COMP41670 Software Engineering

15. Architectural Design

Dr Avishek Nag

UCD School of Computer Science. **Scoil na Ríomheolaíochta UCD.**

Table of Contents

What is Architectural Design?

Architectural Design

- … is the process of determining how a software system should be organised at the high level.
- It identifies the main structural components in the system and the relationships between them.
- The output of the architectural design process is an architectural model that describes how the system is organised as a set of communicating components.

Architectural Design

- **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.
- **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.

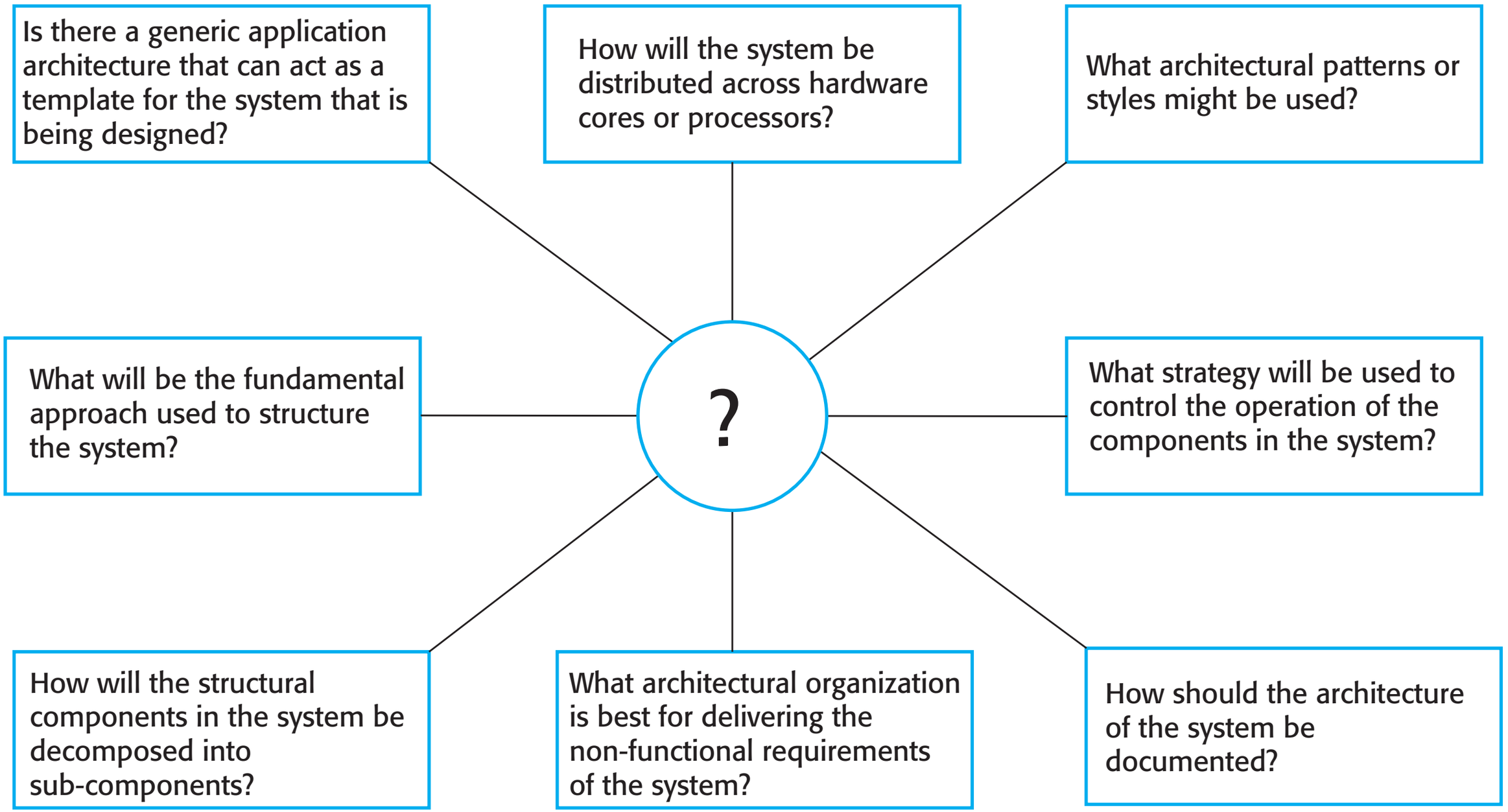
Really, this is design

Architectural Design

- The architectural model:
  - Facilitates understanding and discussion between team members
  - Documents the architecture
  - Is the basis of design and implementation

Architectural Design Decisions

Architectural Design Decisions



Architectural Design Concerns

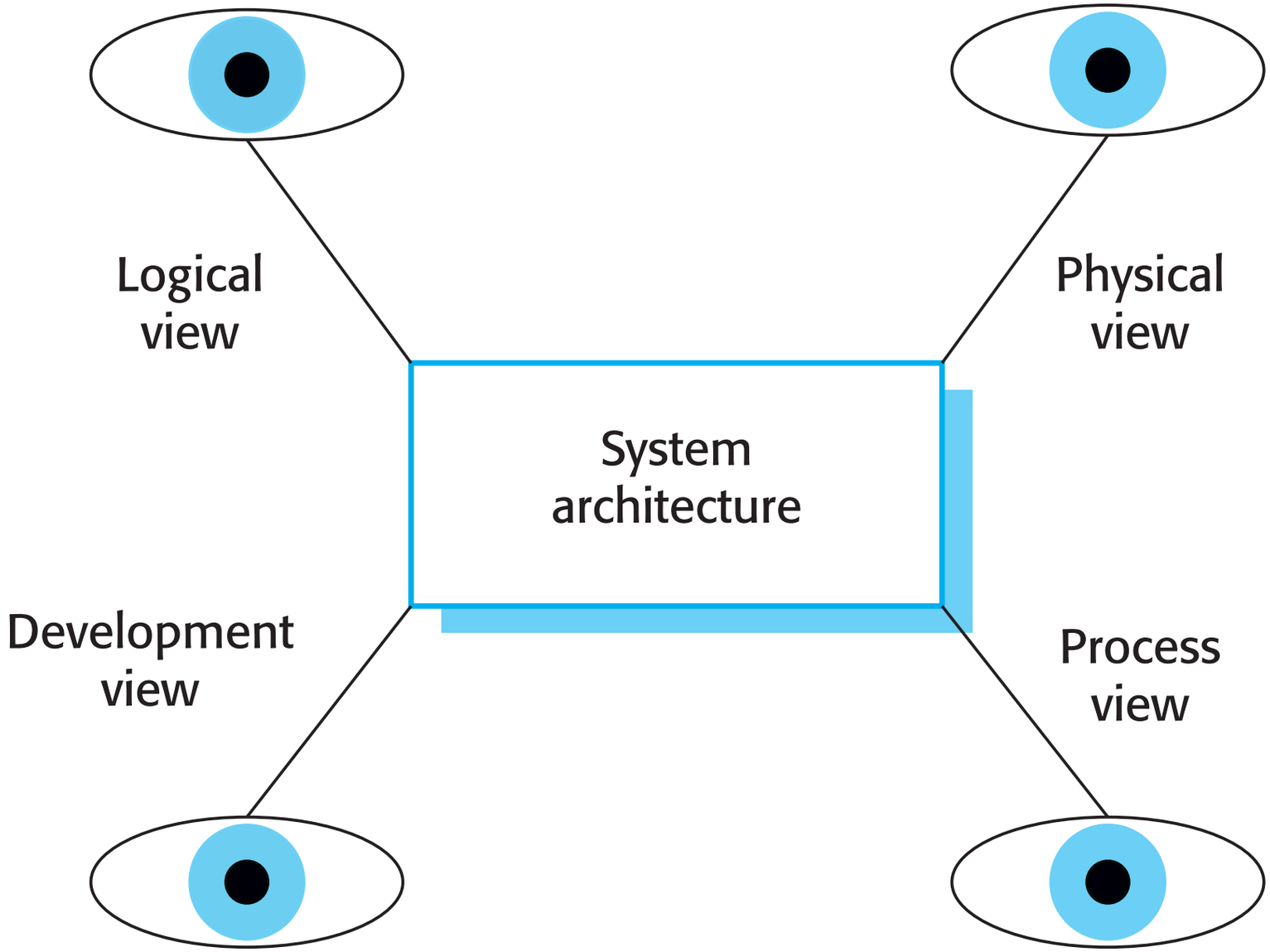
- Performance
- Security
- Safety-critical features
- Availability / Fault tolerance
- Maintainability

Architecture Re-use

- 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’.

Architectural Views

Architectural Views



Architectural Views

- **Logical view**: shows the key abstractions in the system as objects or object classes.
  - e.g., class diagrams, state diagrams
- **Process view**: shows how, at run-time, the system is composed of interacting processes.
  - e.g., sequence diagram, activity diagram
- **Development view**: shows how the software is decomposed for development.
  - e.g., package diagram
- **Physical view**: shows the system hardware and how software components are distributed across the processors in the system.
  - e.g., deployment diagram

https://medium.com/javarevisited/4-1-architectural-view-model-in-software-ec407bf27258#:~:text=The%20model%20has%20four%20views,The%204%2B1%20Architectural%20View.

Architectural Patterns

Architectural Patterns

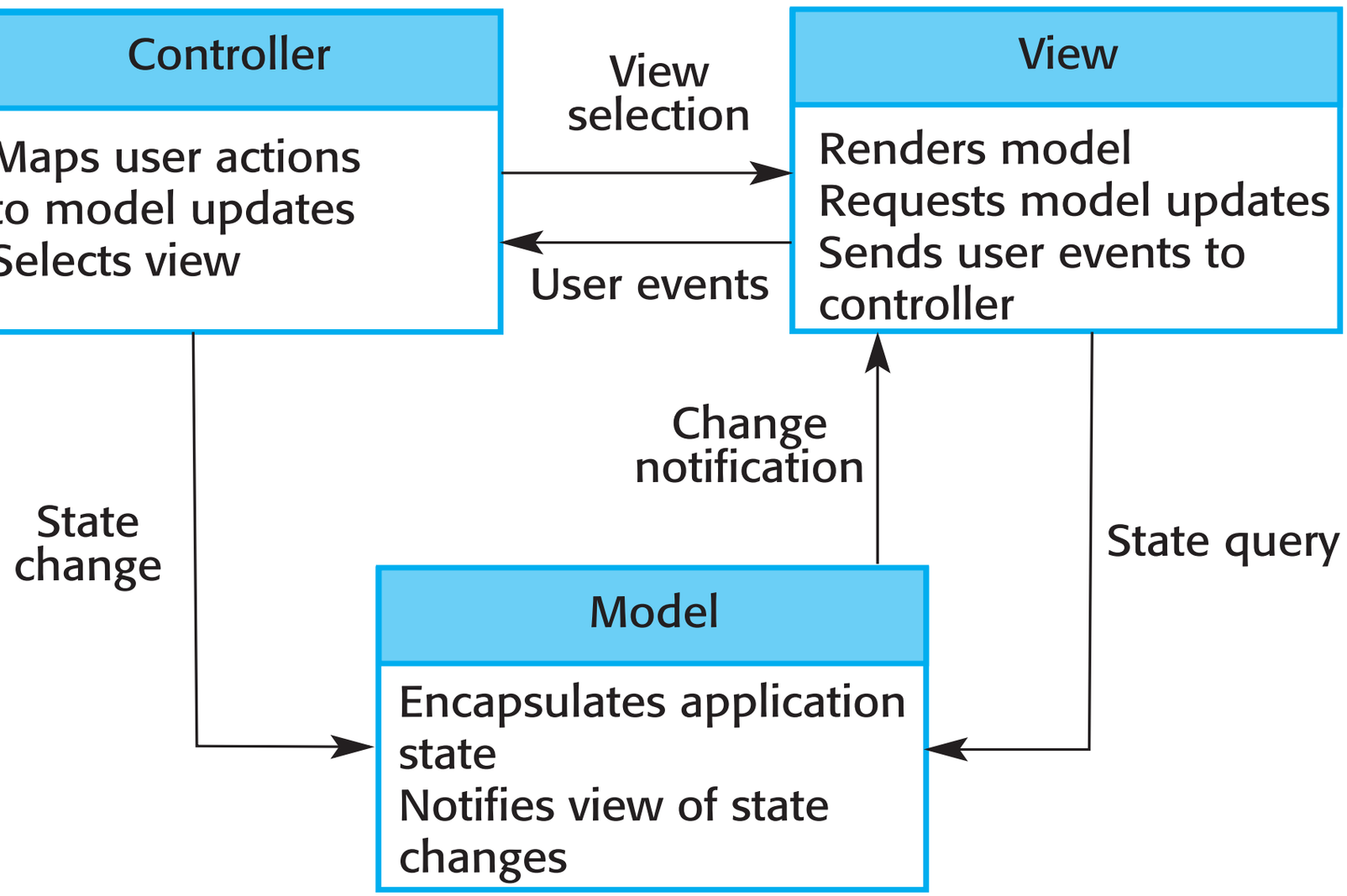
- A **pattern** is a high quality transferrable solution to a commonly occurring design problem.
- A pattern can be documented and re-used for similar design problems.
- Popular architectural patterns:
  - Model-View-Controller (MVC)
  - Layered architecture
  - Repository architecture
  - Client-server architecture
  - Pipe and filter architecture

[Buschmann, at al., *Pattern-Oriented Software Architecture, Volume 1-5*, Wiley, 1996.]

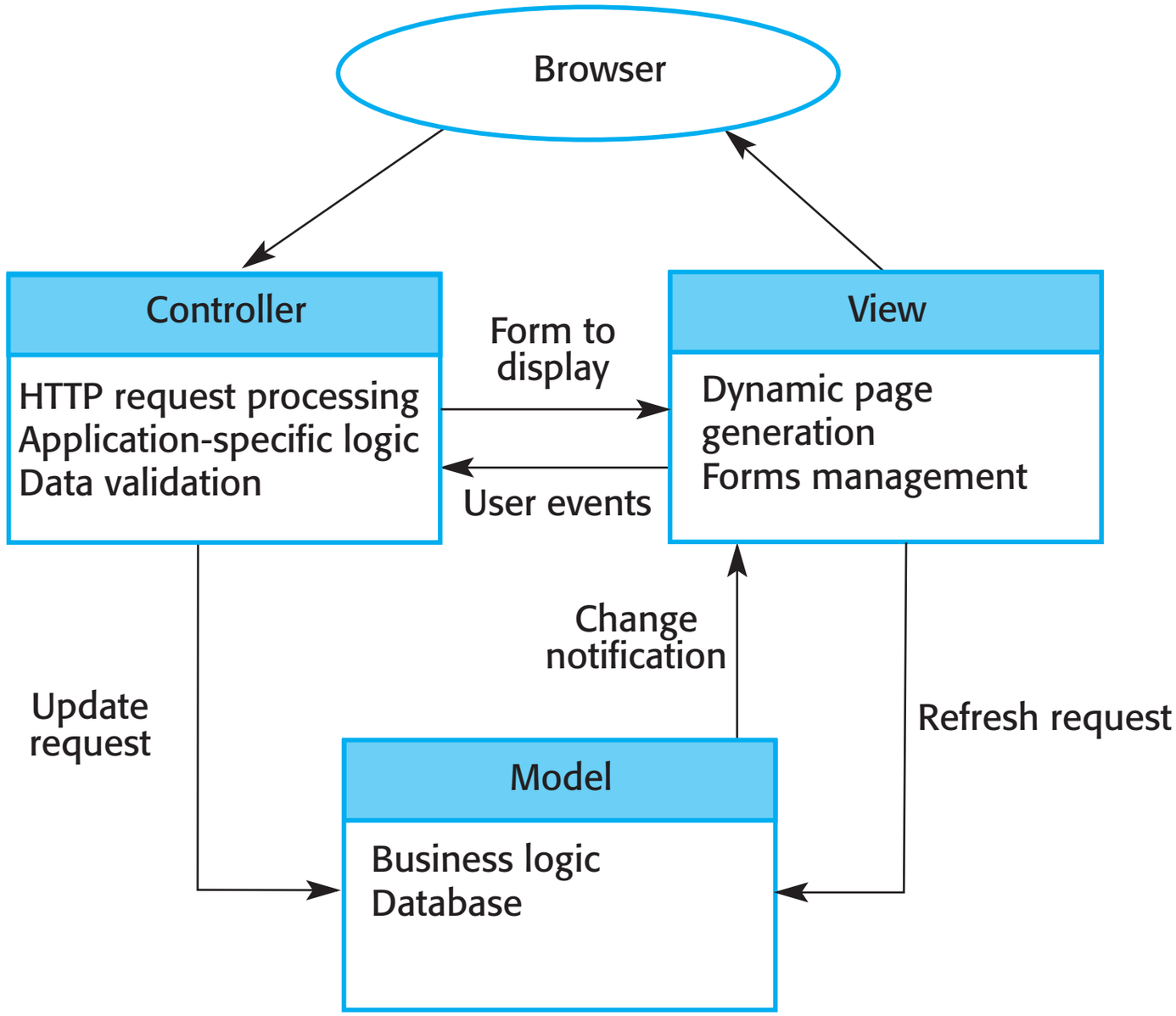
Model-View-Controller

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	Next slide shows the architecture of a web-based application system organised 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.

Model-View-Controller



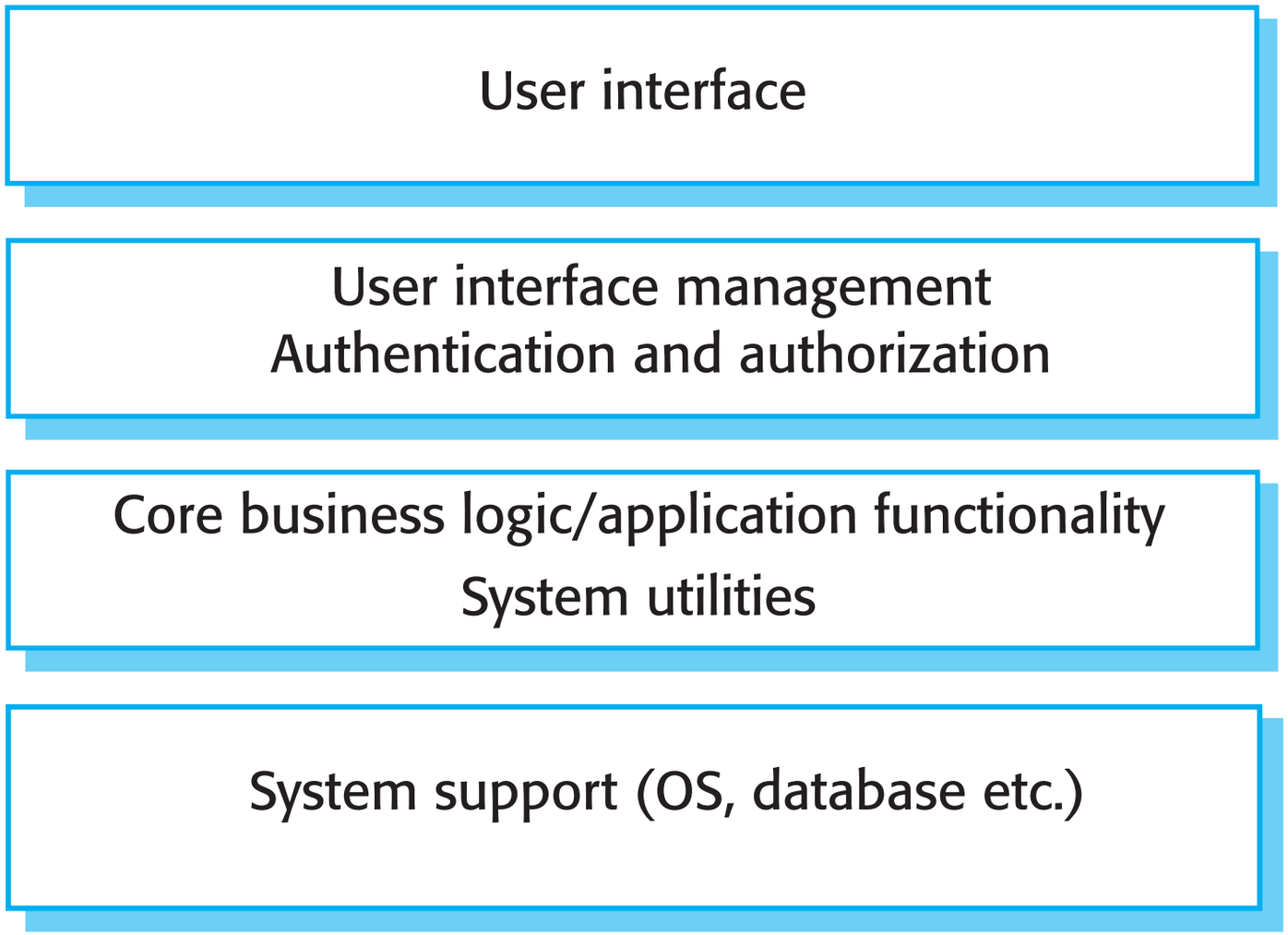
Model-View-Controller: Application Example (web site)



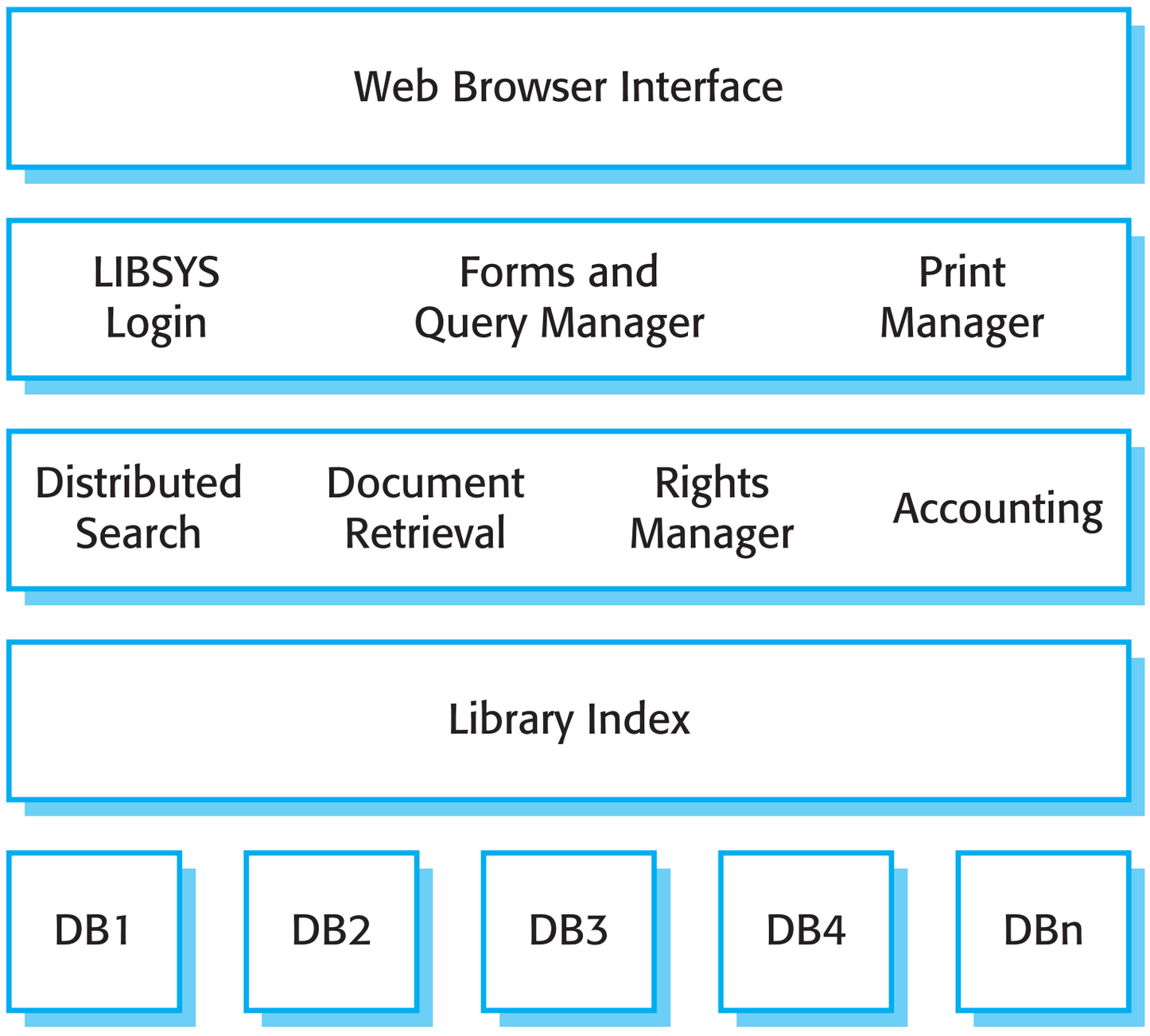
Layered Architecture

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.
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.

Layered Architecture



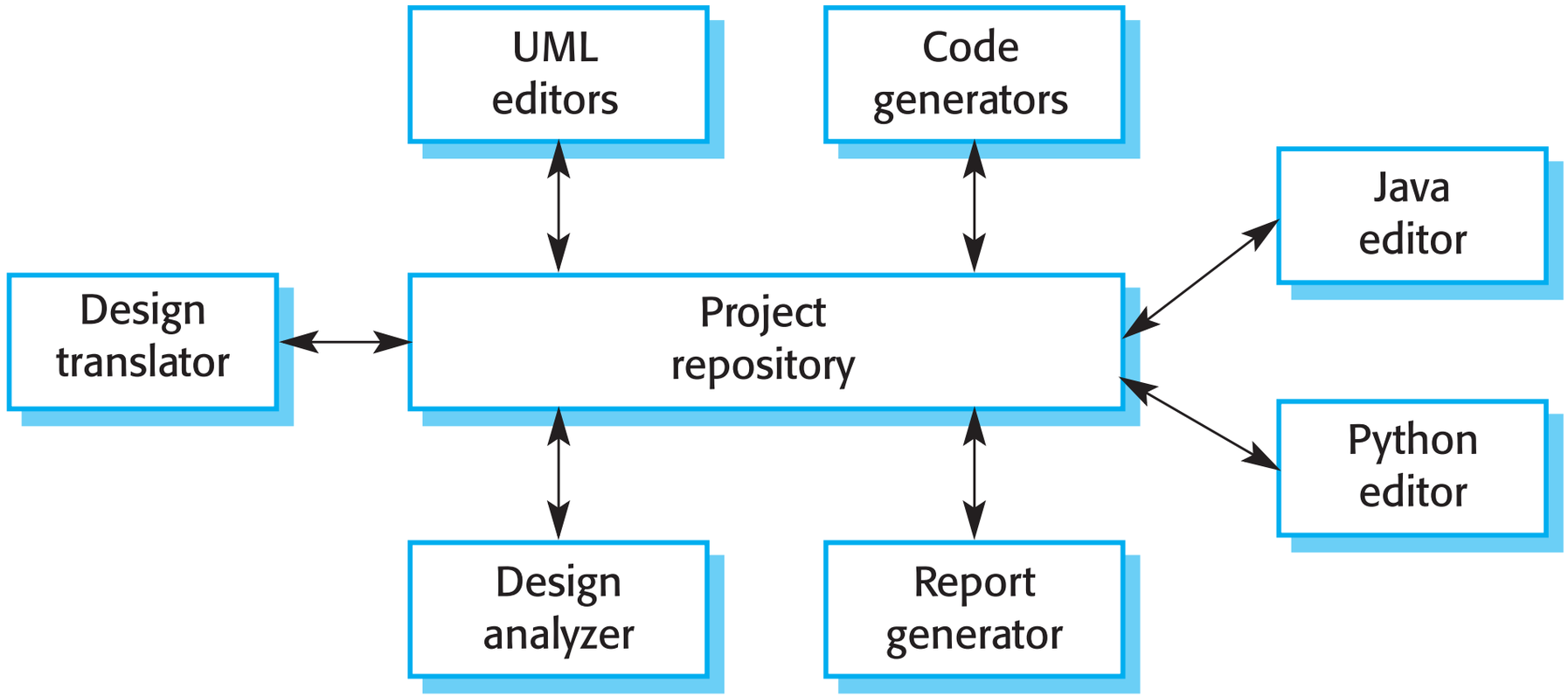
Layered Architecture: Application Example (LIBSYS)



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	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.

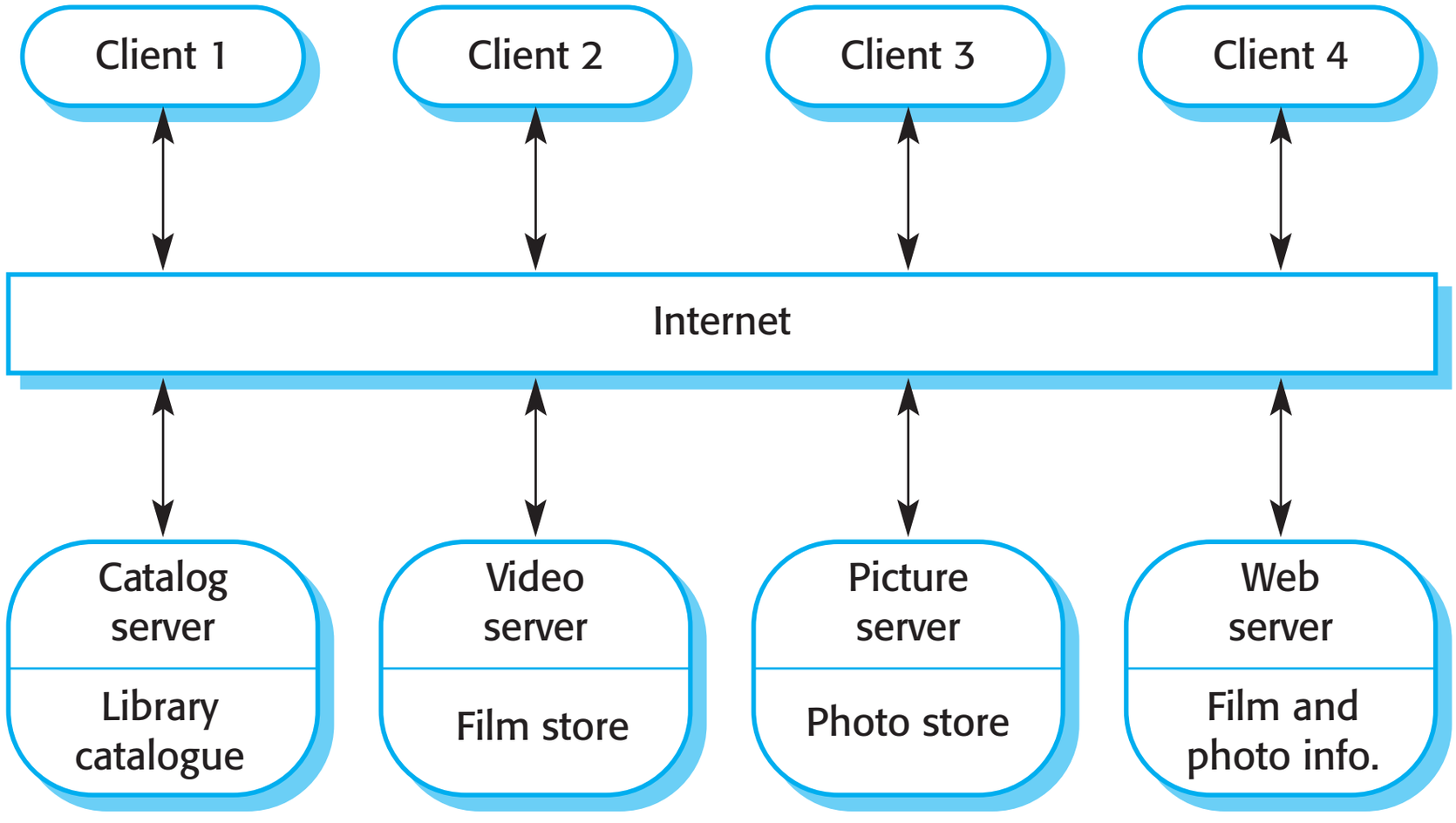
Repository Pattern: Application Example (IDE)



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	A film and video/DVD library organised 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.

Client-Server Pattern: Application Example (film library)



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	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.

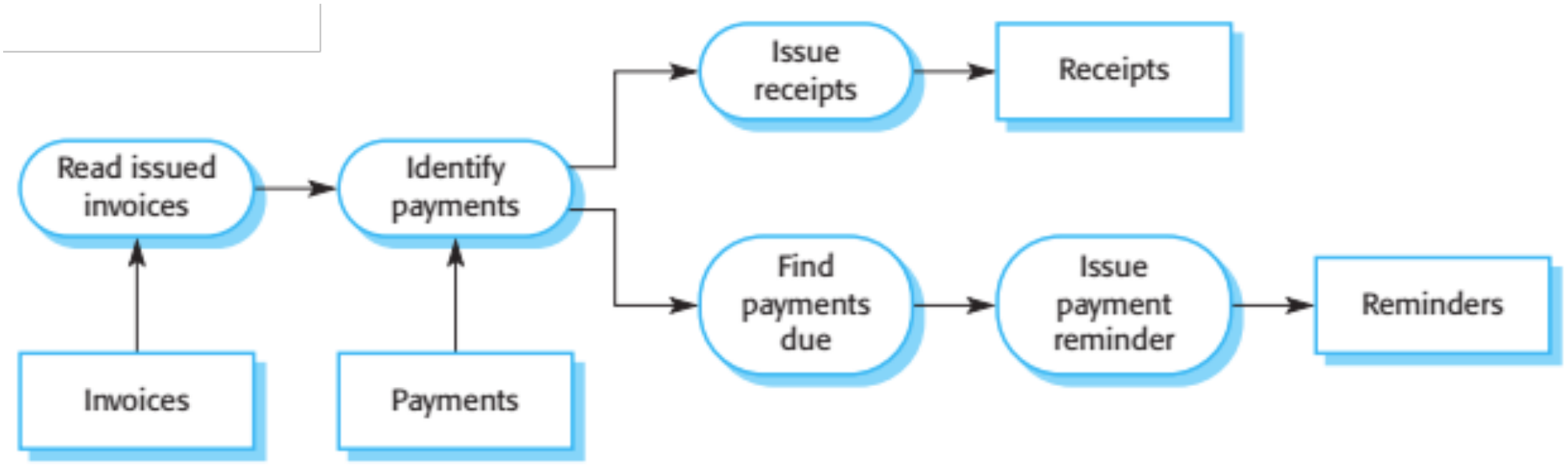
Pipe and Filter Pattern

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	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.

Pipe and Filter Pattern

- Typically used for transaction processing systems (aka transaction-based information system).
- Transaction processing systems are typically interactive systems to which users make asynchronous (anytime) requests for service.
- A transaction manager control the processing of the request which usually involves access to the database.
- When the transaction is complete, the user is notified.
- E.g. ATM account query

Pipe and Filter Pattern: Application Example (invoice batch processing)



Summary

- A **software architecture** is a description of how a software system is organised.
- 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 abstract architecture, explain when it may be used and describe its advantages and disadvantages.