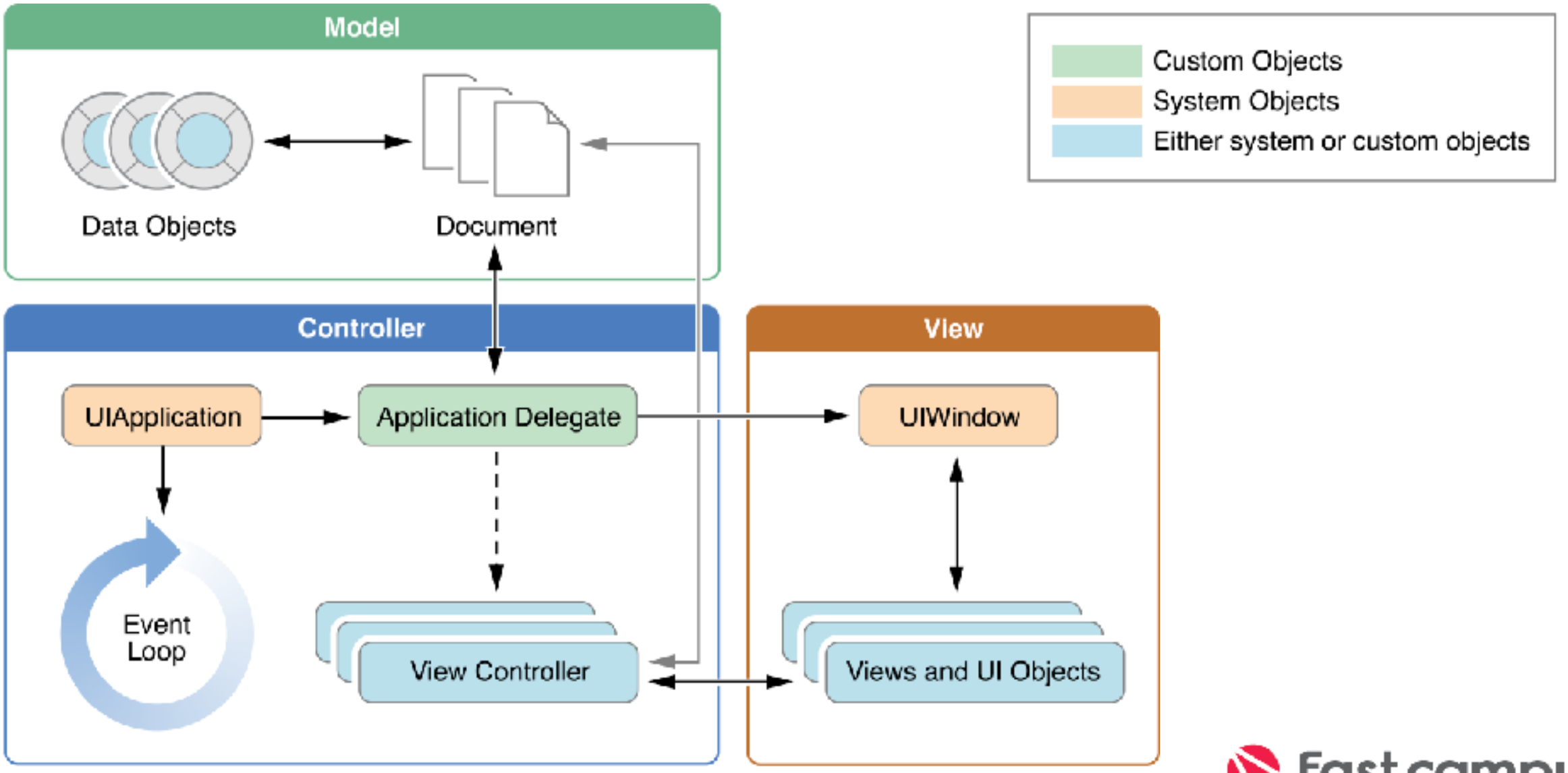
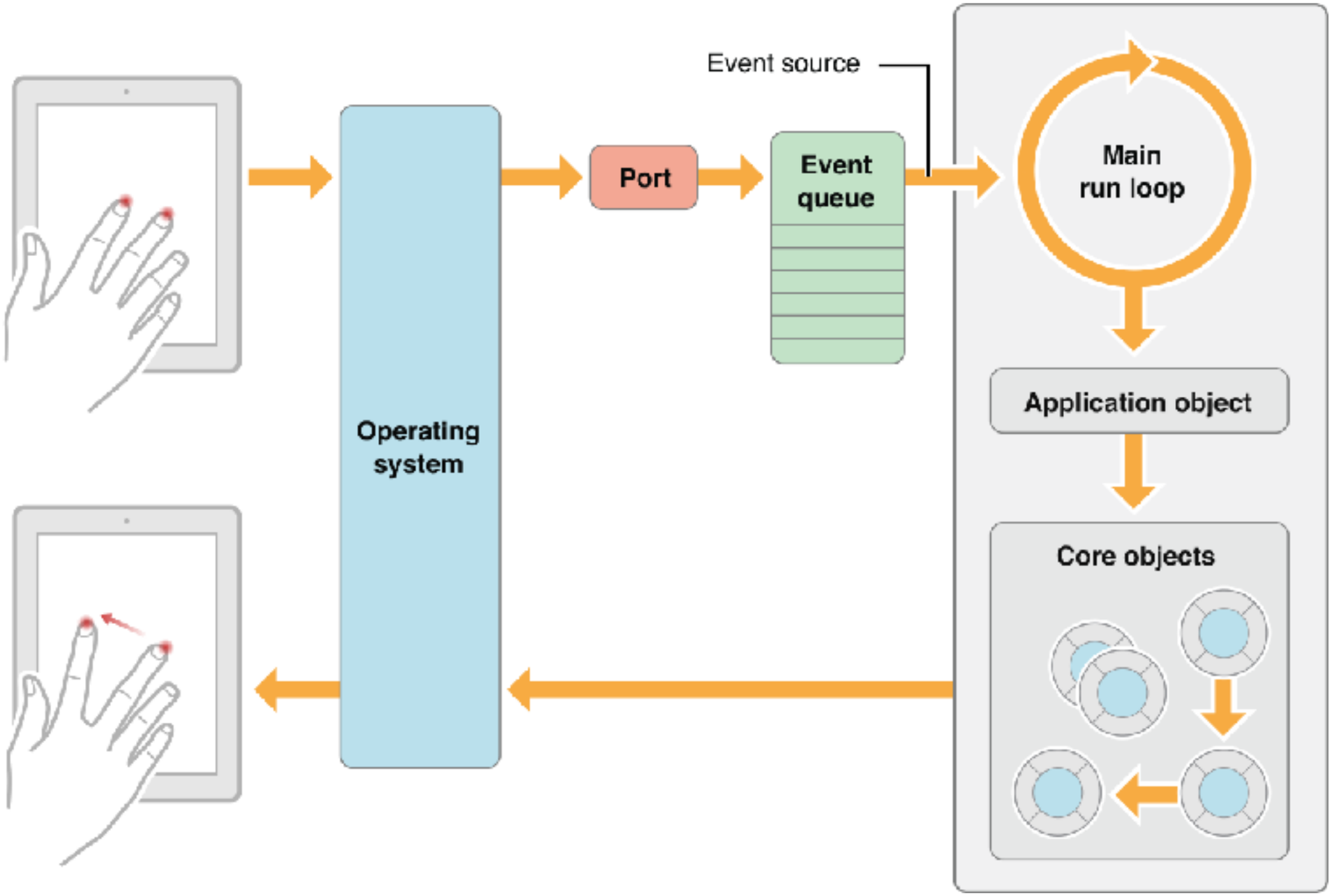
Application Life Cycle

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



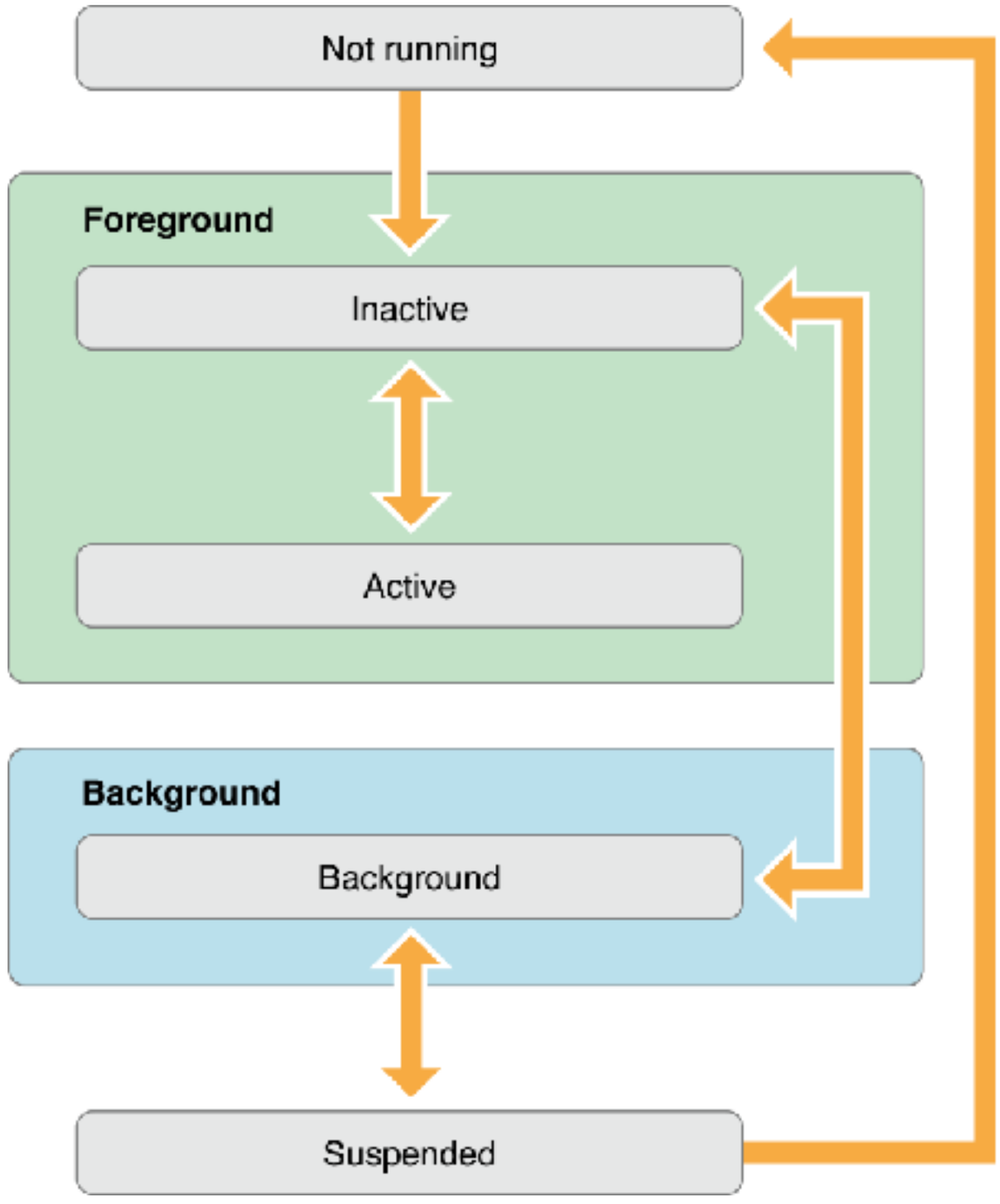
The Main Run Loop



Event에 대한 처리

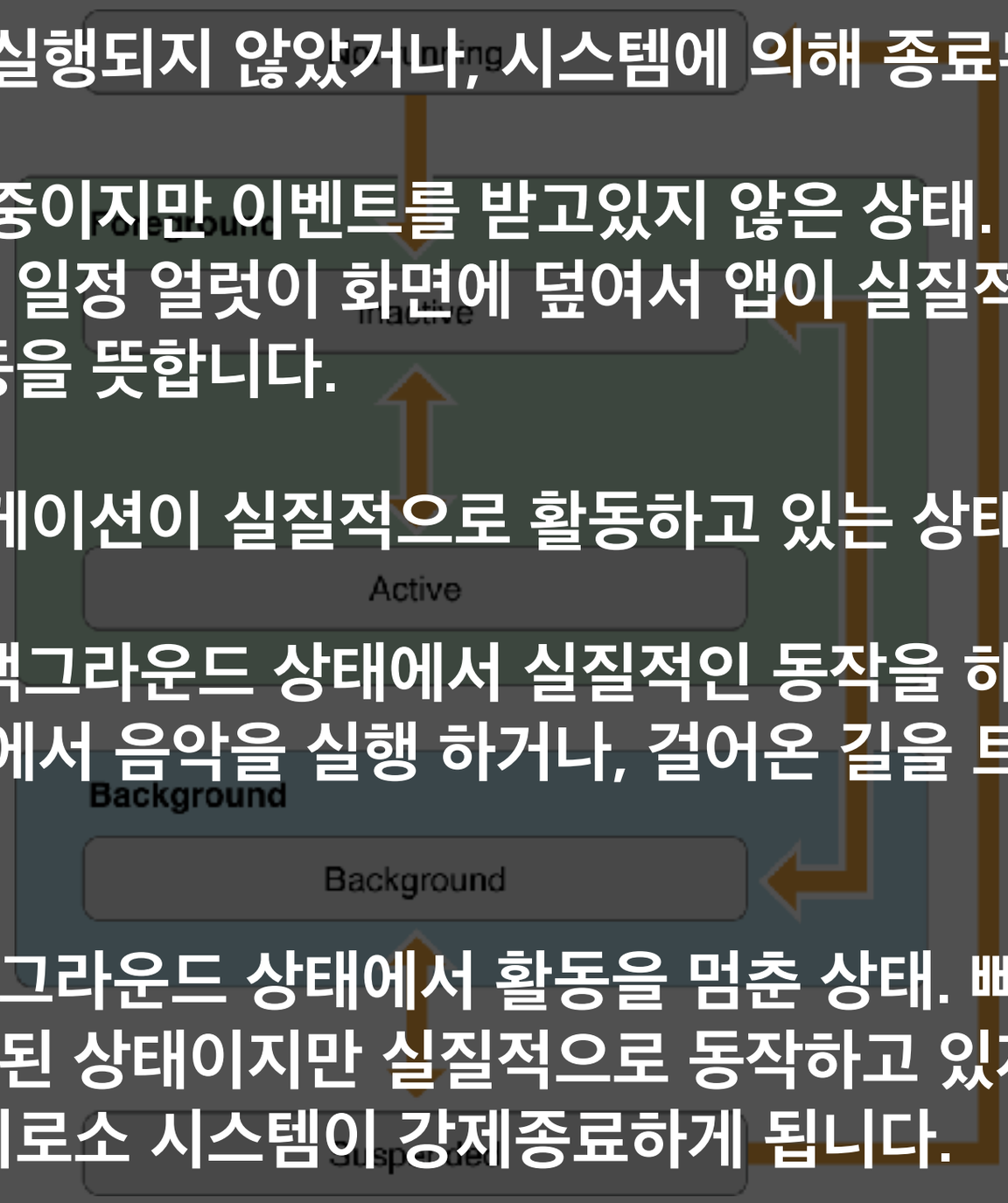
- Touch : 발생된 이벤트에 대한 뷰가 처리
- Remote control & Shake motion events : First responder객체
- Accelerometer/Magnetometer/Gyroscope : 각각의 객체로 전달
- Location :CoreLocation 객체
- Redraw : 업데이트를 원하는 뷰가 처리

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 얼럿이 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.



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 얼럿이 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.

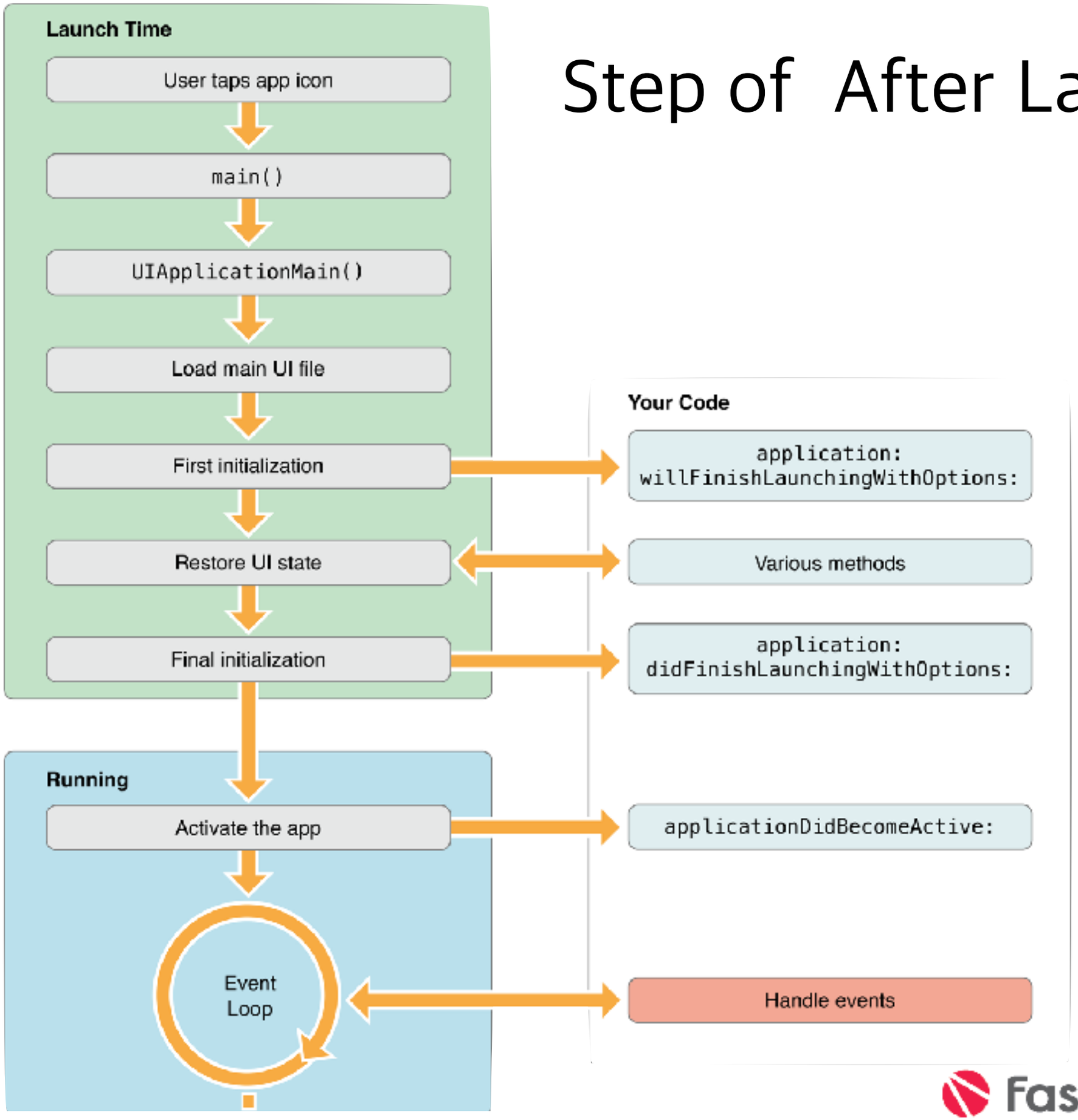
<출력용>

Call to the methods of your app delegate object

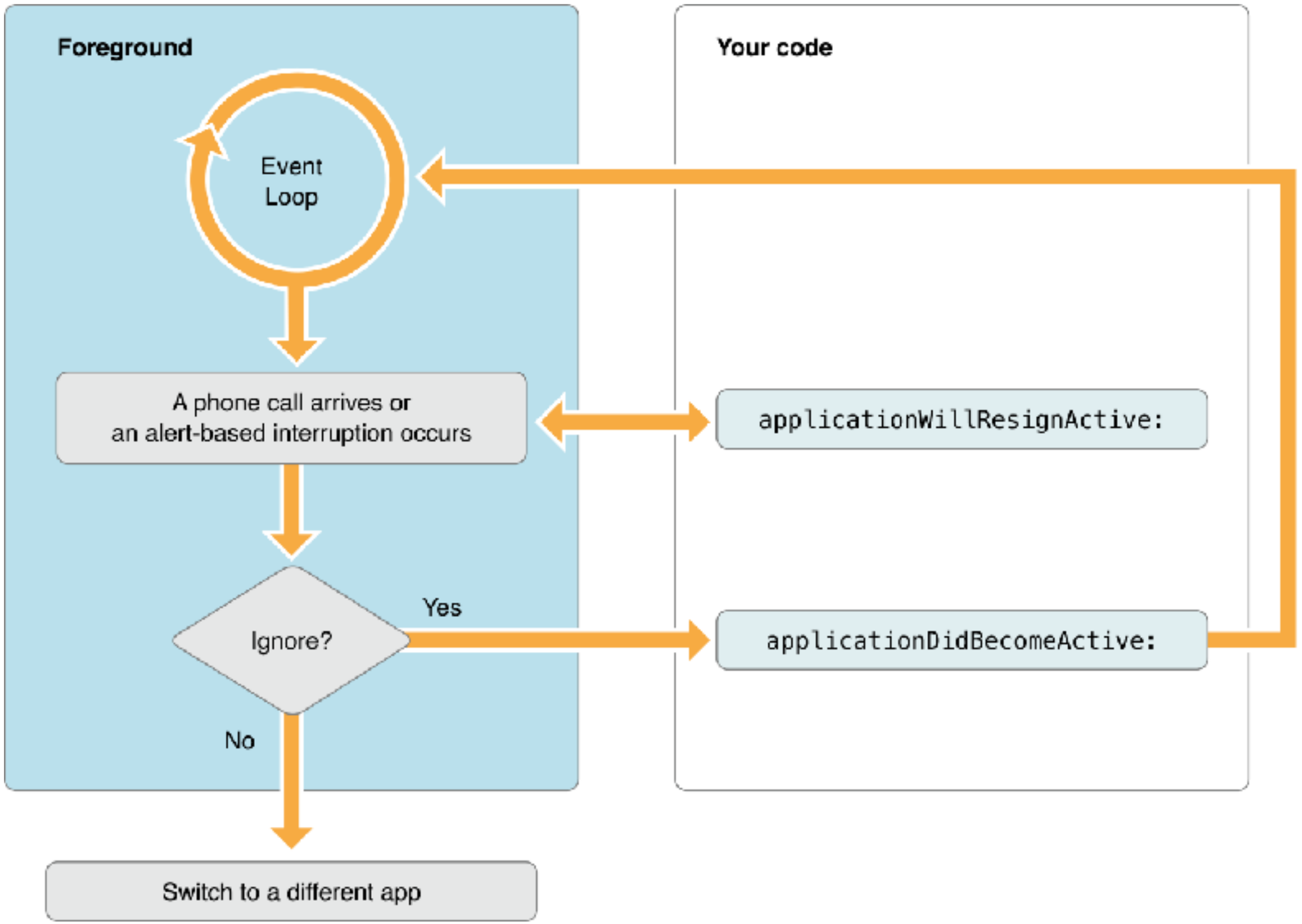
대부분의 상태변화를 app delegate 객체에 호출되는 메소드를 오버라이드하여 알아챌 수 있습니다.

- `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.

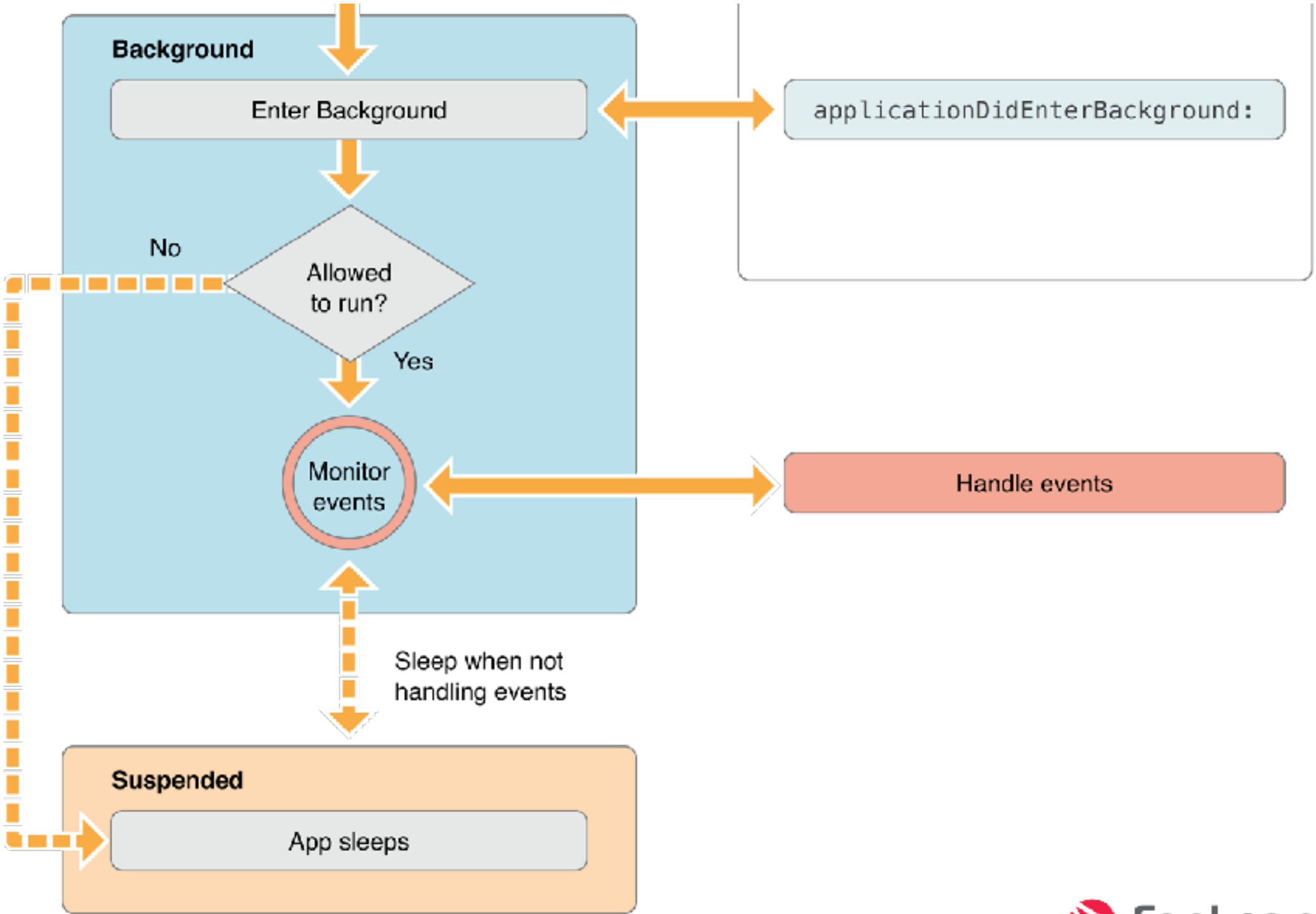
Step of After Launch



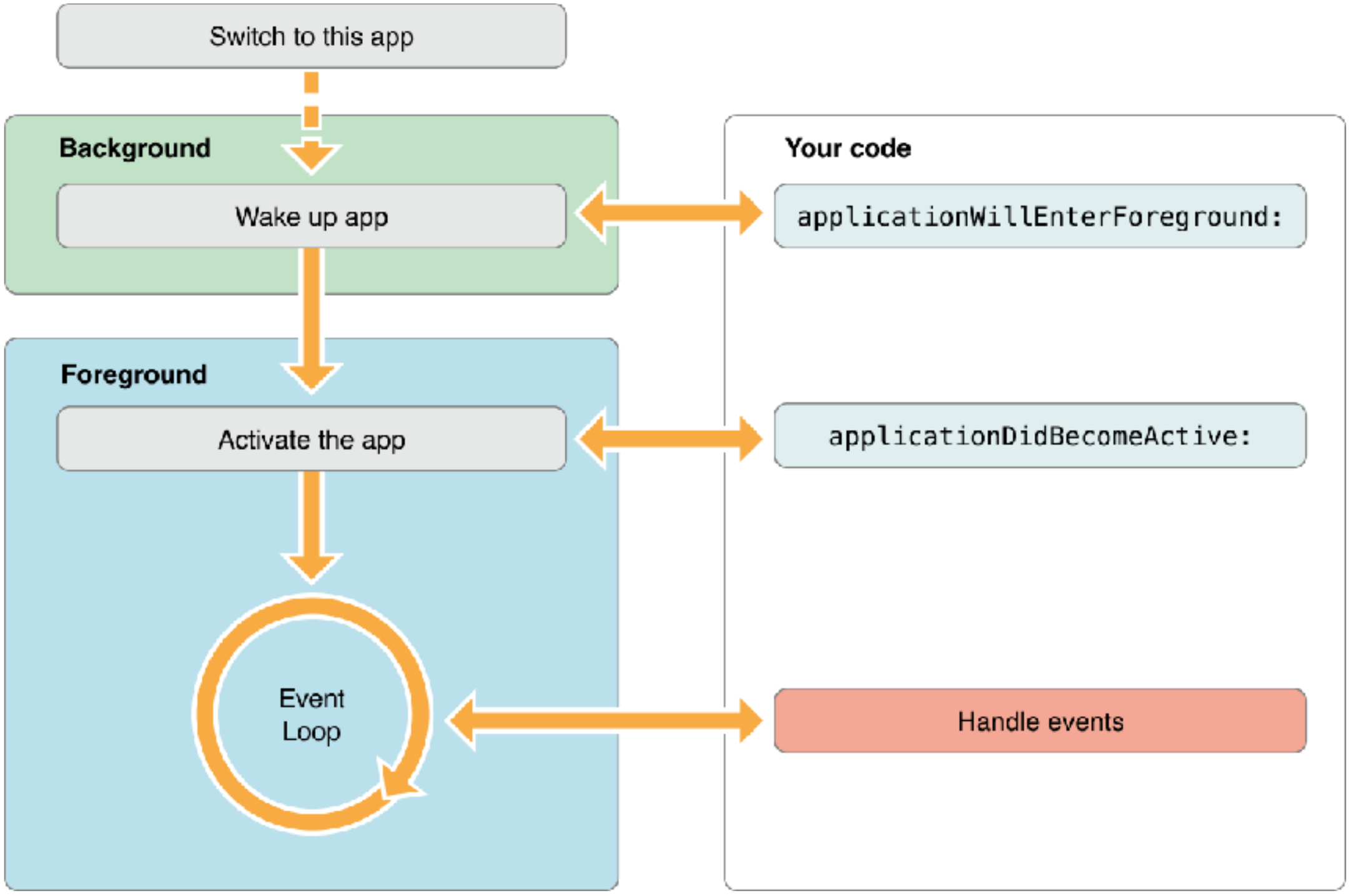
Step of Interruptions



Step of Enter Background



Step of Enter Forground



Supported Background Tasks

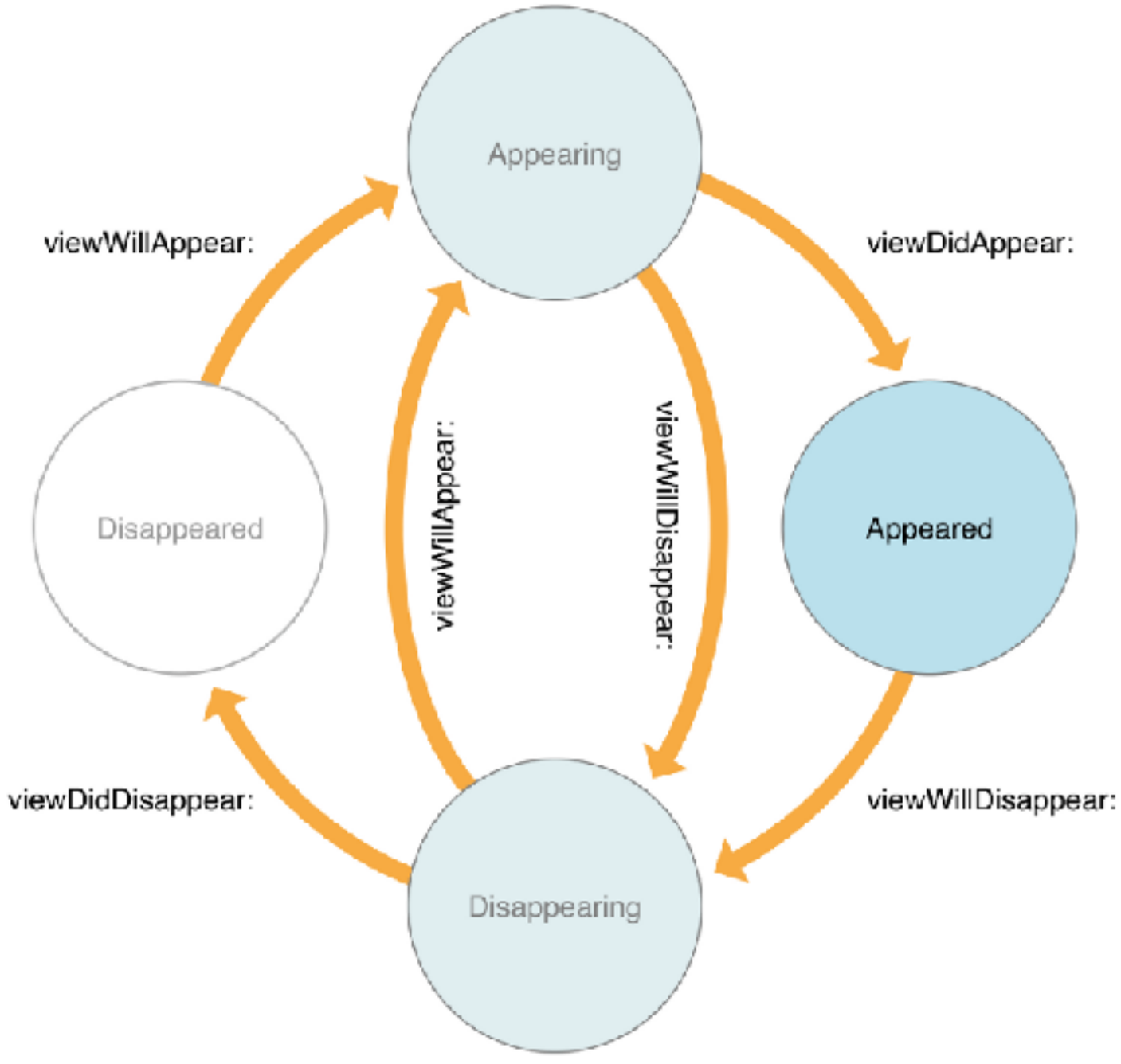
- Audio and AirPlay (음악)
- Location updates (위치 정보)
- Voice over IP (인터넷을 사용한 음성통화)
- Newsstand downloads(뉴스 스탠드 다운로드)
- External accessory communication (기타 하드웨어 액세사리)
- Bluetooth LE accessories (블루투스 액세사리 사용)
- Background fetch (네트워크를 통한 일반적인 다운로드나 미완료된 작업)
- Remote notifications (PushNotification)

확인해 볼까요?

- 실제 로그를 찍어 상태를 확인해 봅시다.

UIViewController의 생명주기 메소드

- 프로그래머가 직접 호출 불가
- 오버라이드 하는 메소드이므로 꼭 해당 메소드 내에서 super.method을 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.

접근 제어

접근 수준

- 외무 모듈에서의 접근을 제어하는 수단.
- 캡슐화, 은닉화를 위해 사용

모듈 & 소스파일

- 모듈 : 배포할 코드의 묶음 단위, 통상 프레임워크나 라이브러리, 어플리케이션이 모듈의 단위가 될수 있다.
- 소스파일 : 하나의 스위프트 소스코드 파일

접근제어

- Open (개방 접근수준) : 모듈 외부까지 접근 가능
- public (공개 접근수준) : 모듈 외부까지 접근 가능
- internal (내부 접근수준) : 모듈 내부에서 접근가능, 기본 지정값
- fileprivate (파일외 비공개) : 파일 내부에서만 접근가능
- private (비공개) : 기능 정의 내부에서만 가능

Open VS Public

- Open을 제외한 다른 모든 접근수준의 클래스는 그 클래스가 정의된 모듈 안에서만 상속될 수 있다.

- Open을 제외한 다른 모든 접근수준의 클래스 멤버는 그 멤버가 정의된 모듈 안에서만 재정의 될 수 있다.

- Open 수준의 클래스는 그 클래스가 정의된 모듈 밖의 다른 모듈에서도 상속되고, 재정의 될수 있다.

- 클래스를 Open으로 명시하는 것은 그 클래스를 다른 모듈에서도 부모클래스로 사용할수 있다는 얘기

예시

```
public class SomePublicClass {
    public var somePublicProperty = 0
    var someInternalProperty = 0
    fileprivate func someFilePrivateMethod() {}
    private func somePrivateMethod() {}
}

class SomeInternalClass {
    var someInternalProperty = 0
    fileprivate func someFilePrivateMethod() {}
    private func somePrivateMethod() {}
}

fileprivate class SomeFilePrivateClass {
    func someFilePrivateMethod() {}
    private func somePrivateMethod() {}
}

private class SomePrivateClass {
    func somePrivateMethod() {}
}
```

접근수준 확인하기

- Test클래스 생성
- Public, internal, fileprivate, private 접근 수준을 포함한 메소드 만들기
- 각각의 상황에서 메소드 호출 해보기

클래스와 구조체

Classes & Structures

"*Classes* and *structures* are general-purpose, flexible constructs that become the building blocks of your program's code.
You define properties and methods to add functionality to your classes and structures by using exactly the same syntax as for constants, variables, and functions."

Classes & Structures

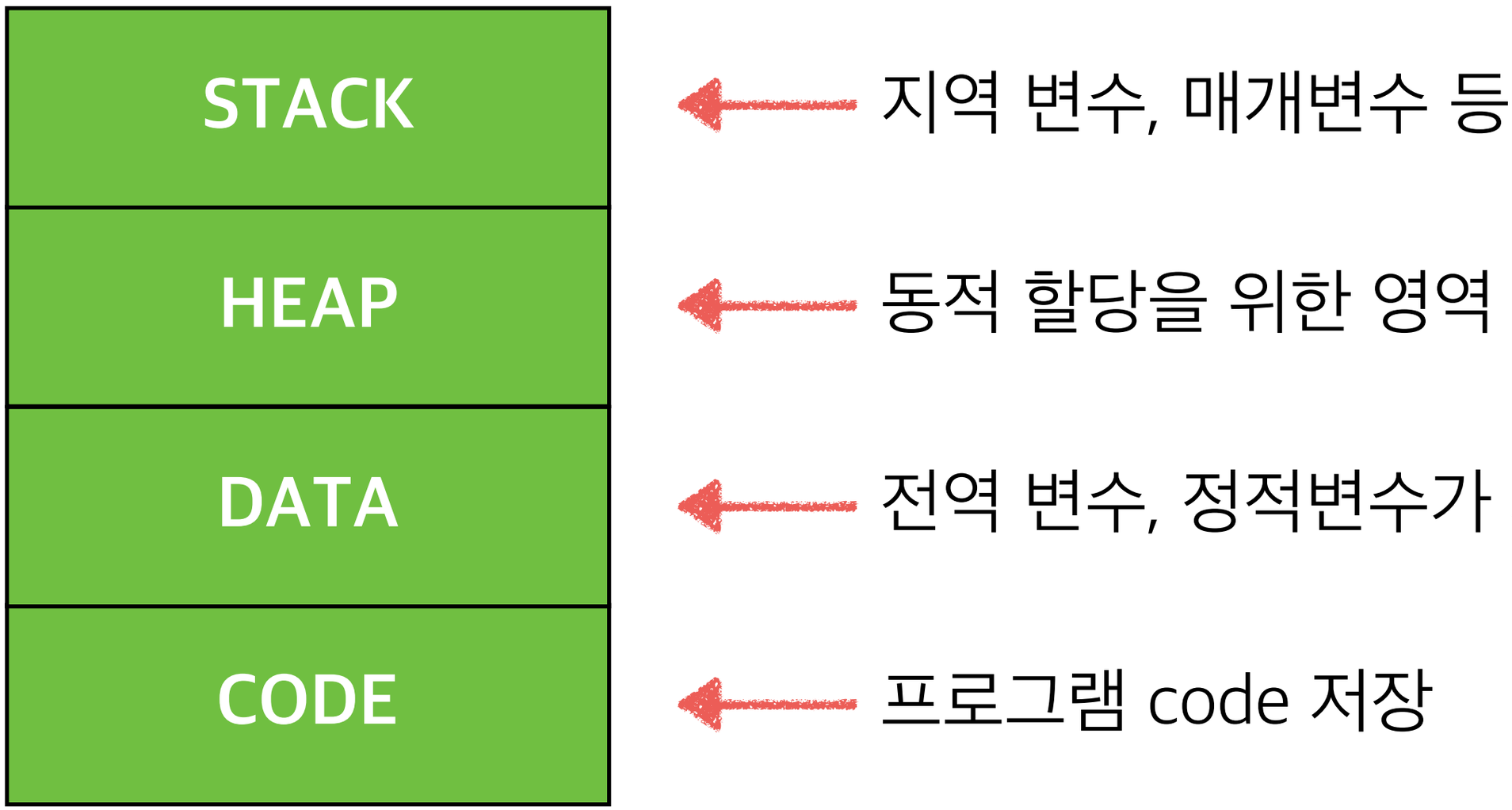
- 프로그램 코드 블럭의 기본 구조이다.
- 변수, 상수, 함수를 추가 할수 있다. (두 구조의 문법 같음)
- 단일 파일에 정의 되며 다른 코드에서 자동으로 사용 할수 있습니다.(접근 제한자에 따라 접근성은 차이가 있다. internal 기본 접근제한자)
- 초기 상태를 설정하기 위해 initializer가 만들어 지고, 사용자가 추가로 정의할 수 있다.
- 사용 시 인스턴스(instance)라고 불린다.
- 기본 구현된 내용에 기능을 더 추가해서 확장 할수 있다. (Extensions)
- 프로토콜을 상속받아 사용할수 있다. (Protocols)

Classes VS Structures

- Class는 참조 타입이며, Structure는 값 타입이다.
- Class는 상속을 통해 부모클래스의 특성을 상속받을수 있다.
- Class는 Type Casting을 사용할수 있다.

링크로 데이터 접근하기(pointer)

Memory구조

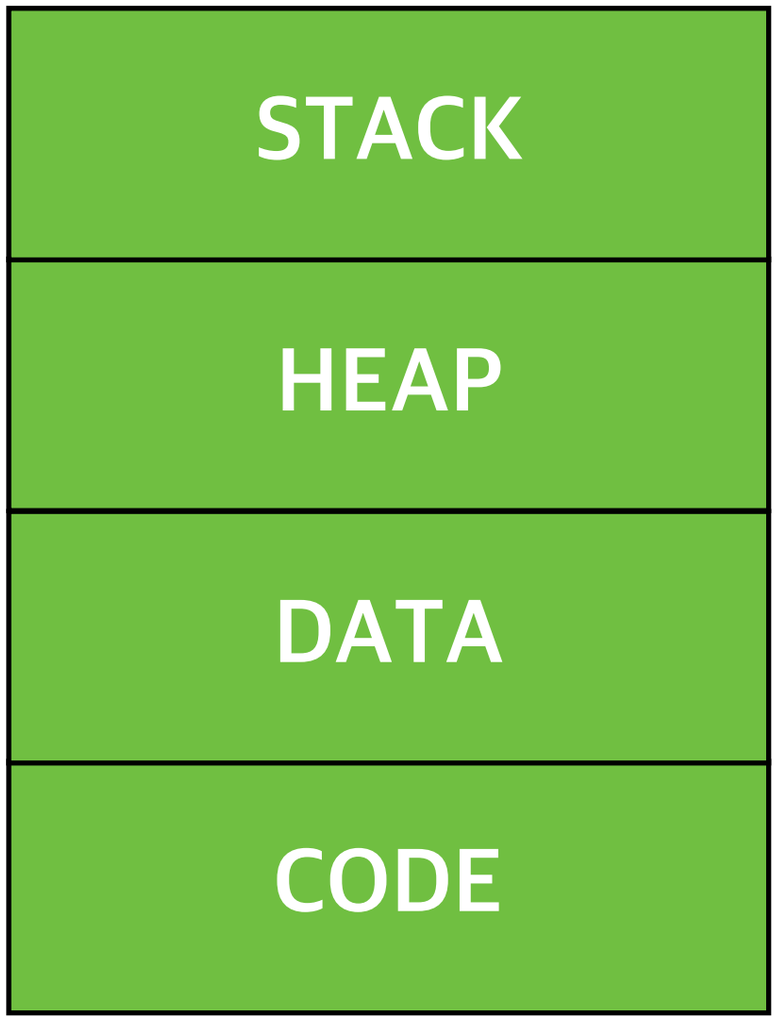


Memory구조 파악하기

다음 코드가 메모리에 어떻게 들어 갈까요?

```
var num:Int = 4
var num2:Int = 5
```

Memory구조 파악하기



a = 4, b = 5

프로그램 code 저장

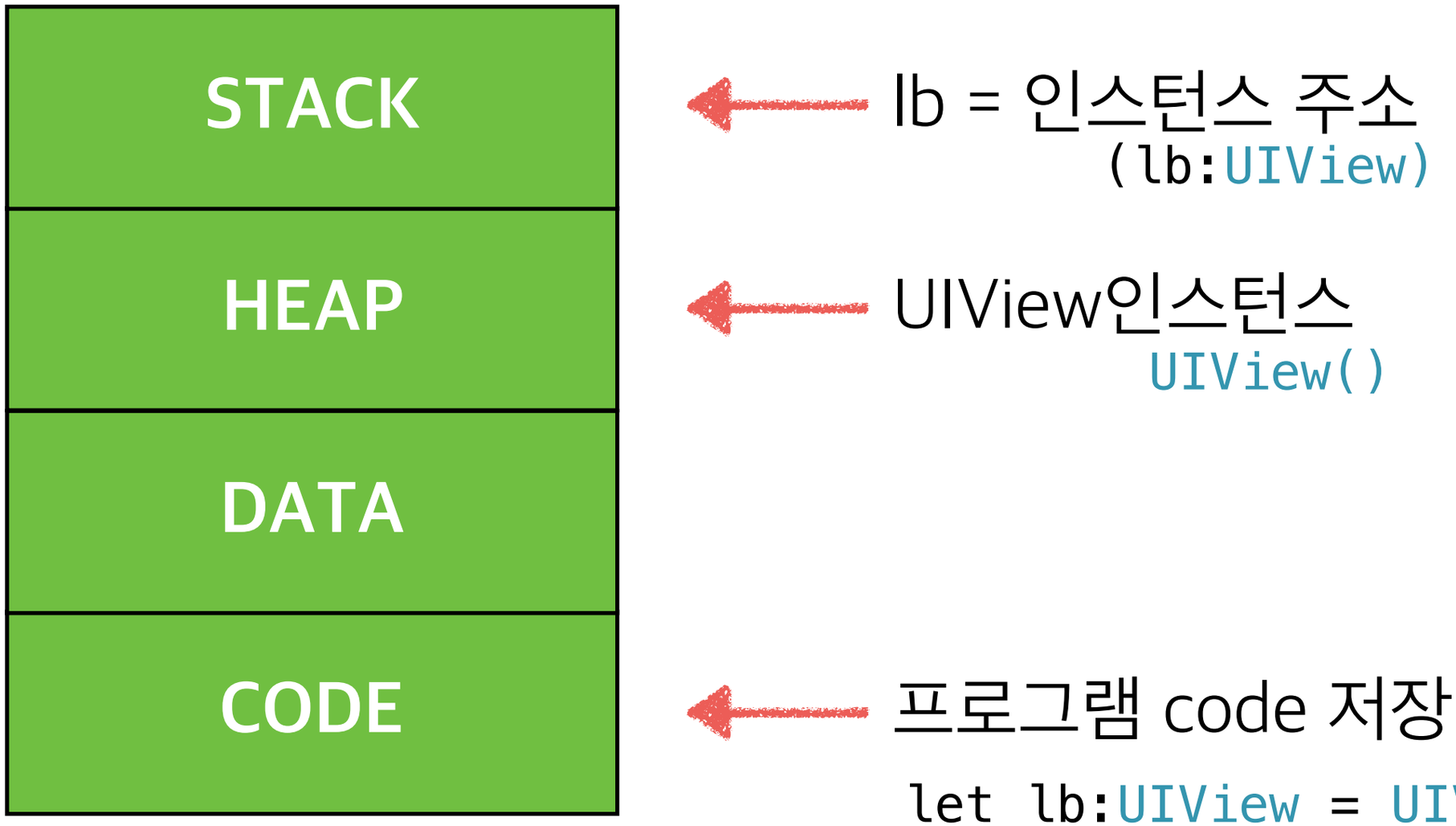
```
var num:Int = 4;
var num2:Int = 5;
```

Memory구조 파악하기

다음 코드가 메모리에 어떻게 들어 갈까요?

```
let lb:UIView = UIView()
```

Memory구조 파악하기

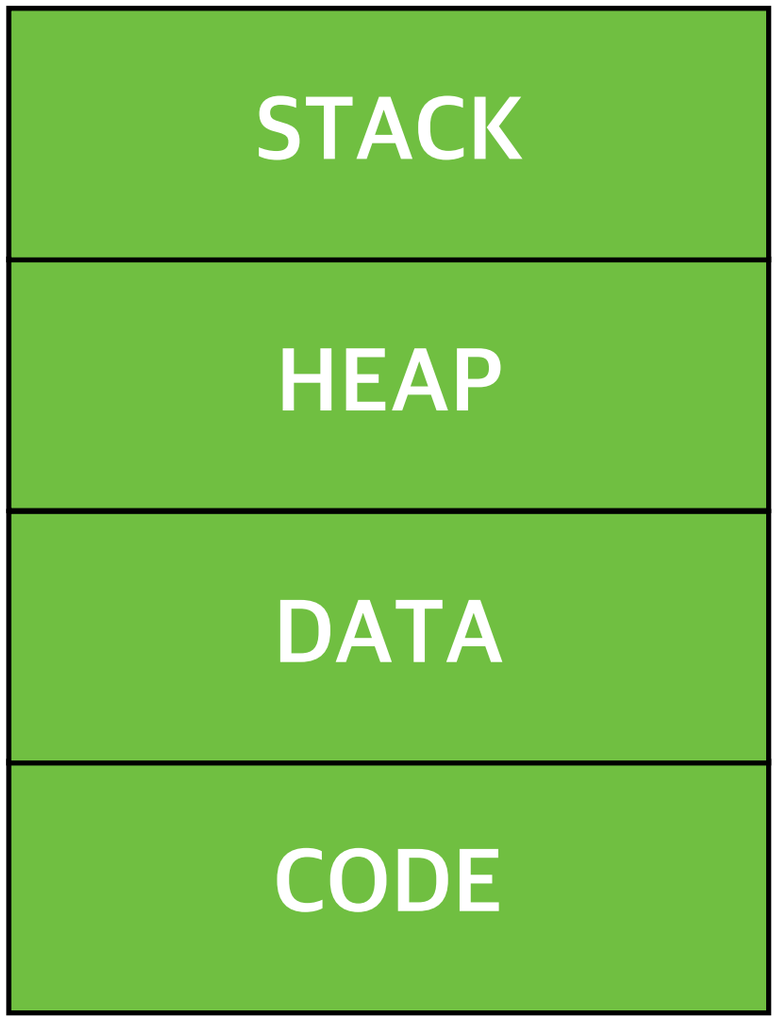


Memory구조 파악하기

다음 코드가 메모리에 어떻게 들어 갈까요?

```
static let number:Int = 5
```

Memory구조 파악하기



number = 5

프로그램 code 저장

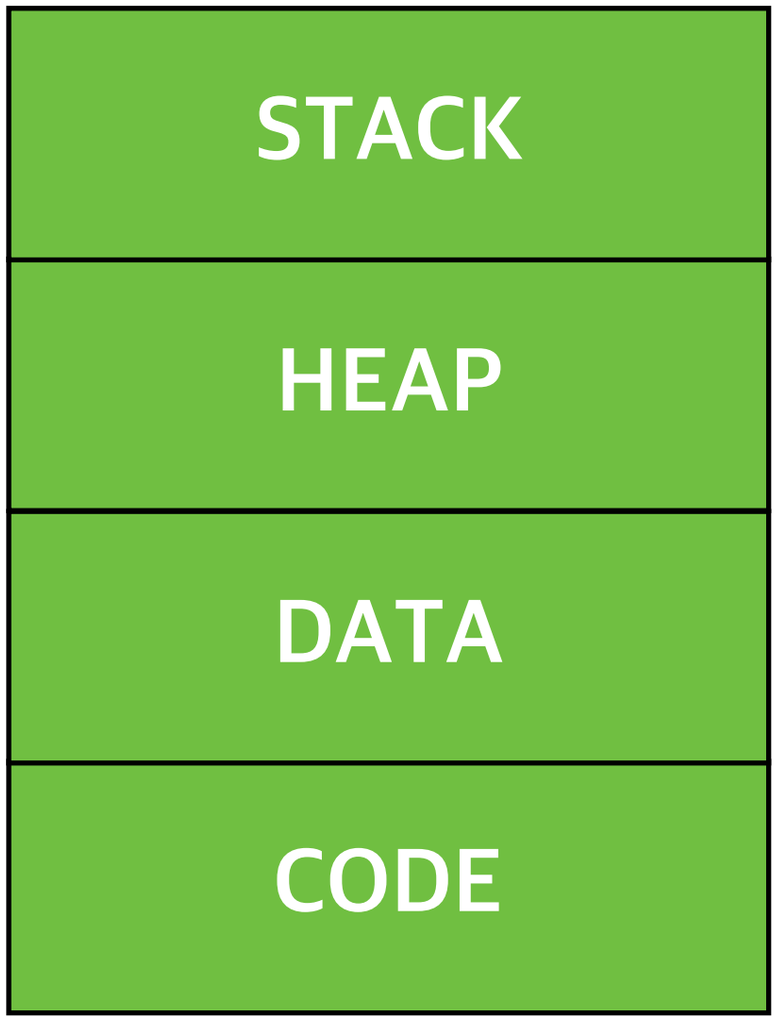
```
static let number:Int = 5
```

Memory구조 파악하기

다음 코드가 메모리에 어떻게 들어 갈까요?

```
func sumTwoNumber(num1:Int, num2:Int) -> Int
{
    return num1 + num2
}
```

Memory구조 파악하기



num1 = 3, num2 = 4

```
sumTooNumber(num1:3, num2:4)
```

프로그램 code 저장

```
func sumTwoNumber(num1:Int, num2:Int)
-> Int {
      return num1 + num2
}
```

Fast campus

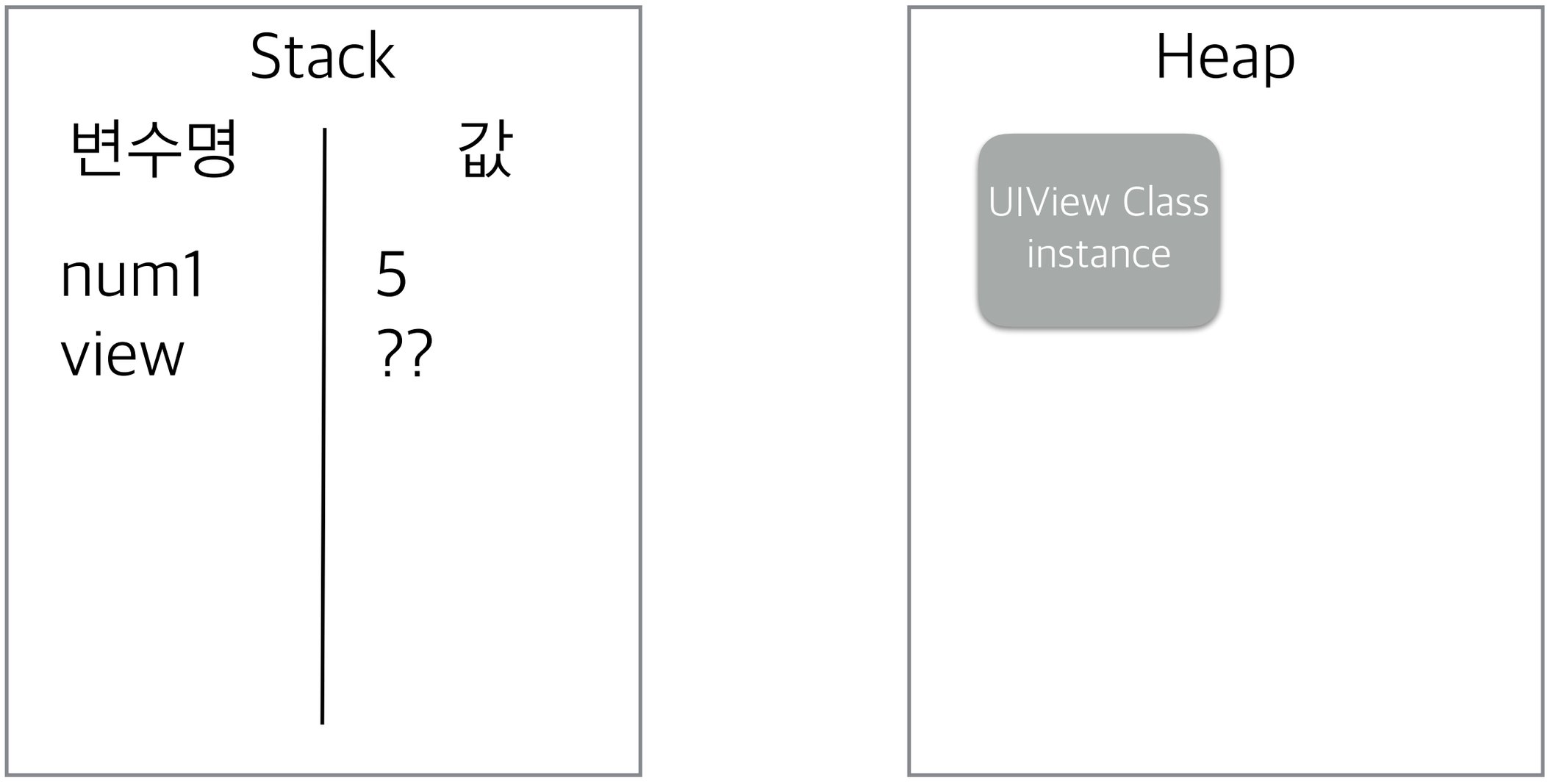
두 코드의 차이점

```
let num2:Int = 5
```

```
let lb:UIView = UIView()
```

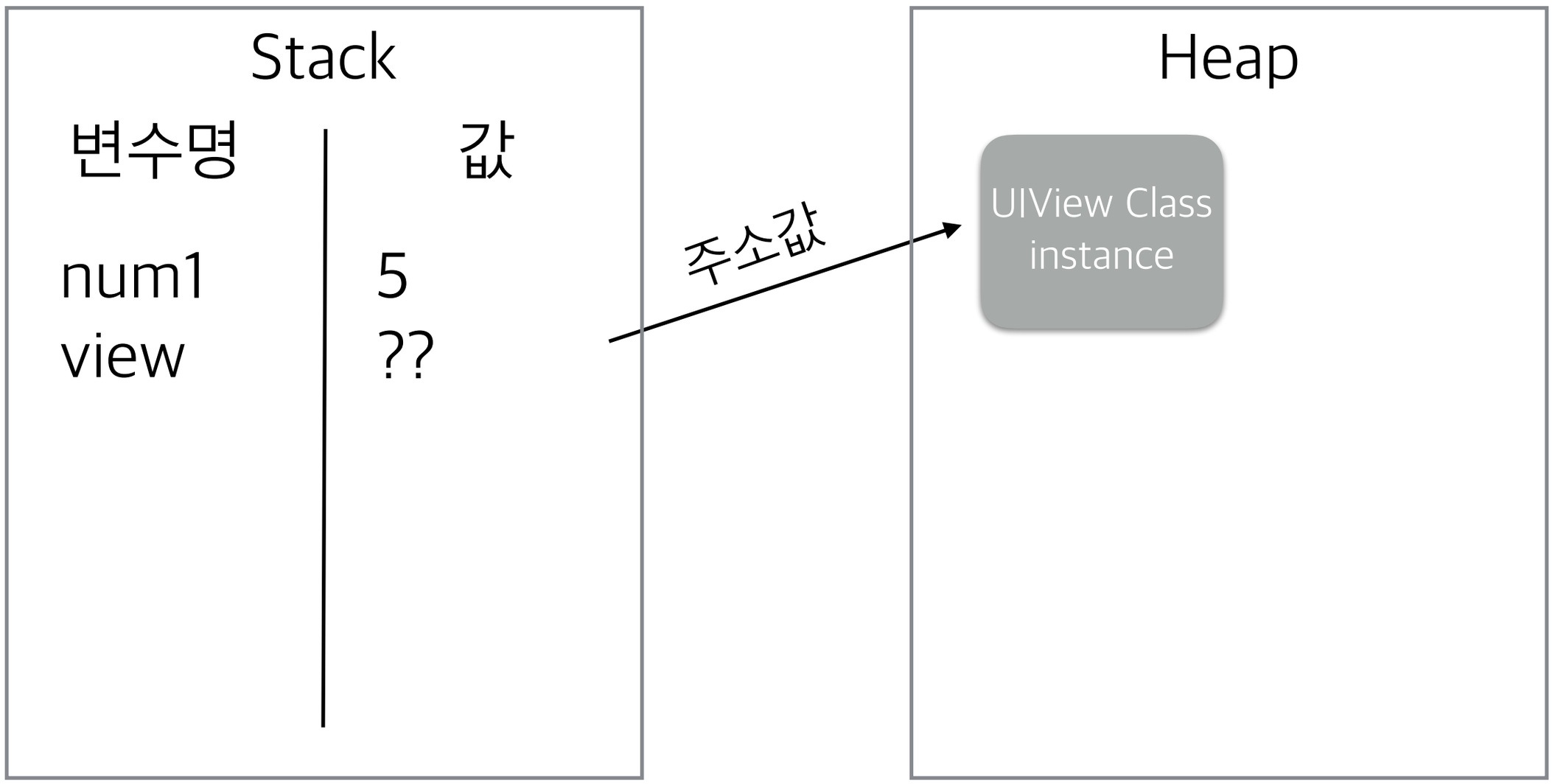
두 코드의 차이점

```
let num1:Int = 5
let view:UIView = UIView()
```



두 코드의 차이점

```
let num1:Int = 5
let view:UIView = UIView()
```



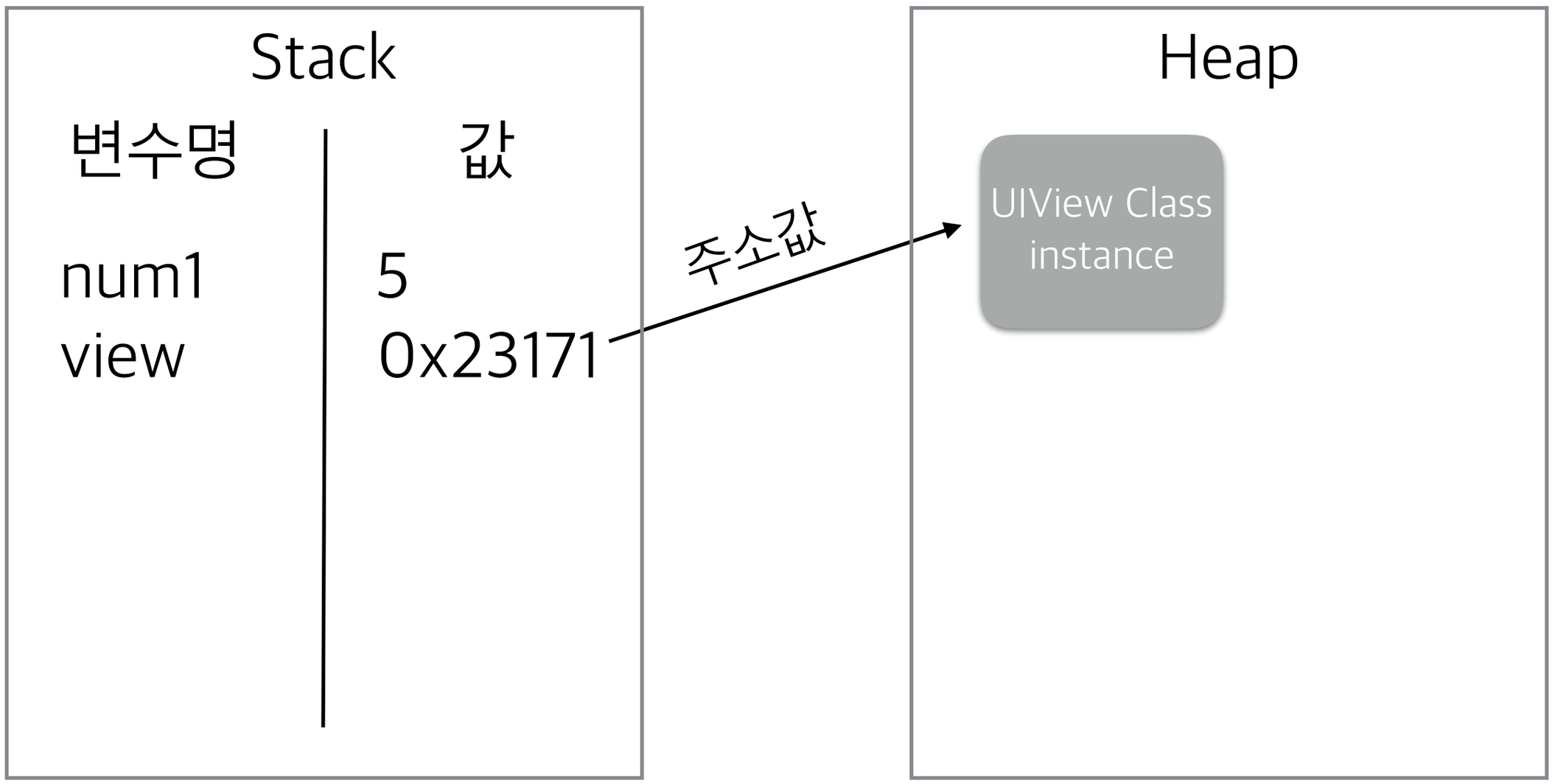
Pointer

- 포인터(pointer)는 프로그래밍 언어에서 다른 변수, 혹은 그 변수의 메모리 공간주소를 가리키는 변수를 말한다.

-Wikipedia-

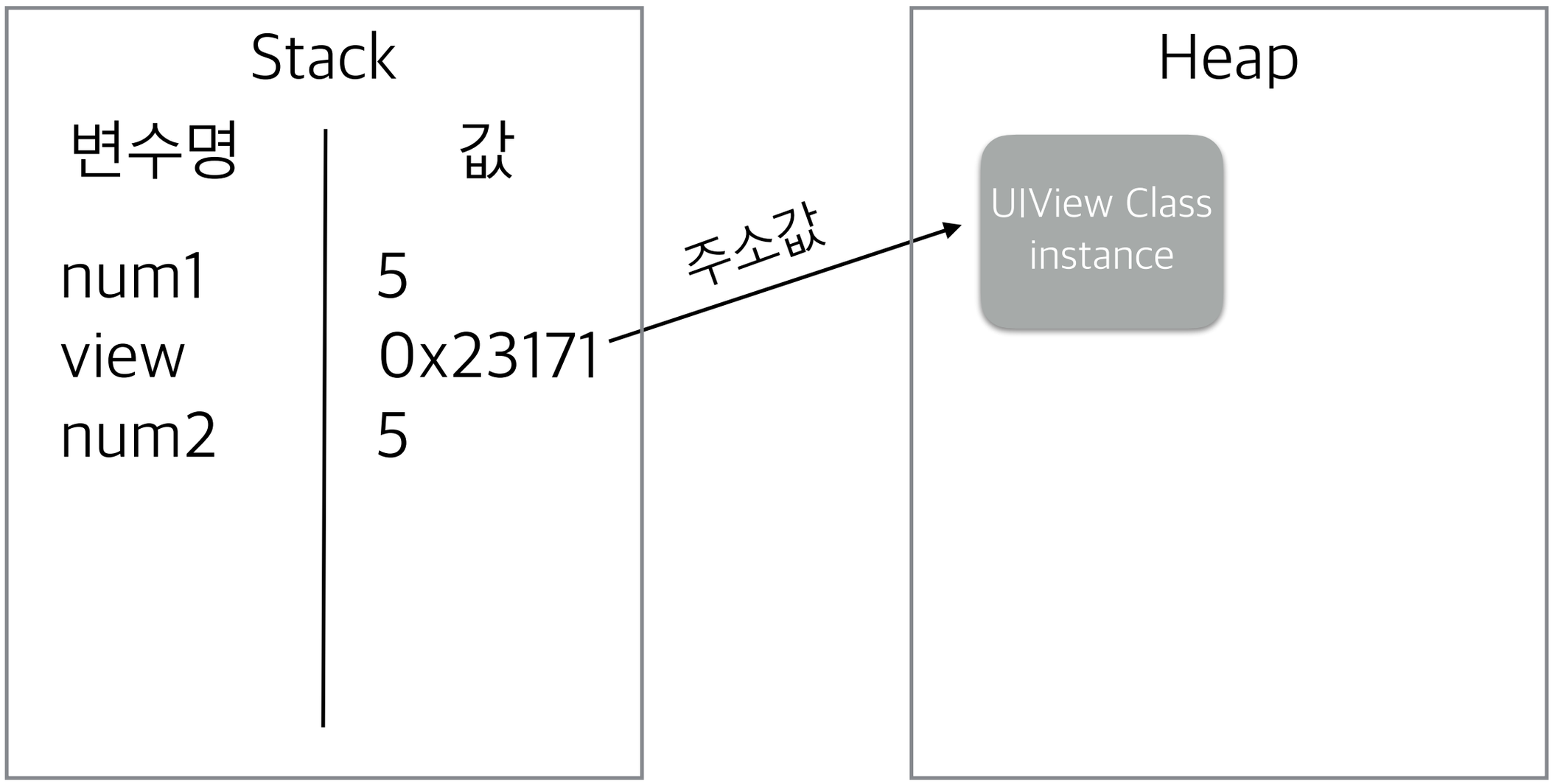
Pointer

```
let num1:Int = 5
let view:UIView = UIView()
```



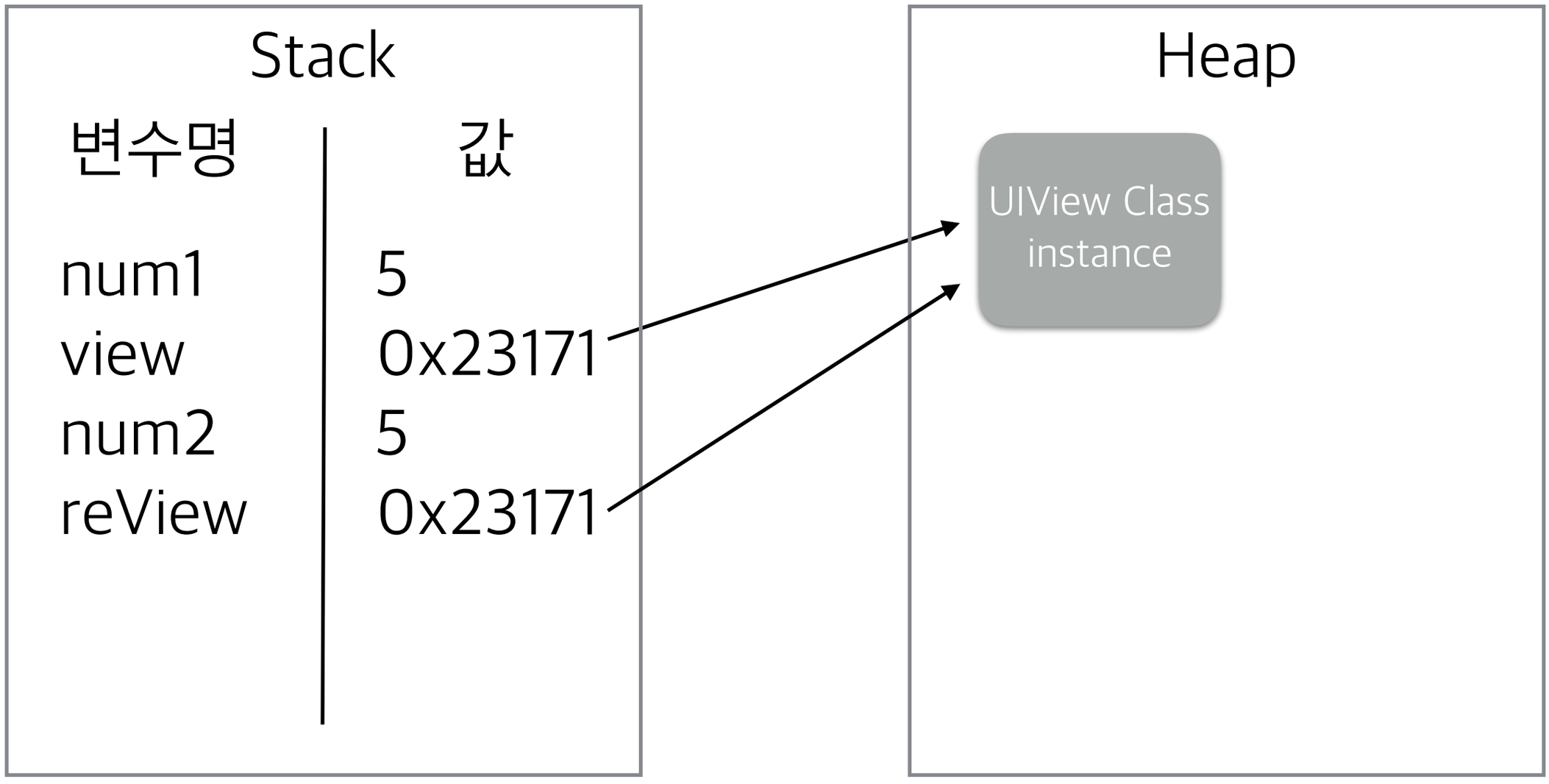
Pointer

```
let num1:Int = 5
let view:UIView = UIView()
let num2 = num1
```



Pointer

```
let view:UIView = UIView()
let num2 = num1
let reView:UIView = view
```



결과값은?

```
let view:UIView = UIView()
let reView:UIView = view
view.backgroundColor = .red
//현재 view의 배경색은? reView의 배경색은?

reView.backgroundColor = .gray
//현재 view의 배경색은? reView의 배경색은?
```

기본 구조

```
class SomeClass {
    // class definition goes here
}
```

```
struct SomeStructure {
    // structure definition goes here
}
```

명명규칙

- 시스템 예약어는 사용할 수 없다.
- 숫자는 이름으로 시작될 수는 없지만 이름에 포함될 수 있다.
- 공백을 포함 할 수 없다.
- 변수 & 함수명을 lowerCamelCase,
  클래스 명은 UpperCamelCase로 작성한다.

기본 구조

```
struct Resolution {
    var width = 0
    var height = 0
}
```

```
class VideoMode {
    var resolution = Resolution()
    var interlaced = false
    var frameRate = 0.0
    var name: String?
}
```

인스턴스

```
let someResolution = Resolution()
let someVideoMode = VideoMode()
```

Properties 접근

```
print("Width of Resolution is \(someResolution.width)”)
print("Width of VideoMode is \(someVideoMode.resolution.width)”)

someVideoMode.resolution.width = 1280
```

Initialization

Initialization is the process of preparing an instance of a class, structure, or enumeration for use.

초기화

- 인스턴스에 설정된 속성의 초기값을 설정과 초기화하는데 목적이 있다.
- 클래스 및 구조체는 인스턴스로 만들어 질때 프로퍼티는 적절한 초기값으로 모두 초기화 해야 한다.
- 모든 구조체는 자동으로 Memberwise Initializers가 만들어 진다.

base Initializers

```
init() {
    // perform some initialization here
}
```

```
struct Fahrenheit {
    var temperature: Double
    init() {
        temperature = 32.0
    }
}

var f = Fahrenheit()
```

Identity Operators

- 동일한 인스턴스인지 확인하는 연산자

Identity 연산자	예제	설명
===	person2 === person1	person1과 person2는 같은 인스턴스를 참조하고 있다.
!==	person2 !== person1	person1과 person2는 다른 인스턴스를 참조하고 있다.

Classes VS Structures

- 어떤걸 선택해서 써야할까요?
- 기본 SDK에 클래스와 구조체의 예제를 찾아봅시다.