University of Tripoli – Faculty of Information Technology

Software Engineering Department

Software Architecture & Design

ITSE411

y Marwa Solla

Software architecture and design

for modern large-scale systems

Lecture 4 :

Architectural views & styles

What We Learn In This Lecture

- Architectural views
- Architecture styles (MCV, C/S)

Architectural views

- It is impossible to represent all relevant information about a system's architecture in a single architectural model, as each model only shows one view or perspective of the system.

- It might show how a system is decomposed into modules, how the run–time processes interact, or the different ways in which system components are distributed across a network.

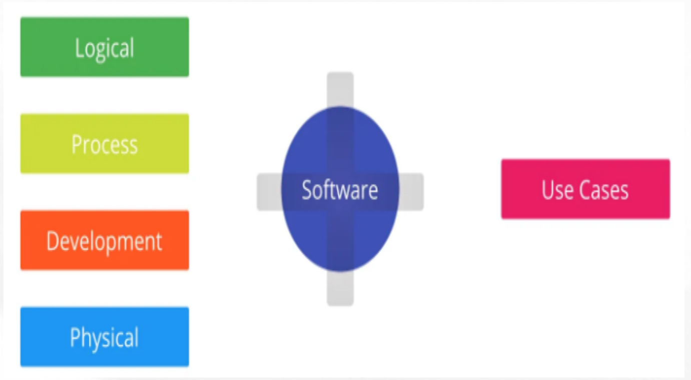
- All of these are useful at different times so, for both design and documentation, you usually need to present multiple views of the software architecture.

Architectural views

There are different opinions as to what views are required. philippe Krutchen (1995), in his well-known **4+1 view model** of software architecture, suggests that there should be four fundamental architectural views, which are related using use cases or scenarios.

The reason behind the name: 4+1

The model has four views: logical, development, process, and physical. In addition, selected use cases or scenarios are utilized as the ‘plus one’ view to show the design. As a result, the model has 4+1 views. Hence the model is called The 4+1 Architectural View.



4 + 1 Architectural View

1. A logical view:

- which shows the key abstractions in the system as objects or object classes. It should be possible to relate the system requirements to entities in this logical view.
- Focuses on the functionality of the system.
- Represents the software components and their interactions without detailing the implementation.
- Represents the hierarchy and structure of the software components.

4 + 1 Architectural View

- UML diagrams are used to represent the logical view, and include class diagrams, and state diagrams.
- The Logical View helps in understanding the system's structure and functionality without getting into the implementation details.
- Viewer: End User.

Note: Logical View: The functionality. The service.

4 + 1 Architectural View

2. A process view,

- which shows how, at run–time, the system is composed of interacting processes. This view is useful for making judgments about non– functional system characteristics such as performance and availability.
- It explains the system processes and how they communicate.
- Captures the concurrency aspects of the design
- distribution, performance, and scalability are all addressed in the process view.
- The sequence diagram, communication diagram, and activity diagram are all UML diagrams that can be used to describe a process view.

Note: Process View: Communication between processes and/or services.

4 + 1 Architectural View

3. A development view,

- It shows how the software is decomposed for development, that is, it shows the breakdown of the software into components that are implemented by a single developer or development team.
- Describes the organization of the software in its development environment. This view can be modelled with UML´s component and package diagrams.
- This view is useful for software managers and programmers.

4 + 1 Architectural View

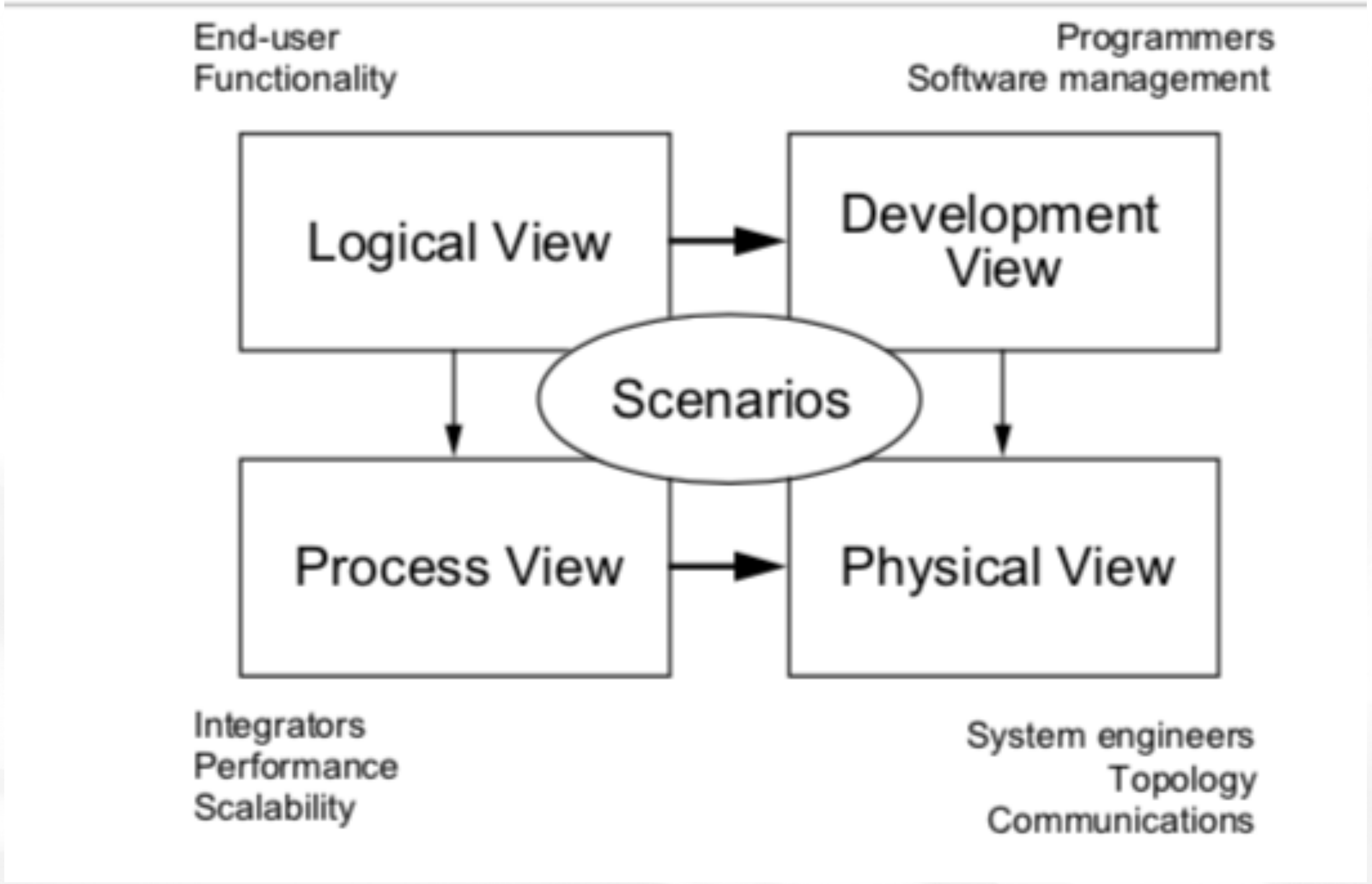
4. A physical view,

- It shows the system hardware and how software components are distributed across the processors in the system. This view is useful for systems engineers planning a system deployment.
- Describes the mapping of the software onto the hardware. Deployment diagrams are used to model this view.

4 + 1 Architectural View

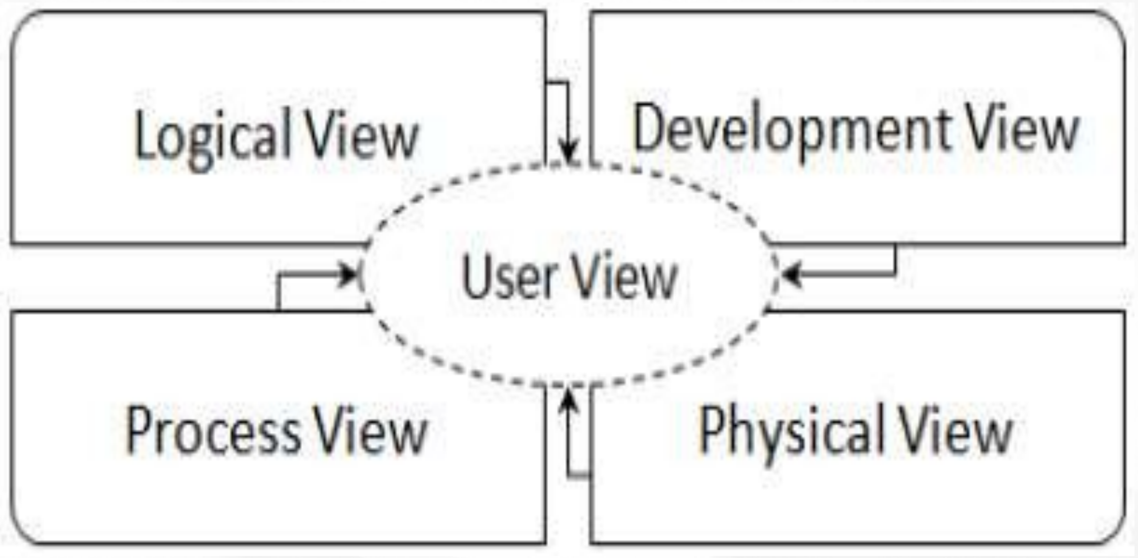
- The +1 comes in from the scenarios view which is what your end users actually care about. It's the system functionality/capabilities
- Use Case View: It is illustrated by selected use cases or scenarios. It contains diagrams describing what the system does from a black box perspective. This view contains use case diagrams.

4 + 1 Architectural View



4 + 1 view model of software architecture

- A logical view Structure (Major elements in the structure)
- A process view Dynamic Behavior (Interaction at run time)
- A development(implementation) view written Code Organization
- A physical view deployment/installation over hardware devices
- Requirement (User View) (+1)



4 + 1 view model of software architecture

Benefits of 4+1:

–Better organization with better separation of concern

–The 4+1 maps stakeholders to the information they need, without requiring specific notations.

Drawbacks:

4+1 was created in 1995, where software development was very different than today, there was no need for more than 2 or 3 views. Nowadays, there are many complex systems which can´t be represented correctly with 4+1.

Architecture styles

A software architecture styles/patterns defines the high-level structure and organization of a software system. It outlines the fundamental components, their interactions, and the overall layout of the system.

An architectural style is a set of principles. Sometimes architecture style is called architecture pattern.

Architecture styles

Architectural Styles Combination

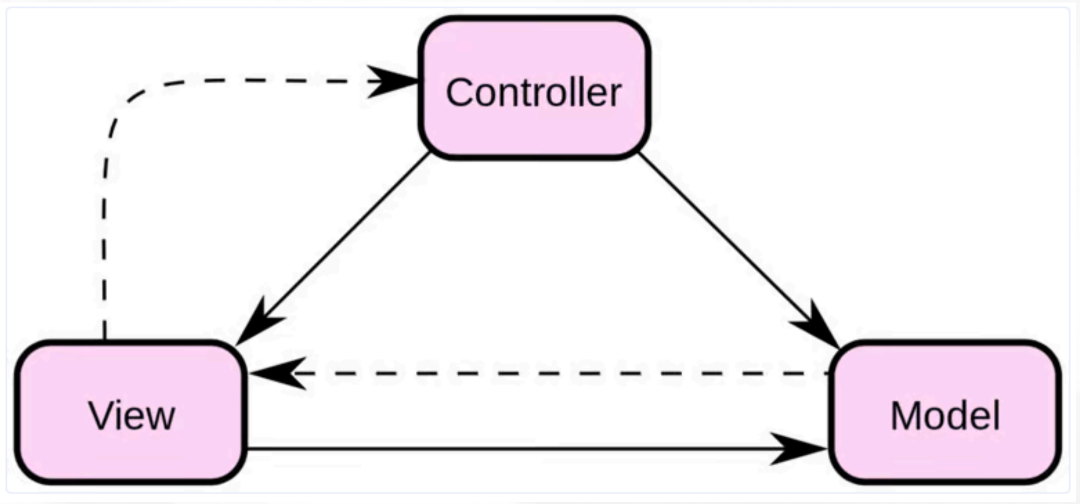
The architecture of a software system is almost never limited to a single architectural style, but is often a combination of architectural styles that make up the complete system.

Many Factors affect the choosing the appropriate style. This includes:

- The capacity of the organization for design and implementation.
- The capabilities and experience of the developers.
- The infrastructure and organizational constraints.

Model–View–Controller pattern

The Model–View–Controller is an architectural pattern (MVC) divides an interactive application into three components. The model contains the core functionality and data. Views display information to the user. Controllers handle user input. Views and controllers together comprise the user interface.



Model–View–Controller pattern

Model:

- The model defines the domain information in the system, independent of how the user interacts with the information.
- Represents the application's data.
- Responsible for managing the data, processing user inputs, and responding to requests from the View and Controller.
- Changes to the data in the Model trigger updates to the View.

Model–View–Controller pattern

View

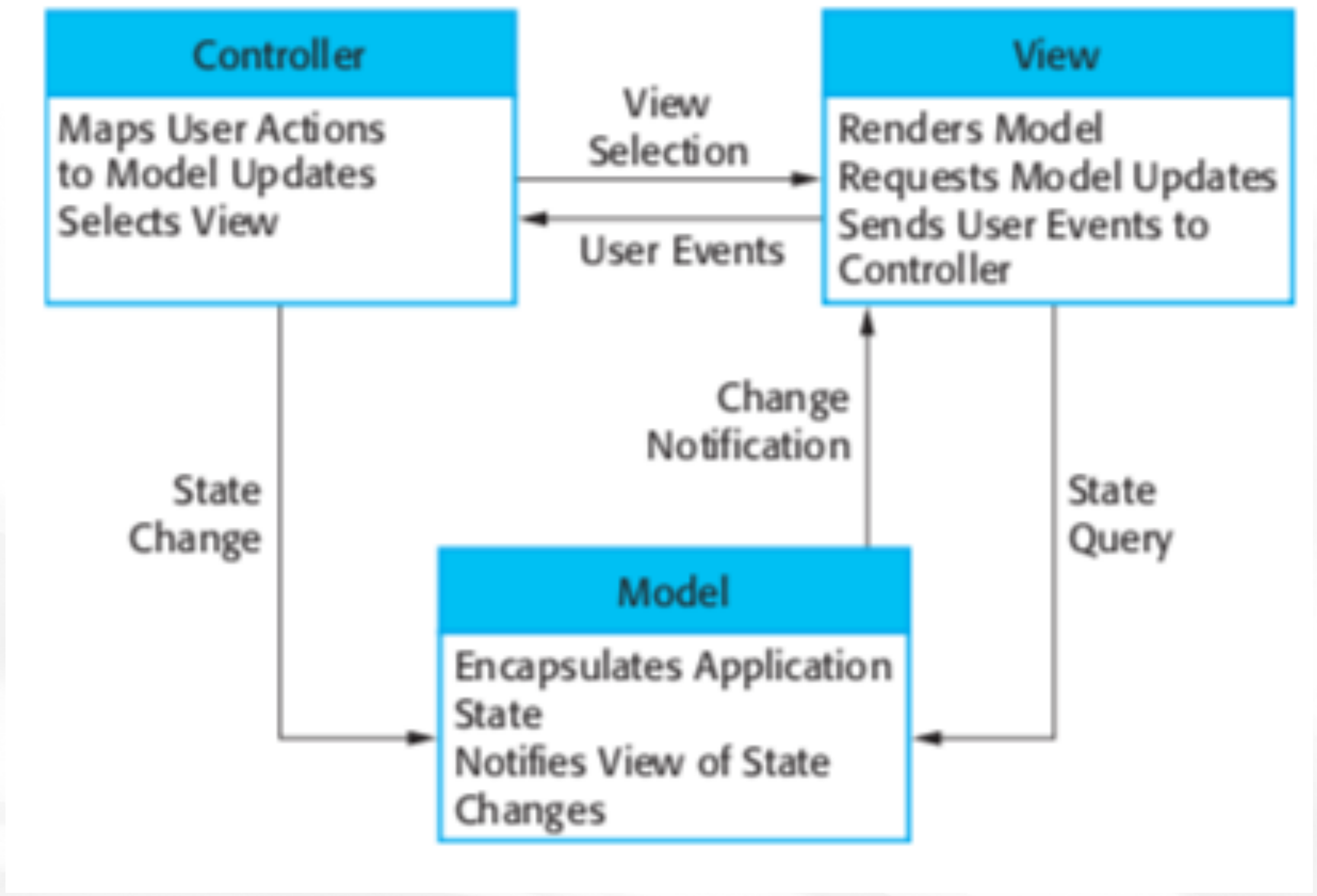
- The view defines the way the information is presented to the human and the acceptable set of manipulation capabilities.
- Represents the user interface (UI) and presentation layer.
- Displays the data from the Model to the user.
- Listens for user input and forwards it to the Controller for handling.

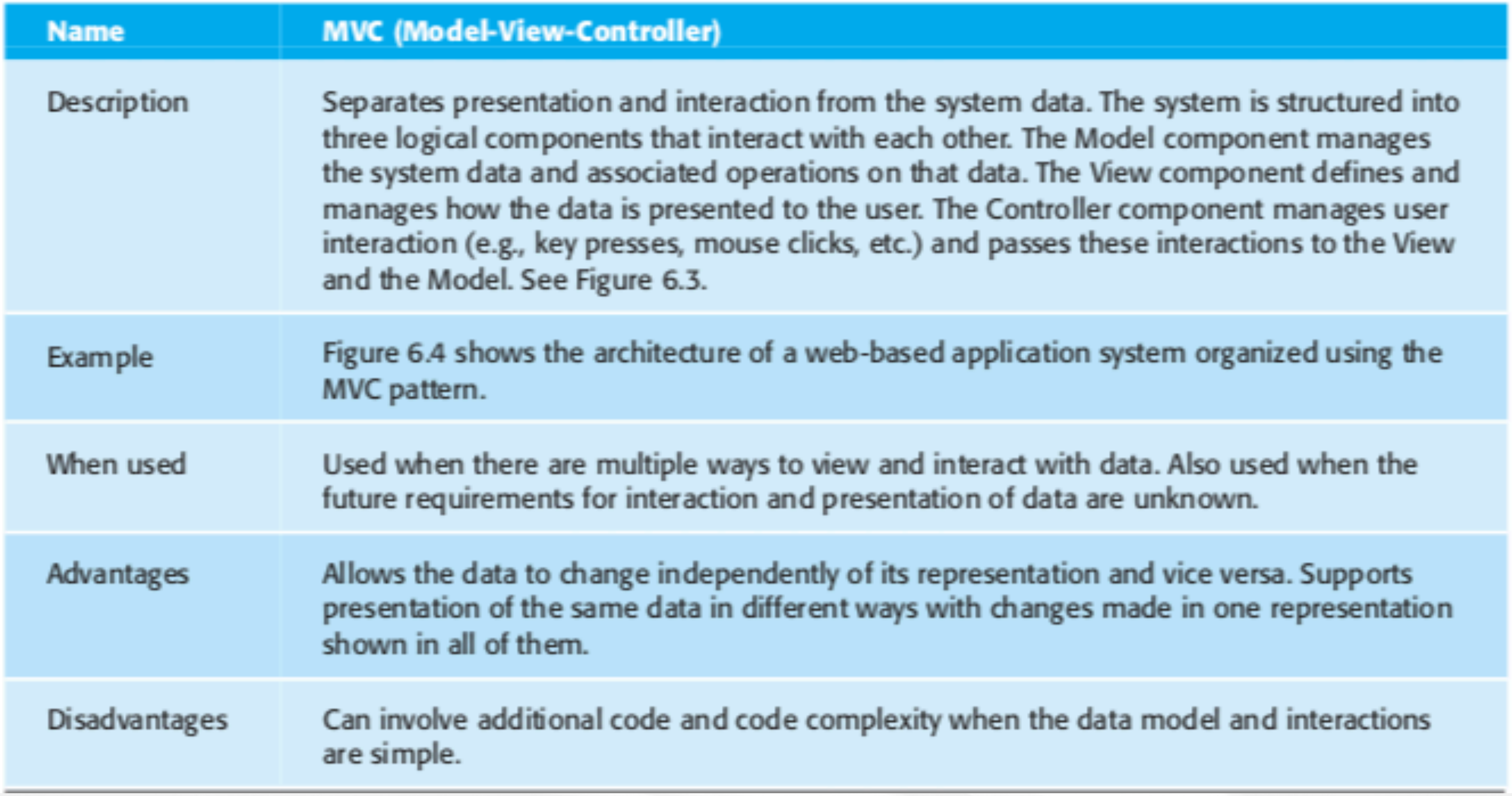
Model–View–Controller pattern

Controller

- Controllers act as an interface between Model and View components to process all the business logic, incoming requests and manipulate data.
- Acts as an intermediary between the Model and the View.
- Handles user input and updates the Model accordingly.
- Typically contains the application's flow and business logic.

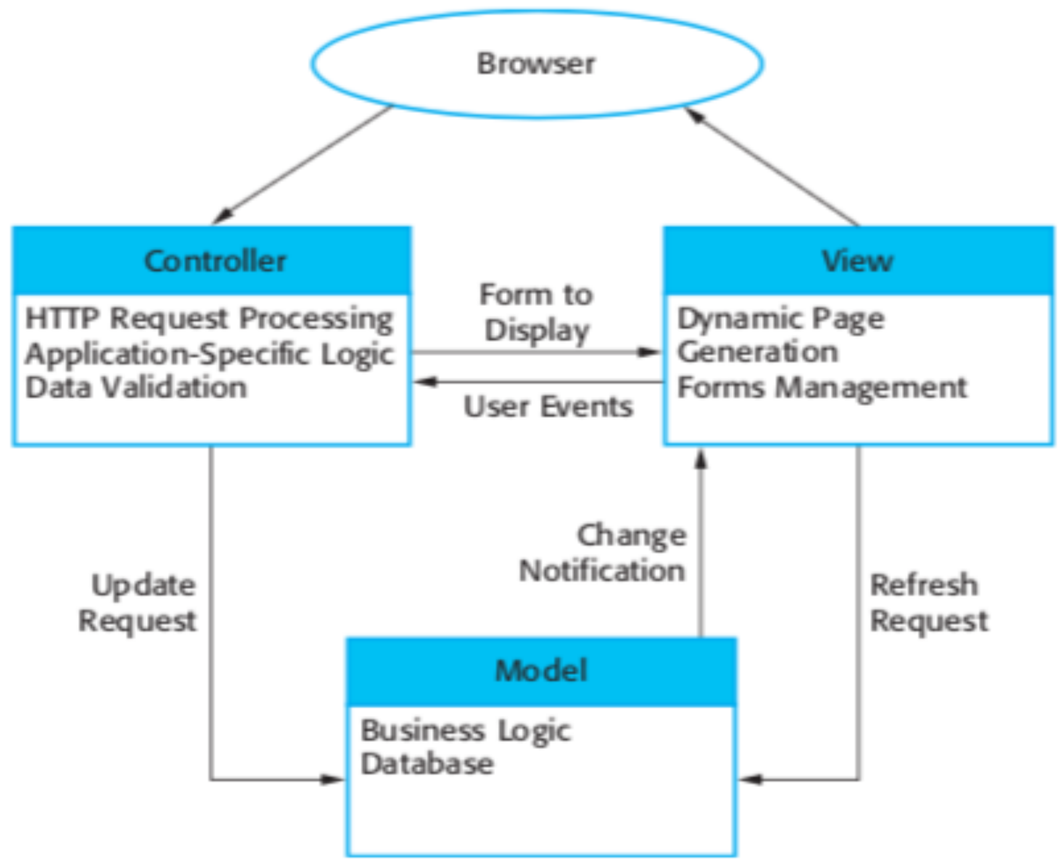
Model–View–Controller pattern





Name	MVC (Model-View-Controller)
Description	Separates presentation and interaction from the system data. The system is structured into three logical components that interact with each other. The Model component manages the system data and associated operations on that data. The View component defines and manages how the data is presented to the user. The Controller component manages user interaction (e.g., key presses, mouse clicks, etc.) and passes these interactions to the View and the Model. See Figure 6.3.
Example	Figure 6.4 shows the architecture of a web-based application system organized using the MVC pattern.
When used	Used when there are multiple ways to view and interact with data. Also used when the future requirements for interaction and presentation of data are unknown.
Advantages	Allows the data to change independently of its representation and vice versa. Supports presentation of the same data in different ways with changes made in one representation shown in all of them.
Disadvantages	Can involve additional code and code complexity when the data model and interactions are simple.

Web application architecture using the MVC pattern



Client–Server architecture

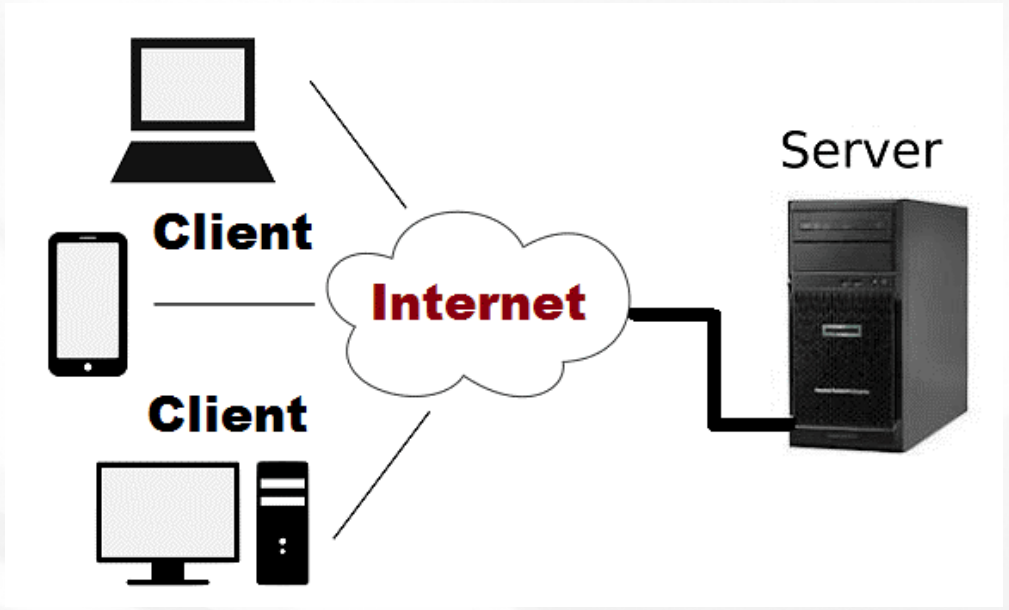
- Client/server describes the relationship between two computer programs in which one program, the **client**, makes a service request from another program, the **server**.
- The simplest form of client/server system involves a server application that is accessed directly by multiple clients, referred to as a 2–Tier architectural style.
- A system that follows the client–server pattern is organized as a set of services and associated servers, and clients that access and use the services.

Client–Server architecture

The major components of this model are:

1. A set of servers that offer services to other components.

Examples of servers include print servers that offer printing services, file servers that offer file management services, and a compile server, which offers programming language compilation services.

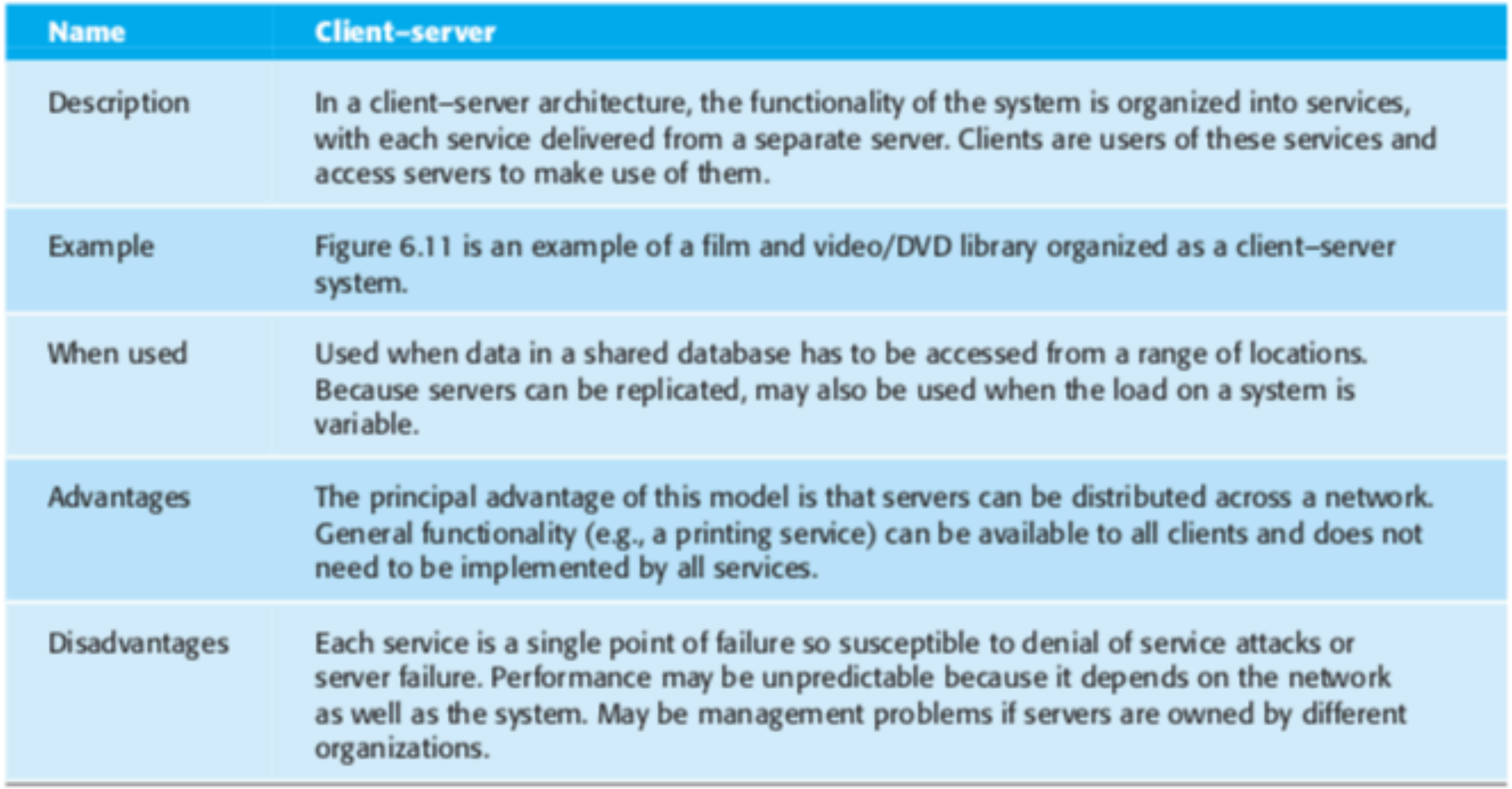


Client–Server architecture

2. A set of clients that call on the services offered by servers. There will normally be several instances of a client program executing concurrently on different computers.

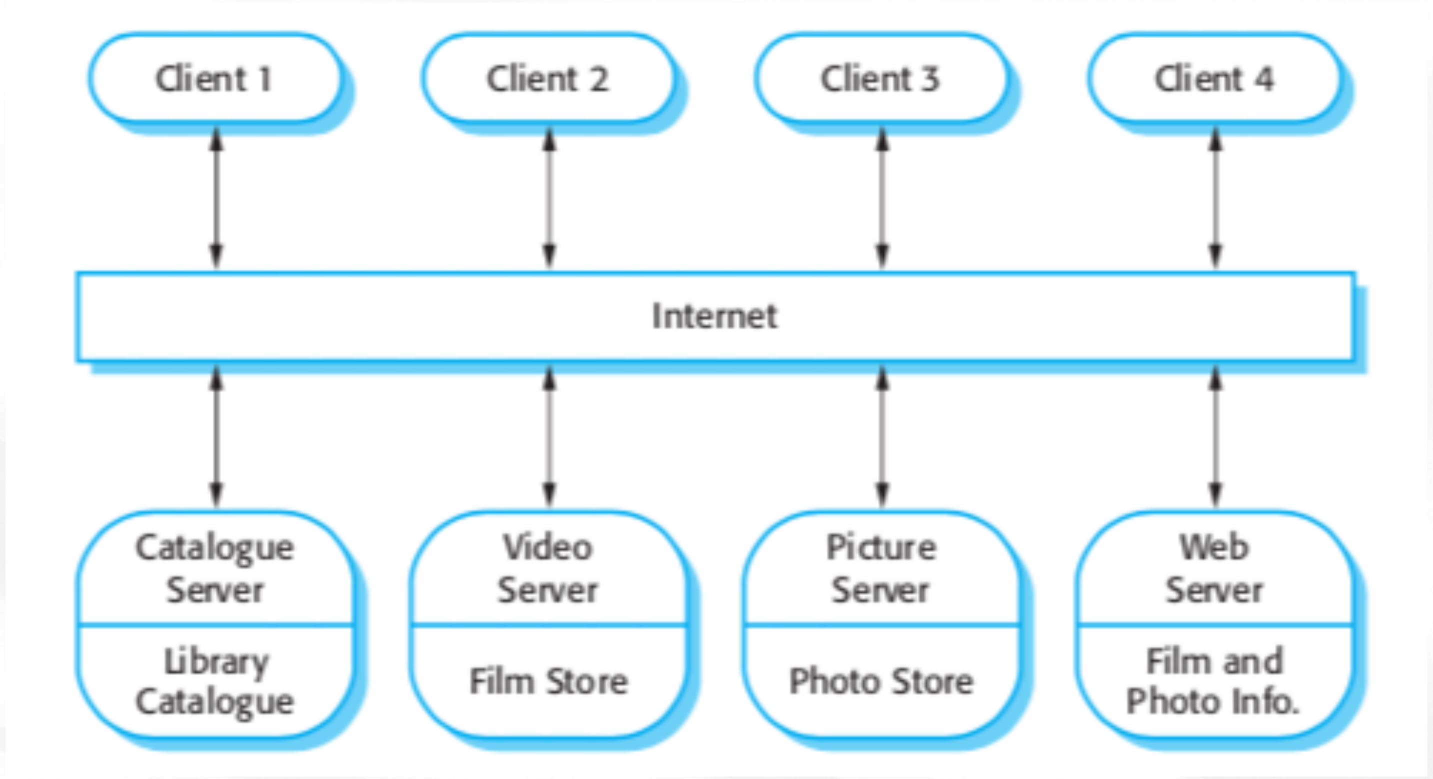
3. A network that allows the clients to access these services. Most client–server systems are implemented as distributed systems, connected using Internet protocols.

Client–Server architecture



Name	Client–server
Description	In a client–server architecture, the functionality of the system is organized into services, with each service delivered from a separate server. Clients are users of these services and access servers to make use of them.
Example	Figure 6.11 is an example of a film and video/DVD library organized as a client–server system.
When used	Used when data in a shared database has to be accessed from a range of locations. Because servers can be replicated, may also be used when the load on a system is variable.
Advantages	The principal advantage of this model is that servers can be distributed across a network. General functionality (e.g., a printing service) can be available to all clients and does not need to be implemented by all services.
Disadvantages	Each service is a single point of failure so susceptible to denial of service attacks or server failure. Performance may be unpredictable because it depends on the network as well as the system. May be management problems if servers are owned by different organizations.

An example of a system that is based on the client–server model. This is a multi–user, web–based system for providing a film and photograph library.



The End