

# Fundamentos iOS / Swift



# Desarrollo nativo para iOS

- Objective C o Swift & Cocoa Touch.
- Su conocimiento es fundamental para desarrollar para iOS.
- Las herramientas son gratis.
- Necesitas un Mac.
- Tiene una enorme demanda laboral en estos momentos.

# Requisitos para crear apps en iOS

- Xcode 11.5      Xcode 10.1 (ordenadores del aula)
- Se requiere un Mac que ejecute High Sierra 10.13.6 o macOS Catalina 10.15.1 o posterior.
- Macincloud: Mac en la nube que corre en un navegador (20\$ al mes). Hay una versión gratuita para un par de días.
- Máquina virtual
- Es recomendable un Mac con Procesador i5 o i7, con 8 o 16 GB de Ram.
- Cuenta de apple para descargar Xcode

[apple.com/es/ios/app-store](https://apple.com/es/ios/app-store)



[developer.apple.com](https://developer.apple.com)

 **Developer**

[macincloud.com](https://macincloud.com)



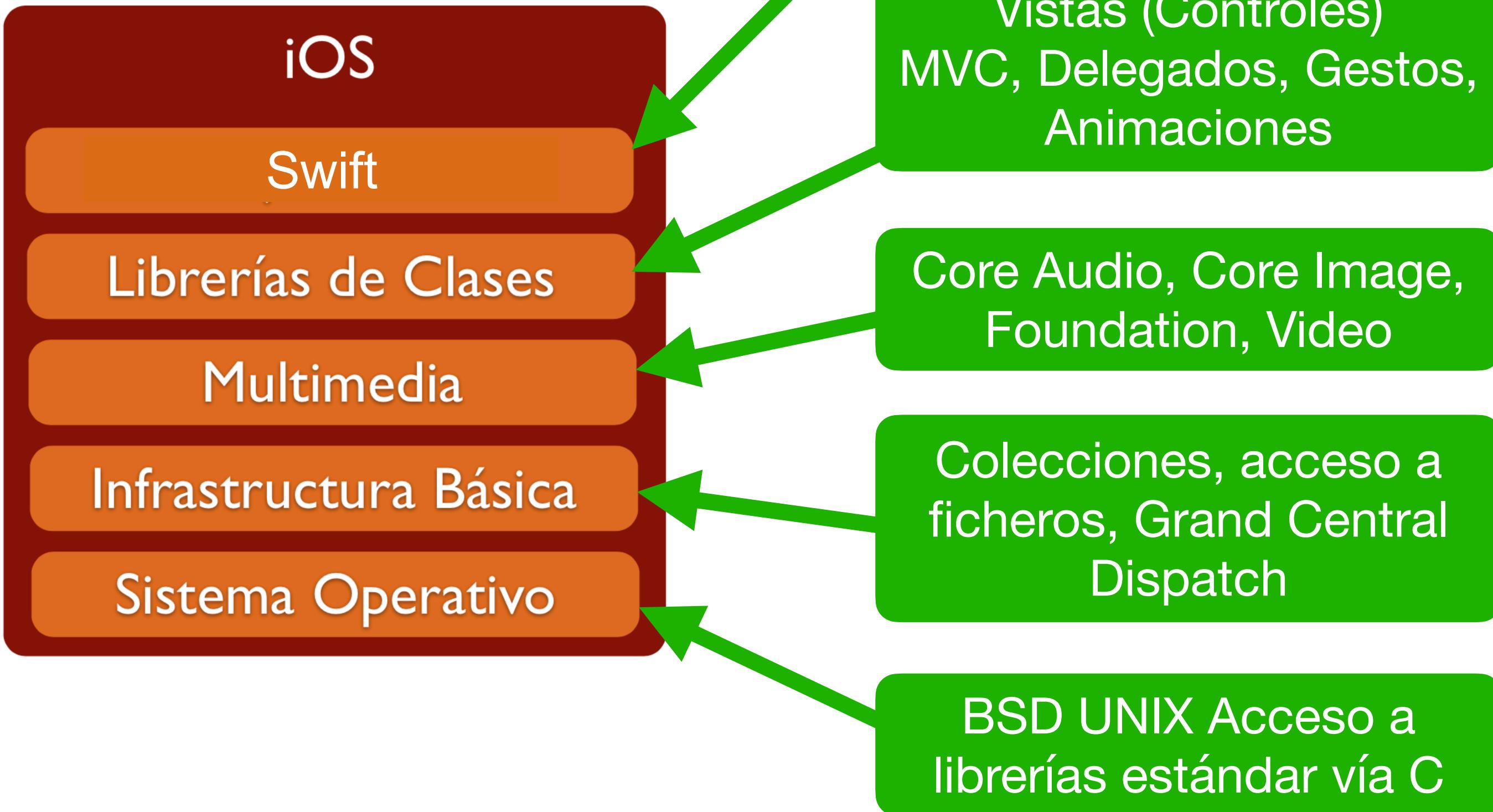
# iOS

- En realidad, es un Unix BSD con micro núcleo Mach.
- Núcleo similar a OSX, WatchOS & tvOS.
- Podríamos usar todas las librerías de C de Unix y BSD...

# Herramientas

- Lenguaje: Swift
- IDE: Xcode
- Frameworks: Foundation, UIKit, Core Data, Core Animation, Core Image, etc...

# Capas del SO



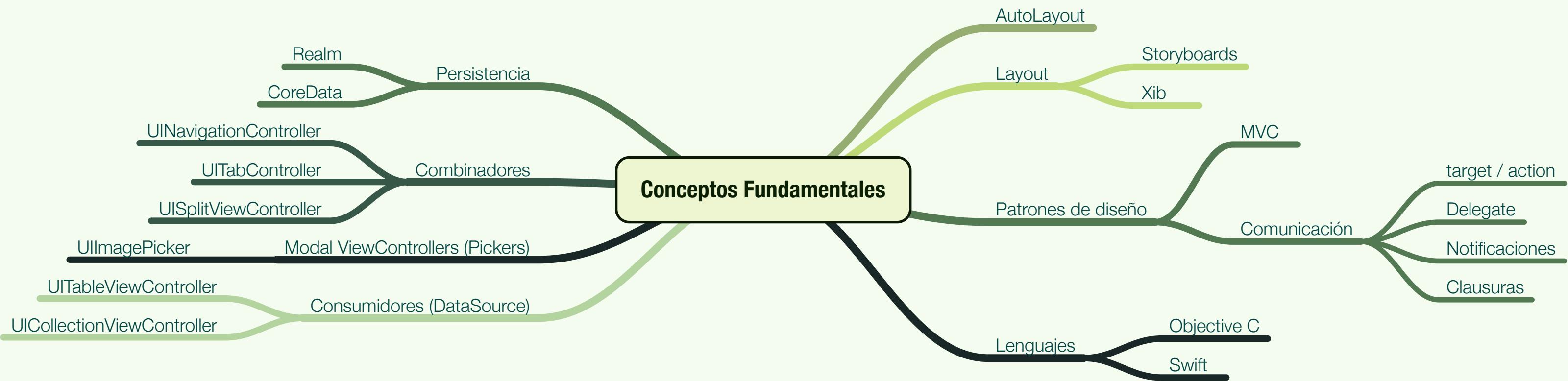
# Recomendaciones

- Mediante el Monitor de actividad controlaremos el uso de CPU y la memoria usada por Xcode y por el simulador. Conviene cerrar todos los programas que no se usen.
- Para lanzar las apps en el iPhone hay que estar dados de alta en el portal Developers de Apple (no es necesario pagar la licencia) y añadir cuenta de desarrollador a Xcode.
- <https://www.adictosaltrabajo.com/2018/05/17/vinculacion-de-iphone-a-xcode-y-aplicacion-hola-mundo-en-swift-4-1/>
- Para poder compilar hay que tener a misma versión de iOS en iPhone y en Xcode (Deployment Target). Para ello descargamos la última versión del SO en el móvil o bajamos el Deployment Target de Xcode.

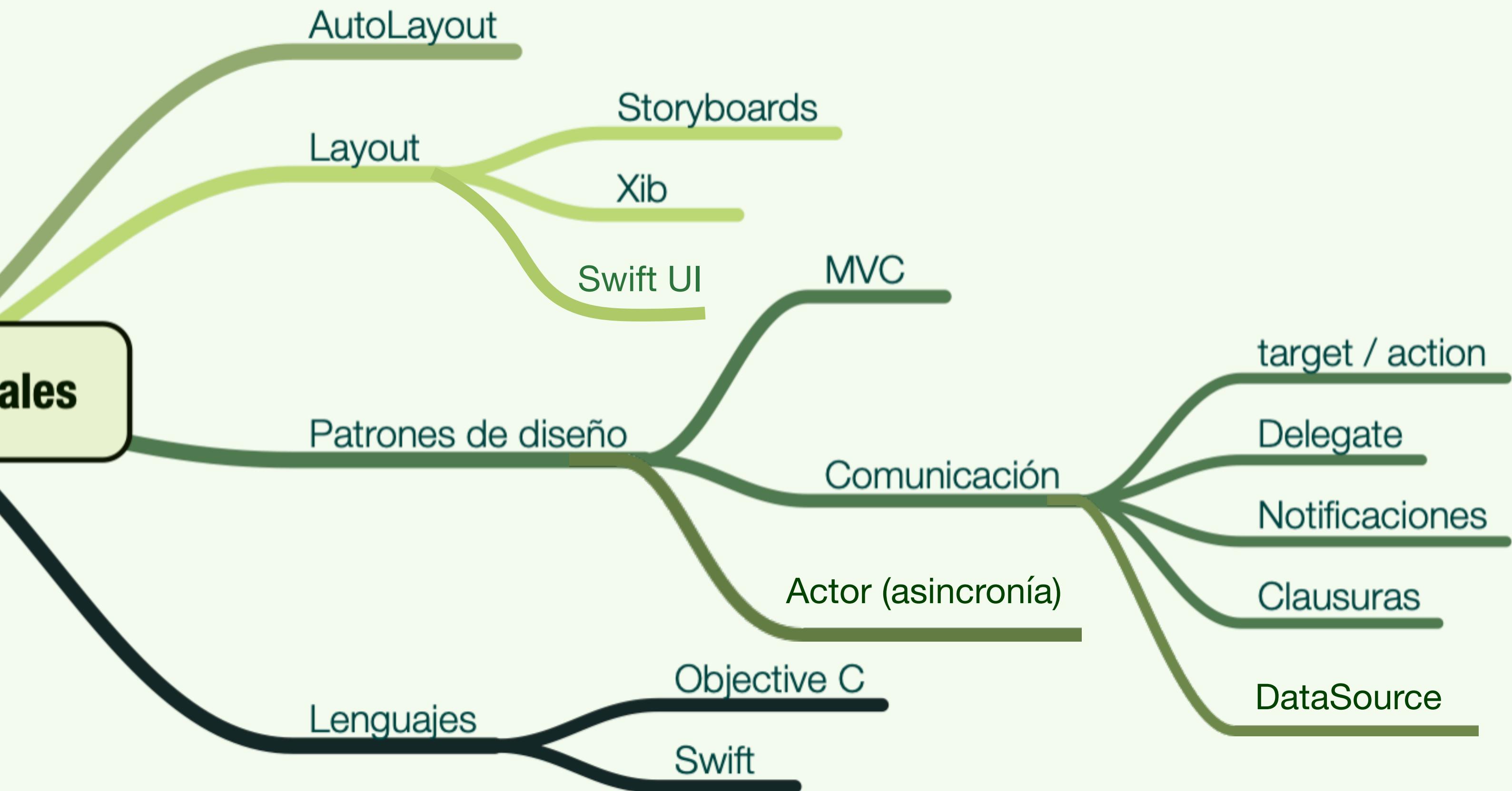
# Recomendaciones

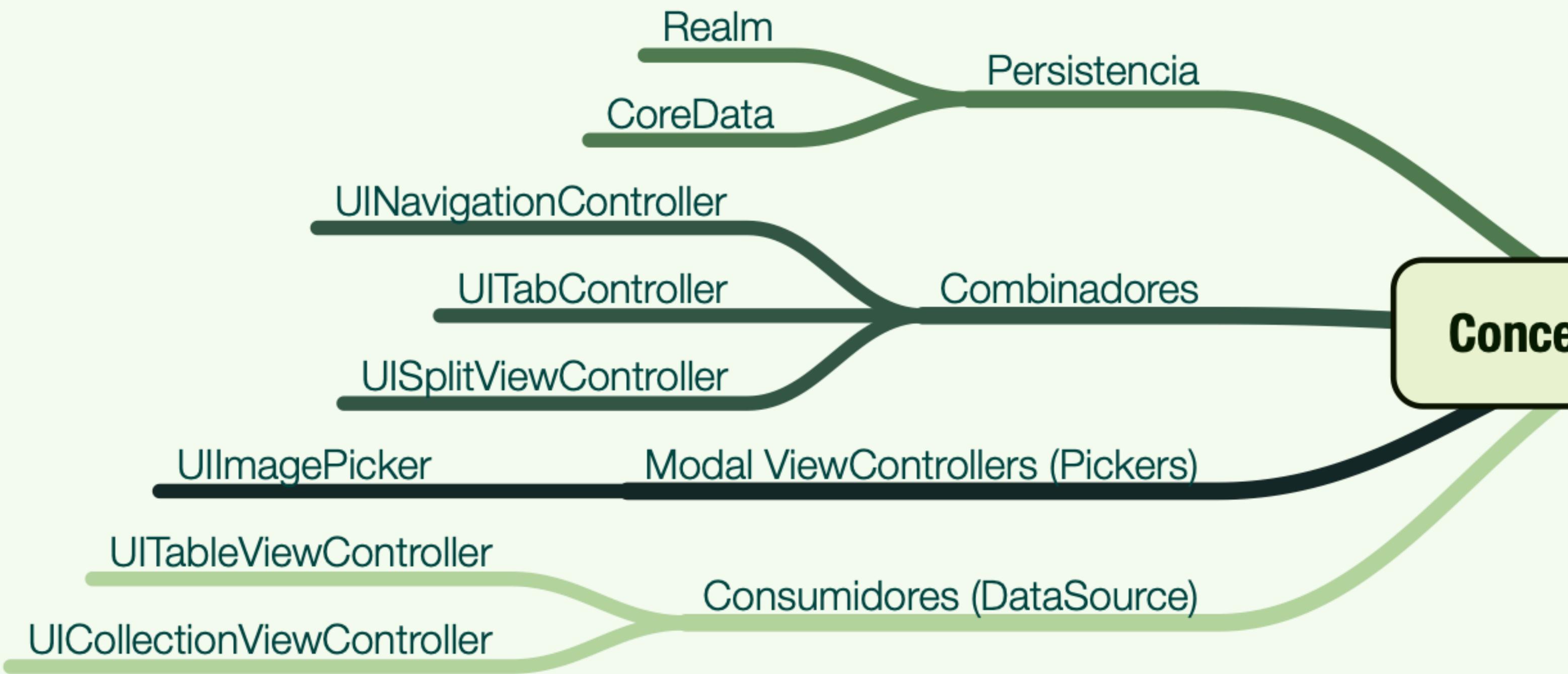
- Para diseño de apps: **Sketch, XD, Photoshop, illustrator, etc.**
- Licencia de desarrollador son 100€ anuales. Permite subir las apps al AppStore o utilizar en el dispositivo funcionalidades avanzadas para compilar en el dispositivo (realidad virtual, realidad aumentada, coreML, probar compras, conectarlos a un servidor de Apple, etc. (Apple Developer Program)).
- Mediante QuickTime podremos ver el iPhone en la pantalla del mac

## Conceptos Fundamentales



# **Conceptos**





# Qué es Foundation

- Framework multiplataforma creada por NeXT.
- Contiene tipos básicos como cadenas, fechas, colecciones, etc.
- Base sobre la cual se construye toda App del ecosistema Apple.
- Se ha traducido entera de Objective C a Swift.

# Un vistazo a Swift

- Fuertemente tipado y estricto. Todo tiene que estar oleada y sacramentado en tiempo de compilación.
- Filosofía muy parecida a la C++ & Java: es C++ con sintaxis más sencilla.
- Muy alejado de Objective C.
- Si os gusta que el compilador os grite, os va a encantar Swift!.

# Las dos caras de Swift

Light Side	Dark Side
OOP	FP
Herencia	Protocolos
Clases	Structs
enums “normales”	enums “raras”

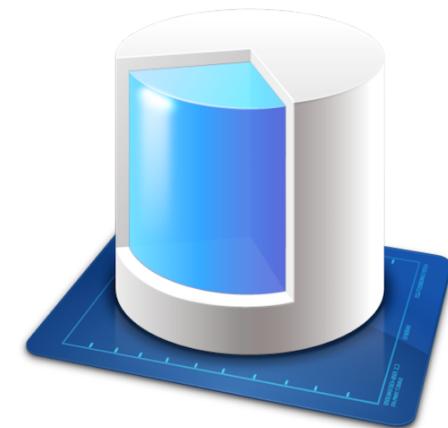
# Comenzamos con Xcode y Swift

- Creamos un ‘playground’ para aprender la teoría de Swift.
- Creamos varios ‘proyectos’ básicos para familiarizarnos con Xcode.

**App: Personajes de**

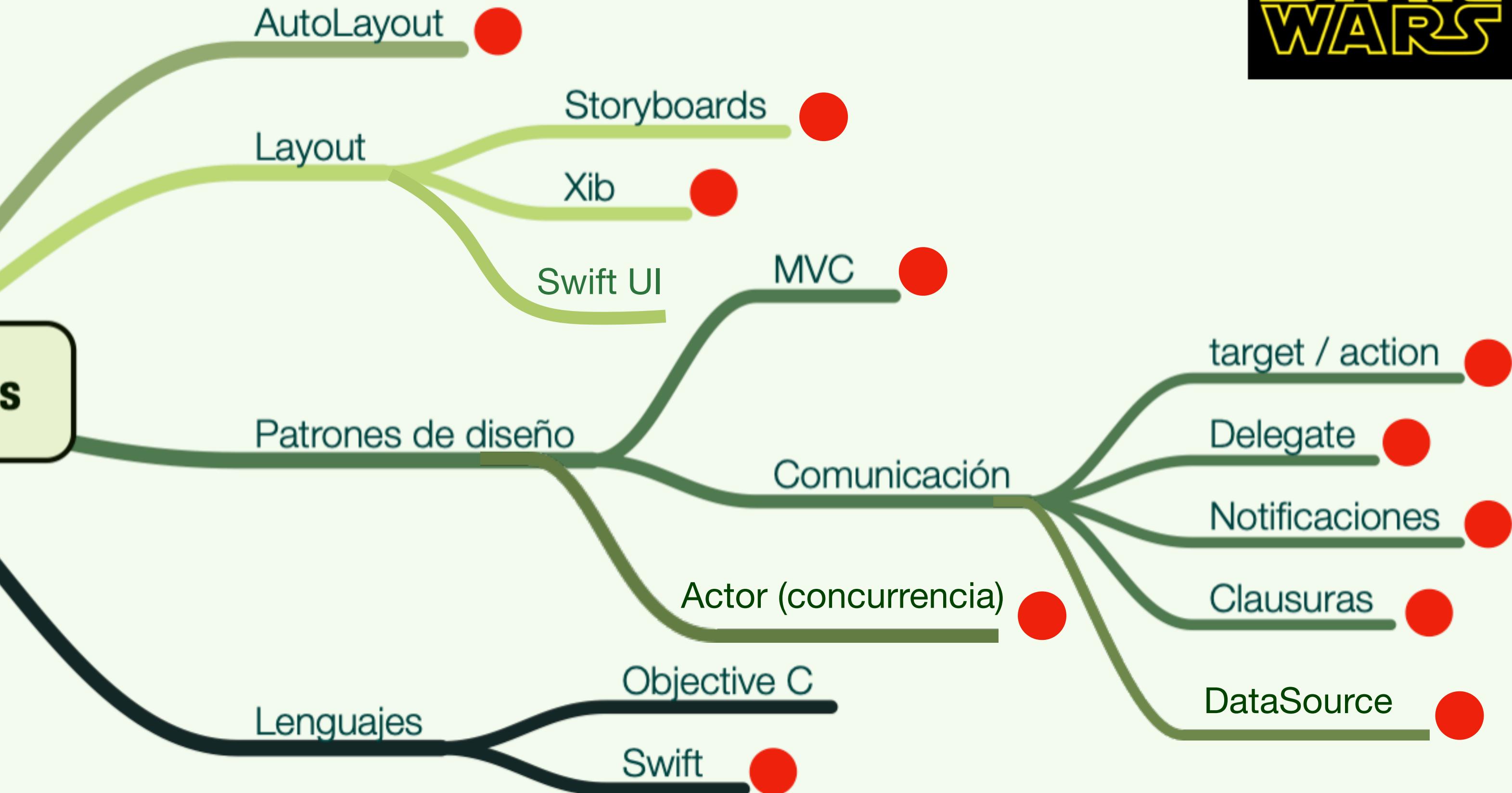


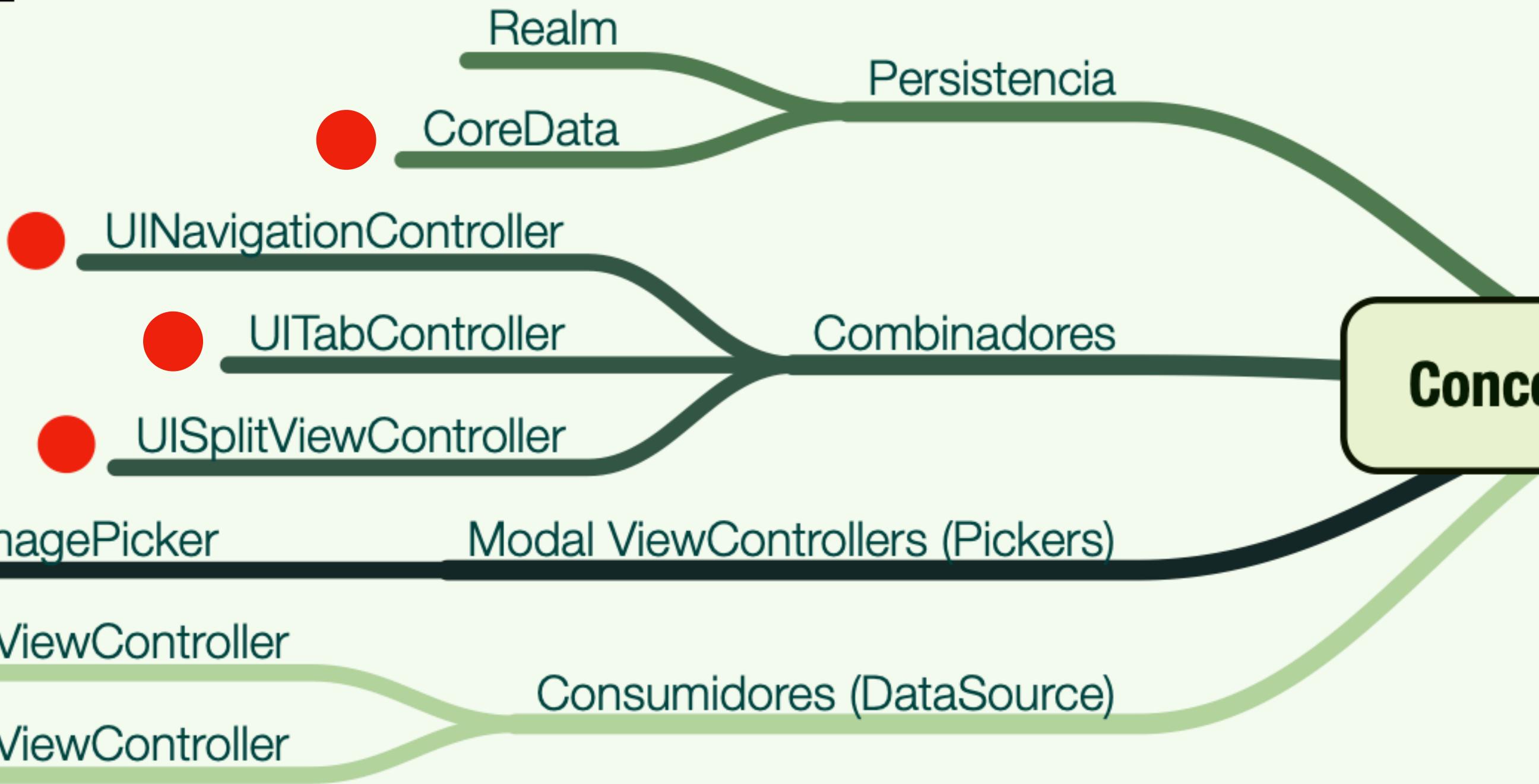
**App: Agenda con CORE DATA**



**Apps: Variedad de ejemplos**

**Playground: Teoría lenguaje SWIFT**





MVC

**Repartimos todos nuestras clases en 3 equipos**

Controlador

Modelo

Vista

# MVC

**Presenta el modelo al usuario**

Controlador

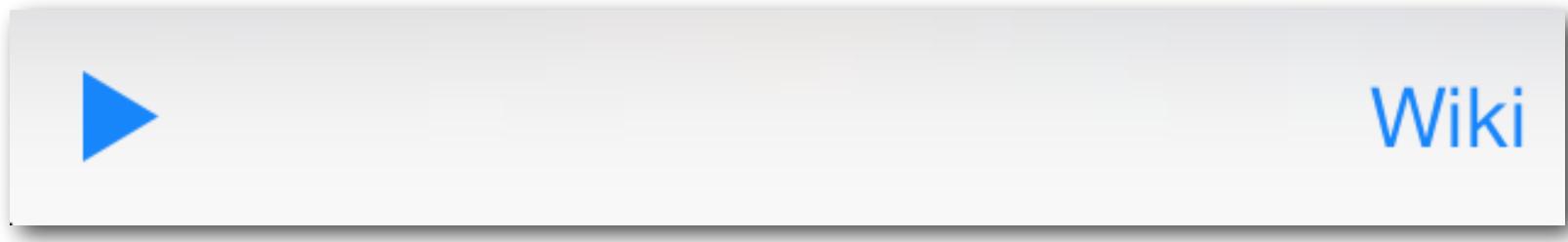
Modelo

Vista

**El alma de tu aplicación**

**La tropa de esclavos  
que usa el controlador**

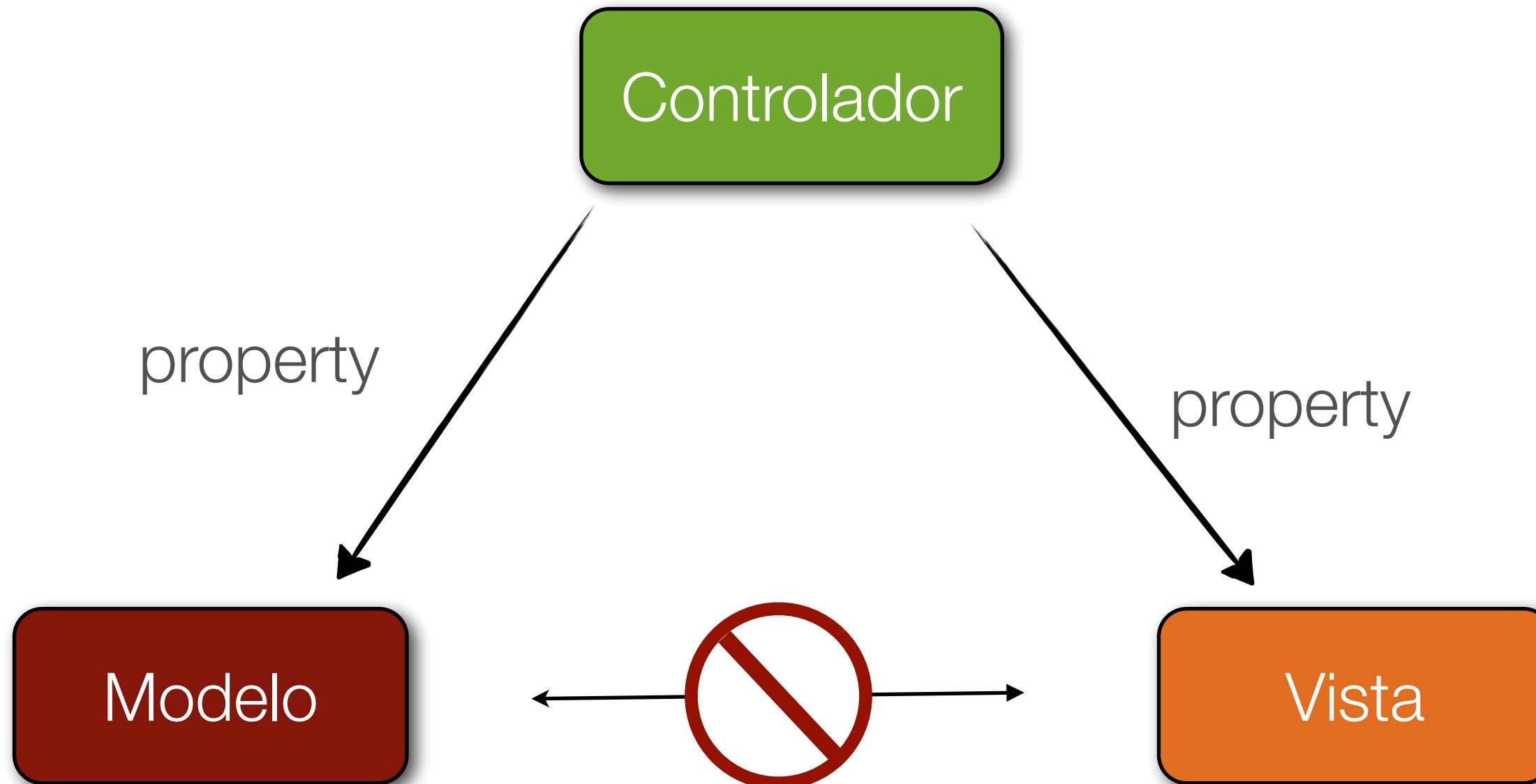
# Ejemplo de Vistas



## **5 vistas en total:**

- 1 barra de navegación
- 2 botones
- 1 imageView
- 1 label

# Comunicación entre los equipos



# Comunicación Vista Controlador

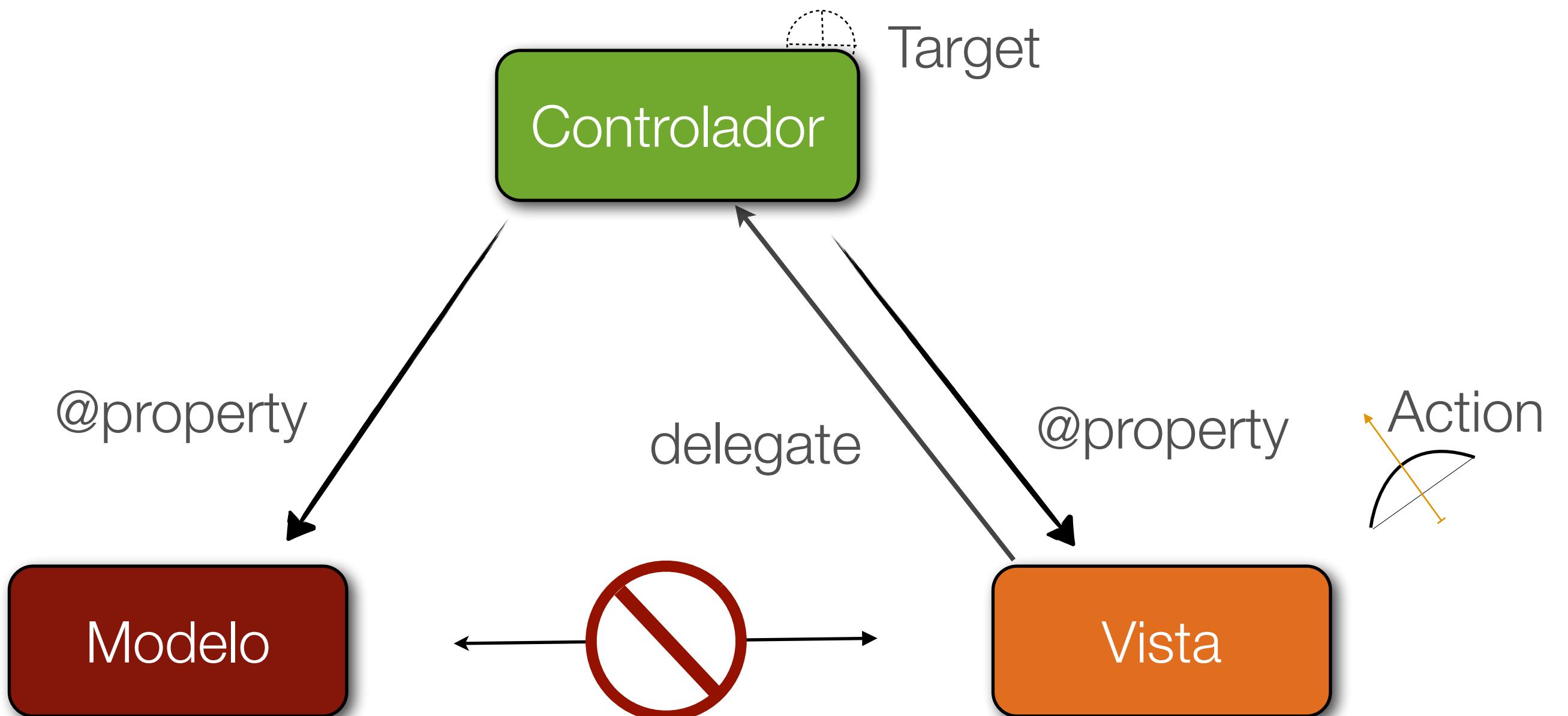
- >> Son un botón (UIButton) y han hecho clic sobre mí.
- >> Soy una UIWebView y han hecho clic sobre un enlace. ¿Lo puedo cargar?
- >> Soy una UIWebView y voy a cargar un enlace.
- >> Soy una UIWebView y acabo de cargar un enlace. Que lo sepas.

# Comunicación Vista Controlador

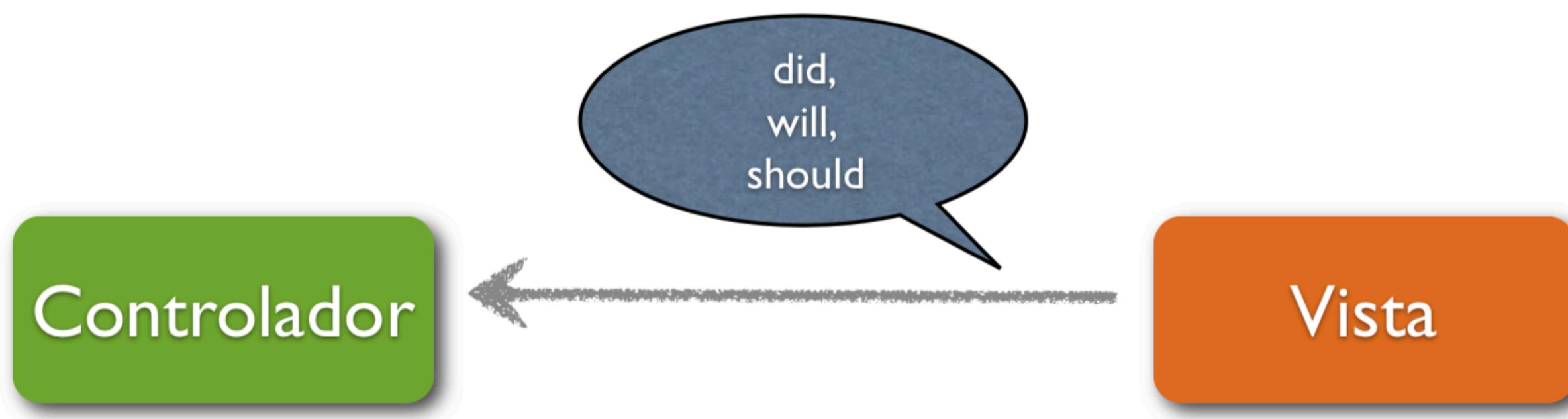
Hay dos formas de comunicarse:

- Target/Action: cuando ocurra algo interesante, al objeto Target le envías el mensaje Action.
- Delegate: cuando necesites más información sobre cómo dibujarte o reaccionar ante las acciones del usuario, me pides más información.

# Comunicación entre los equipos



# Delegate

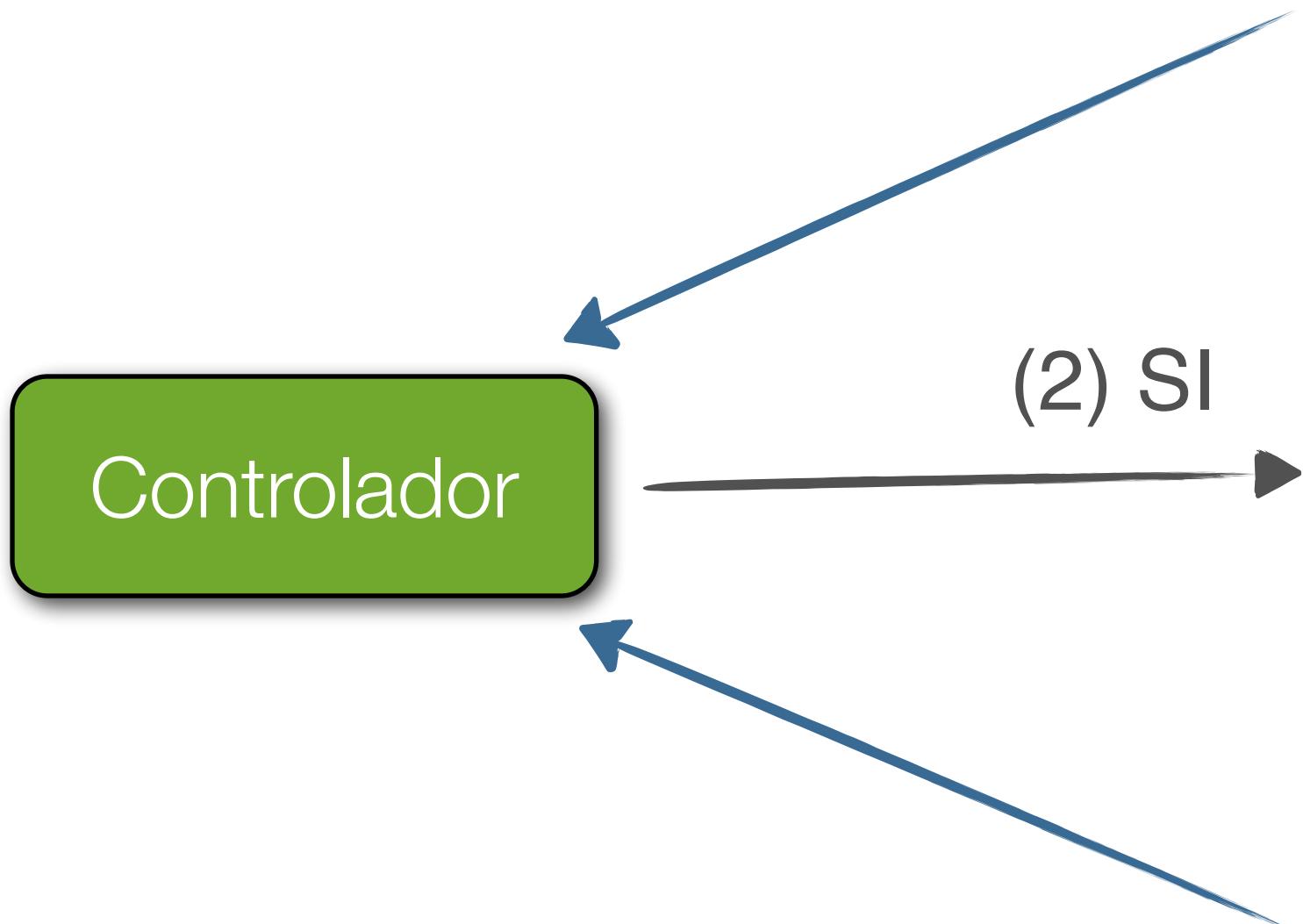


El controlador es el delegado (ayudante) de la vista.

La vista envía mensajes informando de lo que hace y pidiendo ayuda cuando hace falta.

# Ejemplo de Delegate

(1) Soy una UIWebView y han hecho clic sobre un enlace. ¿Lo puedo cargar?



(3) Soy una UIWebView y acabo de cargar un enlace. Que lo sepas.



## **PATRÓN DEL DELEGADO - Michael Knight con su reloj y Kit (coche fantástico)**

- Capacidades de Kit: Volar, nadar, disparar, romper muros, etc.
- Pero él por si mismo no sabe hacer nada, ni sabe tampoco cuándo hacerlo.
- Michael Knight es su delegado.
- Kit le pregunta a Michael qué tiene que hacer cuando llega un muro (Solo puede preguntar las preguntas ya implementadas, no cualquier cosa).
- Michael Knight tiene que tener un protocolo (reloj) para saber responder a Kit.
- Michael Knight le responde al reloj ordenando lo que hay que hacer (una serie de respuestas ya implementadas, no cualquier respuesta).
- Kit rompe el muro.

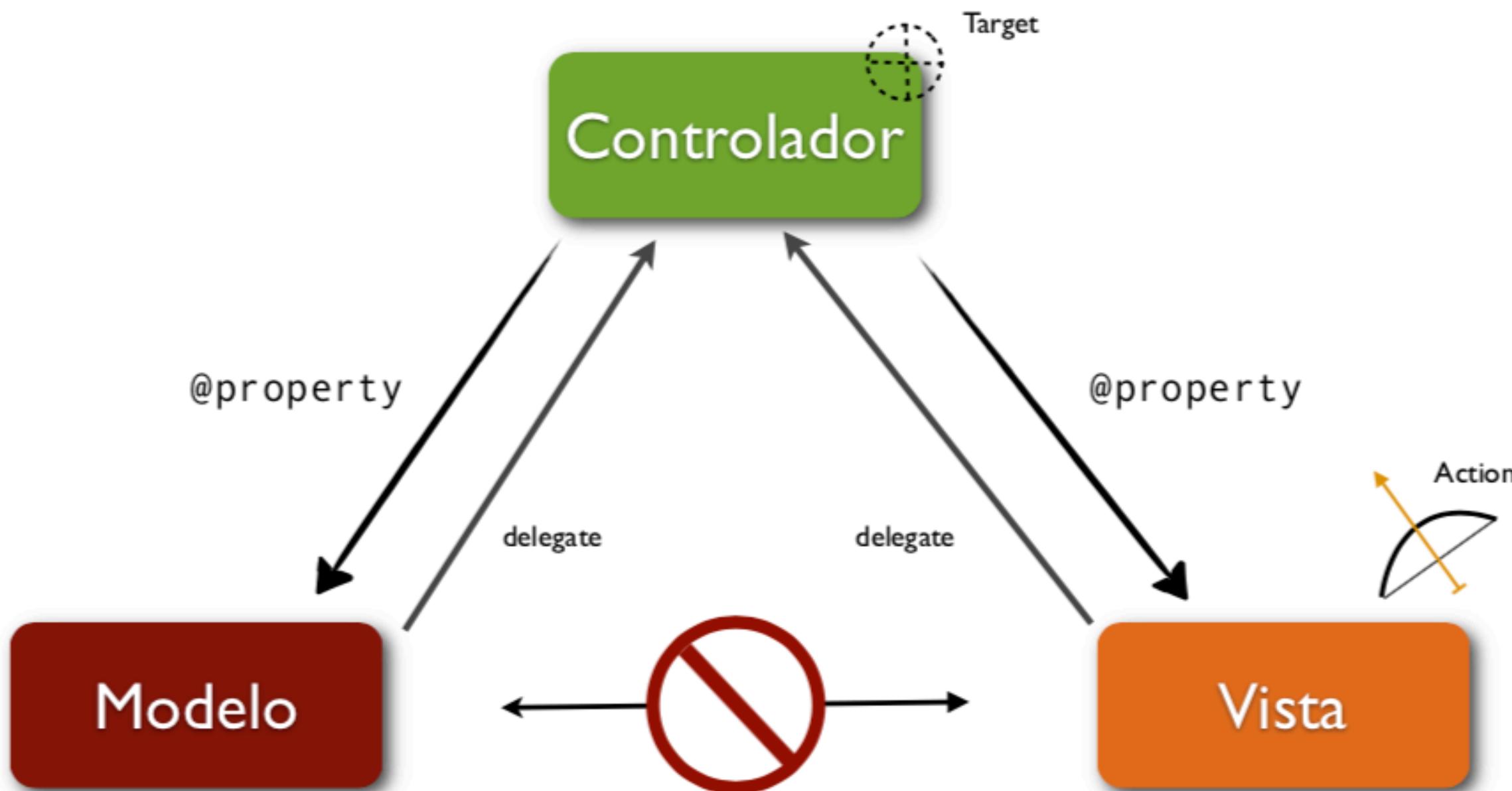


# Comunicación Modelo Controlador

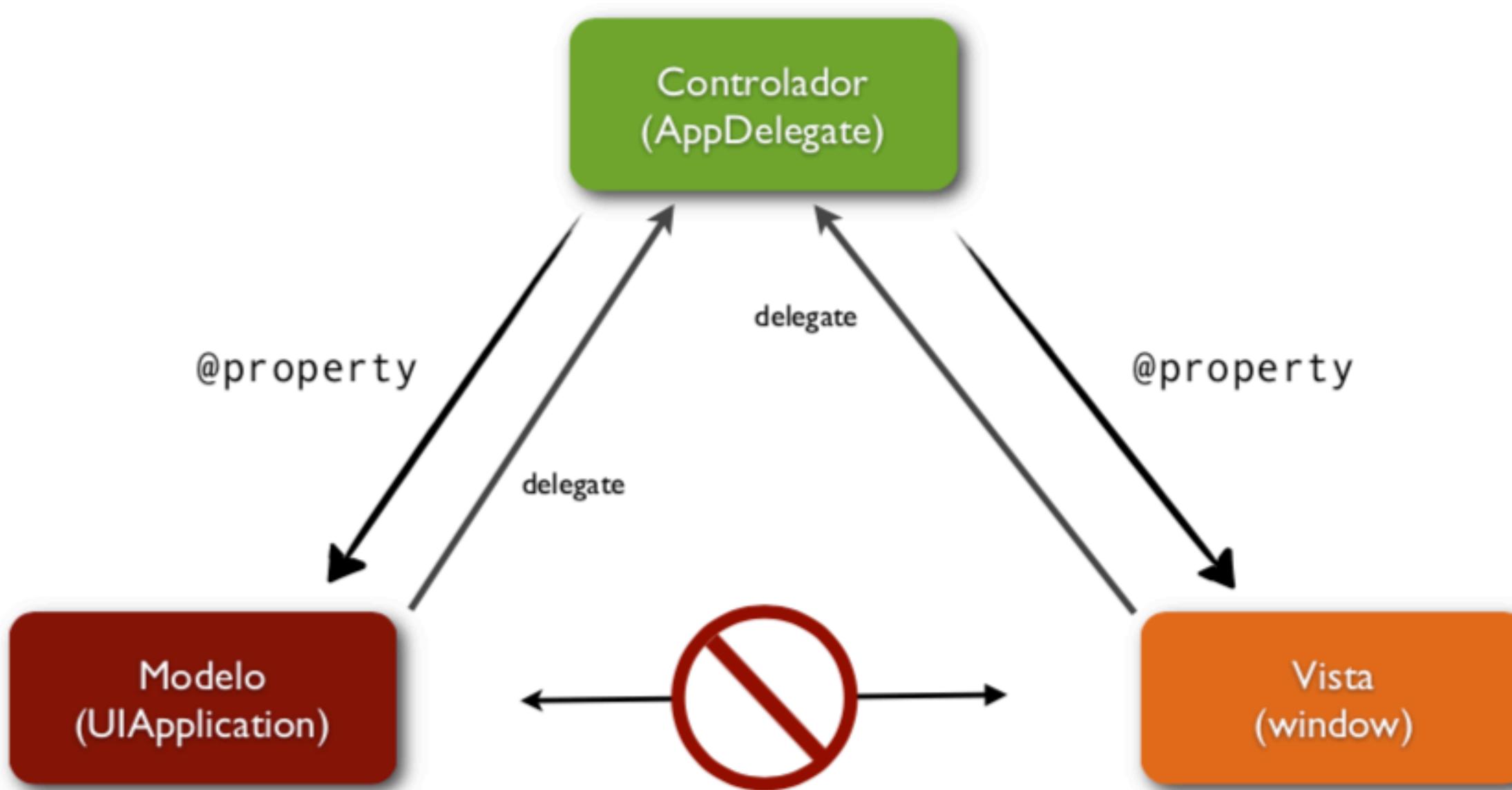
¿Puede el modelo tener algo que decir al controlador?

- Toda modificación que sufra el modelo debe de ser notificada al controlador.
- Por ejemplo, en un programa de bolsa, si el valor de la acción cambia, tendrá que notificar al controlador.
- Slider del audio del menú superior
- La forma más común de comunicación modelo-controlador es mediante notificaciones. Las veremos más adelante.
- De momento, usaremos el delegado, es decir, el controlador será delegado del modelo.

# Resumen MVC



# Empty Application



# AppDelegate

```
import UIKit

// Aquí arranca la aplicación. Se hacen dos cosas:

// 1º-> Crea una instancia de la clase UIApplication. Este objeto representará nuestra aplicación y es el responsable de que podamos comunicarnos con el Sistema operativo. Para ello necesita que exista la clase AppDelegate, su delegado, quien será el encargado de entender los mensajes que envía la aplicación.

// 2º-> Luego entra en un run loop (bucle) en el que está a la espera de la interacción del usuario. Aquí es donde también se activan las aplicaciones gráficas

@UIApplicationMain

// La clase AppDelegate desciende de UIResponder e implementa el protocoloUIApplicationDelegate.

// UIResponder es una clase que permite a sus descendientes, como AppDelegate, que puedan responder a eventos.

// UIApplicationDelegate: Este protocolo lo tiene que incorporar quien quiera ser delegado de una aplicación. Así podrá entender los mensajes que se envían el Sistema Operativo y la aplicación.

class AppDelegate: UIResponder, UIApplicationDelegate {
```

# AppDelegate

```
class AppDelegate: UIResponder, UIApplicationDelegate {  
  
    // Xcode crea una instancia de UIWindow. (En SceneDelegate de xcode 11.5)  
    // Todas las vistas descienden de UIView. UIWindow es una de sus subclases. Su  
    particularidad es que ocupa toda la pantalla. Es opcional porque habrá un momento en el  
    que la app no tendrá todavía la window creada.  
  
    var window: UIWindow?  
  
    func application(_ application: UIApplication, didFinishLaunchingWithOptions  
launchOptions: [UIApplicationLaunchOptionsKey: Any]?) -> Bool {  
  
        // La aplicación acaba de arrancar y nos permite construir la aplicación desde  
        aquí en vez de desde el Mainstoryboard.  
  
        /* Lo que hace Xcode por debajo es:  
         1.- Redimensiona la vista window en toda la pantalla.  
         2.- La app se inicializa con el ViewController del mainStoryboard como root.  
         3.- La clase ViewController se asocia a un interface visual (xml) en el que  
        podremos fácilmente dibujar y poner vistas  
         4.- Por defecto viene con una View de color blanco.  
         5.- Lo muestra en pantalla  
        */  
  
        return true  
    }  
}
```

(En SceneDelegate de xcode 11.5)

```
func applicationWillResignActive(_ application: UIApplication) {
```

// Cuando la aplicación deja de estar activa, por ejemplo cuando llaman por tfn, se lanza este mensaje y sirve para que le demos las órdenes que queramos que se ejecuten en ese momento.

```
}
```

```
func applicationDidEnterBackground(_ application: UIApplication) {
```

// Cuando la aplicación se vaya a segundo plano hay 5 segundos para ejecutar lo que le digamos, por ejemplo, guardar los datos, detener sonidos, cerrar conexiones a red, etc. Después la app quedará congelada y no tendrá acceso al procesador (no puede hacer nada). Pero no está cerrada, está en 2º plano.

```
}
```

```
func applicationWillEnterForeground(_ application: UIApplication) {
```

// Cuando la app esta apunto de volver a primer plano (habrá que recuperar lo que habíamos guardado en el DidEnterBackground, etc.)

```
}
```

```
func applicationDidBecomeActive(_ application: UIApplication) {
```

// Aquí ya esta activa la aplicación (ya podrá recibir toques del usuario): Refrescaremos el interface de usuario, etc.

```
}
```

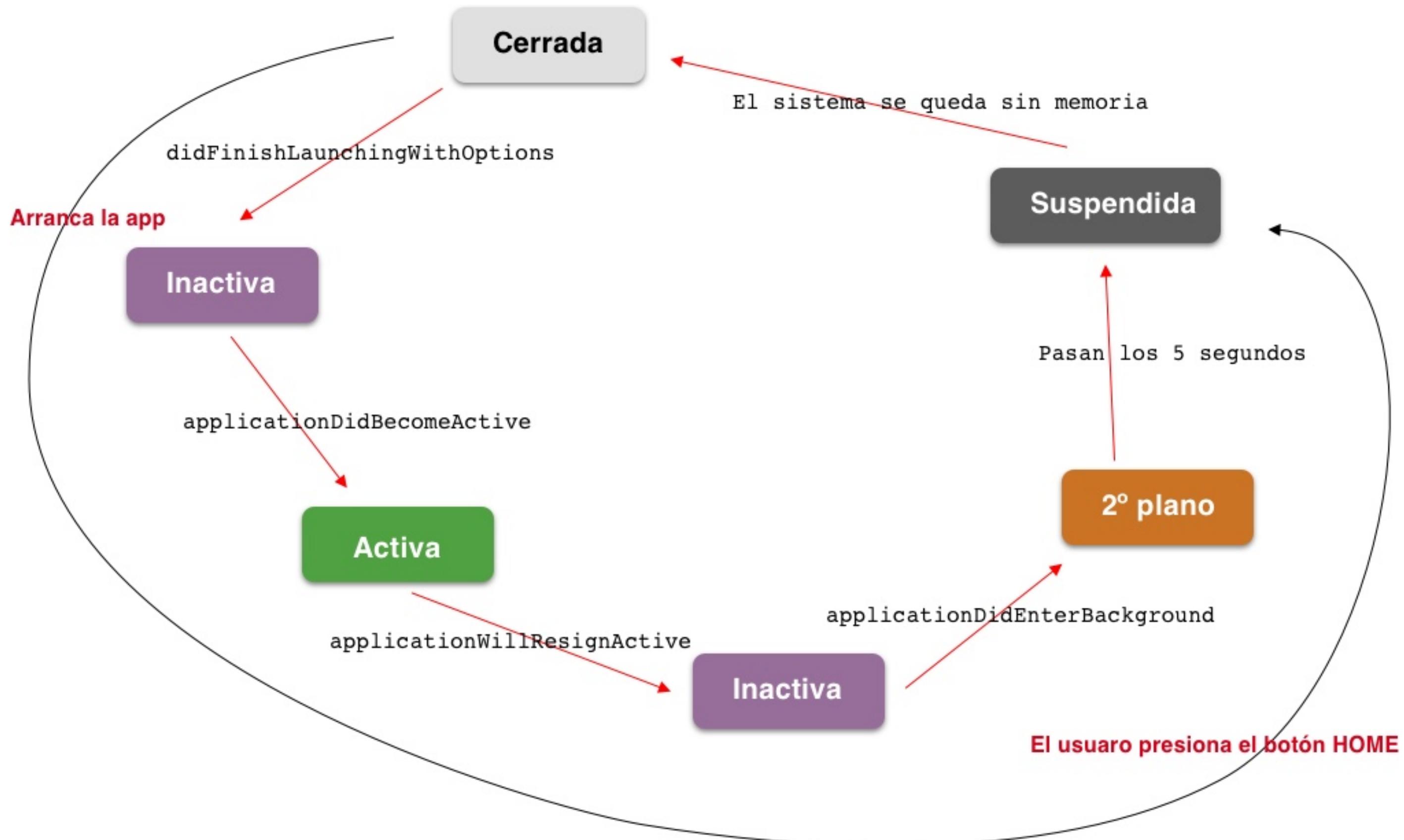
```
func applicationWillTerminate(_ application: UIApplication) {
```

// Cuando la aplicación va a terminar definitivamente, por cualquier causa. Lo más importante que hay que hacer aquí es salvar los datos

```
}
```

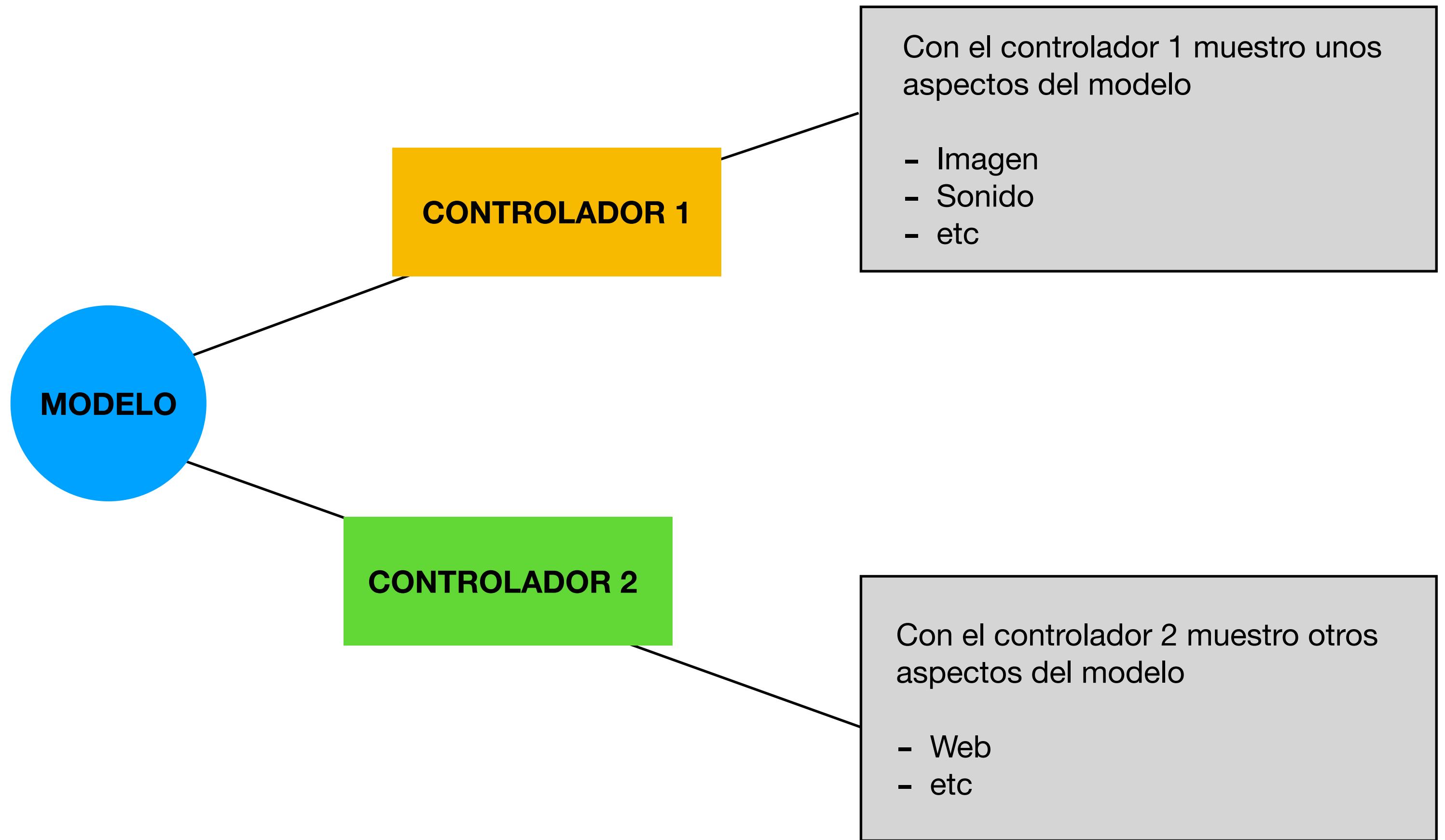
```
}
```

# Ciclo de vida de la App



# Ciclo de vida de la Vista





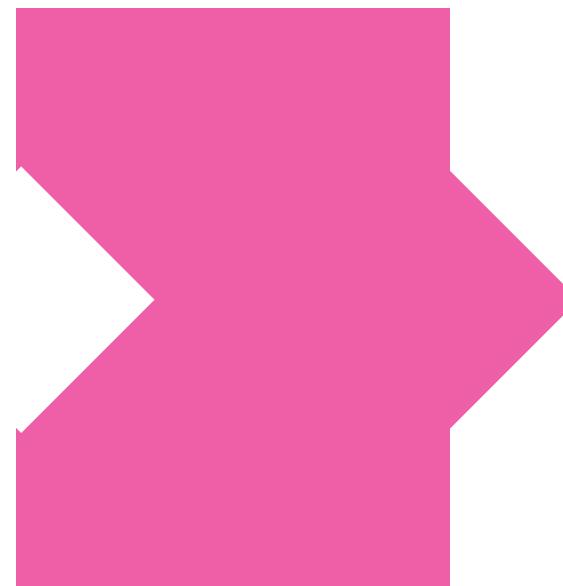
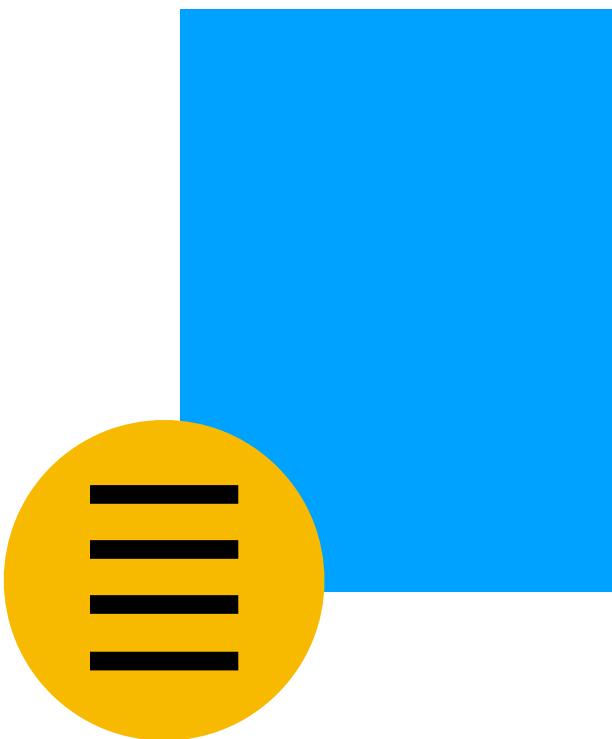
# Concepto de Delegado

- El delegado es un objeto que ayuda a otro.
- Cuando queremos añadir más cosas a una clase lo podemos hacer con herencia o con delegados.
- En Cocoa se usan mucho los delegados y protocolos, y no se hereda tanto como en Java.
- Quien quiere ser delegado de otro debe implementar un protocolo: Una serie de métodos que debe implementar.

## Clase WebView

La Clase A  
(controlador) será  
el **delegado** de la  
clase WebView

### Clase A



### Protocolo:

métodos que tiene  
que implementar la  
clase A

**Personalización de  
la Clase WebView**

# TableViewController

- UITableViewDelegate (will, did, should)
- UITableViewDataSource (nos pide los datos)

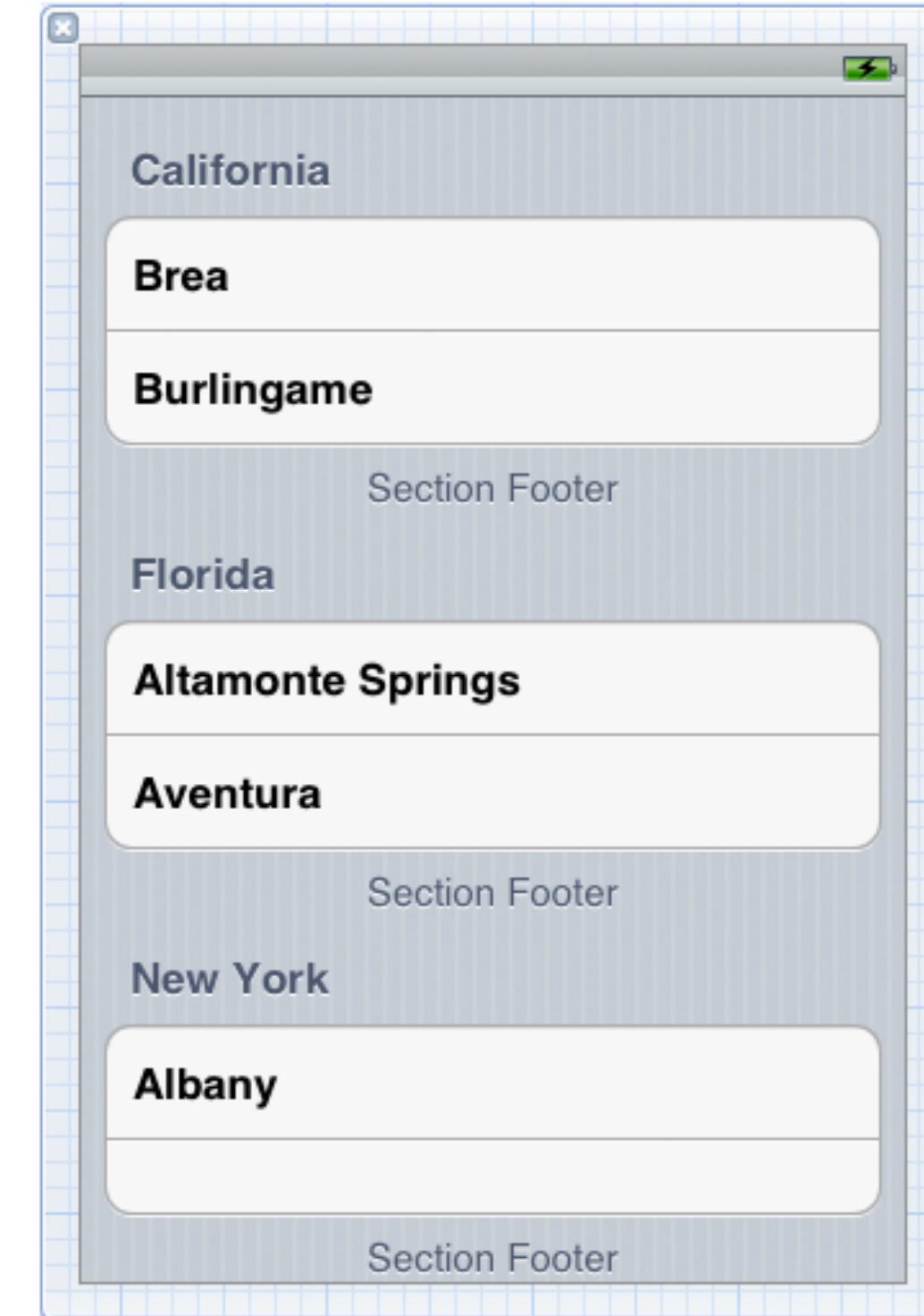
# Tipos de Tablas: Plain y Gruped



A screenshot of a plain table view. The table has a single section header "California" at the top. Below it is a list of ten city names: Brea, Burlingame, Canoga Park, Carlsbad, Chula Vista, Corte Madera, Costa Mesa, Emeryville, Escondido, and a section footer "Section Footer". The table is styled with a light blue grid background and rounded corners for the cells.

California
Brea
Burlingame
Canoga Park
Carlsbad
Chula Vista
Corte Madera
Costa Mesa
Emeryville
Escondido
Section Footer

Plain Table



A screenshot of a grouped table view. The table is divided into three sections by section headers: "California", "Florida", and "New York". Each section contains a list of city names: California has Brea and Burlingame; Florida has Altamonte Springs and Aventura; New York has Albany. Each section also has a "Section Footer" at the bottom. The table uses a light blue grid background and rounded corners for the cells.

California
Brea
Burlingame
Section Footer
Florida
Altamonte Springs
Aventura
Section Footer
New York
Albany
Section Footer

Group Table

# Reciclado de celdas de la tabla

- Las tablas piden solo los datos (celdas) que se ven en pantalla y dos o tres más por arriba y por abajo. Esto optimiza la memoria.
- Cuando hacemos scroll el SO se deshace de las celdas que ya no se ven. Algunas de ellas las guarda en una caché que nos servirá para no tener que crear continuamente las vistas de celda cada vez que vuelvan a mostrarse. Lo que hacen es reciclar
- Existirá una caché para los diferentes tipos de celdas de la tabla y por eso necesitamos identificar a cada tipo de celda con un ID.
- La caché de celdas es un diccionario cuya clave es el identificador, y el valor, un array con las celdas listas para reciclar [ ID : [celdas] ]

# UITableViewCell

- Al igual que las tablas, se crean con un estilo.
- Hay 4 estilos estandar de celda
- Se pueden crear celdas personalizadas

# Anatomía de una tabla



Carrier



7:36 PM



< List

## Table View Components

SECTION HEADER #1

Title



Title



Section footer #1

SECTION HEADER #2

Title



Subtitle

Title

Detail

Title Detail



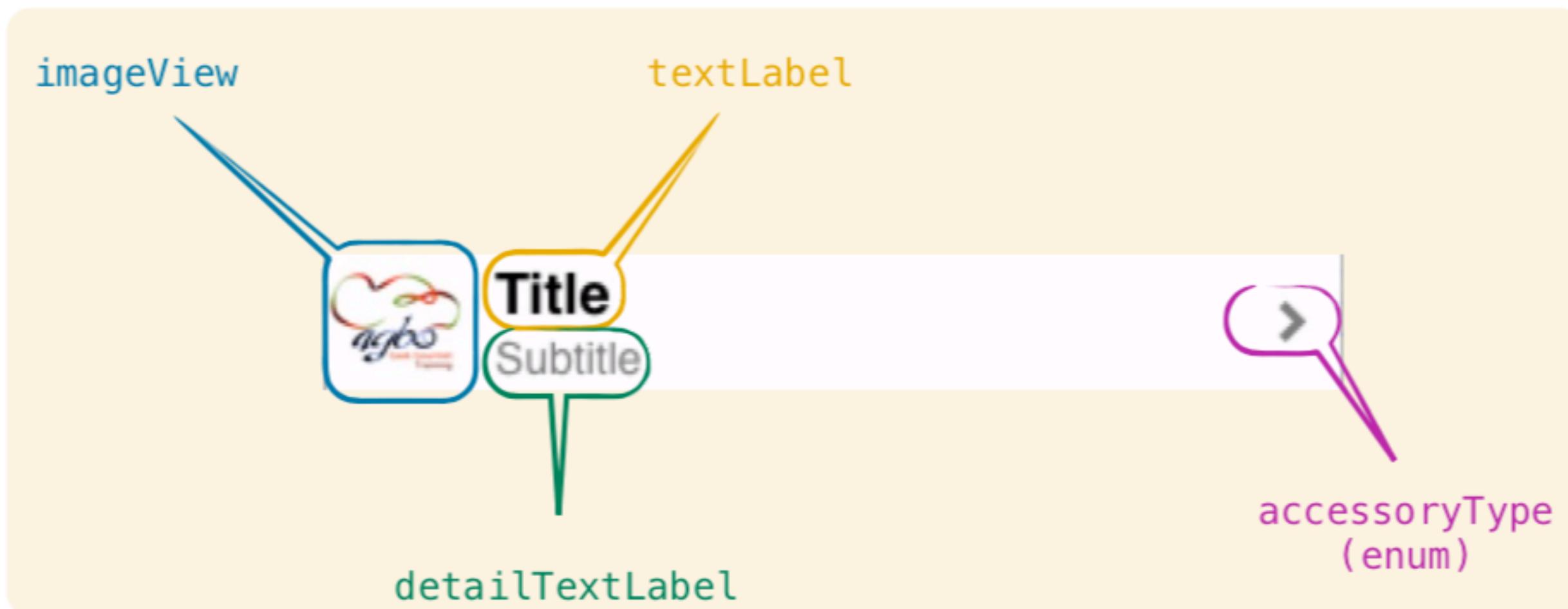
Grouped

Plain

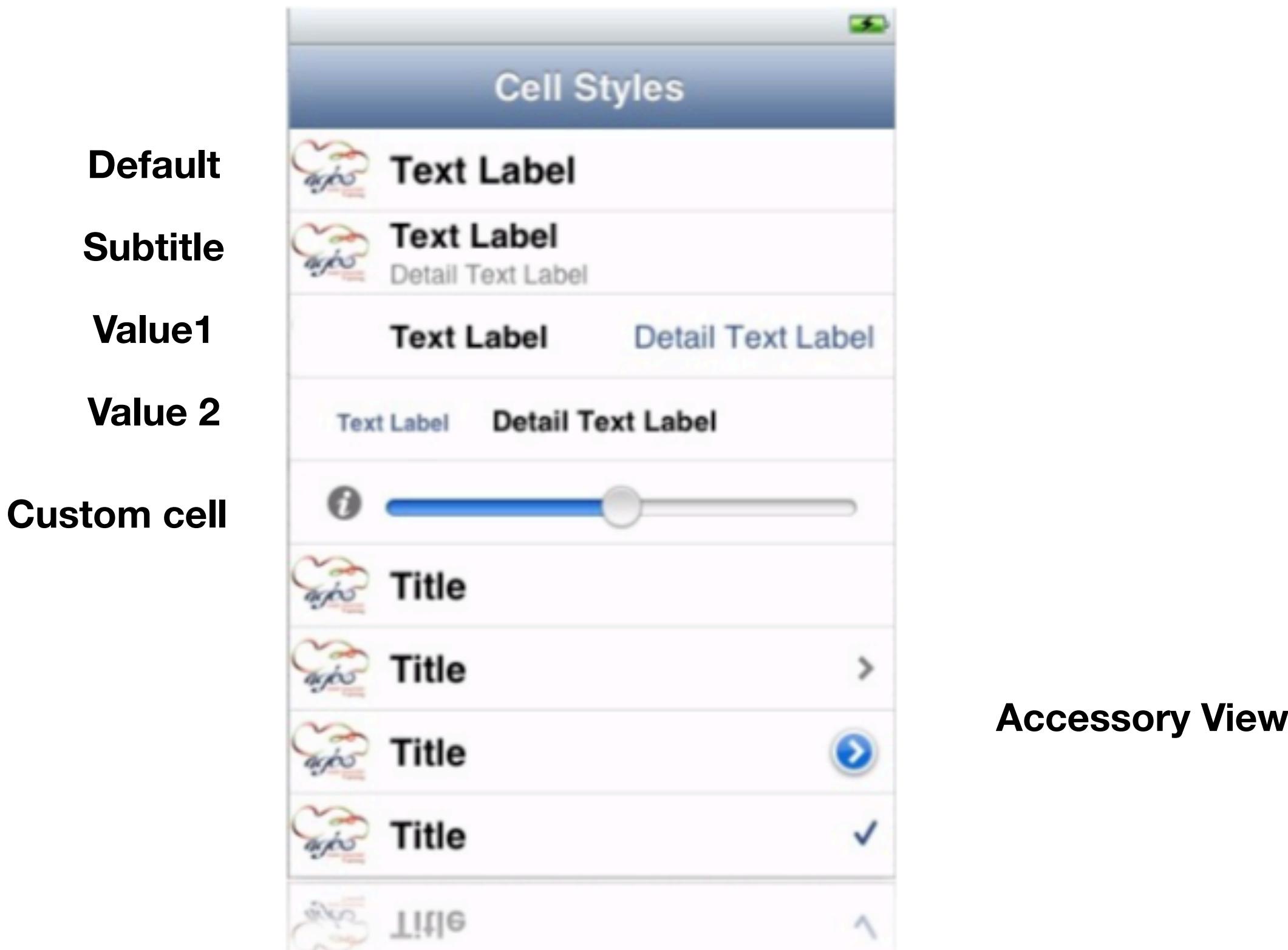
Grouped

Bare

# Anatomía de una celda



# UITableViewCell



# Uso de accessoryView

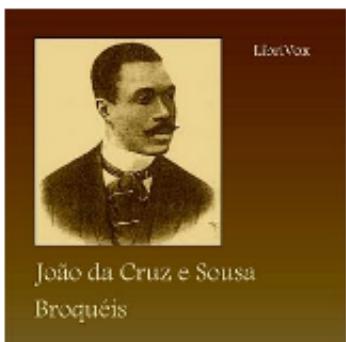


- Podemos sustituir el accessory type por una vista cualquiera, usando la propiedad accessoryView:

[Livros Grátis Produtos](#)

Castro Alves, Antonio Frederico de

Por: R\$ 0,00

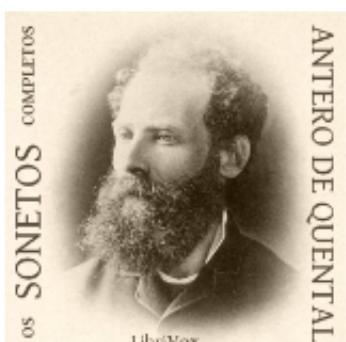


Broquéis

Livros Grátis

Cruz e Sousa, João da

Por: R\$ 0,00

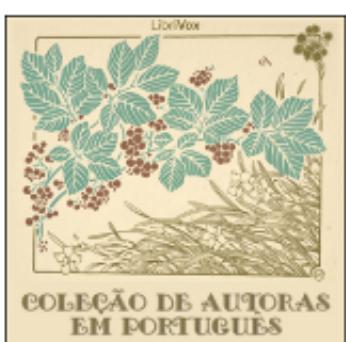


Os Sonetos Completos

Livros Grátis

Quental, Antero de

Por: R\$ 0,00



Coleção de Autoras em Português

Livros Grátis

Various

Por: R\$ 0,00



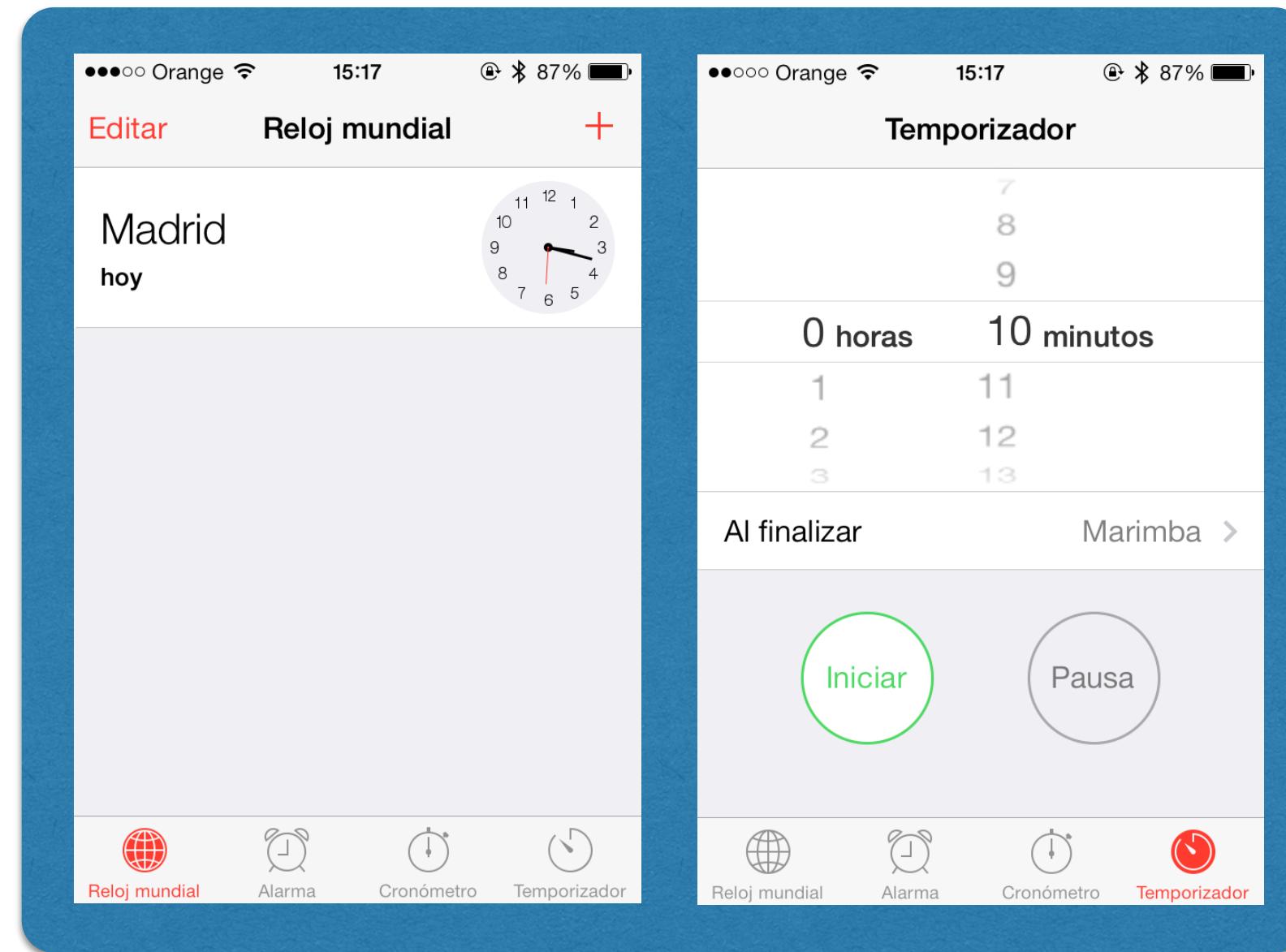
# Combinadores de MVCs

- Hay varios y el funcionamiento suele ser similar.
- TabBarController
- UINavigationController
- SplitViewController
- UITableViewController

# UITableViewController

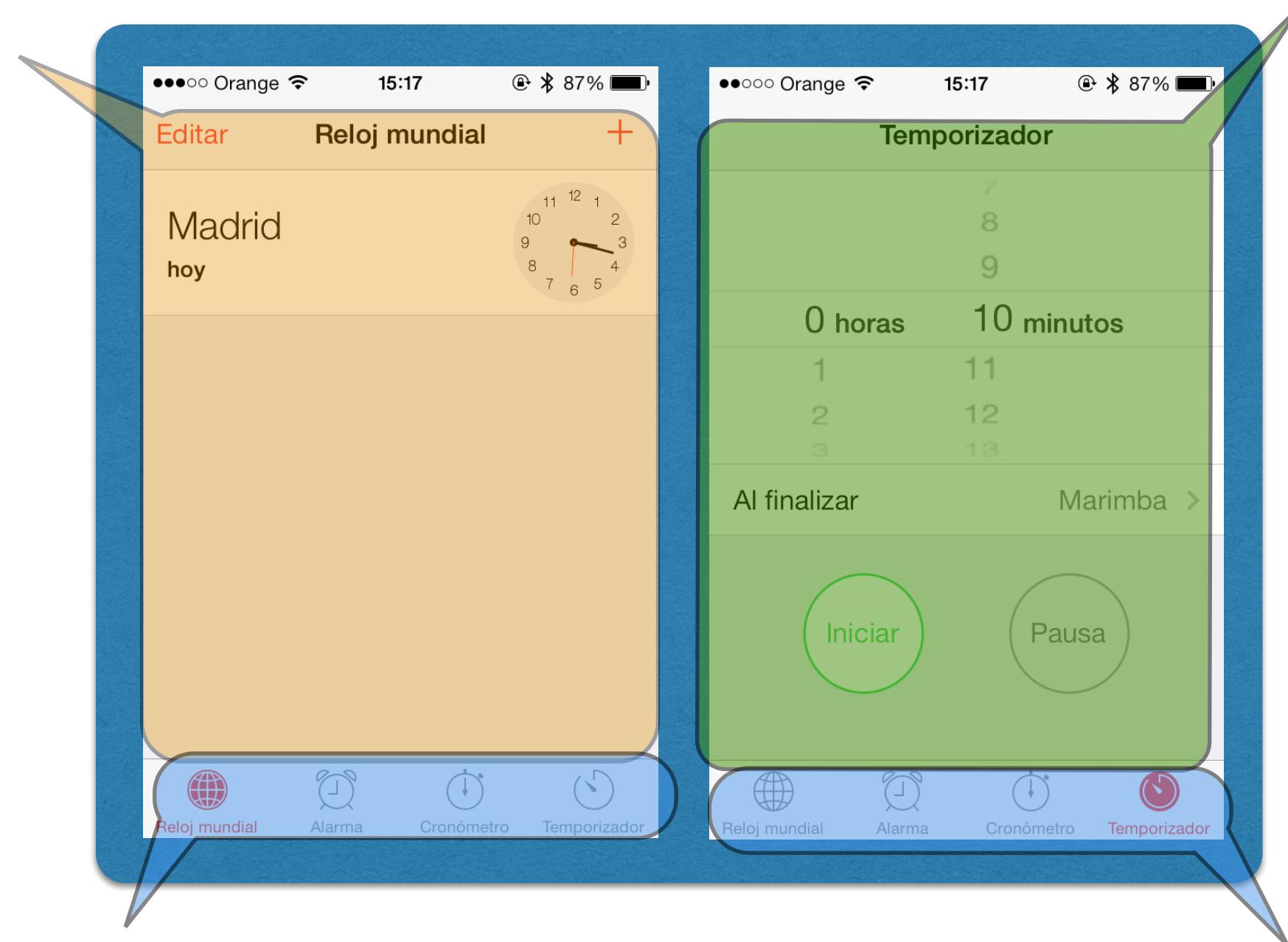


# UITabBarController



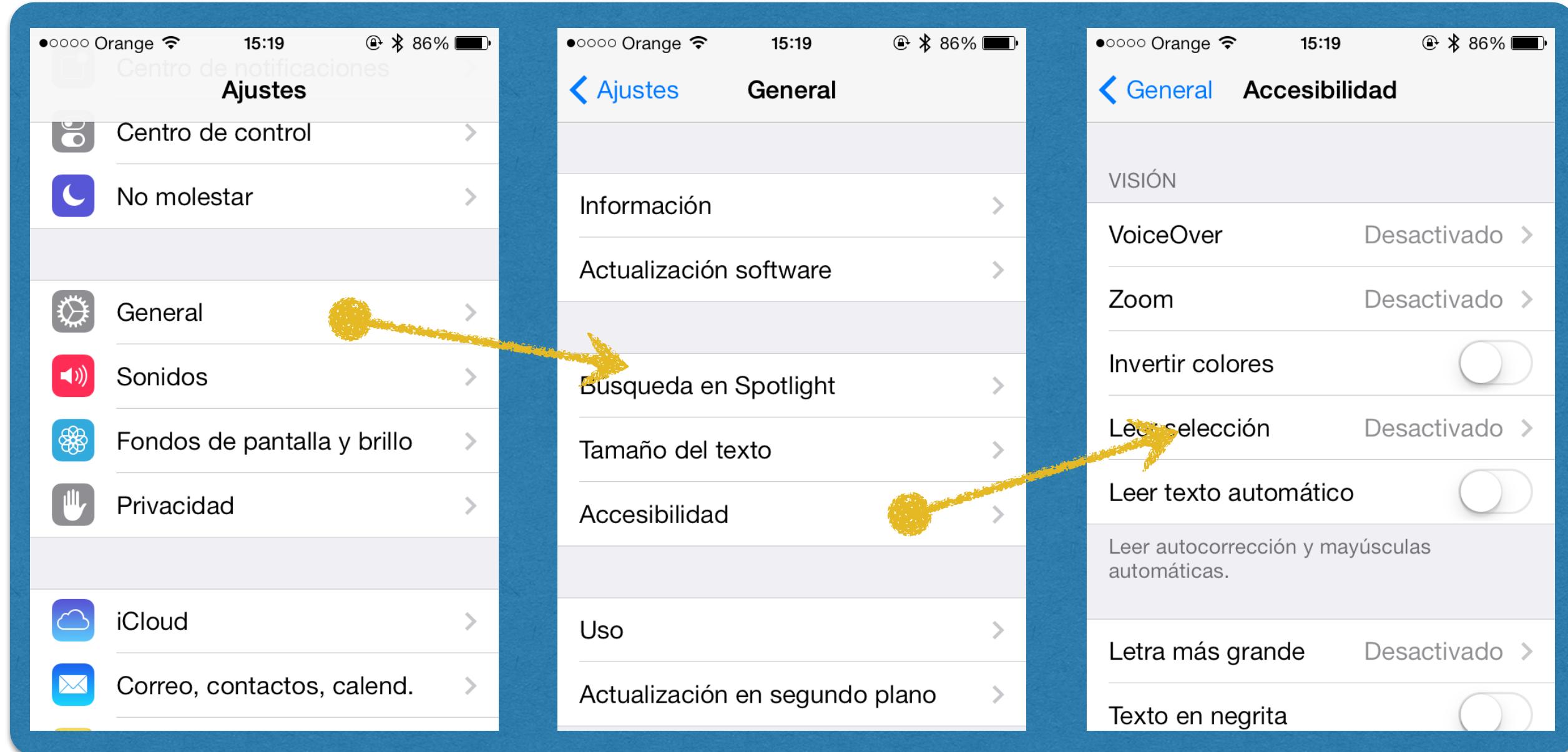
Un controlador con su vista y su modelo

Otro controlador con su vista y su modelo

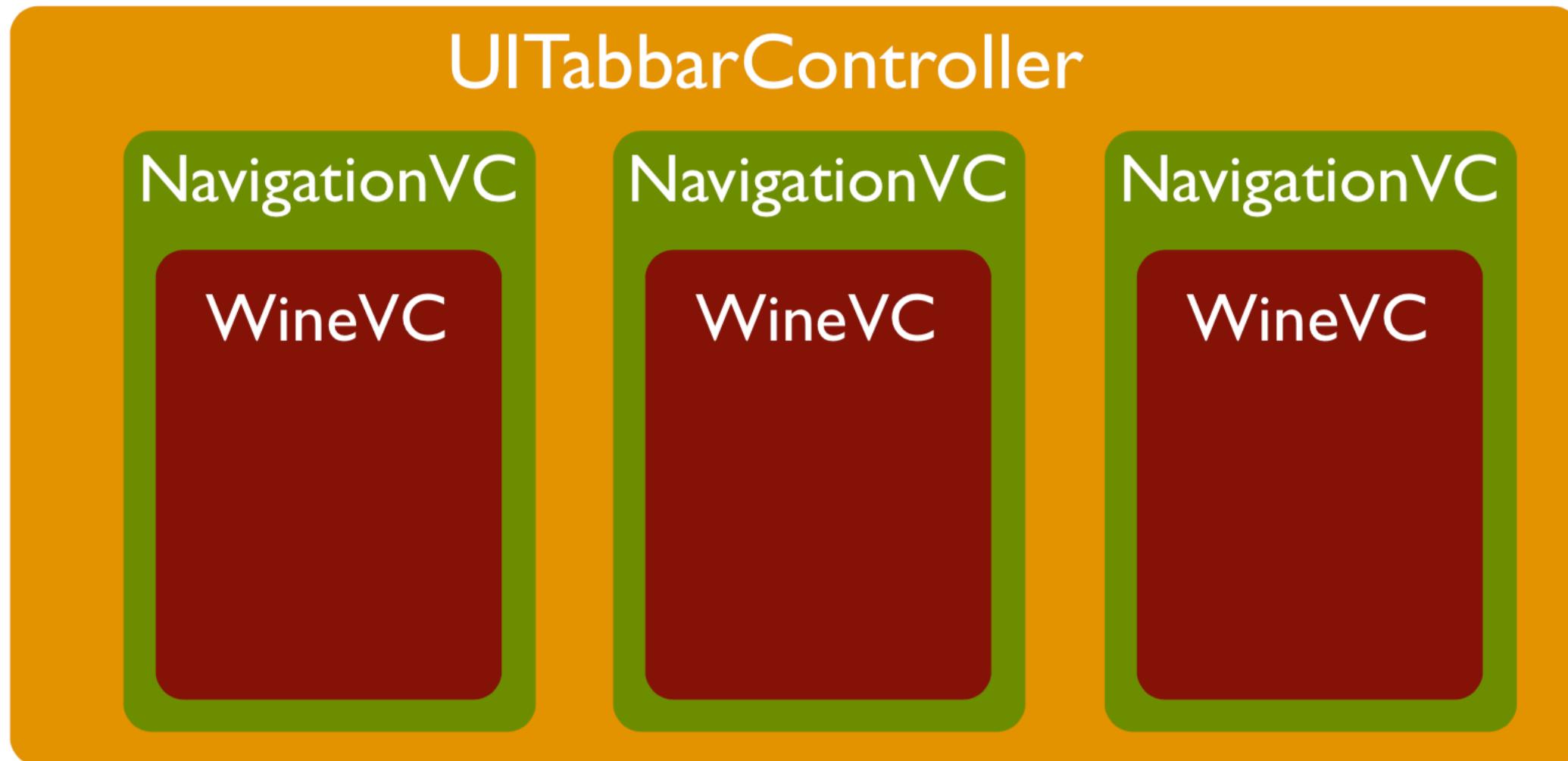


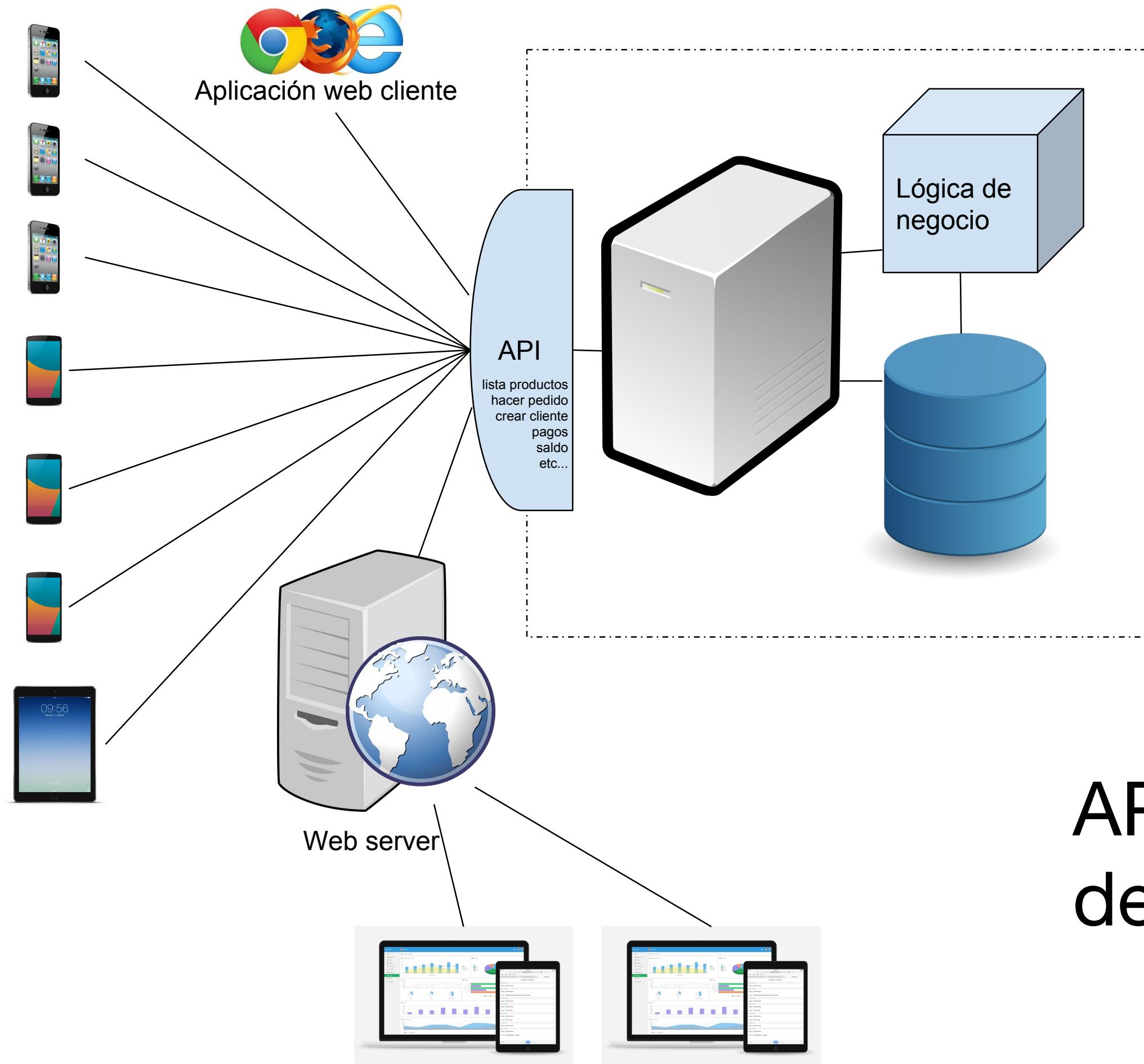
El TabBar contiene a distintos MVCS y nos permite verlos de uno en uno mediante la barra de botones inferior.

# UINavigationController



# Combinadores dentro de Combinadores





API que nos  
devuelve JSON

{



}

**JSON**

# JSON

- Javascript Object Notation.
- Representación textual de datos estructurados.
- Independiente del lenguaje.
- Está sustituyendo a xml al ser una alternativa más sencilla, rápida y que escala mejor.
- <http://www.json.org/json-es.html>

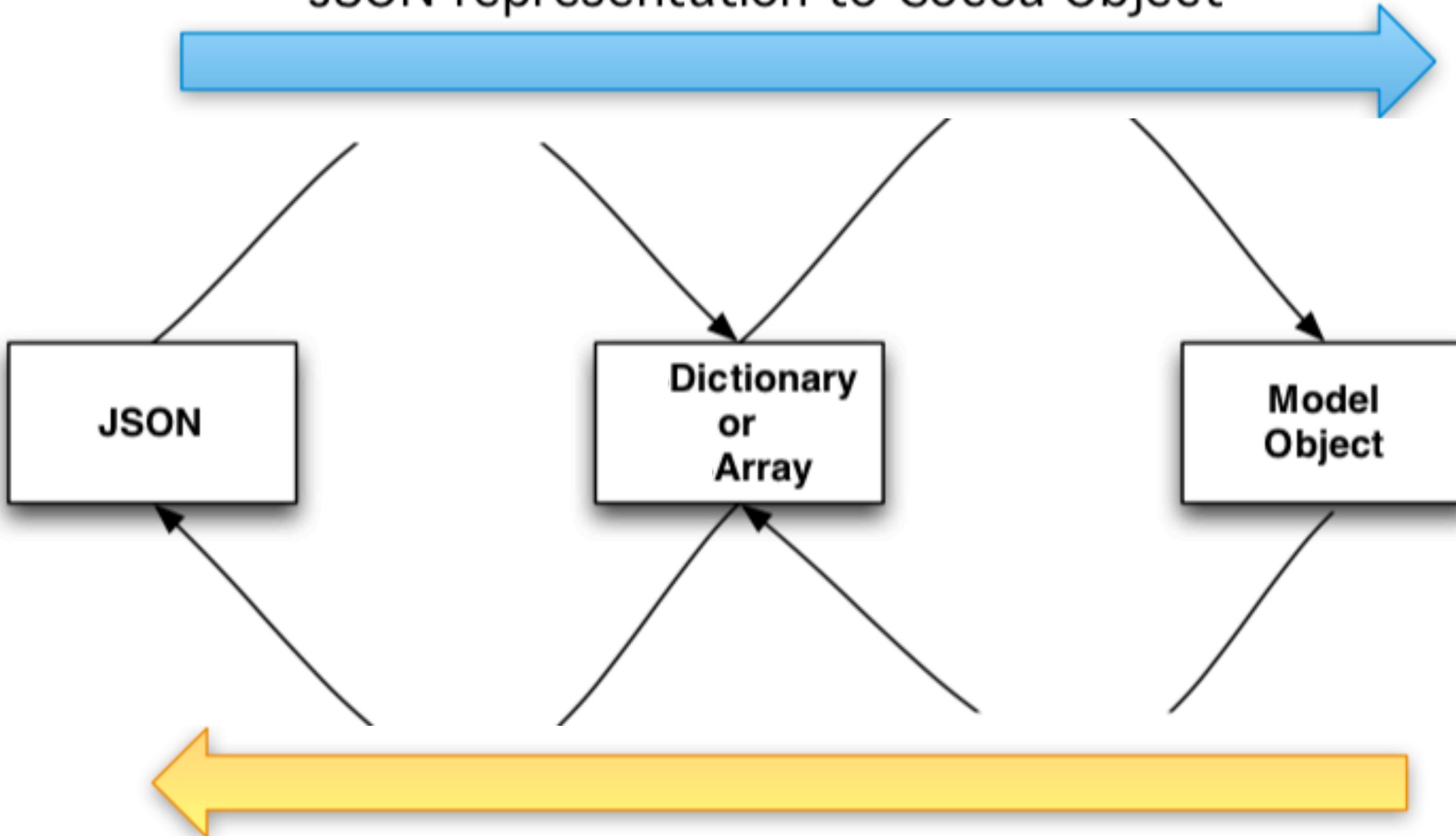
# Tipos de datos

- Número (coma flotante de doble precisión).
- Cadena: unicode entre dobles comillas y usando \ para “escape”.
- Booleanos (verdadero o falso).
- Lista (separado por comas y entre corchetes).
- Objeto: un diccionario que asocia cadenas (nombres) con valores (cualquiera de estos tipos).
- null.

# Objeto vino

```
{  
  "origin" : "RIOJA",  
  "name"   : "Blanco de Guarda Finca La Reñana 2009",  
  "company" : "Luis Alegre",  
  "grapes"  : [  
    {  
      "grape": "90% Viura"  
    },  
    {  
      "grape": "10% Malvasía"  
    }  
  ],  
  "year"   : 2009,  
  "type"   : "Blanco",  
  "rating" : 4  
}
```

JSON representation to Cocoa Object



Serialize Cocoa Object to JSON representation

# UISplitViewController al Detalle

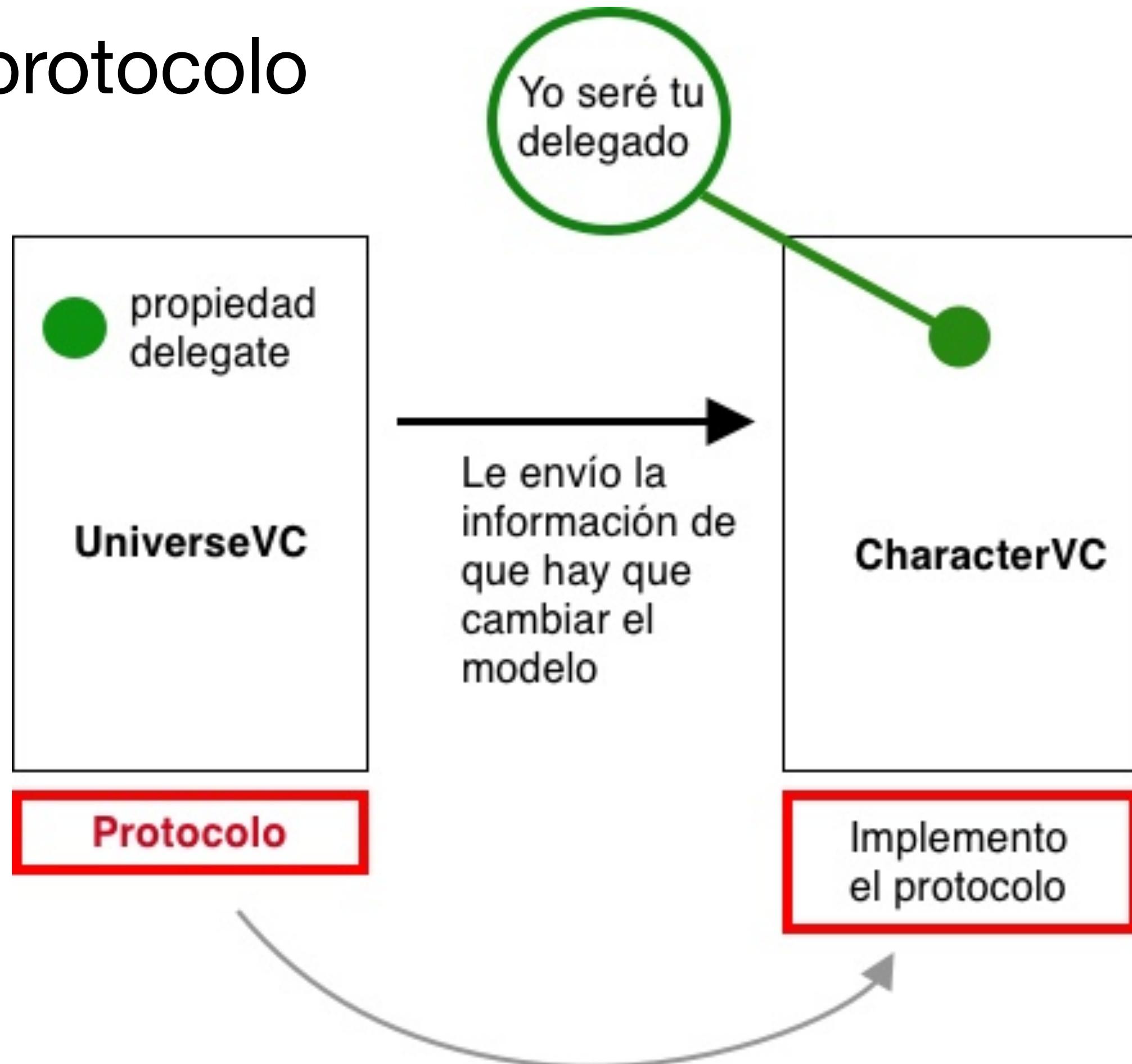


# Esquema de Delegados

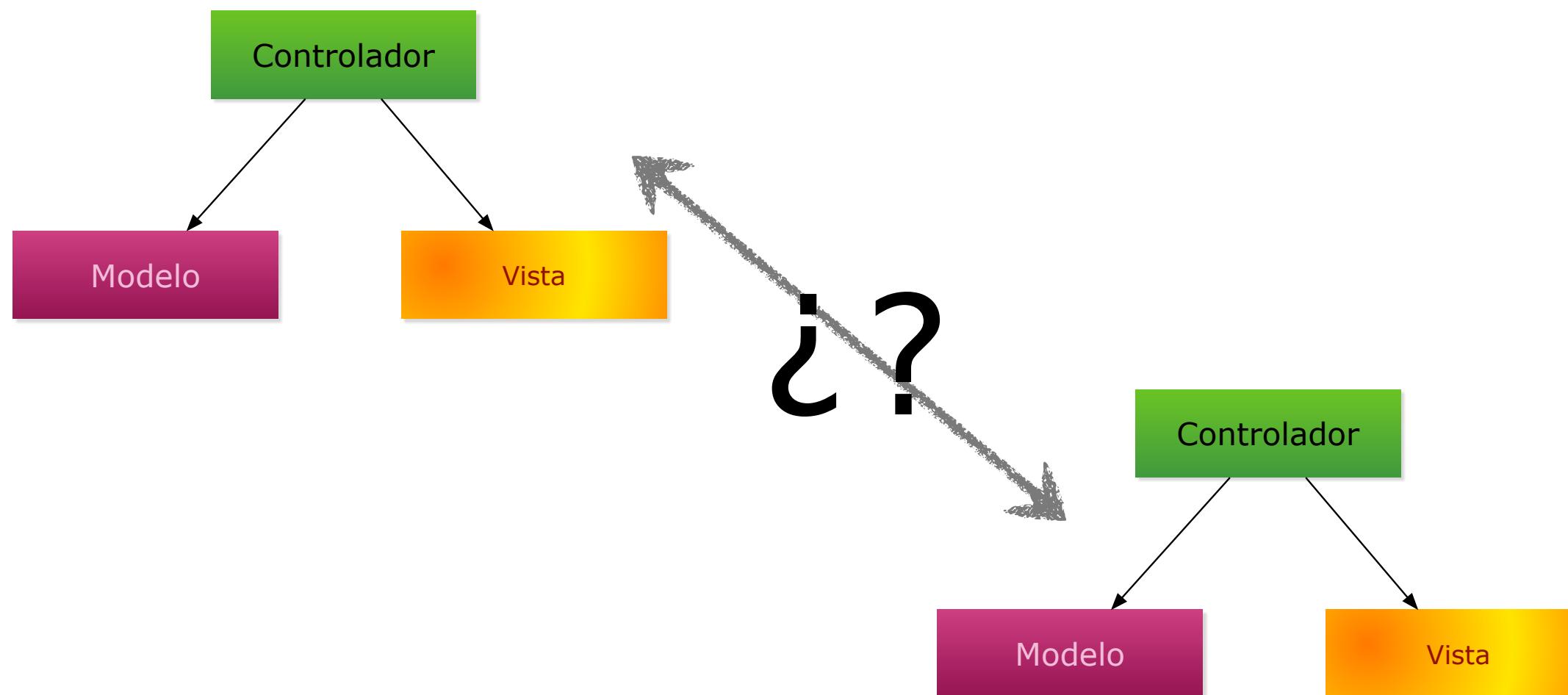


El de la derecha es  
siempre delegado del  
UISplitViewController

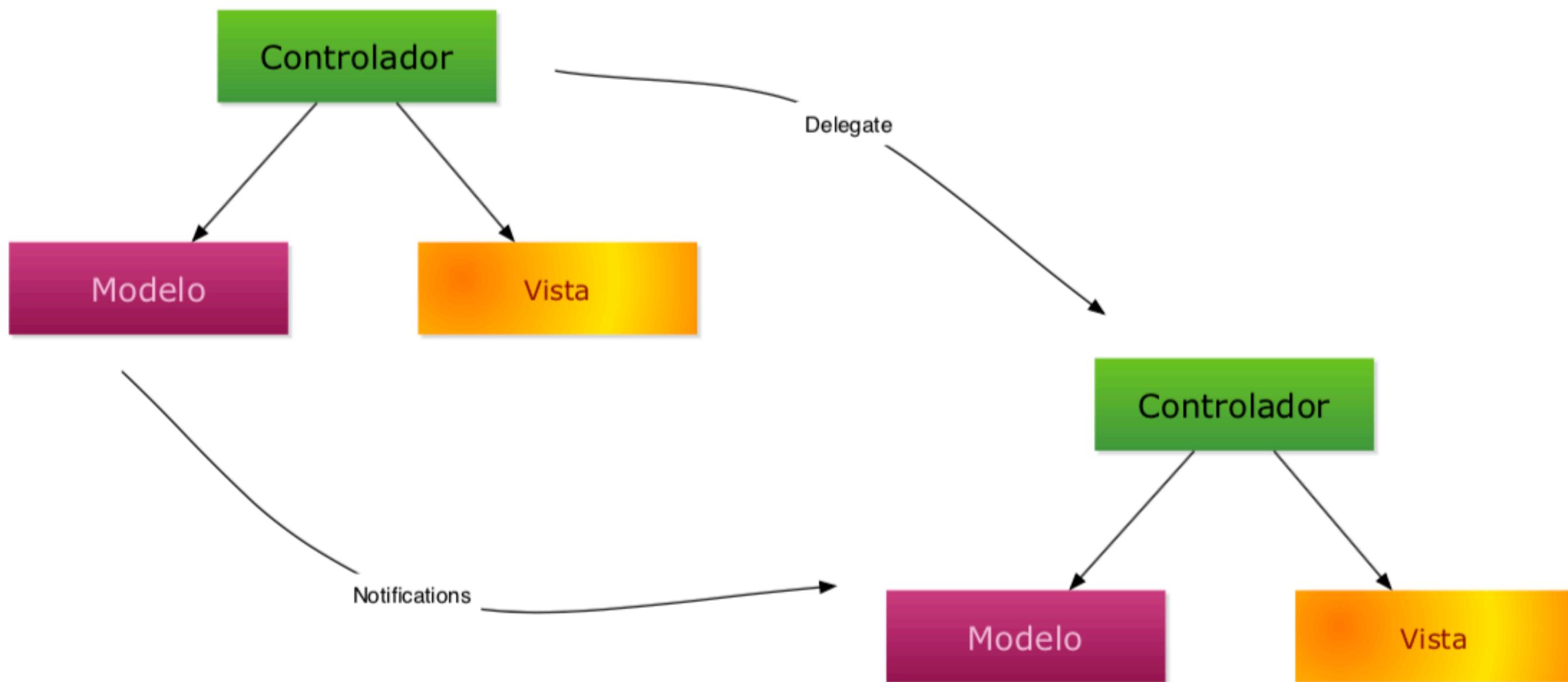
# Creamos un protocolo



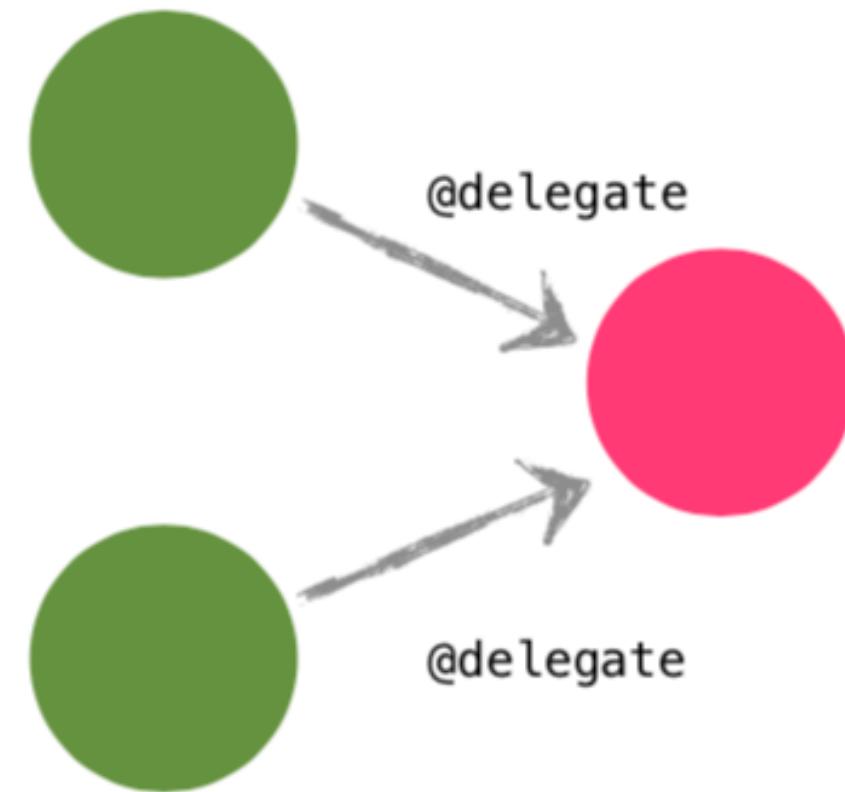
# Comunicación entre distintos MVCS



# Comunicación entre distintos MVCS

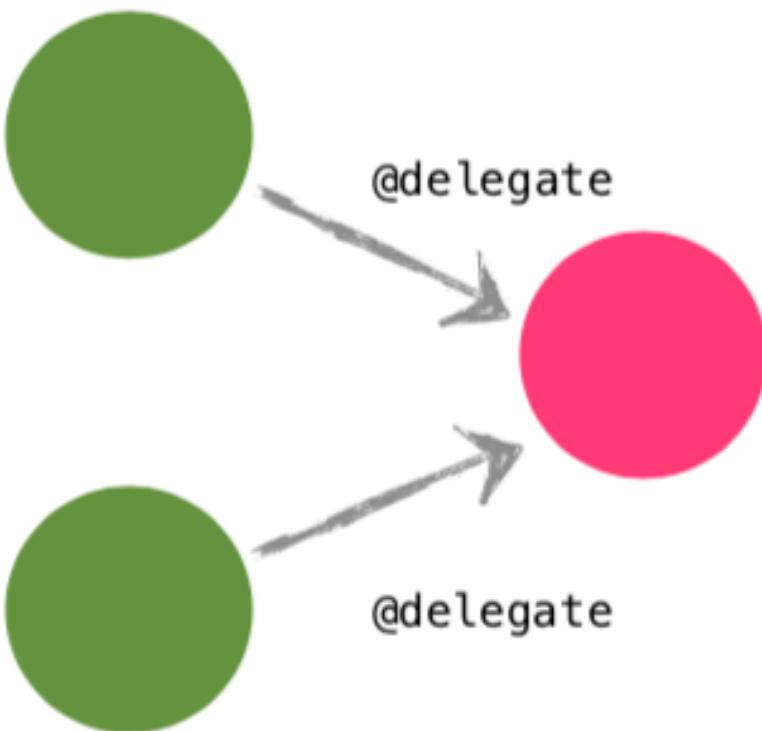


# Limitaciones del Delegate

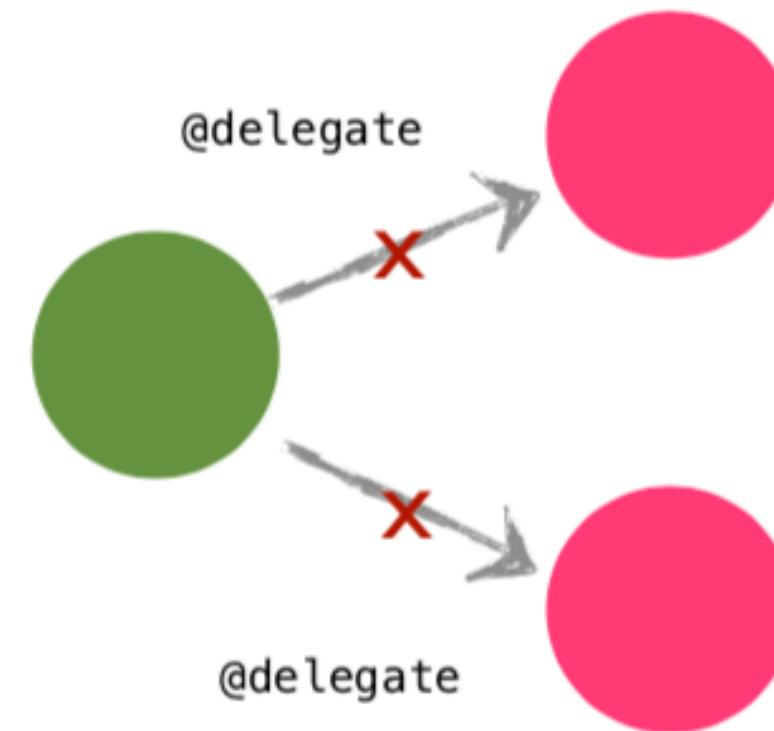


Un objeto puede ser delegado de más de un objeto

# Limitaciones del Delegate

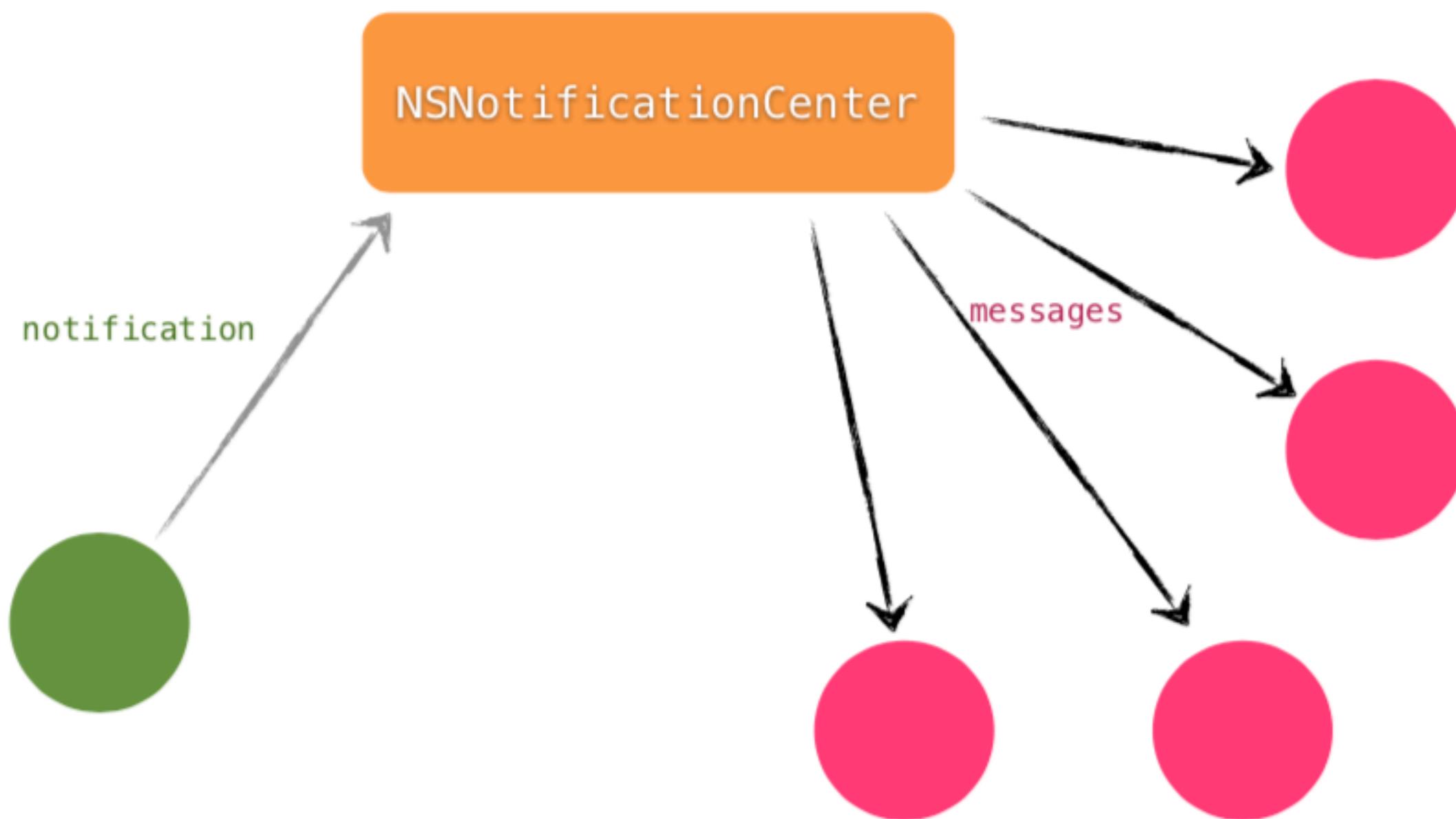


Un objeto puede ser delegado de más de un objeto



Un objeto NO puede tener más de un delegado

# Notificaciones



# Alta en NotificationCenter

- En qué notificación estamos interesados
- De quién
- Qué mensaje queremos que nos manden cuando se produzca la notificación

# 4 mandamientos de comunicación entre objetos

- 1º. Usa siempre el más sencillo
- 2º. Para eventos simples, usa Target / Action
- 3º. Para casos más complejos, usa el Delegate
- 4º. Si el delegate no te sirve, especialmente si necesitas comunicarte con más de un objeto, usa las Notifications.

# Apps Universales



- Una app que funciona tanto en iPhone como iPad.
- Vamos a adaptar nuestra app de iPad para el iPhone.

# Lo que cambia

- `rootViewController`
  - iPad: un `UISplitViewController`
  - iPhone: `UINavigationController`
- El comportamiento cuando el usuario toca sobre una celda de la tabla.

# Detectar el Dispositivo

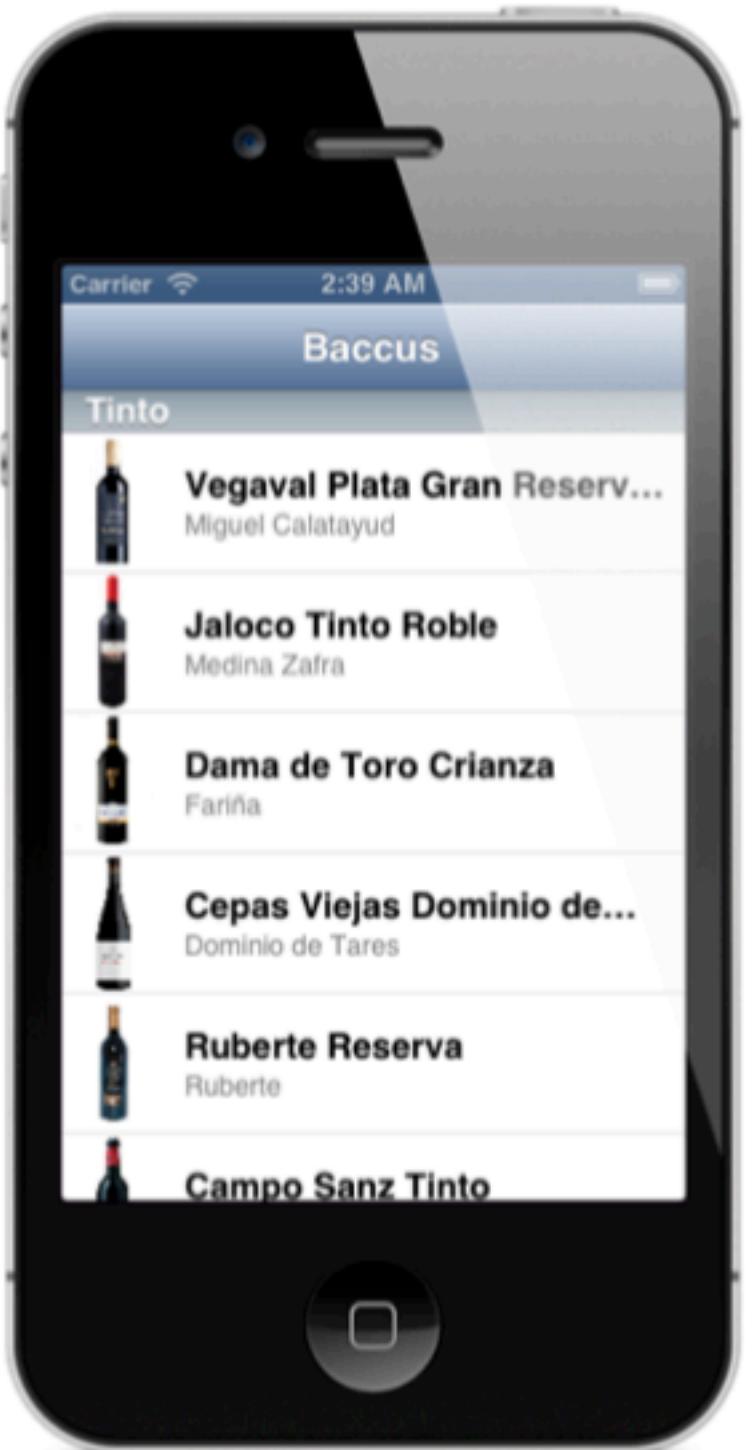
- No se detecta el dispositivo, sino el hardware que nos interesa.
- En nuestro caso, queremos saber si estamos en pantalla grande (tableta) o pequeña (iPhone o iPod)

```
switch UIDevice.current.userInterfaceIdiom {  
    case .phone:  
        print("It's an iPhone")  
        break  
    case .pad:  
        print("It's an iPad")  
        break  
    default:  
        print("Uh, oh! What could it be?")  
        break  
}
```

# Toque sobre una celda

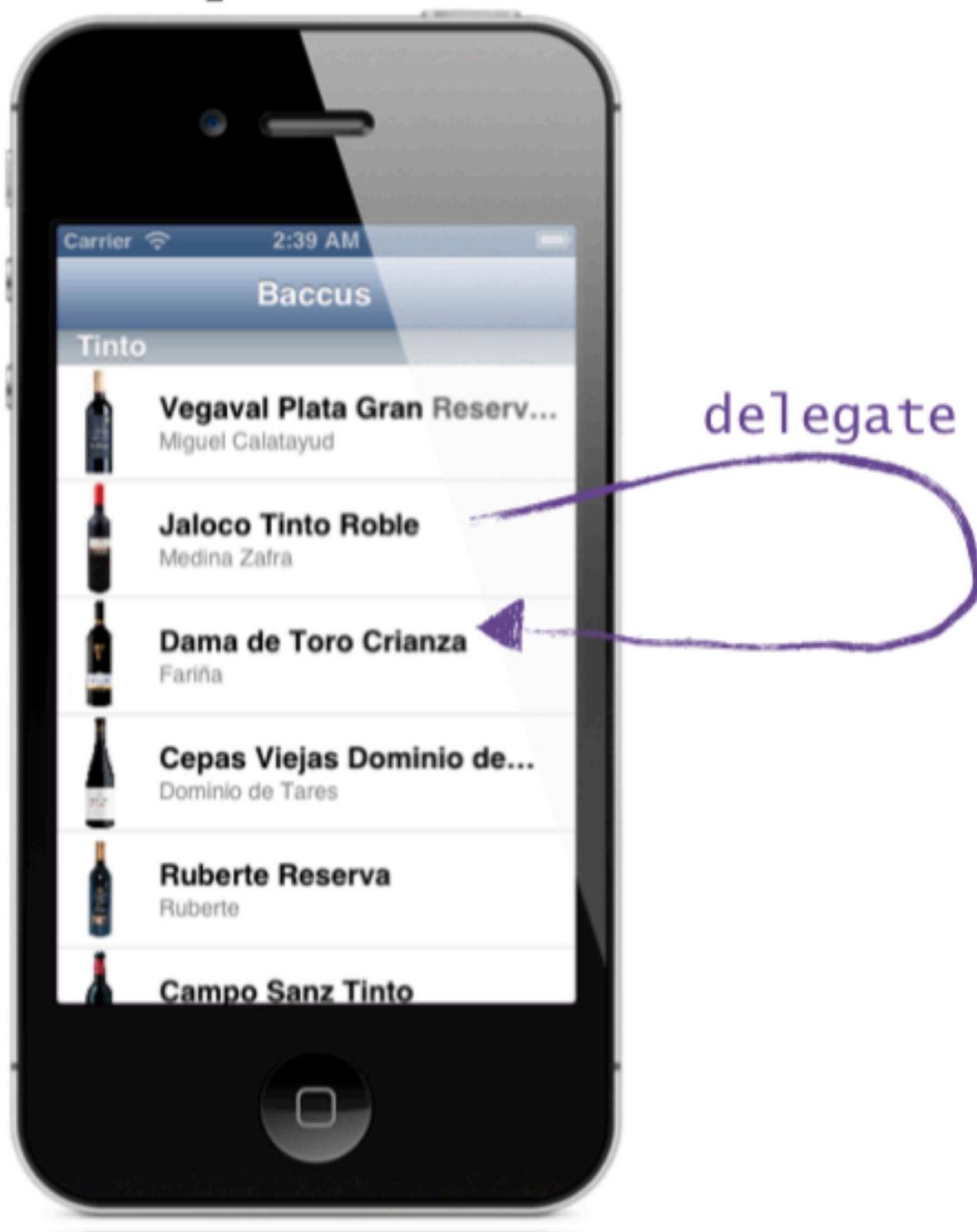


# Toque sobre una celda



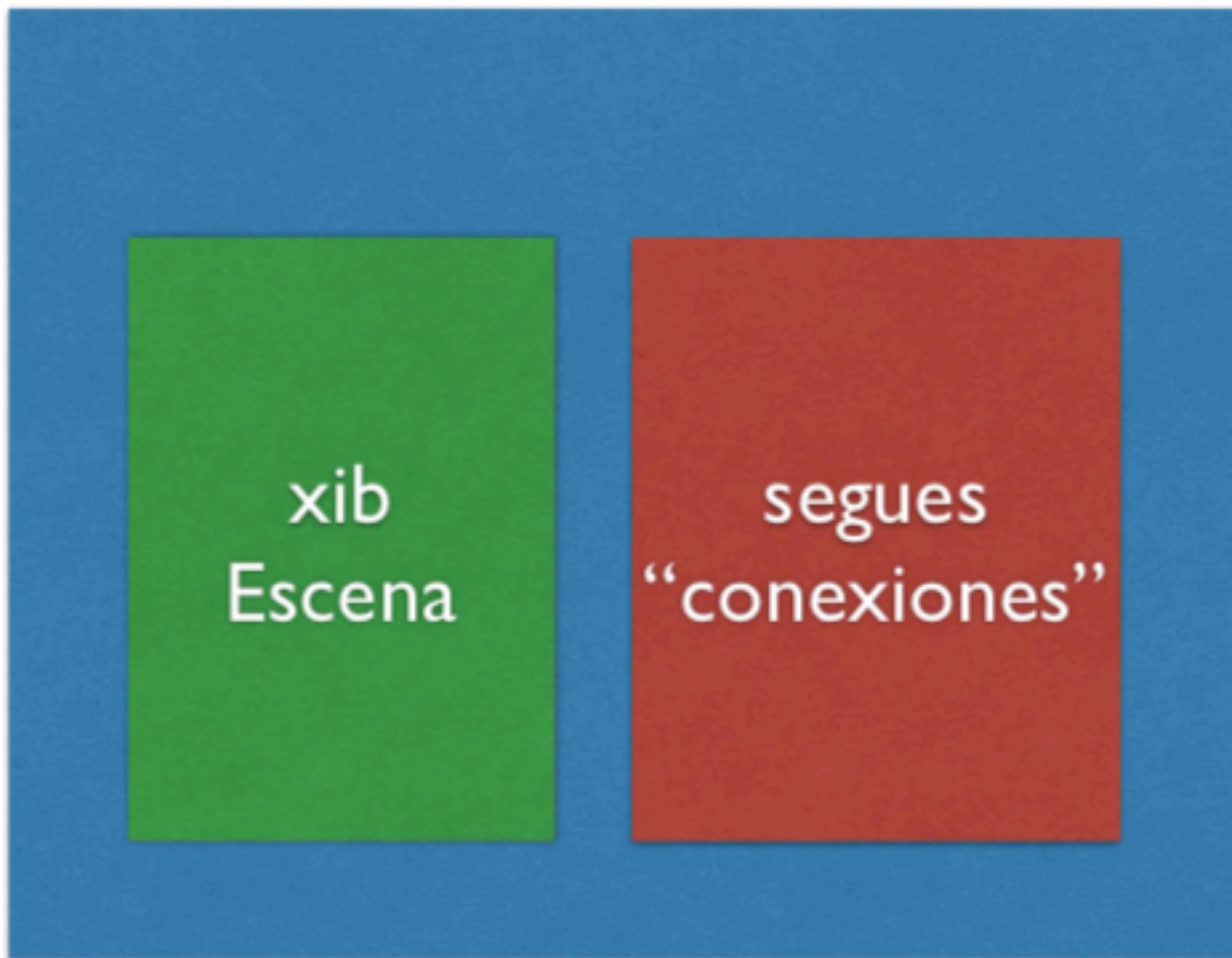
¿Quién puede ser el delegado  
de WineryViewController  
para encargarse de mostrar el  
vino seleccionado?

# Toque sobre una celda

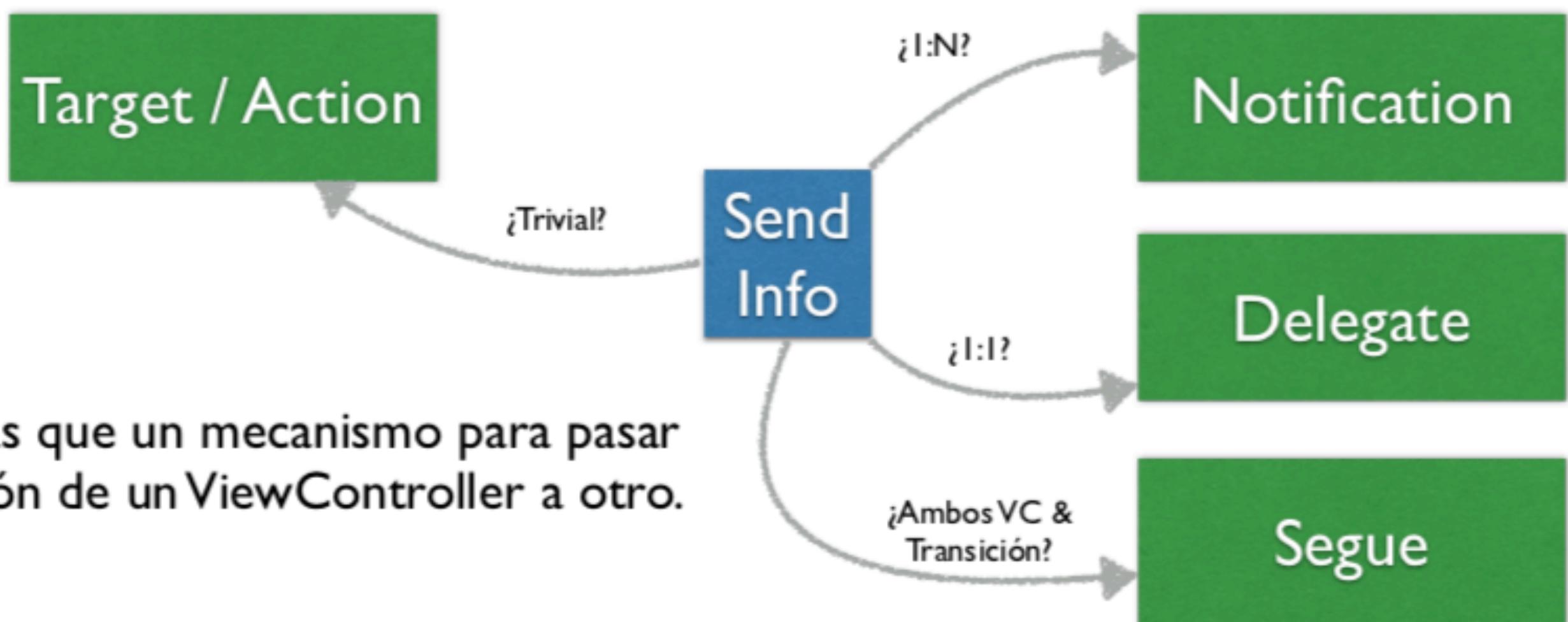


Auto delegación:  
Yo me lo guiso,  
yo me lo como.

# Storyboards



# Segue: ¿y eso qué es?



# Dos tipos de Segues

## Relación

Indican que un VC está contenido dentro de un combinador:

- Navigation → RootVC
- Split → Master / Detail
- Tab → Items

## Acción

Hay 5 tipos de segues de acción.

- Show (push)
- Show Detail (Replace)
- Present Modal
- PresentPopover
- Custom

# Xib - Storyboards

- Cuando se trabaja con Storyboards, al abrir la app, ésta hace lo siguiente:
  1. busca cuál es el Storyboard inicial (por defecto es main).
  2. Crea una instancia del Storyboard.
  3. Extrae su ViewController inicial.
  4. Se lo pasa a la Window.
  5. UITableViewDataSource (nos pide los datos).
- Cuando se trabaja con Xibs tenemos que lanzar la app desde el SceneDelegate con los siguientes pasos:
  1. Crear una window.
  2. Crear un ViewController.
  3. Asignar el ViewController como root.
  4. Hacer visible la window.

# Sistema de Archivos

- Es un sistema de ficheros Unix.
- La seguridad es muy estricta.
- Toda App está encerrada en una “**Sandbox**”
- La Sandbox es una jaula en la que está encerrada tu App.
- Por seguridad: nadie puede sobreescribir tus datos.
- Por privacidad: nadie puede leer tus datos.
- Por higiene: cuando tu App es eliminada, no deja rastros en el sistema.

# Principales Carpetas de la Sandbox

- **App bundle:** tiene tus binarios, imágenes, etc. Es de solo lectura.
- **Documentos:** para guardar datos permanentes creados por el usuario.
- **Caches:** Datos temporales (no se hace copia de seguridad con iTunes).
- **Otros:** ver NSSearchPathDirectory en la documentación.

# Carpetas del Sandbox

```
// Obtenemos una instancia de NSFileManager  
let fm = FileManager.default
```

El método urls obtiene el array de las carpetas de la sandBox: el directorio “tal” con dominio “cual”.

Busco la carpeta “Caches”, pero como carpetas “Caches” puede haber mas de una, en el 2º parámetro le indicamos la cache del usuario.

```
let urls = fm.urls(for: .cachesDirectory, in: .userDomainMask)
```

Si queremos acceder a un fichero, añadimos u nombre a la ruta

```
let url = urls.last?.appendingPathComponent("MyFile")
```

# Persistencia Avanzada

Introducción a Core Data



Managed Object Context

Managed  
Object

Managed  
Object



Persistent Store Coordinator

Persistent Store



SQLite

Main Queue Context

Private Queue Context



Persistent Store Coordinator

Persistent Store



SQLite

# Managed Object Context

- Un “area” donde podemos modificar objetos antes de guardarlos a disco. Se puede tener más de uno, en aplicaciones más complejas.
- Cada uno puede ejecutarse en su propia hebra
- Se encarga de hacer y deshacer los cambios a los objetos. También se encarga de la integridad de los datos.

# Managed Object

- Todos los objetos gestionados por Core Data o son instancias de NSManagedObject o descienden de él.
- Cada NSManagedObject representa una entidad.

# Managed Object Model

- Representa un esquema de datos: descripción detallada de los objetos, sus atributos y relaciones.
- Se suele crear de forma gráfica (como un xib), pero también se puede hacer por código.

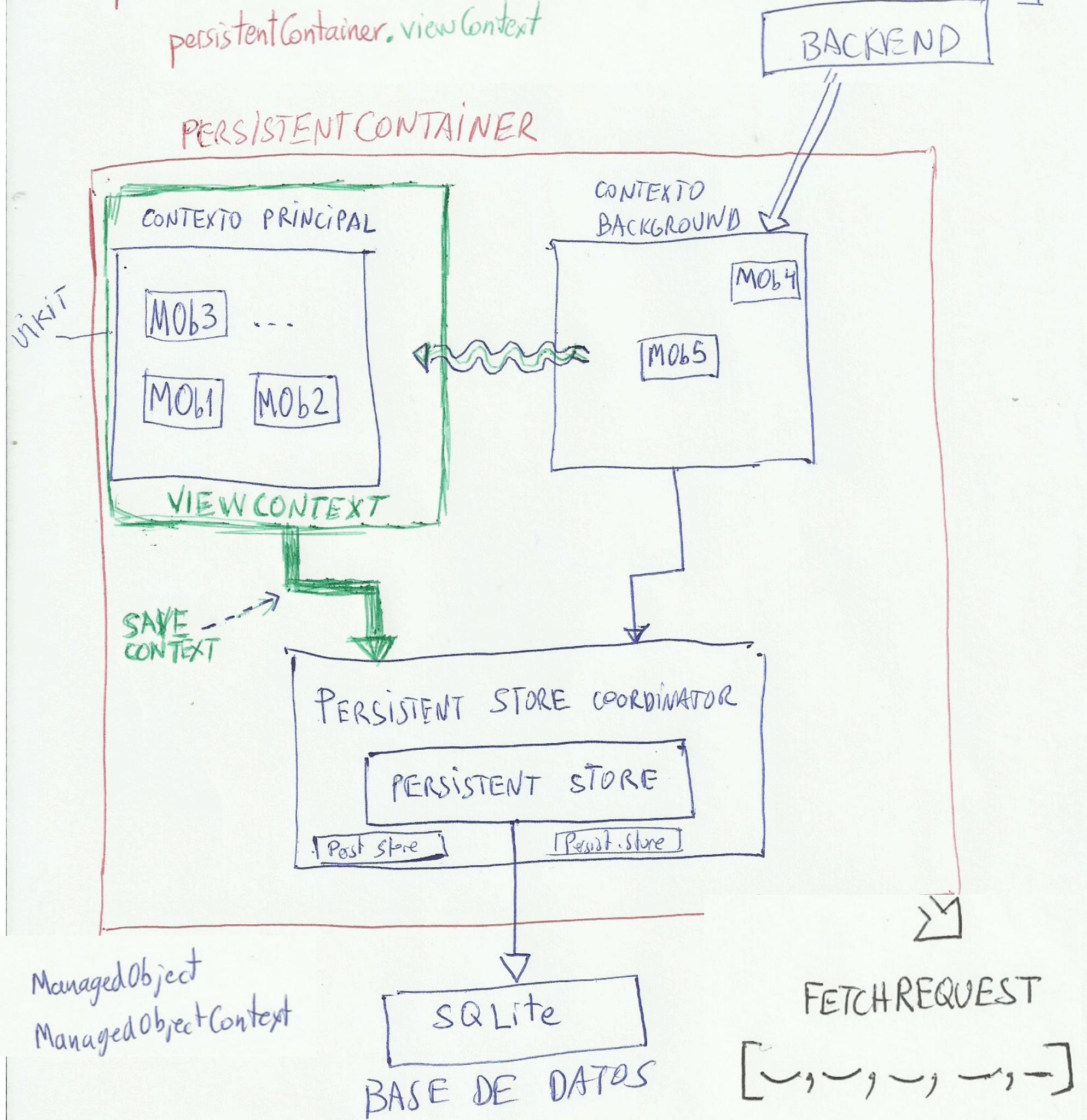
# Persistent Store Coordinator

- Permite manejar varios repositorios Un caso común es
  - Repositorio de datos en disco y
  - otro en memoria como cache de datos online

# Fetched Request

- Representa el resultado de una búsqueda de objetos.
- Devuelve un “array” de “objetos”.
- En realidad, son cascarones vacíos.
- Lleva la carga perezosa a extremos (faulting).

persistentContainer.viewContext



```
// MARK: - Core Data stack

// Se crea un objeto de la clase NSPersistentContainer (CONTENEDOR).
// var aaa: Tipo = { ..... }()
// lazy -> Variable que no se crea hasta que no se usa por 1a vez

lazy var persistentContainer: NSPersistentContainer = {

    // Le pasamos nuestro modelo con el nombre que le hemos puesto. El
    managedObjectModel lo busca en el bundle principal

    let container = NSPersistentContainer(name: "agendaConCoreData")

    // Con el método loadPersistentStores se hace la inicialización del stack. Se carga
    de todo lo que hay en CoreData. Además lo hace en background puesto que es asíncrona.
    Llama a un bloque que te devuelve un error si algo ha ido mal

    container.loadPersistentStores(completionHandler: { (storeDescription, error) in

        if let error = error as NSError? {

            fatalError("Unresolved error \(error), \(error.userInfo)")
        }
    })
    return container
}()
```

```
// MARK: - Core Data Saving support
// Este método es del AppDelegate. Se llama desde la instancia que tengamos en la app
// del AppDelegate

func saveContext () {

    let context = persistentContainer.viewContext

    // En el caso de que hayan habido cambios en los objetos del contexto, o se hayan
    // añadido o borrado

    if context.hasChanges {

        do {
            // Se salva el contexto
            try context.save()

        } catch {

            // Nos envía un mensaje de error
            let nserror = error as NSError {
                fatalError("Unresolved error \(nserror), \(nserror.userInfo)")
            }
        }
    }
}
```

Existen recursos a los que nuestra App no tiene acceso si el usuario no se lo permite:

[Contactos](#)

[Cámara](#)

[Librería de fotos](#)

[Localización](#)

[Eventos](#)

[Calendarios](#)

[Twitter](#)

[Etc...](#)

Para poder acceder a ellos la App le pregunta al usuario si le da permiso para ello.

[Esto se hace en el archivo .plist \(xml\)](#)

Hay un tipo especial de **controladores modales** del sistema que nos permiten acceder a esos recursos sensibles y vienen ya predefinidos con su modelo y su vista.

Ejemplos:

**UIImagePickerController**: obtener imágenes.

**TWTweetComposeViewController**: compartir por Twitter.

**MFMessageComposeViewController**: compartir por email.

etc...

## ¿Cómo se usan?

- 1º. Lo creamos
- 2º. El viewController en el que estamos se hace delegado de ese nuevo controlador
- 3º. Lo mostramos
- 4º. El usuario elige el recurso que quiere mostrar o le da a cancelar
- 5º. Recibes la información correspondiente (el recurso u opción cancelado)

## Usaremos el **UIImagePickerController**:

Trae imágenes de varias fuentes: Cámara, carrete, álbumes, librería de fotos.

Tiene métodos para averiguar si el dispositivo tiene cámara, qué tipo de cámara, si puede grabar video, etc...

