20秋

MOSAD

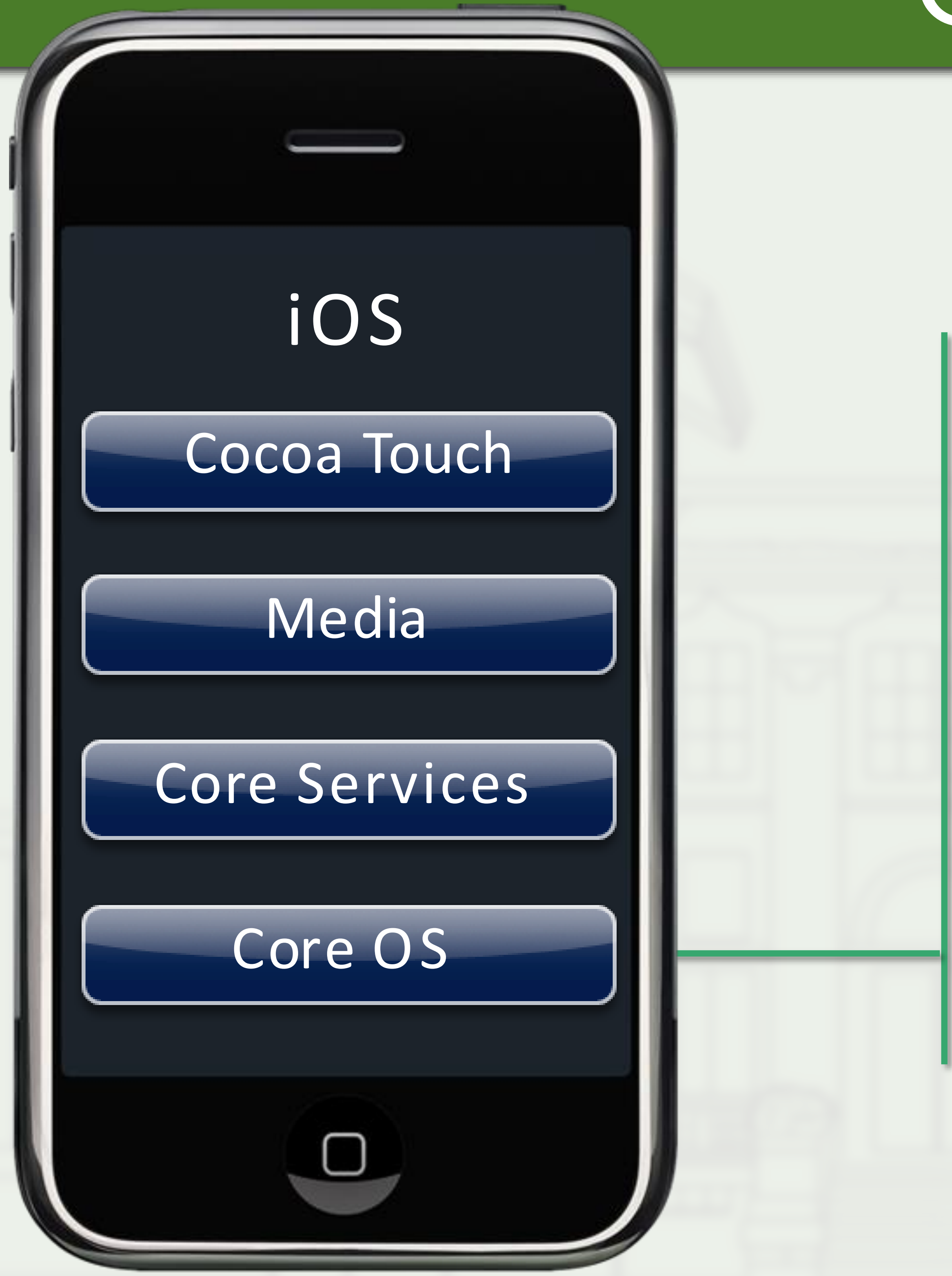
现代操作系统应用开发

#4 iOS平台概述

Content

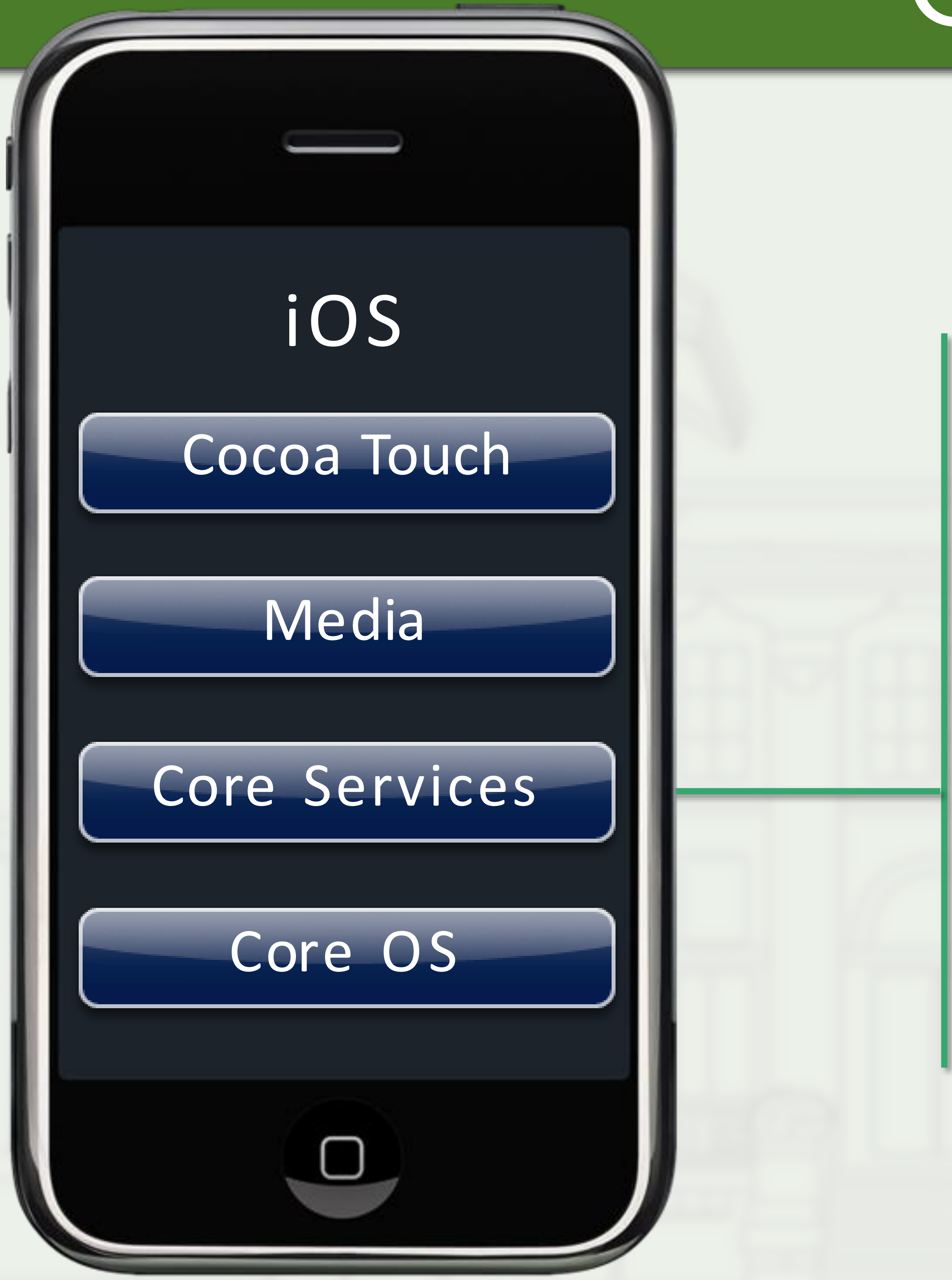
iOS概述

Core OS核心操作系统层



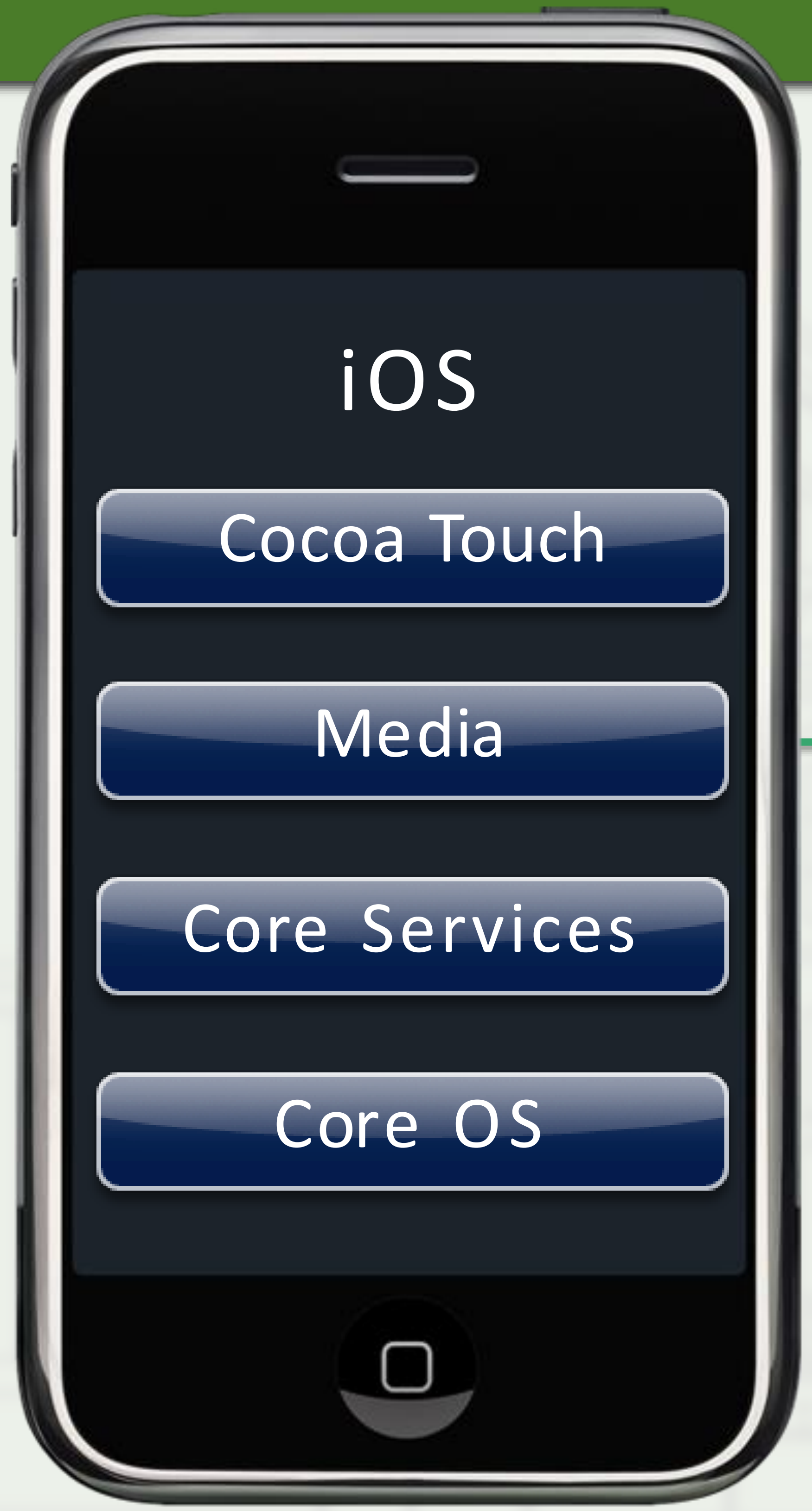
OSX Kernel	Power Management
Mach 3.0	Keychain Access
BSD	Certificates
Sockets	File System
Security	Bonjour

Core Services核心服务层



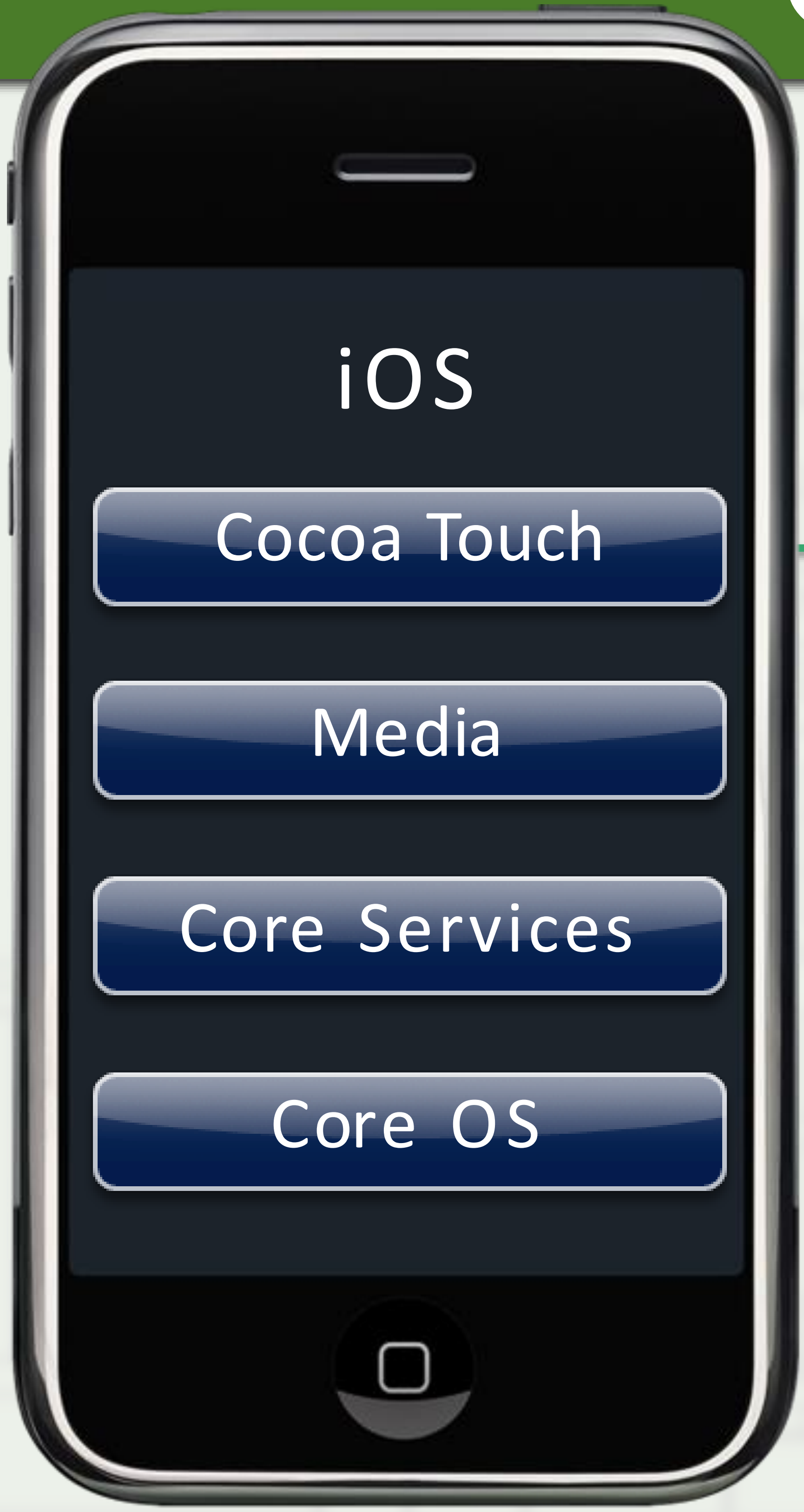
Collections	Core Location
Address Book	Net Services
Networking	Threading
File Access	Preferences
SQLite	URL Utilities

Media媒体层



Core Audio	JPEG, PNG, TIFF
OpenAL	PDF
Audio Mixing	Quartz (2D)
Audio Recording	Core Animation
Video Playback	OpenGL ES

Cocoa Touch界面交互框架层



Multi-Touch	Alerts
Core Motion	Web View
View Hierarchy	Map Kit
Localization	Image Picker
Controls	Camera

开发环境

开发工具

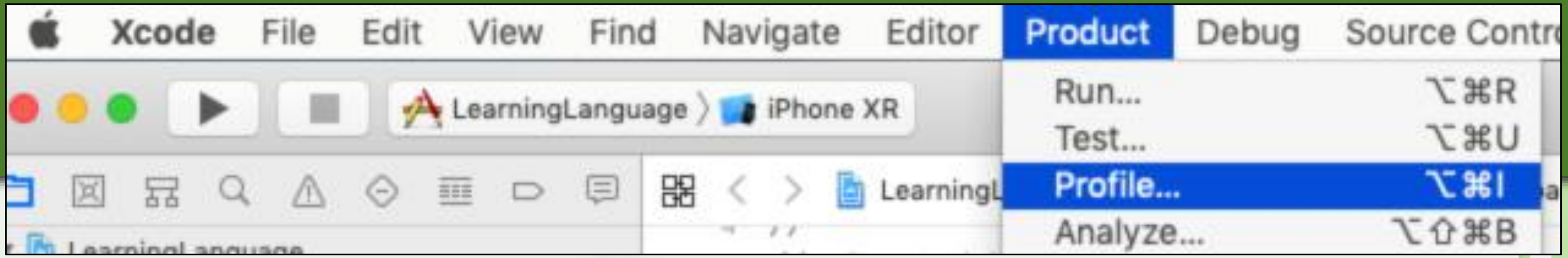
- Xcode：运行于Mac上的集成开发工具

  主要用于开发iOS应用和Mac应用

  能力包括界面布局设计、编码、运行模拟器、调试以及性能检测等
- HomeBrew：Mac平台下的软件包管理工具

  还有其他工具，例如：port
- Git：版本管理工具
- CocoaPods：iOS应用中最常用的类库管理工具

Instrument调试工具



- 应用、进程、设备的性能分析与测试工具

  Examine the behavior of apps or processes

  Examine device-specific features, such as Wi-Fi and Bluetooth

  Perform profiling in a simulator or on a physical device

  Track down problems in your source code

  Conduct performance analysis on your app

  Find memory problems in your app, such as leaks, abandoned memory, and zombies

  Identify ways to optimize your app for greater power efficiency

  Perform general system-level troubleshooting

  Save instrument configurations as templates

  (阅读：https://help.apple.com/instruments/)

Instrument调试工具

- Allocations： 用来检查内存分配，跟踪过程的匿名虚拟内存和堆的对象，提供类名和可选保留/释放历史
- Leaks： 一般的查看内存使用情况，检查泄漏的内存，并提供了所有活动的分配和泄漏模块的类对象分配统计信息，以及内存地址历史记录
- Time Profiler： 分析代码的执行时间，执行对系统的CPU上运行的进程低负载时间为基础采样
- Zombies： 检查是否访问了僵尸对象

CocoaPods

- CocoaPods manages library dependencies for your Xcode projects.
- Install CocoaPods on your computer.
- Using CocoaPods
  - 创建iOS工程
  - 在工程根目录创建Podfile
  - pod search 寻找需要依赖的库并在Podfile填写依赖
  - 工程目录执行pod install

Podfile

```
# add Alamofire to a single target

target 'MyApp' do
  use_frameworks!
  pod 'Alamofire', '~> 3.0'
end
```

开发模式

- **模拟器调试**

  可以将代码编译运行在手机模拟器上面，并进行调试

- **真机调试**

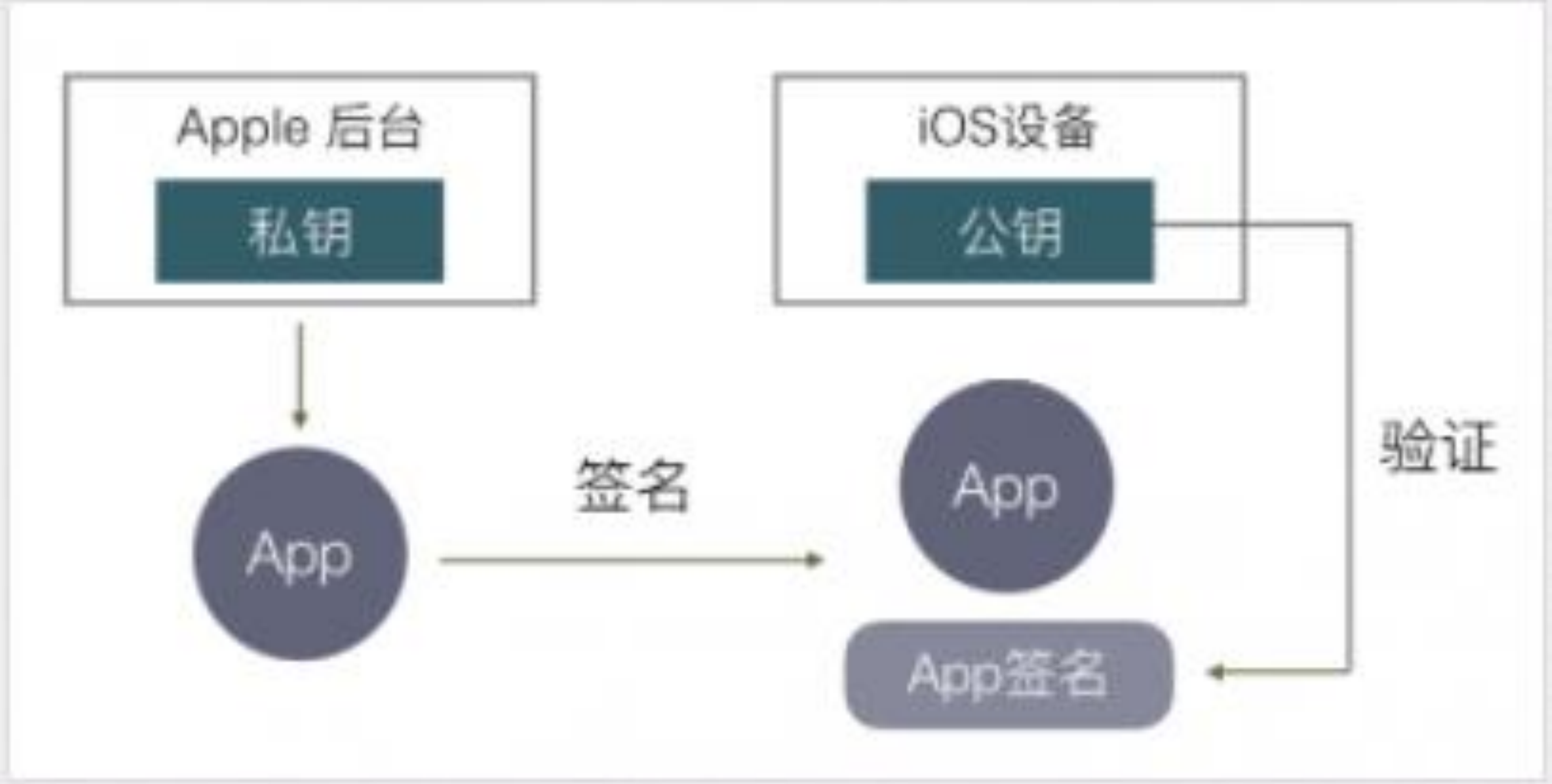
  需要开发者证书才能将代码编译后跑在物理手机上面

  无开发者证书？免费申请iOS证书打包ipa，本人真机安装测试：

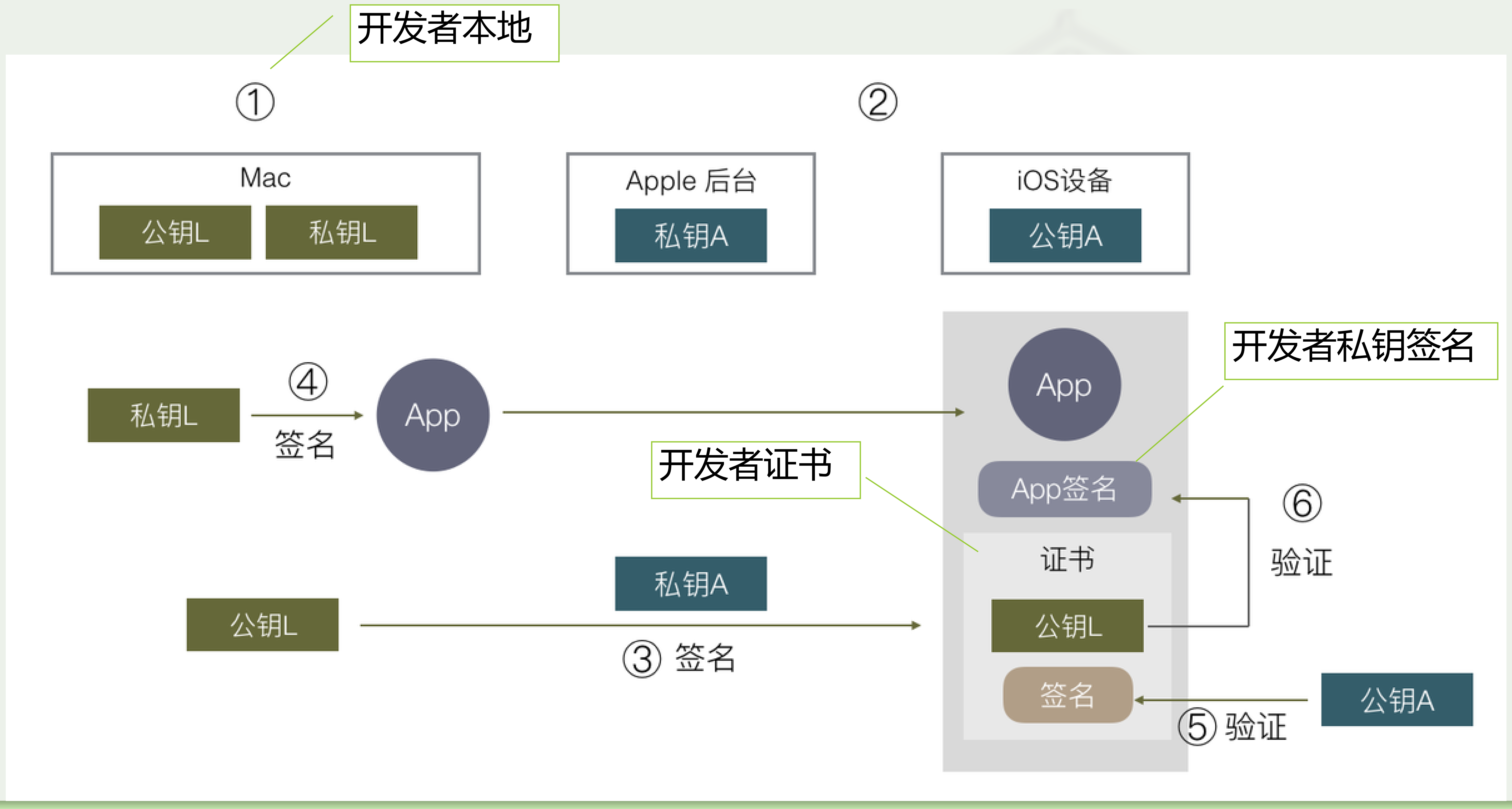
  1. 申请一个苹果账号
  2. 申请iOS测试证书（p12）
  3. 申请iOS描述文件（mobileprovision）
  4. 打包ipa
  5. 安装ipa

AppStore发布应用与安装

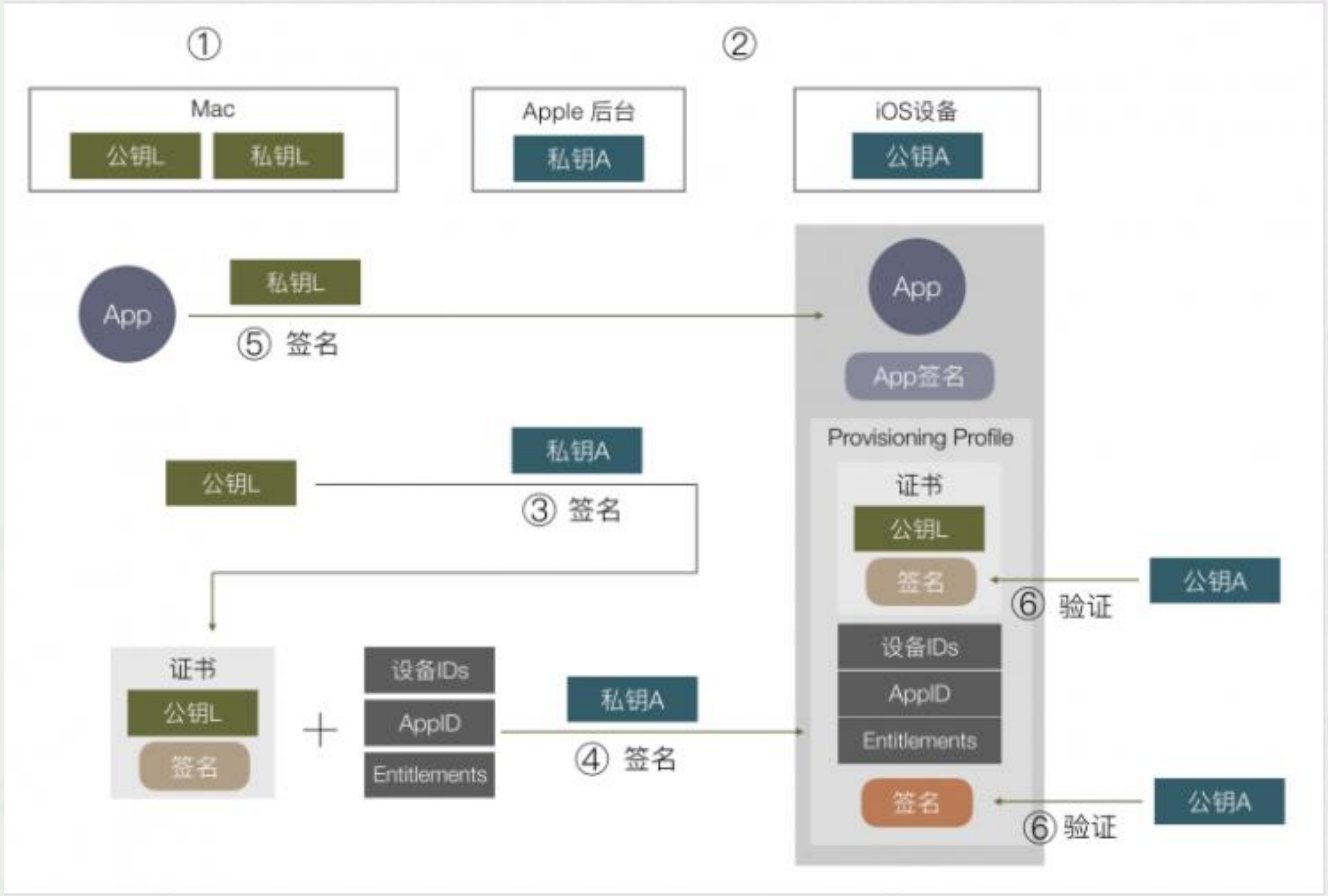
- 配对的公钥、私钥
- 后台发布App时私钥签名
- App安装到设备时用公钥验证



开发阶段真机测试新方式



更多细节

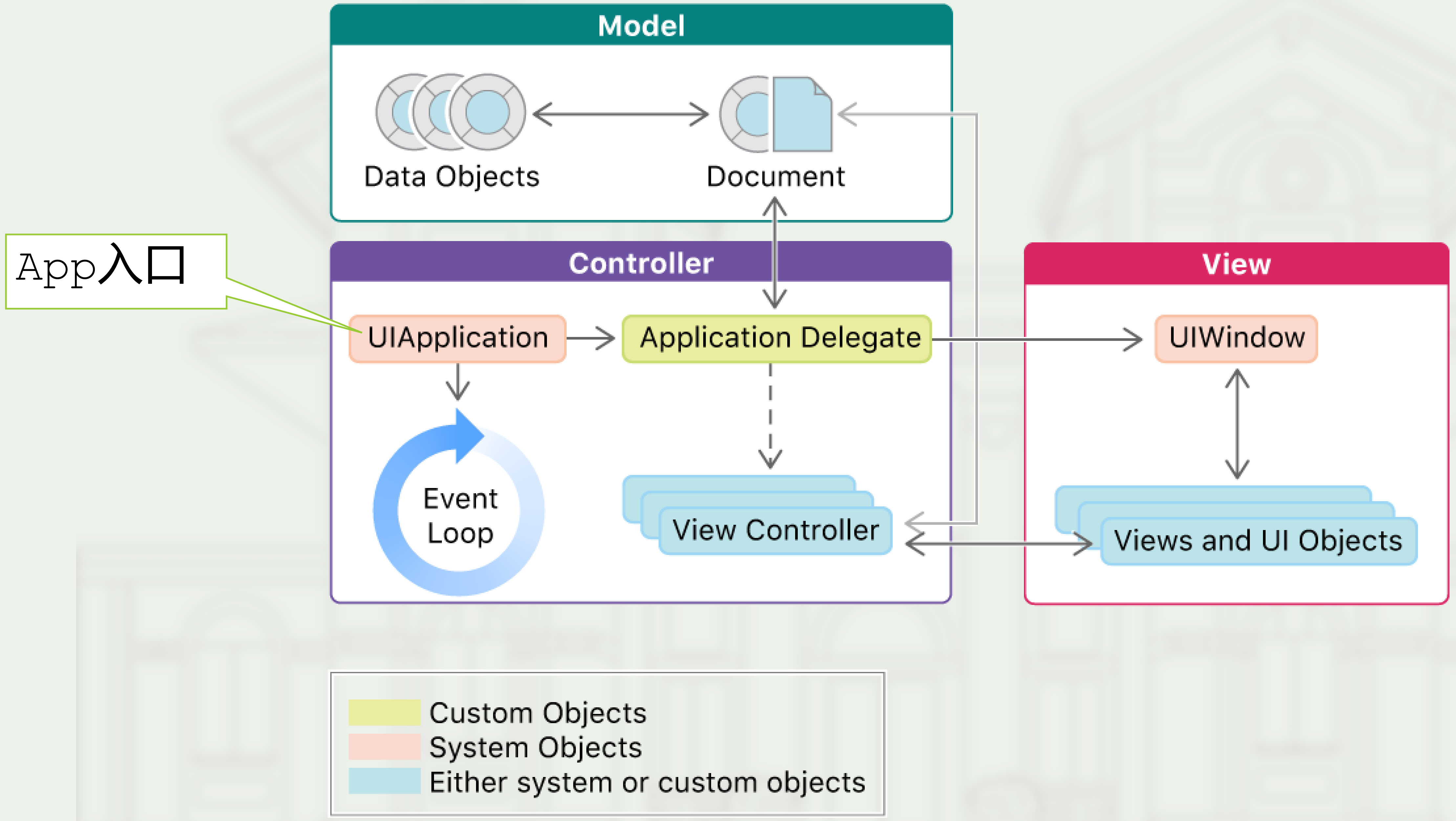

①
②
Mac
公钥L
私钥L
Apple 后台
私钥A
iOS设备
公钥A
App
私钥L
⑤ 签名
App
App签名
Provisioning Profile
证书
公钥L
签名
⑥ 验证
公钥A
公钥L
私钥A
③ 签名
证书
公钥L
签名
+
设备IDs
AppID
Entitlements
私钥A
④ 签名
设备IDs
AppID
Entitlements
签名
⑥ 验证
公钥A

开发证书相关概念

- Entitlements:包含了App权限开关列表
- P12: 本地私钥或者证书或者私钥与证书的二级制形式，可以导入其他电脑，用于签名App
- Provisioning Profile: 包含了证书/Entitlements等数据，并由苹果后台私钥签名的数据包

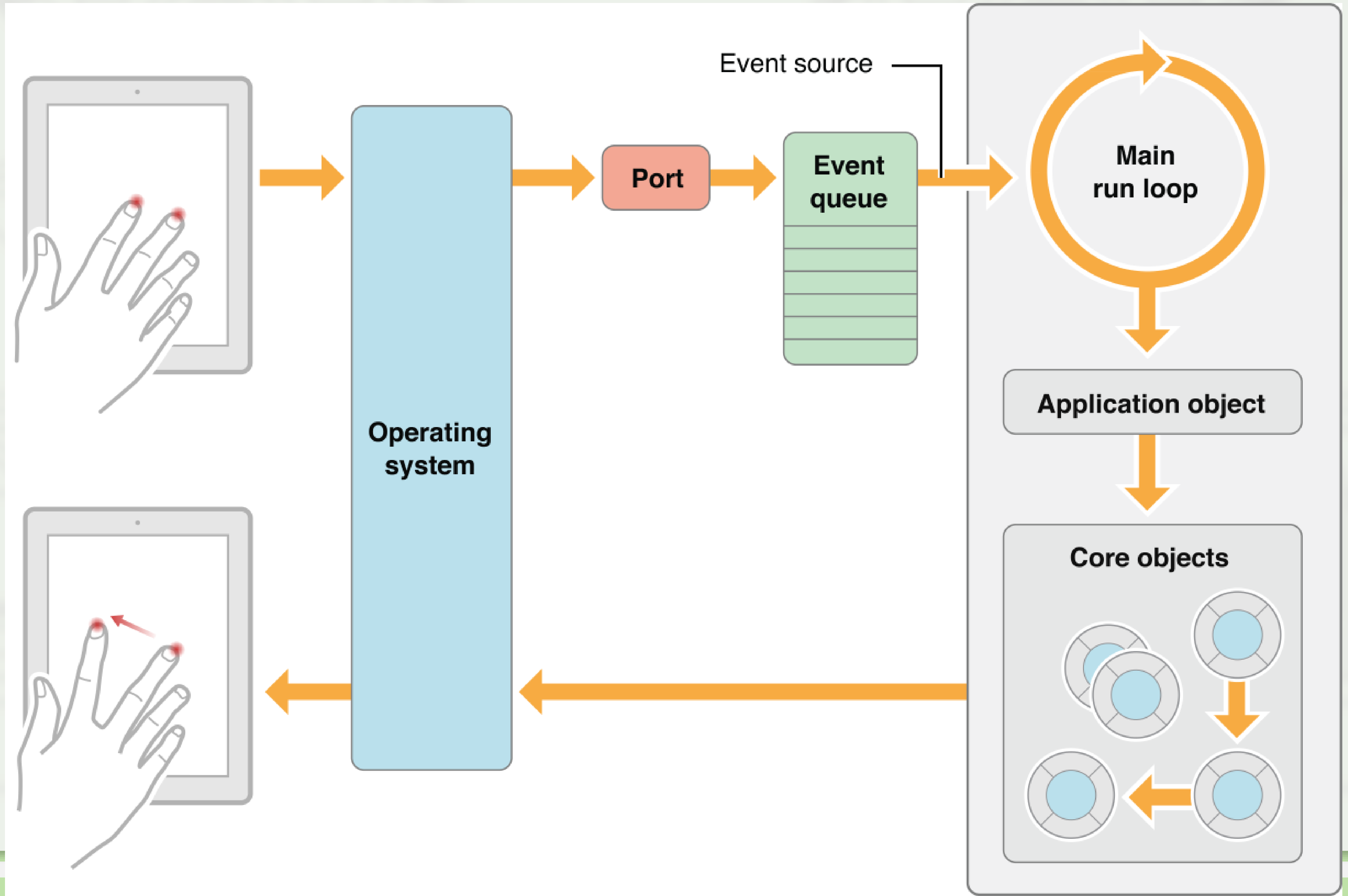
应用程序生命周期

Structure of a UIKit App



应用程序生命周期

- UIApplication将会创建MainRunLoop对象
- 同时将MainThread放到MainRunLoop中



应用程序入口

main.m

- argc, argv来自于main的两个参数
- 第3个参数为nil默认为UIApplication
- 第4个参数是代理类的名称，AppDelegate是UIApplication的代理

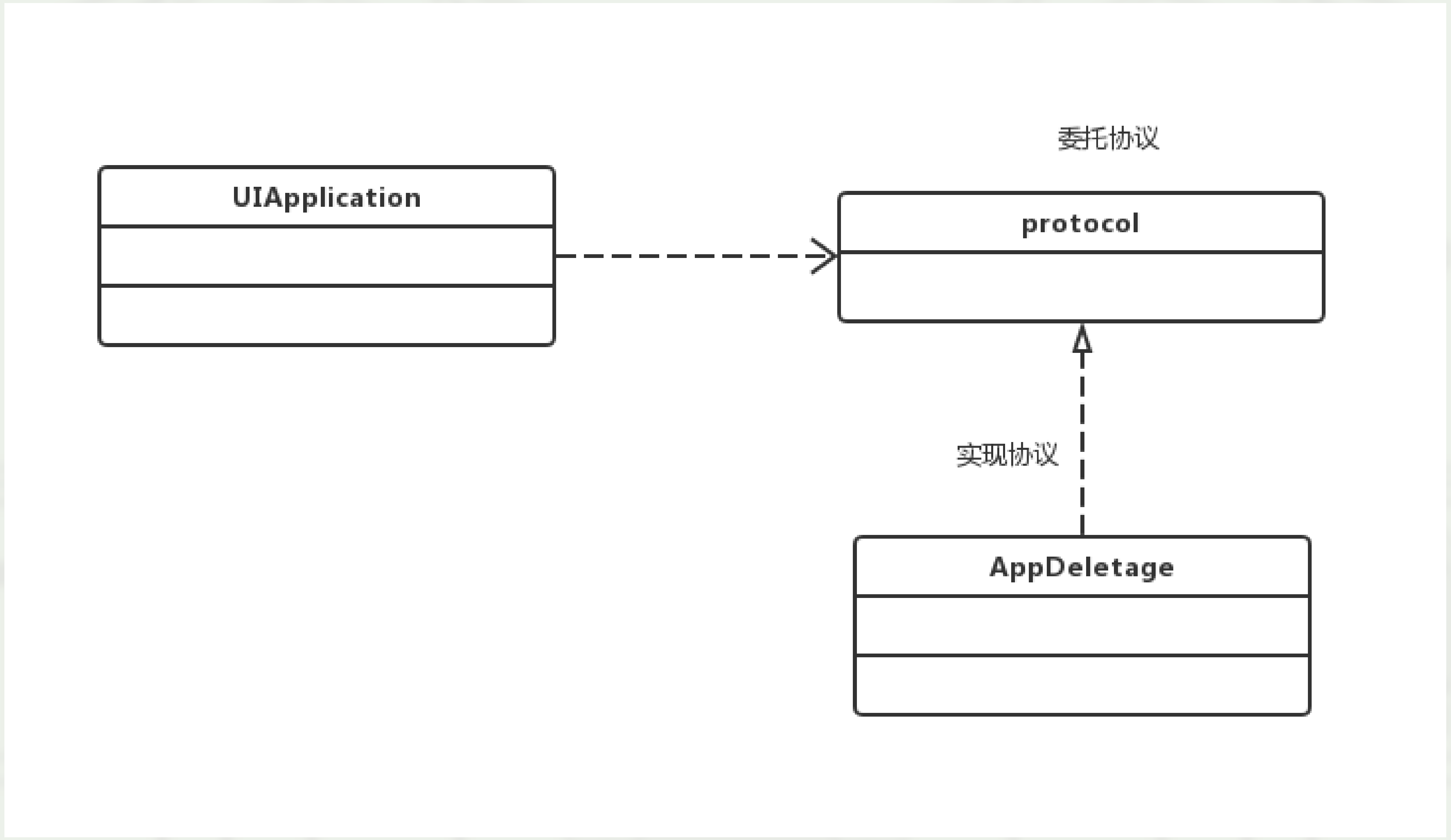
```
#import <UIKit/UIKit.h>
#import "AppDelegate.h"

int main(int argc, char * argv[]) {
  @autoreleasepool {
    return UIApplicationMain(
      argc, argv,
      nil,
      NSStringFromClass([AppDelegate class]));
  }
}
```

UIApplicationMain

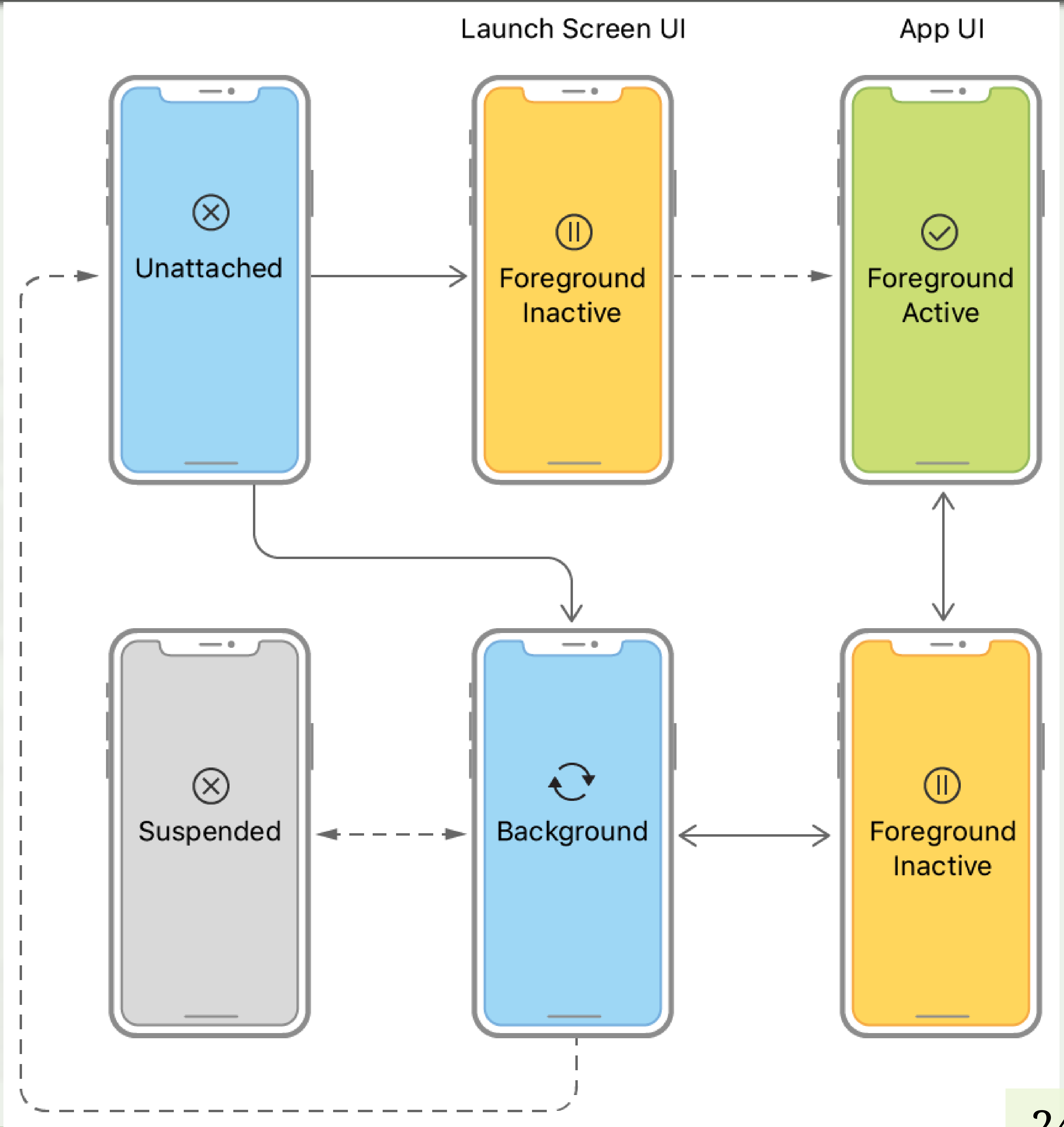
- Instantiates the UIApplication object from the principal class
- Instantiates the delegate (if any) from the given class and sets the delegate for the application.
- Sets up the main event loop, including the application's run loop, and begins processing events.
- Load nib file if Info.plist specifies by key NSMainNibFile.
- Despite the declared return type, this function never returns.
- UIApplication是单例，可执行一些应用级的操作.

UIApplication与AppDelegate关系



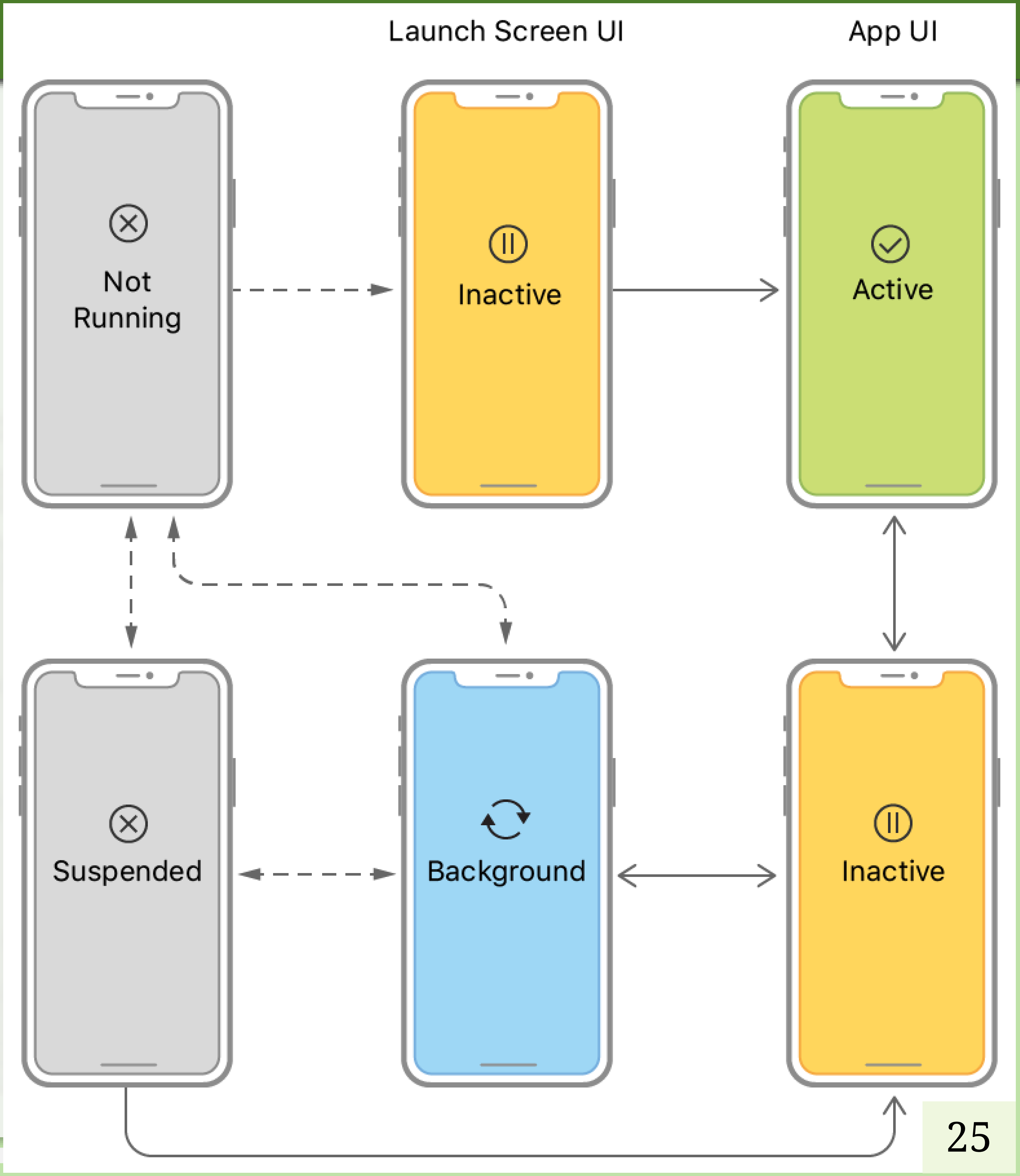
Scene-Based Life-Cycle

- iOS 13+
- 由UISceneDelegate对象处理
- UIApplicationSceneManifest
  Add this key to Info.plist
- User requested: Foreground
- System requested: Background

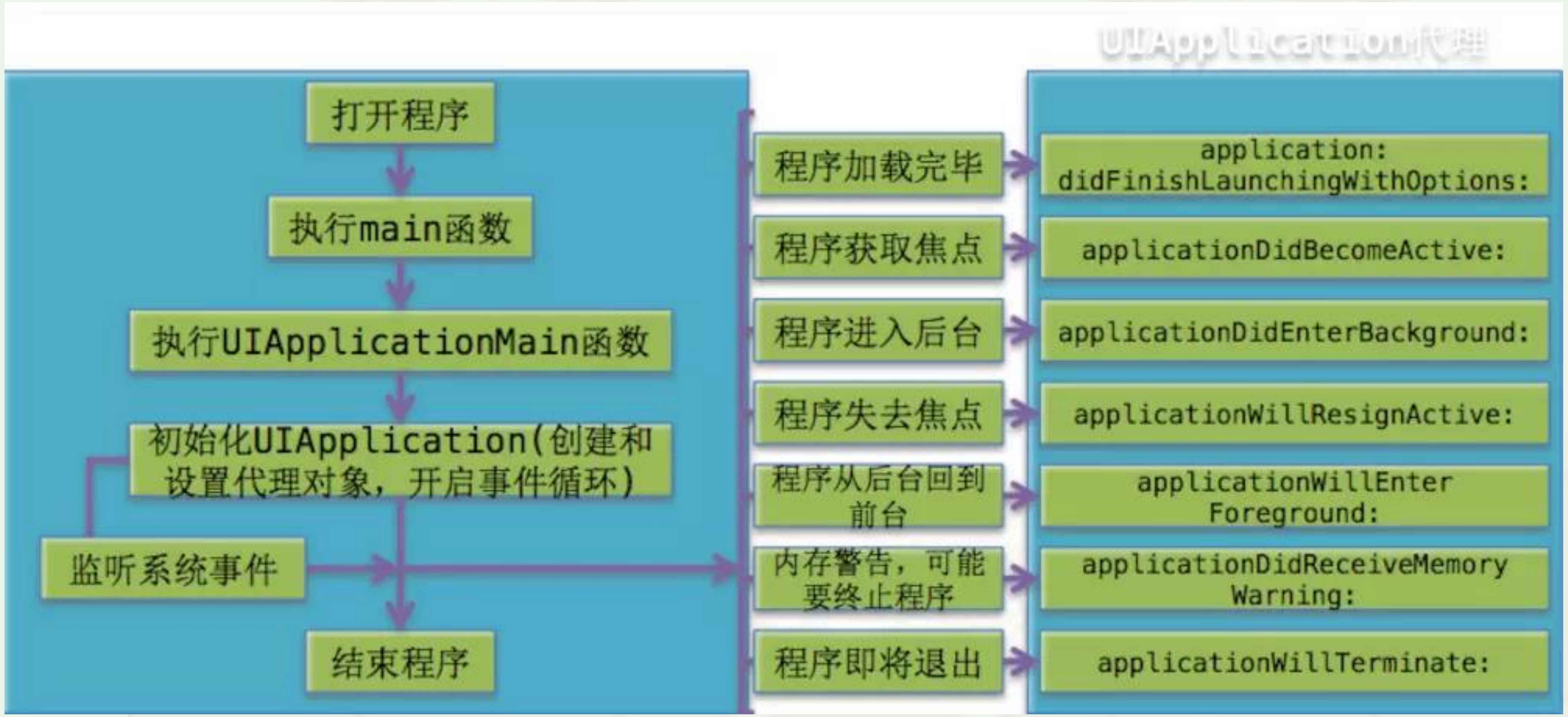


应用程序生命周期

- iOS12 and earlier
- UIApplicationDelegate
- Launch screen UI: Active
- or Background



应用程序生命周期


UIApplication代理
打开程序
执行main函数
执行UIApplicationMain函数
初始化UIApplication(创建和设置代理对象，开启事件循环)
监听系统事件
结束程序
程序加载完毕
程序获取焦点
程序进入后台
程序失去焦点
程序从后台回到前台
内存警告，可能要终止程序
程序即将退出
application: didFinishLaunchingWithOptions:
applicationDidBecomeActive:
applicationDidEnterBackground:
applicationWillResignActive:
applicationWillEnter Foreground:
applicationDidReceiveMemory Warning:
applicationWillTerminate:

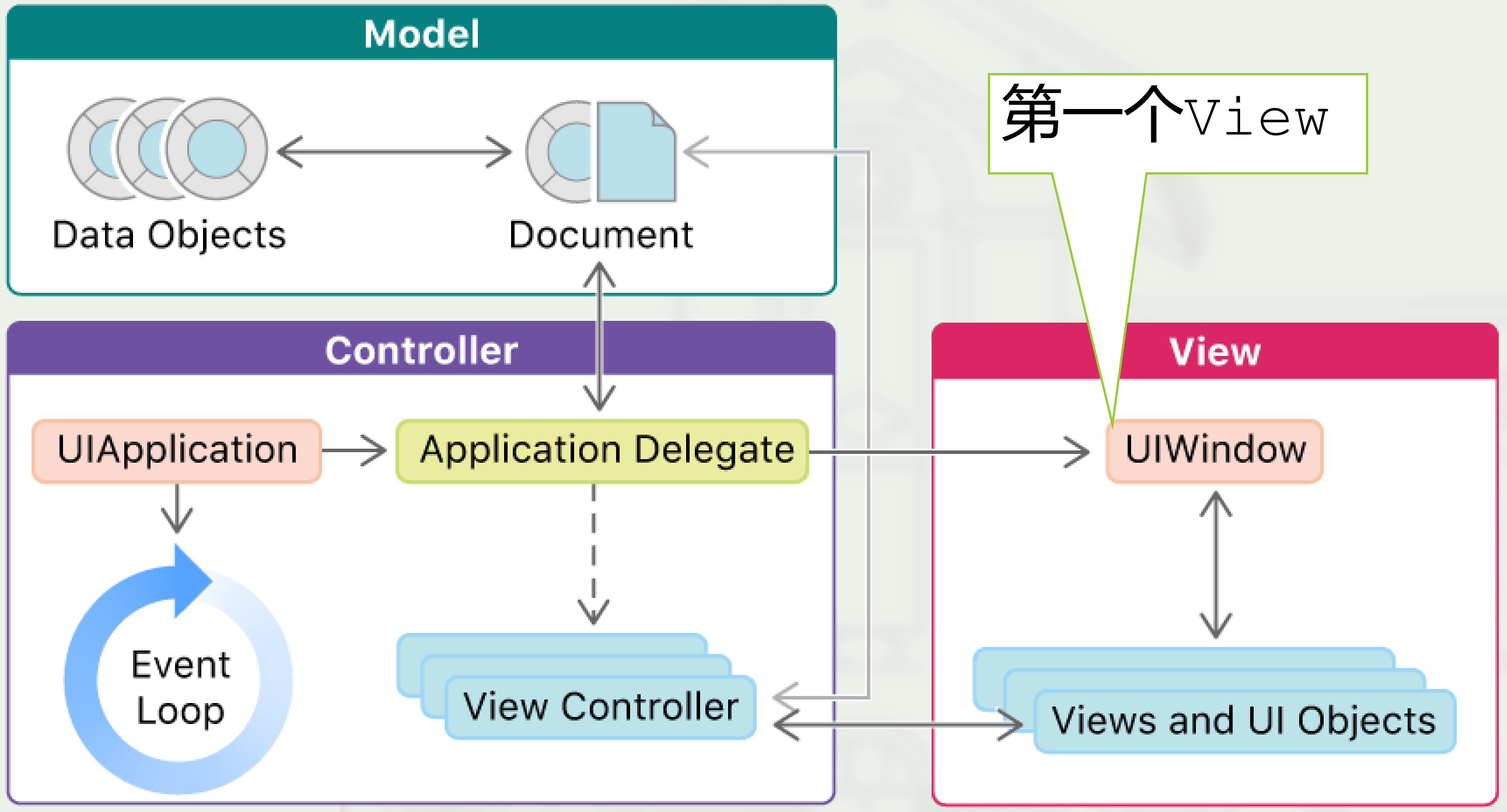
App Termination仅能感知被用户主动杀死的时机，不能感知被系统杀死的时机

生命周期回调函数说明

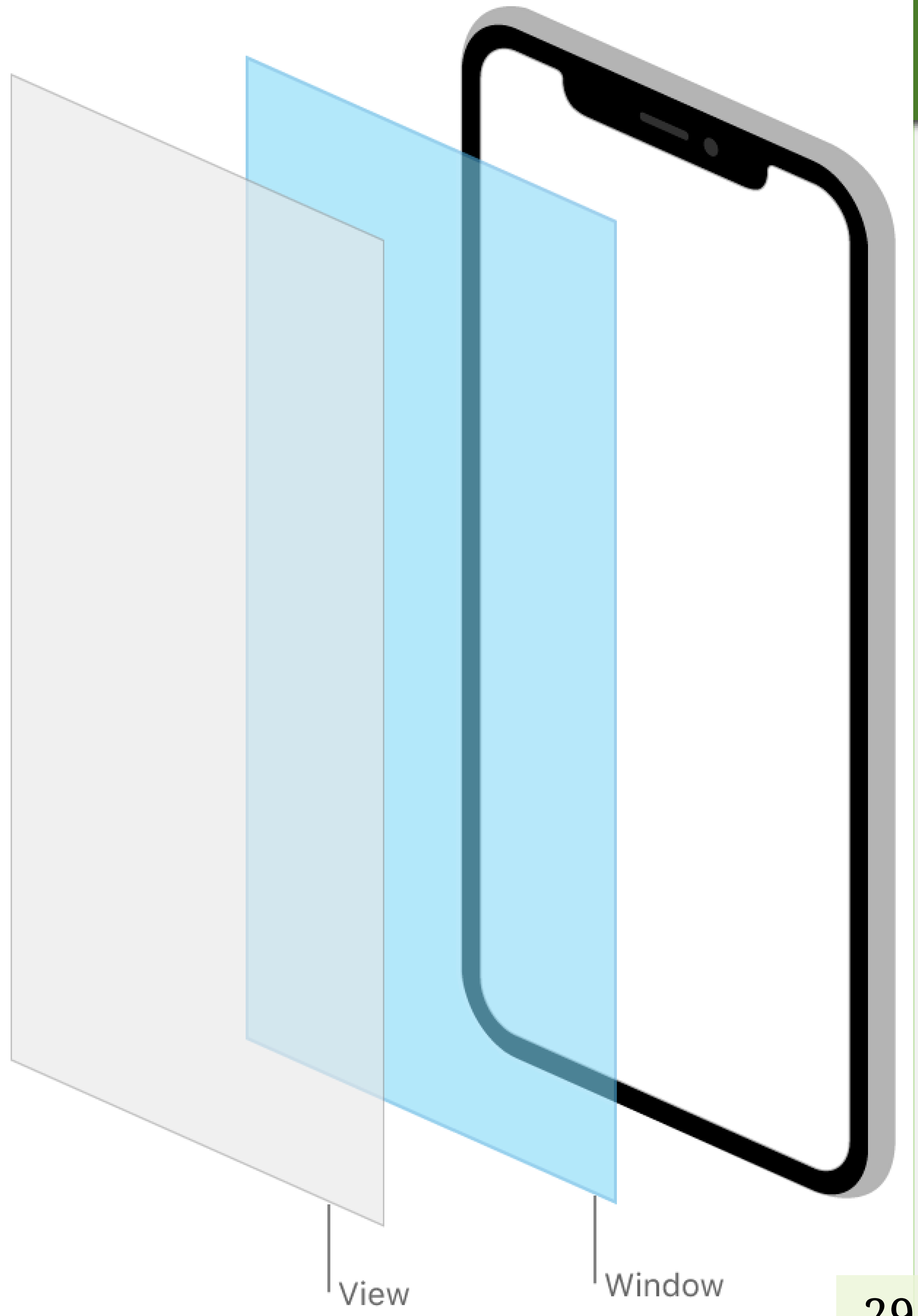
- application:willFinishLaunchingWithOptions:—First chance to execute code at launch time.
- application:didFinishLaunchingWithOptions:—perform any final initialization before your app is displayed to the user.
- applicationDidBecomeActive:—Lets your app know that it is about to become the foreground app. Use this method for any last minute preparation.
- applicationWillResignActive:—Lets you know that your app is transitioning away from being the foreground app. Use this method to put your app into a quiescent state.
- applicationDidEnterBackground:—Lets you know that your app is now running in the background and may be suspended at any time.
- applicationWillEnterForeground:—Lets you know that your app is moving out of the background and back into the foreground, but that it is not yet active.
- applicationWillTerminate:—Lets you know that your app is being terminated. This method is not called if your app is suspended.

UIWindow

Structure of a UIKit App



🟨 Custom Objects
🟥 System Objects
🟦 Either system or custom objects



UIWindow初始化

```
// AppDelegate.h
@interface AppDelegate : UIResponder <UIApplicationDelegate>
@property (strong, nonatomic) UIWindow *window;
@end

// AppDelegate.m
- (BOOL)application:(UIApplication *)app didFinishLaunchingWithOptions:
                   (NSDictionary *)launchOptions {
  self.window = [[UIWindow alloc]initWithFrame:[[UIScreen mainScreen]bounds]];
  self.window.rootViewController=[[ViewController alloc]init];
  self.window.backgroundColor=[UIColor greenColor];
  [self.window makeKeyAndVisible];

  NSLog(@"application:didFinishLaunchingWithOptions:");

  return YES;
}
```

基本概念

- The backdrop for your app's user interface and the object that dispatches events to your views.
- 一个iOS程序之所以能显示到屏幕上，完全是因为它有UIWindow，是一种特殊的UIView，通常在一个App中至少会有一个UIWindow
- iOS程序启动完毕后，创建的第一个视图控件就是UIWindow，接着创建控制器的view，最后将控制器的view添加到UIWindow上，于是控制器的view就显示在屏幕上了

UIWindow Usage

- Setting the z-axis level of your window, which affects the visibility of the window relative to other windows.

  `@property(nonatomic) UIWindowLevel windowLevel;`

- Showing windows and making them the target of keyboard events.

  `@property(nonatomic, readonly, getter=isKeyWindow) BOOL keyWindow;`

- Converting coordinate values to and from the window's coordinate system.
- Changing the root view controller of a window.

  `@property(nonatomic, strong) UIViewController *rootViewController;`

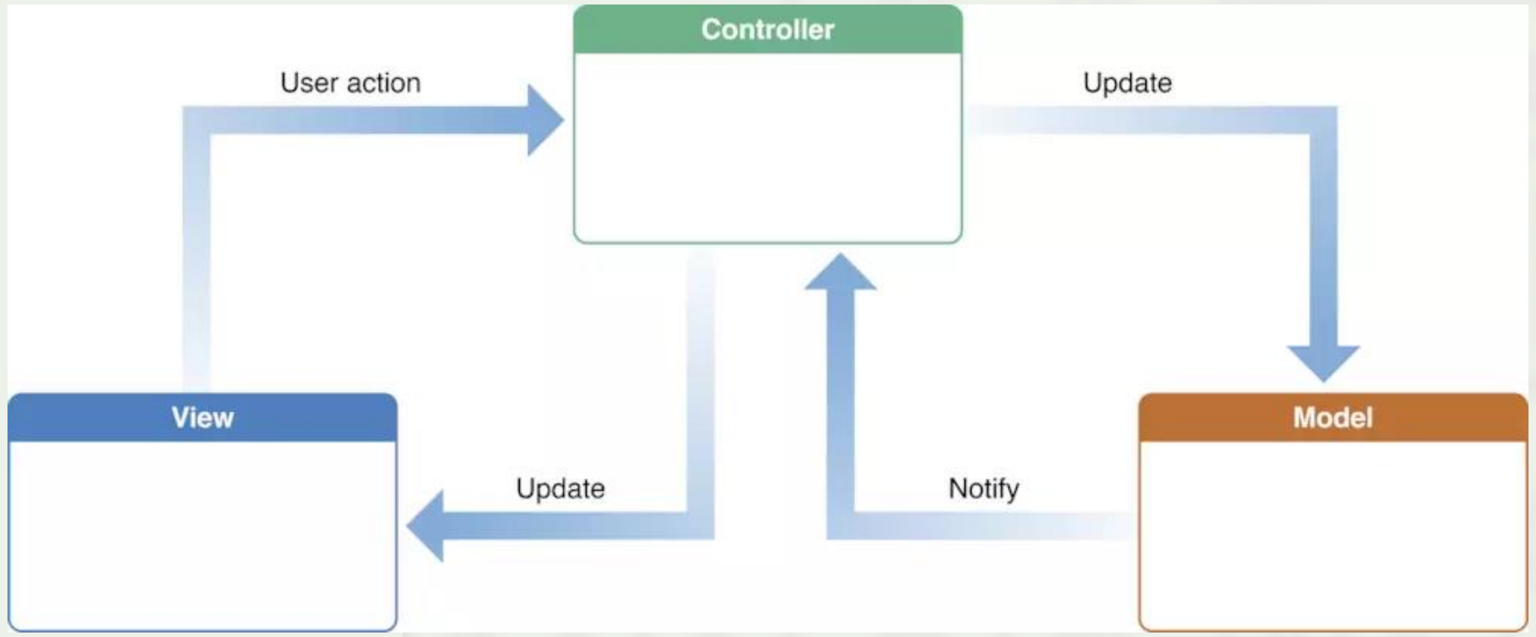
- Changing the screen on which the window is displayed.

MVC架构

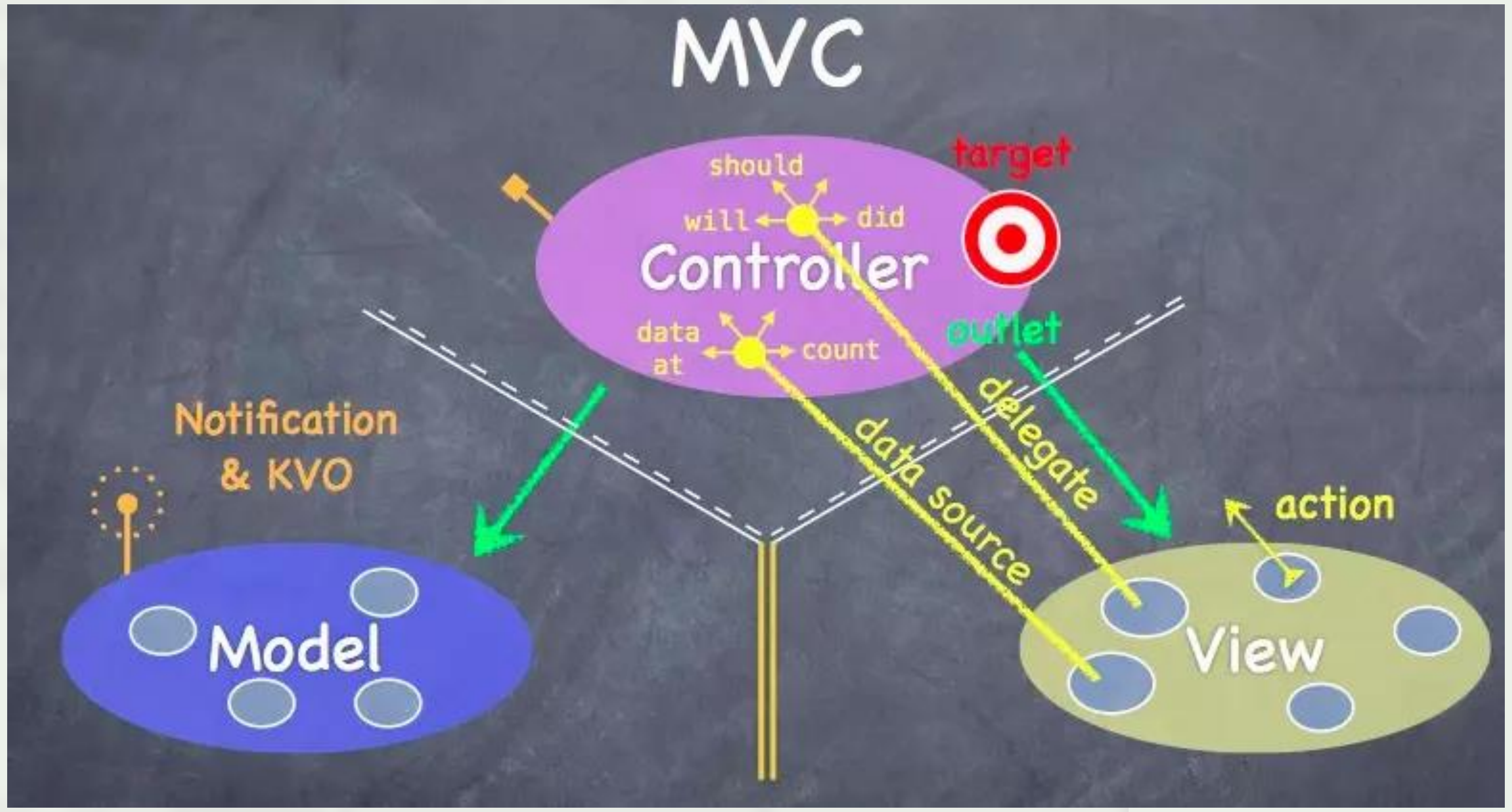
示例图

- 所有对象划分为3个阵营，各司其职

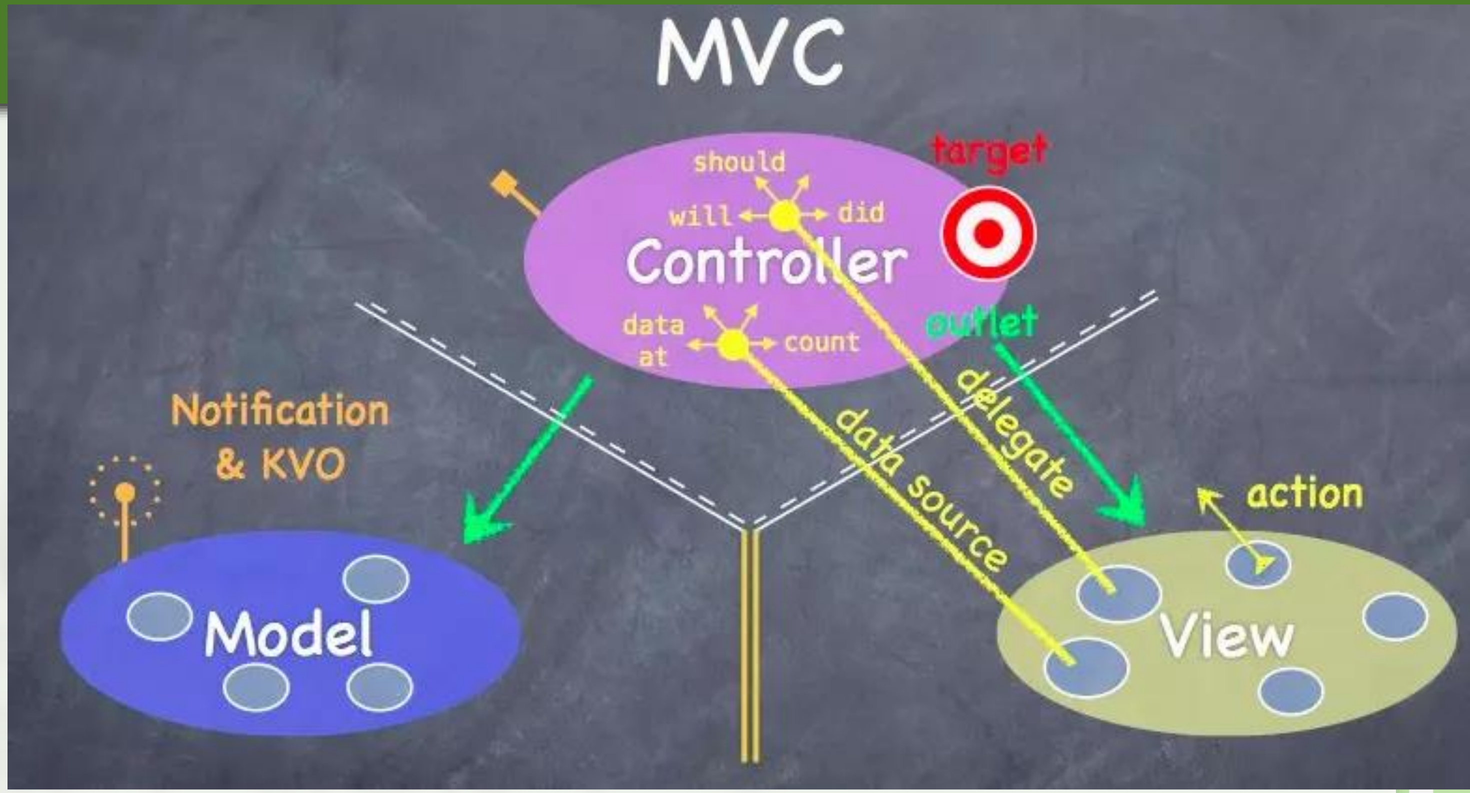
苹果官方给出的示例图：



斯坦福公开课给出的示例图：



MVC架构



- Model: 业务数据的表示
  What your App is (but not how it is displayed)
  通过Notification & KVO广播更新给Controller
- View: 数据的呈现，提供UI和交互
  Your controller's minions.
  不拥有数据
  间接与Controller沟通（action，delegate/protocol，data source）
- Controller: Model和View的中介，处理业务逻辑，获得Model的数据并更新界面
  How your Model is presented to the user (UI logic)
  可直接访问Model和View
  为View解析、格式化Model的信息

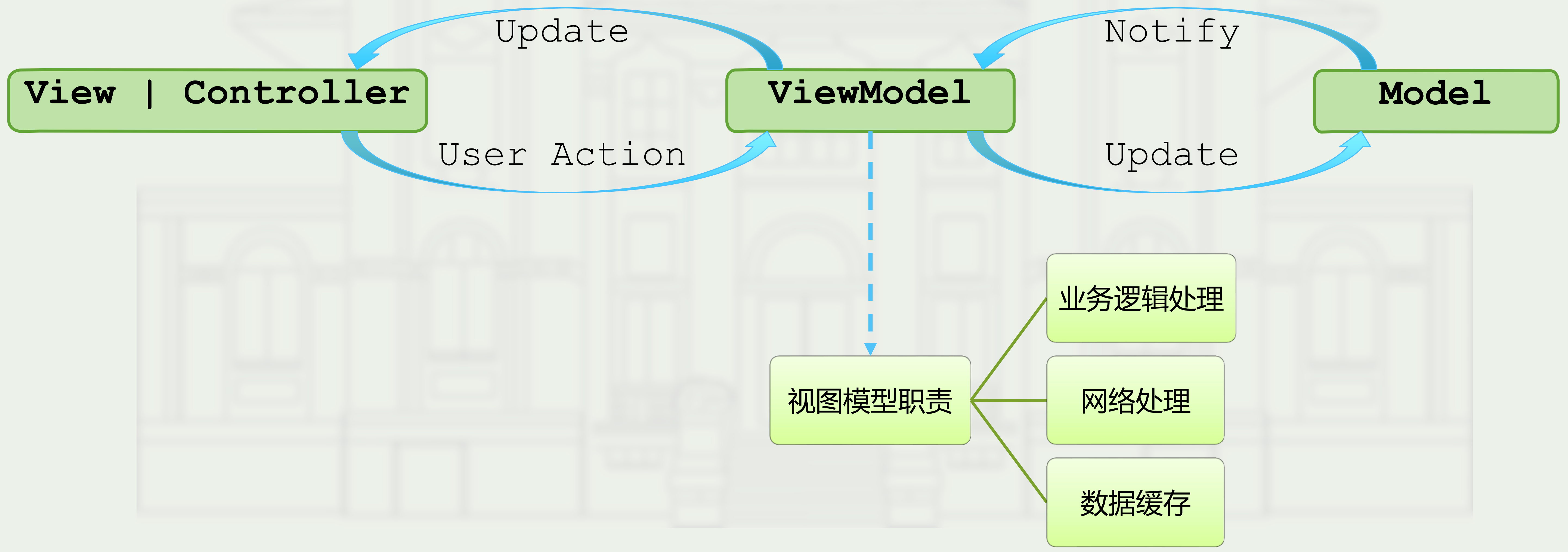
MVC的好处

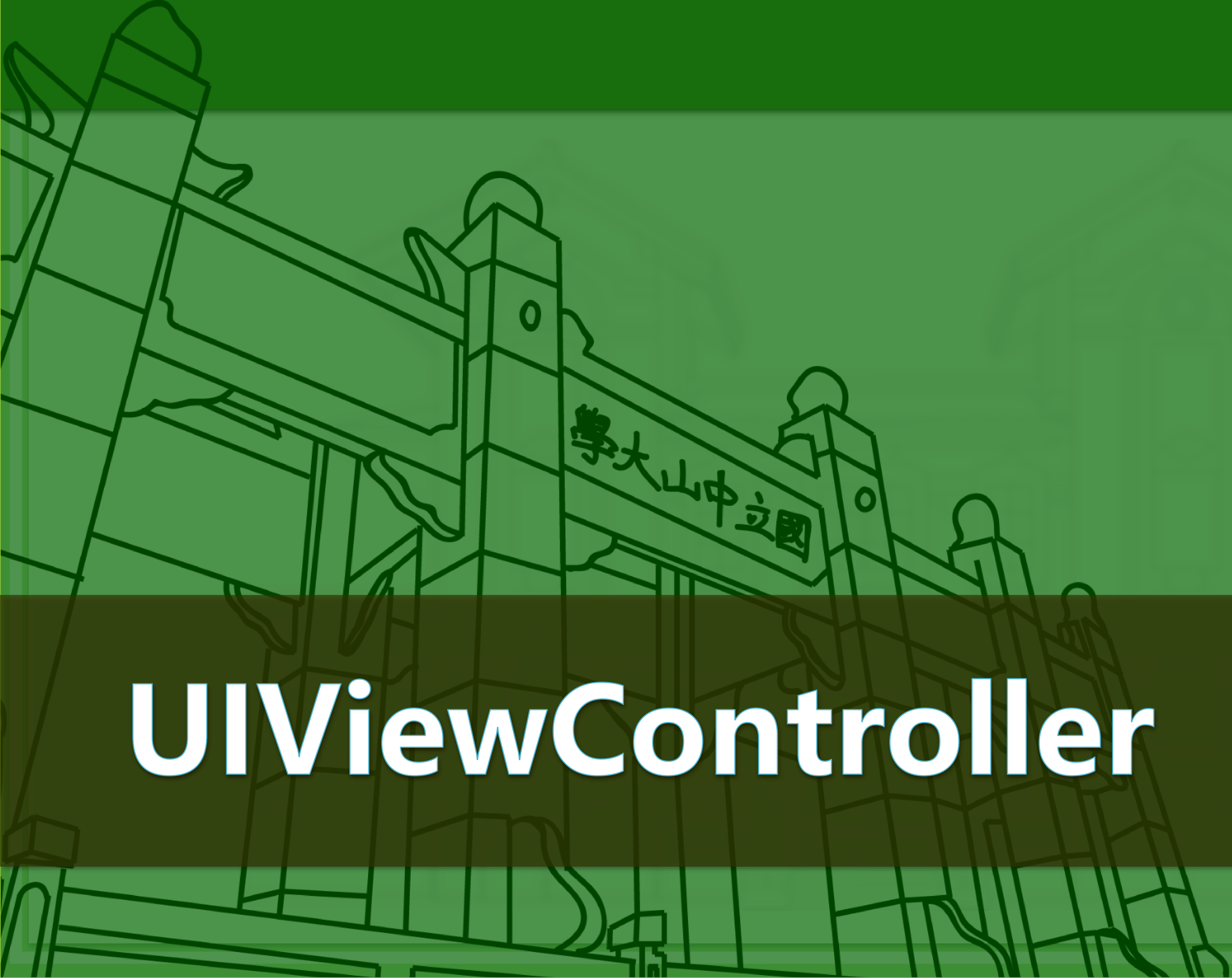
- 各司其职，互不干涉，有利于维护
- 有利于开发分工
- 有利于组件的重用

*MVC为**Massive** View Controller?*

MVVM

- 解决Massive View Controller：将Controller中的展示逻辑抽取出来，放置到专门的ViewModel中
- View和View Controller紧密联系，视为一个组件

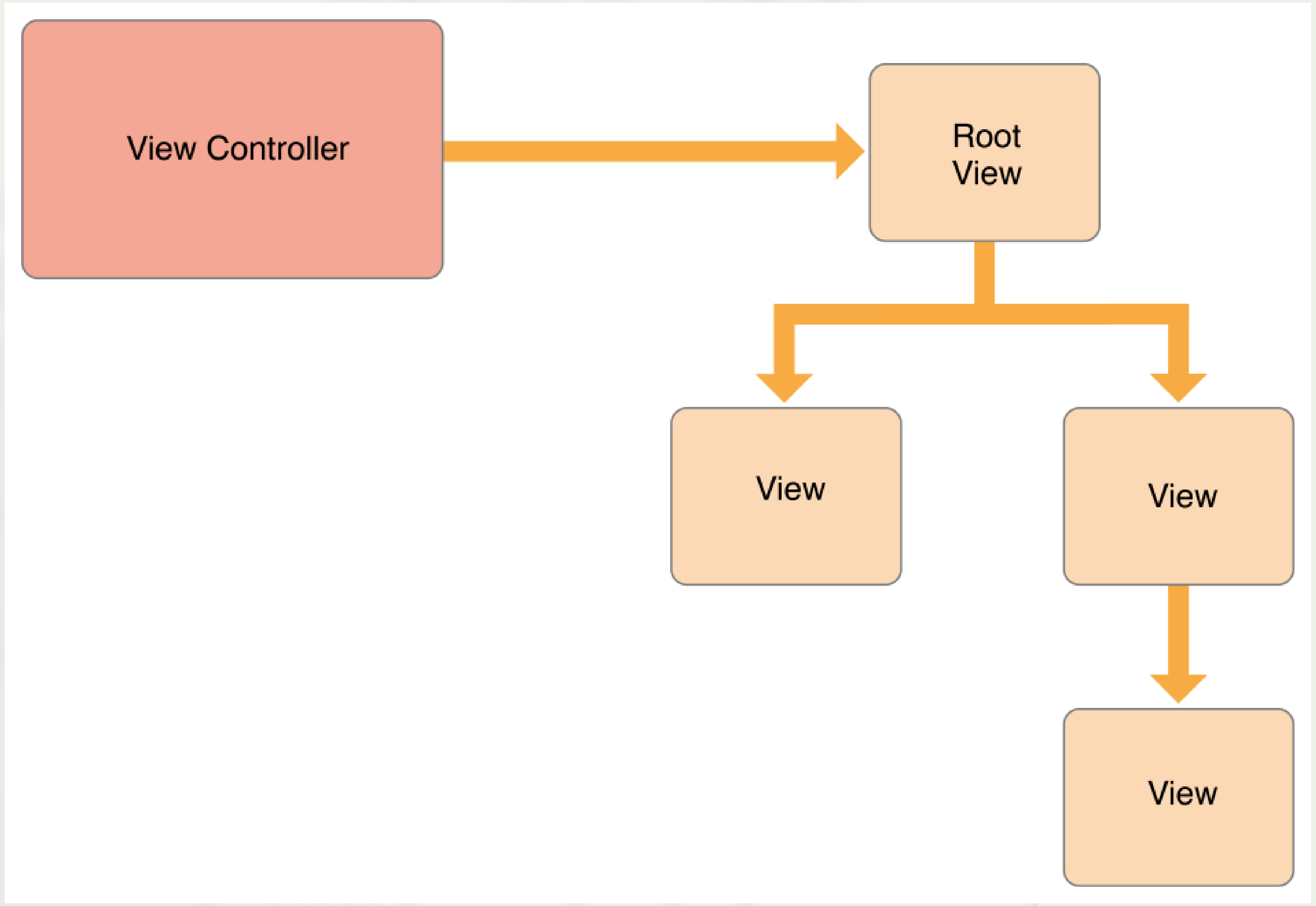




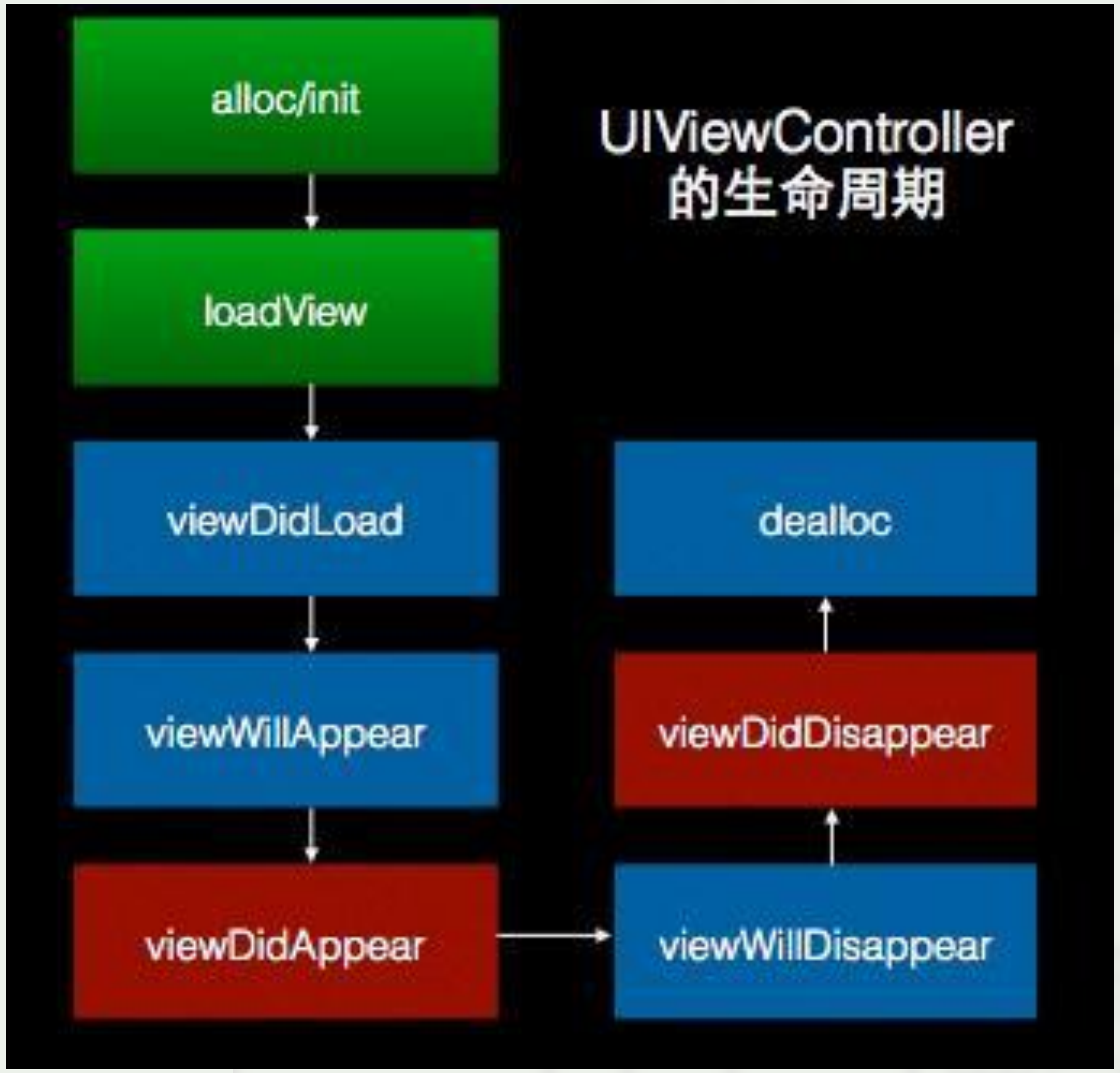
UIViewController

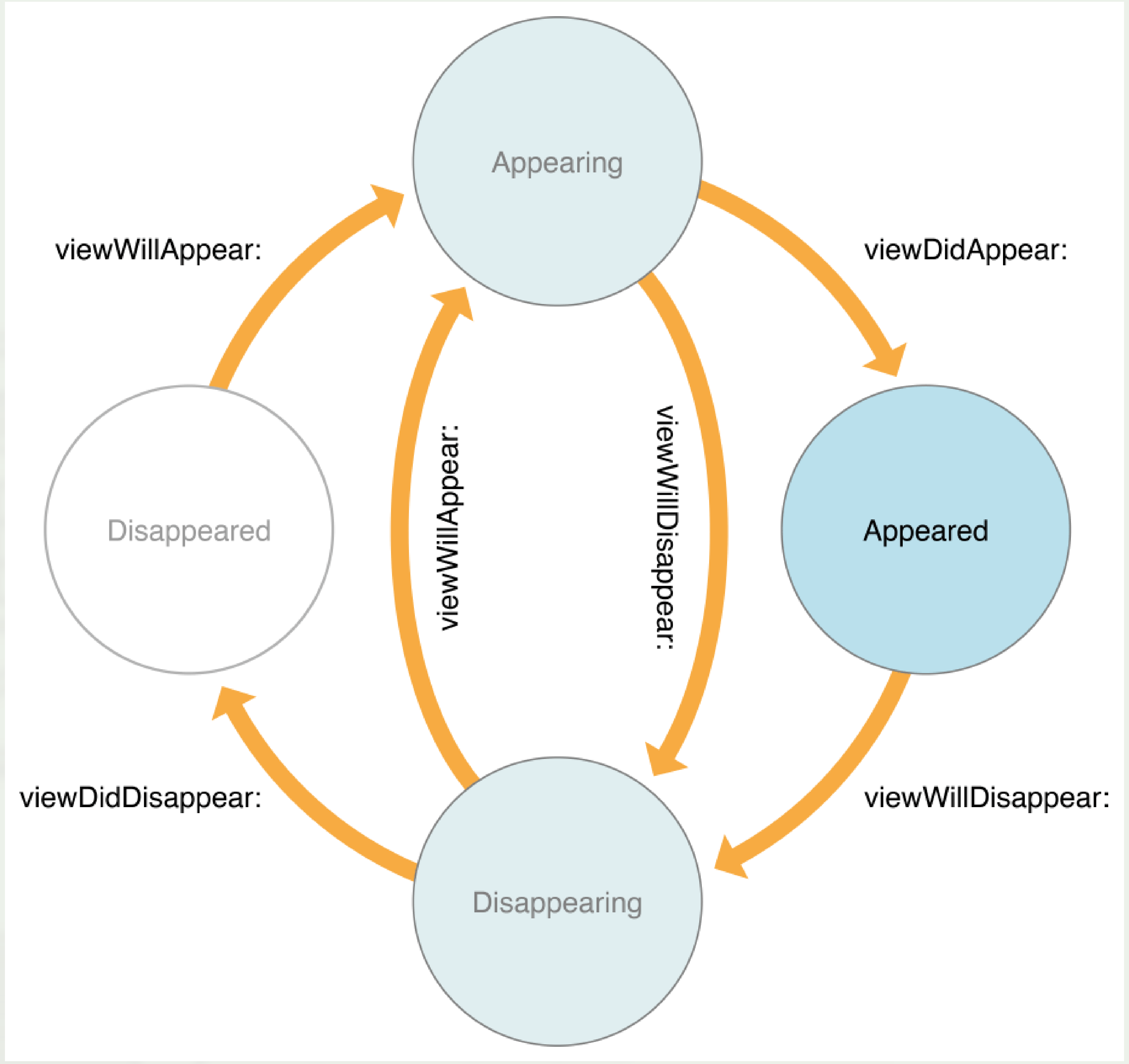
常用ViewController

- UIViewController
  - 内容视图控制器
  - 容器视图控制器
- UINavigationController
  - 导航视图控制器
  - 属容器视图控制器
- UITabBarController
  - 选项卡视图控制器
  - 属容器视图控制器

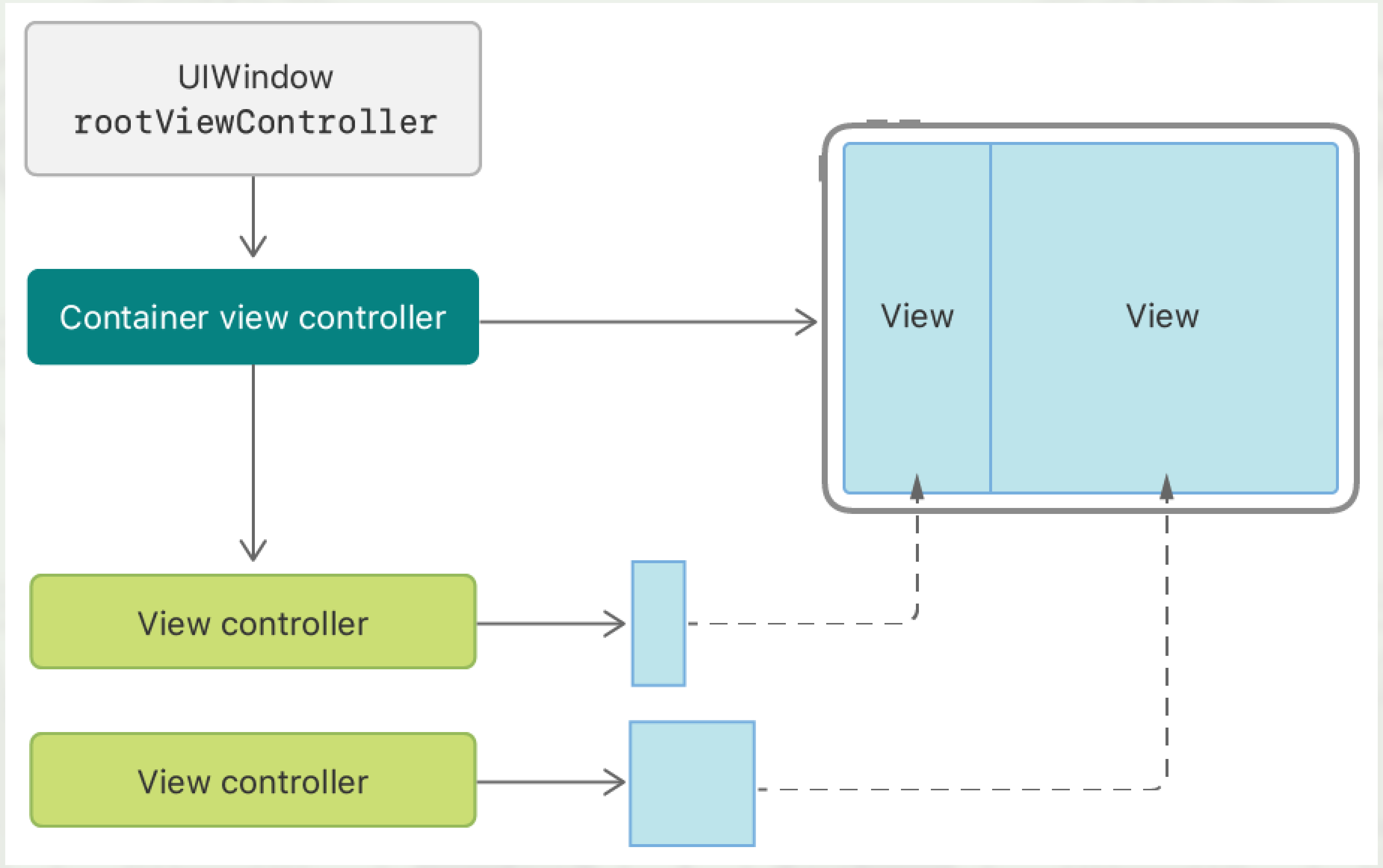


UIViewController的生命周期

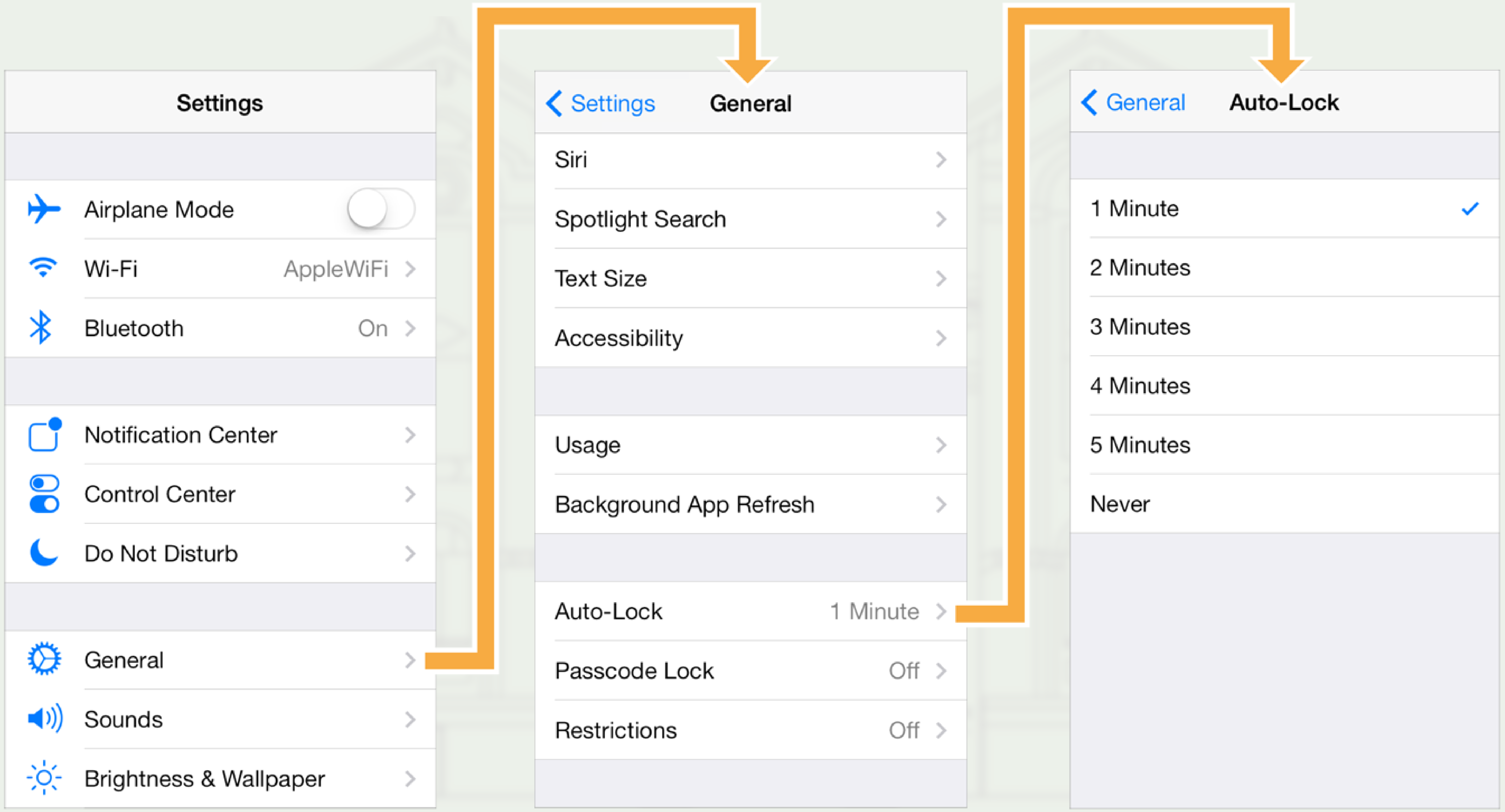




Container view controllers



A sample navigation interface



Settings

Airplane Mode

Wi-Fi AppleWiFi

Bluetooth On

Notification Center

Control Center

Do Not Disturb

General

Sounds

Brightness & Wallpaper

< Settings General

Siri

Spotlight Search

Text Size

Accessibility

Usage

Background App Refresh

Auto-Lock 1 Minute

Passcode Lock Off

Restrictions Off

< General Auto-Lock

1 Minute ✓

2 Minutes

3 Minutes

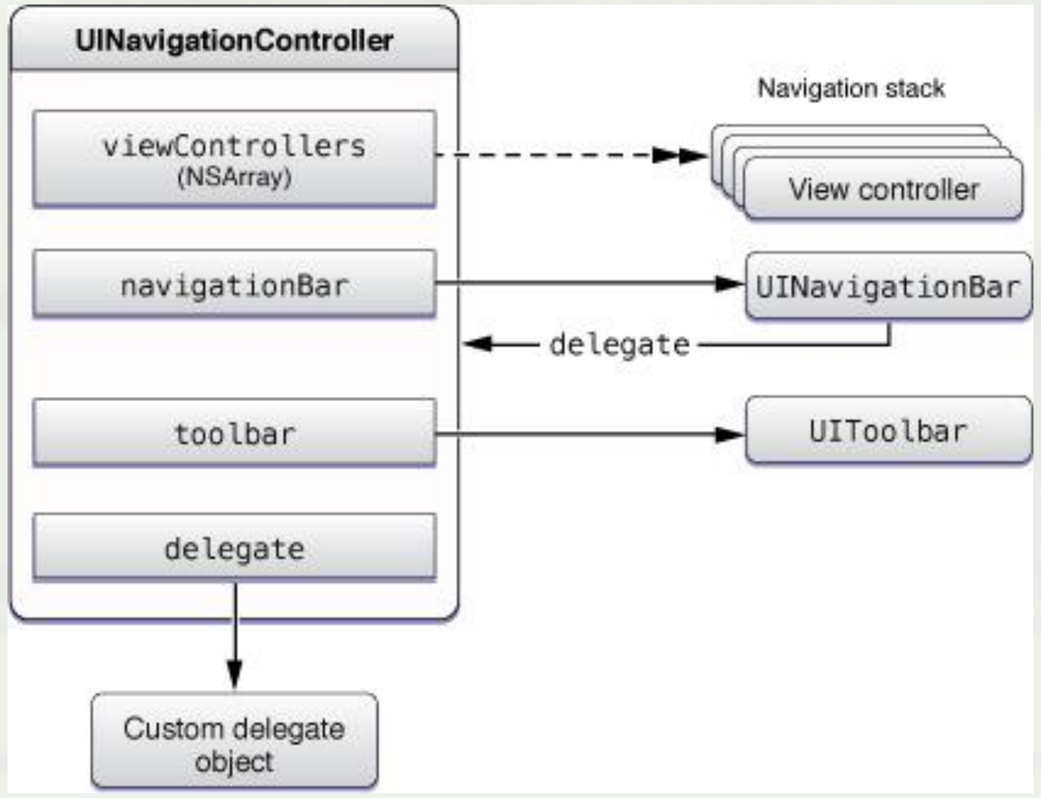
4 Minutes

5 Minutes

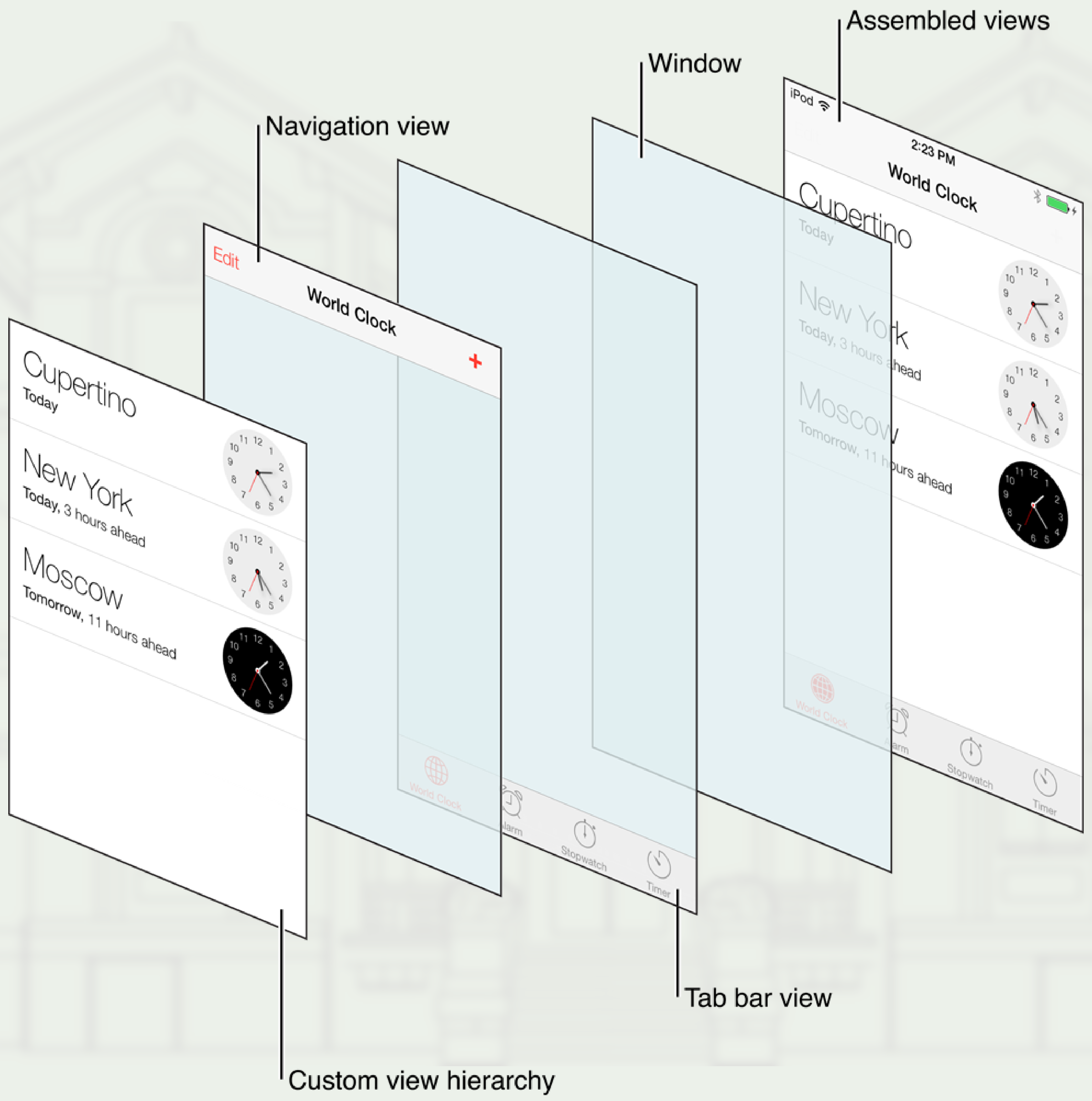
Never

UINavigationController导航控制器

- A container view controller that defines a stack-based scheme for navigating hierarchical content.
- 可以轻松完成多个控制器之间的切换
- 包含导航条（y=20）、栈顶控制器的view、导航控制器的view。

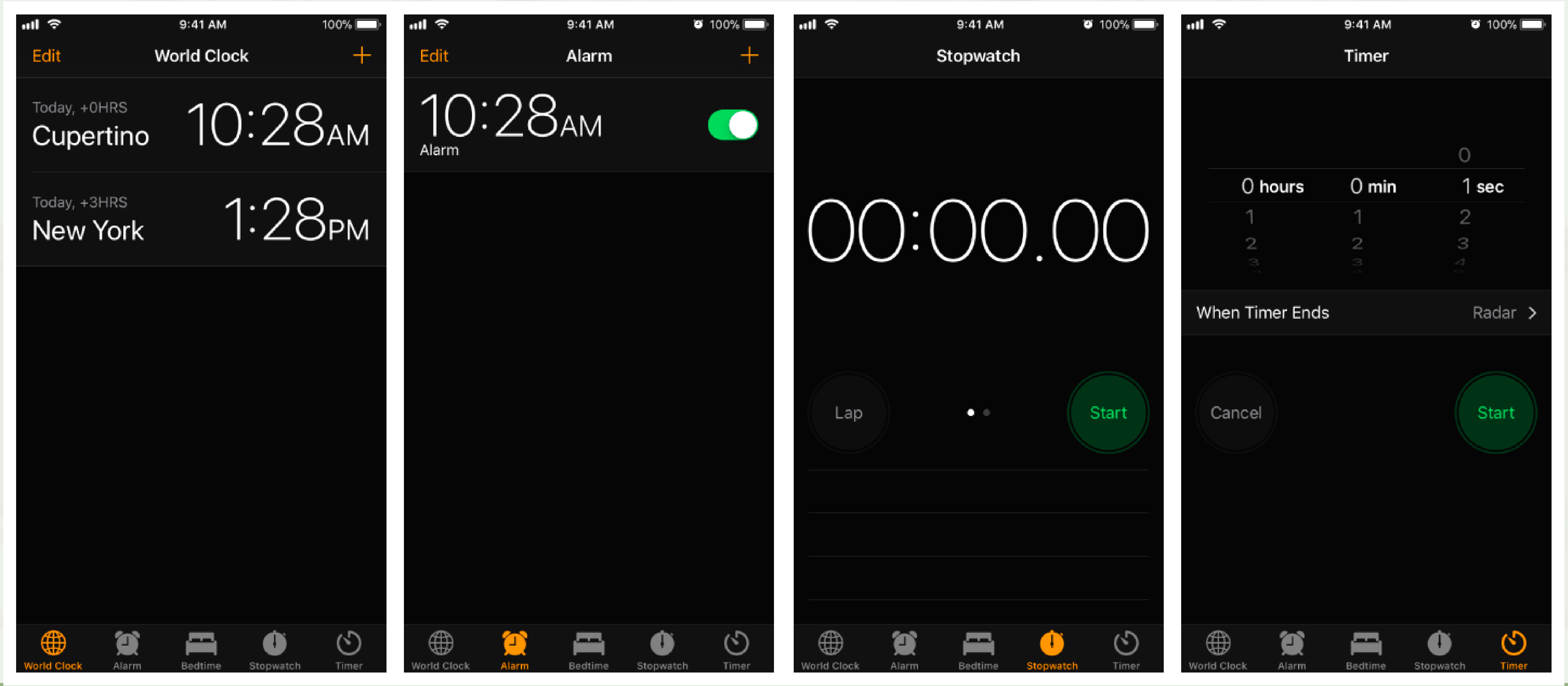


The views of a navigation controller



UITabBarController

- 选项卡控制器与导航控制器一样被广泛应用于iOS应用
- 选项卡控制器在屏幕底部显示一系列“选项卡”图标或文本，触摸它们可切换至不同的界面



页面管理方式

- Add a Child View Controller

  1. Call the addChildViewController: method of your container view controller to configure the containment relationship.

  2. Add the child's root view to your container's view hierarchy.

  3. Add constraints to set the size and position of the child's root view.

  4. Call the didMoveToParentViewController: method of the child view controller to notify it that the transition is complete.

- pushViewController

  导航控制器入栈的方式切换页面

- presentViewController

  模态切换的方式切换页面

课后练习题

课后练习题

- 一个App切换到后台，然后再回到前台，怎么计算App在后台停留的时间?
- 创建一个NavigationController容器，根视图控制器是A，从A跳到另一个视图控制器B，怎么计算在B的页面停留的时间?
- 假设有10个用户的信息（名字+生日），使用MVC的写法，将数据在列表里面显示出来。

自由练习，不需要提交，欢迎分享!

THANKS