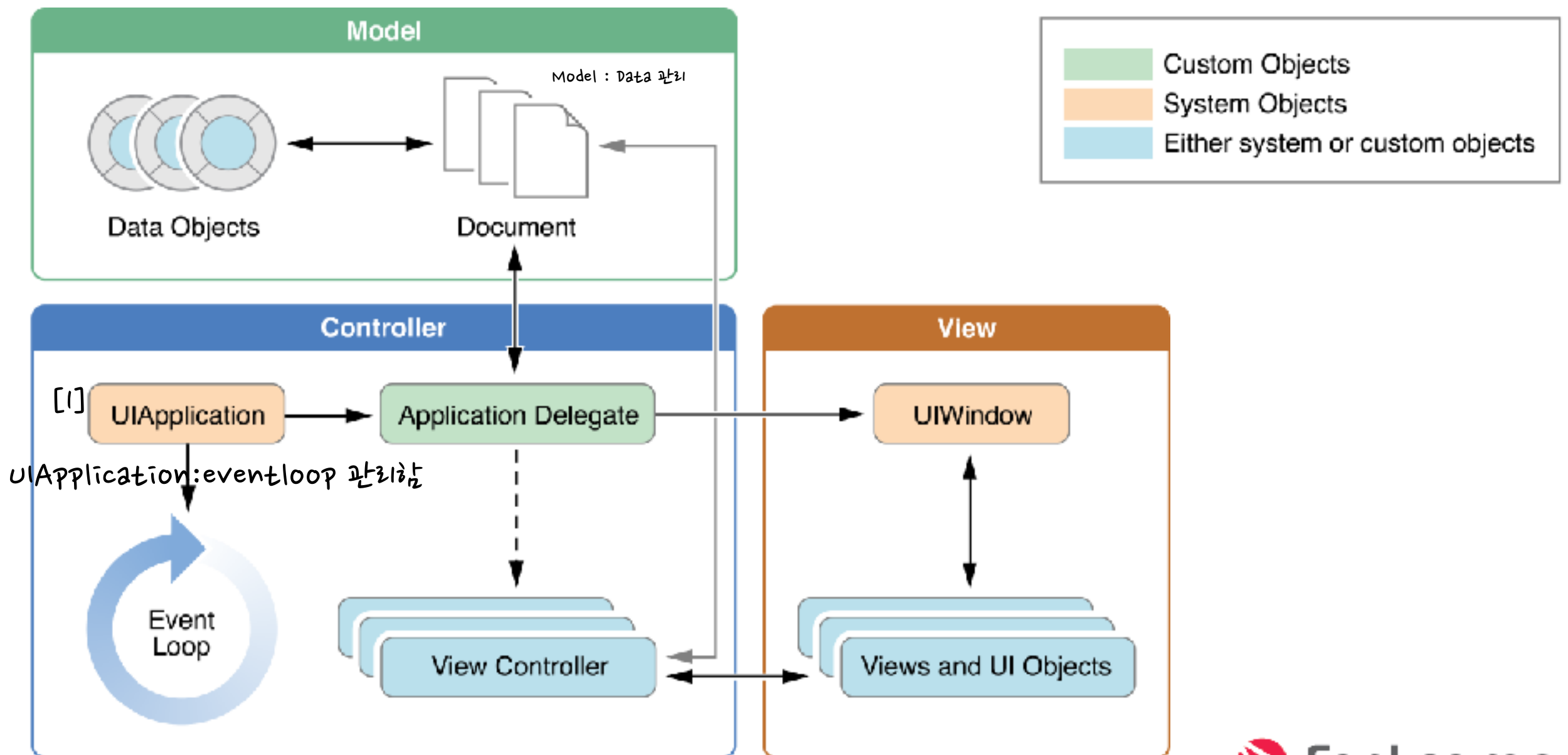


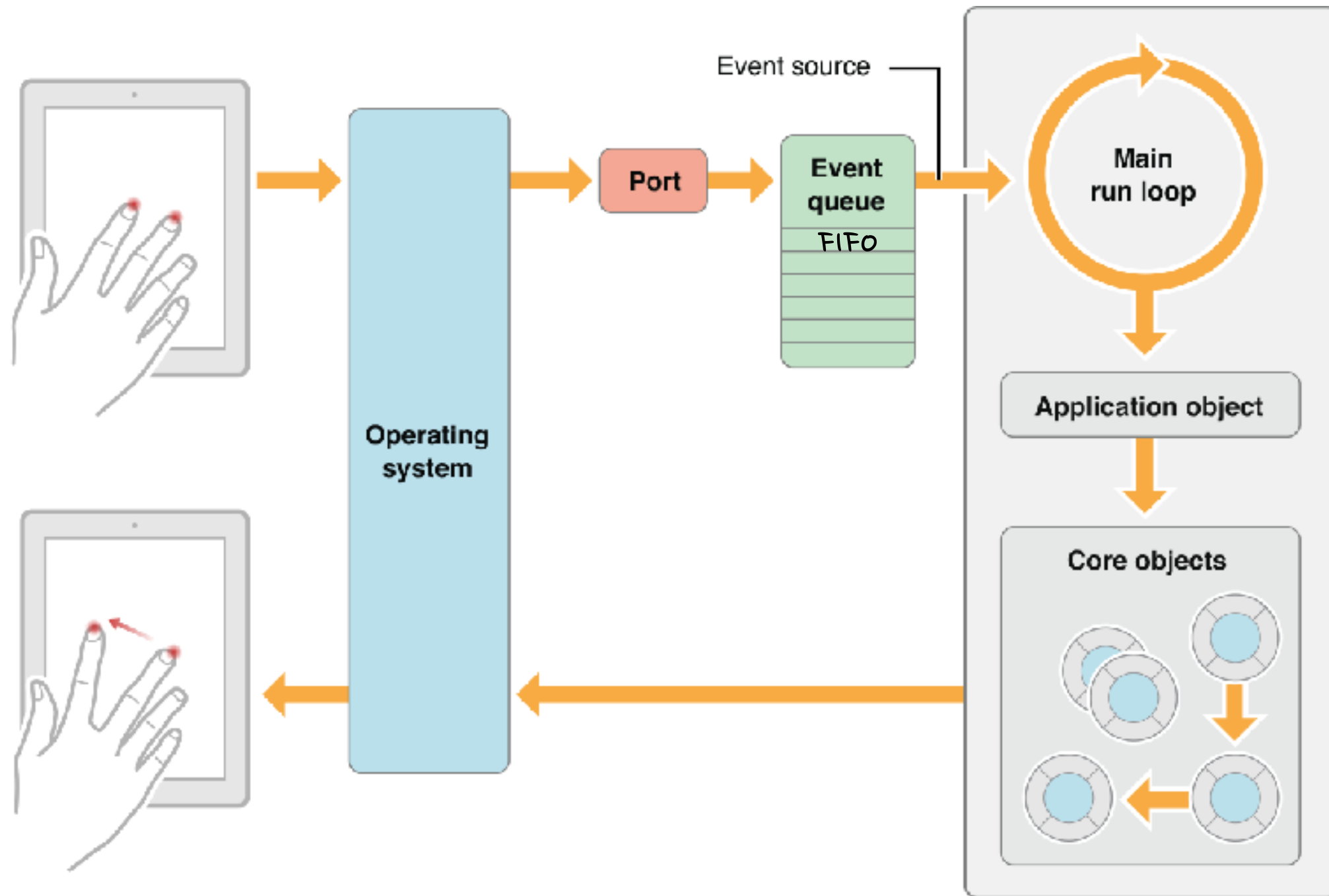
Application Life Cycle

The Structure of an App

During startup, the UIApplicationMain function sets up several key objects and starts the app running. At the heart of every iOS app is the UIApplication object



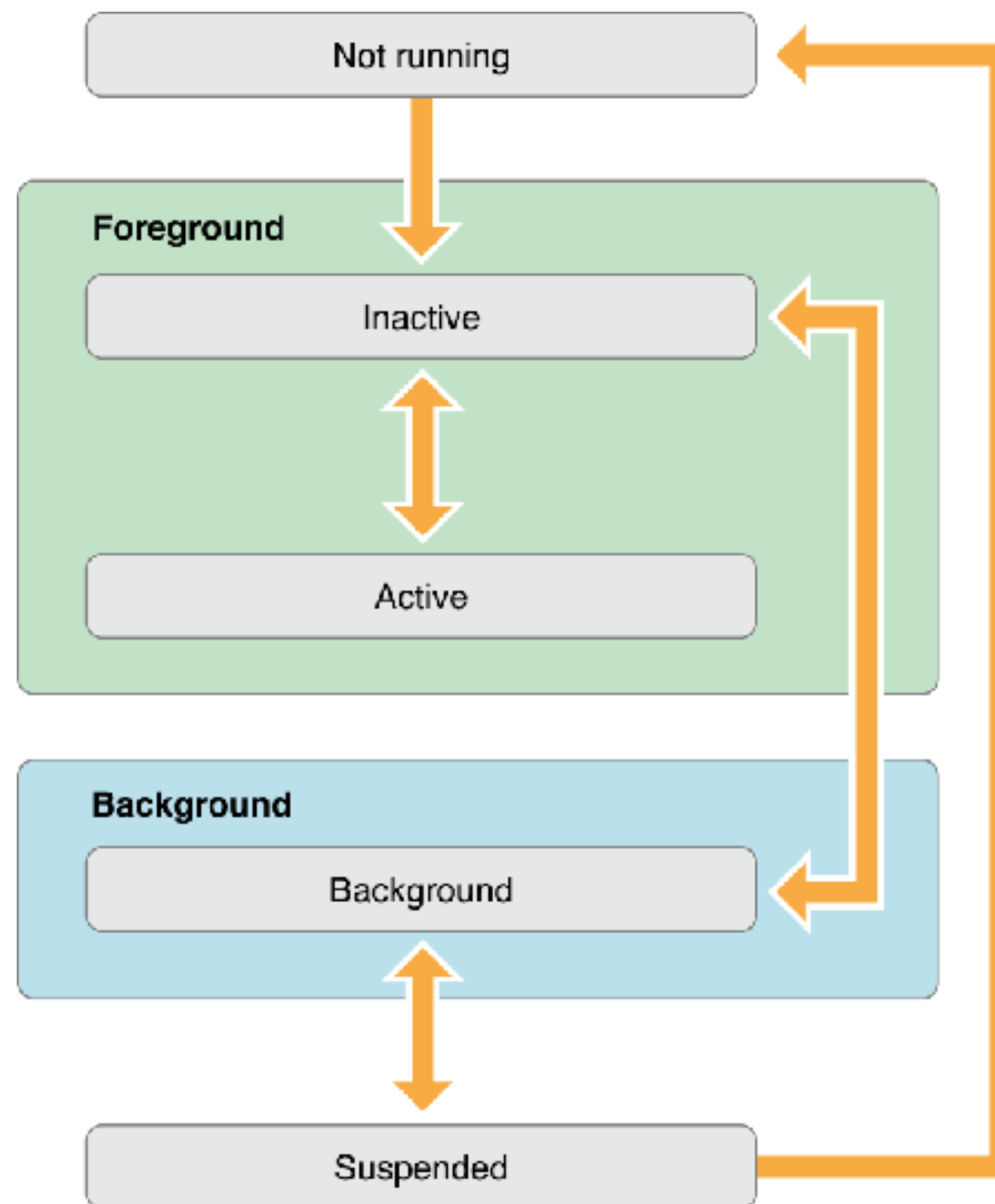
The Main Run Loop



Event에 대한 처리

- Touch : 발생한 이벤트에 대한 뷰가 처리
- Remote control & Shake motion events : First responder객체
- Accelerometer/Magnetometer/Gyroscope : 각각의 객체로 전달
- Location :CoreLocation 객체
- Redraw : 업데이트를 원하는 뷰가 처리

Execution States for Apps



Execution States for Apps

- Not Running : 실행되지 않았거나, 시스템에 의해 종료된 상태
- Inactive : 실행 중이지만 이벤트를 받고있지 않은 상태. 예를들어, 앱 실행 중 미리알림 또는 일정 알럿이 화면에 덮여서 앱이 실질적으로 이벤트를 받지 못하는 상태 등을 뜻합니다.
- Active : 어플리케이션이 실질적으로 활동하고 있는 상태.
- Background : 백그라운드 상태에서 실질적인 동작을 하고 있는 상태. 예를 들어 백그라운드에서 음악을 실행 하거나, 걸어온 길을 트래킹 하는 등의 동작을 뜻합니다.
- Suspended : 백그라운드 상태에서 활동을 멈춘 상태. 빠른 재실행을 위하여 메모리에 적재된 상태이지만 실질적으로 동작하고 있지는 않습니다. 메모리가 부족할 때 비로소 시스템이 강제종료하게 됩니다.

Execution States for Apps

- Not Running : 실행되지 않았거나, 시스템에 의해 종료된 상태
- Inactive : 실행 중이지만 이벤트를 받고있지 않은 상태. 예를들어, 앱 실행 중 미리알림 또는 일정 알람이 화면에 덮여서 앱이 실질적으로 이벤트를 받지 못하는 상태 등을 뜻합니다.
- Active : 어플리케이션이 실질적으로 활동하고 있는 상태.
- Background : 백그라운드 상태에서 실질적인 동작을 하고 있는 상태. 예를들어 백그라운드에서 음악을 실행 하거나, 걸어온 길을 트래킹 하는 등의 동작을 뜻합니다.
- Suspended : 백그라운드 상태에서 활동을 멈춘 상태. 빠른 재실행을 위하여 메모리에 적재된 상태이지만 실질적으로 동작하고 있지는 않습니다. 메모리가 부족할 때 비로소 시스템이 강제종료하게 됩니다.

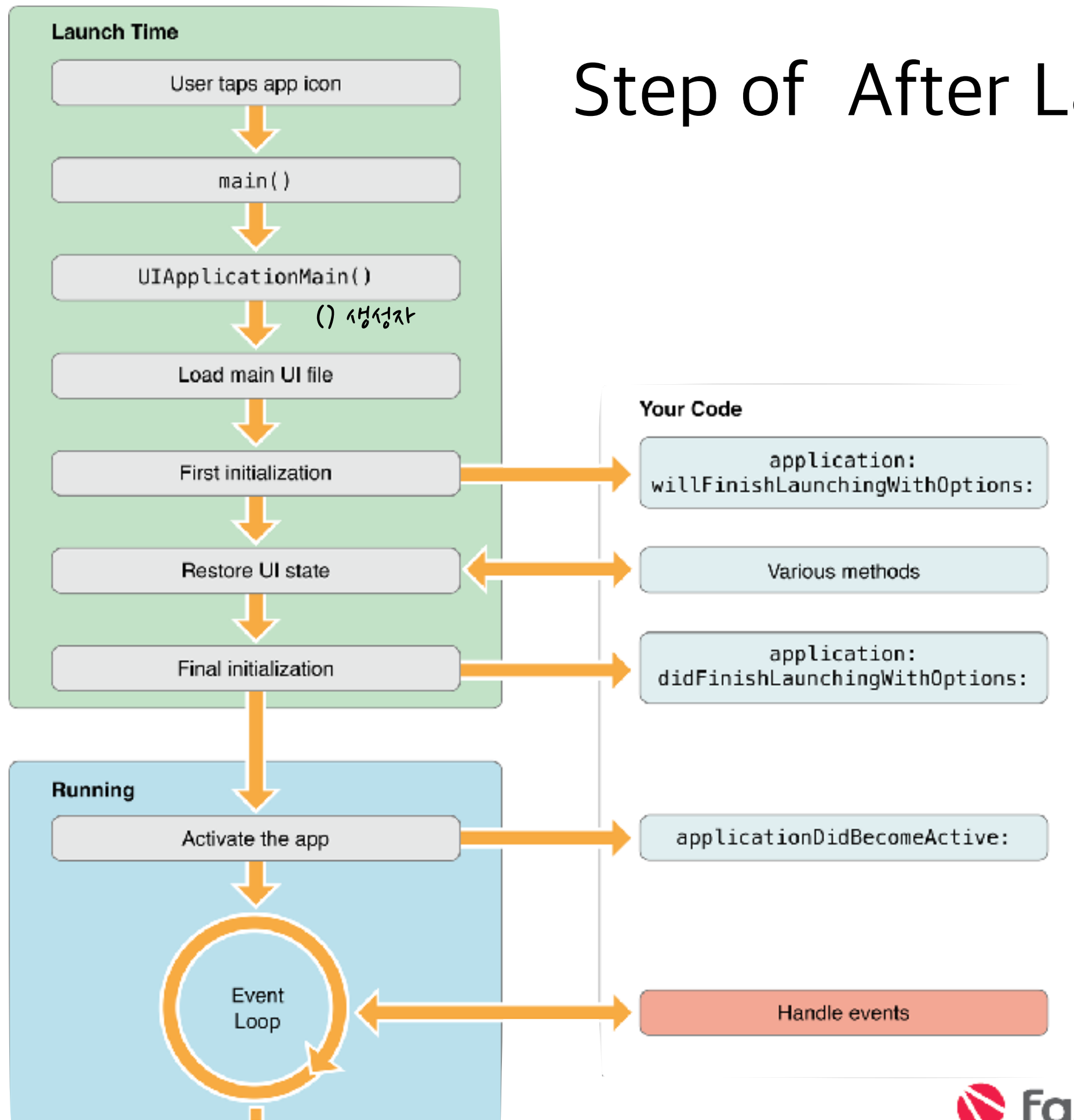
<출력용>

Call to the methods of your app delegate object

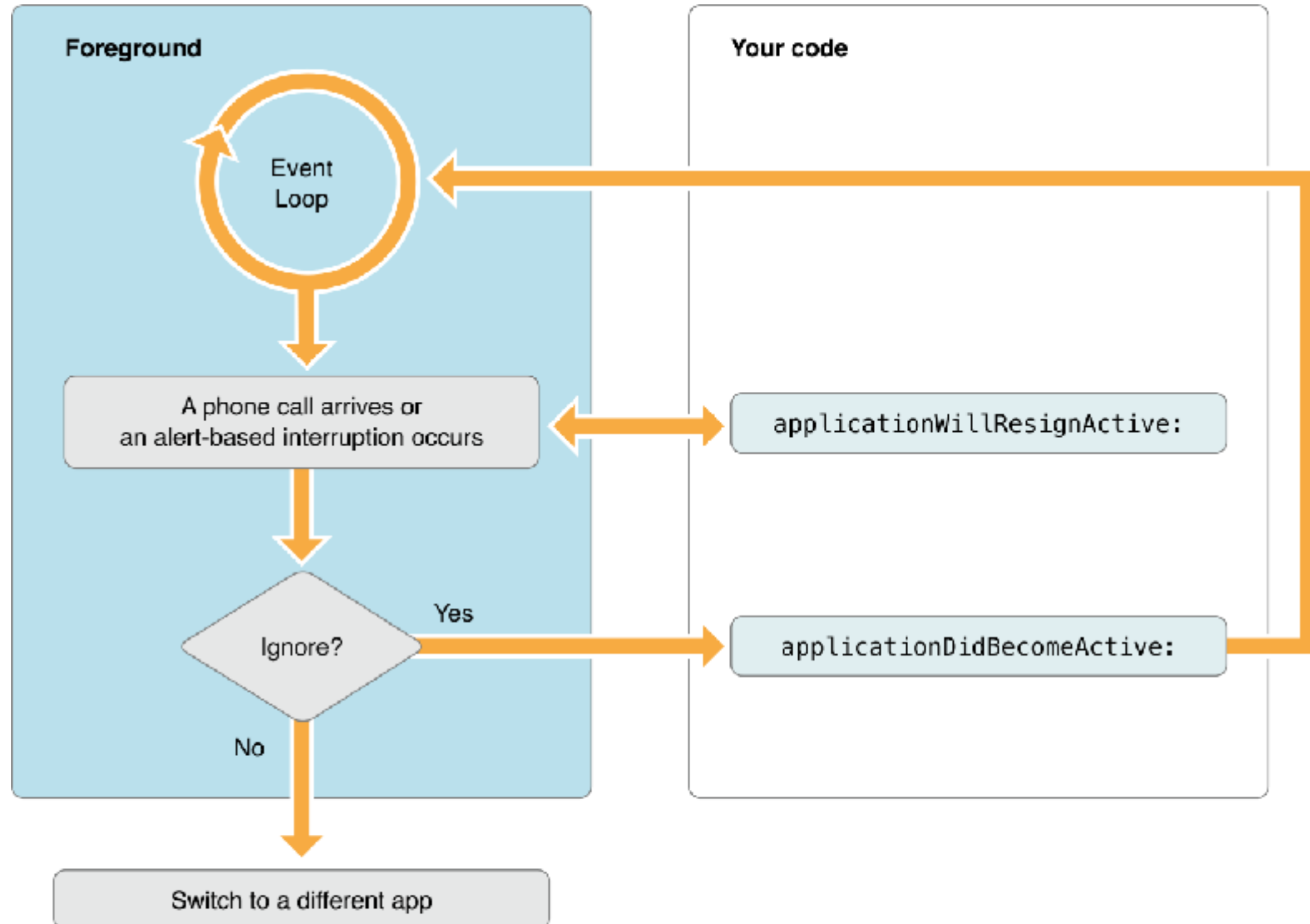
대부분의 상태변화를 app delegate 객체에 호출되는 메소드를 오버라이드하여 알아챌 수 있습니다.

- `application:willFinishLaunchingWithOptions:`
 - 어플리케이션이 최초 실행될 때 호출되는 메소드
- `application:didFinishLaunchingWithOptions:`
 - 어플리케이션이 실행된 직후 사용자의 화면에 보여지기 직전에 호출.
- `applicationDidBecomeActive:`
 - 어플리케이션이 Active 상태로 전환된 직후 호출.
- `applicationWillResignActive:`
 - 어플리케이션이 Inactive 상태로 전환되기 직전 호출
- `applicationDidEnterBackground:`
 - 어플리케이션이 백그라운드 상태로 전환된 직후 호출.
- `applicationWillEnterForeground:`
 - 어플리케이션이 Active 상태가 되기 직전에, 화면에 보여지기 직전의 시점에 호출.
- `applicationWillTerminate:`
 - 어플리케이션이 종료되기 직전에 호출.

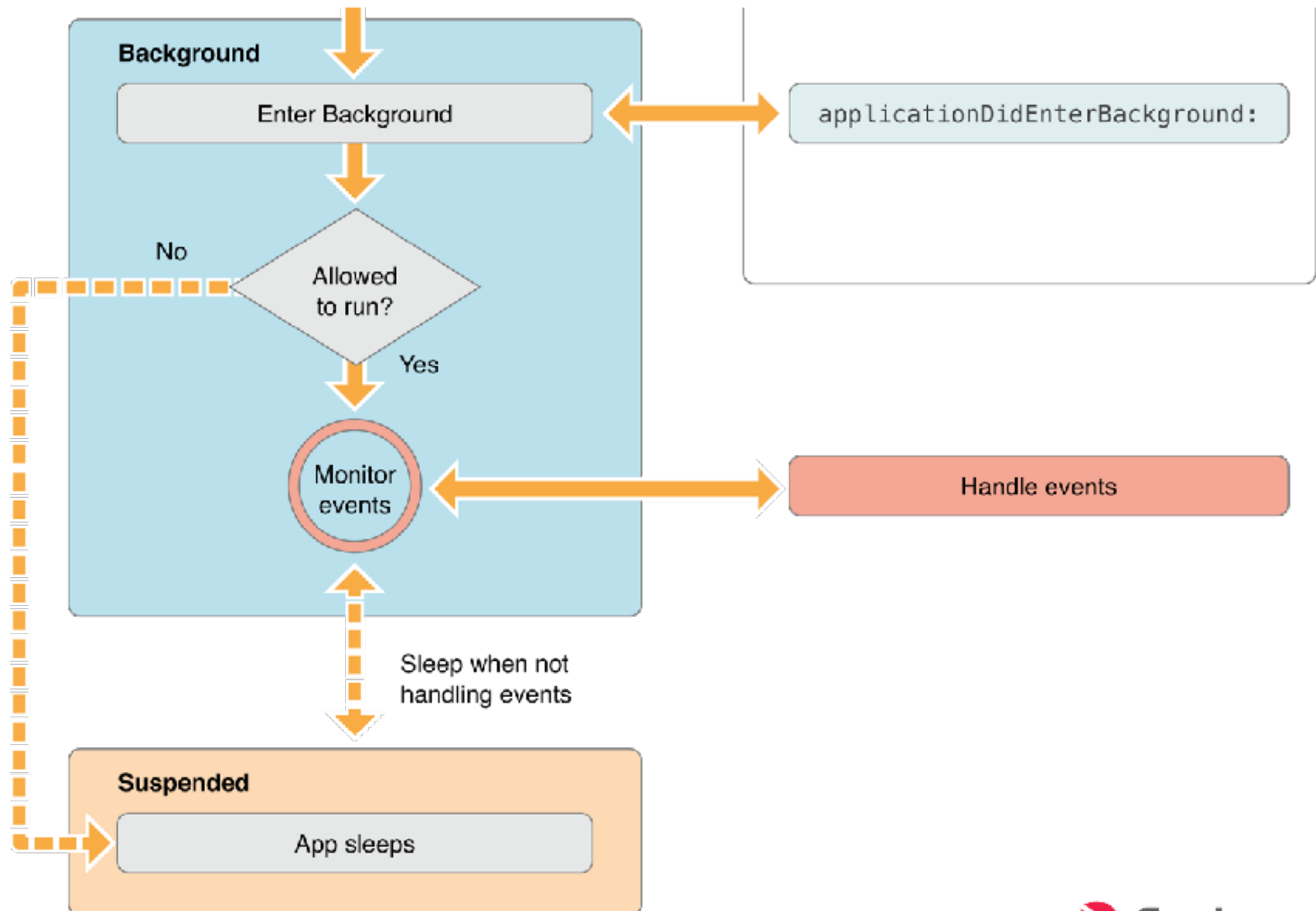
Step of After Launch



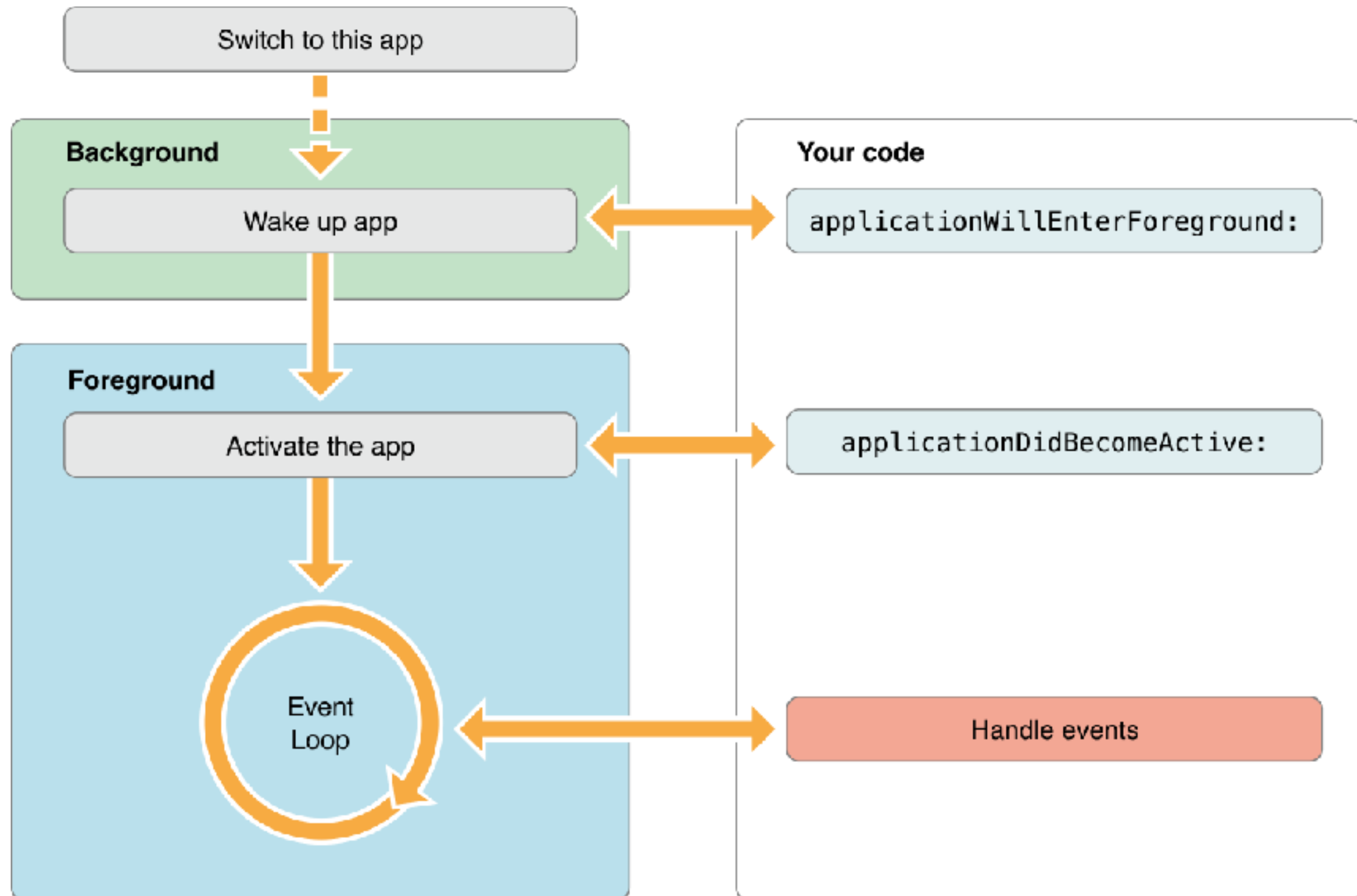
Step of Interruptions



Step of Enter Background



Step of Enter Foreground



Supported Background Tasks

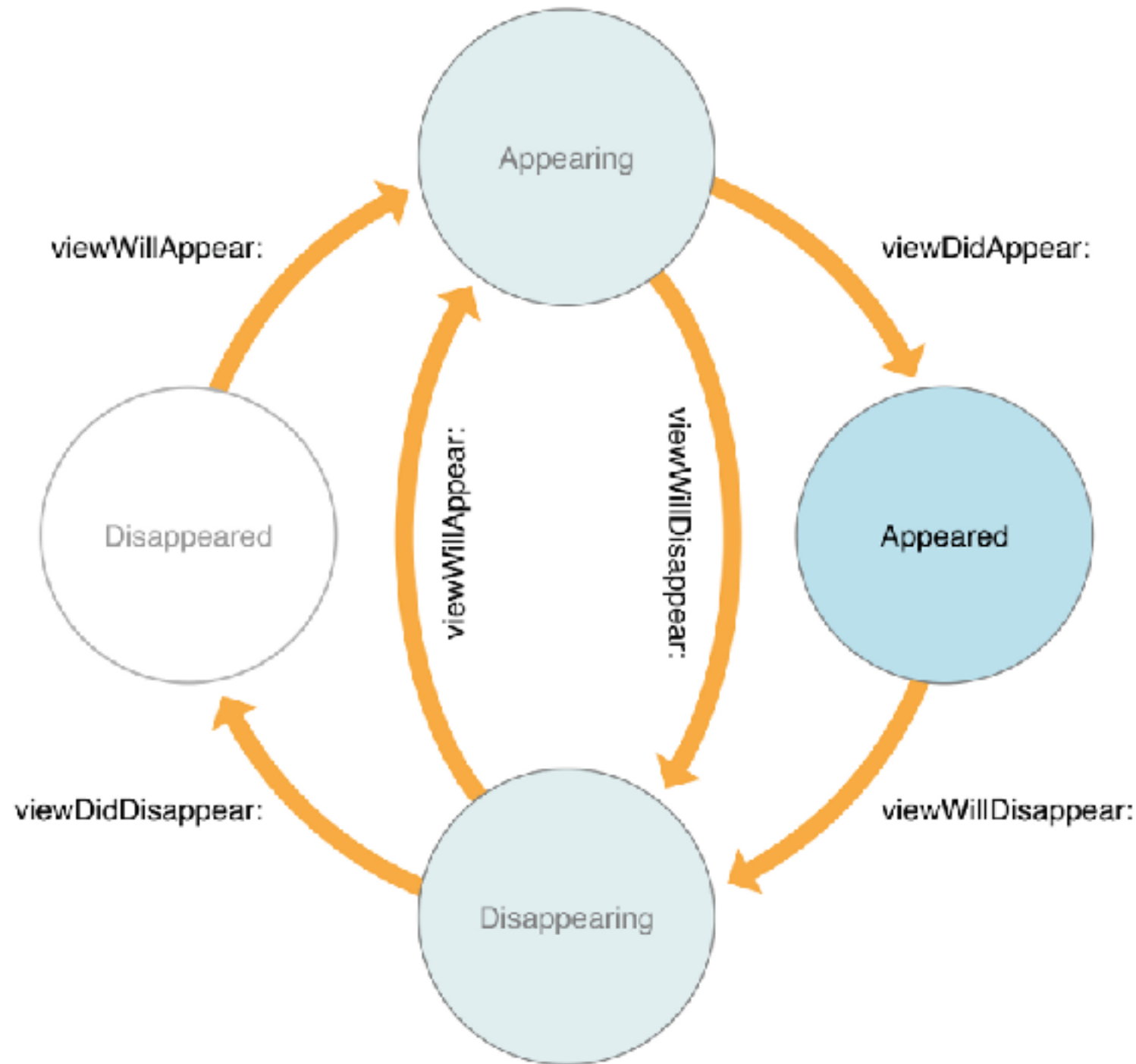
- Audio and AirPlay (음악)
- Location updates (위치 정보)
- Voice over IP (인터넷을 사용한 음성통화)
- Newsstand downloads(뉴스 스탠드 다운로드)
- External accessory communication (기타 하드웨어 액세서리)
- Bluetooth LE accessories (블루투스 액세서리 사용)
- Background fetch (네트워크를 통한 일반적인 다운로드나 미완료된 작업)
- Remote notifications (PushNotification)

확인해 볼까요?

- 실제 로그를 찍어 상태를 확인해 봅시다.

UIViewController의 생명주기 메소드

- 프로그래머가 직접 호출 불가
- 오버라이드 하는 메소드이므로 꼭 해당 메소드 내에서 super.메소드 을 통해 기존 메소드를 꼭 호출해야 된다.



생명주기 메소드

`override func loadView()` : UIViewController의 view가 생성될 때 호출

`override func viewDidLoad()` :

UIViewController가 인스턴스화 된 직후(메모리에 객체가 올라간 직후) 호출 처음 한 번 세팅해 줘야 하는 값들을 넣기에 적절

`override func viewWillAppear(_ animated: Bool)` :

view가 화면에 보여지기 직전에 호출 화면이 보여지기 전에 준비할 때 사용.

animated 파라미터는 뷰가 애니메이션을 동반하여 보여지게 되는지 시스템에서 전달해주는 불리언 값

`override func viewWillLayoutSubviews()` : view의 하위뷰들의 레이아웃이 결정되기 직전 호출

`override func viewDidLayoutSubviews()` :

view의 하위뷰들의 레이아웃이 결정된 후 호출. 주로 view의 하위뷰들이 사이즈 조정이 필요할 때 호출

`override func viewDidAppear(_ animated: Bool)` :

view가 화면에 보여진 직후에 호출. 화면이 표시된 이후 애니메이션 등을 보여주고 싶을 때 유용

`override func viewWillDisappear(_ animated: Bool)` : view가 화면에서 사라지기 직전에 호출

`override func viewDidDisappear(_ animated: Bool)` : view가 화면에서 사라진 직후에 호출

확인해 볼까요?

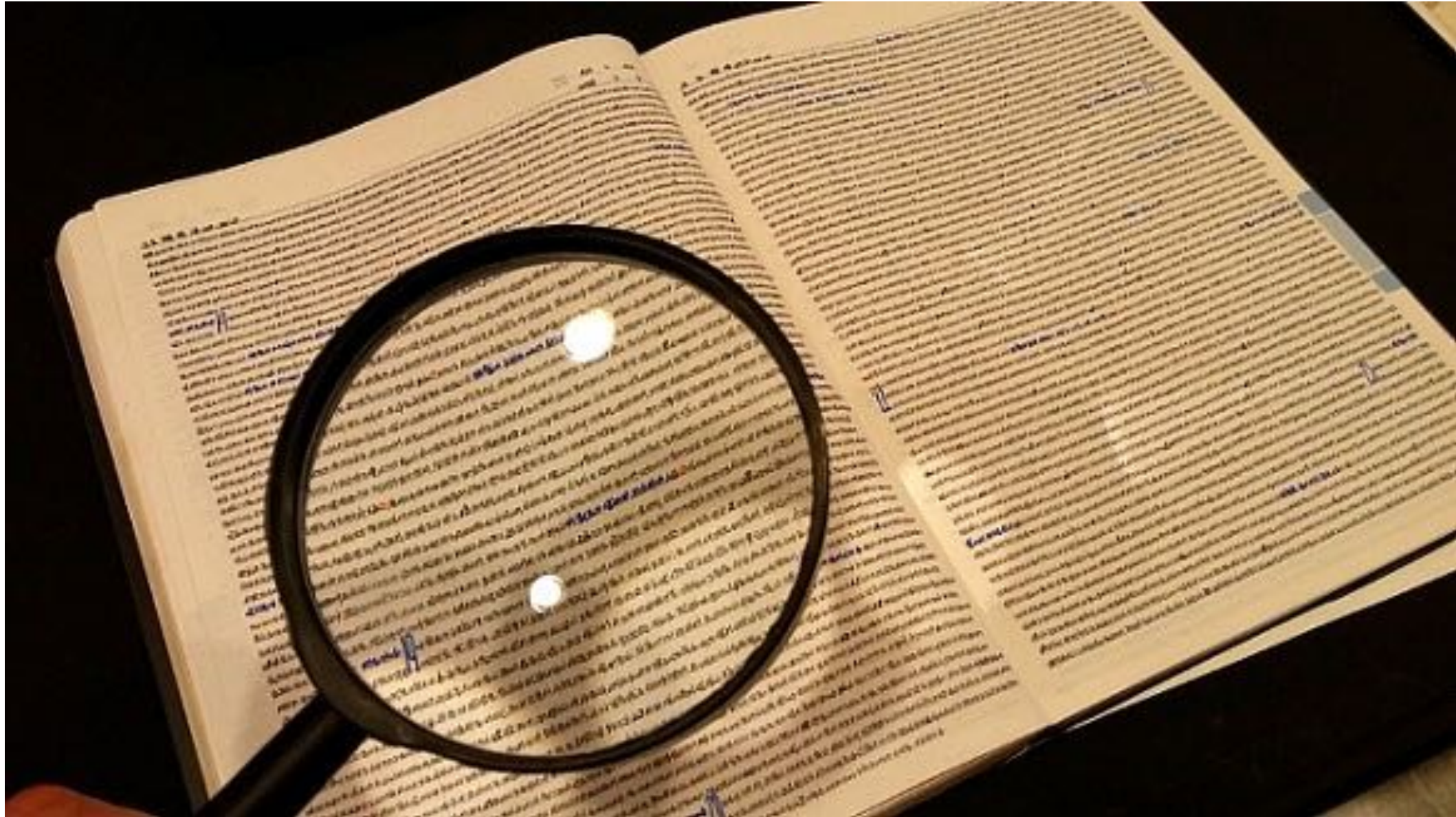
- 실제 로그를 찍어 상태를 확인해 봅시다.

UITextField

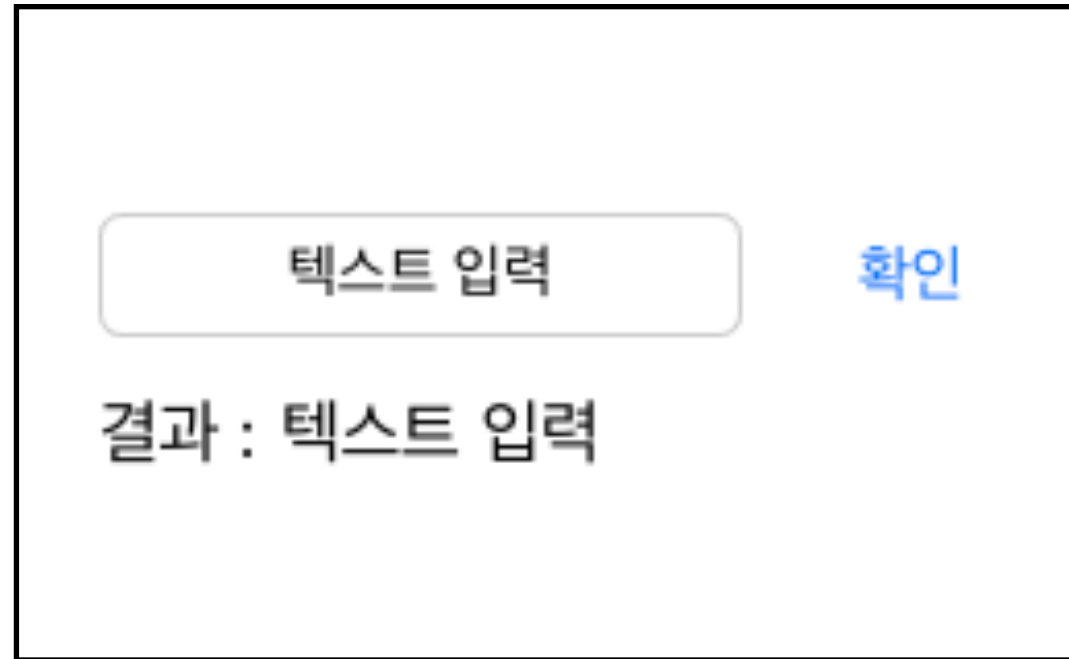
- 사용자 텍스트 입력을 위한 UI Component.



UITextField



UITextField - 실습



텍스트 입력 확인

결과 : 텍스트 입력

1. 텍스트 필드에서 텍스트 입력
2. 확인 버튼 클릭
3. 아래 레이블에 입력된 텍스트 보여주기

Delegate Pattern

Protocol

명세서는 무조건 따라야 한다

프로토콜 : 명세서

- 프로토콜은 원하는 작업이나 기능을 구현되도록 메서드, 프로퍼티등으로 요구 사항의 청사진을 정의합니다.
- 클래스, 구조체, 열거형은 프로토콜을 채택하면, 프로토콜에서 요구한 사항에 대해 구현해야 됩니다.
- 프로토콜을 통해 공통적인 작업을 강제 할수 있으며, 해당 프로토콜을 채택한 사람이 구현한 메소드에 대한 정보도 알수 있다.

Protocol 문법

```
protocol Runnable {  
    var regCount: Int { get set }  
    func run()  
}
```

```
class Animal: Runnable{  
    var regCount: Int = 0  
    func run()  
    {  
  
    }  
}
```

Protocol 채택

```
protocol Runnable {  
    var regCount: Int { get set }  
    func run()  
}
```

```
protocol Flying : Runnable {  
    var wingCount: Int { get set }  
}
```

```
class Animal: Flying {  
    var wingCount: Int = 0  
    var regCount: Int = 0  
    func run()  
    {  
  
    }  
}
```


추상클래스로의 Protocol

프로토콜을 추상 클래스처럼 사용할수 있다.

다음과 같은 클래스가 있고, racing 이라는 함수를 구현하려고 한다면!

```
class Dog: Runnable {  
    //...  
}  
  
class Horse: Runnable {  
    //...  
}  
  
func racing(animals:[Runnable]) -> Runnable  
{  
  
}
```

프로토콜타입으로 사용가능하다.

```
let winner: Runnable = racing (animals: [Dog (), Horse ()])
```

Delegate

- 델리게이트는 클래스나 구조체에서의 일부분의 할 일을 다른 인스턴스에게 대신 하게 하는 디자인 패턴!
- 뷰가 받은 이벤트나 상태를 ViewController에게 전달해주기 위해 주로 사용된다.(ex:UIScrollViewDelegate...)
- ViewController를 통해 View구성에 필요한 데이터를 받는 용도로도 사용(ex:UITableViewDataSource)

직접 예제를 통해 확인해보자!

- CustomDelegate만들기!!

Delegate 선언부

```
class CustomView: UIView {  
    var delegate: CustomViewDelegate?  
  
    override func layoutSubviews() {  
        delegate?.viewFrameChanged(newFrame: self.frame)  
    }  
}  
  
protocol CustomViewDelegate {  
    func viewFrameChanged(newFrame: CGRect )  
}
```

Delegate 구현부

```
class ViewController: UIViewController, CustomViewDelegate {  
  
    override func viewDidLoad() {  
        super.viewDidLoad()  
  
        let custom = CustomView()  
        custom.delegate = self  
    }  
  
    func viewFrameChanged(newFrame: CGRect) {  
        // 뷰의 프레임이 변경될때마다 불림  
    }  
}
```

구조

<Delegate 구현부>

```
class ViewController: CustomDelegate {  
    override func viewDidLoad() {  
        super.viewDidLoad()  
        let custom = CustomView()  
        custom.delegate = self  
    }  
  
    func viewFrameChanged(newFrame:...) {  
        // 뷰의 프레임이 변경될때마다 불림  
    }  
}
```

<Delegate 선언부>

```
class CustomView: UIView {  
    var delegate: CustomViewDelegate?  
    func layoutSubviews() {  
        delegate?.viewFrameChanged...  
    }  
}  
  
protocol CustomViewDelegate {  
    func viewFrameChanged...  
}
```

구조

1. 프로토콜 생성 <delegate 구현부>

2. 클래스에 delegate 프로퍼티 생성

```
class ViewController: CustomDelegate {  
    • 일반적으로 delegate란 이름 사용  
    • 타입은 프로토콜 추상화 타입  
    override func viewDidLoad() {  
        super.viewDidLoad()  
    }  
}
```

3. delegate 인스턴스의 메소드 실행

- 현재 customView 입장에서
let custom = CustomView()
custom.delegate = self
} delegate instance가 존재하는지는
모른다
func viewFrameChanged(newFrame:...) {
 하지만 만약 어떤 instance(A)가 나의
 delegate instance 값을 할당했다
 //나의 프레임이 변경될때마다 불리
}면, 분명 A는 나의 프로토콜을 채택
했으며 (타입이 같기때문에)메소드를
구현 했다는 것을 인지!
• delegate method를 사용해서 메소
드 실행 및 리턴값을 받아와 사용

<Delegate 선언부>

```
class CustomView: UIView {  
    2 var delegate: CustomViewDelegate?  
  
    func layoutSubviews() {  
        delegate?.viewFrameChanged...  
    }  
}  
  
1 protocol CustomViewDelegate {  
    func viewFrameChanged...  
}
```


구조

<Delegate 구현부>

```
class ViewController: CustomViewDelegate {  
    override func viewDidLoad() {  
        super.viewDidLoad()  
  
        let custom = CustomView()  
        custom.delegate = self  
    }  
    func viewFrameChanged(newFrame:...) {  
        //뷰의 프레임이 변경될때마다 불림  
    }  
}
```

<Delegate 선언부>

1. CustomView Delegate 채택

```
class CustomView: UIView {  
    var delegate: CustomViewDelegate?  
    func viewFrameChanged() {  
        delegate?.viewFrameChanged()  
    }  
}
```

3. custom instance의 delegate 프로퍼티에 자기 자신의 인스턴스를 할당 (프로토콜 추상화 타입)

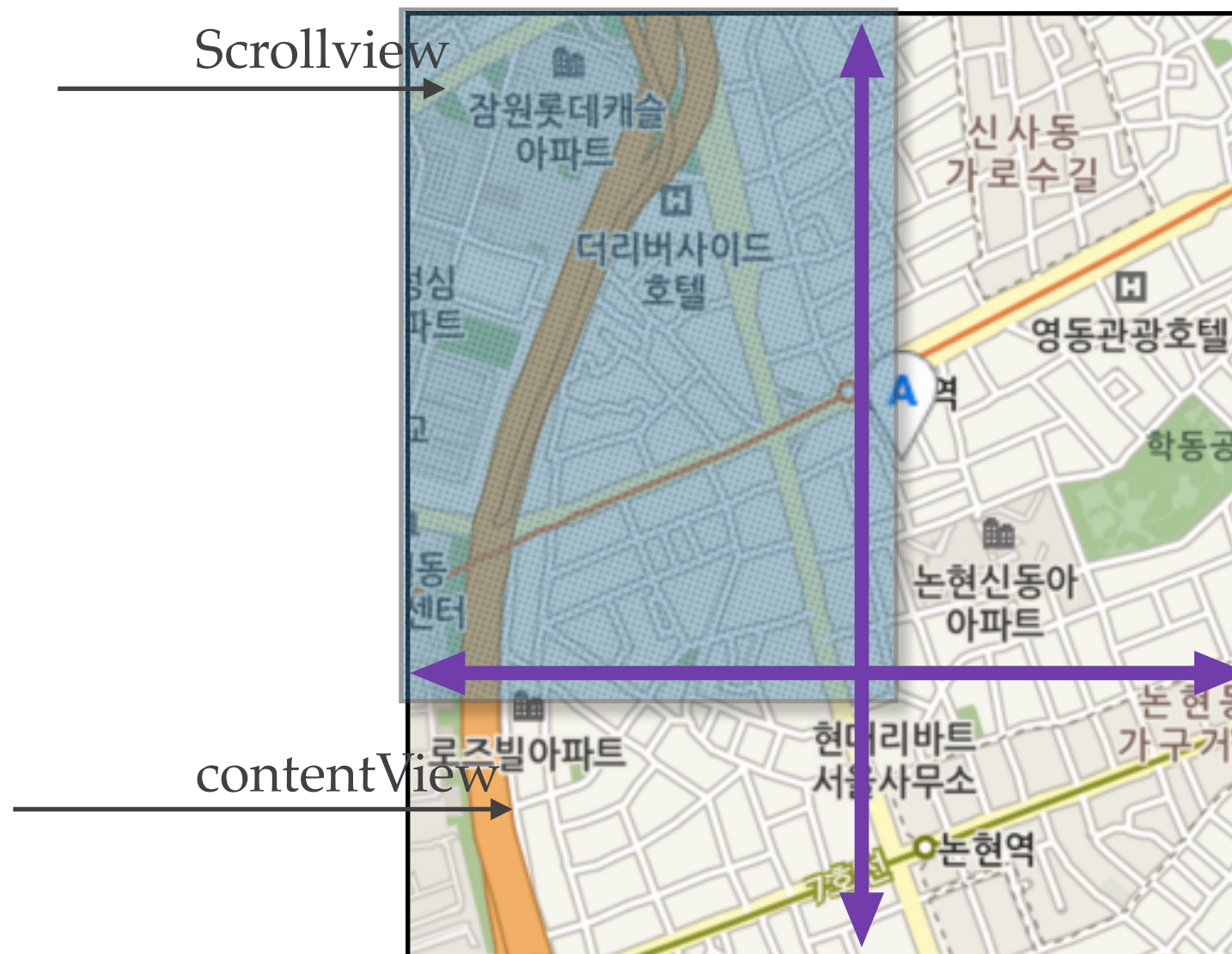
- ViewController입장에선, 내가 구현한 메소드를 실행하진 않지만, customView가 적절한 곳에서 호출했을것이다.
- customView가 특정 위치에서 해당 메소드를 호출할 것을 예상하여 필요한 행동을 구현한다.

UIScrollView

- ViewSize보다 확장된 뷰를 보기위한 View
- UIScrollView에 추가된 View는 ContentView위에 추가 된다.

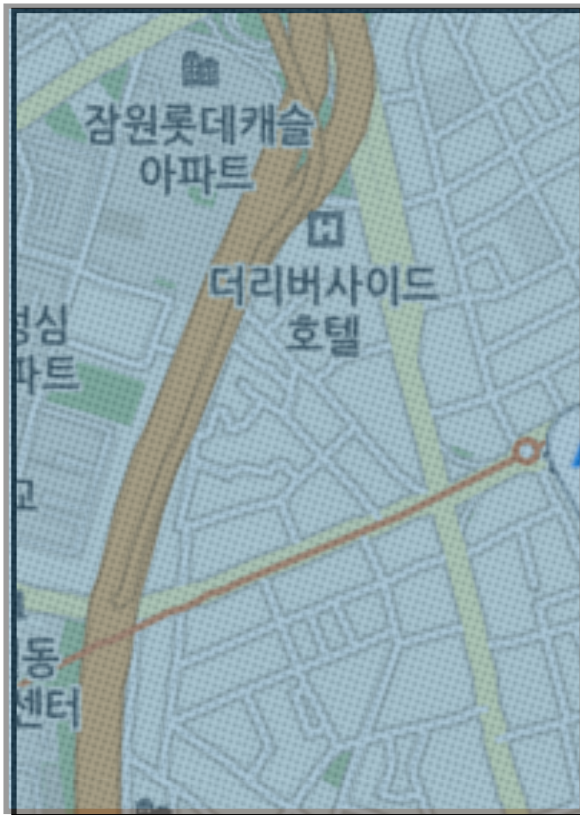


UIScrollView - ContentView



UIScrollView - 스크롤

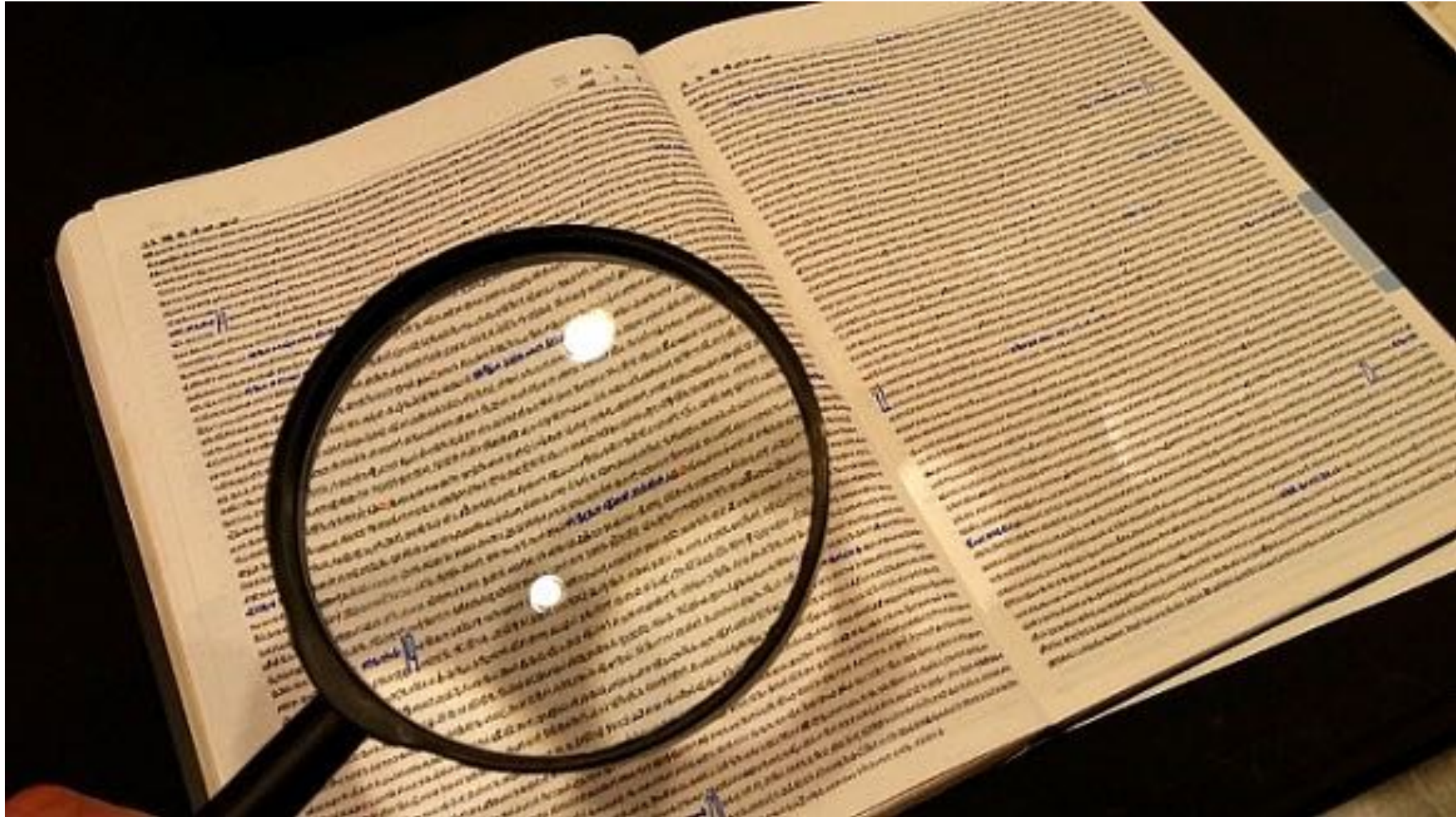
실화면



내부동작



UIScrollView



UIScrollView - 실습

가로 3페이지 스크롤 뷰 만들기(컨텐츠는 이미지뷰)

