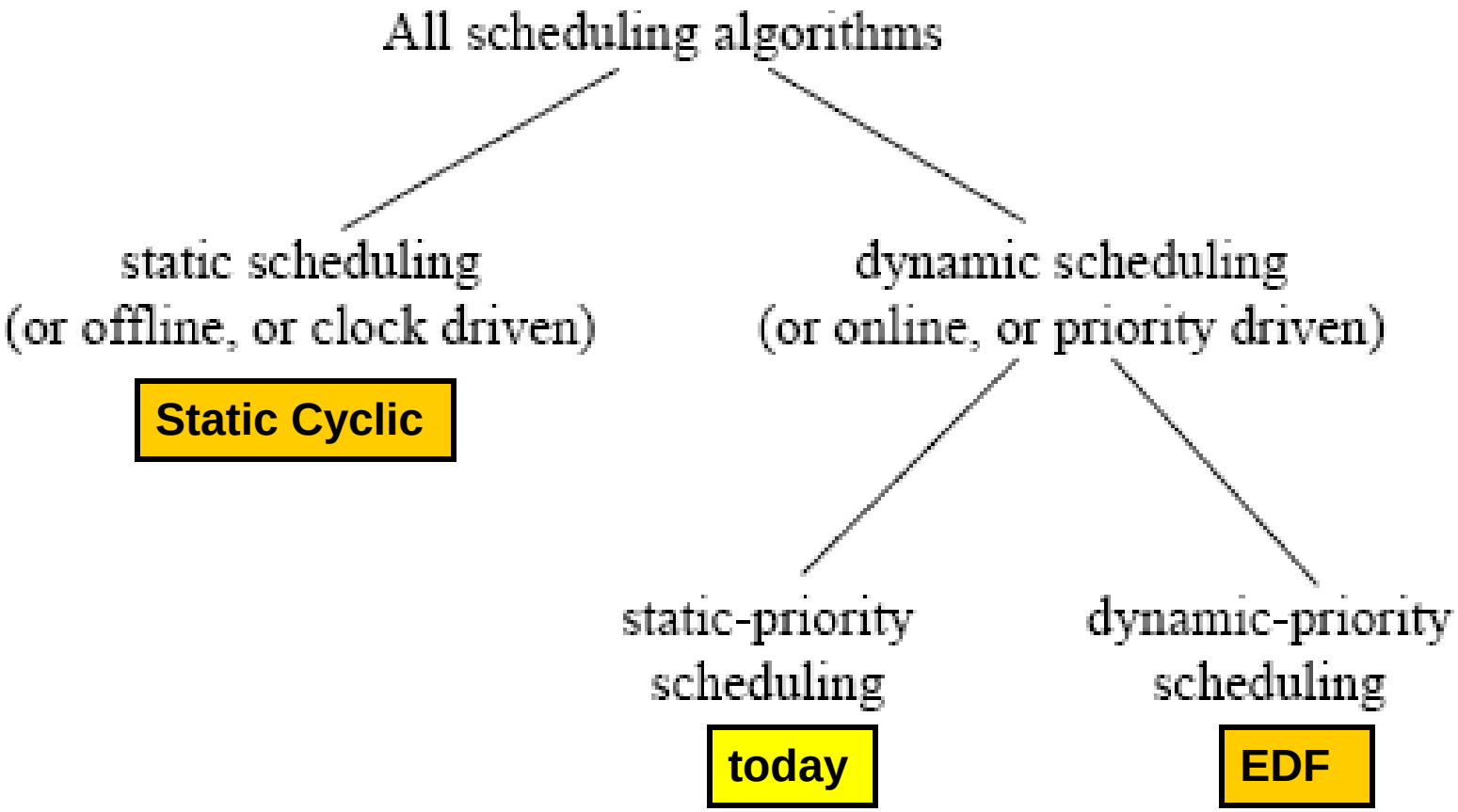
CIS 721 - Real-Time Systems
Lecture 5: Static-Priority Scheduling

Mitch Neilsen
neilsen@ksu.edu

Outline

- Commonly Used Approaches For Real-Time Scheduling (Ch. 4)
  - Clock-Driven Scheduling (Ch. 5)
  - **Priority-Driven Scheduling**
    - **Periodic Tasks (Ch. 6)**
    - Aperiodic or Sporadic Tasks (Ch. 7)

Classification of Scheduling Algorithms



Temporal Parameters

- J_i : **job** – a unit of work
- T_j (or τ_i): **task** - a set of related jobs
- A **periodic task** is sequence of invocations of jobs with identical parameters.
- r_i: **release time** of job J_i
- d_i: **absolute deadline** of job J_i
- D_i: **relative deadline** (or just **deadline**) of job J_i
- e_i: (Maximum) **execution time** of job J_i

Periodic Task Model

- **Tasks:** $T_1, \ldots, T_n$
- Each consists of a set of **jobs**: $T_i = \{J_{i1}, J_{i2}, \ldots\}$
- ϕ_i: p**hase** of task T_i = time when its first job is released
- p_i: p**eriod** of T_i = inter-release time
- H: h**yperperiod** H = lcm($p_1, \ldots, p_n$)
- e_i: e**xecution time** of T_i
- u_i: **utilization** of task T_i is given by $u_i = e_i / p_i$
- D_i: (relative) **deadline** of T_i, typically $D_i = p_i$

Periodic Task

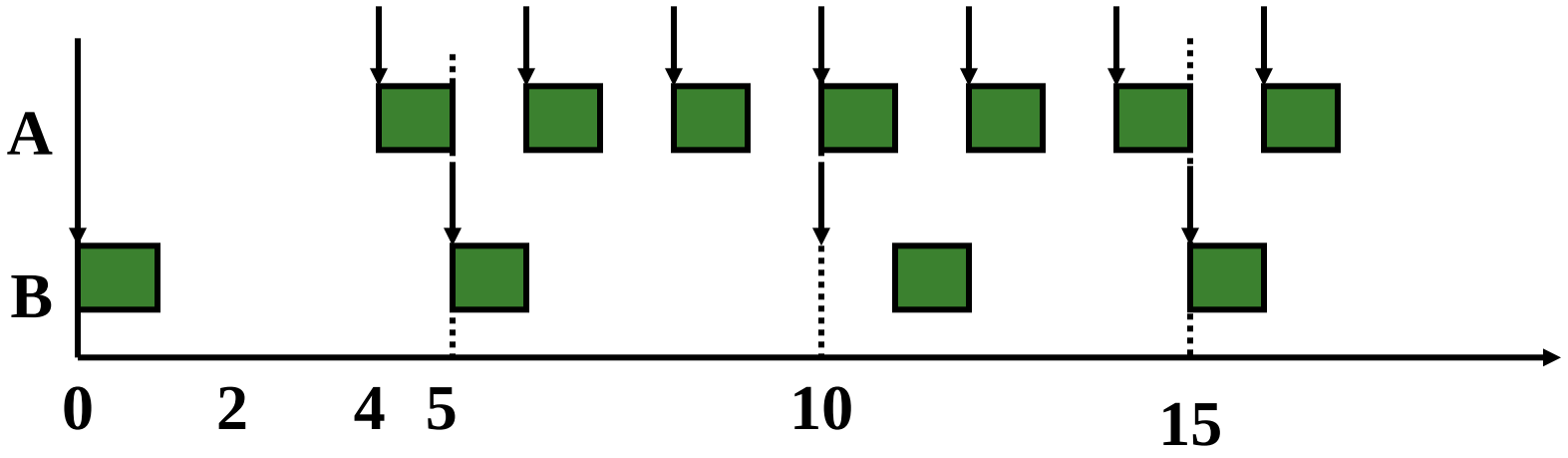
- We refer to a periodic task T_i with phase ϕ_i, period p_i, execution time e_i, and relative deadline D_i by the 4-tuple(ϕ_i, p_i, e_i, D_i).
- Example: (1, 10, 3, 6)
- By default, the phase of each task is 0, and its relative deadline is equal to its period.
- Example: (0, 10, 3, 10) = (10, 3).

Fixed-Priority Scheduling of Periodic Tasks

- We start by considering some examples.
- Then, we consider some methods that can be used to determine the schedulability of a task set:
  - Utilization-based test
  - Time-based test (response time analysis)

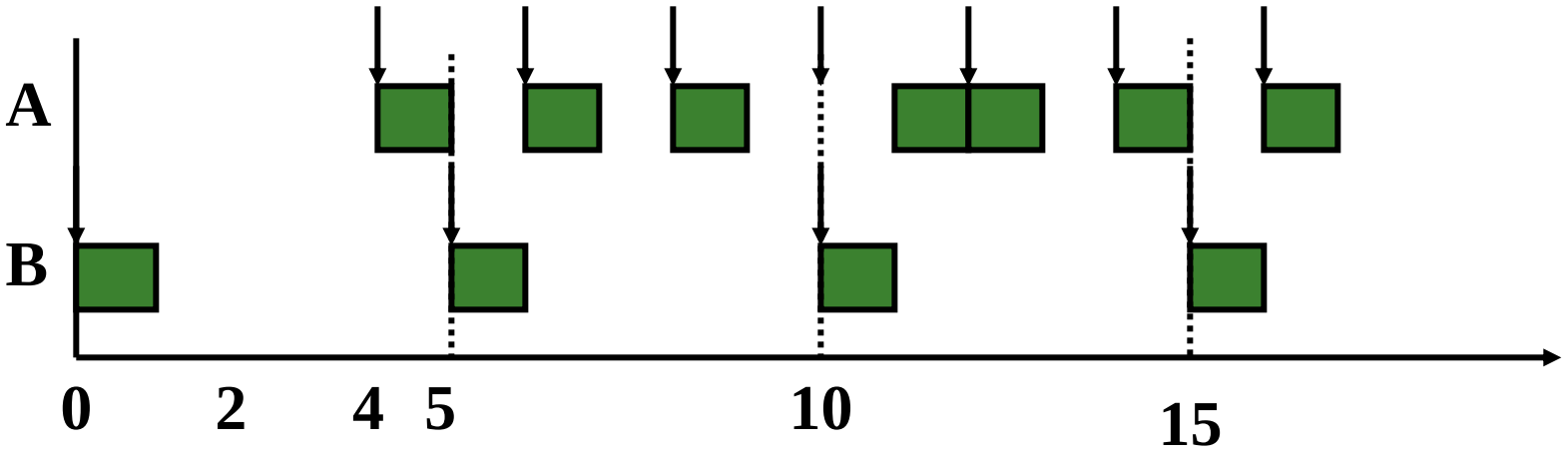
Example #1

Task T_i		Period p_i	Deadline D_i	Run-Time e_i	Phase φ_i
A	(High Priority)	2	2	1	4
B	(Low Priority)	5	5	1	0



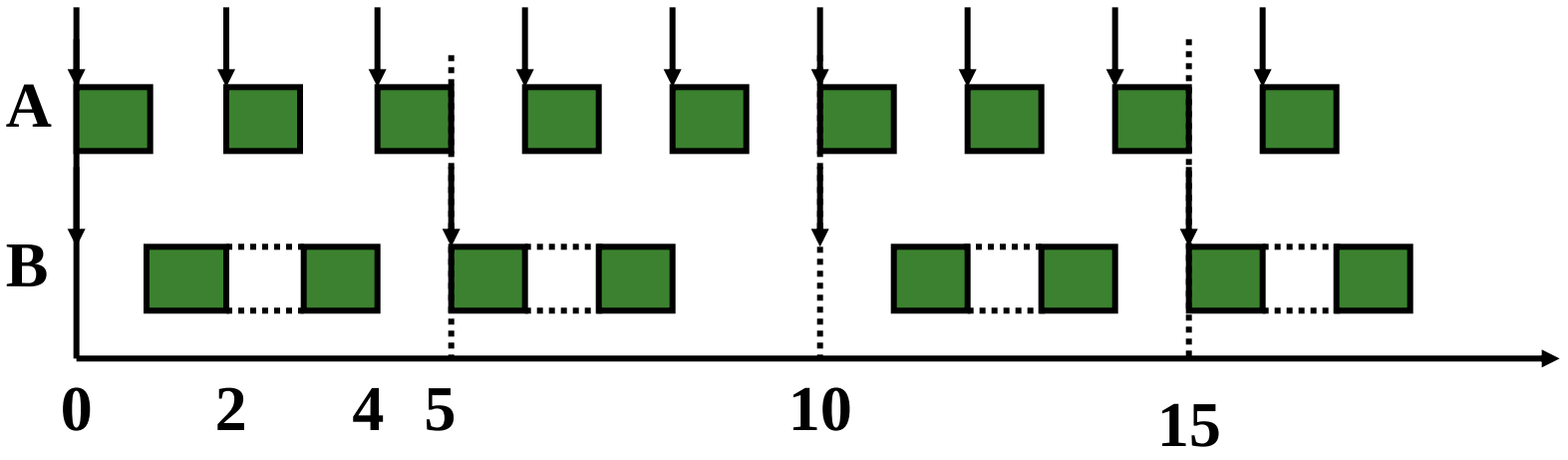
Example #2

Task T_i		Period p_i	Deadline D_i	Run-Time e_i	Phase φ_i
A	(Low Priority)	2	2	1	4
B	(High Priority)	5	5	1	0



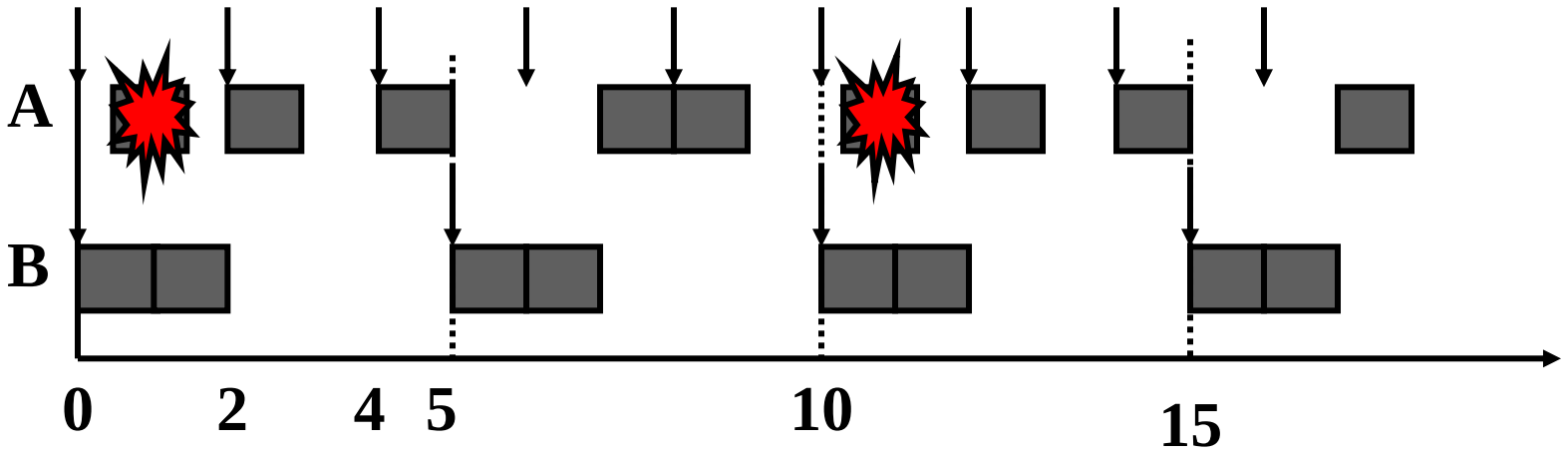
Example #3

Task T_i		Period p_i	Deadline D_i	Run-Time e_i	Phase φ_i
A	(High Priority)	2	2	1	0
B	(Low Priority)	5	5	2	0



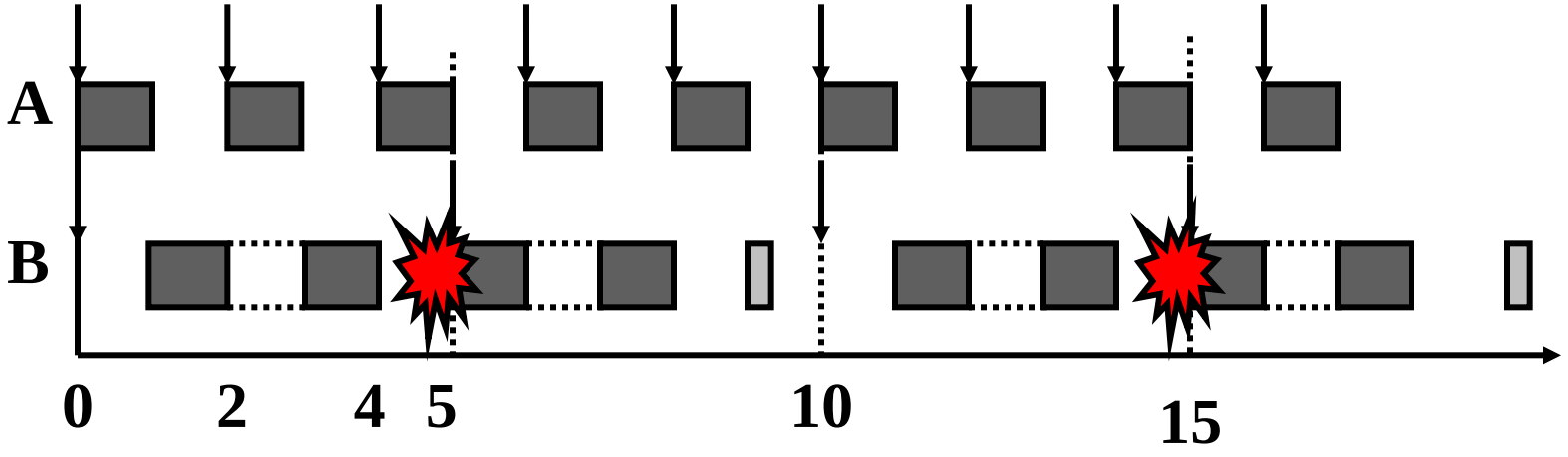
Example #4

Task T_i		Period p_i	Deadline D_i	Run-Time e_i	Phase φ_i
A	(Low Priority)	2	2	1	0
B	(High Priority)	5	5	2	0



Example #5

Task T_i		Period p_i	Deadline D_i	Run-Time e_i	Phase φ_i
A	(High Priority)	2	2	1	0
B	(Low Priority)	5	5	2.1	0



Total task utilization (U) is given by:

$$U = C_1 / T_1 + C_2 / T_2 = 1 / 2 + 2.1 / 5 = 0.92$$

Observations

- Changing the static priorities assigned to each task can impact the task sets feasibility; e.g., Examples 3 and 4.
- Even if the total task utilization is less than 1.0, the task set may not have a feasible priority assignment; e.g., Example 5.
- Is there an upper bound on processor utilization that ensures schedulability?
- Does this bound depend on the scheduling algorithm used?

Issues in Fixed Priority Assignment

- How to assign priorities?
- How to determine which assignment is the best; e.g., how to evaluate a priority assignment algorithm (method)?
- How to compare different priority assignment algorithms?

Rate-Monotonic Algorithm (RM)

- The **rate** of a task is the inverse of its period ($f_i = 1 / p_i$).
- Tasks with **higher rates (shorter periods)** are assigned **higher priorities**.
- C. L. Liu and J. W. Layland, “Scheduling Algorithms for Multiprogramming in a Hard Real-Time Environment”, JACM, Vol. 20, No. 1, pages 46-61, 1973.

Deadline-Monotonic Algorithm (DM)

- Tasks with **shorter relative deadlines** are assigned **higher priorities**.
- When tasks have relative deadlines (D_i) equal to their periods (p_i), the rate-monotonic algorithm is the same as the deadline-monotonic algorithm.

Rate-Monotonic Assumptions

- Tasks may be preempted.
- Tasks are periodic.
- All tasks have phase 0.
- Tasks execution times (e_i) are constant.
- All jobs in a task are assigned the same fixed priority.

Optimal Priority Assignment

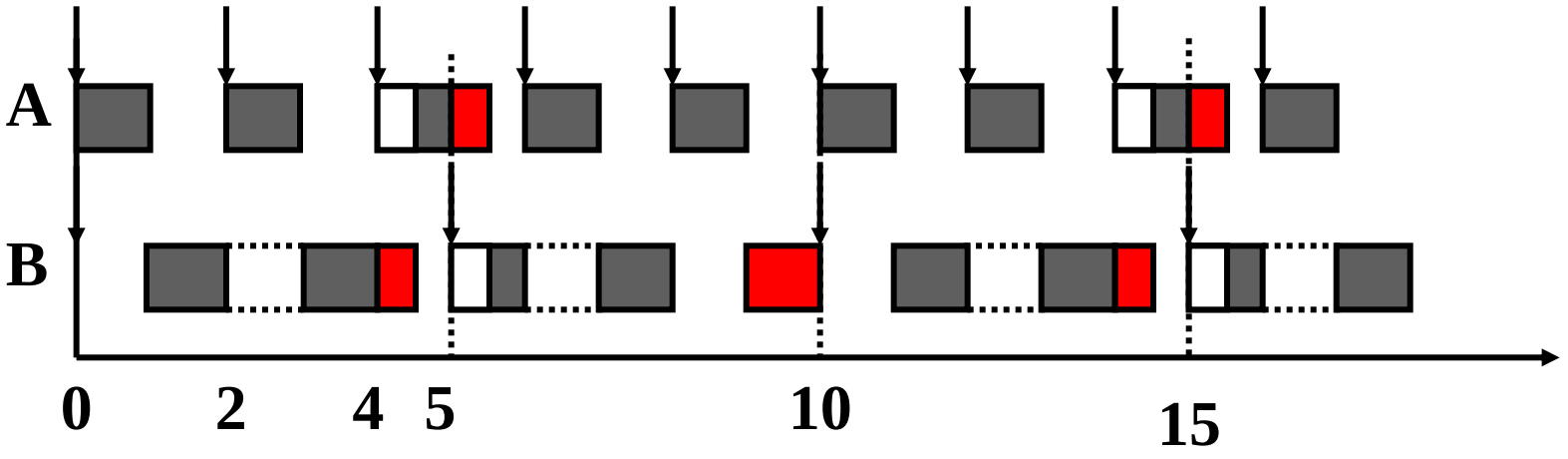
- A given **priority assignment algorithm** is **optimal** if whenever a task set can be scheduled by some **fixed priority assignment**, it can also be scheduled by the given algorithm.
- Liu and Layland show that the rate-monotonic (RM) algorithm is optimal with the given assumptions.

RM and DM are not optimal

- However, note that RM and DM are not optimal among **all** priority-based scheduling algorithms (fixed and dynamic priority).
- Consider the task set consisting of $T_1 = (2,1)$ and $T_2 = (5,2.5)$, both RM and DM would assigns jobs in T_1 the highest priority. As we have seen, the resulting schedule is not feasible.
- If we allow dynamic priority assignment (EDF), then the task set is schedulable.

Example (EDF vs RM)

Task T_i	Period p_i	Deadline D_i	Run-Time e_i	Phase φ_i
A	2	2	1	0
B	5	5	2.5	0



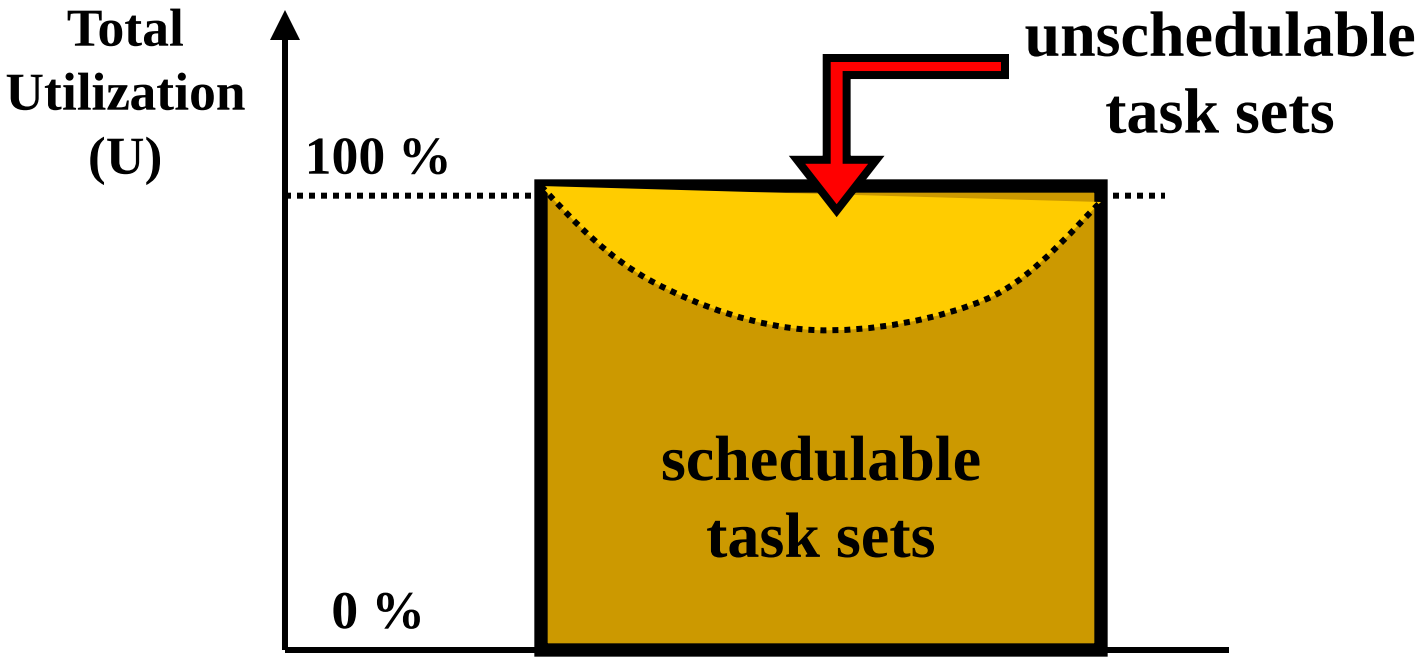
Processor Utilization

- Given a periodic task T_i , the ratio $u_i = e_i / p_i$ is called the **utilization of task T_i** .
- The **total utilization U** of all tasks in a system is the sum of the utilizations of all individual tasks:

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

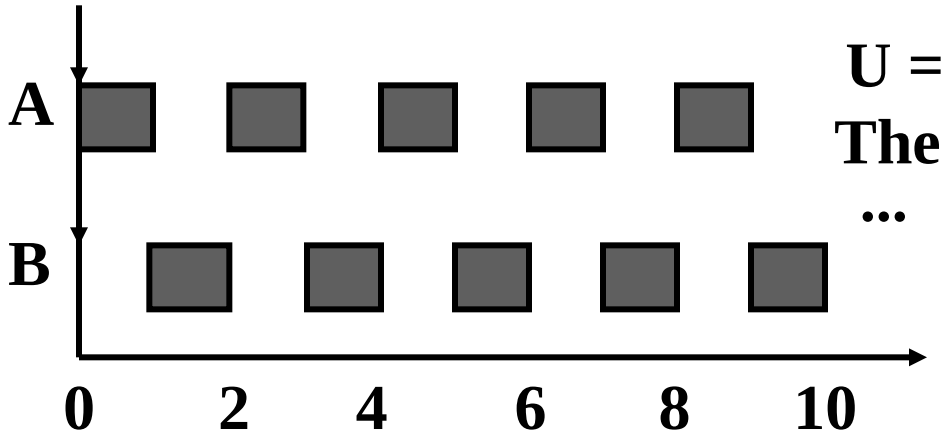
Maximum Achievable Utilization

A task set is **fully utilized** if any increase in run-time would result in a missed deadline.



Example #6

Task T_i		Period p_i	Deadline D_i	Run-Time e_i	Phase φ_i
A	(High Priority)	2	2	1	0
B	(Low Priority)	2	2	1	0



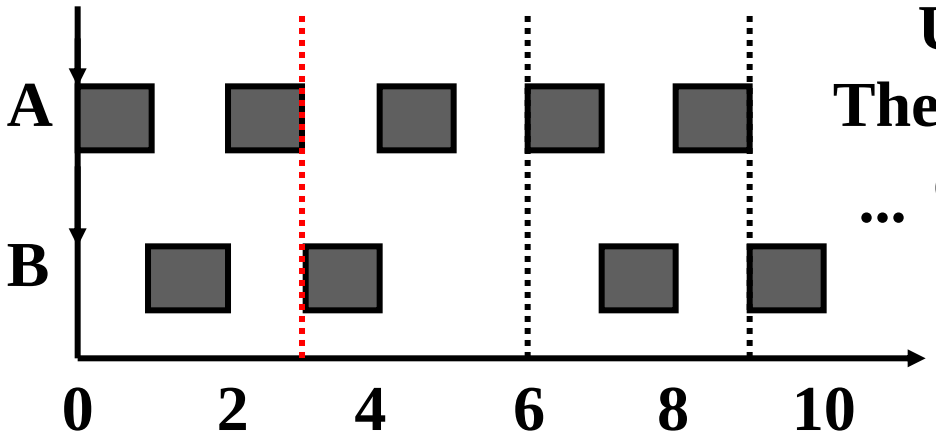
U = 1/2 + 1/2 = 1.0 = 100 %

The task set is fully utilized.

…

Example #7

Task T_i		Period p_i	Deadline D_i	Run-Time e_i	Phase φ_i
A	(High Priority)	2	2	1	0
B	(Low Priority)	3	3	1	0

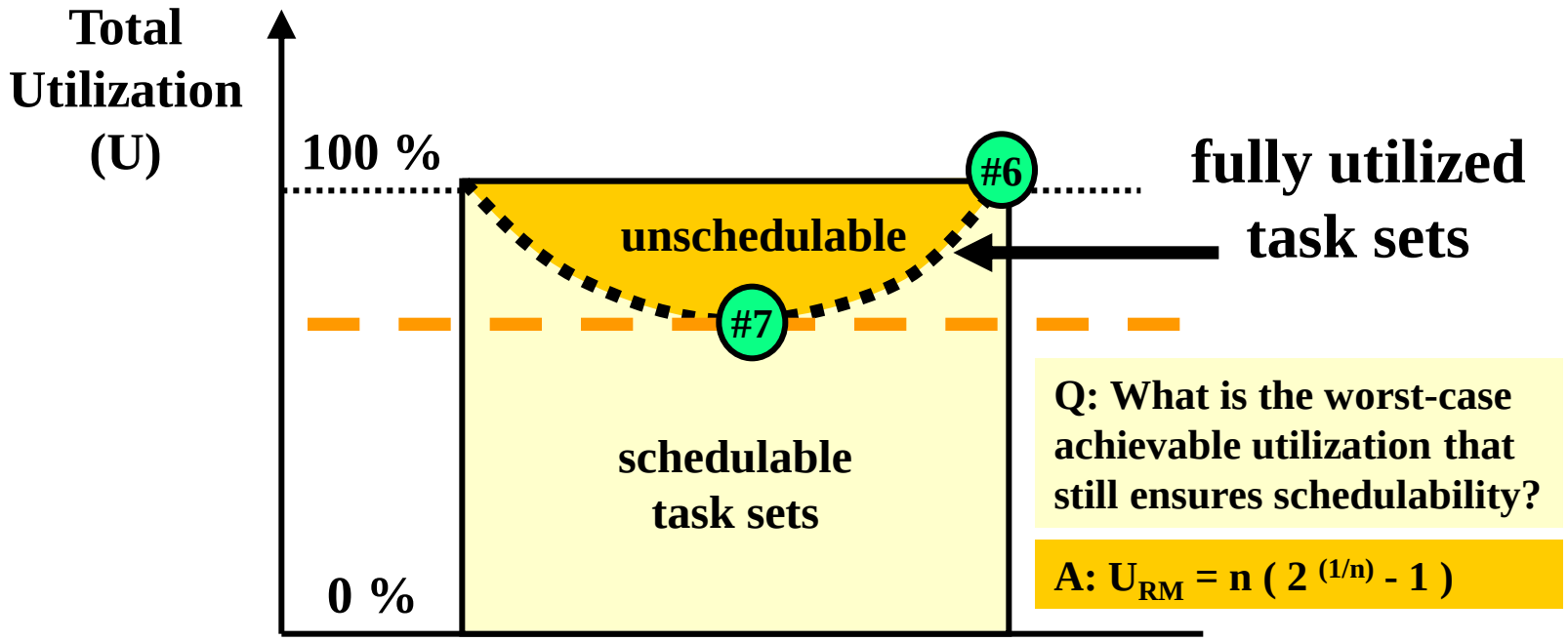


U = 1/2 + 1/3 = 0.8333

The task set is fully utilized, even though U < 1.0.

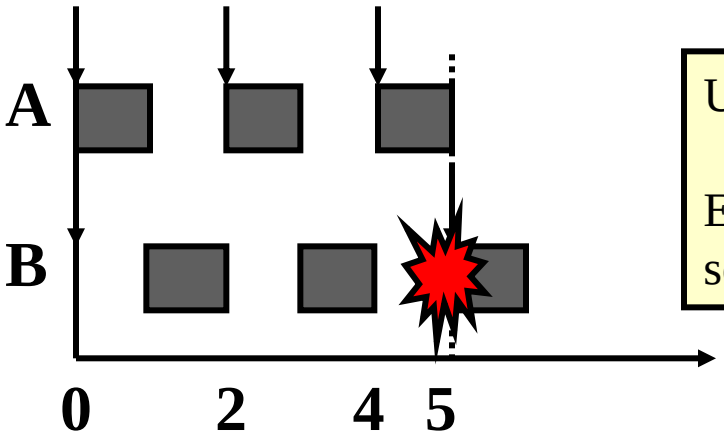
Fully utilized task sets

A task set is **fully utilized** if any increase in run-time would result in a missed deadline.



Example #8

Task T_i		Period p_i	Deadline D_i	Run-Time e_i	Phase φ_i
A	(High Priority)	2	2	1	0
B	(Low Priority)	5	5	2.1	0

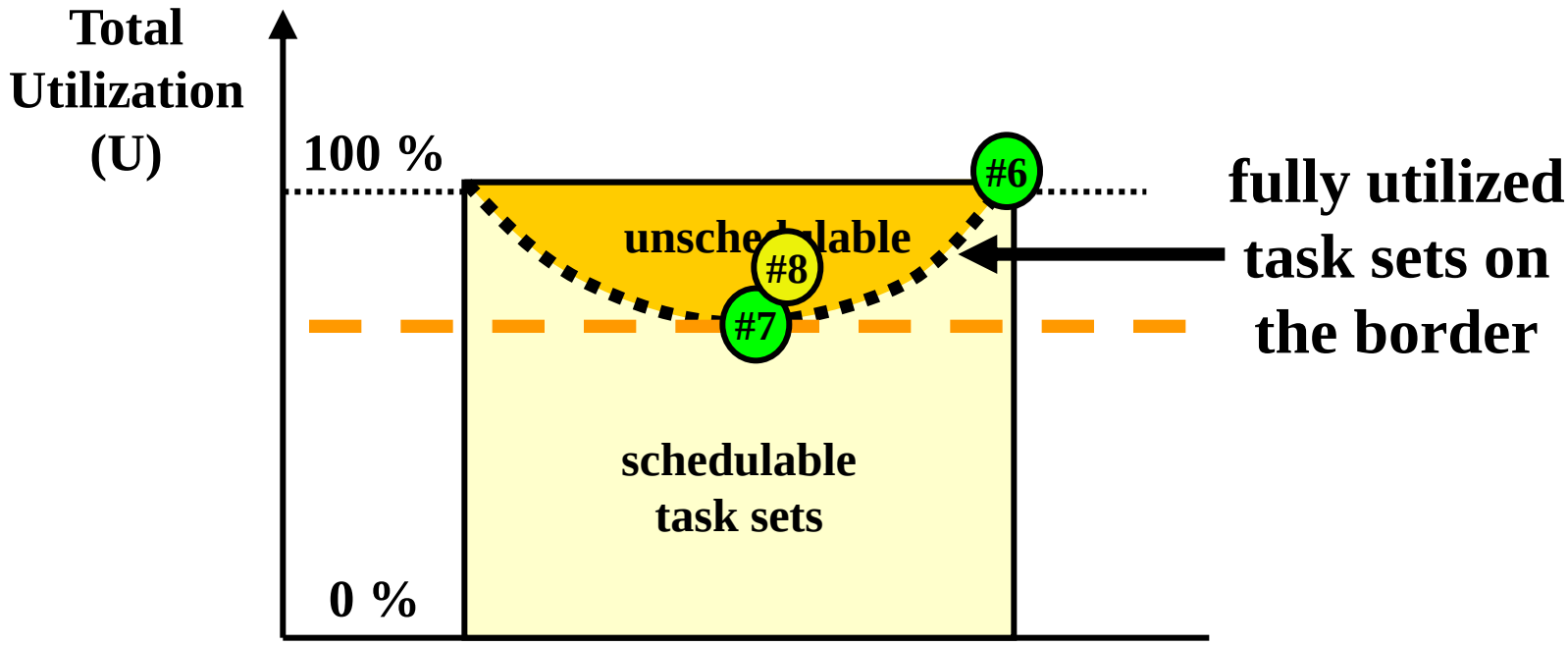


$U = C_1 / T_1 + C_2 / T_2 = 1 / 2 + 2.1 / 5 = 0.92$

Even if $U < 1$, a task set may not be schedulable using fixed priority scheduling.

Fully utilized task sets

A task set is **fully utilized** if any increase in run-time would result in a missed deadline.



Schedulable Utilization

- Every set of periodic tasks with total utilization less than or equal to the **schedulable utilization** of a scheduling algorithm can be feasibly scheduled using that algorithm.
- Schedulable utilization is always less than or equal to 1.0 = 100%.
- In a sense, the higher the schedulable utilization, the better the algorithm.

Examples

- First-In, First-Out (FIFO): $U_{FIFO} = 0$
- Earliest-Deadline-First (EDF): $U_{EDF} = 1$
- Rate-Monotonic Algorithm (RM):

  $U_{RM} = n (2^{(1/n)} - 1)$, where n = number of tasks

Utilization-Based Test

- A **sufficient, but not necessary, test** that can be used to test the schedulability of a task set that is assigned priorities using the given algorithm.
- Compute **total task utilization** U(n) = U.
- Compare with worst-case utilization bound (also called **schedulable utilization**) of a given scheduling algorithm (SA): $U_{SA}(n) = U_{SA}$:
  - If **U > 1**, then the task set is not schedulable.
  - If **$U \leq U_{SA}$**, then the task set is schedulable.
  - Otherwise, no conclusion can be made.

Examples

- **First-In, First-Out (FIFO): $U_{FIFO} = 0$**
- Earliest-Deadline-First (EDF): $U_{EDF} = 1$
- Rate-Monotonic Algorithm (RM):

  $U_{RM} = n (2^{(1/n)} - 1)$, where n = number of tasks

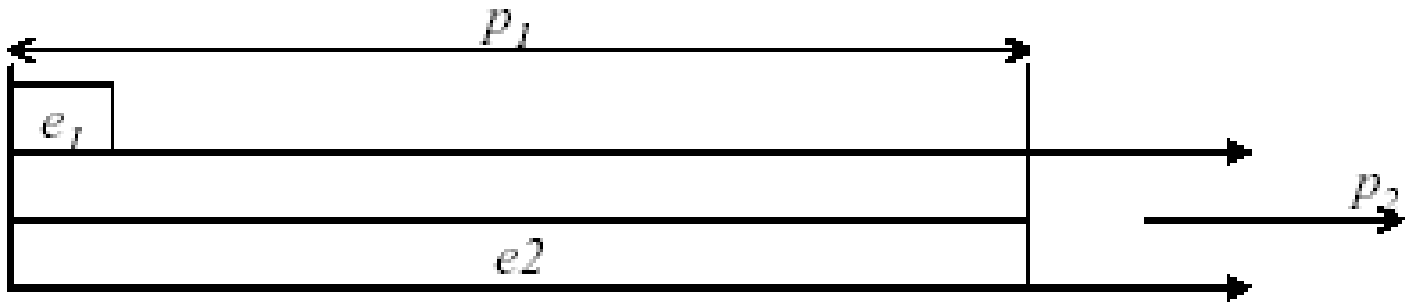
Theorem: $U_{FIFO} = 0$

Proof:

Given any utilization level $\varepsilon > 0$, we can find a task set, with utilization ε, which may not be feasibly scheduled according to FIFO.

Example task set:

$$\left.\begin{array}{lll} T_1 & : & e_1 = \dfrac{\varepsilon}{2} p_1 \\ T_2 & : & p_2 = \dfrac{2}{\varepsilon} p_1 \\ & & e_2 = p_1 \end{array}\right\} \Rightarrow U = \varepsilon$$



Examples

- First-In, First-Out (FIFO): $U_{FIFO} = 0$
- **Earliest-Deadline-First (EDF): $U_{EDF} = 1$**
- Rate-Monotonic Algorithm (RM):

  $U_{RM} = n (2^{(1/n)} - 1)$, where n = number of tasks

Earliest Deadline First (EDF)

Theorem 6-1: [Liu and Layland] A system **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.

Proof: The "only if" part is obvious: If $U > 1$, then some task clearly must miss a deadline. So, we concentrate on the "if" part.

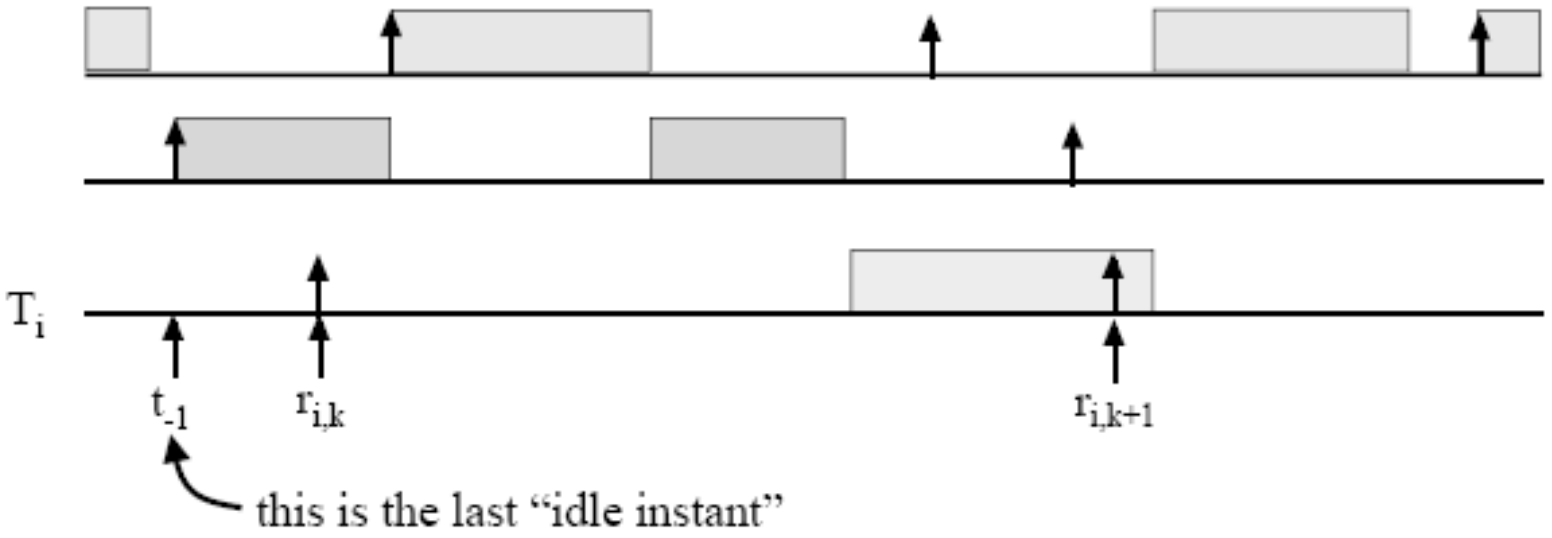
Earliest Deadline First (EDF)

We wish to show: $U \leq 1 \Rightarrow$ **T** is schedulable.

We prove the contrapositive, i.e., T is not schedulable $\Rightarrow U > 1$.

Assume T is not schedulable.

Let $J_{i,k}$ be the first job to miss its deadline.



Earliest Deadline First (EDF)

Because $J_{i,k}$ missed its deadline…

the demand placed on the processor in $[t_{-1}, r_{i,k+1})$ by jobs with deadlines $\leq r_{i,k+1}$ is greater than the available processor time in $[t_{-1}, r_{i,k+1}]$.

Thus,

$$
\begin{aligned}
& r_{i,k+1} - t_{-1} \\
&= \text{available processor time in } [t_{-1}, r_{i,k+1}] \\
&< \text{demand placed on the processor in } [t_{-1}, r_{i,k+1}) \text{ by jobs with deadlines} \leq r_{i,k+1} \\
&= \sum_{j=1}^{N} (\text{the number of jobs of } T_j \text{ with deadlines} \leq r_{i,k+1} \text{ released in } [t_{-1}, r_{i,k+1})) \cdot e_j \\
&\leq \sum_{j=1}^{N} \left\lfloor \frac{r_{i,k+1} - t_{-1}}{p_j} \right\rfloor \cdot e_j \\
&\leq \sum_{j=1}^{N} \frac{r_{i,k+1} - t_{-1}}{p_j} \cdot e_j
\end{aligned}
$$

Earliest Deadline First (EDF)

Thus, we have

$$r_{i,k+1} - t_{-1} < \sum_{j=1}^{N} \frac{r_{i,k+1} - t_{-1}}{p_j} \cdot e_j.$$

Cancelling $r_{i,k+1} - t_{-1}$ yields

$$1 < \sum_{j=1}^{N} \frac{e_j}{p_j},$$

i.e.,

$1 < U$.

This completes the proof.

Note: This proof is actually still valid if **deadlines are larger than periods**.

Assumptions Made

- Tasks may be preempted.
- Tasks are periodic.
- Task execution times (e_i) are constant.
- Task relative deadlines (D_i) equal task periods (p_i).

What if D_i is **less than** p_i for some tasks?

Consider the task set {(2,1,1.9),(2,1,1.9)};

this task set is not feasible even though U <= 1.0.

Sufficient Utilization-Based Test for EDF

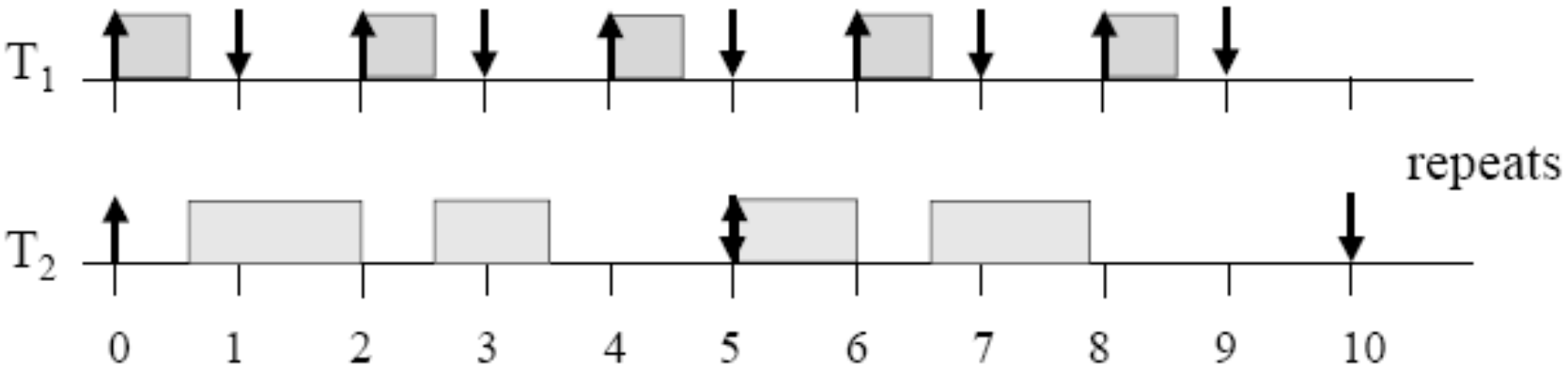
- A Density Test can be used when $D_i < p_i$
- The proof is similar to the previous proof for EDF when all deadlines equal their periods, and is left as a homework exercise ;-).

$$\Delta = \sum_{i=1}^{n} \frac{e_i}{\min\{p_i, D_i\}} \leq 1.0$$

- Density = Δ

Example to show that the Density Test is only sufficient and not necessary

Example: We have two tasks $T_1 = (2, 0.6, 1)$ and $T_2 = (5, 2.3)$. $\Delta = 0.6/1 + 2.3/5 = 1.06$. Nonetheless, we can schedule this task set under EDF:



Improved Utilization-Based Test for EDF

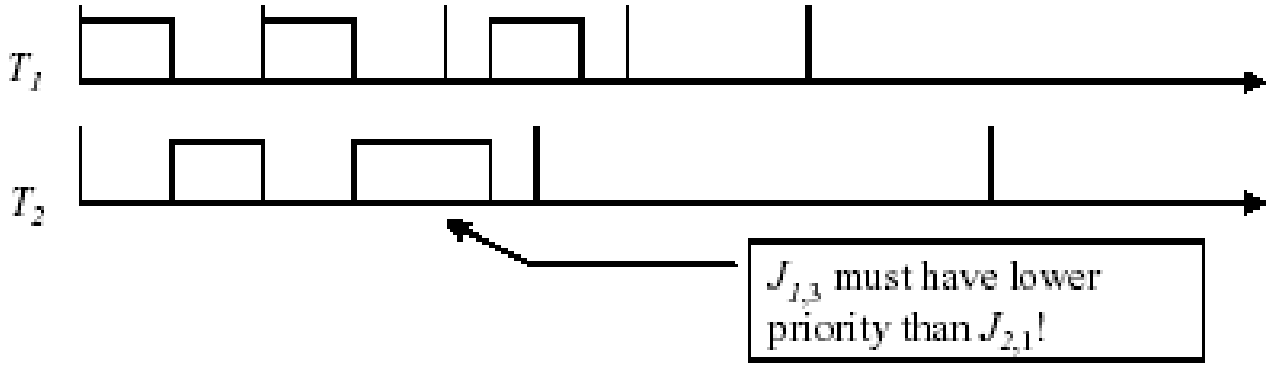
- Devi's Test [2003] assumes that the system of preemptable, asynchronous, periodic tasks with arbitrary relative deadlines are arranged in order of non-decreasing relative deadlines.

$$\sum_{i=1}^{k} \frac{e_i}{p_i} + \frac{1}{D_k} \sum_{i=1}^{k} \left(\frac{p_i - \min\{p_i, D_i\}}{p_i} \right) e_i \leq 1, \forall k$$

What about Static Priority?

- Static-Priority is not optimal!
- Example:

$$\left.\begin{array}{rcl} T_1 & = & (\ 2,\ 1,\ 2\ ) \\ T_2 & = & (\ 5,\ 2.5,\ 5\ ) \end{array}\right\} \qquad U = \frac{e_1}{p_1} + \frac{e_2}{p_2} = 1 \leq 100\%$$



- So: Why bother with static-priority?
  - simplicity
  - predictability

Examples

- First-In, First-Out (FIFO): $U_{FIFO} = 0$
- Earliest-Deadline-First (EDF): $U_{EDF} = 1$
- **Rate-Monotonic Algorithm (RM):**

$$\mathbf{U_{RM} = n (2^{(1/n)} - 1),}$$

where n = number of tasks

- **Special Case: RM + Simply Periodic**

$$\mathbf{U_{RM+SimplyPeriodic} = 1}$$

Simply Periodic Task Sets

Definition: A system of periodic tasks is **simply periodic** if for every pair of tasks T_i and T_k in the system where $p_i < p_k$, p_k is an integer multiple of p_i.

Theorem 6-3: A system **T** of simply periodic, independent, preemptable tasks, whose relative deadlines are at least their periods, is schedulable on one processor according to the RM algorithm if and only if its total utilization is at most one.

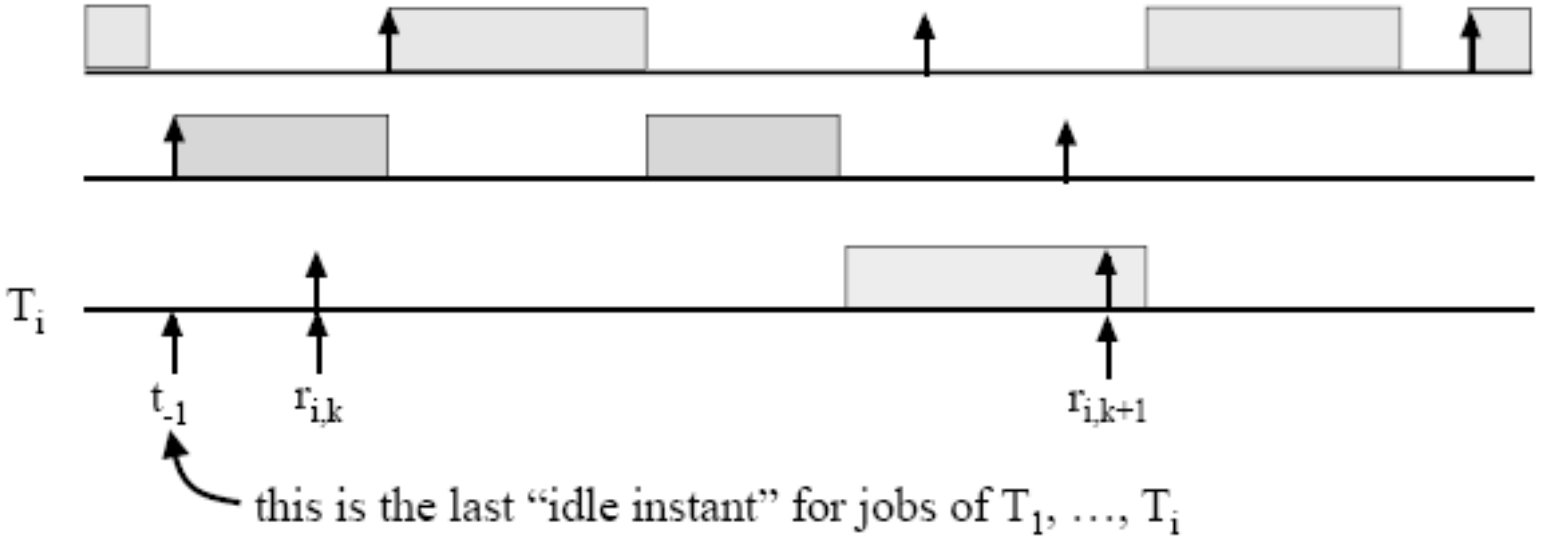
Simply Periodic Task Set Proof

We wish to show: $U \leq 1 \Rightarrow$ **T** is schedulable.

We prove the contrapositive, i.e., **T** is not schedulable $\Rightarrow U > 1$.

Assume **T** is not schedulable.

Let $J_{i,k}$ be the first job to miss its deadline.



Simply Periodic Task Set Proof

Because $J_{i,k}$ missed its deadline…

> the demand placed on the processor in $[t_{-1}, r_{i,k+1})$ by jobs of tasks $T_1, \ldots, T_i$ is greater than the available processor time in $[t_{-1}, r_{i,k+1}]$.

Thus,

$$
\begin{aligned}
& r_{i,k+1} - t_{-1} \\
&= \text{available processor time in } [t_{-1}, r_{i,k+1}] \\
&< \text{demand placed on the processor in } [t_{-1}, r_{i,k+1}) \text{ by jobs of } T_1, \ldots, T_i \\
&= \sum_{j=1}^{i} (\text{the number of jobs of } T_j \text{ released in } [t_{-1}, r_{i,k+1})) \cdot e_j \\
&\leq \sum_{j=1}^{i} \frac{r_{i,k+1} - t_{-1}}{p_j} \cdot e_j
\end{aligned}
$$

[Note : Because the system is simply periodic, $\frac{r_{i,k+1} - t_{-1}}{p_j}$ is an integer.]

Simply Periodic Task Set Proof

Thus, we have

$$r_{i,k+1} - t_{-1} < \sum_{j=1}^{i} \frac{r_{i,k+1} - t_{-1}}{p_j} \cdot e_j.$$

Cancelling $r_{i,k+1} - t_{-1}$ yields

$$1 < \sum_{j=1}^{i} \frac{e_j}{p_j},$$

i.e.,

$$1 < U_i \leq U.$$

This completes the proof.

Liu and Layland's Results

- **Theorem 1:** A critical instant for any task occurs when the task is requested simultaneously with requests for all higher priority tasks.
- **Theorem 2:** If a feasible priority assignment exists for some task set, then the rate-monotonic priority assignment is feasible for that task set.
- **Theorem 3:** For a set of two tasks with fixed priority assignment, the least upper bound to the processor utilization factor is $U_{RM} = 2 (2^{1/2} - 1)$.

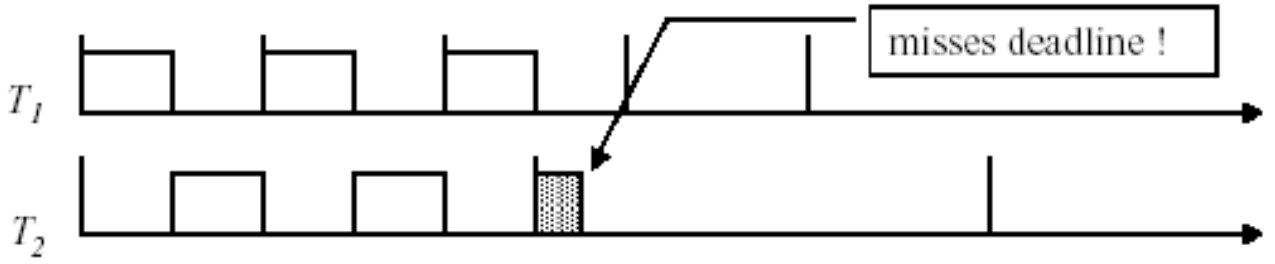
Schedulable Utilization of Tasks with $D_i=p_i$, Using Rate-Monotonic Algorithm

- Theorem:

> [Liu&Layland '73] A system of n independend, preemptable periodic tasks with $D_i=p_i$ can be feasibly scheduled by the RM algorithm if its total utilization U is less or equal to
>
> $$U_{RM}(n) = n(2^{1/n} - 1)$$

- Why not 1.0?

$$T_1 = (2, 1, 2)$$
$$T_2 = (5, 2.5, 5)$$



- Proof:

First, show that theorem is correct for special case where longest period $p_n<2p_1$ (p_1 = shortest period).
Will remove this restriction later.

Summary

- Read Ch. 4-7 + Liu and Layland's paper.
- Homework #1