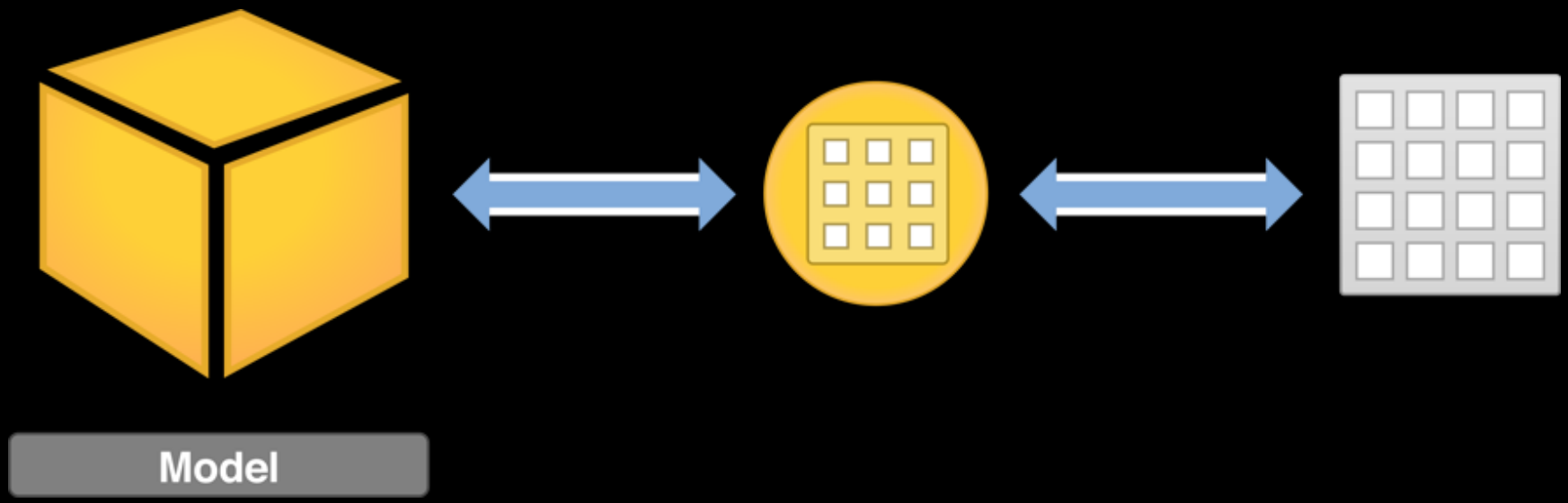
Mobile Application Development
Aileen Pierce

MULTIPLE VIEW CONTROLLERS

Model-View-Controller(MVC)

- MVC is the foundation for iOS app code
- Model: holds the data and classes
  - Should be UI independent
- View: all items for the user interface
- Controller: links the model and the view together. The backbone or brain of the app.
- These categories should never overlap.
- The goal of MVC is to have any object be in only one of these categories.

Model-View-Controller(MVC)



Model

- Model classes manage your data and often map to real-world objects.
- Models should be independent of the view and the controller.
- Model classes should be reusable.

View

- Views hold the user interface for your app
- Everything you do in Interface Builder to build your interface can be done programmatically

Controller

- View controllers act as the glue connecting the view and the model.
- View controller classes need access to the view and the model.
- Controllers are usually paired with a single view that they manage.
- Apps that have multiple views need multiple view controllers

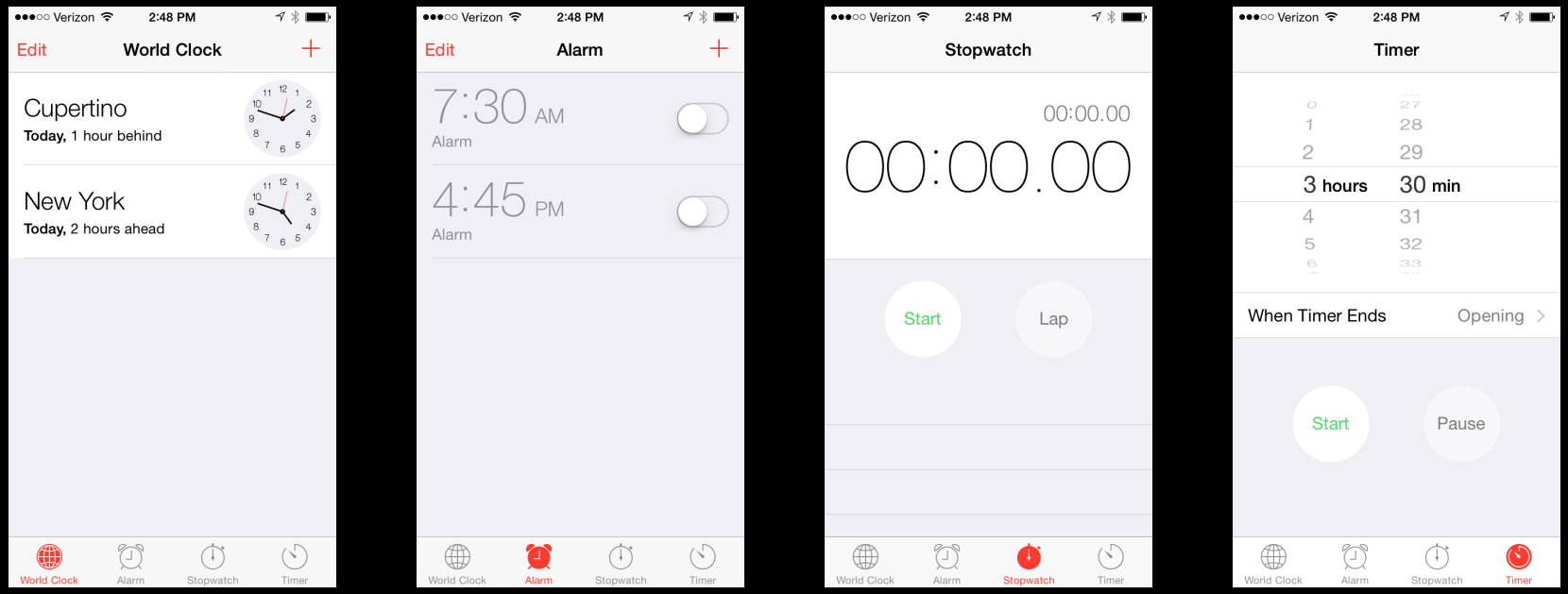
Multiple Controllers

- A modal view controller presents another view on top of the current view
- A tab bar controller organizes views in a tabbed list
- A navigation controller organizes views in an hierarchy, often a table view
- A split view has the navigation in the left column and the content on the right.

Modal View Controller

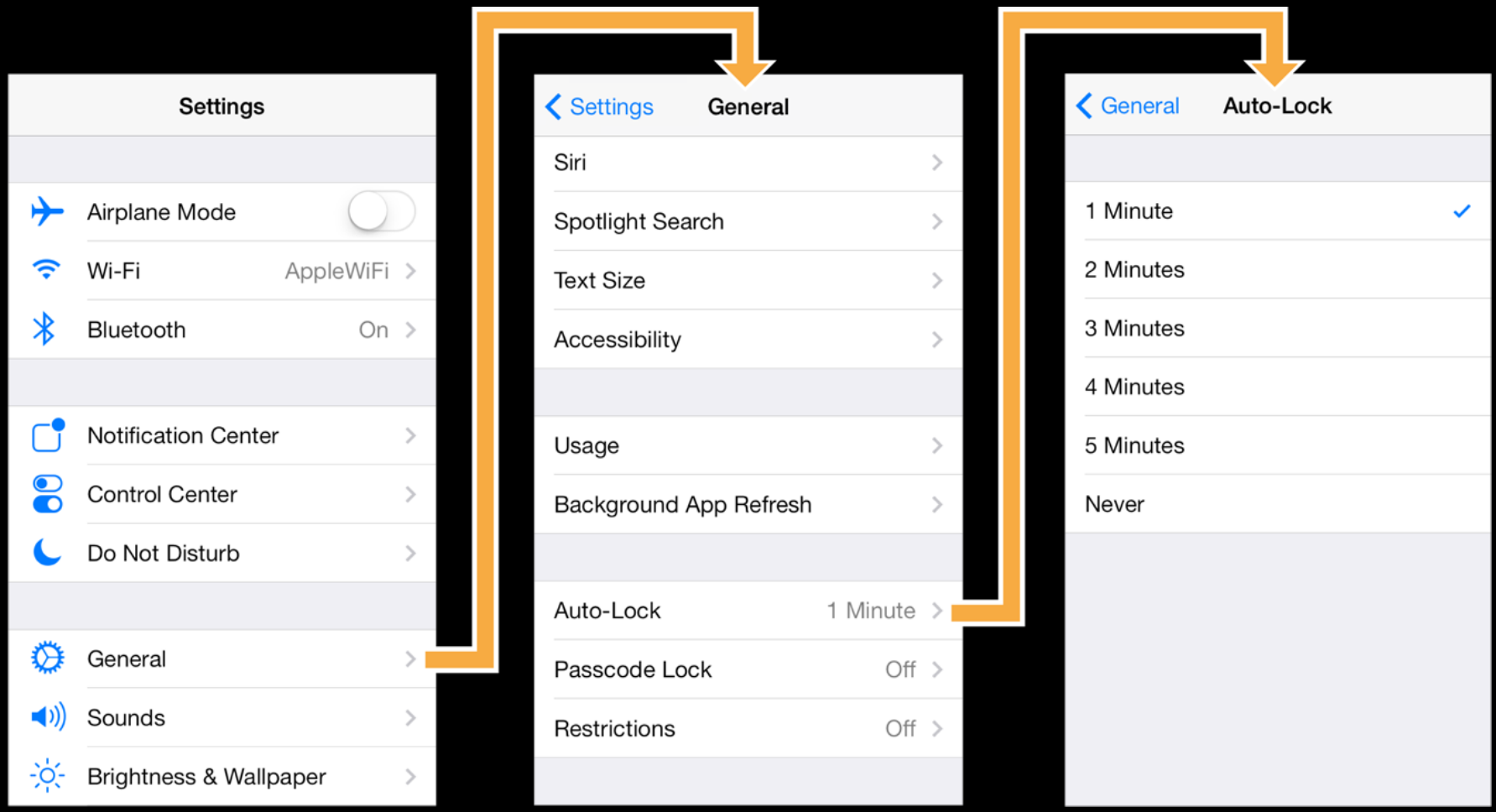
- In utility apps the main view is dedicated to a single task
- A second view is used for configuration settings or to provide more information.

Tab Bar Controller



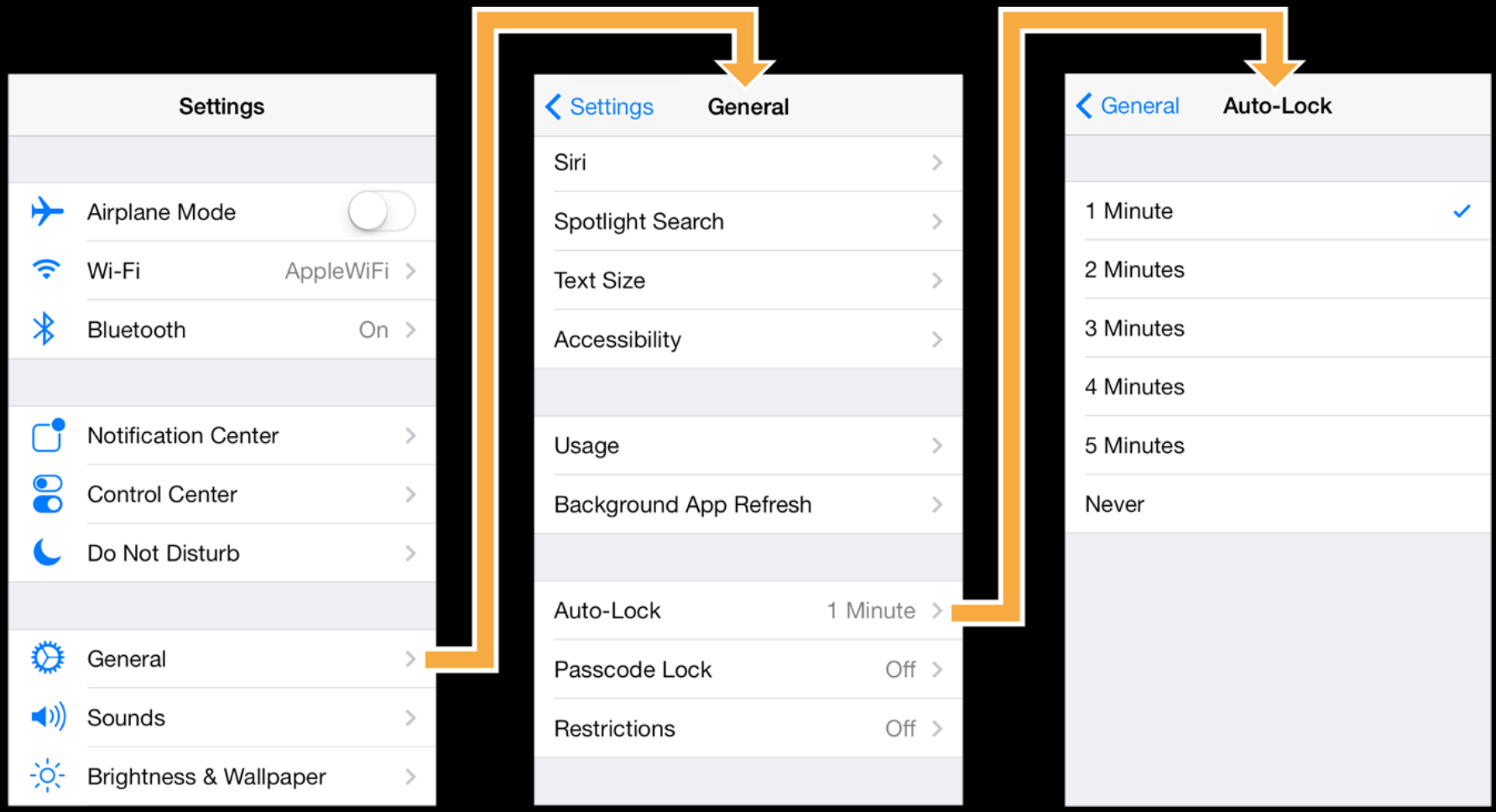
Tab bar interfaces are commonly used either to present different types of information or to present the same information using a completely different style of interface.

Navigation Controller



Navigation controllers manage a hierarchy of views.

Navigation Controller



Navigation controllers manage a hierarchy of views.

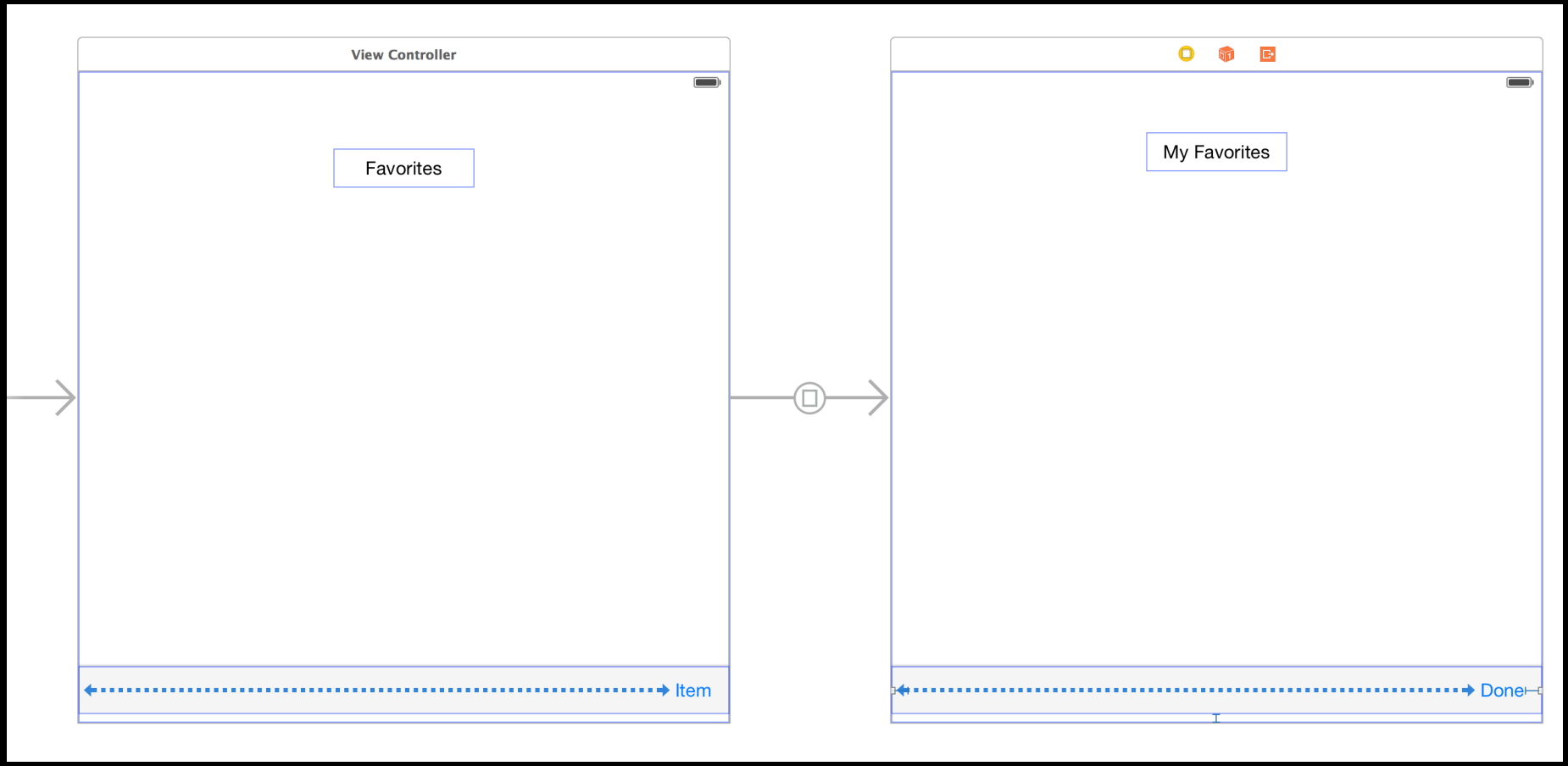
Multiple Controllers

- The primary view controller for an app is called the root controller.
  - Its job is to present views based on the user's input
- On iPhone and iPod touch, the presented view is always full screen.
- Each view is typically controlled by its own controller class.
- A view controller and its view are called a scene.

Storyboards

- Storyboards provide a single visual workspace to build apps with views
- Easily let's you define transitions between views
- Provides a good conceptual overview of all the scenes in an app

Storyboards



Segues

- Segues define how one scene transitions to another scene.
- Makes it easier to visually see and edit the entire flow of your app.
- A segue is called in a controller whenever a controller's view is about to be replaced by another controller's view

Segues

- Three types of storyboard segues
  - Show
    - Shows a view controller in a navigation controller
  - Show detail
    - Replaces controller in split view controllers
  - Present modally
    - One view controller presents another view controller
  - Popover presentation
    - Presents a popover controller
  - Custom
    - A segue that uses a custom class

Segues

- **`UIStoryboardSegue`** class
- **`prepareForSegue(segue, sender)`** is called whenever a segue is being activated but before the transition occurs
- The **`destinationViewController`** property tells us which VC is about to be displayed
- The **`sourceViewController`** property tells us which VC is about to be removed from the display
- The **`identifier`** property is an NSString that can be used to identify the segue

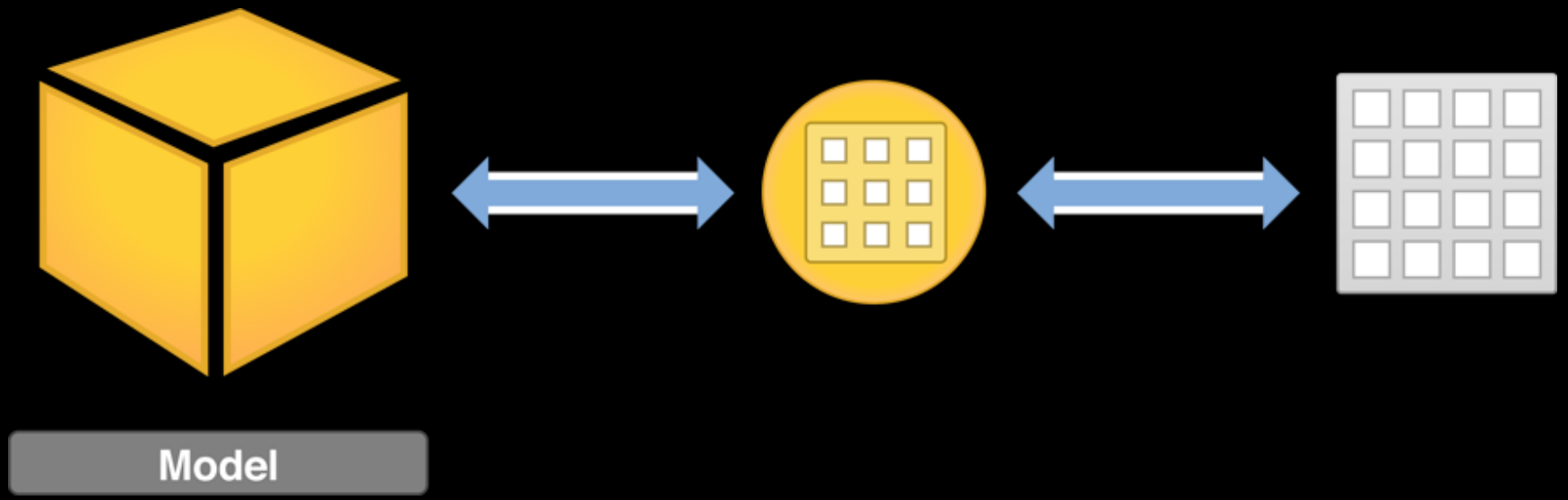
Unwind Segues

- Unwind segues allow you to define the relationship for reversing a segue transition
  - **`IBAction`** method
  - **`UIStoryboardSegue`** parameter
- Define a method in your destination VC to be called by the unwind segue
- Connect the element in the scene to the Exit icon and select the unwind method

Passing Data

- Create the segue and give it a unique identifier
- Implement **`prepareForSegue(segue, sender)`**
- Use the **`destinationViewController`** property to create an object to access the destination VC
- Assign the data to properties of the destination VC
- Use the unwind method or **`viewWillAppear()`** to refresh any needed elements in the destination view

Model-View-Controller(MVC)



- Model classes manage your data
- Create objects of your model class in any VC that interacts with that data