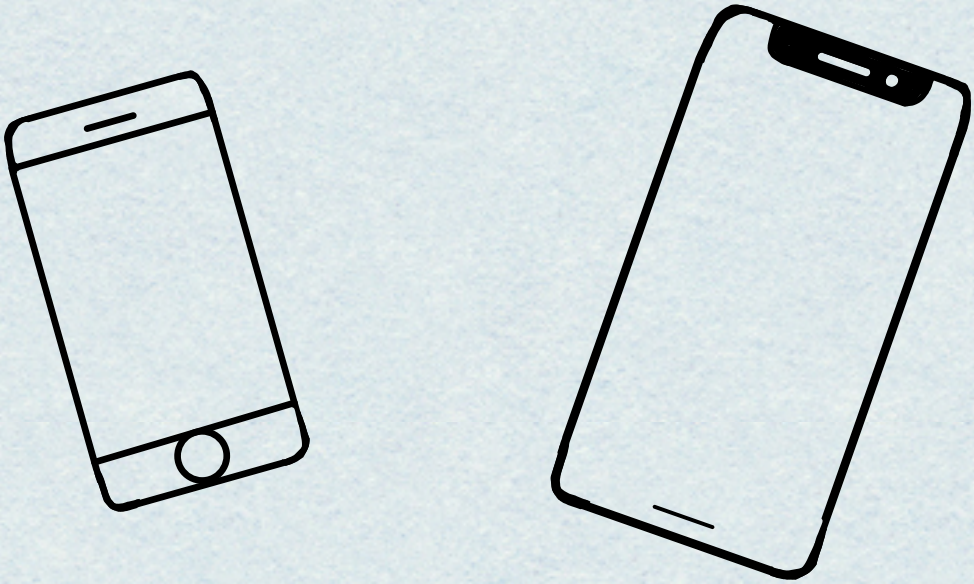
AUTO LAYOUT

AND

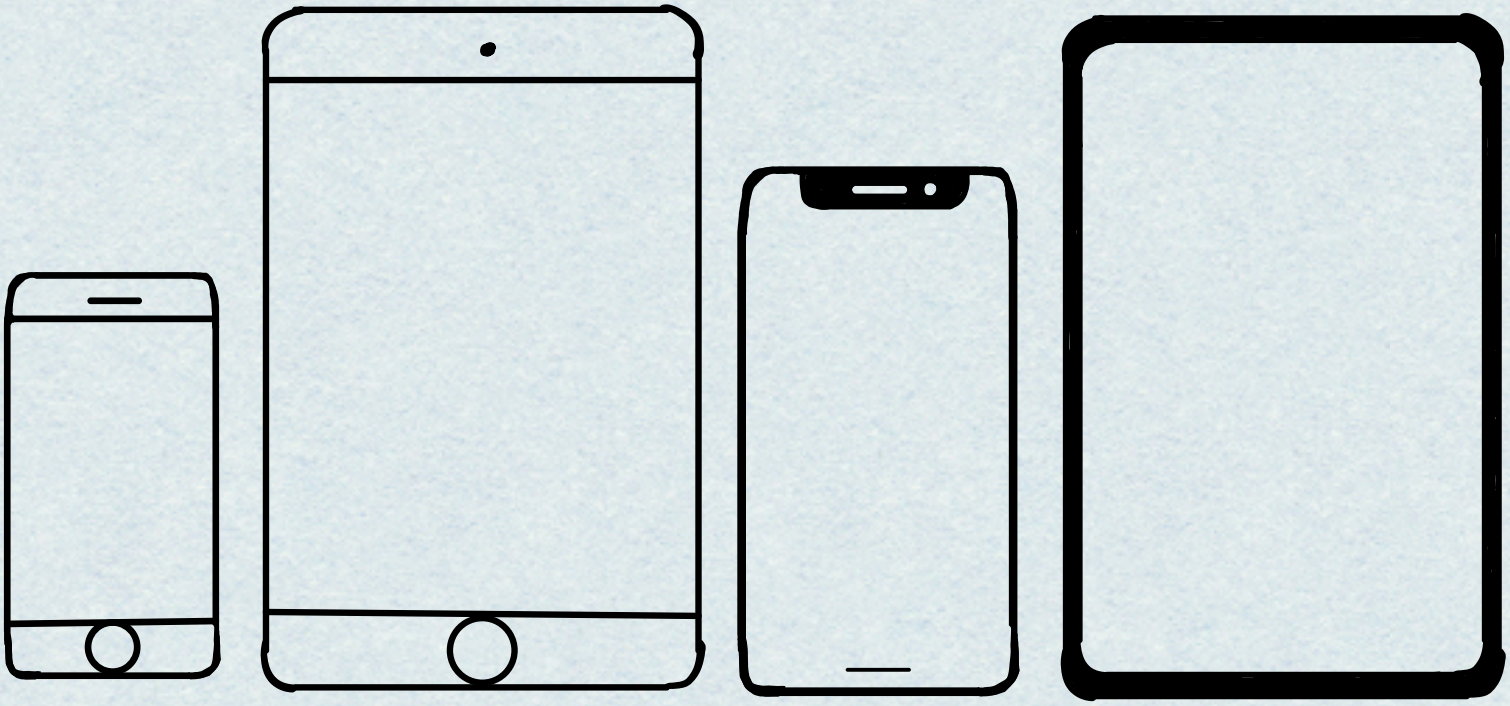
CONSTRAINTS

An Overview



By Claire

#MadeInNotability



Why does Auto Layout matter?

When laying out your UI, you have to take <u>a lot</u> of things into account:

- Device type / screen size
- Device rotation
- Split screen mode (on iPad)
- Different languages with different string lengths
- User interaction causes different content to display

<u>Bottom Line</u>:

We need a dynamic way to display views.

AUTO LAYOUT

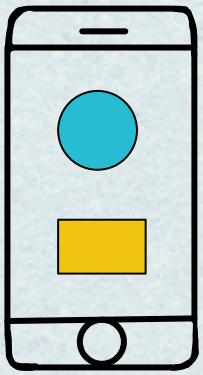
Calculates the **size** and **location** of all the views in your view hierarchy based on <u>**constraints**</u>.

Auto Layout

+

Constraints

=



Your View

CONSTRAINT

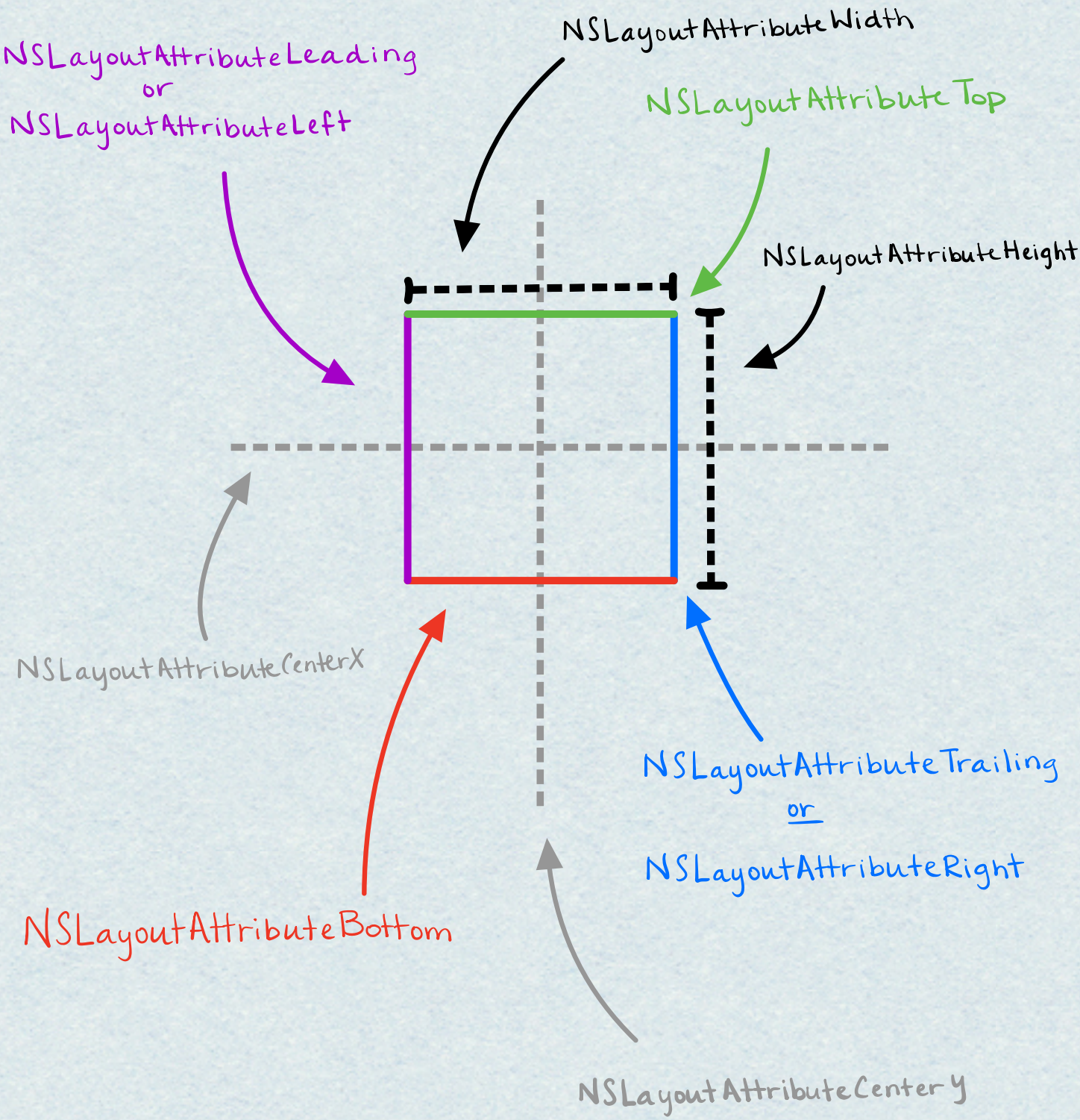
Noun [kuhn-**streynt**]

1. Limitation or constriction
2. Repression of natural feelings and impulses 😞
3. **The relationship between two user interface objects that must be satisfied by constraint-based layout system**

Elements:

1. firstItem – the first view participating in the constraint
2. firstAttribute – the value of firstItem that will be constrained
3. relation – the relationship between the two attributes in the constraint
4. secondItem – the second view participating in the constraint
5. secondAttribute – the value of secondItem that will be constrained
6. multiplier – multiplier applied to secondAttribute
7. constant – constant applied to the multiplied secondAttribute
8. priority – priority of constraint (the higher the number the less likely AutoLayout will break it)
9. identifier – a unique id to access the constraint

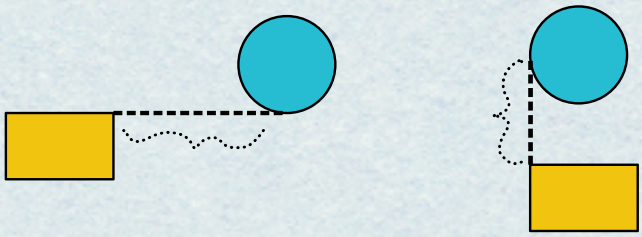
ATTRIBUTES


NSLayoutAttributeWidth
NSLayoutAttributeLeading
or
NSLayoutAttributeLeft
NSLayoutAttributeTop
NSLayoutAttributeHeight
NSLayoutAttributeCenterX
NSLayoutAttributeTrailing
or
NSLayoutAttributeRight
NSLayoutAttributeBottom
NSLayoutAttributeCenterY

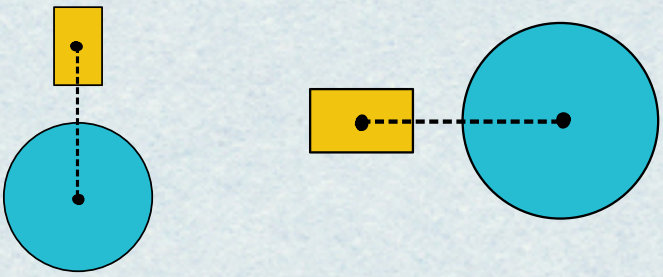
Types of

CONSTRAINTS

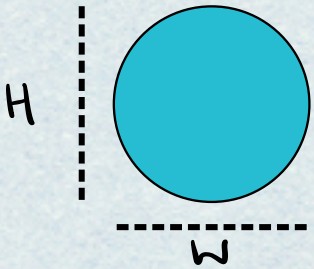
Spacing:



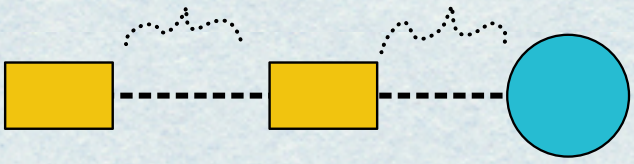
Alignment:



Size:

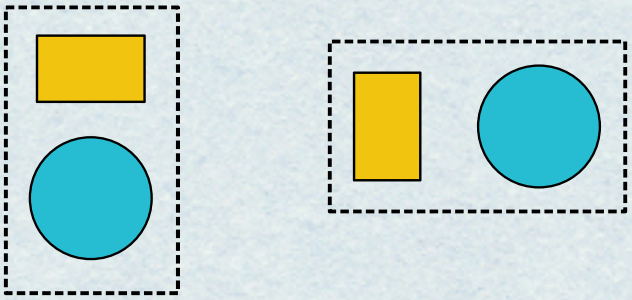


Distribution:



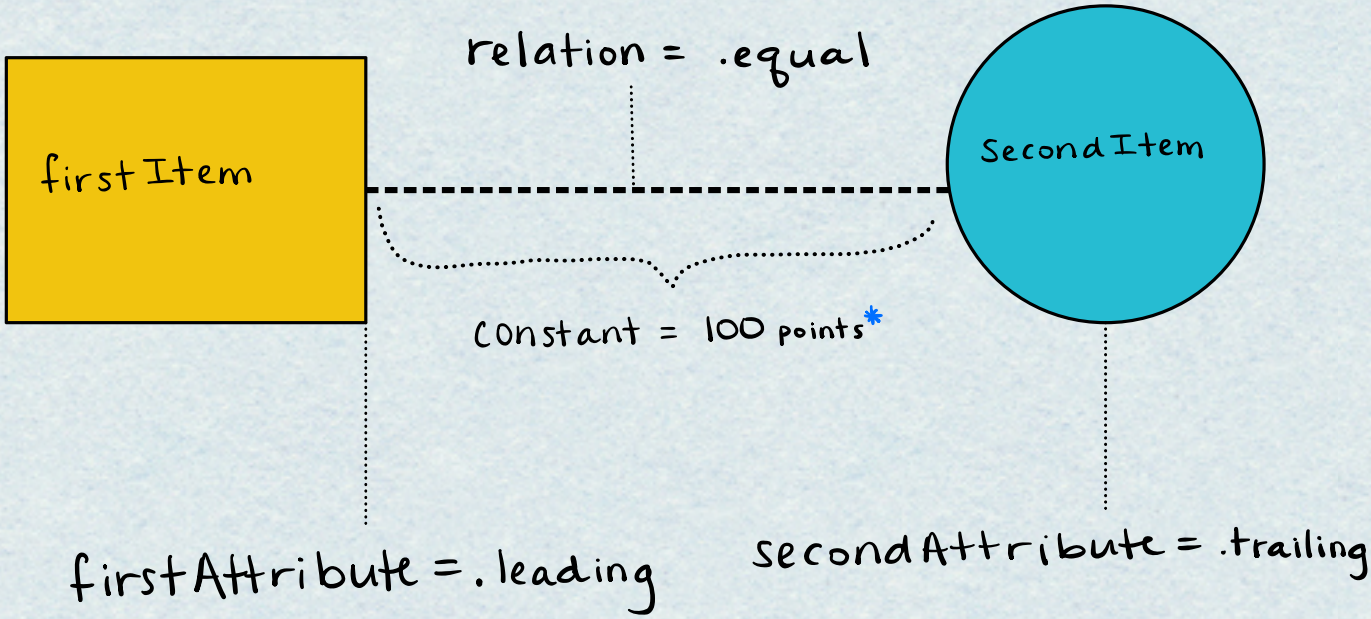
Axis:

(Only for UIStackViews)



CONSTRAINT

EXAMPLE!

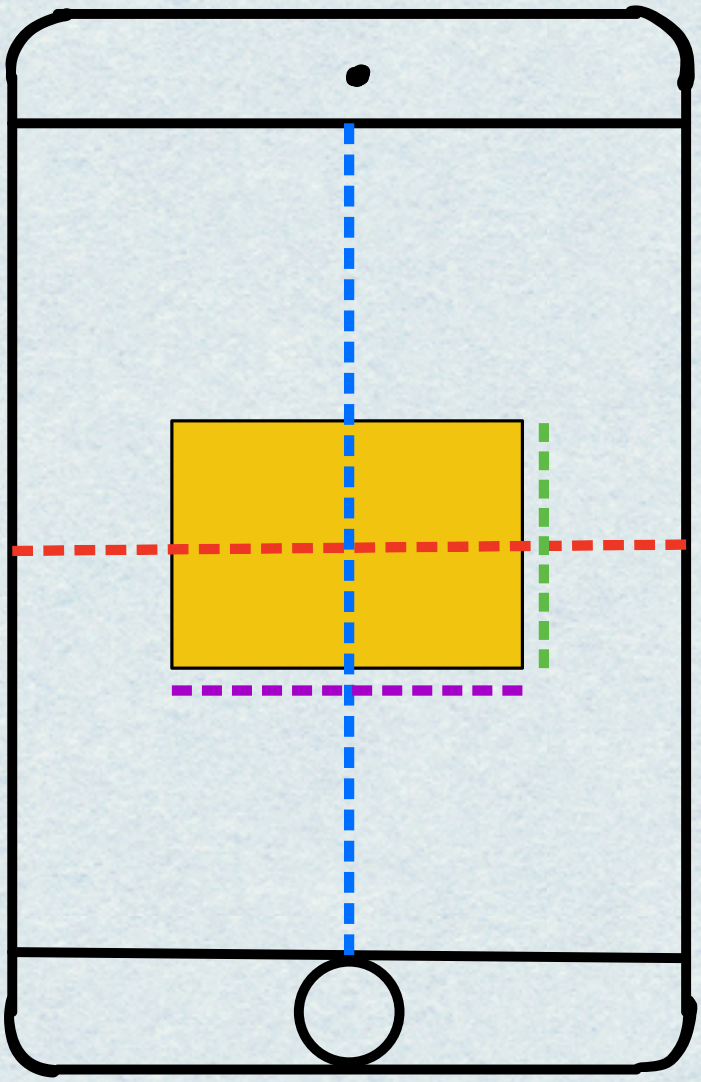


multiplier = 1

* points are a unit of measurement for distance on Apple devices which provide a fixed frame of reference for drawing, regardless of device or screen resolution.

CONSTRAINT

EXAMPLE 2!



1. firstItem = superView
   firstAttribute = centerX
   relation = equal
   secondItem = yellow square
   secondAttribute = centerX
   constant = 0
   multiplier = 1

2. firstItem = superView
   firstAttribute = centerY
   relation = equal
   secondItem = yellow square
   secondAttribute = centerY
   constant = 0
   multiplier = 1

3. firstItem = yellow square
   firstAttribute = height
   relation = equal
   constant = 250
   multiplier = 1

4. firstItem = yellow square
   firstAttribute = width
   relation = equal
   constant = 400
   multiplier = 1

IMPLEMENTATIONS of CONSTRAINTS

#1 NSLayout Constraints

The most simple way to make constraints

```
if (self.universalSizeClass == NIUniversalSizeClassLarge) {
    [arrayToAddConstraintsTo addObject:[NSLayoutConstraint constraintWithItem:self.outerSpacerTop
                                                                     attribute:NSLayoutAttributeTop
                                                                     relatedBy:NSLayoutRelationEqual
                                                                        toItem:self.view
                                                                     attribute:NSLayoutAttributeTop
                                                                    multiplier:1.0
                                                                      constant:0.0]];

    NSLayoutConstraint *keyboardConstraint = [NSLayoutConstraint constraintWithItem:outerSpacerBottom
                                                                           attribute:NSLayoutAttributeBottom
                                                                           relatedBy:NSLayoutRelationEqual
                                                                              toItem:self.view
                                                                           attribute:NSLayoutAttributeBottom
                                                                          multiplier:1
                                                                            constant:-self.keyboardHeight];
    keyboardConstraint.priority = 950;
    [arrayToAddConstraintsTo addObject:keyboardConstraint];
```

Pros:

- Nothing is hidden, there are no constraints being made behind the scenes

Cons:

- It's slow and cumbersome to make each individual constraint in 7 lines of code

#2 Visual Constraint

```
1 UIView *insetContentView = _insetContentView;
2 [self.contentView addSubview:insetContentView];
3 NSArray *constraints = [NSLayoutConstraint constraintsWithVisualFormats:@[@"H:|-(==inset)-[insetContentView]-(==inset)-|",
                                                                            @"V:|-(==inset)-[insetContentView]-(==inset)-|"]
                                                                   options:0
                                                                   metrics:@{@"inset": @(kGLNBCollectionViewCellContentInset)}
                                                                     views:NSDictionaryOfVariableBindings(insetContentView)];
4 [self.contentView addConstraints:constraints];
```

1. Create the view
2. Add the view as a subview
3. Create the constraints
4. Add the constraint to the superview

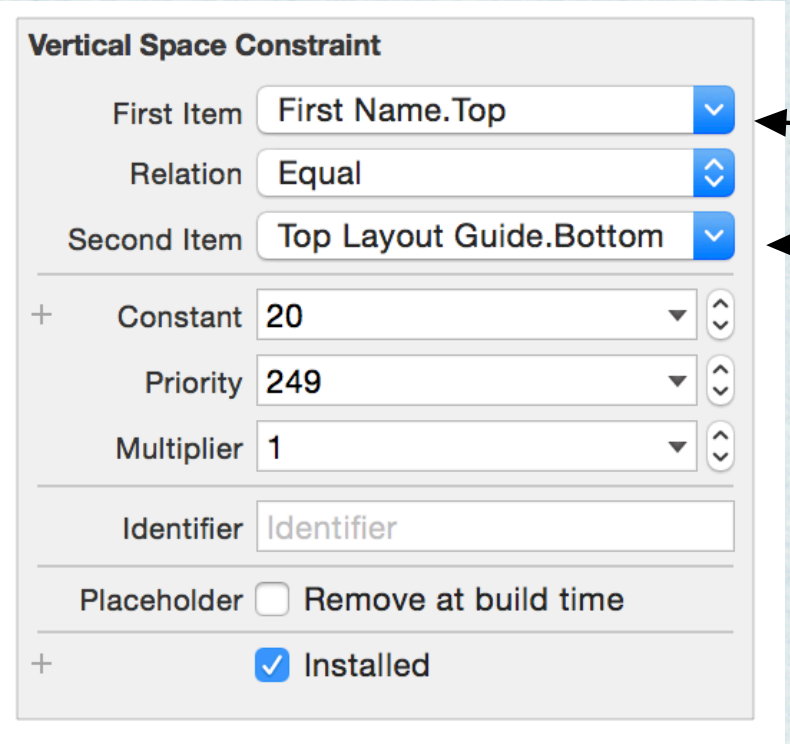
Pros:

- Make more constraints in fewer lines of code

Cons:

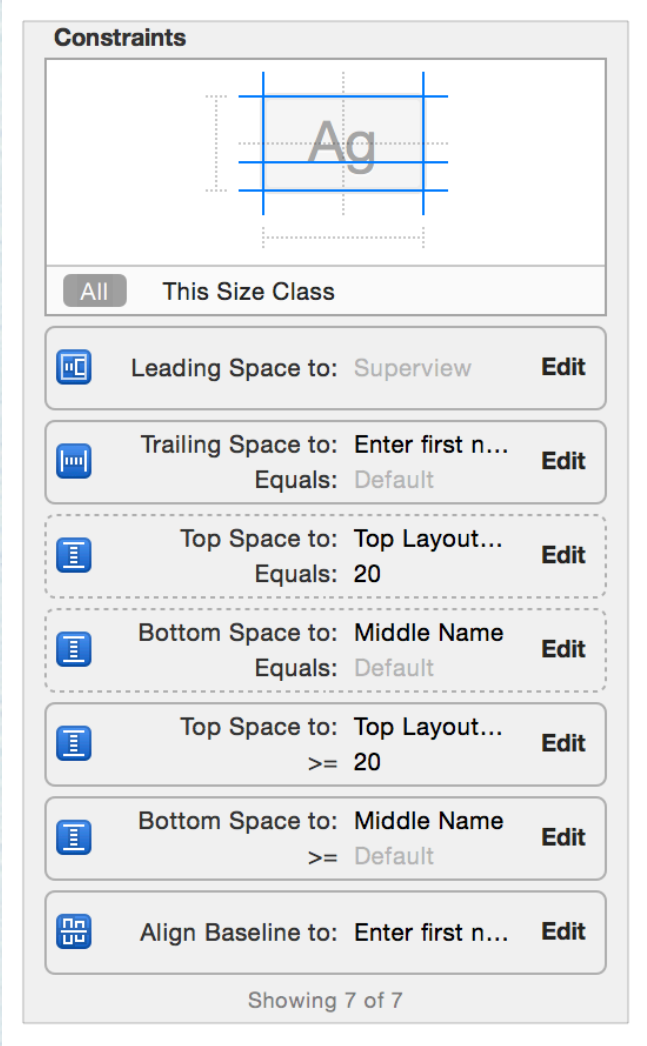
- The syntax is easy to mess up and there are no compiler errors to help guide you when creating the strings

#3 Interface Builder



First Item
&
First Attribute

Second Item
&
Second Attribute



Pros

- Interface Builder is very visual and easy to understand
- The UI helps guide you when making constraints

Cons

- Bad when working with a team and you get crazy merge conflicts
- All constraint code is behind the scenes – not as much control

#4 Pure Layout

A third party framework that provides an easy-to-use API to make constraints programmatically

```
- autoSetContent(CompressionResistance|Hugging)PriorityForAxis:
- autoCenterInSuperview(Margins) // Margins variant iOS 8.0+ only
- autoAlignAxisToSuperview(Margin)Axis: // Margin variant iOS 8.0+ only
- autoPinEdgeToSuperview(Edge:|Margin:)(withInset:) // Margin variant iOS 8.0+ only
- autoPinEdgesToSuperview(Edges|Margins)(WithInsets:)(excludingEdge:) // Margins variant iOS 8.0+
- autoPinEdge:toEdge:ofView:(withOffset:)
- autoAlignAxis:toSameAxisOfView:(withOffset:|withMultiplier:)
- autoMatchDimension:toDimension:ofView:(withOffset:|withMultiplier:)
- autoSetDimension(s)ToSize:
- autoConstrainAttribute:toAttribute:ofView:(withOffset:|withMultiplier:)
- autoPinTo(Top|Bottom)LayoutGuideOfViewController:withInset: // iOS only
- autoPinEdgeToSuperviewSafeArea: // iOS 11.0+ only
- autoPinEdgeToSuperviewSafeArea:withInset: // iOS 11.0+ only
```

Pros:

- Make many constraints with one line of code

Cons:

- Limited to the API

OTHER THINGS

① Conflicting Constraints

AutoLayout will break constraints until there are no conflicts (lower priority first). Remove any old constraints from an old one before adding new ones.

② Stack Views

Use UIStackViews or GLGroupViews to create complex layouts without manually making a lot of constraints.

③ Animations

Make layouts more dynamic by animating constraints.

④ Intrinsic Content Size

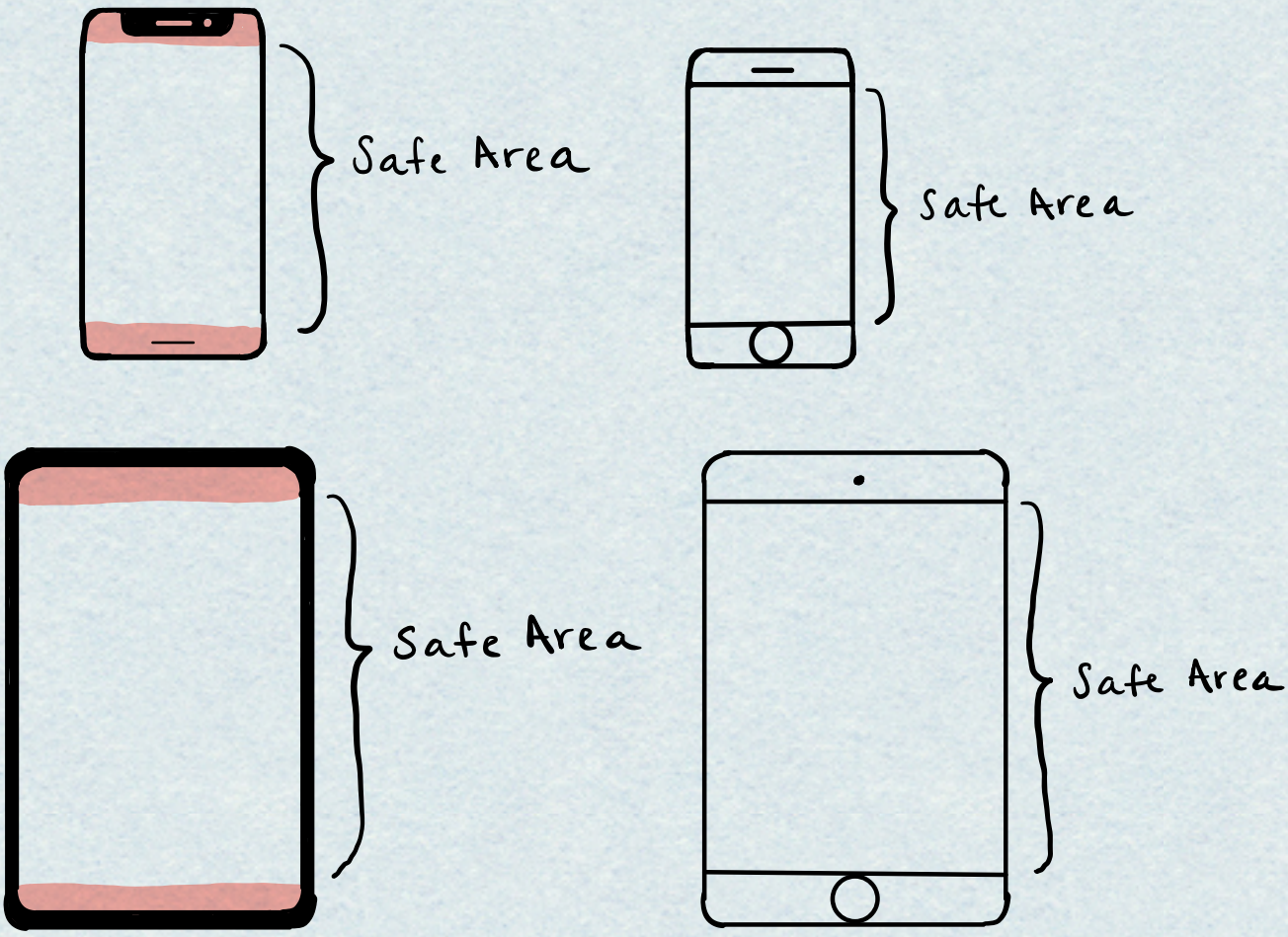
Some views have an intrinsic content size based on content rather than constraints:
Sliders, labels, buttons, switches, text fields, images

⑤ Autoresizing Mask

Creates constraints for a view's size and position on the screen. Do not add more constraints to the view when translatesAutoresizingMaskIntoConstraints is true.

⑥ Safe Area

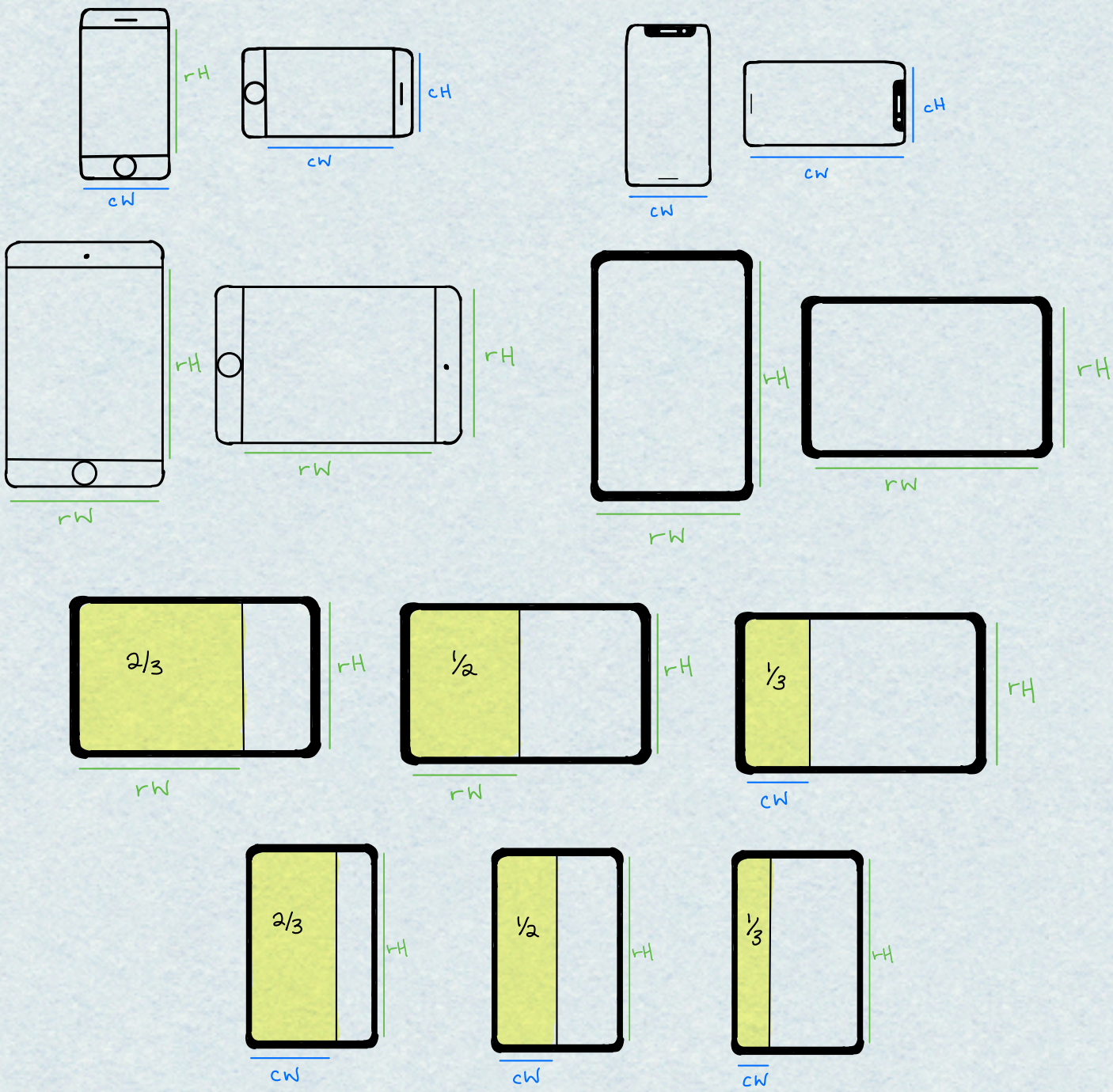
Devices that do not have a physical home button need UI to be above the digital "home button"

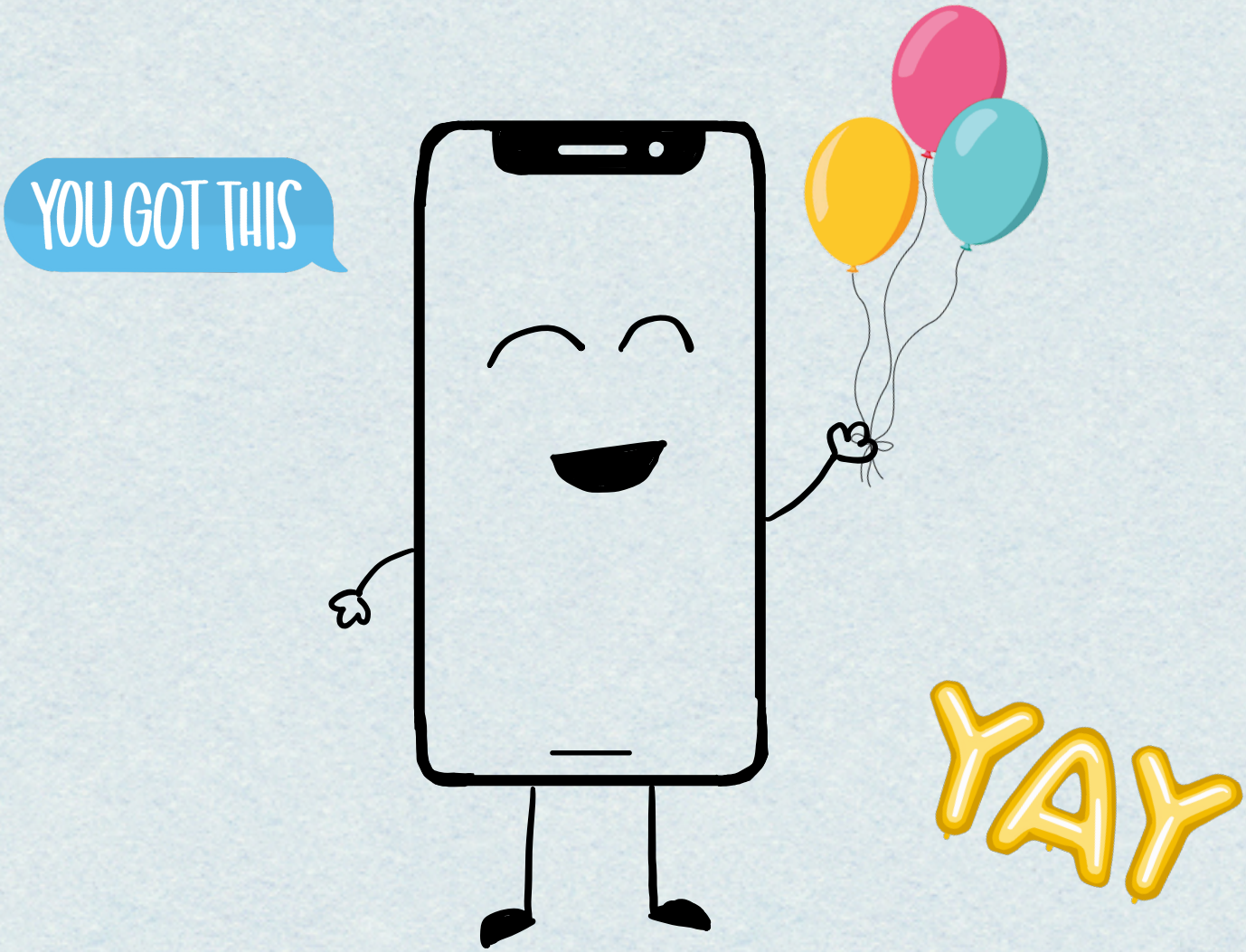


SIZE CLASSES

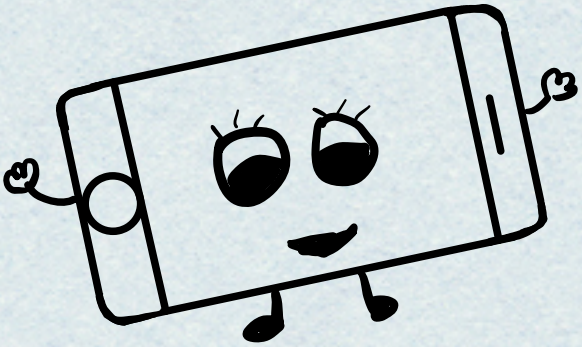
Size classes are traits that are applied to content areas based on size:

regular Width, regular Height, compact Width, compact Height





THE END.



Sources

https://developer.apple.com/library/archive/documentation/UserExperience/Conceptual/AutolayoutPG/index.html

https://www.dictionary.com/browse/constraint?s=t

https://developer.apple.com/documentation/uikit/nslayoutconstraint?language=objc

https://developer.apple.com/library/archive/documentation/WindowsViews/Conceptual/ViewPG_iPhoneOS/WindowsandViews/WindowsandViews.html#//apple_ref/doc/uid/TP40009503-CH2-SW15

https://developer.apple.com/documentation/uikit/nslayoutattribute

https://developer.apple.com/library/archive/documentation/UserExperience/Conceptual/AutolayoutPG/WorkingwithConstraintsinInterfaceBuidler.html

https://developer.apple.com/documentation/uikit/uiview/1622572-translatesautoresizingmaskintoco?language=objc

https://developer.apple.com/design/human-interface-guidelines/ios/visual-design/adaptivity-and-layout/