



APP SCHOOL : iOS

앱 개발 심화

UIKit과 AutoLayout

UIKit과 SwiftUI

다시 소개합니다

- UIKit이란?
 - 2007년도에 만들어진 개발 프레임워크로 iOS와 iPadOS를 위한 앱 개발에 활용되어 왔습니다.
 - iOS용 앱의 사용자 인터페이스를 구성하기 위한 주요 프레임워크로, 현재에도 아이폰에서 작동하는 대다수의 앱들이 UIKit 기술을 기반으로 개발되고 있습니다.

UIKit과 SwiftUI

다시 소개합니다

- SwiftUI란?
 - 2019년도에 발표된 개발 프레임워크로 선언형 사용자 인터페이스 개발 패러다임으로 UIKit보다 더 간결하고 직관적인 코드 작성은 가능하게 해줍니다.
 - SwiftUI는 단일 기술로 iOS뿐만 아니라 iPadOS, macOS, tvOS, watchOS, visionOS 까지 애플의 최신 플랫폼용 앱 개발을 두루 지원합니다.

UIKit과 SwiftUI

다시 소개합니다

- 지금까지 UIKit의 중요성은 누구도 무시할 수 없습니다.
- 그리고 SwiftUI의 활용사례도 날이갈수록 커지고 있어, 이 두가지를 모두 알고 융합해 사용하는 역량이 중요해지고 있습니다.

UIKit과 SwiftUI

우리의 최종 목표

- 기존 UIKit 프로젝트에 SwiftUI 코드를 추가하기
- 새로운 SwiftUI 프로젝트에 UIKit 코드를 추가하기

developer.apple.com

News Discover Design Develop Distribute Support Account

UIKit Documentation / UIKit Language: Swift API Changes: Show

Framework

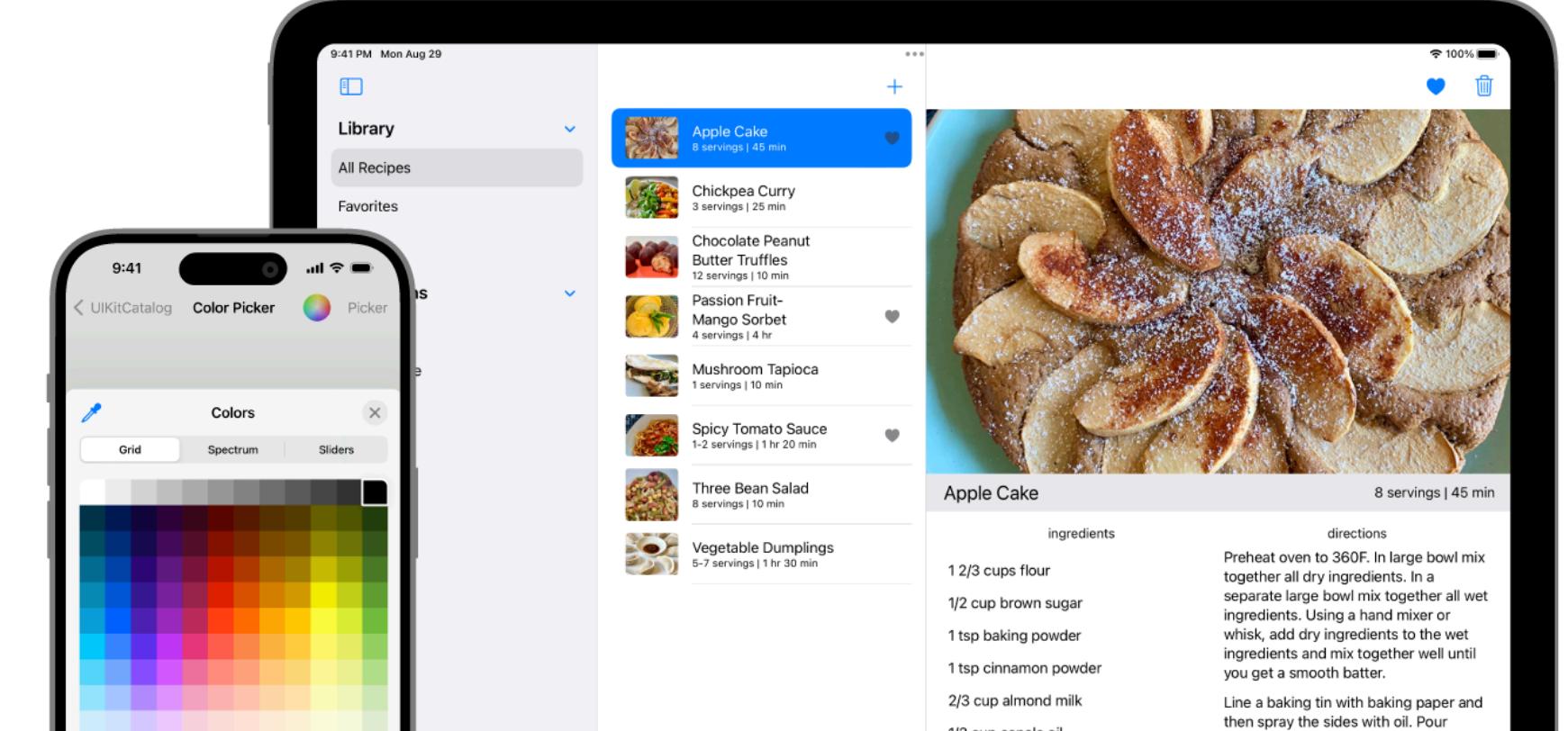
UIKit

Construct and manage a graphical, event-driven user interface for your iOS, iPadOS, or tvOS app.

iOS 2.0+ iPadOS 2.0+ Mac Catalyst 13.0+ tvOS 9.0+ watchOS 2.0+ visionOS 1.0+ Beta

Overview

UIKit provides a variety of features for building apps, including components you can use to construct the core infrastructure of your iOS, iPadOS, or tvOS apps. The framework provides the window and view architecture for implementing your UI, the event-handling infrastructure for delivering Multi-Touch and other types of input to your app, and the main run loop for managing interactions between the user, the system, and your app.



Essentials

- UIKit updates
- About App Development with UIKit
- > Protecting the User's Privacy

App structure

- > App and environment
- > Documents, data, and pasteboard
- > Resource management
- > App extensions
- > Interprocess communication
- > Mac Catalyst

User interface

- > Views and controls
- > View controllers
- > View layout
- > Appearance customization
- > Animation and haptics
- > Windows and screens

User interactions

- > Touches, presses, and gestures
- > Menus and shortcuts
- > Drag and drop
- > Pointer interactions
- > Pencil interactions

Filter /

developer.apple.com

Apple Developer News Discover Design Develop Distribute Support Account

UIKit Documentation / UIKit / View layout Language: Swift API Changes: Show

Essentials

- UIKit updates
- About App Development with UIKit
- > Protecting the User's Privacy

App structure

- > App and environment
- > Documents, data, and pasteboard
- > Resource management
- > App extensions
- > Interprocess communication
- > Mac Catalyst

User interface

- > Views and controls
- > View controllers

> View layout

Essentials

- > **UIStackView**

Localization

- { } Autosizing Views for Localization in iOS

Constraints

- Positioning content within layout margins
- Positioning content relative to the safe area
- > **NSLayoutConstraint**
- > **UILayoutSupport**

Filter /

API Collection

View layout

Use stack views to lay out the views of your interface automatically. Use Auto Layout when you require precise placement of your views.

Overview

When you design your app's interface, you position views and other interface elements in your app's windows and size them appropriately. However, the size and position of those views may need to change at runtime for a variety of reasons:

- The user can resize the window containing your views.
- Variations in the screen sizes of iOS devices (including differences between portrait and landscape orientations) require different layouts for each device and orientation.
- Apps on iPad must adapt to cover different amounts of screen space, ranging from a third of the screen to the entire screen.
- Language changes might require size changes for labels and other text-based views.
- Dynamic Type allows changes to the size of the text, which affects the size of the view.

UIStackView objects adjust the position of their contained views automatically when interface dimensions change. Alternatively, Auto Layout constraints let you specify the rules that determine the size and position of the views in your interface.

Topics

developer.apple.com

Apple Developer News Discover Design Develop Distribute Support Account

UIKit Documentation / View layout / NSLayoutConstraint Language: Swift API Changes: None

Essentials

- UIKit updates
- About App Development with UIKit
- > Protecting the User's Privacy

App structure

- > App and environment
- > Documents, data, and pasteboard
- > Resource management
- > App extensions
- > Interprocess communication
- > Mac Catalyst

User interface

- > Views and controls
- > View controllers
- > View layout

Essentials

- > UIStackView

Localization

- { } Autosizing Views for Localization in iOS

Constraints

- Positioning content within layout margins
- Positioning content relative to the safe area

> NSLayoutConstraint

- Creating constraints

Filter /

Class

NSLayoutConstraint

The relationship between two user interface objects that must be satisfied by the constraint-based layout system.

UIKit AppKit iOS 6.0+ iPadOS 6.0+ macOS 10.7+ Mac Catalyst 13.1+ tvOS 9.0+

visionOS 1.0+ Beta

iOS, iPadOS, Mac Catalyst, tvOS, visionOS

```
@MainActor
class NSLayoutConstraint : NSObject
```

macOS

```
class NSLayoutConstraint : NSObject
```

Overview

Each constraint is a linear equation with the following format:

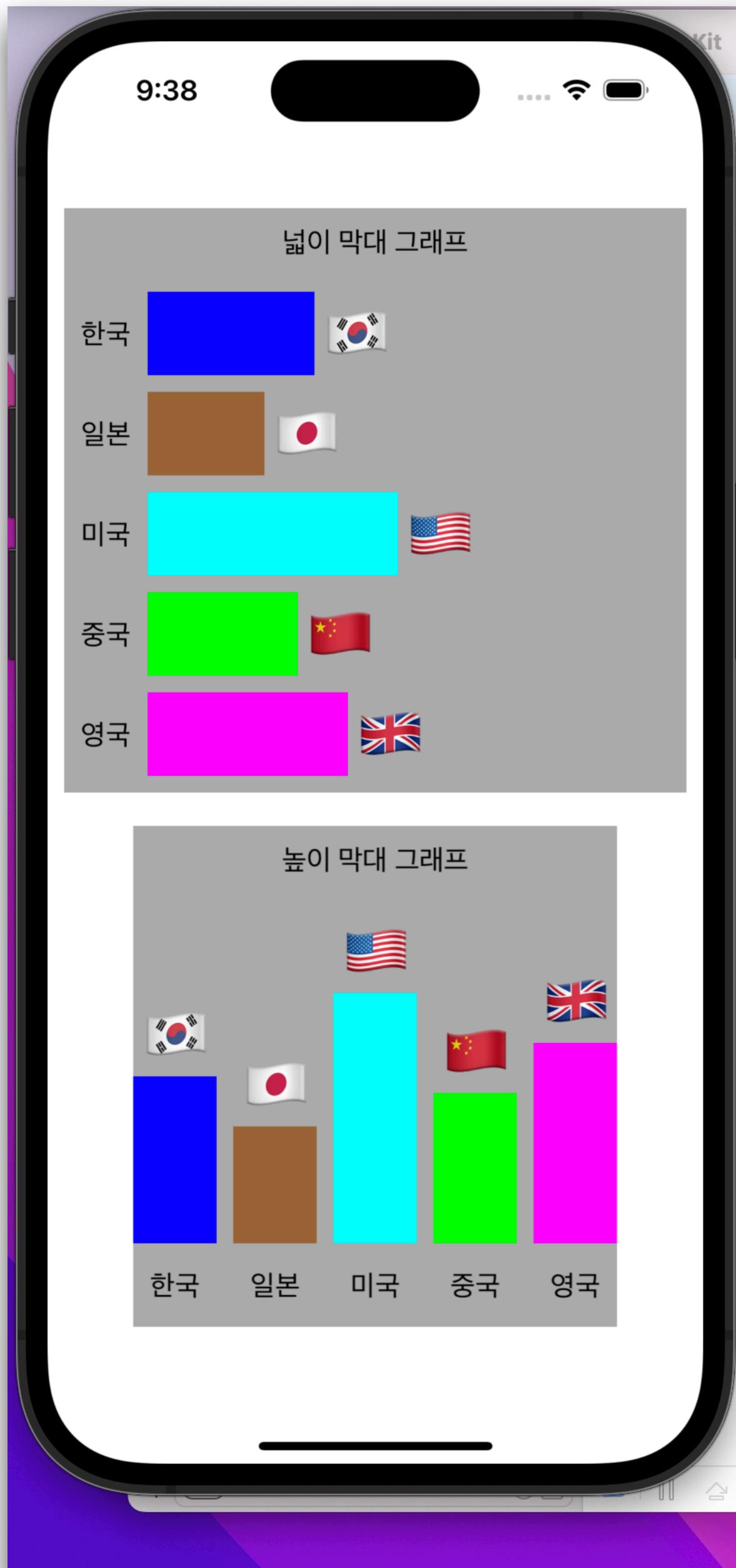
```
item1.attribute1 = multiplier × item2.attribute2 + constant
```

In this equation, attribute1 and attribute2 are the variables that Auto Layout can adjust when solving these constraints. The other values are defined when you create the constraint. For example, If you're defining the relative position of two buttons, you might say "the leading edge of the second button should be 8 points after the trailing edge of the first button." The

절대 좌표의 한계

지난주의 교훈

- 레이아웃을 쉽게 깨지게 한다.
- 미리 화면 크기를 알고 있다고 가정하기 때문이다.

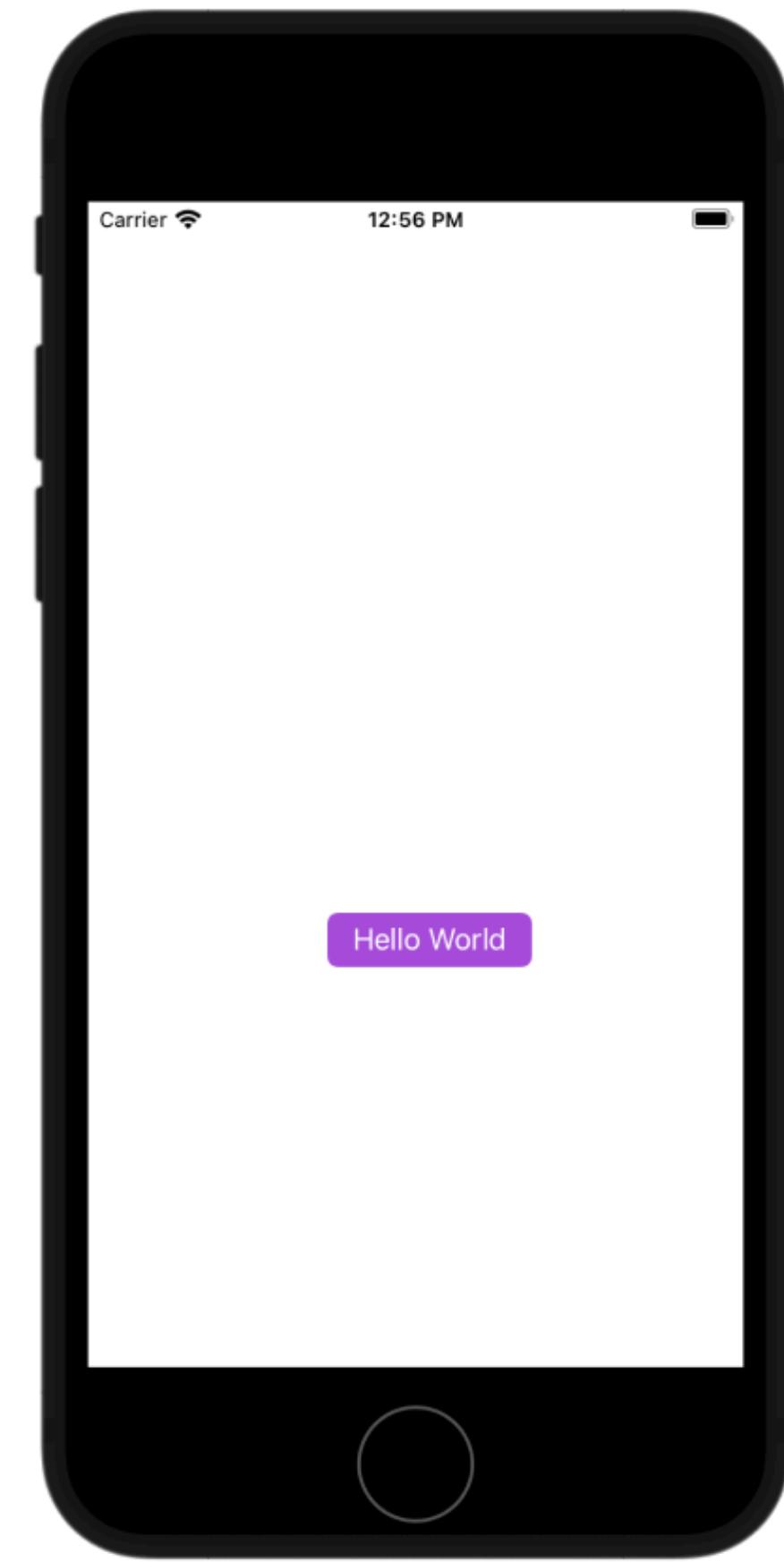


```
Kit  
HelloKit > iPhone 14 Pro  
Running H  
ViewController Main (Base)  
HelloKit > ViewController > viewDidLoad()  
s ViewController: UIViewController {  
override func viewDidLoad() {  
  
    let hGraphViewWidth = hGraphView.frame.size.width  
  
    let hTitleLabel = UILabel()  
    hTitleLabel.text = "넓이 막대 그래프"  
    hTitleLabel.textAlignment = .center  
    hTitleLabel.frame.origin = CGPoint(x: 5, y: 5)  
    hTitleLabel.frame.size = CGSize(width: hGraphViewWidth - 5 * 2, height:  
    hGraphView.addSubview(hTitleLabel)  
  
    var index: Int = 0  
  
    for graphData in graphdataArray {  
  
        let nameLabel = UILabel()  
        nameLabel.text = graphData.name  
        nameLabel.textAlignment = .center  
        nameLabel.backgroundColor = .cyan  
        nameLabel.frame.origin = CGPoint(x: 0 + index * 60, y: 250)  
        nameLabel.frame.size = CGSize(width: 50, height: 50)  
        hGraphView.addSubview(nameLabel)  
  
        let valueView = UIView()  
        valueView.backgroundColor = graphData.color  
        valueView.frame.origin = CGPoint(x: 0 + index * 60, y: 250 - value)  
        valueView.frame.size = CGSize(width: 50, height: value)  
        hGraphView.addSubview(valueView)  
  
        let flagView = UILabel()  
        flagView.text = graphData.flag  
        flagView.font = UIFont(name: "Helvetica", size: 40)  
        flagView.textAlignment = .center  
        flagView.backgroundColor = .cyan  
        flagView.frame.origin = CGPoint(x: 0 + index * 60, y: 250 - value / 2)  
        hGraphView.addSubview(flagView)  
    }  
}
```

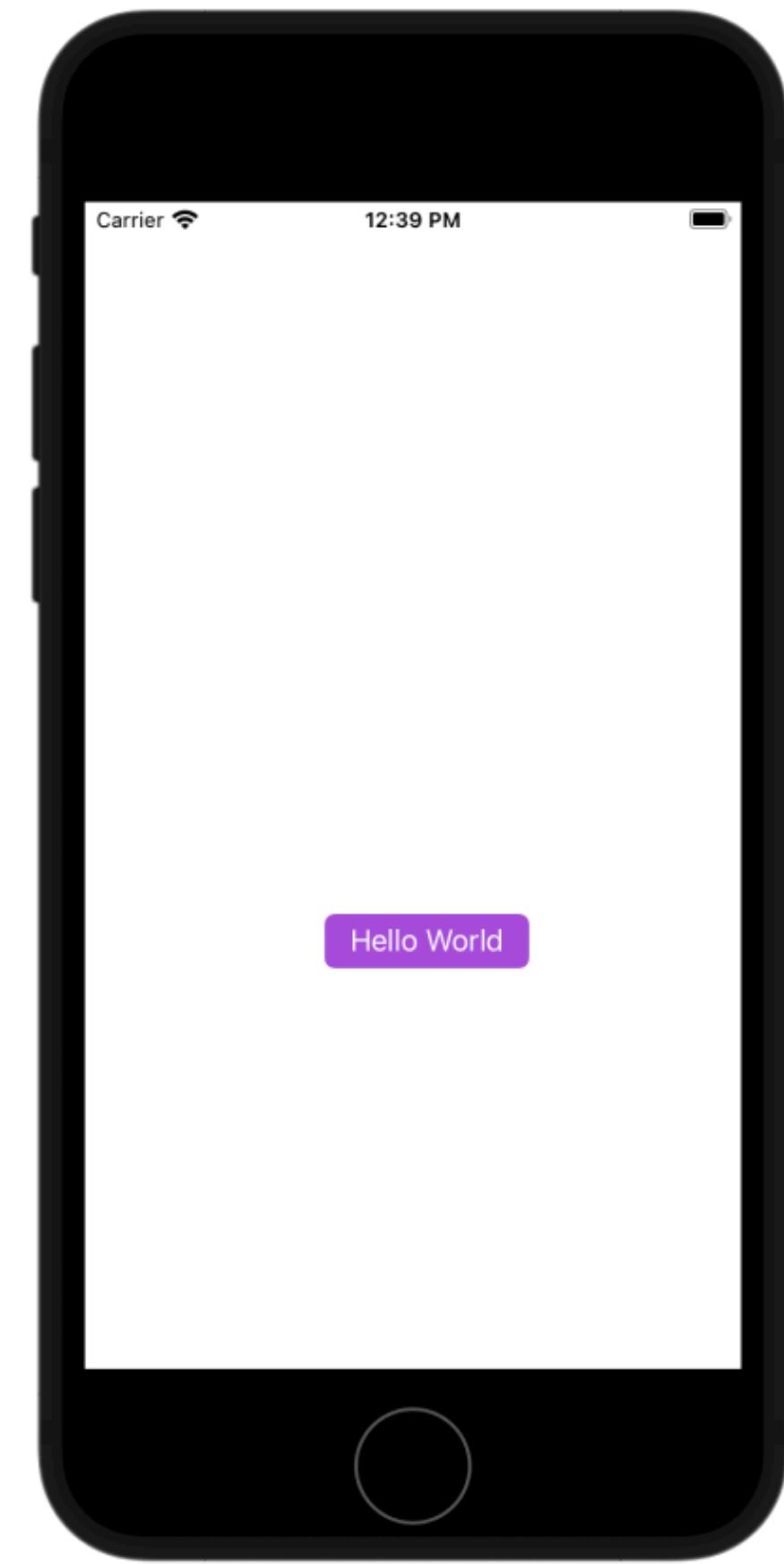
오토 레이아웃 시스템

AutoLayout

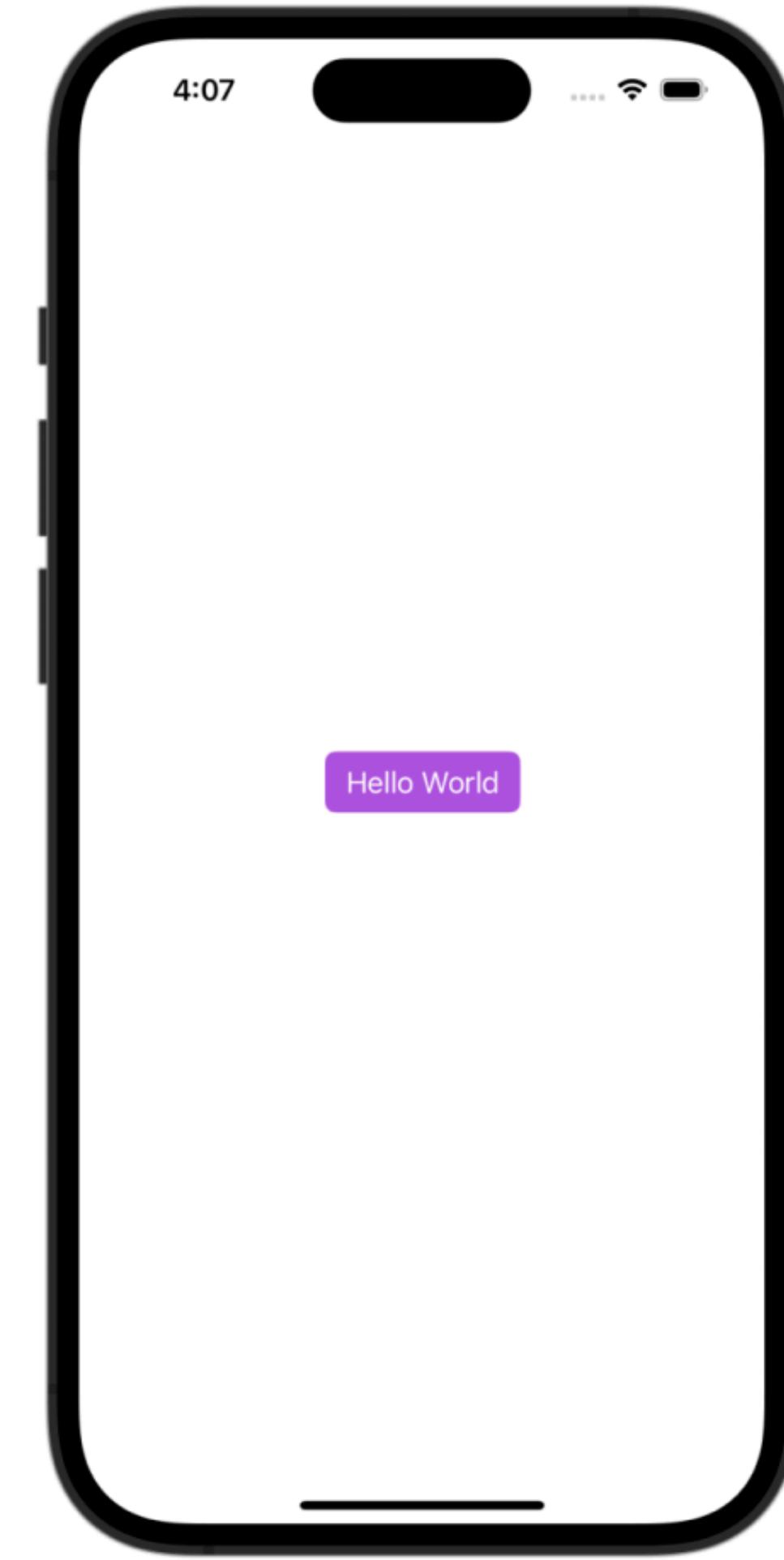
- 오토 레이아웃을 사용하여 상대적인 방식으로 뷰의 레이아웃을 기술할 수 있다.
- 이는 실행할 때 frame을 결정하기 때문에 frame의 정의가 앱이 실행 중인 기기의 화면 크기를 고려할 수 있도록 한다.



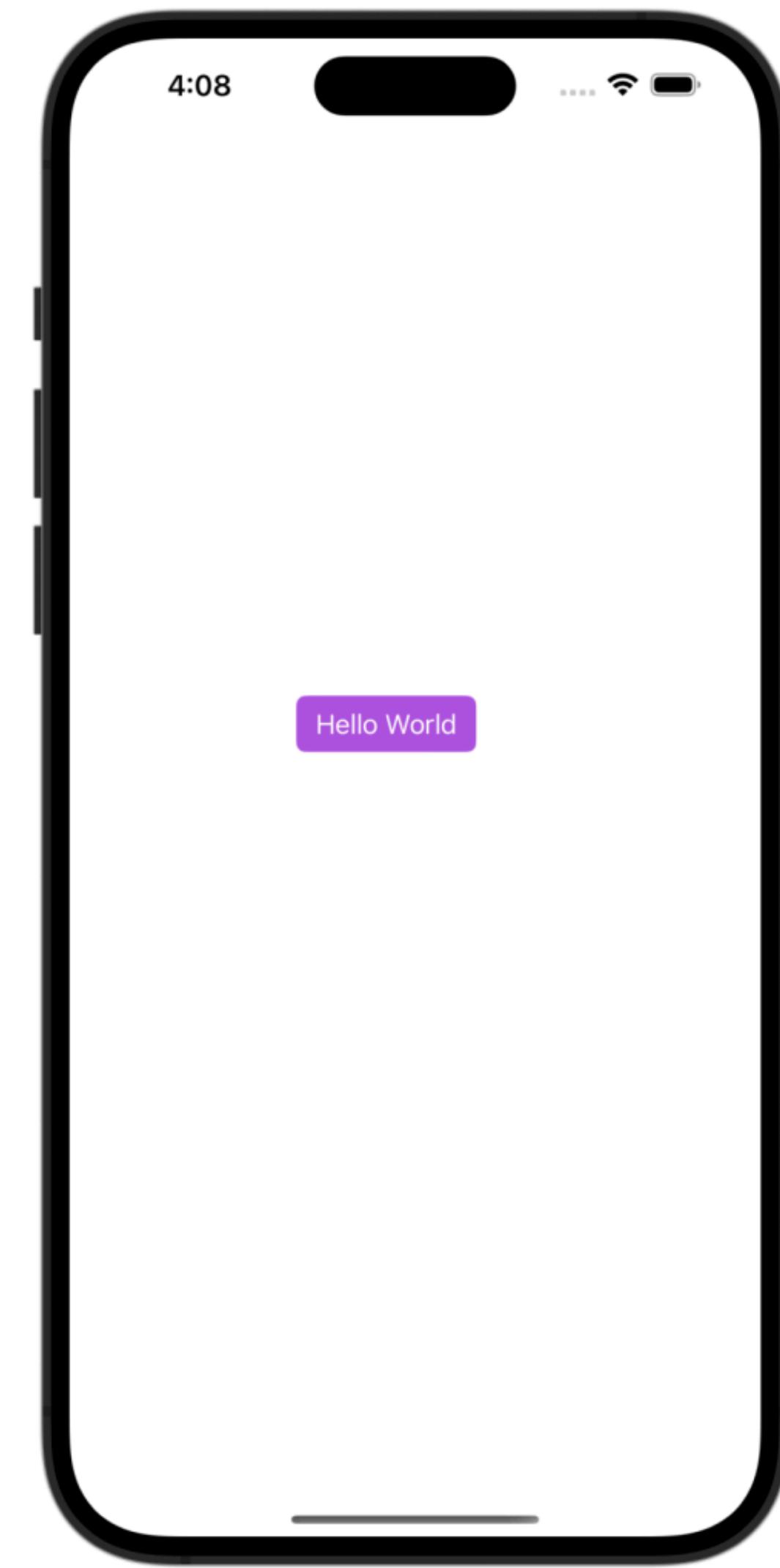
iPhone SE (4.7-inch)



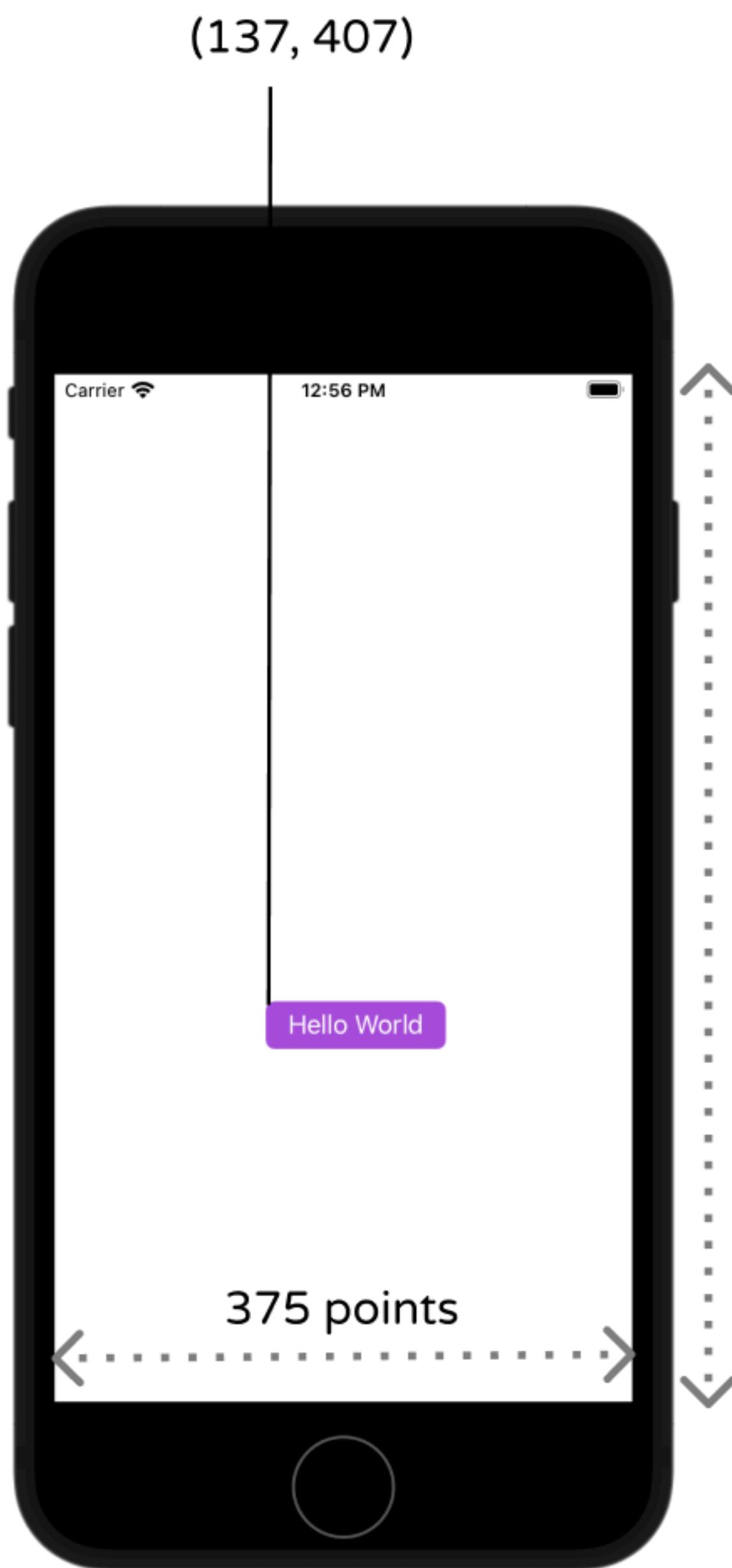
iPhone 8 (4.7-inch)



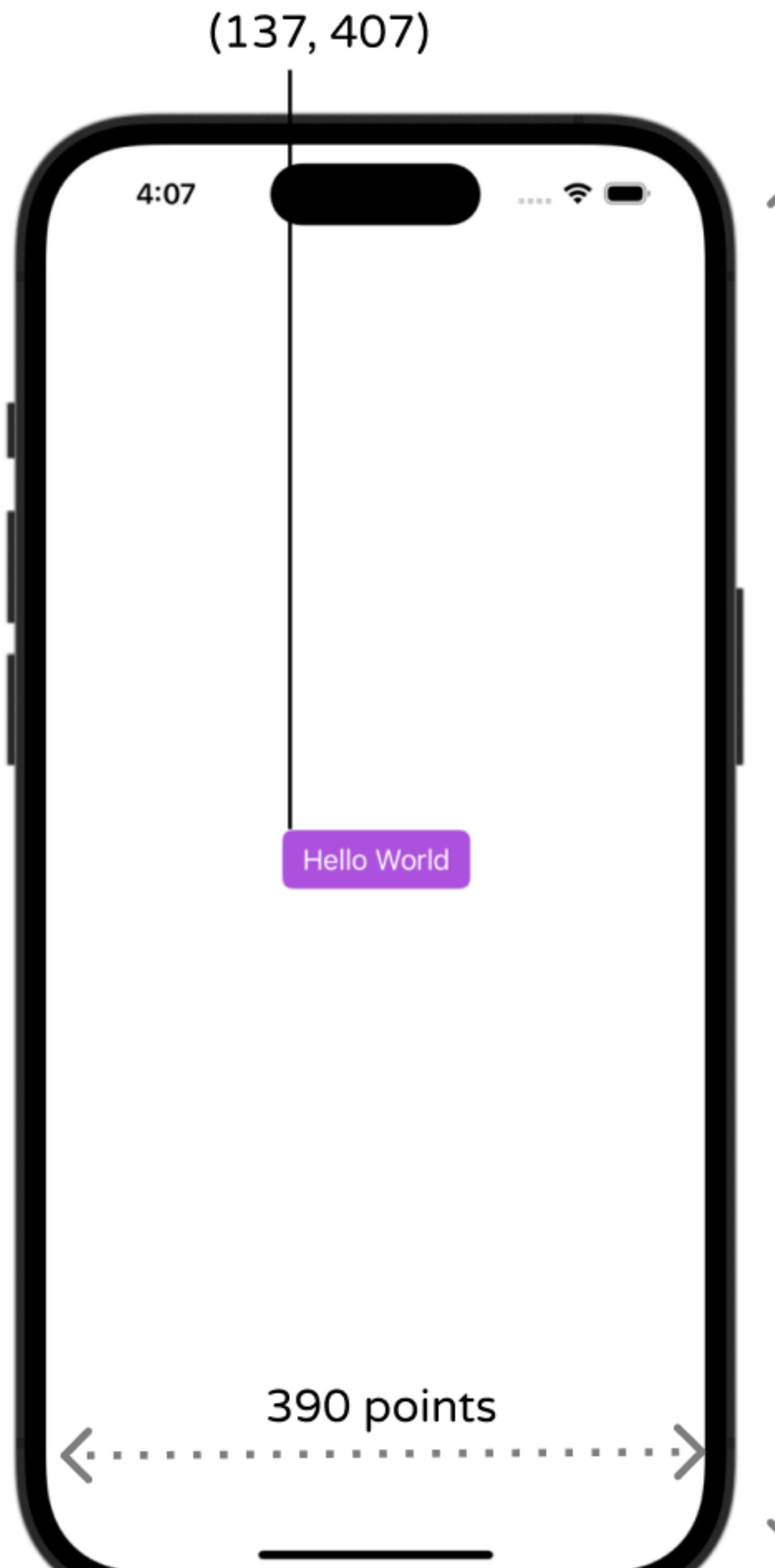
iPhone 14 Pro (6.1-inch)



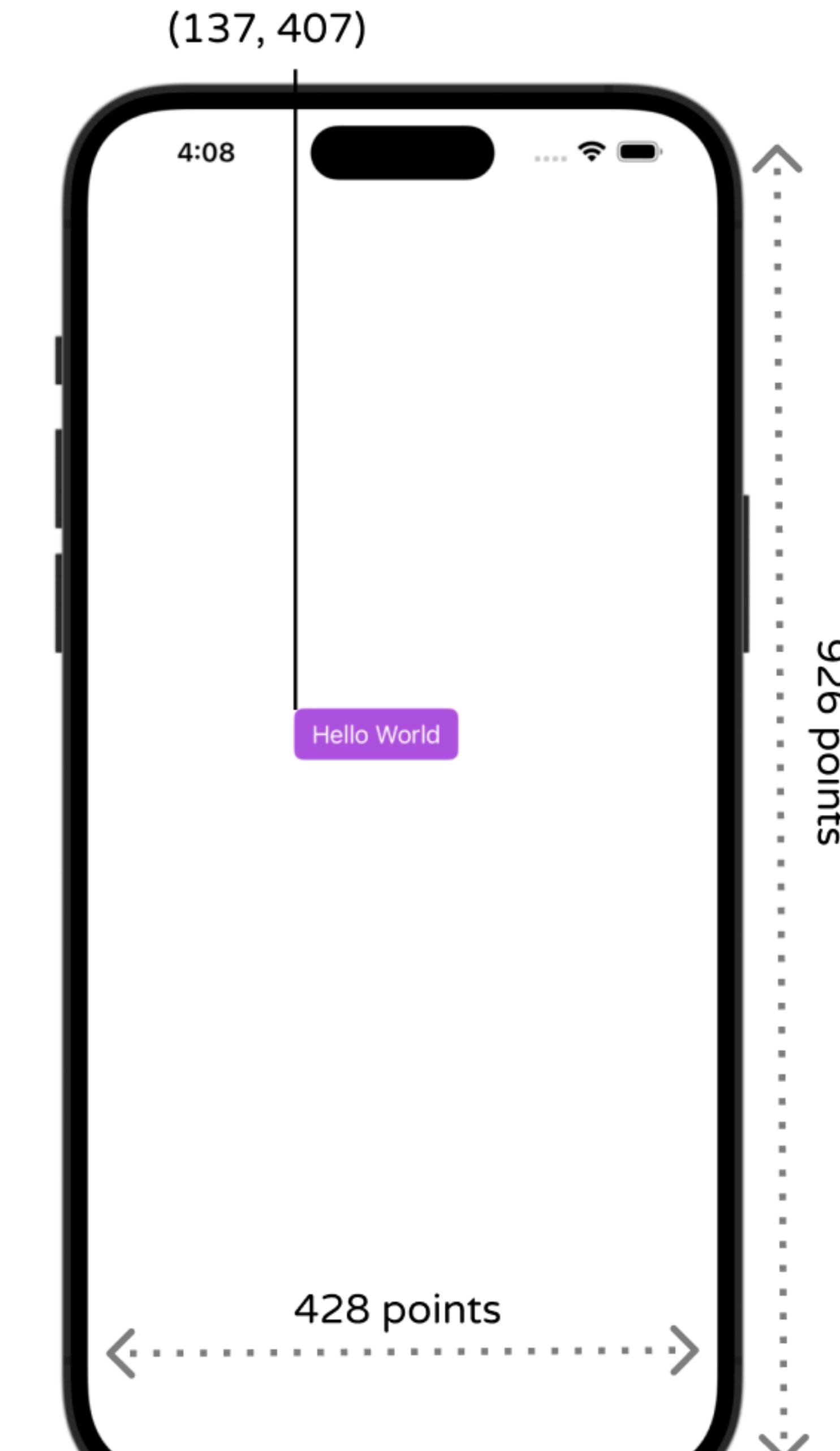
iPhone 14 Pro Max (6.7-inch)



iPhone SE (4.7-inch)



iPhone 14 Pro (6.1-inch)



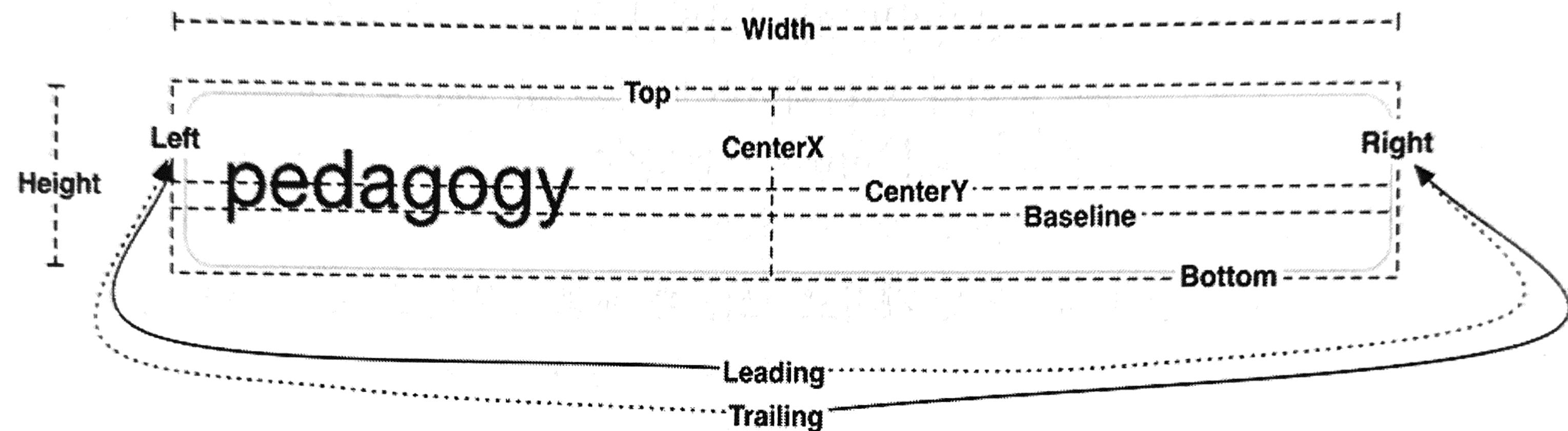
iPhone 14 Pro Max (6.7-inch)

926 points

정렬 사각형과 레이아웃 속성

AutoLayout

- 오토 레이아웃 시스템은 정렬 사각형(alignment rectangle)을 기반으로 한다.
- 이 사각형은 여러 레이아웃 속성(layout attributes)들로 정의한다.
- 다음의 그림은 뷰의 정렬 사각형을 정의하는 레이아웃 속성을 보여준다.



왼쪽에서 오른쪽으로 쓸 때

오른쪽에서 왼쪽으로 쓸 때

정렬 사각형과 레이아웃 속성

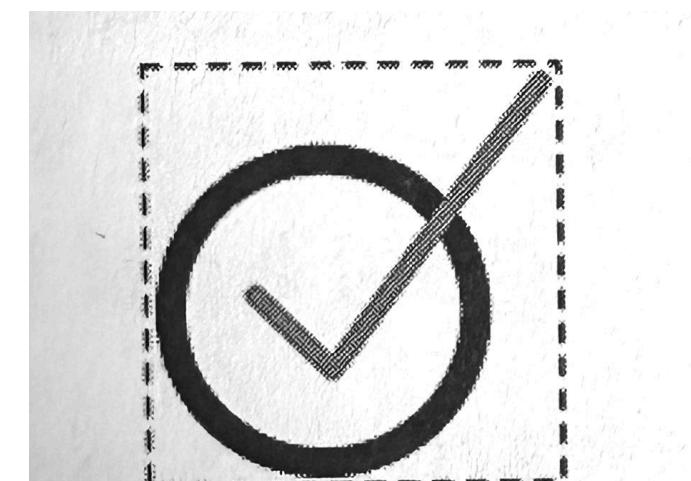
AutoLayout

Width/Height	정렬 사각형의 크기를 결정하는 값이다.
Top/Bottom/Left/Right	정렬 사각형의 가장 자리와 계층 구조에 있는 또 다른 뷰의 정렬 사각형 사이의 여백을 결정하는 값이다.
CenterX/CenterY	이 값은 대부분 bottom 속성과 같다. 하지만 모든 뷰가 그런 것은 아니다. 예를 들어 UITextField는 표시할 텍스트의 밑부분을 정렬 사각형의 bottom 대신에 baseline이 되도록 정의한다. 이것은 텍스트 필드 바로 아래 뷰 때문에 “디센더(descender)” ('g', 'p'와 같이 기준선 아래로 내려가는 글자들)가 모호해지는 것을 막기 위해서다.
Leading/Trailing	이 값들은 언어에 한정적인 속성이다. 기기의 언어가 왼쪽에서 오른쪽으로 읽는 언어로 설정되어 있다면 (예 영어) leading은 left 속성과 같고 trailing은 right 속성과 같다. 오른쪽에서 왼쪽으로 읽는 언어라면 (예 아라비아어) leading은 right 값이고 trailing은 left 값이다. 인터페이스 빌더는 left, right 값보다 leading과 trailing을 더 선호한다. 일반적으로 우리도 이를 더 사용할 것이다.

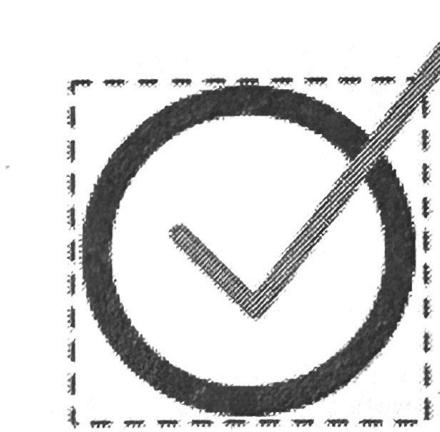
정렬 사각형과 레이아웃 속성

AutoLayout

- 기본적으로 모든 뷰는 정렬 사각형을 가지고, 모든 뷰 계층 구조는 오토 레이아웃을 사용한다.
- 정렬 사각형은 프레임과 매우 유사한다, 사실 두 사각형은 종종 같다. 하지만 프레임은 뷰 전체를 둘러싸는 반면, 정렬 사각형은 정렬 목적으로 사용하길 원하는 내용만을 감싼다.
- 뷰의 정렬 사각형을 직접 정의할 수 없다. 이를 위한 화면 크기와 같은 정보가 충분하지 않다. 대신에 우리는 제약 조건 집합을 공급한다.
- 종합하면 이 제약 조건들은 오토 레이아웃 시스템이 뷰 계층 구조에 있는 각 뷰 레이아웃 속성을 결정하고, 그 결과 정렬 사각형을 결정하도록 해준다.



프레임



정렬 사각형

제약 조건

Constraint

- 제약 조건(constraint)은 뷰 계층 구조에서 하나 이상의 뷰들의 레이아웃 속성을 결정하는데 사용되는 관계를 정의한다.
- 예를 들어 두 뷰 사이의 수직 공간은 항상 8포인트여야 한다" 또는 "이 뷰들은 항상 같은 너비를 갖는다"와 같은 제약 조건을 추가할 수 있다.
- 또한 제약 조건은 "이 뷰의 높이는 항상 44포인트여야 한다"와 같이 뷰에 고정된 크기를 지정할 수 있다.

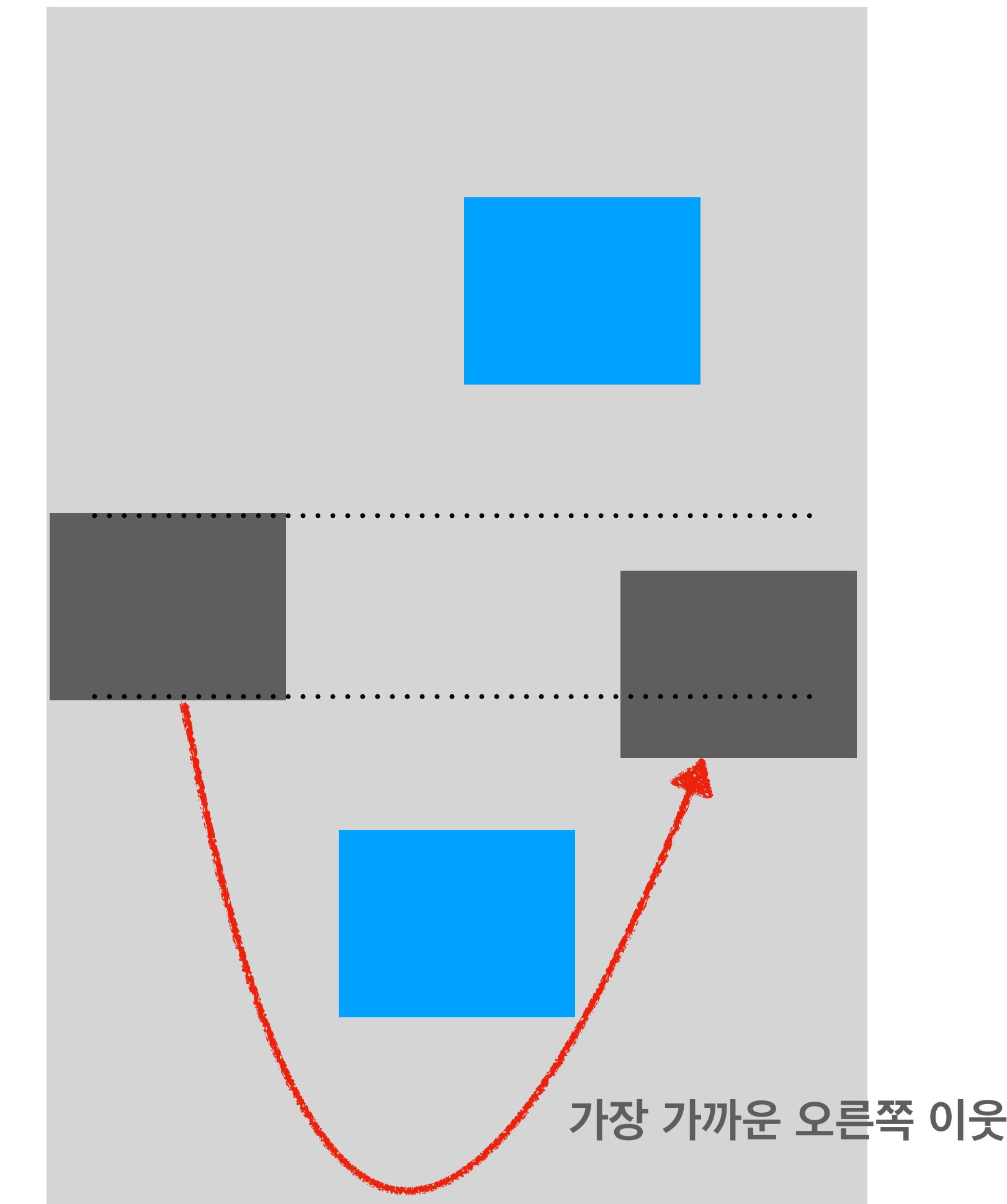
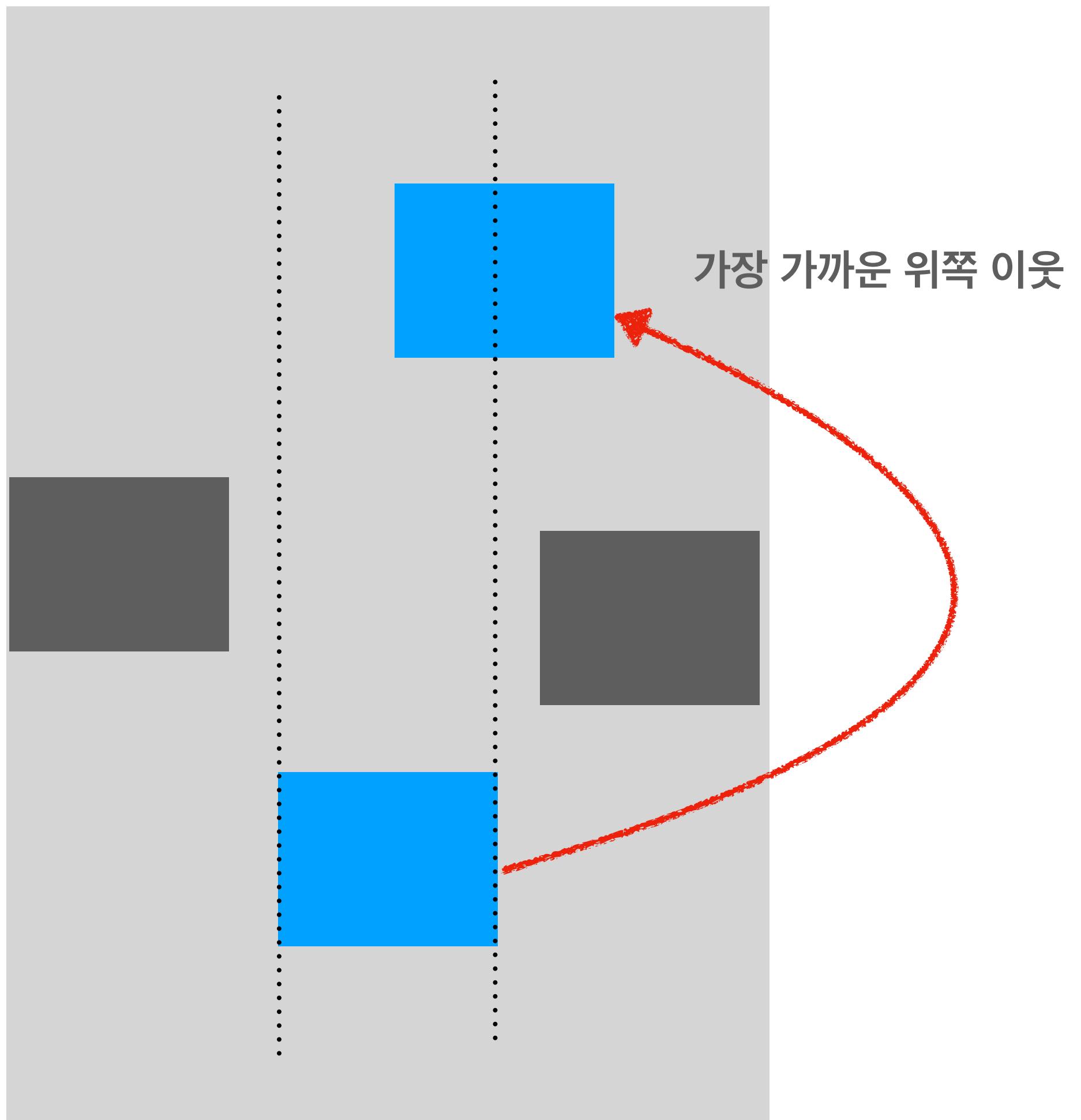
제약 조건

Constraint

- 모든 레이아웃 속성에 관한 제약 조건을 가져야 하는 것은 아니다. 어떤 값은 제약 조건에서 직접 가져올 수 있고 다른 어떤 것들은 관련 레이아웃 속성 값으로부터 계산해낼 수 있다.
- 예를 들어 뷰의 제약 조건으로 좌변과 너비를 설정하면 우변은 저절로 결정된다(좌변+ 너비=우변). 경험으로 보아 면적당 적어도 두 제약 조건(수직과 수평)이 필요하다.
- 만약 모든 제약 조건을 고려한 뒤에도 레이아웃 속성에 여전히 모호하거나 빠진 값이 있다면 오토 레이아웃 시스템에서 오류와 경고를 내보낼 것이다. 그리고 작성된 인터페이스는 모든 기기에서 기대했던 것처럼 보이지 않을 것이다.

제약 조건

Constraint



제약 조건

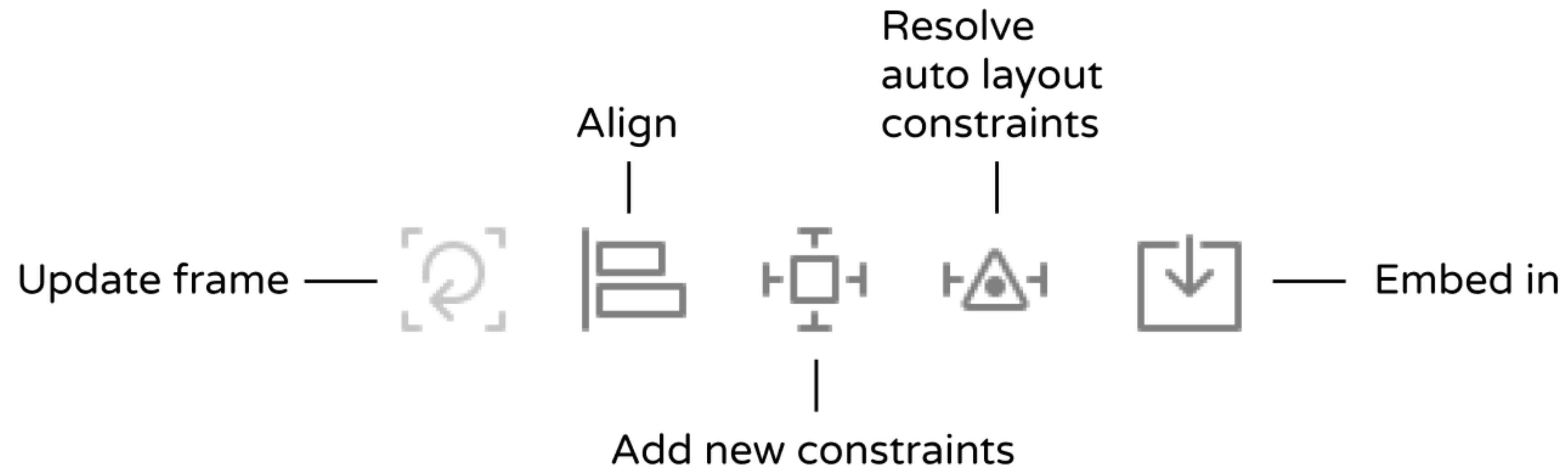
Constraint

- 만약 뷰가 특정 방향에 형제가 하나도 없다면 가장 가까운 이웃은 컨테이너로 알려진 자신의 상위 뷰다.
- 이제 라벨에 대한 제약 조건을 적을 수 있다.
 - 라벨의 윗변은 가장 가까운 이웃(라벨의 컨테이너인 Viewcontroller의 view)에서 8포인트 떨어져야 한다.
 - 라벨의 중심은 상위 뷰의 중심과 같아야 한다.
 - 라벨의 너비는 해당 폰트 크기로 렌더링된 텍스트의 너비와 같아야 한다.
 - 라벨의 높이는 해당 폰트 크기로 렌더링된 텍스트의 높이와 같아야 한다.
- 첫 번째와 네 번째 제약 조건을 생각해보면 라벨의 밑변은 따로 지정할 필요가 없다는 것을 알수 있다. 그것은 라벨의 윗변과 높이 제약 조건으로 결장된다. 마찬가지로 두 번째와 제 번째 제약 조건으로 라벨의 좌변과 우변이 정해진다.

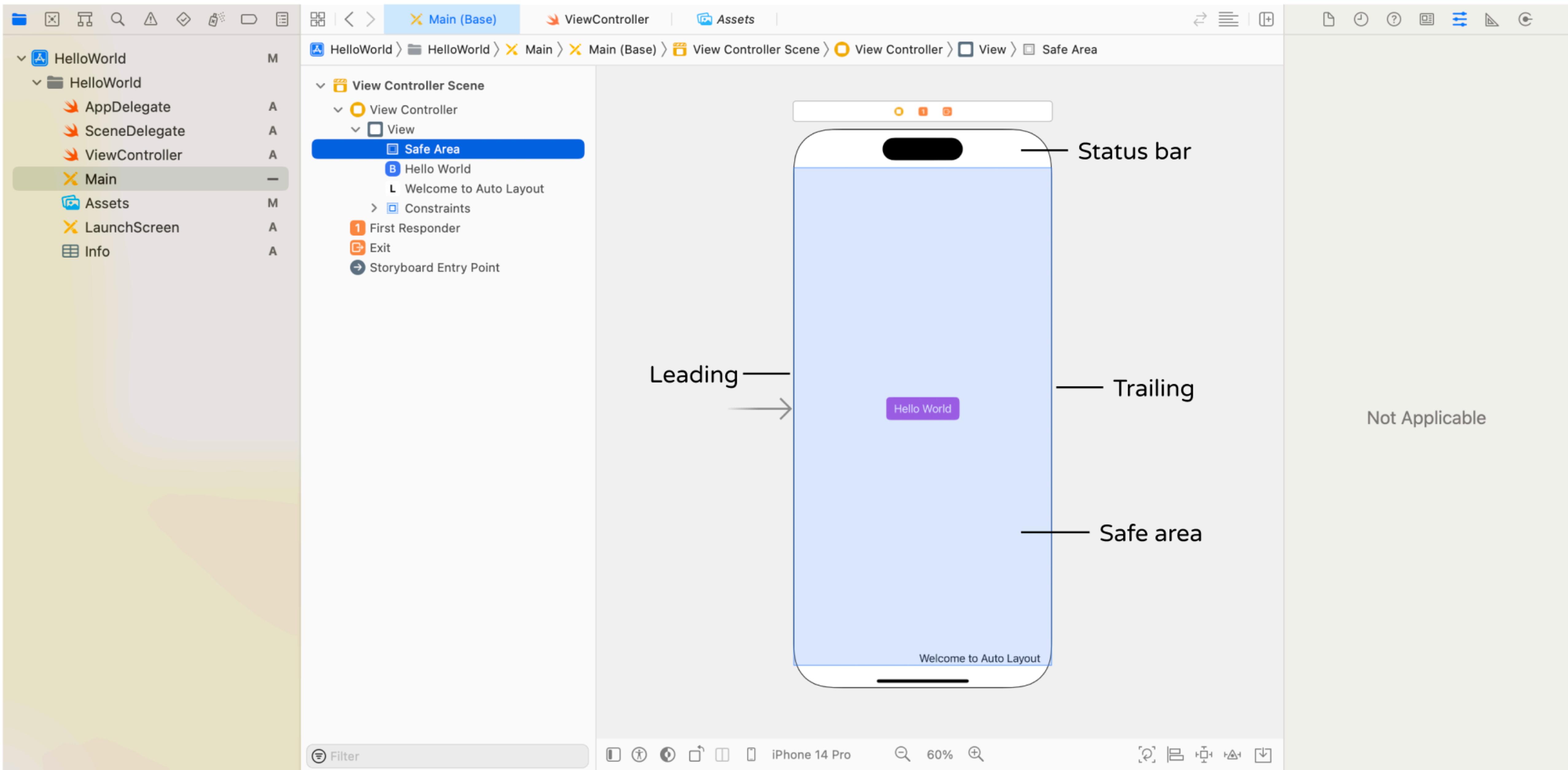
제약 조건

Constraint

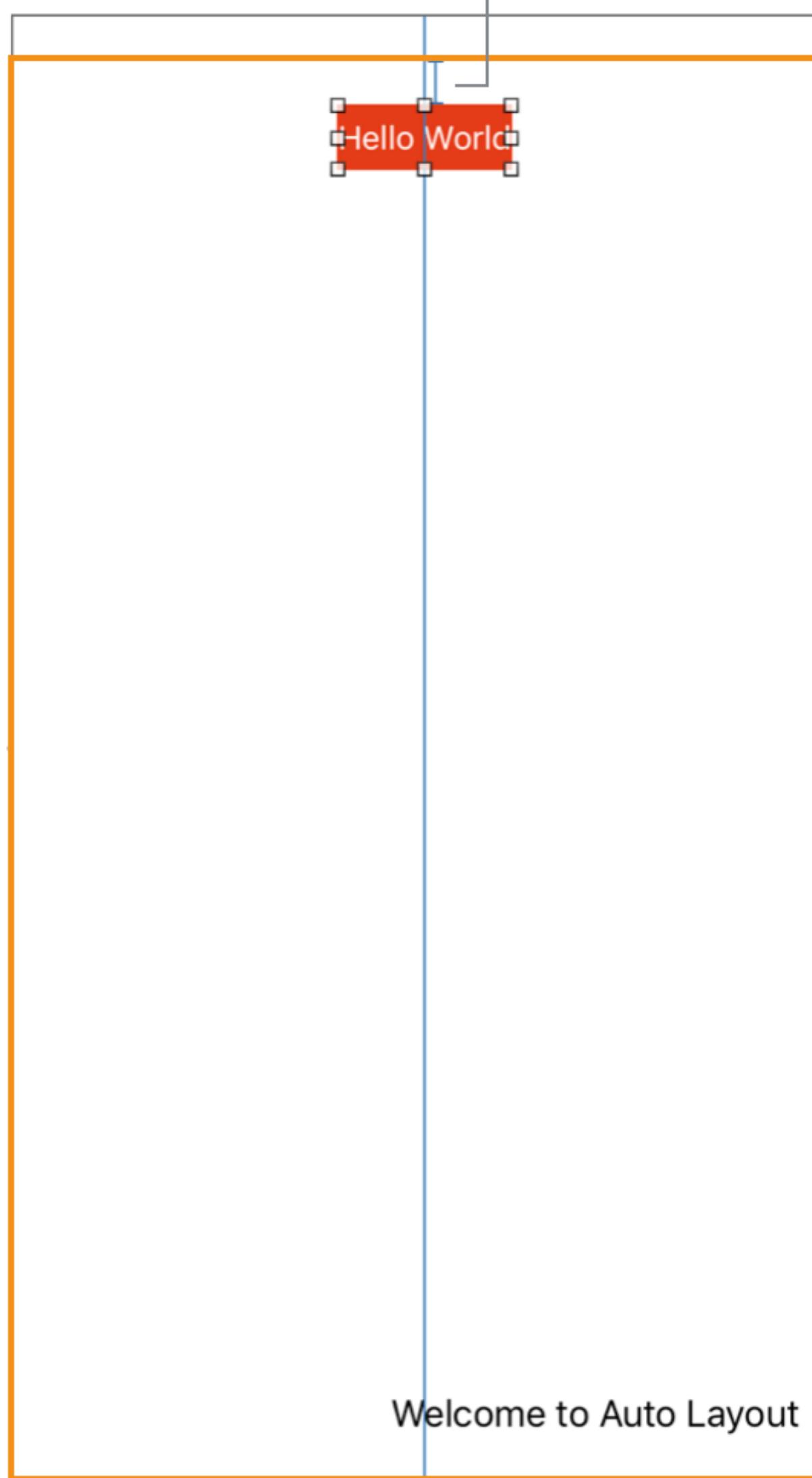
- 이제 상단 라벨을 배치하기 위해 제약 조건들을 추가할 수 있다.
- 제약 조건은 인터페이스 빌 더를 사용하거나 코드로 추가할 수 있다.
- 애플은 가능한 인터페이스 빌더를 사용해 제약 조건을 추가하는 것을 추천한다.
- 여기서도 이 방식을 사용할 것이다.
- 하지만 뷰를 프로그래밍으로 생성하고 구성한다면 제약 조건도 코드로 추가할 수 있다.



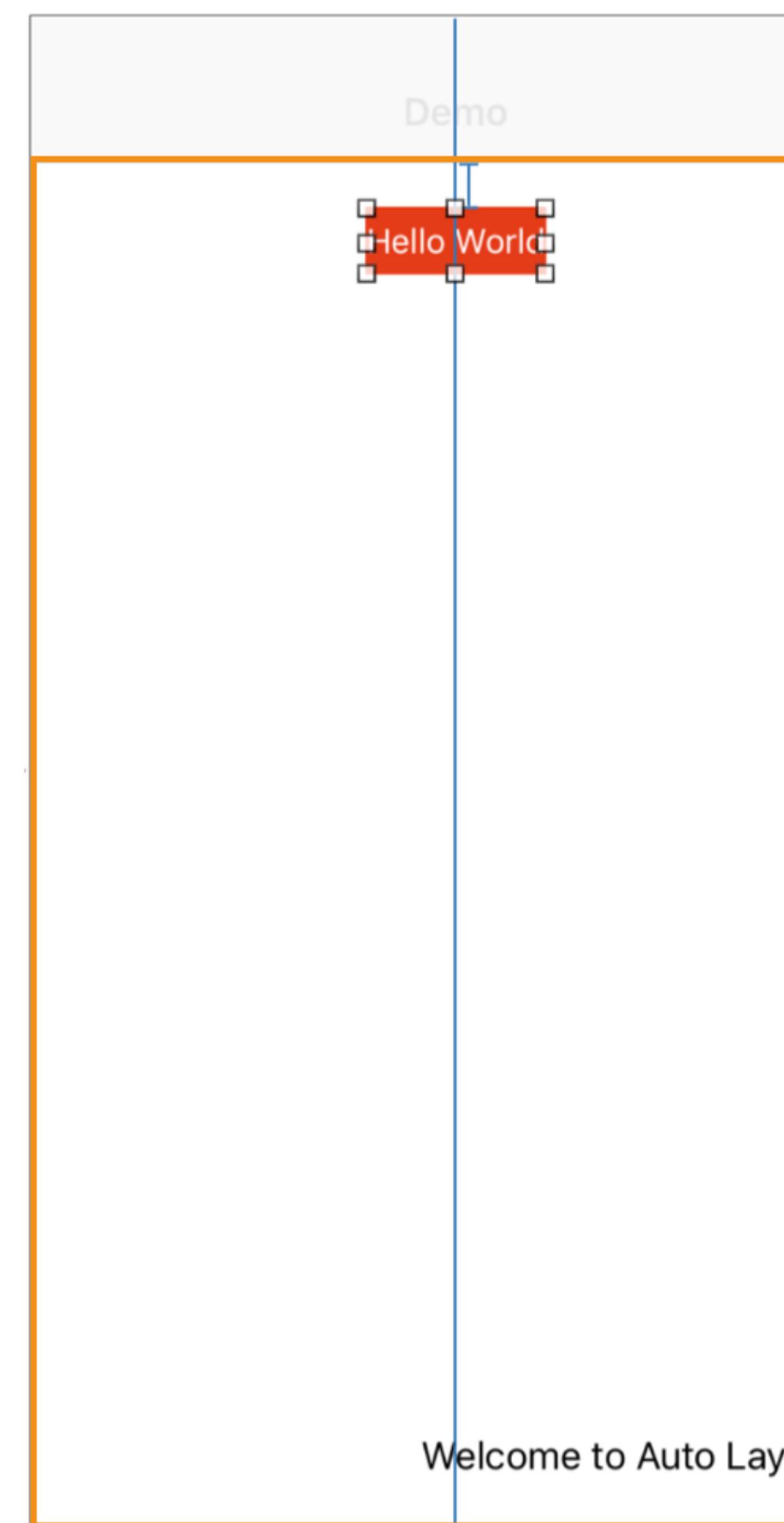
- Align 정렬
 - 두 뷰의 왼쪽 가장자리를 정렬하는 것과 같은 정렬 제약 조건을 만들기
- Add new constraints 새로운 제약 조건 추가
 - UI 컨트롤의 너비를 정의하는 것과 같은 간격 제약 조건을 만들기
- Embed in 포함시키기 – 스택 뷰(또는 다른 뷰)에 뷰를 임베드하기
- Resolve auto layout issues 자동 레이아웃 문제 해결 - 레이아웃 문제 해결.
- Update frames 프레임 업데이트
 - 주어진 레이아웃 제약 조건을 참조하여 프레임의 위치와 크기를 업데이트하기



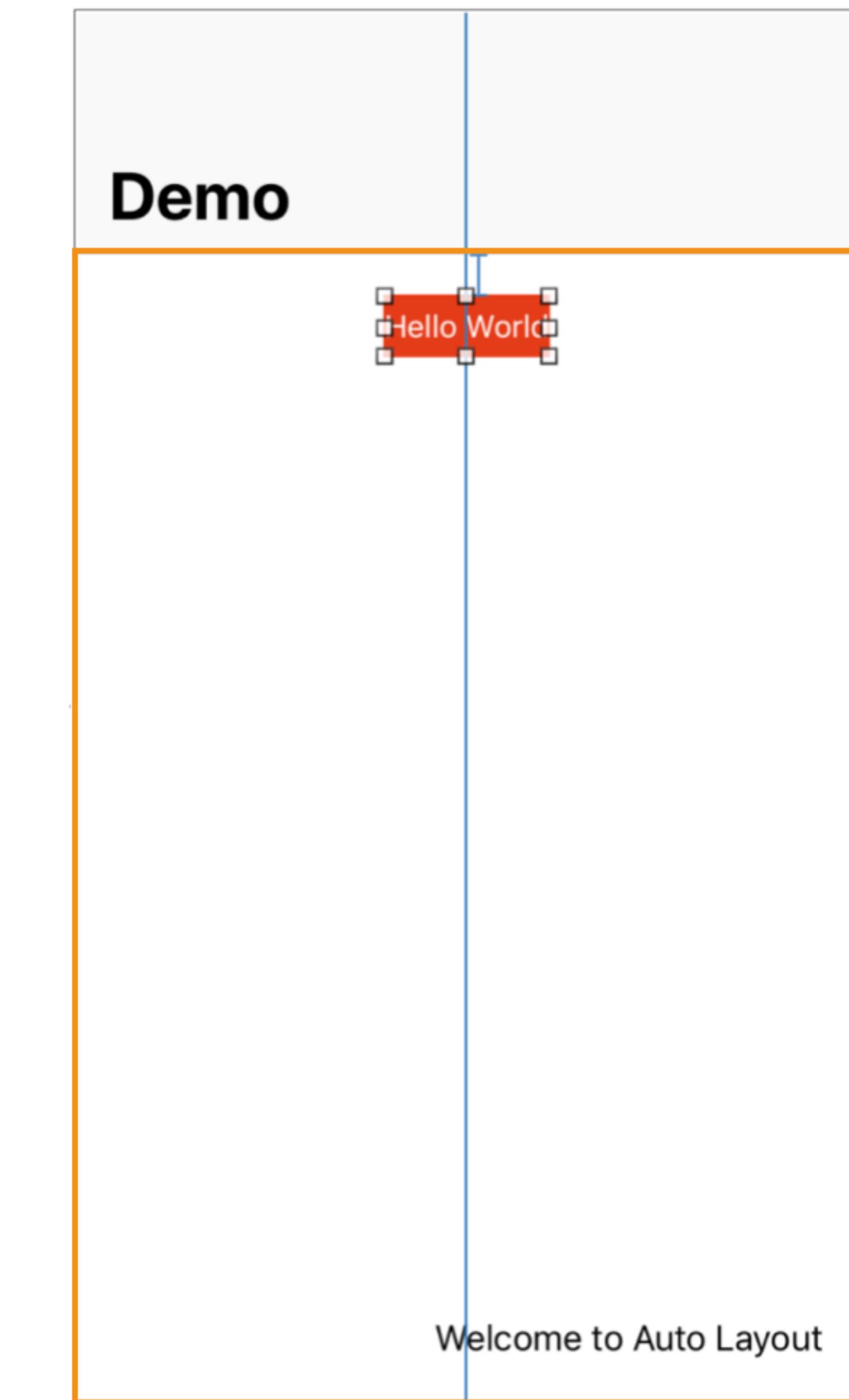
Top space to
Safe Area



Status bar



Navigation bar



Navigation bar
with large title

함께
해요

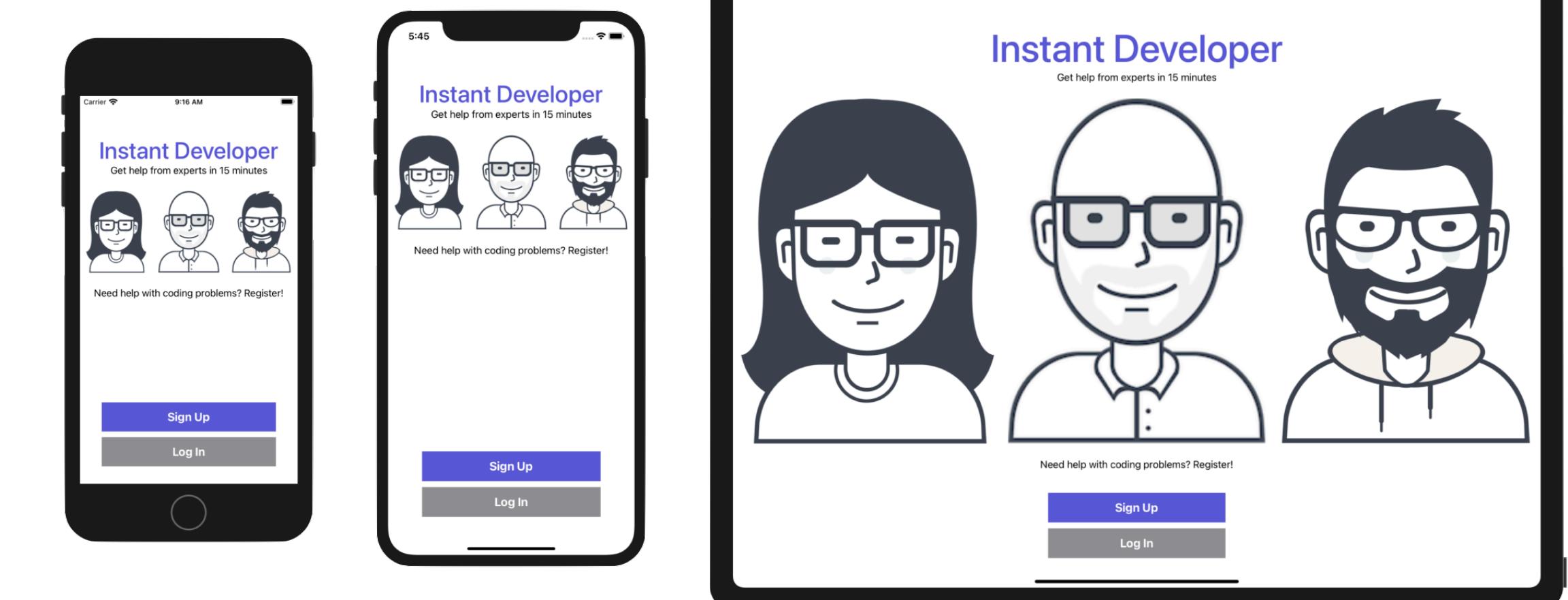
함께 해봅시다

AutoLayout

함께 해봅시다

함께
해요

- 섭씨 온도를 입력하면 화씨로 바꾸어 출력하는 앱을 만듭시다.
- 각종 아이폰과 아이패드 시뮬레이터에서 레이아웃이 잘 적용되었는지 확인해봅시다.
- 앱 타이틀과 큰 이미지 그리고 회원가입과 로그인 버튼이 있는 화면을 만듭시다 (모양만 만듭시다)
- 각종 아이폰과 아이패드 시뮬레이터에서 레이아웃이 잘 적용되었는지 확인해봅시다.
- Interface Builder로도 도전해보고, 코드로도 도전해봅시다.



참고자료

Constraint

- <https://developer.apple.com/documentation/uikit>
- https://developer.apple.com/documentation/uikit/view_layout
- <https://developer.apple.com/documentation/uikit/nslayoutconstraint/>
- <https://www.appcoda.com/learnuikit/auto-layout-intro.html>
- <https://www.appcoda.com/learnuikit/stack-views.html>
- <https://theswiftdev.com/mastering-ios-auto-layout-anchors-programmatically-from-swift/>



감사합니다