

# Lecture 8 – Real-time Scheduling

## CSE 456: Embedded Systems



# Basic Terms

---

## *Real-time systems*

- *Hard:* A real-time task is said to be hard, if missing its deadline may cause catastrophic consequences on the environment under control. Examples are sensory data acquisition, detection of critical conditions, actuator servoing.
- *Soft:* A real-time task is called soft, if meeting its deadline is desirable for performance reasons, but missing its deadline does not cause serious damage to the environment and does not jeopardize correct system behavior. Examples are command interpreter of the user interface, displaying messages on the screen.

# Schedule

---

Given a *set of tasks*  $J = \{J_1, J_2, \dots\}$ :

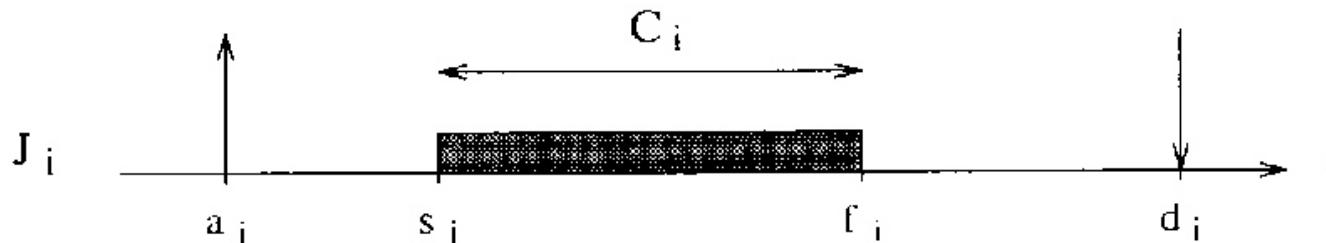
- A *schedule* is an assignment of tasks to the processor, such that each task is executed until completion.
- A schedule can be defined as an *integer step function*  $\sigma: R \rightarrow N$  where  $\sigma(t)$  denotes the task which is executed at time  $t$ . If  $\sigma(t) = 0$  then the processor is called idle.
- If  $\sigma(t)$  changes its value at some time, then the processor performs a *context switch*.
- Each interval, in which  $\sigma(t)$  is constant is called a *time slice*.
- A *preemptive schedule* is a schedule in which the running task can be arbitrarily suspended at any time, to assign the CPU to another task according to a predefined scheduling policy.

# Schedule and Timing

---

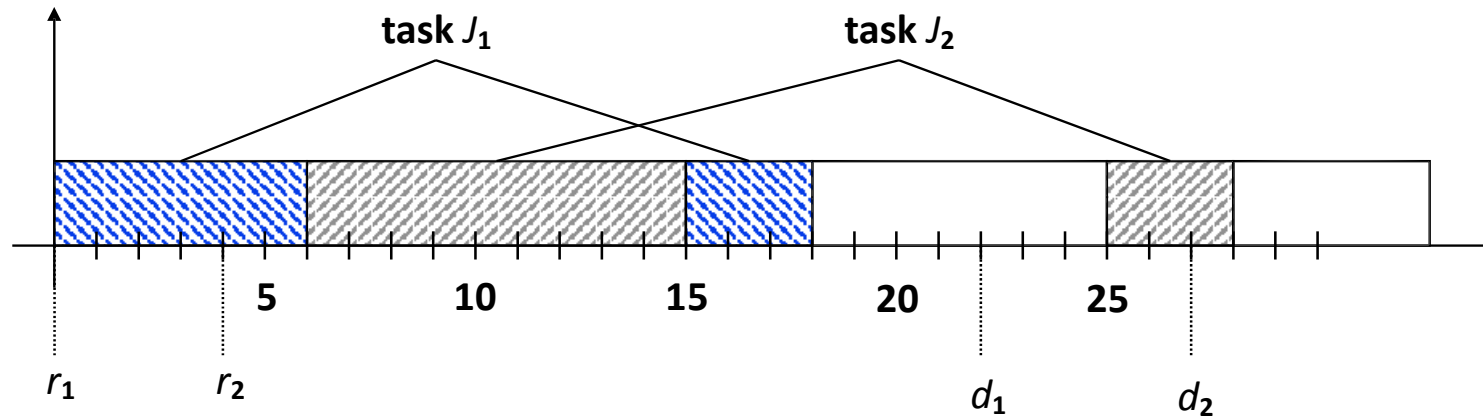
- A schedule is said to be *feasible*, if all task can be completed according to a set of specified constraints.
- A set of tasks is said to be *schedulable*, if there exists at least one algorithm that can produce a feasible schedule.
- *Arrival time*  $a_i$  or *release time*  $r_i$  is the time at which a task becomes ready for execution.
- *Computation time*  $C_i$  is the time necessary to the processor for executing the task without interruption.
- *Deadline*  $d_i$  is the time at which a task should be completed.
- *Start time*  $s_i$  is the time at which a task starts its execution.
- *Finishing time*  $f_i$  is the time at which a task finishes its execution.

# Schedule and Timing



- Using the above definitions, we have  $d_i \geq r_i + C_i$
- **Lateness**  $L_i = f_i - d_i$  represents the delay of a task completion with respect to its deadline; note that if a task completes before the deadline, its lateness is negative.
- **Tardiness or exceeding time**  $E_i = \max(0, L_i)$  is the time a task stays active after its deadline.
- **Laxity or slack time**  $X_i = d_i - a_i - C_i$  is the maximum time a task can be delayed on its activation to complete within its deadline.

# Example for Real-Time Model



Computation times:  $C_1 = 9$ ,  $C_2 = 12$

Start times:  $s_1 = 0$ ,  $s_2 = 6$

Finishing times:  $f_1 = 18$ ,  $f_2 = 28$

Lateness:  $L_1 = -4$ ,  $L_2 = 1$

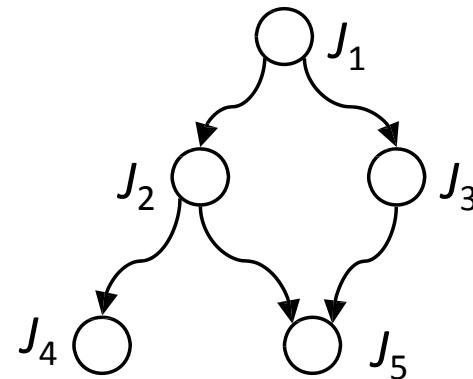
Tardiness:  $E_1 = 0$ ,  $E_2 = 1$

Laxity:  $X_1 = 13$ ,  $X_2 = 11$

# Precedence Constraints

---

- *Precedence relations* between tasks can be described through an *acyclic directed graph*  $G$  where tasks are represented by nodes and precedence relations by arrows.  $G$  induces a partial order on the task set.
- There are different *interpretations* possible:
  - All successors of a task are activated (*concurrent task execution*). We will use this interpretation in the lecture. *non-deterministic choice*
  - One successor of a task is activated: *deterministic choice*.



# Classification of Scheduling Algorithms

---

- With *preemptive algorithms*, the running task can be interrupted at any time to assign the processor to another active task, according to a predefined scheduling policy.
- With a *non-preemptive algorithm*, a task, once started, is executed by the processor until completion.
- *Static algorithms* are those in which scheduling decisions are based on fixed parameters, assigned to tasks before their activation.
- *Dynamic algorithms* are those in which scheduling decisions are based on dynamic parameters that may change during system execution.



# Classification of Scheduling Algorithms

---

- An algorithm is said *optimal* if it minimizes some given cost function defined over the task set.
- An algorithm is said to be *heuristic* if it tends toward but does not guarantee to find the optimal schedule.
- *Acceptance Test*: The runtime system decides whenever a task is added to the system, whether it can schedule the whole task set without deadline violations.

# Metrics to Compare Scheduling Algorithms

---

- Average response time:

$$\overline{t_r} = \frac{1}{n} \sum_{i=1}^n (f_i - r_i)$$

- Total completion time:

$$t_c = \max_i (f_i) - \min_i (r_i)$$

- Weighted sum of response time:

$$t_w = \frac{\sum_{i=1}^n w_i (f_i - r_i)}{\sum_{i=1}^n w_i}$$

- Maximum lateness:

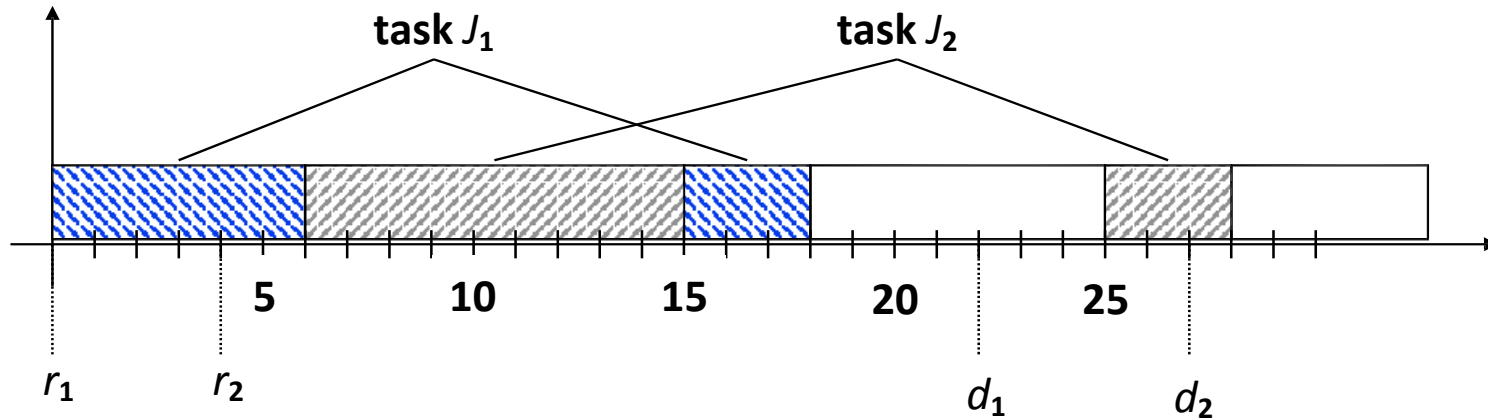
$$L_{\max} = \max_i (f_i - d_i)$$

- Number of late tasks:

$$N_{\text{late}} = \sum_{i=1}^n \text{miss}(f_i)$$

$$\text{miss}(f_i) = \begin{cases} 0 & \text{if } f_i \leq d_i \\ 1 & \text{otherwise} \end{cases}$$

# Metrics Example



Average response time:

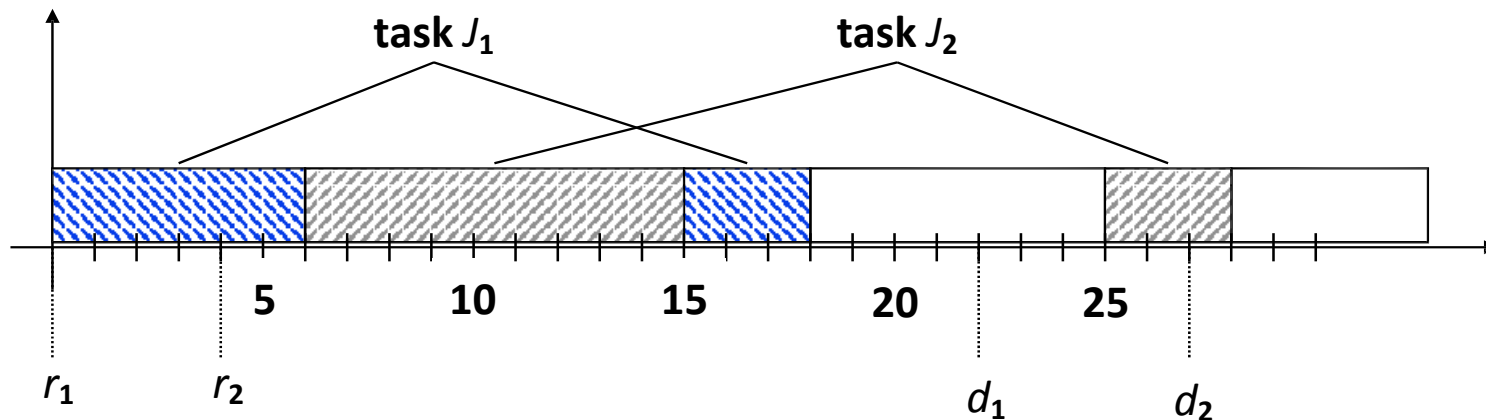
Total completion time:

Weighted sum of response times:

Number of late tasks:

Maximum lateness:

# Metrics Example



Average response time:

$$\bar{t}_r = \frac{1}{2} (18 + 24) = 21$$

Total completion time:

$$t_c = 28 - 0 = 28$$

Weighted sum of response times:  $w_1 = 2, w_2 = 1: t_w = \frac{2 \times 8 + 24}{3} = 20$

Number of late tasks:

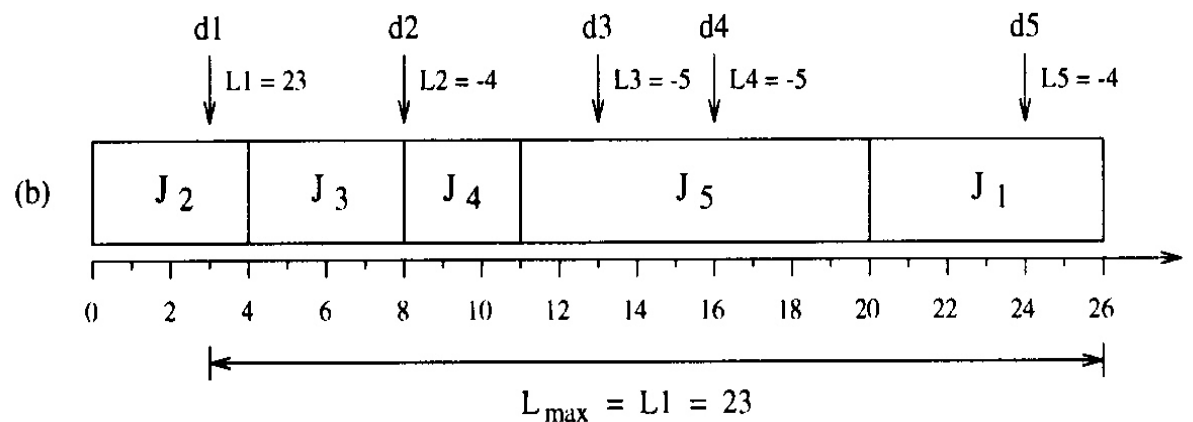
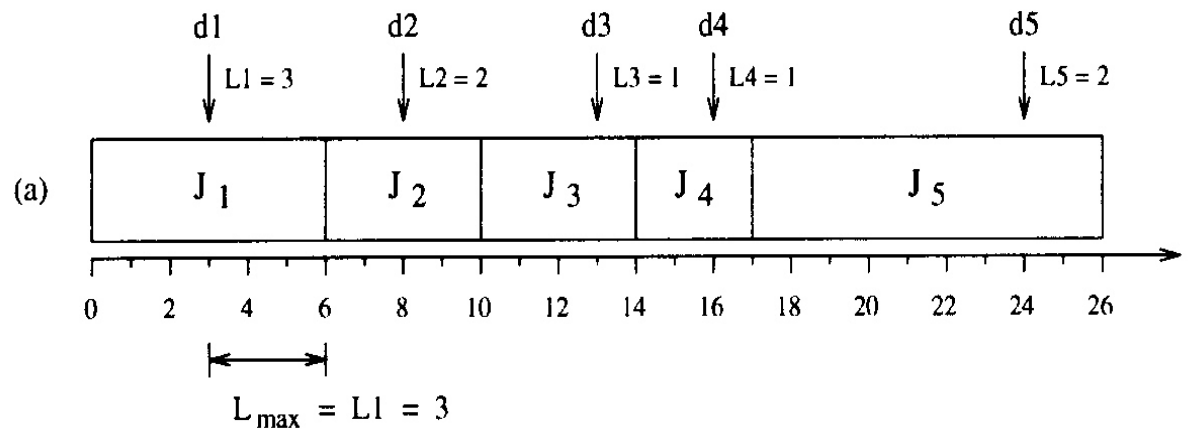
$$N_{\text{late}} = 1$$

Maximum lateness:

$$L_{\text{max}} = 1$$

# Metrics and Scheduling Example

In schedule (a), the *maximum lateness is minimized*, but all tasks miss their deadlines. In schedule (b), the maximal lateness is larger, but only one *task misses* its deadline.



# Real-Time Scheduling of Aperiodic Tasks

# Overview

---

Scheduling of *aperiodic tasks* with real-time constraints:

- Table with some known algorithms:

	Equal arrival times non preemptive	Arbitrary arrival times preemptive
Independent tasks	EDD (Jackson)	EDF (Horn)
Dependent tasks	LDF (Lawler)	EDF* (Chetto)

# Earliest Deadline Due (EDD)

---

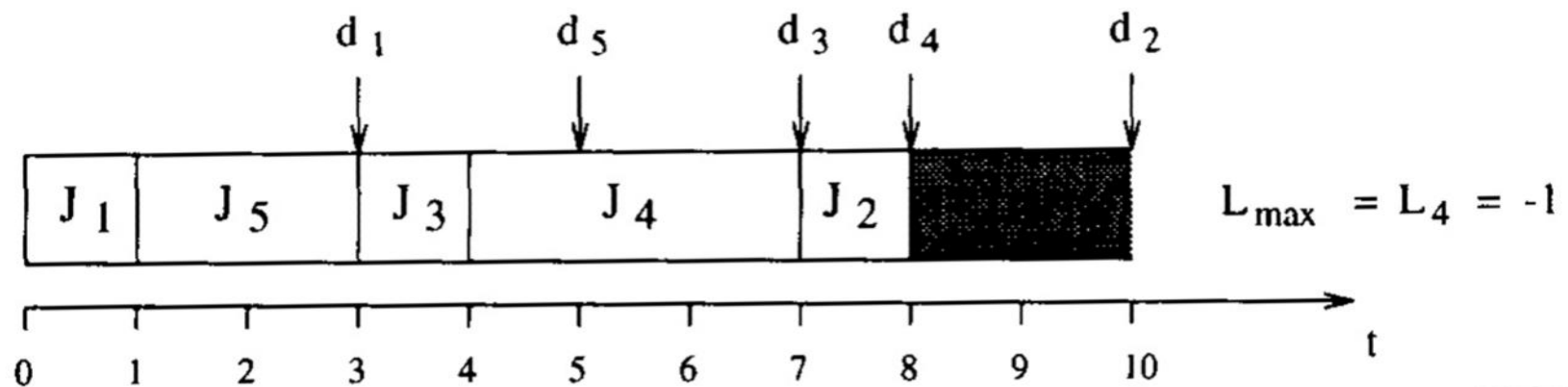
*Jackson's rule:* Given a set of  $n$  tasks. Processing in order of non-decreasing deadlines is optimal with respect to minimizing the maximum lateness.



# Earliest Deadline Due (EDD)

## Example 1:

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



# Earliest Deadline Due (EDD)

---

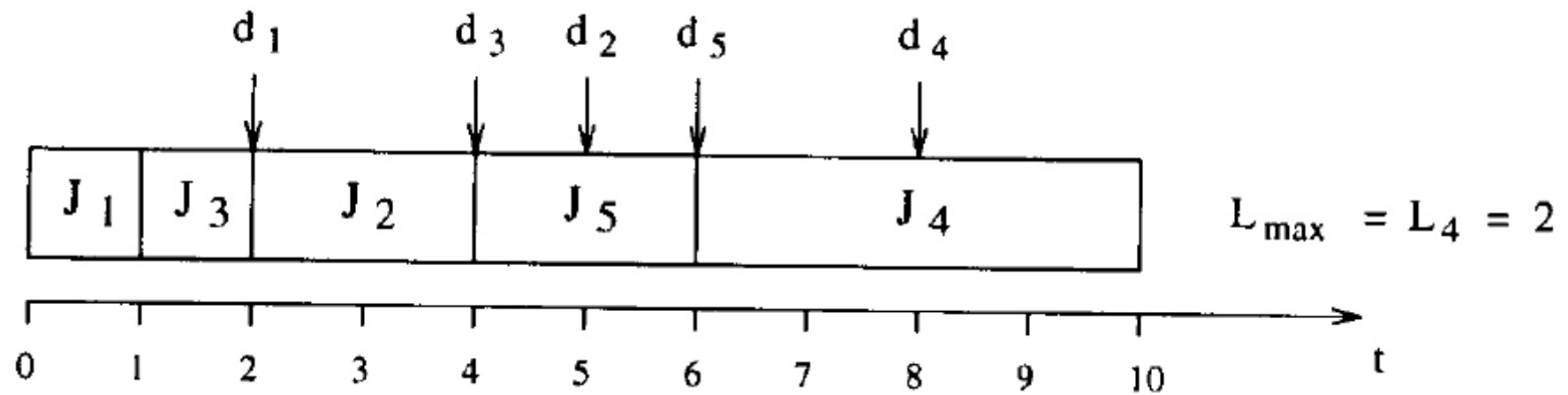
*Example 2:*

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

# Earliest Deadline Due (EDD)

## Example 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 Deadline First (EDF)

---

*Horn's rule:* 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 the ready tasks is optimal with respect to minimizing the maximum lateness.

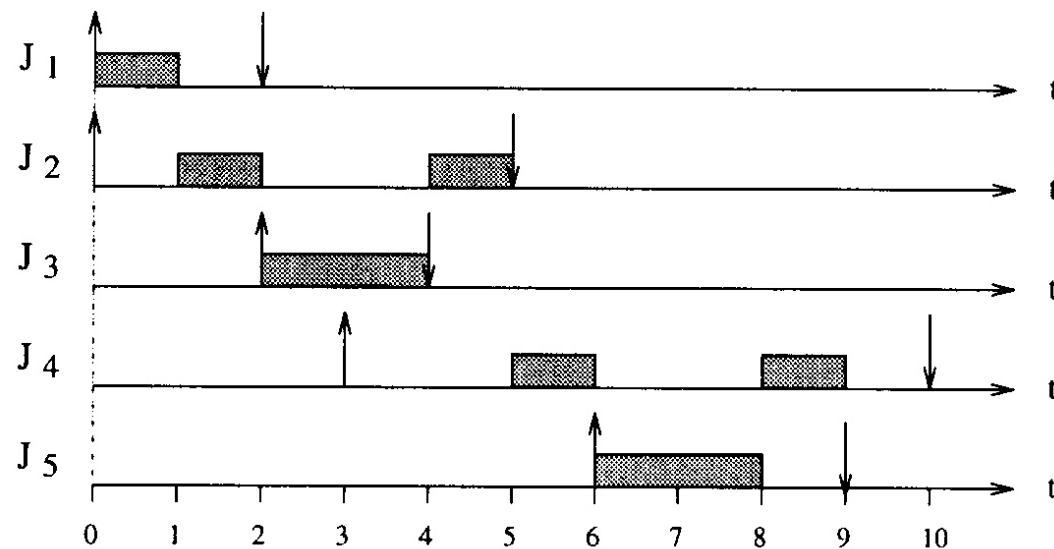
## EDF

At any instant execute the task with **the earliest absolute deadline** among all the **ready** tasks.

# Earliest Deadline First (EDF)

*Example:*

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



# Latest Deadline First (LDF)

---

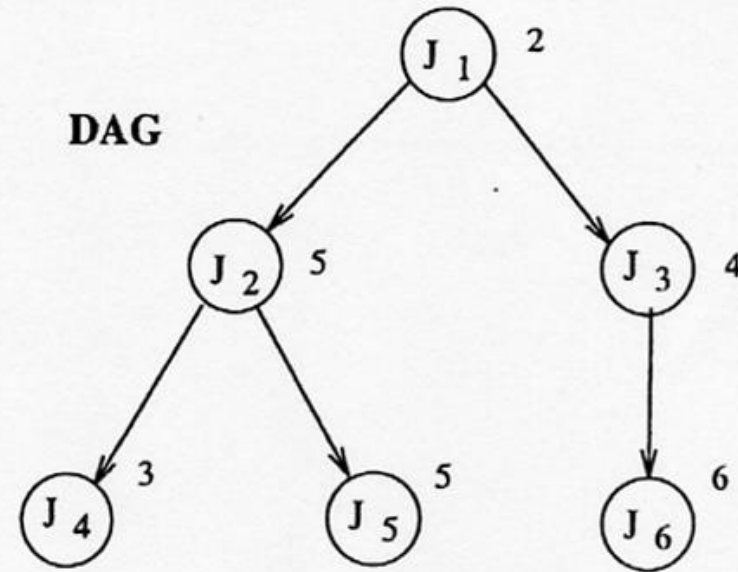
## Latest Deadline First Algorithm

### LDF

- among tasks without successors select the task with the latest deadline
- remove this task from the precedence graph and put it into a stack
- repeat until all tasks are in the stack
- the stack represents the order in which tasks should be scheduled

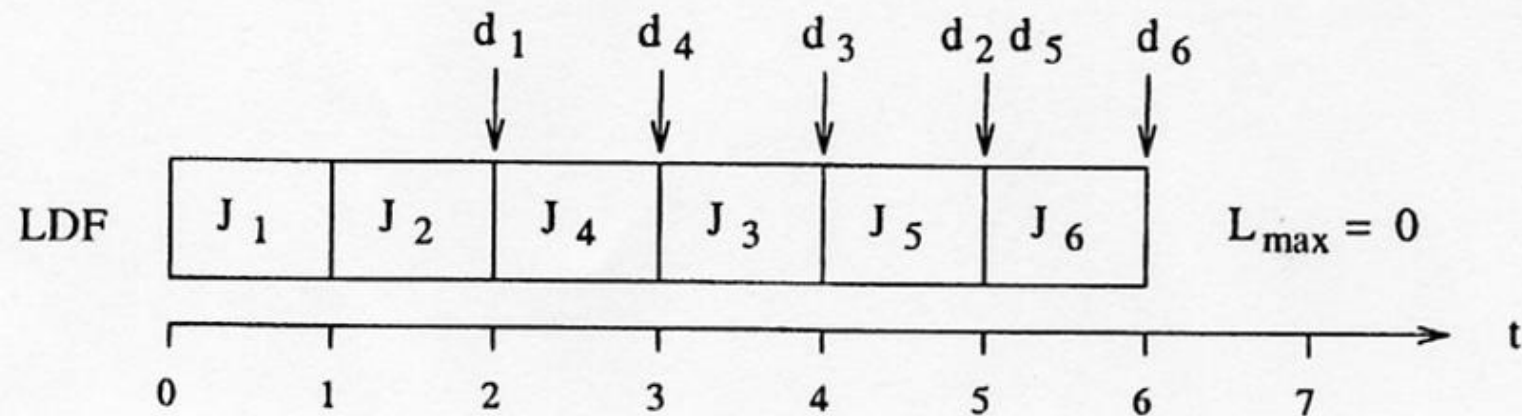
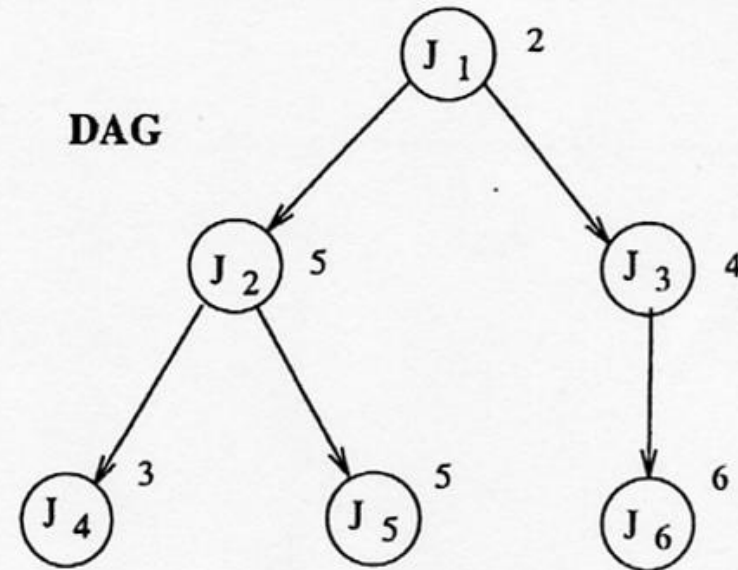
# Latest Deadline First (LDF)

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



# Latest Deadline First (LDF)

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





# Summary

---

- note: different interpretations
  - EDF algorithm = *earliest deadline* job is **scheduled** to run *first*
  - LDF algorithm = *latest deadline* job is **put into schedule** *first*