# SOFTWARE ENGINEERING

## CO3001

## CHAPTER 7 – ARCHITECTURE DESIGN

Anh Nguyen-Duc
Tho Quan-Thanh
Thai-Minh Truong

WEEK 7


Adapted from https://iansommerville.com/software-engineering-book/slides/

# TOPICS COVERED

- ✓ Architectural thinking
- ✓ Architectural patterns
- ✓ Architectural views
- ✓ Application architectures

# ARCHITECTURAL THINKING

# SOFTWARE ARCHITECTURE

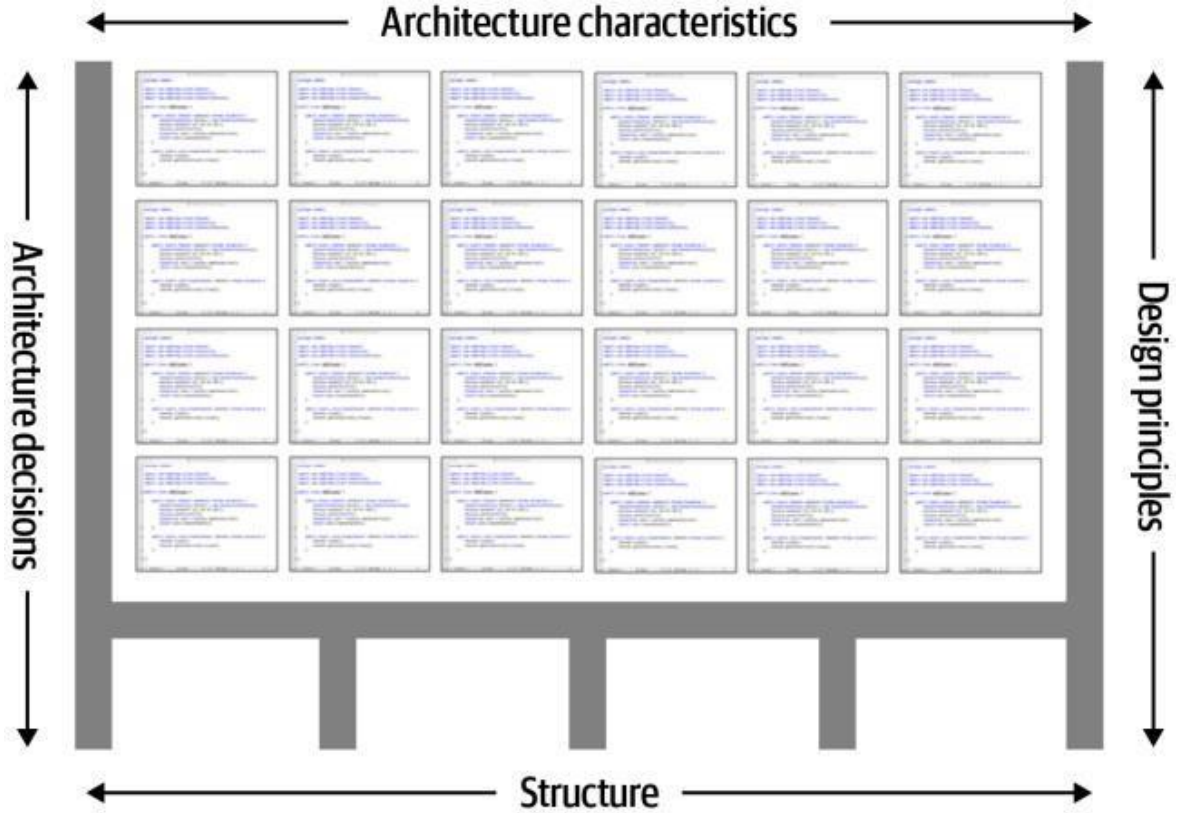
- ✓ Describes how the system is organized as a set of communicating components

Architecture?
"Architecture" can mean: (http://en.wikipedia.org/wiki/Architecture)
*A general term to describe buildings and other physical structures.*
*The art and science of designing buildings and (some) non-building structures.*
*The style of design and method of construction of buildings and other physical structures.*
…

# DEFINING SOFTWARE ARCHITECTURE

*Four dimensions that define software architecture*

- Software architecture consists of the structure of the system, combined with architecture characteristics ("-ilities") the system must support, architecture decisions, and finally design principles.

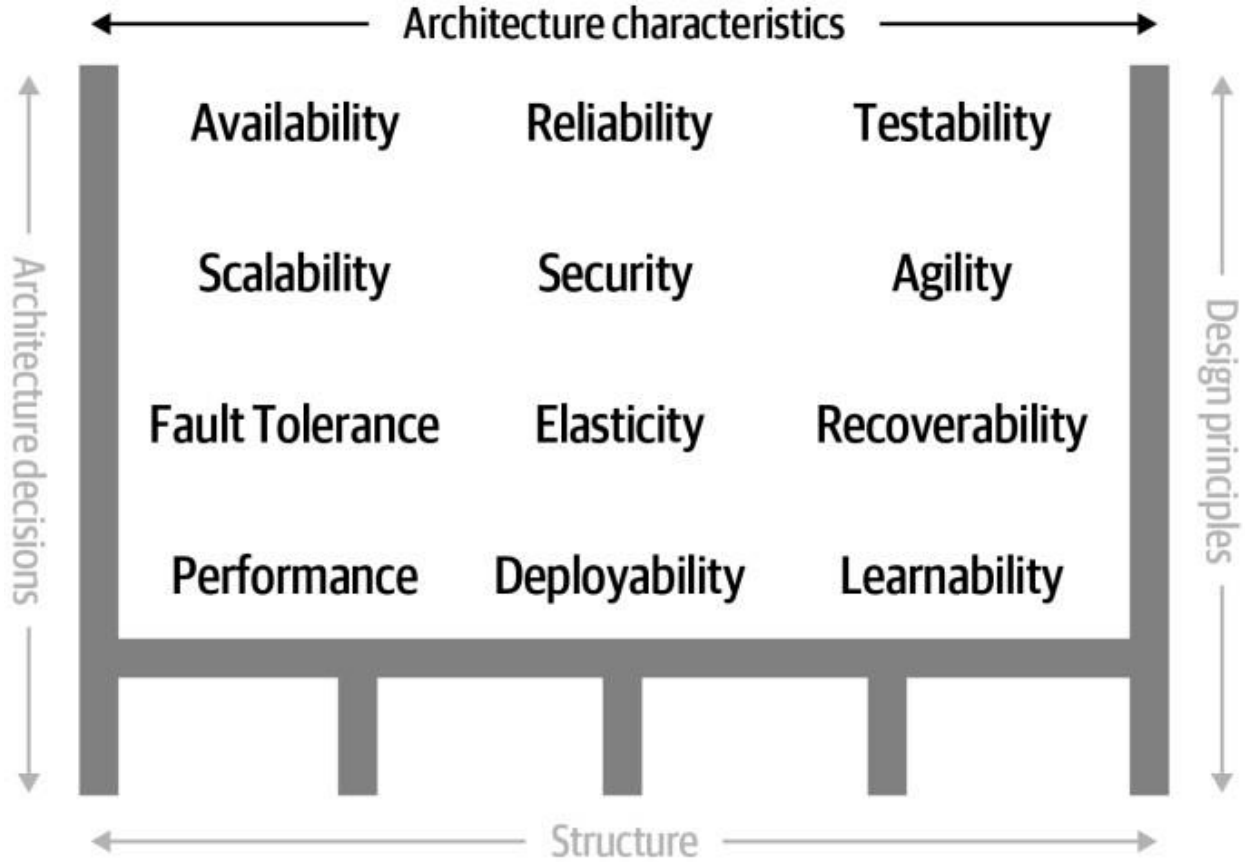



Richards et al., Fundamentals of Software Architecture, 2020 [4]

# DEFINING SOFTWARE ARCHITECTURE

## *Architecture characteristics*

- The architecture characteristics define the success criteria of a system, which is generally orthogonal to the functionality of the system.

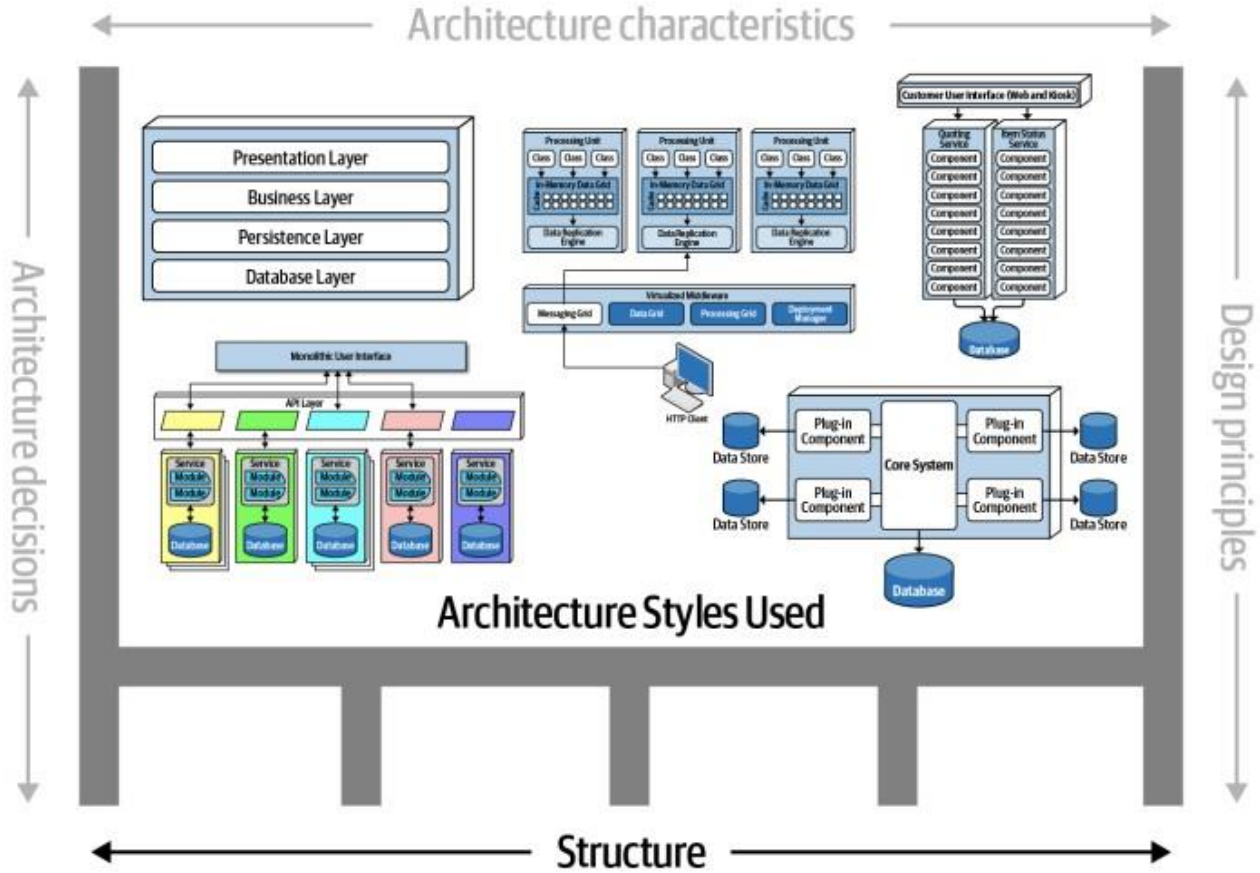



Richards et al., Fundamentals of Software Architecture, 2020 [4]

# DEFINING SOFTWARE ARCHITECTURE

## *Structure of the system*

- Structure refers to the type of architecture styles used in the system such as microservices, layered, or microkernel

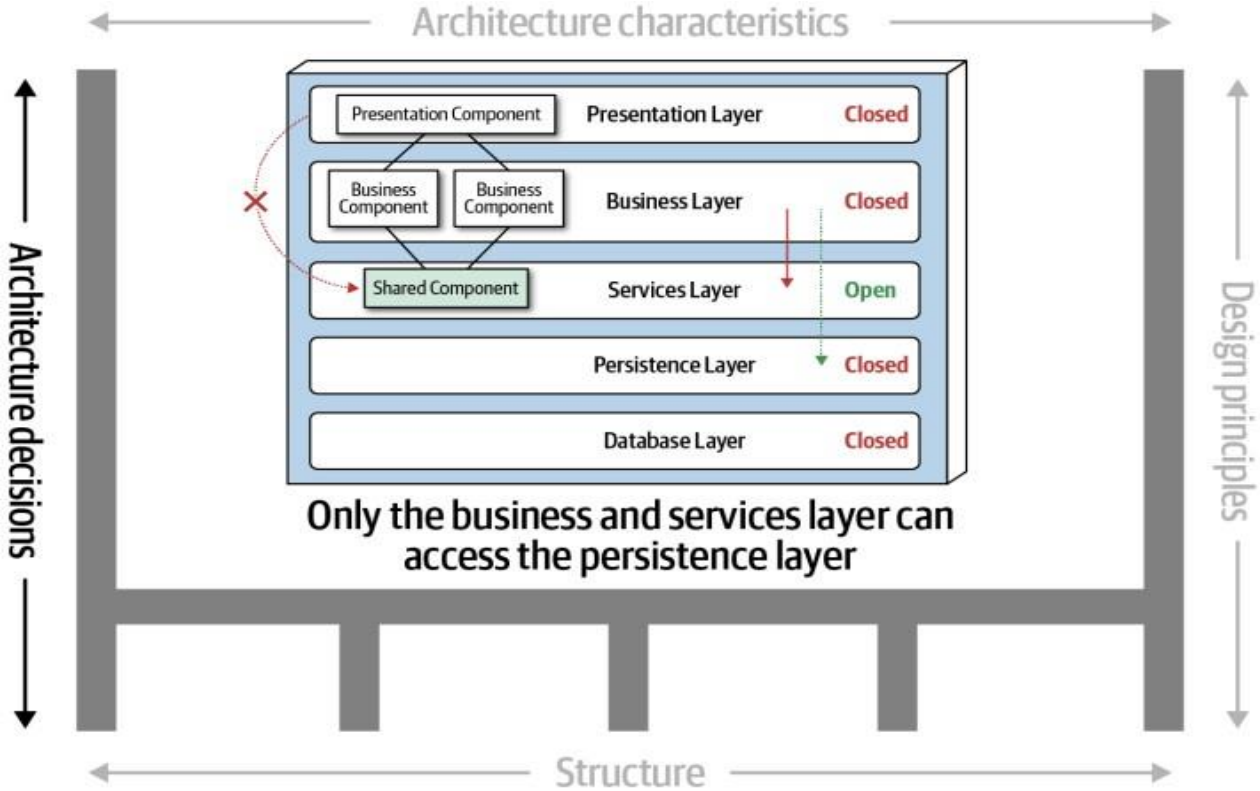



Richards et al., Fundamentals of Software Architecture, 2020 [4]

# DEFINING SOFTWARE ARCHITECTURE

## *Architecture decisions*

- Architecture decisions define the rules for how a system should be constructed. Architecture decisions form the constraints of the system and direct the development teams on what is and what isn't allowed.



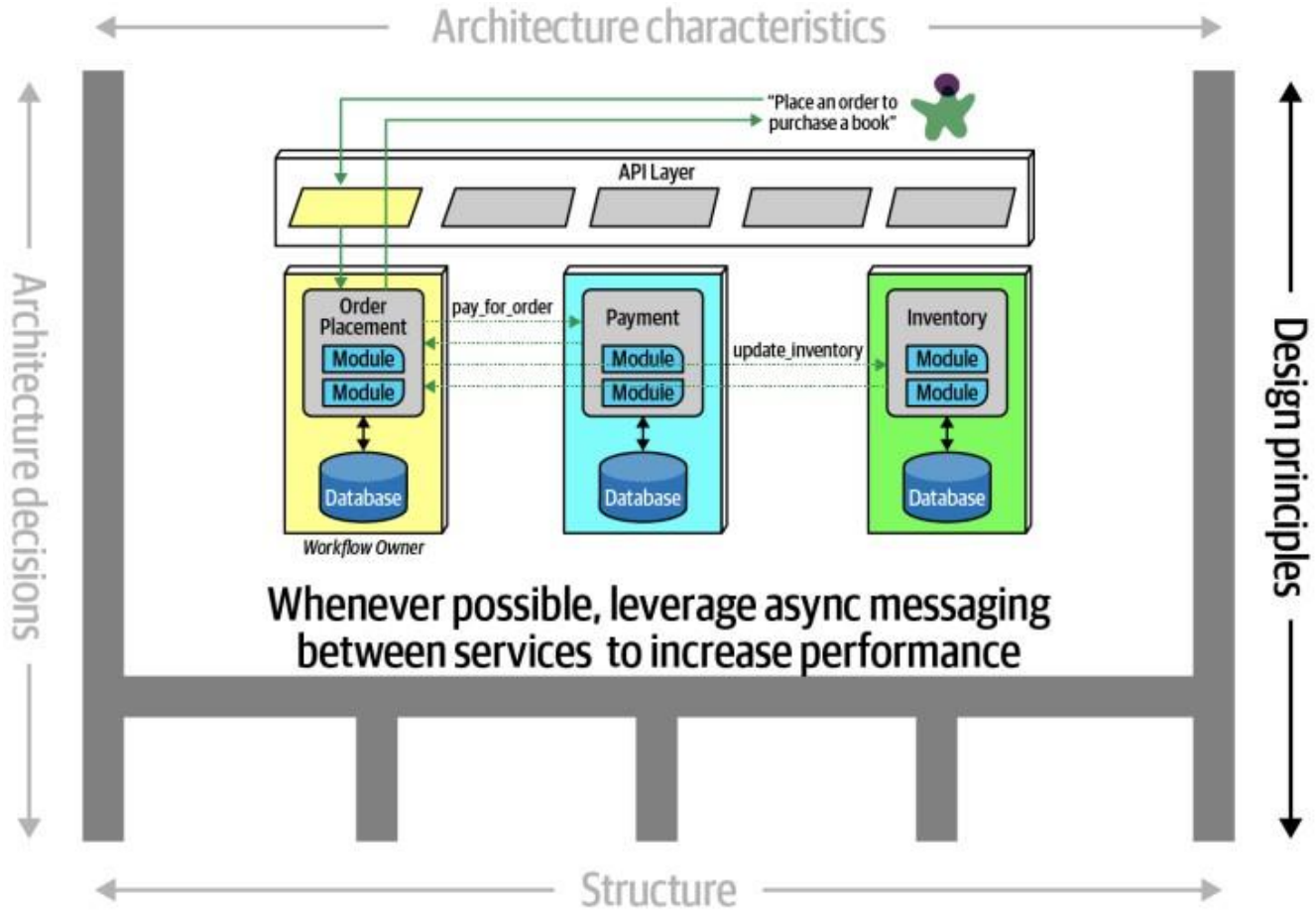

For example, an architect might make an architecture decision that only the business and services layers within a layered architecture can access the persistence layer, restricting the presentation layer from making direct database calls.

Richards et al., Fundamentals of Software Architecture, 2020 [4]

# DEFINING SOFTWARE ARCHITECTURE

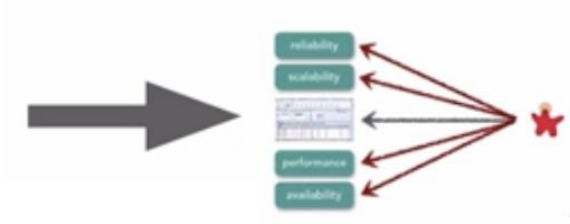
## *Design principles*

- Guidelines for constructing systems



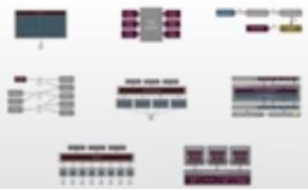

Richards et al., Fundamentals of Software Architecture, 2020 [4]


Problem description




Architecture characteristics

Logical architecture



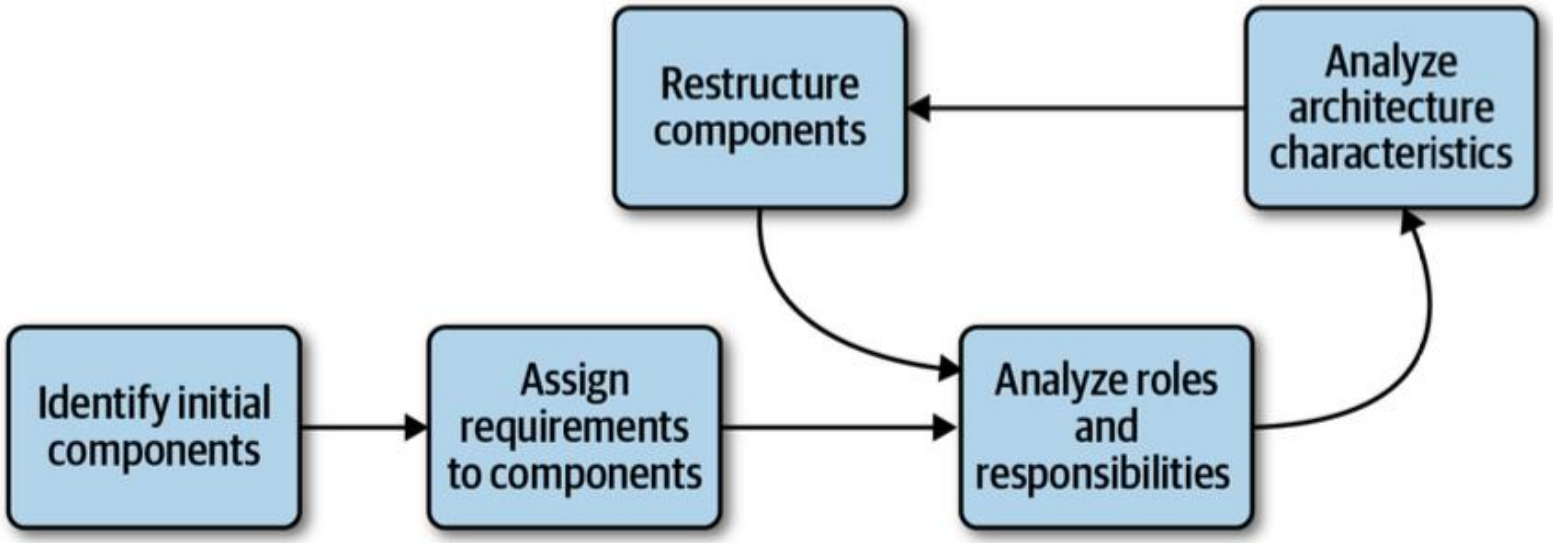
Physical architecture

Documenting architecture

Presenting the solution

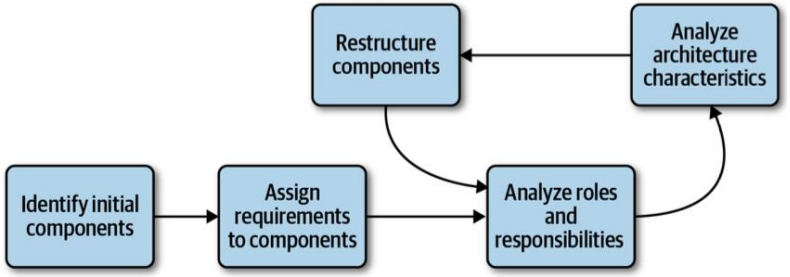
# COMPONENT-BASED THINKING

## Component Identification Flow




Component identification cycle

# COMPONENT-BASED THINKING

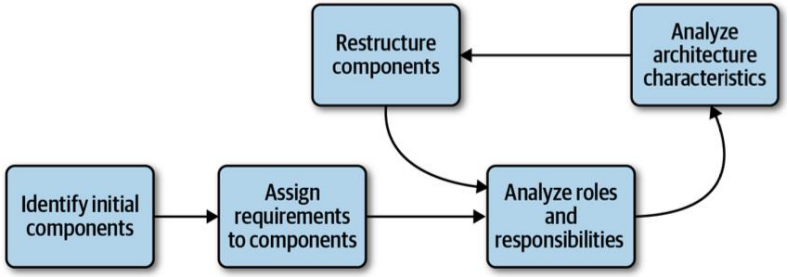



## Component Identification Flow

- **Identifying Initial Components:**
  - Determine what top-level components to begin with, based on what type of top-level partitioning chosen.
  - The likelihood of achieving a good design from this initial set of components is disparagingly small, which is why architects must iterate on component design to improve it.
- **Assign Requirements to Components:**
  - Align requirements (or user stories) to the initial components to see how well they fit.
  - This may entail creating new components, consolidating existing ones, or breaking components apart because they have too much responsibility.

# COMPONENT-BASED THINKING

## Component Identification Flow




- **Analyze Roles and Responsibilities:**
  - Look at the roles and responsibilities elucidated during the requirements to make sure that the granularity matches.
  - Thinking about both roles and behaviors the application must support allows the architect to align the component and domain granularity.
- **Analyze Architecture Characteristics:**
  - When assigning requirements to components, the architect should also look at the architecture characteristics discovered earlier in order to think about how they might impact component division and granularity.
- **Restructure Components:**
  - Architects must continually iterate on their component design with developers.
  - an iterative approach to component design is key.

# COMPONENT-BASED THINKING

## Logical vs Physical Architecture

- ✓ But how? Choose an **architecture style**

### Architecture Styles Worksheet

System/Project: _____

Architect/Team: _____ Date: _____

Selected Architecture(s):

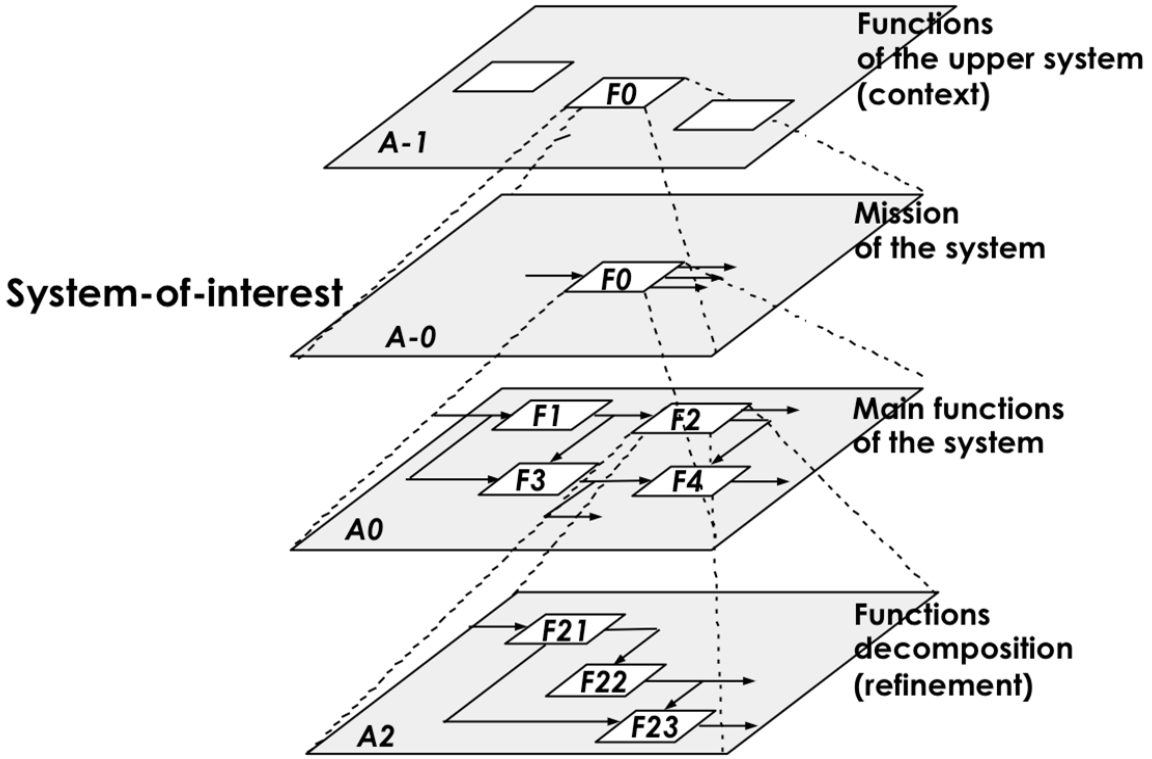
| | layered | modular monolith | microkernel | microservices | service-based | service-oriented | event-driven | space-based |
|---|---|---|---|---|---|---|---|---|
| partitioning | technical | domain | domain | domain | domain | technical | technical | technical |
| cost | $ | $ | $ | $$$$$ | $$ | $$$$ | $$$ | $$$$ |
| maintainability | ★ | ★★ | ★★★ | ★★★★★ | ★★★★ | ★ | ★★★ | ★★★ |
| testability | ★★ | ★★ | ★★★ | ★★★★★ | ★★★★ | ★ | ★★ | ★ |
| deployability | ★ | ★★ | ★★★ | ★★★★★ | ★★★★ | ★ | ★★★ | ★★★ |
| simplicity | ★★★★★ | ★★★★★ | ★★★★ | ★ | ★★★ | ★ | ★★ | ★ |
| scalability | ★ | ★ | ★ | ★★★★★ | ★★★ | ★★★ | ★★★★ | ★★★★★ |
| elasticity | ★ | ★ | ★ | ★★★★ | ★★ | ★★★ | ★★★ | ★★★★★ |
| responsiveness | ★★★ | ★★★ | ★★★ | ★★ | ★★★ | ★★ | ★★★★★ | ★★★★★ |
| fault-tolerance | ★ | ★ | ★ | ★★★★★ | ★★★★ | ★★★ | ★★★★★ | ★★★ |
| evolvability | ★ | ★ | ★★★ | ★★★★★ | ★★★★ | ★ | ★★★★★ | ★★★ |
| abstraction | ★ | ★ | ★★★ | ★ | ★ | ★★★★★ | ★★★★ | ★ |
| interoperability | ★ | ★ | ★★★ | ★★★ | ★★ | ★★★★★ | ★★★ | ★★ |

# ARCHITECTURE DECOMPOSITION

- ✓ Software systems:
  - complexity problem <= inter-relationship
- ✓ Goals:
  - Maximizing cohesion
  - Minimizing coupling

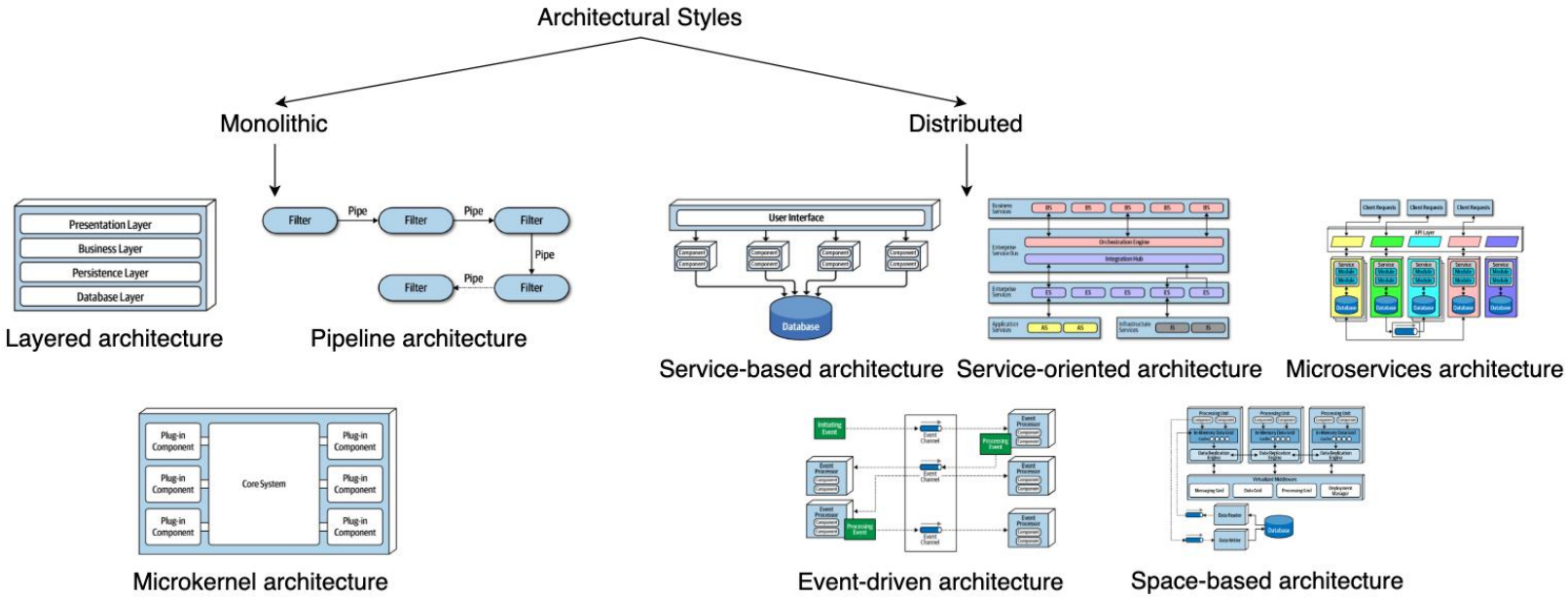
*Cohesion*: degree of communication taken place among the module's elements

*Coupling*: degree of communication among modules



# MONOLITHIC VERSUS DISTRIBUTED ARCHITECTURES

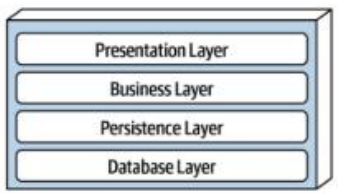
- ✓ Architecture styles can be classified into two main types:
  - ***Monolithic*** (single deployment unit of all code)
  - ***Distributed*** (multiple deployment units connected through remote access protocols).



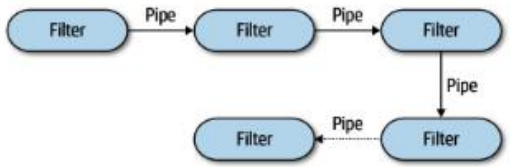
Architectural Styles

Monolithic

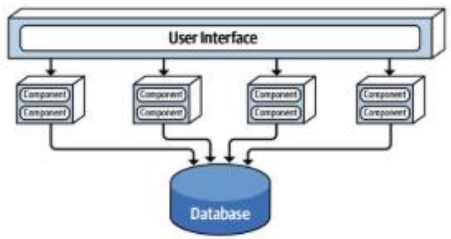
Distributed



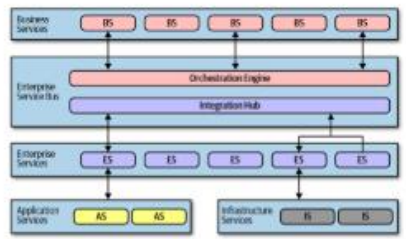

Layered architecture



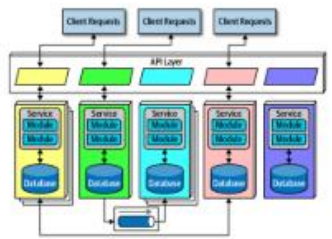

Pipeline architecture



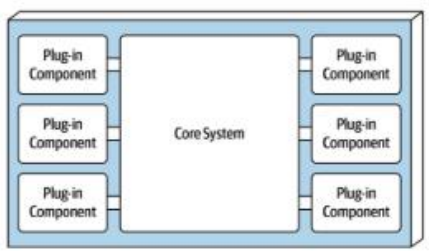

Service-based architecture



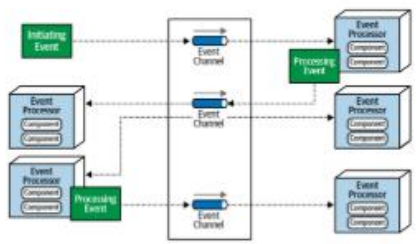
Service-oriented architecture



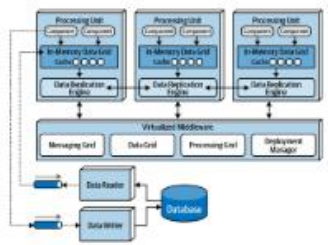
Microservices architecture




Microkernel architecture



Event-driven architecture



Space-based architecture

# ARCHITECTURAL PATTERNS

# ARCHITECTURAL PATTERNS

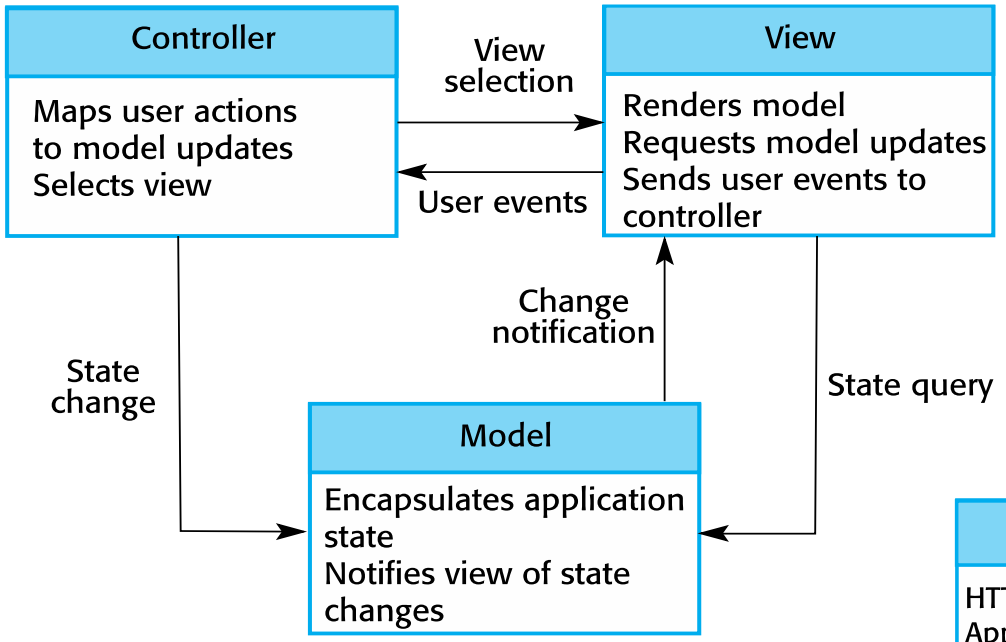
- ✓ Patterns are a means of representing, sharing and reusing knowledge.
  - Patterns should include information about when they are and when they are not useful.
  - Patterns may be represented using tabular and graphical descriptions.

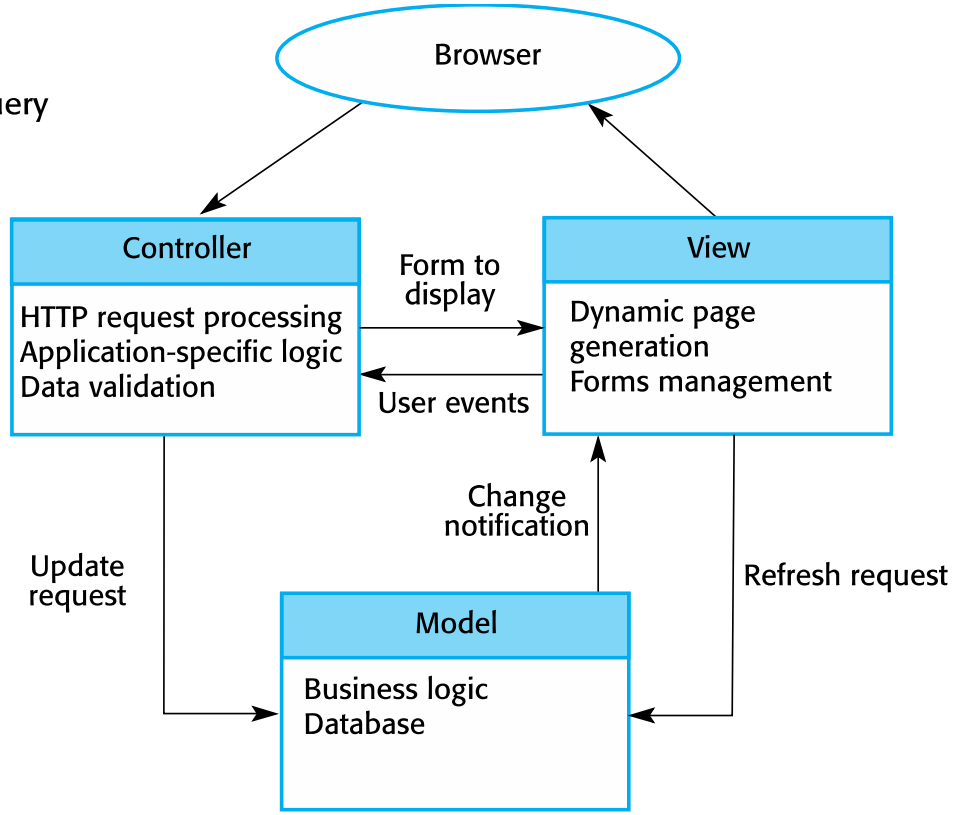
- ✓ An architectural pattern is a stylized description of good design practice, which has been tried and tested in different environments.

# 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. |
| **Example** | The next slide 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. |

# CONCEPTUAL VIEW OF THE MVC AND WEB-BASED MVC



# (ADDITIONAL READING)

READ AND APPLY THE CODE IN
HTTPS://WWW.JAVATPOINT.COM/MVC-ARCHITECTURE-IN-JAVA

TO VIEW ALL MCPS FROM THE BACK OFFICERS' VIEW

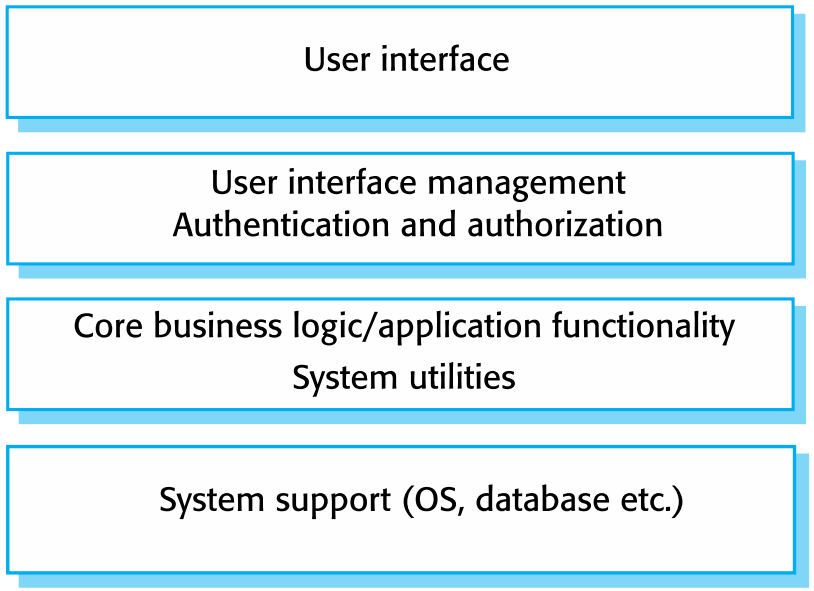
HTTPS://GITHUB.COM/ANHN/CO3001_DESIGN_LECTURE

# THE LAYERED ARCHITECTURE PATTERN

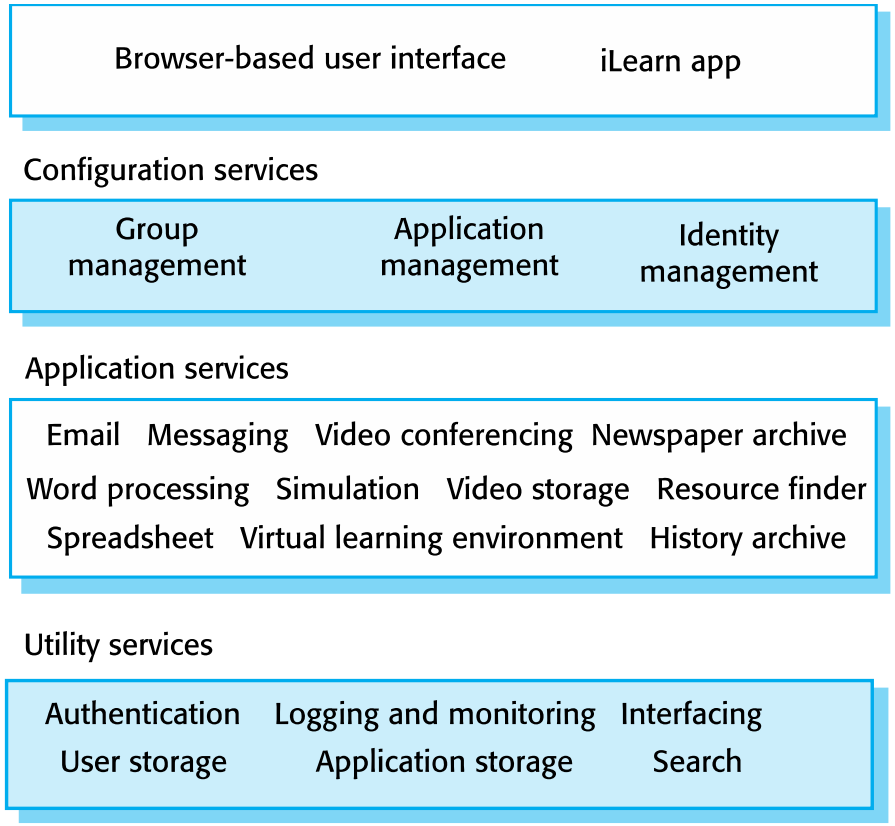
| Name | Layered architecture |
|---|---|
| **Description** | Organizes the system into layers with related functionality associated with each layer. A layer provides services to the layer above it so the lowest-level layers represent core services that are likely to be used throughout the system. See 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

A generic layered architecture




The architecture of the iLearn system



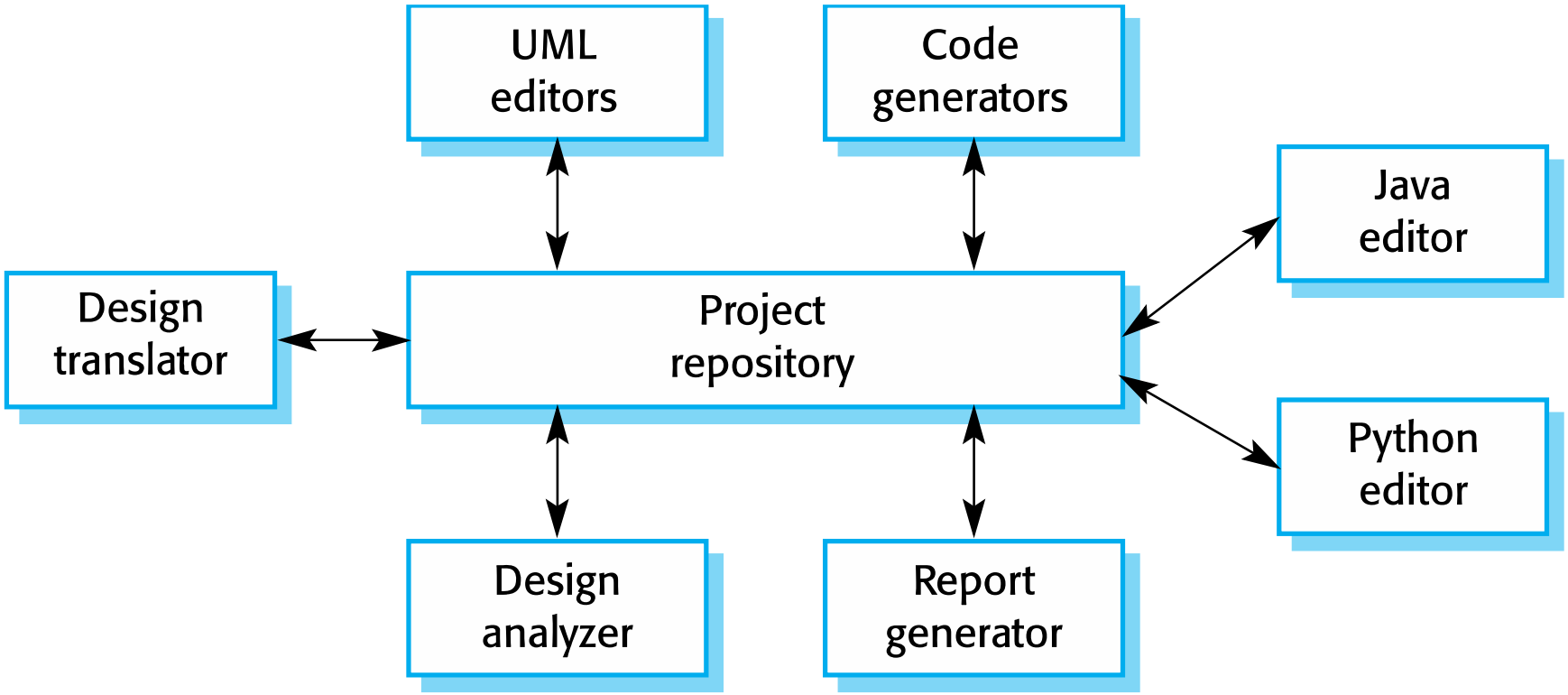
# REPOSITORY ARCHITECTURE

- ✓ 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** | The next is an example of an IDE where the components use a repository of system design information. Each software tool generates information which is then available for use by other tools. |
| **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. |

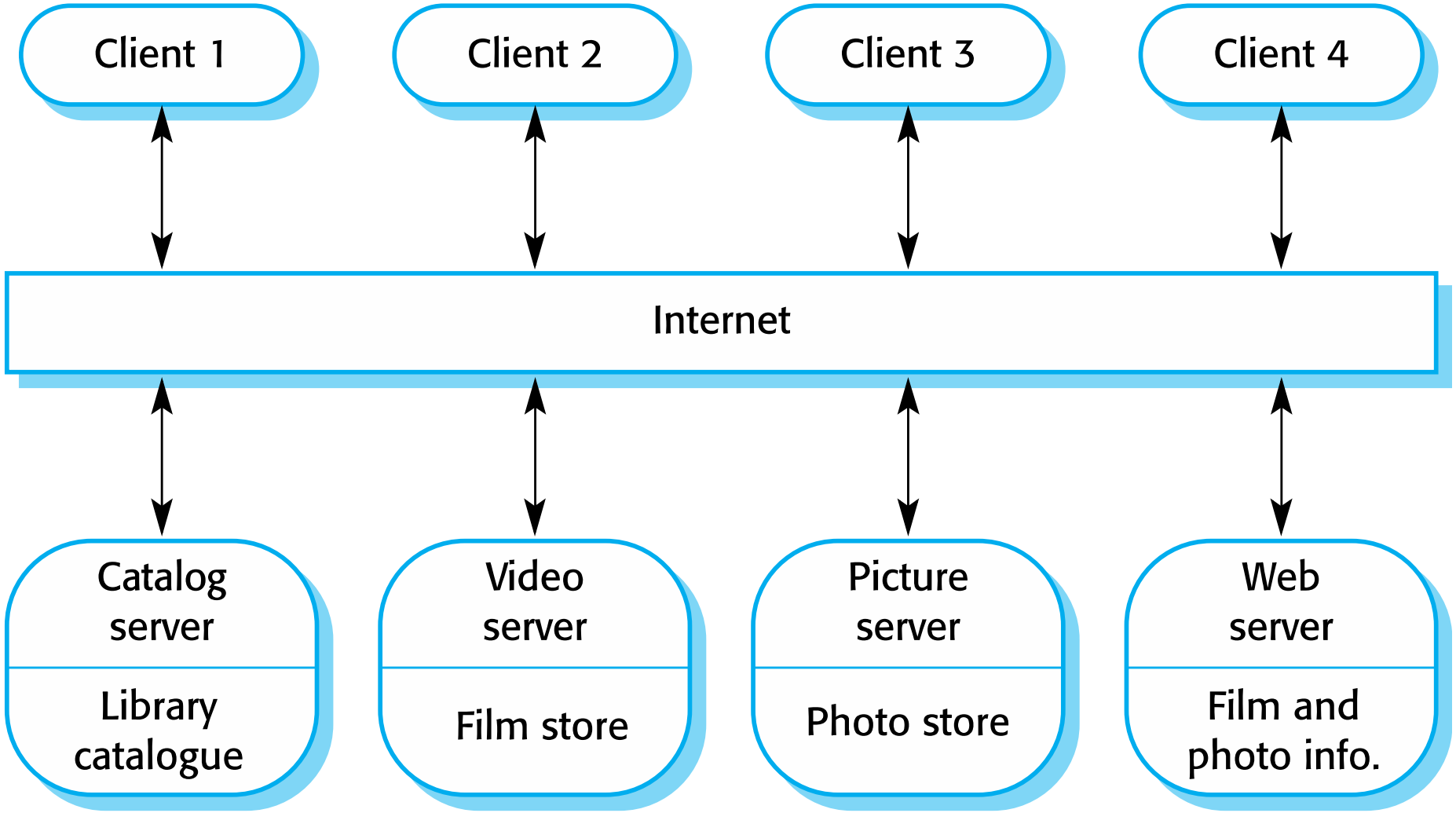
# A REPOSITORY ARCHITECTURE FOR AN IDE (ADDITIONAL READING)



# 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** | Figure in the next slide 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. |

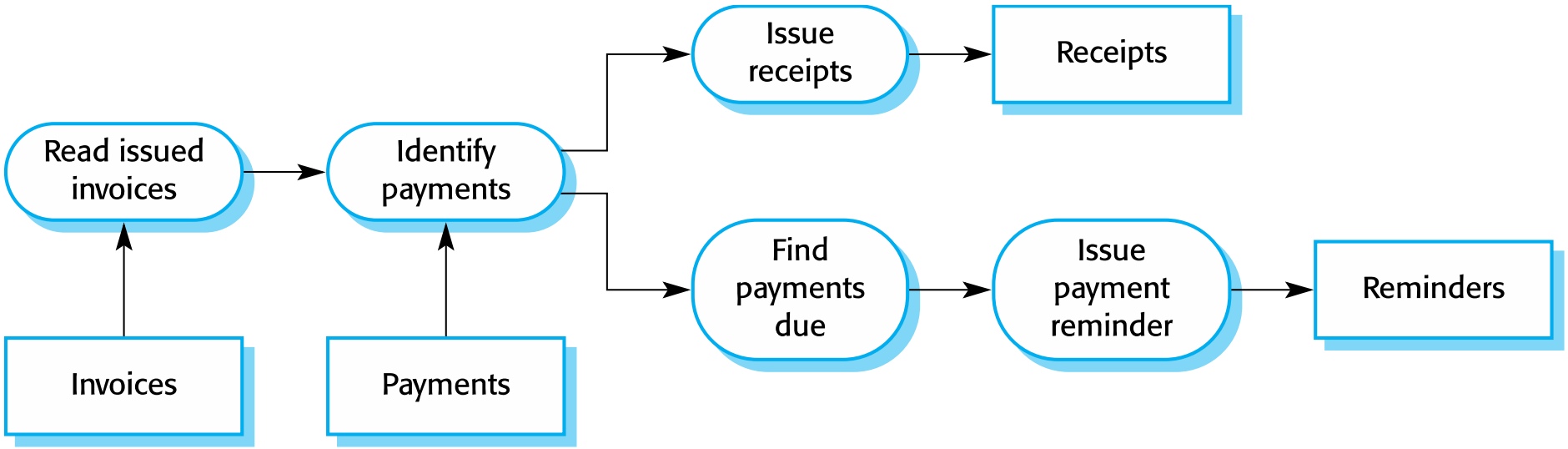
# A CLIENT–SERVER ARCHITECTURE FOR A FILM LIBRARY (ADDITIONAL READING)



# THE PIPE AND FILTER PATTERN

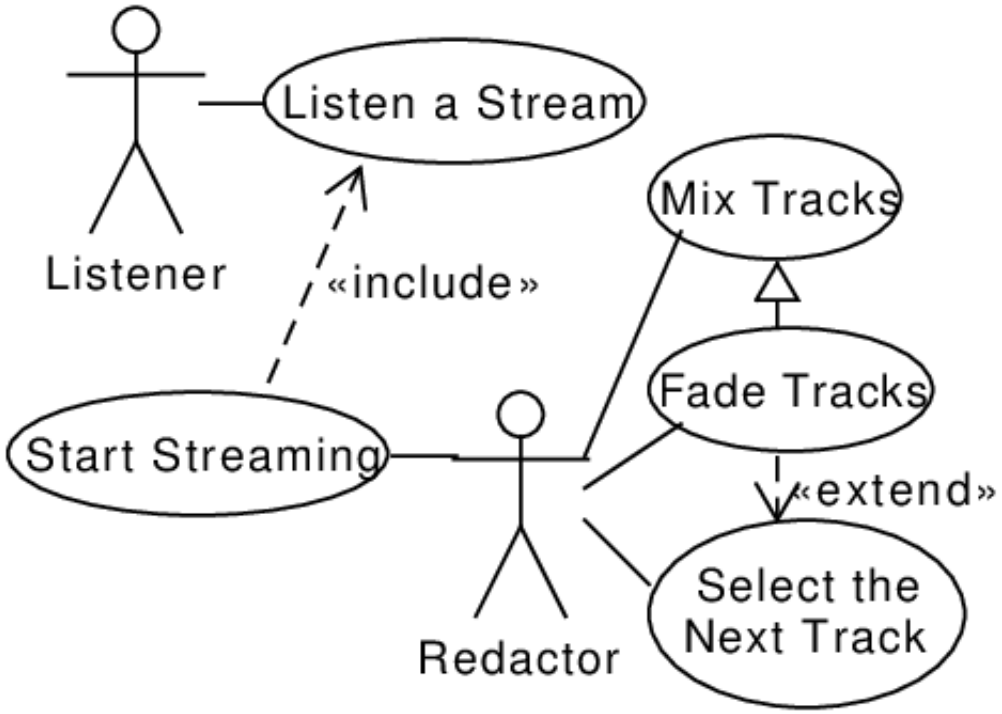
| Name | Pipe and filter |
|---|---|
| **Description** | The processing of the data in a system is organized so that each processing component (filter) is discrete and carries out one type of data transformation. The data flows (as in a pipe) from one component to another for processing. |
| **Example** | Figure in the next slide is an example of a pipe and filter system used for processing invoices. |
| **When used** | Commonly used in data processing applications (both batch- and transaction-based) where inputs are processed in separate stages to generate related outputs. |
| **Advantages** | Easy to understand and supports transformation reuse. Workflow style matches the structure of many business processes. Evolution by adding transformations is straightforward. Can be implemented as either a sequential or concurrent system. |
| **Disadvantages** | The format for data transfer has to be agreed upon between communicating transformations. Each transformation must parse its input and unparse its output to the agreed form. This increases system overhead and may mean that it is impossible to reuse functional transformations that use incompatible data structures. |

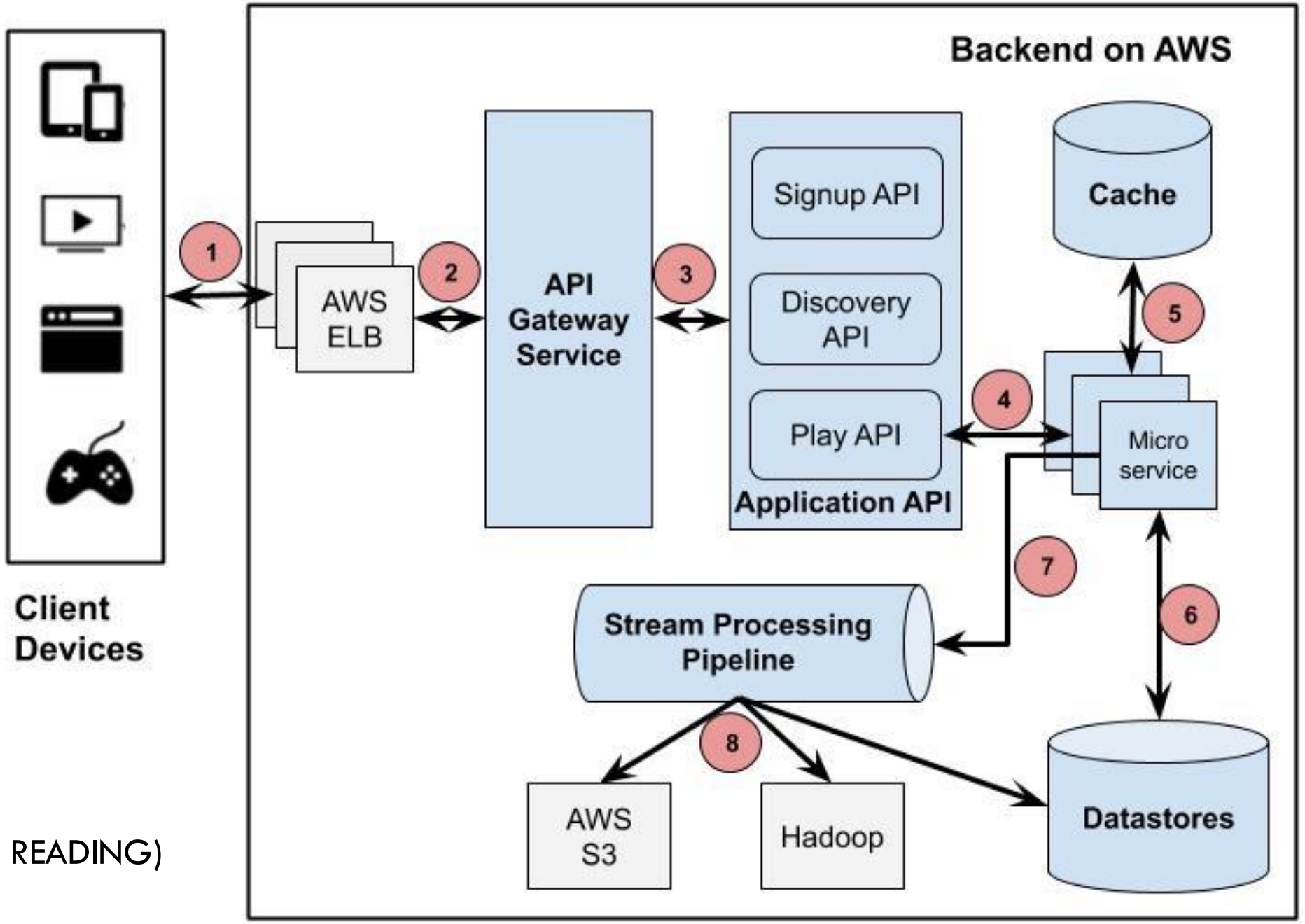
# AN EXAMPLE OF THE PIPE AND FILTER ARCHITECTURE (ADDITIONAL READING)



# ARCHITECTURE FOR A STREAMING PLATFORM (ADDITIONAL READING)

- a streaming platform is an on-demand online entertainment source for TV shows, movies and other streaming media. For example, think of things like Hulu, Netflix, Amazon Prime Video, Vimeo, and Sundance Now.





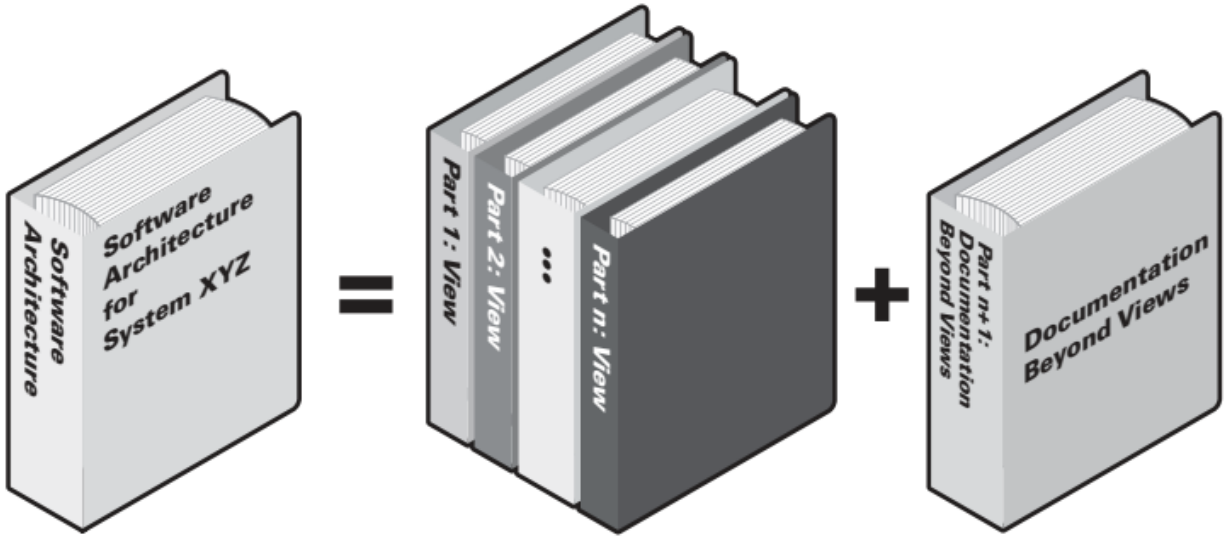

(ADDITIONAL READING)

# ARCHITECTURAL VIEWS

# DOCUMENTING SOFTWARE ARCHITECTURE

- Documenting an architecture is a matter of **documenting** the **relevant views** and then adding **documentation that applies to more than one view.**

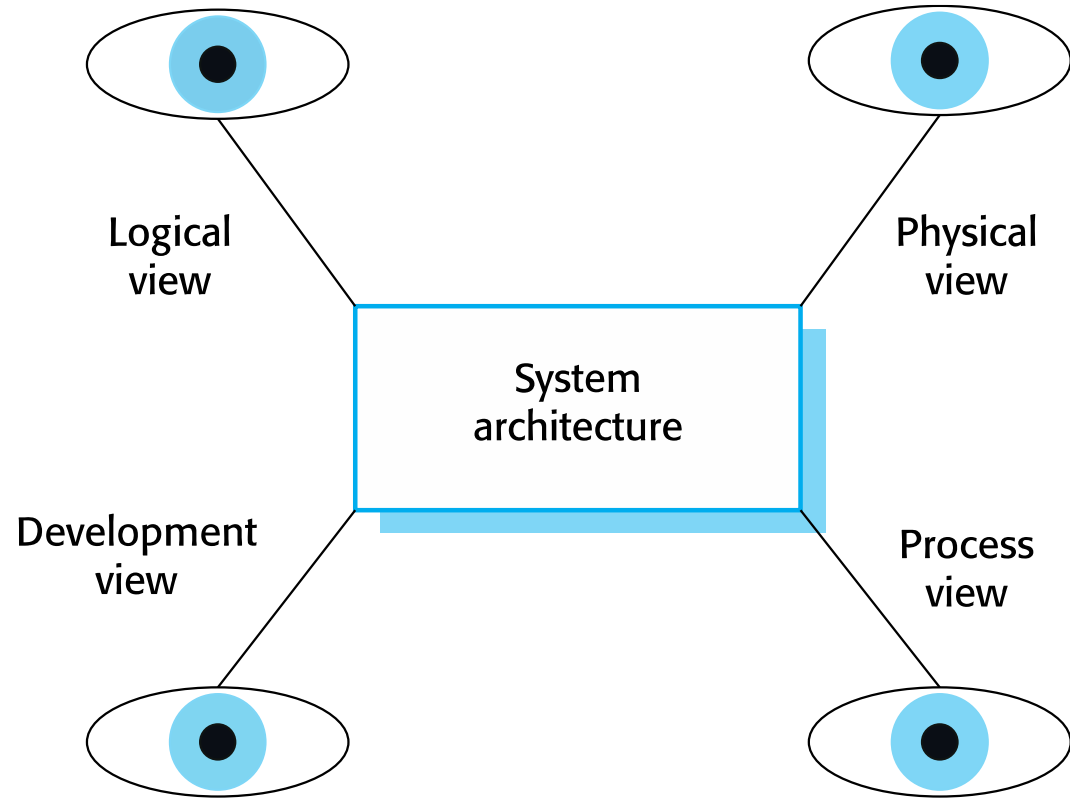
**Figure P.1**
A documentation package for a software architecture can be composed of one or more view documents and documentation that explains how the views relate to one another, introduces the package to its readers, and guides them through it.



# 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.
  - how a system is decomposed into modules
  - how the run-time processes interact
  - system components are distributed across a network

# ARCHITECTURAL VIEWS

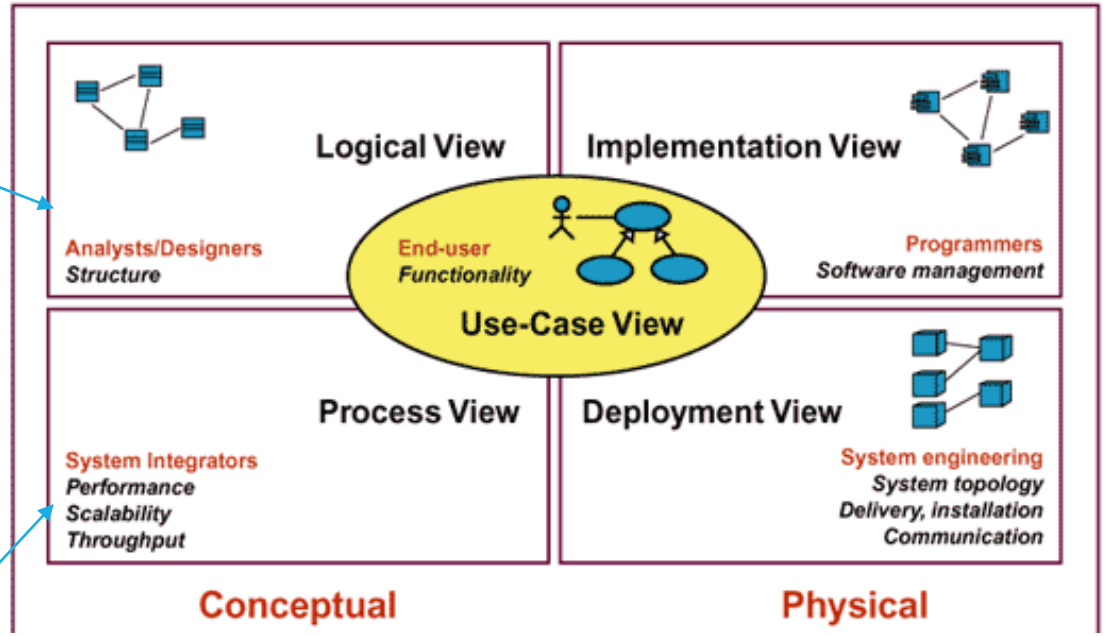



need to present multiple views of the software architecture.

# 4 + 1 VIEW MODEL OF SOFTWARE ARCHITECTURE

shows the key abstractions in the system as objects or classes.

shows how the software is decomposed for development.

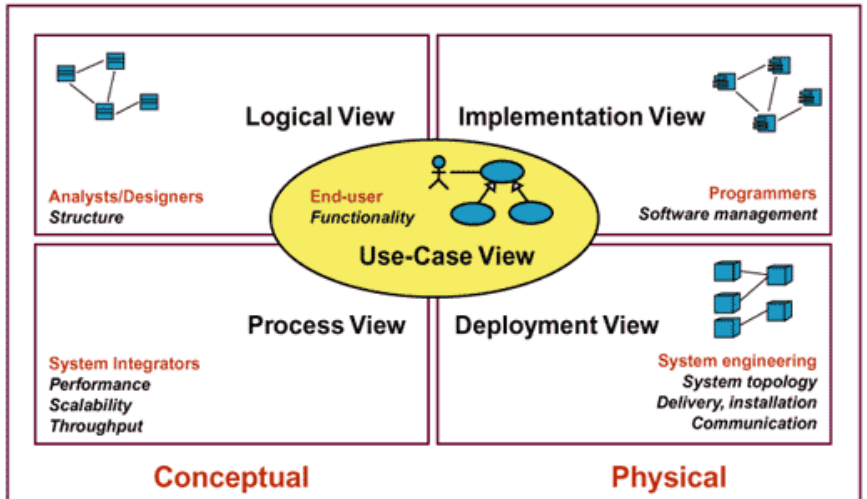



shows how, at run-time, the system is composed of interacting processes.

shows the system hardware and how software components are distributed across the processors in the system.

# 4 + 1 VIEW MODEL VS UML

- ✓ Logical view:
  - Class diagram, Communication diagram, Sequence diagram
- ✓ Process view:
  - Activity diagram
- ✓ Development view:
  - Component diagram, Package diagram.
- ✓ Physical view:
  - Deployment diagram
- ✓ Scenarios (+1):
  - Use-case



# APPLICATION ARCHITECTURES

# APPLICATION ARCHITECTURES

- ✓ Application systems are designed to meet an organizational need.
  - As businesses have much in common, their application systems also tend to have a common architecture that reflects the application requirements.

- ✓ A generic application architecture is an architecture for a type of software system that may be configured and adapted to create a system that meets specific requirements.

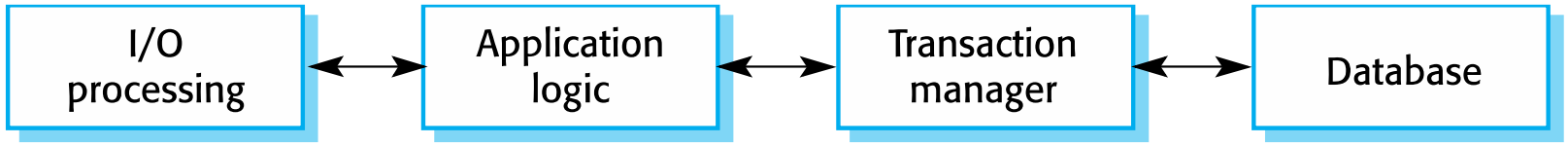
# USE OF APPLICATION ARCHITECTURES

- ✓ As a starting point for architectural design.
- ✓ As a design checklist.
- ✓ As a way of organising the work of the development team.
- ✓ As a means of assessing components for reuse.
- ✓ As a vocabulary for talking about application types.

# EXAMPLES OF APPLICATION TYPES

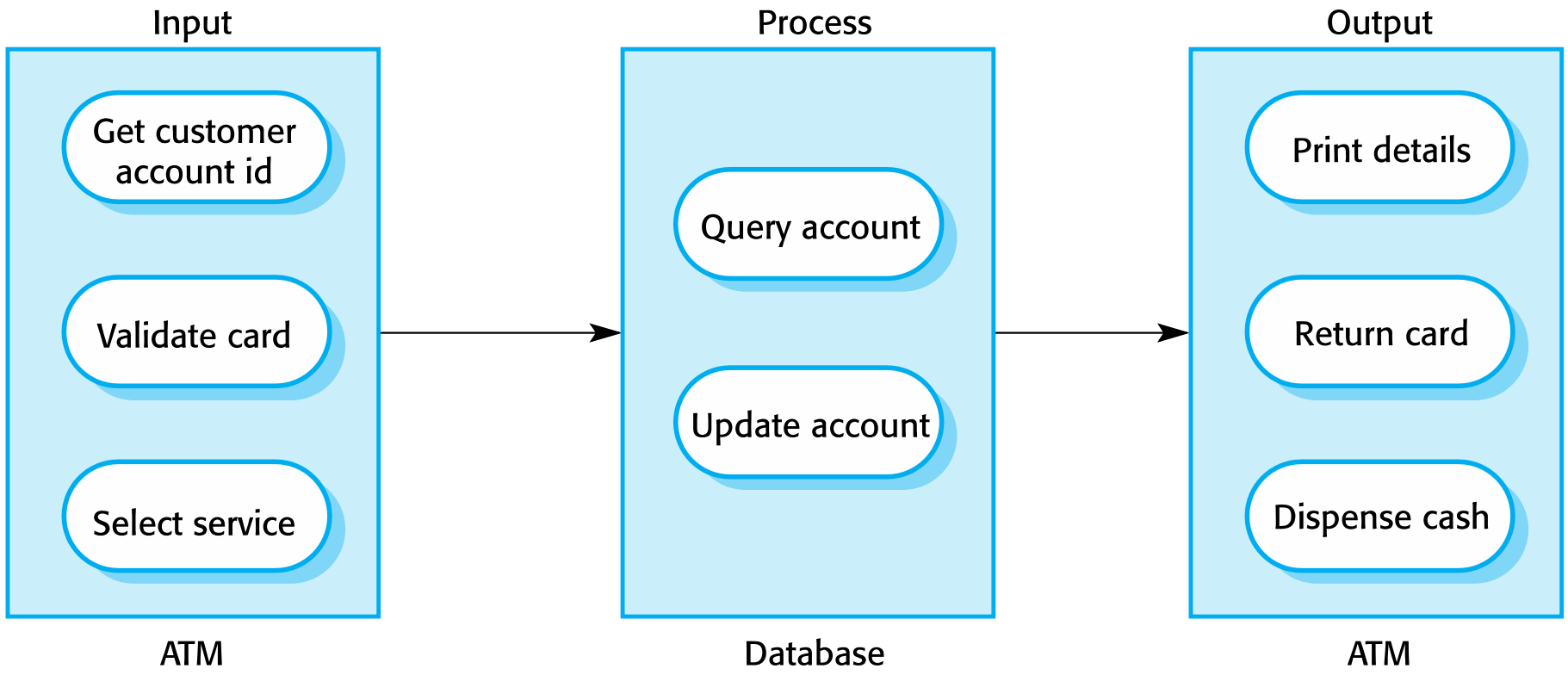
- ✓ Transaction processing applications
  - Database-centred applications that process user requests and update information in a system database.
- ✓ Event processing systems
  - Applications where system actions depend on interpreting events from the system's environment.
- ✓ Language processing systems
  - Applications where the users' intentions are specified in a formal language that is processed and interpreted by the system.

# TRANSACTION PROCESSING SYSTEMS



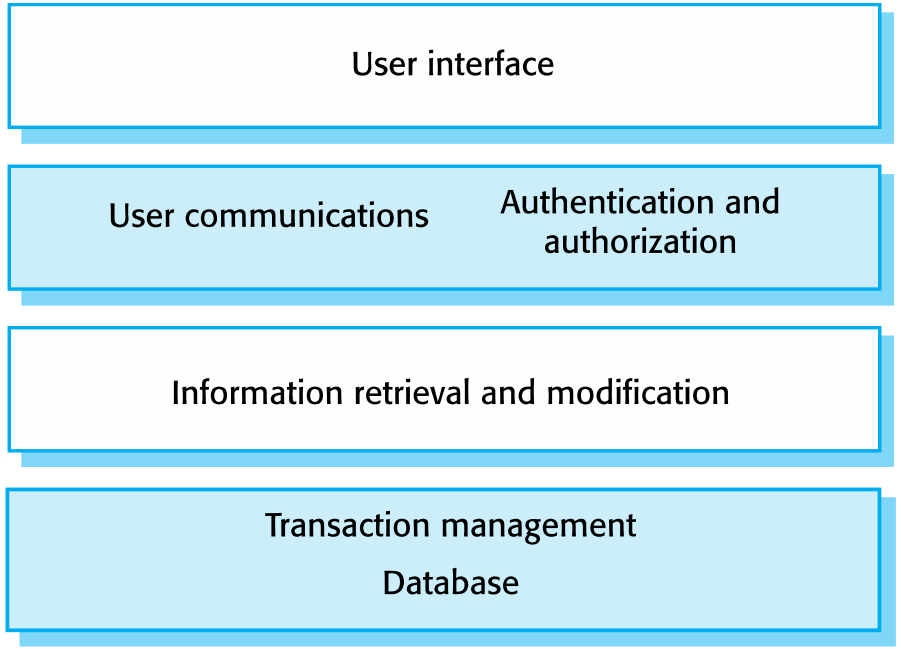

- ✓ Process user requests for information from a database or requests to update the database.
- ✓ From a user perspective a transaction is:
  - Any coherent sequence of operations that satisfies a goal;
  - For example - find the times of flights from London to Paris.
- ✓ Users make asynchronous requests for service which are then processed by a transaction manager.

# THE SOFTWARE ARCHITECTURE OF AN ATM SYSTEM (ADDITIONAL READING)



# INFORMATION SYSTEMS ARCHITECTURE

- ✓ Information systems have a generic architecture that can be organized as a layered architecture.
- ✓ These are transaction-based systems as interaction with these systems generally involves database transactions.
- ✓ Layers include:
  - The user interface
  - User communications
  - Information retrieval
  - System database



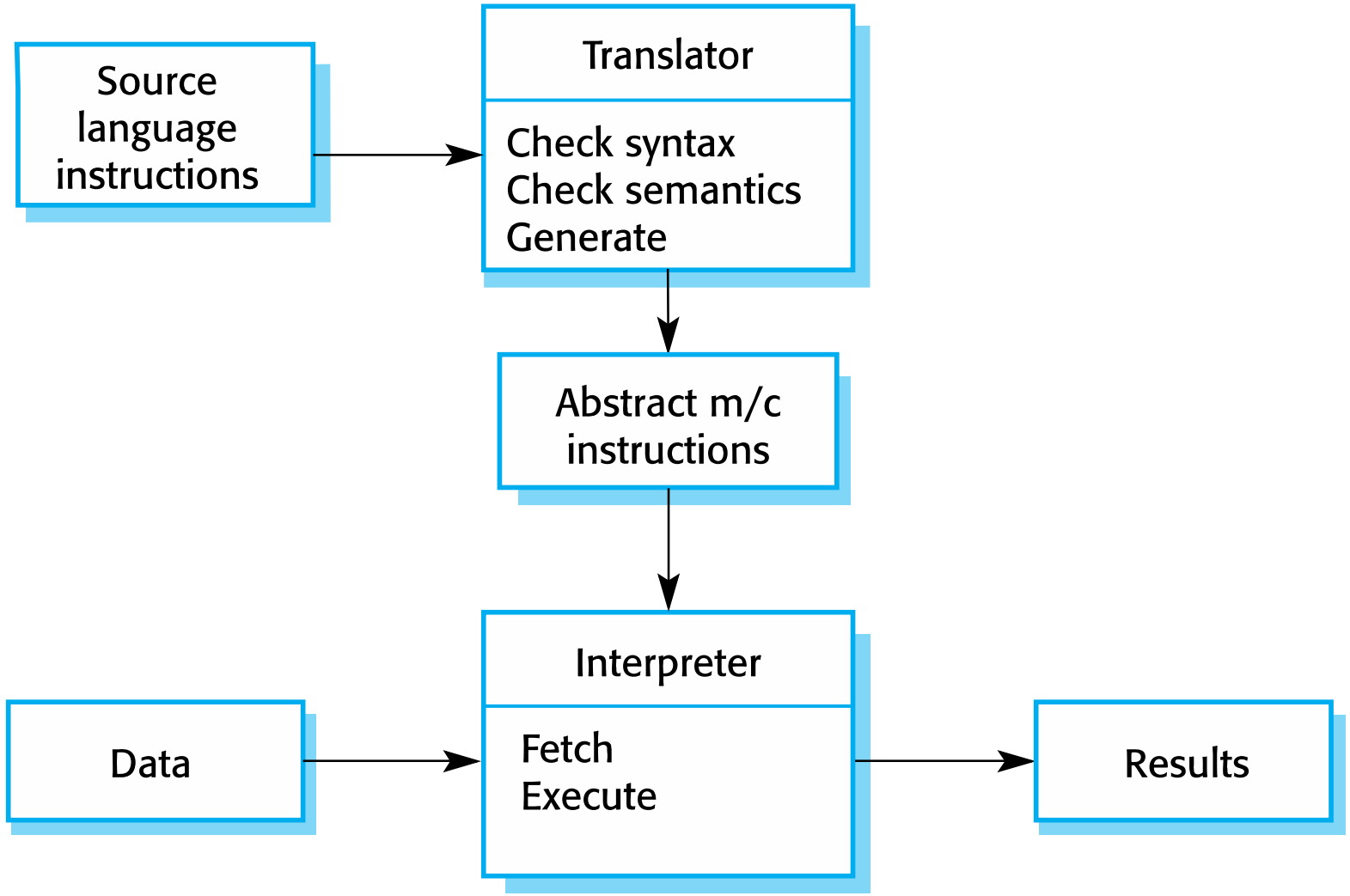
# WEB-BASED INFORMATION SYSTEMS

- ✓ Information and resource management systems are now usually web-based systems where the user interfaces are implemented using a web browser.
- ✓ Often implemented as multi-tier client/server architectures.
  - The web server is responsible for all user communications, with the user interface implemented using a web browser;
  - The application server is responsible for implementing application-specific logic as well as information storage and retrieval requests;
  - The database server moves information to and from the database and handles transaction management.

# LANGUAGE PROCESSING SYSTEMS

- ✓ Accept a natural or artificial language as input and generate some other representation of that language.
- ✓ May include an interpreter to act on the instructions in the language that is being processed.
- ✓ Used in situations where the easiest way to solve a problem is to describe an algorithm or describe the system data
  - Meta-case tools process tool descriptions, method rules, etc and generate tools.

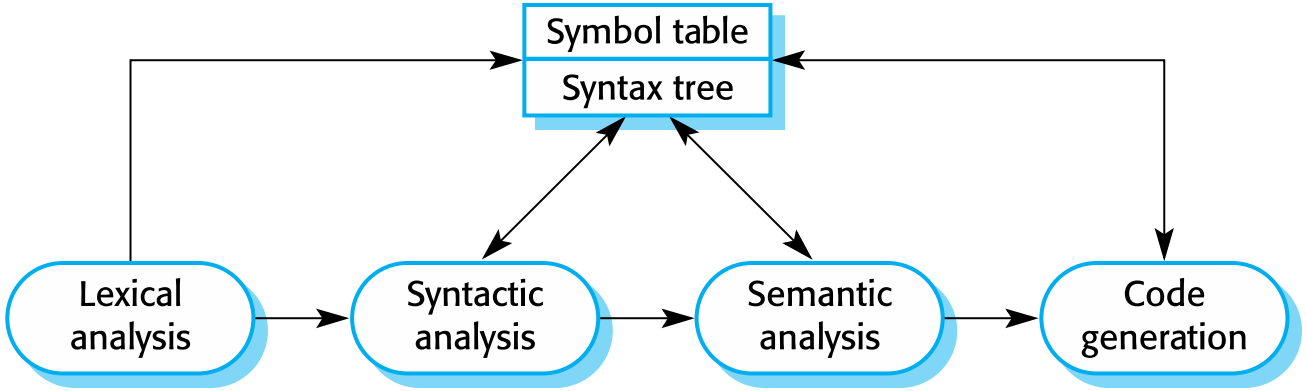
# THE ARCHITECTURE OF A LANGUAGE PROCESSING SYSTEM (ADDITIONAL READING)

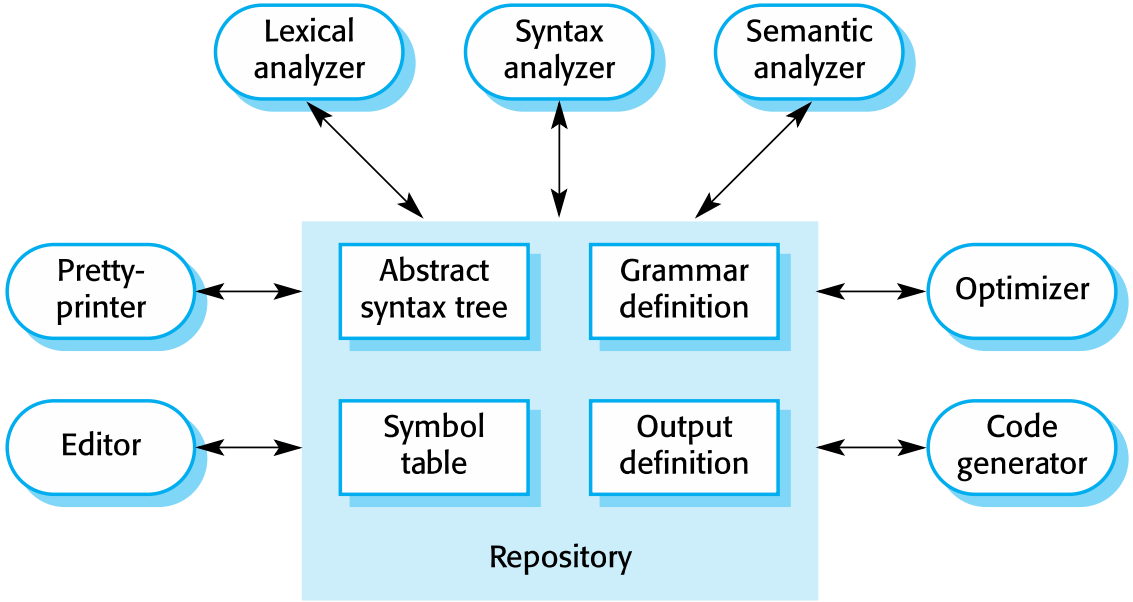


# COMPILER ARCHITECTURE

- ✓ Components
  - A lexical analyzer: takes input language tokens and converts them to an internal form.
  - A symbol table: holds information about the names of entities (variables, class names, object names, etc.) used in the text that is being translated.
  - A syntax analyzer: checks the syntax of the language being translated.
  - A syntax tree: an internal structure representing the program being compiled.
  - A semantic analyzer: uses information from the syntax tree and the symbol table to check the semantic correctness of the input language text.
  - A code generator: 'walks' the syntax tree and generates abstract machine code.

# A PIPE AND FILTER VS REPOSITORY ARCHITECTURE FOR COMPILERS (ADDITIONAL READING)



# SUMMARY

- ✓ A software architecture is a description of how a software system is organized.
- ✓ Software architecture consists of the structure of the system, combined with architecture characteristics the system must support, architecture decisions, and design principles.
- ✓ Architectures may be documented from several different perspectives or views such as a conceptual view, a logical view, a process view, and a development view.
- ✓ Architectural patterns are a means of reusing knowledge about generic system architectures. They describe the architecture, explain when it may be used and describe its advantages and disadvantages.

# SUMMARY (CONT.)

- ✓ Models of application systems architectures help us understand and compare applications, validate application system designs and assess large-scale components for reuse.
- ✓ Transaction processing systems are interactive systems that allow information in a database to be remotely accessed and modified by a number of users.
- ✓ Language processing systems are used to translate texts from one language into another and to carry out the instructions specified in the input language. They include a translator and an abstract machine that executes the generated language.