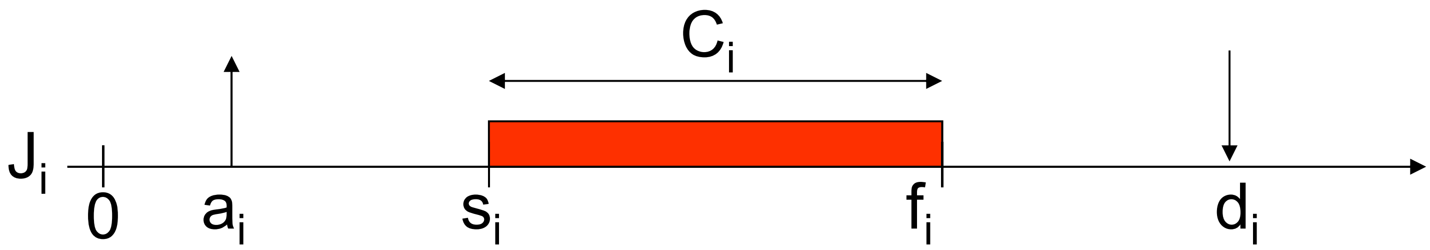
Embedded Systems 16

REVIEW: Aperiodic scheduling



- 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 scheduling algorithm:
    - Preemptive, non-preemptive
    - Off-line / on-line
    - Optimal / heuristic
    - One processor / multi-processor
    - …
  - Cost function:
    - Minimize maximum lateness
    - …
- Find:
  - Feasible schedule
  - Optimal schedule according to given cost function

REVIEW: EDD – Earliest Due Date

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.

REVIEW: EDF – Earliest Deadline First

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

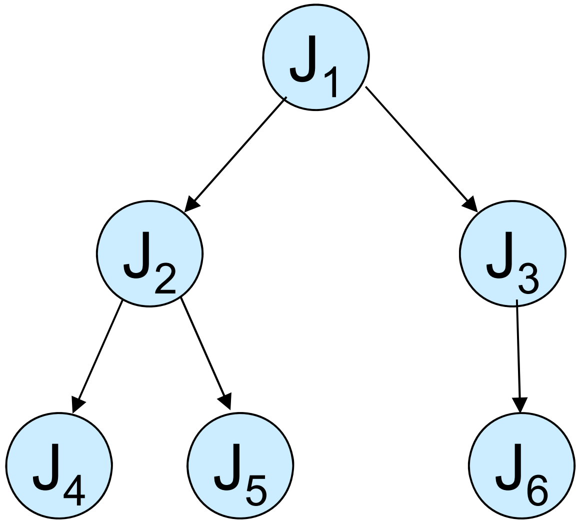
- **Theorem (Horn ’74):**
  Given a set of n independent task **with arbitrary arrival times**, any algorithm that at every instant executes the task with the earliest absolute deadline among all the ready tasks is optimal with respect to minimizing the maximum lateness.

REVIEW: Non-preemptive version

- **Theorem** (Jeffay et al. '91): EDF is an optimal **non-idle** scheduling algorithm also in a non-preemptive task model.
- Non-preemptive scheduling with **idle** schedules allowed is **NP-hard**
- Possible approaches:
  - Heuristics
  - Bratley's algorithm: Branch-and-bound

Scheduling with precedence constraints

	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?

Algorithm 1:

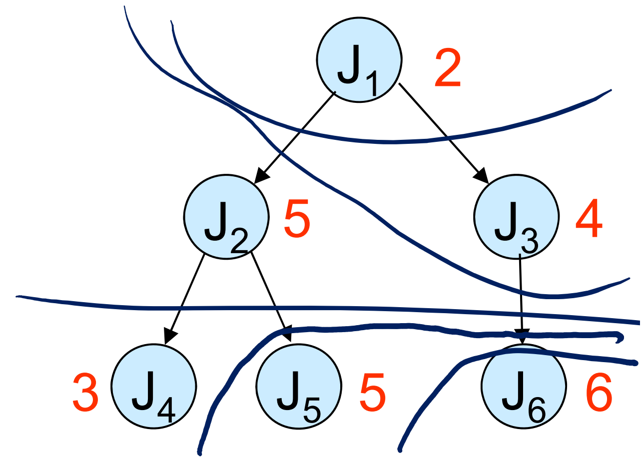
1. Among all **sources** in the precedence graph select the task T with **earliest** deadline. Schedule T **first**.
2. Remove T from G.
3. Repeat.

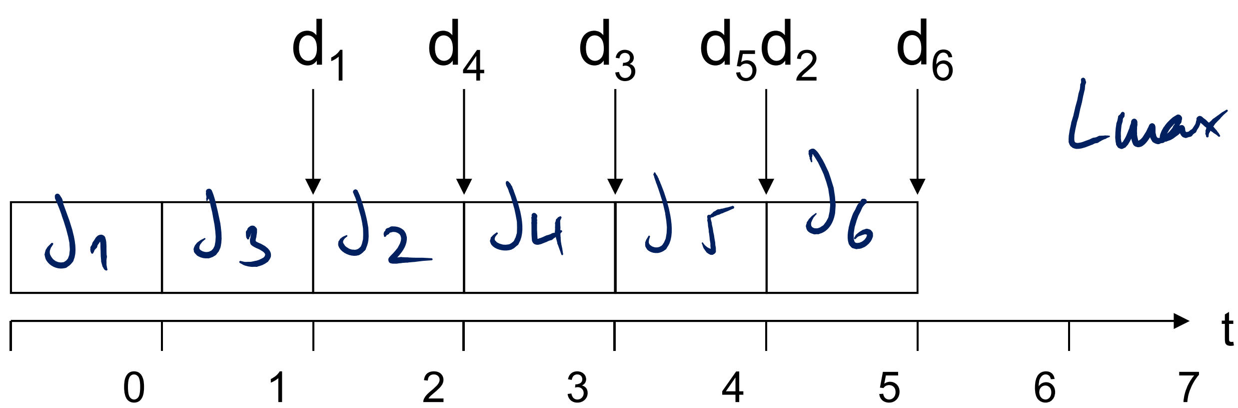
Algorithm 2:

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

Example (continued)

- Algorithm 1:

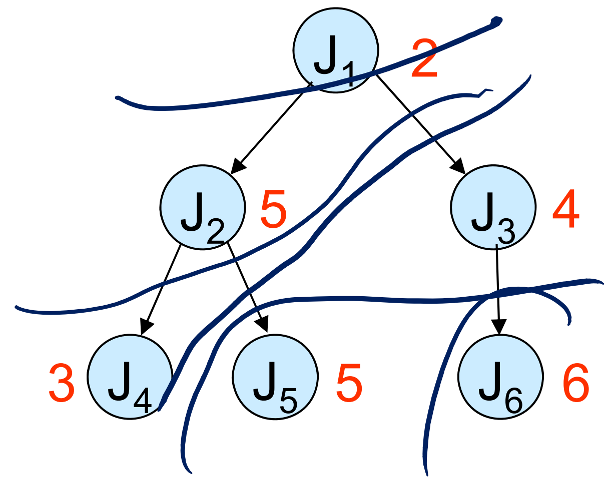


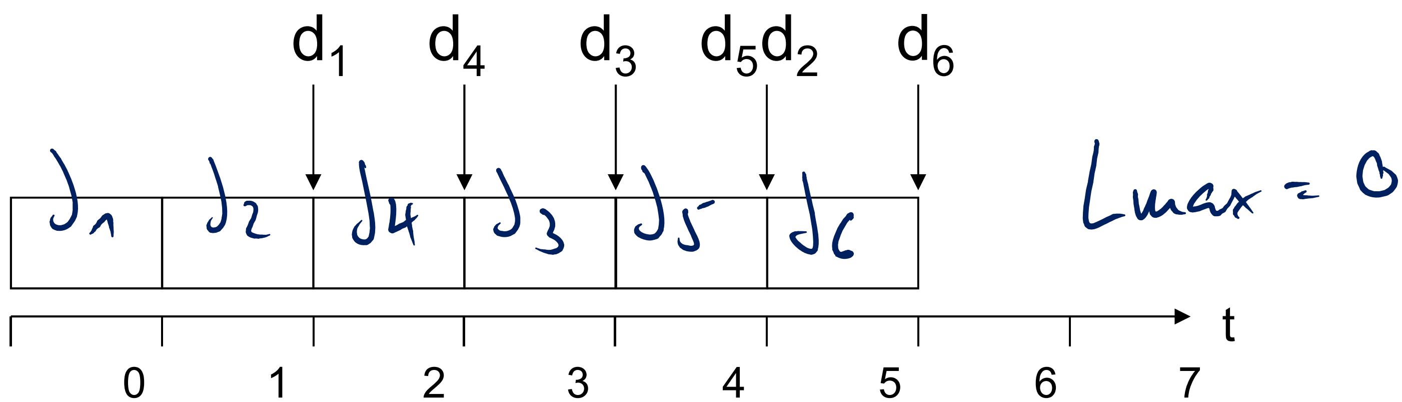


$L_{max} = L_4 = 1$

Example (continued)

- Algorithm 2:





Example (continued)

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

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

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

LDF

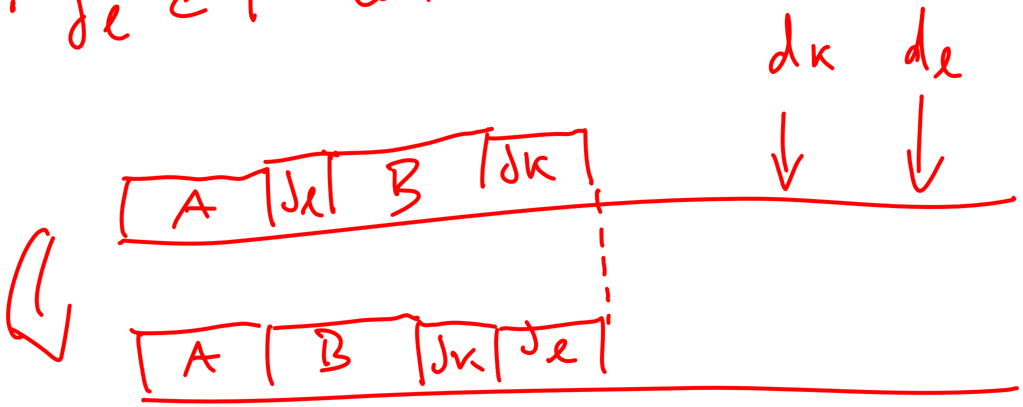
- LDF is optimal.
- LDF can be applied only as off-line algorithm.

- Complexity of LDF:
  - $O(|E|)$ for repeatedly computing the current set Γ of tasks with no successors in the precedence graph $G = (V, E)$.
  - $O(\log n)$ for inserting tasks into the ordered set Γ (ordering wrt. d_i).
  - Overall cost: $O(n * \max(|E|, \log n))$

LDF

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

- Task set $J = \{J_1, \dots, J_n\}$
- $T \subseteq J$ subset without successors
- $J_l \in T$ with latest deadline



We show that we can move J_l to the end with

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

1) precedence is not violated
(J_e does not have successors)

2) $L'_{max} = \max\{L'_{max}(A), L'_{max}(B), L'_k, L'_e\}$

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

$L'_{max}(B) \le L_{max}(B)$ starts & ends earlier

$L'_k < L_k$ starts & ends earlier

$$L'_e = \sum_{i=1}^{n} C_i - d_e < \sum_{i=1}^{n} C_i - d_k = L_k$$

Remove J_e and continue.

Preemptive

- Non-preemptive scheduling with non-synchronous arrival times, deadlines and precedence constraints is **NP-hard.**

- Modified EDF for **preemptive** scheduling, **arbitrary arrival times**

EDF with precedence constraints

1. Modify arrival times

- For any initial node J_i of the precedence graph, set $a_i^* := a_i$.
- For any task J_i such that all predecessors have been processed, set $a_i^* := \max \{a_i, a_h^* + C_h \mid J_h \to J_i\}$

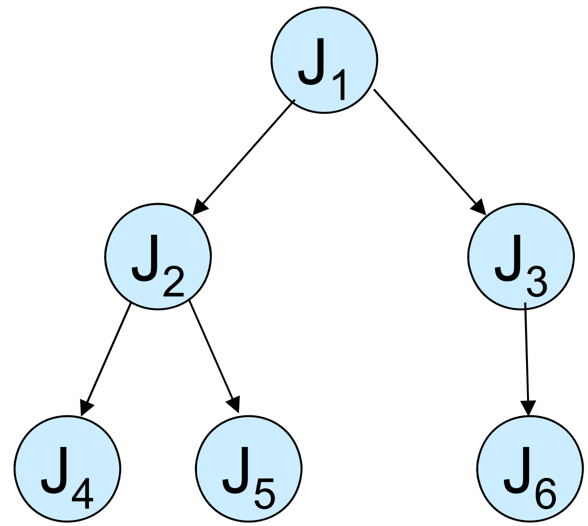
2. Modify deadlines

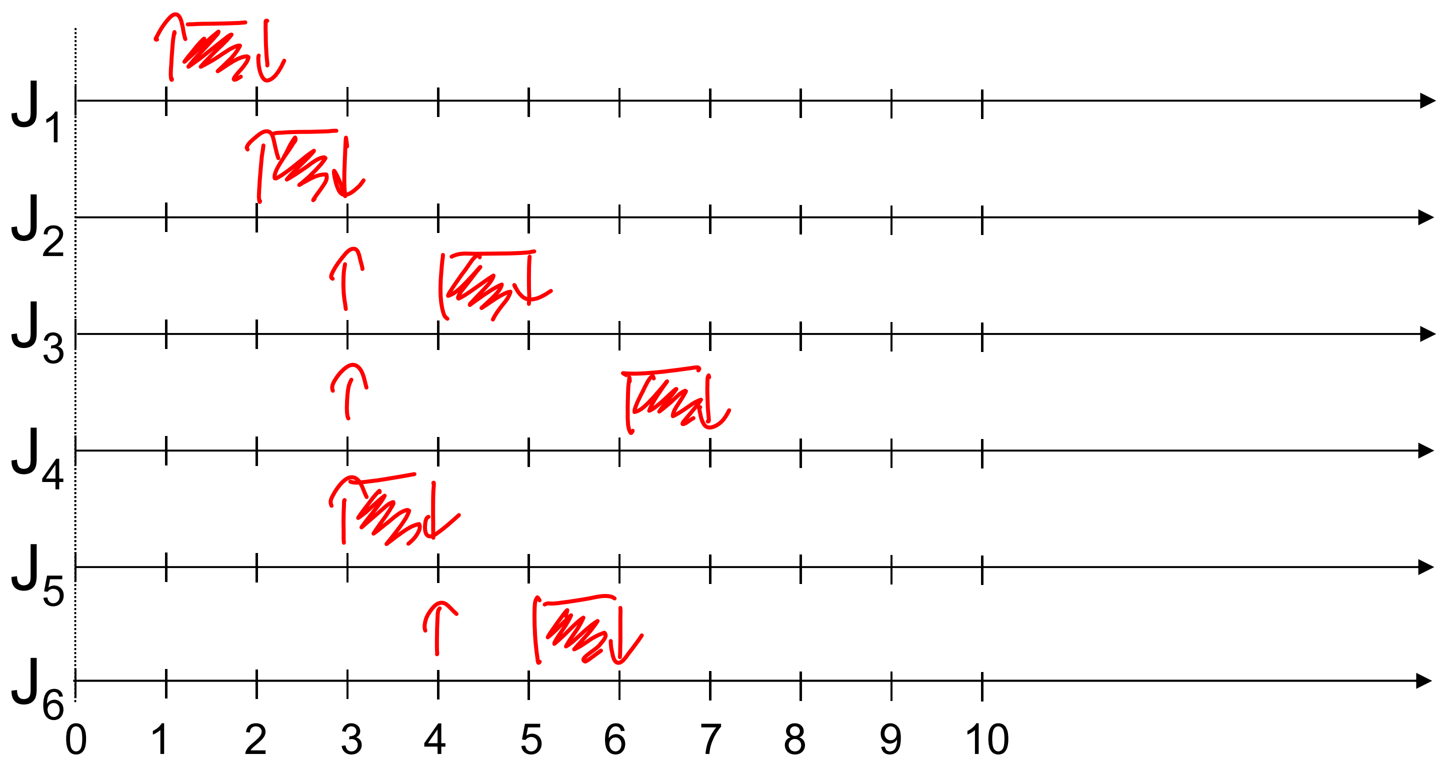
- For any terminal node J_i of the precedence graph, set $d_i^* := d_i$.
- For any task Ji such that all successors have been processed, set $d_i^* := \min \{d_i, d_h^* - C_h \mid J_i \to J_h\}$

($J_h \to J_i$: J_h is a direct predecessor of J_i)

Example

	J_1	J_2	J_3	J_4	J_5	J_6
A_i	1 1	2 0	3 3	3 1	3 1	4 1
C_i	1	1	1	1	1	1
d_i	2 5	3 5	5 6	7 7	4 4	6 6





EDF with precedence constraints

Theorem: The given task set is schedulable such that the precedence constraints are met if and only if the modified task set is schedulable under EDF.

"$\Rightarrow$" In any feasible schedule σ that meets the precedence constraints, we have that

$$s_i \geq \max\{a_i, a_h^* + C_h \mid J_h \to J_i\}$$

and

$$f_i \leq \min\{d_i, d_h^* - C_h \mid J_i \to J_h\}$$

$\Rightarrow$ σ is feasible for the modified task set.

"$\Leftarrow$" For any $J_i < J_j$, we have that

$\underbrace{a_i^* < a_j^*}$ and $\underbrace{d_i^* < d_j^*}$

When J_j arrives then J_i has already arrived and J_i has the earlier deadline

$\Rightarrow$ J_j cannot start before J_i

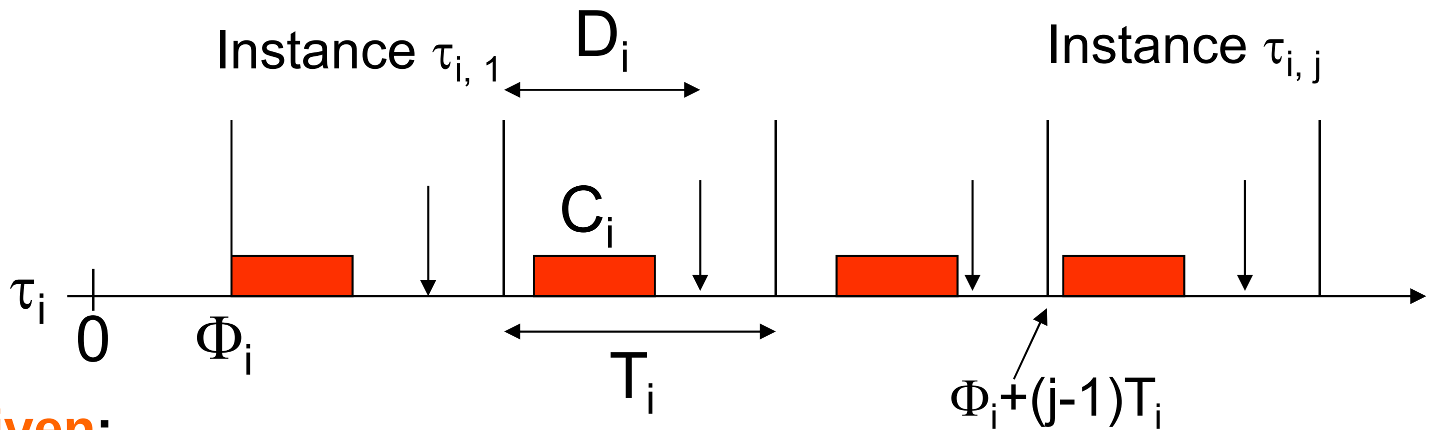
J_j cannot preempt J_i

$\Rightarrow$ precedence constraints are satisfied.

$a_i^* \geq a_i$, $d_i^* \leq d_i$ $\Rightarrow$ timing constraints satisfied.

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 $a_{i,j} = \Phi_i + (j-1)\ T_i$,
    - deadline $d_{i,j} = \Phi_i + (j-1)\ T_i + D_i$,
- **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., $d_{i,j} = \Phi_i + j \cdot T_i$)

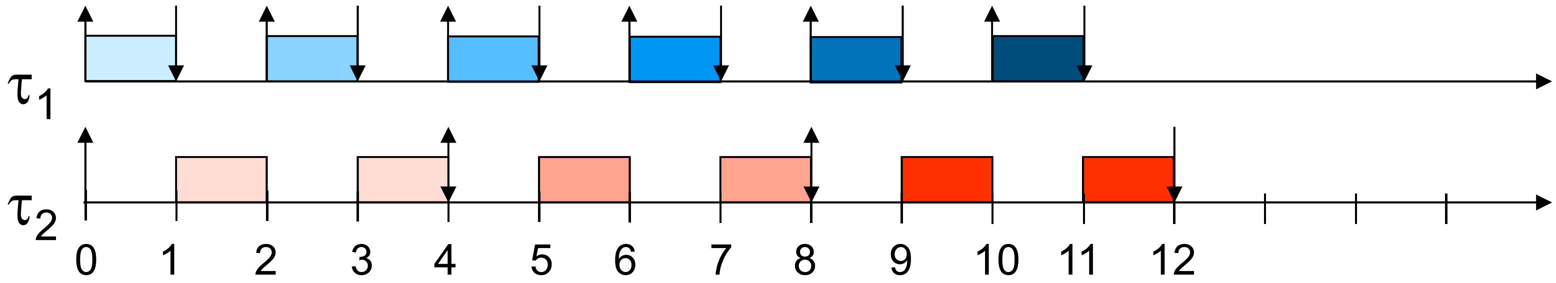
A.4. All tasks in Γ are independent.

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)

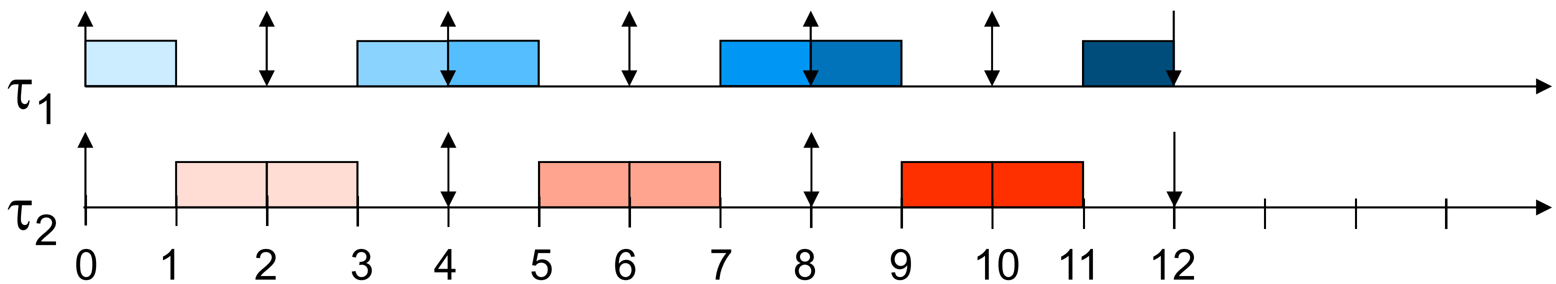
	τ_1	τ_2
Φ_i	0	0
T_i	2	4
C_i	1	2
D_i	1	4



- 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

- No feasible schedule for single processor.

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

$T_1 \cdot T_2 = 12$

Within 12 units:

$\frac{12}{3} = 4$ executions of τ_1

$\frac{12}{4} = 3$ executions of τ_2

- No feasible schedule for single processor.

$4 \times 2 = 8$ units by τ_1

$3 \times 2 = 6$ units by τ_2

14 units computation within 12 units impossible!

Processor utilization

Definition:

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

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

$$U = \frac{2}{3} + \frac{2}{4} = \frac{14}{12} > 1$$

Processor utilization 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

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