

AutoLayout

Problem to solve:

Laying out views when there are changes in:

- Screen size
- Rotations
- Subview content
- Internationalization
- Dynamic Type
- Size class

(20, 20)

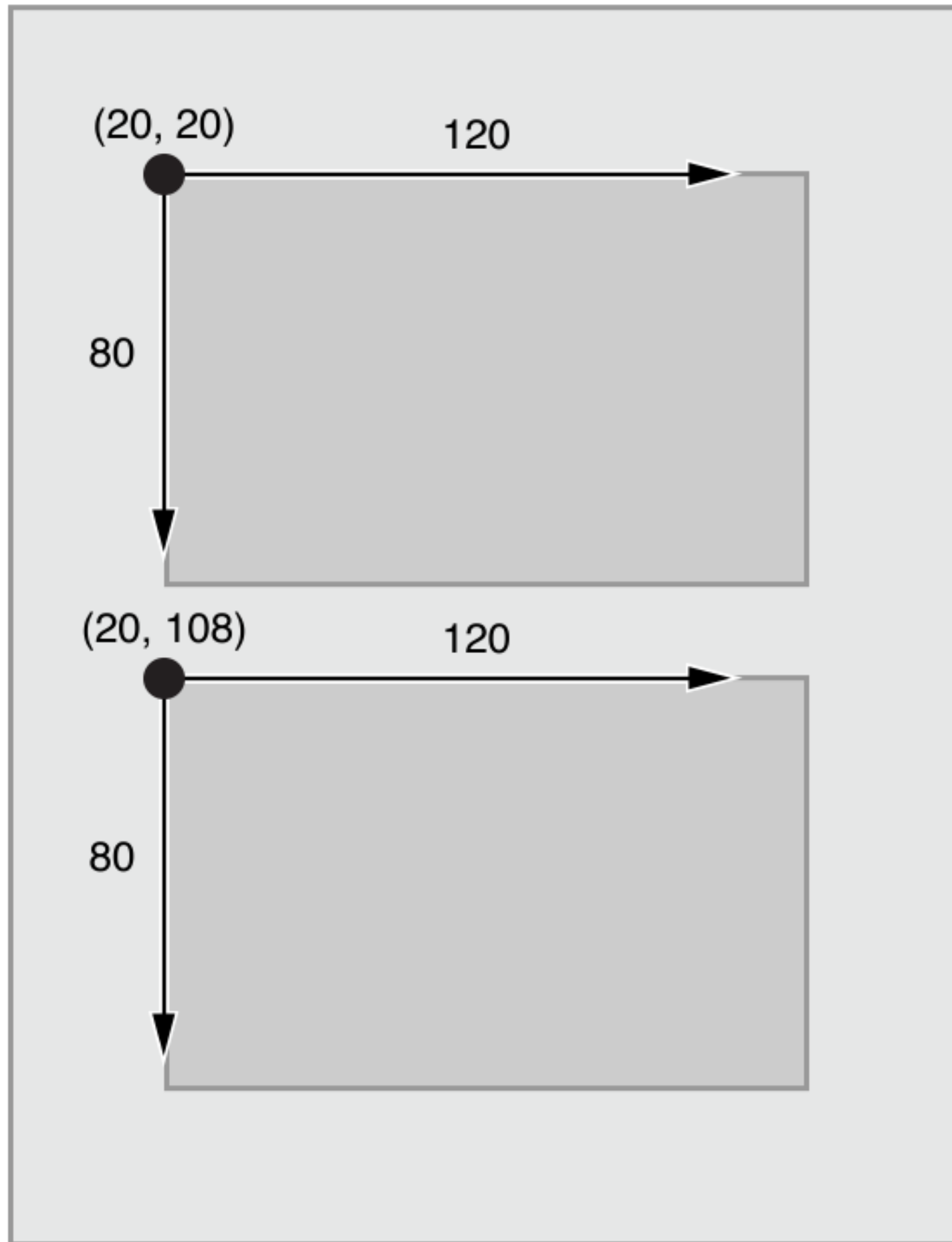
120

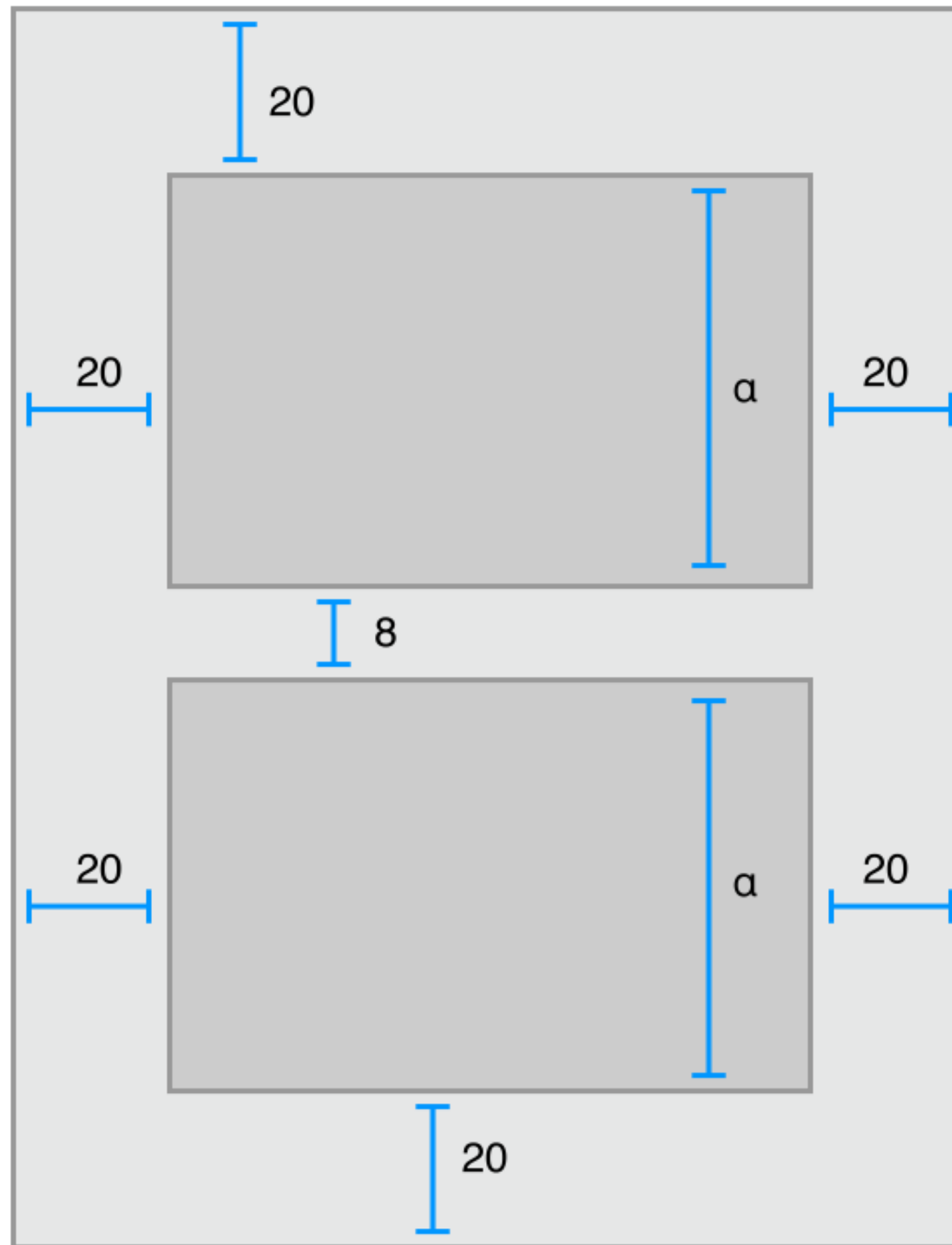
80

(20, 108)

120

80





The logic used to design a set of constraints to create specific behaviors is very different from the logic used to write procedural or object-oriented code.¹

¹ Understanding Auto Layout. developer.apple.com.

Let's back up a bit...

$$\begin{cases} x + y + z = 0 \\ 2x - 4y + 9z = 10 \\ -x + y - 2z = -3 \end{cases}$$

"...the **constraint**-based system"

"...the ~~constraint~~ **equation**-based system"

**AutoLayout is just an equation system
where the variables are the view's frames**

Equation == Constraint

A constraint is just an equation that expresses on an axis either:

- The distance between two sibling views
- The size of a view

If we think about it:

```
view.leadingAnchor.constraint(  
    equalTo: superView.leadingAnchor,  
    constant: 10  
)
```

Can be translated to²:

view.left == superView.left + 10

²Cartography.

How many equations do we need?

Each view has to have a defined frame, which means:

- *origin*: x, y
- *size*: width, height

You add enough constraints until the layout system can solve **all 4 variables for every view.**


$$\begin{cases} x + y + z = 0 \\ 2x - 4y + 9z = 10 \\ -x + y - 2z = -3 \end{cases}$$


$$\begin{cases} x + y + z = 0 \\ 2x - 4y + 9z = 10 \end{cases}$$



Buildtime

Runtime (1)

▼  GifWallet - 78791 1 issue

▼  Layout Issues

 Position is ambiguous for UIView.

$$\begin{cases} x + y + z = 0 \\ 2x - 4y + 9z = 10 \\ -x + y - 2z = -3 \\ 5x - 2y + z = 5 \end{cases}$$

Unable to simultaneously satisfy constraints
Probably at least one of the constraints in the
following list is one you don't want.

Will attempt to recover by breaking constraint
<NSLayoutConstraint:0x698e70 UIView: 0x698e43.width == 30 (active)>

How to add constraints (responsibly):

1. A `UIView` should provide constraints for its subviews positions
2. A `UIView` should provide its size:
 - Using width and height constraints
 - Using `intrinsicContentSize`

How to create constraints:

```
let widthConstraint = redView.widthAnchor.constraint(equalToConstant: 30)
let heightConstraint = redView.heightAnchor.constraint(equalToConstant: 30)
```

And then:

```
NSLayoutConstraint.activate([
    widthConstraint,
    heightConstraint
])
```

What happens next?

Layout cycle

Adding a constraint will indirectly call `setNeedsLayout` to a view, triggering a layout pass where the layout system will:

- Finds the top-most view (normally visible's `UIViewController`'s view)
- Resolve the top-most view hierarchy using the screen size.
 - Positions and sizes for the top-most view's subviews will be resolved.
- The layout system will recursively resolve the subviews hierarchy based on the size of all the subviews.

Layout cycle: customization

The layout system will call:

- viewWillLayoutSubviews on UIViewController
- layoutSubviews on UIView

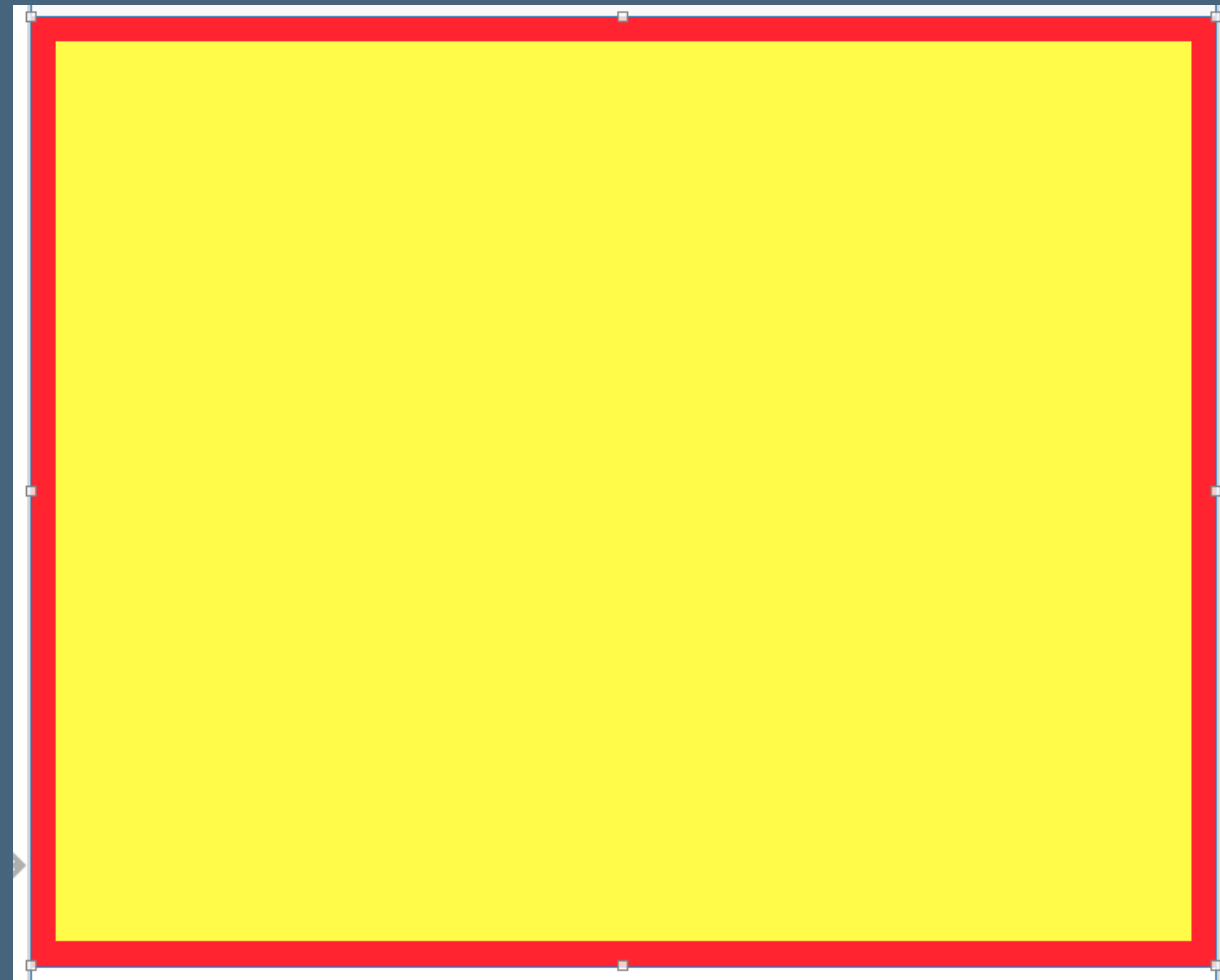
The resolved frame for the current view will be passed as an argument for customization.

Don't call setNeedsLayout during these methods.

Demo

Layout Guides

Defines a rectangular region where layout can occur safely in their owning view's coordinate system.



Layout Guides

```
let layoutGuide: UILayoutGuide = ...
let constraints = [
    label.leadingAnchor.constraint(equalTo: layoutGuide.leadingAnchor),
    label.trailingAnchor.constraint(equalTo: layoutGuide.trailingAnchor),
    label.topAnchor.constraint(equalTo: layoutGuide.topAnchor),
    label.bottomAnchor.constraint(equalTo: layoutGuide.bottomAnchor)
]
NSLayoutConstraint.activate(constraints)
```

Layout Guides

There are three types:

- Margin.
- SafeArea.
- Custom.

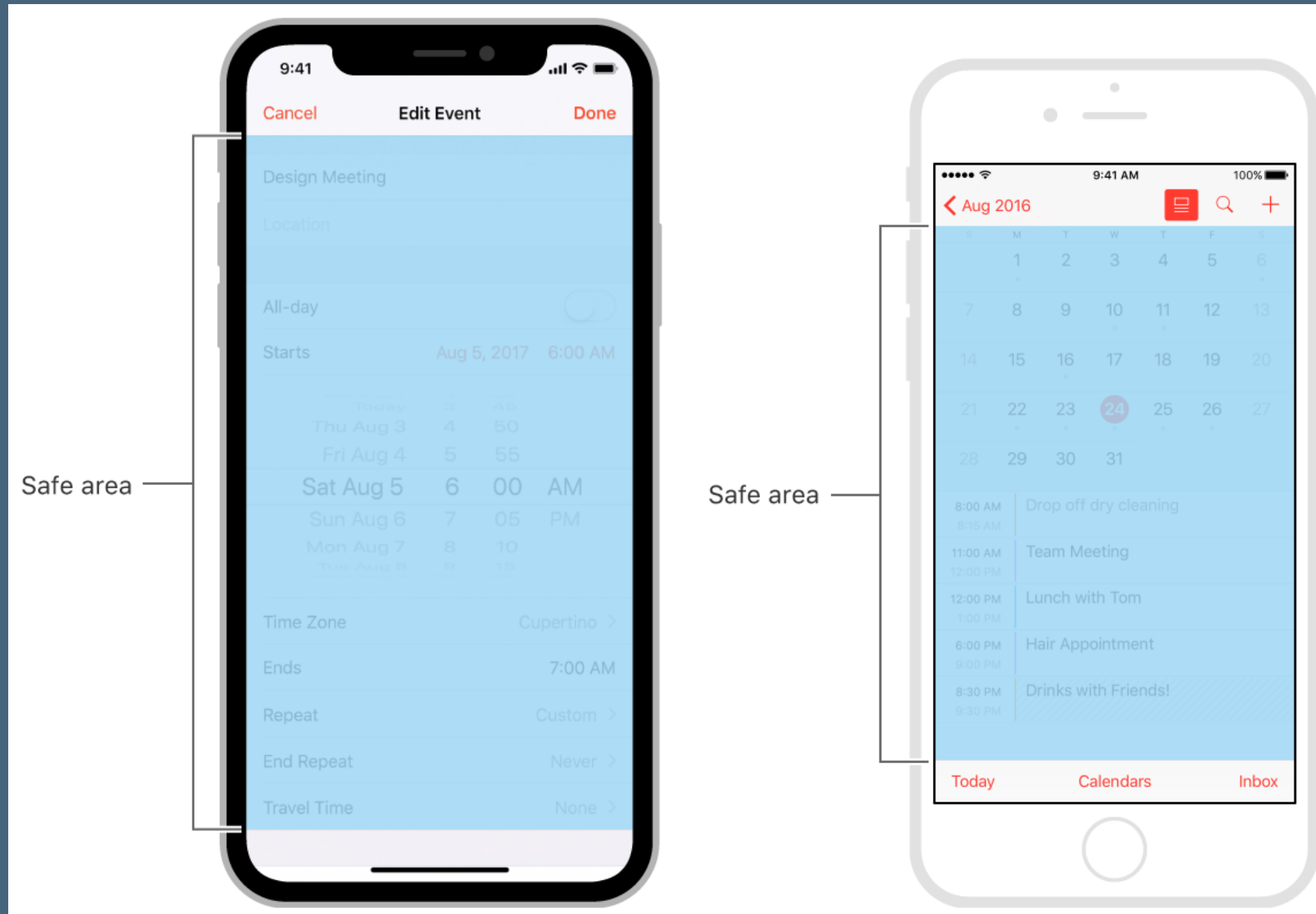
Layout **Margins** Guides

Every UIView has one to reflect the `layoutMargins` property.

```
self.layoutMargins = UIEdgeInsets(top: 5, left: 5, bottom: 5, right: 5)
let constraints = [
    label.leadingAnchor.constraint(equalTo: self.layoutMarginsGuide.leadingAnchor),
    label.trailingAnchor.constraint(equalTo: self.layoutMarginsGuide.trailingAnchor),
    label.topAnchor.constraint(equalTo: self.layoutMarginsGuide.topAnchor),
    label.bottomAnchor.constraint(equalTo: self.layoutMarginsGuide.bottomAnchor)
]
NSLayoutConstraint.activate(constraints)
```

Note: you can't modify a UIViewController's view's margins.

Safe Area Layout Guide



Safe Area Layout Guide

Prevents showing content underneath:

- UINavigationController
- UITabBar
- UIStatusBar
- iPhone X home indicator.

✅ Automatically set for us on UIViewController

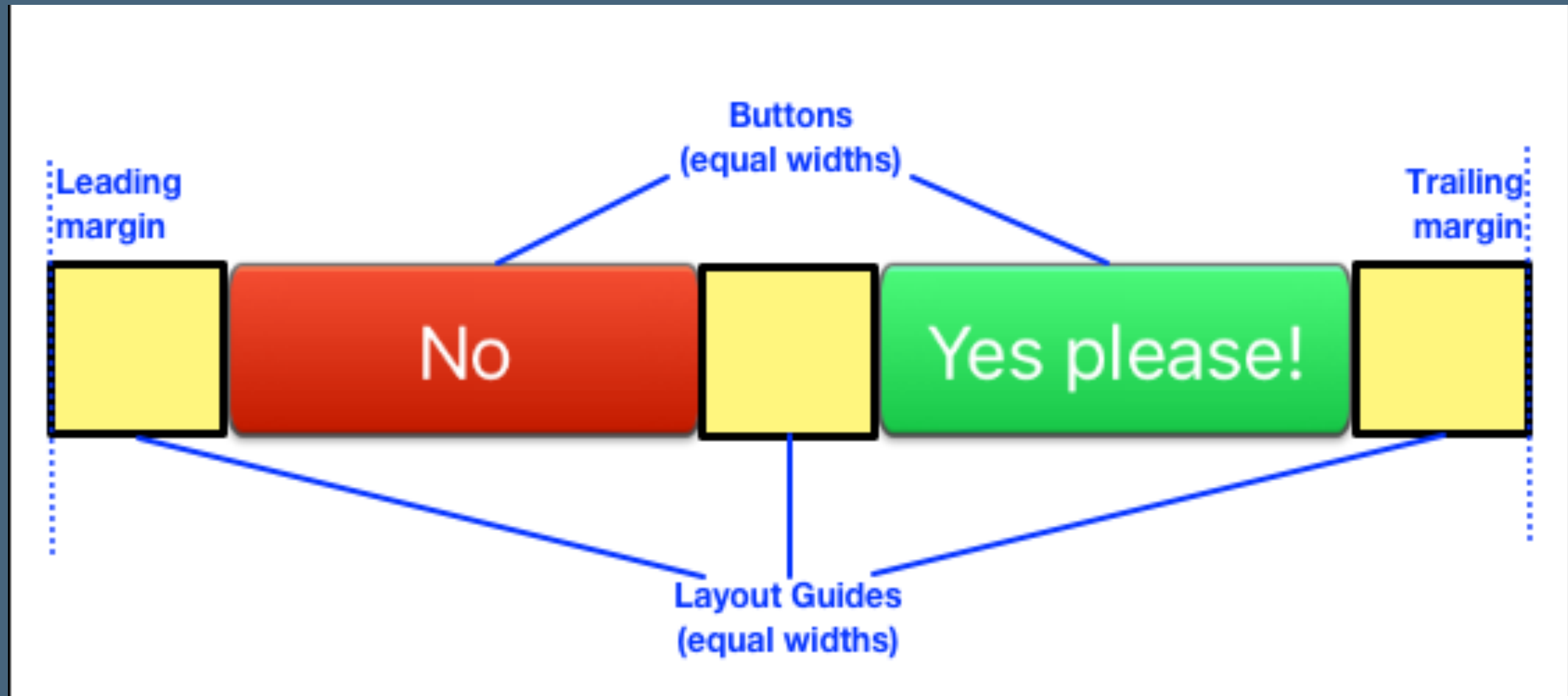
👁️ Can be modified with `additionalSafeAreaInsets`

Custom guides:

No

Yes please!

Custom guides³:



³ Goodbye Spacer Views Hello Layout Guides. useyourloaf.com.

Demo

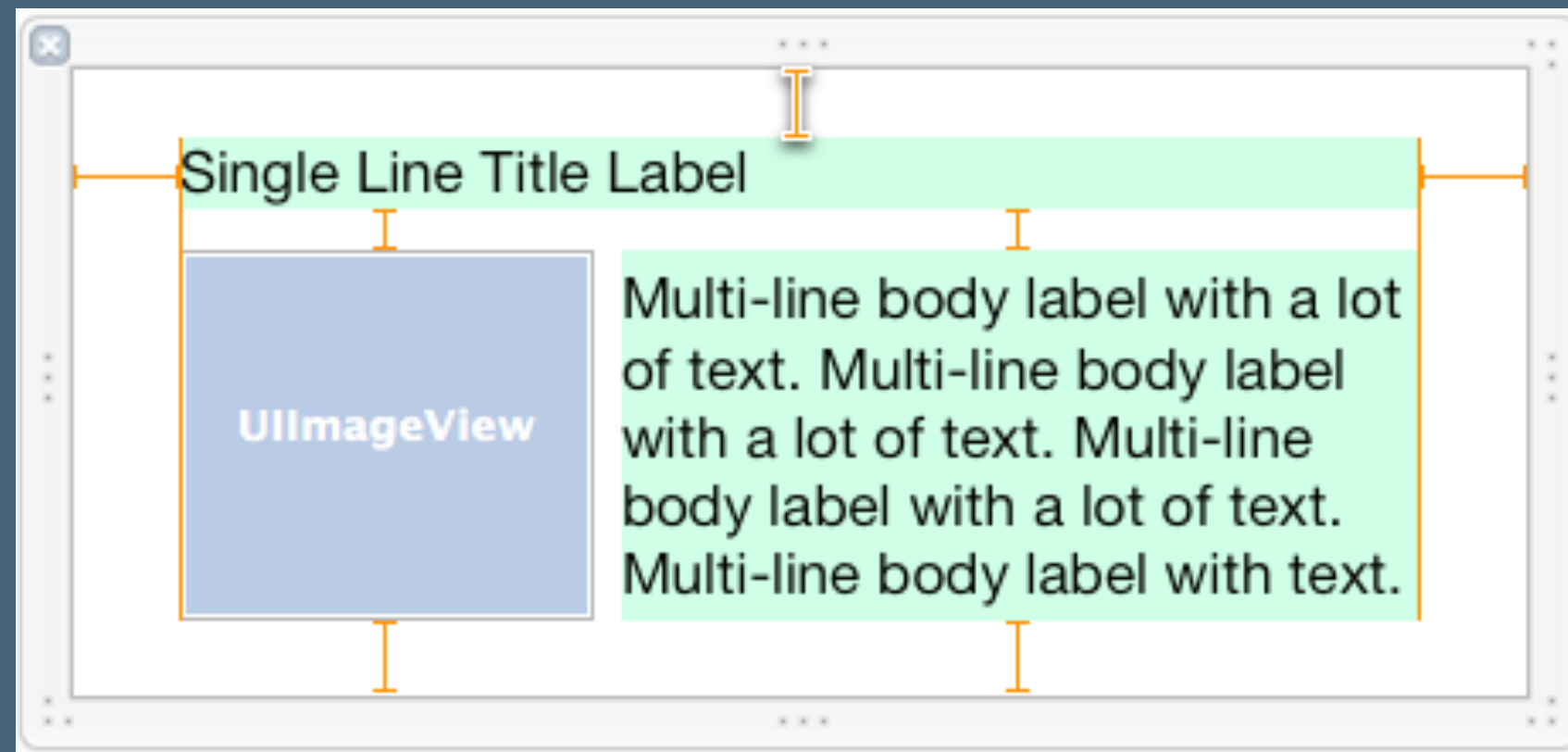
Best practices:

View layout




- From top to bottom on the view hierarchy.
- Create constraints and activate them all at once using `NSLayoutConstraint.activate()`.
- Don't create more constraints than you need.
- Rely on `intrinsicContentSize` for sizes when possible.

intrinsicContentSize

- Your custom view's should not override this method
 - Instead, create constraints that will make the system infer it for you.



UILabel.intrinsicContentSize

-  *Never* constraint the width/height
-  Set `numberOfLines` to 0 and leading and trailing constraints instead of width.
-  Also, set `contentHuggingPriority` and `contentCompressionResistancePriority` appropriately.

`UIImageView.intrinsicContentSize`

- It will default to the image's size
- Create a constraint for:
 - The `aspectRatio`
 - The max allowed `aspectRatio`
 - The width or height.

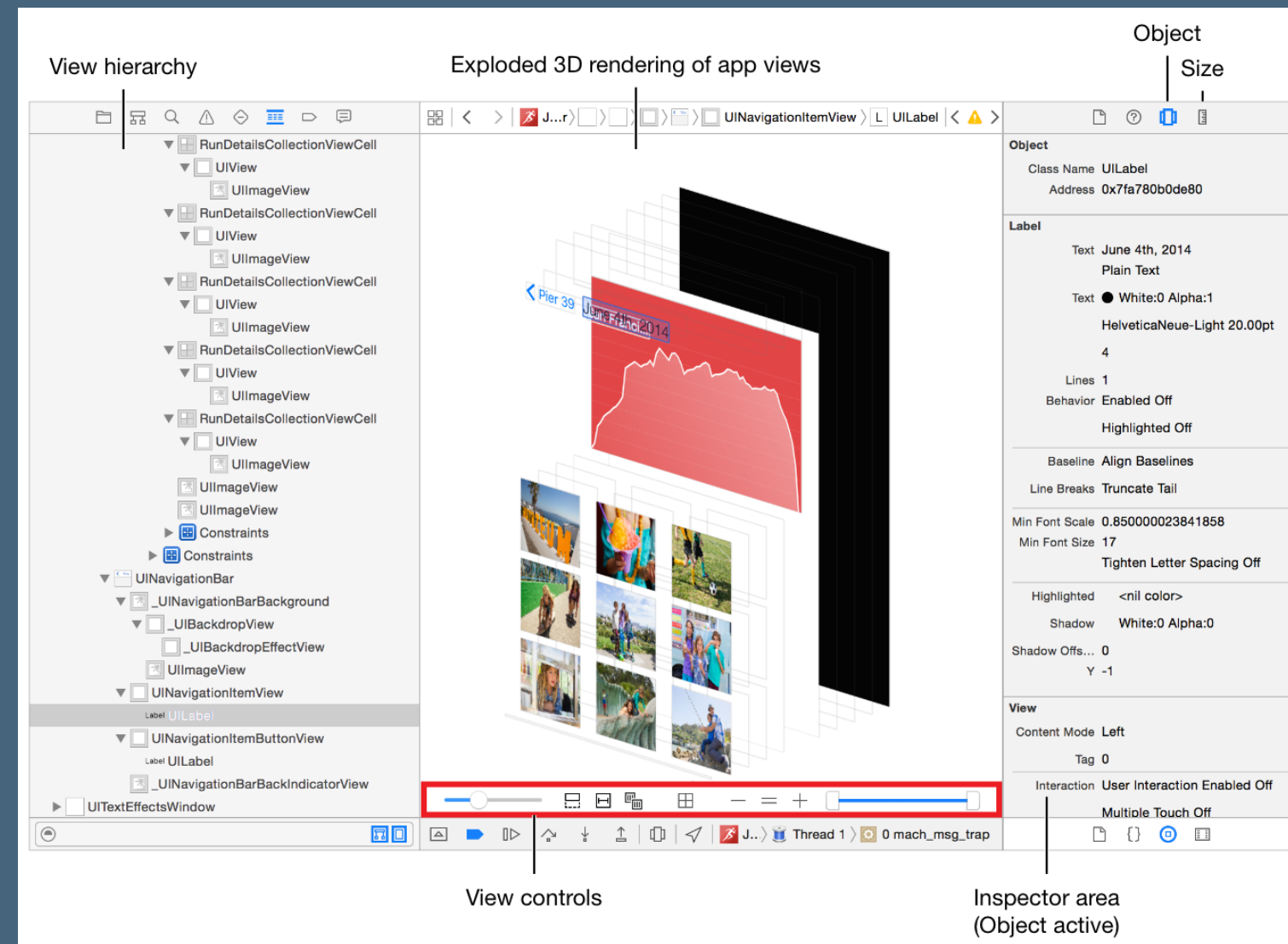
UIStackView

- Use `UIStackView` as much as possible.
- Remember that they don't have an `intrinsicContentSize`.
 - However, if the system can infer a width (vertical) or height (horizontal) for this `stackView`, then it uses the **fitting size** for this `stackView`:

```
let size: CGSize = stackView.systemLayoutSizeFitting(
    CGSize(width: 100, height: 0),
    withHorizontalFittingPriority: .required,
    verticalFittingPriority: .fittingSizeLevel
)
```

Debugging

— Xcode's Visual Hierarchy debugger is your friend.



Debugging

- Remember to turn off `translatesAutoresizingMaskIntoConstraints` .
- If a constraint is being broken by the engine, and the layout looks correct:
 - The engine did the right thing now, but this doesn't mean that it will do the same thing on other OS versions.
 - Set that constraint's priority lower to optimize the performance.

Debugging

— Content sizes are also represented in the layout engine

