

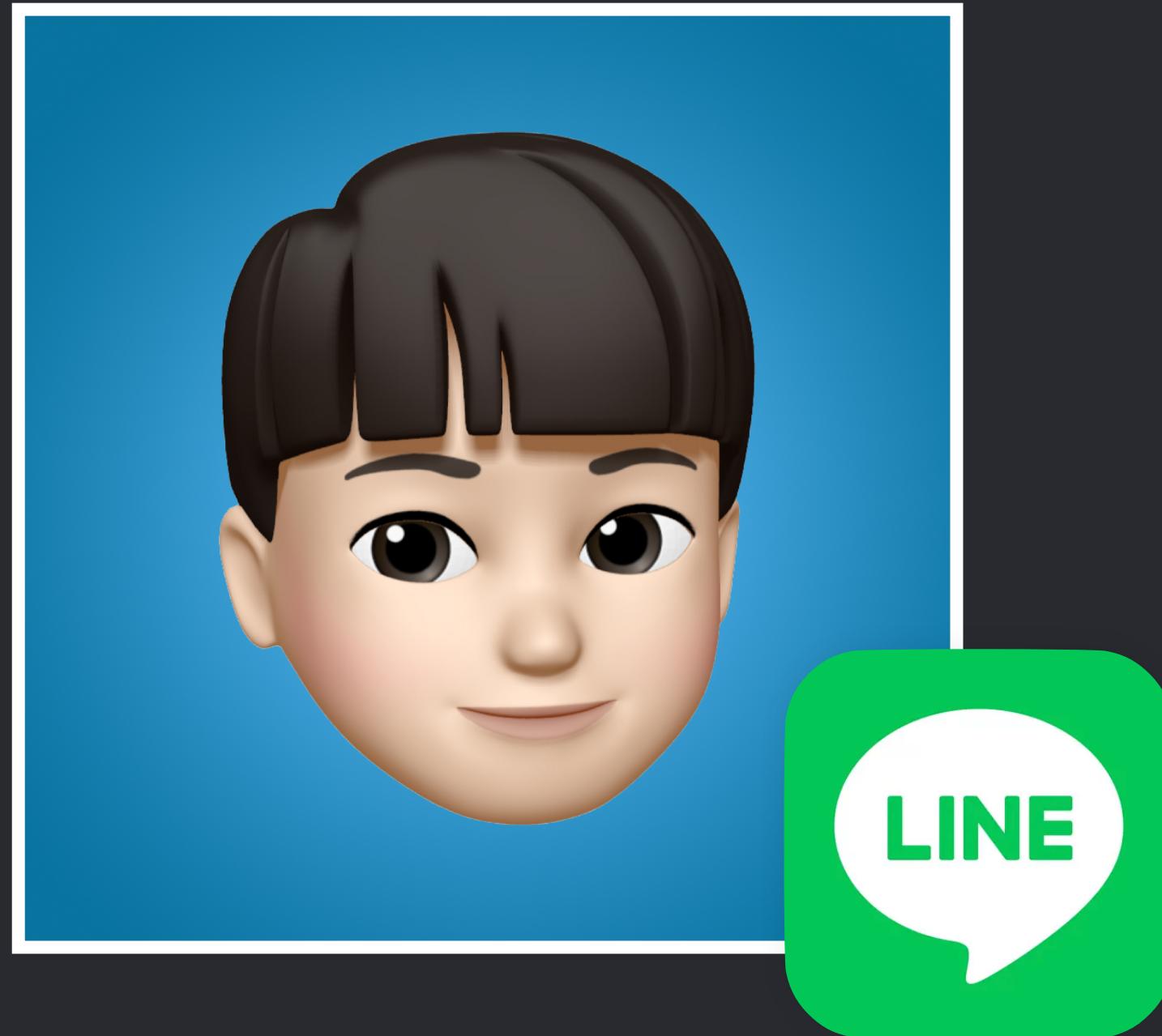
# iOS 세미나 0

2023 와플스튜디오 루키 세미나

2023.09.03, 박신홍

# 세미나장 소개

- 박신홍
  - 17학번 경영/컴공
  - LINE iOS 개발자 8개월차



# 오늘 배울 내용

- iOS 개발에 대한 개괄
- iOS 앱의 기본적인 동작 원리
- 과제에 필요한 내용
  - AutoLayout
  - Xcode로 디버깅하기
  - Storyboard 없이 앱 만드는 법
  - 다른 뷰로 넘어가는 법

# iOS 개발의 장단점

- 장점
  - 뛰어난 생태계: iOS, iPadOS, macOS, visionOS, tvOS 등등 다양한 플랫폼에서 한 가지 언어와 기술 스택으로 개발이 가능
  - OS에 내장된 감성과 디테일, 그리고 그걸 받쳐주는 하드웨어 성능
  - 상당히 낮은 수준의 OS 버전 파편화
  - 뉴진스 MZ폰 등극 → 미래가 밝다
- 단점
  - 애플이 새롭게 내놓는 기기나 기술에 대해 지속적으로 공부해야 함
  - OS 소스 코드 비공개 → 원인을 알 수 없는 OS 버그에 마주하게 될 때 조금은 화가 난다

# iOS 개발 개괄

## 준비물

- Xcode 14.3.1

- 앱스토어는 너무 느림
- <https://xcodes.app/>에서 설치 권장
- 일반적인 IDE와는 달리, Xcode 버전에 따라 사용 가능한 Swift 버전과 SDK 버전이 모두 다름 (IDE에 내장)
- 버전 14.3.1에서 잘 되던게 14.2.1에서는 빌드가 안될수도 있음

### Minimum requirements and supported SDKs

Xcode Version	Minimum OS Required	SDK	Architecture	Deployment Targets	Simulator	Swift
Xcode 15	macOS Ventura 13.4	iOS 17 macOS 14 tvOS 17 watchOS 10 DriverKit 23	i386 x86_64 armv7k arm64 arm64e arm64_32	iOS 12-17 iPadOS 13-17 macOS 10.13-14 tvOS 12-17 watchOS 4-10 DriverKit 19-23	iOS 14.0.1-17 tvOS 14-17 watchOS 7-10	Swift 4 Swift 4.2 Swift 5.9
Xcode 14.3.1	macOS Ventura 13	iOS 16.4 macOS 13.3 tvOS 16.4 watchOS 9.4 DriverKit 22.4	i386 x86_64 armv7k arm64 arm64e arm64_32	iOS 11-16.4 iPadOS 13-16.4 macOS 10.13-13.3 tvOS 11-16.4 watchOS 4-9.4 DriverKit 19-22.4	iOS 13.7-16.4 tvOS 13.4-16.4 watchOS 7-9.4	Swift 4 Swift 4.2 Swift 5.8.1
Xcode 14.3*	macOS Ventura 13	iOS 16.4 macOS 13.3 tvOS 16.4 watchOS 9.4 DriverKit 22.4	i386 x86_64 armv7k arm64 arm64e arm64_32	iOS 11-16.4 iPadOS 13-16.4 macOS 10.13-13.3 tvOS 11-16.4 watchOS 4-9.4 DriverKit 19-22.4	iOS 13.7-16.4 tvOS 13.4-16.4 watchOS 7-9.4	Swift 4 Swift 4.2 Swift 5.8
Xcode 14.2	macOS Monterey 12.5	iOS 16.2 macOS 13.1 tvOS 16.1 watchOS 9.1 DriverKit 22.2	i386 x86_64 armv7k arm64 arm64e arm64_32	iOS 11-16.2 iPadOS 13-16.2 macOS 10.13-13.1 tvOS 11-16.1 watchOS 4-9.1 DriverKit 19-22.2	iOS 12.4-16.2 tvOS 12.4-16.1 watchOS 7-9.1	Swift 4 Swift 4.2 Swift 5.7
Xcode 14.1	macOS Monterey 12.5	iOS 16.1 macOS 13 tvOS 16.1 watchOS 9.1 DriverKit 22.1	i386 x86_64 armv7k arm64 arm64e arm64_32	iOS 11-16.1 iPadOS 13-16.1 macOS 10.13-13 tvOS 11-16.1 watchOS 4-9.1 DriverKit 19-22.1	iOS 12.4-16.1 tvOS 12.4-16.1 watchOS 7-9.1	Swift 4 Swift 4.2 Swift 5.7
Xcode 14.0.x	macOS Monterey 12.5	iOS 16 macOS 12.3 tvOS 16 watchOS 9 DriverKit 22	i386 x86_64 armv7k arm64 arm64e arm64_32	iOS 11-16 iPadOS 13-16 macOS 10.13-12.5 tvOS 11-16 watchOS 4-9 DriverKit 19-22	iOS 12.4-16 tvOS 12.4-16 watchOS 7-9	Swift 4 Swift 4.2 Swift 5.7

# iOS 개발 개괄

## 레퍼런스

- [필독] Swift 공식 문서 (한글 번역본)
  - 기초적인 내용이 쉽게 잘 정리되어 있음
- WWDC 세션
  - 애플 개발자 연례행사
  - 신기술, 응용법 등 실제 용례에 대한 설명이 많음
  - 영상 퀄리티 굿
- 애플 개발자 사이트
  - iOS 개발자들의 성경
- Human Interface Guideline
  - UI/UX 디자이너들의 교과서

# iOS 개발 개괄

## 레퍼런스

- [고급] Swift Evolution
  - Swift는 커뮤니티에 의해 점진적으로 발전해나가는 언어임
  - Swift의 여러 Language Features에 대한 구체적인 디자인, 결정 배경, 예제, 예외 케이스까지 모두 상세하게 작성한 프로포절의 모음집
  - Swift를 더 깊이 공부하고 싶다면 참고하면 좋음

# 들어가기 전에

- 6회라는 제한된 시간 내에 iOS 개발에 필요한 모든 내용을 하나부터 열까지 설명하는 것은 불가능
- 세미나는 큰 그림을 그리고, 앞으로 어떤 방향으로 공부하면 좋을지 안내하는 가이드로 삼아주세요.
- 세미나에서 설명하지 않은 내용이 과제에 등장해도 놀라지 마세요. 구글링 하면 됩니다. 😊

# 깨부하는 법

1. 애플 공식 문서 + Swift 공식 문서는 필수
2. 그래도 해결이 안되면 구글링 / StackOverflow / Kodeco / 각종 블로그
  - 구글링할 때는 항상 신빙성 있는 저자의 기록인지 확인 필요 + 작성일자 확인 필수
3. 책을 사서 보는 것도 추천... 한다고 하네요.
4. 적당히 시도해봤는데(30분 이상) 잘 안되면 Github Discussion에 질문 올리기

# iOS 앱의 작동 원리

: iOS는 어떻게 내가 작성한 코드를 실행하는가

# UIKit?

<https://developer.apple.com/documentation/uikit>

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.

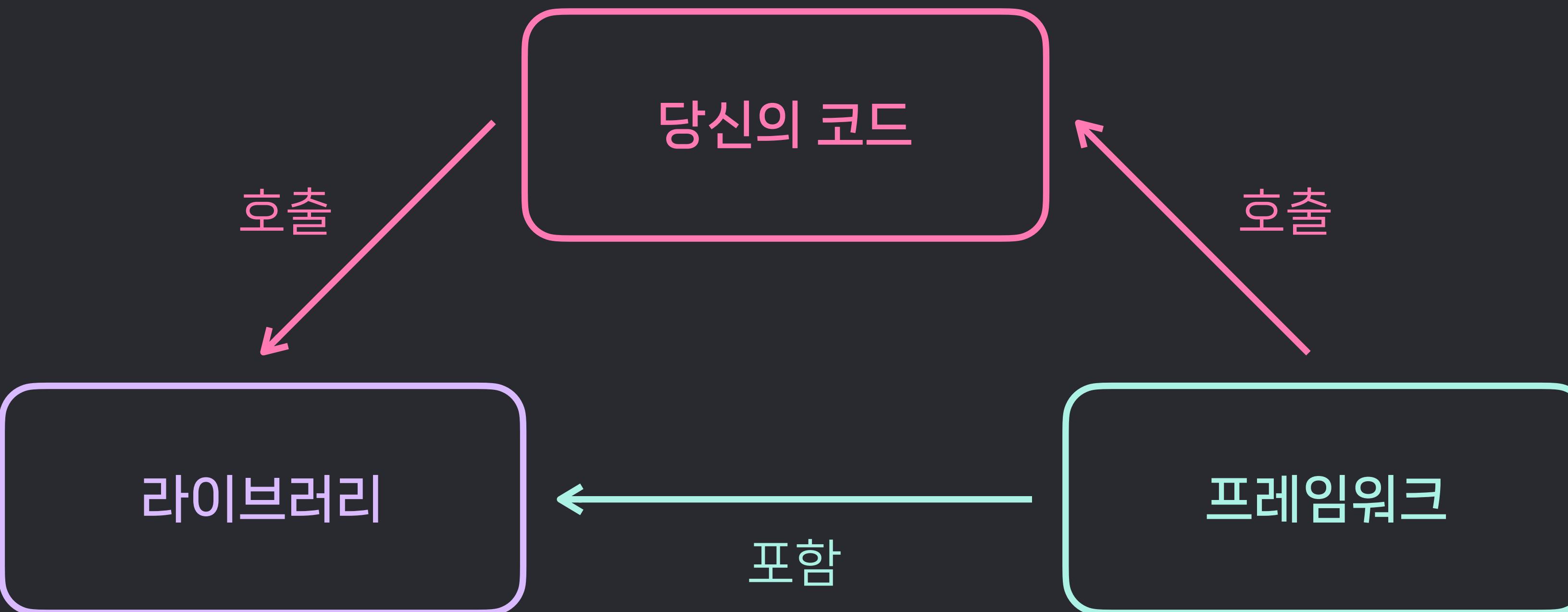
# UIKit?

<https://developer.apple.com/documentation/uikit>

- 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.
  - UI 그리기
  - 멀티터치 등 이벤트 핸들링
  - 런 루프 (?)

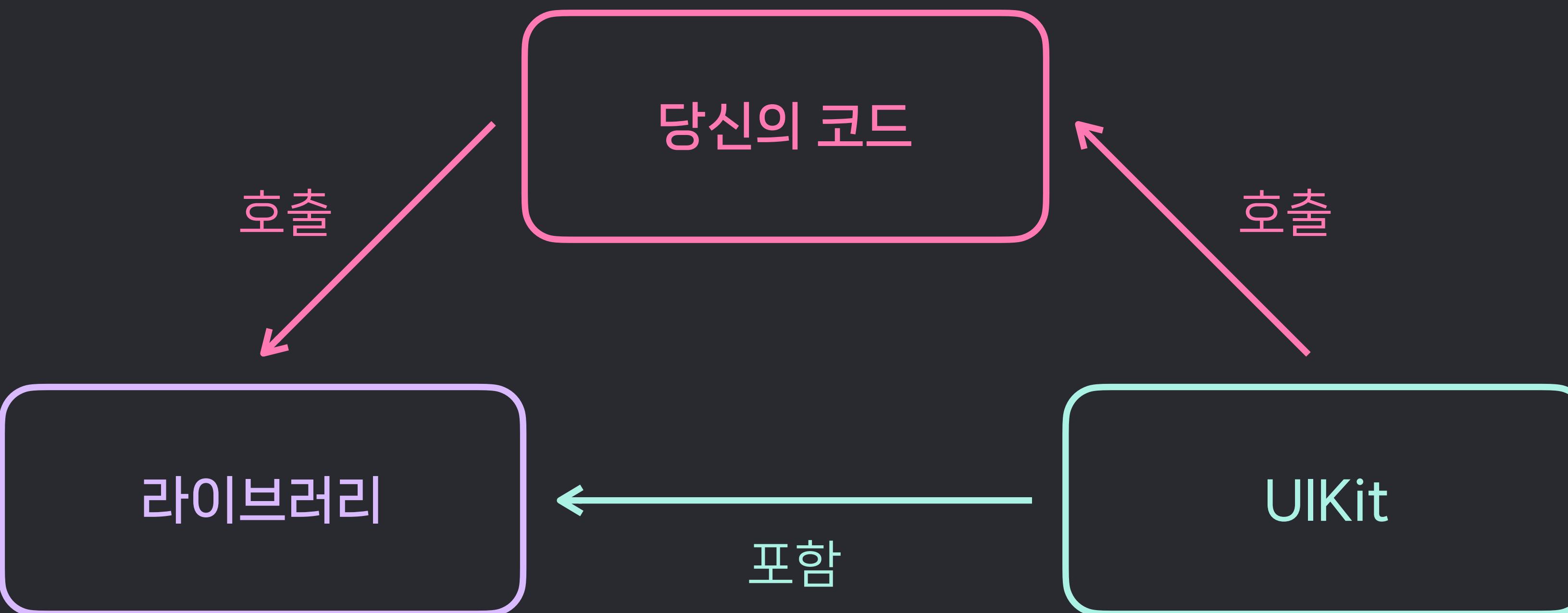
# 프레임워크?

- 라이브러리 vs 프레임워크



# 프레임워크?

- 라이브러리 vs 프레임워크





- 새로운 iOS 프로젝트 만들어보기
- 프로젝트 구성 요소 살펴보기
- Hello World 출력하기

The screenshot shows the Xcode interface with the following details:

- Project Navigator:** Shows the project structure under "seminar0". The "ViewController" file is selected.
- Editor:** Displays the "ViewController.swift" file content. The code is as follows:

```
5 // Created by user on 2023/09/02.  
6 //  
7  
8 import UIKit  
9  
10 class ViewController: UIViewController {  
11  
12     lazy var helloWorldLabel = {  
13         let label = UILabel()  
14         label.text = "Hello world"  
15         return label  
16     }()  
17  
18     override func viewDidLoad() {  
19         super.viewDidLoad()  
20         // Do any additional setup after loading the view.  
21  
22         view.addSubview(helloWorldLabel)  
23         helloWorldLabel.frame = view.frame  
24     }  
25 }  
26  
27 |
```

The code defines a `ViewController` class that creates a `UILabel` with the text "Hello world" and adds it to the view. The file was created on 2023/09/02.

# 순식간에 생겨나는 수많은 의문들..

## 하나하나 확인해봅시다

- 앱의 첫 시작점은 어디일까?
- AppDelegate, SceneDelegate?
- UIViewController?
- viewDidLoad?
- ...

# 프로그램의 Entrypoint

C, Java

```
#include <stdio.h>

int main() {
    printf("Hello, World!\n");
    return 0;
}
```

```
public class Main {
    public static void main(String[] args) {
        System.out.println("Hello, World!");
    }
}
```

# 프로그램의 Entrypoint

## Swift

```
func main() {  
    _ = UIApplicationMain(  
        CommandLine.argv,  
        CommandLine.unsafeArgv,  
        NSStringFromClass(UIApplication.self),  
        NSStringFromClass(AppDelegate.self)  
    )  
}  
  
main()
```

main.swift



# 프로그램의 Entrypoint

## Swift

```
import UIKit

@main
class AppDelegate: UIResponder, UIApplicationDelegate {

    // Other delegate methods...

}
```



@main 만 달아주면 컴파일러와 프레임워크가 알아서  
main 함수를 합성(synthesize) 해준다.

# Delegate 패턴

= 대리인 패턴



## dele·gate

1. 대표 2. 위임하다 3. 뽑다

발음 동사 [ 'delɪgət ]

한줄요약: "내가 해야 할 일을 대리인 시키기"

→ AppDelegate = App의 대리인(Delegate)

# AppDelegate

```
import UIKit

@main
class AppDelegate: UIResponder, UIApplicationDelegate {

    func application(
        _ application: UIApplication,
        didFinishLaunchingWithOptions launchOptions: [UIApplication.LaunchOptionsKey: Any]?) -> Bool {
        // Override point for customization after application launch.
        return true
    }

}
```

앱이 실행이 완료되었을 때 어떤 작업이 수행되어야 하는지, UIKit 프레임워크는 알 수 없다. 따라서 대리인이 대신하도록 하는 수밖에 없다. iOS 앱 개발자가 하는 일은, 이 대리인이 해야 할 일을 정의하는 것이다.

✓ **C** `UITextFormattingCoordinator`

### Applying Updated Text Attributes

**P** `var delegate: UITextFormattingCoordinatorDelegate?`

> **Pr** `UITextFormattingCoordinatorDelegate`

### Font Picker

✓ **C** `UIFontPickerController`

### Responding to font picker interactions

**P** `var delegate: UIFontPickerControllerDelegate?`

> **Pr** `UIFontPickerControllerDelegate`

> **Pr** `UIFontPickerControllerDelegate`

✓  `TextKit`

✓ **C** `NSTextContentManager`

### Customizing and Validating Text Elements

**P** `var delegate: NSTextContentManagerDelegate?`

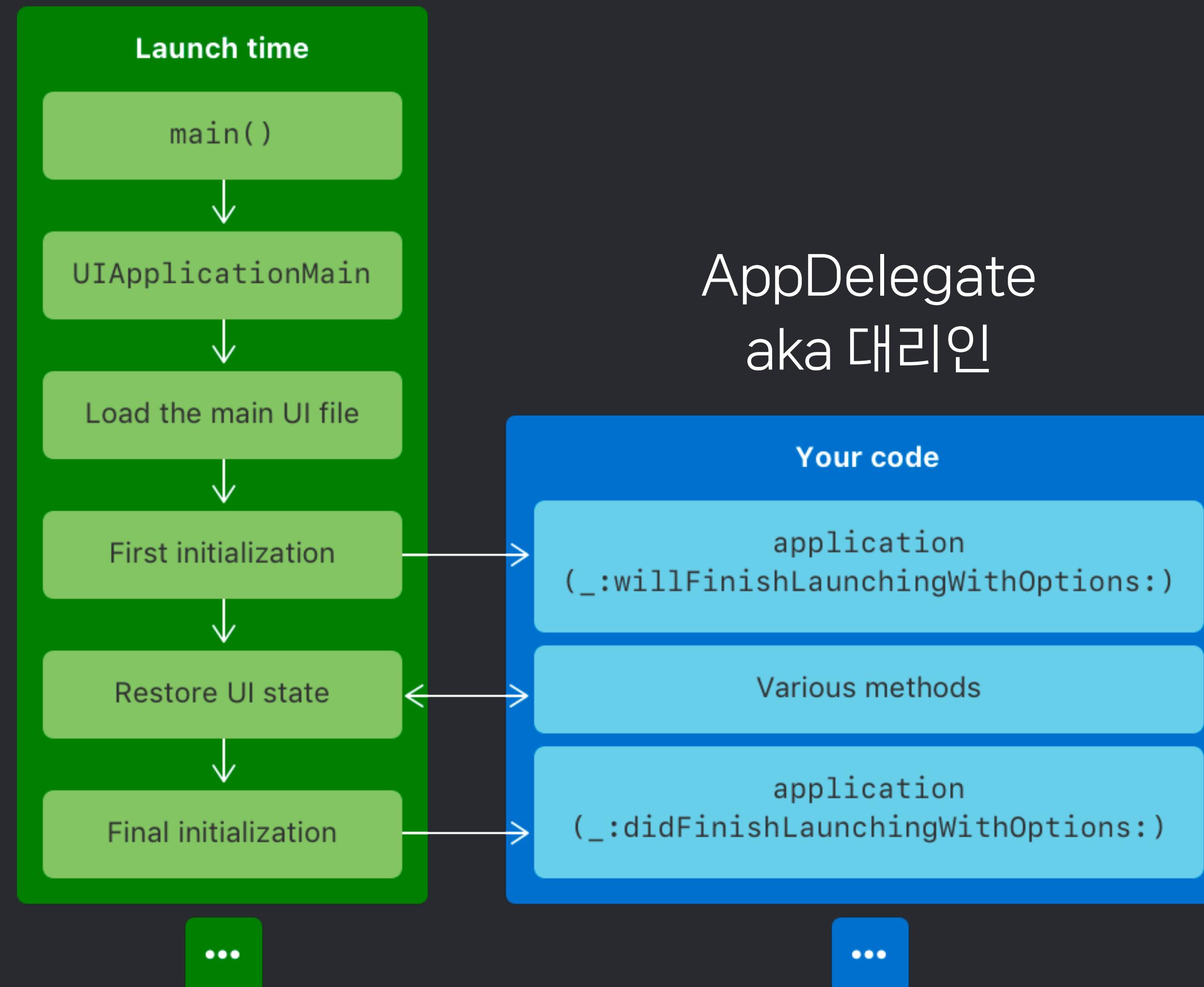
> **Pr** `NSTextContentManagerDelegate`

✓ **C** `NSTextContentStorage`

UIKit에는 Delegate Pattern이  
엄청나게 많이 사용되고 있다!

# AppDelegate

App



AppDelegate  
aka 대리인

# SceneDelegate

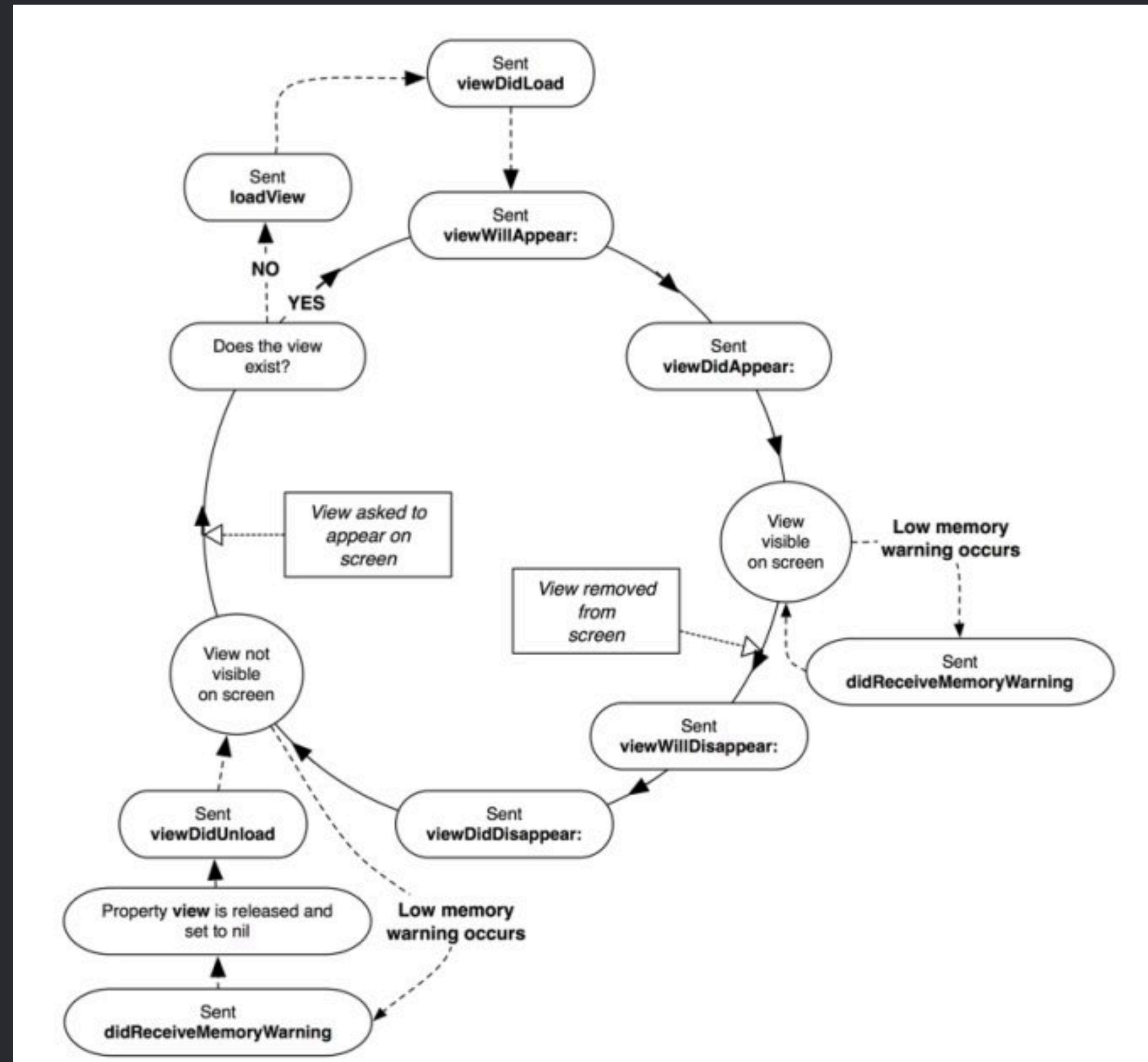
## AppDelegate의 확장판

- 태초에는 AppDelegate 밖에 없었음
- iOS 13부터 아이패드에 하나의 앱을 여러 창(Scene)으로 띄울 수 있게 되면서 SceneDelegate라는 개념이 생겨남.
- App 실행이 완료되었을 때뿐만 아니라, 각각의 Scene들이 백그라운드/포어그라운드로 전환될 때 어떤 작업이 수행되어야 하는지 개별적으로 관리할 수 있어야 하기 때문.
- 이것을 App의 LifeCycle(수명 주기)을 관리한다고 말한다.



# UIViewController

## View의 LifeCycle를 관리해주는 객체



- 모든 View에는 수명주기가 존재함.
- View가 각 수명주기 상태에 도달할 때마다 UIKit 프레임워크는 적절한 수명주기 메서드를 호출함.
- 개발자는 이 수명주기 메서드를 오버라이드해서, 각 수명주기마다 어떤 작업을 실행해야 하는지 정의줄 수 있음.
- 예: viewDidLoad, viewWillAppear, ...

# UIView

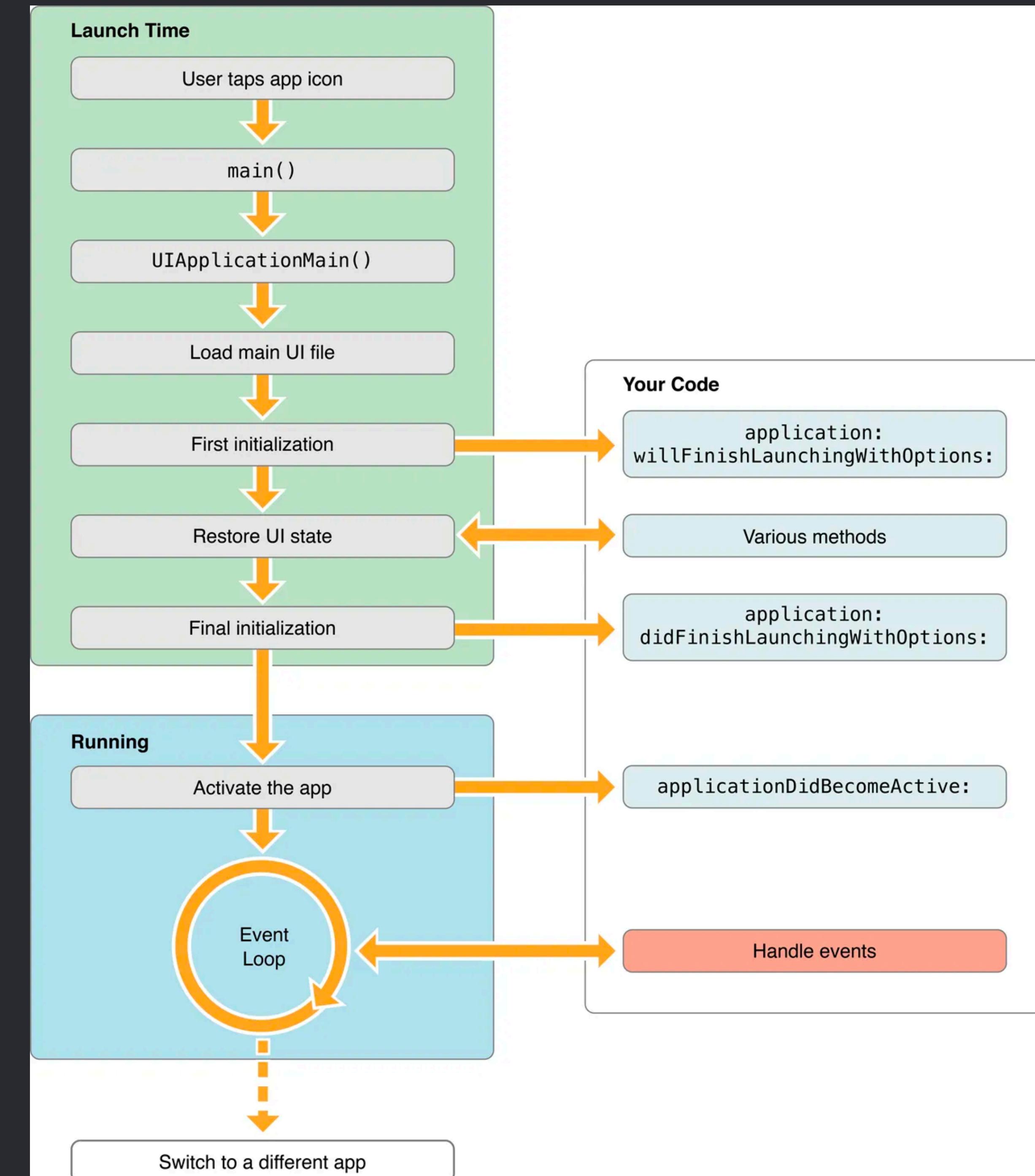
## 모든 View의 어머니

- 앱의 UI를 구성하기 위한 기초적인 Building Block
- UIView 안에 다른 UIView를 추가할 수 있음 (addSubview)
- UIButton, UILabel, UITextView 등등 UIKit의 수많은 컴포넌트가 UIView를 상속함

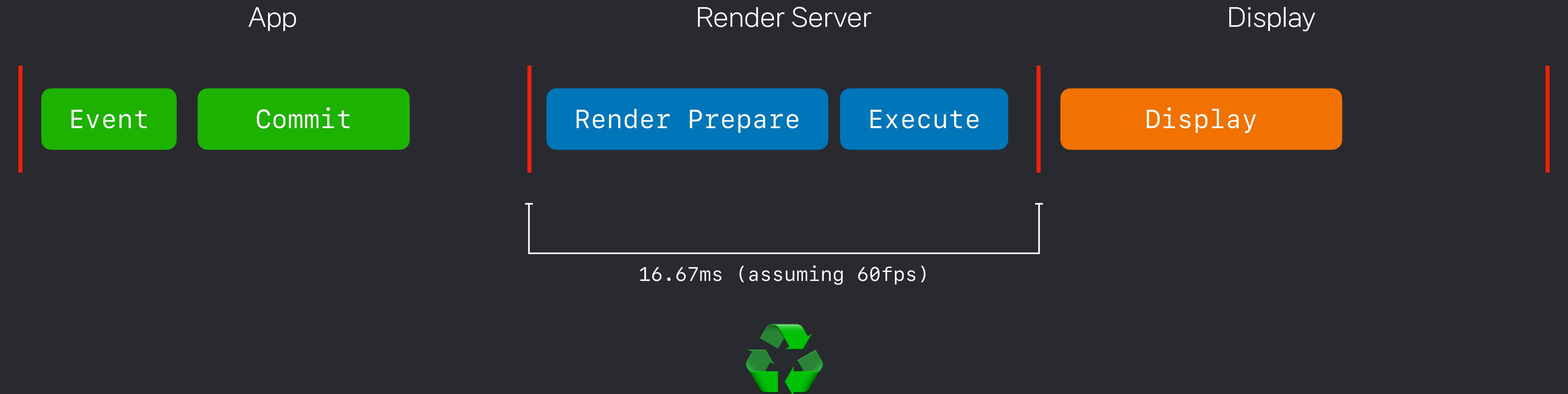
# Run Loop

## iOS 앱의 심장

- 루프를 계속 돌면서 새롭게 처리해야 하는 이벤트(e.g. 터치 입력) 가 있는지 확인
- 처리할 게 있다면, 개발자가 사전에 지정해둔 동작을 수행
- 메인 스레드에서 굉장히 빠르게 돌아가는 루프 이므로, 성능이 절대적으로 중요
- 네트워크 요청이나 복잡한 계산처럼 오래 걸릴 수 있는 작업은 백그라운드에 비동기로 처리해야 함 (이후 세미나에서 다룰 예정)

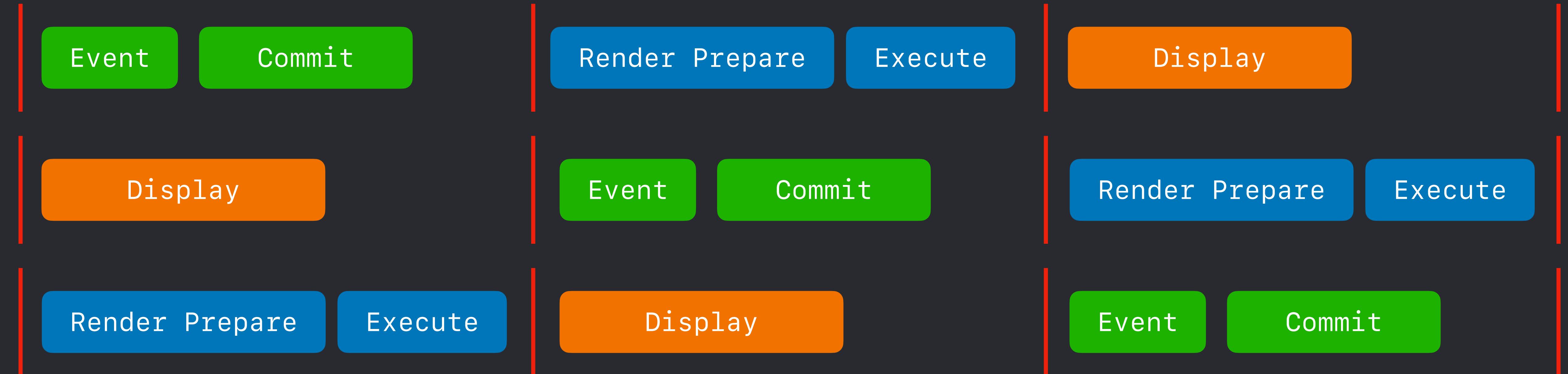


# Render Loop



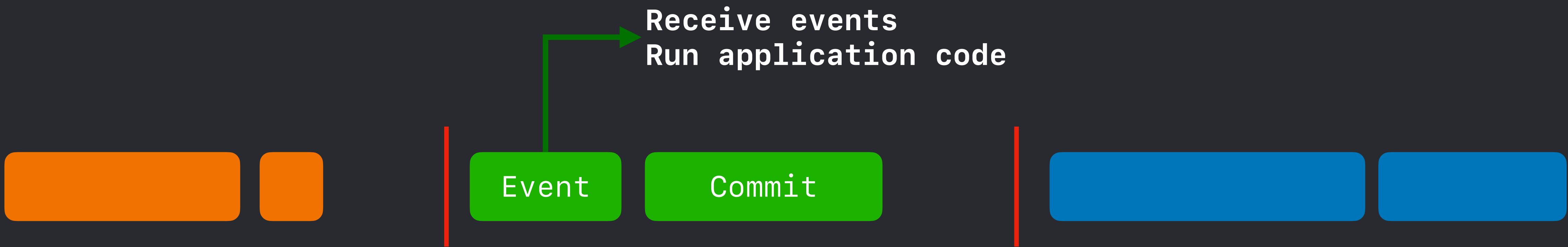
# Render Loop

## 는 병렬화가 가능



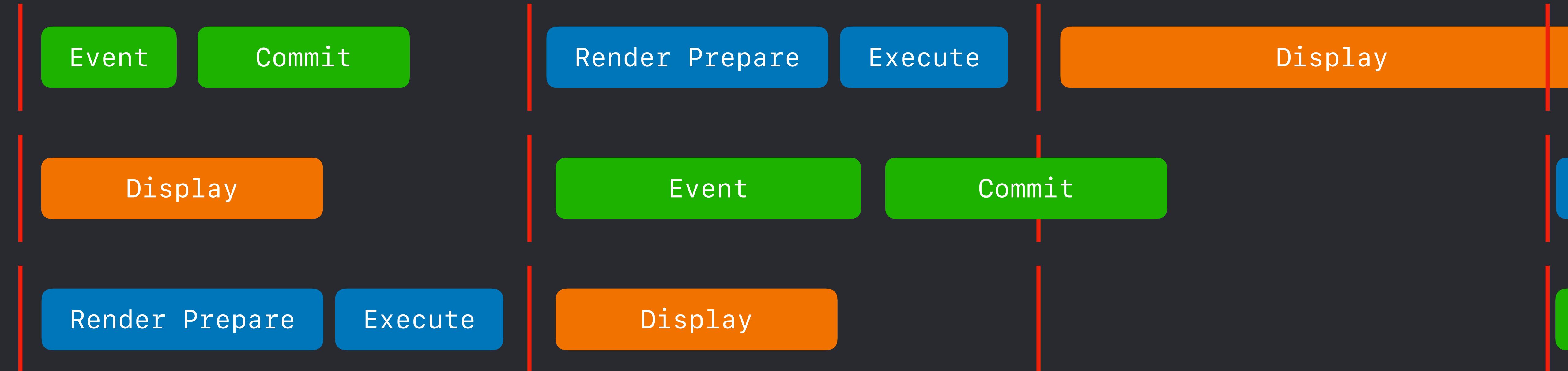
# Render Loop

## Commit Transaction



# Render Loop

16.67ms를 맞추지 못하면 버벅임이 발생





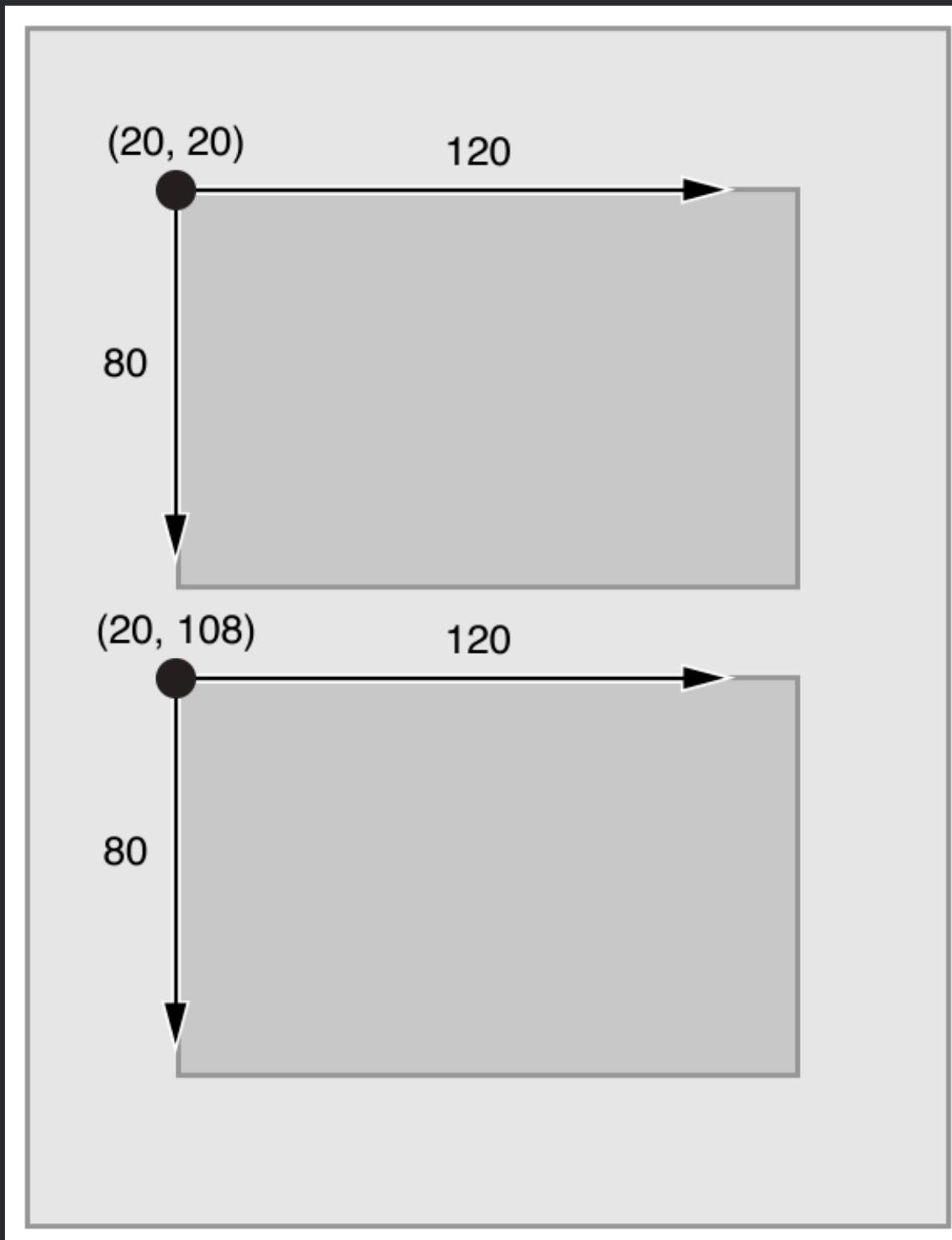
- Run Loop 관찰해보기
- Xcode에서 디버깅하는 법 알아보기

# Auto Layout

뷰의 위치와 크기를 자유자재로 조절하기

# Frame-based Layout

... versus Auto Layout

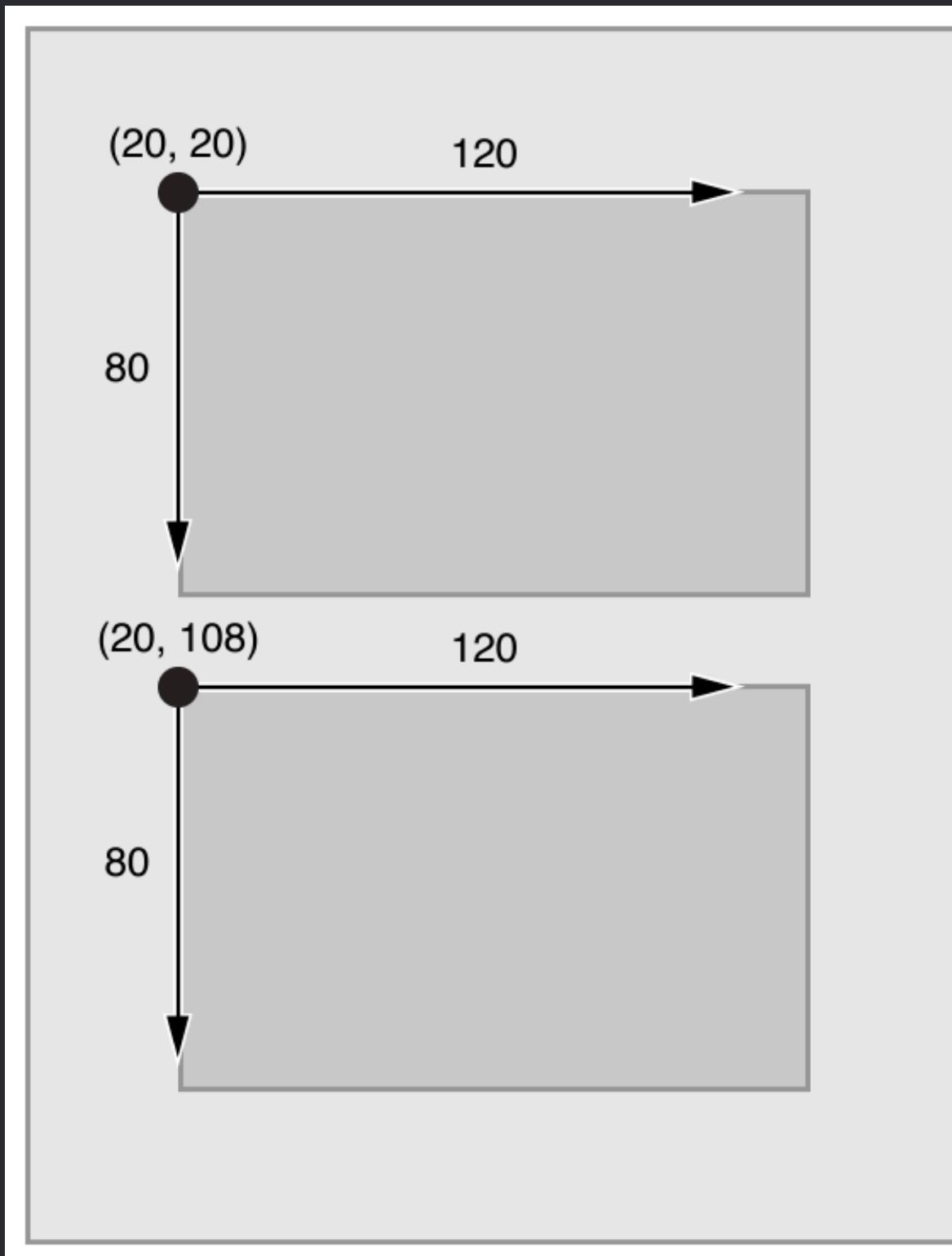


Frame-based Layout

```
class FrameView: UIView {  
    override init(frame: CGRect) {  
        super.init(frame: frame)  
        setupView()  
    }  
  
    private func setupView() {  
        let view1 = UIView(frame: CGRect(  
            x: 20, y: 20, width: 120, height: 80  
        ))  
        addSubview(view1)  
  
        let view2 = UIView(frame: CGRect(  
            x: 20, y: 108, width: 120, height: 80  
        ))  
        addSubview(view2)  
    }  
}
```

# Frame-based Layout

... versus Auto Layout



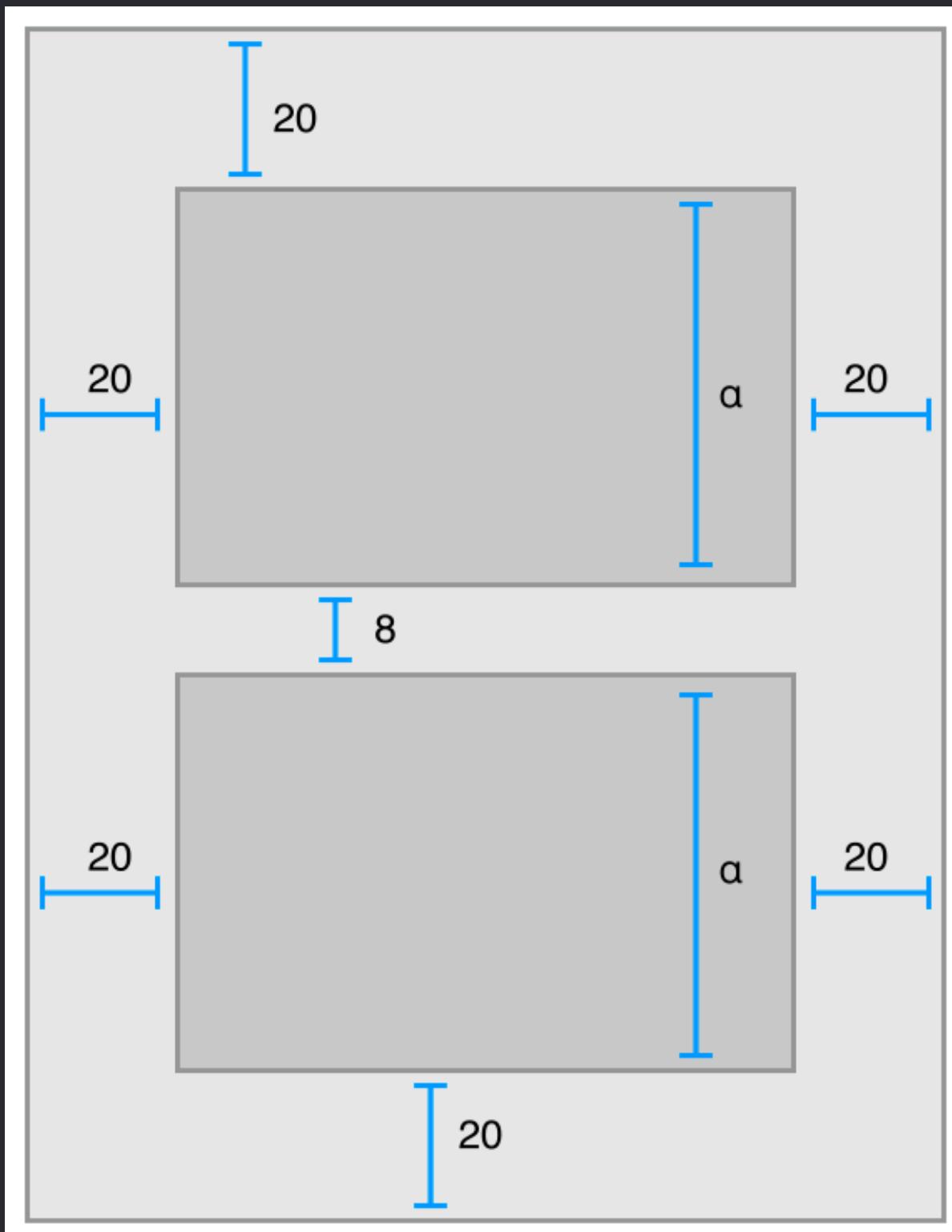
Frame-based Layout

```
class FrameView: UIView {  
    override init(frame: CGRect) {  
        super.init(frame: frame)  
        setupView()  
    }  
  
    private func setupView() {  
        let view1 = UIView(frame: CGRect(x: 20, y: 20, width: 120, height: 80))  
        addSubview(view1)  
        let view2 = UIView(frame: CGRect(x: 20, y: 108, width: 120, height: 80))  
        addSubview(view2)  
    }  
}
```

- Hard-coded values
- Manual calculation
- Difficult to maintain for dynamic content size

# Auto Layout

## ... versus Frame-based Layout



Auto Layout

- Think about view's relationships
- Relationships represented as a set of constraints (i.e. a system of linear equations and inequalities)
- Intuitive and easier to maintain!



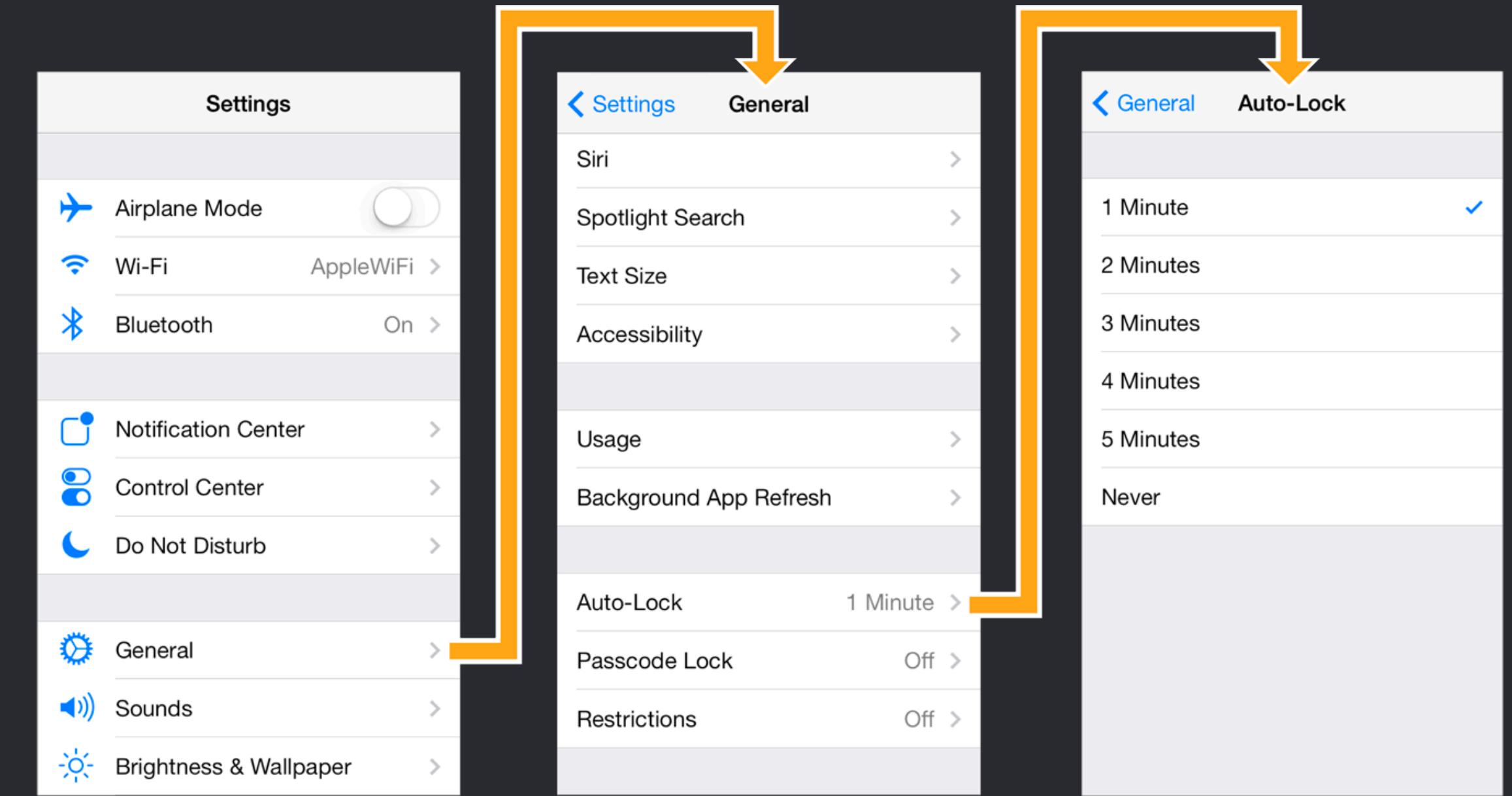
Auto Layout 연습

# UINavigationController 한 VC에서 다른 VC로 넘어가기

# UINavigationController

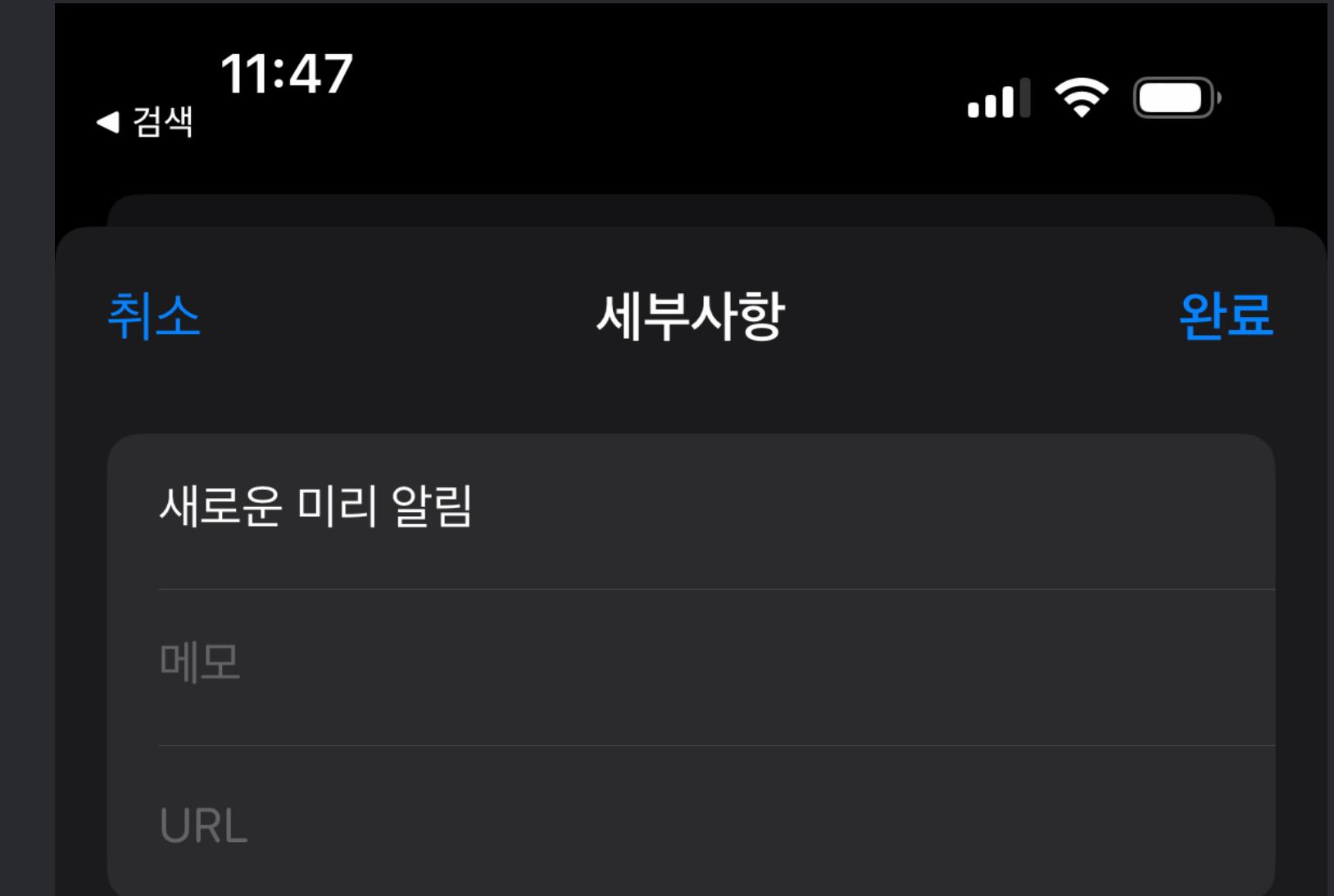
## Container View Controller

- UINavigationController은 UIViewController의 일종이지만, 여러 VC를 관리하는 Container VC라는 특징이 있음.
- 내부적으로 Stack 자료구조를 활용
  - pushViewController
  - popViewController
- 하위 VC에 NavigationBar가 자동으로 생성됨



# Modal Presentation

- 기존 Navigation 흐름에서 벗어나 그 위에 새로운 ViewController를 띄우고 싶을 때, present 메서드를 사용할 수 있음
- 유저의 즉각적인 액션이 필요할 때 사용
- 이렇게 띄운 화면 안에서 Stack Navigation을 사용하고 싶다면 UINavigationController을 present 할 수도 있음



# 세미나O 과제

기한: 9월 17일 오후 2시

## 1. Swift 공식 문서 (한글 번역본) 읽어오기

- 앞으로 여기 나온 문법은 모두 숙지하고 있다는 가정 하에 세미나가 진행될 예정입니다.
- 잘 이해가 되지 않는 개념이 있다면 Github Issue (Discussion) 통해 질문해주세요.

## 2. 간단한 로그인 화면 구현해보기

- 구체적인 스펙과 제출 방법은 세미나 레포에서 확인 가능

# 출결 및 과제 관련 공지

- 대면 참여가 디폴트입니다. 피치 못할 사정이 있다면 전날 DM 주세요.
- 지각 2회 → 결석 1회로 잡겠습니다 (지각은 10분 이후부터).
- 결석 2회 이상 시 세미나 과제 제출 여부와 무관하게 탈락입니다.
- 과제 미제출 / 심각한 구현 누락시 탈락입니다.
- Grace Day는 세미나 전체 통틀어서 3회 사용 가능합니다. (PR 댓글로 알려주세요)
  
- 매 과제마다 체크리스트가 제공될 예정입니다. 채점도 이 체크리스트를 기준으로 진행됩니다.
- 하나라도 충족 못하면 탈락... 은 아니지만, 최대한 전부 채워주세요!

Q & A