Faculty of ECMS / School of Computer Science

Software Engineering & Project
Real-Time Software Design

Architectural Design

Lecture 12

Chapter 15 (20 in Edition 9)
in the course text book

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 a Real-Time Operating System (RTOS)
- To introduce generic process architectures for monitoring and control systems and data acquisition systems

Real-Time system examples

- Access control system
- Car safety system
- Flood monitoring system
- Bushfire monitoring system
- Automated Teller Machine (ATM)
- Flight reservation system
- Rocket thruster control system
- Quadcopter flight control system
- Bridge anti-earthquake system
- TV transitions
- Virtual reality
- Web browsing
- *Our bodies are real-time systems too*

Real-time systems

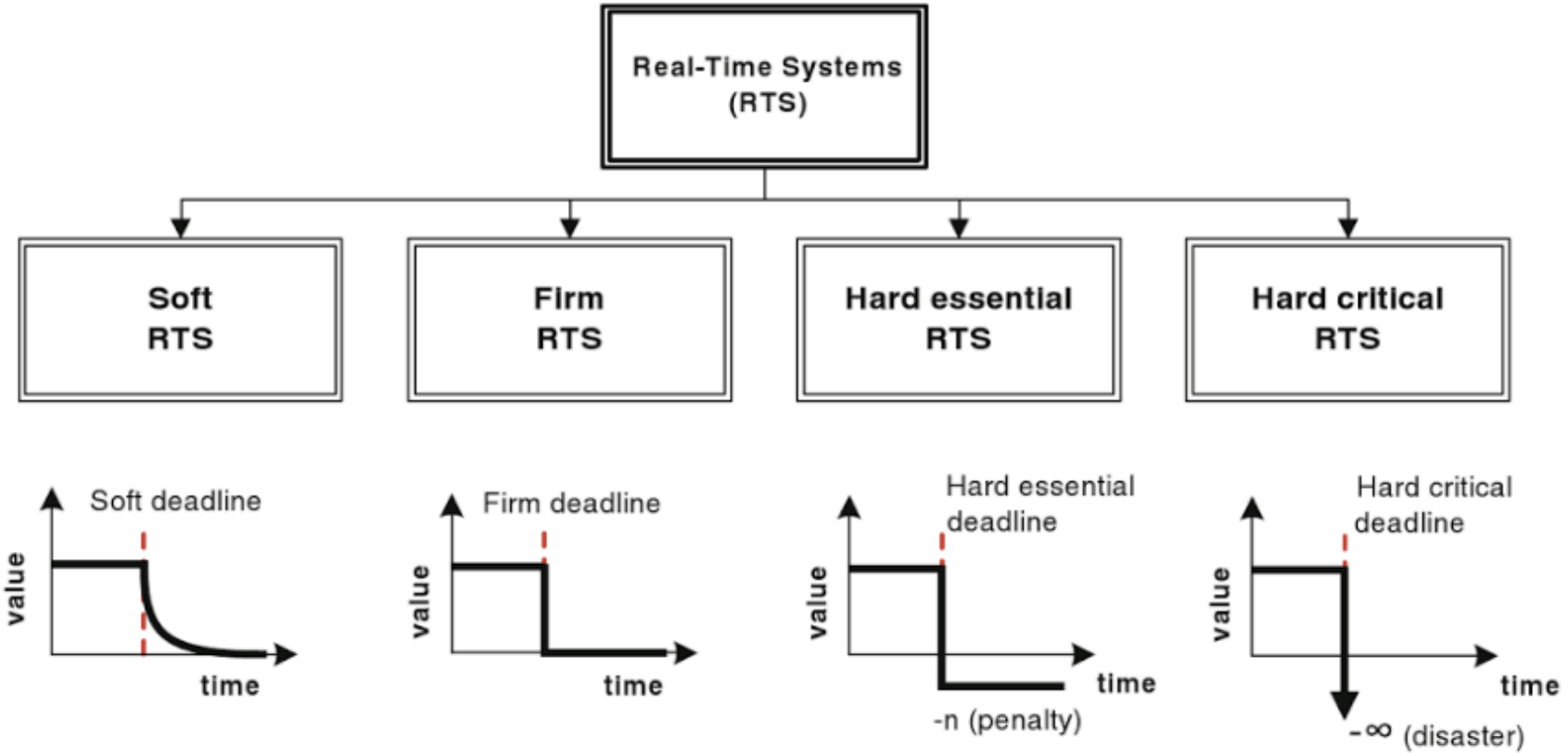
- Systems which monitor and control their environment; generally run continuously and do not terminate
- Interactions **with** the system are usually unpredictable
- Inevitably associated with hardware devices
  - *Sensors*: Collect status data from the system environment (e.g., speed, location, temperature);
  - *Actuators*: Control some equipment and change (in some way) the system's environment (e.g., alert an alarm, engage fan for a ventilation system)

Definitions

- A real-time system (RTS) 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.

- *"A system is said to be real-time if the total correctness of an operation depends not only upon its logical correctness, but also upon the time in which it is performed. The classical conception is that in a hard real-time or immediate real-time system, the completion of an operation after its deadline is considered useless - ultimately, this may cause a critical failure of the complete system. A soft real-time system on the other hand will tolerate such lateness, and may respond with decreased service quality (e.g., omitting frames while displaying a video)." ~ Wikipedia*

Definitions (Cont.)



Definitions (Cont.)

- Soft RTS
  - A system where results that miss deadlines still be value to some extent
- Firm RTS
  - A system where missing deadlines has no benefits gained or costs incurred
- Hard essential RTS
  - A system where missing a deadline has a bounded cost incurred
- Hard critical RTS
  - A system where missing a deadline has a catastrophic cost incurred

Stimulus Types

- RT system = Stimulus/response system
  - Given a stimulus (event), the system must produce a response within a specified time.

- **Periodic** stimuli. Stimuli which occur at predictable time intervals
  - For example, a temperature sensor may be polled 10 times per second.

- **Aperiodic** stimuli. Stimuli which occur at unpredictable times (normally exceptional situations)
  - For example, a system power failure may trigger an interrupt which must be processed by the system.

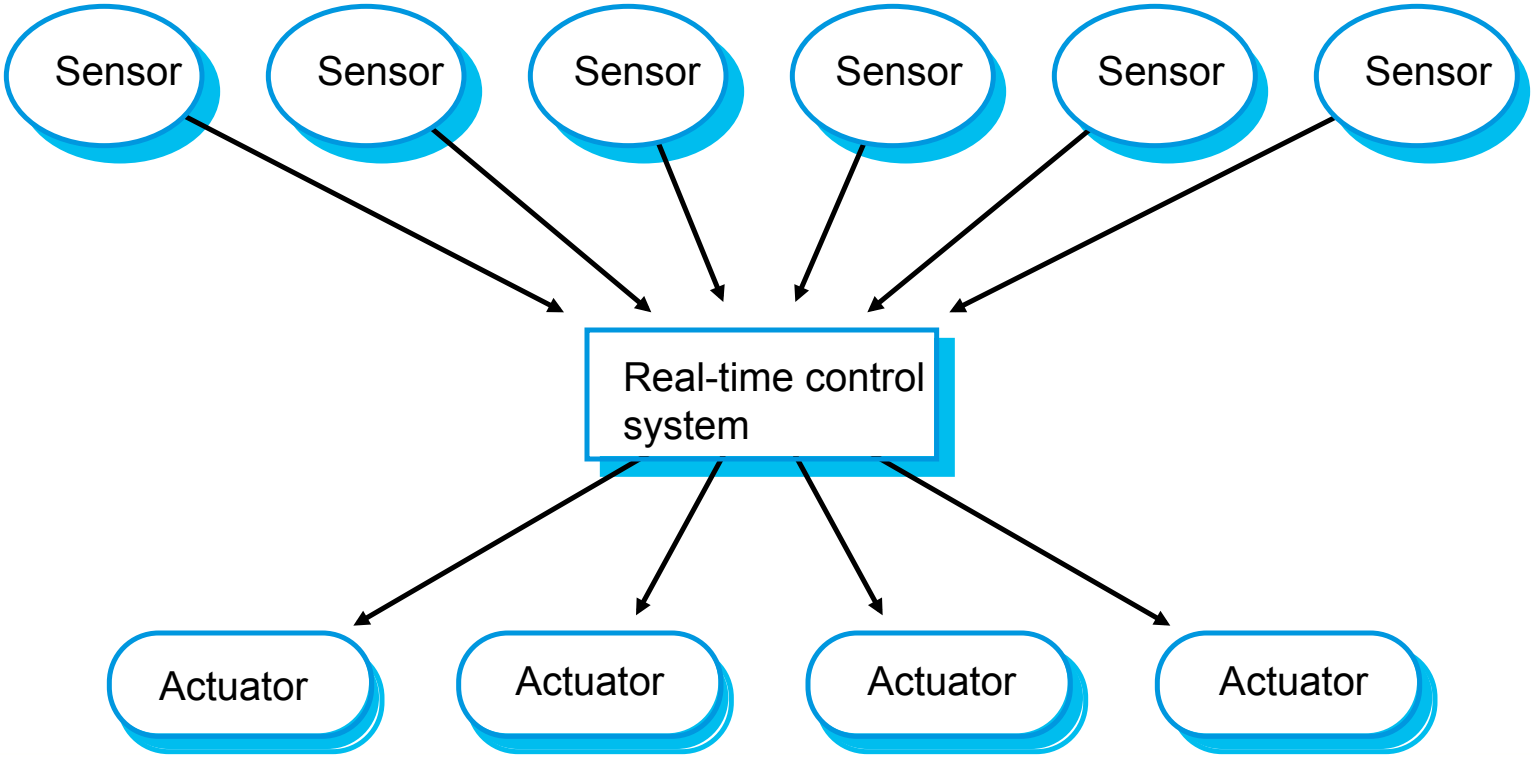
Architectural considerations

- Due to the need to respond correctly 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.

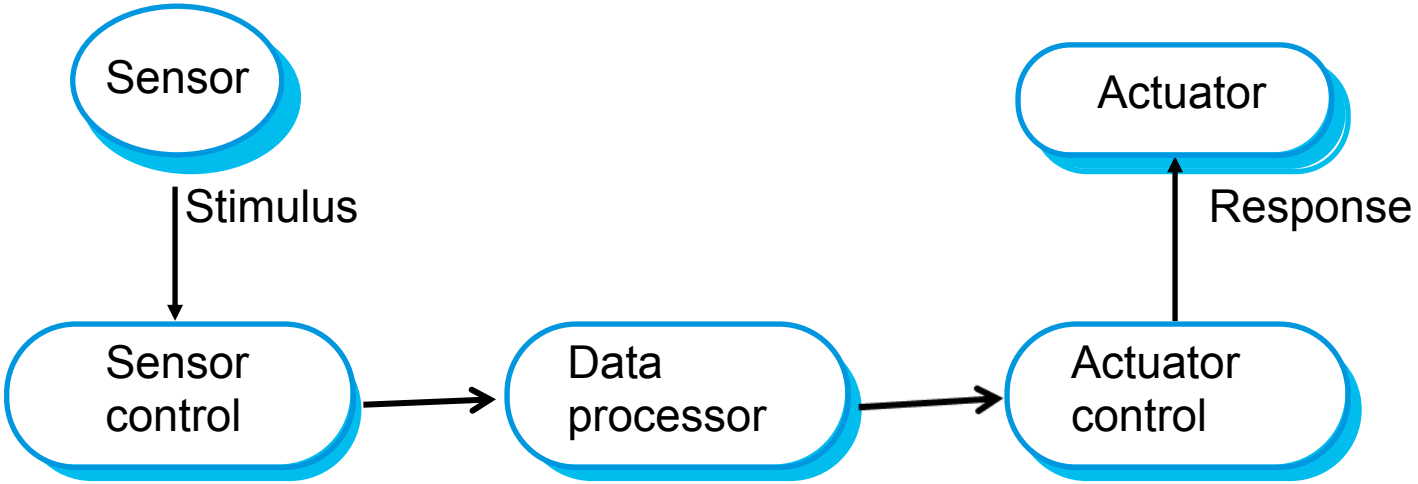
- Real-time systems are therefore usually designed as *concurrent, cooperating processes*

A real-time system model example



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 and computes the system response.
- Actuator control processes
  - Generate control signals for the actuators.

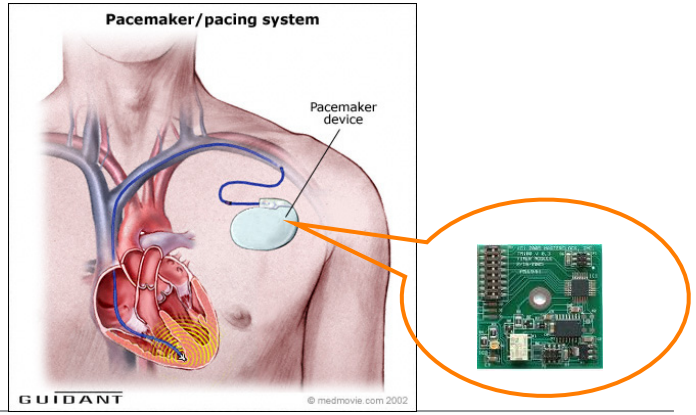


Real-time system programming

- Hard RTS may have to programmed in assembly language to ensure that tight deadlines are met
- System languages such as C allow efficient programs to be written are also widely used
- Systems may also have these language constraints due to system legacy (from both software and hardware)

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 of non-functional system requirements
- Hardware delivers better performance but potentially longer development and less scope for change
- Event-centred



Design process

- Consists several interleaved stages:
  1. Identify the stimuli (events) to be processed and the required responses to these stimuli.
     - E.g., water depth>5meters, alert a warning message
  2. For each stimulus and associated response, identify the timing constraints.
     - E.g., send the warning message in 20 milliseconds
  3. Aggregate the stimulus and response processing into concurrent processes. A process may be associated with each class of stimulus and response.

Design process

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.

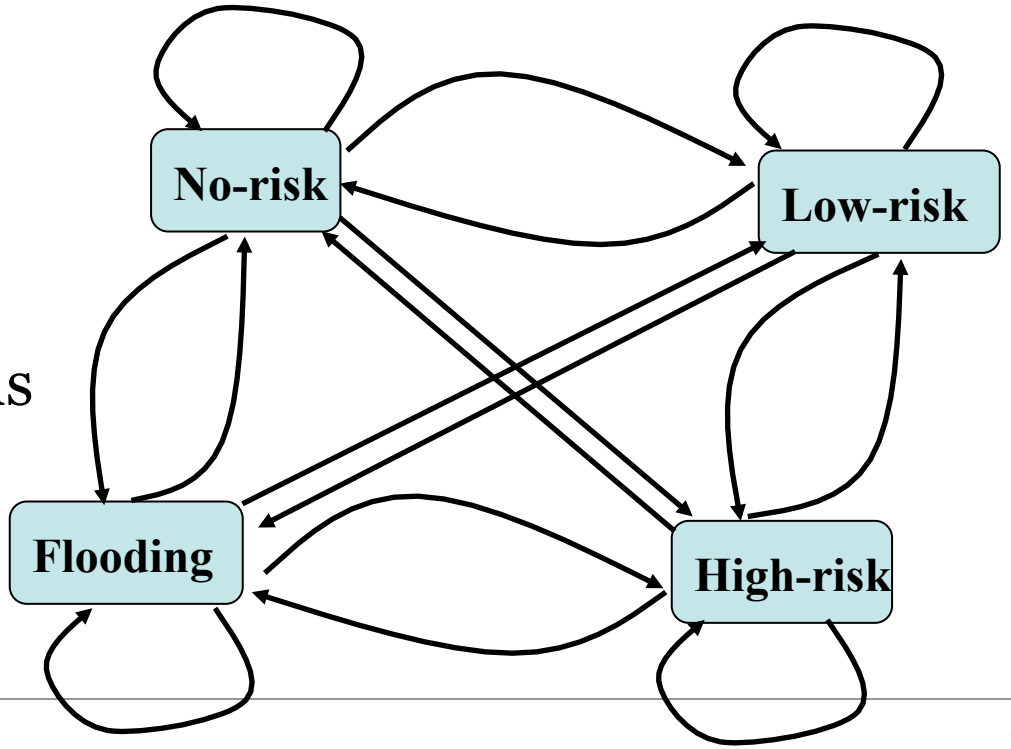
6. Integrate perhaps using a real-time operating system.

System modeling

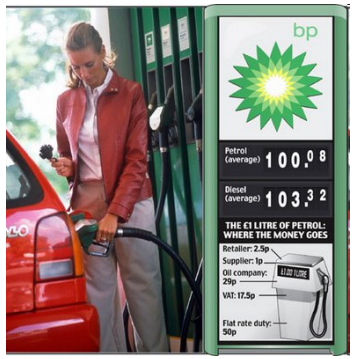
- The effect of a stimulus in a real-time system may trigger a transition from one state to another

- Finite state machines can be used for modelling real-time systems

- A state machine describing transitions of the status of river flooding risk



System modeling example: petrol pump



Real-time operating systems

- Real-time operating systems (RTOS) are *specialised* operating systems which manage the processes in the real-time systems

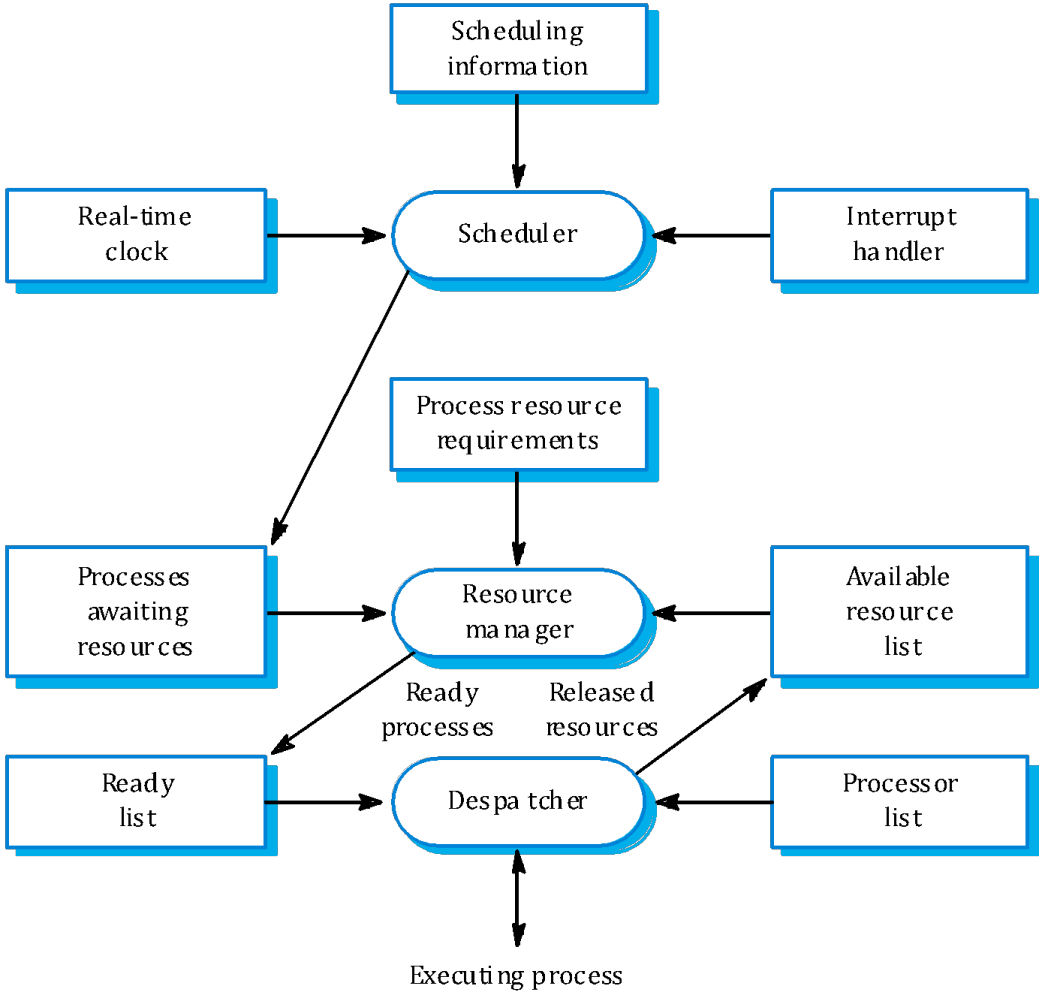
- Responsible for process management and resource (processor and memory) allocation

- 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 operating system components (Cont.)



Process management

- Concerned with managing the set of concurrent processes

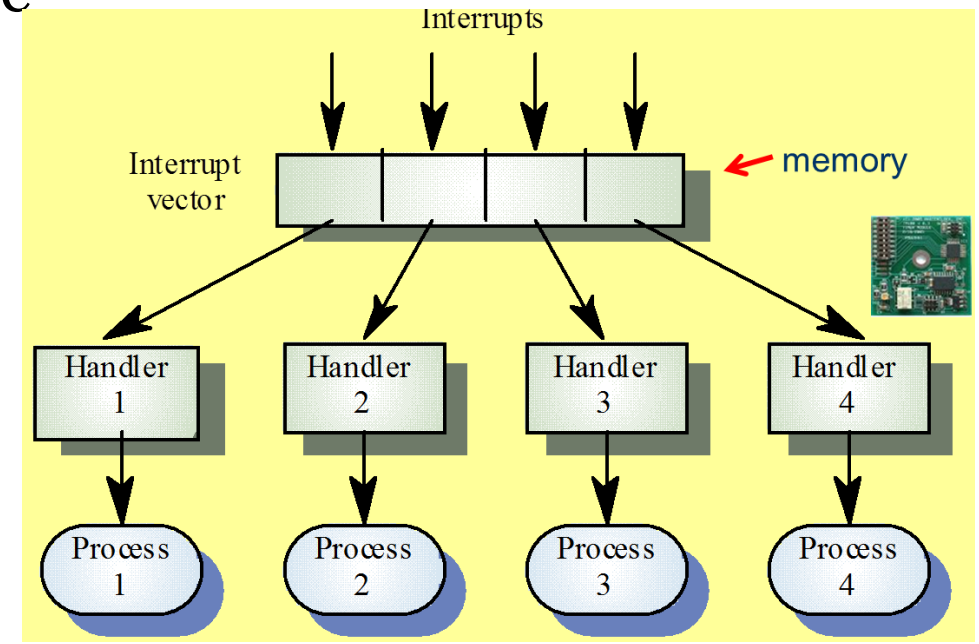
- The processing of some types of stimuli must sometimes take **priority**
  - *Interrupt level priority*. Highest priority which is allocated to processes requiring a very fast response

  - *Clock level priority*. Allocated to periodic processes

- Within these, further levels of priority may be assigned

Interrupt-driven processes

- Processes that respond to asynchronous events
- Control is transferred automatically to a pre-determined memory location
- This location contains an instruction to jump to an interrupt service routine
- Interrupt service routines MUST be short, simple and fast.



Periodic processes

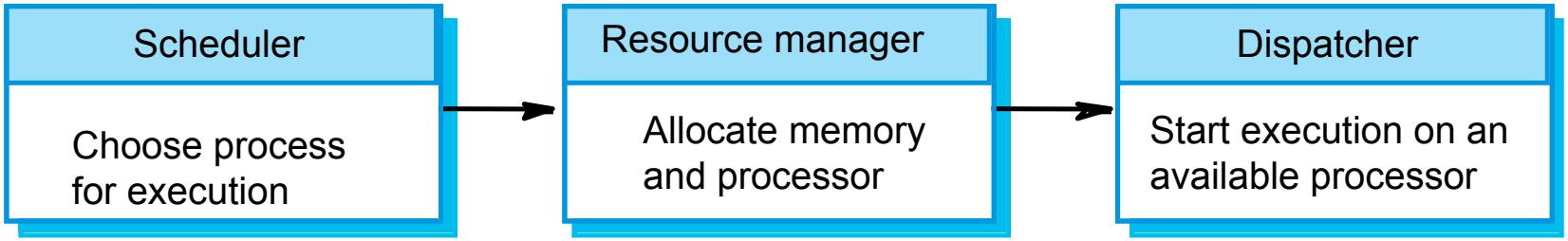
- Periodic processes must be executed at specific time intervals. They have different periods:
  - the time between executions), execution times and deadlines (the time by which processing must be completed).

- 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 (Cont.)

- The 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.
- 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.



Process management (Cont.)

- Two basic 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 input).
  - Pre-emptive scheduling
    - The execution of an executing processes may be stopped if a higher priority process requires service.
- Many scheduling algorithms:
  - Round-robin (each process executed in turn)
  - Rate monotonic (process with the shortest period given priority)
  - Shortest deadline first

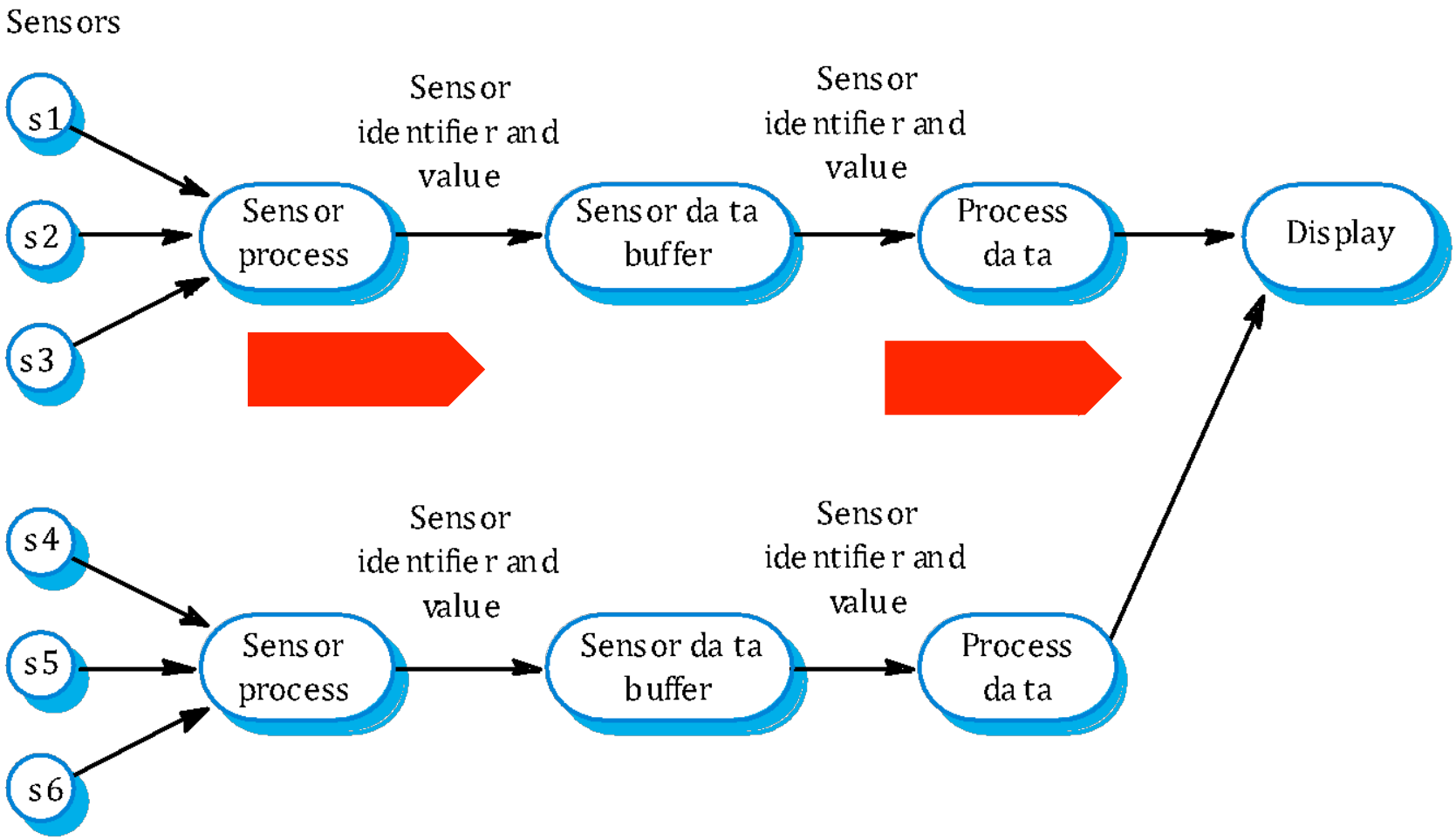
Monitoring and control systems

- Important class of real-time systems
- Continuously check sensors and take actions depending on sensor values
- Monitoring systems examine sensors and report or transform their results
- Control systems take sensor values and control hardware actuators

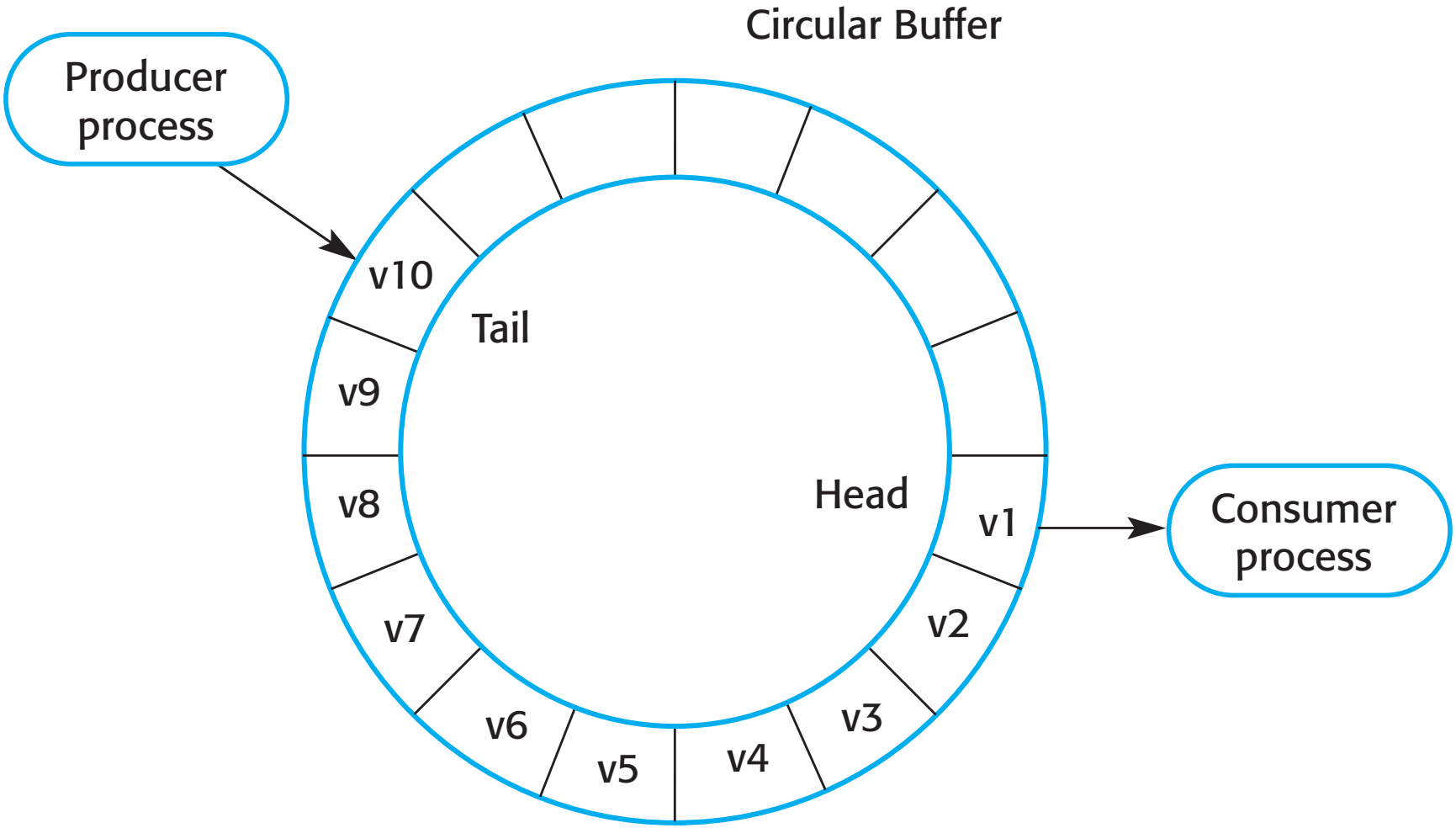
Data acquisition systems

- Collect data from sensors for subsequent processing and analysis
- Data collection processes and processing processes may have different periods and deadlines
- Data collection may be faster than processing
  - e.g. collecting information about an explosion or LHC experiments, high throughput network points
- Circular or ring buffers are a mechanism for smoothing speed differences

Data acquisition systems model example



Data acquisition systems model example: producer/consumer



Mutual exclusion

- Producer processes collect data and add it to the buffer. Consumer processes take data from the buffer and make elements available

- Producer and consumer processes must be mutually excluded from accessing the same element

- The buffer must stop producer processes adding information to a full buffer and consumer processes trying to take information from an empty buffer

Key points revisited

- Real-time system is a software system that must respond to events in real time. Its correctness depends not just on what the system does but also on how fast it reacts.

- A general RT system model involves associating processes with sensors and actuators.

- Real-time systems architectures are usually designed as a number of concurrent processes.

Key points revisited (Cont.)

- Why object-oriented programming languages may not suitable for developing real-time systems?

- An object-oriented approach may result in unacceptable timing delays because structuring a system into objects probably means that there will be a large number of small tasks currently active in a system. The overhead of task communications will slow down the system and may cause timing problems.