Sharif University of Technology
Department of Computer Science and Engineering

Lec. 3.1:

Aperiodic Task Scheduling

Real-Time Computing

S. Safari

2023

Introduction

- Presenting a variety of algorithms for scheduling real-time aperiodic tasks on a single processor.
- Each algorithm represents a solution for a particular scheduling problem.
  - Through a set of assumptions on the task set and by an optimality criterion
- Many of the presented algorithms can be extended to work on multiprocessor or distributed architectures and deal with more complex task models.

Introduction (2)

- To facilitate the description of the scheduling problems presented in this lecture, we introduce a systematic notation that could serve as a basis for a classification scheme.
- Such a notation classifies all algorithms using three fields $\alpha \mid \beta \mid \gamma$, having the following meaning:
  - The first field α describes the machine environment on which the task set has to be scheduled (uniprocessor, multiprocessor, distributed architecture, and so on).
  - The second field β describes task and resource characteristics (preemptive, independent versus precedence constrained, synchronous activations, and so on).
  - The third field γ indicates the optimality criterion (performance measure) to be followed in the schedule.

Examples of Classifications

- $1 \mid prec \mid L_{max}$
  - Denoting the problem of scheduling a set of tasks with precedence constraints on a uniprocessor machine in order to minimize the maximum lateness.
  - If no additional constraints are indicated in the second field, preemption is allowed at any time, and tasks can have arbitrary arrivals.
- $3 \mid no\_preem \mid \sum f_i$
  - Denoting the problem of scheduling a set of tasks on a three-core platform. Preemption is not allowed and the objective is to minimize the sum of the finishing times.
  - Since no other constraints are indicated in the second field, tasks do not have precedence nor resource constraints but have arbitrary arrival times.

Examples of Classifications (2)

- $2 \mid sync \mid \sum Late_i$
  - Denoting the problem of scheduling a set of tasks on a two-core platform. Tasks have synchronous arrival times and do not have other constraints. The objective is to minimize the number of late tasks.

Jackson's Algorithm

- The problem considered by this algorithm is $1 \mid sync \mid L_{max}$.
- That is, a set J of n aperiodic tasks has to be scheduled on a single processor, minimizing the maximum lateness.
- All tasks consist of a single job, have synchronous arrival times, but can have different computation times and deadlines.
- No other constraints are considered.
- Tasks must be independent; that is, cannot have precedence relations and cannot share resources in exclusive mode.

Jackson's Algorithm (2)

- Assuming all tasks are activated at time $t = 0$.
  - Each job J_i can be completely characterized by two parameters:
    - A computation time C_i
    - A relative deadline D_i (equal to the absolute deadline)

$$\mathcal{J} = \{J_i(C_i, D_i),\ i = 1, \ldots, n\}.$$

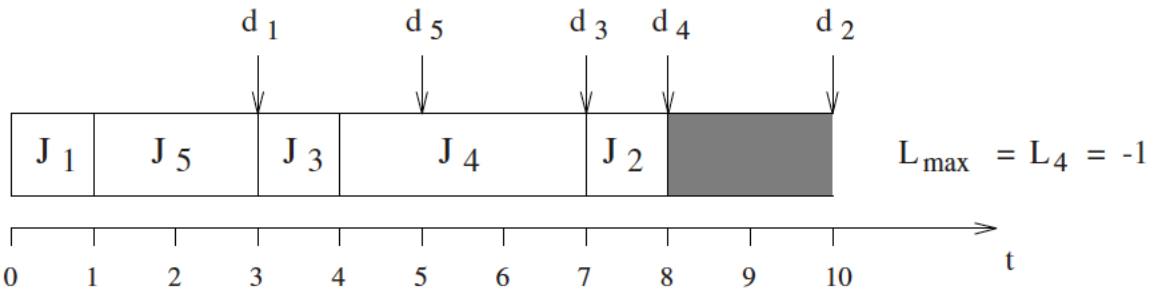
Earliest Due Date (EDD)

- Found by Jackson in 1955
- **Theorem (Jackson's rule).** *Given a set of n independent tasks, any algorithm that executes the tasks in order of non-decreasing deadlines is optimal with respect to minimizing the maximum lateness.*

Examples of Earliest Due Date (EDD)

- Consider a set of five tasks, simultaneously activated at time t = 0, whose parameters (worst-case computation times and deadlines) are indicated in the table.

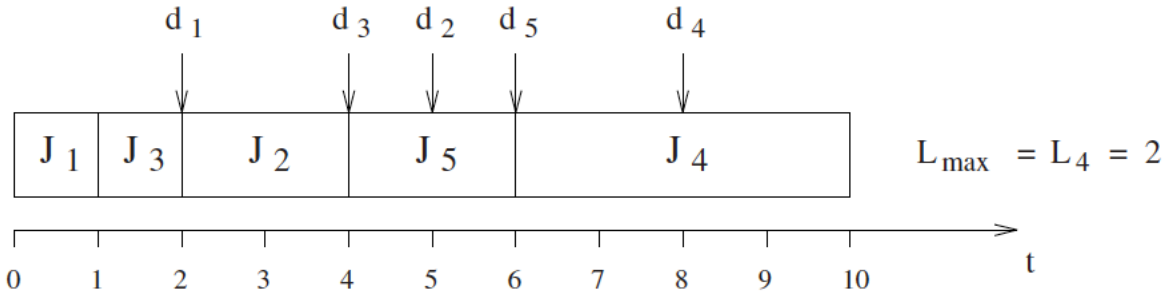
	J_1	J_2	J_3	J_4	J_5
C_i	1	1	1	3	2
d_i	3	10	7	8	5



Examples of Earliest Due Date (EDD) (2)

- Illustrating an example in which the task set cannot be feasibly scheduled.
- Note: EDD produces the optimal schedule that minimizes the maximum lateness. Since J_4 misses its deadline, the maximum lateness is greater than zero ($L_{max} = \mathrm{L4} = 2$).

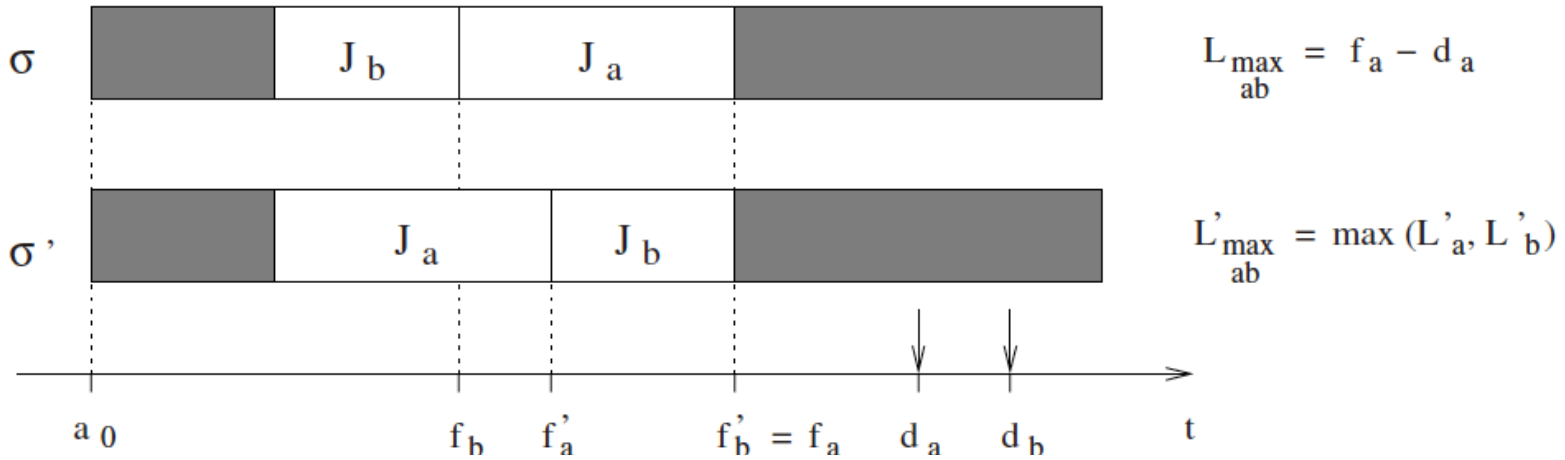
	J_1	J_2	J_3	J_4	J_5
C_i	1	2	1	4	2
d_i	2	5	4	8	6



Earliest Due Date (EDD)

- Found by Jackson in 1955
- **Theorem (Jackson's rule).** *Given a set of n independent tasks, any algorithm that executes the tasks in order of non-decreasing deadlines is optimal with respect to minimizing the maximum lateness.*
- **Proof.** Jackson's theorem can be proved by a simple interchange argument. Let σ be a schedule produced by any algorithm A. If A is different than EDD, then there exist two tasks J_a and J_b, with $d_a \leq d_b$, such that J_b immediately precedes J_a in σ. Now, let σ' be a schedule obtained from σ by exchanging J_a with J_b, so that J_a immediately precedes J_b in σ'.

Optimality of Earliest Due Date (EDD)



1. If $L'_a \geq L'_b$, then $L'_{max}(a,b) = f'_a - d_a$, and, since $f'_a < f_a$, we have $L'_{max}(a,b) < L_{max}(a,b)$.

2. If $L'_a \leq L'_b$, then $L'_{max}(a,b) = f'_b - d_b = f_a - d_b$, and, since $d_a < d_b$, we have $L'_{max}(a,b) < L_{max}(a,b)$.

- In both cases: $L'_{max}(a,b) < L_{max}(a,b)$
- Interchanging J_a and J_b in σ cannot increase the maximum lateness.
- Since in each transposition the maximum lateness cannot increase, σ_{EDD} is optimal.

Guarantee of Earliest Due Date (EDD)

- In the worst case, all tasks should complete before their deadlines.
- For each task, the worst-case finishing time f_i is less than or equal to its deadline d_i:

$$\forall i = 1, \ldots, n \quad f_i \leq d_i.$$

- Assume that tasks J_1, J_2, ..., J_n are listed by increasing deadlines, so that J_1 is the task with the earliest deadline. In this case, the worst-case finishing time of task J_i can be easily computed as:

$$f_i = \sum_{k=1}^{i} C_k.$$

- If the task set consists of n tasks, the guarantee test can be performed by verifying the following n conditions:

$$\forall i = 1, \ldots, n \quad \sum_{k=1}^{i} C_k \leq d_i.$$

Horn's Algorithm

- If tasks are not synchronous but can have arbitrary arrival times.
  - Tasks can be activated dynamically during execution.
  - Preemption becomes an important factor.
- In 1974, Horn found an elegant solution to the problem of scheduling a set of n independent tasks on a uniprocessor system, when tasks may have dynamic arrivals and preemption is allowed ($1 \mid preem \mid L_{max}$).
- The algorithm is called *Earliest Deadline First* (*EDF*).
- **Theorem (Horn).** *Given a set of n independent tasks with arbitrary arrival times, any algorithm that at any instant executes the task with the earliest absolute deadline among all the ready tasks is optimal with respect to minimizing the maximum lateness.*

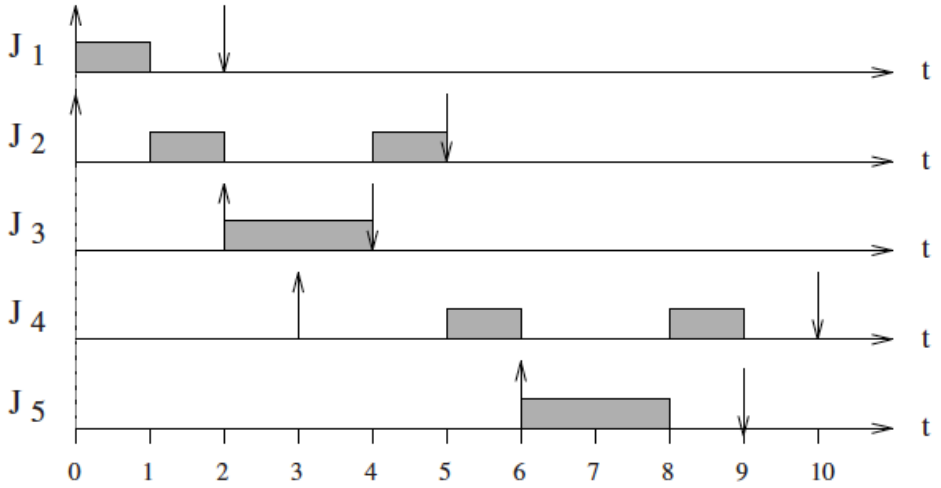
Assignment: EDF Optimality

- The original proof provided by Dertouzos [Der74] shows that EDF is optimal in the sense of feasibility.

Examples of Earliest Deadline First (EDF)

- A set of five tasks
- All tasks meet their deadlines and the maximum lateness is $L_{max}=L_2=0$.

	J_1	J_2	J_3	J_4	J_5
a_i	0	0	2	3	6
C_i	1	2	2	2	2
d_i	2	5	4	10	9



Guarantee of Earliest Deadline First (EDF)

- Let $\mathcal{J}$ be the current set of active tasks, which have been previously guaranteed. Let J_{new} be a newly arrived task. In order to accept J_{new} in the system we have to guarantee that the new task set $\mathcal{J}' = \mathcal{J} \cup \{J_{new}\}$ is also schedulable.
- Assuming that all tasks (including J_{new}) are ordered by increasing deadlines, so that J_1 is the task with the earliest deadline.
- Let $c_i(t)$ be the remaining worst-case execution time of task J_i. Hence, at time t, the worst-case finishing time of task J_i can be easily computed as:

$$f_i = \sum_{k=1}^{i} c_k(t). \qquad f_i = f_{i-1} + c_i(t)$$

- The schedulability can be guaranteed by the following conditions:

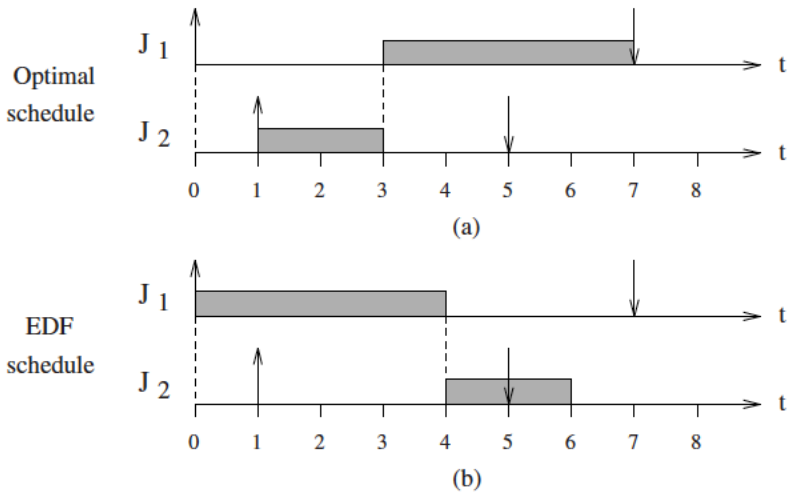
$$\forall i = 1, \ldots, n \qquad \sum_{k=1}^{i} c_k(t) \leq d_i.$$

Non-Preemptive Scheduling

- When preemption is not allowed and tasks can have arbitrary arrivals, the problem of minimizing the maximum lateness and the problem of finding a feasible schedule become **NP-hard**.

- An example that shows that EDF is no longer optimal if tasks cannot be preempted during their execution.
- Although a feasible schedule exists for that task set, EDF does not produce a feasible schedule.

	J_1	J_2
a_i	0	1
C_i	4	2
d_i	7	5

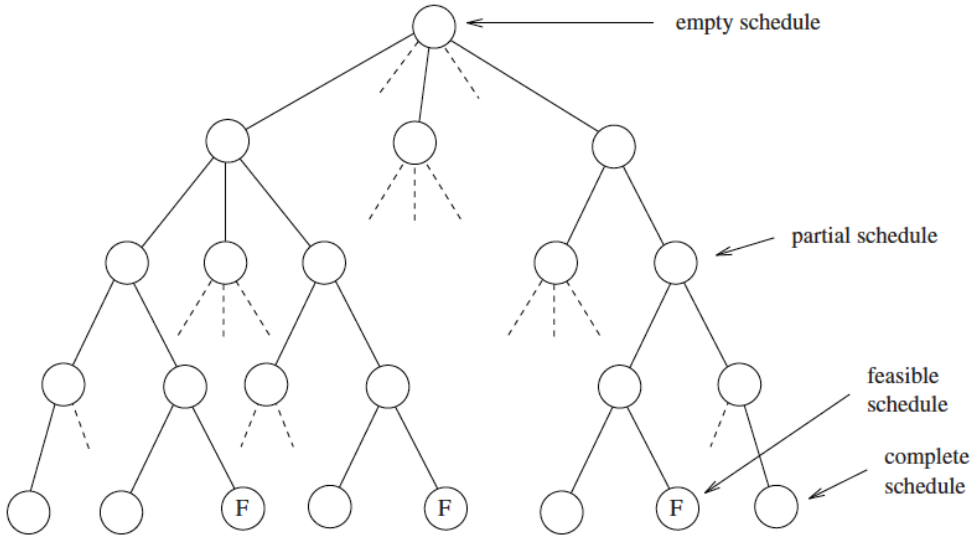


Non-Idle Scheduling Algorithms

- A scheduling algorithm that does not permit the processor to be idle when there are active jobs is called a *non-idle algorithm*.
- By restricting to the case of *non-idle scheduling algorithms*, EDF is still optimal in a non-preemptive task model.

Branch-and-Bound Algorithms

- When arrival times are known a priori, non-preemptive scheduling problems are usually treated by branch-and-bound algorithms.
  - Performing well in the average case but degrade to exponential complexity in the worst case.
  - The structure of the search space is a search tree, where the root is an ***empty schedule***, an intermediate vertex is a ***partial schedule***, and a terminal vertex (leaf) is a ***complete schedule***.



Bratley's Algorithm

- Solving the problem of finding a feasible schedule of a set of non-preemptive tasks with arbitrary arrival times ($1|no\_preem\ |feasible$).
- The algorithm starts with an empty schedule and, at each step of the search, visits a new vertex and adds a task in the partial schedule.
- With respect to the exhaustive search, Bratley's algorithm uses a pruning technique to determine when a current search can be reasonably abandoned.
- A branch is abandoned when:
  - The addition of any node to the current path causes a missed deadline;
  - A feasible schedule is found at the current path.

An Example of Bratley's Algorithm

- Illustrating the paths analyzed in the tree space

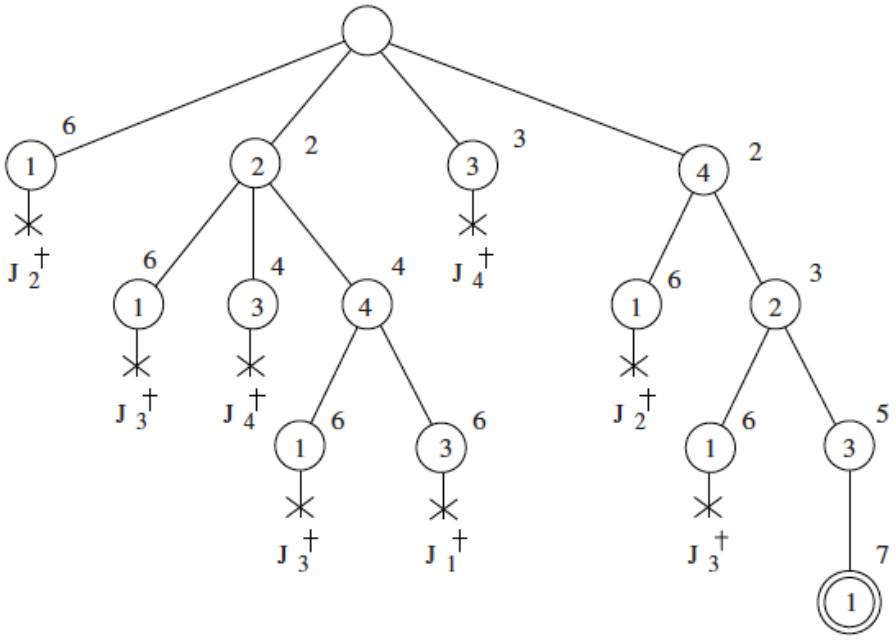
	J_1	J_2	J_3	J_4
a_i	4	1	1	0
C_i	2	1	2	2
d_i	7	5	6	4

Number in the node = scheduled task

Number outside the node = finishing time

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

◎ = feasible schedule



Task Activation List

- In the average case, pruning techniques are very effective for reducing the search space.
  - The worst-case complexity of the algorithm is still $O(n \cdot n!)$.
- Bratley's algorithm can only be used in off-line mode, when all task parameters (including the arrival times) are known in advance.
  - This can be the case of a time-triggered system, where tasks are activated at predefined instants by a timer process.
- As in most off-line real-time systems, the resulting schedule produced by Bratley's algorithm can be stored in a data structure, called ***task activation list***.
  - At run time, a dispatcher simply extracts the next task from the activation list and puts it in execution.

The Spring Algorithm

- Adopted in the Spring kernel [SR87, SR91]
  - A hard real-time kernel designed at the University of Massachusetts at Amherst.
- The objective of the algorithm is to find a feasible schedule when tasks have different types of constraints, such as precedence relations, resource constraints, arbitrary arrivals, non-preemptive properties, and importance levels.
- Extended to include fault-tolerance requirements
- This problem is NP-hard and finding a feasible schedule would be too expensive in terms of computation time, especially for dynamic systems.
- In order to make the algorithm computationally tractable even in the worst case, the search is driven by a heuristic function.

Scheduling with Precedence Constraints

Scheduling with Precedence Constraints

- The problem of finding an optimal schedule for a set of tasks with precedence relations is in general **NP-hard**.
- Optimal algorithms that solve the problem in polynomial time can be found under particular assumptions on the tasks.
- Presenting two algorithms that minimize the maximum lateness by assuming:
  - Synchronous activations
  - Preemptive scheduling

Latest Deadline First (LDF)

- In 1973, Lawler [Law73] presented an optimal algorithm that minimizes the maximum lateness of a set of tasks with precedence relations and simultaneous arrival times.
- $(1 \mid PREC, SYNC \mid L_{MAX})$
- The algorithm is called *Latest Deadline First* (LDF) and can be executed in polynomial time with respect to the number of tasks in the set.

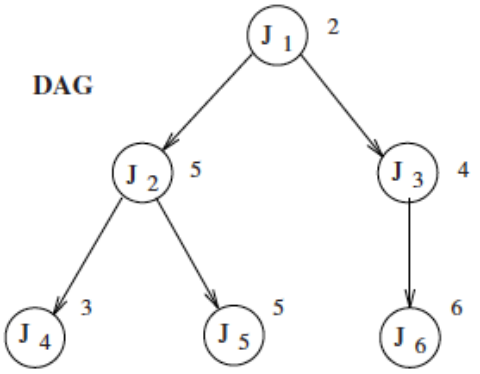
Latest Deadline First (LDF) (2)

- Given a set of n tasks and a directed acyclic graph (DAG) describing their precedence relations.
- LDF builds the scheduling queue from tail to head
  - Among the tasks without successors or whose successors have been all selected, LDF selects the task with the latest deadline to be scheduled last.
- This procedure is repeated until all tasks in the set are selected.
- At run time, tasks are extracted from the head of the queue, so that the first task inserted in the queue will be executed last, whereas the last task inserted in the queue will be executed first.

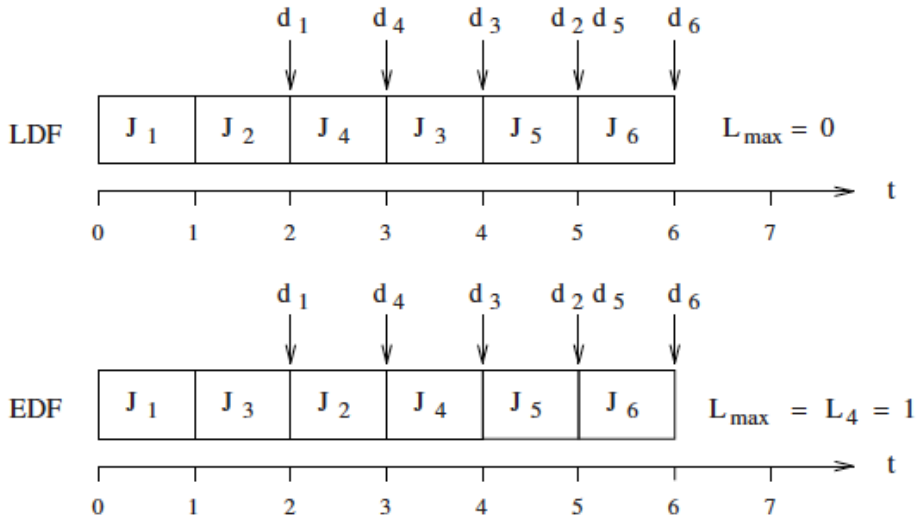
An Example of Latest Deadline First (LDF)

- The numbers beside each node of the graph indicate task deadlines.

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



- Notice that EDF is not optimal under precedence constraints, since it achieves a greater L_{max} with respect to LDF.



EDF with Precedence Constraints

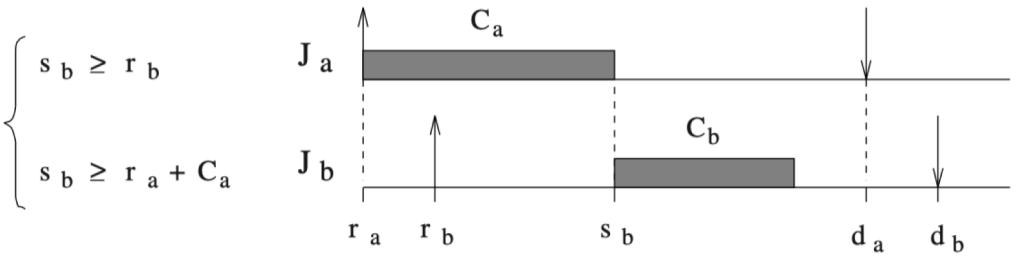
- The problem of scheduling a set of *n* tasks with precedence constraints and dynamic activations can be solved in polynomial time complexity only if tasks are preemptable.
- The basic idea of their approach is to transform a set of dependent tasks into a set of independent tasks by an adequate modification of timing parameters.
- Tasks are scheduled by the Earliest Deadline First (EDF).
- All release times and deadlines are modified so that each task cannot start before its predecessors and cannot preempt their successors.

Modification of the Release Times

$s_b \geq r_b$ (that is, J_b must start the execution not earlier than its release time);

$s_b \geq r_a + C_a$ (that is, J_b must start the execution not earlier than the minimum finishing time of J_a).

$$r_b^* = \max(r_b, r_a + C_a).$$



1. For any initial node of the precedence graph, set $r_i^* = r_i$.
2. Select a task J_i such that its release time has not been modified but the release times of all immediate predecessors J_h have been modified. If no such task exists, exit.
3. Set

$$r_i^* = \max[r_i,\ \max(r_h^* + C_h : J_h \rightarrow J_i)].$$

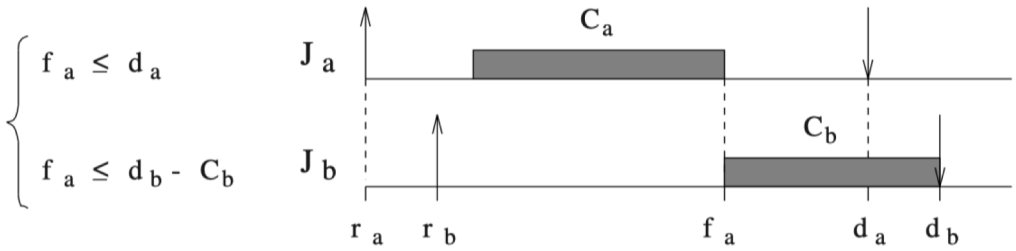
4. Return to step 2.

Modification of the Deadlines

$f_a \leq d_a$ (that is, J_a must finish the execution within its deadline);

$f_a \leq d_b - C_b$ (that is, J_a must finish the execution not later than the maximum start time of J_b).

$$d_a^* = \min(d_a, d_b - C_b).$$



1. For any terminal node of the precedence graph, set $d^*_i = d_i$.
2. Select a task J_i such that its deadline has not been modified but the deadlines of all immediate successors J_k have been modified. If no such task exists, exit.
3. Set $$d_i^* = \min[d_i,\ \min(d_k^* - C_k : J_i \rightarrow J_k)].$$
4. Return to step 2.

Example

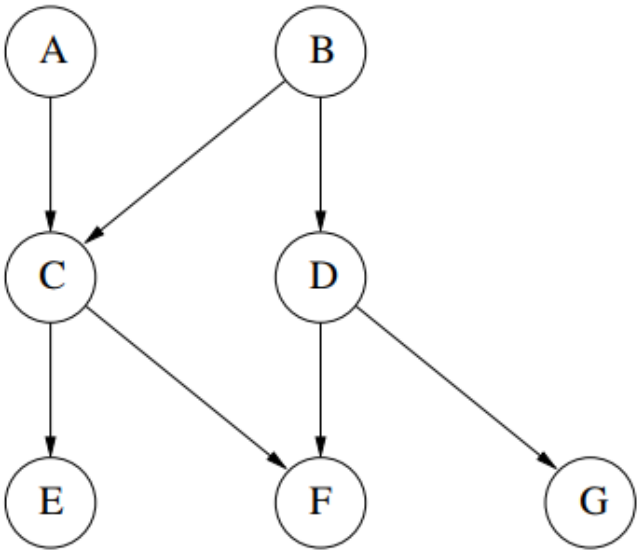
- Given seven tasks, *A*, *B*, *C*, *D*, *E*, *F*, and *G*, construct the precedence graph from the following precedence relations:

$$
\begin{array}{ll}
A \rightarrow C & \\
B \rightarrow C & B \rightarrow D \\
C \rightarrow E & C \rightarrow F \\
D \rightarrow F & D \rightarrow G
\end{array}
$$

  Then, assuming that all tasks arrive at time $t = 0$, have deadline D = 25, and computation times 2, 3, 3, 5, 1, 2, 5, respectively, modify their arrival times and deadlines to schedule them by EDF.

Solution

- The schedule produced by EDF will be *{B, A, D, C, E, F , G}*



	C_i	r_i	$r*_i$	d_i	$d*_i$
A	2	0	0	25	20
B	3	0	0	25	15
C	3	0	3	25	23
D	5	0	3	25	20
E	1	0	6	25	25
F	2	0	8	25	25
G	5	0	8	25	25

Scheduling Algorithms for Aperiodic Tasks

	sync. activation	preemptive async. activation	non-preemptive async. activation
independent	**EDD** (Jackson '55) $O(n \log n)$ Optimal	**EDF** (Horn '74) $O(n^2)$ Optimal	***Tree search*** (Bratley '71) $O(n\, n!)$ Optimal
precedence constraints	**LDF** (Lawler '73) $O(n^2)$ Optimal	**EDF** * (Chetto et al. '90) $O(n^2)$ Optimal	**Spring** (Stankovic & Ramamritham '87) $O(n^2)$ Heuristic

Summary

- Classifications of Scheduling Algorithms
- Jackson's Algorithm
- Earliest Due Date
- Earliest Deadline First
- Non-Preemptive Scheduling
- Scheduling with Precedence Constraints
  - Latest Deadline First