Real-Time Scheduling (Part I)

(Working Draft)

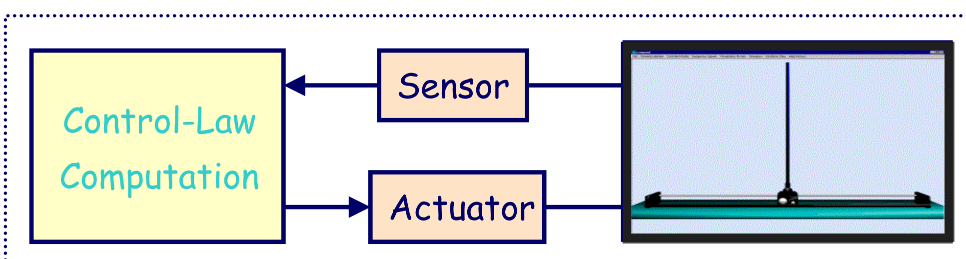
Insup Lee
Department of Computer and Information Science
School of Engineering and Applied Science
University of Pennsylvania
www.cis.upenn.edu/~lee/

CIS 541, Spring 2010

Real-Time System Example

- Digital control systems
  - periodically performs the following job:

    senses the system status and
    actuates the system according to its current status

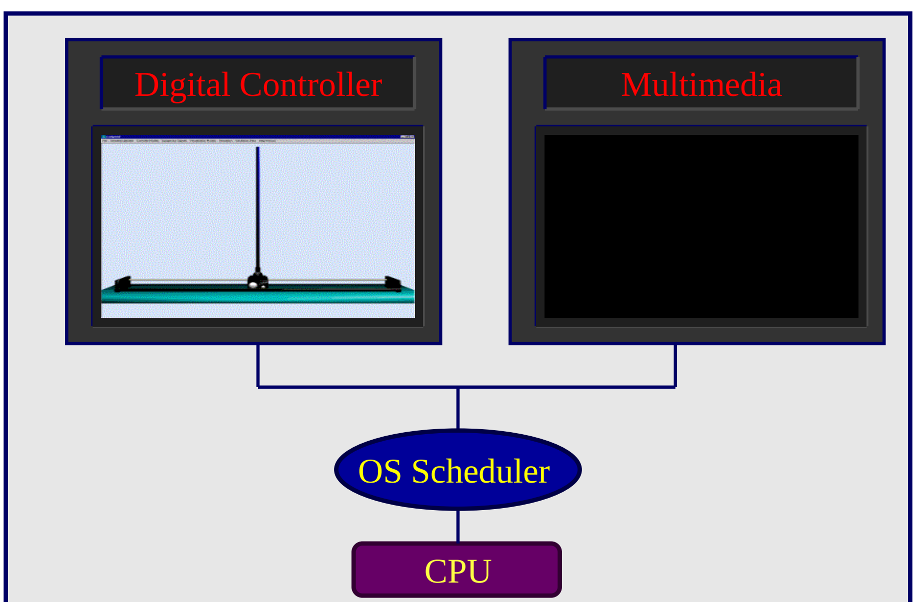


Real-Time System Example

- Multimedia applications
  - periodically performs the following job:

    reads, decompresses, and displays video and audio streams

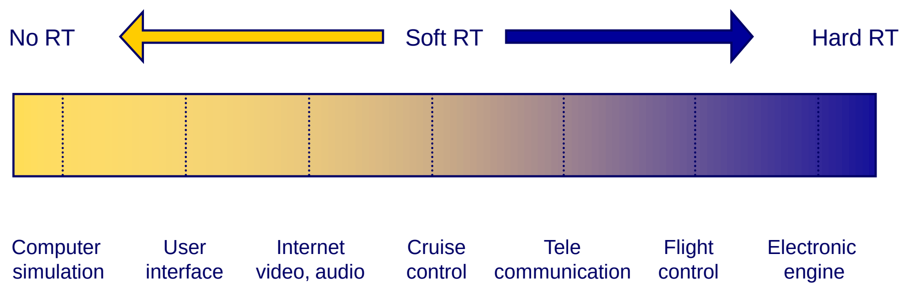
Scheduling Framework Example



Fundamental Real-Time Issue

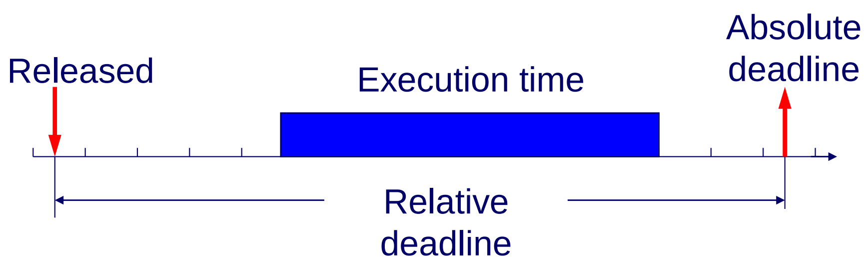
- To specify the timing constraints of real-time systems
- To achieve predictability on satisfying their timing constraints, possibly, with the existence of other real-time systems

Real-Time Spectrum



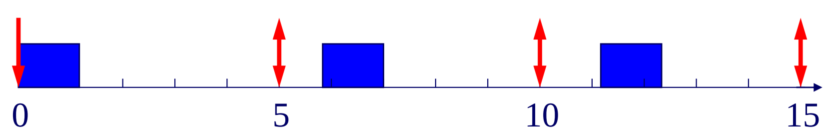
Real-Time Workload

- Job (unit of work)
  - a computation, a file read, a message transmission, etc
- Attributes
  - Resources required to make progress
  - Timing parameters



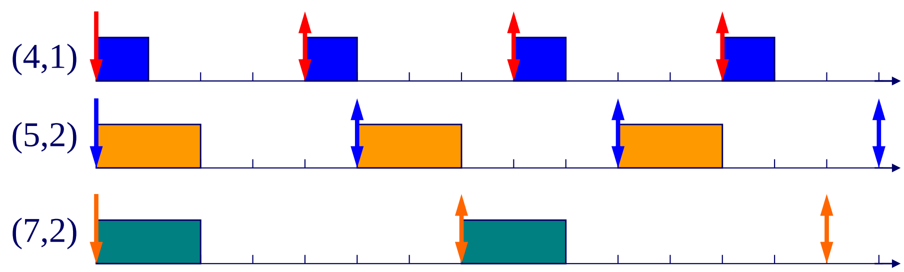
Real-Time Task

- Task : a sequence of similar jobs
  - Periodic task (*p*,*e*)
    - Its jobs repeat regularly
    - Period *p* = inter-release time (0 < *p*)
    - Execution time e = maximum execution time (0 < e < *p*)
    - Utilization U = e/p



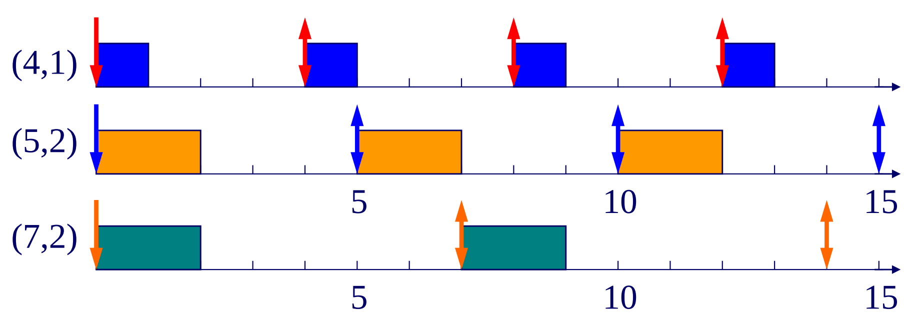
Schedulability

- Property indicating whether a real-time system (a set of real-time tasks) can meet their deadlines



Real-Time Scheduling

- Determines the order of real-time task executions



Optimality

- Scheduling algorithm A is better than B if for every task set T, B can schedule T implies A can schedule T.
- Given a class C of scheduling algorithms, scheduling algorithm A is optimal in C, if A is better than every scheduling algorithm B in C.

Real-Time Scheduling

- Static scheduling
  - A fixed schedule is determined statically
  - E.g., Cyclic Executive
- Static-priority scheduling
  - Assign fixed priorities to processes
  - A scheduler only needs to know about priorities
  - E.g., Rate Monotonic (RM)
- Dynamic-priority scheduling
  - Assign priorities based on current state of the system
  - E.g., Least Completion Time (LCT), Earliest Deadline First (EDF), Least Slack Time (LST)

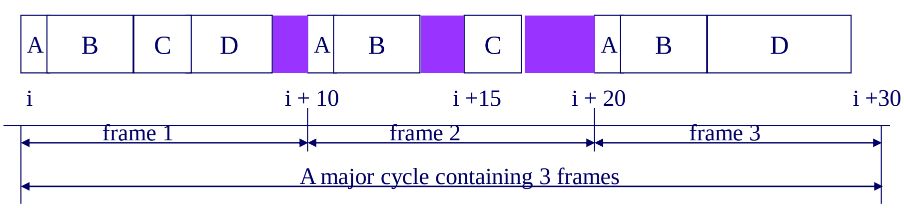
CYCLIC EXECUTIVE

Cyclic Executives

A cyclic executive is a program that deterministically interleaves the execution of periodic tasks on a single processor. The order in which the tasks execute is defined by a cyclic schedule.

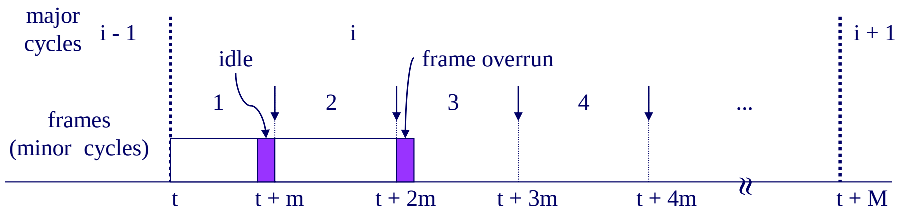
Example:

A = (10, 1), B = (10, 3), C = (15, 2), D = (30,8) ⇒ (30, 3) (30, 5)



Reference: "The Cyclic executive model and ADA" Proc. of RTSS, 1988, by Baker and Shaw

General structure of a major schedule



correct timing enforced at the end of each frame

Rules governing the choice of m for a given $\{(p_i, e_i, d_i)\}$ of n tasks

- $m \le d_i$, i = 1, 2, … , n
- $m \ge e_i$, i = 1, 2, … , n
- M/n = integer (*m* divides p_i for at least one i)
- there must be at least one frame in the interval between the release time and deadline of every request.
  $2m - gcd(m, p_i) \le d_i$, for i = 1, 2, … , n

Advantages of cyclic executive

- Simplicity and predictability:
  - timing constraints can be easily checked
  - the cyclic schedule can be represented by a table that is interpreted by the executive
  - context switching overhead is small
  - it is easy to construct schedules that satisfy precedence constraints & resource constraints without deadlock and unpredictable delay

Disadvantages

- Given major and frame times, structuring the tasks with parameters *pi*, *ei*, and *di* to meet all deadlines is NP-hard for one processor
- Splitting tasks into subtasks and determining the scheduling blocks of each task is time consuming
- Error in timing estimates may cause frame overrun:
  How to handle frame overrun? It is application dependent:
  - suspense or terminate the overrun task, and execute the schedule of the next frame
  - complete the suspended task as background later
  - complete the frame, defer the start of the next frame
  - log overruns. If too many overruns, do fault recovery

Mode changes

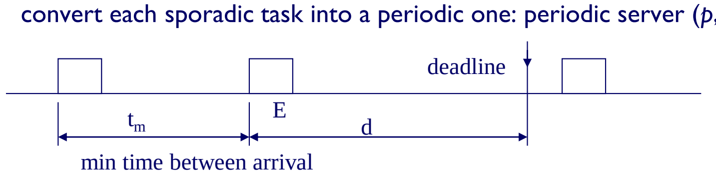
Handling mode change is difficult.

Mode change: deletion and addition of tasks or change the parameters of existing tasks

When to do mode change? Pros and cons of doing it at the end of current frame, current major cycle, execution of the current task, upon interrupt immediately

Handling sporadic tasks

- convert each sporadic task into a periodic one: periodic server (*p*, e, *d*)



$p = \min(t_m, d - e + 1)$ - too pessimistic - guarantees worst case performance by giving max time to the task

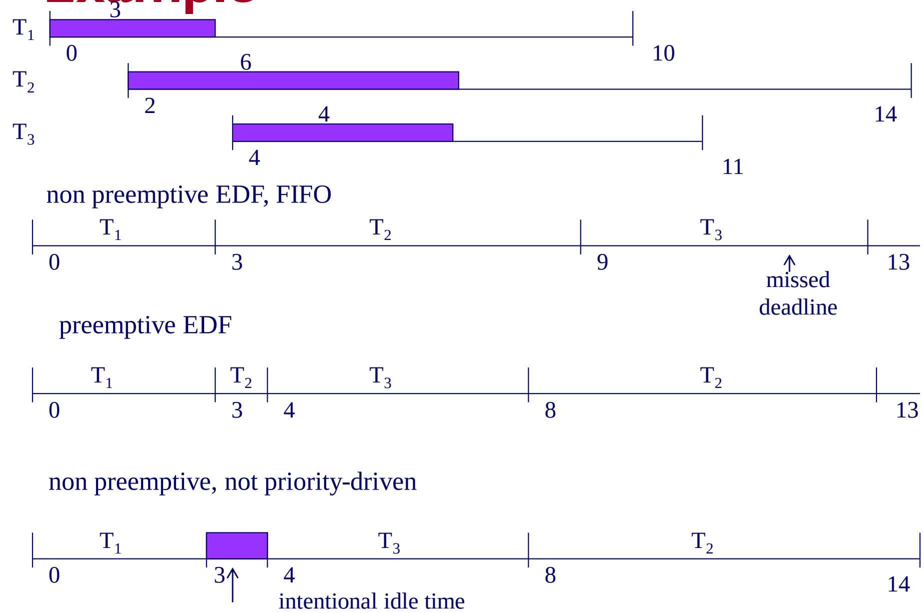
- set aside time in minor cycles for execution of sporadic tasks
  - does not guarantee worst case

Priority-driven algorithms

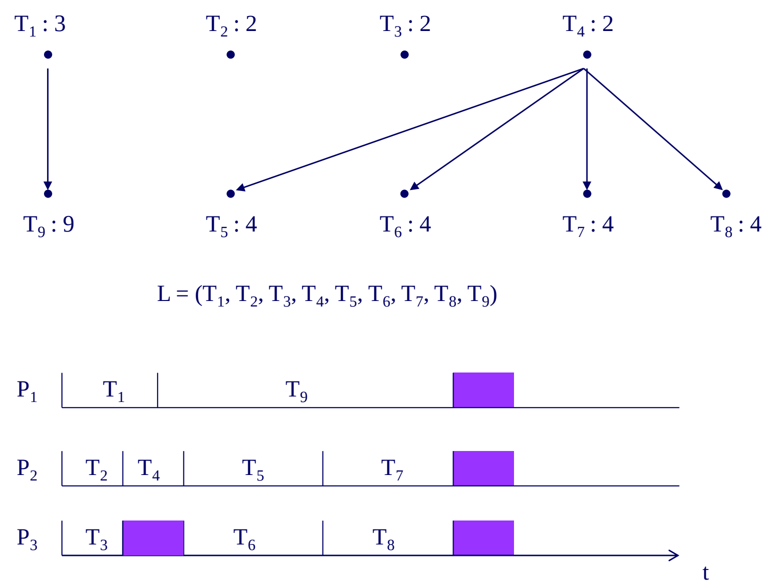
- A class of algorithms that never leave the processor(s) idle intentionally
- Also known as greedy algorithms and list algorithms
- Can be implemented as follows: (preemptive)
  - Assign priorities to tasks
  - Scheduling decisions are made
    - when any task becomes ready,
    - when a processor becomes idle,
    - when the priorities of tasks change
  - At each scheduling decision time, the ready task with the highest priority is executed
- If non-preemptive, scheduling decisions are made only when a processor becomes idle.
- The algorithm is static if priorities are assigned to tasks once for all time, and is dynamic if they change. Static if fixed.

SCHEDULING ANOMALY

Example

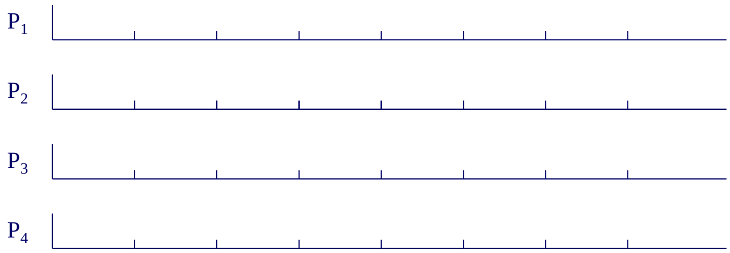


Unexpected behavior of priority-driven scheduling algorithm



Unexpected behavior of priority-driven scheduling algorithm (cont.)

- Suppose that we have 4 processors:



- Suppose that execution times are

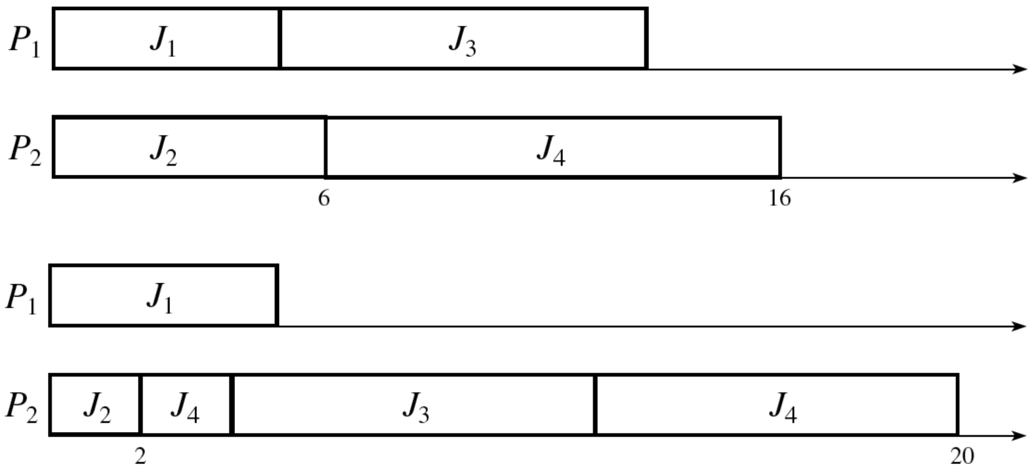
  2, 1, 1, 1, 3, 3, 3, 3, 8
- Suppose that T_4 *before* T_5 and T_4 *before* T_6 are removed

Anomalies of Priority-Driven Systems

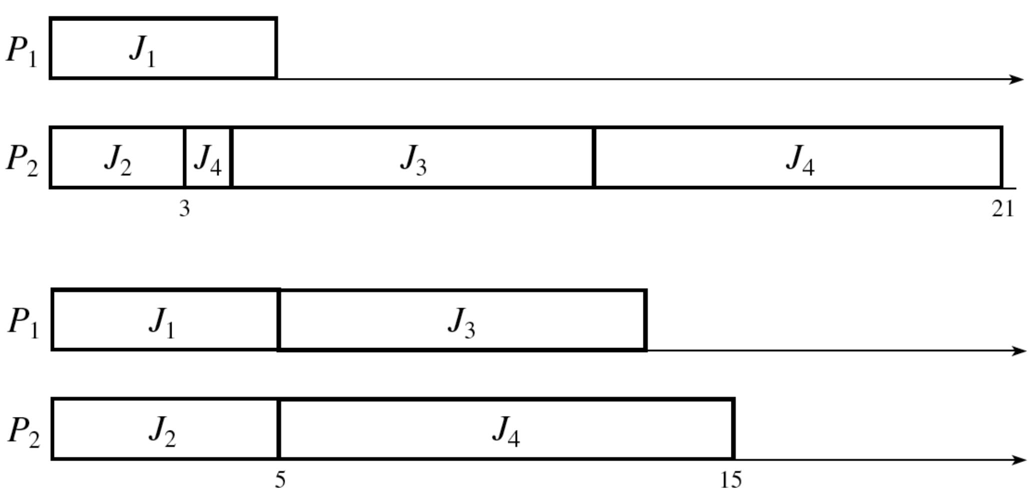
- Given J1, J2, J3, J4
- Priority:
  - J1 > J2 > J3 > J4
- Preemption but no job migration
- Two processors P1 and P2

	r_i	d_i	$[e_i^-, e_i^+]$
J_1	0	10	5
J_2	0	10	[2, 6]
J_3	4	15	8
J_4	0	20	10

Cases: e2 = 6 and e2 = 2



Cases: e2 = 3 and e2 = 5



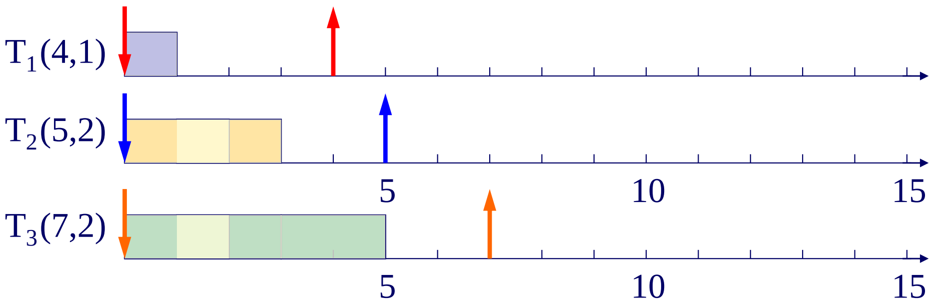
Validation Problem

- Scheduling anomalies make validation hard
- Can anomalies exist even on single processor systems?

EARLIEST DEADLINE FIRST (EDF)

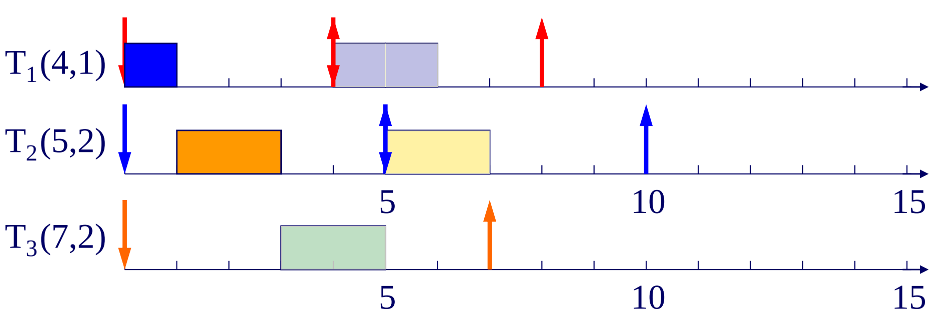
EDF (Earliest Deadline First)

- A task with a shorter deadline has a higher priority
- Executes a job with the earliest deadline
- Optimal dynamic priority scheduling for single processor



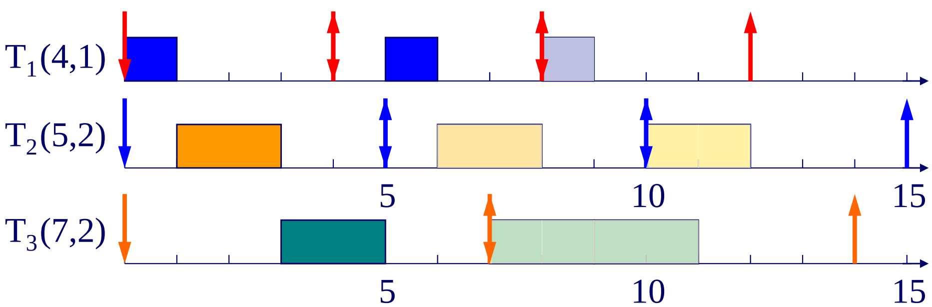
EDF (Earliest Deadline First)

- Executes a job with the earliest deadline



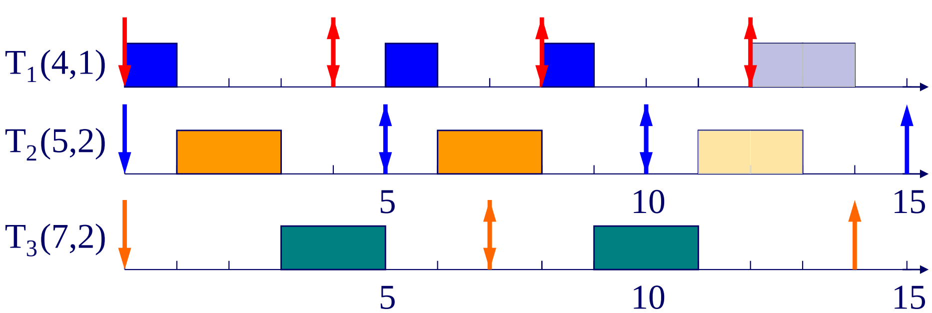
EDF (Earliest Deadline First)

- Executes a job with the earliest deadline



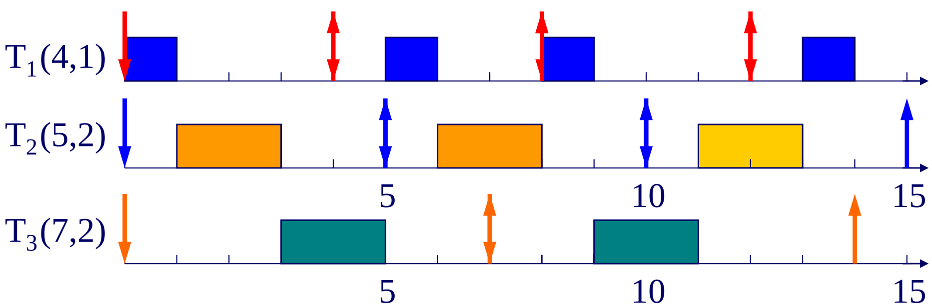
EDF (Earliest Deadline First)

- Executes a job with the earliest deadline



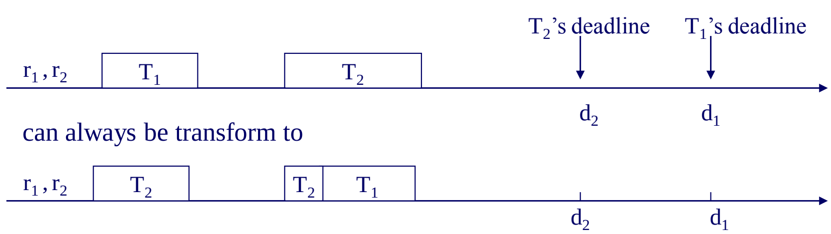
EDF (Earliest Deadline First)

- Optimal scheduling algorithm
  - if there is a schedule for a set of real-time tasks, EDF can schedule it.



Optimality of EDF

- Optimality of the earliest deadline first algorithm for preemptive scheduling on one processor
- Given a task system **T**, if the EDF algorithm fails to find a feasible schedule, then **T** has no feasible schedule, where
  - feasible schedule = one in which all release time and deadline constraints are met



Utilization Bounds

- Intuitively:
  - The lower the processor utilization, U, the easier it is to meet deadlines.
  - The higher the processor utilization, U, the more difficult it is to meet deadlines.
- Question: is there a threshold U_{bound} such that
  - When $U < U_{bound}$ deadlines are met
  - When $U > U_{bound}$ deadlines are missed

EDF – Utilization Bound

- Real-time system is schedulable under EDF if and only if

$$\sum U_i \leq 1$$

Liu & Layland,

"Scheduling algorithms for multi-programming in a hard-real-time environment", Journal of ACM, 1973.

Schedulable Utilization

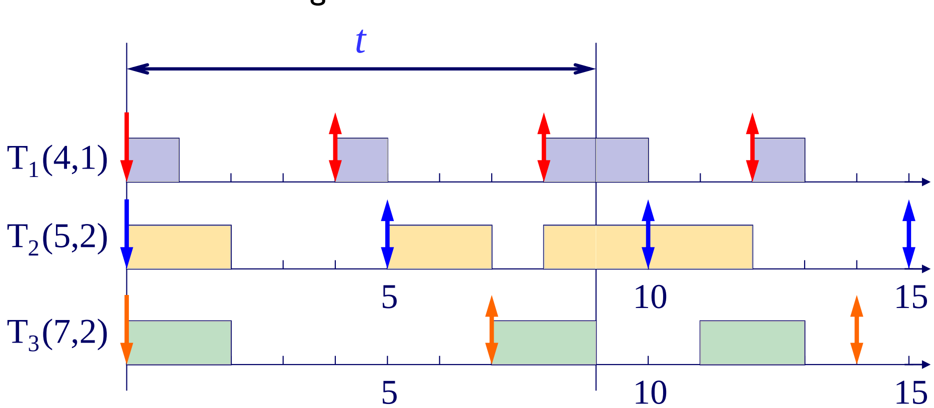
- Utilization of a periodic task (p, τ, d)
  $u = \tau/p \equiv$ the fraction of time the task keeps the processor busy
  U , total utilization of task system
  $\mathbf{T} = \{(p_i, \tau_i, d_i)\}$ contains n tasks

$$U = \sum_{i=1}^{n} \tau_i / p_i \equiv \text{demand for processor time}$$

- A system of n tasks with $d_i = p_i$ can be feasibly scheduled if and only if $U \leq 1$
  - If $U > 1$, the total demand of processor in the time interval $[0, p_1 p_2 \dots p_n]$ is $p_2 p_3 \dots p_n \tau_1 + p_1 p_3 \dots p_n \tau_2 + \dots p_1 p_2 \dots p_{n-1} \tau_n > p_1 p_2 \dots p_n$ clearly, no feasible schedule exists.
  - If $U \leq 1$, the EDF algorithm can always find a feasible schedule. To show this statement is true, we suppose that the EDF algorithm fails to find a feasible schedule. And, then show $U > 1$, which is a contradiction to $U \leq 1$

Processor Demand Bound

- Demand Bound Function : *dbf(t)*
  - the maximum processor demand by workload over any interval of length t



EDF - Schedulability Analysis

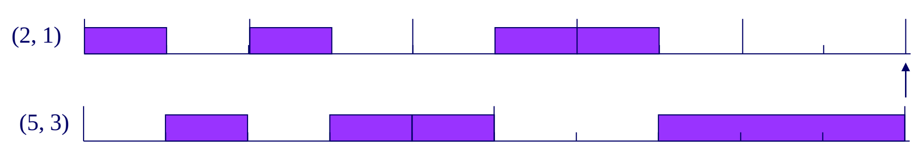
- Real-time system is schedulable under EDF
  if and only if **dbf(t) ≤ t** for all interval t

Baruah et al.
"Algorithms and complexity concerning the preemptive
scheduling of periodic, real-time tasks on one
processor", Journal of Real-Time Systems, 1990.

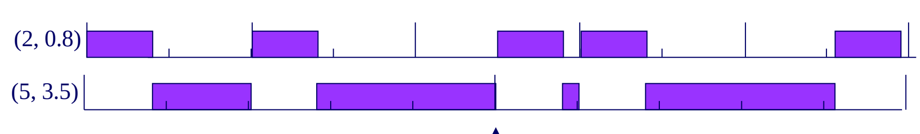
- Demand Bound Function : dbf(t)
  - the maximum processor demand by workload over any interval of length t

Behavior of earliest deadline algorithm

- Schedule (2, 1) (5, 3) with U = 1.1



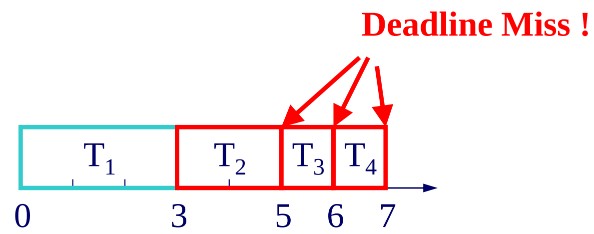
- Schedule (2, 0.8) (5, 3.5) with U = 1.1



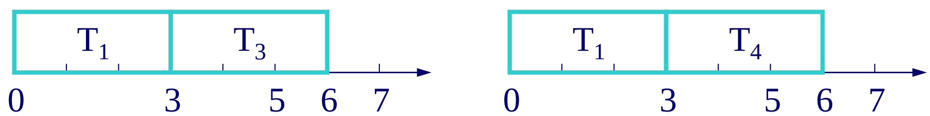
Which deadline will be missed
as U increases cannot be predicted

EDF – Overload Conditions

- Domino effect during overload conditions
  - Example: $T_1(4,3)$, $T_2(5,3)$, $T_3(6,3)$, $T_4(7,3)$



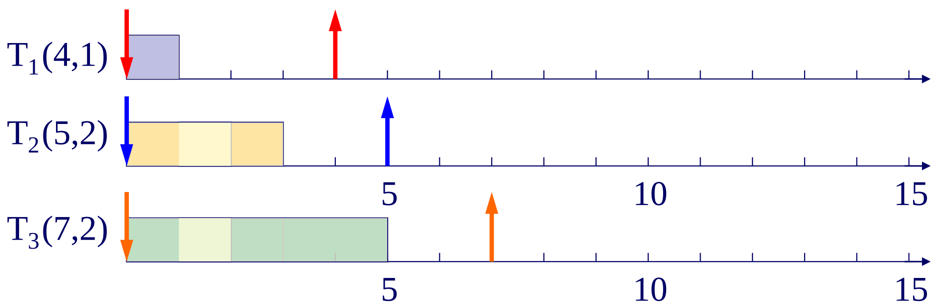
Better schedules :



RATE MONOTONIC (RM)

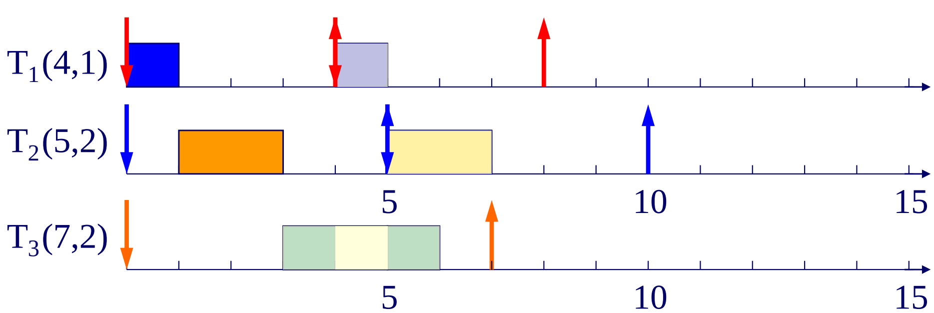
RM (Rate Monotonic)

- It assigns priority according to period
- A task with a shorter period has a higher priority
- Executes a job with the shortest period
- Optimal static-priority scheduling for single processor



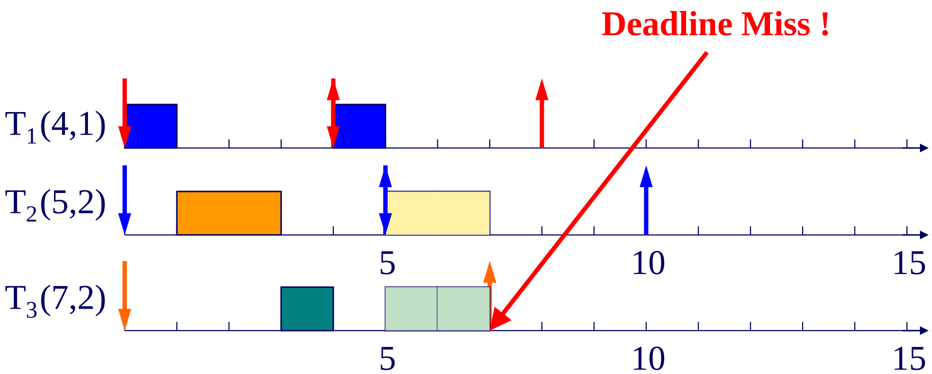
RM (Rate Monotonic)

- Executes a job with the shortest period



RM (Rate Monotonic)

- Executes a job with the shortest period



RM – Utilization Bound

- Real-time system is schedulable under RM if

$$\sum U_i \leq n\,(2^{1/n}-1)$$

Liu & Layland,
"Scheduling algorithms for multi-programming in a hard-real-time environment", Journal of ACM, 1973.

RM – Utilization Bound

- Real-time system is schedulable under RM if
  $$\sum U_i \le n\,(2^{1/n}-1)$$

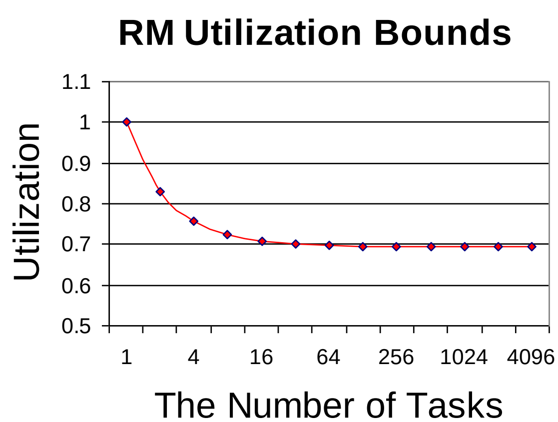
- Example: $T_1(4,1)$, $T_2(5,1)$, $T_3(10,1)$,

$$\sum U_i = 1/4 + 1/5 + 1/10$$
$$= 0.55$$
$$3\,(2^{1/3}-1) \approx 0.78$$

Thus, $\{T_1, T_2, T_3\}$ is schedulable under RM.

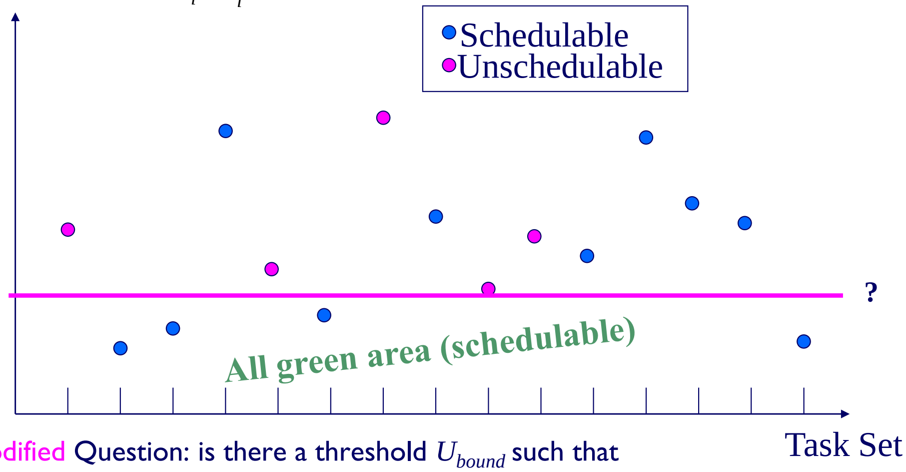
RM – Utilization Bound

- Real-time system is schedulable under RM if
  $$\sum U_i \le n\,(2^{1/n}-1)$$



A Conceptual View of Schedulability

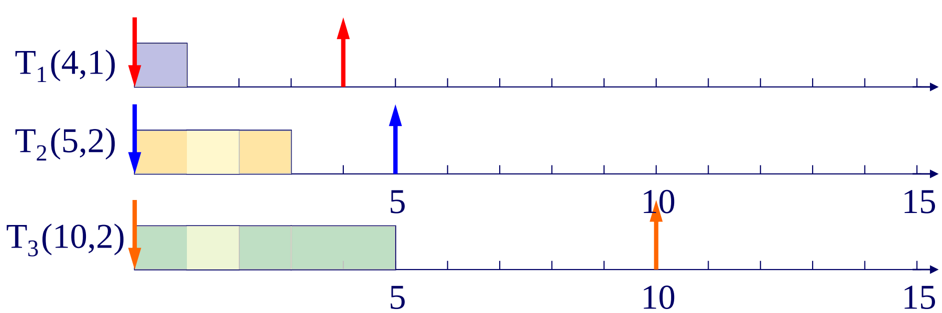
Utilization $= \sum_i \frac{C_i}{P_i}$



- Modified Question: is there a threshold U_{bound} such that
  - When $U < U_{bound}$ deadlines are met
  - When $U > U_{bound}$ deadlines **may or may not be** missed

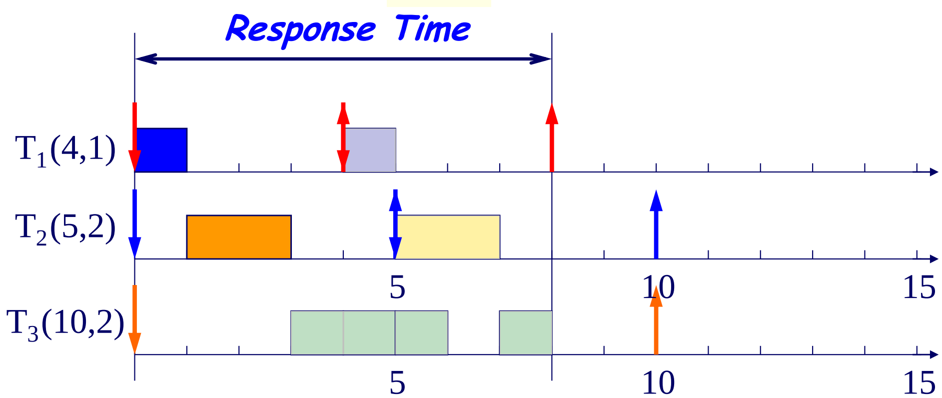
Response Time

- Response time
  - Duration from released time to finish time



Response Time

- Response time
  - Duration from released time to finish time

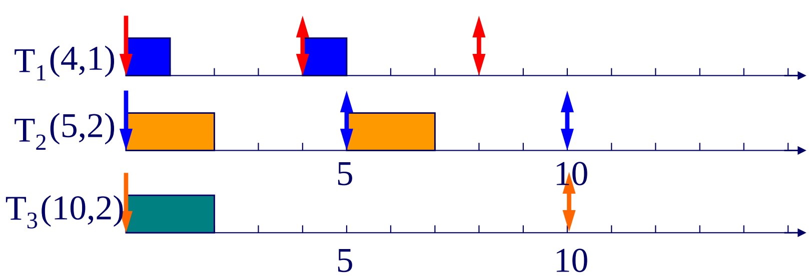


Response Time

- Response Time (r_i) [Audsley et al., 1993]

$$r_i = e_i + \sum_{T_k \in HP(T_i)} \left\lceil \frac{r_i}{p_k} \right\rceil \cdot e_k$$

- HP(T_i) : a set of higher-priority tasks than T_i



RM - Schedulability Analysis

- Real-time system is schedulable under RM
  if and only if $r_i \le p_i$ for all task $T_i(p_i,e_i)$

Joseph & Pandya,
"Finding response times in a real-time system",
The Computer Journal, 1986.

RM vs. EDF

- Rate Monotonic
  - Simpler implementation, even in systems without explicit support for timing constraints (periods, deadlines)
  - Predictability for the highest priority tasks
- EDF
  - Full processor utilization
  - Misbehavior during overload conditions
- For more details: Buttazzo, "Rate monotonic vs. EDF: Judgement Day", EMSOFT 2003.