Cyber-Physical Systems

Deadline based Scheduling

UNIVERSITY AT ALBANY
State University of New York

ICEN 553/453– Fall 2021

Prof. Dola Saha

Real-Time Systems

- The operating system, and in particular the scheduler, is perhaps the most important component

Examples:

- Control of laboratory experiments
- Process control in industrial plants
- Robotics
- Air traffic control
- Telecommunications
- Military command and control systems

- Correctness of the system depends not only on the logical result of the computation but also on the time at which the results are produced
- Tasks attempt to react to events that take place in the outside world
- These events occur in "real time" and tasks must be able to keep up with them

Hard and Soft Real-Time Tasks

- Hard
  - One that must meet its deadline
  - Otherwise it will cause unacceptable damage or a fatal error to the system
- Soft
  - Has an associated deadline that is desirable but not mandatory
  - It still makes sense to schedule and complete the task even if it has passed its deadline

Periodic and Aperiodic Tasks

- **Periodic tasks**
  - Requirement may be stated as:
    - Once per period T
    - Exactly T units apart
- **Aperiodic tasks**
  - Has a deadline by which it must finish or start
  - May have a constraint on both start and finish time

Characteristics of Real Time Systems

Real-time operating systems have requirements in five general areas:

- Determinism
- Responsiveness
- User control
- Reliability
- Fail-soft operation

Determinism

- Concerned with how long an operating system delays before acknowledging an interrupt
- Operations are performed at fixed, predetermined times or within predetermined time intervals
  - When multiple processes are competing for resources and processor time, no system will be fully deterministic

The extent to which an operating system can deterministically satisfy requests depends on:

- The speed with which it can respond to interrupts
- Whether the system has sufficient capacity to handle all requests within the required time

Responsiveness

- Together with determinism make up the response time to external events
  - Critical for real-time systems that must meet timing requirements imposed by individuals, devices, and data flows external to the system
- Concerned with how long, after acknowledgment, it takes an operating system to service the interrupt

Responsiveness includes:

- Amount of time required to initially handle the interrupt and begin execution of the interrupt service routine
- Amount of time required to perform the ISR
- Effect of interrupt nesting

User Control

- Generally much broader in a real-time operating system than in ordinary operating systems
- It is essential to allow the user fine-grained control over task priority
- User should be able to distinguish between hard and soft tasks and to specify relative priorities within each class
- May allow user to specify such characteristics as:

Paging or process swapping → What processes must always be resident in main memory → What disk transfer algorithms are to be used → What rights the processes in various priority bands have

Reliability

- More important for real-time systems than non-real time systems
- Real-time systems respond to and control events in real time so loss or degradation of performance may have catastrophic consequences such as:
  - Financial loss
  - Major equipment damage
  - Loss of life

Fail-Soft Operation

- A characteristic that refers to the ability of a system to fail in such a way as to preserve as much capability and data as possible
- Important aspect is stability
  - A real-time system is stable if the system will meet the deadlines of its most critical, highest-priority tasks even if some less critical task deadlines are not always met

Features common to Most RTOSs

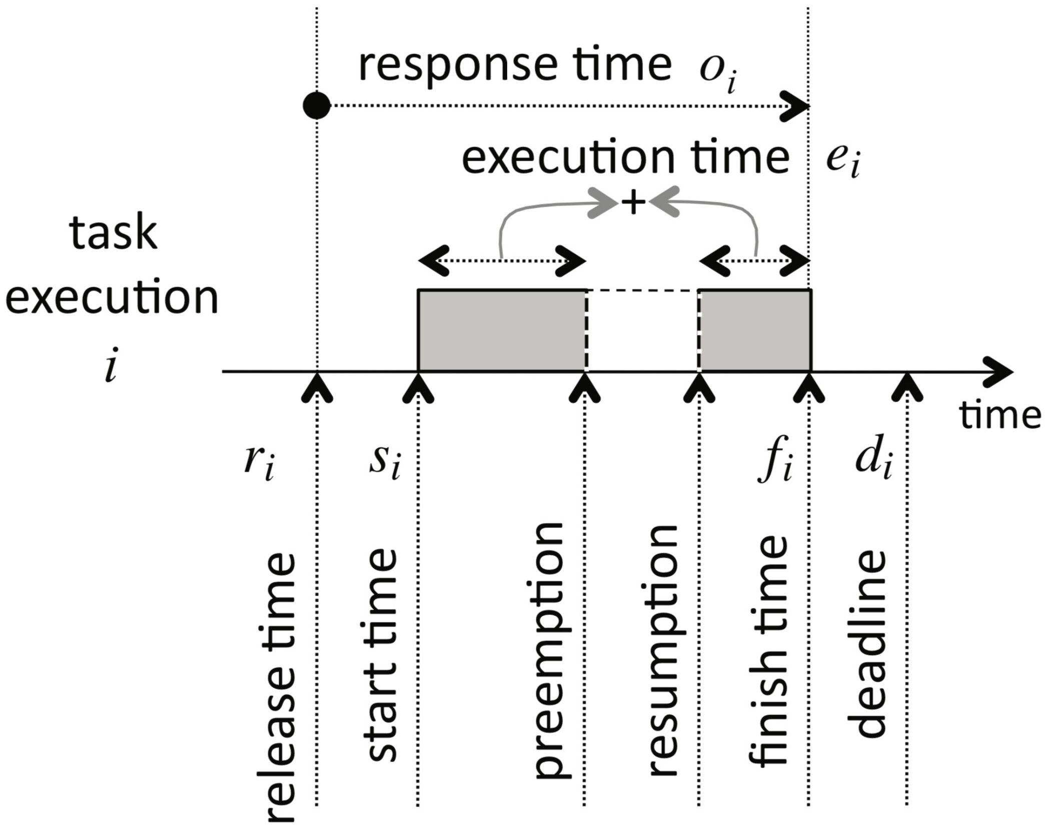
- A stricter use of priorities than in an ordinary OS, with preemptive scheduling that is designed to meet real-time requirements
- Interrupt latency is bounded and relatively short
- More precise and predictable timing characteristics than general purpose OSs

Task Model

$$s_i \geq r_i$$

$$f_i \geq s_i$$

$$o_i = f_i - r_i$$



Scheduling Strategies

- Goal: all task executions meet their deadlines

$$f_i \leq d_i$$

- A schedule that accomplishes this is called a feasible schedule.
- A scheduler that yields a feasible schedule for any task set is said to be optimal with respect to feasibility.

Criteria or Metrices

- Processor Utilization μ
- Maximum Lateness

$$L_{\max} = \max_{i \in T}(f_i - d_i)$$

- Total Completion Time or Makespan

$$M = \max_{i \in T} f_i - \min_{i \in T} r_i$$

- Average Response Time

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

Rate Monotonic Scheduling

- Simple process model: n tasks invoked periodically with:
  - periods T1, … ,Tn, which equal the deadlines
  - known worst-case execution times (WCET) C1, … ,Cn
    - no mutexes, semaphores, or blocking I/O
  - independent tasks, no precedence constraints
  - fixed priorities
  - preemptive scheduling
- Rate Monotonic Scheduling (RMS): priorities ordered by period (smallest period has the highest priority)

Feasibility for RMS

- Feasibility is defined for RMS to mean that every task executes to completion once within its designated period.

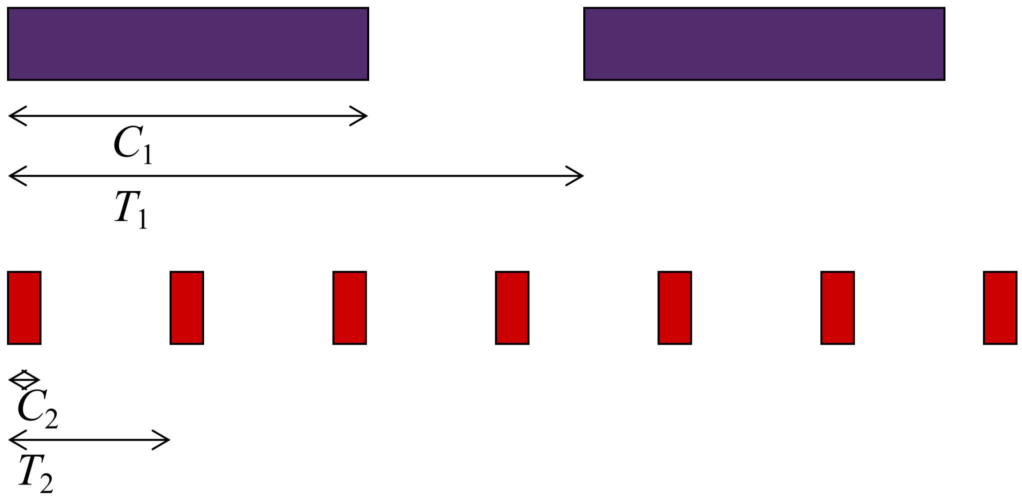
- Theorem: Under the simple process model, if any priority assignment yields a feasible schedule, then RMS also yields a feasible schedule.

- RMS is optimal in the sense of feasibility.

Liu and Layland, "Scheduling algorithms for multiprogramming in a hard-real-time environment," J. ACM, 1973.

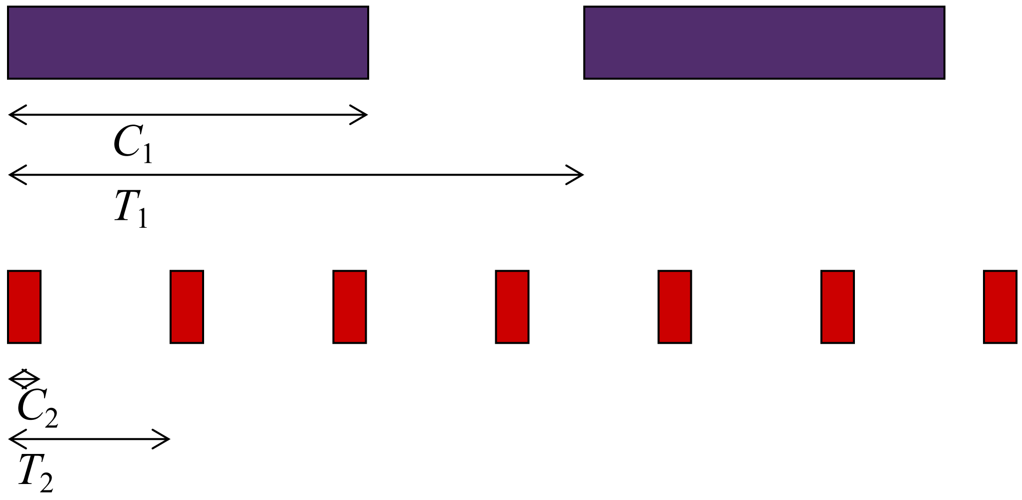
Showing Optimality of RMS:

- Consider two tasks with different periods.
- Is a non-preemptive schedule feasible?



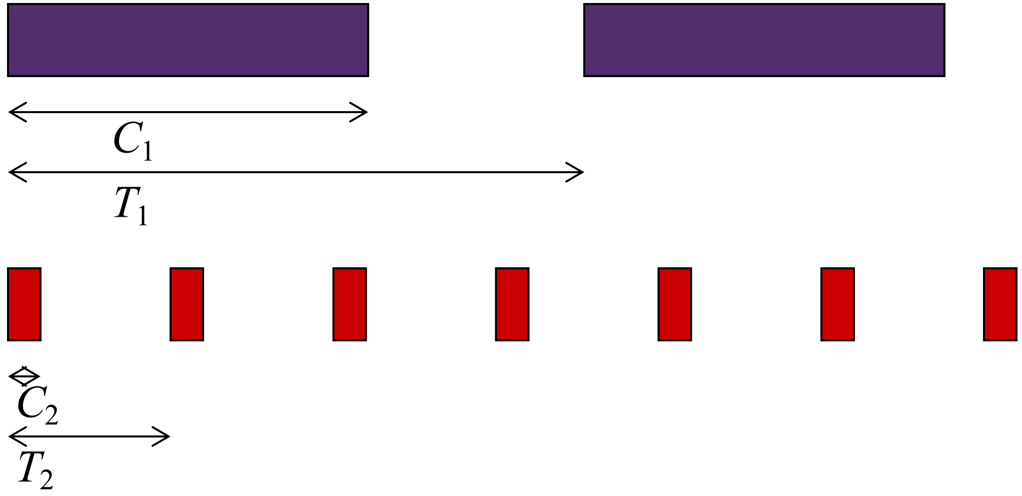
Showing Optimality of RMS:

- Non-preemptive schedule is not feasible. Some instance of the Red Task (2) will not finish within its period if we do non-preemptive scheduling.



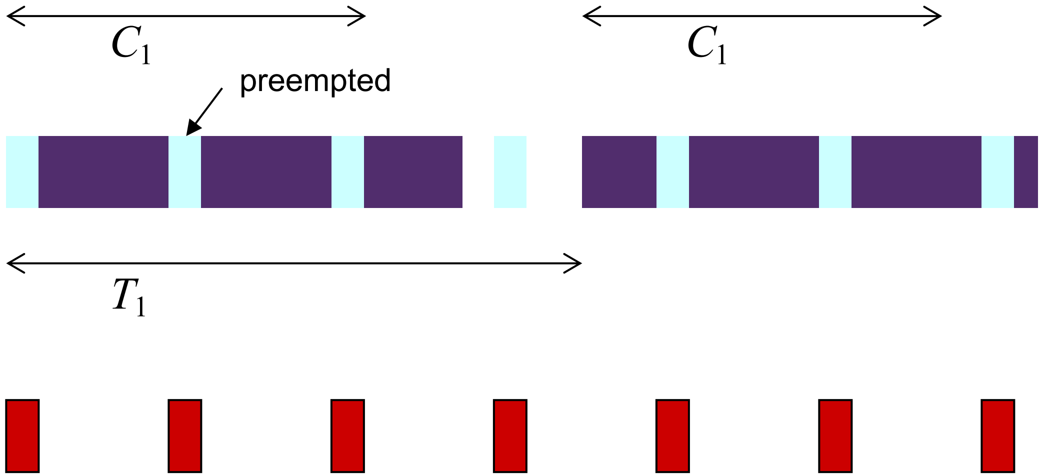
Showing Optimality of RMS:

- What if we had a preemptive scheduling with higher priority for red task?



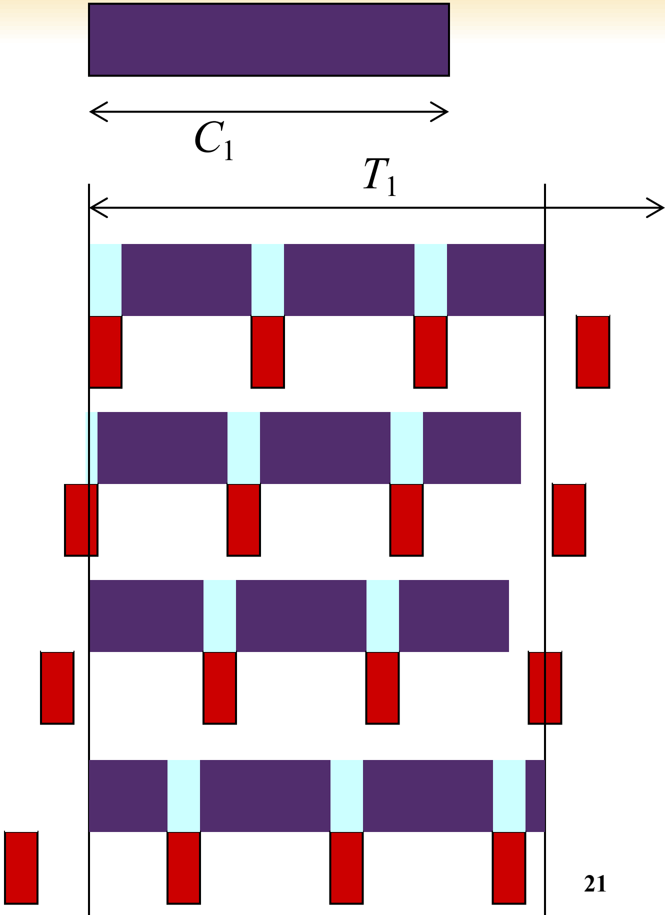
Showing Optimality of RMS:

- Preemptive schedule with the red task having higher priority is feasible. Note that preemption of the purple task extends its completion time.



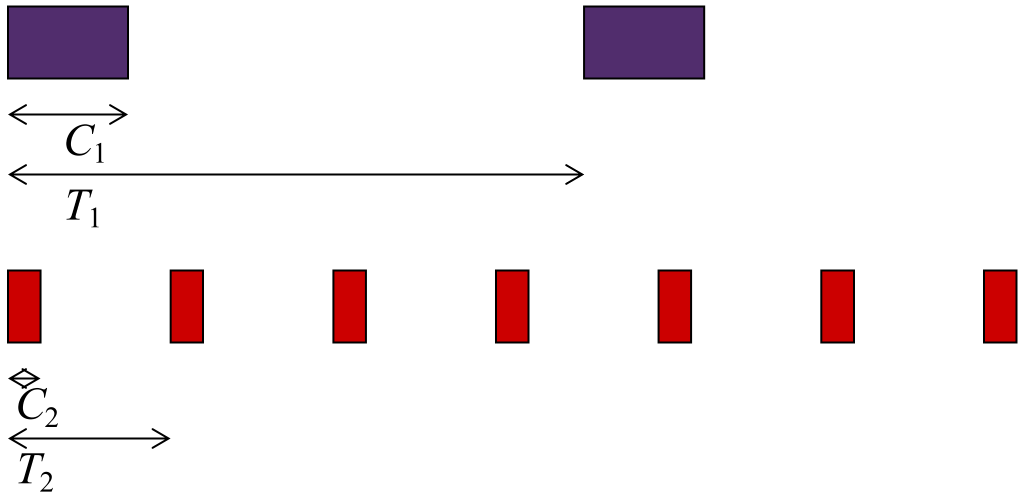
Alignment of tasks

- Completion time of the lower priority task is worst when its *starting phase* matches that of higher priority tasks.
- Thus, when checking schedule feasibility, it is sufficient to consider only the worst case: All tasks start their cycles at the same time.



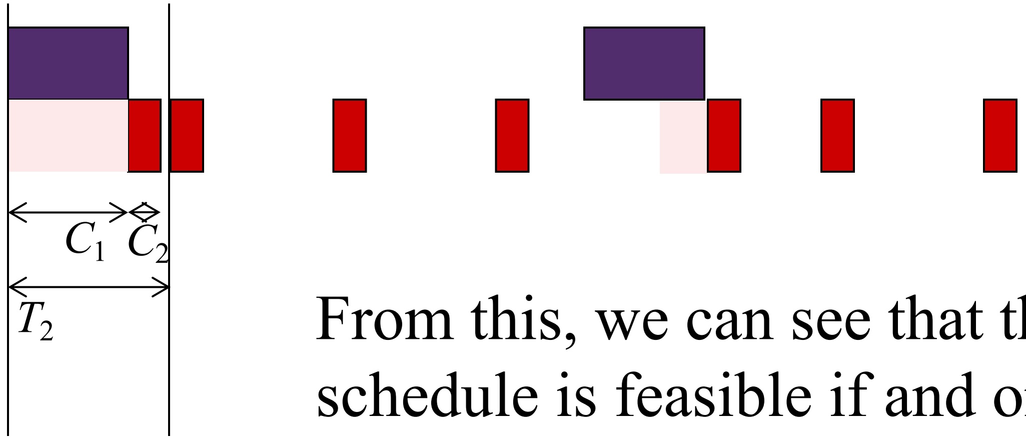
Showing Optimality of RMS: (two tasks)

- It is sufficient to show that if a non-RMS schedule is feasible, then the RMS schedule is feasible.
- Consider two tasks as follows:



Showing Optimality of RMS: (two tasks)

The non-RMS, fixed priority schedule looks like this:



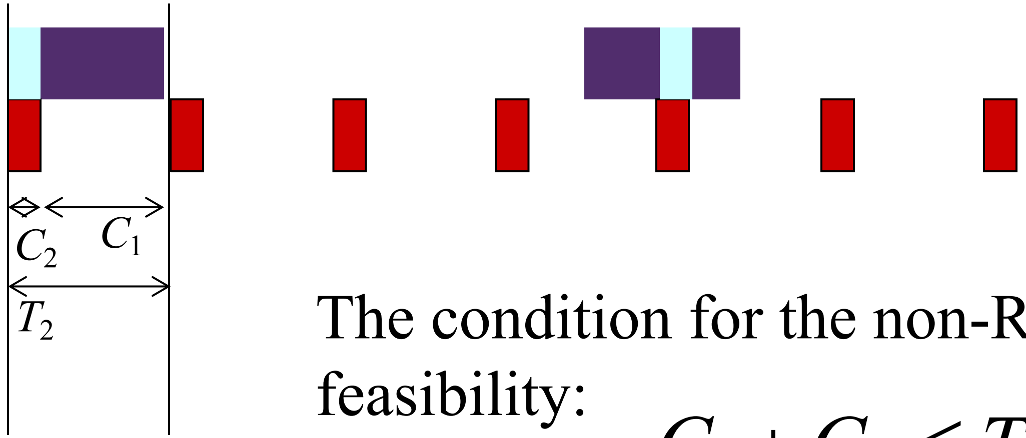
From this, we can see that the non-RMS schedule is feasible if and only if

$$C_1 + C_2 \leq T_2$$

We can then show that this condition implies that the RMS schedule is feasible.

Showing Optimality of RMS: (two tasks)

The RMS schedule looks like this: (task with smaller period moves earlier)



The condition for the non-RMS schedule feasibility:

$$C_1 + C_2 \leq T_2$$

is clearly sufficient (though not necessary) for feasibility of the RMS schedule.

Comments

- This proof can be extended to an arbitrary number of tasks (though it gets much more tedious).
- This proof gives optimality only w.r.t. feasibility.
- Practical implementation:
  - Timer interrupt at greatest common divisor of the periods.
  - Multiple timers

RM Scheduler: Processor Utilization

$$\mu = \sum_{i=1}^{n} \frac{e_i}{p_i}$$

- If μ > 1 for any task set, then that task set has no feasible schedule
- Utilization Bound: RMS is feasible $\mu \leq n(2^{1/n} - 1)$
- As n gets large, $\lim_{n \to \infty} n(2^{1/n} - 1) = \ln(2) \approx 0.693.$
- If a task set with any number of tasks does not attempt to use more than 69.3% of the available processor time, then the RM schedule will meet all deadlines.

Liu and Layland, "Scheduling algorithms for multiprogramming in a hard-real-time environment," J. ACM, 1973.

Jackson's Algorithm: EDD (1955)

- Given n independent one-time tasks with deadlines $d_1, \ldots, d_n$, schedule them to minimize the maximum lateness, defined as

$$L_{\max} = \max_{1 \le i \le n} \{ f_i - d_i \}$$

- where f_i is the finishing time of task i. Note that this is negative iff all deadlines are met.
- Earliest Due Date (EDD) algorithm: Execute them in order of non-decreasing deadlines.
- Note that this does not require preemption.

EDD is Optimal

- Optimal in the Sense of Minimizing Maximum Lateness

  - To prove, use an *interchange argument*.

  - Given a schedule S that is not EDD, there must be tasks a and b where a immediately precedes b in the schedule but $d_a > d_b$. Why?

  - We can prove that this schedule can be improved by interchanging a and b. Thus, no non-EDD schedule achieves smaller max lateness than EDD

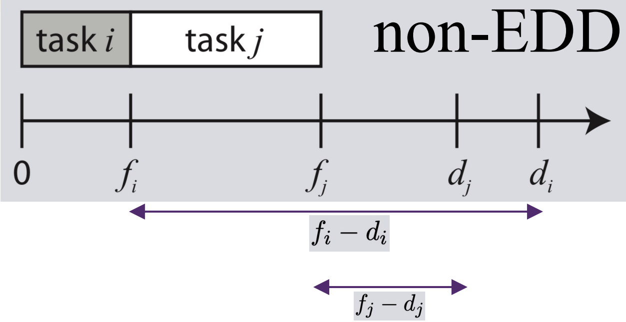
  - So the EDD schedule must be optimal.

Maximum Lateness

➢ First Schedule (non-EDD)

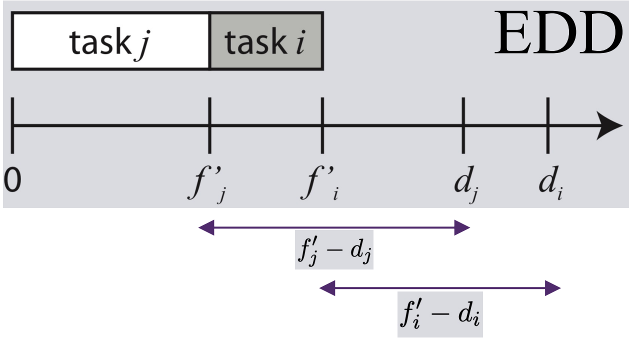
$$L_{\max} = \max(f_i - d_i, f_j - d_j) = f_j - d_j$$

- where $f_i \leq f_j$ and $d_j < d_i$



➢ Second Schedule (EDD)

$$L'_{\max} = \max(f'_i - d_i, f'_j - d_j)$$



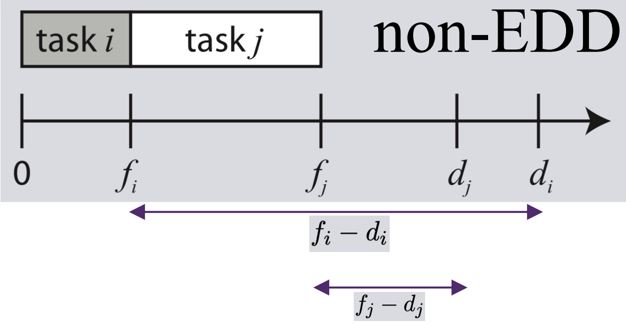
Consider Cases

Case 1: $L'_{\max} = f'_i - d_i$

Since $f'_i = f_j$ $d_j < d_i$

$L'_{\max} = f_j - d_i \le f_j - d_j$

Hence, $L'_{\max} \le L_{\max}$

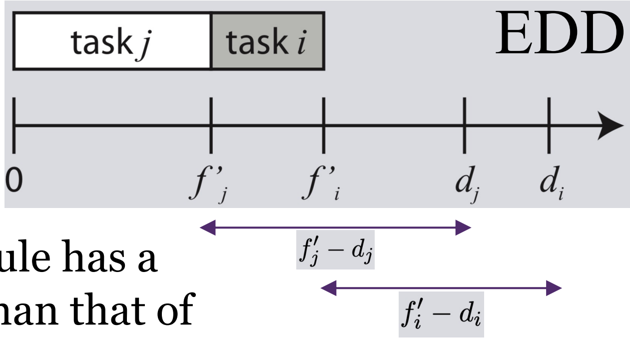


Case 2: $L'_{\max} = f'_j - d_j$

Since $f'_j \le f_j$

$L'_{\max} \le f_j - d_j$

Hence, $L'_{\max} \le L_{\max}$



In both cases, the second schedule has a maximum lateness no greater than that of the first schedule.
EDD minimizes maximum lateness.

Horn’s algorithm: EDF (1974)

- Extend EDD by allowing tasks to “arrive” (become ready) at any time.
- Earliest deadline first (EDF): 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 arrived tasks is optimal w.r.t. minimizing the maximum lateness.
- Proof uses a similar interchange argument.

Using EDF for Periodic Tasks

- The EDF algorithm can be applied to periodic tasks as well as aperiodic tasks.
  - Simplest use: Deadline is the end of the period.
  - Alternative use: Separately specify deadline (relative to the period start time) and period.

RMS vs. EDF? Which one is better?

- What are the pros and cons of each?

Comparison of EDF and RMS

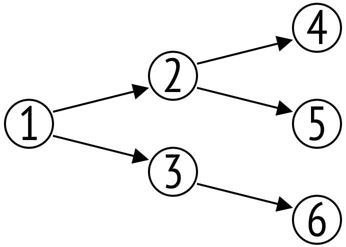
- Favoring RMS
  - Scheduling decisions are simpler (fixed priorities vs. the dynamic priorities required by EDF. EDF scheduler must maintain a list of ready tasks that is sorted by priority.)

Comparison of EDF and RMS

- Favoring EDF
  - Since EDF is optimal w.r.t. maximum lateness, it is also optimal w.r.t. feasibility. RMS is only optimal w.r.t. feasibility.
  - For infeasible schedules, RMS completely blocks lower priority tasks, resulting in unbounded maximum lateness.
  - EDF can achieve full utilization where RMS fails to do that.
  - EDF results in fewer preemptions in practice, and hence less overhead for context switching.
  - Deadlines can be different from the period.

Precedence Constraints

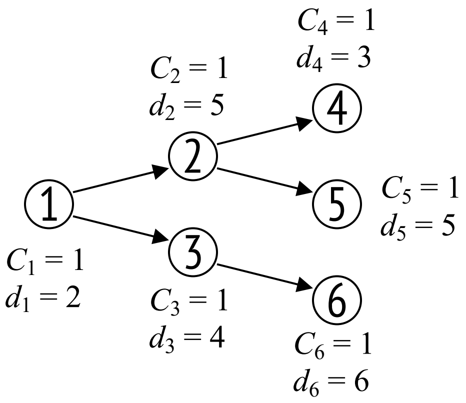
- A directed acyclic graph (DAG) shows precedences, which indicate which tasks must complete before other tasks start.

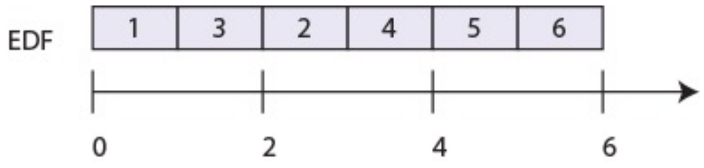


DAG, showing that task 1 must complete before tasks 2 and 3 can be started, etc.

Example: EDF Schedule

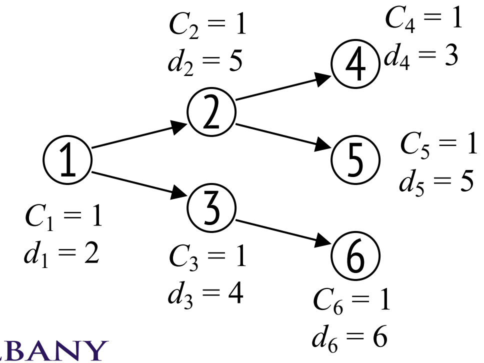
- Is this feasible?

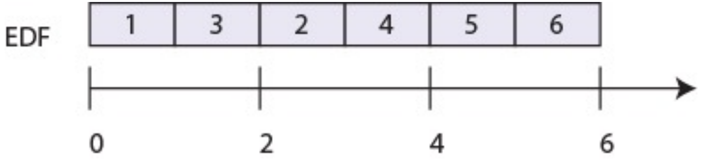




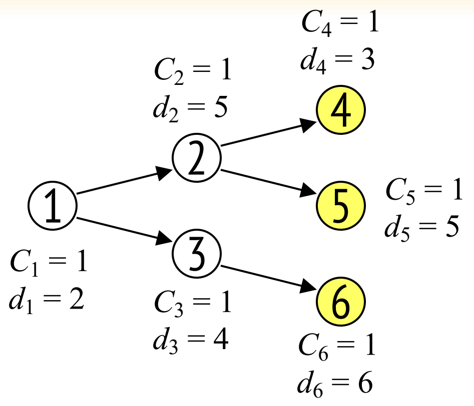
EDF is not optimal under precedence constraints

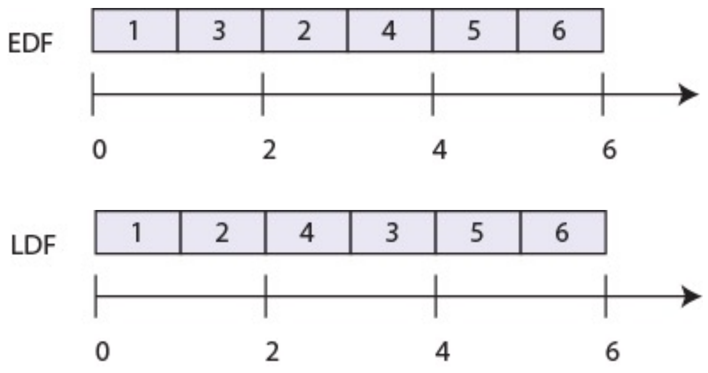
- The EDF schedule chooses task 3 at time 1 because it has an earlier deadline. This choice results in task 4 missing its deadline.





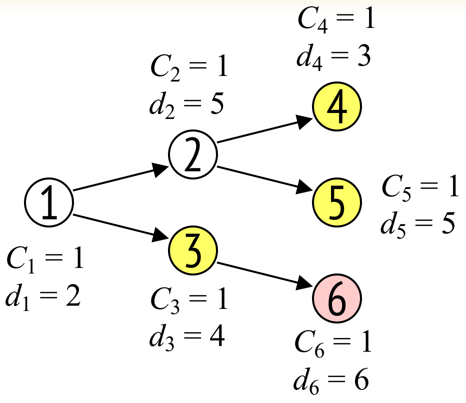
Latest Deadline First (LDF) Lawler 1973

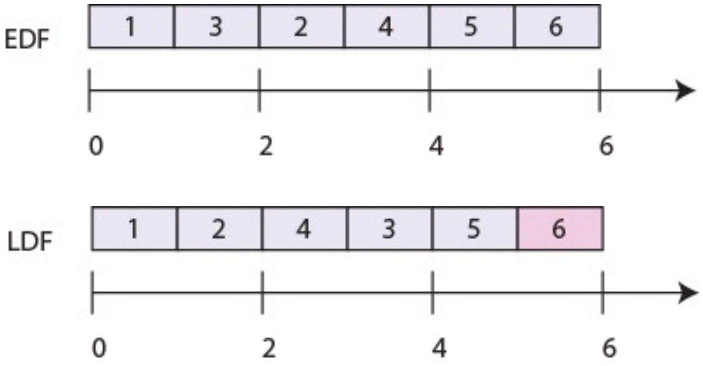




- The LDF scheduling strategy builds a schedule backwards. Given a DAG, choose the leaf node with the latest deadline to be scheduled last, and work backwards.

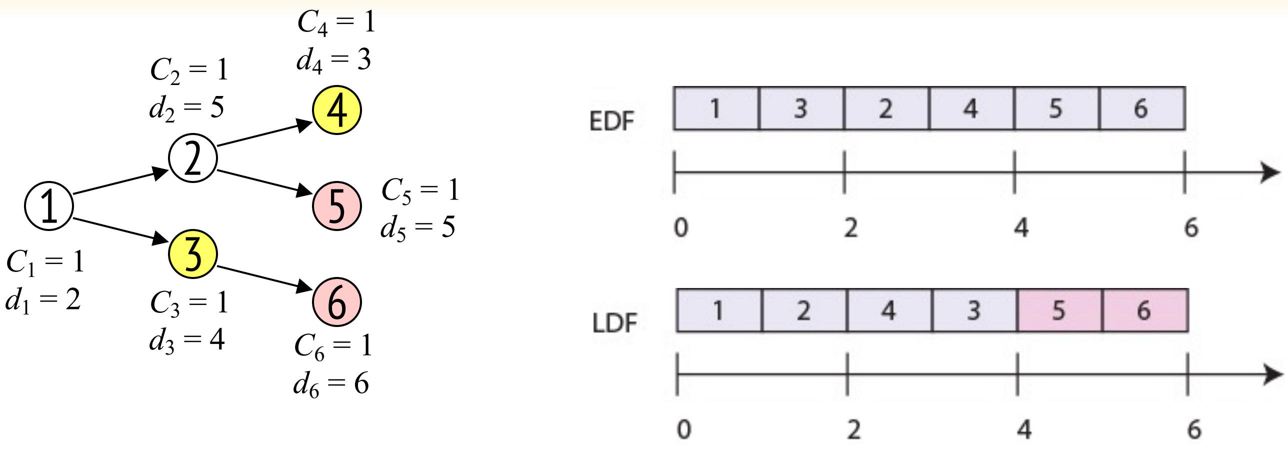
Latest Deadline First (LDF)





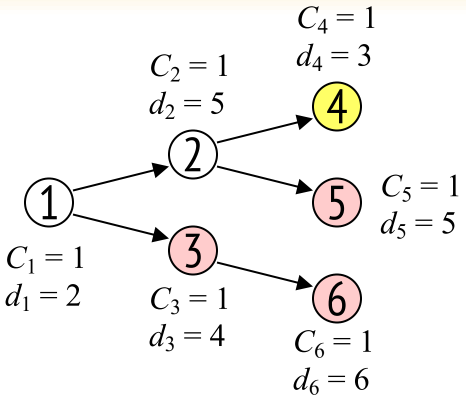
- The LDF scheduling strategy builds a schedule backwards. Given a DAG, choose the leaf node with the latest deadline to be scheduled last, and work backwards.

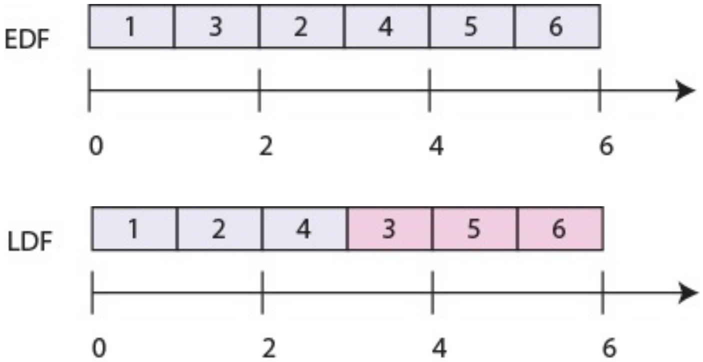
Latest Deadline First (LDF)



- The LDF scheduling strategy builds a schedule backwards. Given a DAG, choose the leaf node with the latest deadline to be scheduled last, and work backwards.

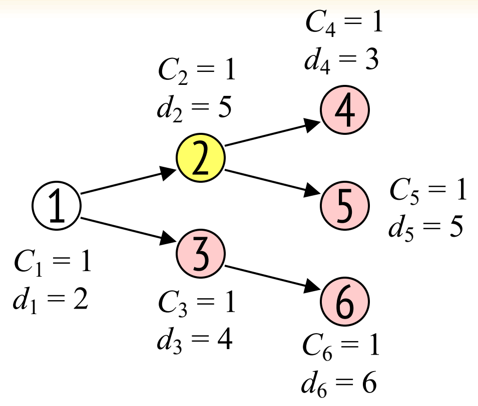
Latest Deadline First (LDF)

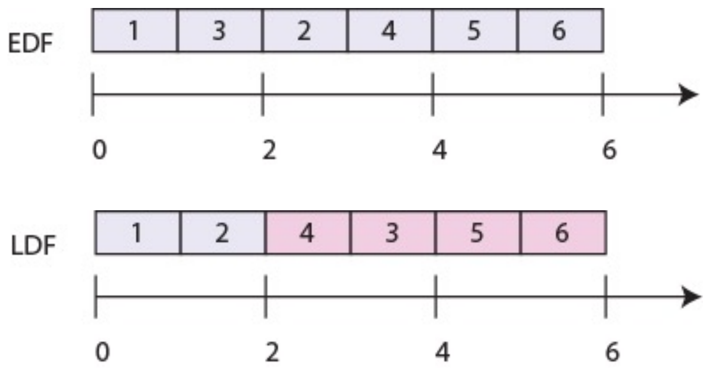




- The LDF scheduling strategy builds a schedule backwards. Given a DAG, choose the leaf node with the latest deadline to be scheduled last, and work backwards.

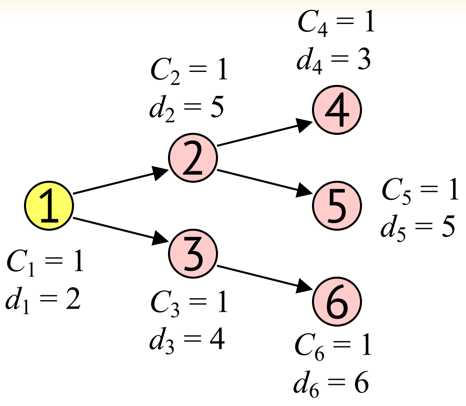
Latest Deadline First (LDF)

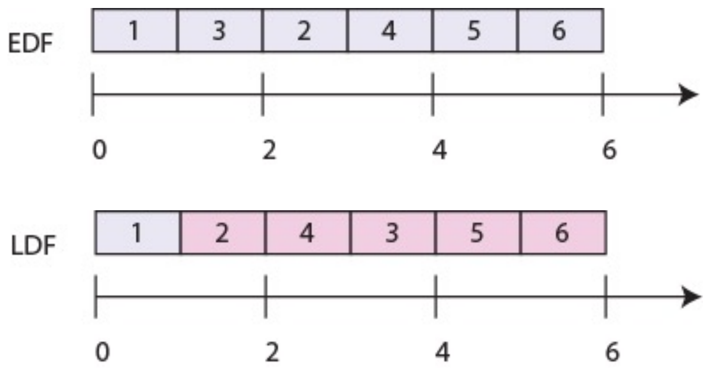




- The LDF scheduling strategy builds a schedule backwards. Given a DAG, choose the leaf node with the latest deadline to be scheduled last, and work backwards.

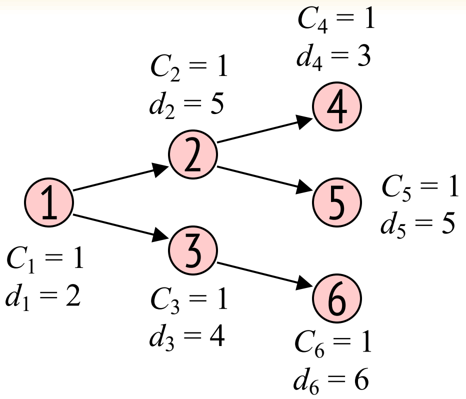
Latest Deadline First (LDF)

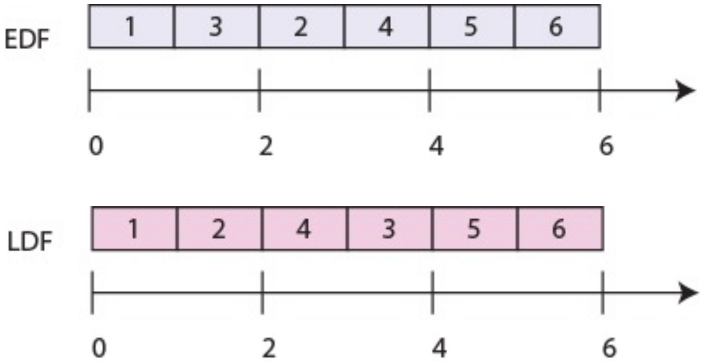




- The LDF scheduling strategy builds a schedule backwards. Given a DAG, choose the leaf node with the latest deadline to be scheduled last, and work backwards.

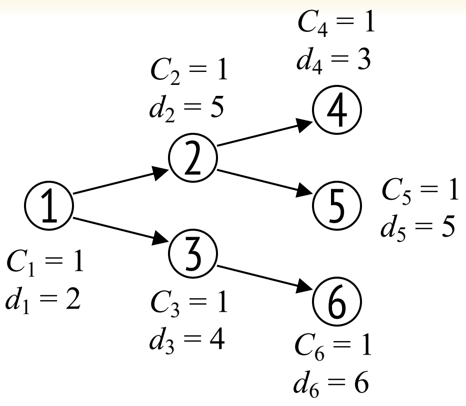
Latest Deadline First (LDF)

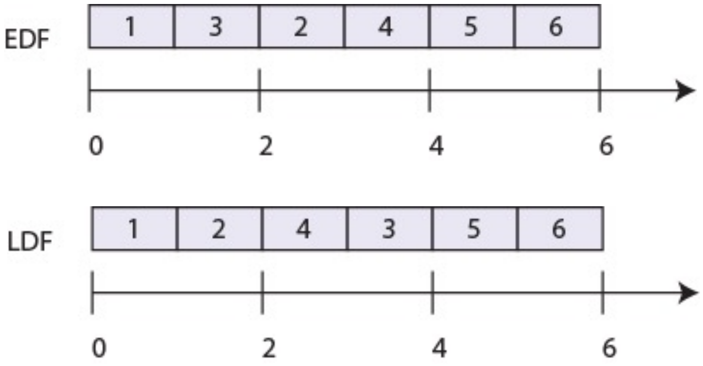




- The LDF scheduling strategy builds a schedule backwards. Given a DAG, choose the leaf node with the latest deadline to be scheduled last, and work backwards.

LDF is optimal for precedence constraints





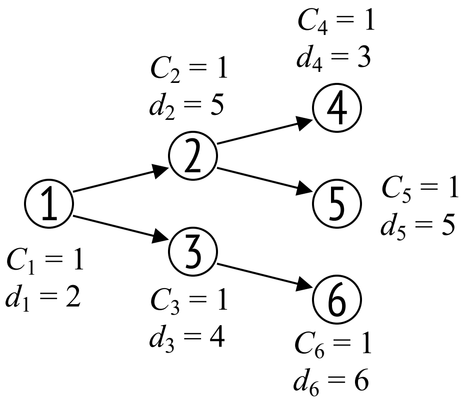
- The LDF schedule shown at the bottom respects all precedences and meets all deadlines.
- Also minimizes maximum lateness

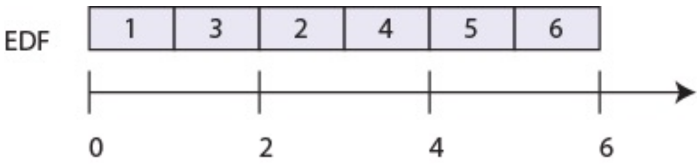
Latest Deadline First (LDF)

- LDF is optimal in the sense that it minimizes the maximum lateness.
- It does not require preemption. (EDF can be made to work with preemption.)
- However, it requires that all tasks be available and their precedences known before any task is executed.

EDF with Precedences or EDF*

- With a **preemptive** scheduler, EDF can be modified to account for precedences and to allow tasks to arrive at arbitrary times.
- Adjust the deadlines and arrival times according to the precedences.





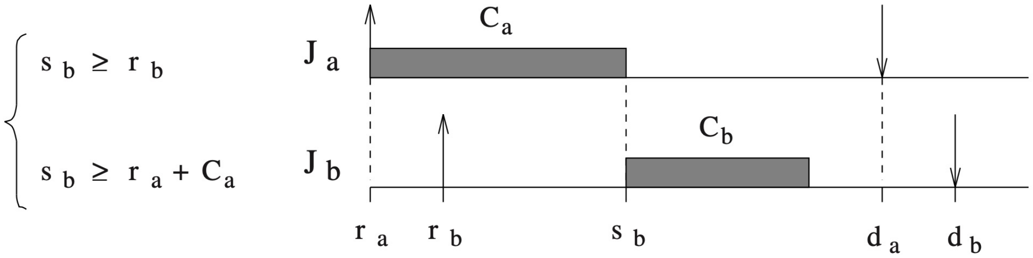
Recall that for the tasks at the left, EDF yields the schedule above, where task 4 misses its deadline.

Modification of Release Times

- **Observations:**

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

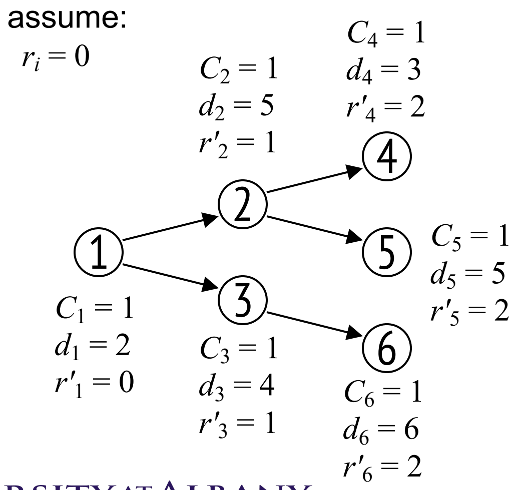
$s_b \geq r_a + C_a$ (that is, J_b must start the execution not earlier than the minimum finishing time of J_a).



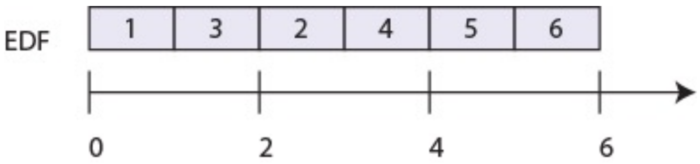
- **Modification:** $r_b^* = \max(r_b, r_a + C_a)$

EDF with Precedences: Modifying Release Times

- Given n tasks with precedences and release times r_i, if task i immediately precedes task j, then modify the release times as follows:



$$r'_j = \max(r_j, r_i + C_i)$$

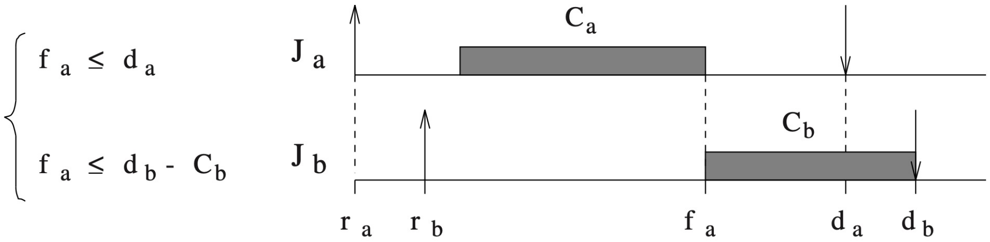


Modification of Deadlines

- Observations:

$f_a \leq d_a$ (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).

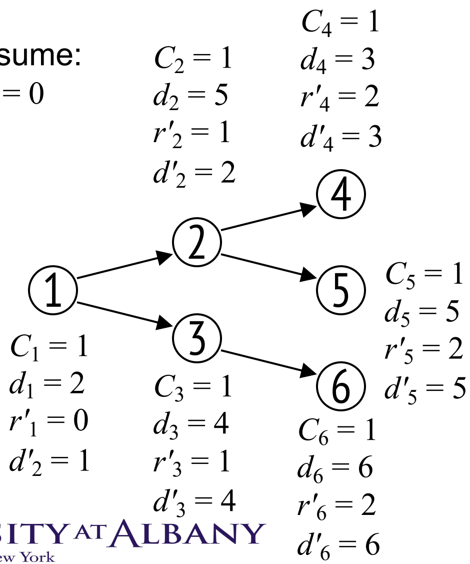


- Modification: $d_a^* = \min(d_a, d_b - C_b)$

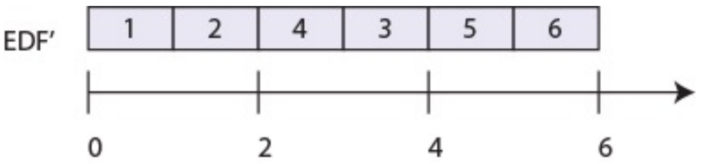
EDF with Precedences: Modifying Deadlines

- Given n tasks with precedences and deadlines d_i, if task i immediately precedes task j, then modify the deadlines as follows:

assume:
$r_i = 0$



$$d'_i = \min(d_i, d'_j - C_j)$$



Using the revised release times and deadlines, the above EDF schedule is optimal and meets all deadlines.

Optimality

- Generalized modified deadline

$$d_i' = \min(d_i, \min_{j \in D(i)} (d_j' - e_j))$$

- EDF with precedences is **optimal** in the sense of minimizing the maximum lateness.