iOS Programming

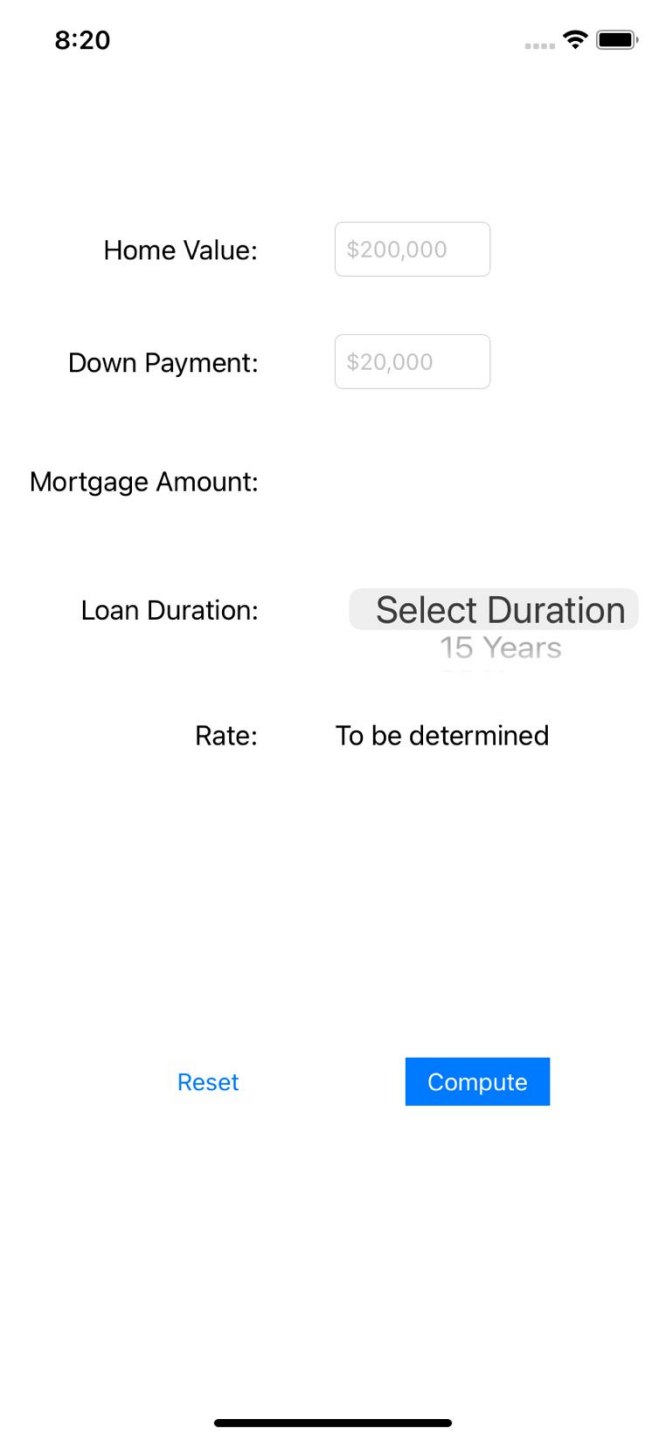
Lecture 9

Recap

We built a single page app today – fully functioning

View Hierarchy

Actions and Outlets



Today

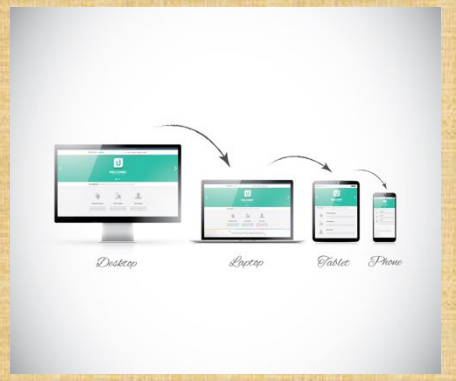
Adaptive UI

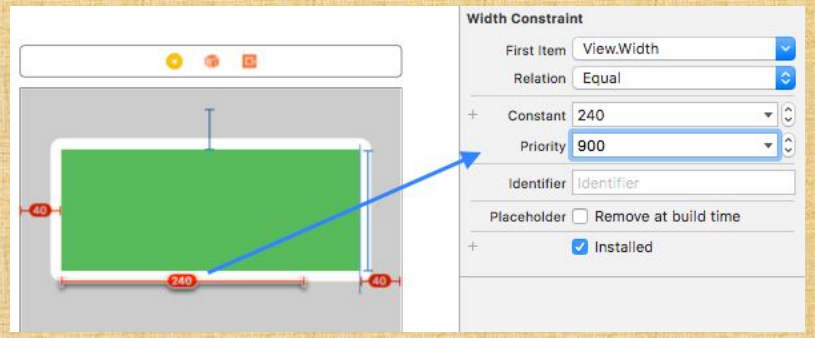
Images

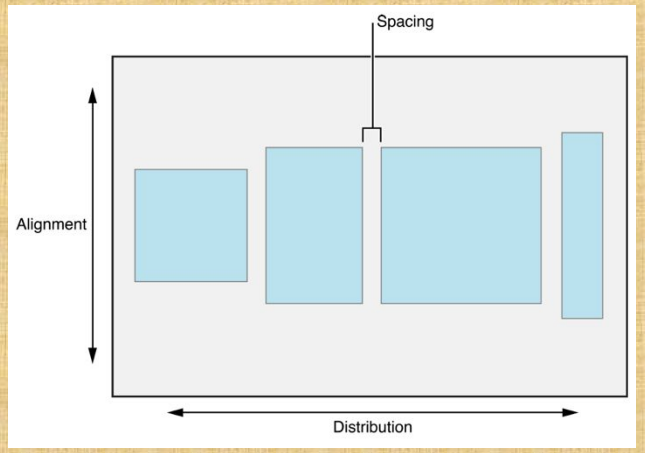
App thinning

Constraints

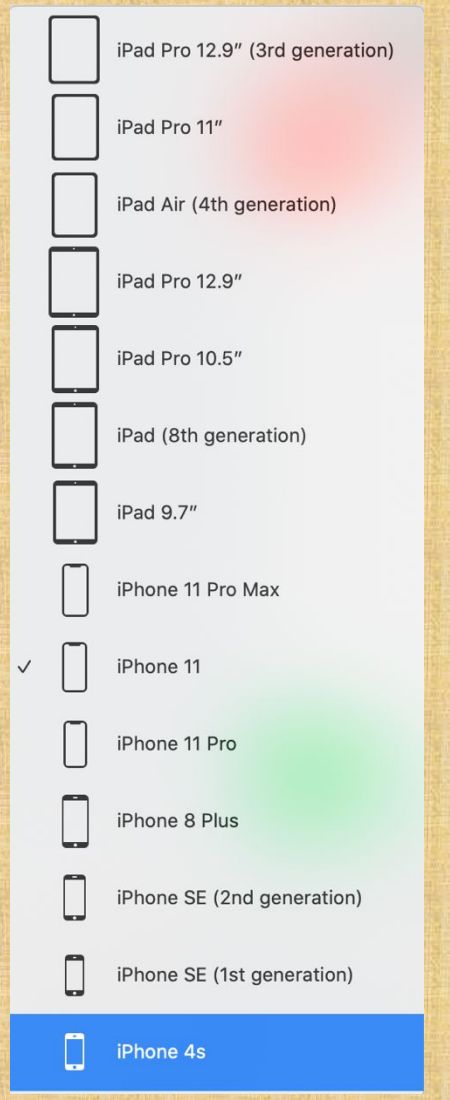
Stack Views



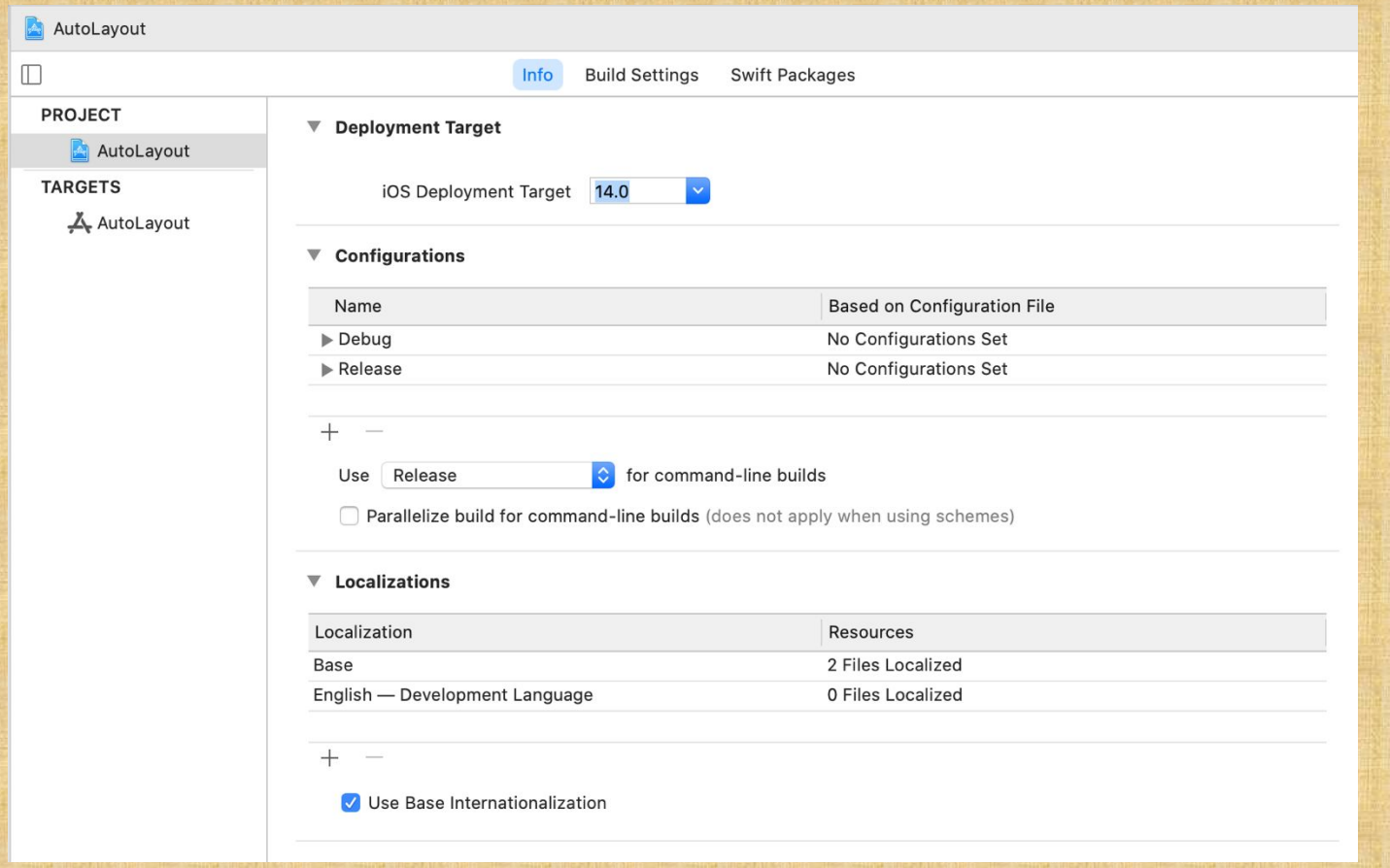




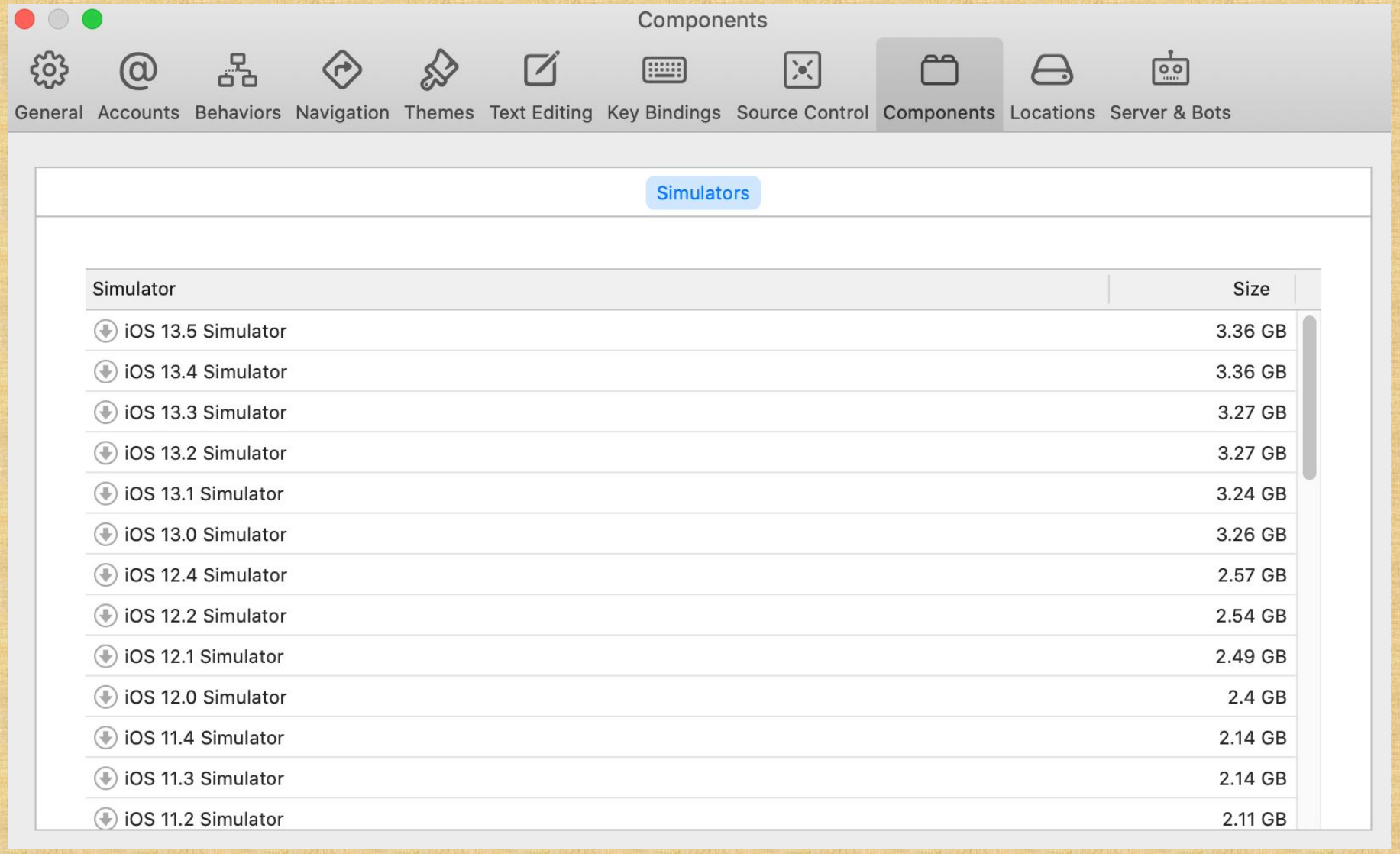
Deployment Targets



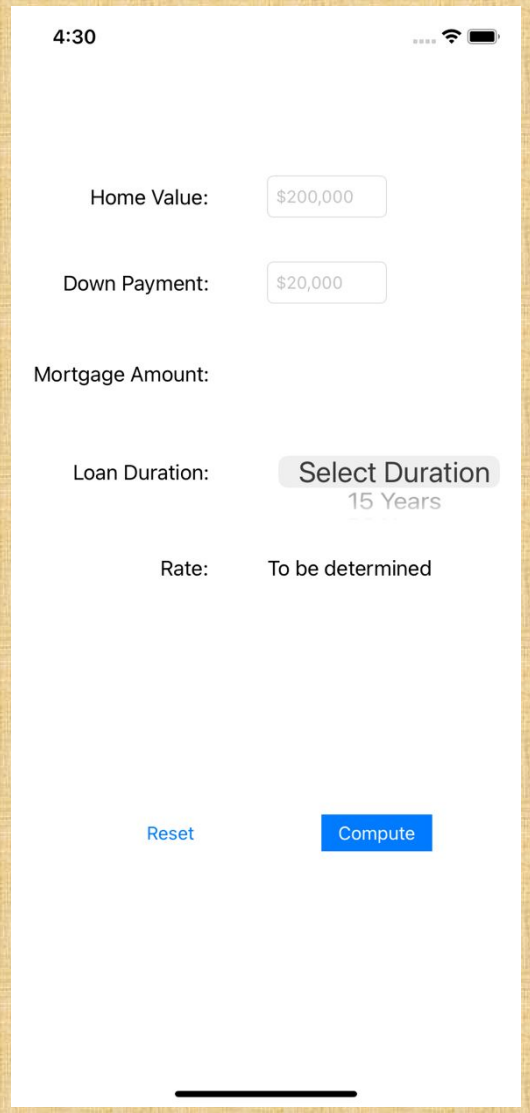
Can I build apps for iPhone 4s still?



How to get more simulators

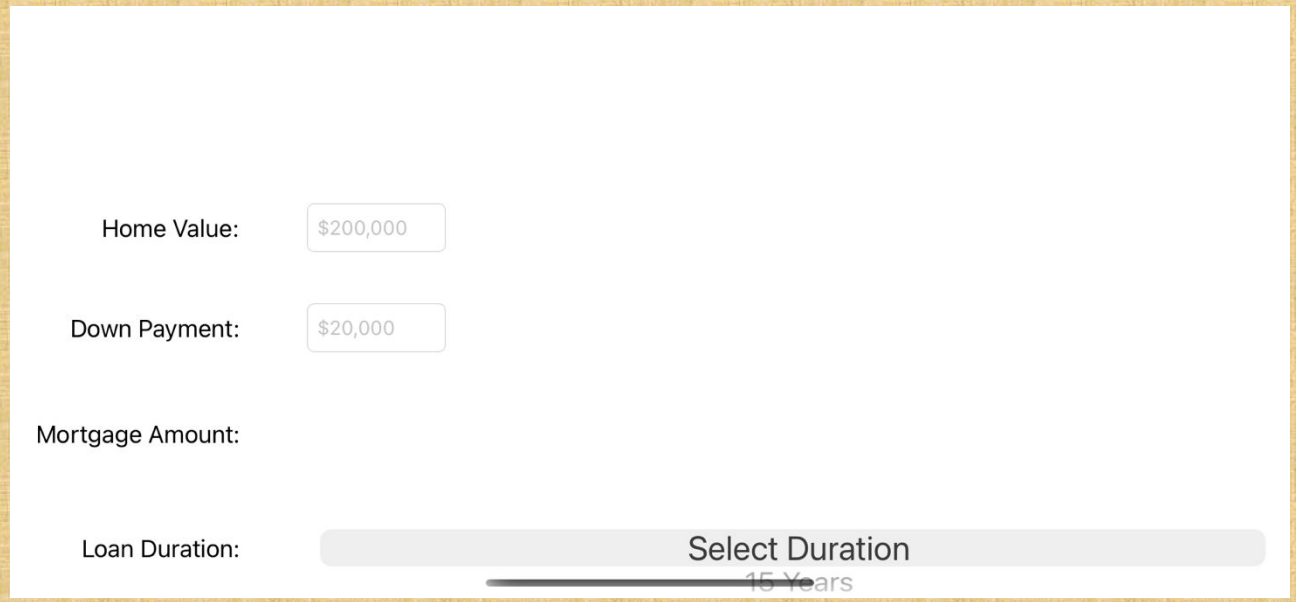


Mortgage App – Do we have issues?



Let's try landscape – What do we observe?

What all issues we see?



Solution 1 – Lock orientation

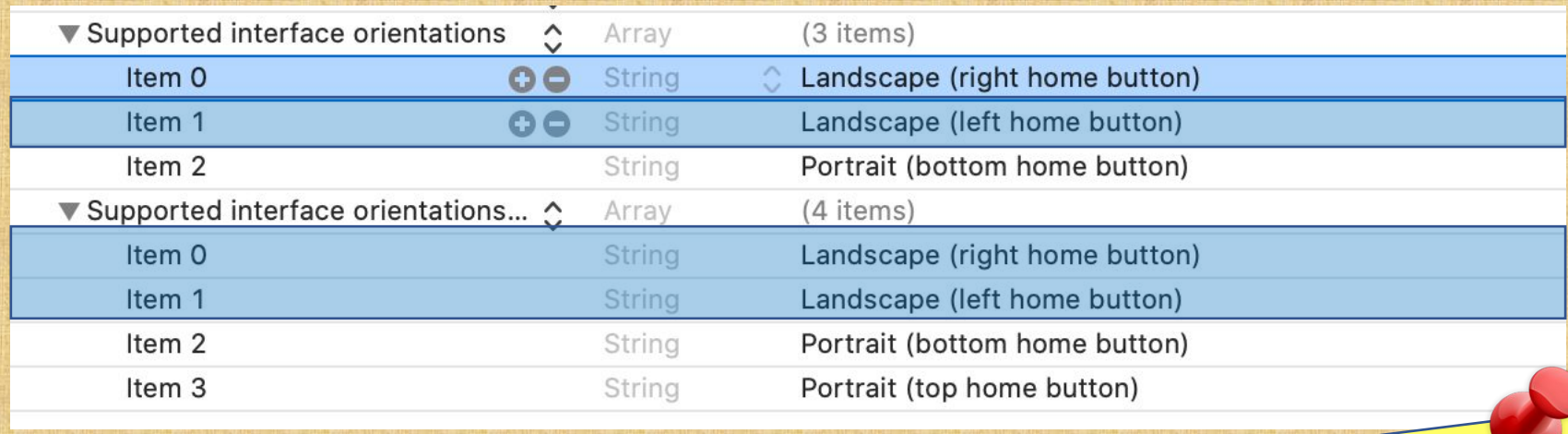
You can do so programmatically by overriding UIViewController functions

```
// Set the shouldAutorotate to False
override open var shouldAutorotate: Bool {
   return false
}


// Specify the orientation.
override open var supportedInterfaceOrientations: UIInterfaceOrientationMask {
   return .portrait
}
```

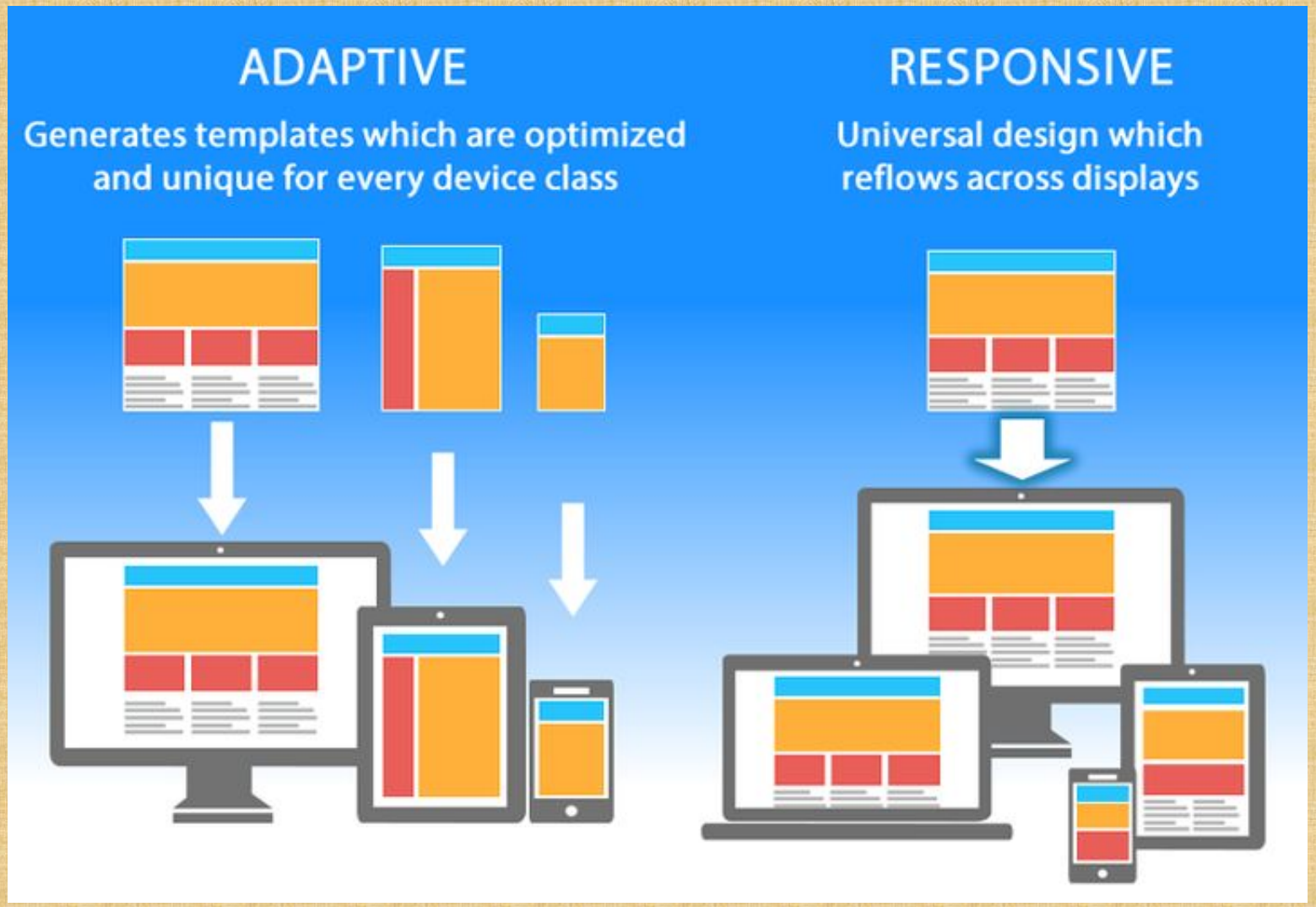
Solution 1 – Lock orientation

You can do so by configuration for entire app – Info.plist



Let's discuss pros and cons of programmatic vs config

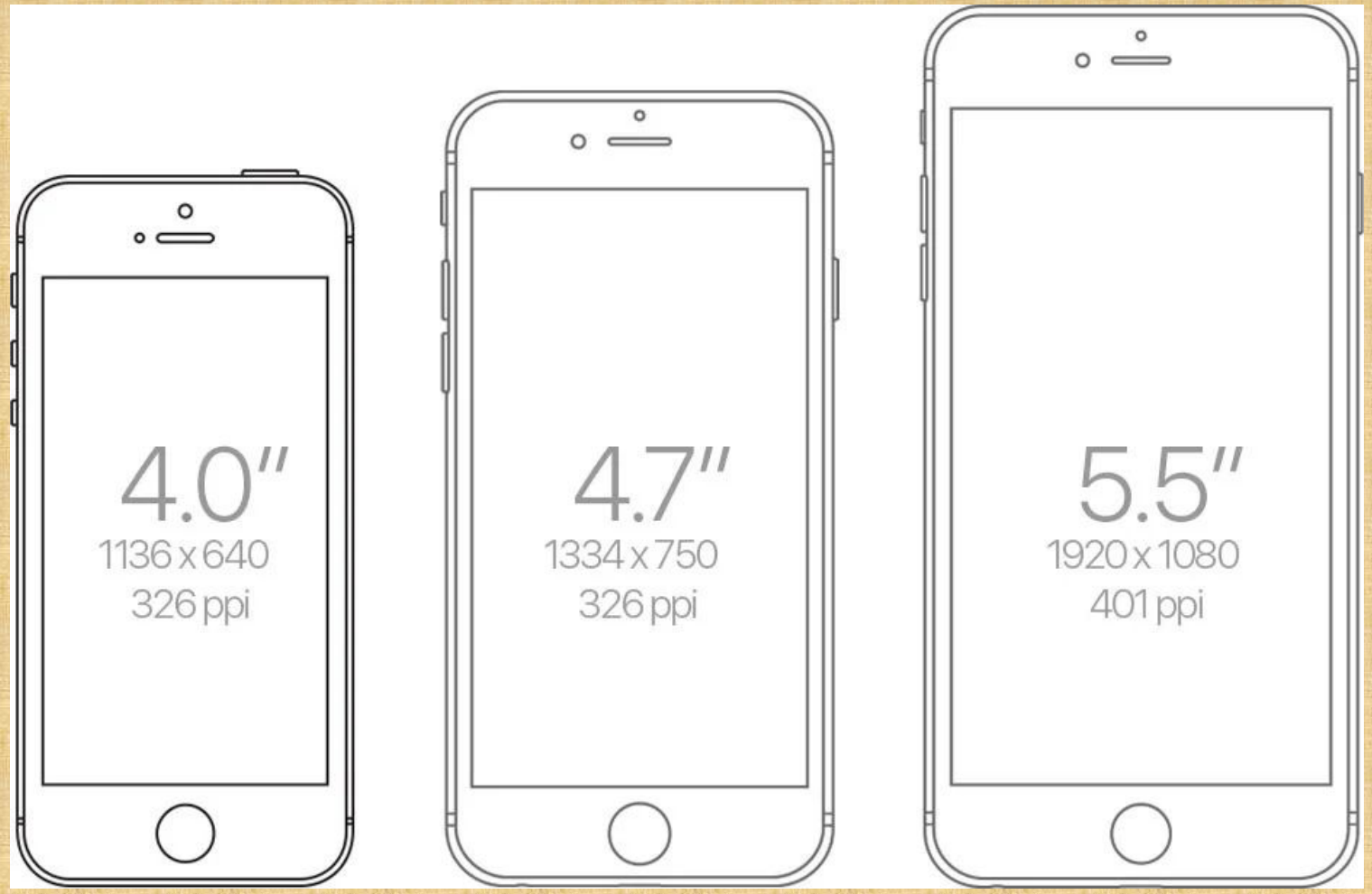
Adaptive vs Responsive



iPhone Screen Sizes

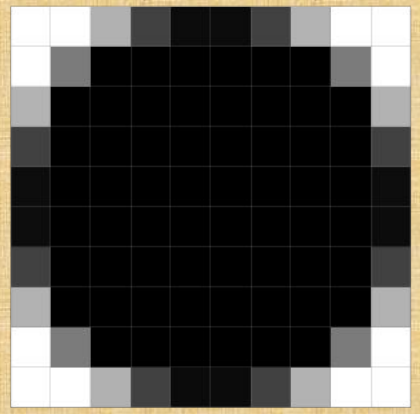
Model	XS Max 11 Pro Max	X, XS 11 Pro	XR 11	8+ 7+ 6+, 6S+	8 7 6, 6S	SE 5, 5S, 5C	4, 4S	3G, 3GS 2G
Points	414x896	375x812	414x896	414x736	375x667	320x568	320x480	320x480
Rendered @	3x	3x	2x	3x	2x	2x	2x	1x
Rendered Pixels	1242x2688	1125x2436	828x1792	1242x2208	750x1334	640x1136	640x960	320x480
Physical Screen	6.5''	5.8''	6.1''	5.5''	4.7''	4''	3.5''	3.5''

iPhone Screen Sizes



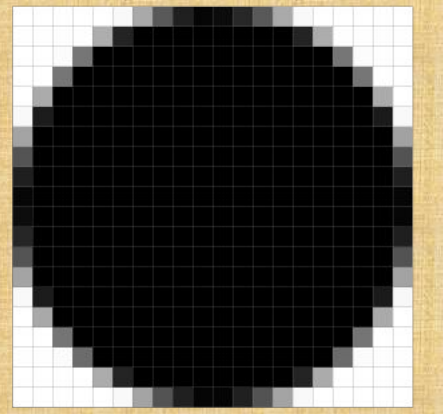
Images – 1x, 2x and 3x

The coordinate system iOS uses to place content onscreen is based on measurements in points, which map to pixels in the display. A standard-resolution display has a 1:1 pixel density (or @1x), where one pixel is equal to one point. High-resolution displays have a higher pixel density, offering a scale factor of 2.0 or 3.0 (referred to as @2x and @3x). As a result, high-resolution displays demand images with more pixels. For example, suppose you have a standard resolution (@1x) image that's 100px × 100px. The @2x version of this image would be 200px × 200px, and the @3x version would be 300px × 300px.



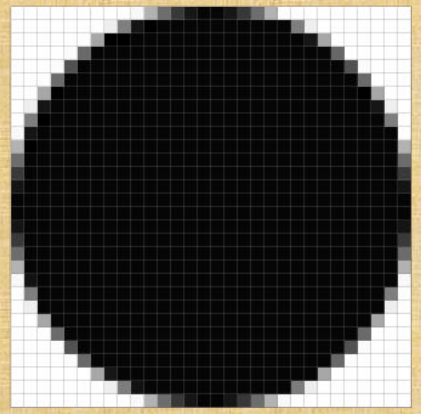
1x

(10 x 10 px)



2x

(20 x 20 px)



3x

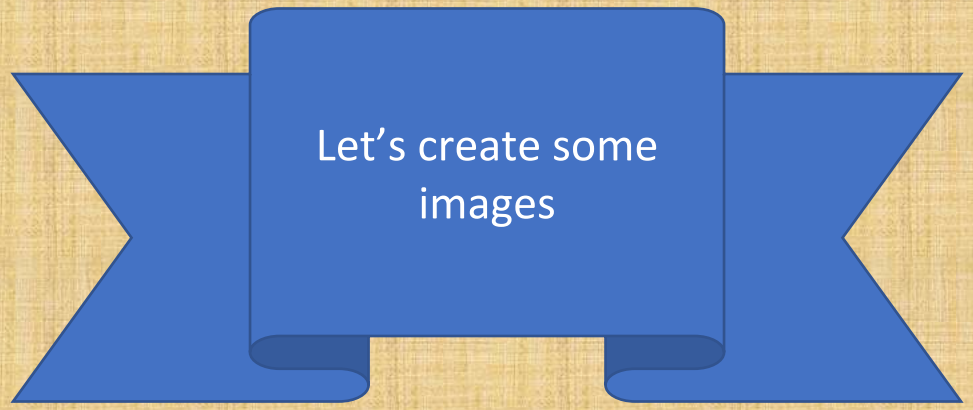
(30 x 30 px)

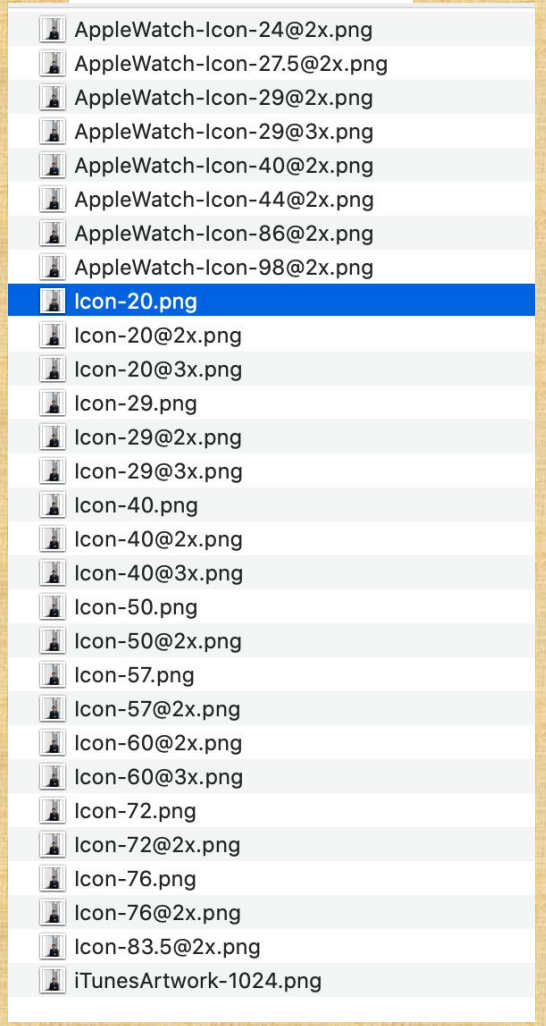
How do I get these images?

In enterprises – UX Designer will share those

For our example we will use:

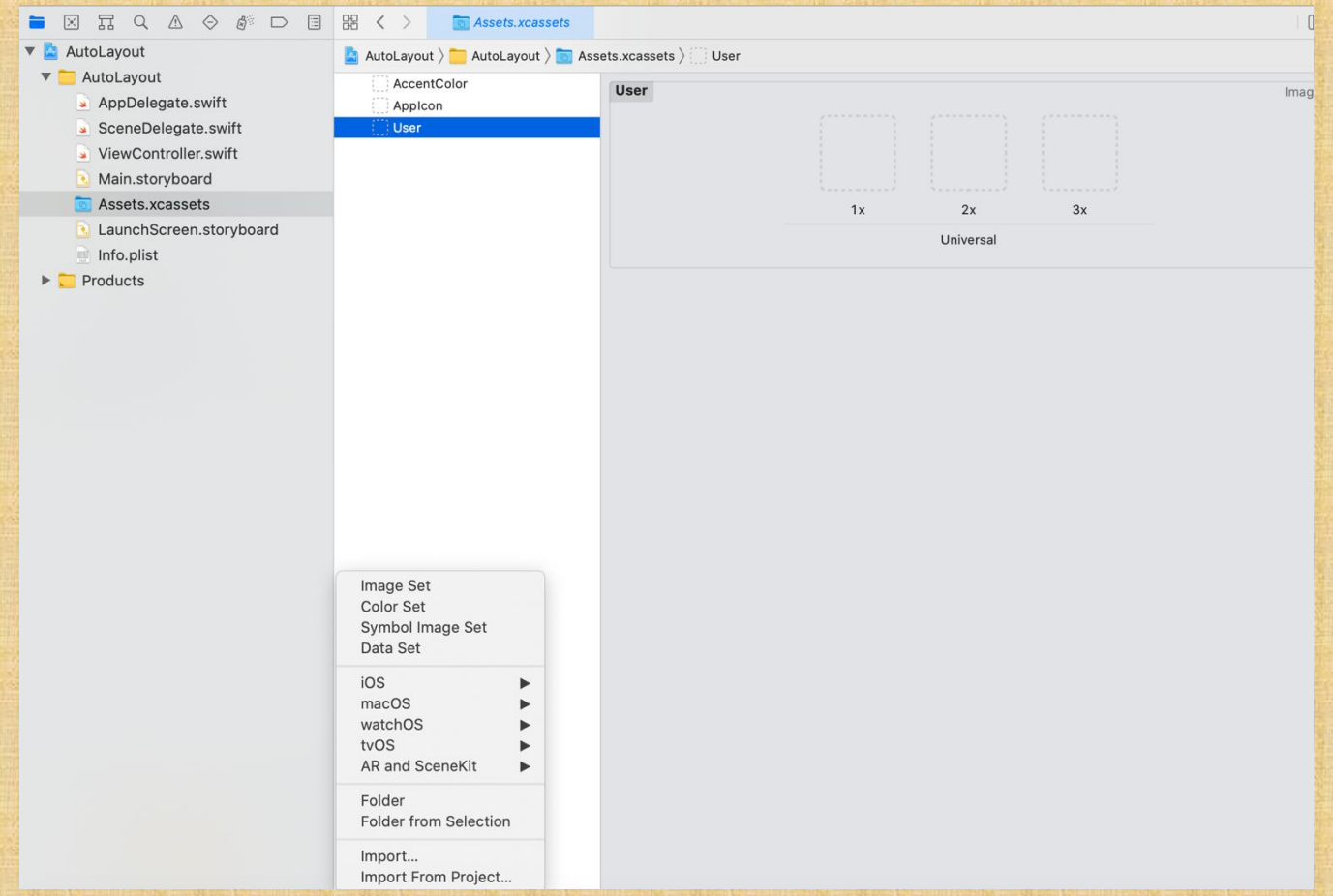
https://hotpot.ai/icon-resizer





Images – Asset Catalogue

You really don't have to worry about figuring which asset to use on which device type. iOS will take care of this.



App Thinning

App thinning is a concept for modern day interactive apps where there are a lot of resources. The App Store and OS install the app according to the device, with a minimal footprint. This helps in making the app which occupies less space, are easy to download and make use of all features. Faster downloads and minimum space occupancy gives a better app experience.

App Thinning - Slicing

Slicing is the process of creating different variations of app bundle for different target devices. A variant contains only the executable architecture and resources those are needed for the target device. The store will create and deliver different variants based on the devices your app supports. Image resources are sliced according to their resolution and device family. GPU resources are sliced according to device capabilities. For tvOS apps, assets in catalogs shared between iOS and tvOS targets are sliced and large app icons are removed. When the user installs an app, a variant for the user's device is downloaded and installed.

Note: Sliced apps are supported on devices running 9.0 and later; otherwise, the store delivers universal apps to customers.

App Thinning - Bitcode

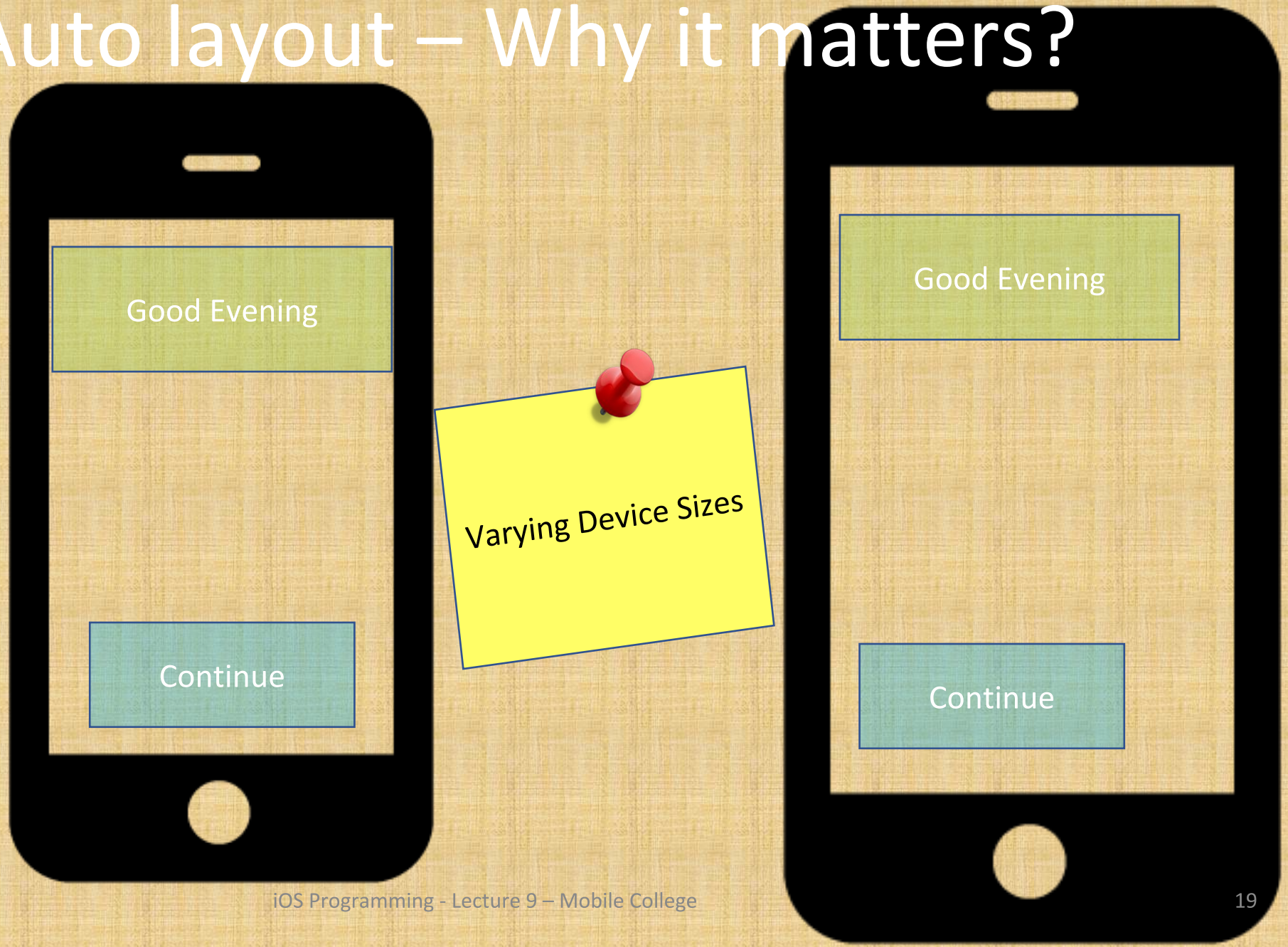
Bitcode is an intermediate representation of a compiled program. Apps you upload to iTunes Connect that contain bitcode will be compiled and linked on the store. Including bitcode will allow Apple to re-optimize your app binary in the future without the need to submit a new version of your app to the store.

Xcode hides symbols generated during build time by default, so they are not readable by Apple. Only if you choose to include symbols when uploading your app to iTunes Connect would the symbols be sent to Apple. You must include symbols to receive crash reports from Apple.

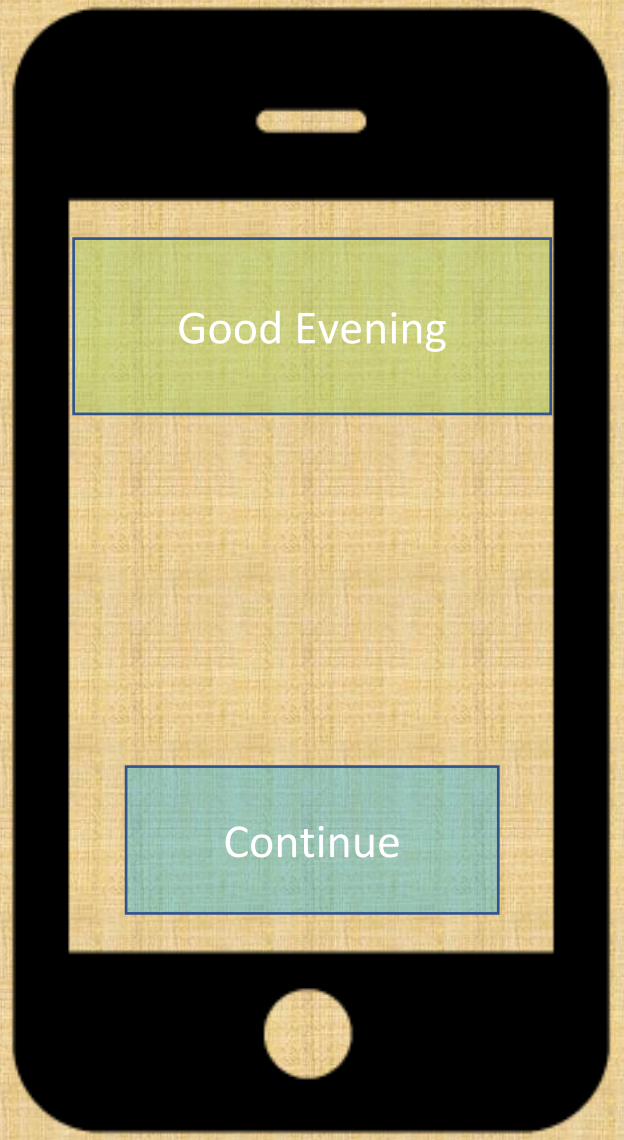
App Thinning - On Demand Resources

On-demand resources are resources—such as images and sounds—that you can tag with keywords and request in groups, by tag. The store hosts the resources on Apple servers and manages the downloads for you. On-demand resources enable faster downloads and smaller app sizes, improving the first-time launch experience. For example, a game app may divide resources into game levels and request the next level of resources only when the app anticipates that the user will move to that level. Similarly, the app can request In-App Purchase resources only when the user buys the corresponding in-app purchase.

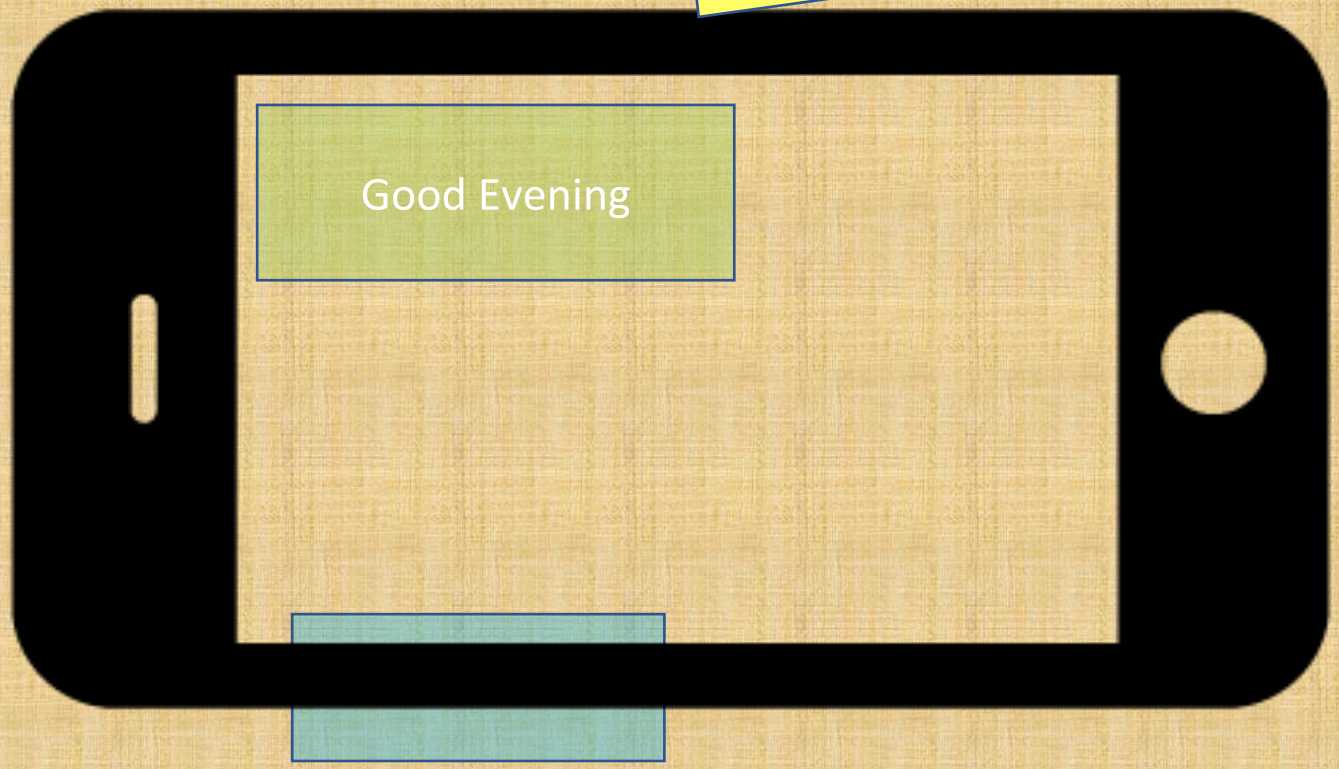
Auto layout – Why it matters?



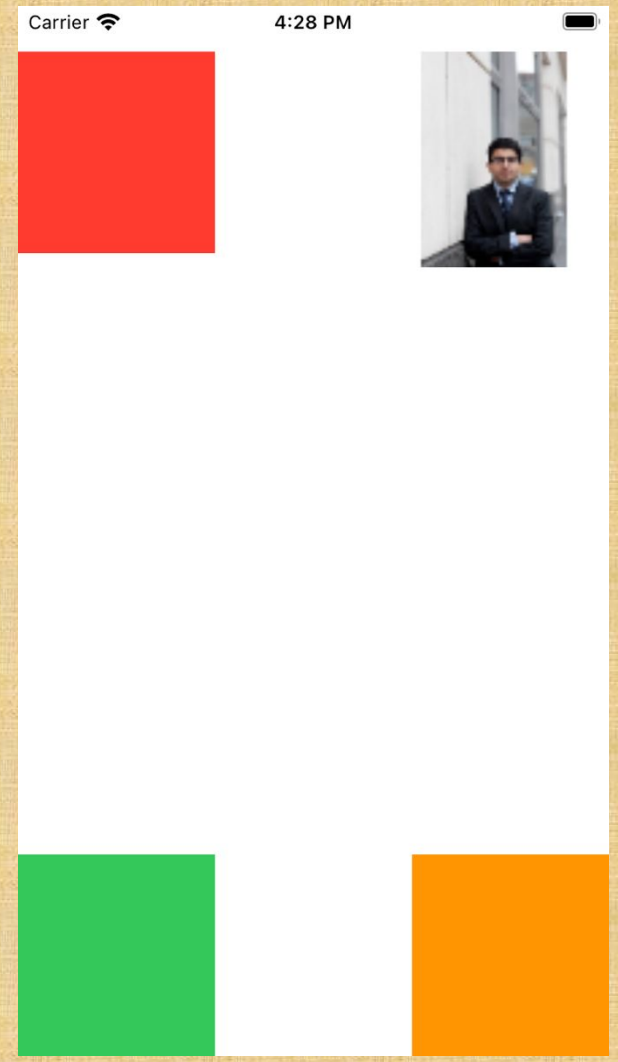
How does it impact



and landscape layout

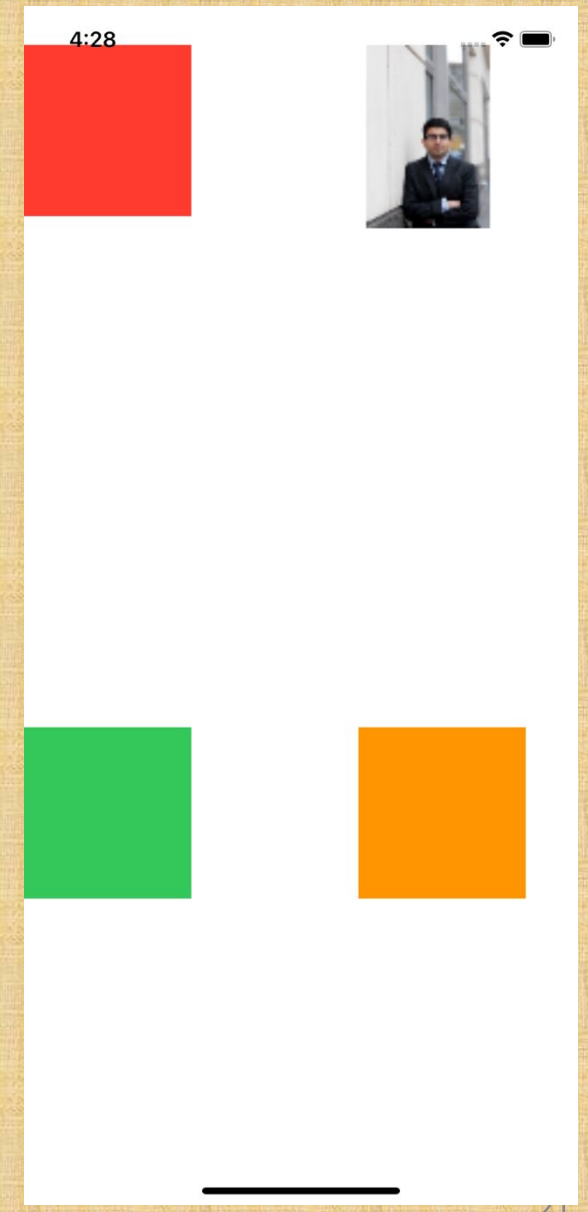


Let’s try with a corner example

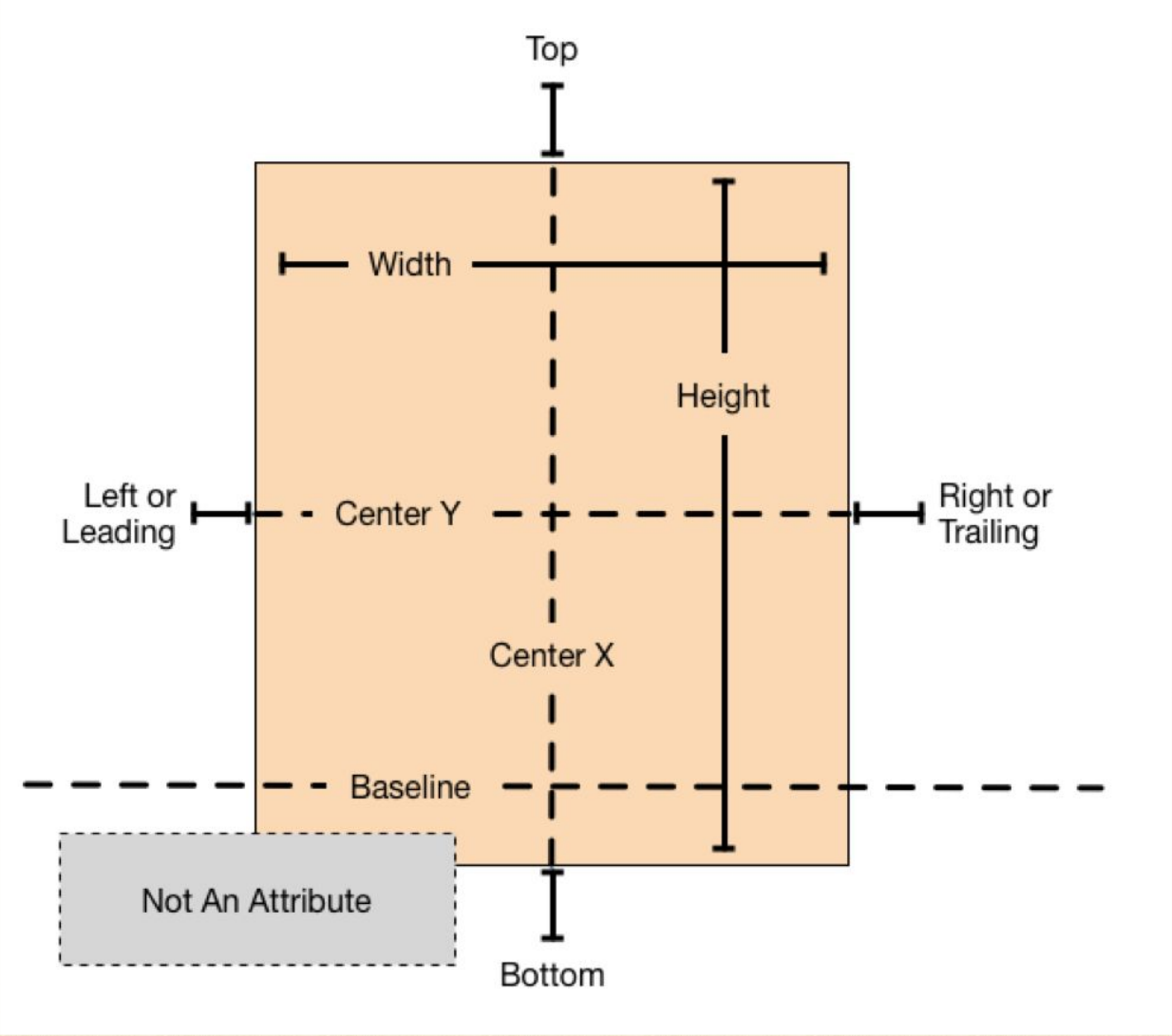


Looks good on iPhone SE2 but pretty weird on iPhone X

What else do we observe?

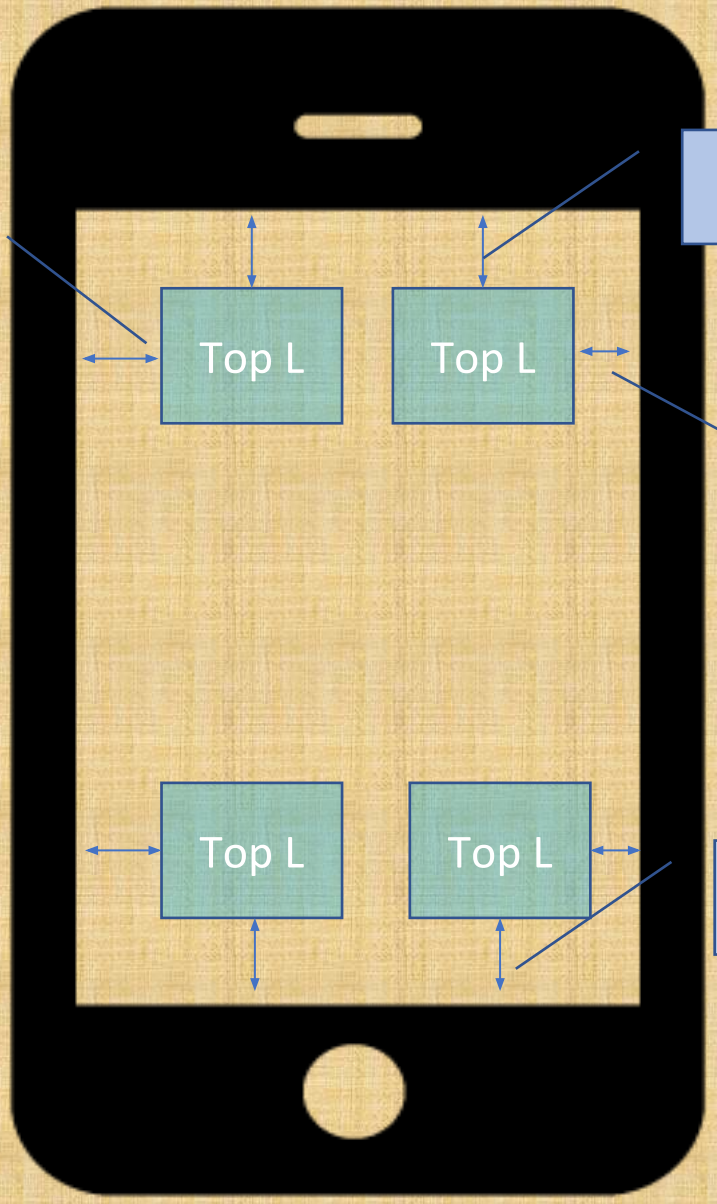


Constraints



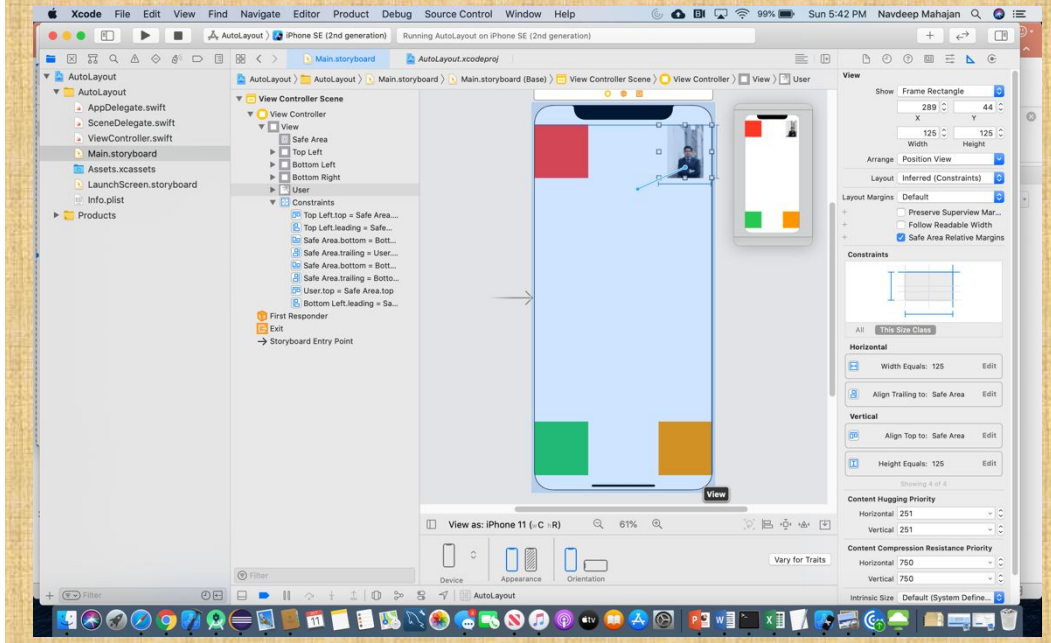
Define relationship between container and in between controls

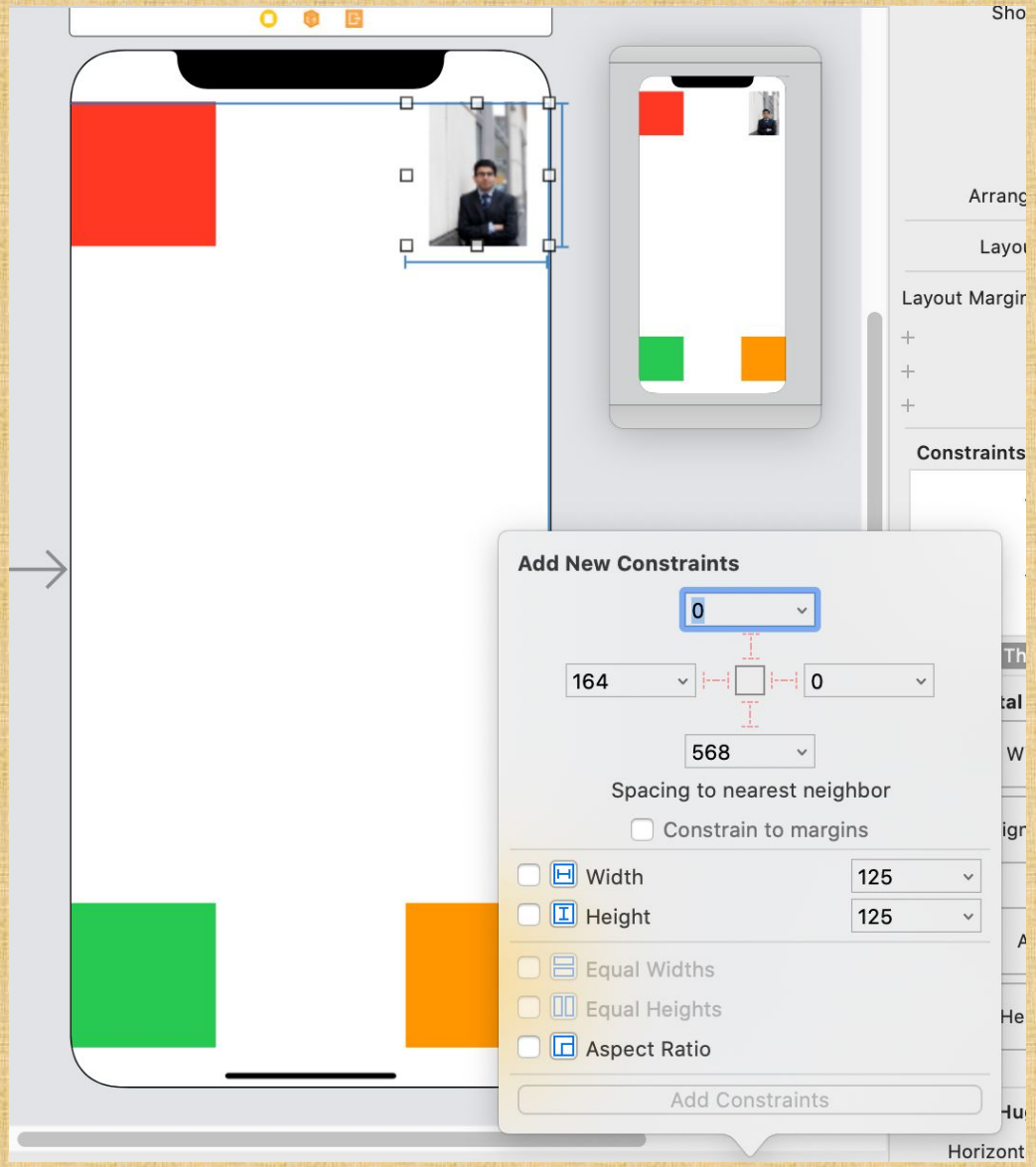
Now let’s see how we should lay it out



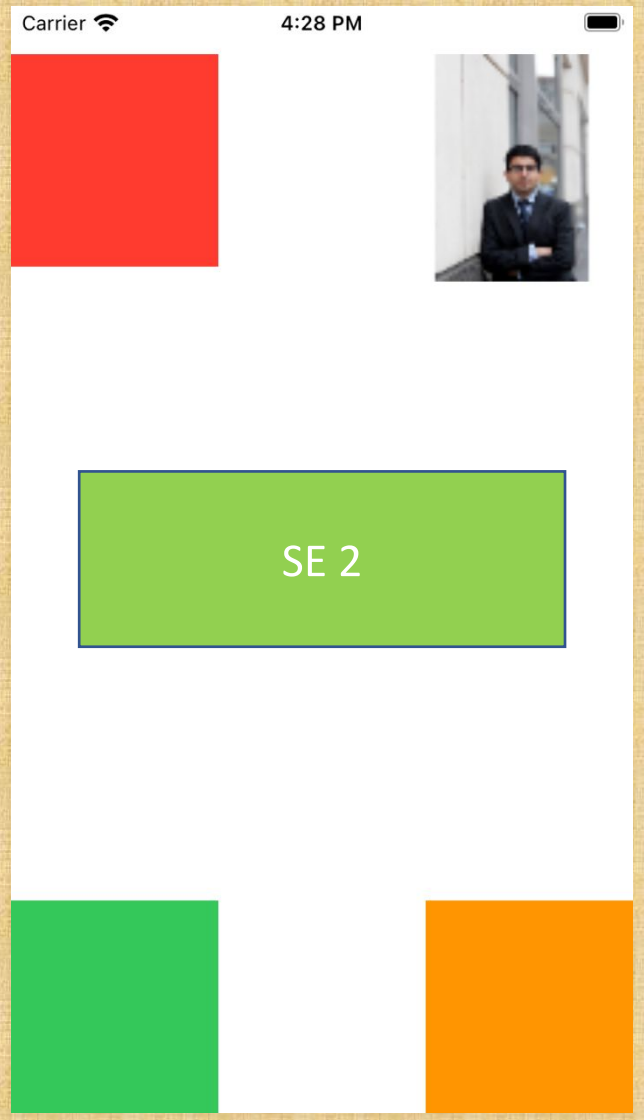
+ We need to specify width and height of the views

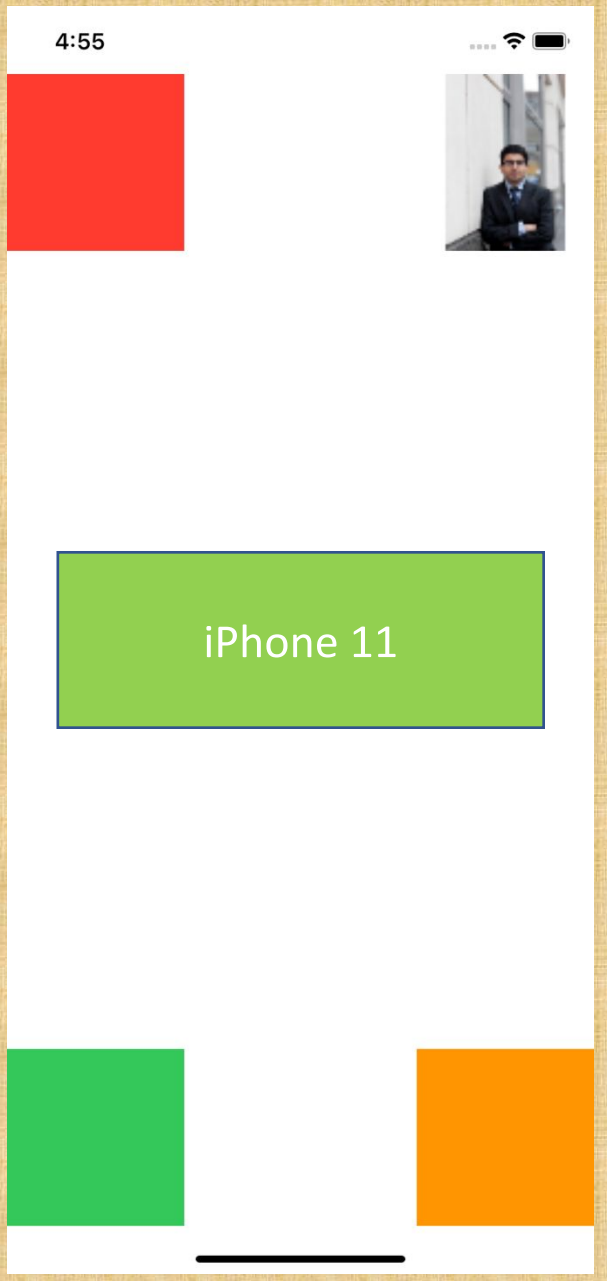
How to add constraints



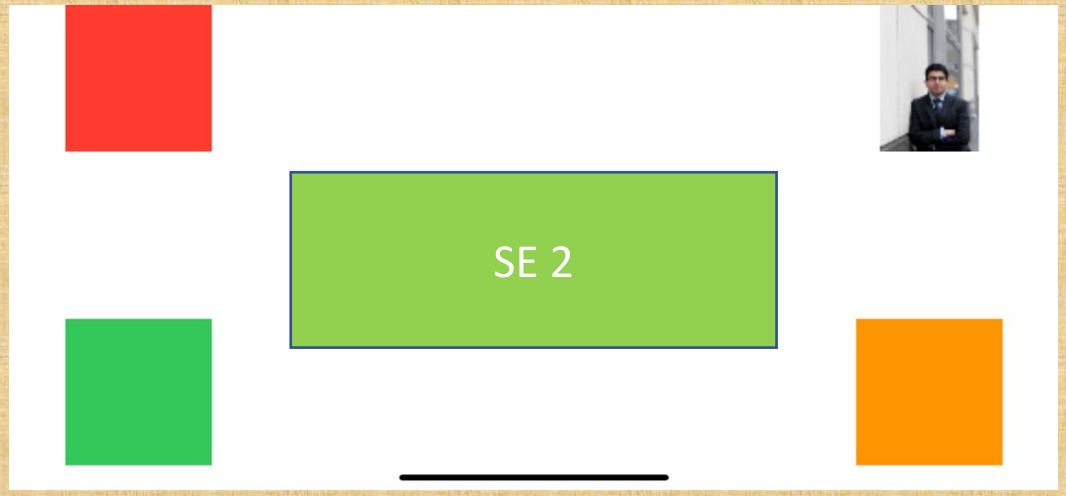


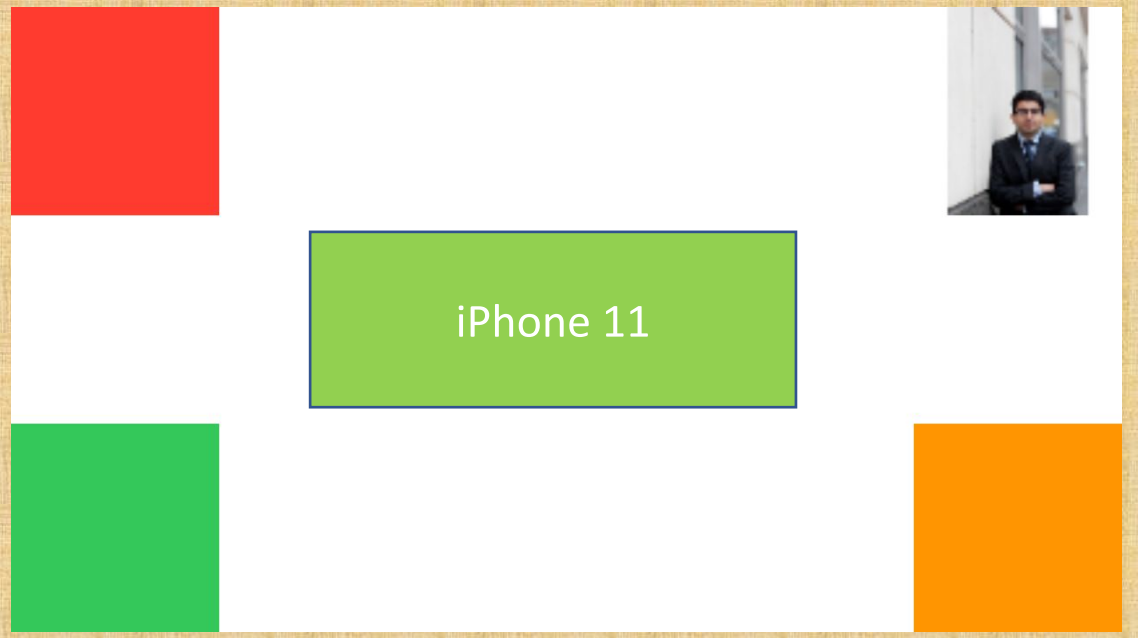
Post Constraints





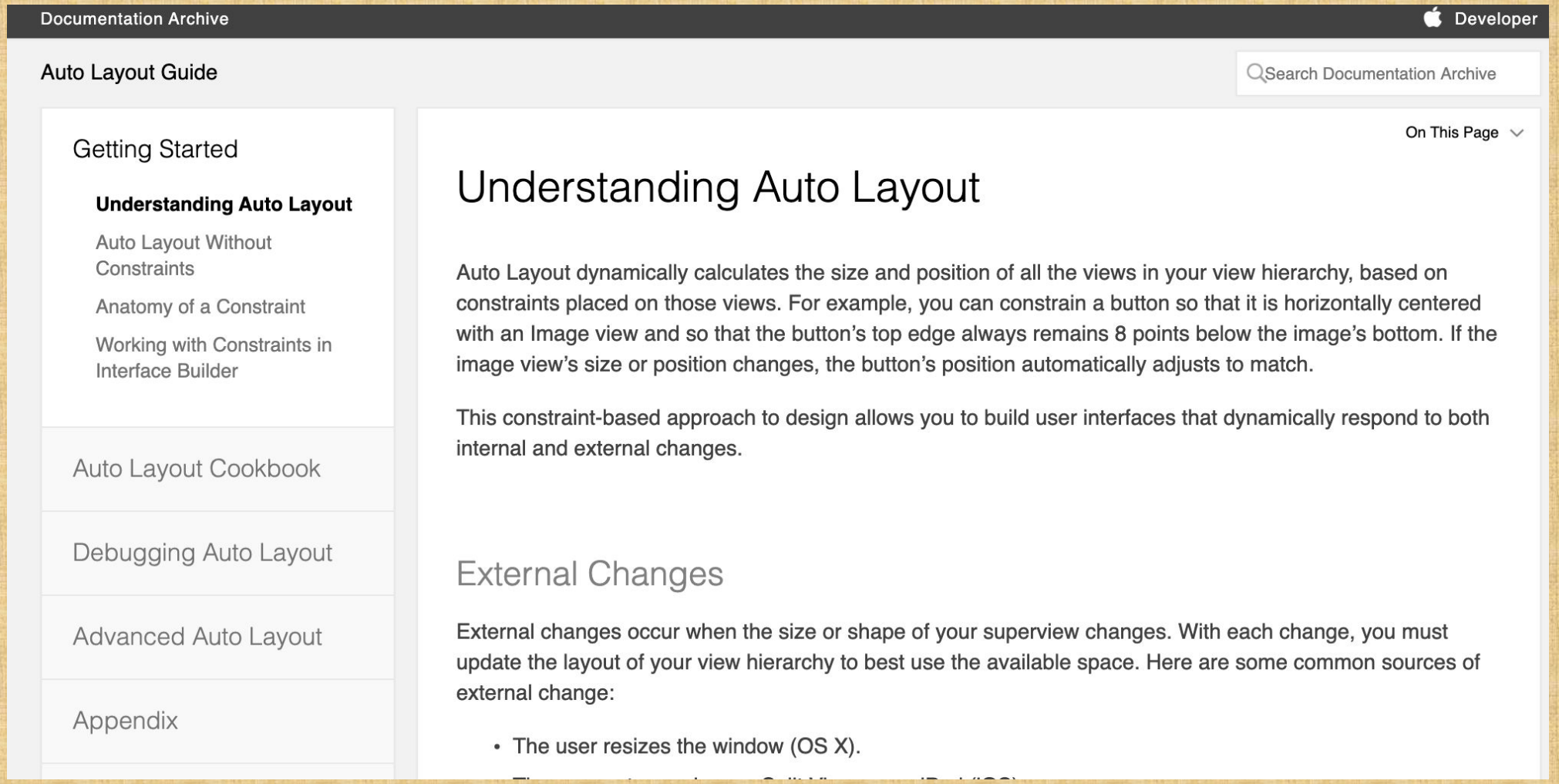
Post Constraints - Landscape





Read more

Apple Cookbook for AutoLayout - Click Here



Parting Notes

Exercise:

Let’s take our Mortgage calculator and do the following:

- Lock in Portrait Only
- Add Scroll View to Mortgage Calculator
- Use Auto layout and constraints to fix display on multiple form factors