Week 6: Architectural Design

Nguyễn Thị Minh Tuyền

cdio

KHOA CÔNG NGHỆ T
TRƯỜNG ĐẠI HỌC K

TRUONG DAI HOC KHOA HOC TU NHIEN
KHTN
TP. HO CHI MINH

Adapted from slides of Ian Sommerville

Architectural Design

1. What is it?
2. Who does it?
3. Why is it important?
4. What are the steps?
5. What is the work product?
6. How do I ensure that I’ve done it right?

Topics covered

1. Architectural design decisions
2. Architectural views
3. Architectural patterns
4. Application architectures

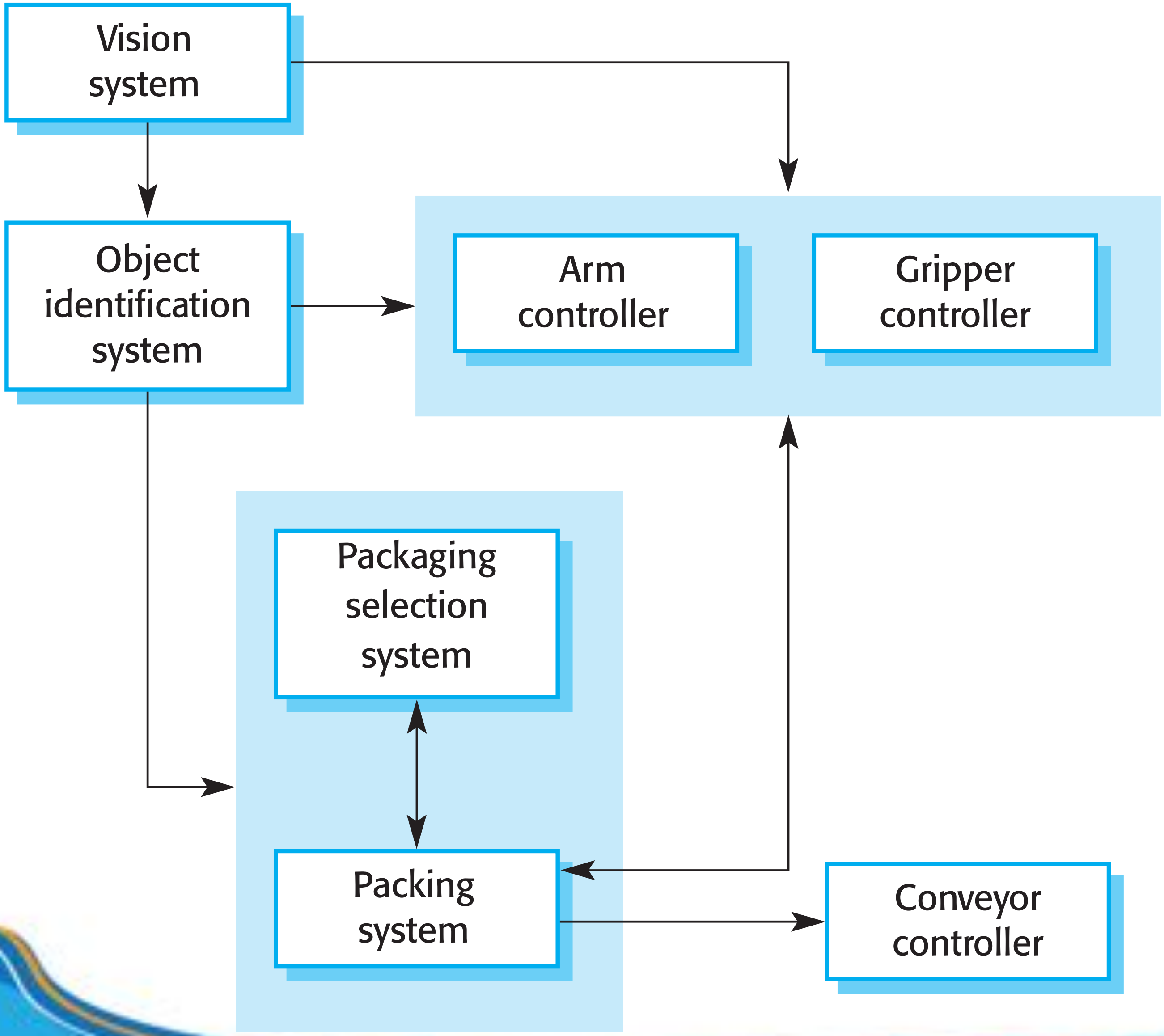
Software architecture

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

Architectural design

- □ Is an early stage of the system design process.
- □ Represents the critical link between specification and design processes.
- □ Often carried out in parallel with some specification activities.
- □ Involves identifying major system components and their communications.

Architecture of a packing robot control system



Architectural abstraction

- Architecture in the small is concerned with the architecture of individual programs.
  - At this level, we are concerned with the way that an individual program is decomposed into components.
- Architecture in the large is concerned with the architecture of complex enterprise systems that include other systems, programs, and program components.
  - These enterprise systems are distributed over different computers, which may be owned and managed by different companies.

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.

Architectural representations

- Simple, informal block diagrams showing entities and 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.

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.

Use of architectural models

- Facilitating discussion about the system design
  - A high-level architectural view of a system is useful for communication with system stakeholders and project planning because it is not cluttered with detail.
  - Stakeholders can relate to it and understand an abstract view of the system. They can then discuss the system as a whole without being confused by detail.
- Documenting an architecture that has been designed
  - The aim here is to produce a complete system model that shows the different components in a system, their interfaces and their connections.

Topics covered

1. Architectural design decisions
2. Architectural views
3. Architectural patterns
4. Application architectures

Architectural design decisions

- Architectural design is a creative process
  - So the process differs depending on the type of system being developed, the background and experience of the system architect, and the specific requirements.
- A number of common decisions span all design processes and these decisions affect the non-functional characteristics of the system.

Architectural design decisions

1. Is there a generic application architecture that can act as a template for the system that is being designed?
2. How will the system be distributed across hardware cores or processors?
3. What Architectural patterns or styles might be used?
4. What will be the fundamental approach used to structure the system?
5. How will the structural components in the system be decomposed into sub-components?
6. What strategy will be used to control the operation of the components in the system?
7. What architectural organization is best for delivering the non-functional requirements of the system?
8. How should the architecture of the system 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’.
  - An architectural pattern is a description of a system organization.
  - These capture the essence of an architecture that has been used in different software systems.

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.

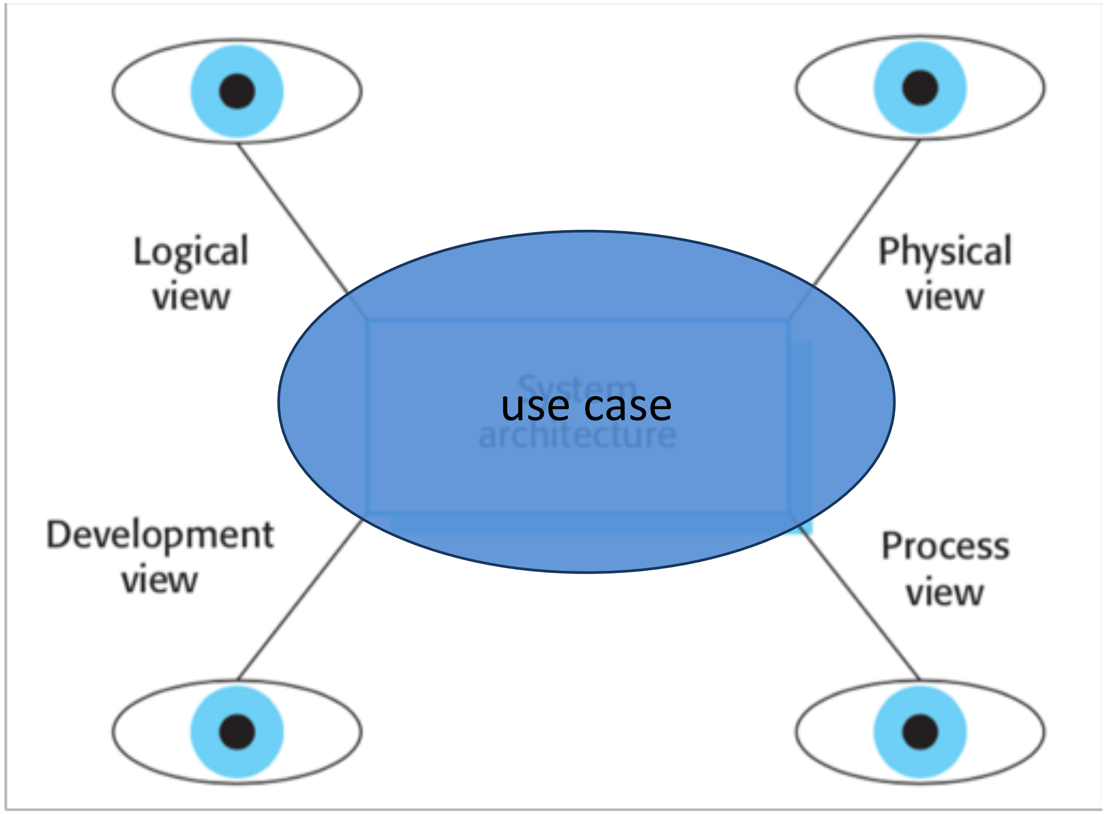
Topics covered

1. Architectural design decisions
2. Architectural views
3. Architectural patterns
4. Application architectures

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.

Architectural model 4 + 1



Topics covered

1. Architectural design decisions
2. Architectural views
3. Architectural patterns
4. Application architectures

Architectural patterns

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

Examples of patterns

□ Model-View-Controller (MVC) pattern

□ Layered architecture pattern

□ Repository pattern

□ Client–server pattern

□ Pipe and filter pattern

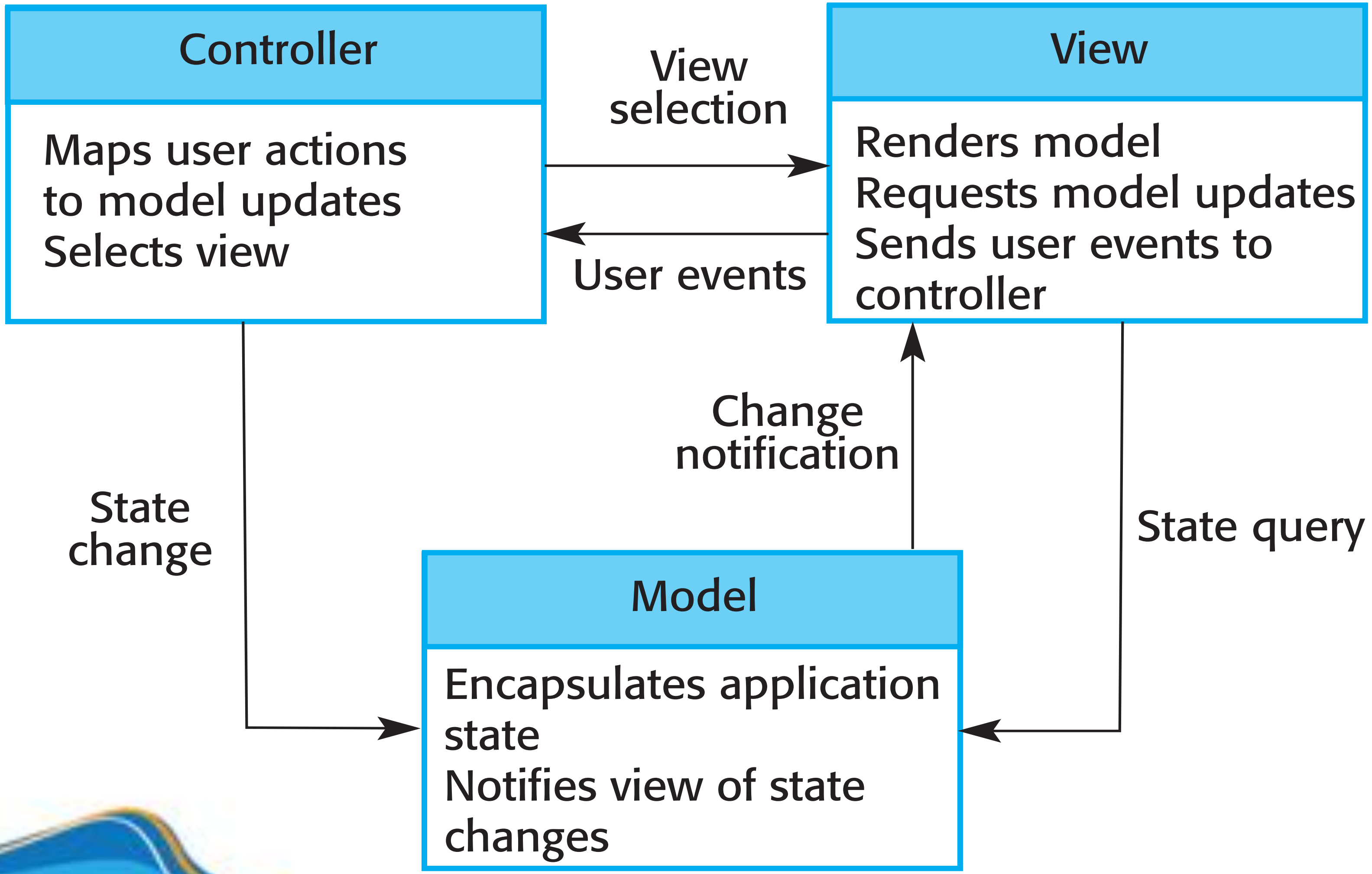
Model-View-Controller (MVC) pattern

- Description
  - Separates presentation and interaction from the system data.
  - 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.
- Is used when
  - There are multiple ways to view and interact with data.
  - The future requirements for interaction and presentation of data are unknown.

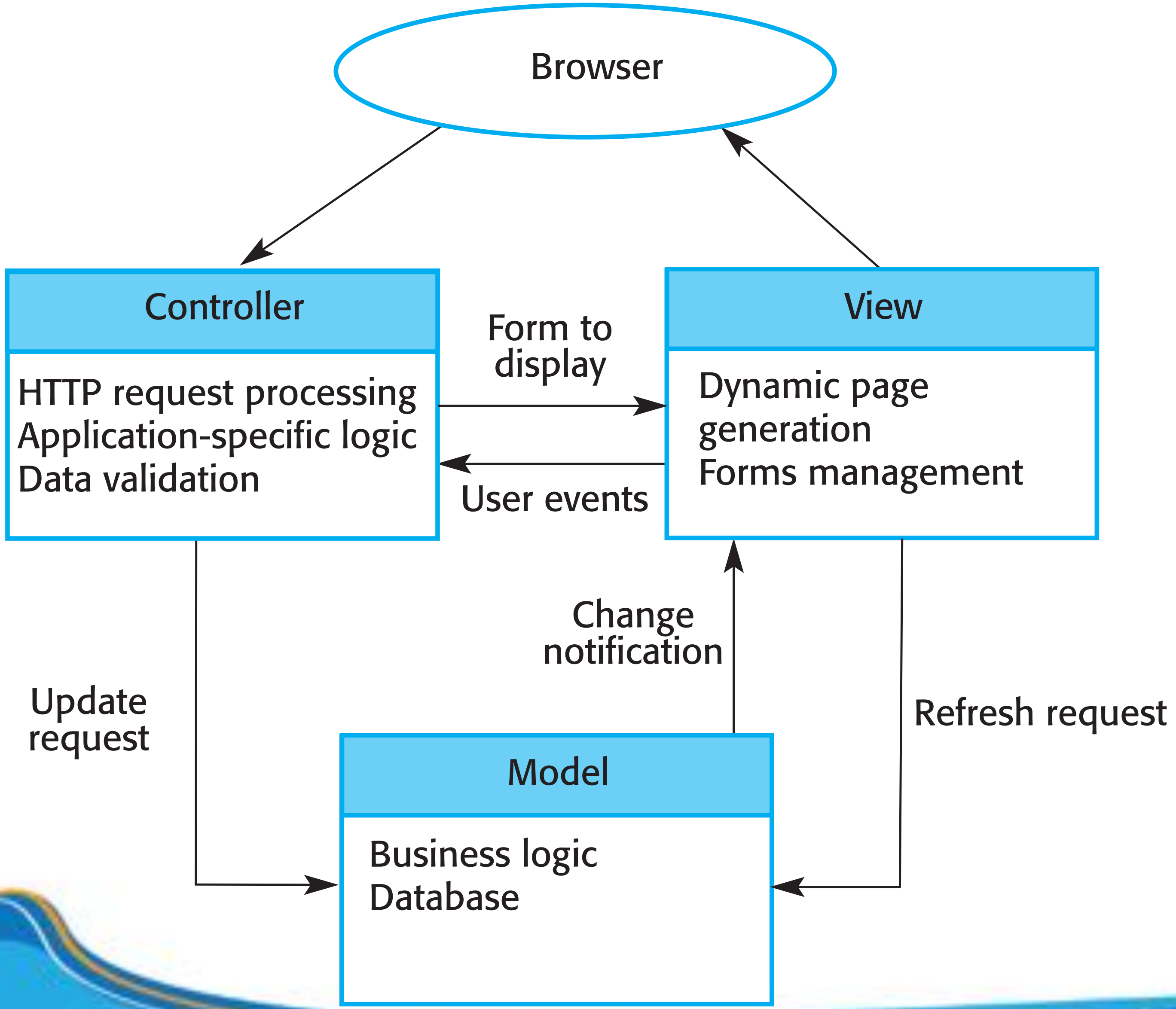
Model-View-Controller (MVC) pattern

- 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.

Organization of the MVC



Web application architecture using the MVC pattern



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.

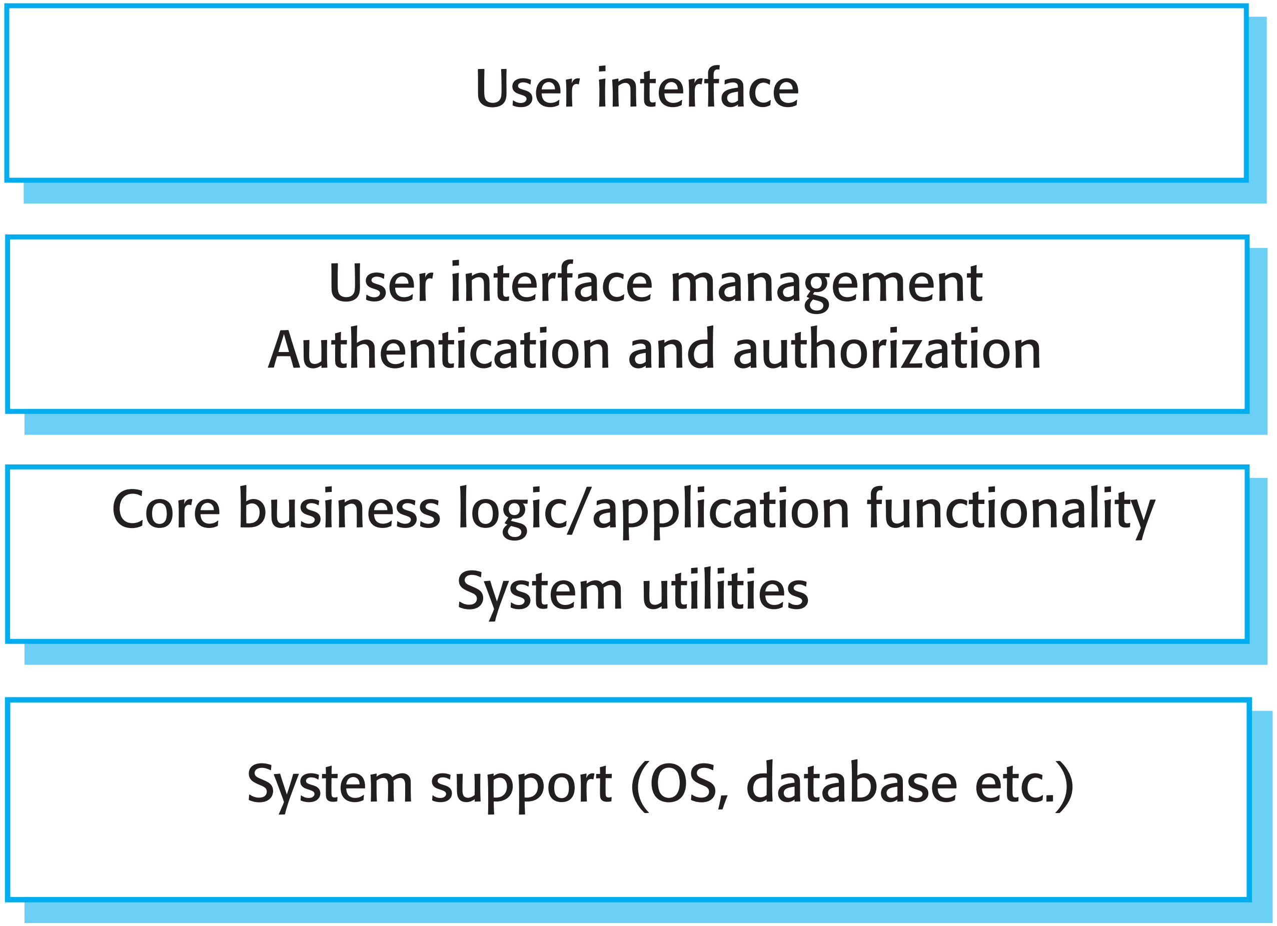
Layered architecture pattern

- 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.
- Used when
  - Building new facilities on top of existing systems;
  - The development is spread across several teams with each team responsibility for a layer of functionality;
  - There is a requirement for multi-level security.

Layered architecture pattern

- 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



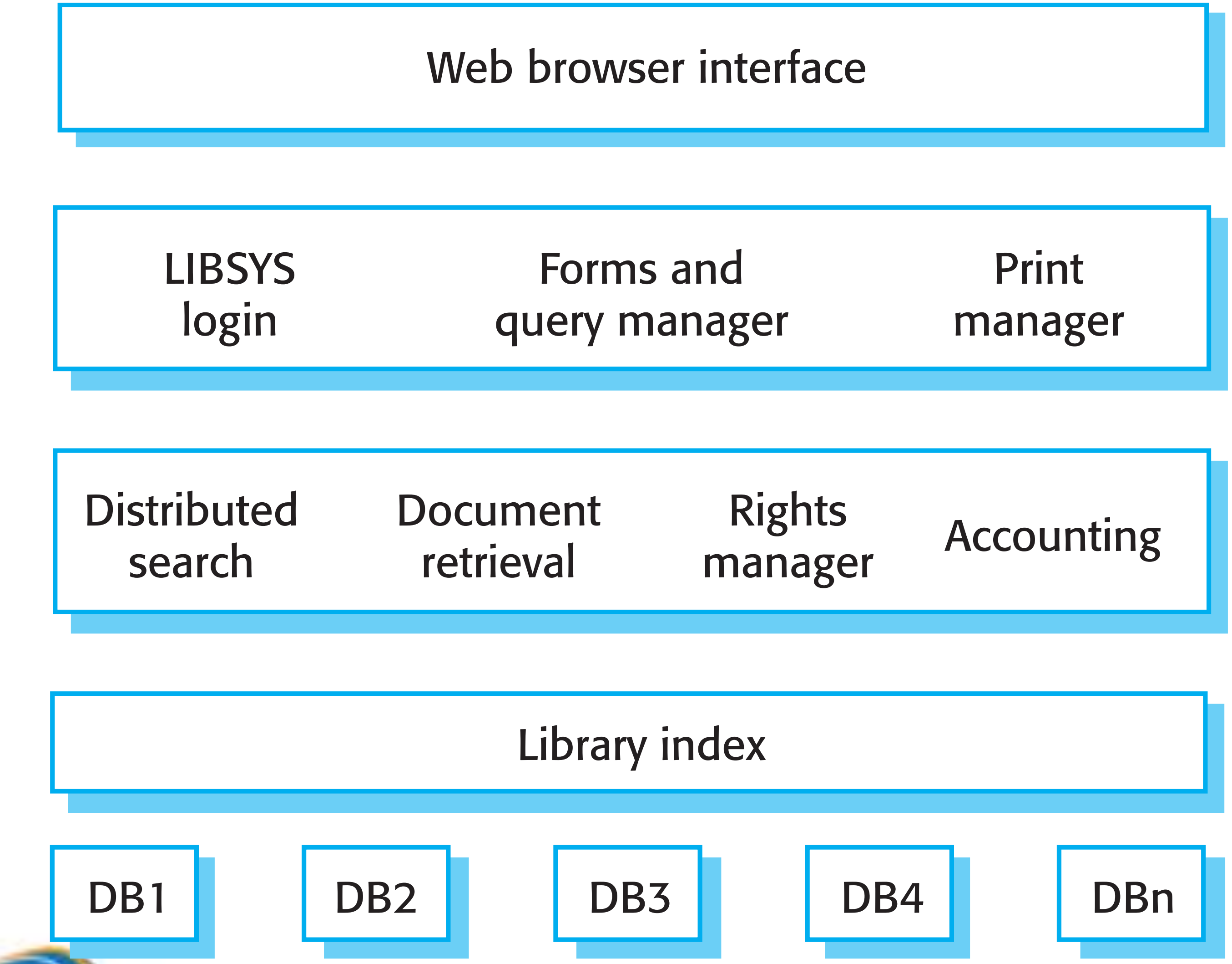
User interface

User interface management
Authentication and authorization

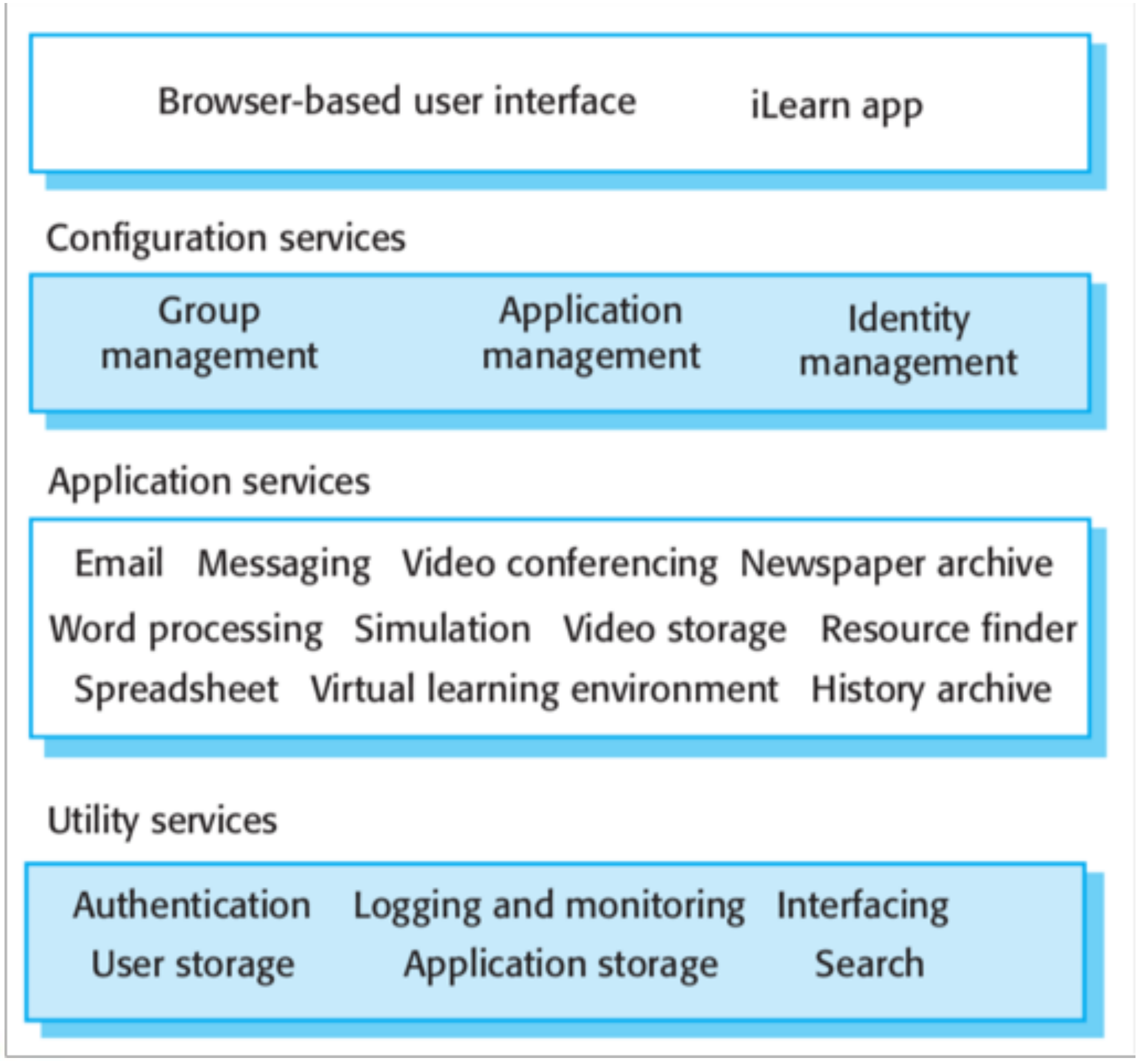
Core business logic/application functionality
System utilities

System support (OS, database etc.)

Architecture of the LIBSYS system



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.

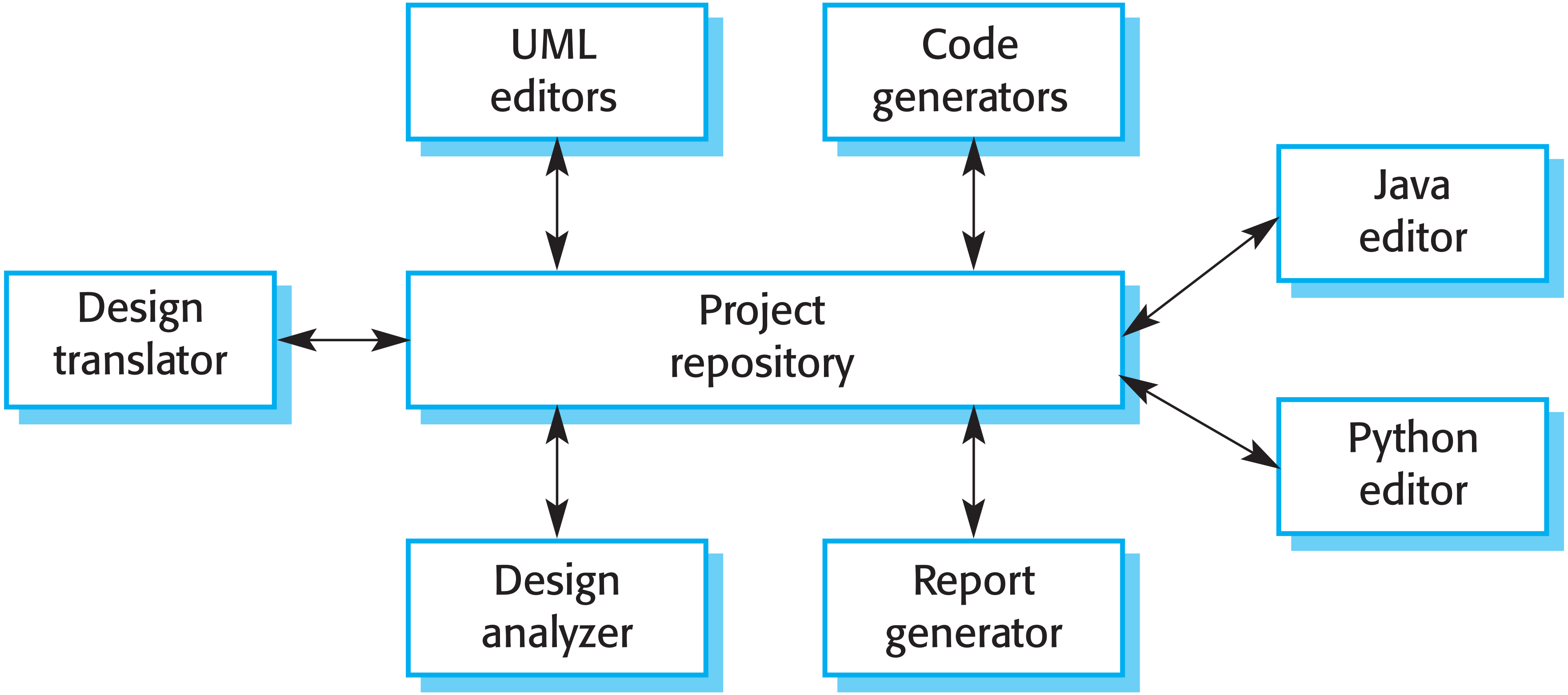
Repository pattern

- 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.
- Used 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.

Repository pattern

- 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



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.
- Set of clients which call on these services.
- Network which allows clients to access servers.

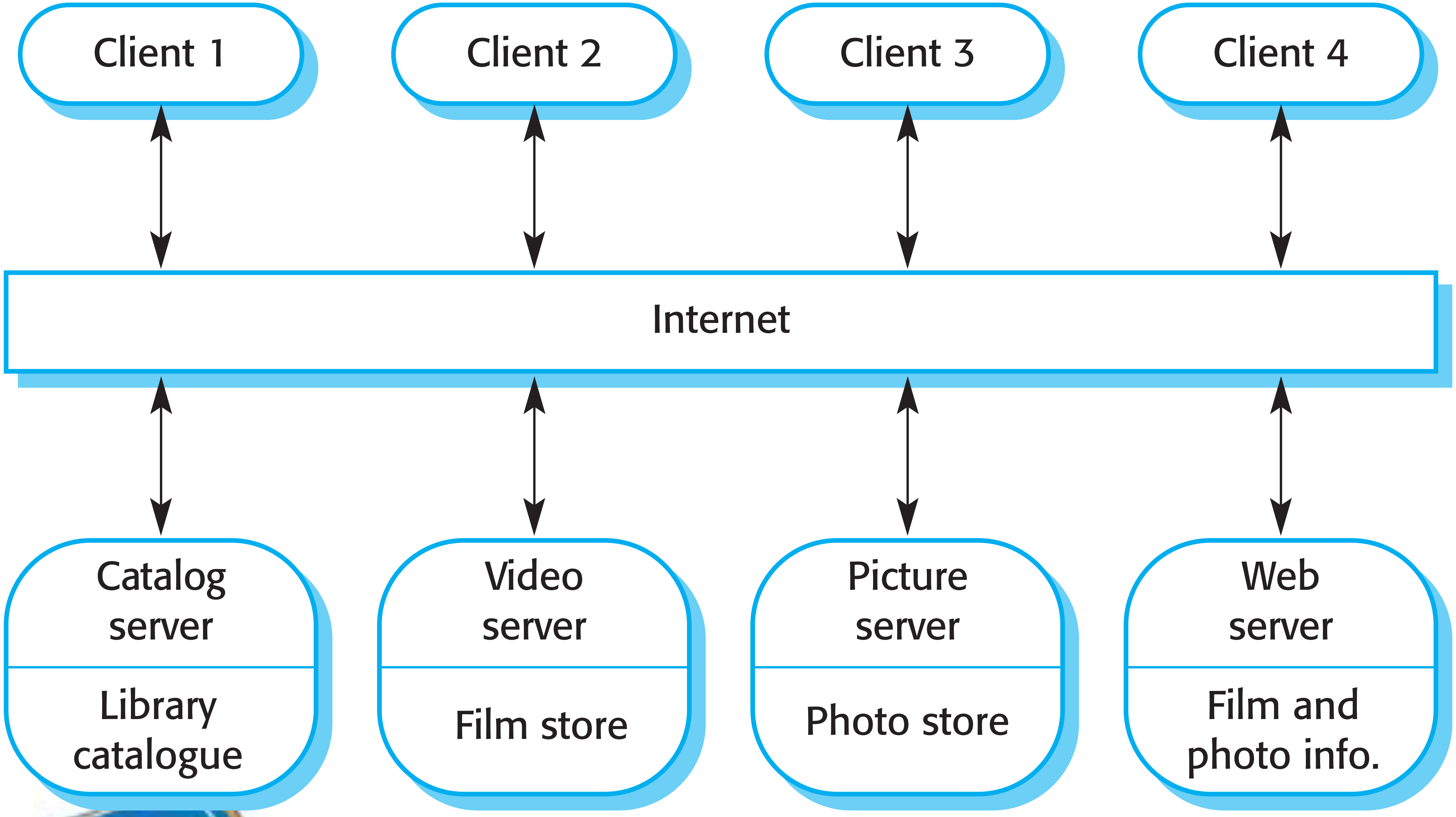
Client–server pattern

- 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.
- Used when
  - Data in a shared database has to be accessed from a range of locations.
  - Load on a system is variable.

Client–server pattern

- Advantages
  - Servers can be distributed across a network.
  - General functionality 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



Pipe and filter architecture

- Functional transformations process their inputs to produce outputs.
- May be referred to as a pipe and filter model.
- Variants of this pattern are very common. When transformations are sequential, this is a batch sequential model which is extensively used in data processing systems.
- Not really suitable for interactive systems.

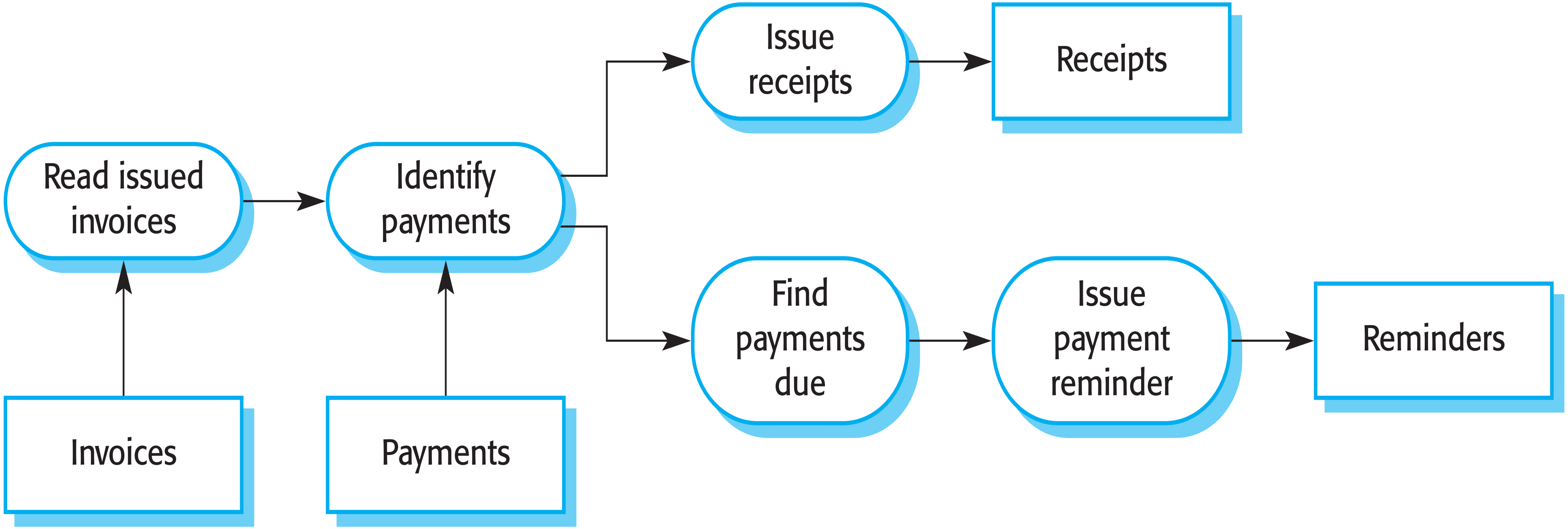
Pipe and filter pattern

- 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.
- Used when
  - Commonly used in data processing applications (both batch- and transaction-based) where inputs are processed in separate stages to generate related outputs.

Pipe and filter pattern

- 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.

Example of the pipe and filter architecture



Topics covered

- □ Architectural design decisions
- □ Architectural views
- □ Architectural patterns
- □ 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 organizing 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

- Data processing applications
  - Data driven applications that process data in batches without explicit user intervention during the processing.
- Transaction processing applications
  - Data-centered 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.

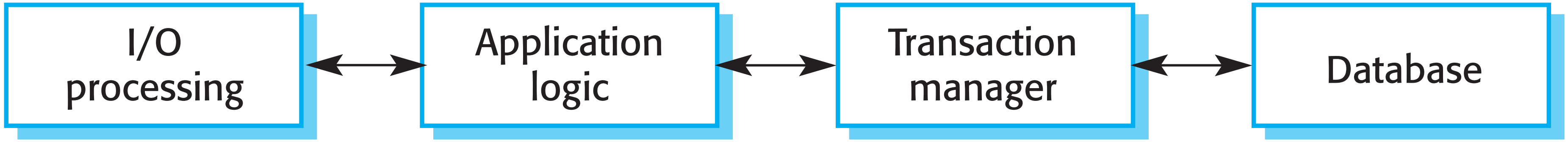
Application type examples

- Focus here is on transaction processing and language processing systems.
- Transaction processing systems
  - E-commerce systems;
  - Reservation systems.
- Language processing systems
  - Compilers;
  - Command interpreters.

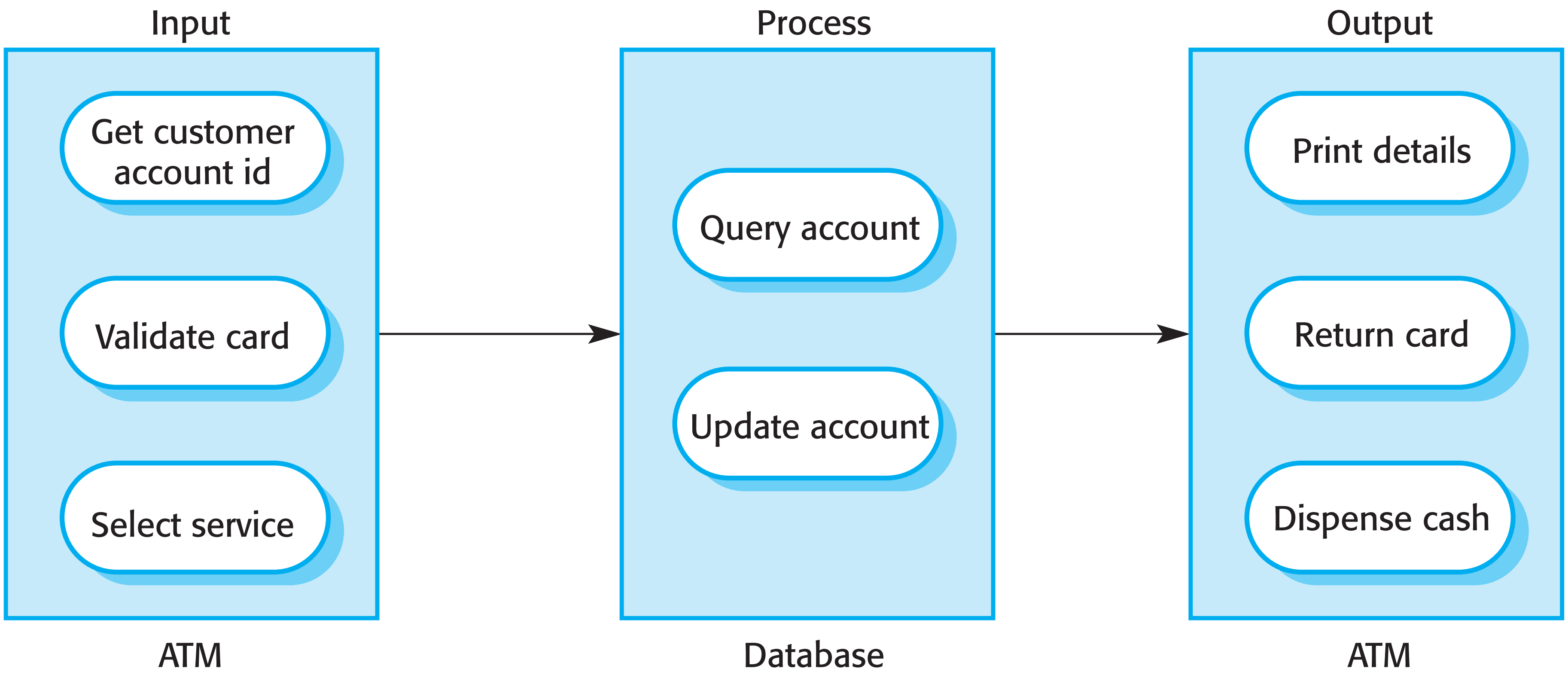
Transaction processing systems

- 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.

Structure of transaction processing applications



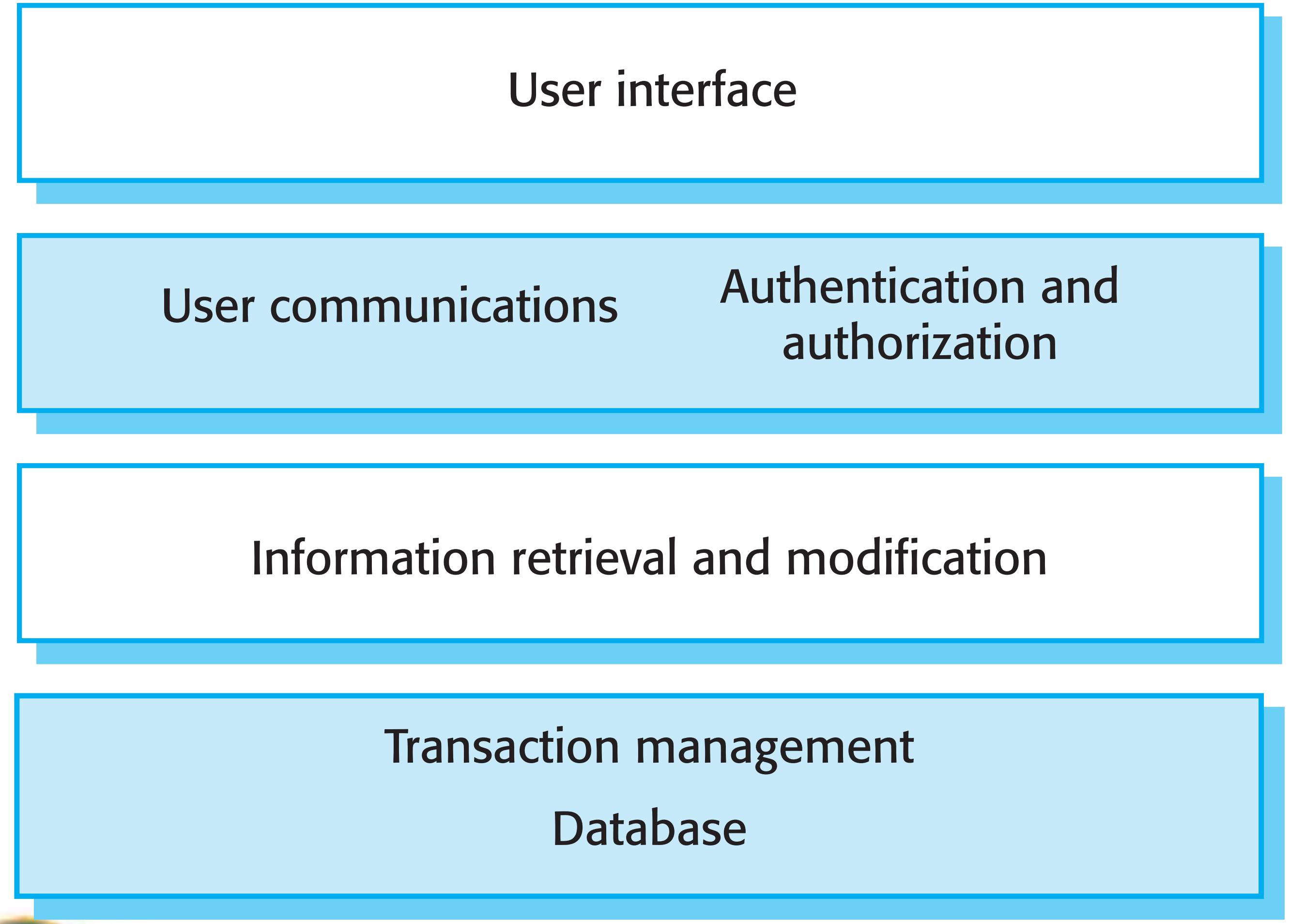
Software architecture of an ATM system



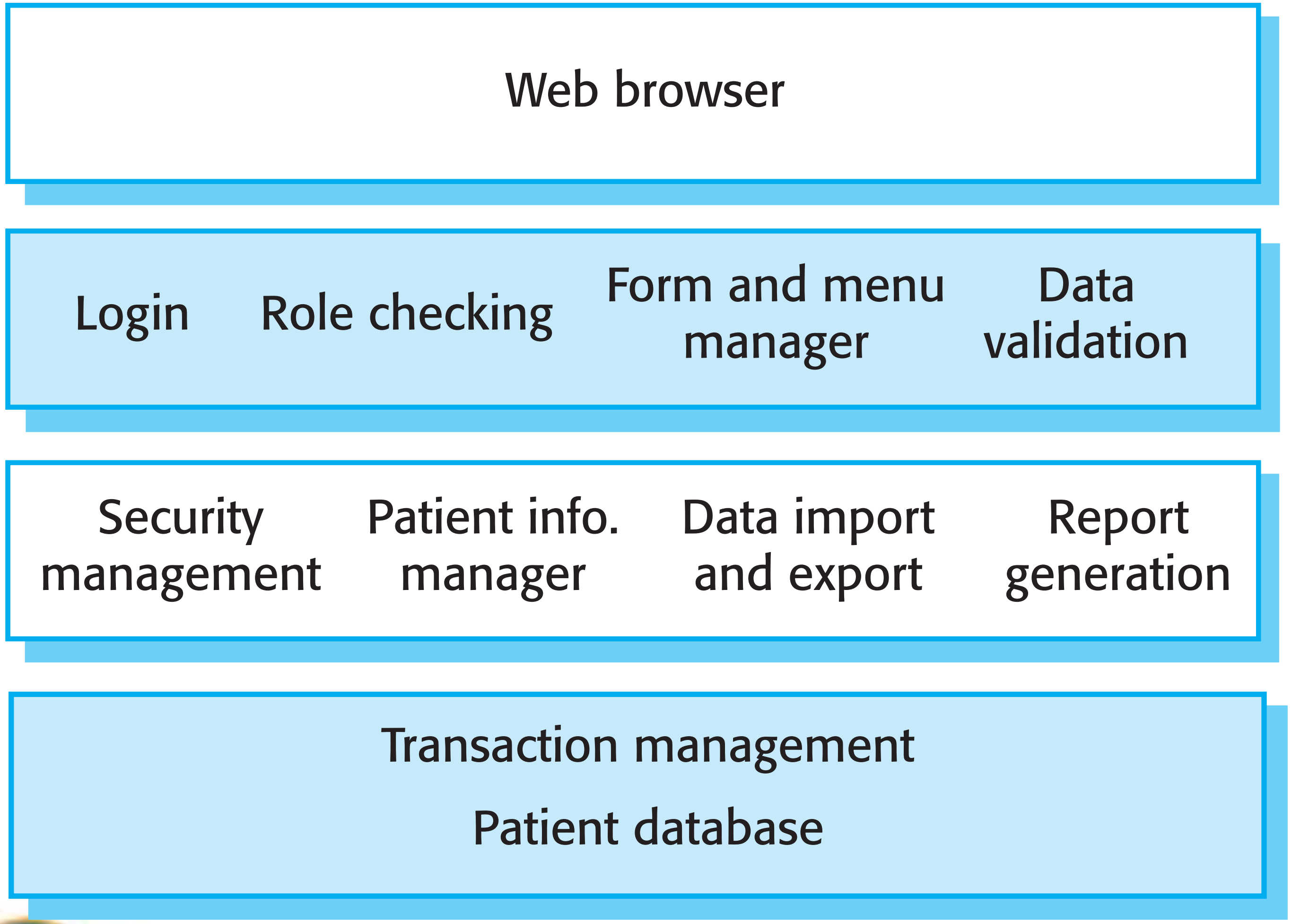
Information systems architecture

- 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

Layered information system architecture


User interface
User communications
Authentication and authorization
Information retrieval and modification
Transaction management
Database

Architecture of the Mentcare



Web browser

Login | Role checking | Form and menu manager | Data validation

Security management | Patient info. manager | Data import and export | Report generation

Transaction management

Patient database

Web-based information systems

- Information and resource management systems are now usually web-based systems
  - user interfaces are implemented using a web browser.
- Example:
  - E-commerce systems are Internet-based resource management systems that accept electronic orders for goods or services and then arrange delivery of these goods or services to the customer.
  - In an e-commerce system, the application-specific layer includes additional functionality supporting a ‘shopping cart’ in which users can place a number of items in separate transactions, then pay for them all together in a single transaction.

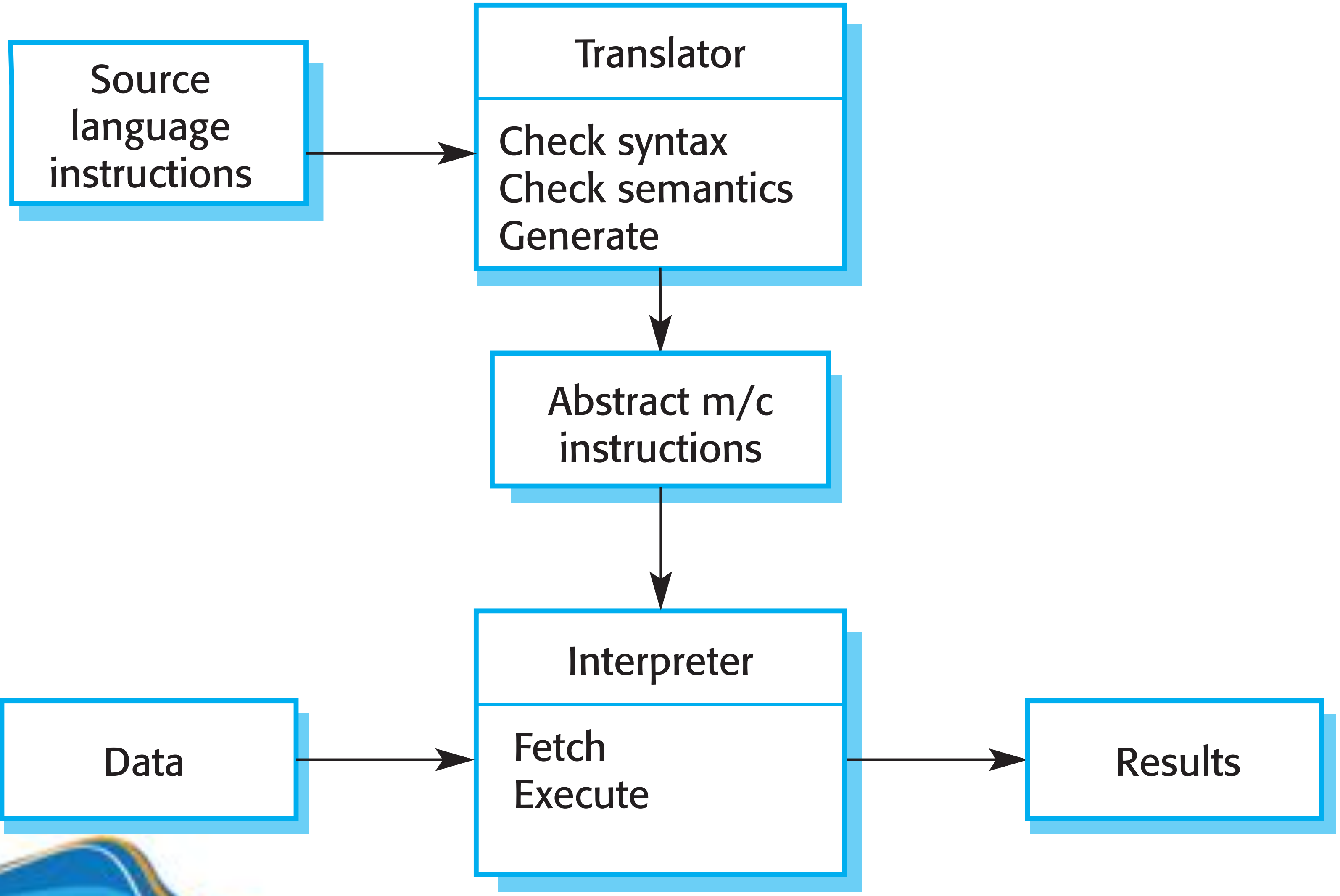
Server implementation

- These systems are 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.

Architecture of a language processing system



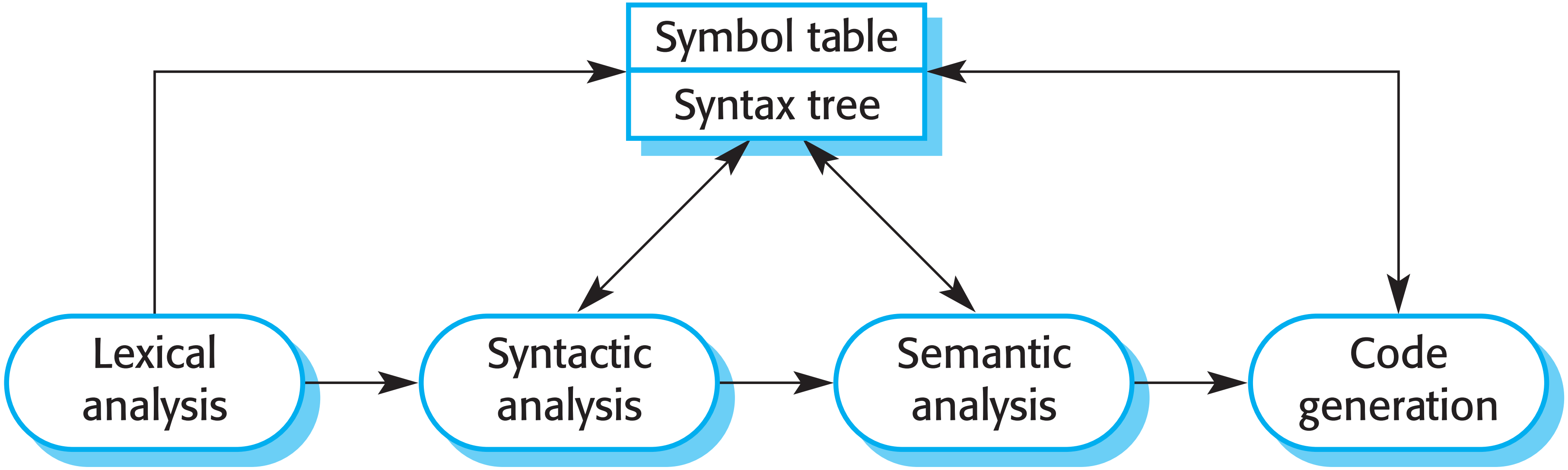
Compiler components

- 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
  - Is an internal structure representing the program being compiled.

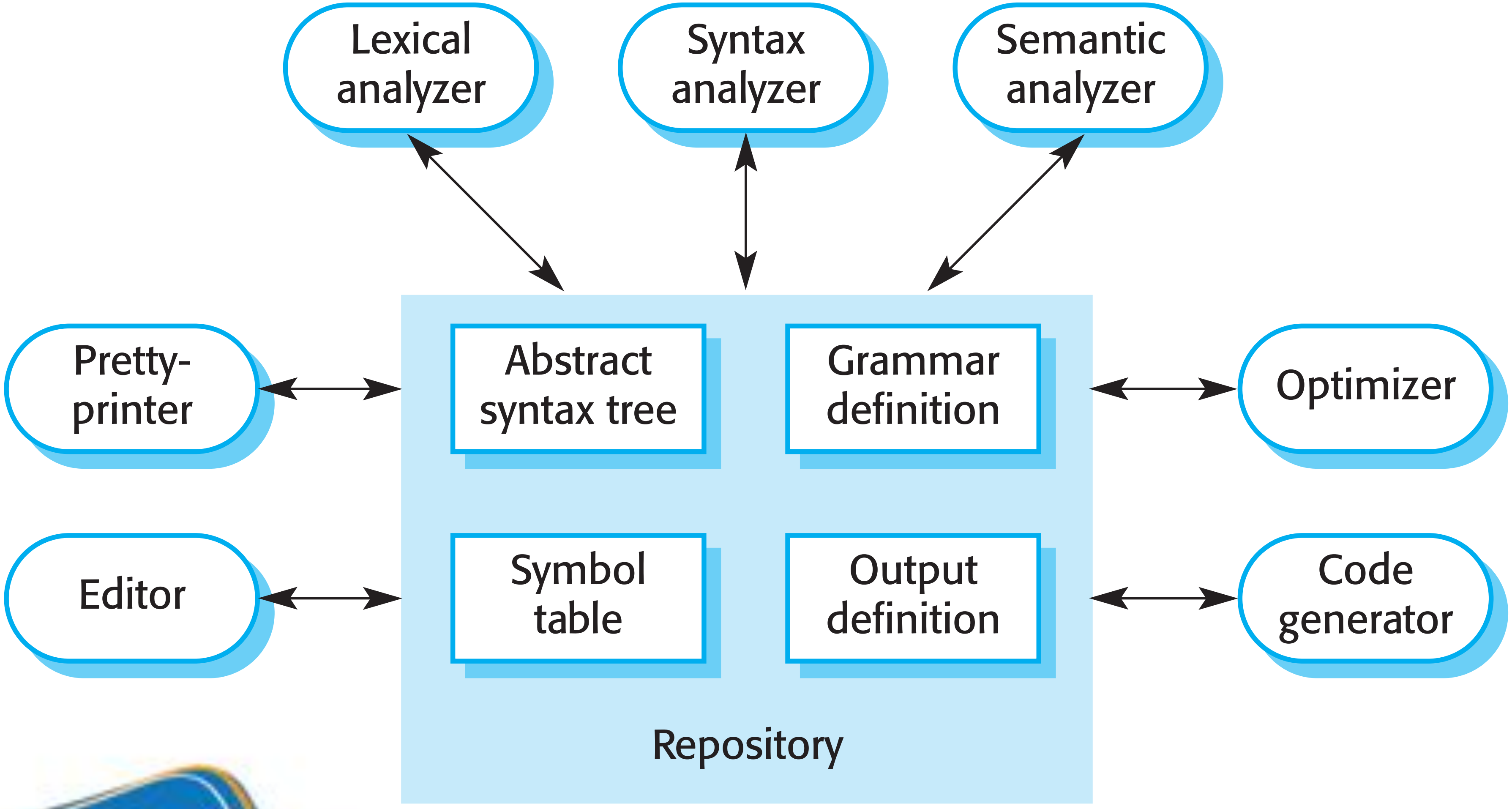
Compiler components

- 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 compiler architecture



A repository architecture for a language processing system


Lexical analyzer
Syntax analyzer
Semantic analyzer
Pretty-printer
Abstract syntax tree
Grammar definition
Optimizer
Editor
Symbol table
Output definition
Code generator
Repository