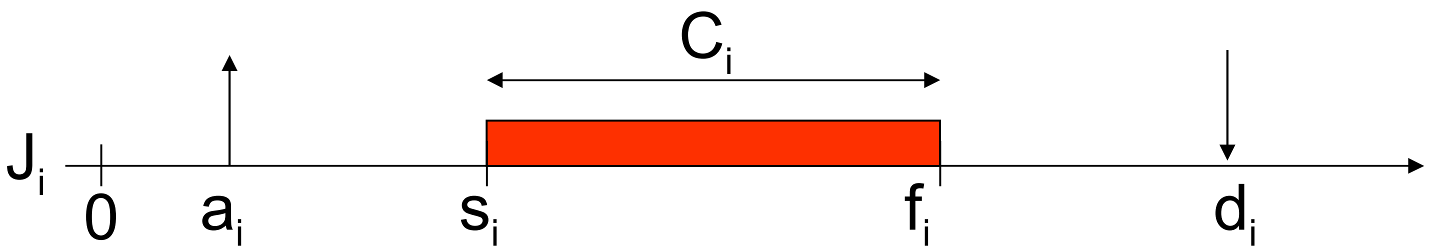
Embedded Systems 15

A-periodic scheduling

REVIEW



- Given:
  - A set of non-periodic tasks $\{J_1, \ldots, J_n\}$ with
    - arrival times a_i, deadlines d_i, computation times C_i
    - precedence constraints
    - resource constraints
  - ~~Class of s~~cheduling algorithm:
    - Preemptive, non-preemptive
    - Off-line / on-line
    - Optimal / heuristic
    - One processor / multi-processor
    - …
  - Cost function:
    - ~~Minimize maximum lateness (s~~oft RT)
    - Minimize maximum number of late tasks (feasibility! – hard RT)
- Find:
  Optimal / good schedule according to given cost function

Case 1: Aperiodic tasks with synchronous release

REVIEW

- A set of (a-periodic) tasks $\{J_1, \ldots, J_n\}$ with
  - arrival times $a_i = 0 \; \forall \; 1 \leq i \leq n$, i.e. “synchronous” arrival times
  - deadlines d_i,
  - computation times C_i
  - no precedence constraints, no resource constraints, i.e. “independent tasks”
- non-preemptive
- single processor
- Optimal
- Find schedule which minimizes maximum lateness (variant: find feasible solution)

EDD – Earliest Due Date

REVIEW

EDD: execute the tasks in order of non-decreasing deadlines

- **Lemma**:
  If arrival times are synchronous, then preemption does not help, i.e. if there is a preemptive schedule with maximum lateness L_{max}, then there is also a non-preemptive schedule with maximum lateness L_{max}.

- **Theorem (Jackson '55):**
  Given a set of n independent tasks with synchronous arrival times, any algorithm that executes the tasks in order of non-decreasing deadlines is optimal with respect to minimizing the maximum lateness.

Case 2: aperiodic tasks with asynchronous release

REVIEW

- A set of (a-periodic) tasks $\{J_1, \ldots, J_n\}$ with
  - **arbitrary** arrival times a_i
  - deadlines d_i,
  - computation times C_i
  - no precedence constraints, no resource constraints, i.e. "independent tasks"
- ~~**preemptive**~~
- Single processor
- Optimal
- Find schedule which ~~minimizes maximum lateness~~ (variant: find feasible solution)

EDF – Earliest Deadline First

- EDF: At every instant execute the task with the earliest absolute deadline among all the ready tasks.

- **Theorem (Horn '74):**
  Given a set of n independent task with arbitrary arrival times, any algorithm that at every instant executes the task with the earliest absolute deadline among all the ready tasks is optimal with respect to minimizing the maximum lateness.

Non-preemptive version

REVIEW

- Changed problem:
  - A set of (a-periodic) tasks $\{J_1, \ldots, J_n\}$ with
    - **arbitrary** arrival times a_i
    - deadlines d_i,
    - computation times C_i
    - no precedence constraints, no resource constraints, i.e. "independent tasks"
  - **~~Non-pr~~eemptive** instead of preemptive scheduling!
  - Single processor
  - Optimal
  - Find schedule which minimizes maximum lateness (variant: find feasible solution)

Non-preemptive version

REVIEW

- **Theorem** (Jeffay et al. ’91): EDF is an optimal *non-idle* scheduling algorithm also in a non-preemptive task model.

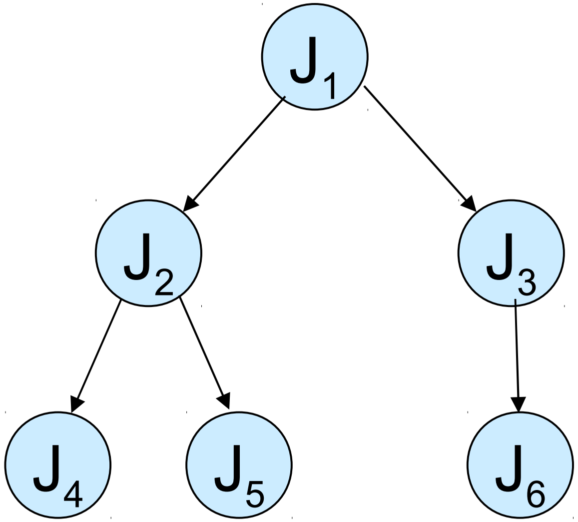
- When idle schedules are allowed: problem is NP-hard.
- Possible approaches:
  - Heuristics
  - Bratley’s algorithm: branch-and-bound

Case 3: Scheduling with precedence constraints

- Non-preemptive scheduling with non-synchronous arrival times, deadlines and precedence constraints is NP-hard.
- Here:
  - Restrictions:
    - Consider synchronous arrival times (all tasks arrive at 0)
    - Allow preemption.
  - 2 different algorithms:
    - Latest deadline “first” (LDF)
    - Modified EDF
- Precedences define a partial order, represented as a DAG
- Scheduling determines a compatible total order
- Method: Topological sorting

Example

	J_1	J_2	J_3	J_4	J_5	J_6
a_i	0	0	0	0	0	0
C_i	1	1	1	1	1	1
d_i	2	5	4	3	5	6



Example

- One of the following algorithms is optimal. Which one?

task list

Algorithm 1:	Algorithm 2:
1. Among all **sources** in the precedence graph select the task T with **earliest** deadline. Schedule T **first**.	1. Among all **sinks** in the precedence graph select the task T with **latest** deadline. Schedule T **last**.
2. Remove T from G.	2. Remove T from G.
3. Repeat.	3. Repeat.

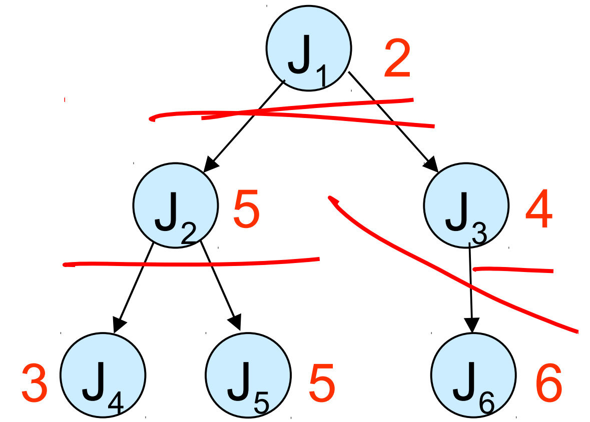
Forward topological sorting

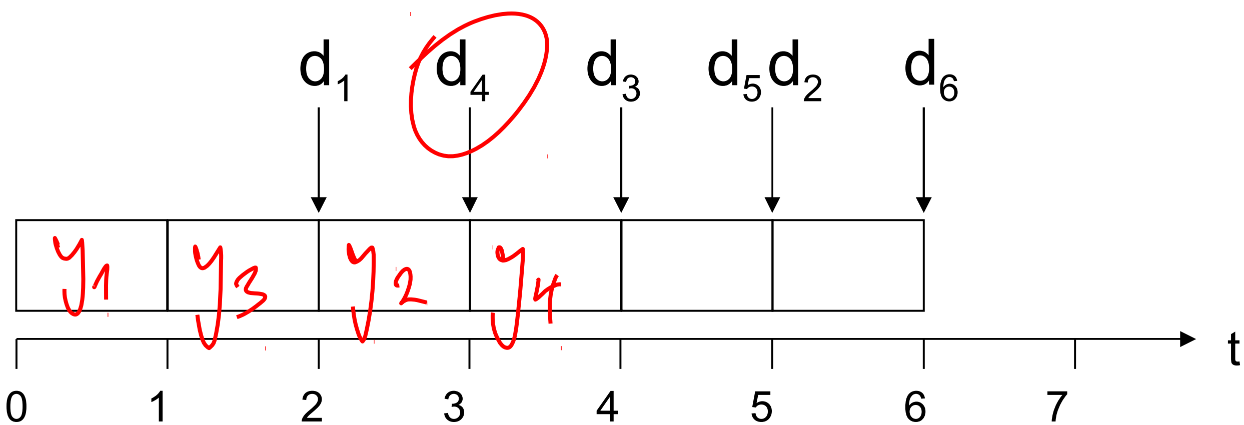
Backward topological sorting

Example (continued)

- Algorithm 1:

	J_1	J_2	J_3	J_4	J_5	J_6
a_i	0	0	0	0	0	0
C_i	1	1	1	1	1	1
d_i	2	5	4	3	5	6



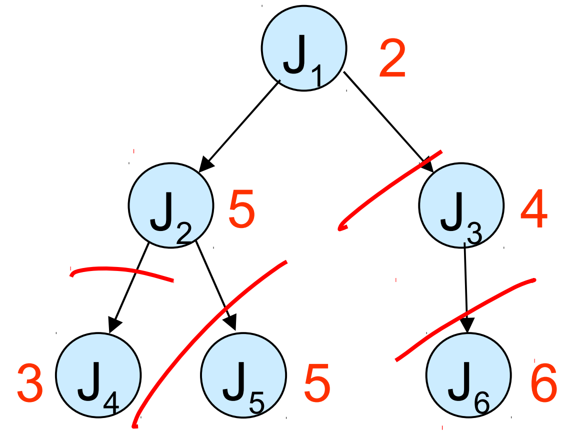


J4 missed the deadline

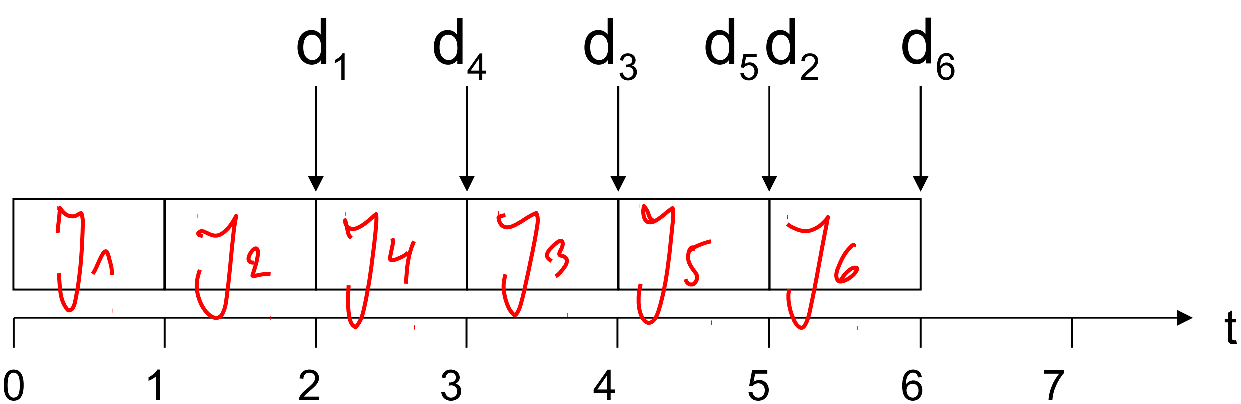
Example (continued)

- Algorithm 2:

	J_1	J_2	J_3	J_4	J_5	J_6
a_i	0	0	0	0	0	0
C_i	1	1	1	1	1	1
d_i	2	5	4	3	5	6



J_1 J_2 J_4 J_3 J_5 J_6



feasible

Example (continued)

- Algorithm 1 is **not** optimal.
- Algorithm 1 is the generalization of EDF to the case with precedence constraints.

- Is Algorithm 2 optimal?
- Algorithm 2 is called Latest Deadline First (LDF).

- **Theorem (Lawler 73):**
  LDF is optimal wrt. maximum lateness.

Proof of optimality

Task set $\mathcal{J} = \{J_1, \dots, J_n\}$, $\Gamma \subseteq \mathcal{J}$ subset without successors,

Let $J_l \in \Gamma$ with the latest deadline.

Consider a schedule σ satisfying constraints

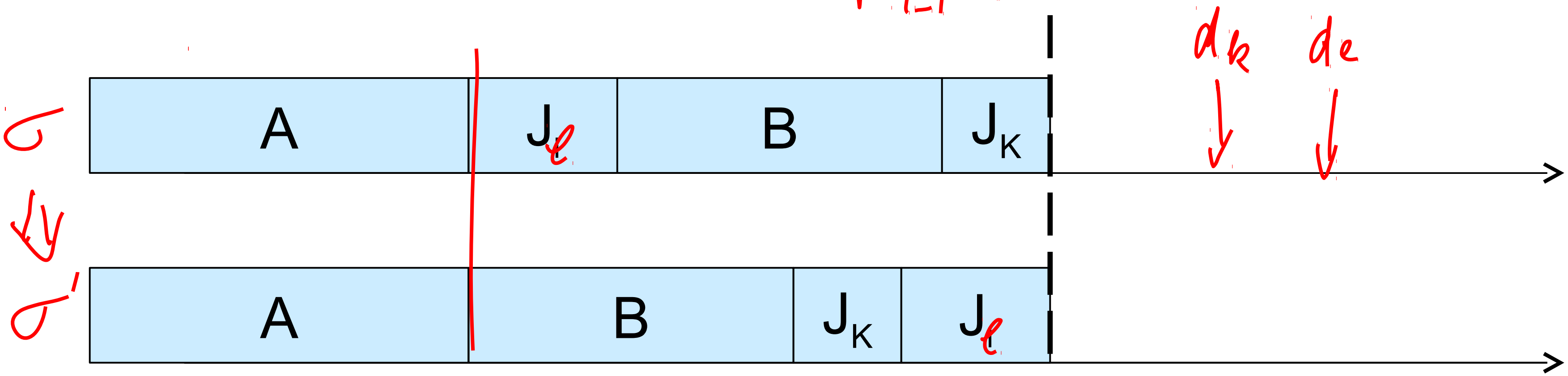
where the last scheduled task is J_k, $k \neq l$, $d_k < d_l$.

It is clear that $J_k \in \Gamma$.

We show that we can move J_l to the end of the schedule with

1) no violation of the precedence
2) no increase in max. lateness.

$f = \sum_{i=1}^{n} C_i$



1. precedence not violated: J_l has no successors in Γ
2. $L'_{max} = \max\{L'_{max}(A), L'_{max}(B), L'_k, L'_l\}$

$L'_{max}(A) = L_{max}(A)$ nothing changed

$L'_{max}(B) < L_{max}(B)$ starts and ends earlier

$L'_k < L_k$ starts and ends earlier

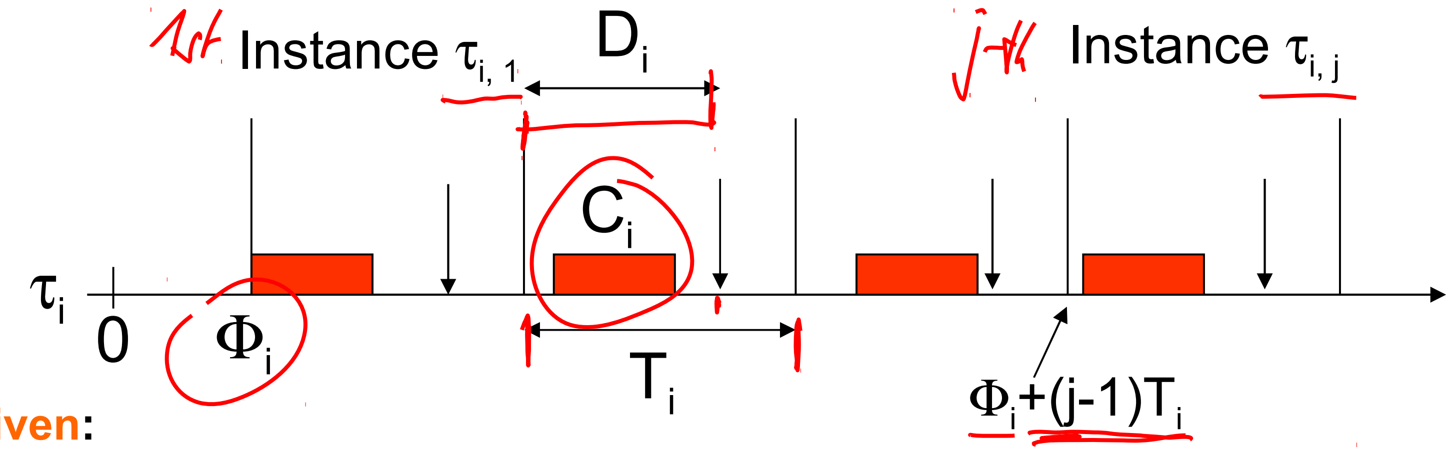
$L'_l = \sum_{i=1}^{n} C_i - d_l < \sum_{i=1}^{n} C_i - d_k$

$\Rightarrow L'_{max} \leq L_{max}$

Remove J_l from Γ and continue

Optimal scheduling algorithms for *periodic* tasks

Periodic scheduling



- **Given**:
  - A set of periodic tasks $\Gamma = \{\tau_1, \ldots, \tau_n\}$ with
    - phases Φ_i (arrival times of first instances of tasks),
    - periods T_i (time difference between two consecutive activations)
    - relative deadlines D_i (deadline relative to arrival times of instances)
    - computation times C_i

  $\Rightarrow$ *j* th instance $\tau_{i,j}$ of task τ_i with
    - arrival time $a_{i,j} = \Phi_i + (j\text{-}1)\,T_i$,
    - deadline $d_{i,j} = \Phi_i + (j\text{-}1)\,T_i + D_i$,
- **Find a feasible schedule**
    - start time $s_{i,j}$ and
    - finishing time $f_{i,j}$

Assumptions

A.1. Instances of periodic task τ_i are regularly activated with constant period T_i.

A.2. All instances have same worst case execution time C_i.

A.3. All instances have same relative deadline D_i, here in most cases equal to T_i (i.e., $d_{i,j} = \Phi_i + j \cdot T_i$)

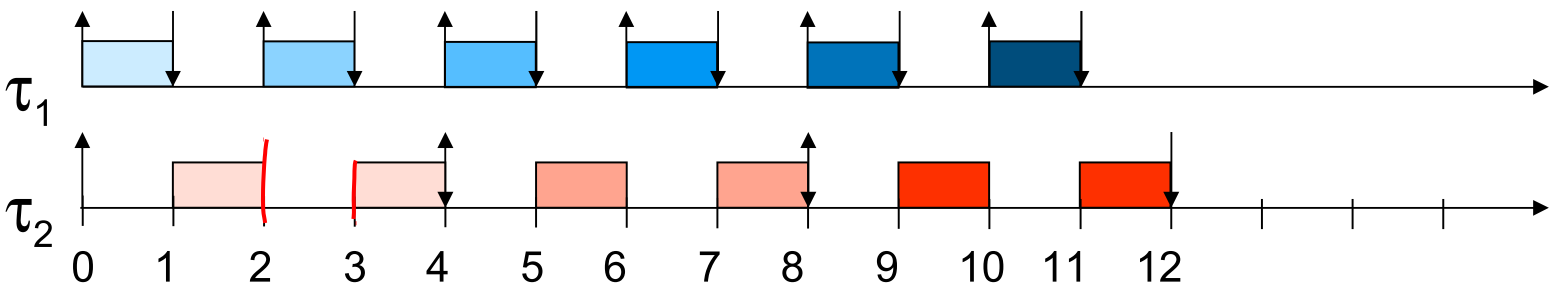
A.4. All tasks in Γ are independent. No precedence relation, no resource constraints.

A.5. Overhead for context switches is neglected, i.e. assumed to be 0 in the theory.

- Basic results based on these assumptions form the core of scheduling theory.
- For practical applications, assumptions A.3. and A.4. can be relaxed, but results have to be extended.

Examples for periodic scheduling (1)

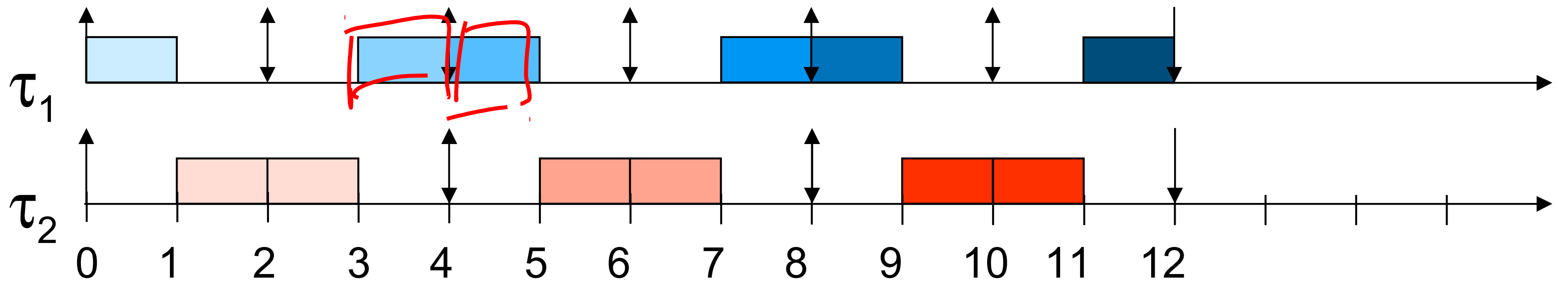
	τ_1	τ_2
Φ_i	0	0
T_i	2	4
C_i	1	2
D_i	1	4



- Schedulable, but only preemptive schedule possible.

Examples for periodic scheduling (2)

	τ_1	τ_2
Φ_i	0	0
T_i	2	4
C_i	1	2
D_i	~~2~~	~~4~~



- Schedulable with non-preemptive schedule.

Examples for periodic scheduling (3)

	τ_1	τ_2
Φ_i	0	0
T_i	3	4
C_i	2	2
D_i	3	4

$T_1 \cdot T_2 = 12$

Number of executions of τ_1 and τ_2 within 12 time units:

$\frac{12}{3} = 4$ executions of τ_1

- No feasible schedule for single processor. $\frac{12}{4} = 3$ " of τ_2

$4 \cdot 2 = 8$ time units by τ_1
$3 \cdot 2 = 6$ " " " τ_2

Within 12 time units ~~not~~ possible to exec. all the instances of τ_1, τ_2

$T = T_1 \cdot \ldots \cdot T_n$ $\frac{T}{T_i}$ = #activations of τ_i in T / $\frac{T}{T_i} \cdot C_i$ time taken by τ_i.

$\sum_{i=1}^{n} \frac{T}{T_i} \cdot C_i$ total time.

Processor utilization

Definition:

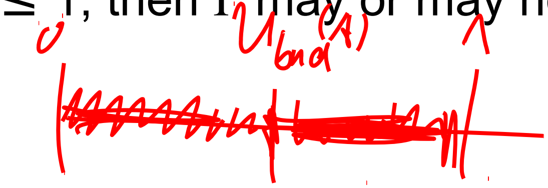
Given a set Γ of n periodic tasks, the **processor utilization U** is given by

$$U = \sum_{i=1}^{n} \frac{C_i}{T_i}.$$

$$\frac{C_i}{T_i}$$

Processor utilization: using it as a schedulability criterion

- Given: a scheduling algorithm A
- Define $U_{bnd}(A) = \inf \{U(\Gamma) \mid \Gamma$ is not schedulable by algorithm A$\}$.
- If $U_{bnd}(A) > 0$ then a simple, sufficient criterion for schedulability by A can be based on processor utilization:
  - If $U(\Gamma) < U_{bnd}(A)$ then Γ is schedulable by A.
  - However, if $U_{bnd}(A) < U(\Gamma) \leq 1$, then Γ may or may not be schedulable by A.



- **Question**:
  Does a scheduling algorithm A exist with $U_{bnd}(A) = 1$?

Processor utilization

- **Question**:
  Does a scheduling algorithm A exist with $U_{bnd}(A) = 1$?
- **Answer:**
  - No, if $D_i < T_i$ allowed.
  - Example:

	τ_1	τ_2
Φ_i	0	0
T_i	2	2
C_i	1	1
D_i	1	1

$\frac{1}{2} + \frac{1}{2} = 1$

  - Yes, if $D_i = T_i$ (or $D_i \geq T_i$) → Earliest Deadline First (EDF)
  - In the following: assume $D_i = T_i$

Earliest Deadline First (EDF)

- EDF is applicable to both periodic and a-periodic tasks.
- If there are only periodic tasks, priority-based schemes like "rate monotonic scheduling (RM)" (see later) are often preferred, since
  - They are simpler due to fixed priorities
    ⇒ use in "standard OS" possible
  - sorting wrt. to deadlines at run time is not needed

RM static
EDF dynamic priorities

EDF and processor utilization factor

- **Theorem:** **A set of periodic tasks $\tau_1, \ldots, \tau_n$ with $D_i = T_i$ is schedulable with EDF iff $U = \sum_{i=1}^{n} C_i / T_i \leq 1$.**

Proof: "$\Rightarrow$" Let $T = T_1 \cdot T_2 \cdot \ldots T_n$

$$\sum_{i=1}^{n} \frac{T}{T_i} \cdot C_i \quad \text{time taken by task set within } T$$

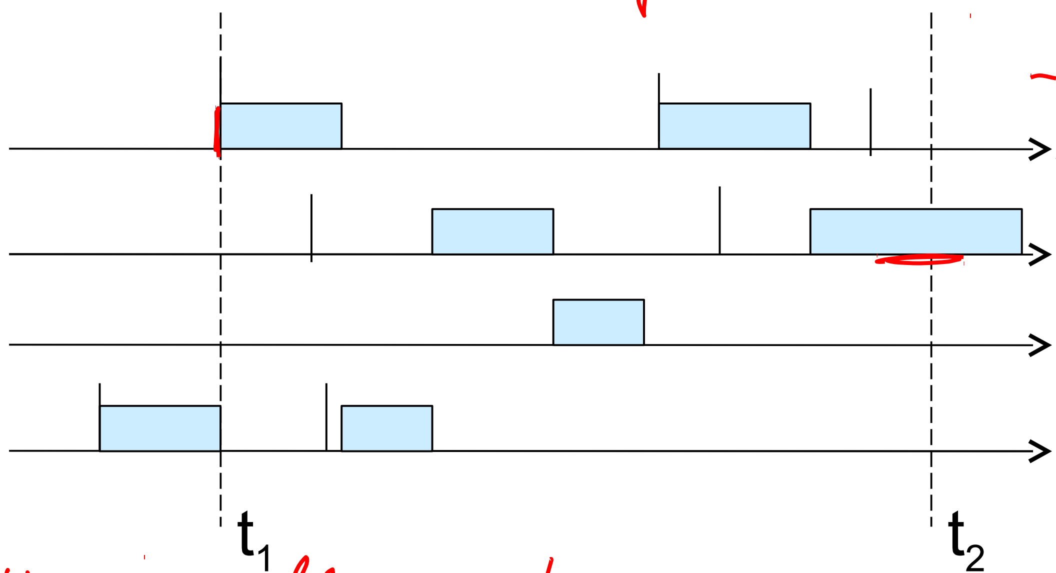
$$= \sum_{i=1}^{n} \frac{C_i}{T_i} \cdot T = U \cdot T$$

Assume $U > 1$. Then $U \cdot T > T$
task set not schedulable.

"$\Leftarrow$": by contradiction: Assume task set not schedulable, but $U \le 1$

time overrun at t_2 in EDF schedule.

Let $[t_1, t_2]$ be the longest interval s.t.



- no idle times
- only instances with deadlines $\le t_2$ executed.

t_1 must be release time of an instance

time overflow at t_2:

$$\Rightarrow \underline{(t_2 - t_1)} < \sum_{a_{ij} \ge t_1,\; d_{ij} \le t_2} C_i = \sum_{i=1}^{n} \left\lfloor \frac{t_2 - t_1}{T_i} \right\rfloor \cdot C_i \le \sum_{i=1}^{n} \frac{t_2 - t_1}{T_i} \cdot C_i$$

$$= (t_2 - t_1) \cdot \sum_{i=1}^{n} \frac{C_i}{T_i} = \underline{(t_2 - t_1) \cdot U} \Rightarrow U > 1 \ ↯$$

Rate monotonic scheduling (RM)

- Rate monotonic scheduling (RM) (Liu, Layland ’73):
  - Assign fixed priorities to tasks τ_i:
    - priority(τ_i) = $1/T_i$
    - I.e., priority reflects release rate
  - Always execute ready task with highest priority
  - Preemptive: currently executing task is preempted by newly arrived task with shorter period.

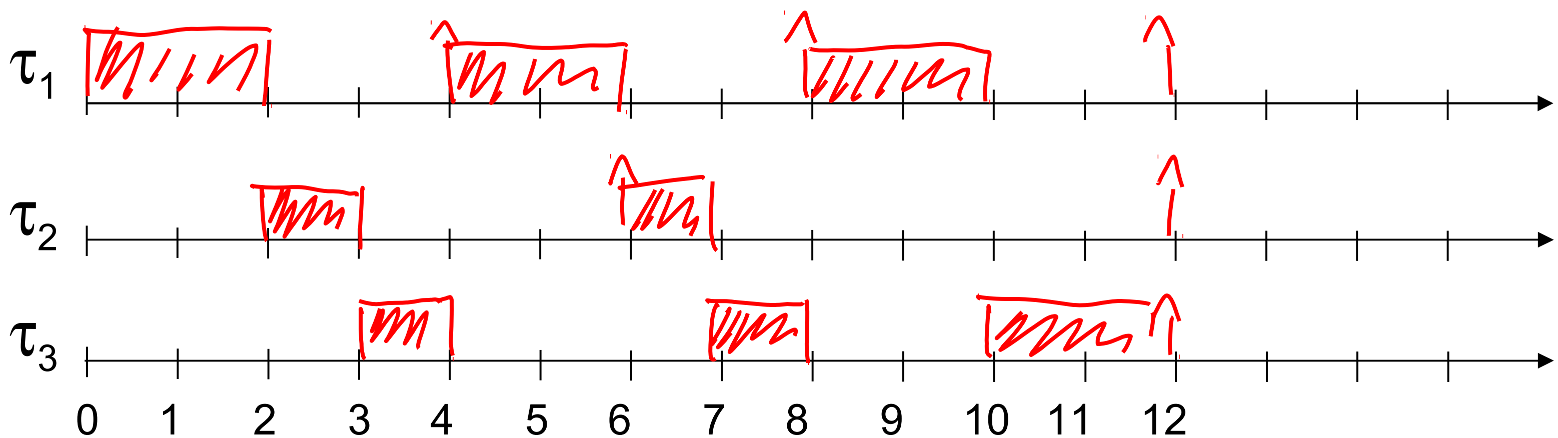
Example for RM (1)

	τ_1	τ_2	τ_3
Φ_i	0	0	0
T_i	4	6	12
C_i	2	1	4
D_i	4	6	12

$prio(\tau_1) = \frac{1}{4} >$

$prio(\tau_2) = \frac{1}{6} >$

$prio(\tau_3) = \frac{1}{12}$



feasible schedule by RM

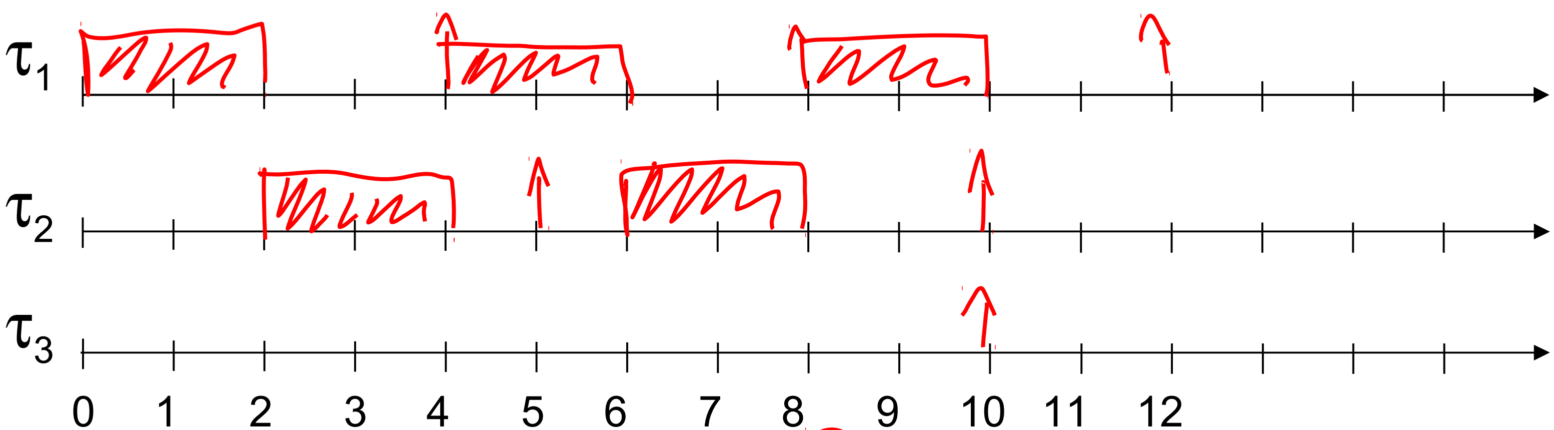
Example for RM (2)

	τ_1	τ_2	τ_3
Φ_i	0	0	0
T_i	4	5	10
C_i	2	2	1
D_i	4	5	10

$prio(\tau_1) = \frac{1}{4}$

$prio(\tau_2) = \frac{1}{5}$

$prio(\tau_3) = \frac{1}{10}$



non-feasible with RM

Optimality of Rate Monotonic Scheduling

- **Theorem (Liu, Layland, 1973):**
  RM is optimal among all fixed-priority scheduling algorithms.

- **Def.:** The **response time $R_{i,j}$** of an instance j of task i is the time (measured from the arrival time) at which the instance is finished: $R_{i,j} = f_{i,j} - a_{i,j}$.
- The critical instant of a task is the time at which the arrival of the task will produce the largest response time.

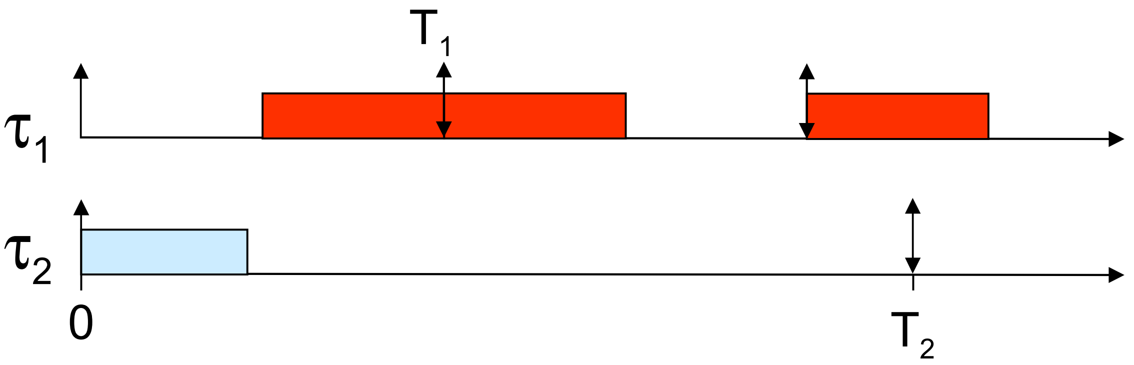
Response times and critical instants

- **Observation**:
  For RM, the critical instant t of a task τ_i is given by the time when $\tau_{i,j}$ arrives together with all tasks τ_1, ..., τ_{i-1} with higher priority.

Response times and critical instants

- For our “worst case task sets” we can assume that there are critical instants where an instance of a task arrives together with all higher priority tasks.
- A task set is schedulable, if the response time at these critical instants is not larger than the relative deadline.

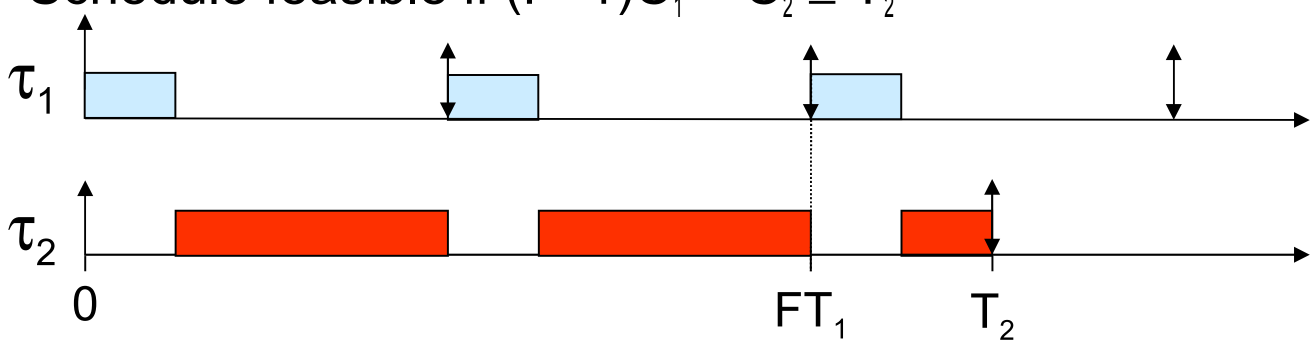
Non-RM Schedule



Schedule feasible iff $C_1 + C_2 \leq T_1$

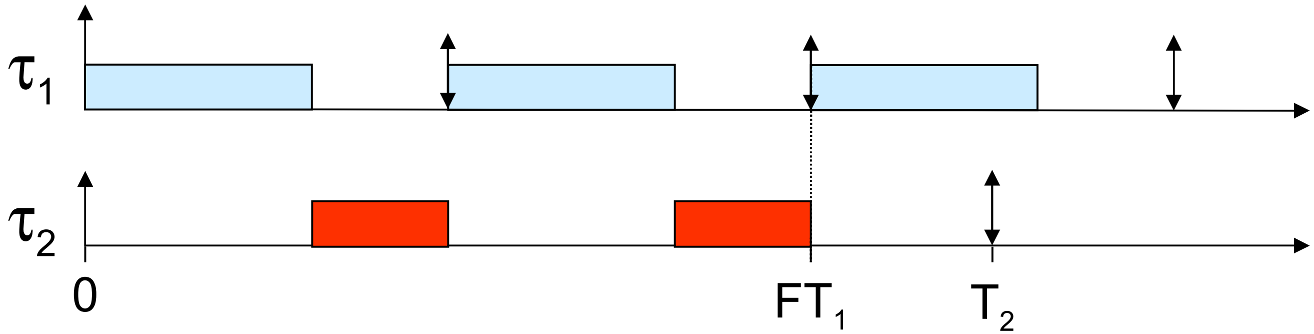
RM-Schedule

- Let $F = \lfloor T_2 / T_1 \rfloor$ be the number of periods of τ_1 entirely contained in T_2.
- Case 1:
  - The computation time C_1 is short enough, so that all requests of τ_1 within period of τ_2 are completed before second request of τ_2.
  - I.e. $C_1 \leq T_2 - F\, T_1$
  - Schedule feasible if $(F+1)C_1 + C_2 \leq T_2$



RM-Schedule

- Case 2:
  - The second request of τ_2 arrives when τ_1 is running.
  - I.e. $C_1 \geq T_2 - F\,T_1$



Schedule feasible if $FC_1 + C_2 \leq FT_1$

Proof of Liu/Layland