Architectural Design

Eunseok Lee, Prof.
School of Software
Sungkyunkwan University

Objectives

- To introduce architectural design and to discuss its importance
- To explain the architectural design decisions that have to be made
- To introduce architectural views and architectural patterns
- To introduce some application architectures

Topics covered

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

Appendix

A-1. Control styles

A-2. Reference architectures

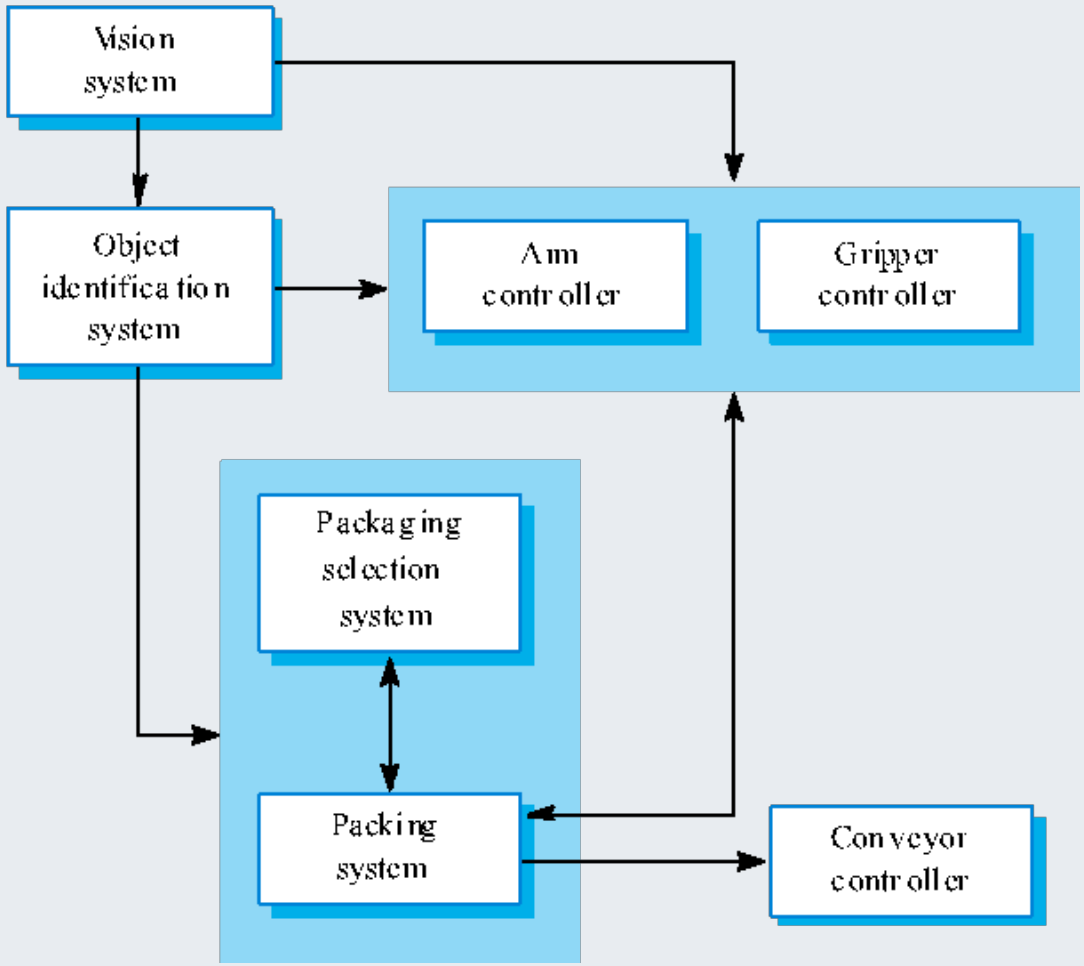
Software architecture

- 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

- Architectural design is concerned with understanding how a software system should be organized and designing the overall structure of that system.
- Architectural design is the critical link between design and requirements engineering, as it identifies the main structural components in a system and the relationships between them.
- The output of the architectural design process is an architectural model that describes how the system is organized as a set of communicating components.

Packing robot control system



Advantages of explicit architecture

(Bass, et al., 2003)

- 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. *Performance*, *reliability*, etc.
- Large-scale reuse
  - The architecture may be reusable across a range of related systems with similar requirements.

Architecture and system characteristics

(Bosch, 2000)

- Performance
  - Localize critical operations within a small number of subsystems and minimize communications. Use large rather than fine-grain components.
- Security
  - Use a layered architecture, with critical assets protected in the inner layers and with high level of security validation to these.
- 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 conflicts

- Using large-grain components improves *performance* but reduces *maintainability*.
- Introducing redundant data improves *availability* but makes *security* more difficult.
- Localizing *safety*-related features usually means more communication so degraded *performance*.

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.
- Depends on the use of architectural models. The requirements for model semantics depends on how the models are used.

Architectural Description Language: C2

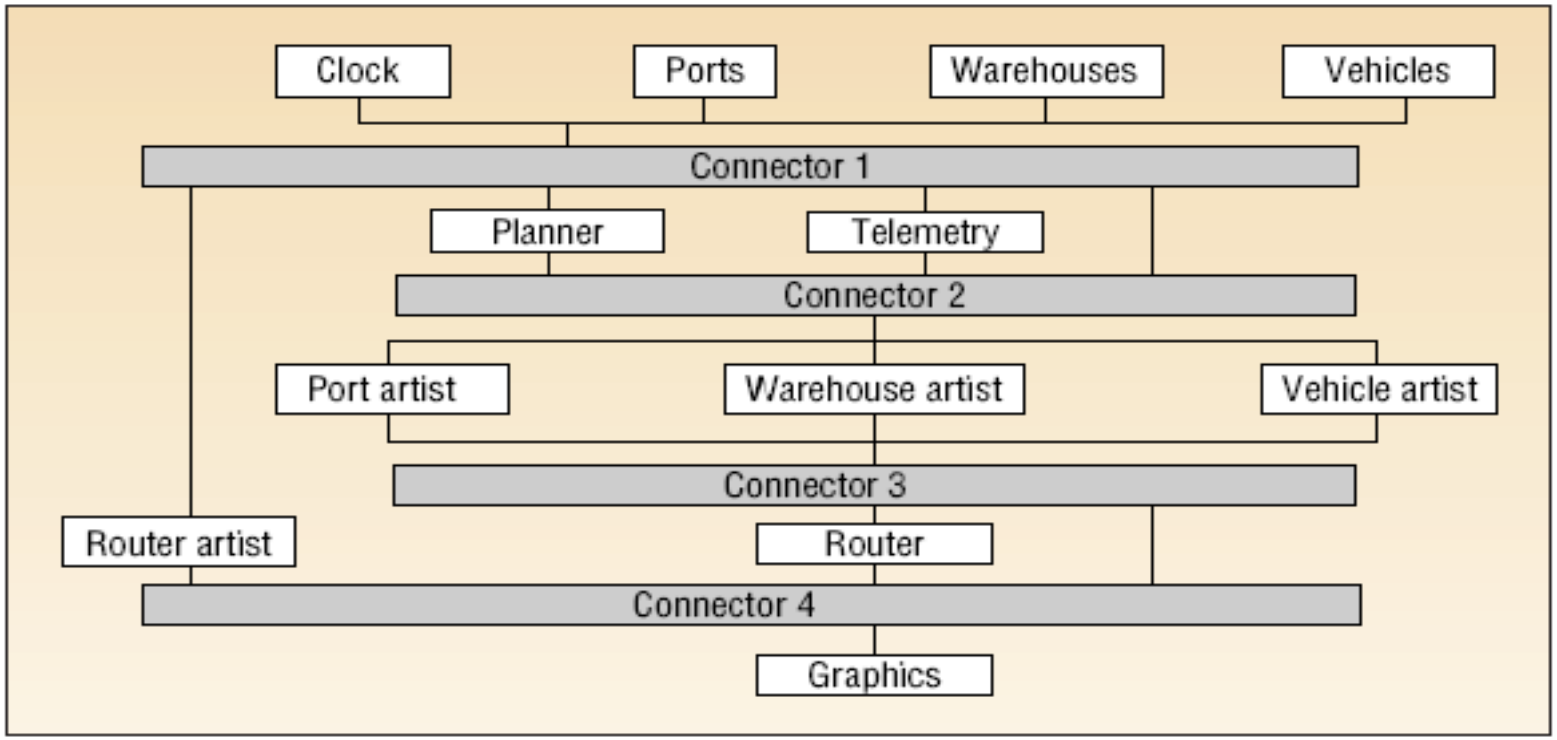


Figure 4. A C2-style architecture for a simple cargo-routing logistics system. Ports, vehicles, and warehouses are components that store application state. The telemetry component tracks en route cargo shipments. The port artist, vehicle artist, warehouse artist, and router artist components graphically depict the state of their respective counterparts. The planner component uses simple heuristics to suggest cargo routes, and the router component handles routing requests initiated by the end user. The graphics component renders the drawing notifications sent from the artists on the end-user's display.

Architectural Description Language: Weave

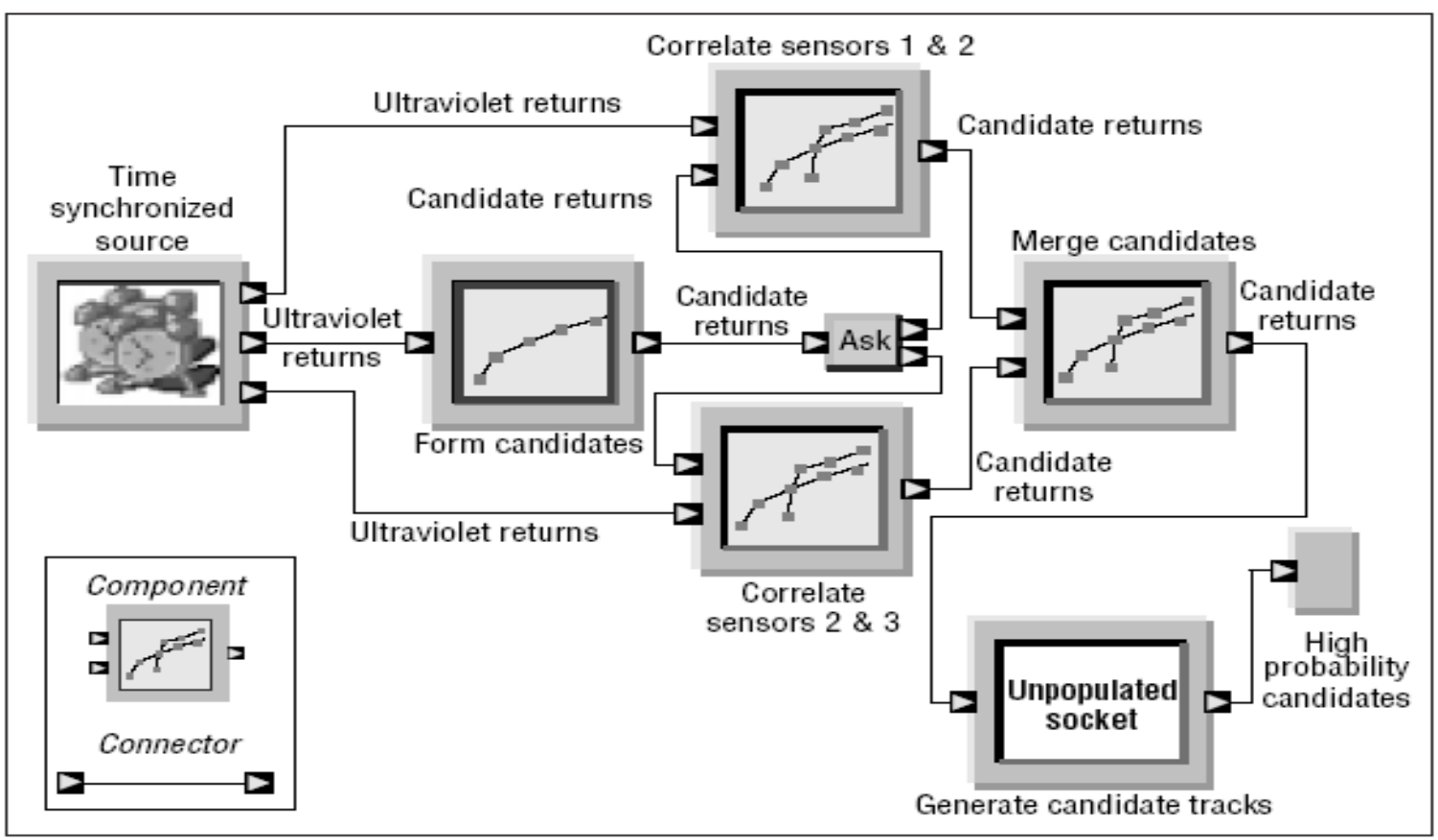
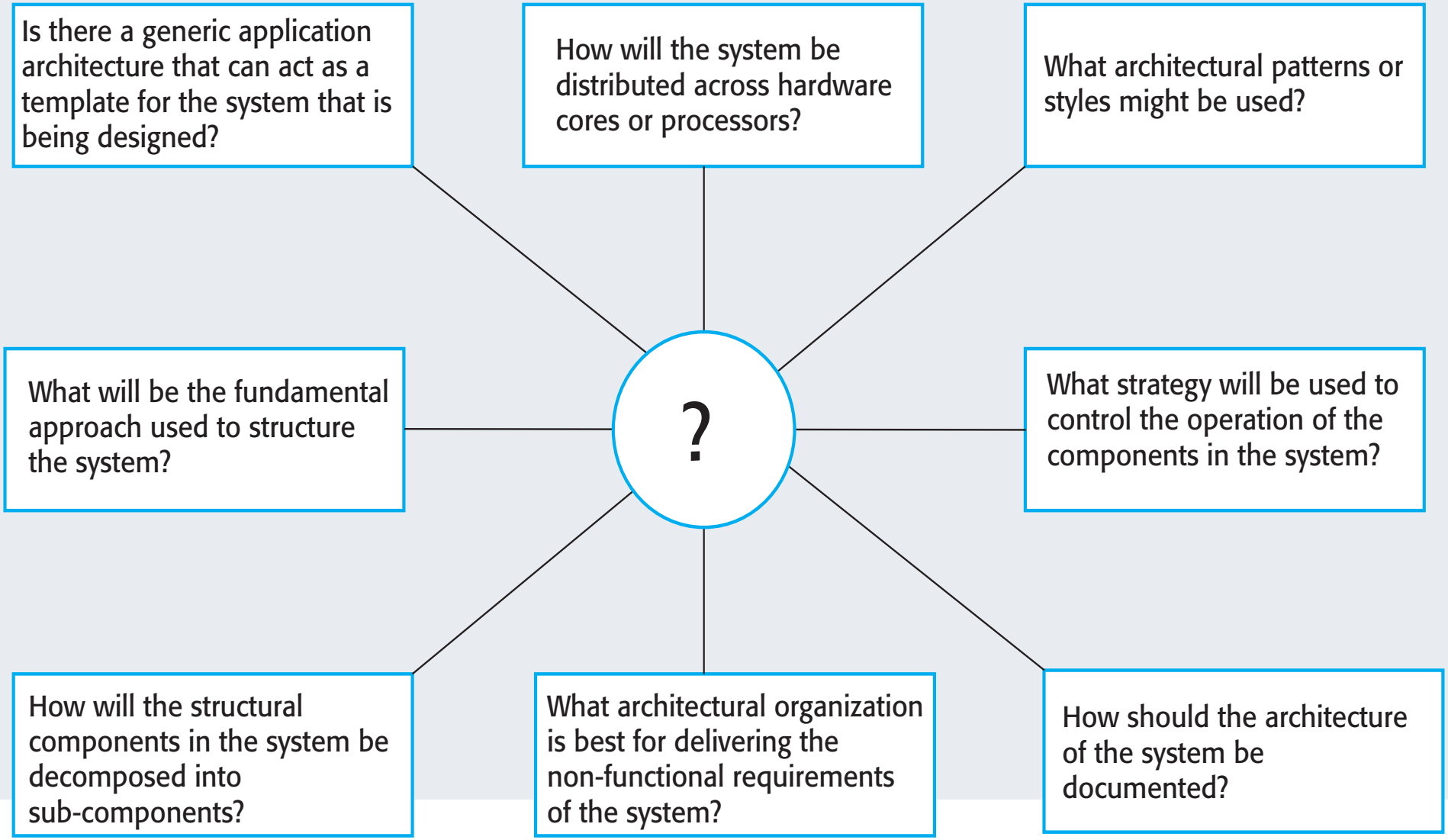


Figure 5. A portion of a Weaves architecture for a stereo-tracking system.

1. 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



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.

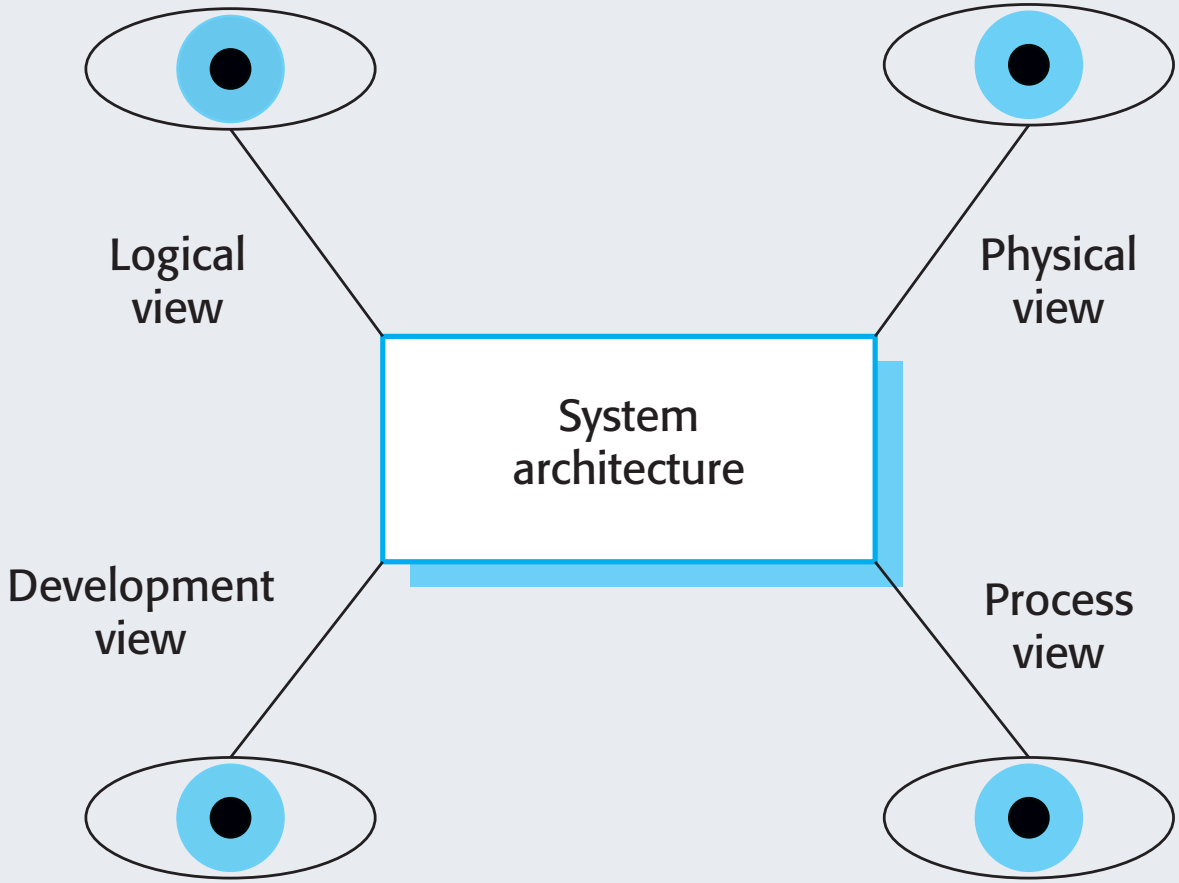
Architectural styles

- A pattern of system organization (David Garlan, 1993)
- The architectural model of a system may conform to a generic architectural model or style such as client-server or layered architecture.
- An awareness of these styles can simplify the problem of defining system architectures.
- However, most large systems are heterogeneous and do not follow a single architectural style.

2. 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 views (4+1 view model)



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 Design

Part 2

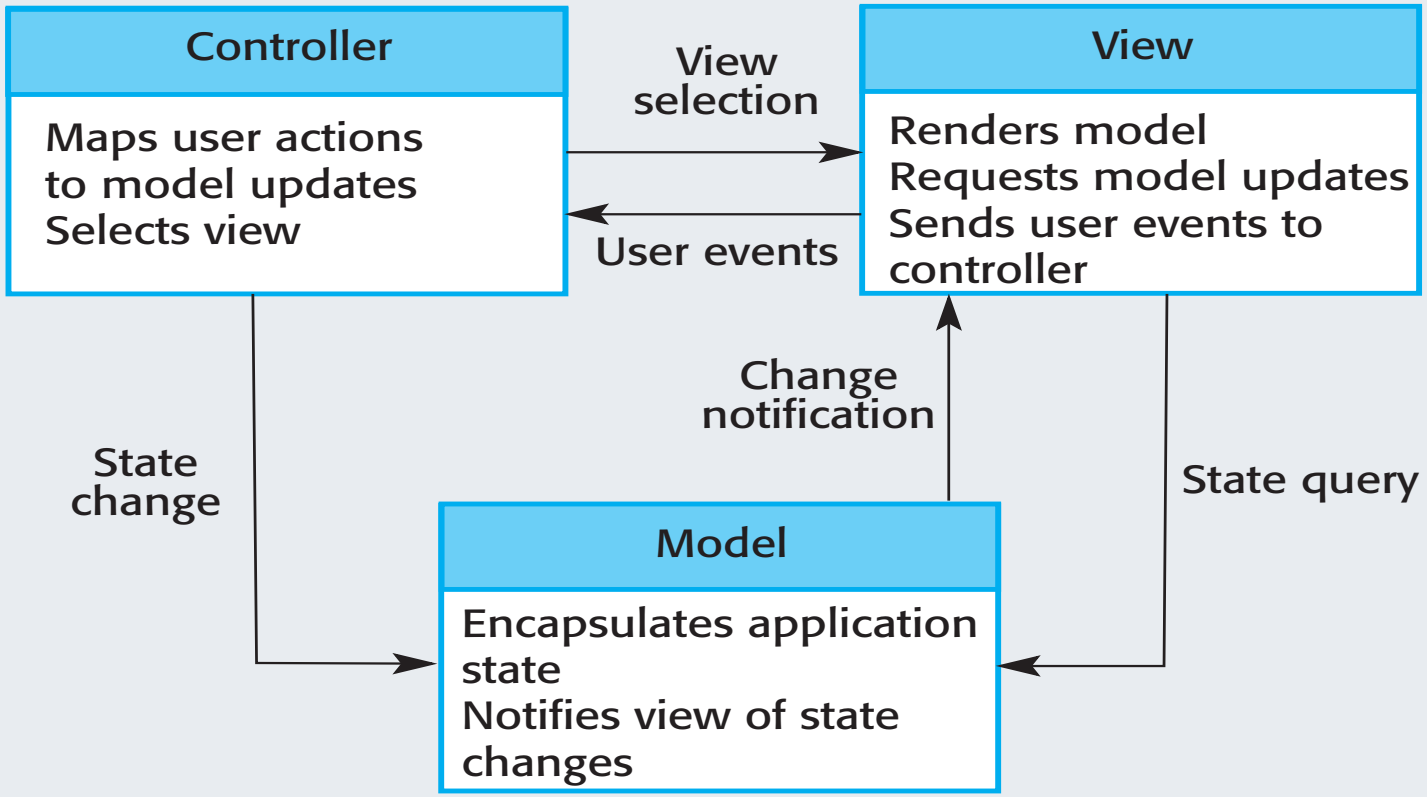
3. Architectural patterns

- 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.
- Patterns may be represented using tabular and graphical descriptions.

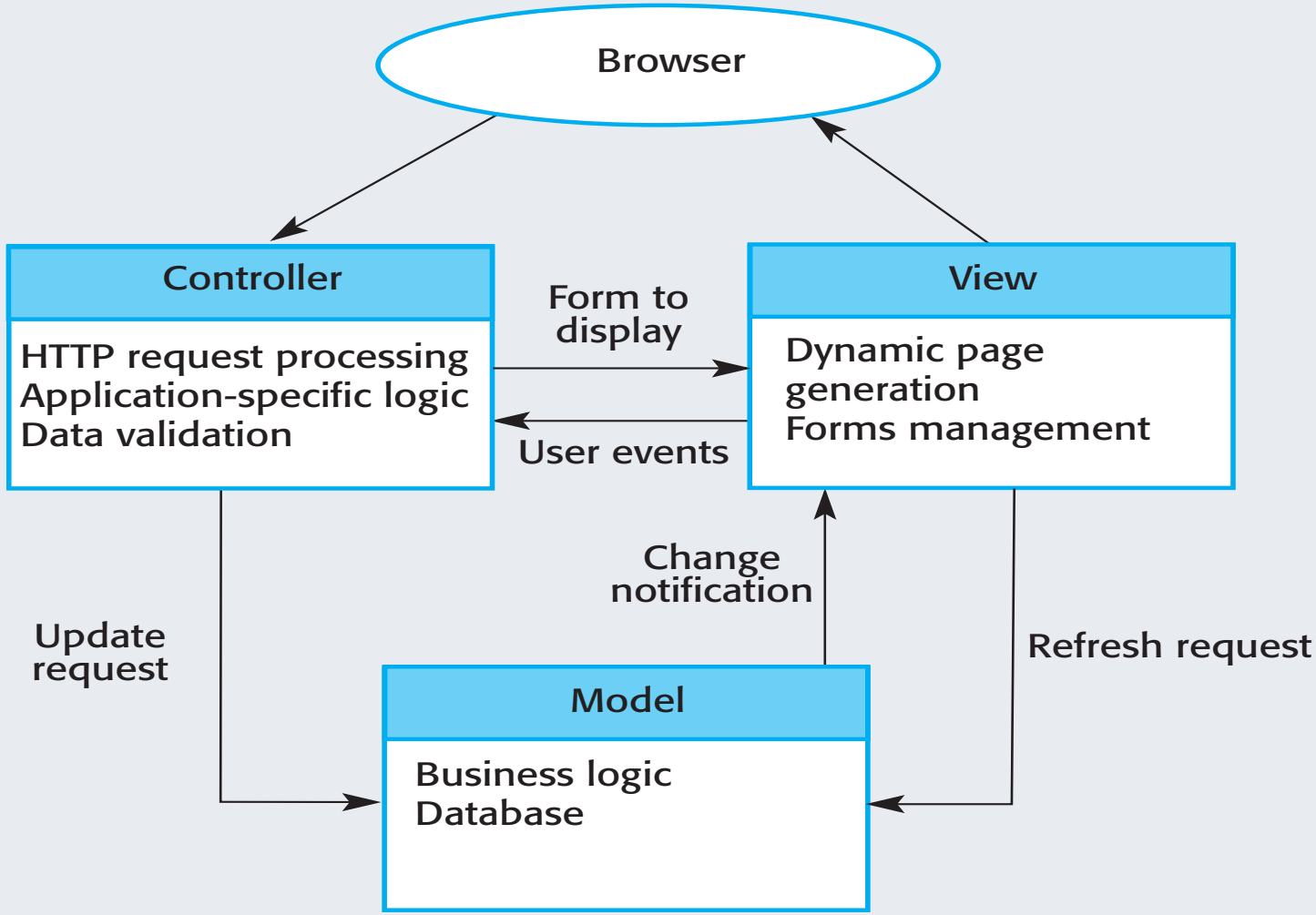
The Model-View-Controller (MVC) pattern

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 6.4 shows the architecture of a web-based application system organized using the MVC pattern.
When used	Used when there are multiple ways to view and interact with data. Also used when the future requirements for interaction and presentation of data are unknown.
Advantages	Allows the data to change independently of its representation and vice versa. Supports presentation of the same data in different ways with changes made in one representation shown in all of them.
Disadvantages	Can involve additional code and code complexity when the data model and interactions are simple.

The organization of the Model-View-Controller



Web application architecture using the MVC pattern



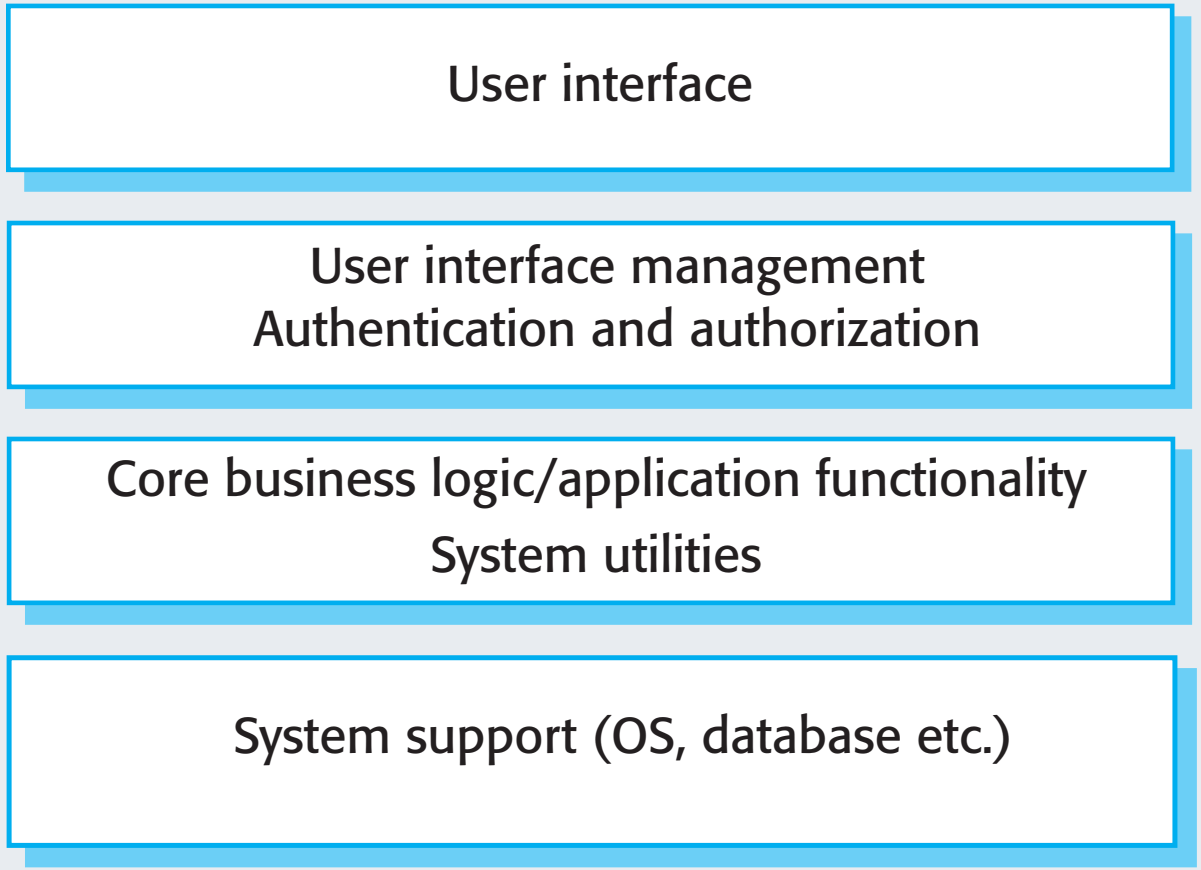
Layered architecture

- 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.
- However, often artificial to structure systems in this way.

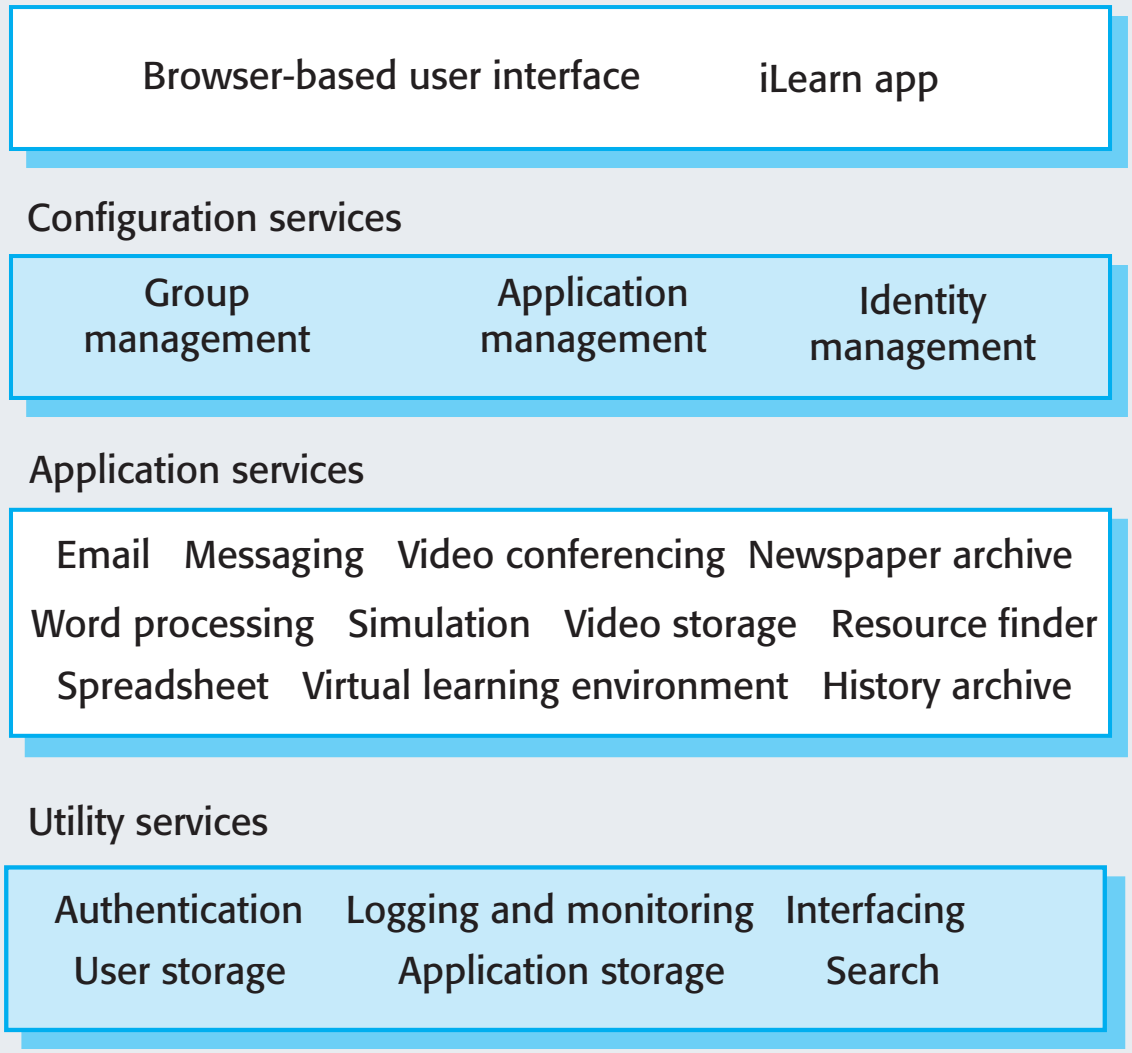
The Layered architecture pattern

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 lo west-level layers represent core services that are likely to be used thro ughout the system. See Figure 6.6.
Example	A layered model of a system for sharing copyright documents held in di fferent libraries, as shown in Figure 6.7.
When used	Used when building new facilities on top of existing systems; when the development is spread across several teams with each team responsi bility for a layer of functionality; when there is a requirement for multi-le vel security.
Advantages	Allows replacement of entire layers so long as the interface is maintain ed. Redundant facilities (e.g., authentication) can be provided in each l ayer to increase the dependability of the system.
Disadvantages	In practice, providing a clean separation between layers is often difficul t and a high-level layer may have to interact directly with lower-level la yers rather than through the layer immediately below it. Performance c an be a problem because of multiple levels of interpretation of a servic e request as it is processed at each layer.

A generic layered architecture



The architecture of the iLearn system



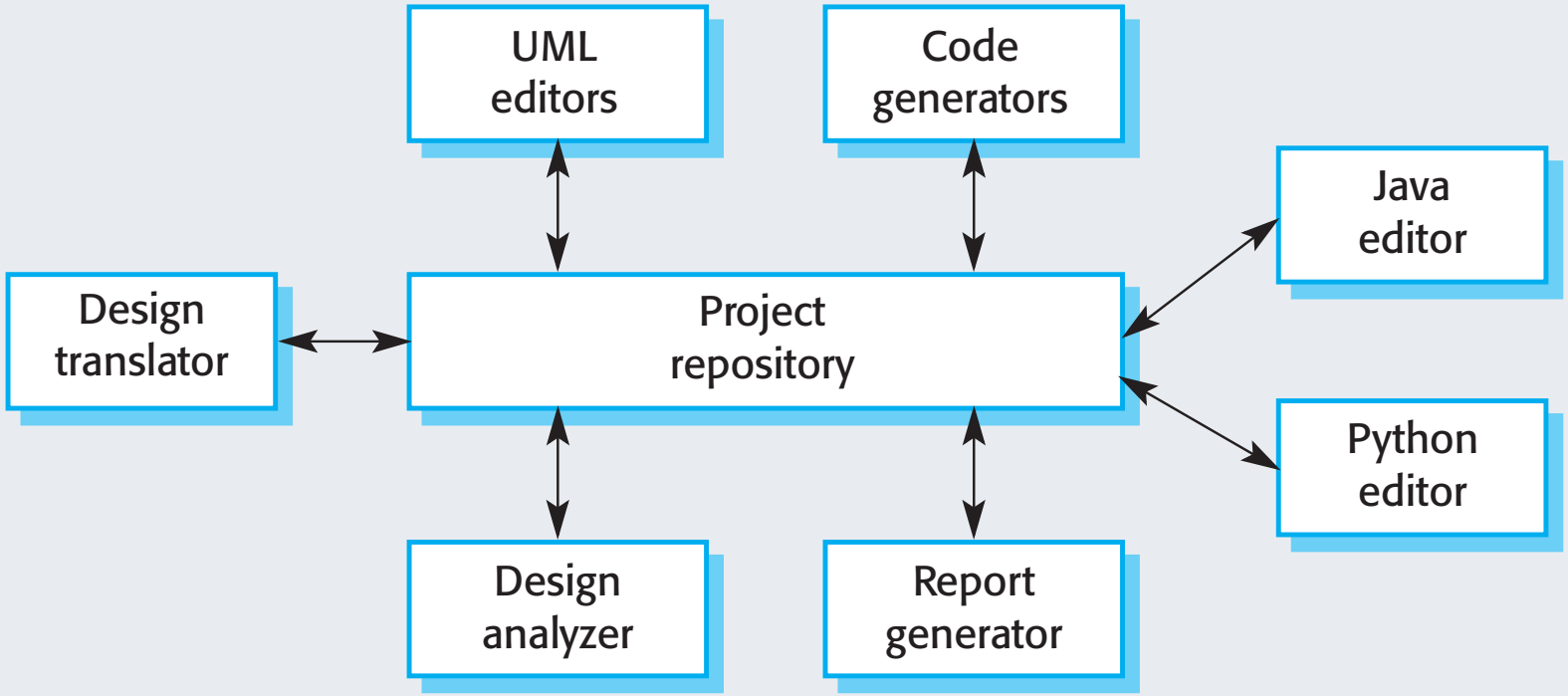
Repository architecture

- **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	Figure 6.9 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



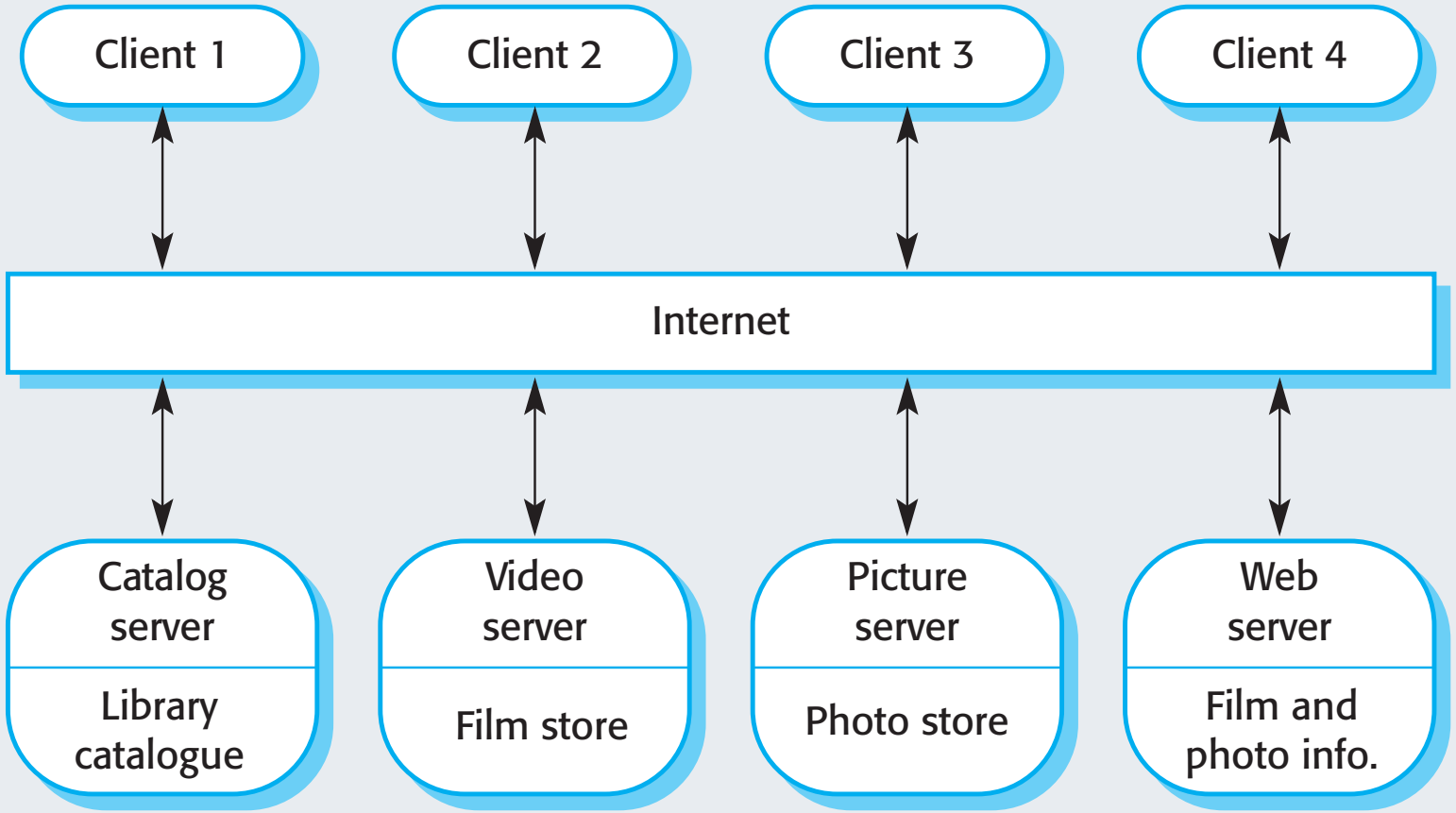
Client-server architecture

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

A client–server architecture for a film library



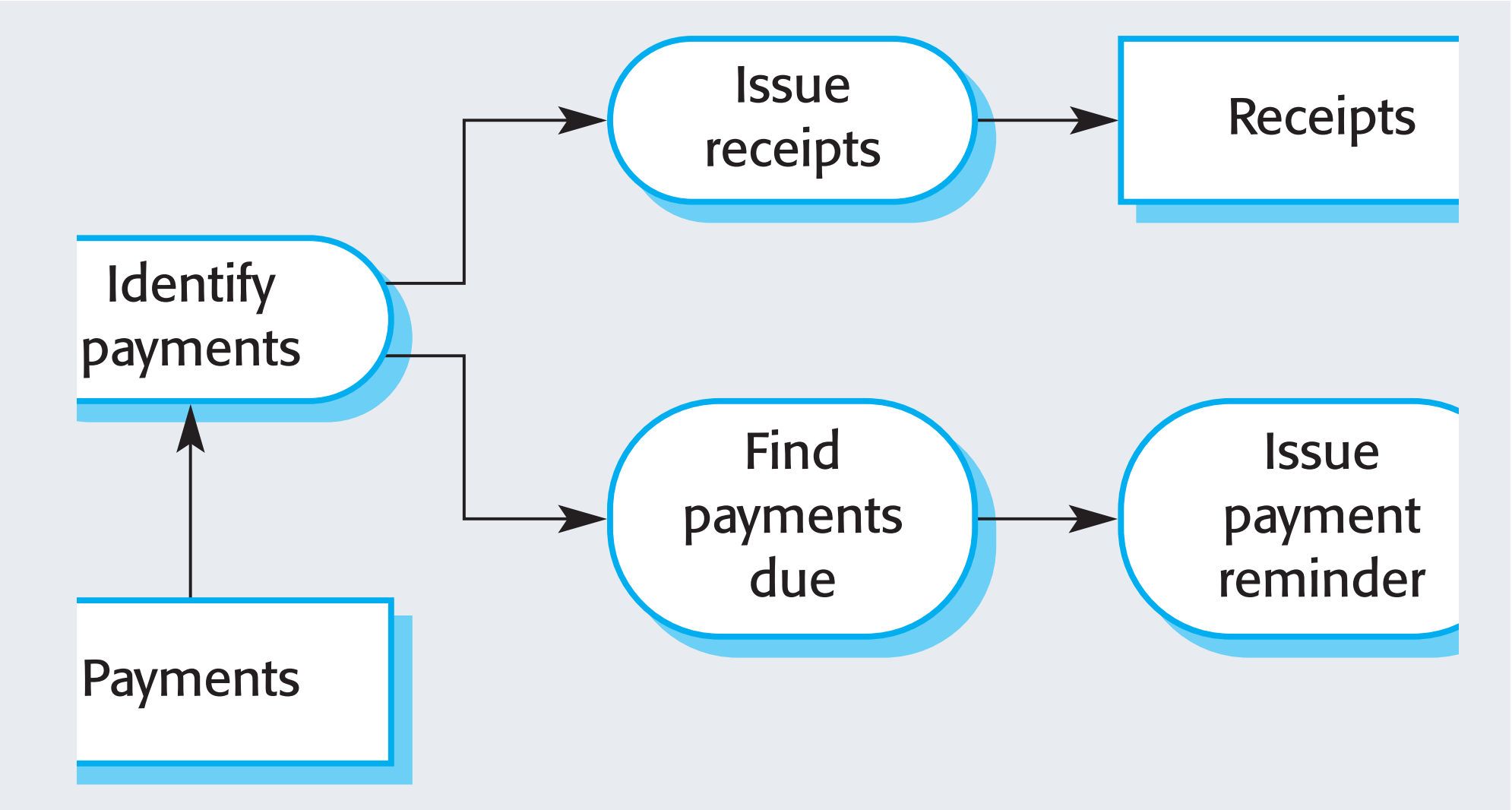
Pipe and filter architecture

- Functional transformations process their inputs to produce outputs.
- May be referred to as a pipe and filter model (as in UNIX shell).
- Variants of this approach 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.

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 6.13 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 used in a payments system



Architectural Design

Part 3

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.

Application types and examples

- **Data processing applications**
  - Data driven applications that process data in batches without explicit user intervention during the processing.
    - Billing systems;
    - Payroll systems.
- **Transaction processing applications**
  - Data-centered applications that process user requests and update information in a system database.
    - E-commerce systems;
    - Reservation systems.

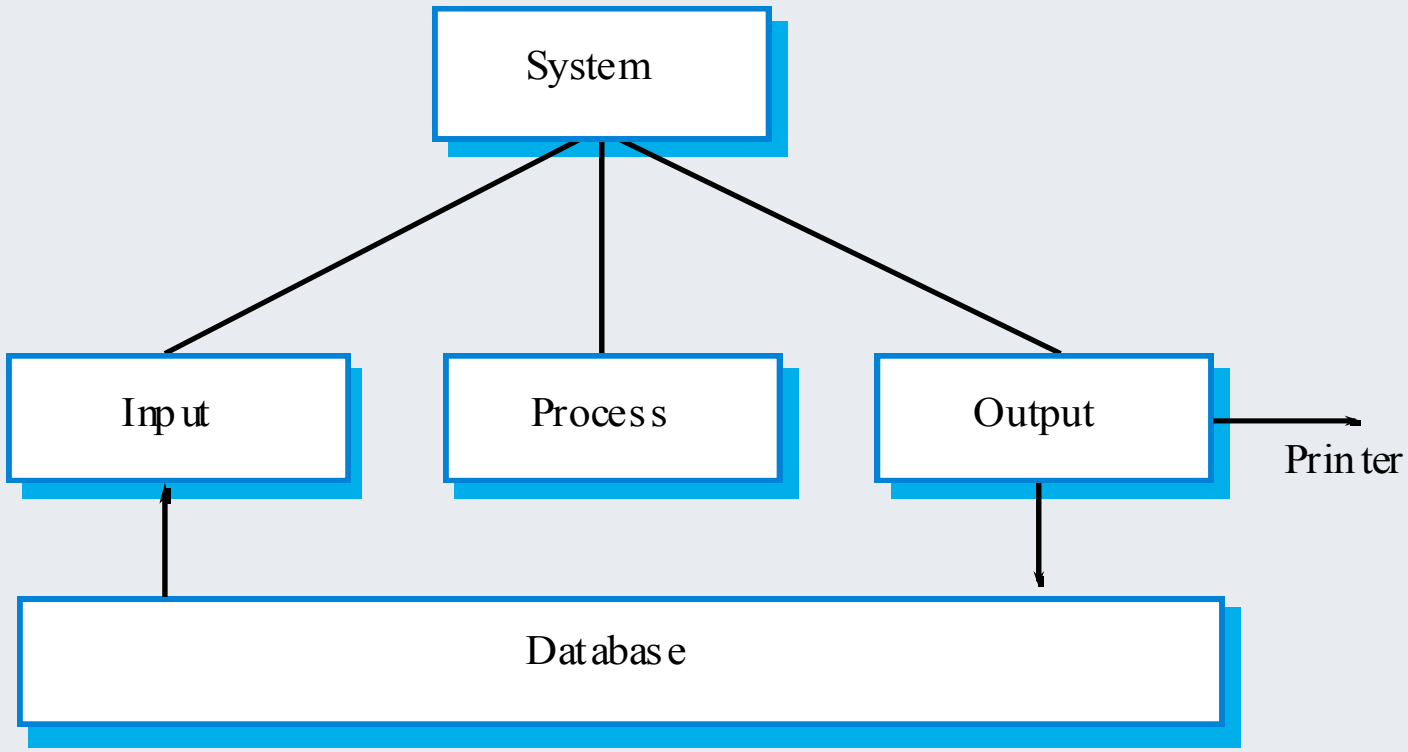
Application types and examples

- **Event processing systems**
  - Applications where system actions depend on interpreting events from the system's environment.
    - Word processors;
    - Real-time systems
- **Language processing systems**
  - Applications where the users' intentions are specified in a formal language that is processed and interpreted by the system.
    - Compilers;
    - Command interpreters.

1. Data processing systems

- Systems that are data-centered where the databases used are usually orders of magnitude larger than the software itself.
- Data is input and output in batches
  - Input: A set of customer numbers and associated readings of an electricity meter;
  - Output: A corresponding set of bills, one for each customer number.
- Data processing systems usually have an input-process-output structure.

Input-process-output model



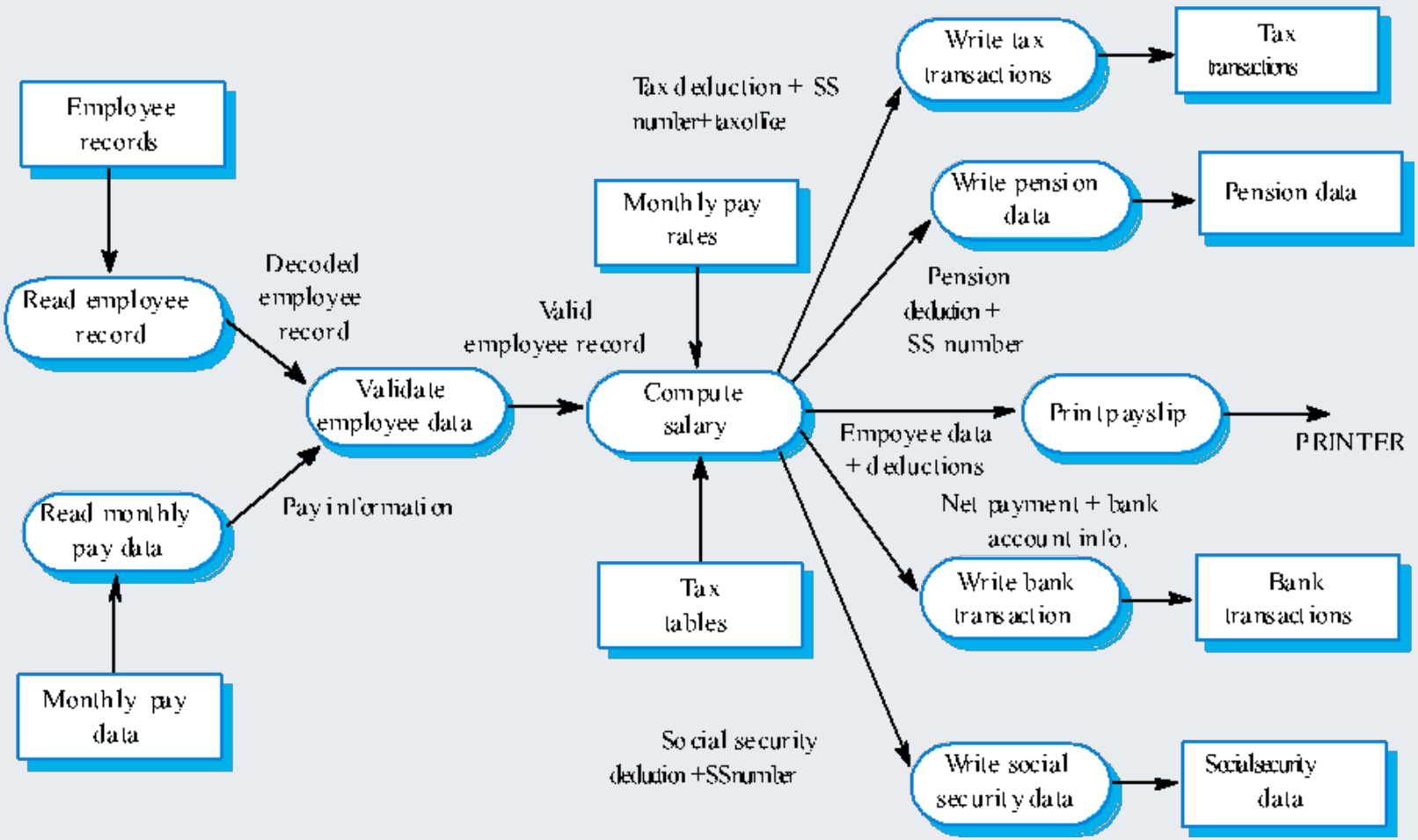
Input-process-output

- The input component reads data from a file or database, checks its validity and queues the valid data for processing.
- The process component takes a transaction from the queue (input), performs computations and creates a new record with the results of the computation.
- The output component reads these records, formats them accordingly and writes them to the database or sends them to a printer.

Data-flow diagrams

- Show how data is processed as it moves through a system.
- Transformations are represented as round-edged rectangles, data-flows as arrows between them and files/data stores as rectangles.

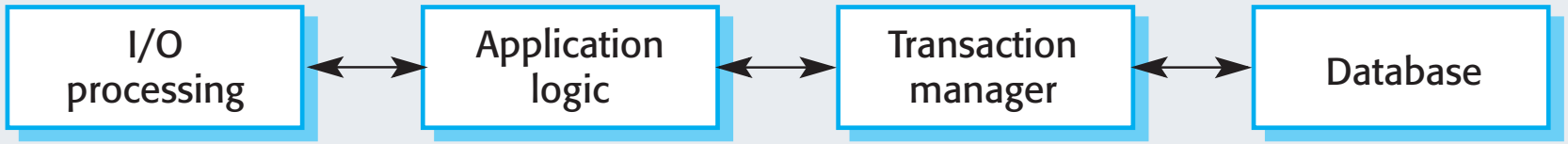
Salary payment DFD



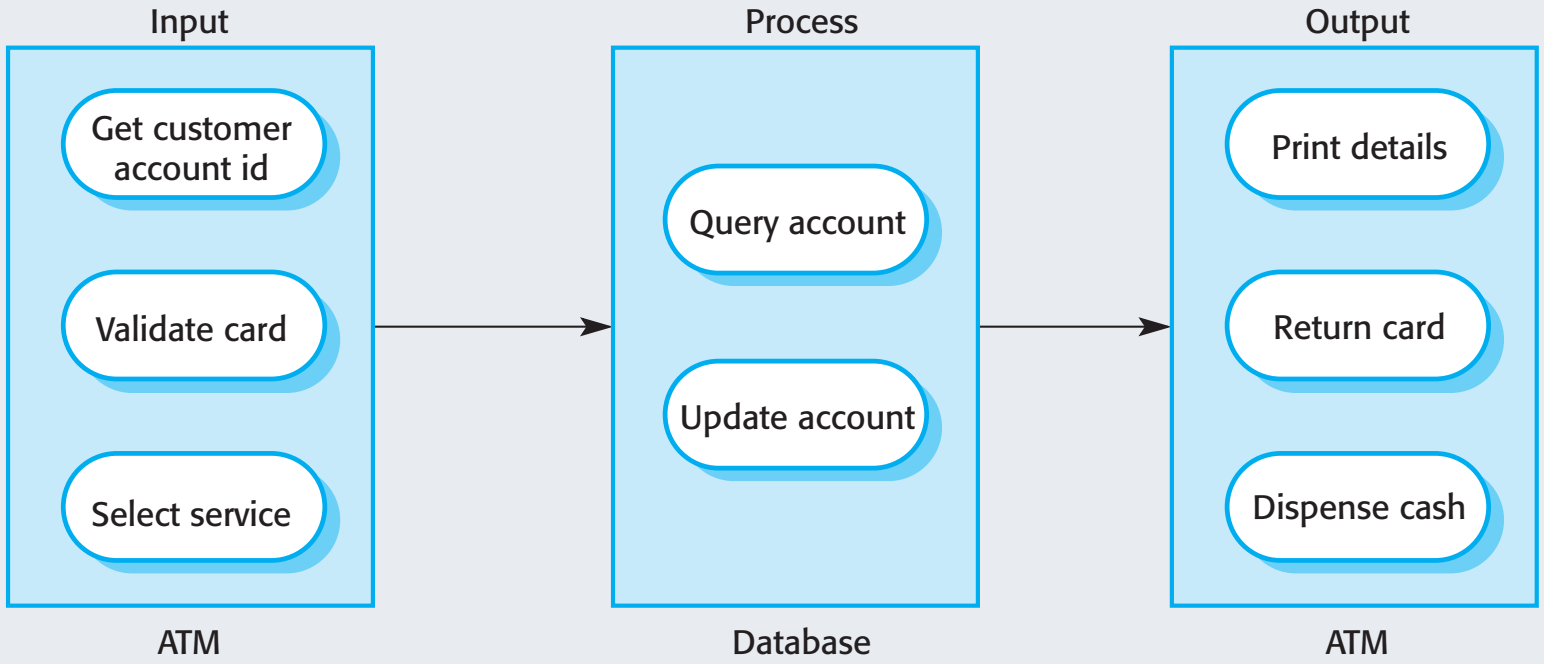
2. Transaction processing systems

- 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;
  - Ex) find the times of flights from London to Paris.
  - Ex) withdraw money using a ATM
- Users make asynchronous requests for service which are then processed by a transaction manager.

The structure of transaction processing applications



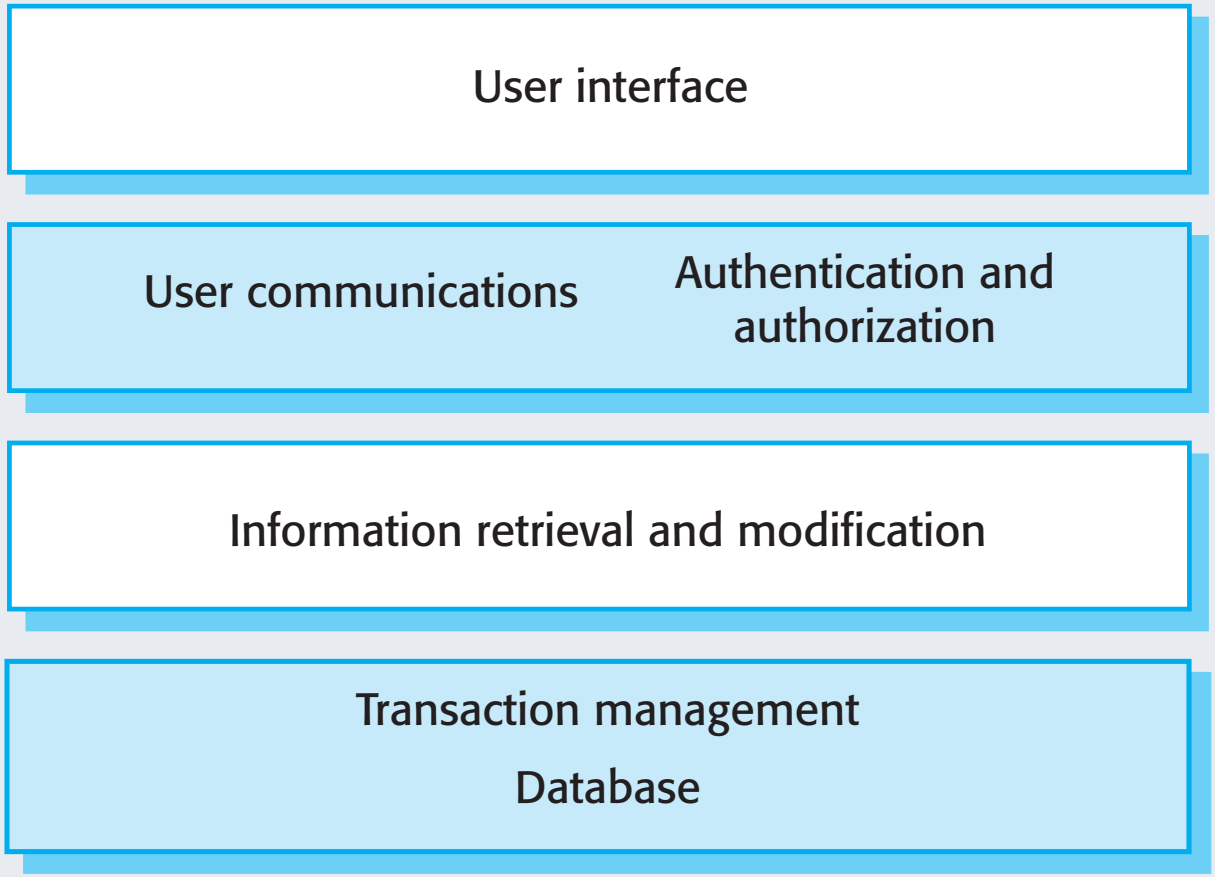
The software architecture of an ATM system



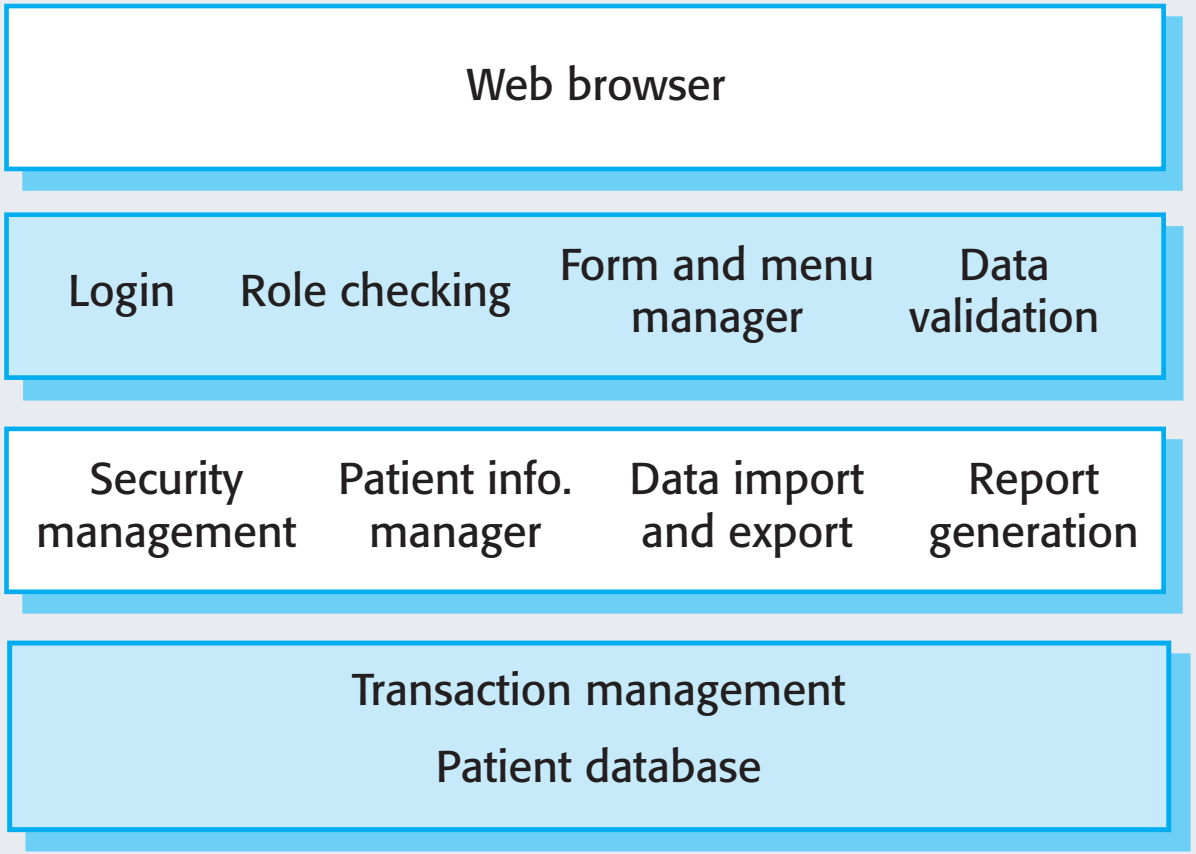
2.1 Information systems architecture

- **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



The architecture of the Mentcare system



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.
- For 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.

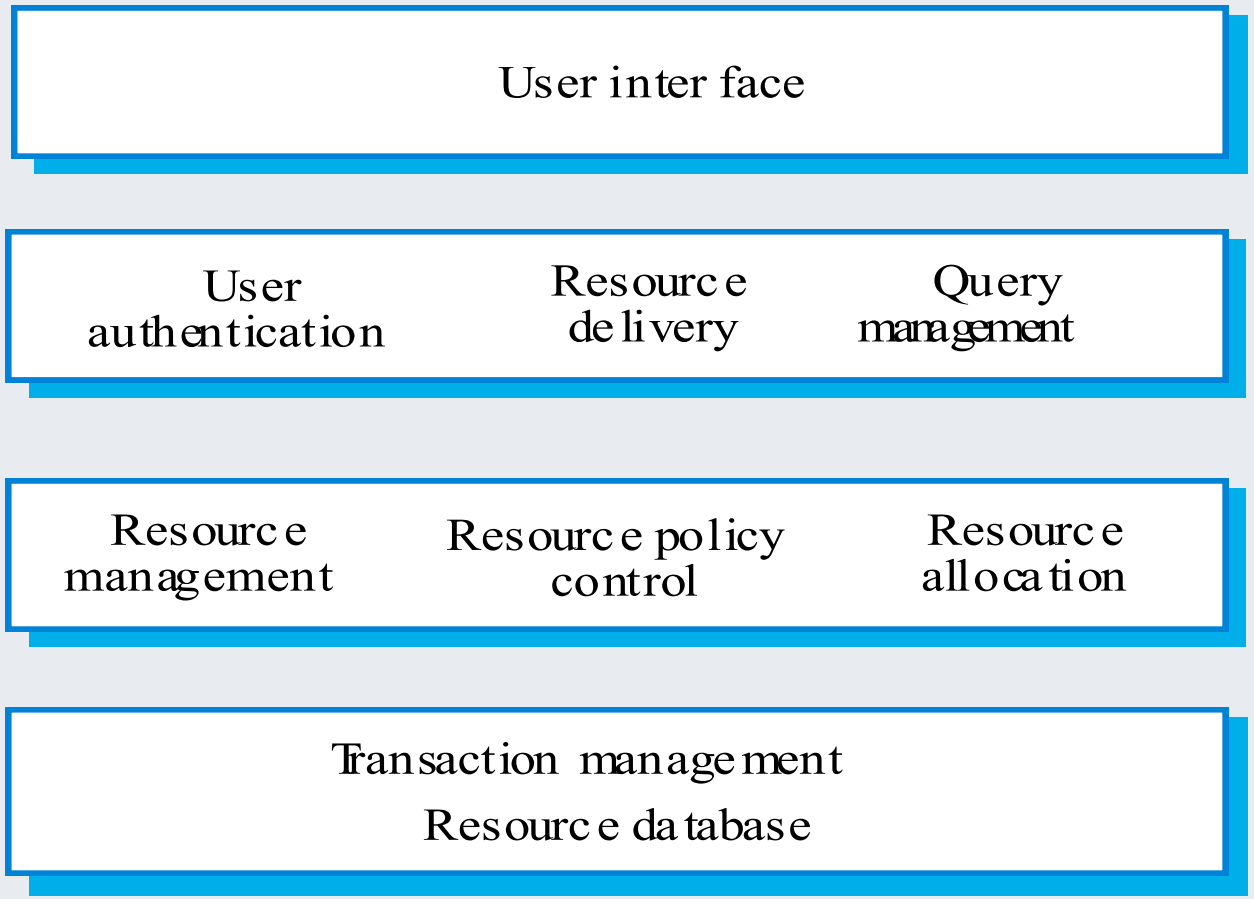
2.2 Resource allocation systems

- Systems that manage a fixed amount of some resource (football game tickets, books in a bookshop, etc.) and allocate this to users.
- Examples of resource allocation systems:
  - Timetabling systems where the resource being allocated is a time period;
  - Library systems where the resource being managed is books and other items for loan;
  - Air traffic control systems where the resource being managed is the airspace.

Resource allocation architecture

- Resource allocation systems are also layered systems that include:
  - A resource database;
  - A rule set describing how resources are allocated;
  - A resource manager;
  - A resource allocator;
  - User authentication;
  - Query management;
  - Resource delivery component;
  - User interface.

Layered resource allocation

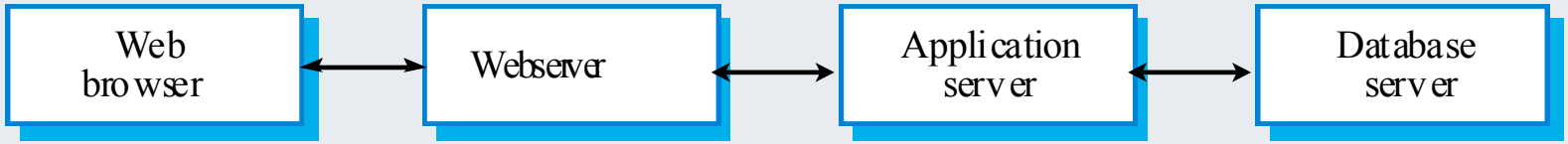


Layered system implementation

- Each layer can be implemented as a large scale component running on a separate server. This is the most commonly used architectural model for web-based systems.
- On a single machine, the middle layers are implemented as a separate program that communicates with the database through its API.
- Fine-grain components within layers can be implemented as web services.

E-commerce system architecture

- E-commerce systems are Internet-based resource management systems that accept electronic orders for goods or services.
- They are usually organized using a multi-tier architecture with application layers associated with each tier.

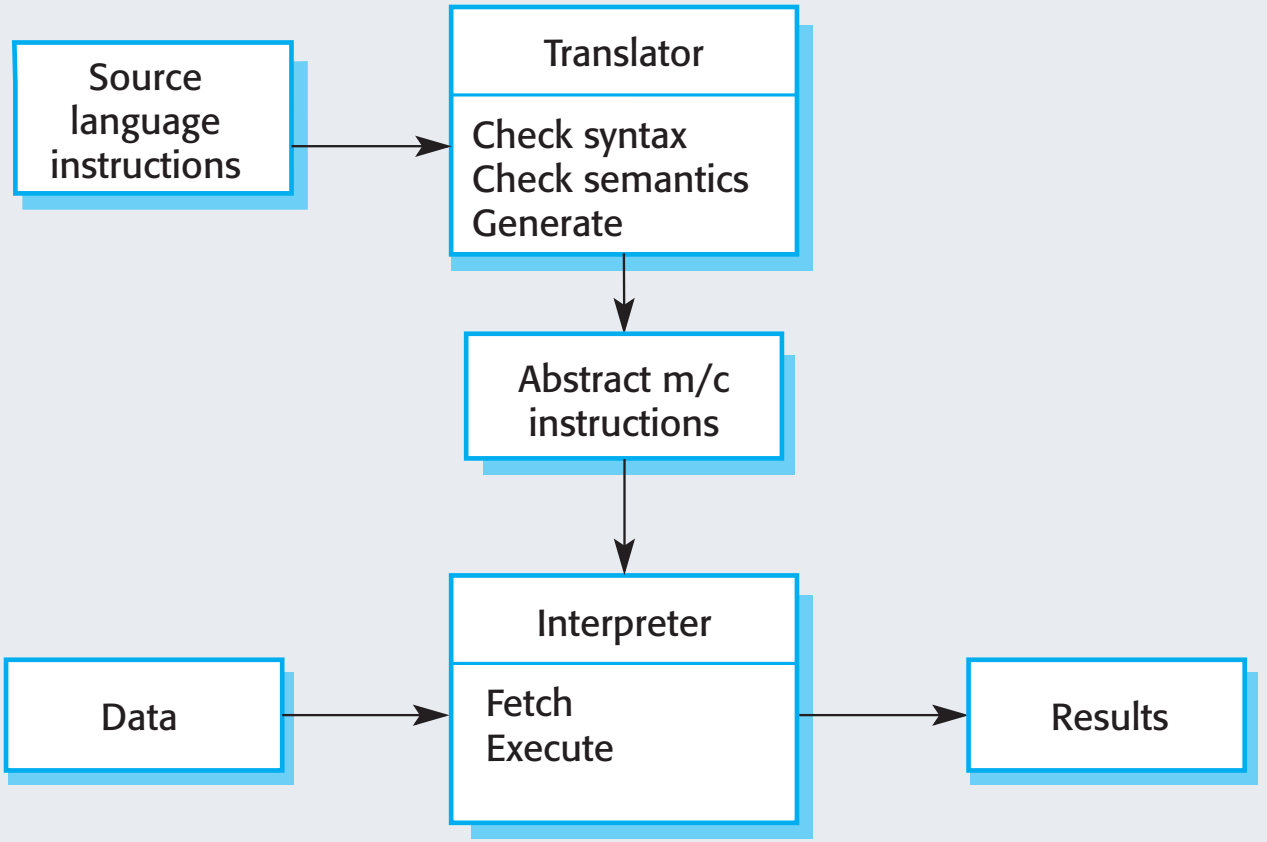


(Multi-tier Internet Transaction Processing System)

3. 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



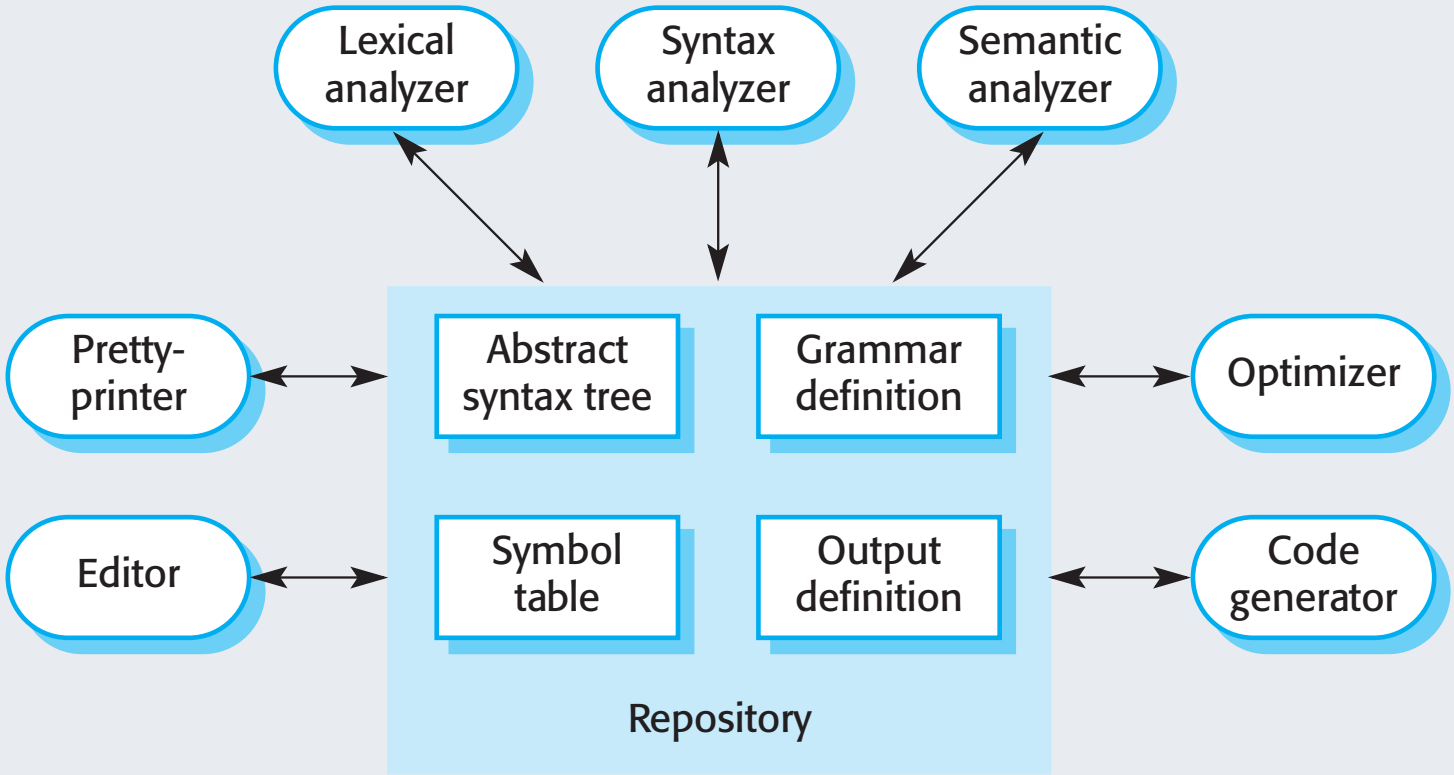
Compiler components

- A lexical analyzer, which takes input language tokens and converts them to an internal form.
- A symbol table, which holds information about the names of entities (variables, class names, object names, etc.) used in the text that is being translated.
- A syntax analyzer, which checks the syntax of the language being translated.
- A syntax tree, which is an internal structure representing the program being compiled.

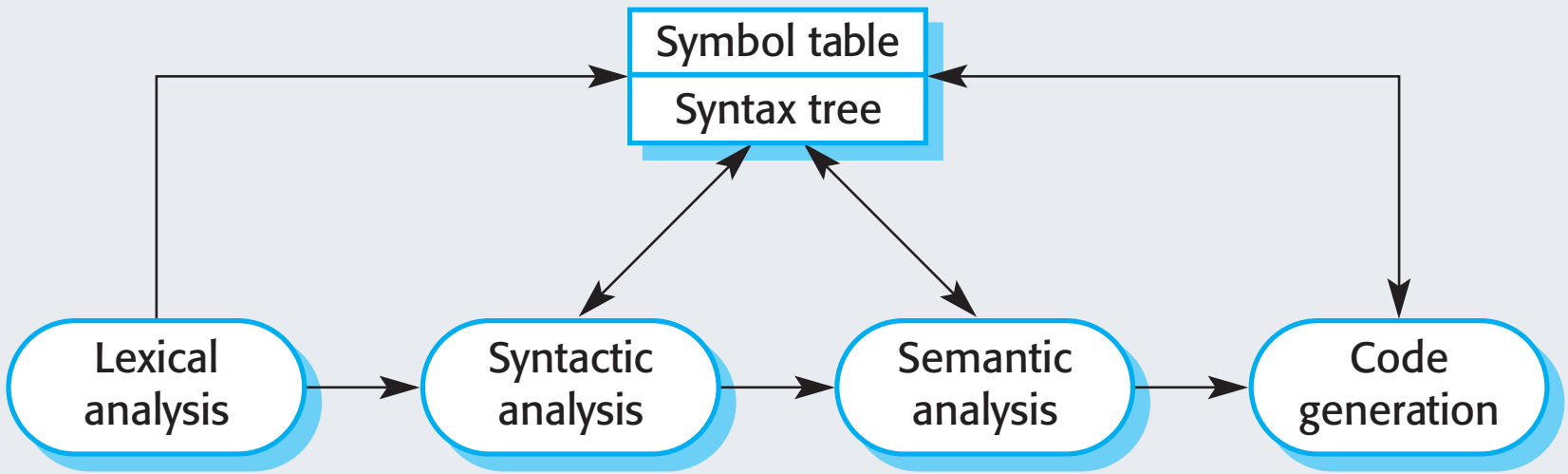
Compiler components

- A semantic analyzer that uses information from the syntax tree and the symbol table to check the semantic correctness of the input language text.
- A code generator that 'walks' the syntax tree and generates abstract machine code.

A repository architecture for a language processing system



A pipe and filter compiler architecture



Key points

- A software architecture is a description of how a software system is organized.
- Architectural design decisions include decisions on the type of application, the distribution of the system, the architectural styles to be used.
- 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.

Key points

- Models of application systems architectures help us understand and compare applications, validate application system designs and assess large-scale components for reuse.

Architectural design

Appendix

- Control styles
- Reference architecture

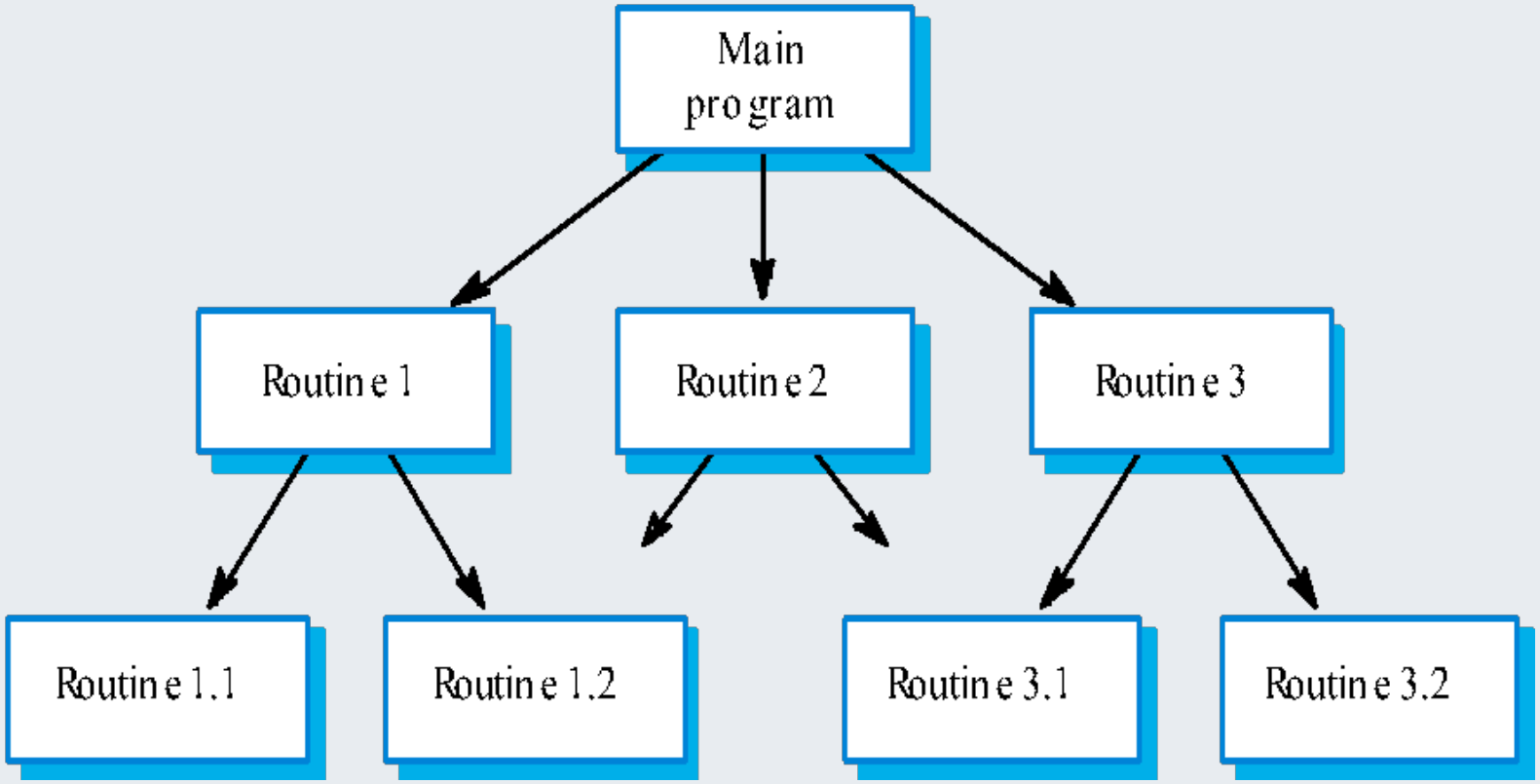
A-1. Control styles

- Are concerned with the control flow between sub-systems. Distinct from the system decomposition model.
- **Centralized control**
  - One sub-system has overall responsibility for control and starts and stops other sub-systems.
- **Event-based control**
  - Each sub-system can respond to externally generated events from other sub-systems or the system's environment.

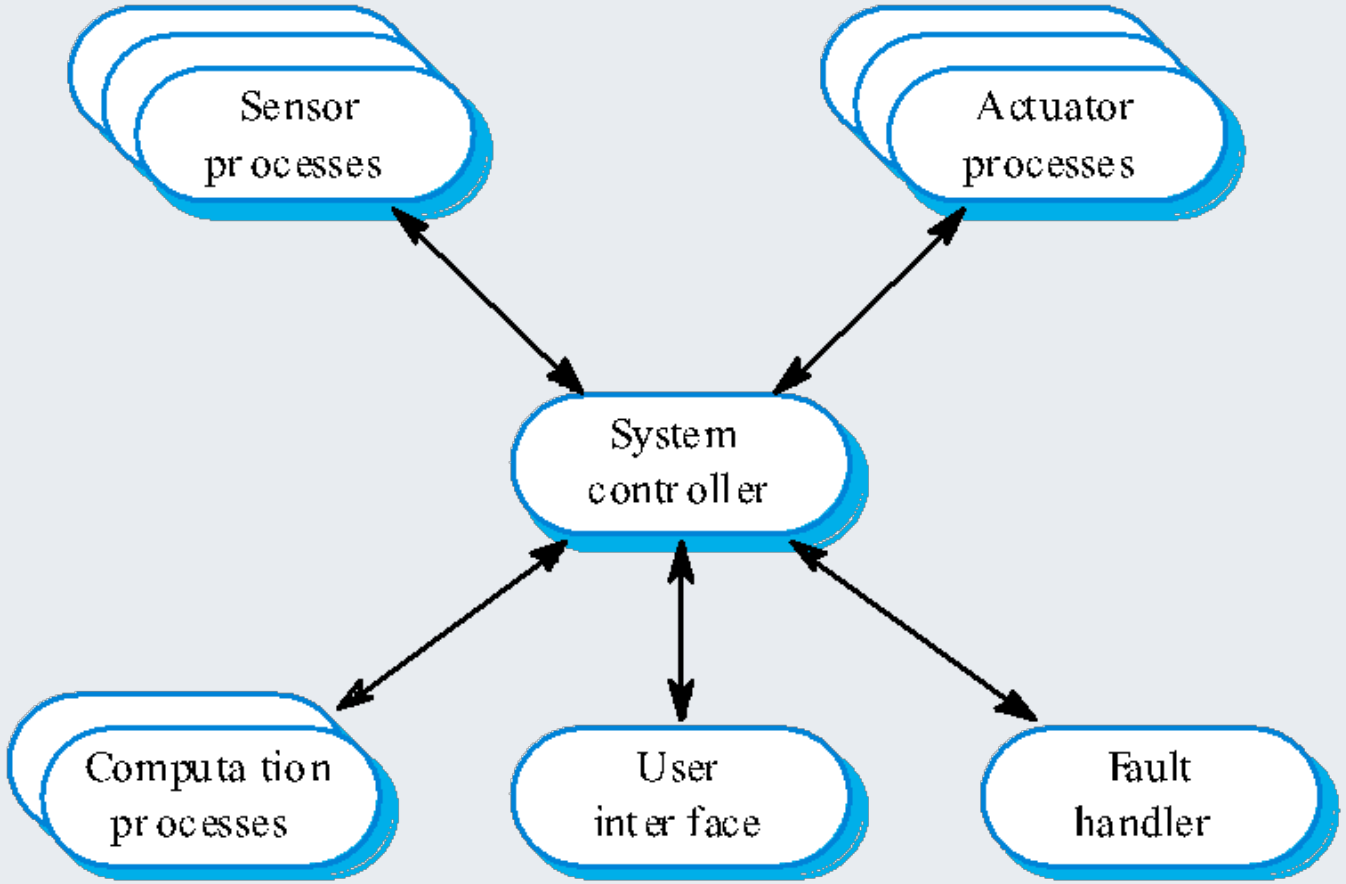
Centralized control

- **A control sub-system takes responsibility for managing the execution of other sub-systems.**
- **Call-return model**
  - Top-down subroutine model where control starts at the top of a subroutine hierarchy and moves downwards. Applicable to sequential systems.
- **Manager model**
  - Applicable to concurrent systems. One system component controls the stopping, starting and coordination of other system processes. Can be implemented in sequential systems as a case statement.

Call-return model



Real-time system control



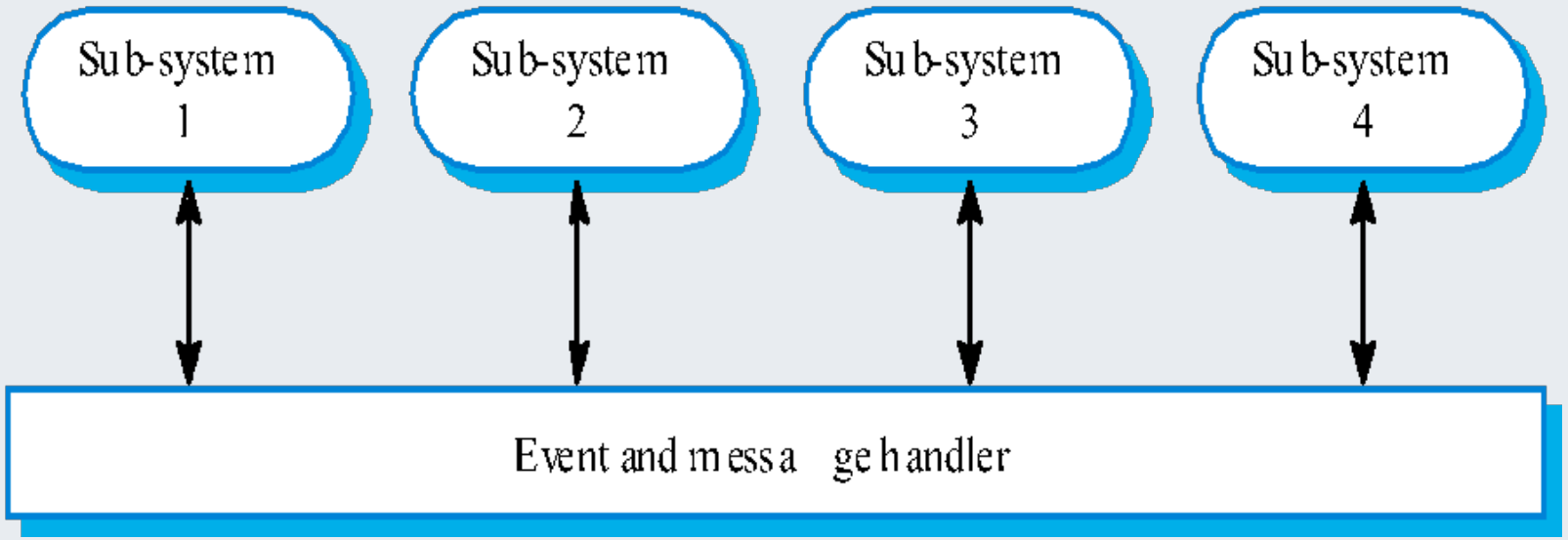
Event-driven systems

- Driven by externally generated events where the timing of the event is outside the control of the sub-systems which process the event.
- Two principal event-driven models(Garlan 1992)
  - Broadcast models. An event is broadcast to all sub-systems. Any sub-system which can handle the event may do so;
  - Interrupt-driven models. Used in real-time systems where interrupts are detected by an interrupt handler and passed to some other component for processing.
- Other event driven models include spreadsheets and production systems.

Broadcast model

- Effective in integrating sub-systems on different computers in a network.
- Sub-systems register an interest in specific events. When these occur, control is transferred to the sub-system which can handle the event.
- Control policy is not embedded in the event and message handler. Sub-systems decide on events of interest to them.
- However, sub-systems don't know if or when an event will be handled.

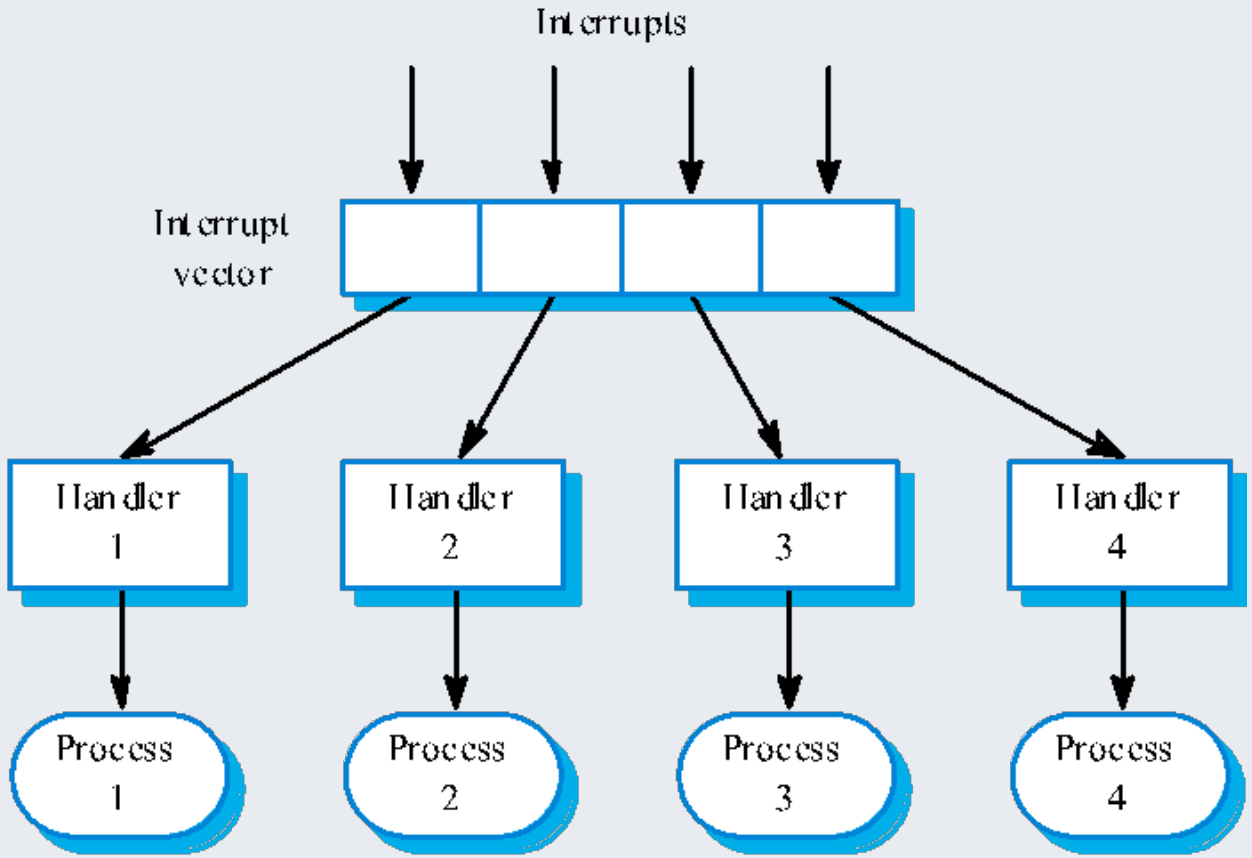
Selective broadcasting



Interrupt-driven systems

- Used in real-time systems where fast response to an event is essential.
- There are known **interrupt types** with a **handler** defined for each type.
- Each type is associated with a memory location and a hardware switch causes transfer to its handler.
- Allows fast response but complex to program and difficult to validate.

Interrupt-driven control



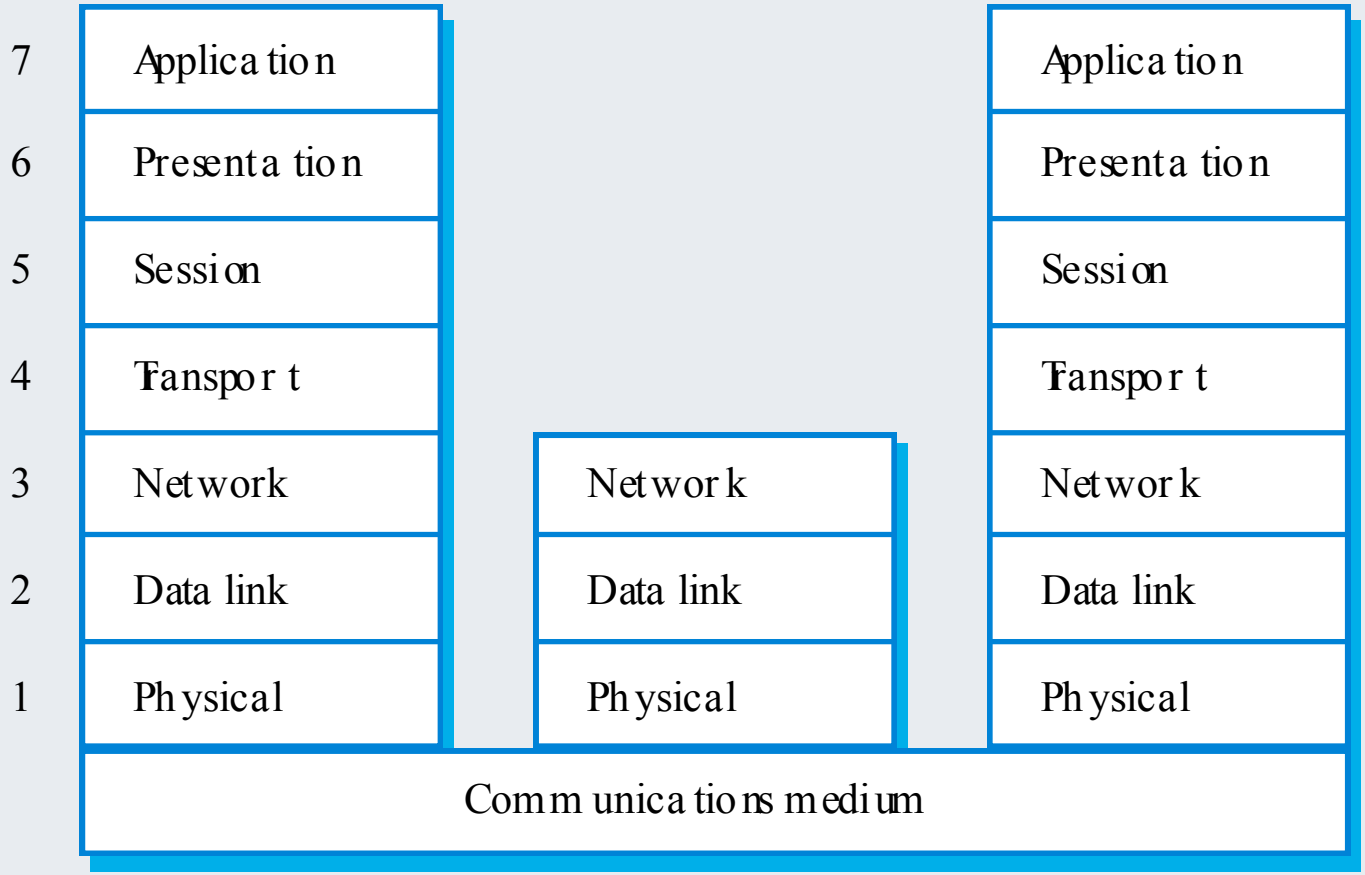
A-2 Reference architectures

- **Architectural models may be specific to some application domain.**
- **Two types of domain-specific model**
  - Generic models which are abstractions from a number of real systems and which encapsulate the principal characteristics of these systems.
  - Reference models which are more abstract, idealized model. Provide a means of information about that class of system and of comparing different architectures.
- **Generic models are usually bottom-up models; Reference models are top-down models.**

Reference architectures

- Reference models are derived from a study of the **application domain** rather than from **existing systems**.
- May be used as a basis for system implementation or to compare different systems. It acts as a **standard** against which systems can be evaluated.
- OSI model is a layered model for communication systems.

OSI reference model



Case reference model

- **Data repository services**
  - Storage and management of data items.
- **Data integration services**
  - Managing groups of entities.
- **Task management services**
  - Definition and enaction of process models.
- **Messaging services**
  - Tool-tool and tool-environment communication.
- **User interface services**
  - User interface development.

The ECMA reference model