Lecture 7
Real Time Task Scheduling

Forrest Brewer

Real Time

- ANSI defines *real time* as

  "*A Real time process is a process which delivers the results of processing in a given time span*"

  - A data may require processing at a priori known point in time, or it may be demanded without any priori knowledge
- Correctness of computation
- Deadlines (latest acceptable time)
  - Soft deadline
    - Diminished functionality as deadlines are missed
    - System does not 'fail'
  - Hard deadline
    - System fails (X-29 wings fall off…)

Real Time

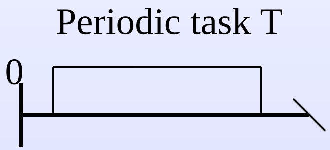
- Processing guarantees for time-critical applications:
  - Predictably fast response to time-critical events & accurate timing information
    - Jitter issues
  - High degree of schedulability
    - High degree of resource utilization below which the processing guarantee is a question…
  - Stability under transient overload
    - Under system overload, critical jobs processing of must be ensured
    - Priority Scheme, process preemption
  - Sharing of Resources
    - Management of Arbitration
  - Low Overhead

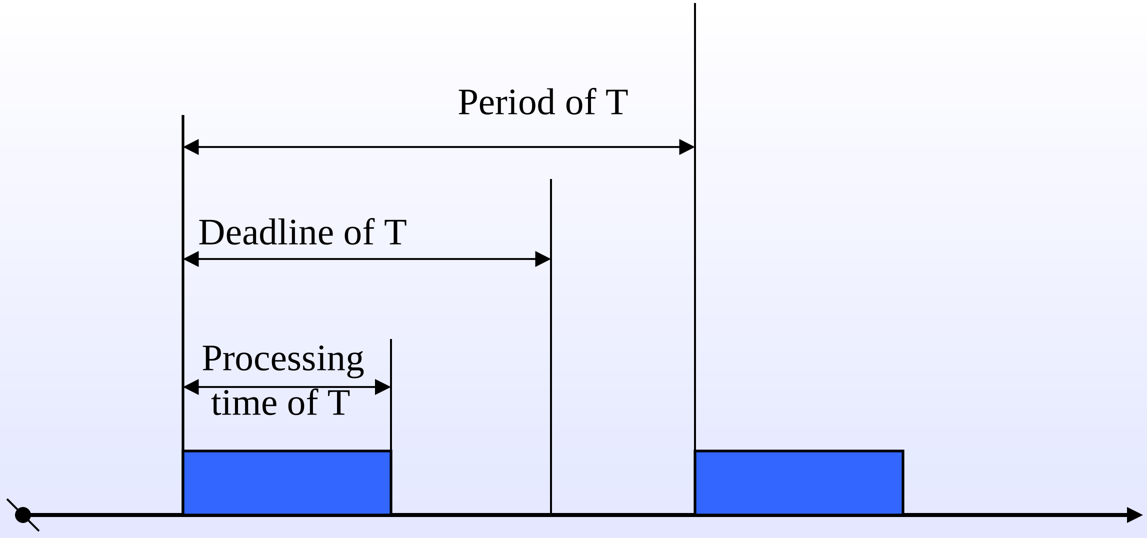
Five Characteristics of Real-Time Operating Systems

- ***Determinism***: concerned with how long an operating system delays before acknowledging an event
- ***Responsiveness***: concerned with how long after acknowledgment, it takes an operating system to finish the event (interrupt) service
  - Determinism and responsiveness together make up the response time to external events which are crucial for real-time systems
- ***User control***: allow the user (dynamic?) fine-grained control over task priority
- ***Reliability***: a transient failure may cause financial loss or major equipment damage or even loss of life.
- ***Fail-soft operation***: during overload, continued operation at a reduced level of service

System Modeling in RT Scheduling

– Tasks are the schedulable unit of the system.
– A task is characterized by timing constraints and resource requirements.
– Periodic task (T)
    – *processing time*
    – *deadline*
    – *period*





Real time scheduling: Periodic system model

- Task: schedulable entity
  - Processing of separate tasks are assumed mutually independent

- Timing constraints of a periodic task τ_i is specified by (s, e, D, p)
  - s_i-(scheduled) Starting Time of Task i
  - e_i-Processing time of i
  - f_i-Finish time of i
  - D_i-Deadline of i
  - p_i-Period of i
  - r_i-Rate of i = ($1/p_i$)

Real time scheduling: Periodic system model

- Tasks can be
  - Preemptive
  - Nonpreemptive
- Guarantee ratio
  - Processing time used by guaranteed tasks versus total processing time
- Utilization:

$$U = \sum_{i=1}^{n} \frac{e_i}{p_i}$$

System Model - Assumptions and Notation

- Assumptions:
  - Periodic tasks without precedence relations
    - Aimed at “vertical” system decomposition
  - No OS overhead
    - time added to every task invocation
    - this is a problem for preemptive task models
  - Time Constraints (non-periodic):
    - $C = \{t1=(s_1, e_1, D_1), t3=(s_3, e_3, D_3), t2=(s_2, e_2, D_2), \ldots\}$

Non-Repeating Schedule

$$A = \{(s_i, f_i, t_i) \mid i = 1, \ldots, n\}$$

$$s_i < f_i$$

$$f_i \leq s_{i+1}$$

$$t_i = k \Rightarrow S_k \leq s_i \ \& \ f_i \leq D_i$$

A schedule is a set of execution intervals
s=start time of interval,
f=finish time of interval,
t=the task executed during the interval

$$A(\tau_k) = \{a = (s_i, f_i, t_i) \mid a \in A \ \& \ t_i = k\}$$

$$Feasible\ Schedule \Rightarrow \sum_{(s_i, f_i, k) \in A(\tau_k)} f_i - s_i \geq e_k$$

A schedule is feasible if every task τ_k receives at least e_k seconds of CPU execution in the schedule

Note: a task may be segmented into several execution intervals

Schedule Example

- C={t1=(0,8,13), t2=(3,5,10), t3=(4,7,20)}
  - A={(0,3,t1),(3,8,t2),(8,13,t1),(13,7,t3)} is a feasible schedule
    - for t1, (3-0) + (13-8) = 3 + 5 = 8
- C={t1=(1,8,12), t2=(3,5,10), t3=(4,7,14)}
  - No feasible schedule

Real-Time Scheduling Policies

- Static table-driven
  - Suitable for periodic tasks/earliest-deadline first scheduling
  - Requires Static analysis of feasible schedule
- Static priority-driven preemptive ➔ rate monotonic algorithm
  - Static analysis to determine priority
  - Traditional priority-driven scheduler is used
- Dynamic planning-based (evaluate priorities on the fly)
  - Create a schedule containing the previously scheduled tasks and the new arrival → if all tasks meets their constraints, the new one is accepted
- Dynamic best effort
  - No feasibility analysis is performed
  - Assigned a priority to the new arrival ➔ then apply earliest deadline first
  - System tries to meet all deadlines and aborts any started process whose deadline is missed

Periodic tasks: Example

- Suppose the tasks tsk 1…tsk 3 have the following properties

name	execution time [msec]	period [msec]	Deadline [msec]
tsk 1	20	100	100
tsk 2	40	150	150
tsk 3	100	350	350

- The tasks get assigned priorities
- Once assigned, these priorities do not change;
- The tasks are scheduled according to their priorities, i.e. a ready task with highest priority is executed until a higher priority task becomes ready. Such higher priority task then pre-empts the lower priority task.

Time Line Scheduling (Cyclic Scheduling)

- **Time Line Scheduling** (Off-line scheduling strategy)– Divide the time line into time slices for scheduling tasks, e.g. use the Greatest Common Divisor of the Task Periods as the time slice:

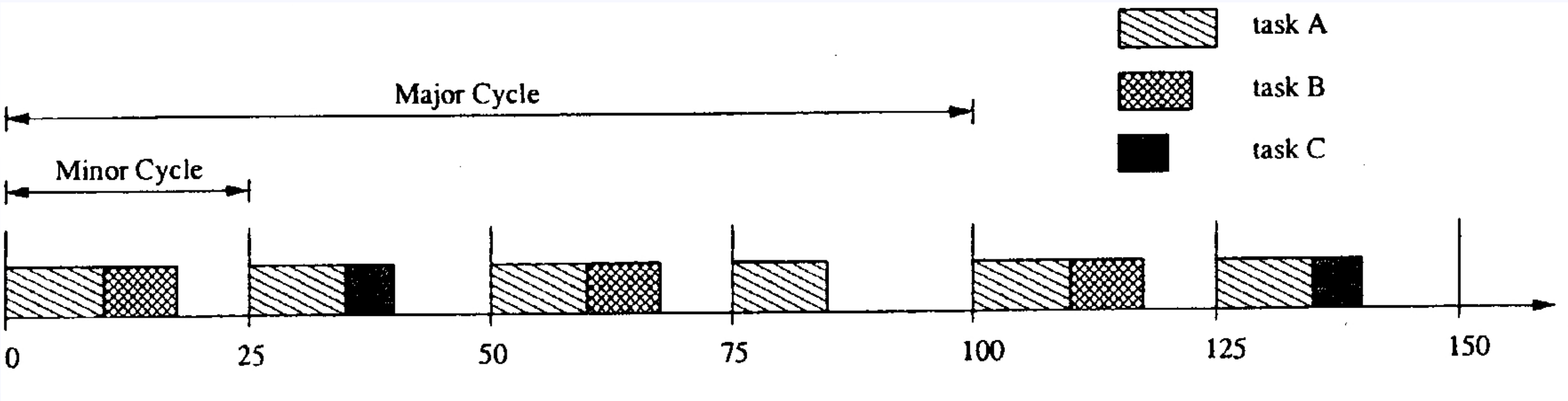


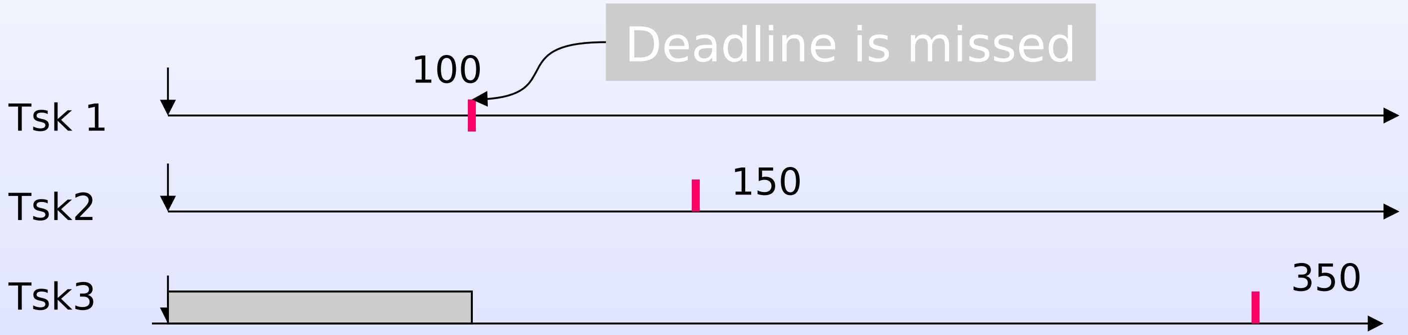
Figure 4.2 Example of timeline scheduling.

Execution time based priority

- Suppose we assign the priorities depending on their (worst) computation time, I.e. the longer the computation time the higher priority

	name	execution time [msec]	period [msec]	Deadline [msec]
L	tsk 1	20	100	100
M	tsk 2	10	150	150
H	tsk 3	100	350	350

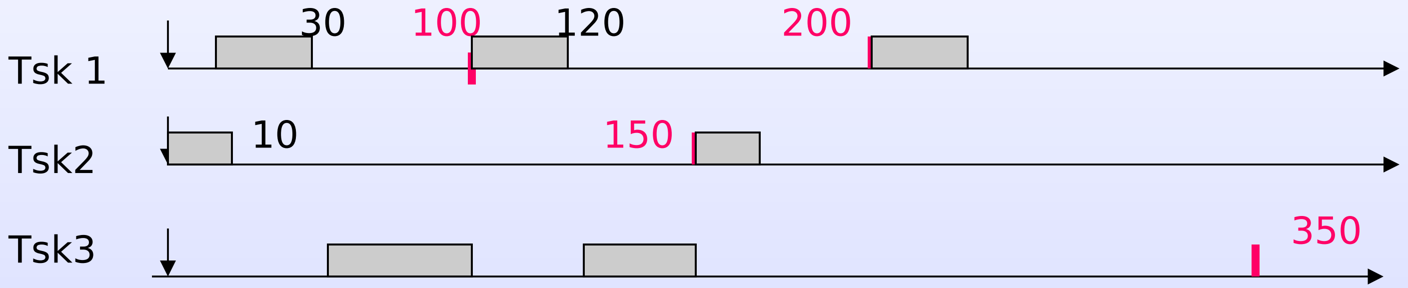
- What will be then the execution?



Execution time based priority

- Suppose we assign the priorities depending on their (worst) computation time, I.e. the shortest the computation time the higher priority

	name	execution time [msec]	period [msec]	Deadline [msec]
M	tsk 1	20	100	100
H	tsk 2	10	150	150
L	tsk 3	100	350	350



Questions

- In this specific case, this priority assignment works
  - Does it always work?
- If it does not work in this specific case is there an assignment that always works?
- Is there a better way (than trace analysis) to decide whether an assignment works?

Rate monotonic scheduling

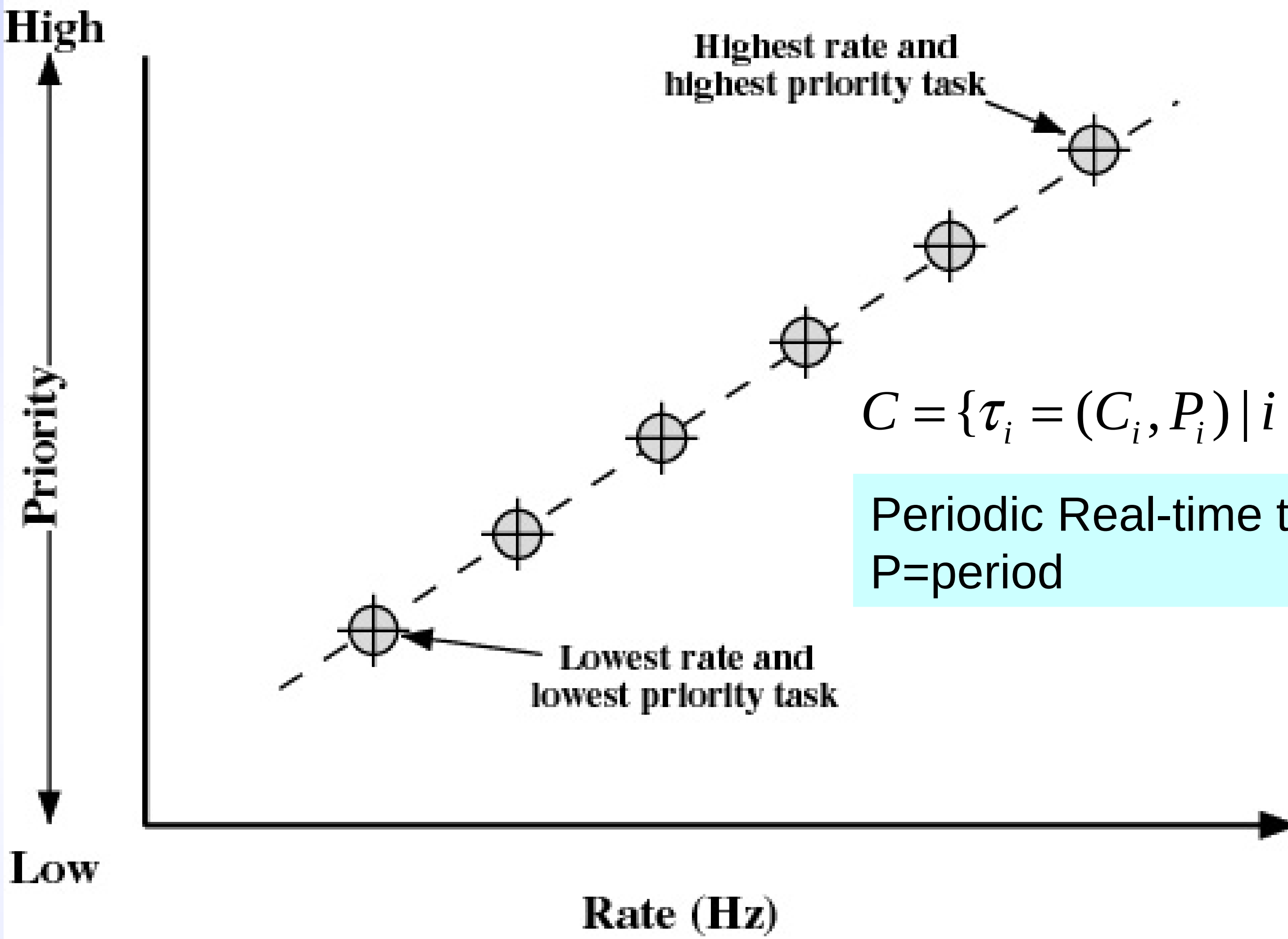
- Classic paper, Liu & Layland, JACM 1973
- m tasks, with periodicities (P_i), deadlines ($D_i = P_i$) and computation time (C_i)
- Monotone Priority:
  - task frequency = f_i = task priority = 1/ P_i),
  - **Always Scheduable if** (but **not** only-if):

$$U = \sum_{i=1}^{n} \frac{C_i}{P_i} \leq n(\sqrt[n]{2} - 1)$$

  - Simple, elegant result
    - No tight upper bound on the Utilization metric is available
    - a trivial upper bound can be summation of the Task Utilization <= 1
    - Note: $\lim_{n \to \infty} n(\sqrt[n]{2} - 1) = \ln(2) = 0.6931...$

Rate Monotonic Scheduling

- Assumptions
  - Tasks are periodic
  - Tasks do not communicate with each other
  - Tasks are scheduled according to priority, and task priorities are fixed (static priority scheduling)
- Note
  - A task set may have feasible schedule, but not by using any static priority schedule
  - Feasible static priority assignment
- Rate Monotonic Scheduling (RMS)
  - Assigns priorities to tasks on the basis of their periods
  - Highest-priority task is the one with the shortest period
  - If $p_h < p_l$, then $Priority_h > Priority_l$



$$C = \{\tau_i = (C_i, P_i) \mid i = 1, \ldots, n\}$$

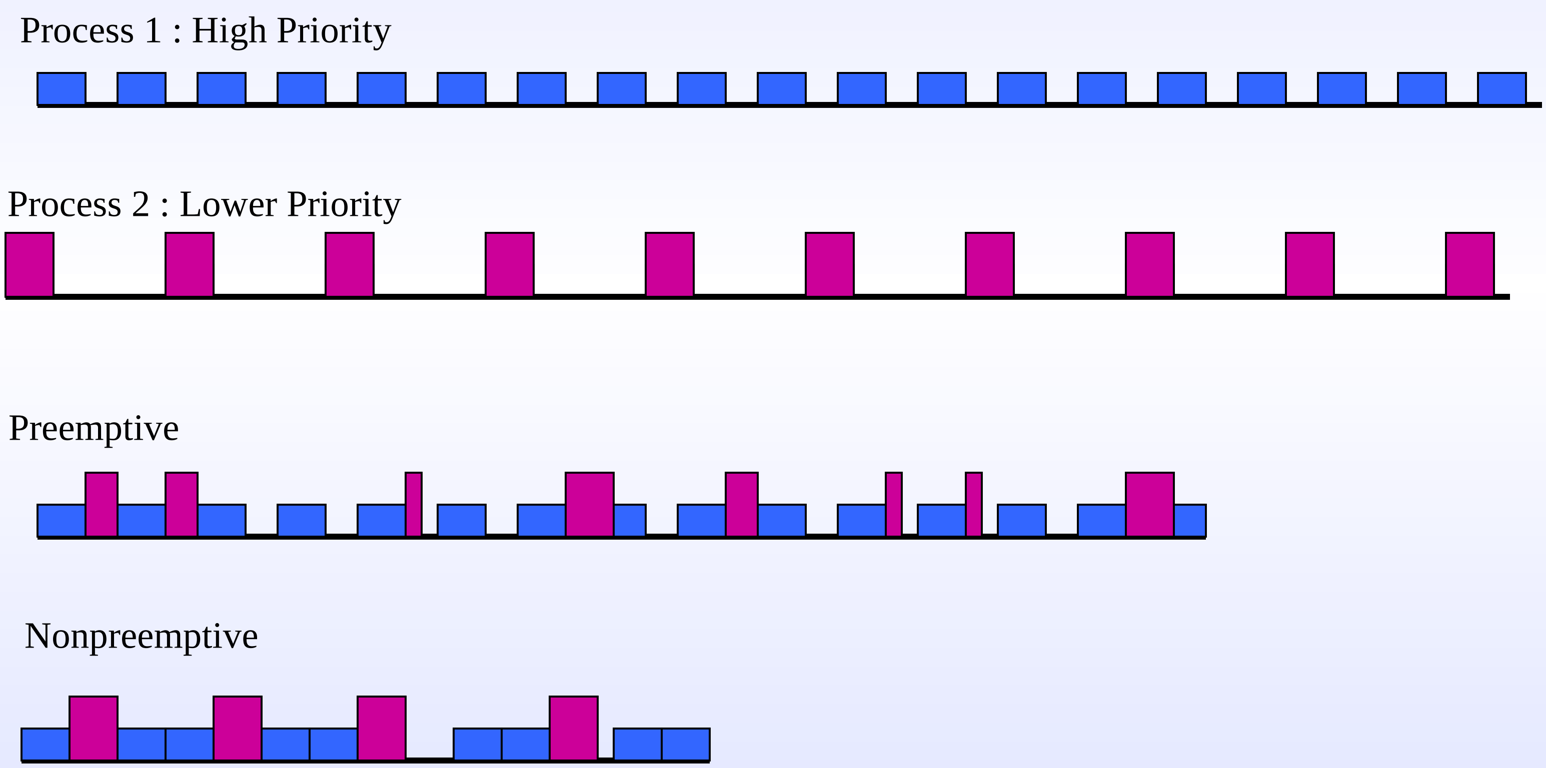
Periodic Real-time task set
P=period

The start time of a new instance of a job is the deadline of the last instance

Figure 10.8 A Task Set with RMS [WARR91]

Rate Monotonic Scheduling

Process Priority determined by arrival rate (since rate = 1/period)



Example of Rate Monotonic Scheduling

- P1: C1 = 1; T1= 2; C1/T1= 0.5
- P2: *C2* = 1; T2= 3; C2/T2= 0.333
- P3: C3= 1; T3= 6; C3/T3= 0.166

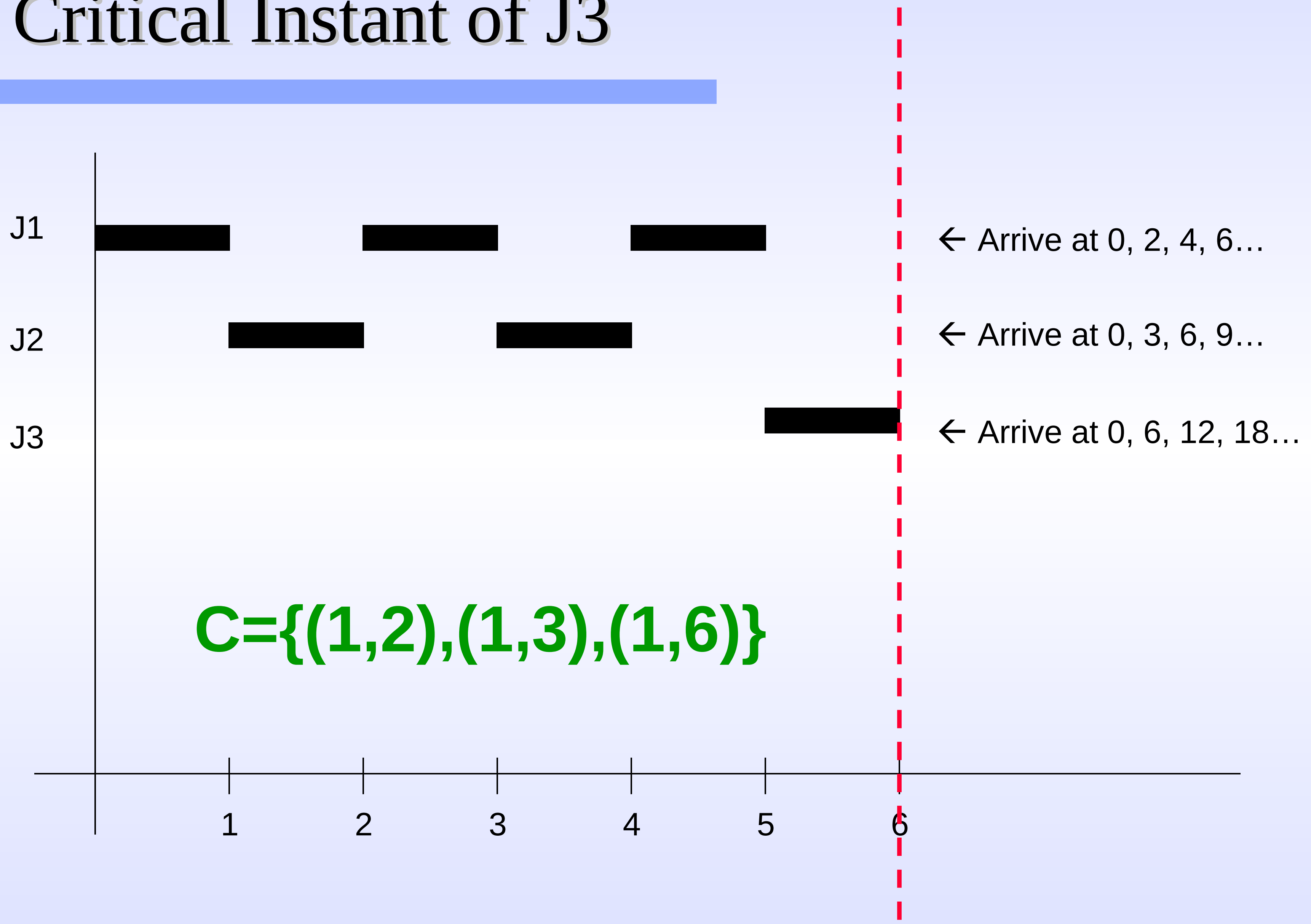
Total utilization = 1.0

Since: $1.0 <= 1.0 < 3\,(2^{1/3} - 1) = 0.779$

May or may not be schedulable…

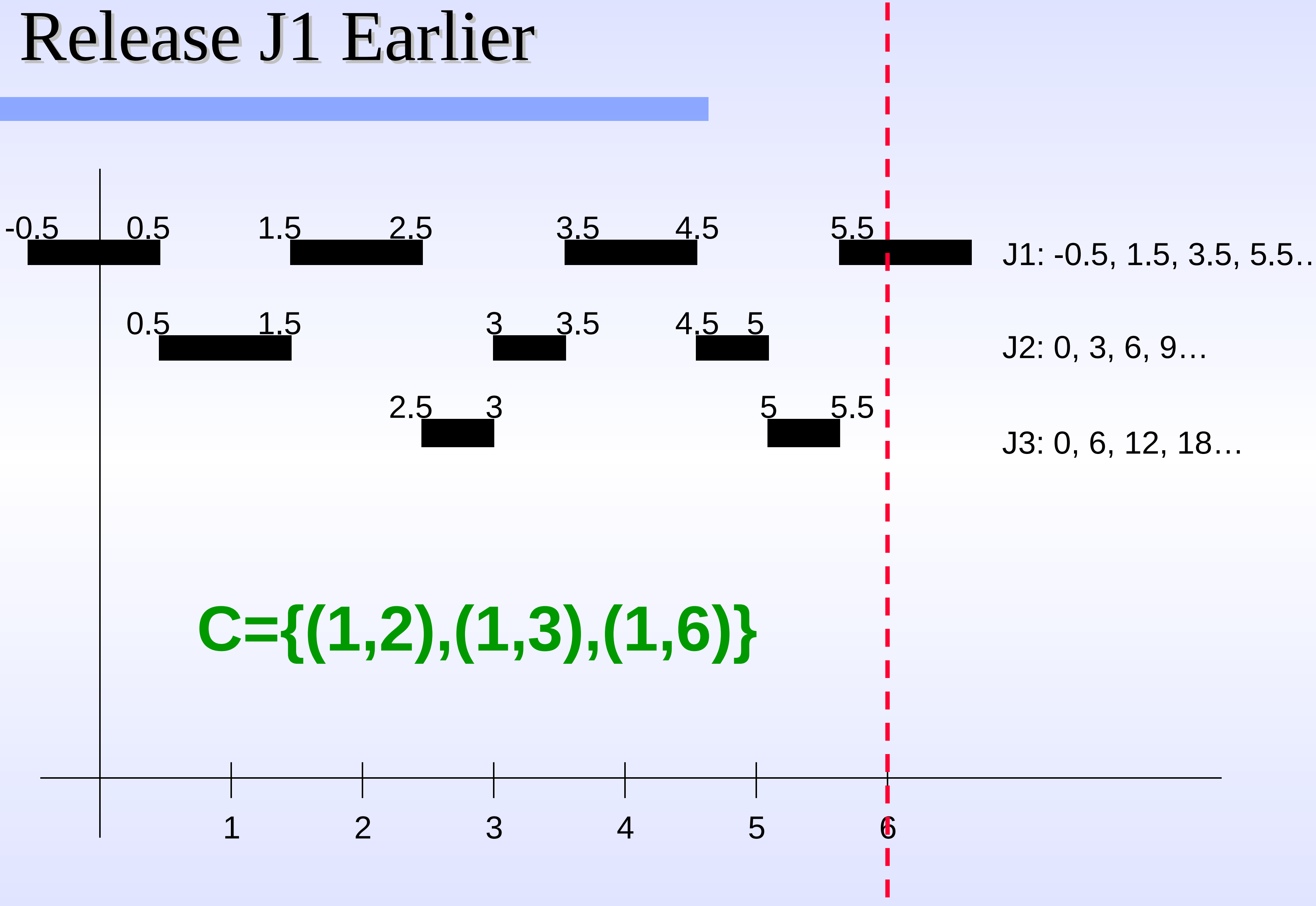
However if C1 = ½ the total utilization would be 0.75 and the system will *always* be schedulable.

Critical Instant of J3

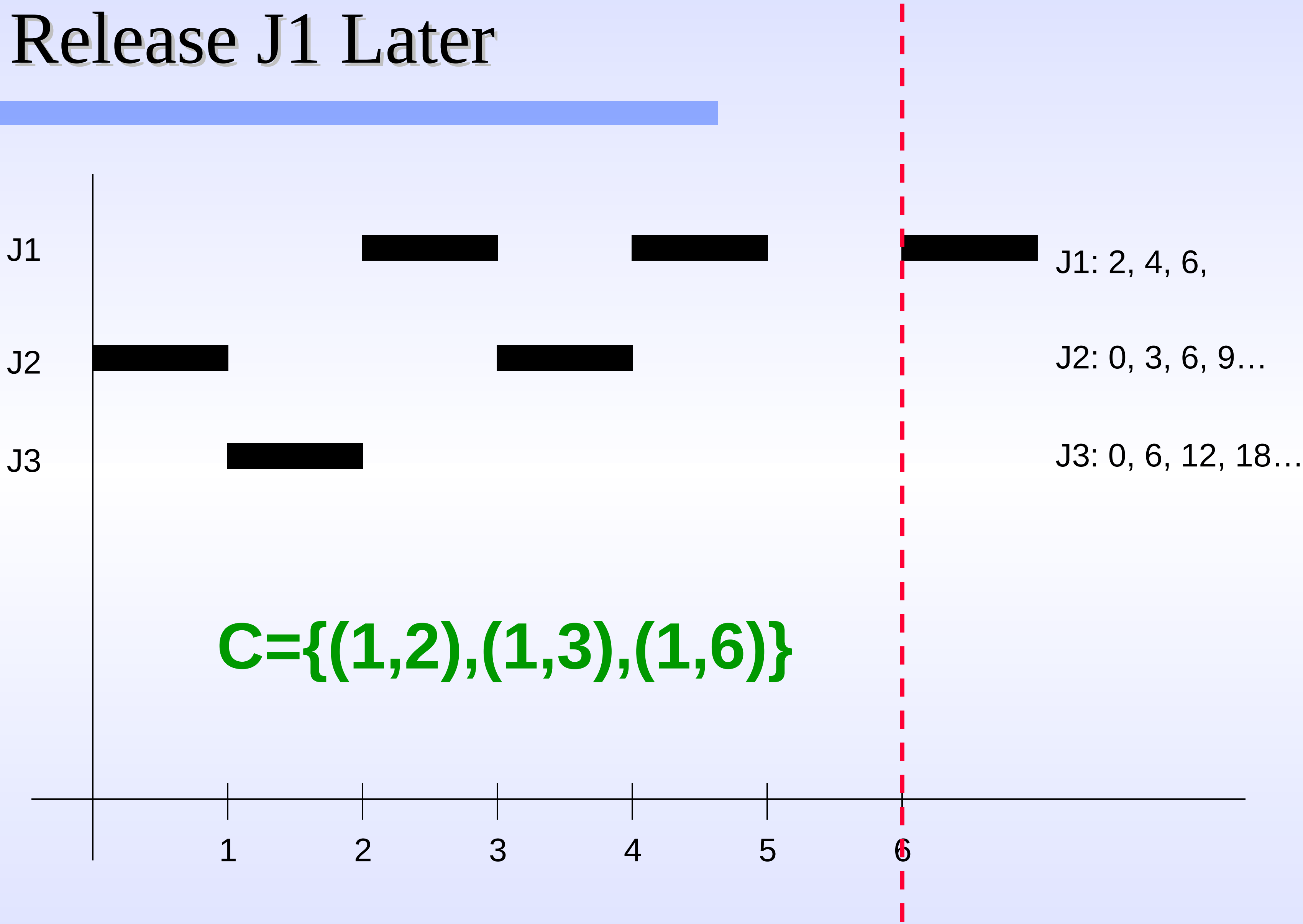


C={(1,2),(1,3),(1,6)}

Release J1 Earlier



Release J1 Later



RMS is Optimal…

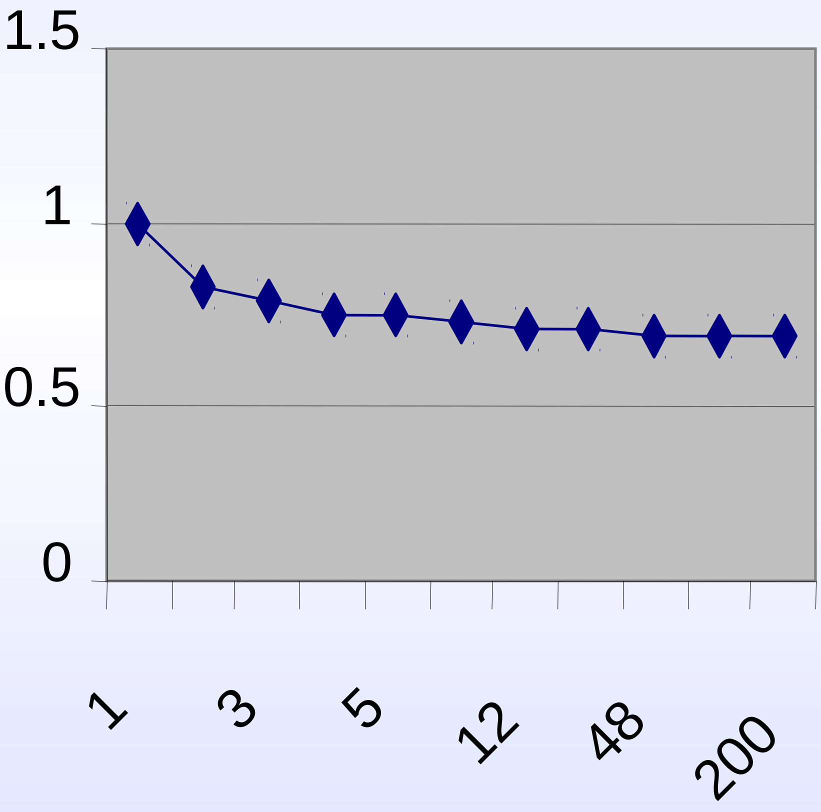
- If a set of periodic tasks has a feasible static priority assignment, RMS is a feasible static priority assignment

- Outline of Proof:
  - Hint: if there is a non-RMS feasible static priority assignment
  - List the tasks in decremented order of priority
  - Because non-RMS, there must be τ_i and τ_{i+1} such that

  $T_i > T_{i+1}$
  - Prove exchange τ_i and τ_{i+1} and the schedule is feasible
    - Repeat the priority exchange

Value of the threshold factor

m	m*(2^{1/m}-1)
1	1
2	0.828427125
3	0.77976315
4	0.75682846
5	0.743491775
6	0.73477229
12	0.713557132
24	0.703253679
48	0.698176077
96	0.695655573
200	0.694349702

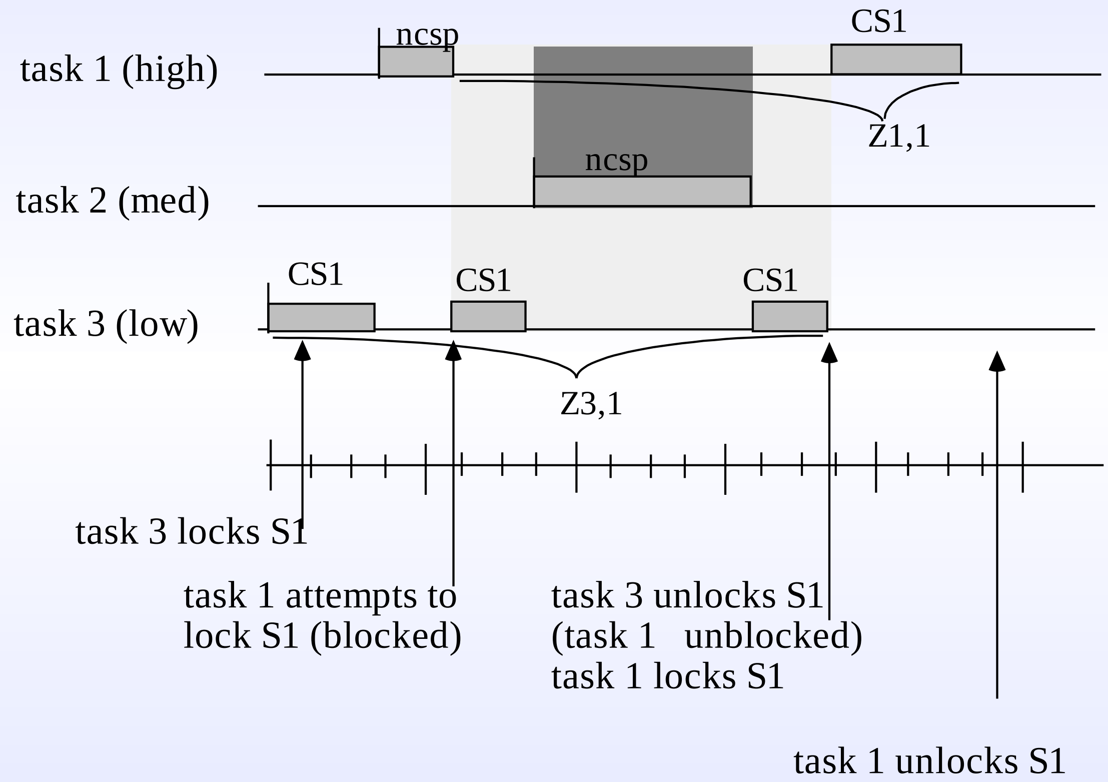


Priority Inversion

- RMA assumption: the processes are *independent*
- Issue: real RT-processes often are required to share resources that are unique to the system
  - under such circumstances processes can block each other. In particular: execution of high priority task can be blocked by execution of a low priority task which has locked a required resource
  - In other word: priorities are effectively inverted

Priority inversion: Example

- tasks 1 and 3 share a resource (S1)
- prio(task1) >prio(task2) >prio(task3)
- Task 2 can run for any amount of time… it blocks Task 3 from finishing and unlocking resource needed by task 1.
- Infamous Mars pathfinder Priority Inversion Bug



Example Continued

- Conclusion
  - High priority task (task 1) is blocked by low priority task (task 3)
  - the blocking period can be arbitrarily long
- Possible solutions:
  - no preemption during critical section (Interrupt-Masking Protocol)
  - good for short critical sections, otherwise bad: unnecessary CS blocking
  - in critical section (CS), raise the task's priority to a level higher than all tasks ever using that CS (Priority inheritance protocol)
    - In example, this prohibits Task 2 from preempting Task 3
  - disadvantage: unnecessary (priority) blocking, possibility of deadlock

Priority Ceiling Protocol

- Each ***resource*** is assigned a priority equal to that of the highest priority task that uses that resource.
- Tasks then inherit the priority of the resource while it is locked
- Tasks are not scheduled if any resource it may need it already locked by another task

- This scheme prevents improper nesting of the priorities of critical section and thus prevents deadlocks

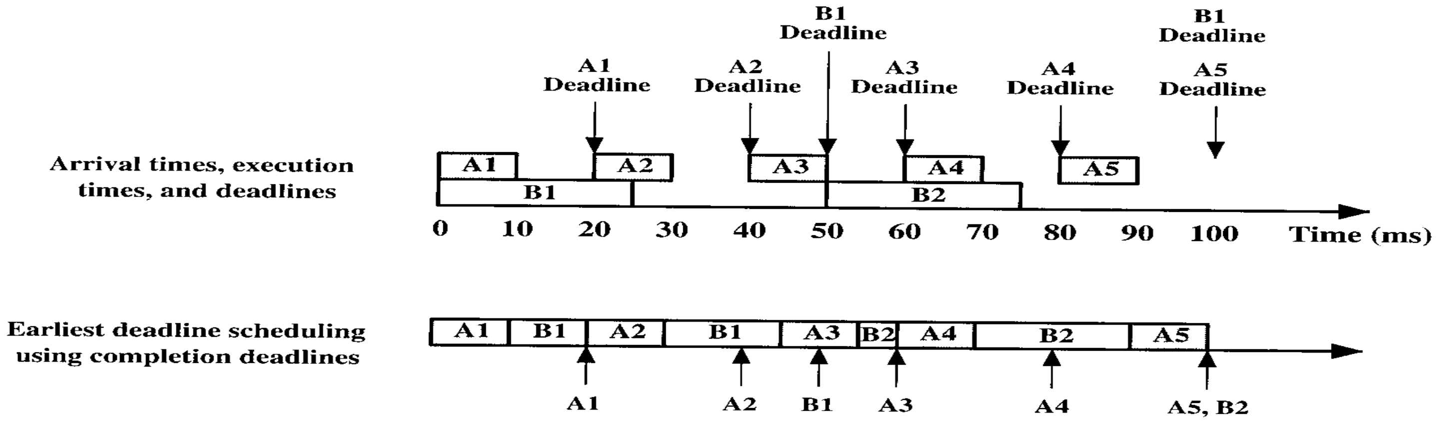
Ref: Lui Sha, Ragunathan Rajkumar, and John P. Lehoczky (September 1990). "Priority Inheritance Protocols: An Approach to Real-Time Synchronization". IEEE Transactions on Computers 39 (9): 1175–1185. doi:10.1109/12.57058

Deadline Scheduling

- Deadline Scheduling: the task which has the earliest deadline, will be scheduled first

- A system that collects and processes data from two sensors, A and B. The deadline for collecting data from sensor A must be met every 20 ms, and that for B every 50 ms. It takes 10 ms to process each sample of data from A and 25 ms to process each sample of data from B.

Example of Deadline Scheduling (cont.)



Earliest Deadline First Algorithm

- Very well known for real-time processing
- Optimal dynamic algorithm - produces a valid schedule whenever one exists.
- If priorities are used, earliest deadline gets the highest priority.
  - Complexity of algorithm is O(n^2).
  - Upper bound of process utilization is 100%.
  - Time Driven Scheduler - extension of EDF
    - handles overload situation by aborting tasks if overload occurs. It also removes tasks from the queue with low priority.

Earliest Deadline First (EDF) Algorithm

- Best known algorithm for real time processing
- At every new ready state, the scheduler selects the task with earliest deadline among the tasks that are ready & not fully processed
  - The processing of the interrupted task is done according to EDF algorithm later on

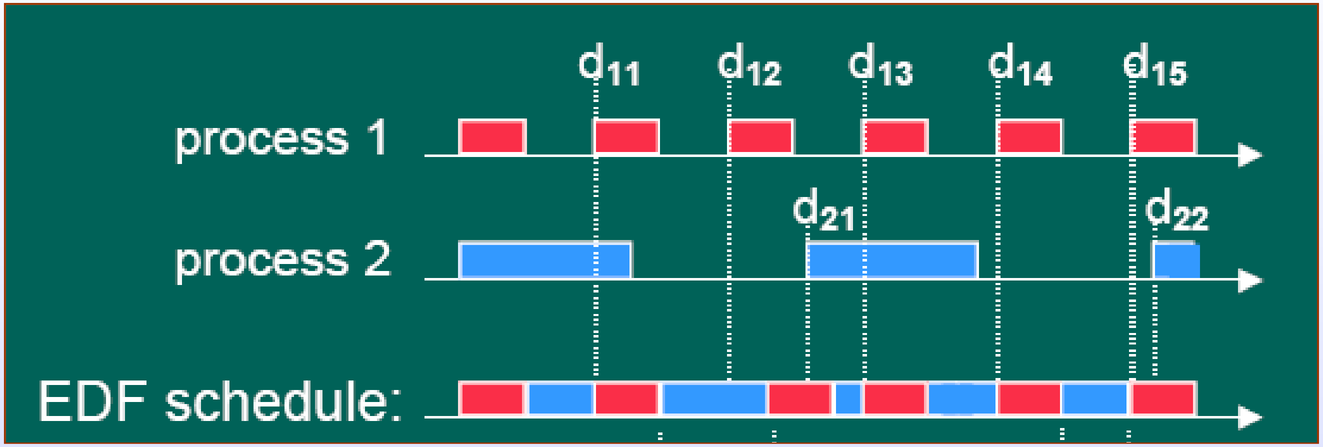
- *Optimal algorithm*
- *Dynamic algorithm*

Earliest Deadline First (EDF) Algorithm

- Optimal
  - Produces a valid schedule whenever exists
  - If a task can be scheduled using any static priority assignment, it can also be scheduled by EDF
- Dynamic
  - Schedules every instances of incoming task according to its specific demands
- Each task is assigned a priority according to its deadline
  - Highest priority to the task with earliest deadline

Earliest Deadline First (EDF) Algorithm

- Overhead in rearranging priorities
- TDS-Time driven Scheduler
  - An extension of EDF
  - Handles overload
    - Aborts all the tasks that cannot meet their deadlines anymore
    - If there is still overload, tasks with low value densities(importance of a task for the system) are removed

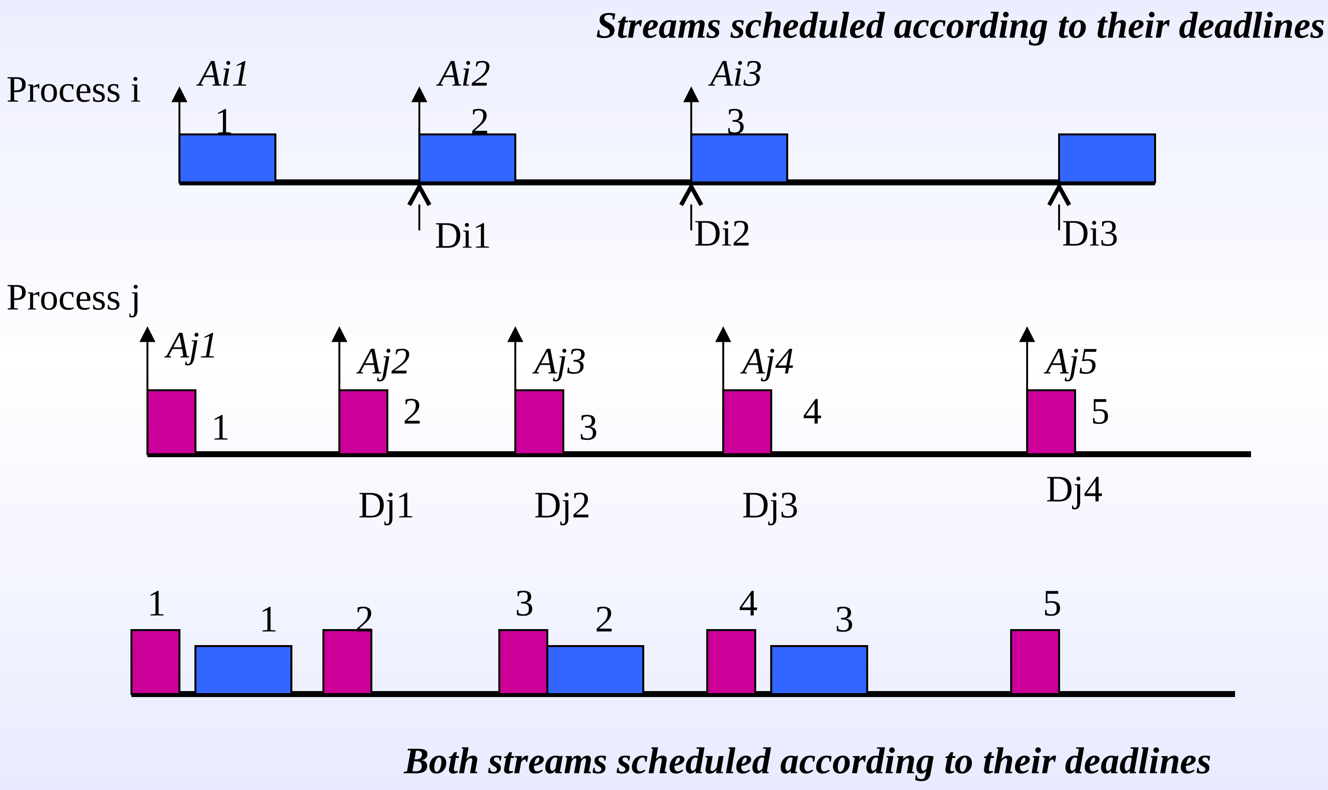


EDF

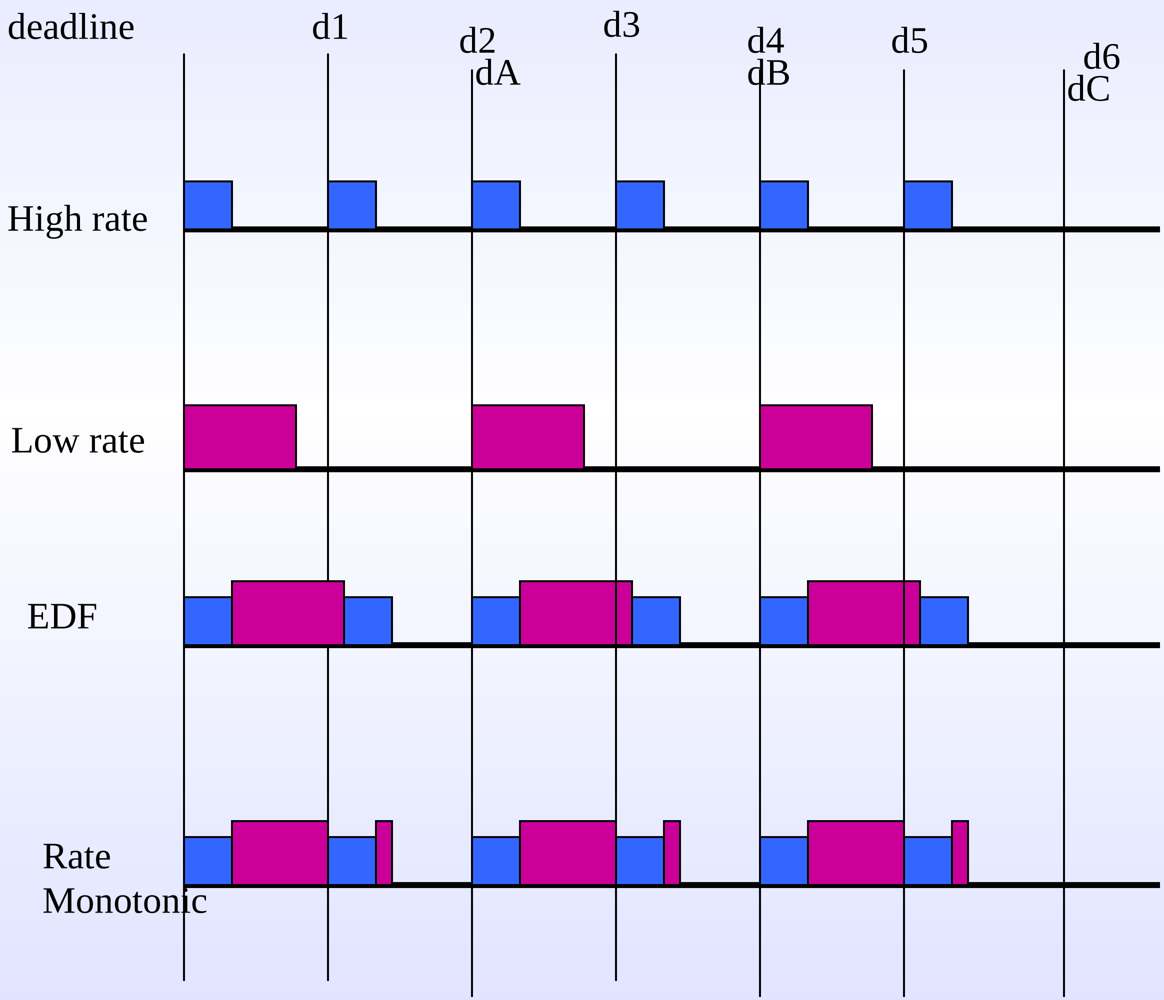
Earliest Deadline First (EDF) Algorithm

- Another variation handles every task as consisting of two parts, *mandatory part* and *optional part*
  - A task is scheduled for it's mandatory part
  - Optional part is processed, if the resource capacity is not fully utilized
  - A set of task is schedulable if all tasks can meet the deadlines of their mandatory part
  - Improves the system performance at the expense of media quality

EDF Scheduling



Comparison of EDF and Rate Monotonic Scheduling



In terms of context switching, EDF is better is more than one stream is processed concurrently.

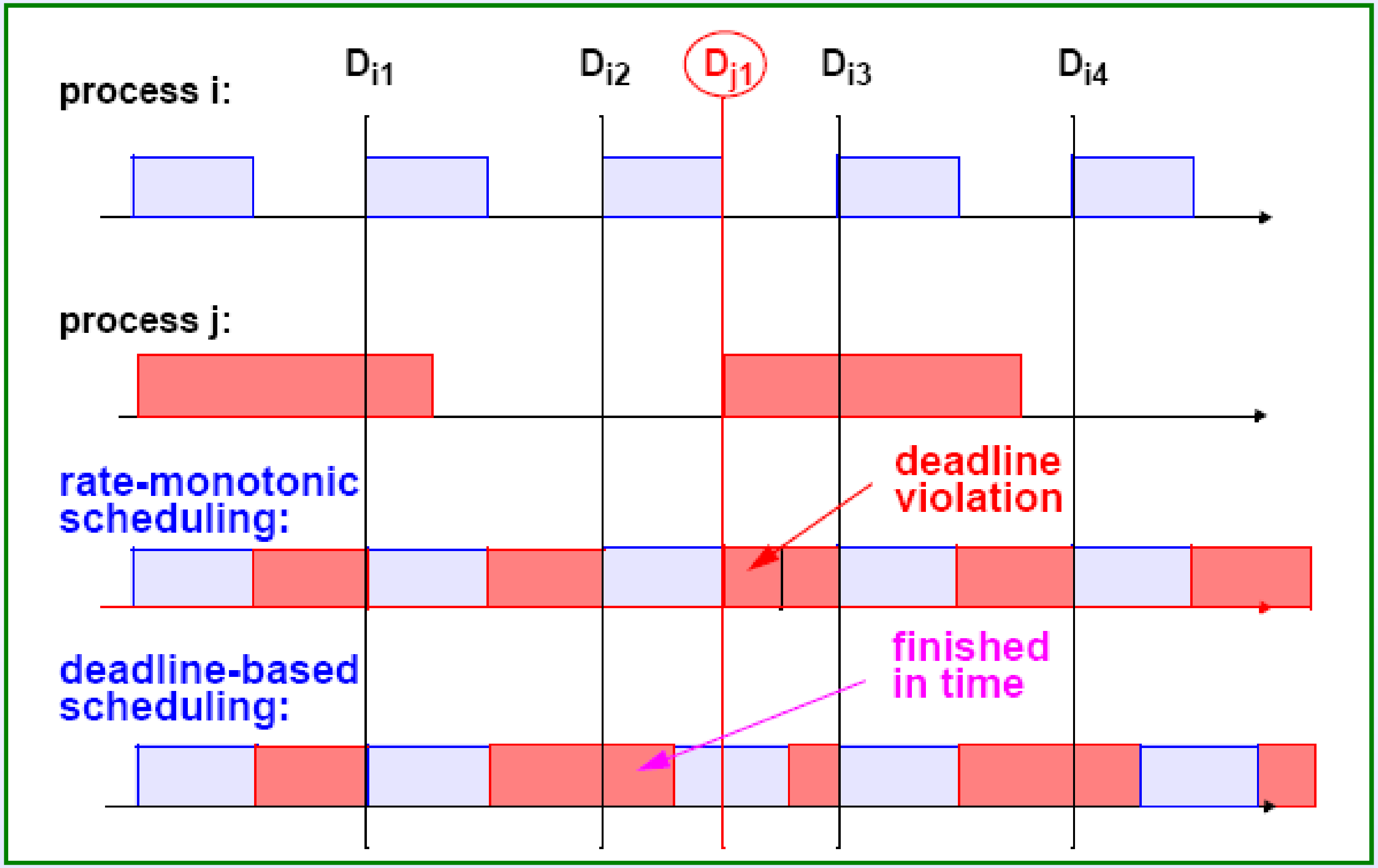
Context switches : EDF & Rate Monotonic

- Audio stream have the rate of 1/75 s/sample & video stream have the rate of 1/25 s/frame
  - Priority assigned to an audio stream is then higher
  - Arrival of messages from audio stream will interrupt video frame
  - The context switches with rate monotonic algorithm will be more than EDF in the presence of more than one stream

Processor Utilizations : *EDF & Rate Monotonic*

- Processor utilization in rate monotonic
  - Upper bound of processor utilization is determined by critical instant
  - For each number of n independent tasks t(j), a constellation can be found where maximum possible processor utilization is minimal

Processor Utilizations：EDF & Rate Monotonic



Scheduling of Periodic Dependant tasks

- The sharing of (data) resource, when the use of the resource must be atomic, and the tasks must realise deadlines , necessitates choosing:
  - a synchronisation primitive (semaphors, regions etc.)
  - an allocation policy (what happens when request is made but the resource is taken);
  - an execution priority during the use of the resource (change? of priority while using of a resource);
- Definition: Combined choice is called a *synchronisation protocol*;

Examples of synchronisation protocols:

- FIFO semaphores ;
  - semaphores are used to implement the critical section
  - if the resource is busy, queueing is performed in FIFO order;
  - the task that is using the resource does not adjust its execution priority;
- interrupt masking
  - interrupt masking;
  - disable pre-emption
  - set interrupt level to the maximum level;

Preemptive vs. Non preemptive scheduling

- The best scheduling algorithm maximizes the number of completed tasks
- Tasks are usually treated as preemptive, to guarantee the processing of periodic processes
  - High preemtability minimizes priority inversion
  - There may not be any feasible schedule for non-preemptive schedule
- Scheduling of non-preemptive tasks is less favorable because number of schedulable task sets is smaller compared to preemptive tasks