# Best Practices for Mastering Auto Layout

## for OS X and iOS

Session 228

**Peter Ammon**
AppKit Engineer

Make your layouts
simpler to write
simpler to modify
easier to understand

# Auto Layout

# Auto Layout

# Auto Layout

# Auto Layout


Identical APIs!

# Auto Layout

Almost Identical APIs!

# Auto Layout

# Auto Layout


“View”

# Auto Layout

"View"

NSView

UIView

# NSLayoutConstraint

# NSLayoutConstraint

- One new class—NSLayoutConstraint

# NSLayoutConstraint

- One new class—NSLayoutConstraint
- Constraints express geometric properties of views

# NSLayoutConstraint

- One new class—NSLayoutConstraint
- Constraints express geometric properties of views


```
foo.width = 120
```

# NSLayoutConstraint

- One new class—NSLayoutConstraint
- Constraints express geometric properties of views


```
foo.width = 120
```

- Constraints also express geometric relationships between views

# NSLayoutConstraint

- One new class—NSLayoutConstraint
- Constraints express geometric properties of views


```
foo.width = 120
```

- Constraints also express geometric relationships between views


```
foo.width = bar.width
```

# NSLayoutConstraint

- One new class—NSLayoutConstraint
- Constraints express geometric properties of views


```
foo.width = 120
```

- Constraints also express geometric relationships between views


```
foo.width = bar.width
```

- Relationships have a coefficient and a constant

# NSLayoutConstraint

- One new class—NSLayoutConstraint
- Constraints express geometric properties of views


```
foo.width = 120
```

- Constraints also express geometric relationships between views


```
foo.width = bar.width
```

- Relationships have a coefficient and a constant

```
foo.width = bar.width * 2 - 20
```

# NSLayoutConstraint

# NSLayoutConstraint

- Constraints can be equalities or inequalities

# NSLayoutConstraint

- Constraints can be equalities or inequalities


```
foo.width >= 120
```

# NSLayoutConstraint

- Constraints can be equalities or inequalities


```
foo.width >= 120
```

- Constraints have priorities

# NSLayoutConstraint

- Constraints can be equalities or inequalities


```
foo.width >= 120
```

- Constraints have priorities


```
foo.width = 120 with priority 500
foo.width = 75 with priority 1000
```

# NSLayoutConstraint

- Constraints can be created three ways
  - Interface Builder
  - Visual format language
  - Base API
- Prefer this order

# Auto Layout

- Thinking in constraints
  - Transitioning to constraints
- Debugging constraint-based layouts
  - Ambiguity
  - Unsatisfiability
  - Reading log messages
- Unleashing the power of constraints
  - Animation
  - Writing a custom control
  - Internationalization

# Auto Layout

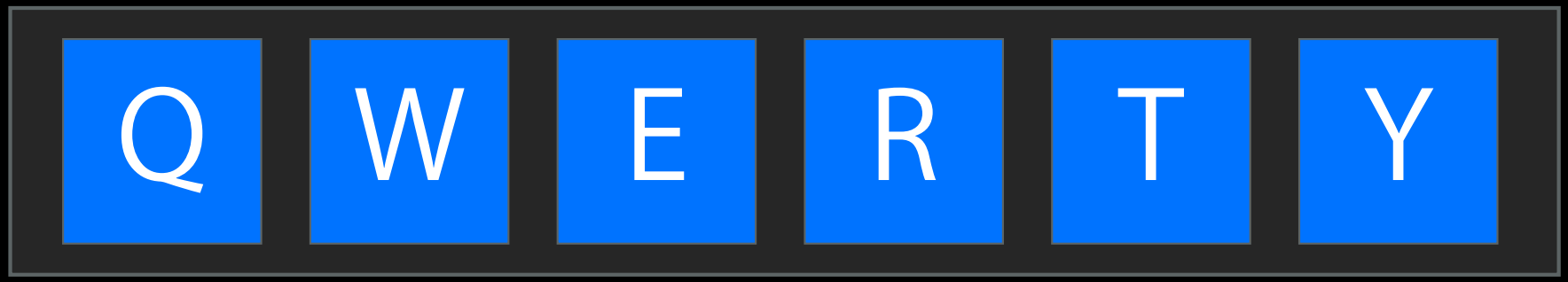
- Thinking in constraints
  - Transitioning to constraints
- Debugging constraint-based layouts
  - Ambiguity
  - Unsatisfiability
  - Reading log messages
- Unleashing the power of constraints
  - Animation
  - Writing a custom control
  - Internationalization

# Thinking in Constraints

- Auto Layout can be used like springs and struts
- But you get the most benefits from a shift in thinking
- Let your layouts becomes declarative

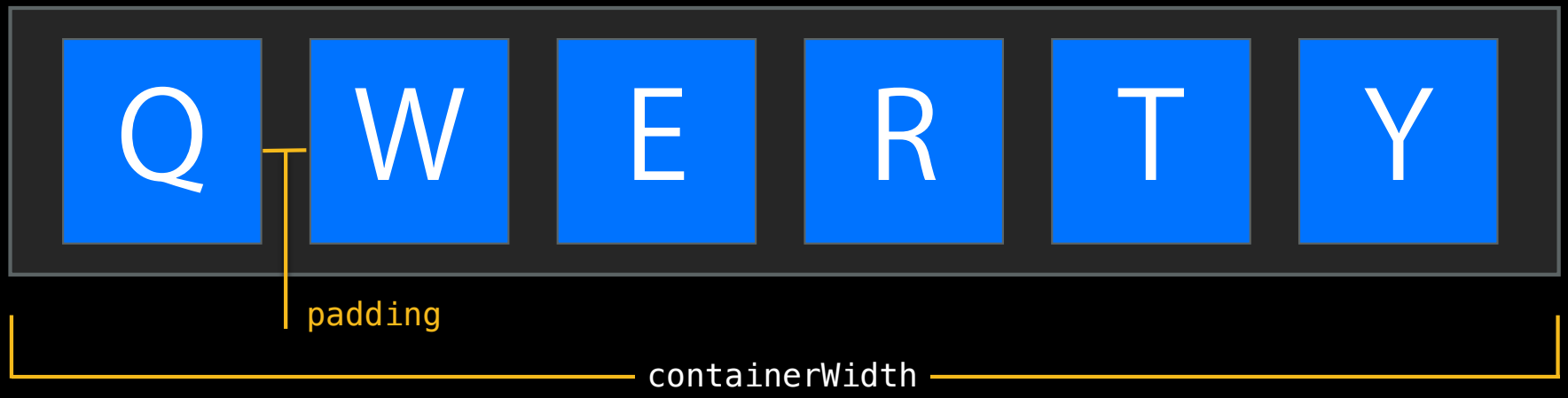
# Thinking in Constraints

# Thinking in Constraints



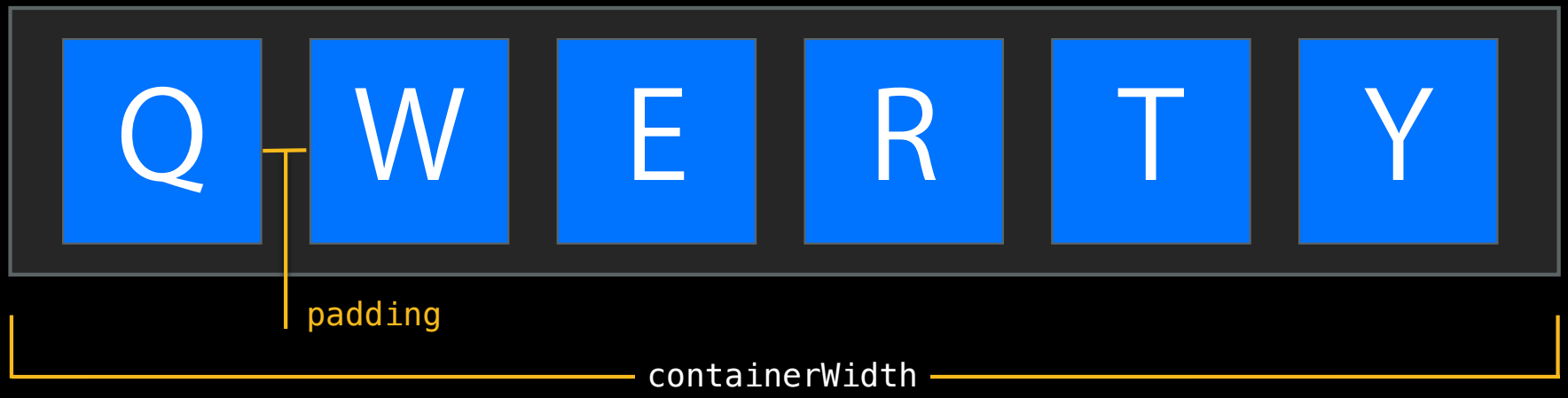
# Thinking in Constraints

# Thinking in Constraints



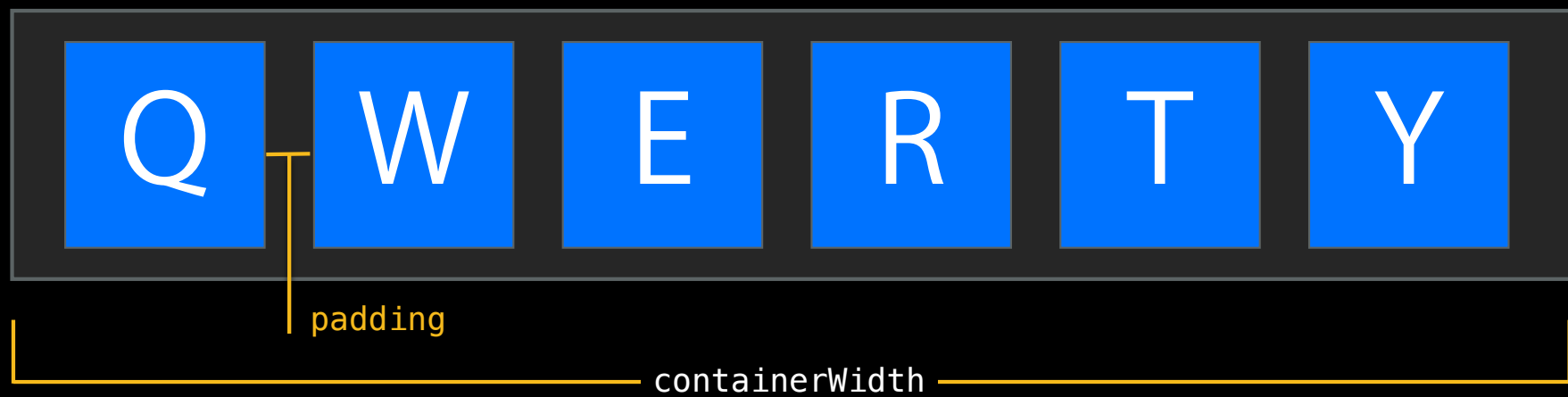
# Thinking in Constraints

## Springs and Struts



# Thinking in Constraints

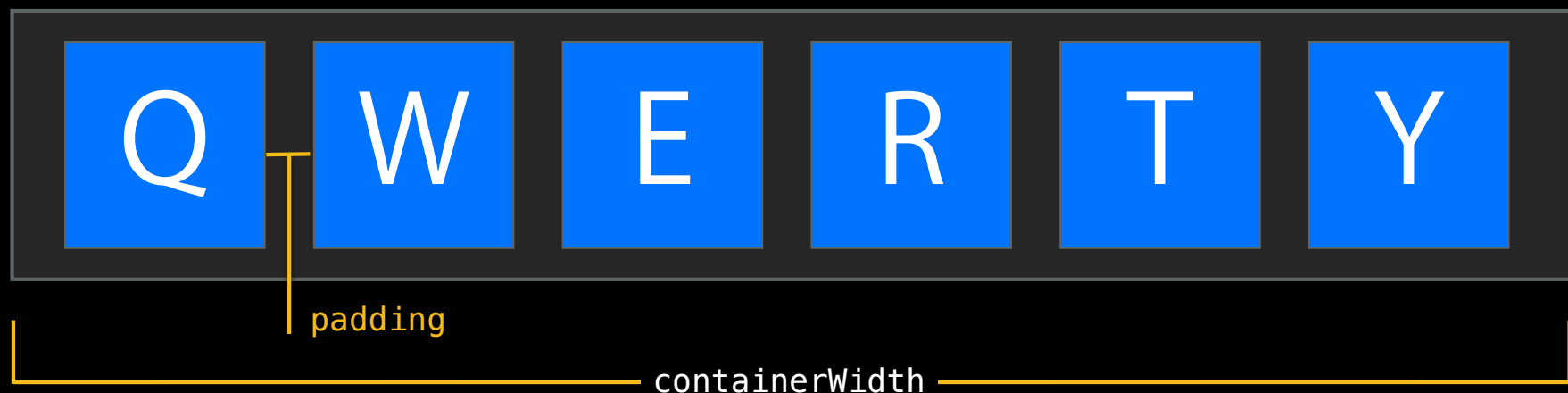
## Springs and Struts




```
keyWidth = containerWidth / keyCount
               - padding * (keyCount + 1) / keyCount
```

# Thinking in Constraints

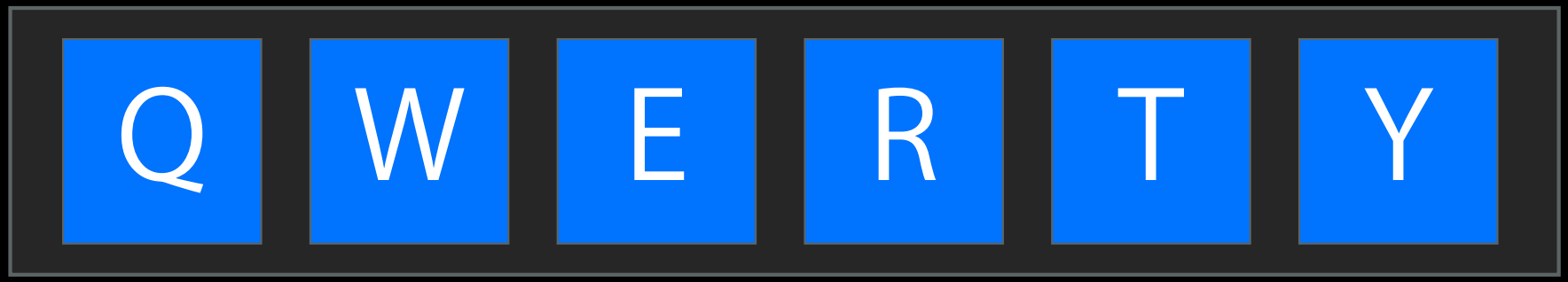
## Springs and Struts




```
keyWidth = containerWidth / keyCount
               - padding * (keyCount + 1) / keyCount

keyOffset = padding +
                keyIndex * (keyWidth + padding)
```

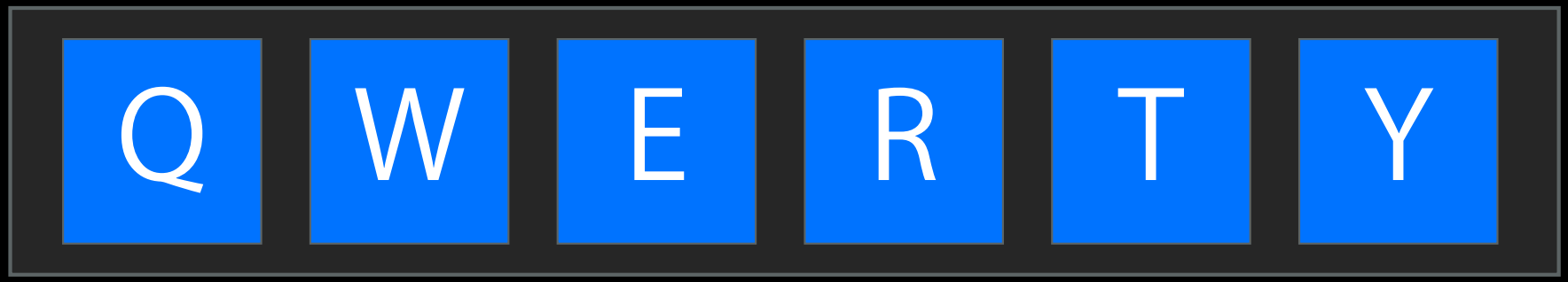
# Thinking in Constraints

Springs and Struts



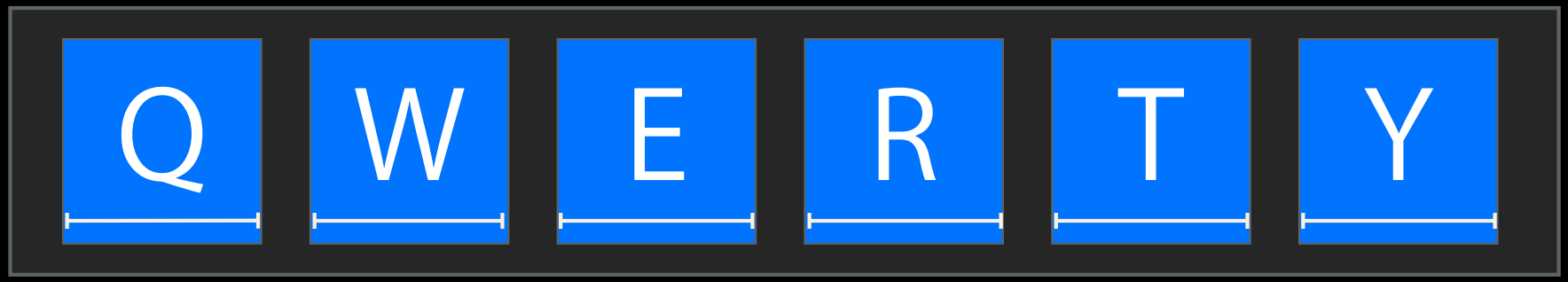
# Thinking in Constraints

Auto Layout



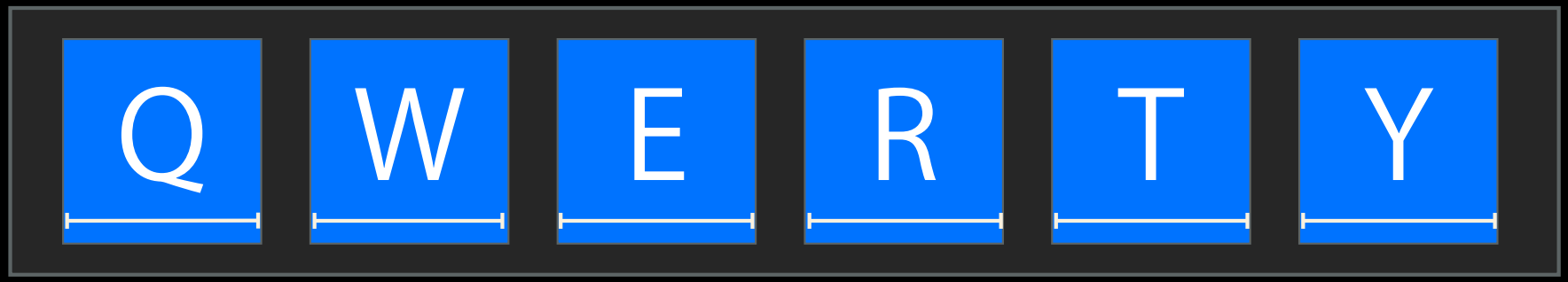
# Thinking in Constraints

Auto Layout



# Thinking in Constraints

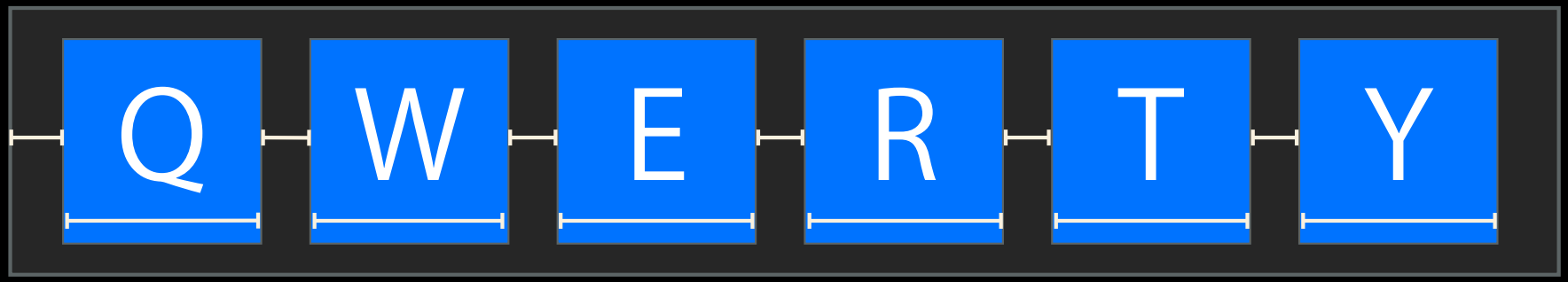
Auto Layout




```
Q.width = container.width / keyCount
                - padding * (keyCount + 1) / keyCount
```

# Thinking in Constraints

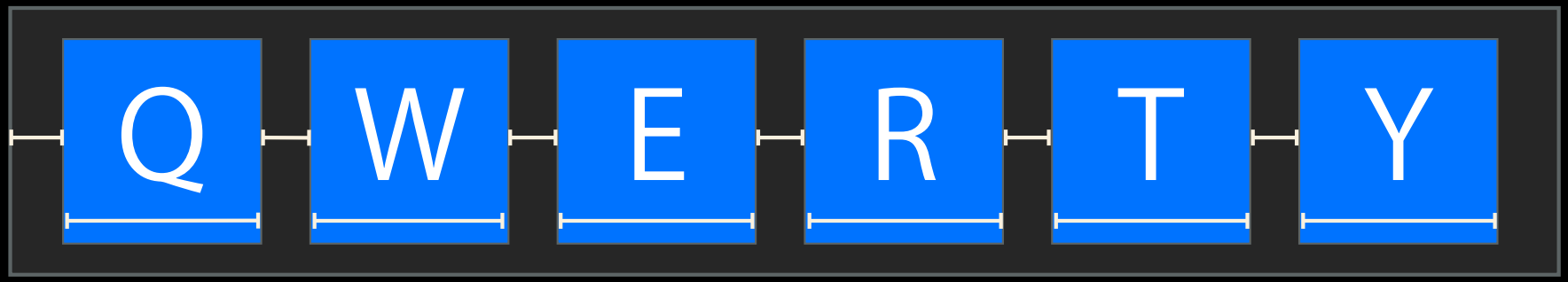
Auto Layout




```
Q.width = container.width / keyCount
                - padding * (keyCount + 1) / keyCount
```

# Thinking in Constraints

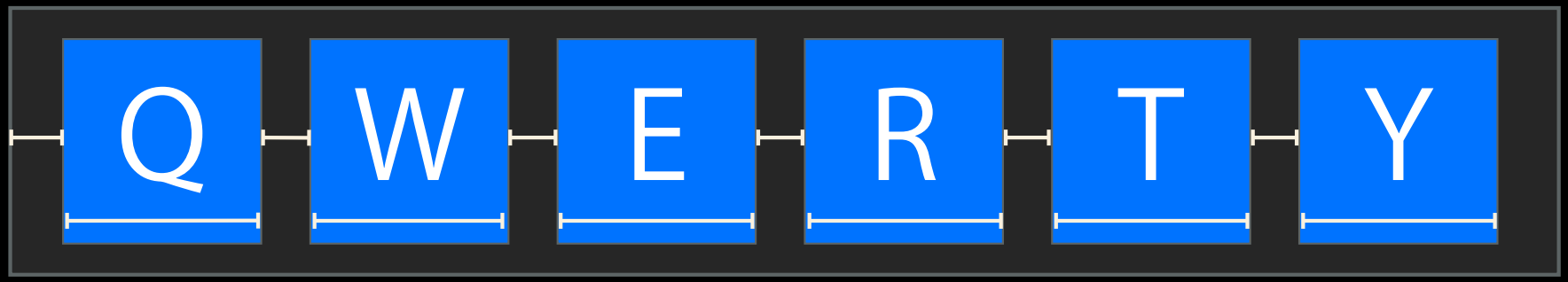
Auto Layout




```
Q.width = container.width / keyCount
                - padding * (keyCount + 1) / keyCount

Q.minX = container.minX + padding
W.minX = Q.maxX + padding
E.minX = W.maxX + padding...
```

# Thinking in Constraints

Auto Layout

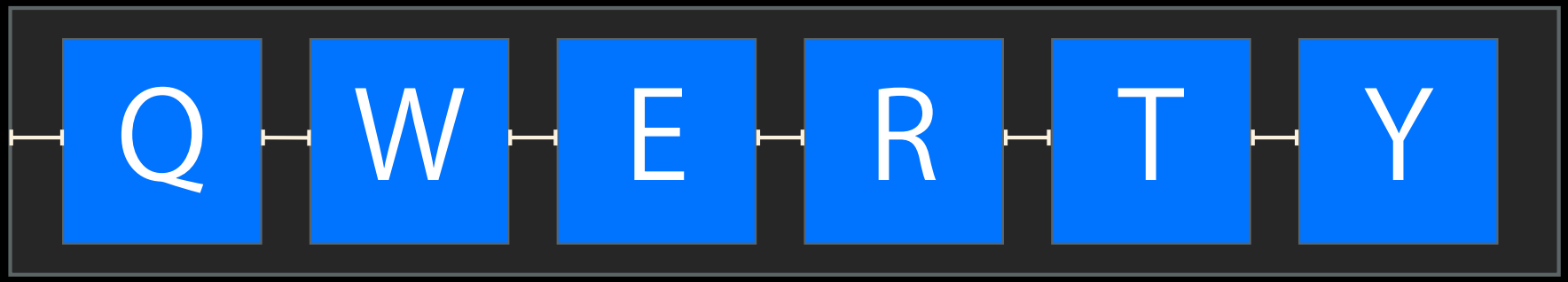



```
Q.width = container.width / keyCount
                - padding * (keyCount + 1) / keyCount

Q.minX = container.minX + padding
W.minX = Q.maxX + padding
E.minX = W.maxX + padding...
```

Relationships

# Thinking in Constraints

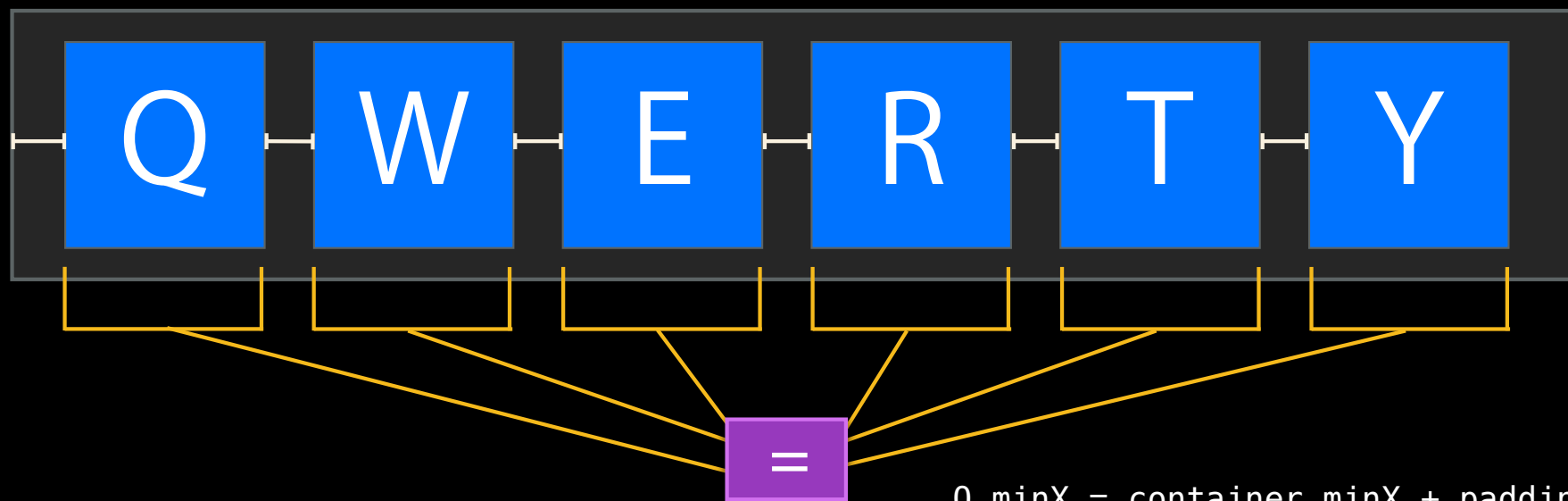
Auto Layout




```
Q.minX = container.minX + padding
W.minX = Q.maxX + padding
E.minX = W.maxX + padding...
```

# Thinking in Constraints

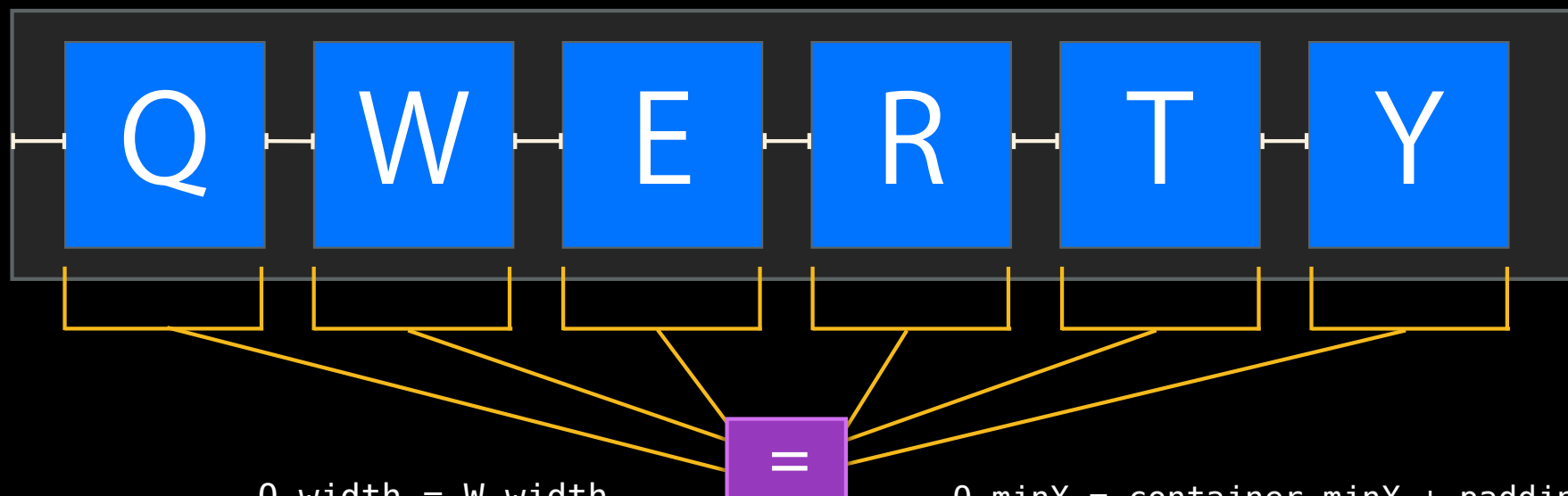
Auto Layout




```
Q.minX = container.minX + padding
W.minX = Q.maxX + padding
E.minX = W.maxX + padding...
```

# Thinking in Constraints

Auto Layout

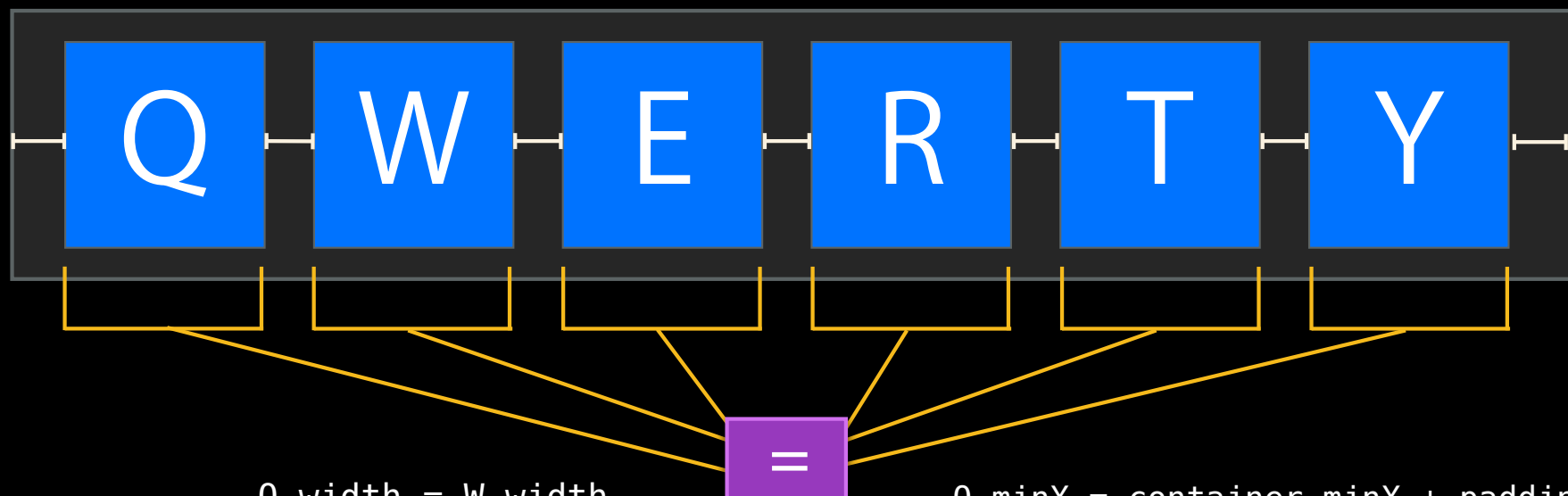



```
Q.width = W.width
W.width = E.width
...
```

```
Q.minX = container.minX + padding
W.minX = Q.maxX + padding
E.minX = W.maxX + padding...
```

# Thinking in Constraints

Auto Layout

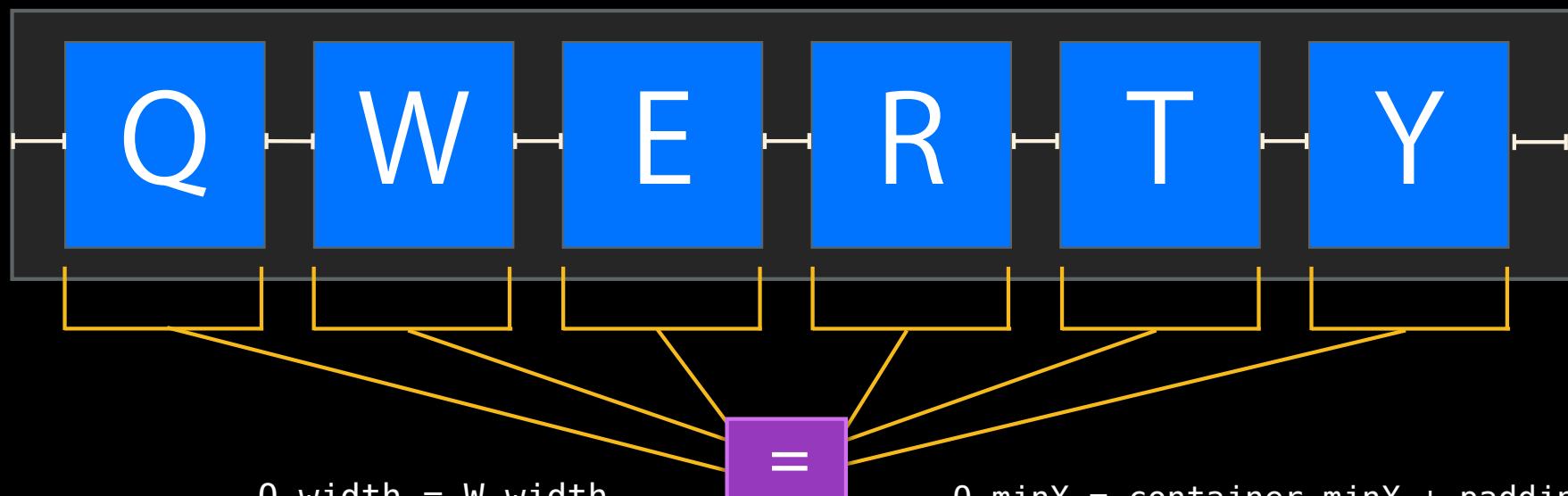



```
Q.width = W.width
W.width = E.width
...
```

```
Q.minX = container.minX + padding
W.minX = Q.maxX + padding
E.minX = W.maxX + padding...
```

# Thinking in Constraints

Auto Layout




```
Q.width = W.width
W.width = E.width
...
```

```
Q.minX = container.minX + padding
W.minX = Q.maxX + padding
E.minX = W.maxX + padding...

container.maxX = Y.maxX + padding
```

*Demo*

# Thinking in Constraints

- Layout becomes distributed
- Decompose sophisticated layouts into components
- Each component contributes the constraints it cares about
- Layout becomes “owned”

# Thinking in Constraints

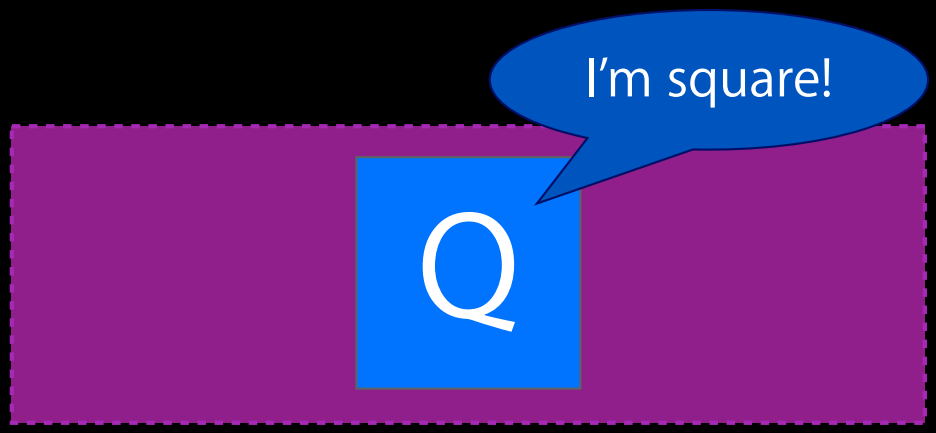
- Layout becomes distributed
- Decompose sophisticated layouts into components
- Each component contributes the constraints it cares about
- Layout becomes “owned”

# Thinking in Constraints

- Layout becomes distributed
- Decompose sophisticated layouts into components
- Each component contributes the constraints it cares about
- Layout becomes “owned”

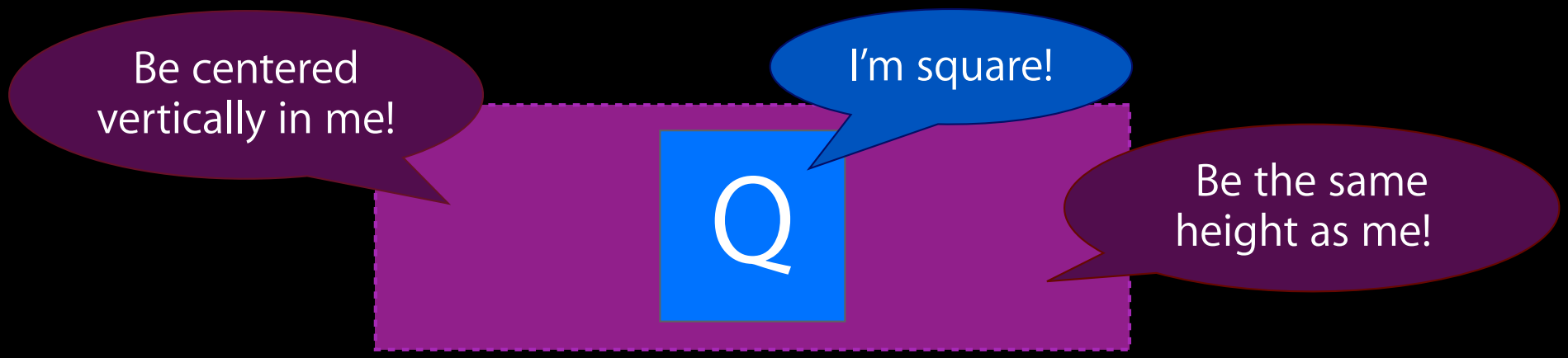
# Thinking in Constraints

- Layout becomes distributed
- Decompose sophisticated layouts into components
- Each component contributes the constraints it cares about
- Layout becomes “owned”



# Thinking in Constraints

- Layout becomes distributed
- Decompose sophisticated layouts into components
- Each component contributes the constraints it cares about
- Layout becomes "owned"



# Auto Layout

- Thinking in constraints
  - Transitioning to constraints
- Debugging constraint-based layouts
  - Ambiguity
  - Unsatisfiability
  - Reading log messages
- Unleashing the power of constraints
  - Animation
  - Writing a custom control
  - Internationalization

# Migrating from Springs and Struts

## 1. Plan your attack

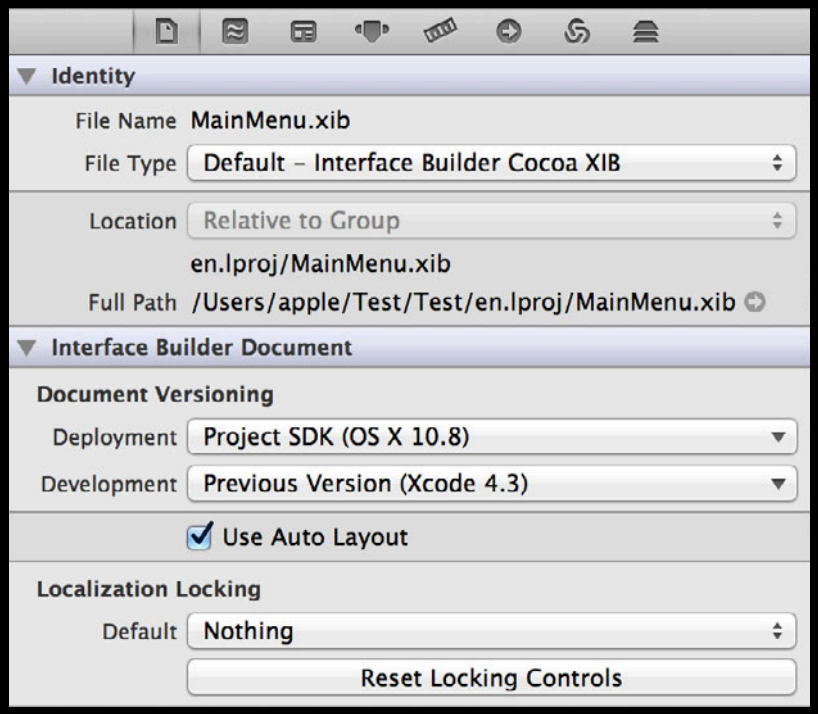
- A partial conversion lets you use Auto Layout just where you need it
  - Some compatibility issues to be aware of
- A full conversion will pay major dividends

# Migrating from Springs and Struts

2. Turn on Auto Layout in your nibs

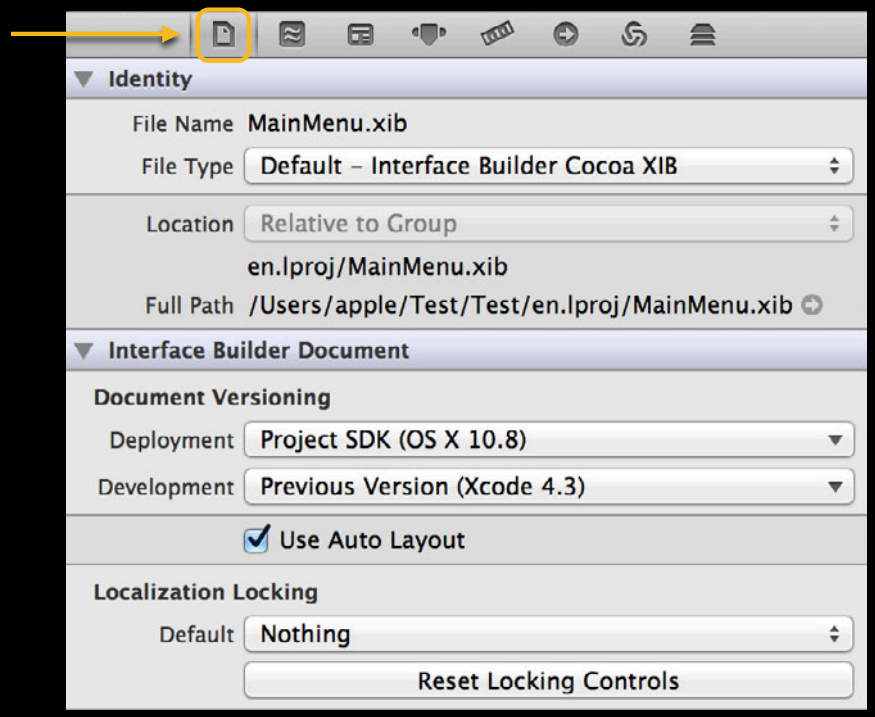
# Migrating from Springs and Struts

## 2. Turn on Auto Layout in your nibs



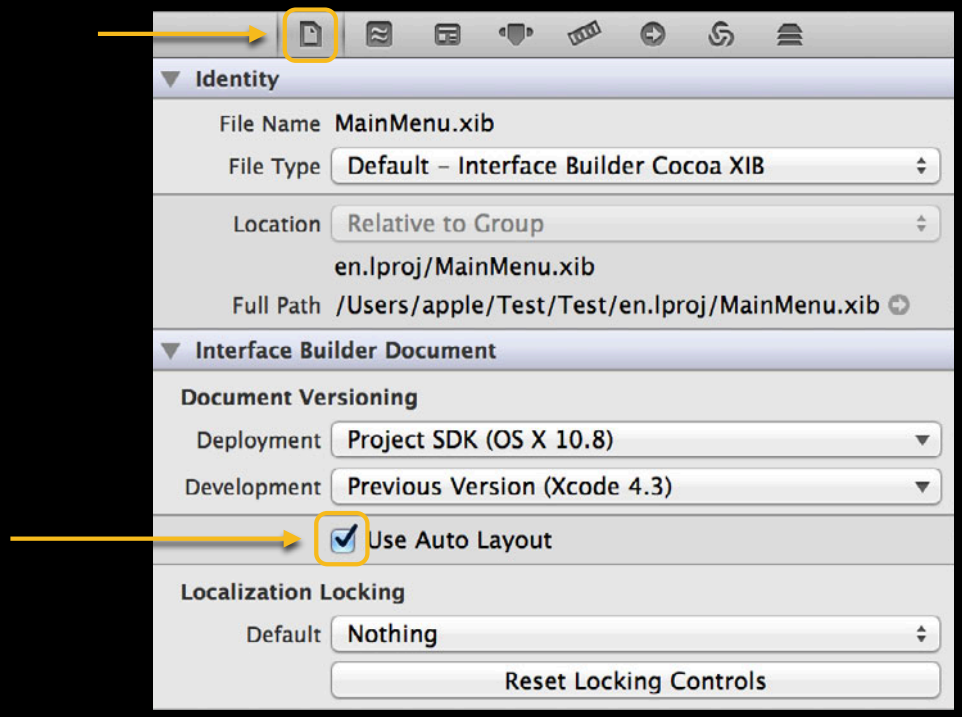
# Migrating from Springs and Struts

## 2. Turn on Auto Layout in your nibs



# Migrating from Springs and Struts

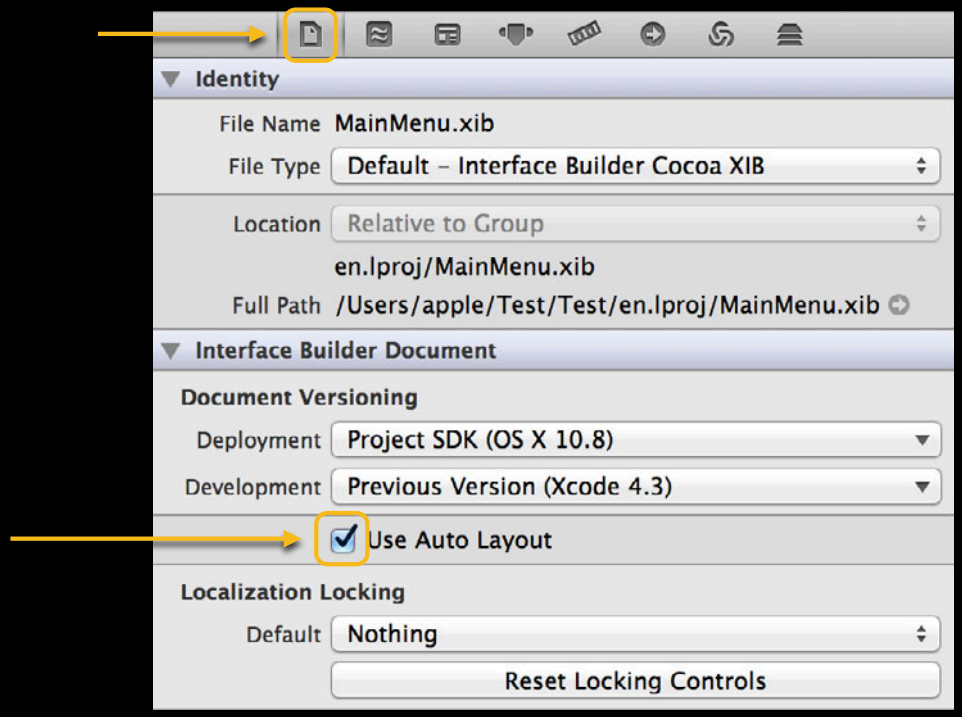
## 2. Turn on Auto Layout in your nibs



# Migrating from Springs and Struts

## 2. Turn on Auto Layout in your nibs

- Create the constraints you want in IB
- IB will create constraints that reflect your existing layout
- Add to or modify them



# Migrating from Springs and Struts

## 3. Turn off autoresizing mask translation for every view you create programmatically

```
[view setTranslatesAutoresizingMaskIntoConstraints:NO]
```

- If you forget, you'll get unsatisfiable constraint warnings quickly

# Migrating from Springs and Struts

4. Look for places where you perform layout

# Migrating from Springs and Struts

## 4. Look for places where you perform layout

```
- (void)layoutSubviews {…
```

# Migrating from Springs and Struts

## 4. Look for places where you perform layout

```
- (void)layoutSubviews {…

[view setFrame:rect]
[view setFrameSize:size]
[view setFrameOrigin:point]
```

# Migrating from Springs and Struts

## 4. Look for places where you perform layout

```
- (void)layoutSubviews {…

[view setFrame:rect]
[view setFrameSize:size]
[view setFrameOrigin:point]
```

They all have to go!

# Migrating from Springs and Struts

## 4. Look for places where you perform layout

```
- (void)layoutSubviews {…

[view setFrame:rect]
[view setFrameSize:size]
[view setFrameOrigin:point]
```

They all have to go!

(But what do I replace them with?)

# Migrating from Springs and Struts

# Migrating from Springs and Struts

- Stop and think
  - Don’t try to merely replicate what the existing code is doing
  - Think about the underlying layout

# Migrating from Springs and Struts

- Stop and think
  - Don’t try to merely replicate what the existing code is doing
  - Think about the underlying layout
- Try replacing it with nothing!
  - Are you working around a limitation of springs and struts?
  - Does the code implement a relationship?

# Migrating from Springs and Struts

- Stop and think
  - Don’t try to merely replicate what the existing code is doing
  - Think about the underlying layout
- Try replacing it with nothing!
  - Are you working around a limitation of springs and struts?
  - Does the code implement a relationship?
- Otherwise, add some constraints

# Migrating from Springs and Struts

# Migrating from Springs and Struts

- Think about which component should own each constraint

# Migrating from Springs and Struts

- Think about which component should own each constraint
- Consider centralizing it in updateConstraints

# Migrating from Springs and Struts

5. Test it

# Migrating from Springs and Struts

## 5. Test it

- Verify the layout is correct
- Fix issues you may have

# Migrating from Springs and Struts

## 5. Test it

- Verify the layout is correct
- Fix issues you may have

- Wait, what?

# Auto Layout

- Thinking in constraints
  - Transitioning to constraints
- Debugging constraint-based layouts
  - Ambiguity
  - Unsatisfiability
  - Reading log messages
- Unleashing the power of constraints
  - Animation
  - Writing a custom control
  - Internationalization

# What Can Go Wrong?

- Constraints that provide insufficient information
  - Ambiguity
- Constraints that provide conflicting information
  - Unsatisfiability
- Constraints that are satisfied in unexpected ways

# What Can Go Wrong?

- Interface Builder prevents unsatisfiable or ambiguous constraints
- Rely on Interface Builder as much as possible
- You can reference constraints with outlets

# Auto Layout

- Thinking in constraints
  - Transitioning to constraints
- Debugging constraint-based layouts
  - Ambiguity
  - Unsatisfiability
  - Reading log messages
- Unleashing the power of constraints
  - Animation
  - Writing a custom control
  - Internationalization

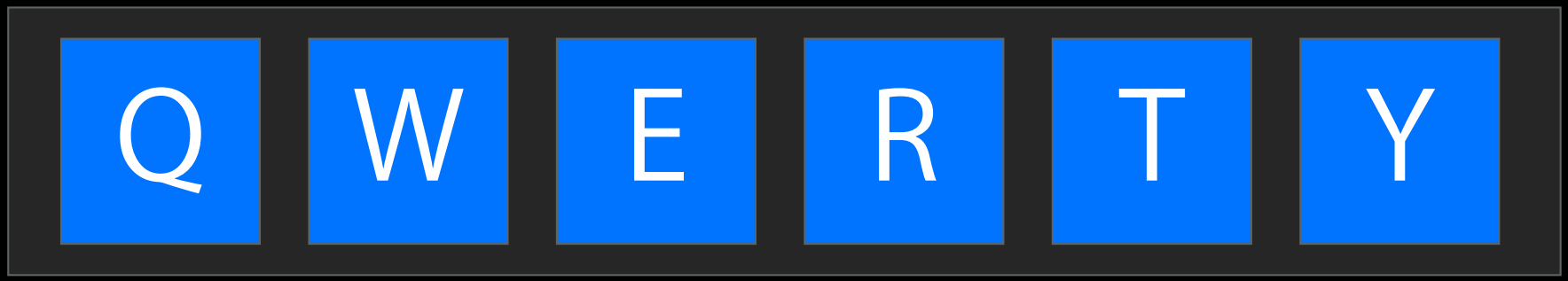
# Ambiguity

# Ambiguity

- Ambiguity means multiple layouts satisfy all constraints equally well

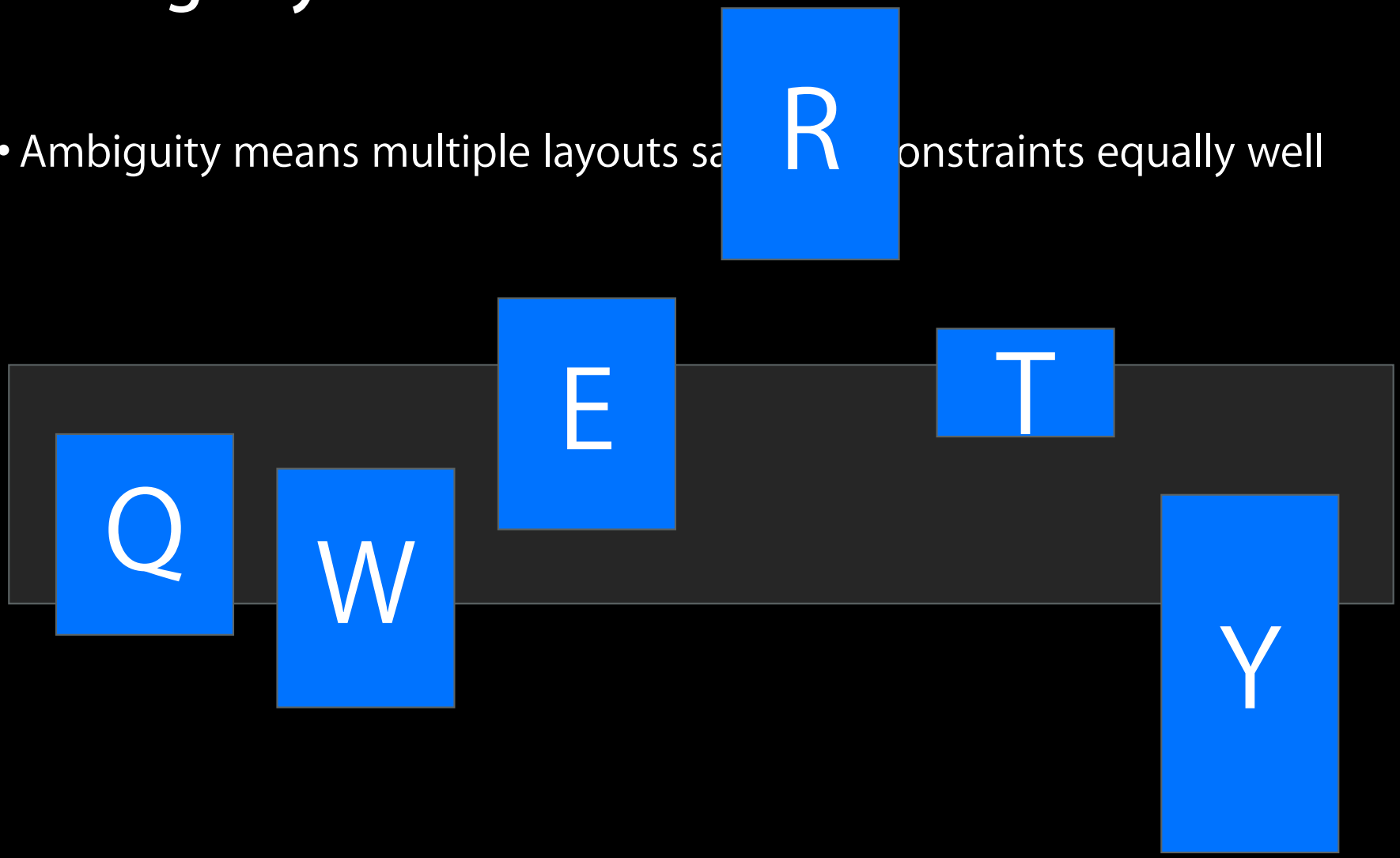
# Ambiguity

- Ambiguity means multiple layouts satisfy all constraints equally well



# Ambiguity

- Ambiguity means multiple layouts sa onstraints equally well



# Ambiguity

- Ambiguity means multiple layouts satisfy all constraints equally well

# Ambiguity

- Ambiguity means multiple layouts satisfy all constraints equally well
- A common symptom is that your views will cycle between those layouts

# Ambiguity

- Ambiguity means multiple layouts satisfy all constraints equally well
- A common symptom is that your views will cycle between those layouts
- Views "jump" or disappear entirely (jump to zero size)

# Ambiguity

# Ambiguity

- Usually it means you need more constraints

# Ambiguity

- Usually it means you need more constraints
- Each view needs four properties (two in each dimension)
  - MinX, Width, MinY, Height
  - MinX, MaxX, CenterY, MaxY
  - CenterX, Width, Baseline, Height
  - etc.

# Ambiguity

- Usually it means you need more constraints
- Each view needs four properties (two in each dimension)
  - MinX, Width, MinY, Height
  - MinX, MaxX, CenterY, MaxY
  - CenterX, Width, Baseline, Height
  - etc.
- Inequalities by themselves are usually not enough
  - view.width ≥ 20 – is it 20? 200? 2 billion?
  - Inequalities don't care how much larger or smaller you are
  - But equalities care

# Ambiguity

# Ambiguity

- Rarely, ambiguity means you need to adjust priorities

# Ambiguity

- Rarely, ambiguity means you need to adjust priorities

```
[view(24@500)
```

```
[view(>=30@500)]
```

# Ambiguity

- Rarely, ambiguity means you need to adjust priorities

```
[view(24@500)
```

```
[view(>=30@500)]
```

- It can't satisfy both
- They have equal priorities
- Ambiguity!

# Ambiguity

- Rarely, ambiguity means you need to adjust priorities

```
[view(24@500)
```

```
[view(>=30@525)]
```

# Ambiguity

- Rarely, ambiguity means you need to adjust priorities

```
[view(24@500)
```

```
[view(>=30@525)]
```

- It still can't satisfy both

# Ambiguity

- Rarely, ambiguity means you need to adjust priorities

```
[view(24@500)
```

```
[view(>=30@525)]
```

- It still can't satisfy both
- The inequality has a higher priority, so it will be satisfied first

# Ambiguity

- Rarely, ambiguity means you need to adjust priorities

```
[view(24@500)
```

```
[view(>=30@525)]
```

- It still can't satisfy both
- The inequality has a higher priority, so it will be satisfied first
- The equality will be satisfied as closely as possible

# Ambiguity

- Rarely, ambiguity means you need to adjust priorities

```
[view(24@500)
```

```
[view(>=30@525)]
```

- It still can't satisfy both
- The inequality has a higher priority, so it will be satisfied first
- The equality will be satisfied as closely as possible
- No ambiguity!

# Ambiguity

- Rarely, ambiguity means you need to adjust priorities

```
[view(24@500)
```

```
[view(>=30@525)]
```

- It still can't satisfy both
- The inequality has a higher priority, so it will be satisfied first
- The equality will be satisfied as closely as possible
- No ambiguity!
- view.width = 30

# Ambiguity

- Is my layout ambiguous?

```
[view hasAmbiguousLayout]
```

- What is ambiguous about it?

```
[view exerciseAmbiguityInLayout]
```

```
[window visualizeConstraints: @[] ]
```

# Auto Layout

- Thinking in constraints
  - Transitioning to constraints
- Debugging constraint-based layouts
  - Ambiguity
  - Unsatisfiability
  - Reading log messages
- Unleashing the power of constraints
  - Animation
  - Writing a custom control
  - Internationalization

# Unsatisfiability

# Unsatisfiability

- Unsatisfiability means no layout can satisfy all required constraints

# Unsatisfiability

- Unsatisfiability means no layout can satisfy all required constraints
- Only required constraints can contribute to unsatisfiability
  - Constraints are required by default!

# Unsatisfiability

- Unsatisfiability means no layout can satisfy all required constraints
- Only required constraints can contribute to unsatisfiability
  - Constraints are required by default!
- Sizes are implicitly required to be at least zero

# Unsatisfiability

- Unsatisfiability is immediately reported
- Ambiguity can be temporarily tolerated
- Remove constraints as soon as they might become invalid
- Create valid constraints again in updateConstraints

# Help, I Don’t See Anything!

# Help, I Don’t See Anything!

- Where are your views?
  - Check their -frame

# Help, I Don't See Anything!

- Where are your views?
  - Check their -frame
- What constraints are making them that size?
  - Output [view constraintsAffectingLayoutForOrientation/Axis: NSLayoutConstraintOrientationHorizontal/Vertical]

# Help, I Don't See Anything!

- Where are your views?
  - Check their -frame
- What constraints are making them that size?
  - Output [view constraintsAffectingLayoutForOrientation/Axis: NSLayoutConstraintOrientationHorizontal/Vertical] ← 0 or 1

# Help, I Don't See Anything!

- Where are your views?
  - Check their -frame
- What constraints are making them that size?
  - Output [view constraintsAffectingLayoutForOrientation/Axis: NSLayoutConstraintOrientationHorizontal/Vertical] ← 0 or 1
- Is the layout ambiguous?
  - Call [view hasAmbiguousLayout]
  - Call [view exerciseAmbiguityInLayout]

# Help, I Don’t See Anything!

- Some layouts are only satisfiable at 0 size!

# Help, I Don’t See Anything!

- Some layouts are only satisfiable at 0 size!

```
foo.width = bar.width * 2
```

```
bar.width = foo.width * 3
```

# Help, I Don’t See Anything!

- Some layouts are only satisfiable at 0 size!

```
foo.width = bar.width * 2
```

```
bar.width = foo.width * 3
```

→

```
foo.width = bar.width = 0
```

*Demo*

# Auto Layout

- Thinking in constraints
  - Transitioning to constraints
- Debugging constraint-based layouts
  - Ambiguity
  - Unsatisfiability
  - Reading log messages
- Unleashing the power of constraints
  - Animation
  - Writing a custom control
  - Internationalization

# Anatomy of an Unsatisfiability Log

# Anatomy of an Unsatisfiability Log

```
Unable to simultaneously satisfy constraints:
(
    "<NSLayoutConstraint:0x10441ced0 LetterView-'H'.centerX == LetterPile:
0x102b25230.centerX - 11   (Names: LetterView-'H':0x10423c390 )>",
    "<NSLayoutConstraint:0x10441ce70 LetterView-'H'.centerX == LetterPile:
0x102b25230.centerX + 170   (Names: LetterView-'H':0x10423c390 )>"
)

Will attempt to recover by breaking constraint
<NSLayoutConstraint:0x10441ce70 LetterView-'H'.centerX == LetterPile:
0x102b25230.centerX + 170   (Names: LetterView-'H':0x10423c390 )>

Set the NSUserDefault
NSConstraintBasedLayoutVisualizeMutuallyExclusiveConstraints to YES to have
-[NSWindow visualizeConstraints:] automatically called when this happens.
And/or, break on objc_exception_throw to catch this in the debugger.
```

# Anatomy of an Unsatisfiability Log

```
Unable to simultaneously satisfy constraints:
(
    "<NSLayoutConstraint:0x10441ced0 LetterView-'H'.centerX == LetterPile:
0x102b25230.centerX - 11   (Names: LetterView-'H':0x10423c390 )>",
    "<NSLayoutConstraint:0x10441ce70 LetterView-'H'.centerX == LetterPile:
0x102b25230.centerX + 170   (Names: LetterView-'H':0x10423c390 )>"
)

Will attempt to recover by breaking constraint
<NSLayoutConstraint:0x10441ce70 LetterView-'H'.centerX == LetterPile:
0x102b25230.centerX + 170   (Names: LetterView-'H':0x10423c390 )>

Set the NSUserDefault
NSConstraintBasedLayoutVisualizeMutuallyExclusiveConstraints to YES to have
-[NSWindow visualizeConstraints:] automatically called when this happens.
And/or, break on objc_exception_throw to catch this in the debugger.
```

# Anatomy of an Unsatisfiability Log

```
Unable to simultaneously satisfy constraints:
(
    "<NSLayoutConstraint:0x10441ced0 LetterView-'H'.centerX == LetterPile:
0x102b25230.centerX - 11   (Names: LetterView-'H':0x10423c390 )>",
    "<NSLayoutConstraint:0x10441ce70 LetterView-'H'.centerX == LetterPile:
0x102b25230.centerX + 170   (Names: LetterView-'H':0x10423c390 )>"
)

Will attempt to recover by breaking constraint
<NSLayoutConstraint:0x10441ce70 LetterView-'H'.centerX == LetterPile:
0x102b25230.centerX + 170   (Names: LetterView-'H':0x10423c390 )>

Set the NSUserDefault
NSConstraintBasedLayoutVisualizeMutuallyExclusiveConstraints to YES to have
-[NSWindow visualizeConstraints:] automatically called when this happens.
And/or, break on objc_exception_throw to catch this in the debugger.
```

# Anatomy of an Unsatisfiability Log

```
Unable to simultaneously satisfy constraints:
(
    "<NSLayoutConstraint:0x10441ced0 LetterView-'H'.centerX == LetterPile:
0x102b25230.centerX - 11   (Names: LetterView-'H':0x10423c390 )>",
    "<NSLayoutConstraint:0x10441ce70 LetterView-'H'.centerX == LetterPile:
0x102b25230.centerX + 170   (Names: LetterView-'H':0x10423c390 )>"
)

Will attempt to recover by breaking constraint
<NSLayoutConstraint:0x10441ce70 LetterView-'H'.centerX == LetterPile:
0x102b25230.centerX + 170   (Names: LetterView-'H':0x10423c390 )>

Set the NSUserDefault
NSConstraintBasedLayoutVisualizeMutuallyExclusiveConstraints to YES to have
-[NSWindow visualizeConstraints:] automatically called when this happens.
And/or, break on objc_exception_throw to catch this in the debugger.
```

# Anatomy of an Unsatisfiability Log

```
Unable to simultaneously satisfy constraints:
(
    "<NSLayoutConstraint:0x10441ced0 LetterView-'H'.centerX == LetterPile:
0x102b25230.centerX - 11   (Names: LetterView-'H':0x10423c390 )>",
    "<NSLayoutConstraint:0x10441ce70 LetterView-'H'.centerX == LetterPile:
0x102b25230.centerX + 170   (Names: LetterView-'H':0x10423c390 )>"
)

Will attempt to recover by breaking constraint
<NSLayoutConstraint:0x10441ce70 LetterView-'H'.centerX == LetterPile:
0x102b25230.centerX + 170   (Names: LetterView-'H':0x10423c390 )>

Set the NSUserDefault
NSConstraintBasedLayoutVisualizeMutuallyExclusiveConstraints to YES to have
-[NSWindow visualizeConstraints:] automatically called when this happens.
And/or, break on objc_exception_throw to catch this in the debugger.
```

# Anatomy of an Unsatisfiability Log

```
Unable to simultaneously satisfy constraints:
(
    "<NSLayoutConstraint:0x10441ced0 LetterView-'H'.centerX == LetterPile:
0x102b25230.centerX - 11   (Names: LetterView-'H':0x10423c390 )>",
    "<NSLayoutConstraint:0x10441ce70 LetterView-'H'.centerX == LetterPile:
0x102b25230.centerX + 170   (Names: LetterView-'H':0x10423c390 )>"
)

Will attempt to recover by breaking constraint
<NSLayoutConstraint:0x10441ce70 LetterView-'H'.centerX == LetterPile:
0x102b25230.centerX + 170   (Names: LetterView-'H':0x10423c390 )>

Set the NSUserDefault
NSConstraintBasedLayoutVisualizeMutuallyExclusiveConstraints to YES to have
-[NSWindow visualizeConstraints:] automatically called when this happens.
And/or, break on objc_exception_throw to catch this in the debugger.
```

# Anatomy of an Unsatisfiability Log

```
Unable to simultaneously satisfy constraints:
(
    "<NSLayoutConstraint:0x10441ced0 LetterView-'H'.centerX == LetterPile:
0x102b25230.centerX - 11   (Names: LetterView-'H':0x10423c390 )>",
    "<NSLayoutConstraint:0x10441ce70 LetterView-'H'.centerX == LetterPile:
0x102b25230.centerX + 170   (Names: LetterView-'H':0x10423c390 )>"
)
```

```
Will attempt to recover by breaking constraint
<NSLayoutConstraint:0x10441ce70 LetterView-'H'.centerX == LetterPile:
0x102b25230.centerX + 170   (Names: LetterView-'H':0x10423c390 )>
```

Set the NSUserDefault
NSConstraintBasedLayoutVisualizeMutuallyExclusiveConstraints to YES to have
-[NSWindow visualizeConstraints:] automatically called when this happens.
And/or, break on objc_exception_throw to catch this in the debugger.

# Anatomy of an Unsatisfiability Log

```
Unable to simultaneously satisfy constraints:
(
    "<NSLayoutConstraint:0x10441ced0 LetterView-'H'.centerX == LetterPile:
0x102b25230.centerX - 11   (Names: LetterView-'H':0x10423c390 )>",
    "<NSLayoutConstraint:0x10441ce70 LetterView-'H'.centerX == LetterPile:
0x102b25230.centerX + 170   (Names: LetterView-'H':0x10423c390 )>"
)

Will attempt to recover by breaking constraint
<NSLayoutConstraint:0x10441ce70 LetterView-'H'.centerX == LetterPile:
0x102b25230.centerX + 170   (Names: LetterView-'H':0x10423c390 )>

Set the NSUserDefault
NSConstraintBasedLayoutVisualizeMutuallyExclusiveConstraints to YES to have
-[NSWindow visualizeConstraints:] automatically called when this happens.
And/or, break on objc_exception_throw to catch this in the debugger.
```

# Anatomy of an Unsatisfiability Log

```
Unable to simultaneously satisfy constraints:
(
    "<NSLayoutConstraint:0x10441ced0 LetterView-'H'.centerX == LetterPile:
0x102b25230.centerX - 11   (Names: LetterView-'H':0x10423c390 )>",
    "<NSLayoutConstraint:0x10441ce70 LetterView-'H'.centerX == LetterPile:
0x102b25230.centerX + 170   (Names: LetterView-'H':0x10423c390 )>"
)
```

```
Will attempt to recover by breaking constraint
<NSLayoutConstraint:0x10441ce70 LetterView-'H'.centerX == LetterPile:
0x102b25230.centerX + 170   (Names: LetterView-'H':0x10423c390 )>

Set the NSUserDefault
NSConstraintBasedLayoutVisualizeMutuallyExclusiveConstraints to YES to have
-[NSWindow visualizeConstraints:] automatically called when this happens.
And/or, break on objc_exception_throw to catch this in the debugger.
```

# Anatomy of an Unsatisfiability Log

```
    "<NSLayoutConstraint:0x10441ced0 LetterView-'H'.centerX == LetterPile:
0x102b25230.centerX - 11   (Names: LetterView-'H':0x10423c390 )>",
```

# Anatomy of an Unsatisfiability Log

```
NSLayoutConstraint:0x10441ced0

LetterView-'H'.centerX == LetterPile:0x102b25230.centerX - 11

(Names: LetterView-'H':0x10423c390 )
```

# Anatomy of an Unsatisfiability Log

```
NSLayoutConstraint:0x10441ced0

LetterView-'H'.centerX == LetterPile:0x102b25230.centerX - 11

(Names: LetterView-'H':0x10423c390 )
```

# Anatomy of an Unsatisfiability Log

```
NSLayoutConstraint:0x10441ced0
```

Constraint's address

```
LetterView-'H'.centerX == LetterPile:0x102b25230.centerX - 11

(Names: LetterView-'H':0x10423c390 )
```

# Anatomy of an Unsatisfiability Log

```
NSLayoutConstraint:0x10441ced0

LetterView-'H'.centerX == LetterPile:0x102b25230.centerX - 11

(Names: LetterView-'H':0x10423c390 )
```

# Anatomy of an Unsatisfiability Log

```
NSLayoutConstraint:0x10441ced0

LetterView-'H'.centerX == LetterPile:0x102b25230.centerX - 11

(Names: LetterView-'H':0x10423c390 )
```

Map from identifier to view

# Anatomy of an Unsatisfiability Log

```
NSLayoutConstraint:0x10441ced0

LetterView-'H'.centerX == LetterPile:0x102b25230.centerX - 11

(Names: LetterView-'H':0x10423c390 )
```

Map from identifier to view

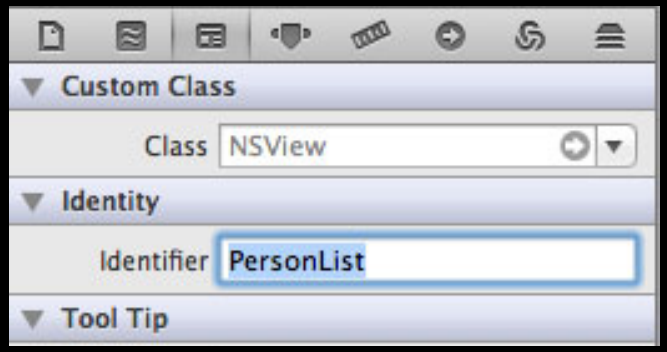
- View identifiers make logs easier to read

# Anatomy of an Unsatisfiability Log

```
NSLayoutConstraint:0x10441ced0

LetterView-'H'.centerX == LetterPile:0x102b25230.centerX - 11

(Names: LetterView-'H':0x10423c390 )
```

Map from identifier to view

- View identifiers make logs easier to read

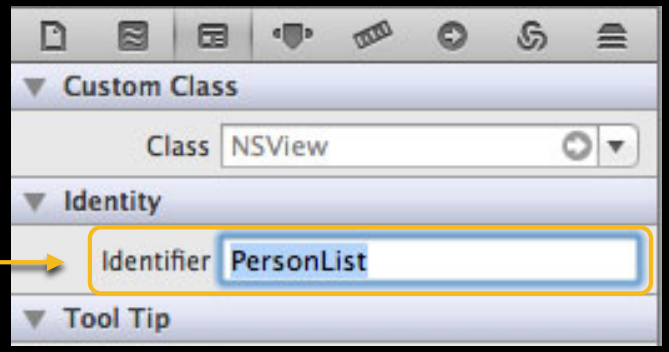


# Anatomy of an Unsatisfiability Log

```
NSLayoutConstraint:0x10441ced0

LetterView-'H'.centerX == LetterPile:0x102b25230.centerX - 11

(Names: LetterView-'H':0x10423c390 )
```

Map from identifier to view

- View identifiers make logs easier to read



# Anatomy of an Unsatisfiability Log

```
NSLayoutConstraint:0x10441ced0

LetterView-'H'.centerX == LetterPile:0x102b25230.centerX - 11

(Names: LetterView-'H':0x10423c390 )
```

# Anatomy of an Unsatisfiability Log

View's identifier

```
NSLayoutConstraint:0x10441ced0

LetterView-'H'.centerX == LetterPile:0x102b25230.centerX - 11

(Names: LetterView-'H':0x10423c390 )
```

# Anatomy of an Unsatisfiability Log

View's identifier

```
NSLayoutConstraint:0x10441ced0

LetterView-'H'.centerX == LetterPile:0x102b25230.centerX - 11

(Names: LetterView-'H':0x10423c390 )
```

Attribute

# Anatomy of an Unsatisfiability Log

View's identifier

Relation

```
NSLayoutConstraint:0x10441ced0

LetterView-'H'.centerX == LetterPile:0x102b25230.centerX - 11

(Names: LetterView-'H':0x10423c390 )
```

Attribute

# Anatomy of an Unsatisfiability Log

View's identifier

Relation

```
NSLayoutConstraint:0x10441ced0

LetterView-'H'.centerX == LetterPile:0x102b25230.centerX - 11

(Names: LetterView-'H':0x10423c390 )
```

Attribute

Other View

# Anatomy of an Unsatisfiability Log

View's identifier

Relation

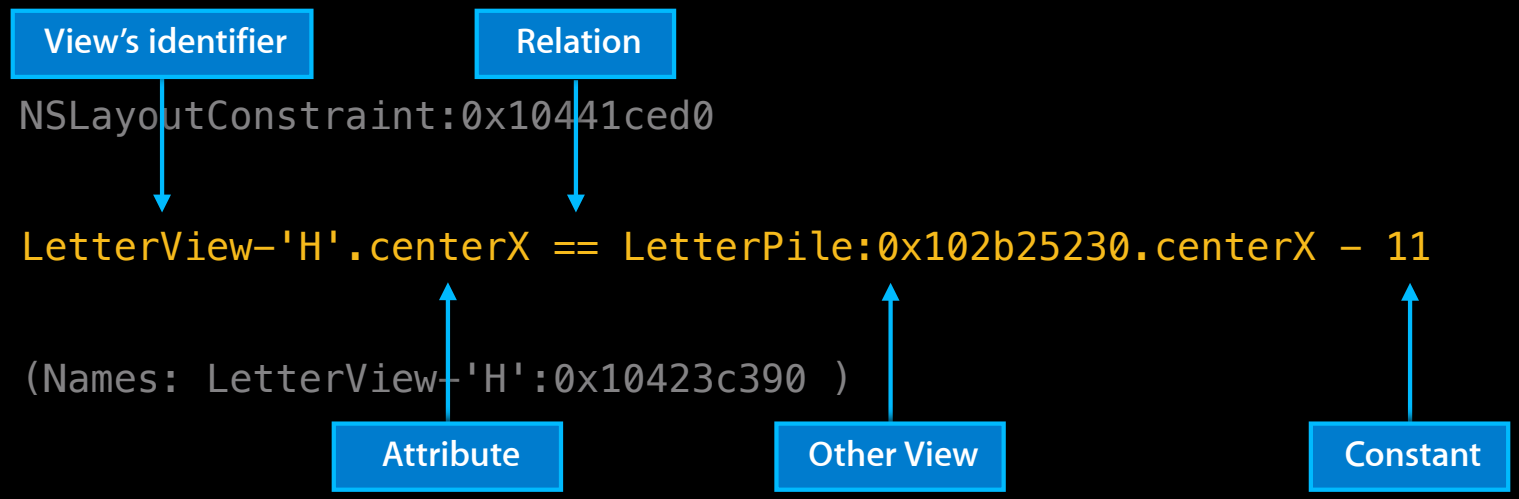
```
NSLayoutConstraint:0x10441ced0

LetterView-'H'.centerX == LetterPile:0x102b25230.centerX - 11

(Names: LetterView-'H':0x10423c390 )
```

Attribute

Other View

Constant

# Anatomy of an Unsatisfiability Log




"The letter view's center should be 11 points to the left of the pile's center"

# Anatomy of an Unsatisfiability Log

# Anatomy of an Unsatisfiability Log

- Logs use the visual format syntax when possible

# Anatomy of an Unsatisfiability Log

- Logs use the visual format syntax when possible

```
H:[NSView:0x102b5b3a0(250)]
```

# Anatomy of an Unsatisfiability Log

- Logs use the visual format syntax when possible

```
H:[NSView:0x102b5b3a0(250)]
```

"The view's width is 250"

# Anatomy of an Unsatisfiability Log

- Logs use the visual format syntax when possible

```
H:[NSView:0x102b5b3a0(250)]
```

"The view's width is 250"

```
H:[NSView:0x10480cd00]-(>=50)-[NSView:0x10481e9a0]>
```

# Anatomy of an Unsatisfiability Log

- Logs use the visual format syntax when possible

```
H:[NSView:0x102b5b3a0(250)]
```

"The view's width is 250"

```
H:[NSView:0x10480cd00]-(>=50)-[NSView:0x10481e9a0]>
```

"This view is at least 50 points to the right of that view"

# Anatomy of an Unsatisfiability Log

# Anatomy of an Unsatisfiability Log

```
<NSAutoresizingMaskLayoutConstraint:0x10590a360 h=-&- v=&--
V:[NSView:0x102e2af20(50)]>
```

# Anatomy of an Unsatisfiability Log

```
<NSAutoresizingMaskLayoutConstraint:0x10590a360 h=-&- v=&--
V:[NSView:0x102e2af20(50)]>
```

- `translatesAutoresizingMaskIntoConstraints` is on for this view

# Anatomy of an Unsatisfiability Log

```
<NSAutoresizingMaskLayoutConstraint:0x10590a360 h=-&- v=&--
V:[NSView:0x102e2af20(50)]>
```

Autoresizing Mask

- `translatesAutoresizingMaskIntoConstraints` is on for this view
- That produces more than one constraint

# Anatomy of an Unsatisfiability Log

```
<NSAutoresizingMaskLayoutConstraint:0x10590a360 h=-&- v=&--
V:[NSView:0x102e2af20(50)]>
```

Autoresizing Mask

- `translatesAutoresizingMaskIntoConstraints` is on for this view
- That produces more than one constraint

# Anatomy of an Unsatisfiability Log

```
<NSAutoresizingMaskLayoutConstraint:0x10590a360 h=-&- v=&--
V:[NSView:0x102e2af20(50)]>
```

"This view's height is 50."

Autoresizing Mask

- `translatesAutoresizingMaskIntoConstraints` is on for this view
- That produces more than one constraint

```
<NSAutoresizingMaskLayoutConstraint:0x103b25030 h=-&- v=&--
  H:|-(200)-[NSView:0x103b25090]
 (Names: '|':FlippedView:0x102e163f0 )>
```

Autoresizing constraint

```
<NSAutoresizingMaskLayoutConstraint:0x103b25030 h=-&- v=&--
  H:|-(200)-[NSView:0x103b25090]
 (Names: '|':FlippedView:0x102e163f0 )>
```

Autoresizing constraint

Autoresizing mask

```
<NSAutoresizingMaskLayoutConstraint:0x103b25030 h=-&- v=&--
  H:|-(200)-[NSView:0x103b25090]
 (Names: '|':FlippedView:0x102e163f0 )>
```

```
<NSAutoresizingMaskLayoutConstraint:0x103b25030 h=-&- v=&--
  H:|-(200)-[NSView:0x103b25090]
 (Names: '|':FlippedView:0x102e163f0 )>
```

Horizontal

```
<NSAutoresizingMaskLayoutConstraint:0x103b25030 h=-&- v=&--
  H:|-(200)-[NSView:0x103b25090]
 (Names: '|':FlippedView:0x102e163f0 )>
```

Horizontal

```
<NSAutoresizingMaskLayoutConstraint:0x103b25030 h=-&- v=&--
  H:|-(200)-[NSView:0x103b25090]
 (Names: '|':FlippedView:0x102e163f0 )>
```

200 points between

Horizontal

```
<NSAutoresizingMaskLayoutConstraint:0x103b25030 h=-&- v=&--
  H:|-(200)-[NSView:0x103b25090]
 (Names: '|':FlippedView:0x102e163f0 )>
```

200 points between

this view's left edge

Horizontal

```
<NSAutoresizingMaskLayoutConstraint:0x103b25030 h=-&- v=&--
  H:|-(200)-[NSView:0x103b25090]
 (Names: '|':FlippedView:0x102e163f0 )>
```

Superview

200 points between

this view's left edge

```
<NSAutoresizingMaskLayoutConstraint:0x103b25030 h=-&- v=&--
  H:|-(200)-[NSView:0x103b25090]
 (Names: '|':FlippedView:0x102e163f0 )>
```

Superview's description

```
<NSAutoresizingMaskLayoutConstraint:0x103b25030 h=-&- v=&--
  H:|-(200)-[NSView:0x103b25090]
 (Names: '|':FlippedView:0x102e163f0 )>
```

```
<NSAutoresizingMaskLayoutConstraint:0x103b25030 h=-&- v=&--
  H:|-(200)-[NSView:0x103b25090]
 (Names: '|':FlippedView:0x102e163f0 )>
```

```
<NSAutoresizingMaskLayoutConstraint:0x103b25030 h=-&- v=&--
  H:|-(200)-[NSView:0x103b25090]
 (Names: '|':FlippedView:0x102e163f0 )>
```

“This view’s left edge is 200 points from that of its superview, which is a FlippedView”

# Auto Layout

- Thinking in constraints
  - Transitioning to constraints
- Debugging constraint-based layouts
  - Ambiguity
  - Unsatisfiability
  - Reading log messages
- Unleashing the power of constraints
  - Animation
  - Writing a custom control
  - Internationalization

# Auto Layout

- Thinking in constraints
  - Transitioning to constraints
- Debugging constraint based-layouts
  - Ambiguity
  - Unsatisfiability
  - Reading log messages
- Unleashing the power of constraints
  - Animation
  - Writing a custom control
  - Internationalization

# Animation

# Animation

- How do you animate layout changes?

# Animation

- How do you animate layout changes?
- Apply the new layout and let CoreAnimation handle animation
  - Very fast
  - May transiently appear to violate constraints

# Animation

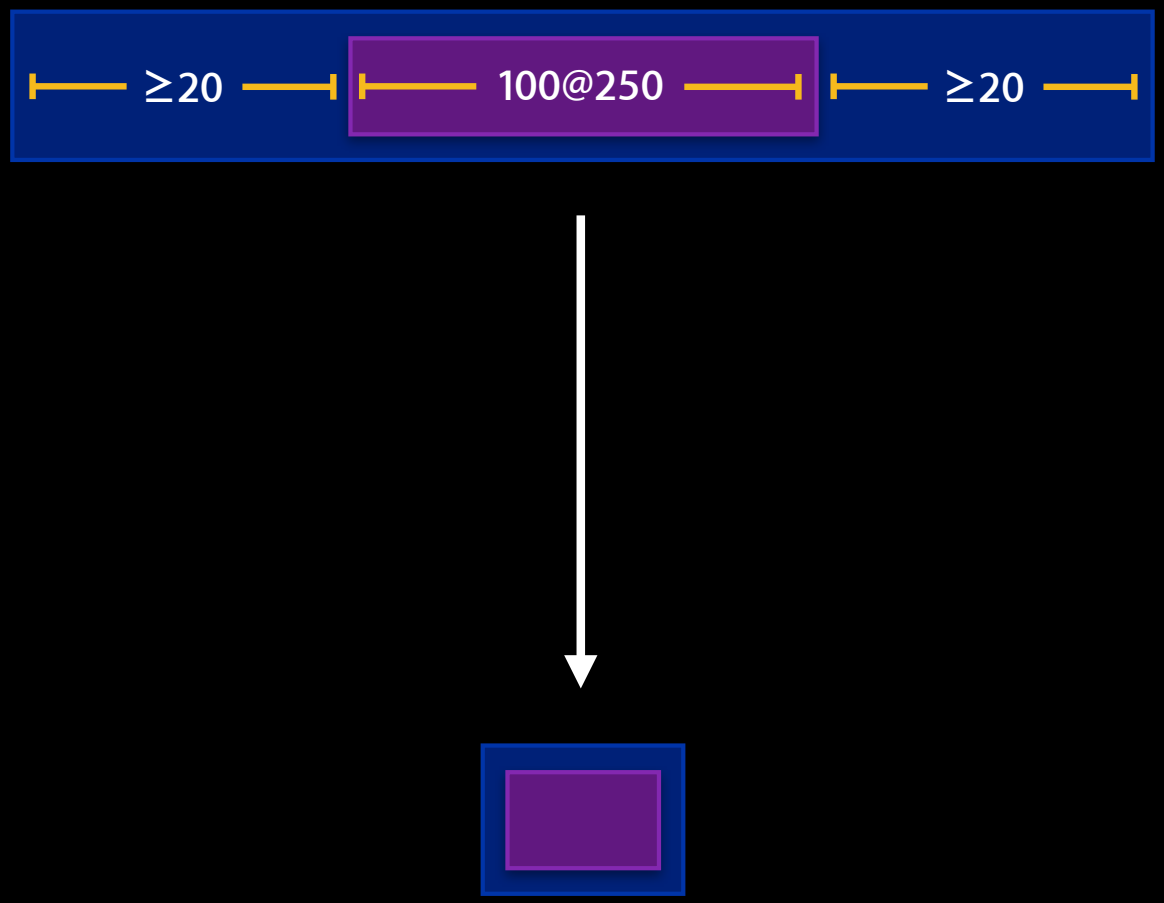
- How do you animate layout changes?
- Apply the new layout and let CoreAnimation handle animation
  - Very fast
  - May transiently appear to violate constraints
- Animate constraints directly
  - Pretty fast
  - Produces a correct layout at every frame

# Animation

# Animation

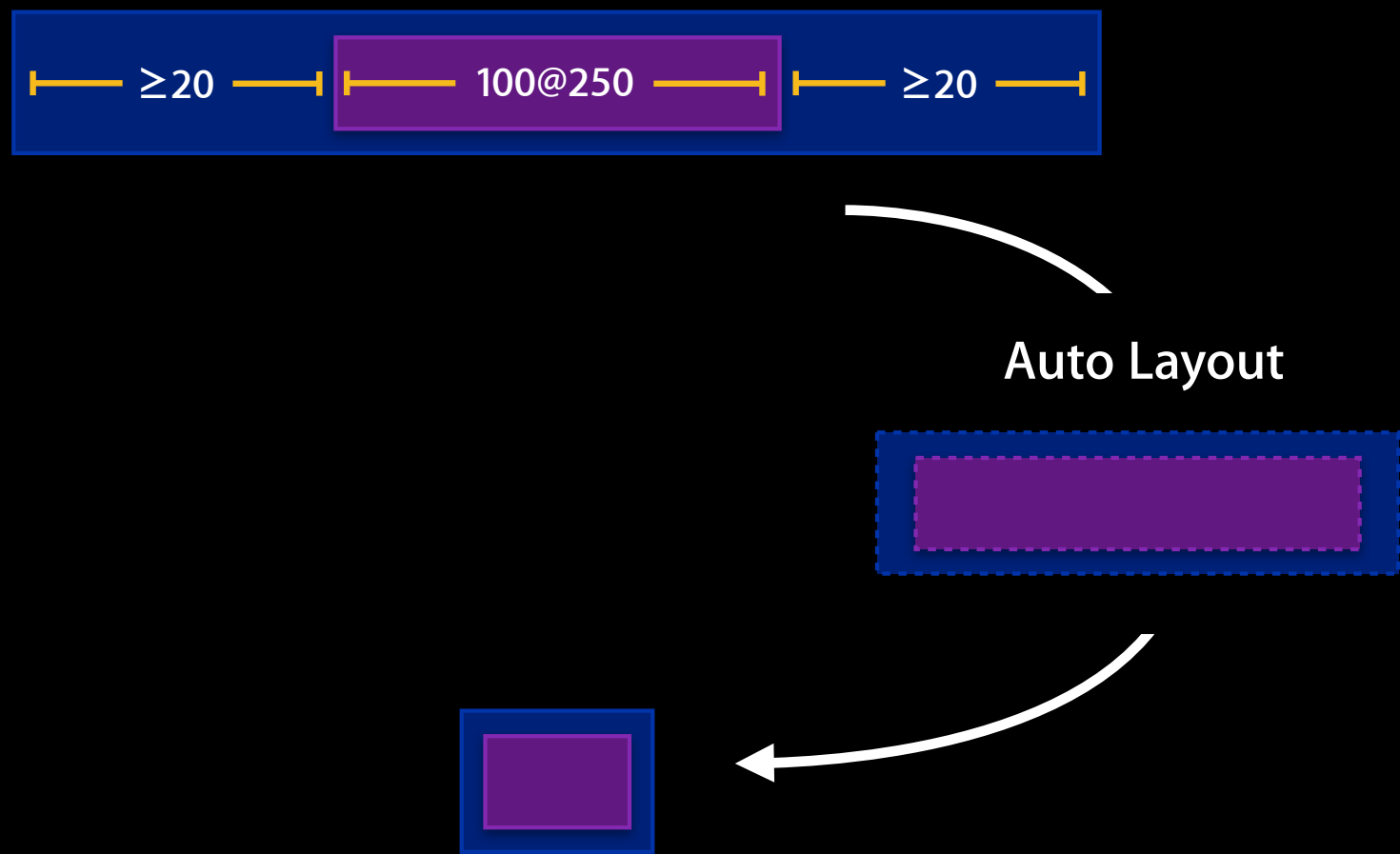
# Animation

# Animation

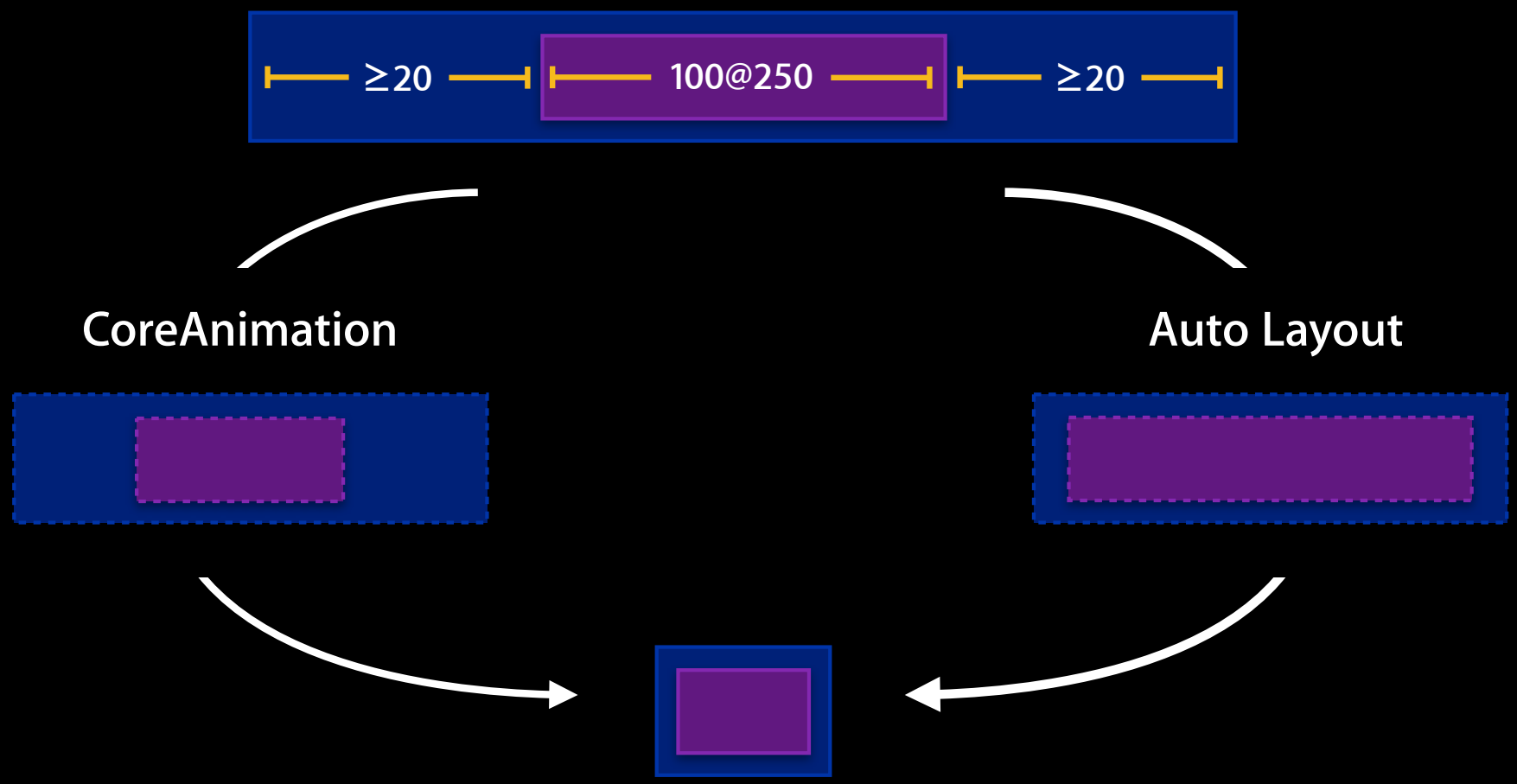


# Animation


# Animation



# Animation



# Animation with CoreAnimation

# Animation with CoreAnimation

- Adjust your constraints
- Within an animation block, call

  `[view layoutIfNeeded]` on iOS

  `[view layoutSubtreeIfNeeded]` on OS X

# Animation with CoreAnimation

- NSView

```
[NSAnimationContext runAnimationGroup:^(NSAnimationContext *context) {
    [context setDuration:0.5];
    [context setAllowsImplicitAnimation:YES];
    [view layoutSubtreeIfNeeded];
} completionHandler:nil]
```

- UIView

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

# Animation with NSLayoutConstraint

# Animation with NSLayoutConstraint

```
@interface NSLayoutConstraint
  @property (readonly) NSLayoutAttribute firstAttribute;
  @property (readonly) CGFloat multiplier;
  @property (readwrite) CGFloat constant;
@end
```

# Animation with NSLayoutConstraint

```
@interface NSLayoutConstraint
  @property (readonly) NSLayoutAttribute firstAttribute;
  @property (readonly) CGFloat multiplier;
  @property (readwrite) CGFloat constant;
@end
```

- The constant may be modified after creation

# Animation with NSLayoutConstraint

```
@interface NSLayoutConstraint
  @property (readonly) NSLayoutAttribute firstAttribute;
  @property (readonly) CGFloat multiplier;
  @property (readwrite) CGFloat constant;
@end
```

- The constant may be modified after creation
- Permits efficient relayouts

# Animation with NSLayoutConstraint

```
@interface NSLayoutConstraint
  @property (readonly) NSLayoutAttribute firstAttribute;
  @property (readonly) CGFloat multiplier;
  @property (readwrite) CGFloat constant;
@end
```

- The constant may be modified after creation
- Permits efficient relayouts
- Use an NSTimer

# Animation with NSLayoutConstraint

```
@interface NSLayoutConstraint
  @property (readonly) NSLayoutAttribute firstAttribute;
  @property (readonly) CGFloat multiplier;
  @property (readwrite) CGFloat constant;
@end
```

- The constant may be modified after creation
- Permits efficient relayouts
- Use an NSTimer

```
constraint.constant += 10
```

# Animation with NSLayoutConstraint

```
@interface NSLayoutConstraint
  @property (readonly) NSLayoutAttribute firstAttribute;
  @property (readonly) CGFloat multiplier;
  @property (readwrite) CGFloat constant;
@end
```

- The constant may be modified after creation
- Permits efficient relayouts
- Use an NSTimer
- Use the animator proxy

```
constraint.constant += 10
```

# Animation with NSLayoutConstraint

```
@interface NSLayoutConstraint
  @property (readonly) NSLayoutAttribute firstAttribute;
  @property (readonly) CGFloat multiplier;
  @property (readwrite) CGFloat constant;
@end
```

- The constant may be modified after creation
- Permits efficient relayouts
- Use an NSTimer `constraint.constant += 10`
- Use the animator proxy `constraint.animator.constant = 10`

# Animation with NSLayoutConstraint

```
@interface NSLayoutConstraint
  @property (readonly) NSLayoutAttribute firstAttribute;
  @property (readonly) CGFloat multiplier;
  @property (readwrite) CGFloat constant;
@end
```

- The constant may be modified after creation
- Permits efficient relayouts
- Use an NSTimer
- Use the animator proxy

```
constraint.constant += 10
```

```
constraint.animator.constant = 10
```

OS X only

*Demo*

# Auto Layout

- Thinking in constraints
  - Transitioning to constraints
- Debugging constraint-based layouts
  - Ambiguity
  - Unsatisfiability
  - Reading log messages
- Unleashing the power of constraints
  - Animation
  - Writing a custom control
  - Internationalization

# Alignment Rects

- Constraints operate on content, not frames
- The content area is called the alignment rect

# Alignment Rects

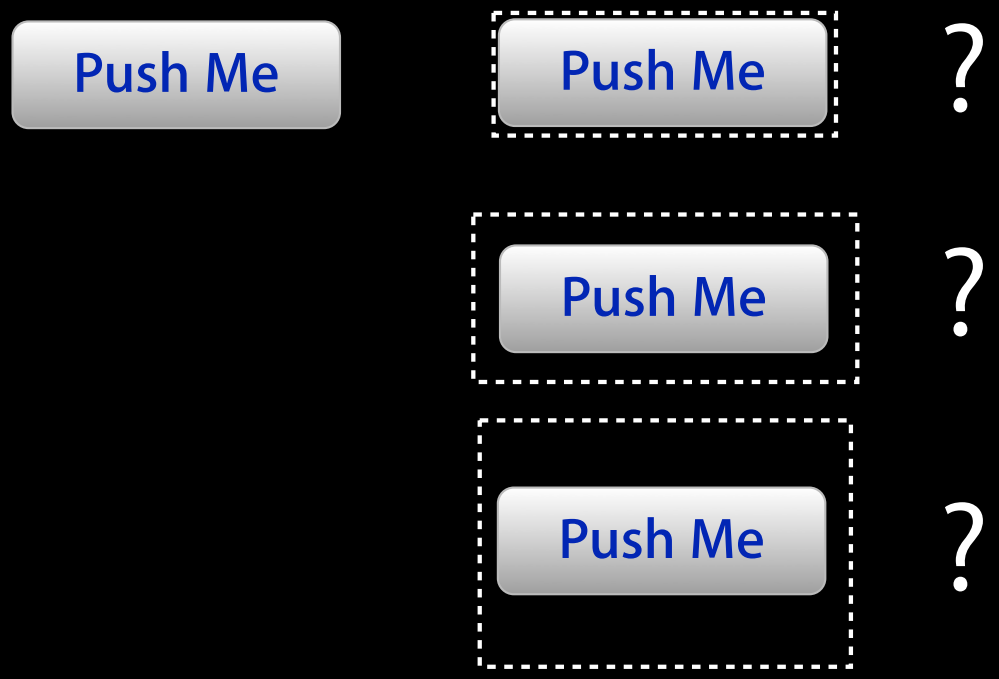
- Constraints operate on content, not frames
- The content area is called the alignment rect

# Alignment Rects

- Constraints operate on content, not frames
- The content area is called the alignment rect

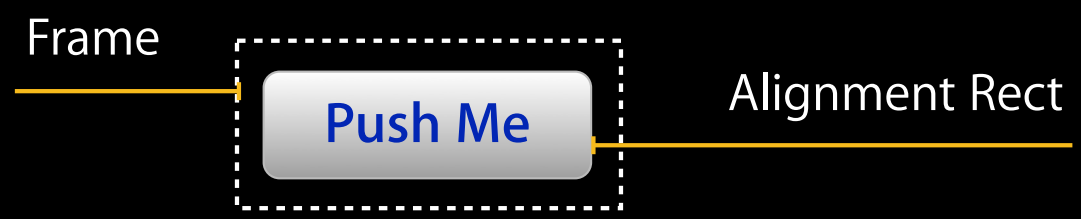
# Alignment Rects

- Constraints operate on content, not frames
- The content area is called the alignment rect



# Alignment Rects

- Constraints operate on content, not frames
- The content area is called the alignment rect



# Alignment Rects

- Constraints operate on content, not frames
- The content area is called the alignment rect

# Alignment Rects

- Constraints operate on content, not frames
- The content area is called the alignment rect

Edit 

# Alignment Rects

- Constraints operate on content, not frames
- The content area is called the alignment rect

# Alignment Rects

- Constraints operate on content, not frames
- The content area is called the alignment rect

# Alignment Rects

- Constraints operate on content, not frames
- The content area is called the alignment rect

# Alignment Rects

# Alignment Rects

- You can convert between alignment rects and frames

```
@implementation (NS,UI)View

- (CGRect)alignmentRectForFrame:(CGRect)frame

- (CGRect)frameForAlignmentRect:(CGRect)alignmentRect
```

# Intrinsic Content Size

# Intrinsic Content Size

- Many views are equally happy at any size
- Some views have a preferred size

  `sizeToFit`
  `sizeThatFits:`

- In Auto Layout, this is the `intrinsicContentSize`

# Intrinsic Content Size

- Many views are equally happy at any size
- Some views have a preferred size

  `sizeToFit`
  `sizeThatFits:`

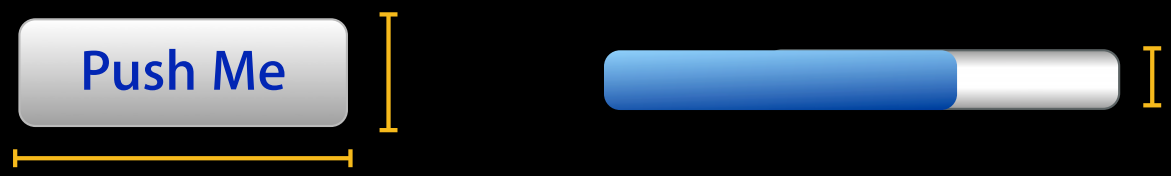
- In Auto Layout, this is the `intrinsicContentSize`

# Intrinsic Content Size

- Many views are equally happy at any size
- Some views have a preferred size

  `sizeToFit`
  `sizeThatFits:`

- In Auto Layout, this is the `intrinsicContentSize`



# Intrinsic Content Size

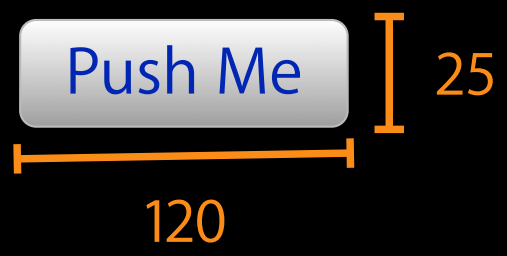
- Many views are equally happy at any size
- Some views have a preferred size

  `sizeToFit`
  `sizeThatFits:`

- In Auto Layout, this is the `intrinsicContentSize`

# Intrinsic Content Size

- An intrinsic content size generates two constraints per dimension

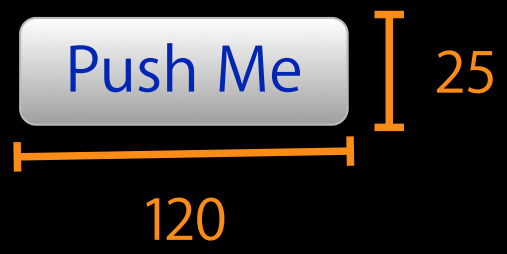



```
V:[button(>=25)]
V:[button(<=25)]
```

```
H:[button(>=120]
H:[button(<=120]
```

# Intrinsic Content Size

- An intrinsic content size generates two constraints per dimension



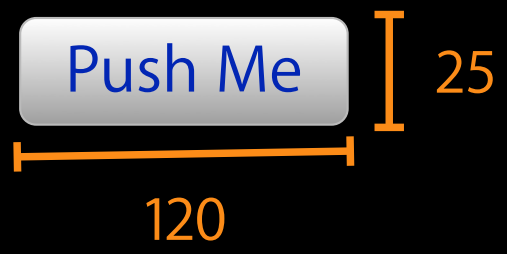

```
V:[button(>=25)]
V:[button(<=25)]
```

Compression Resistance

```
H:[button(>=120]
H:[button(<=120]
```

# Intrinsic Content Size

- An intrinsic content size generates two constraints per dimension



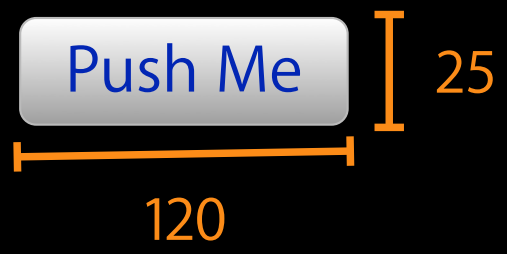

```
V:[button(>=25)]
V:[button(<=25)]
```

Compression Resistance → `H:[button(>=120]`

Content Hugging → `H:[button(<=120]`

# Intrinsic Content Size

- An intrinsic content size generates two constraints per dimension




```
V:[button(>=25)]
V:[button(<=25)]
```

Compression Resistance → `H:[button(>=120]`

Content Hugging → `H:[button(<=120]`

- This is sufficient to unambiguously size the view!

# Intrinsic Content Size

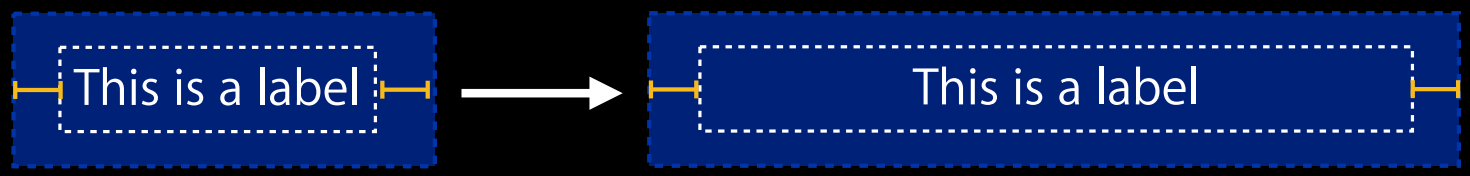
- Why two constraints?
- Because they can have different priorities!

# Intrinsic Content Size

- Why two constraints?
- Because they can have different priorities!

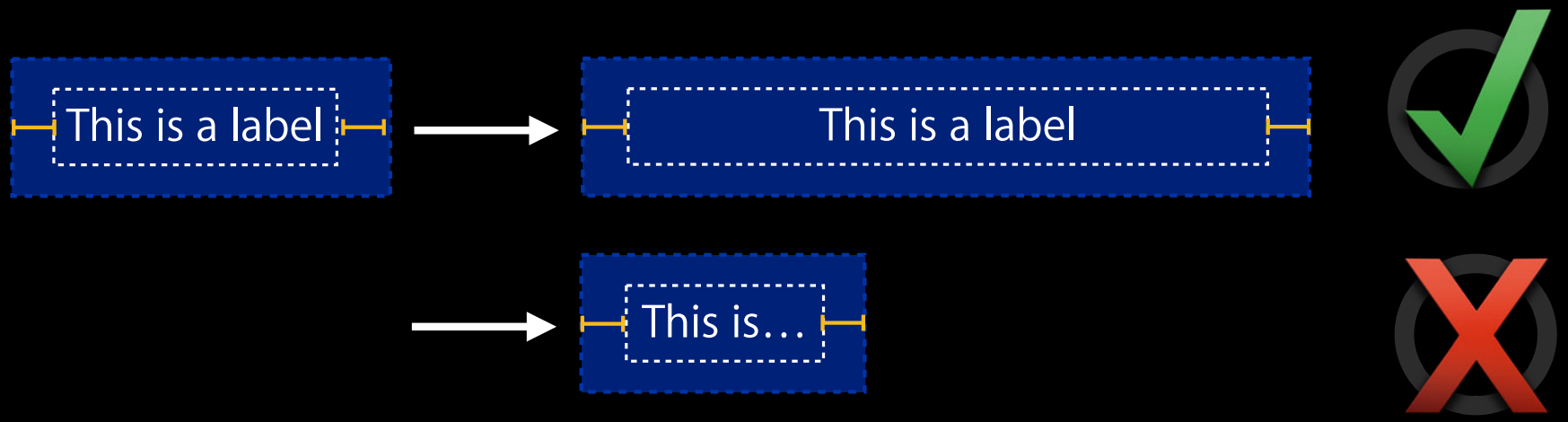
# Intrinsic Content Size

- Why two constraints?
- Because they can have different priorities!



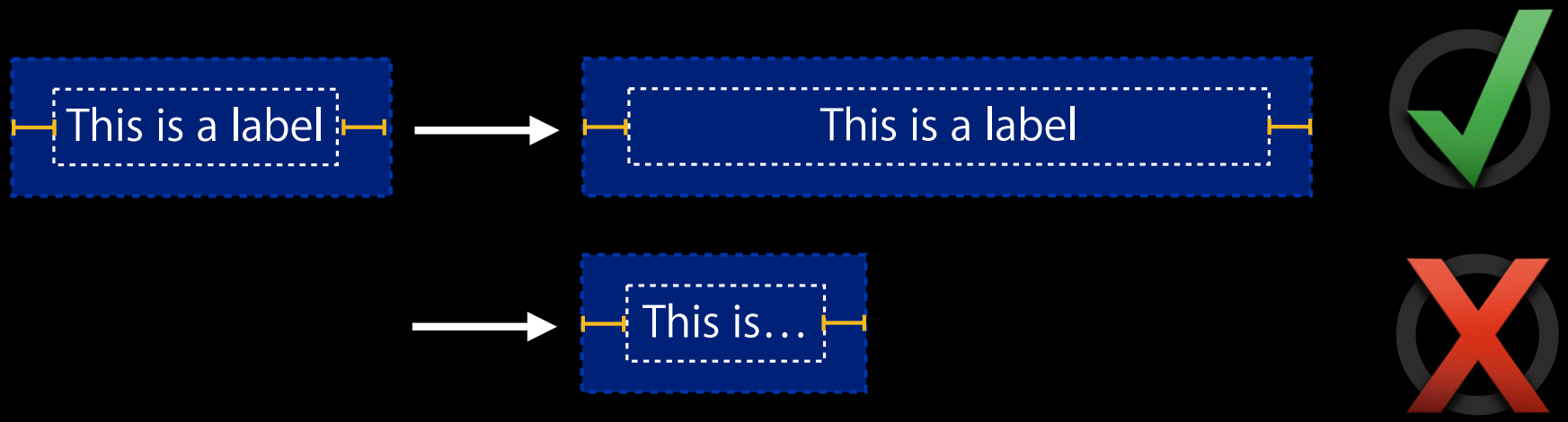

# Intrinsic Content Size

- Why two constraints?
- Because they can have different priorities!



# Intrinsic Content Size

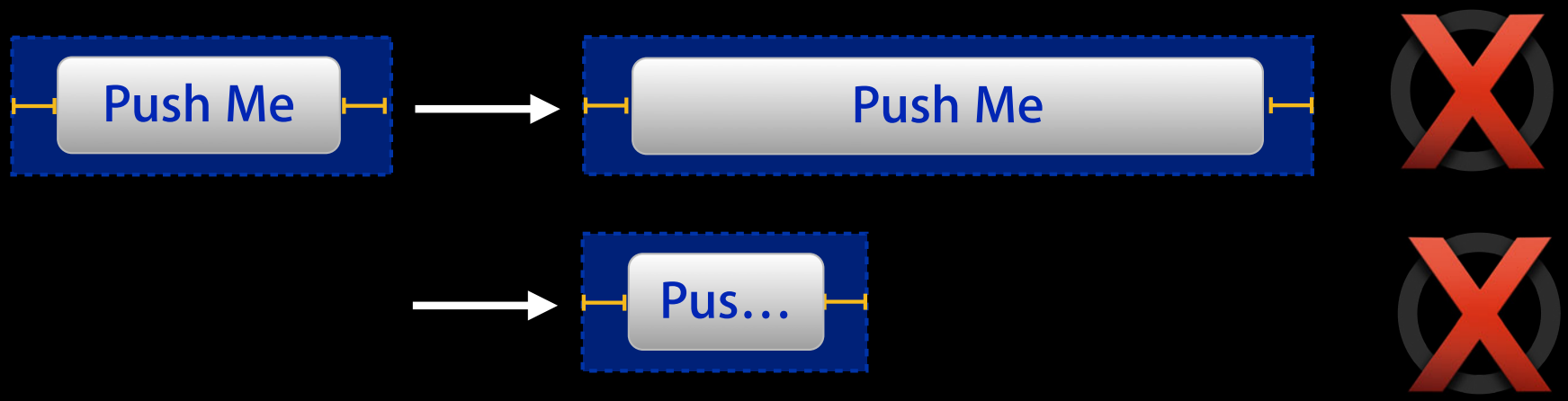
- Why two constraints?
- Because they can have different priorities!




- Low content hugging priority, high compression resistance priority

# Intrinsic Content Size

- Why two constraints?
- Because they can have different priorities!



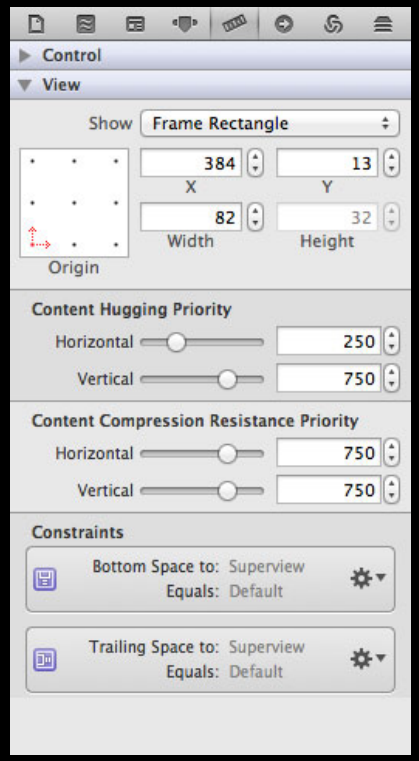

- High content hugging priority, high compression resistance priority

# Intrinsic Content Size

- Intrinsic content size is not settable
- The constraint priorities are settable

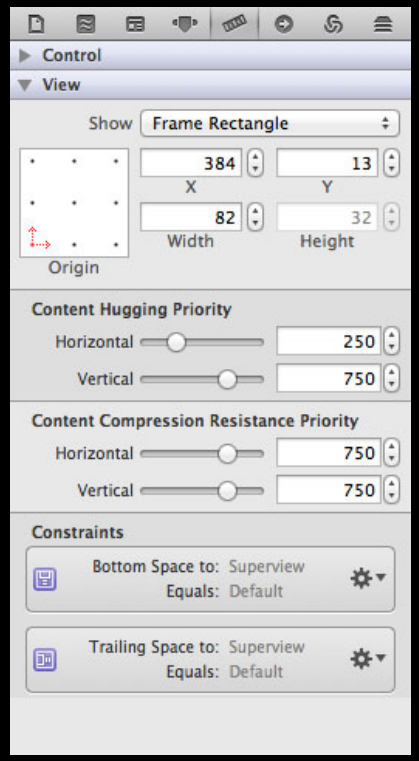
# Intrinsic Content Size

- Intrinsic content size is not settable
- The constraint priorities are settable



# Intrinsic Content Size

- Intrinsic content size is not settable
- The constraint priorities are settable



# Intrinsic Content Size

- Intrinsic content size is not settable
- The constraint priorities are settable

```
@implementation NSView
- (void)setContentHuggingPriority:(NSLayoutPriority)priority
        forOrientation:(NSLayoutConstraintOrientation)orientation;

- (void)setContentCompressionResistancePriority:(NSLayoutPriority)priority
        forOrientation:(NSLayoutConstraintOrientation)orientation;
@end
```

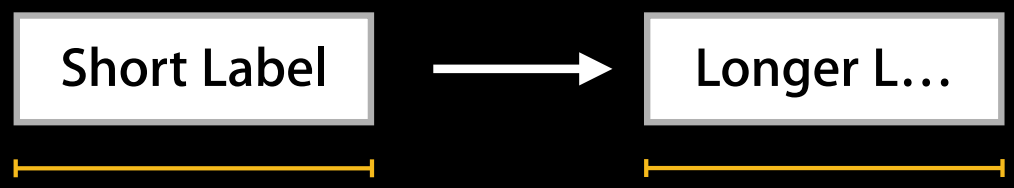
# Intrinsic Content Size

- Intrinsic content size is not settable
- The constraint priorities are settable

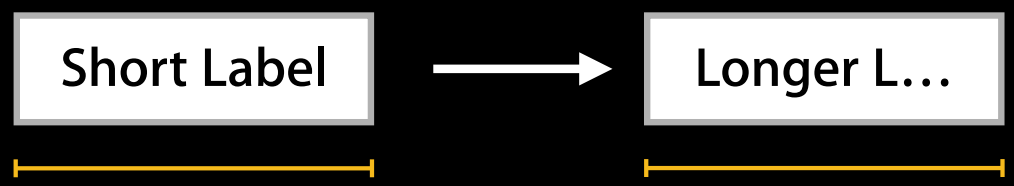
```
@implementation UIView
- (void)setContentHuggingPriority:(UILayoutPriority)priority
        forAxis:(UILayoutConstraintAxis)axis;

- (void)setContentCompressionResistancePriority:(UILayoutPriority)priority
        forAxis:(UILayoutConstraintAxis)axis;
@end
```

# Intrinsic Content Size

# Intrinsic Content Size



# Intrinsic Content Size




- The view calls `[self invalidateIntrinsicContentSize]` whenever its content changes
- Auto Layout reestablishes the sizing constraints
- If you implement a custom control, call this whenever your intrinsicContentSize might change

# Intrinsic Content Size


- The view calls `[self invalidateIntrinsicContentSize]` whenever its content changes
- Auto Layout reestablishes the sizing constraints
- If you implement a custom control, call this whenever your intrinsicContentSize might change

# Intrinsic Content Size

# Intrinsic Content Size

`intrinsicContentSize` as a better `sizeToFit`

# Intrinsic Content Size

`intrinsicContentSize` as a better `sizeToFit`

- sizeToFit must preserve binary compatibility
- It may be wrong for the current artwork
- intrinsicContentSize can change
- Use intrinsicContentSize as a better sizeToFit

# Intrinsic Content Size

`intrinsicContentSize` as a better `sizeToFit`

- sizeToFit must preserve binary compatibility
- It may be wrong for the current artwork
- intrinsicContentSize can change
- Use intrinsicContentSize as a better sizeToFit

```
NSRect alignmentRect = (NSRect){NSZeroPoint,
                                [control intrinsicContentSize]};
[control setFrameSize:
      [control frameForAlignmentRect:alignmentRect].size];
```

# Writing a Custom Control

## Override intrinsicContentSize

# Writing a Custom Control

## Override intrinsicContentSize

- Do
  - Measure text or images
  - Hard-code values

# Writing a Custom Control

## Override intrinsicContentSize

- Do
  - Measure text or images
  - Hard-code values
- Do not
  - Inspect your position, size, or constraints
  - Call super and “tweak” its value
  - Use it as a substitute for explicit constraints

# Writing a Custom Control

## Indicate your alignment rect

# Writing a Custom Control

## Indicate your alignment rect

- Do
  - Consider using the default implementation
  - Override - `(NS/UIEdgeInsets)alignmentRectInsets;`

# Writing a Custom Control

## Indicate your alignment rect

- Do
  - Consider using the default implementation
  - Override `- (NS/UIEdgeInsets)alignmentRectInsets;`
- Do not
  - Inspect your position, size, or constraints
  - Call super and “tweak” its value
  - Use it as a substitute for explicit constraints

# Overriding layout / layoutSubviews

# Overriding layout / layoutSubviews

## NSView

`-layout` sets the receiver's frame to the values determined by the constraints

# Overriding layout / layoutSubviews

## NSView

`-layout` sets the receiver's frame to the values determined by the constraints

## UIView

`-layoutSubviews` sets the receiver's center and bounds to the values determined by the constraints

# Overriding layout / layoutSubviews

**NSView**

`-layout` sets the receiver's frame to the values determined by the constraints

**UIView**

`-layoutSubviews` sets the receiver's center and bounds to the values determined by the constraints

- Afterwards, the constraints and frames agree
- Override it to do custom layout as long as you maintain that invariant

# Overriding layout / layoutSubviews

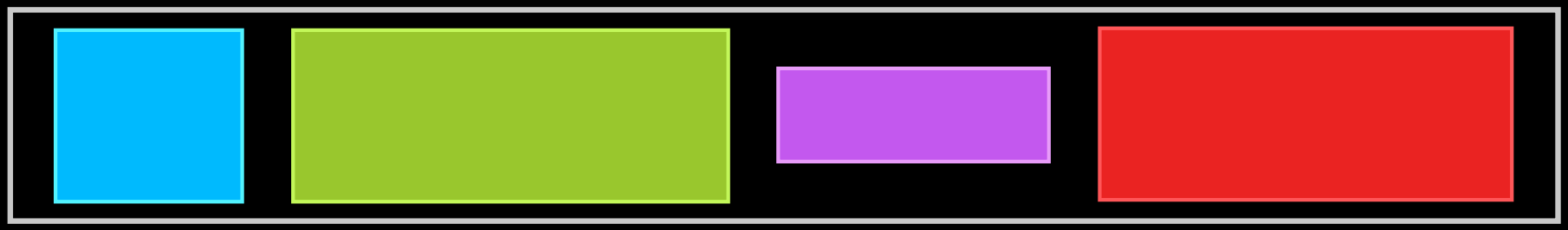
## Achieving a layout-dependent view hierarchy

# Overriding layout / layoutSubviews

## Achieving a layout-dependent view hierarchy

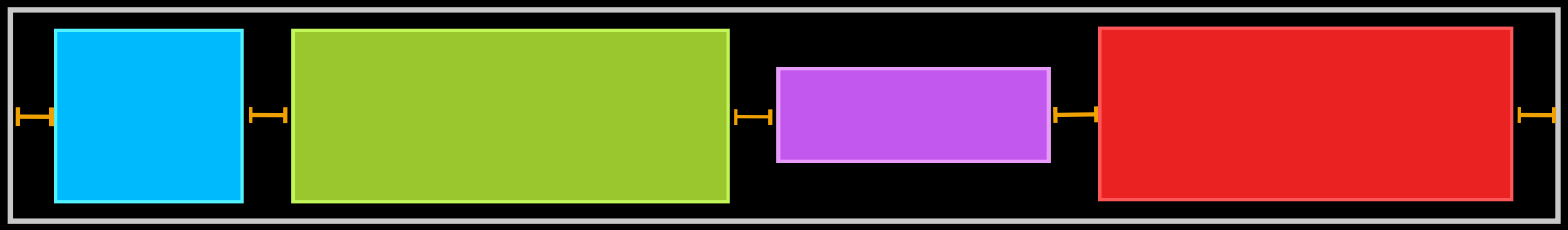
# Overriding layout / layoutSubviews

## Achieving a layout-dependent view hierarchy



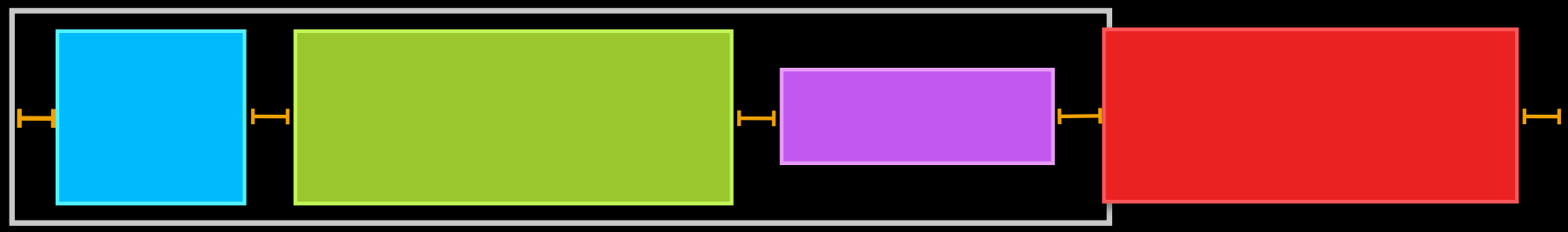
# Overriding layout / layoutSubviews

## Achieving a layout-dependent view hierarchy



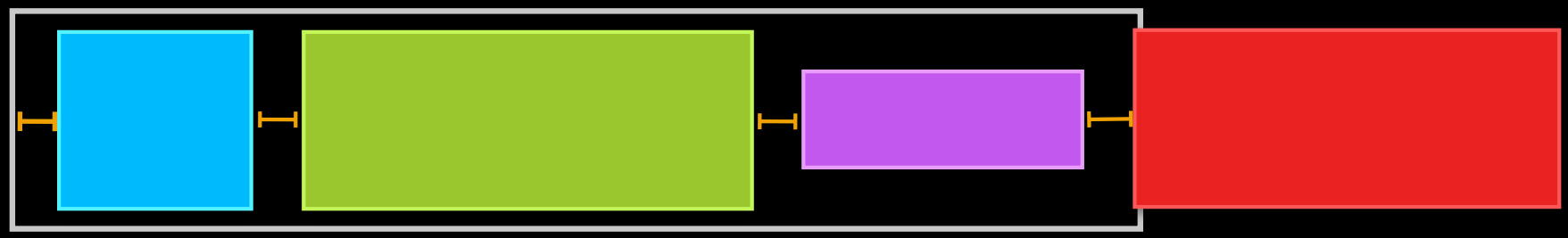
# Overriding layout / layoutSubviews

## Achieving a layout-dependent view hierarchy



# Overriding layout / layoutSubviews

## Achieving a layout-dependent view hierarchy



# Overriding layout / layoutSubviews

## Achieving a layout-dependent view hierarchy

# Overriding layout / layoutSubviews

## Achieving a layout-dependent view hierarchy

- Override `-layout` / `-layoutSubviews`
- Call super
- Inspect the resulting view positions and sizes
- Adjust subviews and constraints
- Call super again
- Repeat!

*Demo*

# Auto Layout

- Thinking in constraints
  - Transitioning to constraints
- Debugging constraint-based layouts
  - Ambiguity
  - Unsatisfiability
  - Reading log messages
- Unleashing the power of constraints
  - Animation
  - Writing a custom control
  - Internationalization

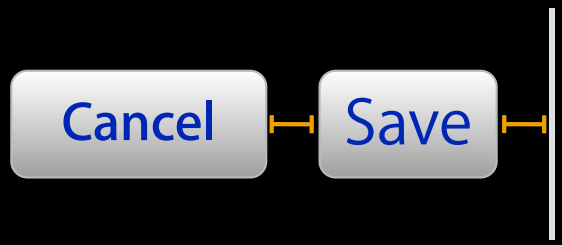
# Internationalization

# Internationalization

- Auto Layout makes internationalization easier
  - Controls size according to their content
  - The same constraints still work across different localizations

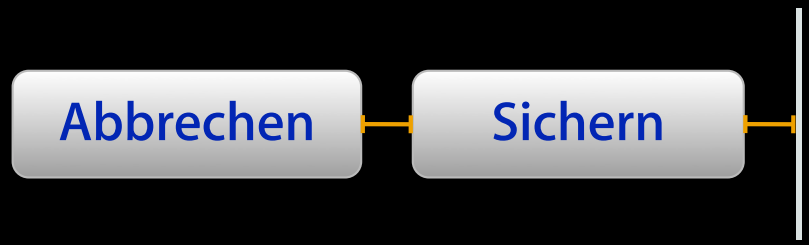
# Internationalization

- Auto Layout makes internationalization easier
  - Controls size according to their content
  - The same constraints still work across different localizations



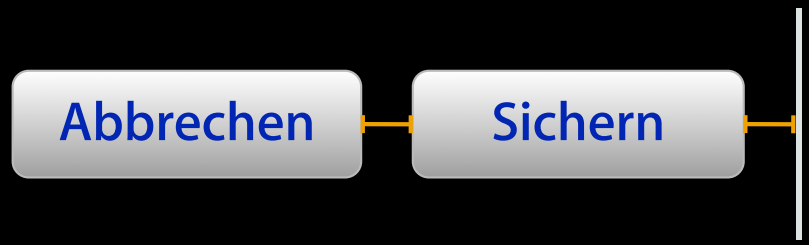
# Internationalization

- Auto Layout makes internationalization easier
  - Controls size according to their content
  - The same constraints still work across different localizations



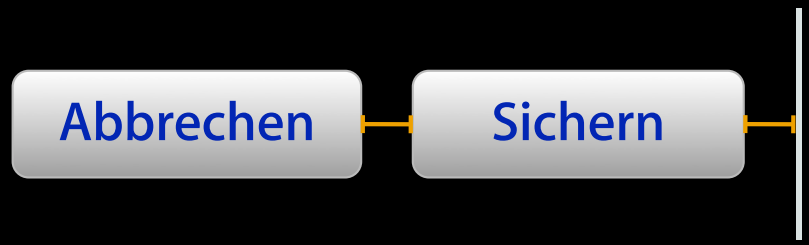
# Internationalization

- One nib can now service multiple localizations
- Control content is translated with a strings file at runtime
- You can fall back to separate nibs when necessary



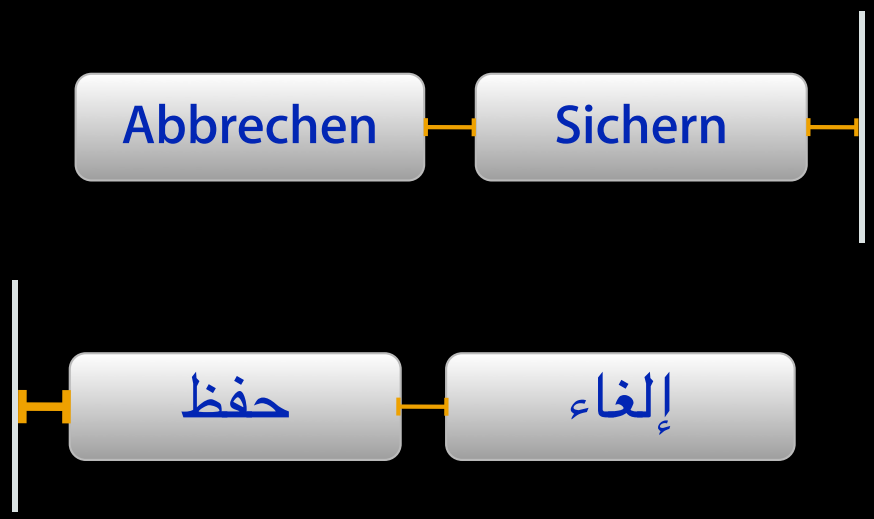
# Internationalization

- Right-to-left support is built in
- The leading and trailing edges flip under right-to-left localizations



# Internationalization

- Right-to-left support is built in
- The leading and trailing edges flip under right-to-left localizations



*Demo*

# Summary

- Auto Layout allows for powerful layout with less (or no) code
- Think declaratively
- Be wary of ambiguity and unsatisfiability
- The log messages are there to help
- Judicious overriding lets your custom views integrate with Auto Layout
- Localize with a single nib and multiple strings files

# More Information

**Paul Marcos**
Frameworks Evangelist
pmarcos@apple.com

**Documentation**
Cocoa Auto Layout Guide
http://developer.apple.com/library/mac/#documentation/UserExperience/Conceptual/AutolayoutPG/

**Apple Developer Forums**
http://devforums.apple.com

# Related Sessions

| | |
|---|---|
| Auto Layout by Example | Mission<br>Thursday 11:30AM |

# Labs

| | |
|---|---|
| Auto Layout Lab | App Services Lab B<br>Thursday 2:00PM |

WWDC2012