Unit 6. Architectural Design

System Modeling {Review}

- ✧ An external perspective, where you model the context or environment of the system. (Context Model: Simple Block Diagram)
- ✧ An interaction perspective, where you model the interactions between a system and its environment, or between the components of a system. (Interaction Model: Use-case Diagram , Sequence Diagram)
- ✧ A structural perspective, where you model the organization of a system or the structure of the data that is processed by the system. (Structural Model: Class Diagram)
- ✧ A behavioral perspective, where you model the dynamic behavior of the system and how it responds to events. (Behavioral Model: State Transition Diagram)
- ✧ Data perspective, where you model the flow of data and relationship between different entities. (Data Model: Data Flow Diagram, Data Dictionary, Entity Relationship Diagram)

Unit 6: Architectural Design (6 Hrs.)

Introduction; Architectural Design Decisions; Architectural Views; Architectural Patterns; Application Architectures

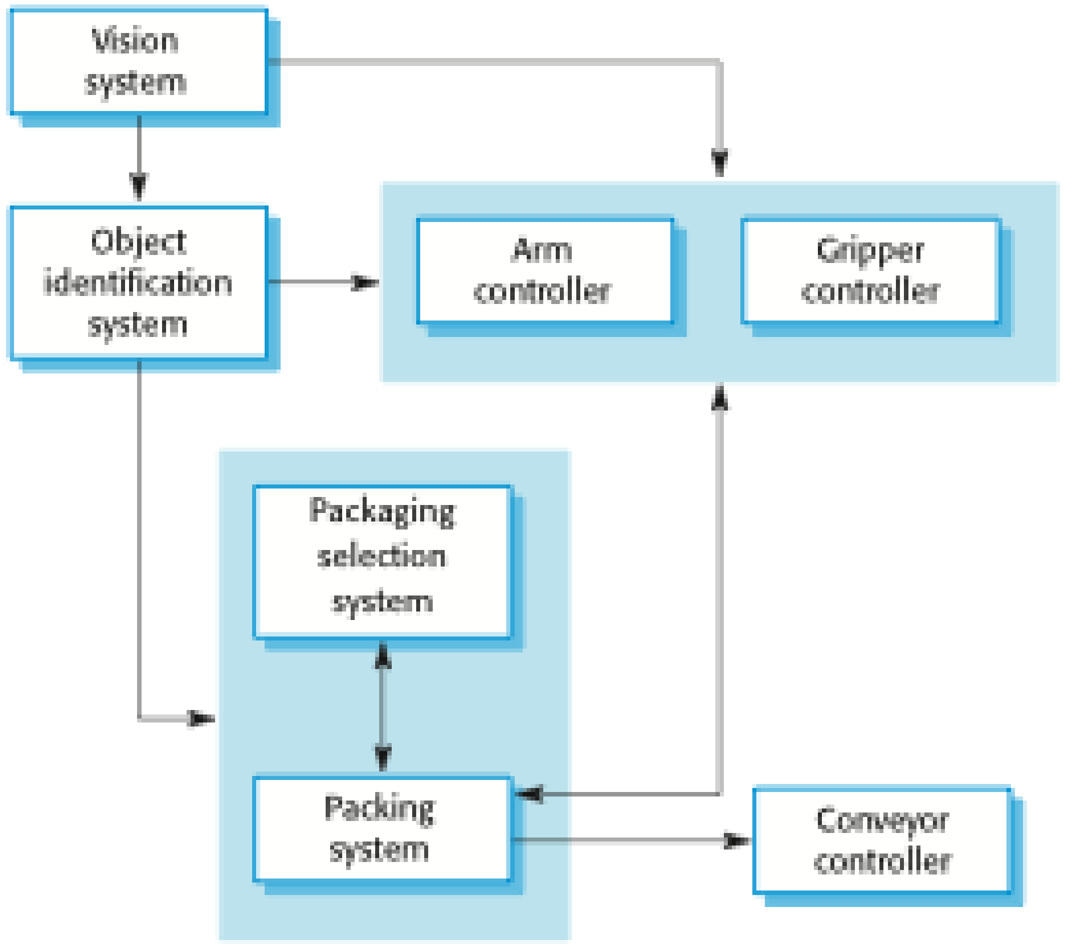
Software architecture

- The design process for identifying the sub-systems making up a system and the framework for sub-system control and communication
- The output of this design process is a description of the software architecture.

Architectural design

- ✧ An early stage of the system design process.
- ✧ Represents the link between specification and design processes.
- ✧ Often carried out in parallel with some specification activities.
- ✧ It involves identifying major system components and their communications.

The architecture of a packing robot control system



Advantages of explicit architecture

- ✧ Stakeholder communication
  - Architecture may be used as a focus of discussion by system stakeholders.
- ✧ System analysis
  - Means that analysis of whether the system can meet its non-functional requirements is possible.
- ✧ Large-scale reuse
  - The architecture may be reusable across a range of systems
  - Product-line architectures may be developed.

Architectural representations

- ✧ Simple, informal block diagrams showing components and their relationships are the most frequently used method for documenting software architectures.
- ✧ But these have been criticized because they lack semantics, do not show the types of relationships between entities nor the visible properties of entities in the architecture.
- ✧ Depends on the use of architectural models.

Architectural design decisions

- ✧ Architectural design is a creative process so the process differs depending on the type of system being developed.
- ✧ However, a number of common decisions span all design processes and these decisions affect the non-functional characteristics of the system.

Architectural design decisions

- ✧ Is there a generic application architecture that can be used?
- ✧ How will the system be distributed?
- ✧ What architectural styles are appropriate?
- ✧ What approach will be used to structure the system?
- ✧ How will the system be decomposed into modules?
- ✧ What control strategy should be used?
- ✧ How will the architectural design be evaluated?
- ✧ How should the architecture be documented?

Architecture reuse

- ✧ Systems in the same domain often have similar architectures that reflect domain concepts.
- ✧ Application product lines are built around a core architecture with variants that satisfy particular customer requirements.
- ✧ The architecture of a system may be designed around one of more architectural patterns or ‘styles’.
  - ▪ These capture the essence of an architecture and can be instantiated in different ways.

Architecture and system characteristics

- Performance
  - Localize critical operations and minimize communications.
- Security
  - Use a layered architecture with critical assets in the inner layers.
- Safety
  - Localize safety-critical features in a small number of sub-systems.
- Availability
  - Include redundant components and mechanisms for fault tolerance.
- Maintainability
  - Use fine-grain, replaceable components.

Architectural views

- ✧ What views or perspectives are useful when designing and documenting a system's architecture?
- ✧ What notations should be used for describing architectural models?
- ✧ Each architectural 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.
  - ▪ For both design and documentation, you usually need to present multiple views of the software architecture.

4 + 1 view model of software architecture

- ✧ A logical view, which shows the key abstractions in the system as objects or object classes.
- ✧ A process view, which shows how, at run-time, the system is composed of interacting processes.
- ✧ A development view, which shows how the software is decomposed for development.
- ✧ A physical view, which shows the system hardware and how software components are distributed across the processors in the system.
- ✧ Related using use cases or scenarios (+1)

Architectural patterns

- ✧ Patterns are a means of representing, sharing and reusing knowledge.
- ✧ An architectural pattern is a conventional description of good design practice, which has been tried and tested in different environments.
- ✧ Patterns should include information about when they are and when the are not useful.
- ✧ Patterns may be represented using tabular and graphical descriptions.

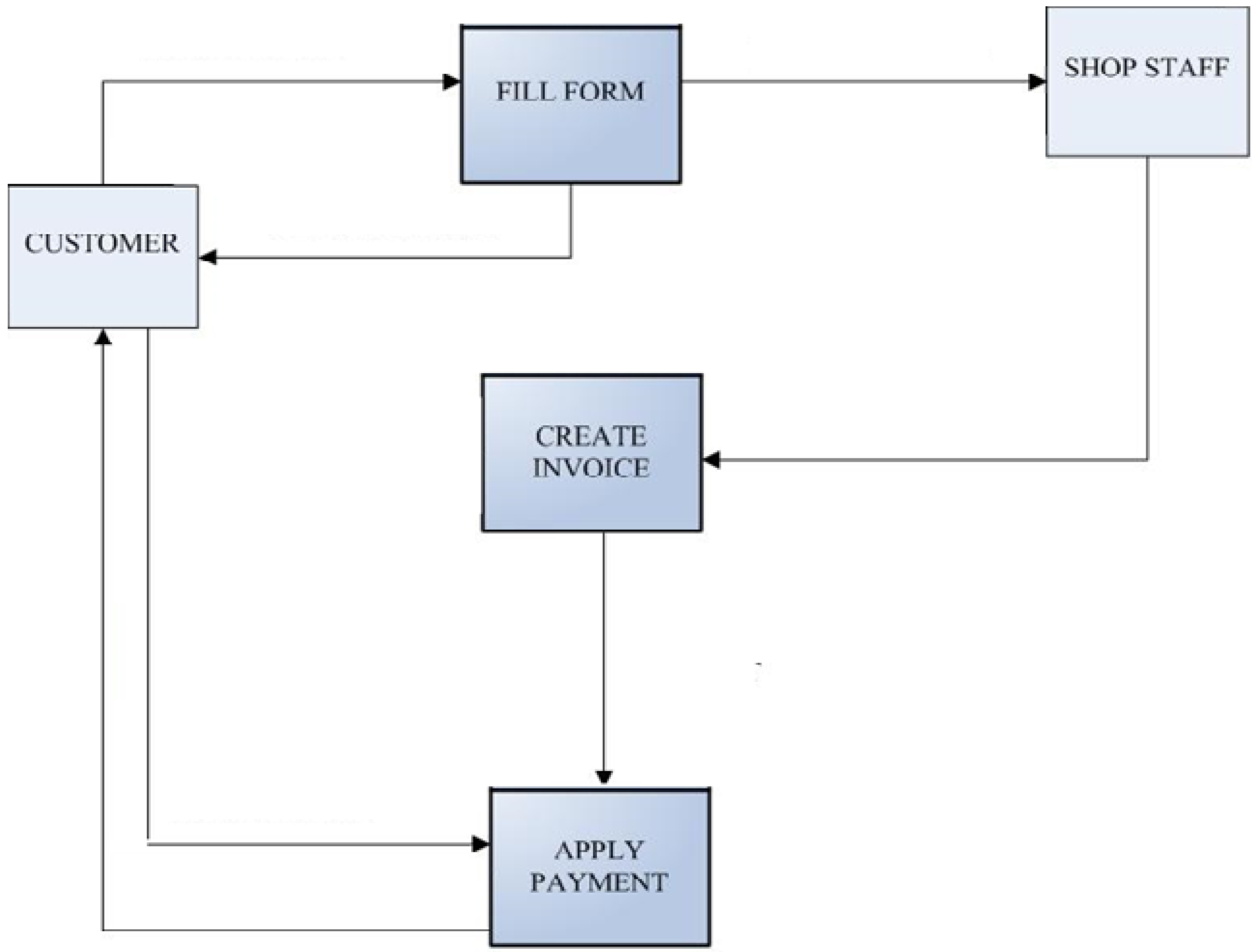
Architectural patterns

- ✧ Box and Line Diagram
- ✧ MVC
- ✧ Layered Architecture
- ✧ Repository Architecture
- ✧ Client Server Architecture
- ✧ P2P architecture

Box and line diagrams

- ✧ Very abstract - they do not show the nature of component relationships nor the externally visible properties of the sub-systems.
- ✧ However, useful for communication with stakeholders and for project planning.

Box and line diagrams



Model-View-Controller (MVC) pattern

- **Model**
  - Corresponds to all the data-related logic that the user works with
  - This can represent either the data that is being transferred between the View and Controller components or any other business logic-related data.
  - For example, a Customer object will retrieve the customer information from the database, manipulate it and update it data back to the database or use it to render data.

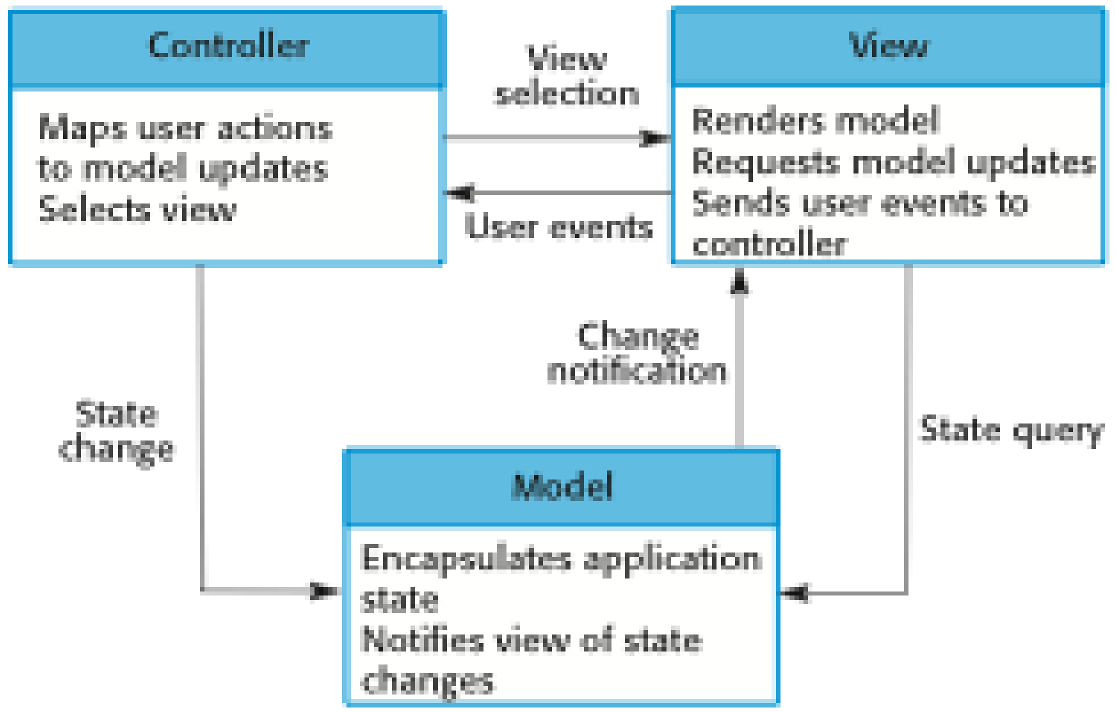
Model-View-Controller (MVC) pattern

- ✧ **View**
  - The View component is used for all the UI logic of the application.
  - For example, the Customer view will include all the UI components such as text boxes, dropdowns, etc. that the final user interacts with.
- ✧ **Controller**
  - 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 Model-View-Controller (MVC) 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 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.

The organization of the Model-View-Controller



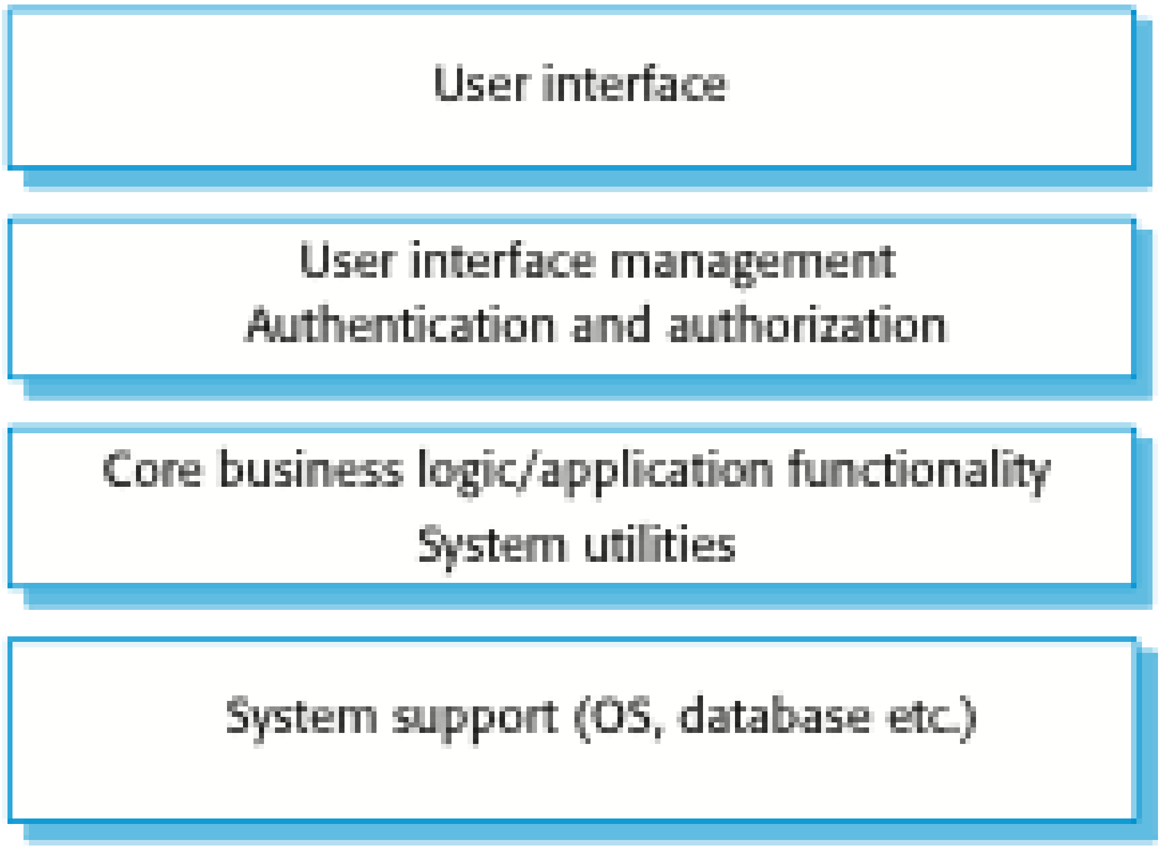
Layered architecture

- ✧ Used to model the interfacing of sub-systems.
- ✧ Organises the system into a set of layers (or abstract machines) each of which provide a set of services.
- ✧ Supports the incremental development of sub-systems in different layers. When a layer interface changes, only the adjacent layer is affected.
- ✧ However, often artificial to structure systems in this way.

The Layered architecture pattern

Name	Layered architecture
Description	Organizes the system into layers with related functionality associated with each layer. A layer provides services to the layer above it so the lowest-level layers represent core services that are likely to be used throughout the system. See Figure 6.6.
Example	A layered model of a system for sharing copyright documents held in different libraries
When used	Used when building new facilities on top of existing systems; when the development is spread across several teams with each team responsibility for a layer of functionality; when there is a requirement for multi-level security.
Advantages	Allows replacement of entire layers so long as the interface is maintained. Redundant facilities (e.g., authentication) can be provided in each layer to increase the dependability of the system.
Disadvantages	In practice, providing a clean separation between layers is often difficult and a high-level layer may have to interact directly with lower-level layers rather than through the layer immediately below it. Performance can be a problem because of multiple levels of interpretation of a service request as it is processed at each layer.

A generic layered architecture


User interface
User interface management
Authentication and authorization
Core business logic/application functionality
System utilities
System support (OS, database etc.)

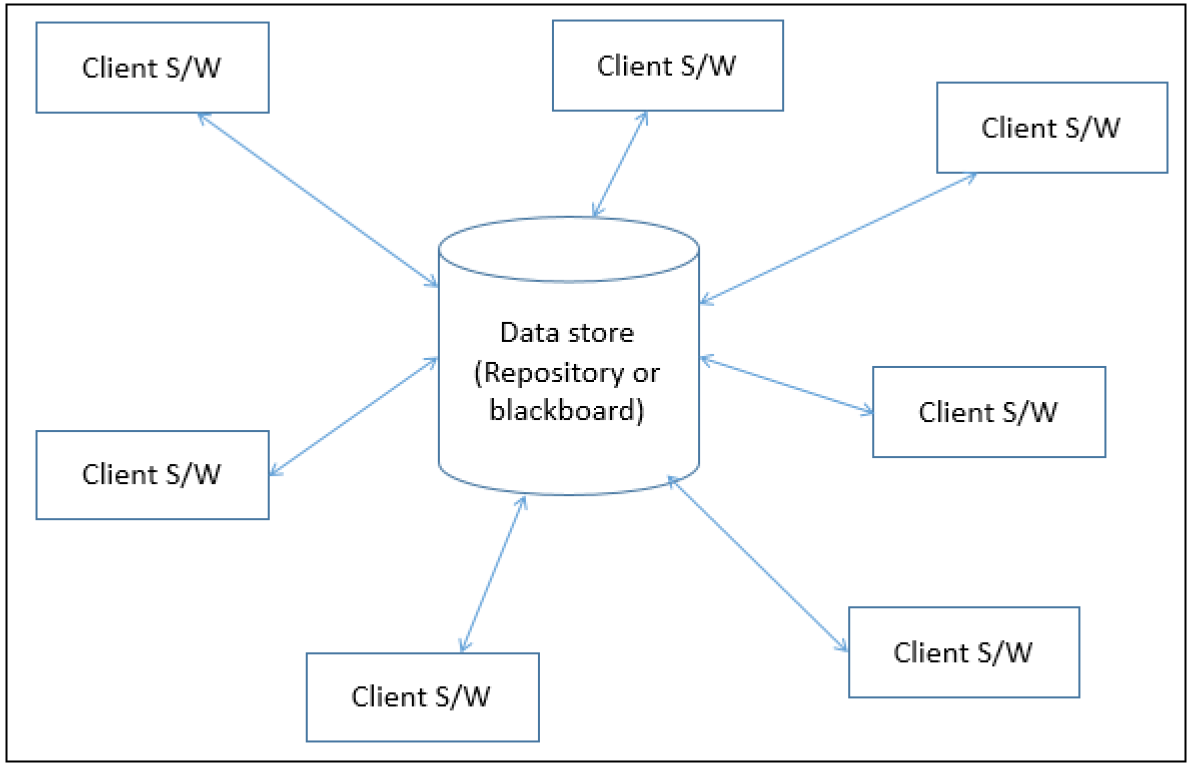
Repository architecture

- ✧ Sub-systems must exchange data. This may be done in two ways:
  - ▪ Shared data is held in a central database or repository and may be accessed by all sub-systems;
  - ▪ Each sub-system maintains its own database and passes data explicitly to other sub-systems.
- ✧ When large amounts of data are to be shared, the repository model of sharing is most commonly used a this is an efficient data sharing mechanism.

The Repository pattern

Name	Repository
Description	All data in a system is managed in a central repository that is accessible to all system components. Components do not interact directly, only through the repository.
Example	Figure in the next slide is an example of a repository of system design information.
When used	You should use this pattern when you have a system in which large volumes of information are generated that has to be stored for a long time. You may also use it in data-driven systems where the inclusion of data in the repository triggers an action or tool.
Advantages	Components can be independent—they do not need to know of the existence of other components. Changes made by one component can be propagated to all components. All data can be managed consistently (e.g., backups done at the same time) as it is all in one place.
Disadvantages	The repository is a single point of failure so problems in the repository affect the whole system. May be inefficiencies in organizing all communication through the repository. Distributing the repository across several computers may be difficult.

A repository architecture



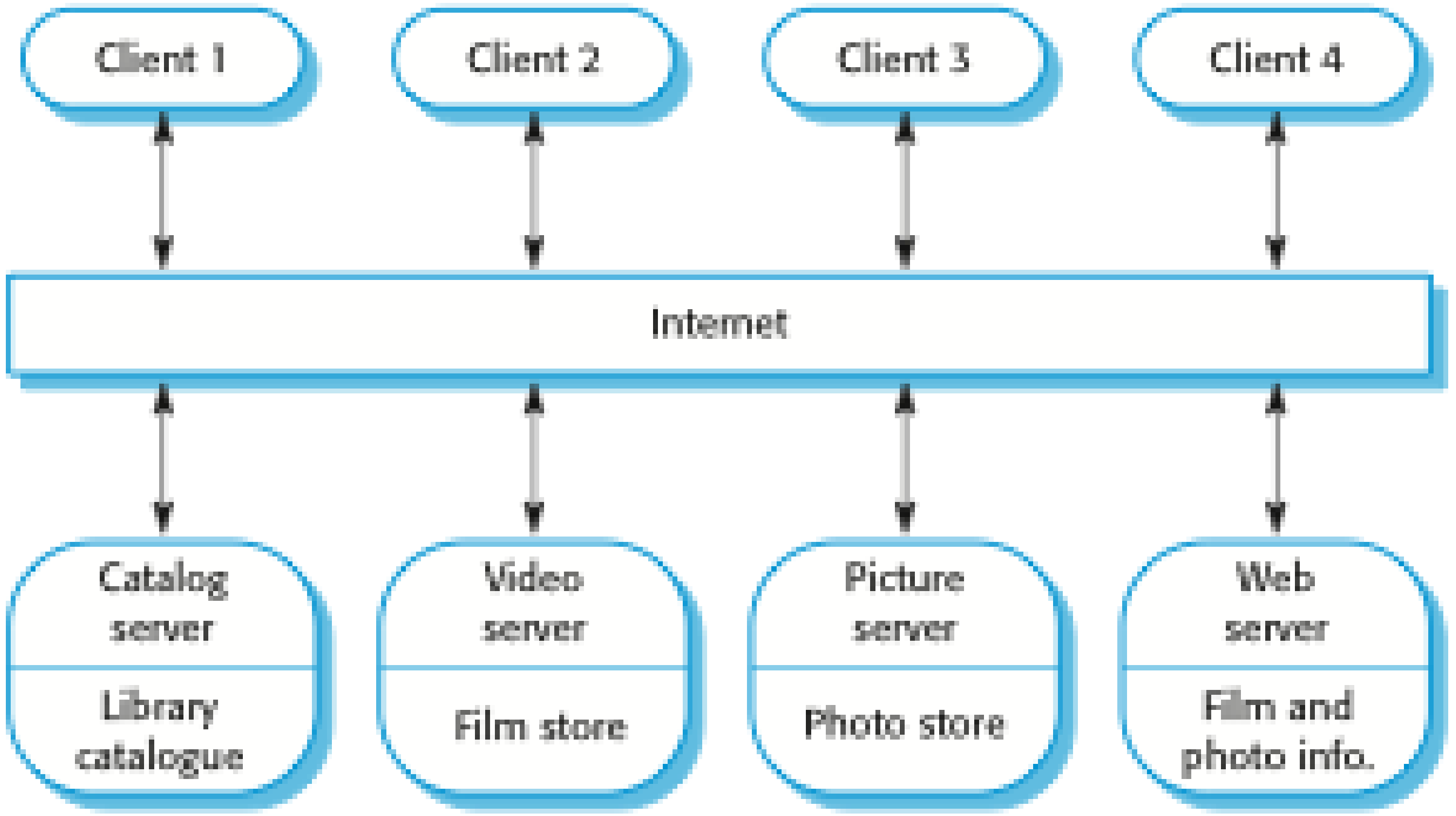
Client-server architecture

- ✧ Distributed system model which shows how data and processing is distributed across a range of components.
  - ▪ Can be implemented on a single computer.
- ✧ Set of stand-alone servers which provide specific services such as printing, data management, etc.
- ✧ Set of clients which call on these services.
- ✧ Network which allows clients to access servers.

The Client–server pattern

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	Example of a film and video/DVD library organized as a client–server system in the next slide
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.

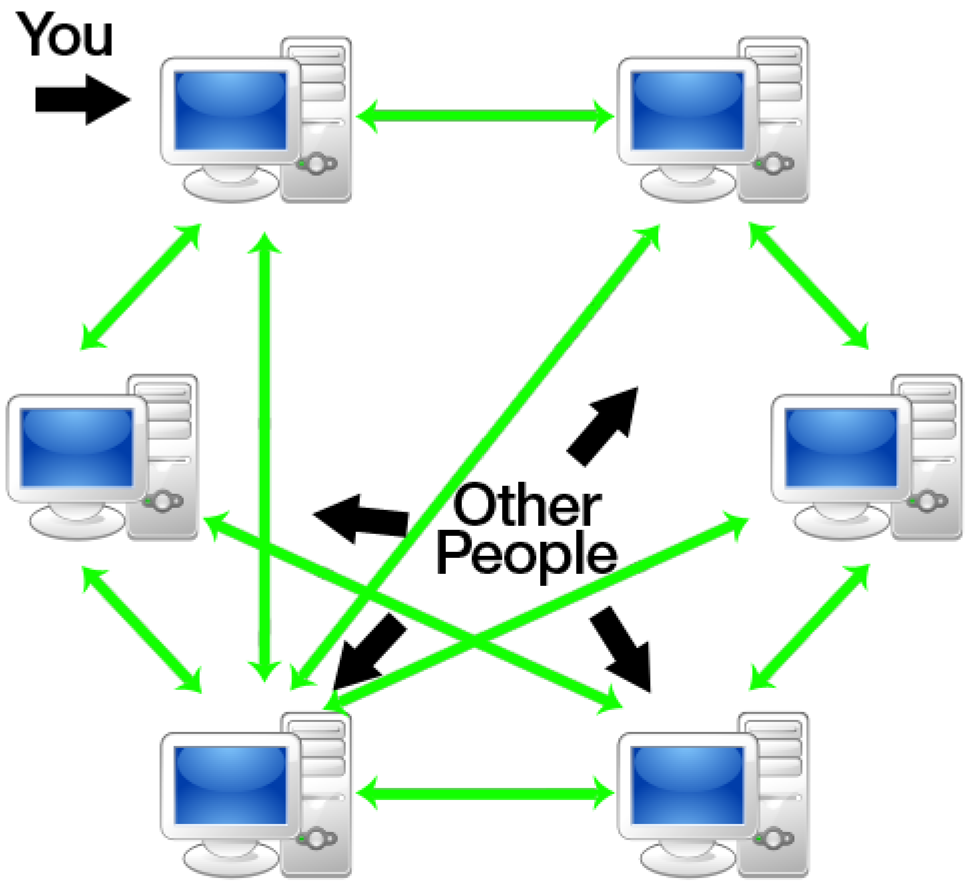
A client–server architecture for a film library



Peer-to-Peer (P2P) Architecture

- ✧ Peer-to-peer architecture (P2P architecture) is a commonly used computer networking architecture in which each workstation, or node, has the same capabilities and responsibilities
- ✧ It is often compared to the classic client/server architecture, in which some computers are dedicated to serving others.

Peer-to-Peer (P2P) Architecture



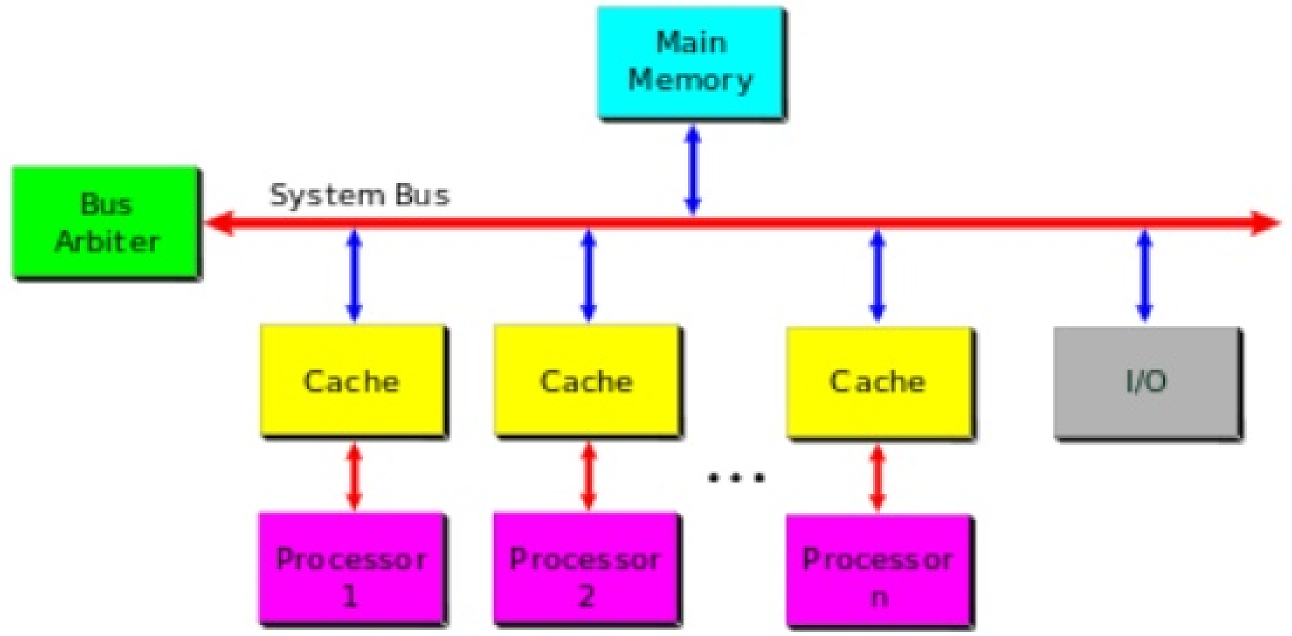
Multiprocessor Architecture

- ✧ Multiprocessor
  - ▪ A computer system in which two or more CPUs share full access to a common RAM
- ✧ In a multiprocessing system, all CPUs may be equal, or some may be reserved for special purposes.
- ✧ A combination of hardware and operating-system software design considerations determine the symmetry.
- ✧ Systems that treat all CPUs equally are called symmetric multiprocessing (SMP) systems.
- ✧ If all CPUs are not equal, system resources may be divided in a number of ways, including asymmetric multiprocessing (ASMP), non-uniform memory access (NUMA) multiprocessing, and clustered multiprocessing.

Multiprocessor Architecture

- Tightly coupled multiprocessor
  - A multiprocessor system with central shared memory is classified as shared-memory or tightly coupled multiprocessor.
  - Each processor have their cache memory along with a common global memory
- Loosely coupled multiprocessor
  - A multiprocessor system with distributed memory is classified as loosely coupled multiprocessor.
  - Each processor have their private local memory

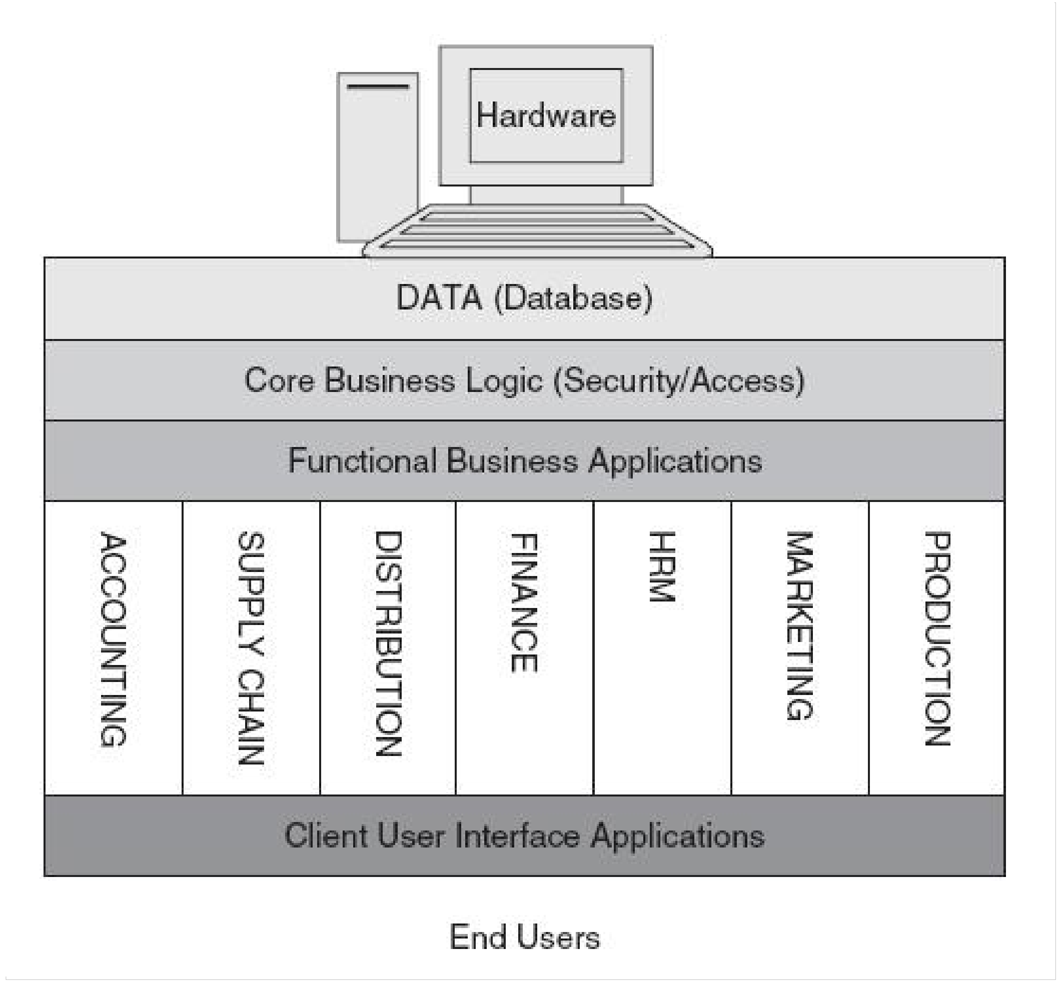
Multiprocessor Architecture



Architecture Domain

- An **architecture domain** is a broad view of an enterprise or system.
- It is a partial representation of a whole system that addresses several concerns of several stakeholders.
- It is a description that hides other views or facets of the system described
- Can be viewed as superset of
  - Business architecture
  - Data Architecture
  - Technology Architecture
  - Applications Architecture

Architecture Domain


Hardware
DATA (Database)
Core Business Logic (Security/Access)
Functional Business Applications
ACCOUNTING
SUPPLY CHAIN
DISTRIBUTION
FINANCE
HRM
MARKETING
PRODUCTION
Client User Interface Applications
End Users

Architecture Domain

- Business Architecture
  - The structure and behavior of a business system (not necessarily related to computers).
  - Covers business goals, business functions or capabilities, business processes and roles etc.
- Data Architecture
  - The data structures used by a business and/or its applications.
  - Descriptions of data stores, data groups and data items.
  - Mappings of those data artifacts to data qualities, applications, locations etc.
- Technology Architecture
  - The structure and behavior of the IT infrastructure.
  - Covers the client and server nodes of the hardware configuration, the infrastructure applications that run on them, the infrastructure services they offer to applications, the protocols and networks that connect applications and nodes.

Architecture Domain

- Application Architecture
  - The structure and behavior of applications used in a business, focused on how they interact with each other and with users.
  - In general, application architecture defines how applications interact with middleware, databases and other applications.
  - Application architectures usually follow software design principles
  - Larger software publishers, including Microsoft, typically issue application architecture guidelines to help third-party developers create applications for their platform.
  - In its case, Microsoft offers an Azure Application Architecture Guide to help developers producing cloud applications for Microsoft Azure public cloud computing platform.

Architecture Domain

Application Architecture:

- ✧ Similarly, the Object Management Group (OMG), a standards consortium of vendors, end-users, academic institutions and government agencies, operates multiple task forces to develop enterprise integration standards.
- ✧ OMG’s modeling standards include the Unified Modeling Language (UML) and the Model Driven Architecture.
- ✧ Overall, application architecture helps IT and business planners work together so that the right technical solutions are available to meet the business objectives. More specifically, application architecture:

Architecture Domain

Application Architecture:

- ✧ Reduces cost by identifying redundancies, such as the use of two separate databases that can be replaced by one;
- ✧ Improves efficiency by identifying gaps, such as where one application cannot work with another or where mobile users cannot access essential services need;
- ✧ Makes an enterprise platform for accessible and attractive to third-party developers;
- ✧ Produces interoperable, modular systems that are easier to use and maintain;
- ✧ Helps architect see the big picture and map that against the organization's objectives.