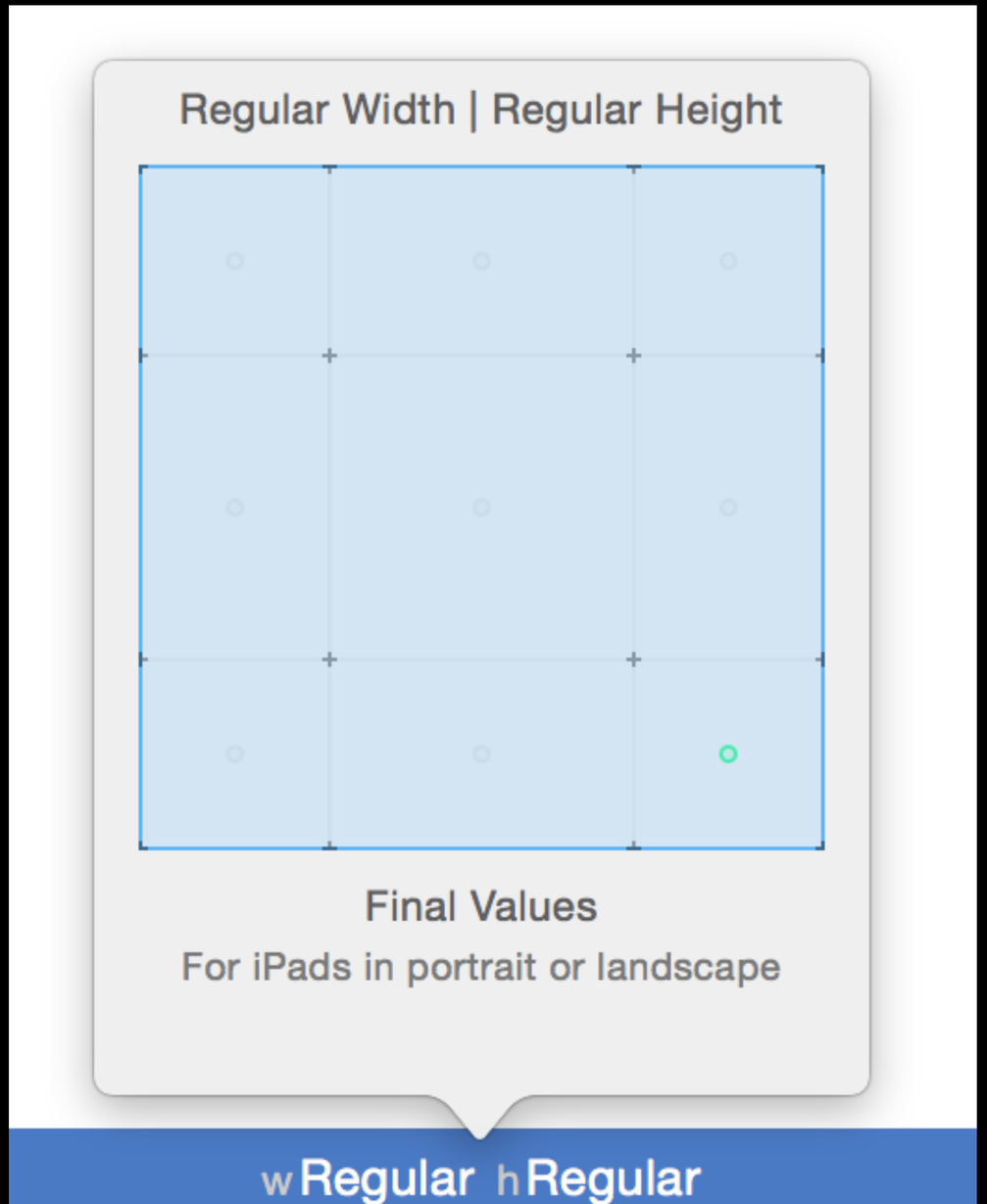
Size Class Grid



wAny hAny

- This is the ‘base’ size class
- Apple recommends you do your initial layout in this size class
- And then, if you need to specialize it for different devices, you can change the size class after laying it out here and make any changes/ additions

Size Class Grid

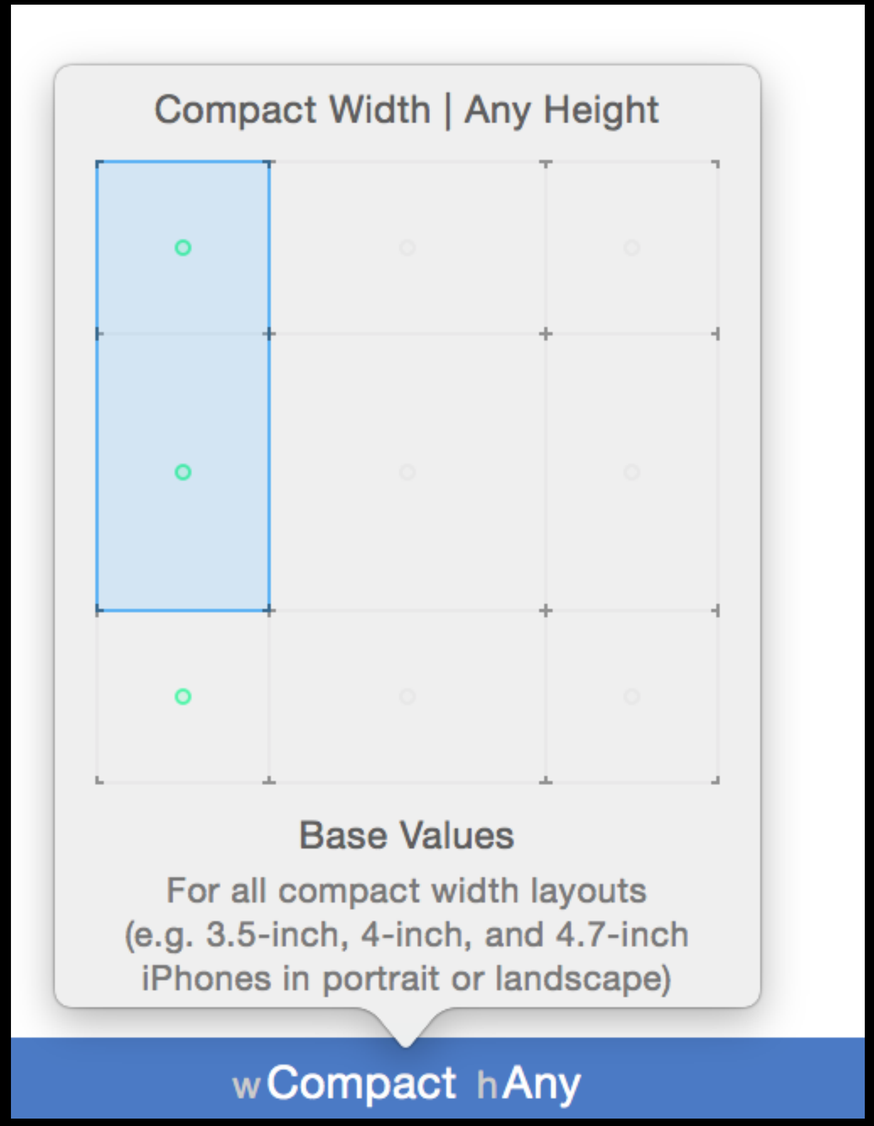


wRegula hRegular

- Regular width Regular height is for iPads in both landscape and portrait.
- Currently, there is no way to tell if the iPad is in landscape or portrait using size classes.

Size Class Grid

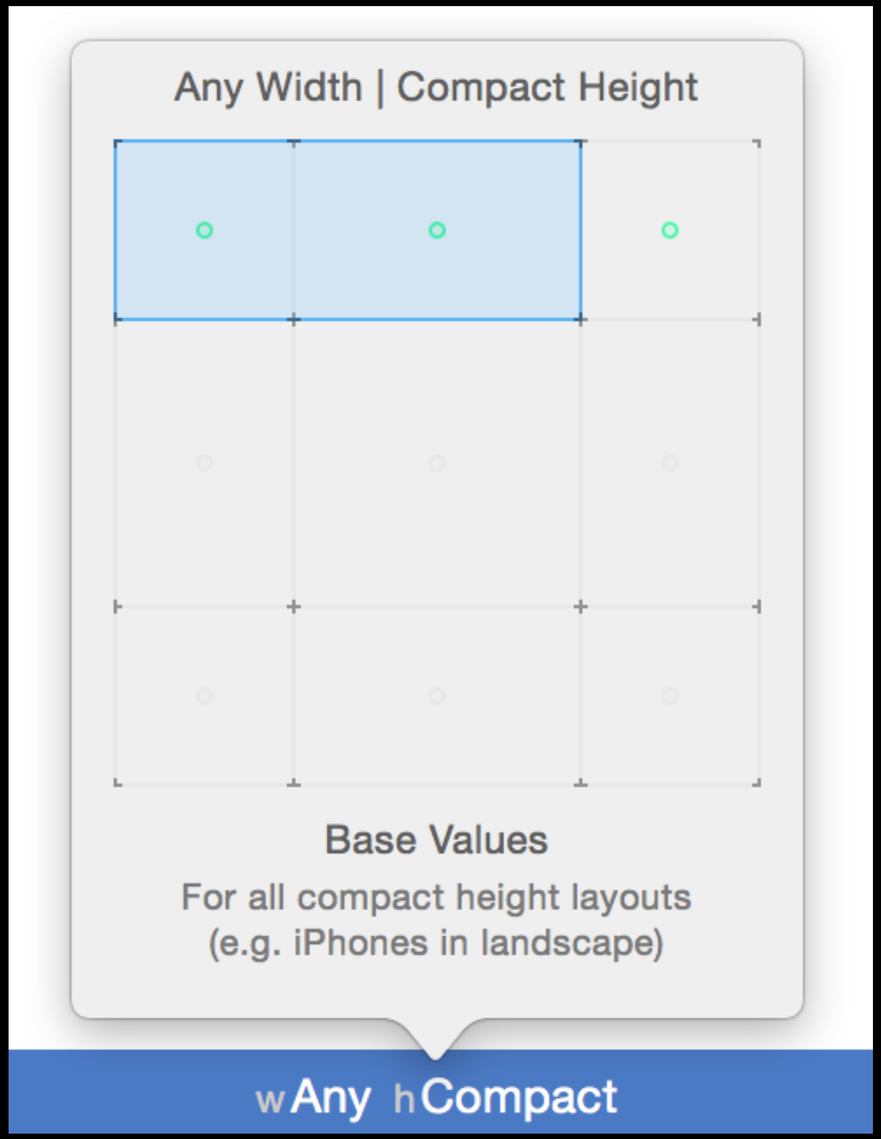
wCompact hAny



- This size class is for all iPhones except 6 Plus in landscape

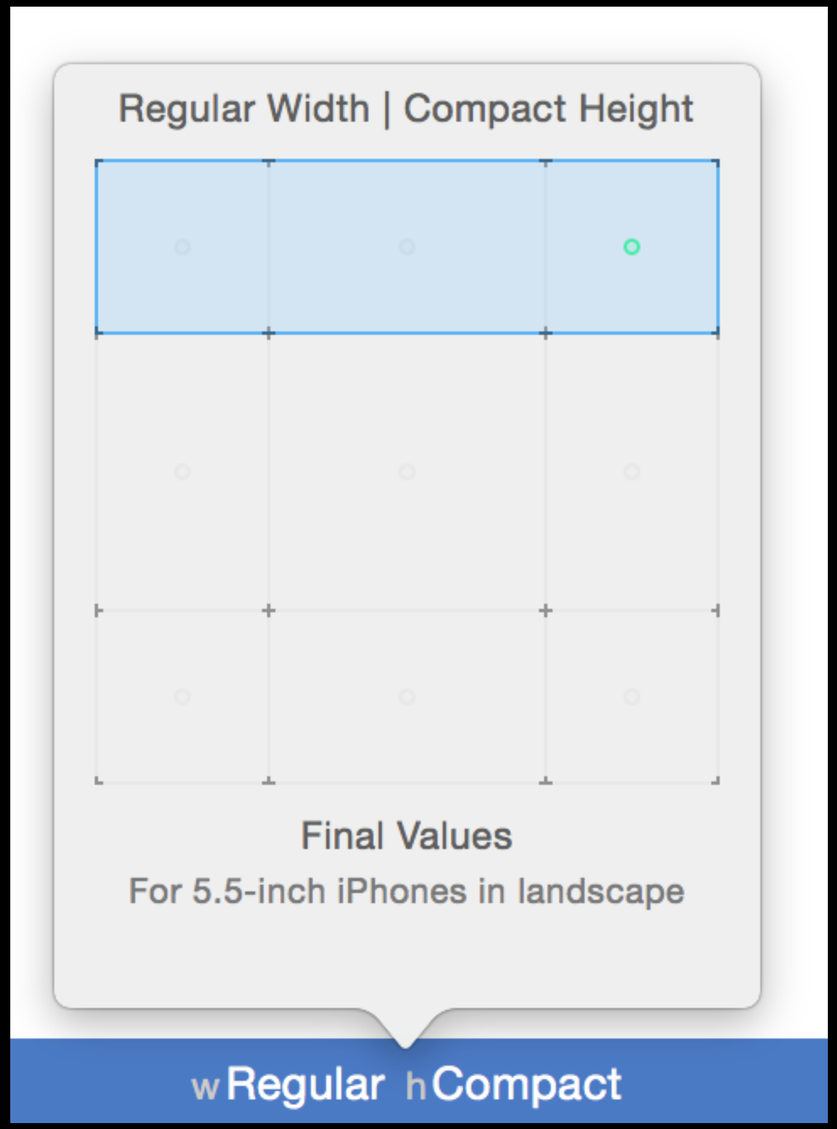
Size Class Grid

wAny hCompact



- This Size class is for all iPhones in landscape

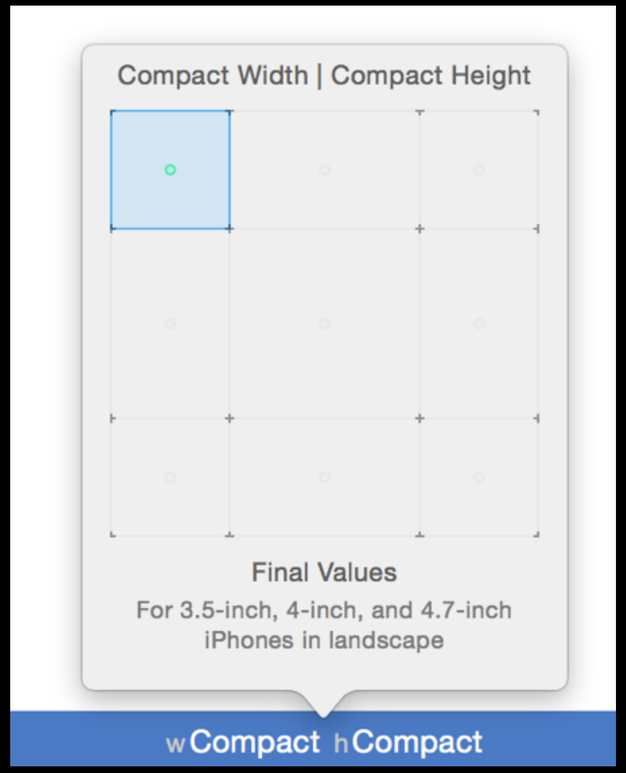
Size Class Grid



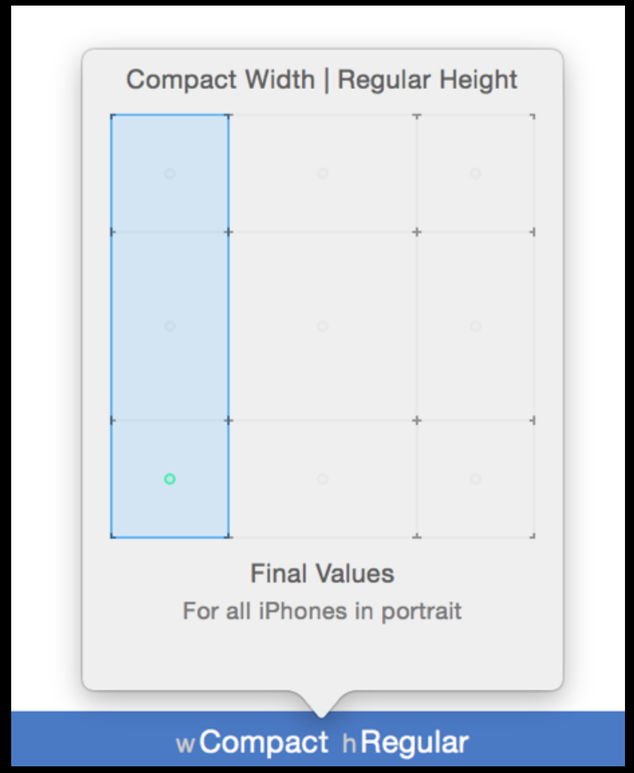
wRegular hCompact

- This Size class is for iPhone 6 Plus in landscape
- (Special guy gets its own size class. must be nice)

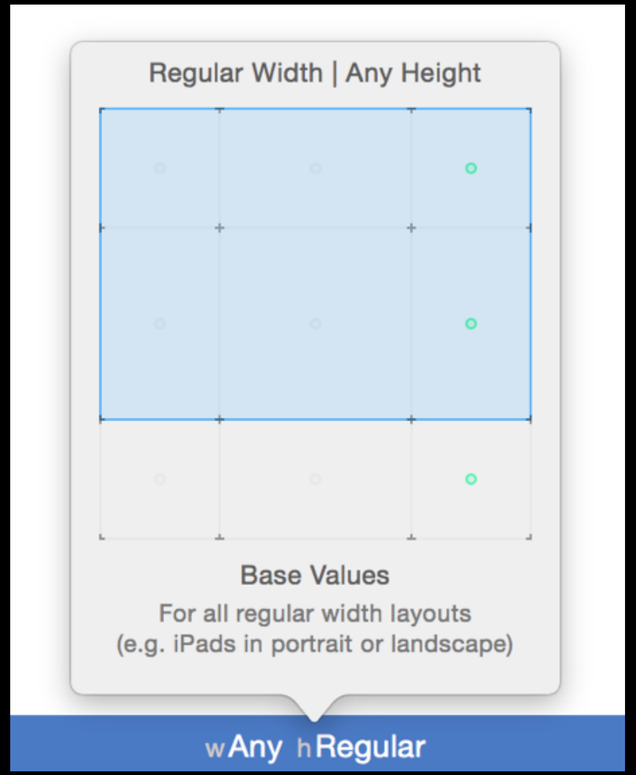
Size Class Grid



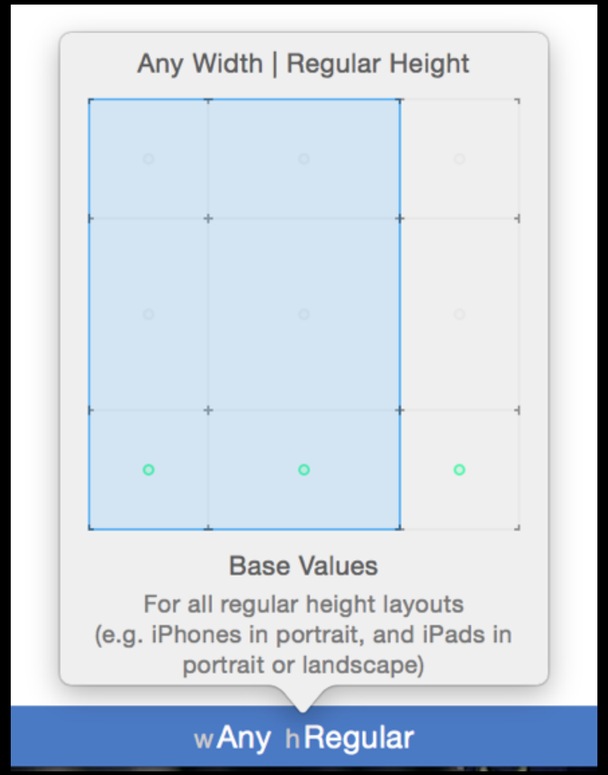
non 6 plus iphones in landscape



all iphones in portrait



ipad in portrait and landscape and 6 plus in landscape



all iphones in portrait
all ipads in portrait
and landscape

iPad

- Developing for iPad is 99% the same as developing for iPhone.
- Of course, you get a much larger view to work with.
- The only classes that works differently on iPad vs iPhone are:
  - **UISplitViewController** (shows a split interface, one view controller on the left —called the master, and one on the right — called the detail)
  - **UIAlertController** (shows a popup menu from the location of your choice — usually the button that triggered it)
  - **UIPopOverController** (for use exclusively on iPad, although iPhones AND iPads can now use the more modern UIPopOverPresentationController, which all view controllers can work with)

iPad in code

- Sometimes your code needs to take separate paths based on the device the user is on
- So how do you know if your universal app is running on the user's iPhone or iPad?
- A couple ways:
  - One of the properties of a UITraitCollection is called userInterfaceIdiom. Its type is UIUserInterfaceIdiom, which is an enum with 3 options: Unspecified, Phone, **Pad**
  - There is a class called UIDevice, which gives you a singleton that represents the current device. That singleton has a property that is also type userInterfaceIdiom.
- I always use the 2nd way.

Demo