https://www.nsf.gov/news/mmg/mmg_disp.jsp?med_id=78274&from=

Real-Time System Analysis

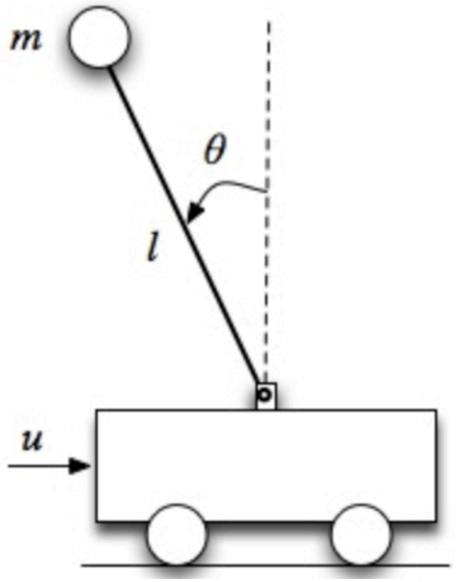
Jin Hyun Kim

References

- CIS 441/541: Embedded Software for Life- Critical CPS/IoT Applications
  - Insup Lee
  - Department of Computer and Information Science School of Engineering and Applied Science University of Pennsylvania

Contents

Real-Time System Example

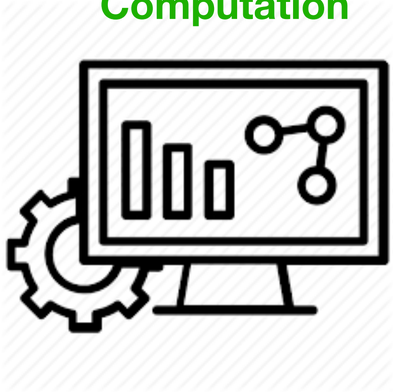


https://www.youtube.com/watch?v=855O9x0Pgf0

- https://sites.google.com/a/cyphylab.ee.ucla.edu/pessoa/documentation/examples-1/inverted-pendulum

Real-Time System Example

Control Law Computation



Car Antilock Brake System

Digital control systems periodically performs the following job:

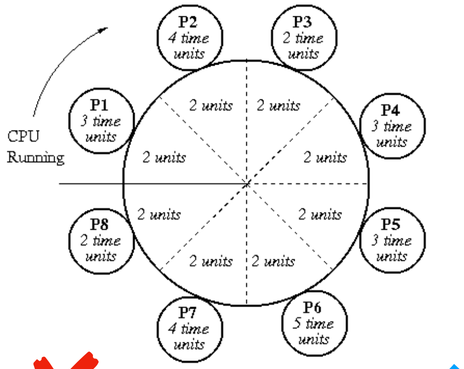
- senses the system status and
- actuates the system according to its current status

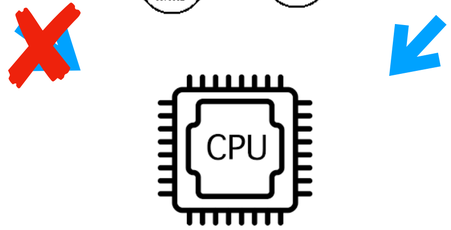
Real-Time System Example

- Navigation applications periodically performs the following job:
  - Reads GPS, compute position, and displays map

Scheduling Framework

Navigation





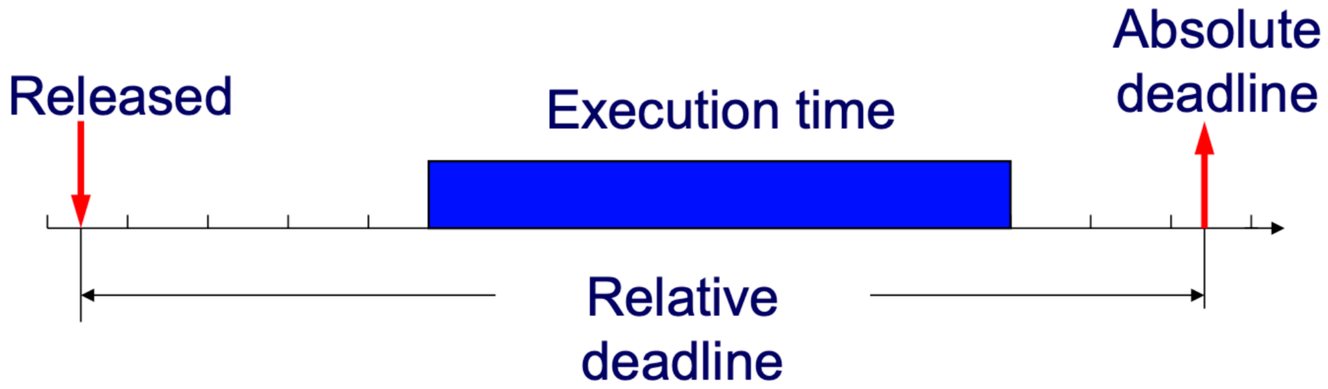
ABS

Fundamental Issue

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

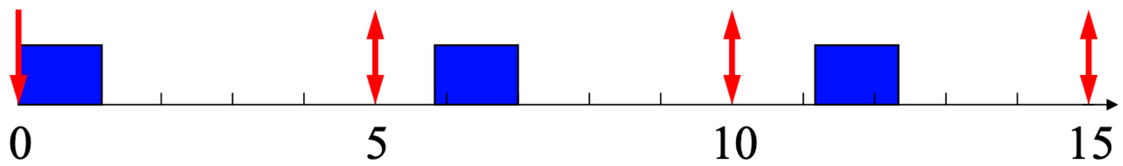
Real-Time Workload

- Job (A unit of work)
  - A computation, a file read, a message transmission etc.
  - Resource required to make it progress
  - Timing parameters



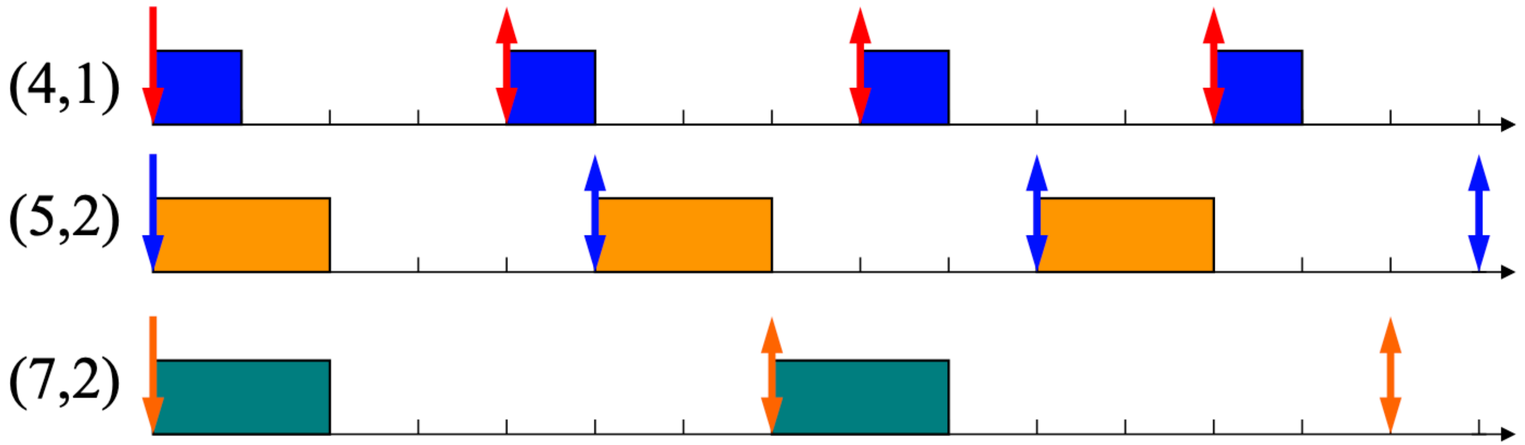
Real-Time Task

- Task: A sequence of similar jobs
  - A period task $T = (p, e)$
    - p: Period, inter-release time ($0 < p$)
    - e: Worst-case (Maximum) execution time $(0 < e < p)$
    - Utilization $U = \frac{e}{p}$



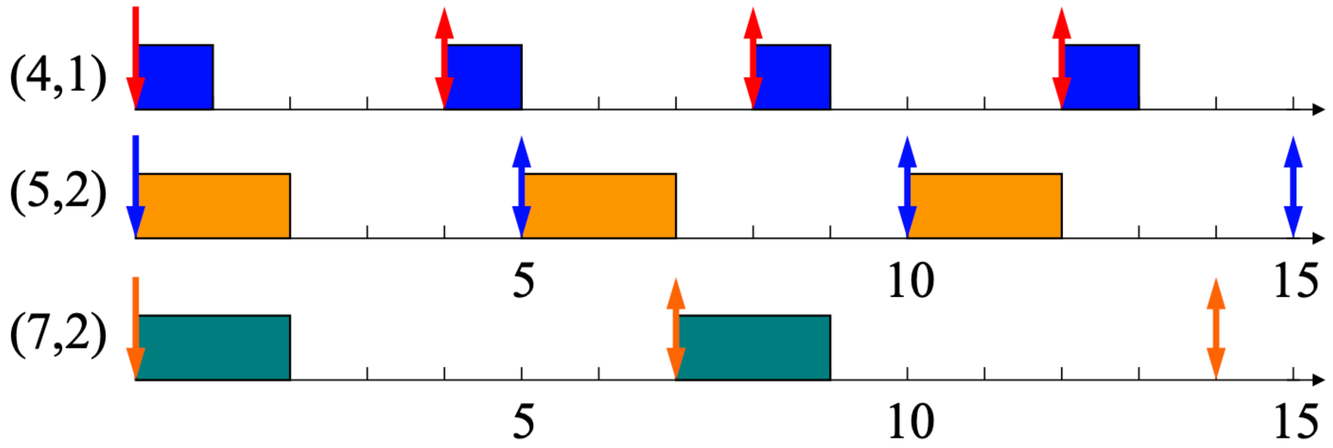
Schedulability

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



Real-Time Scheduling

- Determine the order of real-time task executions



Optimality

- Scheduling algorithm A is as good as 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 as good as every other scheduling algorithm B in C.

Real-Time Scheduling

- Static scheduling
  - A fixed scheduling is determined statically
  - E.g., Cyclic Executive (i.e., timeline scheduling)

Real-Time Scheduling

- Static-Priority Scheduling
  - Assigned fixed priority to tasks
  - A scheduling only needs to know about priorities of tasks
  - E.g., Rate Monotonic (RM
- Dynamic Scheduling
  - Assign priorities based on current state of the system
  - E.g., Least Completion Time (LCT), Earliest Deadline First (EDF), Least Slack Time (LST)

https://www.nsf.gov/news/mmg/mmg_disp.jsp?med_id=78274&from=

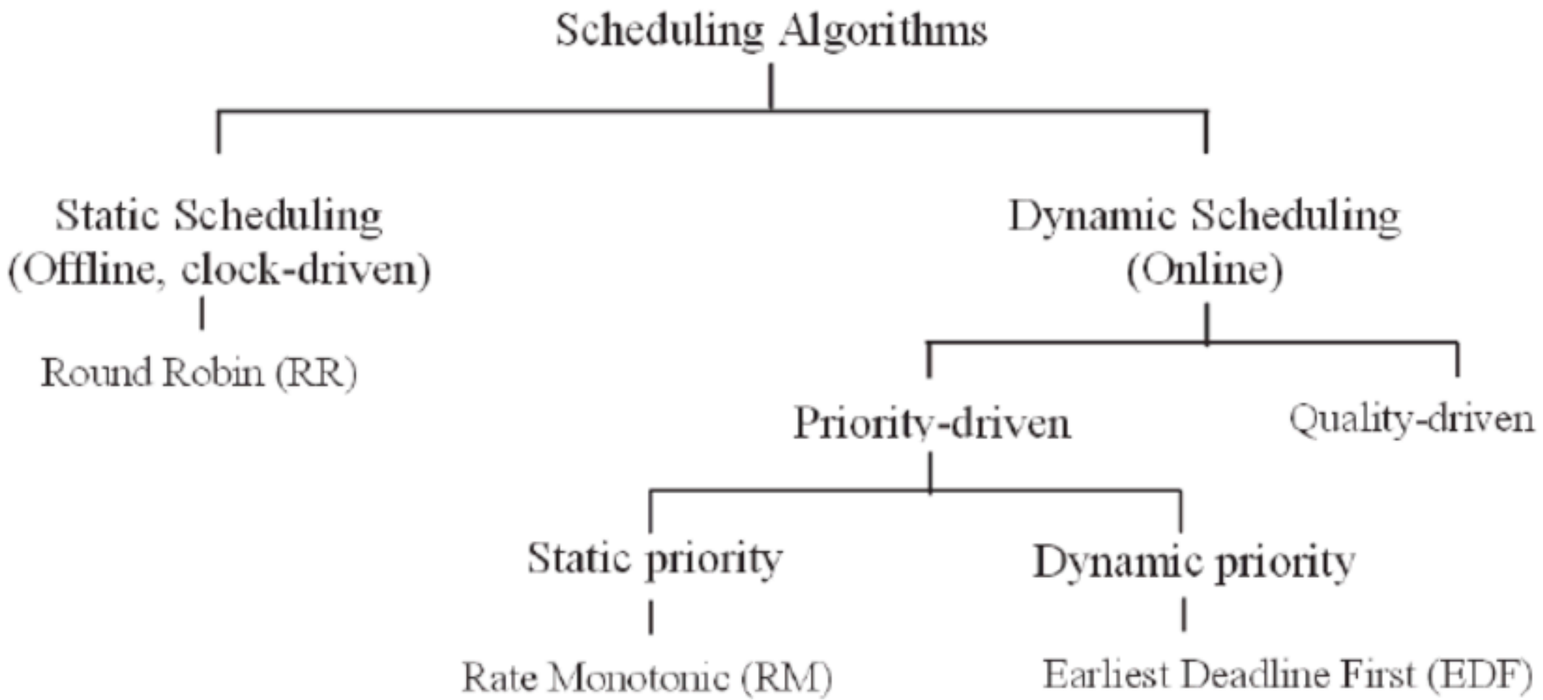
Real-Time System Analysis
Part 2

Jin Hyun Kim

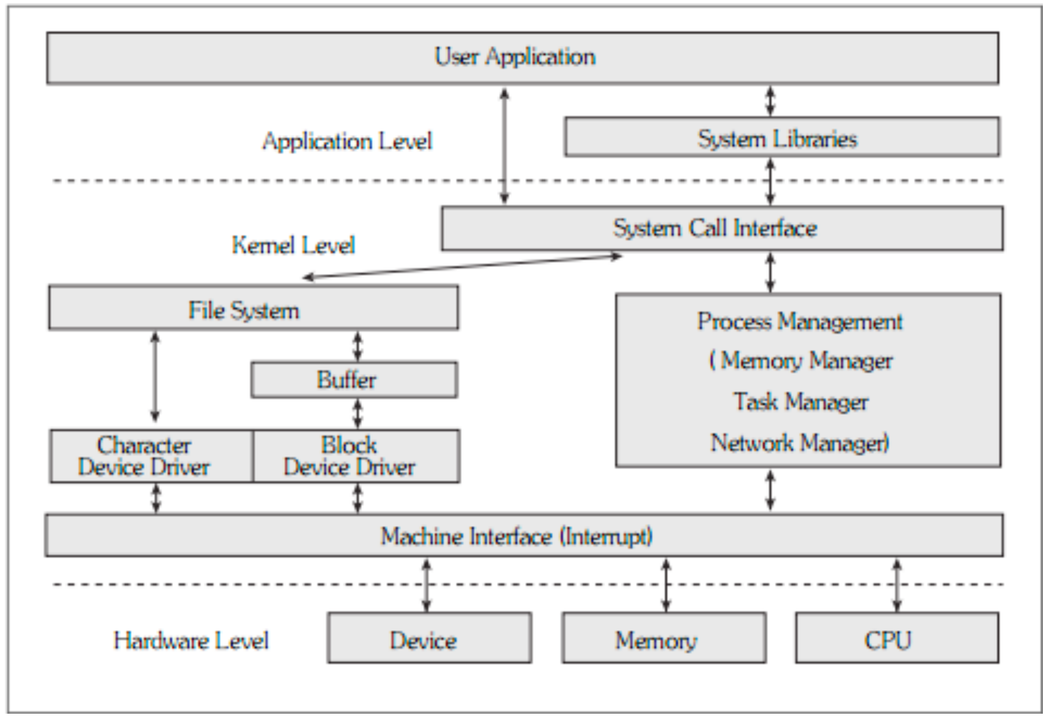
Contents

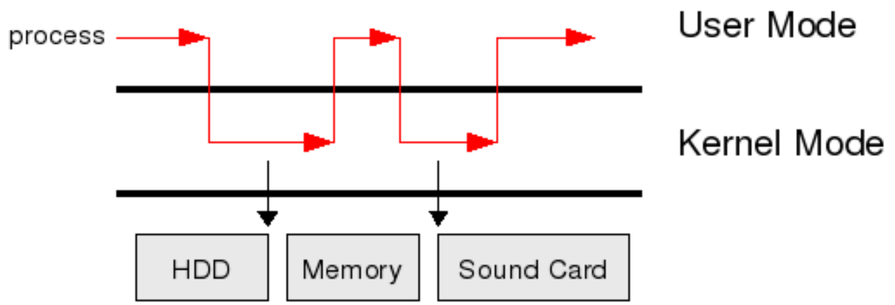
Contents

Scheduling Algorithms

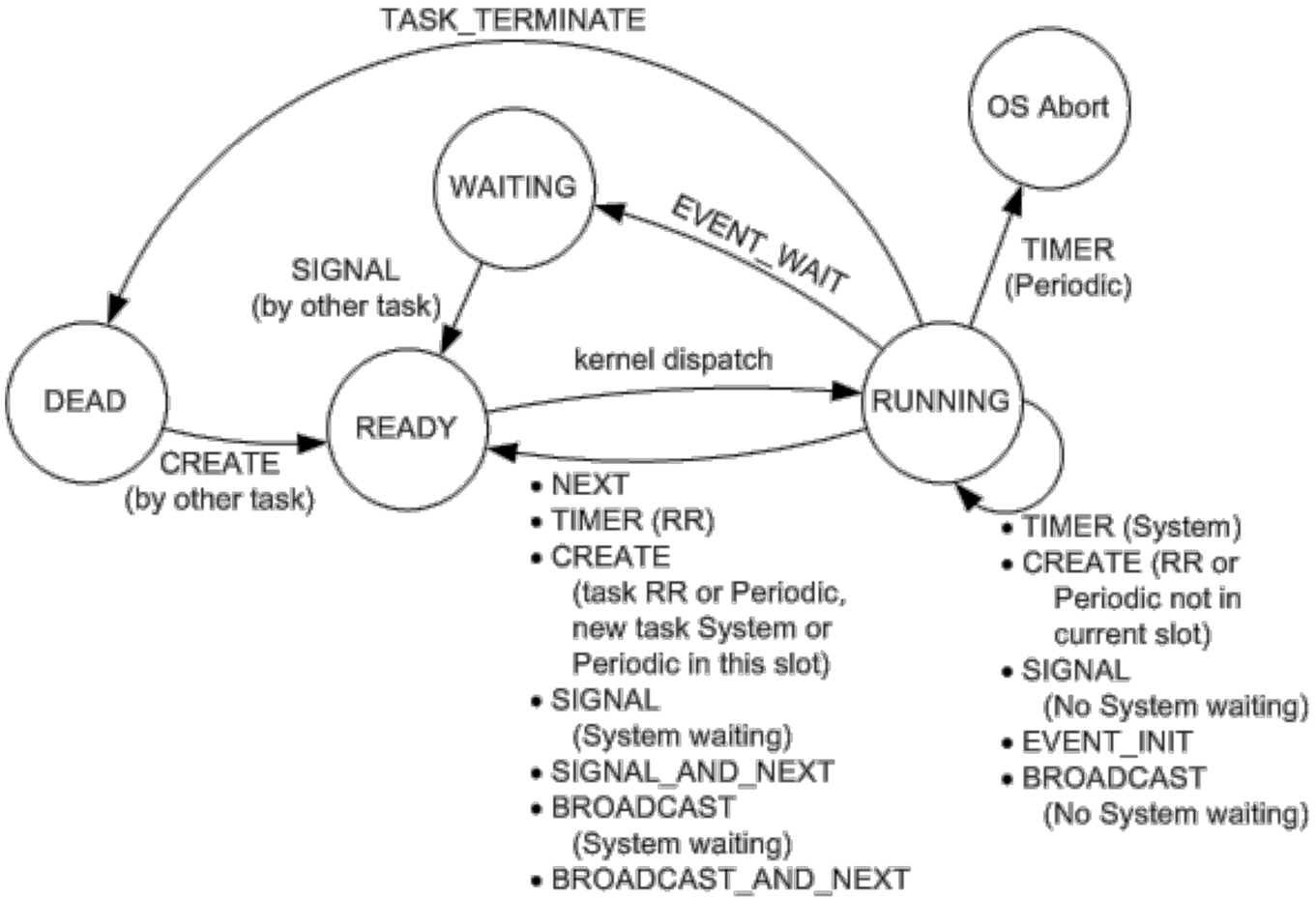


Operating System Structure

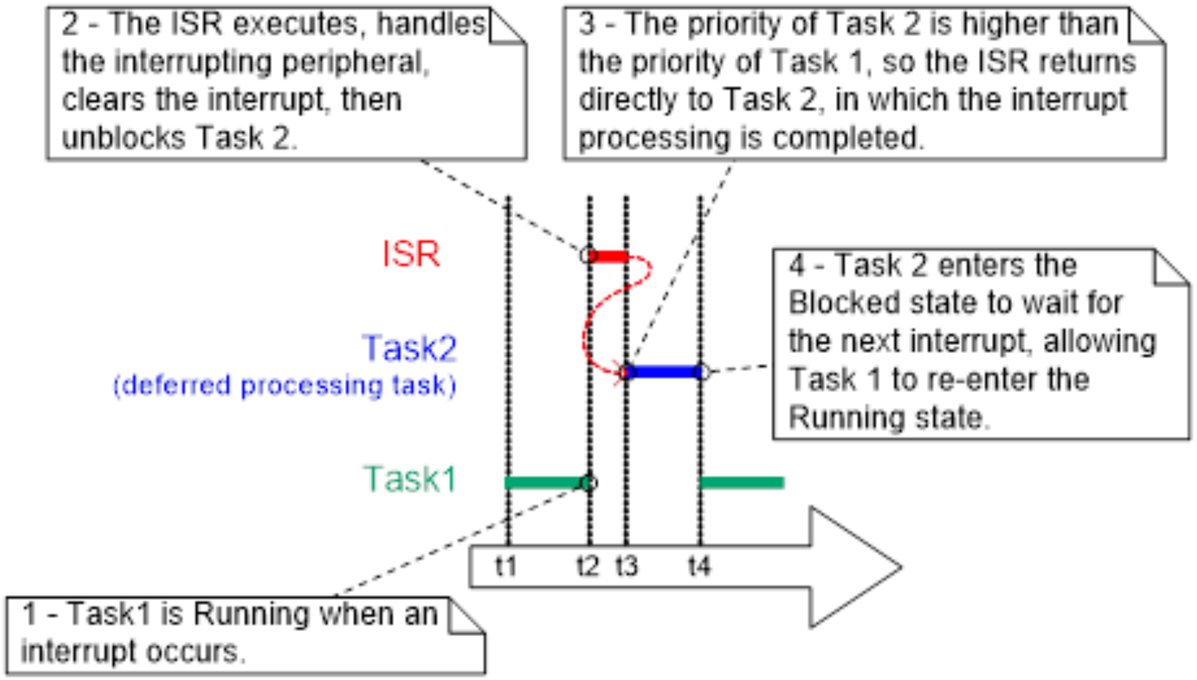




Task Behavior



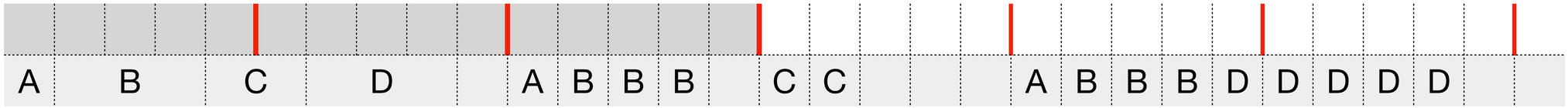
Interrupt Service Routine (ISR)



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)



- A cyclic scheduling pattern is repeated

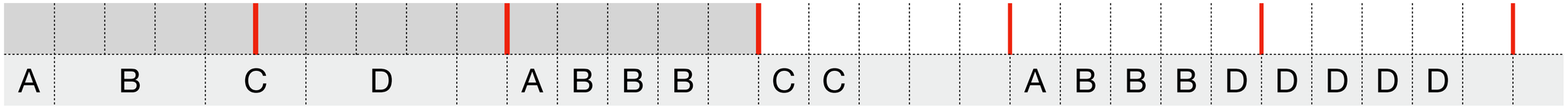
- “The Cyclic executive model and ADA” Proc. of RTSS, 1988, by Baker and Shaw

Pros

- 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

Cons

- Given major and frame times, structuring the tasks with parameters p_i, e_i, and d_i 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 back ground later
  - complete the frame, defer the start of the next frame
  - log overruns. If too many overruns, need to do fault recovery

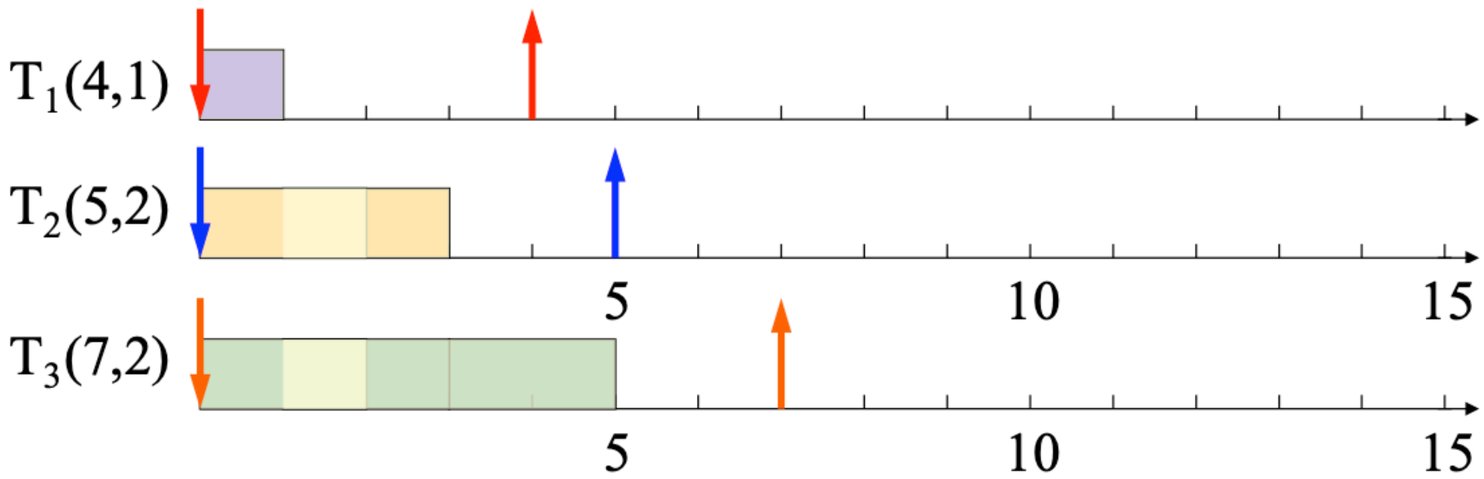


Priority-driven algorithms

- A class of algorithms that never leave the processor(s) idle intentionally (i.e., work-conserving)
- 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 a priority of a task changes
  - At each scheduling decision time, a ready task with the highest priority is executed
- If non-preemptive, scheduling decisions are made only when a processor becomes idle.
- A scheduling algorithm is called static if priorities are assigned to tasks once for all time (that is, fixed).
- The algorithm is called dynamic if priorities can change over time.

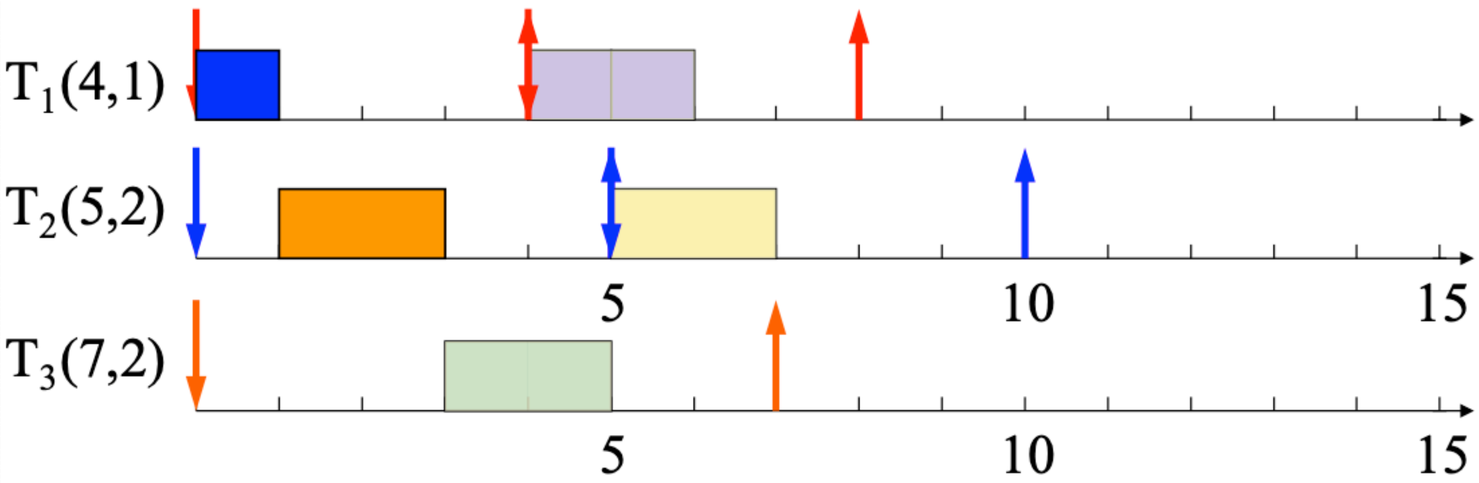
Earliest Deadline First (EDF)

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



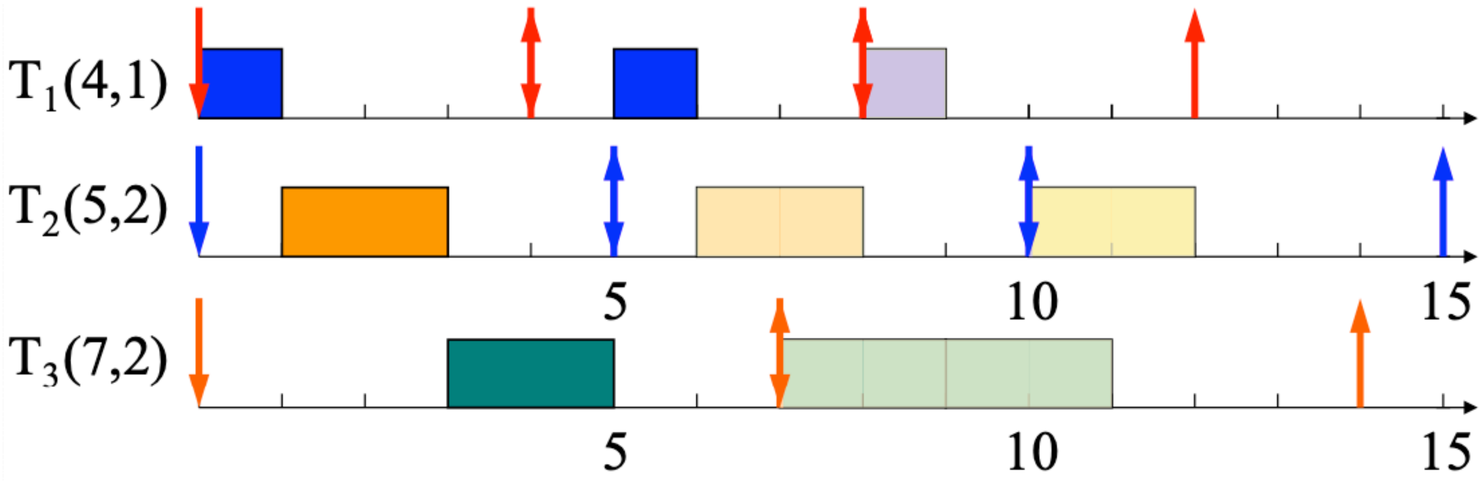
EDF

- Executes a job with the earliest deadline



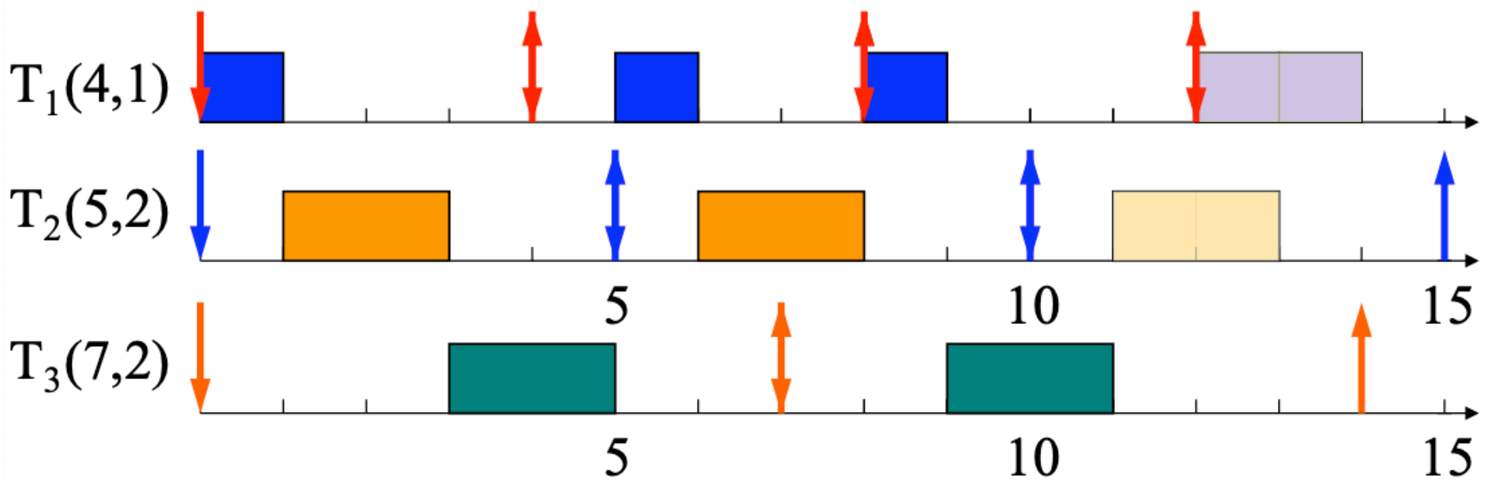
EDF

- Executes a job with the earliest deadline



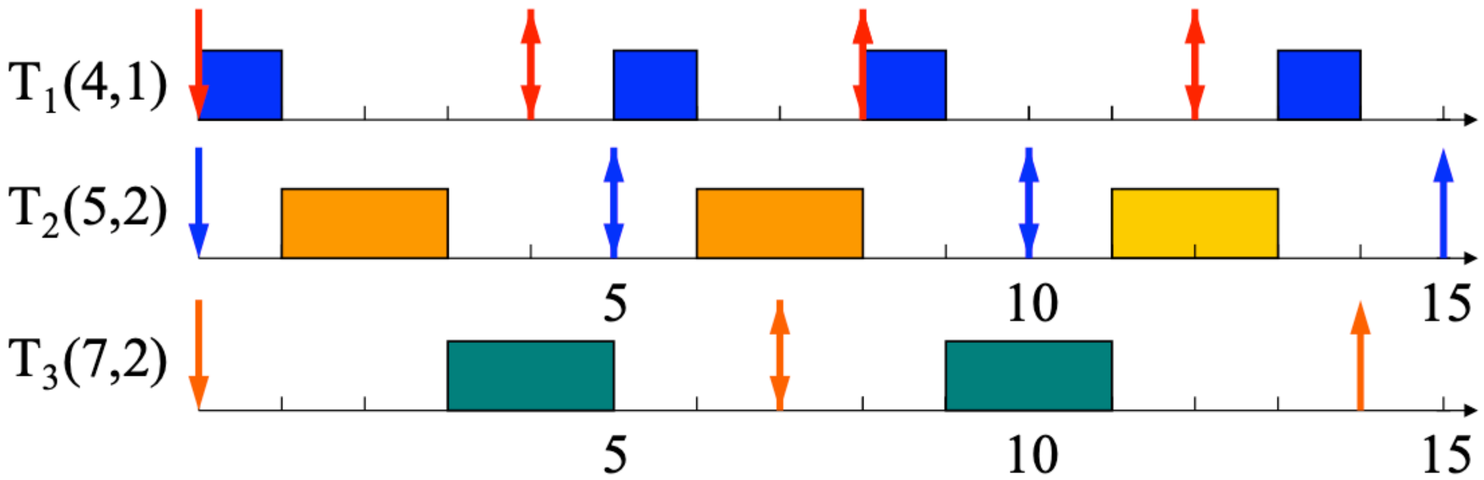
EDF

- Executes a job with the earliest deadline



EDF

- 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

- Resource (CPU) utilization : $U = \frac{e}{p}$
- 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.

Utilization Bounds

- Question: Is there a threshold U_{bound} such that
  - When $U < U_{bound}$, all deadlines are met
  - When $U > U_{bound}$, some deadlines are missed

EDF- Utilization Bound

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

$$\sum U_i \leq 1$$

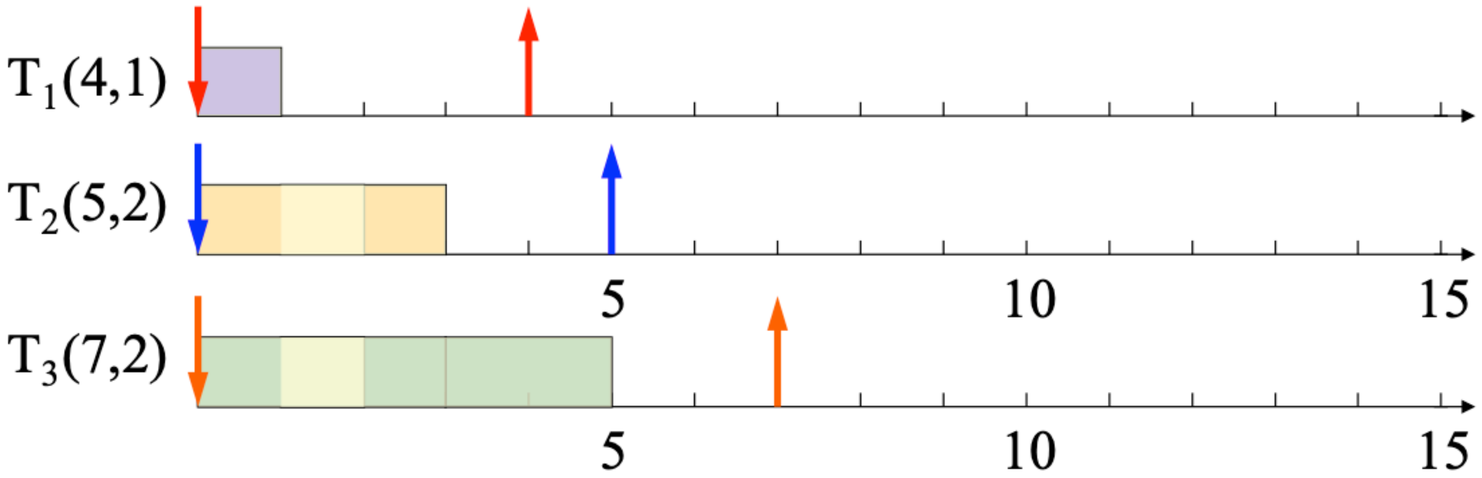
- Lie and Layland, “Scheduling algorithms for multi- programming in a hard-real-time environment”, Journal of ACM, 1973.

Schedulable Utilization

- Utilization of a periodic task (p, e, d),
  - $u = e/p$: The fraction of time that the task keeps the processor busy
  - $T = \{(p_i, e_i, d_i)\}$ contains n tasks
  - $U = \sum_{i=1}^{n} e_i/p_i$ total utilization of task systems
- **A system of n tasks with $d_i = p_i$ can be feasibly scheduled if and only if $U \leq 1$**

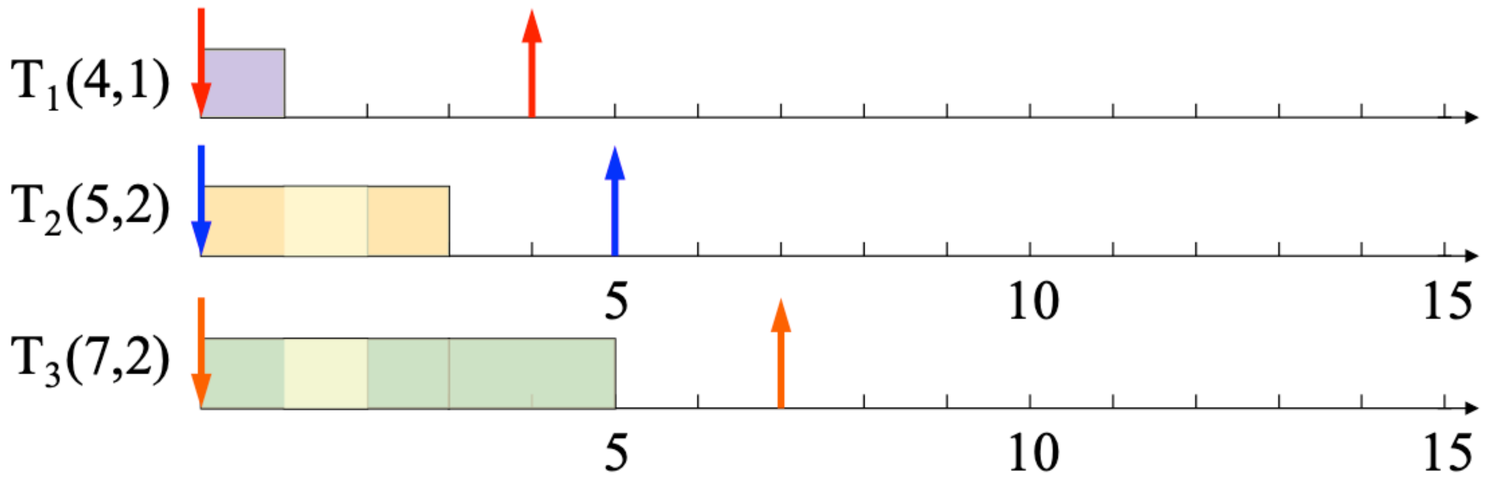
Rate Monotonic

- 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



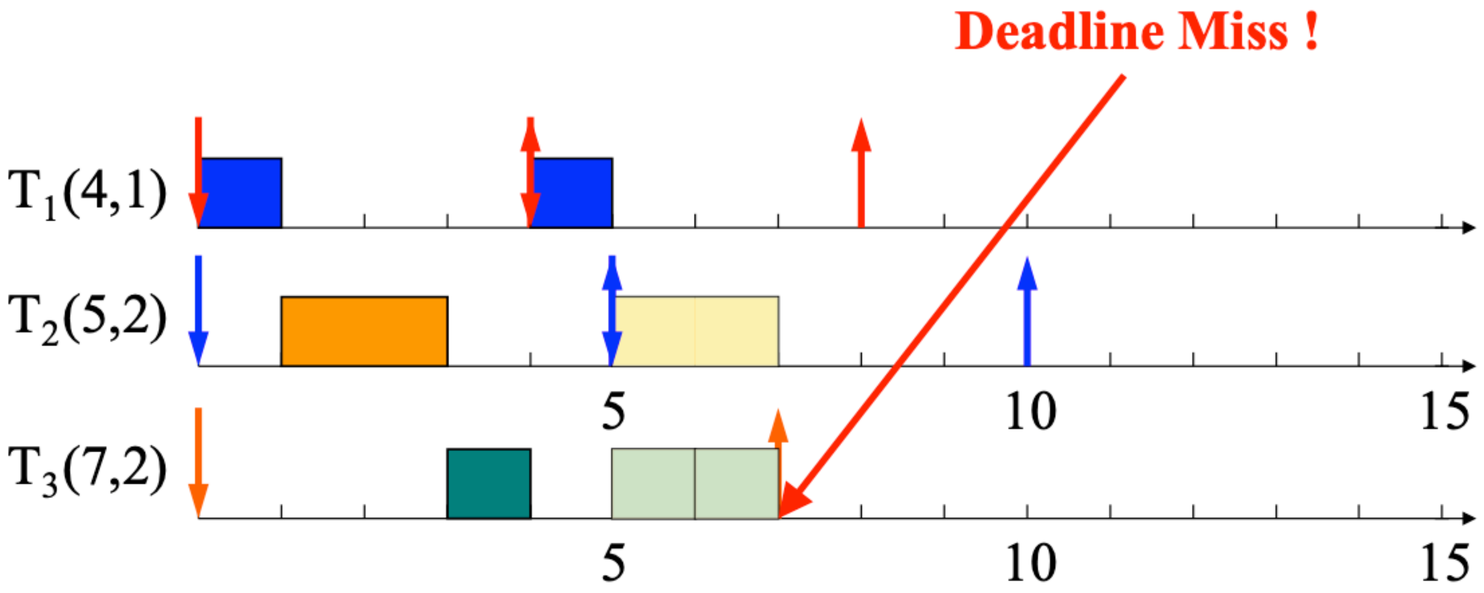
Rate Monotonic

- Executes a job with the shortest period



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

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

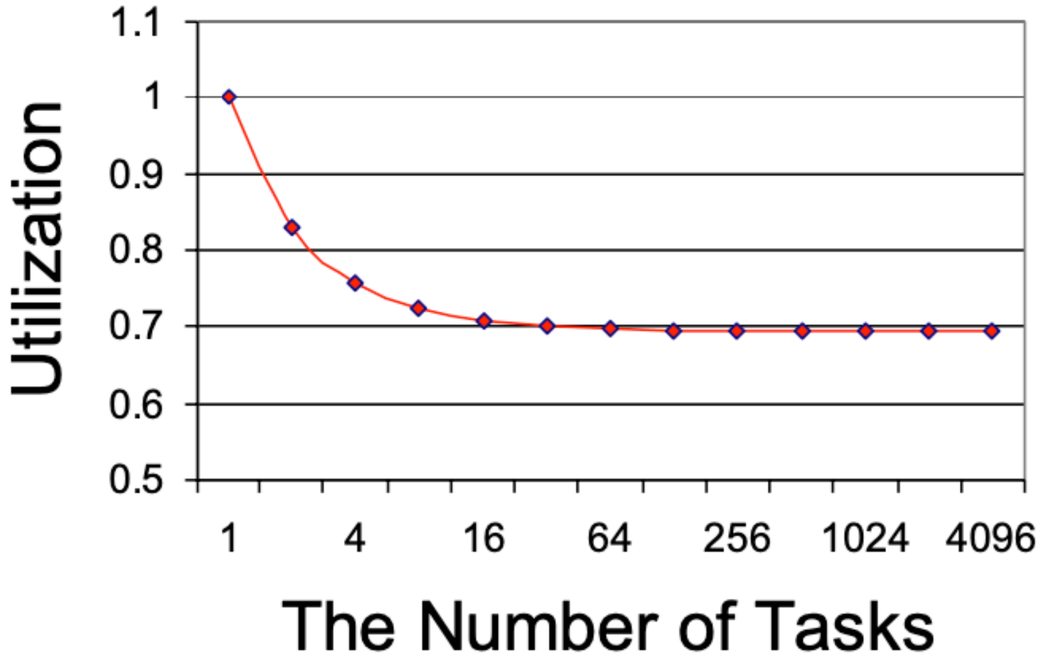
- Example: $T_1(4,1), T_2(5,1), T_3(10,1)$ is schedulable under RM since

$$\sum U_i = 1/4 + 1/5 + 1/10 = 0.55$$

$$3(2^{1/3} - 1) \approx 0.78$$

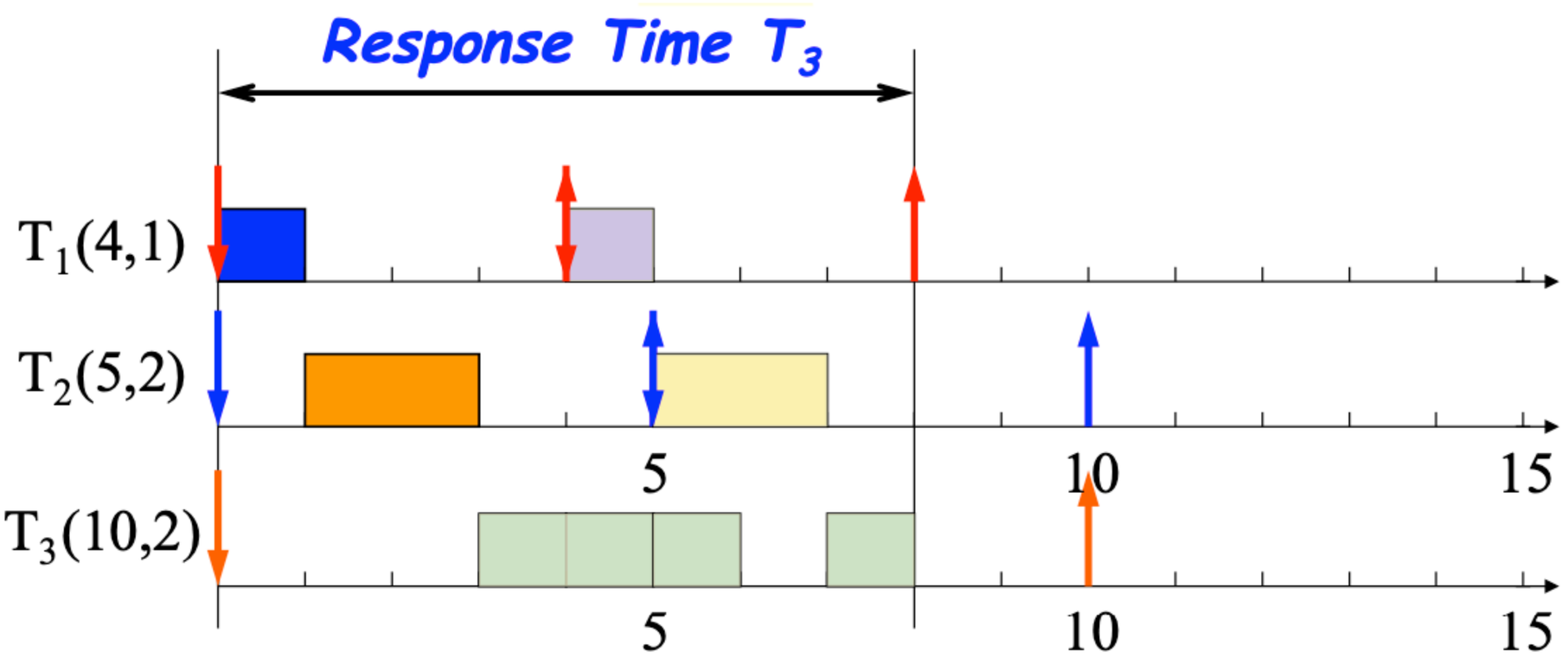
RM - Utilization Bound

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



Response Time

- Duration from released time to finish time



Response Time

- Response Time (R_i)

$$R_i = e_i + I$$

  - where I is the interference from higher priority tasks

- If the task T_i is schedulable if and only if $R_i \leq D_i$

Response Time

- During R_i, task j (with $pri_j > pri_i$) is released $\left\lceil \frac{R_i}{p_j} \right\rceil$ number of times

- Total interference by task j is given by

$$\left\lceil \frac{R_i}{p_j} \right\rceil e_j$$

  - The ceiling function, $\lceil x \rceil$: the smallest integer greater than x, e.g., $\lceil 0.25 \rceil = 1$

Response Time

- Worst Case Response Time

$$R_i = e_i + \sum_{k \in hp(i)} \left\lceil \frac{R_i}{p_k} \right\rceil e_k$$

  - where $hp(i)$ is the set of tasks with higher priories than task *I*

Response Time

$$R_i = e_i + \sum_{k \in hp(i)} \left\lceil \frac{R_i}{p_k} \right\rceil e_k$$

- Solve by forming a recurrence relationship

$$R_i^{n+1} = e_i + \sum_{k \in hp(i)} \left\lceil \frac{R_i^n}{T_k} \right\rceil e + k$$

Response Time - Example

$$R_i^{n+1} = e_i + \sum_{k \in hp(i)} \left\lceil \frac{R_i^n}{T_k} \right\rceil e + k$$

Task	Period	Computation Time	Priority
a	7	3	3
b	12	3	2
c	20	5	1

$$R_c^0 = 5$$

$$R_c^1 = 5 + \lceil 5/7 \rceil 3 + \lceil 5/12 \rceil 3 = 14$$

$$R_c^2 = 5 + \lceil 11/7 \rceil 3 + \lceil 11/12 \rceil 3 = 14$$

$$R_c^4 = 5 + \lceil 14/7 \rceil 3 + \lceil 14/12 \rceil 3 = 17$$

$$R_c^4 = 5 + \lceil 17/7 \rceil 3 + \lceil 17/12 \rceil 3 = 20$$

$$R_c^5 = 5 + \lceil 20/7 \rceil 3 + \lceil 20/12 \rceil 3 = 20$$

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.

In Next Class

- RTOS Concept