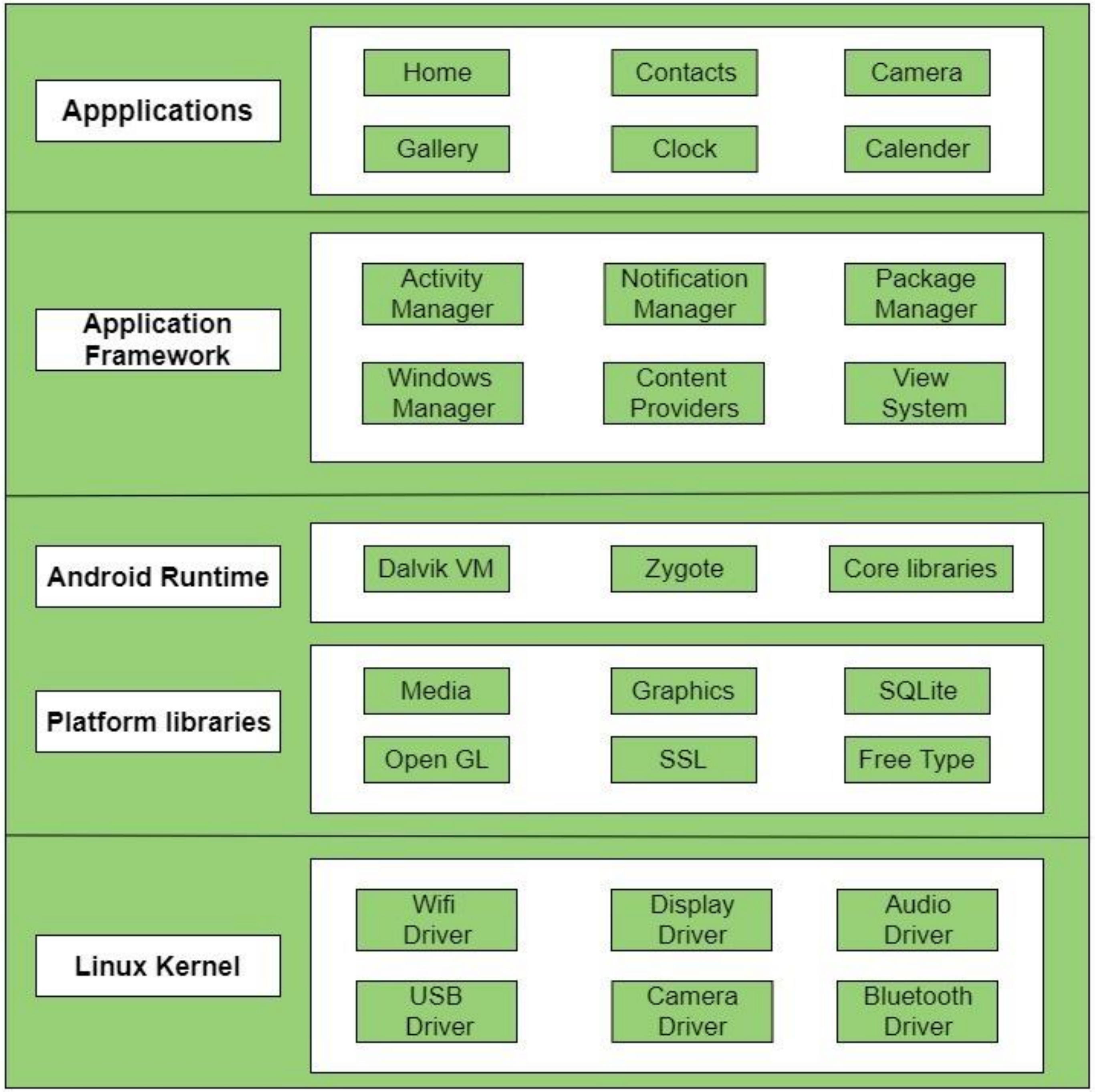
Android Architecture


Appplications
Home
Contacts
Camera
Gallery
Clock
Calender
Application Framework
Activity Manager
Notification Manager
Package Manager
Windows Manager
Content Providers
View System
Android Runtime
Dalvik VM
Zygote
Core libraries
Platform libraries
Media
Graphics
SQLite
Open GL
SSL
Free Type
Linux Kernel
Wifi Driver
Display Driver
Audio Driver
USB Driver
Camera Driver
Bluetooth Driver

Introduction

- Android architecture contains different number of components to support any android device needs.
- Android software contains an open-source Linux Kernel having collection of number of C/C++ libraries which are exposed through an application framework services.
- Among all the components Linux Kernel provides main functionality of operating system functions to smartphones and Dalvik Virtual Machine (DVM) provide platform for running an android application.
- The main components are the following.
  - Applications
  - Application Framework
  - Android Runtime
  - Platform Libraries
  - Linux Kernel

Applications

- Applications is the top layer of android architecture.
- The pre-installed applications like home, contacts, camera, gallery etc and third party applications downloaded from the play store like chat applications, games etc. will be installed on this layer only.

- It runs within the Android run time with the help of the classes and services provided by the application framework.

Application framework

- Application Framework provides several important classes which are used to create an Android application.
- It provides a generic abstraction for hardware access and also helps in managing the user interface with application resources.
- Generally, it provides the services with the help of which we can create a particular class and make that class helpful for the Applications creation.
- It includes different types of services activity manager, notification manager, view system, package manager etc. which are helpful for the development of our application according to the prerequisite.

Application runtime

- Android Runtime environment is one of the most important part of Android.
- It contains components like core libraries and the Dalvik virtual machine(DVM).
- Mainly, it provides the base for the application framework and powers our application with the help of the core libraries.
- Like Java Virtual Machine (JVM), **Dalvik Virtual Machine (DVM)** is a register-based virtual machine and specially designed and optimized for android to ensure that a device can run multiple instances efficiently.
- It depends on the layer Linux kernel for threading and low-level memory management.
- The core libraries enable us to implement android applications using the standard JAVA or Kotlin programming languages.

Platform libraries

- The Platform Libraries includes various C/C++ core libraries and Java based libraries such as Media, Graphics, Surface Manager, OpenGL etc. to provide a support for android development.
- **Media** library provides support to play and record an audio and video formats.
- **Surface manager** responsible for managing access to the display subsystem.
- **SGL** and **OpenGL** both cross-language, cross-platform application program interface (API) are used for 2D and 3D computer graphics.
- **SQLite** provides database support and **FreeType** provides font support.
- **Web-Kit** This open source web browser engine provides all the functionality to display web content and to simplify page loading.
- **SSL (Secure Sockets Layer)** is security technology to establish an encrypted link between a web server and a web browser.

Linux Kernel

- Linux Kernel is heart of the android architecture.
- It manages all the available drivers such as display drivers, camera drivers, Bluetooth drivers, audio drivers, memory drivers, etc. which are required during the runtime.
- The Linux Kernel will provide an abstraction layer between the device hardware and the other components of android architecture.
- It is responsible for management of memory, power, devices etc.

- The features of Linux kernel are:
- **Security:** The Linux kernel handles the security between the application and the system.
- **Memory Management:** It efficiently handles the memory management thereby providing the freedom to develop our apps.
- **Process Management:** It manages the process well, allocates resources to processes whenever they need them.
- **Network Stack:** It effectively handles the network communication.
- **Driver Model:** It ensures that the application works properly on the device and hardware manufacturers responsible for building their drivers into the Linux build.

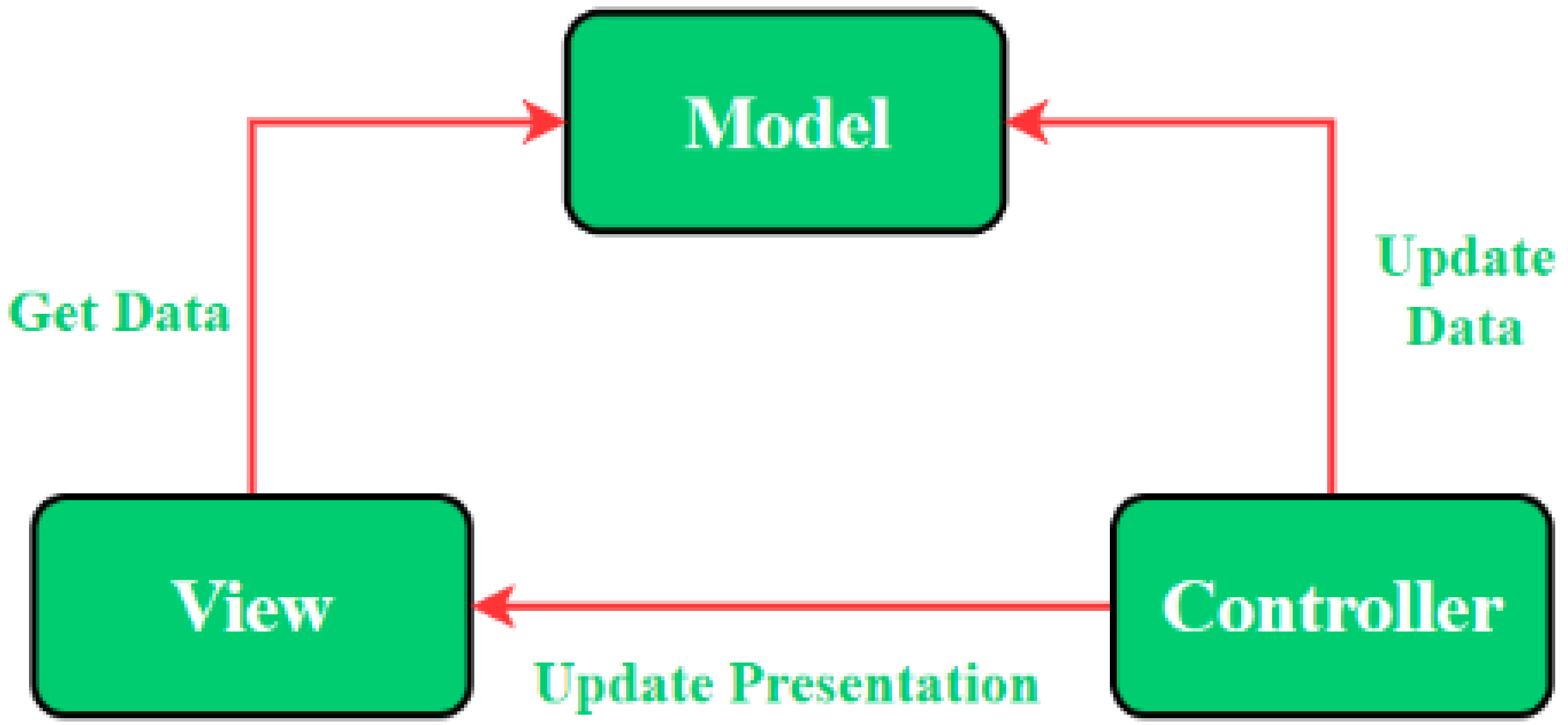
Difference Between MVC, MVP and MVVM Architecture Pattern in Android

Introduction

- Developing an android application by applying a software architecture pattern is always preferred by the developers.
- An architecture pattern gives modularity to the project files and assures that all the codes get covered in Unit testing.
- It makes the task easy for developers to maintain the software and to expand the features of the application in the future.
- MVC (Model — View — Controller)
- MVP (Model — View — Presenter) and
- MVVM (Model — View — ViewModel)
- is the most popular and industry-recognized android architecture pattern among developers.

MVC (Model – View – Controller)

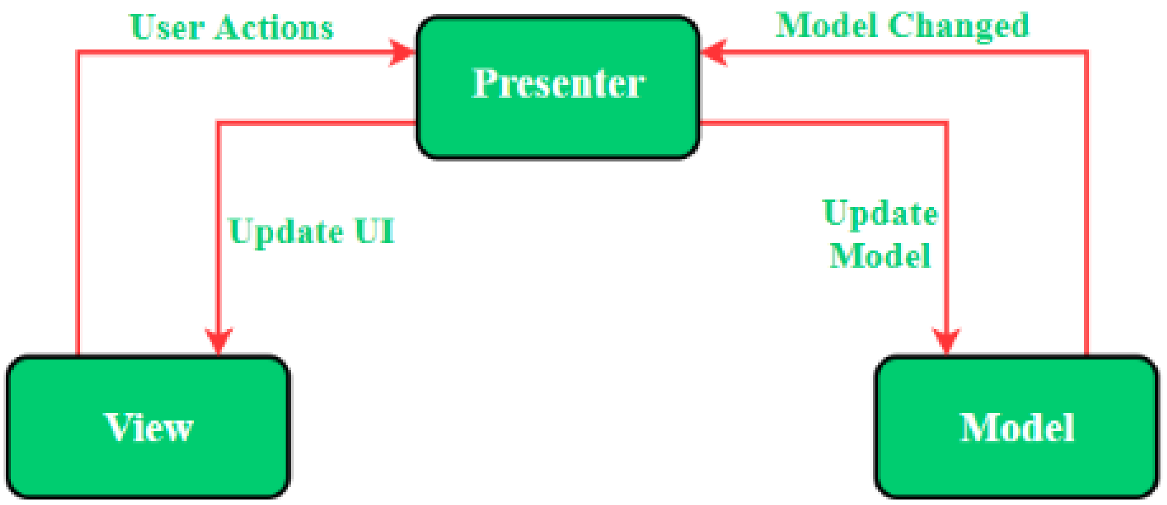
- The MVC pattern suggests splitting the code into 3 components.
- While creating the class/file of the application, the developer must categorize it into one of the following three layers:
- **Model**
  - This component stores the application data. It has no knowledge about the interface. The model is responsible for handling the domain logic(real-world business rules) and communication with the database and network layers.
- **View**
  - It is the UI(User Interface) layer that holds components that are visible on the screen. Moreover, it provides the visualization of the data stored in the Model and offers interaction to the user.
- **Controller**
  - This component establishes the relationship between the View and the Model. It contains the core application logic and gets informed of the user’s response and updates the Model as per the need.


Model
Get Data
Update Data
View
Controller
Update Presentation

The Model—View—Presenter(MVP) Pattern

- MVP pattern overcomes the challenges of MVC and provides an easy way to structure the project codes.
- The reason why MVP is widely accepted is that it provides modularity, testability, and a more clean and maintainable codebase.
- It is composed of the following three components:
- Model
- View
- Presenter

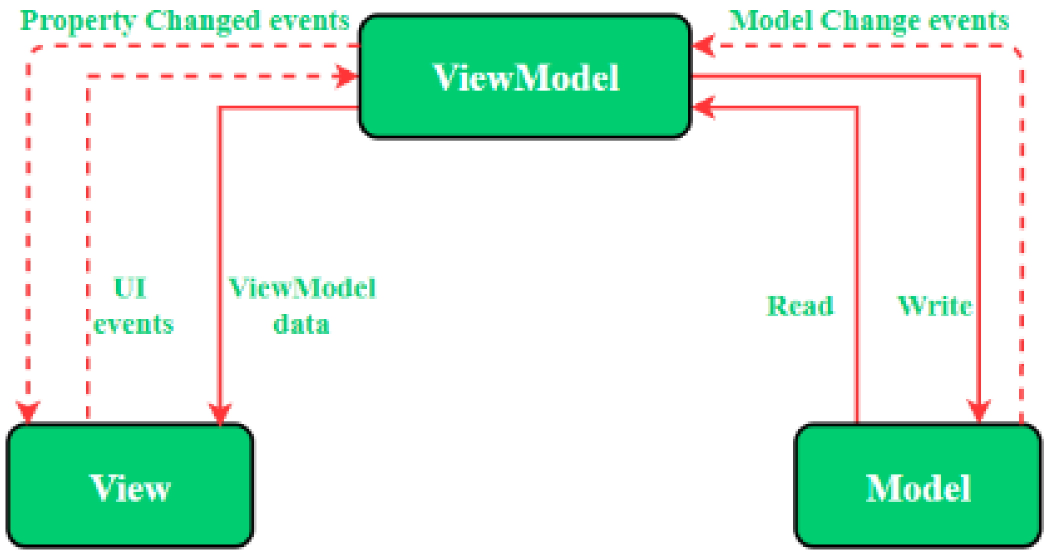
- **Model**
  - Layer for storing data. It is responsible for handling the domain logic(real-world business rules) and communication with the database and network layers.
- **View**
  - UI(User Interface) layer. It provides the visualization of the data and keep a track of the user's action in order to notify the Presenter.
- **Presenter**
  - Fetch the data from the model and applies the UI logic to decide what to display. It manages the state of the View and takes actions according to the user's input notification from the View.



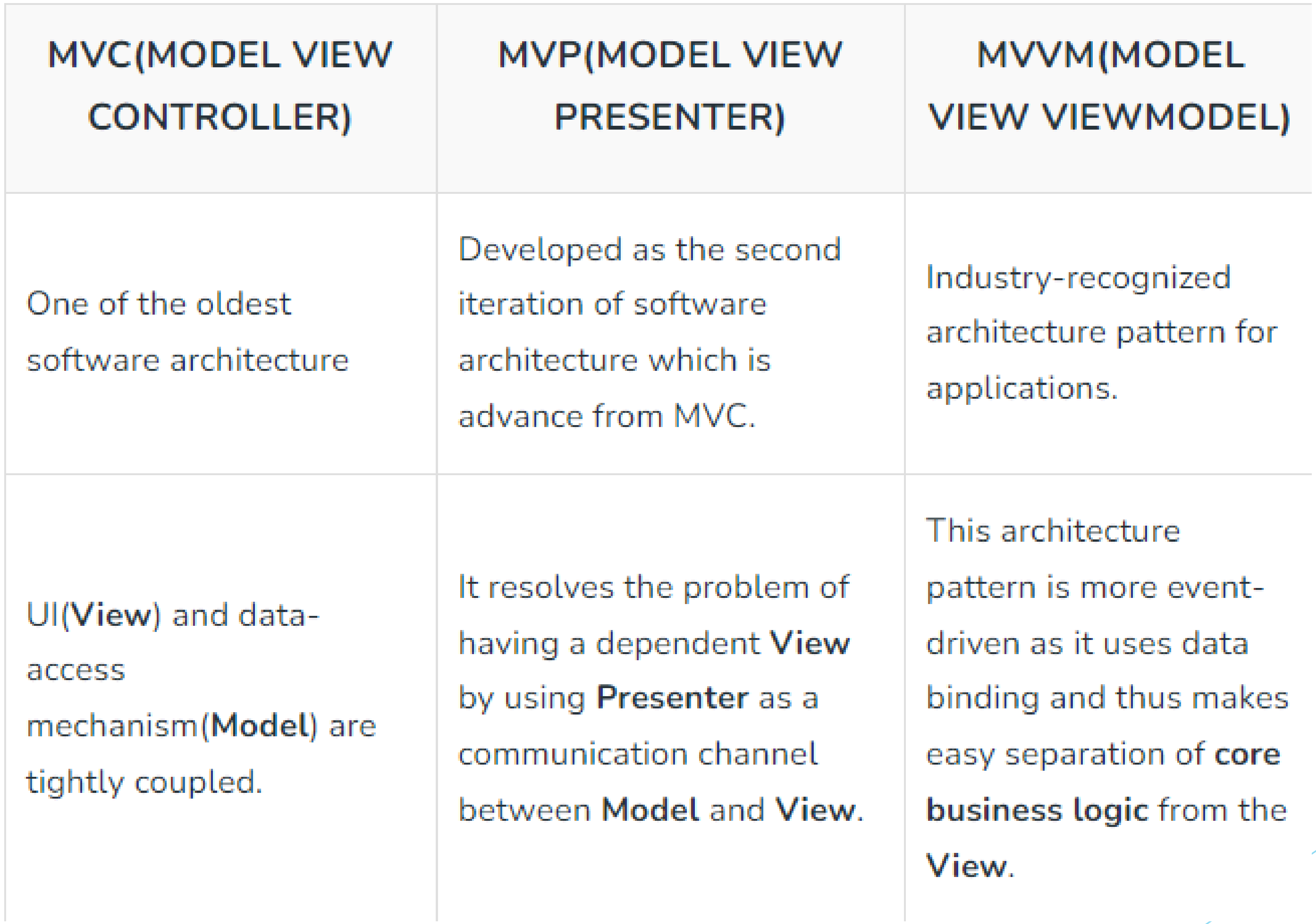
The Model — View — ViewModel (MVVM) Pattern

- MVVM pattern has some similarities with the MVP(Model — View — Presenter) design pattern as the Presenter role is played by the ViewModel.
- However, the drawbacks of the MVP pattern has been solved by MVVM.
- It suggests separating the data presentation logic(Views or UI) from the core business logic part of the application.
- The separate code layers of MVVM are:
- Model
- View
- ViewModel

- **Model**
  - This layer is responsible for the abstraction of the data sources. Model and ViewModel work together to get and save the data.
- **View**
  - The purpose of this layer is to inform the ViewModel about the user’s action. This layer observes the ViewModel and does not contain any kind of application logic.
- **ViewModel**
  - It exposes those data streams which are relevant to the View. Moreover, it servers as a link between the Model and the View.



Difference Between MVC, MVP, and MVVM Design Pattern



MVC(MODEL VIEW CONTROLLER)	MVP(MODEL VIEW PRESENTER)	MVVM(MODEL VIEW VIEWMODEL)
One of the oldest software architecture	Developed as the second iteration of software architecture which is advance from MVC.	Industry-recognized architecture pattern for applications.
UI(**View**) and data-access mechanism(**Model**) are tightly coupled.	It resolves the problem of having a dependent **View** by using **Presenter** as a communication channel between **Model** and **View**.	This architecture pattern is more event-driven as it uses data binding and thus makes easy separation of **core business logic** from the **View**.

Controller and **View** exist with the one-to-many relationship. One Controller can select a different View based upon required operation.	The one-to-one relationship exists between **Presenter** and **View** as one Presenter class manages one View at a time.	Multiple **View** can be mapped with a single **ViewModel** and thus, the one-to-many relationship exists between View and ViewModel.
The **View** has no knowledge about the **Controller**.	The **View** has references to the **Presenter**.	The **View** has references to the **ViewModel**
Difficult to make changes and modify the app features as the code layers are tightly coupled.	Code layers are loosely coupled and thus it is easy to carry out modifications/changes in the application code.	Easy to make changes in the application. However, if **data binding logic** is too complex, it will be a little harder to debug the application.

User Inputs are handled by the **Controller**.	The **View** is the entry point to the Application	The **View** takes the input from the user and acts as the entry point of the application.
Ideal for small scale projects only.	Ideal for simple and complex applications.	Not ideal for small scale projects.
Limited support to **Unit testing**.	Easy to carry out **Unit testing** but a tight bond of View and Presenter can make it slightly difficult.	**Unit testability** is highest in this architecture.
This architecture has a high dependency on Android APIs.	It has a low dependency on the Android APIs.	Has low or no dependency on the Android APIs.
It does not follow the modular and single responsibility principle.	Follows modular and single responsibility principle.	Follows modular and single responsibility principle.

- The key difference between MVP and MVC is that the Presenter in MVP has a more active role in the communication between the Model and the View, and is responsible for controlling the flow of data between the two. MVVM stands for Model-View-ViewModel.
- In the MVVM pattern, the “Model” represents the data and logic of the application, the “View” represents the UI of the application, and the “ViewModel” is a layer that sits between the Model and the View and is responsible for exposing the data and logic of the Model to the View in a way that is easier to work with.
- The ViewModel also handles user input and updates the Model as needed.
- Overall, the main difference between these patterns is the role of the mediator component.
- MVC and MVP both involve a Controller or Presenter that acts as a mediator between the Model and the View, while MVVM involves a ViewModel that serves as the mediator between the Model and the View.
- MVC is the simplest of these patterns, while MVP and MVVM are more flexible and allow for a cleaner separation of concerns between the different layers of the application.