High performance AutoLayout

데릭

보리입니다 데스

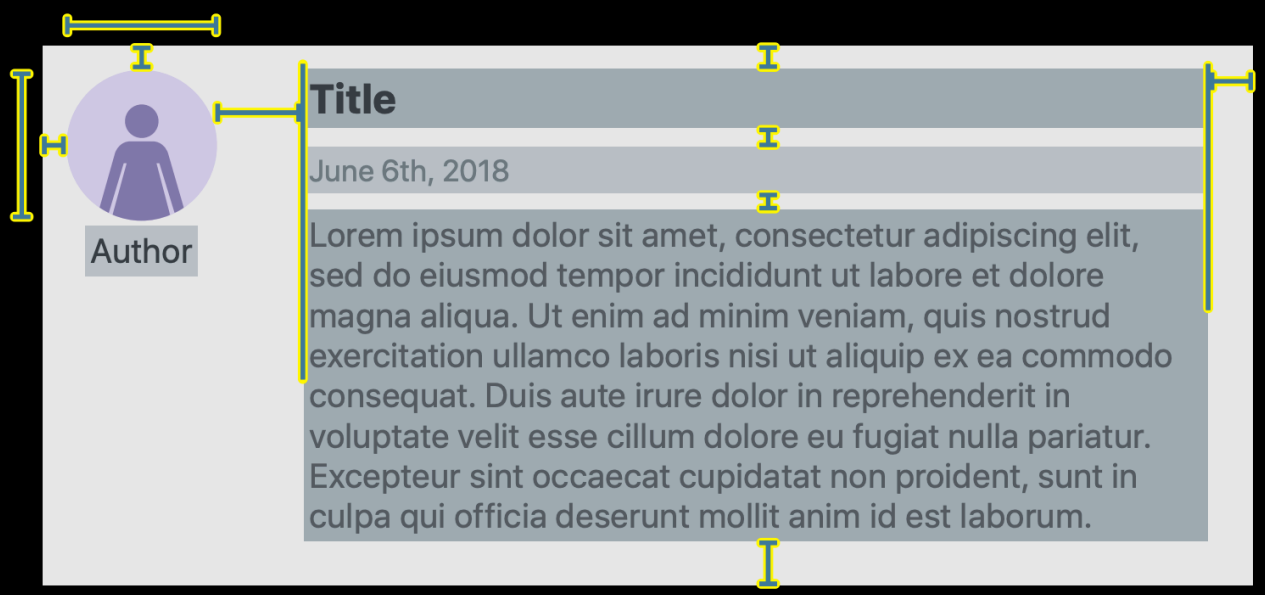
바드

1~ 64 페이지 보리
65~105 페이지 바드
106~150 페이지 데릭

D2Coding으로 할 것

보리

Auto Layout



오토레이아웃은 우리가 Mac의 iOS에 콘텐츠를 배치하는 방법
뷰, 제약조건으로 수행.

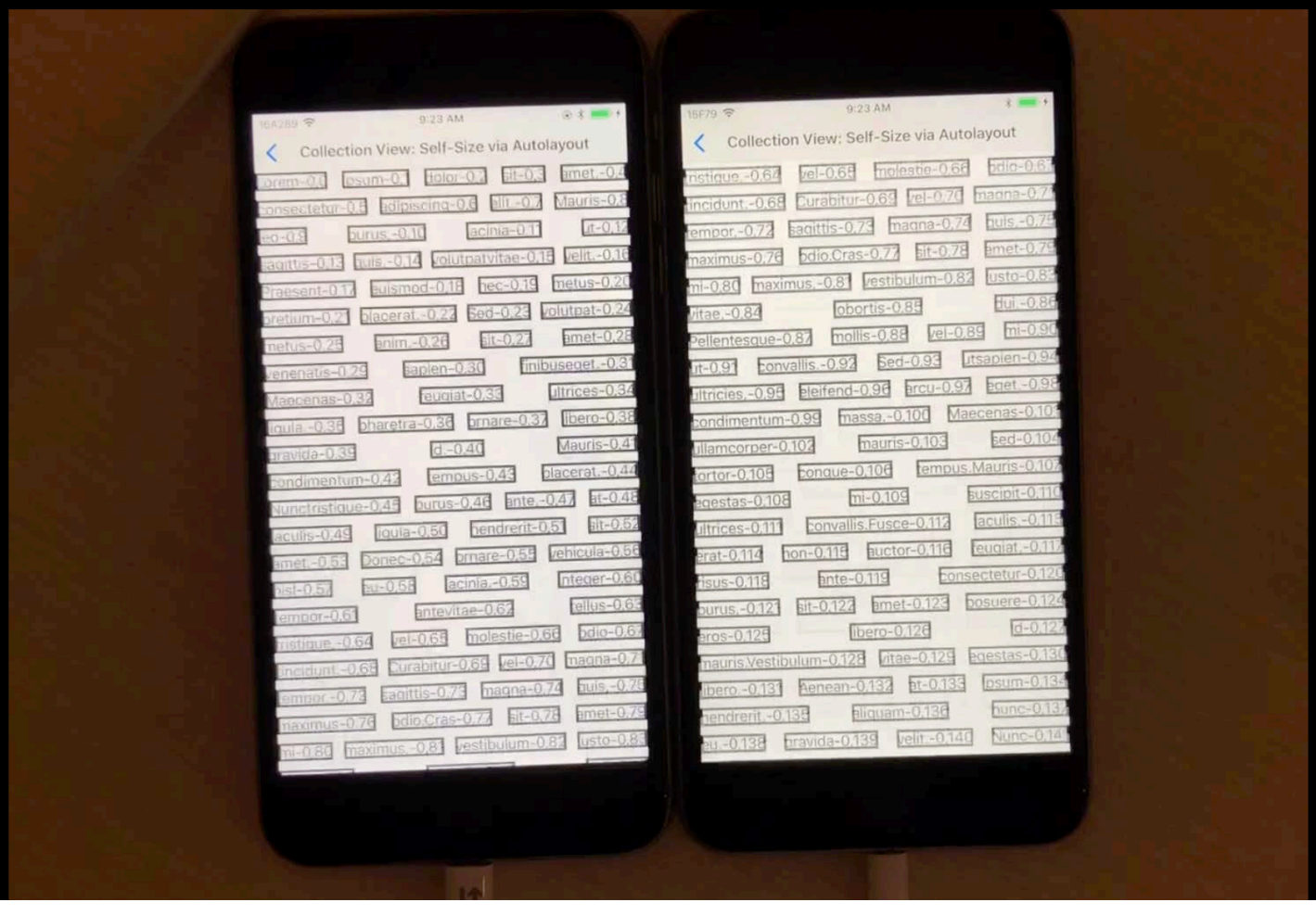
다만, 두 버튼간의 거리가 20이어야 한다고 말한경우 기계가 수행하는 단계별 프로세스를 이해하기 어려울 수 있따는 점이다.
즉 성능에 대한 기대치를 이해하고, 빠른것과 그렇지 않은 것 등 일반적으로 작동 방식을 이해하기 어렵다는 것임

iOS12 릴리스에서 성능에 관련된 수행의 작업중 일부를 보여줌으로써 시작하고 수행의 단계를 보여줄 것

>> 그냥 오토레이아웃관련 제약조건 등등에 대한 성능을 측정하기 어려울테니 이번 강의에서 그 수행 스텝에 대해 설명하고 어떻게 작동하는지에 대해서 이야기할것이라고 함.

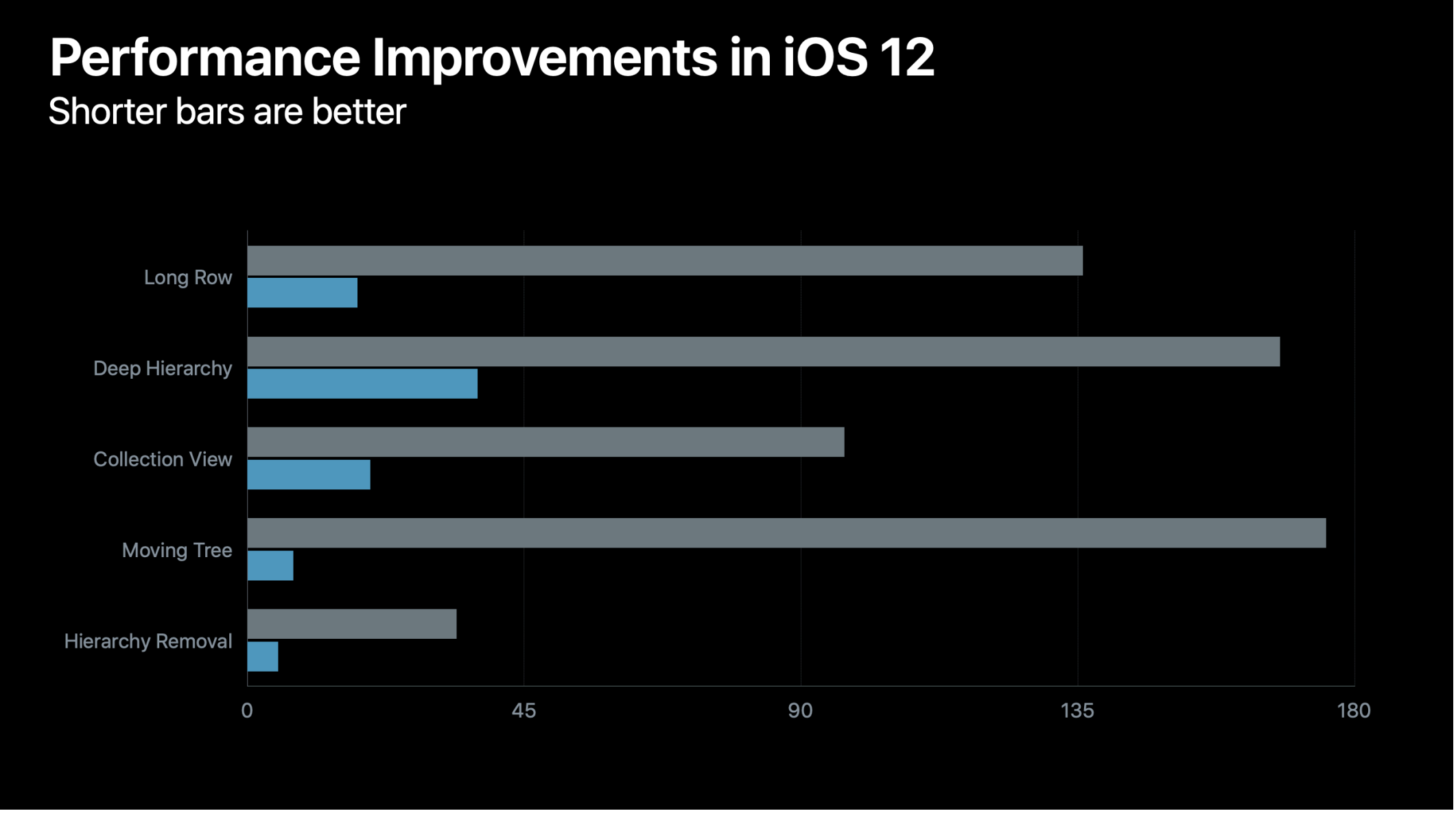
Performance Improvements in iOS 12

You get it for free



이건 자체 크기 조정 셀이 있는 UICollectionViewCell이고 이는 iOS11에서 이렇게 수행됨

솔직히 뭐가 다른지 모르겠는데 11이 더 구리다고함 ...(?)
전체 프레임 속도에 도달한다고 함.



암튼 회색이 11이고 파랑이 12임

(솔직히 무슨말인지 모르겠고, 이 부분은 검색해도 안나오지만 추측하자면...)

12로 넘어가면서 UICollectionView에 오토레이아웃 적용 가능하도록 변경이 일어났고 이로 인해 성능 향상이 이뤄진 부분이 많다는 얘기같음

iOS 12 improvements

Internals and intuition

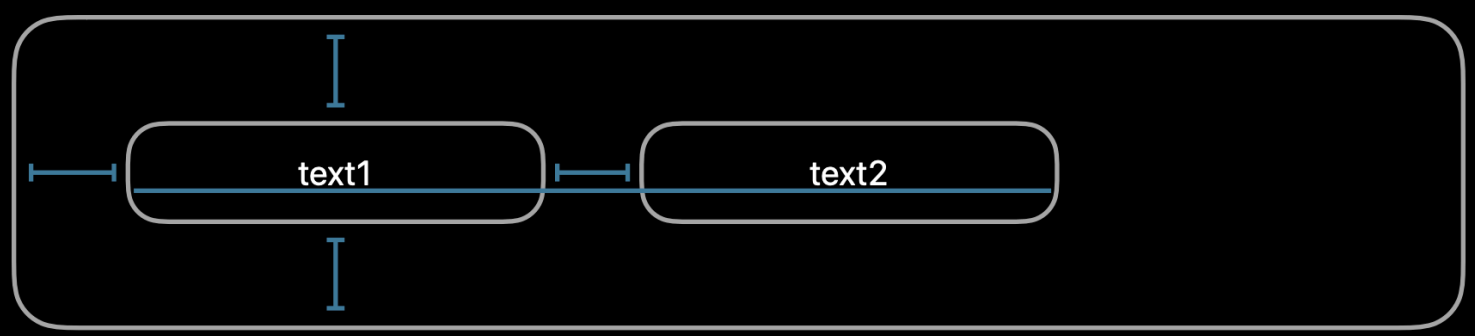
Building efficient layouts

암튼 이런 과정을 거치면서 모든 개선을 할 수 있었던 이유는 사물이 어떻게 구성되고 작동하는지, 어떻게 동작하는지에 대한 좋은 정신적 모델을 가지고 있었기 때문이라 생각

우리는이 모델을 구성하는거에 도움을 주고싶은데 여기엔 몇가지 문제가 있음

클라이언트 앱을 검토할 때 가장 일반적인 문제를 선택함

Simple Layout



매우 간단한 레이아웃을 생성해보자.

인터페이스 빌더에선 이를 만들어줄것이라고 생각함.

근데 코드로 짜면?

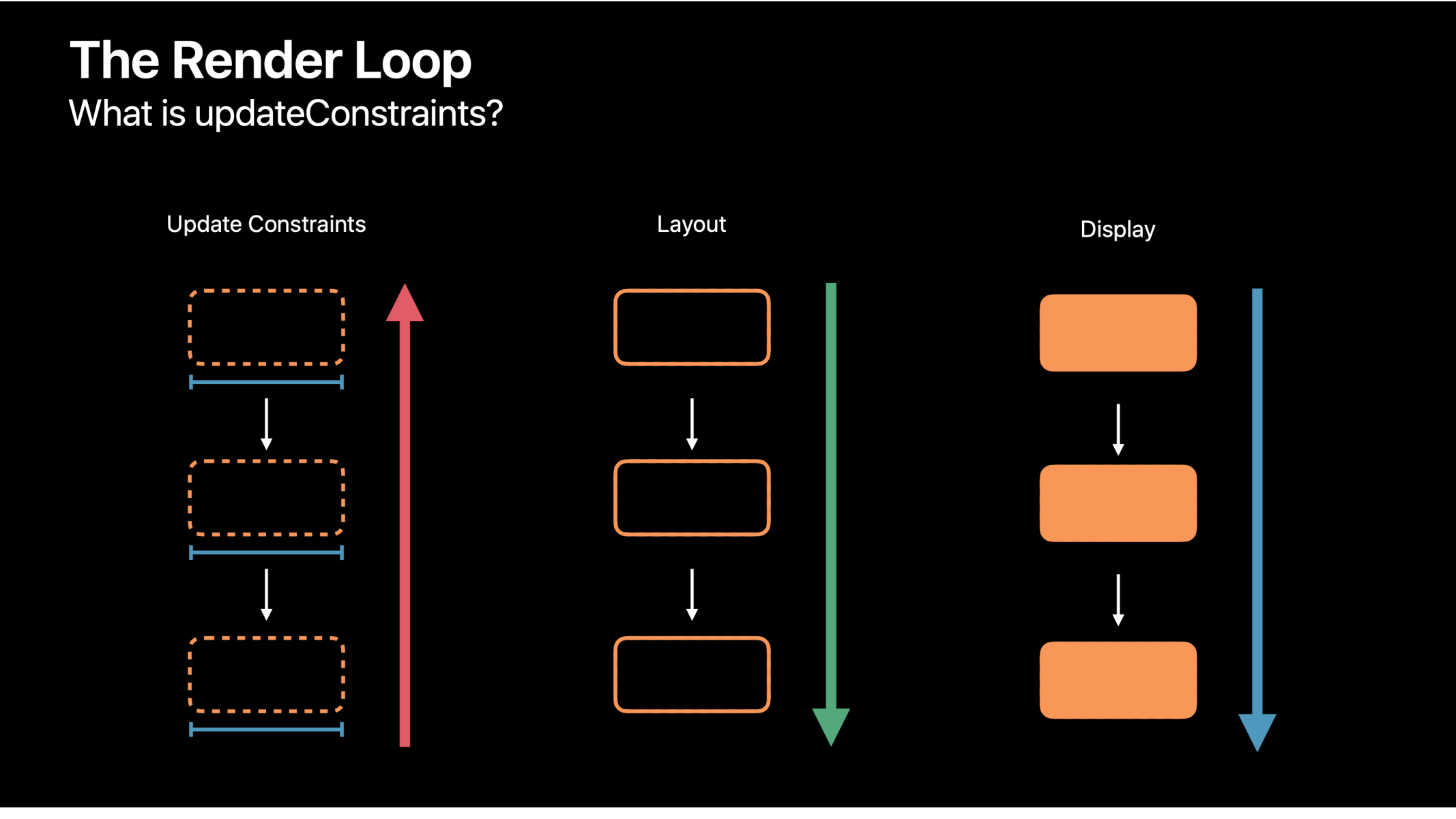
```
// Don't do this! Removes and re-adds constraints potentially at 120 frames per second
1 override func updateConstraints() {
    2 NSLayoutConstraint.deactivate(myConstraints)
      myConstraints.removeAll()
      let views = ["text1":text1, "text2":text2]
    3 myConstraints += NSLayoutConstraint.constraints(withVisualFormat: "H:|-[text1]-[text2]",
                                                      options: [.alignAllFirstBaseline],
                                                      metrics: nil,
                                                      views: views)
      myConstraints += NSLayoutConstraint.constraints(withVisualFormat: "V:|-[text1]-|",
                                                      options: [],
                                                      metrics: nil,
                                                      views: views)
    3 NSLayoutConstraint.activate(myConstraints)
      super.updateConstraints()
  }
```

하나씩 작동하는 원리를 보자

1. UIView메서드의 updateConstraints를 재정의함
2. myConstraints의 모든 제약조건을 비활성화함
3. 우리가 방금 본 그 레이아웃을 구성하는 제약조건을 만든다.
4. 이러한 제약조건을 활성화 하고, 설치하고, super의 업데이트컨스트레인트를 호출함.

이 메서드가 동작하기 위해서는 UIView 레벨의 구현이 필요하기 때문이라는것 같음.

이는 잘 동작하는 코드지만 우리가 재정의 한 이 메서드가 정확히 뭔지 알아보자



이는 렌더루프의 기본 요소중 하나임
렌더루프는 초당 120회 실행되는 프로세스임
렌더루프는 모든 컨텐츠가 각 프레임에 대해 준비 되었는지 확인한다.

이는 3단계로 구성되있는데 제약조건 업데이트, 레이아웃, 디스플레잉 3단계이다.

1. 모든 뷰들이 Leaf부터 window까지 뷰 Hierarchy를 따라 올라가며 updateConstraints를 수신
2. 모든 뷰들이 window부터 Leaf까지 내려가며 layoutSubviews를 수신한다
3. 마지막으로 필요한 경우 모든 view가 그려진다

왜이렇게할까. 이들은 모두 똑같은(병렬적인 목적을 가지고있음,

The Render Loop

What is updateConstraints?

Update Constraints	Layout	Display
`updateConstraints()`	`layoutSubviews()`	`draw(_:)`
`setNeedsUpdateConstraints()`	`setNeedsLayout()`	`setNeedsDisplay()`
`updateConstraintsIfNeeded()`	`layoutIfNeeded()`	-

이 들의 공통적 목표는 낭비되는 작업을 피하며 연기하고 완전히 건너뛰게 하는것이다.

예를들어, UILabel에는 크기를 설명하는 제약조건이 있어야함. 그리고 크기를 구성하는데 있어서도 많은 속성들이 있음.
텍스트 속성, 글꼴, 텍스트 크기 등등...

이를 수행하는 한가지 방법은 이러한 속성중 하나가 변경될 때 마다 텍스트 크기를 다시 측정하는것임.
그러나 일반적으로 이러한 항목들은 연속적으로 변경되기 때문에 매우 비효율적임.
처음 label을 설정할 때 아마도 이런 property Setter를 호출할 것이고 각각의 텍스트를 다시 측정한다면 중간 것들이 낭비됨. 우리는 마지막에만 측정되길 원함

이게 renderloop가 제공하는것임
property setter 에서 setNeedsUpdateConstraints를 호출하면 프레임이 화면으로 이동하기 전에 updateConstraints를 호출하게 할 수 있다.

The Render Loop

What is updateConstraints?

Update Constraints	Layout	Display
`updateConstraints()`	`layoutSubviews()`	`draw(_:)`
`setNeedsUpdateConstraints()`	`setNeedsLayout()`	`setNeedsDisplay()`
`updateConstraintsIfNeeded()`	`layoutIfNeeded()`	-

계속 이야기 하기 전에 다시 짚고 넘어가 보자.

1. 초당 120프레임으로 많이 실행된다.
2. 병렬로 된다. 직감으로 사용할 수 있음 레이아웃 단계를 이해하고 있따고 느끼거나 그것에 대해 어느정도 느낌이 있다면, updateConstraints에 대해 생각하거나 디스플레이에 대해 생각할 때 만찬가지일것
3. 결국 이런게 존재하는건 낭비되는 작업을 피하고 작업을 연기하고 완전히 건너뛰기 위한것

```
// Don't do this! Removes and re-adds constraints potentially at 120 frames per second
override func updateConstraints() {
    NSLayoutConstraint.deactivate(myConstraints)
    myConstraints.removeAll()
    let views = ["text1":text1, "text2":text2]
    myConstraints += NSLayoutConstraint.constraints(withVisualFormat: "H:|-[text1]-[text2]",
                                                    options: [.alignAllFirstBaseline,]
                                                    metrics: nil,
                                                    views: views)
    myConstraints += NSLayoutConstraint.constraints(withVisualFormat: "V:|-[text1]-|",
                                                    options: [],
                                                    metrics: nil,
                                                    views: views)
    NSLayoutConstraint.activate(myConstraints)
    super.updateConstraints()
}
```

다시 코드로 돌아와보자.
우리는 호출될 때 마다 제약조건을 비활성화 한 다음 다시 새로운 제약조건을 활성화 함
이거는 layoutSubview랑 유사 > 모든 서브뷰를 remove, add하는거랑 동일함.

```
// Don't do this! Removes and re-adds constraints potentially at 120 frames per second
override func layoutSubviews() {
    text1.removeFromSuperview()
    text1 = nil
    text1 = UILabel(frame: CGRect(x:  20, y: 20, width: 300, height: 30))
    self.addSubview(text1)

    text2.removeFromSuperview()
    text2 = nil
    text2 = UILabel(frame: CGRect(x: 340, y: 20, width: 300, height: 30))
    self.addSubview(text2)
    super.layoutSubviews()
}
```

이렇게. 우리가 해야할건 한번 이상 하지 않도록 해야함.
어떻게? 미루도록 하자

```
// This is ok! Doesn't do anything unless self.myConstraints has been nil'd out
override func updateConstraints() {
    if self.myConstraints == nil {
        var constraints = [NSLayoutConstraint]()
        let views = ["text1":text1, "text2":text2]
        constraints += NSLayoutConstraint.constraints(withVisualFormat: "H:|-[text1]-[text2]",
                                                      options: [.alignAllFirstBaseline],
                                                      metrics: nil,
                                                      views: views)
        constraints += NSLayoutConstraint.constraints(withVisualFormat: "V:|-[text1]-|",
                                                      options: [],
                                                      metrics: nil,
                                                      views: views)
        NSLayoutConstraint.activate(constraints)
        self.myConstraints = constraints
    }
    super.updateConstraints()
}
```

우리가 이 작업을 이미 했다면 아무것도 하지않아야함.
아무것도 수행안했다면 한번은 제약조건을 설정해야함

The Render Loop

Do you really need to be part of it?

Useful: Avoiding wasted work

Dangerous: Runs a lot

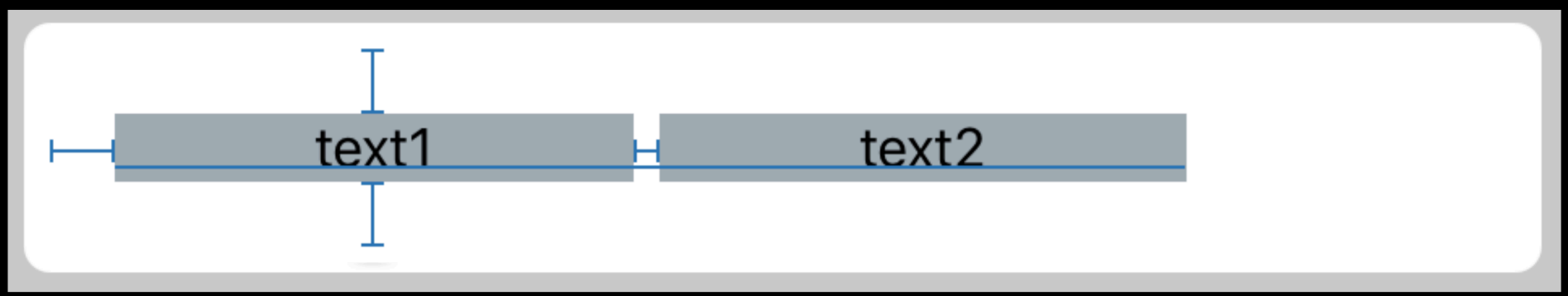
렌더루프는 실제로 불필요한 작업을 피하기 위해 유용함.

다만 자주 실행되서 위험할 수 있음. 민감함.

민감한 코드를 작성하는 빈도를 줄여야함.

한번만 수행하고 updateConstraints에 넣지 않을 수 있는지 다시 생각해야함.
이를 수행하는 좋은 방법은 InterfaceBuilder를 사용하는것.

Interface Builder



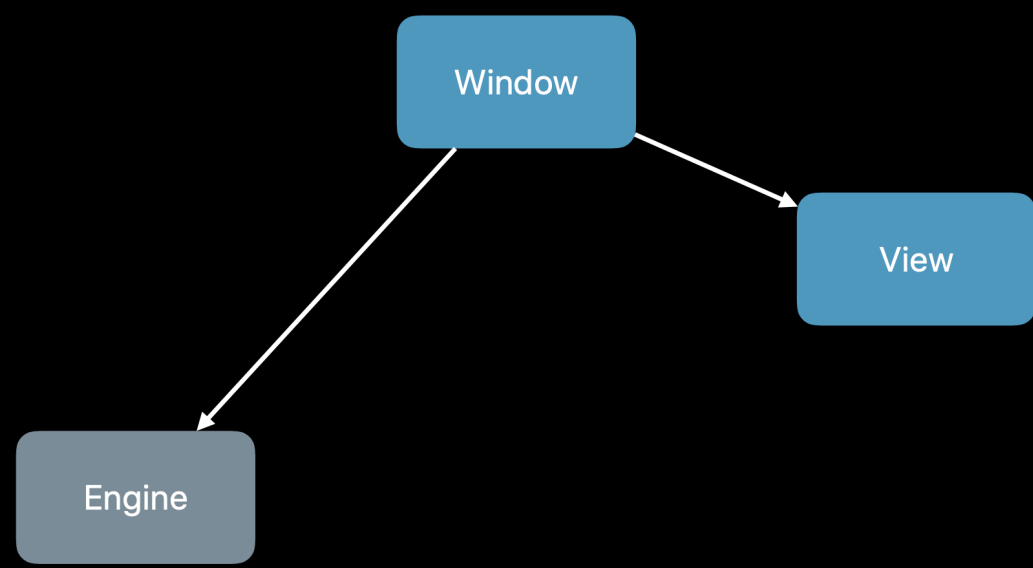
빌더를 쓴다면 좋은 길로 인도할 것

암튼 우리는 문제가 되는걸 잘이해했지만 이강연은 유추해서 이해하는걸로 끝나는게 아님.
더나은걸 원함

이제 실제로 무슨일이 일어나는지 볼 것.

Activating a Constraint

Internal structure

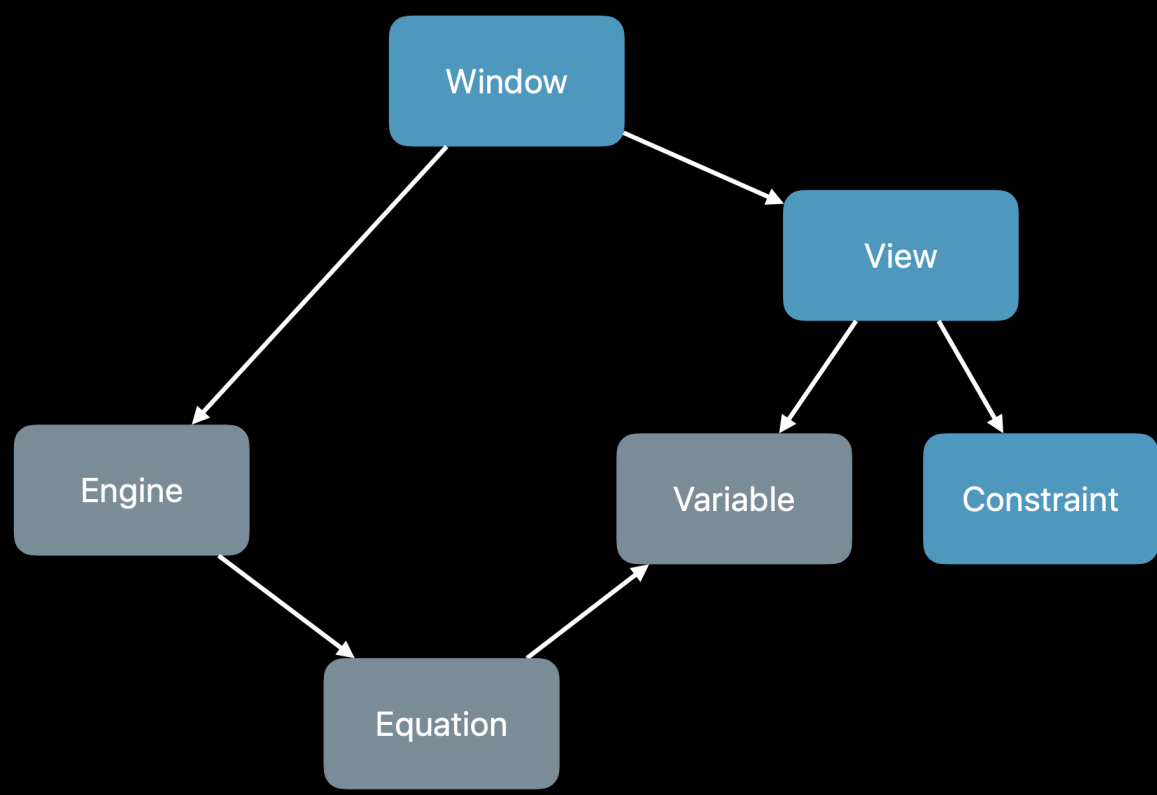


뷰가 존재함 이 뷰는 window에 있음. 이 뷰에 제약조건을 추가하고자 함

윈도우에는 Engine이라는 내부 객체를 들고 있음(매달려 있다, hanging off 라 한거 보니 윈도우에 의존적인 엔진이라는거같음)
이 엔진은 오토 레이아웃의 컴퓨팅 코어임. 제약조건이 추가되면 제약조건에 해당하는 방정식을 만들고 해당 방정식을 엔진에 추가함

Activating a Constraint

Internal structure

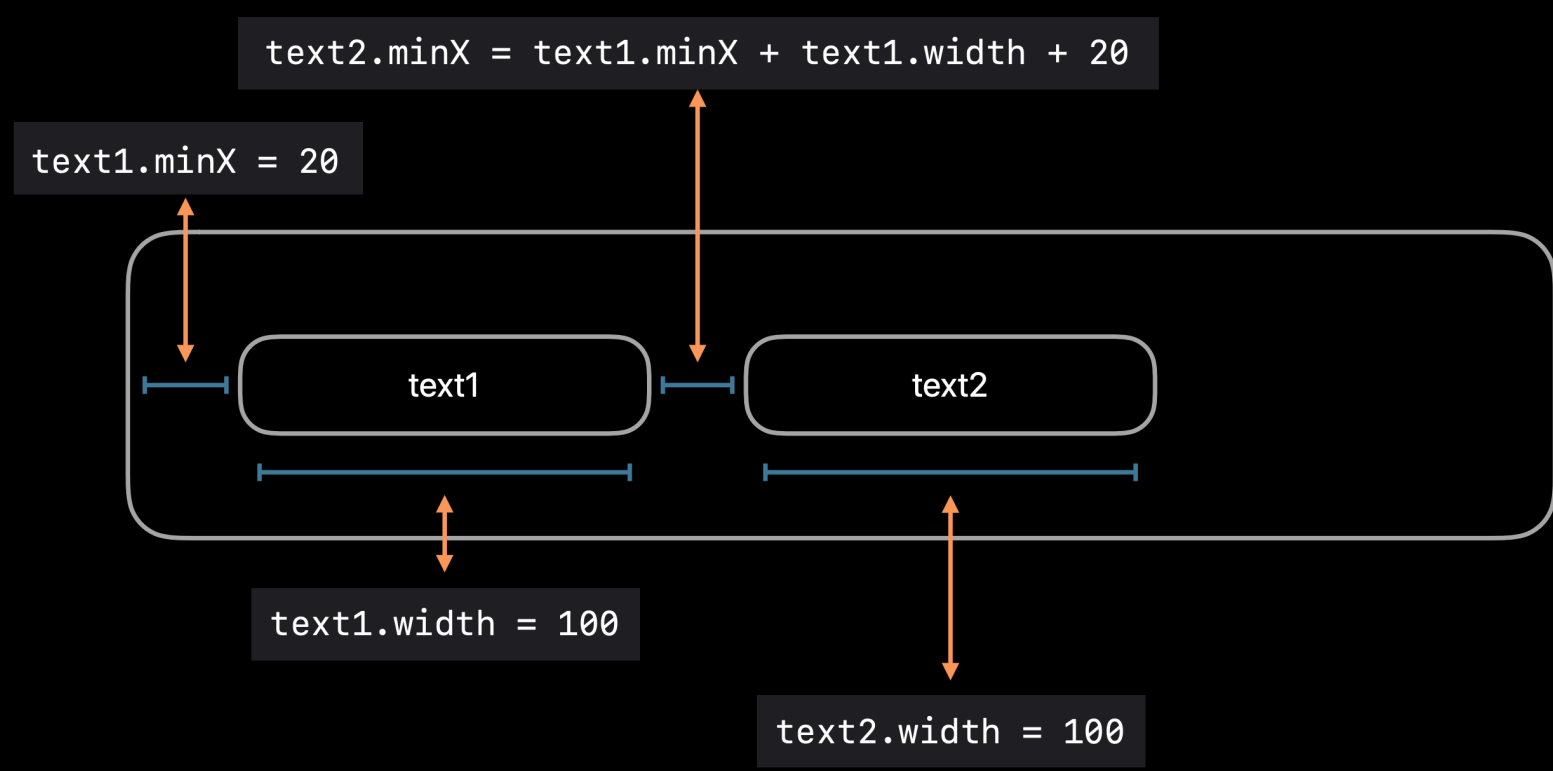


방정식(Equation)이 엔진에 추가되고나서 변수라는 개념이 생김

- 만일 방정식을 "X에 대한 해결"이라고 말한다면, "X는 변수"이다.

이 방정식에서 x는 뷰의 프레임 데이터임
즉 뷰가 추가되면 윈도우를 통해 엔진에 뷰를 그리기위한 방정식이 추가되고,
이 방정식에 들어가는 변수는 뷰의 프레임 데이터인것.

Activating a Constraint



모든 뷰에는 최소 x, y, 너비 높이의 4가지 변수가 존재

엔진은 이 변수를 해결하려고함

Engine

```
text1.minX  = 8
text1.width = 100
text2.minX = text1.minX + text1.width + 20
```

엔진은 대수학적으로 표시할 수 있게 수행함

1의 minx 구하고 너비 구하고
2의 minx는 식으로 구함

그리고 식을 쭉쭉 풀어서 각 값으로 대체하고 계산함

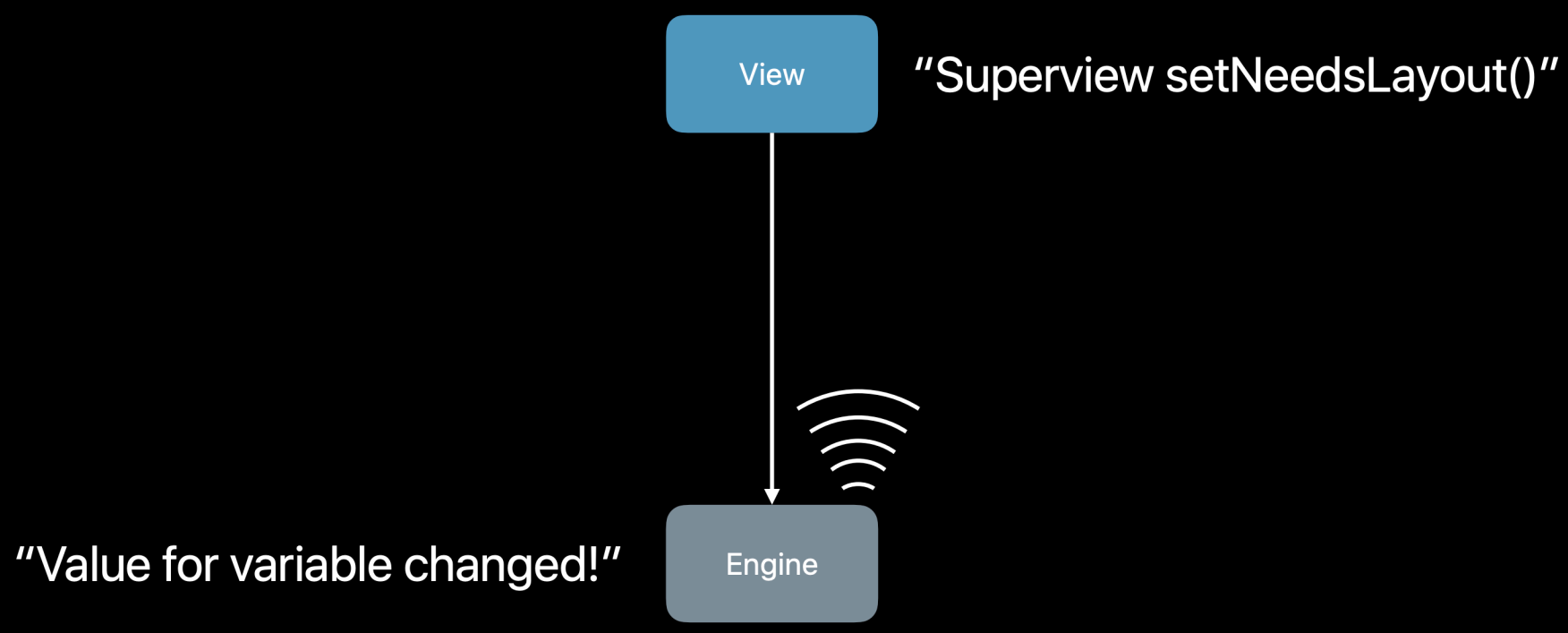
```
text1.minX  = 8
text1.width = 100
text2.minX = 128
text2.width = 100
```

이제 변수들이 전부 해결된 것 처럼 보임.

이런 작업은 비싸지 않음. 다만 이건 변수를 처리한거지 아직 제약조건이아님

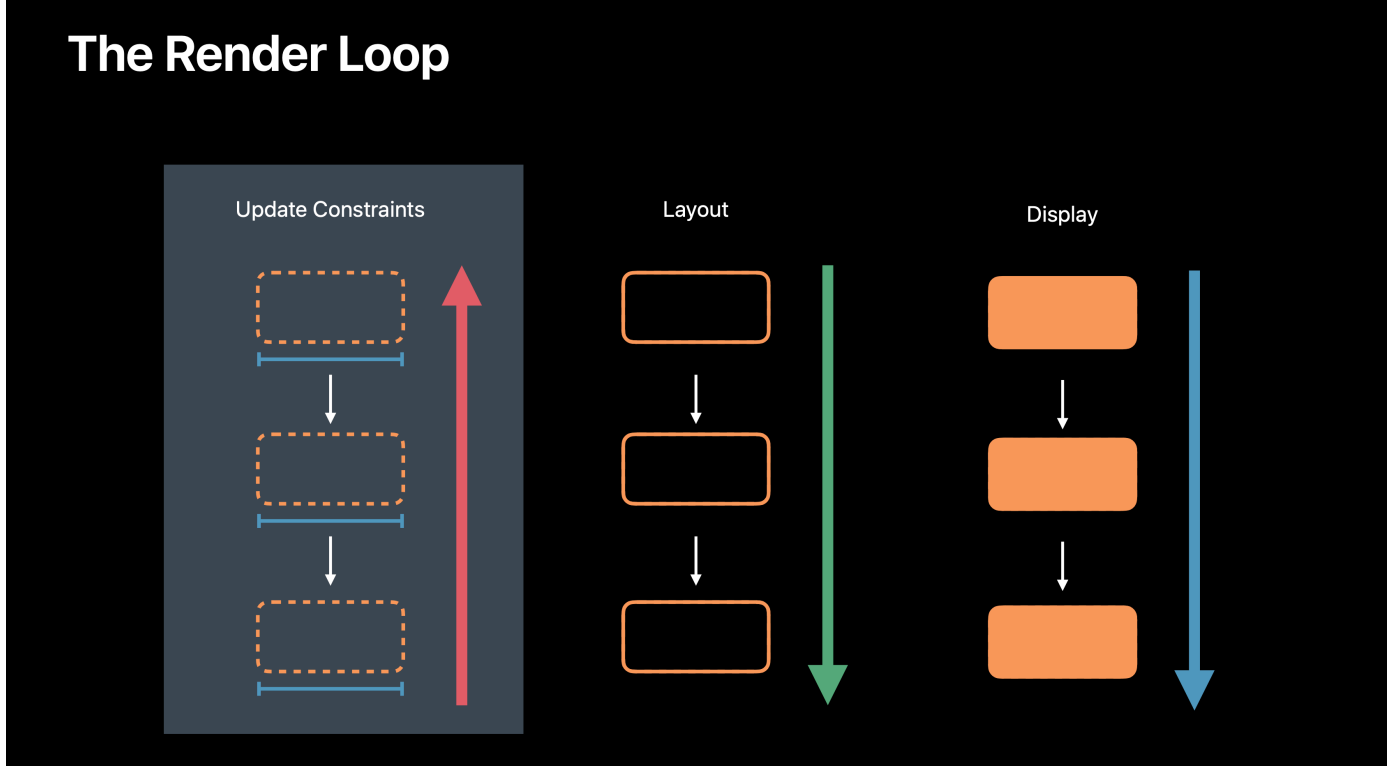
Activating a Constraint

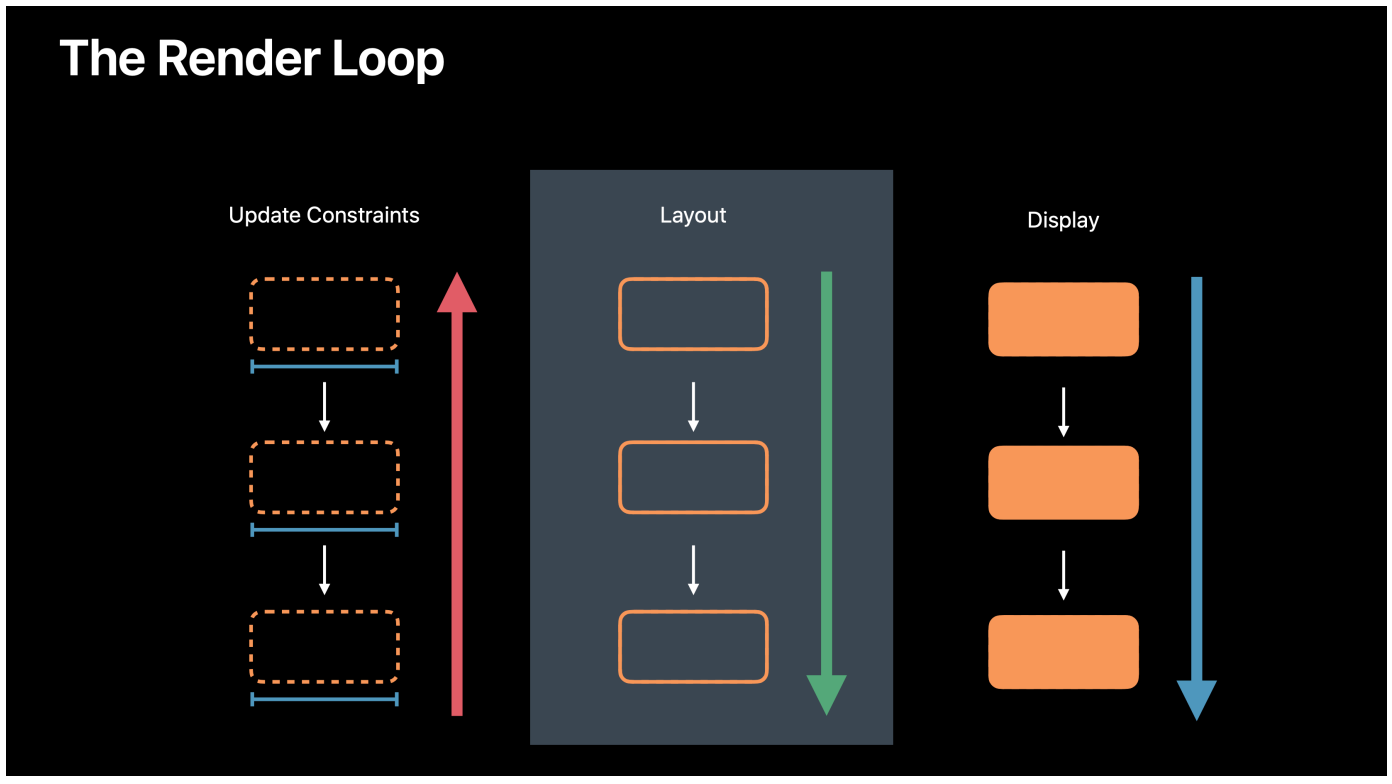
Triggering the layout pass



프로세스의 남은 부분은, 엔진이 이러한 변수 중 하나에 값을 할당할 때 마다 뷰에 변수의 출처를 알리고 이것이 변경되엇다고 하는것

이제 응답으로 슈퍼뷰의 SetNeedsLayout이라고 하는것.



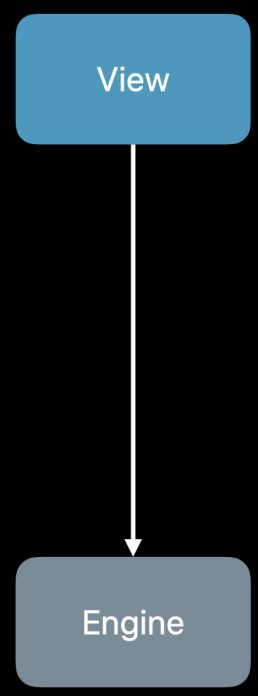


update constraints에서 setNeedsLayout이 호출되었으니 이제 Layout 단계로 넘어간다.

마지막은 UIView가 레이아웃을 수신하고 subView가 엔진에서 프레임으로 해당 데이터를 복사하는 것.

UIView.layoutSubviews()

Copying data from engine to subview



"Engine, what are
the values for the variables?"

"Subview setBounds(),
subview setCenter()"

layout단계에서 일어나는 LayoutSubview는 엔진에서 프레임으로 데이터를 복사하는 과정

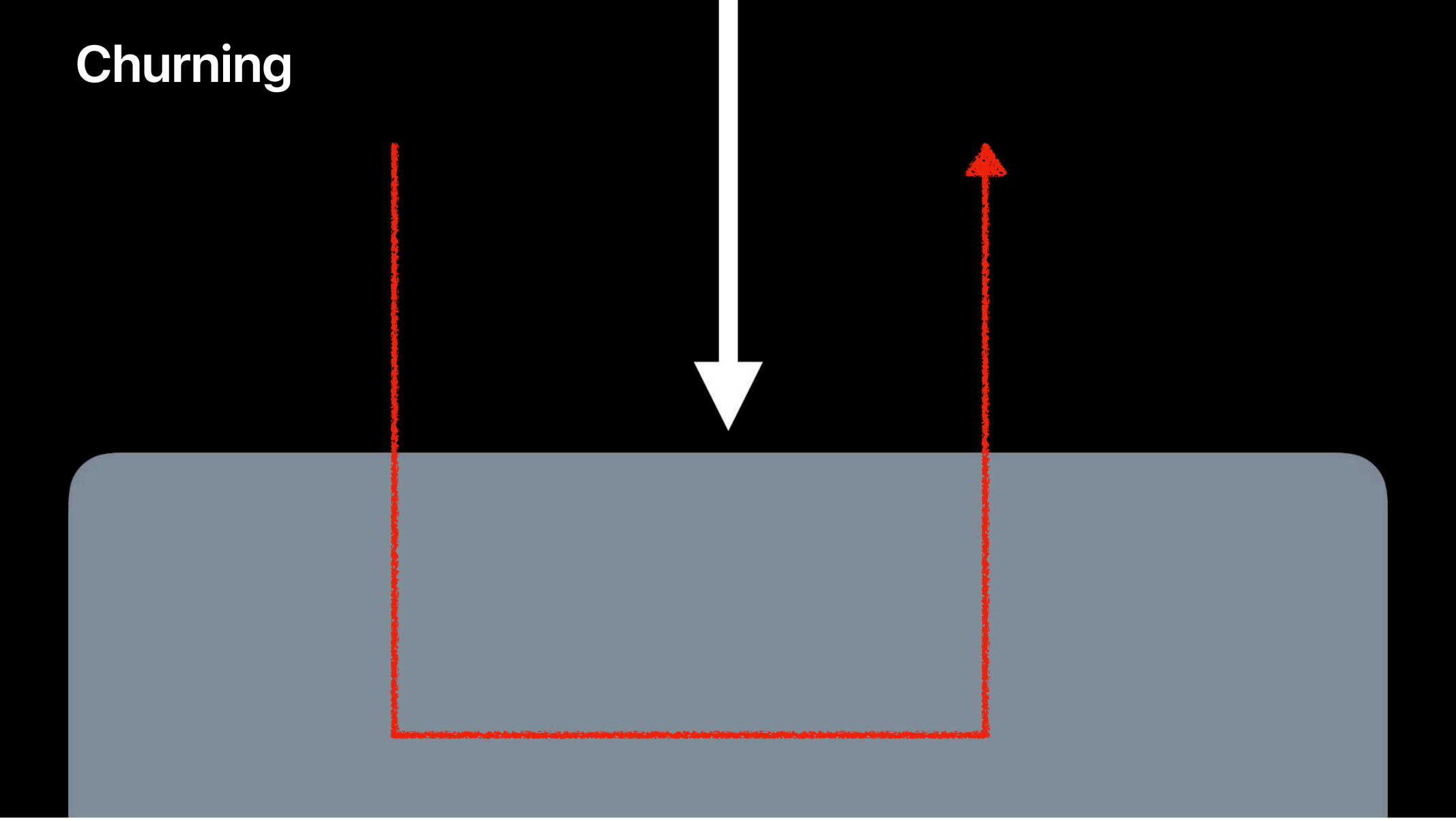
view는 엔진에 변수값이 뭐냐고 물어봄

엔진은 이를 알려주고 해당 뷰의 set SuperView를 호출하고 해당 하위뷰의 setCenter에서
setBounds를 호출

바드

```swift
// Don't do this! Removes and re-adds constraints potentially at 120 frames per second
override func updateConstraints() {
    NSLayoutConstraint.deactivate(myConstraints)
    myConstraints.removeAll()
    let views = ["text1":text1, "text2":text2]
    myConstraints += NSLayoutConstraint.constraints(withVisualFormat: "H:|-[text1]-[text2]",
                                                    options: [.alignAllFirstBaseline],
                                                    metrics: nil,
                                                    views: views)
    myConstraints += NSLayoutConstraint.constraints(withVisualFormat: "V:|-[text1]-|",
                                                    options: [],
                                                    metrics: nil,
                                                    views: views)
    NSLayoutConstraint.activate(myConstraints)
    super.updateConstraints()
}
```

실질적으로 어떻게 진행되고 있는지 보자
우리가 이 제약조건을 비활성화하고 다시 활성화하는 이 방법을 살펴보고
방금 수행한 작업에 대해 생각하고 다시 엔진이 무엇을 할지 생각해봐라

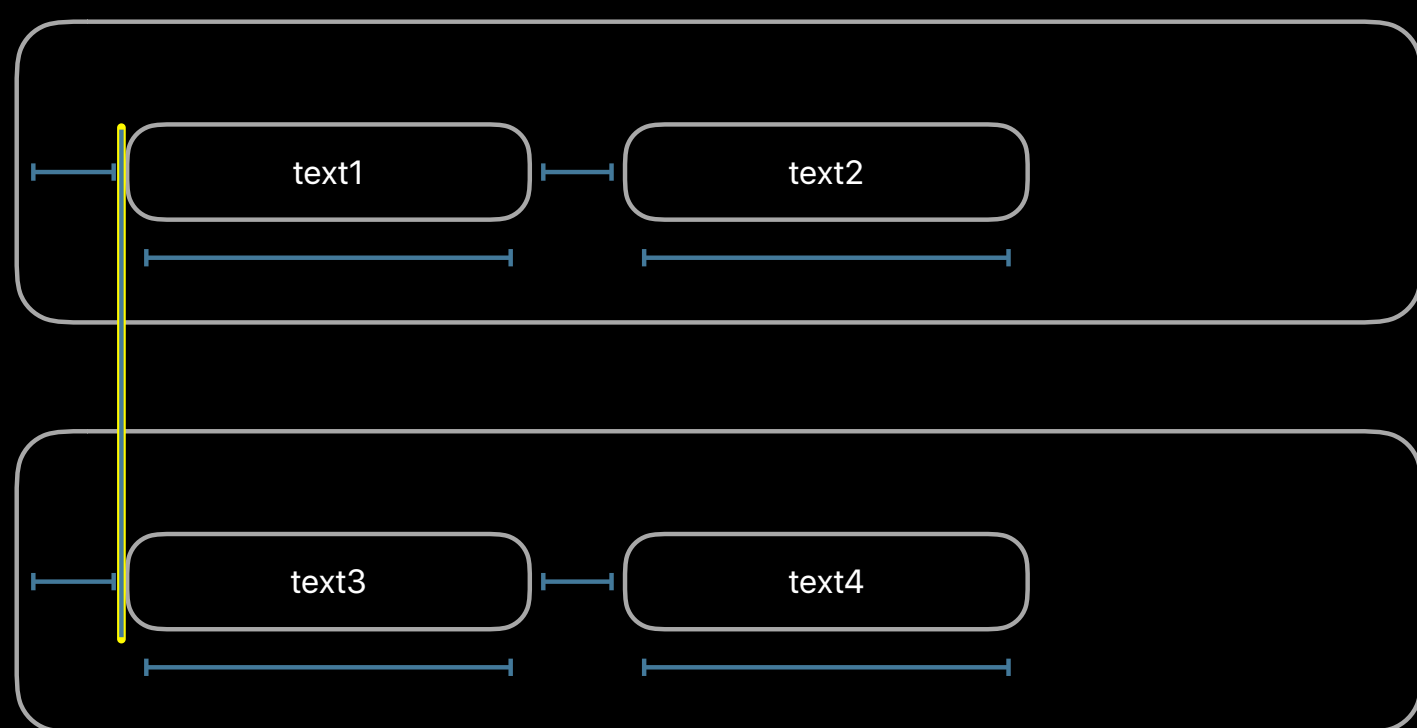


빨간 화살표대로 값들이 들어갈 것임 -> Churning(휘젓기)라고 부름

수행하는 각 작업은 매우 비싸지는 않지만, 많은 작업을 수행하고 있으며 완전히 불필요
이러한 작업은 낭비임

Local Versus Global Layout

You don’t pay for what you don’t use



정말 좋은 성능 모델을 원한다면 오토레이아웃으로 사용한 만큼만 비용을 지불해라

위 ppt를 보았을 때 두 종류의 독립적인 계층 구조에 4개의 텍스트필드가 있음

이제 우리가 해야하는 것은 이와 같이 계층 구조를 가로지르는 제약 조건을 만드는 것
즉, 동일한 superView가 없더라도 text1은 text3과 정렬되어야 함

사람들은 이것이 가능하기에 어떤것이 언제든지 어떤것에 영향을 미칠 수 있기 때문에 일반적으로 상당히 느려질 것이라는 인상을 받을 것임

대부분의 경우 뷰는 부모와 형제에게만 제한됨

Unrelated Views Don’t Interact

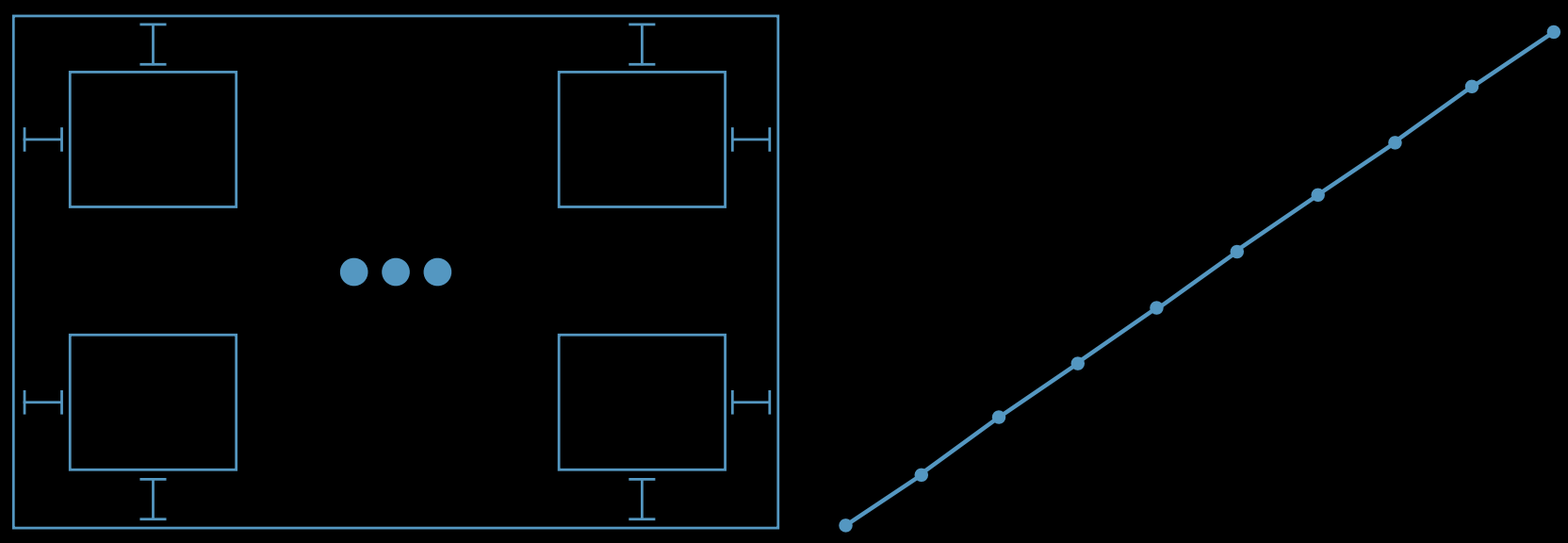
```
text1.minX  = 8
text1.width = 100
text2.minX  = 128
text2.width = 100
```

```
text3.minX  = 8
text3.width = 100
text4.minX  = 128
text4.width = 100
```

여기서 볼수 있는 것은 우리가 이 두 개의 독립적인 블록을 가지고 있기 때문에 엔진 내부를 보면 서로 완전히 상호작용하지 않는 두 개의 독립적인 방정식 블록이 있다는 것
한마디로 중첩변수가 없음

Performance Scales Linearly When No Interaction

How much work would it take by hand?



서로가 연관이 없으면 시간이 두배로 걸림
한 번 레이아웃을 잡을 때마다 연관이 없기에 오른쪽 선분이 늘어남
진행 방식이 선형인 이유는 이 레이아웃 사이에 어떠한 종속성도 없기 때문
종속성이 있는 경우 해당 방정식 블록을 함께 묶고 처리해야 할 계산이 더 많아지지만, 종속성이 있는 경우에만 해당됨

The engine is a layout cache
and dependency tracker.

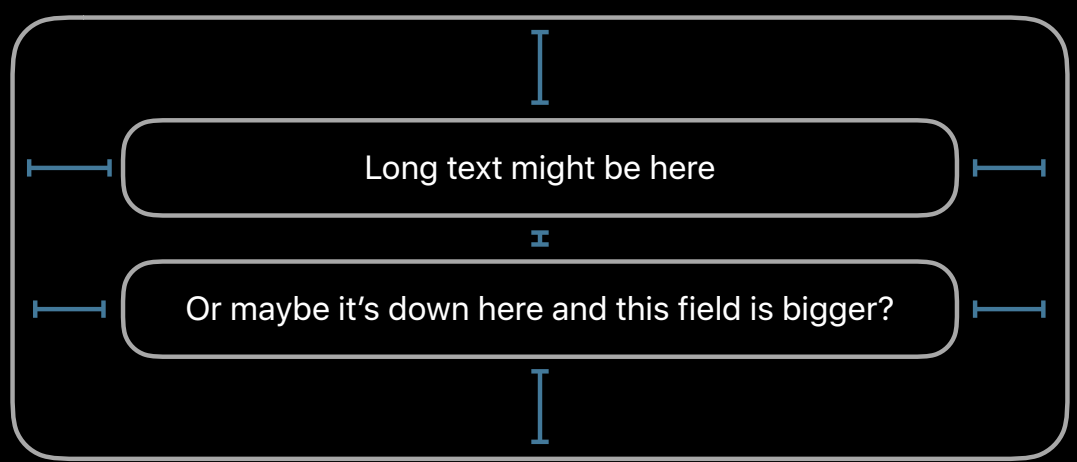
우리가 Cocoa에서 목표로 삼는 것은 단순한 것이 단순하고 복잡한 것이 가능하다는 것
종속성이 있다면 비용이 비싸지지만, 사용하지 않는다면 비용을 지불하지 않을 것

이것은 실제로 전체 엔진을 다시 직관의 관점에서 생각하는 올바른 방법
레이아웃을 위한 캐시와 종속성 추적기로 생각할 수 있음

어떤 제약 조건이 어떤 뷰에 영향을 미치는지 이해하고 변경하면 정확히 필요한 항목만 업데이트함
그리고 이것은 코드 작성 방법에도 영향을 미침

Model The Problem Naturally

Don’t avoid using constraints



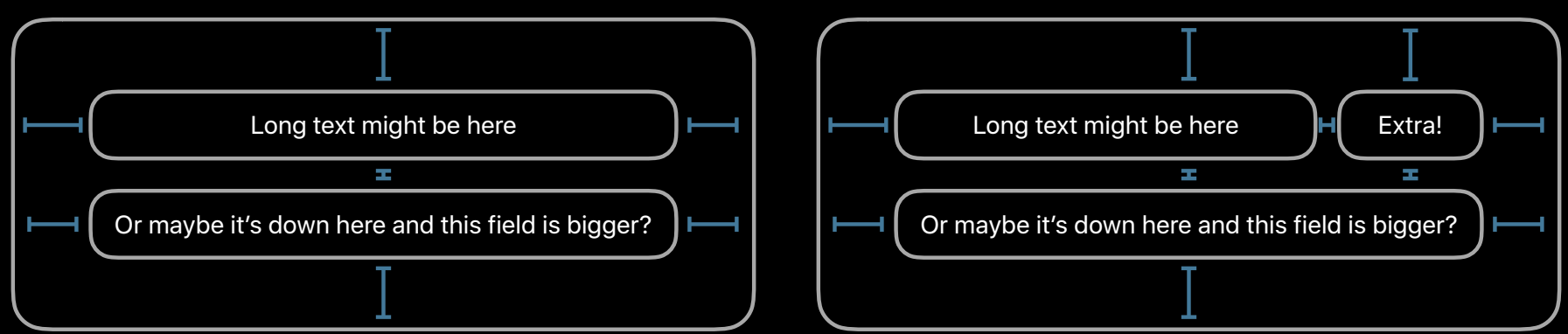
때때로 사람들이 비용이 많이 들 것이라는 인상을 가지고 있기 때문에 제약을 피하기 위해 많은 노력을 기울임

만들고 있는 제약이 해결되고 있는 문제와 밀접하게 일치하는 한, 당신이 무엇을 하든, 그것을 피하려고 하면 더 나은 성능이 될 가능성은 거의 없음

종종 우리는 사람들이 매우 복잡한 측정을 수행하고 정보를 추가하고 정보를 추출한 다음, 다시 밀어 넣으려고 시도하는 것을 보게 될 것
이는 거의 당신이 추구하는 것을 제약으로 직접 표현하는 것보다 비용이 많이 듬

Model The Problem Naturally

Don’t wedge two layouts into one set of constraints



그 반대의 경우는 때때로 우리가 이와같은 계층 구조를 볼 수 있다는 것
여기에서 우리는 많은 제약과 많은 우선 순위를 볼 수 있으며 실제로 무슨일이 일어나고 있는지 명확하지 않음

위 상황은 두 개의 완전히 분리된 레이아웃이 있는 상황이고 우리는 하나의 제약 조건 세트로 함께 묶고 모든 작업을 하나로 수행하려고 함

함께 묶어서 수행하는 것은 좋은 생각이 아님
그래서 그것은 사물이 실제로는 상호작용하지 않는 것처럼 보이는 곳에서 많은 잘못된 종속성을 생성함
알아채지 못한다면 디버깅도 거의 불가능함

따라서 전반적인 조언은 가능한 직접적으로 문제를 모델링 하는 것

결론은 오토레이아웃을 피하지 말고 잘 활용해라

Is That Everything?

Costs of other auto layout features

Inequality

NSLayoutConstraint set constant

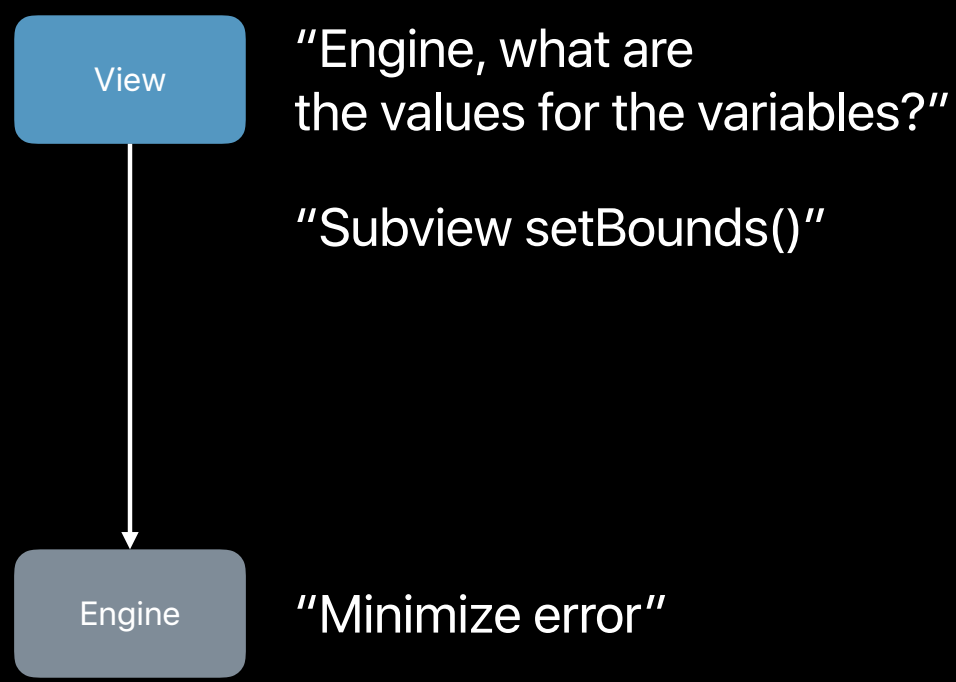
Priority

우리는 레이아웃 성능 특성에 대한 전반적인 정신적 모델을 구축하려고 노력하고 있으므로 최소한 모든 주요 기능을 다루고 싶음

일부 특정 뷰는 너비가 100포인트 이상이어야 한다고 말할 수 있음
이러한 상황에 Inequality를 사용할 수 있음
비용은 아주 조금밖에 안듬
변수 하나만 추가하거나, constant를 호출할 수도 있음

Error Minimization

Constraints with non-required priority



priority에 대해 더 얘기해보자
오류 최소화 단계를 보자

뷰가 레이아웃 하위 뷰의 일부로 엔진에 요청하고 이 변수의 값이 무엇인지 물어봄
엔진은 먼저 모든 오류가 최소화되었는지 확인을 함

Performance Intuition

Don’t churn

Basic algebra

Engine is a layout cache and tracker

You pay for what you use

이것이 오토레이아웃 주변의 성능 특성에 대한 직관적인 이해를 구축하려는 시도임

중요하지 않은 모든 작업을 수행할 때 제약을 휘저을려고 하지 마라
제약 조건을 사용하여 작업할 때는 기본 대수(Algebra)이며 제약 조건을 추가할 때, 제약조건을 제거할 때, constant 설정을 호출할 때 발생하는 대수학이 기본 시간임

when you do work with constraints it’s basic algebra and that algebra is happening when you add constraints, when you remove constraints, when you call set constant, that’s the primary times

Performance Intuition

Don’t churn

Basic algebra

Engine is a layout cache and tracker

You pay for what you use

오토 레이아웃이 하는 일에 대해 생각하는 방법은 레이아웃을 위한 캐싱이라는 것
우리는 엔진이 모든 해결된 값을 포함하고 종속성 추적기이므로 상황이 변경될 때 이러한 값을 슈퍼 타겟으로 업데이트 할 수 있다는 것을 알게 됨

사용중인 기능에 대해서만 비용을 지불한다는 것이 마지막임

#WWDC18

Building Efficient Layouts

Kasia Wawer, iOS Keyboards

© 2018 Apple Inc. All rights reserved. Redistribution or public display not permitted without written permission from Apple.

이번엔 효율적인 레이아웃 구축에 대해 이야기할 것 임

Constraint Churn

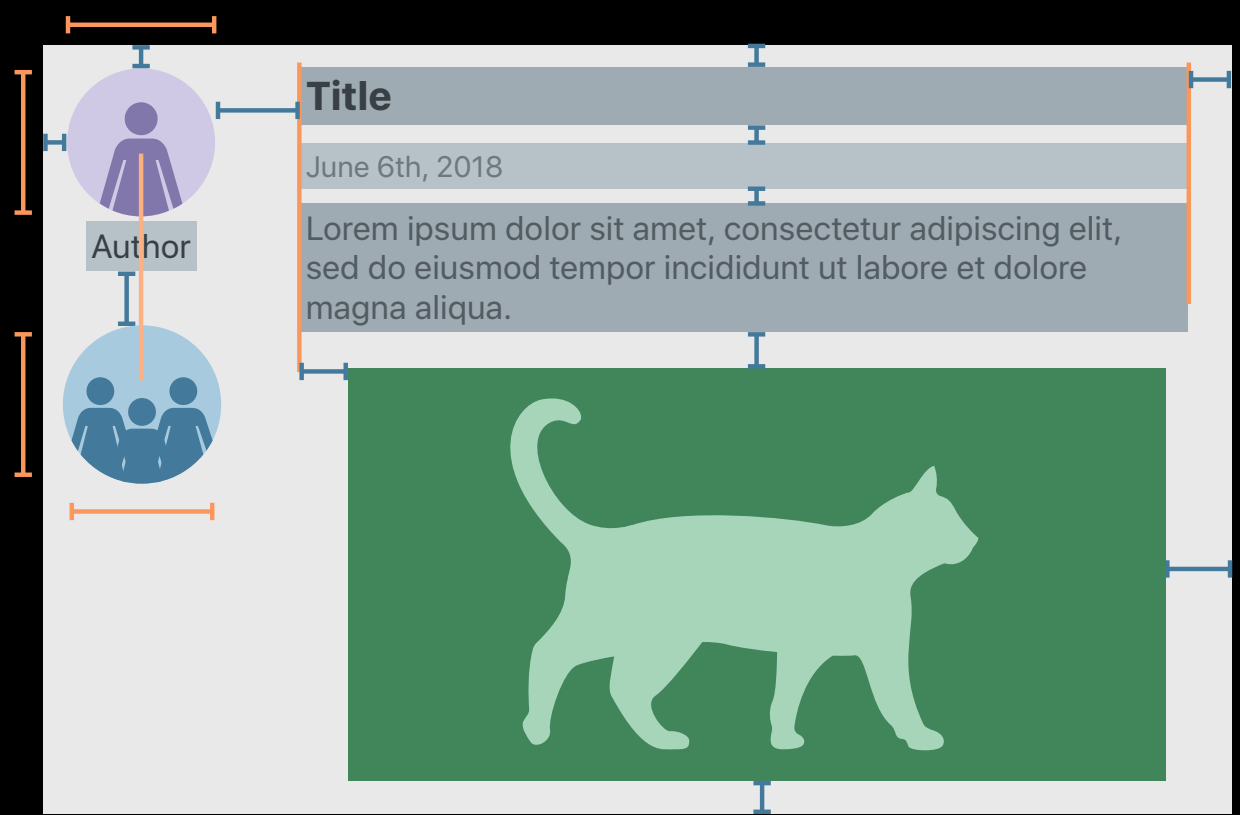
New solution, same layout

Extra work happens with no visible result

Enough churn can affect performance

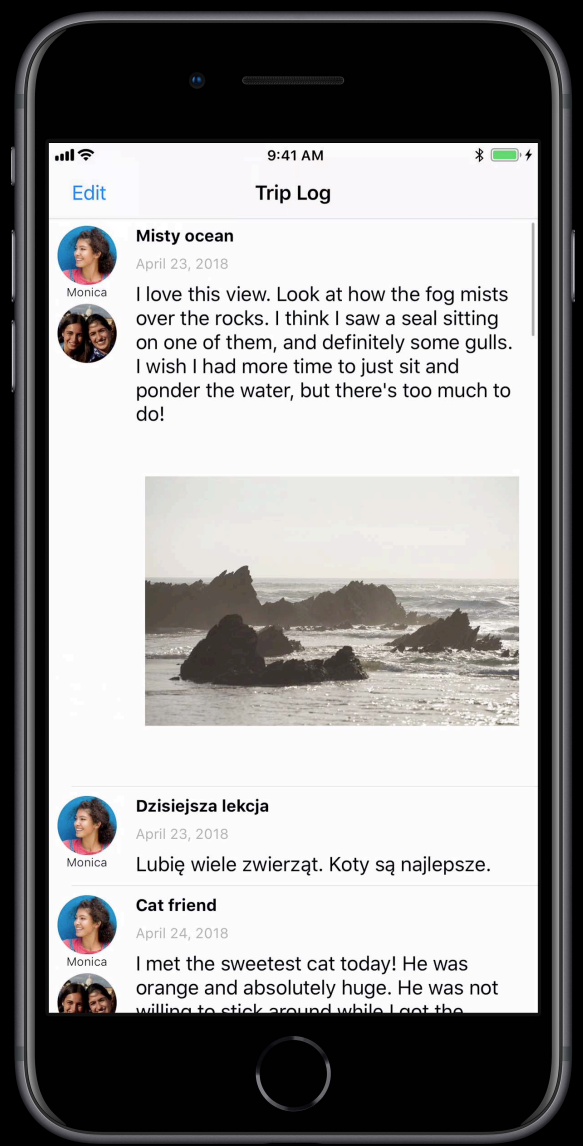
Constraint churn은 제약 조건을 변경할 때 발생하지만 실제 뷰는 이동할 필요가 없기 때문에 엔지에 추가작업을 보내고 그 정도가 성능에 영향을 미칠 수 있음
그래서 피하고 싶은 경향이 있음

Building a Layout



해당 뷰에서 어떤 사람은 공유 버튼을 빼고 싶고, 어떤사람은 고양이 이미지를 빼고 싶을 수도 있음

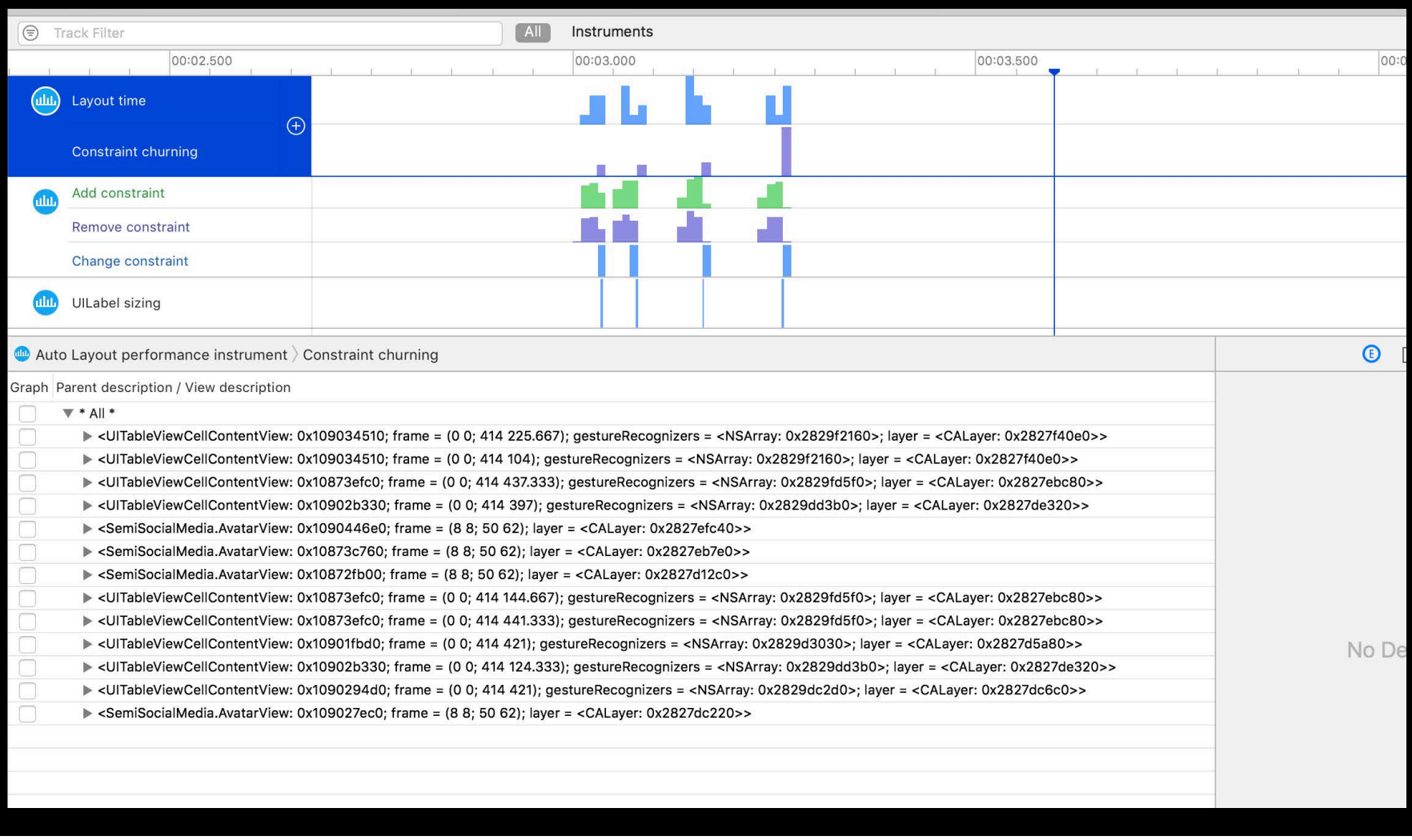
이러한 뷰들은 동일하지 않으면 테이블 뷰 셀이 화면에 나타날 때 약간의 조정이 필요함



스크롤 기능이 있는 앱은 어떻게 성능을 개선할 수 있을까?

데릭

Finding the Problem



상단 트랙은 얼마나 많은 CPU가 사용되고 있는지에 대한 표준이다.

피크가 많으면 레이아웃에서 확인해야 할 것이 있을 수 있음을 나타낸다.

그리고 꽤 평평한 경우 성능 문제가 다른 곳에서 발생했을 수 있다.

그래서 그 아래 Constraint 조건 변동을 구체적으로 추적할 것이다.

첫 번째 Bar
이 인스턴스의 막대 높이는 제약 조건 변동이 발생하는 뷰의 수에 해당한다.
- 따라서 큰 작품을 볼 때 많은 조회수가 영향을 받는다는 것을 알 수 있다.

두 번째 Bar
- 제약 조건 제거 및 변경하는 방법과 UILabel 및 기타 텍스트 뷰의 크기를 조정하는 방법을 보여줄 거다.

Finding the Problem

Auto Layout performance instrument 〉 Constraint churning

Graph	Parent description / View description
	▼ * All *
	▼ <UITableViewCellContentView: 0x10873efc0; frame = (0 0; 414 437.333); gestureRecognizers = <NSArray: 0x2829fd5f0>; layer = <CALayer: 0x2827ebc80>>
	<UILabel: 0x10873e7b0; frame = (74 8; 324 17); text = 'Quite a hike'; userInteractionEnabled = NO; layer = <_UILabelLayer: 0x2804b3e80>>
	<UILabel: 0x10873eaa0; frame = (74 55.3333; 324 81.3333); text = 'Hiked up the steps to the...'; userInteractionEnabled = NO; layer = <_UILabelLayer: 0x2804b8050>>
	<SemiSocialMedia.AvatarView: 0x10873c760; frame = (8 8; 50 62); layer = <CALayer: 0x2827eb7e0>>
	<UILabel: 0x108737f70; frame = (74 33; 324 14.3333); text = 'May 25, 2018'; userInteractionEnabled = NO; layer = <_UILabelLayer: 0x2804b3f70>>
	▼ <SemiSocialMedia.AvatarView: 0x10873c760; frame = (8 8; 50 62); layer = <CALayer: 0x2827eb7e0>>
	<UILabel: 0x10873e290; frame = (0 50; 50 12); text = 'Monica'; userInteractionEnabled = NO; layer = <_UILabelLayer: 0x2804b2df0>>
	<UIImageView: 0x10873e060; frame = (0 0; 50 50); opaque = NO; userInteractionEnabled = NO; layer = <CALayer: 0x2827eb660>>

이 보기를 강조 표시하고 Instruments의 상세 보기로 이동하면 보기 설명에 의한 변동의 영향을 받는 보기 목록이 표시됩니다.

그리고 Superview로 그룹화하여 테이블 뷰 셀의 인스턴스에서 다른 컨텍스트가 아닌 특정 컨텍스트에서 반복해서 발생하는 것을 쉽게 확인할 수 있다.

Finding the Problem

Auto Layout performance instrument 〉 Constraint churning

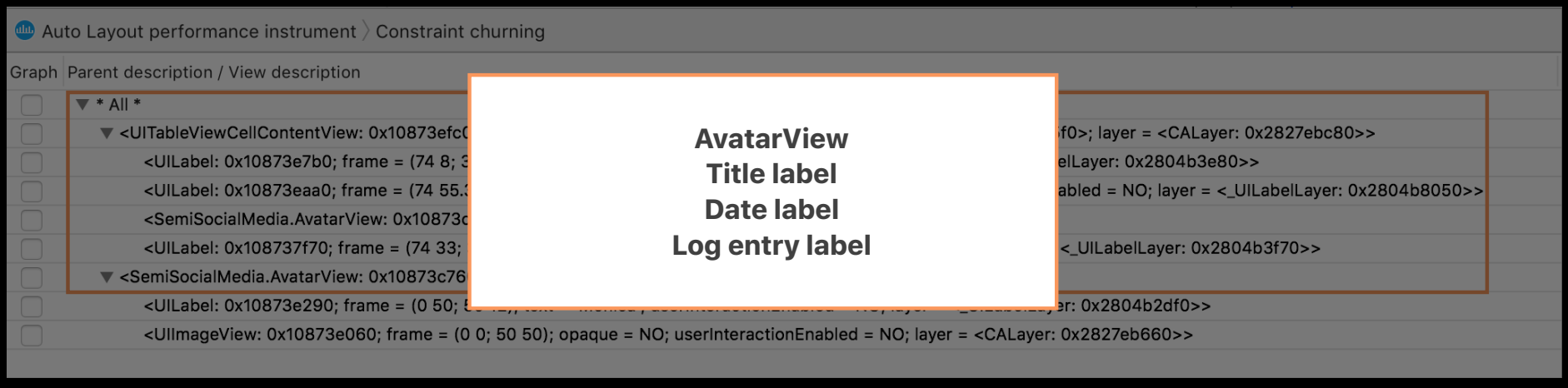
Graph | Parent description / View description

- * All *
  - <UITableViewCellContentView: 0x10873efc0; frame = (0 0; 414 437.333); gestureRecognizers
    - <UILabel: 0x10873e7b0; frame = (74 8; 324 17); text = 'Quite a hike'; userInteractionEnable
    - <UILabel: 0x10873eaa0; frame = (74 55.3333; 324 81.3333); text = 'Hiked up the steps to t
    - <SemiSocialMedia.AvatarView: 0x10873c760; frame = (8 8; 50 62); layer = <CALayer: 0x28
    - <UILabel: 0x108737f70; frame = (74 33; 324 14.3333); text = 'May 25, 2018'; userInteractio
  - <SemiSocialMedia.AvatarView: 0x10873c760; frame = (8 8; 50 62); layer = <CALayer: 0x2827e
    - <UILabel: 0x10873e290; frame = (0 50; 50 12); text = 'Monica'; userInteractionEnabled = N
    - <UIImageView: 0x10873e060; frame = (0 0; 50 50); opaque = NO; userInteractionEnabled = NO; layer = <CALayer: 0x2827eb660>>

SemiSocialMedia.AvatarView
UILabel "Quite a hike"
UILabel "May 25, 2018"
UILabel "Hiked up the steps to the..."

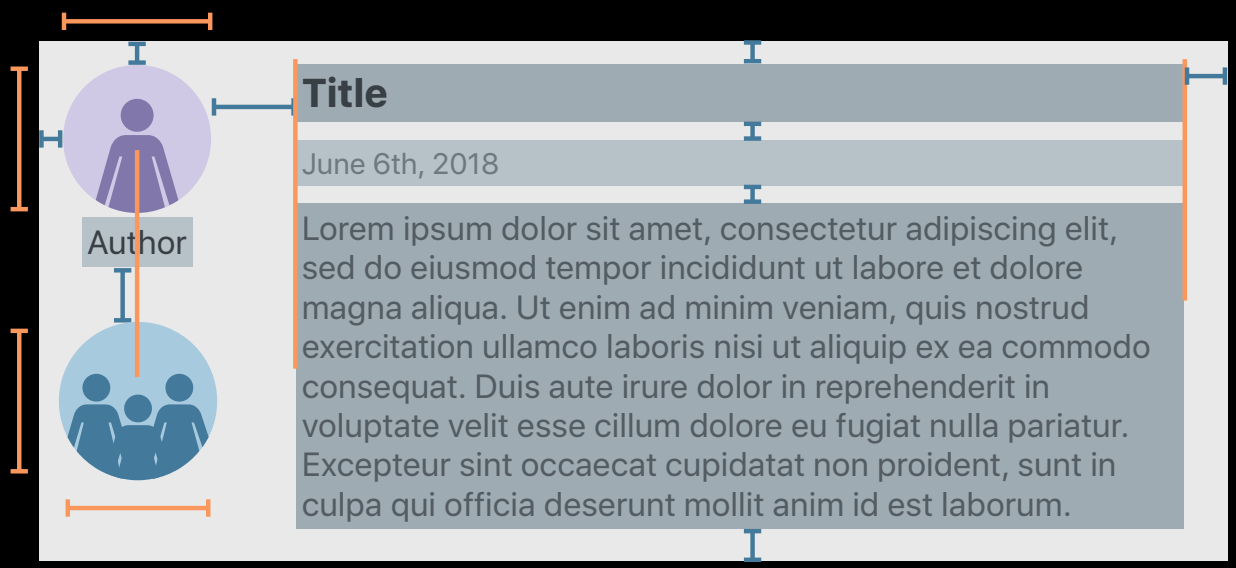
이 경우 아바타 뷰와 3개의 레이블에서 변동이 발생하는 것을 볼 수 있다.

Finding the Problem



그리고 기기를 통해 레이블이 제목 레이블, 날짜 레이블 및 로그 항목 레이블에 해당한다는 것을 알고 있다.

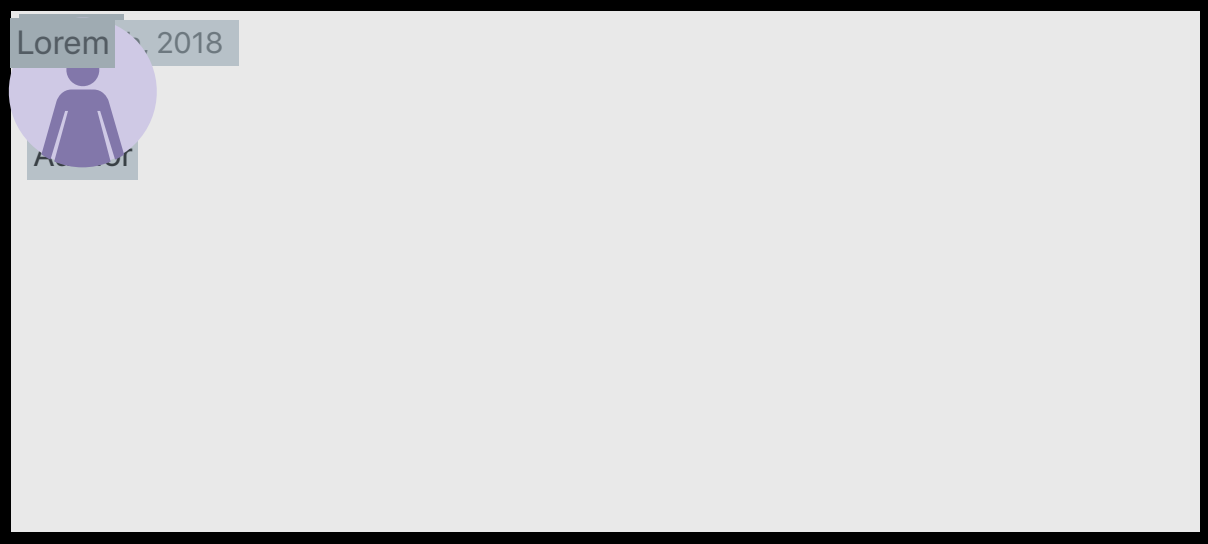
Building a More Performant Layout



UpdateConstraints가 재정의되고 있음을 확인

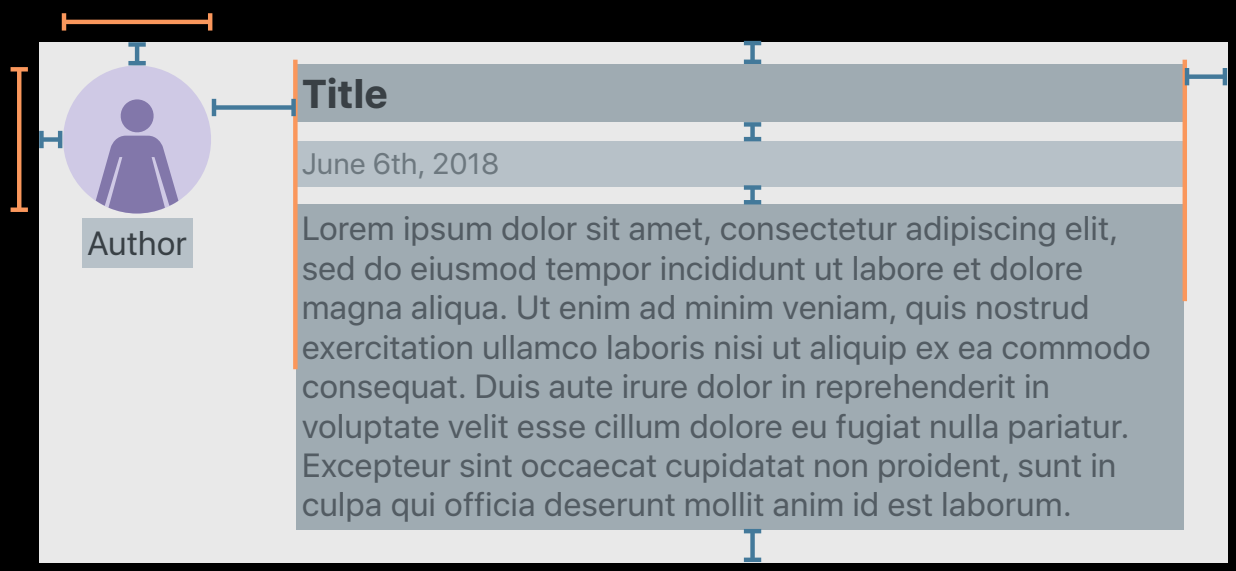
그리고 이 메서드에서 변경 사항이 있거나 UpdateConstraints가 실행될 때 모든 제약 조건을 제거한 다음 여전히 필요하다고 생각되는 제약 조건을 다시 추가한다.

Building a More Performant Layout



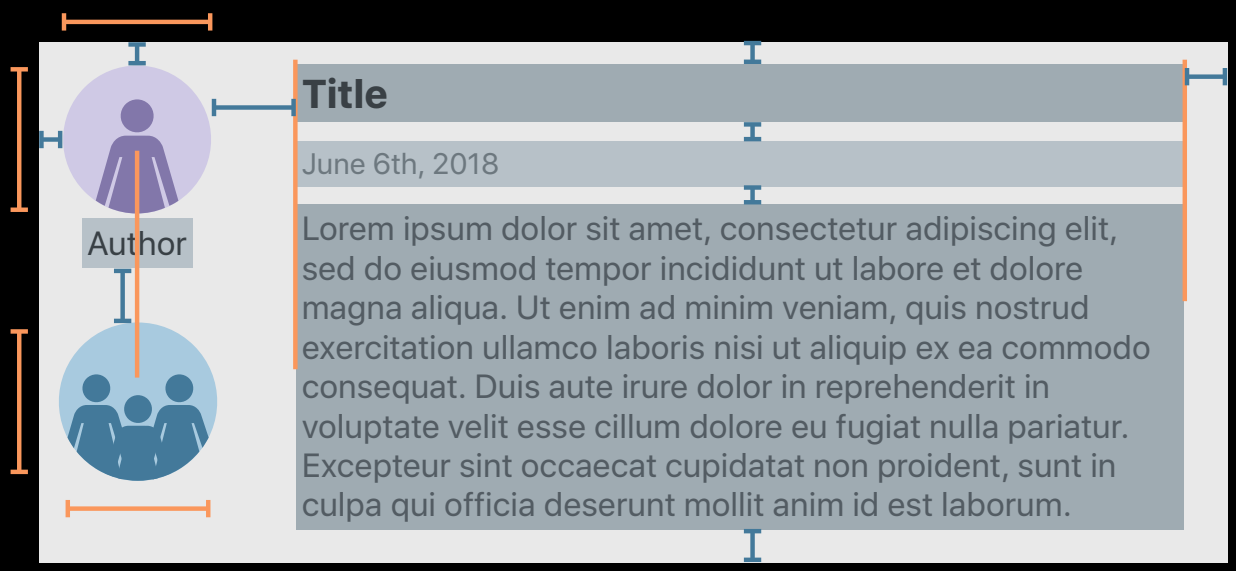
제거는 성능 문제에 기여

Building a More Performant Layout



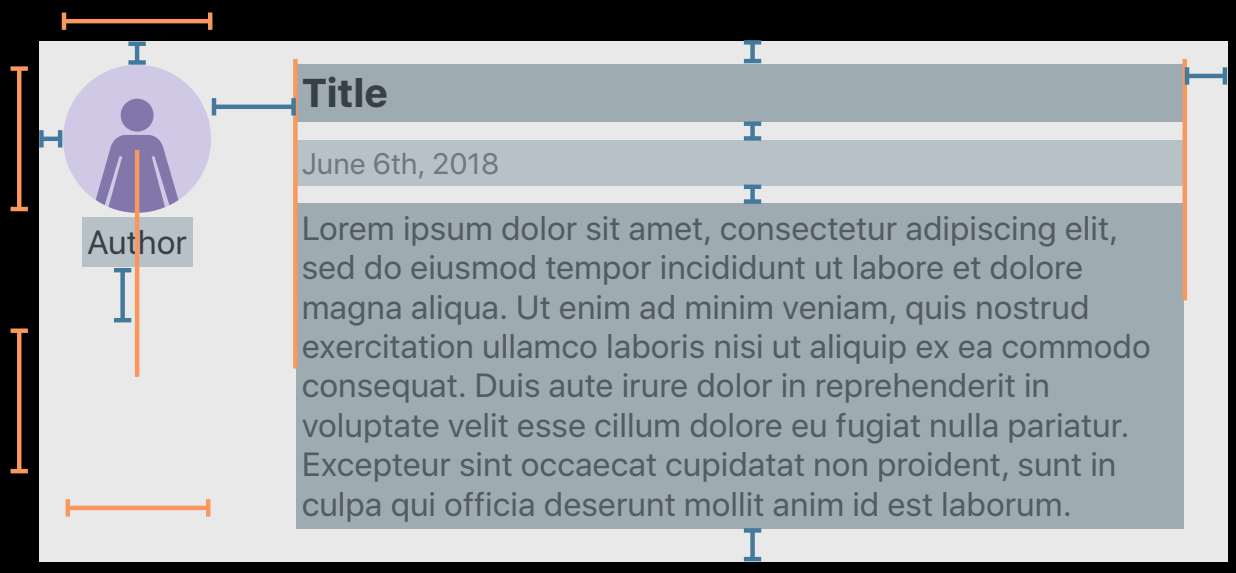
따라서, 여기 소셜 레이블의 경우 소셜 아바타가 추가되고 제거되므로 실제로 완전
제거할 필요는 없다

Building a More Performant Layout



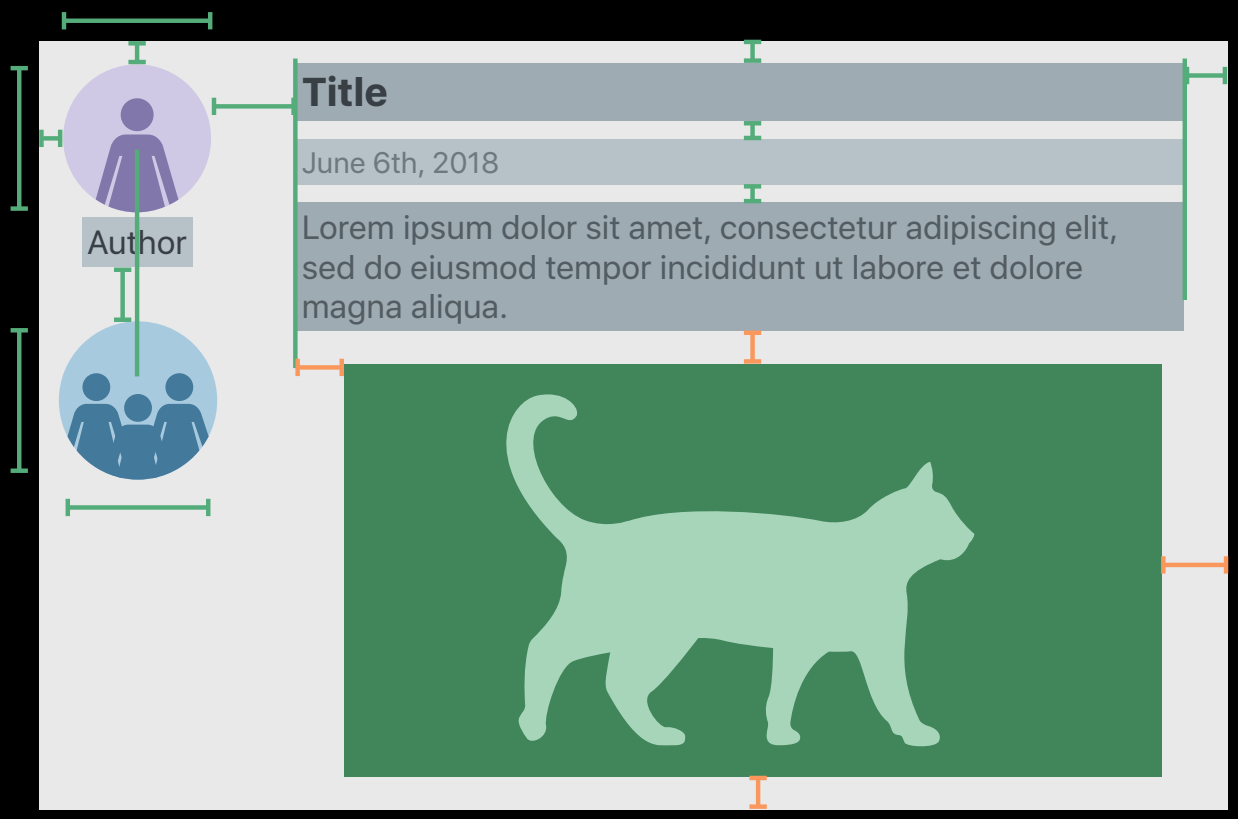
setHidden이라는 기능을 사용

Building a More Performant Layout



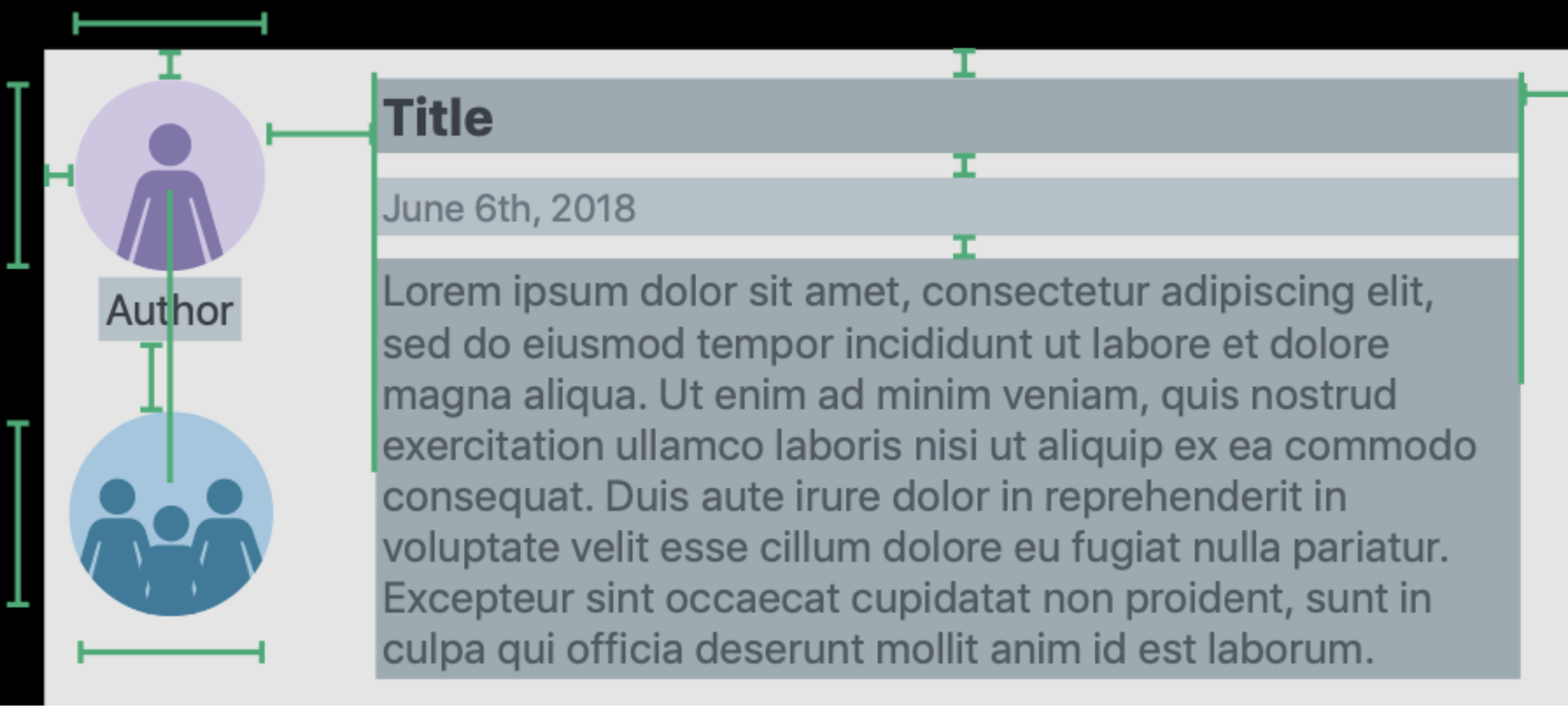
주변의 뷰에 영향을 주지 않기 때문에 그냥 사라지고 제약 조건이 그대로 유지되며
계층에서 뷰를 제거하는 대신 뷰를 숨기고 표시하는 매우, 매우, 매우 저렴한 방법

Building a More Performant Layout



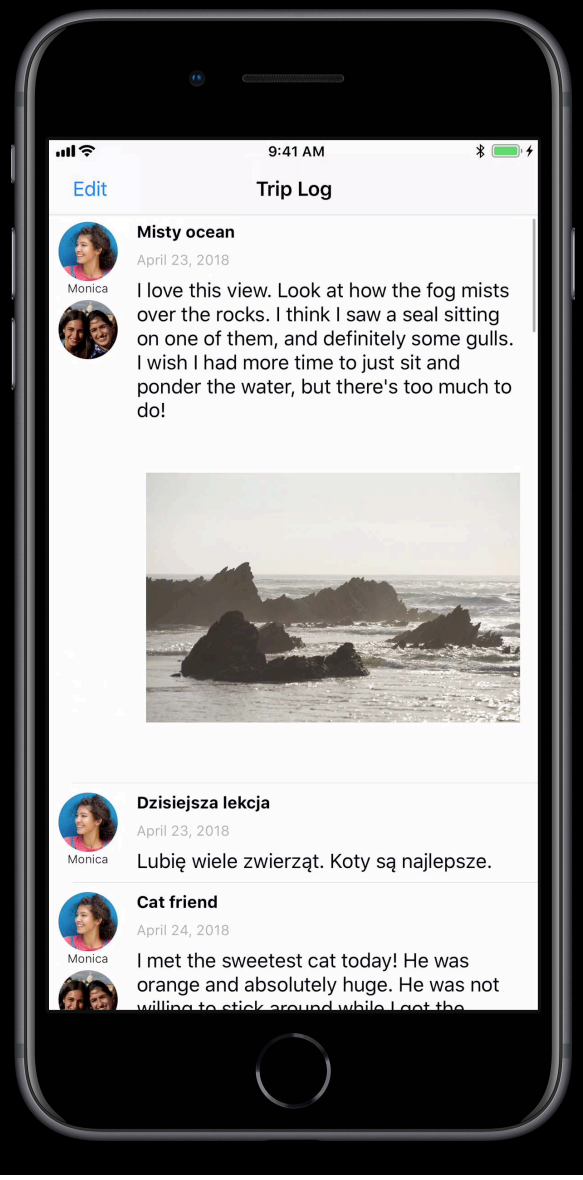

```
imageConstraints =
[ ⊢, I, ⊢, I ]

noImageConstraints =
[ I ]
```



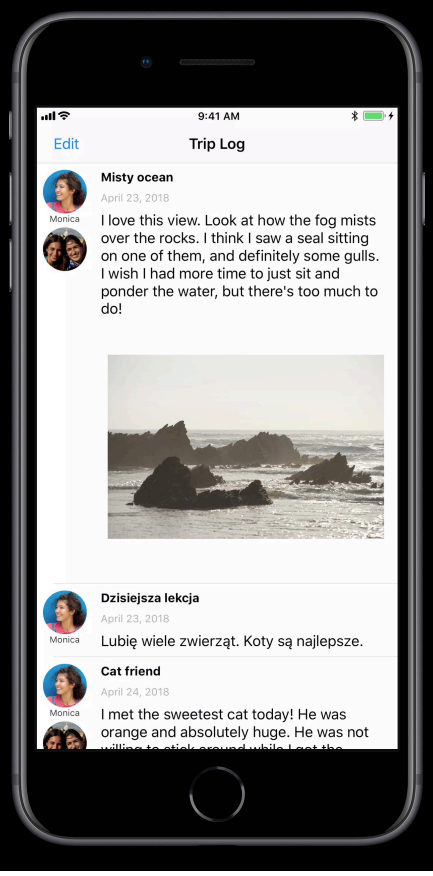
좋은 점은 이와 같이 제약 조건을 적절하게 추적하고 있고 이 이미지 보기를 사용자 앞에 실시간으로 추가하려는 경우 이러한 noImageConstraints를 비활성화하고 ImageConstraints를 활성화하고 필요한 내부에서 레이아웃을 호출할 수 있다는 것이다. 애니메이션 보기 블록을 사용하면 뷰에 애니메이션으로 표시된다.

Before

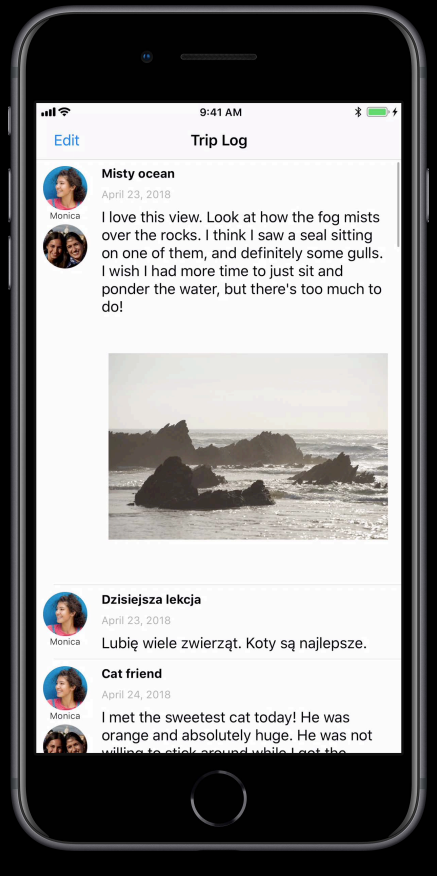


아주 안 좋음.

After (iOS 11)



After (iOS 12)



훨씬 더 부드러움

Constraint Churn

Avoid removing all constraints

Add static constraints once

Only change the constraints that need changing

Hide views instead of removing them

- 모든 제약 조건을 제거하지 마라.
  - 그러면 일반적으로 많은 프레임을 다시 넣어야 하는 상황에 놓이게 되고 필요하지 않은 프레임을 전달하거나 필요하지 않은 뷰를 전달하는 위치에 놓이게 된다.

- 레이아웃에 공통적으로 적용되는 일련의 제약 조건이 있는 경우 하나를 추가한 다음 그대로 둔다.
  - 이는 Interface Builder와 앱의 초기 레이아웃에 유용하다.

Intrinsic Content Size

Not all views need it

Views with non view content:

- UIImageView returns its image size
- UILabel returns its text size

UIView uses it to make constraints

- 모든 뷰가 실제로 본질적인 콘텐츠 크기를 필요로 하는 것은 아니다.
  - 콘텐츠가 있는 뷰는 뷰가 아닌 콘텐츠를 기반으로 고유한 콘텐츠 크기에 대한 크기를 반환하는 것을 좋아한다.

- 두 가지 예는 이미지 크기를 사용하여 고유 콘텐츠 크기를 계산하는 UIImageView와 텍스트를 측정하고 이를 사용하여 고유 콘텐츠 크기를 반환하는 UILabel이다
  본질적인 콘텐츠 크기는 UIView에서 제약 조건을 만드는 데 사용된다.
  제약 조건에서 모든 크기 조정을 직접 정의하고 이 모든 것을 건너뛸 수 있다.

Overriding

Optionally override for text measurement

- Return size if known without text measurement
- Use `UIView.noIntrinsicMetric` and constraints

```
override var intrinsicContentSize: CGSize {
    return CGSize(width: UIView.noIntrinsicMetric, height: UIView.noIntrinsicMetric)
}
```

- 그러나 재정의하면 성능에 도움이 될 수 있는 상황이 있다.

- 텍스트 측정은 비쌀 수 있다.

- 모든 텍스트 측정을 수행하지 않고 텍스트에 필요한 크기를 알고 있는 경우 해당 크기와 고유한 콘텐츠 크기를 반환할 수 있다.
  - 또는 이 뷰를 화면에 표시할 때 제약 조건이 완전히 내부 텍스트의 크기에 관계없이 크기를 정의한다. 예를 들어 제약 조건은 항상 텍스트의 양보다 더 크게 만들 것이다. 그런 다음 고유 콘텐츠 크기의 너비와 높이에 대한 고유 메트릭을 반환하지 않고 계속 진행할 수 있다.

System Layout Size Fitting Size

```
systemLayoutSizeFitting(_ targetSize:)
```

Used when combining with frames

- 본질적인 콘텐츠 크기는 엔진에 넣을 크기 정보를 전달하는 방법이다.

- 시스템 레이아웃 크기 피팅 크기는 엔진에서 크기 정보를 다시 가져오는 방법이다
  - 이것은 자동 레이아웃을 사용하여 하위 뷰를 관리하는 뷰에서 프레임 정보가 필요한 이유가 있는 일종의 혼합 레이아웃에서 사용

Getting Size from the Engine

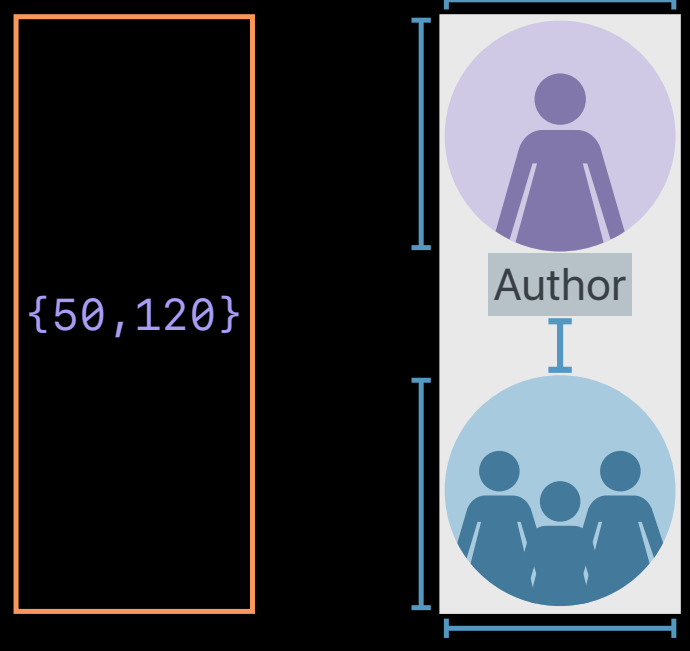
Engine is created

Constraints are added

Layout is solved

Size of the top view is returned

Engine is discarded



- 시스템 레이아웃 크기 피팅 크기를 호출하면 엔진이 생성된다.

  제약 조건이 이 엔진에 추가되고 레이아웃이 해결된 다음 상위 뷰 프레임의 크기가 반환된 다음 엔진이 삭제된다. 따라서 이 메서드를 호출할 때마다 엔진이 생성되고 폐기된다.

- 때때로 사용하는 용도 중 하나는 자체 크기 조정 컬렉션 또는 테이블 뷰 셀에서 콘텐츠 뷰로 호출을 전달하는 것
  - 그렇게 하면 실제로 스크롤을 만들기 위해 만든 일부 최적화를 재정의하고 해당 보기에서 더 빠르게 스크롤하고 추가 엔진을 추가하게 된다

Getting Size from the Engine

Engine is created

Constraints are added

Layout is solved

Size of the top view is returned

Engine is discarded

{50,120}

- 만족할 수 없는 제약 조건은 무엇입니까?
  이 뷰는 50포인트 너비여야 하고 또한 200포인트 너비여야 할 때
  - 엔진은 사용 가능한 레이아웃이 없다는 사실을 계산하고 일종의 레이아웃을 생성하기 위해 제약 조건을 깨뜨려야 한다.
  - 해당 제약 조건을 위반하면 디버거에 매우 자세한 로그를 보낸다.

PPT 페이지 명시할 것 피피티 페이지 링크 걸수 있음
상세한 것은 각 페이지에 주석으로 달기

	궁금한 점	이해 안된 점
보리		

PPT 페이지 명시할 것 피피티 페이지 링크 걸수 있음
상세한 것은 각 페이지에 주석으로 달기

	궁금한 점	이해 안된 점
바드		

PPT 페이지 명시할 것 피피티 페이지 링크 걸수 있음
상세한 것은 각 페이지에 주석으로 달기

	궁금한 점	이해 안된 점
데릭		