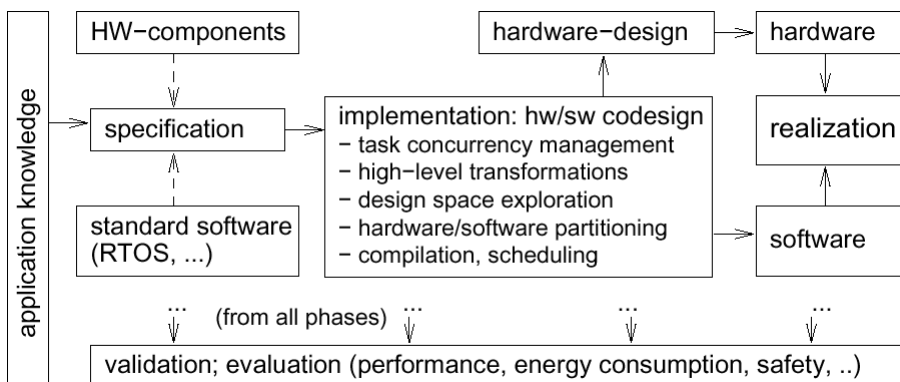




## Overview of embedded systems design



## Scheduling



**Scheduling: determine the order in which tasks are to be executed**

- **Task** ( $\approx$  process  $\approx$  thread): computation to be executed by the CPU in sequential fashion
- **Resources**: processor(s) (also: memory, disks, busses, communication channels, ...)
- Scheduler assigns **resources** to **tasks** for **durations of time**
- Other shared resources with exclusive access may complicate scheduling

## Point of departure: Scheduling general IT systems

- In general IT systems, not much is known about the computational processes a priori
  - The set of processes to be scheduled is open:
    - New software may be inserted into the running system
    - Software is run with “random” activation patterns
  - The power of schedulers thus is inherently limited by lack of knowledge  $\rightarrow$  only online scheduling is possible

## Scheduling processes in ES: The difference in process characterization

- Most ES are “closed shops”
  - Task set of the system is known
  - at least part of their activation pattern is known
    - **Periodic activation** in, e.g., signal processing
    - Maximum activation frequencies of asynchronous events determinable from environment dynamics  
→ **minimal inter-arrival times**
  - Possible to determine bounds on their execution time (**WCET**)  
(If they are well-built and we invest enough analysis effort)

→ Much better prospects for **guaranteeing response times!**

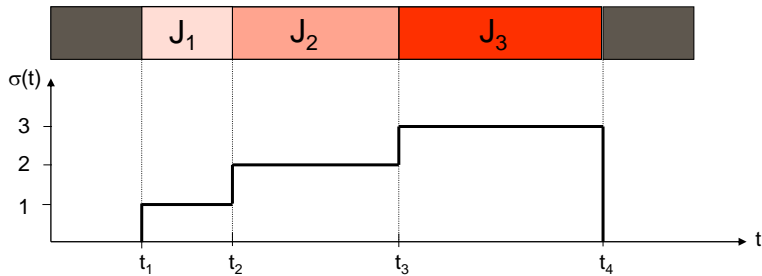
## Scheduling processes in ES: The difference in goals

- In **classical OS**, quality of scheduling is normally measured in terms of performance (throughput, reaction times) in the **average case**
- In **embedded real-time systems** the schedules often have to meet stringent quality criteria under all possible execution scenarios:
  - Tasks are often connected with **hard deadlines**, which must be met under all circumstances
  - Real-time systems have to be designed for **peak load**.  
Scheduling should work for all anticipated situations.

## Schedules

**Def.:** Given a set of tasks  $J = \{J_1, \dots, J_n\}$ ,  
 a **schedule** is a function  $\sigma : \mathbb{R}^+ \rightarrow \{0..n\}$  such that  
 $\forall t \in \mathbb{R}^+, \exists t_1, t_2 \in \mathbb{R}^+. t \in [t_1, t_2) \forall t' \in [t_1, t_2) \sigma(t) = \sigma(t')$ .

In other words:  $\sigma$  is an integer step function  
 $\sigma(t) = k$ , with  $k > 0$ , means that task  $J_k$  is executed at time  $t$ ,  
 $\sigma(t) = 0$  means that the CPU is idle.

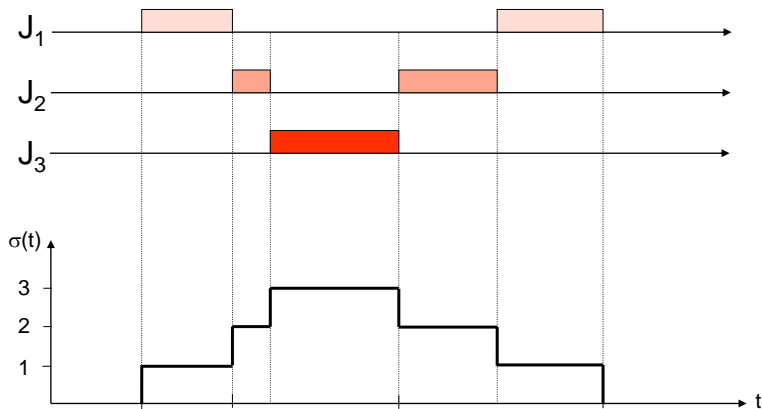


BF - ES

- 7 -

## Preemptive schedules

**Preemption:** the running task is interrupted

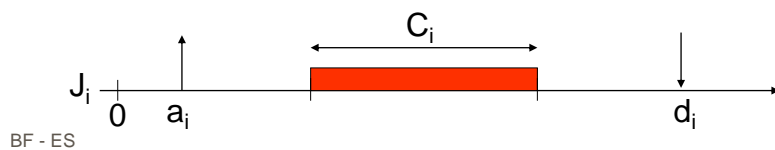


BF - ES

- 8 -

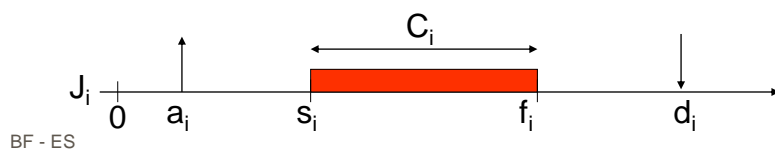
## Timing constraints of an aperiodic task $J_i$

- **Arrival time  $a_i$** : time at which task becomes ready for execution
- **Computation time  $C_i$** : time necessary to the processor for executing the task without interruption
- **Deadline  $d_i$** : time before which a task should be complete to avoid damage to the system
- **Slack time  $X_i$** :  $X_i = d_i - a_i - C_i$ , **maximum** time a task can be **delayed** on its activation to complete within its deadline



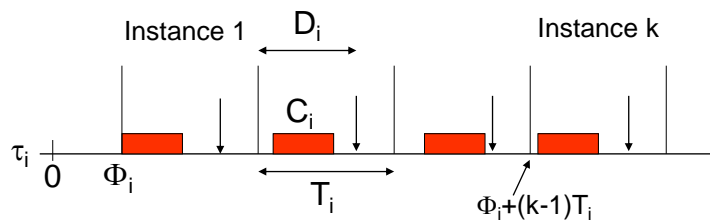
## Timing parameters of $J_i$ in schedule

- **Start time  $s_i$** : time at which a task starts its execution
- **Finishing time  $f_i$** : time at which task finishes its execution
- **Lateness  $L_i$** :  $L_i = f_i - d_i$ , delay of task completion with respect to deadline
- **Exceeding time  $E_i$** :  $E_i = \max(0, L_i)$



## Timing constraints of periodic task $\tau_i$

- **Phase  $\Phi_i$** : activation time of first periodic instance
- **Period  $T_i$** : time difference between two consecutive activations
- **Relative deadline  $D_i$** : time after activation time of an instance at which it should be complete



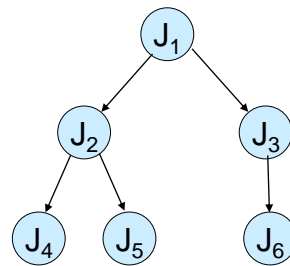
BF - ES

- 11 -

## Precedence constraints

Precedence constraints describe a partial order  $\langle$  in which the tasks can be executed:

$$\begin{aligned} J_1 &\langle J_2 & J_1 &\langle J_3 \\ J_2 &\langle J_4 & J_2 &\langle J_5 \\ J_3 &\langle J_6 \end{aligned}$$



$J_1 \langle J_2$  :  $J_1$  is a **predecessor** of  $J_2$   
 $\rightarrow J_1$  must be executed before  $J_2$

BF - ES

- 12 -

## Schedulability

A schedule is **feasible**, if all tasks can be completed according to a set of specified constraints.

A set of tasks is **schedulable** if there exists at least one feasible schedule.

**Optimal schedule:** Scheduling algorithms often aim at an optimal schedule with respect to a **cost function**

Example: Maximum lateness ( $\max_i L_i$ )

## Spectrum of scheduling algorithms

- **preemptive vs. non-preemptive:**  
Preemptive: tasks may be interrupted  
Non-preemptive: tasks always run to completion
- **static vs. dynamic:**  
Static scheduling: takes decisions at compile time  
Dynamic scheduling: takes decisions at runtime
- **uniprocessor vs. multiprocessor**
- **optimal vs. heuristic**

## Uniprocessor aperiodic task scheduling

### Aperiodic tasks with synchronous release

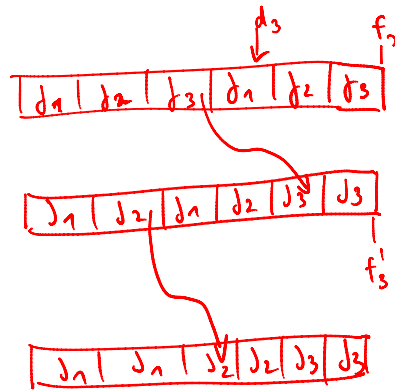
- A set of (aperiodic) tasks  $\{J_1, \dots, J_n\}$  with
  - arrival times  $a_i = 0 \forall 1 \leq i \leq n$ , i.e. “synchronous” arrival times
  - deadlines  $d_i$ ,
  - computation times  $C_i$
  - no precedence constraints, i.e. “independent tasks”
- non-preemptive
- single processor
- optimal
-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



## Preemption

- **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  $L_{\max}$ .



$$f_3' - d_3 = f_3 - d_3$$

-  $J_3$  has same lateness

- others may have smaller lateness

BF - ES

- 17 -

## Proof

- Consider a preemptive schedule with maximum lateness  $L_{\max}$ .
- If there is a task which is preempted, then choose the last point  $t$  of preemption. Let  $J_1$  be the task preempted in the schedule. Reshuffle the schedule by postponing all of  $J_1$ 's runtime allocated immediately before  $t$  s.t. that it happens immediately before the time  $t'$  of resumption of  $J_1$ , thus removing the preemption at  $t$ . This will not change the lateness of  $J_1$  and will at most reduce lateness of all other tasks, as those are unaffected or shuffled forward.
- Repeat this reshuffling until there is no further preemption.

BF - ES

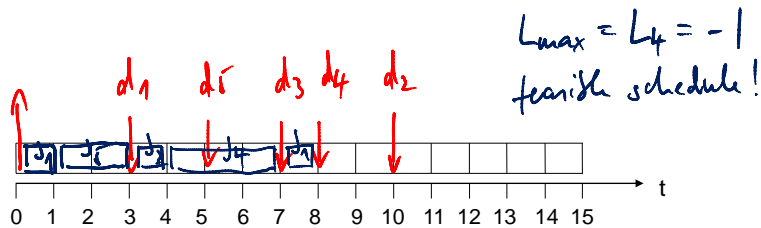
- 18 -

## EDD – Earliest Due Date

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

Example 1:

	J <sub>1</sub>	J <sub>2</sub>	J <sub>3</sub>	J <sub>4</sub>	J <sub>5</sub>
C <sub>i</sub>	1	1	1	3	2
d <sub>i</sub>	3	10	7	8	5



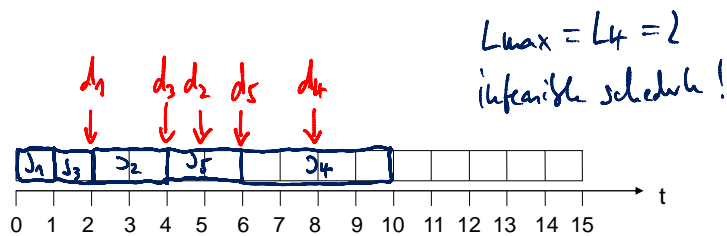
BF - ES

- 19 -

## EDD

Example 2:

	J <sub>1</sub>	J <sub>2</sub>	J <sub>3</sub>	J <sub>4</sub>	J <sub>5</sub>
C <sub>i</sub>	1	2	1	4	2
d <sub>i</sub>	2	5	4	8	6



BF - ES

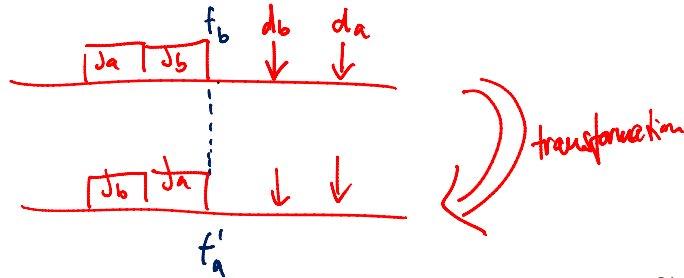
- 20 -

## EDD

- **Theorem (Jackson '55):**

Given a set of 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**.

- **Remark:** Minimizing maximum lateness includes finding a feasible schedule, if it exists. The reverse is not necessarily true.



BF - ES

- 21 -

$$L'_{\max} = \max_{a,b} \{ f'_a - d_a, f'_b - d_b \}$$

$$f'_a - d_a = f_b - d_a \leq f_b - d_b$$

$$f'_b - d_b \leq f_b - d_b$$

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

BF - ES

- 22 -

## EDD

- **Complexity of EDD scheduling:**
  - Sorting  $n$  tasks by increasing deadlines  
→  $O(n \log n)$
- **Test of Schedulability:**

If the conditions of the EDD algorithm are fulfilled, schedulability can be checked in the following way:

  - **Sort** task wrt. non-decreasing deadline.  
Let w.l.o.g.  $J_1, \dots, J_n$  be the sorted list.
  - **Check** whether in an EDD schedule  $f_i \leq d_i \forall i = 1, \dots, n$ .  
Since  $f_i = \sum_{k=1}^i C_k$ , we have to check
$$\forall i = 1, \dots, n \quad \sum_{k=1}^i C_k \leq d_i$$
  - Since EDD is optimal, non-schedulability by EDD implies non-schedulability in general.

BF - ES

- 23 -

## Aperiodic tasks with asynchronous release

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

BF - ES

- 24 -

## EDF – Earliest Deadline First

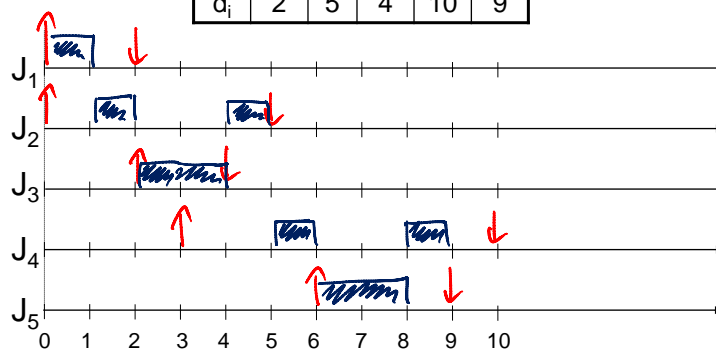
- At every instant, execute the task with the earliest deadline among all the ready tasks.
- Remark:
  1. If a new task arrives with an earlier deadline than the running task, the running task is immediately preempted.
  2. Here we assume that the time needed for context switches is negligible (we'll later see that this is unrealistic).

BF - ES

- 25 -

## EDF - Example

	J <sub>1</sub>	J <sub>2</sub>	J <sub>3</sub>	J <sub>4</sub>	J <sub>5</sub>
a <sub>i</sub>	0	0	2	3	6
C <sub>i</sub>	1	2	2	2	2
d <sub>i</sub>	2	5	4	10	9



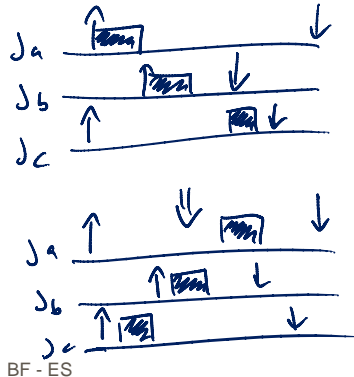
BF - ES

- 26 -

## EDF

- **Theorem (Horn '74):**

Given a set of independent tasks with arbitrary arrival times, any algorithm that at every instant executes the task with the earliest deadline among all the ready tasks is optimal with respect to minimizing the maximum lateness.



$$f_a' - d_a \leq f_c - d_c$$

$$f_c' - d_c \leq f_c - d_c$$

(same idea as for Jackson's Thm.)

BF - ES

- 27 -

## Non-preemptive version

- **Changed problem:**

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

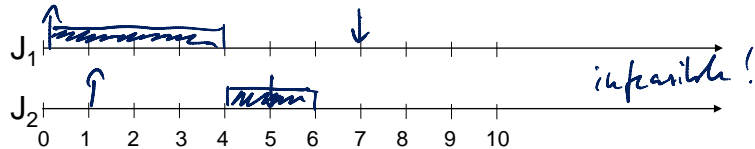
BF - ES

- 28 -

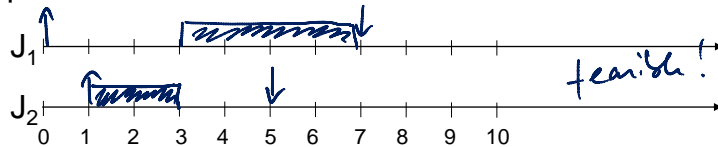
## Example

	$J_1$	$J_2$
$a_i$	0	1
$C_i$	4	2
$d_i$	7	5

- Non-preemptive EDF schedule:



- Optimal schedule:



BF - ES

- 29 -

## Example

- Observation:
  - In the optimal schedule the processor remains idle in interval  $[0,1)$  although task  $J_1$  is ready to execute.
- If arrival times are not known a-priori, then no on-line algorithm is able to decide whether to stay idle at time 0 or to execute  $J_1$ .
- Theorem** (Jeffay et al. '91): EDF is an optimal *non-idle* scheduling algorithm also in a *non-preemptive* task model.

BF - ES

- 30 -

## Non-preemptive scheduling: better schedules through introduction of idle times

- Assumptions:
  - Arrival times known a priori.
  - Non-preemptive scheduling
  - "Idle schedules" are allowed.
- Goal:
  - Find **feasible** schedule
- Problem is NP-hard.
- Possible approaches:
  - Heuristics
  - Branch-and-bound

BF - ES

- 31 -

## Bratley's algorithm

- Bratley's algorithm
  - Finds feasible schedule by branch-and-bound, if there exists one
  - Schedule derived from appropriate permutation of tasks  $J_1, \dots, J_n$
  - Starts with empty task list
  - Branches: Selection of next task (one not scheduled so far)
  - Bound:
    - Feasible schedule found at current path -> search path successful
    - There is some task not yet scheduled whose addition causes a missed deadline -> search path is blind alley

BF - ES

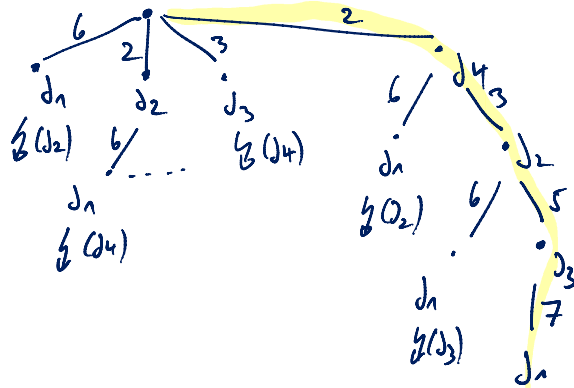
- 32 -



## Bratley's algorithm

- Example:

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



## Bratley's algorithm

- Due to exponential worst-case complexity only applicable as off-line algorithm.
- Resulting schedule stored in task activation list.
- At runtime: dispatcher simply extracts next task from activation list.

### Case 3: Scheduling with precedence constraints

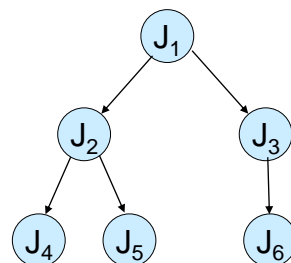
- Non-preemptive scheduling with non-synchronous arrival times, deadlines and precedence constraints is **NP-hard**.
- Here restriction: **synchronous arrival times** (all tasks arrive at 0)

BF - ES

- 35 -

### Example

	J <sub>1</sub>	J <sub>2</sub>	J <sub>3</sub>	J <sub>4</sub>	J <sub>5</sub>	J <sub>6</sub>
a <sub>i</sub>	0	0	0	0	0	0
C <sub>i</sub>	1	1	1	1	1	1
d <sub>i</sub>	2	5	4	3	5	6



BF - ES

- 36 -

## 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:

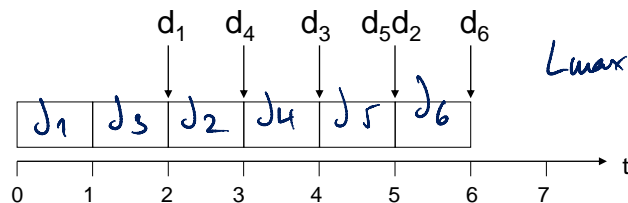
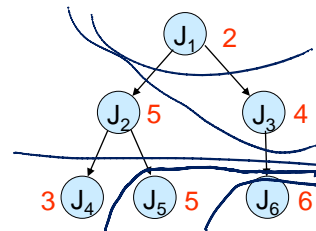
1. Among all **sinks** in the precedence graph select the task T with **latest** deadline. Schedule T **last**.
2. Remove T from G.
3. Repeat.

BF - ES

- 37 -

## Example (continued)

- Algorithm 1:

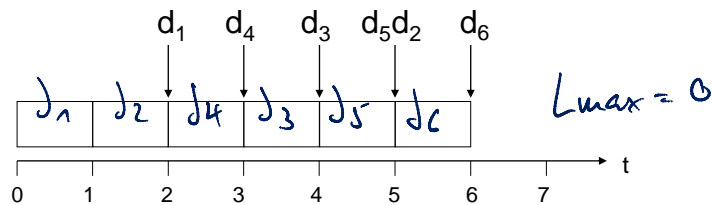
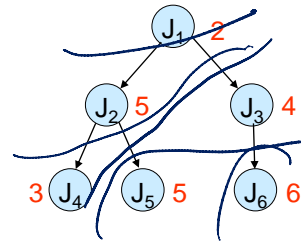


BF - ES

- 38 -

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