Auto Layout

Mobile Computing - iOS

Objectives

- Students will be able to:
    - explain the need for auto layout
    - create constraints using interface builder
    - diagnose constraint issues in interface builder
    - create constraints programmatically (maybe)

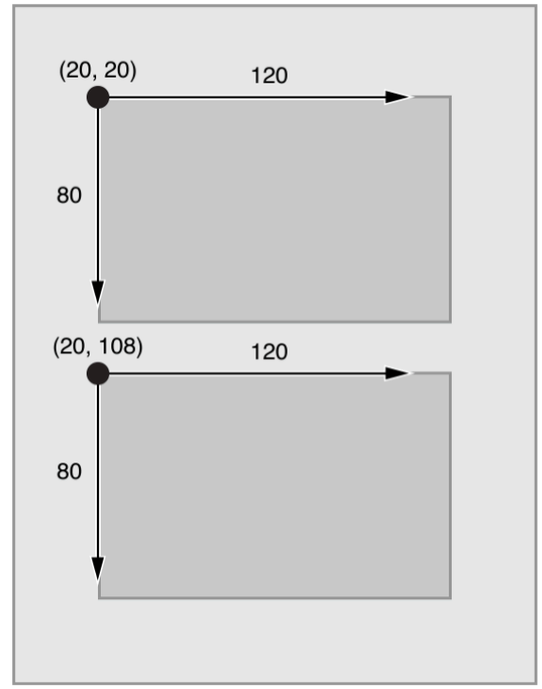
Frames

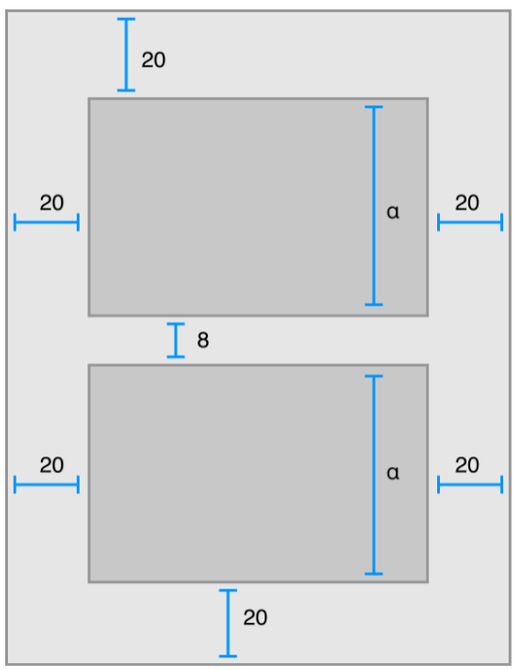
- A frame stores a UIView's origin and size. Every UIView (and its subclasses, like UIButton, UITextField, UILabel, etc.) has one.

- In ancient, halcyon days, you would define a frame explicitly (e.g., the bottom view's origin is (20,108), and its size is 120x80).

- In these modern times, you **specify constraints**, (e.g., the bottom view is 20 points from the container, 8 points below the top view, the same height as the top view, etc.), and then the **frames are calculated** so that they meet those constraints

- This allows us to define one layout that works for various sized devices, in various orientations.





Auto Layout

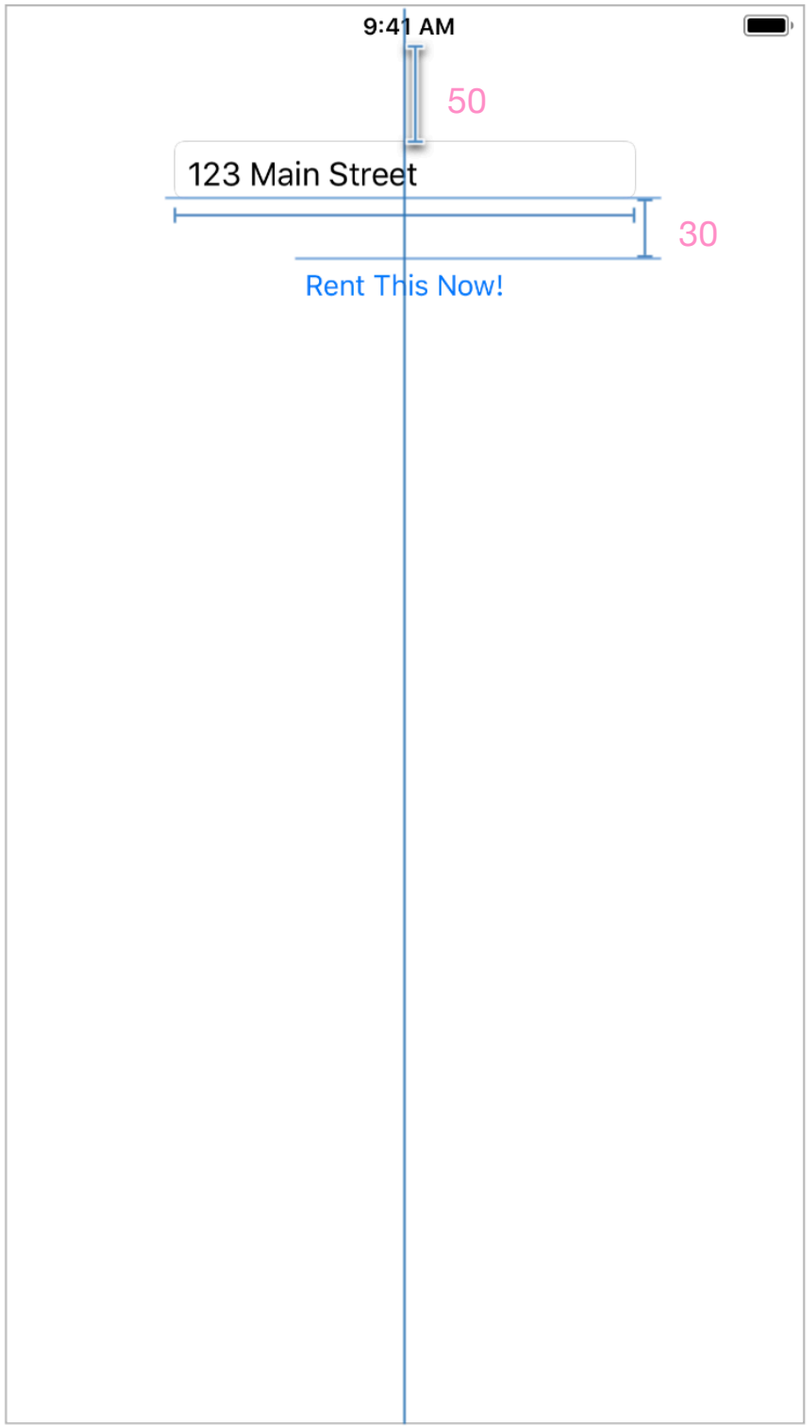
- Auto Layout calculates the **size** and **position** (aka the **frames**) of all your views in a view hierarchy based on **constraints** — relationships between views

- e.g., a button (Rent This Now!) might be horizontally centered relative to its containing view, and its top edge might be 30 pixels below a text field's bottom edge. Meanwhile, the text field might be 50 pixels below the top of the safe area (the screen), and also centered.

- On the basis of this, UIKit can calculate the size and position of the button.

    - if the text field moves, the button will do so as well, so that it continues to be 10 pixels below it (still horizontally centered)

    - if the device rotates, the button will still be horizontally centered, even if it is (in landscape) over a much larger area

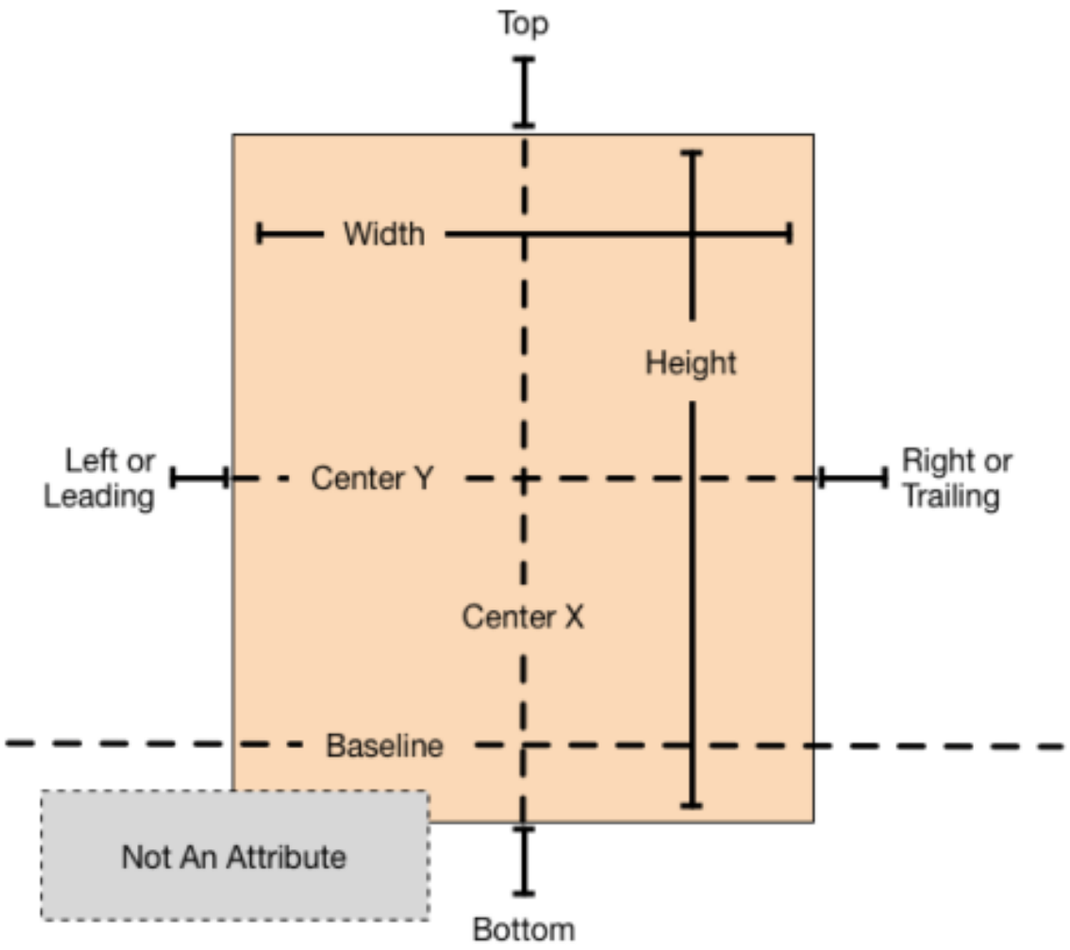


The Mathematics of Constraints

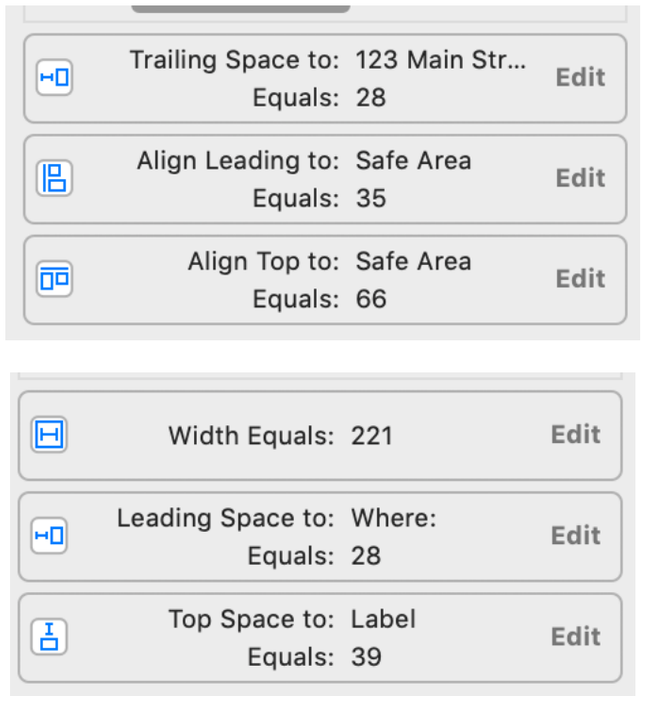
- Every constraint involves **attributes** — width, height, distance from its enclosing container (top, bottom, leading, trailing), and if it is centered

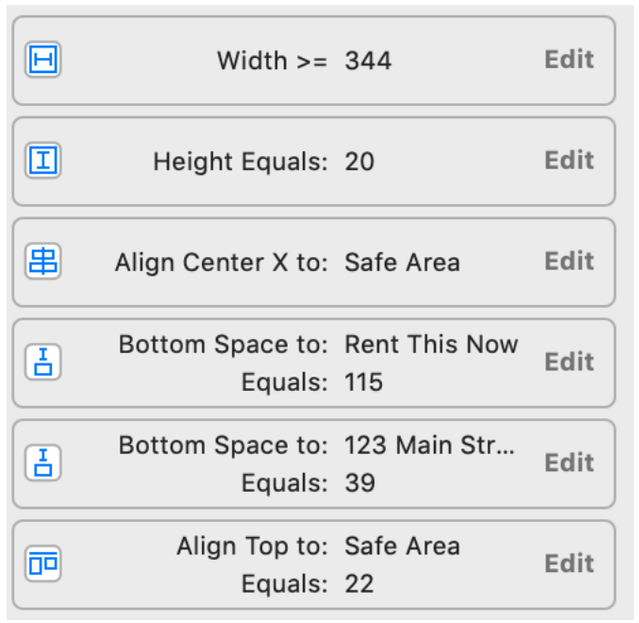
- A given constraint expresses the value of one view's attribute relative to another view's attribute, as a mathematical equation or inequality

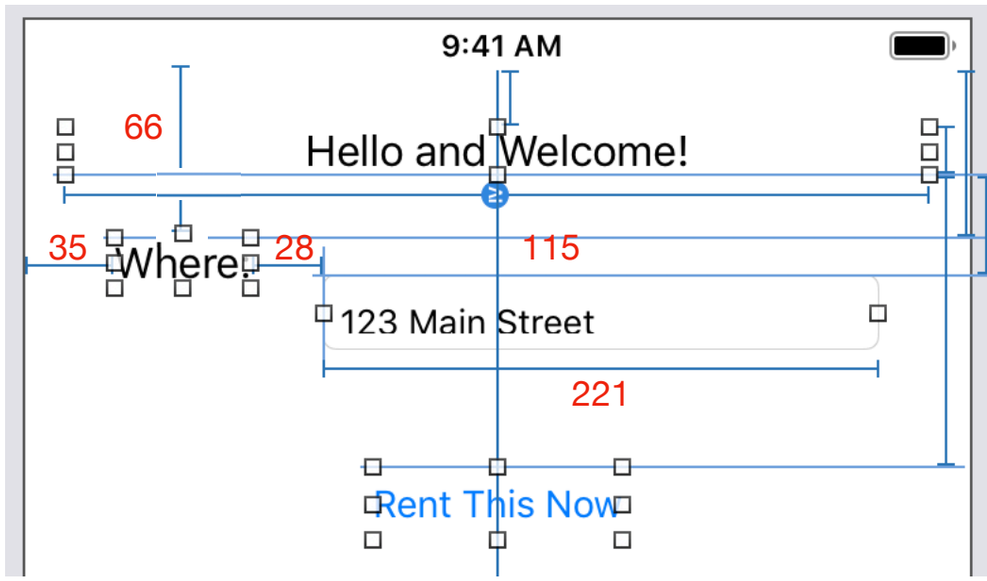
- At runtime, all these linear equations are solved, and the solution is what you see on the screen



The Mathematics of Constraints







- Where is 35 points away from the left edge (Where.leading = SafeArea.leading + 35)
- Where and 123 Main are aligned vertically (Where.centerY = 123Main.centerY)
- Hello's 22 points beneath the SafeArea (Hello.Top = SafeArea.Top + 22)
- Hello's width is at least 344, and its height is 20 (Hello.width ≥ 344, Hello.height = 20)
- 123 Main is 50 points below Hello and Welcome (123 Main.Top = Hello.Bottom + 50)
- 123 Main is 28 to the right of Where (123 Main.Leading = Where.Trailing + 28)
- Rent This Now is centered horizontally relative to its container and 115 below Hello and Welcome (Rent.CenterX = SafeArea.CenterX, Rent.Top = Hello.Bottom + 115)

Constraints in Storyboard

- When you first drag views into the ViewController's view, a set of **prototype constraints** are created for you.

- These define the position of each view relative to the top-left corner of the containing view.

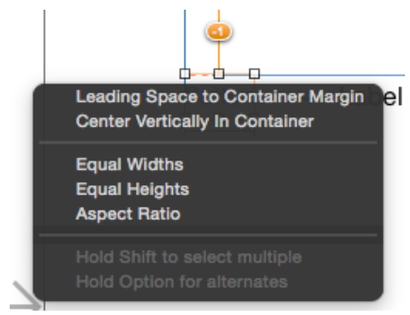
- Generally it is not a good idea to leave them that way, because ... well, just change the orientation of your device to see why: everything stays stuck relative to the left hand corner.

- As soon as you establish one "real" constraint, all of the prototype constraints vanish, and we must define our own.

- To define constraints for an object, there are 3 strategies

Strategy I

- Control-drag from a view to
  - *another View* in the container (set left, top, right, bottom, etc.) to create a constraint between those two views
  - *its container* (to set trailing or leading space, or center horizontally or vertically)
  - *itself* (to set width, height or aspect ratio)



- The direction of dragging affects the options to be displayed: drag left or right for leading/trailing constraints; drag up or down for vertical spacing constraints

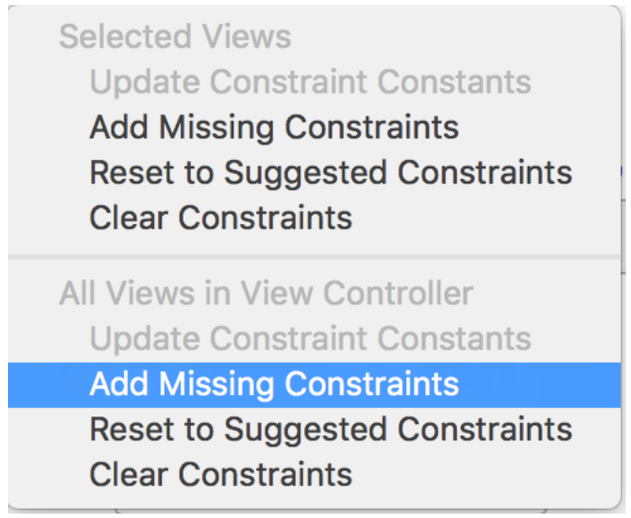
Strategy II

- Select the view, and then use the **align** and **pin** tools to align that view, set its width, height, distance from other views, etc. The stack tool is also very useful when you have to align a series of views that are arranged in a row or column (select the objects, then click on the Stack tool to embed them in a StackView).

- Inspect (and edit) the resulting constraints in the Size Inspector 

Strategy III

- Lay out the views the way you wish them to be
- Select add missing constraints to have IB create all the constraints
- Edit constraints as necessary in the Size Inspector
- Apple suggests that this might be dodgy unless the layout is very simple



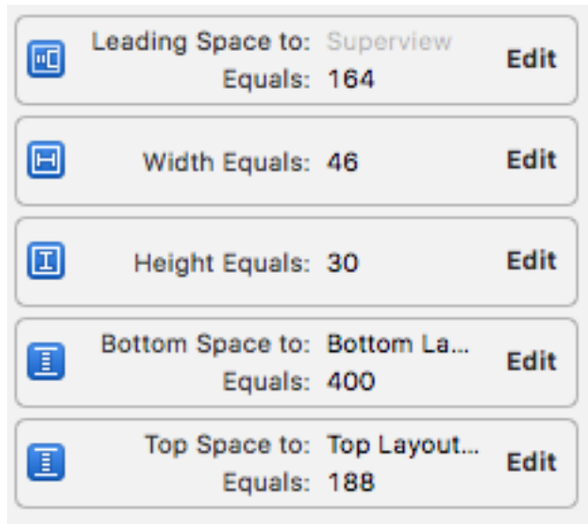
Unambiguous, Satisfiable Layouts

- When specifying multiple constraints, they must be **unambiguous** -- there must be *enough* to specify a solution -- and be **satisfiable** -- they must *have* a solution.

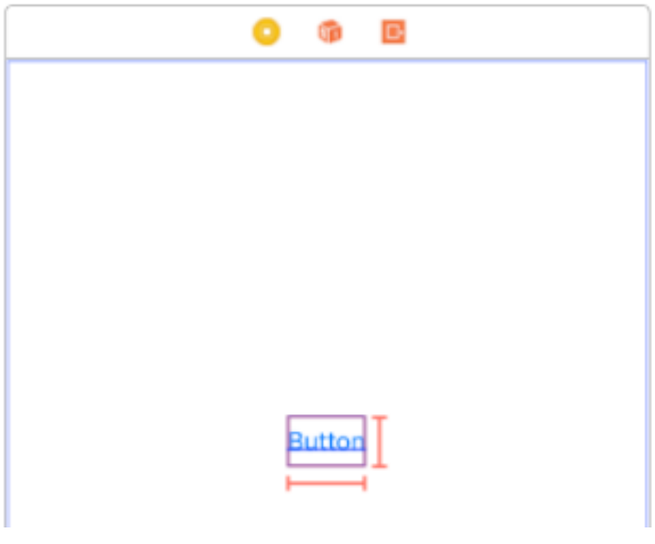
- e.g., specifying a) width and height, or b) leading space, width and height, is ambiguous. We need a *vertical* specification, too

The screen is 647 pixels tall, that's non-negotiable. But setting constraints for the top &bottom space & height leads to a conflict:
188 + 30 + 400 = 618 😢
We can't satisfy all constraints.
Solution? Delete the bottom space constraint (fewer constraints are better), or set it to 429.

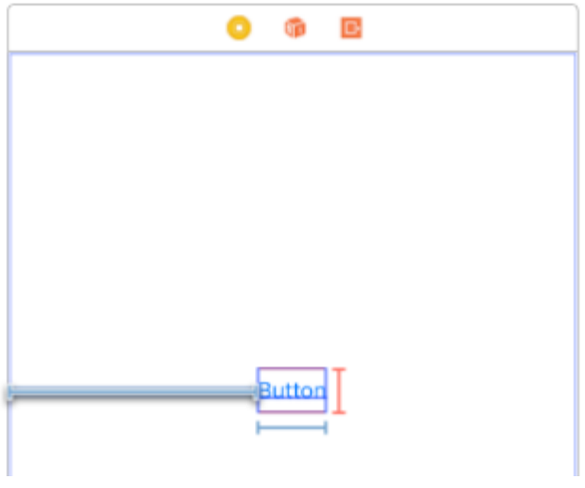
😢



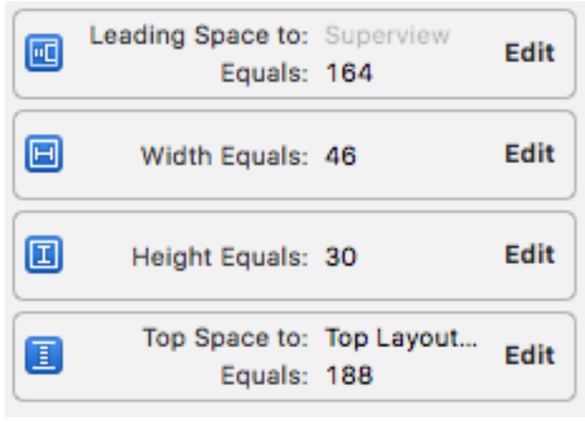
😐

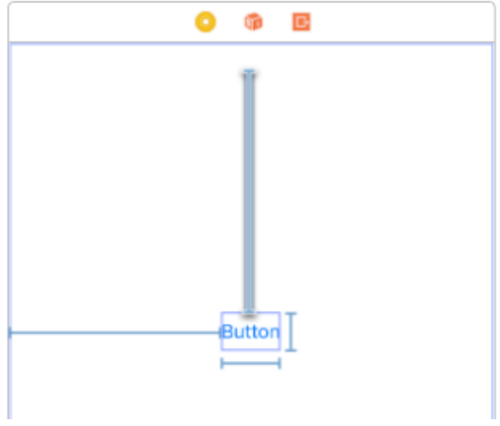


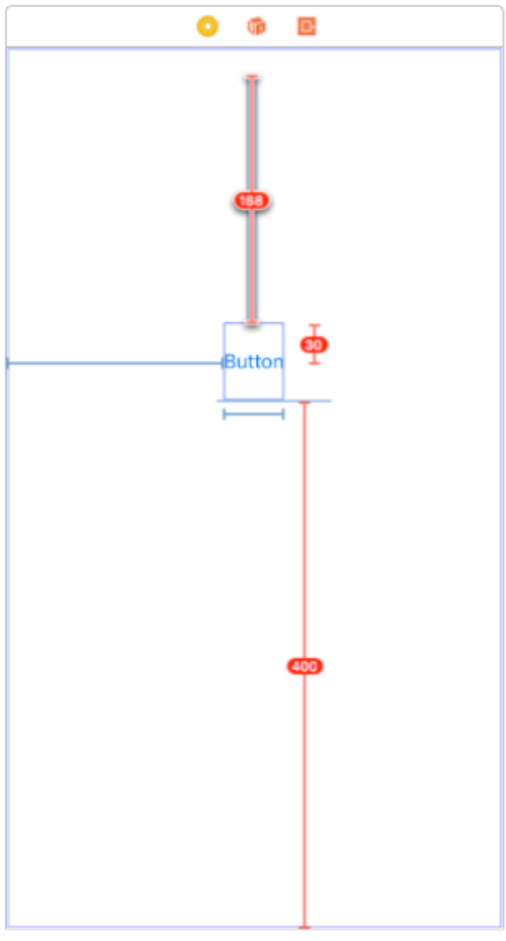
😐



😍

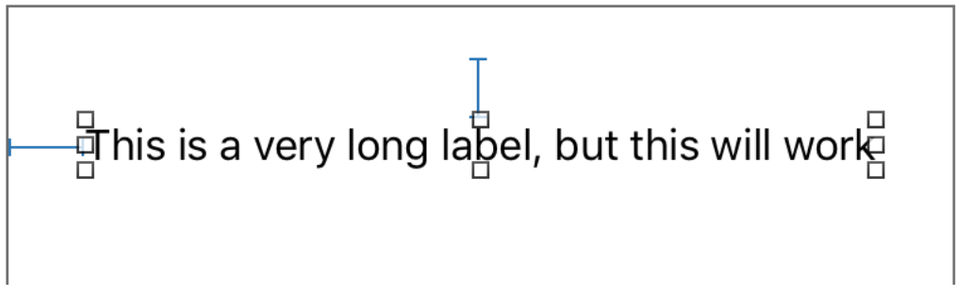




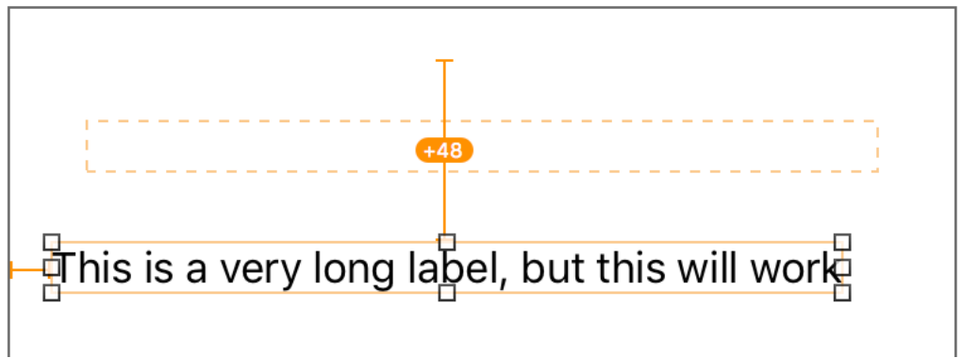


A Common Problem: Misplaced Views

- Say you have a properly constrained view, and then move or resize it (i.e., change its frame)

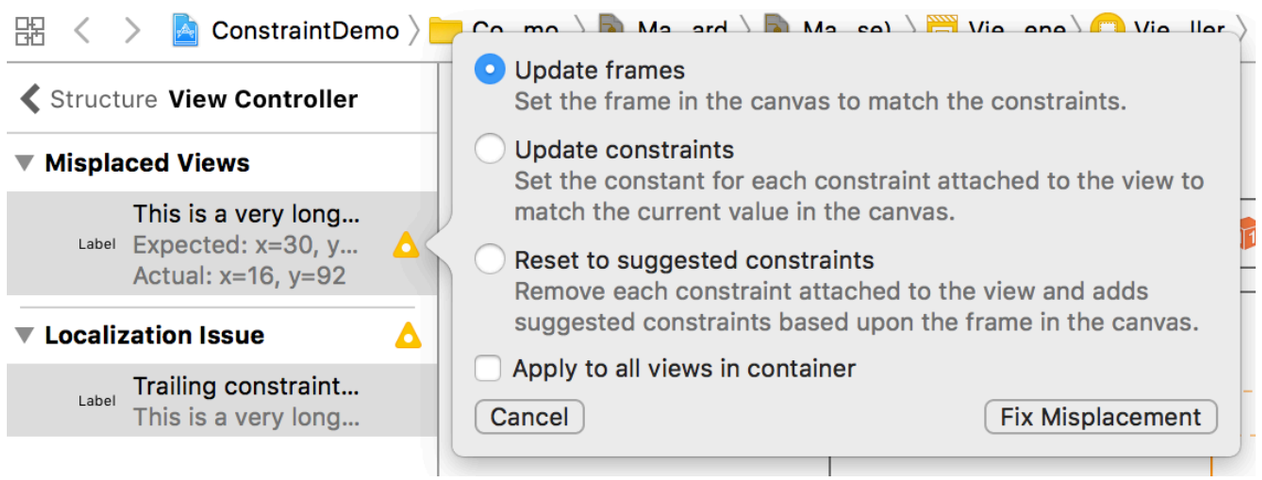


- The constraints are unchanged, and the dashed lines show where the constraint should be



- The solid orange line shows where the frame is (where you moved it to)

- To resolve this, either change the constraints to reflect the position of the frame, or change the frame to agree with the pre-existing constraints



Some Helpful Hints While Creating Constraints

- Drag all your views into place first
- Organize/align them as you like. Use the blue dashed lines that appear as you near the edges and or center of the view controller's view to help get things right
- Establish view constraints in "reading order": from left-to-right, top-to-bottom
  - Use ≥ rather than == for leading and trailing constraints
  - Use Center Align to keep a label and its text field vertically aligned
- See the presentation on Size Classes to make constraints work for different devices in different orientations

Designing for Different-Sized Devices

- What we have studied thus far works for a category of devices — iPhones or iPads — but what if you want to create a UI that works on both?

- Apple has a concept called size classes — constraints can be installed only for devices of a particular size

- This is an advanced topic, however, and we will postpone it for a later date

ICE

- Pending ...