Software Engineering

Part 3. Design

- Application Architectures
- Object-Oriented Design
- Real-Time Software Design

JUNBEOM YOO
jbyoo@konkuk.ac.kr
http://dslab.konkuk.ac.kr

Ver. 1.7

※ This lecture note is based on materials from Ian Sommerville 2006.

Chapter 13.
Application Architectures

Objectives

- To explain two fundamental models of business systems - batch processing system and transaction processing system
- To describe abstract architecture of resource management systems
- To explain how generic editors are event processing systems
- To describe the structure of language processing systems

Generic Application Architectures

- As businesses have much in common,
  - their application systems also tend to have a common architecture that reflects the application requirements.

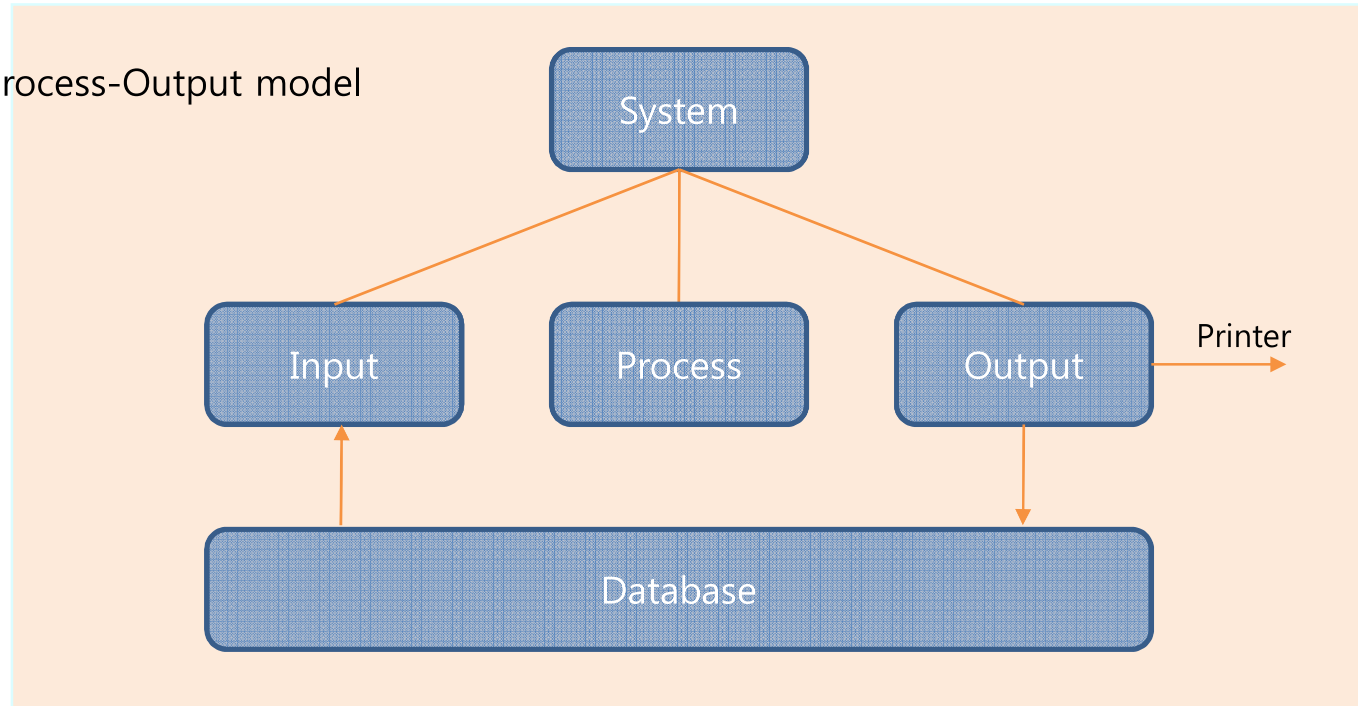
Application Types

- Application types
  1. Data processing application
     - Data driven applications
     - Process data in batches without user intervention during the processing.
     - Ex) Billing system, Payroll system
  2. Transaction processing application
     - Data-centered applications
     - Process user requests and update information in a system database.
     - Ex) E-commerce system, Reservation system
  3. Event processing system
     - System actions depend on interpreting events from the system's environment.
     - Ex) Word processor, Real-time system
  4. Language processing system
     - Users' intentions are specified in a formal language.
     - Processed and interpreted by the system.
     - Ex) Compiler, Command interpreter

1. Data Processing System

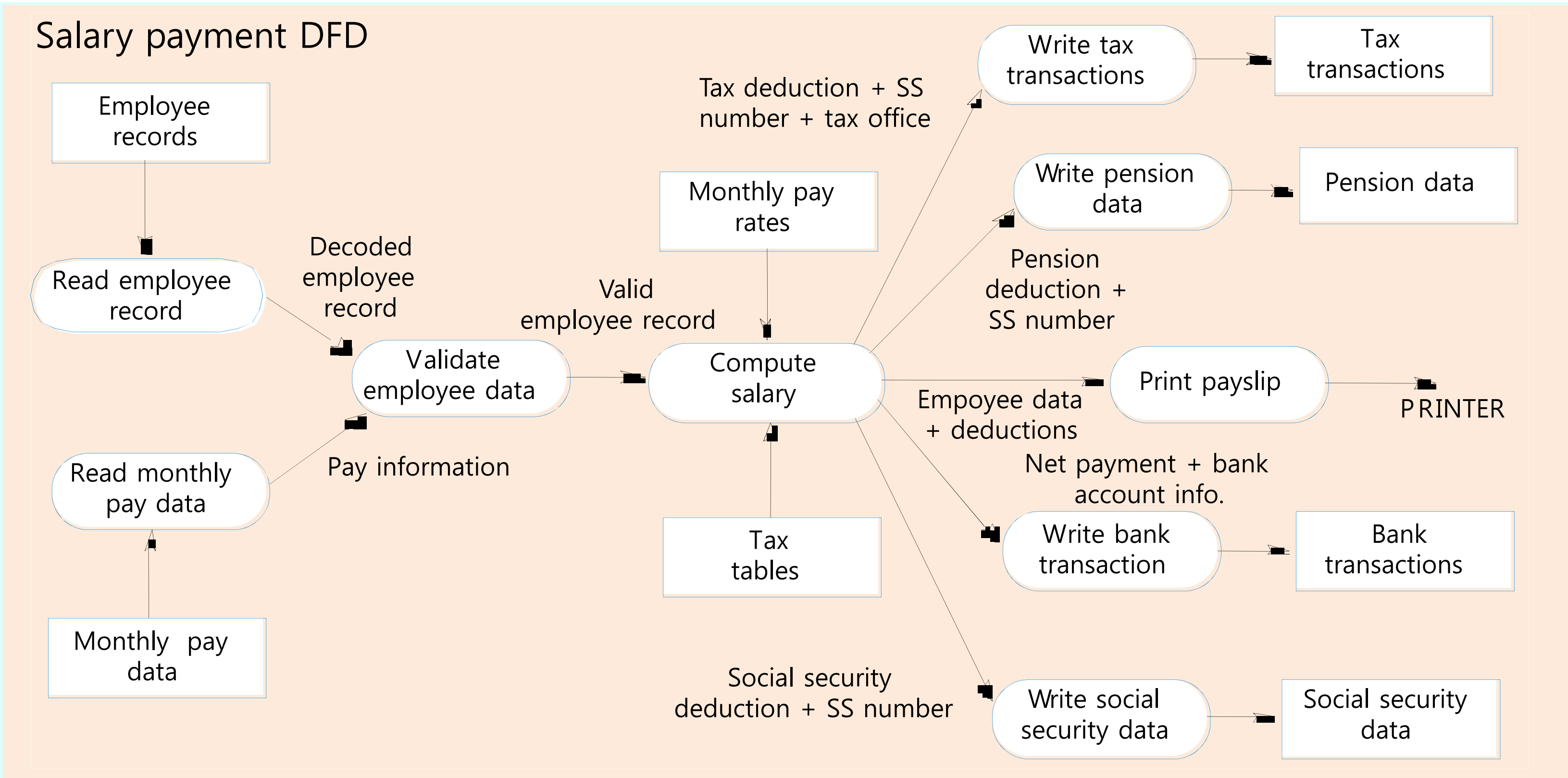
- Data-centered system, where databases used are usually orders of magnitude larger than the software itself.
  - Data is input and output in batches.
  - Have an input-process-output structure

Input-Process-Output model



Data-Flow Diagram

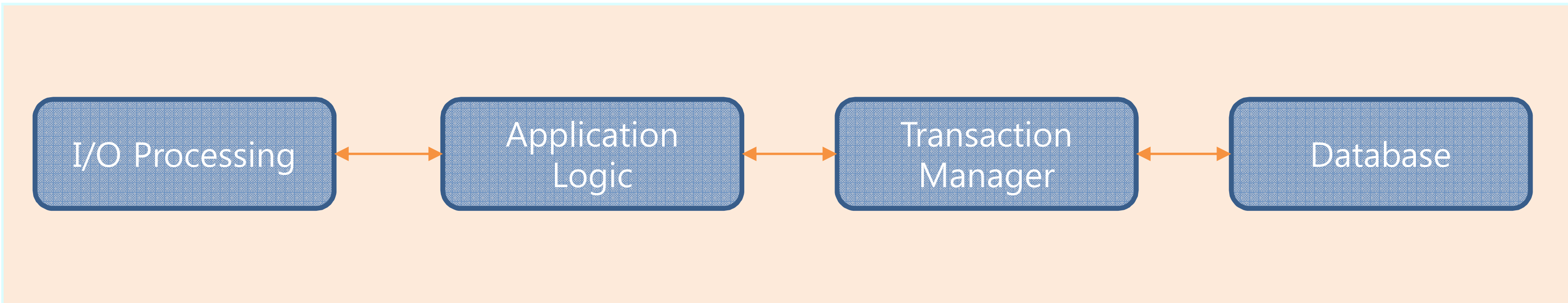
- DFD shows how data is processed as it moves through a system.
  - Round-edged rectangles : transformations
  - Arrows : data-flows
  - Rectangles : data (input/output)



2. Transaction Processing System

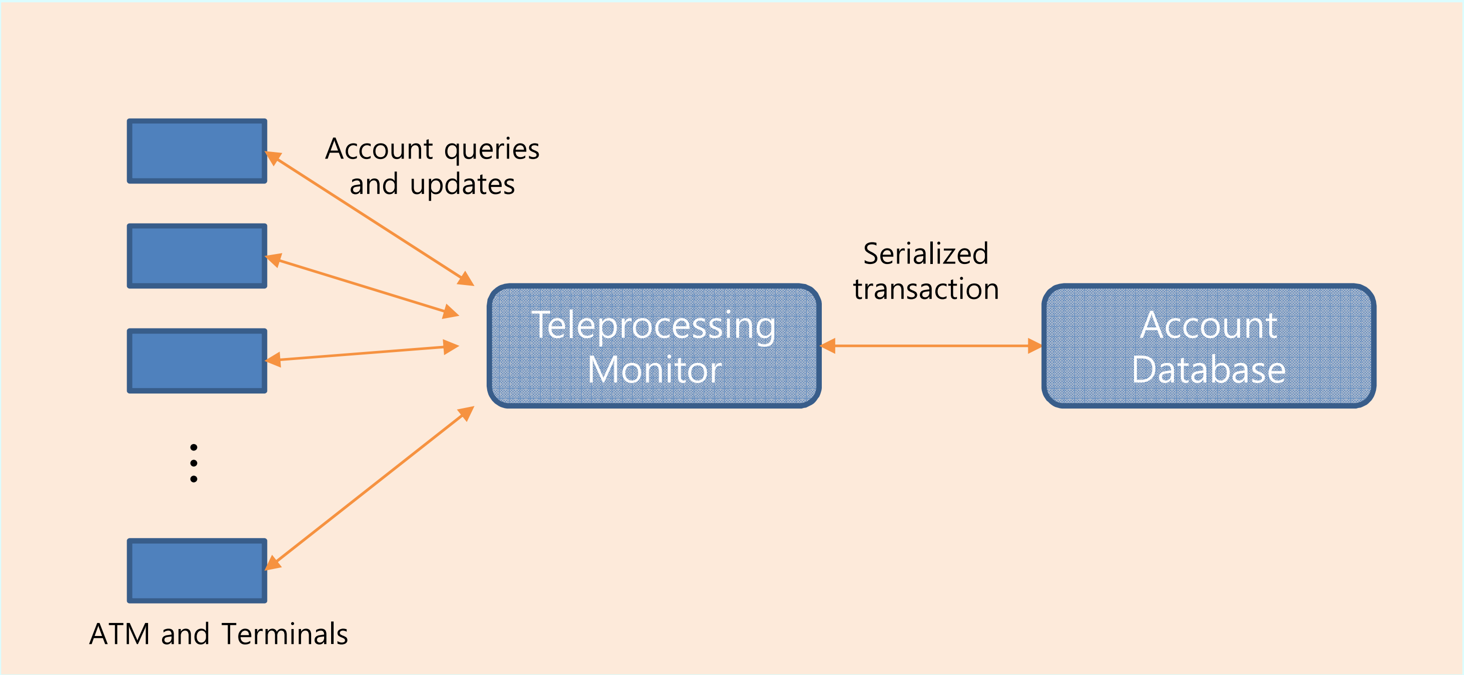
- Transaction processing systems process
  - User requests for information from a database or
  - User requests to update the database.

  - Users make asynchronous requests for service which are then processed by a transaction manager.
  - Many examples
    - Transaction processing middleware
    - Information system architecture
    - Resource allocation system
    - E-commerce system architecture



Transaction Processing Middleware

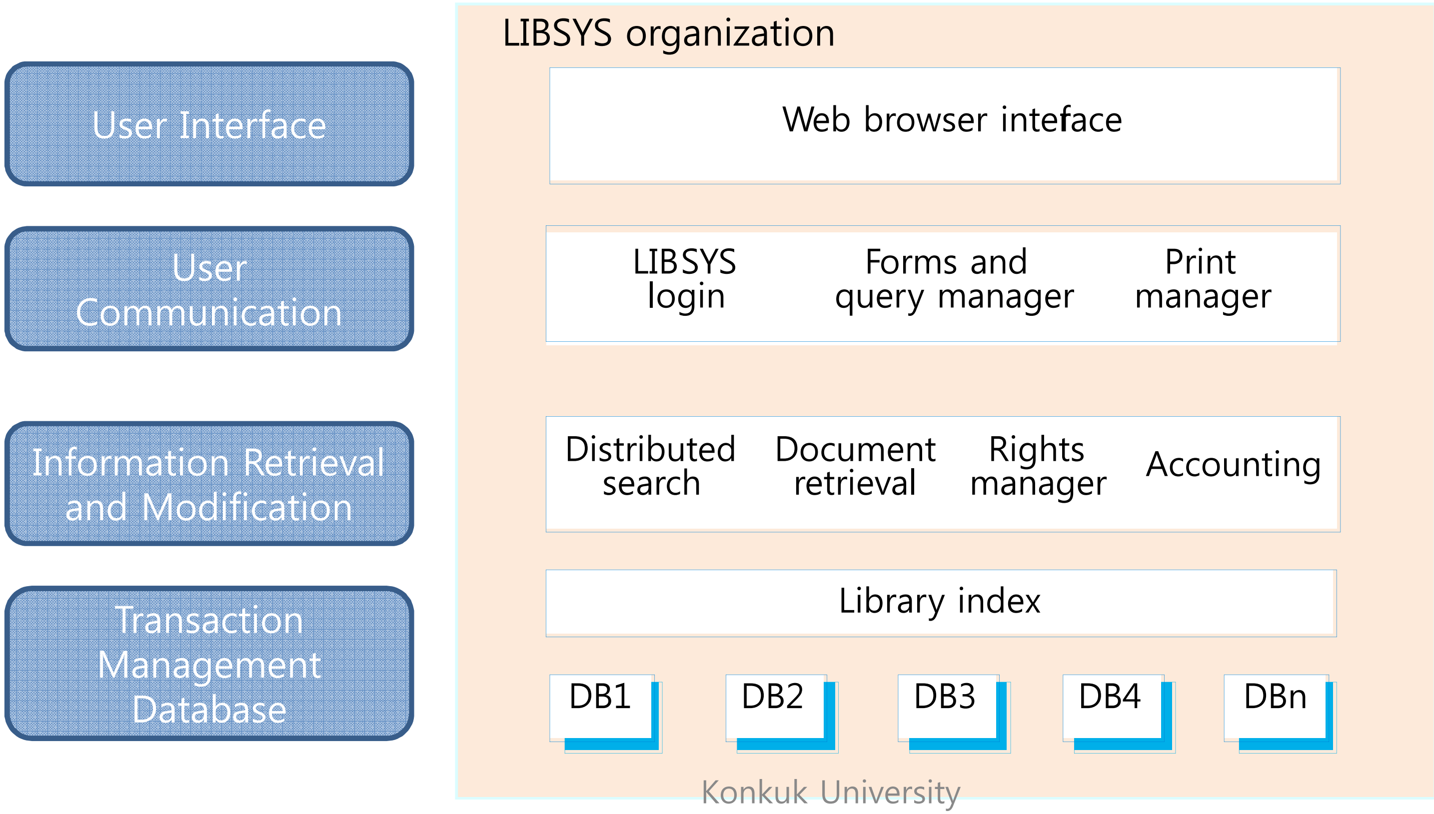
- Transaction management middleware or teleprocessing monitors
  - Handle communications with different terminal types, serializes data and sends it for processing
  - Query processing takes place in the system database and results are sent back through the transaction manager to the user's terminal.



Information System Architecture

- Information systems can be organized as a layered architecture.

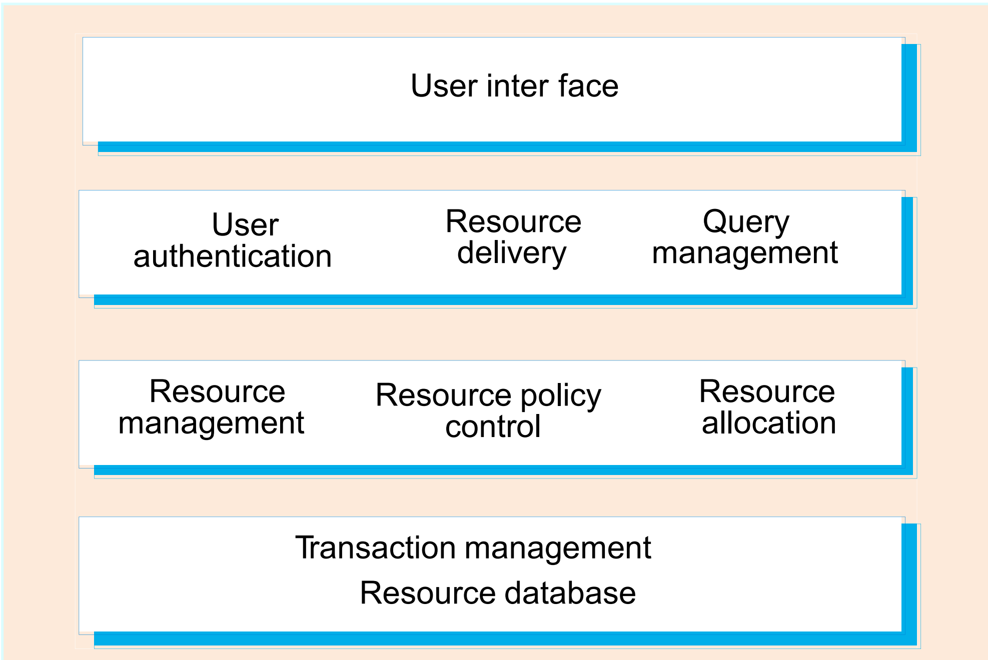
- LIBSYS example :



Resource Allocation System

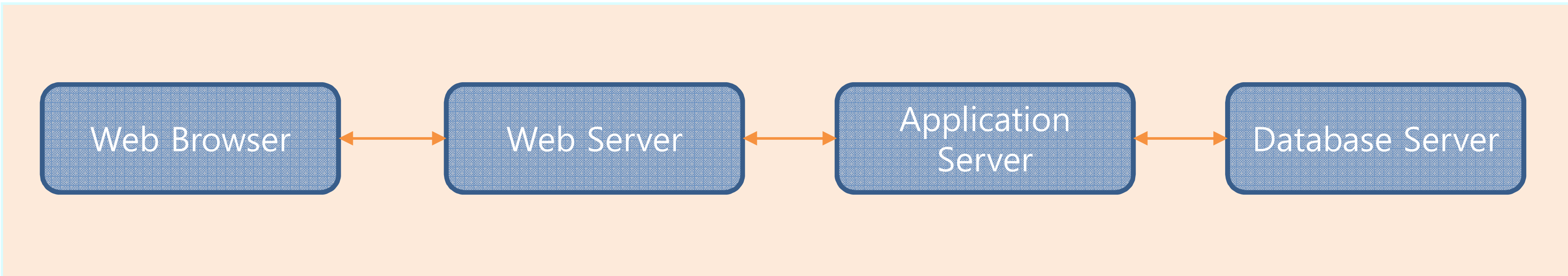
- Resource allocation systems manage fixed amount of resource and allocate them to users.
  - Timetabling system : the resource being allocated is a time period
  - Library system : the resource being managed is books for loan
  - Air traffic control system : the resource being managed is the airspace

- Layer resource allocation architecture



E-commerce System Architecture

- E-commerce systems are internet-based resource management systems
  - Accept electronic orders for goods or services
  - Organized using a multi-tier architecture with application layers associated with each tier

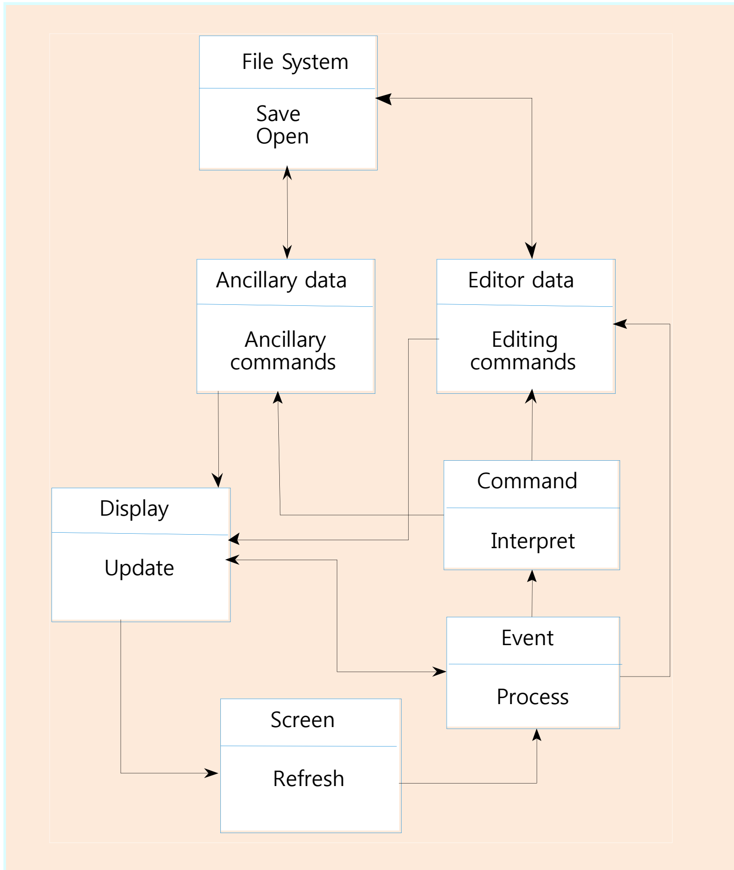


3. Event Processing Systems

- Event processing systems respond to events in the system's environment.
  - Event timing is unpredictable, so the architecture has to be organized to handle this.
  - Many common systems:
    - Word processors
    - Games
    - Real-time systems
    - Etc.

Editing System

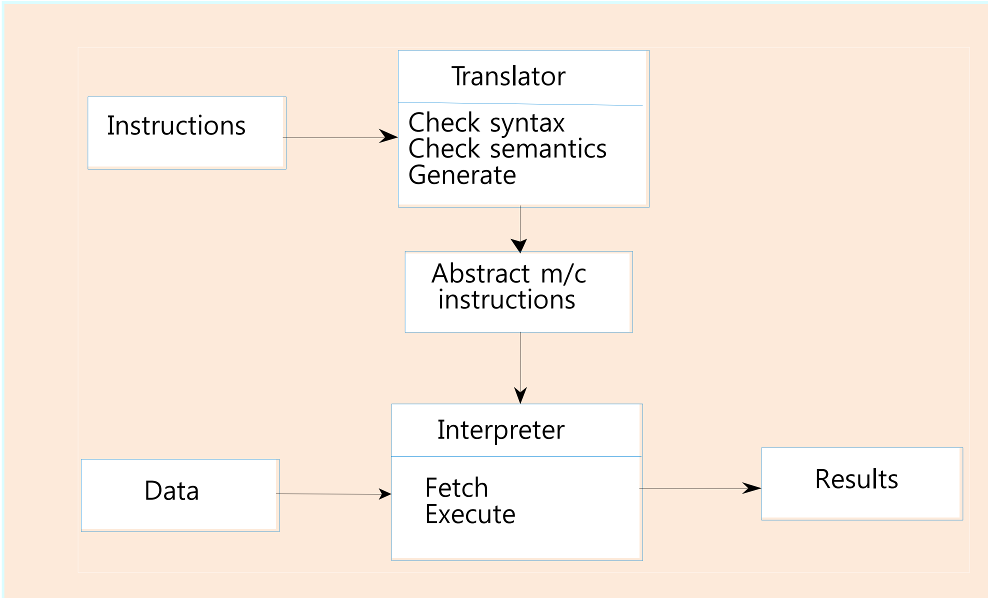
- <u>Editing systems</u> are the most common types of event processing system.
  - Single user system
  - Must provide rapid feedback to user actions
  - Organized around long transactions
  - May include recovery facilities



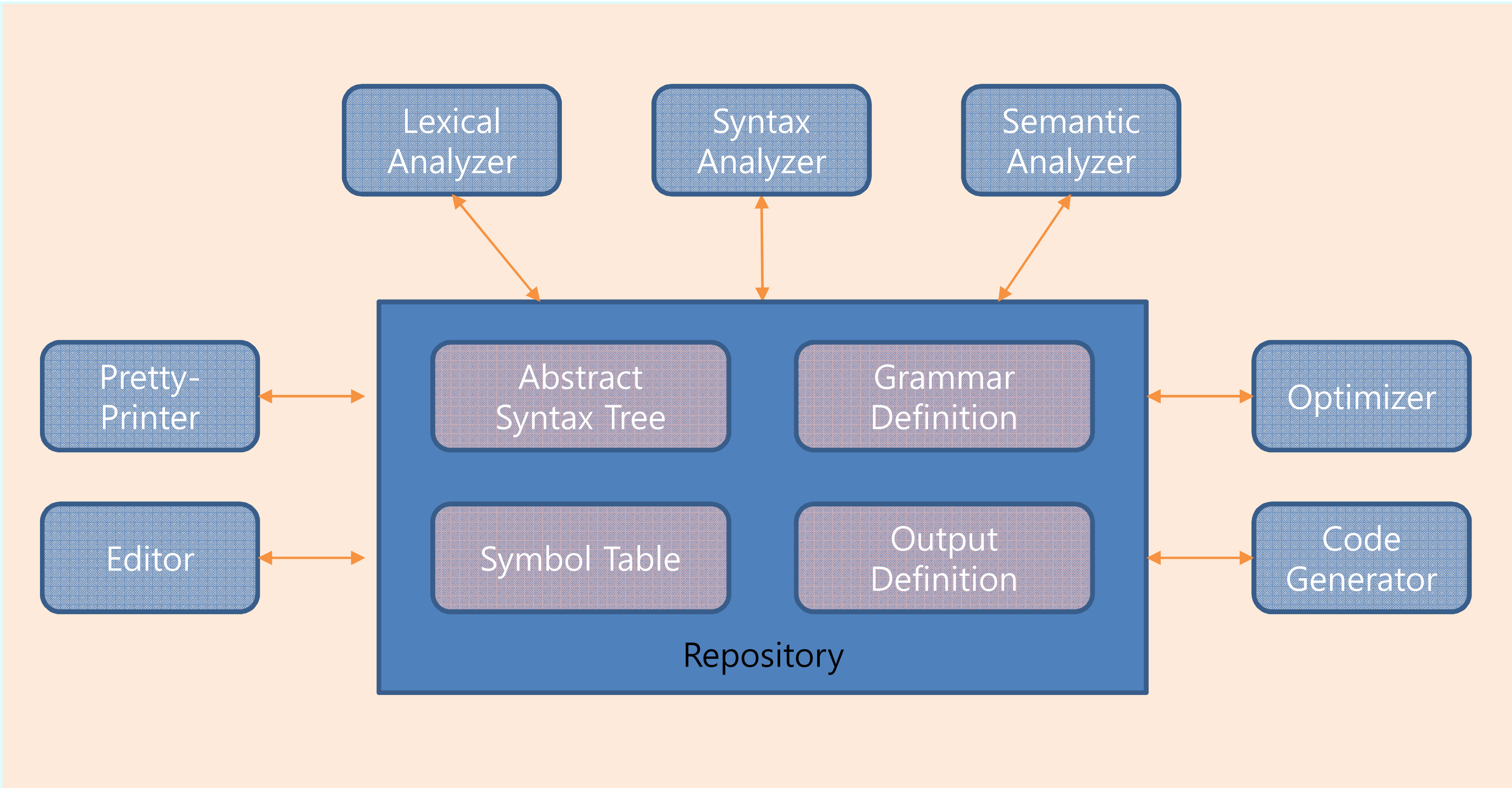
4. Language Processing System

- Language processing systems accept a natural or artificial language as input and generate some other representation of that language.
  - May include an interpreter

- Components of language processing systems
  - Lexical analyzer
  - Symbol table
  - Syntax analyzer
  - Syntax tree
  - Semantic analyzer
  - Code generator



Repository Model of Compiler



Summary

- Generic models of application architectures help us understand and compare applications.
- Important classes of application are data processing systems, transaction processing systems, event processing systems and language processing system.
- Data processing systems operate in batch mode and have an input-process-output structure.
- Transaction processing systems allow information in a database to be remotely accessed and modified by multiple users.
- Event processing systems include editors and real-time systems.
- In an editor, user interface events are detected and an in-store data structure is modified.
- Language processing systems translate texts from one language to another and may interpret the specified instructions.

Chapter 14.
Object-Oriented Design

Objectives

- To explain how a software design may be represented as a set of interacting objects that manage their own states and operations
- To describe the activities in object-oriented design process
- To introduce various models that can be used to describe an object-oriented design
- To show how the UML may be used to represent these models

Object-Oriented Development

- Object-oriented analysis, design and programming are related but distinct.
  - OOA : concerned with developing an object model of the application domain
  - OOD : concerned with developing an object-oriented system model to implement requirements
  - OOP : concerned with realizing an OOD using an OO programming language such as Java or C++

- Characteristics of OOD
  - Objects are abstractions of real-world or system entities.
  - Objects encapsulate state and representation information.
  - System functionality is expressed in terms of object services.
  - Shared data areas are eliminated.
  - Objects communicate by message passing.
  - Objects may be distributed and may execute sequentially or in parallel.

Advantages of OOD

- Easier maintenance
  - Objects may be understood as stand-alone entities.
- Objects are potentially reusable components.
- Easy to implement for some systems
  - There may be an obvious mapping from real world entities to system objects.

Objects and Object Classes

- Objects are entities in software system
  - Represent instances of real-world and system entities
- Object classes are templates for objects
  - Used to create objects
  - May inherit attributes and services from other object classes

An object is an entity that has a state and a defined set of operations which operate on that state. The state is represented as a set of object attributes. The operations associated with the object provide services to other objects (clients) which request these services when some computation is required.

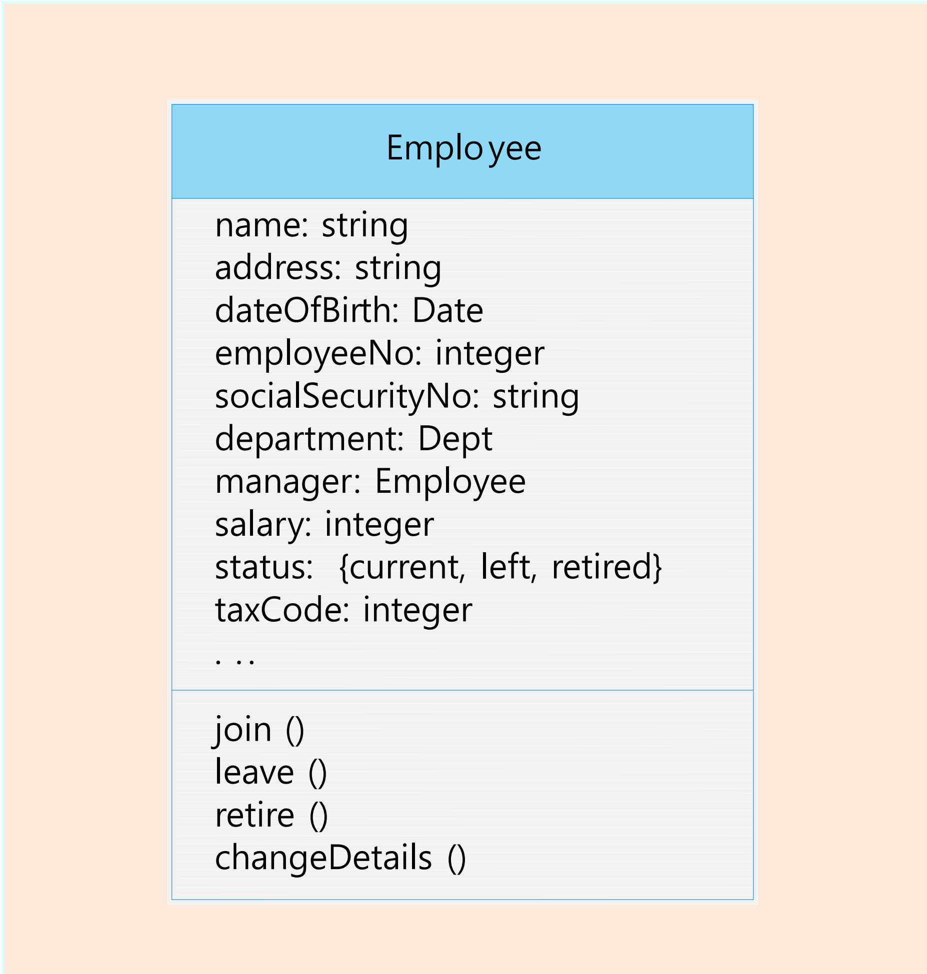
Objects are created according to some object class definition. An object class definition serves as a template for objects. It includes declarations of all the attributes and services which should be associated with an object of that class.

Unified Modelling Language

- Several different notations for describing object-oriented designs were proposed in the 1980s and 1990s.

- Unified Modelling Language(UML) is an integration of these.
  - Describes notations for a number of different models that may be produced during OO analysis and design
  - A de facto standard for OO modelling

Class Example: Employee Object



Object Communication

- Conceptually, objects communicate by message passing.
- Messages
  - Name of service requested by calling object
  - Copies of information required to execute the service

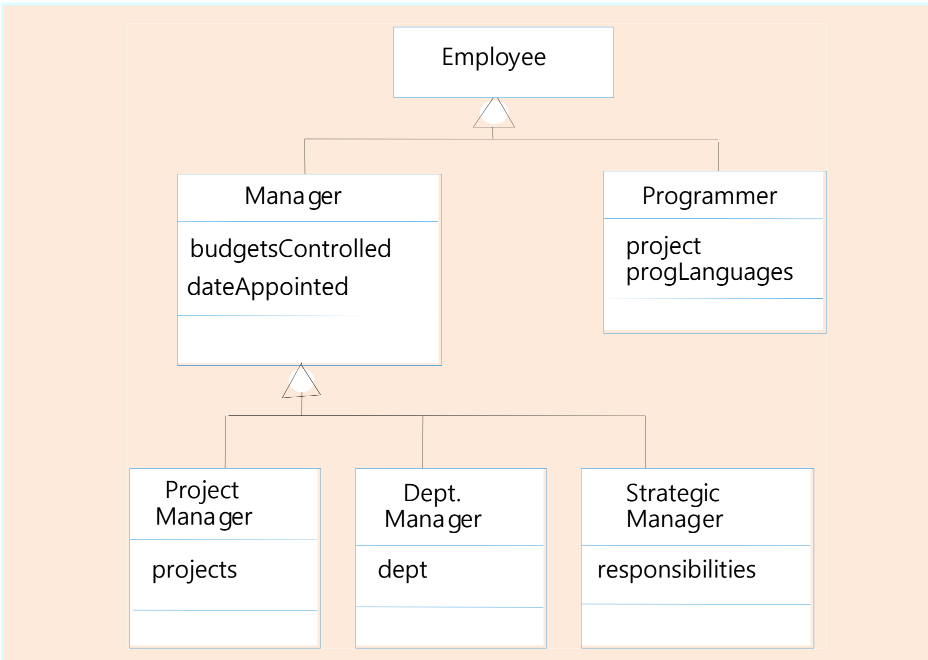
- In practice, messages are often implemented by procedure calls.
  - Name = procedure name
  - Information = parameter list

```
// Call a method associated with a buffer object that returns the next value in the buffer
        v = circularBuffer.Get ( ) ;

// Call the method associated with a thermostat object that sets the temperature
// to be maintained
        thermostat.setTemp (20) ;
```

Generalization and Inheritance

- Classes may be arranged in a class hierarchy, where one class (a super-class) is a generalization of one or more other classes (sub-classes).
  - A sub-class inherits the attributes and operations from its super class and may add new methods or attributes of its own.
  - Generalization in the UML is implemented as an inheritance in OO programming languages.



Features of Inheritance

- Advantages:
  - Abstraction mechanism : may be used to classify entities.
  - Reuse mechanism at both the design and the programming level.

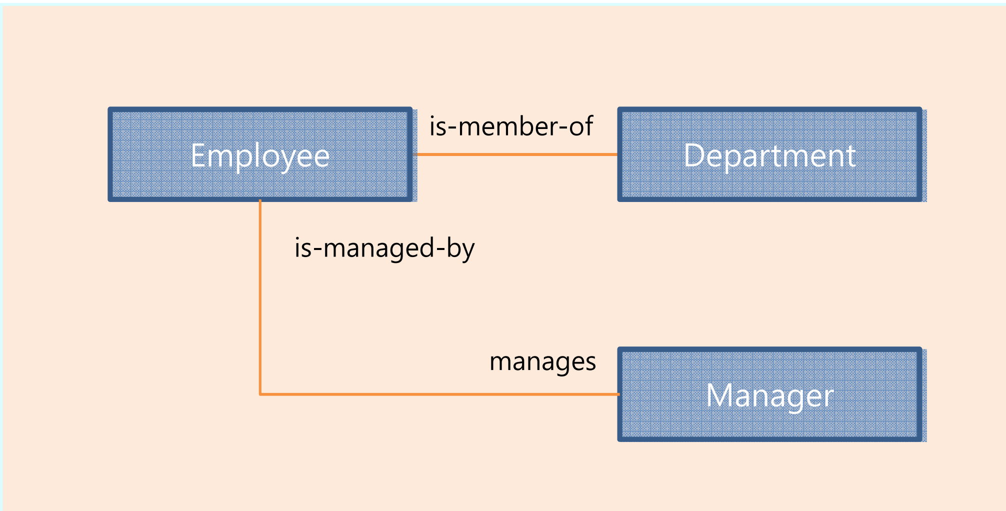
  - Inheritance graph is a source of organizational knowledge about domains and systems.

- Problems:
  - Object classes are not self-contained. They cannot be understood without reference to their super-classes.
  - Designers have a tendency to reuse the inheritance graph created during analysis. It may lead to significant inefficiency.
  - Inheritance graphs of analysis, design and implementation have different functions and should be separately maintained.

UML Association

- Objects and object classes participate in relationships with other objects and object classes.

- In the UML, a generalized relationship is indicated by an association.
  - May be annotated with information that describes the association
    - May indicate that an attribute of an object is an associated object
    - May indicate that a method relies on an associated object



Concurrent Object

- The nature of objects :
  - Self-contained entities are suitable for concurrent implementation.
  - Message-passing model of object communication can be implemented directly if objects are running on separate processors in a distributed system.

- Servers
  - The object is implemented as a parallel process (server) with entry points corresponding to object operations.
  - If no calls are made to it, the object suspends itself and waits for further requests for service.

- Active objects
  - Objects are implemented as parallel processes and the internal object state may be changed by the object itself and not simply by external calls.
  - Thread in Java is a simple construct for implementing concurrent objects.

Java Thread

- Thread in Java is a simple construct for implementing concurrent objects.
  - Threads must include a method called run( ) and this is started up by the Java run-time system.
  - Active objects typically include an infinite loop so that they are always carrying out the computation.

Object-Oriented Design Process

- Structured design processes involve developing a number of different system models.
  - Require a lot of effort for development and maintenance of these models
  - For small systems, it may not be cost-effective.
  - However, for large systems developed by different groups, design models are an essential communication mechanism.

- <u>Common key activities</u> for OOD processes
  1. Define the context and modes of use of the system
  2. Design the system architecture (Architectural design)
  3. Identify the principal system objects (Object identification)
  4. Develop design models
  5. Specify object interfaces (Object interface specification)

Example: Weather Mapping System Description

A weather mapping system is required to generate weather maps on a regular basis using data collected from remote, unattended weather stations and other data sources such as weather observers, balloons and satellites. Weather stations transmit their data to the area computer in response to a request from that machine.

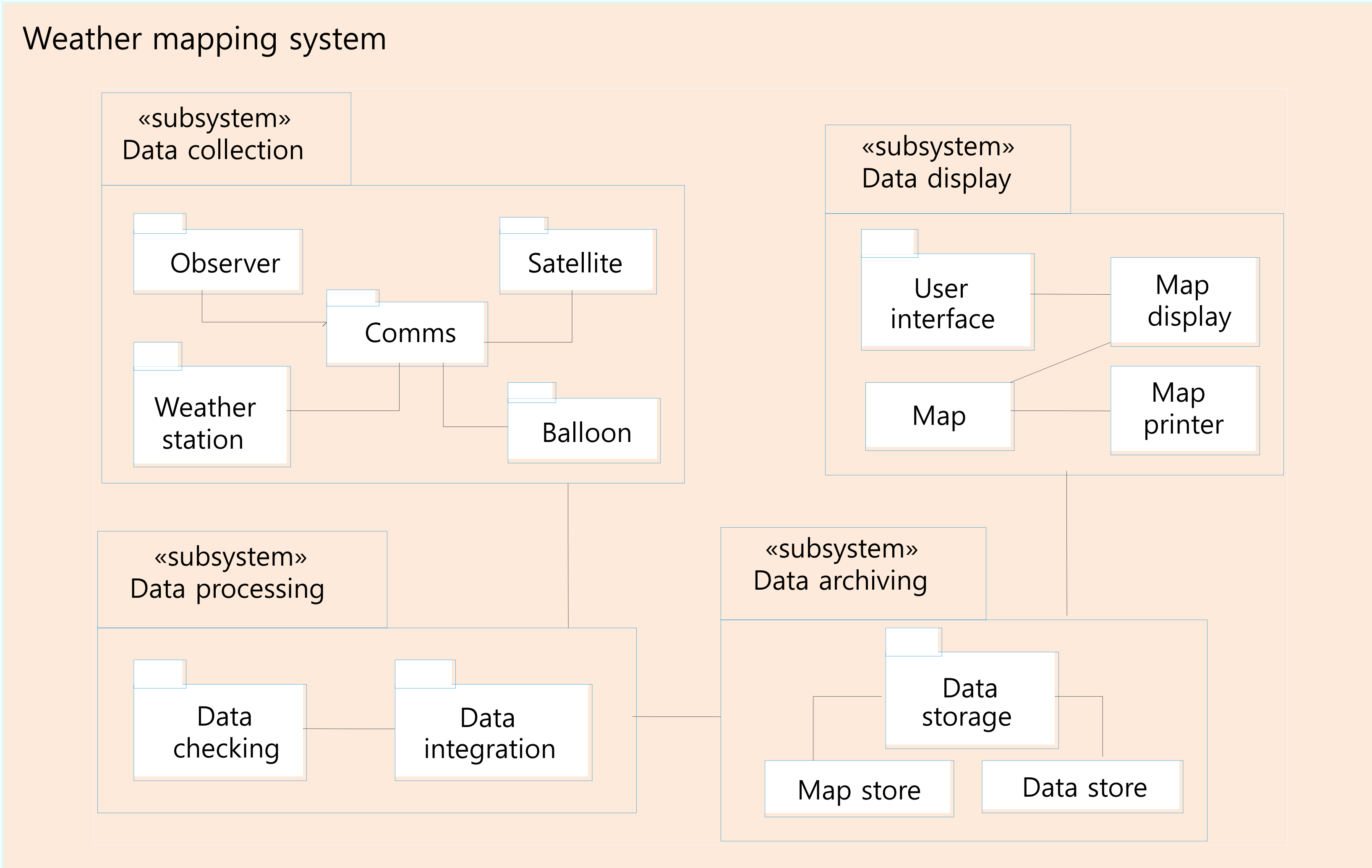
The area computer system validates the collected data and integrates it with the data from different sources. The integrated data is archived and, using data from this archive and a digitised map database a set of local weather maps is created. Maps may be printed for distribution on a special-purpose map printer or may be displayed in a number of different formats.

1. System Context and Models of System Use

- Develop an understanding of the relationships between the software being designed and its external environment

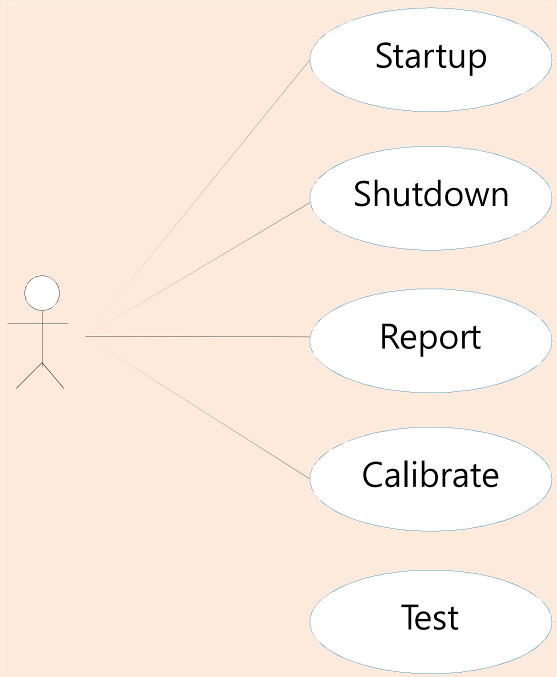
- System context
  - A static model that describes other systems in the environment
  - Use a subsystem model to show other systems
- Model of system use
  - A dynamic model that describes how the system interacts with its environment
  - Use use-cases to show interactions

Subsystem Model



Use-Case Model

Weather station use-case



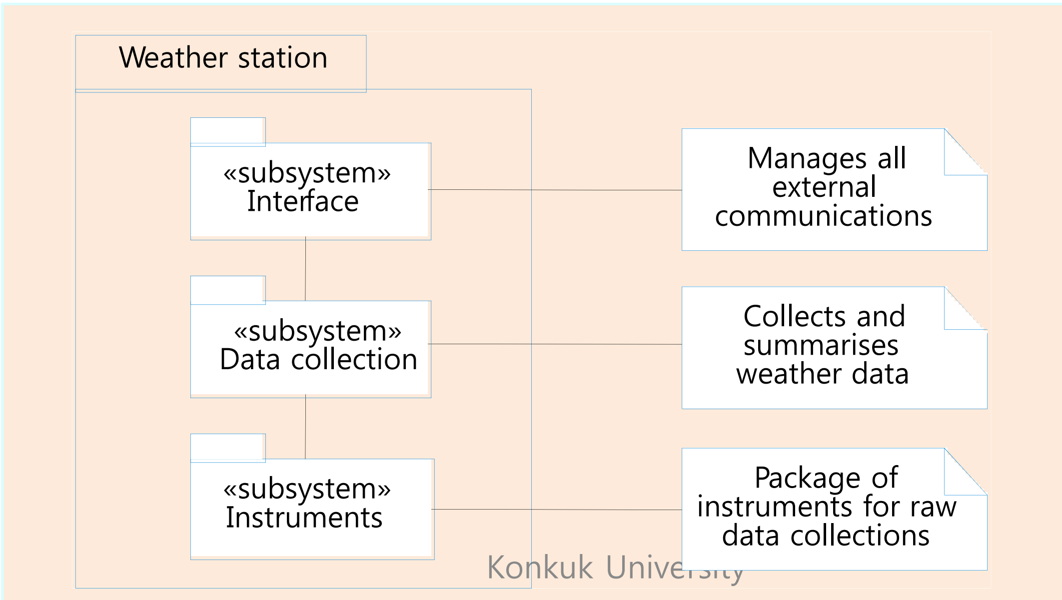
Use-case description

System	Weather station
Use-case	Report
Actors	Weather data collection system, Weather station
Data	The weather station sends a summary of the weather data that has been collected from the instruments in the collection period to the weather data collection system. The data sent are the maximum minimum and average ground and air temperatures, the maximum, minimum and average air pressures, the maximum, minimum and average wind speeds, the total rainfall and the wind direction as sampled at 5 minute intervals.
Stimulus	The weather data collection system establishes a modem link with the weather station and requests transmission of the data.
Response	The summarised data is sent to the weather data collection system
Comments	Weather stations are usually asked to report once per hour but this frequency may differ from one station to the other and may be modified in future.

2. Architectural Design

- Design the system architecture using the understanding about the interactions between the system and its environment.

- A layered architecture is appropriate for the weather station
  - Interface layer for handling communications
  - Data collection layer for managing instruments
  - Instruments layer for collecting data

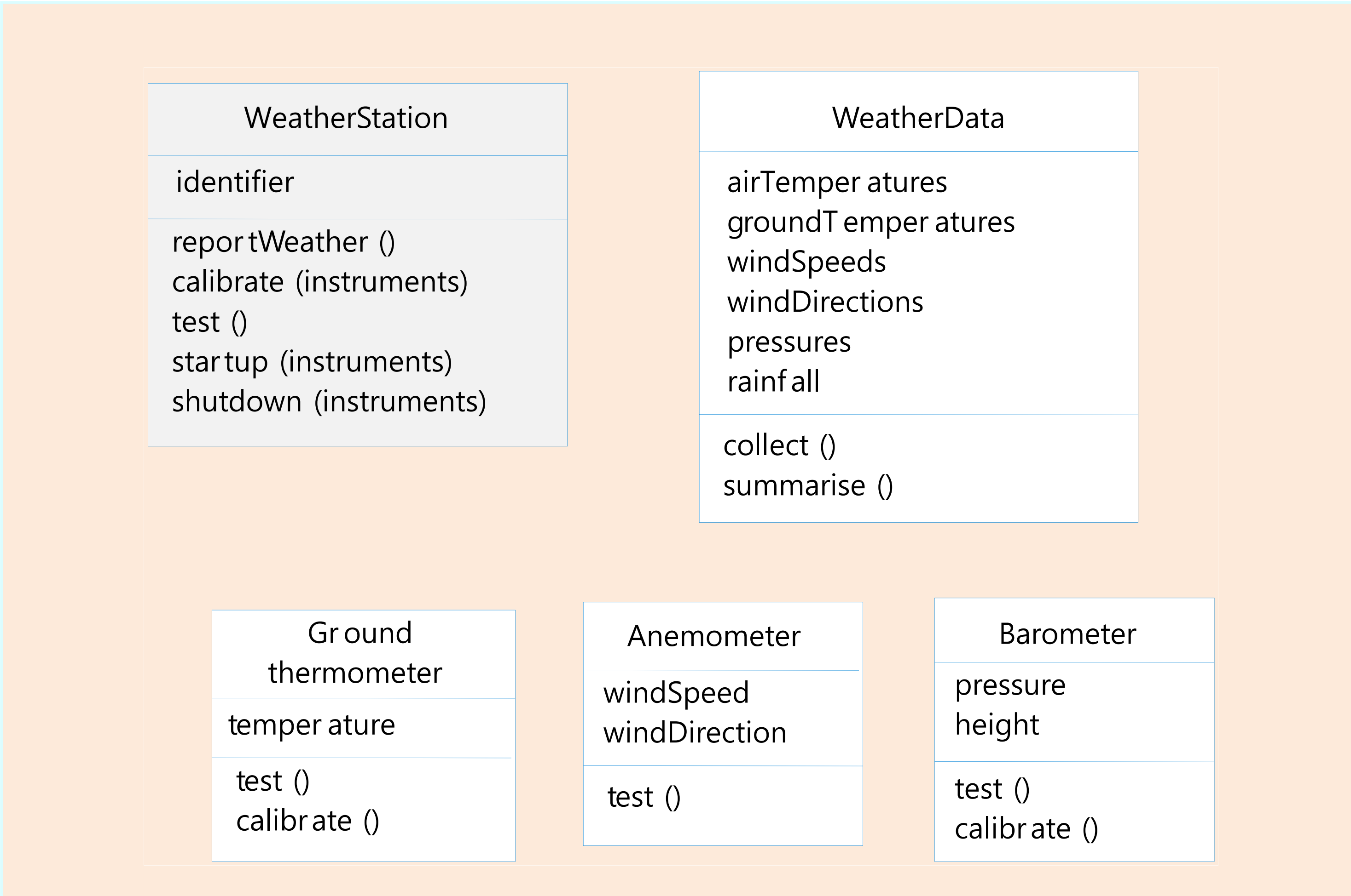


3. Object Identification

- Identifying objects (or object classes) is the most difficult part of object oriented design.
  - No 'magic formula' for object identification.
  - Relies on the skill, experience and domain knowledge of system designers
  - An iterative process

- Approaches to object identification:
  - Use a grammatical approach based on a natural language description of the system (used in Hood OOD method)
  - Based on the identification on tangible things in the application domain
  - Use a behavioural approach and identify objects based on what participates in what behaviour
  - Use a scenario-based analysis. The objects, attributes and methods in each scenario are identified.

Weather Station Object Classes



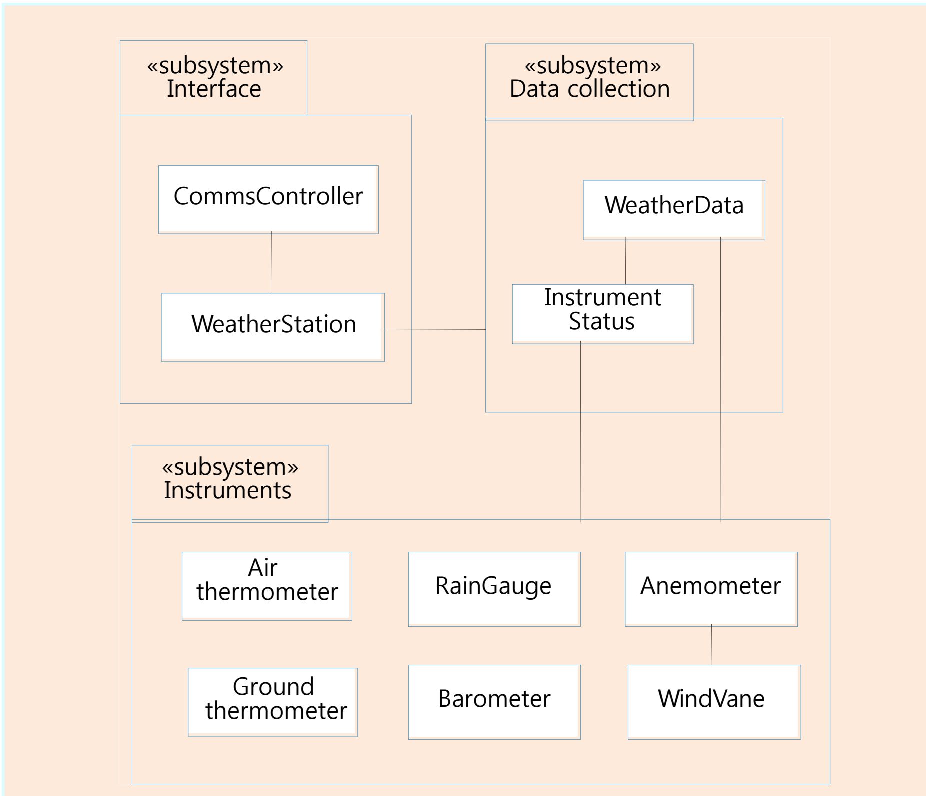
4. Developing Design Model

- Design models show the objects, object classes and relationships between these entities.
  - Static models describe the static structure of the system in terms of object classes and relationships.
  - Dynamic models describe the dynamic interactions between objects.

- Examples of design models:
  - Sub-system model : shows logical groupings of objects into coherent subsystems
  - Sequence model : shows the sequence of object interactions
  - State machine model : show how individual objects change their state in response to events.
  - Other models include use-case models, aggregation models, generalisation models, etc.

Subsystem Model

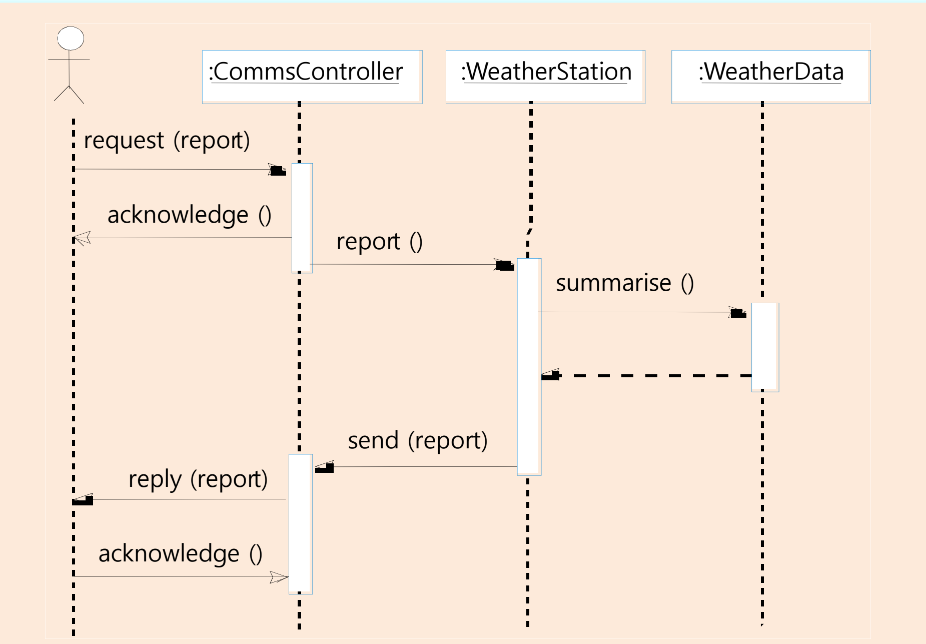
- Show how the design is organized into logically related groups of objects.
  - A logical model
  - The actual organization of objects may be different.
  - In the UML, these are shown using packages



Sequence Model

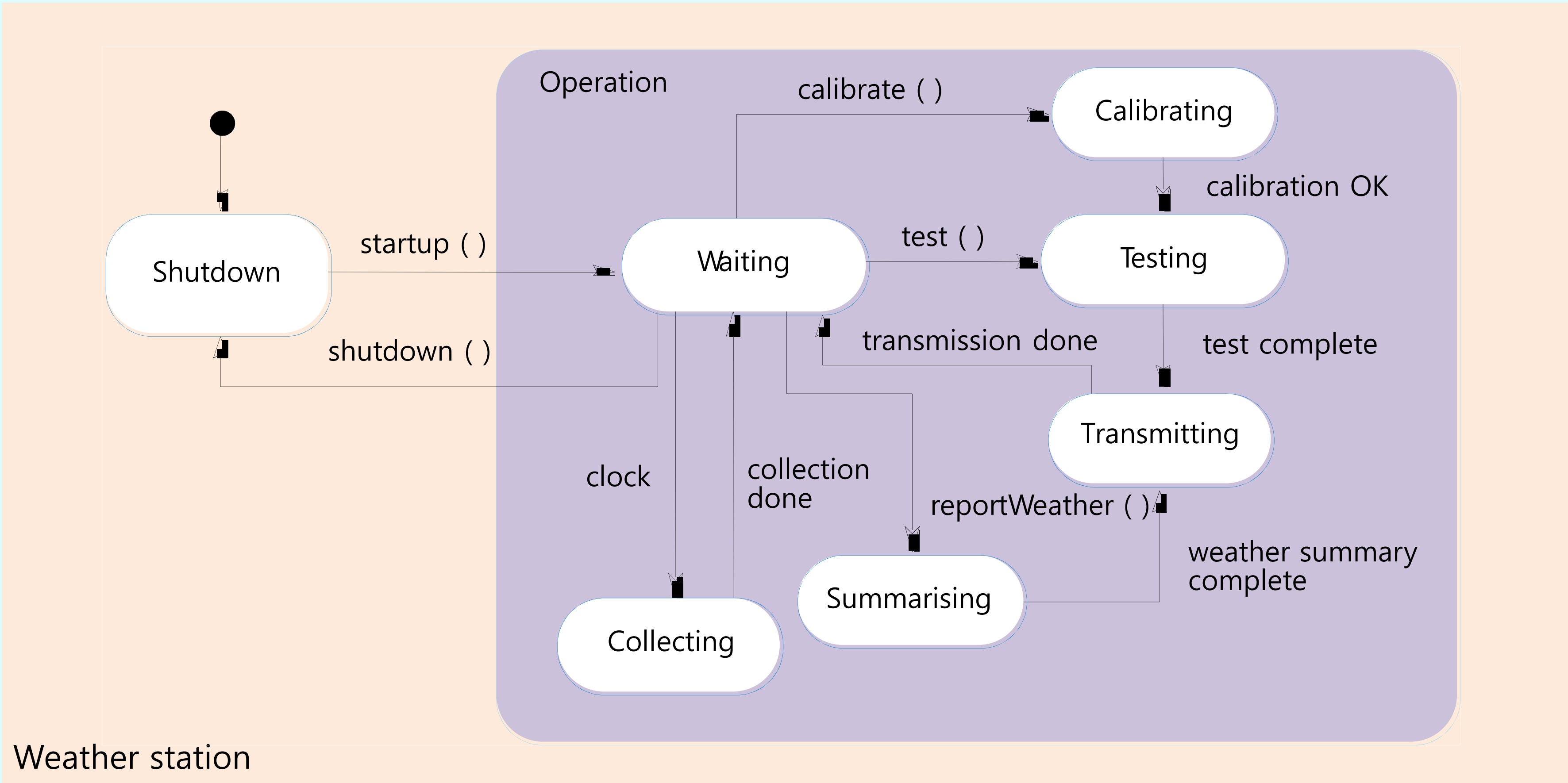
- Show the sequence of object interactions that take place
  - Objects are arranged horizontally across the top.
  - Time is represented vertically, so models are read top to bottom.
  - Interactions are represented by labelled arrows.
  - Different styles of arrow represent different types of interaction.
  - Thin rectangle in an object lifeline represents the time when the object is the controlling object in the system.

Data collection



State Machine Model: Statecharts

- Show how objects respond to different service requests and the state transitions triggered by these requests



5. Object Interface Specification

- Object interfaces specification make the design of objects and other components performed in parallel.
  - Objects may have several interfaces (viewpoints).
  - The UML uses class diagram for interface specification

```
interface WeatherStation {

    public void WeatherStation () ;

    public void startup () ;
    public void startup (Instrument i) ;

    public void shutdown () ;
    public void shutdown (Instrument i) ;

    public void reportWeather ( ) ;

    public void test () ;
    public void test ( Instrument i ) ;

    public void calibrate ( Instrument i) ;

    public int getID () ;

} //WeatherStation
```

Summary

- OOD is an approach to design so that design components have their own private state and operations.
- Objects should have constructor and inspection operations. They provide services to other objects.
- Objects may be implemented sequentially or concurrently.
- The Unified Modelling Language provides different notations for defining different object models.
- A range of different models may be produced during an object-oriented design process. These include static and dynamic system models.
- Object interfaces should be defined precisely using a programming language like Java.

Chapter 15.
Real-Time Software Design

Objectives

- To explain the concept of a real-time system and why these systems are usually implemented as concurrent processes
- To describe a design process for real-time systems
- To explain the role of real-time operating systems
- To introduce generic process architectures for monitoring and control and data acquisition systems

Real-Time systems

- Systems which monitor and control their environment
- Inevitably associated with hardware devices
  - Sensors : collect data from the system environment
  - Actuators : change the system's environment (in some way)
- Time is critical.
  - Real-time systems MUST respond within specified times.

Definition

- Real-time system is a software system where the correct functioning of the system depends on
  - the results produced by the system and
  - the time at which these results are produced

- Soft real-time system
  - Operation is degraded if results are not produced according to the specified timing requirements.

- Hard real-time system
  - Operation is incorrect if results are not produced according to the timing specification.

Stimulus/Response Systems

- Given a stimulus, the system must produce a response within a specified time.

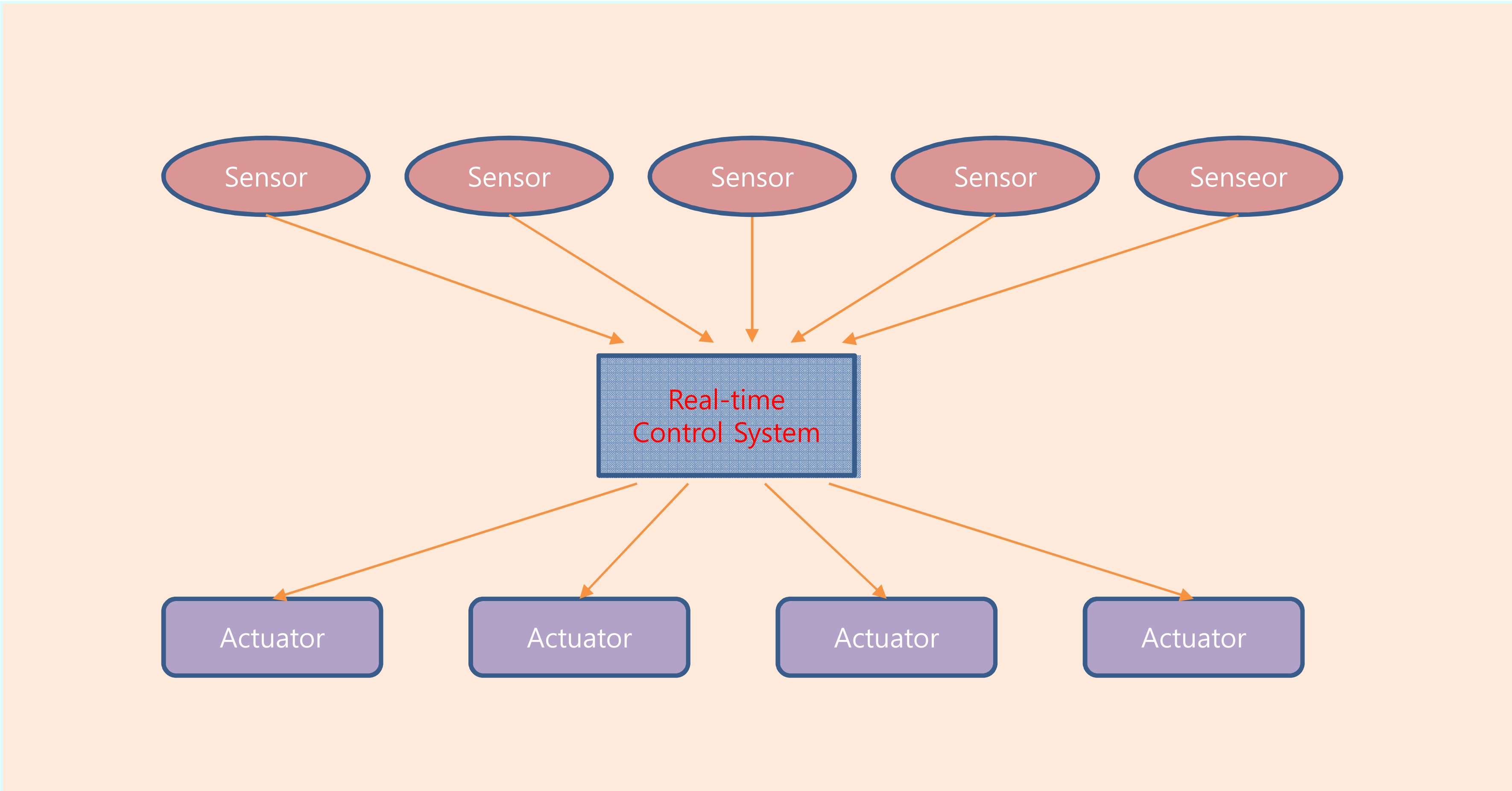
- Periodic stimuli
  - Stimuli which occur at predictable time intervals
  - Example: a temperature sensor may be polled 10 times per second.

- Aperiodic stimuli
  - Stimuli which occur at unpredictable times
  - Example: a system power failure may trigger an interrupt which must be processed by the system.

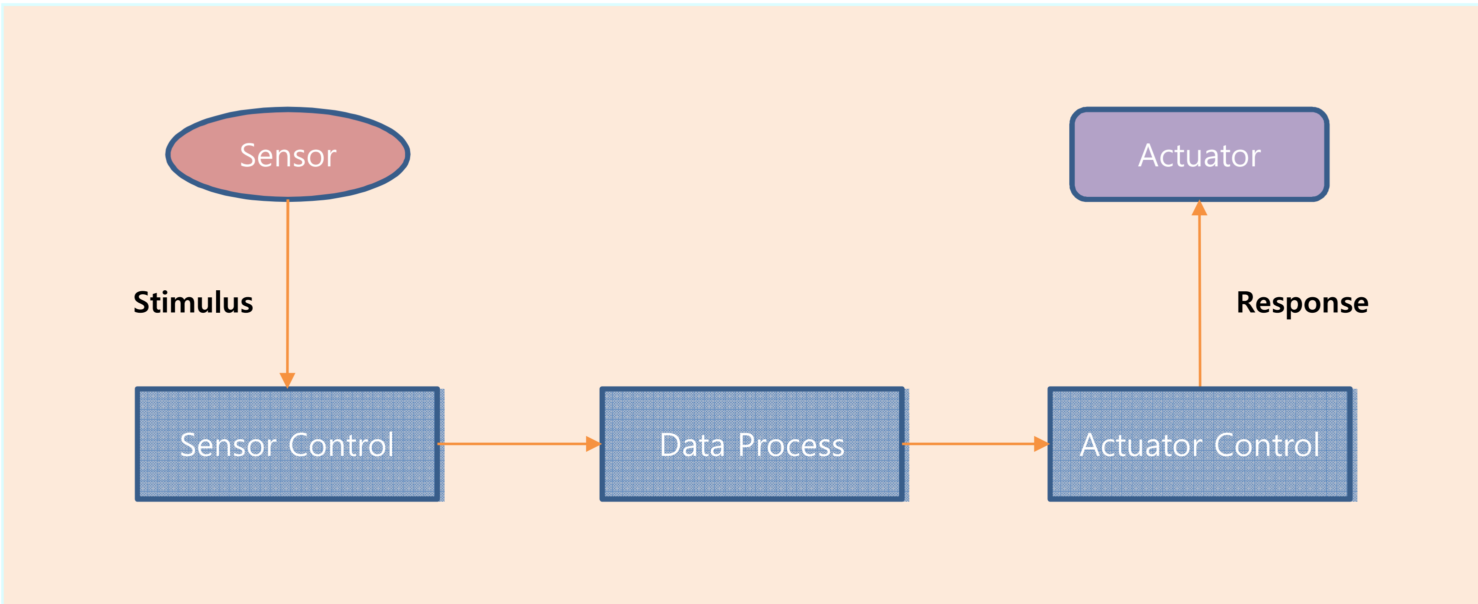
Architectural Considerations

- Because of the need to respond to timing demands made by different stimuli/responses, the system architecture must allow for fast switching between stimulus handlers.
- Timing demands of different stimuli are different so a simple sequential loop is not usually adequate.
- Real-time systems are therefore usually designed as cooperating processes with a real-time executive controlling these processes.
  - Cooperating processes
  - One real-time executive

A Real-Time System Model



Sensor/Actuator Processes



System Elements

- Sensor control processes
  - Collect information from sensors
  - May buffer collected information in response to a sensor stimulus.
- Data processor
  - Carries out processing of collected information
  - Computes the system response
- Actuator control processes
  - Generates control signals for the actuators

Real-Time Programming

- Hard-real time systems may have to programmed in assembly language to ensure that deadlines are met.
  - Languages such as C allow efficient programs to be written, but do not have constructs to support concurrency or shared resource management.
  - Java supports lightweight concurrency (threads and synchronized methods) and can be used for some soft real-time systems.

- Real-time versions of Java are now available addressing problems like
  - Not possible to specify thread execution time
  - Different timing in different virtual machines
  - Uncontrollable garbage collection
  - Not possible to discover queue sizes for shared resources
  - Not possible to access system hardware
  - Not possible to do space or timing analysis

System Design

- Design both the hardware and the software associated with system
  - Partition functions to either hardware or software
  - Design decisions should be made on the basis on non-functional system requirements.

  - Hardware delivers better performance but potentially longer development and less scope for change.

Real-Time Systems Design Process

1. Identify the stimuli to be processed and the required responses to these stimuli.
2. For each stimulus and response, identify the timing constraints.
3. Aggregate the stimulus and response processing into concurrent processes. A process may be associated with each class of stimulus and response.
4. Design algorithms to process each class of stimulus and response. These must meet the given timing requirements.
5. Design a scheduling system which will ensure that processes are started in time to meet their deadlines.
6. Integrate using a real-time operating system.

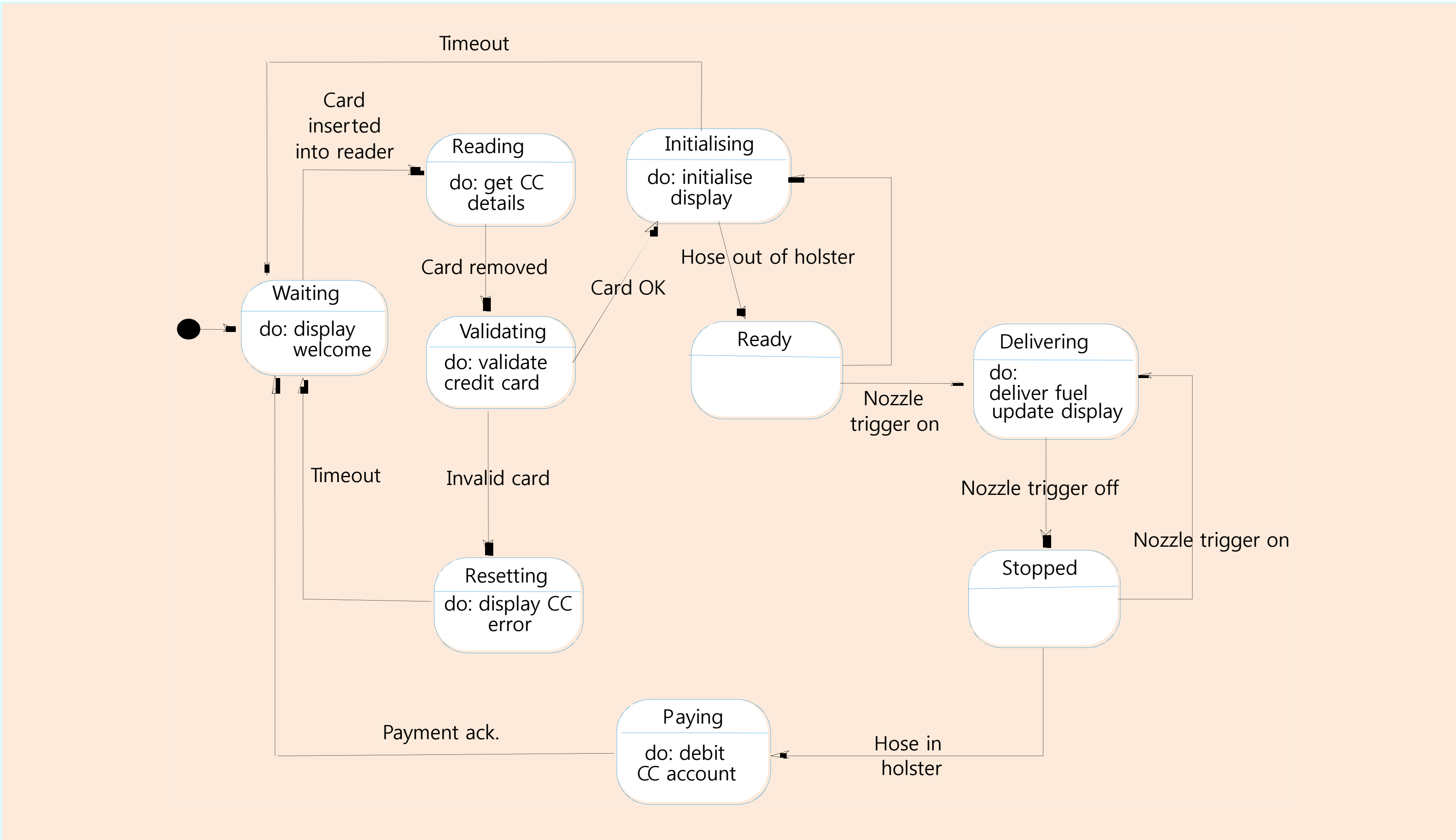
Timing Constraints

- May require extensive simulation and experiment to ensure that these are met by the system
- May mean that certain design strategies such as object-oriented design cannot be used because of the additional overhead involved
- May mean that low-level programming language features have to be used for performance reasons

Real-Time System Modelling

- The effect of a stimulus in a real-time system may trigger a transition from one state to another.
- <u>Finite State Machines (FSM)</u> can be used for modelling real-time systems.
  - However, FSM models lack structure. Even simple systems can have complex models.
  - The UML includes notations for defining state machine models.
- See Chapter 8 for further examples of state machine models.

Petrol Pump State Model

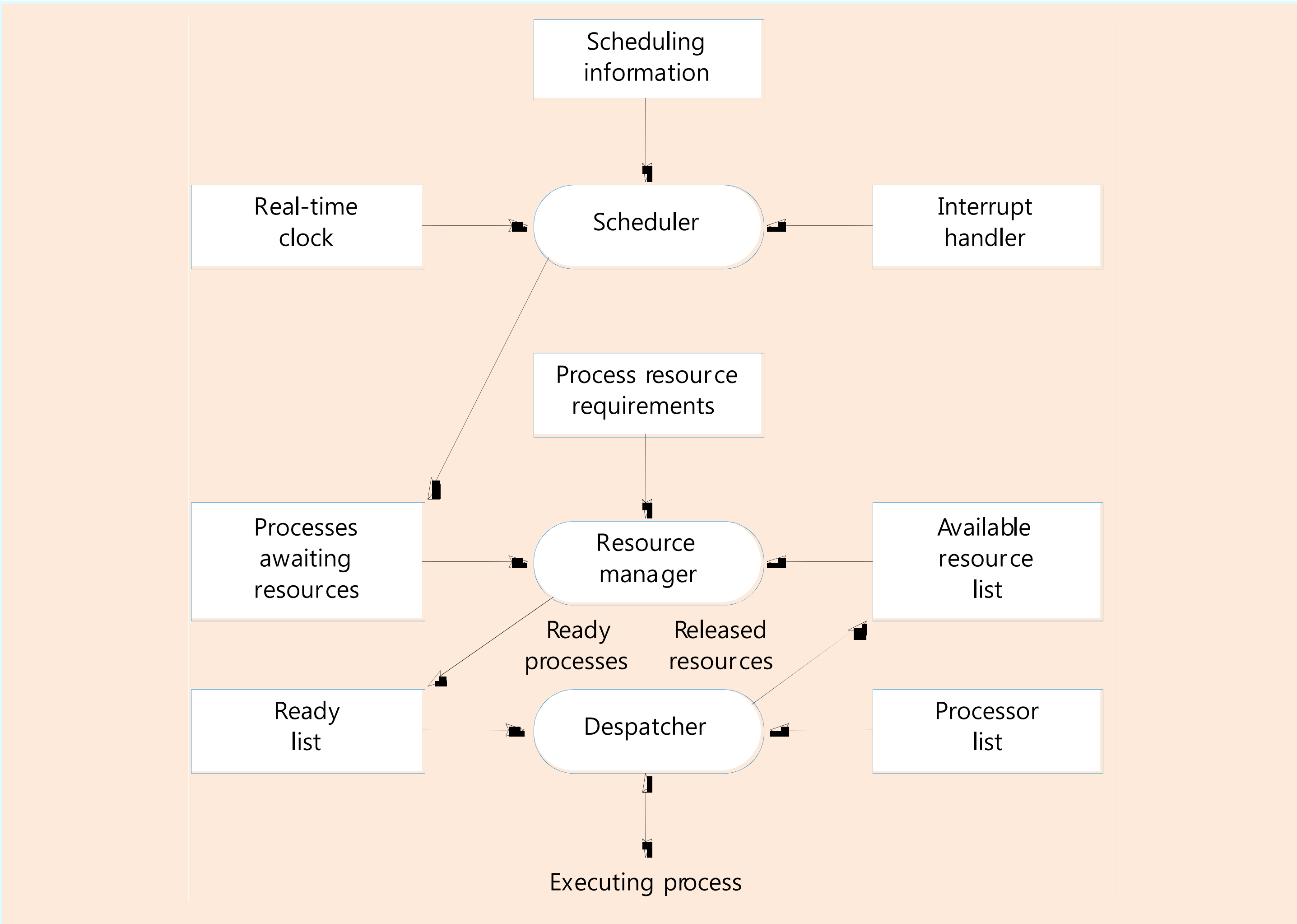


Real-Time Operating Systems

- Real-time operating systems are specialized operating systems which manage the processes in the RTS.
  - 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
  - Do not normally include facilities such as file management

- Real-time operating system components
  - Real-time clock : provides information for process scheduling
  - Interrupt handler : manages aperiodic requests for service
  - Scheduler : chooses the next process to be run
  - Resource manager : allocates memory and processor resources
  - Dispatcher : starts process execution

Real-Time OS Components



Non-Stop System Components

- Configuration manager
  - Responsible for the dynamic reconfiguration of the system software and hardware.
  - Hardware modules may be replaced and software upgraded without stopping the systems.
- Fault manager
  - Responsible for detecting software and hardware faults and taking appropriate actions (e.g. switching to backup disks)
  - To ensure that the system continues in operation

Process Priority

- The processing of some types of stimuli must sometimes take priority.
  - Interrupt level priority
    - Highest priority
    - Allocated to processes requiring a very fast response
  - Clock level priority
    - Allocated to periodic processes
- Within these, further levels of priority may be assigned.

Interrupt Servicing

- Control is transferred automatically to a pre-determined memory location.
  - This location contains an instruction to jump to an interrupt service routine.
  - Further interrupts are disabled, the interrupt serviced and the control returned to the interrupted process.

- Interrupt service routines MUST be short, simple and fast.

Periodic Process Servicing

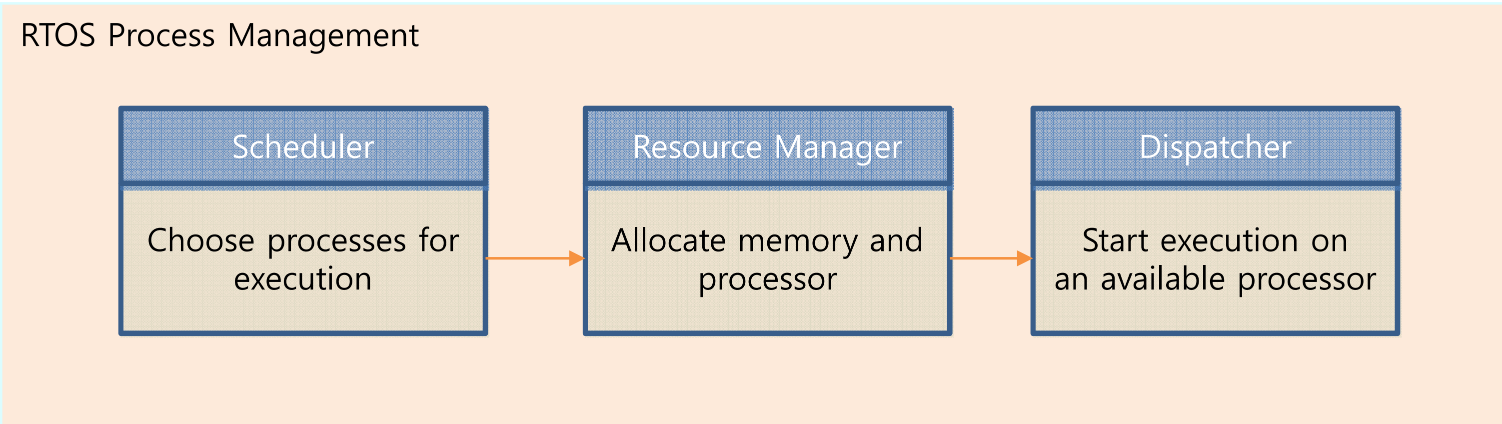
- In most real-time systems, there will be several classes of periodic process, each with different periods (the time between executions), execution times and deadlines (the time by which processing must be completed).

- The real-time clock ticks periodically and each tick causes an interrupt which schedules the process manager for periodic processes.
- The process manager selects a process which is ready for execution.

Process Management

- Concerned with managing the set of concurrent processes.
- Periodic processes are executed at pre-specified time intervals.

- The RTOS uses the real-time clock to determine when to execute a process taking into account
  - Process period : time between executions.
  - Process deadline : the time by which processing must be complete.



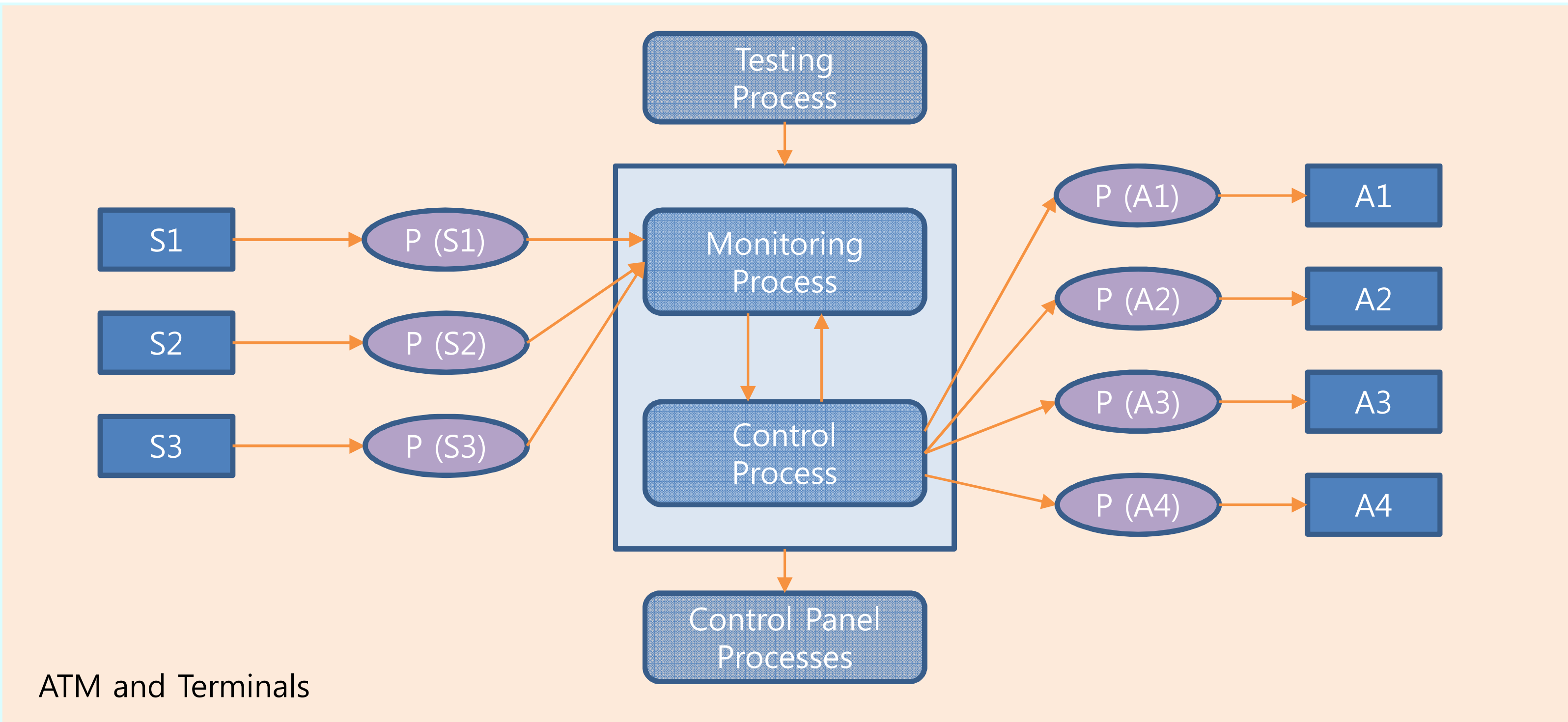
Process Switching

- The scheduler chooses the next process to be executed by the processor.
  - Depends on a scheduling strategy.
- The resource manager allocates memory and a processor for the process to be executed.
- The dispatcher takes the process from ready list, loads it onto a processor and starts execution.

- Scheduling strategies
  - 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, many others.

Monitoring and Control Systems

- Continuously check sensors and take actions depending on sensor values.
- Monitoring systems examine sensors and report their results.
- Control systems take sensor values and control hardware actuators.



Summary

- Real-time system correctness depends not just on what the system does but also on how fast it reacts.
- A general real-time system model involves associating processes with sensors and actuators.
- Real-time systems architectures are usually designed as a number of concurrent processes.
- Real-time operating systems are responsible for process and resource management.
- Monitoring and control systems poll sensors and send control signal to actuators.