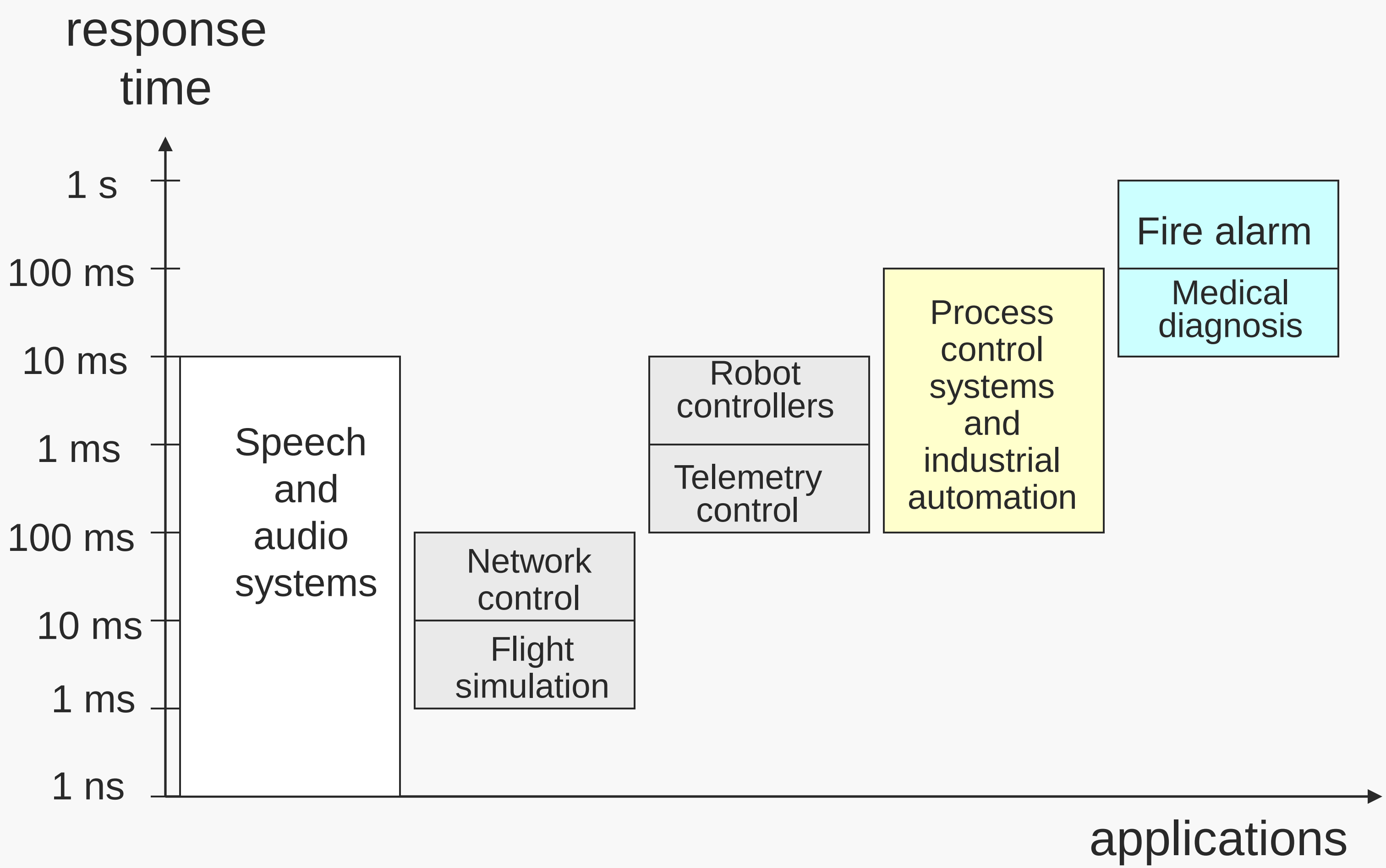
Introduction to Real-Time Systems

Background

- Definitions
  - Real-time systems
    - correctness of system operation depends on temporal characteristics as well as logical and functional characteristics
  - Timing constraints
    - deadline, period, execution time, etc.
  - Real-time applications
    - those that must satisfy timing constraints, typically, hard real-time

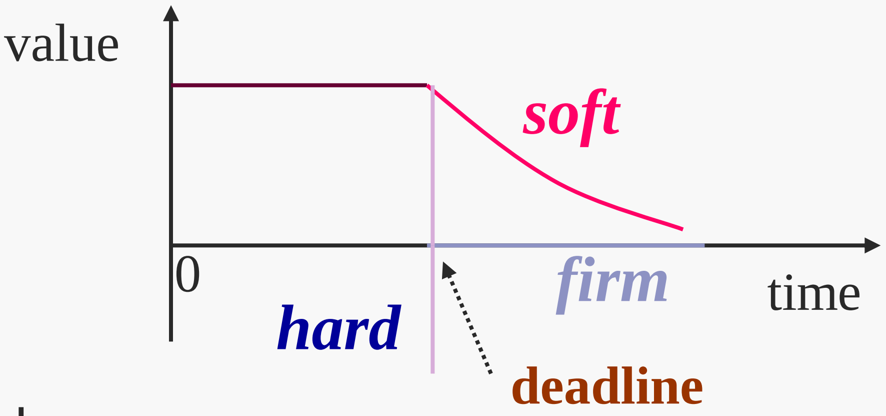
- Embedded computer systems
  - All embedded systems are real-time systems, but not all real-time systems are embedded systems.
  - real-time vs. embedded: often interchangeable (cf. rectangles vs. squares)
- Real-time vs. General-purpose
  - Real-time computer systems differ from their general-purpose counterparts in two important ways.
    (1) They are much more specific in their applications.
    (2) The consequences of their failure are more drastic.

Response time requirements for real-time applications



Task classes

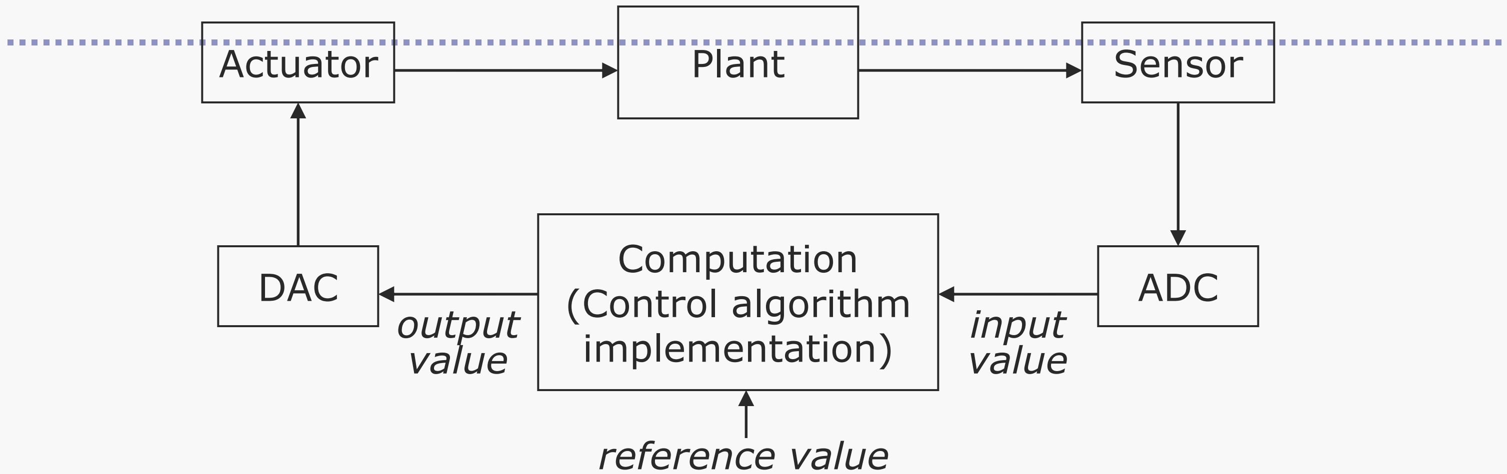
- Task classes
  - hard *vs.* soft *vs.* firm real-time tasks
    - task value functions



  - periodic *vs.* aperiodic tasks
    - *cf*. sporadic tasks: aperiodic tasks with a bounded interarrival time
  - critical *vs.* noncritical tasks

Real-Time Applications

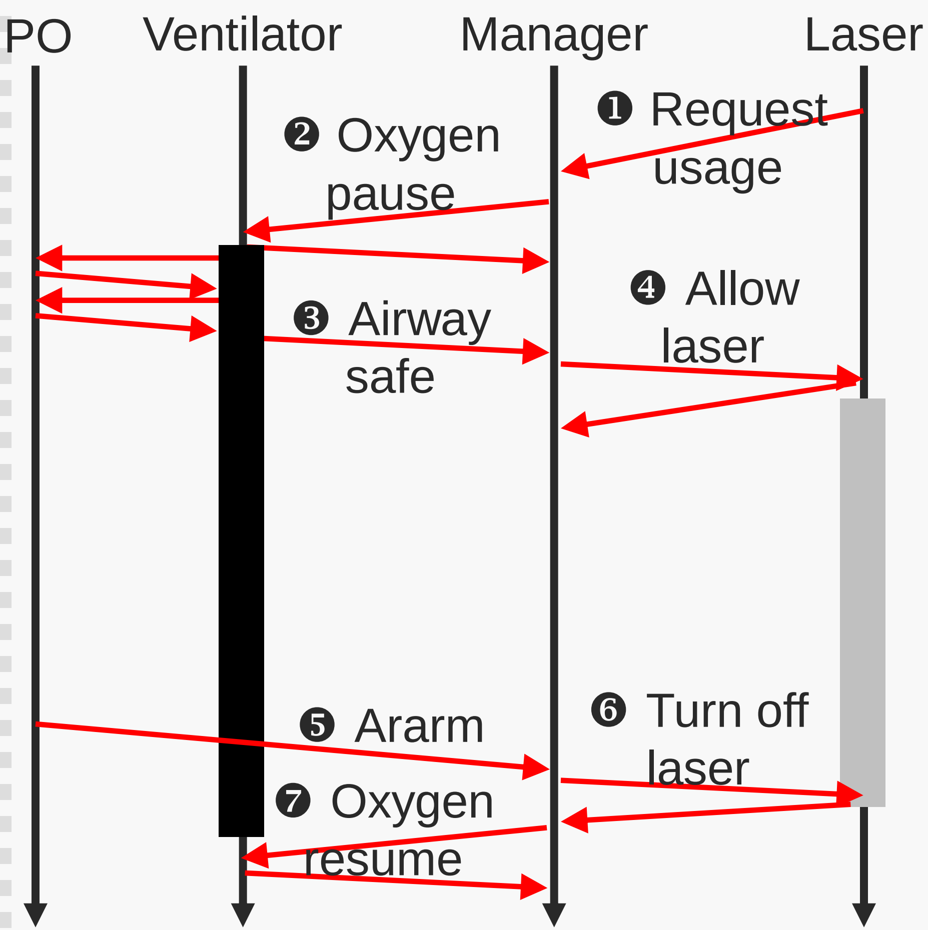
- Industry, defense, weapons
  - Nuclear plants
  - Process control
  - Patient monitoring
  - Fly-by-wire avionics, Spacecraft
  - Guided missile control
  - Telematics
  - Signal processing (e.g. radar)
- Business, entertainment
  - OLTP, OLAP, stock ordering system
  - Multimedia applications (e.g. VOD)



Implementation with an infinite loop: An example

```
initialize I/O ports, internal control variables;
set timer to interrupt periodically with period T;
at each timer interrupt, do
      obtain input;
      compute control output;
      send output to the plant;
end do;
```

Real-Time Patient Control – An example



- Safety requirements
  - The airway laser and the oxygen concentration should not be activated together
  - Airway lasers should not be activated if proportion of oxygen in the airway is higher than a predetermined threshold (e.g. 25%)
  - If patient's SpO2 is lower than threshold, the oxygen concentration must be activated

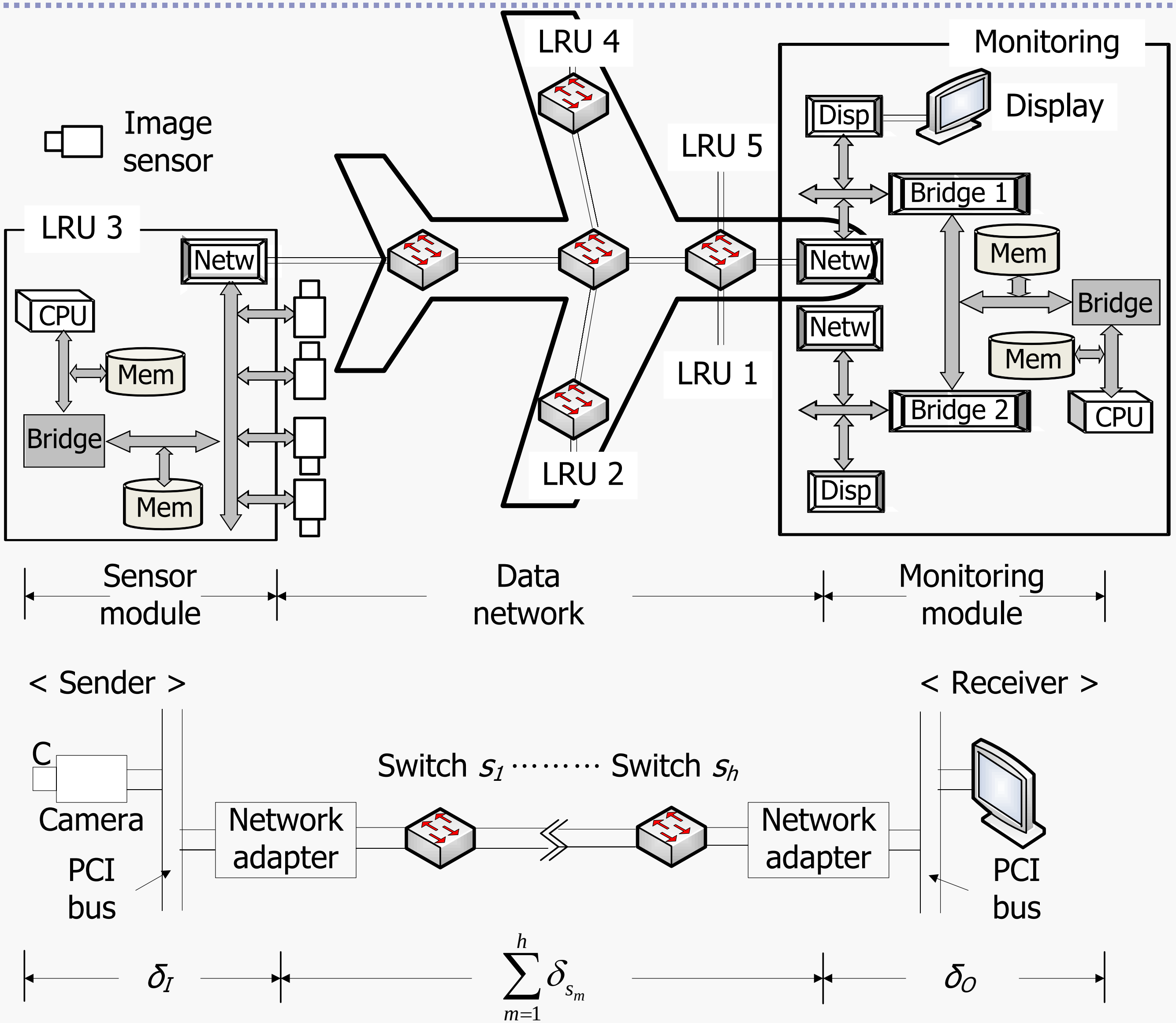
John A. Stankovic, “Misconceptions About Real-Time Computing: A Serious Problem for Next Generation Systems”, IEEE Computer, 1988

Challenges in Real-time Computing

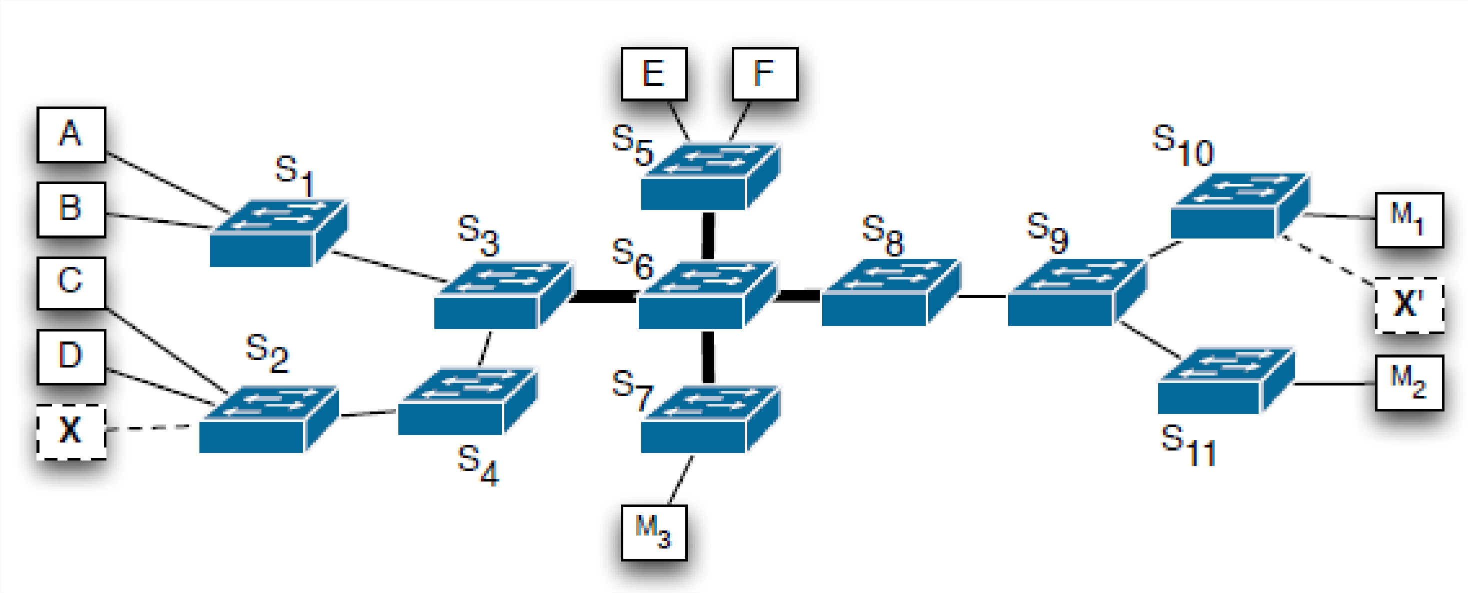
- Current (as of late 1980’s) brute force techniques will not scale to meet the requirements of guaranteeing real-time constraints of next-generation systems.

- Next generation real-time systems will be distributed and capable of exhibiting intelligent, adaptive, and highly dynamic behavior.
  - Rapid advance in hardware
    - Distributed across a network, multiprocessing
  - Artificial intelligence capabilities
    - Impossible to precalculate all possible combinations of tasks that might occur
    - On-line guaranteed and incremental algorithms are needed.

Incremental analysis



Incremental analysis



Misconceptions (Stankovic)

- There is no science in real-time system design.
  - Real-time system design is mostly ad hoc. → a scientific approach is not possible. (?)
  - Counter example
    - Since early 1970's, real-time scheduling and synchronization theories are successfully developed and deployed in commercial real-time operating systems.
      - Priority-driven preemptive scheduling
      - Priority Inheritance protocols
    - Specification of timing constraints at programming language level
      - Ada, Real-time Java

Misconceptions (Con.D)

- Advances in supercomputer hardware will take care of real-time requirements.
  - Advances in supercomputer based on parallel architecture
    - Improve system throughput → timing constraints will be met automatically. (?)
    - In fact, real-time task and communication scheduling problems will likely get worse as more (complex) hardware is used.

Misconceptions (Con.D)

- <u>Real-time computing is equivalent to fast computing.</u>
  - Fast computing → minimize average response time
  - Real-time computing means predictable computing.
    - Functional and timing behavior should be as deterministic as necessary to satisfy system specification.
  - In many cases, real-time computing environment is with very limited hardware resources.
    - The misconception may lead to waste of system resources.
    - → not cost-effective

Misconceptions (Con.D)

- Real-time programming is assembly coding, priority interrupt programming, and device driver writing.
  - Machine-level optimization techniques may be essential to meet tight timing constraint.
    - These techniques are labor intensive and sometimes introduce additional timing assumptions.
      - A major source of bugs
    - May limit maintainability and portability
  - We would like to automate (as much as possible) real-time system design, instead of relying on clever hand-crafted code.

Misconceptions (Con.D)

- Real-time systems research is performance engineering.
  - Performance Engineering
    - Efforts to minimize average response time (raw performance)
  - Robustness (Predictability)
    - One of important features of real-time computing
    - To come up with the worst-case, though it occurs rarely
  - Different development methodology from general-purpose computing
    - Simulate or estimate the worst-case scenario rather than general testing or profiling

Misconceptions (Con.D)

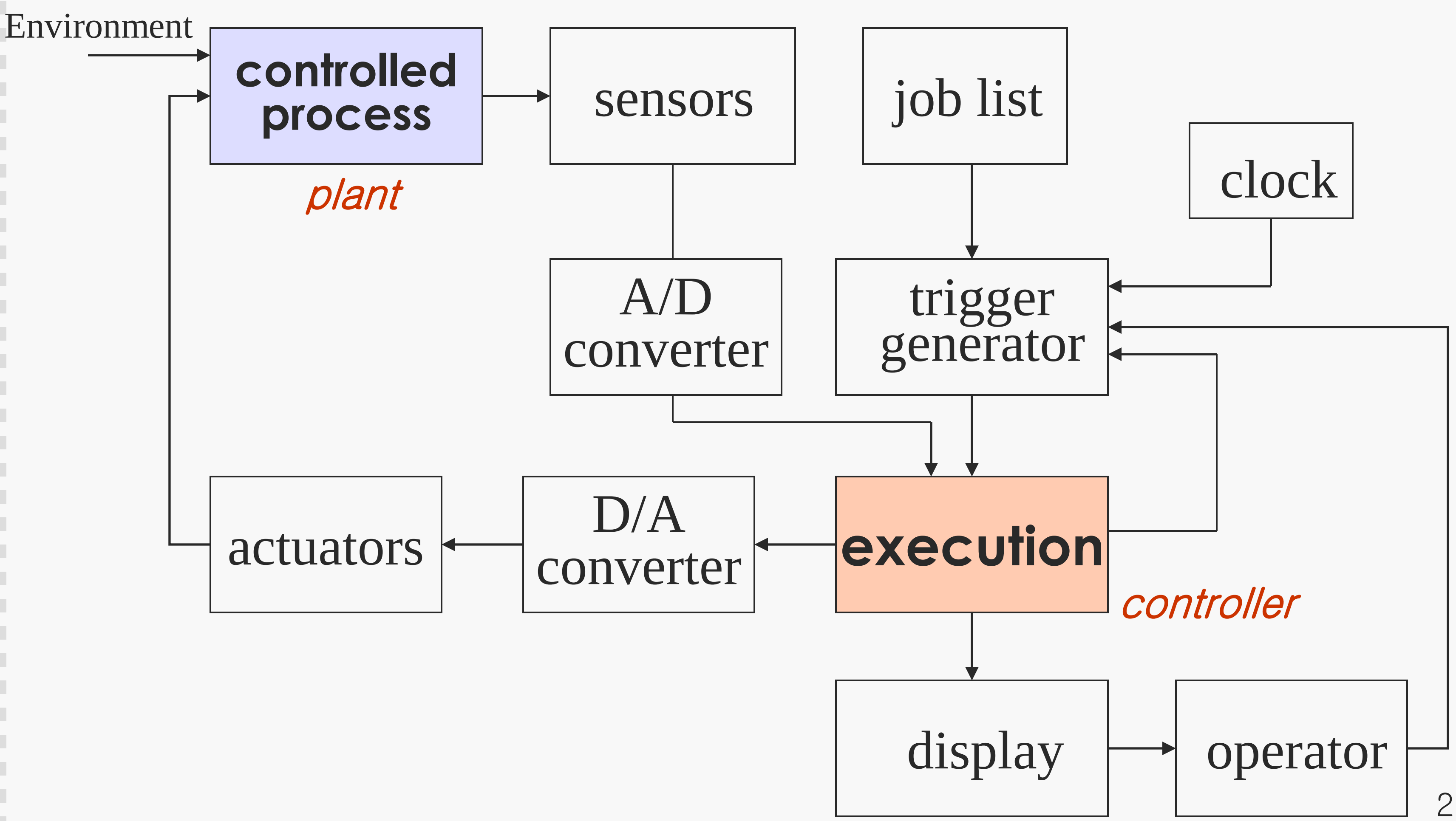
- The problems in real-time system design have all been solved in other areas of computer science or operations research.
  - General-purpose CS researchers are usually interested in optimizing average-case performance.
  - Operation research typically uses stochastic queuing models or one-shot scheduling models to reason about systems.
    - In real-time computing, many tasks recur infinitely either periodically or at irregular intervals, and may have to synchronize or communicate with each other.

Misconceptions (Con.D)

- *It is not meaningful to talk about guaranteeing real-time performance because of imperfect software/hardware/environment.*
  - We certainly cannot guarantee anything outside our control, but what we can guarantee, we should.
- *Real-time systems function in a static environment.*
  - Not true.
  - We consider systems in which the operating mode may change dynamically.
    - Different sets of timing constraints at different time instants

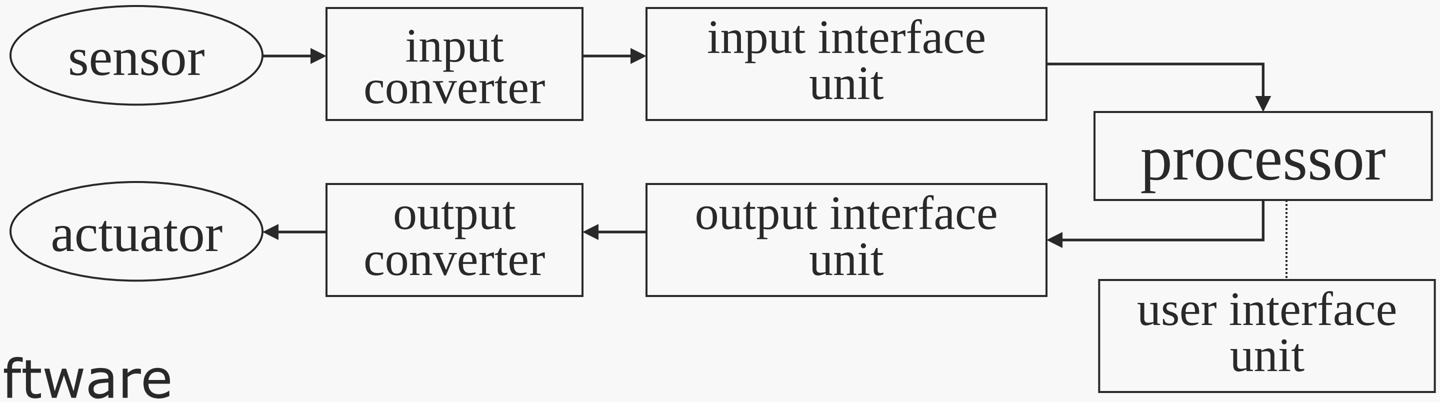
Structure of a Real-Time System

- Logical view

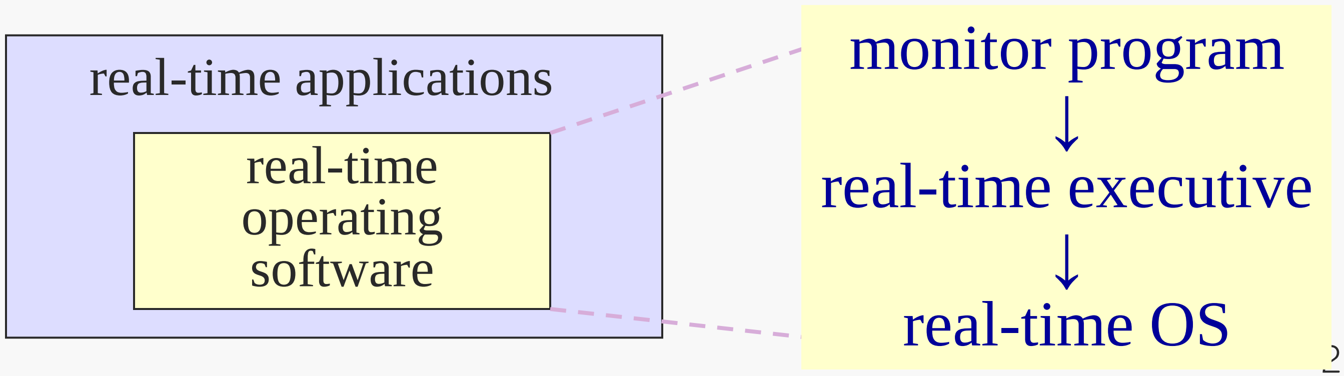


Structure of a Real-Time System

- Structural view
  - hardware



  - software



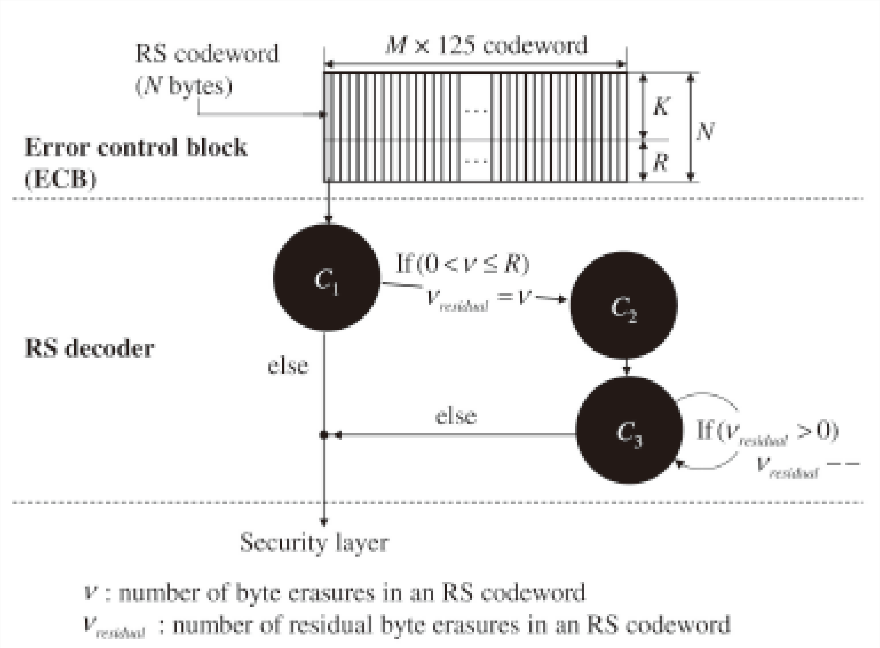
Issues in Real-Time Computing

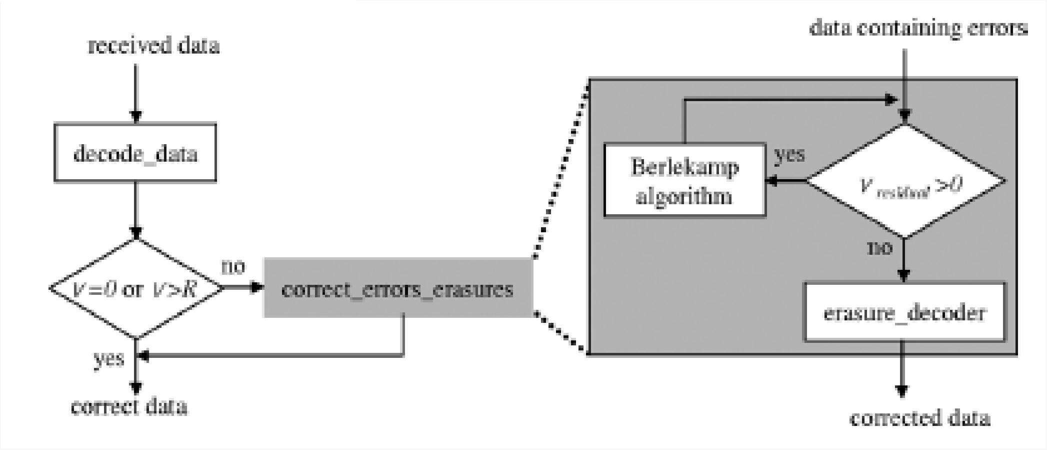
- Specification and verification
  - Modeling and verification of systems that are subject to timing constraints
- Real-Time scheduling theory
  - Meet the specified timing requirements
  - Support the utilization bound to meet all deadlines
  - Meet as many deadlines as possible, if it is impossible to meet all deadlines
- Real-time operating systems
  - Guarantee real-time constraints
  - Support fault tolerance and distribution
  - Scheduling time-constrained resource allocation

Issues in Real-Time Computing

- Real-time programming language and design methodology
  - High level abstraction to deal with complex real-time systems
  - Management of time
  - Schedulability check
  - Reusable real-time software module
- Distributed real-time databases
  - Concurrency in transaction processing
  - Real-time scheduling algorithm

High-level abstraction example





Issues in Real-Time Computing

- Fault tolerance
  - Formal specification of the timing constraints
  - Error handling
- Real-time system architectures
  - Interconnection topology (process, I/O)
  - Fast, reliable, and time-constrained communication
  - Cost-effective and integrated fashion
  - WCET (Worst Case Execution Time) analysis

Issues in Real-Time Computing

- Real-time communication
  - End-to-end deadlines
  - Packet scheduling
  - Dynamic routing
  - Network buffer management
  - CSMA/CA vs. CDMA vs. TDMA
- Embedded computing systems

Real-Time Systems
- Basic Concepts on Real-Time Scheduling

- This presentation is based on the book "Hard Real-Time Computing Systems" by G. Buttazzo.

Contents

- Introduction
- General Scheduling Model
- Task Model
- Scheduling Issues
- Types of Scheduling Algorithm
- Common Approaches in RT Scheduling
- Metrics for Performance Evaluation

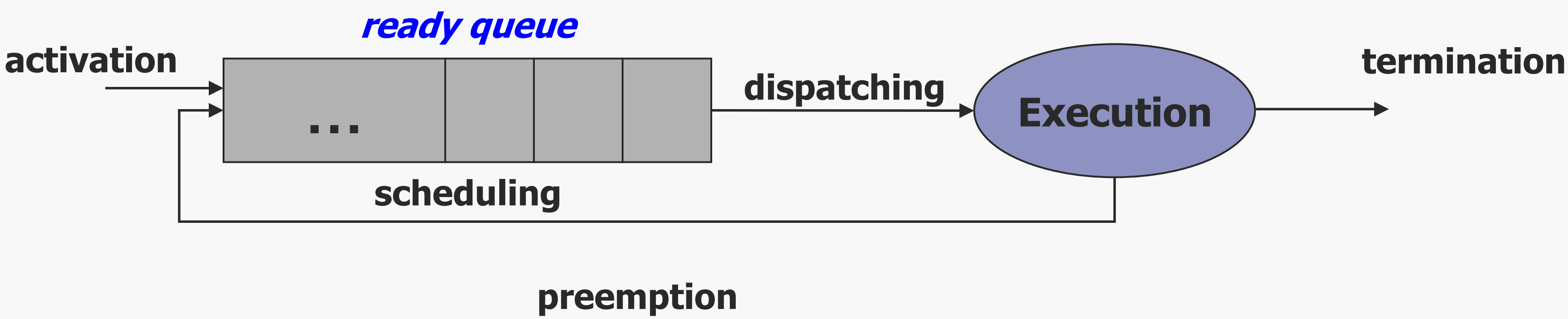
Introduction

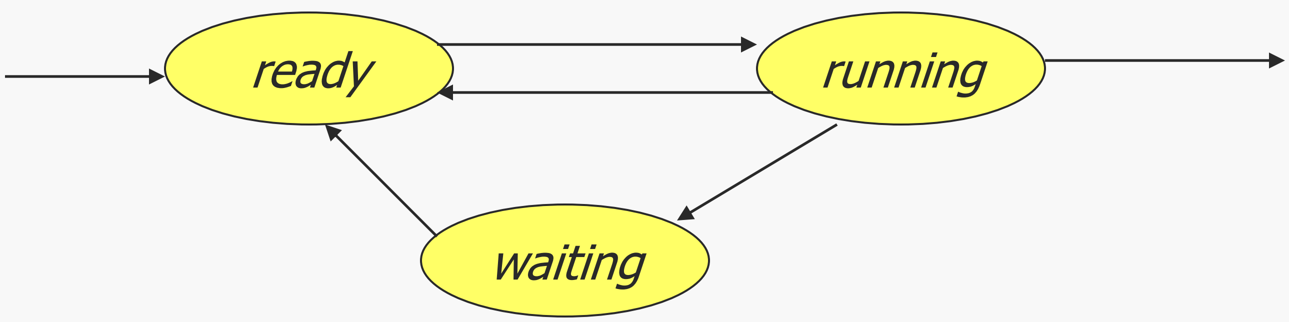
- Task (Process)
  - A computation executed by the CPU in a sequential fashion
- Task instance (Job)
  - A unit of work; A task consists of a sequence of identical jobs.
- Scheduling Algorithm
  - The set of rules that determines the order in which tasks are executed
- Dispatching
  - The specific operation of allocating the CPU to a task selected by the scheduling algorithm
- Scheduler
  - A module implementing scheduling algorithms
- Schedule
  - assignment of all jobs to available processors, produced by scheduler

Introduction (cont'd)

- Feasible Schedule
  - A schedule is said to be ***feasible*** if all tasks can be completed according to the set of specified constraints
  - A set of tasks is said to be ***schedulable*** if there exists at least one algorithm that can produce a feasible schedule
- Optimal Scheduling Algorithm
  - A scheduling algorithm is ***optimal*** if there exists a feasible schedule for a given task set, then the algorithm is able to find it.
- Schedulability Analysis
  - Analyze whether the deadlines of all tasks can be met using a given scheduling policy → predictability

General Scheduling Model





Task Model: Temporal Parameters

- Task model
  - Characterize the behavior of tasks
  - Must be defined to be able to analyze the temporal behavior of a set of tasks
- The temporal behavior of a task is characterized by
  - Static parameters
    - Derived from the specification or implementation of the system
    - Example) period, deadline, worst-case execution time
  - Dynamic parameters
    - Are a functions of the run-time system and the characteristics of other tasks
    - Example) start time, completion time, response time

Task Model: Temporal Parameters

- Release time: r_i
  - The time at which a task becomes ready for execution
  - Sometimes, a range of release time called *release-time jitter* is specified.
  - Also called as arrival time a_i
- Computation (Execution) time: C_i
  - The time necessary to the CPU for executing the task without interruption
  - In general, C_i is the worst case execution time.
- (Absolute) Deadline: d_i
  - The time before which a task should be complete to damage to the system
- (Relative) Deadline: D_i
  - Deadline relative to the release time

Task Model: Temporal Parameters

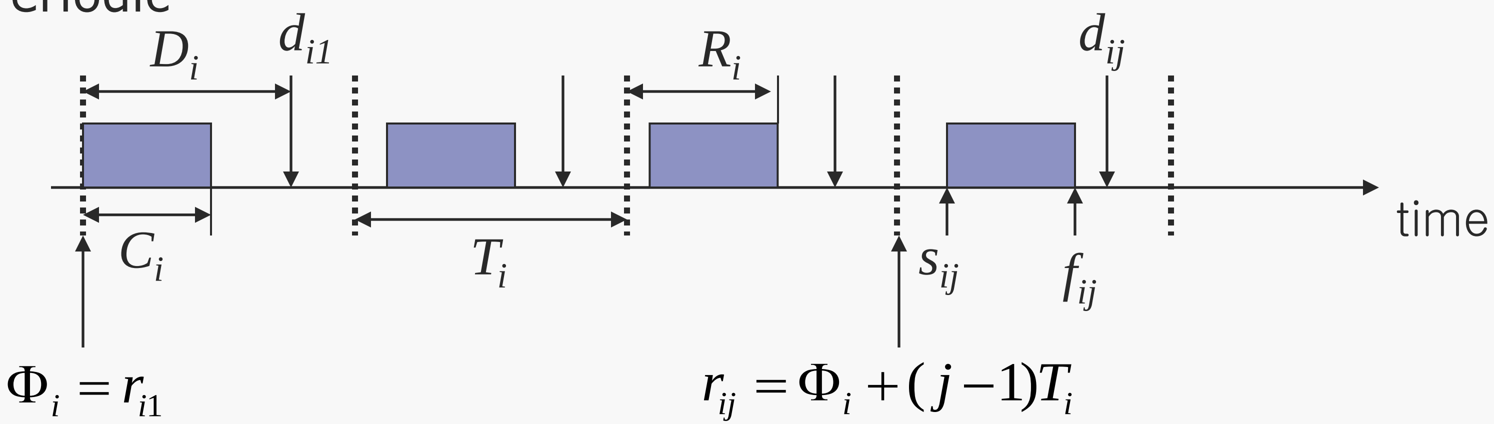
- Start time: s_i
  - The time at which a task actually starts its execution
- Finishing (completion) time: f_i
  - The time at which a task finishes its execution
- Response time: R_i
  - The time interval between the release time (r_i) and the finishing time (f_i) of a task instance
- Period: T_i
  - The time interval between two consecutive activations of task
- Laxity (Slack time): $X_i = d_i - r_i - C_i$
  - The maximum time a task can be delayed on its activation to complete within its deadline

Task Model: Temporal Parameters

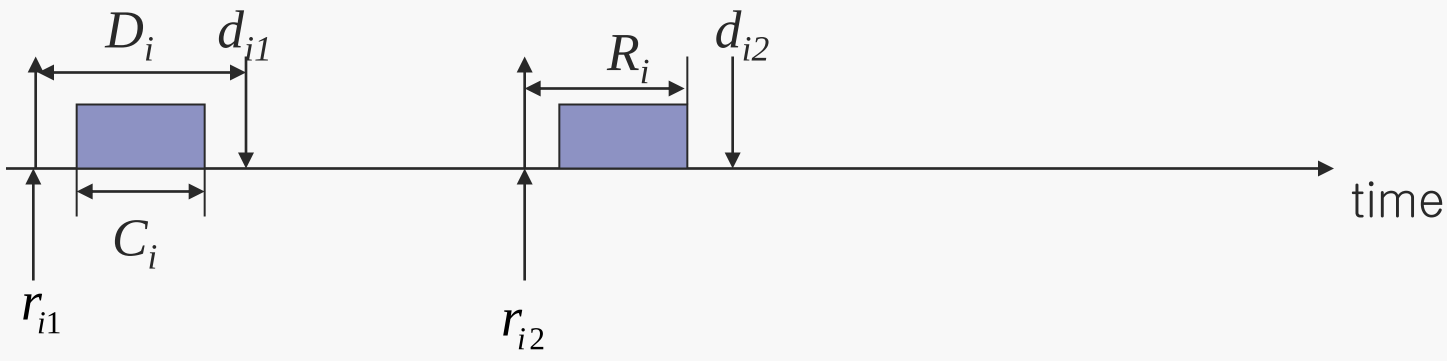
- Types of tasks
  - Consequences of deadline miss
    - Hard
    - Soft
  - Regularity of activation
    - Periodic
      - A type of tasks that consists of a sequence of identical instances, *activated at regular intervals*
    - Aperiodic
      - A type of tasks that consists of a sequence of identical instances, *activated at irregular intervals*
    - Sporadic
      - An *aperiodic task* characterized by *a minimum interarrival time* between consecutive instances

Task Model: Temporal Parameters

- Periodic



- Aperiodic



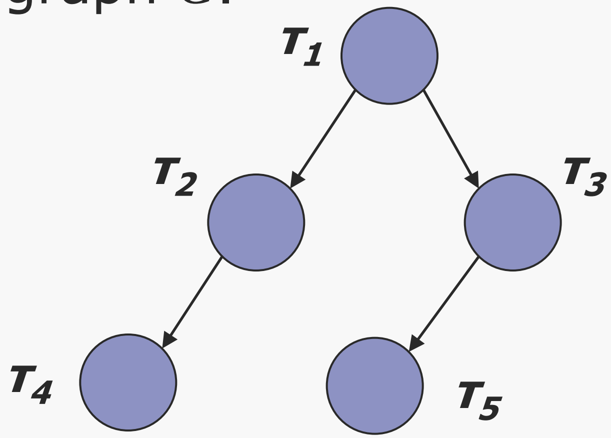
Task Model: Temporal Parameters

- More Terminologies on Periodic Task Systems
  - Critical Instant
    - The time instant at which the release of a task instance produces the largest response time
    - E.g) If all tasks are independent, the time instant at 0 is the critical instant.
  - Hyperperiod
    - The least common multiple (LCM) of T_i
  - Task Utilization: $U_i = C_i / T_i$
  - System Utilization:

$$U = \sum_{i \in [1,n]} U_i = \sum_{i \in [1,n]} \frac{C_i}{T_i}$$

Task Model: Precedence Constraints

- Reflects data and control dependencies.
  - For example, task τ_i may be constrained to be released only after task τ_j completes.
  - Precedence is typically modeled as a partial order relation $<$
    - $\tau_i < \tau_j$: τ_i *is a predecessor of* τ_j
  - Precedence relations are described through a directed acyclic graph G.

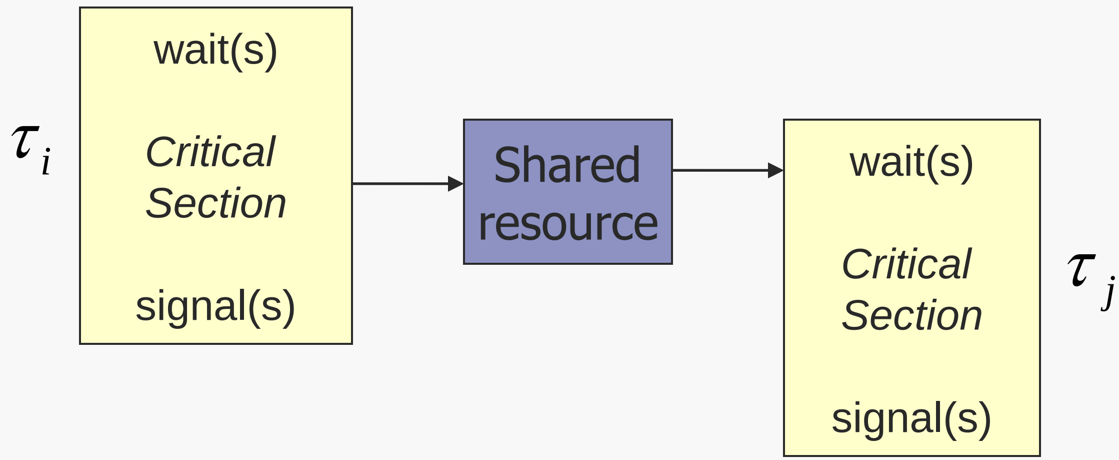


- Tasks with no dependencies are called *independent*.

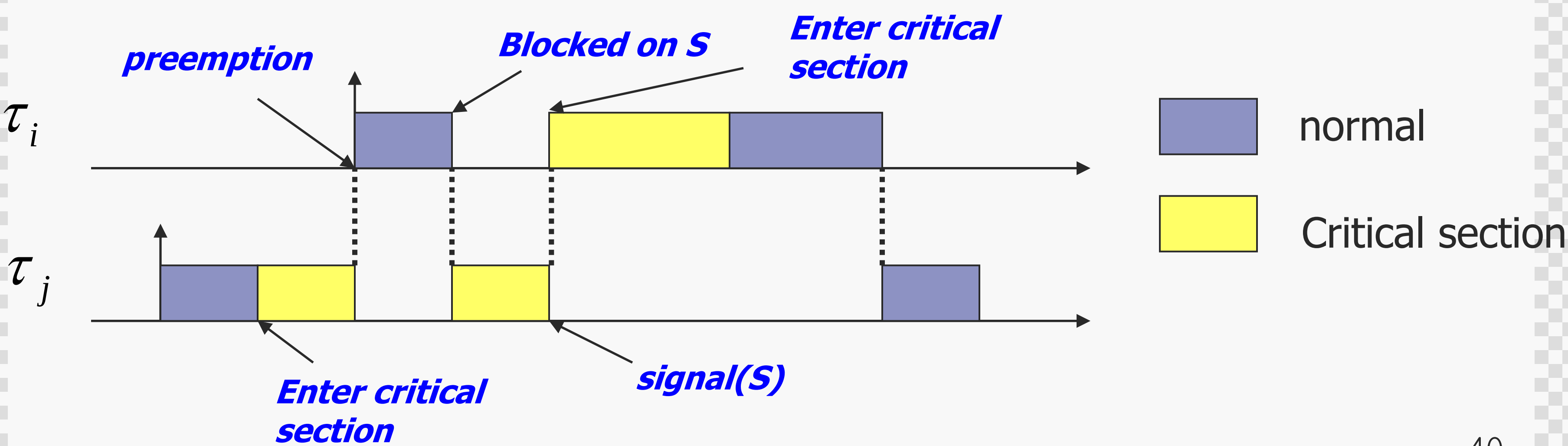
Task Model: Resource Constraints

- Resource
  - Any entity (processor, memory, function, data, …) that can be used by tasks to carry on their computation
- Resource Constraint
  - Dependency relation among tasks that share a common resource used in exclusive mode
  - In real-time scheduling domain, in many cases, the shared resources considered are critical sections or shared data.
  - Accesses to the shared resources are protected via some synchronization mechanism usually provided by OS.

Task Model: Resource Constraints



Potential Problem: Priority Inversion



Task Model: Functional Parameters

- Preemptivity
  - Preemption: Suspension of execution of job to give processor to more urgent job.
- Criticality
  - Can associate weight with jobs to indicate criticalness with respect to other jobs.
  - Schedulers and resource access protocols then optimize weighted performance measures.

Scheduling Issues

- Scheduling of independent periodic tasks
- Scheduling of dependent tasks
- Scheduling with dynamic priority assignment
- Hybrid task set scheduling
- Scheduling schemes for handling overload
- Multiprocessor scheduling
- Joint scheduling of tasks and messages in distributed systems

Types of Scheduling Algorithms

- Preemptive vs. Non-preemptive
  - Preemptive
    - The running task can be interrupted at any time to assign the processor to anther active task, according to a predefined scheduling policy
  - Non-preemptive
    - A task, once started, is executed until completion.
- Static vs. Dynamic
  - Static
    - Scheduling decisions are made on fixed parameters, assigned to tasks before their activation.
  - Dynamic
    - Scheduling decisions are made on dynamic parameters that may change during system evolution.

Types of Scheduling Algorithms

- Off-line vs. On-line: : When are schedules are generated ?
  - Off-line
    - A schedule is generated before actual task activation
    - The schedule is stored in a table and later executed by a dispatcher
  - On-line
    - Scheduling decisions are taken at runtime every time a new task enters the system or when a running task terminates
- Optimal vs. Heuristic
  - Optimal
    - An algorithm is said to be optimal if it minimizes somes given cost function defined over the task set.
  - Heuristic
    - An algorithm is said to be heuristic if it tends toward but does not guarantee to find the optimal schedule.

Common Approaches in RT Scheduling

- Clock-Driven
  - Determines which job to execute when all parameters of hard real-time tasks are fixed and known
  - Schedule is stored in a table
  - Scheduler invoked by a timer
- Priority-Driven
  - Assigns priorities to tasks and executes tasks in priority
  - Priorities are assigned off-line or on-line
    - Static priority: RM (Rate Monotonic), DM (Deadline Monotonic)
    - Dynamic priority: EDF (Earliest Deadline First)

Metrics for Performance Evaluation

- Why do we evaluate performance ?
  - To evaluate different design solutions and choose the best one among them
- How can we do it ?
  - Quantify system performance
    - Choose useful performance measures (metrics)
  - Perform objective performance analysis
    - Choose suitable evaluation methodology
    - Examples: theoretical and/or experimental analysis
  - Compare performance of different designs
    - Make trade-off analysis using chosen performance measures
  - Identify fundamental performance limitations
    - Find “bottleneck” mechanisms that affect performance

Metrics for Performance Evaluation

- Traditional Performance Metrics
  - Throughput
    - Average number of
  - Response Time
    - Average response time between the release time and the completion time of a job
  - Reliability
    - Probability that system will not fail in a given time interval
  - Availability
    - Fraction of time for which system is in action

  → Does not consider deadlines

Metrics for Performance Evaluation

- Terminologies
  - Value: v_i
    - The relative importance of the task with respect to the other tasks in the system
  - Weighted sum completion time: $t_w = \sum_{i=1}^{n} w_i f_i$
  - Lateness: $L_i = f_i - d_i$
    - The delay of a task completion with respect to its deadline
  - Tardiness (Exceeding time): $E_i = max(0, L_i)$
    - The time a task stays active after its deadline

Metrics for Performance Evaluation

- Candidates for Real-time Performance Metrics
  - Maximize completion ratio / minimize miss ratio
    - For soft real-time tasks
  - Maximize total value
    - Minimize weighted sum of completion times
  - Minimize the maximum lateness
    - Useful at design time
    - Does not guarantee that no task misses its deadline
  - Minimize error for imprecise tasks
    - In imprecise computation, a task consists of a mandatory part and an optional part.
    - Error in imprecise computation: $\varepsilon_i = o_i - \sigma_i$

Reading assignment

[Liu73] C. L. Liu and J.W. Layland, "Scheduling algorithms for multiprogramming in a hard real-time environment", Journal of the ACM, 1973

[Baruah90] S. K. Baruah, L.E.Rosier, and R.R. Howell, "Algorithms and complexity concerning the preemptive scheduling of periodic, real-time tasks on one processor", Journal of Real-time Systems, 1990

[Jeffay93] K. Jeffay and D.L. Stone, "Accounting for interrupt handling costs in dynamic priority task systems", RTSS, 1993