# Architectural Design

# Topics covered


- Architectural design decisions
- Architectural views
- Architectural patterns

# 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.

# Architectural design

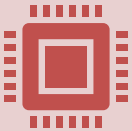

Once interactions between the system and its environment have been understood, you use this information for designing the system architecture

You identify the major components that make up the system and their interactions

You then may organize the components using an architectural pattern (e.g., a layered or client-server model)

# Agility and architecture


It is generally accepted that an early stage of agile processes is to design an overall systems architecture.

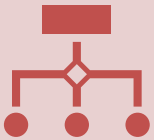


Refactoring the system architecture is usually expensive because it affects so many components in the system

# Build or buy

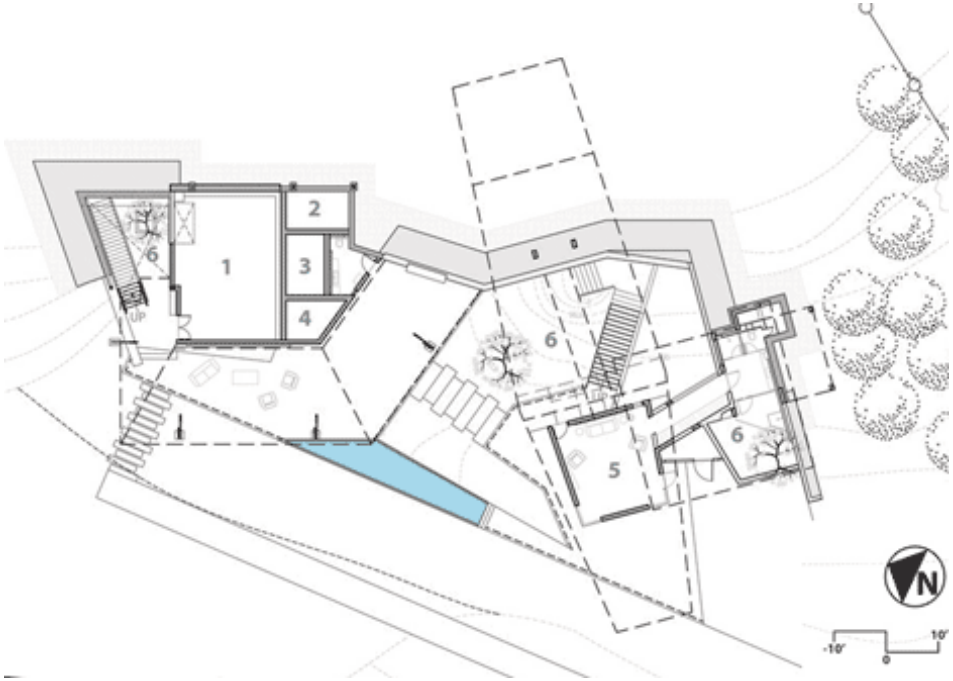

In a wide range of domains, it is possible to buy off-the-shelf systems (COTS) that can be adapted and tailored to the users' requirements.

For example, if you want to implement a medical records system, you can buy a package that is already used in hospitals. It can be cheaper and faster to use this approach rather than developing a system in a conventional programming language.



When you develop an application in this way, the design process becomes concerned with how to use the configuration features of that system to deliver the system requirements.

# Architectural abstraction




## Architecture in the small

- It 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

- It 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 NFRs is possible.

**Large-scale reuse**

The architecture may be reusable across a range of systems

Product-line architectures may be developed.

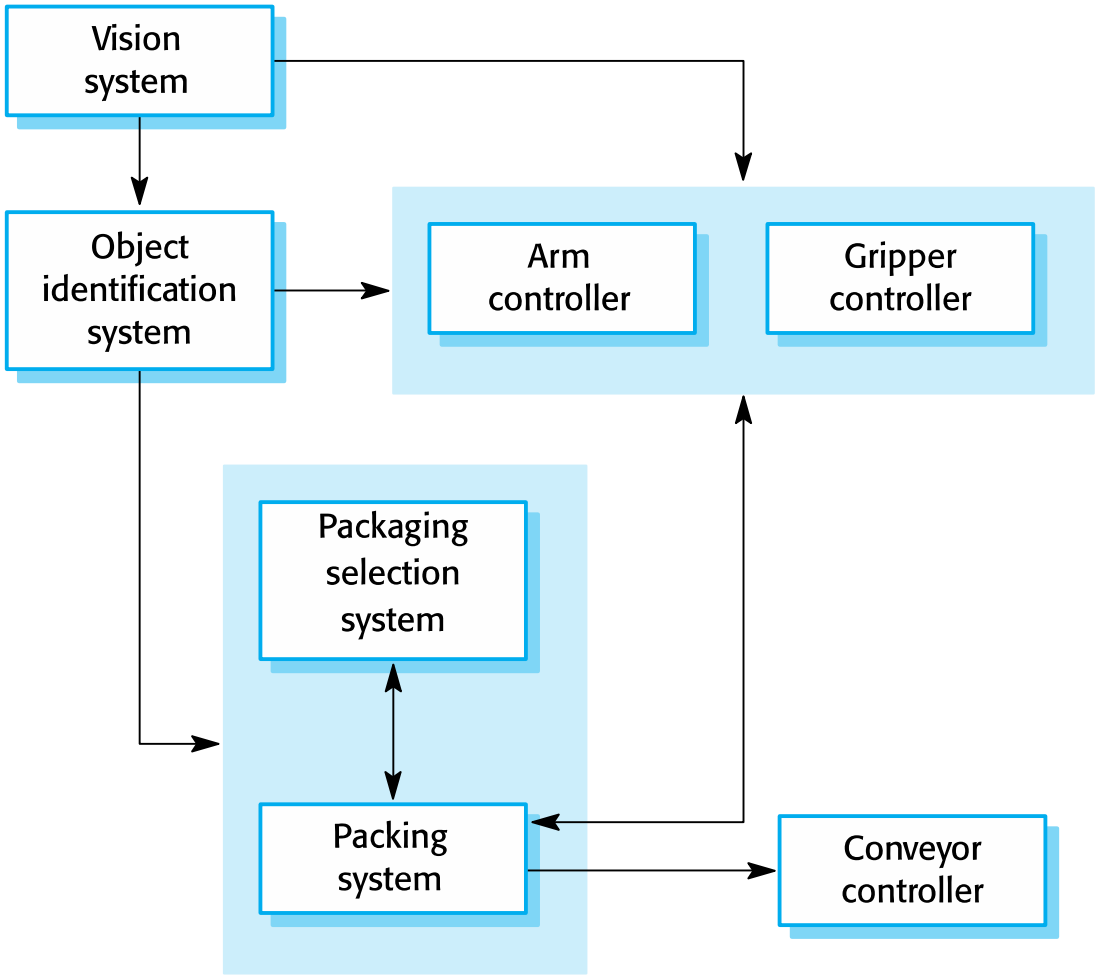
# Architectural representations : Block diagrams


Simple, informal **block diagrams showing entities and relationships** are the most frequently used method for documenting software architectures.

But these have been **criticised** 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**.

# The architecture of a packing robot control system

# 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


As a way of facilitating discussion about the system design

A high-level architectural view of a system is useful for communication with stakeholders and project planning because it is not cluttered with detail.

Stakeholders can relate to it and understand an abstract view of the system.

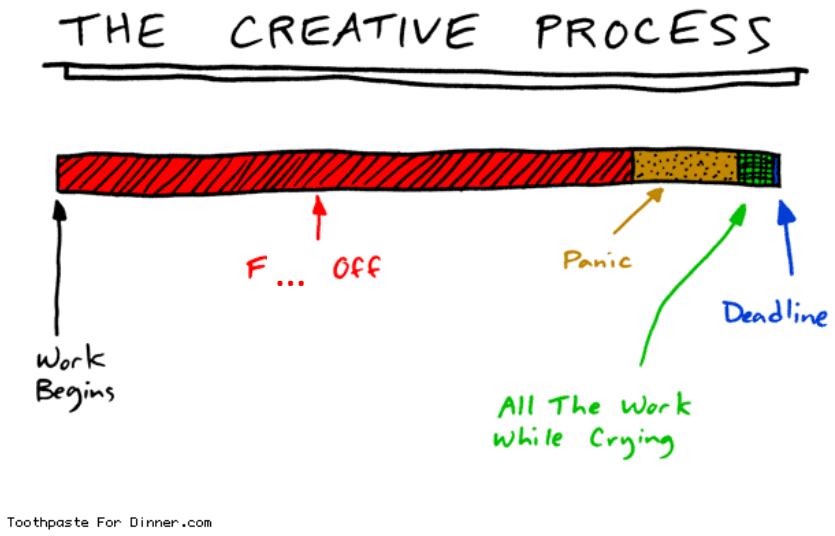
They can then discuss the system as a whole without being confused by detail.

As a way of 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.

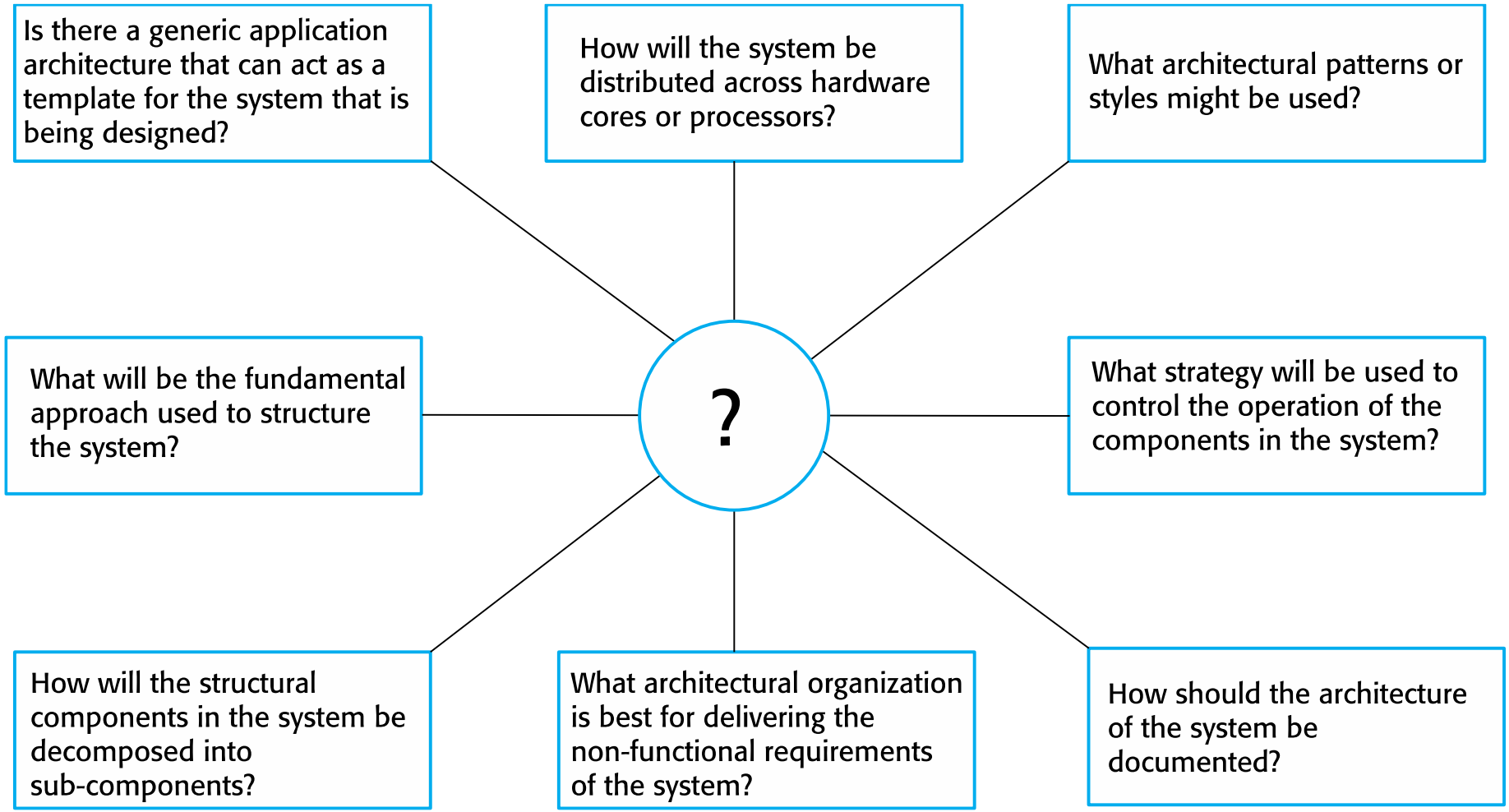

# Architectural design decisions

# Architectural design decisions

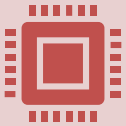

- Architectural design is a creative process so the process differs depending on the type of system being developed.
- However, a number of common decisions span all design processes and these decisions affect the NF characteristics of the system.

# 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.

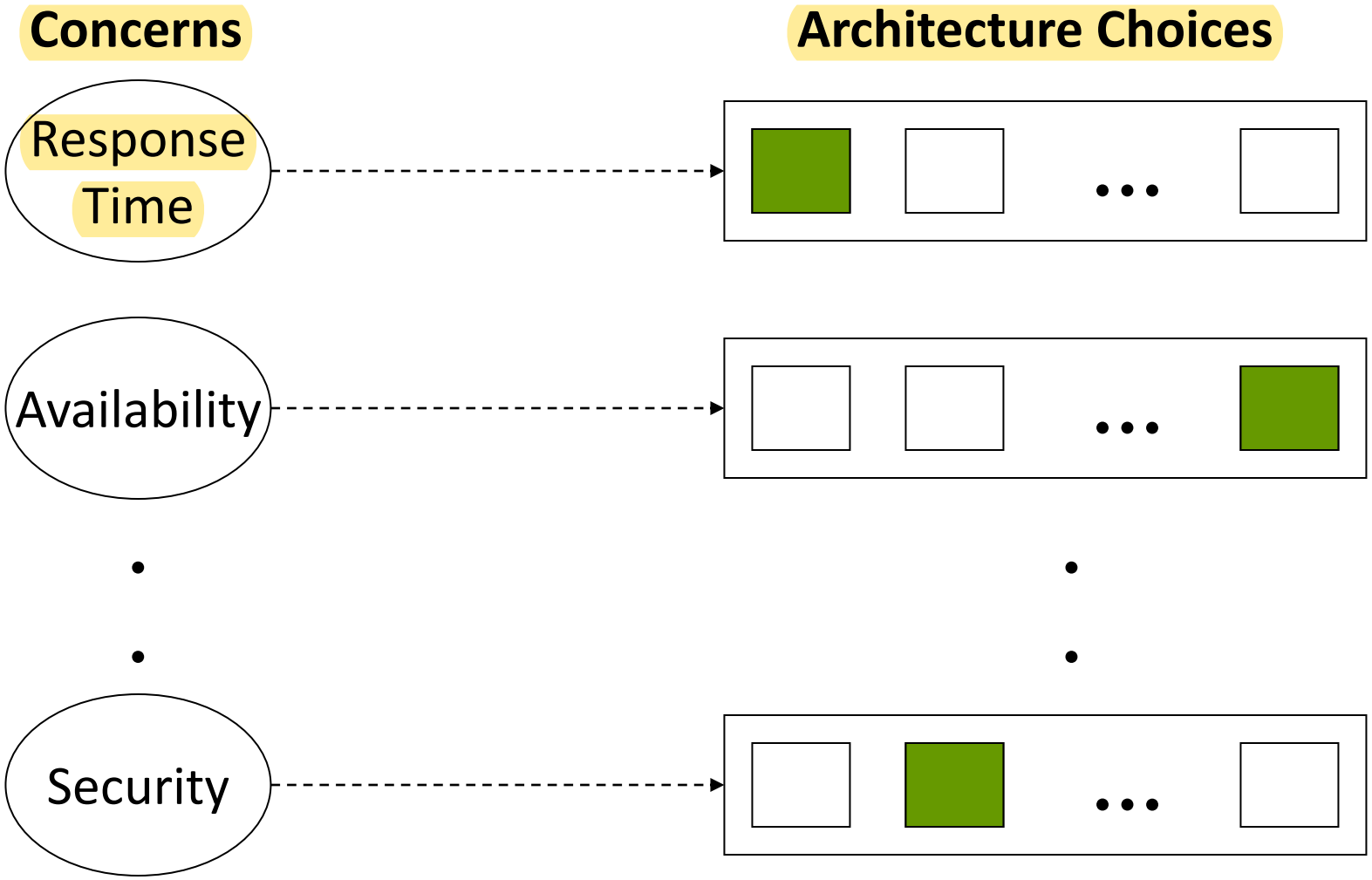
The architecture of a system may be designed around one of more architectural patterns or 'styles'.

These capture the essence of an architecture and can be instantiated in different ways.

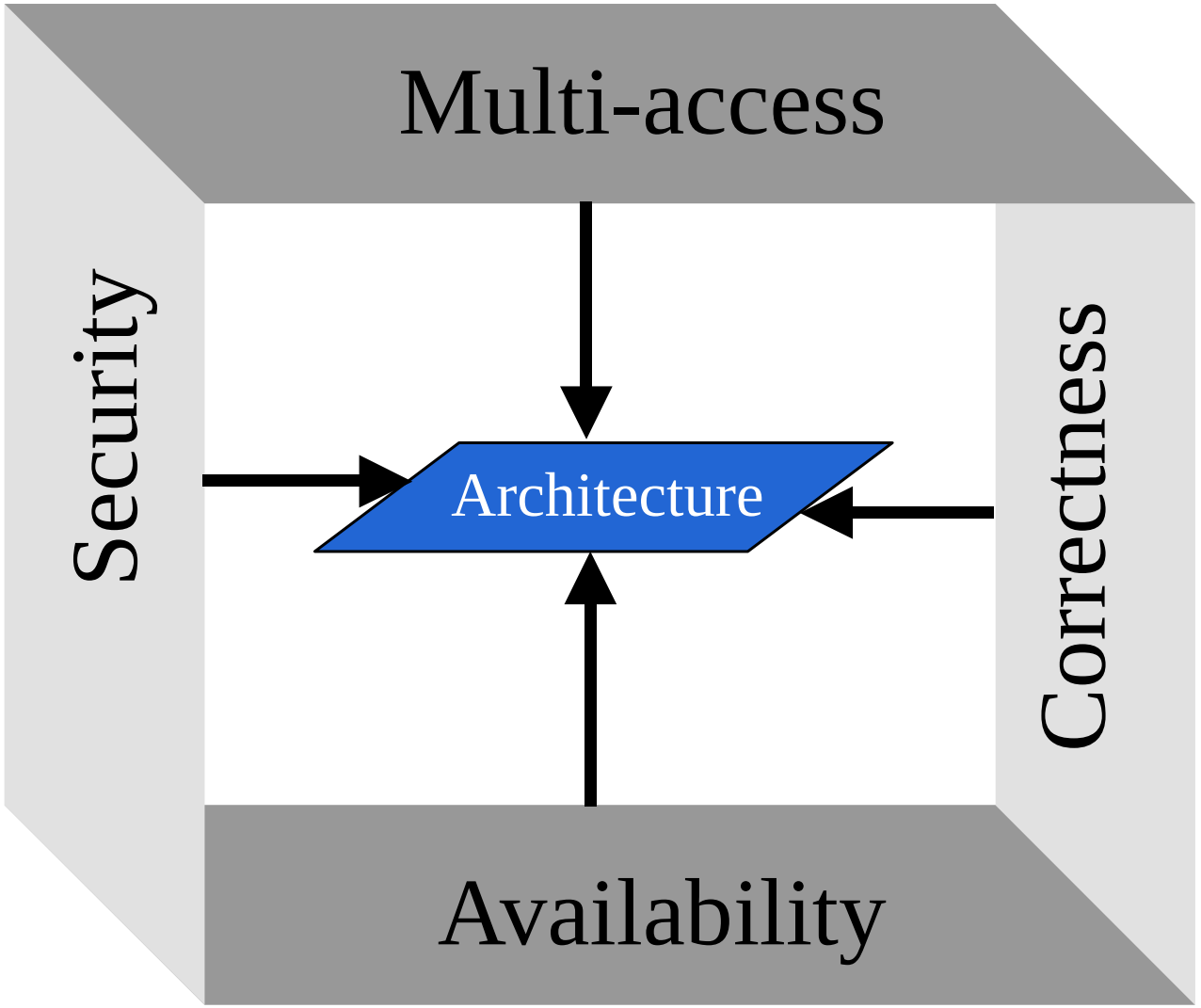
# Software quality attributes


| | | |
|---|---|---|
| Safety | Understandability | Portability |
| Security | Testability | Usability |
| Reliability | Adaptability | Reusability |
| Resilience | Modularity | Efficiency |
| Robustness | Complexity | Learnability |

# Architectural Choices

# Where Does the Final Architecture Reside?

# Conflicts between quality attributes


| Concerns | Response Time | Availa-bility | Security | Legal Issues | Compat-ibility | Correct-ness | Multi-Access |
|---|---|---|---|---|---|---|---|
| Response Time | | + | – | | | – | – |
| Availability | + | | - | | | + | + |
| Security | - | + | | | | + | - |
| Legal Issues | | | | | + | + | |
| Compatibility | + | + | + | | | + | + |
| Correctness | - | + | | | + | | - |
| Multi-Access | - | - | - | | | - | |

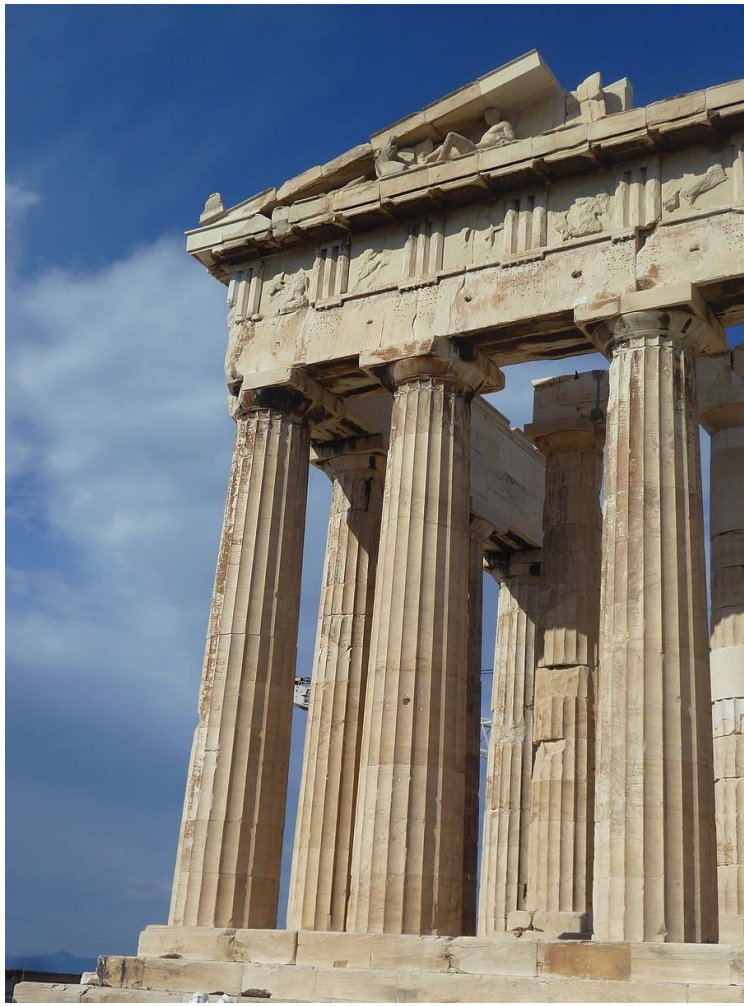
# Architecture and quality attributes


| | |
|---|---|
| Performance | • Localise critical operations and minimise communications. Use large rather than fine-grain components. |
| Security | • Use a layered architecture with critical assets in the inner layers. |
| Safety | • Localise 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 views

# 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.



# Views and UML diagrams

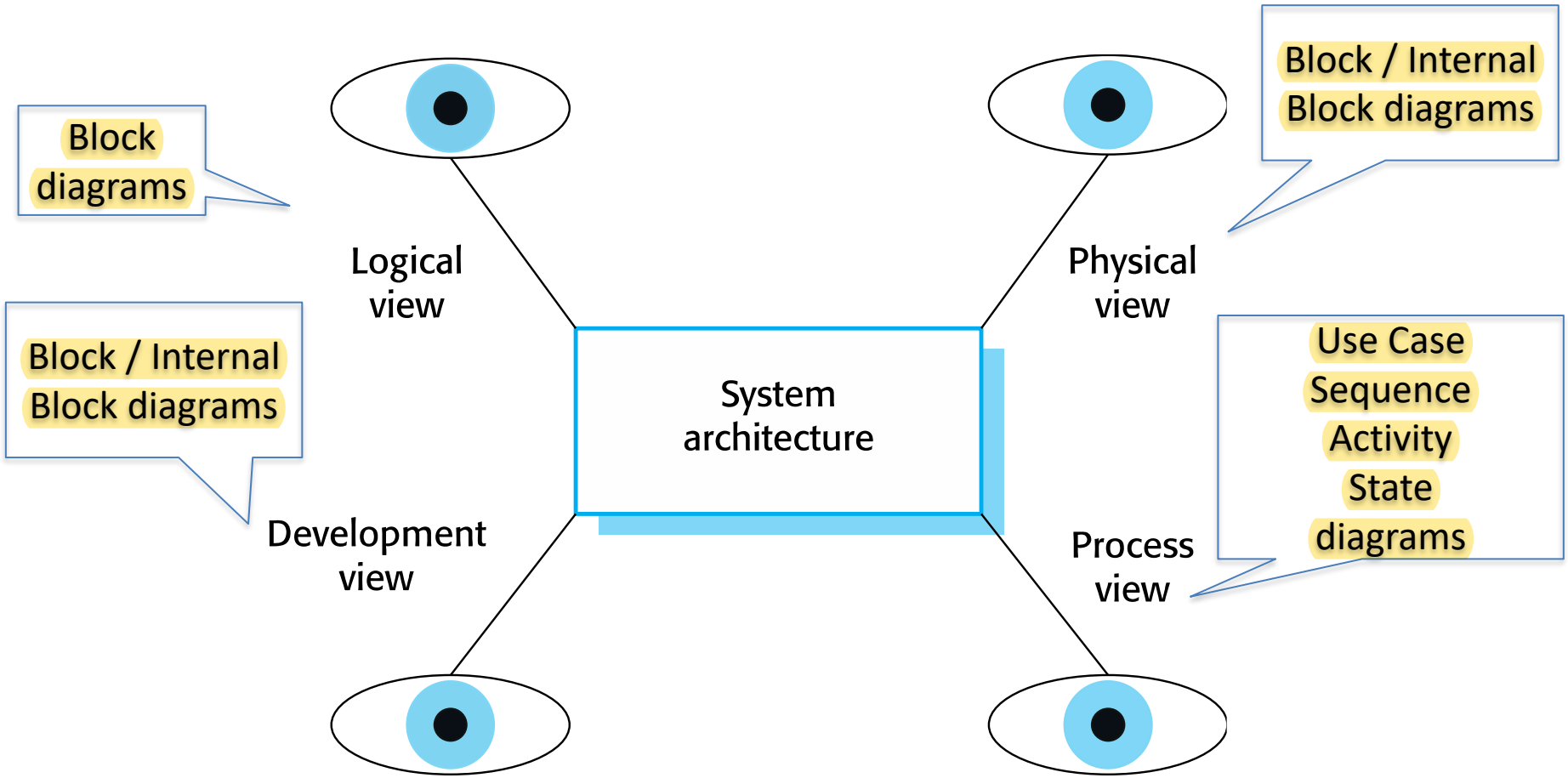

**4+1 View Model (Philippe Kruchten)**

- Logical View
- Process View
- Deployment View
- Development View
- Scenario View
  Or Use Case View

**UML (industry standard)**

- Class Diagram
- Use Case Diagram
- Object diagram
- State-chart diagram
- Sequence diagram
- Collaboration diagram
- Activity diagram
- Component diagram
- Deployment diagram

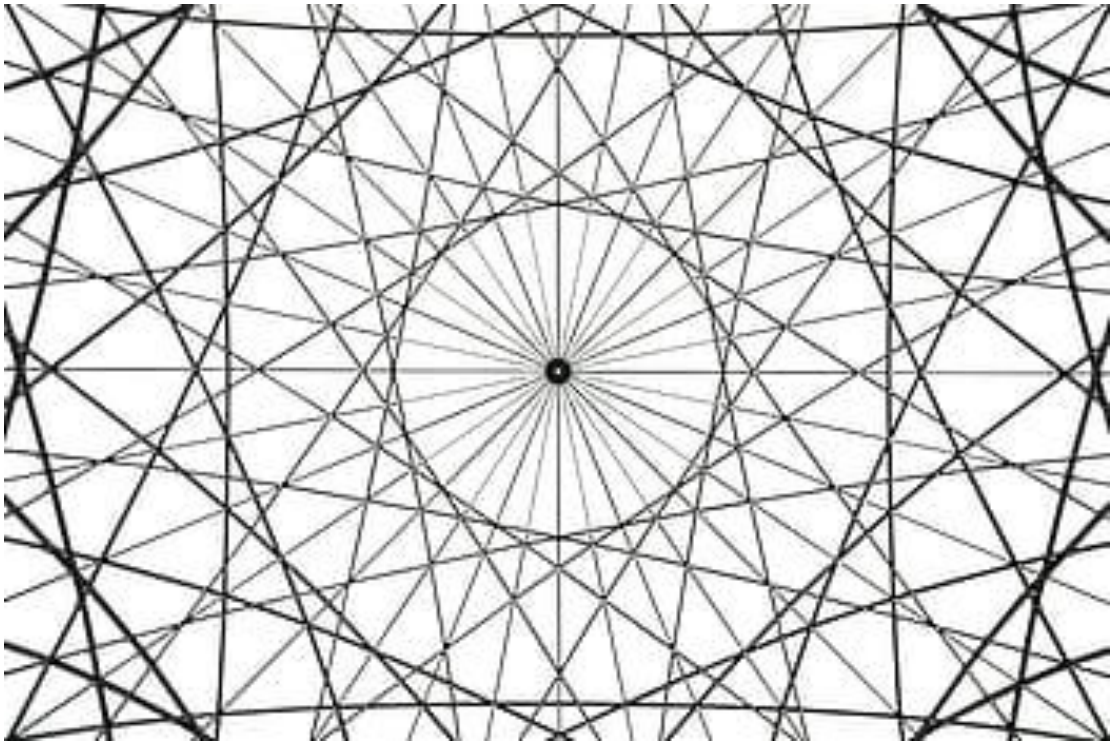
# Architectural views and SysML

# 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 (deployment) view, which shows the system hardware and how software components are distributed across the processors in the system.
- **Use case view**, shows use cases (+1)


# Architectural patterns



# 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.

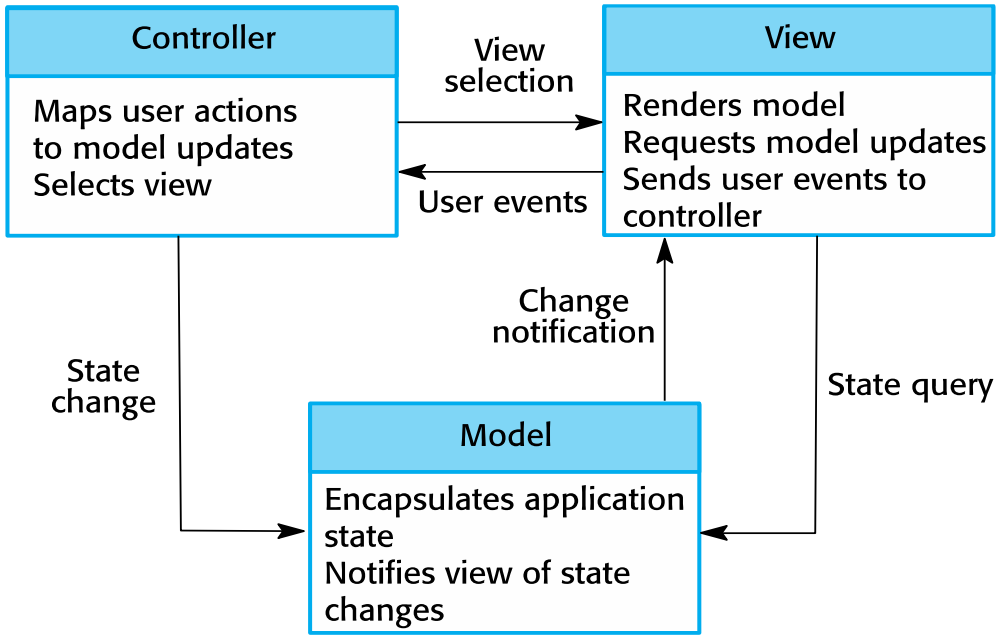
Patterns should include information about when they are and when they are not useful.

Patterns may be represented using tabular and graphical descriptions.

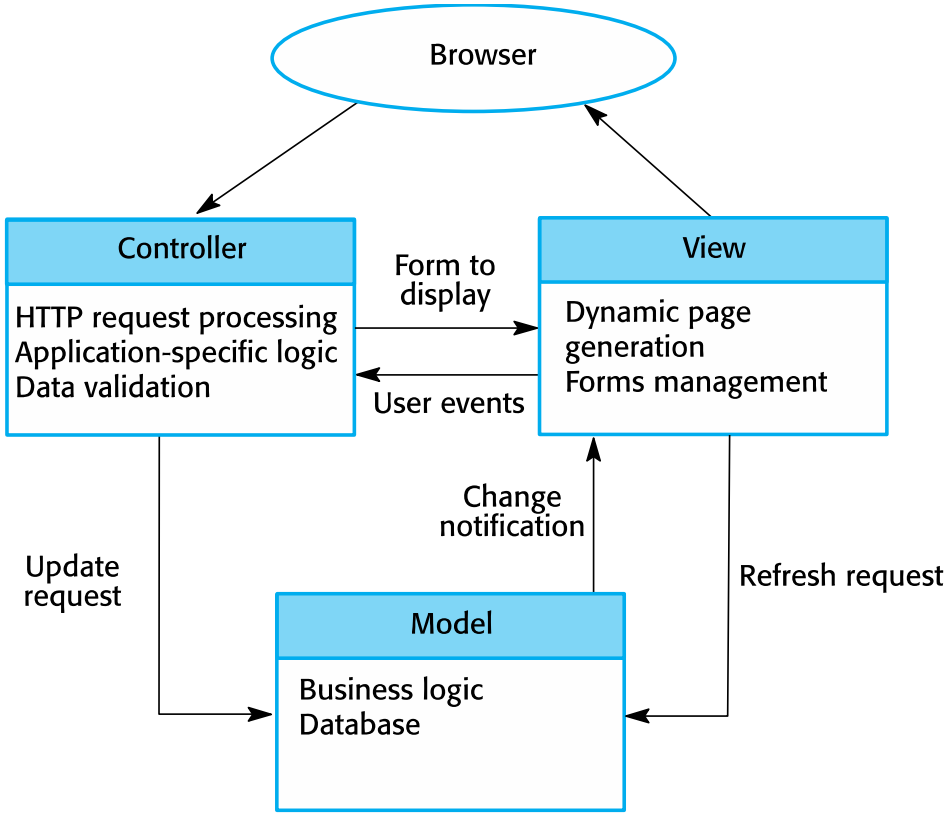
# The Model-View-Controller (MVC) pattern


| 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** | Next Figure 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, i.e. the complexity is high to develop the applications using this pattern.<br>Not right suitable for small applications which has adverse effect in the application's performance and design. |

# 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.

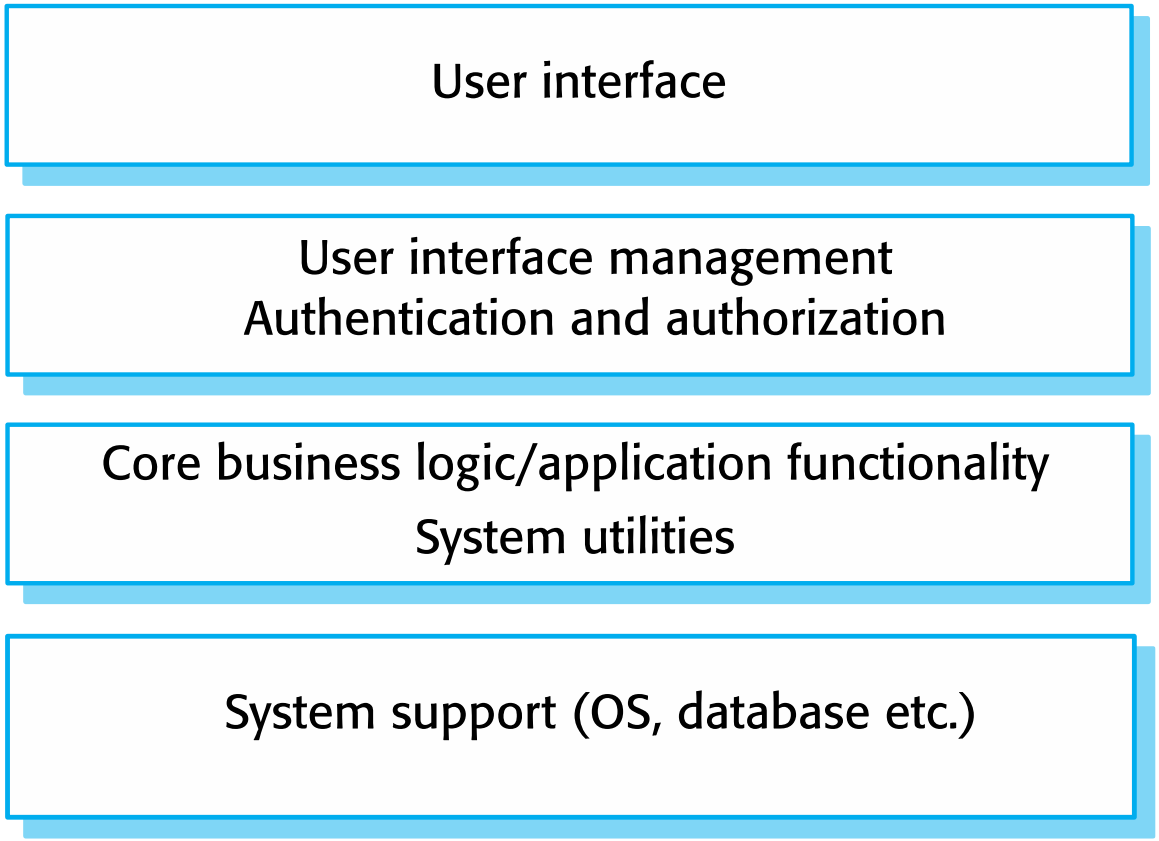
Supports the incremental development of sub-systems in different layers. When a layer interface changes, only the adjacent layer is affected.

However, often artificial to structure systems in this way.

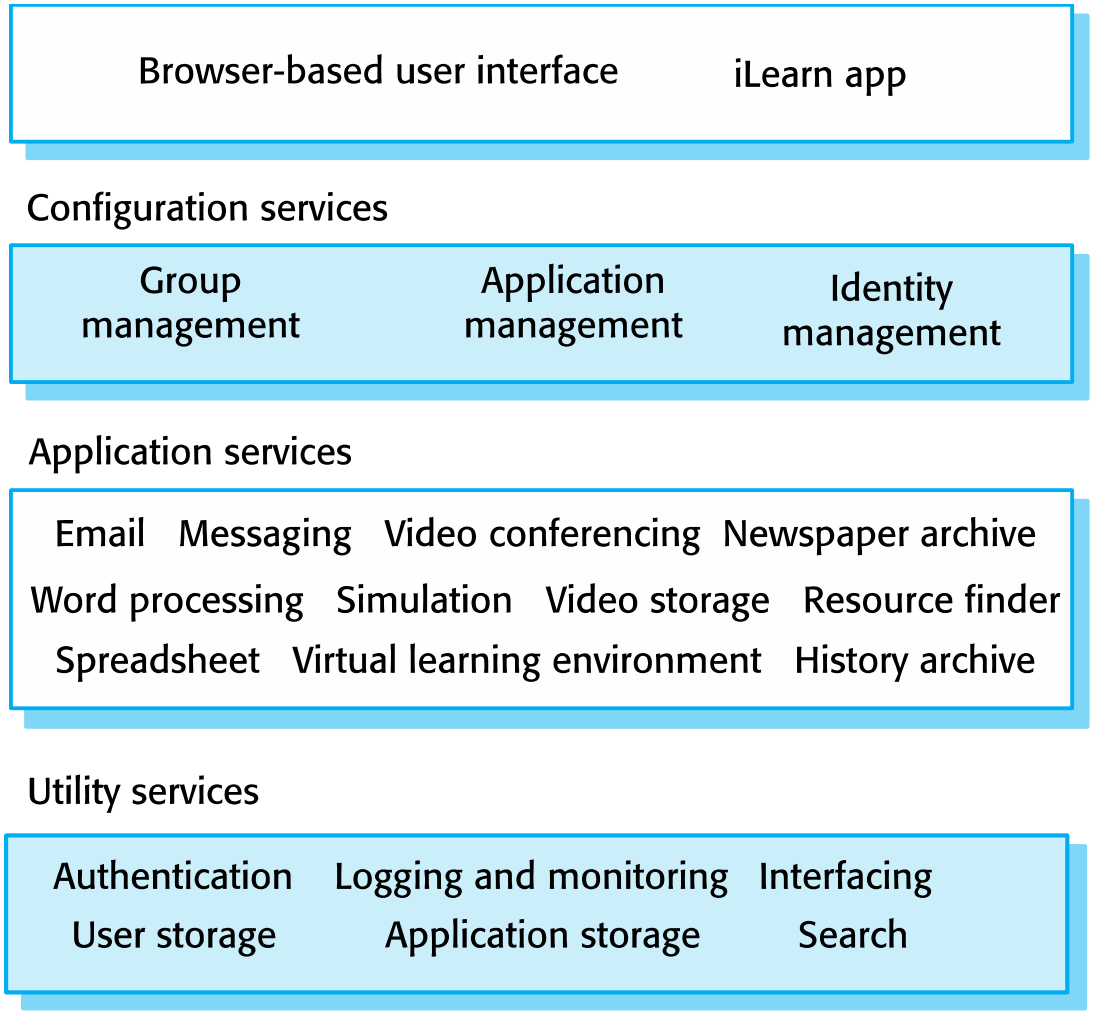
# The Layered architecture pattern


| Name | Layered architecture |
|---|---|
| **Description** | • Organizes the system into layers with related functionality associated with each layer.<br>• 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;<br>• when the development is spread across several teams with each team responsibility for a layer of functionality;<br>• when there is a requirement for multi-level security. |
| **Advantages** | • Allows replacement of entire layers so long as the interface is maintained.<br>• 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.<br>• Performance can be a problem because of multiple levels of interpretation of a service request as it is processed at each layer. |

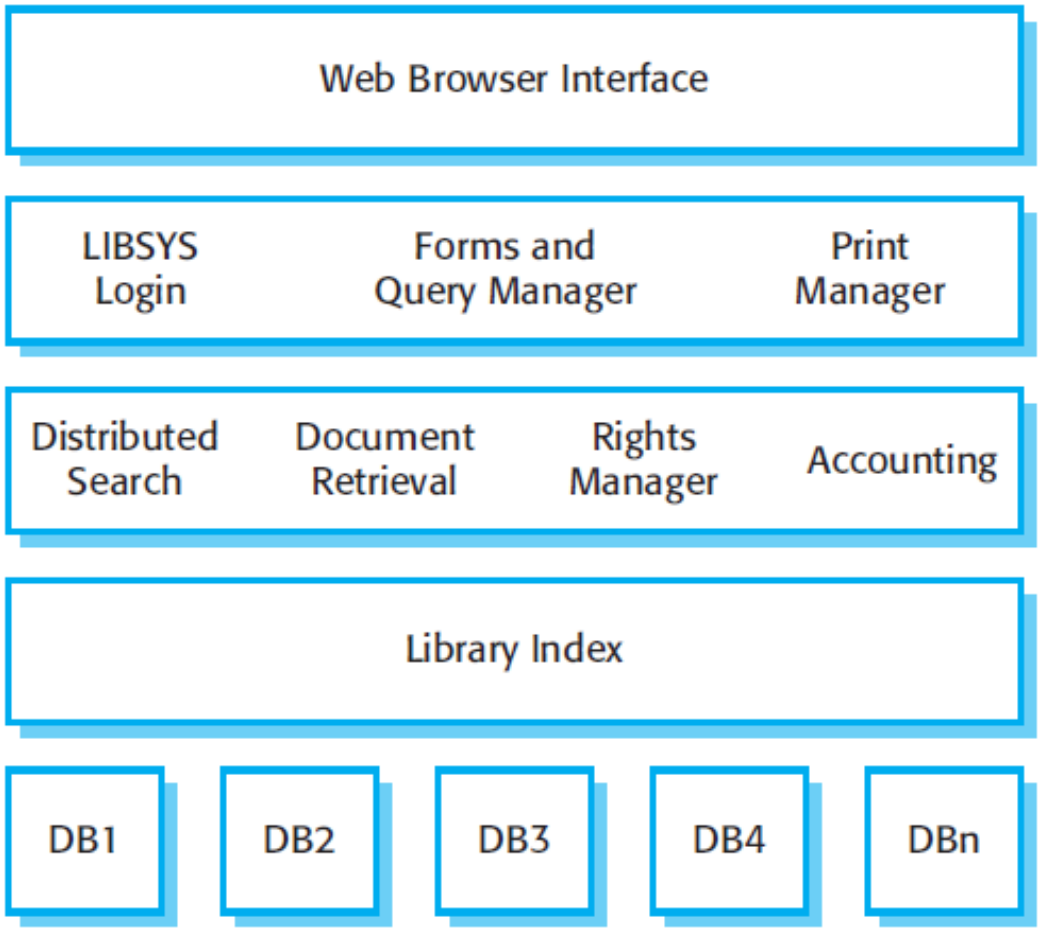
# A generic layered architecture

# The architecture of the iLearn system

# The architecture of the Library system

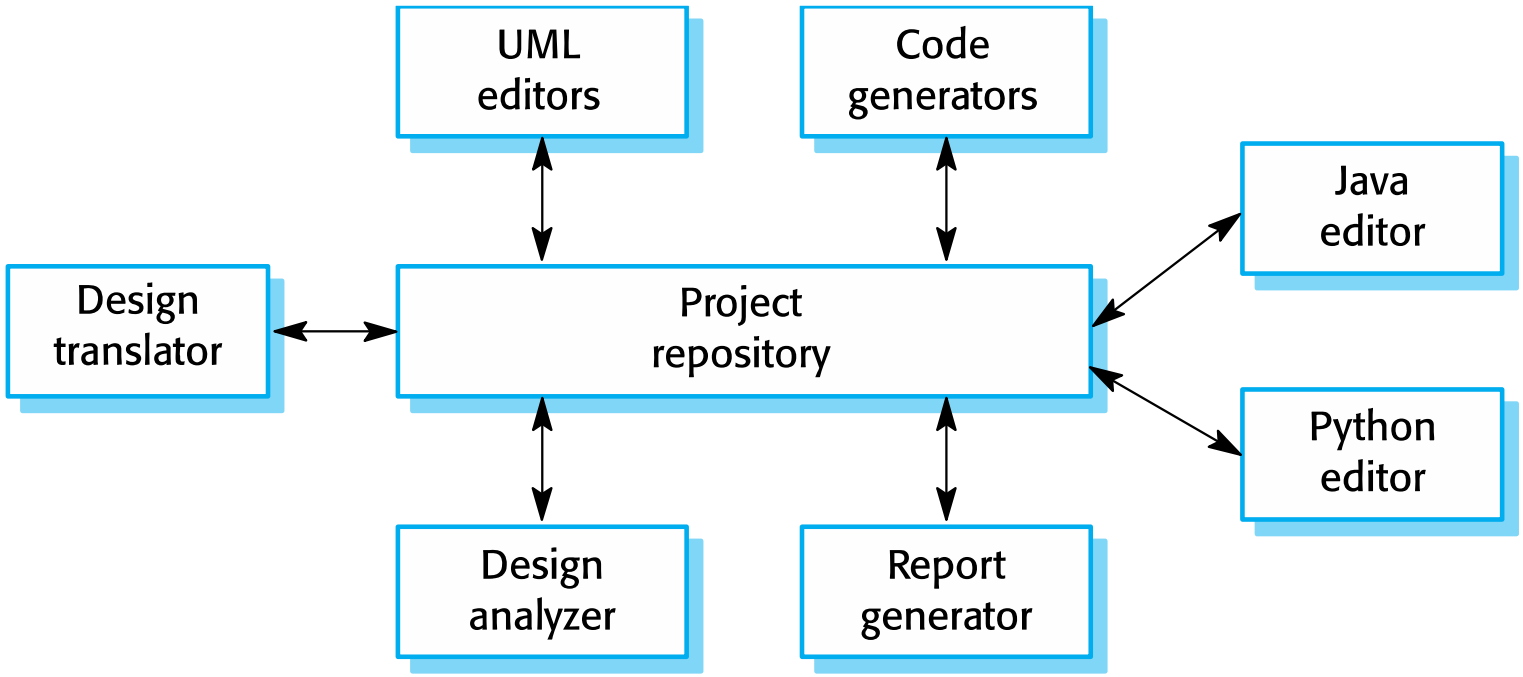
# Repository architecture


- 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 as 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.<br>• Components do not interact directly, only through the repository. |
| **Example** | • IDE where the components use a repository of system design information.<br>• 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 to be stored for a long time.<br>• 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.<br>• Changes made by one component can be propagated to all components.<br>• 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.<br>• May be inefficiencies in organizing all communication through the repository.<br>• 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.

Set of stand-alone servers which provide specific services such as printing, data management, etc.

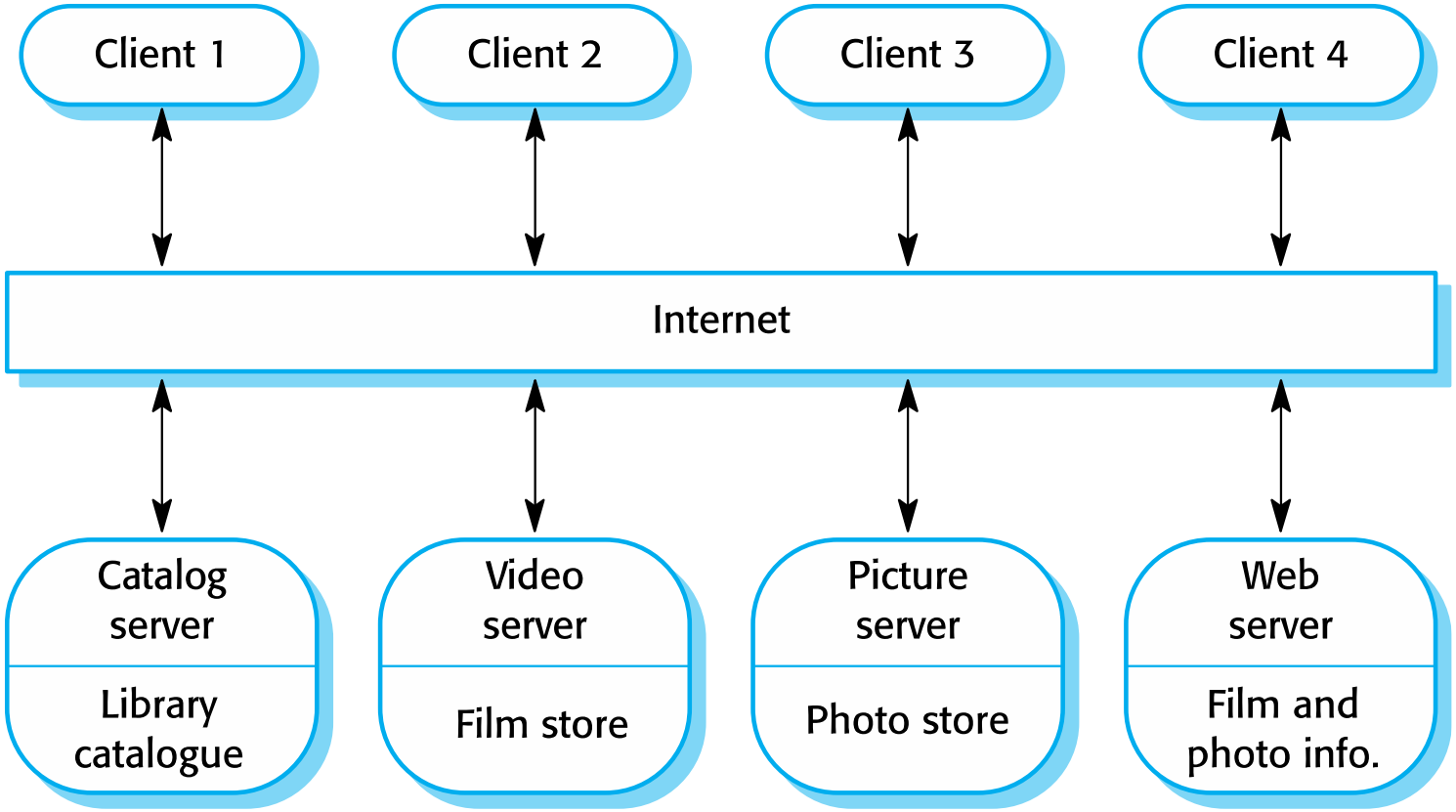
Set of clients which call on these services.

Network which allows clients to access servers.

# The Client–server pattern

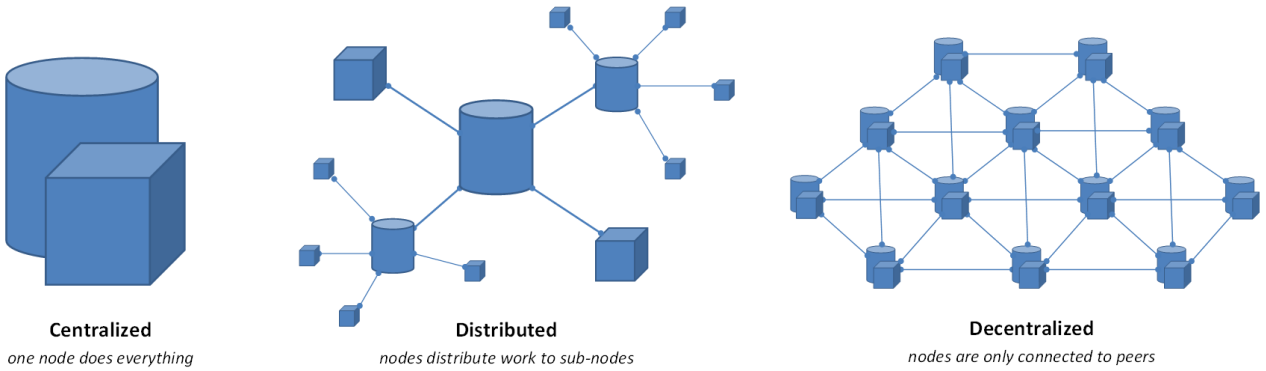
| Name | Client-server |
|---|---|
| **Description** | • In a client–server architecture, the functionality of the system is organized into services, with each service delivered from a separate server.<br>• Clients are users of these services and access servers to make use of them. |
| **Example** | • 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.<br>• 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.<br>• 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.<br>• Performance may be unpredictable because it depends on the network as well as the system.<br>• May be management problems if servers are owned by different organizations. |

# A client–server architecture for a film library

# Control models

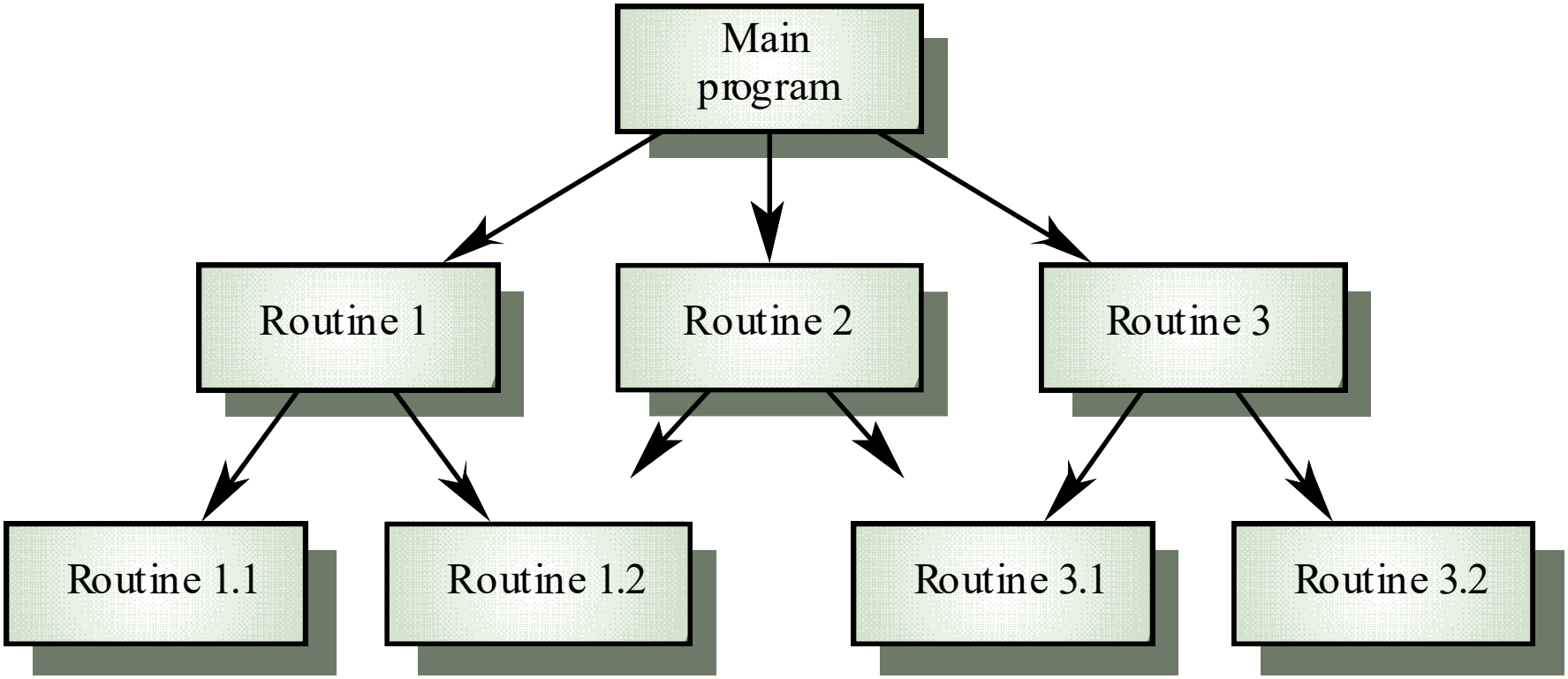

- Are concerned with the control flow between sub-systems. Distinct from the system decomposition model
- Centralised control
  - One sub-system has overall responsibility for control and starts and stops other sub-systems
- Event-based control
  - Each sub-system can respond to externally generated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



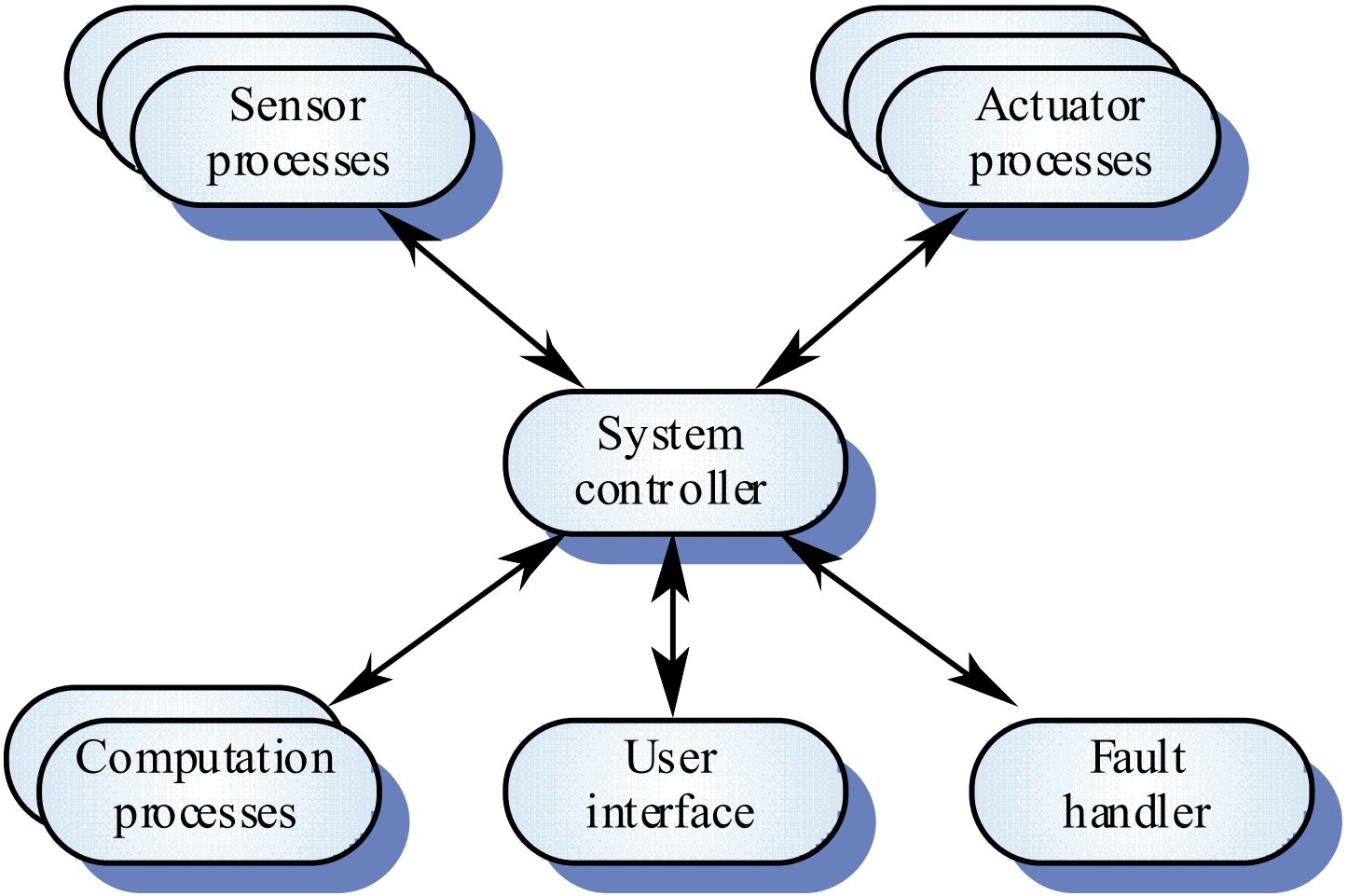
# Centralised control


- 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

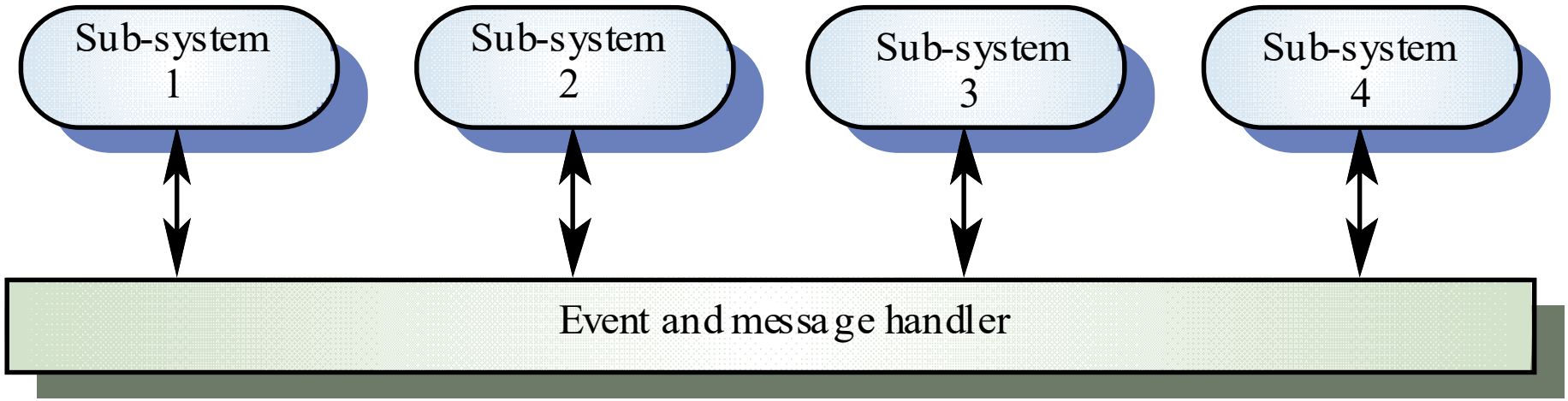

Driven by externally generated events where the timing of the event is outwith the control of the sub-systems which process the event

Example: Broadcast models. An event is broadcast to all sub-systems. Any sub-system which can handle the event may do so

# 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

# 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.