Architectural Style & Patterns

Lecture-4

Architectural Style Patterns

- Client-Server Architecture
- Model-View-Controller (MVC) Pattern
- Layered Architecture
- Microservices Architecture
- Event-Driven Architecture
- Realtime System Architecture
- Clean Architecture

Client - Server Architecture – Introduction

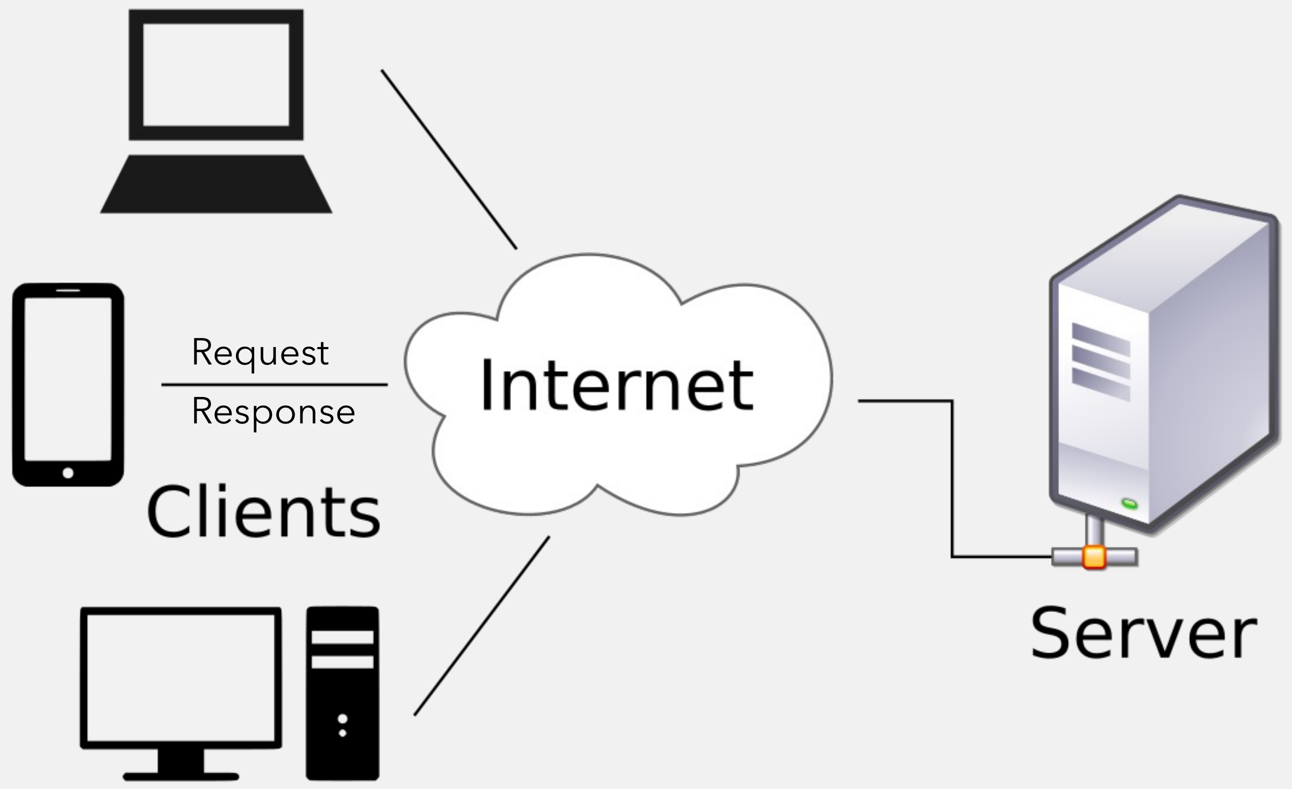
A network architecture in which each computer or process on the network is either a client or a server

OR

Server software accepts requests for data from client software and returns the results to the client

Client - Server Architecture – Components

- Client
- Server
- Communication Networks



Client/Server Applications

- The application is modeled as a set of services that are provided by servers and a set of clients that use these services.
- The client must know about servers but the servers don’t need to know their clients
- Client and servers are logical processes

Application Layers

- **Presentation Layer**

  concerned with presenting the results of a computation to system users and with collecting user inputs

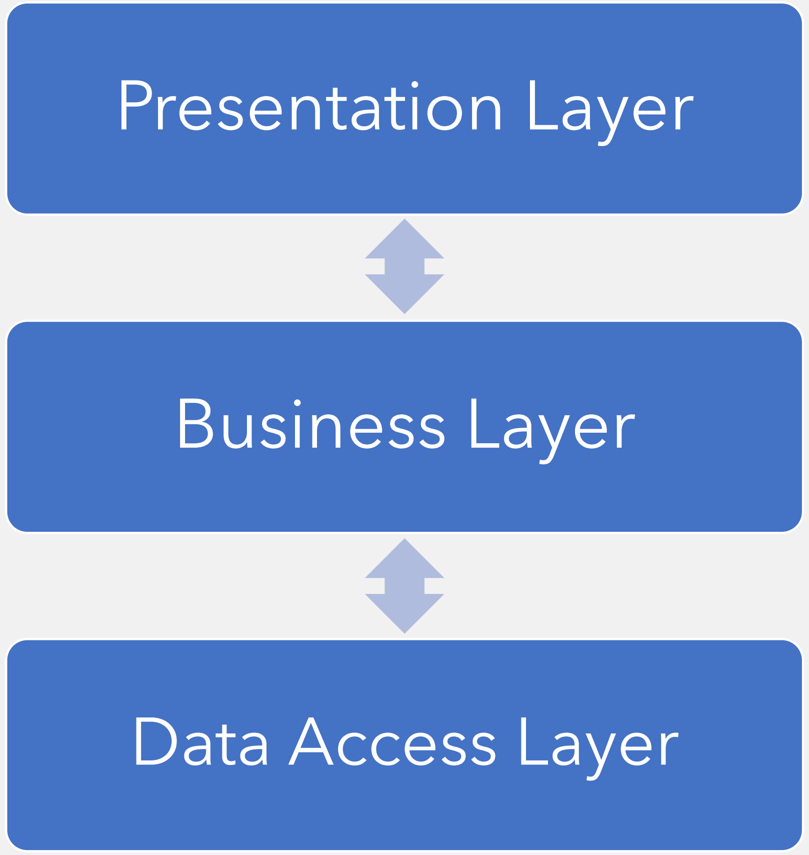
- **Application Processing L ayer / Business Layer**

  concerned with providing application-specific functionality

- **Data Access Layer**

  concerned with managing the system databases

Application Layers



Types of Client - Server Architectures

- **Two Tier**

  The three application layers are mapped onto two computer systems – the client and the server

  The client can be **“thin”** or **“fat”**

- **Three Tier**

  The three application layers are mapped onto three logically separate processes that execute on different processors.

Thin and Fat Clients

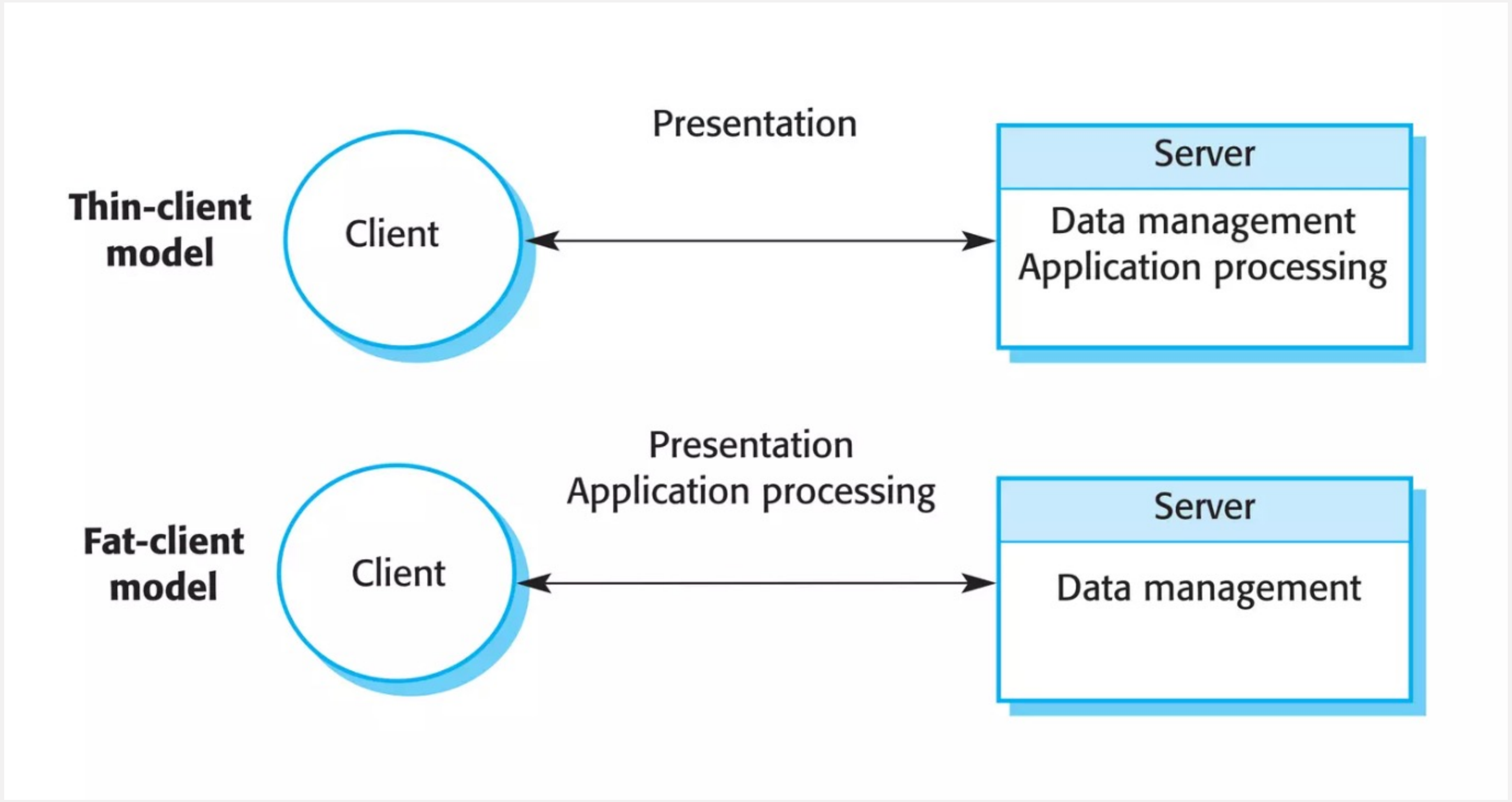
- **Thin Client Model**

  In a thin client model, all of the application processing and data management is carried out on the server. The client is simply responsible for running the presentation software.

- **Fat Client Model**

  In this model, the server is only responsible for data management. The software on the client implements the application logic and the interactions with the system users.

Thin and Fat Clients



Thin Client Model

- Usually used when legacy system migrated to client-server architecture
  - *The legacy system will act as a server in its own right with a graphical interface implemented on a client*

- A major disadvantage is that it places a heavy processing load on both the server and the network.

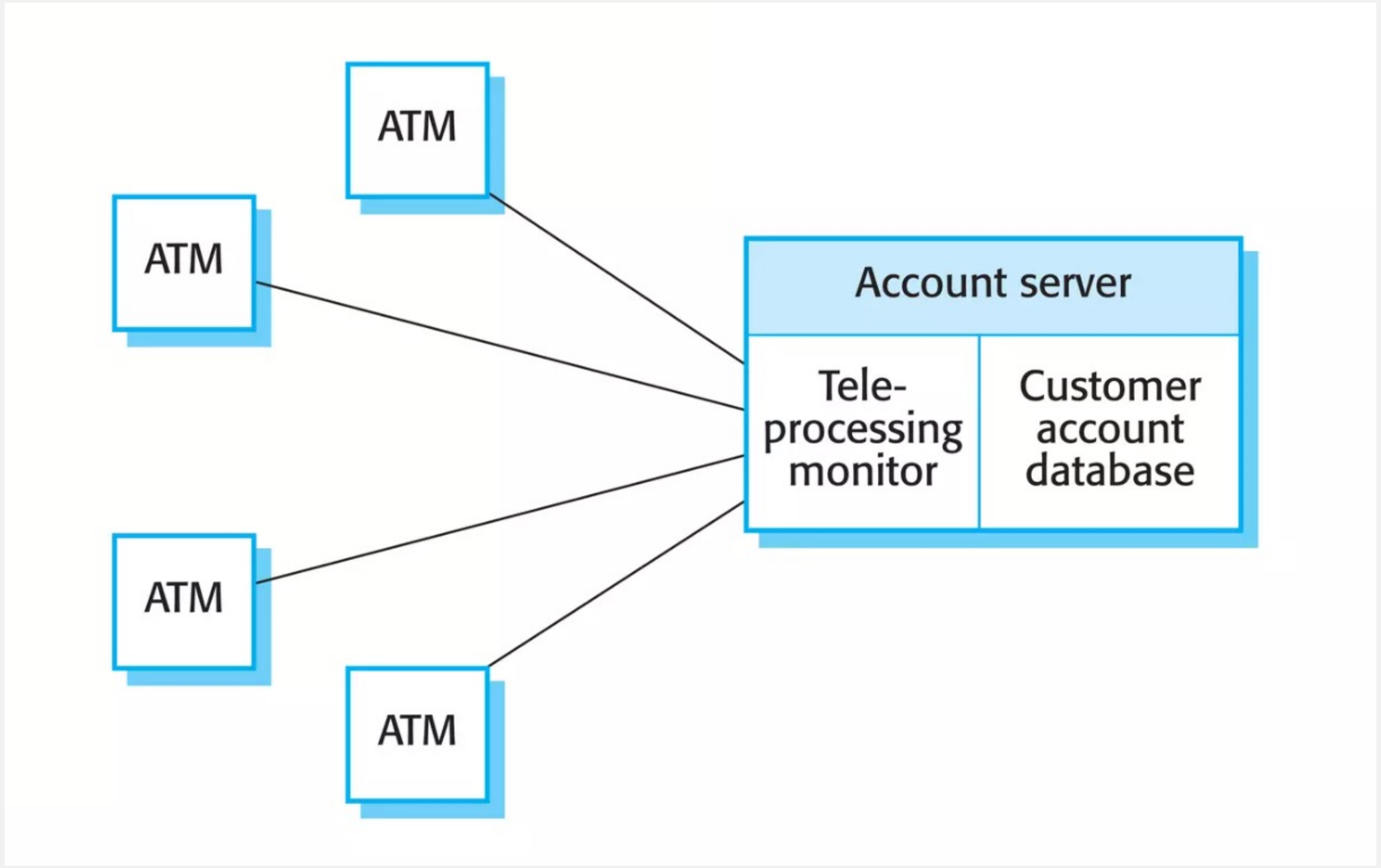
Fat Client Model

- More processing is delegated to the client as the application processing is locally executed

- More suitable for the new C/S systems where the capabilities of the client system are known in advance.

- More complex than a thin client model especially for management. The new version of the application has to be installed on all clients

Two Tier Architecture



Advantages of 2-Tier Architecture

- **Simplicity**: Two-tier architecture is relatively simple to implement and manage.
- **Cost-Effective**: It requires fewer hardware resources, making it cost-effective for smaller applications.
- **Faster Performance**: With fewer layers, data retrieval can be faster.

Limitation of 2-Tier Architecture

- **Scalability Issues**: It can be challenging to scale as all the processing is done on the server.
- **Limited Security**: Security measures are limited to the server-client communication.

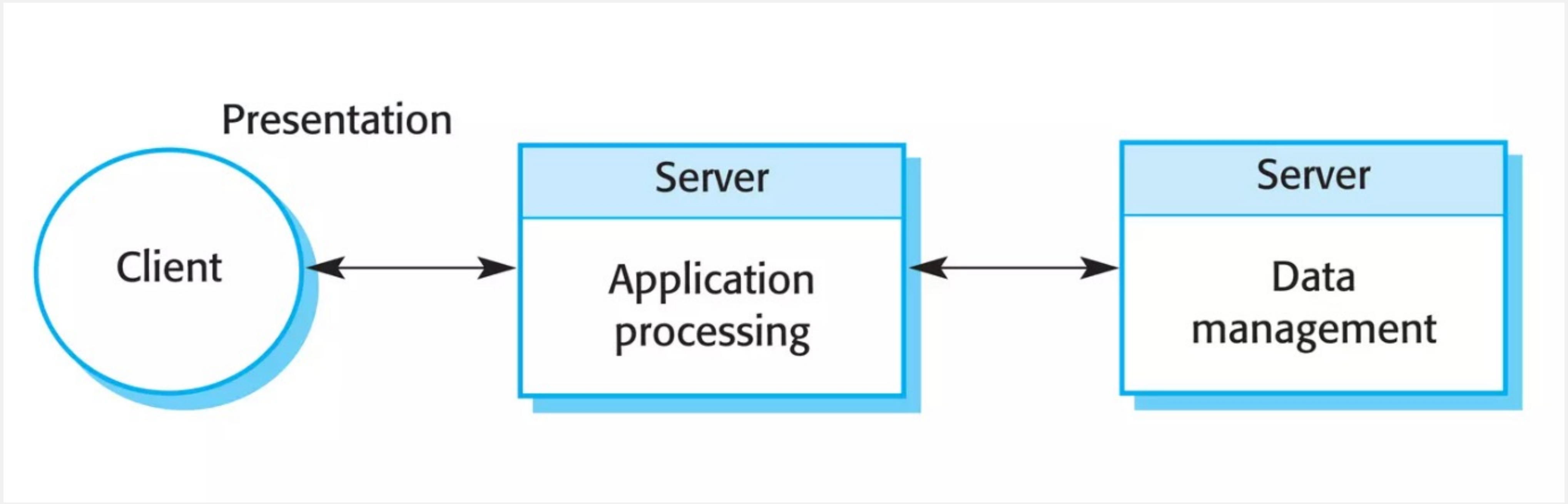
Three Tier Architecture

- Each of the application layer may executes on a separate processor

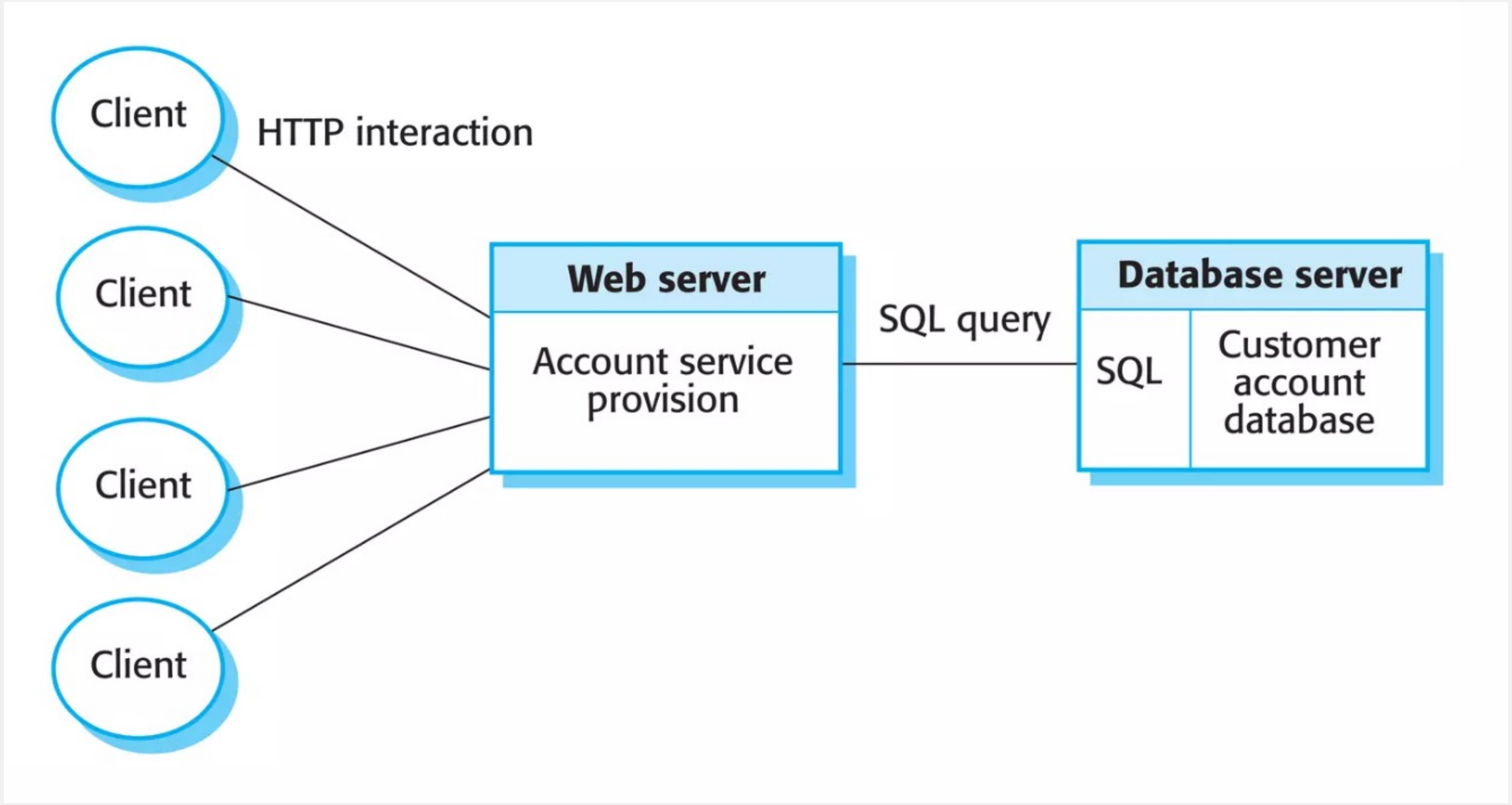
- Allows for better performance than a thin client model and is simpler to manage than a fat client model

- A more scalable architecture - as demand increases, extra servers can be added

Three Tier Architecture



Three Tier Architecture



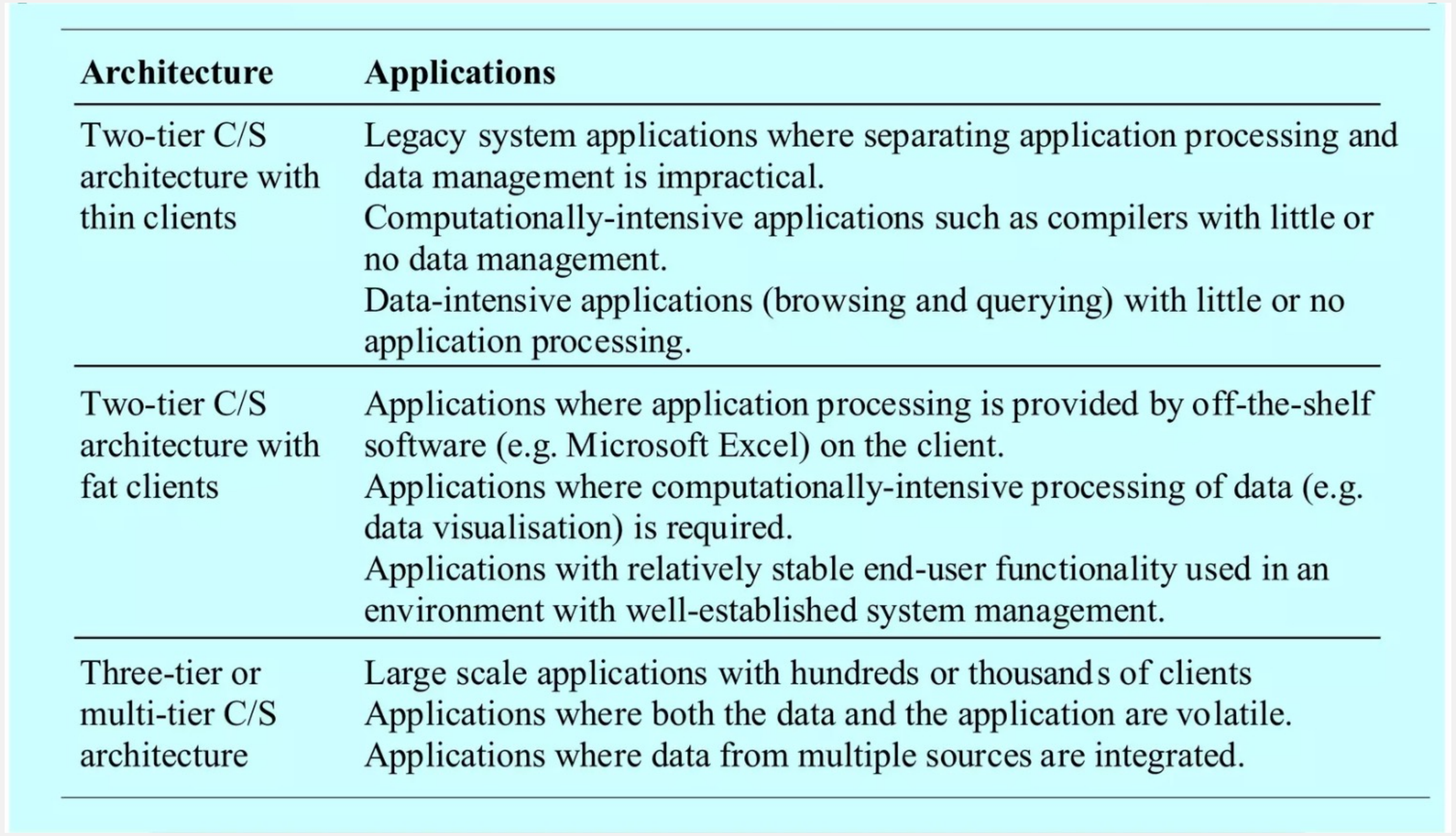
Advantages of 3-Tier Architecture

- **Scalability**: It offers better scalability as the application server can be scaled independently.
- **Enhanced Security**: Data security is improved with the additional layer.
- **Improved Maintenance**: Changes in the application logic can be made without affecting the database layer.

Limitation of 3-Tier Architecture

- **Complexity**: It is more complex to implement and maintain.
- **Higher Costs**: The additional layer (application server) can increase infrastructure costs.

Application of Tiered Architecture



Architecture	Applications
Two-tier C/S architecture with thin clients	Legacy system applications where separating application processing and data management is impractical. Computationally-intensive applications such as compilers with little or no data management. Data-intensive applications (browsing and querying) with little or no application processing.
Two-tier C/S architecture with fat clients	Applications where application processing is provided by off-the-shelf software (e.g. Microsoft Excel) on the client. Applications where computationally-intensive processing of data (e.g. data visualisation) is required. Applications with relatively stable end-user functionality used in an environment with well-established system management.
Three-tier or multi-tier C/S architecture	Large scale applications with hundreds or thousands of clients Applications where both the data and the application are volatile. Applications where data from multiple sources are integrated.

Comparison b/w Two & Three Tier Architecture

1. **Scalability**
   - **Two-Tier**: Limited scalability due to centralized processing.
   - **Three-Tier**: Better scalability with the application server layer.
2. **Maintenance and Upgrades**
   - **Two-Tier: Simpler maintenance but can be disruptive for upgrades.**
   - **Three-Tier: Easier to maintain and upgrade without affecting the database layer.**
3. **Security**
   - **Two-Tier: Limited security measures between the client and server.**
   - **Three-Tier: Enhanced security with the added application server layer.**
4. **Performance**
   - **Two-Tier: Faster data retrieval but limited scalability.**
   - **Three-Tier: Slightly slower data retrieval but better overall performance and scalability.**

MVC Architecture

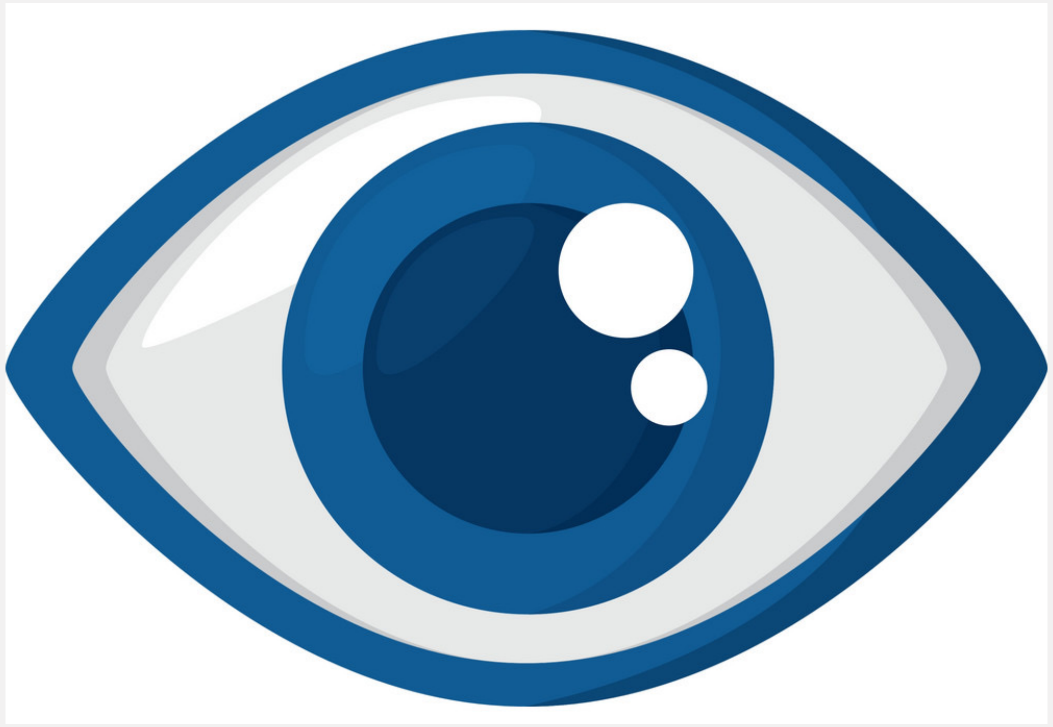
MVC is a pattern used to design user interfaces, data, and application logic to achieve separation of concerns.

The Model

It is the specific representation of the information with which the system operates. Logic ensures the integrity of data and allows to derive it.

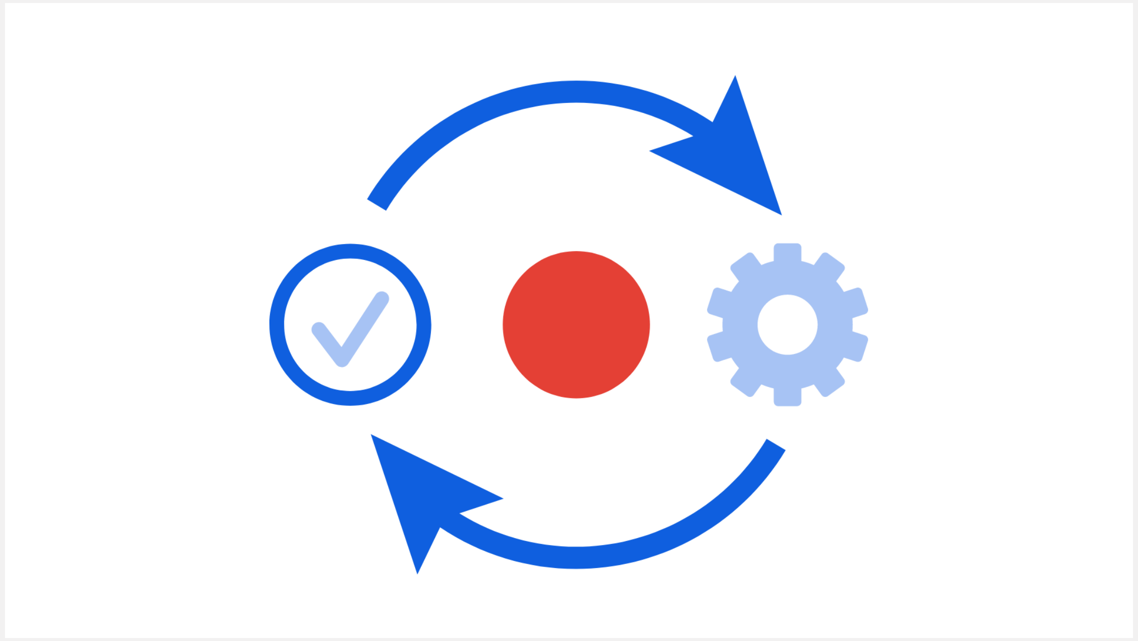
The View

Represent the model in a suitable format to interact and access the data, usually called "User Interface"

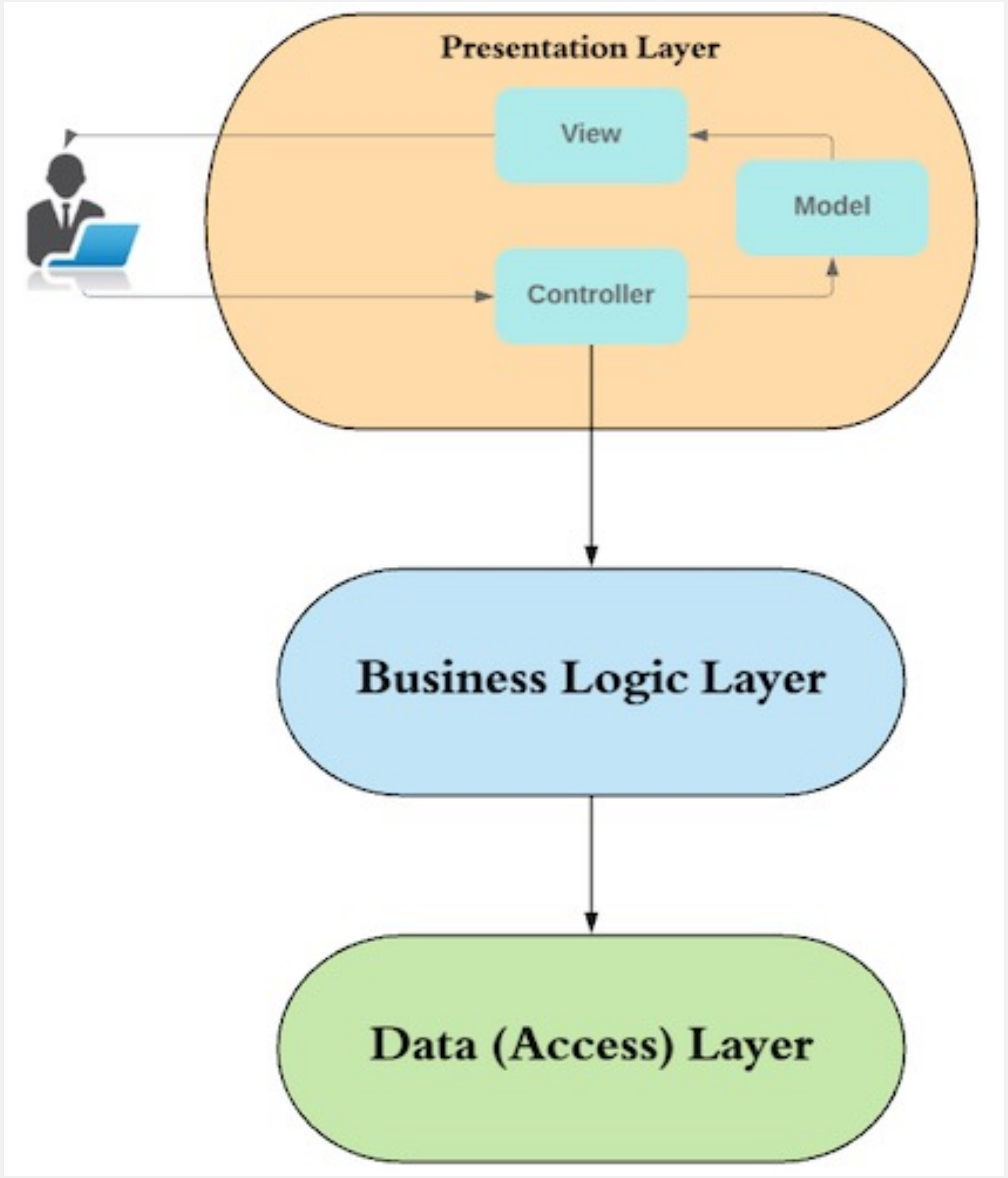


The Controller

It is the link between the view and the model, and is responsible for receiving and responding to events, typically user action and invoke changes on the model and probably on the view



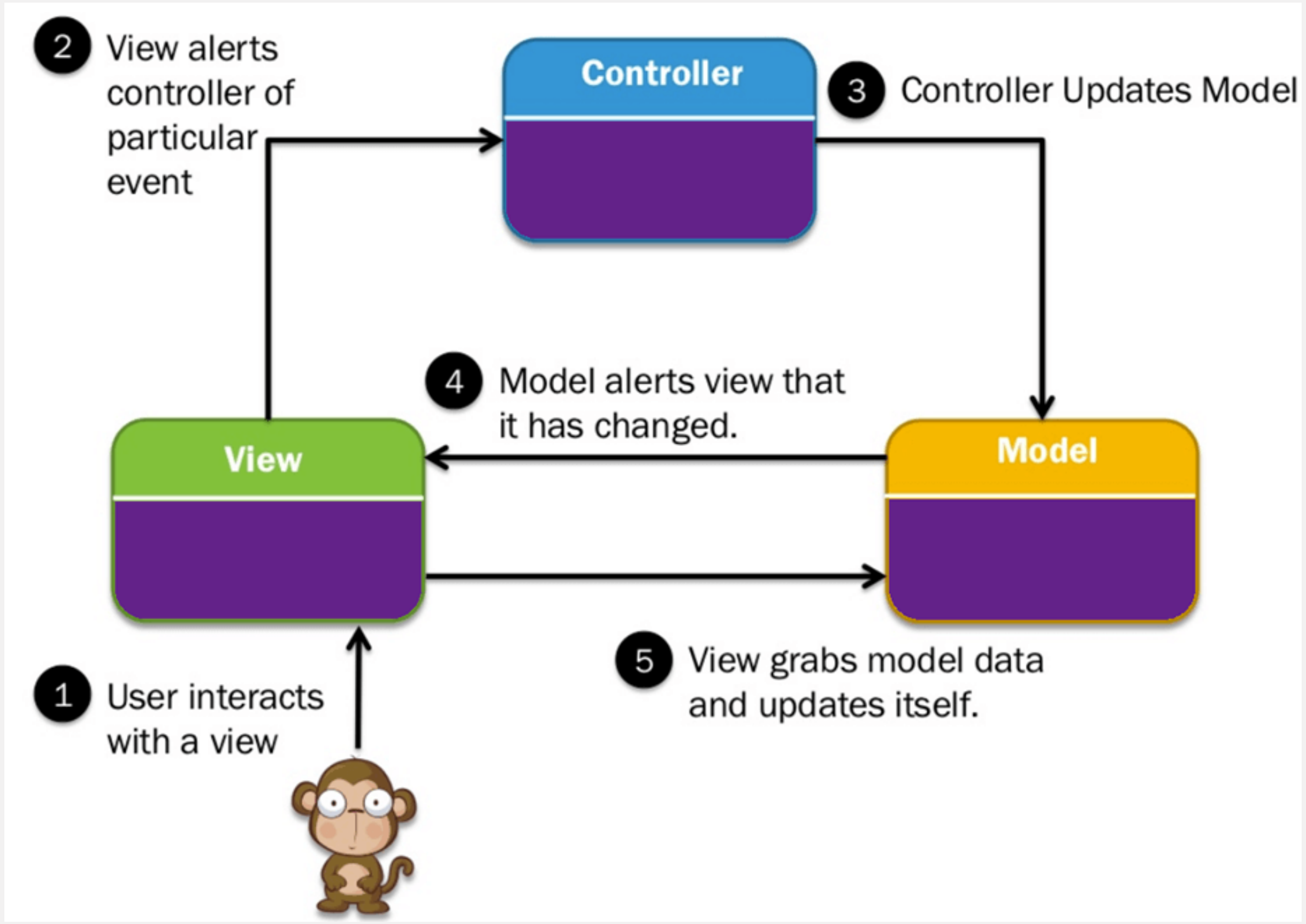
3 Tier Architecture vs MVC Pattern



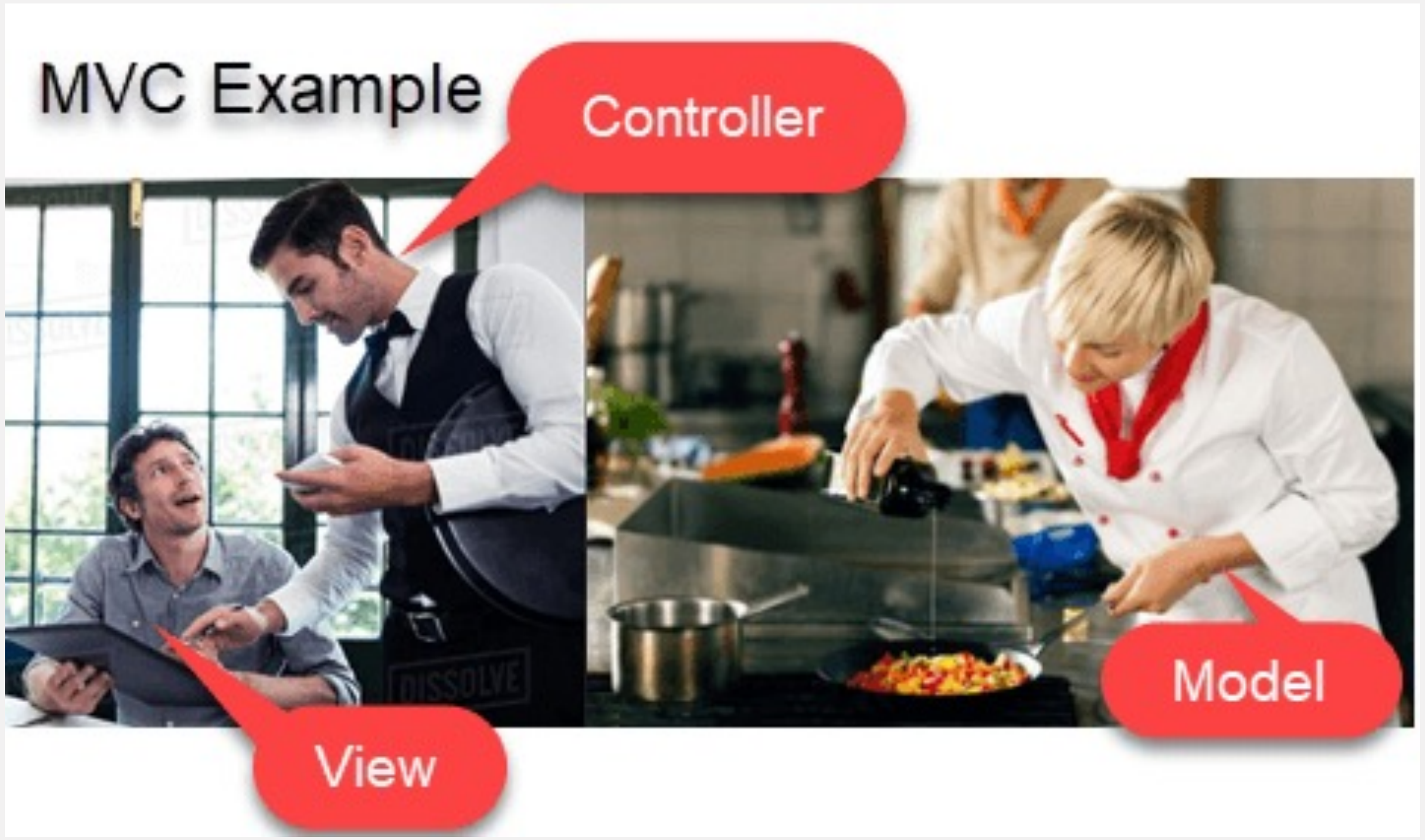
Feature of MVC

- Easy and frictionless testability. Highly testable, extensible and pluggable framework
- To design a web application architecture using the MVC pattern, it offers full control over your HTML as well as your URLs
- Leverage existing features provided by ASP.NET, JSP, Django, etc.
- Clear separation of logic: Model, View, Controller. Separation of application tasks viz. business logic, Ul logic, and input logic
- URL Routing for SEO Friendly URLs. Powerful URL- mapping for comprehensible and searchable URLs
- Supports for Test Driven Development (TDD)

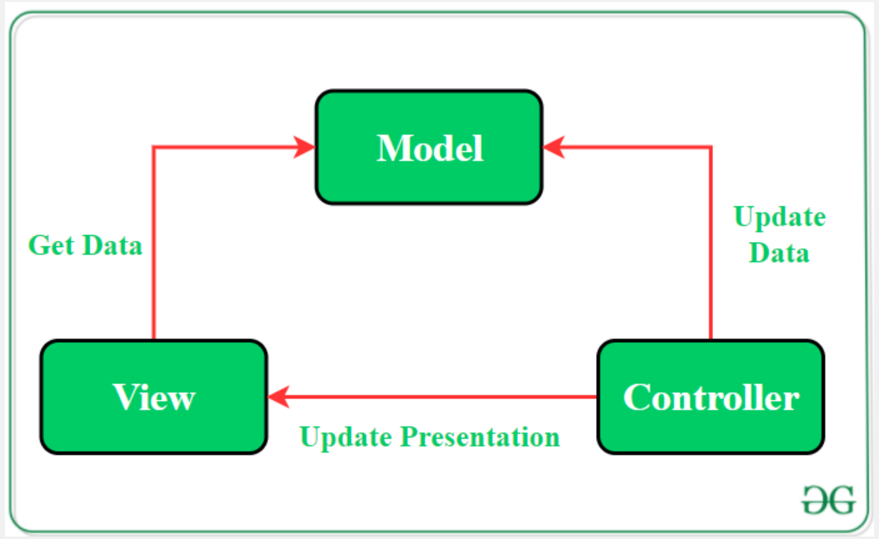
Detailed Architecture of MVC Architecture

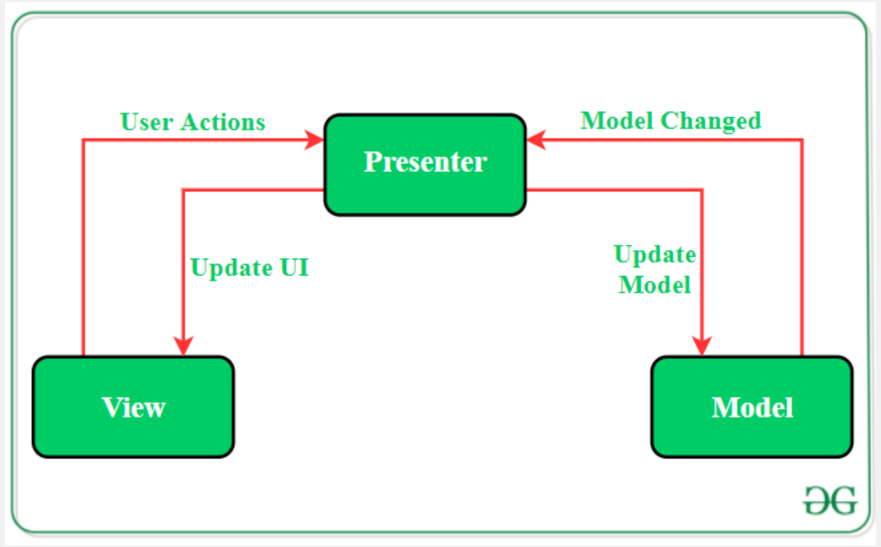

2 View alerts controller of particular event
Controller
3 Controller Updates Model
4 Model alerts view that it has changed.
View
Model
5 View grabs model data and updates itself.
1 User interacts with a view

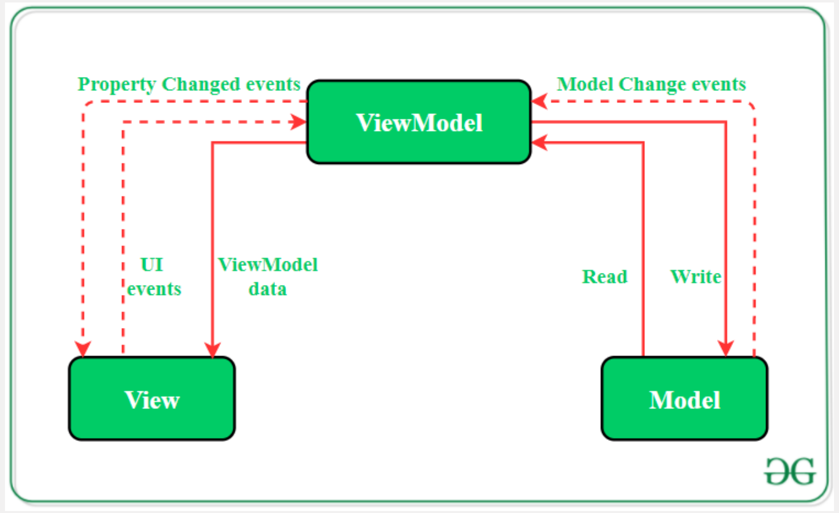
MVC Real Life Example



MVC vs MVP vs MVVM (https://www.linkedin.com/pulse/when-use-mvc-mvp-mvvm-nothing-ahmed-adel)







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	Follows modular and single responsibility	Follows modular and single responsibility

MVC Advantages & Disadvantages

- **Advantages of MVC architecture pattern**
- MVC pattern increases the code testability and makes it easier to implement new features as it highly supports the separation of concerns.
- Unit testing of Model and Controller is possible as they do not extend or use any Android class.
- Functionalities of the View can be checked through UI tests if the View respect the single responsibility principle(update controller and display data from the model without implementing domain logic)
- **Disadvantages of MVC architecture pattern**
- Code layers depend on each other even if MVC is applied correctly.
- No parameter to handle UI logic i.e., how to display the data.

MVP Advantages & Disadvantages

- **Advantages of MVP Architecture**
- No conceptual relationship in android components
- Easy code maintenance and testing as the application’s model, view, and presenter layer are separated.
- **Disadvantages of MVP Architecture**
- If the developer does not follow the single responsibility principle to break the code then the Presenter layer tends to expand to a huge all-knowing class.

MVVM Advantages & Disadvantages

- **Advantages of MVVM Architecture**
- Enhance the reusability of code.
- All modules are independent which improves the testability of each layer.
- Makes project files maintainable and easy to make changes.
- **Disadvantages of MVVM Architecture**
- This design pattern is not ideal for small projects.
- If the data binding logic is too complex, the application debug will be a little harder.