Adapting CLEAN Architecture in Android Apps

UNDERSTANDING ARCHITECTURAL PATTERNS

Kaushal Dhruw
APP DEVELOPER / AUTHOR

@drulabs linkedin.com/in/kaushal-dhruw/

Overview

- Importance of software architecture
- CLEAN architecture
- Local & remote data sources
- Unit & instrumentation testing
- Delivering data to UI

Prerequisites

A basic experience with:

- Android Studio
- Android app development
- Kotlin
- Reactive Programming (RxJava)

Goals

CLEAN Architecture

Pragmatic Reactive Programming

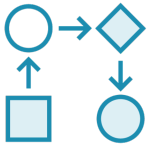
Android Architecture Components

Dependency Injection (Dagger2)

Testing with JUnit And Espresso

Change is inevitable

CLEAN Architecture Produces



Loosely coupled systems

Framework independent systems

UI independent systems

Why Architecture?

Construction architecture is NOT about

- Bricks and mortar
- Placement of light bulbs
- Type of glass for windows
- Shape of tiles
- Or color of curtains

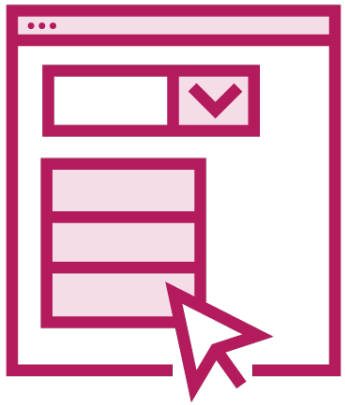
Construction architecture is about

- Number and size of rooms
- Placement of rooms and garden
- Accessibility of each room
- Number of floors
- Provision for air conditioning, communications etc.

Software Architecture Is Not About

Tools

(e.g. Database and libraries)



User Interface

(e.g. animations and theme)

External Agencies

(e.g. third party services)

Software Architecture

Software architecture refers to the high level structure of a software system. Each structure comprises of components, their properties and relationship with other components.

Why Software Architecture?

- Incorporate changes quickly
- Reduce cost
- Readability
- Easy communication and onboarding
- Testability

Testability

High Testability

- Reduced testing efforts
- Easy to find bugs
- Low coupling
- Quick changes

Untestable Code

```
class GreetingCreator(private val name: String){

    val calendar = Calendar.getInstance()

    fun generate(): String {
        return when (calendar.get(Calendar.HOUR_OF_DAY)) {
            in 5..11 -> "Good Morning $name!!"
            in 12..17 -> "Good Afternoon $name!!"
            else -> "Good Evening $name!!"
        }
    }
}
```

Making GreetingCreator Testable

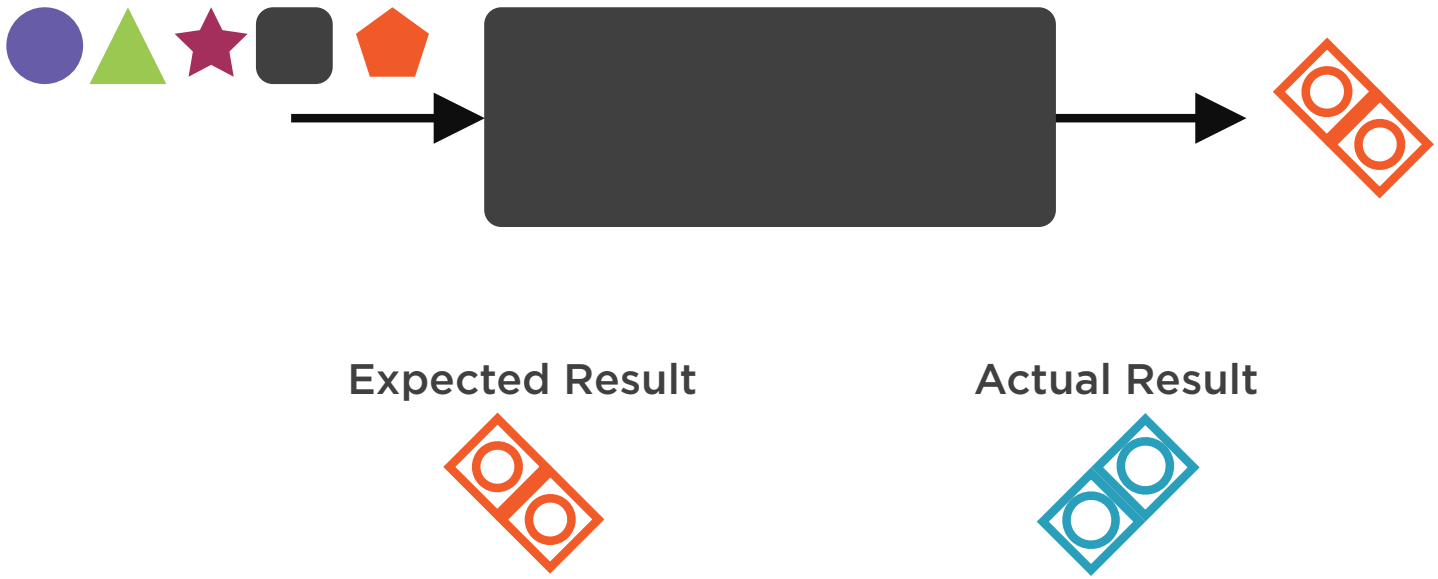
```
class GreetingCreator(
    private val name: String,
    private val calendar: Calendar
) {
    fun generate(): String {
        return when (calendar.get(Calendar.HOUR_OF_DAY)) {
            in 5..11 -> "Good Morning $name!!"
            in 12..17 -> "Good Afternoon $name!!"
            else -> "Good Evening $name!!"
        }
    }
}
```

Testing GreetingCreator

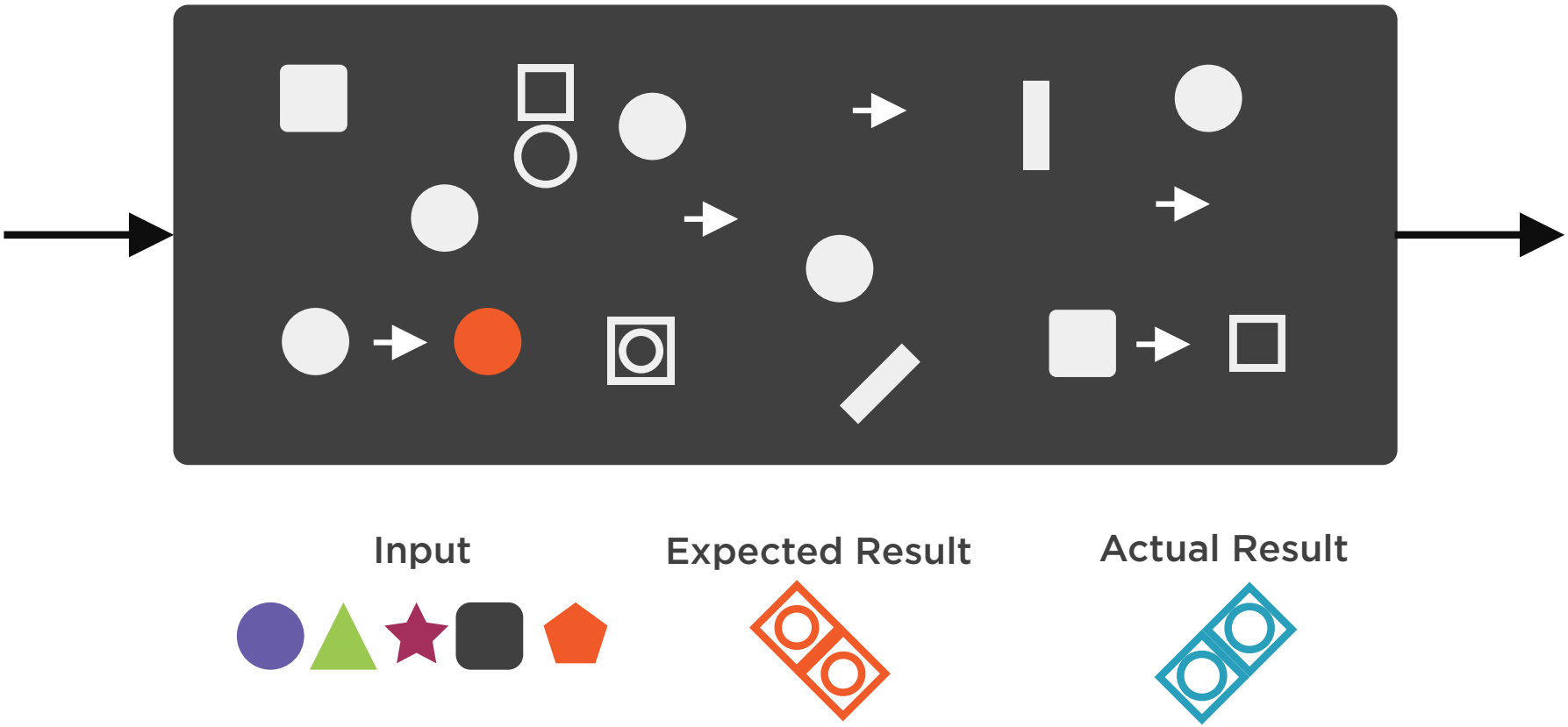
```
// Set the time to morning time
calendar.set(HOUR_OF_DAY, 7) // 07:00 AM
val greeting = GreetingCreator("John”, calendar)
```

```
// Set the time to evening time
calendar.set(HOUR_OF_DAY, 22) // 10:00 PM
val greeting = GreetingCreator("John”, calendar)
```

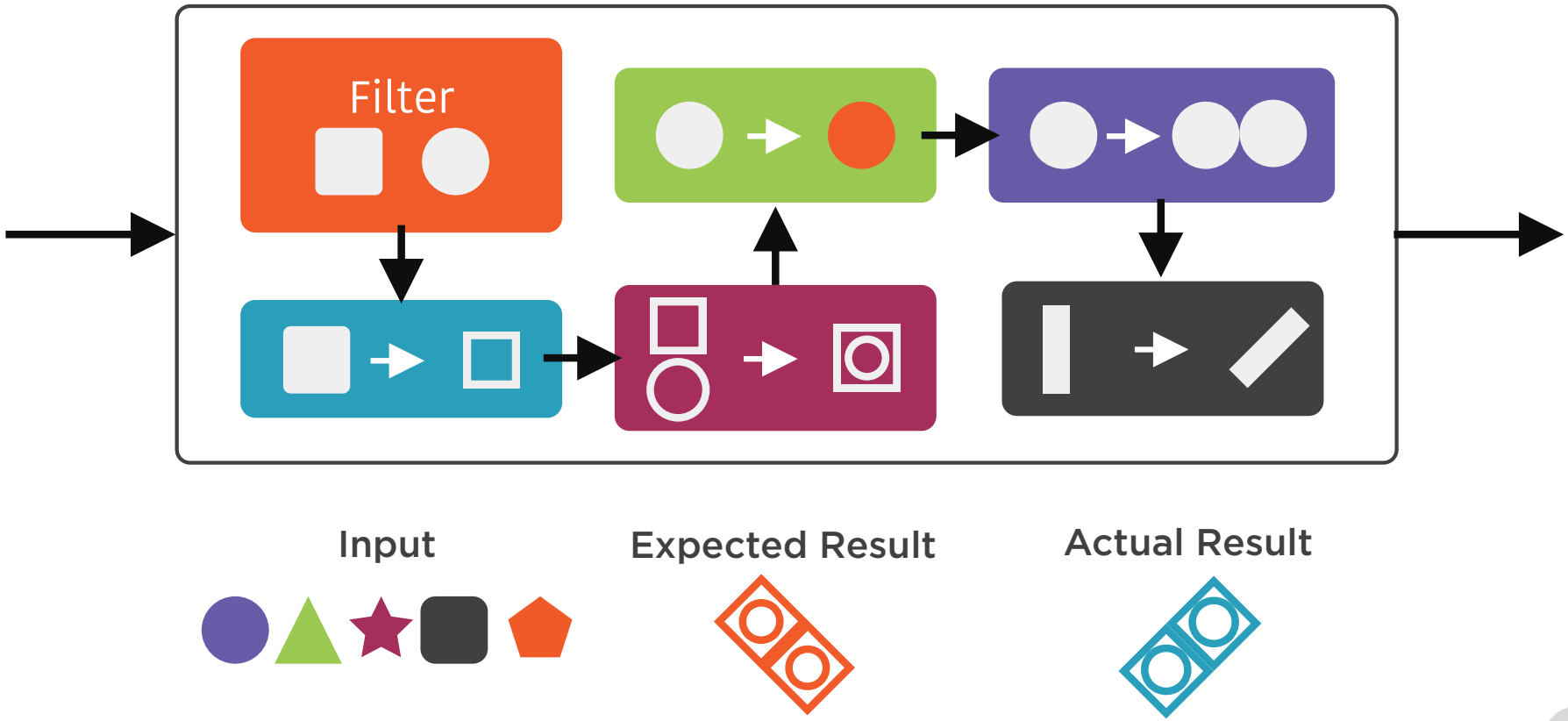
Testing a Software Module



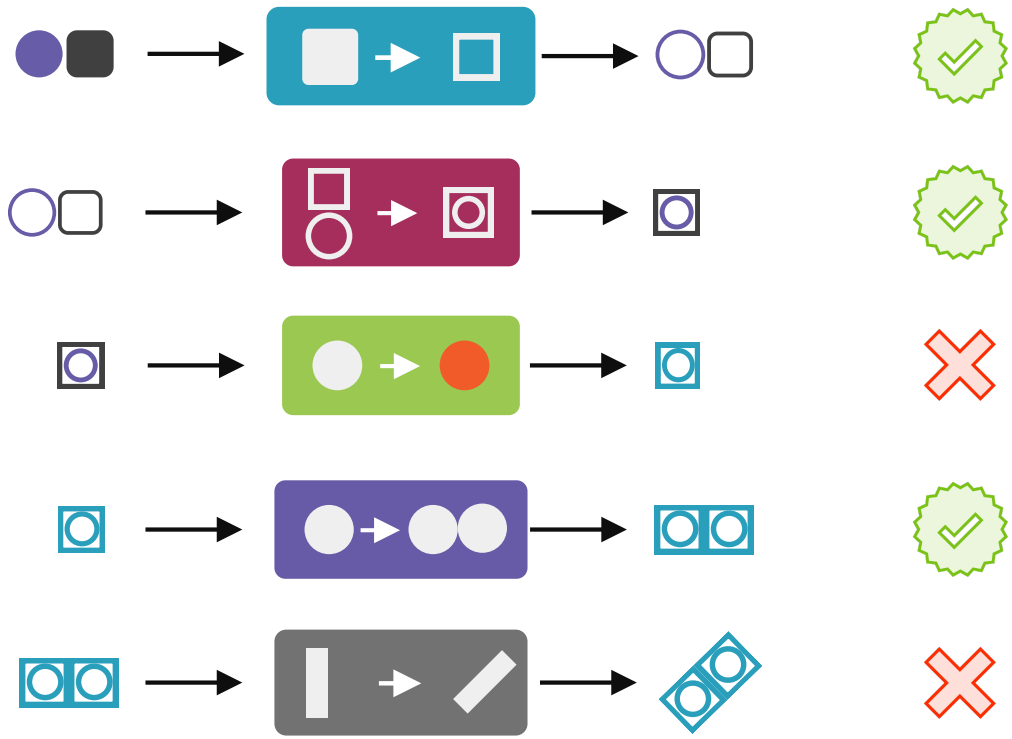
An Untestable Module



A Testable Module

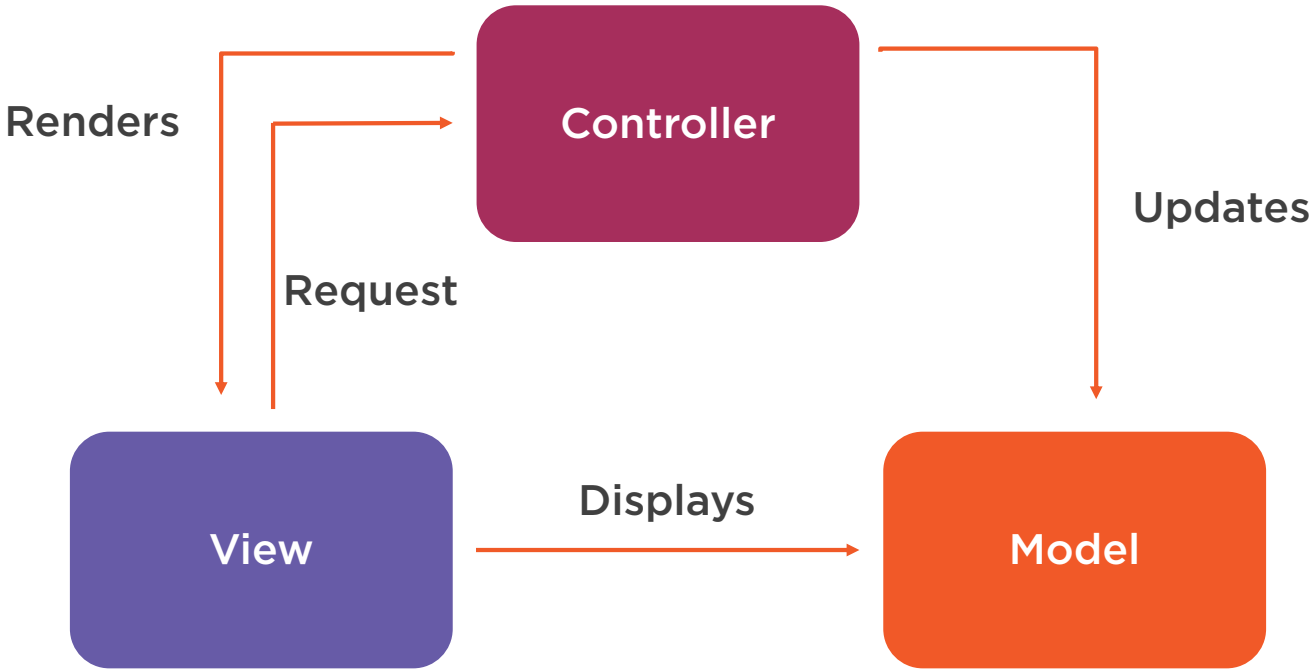


Testing Individual Components

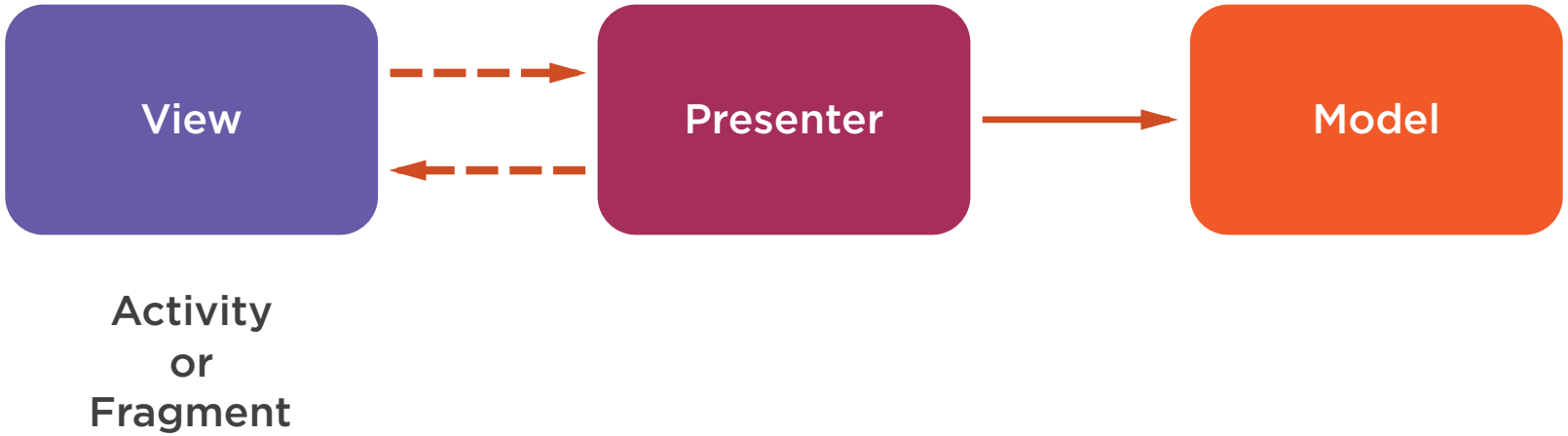


MV* Architectural patterns

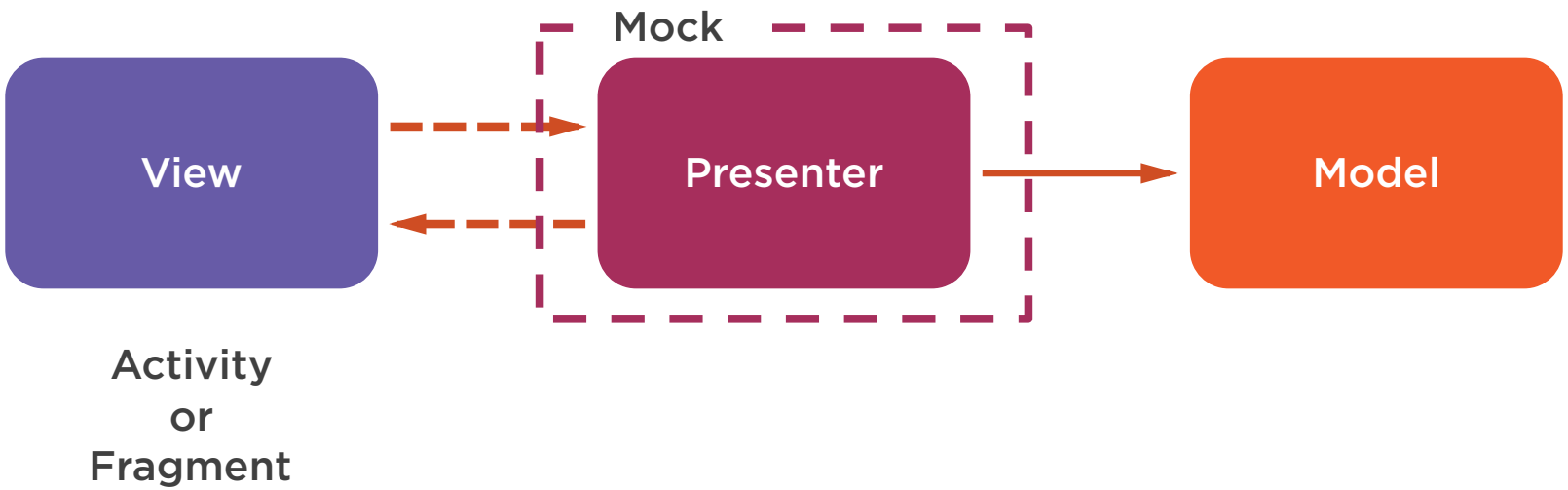
MVC – Model View Controller



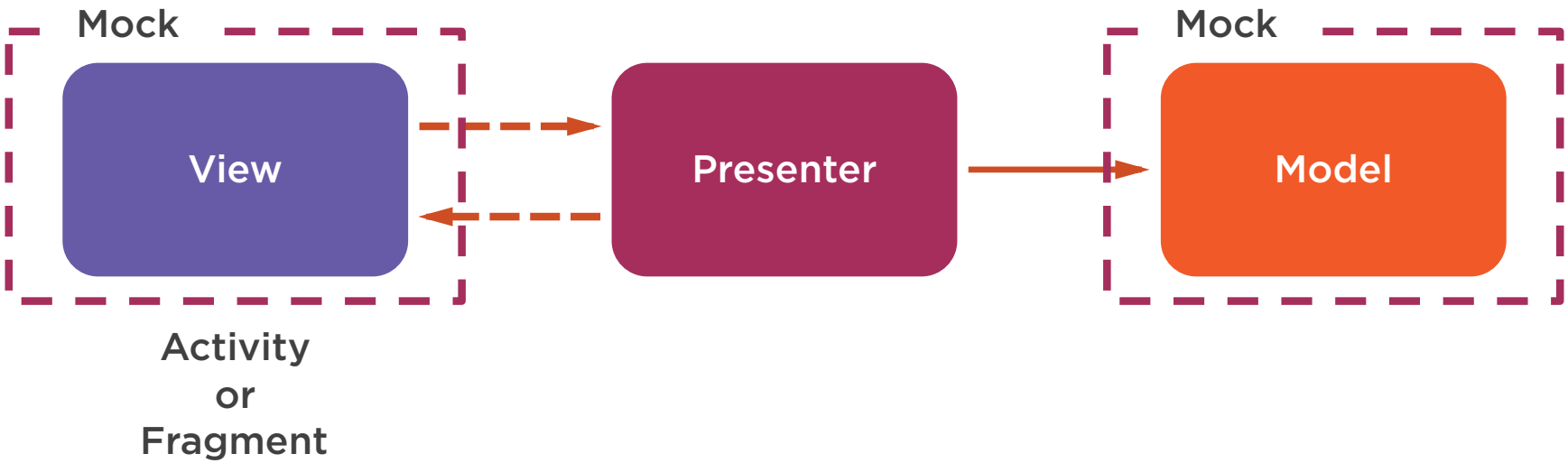
MVP – Model View Presenter



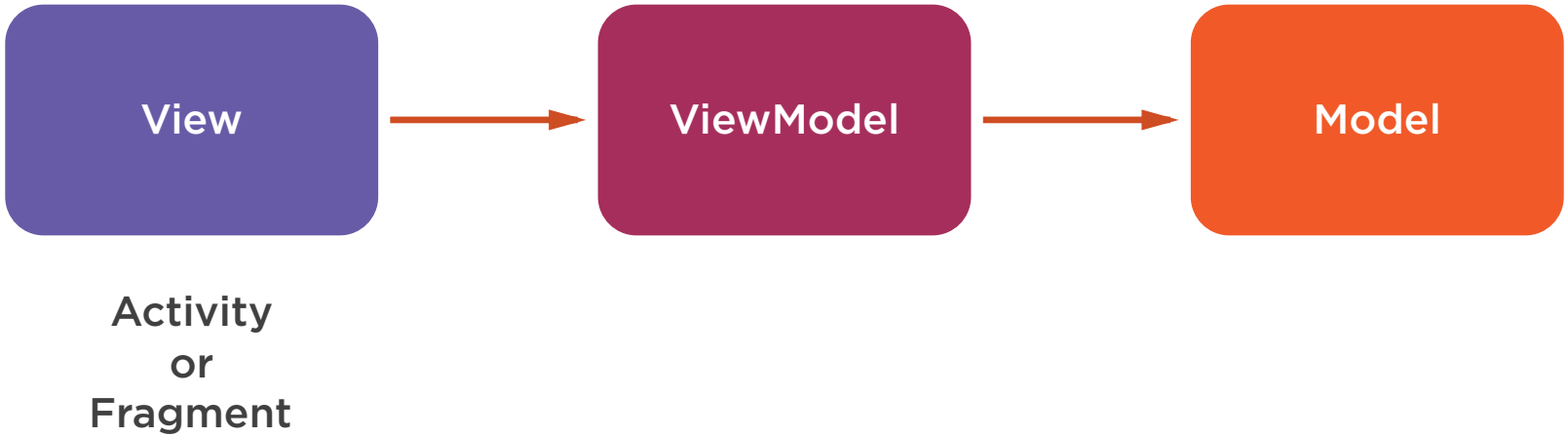
MVP – Model View Presenter



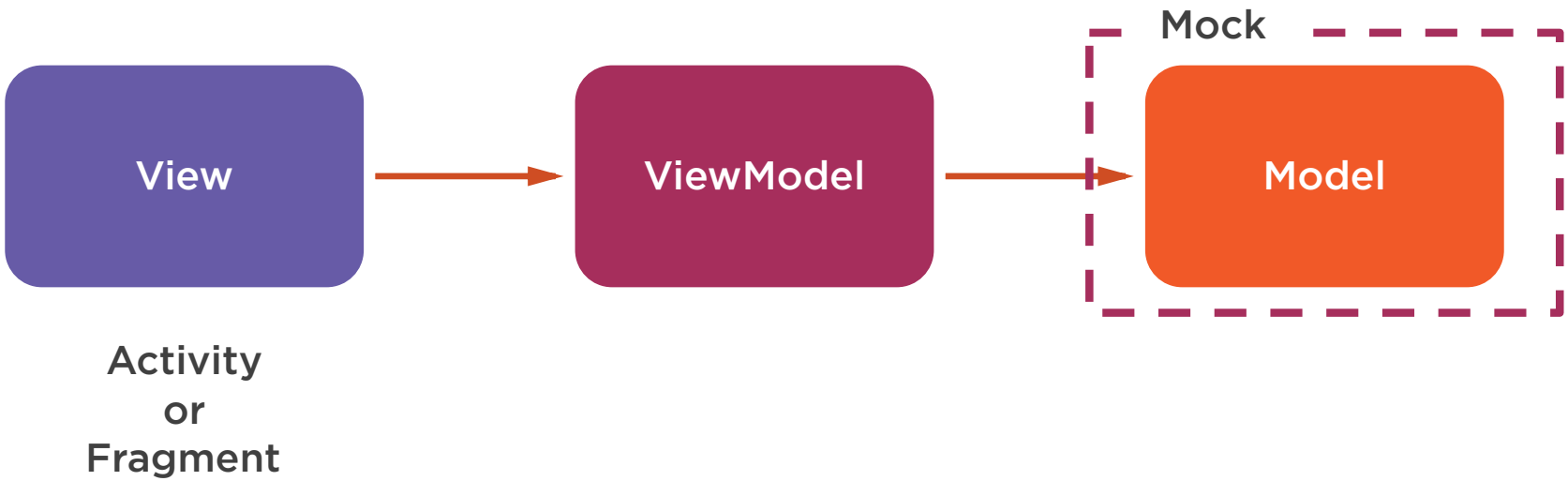
MVP – Model View Presenter



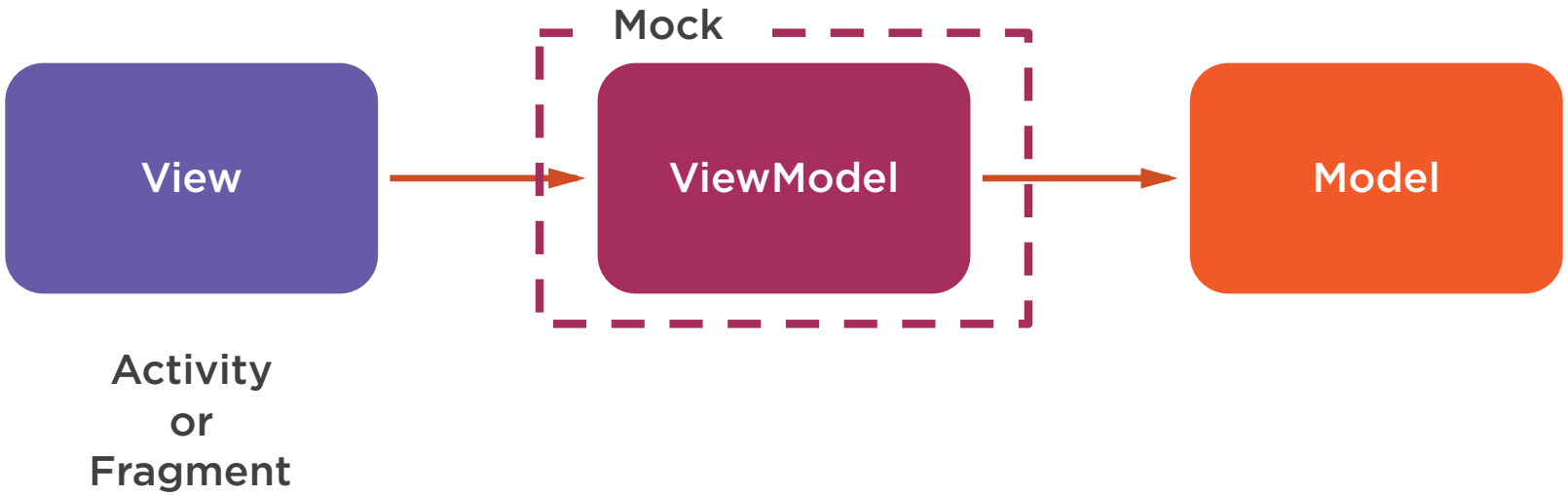
MVVM – Model View ViewModel



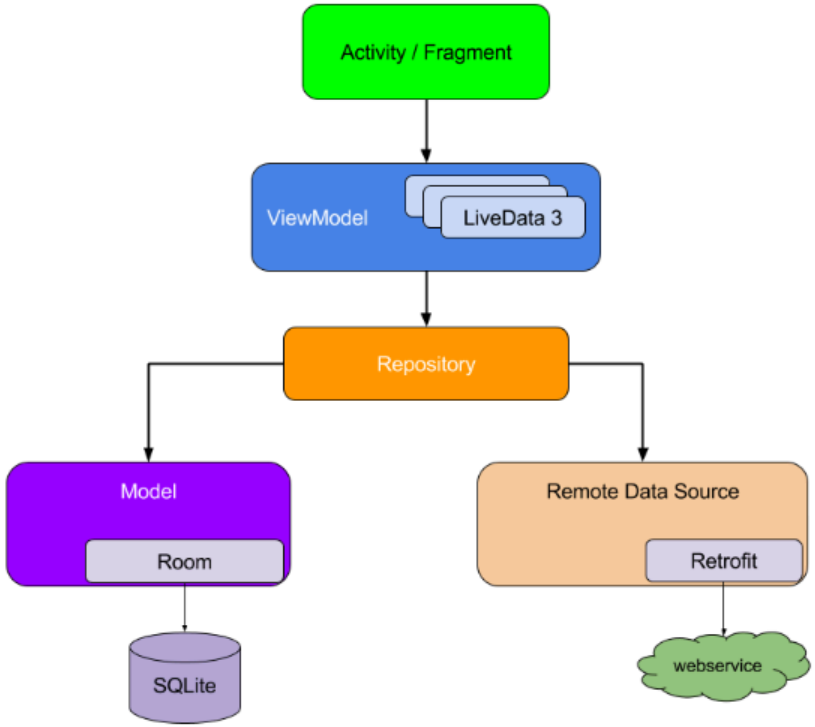
MVVM – Model View ViewModel



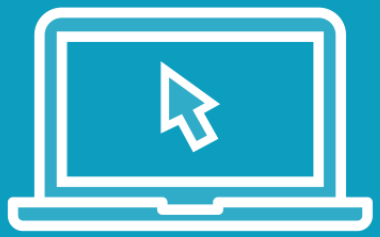
MVVM – Model View ViewModel



Guide to App Architecture by Google



Demo



Banking App Screens

Filtering Transactions

Changing Transaction Status

Viewing Cached Transactions

Summary

- Goals
- Importance of software architecture
- Testability
- MV* architectural patterns
- Banking app demo