Learning Module II: Real-Time Systems Design

Outline

- Introduction to real-time systems
- Timing requirements and timing analysis
  - Concept of Worst-Case Execution Time (WCET)
  - Why it is hard for analyzing WCET?
  - Overall approach: modularization
  - Program path analysis
  - Static analysis
- Uniprocessor scheduling algorithms
  - Task models
  - Performance metrics of scheduling algorithms
  - Static scheduling algorithms
  - Dynamic scheduling algorithms
- Multicore and distributed real-time systems
  - Multiprocessor scheduling
  - Distributed real-time systems
- What happens when things go wrong?

Suggested Readings

Books:

Giorgio C. Buttazzo, “Hard Real-Time Computing Systems: Predictable Scheduling Algorithms and Applications”, Springer, Second Edition, 2004. ISBN: 0-387-23137-4

Jane W.S. Liu, “Real-Time Systems,” Prentice Hall, First Edition, 2000. ISBN:0130996513

John A. Stankovic, Marco Spuri, Krithi Ramamritham, Giorgio C Buttazzo, “Deadline Scheduling for Real-Time Systems: EDF and Related Algorithms”, Springer Science & Business Media, 1998. ISBN 978-1-4615-5535-3

Papers:

Robert I. Davis and Alan Burns. 2011. A survey of hard real-time scheduling for multiprocessor systems. ACM Comput. Surv. 43, 4, Article 35 (October 2011), 44 pages.

Sanjoy K. Baruah, Kirk Pruhs: Open problems in real-time scheduling. J. Scheduling 13(6): 577-582 (2010)

Outline

- Introduction to real-time systems
- Timing requirements and timing analysis
  - Concept of Worst-Case Execution Time (WCET)
  - Why it is hard for analyzing WCET?
  - Overall approach: modularization
  - Program path analysis
  - Static analysis
- Uniprocessor scheduling algorithms
  - Task models
  - Performance metrics of scheduling algorithms
  - Static scheduling algorithms
  - Dynamic scheduling algorithms
- Multicore and distributed real-time systems
  - Multiprocessor scheduling
  - Distributed real-time systems
- What happens when things go wrong?

INTRODUCTION TO REAL-TIME SYSTEMS

- A **real-time system** is a system whose specification includes both logical and temporal correctness requirements.
- Logical correctness (“the results are correct”)
    - Requires functional analysis
- Temporal correctness (“the results are delivered in/on time”)
    - Requires non-functional analysis
- High reactivity and high dependability are more important than performance

- In these four lectures, we focus on techniques and technologies for achieving and checking temporal correctness.

INTRODUCTION TO REAL-TIME SYSTEMS

When we design real-time systems, we need to consider how much system resources we have to realize the timing requirements of the applications.

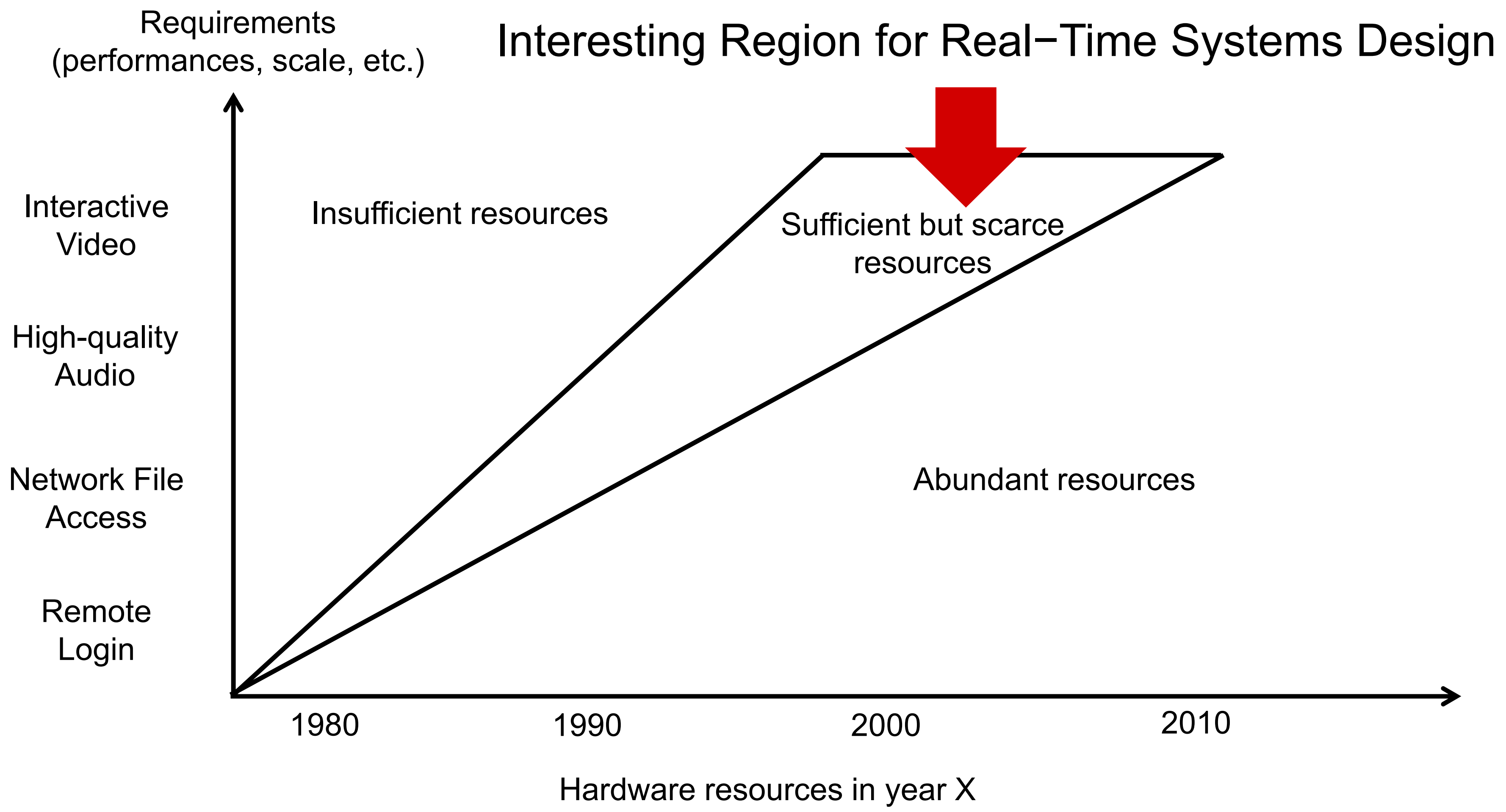
The “Window of Scarcity”

Resources may be categorized as:

- **Abundant:** Virtually any system design methodology can be used to realize the timing requirements of the application.
- **Insufficient:** The application is ahead of the technology curve; no design methodology can be used to realize the timing requirements of the application.
- **Sufficient but scarce:** It is possible to realize the timing requirements of the application, but careful resource allocation is required.

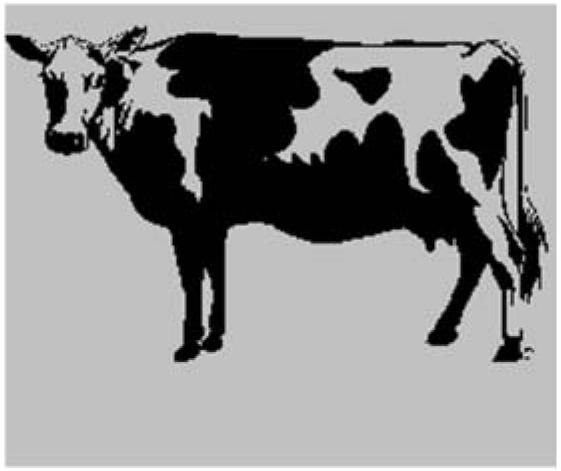
INTRODUCTION TO REAL-TIME SYSTEMS

Scarcity of Resources

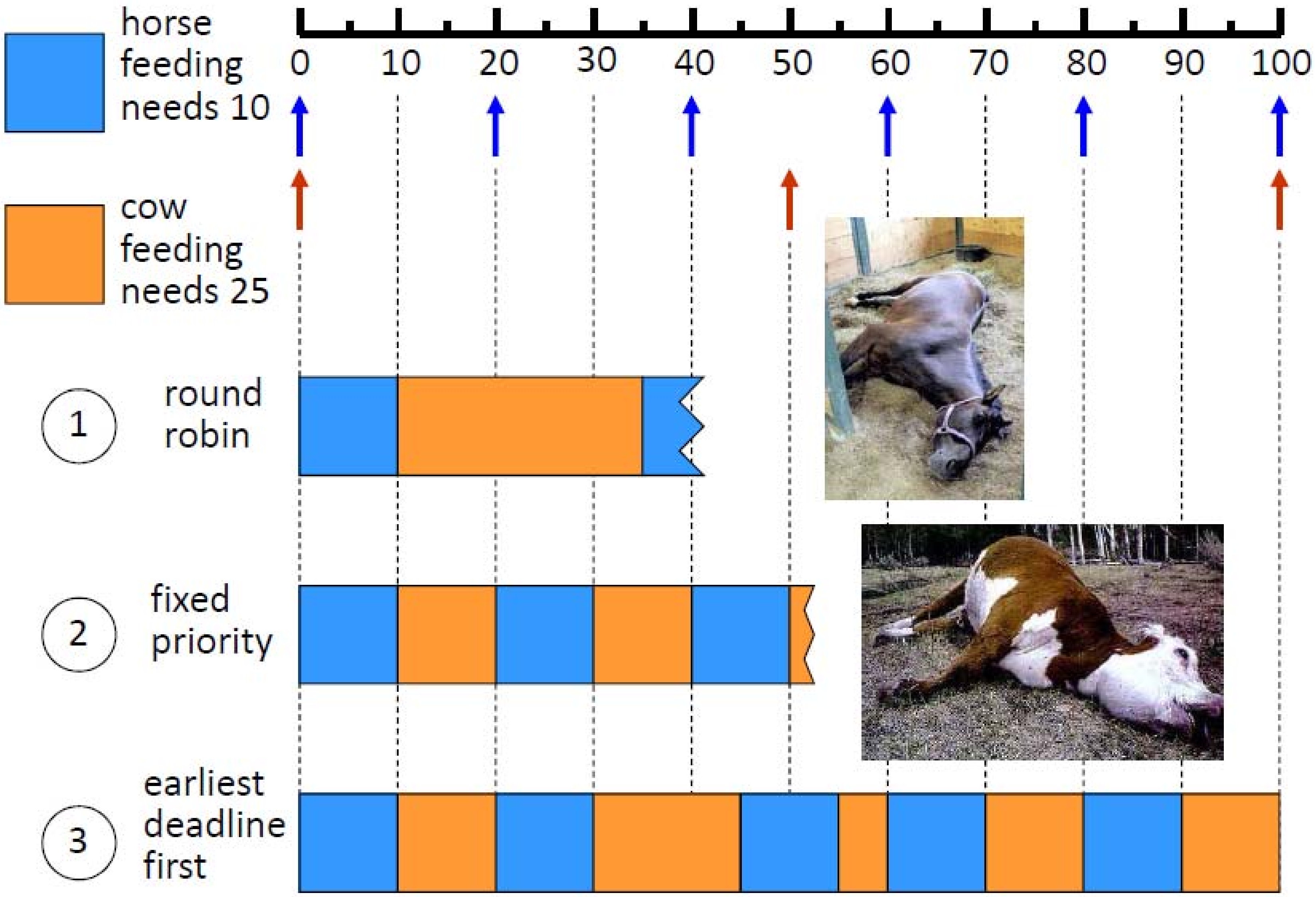


The Cowboy Scheduling Problem

Feed horse periodically every 20 minutes for 10 minutes each period



Feed cow periodically every 50 minutes for 25 minutes each period


horse feeding needs 10
cow feeding needs 25
0
10
20
30
40
50
60
70
80
90
100
1 round robin
2 fixed priority
3 earliest deadline first

INTRODUCTION TO REAL-TIME SYSTEMS

- Examples for Real-Time Systems
  - Chemical & Nuclear Power Plants
  - Railway Switching Systems
  - Flight Control Systems
  - Space Mission Control
  - Automotive Systems
  - Healthcare Systems
  - Robotics
  - Telecommunications Systems
  - Stock Market and Trading System
  - Multimedia Systems
  - Virtual Reality

  . . .

Hard Real-Time Systems

Catastrophic results if some deadlines are missed

Firm Real-Time Systems

The results are useless if the deadlines are missed

Soft Real-Time Systems

The results are not very useful if the deadlines are missed

INTRODUCTION TO REAL-TIME SYSTEMS

- Typical Characteristics of Real-Time Systems
  - Timeliness
  - High cost of failure
  - Concurrency/multiprogramming
  - Design for worst cases
  - Reliability/fault-tolerance requirements
  - <u>Predictable behavior</u>

INTRODUCTION TO REAL-TIME SYSTEMS

Frequent Misconceptions about Real-Time Systems

- There is no science in real-time system design.
  - We shall see in the following lectures.
- Real-time computing is equivalent to fast computing.
  - Real-time computing means <u>predictable</u> and <u>reliable</u> computing.
- “Real-time” is performance engineering/tuning.
  - Timeliness is more important than raw performance.
- Advances in hardware will take care of real-time requirements.
  - Buying a “faster” processor may result in timeliness violation.

INTRODUCTION TO REAL-TIME SYSTEMS

Frequent Misconceptions about Real-Time Systems (Cont.)

- Real-time programming is assembly coding.
  - We would like to automate (as much as possible) real-time systems design
- “Real-time problems” have all been solved in other areas of computer sciences and operations research.
  - OR people typically use stochastic queuing models or one-shot scheduling models to reason about systems.
  - In other CS areas, people are usually interested in optimizing average-case performance.
- Real-time systems function only in a static environment.
  - We also consider systems in which the environment may change dynamically.

INTRODUCTION TO REAL-TIME SYSTEMS

The Focus of these four lectures:

- Scheduling Algorithms for Real-Time Systems
  - Earliest-Deadline-First Scheduling, Rate Monotonic Scheduling, Deadline Monotonic Scheduling, etc. in uniprocessor system
  - Global scheduling and partitioned scheduling in multiprocessor systems
- Analysis of Timeliness in Real-Time Systems
  - Worst-case Execution Time (WECT) analysis
  - Schedulability analysis of scheduling algorithms

After these four lectures, you are expected to know

- Fundamental scheduling theories in real-time systems
- Schedulability analysis of scheduling algorithms in real-time systems

Outline

- Introduction to real-time systems
- Timing requirements and timing analysis
  - Concept of Worst-Case Execution Time (WCET)
  - Why it is hard for analyzing WCET?
  - Overall approach: modularization
  - Program path analysis
  - Static analysis
- Uniprocessor scheduling algorithms
  - Task models
  - Performance metrics of scheduling algorithms
  - Static scheduling algorithms
  - Dynamic scheduling algorithms
- Multicore and distributed real-time systems
  - Multiprocessor scheduling
  - Distributed real-time systems
- What happens when things go wrong?

IMPORTANT QUESTIONS FOR REAL-TIME SCHEDULING

1. What scheduler is guaranteed to meet all task deadlines for a given workload?

2. Given a scheduler, how do we know that it will work for a given workload?

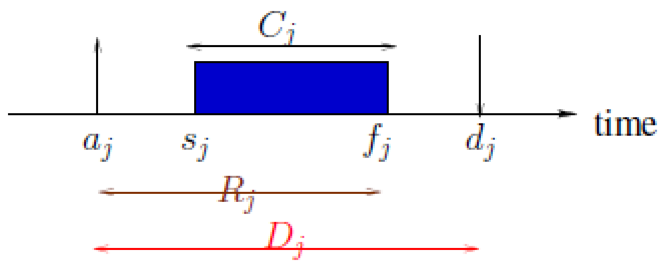
3. Is there an “optimal” scheduler independent of workload?

FUNDAMENTALS

- Algorithm:
  - It is the logical procedure to solve a certain problem
  - It is informally specified a sequence of elementary steps that an “execution machine” must follow to solve the problem
  - It is not necessarily (and usually not) expressed in a formal programming language
- Program:
  - It is the implementation of an algorithm in a programming language
  - It can be executed several times with different inputs
- Process/job/task:
  - An instance of a program that given a sequence of inputs produces a set of outputs

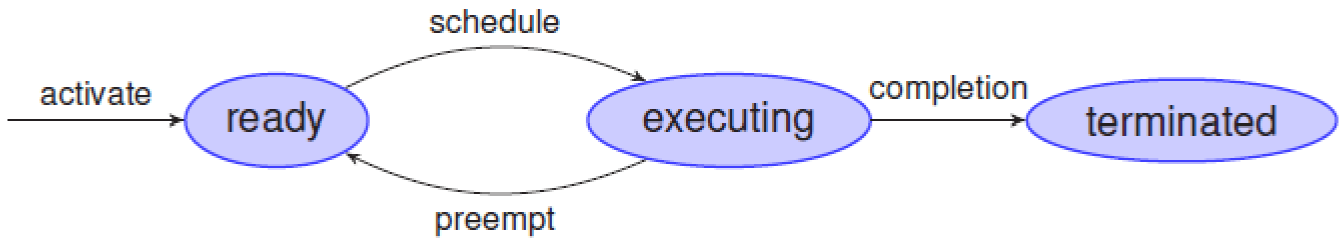
TIMING PARAMETERS OF A JOB J_J

- Arrival time (a_j) or release time (r_j) is the time at which the job becomes ready for execution
- Computation (execution) time (C_j) is the time necessary to the processor for executing the job without interruption.
- Absolute deadline (d_j) is the time at which the job should be completed.
- Relative deadline (D_j) is the time length between the arrival time and the absolute deadline.
- Start time (s_j) is the time at which the job starts its execution.
- Finishing time (f_j) is the time at which the job finishes its execution.
- Response time (R_j) is the time length at which the job finishes its execution after its arrival, which is f_j - a_j.

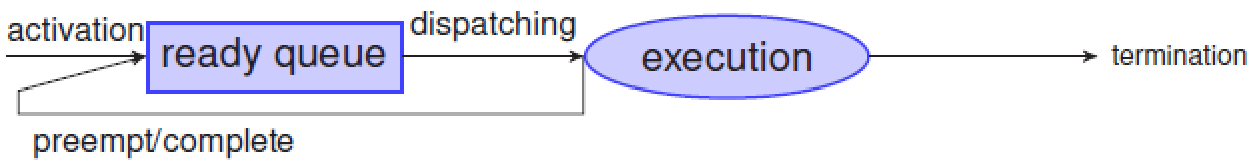


SCHEDULING CONCEPTS

- Scheduling Algorithm: determines the order that jobs execute on the processor
- Jobs may be in one of three states:



- A task that is ready or executing is active:

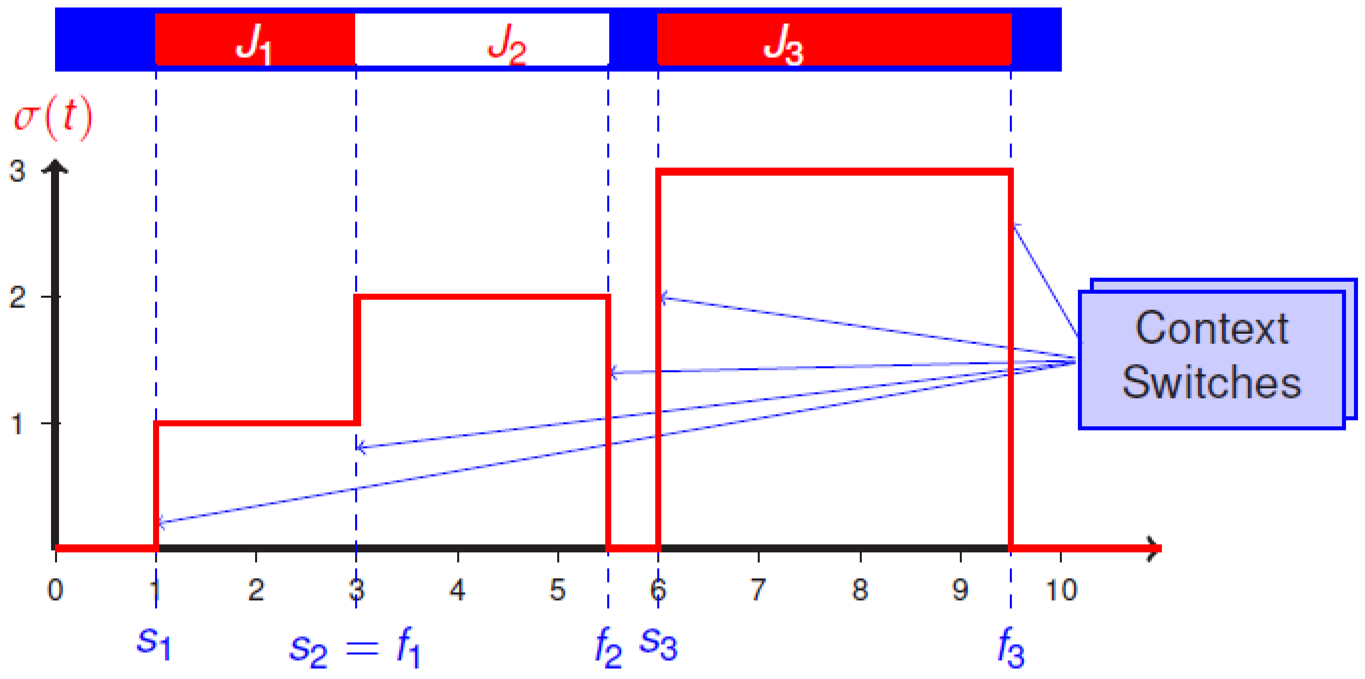


SCHEDULES FOR A SET OF JOBS $\{J_1, J_2, \ldots, J_N\}$

- A schedule is an assignment of jobs to the processor, such that each job is executed until completion.
- A schedule can be defined as an integer step function σ: ***R* -> *N***, where $\sigma(t) = j$ denotes job J_j is executed at time t, and $\sigma(t) = 0$ denotes the system is idle at time t.
- If $\sigma(t)$ changes its value at some time t, then the processor performs a context switch at time t.
- Non-preemptive scheduling: there is only one interval with $\sigma(t) = j$ for every J_j , where t is covered by the interval.
- Preemptive scheduling: there could be more than one interval with $\sigma(t) = j$.

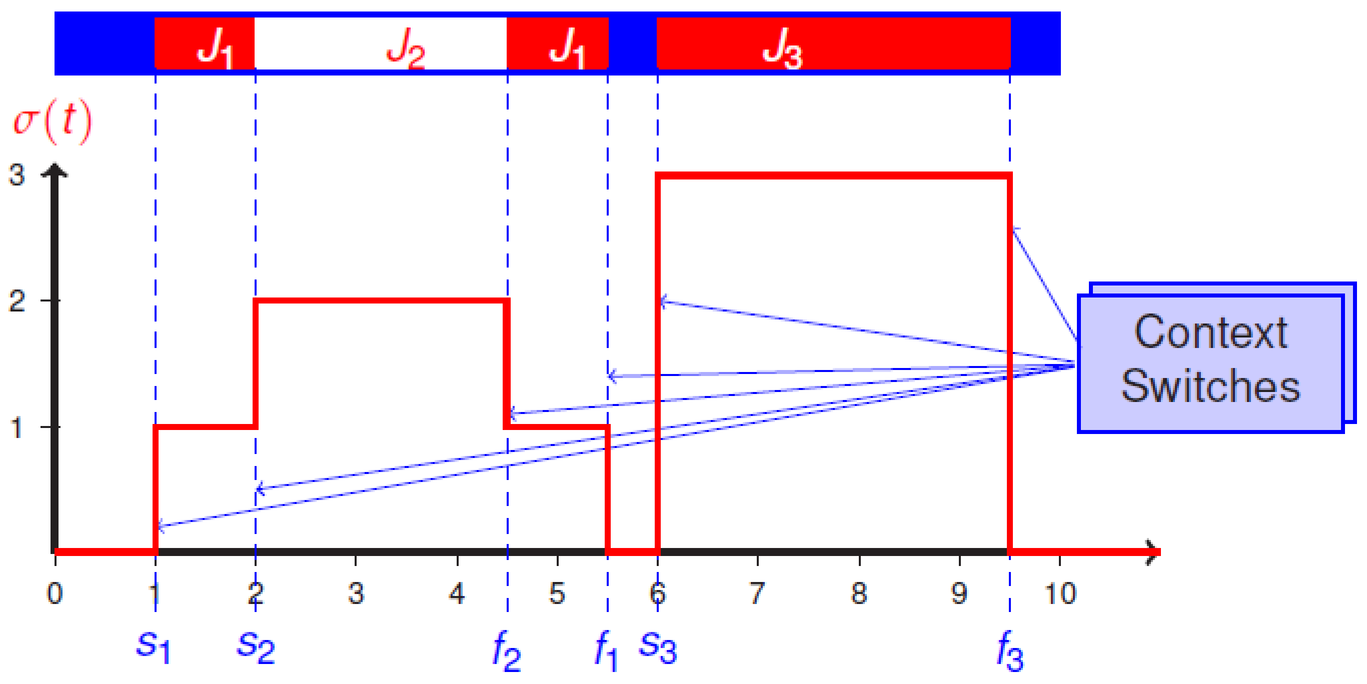
SCHEDULING CONCEPT: NON-PREEMPTIVE

- Schedule: *σ:* ***R* -> *N*** function of processor time to jobs



SCHEDULING CONCEPT: PREEMPTIVE

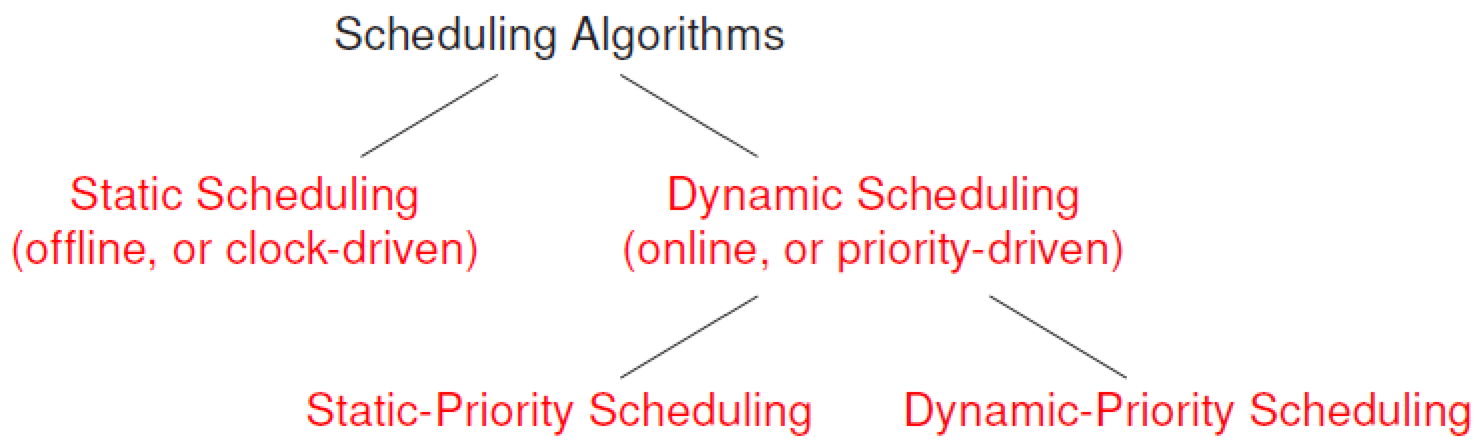
- Schedule: *σ:* ***R*** **->** ***N*** function of processor time to jobs



FEASIBILITY OF SCHEDULES AND SCHEDULABILITY

- A schedule is feasible if all jobs can be completed according to a set of specified constraints.
- A set of jobs is schedulable if there exists a feasible schedule for the set of jobs.
- A scheduling algorithm is optimal if it always produces a feasible schedule when one exists (under any scheduling algorithm).

SCHEDULING ALGORITHMS


Scheduling Algorithms
Static Scheduling
(offline, or clock-driven)
Dynamic Scheduling
(online, or priority-driven)
Static-Priority Scheduling
Dynamic-Priority Scheduling

- Preemptive vs. Non-preemptive
- Guarantee-Based vs. Best-Effort

EVALUATING A SCHEDULE

- For a job J_j:
- Lateness L_j: delay of job completion with respect to its deadline.

$$L_j = f_j - d_j$$

- Tardiness E_j: the time that a job stays active after its deadline.

$$E_j = max\{0, L_j\}$$

- Laxity (or Slack Time)(X_j): The maximum time that a job can be delayed and still meet its deadline.

$$X_j = d_j - a_j - C_j$$

METRICS OF SCHEDULING ALGORITHMS (FOR JOBS)

- Given a set ***J*** of *n* jobs, the common metrics are to minimize:

- Average response time: $\sum_{J_j \in \mathbb{J}} \frac{f_j - a_j}{|\mathbb{J}|}$

- Makespan (total completion time): $\max_{J_j \in \mathbb{J}} f_j - \min_{J_j \in \mathbb{J}} a_j$

- Total weighted response time: $\sum_{J_j \in \mathbb{J}} w_j(f_j - a_j)$

- Maximum latency: $L_{\max} = \max_{J_j \in \mathbb{J}} (f_j - d_j)$

- Number of late jobs: $N_{late} = \sum_{J_j \in \mathbb{J}} miss(J_j)$, where *miss(J$_j$) = 0* if f_j <= d_j, and *miss(J$_j$) = 1* otherwise.

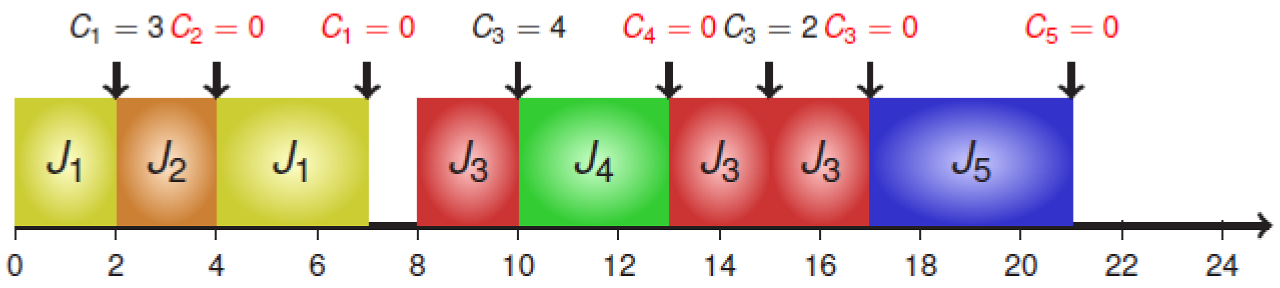
AN EXAMPLE: SHORTEST-JOB-FIRST (SJF)

- At any moment, the system executes the job with the shortest remaining time among the jobs in the ready queue.

	J_1	J_2	J_3	J_4	J_5
a_j	0	2	8	10	15
C_j	5	2	6	3	4
d_j	6	8	20	14	22

Exercise

What is the average response time of the above schedule?

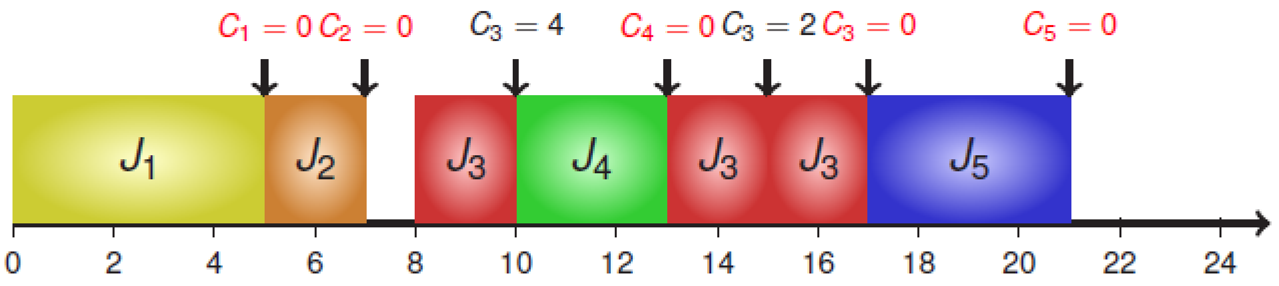


AN EXAMPLE: EARLIEST-DEADLINE-FIRST (EDF)

- At any moment, the system executes the job with the earliest absolute deadline among the jobs in the ready queue.

	J_1	J_2	J_3	J_4	J_5
a_j	0	2	8	10	15
C_j	5	2	6	3	4
d_j	6	8	20	14	22

Exercise
What is the average response time of the above schedule?

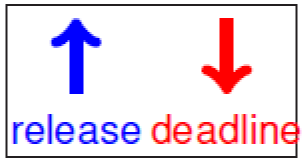


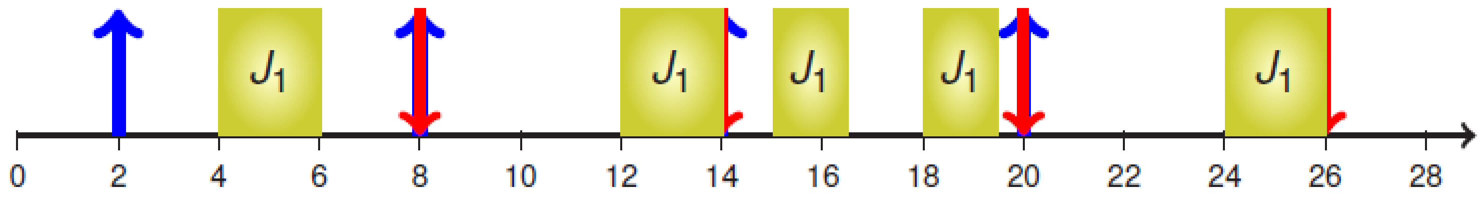
RECURRENT TASK MODELS

- When jobs (usually with the same computation requirement) are released recurrently, they can be modeled by a recurrent task
- Periodic Task t_i:
  - A job is released exactly and periodically by a period T_i
  - A phase φ_i indicates when the first job is released
  - A relative deadline D_i for each job from task t_i
  - (φ_i, C_i , T_i, D_i) is the specification of periodic task t_i, where C_i is the worst-case execution time.
- Sporadic Task t_i:
  - T_i is the minimal time between any two consecutive job releases
  - A relative deadline D_i for each job from task t_i
  - (C_i, T_i, D_i) is the specification of sporadic task t_i , where C_i is the worst-case execution time.
- Aperiodic Task: Identical jobs released arbitrarily.

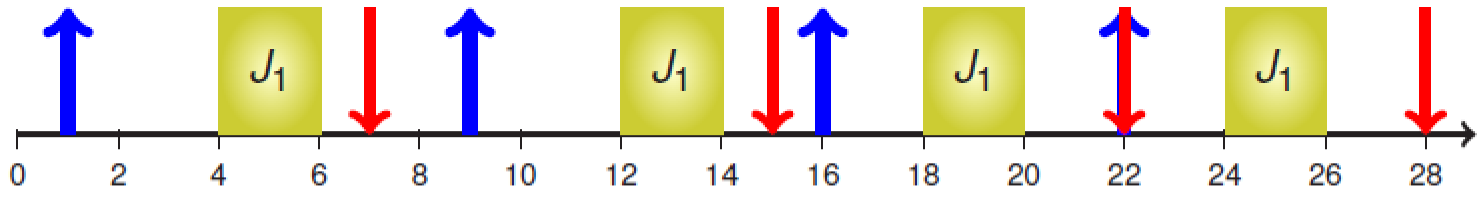
EXAMPLES OF RECURRENT TASK MODELS

Periodic task: $(\phi_i, C_i, T_i, D_i) = (2, 2, 6, 6)$





Sporadic task: $(C_i, T_i, D_i) = (2, 6, 6)$



EVALUATING A SCHEDULE FOR TASKS

- For a job J_j:
  - Lateness L_j: delay of job completion with respect to its deadline.

$$L_j = f_j - d_j$$

  - Tardiness E_j: the time that a job stays active after its deadline.

$$E_j = max\{0, L_j\}$$

  - Laxity (or Slack Time)(X_j): The maximum time that a job can be delayed and still meet its deadline.

$$X_j = d_j - a_j – C_j$$

- For a task t_j:
  - Lateness L_i: maximum lateness of jobs released by task t_i
  - Tardiness E_i: maximum tardiness of jobs released by task t_i
  - Laxity X_i: D_i - C_i;

RELATIVE DEADLINE <=> PERIOD

- For a task set, we say that the task set is with

- implicit deadline when the relative deadline D_i is equal to the period T_i , *i.e.*, $D_i = T_i$, for every task t_i,
- constrained deadline when the relative deadline D_i is no more than the period T_i, *i.e.*, $D_i <= T_i$, for every task t_i, or
- arbitrary deadline when the relative deadline D_i could be larger than the period T_i for some task t_i.

SOME DEFINITIONS FOR PERIODIC TASKS

- The jobs of task t_i are denoted $J_{i,1}$, $J_{i,2}$, ...
- Synchronous system: Each task has a phase of 0.
- Asynchronous system: Phases are arbitrary.
- Hyperperiod: Least common multiple (LCM) of T_i.
- Task utilization of task t_i : $u_i = C_i / T_i$.
- System utilization: $\sum_{\tau_i} u_i$.

FEASIBILITY AND SCHEDULABILITY FOR RECURRENT TASKS

- A schedule is feasible if all the jobs of all tasks can be completed according to a set of specified constraints.
- A set of tasks is schedulable if there exists a feasible schedule for the set of tasks.
- A scheduling algorithm is optimal if it always produces a feasible schedule when one exists (under any scheduling algorithm).

MONOTONICITY OF SCHEDULING ALGORITHMS

- A good scheduling algorithm should be monotonic

- If a scheduling algorithm derives a feasible solution, it should also guarantee the feasibility with
  - less execution time of a task/job,
  - less number of tasks/jobs, or
  - more number of processors/machines.

SCHEDULABILITY ANALYSIS

- Schedulability for Static-Priority Scheduling
  - Utilization-Based Analysis (Relative Deadline = Period)
  - Demand-Based Analysis

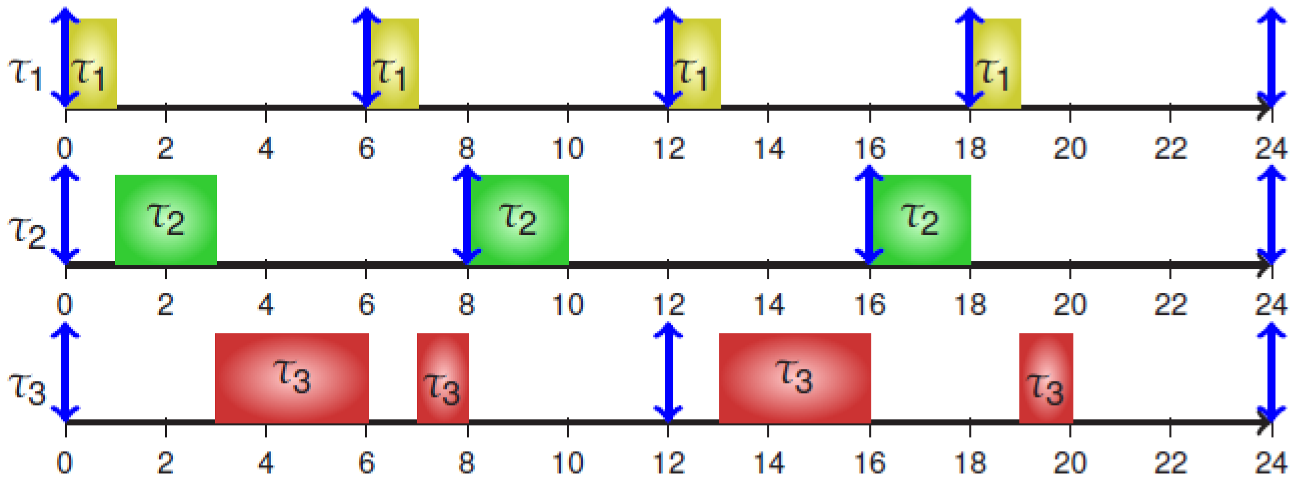
- Schedulability for Dynamic-Priority Scheduling

STATIC-PRIORITY SCHEDULING

- Different jobs of a task are assigned the same priority.
  - π_i is the priority of task t_i.
  - HP_i is the subset of tasks with higher priority than t_i.
  - Note: we will assume that no two tasks have the same priority.
- We will implicitly index tasks in decreasing priority order, *i.e.,* t_i has higher priority than t_k if $i < k$.
- Which strategy is better or the best?
  - largest execution time first?
  - shortest job first?
  - least-utilization first?
  - most importance first?
  - least period first?

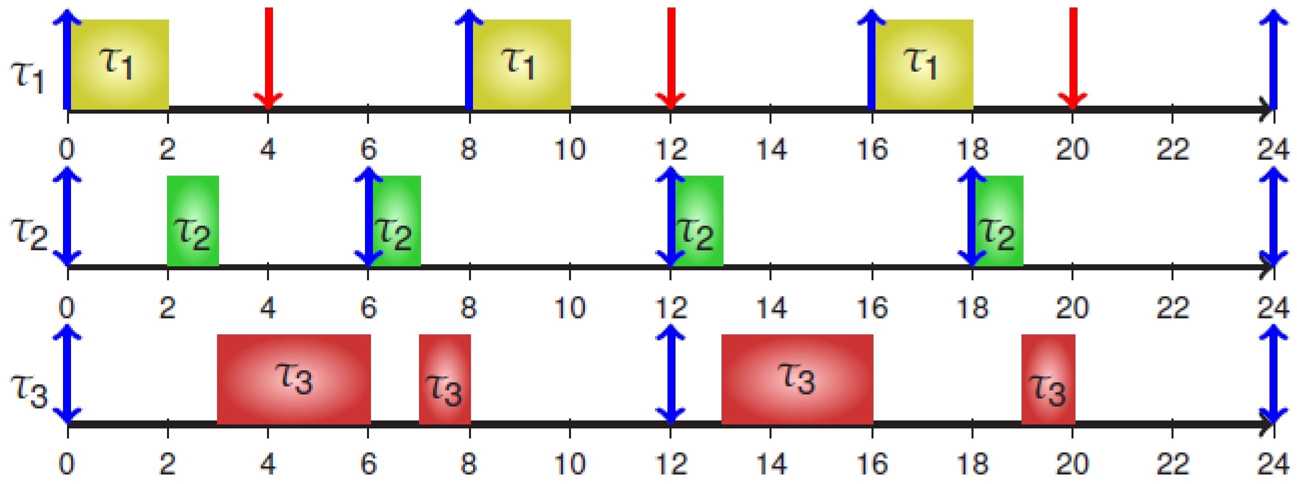
RATE-MONOTONIC (RM) SCHEDULING

- Priority Definition: A task with a smaller period has higher priority, in which ties are broken arbitrarily.
- Example Schedule: t_1 = (1, 6, 6), t_2 = (2, 8, 8), t_3 = (4, 12, 12). [(C_i, T_i, D_i)]



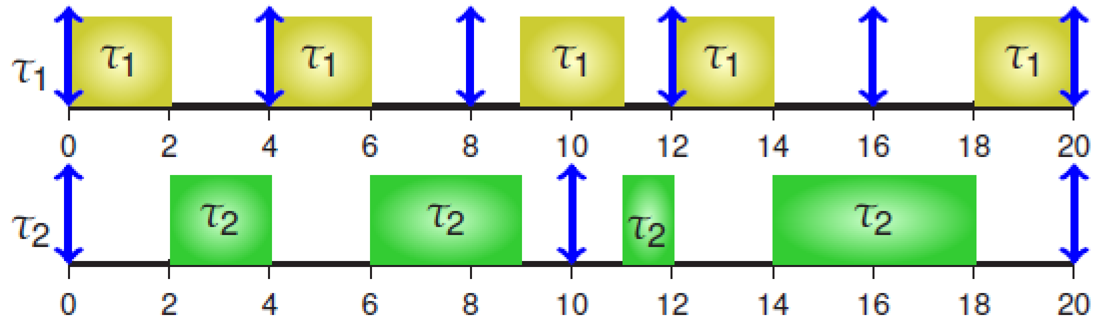
DEADLINE-MONOTONIC (DM) SCHEDULING

- Priority Definition: A task with a smaller relative deadline has higher priority, in which ties are broken arbitrarily.
- Example Schedule: t_1 = (2, 8, 4), t_2 = (1, 6, 6), t_3 = (4, 12, 12). [(C_i, T_i, D_i)]



OPTIMALITY (OR NOT) OF RM AND DM

- Example Schedule: t_1 = (2, 4, 4), t_2 = (5, 10, 10)



- The above system is schedulable.
- No static-priority scheme is optimal for scheduling periodic tasks: However, a deadline will be missed, regardless of how we choose to (statically) prioritize t_1 and t_2.

- Corollary: Neither RM nor DM is optimal

OPTIMALITY AMONG STATIC-PRIORITY ALGORITHMS

- Theorem: A system of T independent, preemptable, synchronous periodic tasks that have relative deadlines equal to their respective periods can be feasibly scheduled on one processor according to the RM algorithm whenever it can be feasibly scheduled according to any static priority algorithm.

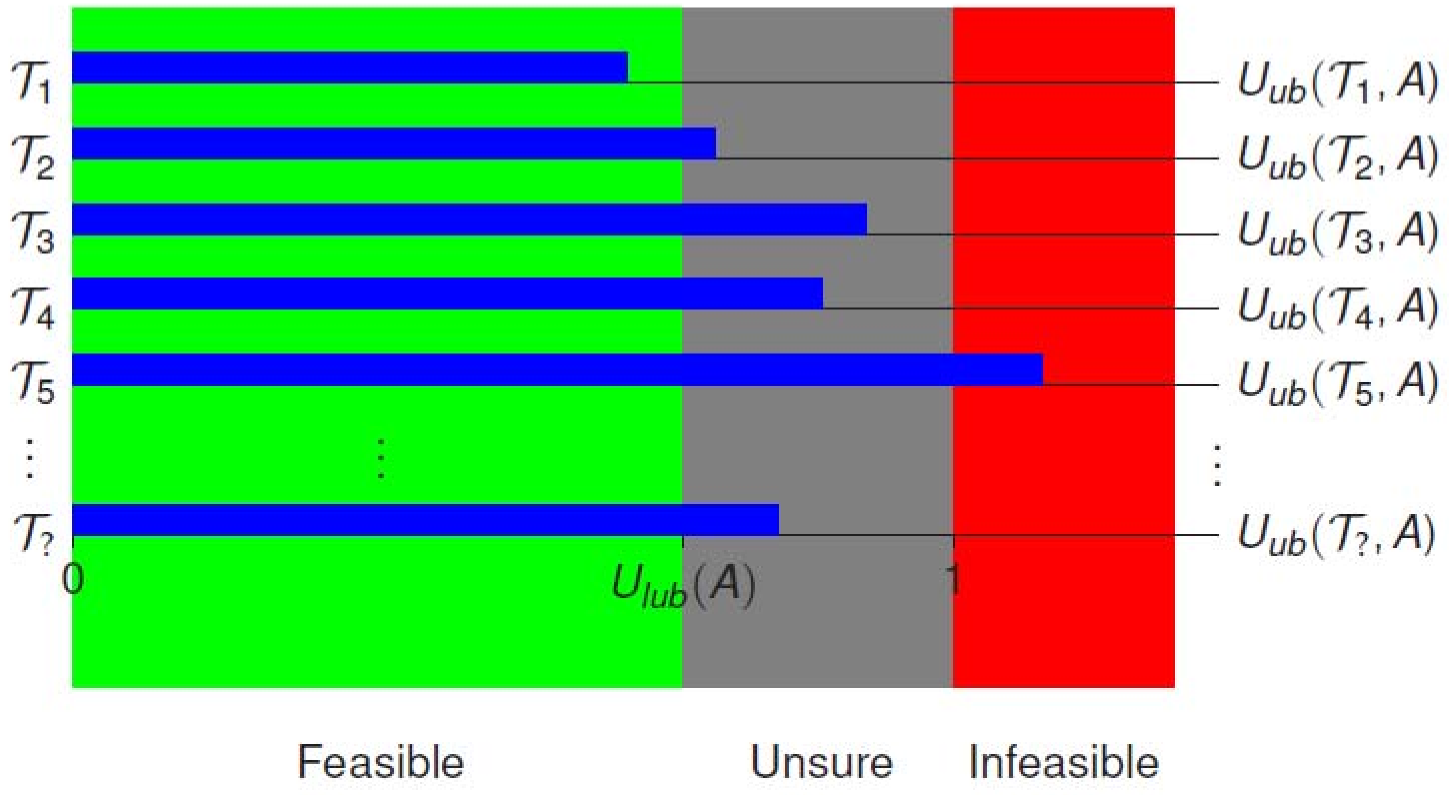
- Note: When $D_i \le T_i$ for all tasks, DM can be shown to be an optimal static-priority algorithm using similar argument. Proof left as an exercise.

UTILIZATION-BASED SCHEDULABILITY TEST

- Task utilization: $u_i = \frac{C_i}{T_i}.$
- System (total) utilization: $U(\mathcal{T}) = \sum_{\tau_i \in \mathcal{T}} \frac{C_i}{T_i}.$
- A task system *T* fully utilizes the processor under scheduling algorithm *A* if any increase in execution time (of any task) causes *A* to miss a deadline. In this case, *U(T)* is a upper bound on utilization for *A*, denoted $U_{ub}(T, A)$.
- $U_{lub}(A)$ is the least upper bound for algorithm *A*:

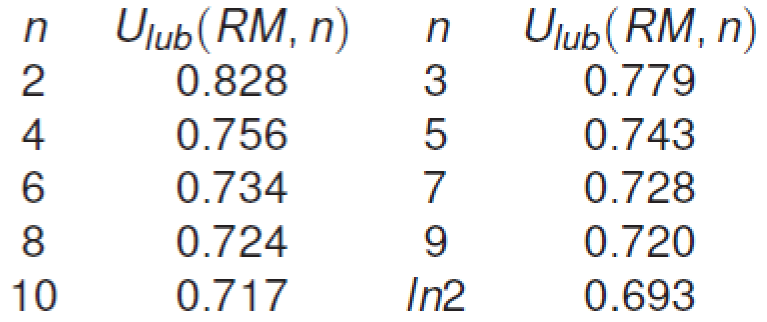
$$U_{lub}(A) = \min_{\mathcal{T}} U_{ub}(\mathcal{T}, A)$$

WHAT IS $U_{LUB}(A)$ FOR?


$\mathcal{T}_1$
$\mathcal{T}_2$
$\mathcal{T}_3$
$\mathcal{T}_4$
$\mathcal{T}_5$
$\mathcal{T}_?$
$U_{ub}(\mathcal{T}_1, A)$
$U_{ub}(\mathcal{T}_2, A)$
$U_{ub}(\mathcal{T}_3, A)$
$U_{ub}(\mathcal{T}_4, A)$
$U_{ub}(\mathcal{T}_5, A)$
$U_{ub}(\mathcal{T}_?, A)$
0
$U_{lub}(A)$
1
Feasible
Unsure
Infeasible

LIU AND LAYLAND BOUND

- Theorem: [Liu and Layland] A set of n independent, preemptable periodic tasks with relative deadlines equal to their respective periods can be scheduled on a processor according to the RM algorithm if its total utilization U is at most $n(2^{1/n} - 1)$. In other words, $U_{lub}(RM, n) = n(2^{1/n} - 1) >= 0.693$.



n	$U_{lub}(RM, n)$	n	$U_{lub}(RM, n)$
2	0.828	3	0.779
4	0.756	5	0.743
6	0.734	7	0.728
8	0.724	9	0.720
10	0.717	$ln2$	0.693

SCHEDULABILITY ANALYSIS

- Schedulability for Static-Priority Scheduling
  - Utilization-Based Analysis (Relative Deadline = Period)
  - Demand-Based Analysis
- Schedulability for Dynamic-Priority Scheduling

UTILIZATION-BASED TEST FOR EDF SCHEDULING

- Theorem: A task set T of independent, preemptable, periodic tasks with relative deadlines equal to their periods can be feasibly scheduled (under EDF) on one processor if and only if its total utilization U is at most one.

RELATIVE DEADLINES LESS THAN PERIODS

- Theorem: A task set T of independent, preemptable, periodic tasks with relative deadlines equal to or less than their periods can be feasibly scheduled (under EDF) on one processor if:

$$\sum_{k=1}^{n} \frac{C_k}{\min\{D_k, T_k\}} \leq 1.$$

- Note: This theorem only gives sufficient condition.

COMPARISON BETWEEN RM AND EDF (IMPLICIT DEADLINES)

RM

- Low run-time overhead: *O(1)* with priority sorting in advance
- Optimal for static-priority
- Schedulability test is NP-hard (even if the relative deadline = period)
- Least upper bound: 0.693
- In general, more preemption

EDF

- High run-time overhead: *O(log n)* with balanced binary tree
- Optimal for dynamic-priority
- Schedulability test is easy (when the relative deadline = period)
- Least upper bound: 1
- In general, less preemption