

# CMPE 117

Lectures:  
Aperiodic Task Scheduling

© Luca de Alfaro, 2002-3

## Scheduling: vocabulary

$a$ : arrival time

$C$ : computation time

$d$ : deadline

$s$ : starting time

$f$ : finishing time

$L = f - d$ : lateness

$E = \max(0, L)$ : tardiness

$X = d - a - C$ : slack

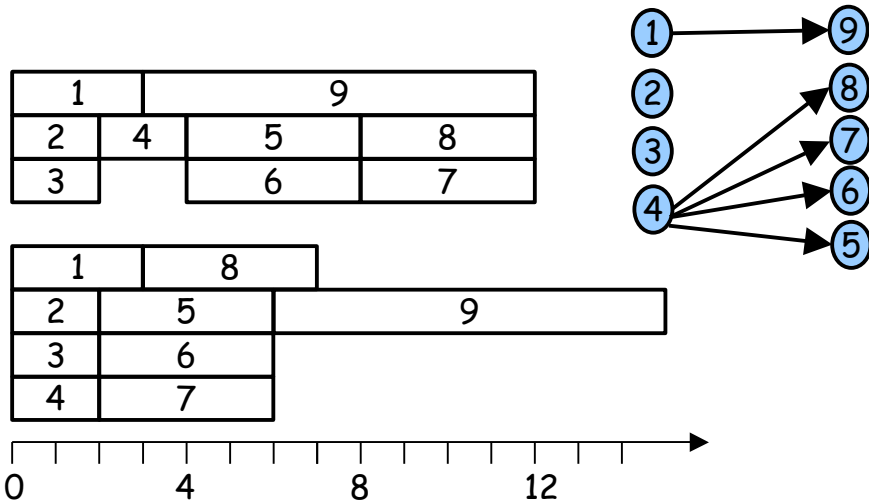
## Scheduling: types

- Preemptive vs. non-preemptive.
- Synchronous (all processes arrive at once) or asynchronous (processes arrive one by one)
- Static (a, C, d of all processes known in advance) or on-line.
- Independent, or with dependency relation.
- Aperiodic vs. periodic vs. hybrid (mixed)

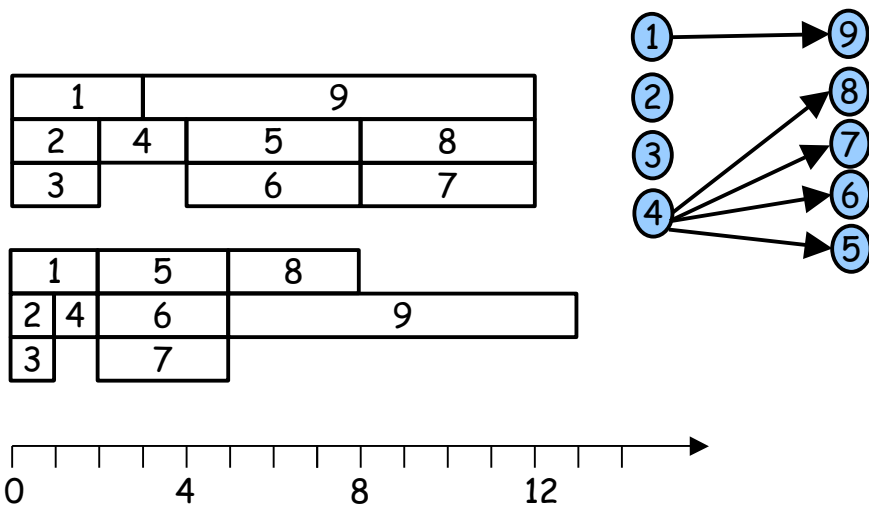
## Scheduling Anomalies

- Theorem [Graham, 1976]:  
If a task set is optimally scheduled on a multiprocessor with some priority assignment, fixed execution times, and precedence constraints, then the following can **increase** the schedule length:
  - **increasing** the number of processors
  - **reducing** execution times
  - **weakening** the precedence relation

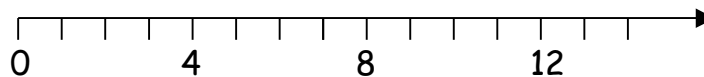
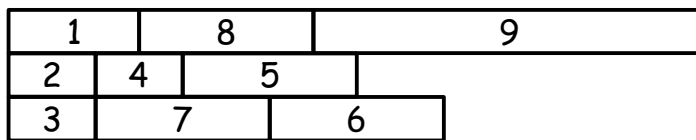
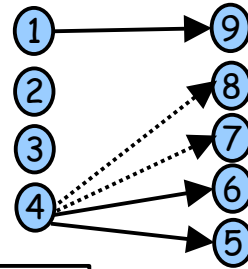
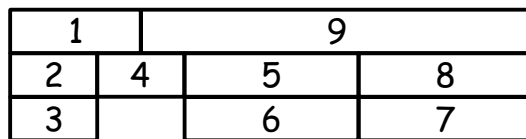
## Scheduling anomalies - example Adding one processor



## Scheduling anomalies - example Decreasing the running time by 1



## Scheduling anomalies - example Weakening the precedence relation



## Where's the catch?

- Theorem [Graham, 1976]:  
If a task set is optimally scheduled on a multiprocessor with some **priority assignment**, fixed execution times, and precedence constraints, then the following can increase the schedule length...
- Priorities are a bad idea! They confuse two concepts:
  - How important is something?
  - What should be scheduled next?

## Where's the catch?

- Theorem [Graham, 1976]:  
If a task set is **optimally scheduled** on a multiprocessor with some priority assignment, fixed execution times, and precedence constraints, then the following can increase the schedule length...
- Classical scheduling methods for multiprocessors may not be optimal, but optimality may be hard to achieve (NP-complete).

## Multiprocessor Scheduling is NP-complete

- INSTANCE: a set  $J$  of tasks, a number  $m$  of processors, a global deadline  $d$ , and for each  $j \in J$ , a computation time  $C_j$ .
- QUESTION: is there a partition
$$J = J_1 \cup J_2 \cup \dots \cup J_m$$
such that
$$\max_{1 \leq p \leq m} \sum_{j \in J_p} C_j \leq d ?$$

[Garey & Johnson: Computers and Intractability. A Guide to the Theory of NP-Completeness]

## Scheduling without Precedence Constraints

### Earliest Due Date (EDD) 1|synch|Lmax

Use when:

- Synch: all jobs have  $r=0$  (arrive immediately)
- Optimizes maximum lateness
- No precedence relations
- No preemption necessary

## Earliest Due Date (EDD)

Given a set of processes

$\{[C_1, d_1], [C_2, d_2], \dots, [C_m, d_m]\}$ ,

in order to minimize  $L_{max}$  it suffices to schedule the processes in order of increasing deadlines.

- **EDD Algorithm:**

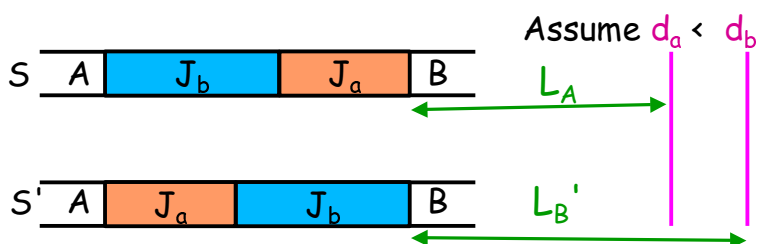
- 1) Sort the deadlines in increasing order

- 2) Schedule the processes in order of increasing deadline.

- **Schedulability criterion:**

Once ordered so that  $d_1 < d_2 < \dots < d_m$ , check that  $\sum_{i=1}^k C_i < d_k$  for all  $k \in \{1, \dots, m\}$ .

## EDD Optimality



$$\begin{array}{ll} L'_a < L_a & L'_A = L_A \\ L'_b < L_a & L'_A = L_A \end{array}$$

Hence

$$L' = \max \{L'_a, L'_b, L'_A, L'_B\} < L = \max \{L_a, L_b, L_A, L_B\}$$

## Earliest Deadline First (EDF)

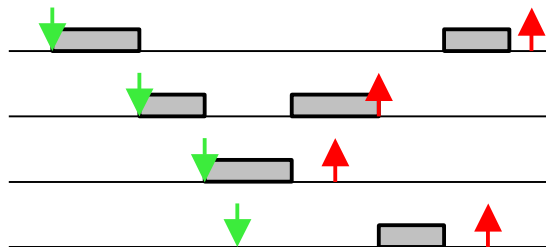
$1|asynch|L_{max}$

Use when:

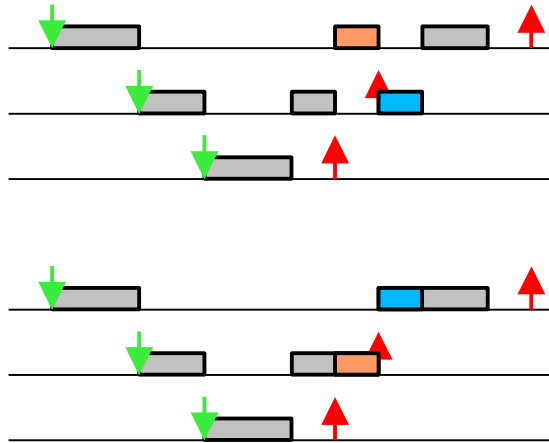
- Tasks can have arbitrary arrival time.
- They are preemptable. This is important!

## EDF Algorithm

[Horn]: to minimize  $L_{max}$ , always execute the process that has the earliest deadline



## Proof of EDF Optimality

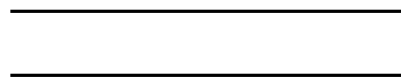
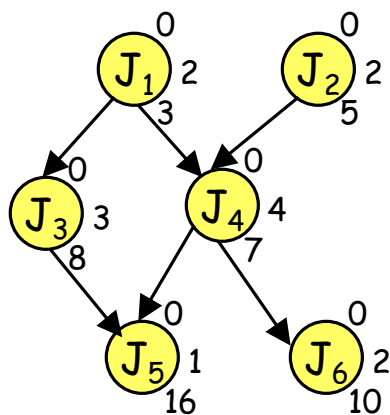
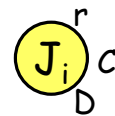


## Scheduling with Precedence Constraints

## Scheduling with precedence constraints (1 | prec,syn | Lmax): LDF (Latest Deadline First)

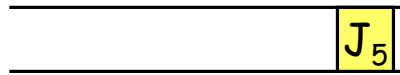
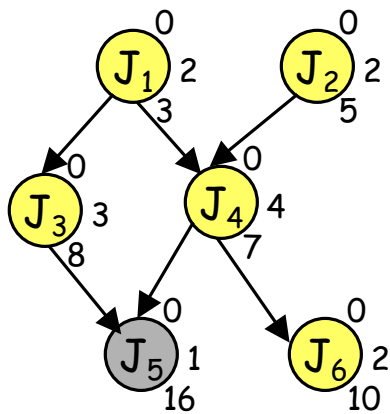
- Build the schedule from last to first, by recursively:
  - Let  $Q$  be the subset of jobs all whose descendants have been scheduled.
  - Pick the job  $p$  from  $Q$  that has the latest deadline.
  - Add  $p$  to the front of the schedule.
- Should be really called LDL (Latest deadline last).
- Optimal wrt.  $L_{max}$ .

### LDF Example



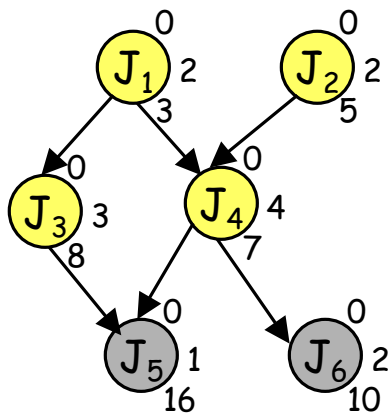
# LDF Example

$$J_i^r c_D$$



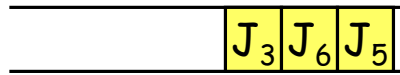
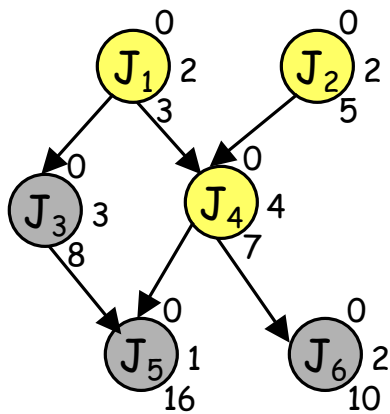
# LDF Example

$$J_i^r c_D$$



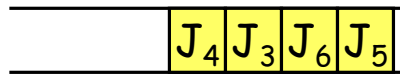
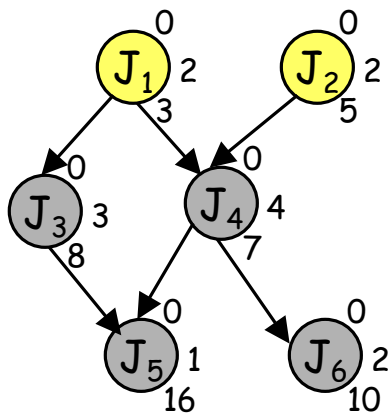
# LDF Example

$$J_i^r \begin{matrix} c \\ D \end{matrix}$$

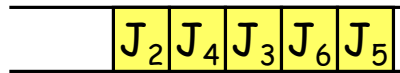
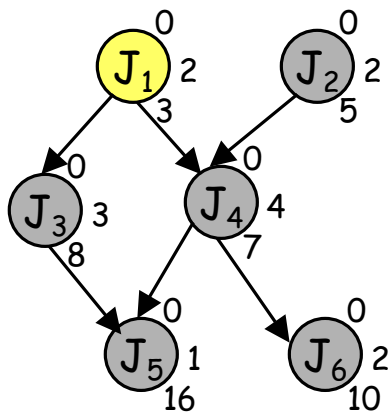
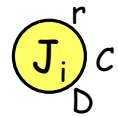


# LDF Example

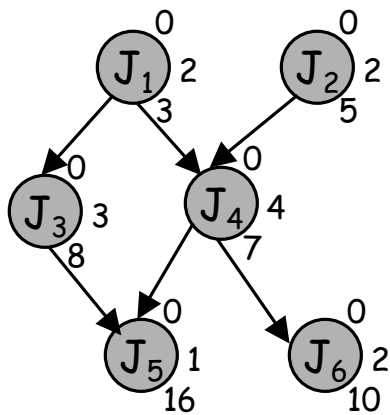
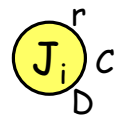
$$J_i^r \begin{matrix} c \\ D \end{matrix}$$



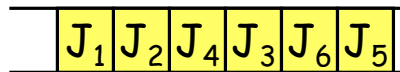
# LDF Example



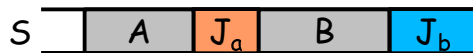
# LDF Example



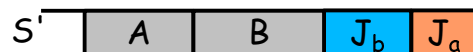
Schedule:



## LDF: Proof of Optimality



$J_a$  should be last according to LDF



$$L_{\max} = \max \{L_A, L_B, L_a, L_b\}$$

$L_A$  unchanged

$L'_B \leq L_B$  as B starts earlier

$L'_b \leq L_b$  as  $J_b$  starts earlier

$L'_a = f'_a - d_a \leq f_b - d_b$  (as  $d_a > d_b$ ) =  $L_b$

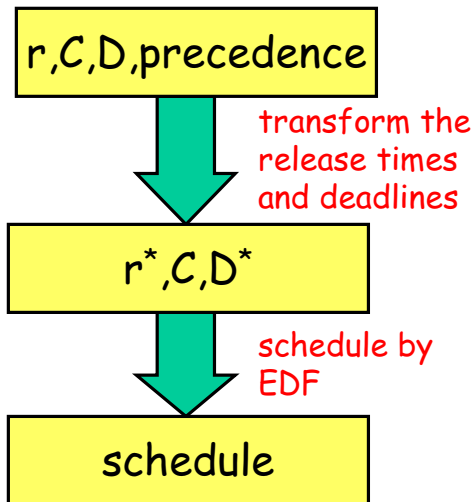
So  $L'_{\max} \leq L_{\max}$ .

The argument is concluded by induction.

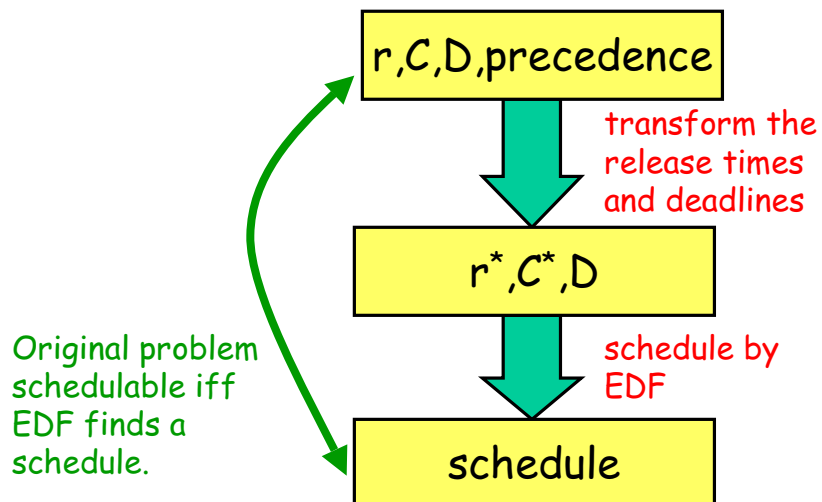
## EDF with precedence constraints (1 | async, prec, preem | Lmax)

- Algorithm by Chetto, Silly, Bouchenouf (1990).
- Idea: transform timing constraints to enforce precedence (cannot start before predecessors, cannot pre-empt successors), then use EDF.

## EDF with precedence constraints ( 1 | async, prec, preem | Lmax)



## EDF with precedence constraints ( 1 | async, prec, preem | Lmax)



## EDF with precedence constraints ( 1 | async, prec, preem | Lmax)

- When  $J_a \prec J_b$ , then:
  - $s_b \geq r_b$  ( $J_b$  must start after being released)
  - $s_b \geq r_a + C_a$  (so  $J_a$  has time to finish)
- Thus, replace  $r_b$  by  $r_b^* = \max\{r_b, r_a + C_a\}$

## EDF with precedence constraints ( 1 | async, prec, preem | Lmax)

- Algorithm for release times:
  - For all roots  $J_a$  of dependency graph, let  $r_a^* = r_a$ .
  - Select a task  $J_b$  such that all predecessors of  $J_b$  have already been modified; let  $H$  be this set.
  - Let  $r_b^* = \max\{r_b, \max_{c \in H} (r_c^* + C_c)\}$
- And recur.

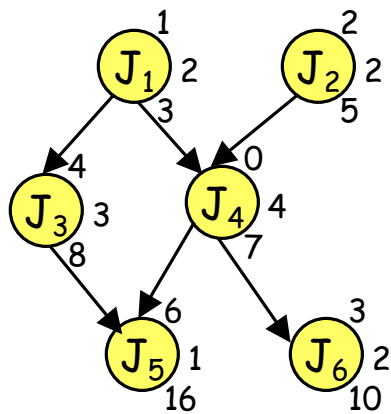
## EDF: Modification of the Deadlines

- Given  $J_a$  and  $J_b$  s.t.  $J_a \prec J_b$ , then the following conditions must hold:
  - $f_a \leq d_a$  (or else we miss the deadline)
  - $f_a \leq d_b - C_b$  (or there won't be enough time for  $a$  to finish).
- Hence, replace  $d_a$  by  $d_a^* = \min\{d_a, d_b - C_b\}$

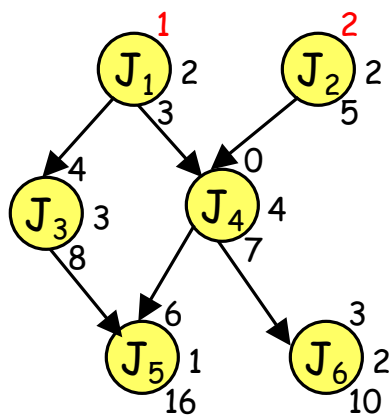
## EDF: Modification of the Deadlines

- Algorithm for deadline modification:
  - For all sinks  $J_a$  of dependency graph, let  $d_a^* = d_a$ .
  - Select a task  $J_b$  such that all successors of  $J_b$  have already been modified; let  $H$  be this set.
  - Let  $d_b^* = \min\{d_b, \min_{c \in H} (d_c^* - C_c)\}$
- And recur.

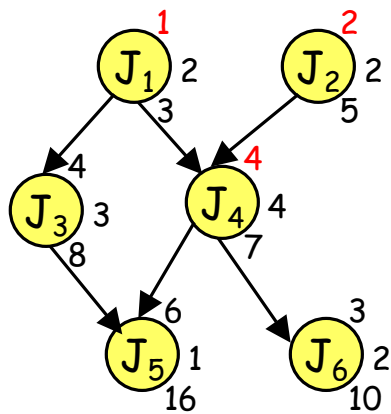
## EDF Modification Example



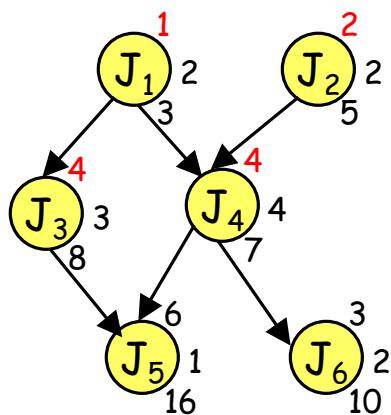
## EDF Modification Example



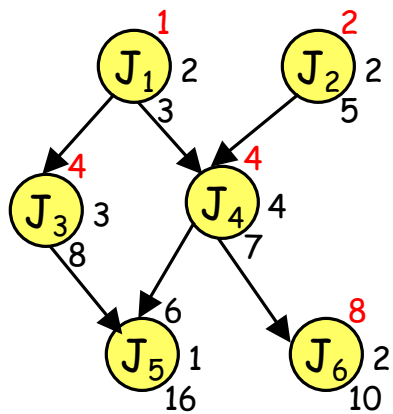
## EDF Modification Example



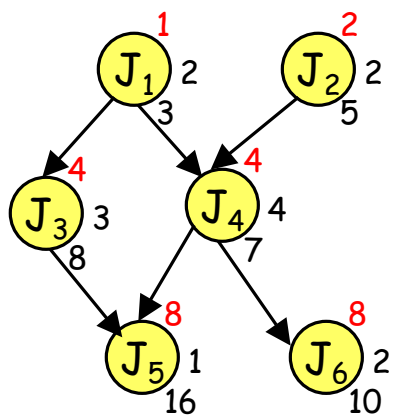
## EDF Modification Example



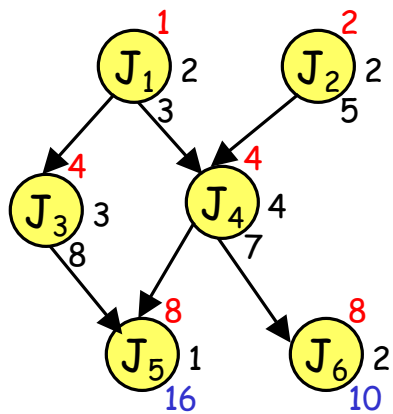
## EDF Modification Example



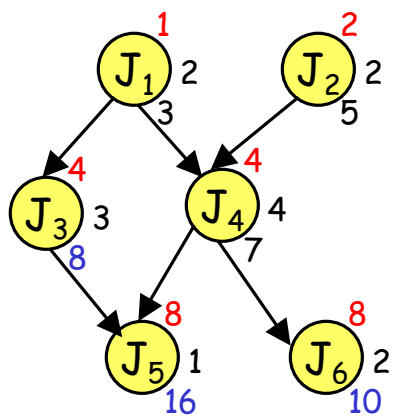
## EDF Modification Example



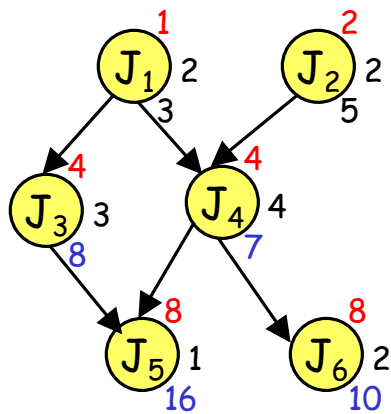
## EDF Modification Example



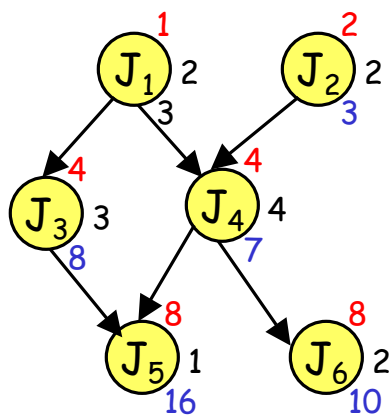
## EDF Modification Example



## EDF Modification Example

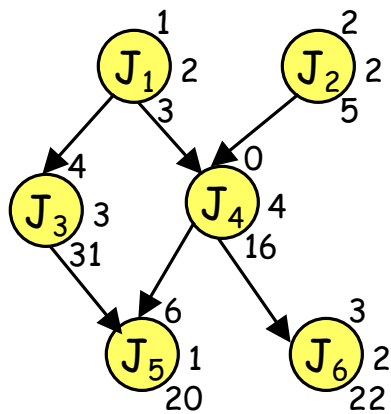


## EDF Modification Example

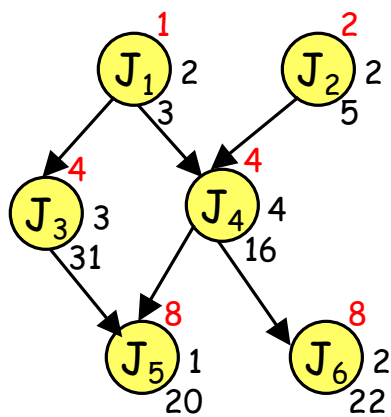


Sometimes we can see right away that the problem is not schedulable

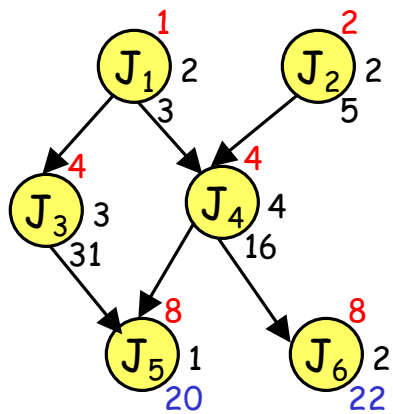
## Another Example



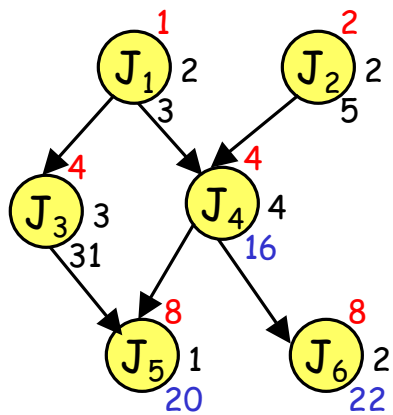
## Another Example



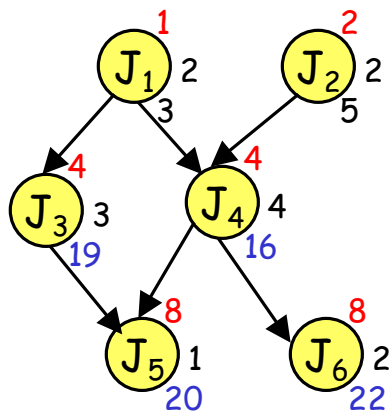
## Another Example



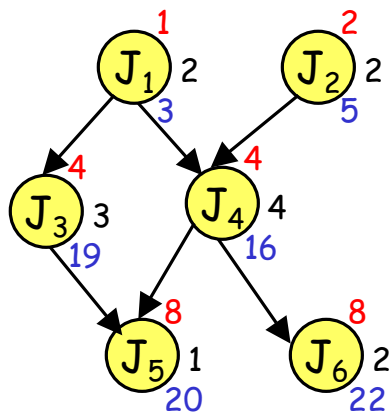
## Another Example



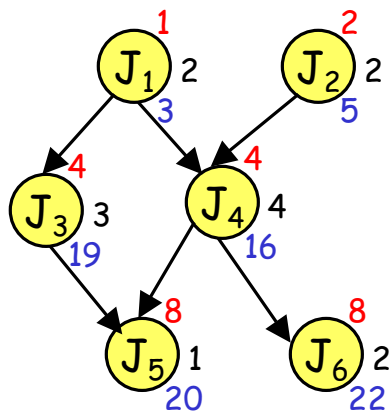
## Another Example



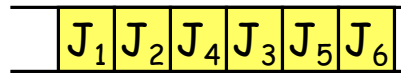
## Another Example



## Another Example



Schedule:



Schedulable.

## EDF + prec: Proof of optimality

- The algorithm provides a translation EDF+Prec  $\rightarrow$  modified-EDF.
- We can show that the translation preserves schedulability.
- First direction: if EDF+Prec is schedulable, then modified-EDF is also schedulable (the same schedule works).

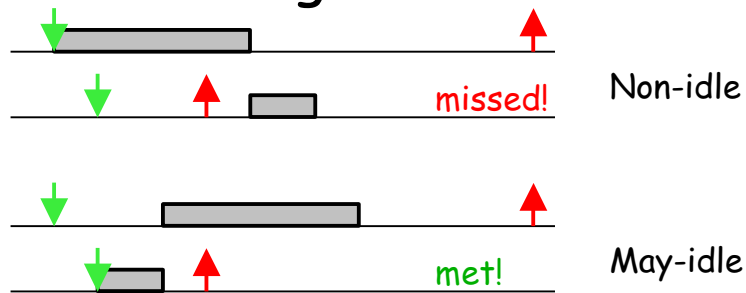
## EDF + prec: Proof of optimality

- Converse: if modified-EDF is schedulable, the same schedule shows that EDF+Prec is schedulable.
- Clearly, the schedule produced by modified EDF meets all release times and deadlines of the original problem; we must only prove that the precedence relation is respected.
- Assume  $J_a \prec J_b$ , and  $C_b > 0$ . Then,  $D_a^* < D_b^*$ , and  $r_a^* < r_b^*$ , so EDF schedules  $J_a$  before  $J_b$ .

## Non-Preemptive Scheduling

- Jobs have release times, arrive dynamically, and there are no precedence relations.
- There are two kinds of algorithms:
  - Non-idle, that always execute a job as long as there is one ready
  - May-Idle, that may choose to do nothing even though there are jobs ready.
- There are cases when may-idle algorithms outperform non-idle ones....

## Being idle is sometimes a good idea



But: unless the arrival times are known, are may-idle algorithms reasonable? They require knowledge about the future!

## EDF is best of non-idle $1|async|L_{max}$ algorithms

- This is a theorem by Jeffay, Stanat, and Martel (1991).
- Intuition: if you can't help being active, at least do the most useful thing at hand, that is, the one with the earliest deadline.

## Aperiodic scheduling: summary

	sync	preempt async	non-preempt async
independent	EDD	EDF	if non-idle, EDF
prec constr.	LDF	EDF with modificati ons	Search