Software Engineering Theory and Practice

School of Computing	
Title	Software Engineering Theory and Practice
Module Coordinator	Steven Ossont
Email	steven.ossont@port.ac.uk
Code	M30819
Moodle	https://moodle.port.ac.uk/course/view.php?id=11429

U30819: Software Engineering Theory and Practice

Architectural Design

https://moodle.port.ac.uk/course/view.php?id=11429

Steven Ossont steven.ossont@port.ac.uk

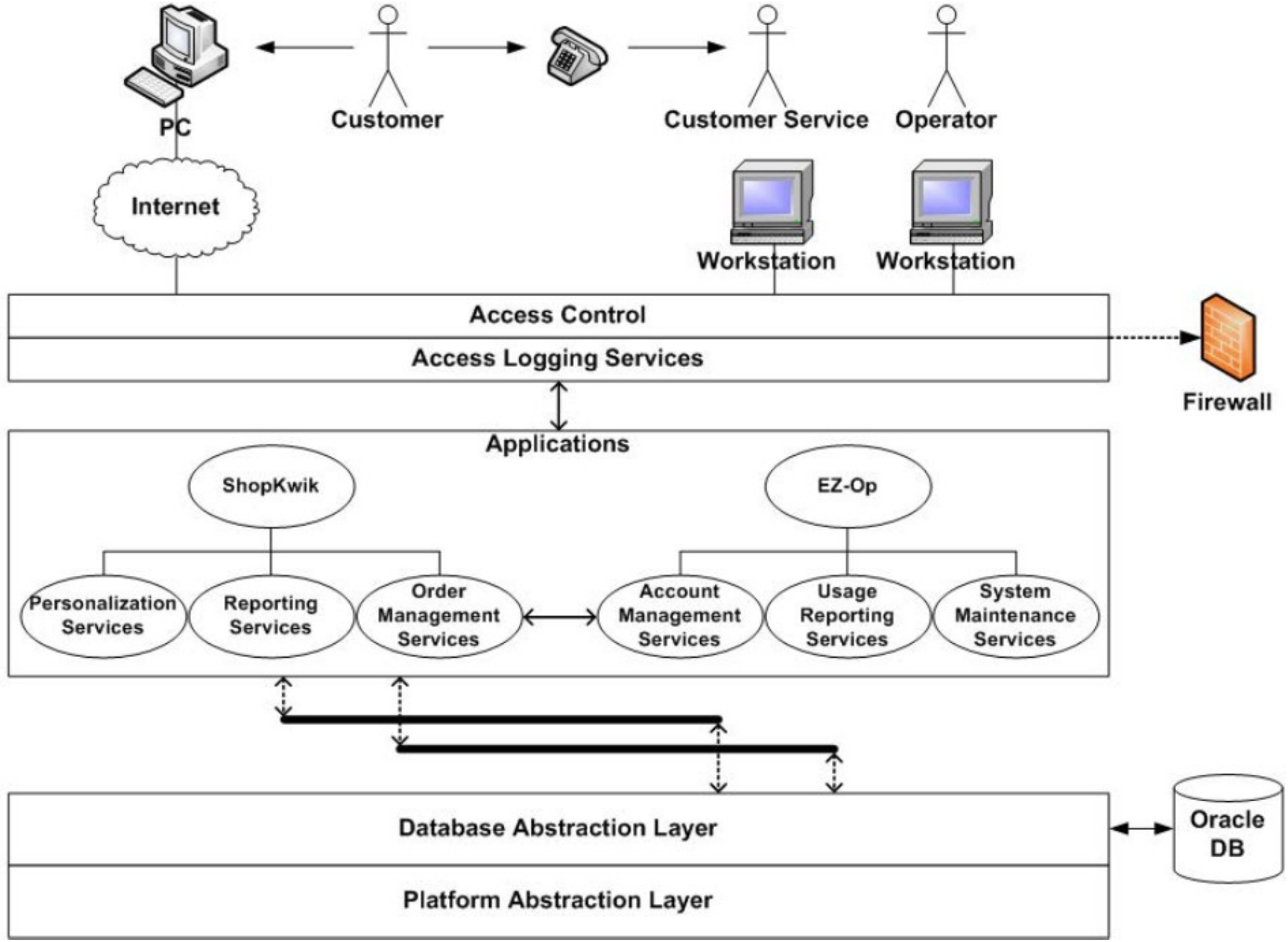
Architectural Design

Understanding how a system should be organized and designing the overall structure of that system

Output: architectural design model

Software Architecture is:

1. how the defining components of a software system are organized and assembled.
2. how they communicate with each other.
3. the constraints that govern the whole system


PC
Customer
Customer Service
Operator
Internet
Workstation
Workstation
Access Control
Access Logging Services
Firewall
Applications
ShopKwik
EZ-Op
Personalization Services
Reporting Services
Order Management Services
Account Management Services
Usage Reporting Services
System Maintenance Services
Database Abstraction Layer
Platform Abstraction Layer
Oracle DB

What does the Picture Tell Us?

- The system consists of many elements
- The elements interact with each other
- Some of the elements represent layers and their relationship to one another
- The applications involved and the elements they comprise
- The system has multiple layers
- The data, control, and communications mechanisms that are used

What does the Picture Omit?

- What is the nature of the elements?
  - What is the significance of their separation?
  - Do they exist at runtime?
  - Do they run at separate times?
  - Are the processes, programs, or hardware?
  - Are they objects, tasks, functions, or processes?
  - Are they distributed programs or systems?
- What are the responsibilities of the elements?
  - What do they do?
  - What functions do they provide in the context of the system?

What does the Picture Omit?

- What is the significance of the connections between the elements?
  - Do the elements communicate with each other?
  - Do the elements control each other?
  - Send data to each other?
  - Use each other?
  - Invoke each other?
  - Synchronize with each other?
- What is the significance of how the elements are positioned on the diagram?

Architectural model

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 is the critical link between design and requirements engineering, as it identifies the main structural components in a system and the relationships between them.

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

Architectural Design Decisions

What style of house – bungalow, 2-storey house, cottage, townhouse?

- What **architectural patterns** or styles might be used?

What are the rooms designed for at each floor?

- How will the structural components in the system be decomposed into subcomponents?

How are the connecting stairs between the two floors designed?

- What strategy will be used to control the operation of the components in the system?

Architectural Design Decisions

Where will the doors/windows go?

- What architectural organization is best for delivering the non-functional requirements of the system?

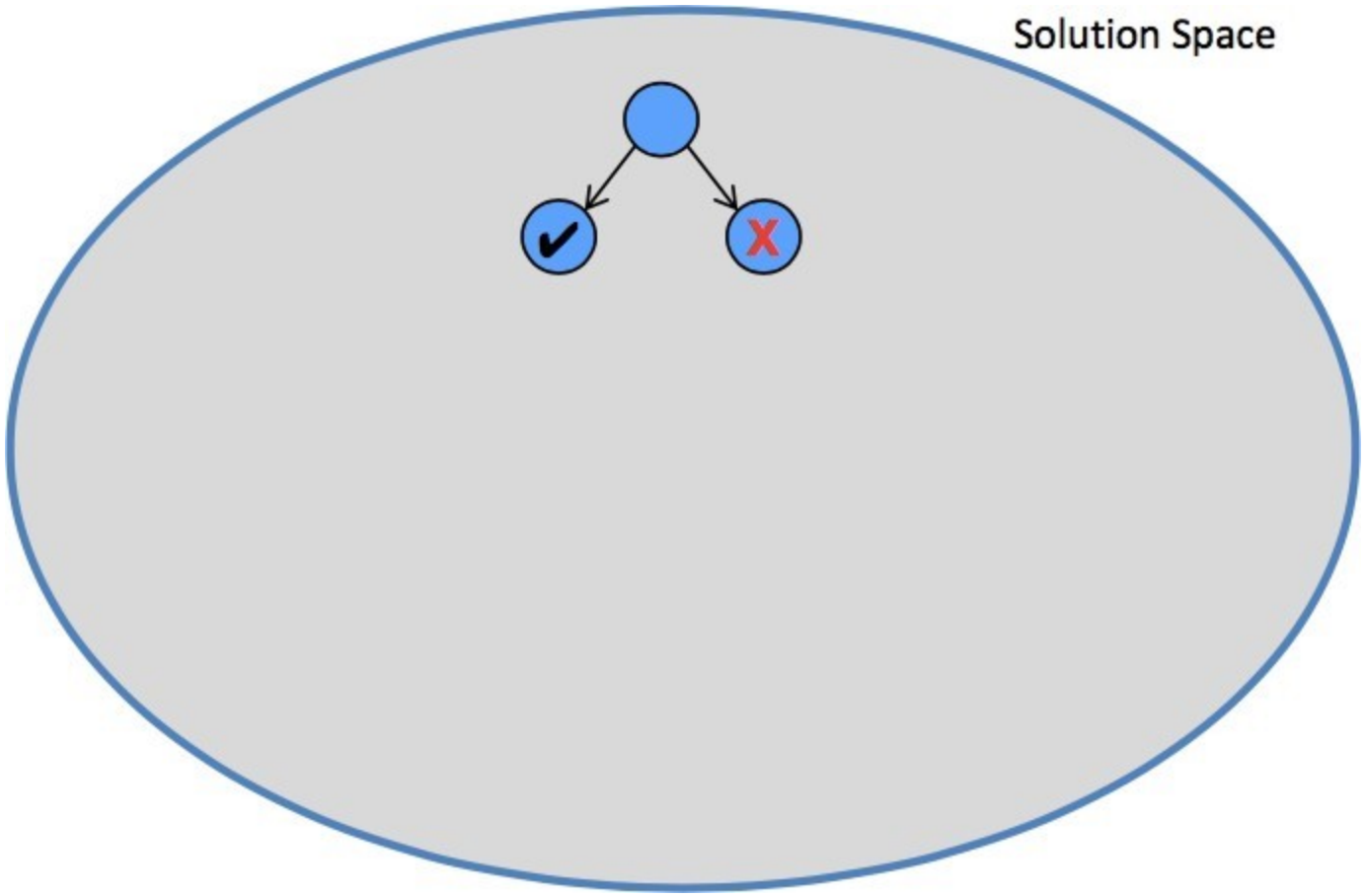
How will we test that the house complies with all regulations for new homes?

- How will the architectural design be evaluated?

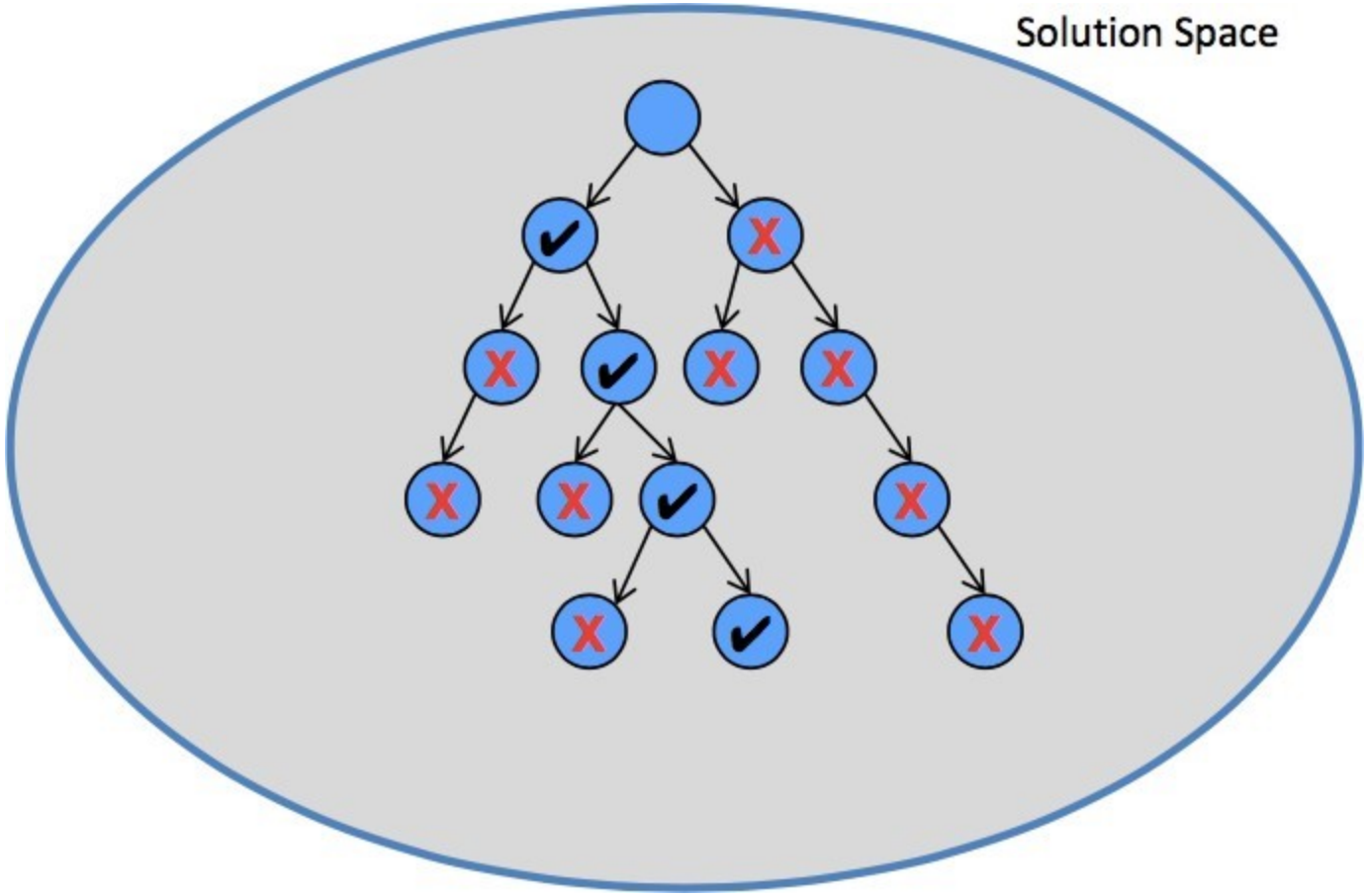
What type of blueprint will be used to represent the house?

- How should the architecture of the system be documented?

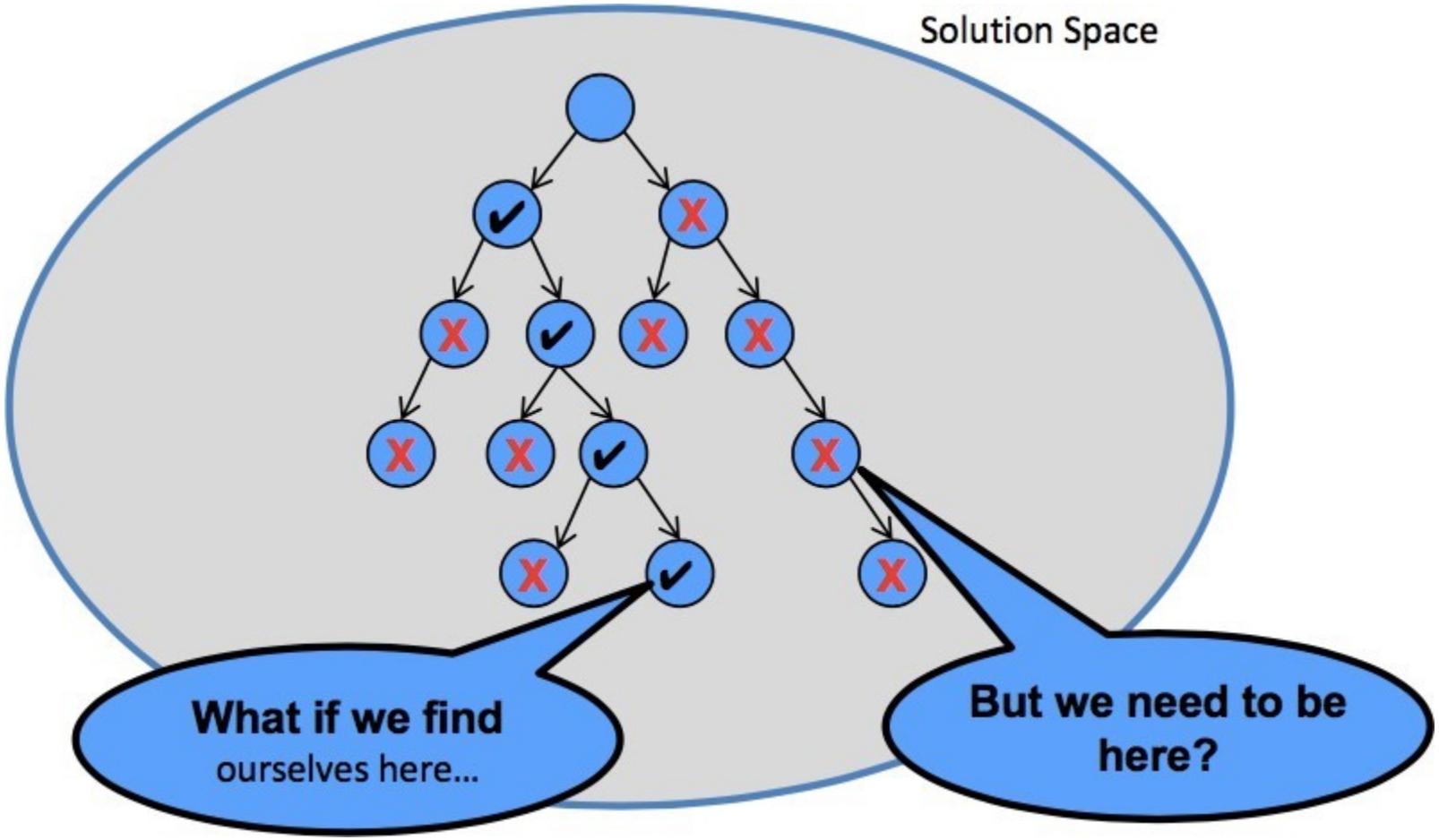
Why Architectural Decisions Matter?



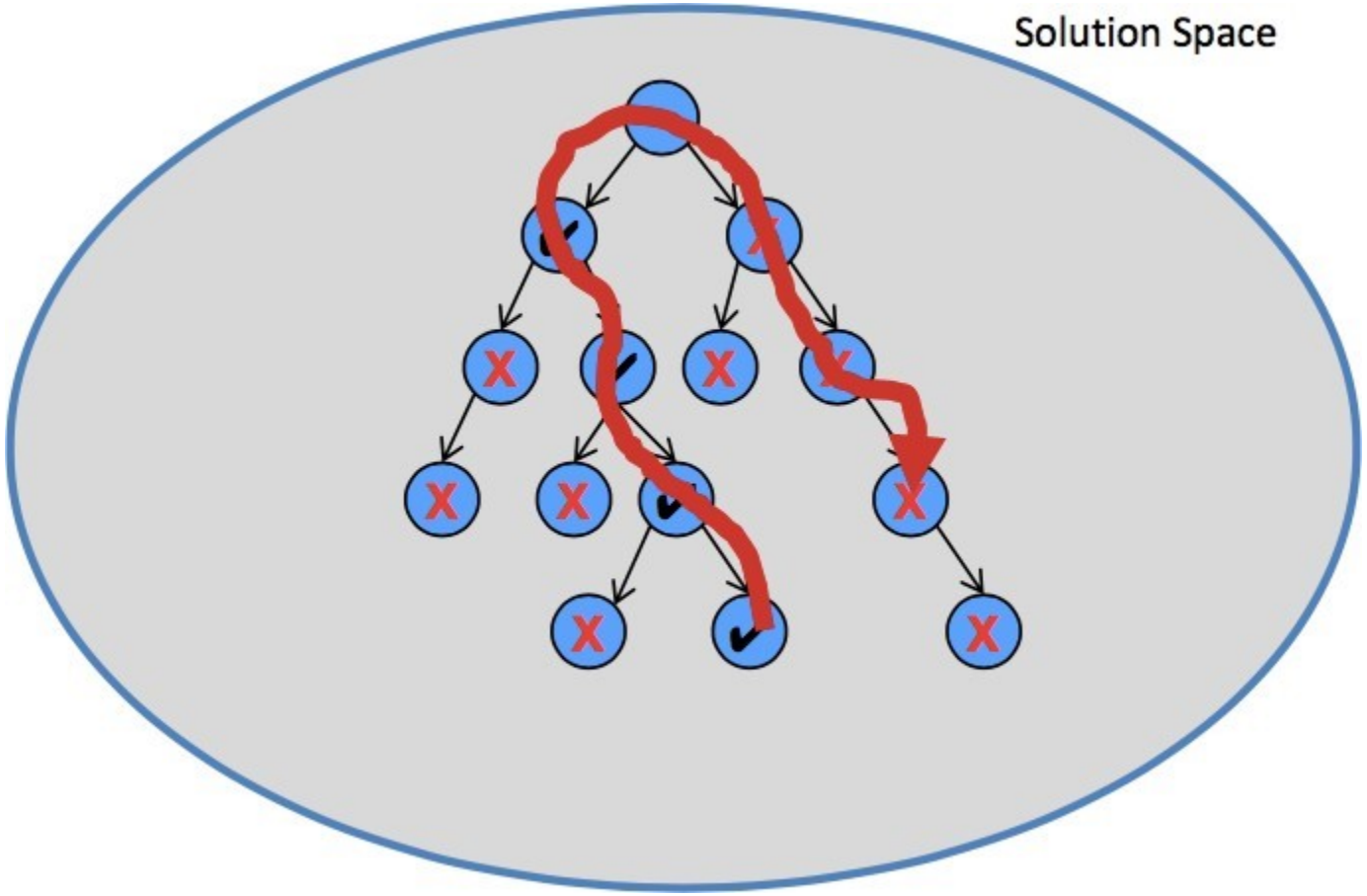
Why Architectural Decisions Matter?



Why Architectural Decisions Matter?



Why Architectural Decisions Matter?



Architectural abstraction

- Architecture (small scale) 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 (large scale) 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
- Product-line architectures may be developed.

Architectural representations

- 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.

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 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.

As a way of documenting an architecture that has been designed

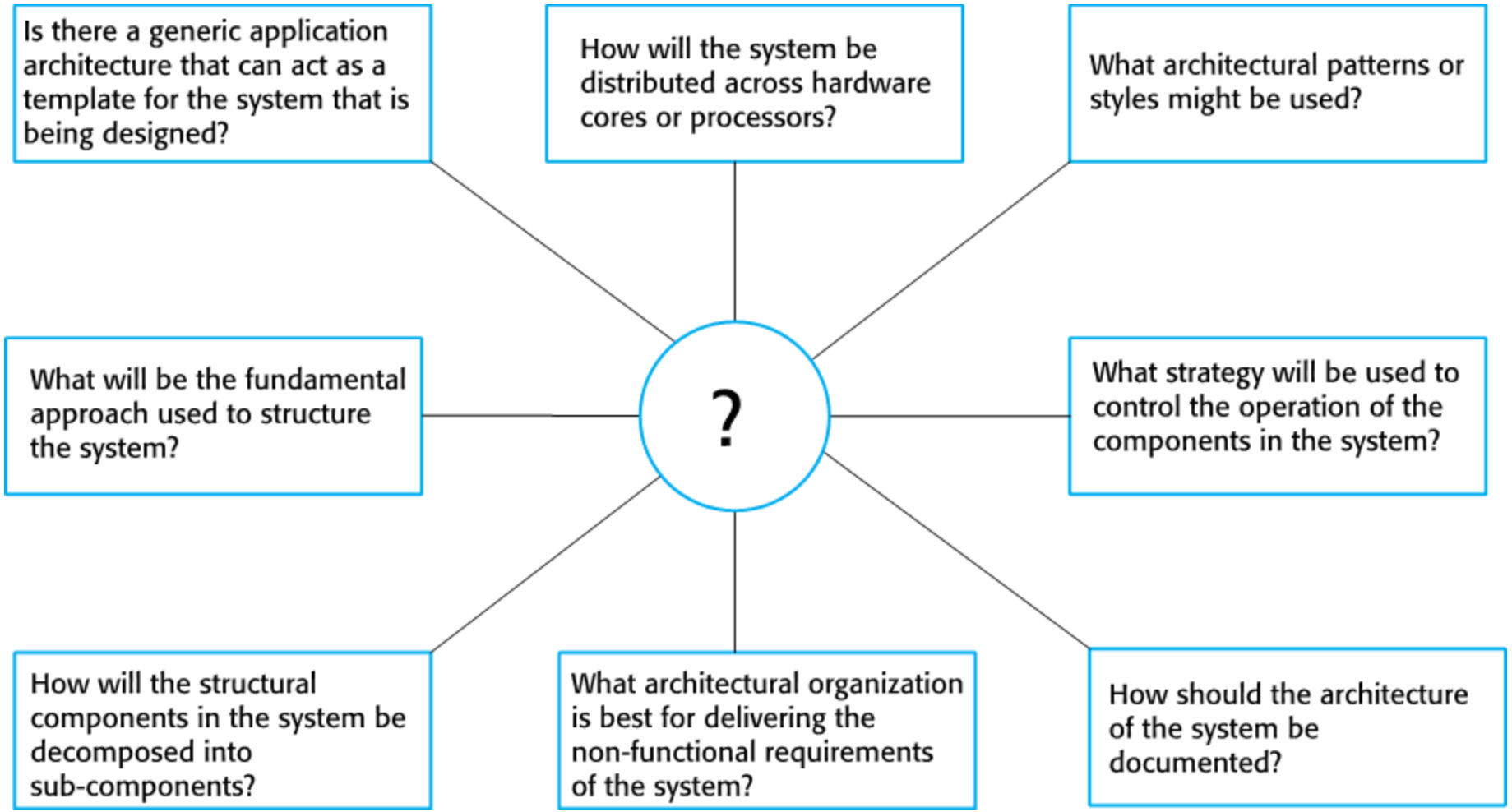
- The aim here is to produce a complete system model that shows the different components in a system, their interfaces and their connections.

Architectural design decisions

Architectural design is a creative process and differs on the system being developed

- 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
- 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’

Architecture and system characteristics

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.

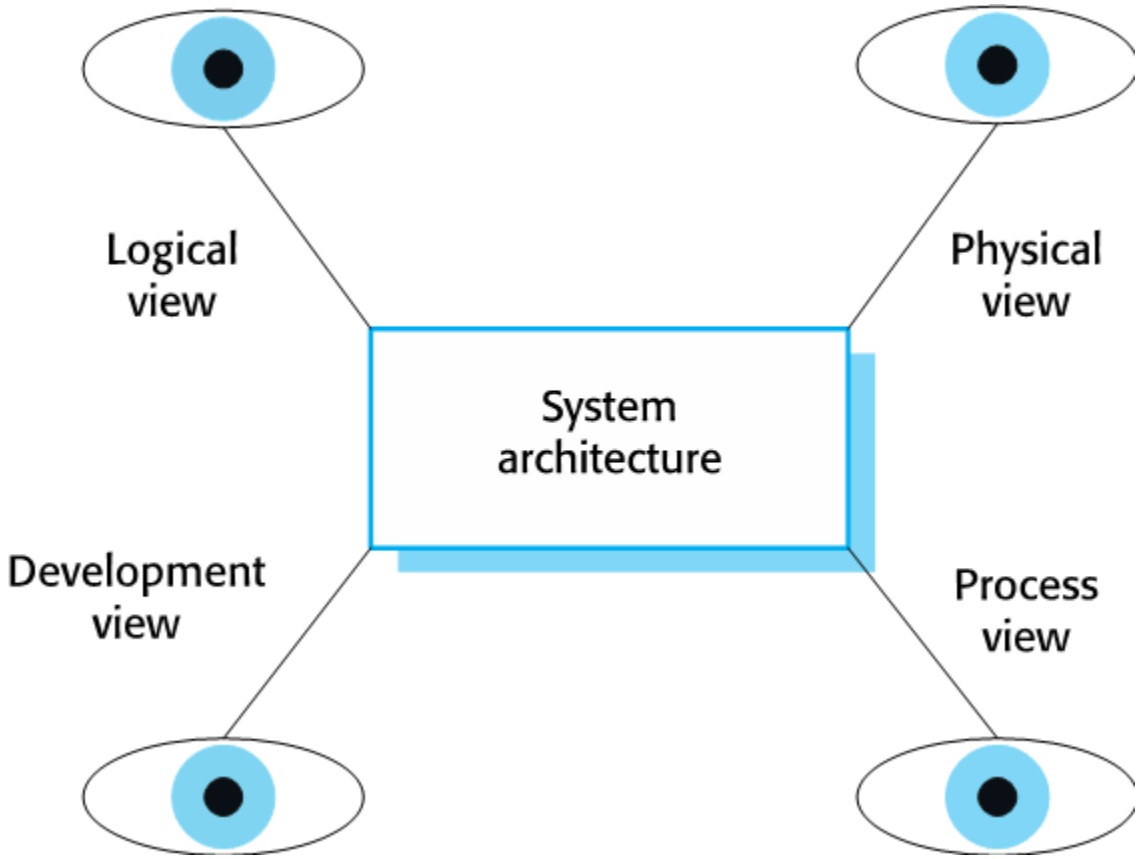
Availability

- Include redundant components and mechanisms for fault tolerance.

Maintainability

- Use fine-grain, replaceable components.

Architectural views



> Each architectural model only shows one view or perspective of the system.

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)

Representing architectural views

- The Unified Modeling Language (UML) a notation for describing and documenting system 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 the are not useful.
- Patterns may be represented using tabular and graphical descriptions.

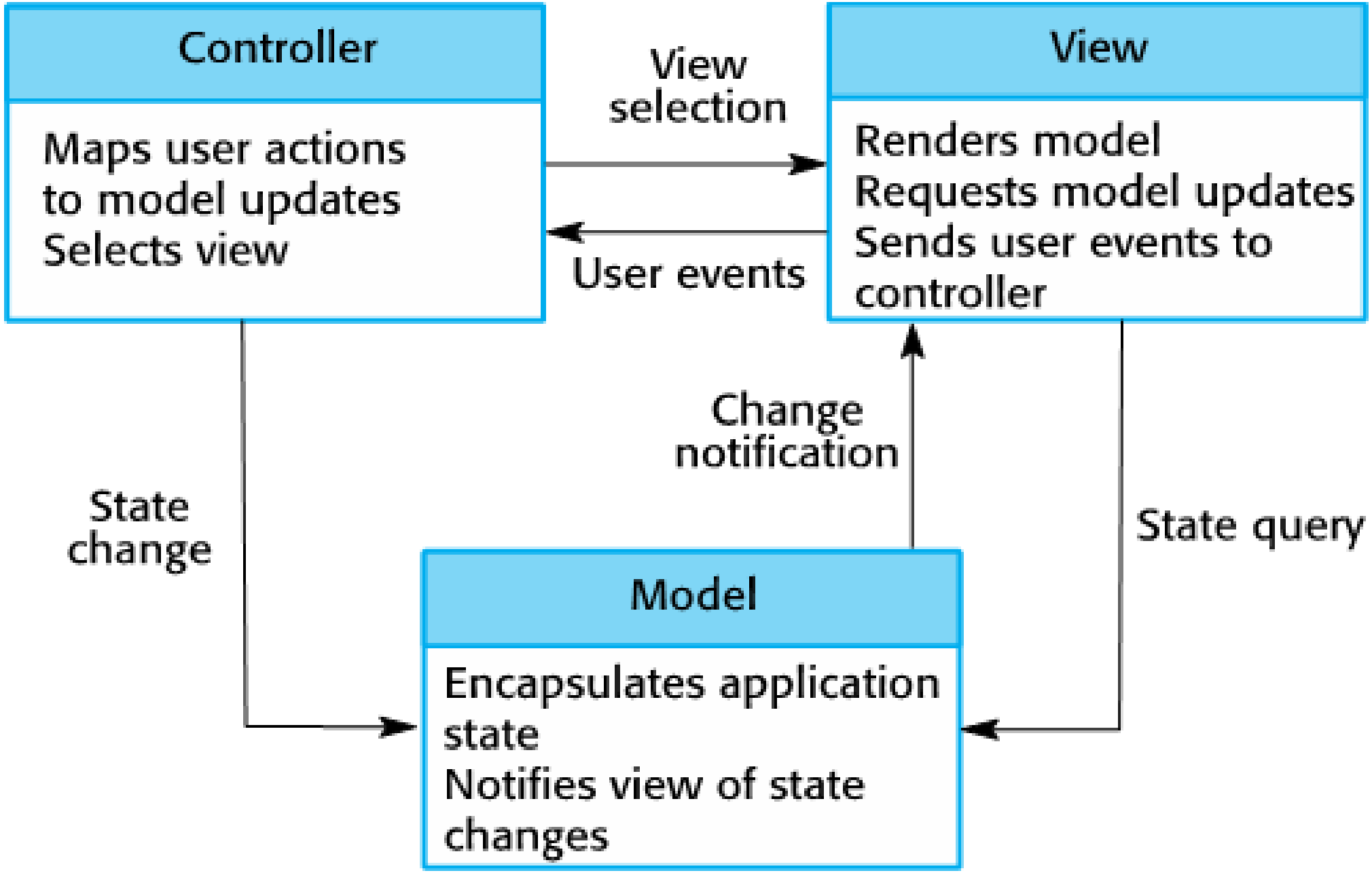
The Model-View-Controller (MVC) pattern

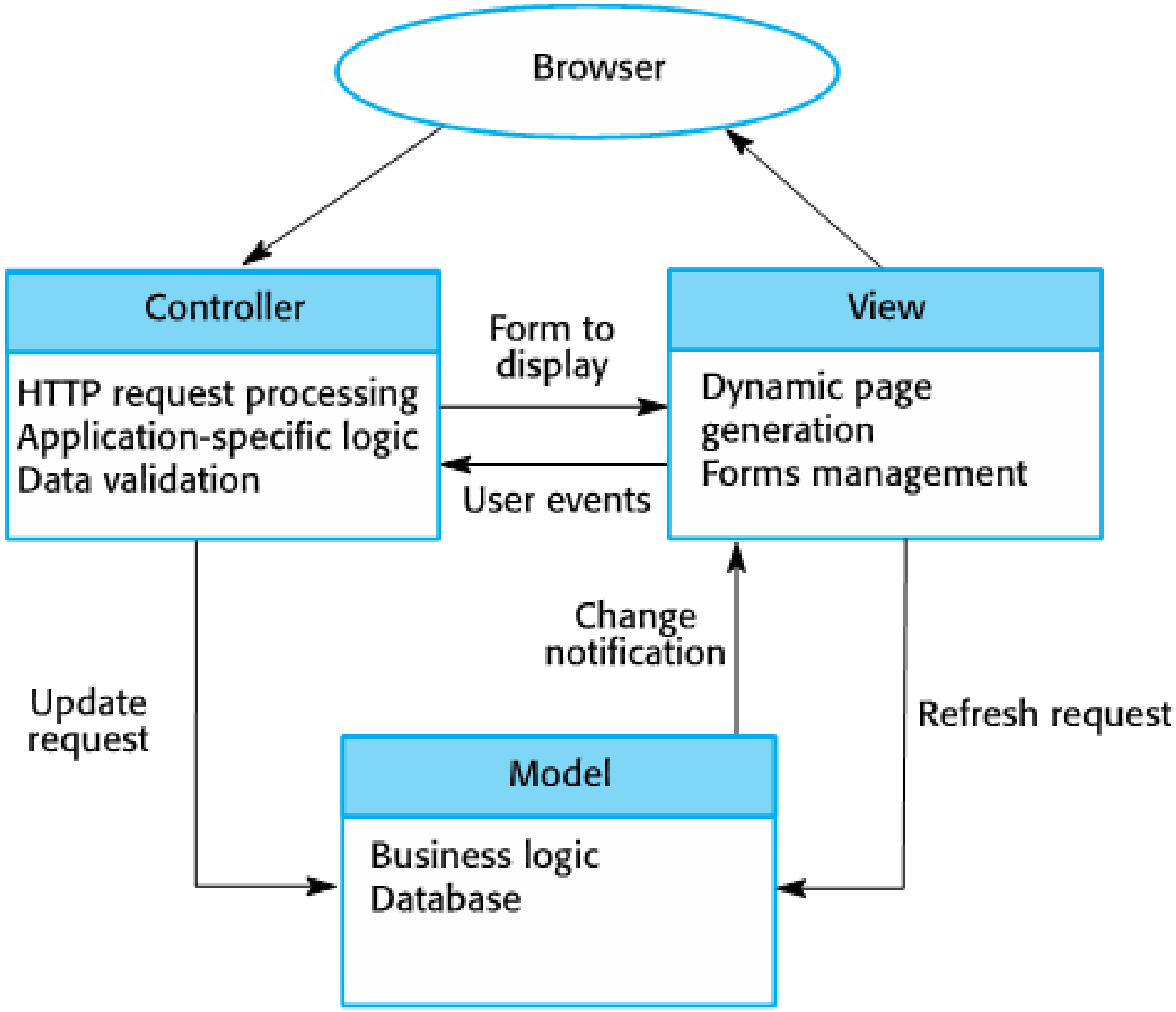
The Model-View-Controller (MVC) pattern

Separates presentation and interaction from the system data

- Model: manages the data and operations on the data
- View: defines and manages how data is presented to the user
- Controller: manages user interaction and passes interaction to the View and the Model

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.


Controller
Maps user actions to model updates
Selects view
View selection
User events
View
Renders model
Requests model updates
Sends user events to controller
State change
Change notification
State query
Model
Encapsulates application state
Notifies view of state changes

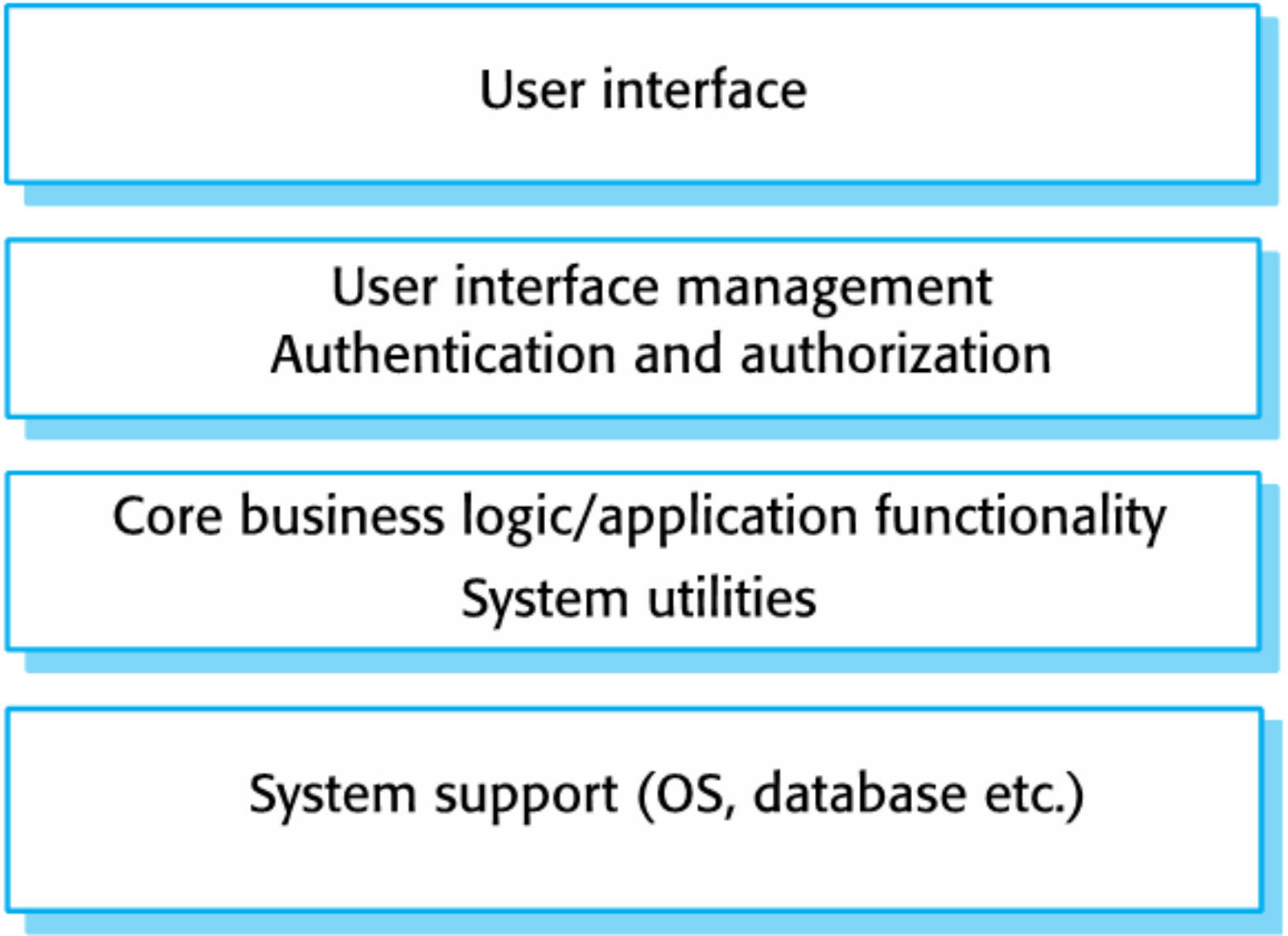

Browser
Controller
HTTP request processing
Application-specific logic
Data validation
Form to display
User events
View
Dynamic page generation
Forms management
Change notification
Update request
Refresh request
Model
Business logic
Database

Layered architecture

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.
- Can artificial to structure systems in this way.

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. See Figure 6.6.
Example	A layered model of a system for sharing copyright documents held in different 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 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.

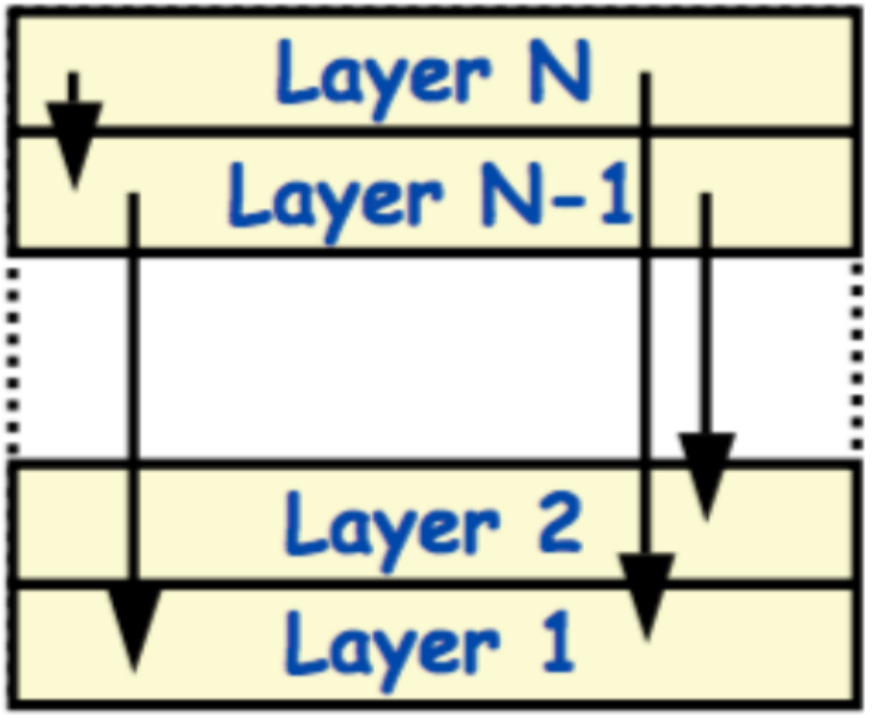

User interface
User interface management
Authentication and authorization
Core business logic/application functionality
System utilities
System support (OS, database etc.)

Uses

- Operating systems
- network communication protocols

Open Layered Architecture

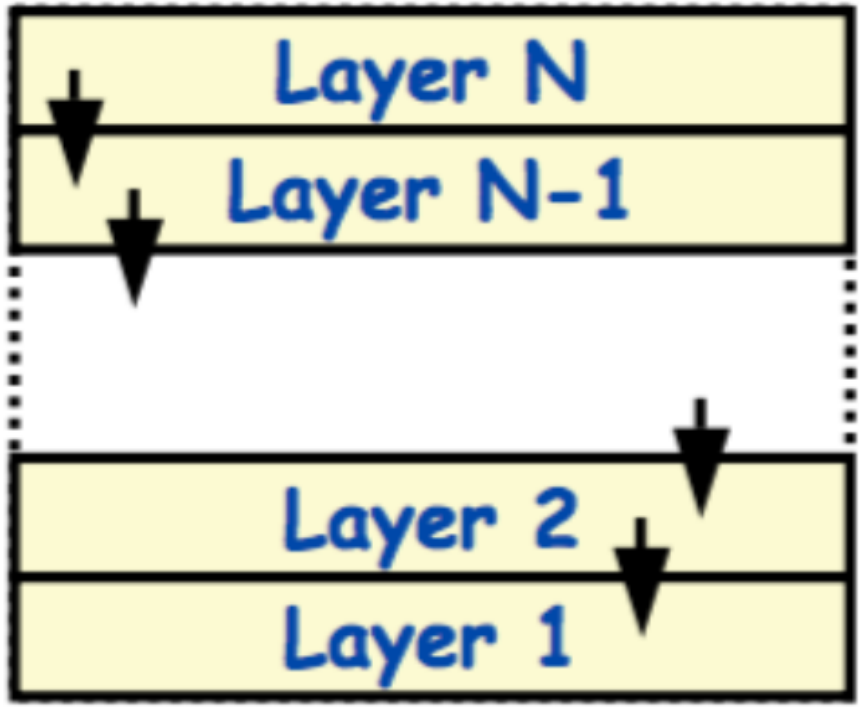
- A layer can use services from **any** lower layer
- More compact code
- Increases dependency between layers



Closed Layered Architecture

- Each layer only uses services of the layer immediately below
- Minimizes dependencies between layers
- Reduces the impact of change

> https://wiki.c2.com/?FourLayerArchitecture



Number of layers

2-layers

- Applications layer
- Database layer

3-layers

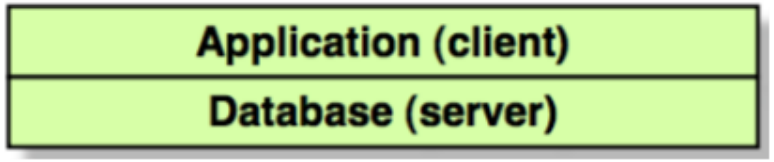
- Separate logic layer

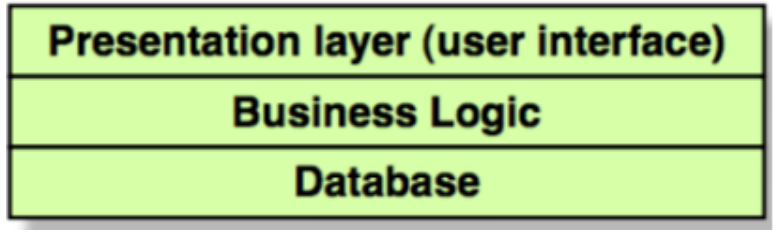
4-layers

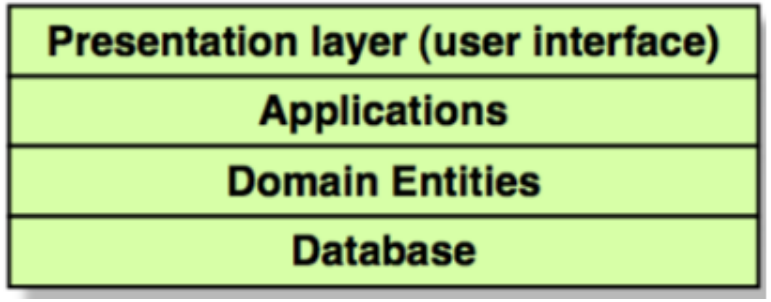
- Separate applications from domain entities

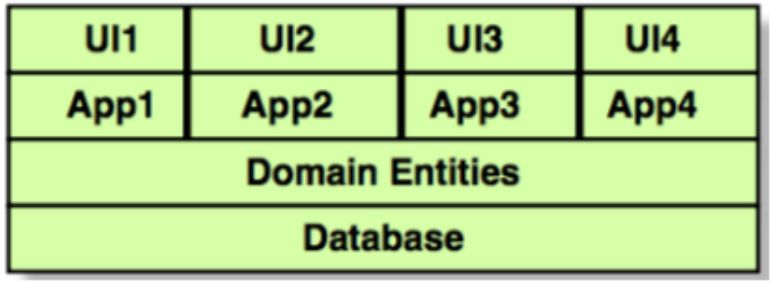
Partitioned 4-layers

- Identify separate applications









Repository architecture

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.

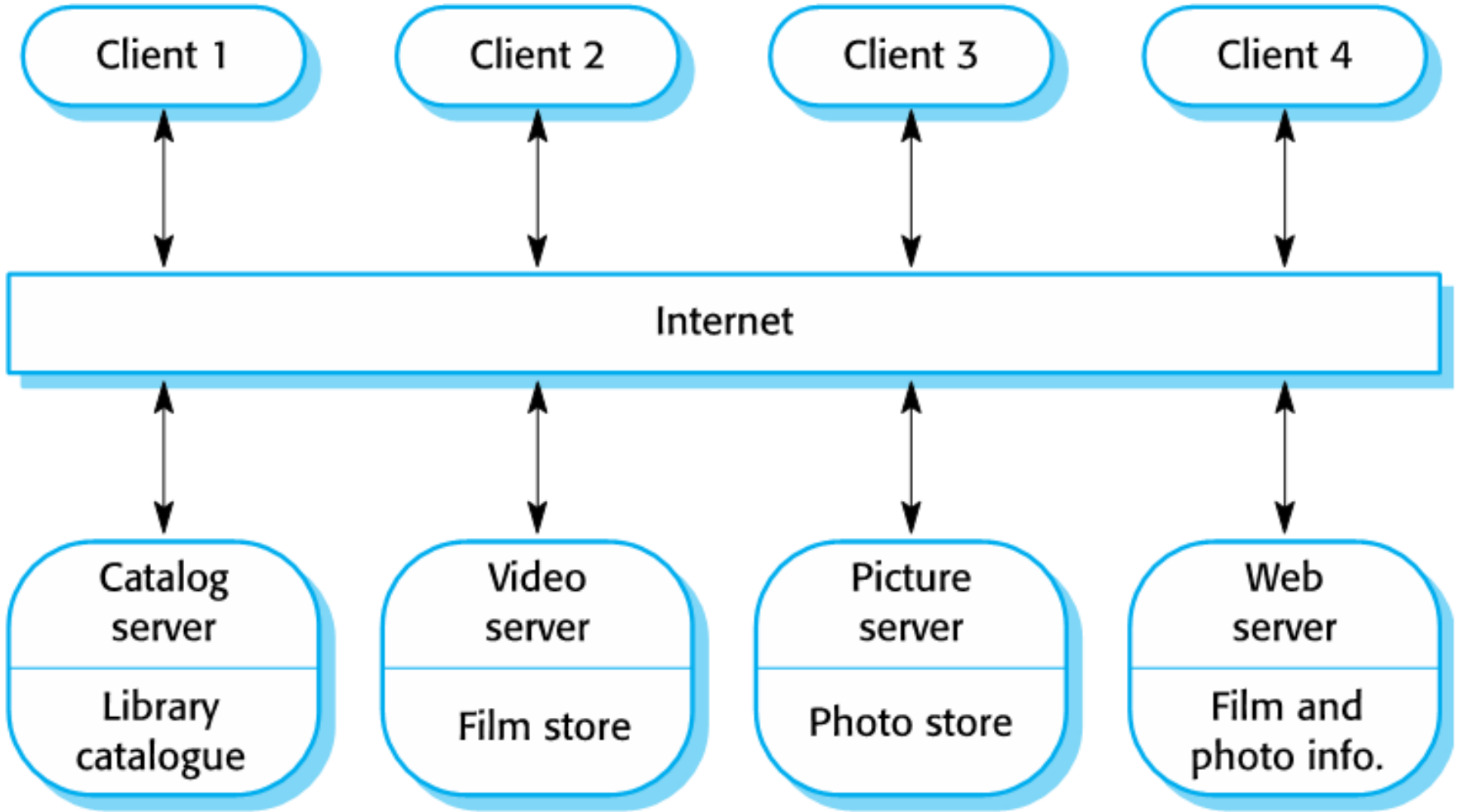
Name	Repository
Description	All data in a system is managed in a central repository that is accessible to all system components. Components do not interact directly, only through the repository.
Example	Figure 6.9 is an example of an IDE where the components use a repository of system design information. 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.

Client-server architecture

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.

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.


Client 1
Client 2
Client 3
Client 4
Internet
Catalog server
Library catalogue
Video server
Film store
Picture server
Photo store
Web server
Film and photo info.

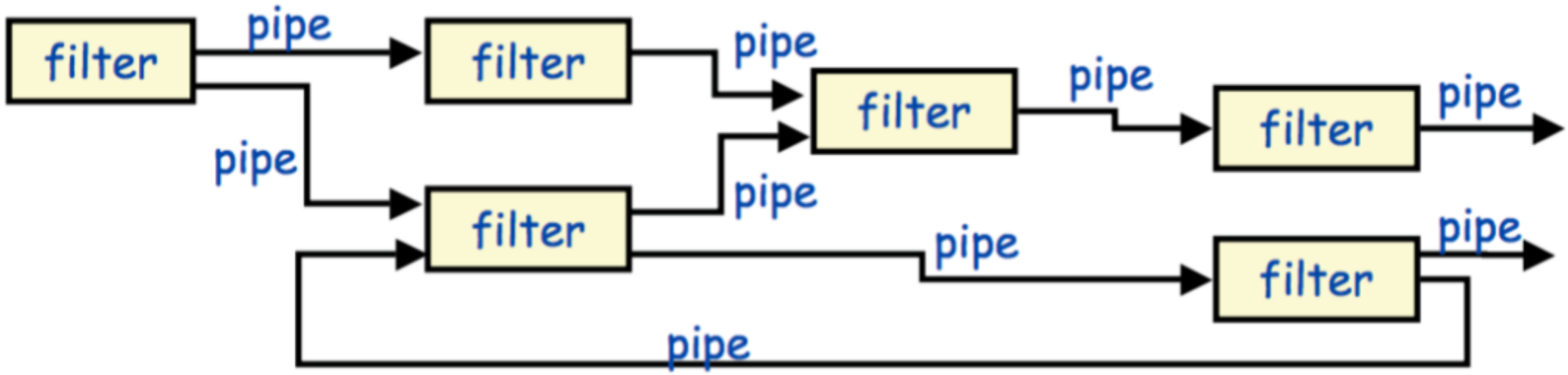
Pipe and filter architecture

Pipe and filter architecture

- Apply a function to an 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.
- Filters can be implemented in parallel

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.

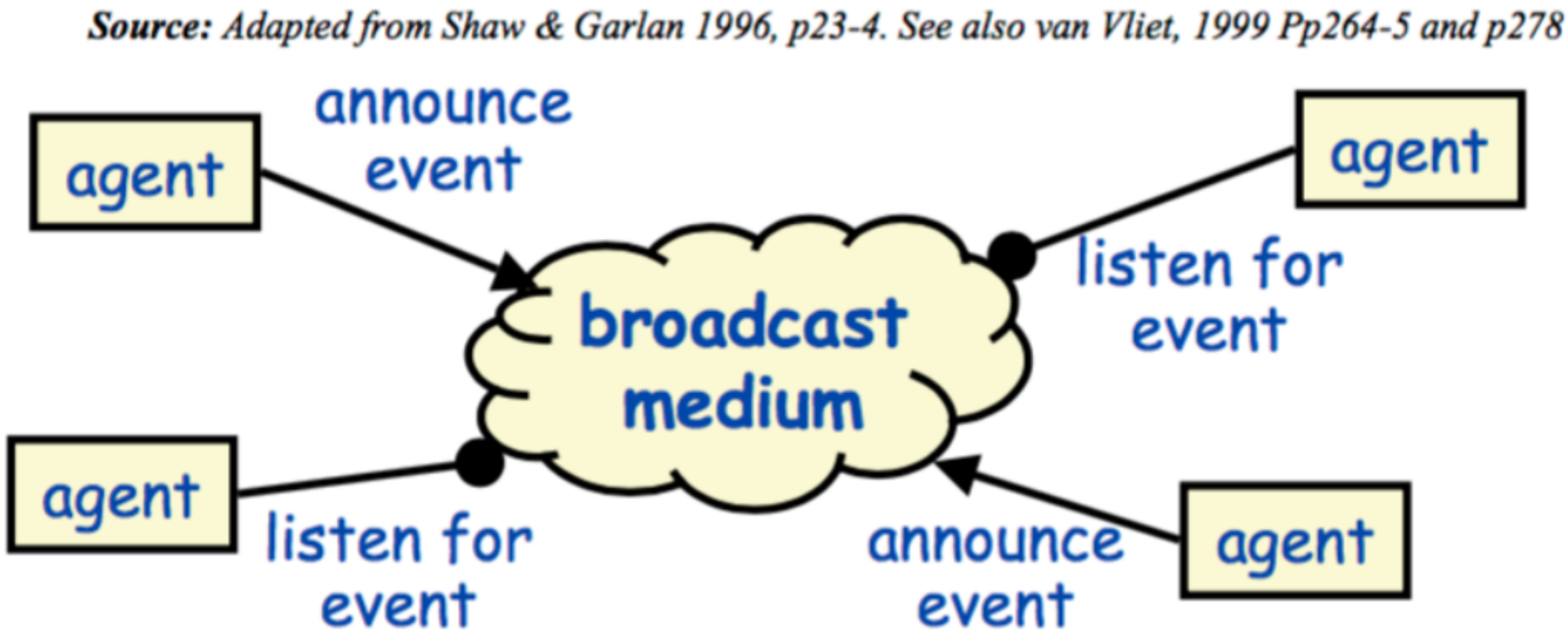
Source: Adapted from Shaw & Garlan 1996, p21-2. See also van Vliet, 1999 Pp266-7 and p279



Event Based Architecture

Event Based Architecture

Components react to externally generated events and communicate with other components through events



Event Based Architecture

- Listeners of events subscribe to events
- Announcers of events don’t need to know who will handle the event
- Adding new agents is easy; evolution made easy

Event Based Architecture: Use

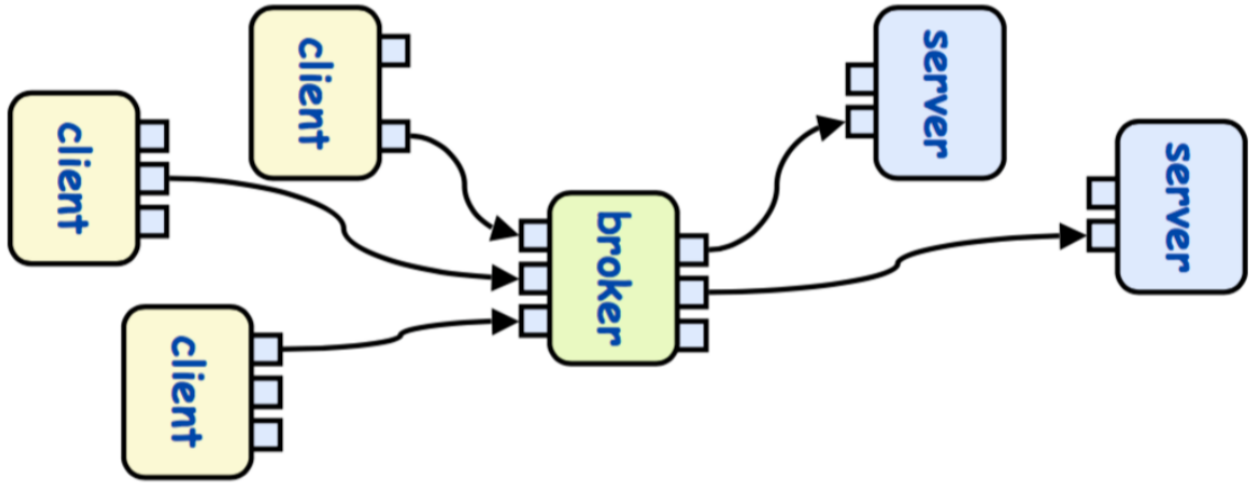
- Debugging systems: listen for breakpoints
- Graphical User Interfaces: listen for user inputs
- Database management systems: data integrity checks

Event Based: Benefits and Drawback

- Loosely coupled and well distributed
- Workflow is usually not obvious
- Support broadcast communication in a distributed context
- Performance may be a concern

Object Broker Architecture

- Add a broker between the clients and the servers
- Clients don't need to know which server they are using
- Can have multiple brokers and multiple servers



Many different types -- Remote procedure call

- CORBA https://www.corba.com/
- D-bus https://www.freedesktop.org/wiki/Software/dbus/

Many different types -- Message Queueing Systems

- IBM MQSeries https://www.ibm.com/products/mq
- MQTT https://mqtt.org/
- Jakarta Messaging API (Java Message Service) https://jakarta.ee/specifications/messaging/2.0/apidocs

Many different types -- Distributed Object Schemes

- Remote Method Invocation https://www.oracle.com/java/technologies/javase/remote-method-invocation-home.html

Many different types -- Wired XML Schemes

- SOAP https://www.w3.org/TR/2000/NOTE-SOAP-20000508/
- XML-RPChttp://xmlrpc.com/

Many different types -- COM and DCOM

- MS-DCOM https://docs.microsoft.com/en-us/openspecs/windows_protocols/ms-dcom/
- OpenCOM

Questions?

- Tsui Frank, Karam Orlando, Bernal Barbara. Essentials of software engineering
- Sommerville Ian. 2010. Software Engineering, 10th Edition, Pearson
- Grace Lewis, Carnegie Mellon University, SEI
- https://developer.ibm.com/articles/an-introduction-to-uml/
- https://docs.microsoft.com/en-us/azure/architecture/serverless-quest/serverless-overview