

Practical MVP and MVVM Patterns for your iOS Apps

Priya Rajagopal

Twitter: @rajagp

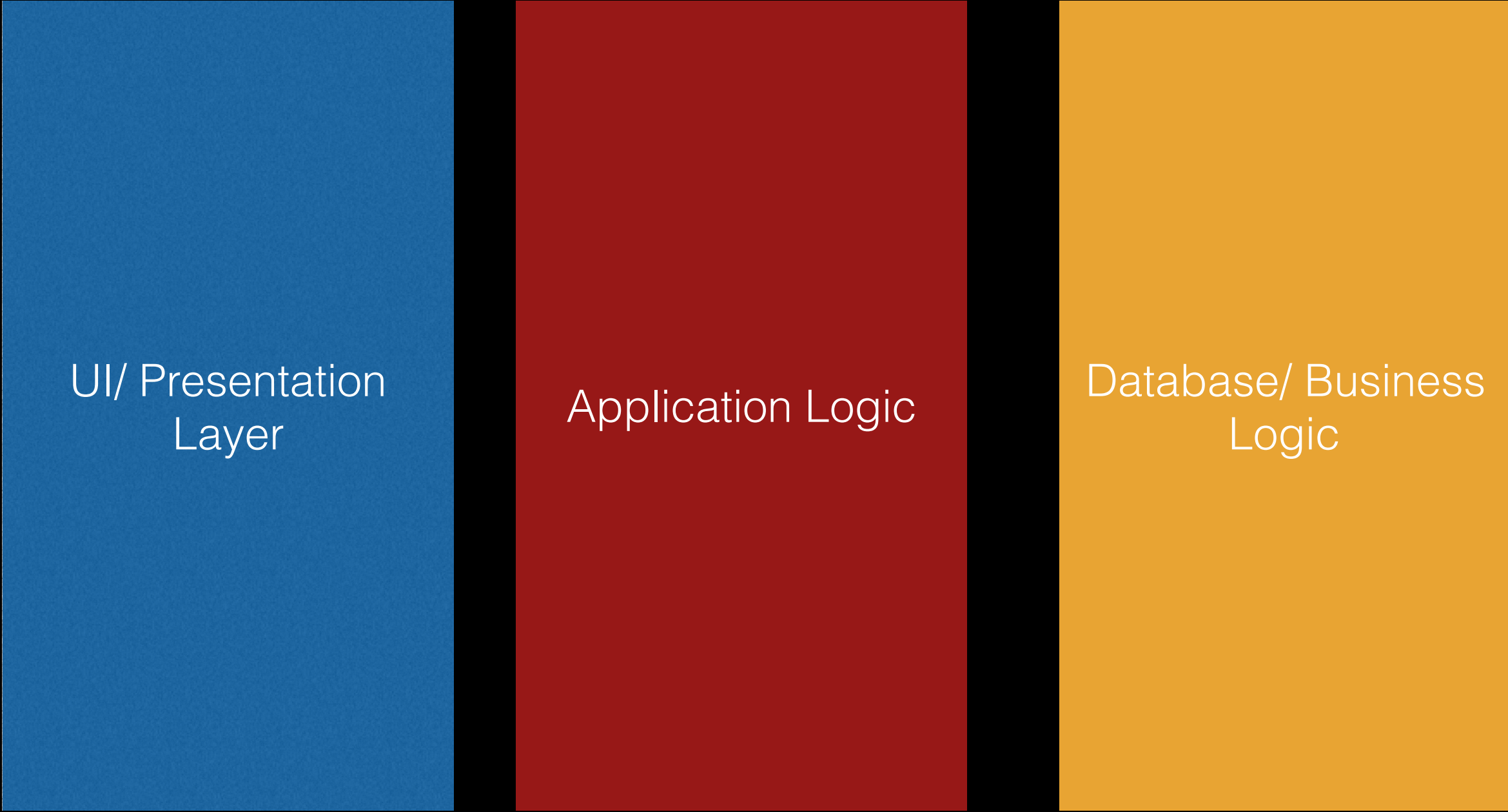
blog: www.priyaontech.com

Software Pattern



- Template to solve a repeatable problem
- Architectural Patterns
- Design Patterns

App Architecture

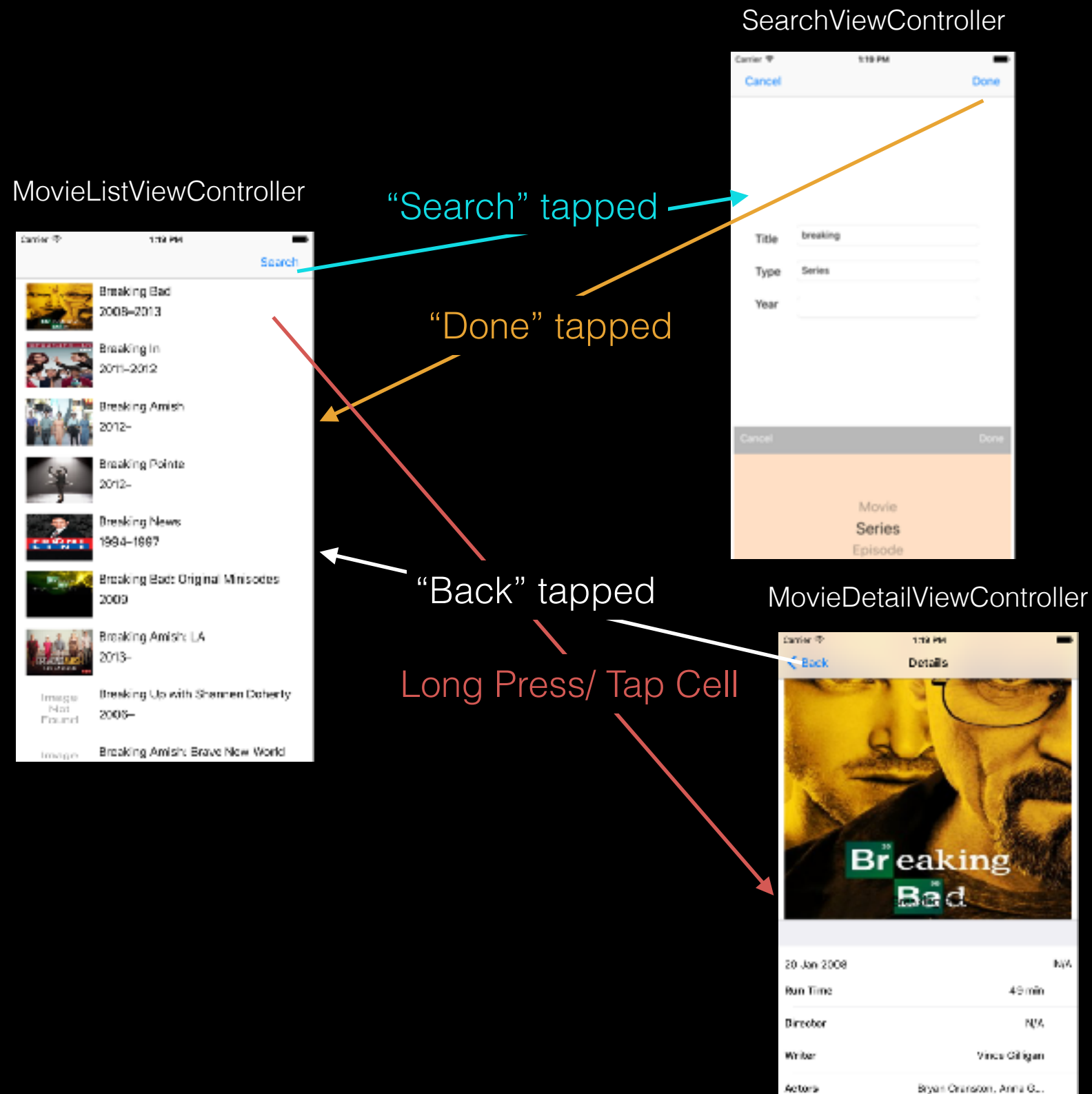


UI/ Presentation
Layer

Application Logic

Database/ Business
Logic

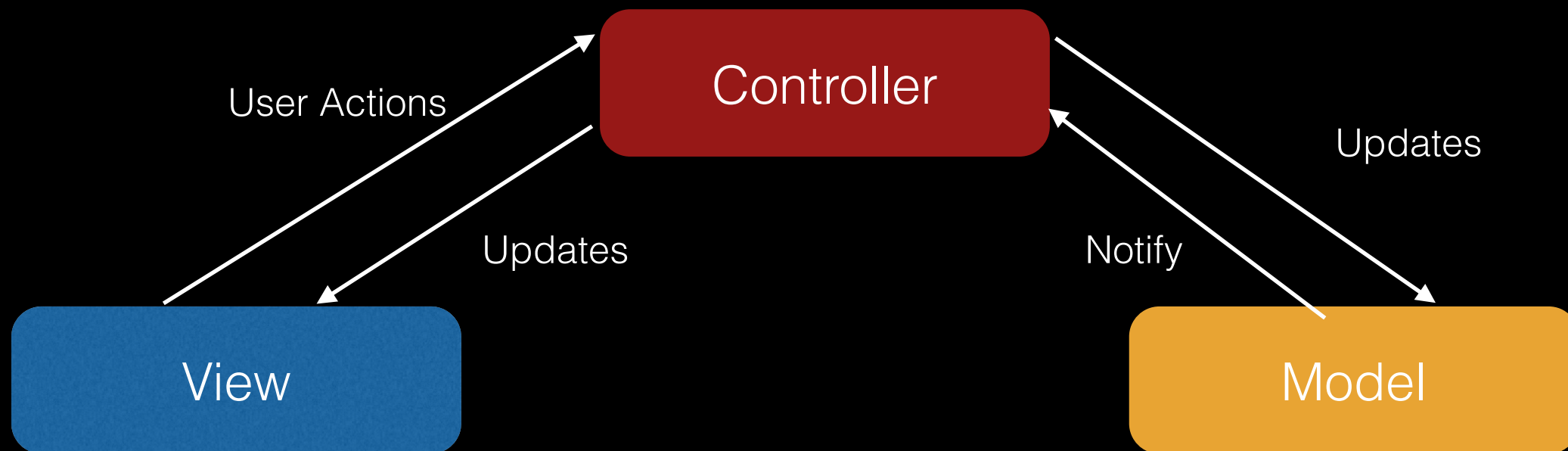
MovieBuff - App Logic



Demo:

MVC

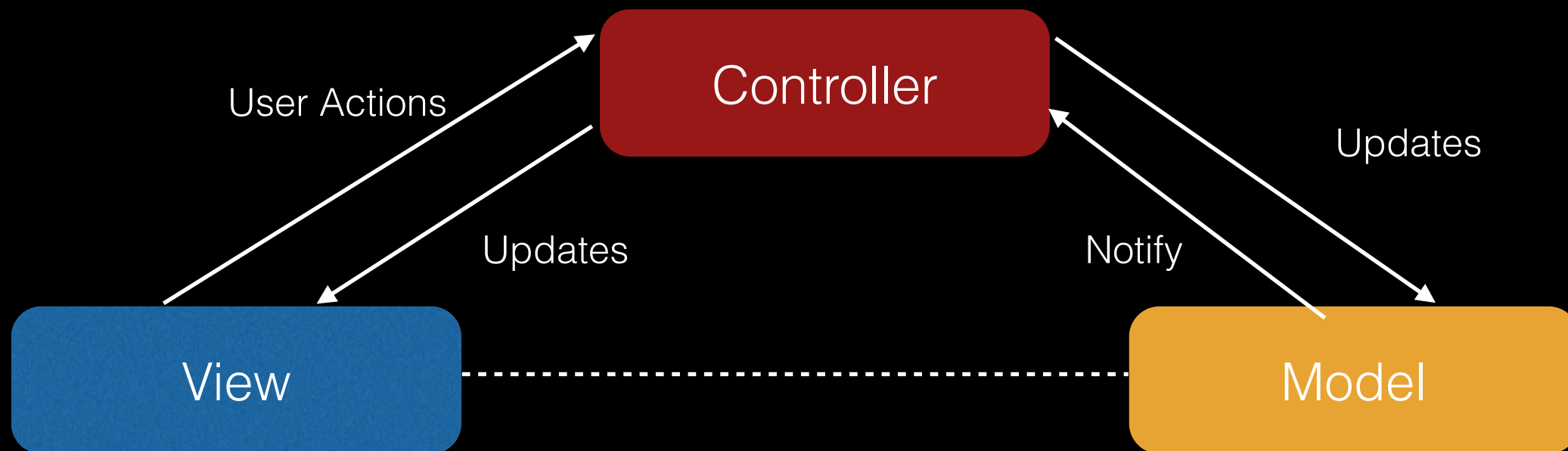
Model View Controller



- UI/ Presentation Layer : View
- Data/ Business Logic Layer: Model
- Application Logic: Controller
- Theoretically, Model - View Direct Communication Allowed
 - iOS advocates all communication through controller
- iOS UIViewController = View or Controller ?
- View: Controller = Many:1
- Views not passive

MVC

Model View Controller



- UI/ Presentation Layer : View
- Data/ Business Logic Layer: Model
- Application Logic: Controller
- Theoretically, Model - View Direct Communication Allowed
 - iOS advocates all communication through controller
- iOS UIViewController = View or Controller ?
- View: Controller = Many:1
- Views not passive

The Controller

- Delegate
- Data Source
- Navigation
- Target-Actions



The Controller

- Delegate
- Data Source
- Navigation
- Target-Actions



Easy To go wrong ...

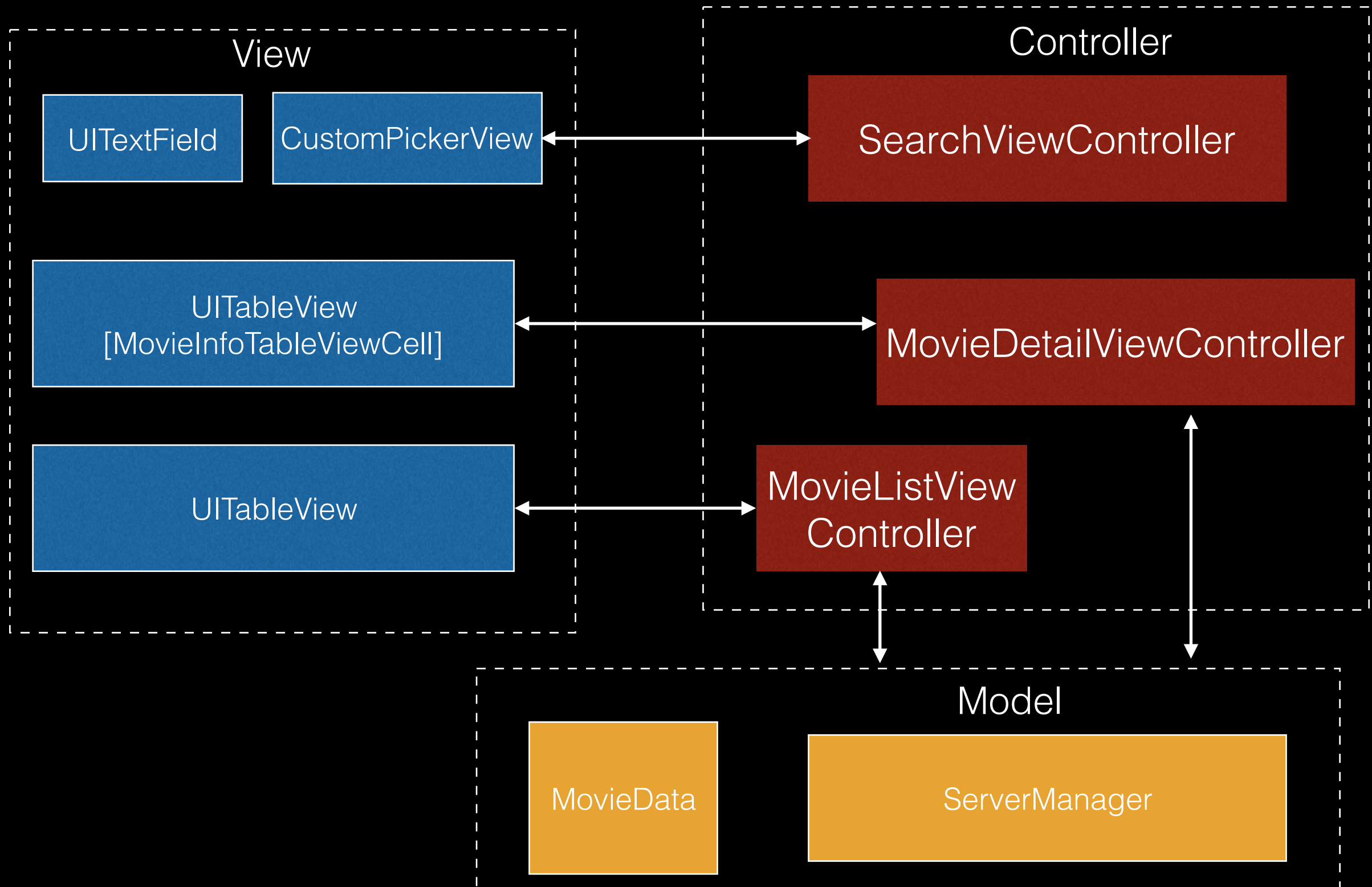
The Controller

- Delegate
- Data Source
- Navigation
- Target-Actions



Easy To go wrong ...

MVC in MovieBuff



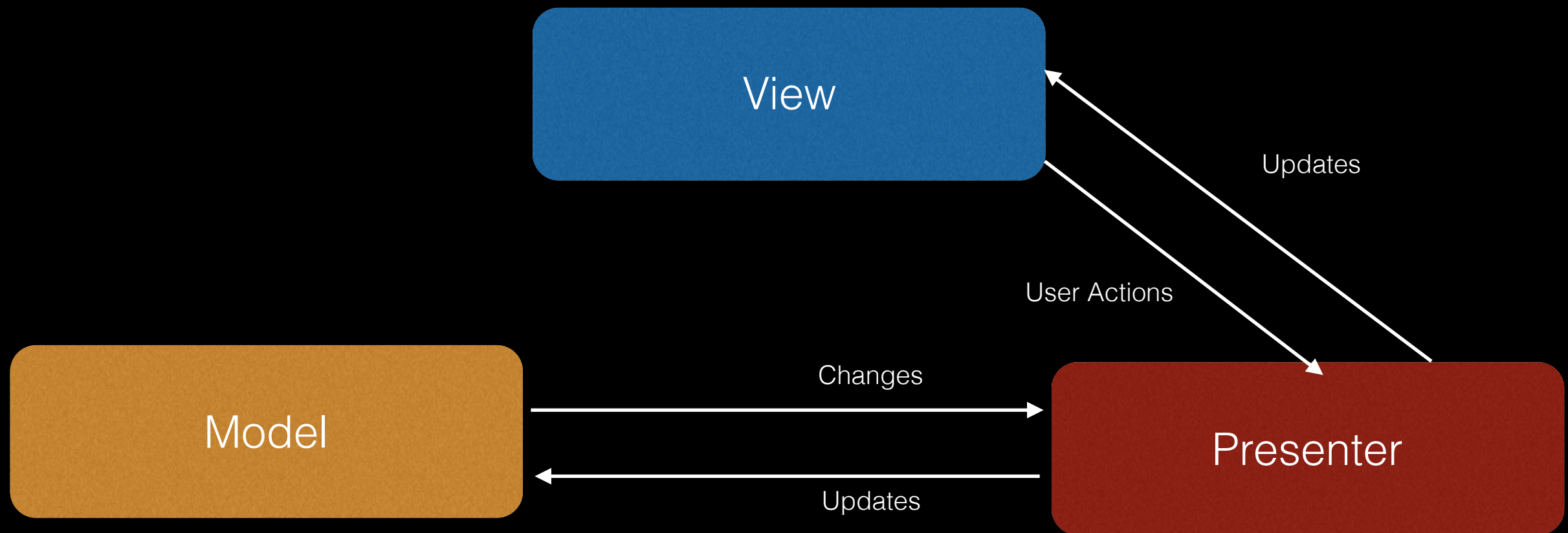
SHOW ME

CODE!

memegenerator.net

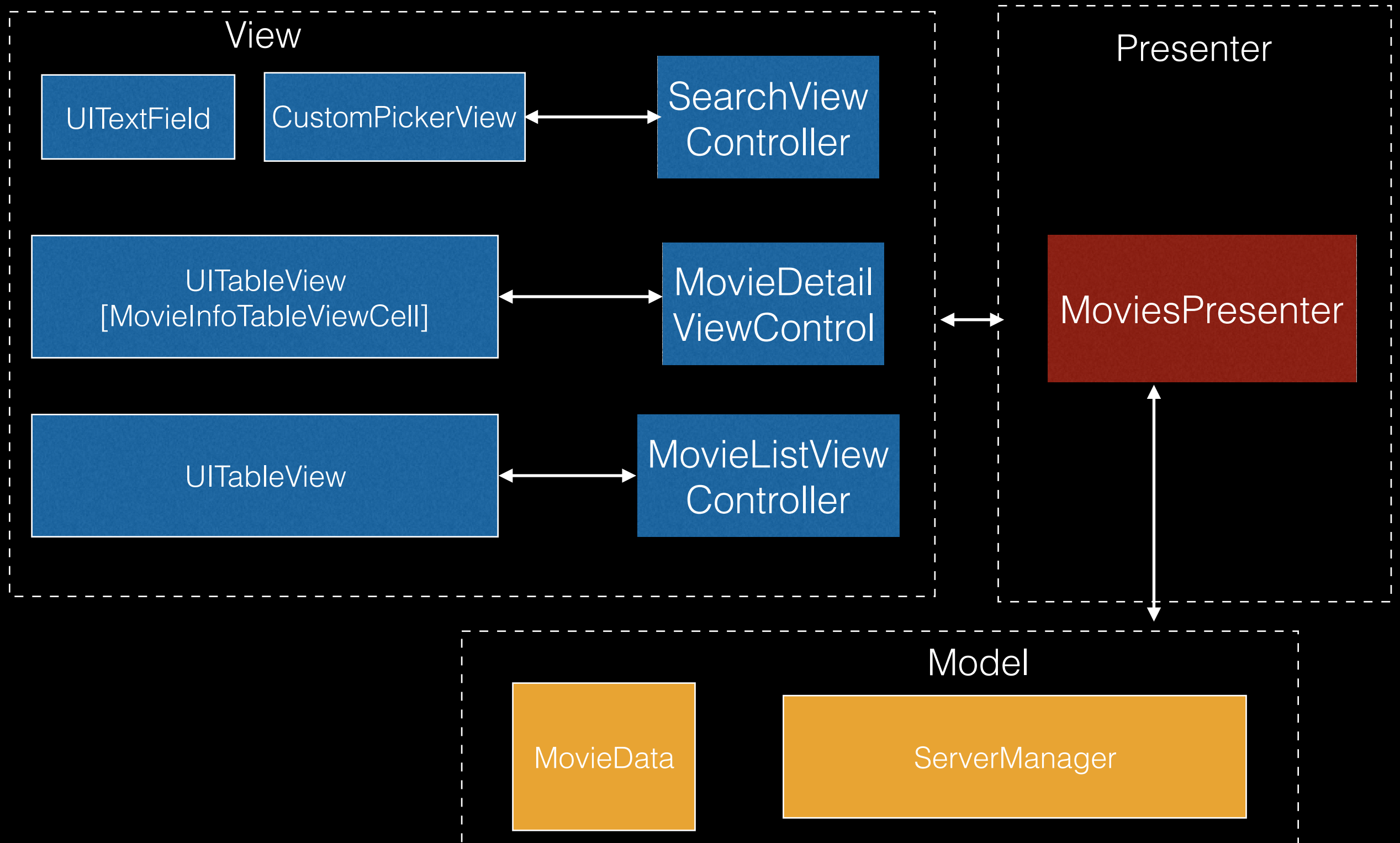
MVP

Model View Presenter

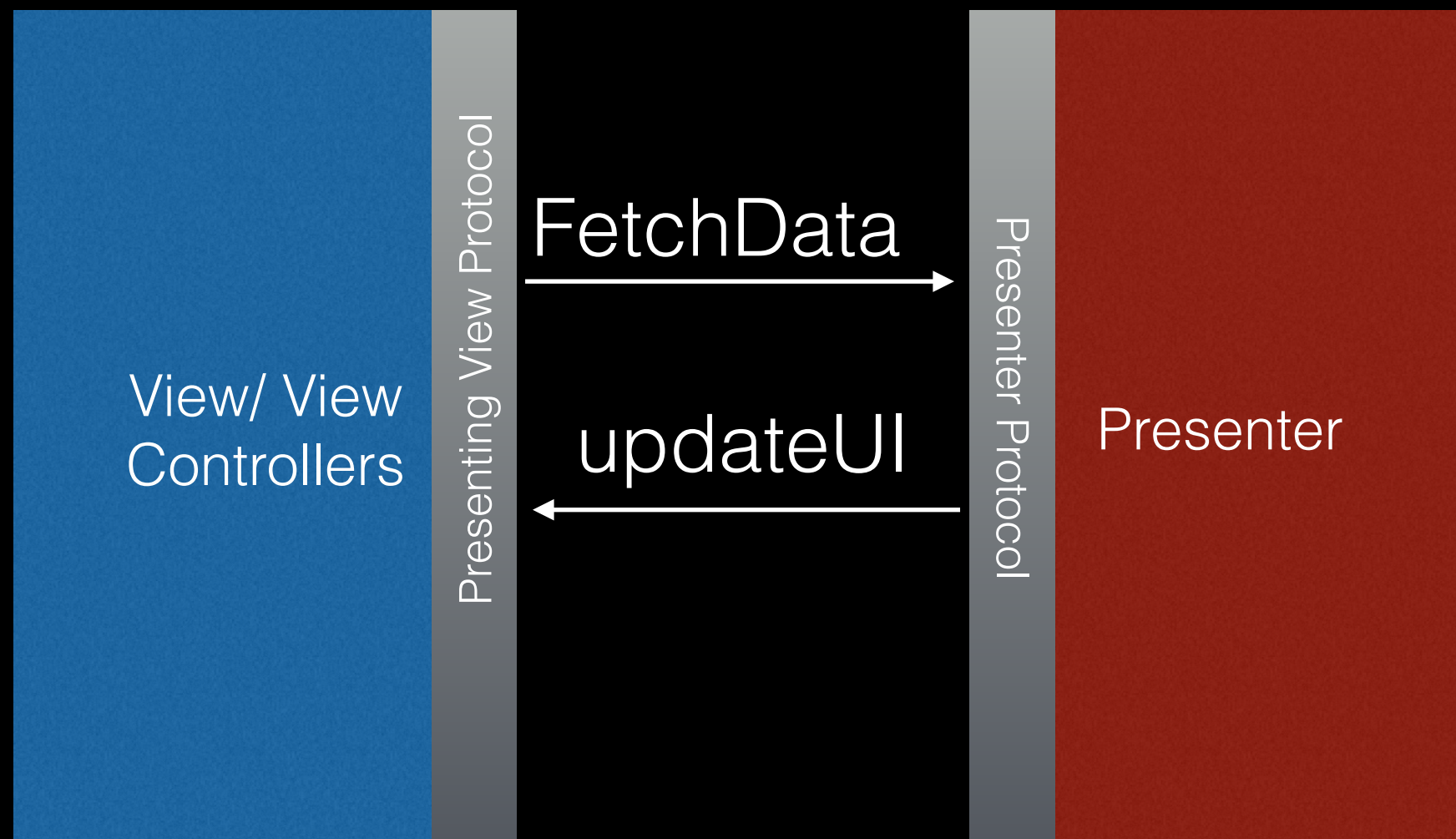


- A Variant of MVC
- UI/ Presentation Layer : View
- Data/ Business Logic Layer: Model
- Application Logic: Presenter
- Presenter *ideally* should handle all presentation logic
- View is passive
- View: Presenter typically 1:1
- Strict separation between View and Model

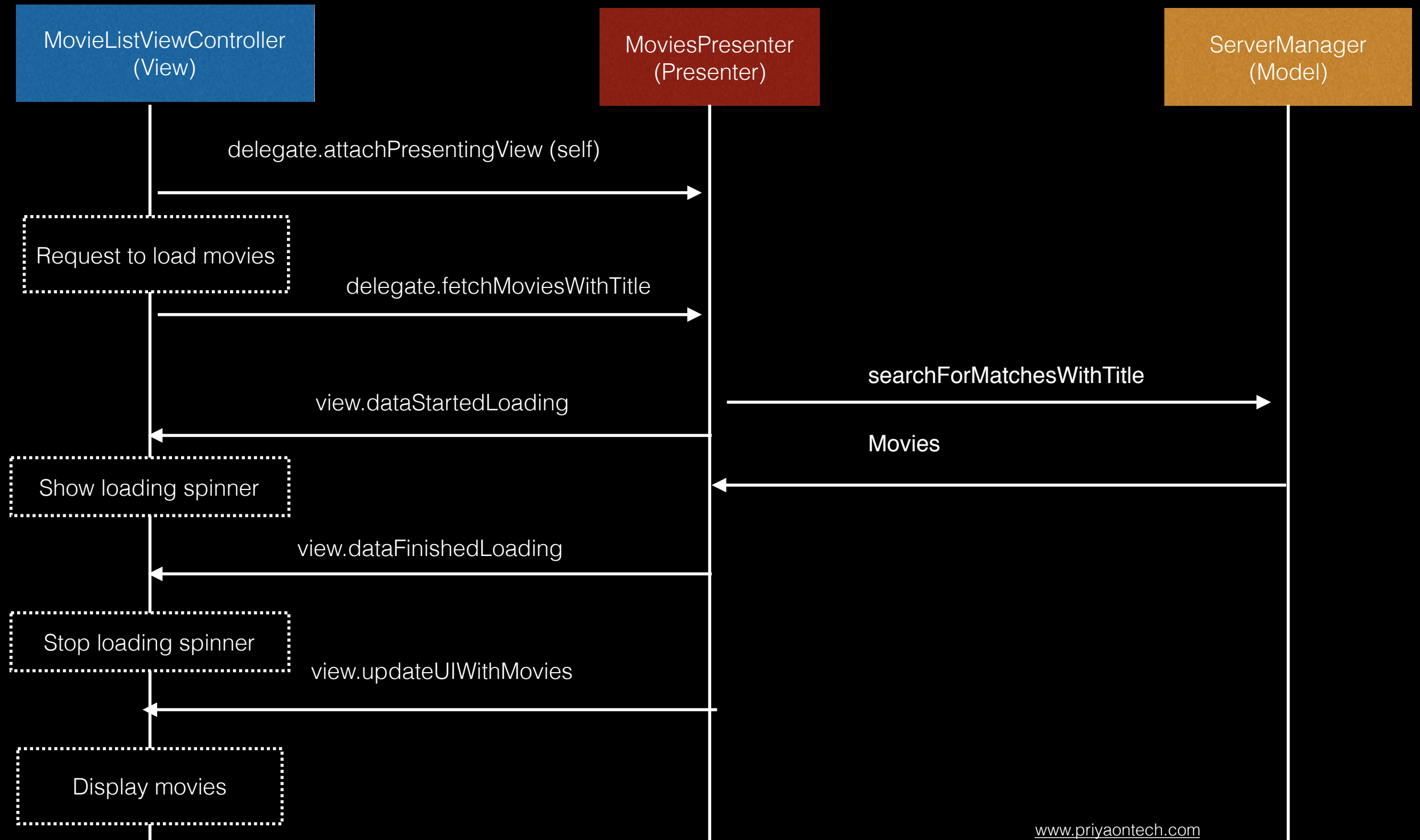
MVP in MovieBuff



View-Presenter w/ Delegation Pattern



MVP Flow



SHOW ME THE

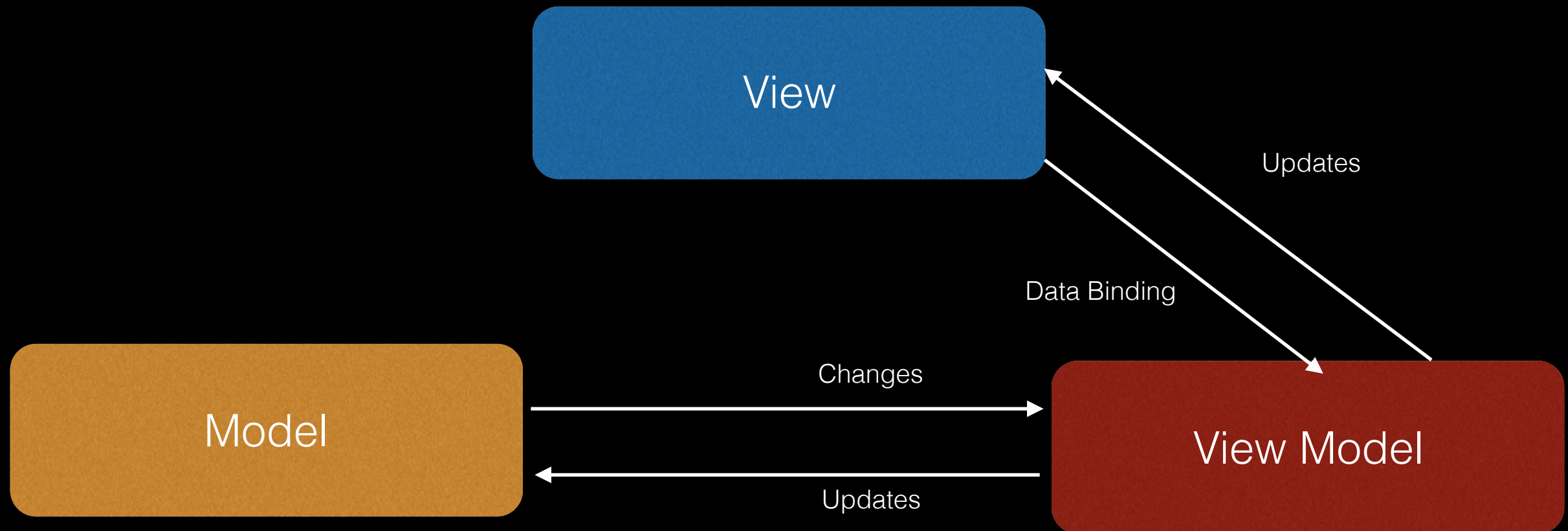


CODE!

memegenerator.net

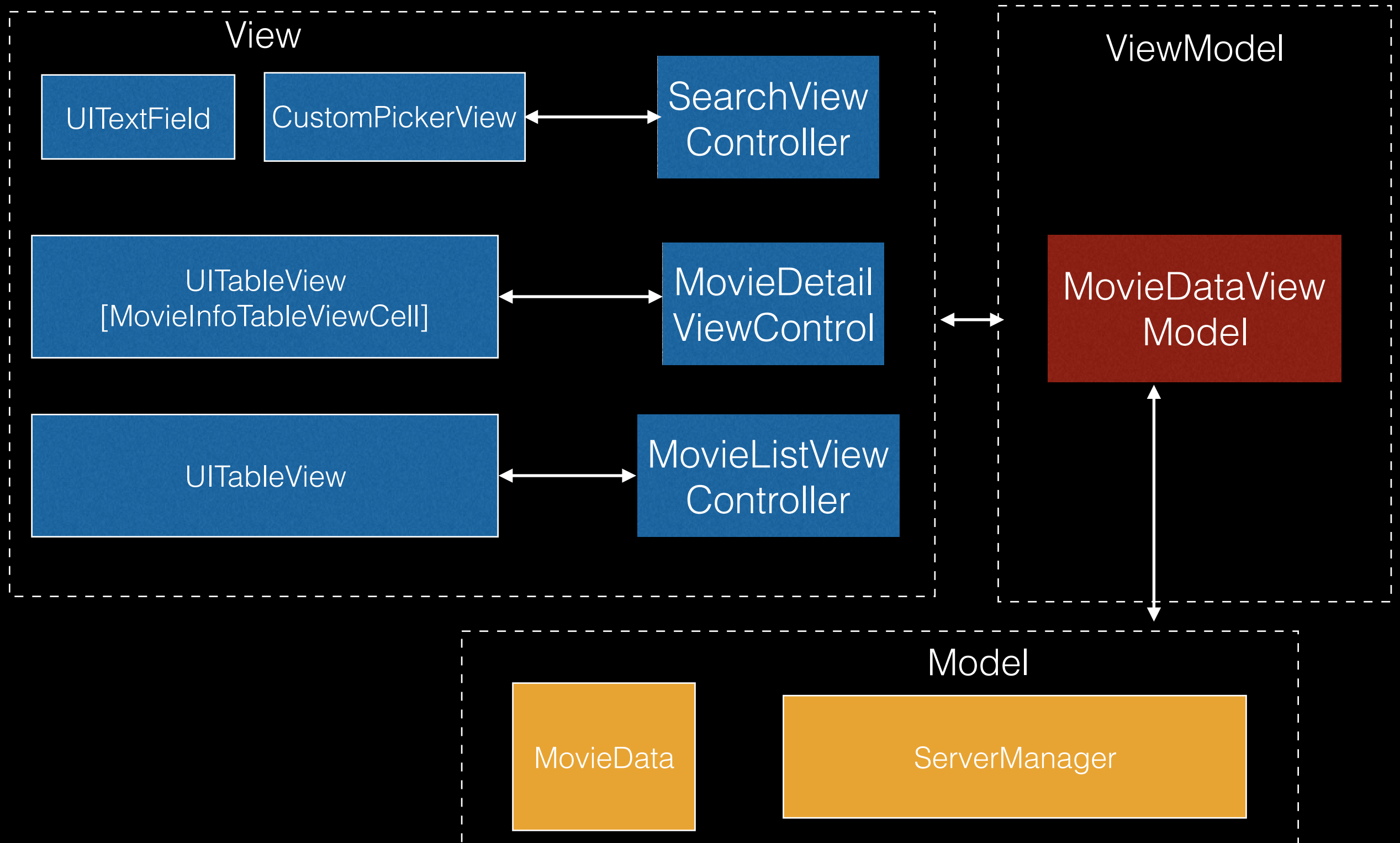
MVVM

Model View View Model



- A Variant of MVC
- UI/ Presentation Layer : View
- Data/ Business Logic Layer: Model
- Application Logic: Model View
- “Binding” between view and view-model
- View: Model View could be 1:Many

MVVM in MovieBuff



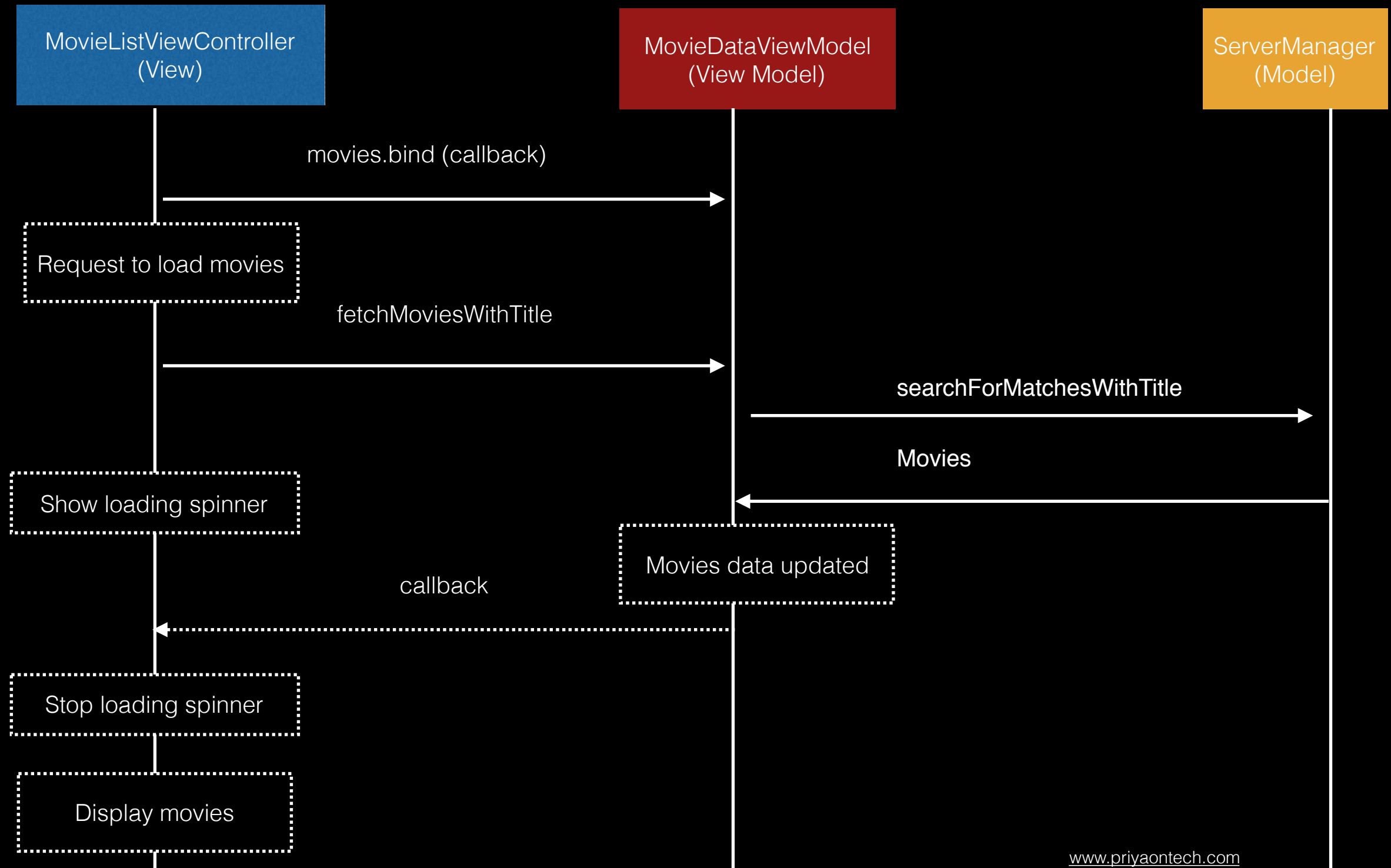
View-View Model Binding using Key-Value Observing (KVO)

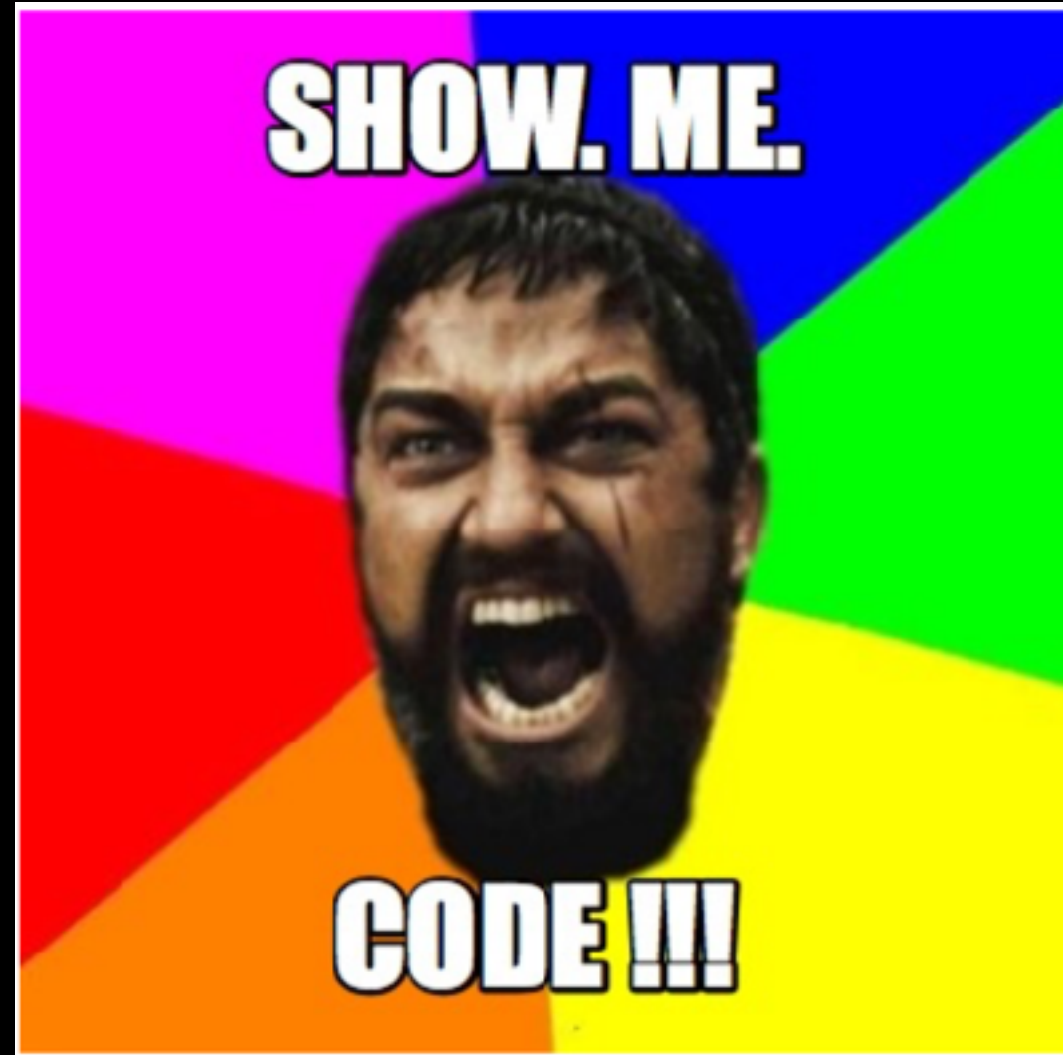
- KVO in Cocoa :
 - Object can be automatically notified of changes to state of another object
 - Observe properties on other object
- In ObjC - built into the runtime
- Swift - Not out of the box
 - One option dynamic modifier , NSObject only , ugly syntax
 - Another option ...

Wrapper around Observable Properties

```
public struct DynamicType<T> {  
    typealias ModelEventListener = (T)->Void  
    typealias Listeners = [ModelEventListener]  
  
    private var listeners:Listeners = []  
    var value:T? {  
        didSet {  
            for (_,observer) in listeners.enumerated() {  
                if let value = value {  
                    observer(value)  
                }  
            }  
        }  
    }  
  
    mutating func bind(_ listener:@escaping (T)->Void) {  
        listeners.append(listener)  
        if let value = value {  
            listener(value)  
        }  
    }  
}
```

MVVM Flow Example





In Summary....

- MVC, MVP, MVVM have a common goal
 - Separation of concerns
 - Centerpiece is Controller/Presenter/ View Model
- MVP and MVVM are variants of MVC
- Don't get bogged down by semantics
- On iOS
 - MVP using Delegation Pattern
 - MVVM using Key-Value Observer Pattern

Thank You !

Priya Rajagopal
Twitter: @rajagp