

iOS Dev Accelerator

Week2 Day4

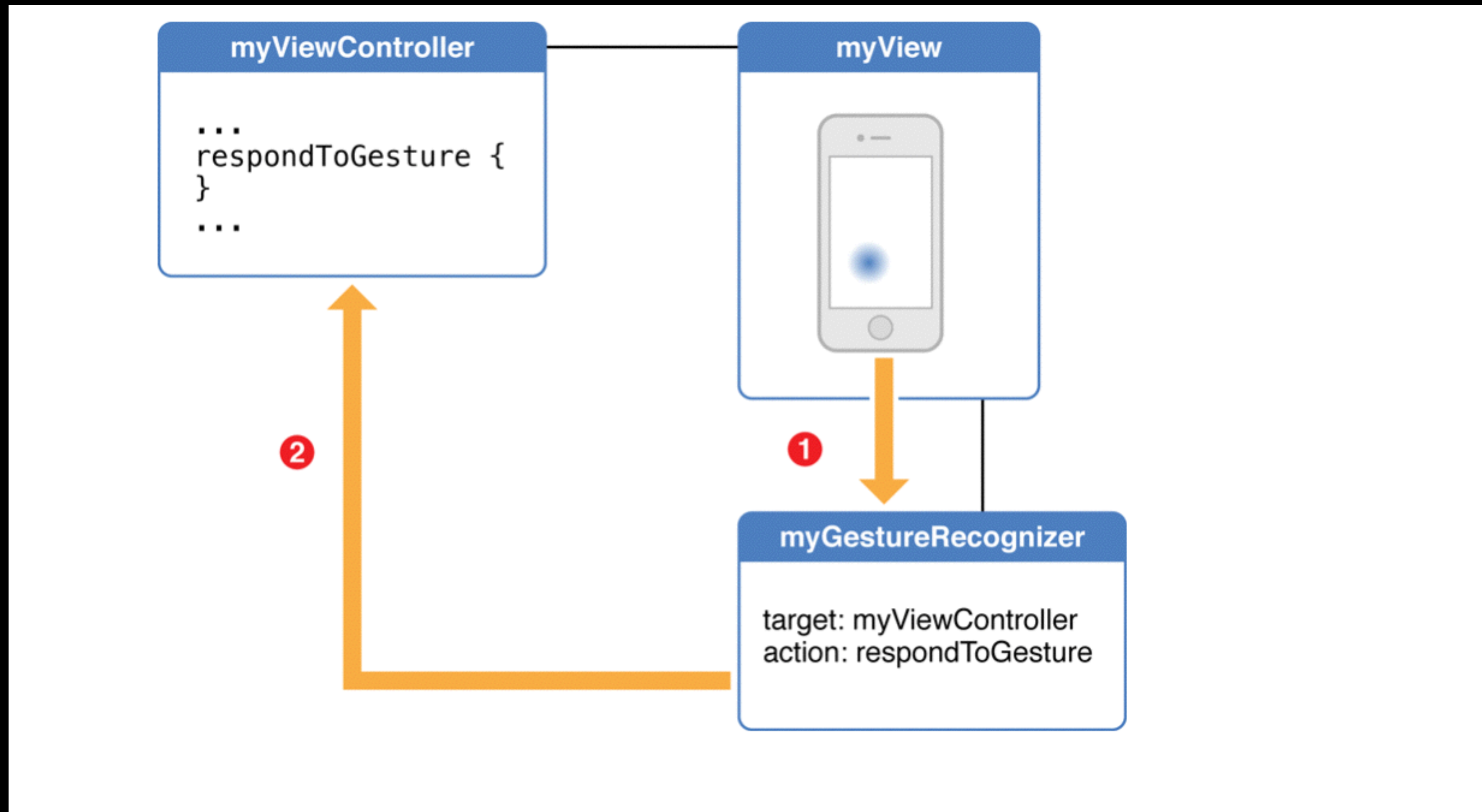
- Gesture Recognizers
- CollectionViewFlowLayout
- Localization
- Accessibility
- Technical Thursday: Queues

Gesture Recognizers

Gesture Recognizers

- “Gesture Recognizers convert low level event handling code into higher level actions”
- Gesture recognizers are attached to views.
- If the desired gesture is detected on the view the recognizer is attached to, an action message is sent to a target object.
- The target is usually the view’s view controller.

Gesture Recognizers



Predefined vs Custom Gesture Recognizers

- UIKit has a good amount of predefined gesture recognizers that you should always use when possible.
- It is much more simple to use one of their recognizers vs implementing your own.
- If your app needs to recognize a custom gesture, like a figure 8 or checkmark, you will need to implement your own custom gesture recognizer.

Built-in Gesture Recognizers

- UITapGestureRecognizer - any number of taps
- UIPinchGestureRecognizer - pinch in and out for zooming
- UIPanGestureRecognizer - panning or dragging
- UISwipeGestureRecognizer - swiping in any direction
- UIRotationGestureRecognizer - finger moving in opposite direction
- UILongPressGestureRecognizer - touch and hold for a certain amount of time
- Refer to the HIG for recommended usage for each type of gesture

Discrete vs Continuous Gestures

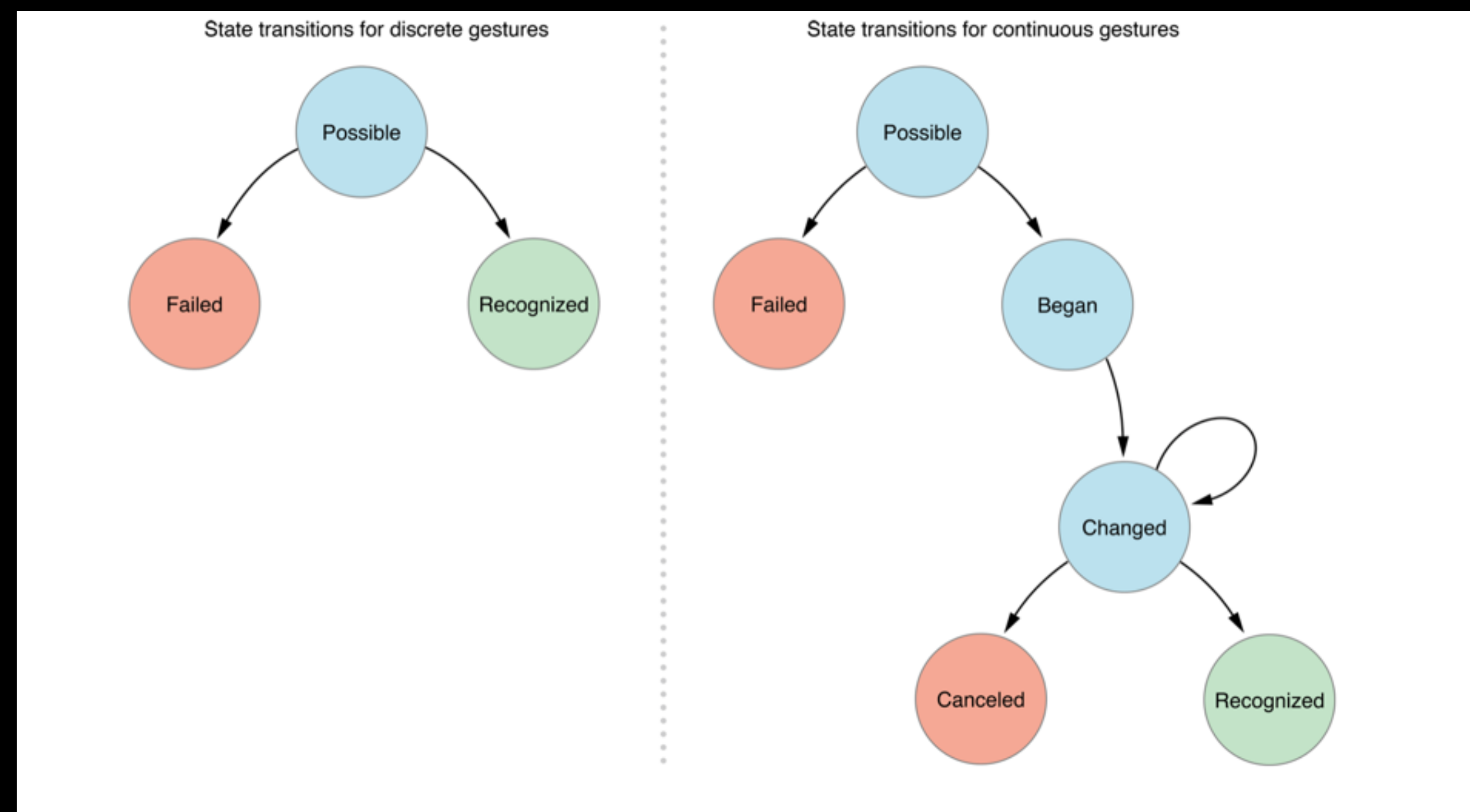
- Gestures are either discrete or continuous.
- A discrete gesture only happens once it is detected. Like a tap.
- A continuous gesture takes place over time, like a pan.
- If it is discrete, only one action message is sent. If it is continuous, many action messages are sent until the gesture is over.

Gesture Recognizer Setup

1. Create and configure a gesture recognizer instance. Either in code or in storyboard. If its storyboard, this includes step 2.
2. Attach the gesture recognizer to a view.
3. Implement the action method that handles the gesture.

Gesture Recognizer State

- Gesture Recognizers transition from one state to another in a predefined way.
- From each state, they can move to one of several possible next states based on whether they meet certain conditions:



Demo

CollectionView
FlowLayout

UICollectionViewLayout

- Computes layout attributes as needed for:
- CollectionView Cells
- CollectionView Supplementary Views
- Decoration Views

UICollectionViewLayout

- Every collection view uses a layout object to determine where each view it manages should be placed and behave on screen.
- Apple provides a concrete subclass of UICollectionViewLayout called UICollectionViewFlowLayout that gives us a line based layout that we can use right out of the box.
- A collection view's layout is highly customizable. When you want to create a custom layout, you first need to determine if it is suitable for you to subclass flow layout (less work), or create a brand new subclass of UICollectionViewLayout (more work).

UICollectionViewLayoutAttributes

- Manages the following layout-related attributes for a given item in a collection view:
 - Position
 - Size
 - Opacity
 - zIndex (overlapping cells, above or below)
 - Transforms
- One attribute instance per view!

UICollectionViewFlowLayout

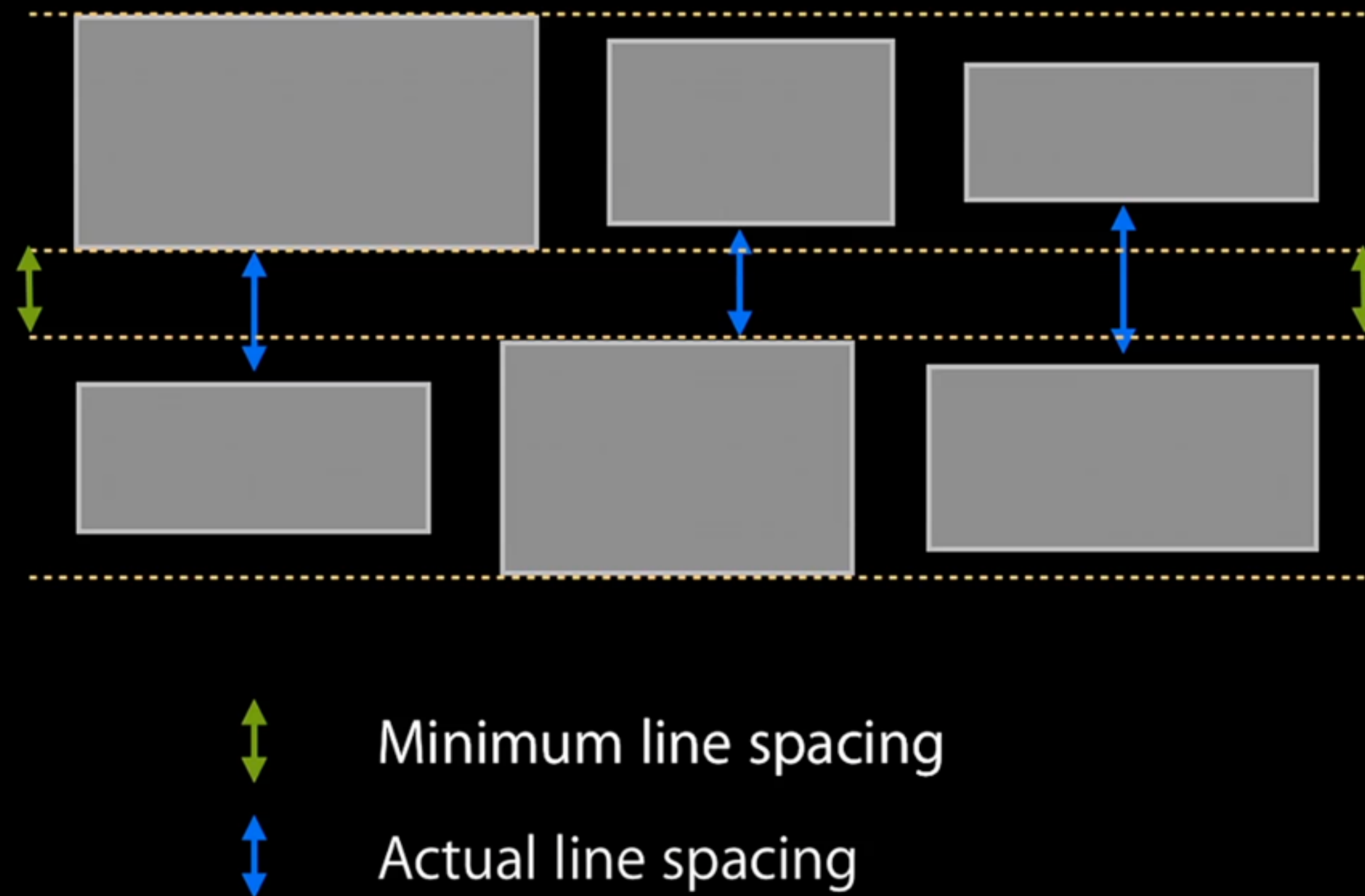
- Flow layout is a line-oriented layout. The layout object places cells on a linear path and fits as many cells as it can along the line. When the line runs out of room, it creates a new line and continues the process.
- Can be configured as a grid or as a group of lines.
- Out of the box, it has lots of things you can customize:
 - Item Size
 - Line Spacing and Inter Cell spacing
 - Scrolling direction
 - Header and footer size
 - Section Inset
- And you customize each of those things either globally with a single property, or through a delegate

Item Size

- The item size for each cell can be set globally by setting the `itemSize` property on your flow layout.
- Or if you want different size per item, you can do it through the delegate method `collectionView:layout:sizeForItemAtIndexPath()`

Line Spacing

- You can set a minimum line spacing, either globally or through the delegate:



Inter-item Spacing

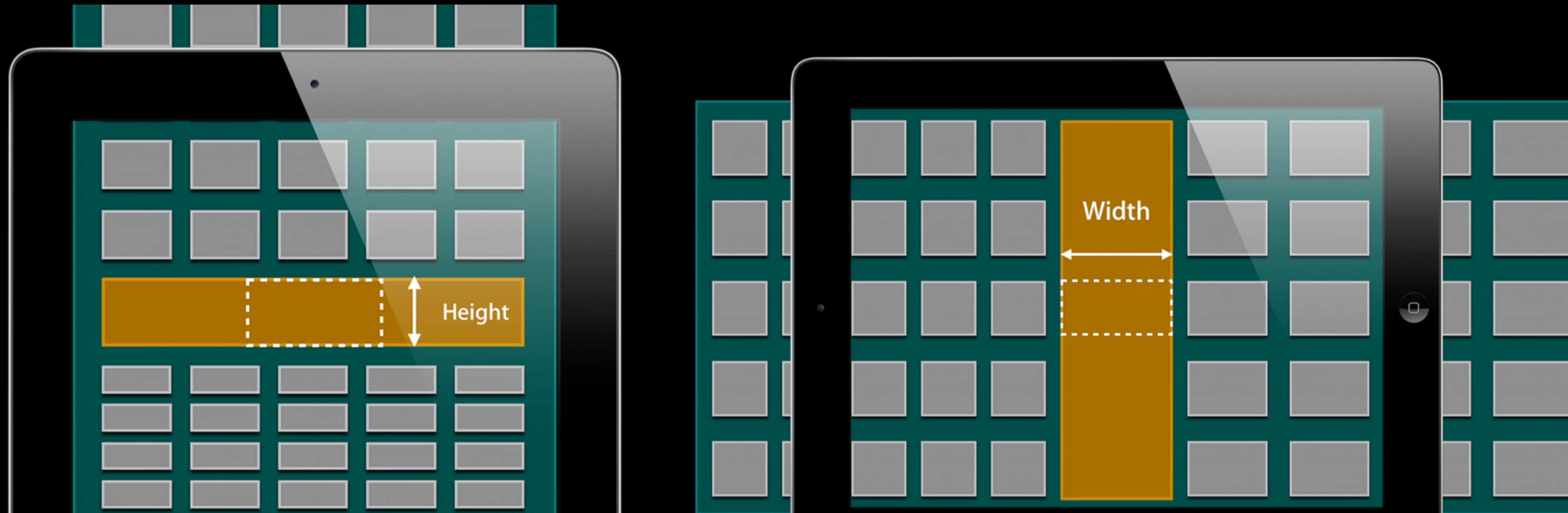
- Same with spacing between individual items:



↔ Actual interitem spacing
↔ Minimum interitem spacing

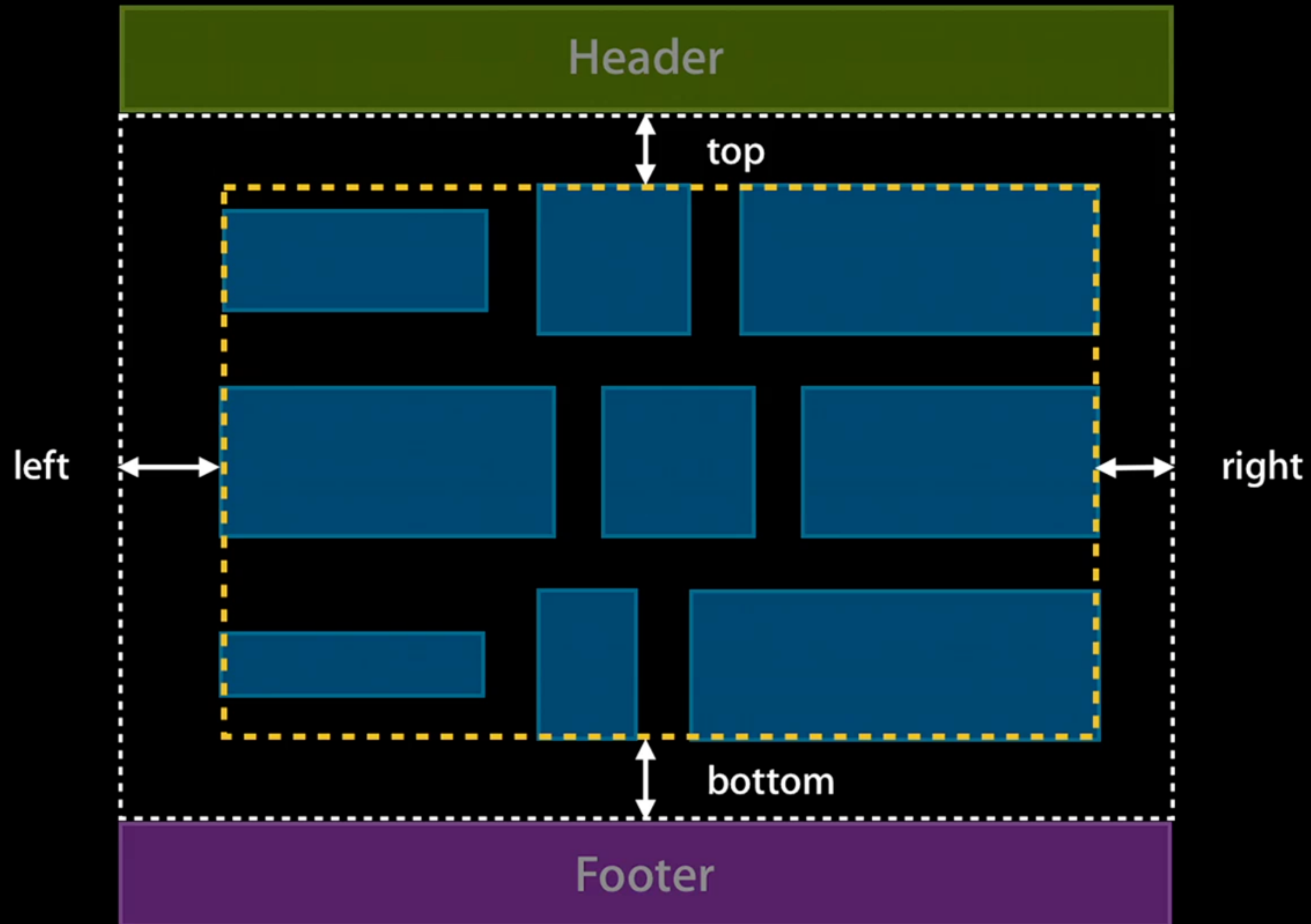
Scrolling Direction

- The scroll direction of your flow layout can defines the base behavior of your entire flow layout
- Dictates the dimensions of the header and footer views:



Section Insets

```
inset = UIEdgeInsetsMake(top, left, bottom, right)
```



Changing the layout

- When you want your layout to change, you need to invalidate your layout.
- You can call `invalidateLayout` to trigger a layout update.
- You can use `performBatchUpdates:completion:` and anything you change inside the update block will invalidate the layout AND cause awesome animations.
- Whenever the bounds of the collection view changes, the layout is invalidated (rotation, scrolling)

Demo

When to go custom?

- If you are constantly changing the location of all the cells.
- Basically, if your collection view doesn't resemble a grid, its time to go custom.

Required overrides on UICollectionViewLayout

- `collectionViewContentSize`: Returns the width and height of the collection view's contents. **This is the entire size of the collection view's content, not just what is visible.**
- `layoutAttributesForElementsInRect`: Returns the layout information for the cells and views that intersect the specified rectangle. **In order for the collection view to know which attribute goes to cells or views, you must specify the `elementCategory` on the attribute (cell, supplementary view, decoration view)** **This is constantly called, every time the user scrolls the collection view. Yikes.**
- `layoutAttributeForItemAtIndexPath`: Use this method to provide layout information for your collection view's cells. Do not use this method for supplementary or decoration views.
- `layoutAttributesForSupplementaryViewOfKind:atIndexPath:` &
`layoutAttributesForDecorationViewWithReuseIdentifier:atIndexPath:` same as above but for supplementary views

UICollectionViewLayout Order of Operations

1. `prepareLayout`
2. `collectionViewContentSize`
3. `layoutAttributeForElementsInRect` (which will probably call `layoutAttributesForIndexPath`)

If the layout is invalidated, `prepareLayout` is called and this cycle is repeated.

Internationalization & Localization

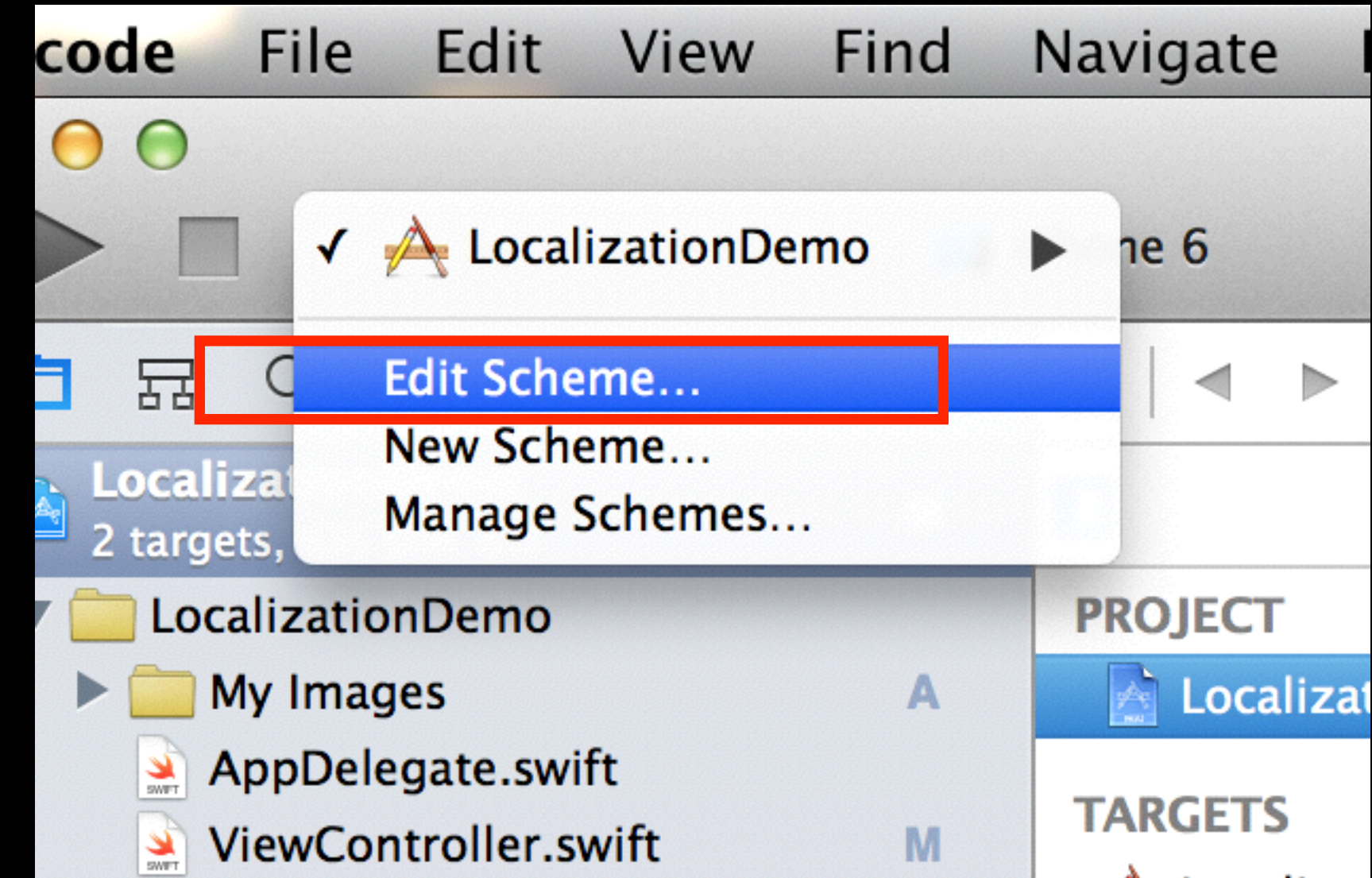
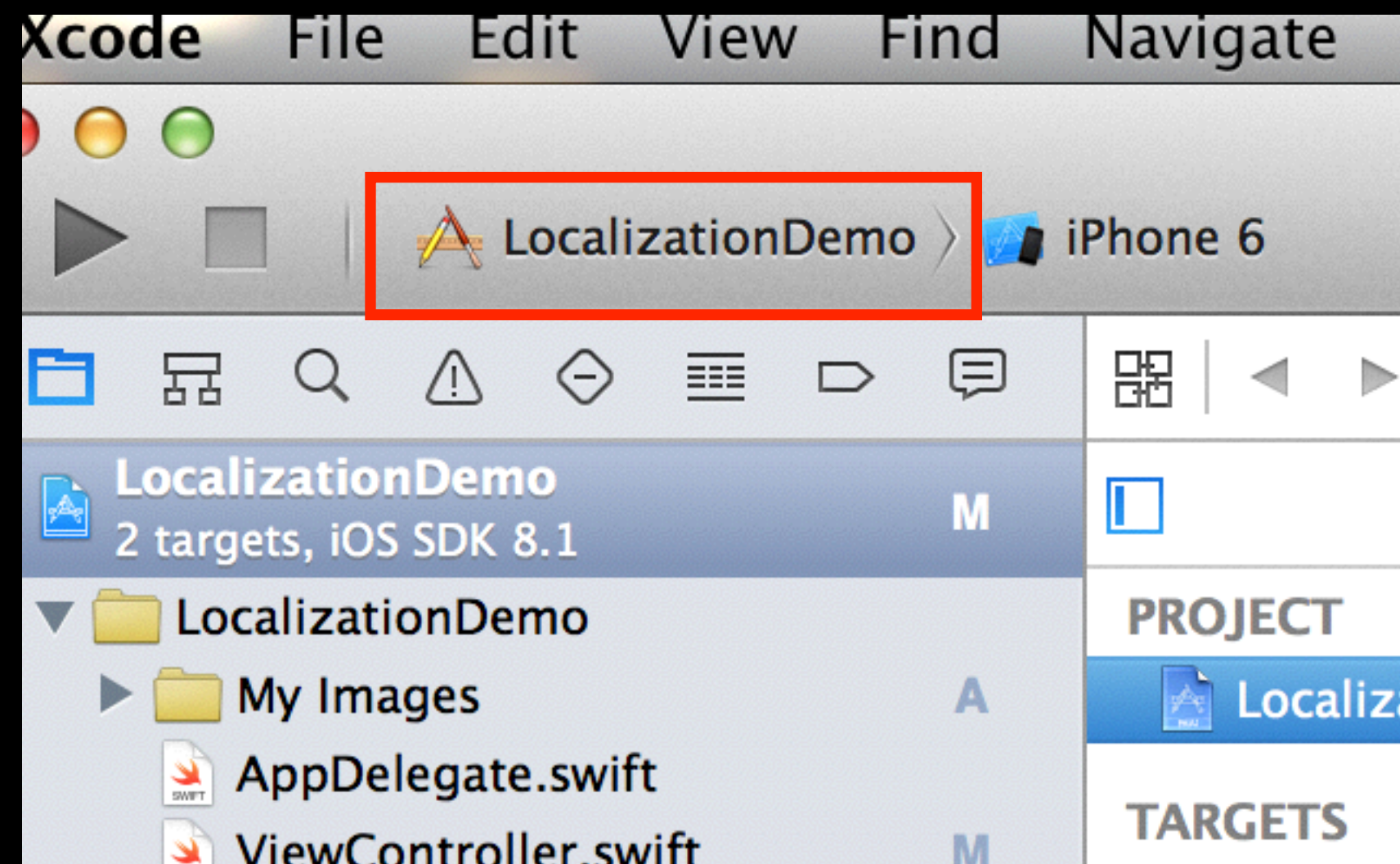
“It’s like flossing, everyone knows
they should do it, but they probably don’t do it” - NSHipster

Localization and Internationalization

- **Localization** is the process of translating your app into multiple languages
- **Internationalization** is the process of creating an app that adapts to multiple languages and regions
- Internationalization is what you do; Localization is what a translator does
- Most people just refer to it all as localization. But its good to know the difference.

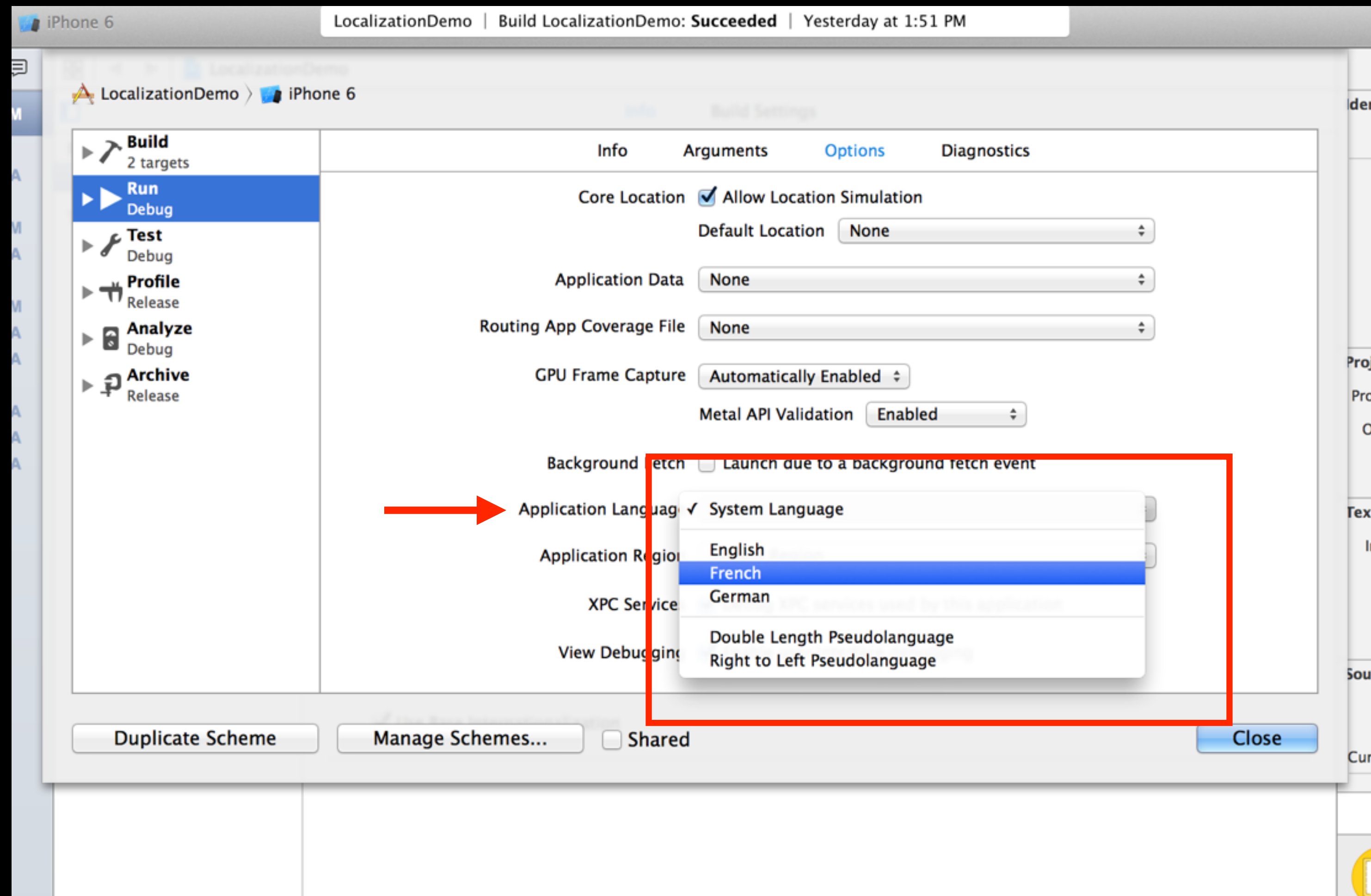
Switching Locales on the simulator via Xcode

3 easy steps!



Switching Locales on the simulator via Xcode

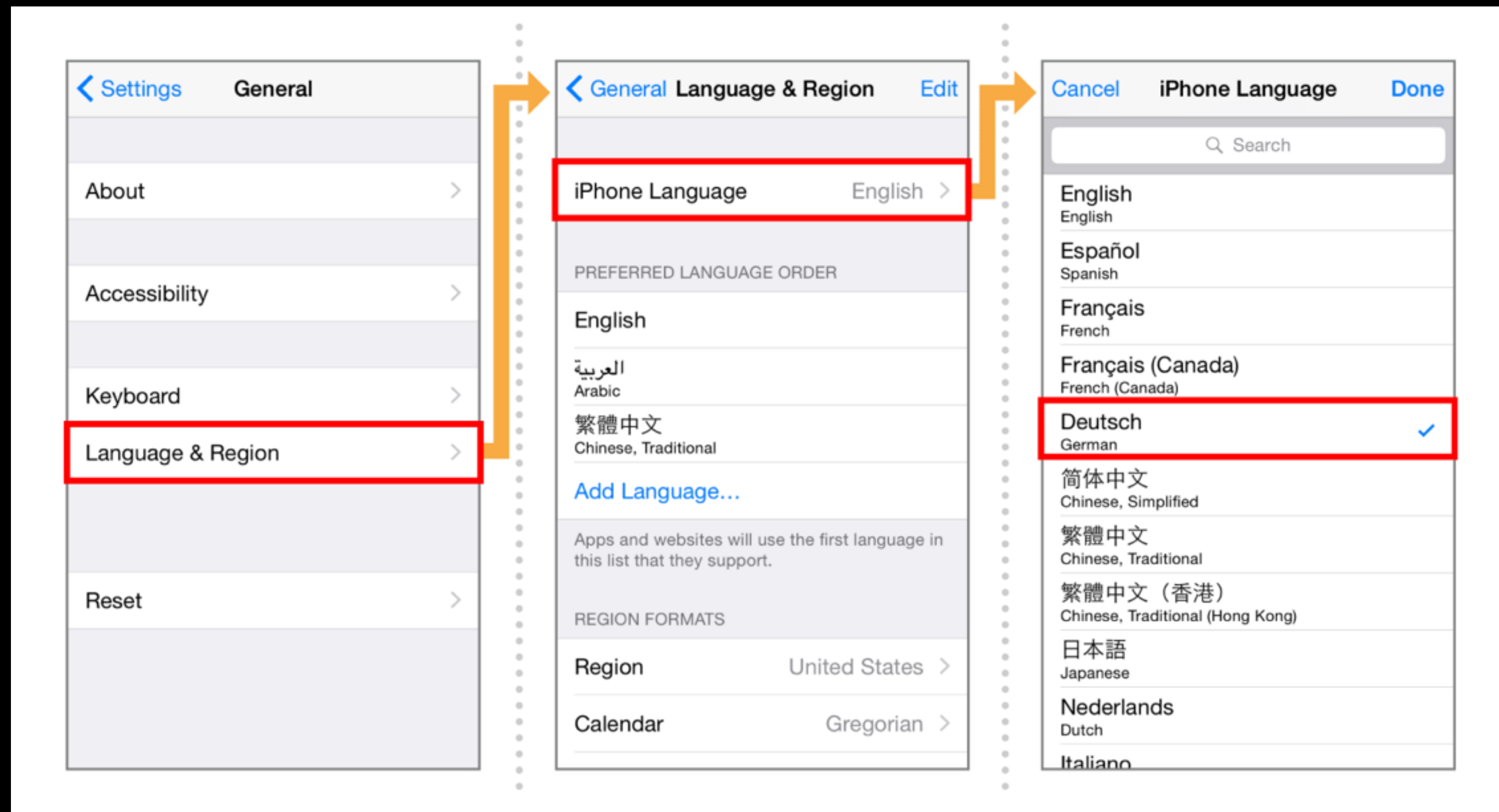
last step



Switching Locales on a Device

Use the **Settings** app to change Locales on Simulator or iOS

Be careful! Difficult to return to previous Language



Base Internationalization

- Technique to separate user-facing strings from Storyboard and XIBs
- Relieves localizers from having to modify Storyboards and XIBs
- Xcode will generate language specific files for each .storyboard and .xib file
- Enabled by default in Xcode 5 and later
- **Just go to your Project's Info screen, and hit the + button under localizations to add a language.**

Expected length of Text

According to IBM's Globalization Guidelines, expect translations from English to many European languages to have 2 or 3 times to number of characters

| # of chacters | Additional space |
|---------------|------------------|
| < 10 | 100 - 200% |
| 11 - 20 | 80 - 100% |
| 21- 30 | 60 - 80% |
| 31 - 50 | 40 - 60% |
| 51 - 70 | 31 - 40% |
| 70+ | 30% |

Example Translations

| Language | Example Text | Length | Expansion |
|----------|---|--------|-----------|
| English | Set the power switch to 0. | 26 | |
| 11 - 20 | Placez l'interrupteur de tension à 0. | 37 | 42% more |
| 21- 30 | Ponga el interruptor de alimentación de corriente en 0. | 55 | 112% more |

AutoLayout Techniques

- Remove fixed width constraints
- Use intrinsic content size when you can
- Use leading and trailing attributes
- Pin views to adjacent views
- Test, test, and test layout changes
- Don't set min or max window size (OS X)

Internationalizing Code

- All user facing programmatically generated Strings need to be localized
- Titles, Labels, Error messages
- Use NSLocalizedString macro

NSString

- Use this for all user facing strings. Keys will be replaced by localized strings.

- Where you might have typed out

```
self.title = "Code Fellows"
```

- Instead you type:

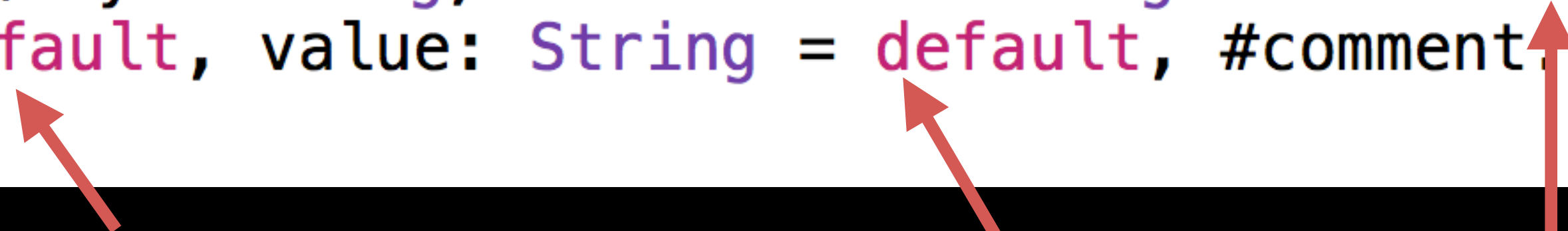
```
self.myLabel.text = NSLocalizedString("Hello",  
                                     tableName: nil,  
                                     bundle: NSBundle.mainBundle(),  
                                     value: "nope",  
                                     comment: "This is a greeting")
```

Well this sucks! Actually we can make it much shorter....

NSLocalizedString

- So NSLocalizedString is a global function in Foundation:

```
/// Returns a localized string, using the main bundle if one is not
/// specified.
func NSLocalizedString(key: String, tableName: String? = default,
    bundle: NSBundle = default, value: String = default, #comment:
    String) -> String
```



- The parameters marked with = default can actually be completely dropped when calling the function!

```
self.myLabel.text = NSLocalizedString("Hello", comment: "This is a greeting")
```

String file

- A file you can generate with Xcode (or a tool called genstring - better option)
- Must be named localizable.strings
- Works like a dictionary, key-value pairing
- You need this to localize strings that aren't set via storyboard or xibs.
- **Keep in mind you only need to translate strings that will be seen by the user!!**
- After creating your set of string files, you can use the NSLocalizedString() macro to reference the strings you defined in the key-value pairings.
- Every line must end with a semi colon;

genstrings

- genstrings is command line tool installed on your mac
- to use it, cd into the directory containing the swift source code (aka swift files) in terminal
- next, run the command `genstrings *.swift`
- this will parse through your code, looking for any `nslocalizedstrings` and generate a `localizable.strings` file with all your localized strings complete with the comments as well

String file

English/base file

```
1  /*
2   Localizable.strings
3   localtest
4
5   Created by Bradley Johnson
6   Copyright (c) 2015 BPJ. All rights reserved.
7  */
8
9
10 "Hello" = "Hello";
11 "Goodbye" = "Goodbye";
12 "Cool" = "Cool";
```

French file

```
1  /*
2   Localizable.strings
3   localtest
4
5   Created by Bradley Johnson
6   Copyright (c) 2015 BPJ. All rights reserved.
7  */
8
9
10 "Hello" = "Bonjour";
11 "Goodbye" = "Au Revoir";
12 "Cool" = "Frais";
```


Workflow

- Use `NSString` for all user facing strings
- Create Base Internationalized version for XIB and Storyboard files
- Setup AutoLayout with variable length text content in mind
- Create String files for each language supported
- Test app with different languages, looking for layout issues.

Demo

Accessibility

Accessibility

- Set of developer tools and APIs that allows iOS app to the provide best mobile experience for customers of all needs.
- Captioning— iOS supports captioned video during playback
- VoiceOver— Screen reader to drive interface using Voice
- Speech— Read selected text aloud in multiple languages
- Guided Access— Limits iOS to running one app

Accessibility; Why?

- It's the right thing to do
- Increases your user base
- Allows people to use app without seeing the Screen
 - Apple is a big advocate of Accessibility.
 - First class citizen on iOS

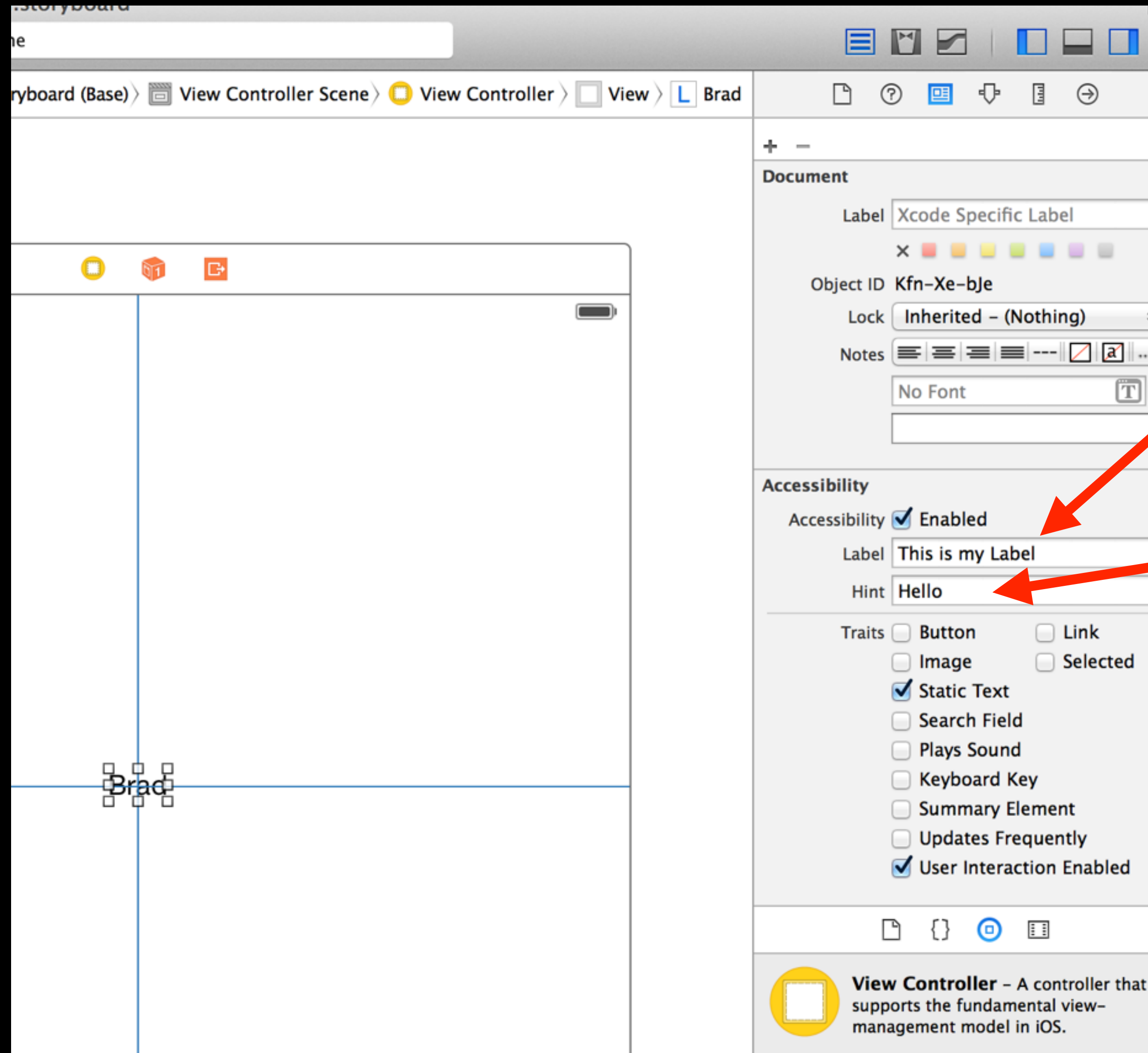
VoiceOver

- VoiceOver is a gesture based screen reader built into iOS that lets you use the phone if you can't see the screen.
- Uses a cursor or focus ring to denote current UI element
- Use touch and drag events to read whats onscreen.
- Works with all default iOS apps out of box.
- Add support to your app today

VoiceOver Details

- Almost all Apple created UI elements have accessibility functionality built into them.
- Voice over functionality is very easy to setup, so theres no excuse!
- For the most part you can just do it all via storyboard, and doing it in code is also as easy as a few lines of code.

VoiceOver Storyboard



This is what the label will speak to the user, if its blank the system will just speak what the label actually says

The hint will play after a short pause if the user stays selected on the UI element

These properties can be accessed via code, like most other storyboard properties

Accessibility Attributes

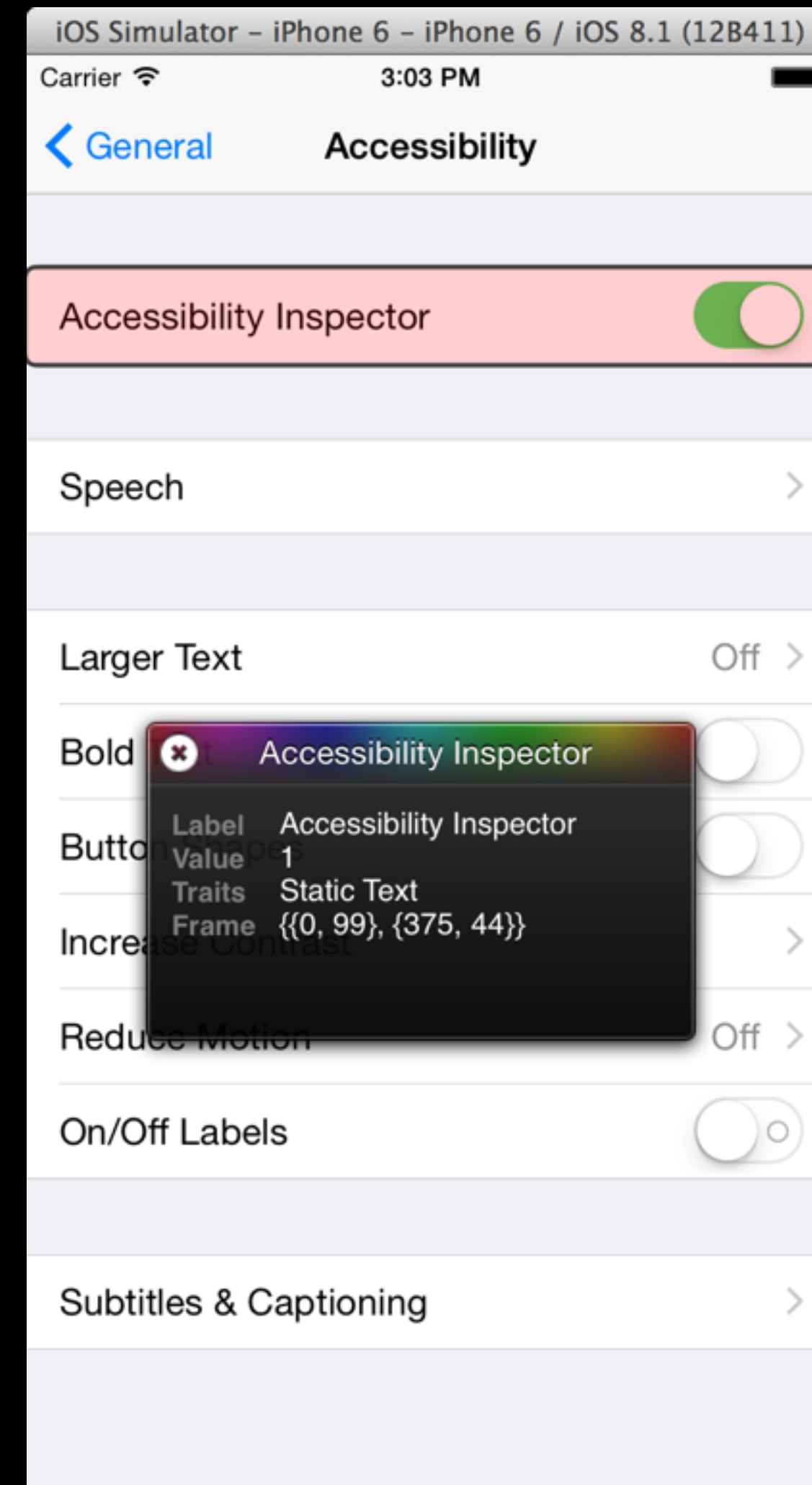
- You just saw 'Label' and 'Hint', there are 3 more:
- Traits: a combo of one or more individual traits, each describing a single aspect of an elements state, like if its a button or if user interaction is enabled or disabled on it.
- Frame: The frame of the element in screen coordinates (just like regular frame)
- Value: Used to store the value if that is applicable, like a slider "50%", or a switch "on/off"

VoiceOver Demo

Accessibility Inspector

- Displays accessibility information about each accessible element in an app
- Runs in simulator and lets you see the accessibility label, value, hint, traits, and frame for each element onscreen.
- Helpful for testing the accessibility of your app during development, but..
- it's pretty good but you will still want to test your app with VoiceOver on a physical device

Accessibility Inspector



Demo