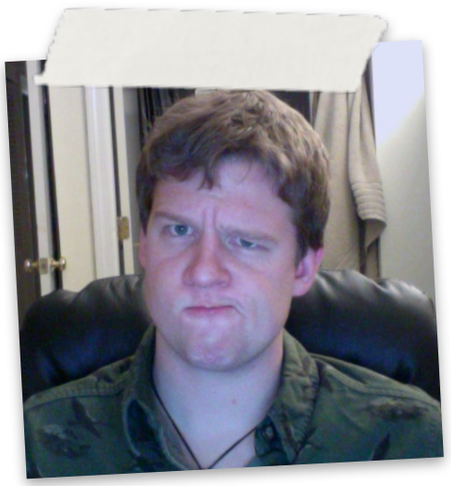
AUTO LAYOUT

“Fun” with Apple’s new layout system

BACK ◆ FORTY



I'M FLIP

@flipsasser

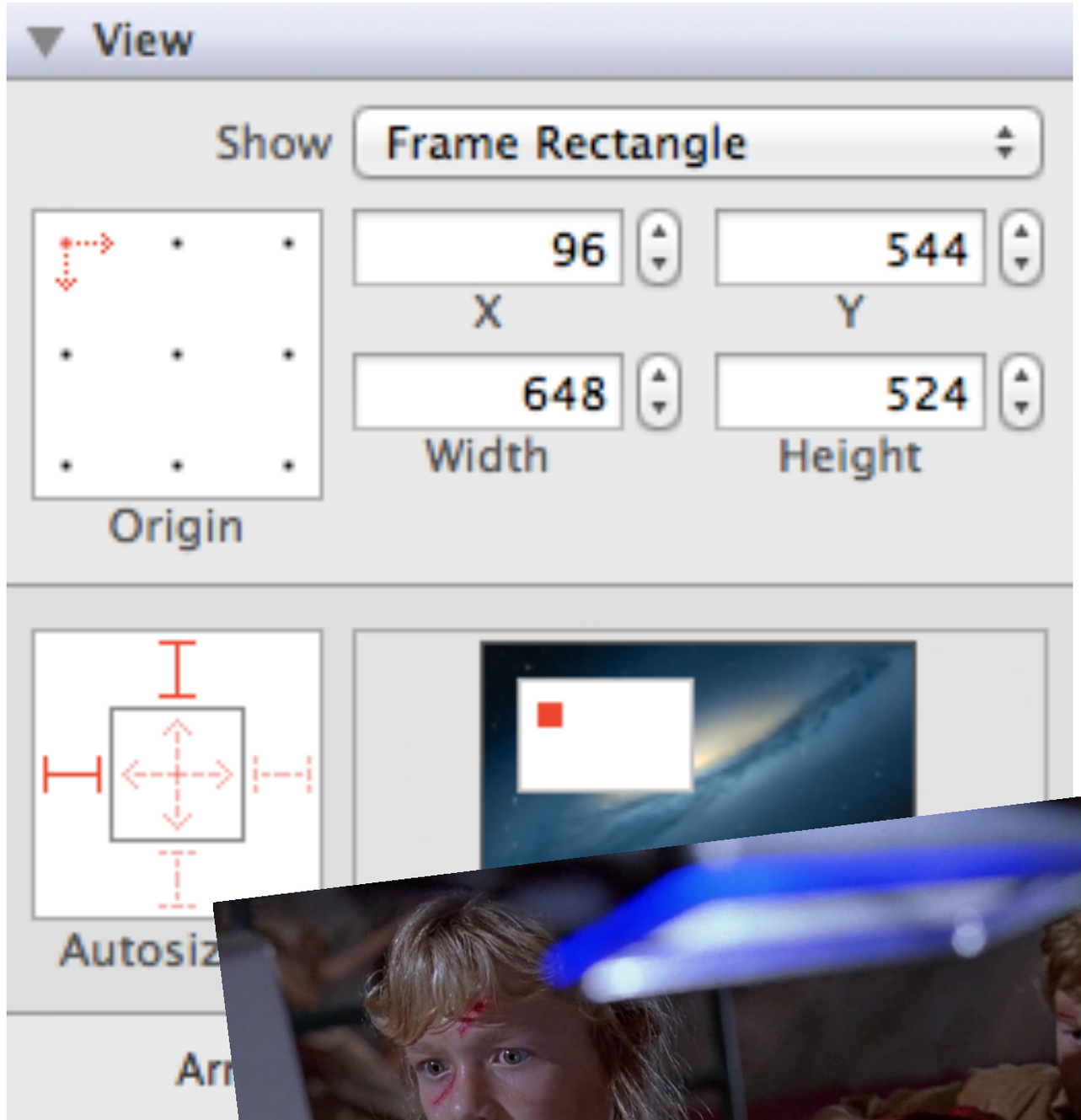
https://github.com/flipsasser

flip@inthebackforty.com

https://github.com/BackForty/actually_invented_here

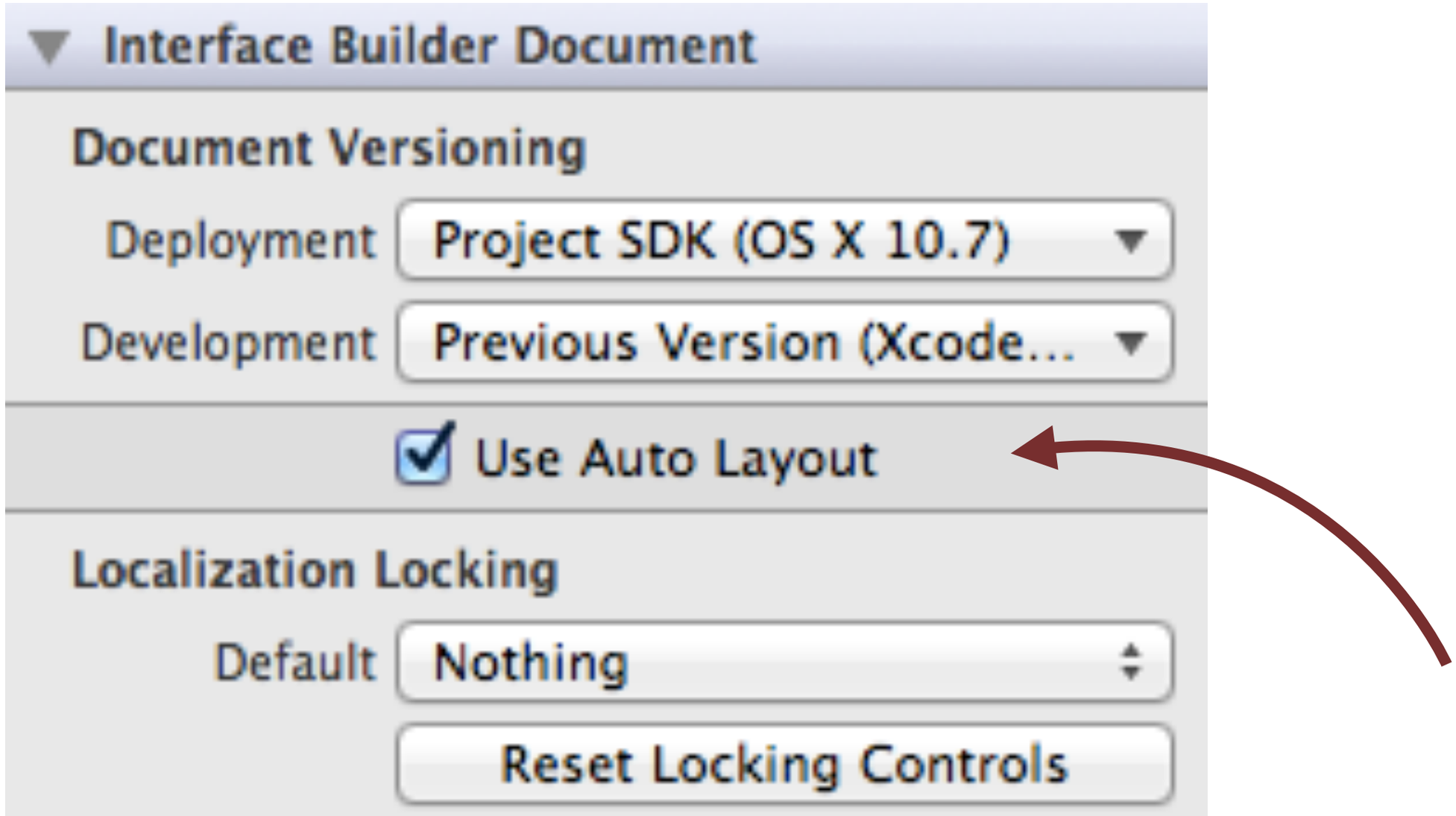
WTF IS AUTO LAYOUT?

It’s a significant upgrade to the “springs and struts” layout system

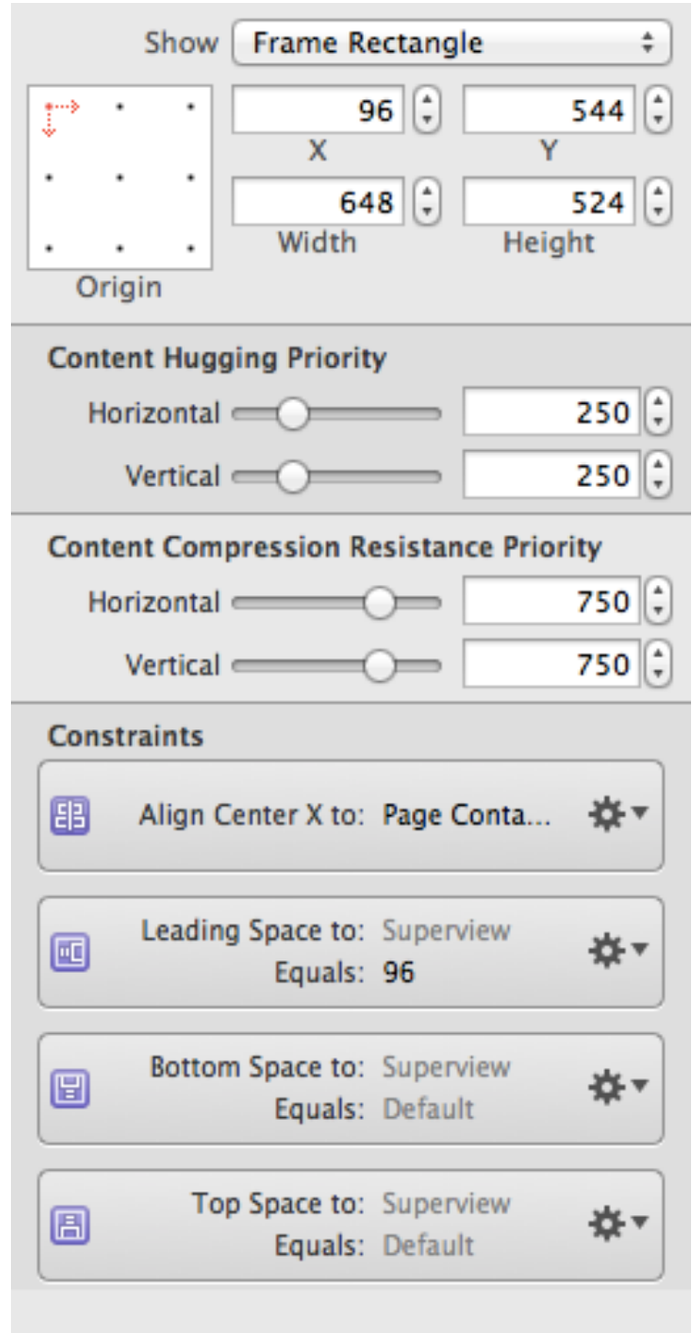


BEFORE: UTS

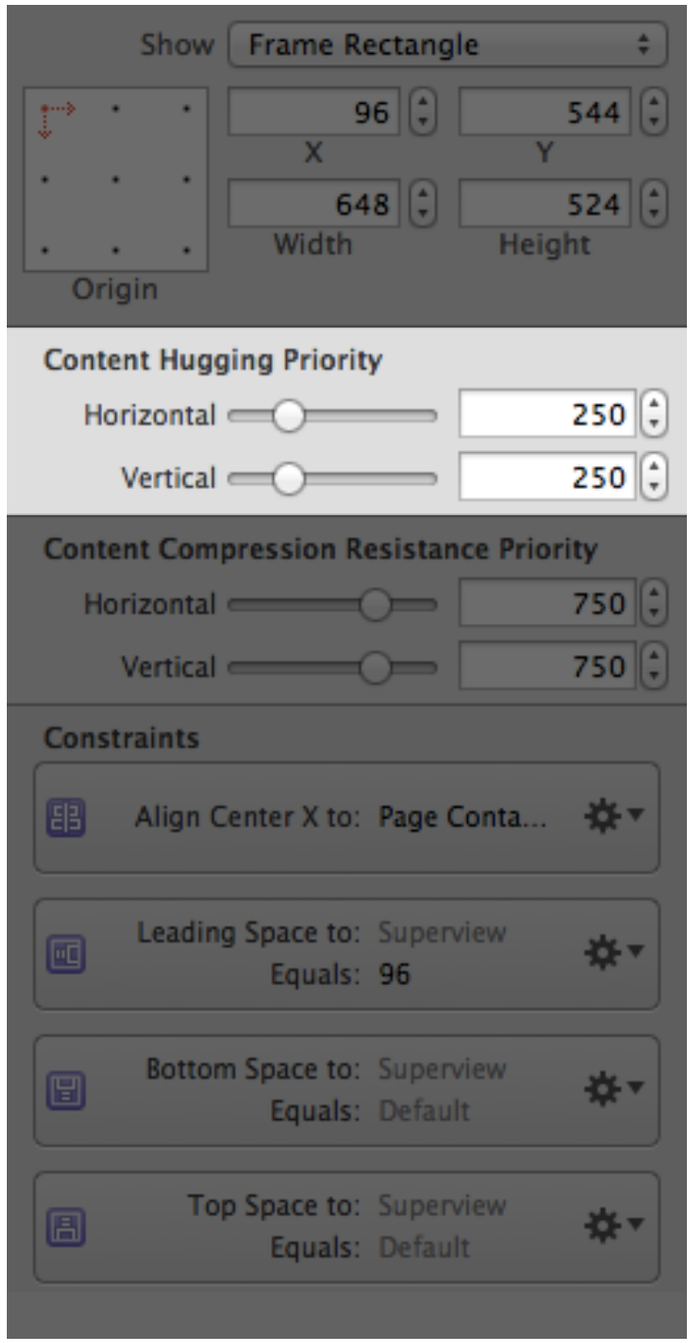
I know this... It's ~~unix~~ springs & struts!



ONE CHECKBOX LATER AND...


Show
Frame Rectangle
96
X
544
Y
648
Width
524
Height
Origin
Content Hugging Priority
Horizontal
250
Vertical
250
Content Compression Resistance Priority
Horizontal
750
Vertical
750
Constraints
Align Center X to: Page Conta...
Leading Space to: Superview
Equals: 96
Bottom Space to: Superview
Equals: Default
Top Space to: Superview
Equals: Default

“WUT”

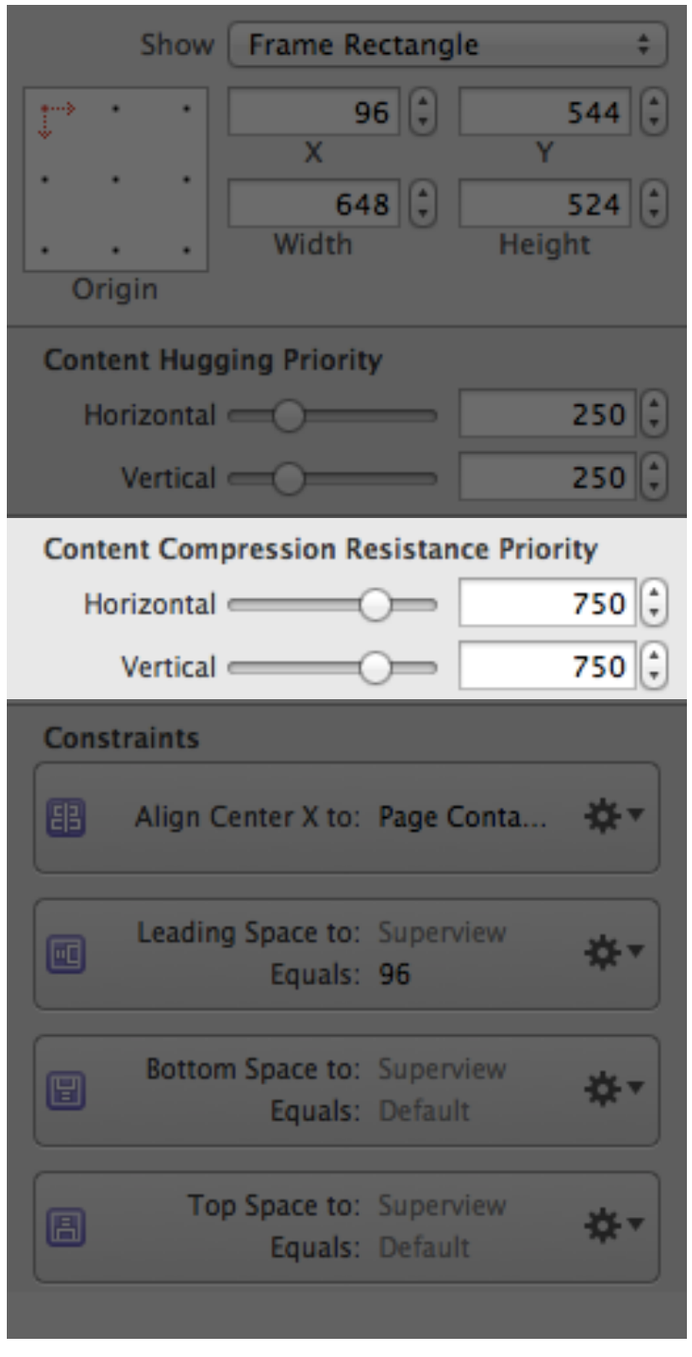


FIRST: CONTENT HUGGING PRIORITY

How close a parent's bounds get to the child

Greater than 500: bounds stand their ground

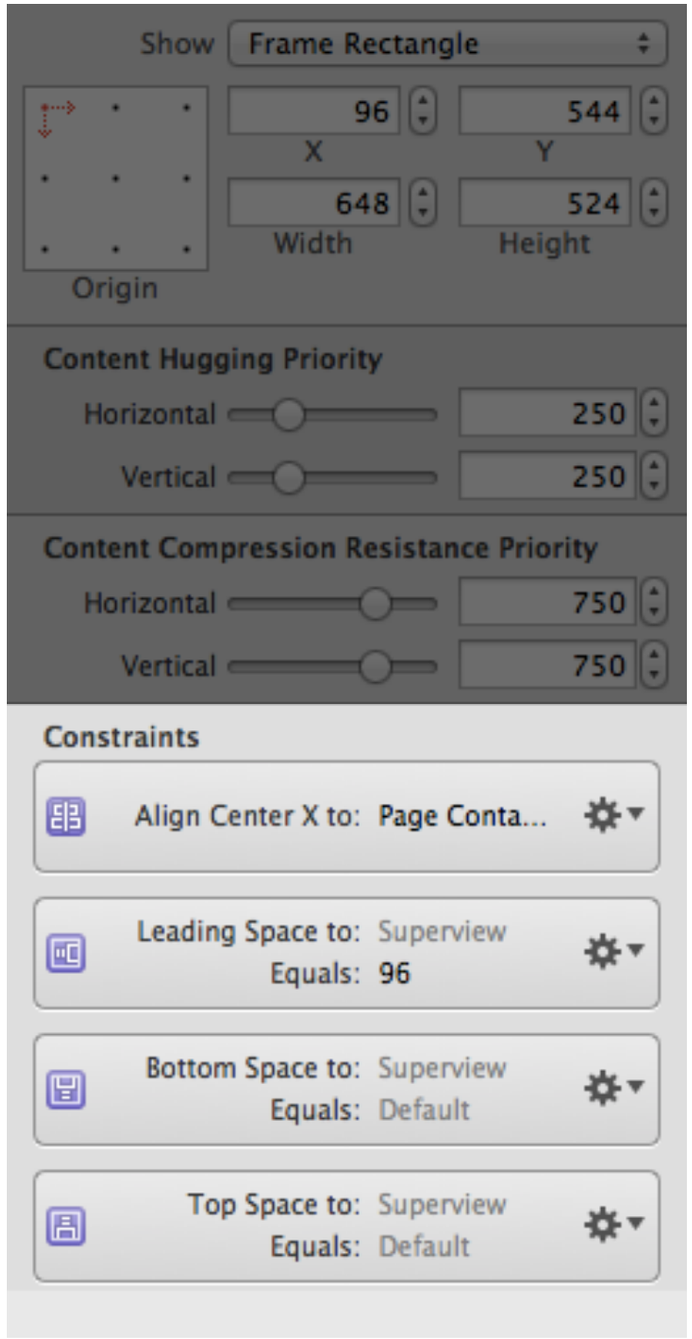
Less than 500: bounds expand with parent



SECOND: CONTENT COMPRESSION RESISTANCE PRIORITY

How much a child pushes back on parent's bounds

Same value rules as content



THIRD: CONSTRAINTS

Define absolute or relative margins and positioning based on **siblings or parent views**

THUS, ALL CONTENT INTELLIGENTLY RESIZES ACCORDING TO RULES

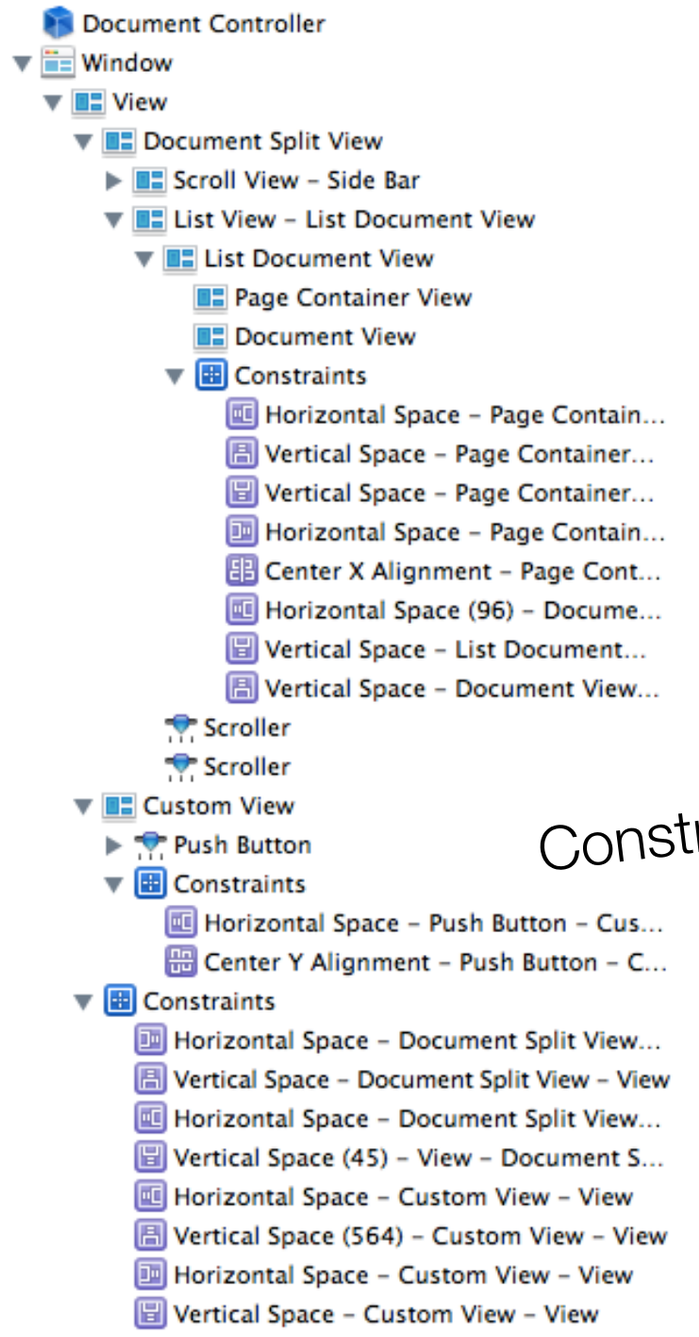
Caveat: this rarely looks “intelligent”

IT'S BASED ON "INTRINSIC SIZE"

Or, "how would something look if there were no constraints applied?"

AUTO LAYOUT IN THE UI

... is for suckers



Constraints on parent applied to siblings

THIS IS WHY AUTO LAYOUT IS SO FRUSTRATING TO BEGINNERS

BOTTOM LINE:

Don’t do it. It sucks.

AUTO LAYOUT IN CODE

... is for happy developers

Well, happy-ish

THE WEIRD WAY

`constraintsWithVisualFormat:` uses an NSPredicate-style “language”

|[VIEW]-|

Flush to the left edge of superview; Apple UI standard space to the right

V:[SIBLING]-10-[VIEW(100)]

10 pixels down from siblings;
exactly 100 pixels wide

[VIEW(==SIBLING)]

The same width as sibling

V:[VIEW(==SIBLING)]

The same height as sibling

BUT HOW DOES IT KNOW WHAT VIEW AND SIBLING ARE?

THE MOST AMAZING METHOD EVER

NSDictionaryOfVariableBindings

BEFORE YOU KNEW ABOUT NSDICTIONARYOFVARIABLEBINDINGS

```
NSValue *someValue = [MyClass
  valueWithThingAndAThing:@"Yup"];

NSDictionary *myNeatDict = [NSDictionary
  dictionaryWithObjectsAndKeys:someValue,
  @"someValue",
  nil];
```

AFTER YOU KNEW ABOUT NSDICTIONARYOFVARIABLEBINDINGS

```
NSValue *someValue = [MyClass
  valueWithThingAndAThing:@"Yup"];

NSDictionary *myNeatDict =
  NSDictionaryOfVariableBindings(someValue);
```

BOTH PRODUCE THE SAME THING:

{“someValue” => [NSValue instance]}

USED WITH CONSTRAINTS

```
NSView *sibling =
  [self.superview.subviews objectAtIndex:1];

NSDictionary *views =
  NSDictionaryOfVariableBindings(sibling);

NSArray *constraints =
  [NSLayoutConstraint
    constraintsWithVisualFormat:@"V:|-
[sibling]-|"
    options:nil metrics:nil views:views];
```

*tell the sibling view to keep the uniform distance
to the left and right of the parent

NOW APPLY THE ARRAY OF CONSTRAINTS

```
[self.superview
  addConstraints:constraints];
```

THE NORMAL WAY

`constraintWithItem`, because you program for a living

GET A SINGLE CONSTRAINT

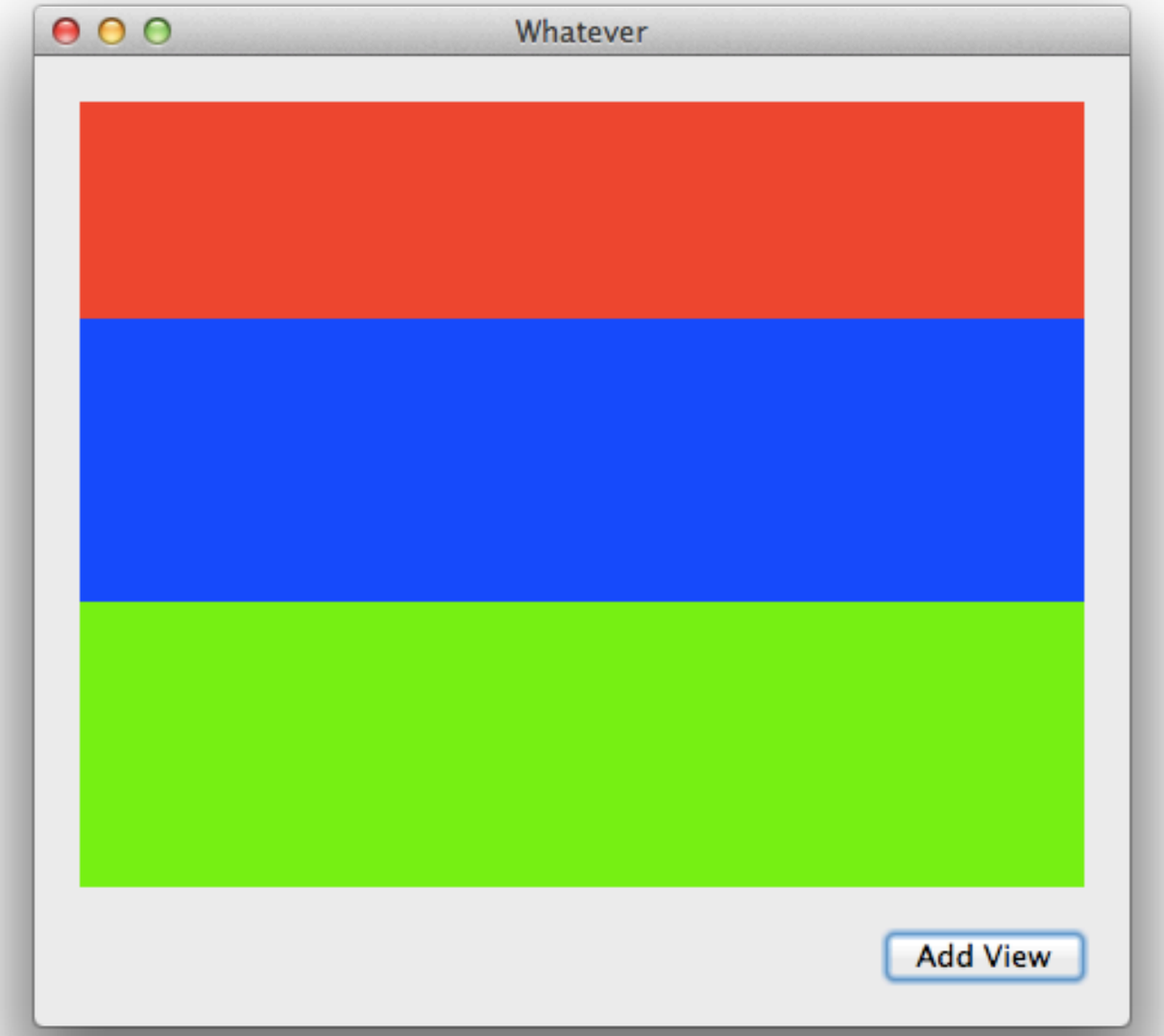
```
NSLayoutConstraint *widthConstraint =
  [NSLayoutConstraint
    constraintWithItem:view
    attribute:NSLayoutAttributeWidth
    relatedBy:NSLayoutRelationEqual
    toItem:self
    attribute:NSLayoutAttributeWidth
    multiplier:1.0 constant:0];
```

AMAZING AWESOMENESS OF THIS APPROACH

* Applies directly to related views
* Specifies relationship granularly
* Multipliers (e.g. “1.5 x sibling”)
* Constants (“always 100px”)
* You don’t forget to instantiate an array because it’s just an NSLayoutConstraint

THE HOLY GRAIL

StackableView



STACKS VIEWS OF RANDOM HEIGHTS BELOW EACH OTHER

BOOM!

HOW IT WORKS

`addSubview` creates constraints:

* Top constrained to the previous child view's bottom
* Left and right constrained to parent view's left and right
* Bottom constrained to the next child's top or the parent view's bottom

GOTCHAS

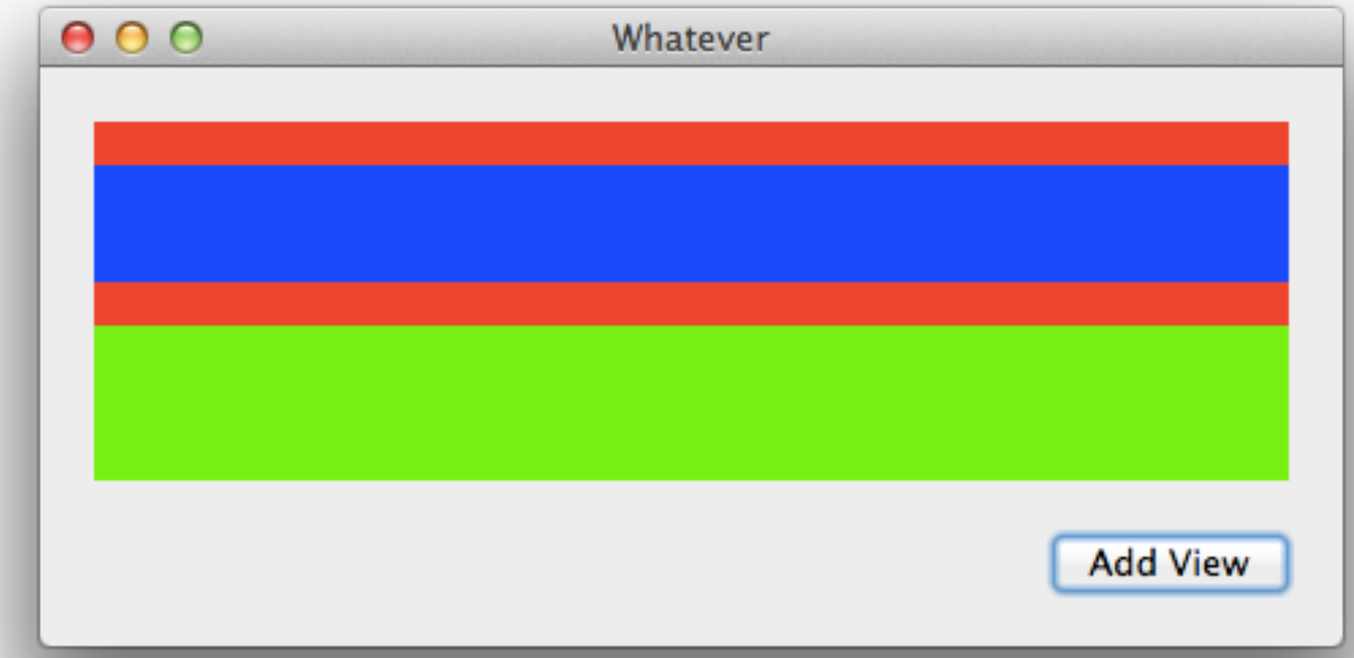
NESTING IS A DISASTER

A view whose intrinsic size is the sum of its children’s intrinsic sizes is a disaster waiting to happen

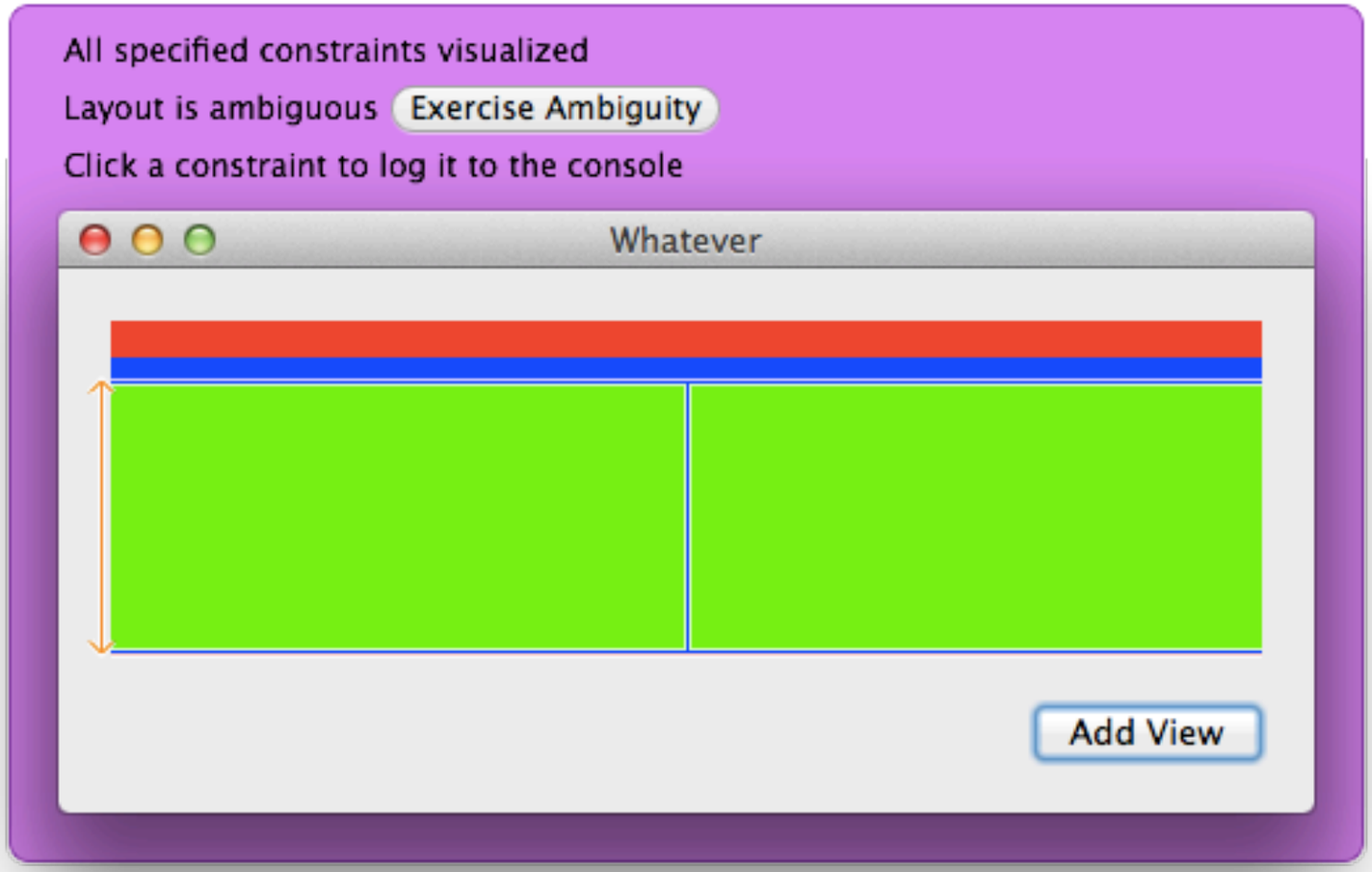
DEBUGGING IS KEY

Edit Scheme -> Run -> Arguments ->

```
-NSConstraintBasedLayoutVisualizeMutuallyExclusiveConstraints YES
```



TURNS THIS...


All specified constraints visualized
Layout is ambiguous
Exercise Ambiguity
Click a constraint to log it to the console
Whatever
Add View

INTO THIS!

DRAWBACKS

It ain’t intuitive

It can be tricky to debug

Also...

YOU MAY HAVE TO DO STUFF LIKE THIS

```
  // Remove bottom layout constraints from the document view
  for (NSLayoutConstraint *constraint in self.view.superview.constraints) {
    if (constraint.secondItem == self.view && constraint.firstAttribute ==
NSLayoutAttributeBottom) {
      [self.view.superview removeConstraint:constraint];
      NSLayoutConstraint *saneConstraint = [NSLayoutConstraint
constraintWithItem:self.view attribute:NSLayoutAttributeBottom
relatedBy:NSLayoutRelationGreaterThanOrEqual toItem:self.view.superview
attribute:NSLayoutAttributeBottom multiplier:0.0 constant:20.0];
      [self.view.superview addConstraint:saneConstraint];
    }
  }
```

FEED ME YOUR QUESTIONS I AM A QUESTION MONSTER AND I NEED THEM TO SURVIVE

BACK ◆ FORTY

THNAKS!