EO1. *Embedded Software Architecture Design*

Spring 2014

Sungwon Kang

Acknowledgement Much of the following lecture slides are based on Ian Sommervilles' lecture slides for his Software Engineering textbook.

Embedded Software

- Computers are used to control a wide range of systems such as
  - domestic machines
  - games controllers
  - manufacturing plants

- The software in these systems is embedded in system hardware, often in read-only memory
  - Usually responds, in real time, to events from the system's environment.
  - Often, issue control signals in response to these events

Responsiveness

- Responsiveness in real-time is the critical difference between embedded systems and other software systems (e.g. information systems, web-based systems or personal software systems):
  - Non-real-time systems: correctness can be defined by specifying how system inputs map to corresponding outputs that should be produced by the system.
  - Real-time systems: correctness depends both on the response to an input and the time taken to generate that response.

Definition

- Real-time system: 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.

Embedded System Characteristics

- Generally run continuously without termination.
- Interactions with the system's environment are uncontrollable and unpredictable.
- There may be physical limitations (e.g. power) that affect the design of a system.
- Direct hardware interaction may be necessary.
- Issues of safety and reliability may dominate the system design.

Table of Contents

1. Embedded System Design

1) Process
2) Reactive Systems
3) General Model of an Embedded Real-time System
4) Architectural Considerations
5) Design Activities
6) Process Coordination
7) Real-time System Modeling

1) Process

- Has to consider, in detail, the design and performance of the system hardware.
  - Involves deciding which system capabilities are to be implemented in software and which in hardware.
  - Low-level decisions on hardware, support software and system timing must be considered early in the process.
    => Additional software functionality, such as battery and power management, may have to be included.

2) Reactive Systems

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

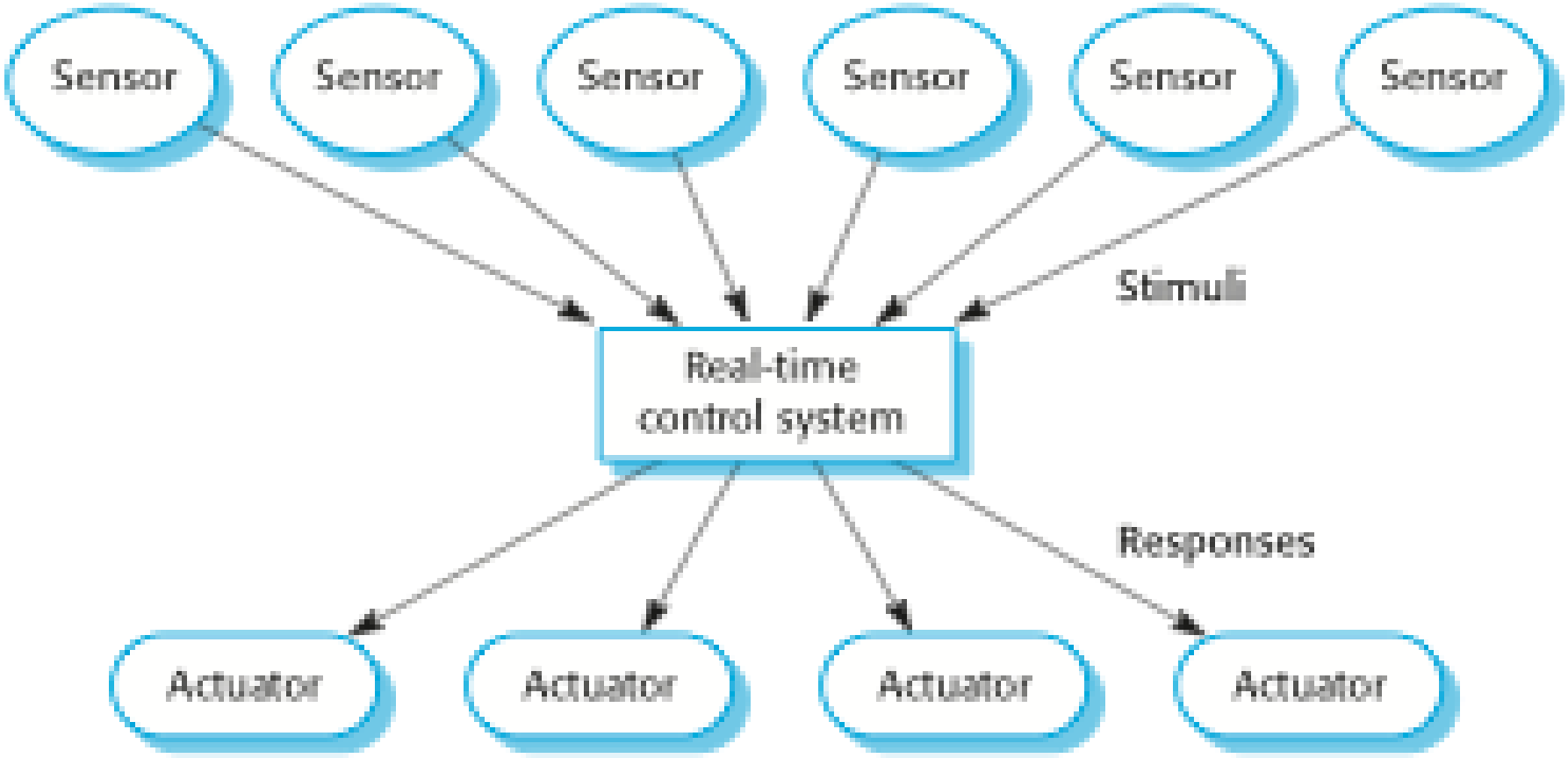
  - Periodic stimuli: Stimuli occur at predictable time intervals
    - **Example** A temperature sensor may be polled 10 times per second.
  - Aperiodic stimuli: Stimuli occur at unpredictable times
    - **Example** A system power failure may trigger an interrupt which must be processed by the system.

Stimuli/Responses for a Burglar Alarm System

Stimulus	Response
Single sensor positive	Initiate alarm; Turn on lights around site of positive sensor.
Two or more sensors positive	Initiate alarm; Turn on lights around sites of positive sensors; Call police with location of suspected break-in.
Voltage drop of between 10% and 20%	Switch to battery backup; Run power supply test.
Voltage drop of more than 20%	Switch to battery backup; Initiate alarm; Call police; Run power supply test.
Power supply failure	Call service technician.
Sensor failure	Call service technician.
Console panic button positive	Initiate alarm; Turn on lights around console; Call police.
Clear alarms	Switch off all active alarms; Switch off all lights that have been switched on.

3) General Model of an Embedded Real-time System

Can be called Generic Architecture or Architecture Style or Architecture Pattern



4) Architectural Considerations

- Need to respond to timing demands

  $\Rightarrow$ System architecture must allow for fast switching between stimulus handlers.

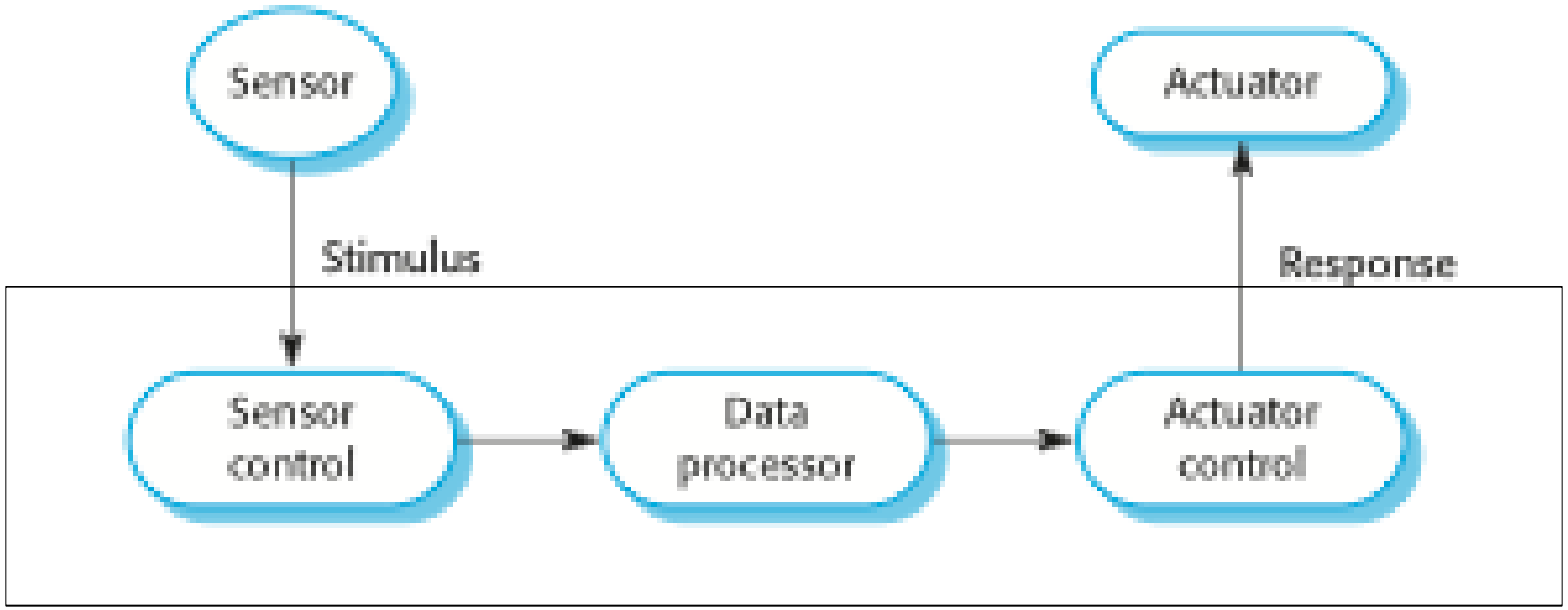
- Timing demands of different stimuli are different

  => A simple sequential loop is not usually adequate.

  => Usually designed as cooperating processes

  with a real-time executive controlling these processes.

Sensor and Actuator Processes



System Elements

- Sensor control processes
  - Collect information from sensors.
  - May buffer information collected 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.

5) Design Activities

- Platform selection
- Stimuli/response identification
- Timing analysis
- Process design
- Algorithm design
- Data design
- Process scheduling

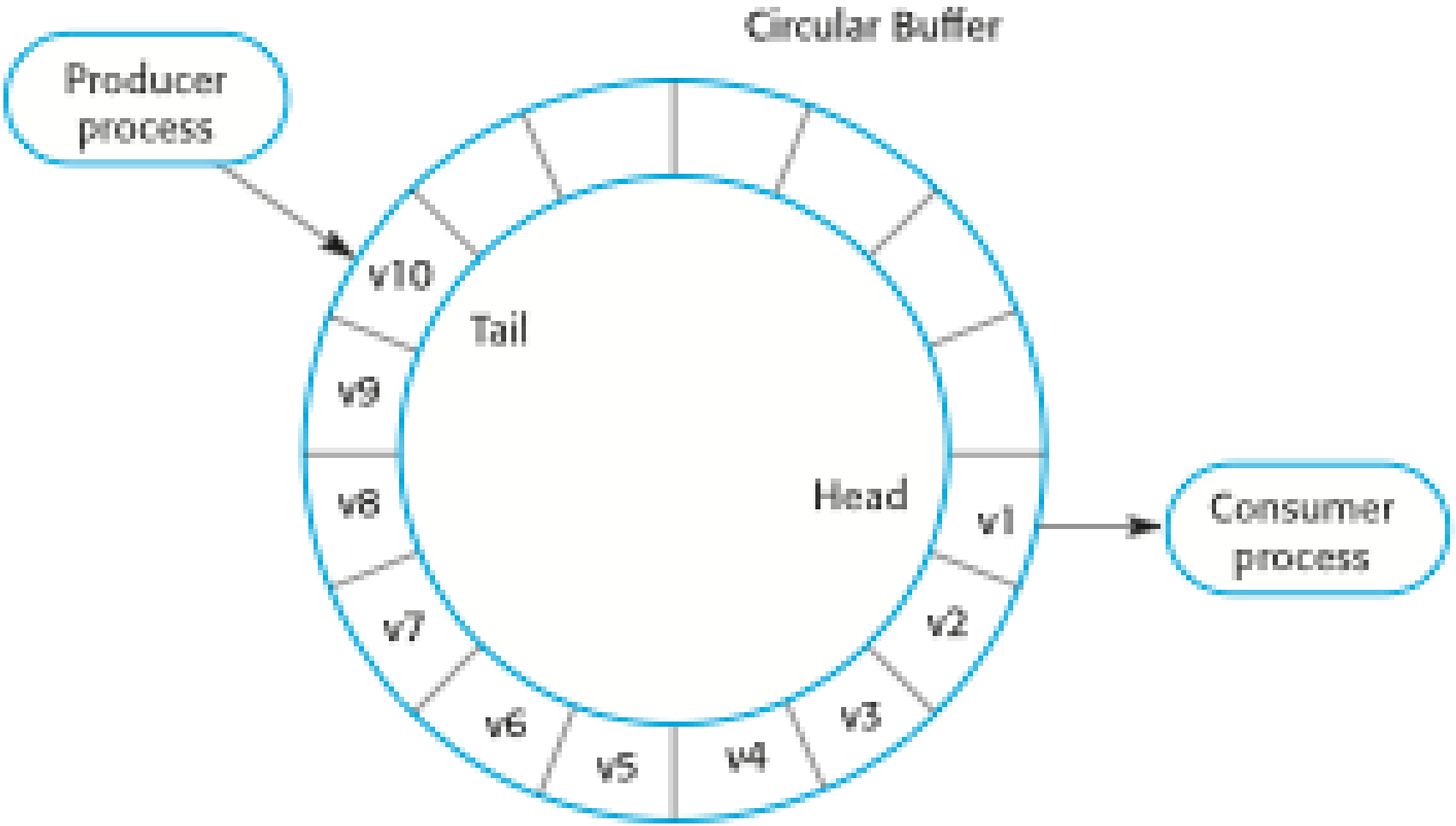
=> Precise order of activities depends on the system and the design method used.

6) Process Coordination

- Processes in a real-time system have to be coordinated and share information.

- Process coordination mechanisms ensure mutual exclusion to shared resources.
  - When one process is modifying a shared resource, other processes should not be able to change that resource.

Producer/Consumer Processes Sharing a Circular Buffer



Producer/Consumer Processes Sharing a Circular Buffer

- Producer processes collect data and add it to the buffer.
- Consumer processes take data from the buffer and make elements available.
- The buffer must not allow:
  - producer processes to add information to a full buffer
  - consumer processes to take information from an empty buffer

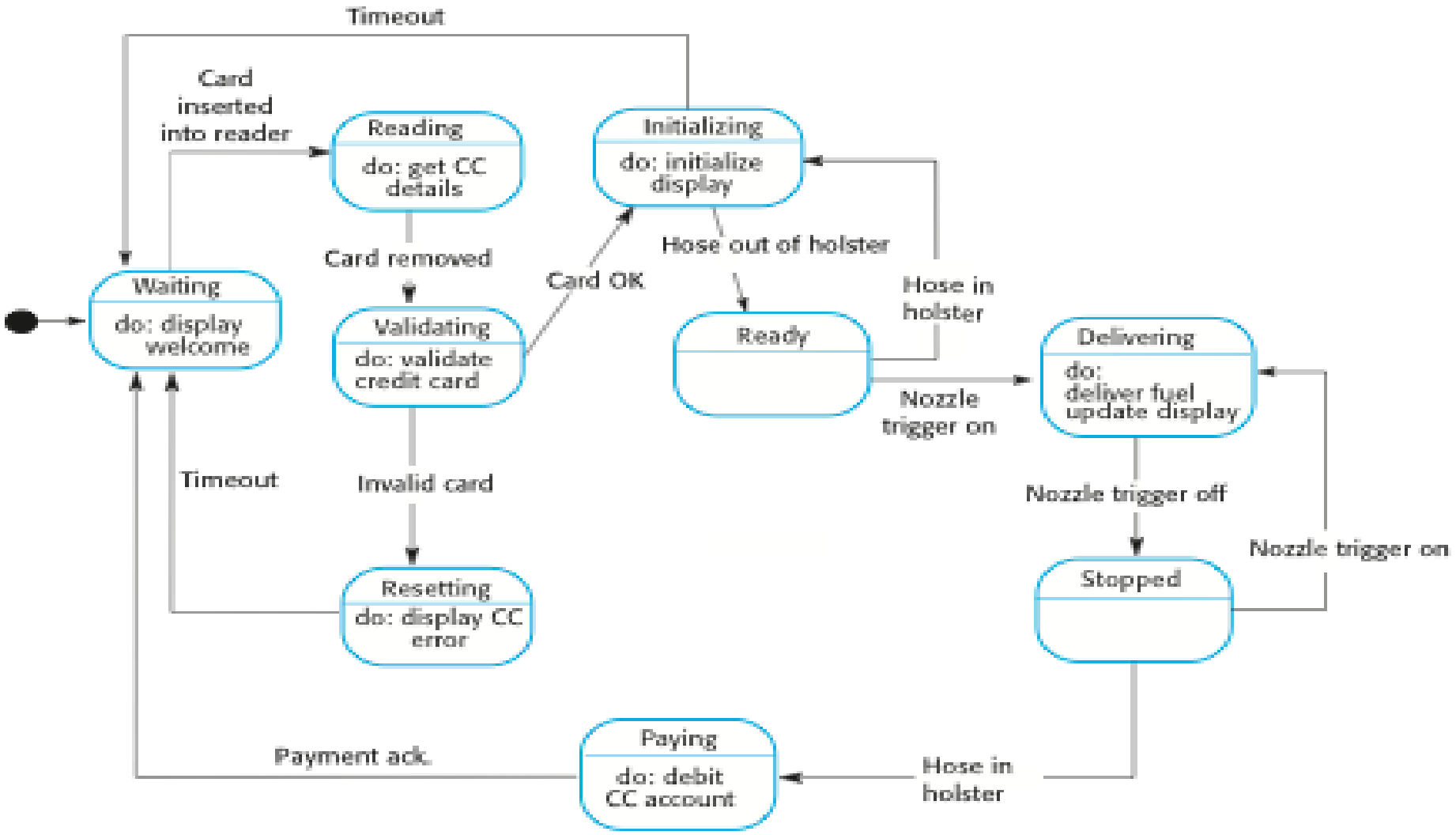
☛ Mutual Exclusion: Producer and consumer processes are mutually excluded from accessing the same element.

7) Real-time System Modelling

- A stimulus in a real-time system may trigger a transition from one state to another.
  - State models are therefore often used to describe embedded real-time systems.

    ⇒ UML state diagrams may be used to show the states and state transitions in a real-time system.

State Machine Model of a Gasoline Pump



2. Architectural Patterns for Embedded Systems

A. Observe and React Pattern

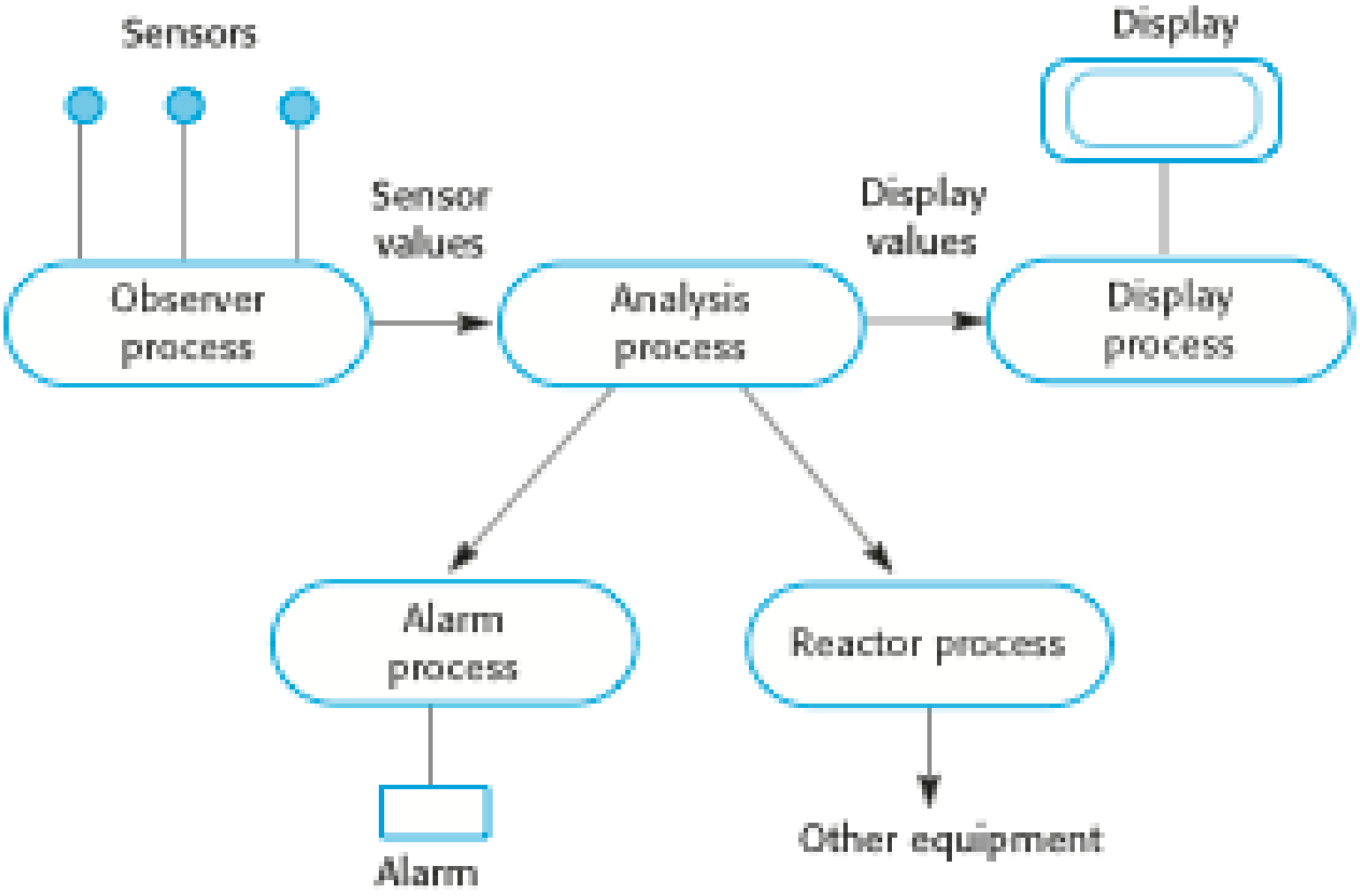
B. Environmental Control Pattern

C. Process Pipeline Pattern

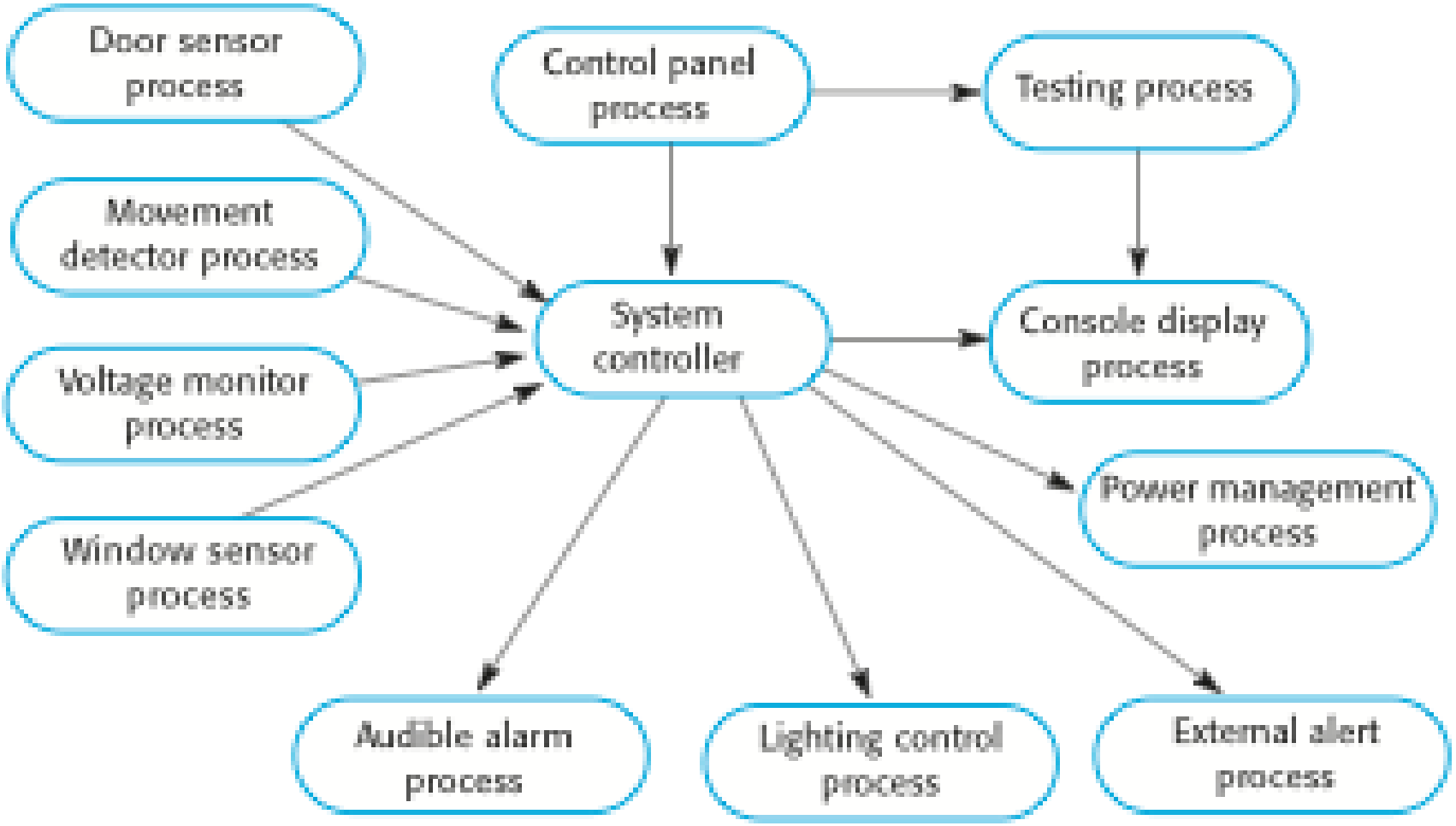
A. The Observe and React pattern

Name	Observe and React
Description	- The input values of a set of sensors of the same types are collected and analyzed. - These values are displayed in some way. - If the sensor values indicate that some exceptional condition has arisen, then actions are initiated to draw the operator's attention to that value and, in certain cases, to take actions in response to the exceptional value.
Stimuli	Values from sensors attached to the system.
Responses	Outputs to display, alarm triggers, signals to reacting systems.
Processes	Observer, Analysis, Display, Alarm, Reactor.
Used in	Monitoring systems, alarm systems.

Observe and React Process Structure



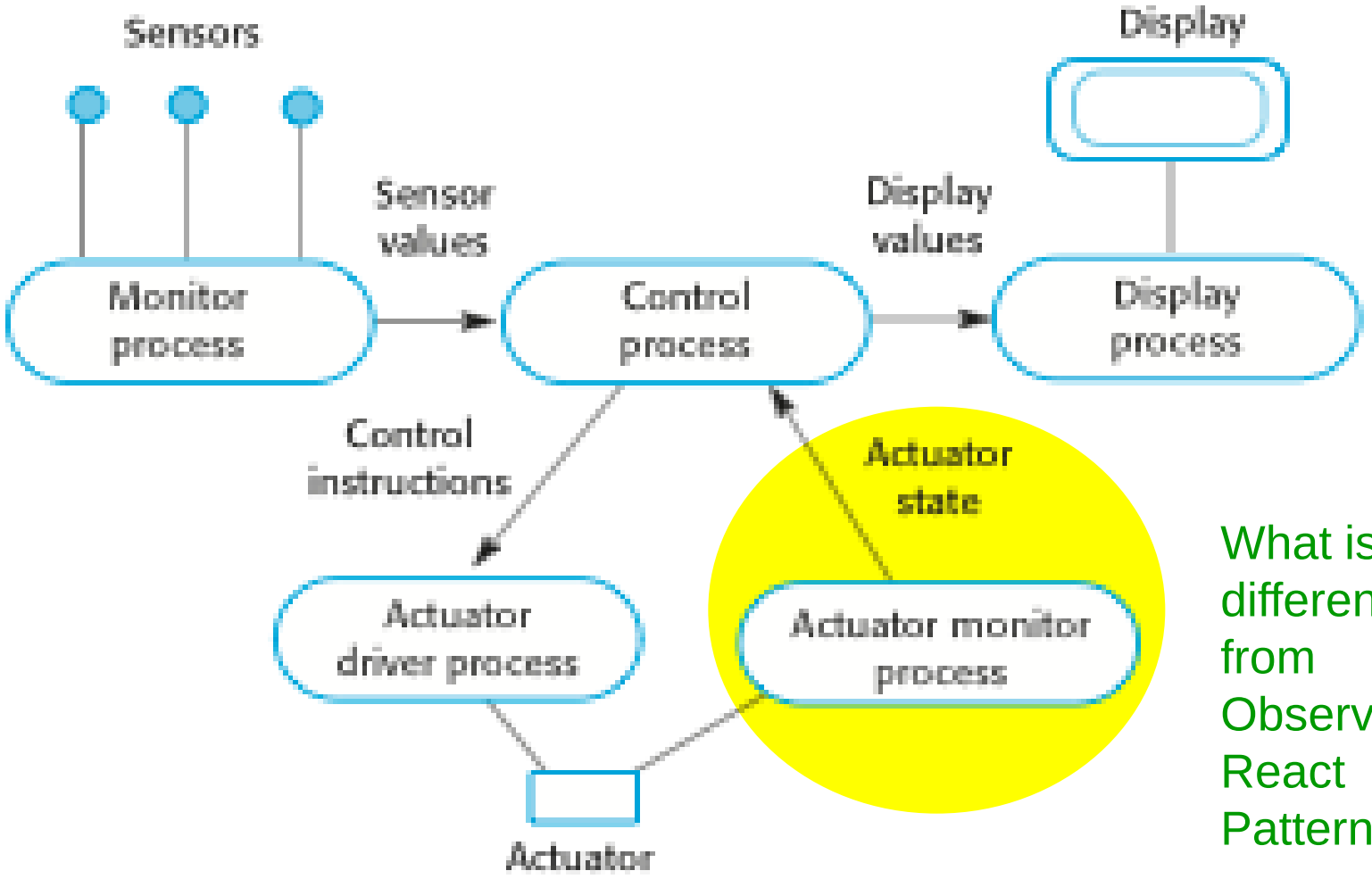
Process Structure for a Burglar Alarm System



B. The Environmental Control pattern

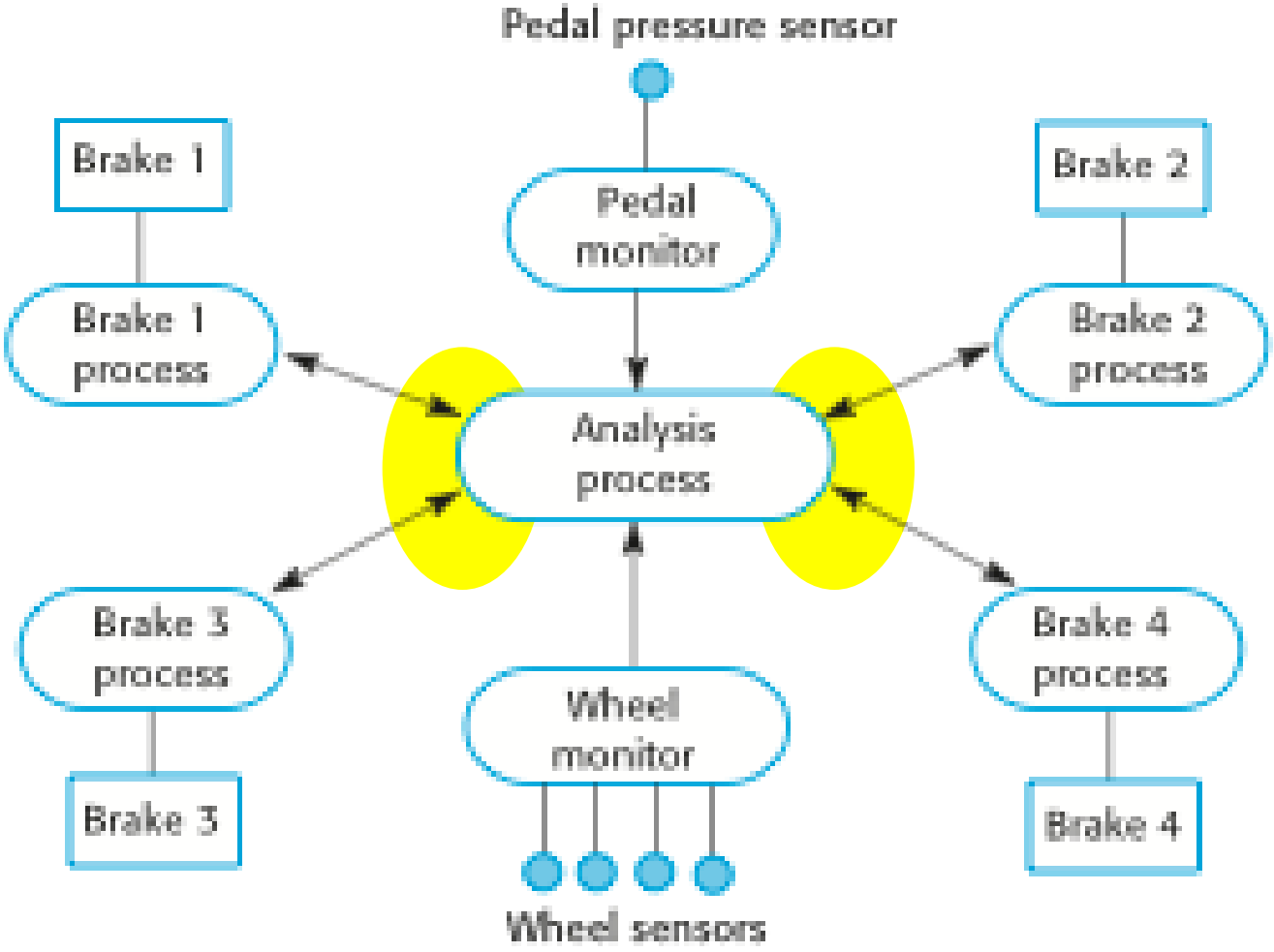
Name	Environmental Control
Description	- The system analyses information from a set of sensors that collect data from the system's environment. - Further information may also be collected on the state of the actuators. - Based on the data from the sensors and actuators, control signals are sent to the actuators. - Information about the sensor values and the state of the actuators may be displayed.
Stimuli	Values from sensors attached to the system and the state of the system actuators.
Responses	Control signals to actuators, display information.
Processes	Monitor, Control, Display, Actuator Driver, Actuator monitor.
Used in	Control systems.

Environmental Control Process Structure



What is the difference from Observe and React Pattern?

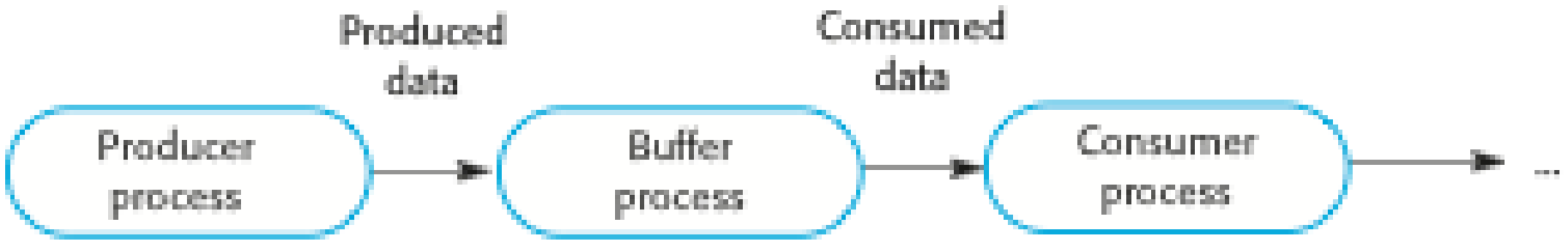
Control System Architecture for an Anti-Skid Braking System



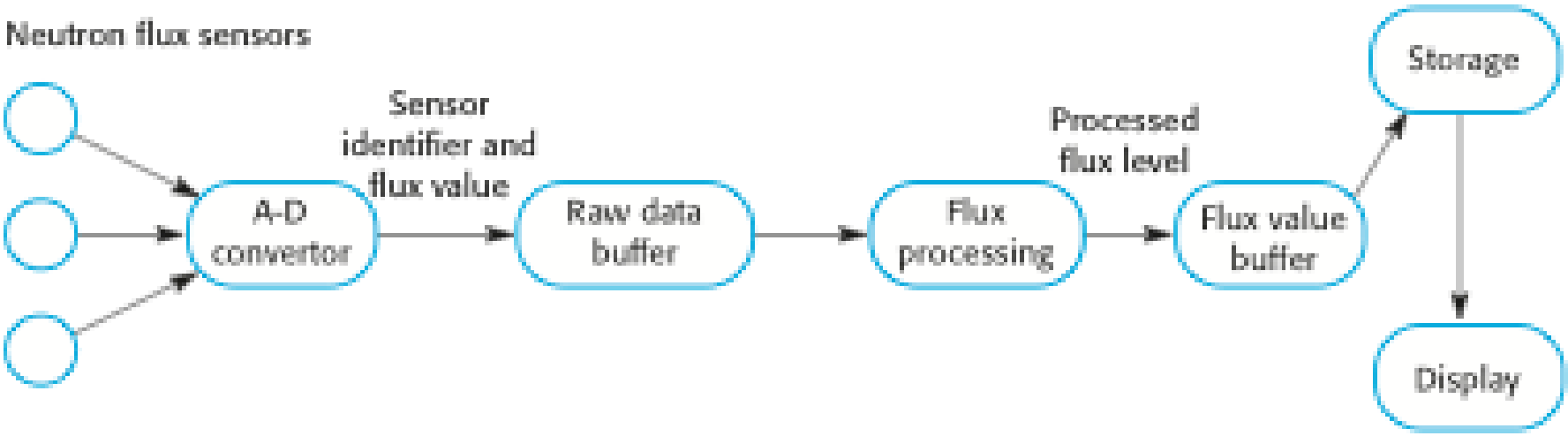
C. The Process Pipeline Pattern

Name	Process Pipeline
Description	- A pipeline of processes is set up with data moving in sequence. - The processes are often linked by synchronized buffers to allow the producer and consumer processes to run at different speeds. - The pipeline may terminate with display or data storage or an actuator.
Stimuli	Input values from the environment or some other process
Responses	Output values to the environment or a shared buffer
Processes	Producer, Buffer, Consumer
Used in	Data acquisition systems, multimedia systems

Process Pipeline Process Structure



Neutron Flux Data Acquisition



Structural Patterns (Source: [Gomaa 11])

Table A.1. Software architectural structure patterns

Software architectural structure patterns	Pattern description	Reference chapter
Broker Pattern	Section A.1.1	Chapter 16, Section 16.2
Centralized Control Pattern	Section A.1.2	Chapter 18, Section 18.3.1
Distributed Control Pattern	Section A.1.3	Chapter 18, Section 18.3.2
Hierarchical Control Pattern	Section A.1.4	Chapter 18, Section 18.3.3
Layers of Abstraction Pattern	Section A.1.5	Chapter 12, Section 12.3.1
Multiple Client/Multiple Service Pattern	Section A.1.6	Chapter 15, Section 15.2.2
Multiple Client/Single Service Pattern	Section A.1.7	Chapter 15, Section 15.2.1
Multi-tier Client/Service Pattern	Section A.1.8	Chapter 15, Section 15.2.3

A.1.2 Centralized Control Pattern

Pattern name	Centralized Control
Aliases	Centralized Controller, System Controller
Context	Centralized application where overall control is needed
Problem	Several actions and activities are state-dependent and need to be controlled and sequenced.
Summary of solution	There is one control component, which conceptually executes a statechart and provides the overall control and sequencing of the system or subsystem.
Strengths of solution	Encapsulates all state-dependent control in one component
Weaknesses of solution	Could lead to overcentralized control, in which case decentralized control should be considered.
Applicability	Real-time control systems, state-dependent applications
Related patterns	Distributed Control, Hierarchical Control

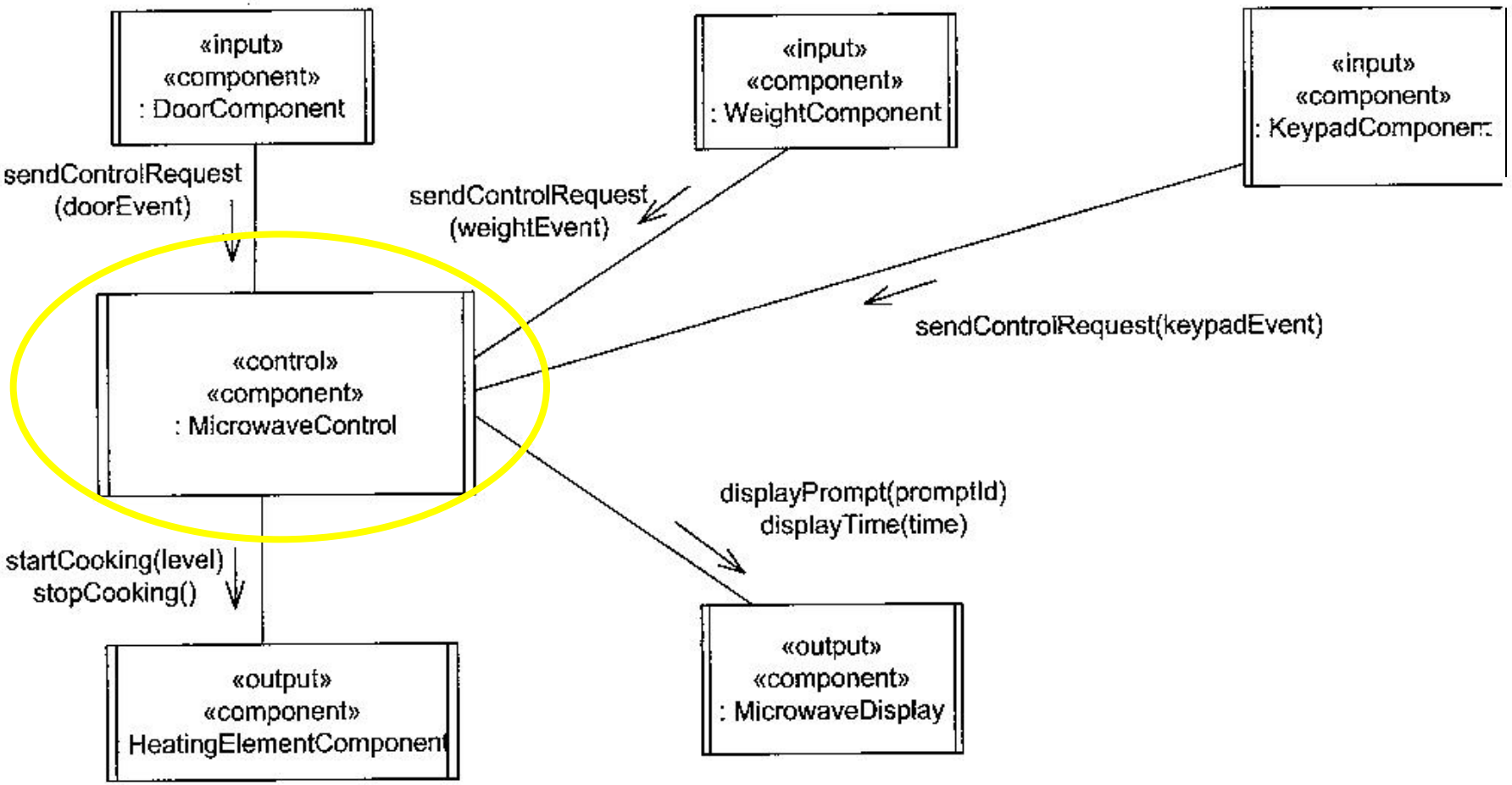


Figure A.2. Centralized Control pattern: Microwave Oven Control System example

A.1.3 Distributed Control Pattern

Pattern name	Distributed Control
Aliases	Distributed Controller
Context	Distributed application with real-time control requirement
Problem	Distributed application with multiple locations where real-time localized control is needed at several locations
Summary of solution	There are several control components, such that each component controls a given part of the system by conceptually executing a state machine. Control is distributed among the various control components; no single component has overall control.
Strengths of solution	Overcomes potential problem of overcentralized control.
Weaknesses of solution	Does not have an overall coordinator. If this is needed, consider using Hierarchical Control pattern.
Applicability	Distributed real-time control, distributed state-dependent applications
Related patterns	Hierarchical Control, Centralized Control

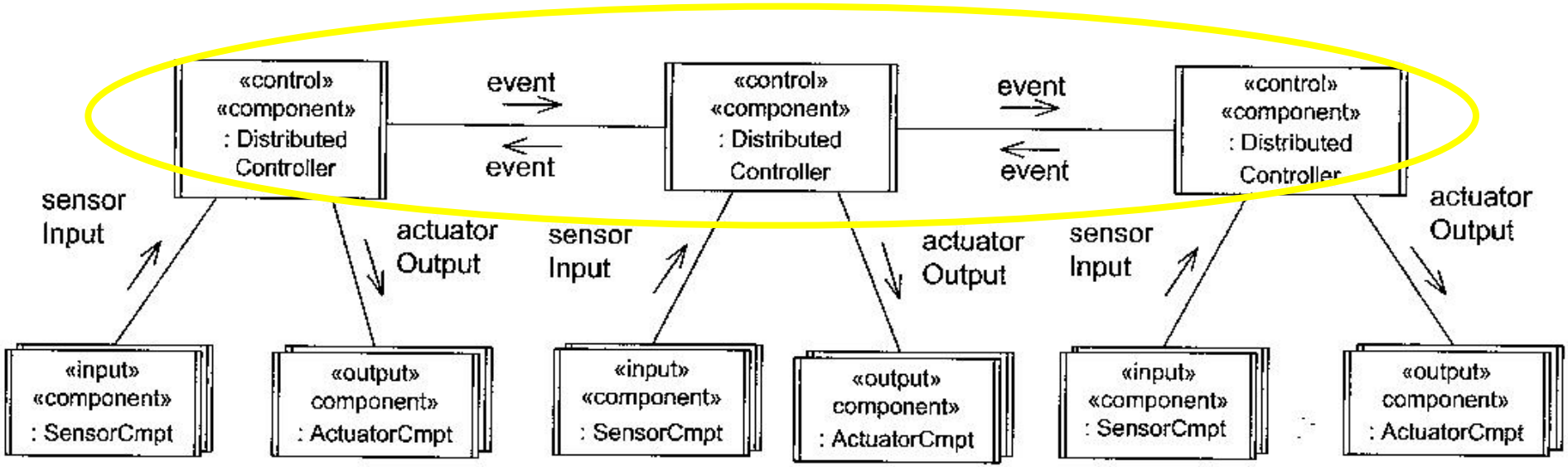


Figure A.3. Distributed Control pattern

A.1.4 Hierarchical Control Pattern

Pattern name	Hierarchical Control
Aliases	Multilevel Control
Context	Distributed application with real-time control requirement
Problem	Distributed application with multiple locations where both real-time localized control and overall control are needed
Summary of solution	There are several control components, each controlling a given part of a system by conceptually executing a statechart. There is also a coordinator component, which provides high-level control by deciding the next job for each control component and communicating that information directly to the control component.
Strengths of solution	Overcomes potential problem with Distributed Control pattern by providing high-level control and coordination
Weaknesses of solution	High-level coordinator may become a bottleneck when the load is high and is a single point of failure.
Applicability	Distributed real-time control, distributed state-dependent applications
Related patterns	Distributed Control, Centralized Control

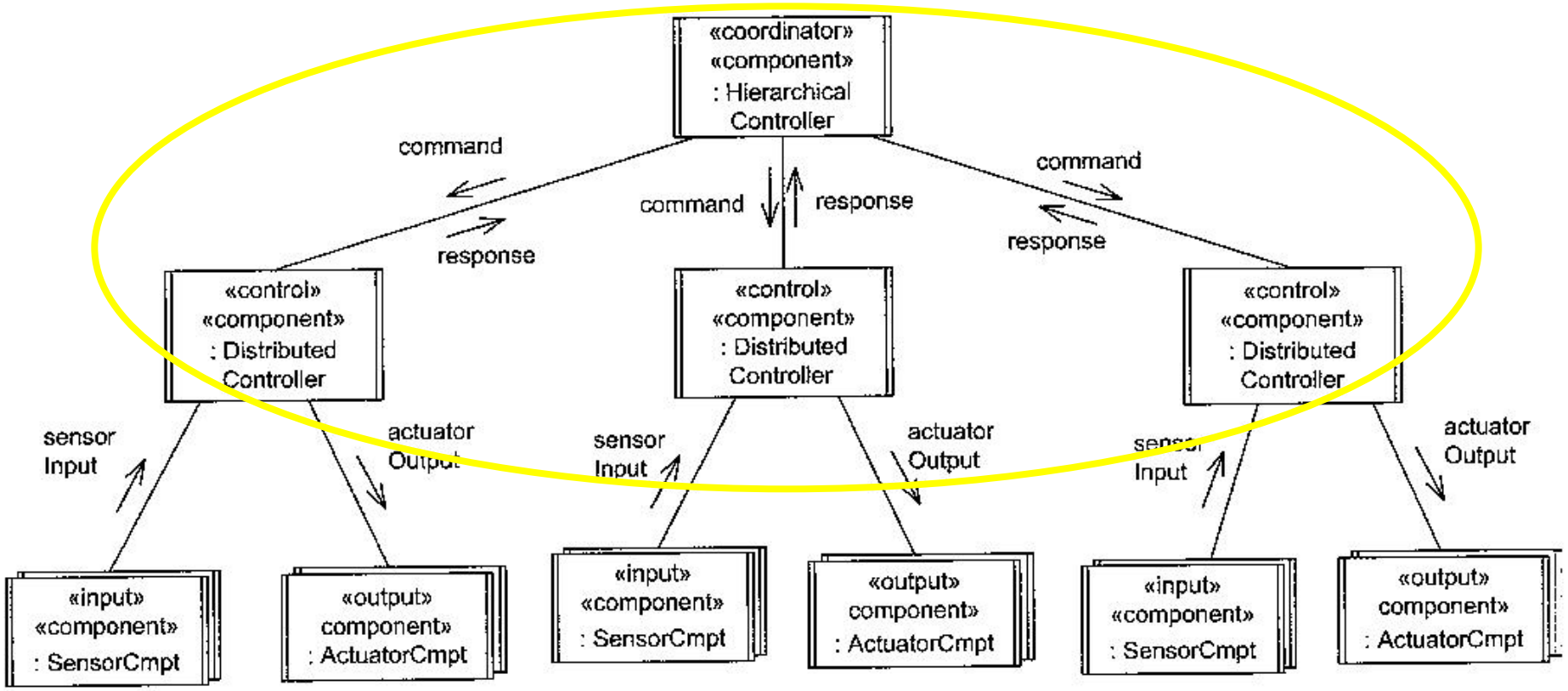


Figure A.4. Hierarchical Control pattern

3. Timing Analysis

Timing Analysis

- Correctness of real time systems depends on response time.

  => Timing design is important

- Timing design is challenging, esp. when both period stimuli and aperiodic stimuli have to be dealt with.

- As computer processing speed increases, stimuli that were treated as aperiodic can be treated as periodic.

  Example In the case of power failure, slow shut down of attached equipment may result in damage.

  => Could be implemented as a “power failure interrupt”.

  => Could also be implemented as a periodic process that runs very frequently.

Timing Analysis

- Assuming that only periodic processes are needed,
  - Key factors for timing analysis:
    - Deadline
    - Frequency
    - Execution time

Example Assume that after a power failure event, it takes 50 ms for the supplied voltage to drop to a level where the equipment may be damaged.

⇒ Equipment shutdown process must begin within 50ms of a power failure.

⇒ It is safe to issue and perform shutdown instructions within 40 ms.

⇒ To detect power failure from voltage dropping, suppose the process runs every 4 ms.

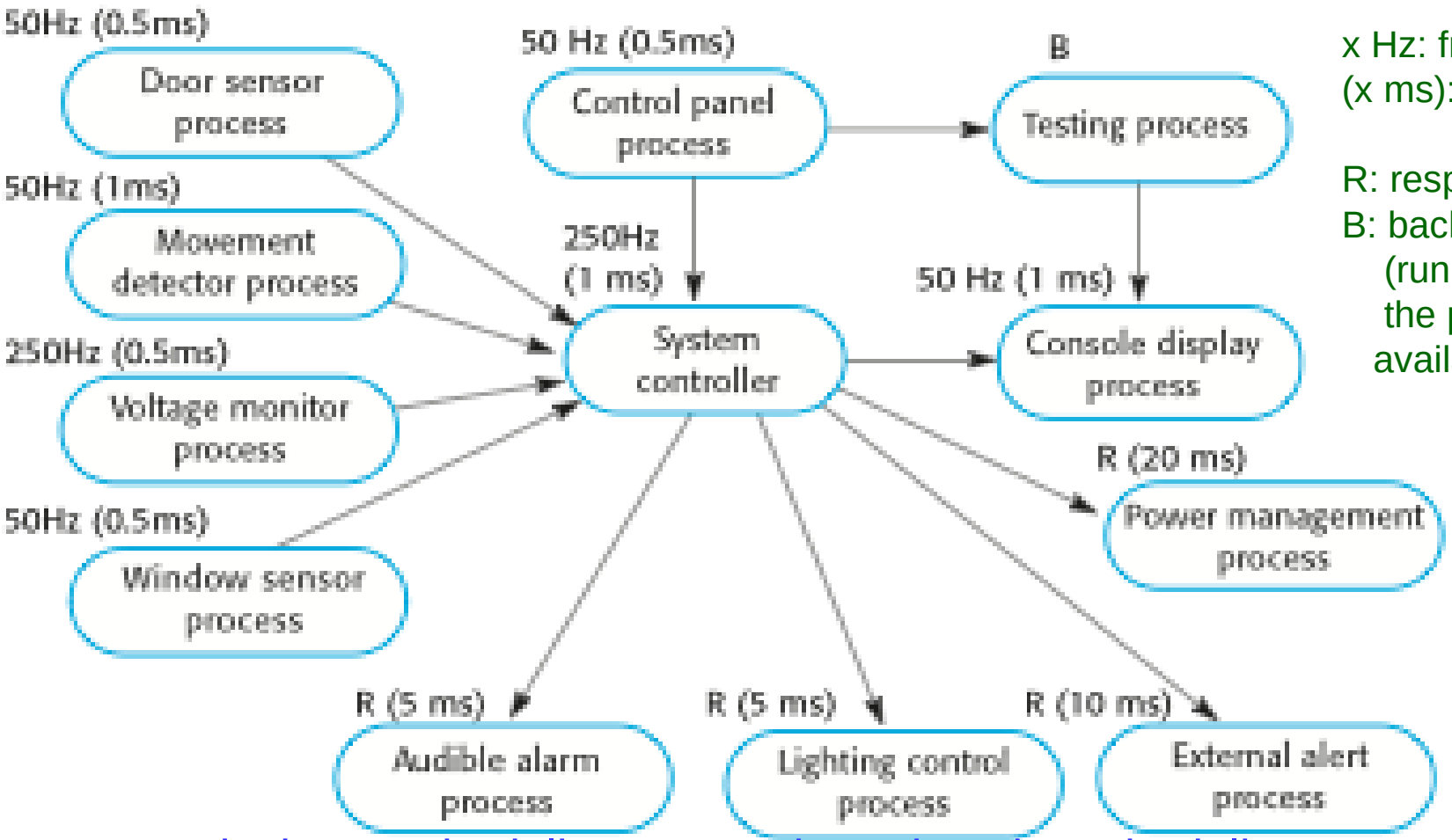
⇒ Then detection should allow 8 ms for worst case execution time.

☛ Would allow much shorter period in reality just in case !

Timing Requirements for the Burglar Alarm System

Stimulus/ Response	Timing requirements
Power failure	The switch to backup power must be completed within a deadline of 50 ms.
Door alarm	Each door alarm should be polled twice per second.
Window alarm	Each window alarm should be polled twice per second.
Movement detector	Each movement detector should be polled twice per second.
Audible alarm	The audible alarm should be switched on within half a second of an alarm.
Lights switch	The lights should be switched on within half a second of an alarm.
Communications	The call to the police should be started within 2 seconds of an alarm.
Voice synthesizer	A synthesized message should be available within 2 seconds of an alarm.

Alarm Process Timing



x Hz: frequency
(x ms): execution time
R: response
B: background (run only when the processor is available)

- Next, design a scheduling system based on the scheduling approaches supported by the real time OS

4. Real-time Operating Systems

Components

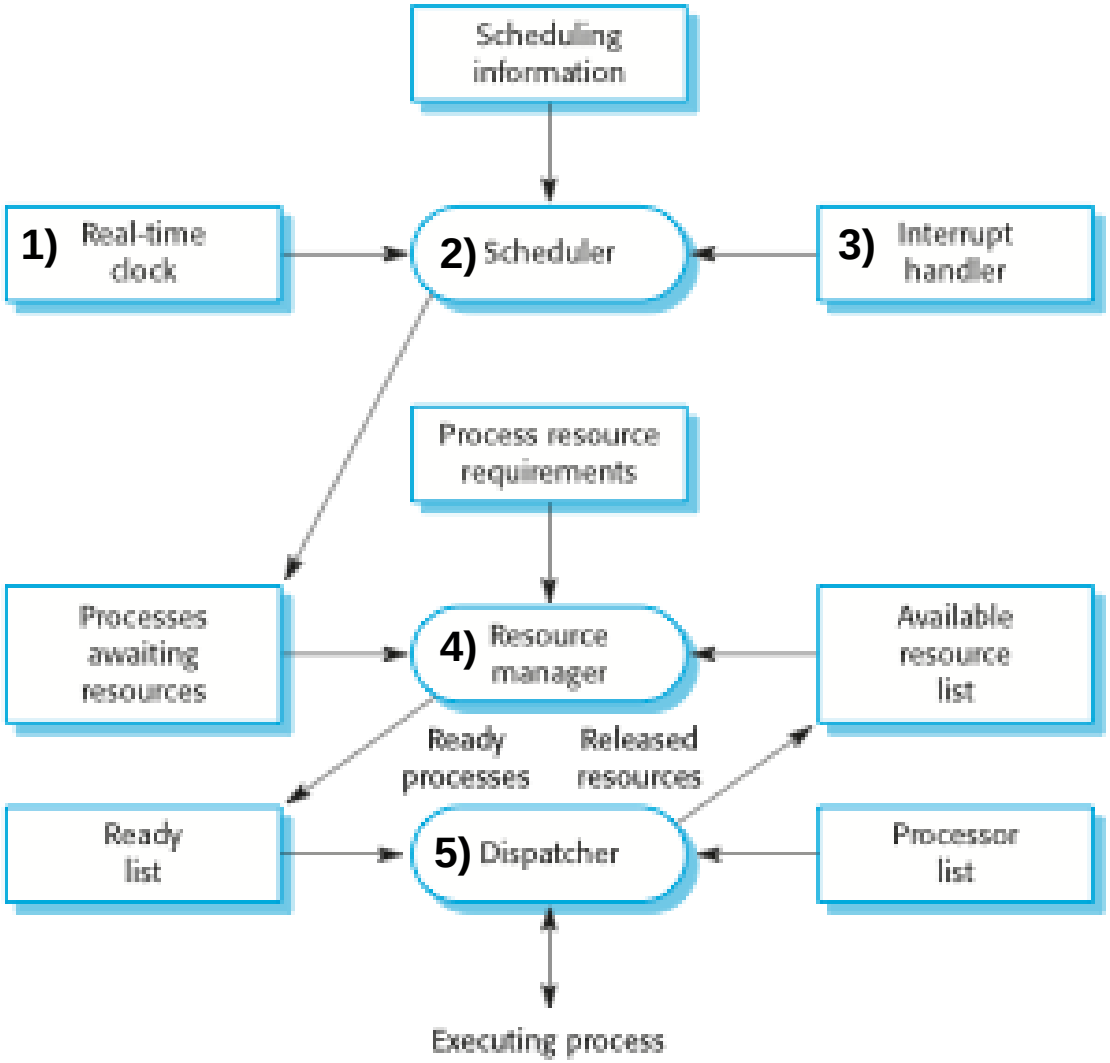
1) Real-time clock
   - Provides information for process scheduling.
2) Interrupt handler
   - Manages aperiodic requests for service.
3) Scheduler
   - Chooses the next process to be run.
4) Resource manager
   - Allocates memory and processor resources.
5) Dispatcher
   - Starts process execution.

Additional 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)

Components of a Real-time Operating System



Process Priority

- The processing of some types of stimuli must sometimes take priority:
  - Interrupt level priority: Highest priority which is 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 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).

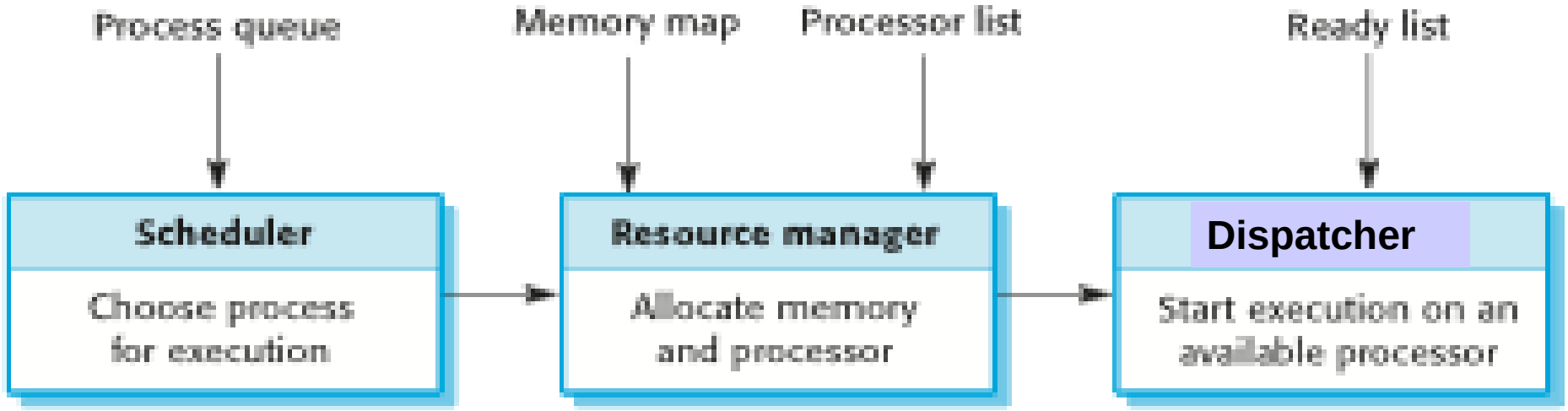
- The real-time clock ticks periodically and each tick causes an interrupt which schedules the process manager for periodic processes.

- 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



RTOS actions required to start a process

- Scheduler: chooses the next process to be executed by the processor. This depends on a scheduling strategy which may take the process priority into account.
- Resource manager: allocates memory and a processor for the process to be executed.
- 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 Period First)
  - Shortest deadline first

Questions?