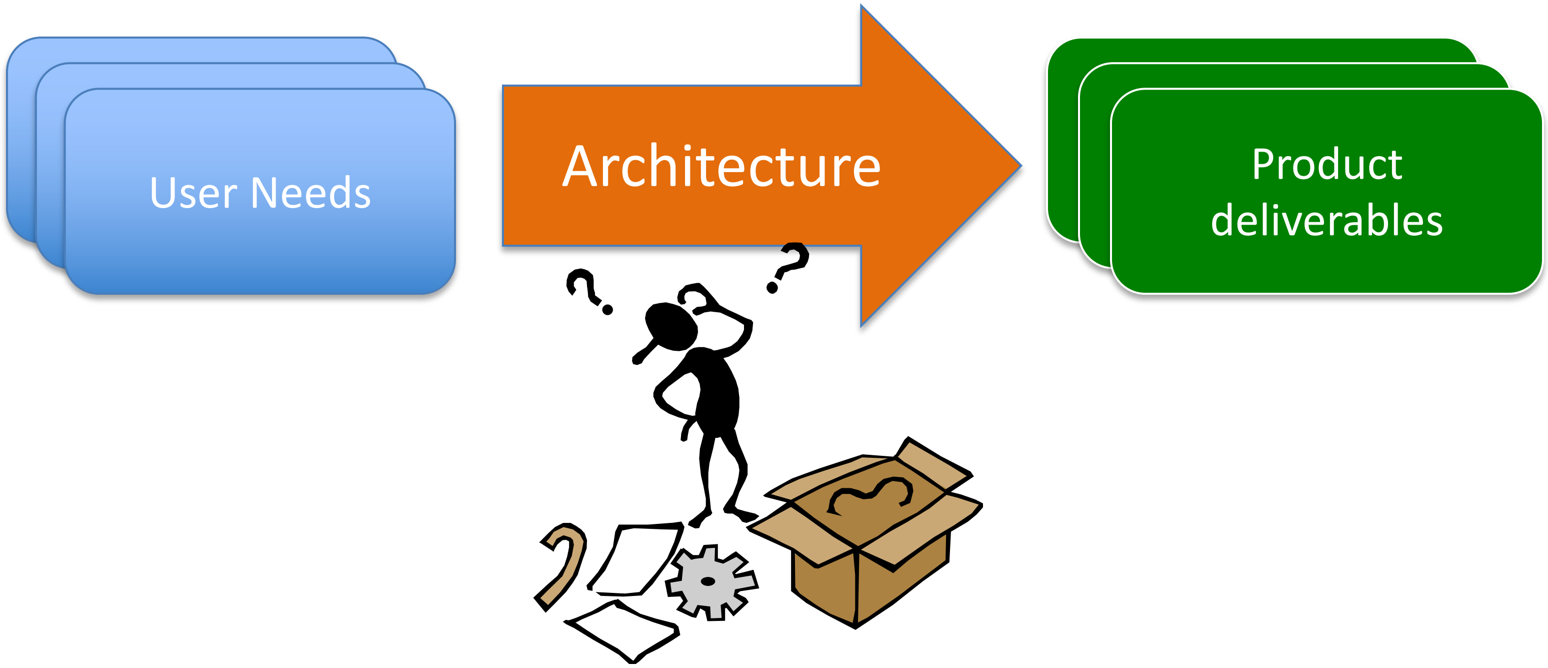
SSW-540: Fundamentals of Software Engineering

Software Architecture Styles

Roberta (Robbie) Cohen, Ph.D.
Industry Professor
School of Systems and Enterprises

Software architecture & design

- Architecture represents a critical link between requirements and implementation.
  - "There is a significant overlap between the processes of requirements engineering and architectural design."



Architectural design

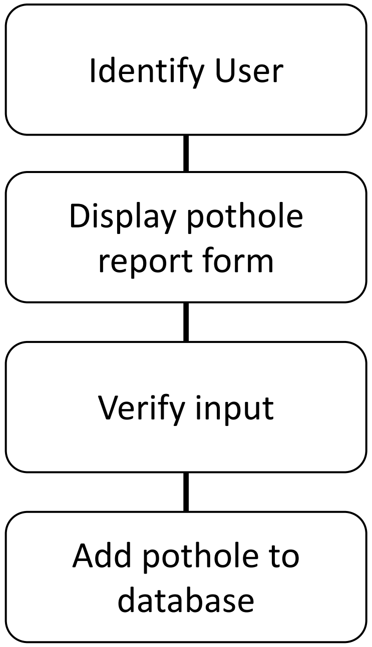
- Architectural design is
  - how a software system should be organized; its overall structure and the interfaces it supports
    - This includes its context, module (& class) structure and interfaces
  - the link between requirements engineering and detailed design
    - This might be represented with use cases, sequence, activity and state diagrams showing how the system delivers its functionality
- An early stage of agile processes often includes design of an overall systems architecture
  - Redoing a system's architecture is usually costly because it affects so many components in the system
  - Providing some organization to the development, up front, can save considerable re-work at later stages

Architectural Abstraction

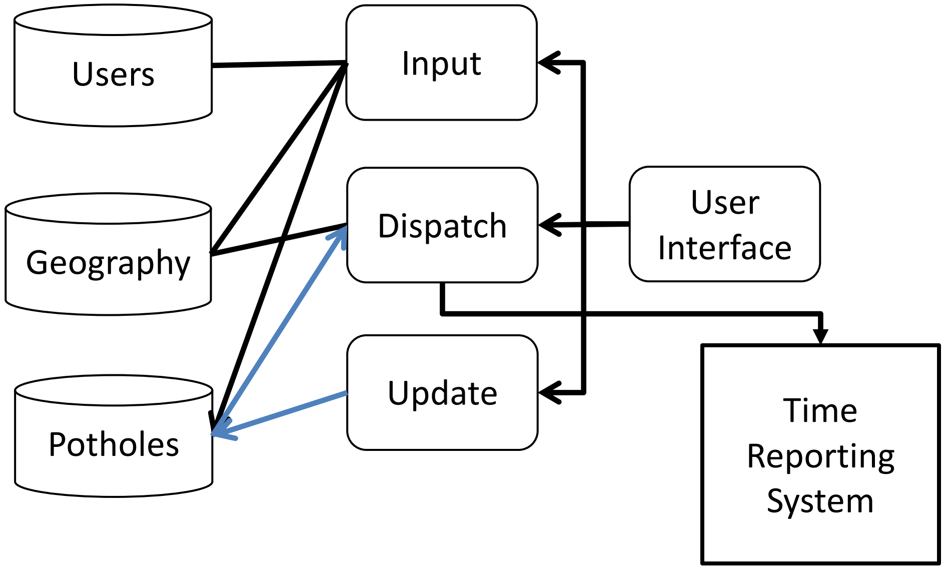
In the small… In the large…

Pothole tracking software

Input Program



Software System

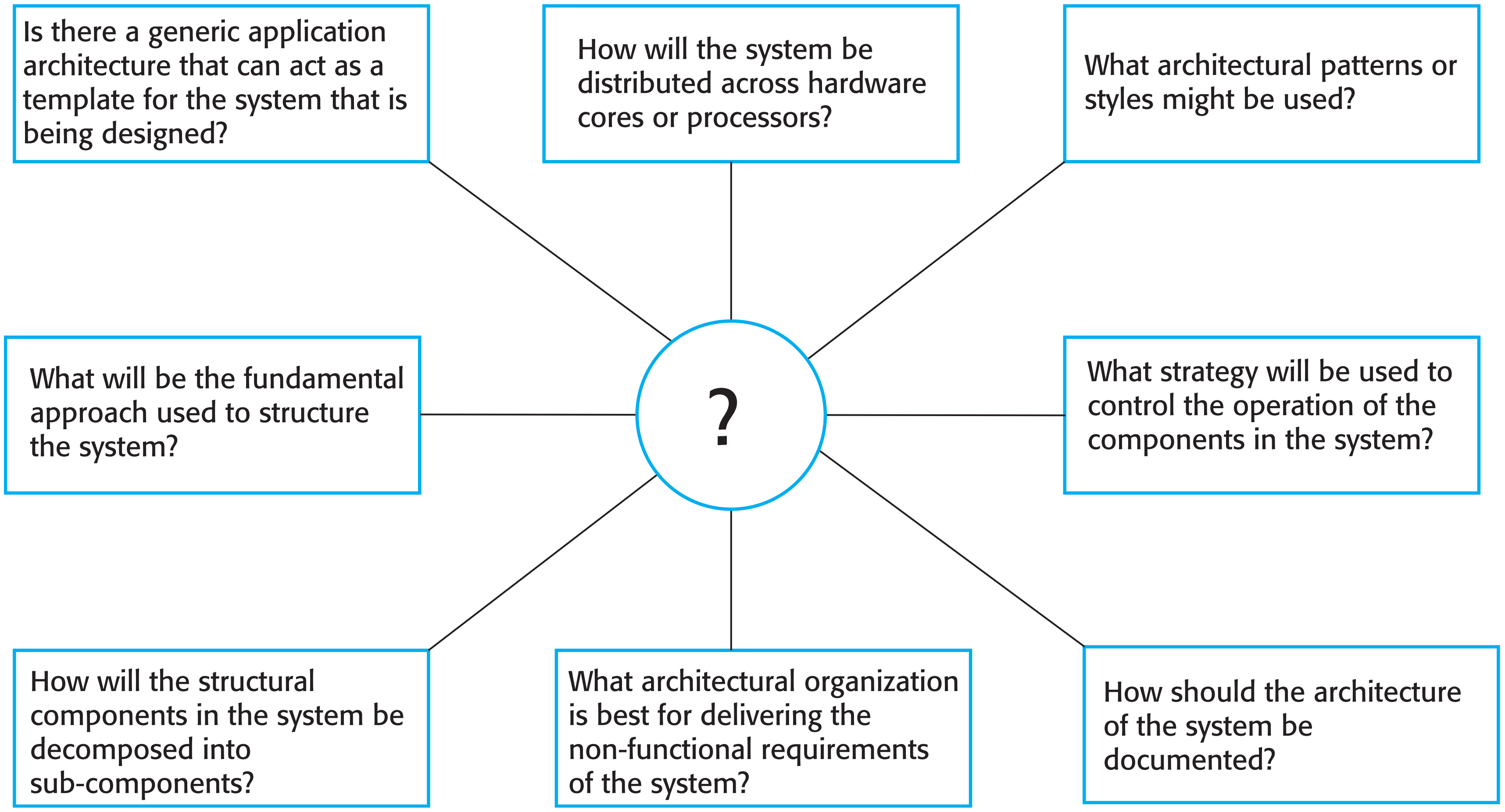


To create an architecture…

- One models what the system will do
- One identifies the system’s interfaces and **constraints**
- One architects the system to satisfy **NFR’s** like performance, security, correctness, safety, availability, usability and maintainability!
- To do this, one must identify software **components** and **relationships**, based on customer’s requirements
- These form the content used for the software system models and for the primary architecture of the software system

Architectural design decisions

Decisions driven by primary use cases, measurable quality attributes, and constraints.


Is there a generic application architecture that can act as a template for the system that is being designed?
How will the system be distributed across hardware cores or processors?
What architectural patterns or styles might be used?
What will be the fundamental approach used to structure the system?
?
What strategy will be used to control the operation of the components in the system?
How will the structural components in the system be decomposed into sub-components?
What architectural organization is best for delivering the non-functional requirements of the system?
How should the architecture of the system be documented?

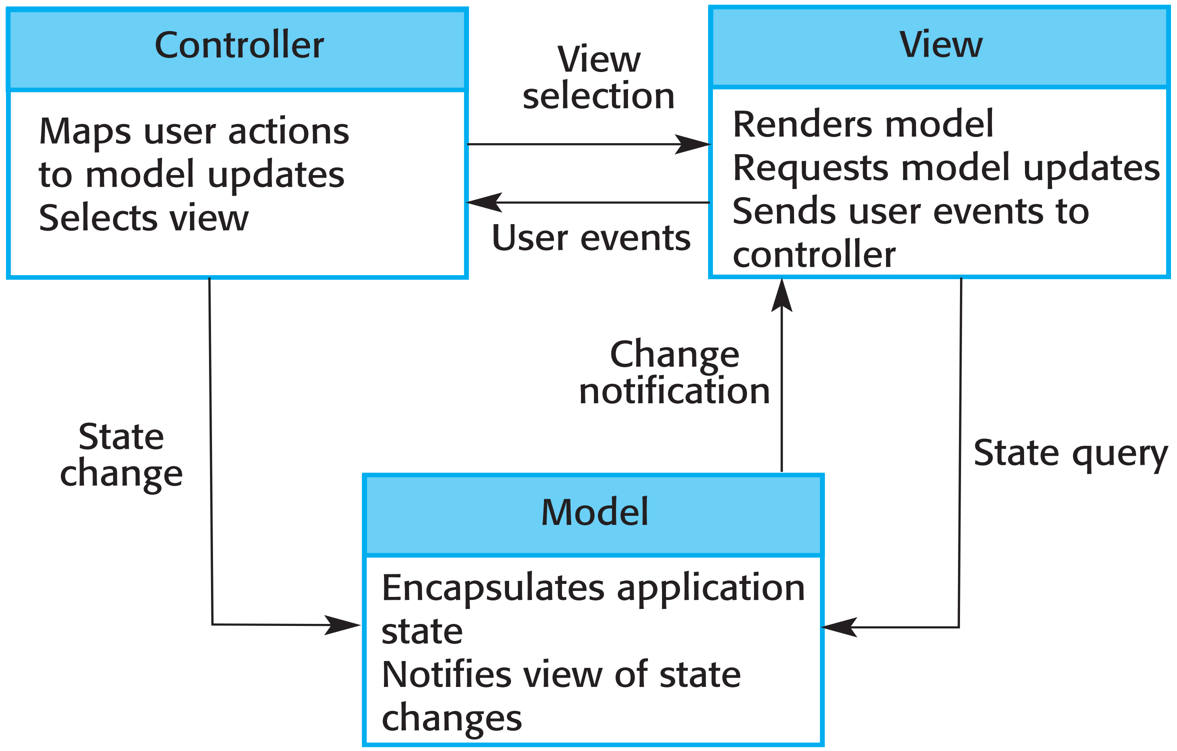
Simplifying design activities

- When we design or architect a system, we never know enough!
  - Requirements keep changing (so the final needs aren’t known)
  - Until we code, the impact of constraints is not yet felt (or known)
  - The system’s intended environment might not be known
- Using proven architectural styles and patterns helps
  - Directs us to re-use known means for achieving our aims
  - Frees us to focus on the unknowns
- Patterns and styles are *tools* for the architect and designer

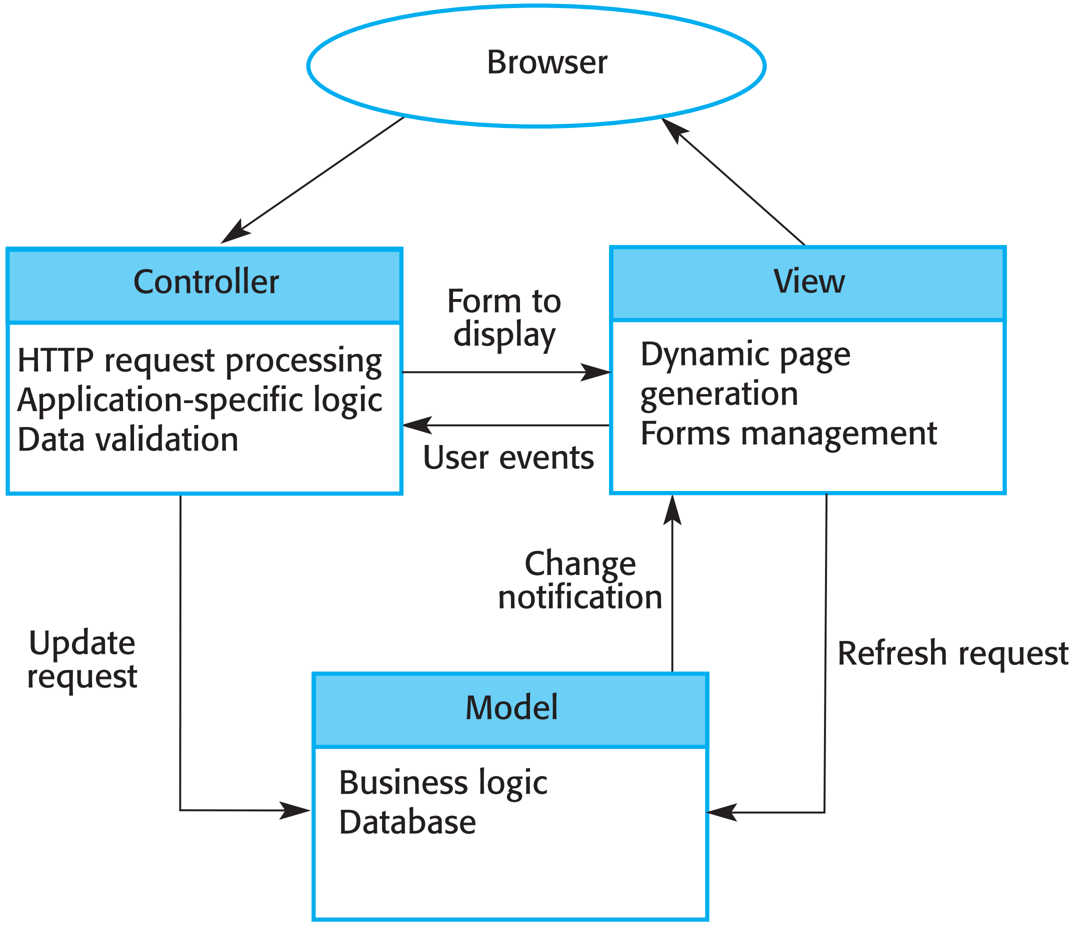
Architectural patterns

- ✧ Patterns are a means of representing, sharing and reusing knowledge. They exist at varying levels of abstraction
- ✧ An architectural pattern is a stylized description of good design practice, which has been tried and tested in different environments.
- ✧ Patterns appear "obvious" because they so often appear!
- ✧ Patterns include information about when they are and when they are not useful.
- ✧ Patterns may be represented using tabular and graphical descriptions.

One such pattern: the Model-View-Controller



Web application architecture using the MVC pattern



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 earlier figure.
Example	Previous 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.

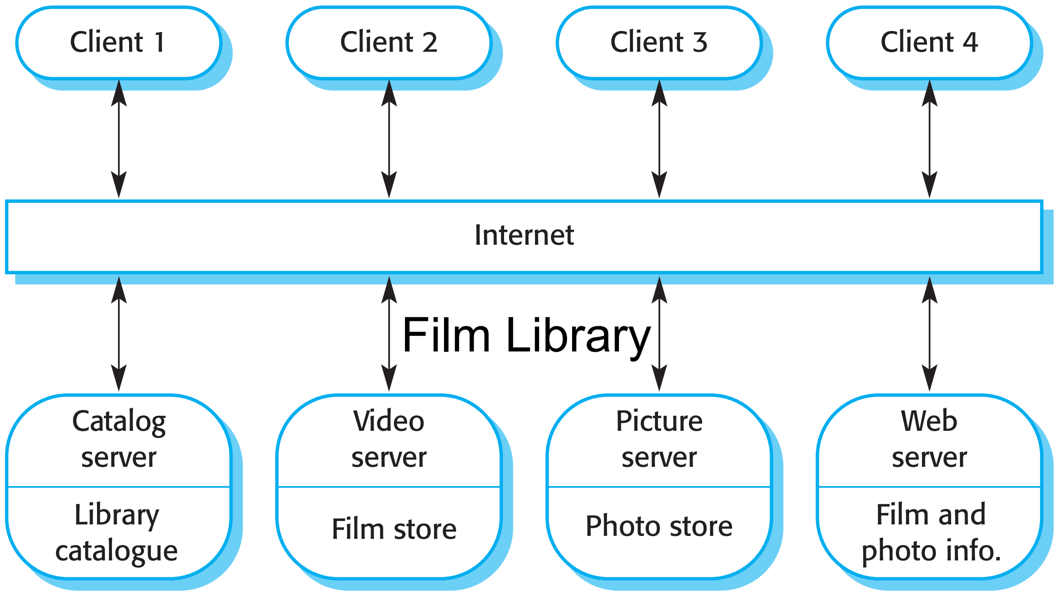
Value of the MVC

- ✧ It removes the viewing dependency from the model.
- ✧ It keeps the view reusable without modification even when the controller changes.
- ✧ The controller interprets users' actions and provides data to the view.
- ✧ The view displays the model's data (and sends user actions to the controller)
- ✧ The model is the data and does nothing with it.
- ✧ This reduces complexity, adds flexibility and reusability!

MVC's place

- ✧ Model-View-Controller is now a common pattern for web-based exchanges
  - Browser provides the view
  - Web apps provide the controller
  - Web pages are the model
- ✧ The rapidity of web app development is due in large part to MVC

Client-server architecture

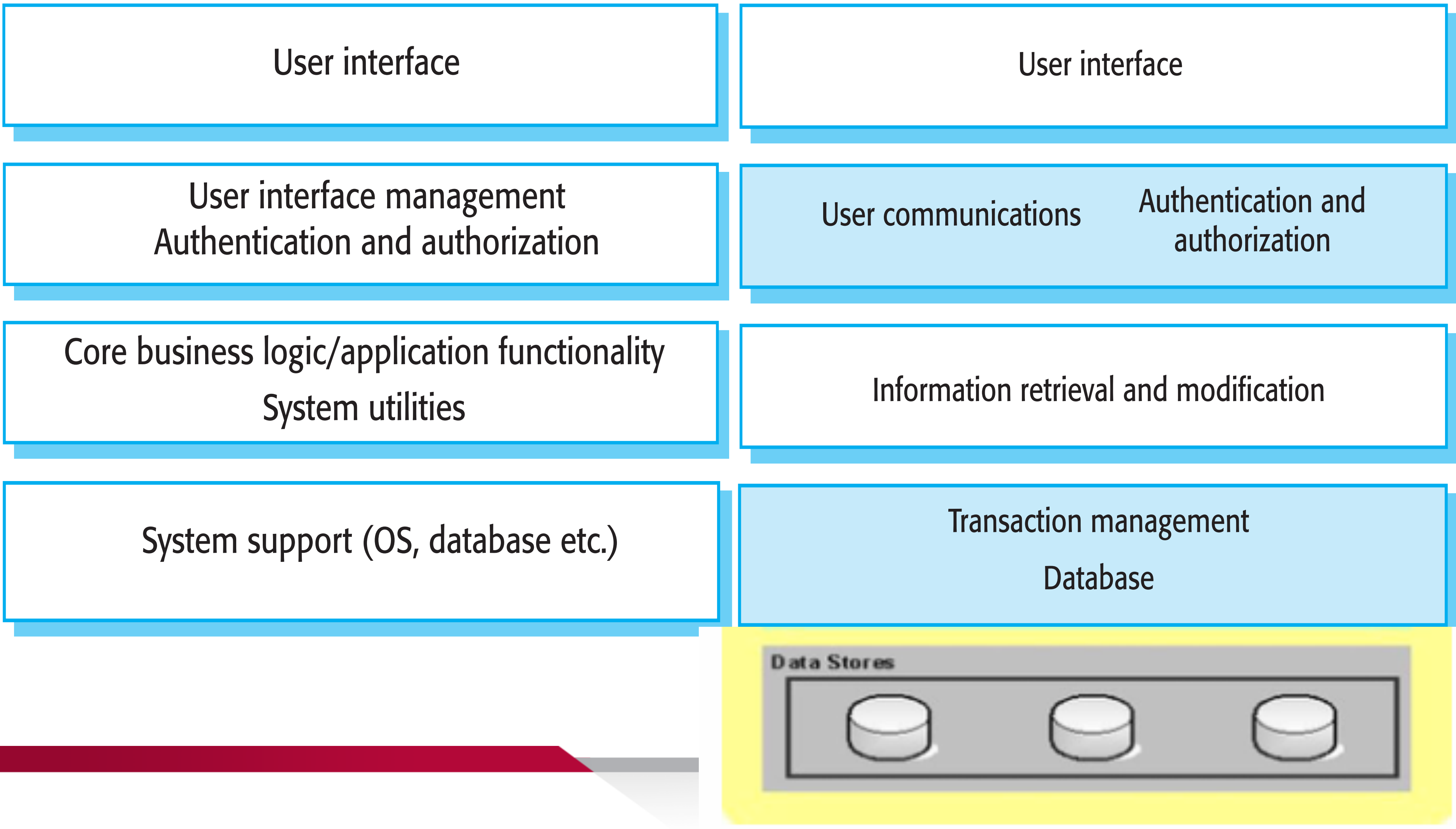


- Distributed system model which shows how data and processing is distributed across a range of components.
  - Can be implemented on a single computer.
  - Often implemented with a service on multiple redundant computers
- Includes
  - Stand-alone servers which provide specific services such as printing, data management, etc.
  - Clients which call on these services.
  - Network which allows clients to access servers.

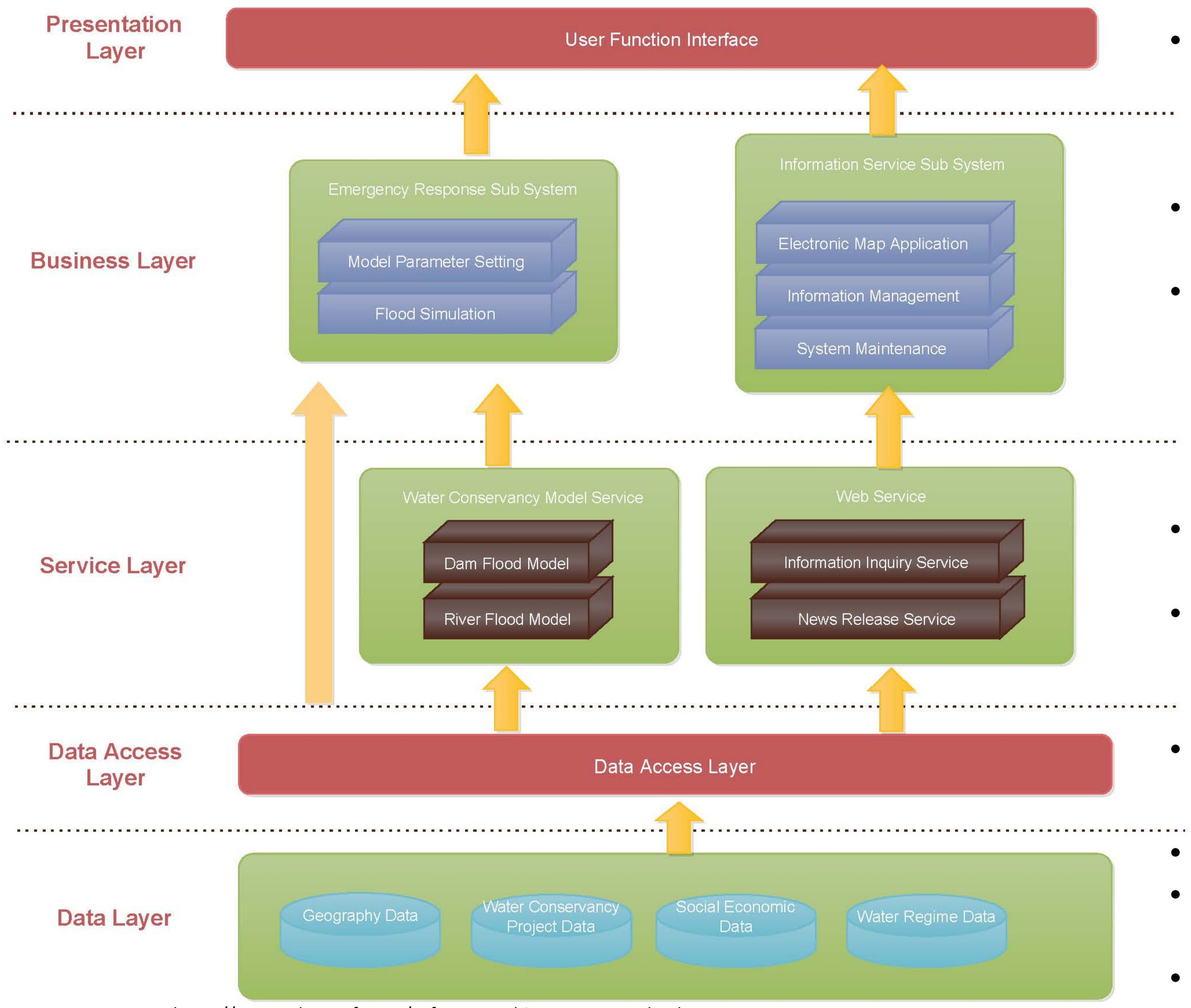
Layered architecture

- ✧ Layered architectures are used to model the interfacing of sub-systems.
- ✧ Organizes the system into a set of layers (or abstract machines) each of which provide a set of services to the layer above it.
- ✧ Supports the incremental development of sub-systems in different layers. When a layer interface changes, only the adjacent layer is affected.
- ✧ Layers can be distributed (vertically) across computer instances.
- ✧ However, it is sometimes artificial to structure systems in this way.

A generic and an information systems layered architecture


User interface
User interface management
Authentication and authorization
Core business logic/application functionality
System utilities
System support (OS, database etc.)
User interface
User communications
Authentication and authorization
Information retrieval and modification
Transaction management
Database
Data Stores

A layered water oversight system



https://www.edrawsoft.com/software-architecture-example.php

- User Function interface
- Emergency Response sub-system
- Information Service sub-system
- Water Conservancy Model service
- Web service
- Data Access Layer
- Geography data
- Water Conservancy Project data
- Social Economic data
- Water Regime data

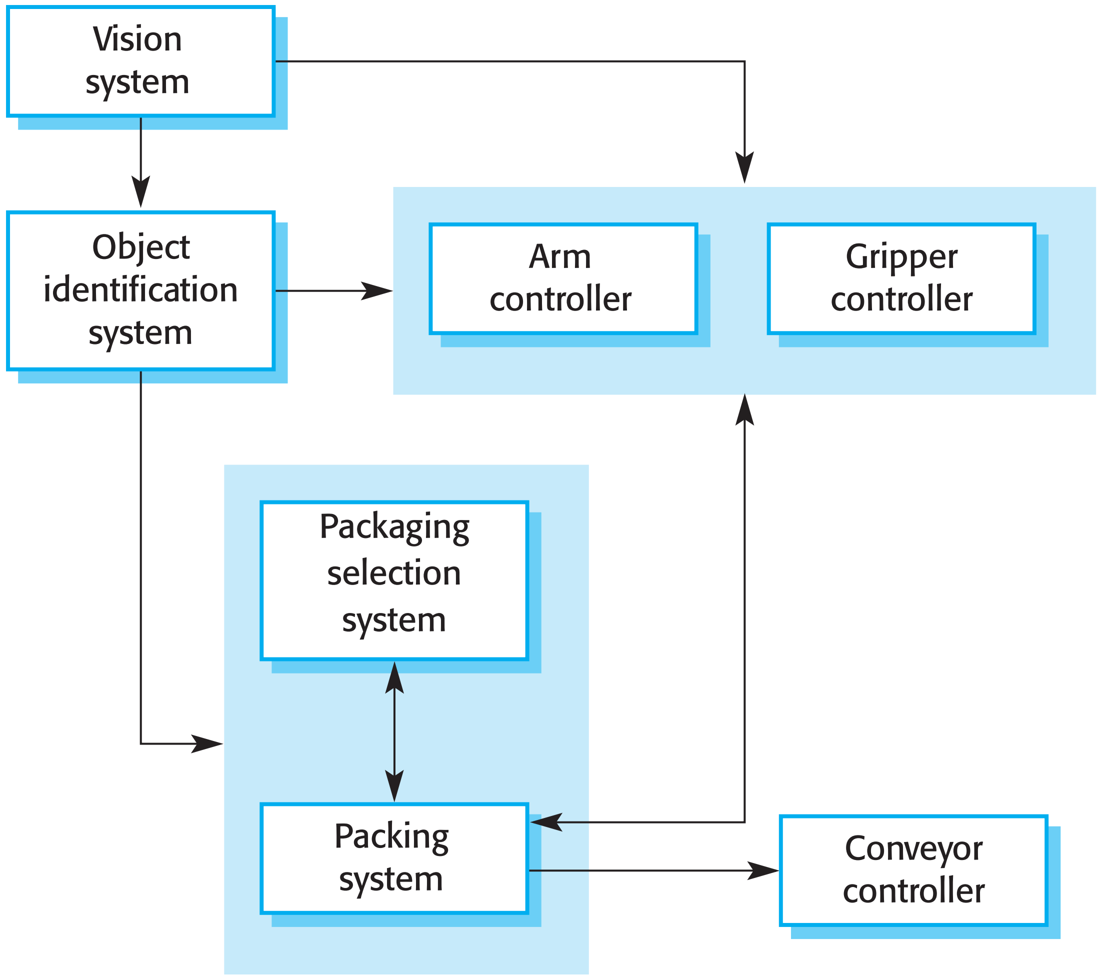
Real-time & “embedded” systems

- Software is used to control a wide range of systems from simple domestic machines, through games controllers, to vehicles and entire manufacturing plants.
- Such software must react to events generated by the hardware and, often, issue control signals in response to these events.
- The software for these systems is *embedded* in system hardware, often in read-only memory, and usually responds, in real time, to events from the system’s environment.
- Responsiveness in real-time is the critical difference between embedded systems and other software systems, such as information systems, web-based systems or personal software systems.

Characteristics of *embedded* software

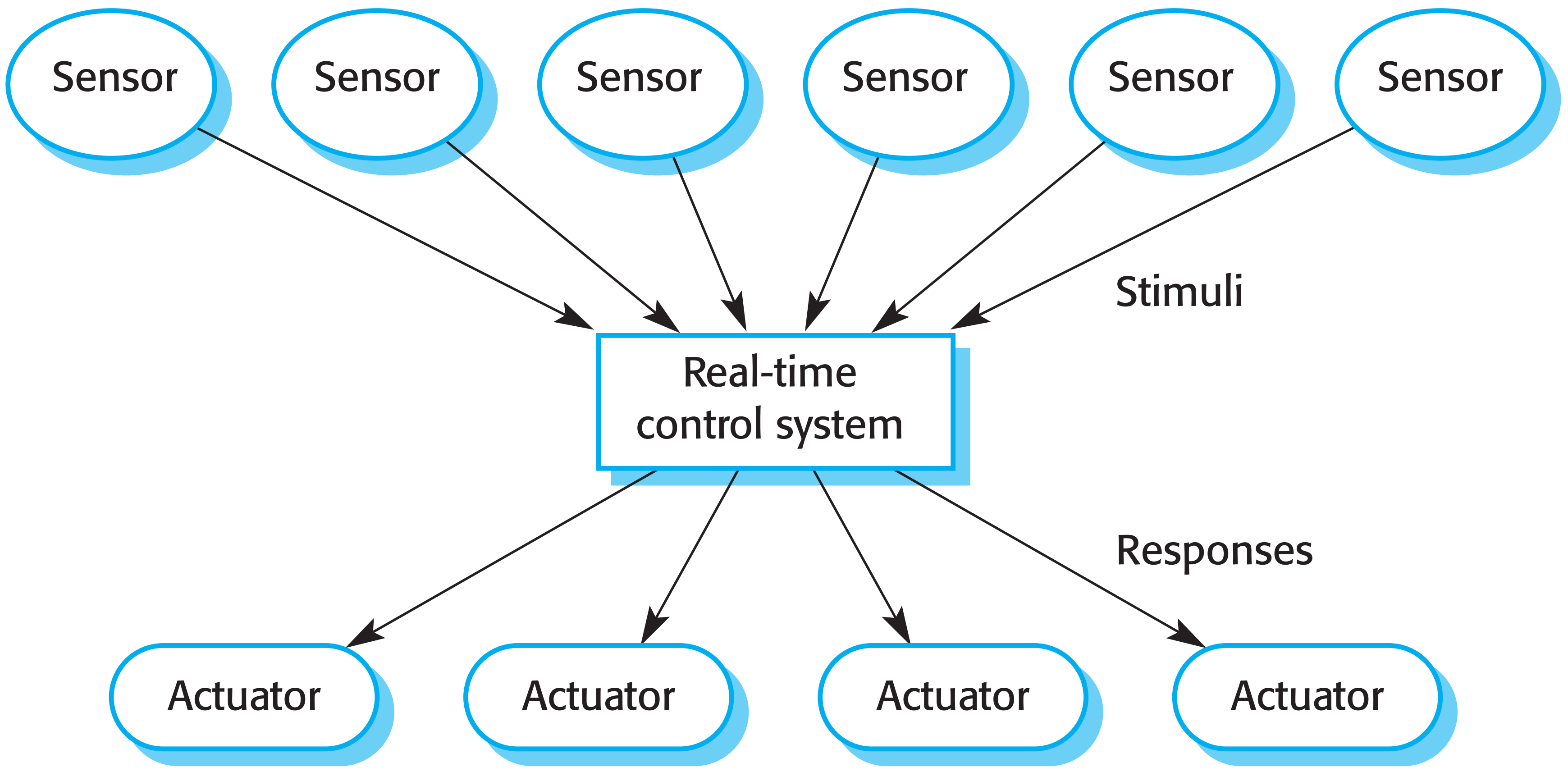
- ✧ Embedded systems generally run continuously and do not terminate.
- ✧ Interactions with the system's environment are unpredictable.
- ✧ There may be physical limitations that affect the design of a system.
- ✧ Direct hardware interaction may be necessary.
- ✧ Issues of safety and reliability may dominate the system design.
- ✧ These systems use different architecture patterns than those typically used by information systems

Example: A packing robot control system

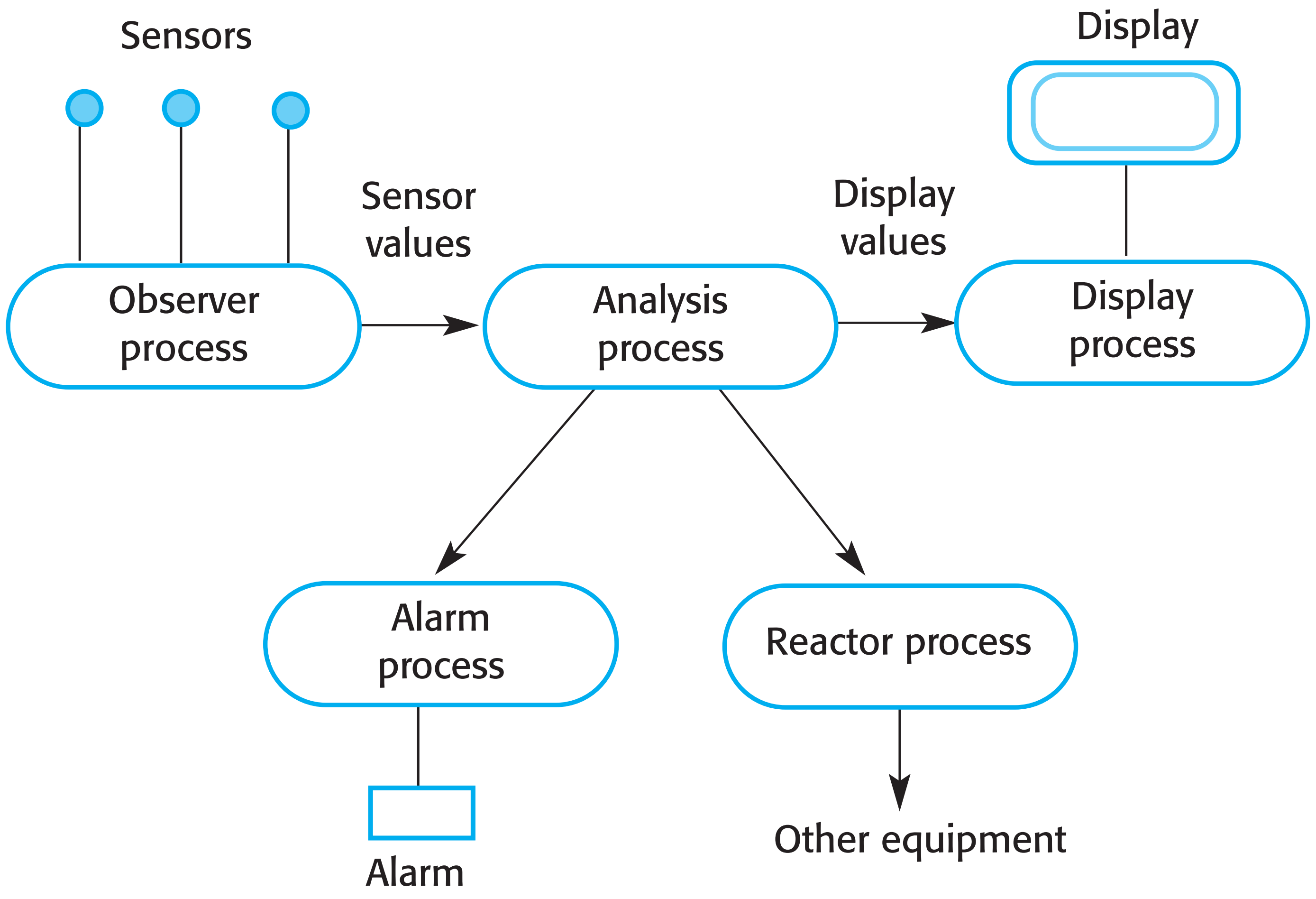


- Robots are reactive systems
  - Given a stimulus, they must react within a specified time
  - Correctness depends on both content and timing
- Reactive systems may use
  - Periodic stimuli that occur at predictable intervals
  - Aperiodic stimuli that occur at unpredictable times.

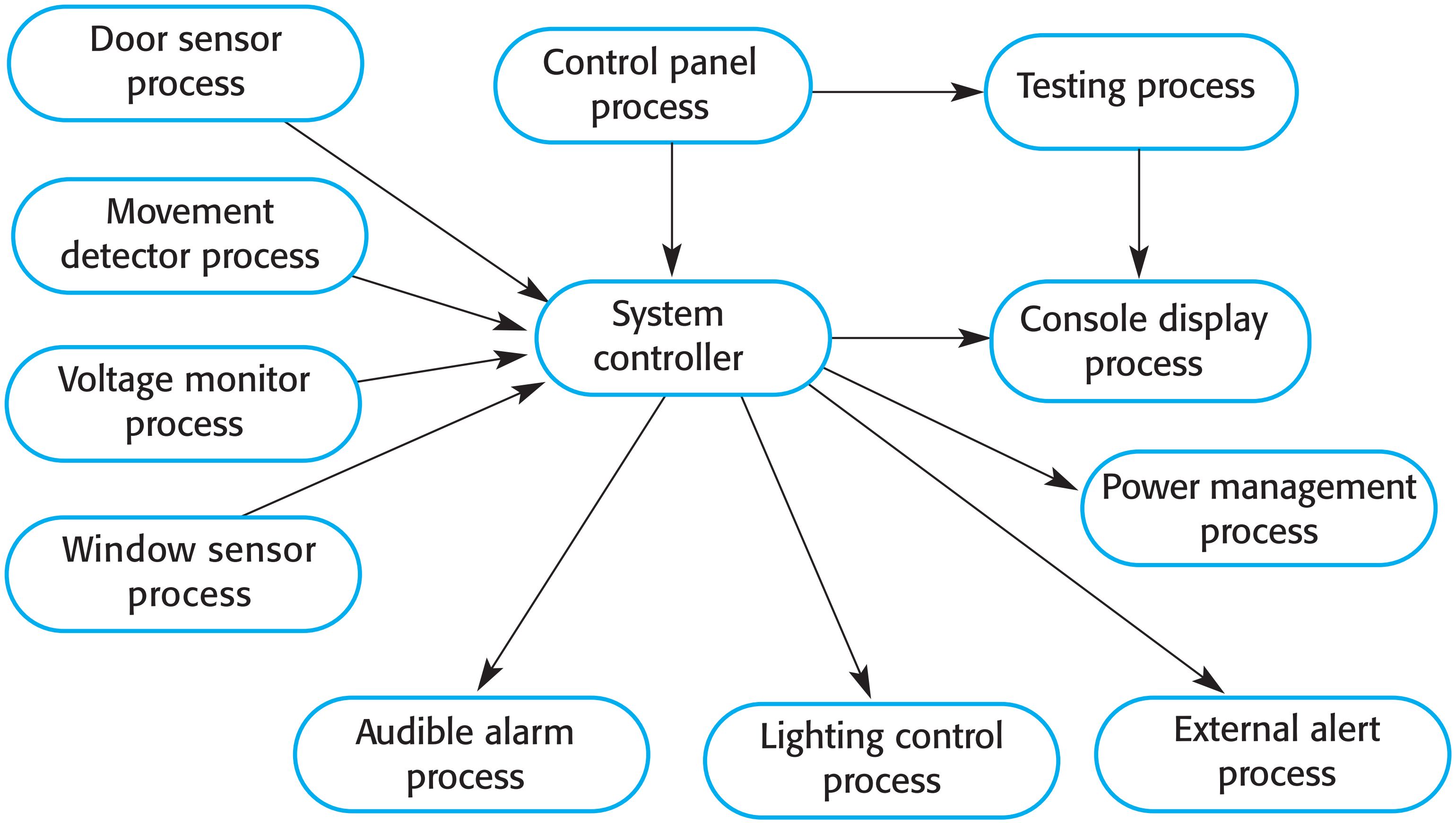
Generic model: embedded real-time system



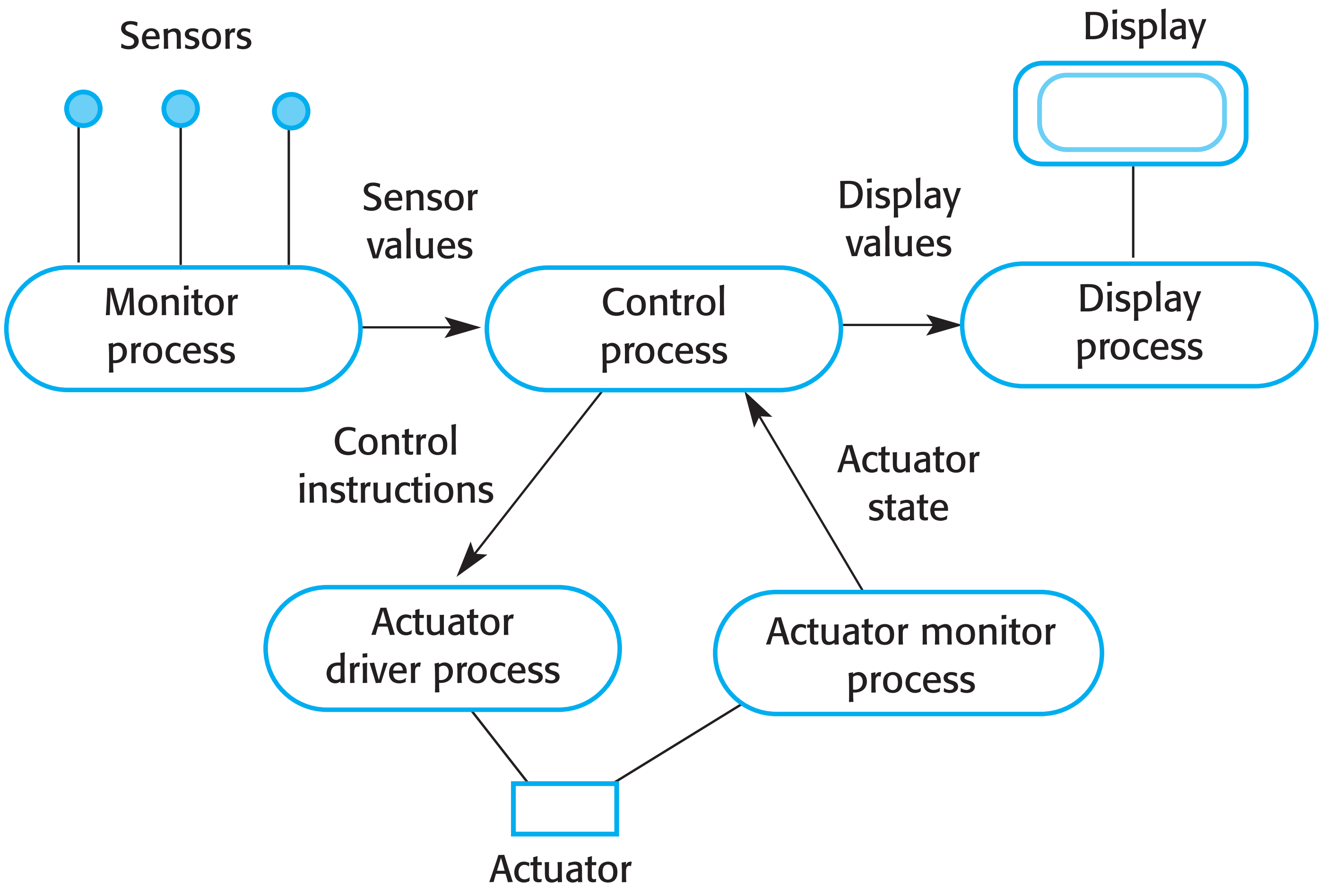
Observe and React process pattern



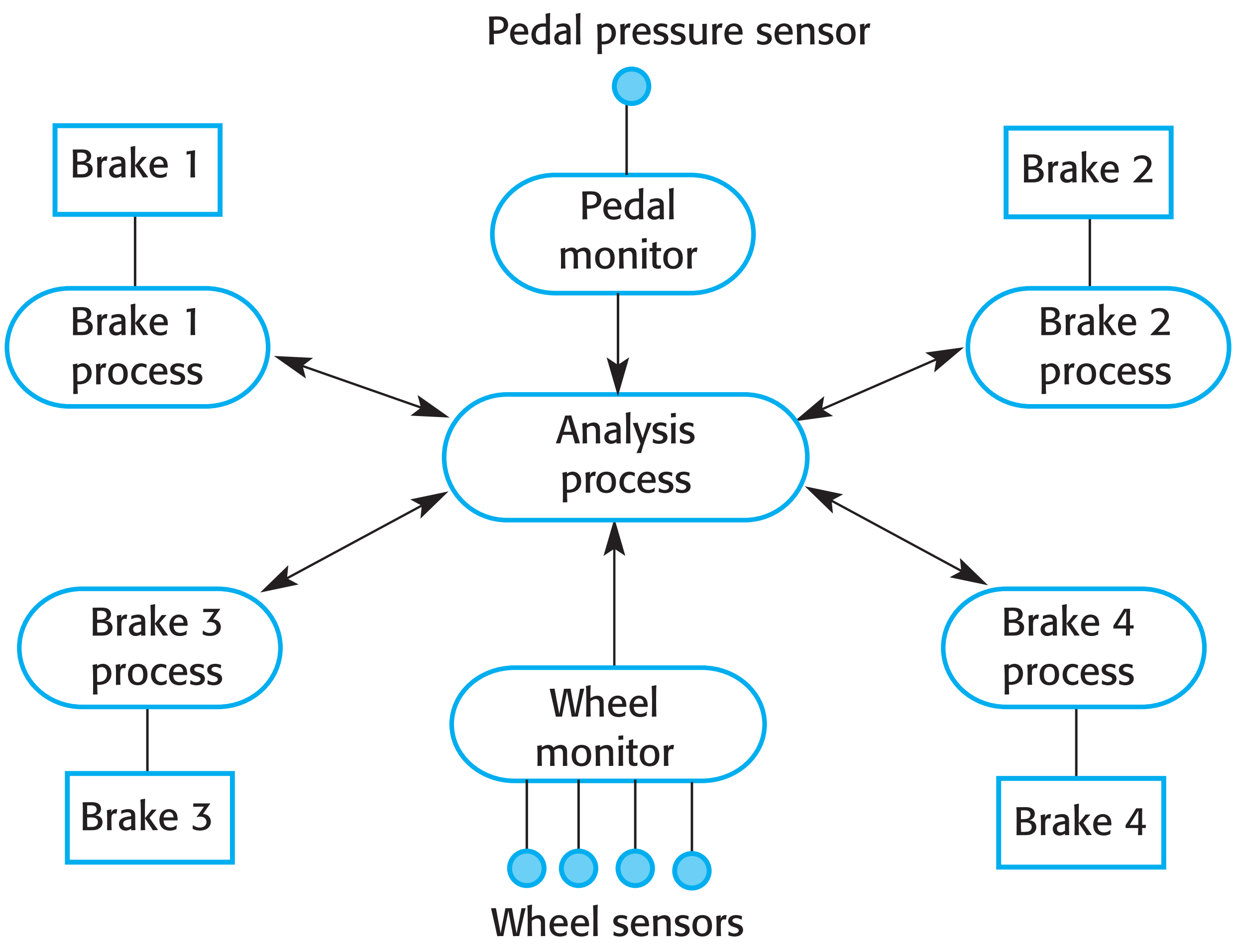
Process structure for a burglar alarm system



Environmental Control process structure



Control system architecture for an anti-skid braking system



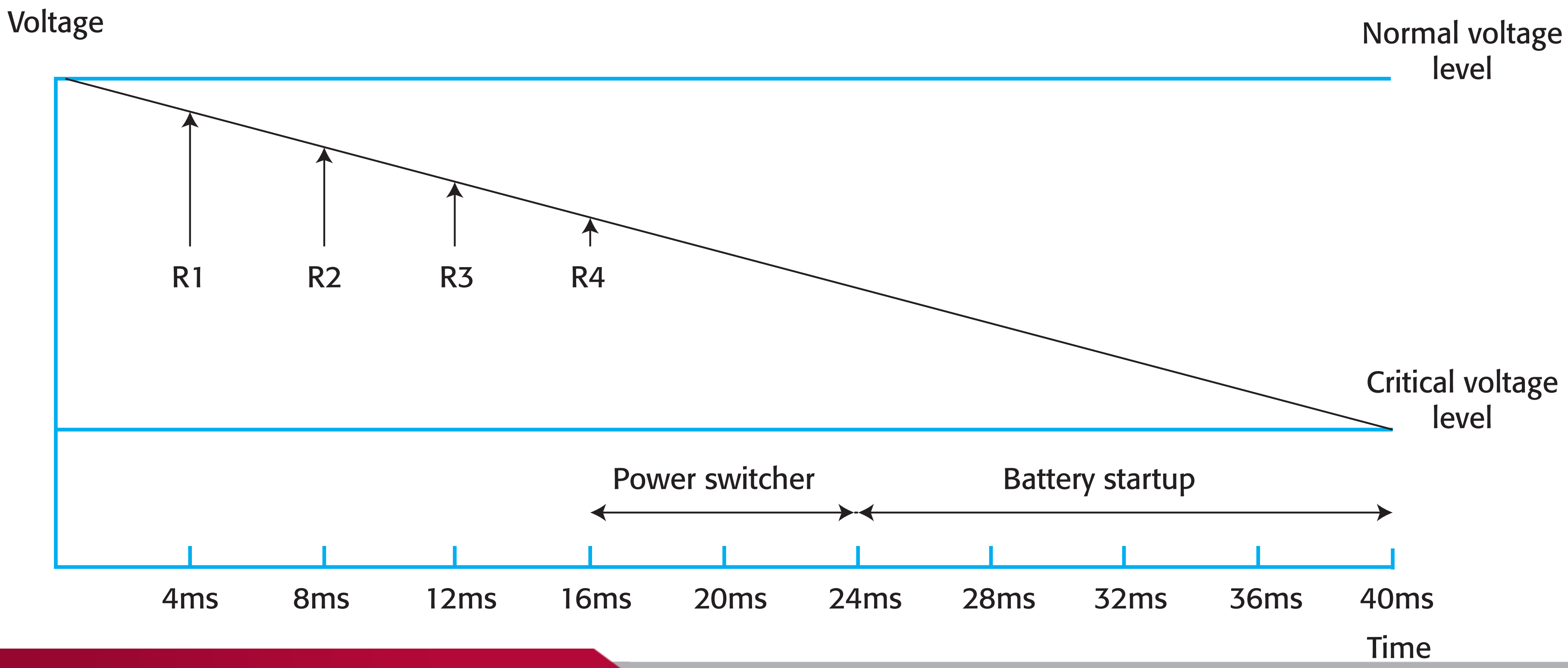
Some words on timing analysis

- The correctness of a real-time system depends not just on the correctness of its outputs but also on the time at which these outputs were produced.
- In a timing analysis, you calculate how often each process in the system must be executed to ensure that all inputs are processed and all system responses produced in a timely way.
- The results of the timing analysis are used to decide how frequently each process should execute and how these processes should be scheduled by the real-time operating system.
- The factors examined includes
  - **Deadlines**, the times by which stimuli must be processed and system responds
  - **Frequency**, how often a process must execute to surely meet its deadlines
  - **Execution time**, the time required to process a stimulus and respond

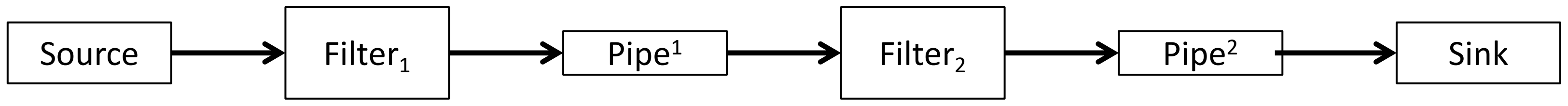
Power failure timing analysis

- It takes 3 readings of sustained significant voltage drop to recognize power failure.
- It takes 50 milliseconds (ms) for the supplied voltage to drop to a level where the equipment may be damaged.
- System has a battery backup that does not come online instantly.
- It takes 16ms from starting the backup power supply to the supply being fully operational and 8ms to switch to the battery power

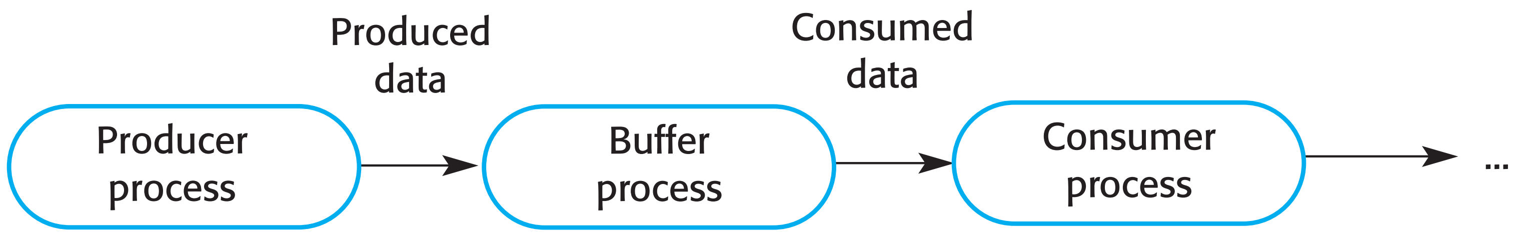
How often should one measure the voltage?



Pipe and filter architecture

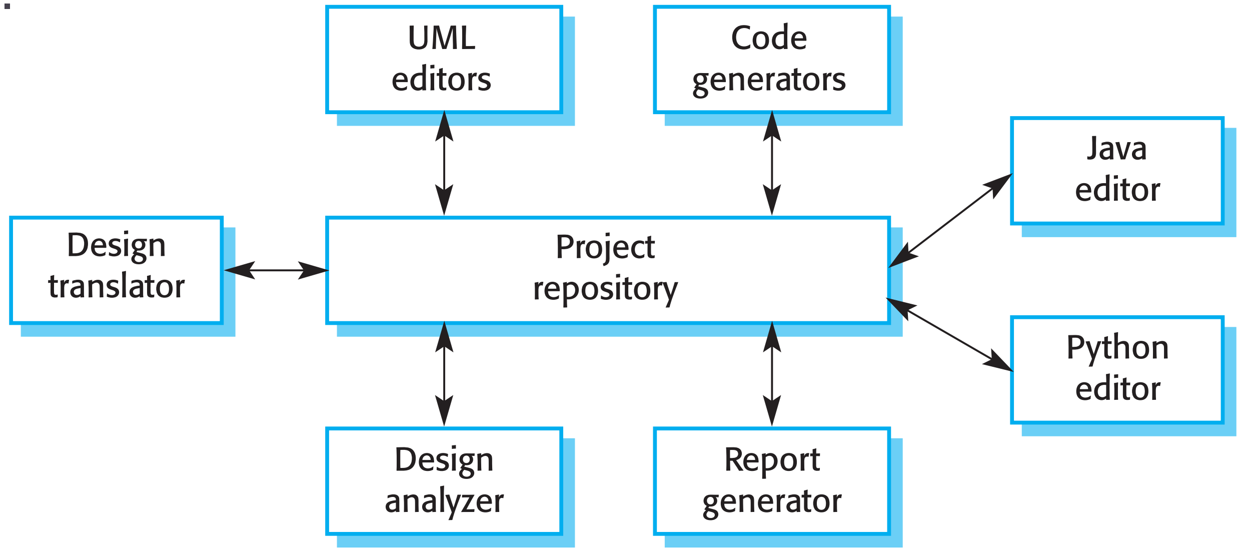


- ✧ Functional transformations process their inputs to produce outputs.
- ✧ Sometimes referred to as a pipe and filter model (as in UNIX shell).
- ✧ Variants of this approach are very common. When transformations are sequential, this is a batch sequential model which is extensively used in data processing systems.
- ✧ Not suitable for interactive systems but used in both information and embedded systems.
- ✧ Embedded variation called a **Process Pipeline pattern**



Repository architecture

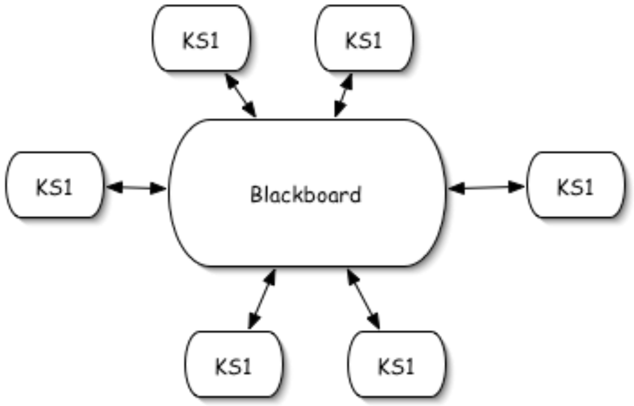
- Sub-systems must exchange data. This may be done in two ways:
  - Shared data is held in a central database or repository and may be accessed by all sub-systems;
  - Each sub-system maintains its own database and passes data explicitly to other sub-systems.
- When large amounts of data are to be shared, the repository model of sharing is most commonly used since this is an efficient data sharing mechanism.

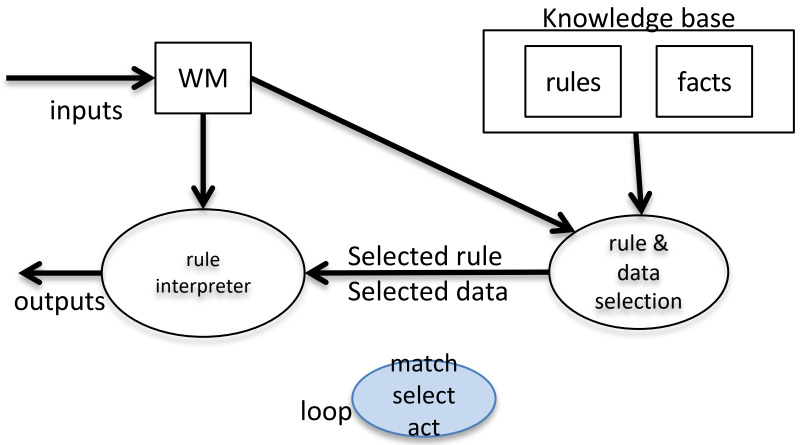


A repository architecture for an Integrated Development Environment (IDE)

Repository variant: rule-based architectures

- ✧ Artificial intelligence and language translation systems often use variants of the repository architecture
- ✧ Knowledge sources and/or knowledge bases are used to determine how the program will react to input (the state of the data determine the reaction)





How patterns play out

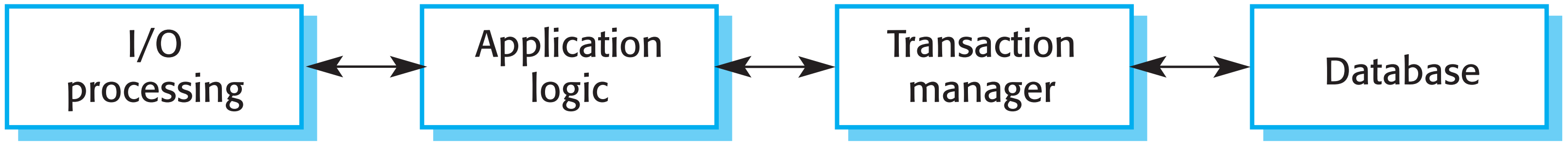
- ✧ Different kinds of software applications lend themselves to different design patterns.
  - Application systems are designed to meet an organizational need.
  - Since business domains have much in common, their application systems also tend to have a common architecture that reflects the application 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.
  - Uses of application architectures
    - As a starting point for architectural design.
    - As a design checklist.
    - As a way of organizing the work of the development team.
    - As a means of assessing components for reuse.
    - As a vocabulary for talking about application types.

Examples of application types

- Data processing applications, e.g., a billing system
  - Data driven applications that process data in batches without explicit user intervention during the processing.
- Transaction processing applications, e.g., reservation system
  - Data-centered applications that process user requests and update information in a system database.
- Event processing systems, e.g., an alarm system
  - Applications where system actions depend on interpreting events from the system's environment.
- Language processing systems, e.g., compiler or interpreter
  - Applications where the users' intentions are specified in a formal language that is processed and interpreted by the system.

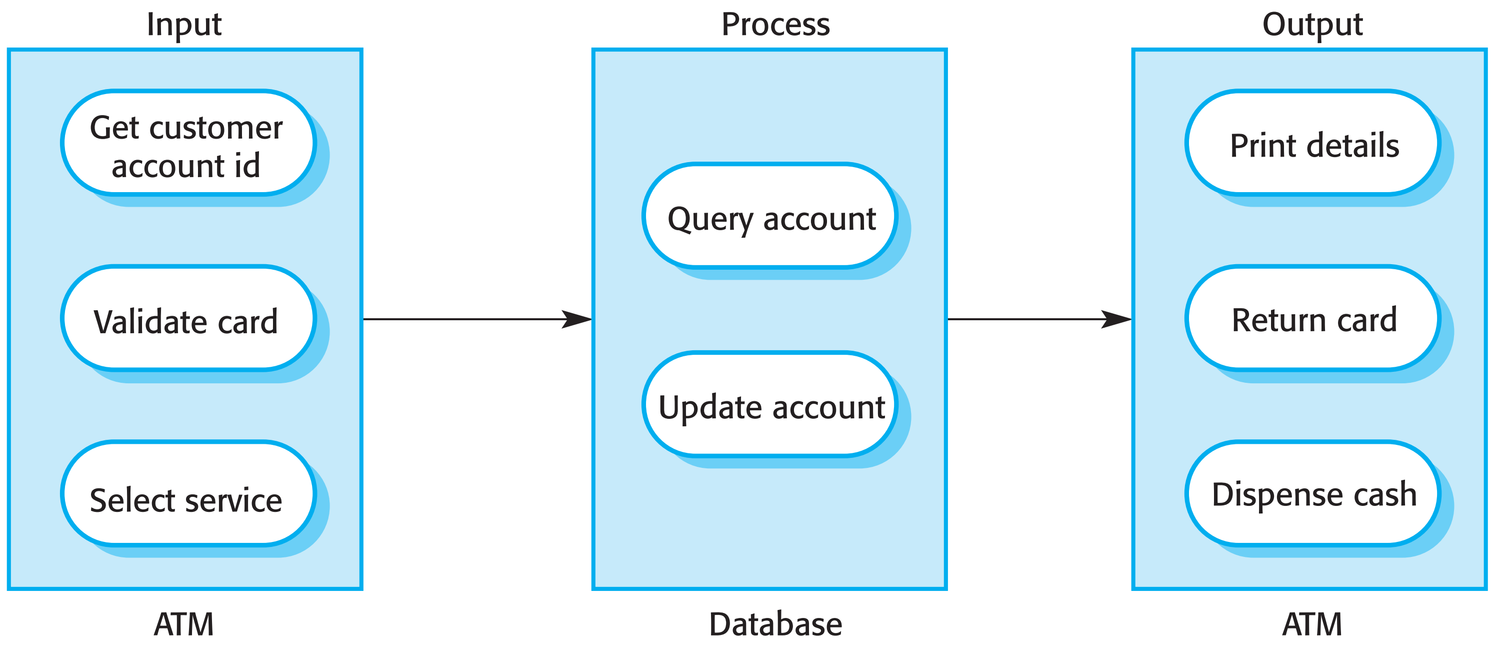
Transaction processing systems

- ✧ Process user requests for information from a database or requests to update the database.
- ✧ From a user perspective a transaction is:
  - Any coherent sequence of operations that satisfies a goal;
  - For example - find the times of flights from London to Paris.
- ✧ Users make asynchronous requests for service which are then processed by a transaction manager.



The structure of transaction processing applications

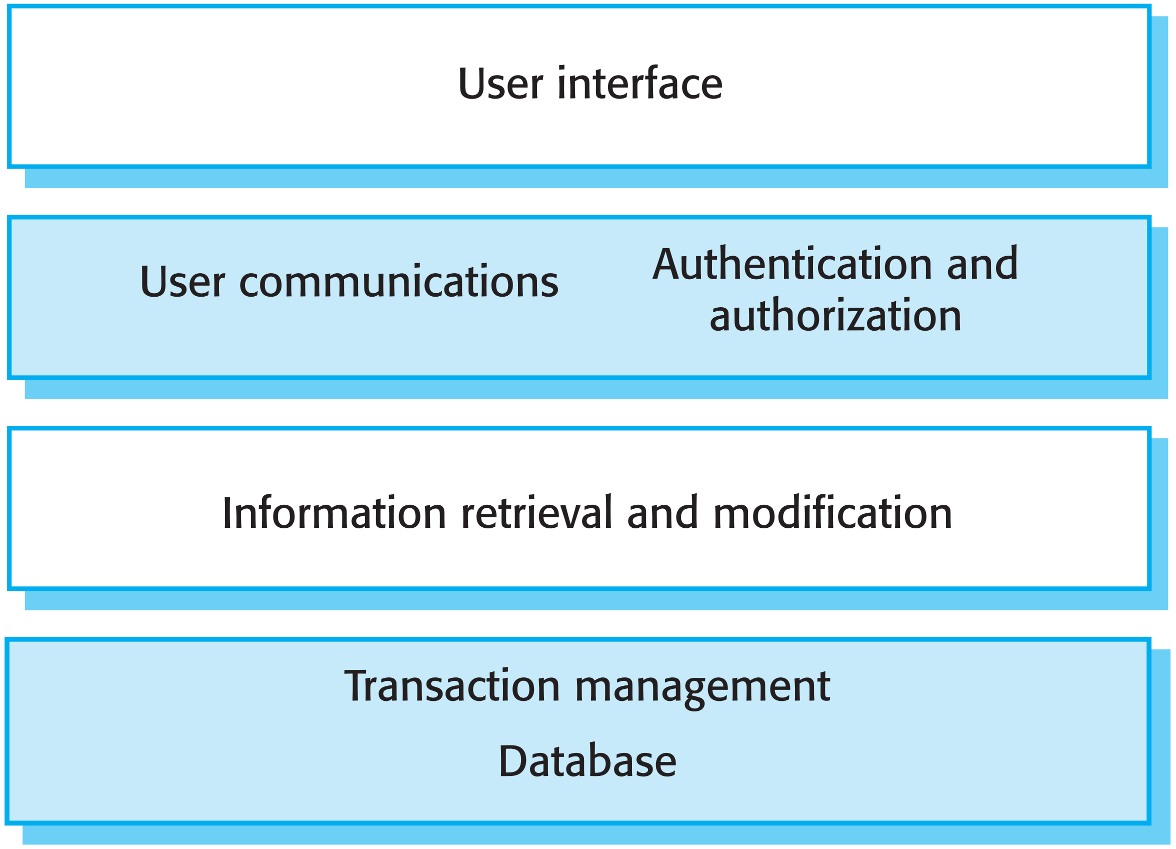
The software architecture of an ATM system



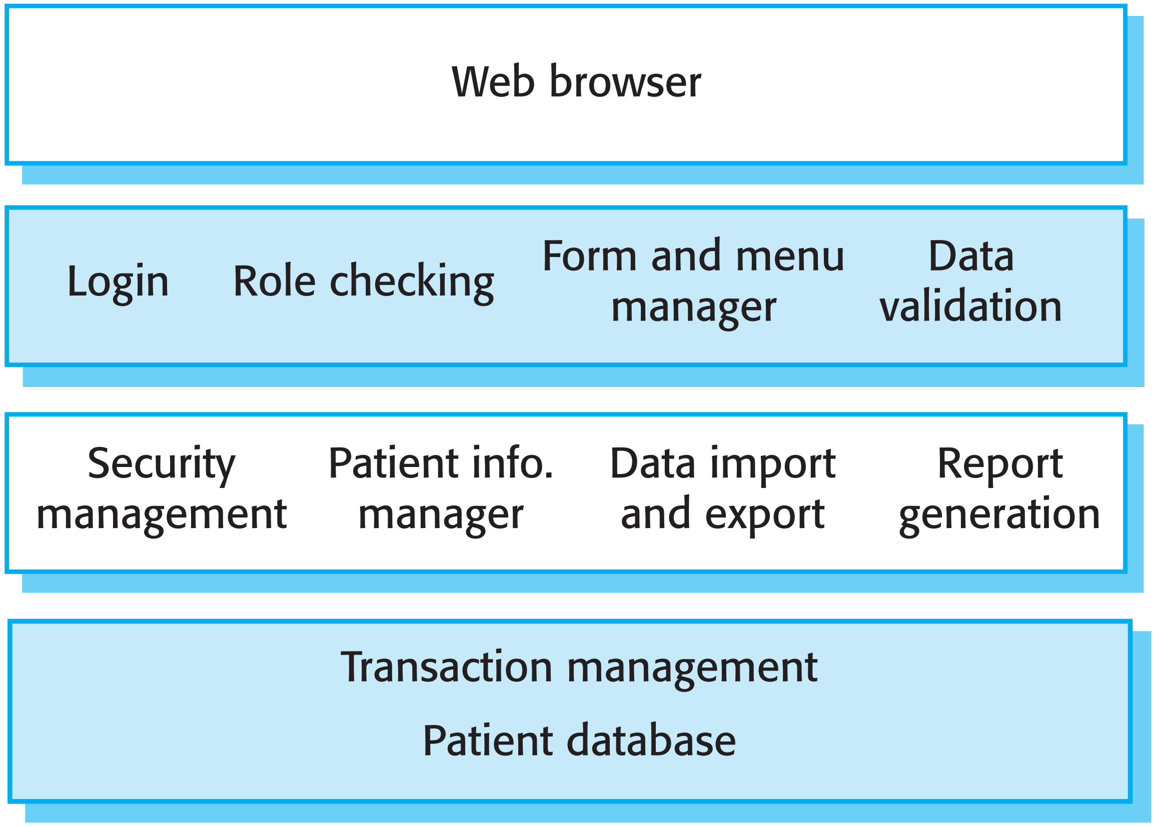
Information systems architecture

- Information systems have a generic architecture that can be organized as a layered architecture.
- These are transaction-based systems as interaction with these systems generally involves database transactions.
- Layers include:
  - The user interface
  - User communications
  - Information retrieval
  - System database

Layered information system architecture


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

The architecture of the Mentcare system



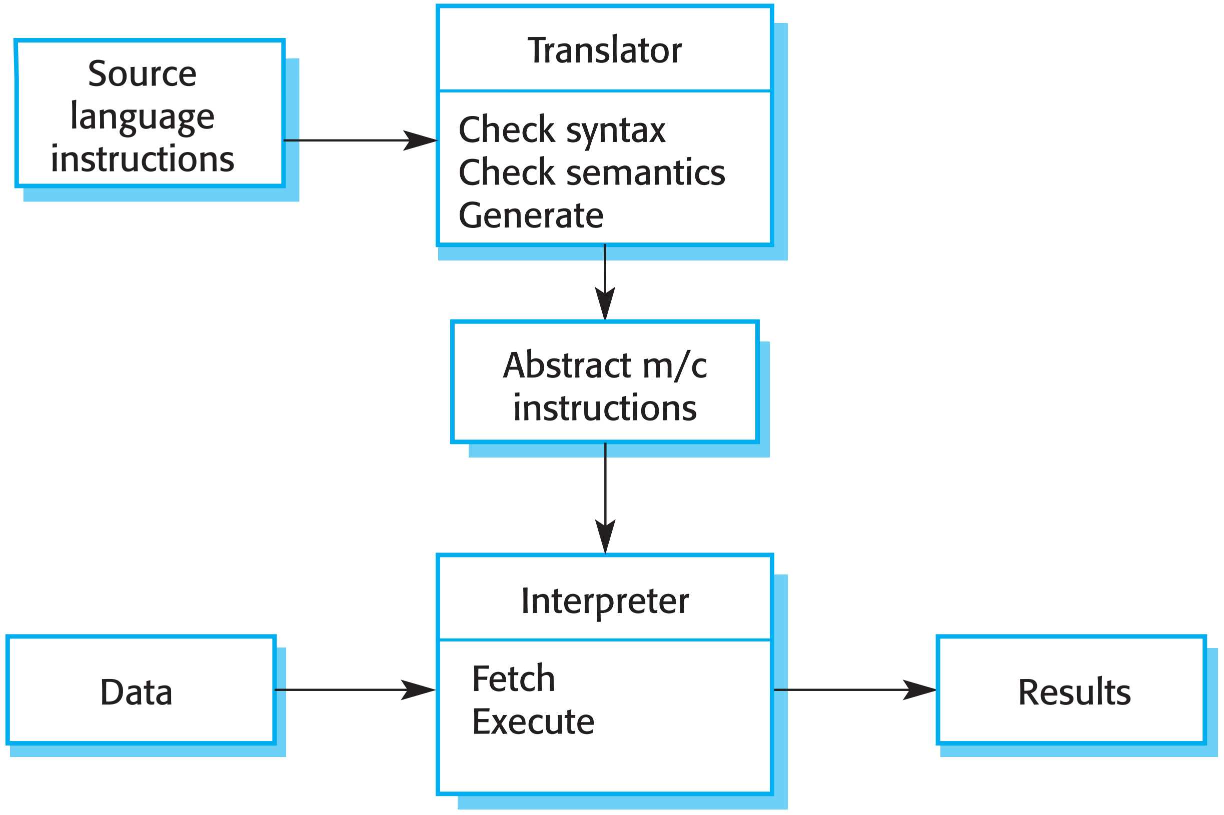
Web-based information systems

- Information and resource management systems are now usually web-based systems where the user interfaces are implemented using a web browser.
- These systems are often implemented as multi-tier client server/architectures
  - The web server is responsible for all user communications, with the user interface implemented using a web browser;
  - The application server is responsible for implementing application-specific logic as well as information storage and retrieval requests;
  - The database server moves information to and from the database and handles transaction management.

Language processing systems

- ✧ Accept a natural or artificial language as input and generate some other representation of that language.
- ✧ May include an interpreter to act on the instructions in the language that is being processed.
- ✧ 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.

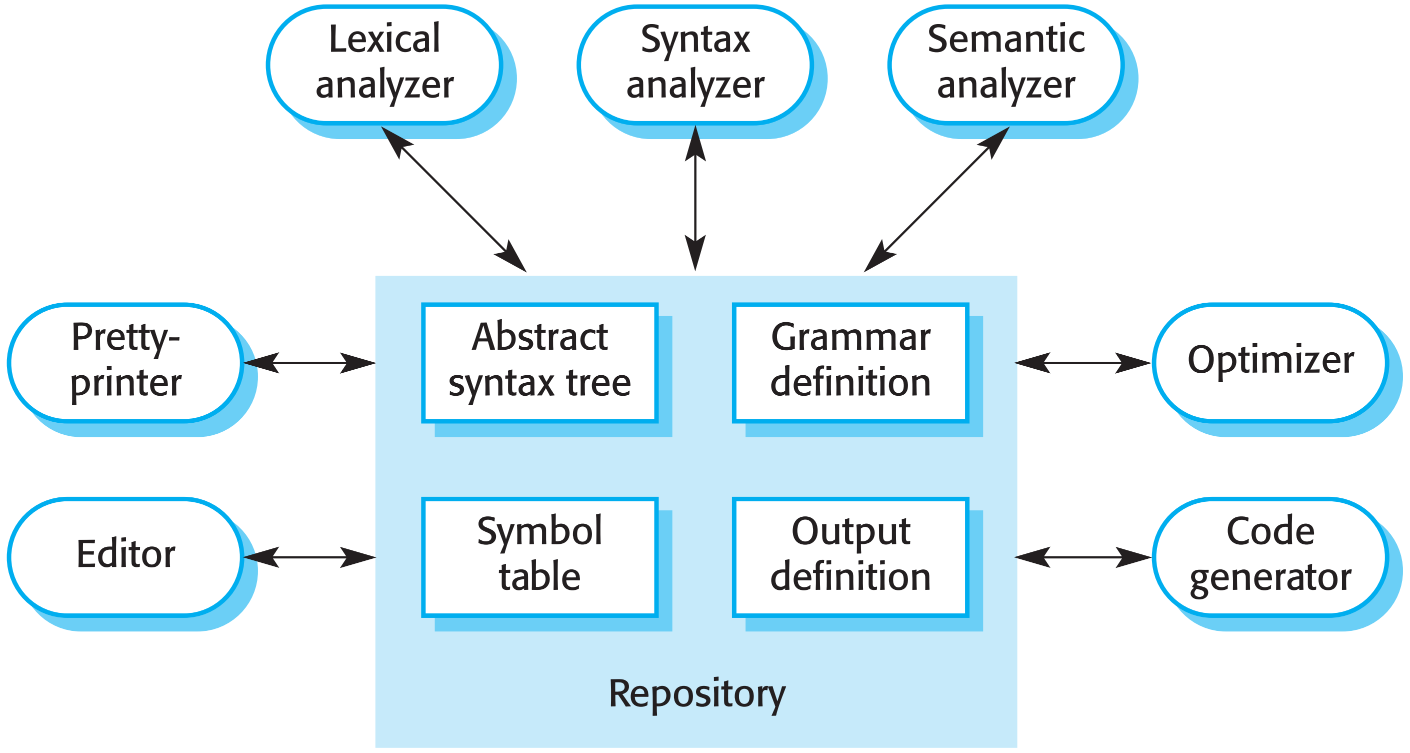
The architecture of a language processing system



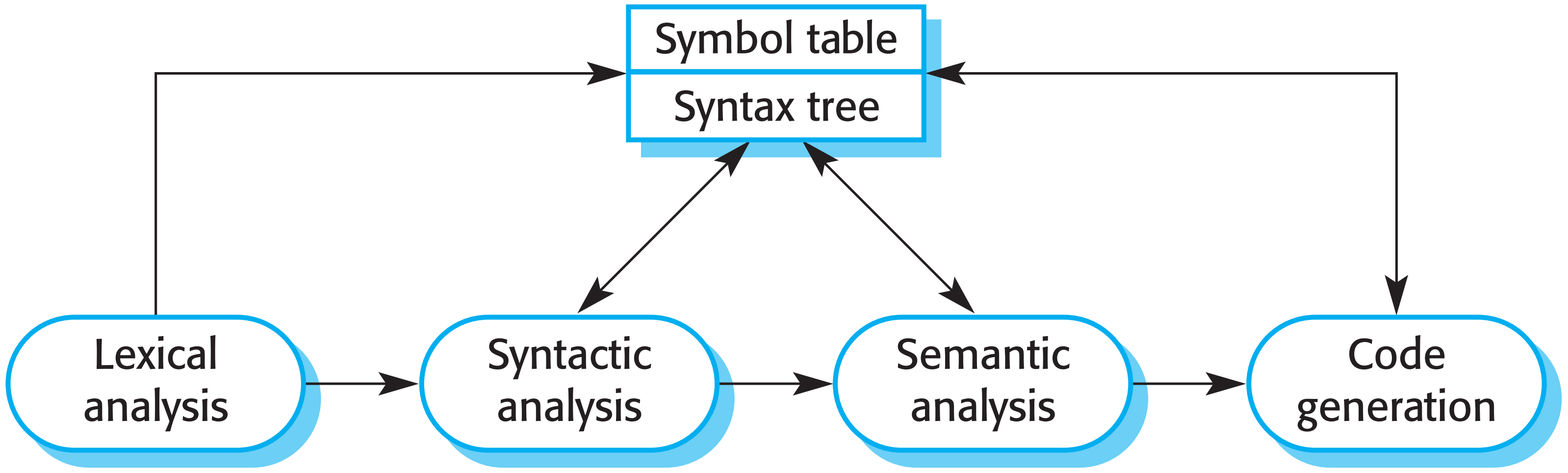
Compiler components

- 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.
- 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



Summary

- An architecture contains
  - components (modules, entities)
  - connectors (relationships)
  - rationale
    - qualities to be achieved
    - constraints on the organization and contents of the software
- Similar problems are often addressed with similar *styles* of architecture
- An appropriate architectural style offers a useful starting point for architecting a software system

Questions?