Programming Technology
BECE IV (Pokhara University)

Prepared by

Asst. Prof. Madan Kadariya

HoD, Department of Information Technology

Nepal College of Information Technology (NCIT)

Chapter 2: Programming Architecture

OUTLINE

- Model View Controller (MVC)
- Client Server Model
  - 2 tier client server architecture
  - 3 tier client server architecture
  - N tier client server architecture
  - Comparison between client server architectures
- MVC vs Client Server Model

Introduction

1. **Design pattern** is a general repeatable solution to a commonly occurring problem in software design.
2. A design pattern isn't a finished design that can be transformed directly into code.
3. It is a description or template for how to solve a problem that can be used in many different situations.
4. **OO design** patterns typically show relationships and Interactions between **classes** or **objects**, without specifying the final application classes or objects that are involved.
5. One of the design patterns is **Model-View-Controller** (**MVC**).

Model View Controller (MVC) Architecture

1. MVC Architecture have three main layers: **Presentation (UI)**, **Application logic**, and **Resource management**
2. The presentation layer is split into **Controller** and **View**.
3. The most important separation is between **Presentation** and **Application logic**.
4. **MVC** encompasses more of the architecture of an application than is typical for a design pattern.
5. The term architectural pattern may be useful, or perhaps an **aggregate design pattern**.

Model View Controller (MVC) Architecture

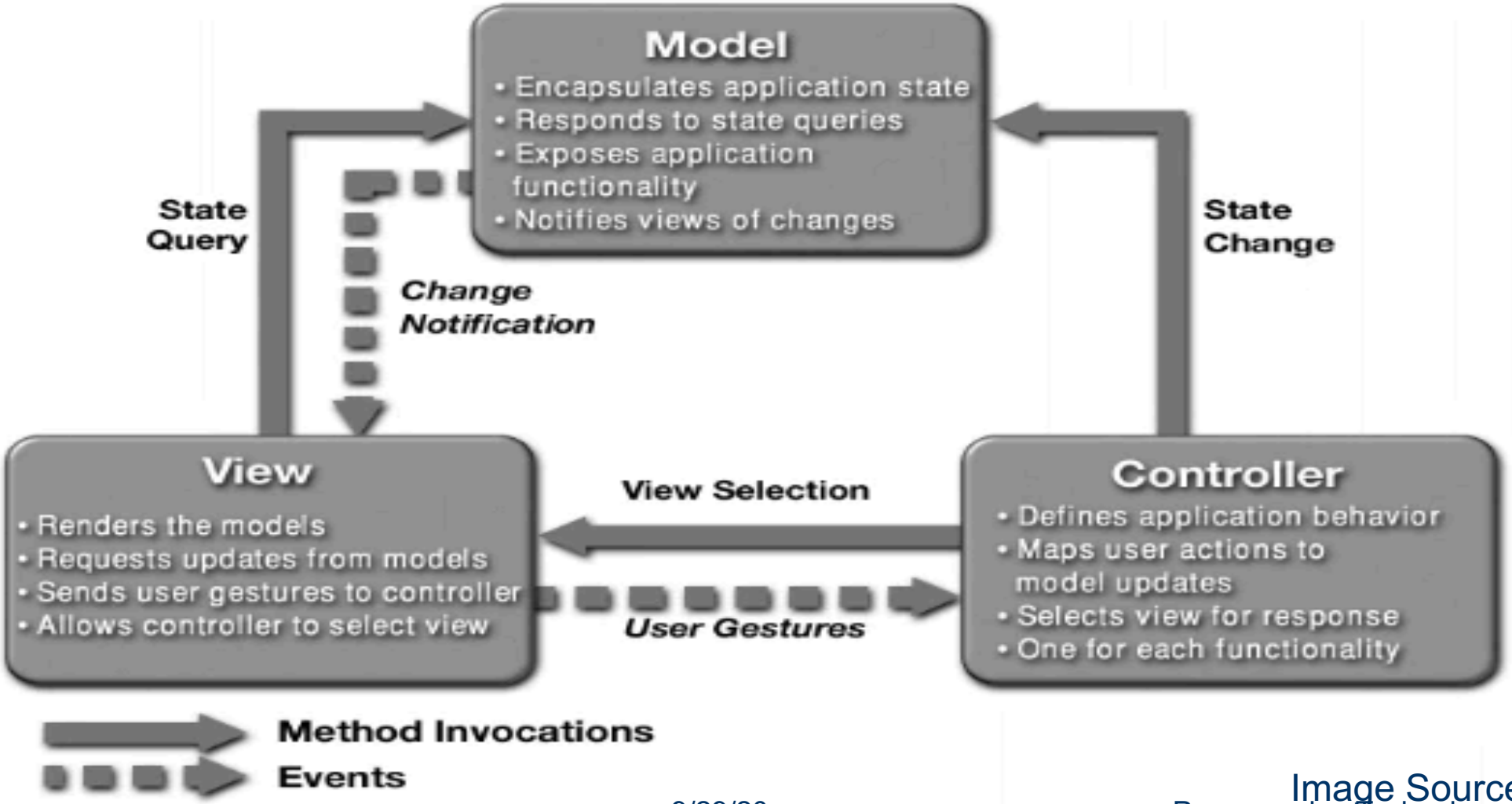


Image Source: Oracle.com

Model View Controller (MVC) Architecture

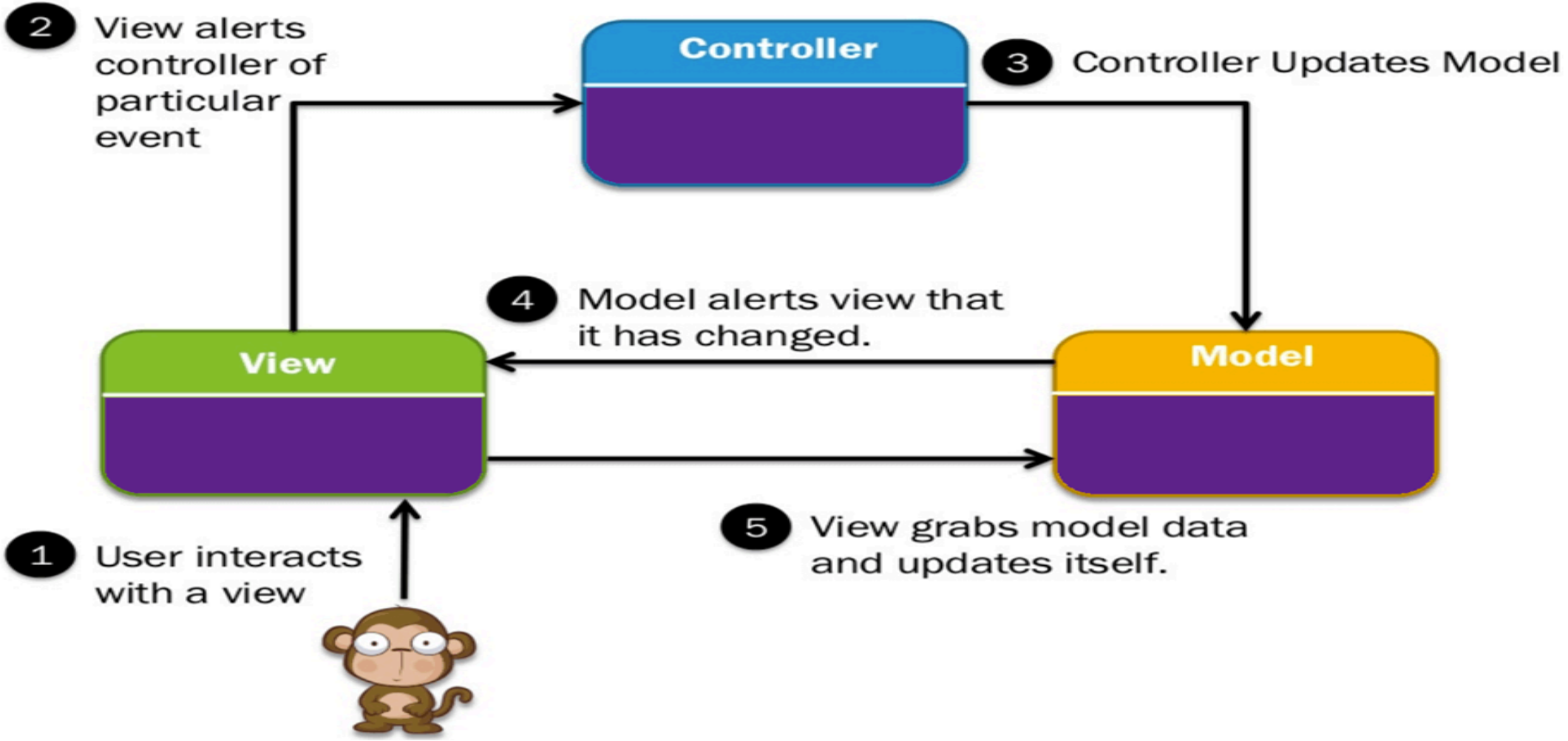


Image source: guru99.com

Model

- The **model** component stores **data** and its related **logic**.
- It represents **data** that is being transferred between **controller** components or any other related **business logic**.
- For example:
  - **Controller** object will retrieve the customer info from the database. It manipulates data and send back to the database or use it to render the same data.
- It responds to the **request** from the **views** and also responds to instructions from the **controller** to update itself.
- It is also the lowest level of the pattern which is responsible for maintaining data.

View

- **Renders** the **model** into a form suitable for interaction, typically a user interface element.
- **MVC** is often seen in web applications, where the view is the HTML page and the code which gathers **dynamic data** for the page.
- A **View** is that part of the application that represents the presentation of data.
- **Views** are created by the data collected from the model data.
- A **view** requests the model to give information so that it resents the output **presentation** to the user.
- The **view** also represents the data from chats, diagrams, and table. For example, any customer view will include all the UI components like text boxes, drop downs, etc.

Controller

- Processes and responds to **events**, typically user actions, and may invoke changes on the **model** and **view**.
- **Controllers** act as an interface between **Model** and **View** components to process all the business logic and incoming requests
- Manipulate data using the **Model** component and interact with the **Views** to render the final output.
- For example, the Customer controller will handle all the interactions and inputs from the Customer View and update the database using the Customer Model.
- The same controller will be used to view the Customer data.

Source:Tutorialspoint.com

Scenario (MVC)

1. The user interacts with the user interface in some way (e.g., user presses a button)
2. A controller handles the input event from the user interface, often via a registered handler or callback.
3. The controller accesses the model, possibly updating it in a way appropriate to the user's action.
4. A view uses the model to generate an appropriate user interface (e.g., view produces a screen listing the shopping cart contents). The view gets its own data from the model. The model has no direct knowledge of the view. (However, the observer pattern can be used to allow the model to indirectly notify interested parties, potentially including views, of a change.)
5. The user interface waits for further user interactions, which begins the cycle anew.

MVC (In Short)

- **Model**
  - Stores the application state
  - Responds to requests for data
  - Encapsulates business logic
- **View**
  - Renders the user interface (UI)
  - Requests data from the Model
  - Sends events to the Model
  - Allows the Controller to select the next View

- **Controller**

  -Handles routing to the correct page

  -Maps the UI data changes to the Model

  -Passes data to and from the Model

Advantages and Disadvantages of MVC

Advantages

- Multiple developers can work simultaneously on the **model**, **controller** and **views**.
- MVC enables **logical grouping** of related actions on a controller together.
- The **views** for a specific model are also grouped together.
- **Models** can have multiple views.

Disadvantages

- The framework navigation can be complex because it introduces new layers of **abstraction** and requires users to adapt to the decomposition criteria of MVC.
- Knowledge on multiple technologies becomes the norm.
- Developers using MVC need to be skilled in multiple technologies.

Client Server Model

Introduction

- Early computer system can best be described as monolithic system in which all processing occurred on the same machine.
- User interacted with the monolithic system through dumb terminal.
- It is desirable to move away from such monolithic system and divide the work between several different computers- this change signaled the birth of client/server systems.
- **Business data** is the most important asset that most companies posses.
- Maintaining database on server separate from the client allows for **greater security, reliability and performance** with regard to data storage while simultaneously reducing the high cost associated with monolithic system.

Example

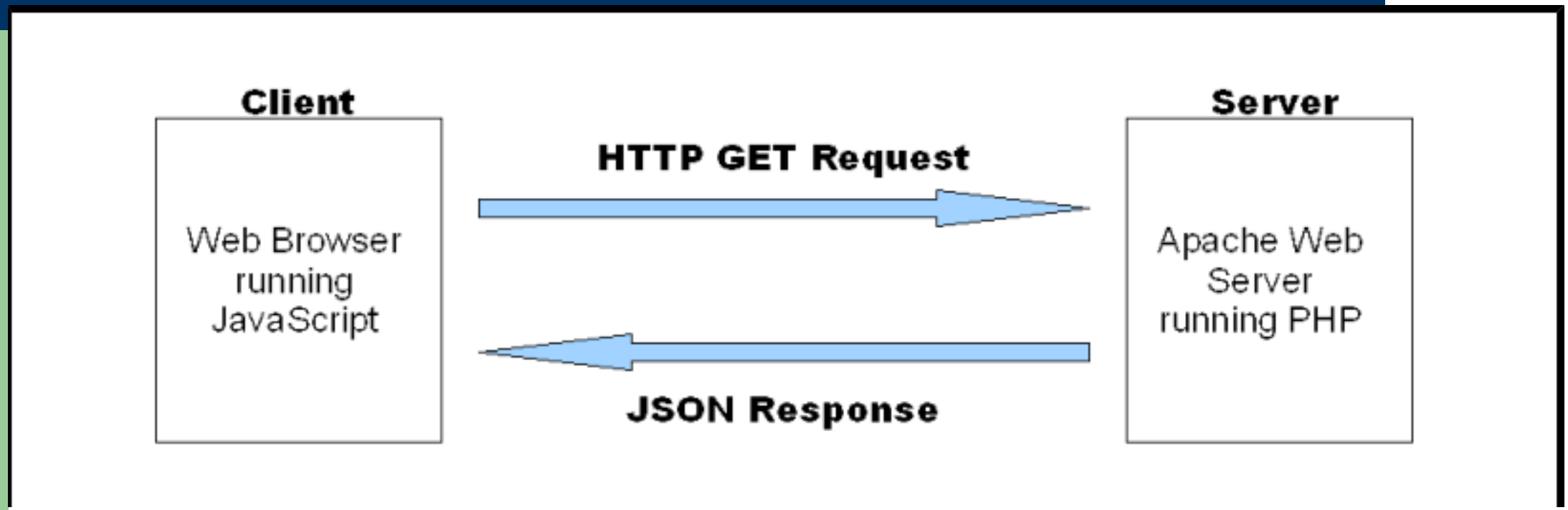


Image Source: researchgate

Client-Server Model

- The client-server model is a **distributed** application structure in computing that partitions tasks or workloads between the providers of a **resource** or **service**, called **servers**, and service **requesters**, called **clients**.
- **Clients** and **servers** communicate over a **computer network** on separate hardware, but both **client** and **server** may reside in the same system.
- A **server** is a host that is running one or more server programs which share their resources with clients.
- A **client does not share any of its resources**, but **requests a server's content** or service function.
- Clients therefore initiate communication sessions with **servers** that await incoming requests.

Client-Server Model

- The **client server** computing works with a system of **request** and response.
- The **client** sends a **request** to the **server** and the **server** responds with the desired information.
- The **client** and **server** should follow a common communication protocol so they can easily interact with each other.
- All the communication **protocols** are available at the application layer.
- A server can only accommodate a limited number of client **requests** at a time.
- An example of a **client server** computing system is a **web server**.
- It returns the web pages to the clients that requested them.

Advantages of Layers and tiers

- **Layering helps** to
  - Maximize **maintainability** of the code
  - Optimize the way that the application works when deployed in different ways
  - Provide a clear delineation between locations where certain technology or design decisions must be made.
- **Placing layers on separate physical tiers** can help
  - **Performance** by distributing the load across multiple servers.
  - It can also help with security by segregating more sensitive components and layers onto different networks or on the Internet versus an intranet.
- A 1-tier application could be a 3-layer application.

Types of Servers

- **Iterative Server:** This is the simplest form of server where a server process serves one **client** and after completing first request then it takes request from another **client**. Meanwhile another client keeps waiting.

- **Concurrent Servers:** This type of **server** runs multiple **concurrent** processes to serve many request at a time. Because one process may take longer and another client cannot wait for so long.

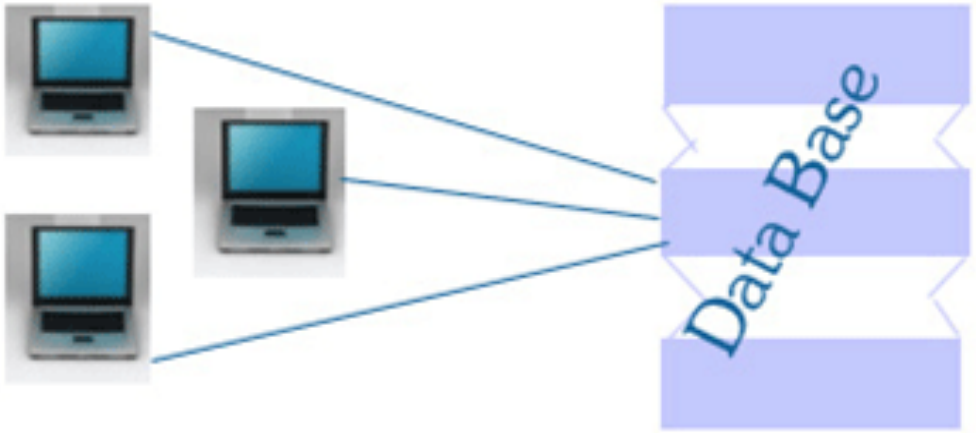
Different Types Of C/S Architectures

1. Client Application (One Tier)
2. 2-tier Client-Server Architecture
3. 3-tier Client-Server Architecture
4. N-Tier Client-Server Architecture

Client Application (One Tier)

- The **program** runs on a single machine.
- In most cases there is no separate application layers.
- Database applications are on the same machine.
- At any given moment, only one user can use the program

2-tier Client-Server Architecture



- The program runs on both machines.
- The **two-tier** architecture is like **client server** application.
- The direct communication takes place between **client** and **server**. There is no intermediate between client and server.
- This architecture increases the **performance**, **flexibility** of the system.
- Example: An architecture that is needed to save the employee details in database. The two tiers of two-tier architecture is
  - Client Application (client tier)
  - Database (data tier)
- This is one to one communication between client and server
- In client application the client writes the program for saving the record in SQL Server and thereby saving the data in the database.

Advantages & Disadvantages Of 2 tier C/S model

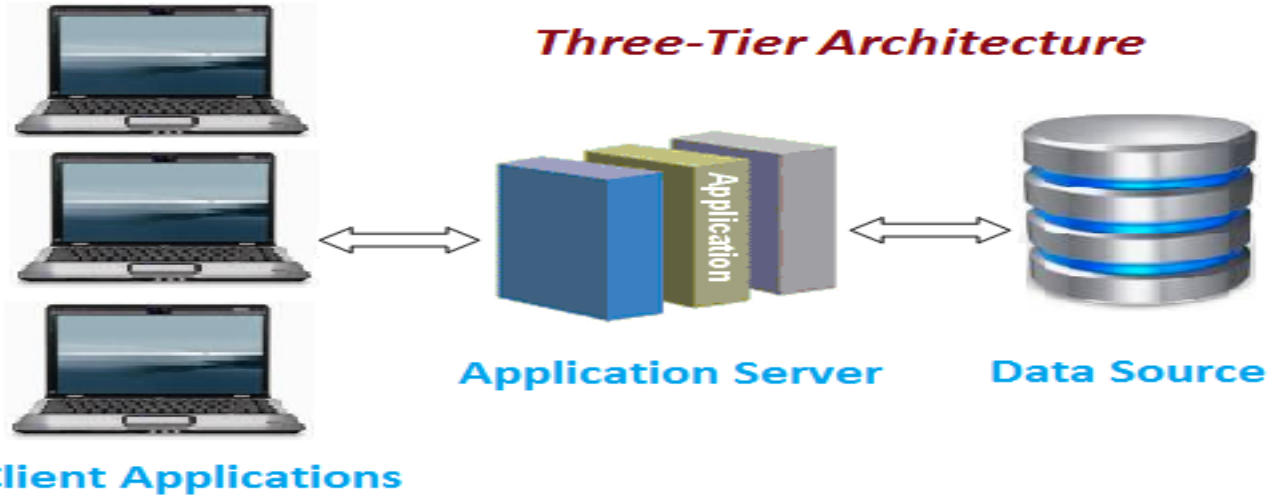
Advantages

- Easy to understand and maintain the architecture
- Cost efficient requires less maintenance cost and Data recovery is possible.
- The capacity of the Client and Servers can be changed separately.

Disadvantages

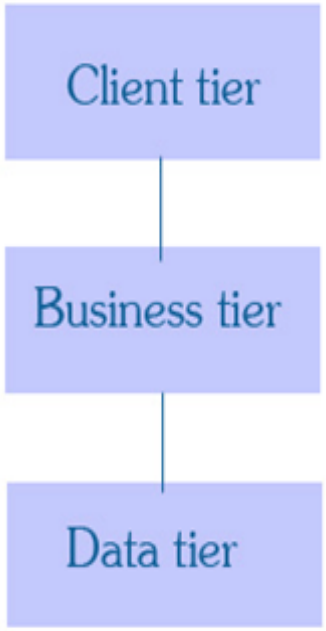
- Database may be too huge to store in the single system (server)
- Performance will be reduced when there are more users.

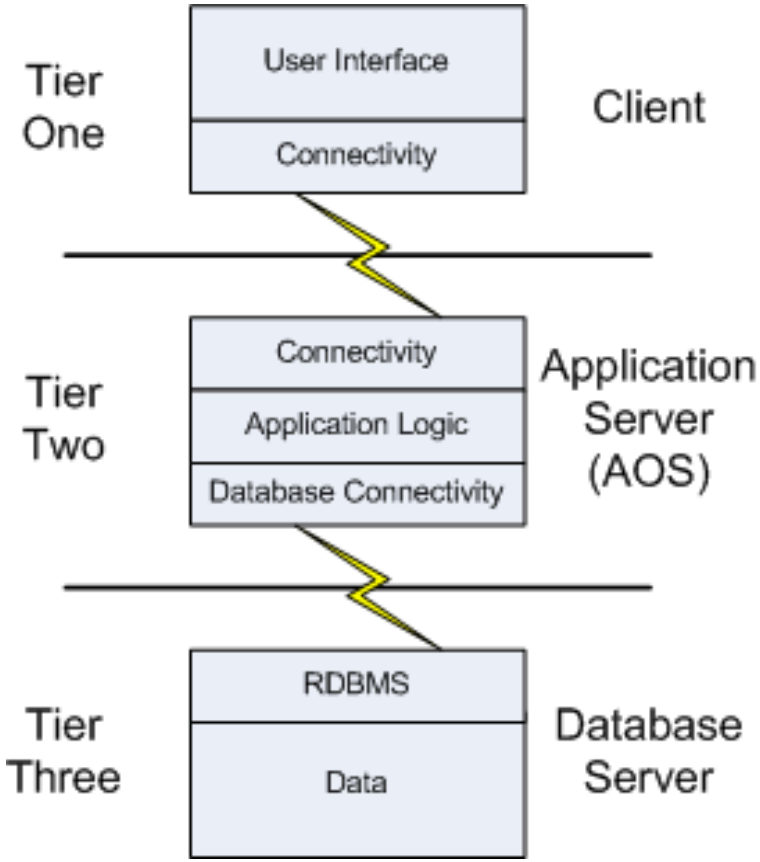
3-tier Client-Server Architecture

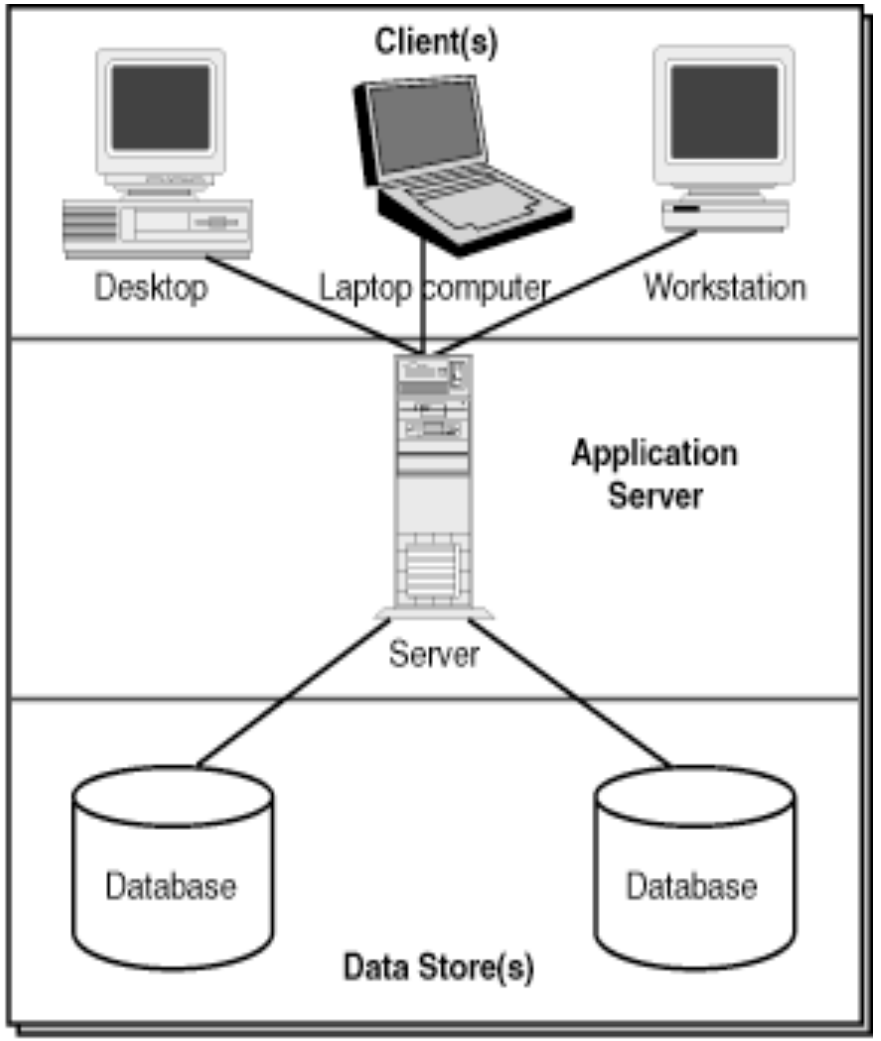


- Comes To Solve **Client/Server** Problems
- There is an additional layer in between client and server that helps clients to get accessed in the database in an efficient way(**application server/business logic**).
- Three tier architecture having **three layers**.
  - **Client layer**: Here we design a client application (say a form using textbox, label etc.)
  - **Business logic layer**: It is the intermediate layer which has the functions for client layer and it is used to make communication faster between **client** and **data layer**. It provides the business processes **logic** and the **data access**.
  - **Data layer**: it has the database. The data tier may the collection of large set of data base servers interconnected with each other in a distributed network.

3-Tier Architecture (Examples)







Advantages and Disadvantages of 3-Tier Architecture

Advantages

- Easy to modify data/application logics with out affecting other modules
- Fast communication
- Performance will be good in three tier architecture.

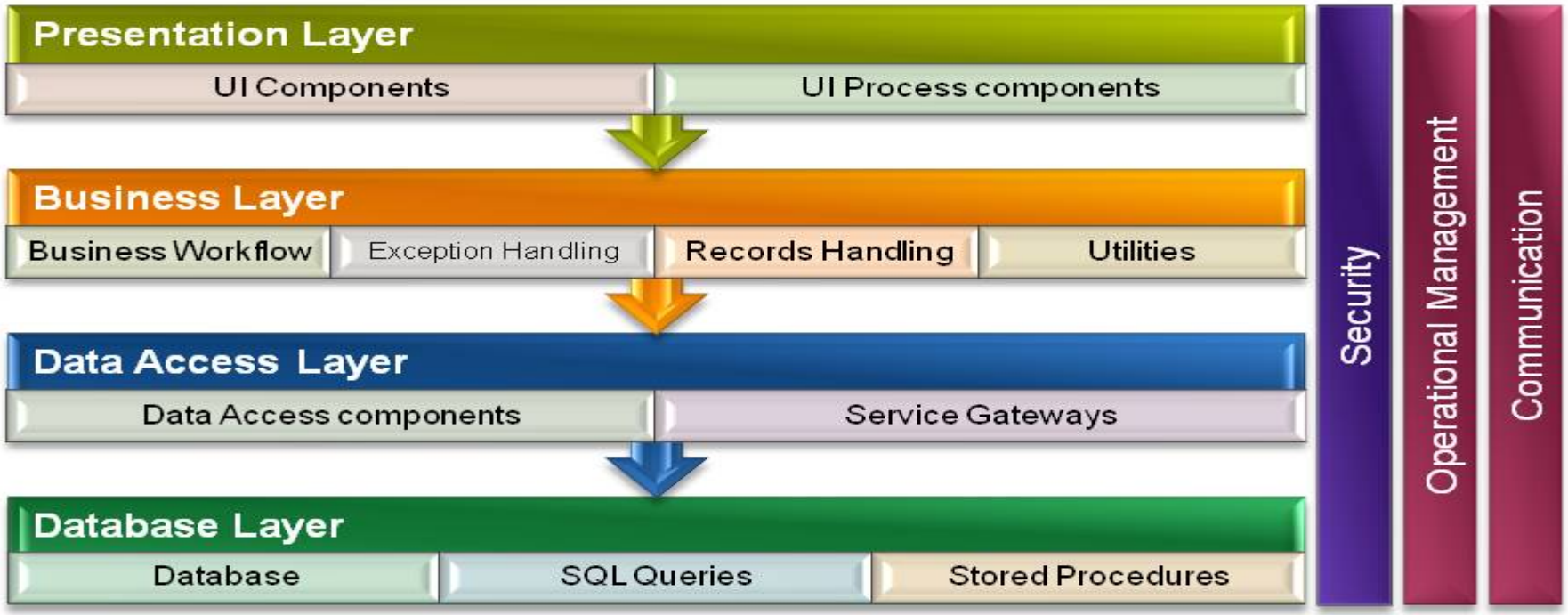
Disadvantages

- Need of database expert e.g, DBA
- Concurrency control mechanism if data is not distributed

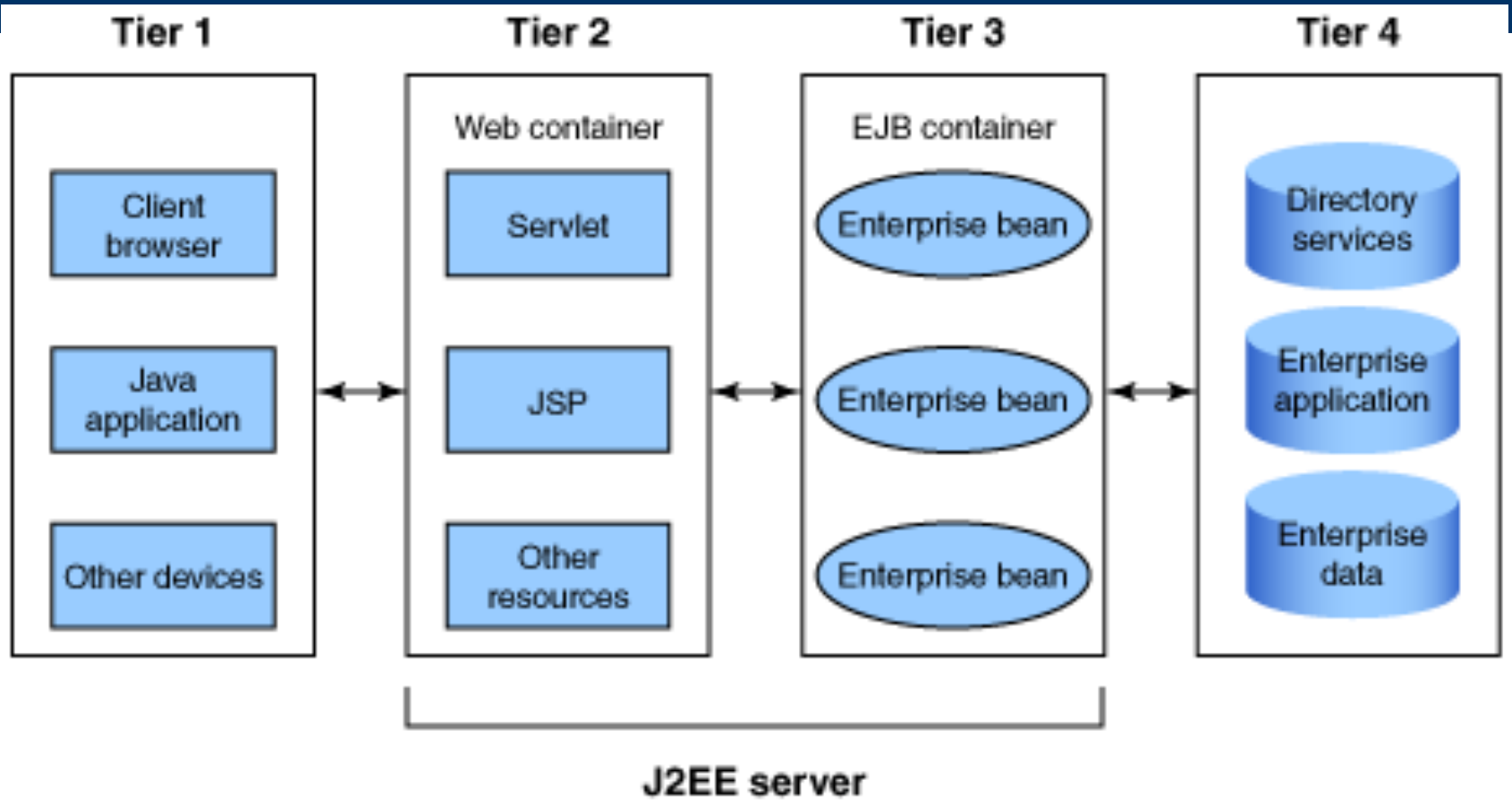
N-Tier Client Server Architecture

- **N-Tier**: An unlimited number of tiers.
- Each tier may have multiple computers.
- Adding some more layers like service layer we can make an N-tier architecture.
  - **Client layer** - Presentation Layer containing UI (eg.written in asp.net
  - **Business layer** – Business Logic Layer containing business logic. UI will call this layer instead of calling data layer directly for security reasons
  - **Service Layer** – problem specific logic or data access layer
  - **Data layer** - Data Access Layer so that all calls to database are abstracted and no-one can fire any query directly into database.

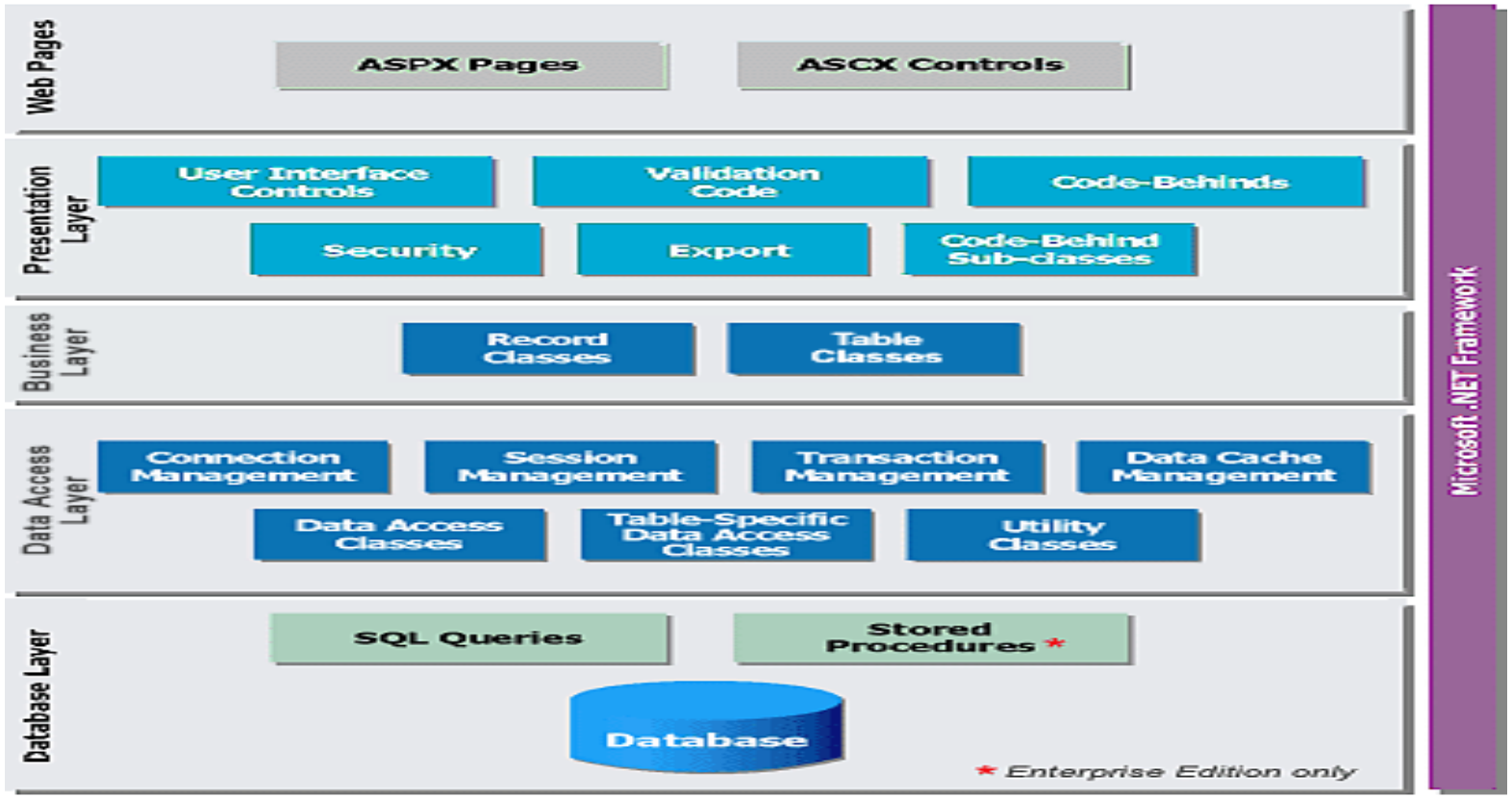
N-Tier Architecture – Example 1



N-Tier Architecture – Example 2



N-Tier Architecture – Example 3



Features (Major Quality Attributes) of N-Tier architectures

1. **Performance**: Greater Performance
2. **Reliability** : New methods are needed
3. **Usability**: Separation makes usability easier to achieve
4. **Security**: Tiers provide for security “walls”
5. **Availability**: Tiers enhance redundancy
6. **Scalability**: Fairly easy to expand services
7. **Maintainability**: Good design – maintenance is easy, bad design – maintenance is hard

Advantages and Disadvantages of N-tier architecture

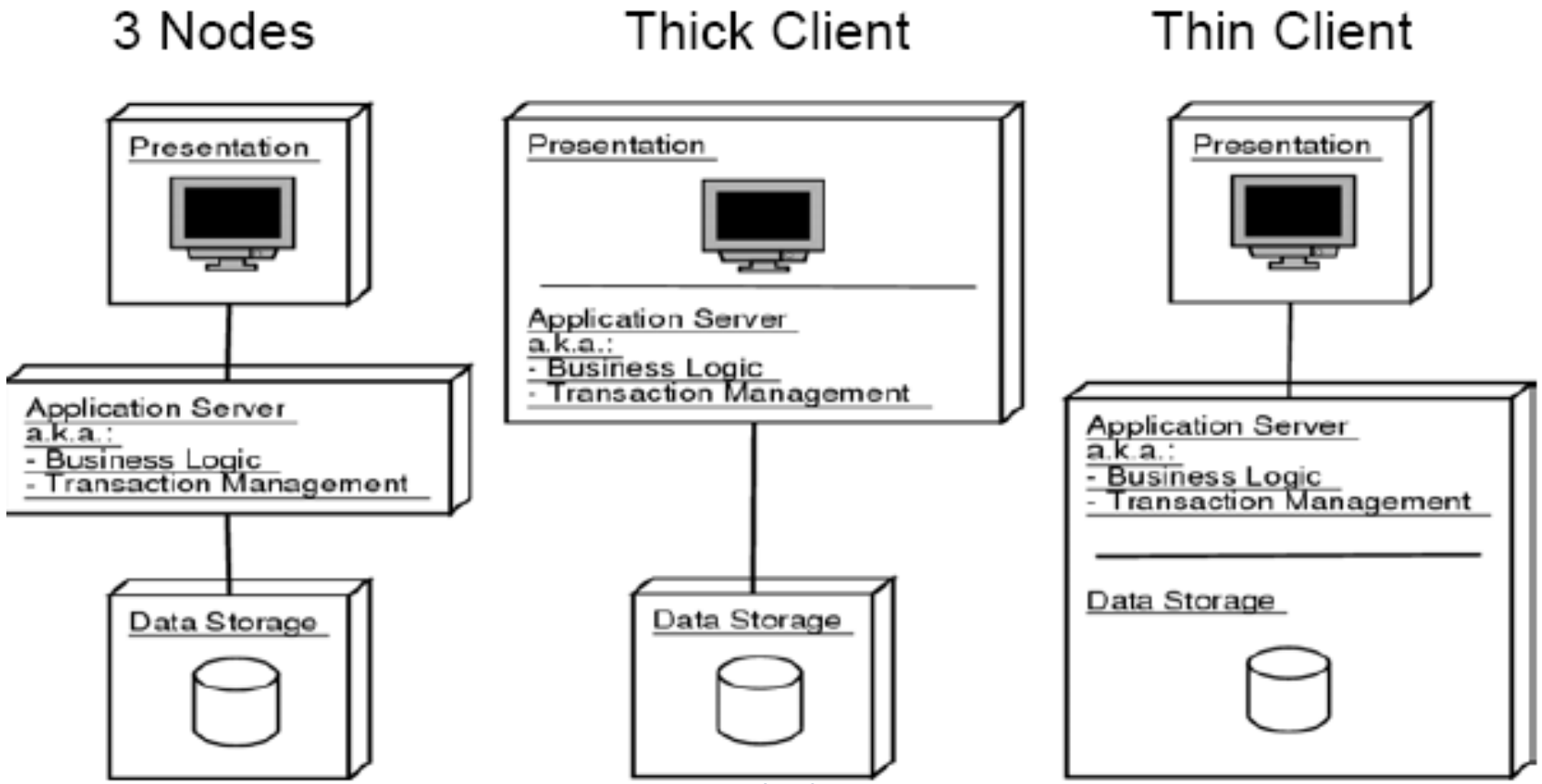
Advantages:

- More **powerful** applications
- Many **services** to many clients
- **Enhanced security**, **scalability** and **availability**

Disadvantages:

- Much more complicated to design and model
- Performance risks
- Reliability is more difficult to achieve
- More difficult to maintain software

Thick and Thin Client Architectures



Comparison of Architectures

Architecture	Advantage	Disadvantage
One tier	Simple Very high performance, Self-contained	No networking – can't access remote services Potential for spaghetti code
Two tiers	Clean, modular design Less network traffic Secure algorithms Can separate UI form business logic	Must design/implement protocol Must design/implement reliable data storage
Three tiers	Can separate UI, logic, and storage Reliable, replicable data Concurrent data access via transactions Efficient data access	Need to buy DB product Need to hire DBA Need to learn SQL Object-relational mapping is difficult
N tiers	Support multiple applications more easily Common protocol/API	Less efficient Must learn API (CORBA, RMI, etc.) Expensive products More complex, more faults Load balancing is hard

MVC Vs C/S Model

Communication: A fundamental rule in a three-tier architecture is the **client tier** never communicates directly with the **data tier**; in a three-tier model all communication must pass through the **middle tier**. Conceptually the three-tier architecture is linear. However, the **MVC** architecture is **triangular**: the **view** sends updates to the **controller**, the **controller** updates the **model**, and the **view** gets updated directly from the model.

History: The three-tier architecture concept emerged in the 1990s from observations of distributed systems (e.g., web applications) where the client, middleware and data tiers ran on physically separate platforms.
Whereas **MVC** comes by work at Xerox PARC in the late 1970s and early 1980s and is based on observations of applications that ran on a single graphical workstation; **MVC** was applied to distributed applications later in its history.

End of Chapter 2