25

# Important Software Architecture & Design Patterns Interview Questions

WWW.QUESTPOND.COM

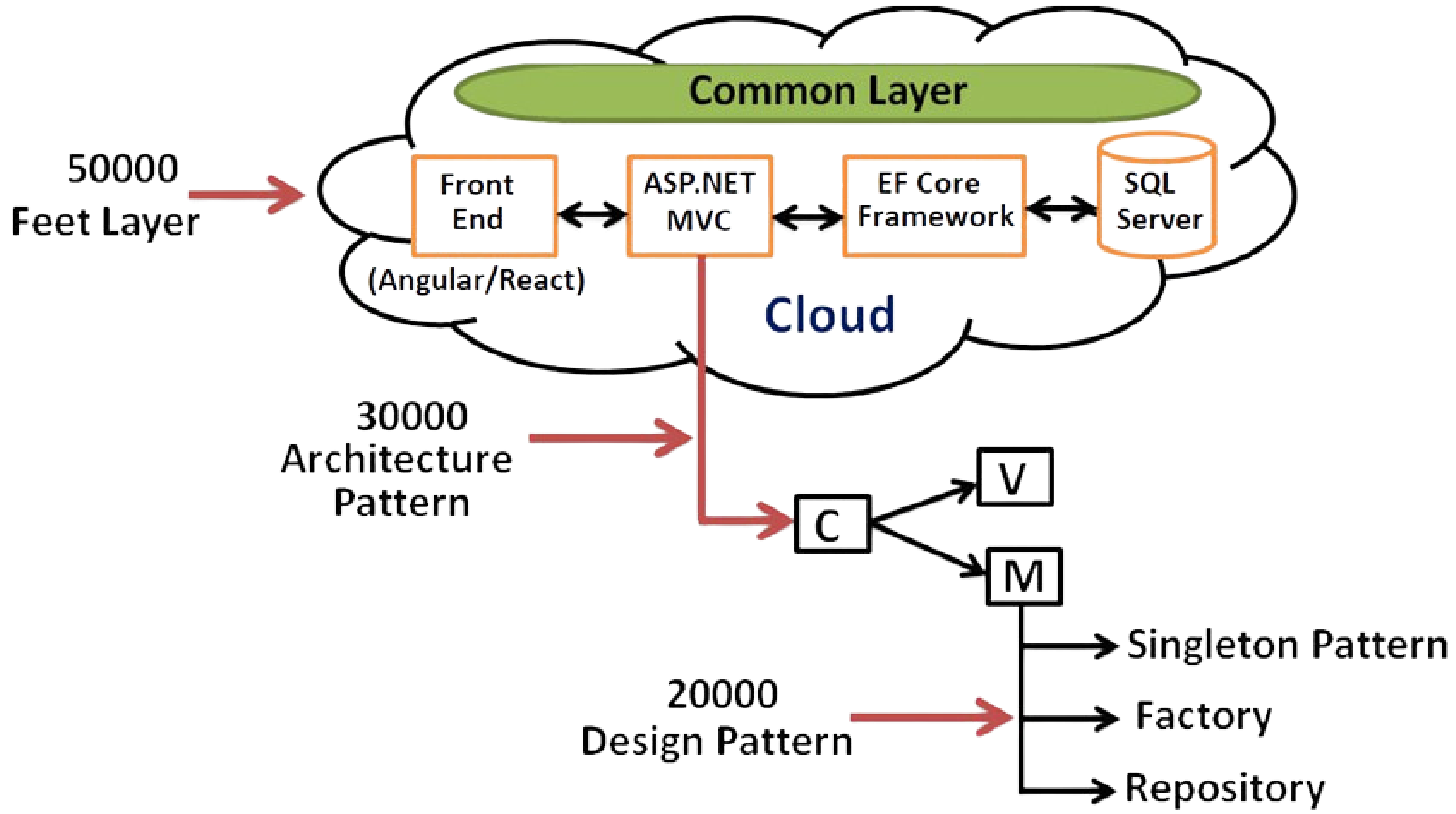
# Q.1 Explain your Project Architecture?

So, the way to answer this question is to start from 50000 level go to 30000 level and land to 10000 level.

**1. The Top Layers: -** It's a 4-layer architecture with front end in Angular / React, Middle Layer in ASP.NET MVC, Data layer is EF and Database is SQL Server.

**2. Architecture Patterns: -** Then ZOOM on the architecture patterns of the layers. So like the ASP.NET layer follows MVC architecture where in the model has business logic,view has the UI controls / look and feel and the controller connects the model with the view. If your front end is angular you can talk about MVC (Model view component) architecture.

**3. Design Patterns: -** At the last level talk about any design patterns you have used. For example, you can say data access layer uses repository /UOW, factory pattern for centralizing object creation and so on.



Below is how a typical answer in Interview should look like.

It's a 4-layer architecture with front end in Angular, Middle Layer in MVC, Data access layer uses EF and DB in SQL Server.

ASP.NET follows MVC architecture where model has business logic, view look and feel and controller connect the model and the view.

In projects we have implemented best practices and design patterns like repository / UOW / Factory, iterator pattern and so on.

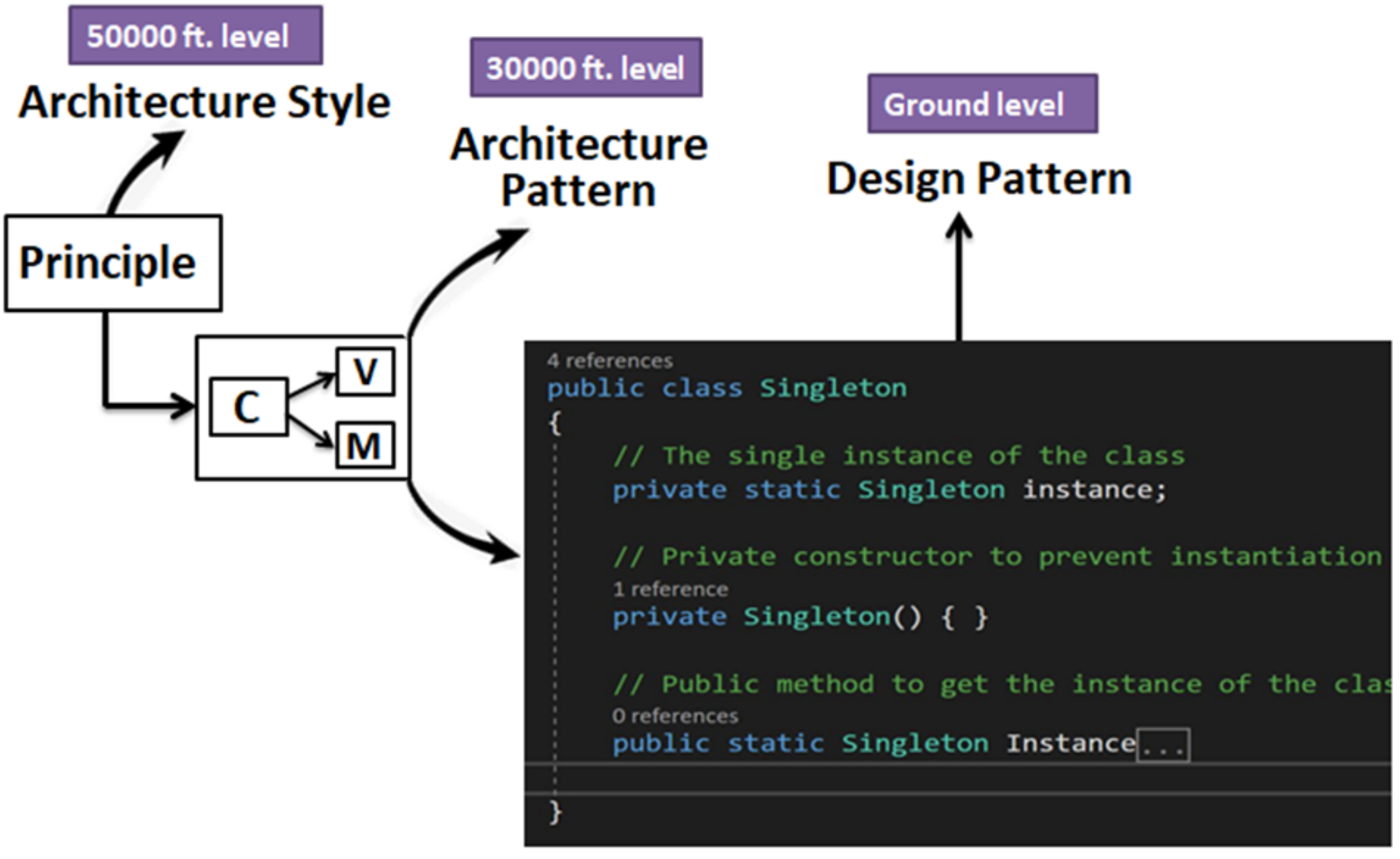
**Note: -** For this question do not answer ONE LINERS like MVC, 3 Layer. Try to give a detailed answer. One special note my architecture cannot be your architecture, so my answer can not be your answer. So, try to see your architecture and express accordingly.

# Q.2 Architecture Style VS Architecture Pattern VS Design pattern

So, for example, REST provides general principle on how to design networked application. Like its should be stateless, resource abstraction, manipulation of resource through representation and standard uniform interfaces.

Architecture pattern provides general structures / layers for designing software system. Some examples are MVC, Layered architecture, MVVM and so on.

Design pattern is solution for recurring problems in software design. And it has a very clear context. It operates at a very lower level, almost at the code level.




**Note :-** One of the important things during architect interview is to avoid vocabulary confusion. If you have vocabulary confusion the interviewer can lose confidence.

An architecture style represents a set of principles or guidelines that shape the overall structure and organization of a software system. Some examples are REST, SOA and so on.

# Q.3 What are Design Patterns?

Design patterns are time tested solution for recurring architecture problems.

Also, with the above definition try to give one or two design patterns as examples. While giving examples remember three things: -

**1.** Do not give example of Singleton pattern so that you can stand out in the crowd.

**2.** Give one example of non-GOF pattern like repository, CQRS and so on.

**3.** And whatever patterns you say in example make sure you are fully aware of the same.

Quest Pond

Below are some examples of design pattern you can speak during interview.

- **“Adapter pattern”** helps to make incompatible interfaces compatible.

- **“Iterator pattern”** helps iterate over elements of an aggregate object sequentially without exposing the underlying representation of the object.

- **“Template pattern”** define a skeleton of the algorithm in parent class and let subclass override specific steps.

- **“Repository pattern”** helps to abstract and centralize the data access logic in an application. It provides a layer of separation between the application's business logic and the data access code, making the code more modular, maintainable, and testable.

# Q.4 Which are the different types of design patterns?

There are 3 categories of design pattern: -

**1. Creational : -** Problems and solutions around object creation issues.

- **Prototype -** A fully initialized instance to be copied or cloned

- **Singleton -** A class of which only a single instance can exist

**2. Behavioral: -** Problems and solutions around Communication between objects.

- **Chain of responsibility -** A way of passing a request between a chain of objects

- **Command -** Encapsulate a command request as an object

- **Mediator -** Defines simplified communication between classes

- **Memento -** Capture and restore an object's internal state

- **Template method -** Defer the exact steps of an algorithm to a subclass

**3. Structural: -** Solving concerns around class structure and object composition.

- **Adapter -** Match interfaces of different classes

- **Composite -** A tree structure of simple and composite objects

- **Decorator -** Add responsibilities to objects dynamically

# Q.5 Which Design Pattern Have you used in your Project?

**Note: - F**irst and foremost do not answer SINGLETON pattern as it will not help you stand in the crowd. and remember: -

- My choice of patterns is not yours so pick what you are comfortable.

- Only talk about patterns you are confident of. Many patterns you have already used in your projects. If you can have closer look on them you can speak not only confidently but also naturally. When answers are natural interviewers love it.

- Do not just talk GOF but also talk about Non-GOF patterns.

- Yes, avoid singleton stand in the crowd. Its over used and abused in interviews.

Some of the most used Patterns are: -

Most of the applications are CRUD so repository pattern comes at the top.

**1. Repository pattern: -** Acts like an abstract layer between Models and Data access technologies like EF, ADO.NET and so on. Data access logic is centralized making code maintainable, testable and modular.
With repository pattern, UOW goes like hand in gloves.

**2. UOW (Unit of work): -** This pattern helps to manage transactions and changes made to objects. It goes with repository pattern well.

We all know we use FOR EACH so much so the next pattern is also very must used. One important point to note here talk about IEnumerator and IEnumerable as they implement iterator by default.

**3. Iterator pattern: -** helps iterate over elements of an aggregate object sequentially without exposing the underlying representation of the object.

**4. Factory pattern :-** From creational pattern this is the most used one. Creates an instance of several derived classes.
Used when third party components are used.

**5. Adapter pattern :-** Makes incompatible interfaces compatible.

**6. Decorator pattern :-** Add behavior dynamically.

**7. Command pattern:-** Treat command as objects. Heavy use in CQRS

**8. Façade :-** Represent subsystem in a simplified way.

**9. Composite -** A tree structure of simple and composite objects

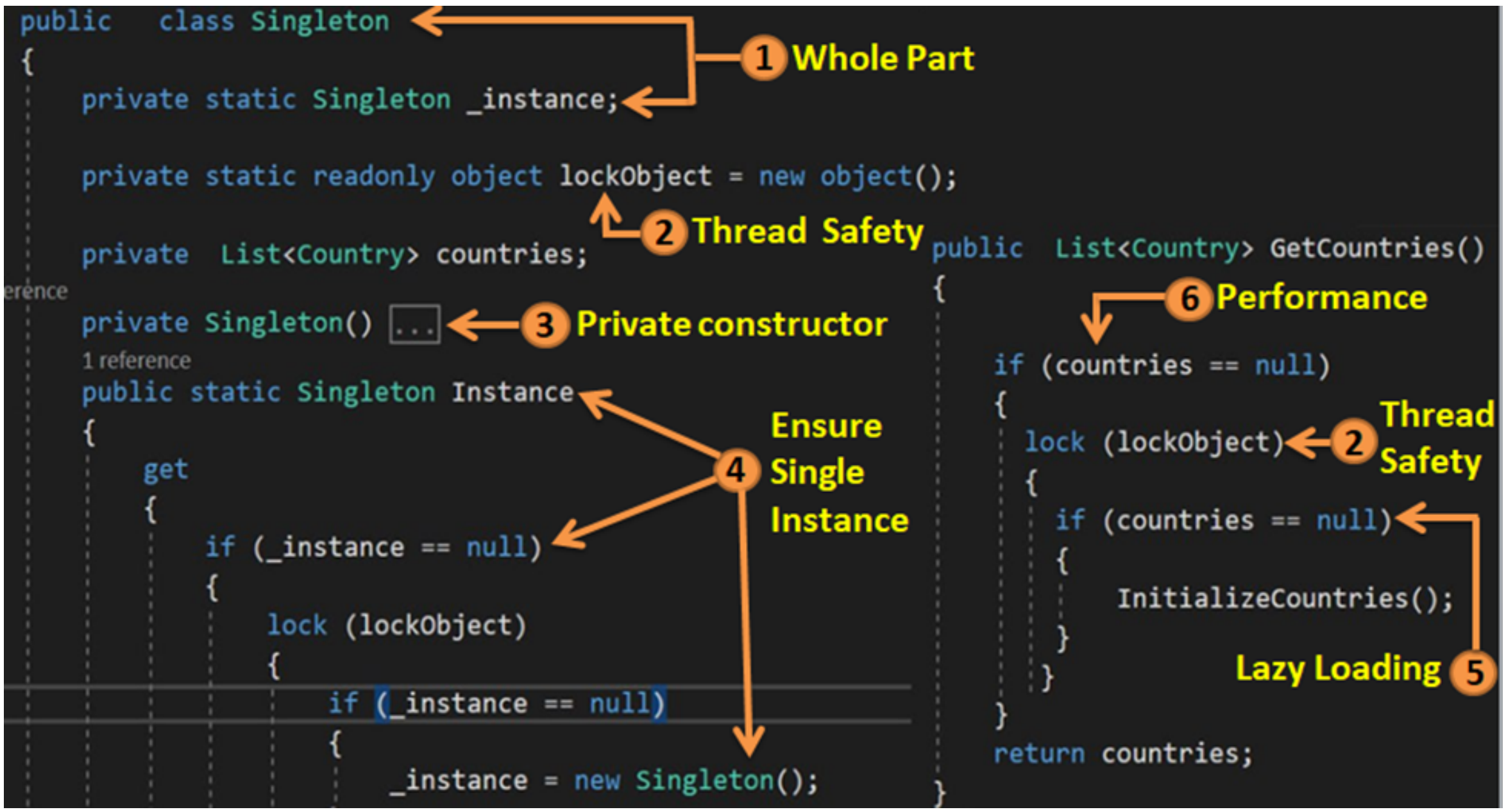
**10.Template method -** Defer the exact steps of an algorithm to a subclass

# Q.6 Explain Singleton Pattern and the use of the same?

Singleton pattern helps to create a single instance of an object. Some of the uses of Singleton patterns are: -

- Caching of data like Countries, States, Currencies and so on.
- Global sharing of data like common themes , hit counters and so on.

# Q.7 How did you implement singleton pattern?




To Implement Singleton Pattern (Check the number with the above figure).

Look at the numbers and check the explanation in the next slide. Explanation and Numbers will help to connect the DOTS.

**1. Whole part relationship: -** The first thing needed in singleton is a root class through which all shared objects should be exposed. This root class instance should be created inside the class and it should be static.

**2. Thread safety: -** Whenever we are loading the singleton object use the “lock” keyword to make sure only one thread manipulates at a time.

**3. Private constructor: -** Make sure the root class can not be instantiated from outside.

**4. Ensure root object is single instance:-** use the double null check and make sure that only one instance is created and also lock is not executed unnecessarily.

**5. Lazy loading implementation:-** We would like to load the objects when demanded then loading it unnecessarily.

**6. Performance for threading: -** NULL check before the locking to ensure that we do not execute lo

# Q.8 Can we use Static class rather than using a private constructor?

# Q.9 Static vs Singleton pattern?

Lets answer both the questions in One Go.

• **Keyword VS Design Pattern: -** Static is a language keyword while Singleton is a Design pattern.

• **Loose many OOP features: -** If you make a class static you cannot implement interfaces, cannot inherit and so on.

• **Lazy loading / Thread Safety: -** When you make a class static, object is created in the first call itself without giving you option of when to load and when not. Second you should also make sure its thread safe as it's a global object.

# Q.10 How did you implement thread safety in Singleton?

C# “lock” keyword helps to implement thread safety.

Whatever code is in the scope of “lock” keyword will get executed by only ONE thread at time avoiding any thread unsafe situation.

```
1 reference
public static Singleton Instance // 4. Giving access
{
    get
    {
        if (_instance == null)
        {
            lock (lockObject)
            {
                if (_instance == null)
                {
                    _instance = new Singleton();
                }
            }
        }
        return _instance;
    }
}
```

Thread Safety

# Q.11 What is double null check in Singleton?

```
1 reference
public static Singleton Instance // 4. Giving access
{
    get
    {
        if (_instance == null)
        {
            lock (lockObject)
            {
                if (_instance == null)
                {
                    _instance = new Singleton();
                }
            }
        }
        return _instance;
    }
}
```

Performance

Double Null Check

Performance Intensive

Lazy loading

Double null check is done for two purposes.

**Internal null check: -** This is done to make sure the instance load only once and its loaded OnDemand.

**External null check :-** This is done so that “lock” is not acquired unnecessarily. “lock” is an intensive process and should be executed only when the singleton is null.

# Q.12 Can Singleton Pattern Code be made easy with Lazy keyword?

Yes, by using LAZY keyword we can make the code size smaller. LAZY makes code thread safe and also does late initialization.

```
public List<Country> GetCountries()
{
    // 6. Lock performance
    if (countries == null)
    {
      lock (lockObject) // 2. Thread
      {
        if (countries == null) // 5.
        {
            InitializeCountries();
        }
      }
    }
    return countries;
}
```

11 Lines of code

```
public List<Country> GetCountries()
{
    return countries.Value;
}
```

Compacted to Single line of code

# Q.13 Can We Rid of This Double Null Check Code?

Yes by using LAZY keyword you can get rid of double null check.

# Q.14 What are GUI architecture patterns, can you name some?

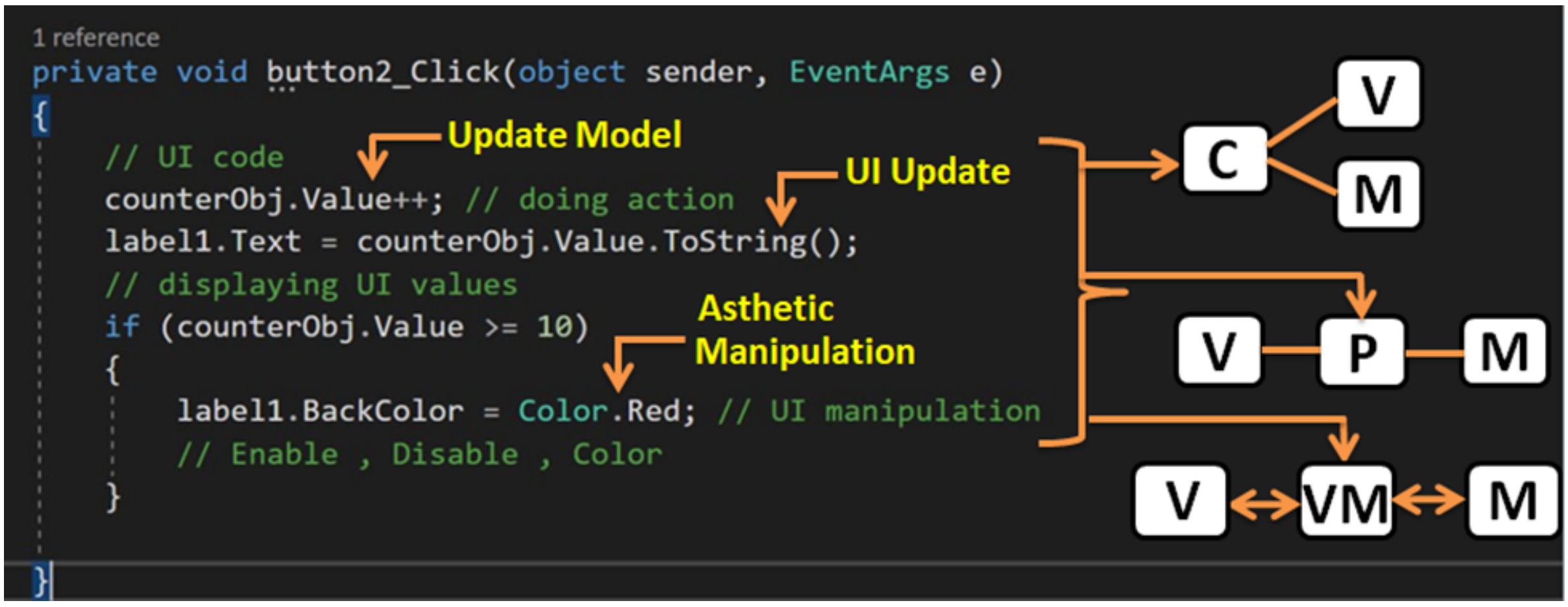
GUI Architecture Pattern refers to how to organize structurally the interaction between User, View (UI) and components (Models, Business logic) in a software application. Its helps to maintain clear separation of concerns in distinct layers.

Some of the GUI architecture patterns are: -

**MVC: -** Model View Controller.

**MVP: -** Model View Presenter.

**MVVM: -** Model View ViewModel.




If you take an example of a typical GUI code it has lot of plumbing code as shown in the below figure. This makes the code more complex and difficult to maintain. A typical UI code does following things:-

- Calling, loading and updating model.
- Updating UI from the Model and vice-versa.
- Enabling, Disabling, changing color and so on.

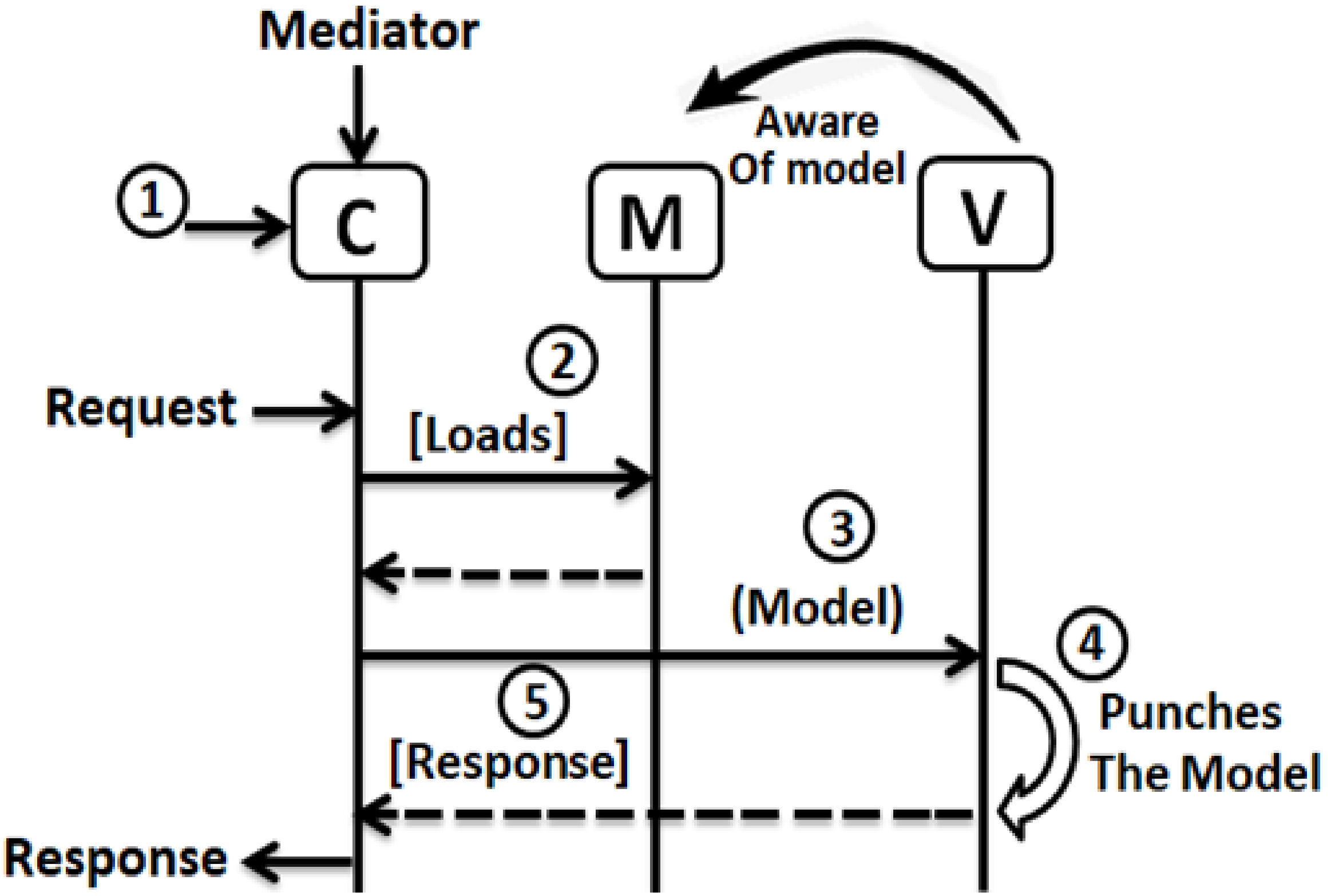
So rather than GUI getting loaded with all CONCERNS we put them in to different sections and layers.

# Q.15 Explain term Separation of concerns (SOC) ?

Separation of Concerns (SoC) is a design principle that advocates breaking a software system into distinct sections, each addressing a separate concern or aspect of functionality.

For example MVC , MVP and MVVM apply separation of concerns concept.

# Q.16 Explain MVC Architecture Pattern?



In MVC architecture we divide the project in to three primary layers: Controller, View and Model.

**Controller: -** Takes user inputs, loads model and passes to the view.

**Model: -** This section has business logic.

**View: -** This section handles the look and feel, positioning and alignment etc.

In MVC the first hit comes to the controller, controller then loads the model and attaches it to the view. Controller acts like a mediator between View and Model.

# Q.17 Explain MVP Architecture Pattern?

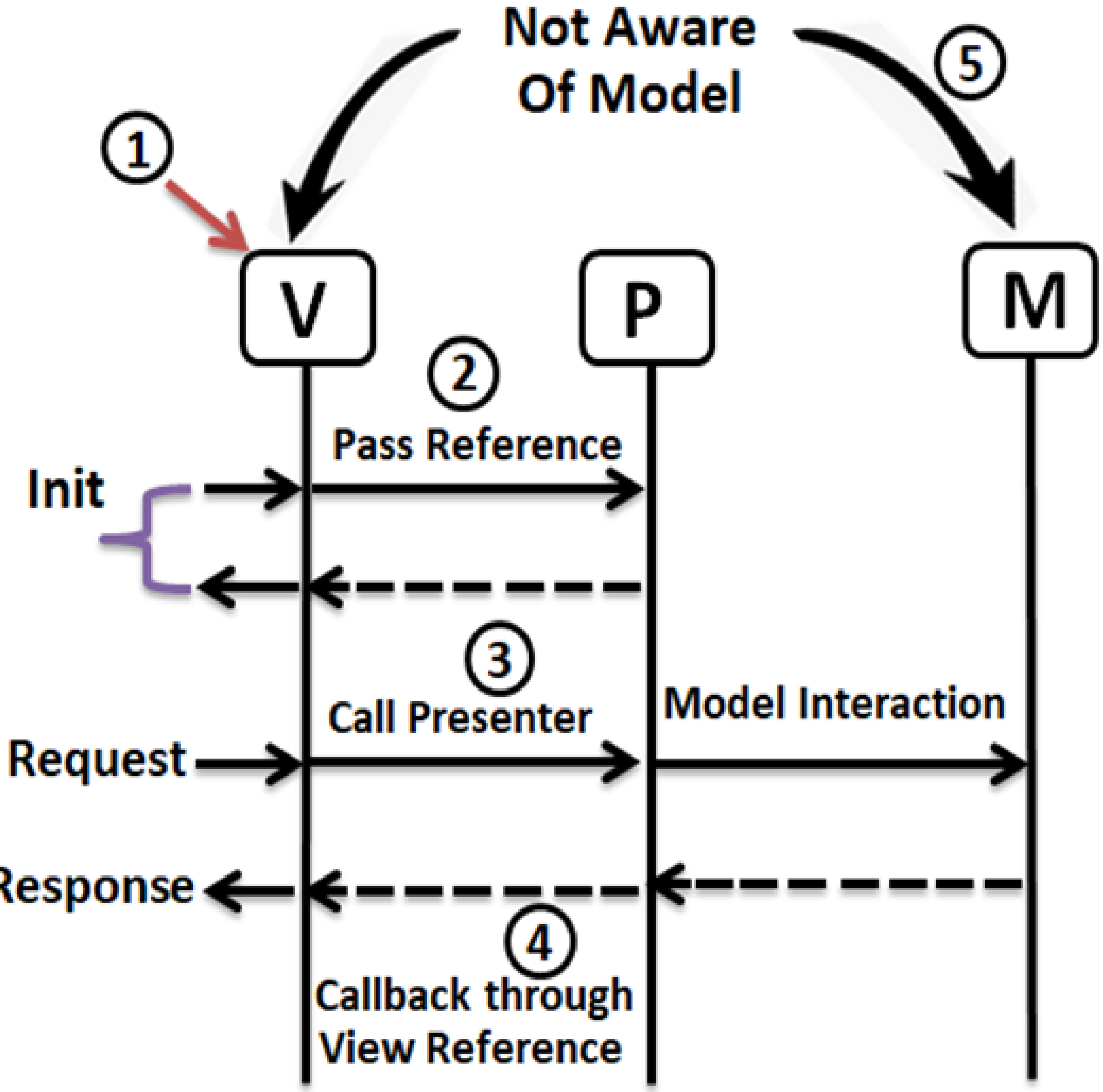
In MVP architecture we divide the project in to three primary layers: Presenter, View and Model.

**View: -** This section takes user control, handles the look and feel, positioning and alignment etc. It forwards all user interaction to the presenter.

**Presenter: -** Takes the input from the view, loads the model and updates the view accordingly.

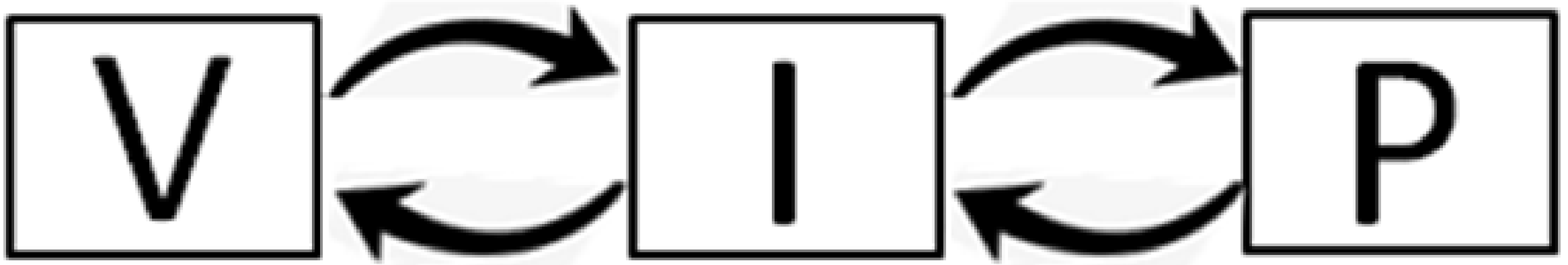
**Model: -** This section has business logic.

In MVP the first hit comes to the view, view passes the call to presenter, presenter calls the model and updates the view.
Presenter updates the view via a callback.


Not Aware
Of Model
1
V
P
M
5
2
Pass Reference
Init
3
Call Presenter
Model Interaction
Request
Response
4
Callback through
View Reference

# Q.18 What is the importance of interface in MVP ?

Interface helps to connect the view and presenter. Presenter calls / Updates the View via the Interface callback.



# Q.19 What is Passive View?

Passive view is view which does not contain any logic related to user interaction and presentation. Passive View submits itself to the presenter for all UI interaction logic and follows the presenter like deity.

Here view does two things: -

- Has User controls.
- Passes the UI interaction to the presenter.
- Waits for the presenter call back to update the UI.

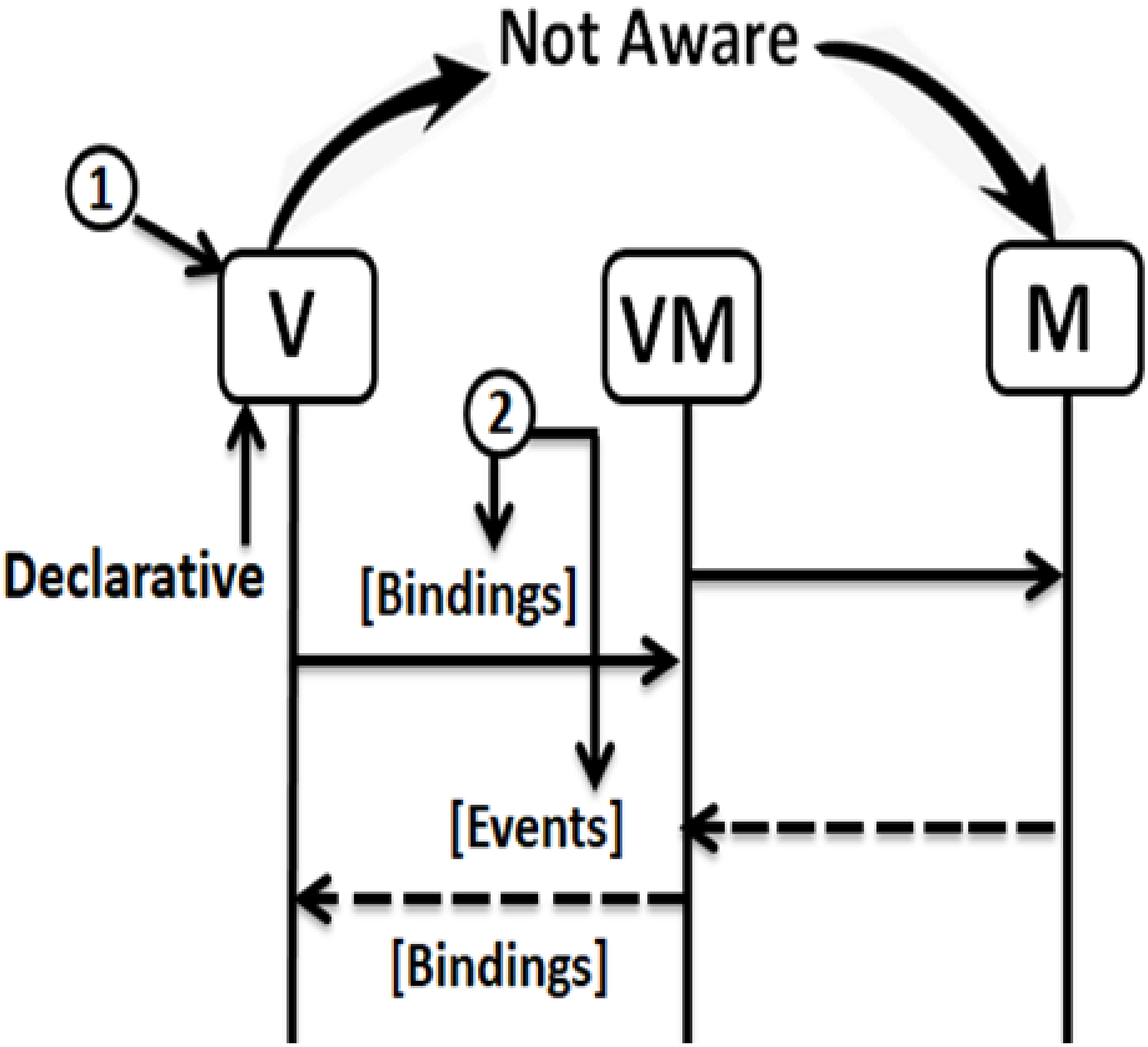
# Q.20 Explain MVVM Architecture Pattern?

In MVM architecture we divide the project in to three primary layers: ViewModel, View and Model.

**View: -** This section takes user control, handles the look and feel, positioning and alignment etc. All Actions and updates to the view happens through automated bindings.

**ViewModel: -** This layer is the mediator between View and Model and does all heavy lifting. It talks with the model , updates the view. ViewModel and the view are binded through automated bindings.

**Model: -** This section has business logic.




In MVVM the first hit comes to the view, view interacts with the viewmodel through automated declarative bindings and commands. ViewModel loads the model , has UI logic and updates the view through automated bindings.

# Q.21 What is the Difference Between MVP and MVVM ?

MVP does not have automated bindings and it Submits itself completely to presenter.

While In case of MVVM view talks to View Model Through bindings.

In both cases the first hit comes to view.

# Q.22 What is a ViewModel?

**Note:-** We have ViewModel term in MVVM also and also generically ViewModel class is used. This answer is from the more generic perspective.

ViewModel class is used generically in MVC, MVP and in many other architectures as well. The ViewModel class in MVVM has specialty of bindings.

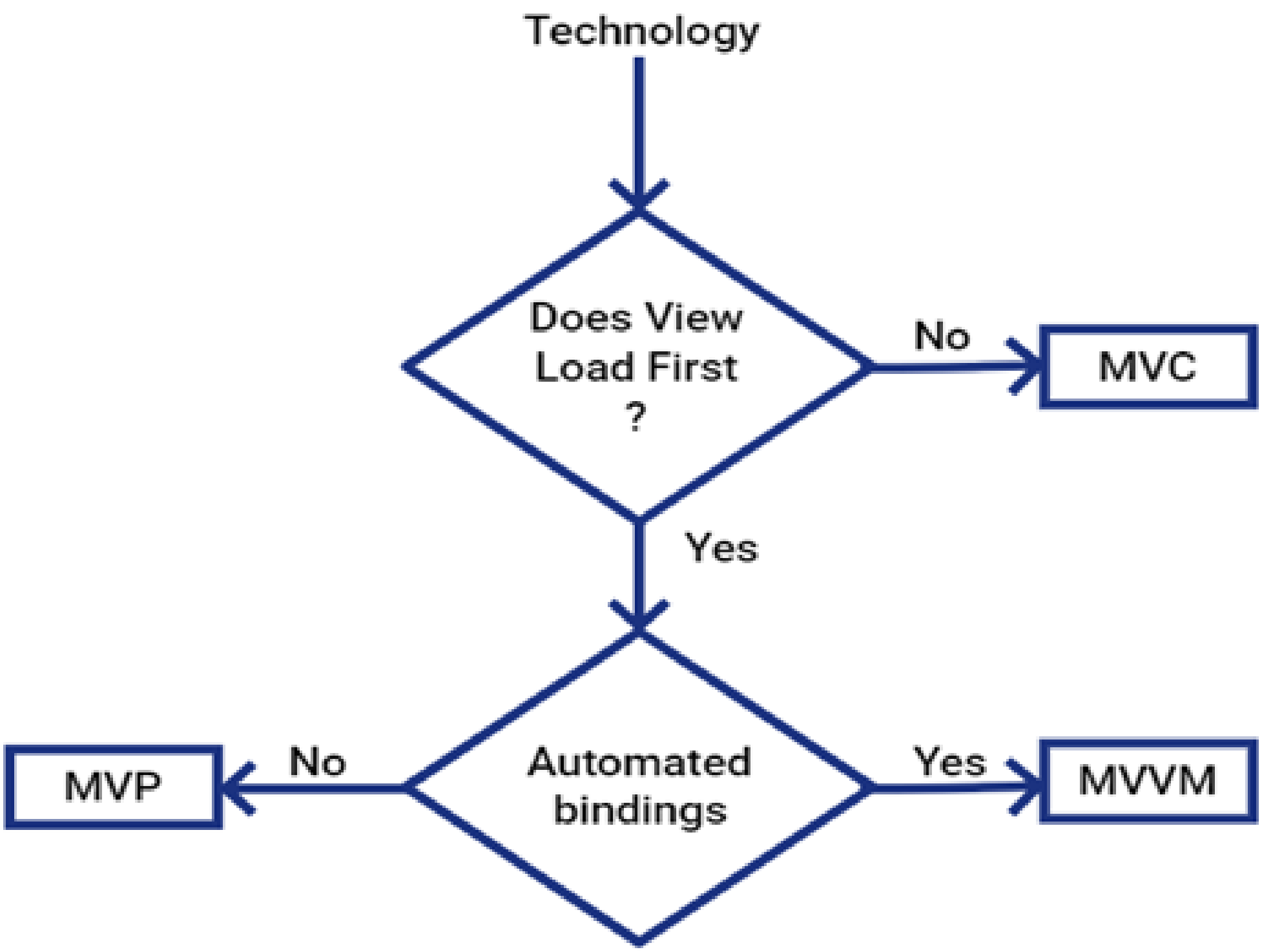
ViewModel acts like a mediator between View and Model. ViewModel encapsulates the model and provides all things needed by view. ViewModel provides two things to view : -

- **Model Data:-** examples - Customer code Customer name.

- **View Data:-** Enable , Disable , Color , Aesthetics and so on.

# Q.23 When to use what MVP / MVC / MVVM?

Putting it simple choice of these Architecture Pattern is not in the hand of the Architect but Rather what kind of Technical Support that Technology give. So here is the Decision road Map.



- If your technology is loading the view first then you have only choice of MVP and MVVM.

- If your technology supports automated bindings then MVVM works well.

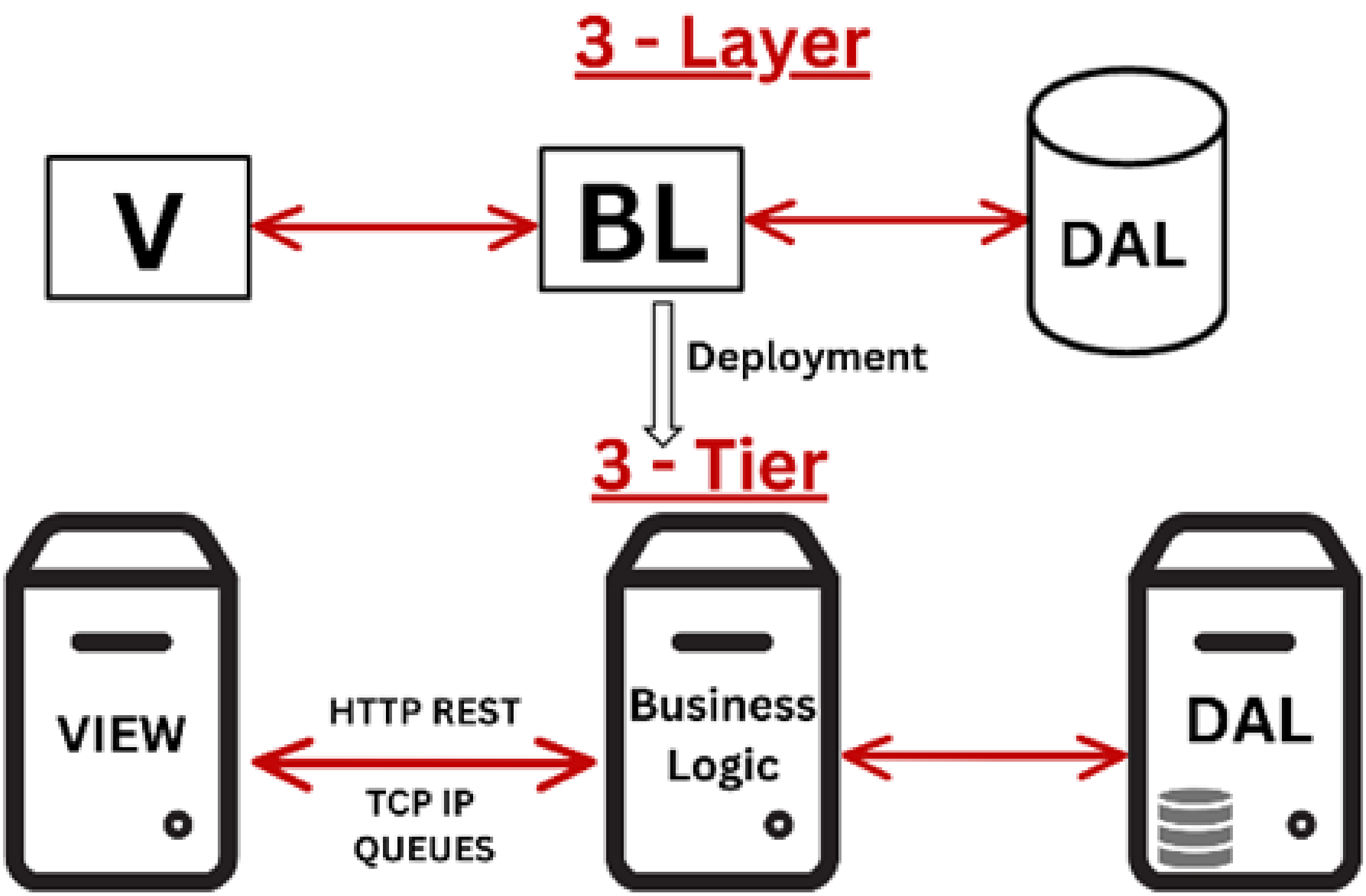
- If your technology does not load view first , like controller / component is loading the view then MVC suits well.

**Note:-** You can end up with combinational architecture also like Angular , component loads the view and also it has automated bindings. So it uses MVVM and MVC both.

# Q.24 MVC vs MVP vs MVVM?

| | MVC | MVP | MVVM |
|---|---|---|---|
| **Flow** | Controller→Model/View | View→ Presenter →Model | View→VM→Model |
| | In MVC direct hit comes to controller while in case of MVP and MVVM direct hit comes to the view. | | |
| **Where to use?** | ASP.NET MVC,Spring,Laravel,Angular | Webform,Winform,GWT,VAADIN Legacy. | WPF,Angular,Aurelia,Knockout, React |
| **Automated Bindings** | NA | NA | Declarative bindings. |

# Q.25 Layered Architecture vs Tiered Architecture?



Both layer and Tier emphasize SOC (Separation of concerns). So in both 3-layer and 3-tier we have 3 different units View, BL and Dal.

**View: -** Takes care of Look and Feel, Positioning, Aesthetics etc.

**Business Logic: -** This section as Business validations.

**Dal:-** This section interacts with data store.

**When we use the word 3-Layer: -** We are discussing about how the project is organized logically.

**When we use the word 3-tier: -** We are emphasizing that these sections are deployed in different machines/hardware.

So in case of 3 tier the complexity increases as we need to now think on network communication ( HTTP,TCP etc ) and so on between these tiers.

WWW.QUESTPOND.COM

# FOR DETAILED AND PRACTICAL ANSWERS WATCH 2 HOURS OF SOFTWARE ARCHITECTURE & DESIGN PATTERN INTERVIEW QUESTIONS VIDEO

LINK IN COMMENT

FOLLOW US ON