Thinking In NSLayoutConstraints

Who Am I?

- I run M Cubed Software (mcubedsw.com)
- Built many apps using Auto Layout
- Last year I talked about how Auto Layout thinks
- This year I'll talk about how you should think

What Is Auto Layout?

- Constraint-based layout system for iOS & Mac
- Define relationships between views
- Introduced in Mac OS X 10.7 and iOS 6
- Make previously complex layout problems simple
- Requires a different way of thinking about layout
- Fits more closely to your natural mental model

Constraints: How Do They Work?

CONSTRAINTS

- Represented by NSLayoutConstraint
- Defines relationship between two attributes
- Attributes are effectively variables
- Treat a constraint as small function modifying a variable

```
y = mx + c
```

CONSTRAINTS

- Represented by NSLayoutConstraint
- Defines relationship between two attributes
- Attributes are effectively variables
- Treat a constraint as small function modifying a variable

```
view1.attribute = m * x + c
```

CONSTRAINTS

- Represented by NSLayoutConstraint
- Defines relationship between two attributes
- Attributes are effectively variables
- Treat a constraint as small function modifying a variable

```
view1.attribute = m * view2.attribute + c
```

Constraints

- Represented by NSLayoutConstraint
- Defines relationship between two attributes
- Attributes are effectively variables
- Treat a constraint as small function modifying a variable

```
view1.attribute = multiplier * view2.attribute + constant
```

Constraints

- Represented by NSLayoutConstraint
- Defines relationship between two attributes
- Attributes are effectively variables
- Treat a constraint as small function modifying a variable

```
v1.attr = multipler * v2.attr + constant
```

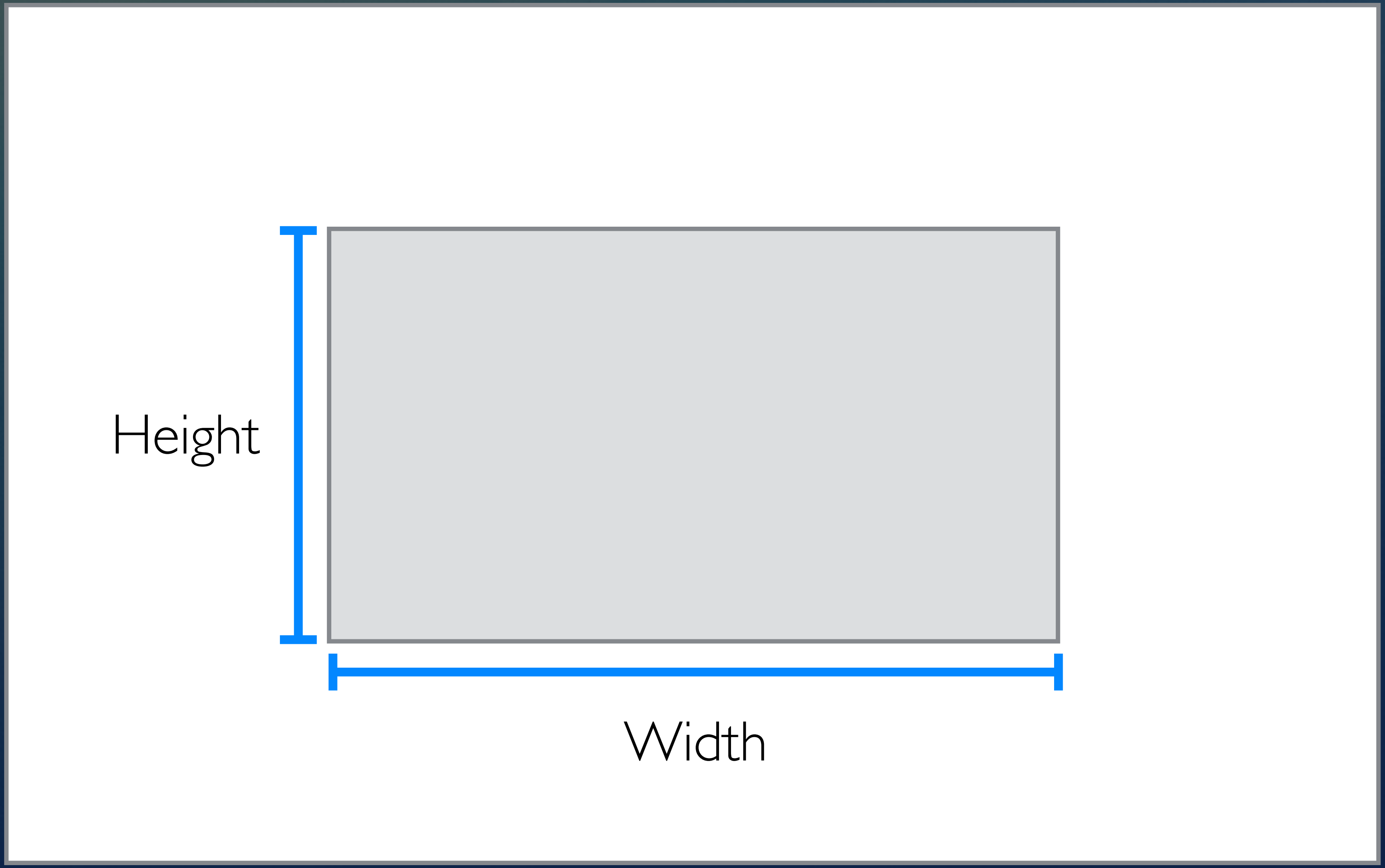
```
v1.attr = multipler * v2.attr + constant
```

Constraints

- Represented by NSLayoutConstraint
- Defines relationship between two attributes
- Attributes are effectively variables
- Treat a constraint as small function modifying a variable

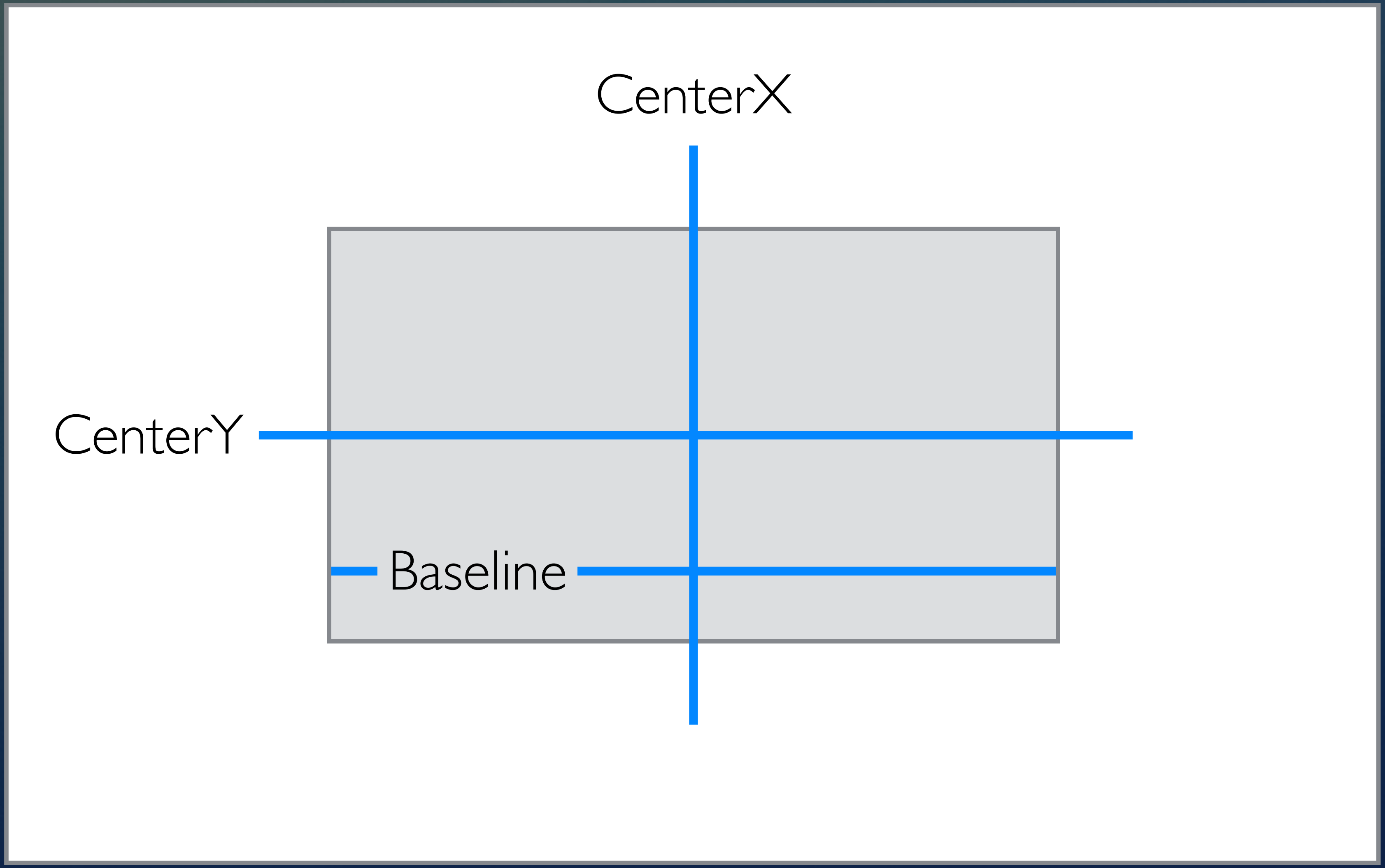
`v1.attr = multipler * v2.attr + constant`

Attributes



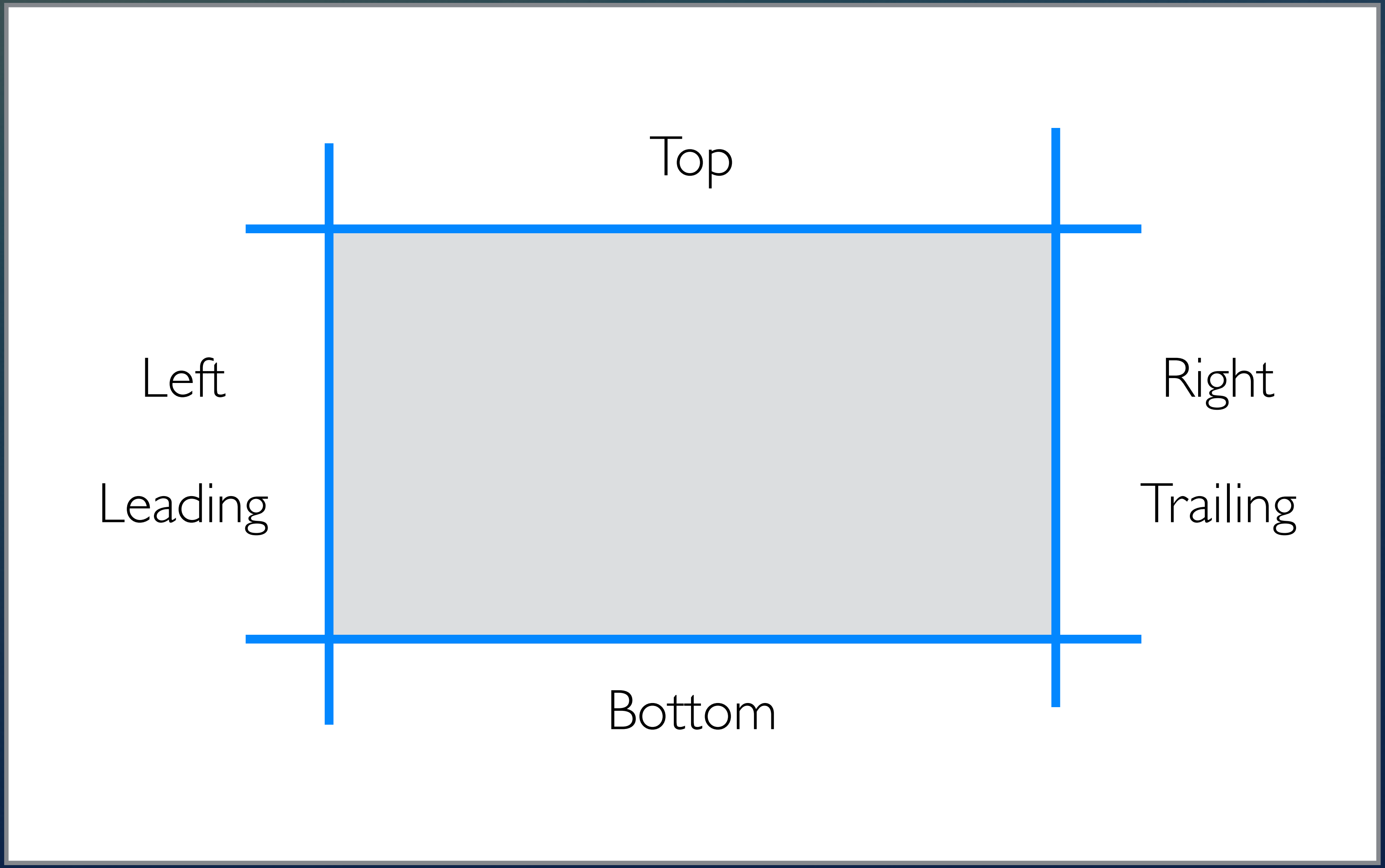
```
v1.attr = multipler * v2.attr + constant
```

Attributes



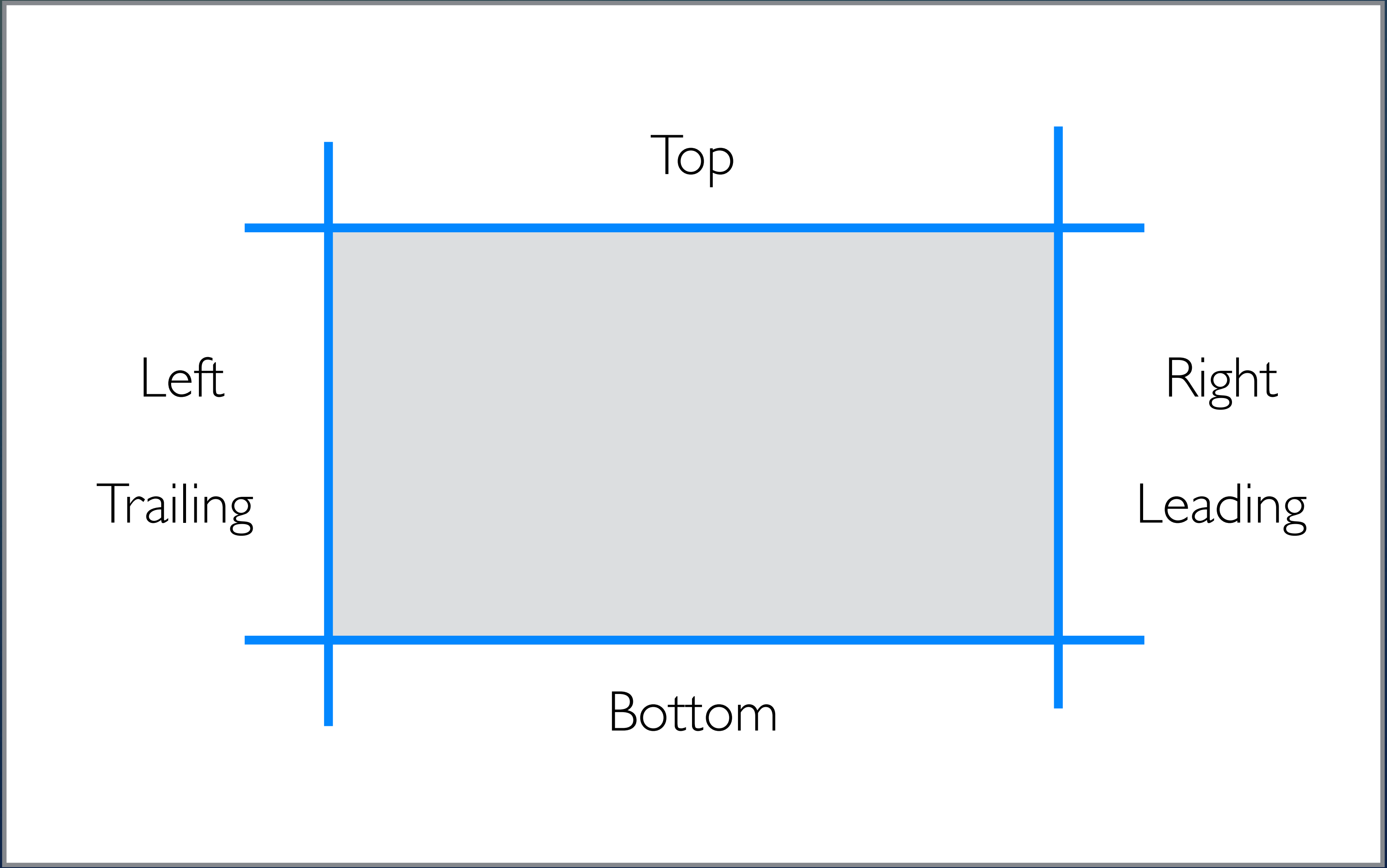
```
v1.attr = multipler * v2.attr + constant
```

Attributes



```
v1.attr = multipler * v2.attr + constant
```

ATTRIBUTES



```
v1.attr = multipler * v2.attr + constant
```

Relationships

- Equal
- Greater than or equal to
- Less than or equal to

`v1.attr = multipler * v2.attr + constant`

Multiplier and Constant

- Multiplier - The ratio between two attributes
- Constant - The difference between two attributes

`v1.attr = multipler * v2.attr + constant`

Priority

- How strongly should a constraint be satisfied
- Constraints required by default
- Optional constraints can be broken without errors
- Required constraints have priority 1000
- Lower priority constraints are broken to satisfy higher priority ones

Your New Mental Model

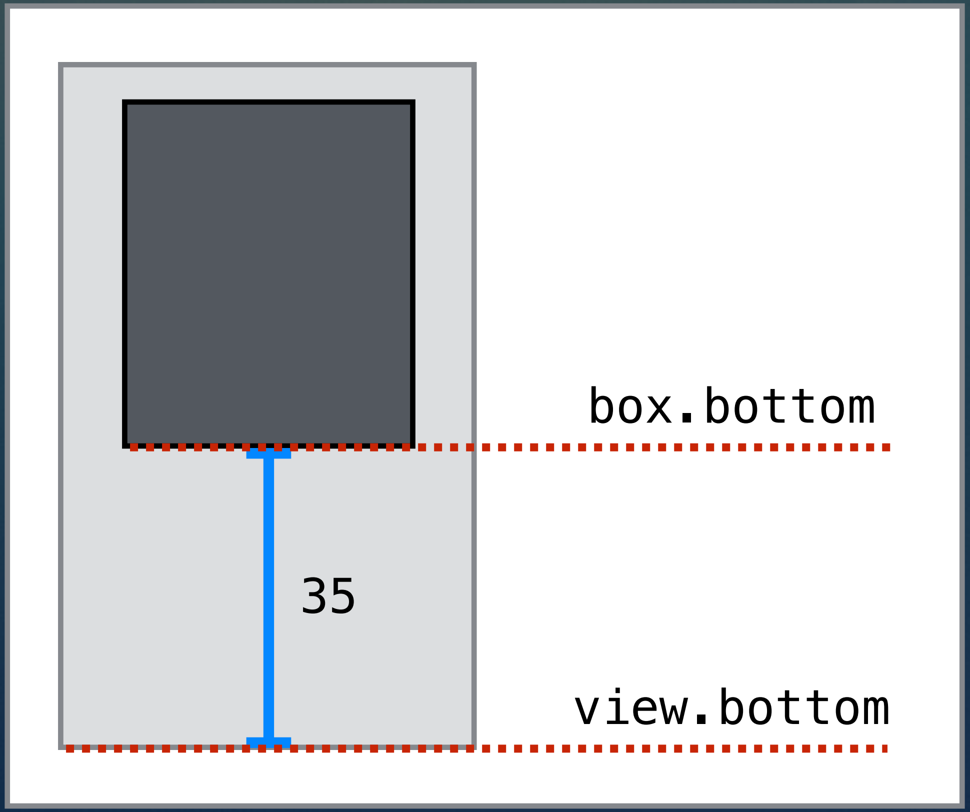
Relative vs Absolute

- Don’t think in frames, think in relationships
- Most constraints are relative to other attributes
- No need to do complex calculations based on other views

Thinking In Values

- Can be hard to work out what attributes, constant etc to use
- Don't think of them as abstract values
- Substitute in numbers

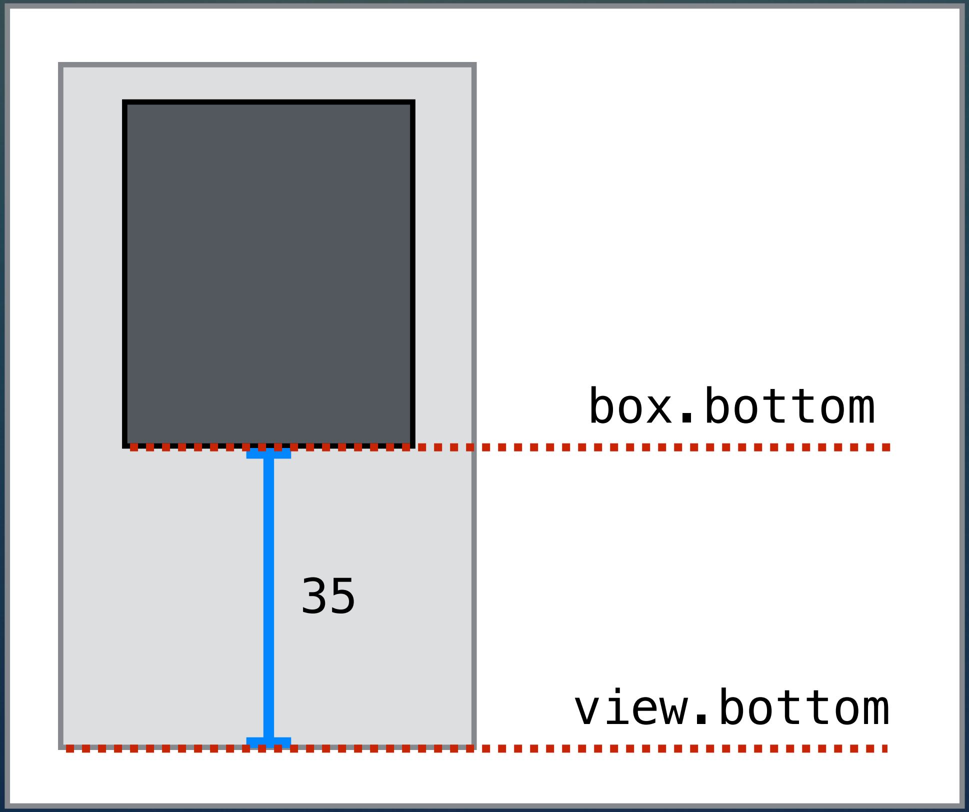
Thinking In Values



- Relationship between **`box.bottom`** and **`view.bottom`**
- Distance between is **35**

```
y = mx + c
```

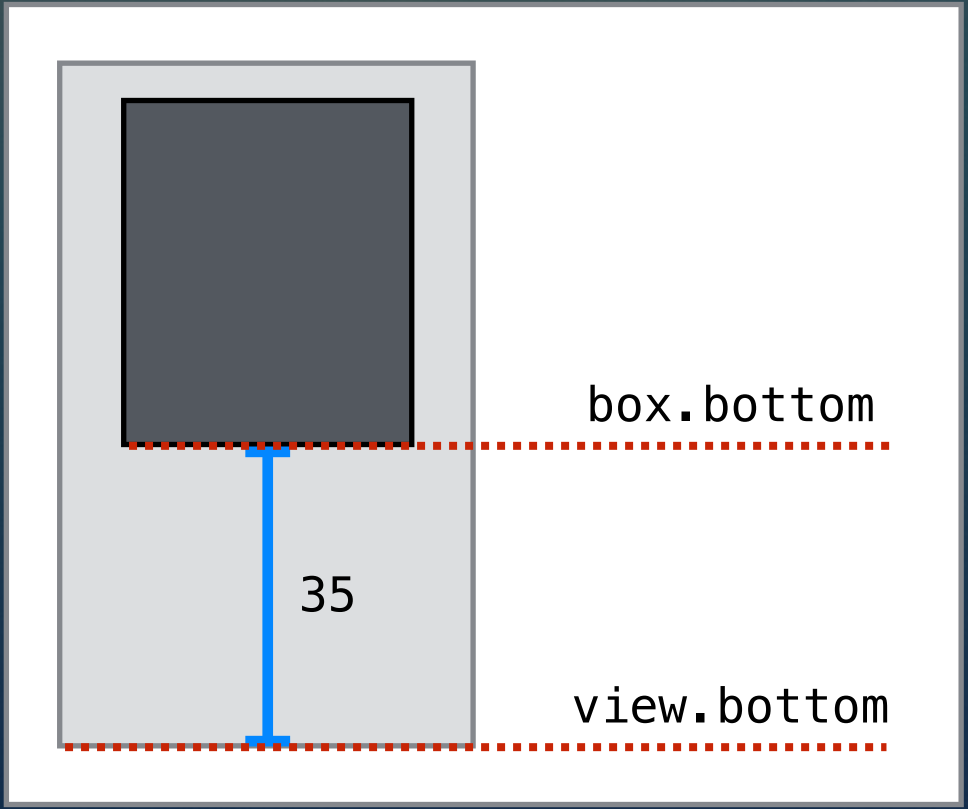
Thinking In Values



- Relationship between **`box.bottom`** and **`view.bottom`**
- Distance between is 35

```
box.bottom = mx + c
```

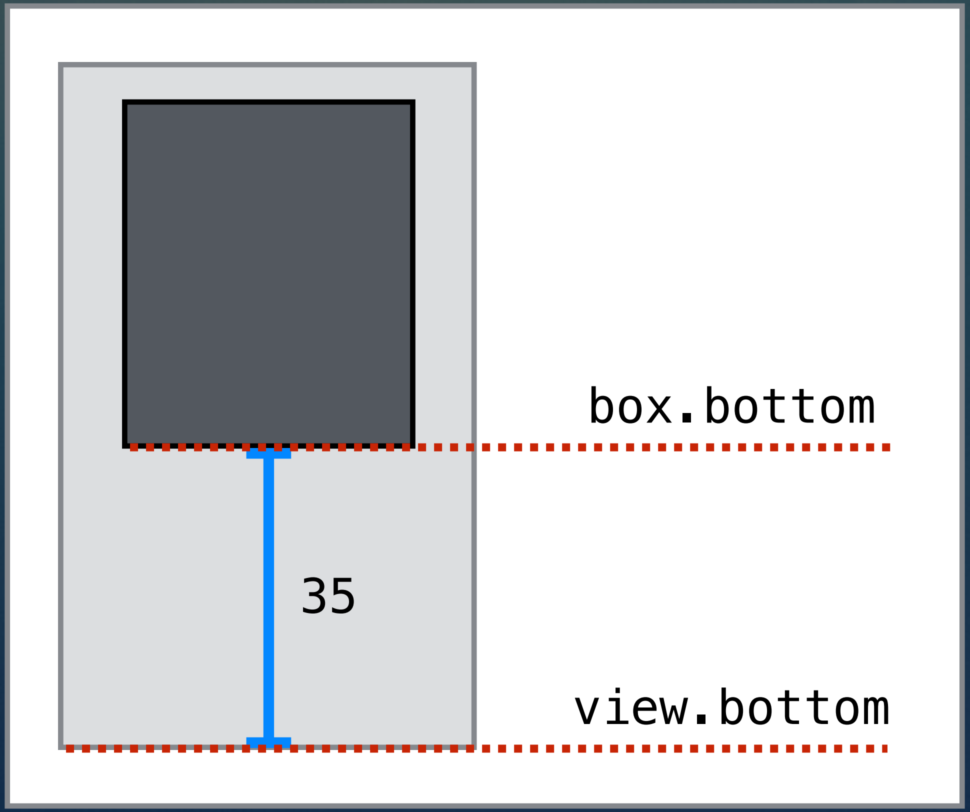
Thinking In Values



- Relationship between **`box.bottom`** and **`view.bottom`**
- Distance between is 35

```
box.bottom = x + c
```

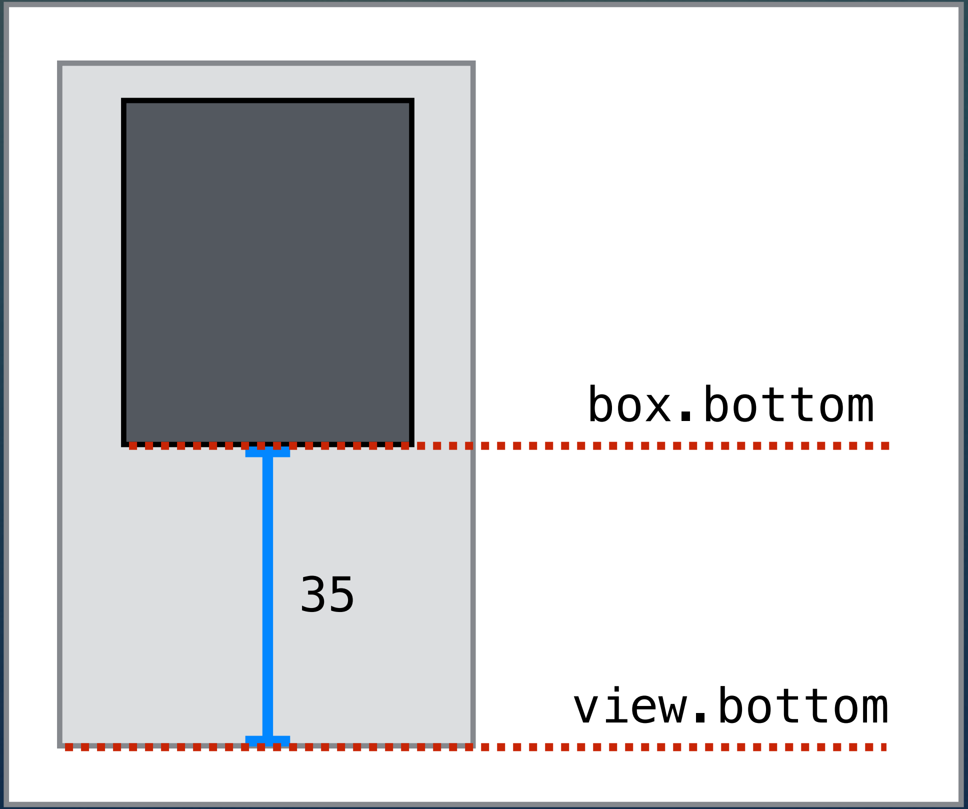
Thinking In Values



- Relationship between `box.bottom` and `view.bottom`
- Distance between is 35

```
box.bottom = view.bottom + c
```

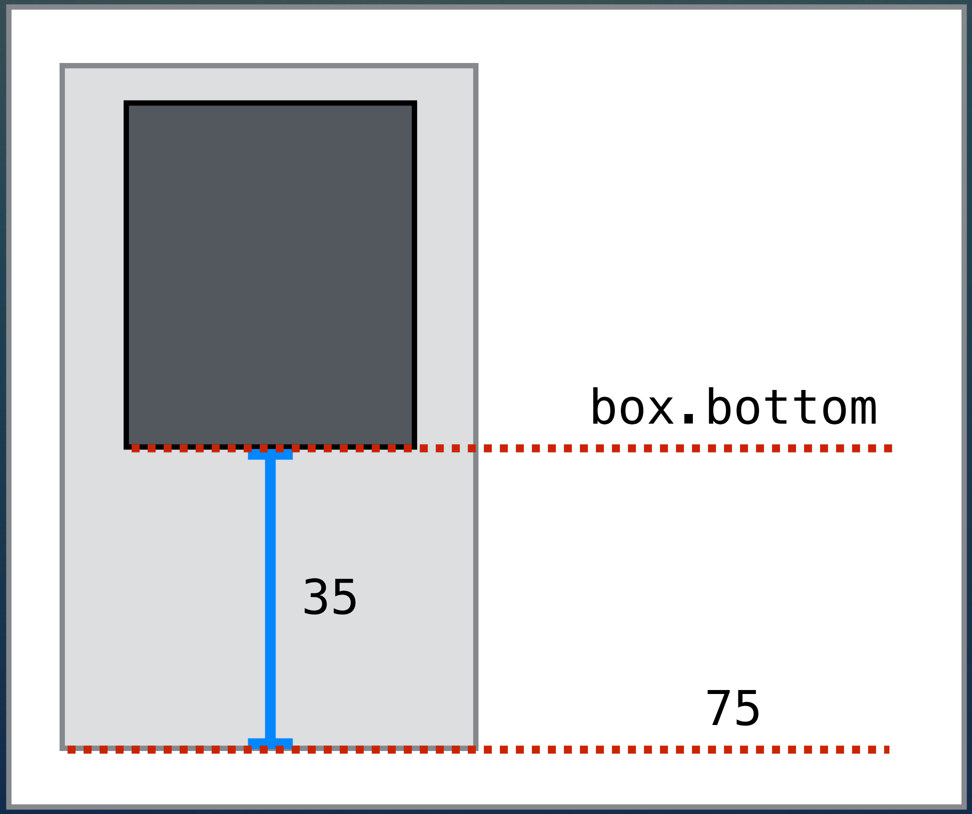
THINKING IN VALUES



- Relationship between `box.bottom` and `view.bottom`
- Distance between is 35

```
box.bottom = view.bottom ± 35
```

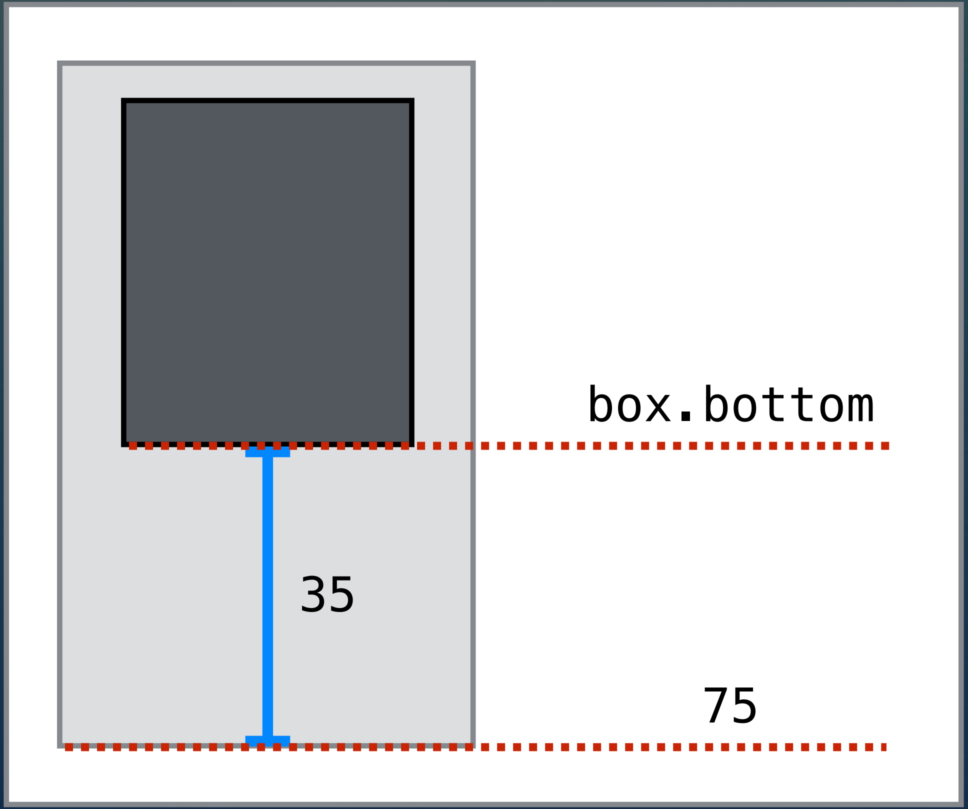
THINKING IN VALUES



- Relationship between **`box.bottom`** and **`view.bottom`**
- Distance between is 35

```
box.bottom = view.bottom ± 35
```

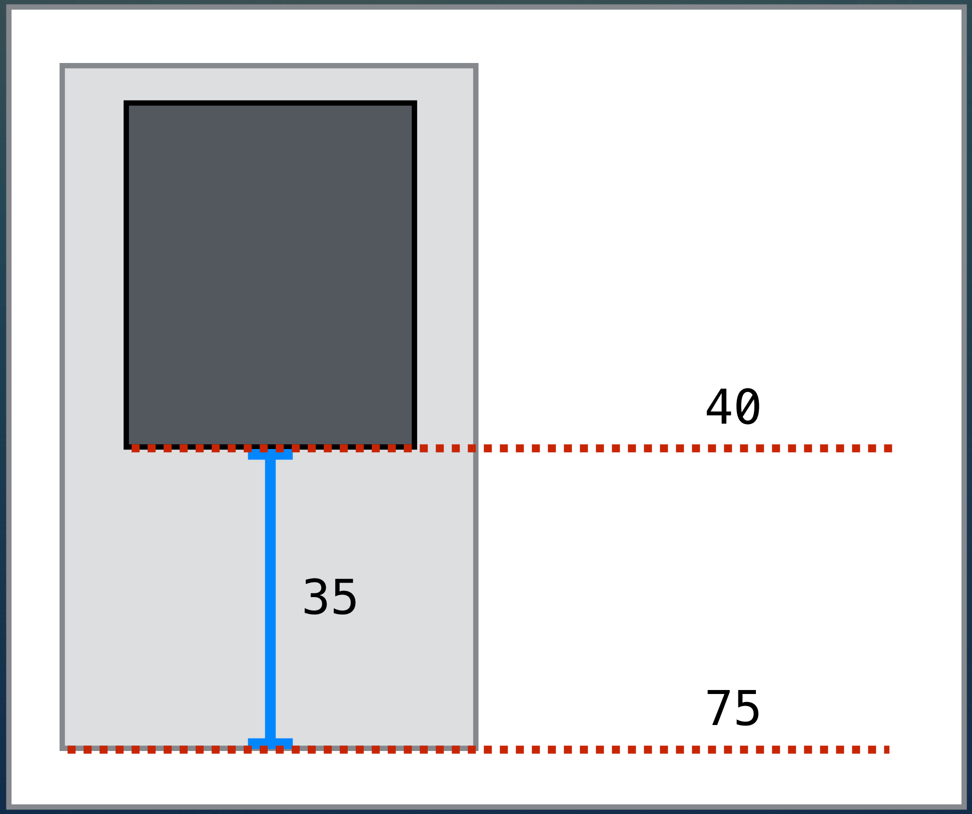
Thinking In Values



- Relationship between **`box.bottom`** and **`view.bottom`**
- Distance between is 35

```
box.bottom = view.bottom - 35
```

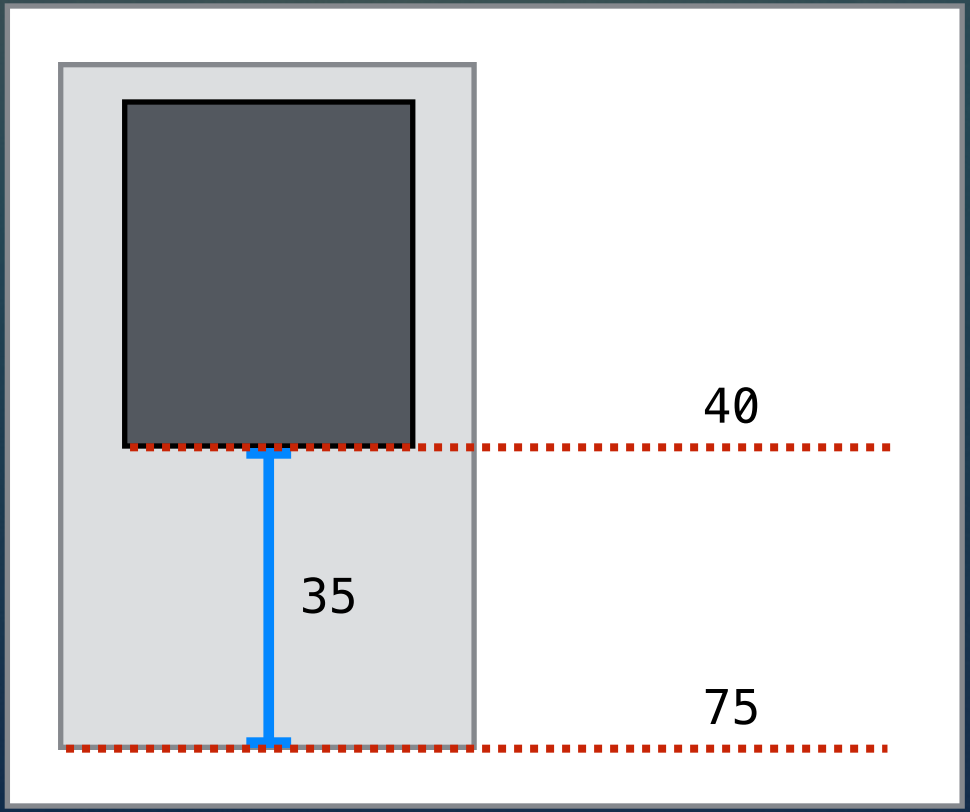
Thinking In Values



- Relationship between `box.bottom` and `view.bottom`
- Distance between is 35

```
box.bottom = view.bottom - 35
```

Thinking In Values



- Relationship between `box.bottom` and `view.bottom`
- Distance between is 35

```
view.bottom = box.bottom + 35
```

Constraining a View

- All views need at least 4 constraints
- Need to position and size in both horizontal and vertical axes

```
leading

top

width

height
```

Constraining a View

- All views need at least 4 constraints
- Need to position and size in both horizontal and vertical axes

```
trailing

bottom

width

height
```

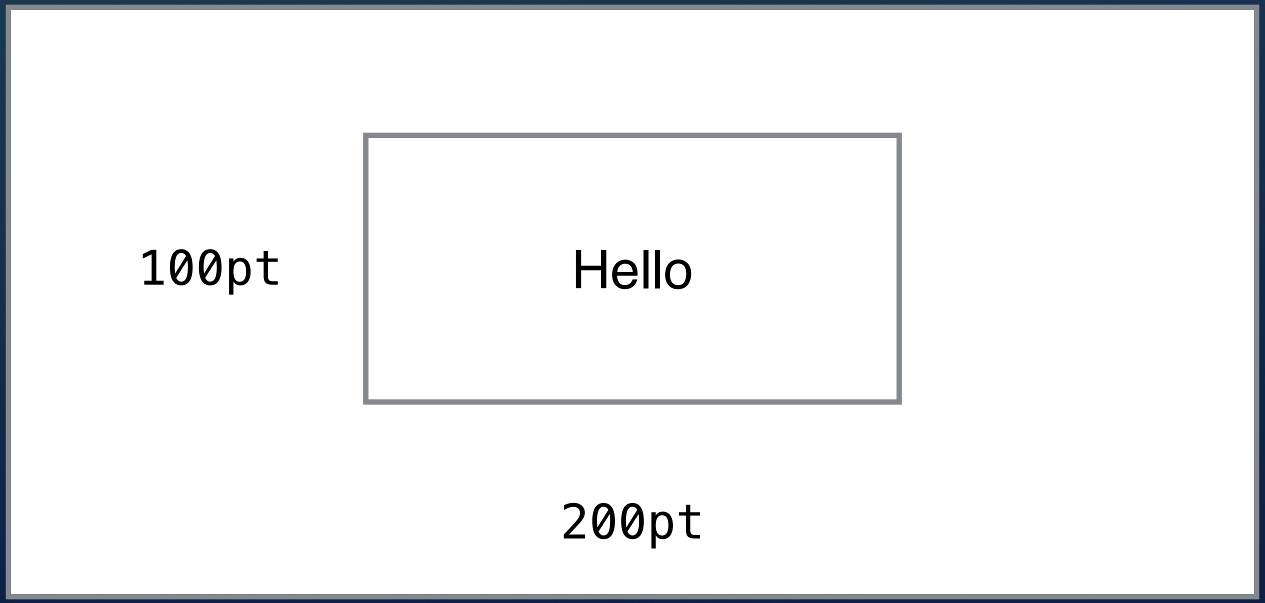
Constraining a View

- All views need at least 4 constraints
- Need to position and size in both horizontal and vertical axes

```
top
bottom
leading
trailing
```

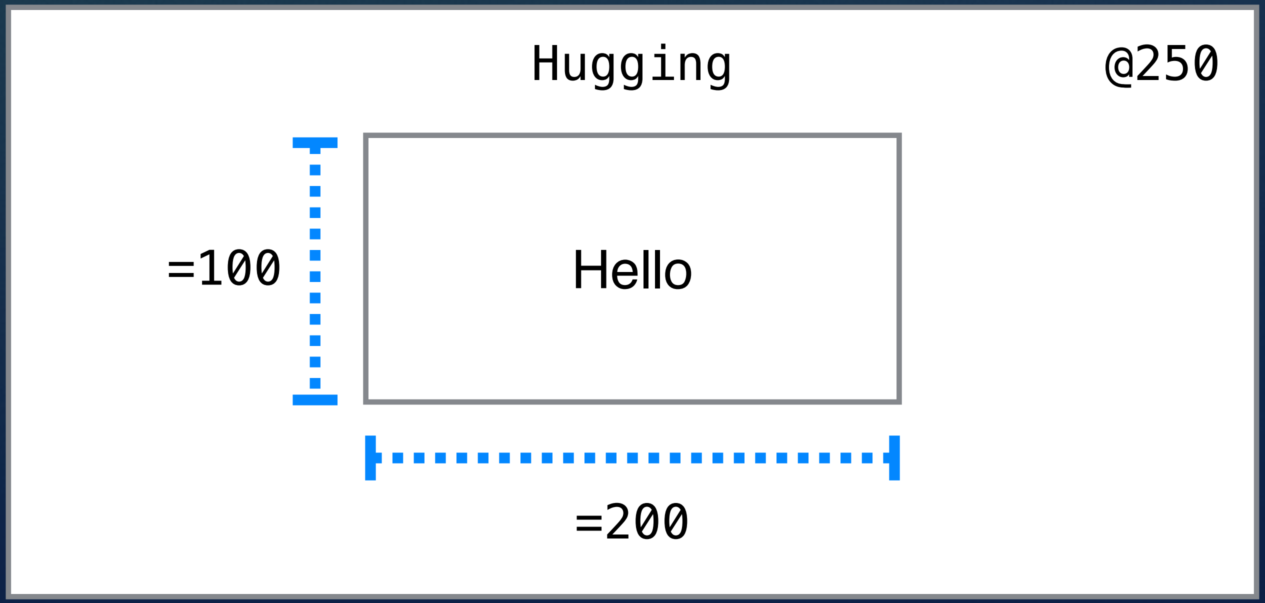
Intrinsic Content Size

- Views know how to layout some content
- Therefore they know the smallest size to display that content
- Implicit constraints defining intrinsic width & height



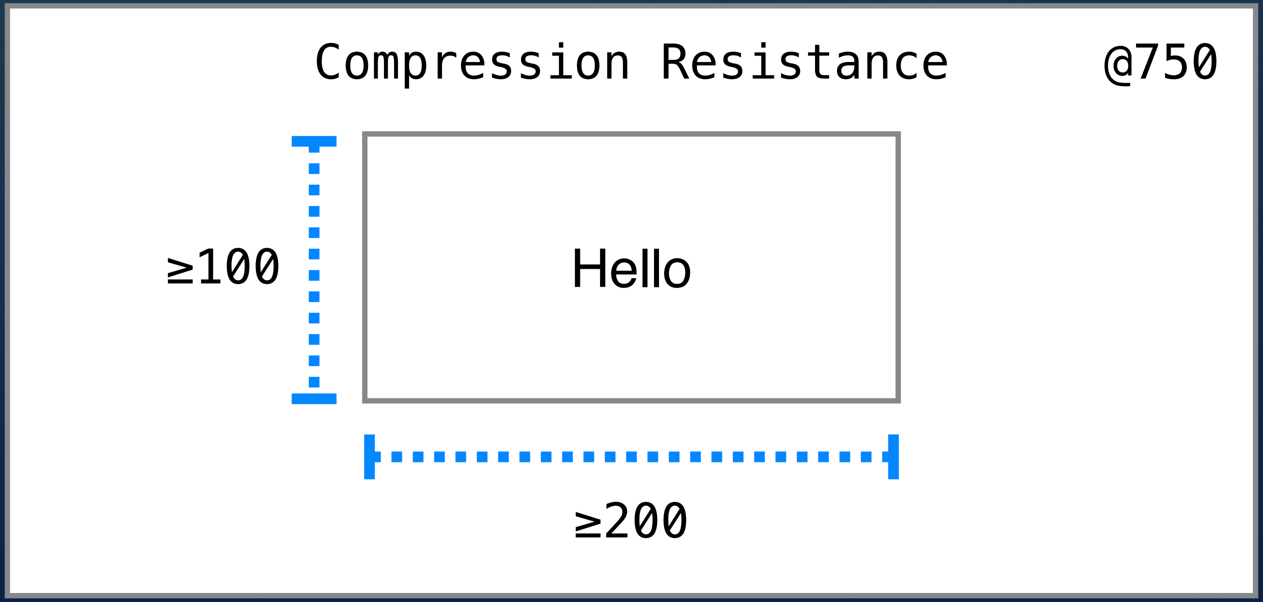
Intrinsic Content Size

- Views know how to layout some content
- Therefore they know the smallest size to display that content
- Implicit constraints defining intrinsic width & height

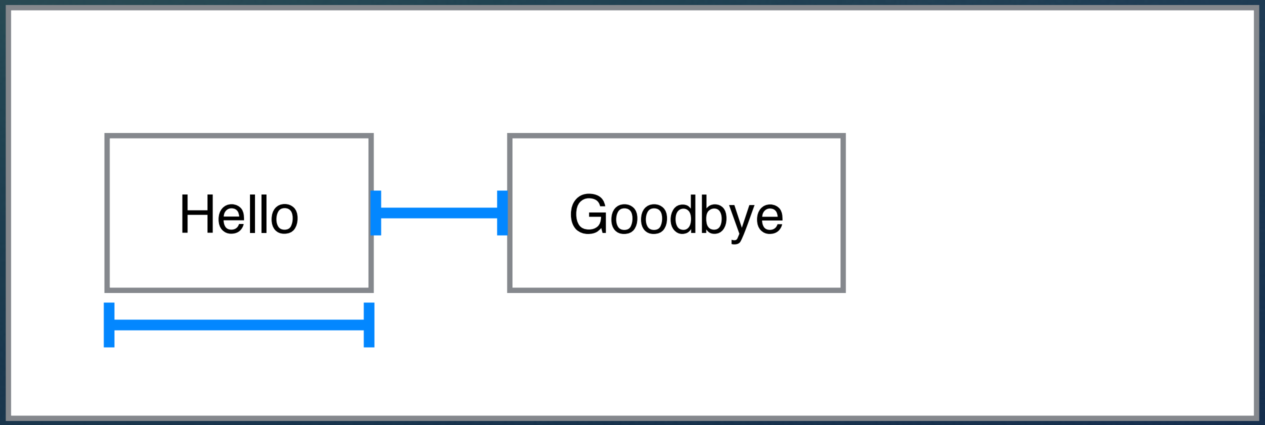


Intrinsic Content Size

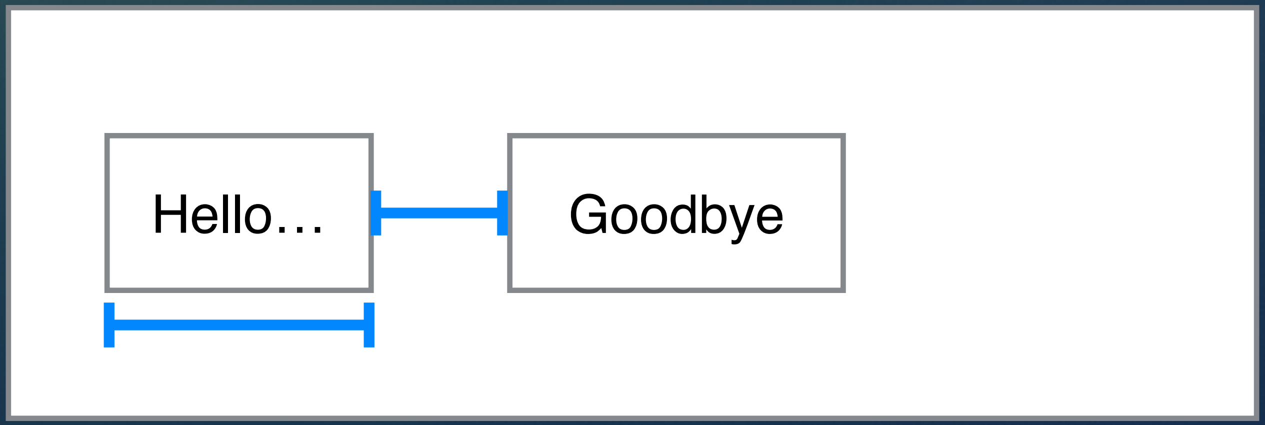
- Views know how to layout some content
- Therefore they know the smallest size to display that content
- Implicit constraints defining intrinsic width & height



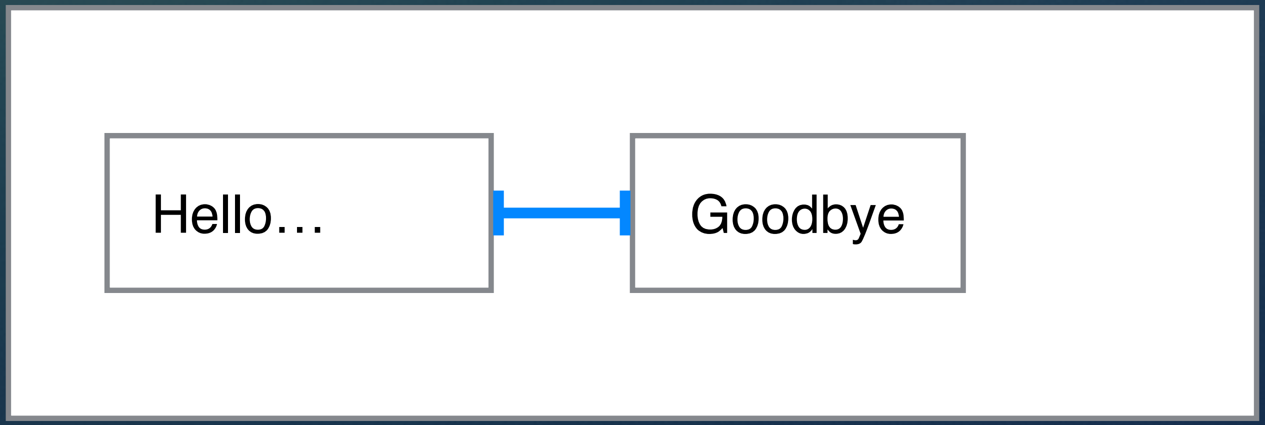
Intrinsic Content Size



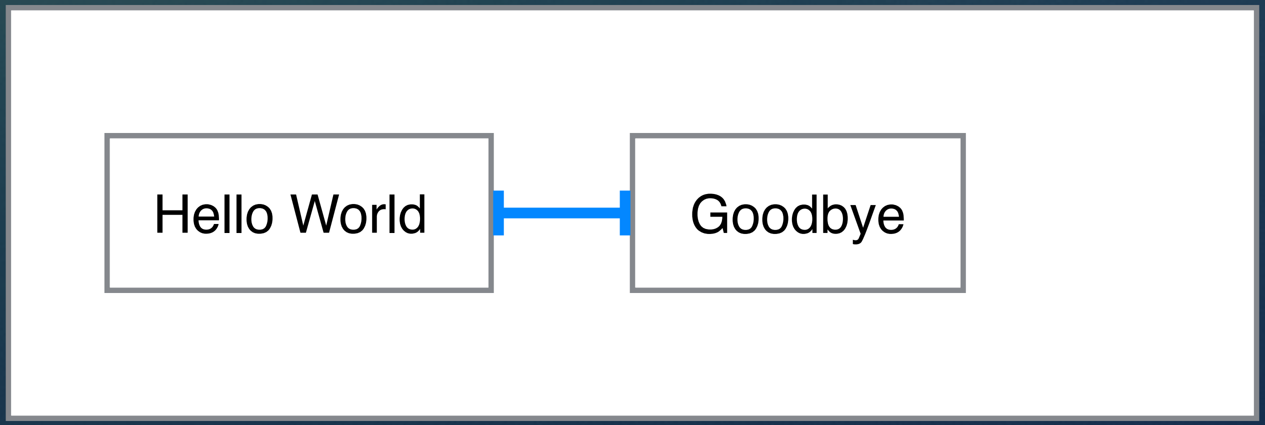
Intrinsic Content Size



Intrinsic Content Size



Intrinsic Content Size



Calculating UITableViewCell Heights

Auto Layout & UITableView

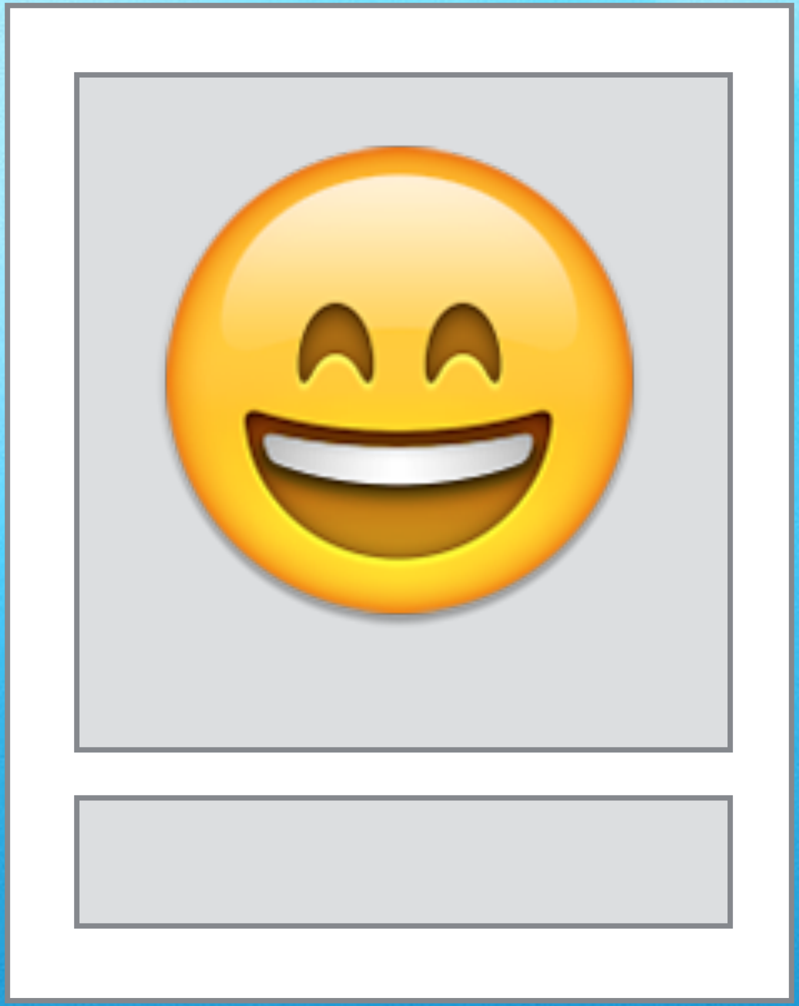
- Create table cells as any view, adding constraints to define height
- Use `-systemLayoutSizeFittingSize:` to return height
- Get cell from table view
    - Set a vertical constraint to have priority 999
- Or use template cell

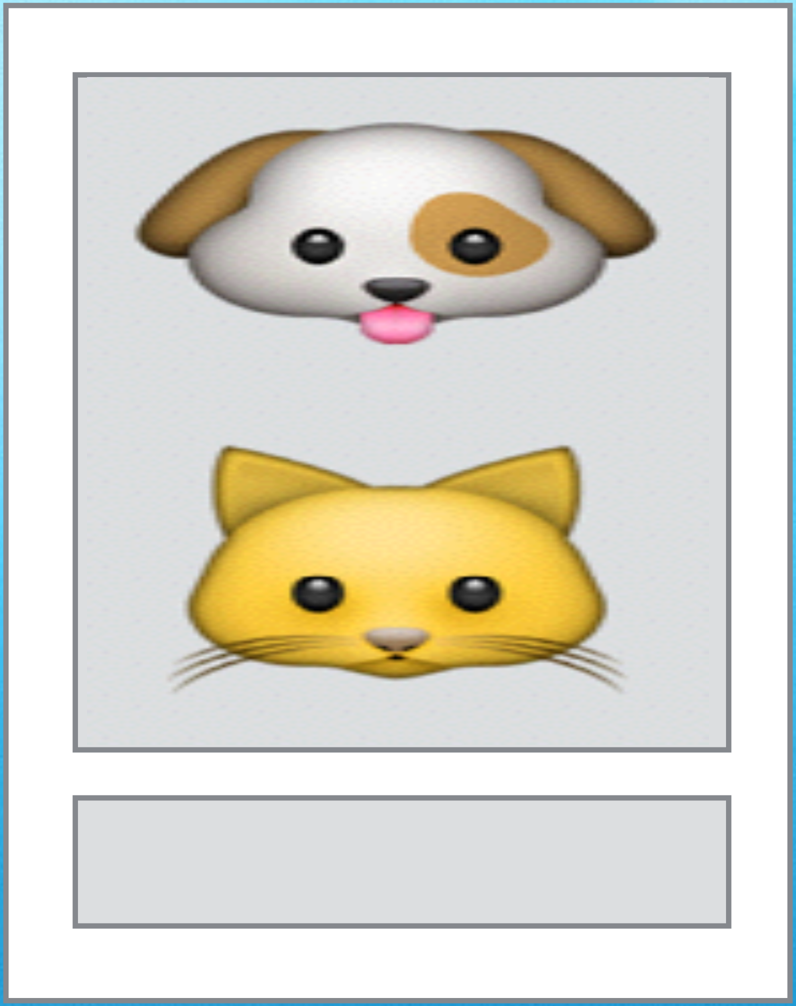
Auto Layout & UITableView

iOS 8

- Create table cells as any view, adding constraints to define height
- Set `estimatedRowHeight` to most common height
- Ensure `rowHeight` is `UITableViewAutomaticDimension`

Auto-resizing UIImageView





AUTORESIZING

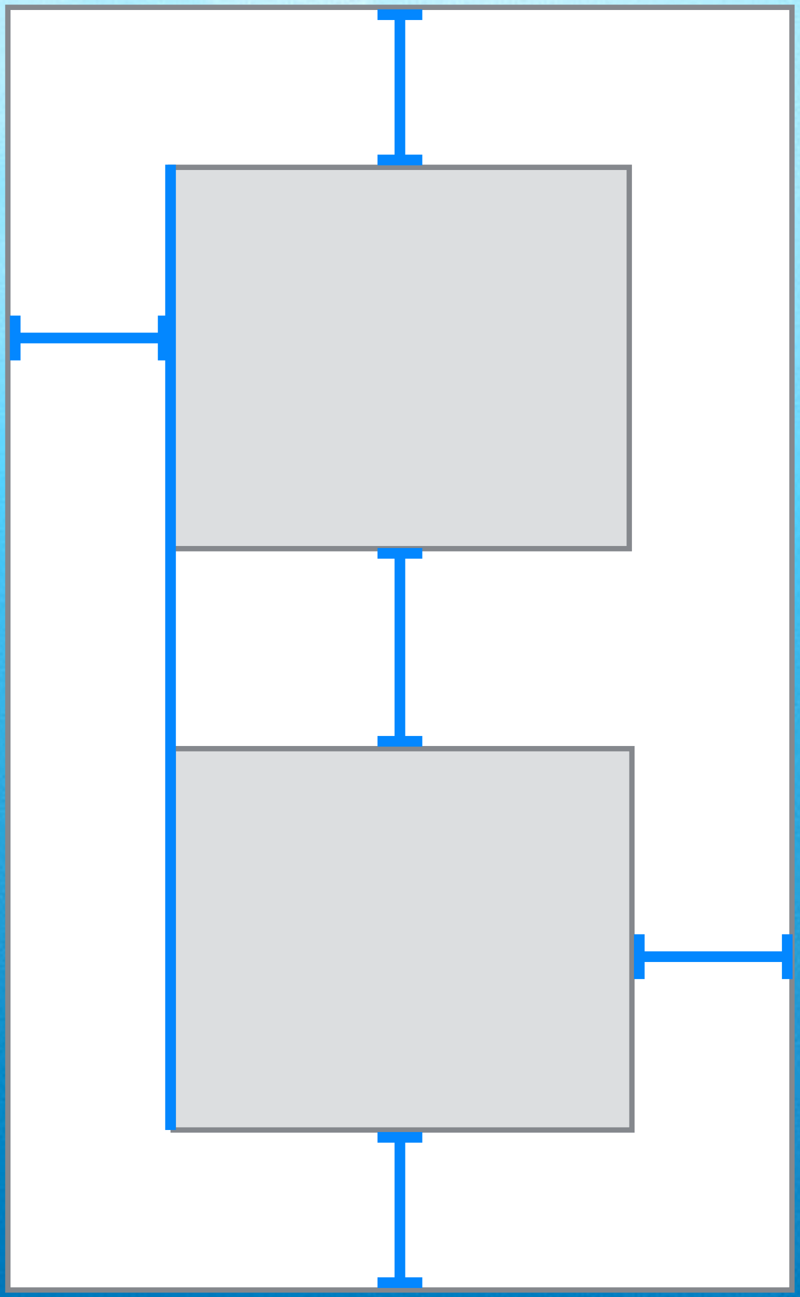
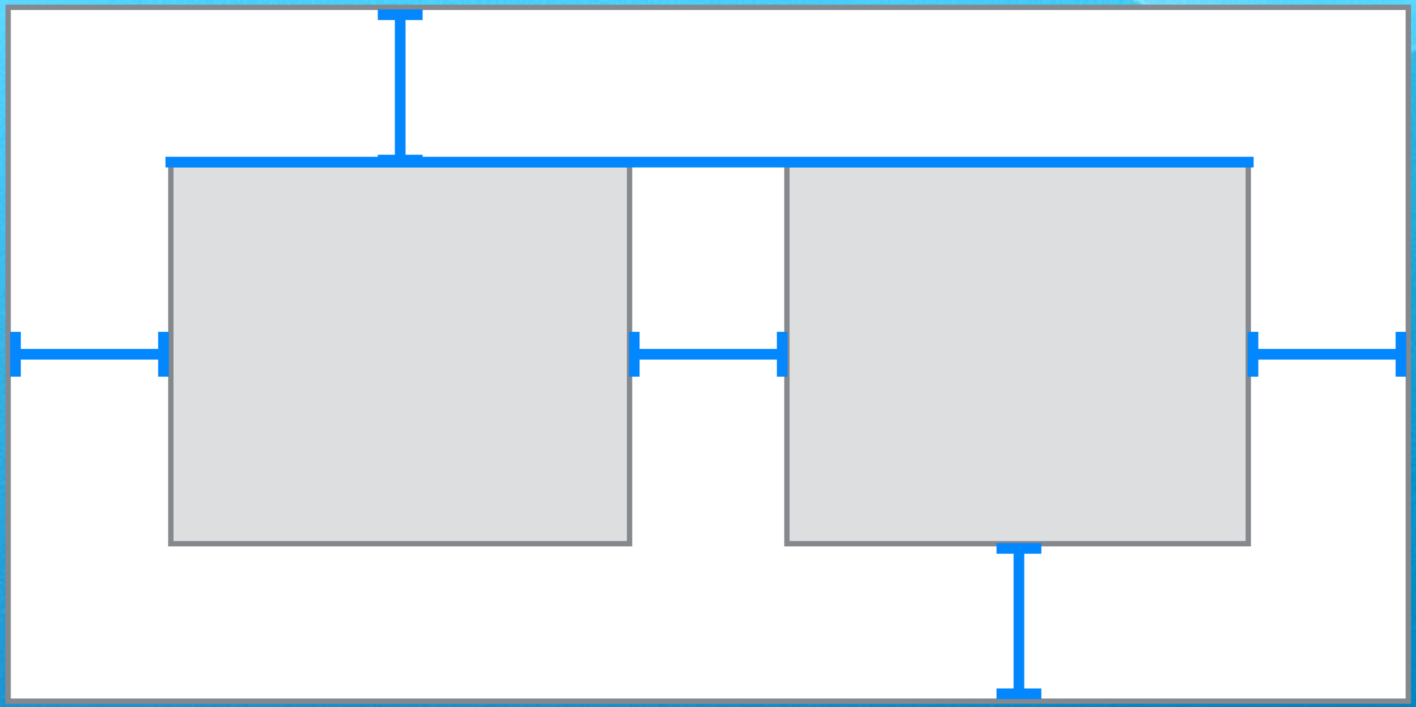
- Subclass UIImageView
- Add following:

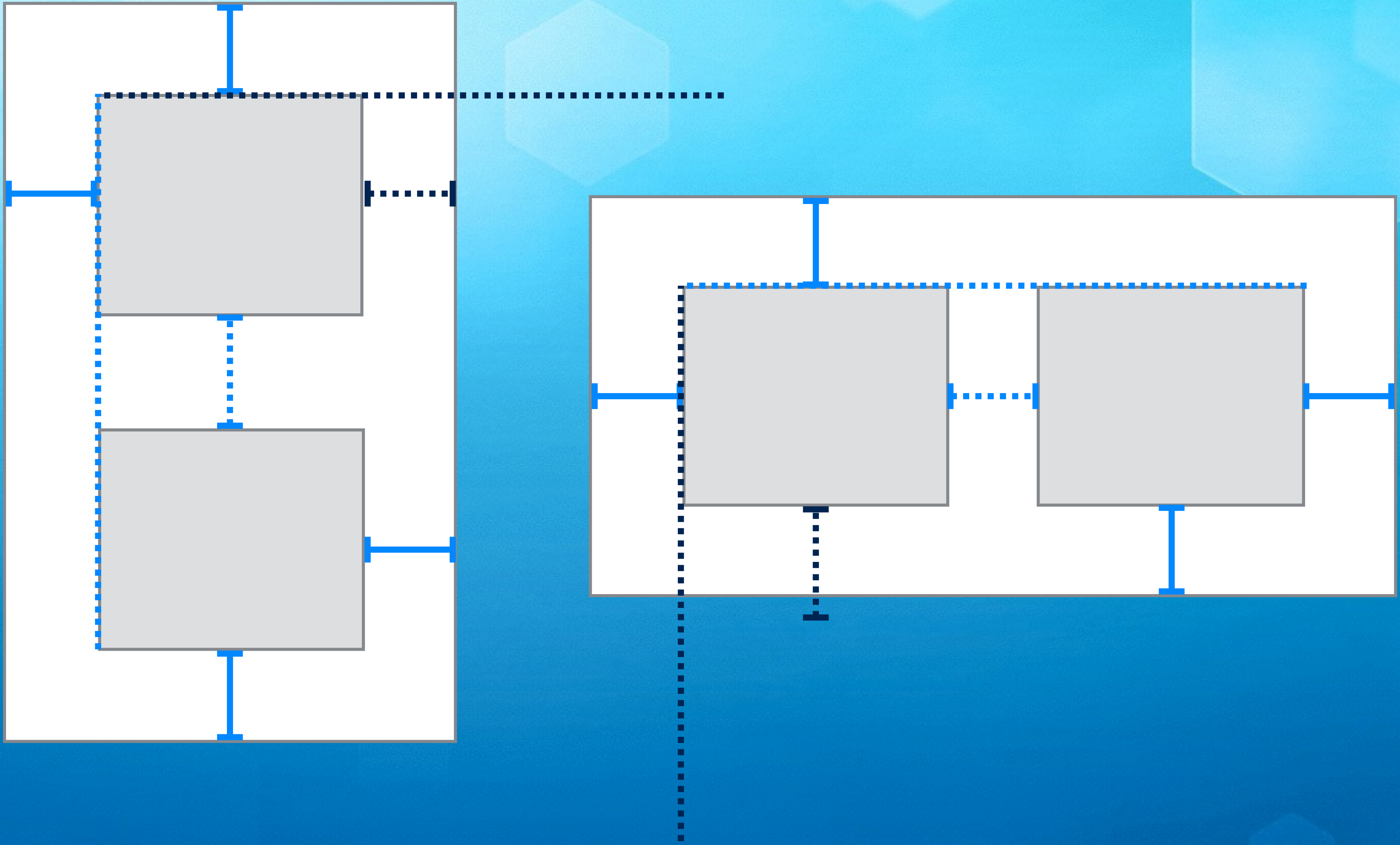
```
- (CGSize)intrinsicContentSize {
  return self.image.size;
}


- (void)setImage:(UIImage *)aImage {
  [super setImage:aImage];
  [self invalidateIntrinsicContentSize];
}
```

AUTORESIZING (WITH LIMITS)

```
- (CGSize)intrinsicContentSize {
  CGSize imageSize = self.image.size;
  CGSize maxSize = self.preferredMaxSize;

  if (imageSize.height > maxSize.height) {
    imageSize.width *= maxSize.height / imageSize.height;
    imageSize.height = maxSize.height;
  }
  if (imageSize.width > maxSize.width) {
    imageSize.height *= maxSize.width / imageSize.width;
    imageSize.width = maxSize.width;
  }
  return imageSize;
}
```

Switching Orientation



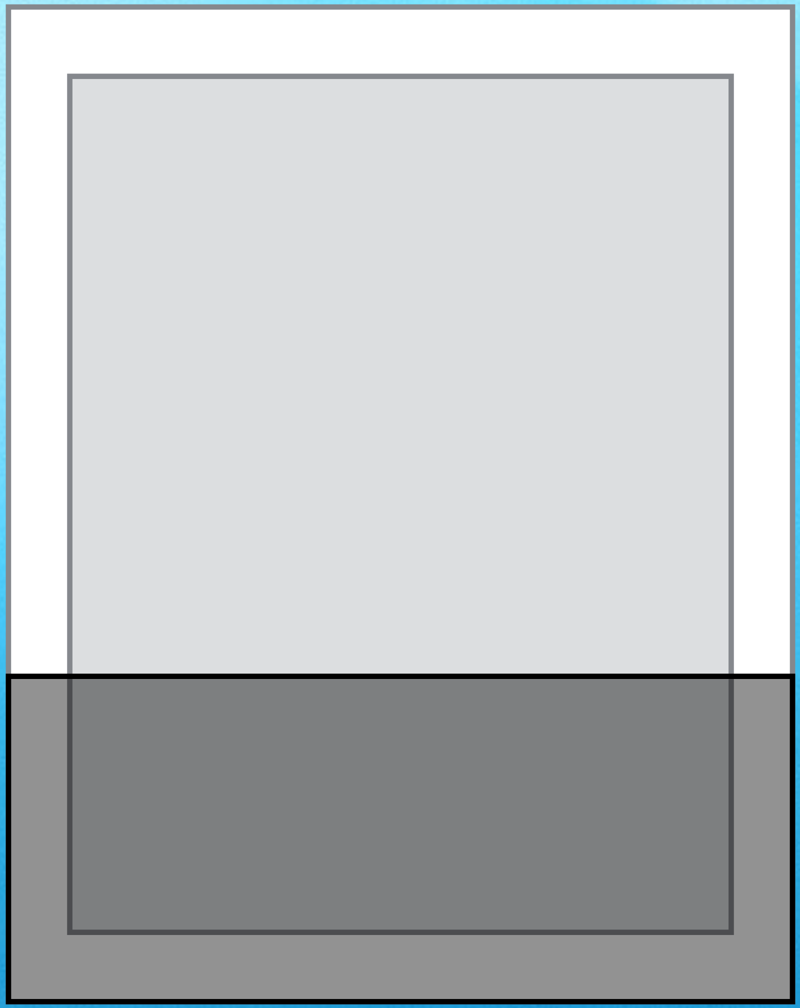
Disabling/Enabling Constraints

- Make constraints optional
- Set constraint priorities to 999 to enable
- Set to 1 to disable

Disabling/Enabling Constraints iOS 8

- New `active` property
- `+[NSLayoutConstraint (de)activateConstraints:]` for bulk changes
- Use NIBs with size classes

Animation

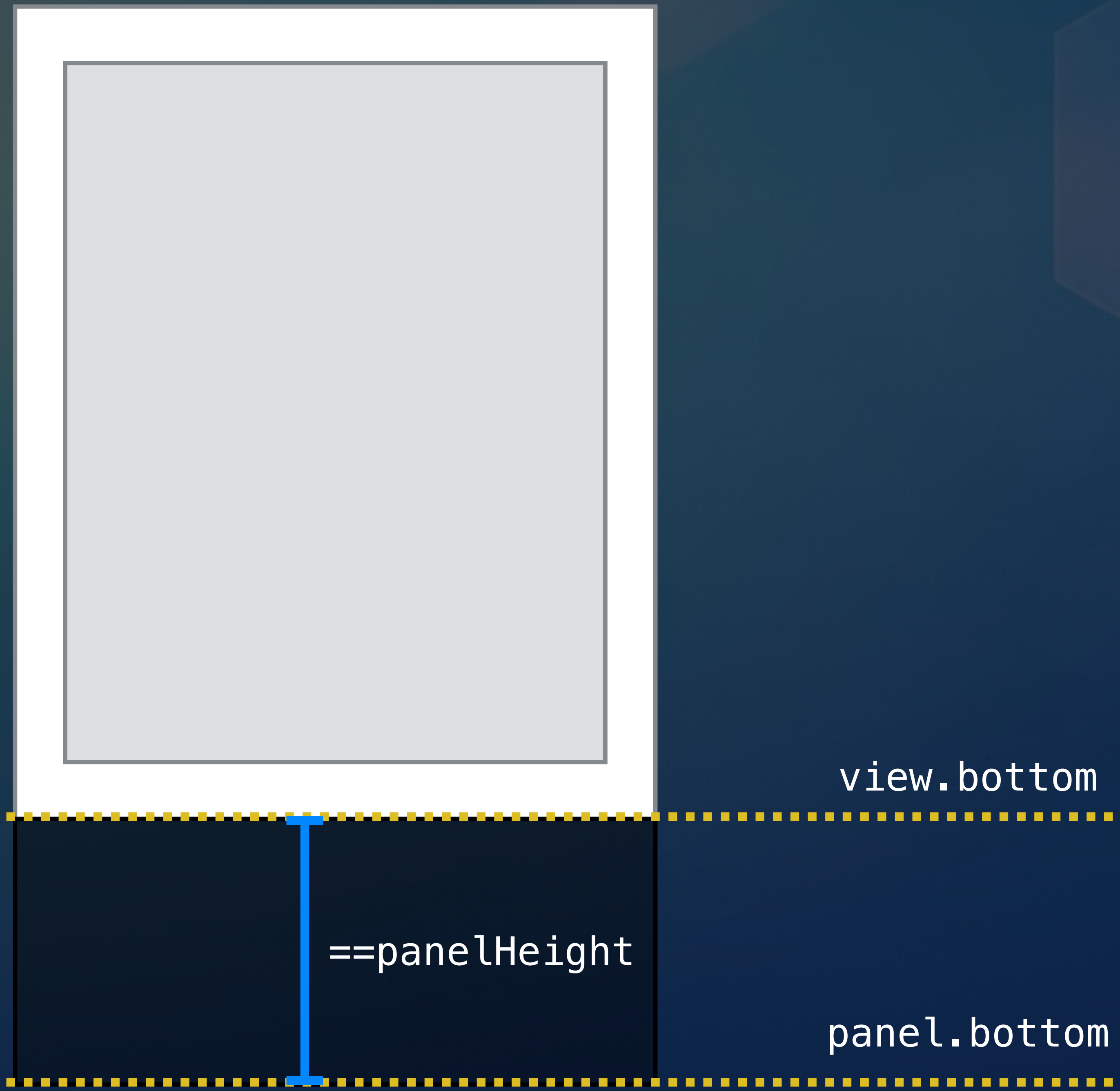


Frame Based Animation

```
CGFloat panelHeight = 150;
[panel setFrame:CGRectMake(0,
                           CGRectGetHeight(view.frame),
                           CGRectGetWidth(view.frame),
                           panelHeight)];

[view addSubview:panel];


[UIView animateWithDuration:0.5 animations:^{
  CGFloat y = CGRectGetHeight(view.frame) - panelHeight;
  [panel setFrame:CGRectMake(0,
                             y,
                             CGRectGetWidth(view.frame),
                             panelHeight)];
}];
```

Auto Layout Based Animation

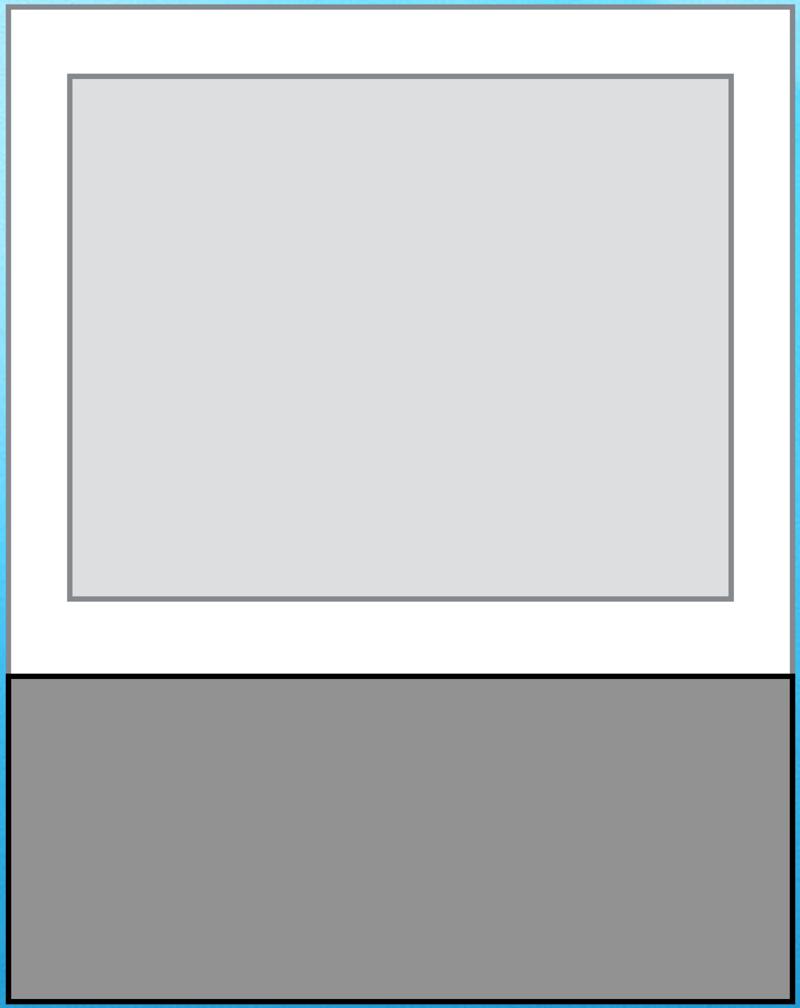
```
CGFloat panelHeight = 150;
[view addSubview:panel];
[view addConstraints:[NSLayoutConstraint constraintWithVisualFormat:@“|[panel]|”
                                                              options:0
                                                              metrics:nil
                                                                views:@{@“panel”:panel}];
[view addConstraints:[NSLayoutConstraint constraintWithVisualFormat:@“V:[panel(==height)]"
                                                              options:0
                                                              metrics:@{@"height":panelHeight}
                                                                views:@{@“panel”:panel}];
```



```
panel.bottom = view.bottom + panelHeight
```

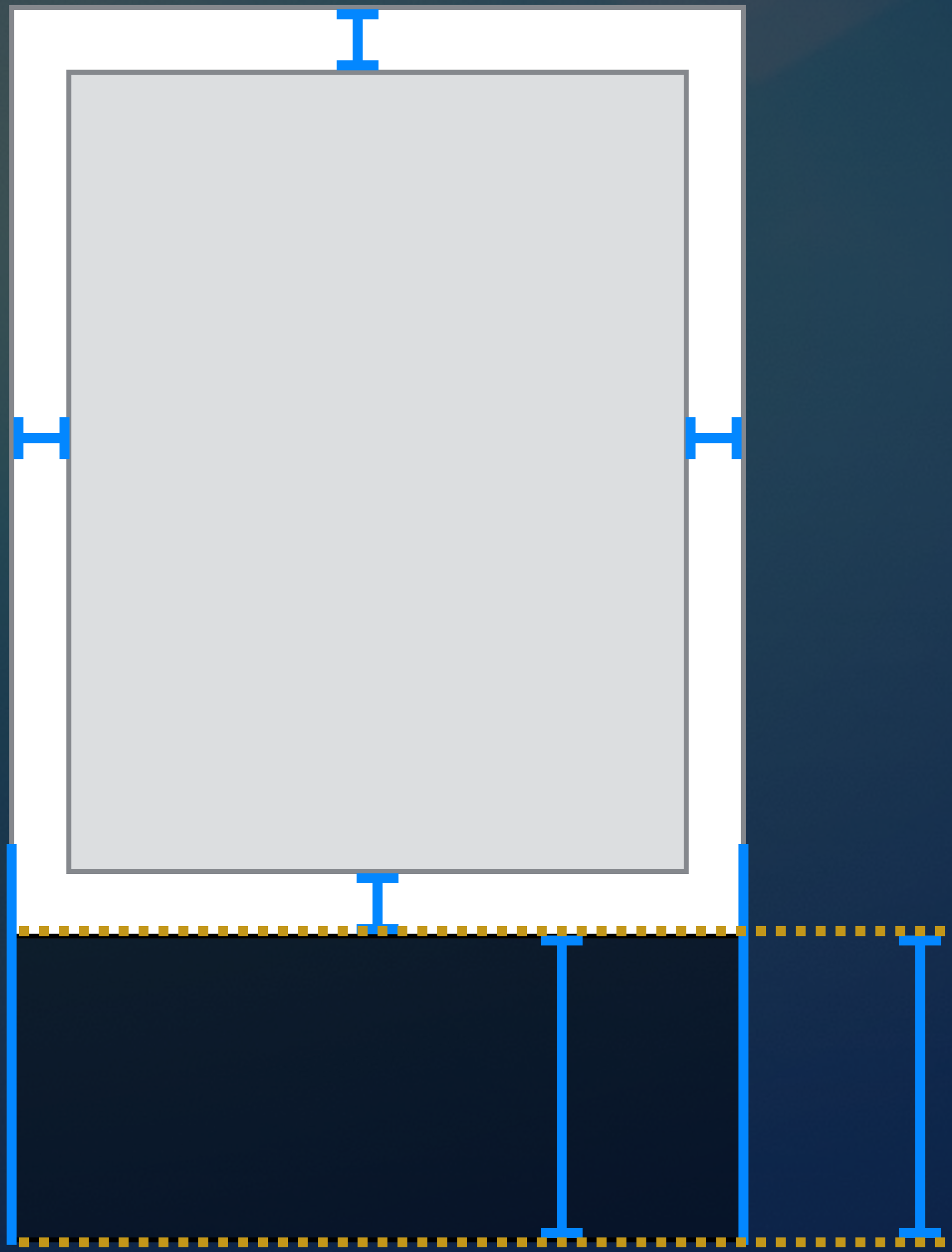
Auto Layout Based Animation

```
CGFloat panelHeight = 150;
[view addSubview:panel];
[view addConstraints:[NSLayoutConstraint constraintWithVisualFormat:@“|[panel]|”
                                                            options:0
                                                            metrics:nil
                                                              views:@{@“panel”:panel}];
[view addConstraints:[NSLayoutConstraint constraintWithVisualFormat:@“V:[panel(==height)]"
                                                            options:0
                                                            metrics:@{@"height":panelHeight}
                                                              views:@{@“panel”:panel}];
id bottom = [NSLayoutConstraint constraintWithItem:panel
                                         attribute:NSLayoutAttributeBottom
                                         relatedBy:NSLayoutRelationEqual
                                            toItem:view
                                         attribute:NSLayoutAttributeBottom
                                        multiplier:1
                                          constant:panelHeight];
[view addConstraint:bottom];
[view layoutIfNeeded];
[UIView animateWithDuration:0.5 animations:^{
  [bottom setConstant:0];
  [view layoutIfNeeded];
}];
```



Frame Based Animation

```
CGFloat panelHeight = 150;
CGFloat margin = 20;

[UIView animateWithDuration:0.5 animations:^{
  CGFloat viewHeight = CGRectGetHeight(view.frame);
  CGFloat viewWidth = CGRectGetWidth(view.frame);
  CGFloat panelHeight = CGRectGetHeight(panel.frame);

  CGFloat panelY = viewHeight - panelHeight;
  [panel setFrame:CGRectMake(0, panelY, viewWidth, panelHeight)];

  CGFloat subviewWidth = viewWidth - (margin * 2)
  CGFloat subviewHeight = viewHeight - panelHeight - (margin * 2);
  [subview setFrame:CGRectMake(margin, margin, subviewWidth, subviewHeight)];
}];
```



Auto Layout Based Animation

```
[UIView animateWithDuration:0.5 animations:^{
  [bottomConstraint setConstant:0];
  [view layoutIfNeeded];
}];
```

```
[UIView animateWithDuration:0.5 animations:^{
  [bottomConstraint setConstant:CGRectGetHeight(panel.frame)];
  [view layoutIfNeeded];
}];
```

Where To Find Me

- I code (mcubedsw.com)
- I blog (pilky.me)
- I tweet (@pilky)
- I'm writing a book (autolayoutguide.com)

Questions?