Least Laxity First (LLF)

So we have another (better ?) algorithm than EDF that takes into account the remaining time of computation.

Laxity (or **slack):** d_i - t - $c_i(t)$ *(Recall – Chapter 2 definition!)*

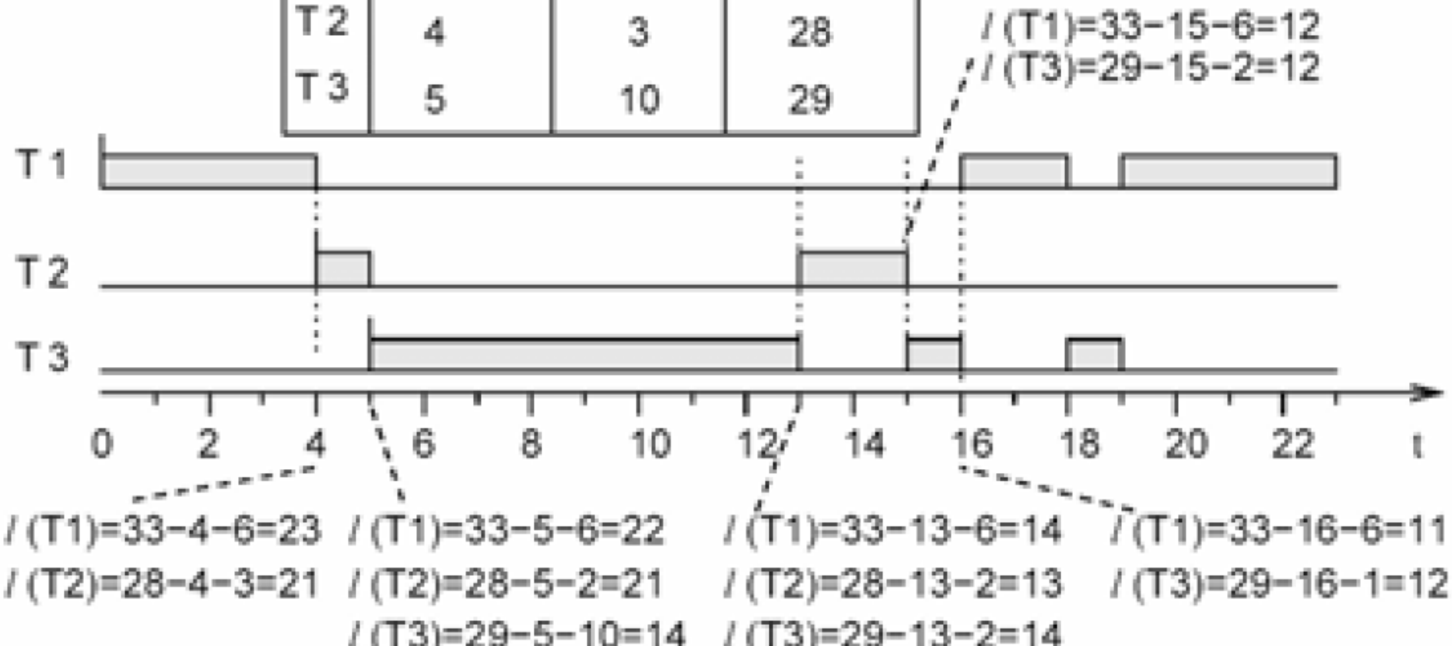
where $c_i(t)$ is the residual WCET.

- **Scheduling Based on Slack / Laxity**
  - The processor is assigned to the process with the shortest remaining delay time (least laxity first, LLF, least slack time)
  - Laxity: time between earliest completion time and deadline, thus EDF plus remaining computation time

LLF Example

- Priorities = decreasing function of the laxity (the less laxity, the higher the priority); dynamically changing priority; preemptive.

	arrival	duration	deadline
T1	0	10	33
T2	4	3	28
T3	5	10	29



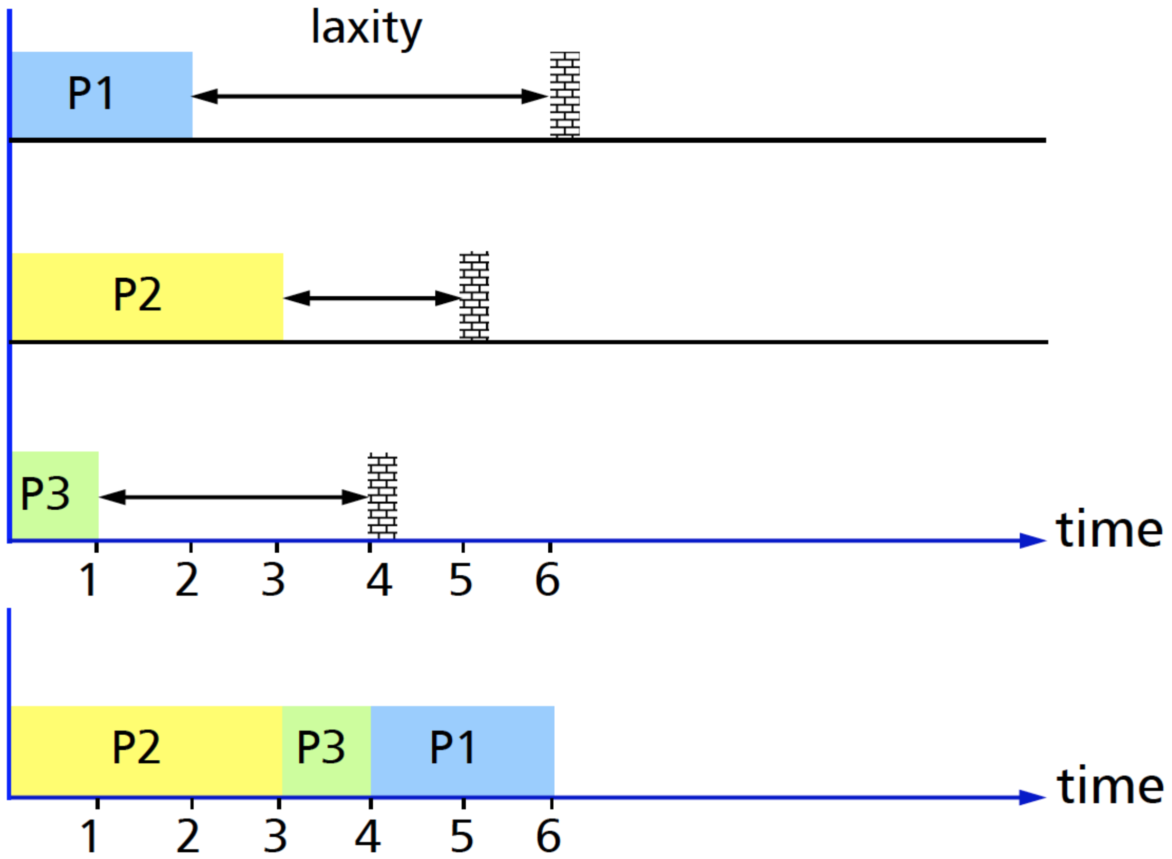
LLF – Implementation issues

- Although the schedule gets generated, following are the issues when we attempt to implement:

- No look-ahead style of working – decides based on the current status; This means when the time intervals of arrivals are very short with short deadlines, the schedule generated by LLF may not be optimal; It can assure only a good (!) solution.
- Overhead in determining the T(i) at every interval O(n) (n: tasks)
- Space complexity – Entire status of each task need to be maintained until it gets completed;

LLF – Identical ready times

- **Example**: equal ready times, static schedule



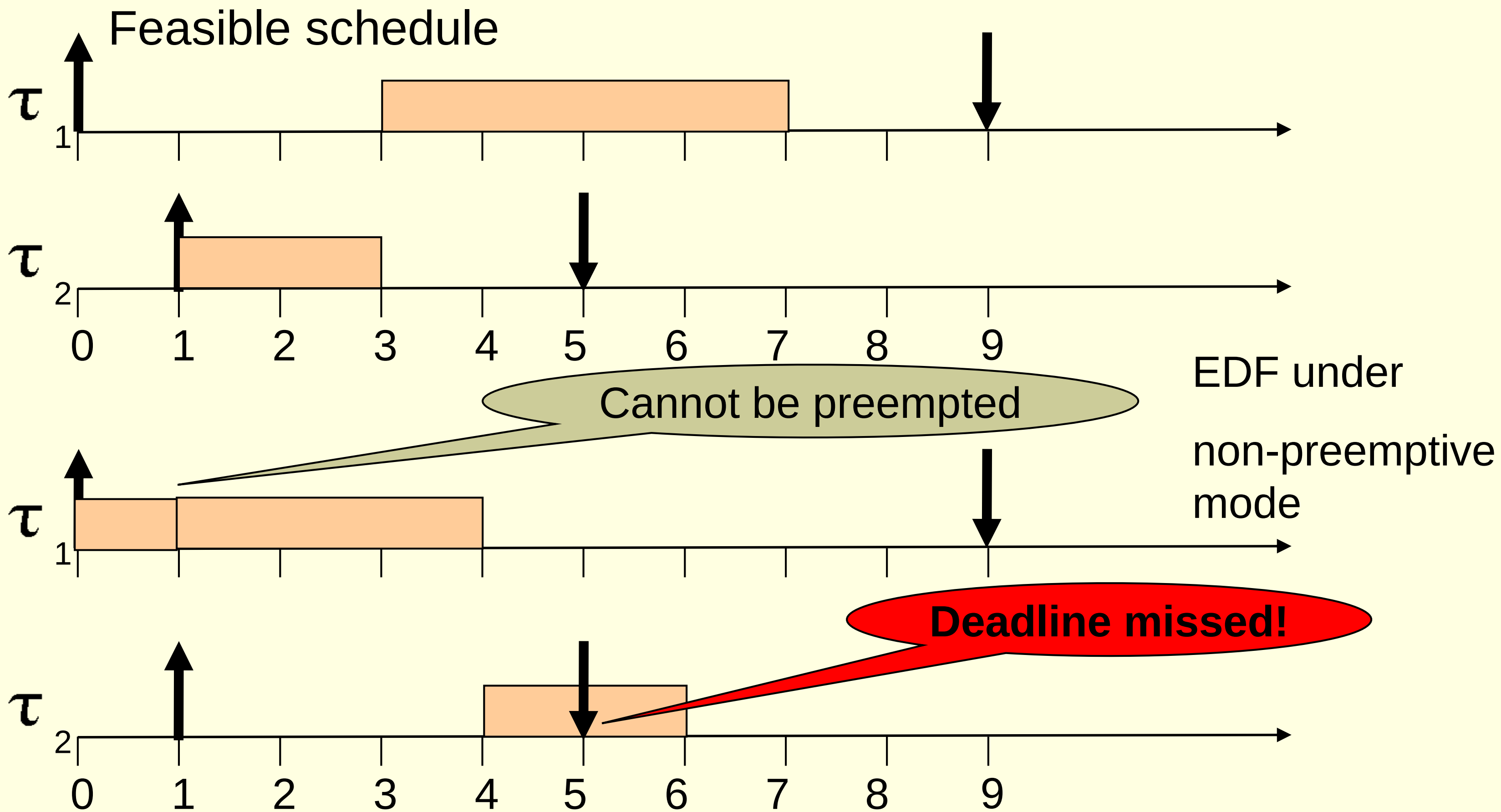
Time complexity issues

- **EDD**
  - O(*n log n*) to order the task set
  - O(*n*) to guarantee the whole task set
- **EDF**
  - O(*n*) to insert a new task in the queue
  - O(*n*) to guarantee a new task

Remark on an important property of Optimal algorithms - If an optimal algorithm (in the sense of feasibility) produces an infeasible schedule, then no algorithm can generate a feasible schedule.

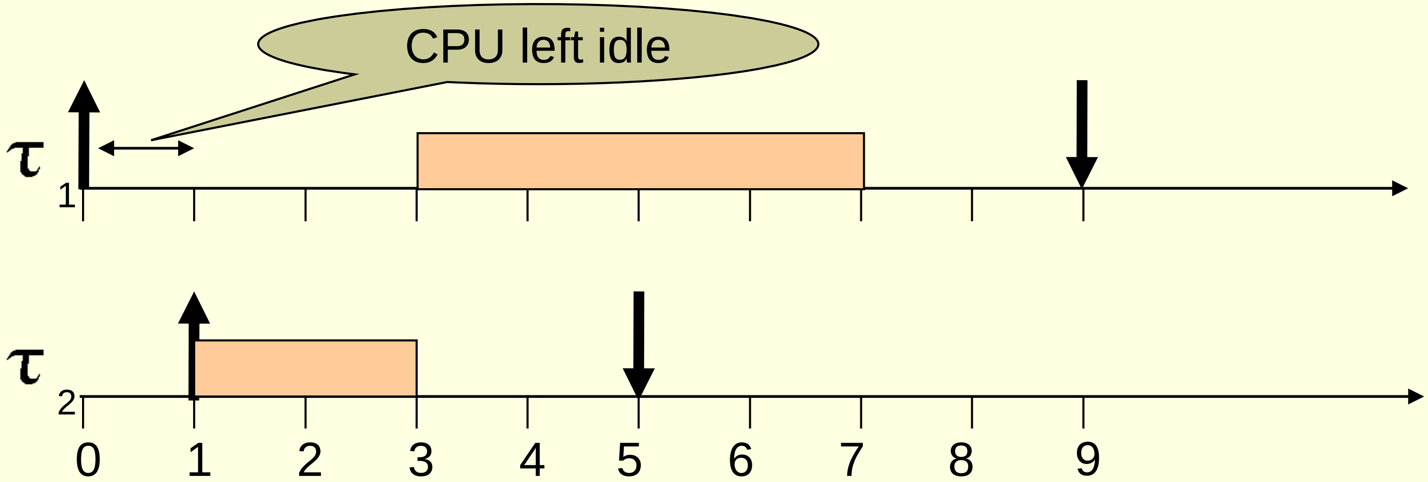
EDF on Non-Preemptive Scheduling criteria

- Under non-preemptive execution, EDF is <u>not</u> optimal.



Clairvoyant strategy

- To achieve optimality, an algorithm should be clairvoyant, and decide to leave the CPU idle in the presence of ready tasks.



Note: If we forbid to leave the CPU idle in the presence of ready tasks, then EDF is optimal.

Non-Preemptive Scheduling

Non-Preemptive-EDF is optimal among work-conserving scheduling algorithms

- Work-conserving: Defined as an algorithm that does not leave the processor idle, if there is work to do i.e., non-idle algorithm.

Non-Preemptive Scheduling Algorithms

- The problem of finding a feasible schedule is NP hard and is treated off-line with tree search algorithms.
- Examples of tree algorithm
  - Bratley’s Algorithm
  - Spring algorithm (*Self-learning exercise!*)

Time-Triggered Systems - Non-Preemptive tasks with arbitrary arrival times - Bratley's Algorithm to generate a feasible schedule

- Time-triggered systems – System that triggers events at predefined time instants for autonomous control
- Assumption – Arrival times (arbitrary) are known in advance; No preemption allowed;
- Key Idea – At every step of the search do the following:

  - (a) Check if a task misses its deadline;

  - (b) See if you have obtained a feasible schedule;

If (a): fully abandon the search along that path (pruning technique);

Feasible solution sequence - Backtrack

Refer to an example shown in the next slide

Example – Bratley's algorithm

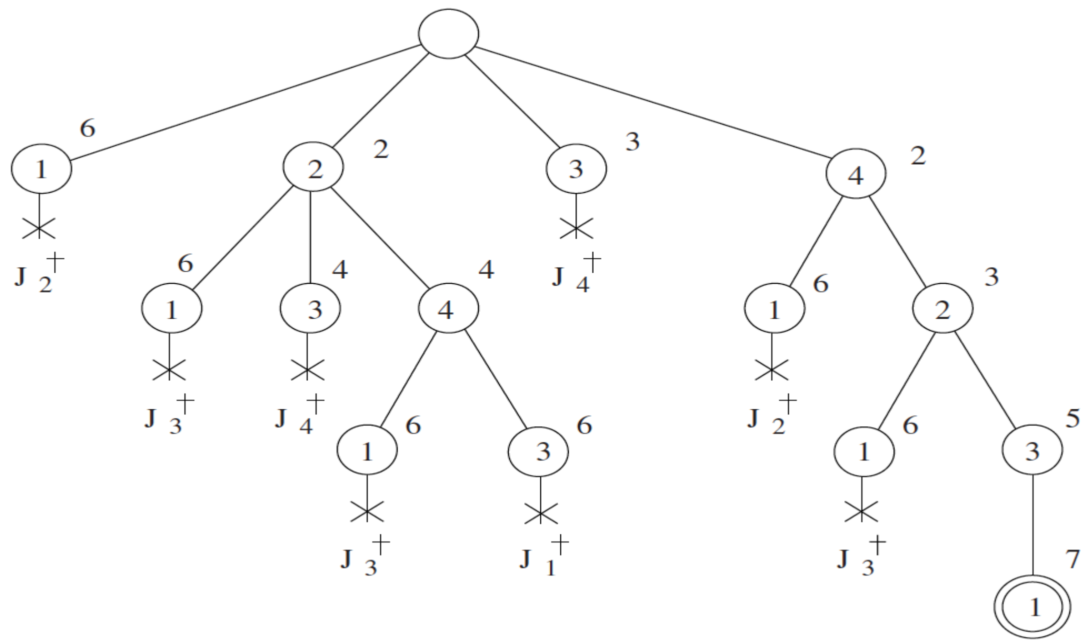
	J_1	J_2	J_3	J_4
a_i	4	1	1	0
C_i	2	1	2	2
d_i	7	5	6	4

Number in the node = scheduled task

Number outside the node = finishing time

$J_i^{\dagger}$ = task that misses its deadline

◎ = feasible schedule



Arrival times are known in advance; No preemption allowed;

Time Complexity ?