# SWE3002-42: Introduction to Software Engineering

## Lecture 6 – Architectural Design

Sooyoung Cha

Department of Computer Science and Engineering

# Topics covered

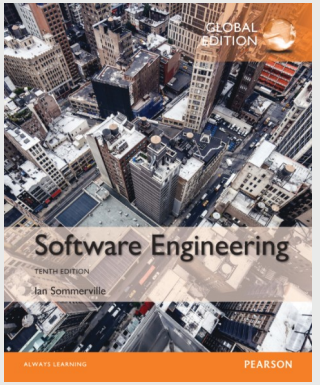
01 Architectural Design

02 Architectural Design Decisions

03 Architectural Views

04 Architectural Patterns

05 Application Architectures




Chapter 6. (p.168 ~ p.195)

# Architectural design

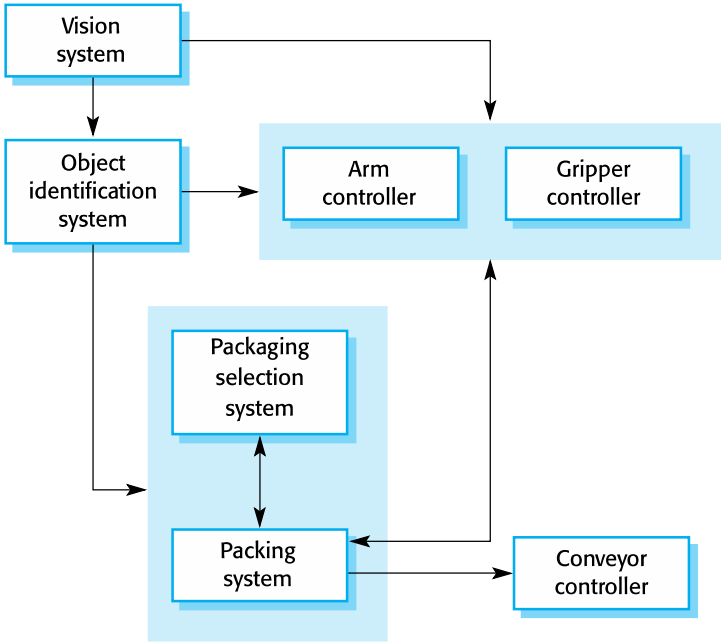
## [About architectural design]

- Designing the overall structure of a SW system.
- Identifying the main structural components in a system and the relationships between them.
- Designing a system organization that will satisfy the functional and non-functional requirements of a system.
- Depending on the system type, the background and experience of the system architect, and the specific requirements for the system.

# Architectural design

## [The architecture of a packing robot control system]

- Block diagram
  - Box (Boxes within boxes): A component (or sub-components)
  - Arrow: The flow of data or control signals from component to component.



# Architectural design

## [SW architectures at two levels of abstraction]

- *Architecture in the small* (Today's topic)
  - The architecture of individual programs.


- *Architecture in the large*
  - The architecture of complex enterprise systems that include other systems, programs, and program components.

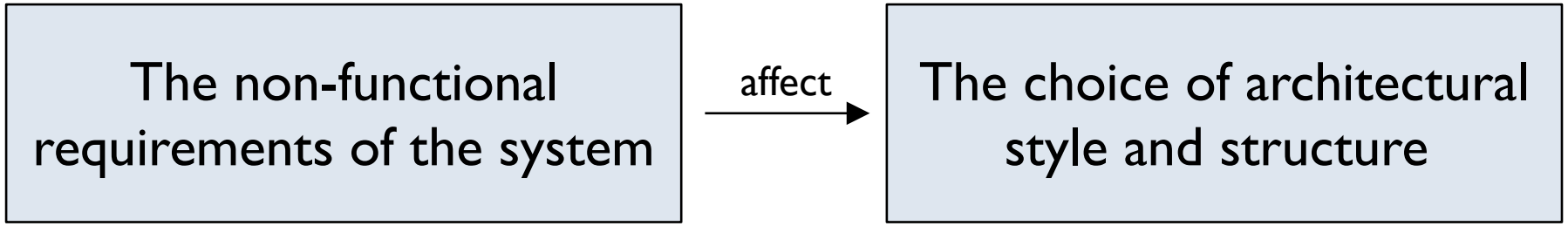
# Architectural design

## [Three advantages for designing SW architecture]

- Stakeholder communication
  - The architecture can be used as a focus for discussion by different stakeholders.
- System analysis
  - Architectural design decisions can affect whether the system can meet critical requirements such as performance, reliability, and maintainability.
- Large-scale reuse
  - The system architecture can support large-scale software reuse.

    (The system architecture is often the same for systems with similar requirements.)

# Architectural design decisions




- Non-functional requirements.
  - Constraints on the services offered by the system.

# Architectural design decisions

## [The five non-functional requirements]

- Performance
  - Localize critical operations within a small number of components.

    (The same computer > distributed across the network)
  - Reduce the number of component communications by using large components.
- Security
  - Use a layered structure for the architecture.

    (= Protecting the most critical assets in the innermost layers.)
- Safety
  - Reduce the costs and problems of safety validation.

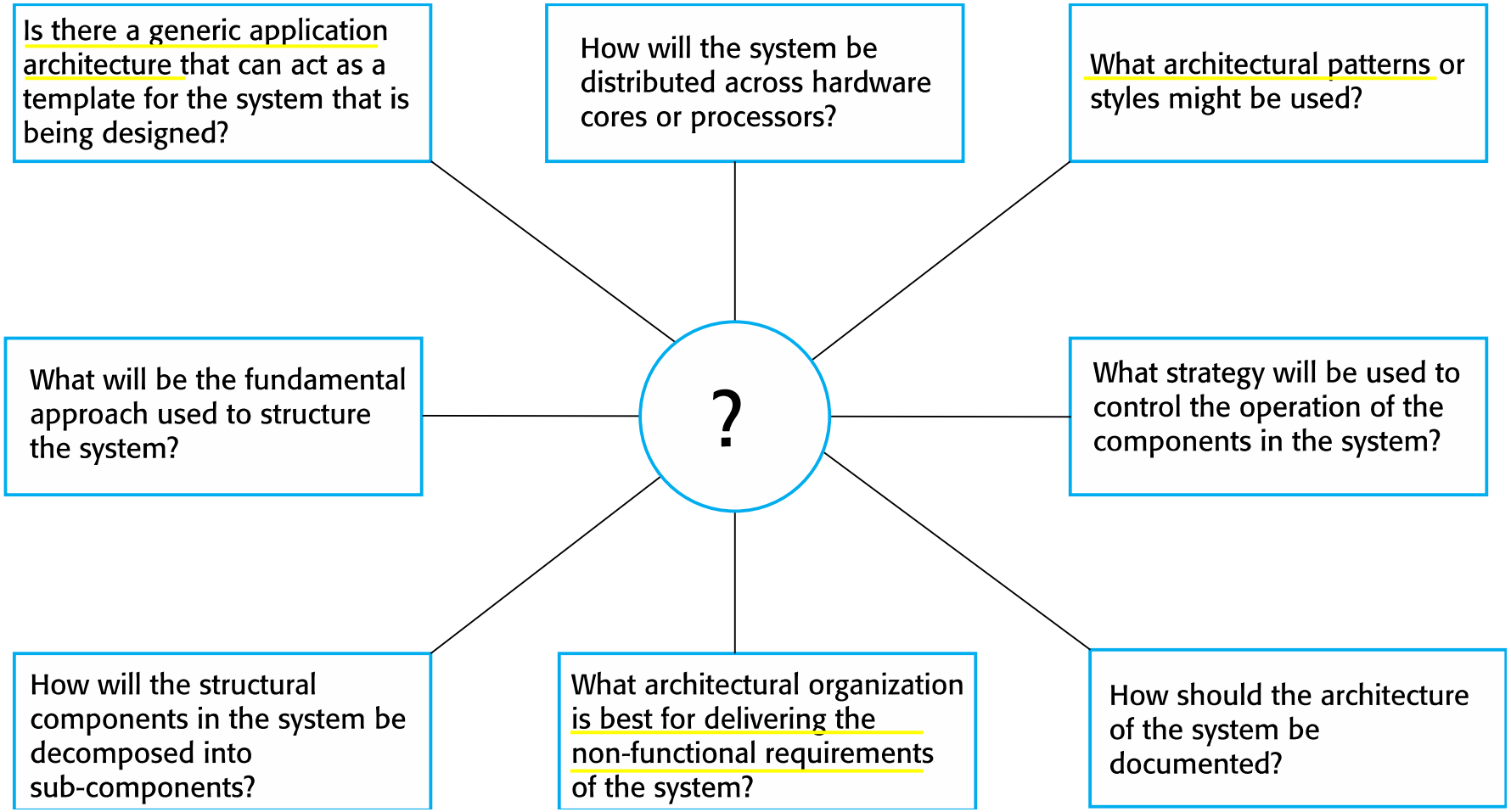
    (e.g., Putting safety-related operations together in a single component.)

# Architectural design decisions

## [The five non-functional requirements]

- Availability
  - Include redundant components to replace and update components without stopping the system.
- Maintainability
  - Use fine-grain, self-contained components that may readily be changed.

# Architectural design decisions

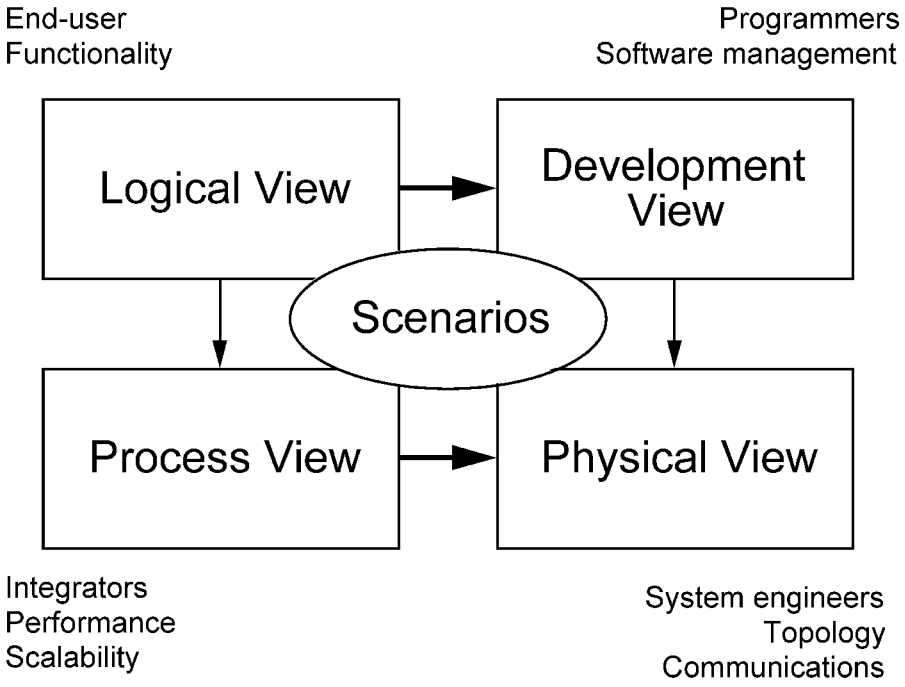


# Architectural views

## [Architectural views]

- What views or perspectives are useful when designing and documenting a system's architecture?

*The "4+1" view model*[1]






1: Philippe Kruchten "Architectural Blueprints—The "4+1" View Model of Software Architecture"

# 4+1 view model of software architecture

## [4+1 view model of software architecture]

| | |
|---|---|
| **A logical view** | which shows the key abstractions in the system as objects. (functional requirements). |
| **A process view** | which shows how, at run-time, the system is composed of interacting processes. (non-functional requirements: performance) |
| **A development view** | which shows how the software is decomposed for development. (Useful for software managers and programmers) |
| **A physical view** | which shows how the system HW and SW components are distributed across the processors in the system. (systems engineers) |
| **Scenarios** | which shows the sequences of interactions between objects or between processes. |

# Architectural patterns

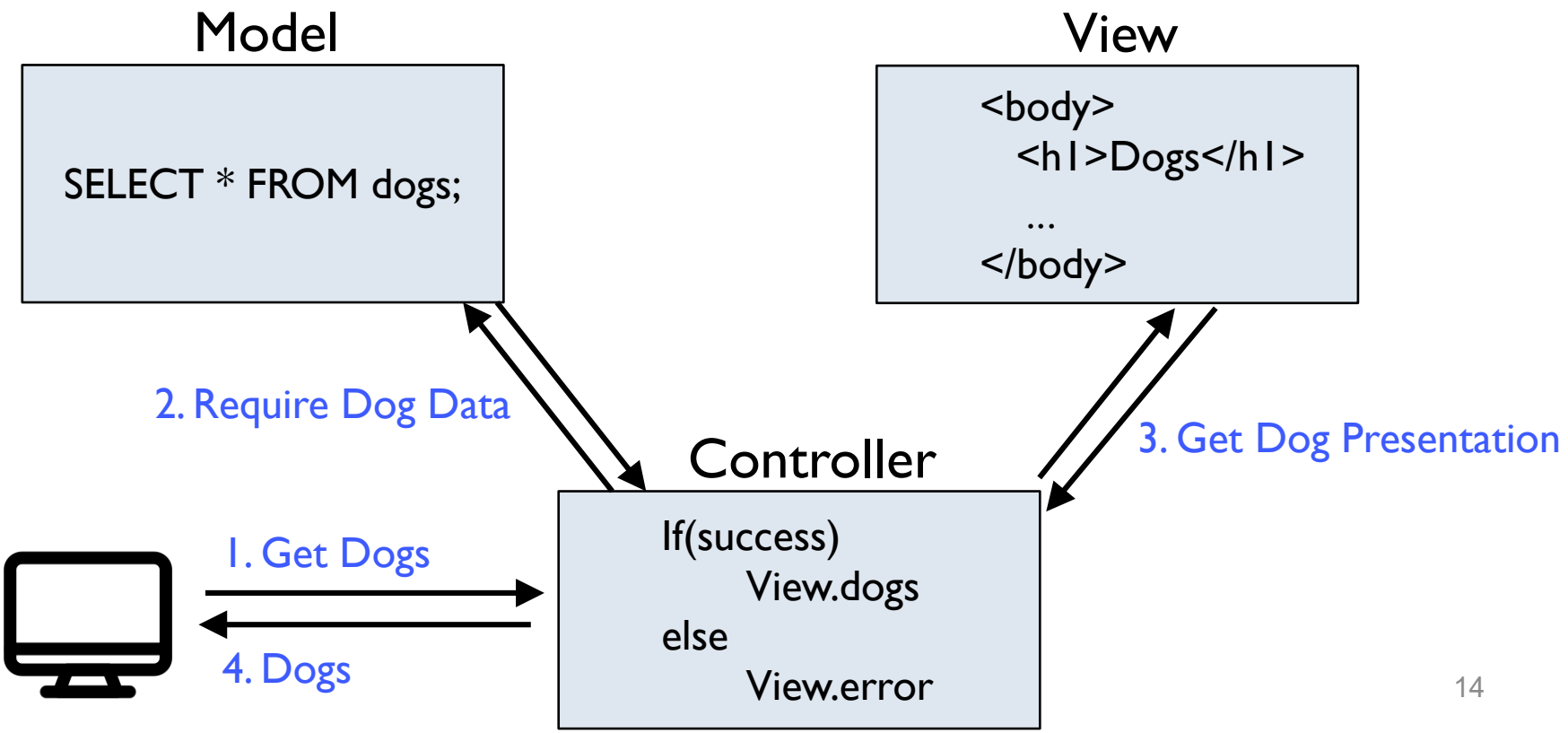
## [Architectural patterns]

- A stylized, abstract description of good practice, which has been tried and tested in different systems and environments.
- Describing a system organization that has been successful in previous systems.
- Including information on when it is and is not appropriate to use that pattern, and details on the pattern's strengths and weaknesses.

# Architectural patterns

## [Model-View-Controller (MVC) pattern]

- Separate presentation and interaction from the system data.



# Architectural patterns

## [Model-View-Controller (MVC) pattern]

- Separates presentation and interaction from the system data.
- The system is structured into three logical components that interact with each other.

| | |
|---|---|
| **Model component** | manages the system data and associated operations on that data. |
| **View component** | defines and manages how the data is presented to the user. |
| **Controller component** | manages user interaction (e.g., key presses, mouse clicks, etc.) and passes these interactions to the View and the Model. |

# Architectural patterns

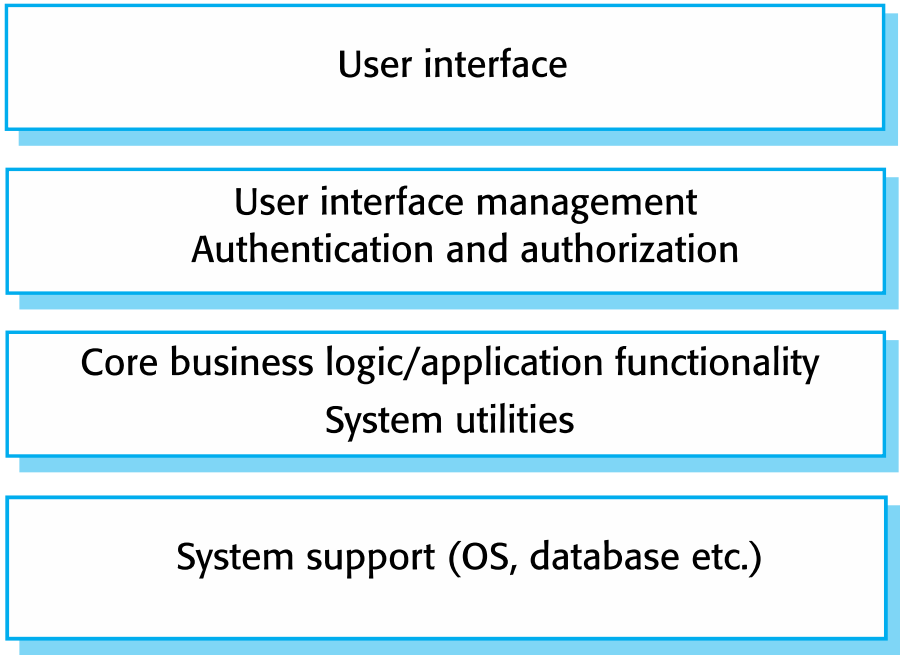
## [Model-View-Controller (MVC) pattern]

- When used?
  - There are multiple ways to view and interact with data.
  - The future requirements for interaction and presentation of data are unknown.
- Advantages
  - Allow the data to change independently of its representation and vice versa.
  - Support presentation of the same data in different ways.
- Disadvantages
  - Involve additional code and code complexity when the data model and interactions are simple.

# Layered Architecture

## [Layered Architecture]

- Organizing the system into layers, where each layer provides services to the layer above it.



# Layered Architecture

## [Layered Architecture]

- Achieving separation and independence.
- Supporting the incremental development of systems.
  - Some of the services provided by each layer are available to users.
  - Only the adjacent layer is affected when layer interfaces change.
- When used?
  - When building new facilities on top of existing systems.
  - When several teams develop a system and each team is responsible for a layer of functionality.
  - When there is a requirement for multi-level security.

# Layered Architecture

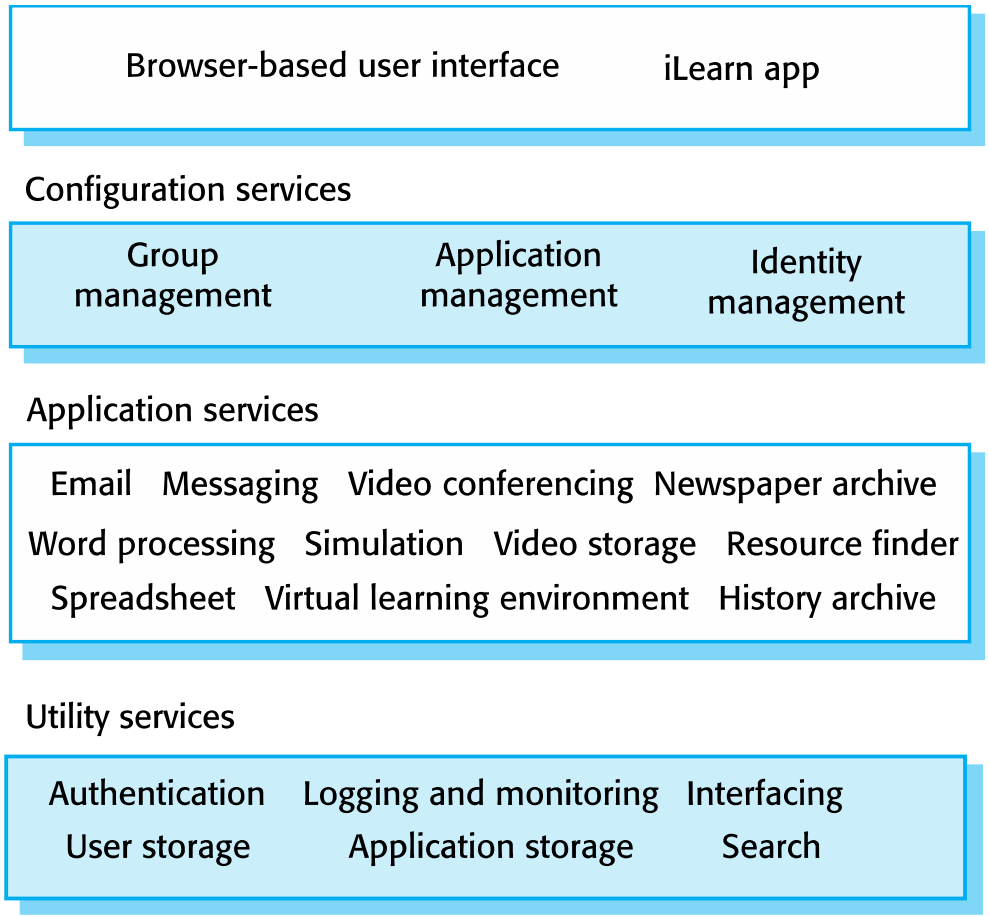
## [Layered Architecture]

- Advantages
  - Allowing 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
  - Providing a clean separation between layers is often difficult.
  - Not interacting 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.

# Layered Architecture

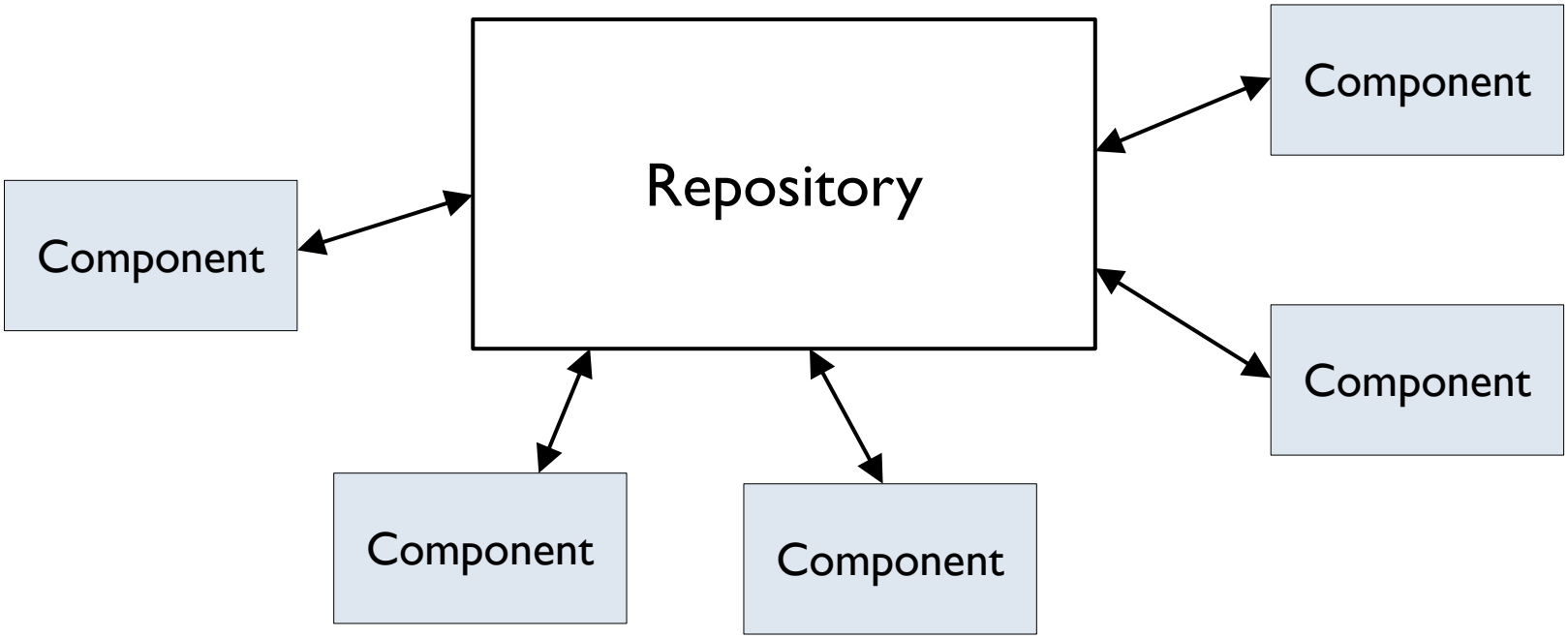
## [The architecture of the iLearn system]



# Repository Architecture

## [Repository architecture]

- All data is managed in a repository accessible to all system components.
- Components do not interact directly, only through the repository.



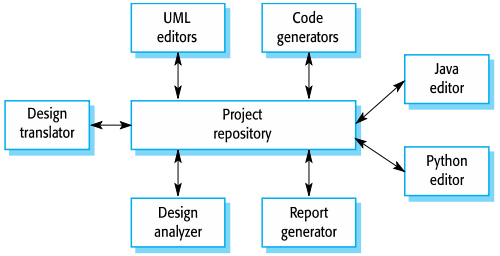
# Repository Architecture

## [Repository architecture]

- When used?
  - When having a system in which large volumes of information are generated that has to be stored for a long time.
  - Data-driven systems where the inclusion of data in the repository triggers an action or tool.
- Advantages
  - Components can be independent;

    (Don't need to know of the existence of other components.)
  - All data can be managed consistently as it is all in one place.

# Repository Architecture




## [Repository architecture]

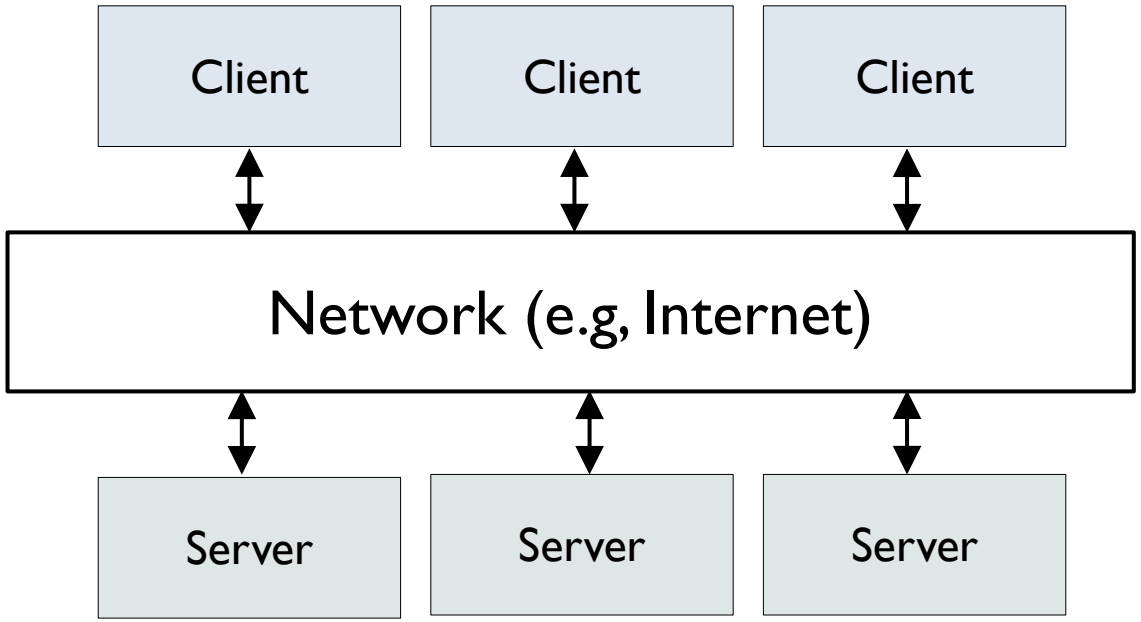
- Disadvantages
  - The repository is a single point of failure so problems in the repository affect the whole system.
  - May be inefficiencies in organizing all communication through the repository.
  - Distributing the repository across several computers may be difficult.


https://www.slashfilm.com/606595/the-massive-mistake-that-nearly-destroyed-toy-story-2/

# Client-Server Architecture

## [Client-Server Architecture]

- The system is presented as a set of services, with each service delivered by a separate server.
- Clients (e.g., users of these services) access servers to make use of them.



# Client-Server Architecture

## [The major components of Client-Server Architecture]

- A set of servers that offer services to other components.
  - ex) Print servers that offer printing services.
  - ex) File servers that offer file management services.
- A set of clients that call on the services offered by servers.
- A network that allows the clients to access these services.

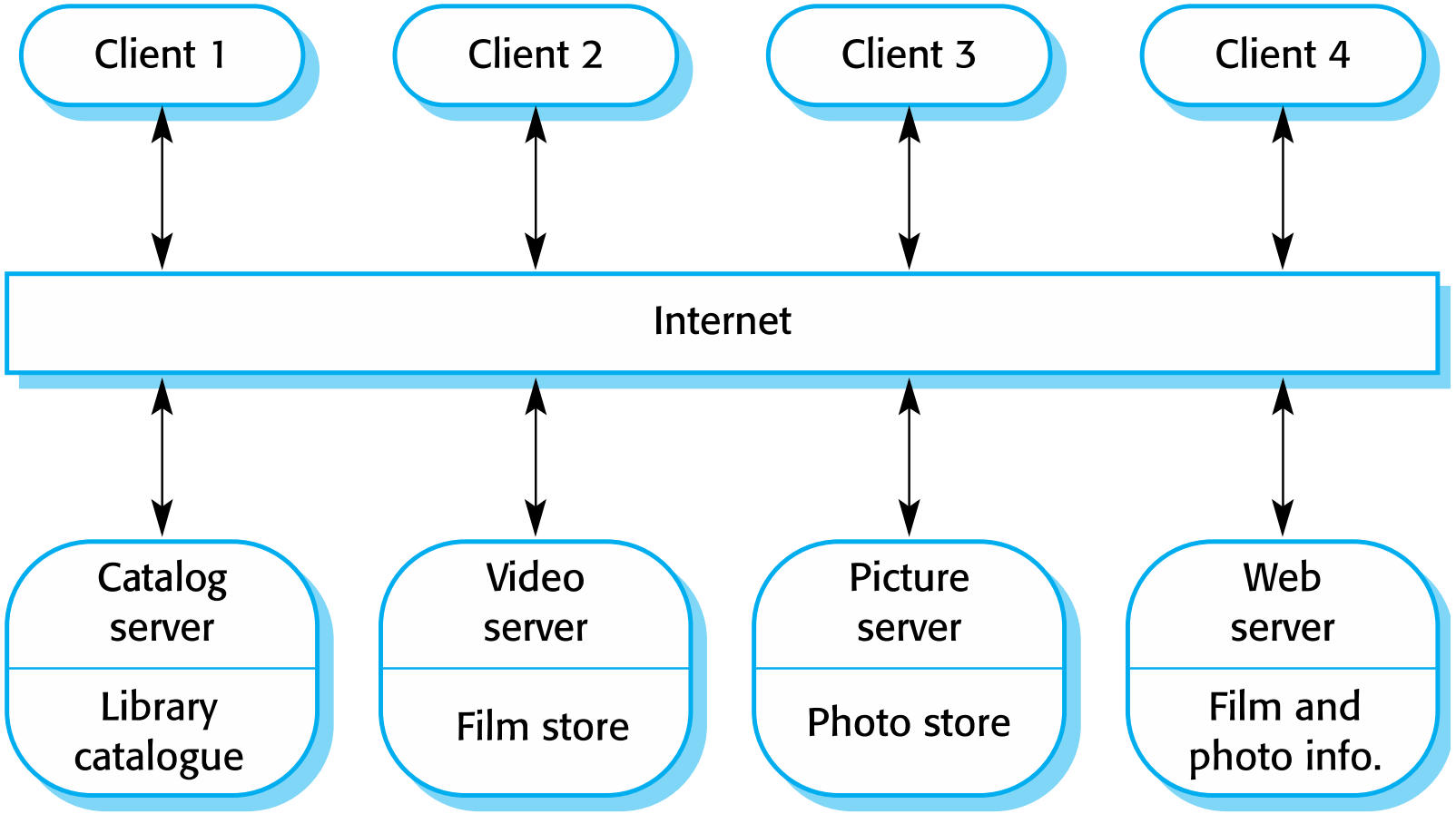
# Client-Server Architecture

## [Client-Server Architecture]

- When used?
  - Used when data in a shared DB has to be accessed from a range of locations.
- Advantages
  - 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 susceptible to denial of service (DoS) attacks or server failure.
  - Performance problems as it depends on the network.
  - Management problems if servers are owned by different organizations.

# Client-Server Architecture

**[A client–server architecture for a film library]**



# Pipe and filter Architecture

## [Pipe and filter Architecture]

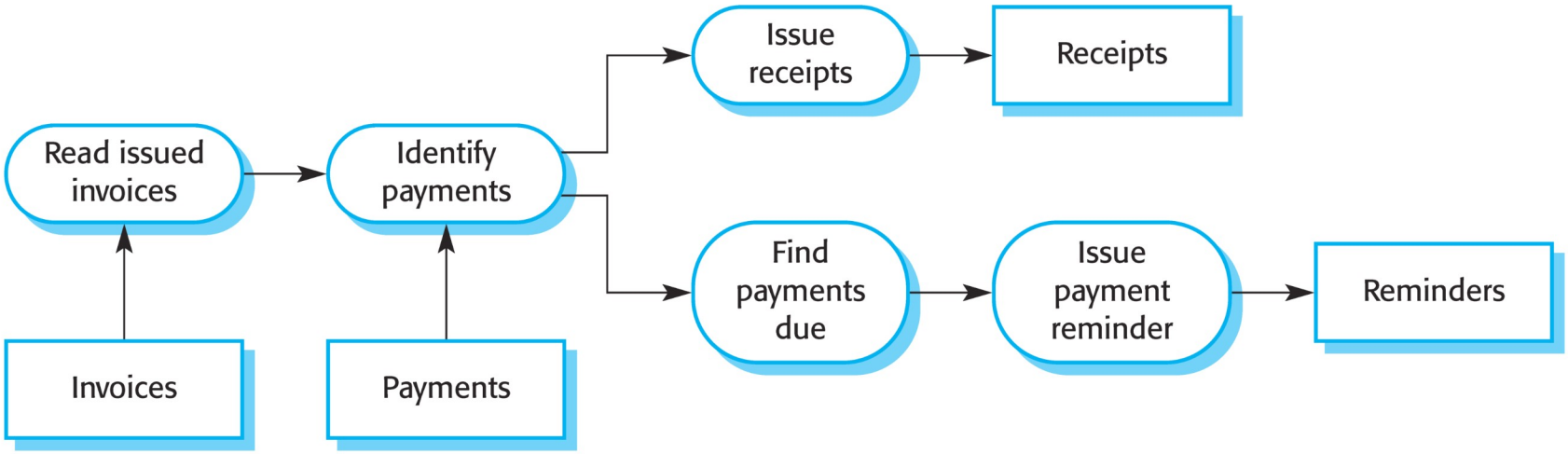
- The processing of the data is organized so that each processing component (filter) is discrete and carries out one type of data transformation.
- The data flows (as in a pipe) from one component to another for processing.

- When used?
  - Commonly used in data-processing applications where inputs are processed in separate stages to generate related outputs.

```
$cat test.c | grep 'cha'
```

# Pipe and filter Architecture

**[An example of the pipe and filter architecture]**



# Pipe and filter Architecture

## [Pipe and filter Architecture]

- Advantages
  - Easy to understand and supports transformation reuse.
  - Workflow style matches the structure of many business processes.
  - Evolution by adding transformations is straightforward.
- Disadvantages
  - The format for data transfer has to be agreed between communicating transformations.
  - It can not reuse architectural components that use incompatible data structures.

# Application architectures

## [Application architectures]

- Application systems used by the businesses have much in common.
- Application architectures encapsulate the principal characteristics of a class of systems.
  - Real-time system (e.g., data collection systems or monitoring systems)
- Use case of application architecture models.
  - As a starting point for architectural design process.
  - As a design checklist.
  - As a way of organizing the work of the development team.
  - As a vocabulary for talking about application types.

    (Using the concepts identified in the generic architecture.)

# Application architectures

## [The architectures of two types of application]

- Transaction processing applications
  - Database-centered applications that process user requests for information and update the information in a database.
  - The most common types of interactive business systems.

    ex) Interactive banking systems, e-commerce systems, booking systems.

- Language processing systems
  - Systems in which the user's intentions are expressed in a formal language, such as a programming language.
  - Processing the language into an internal format and interpreting this internal representation

    ex) Compilers (high-level language programs → machine code).

# Transaction processing systems

## [Transaction processing systems]

- Processing user requests for information from a database, or requesting to update a database.

- A transaction
  - Any coherent sequence of operations that satisfies a goal.

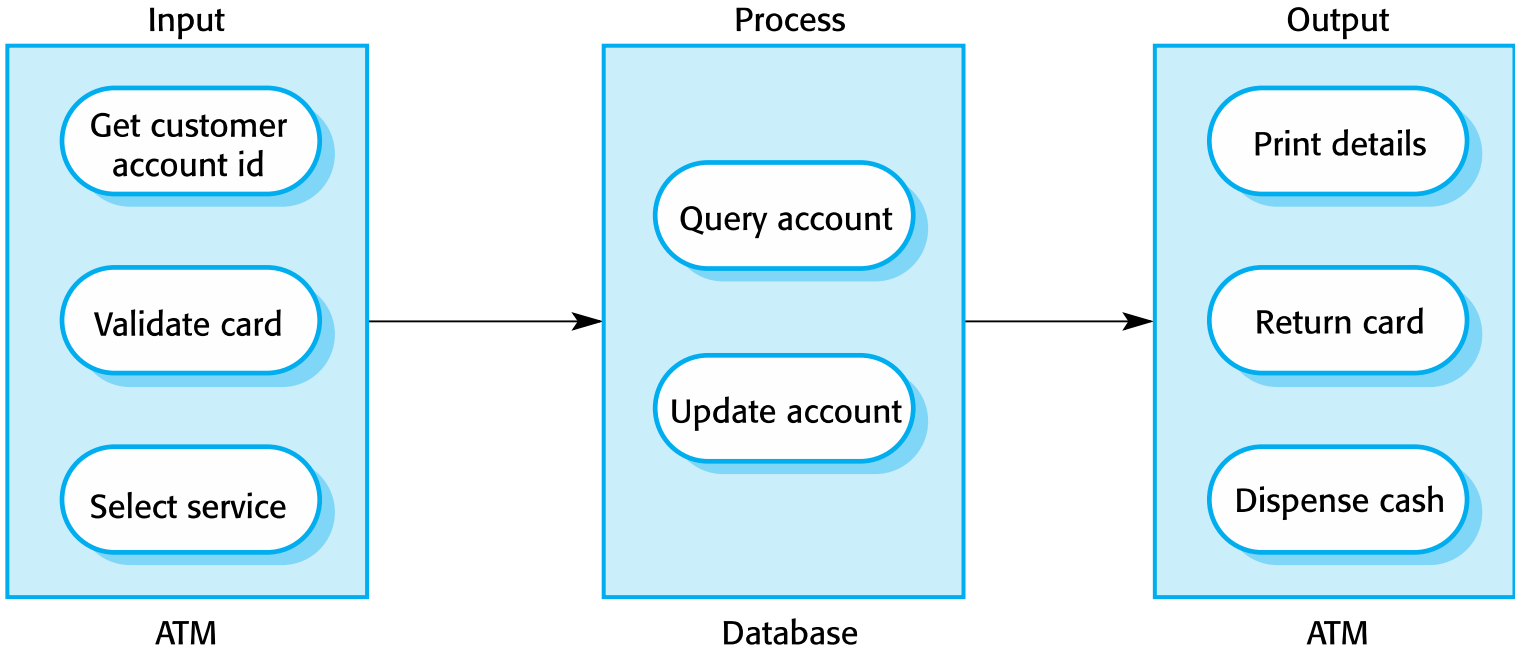
    ex) "find the times of flights from Seoul to Pittsburgh :)"

    ex) "a customer request to withdraw money from a bank account using an ATM"

# Transaction processing systems

## [The software architecture of an ATM system]

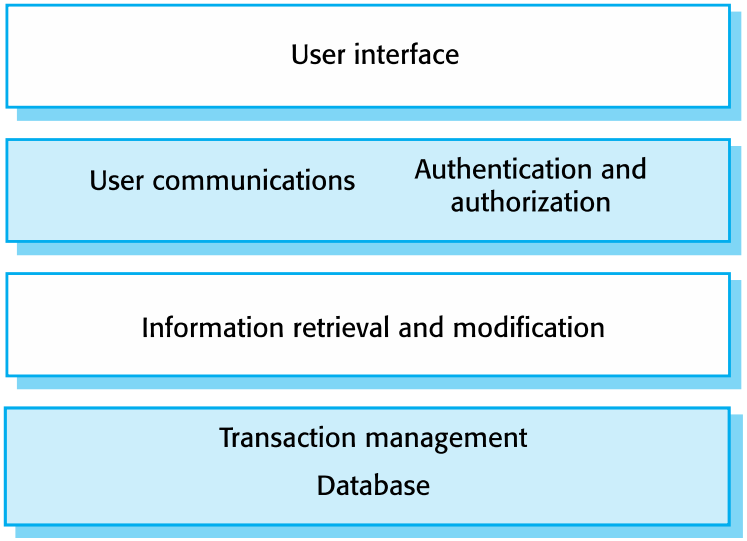
- Transaction processing systems can be organized as a "pipe and filter architecture", with system components.
  - Two components: ATM software and the account processing software.



# Information systems

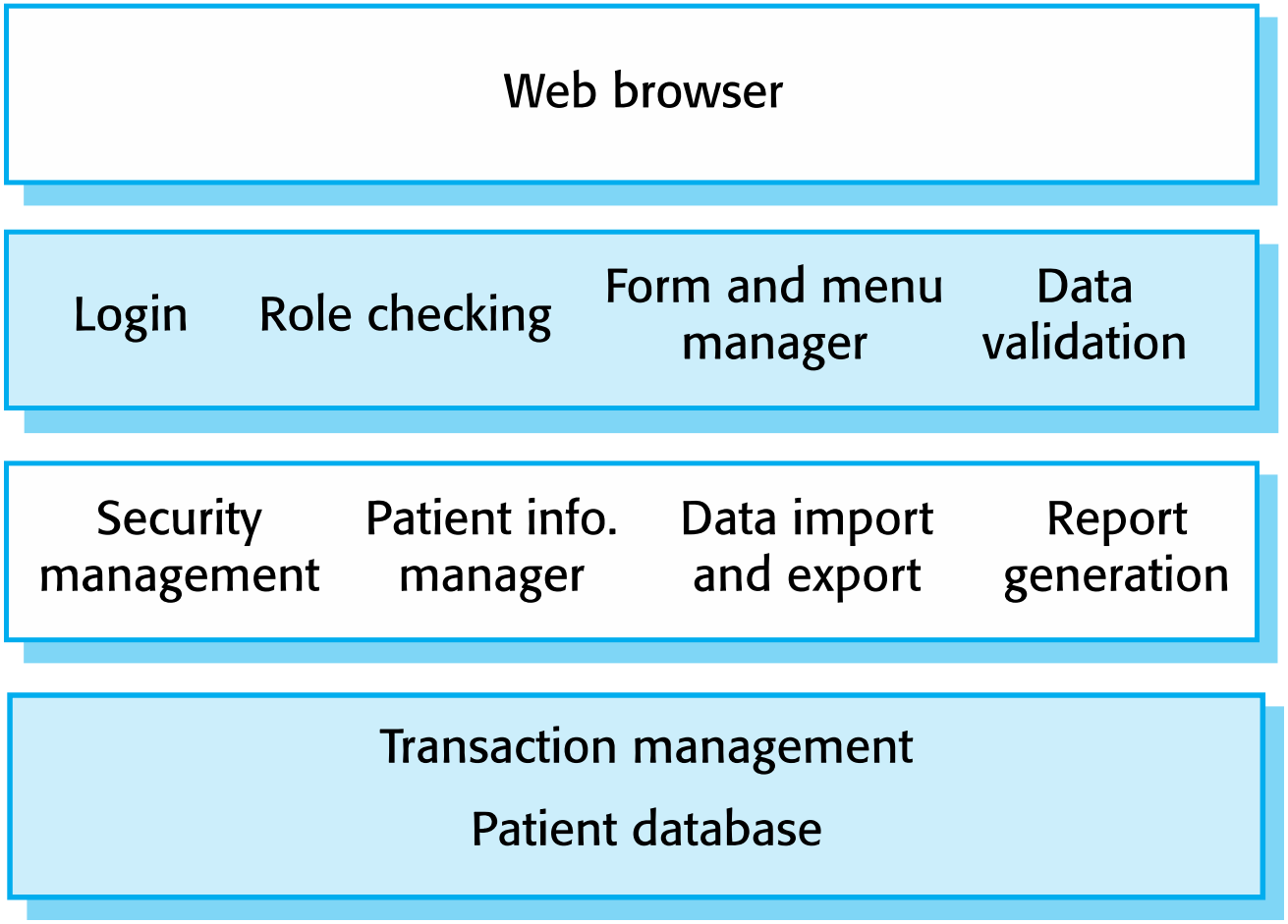
## [About Information systems]

- All systems that involve interaction with a shared database.
- Web-based systems, where the user interface is implemented in a web browser.
- The Information system is modeled using a layered approach.
  - The top layer (e.g., user interface)
  - The bottom layer (e.g., system database)



# Information systems

## [The architecture of the Mentcare system]



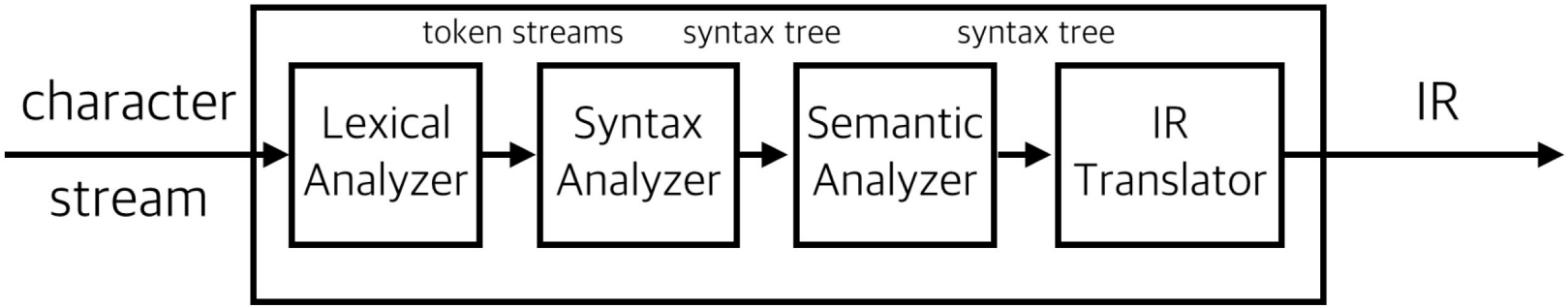
# Language processing systems

## [Language processing systems]

- Translate one language into an alternative representation of that language.
- Compiler
  - Translate a program written in one language (“source language”) into a program written in another language (“target language”).

# Language processing systems

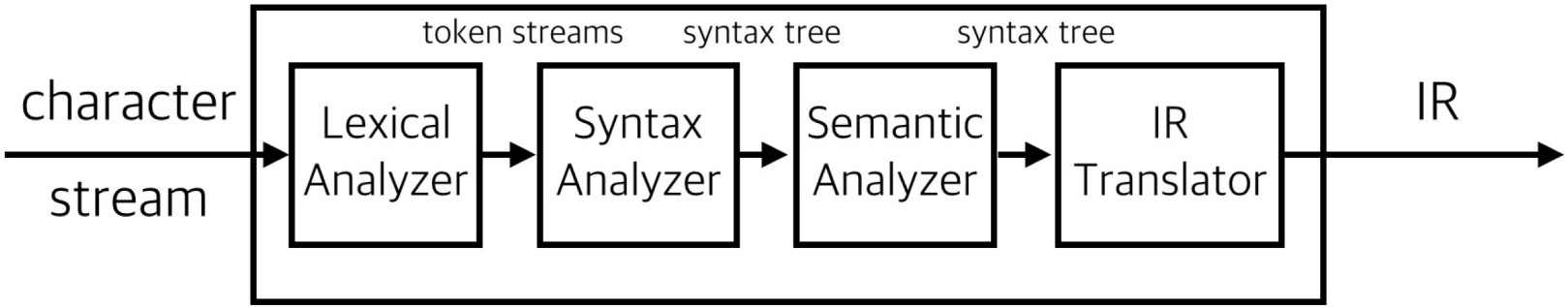
## [Front End of Modern Compilers]




- The lexical analyzer transforms the character stream into a stream of tokens.
- The syntax analyzer transforms the stream of tokens into a syntax tree.
- The semantic analyzer checks if the program is semantically well-formed.
- The IR translator translates the syntax tree into IR.

# Language processing systems

## [Lexical Analyzer]




- The lexical analyzer transforms the character stream

  pos = init + rate * 10

  into a sequence of lexemes

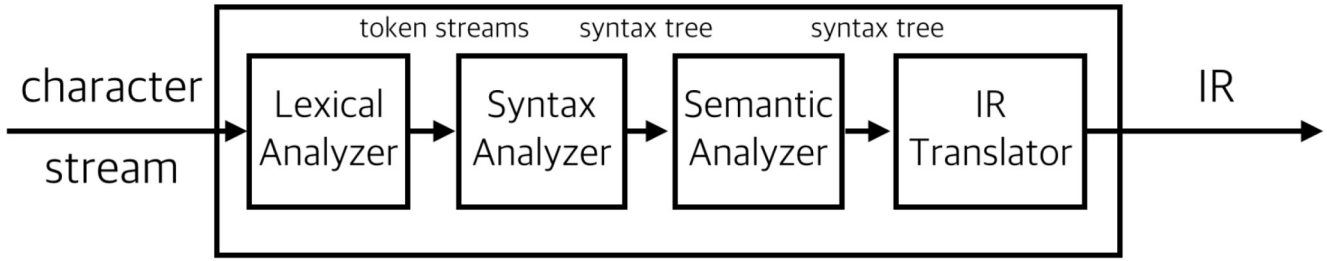
  “pos”, “=”, “init”, “+”, “rate”, “*”, “10”

  and then produces a token sequence.

  (ID, pos), ASSIGN, (ID, init), PLUS, (ID, rate), MULT, (NUM, 10)

# Language processing systems

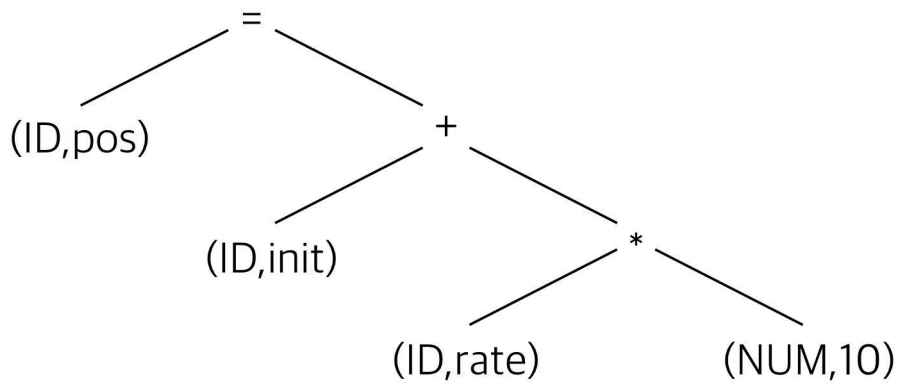
## [Syntax Analyzer]




- The parser transforms the sequence of tokens

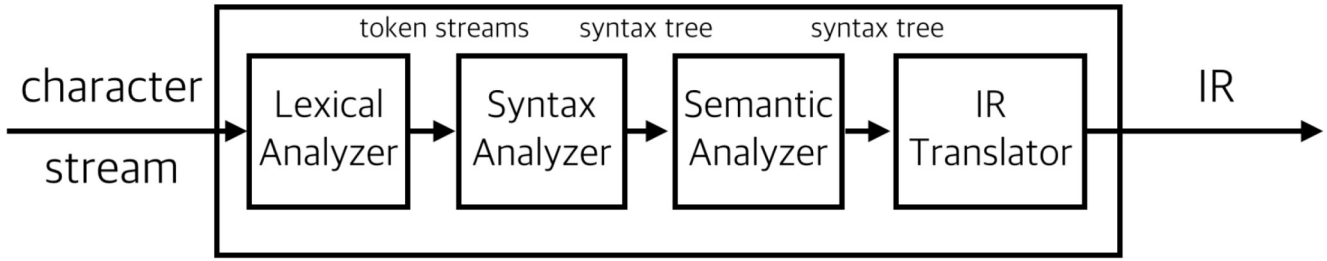
  (ID, pos), ASSIGN, (ID, init), PLUS, (ID, rate), MULT, (NUM,10)

  into the syntax tree:



# Language processing systems

## [Semantic Analyzer]



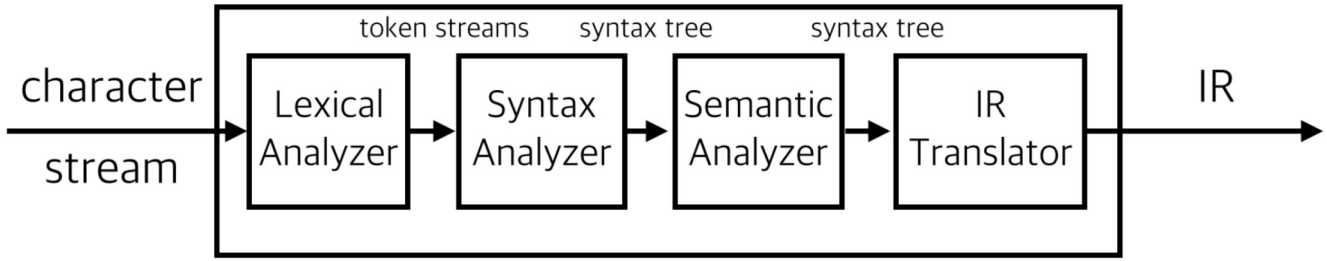

- ex) Type errors:

```
int x = 1;
string y = "hello";
int z = x + y;
```

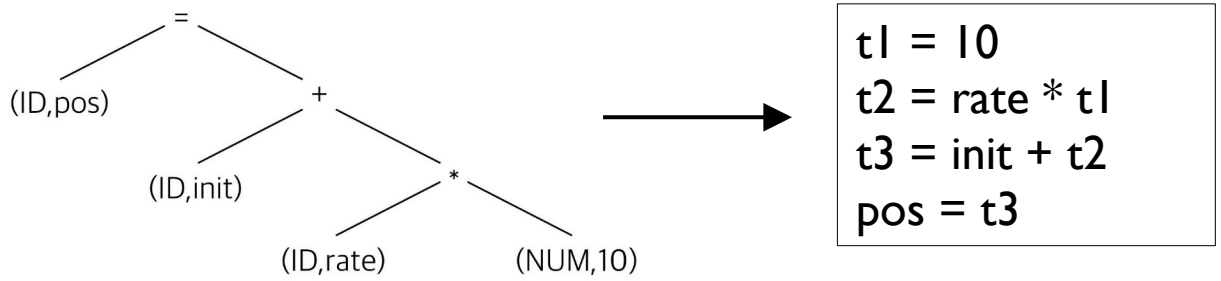
- Other semantic errors:
  - array out of bounds
  - null-dereference
  - divide-by-zero

# Language processing systems

## [IR Translator]



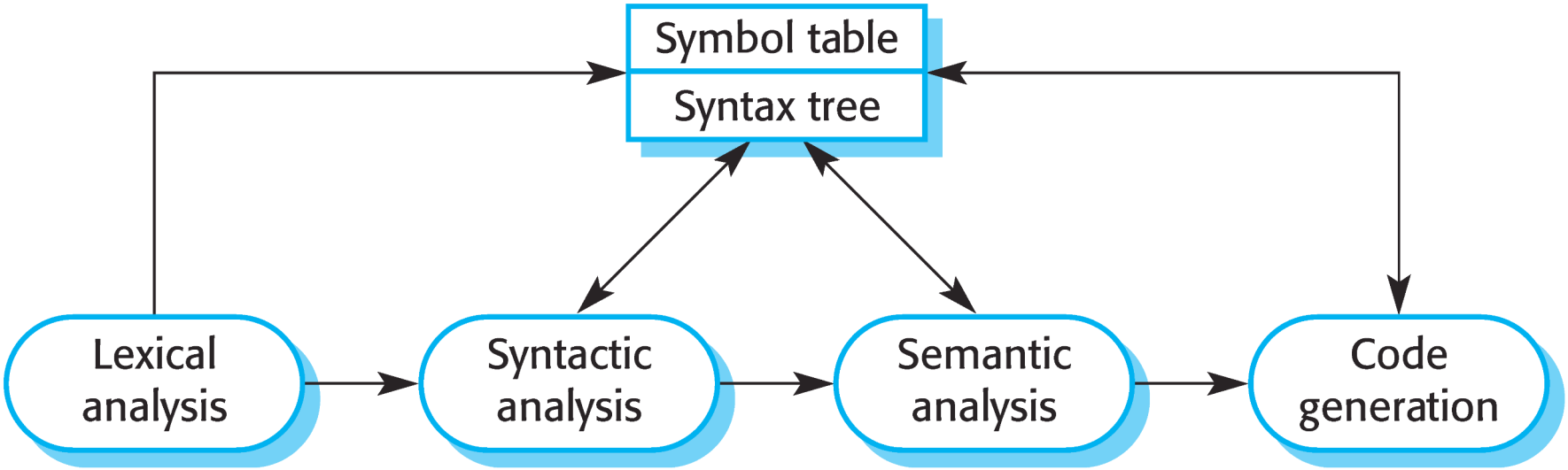

- Intermediate Representation (IR):
  - lower-level than the source language
  - higher-level than the target language
- ex) translate the syntax tree into three-address code:




```
t1 = 10
t2 = rate * t1
t3 = init + t2
pos = t3
```

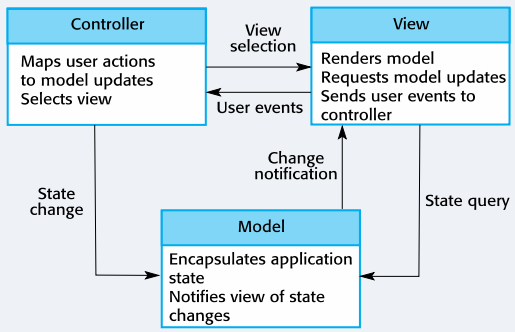
# Language processing systems

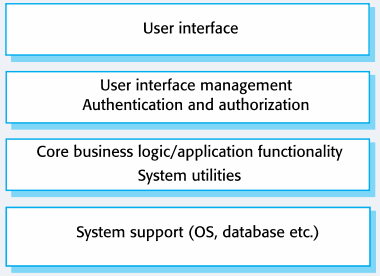
## [A pipe and filter compiler architecture]

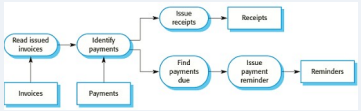


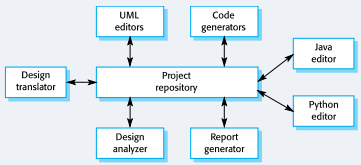
# Summary

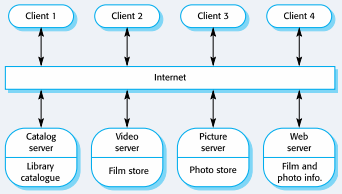
## Architectural patterns




## Application architectures