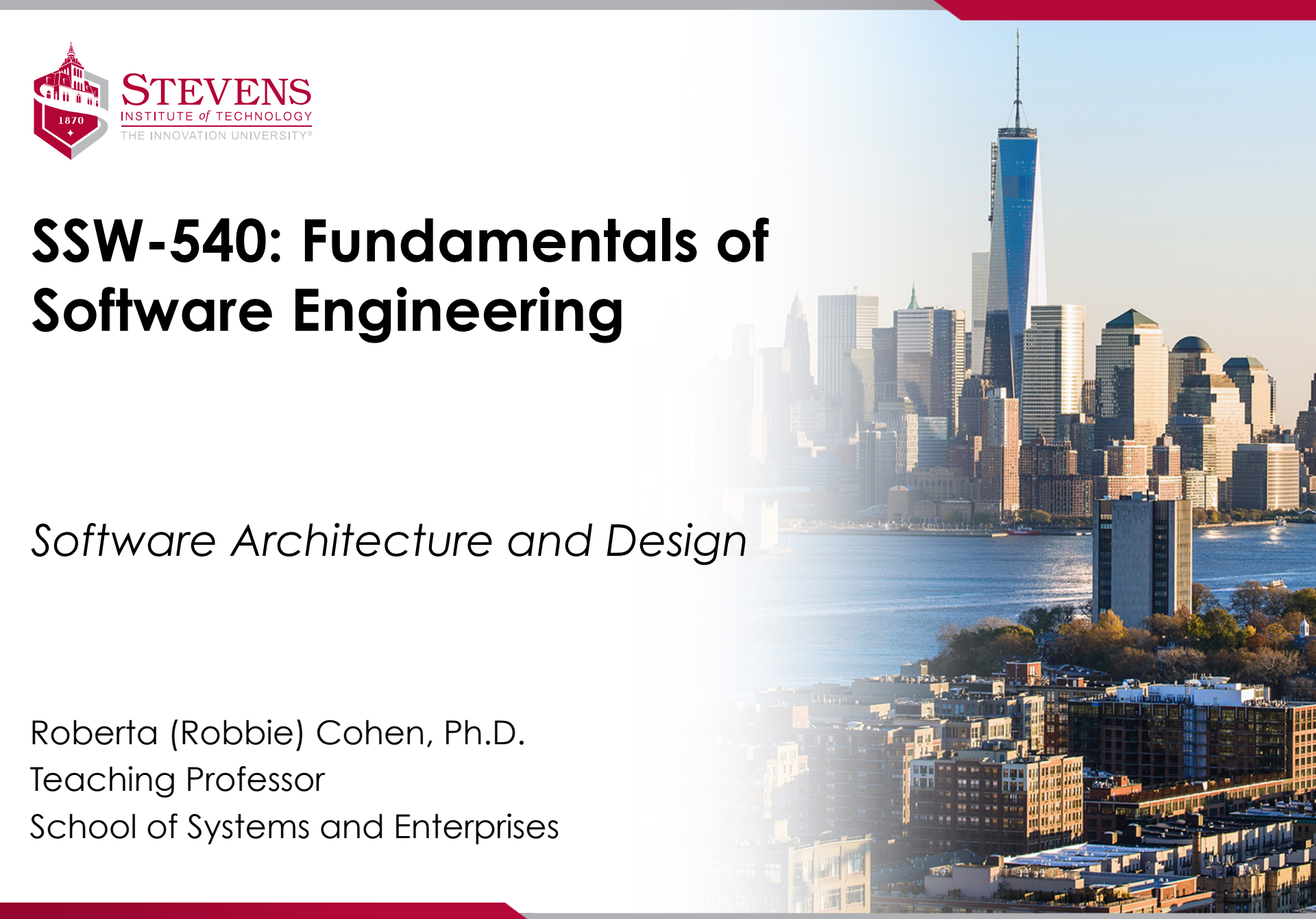


SSW-540: Fundamentals of Software Engineering

Software Architecture and Design

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

Python Pointers

Bounding your input is a critical skill for programmers since it reduces the likelihood of untoward effects in execution. Usually, one creates boundaries that limit the value of numeric input, but the same procedures can be used to insure that your input is provided in the form you need.

So, when you need numeric input…

```
error_message = "Bad score. Scores should be a numeric percentage between 0 and 1 and should not contain letters or punctuation."
try:
        score = float ( raw_input ( 'Enter a numeric percentage score between 0 and 1: ' ) )
        if score <= 1.0 and score >= 0:
                print "Okay!"
        else:
                print error_message
except:
        print error_message
```

Converting the input to a floating point number in the try section catches alpha input, and I go on to test numeric boundaries with an if statement.

Architectural design

- Architectural design is
  - how a software system should be organized; its overall structure
    - for embedded systems, software design must consider the design and performance of the system hardware and the site for capabilities (SW/HW?)
  - the critical link between detailed design and requirements engineering
    - for embedded systems, software design must also consider additional requirements such as power and battery management
  - an architectural model that describes how the system is organized as a set of communicating components.
- An early stage of agile processes often includes design of an overall systems architecture
  - Refactoring a system's architecture is usually costly because it affects so many components in the system

Advantages of explicit architecture

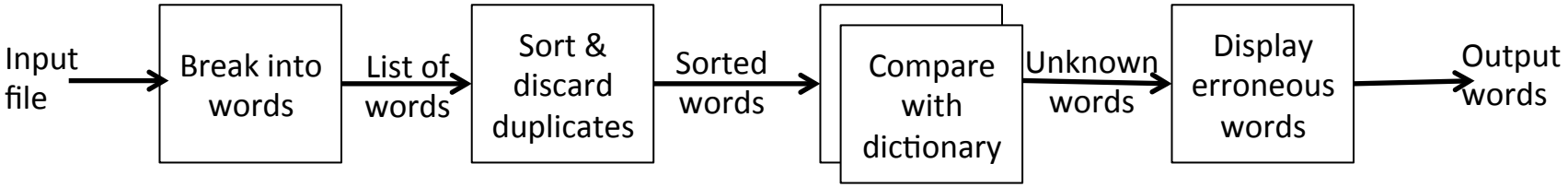
- Stakeholder communication
  - Architecture may be used as a focus of discussion by system stakeholders.
- System analysis
  - Provides a way to analyse whether a system can meet its non-functional requirements.
- Large-scale reuse
  - The architecture may be reusable across a range of systems
  - Product-line architectures may be developed.

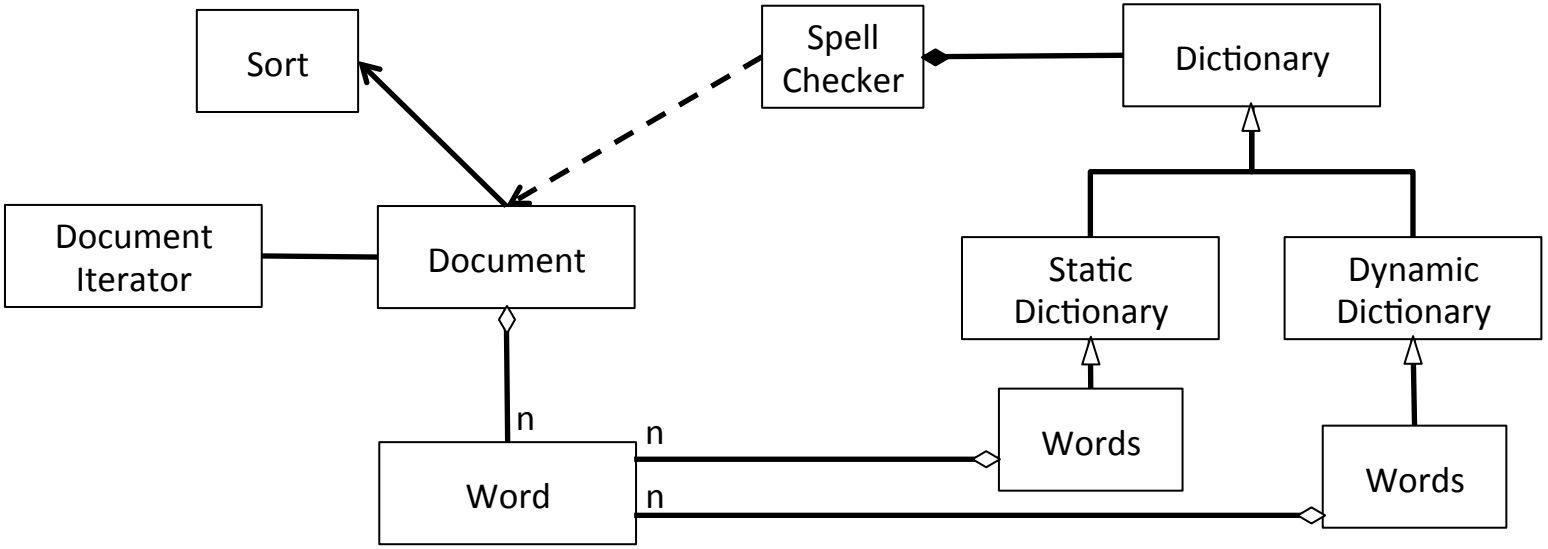
Why create an architectural design?

- Through architecture, we satisfy NFRs like performance, security, correctness, safety, availability, usability and maintainability!
  - "[The client] never mentioned [a requirement] that webpages could not take more than two seconds to load, but he complained about it afterward."
- NFRs drive decisions about
  - The content of and connections among components
  - Architectural patterns
  - Implementation strategies
  - Technological platforms

A quick example

- A spell checker with a main and private dictionary is needed.
- Here are two architectures; which one do you prefer? Why?

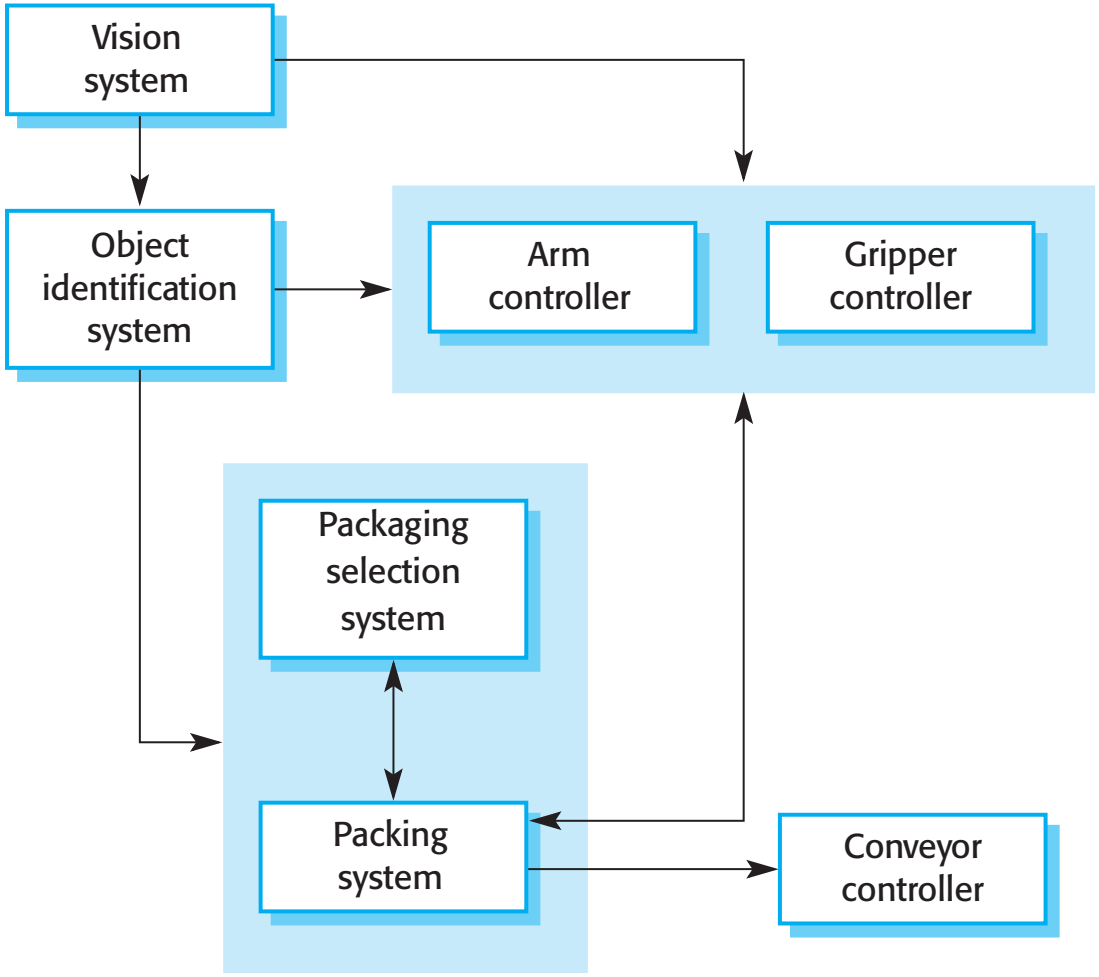




Architectural representations

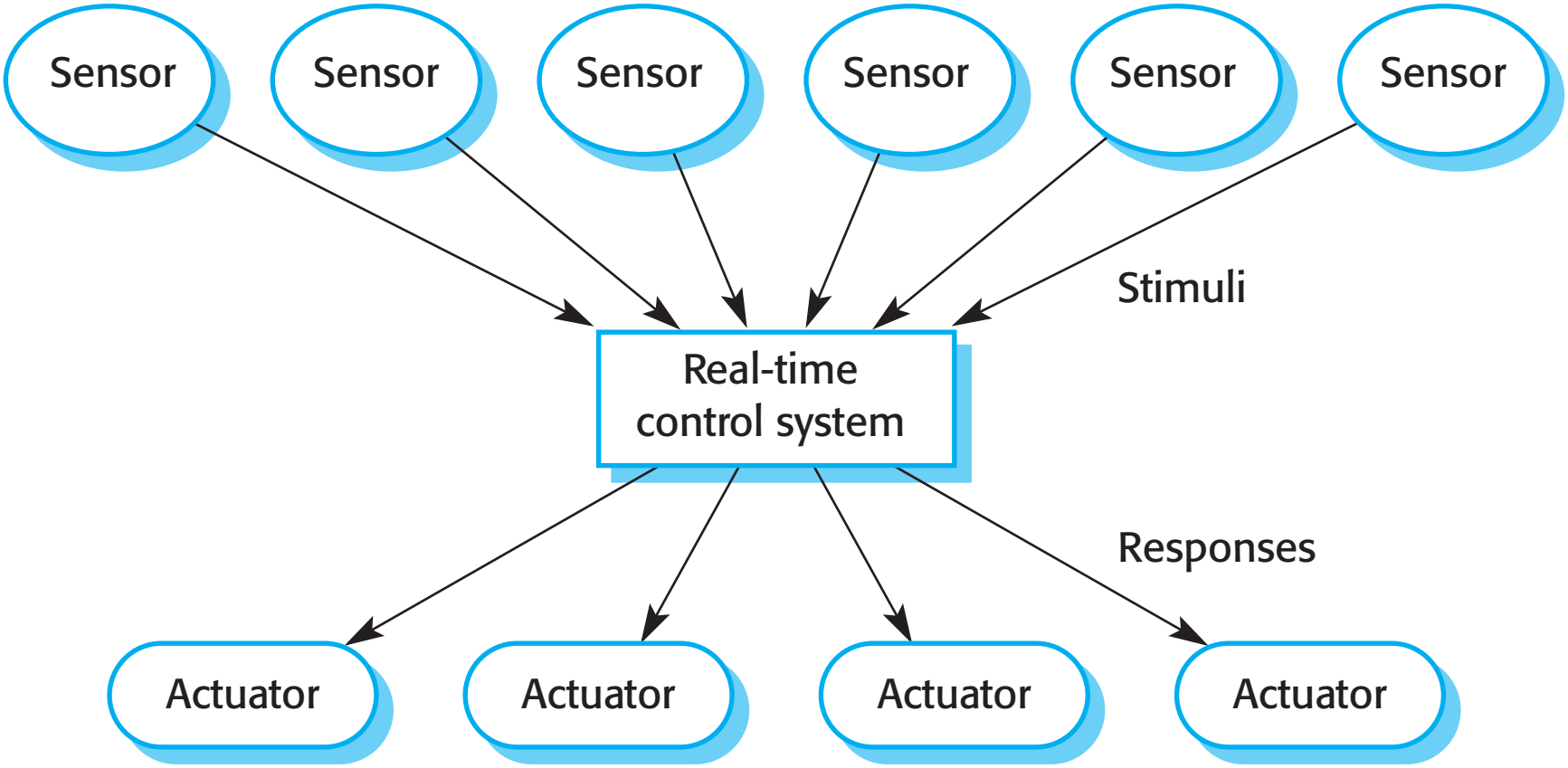
- Simple, informal block diagrams (box and line diagrams) showing entities and relationships are the most often used method for documenting software architectures.
  - Very abstract - they do not show the nature of component relationships nor the externally visible properties of the sub-systems.
  - However, useful for communication with stakeholders and for project planning.
- Such drawings lack semantics and do not show the types of relationships between entities nor the visible properties of entities in the architecture.
  - The requirements for model semantics depend on how the models are used.
  - If for communication, boxes and lines will do; but if for development a complete and semantically rich system model is needed showing the different components in a system, their interfaces and their connections.

The architecture of a packing robot control system



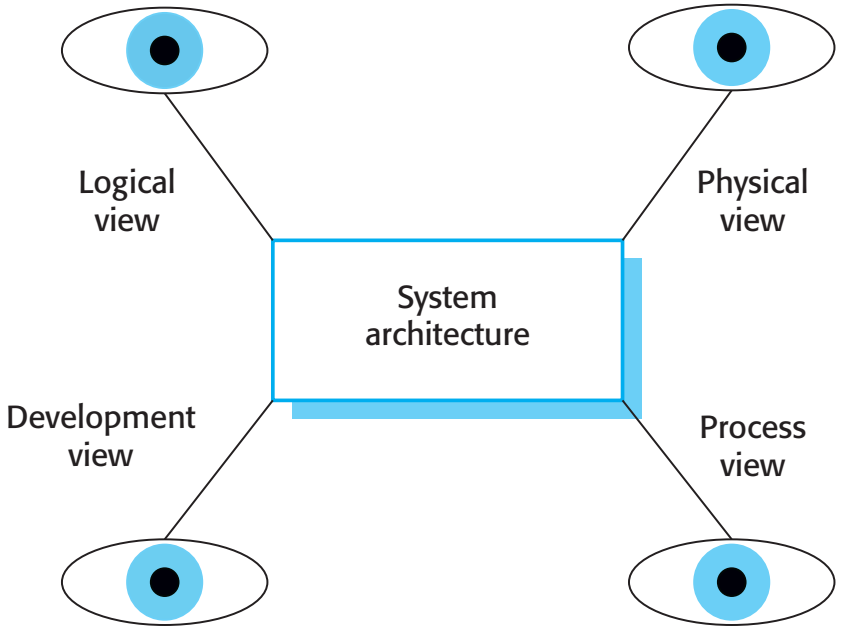
- Robots are reactive systems
  - Given a stimulus, they must react within a specified time
  - Correctness depends on both content and timing
- Reactive systems may use
  - Periodic stimuli that occur at predictable intervals
  - Aperiodic stimuli that occur at unpredictable times.

A general model of an embedded real-time system



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.



4 + 1 view model of software architecture (Kruchten)

Logical View
Objects and object classes - the system's data and methods

Process View
An "alive view" of interacting processes

Use case views/scenarios
Requirements used as test cases for the architecture

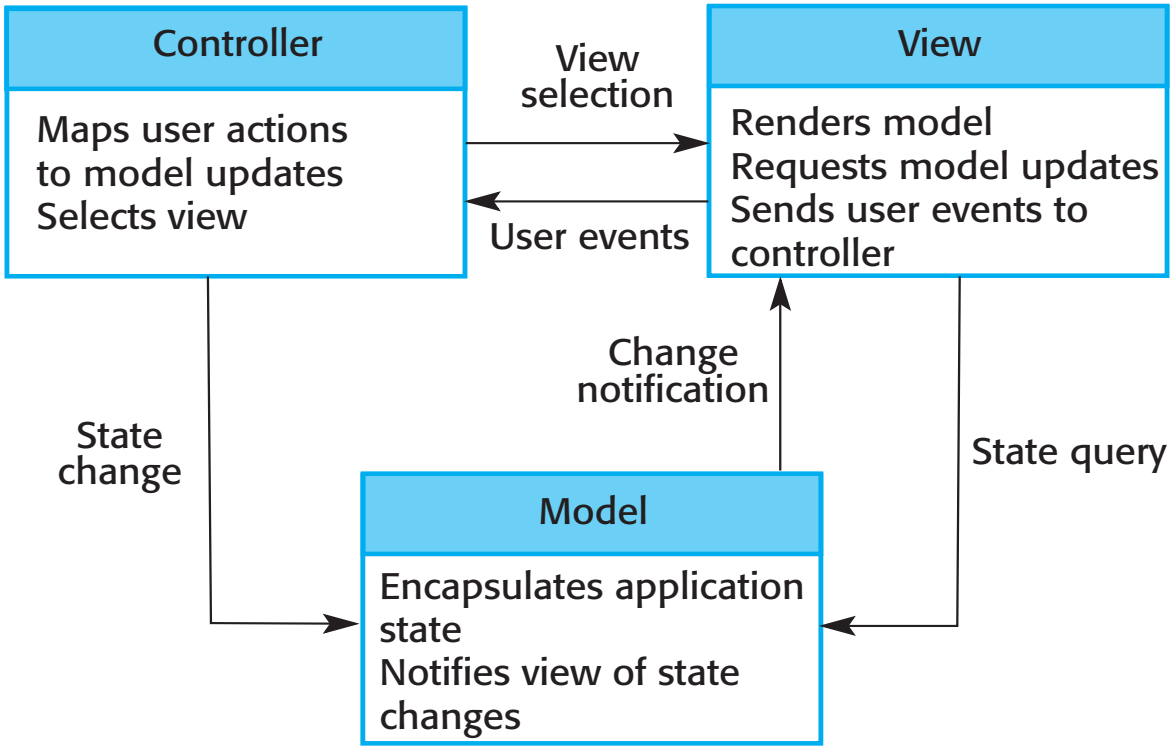
Physical View
Software distributed across hardware; communications

Development View
Software decomposed for development

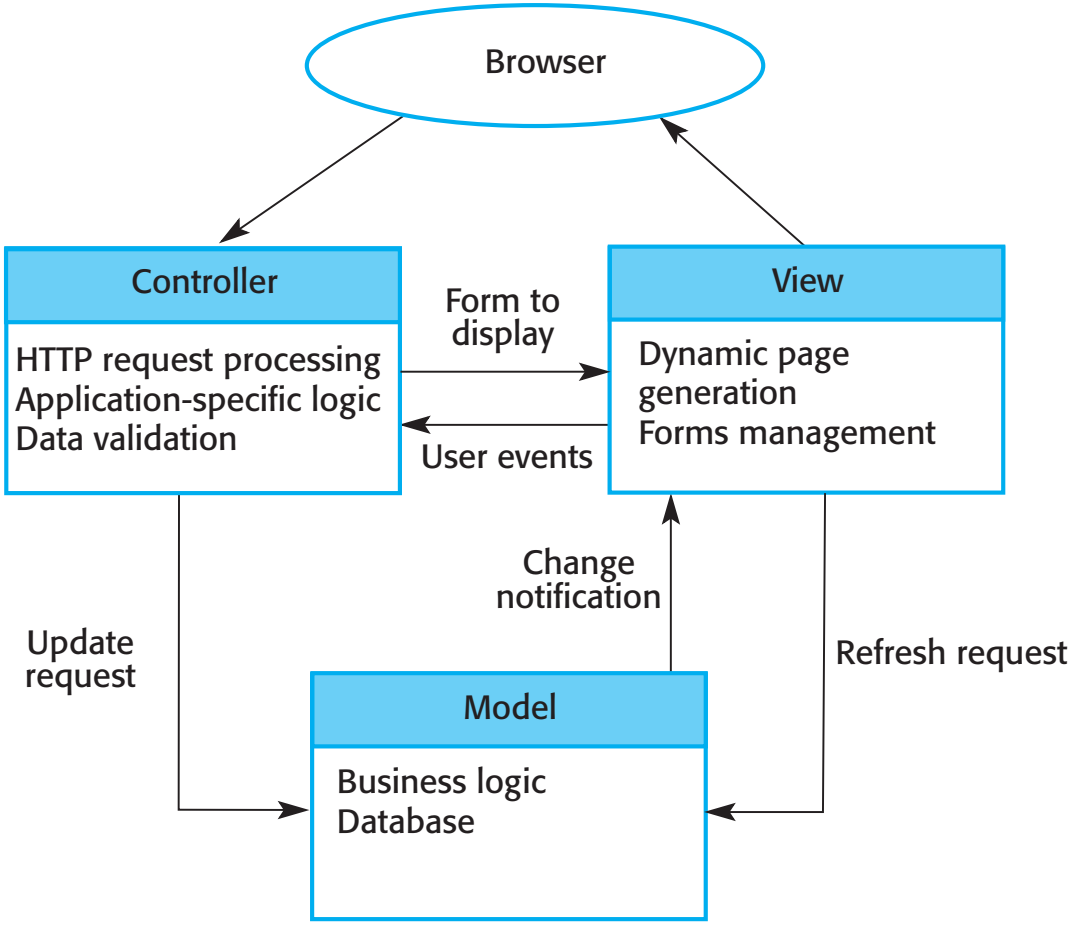
Architectural patterns

- 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 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.

The organization of the Model-View-Controller



Web application architecture using the MVC pattern



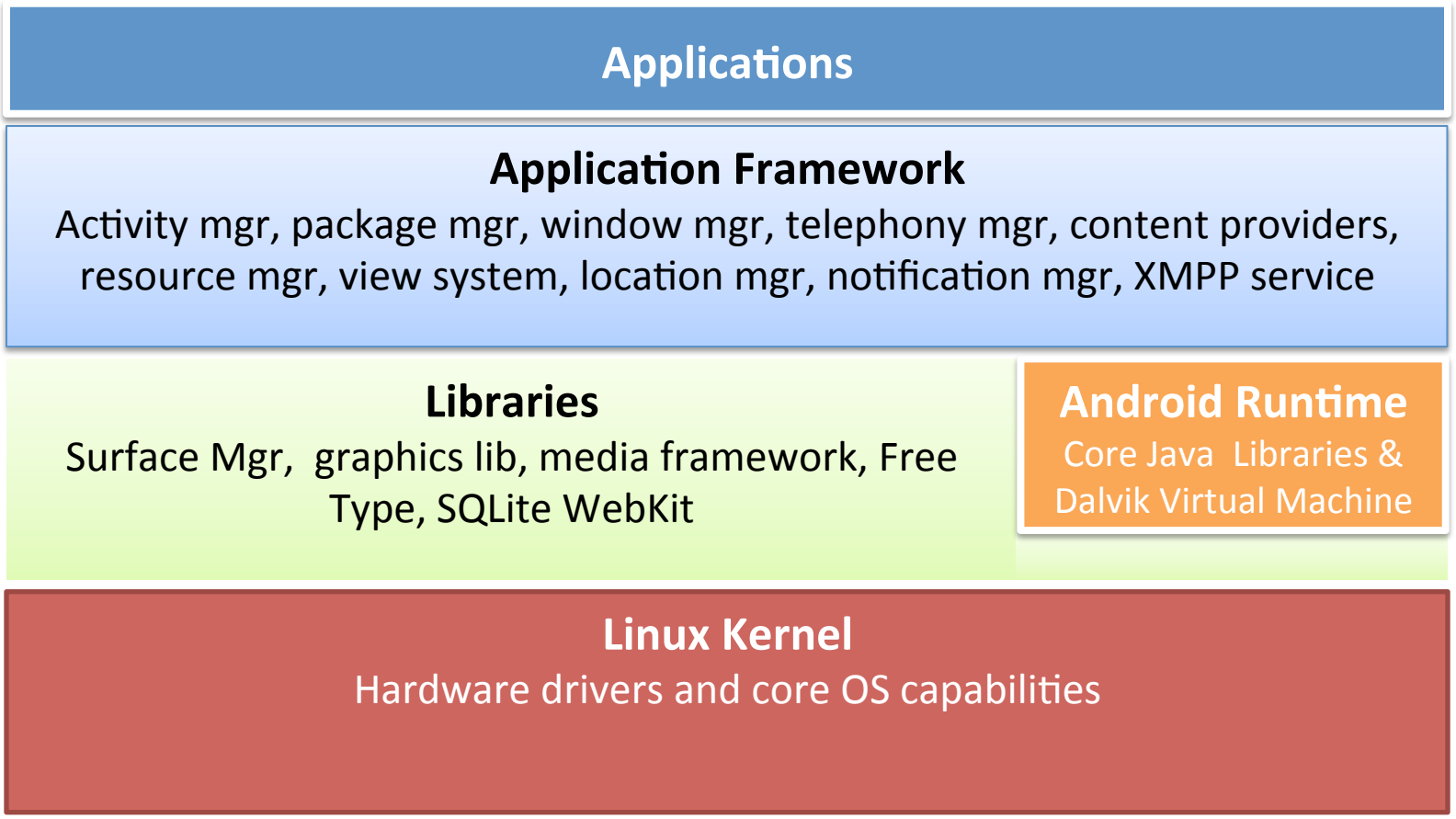
The Model-View-Controller (MVC) pattern

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 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.

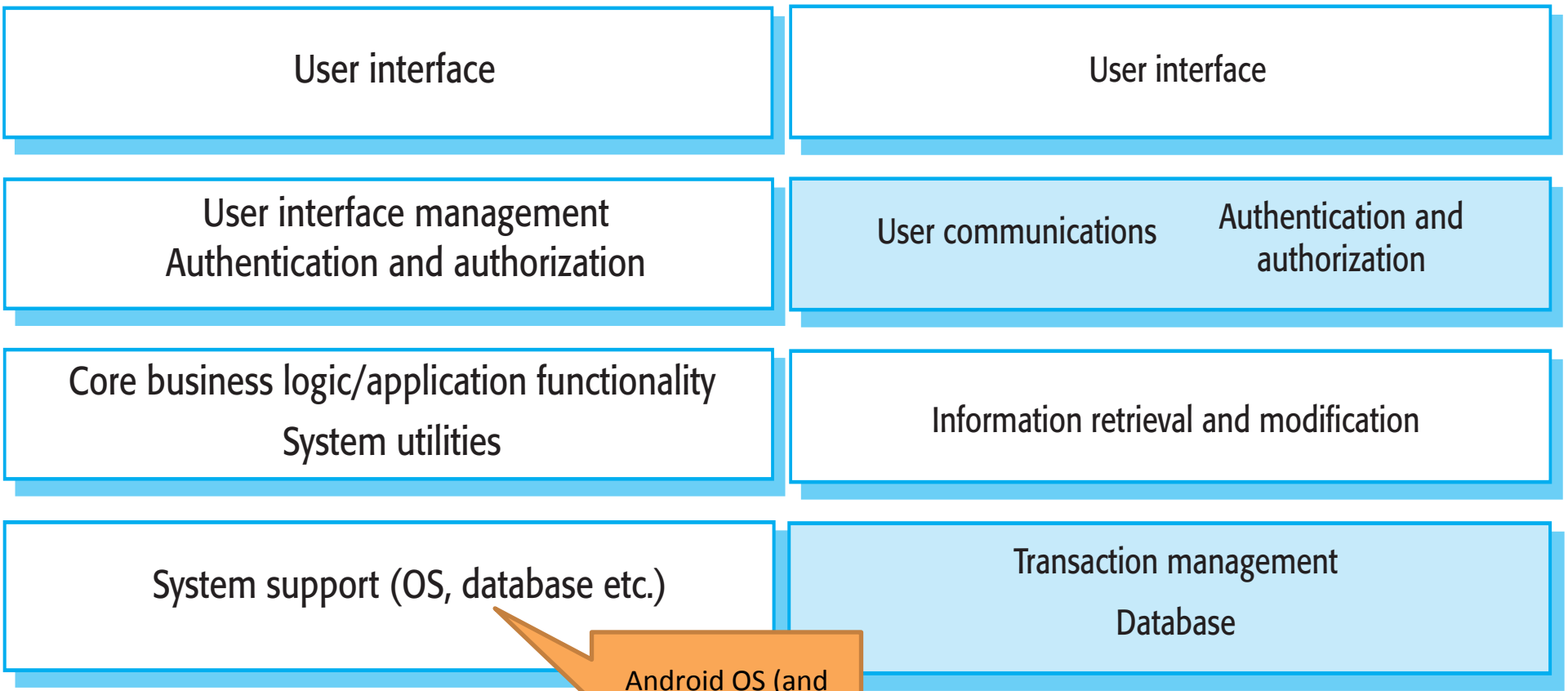
Layered architecture

- 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.
- Supports the incremental development of sub-systems in different layers. When a layer interface changes, only the adjacent layer is affected.
- However, it is sometimes artificial to structure systems in this way.

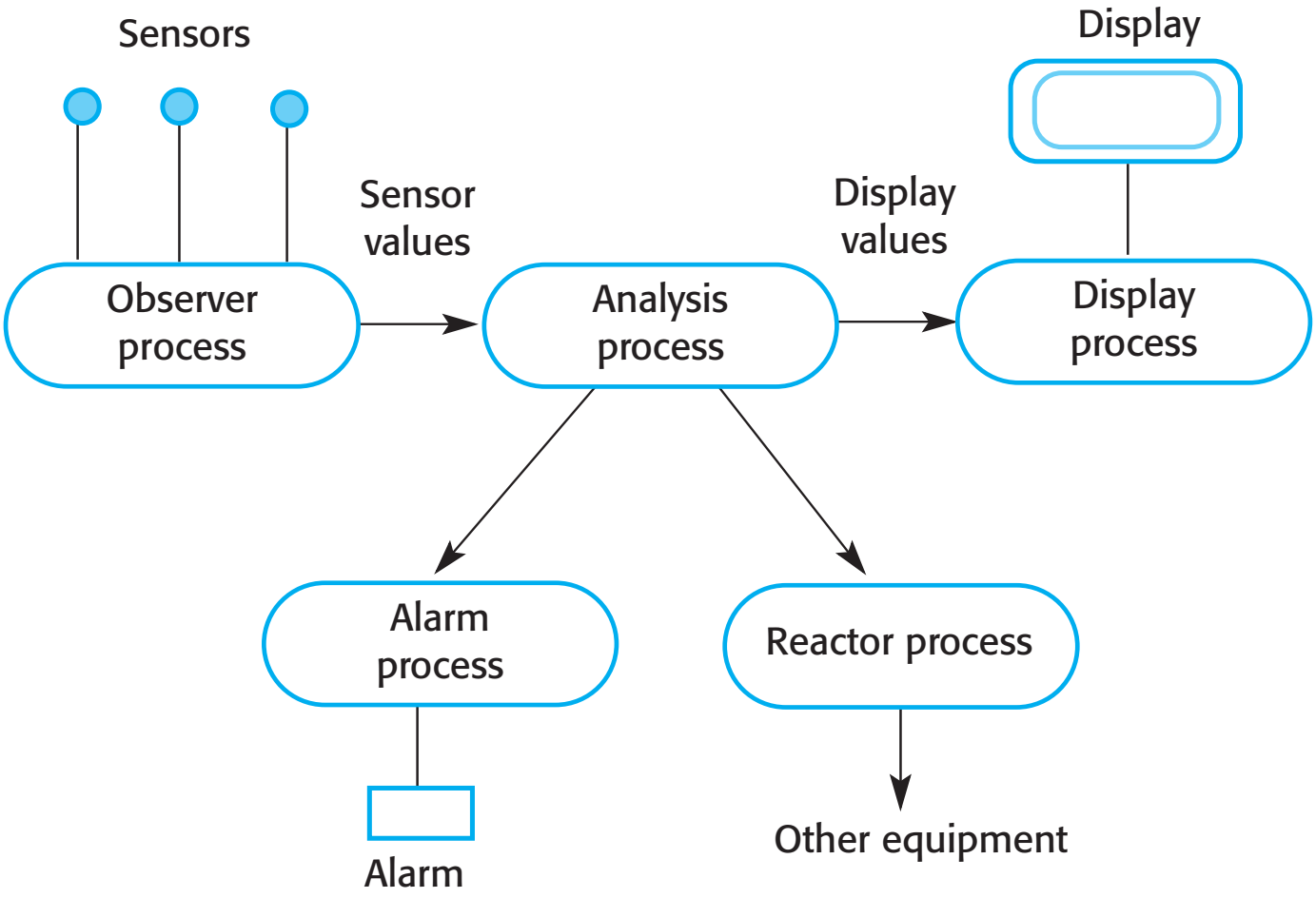
The architecture of the Android operating system



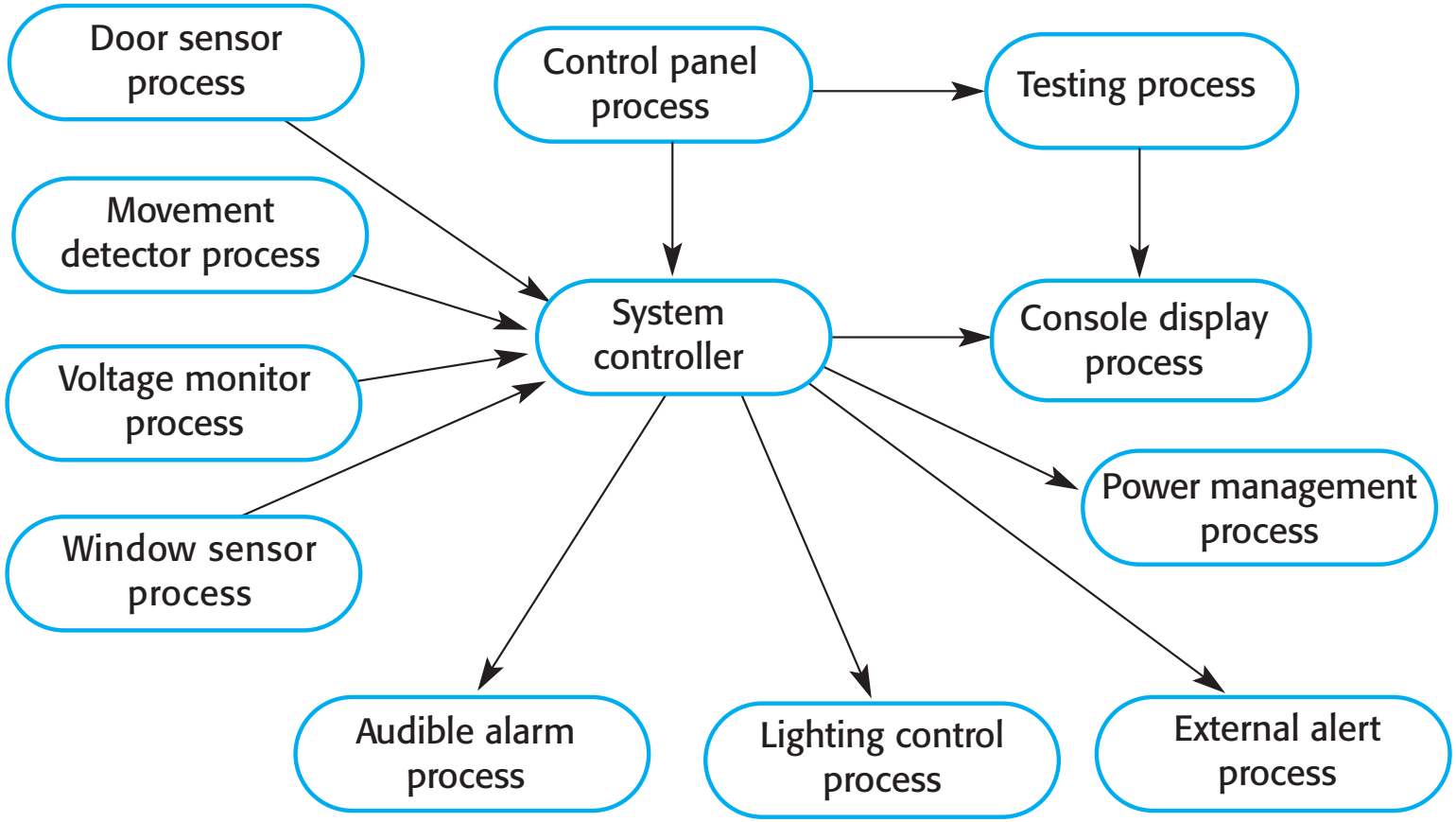
A generic and an information systems layered architecture



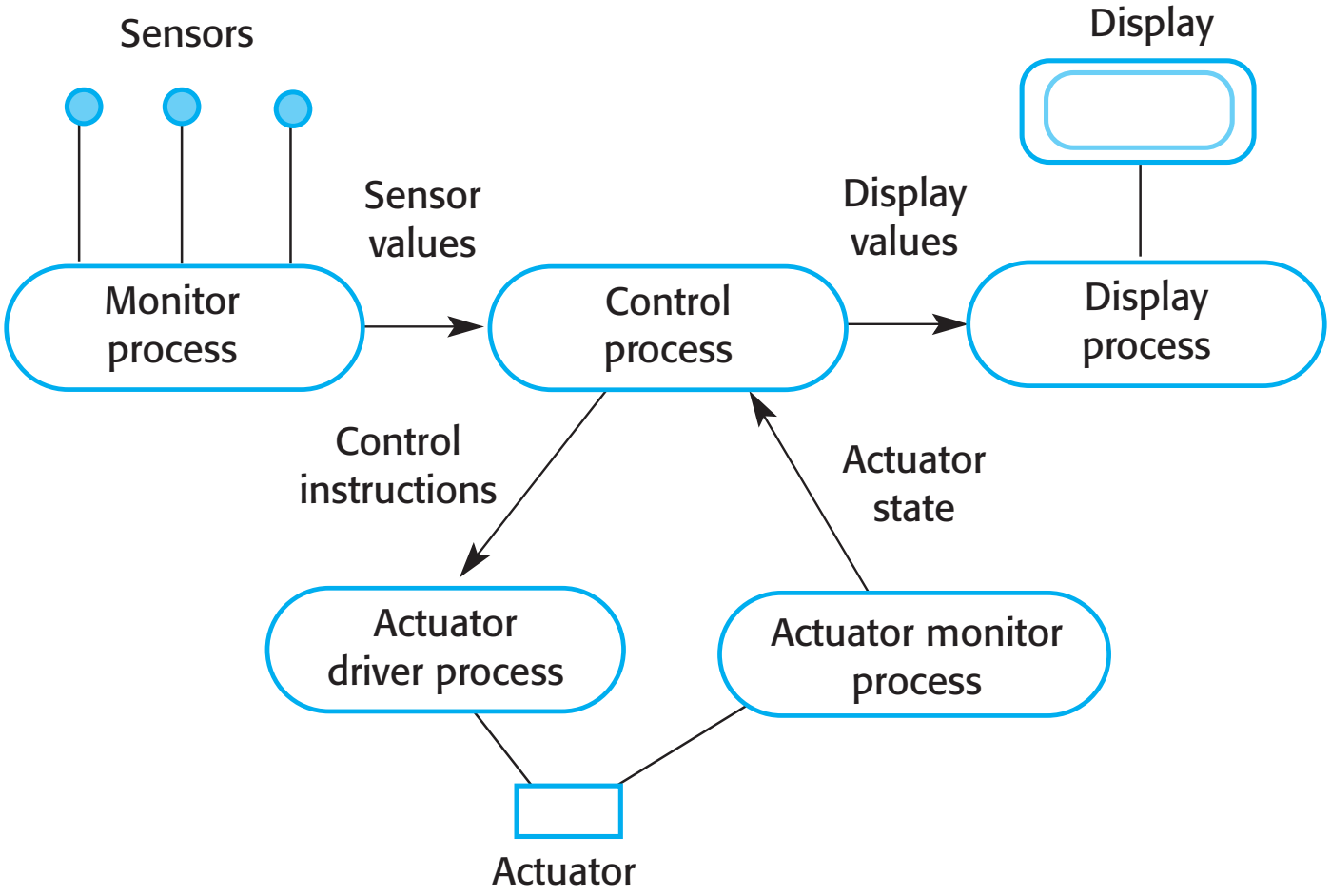
Observe and React process pattern



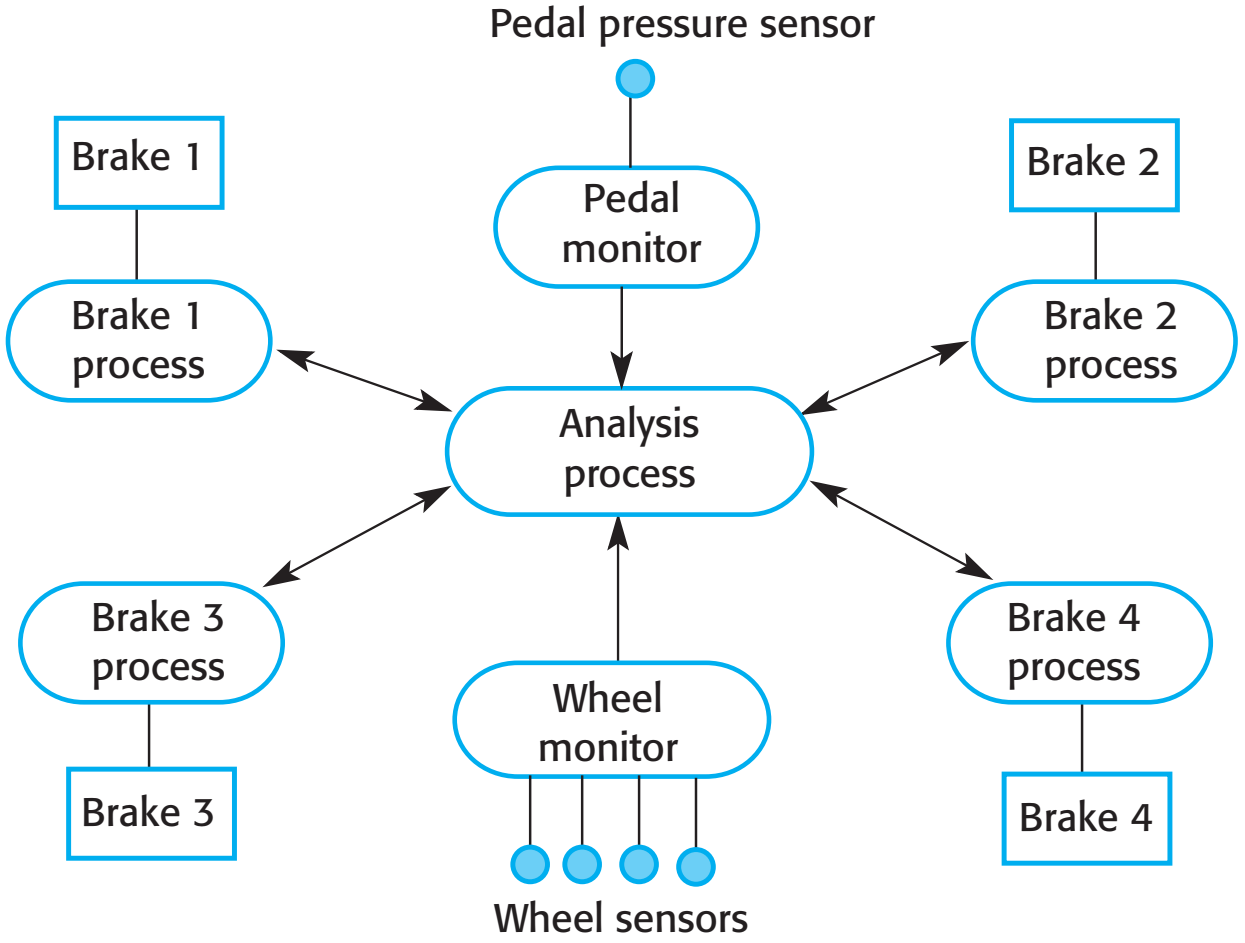
Process structure for a burglar alarm system



Environmental Control process structure



Control system architecture for an anti-skid braking system



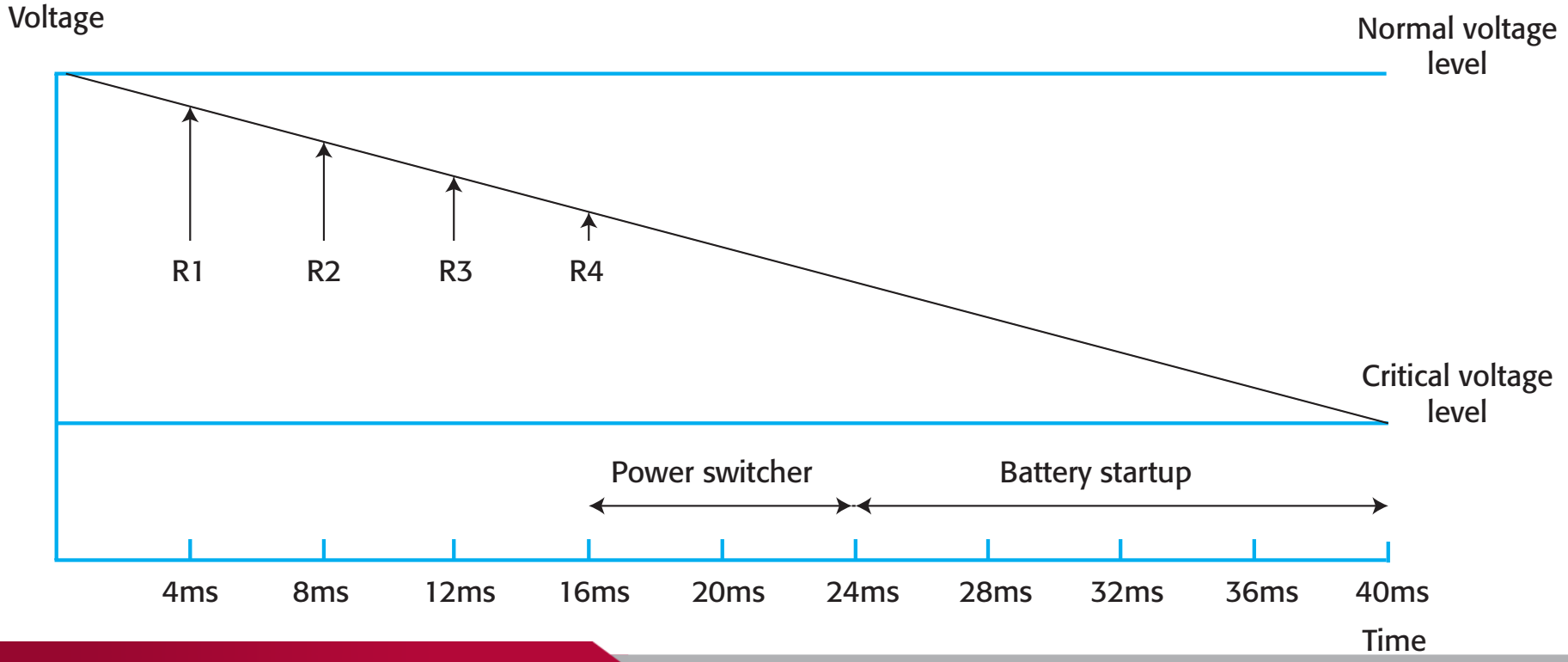
Some words on timing analysis

- 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

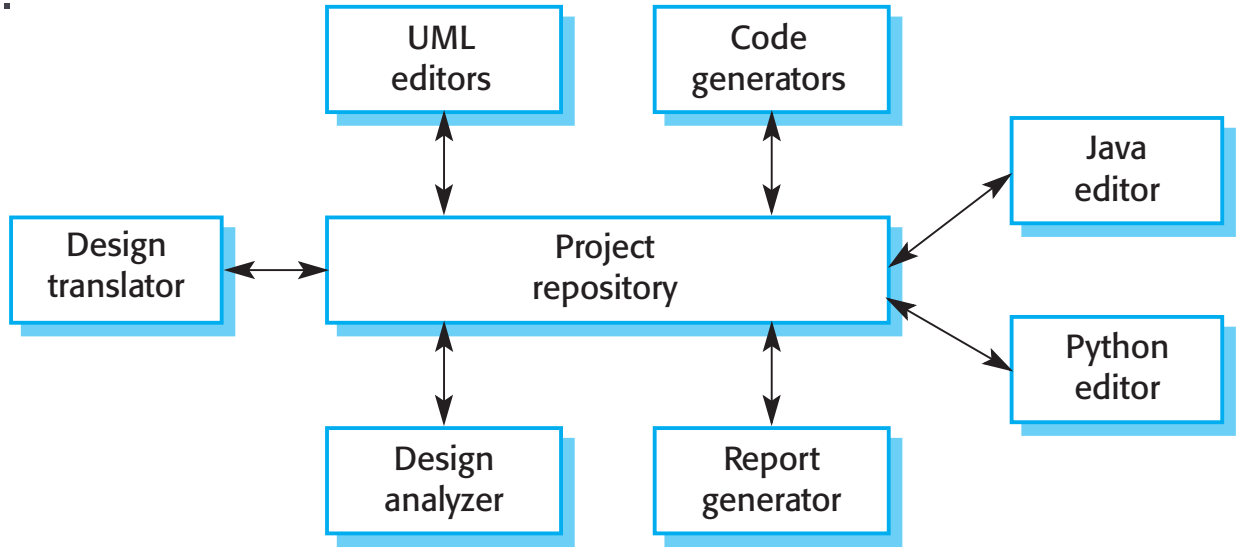
How often should one measure the voltage?



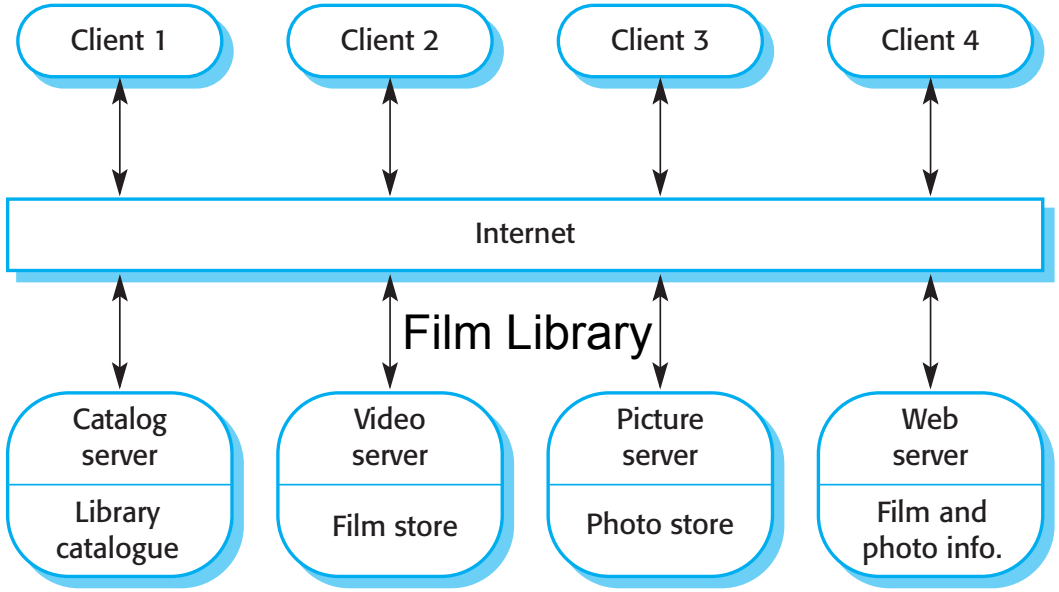
Repository architecture

- 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)

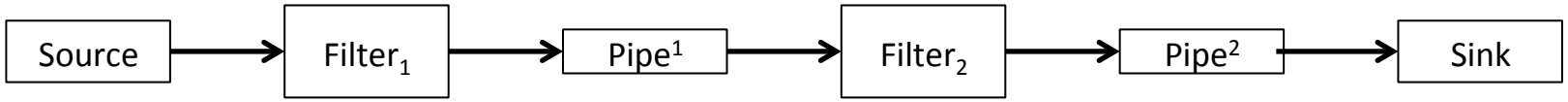


Client-server architecture

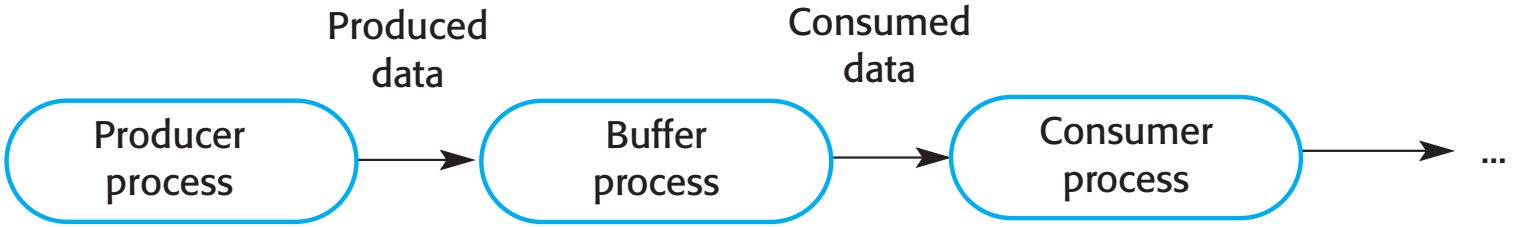


- Distributed system model which shows how data and processing is distributed across a range of components.
  - Often implemented with a service on multiple redundant computers
  - Can be implemented on a single computer.
- Set of stand-alone servers which provide specific services such as printing,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

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.
- Embedded variation called a **Process Pipeline pattern**



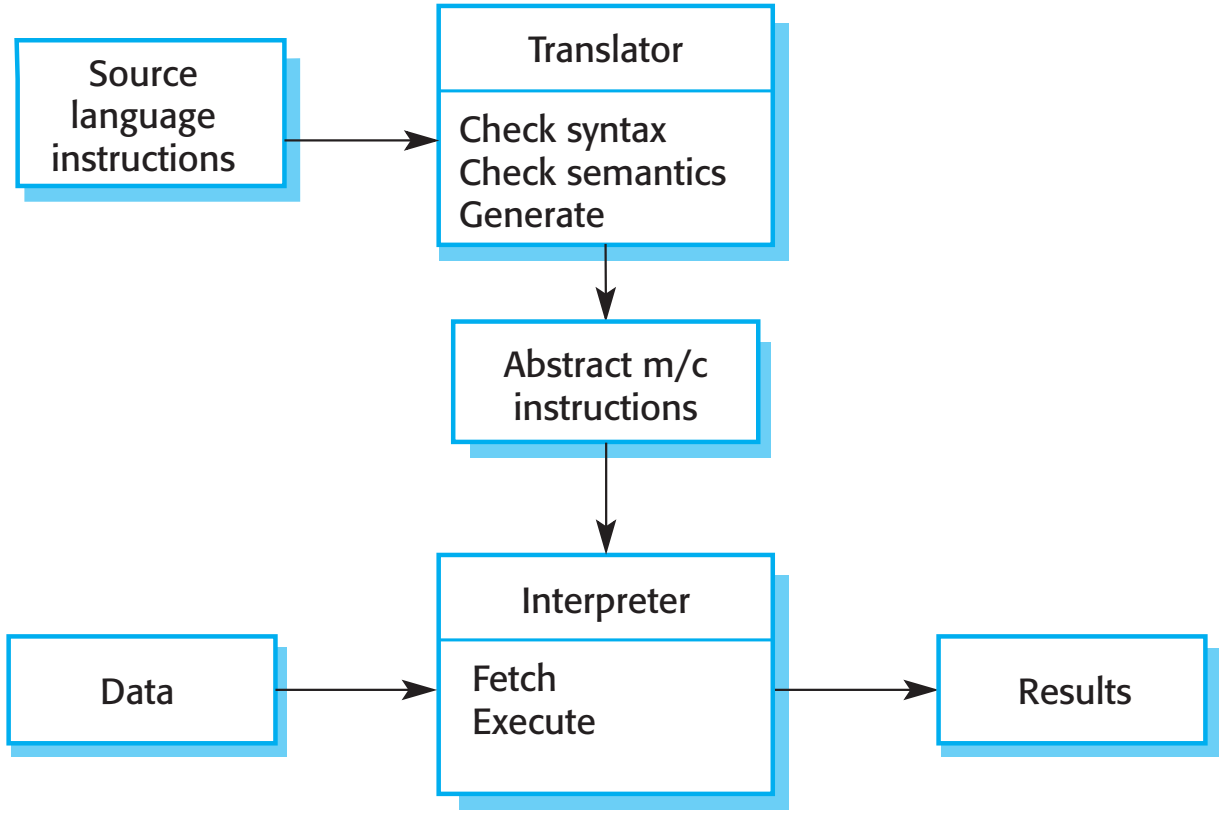
Application architectures

- Application systems are designed to meet an organizational need.
- As 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.

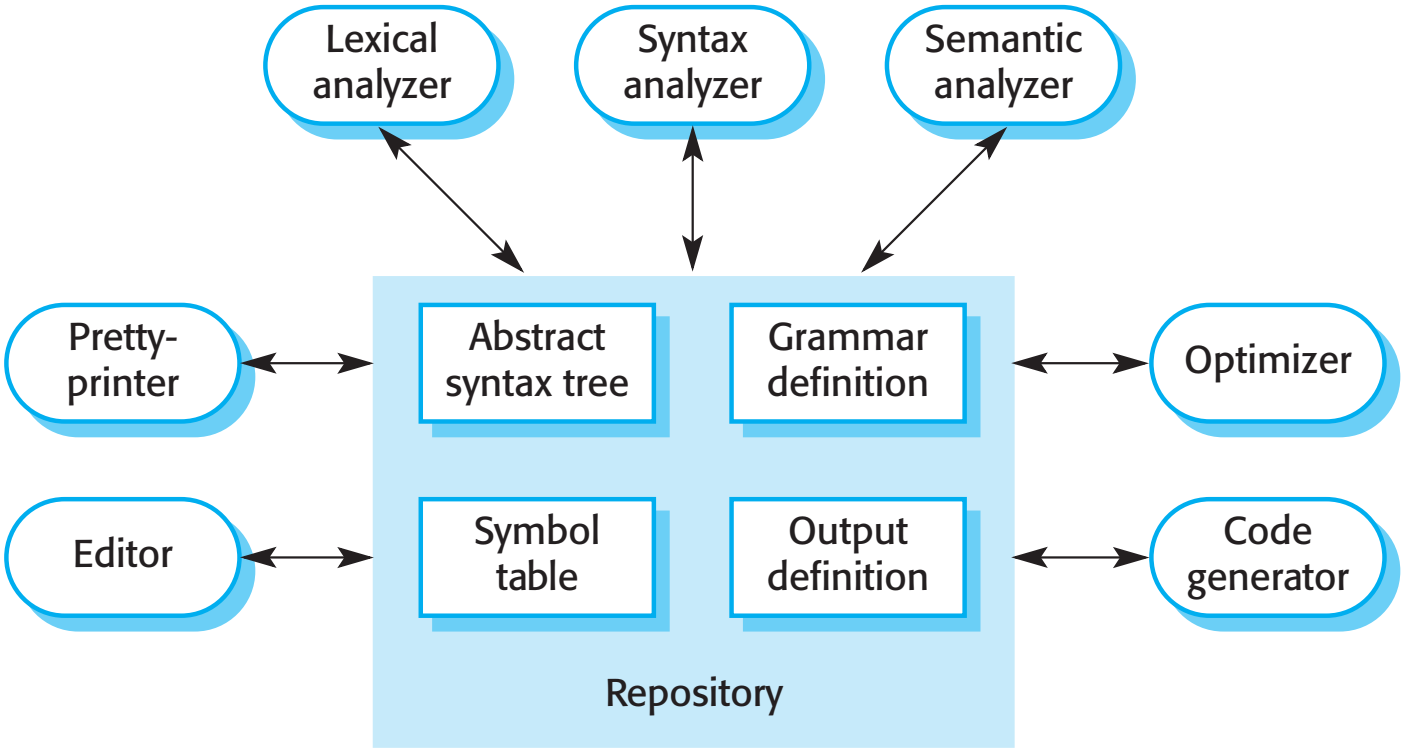
The architecture of a language processing system



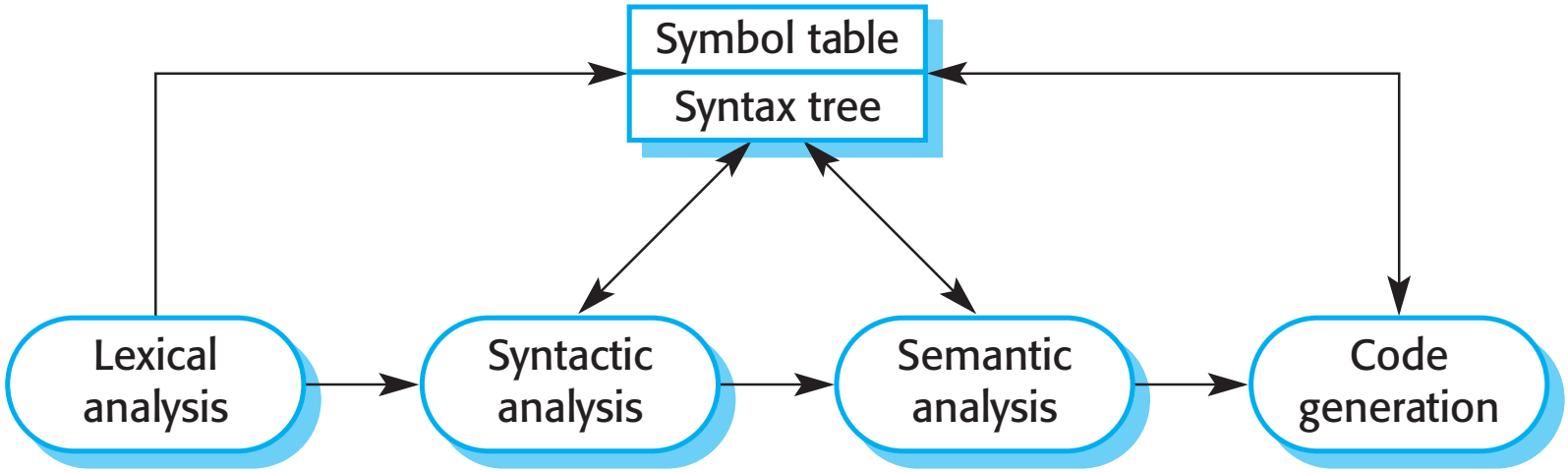
Compiler components

- 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

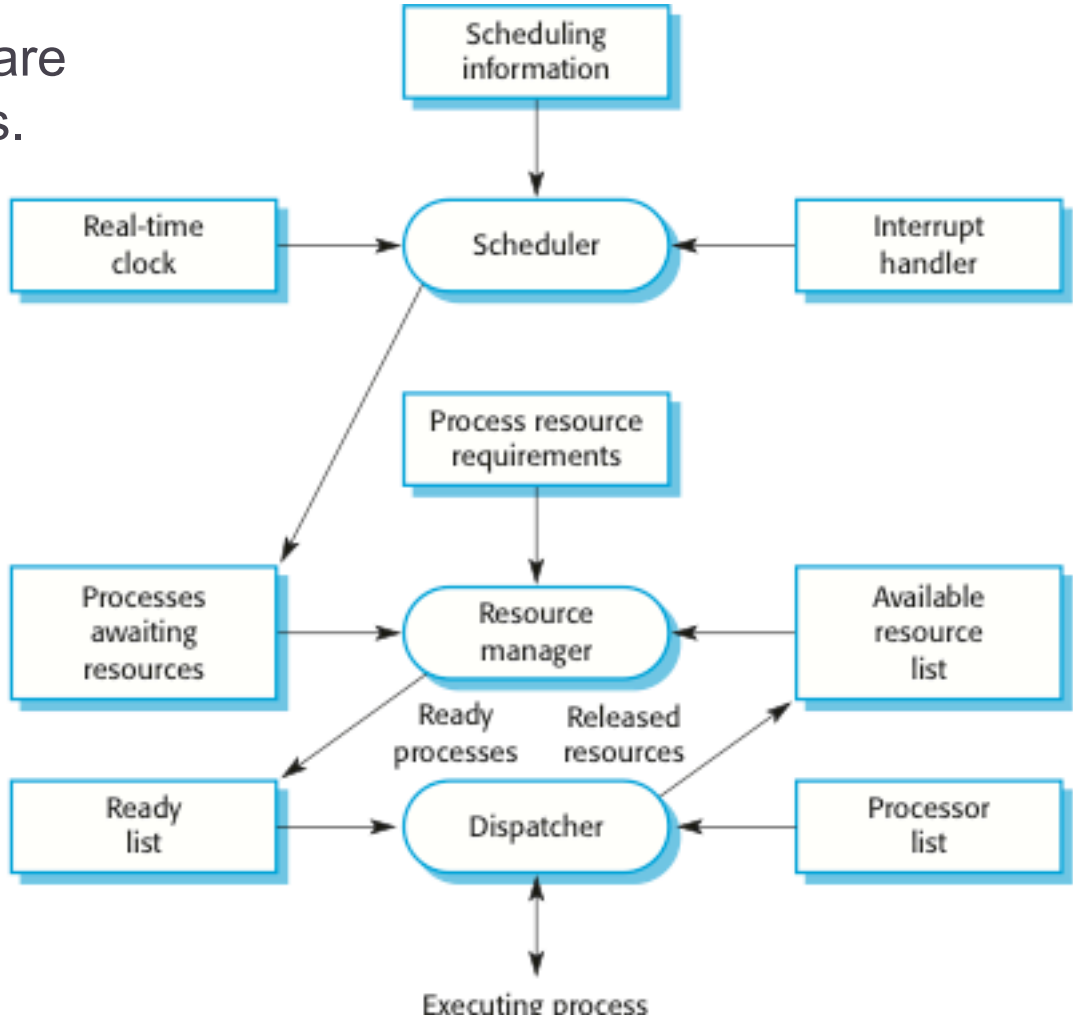


A pipe and filter compiler architecture



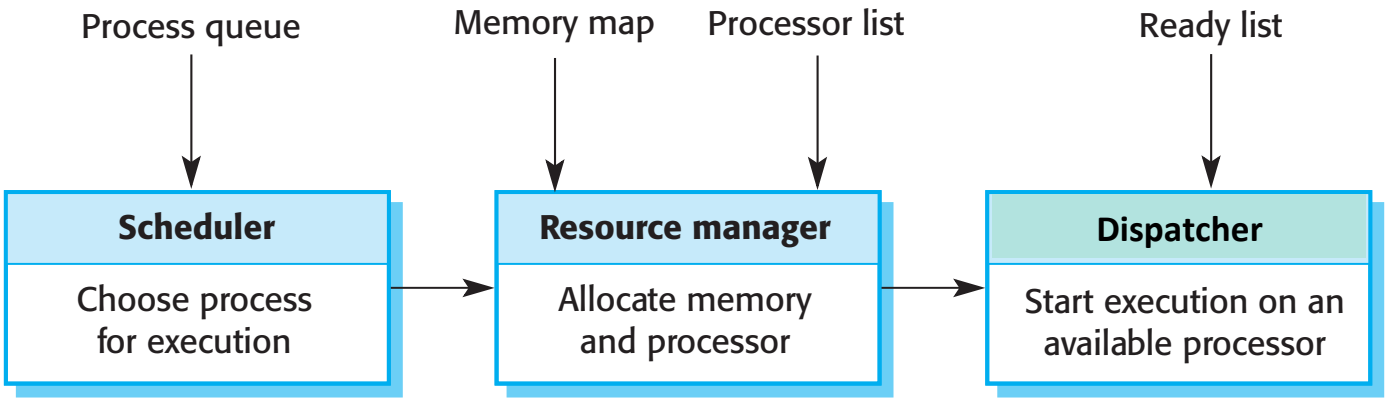
A very different example: the Real-time Operating System (RTOS)

- Real-time operating systems are specialised operating systems.
- They are responsible for process management and resource (processor and memory) allocation.
- May be based on a standard kernel which is used unchanged or modified for a particular application.
- They do not normally include facilities such as file management.

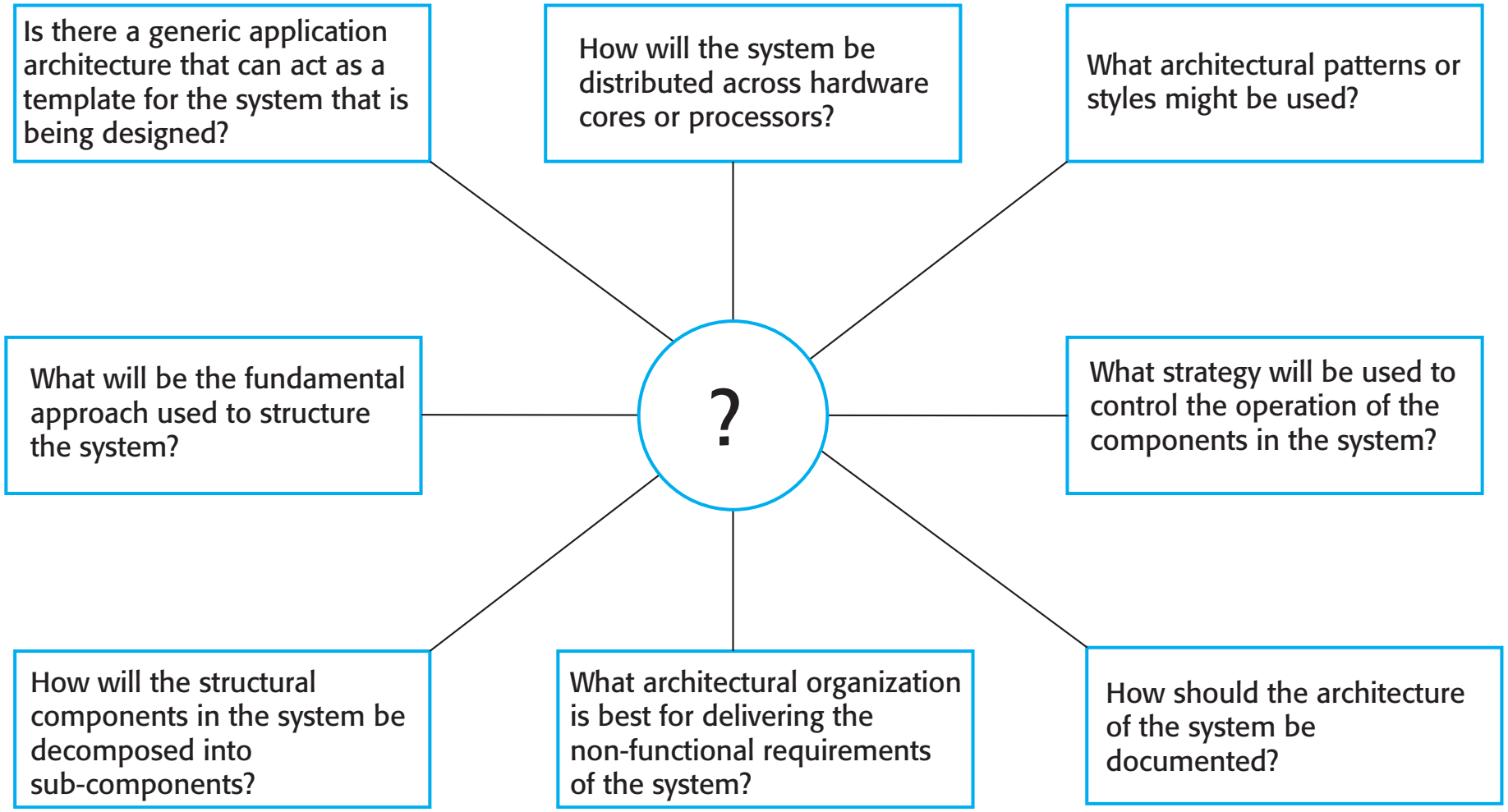


RTOS actions to start a process

- Non pre-emptive scheduling
  - Once a process has been scheduled for execution, it runs to completion or until it is blocked for some reason (e.g. waiting for I/O).
- Pre-emptive scheduling
  - The execution of an executing processes may be stopped if a higher priority process requires service.
- Scheduling algorithms
  - Round-robin
  - Rate monotonic
  - Shortest deadline first



Architectural design decisions



Classroom Activity – within your EMSS project teams

Activity:

Discuss within your teams the architectural patterns/styles presented in this week's assigned reading, and determine their applicability to the EMSS you are designing. As you do so, identify the non-functional requirements (NFRs) for the EMSS that will influence your architecture and design.

Class readout should point to patterns/styles that you may use and one or more NFR that contributed to your choice.

Submission to complete the Canvas Assignment EMSS Week 5:

Document your selection of architectural patterns/styles, noting the advantages each brings to your project. Also, submit the non-functional requirements you have identified at this point in the project.