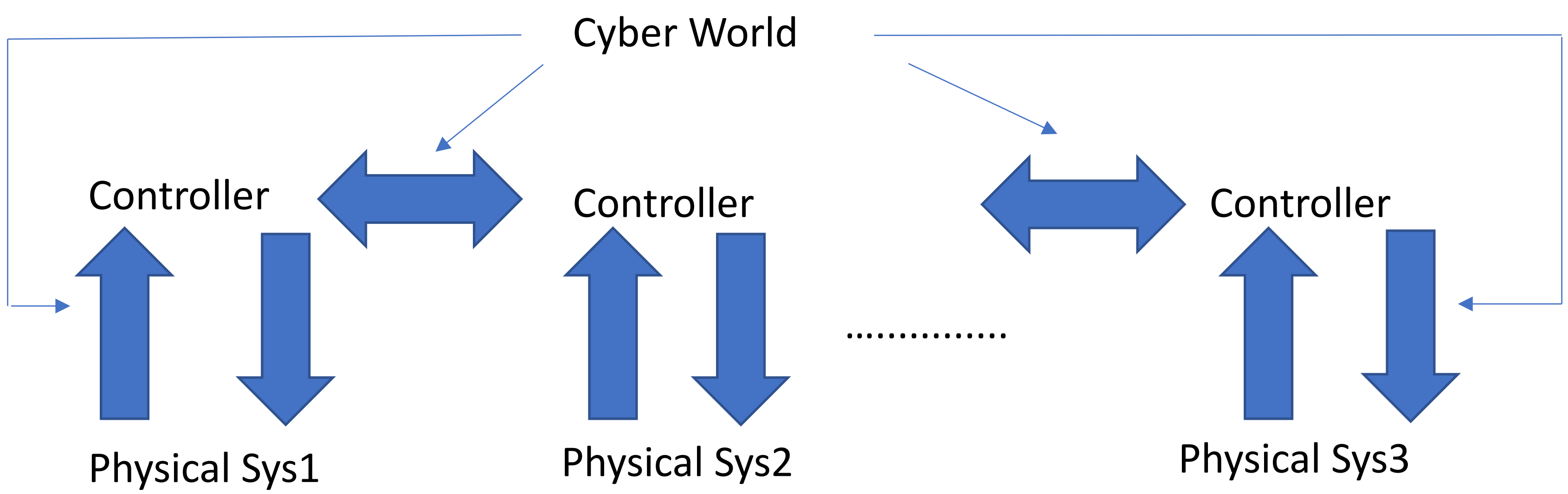
Computational Foundations of Cyber Physical Systems (CS61063)



- Soumyajit Dey
- CSE, IIT Kharagpur

CPS Compute Platforms

- CPS involves significant on-board computation
  - Signal processing – filtering the plant state data
  - State estimation
  - On-board intelligence (can run several optimizations for real time problem solving)
- Low power computation
  - Typically performed with help from on board battery
- Need to use low power processors instead of workstation class processors
  - RISC CPUs- ARM, PowerPC
  - Microcontrollers – Atmel (8 bit RISC ATmega328)

CPS : Platform OS choice: Key requirements

- We want every CPS task (which is *critical* in nature) to execute within *deadline*
- If one task maps to one processing element (processor/ ASIC / FPGA) – no need of OS
- Multiple tasks **mapped** to one processor
  - Can have a *barebone scheduler*
  - Can provide an OS which interfaces among low level drivers and high level tasks

CPS : Platform OS choice: Key requirements

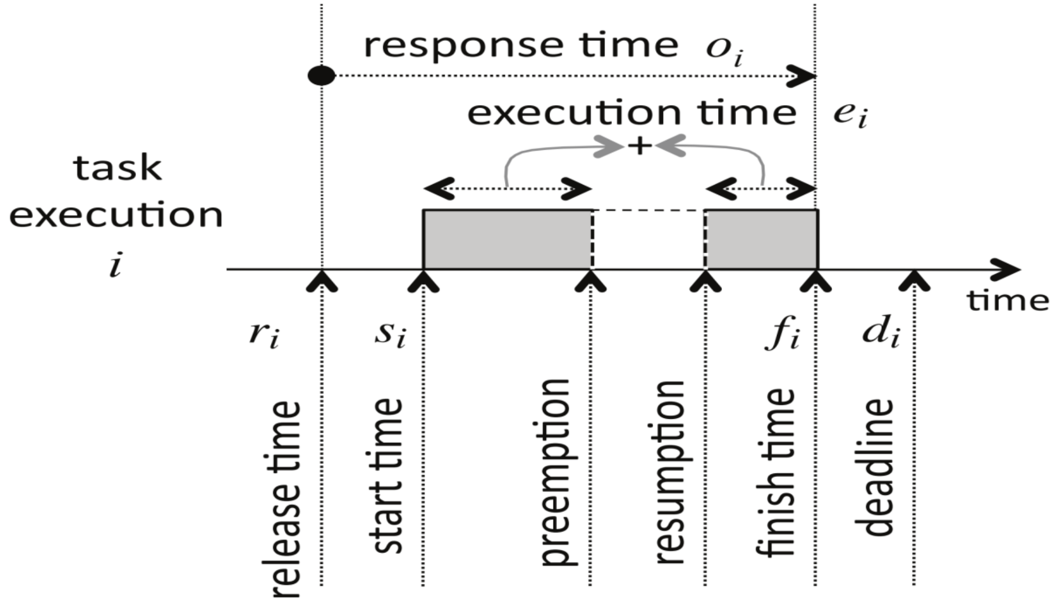
- Mapping – which task gets mapped to which processor
- Ordering – in which order will each processor execute tasks assigned to itself
- Timing – time at which the task executes

- If all decisions are at design time – fully static / off-line scheduler
- If all decisions are at run time – fully dynamic scheduler
- Allow/disallow task pre-emption before completion – preemptive/non-preemptive scheduler

Task Models

- Scheduler may/may not support arrival of tasks
- Periodic – task needs to be executed/arrives once every T time units
- Aperiodic – no such T exists
- Sporadic – T has a lower bound
- Precedence constraints – $i < j$, Task i **can** start executing after task j finishes / task i is **enabled**

TASK execution attributes



- Execution time
- Release time
- Finish time
- Response time

Source : this slide and other slides: book on CPS by Lee and Seshia

Scheduling types

- Hard real time (hard deadlines) |Soft real time (Soft deadlines)
- Non-Preemptive / Preemptive | Fixed/Dynamic Priority
- Priority – can be different than deadline. Can change / remain constant during task execution
- A **preemptive priority-based scheduler** supports arrivals of tasks and at all times is executing the enabled task with the highest priority.
- A **non-preemptive priority-based scheduler** uses priorities to determine which task to execute next after the current task execution completes,
  - never interrupts a task during execution to schedule another task.

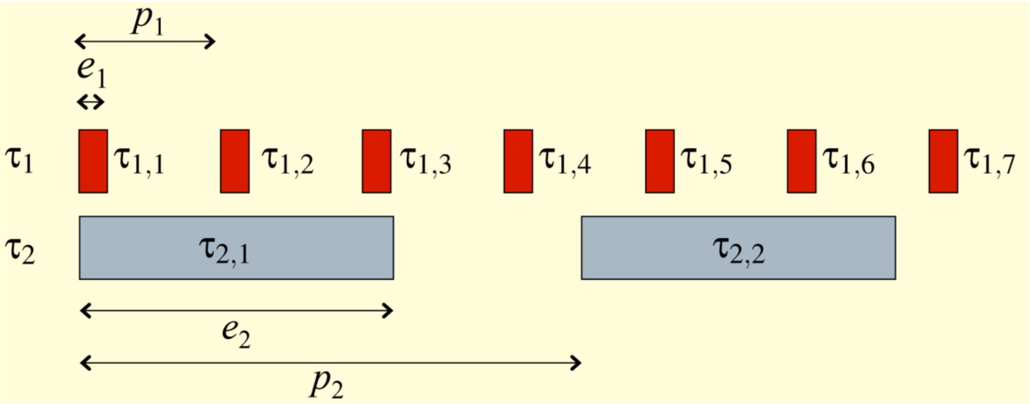
Optimality

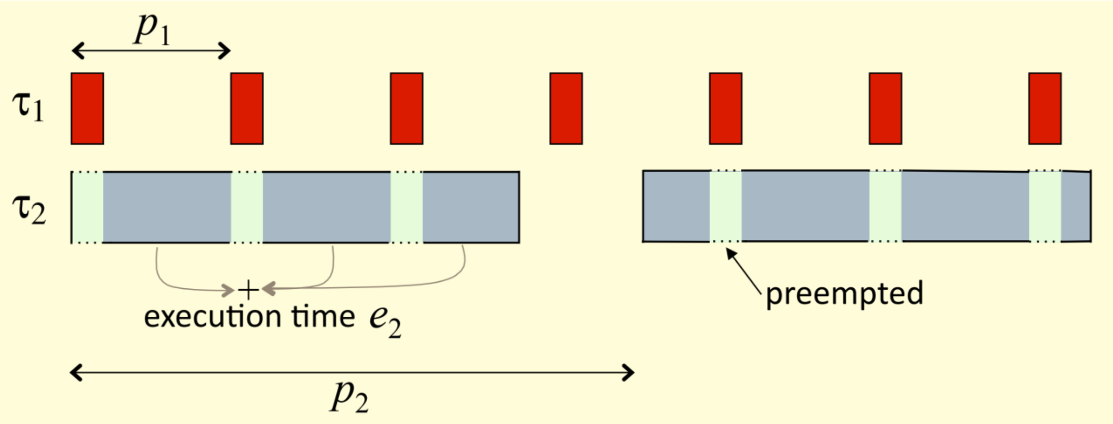
- Scheduling goal : all task executions meet their deadlines. $f_i \leq d_i$
- Feasible schedule : any schedule which does the above
- Optimal w.r.t. feasibility : a scheduler that yields a feasible schedule for any task-set (if such a schedule really exists)
- Goodness of soft real-time schedulers : check *maximum lateness :* $L_{max} = \max_{i \in T}(f_i - d_i)$
- Another criteria : schedule makespan : $M = \max_{i \in T} f_i - \min_{i \in T} r_i$

Rate Monotonic Scheduling

- Set of tasks $\mathrm{T} = \{\tau_1, \tau_2, \cdots \tau_n\}$
- Task τ_i with release time r_i, period p_i has deadline $r_i + jp_i$ for the i-th execution
- RM by Liu and Leyland (1973): preemptive scheduling strategy
  - Optimal w.r.t. feasibility among fixed priority uniprocessor scheduling stratregies
  - Gives higher priority to tasks with lower periods

RMS





Preemptive schedule giving priority to τ_1

No Nonpreemptive schedule is feasible

Note that Preemptive schedule giving priority to τ_2 is not feasible

RMS

- Among all preemptive fixed priority schedulers, RM is optimal with respect to feasibility, under the assumed task model with negligible context switch time.

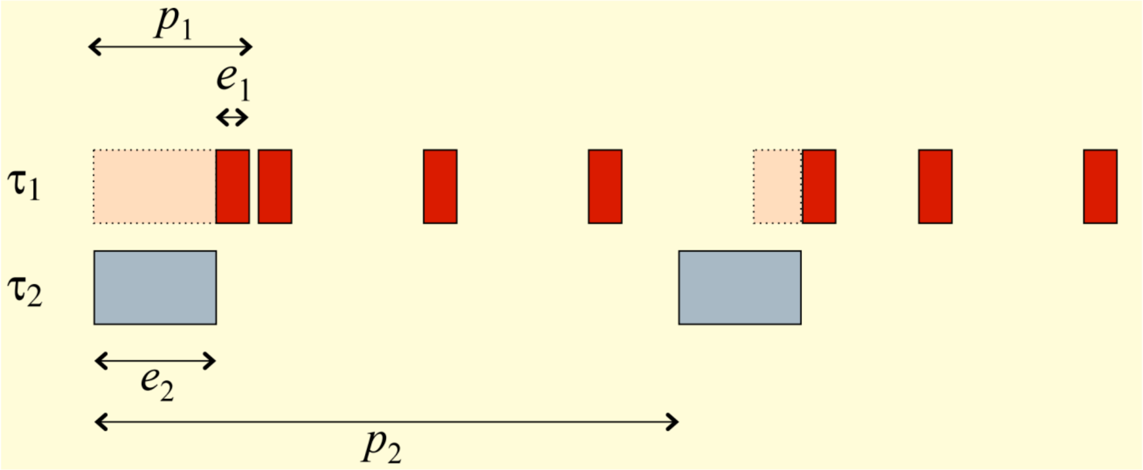


Figure 11.5: The non-RM schedule gives higher priority to τ_2. It is feasible if and only if $e_1 + e_2 \leq p_1$ for this scenario.

RMS

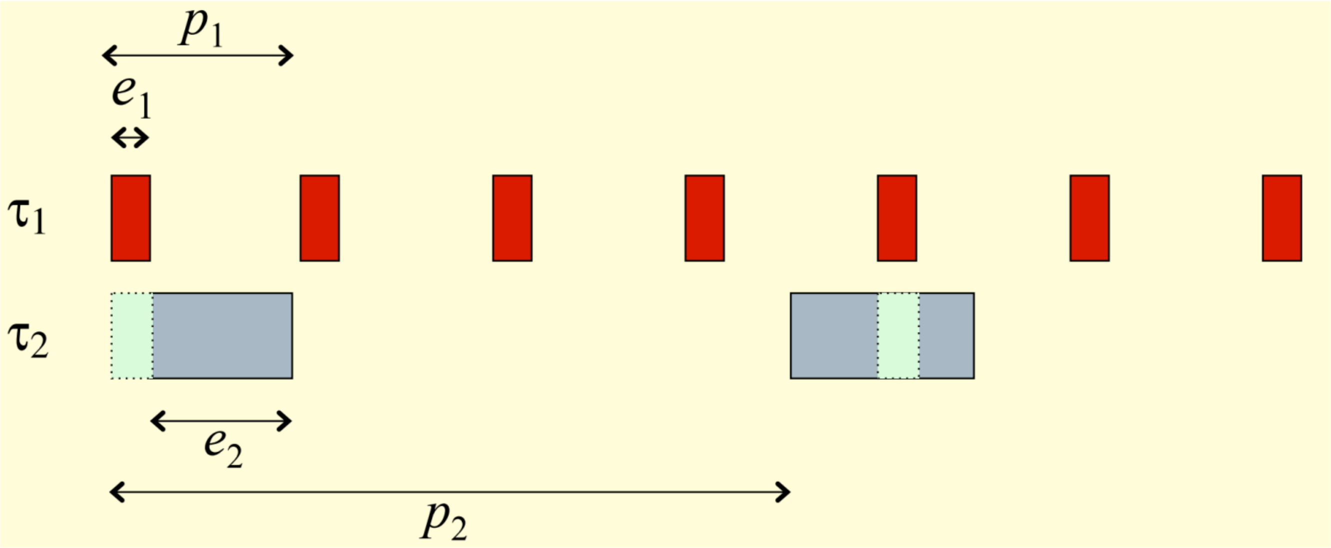


Figure 11.6: The RM schedule gives higher priority to τ_1. For the RM schedule to be feasible, it is sufficient, but not necessary, for $e_1 + e_2 \leq p_1$.

RMS

- *Given a preemptive, fixed priority scheduler and a finite set of repeating tasks T = {τ1,τ2,··· ,τn} with associated periods p1, p2,··· , pn and no precedence constraints, if any priority assignment yields a feasible schedule, then the rate monotonic priority assignment yields a feasible schedule.*
- *Implementation : use timer interrupts*
- Utilization

$$\mu = \sum_{i=1}^{n} \frac{e_i}{p_i}. \qquad \mu \leq n(2^{1/n} - 1),$$

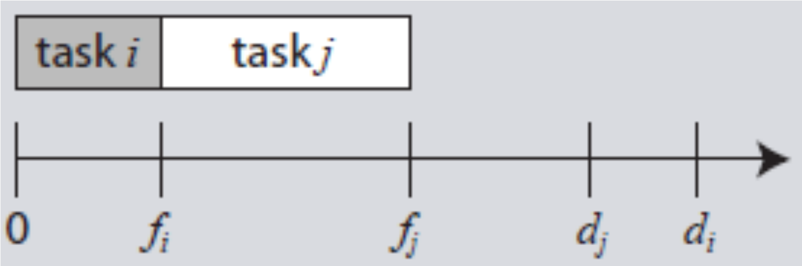
How to get better utilization ?

- We relax the fixed priority constraint and show that dynamic priority schedulers can do better than fixed priority schedulers
- Given a finite set of non-repeating tasks with deadlines and no precedence constraints, a simple scheduling algorithm is earliest due date (EDD), also known as Jackson's algorithm
- Given a finite set of non-repeating tasks $T = \{ \tau_1, \tau_2, \ldots, \tau_n \}$ with associated deadlines $d_1, d_2, \ldots, d_n$ and no precedence constraints, an EDD schedule is optimal in the sense that it minimizes the maximum lateness, compared to all other possible orderings of the tasks.

Proof

- A non-EDD Schedule must have a task T_i preceding a task T_j with $d_j < d_i$
- Since T_i and T_j are independent, reversing them yields a valid schedule

Actual (Schedule 1)

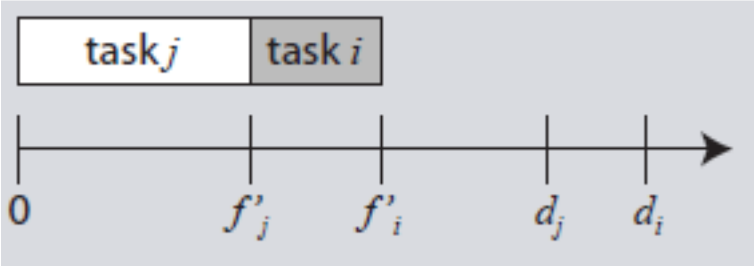


- **Max Lateness**

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

Since $f_i \leq f_j$ and $d_j < d_i$

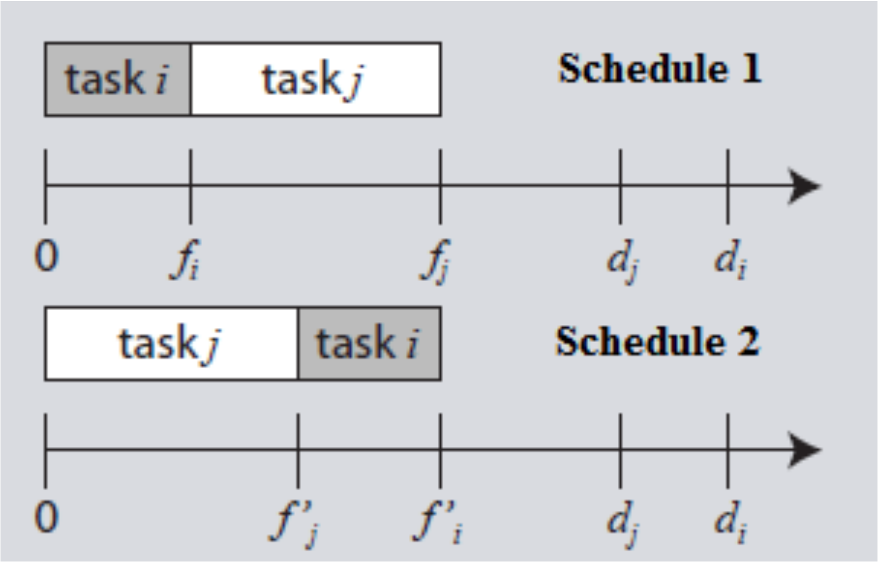
Reversed (Schedule 2)



- **Max Lateness**

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

Proof Continued



- **Case 1: $L'_{max} = f'_i - d_i$**
  Since $f'_i = f_j$ and $d_j < d_i$, $L'_{max} = f'_i - d_i \leq f_j - d_j$
  Hence, $L'_{max} \leq L_{max}$
- **Case 2: $L'_{max} = f'_j - d_j$**
  Since $f'_j \leq f_j$, $L'_{max} \leq f_j - d_j$
  Hence, $L'_{max} \leq L_{max}$
- Schedule 2 has maximum lateness no greater than that of Schedule 1
- **EDD Schedule (Schedule 2) has minimum maximum lateness of all schedules.**

Earliest Deadline First (EDF)

- Limitations of EDD
  - Does not support arrival of tasks
  - Does not support periodic execution of tasks
- EDD is extended to support these – EDF or Horn’s algorithm
- Given a finite set of non-repeating tasks $T = \{ \tau_1, \tau_2, \ldots, \tau_n \}$ with associated deadlines $d_1, d_2, \ldots, d_n$ and arbitrary arrival times, any algorithm that at any instant executes the task with the earliest deadline among all arrived tasks is optimal with respect to minimizing the maximum lateness.

More on EDF

- Dynamic priority scheduling algorithm
- Optimal w.r.t. feasibility among dynamic priority schedulers
- Minimizes the maximum lateness
- Results in fewer preemptions
- An EDF schedule with less than 100% utilization can tolerate increases in execution times and/or reductions in periods and still be feasible
- Not optimal if there are precedences

Latest Deadline First (LDF)

- Optimal with precedences
- Constructs the schedule backwards, choosing first the last task to execute
- The **last task** to execute is the one on which no other task depends that has the latest deadline
- Does not support arrival of tasks

EDF with Precedences (EDF*)

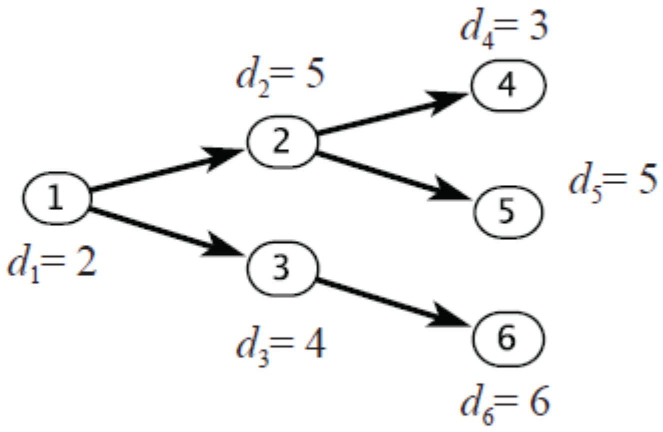
- Supports arrivals, precedences and minimizes the maximal lateness
- Adjust the deadlines of all the tasks
- Suppose the set of all tasks is T
- For a task execution i ϵT, let D(i) ⊂ T be the set of task executions that immediately depend on i in the precedence graph
- For all executions i ϵ T, modified deadline is defined as,

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

- EDF* is then just like EDF with modified deadlines

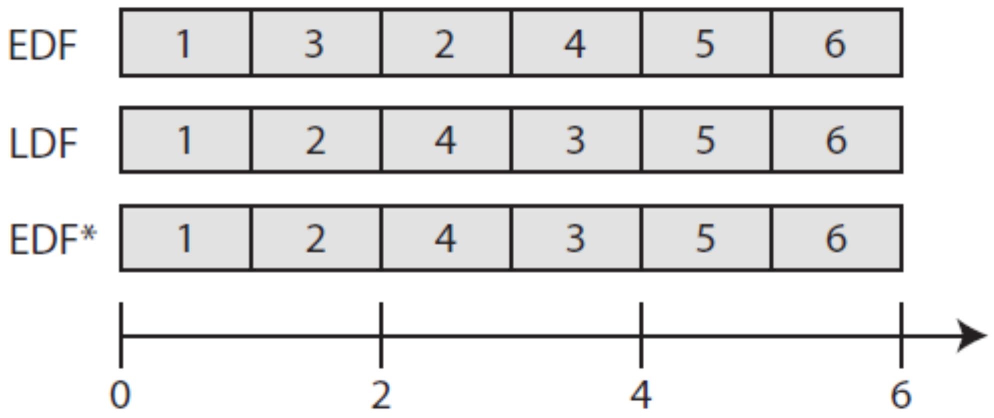
An Example : EDF*

Precedence graph



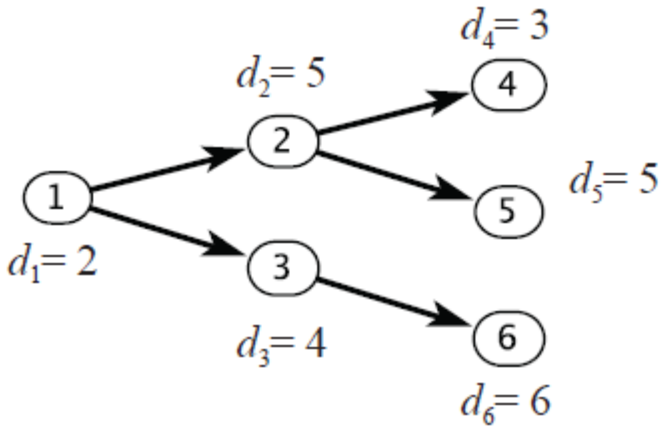
Schedule

- Execution time is 1 unit for all tasks



An Example

Precedence graph



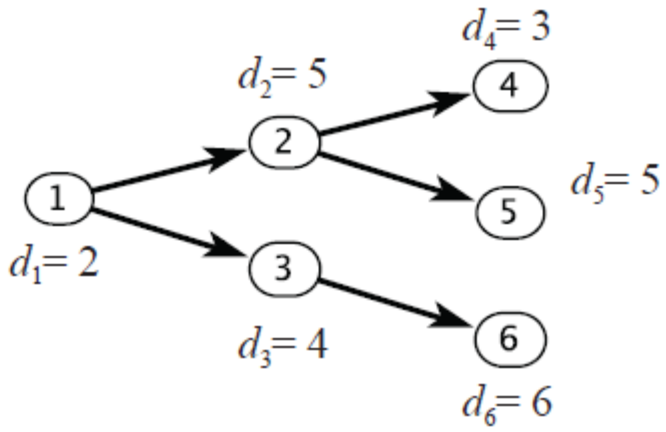
$$d'_i = min(d_i, \min_{j \in D(i)} (d'_j - e_j))$$

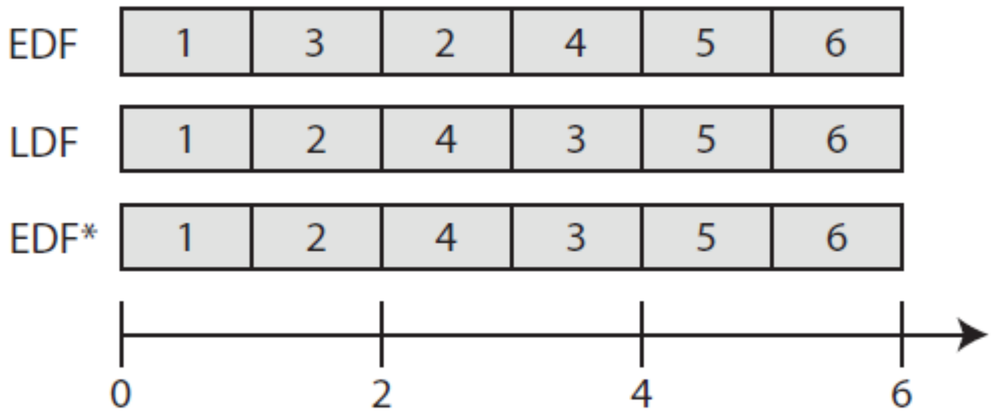
Work the relative deadlines backward

$$d'_4 = d_4 = 3, d'_5 = d_5 = 5, d'_6 = d_6 = 6,$$
$$d'_2 = min(d_2, min(d_4 - e_4, d_5 - e_5)) = min(5, min(2, 4)) = 2$$
$$d'_3 = min(d_3, min(d_6 - e_6)) = min(4, 5) = 4$$

An Example

Precedence graph





$$d'_4 = d_4 = 3, d'_5 = d_5 = 5, d'_6 = d_6 = 6,$$
$$d'_2 = min(d_2, min(d_4 - e_4, d_5 - e_5)) = min(5, min(2, 4)) = 2$$
$$d'_3 = min(d_3, min(d_6 - e_6)) = min(4, 5) = 4$$

1. Initial enabled task : task 1
2. After 1: task 2 and 3 enabled, task 2 has most immediate relative deadline
3. After 2: task 3, 4, 5 are all enabled, task 4 has most immediate relative deadline
4. After 4: task 3, 5 remain enabled, task 3 has most immediate relative deadline
5. After 3: task 5, 6 remain enabled, task 5 has most immediate relative deadline

THANK YOU