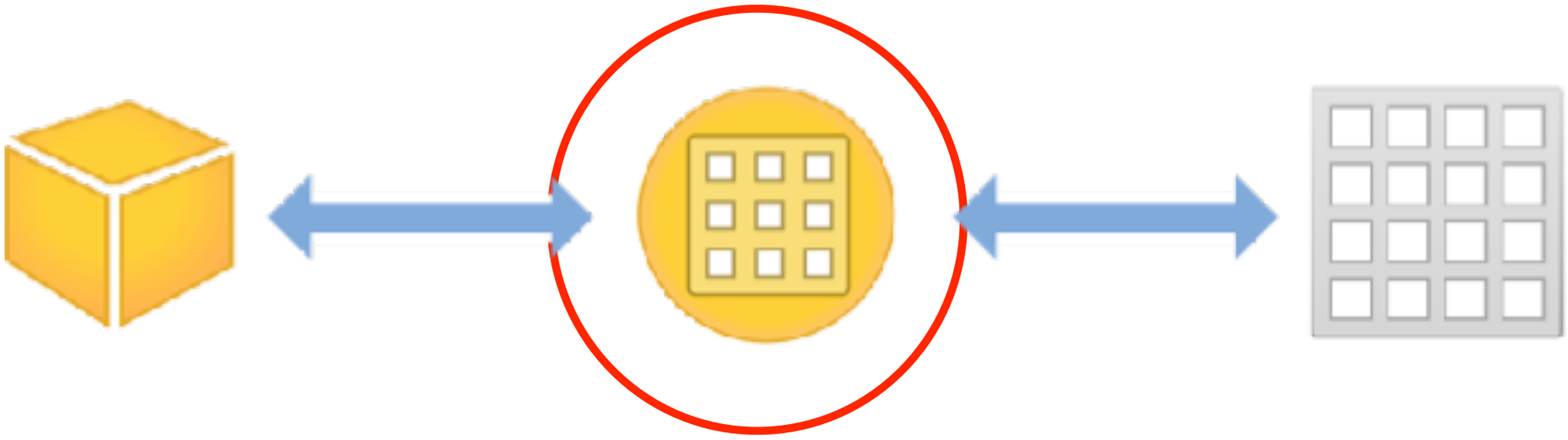
View Controller

谷方明
fmgu2002@sina.com

MVC



Model **Controller** **View**

Reference

- View Controller Programming Guide for iOS
  - https://developer.apple.com/library/archive/featuredarticles/ViewControllerPGforiPhoneOS/index.html#//apple_ref/doc/uid/TP40007457

- UIViewController Class Reference.
  - https://developer.apple.com/documentation/uikit/uiviewcontroller

- Customizing the Behavior of Segue-Based Presentations
  - https://developer.apple.com/documentation/uikit/resource_management/customizing_the_behavior_of_segue-based_presentations

View Controller

- View Controller：
  - View controllers are the foundation of your app’s internal structure.
  - Every app has at least one view controller, and most apps have several.
  - Each view controller manages a portion of your app’s user interface as well as the interactions between that interface and the underlying data. View controllers also facilitate transitions between different parts of your user interface.
- UIViewController Class： defines the shared behavior that is common to all view controllers.

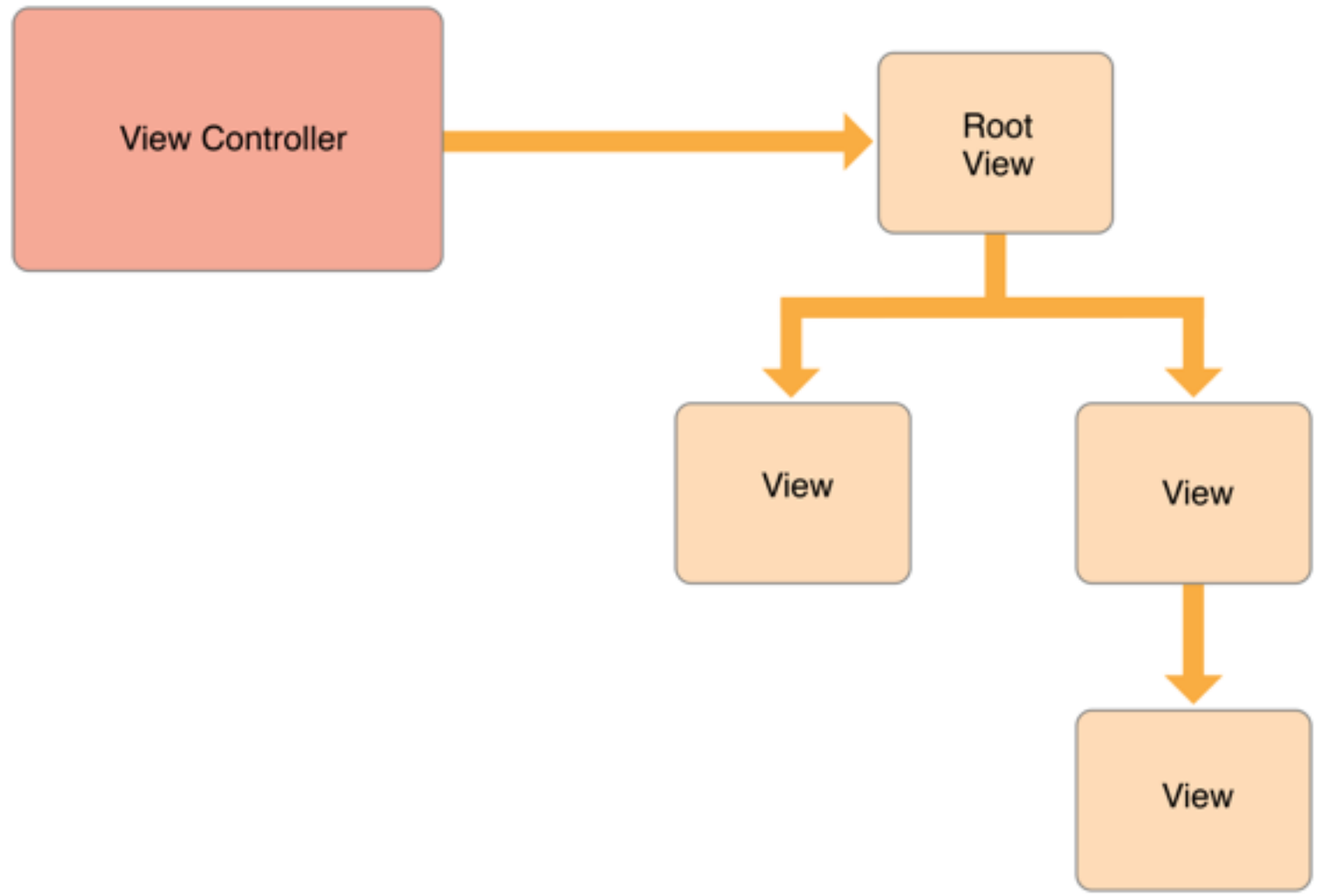
View Controller's main Responsibilities

- Updating the contents of the views, usually in response to changes to the underlying data. (**data marshaling**)
- Responding to user interactions with views.(**events handling**)
- Resizing views and managing the layout of the overall interface.(**views management**)
- Coordinating with other objects—including other view controllers—in your app.(**transition/segue**)

Views Management

- The most important role of a view controller is to manage a hierarchy of views.

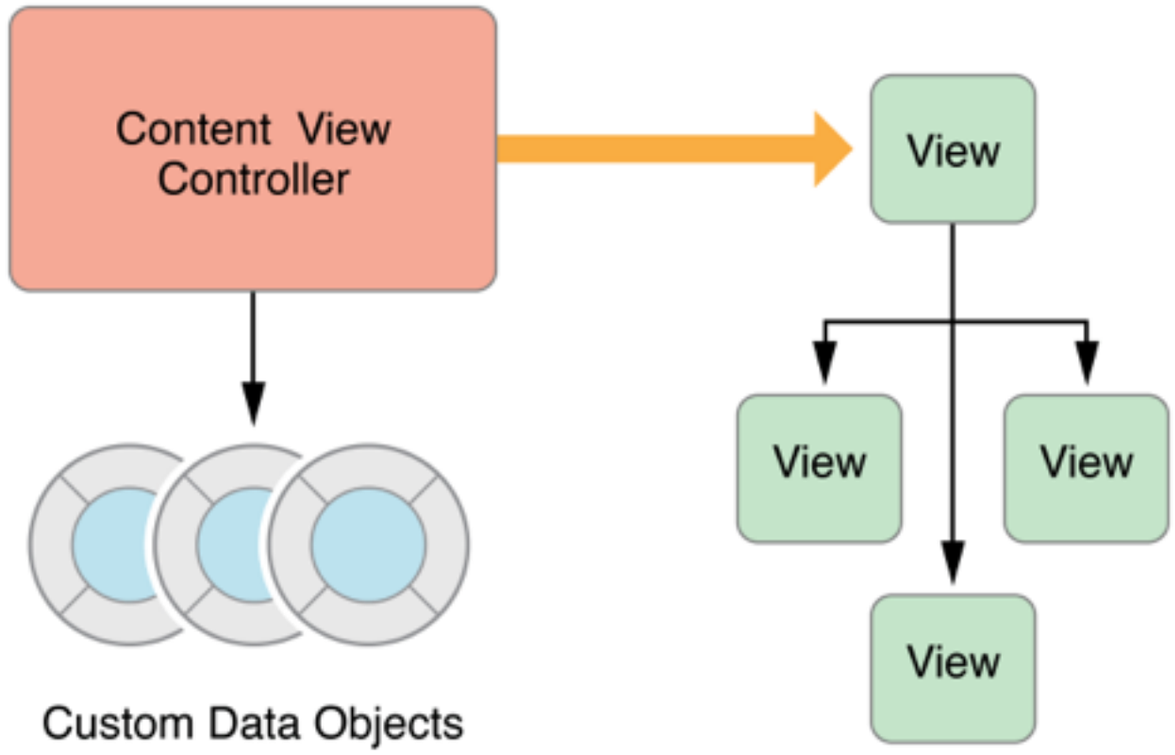
- Every view controller has a single root view that encloses all of the view controller's content. To that root view, you add the views you need to display your content.



- var view: UIView!
- The view that the controller manages.

Data Marshaling

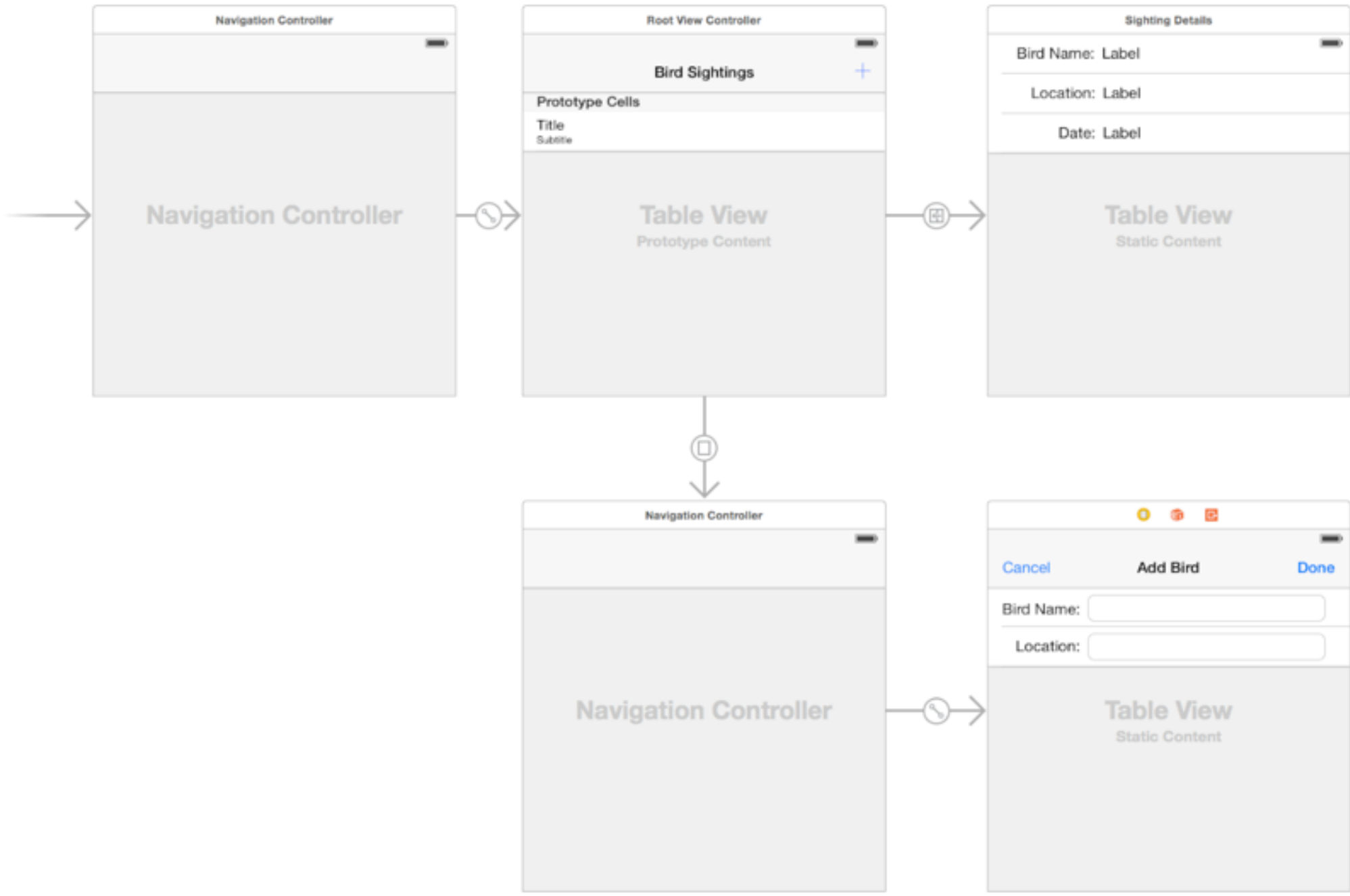
- A view controller acts as an **intermediary** between the views it manages and the data of your app.
- The methods and properties of the UIViewController class let you manage the visual presentation of your app. When you **subclass** UIViewController, you add any variables you need to manage your data in your subclass.
- You should always maintain a clean separation of responsibilities within your view controllers and data objects. Most of the logic for ensuring the integrity of your data structures belongs in the data objects themselves.



Events Handling

- View controllers define action methods for handling higher-level events. Action methods respond to:
  - Specific actions. Controls and some views call an action method to report specific interactions.
  - Gesture recognizers. Gesture recognizers call an action method to report the current status of a gesture. Use your view controller to process status changes or respond to the completed gesture.
- **View controllers observe notifications sent by the system or other objects.**
  - Notifications report changes and are a way for the view controller to update its state.
- V**iew controllers act as a data source or delegate for another object. .**
  - View controllers are often used to manage the data for tables, and collection views.

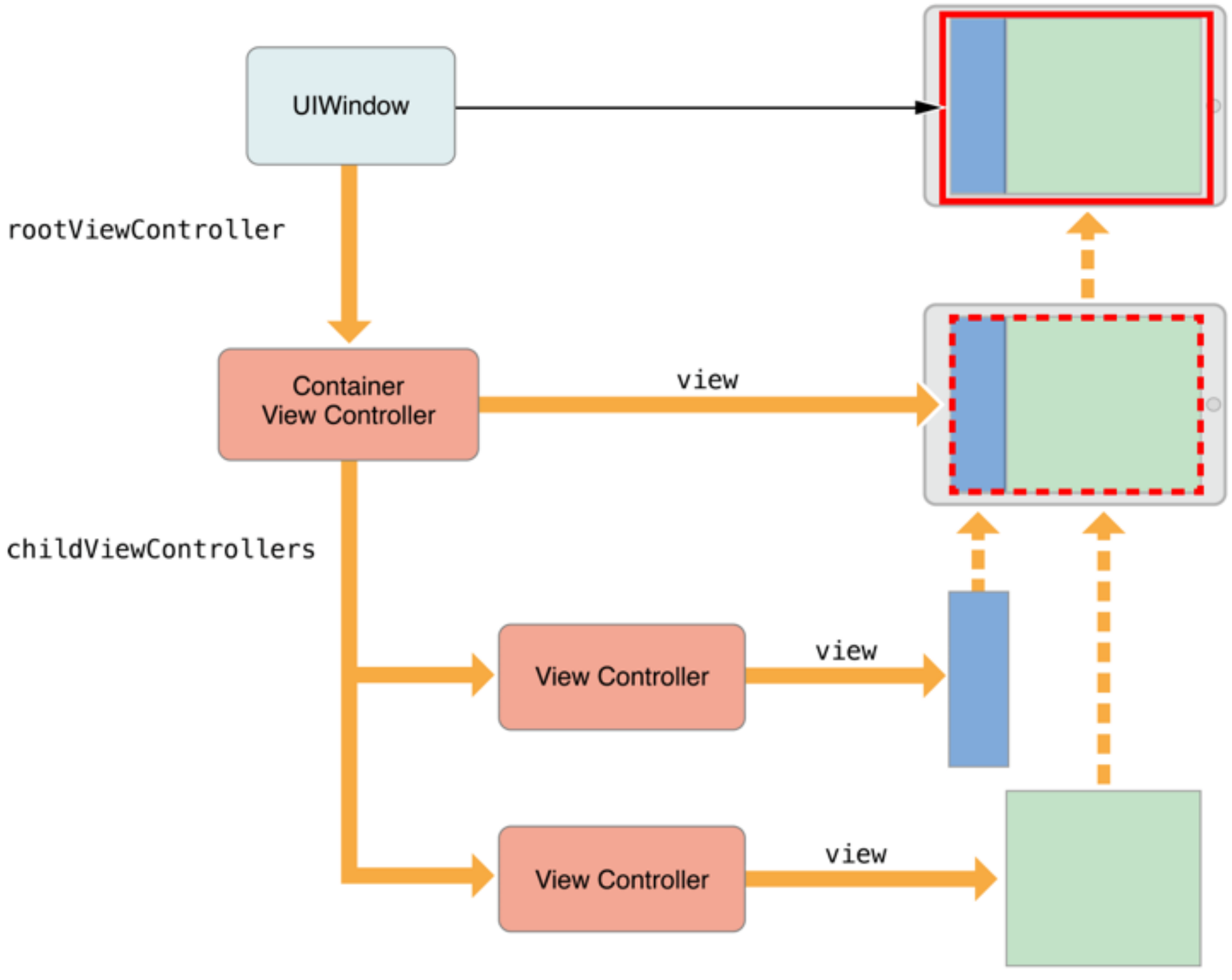
transition/segue



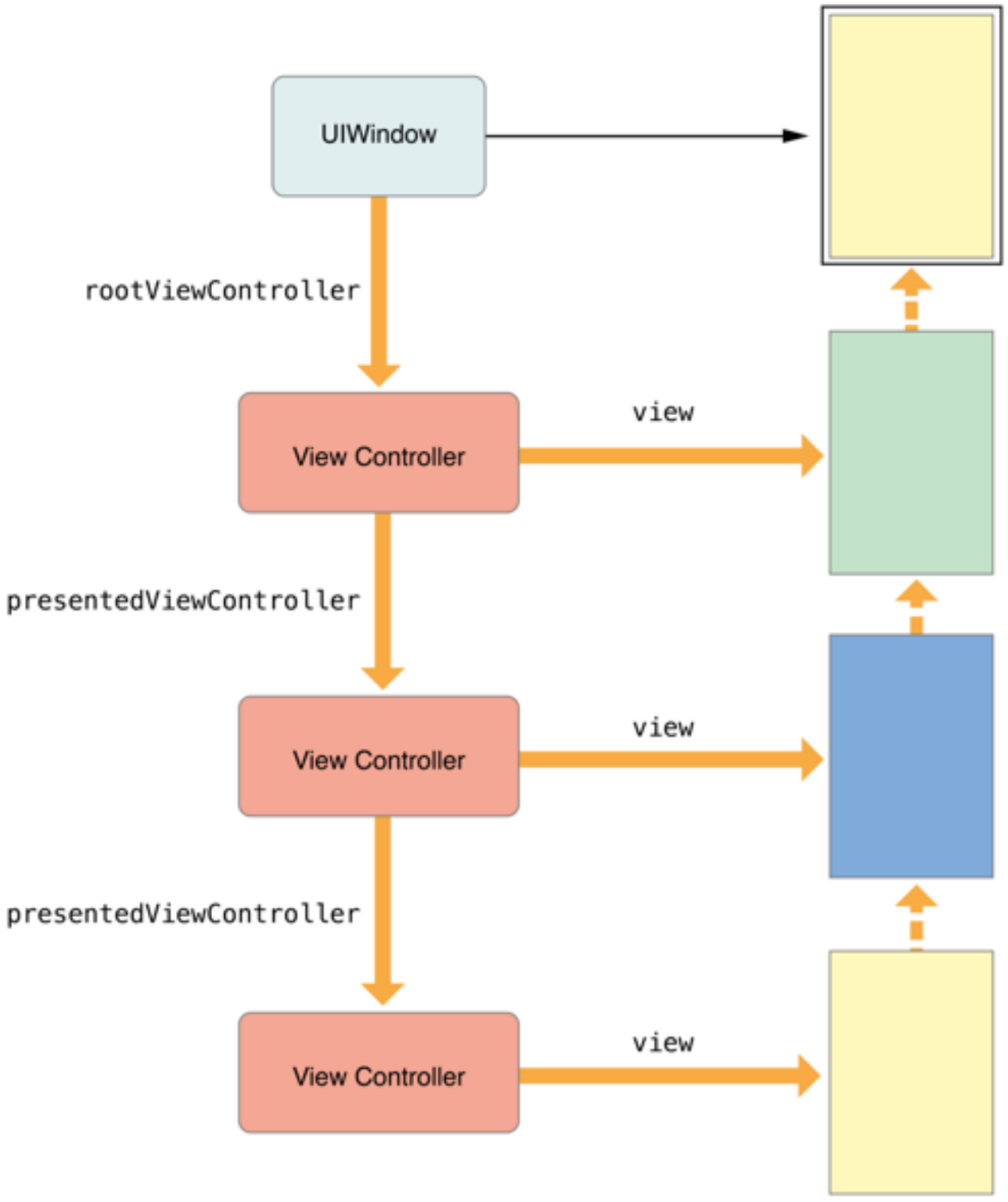
two types of view controllers

- *Content view controllers* manage a discrete piece of your app's content and are the main type of view controller that you create.
- *Container view controllers* collect information from other view controllers (known as child view controllers) and present it in a way that facilitates navigation or presents the content of those view controllers differently.
- Most apps are a mixture of both types of view controllers.

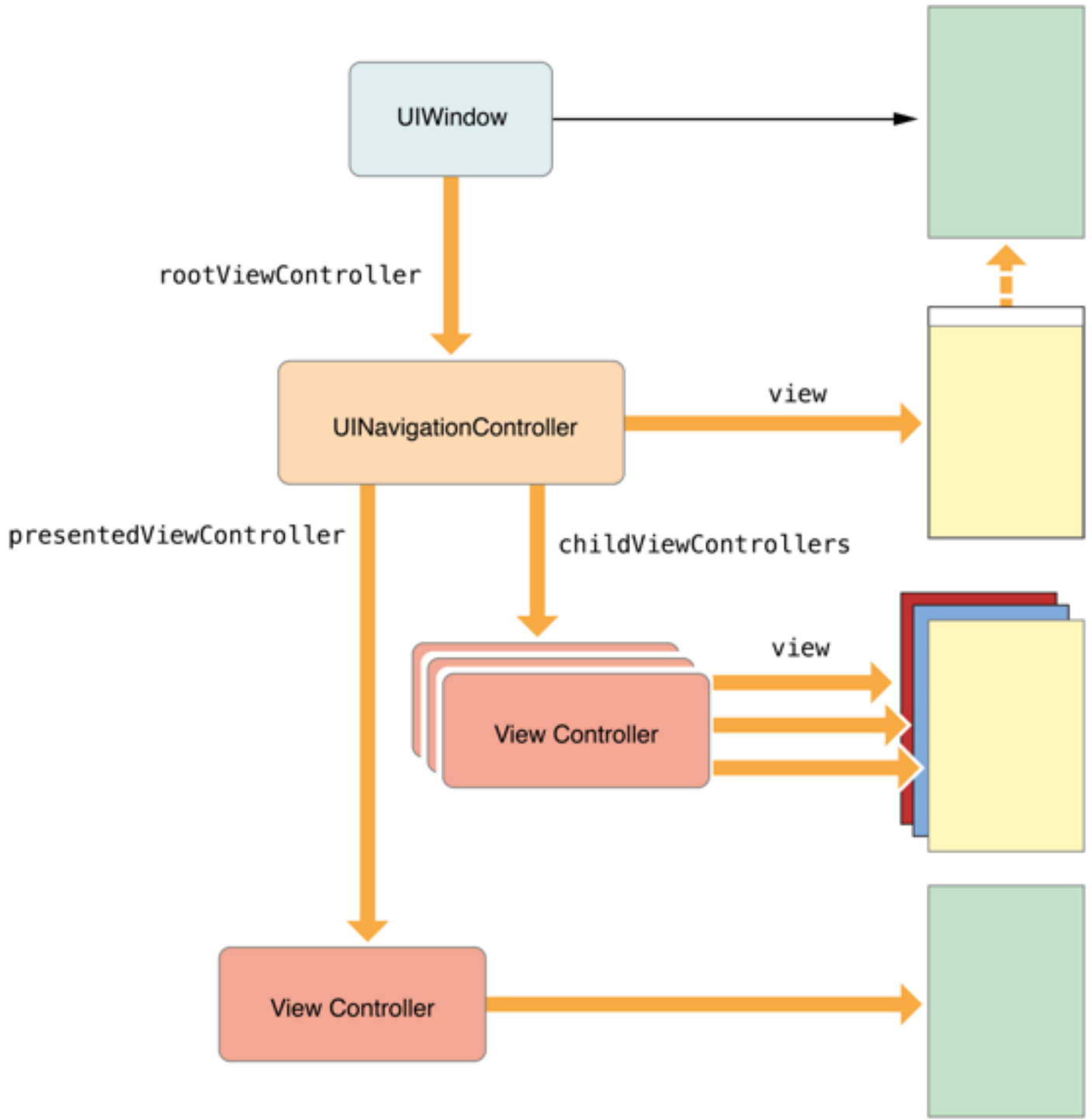
A container acting as the root view controller



Presented view controllers

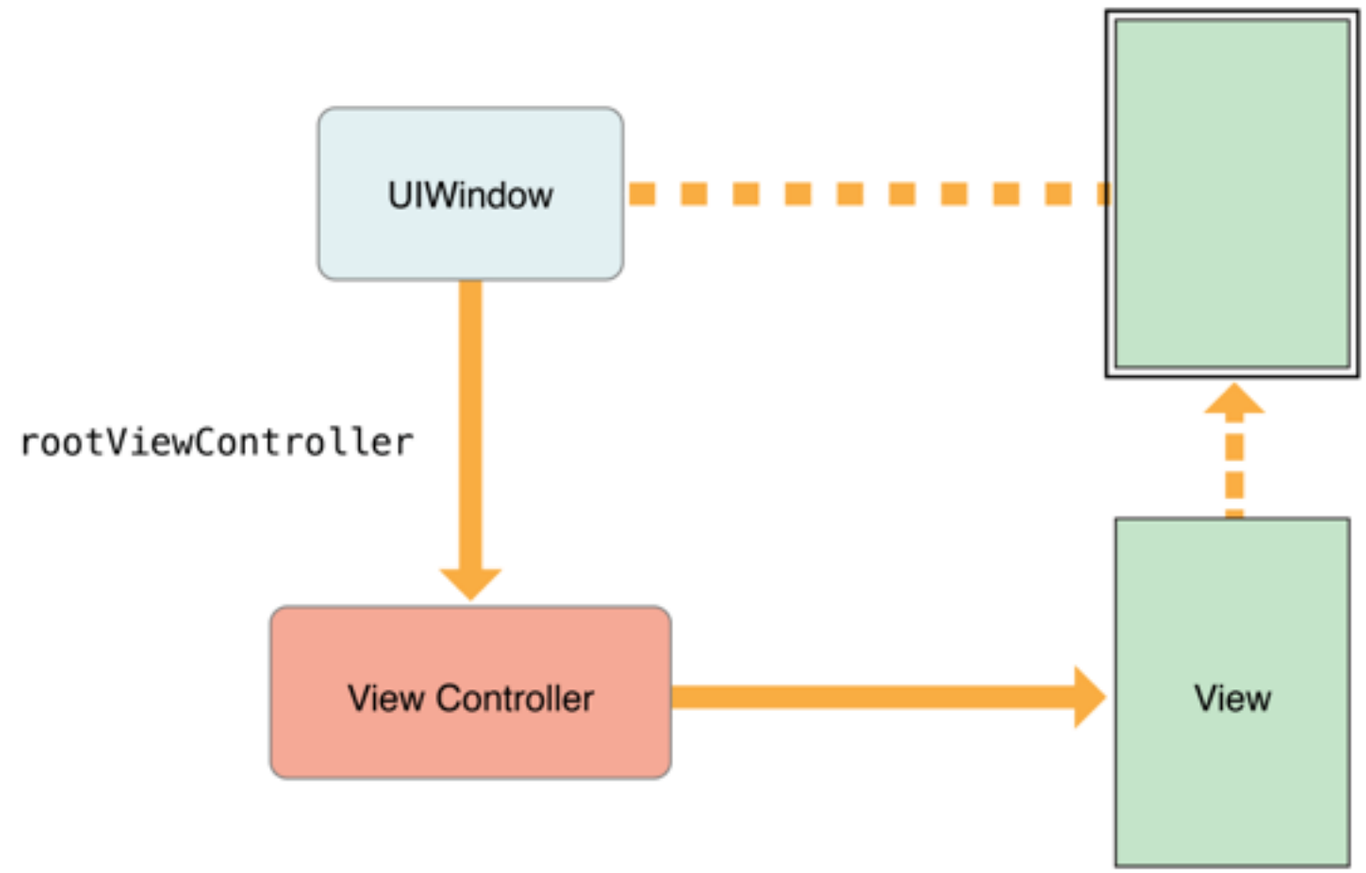


A container and a presented view controller



Root View Controller

- The root view controller is the anchor of the view controller hierarchy.
- Every window has exactly one root view controller whose content fills that window.
- The root view controller is accessible from the `rootViewController` property of the `UIWindow` object. When you use storyboards to configure your view controllers, UIKit sets the value of that property automatically at launch time.



- For windows you create programmatically, you must set the root view controller yourself.

```
    func application(_ application: UIApplication,
didFinishLaunchingWithOptions launchOptions:
[UIApplicationLaunchOptionsKey: Any]?) -> Bool {
        window = UIWindow(frame: UIScreen.main.bounds)
        window?.rootViewController = RootViewController()
        window?.makeKeyAndVisible()
        return true
    }
```

Views’ Load Process

- UIKit automatically loads views from your storyboard file when they are needed. As part of the loading process, UIKit performs the following sequence of tasks:

1 Instantiates views using the information in your storyboard file.
2 Connects all outlets and actions.
3 Assigns the root view to the view controller’s `view` property.
4 Calls the view controller’s `awakeFromNib` method.
When this method is called, the view controller’s trait collection is empty and views may not be in their final positions.
5 Calls the view controller’s `viewDidLoad` method.
Use this method to add or remove views, modify layout constraints, and load data for your views.

viewDidLoad

- After instantiation and outlet-setting, viewDidLoad is called
- This is an exceptionally good place to put a lot of setup code.It's better than an init because your outlets are all set up by the time this is called.

```
override func viewDidLoad() {
    super.viewDidLoad() // always let super have a chance in all lifecycle methods

    // do some setup of my MVC

}
```

- One thing you may well want to do here is update your UI from your Model. Because now you know all of your outlets are set.
- But be careful because the geometry of your view (its bounds) is not set yet!

awakeFromNib()

- This method is sent to all objects that come out of a storyboard (including your Controller).
  - NSObjcet Class
- Put code somewhere else if at all possible (e.g. viewDidLoad).

After Loading Process

- These pairs will be called each time your Controller's view goes on/off screen
  - viewWillAppear
  - viewDidAppear
  - viewWillDisappear
  - viewDidDisappear
- These "geometry changed" methods might be called at any time after viewDidLoad …
  - viewWillLayoutSubviews
  - viewDidLayoutSubviews
- If memory gets low, you might get …
  - didReceiveMemoryWarning

viewWillAppear

- Just before your **view** appears on screen, you get notified

  func viewWillAppear(_ animated: Bool)

  // animated is whether you are appearing over time

- Your view will only get “loaded” once, but it might appear and disappear a lot.
- Do something here if things your display is changing while your MVC is off-screen.You could use this to optimize performance by waiting until this method is called(as opposed to viewDidLoad) to kick off an expensive operation (probably in another thread)
- Your view’s geometry is set here, also there are other places to react to geometry.
- There is a “did” version of this as well

  func viewDidAppear(_ animated: Bool)

viewWillDisappear

- you get notified when you will disappear off screen

```
func viewWillDisappear(_ animated: Bool) {
    super.viewWillDisappear(animated)
// do some clean up now that removed from the screen
// not do anything time-consuming here, or app will be sluggish
// maybe even kick off a thread to do stuff here
    }
```

- There is a “did” version of this as well

```
func viewDidDisappear(_ animated: Bool)
```

Geometry changed

- Most of the time this will be automatically handled with Autolayout. But you can get involved in geometry changes directly with these methods

  func viewWillLayoutSubviews()

  func viewDidLayoutSubviews()

- They are called any time a view's frame changed and its subviews were thus re-layed out.

- You can reset the frames of your subviews here or set other geometry-related properties.

  Between "will" and "did", autolayout will happen.

  These methods might be called more often than you'd imagine (e.g. for pre- and post- animation arrangement, etc.).So don't do anything in here that can't properly be done repeatedly.

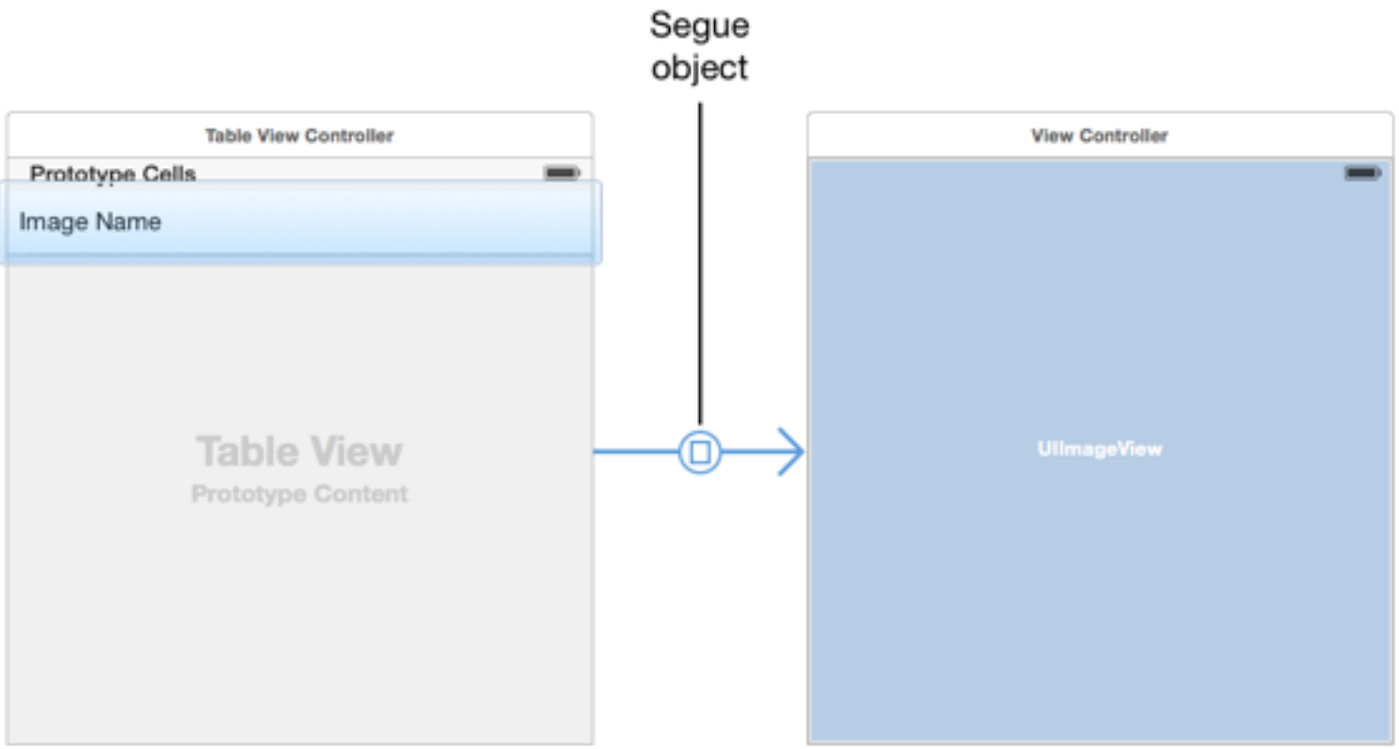
didReceiveMemoryWarning

- In low-memory situations, didReceiveMemoryWarning gets called
- This rarely happens, but well-designed code with big-ticket memory uses might anticipate it. Examples: images and sounds.
- Anything “big” that is not currently in use and can be recreated relatively easily should probably be released (by setting any pointers to it to nil)

Segue

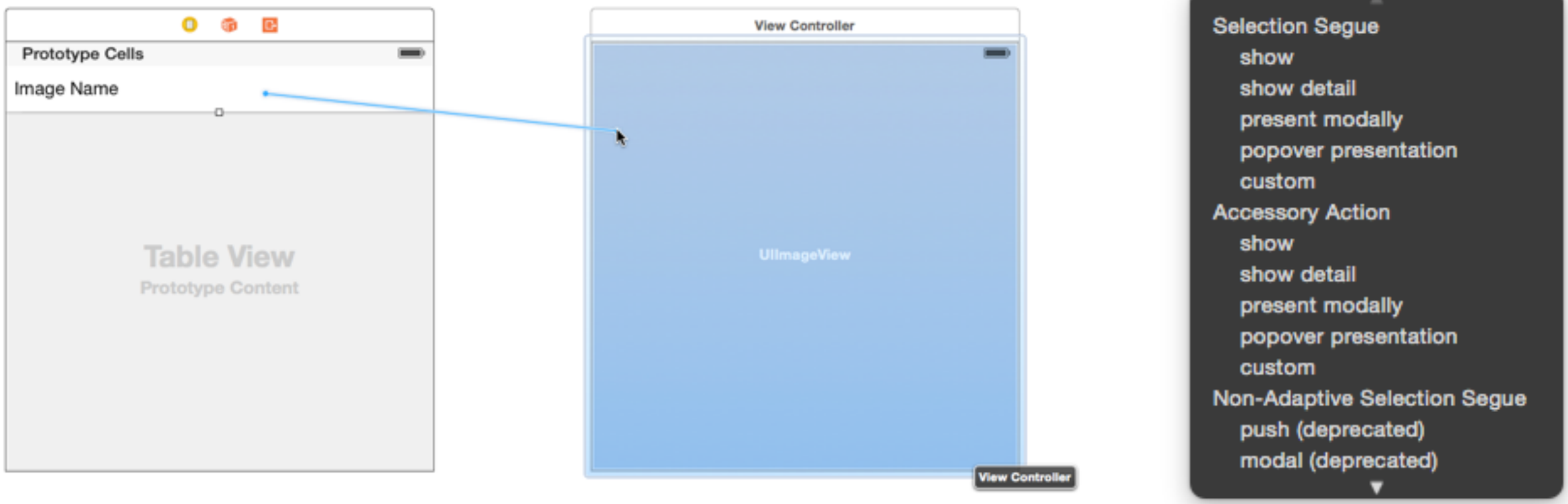
- A *segue* defines a transition between two view controllers

  The starting point of a segue is the button, table row, or gesture recognizer that initiates the segue. The end point of a segue is the view controller you want to display.



Create a Segue

- Control-click an appropriate element in the first view controller and drag to the target view controller
- Select type of segue

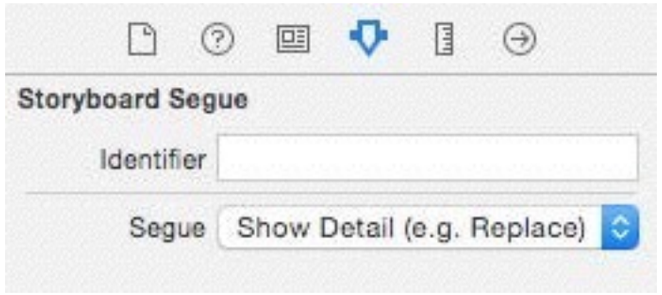


segue type

Segue type	Behavior
Show (Push)	For most view controllers, this segue presents the new content modally over the source view controller. eg. a navigation controller pushes the new view controller onto its navigation stack.
Show Detail (Replace)	relevant only for view controllers embedded inside a `UISplitViewController` object. With this segue, a split view controller replaces its second child view controller (the detail controller) with the new content.
Present Modally	displays the view controller modally using the specified presentation and transition styles.
Present as Popover	In a horizontally regular environment, the view controller appears in a popover. In a horizontally compact environment, the view controller is displayed

segue Identifier

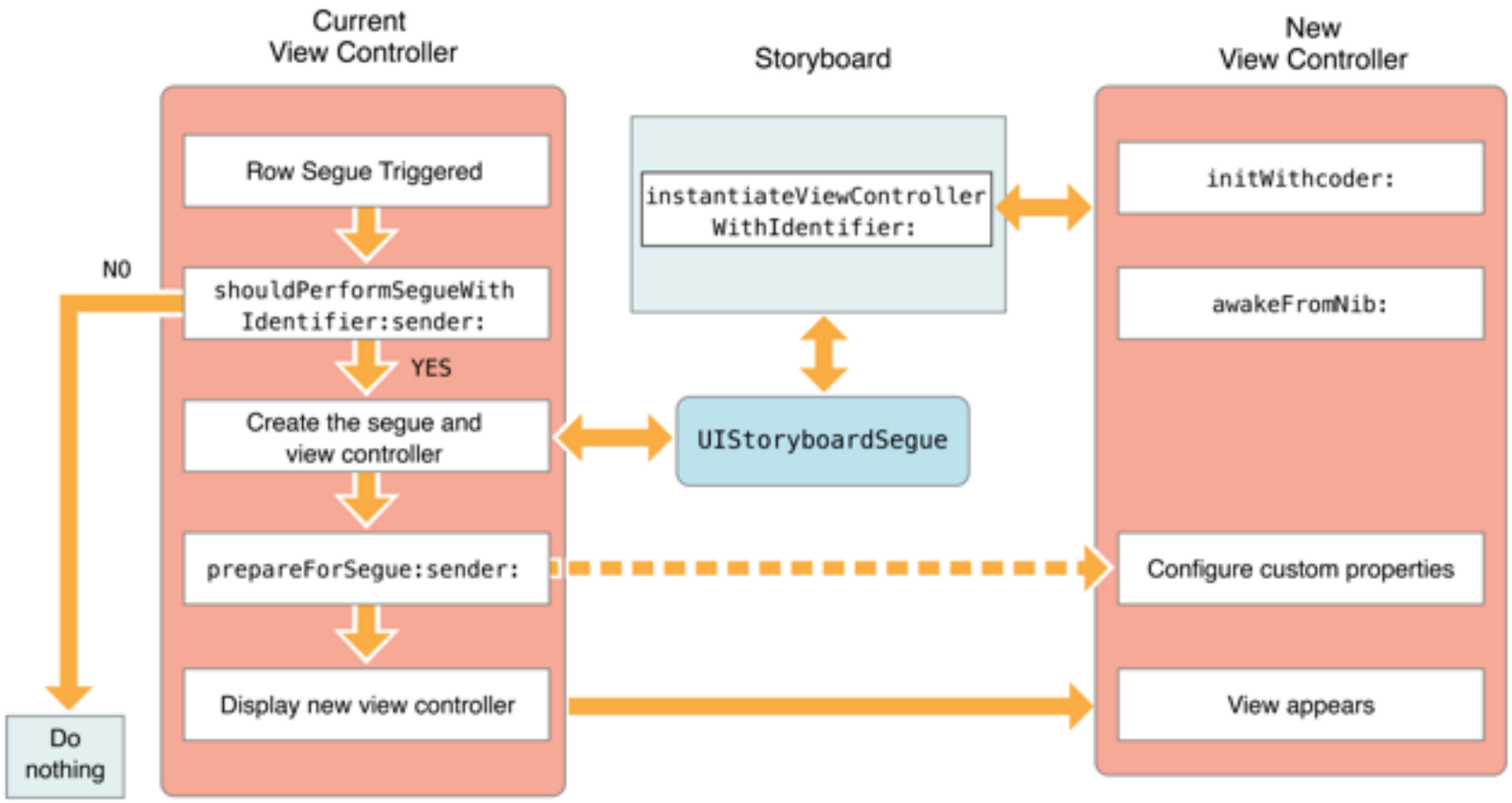
- After creating a segue, select the segue object and assign an identifier to it using the attributes inspector.



- During a segue, you can use the identifier to determine which segue was triggered, which is especially useful if your view controller supports multiple segues. The identifier is included in the UIStoryboardSegue object delivered to your view controller when the segue is performed.

segue process

- At runtime, UIKit loads the segues associated with a view controller and connects them to the corresponding elements.
- what happens when a segue is triggered



Preparing for a segue

- The **prepareForSegue:sender:** method of the source view controller lets you pass data from the source view controller to the destination view controller.

```
func prepare(for segue: UIStoryboardSegue, sender: Any?)      {
      if let identifier = segue.identifier {
            switch identifier        {
            }
}
```

The **segue** passed in contains important information about this segue:

1. the identifier from the storyboard
2. the Controller of the MVC segueing to (which was just created)

The **sender** is the instigating object (e.g. the button that is causing the segue).

Notices

- Segues always create a new instance of an MVC(except unwind)

- It is crucial to understand that this preparation is happening BEFORE outlets get set! It is a very common bug to prepare an MVC thinking its outlets are set.

Preventing segue

- Just return false from this **shouldPerformSegueWithIdentifier:sender:**meth od of the source view controller

func shouldPerformSegue(withIdentifier identifier: String?, sender: Any?) -> Bool

The identifier is the one in the storyboard.

The sender is the instigating object (e.g. the button that is causing the segue).

invoke segue from code

- Using UIViewController's method.

  func performSegue(withIdentifier: String, sender: Any?)

- Mostly, UIKit notifies that the segue is about to occur, and executes the transition. You can use the notifications to pass data to the new view controller or prevent the segue from happening altogether.

Segue's type

- Show
- Modal
- Unwind
- Popover
- Cutsom

标签导航

- UITabBarController
  - UITabBar
  - UITabBarItem
- 示例： 简历
  - Step 1：从对象库拖一个UITabBarController控件到故事板，默认有两个Scene
  - Step 2：建立UITabBarController到其它ViewController的关系segue：CTRL + Drag，选择“view controllers”
  - Step 3： 设置每个Scene中的Item的Title
  - Step 4： 设置每个Scene

树形导航

- 导航控制器（UINavigationController）与表视图结合使用，主要用于构建有“从属关系”的导航。这种导航方式采用分层组织信息的方式，可以帮助构建效率型应用程序。

- 例子： FoodTracker