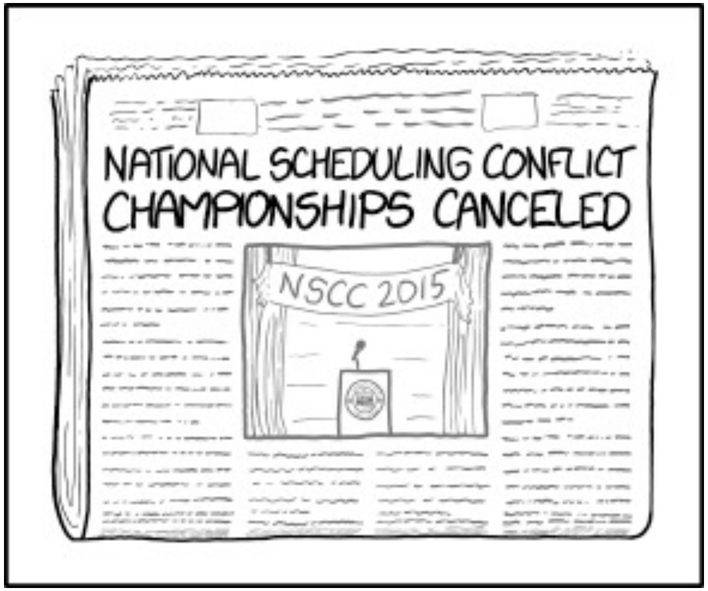
Lecture 15/16 : Scheduling

Slides were originally developed by Profs. Edward Lee and Sanjit Seshia, and subsequently updated by Profs. Gavin Buskes and Iman Shames.

Outline

- Static vs dynamic scheduling
- Various scheduling algorithms:
  - Rate monotonic scheduling
  - Earliest due date
  - Earliest deadline first
  - Latest deadline first



- Reference:
  - Chapter 12 : Edward A. Lee and Sanjit A. Seshia, Introduction to Embedded Systems, A Cyber-Physical Systems Approach
  - Giorgio C. Buttazzo, Hard Real-Time Computing Systems, Springer, 2004.

Responsibilities of a Microkernel (a small, custom OS)

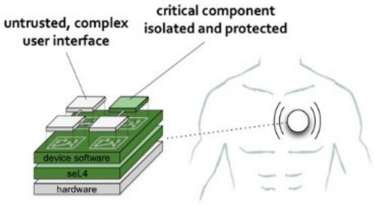
- Scheduling of threads or processes
  - Creation and termination of threads
  - Timing of thread activations
- Synchronisation
  - Semaphores and locks
- Input and output
  - Interrupt handling

A Few More Advanced Functions of an Operating System – Not discussed here…

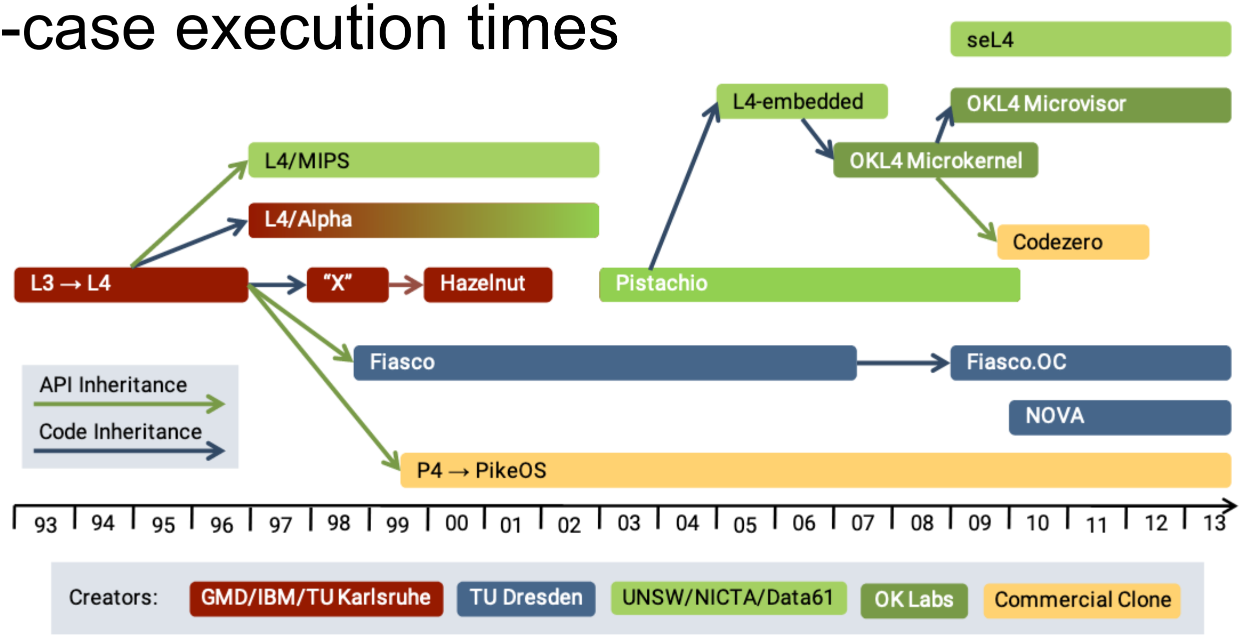
- Memory management
  - Separate stacks
  - Segmentation
  - Allocation and de-allocation
- File system
  - Persistent storage
- Networking
  - TCP/IP stack
- Security
  - User vs. kernel space (see https://www.newscientist.com/article/mg22730392-600-unhackable-kernel-could-keep-all-computers-safe-from-cyberattack-2/ for some pop-sci explanation)
  - Identity management

The seL4® Microkernel

- High-assurance, high-performance operating system microkernel

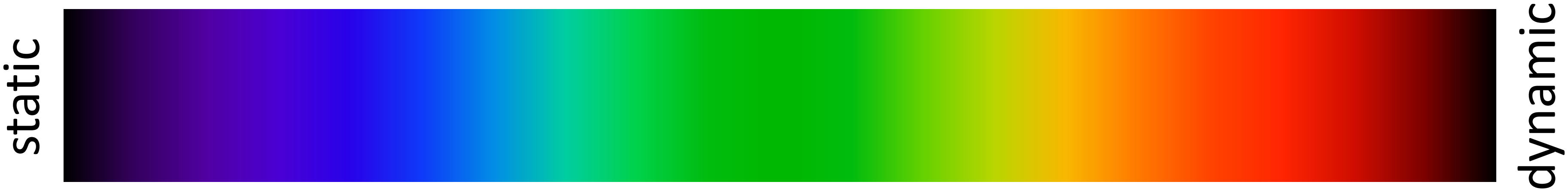

untrusted, complex user interface
critical component isolated and protected
device software
seL4
hardware

- Its implementation is formally (mathematically) proven correct (bug-free) against its specification
- Proven strong security properties and upper bounds on worst-case execution times



Scheduling

- When there are fewer processes than tasks and/or when tasks must be performed at a particular time, a scheduler intervenes: Assignment, Ordering, Timing
- Static [decisions made at design time] and Dynamic [decisions made at run-time]



fully-static scheduler

static-order scheduler (off-line scheduler)

static assignment scheduler + run-time scheduler

fully-dynamic scheduler (on-line scheduler)

Real-time scheduling

- Real-time systems, have all sort of timing constraints:
  - Deadlines: physical (clock-on-the-wall) time value where a task is to be completed
  - Precedence: a task is done before another task
  - Other temporal logic propositions (later lectures): a task not to be executed earlier than a certain time, or not be executed for longer than a certain period, or be executed periodically, or any

- When the deadlines are hard:

  hard real-time scheduling

- When the deadlines can be violated (by small amounts):

  soft real-time scheduling

Outline of a Microkernel Scheduler

- Main Scheduler Thread (Task):
  - set up periodic timer interrupts;
  - create default thread data structures;
  - dispatch a thread (procedure call);
  - execute main thread (idle or power save, for example).
- Thread data structure:
  - copy of all state (machine registers)
  - address at which to resume executing the thread
  - status of the thread (e.g., blocked on mutex)
  - priority, WCET (worst case execution time), and other info to assist the scheduler

Outline of a Microkernel Scheduler

- Timer interrupt service routine:
  - dispatch a thread.
- Dispatching a thread:
  - *disable interrupts;*
  - determine which thread should execute (scheduling);
  - if the same one, enable interrupts and return;
  - save state (registers) into current thread data structure;
  - save return address from the stack for current thread;
  - copy new thread state into machine registers;
  - replace program counter on the stack for the new thread;
  - *enable interrupts;*
  - return.

How to decide which thread to schedule?

- Considerations:
  - Preemptive vs. non-preemptive scheduling
  - Periodic vs. aperiodic tasks
  - Fixed priority vs. dynamic priority
  - Priority inversion anomalies
  - Other scheduling anomalies

When can a new thread be dispatched?

- *Under Non-Preemptive scheduling*:
  - When the current thread completes.
- *Under Preemptive scheduling:*
  - Upon a timer interrupt
  - Upon an I/O interrupt (possibly)
  - When a new thread is created, or one completes.
  - When the current thread blocks on or releases a mutex
  - When the current thread blocks on a semaphore
  - When a semaphore state is changed
  - When the current thread makes any OS call
    - file system access
    - network access
    - …

Preemptive Scheduling

Assumptions:

1. All threads have priorities
   - either statically assigned (constant for the duration of the thread)
   - or dynamically assigned (can vary).
2. Kernel keeps track of which threads are “enabled” (able to execute, e.g., not blocked waiting for a semaphore or a mutex or for a time to expire).

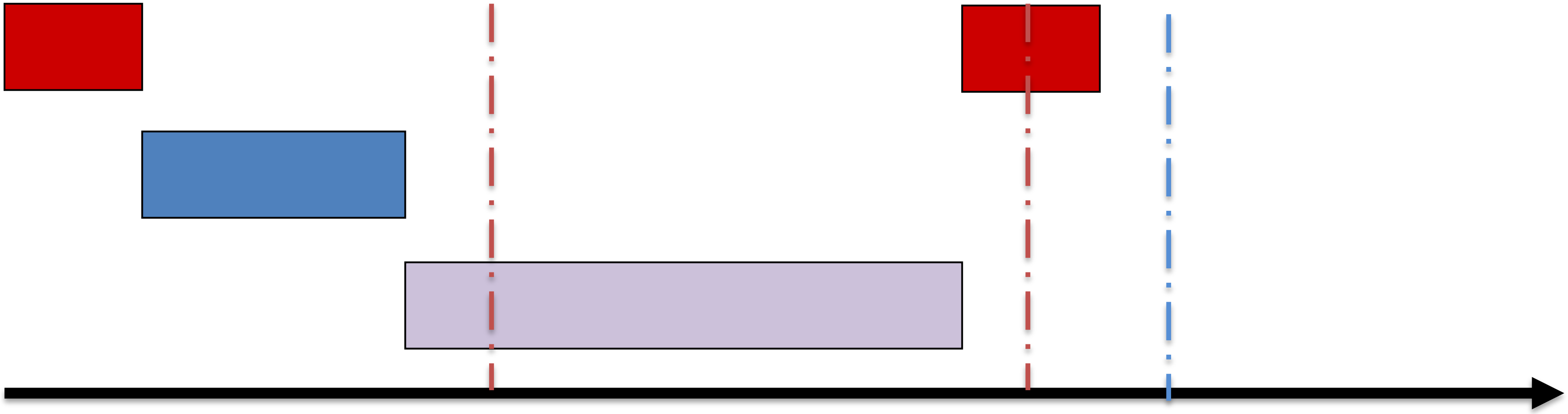
Preemptive scheduling:

At any instant, the enabled thread with the highest priority is executing.

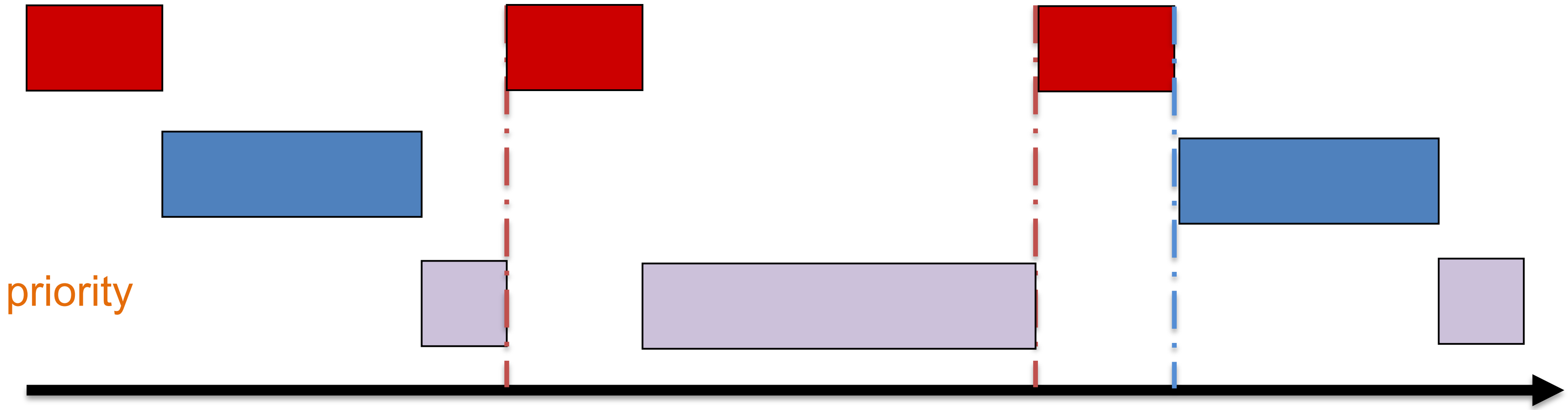
Whenever any thread changes priority or enabled status, the kernel can dispatch a new thread (**the first thread is preempted so that a new one is executed**).

Example

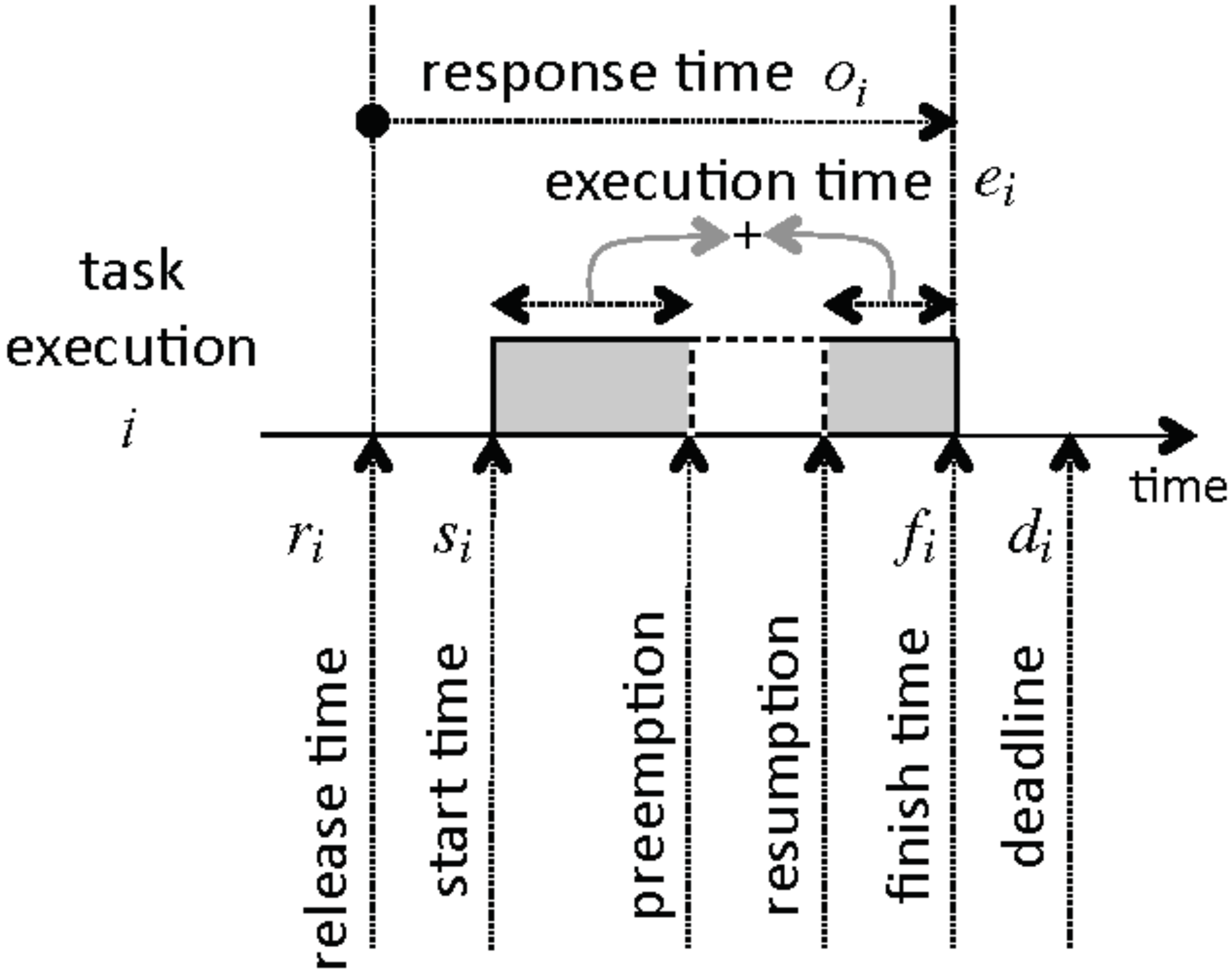
Without preemption:



With preemption:



Useful Terminology



Rate Monotonic Scheduling (RMS)

- Assume *n* tasks invoked periodically with:
  - periods $p_1, \ldots, p_n$ (impose real-time constraints, hard deadlines: end of period)
  - worst-case execution times (WCET) $e_1, \ldots, e_n$
    - assumes no mutexes, semaphores, or blocking I/O
  - no precedence constraints
  - fixed priorities
  - preemptive scheduling

- Rate Monotonic Scheduling: Priorities ordered by period (smallest period has the highest priority)

Rate Monotonic Scheduling

- Assume *n* tasks invoked periodically with:
  - periods $p_1, \ldots, p_n$ (impose real-time constraints)
  - worst-case execution times (WCET) $e_1, \ldots, e_n$
    - assumes no mutexes, semaphores, or blocking I/O
  - no precedence constraints
  - fixed priorities
  - preemptive scheduling

Theorem: Under no precedence constraints, if any priority assignment yields a feasible schedule, then RMS (smallest period has the highest priority) also yields a feasible schedule.

Liu and Layland, "Scheduling algorithms for multiprogramming in a hard-real-time environment," J. ACM, 20(1), 1973.

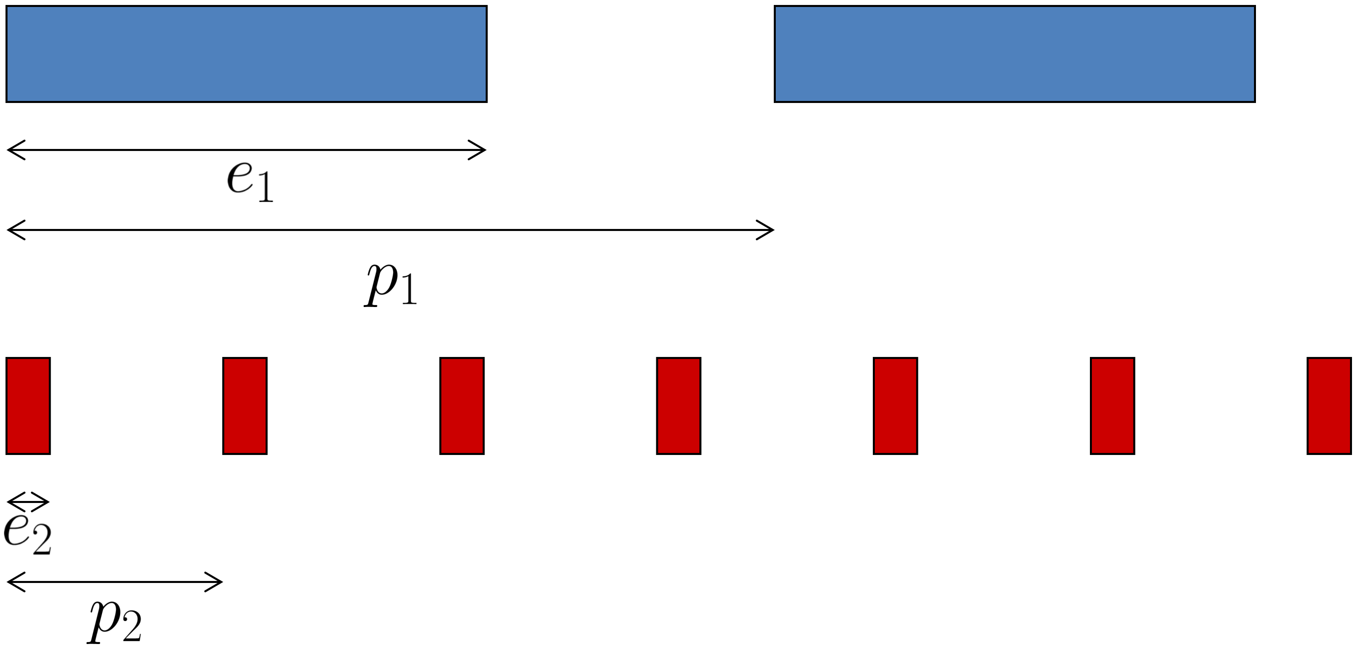
Feasibility for RMS

- Feasibility is defined for RMS to mean that every task executes to completion once within its designated period.
- In other words: all tasks meet their deadlines.

RMS is optimal with respect to feasibility.

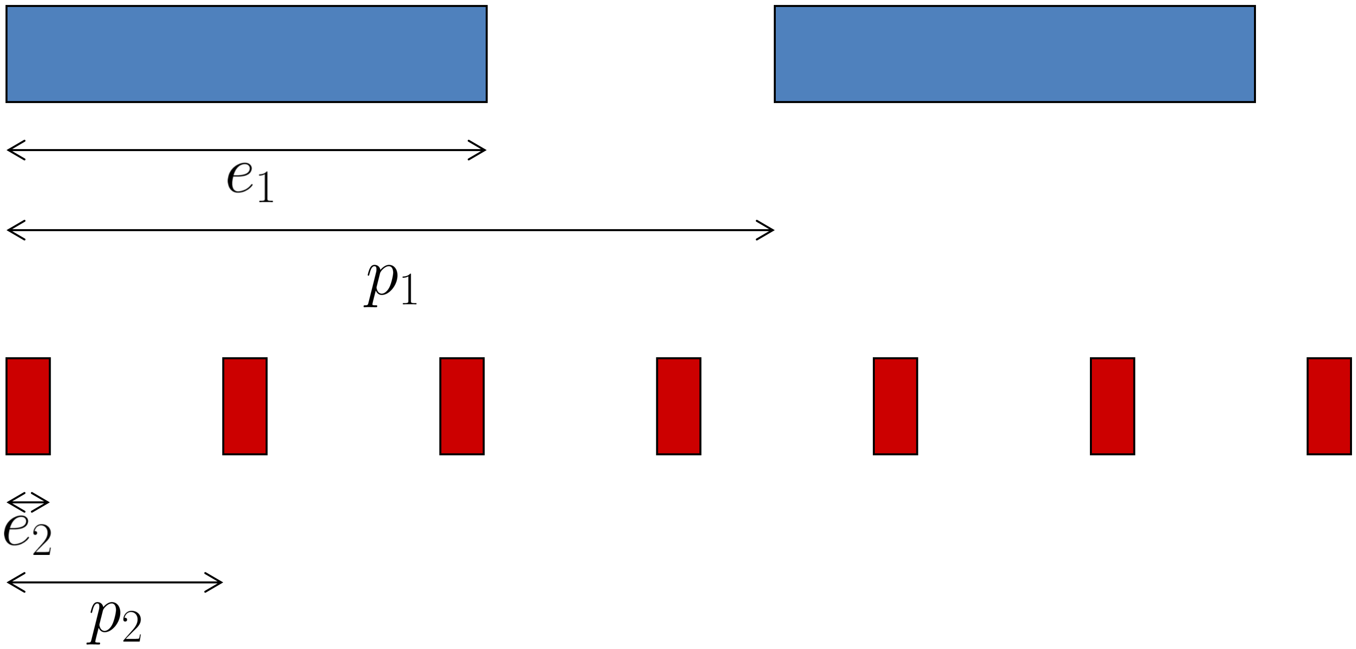
- A scheduler that returns a feasible schedule for any task set for which a feasible schedule exists is called *optimal with respect to feasibility.*

Showing Optimality of RMS: Consider two tasks with different periods



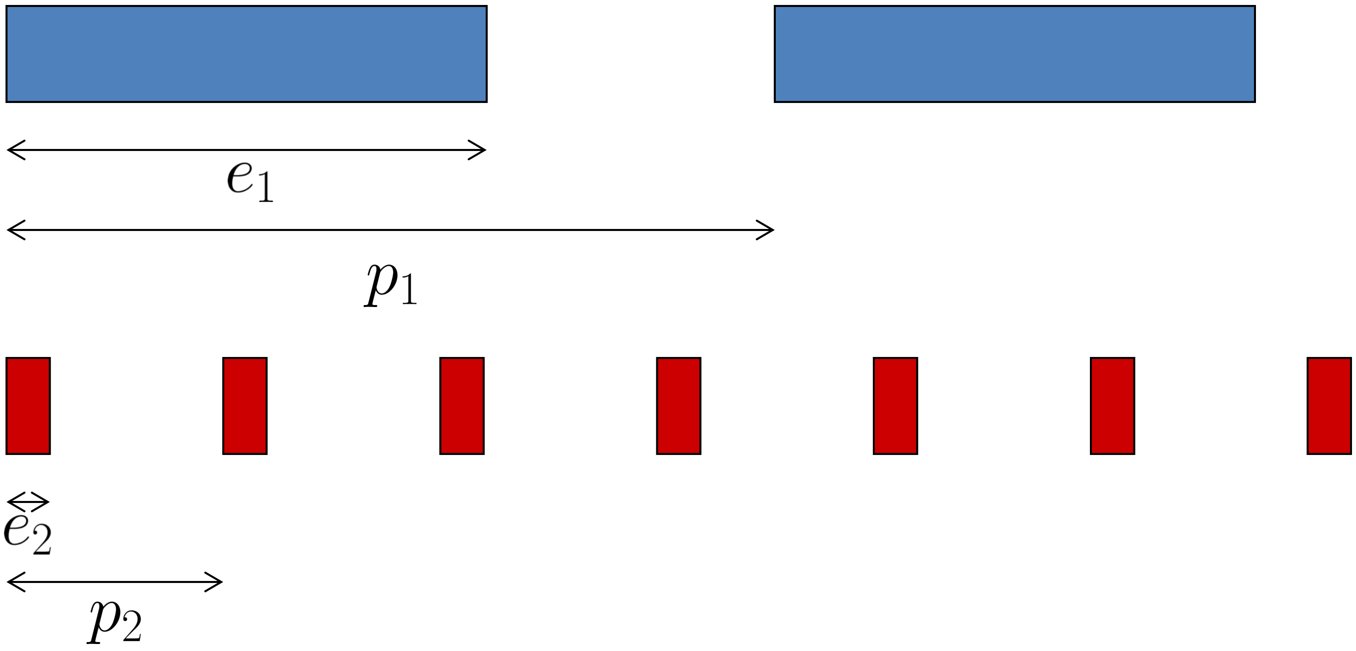
- Is a non-preemptive schedule feasible?

Showing Optimality of RMS: Consider two tasks with different periods



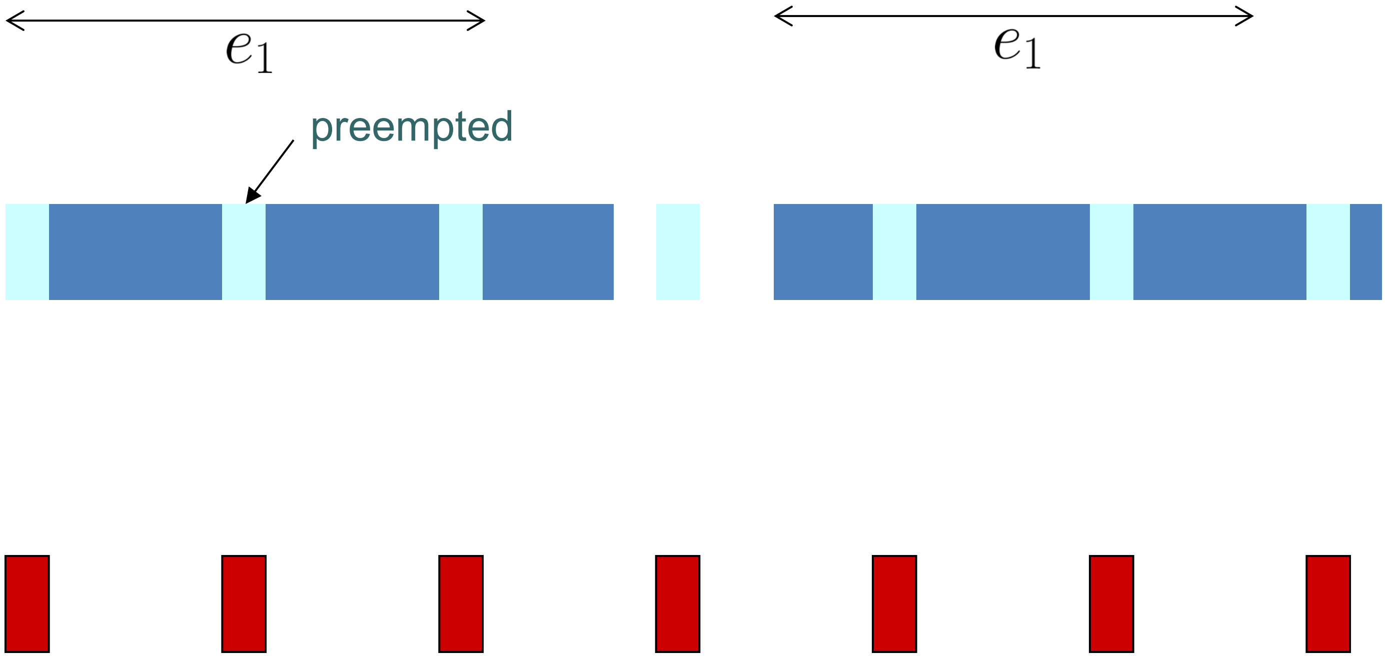
- Non-preemptive schedule is not feasible. Some instance of the Red Task (2) will not finish within its period if we do non-preemptive scheduling.

Showing Optimality of RMS: Consider two tasks with different periods



- What if we had a preemptive scheduling with higher priority for the red task (the task with the smaller period)?

Showing Optimality of RMS: Consider two tasks with different periods



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

Optimality of RMS

Any Implicit Assumptions in Our Reasoning?

Assume zero time needed to switch between tasks (no context switch time).

Theorem: Under no precedence constraints, if any priority assignment yields a feasible schedule, then RMS (smallest period has the highest priority) also yields a feasible schedule.

Contrapositive of the statement: If RMS fails to yield a feasible schedule, then no other priority assignment yields a feasible schedule as well.

More on RMS

- The processor utilization μ is the fraction of time spent in task execution as opposed to idling. For *n* periodic tasks:

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

- If $\mu \geq 1$ then no feasible schedule exists.
- A rate monotonic schedule for *n* periodic tasks is feasible if [Liu and Layland, 1973]:

$$\mu \leq n(2^{\frac{1}{n}} - 1)$$

- In the limit $n \rightarrow \infty$: $\mu \leq \ln 2 \approx 0.693$

Comments

- Proof can be extended to an arbitrary number of tasks (though it gets much more tedious).

- Proof gives optimality only w.r.t. feasibility. It says nothing about other optimality criteria.

- Practical implementation:
  - Timer interrupt at greatest common divisor of the periods.
  - Multiple timers
  - It may have a negative impact on the processor *utilization.*

- FreeRTOS implements preemptive scheduling.

Deadline Driven Scheduling: 1. Jackson’s Algorithm: EDD (1955)

- Given *n* independent *one-time* tasks with deadlines $d_1 , \dots , d_n$, schedule them to minimise 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.

- We might tolerate ‘’small’’ positive values for soft real-time systems.

- *Earliest Due Date (EDD) (aka Jackson’s) algorithm*: Execute them in order of non-decreasing deadlines.

- Note that this does not require preemption.

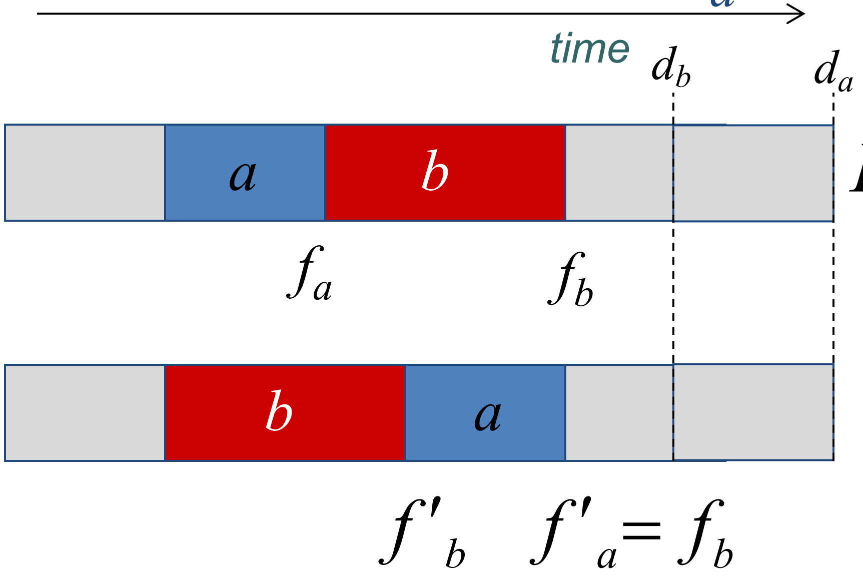
Theorem: EDD is Optimal in the Sense of Minimising Maximum Lateness

- 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.

Theorem: Under no precedence constraints and non-repeating tasks, and EDD is optimal, in the sense that it yields the smallest maximum lateness compared to all other schedules.

Consider a non-EDD Schedule S

There must be tasks a and b where a immediately precedes b in the schedule but $d_a > d_b$



$$L_{\max} = \max\{f_a - d_a, f_b - d_b\} = f_b - d_b$$

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

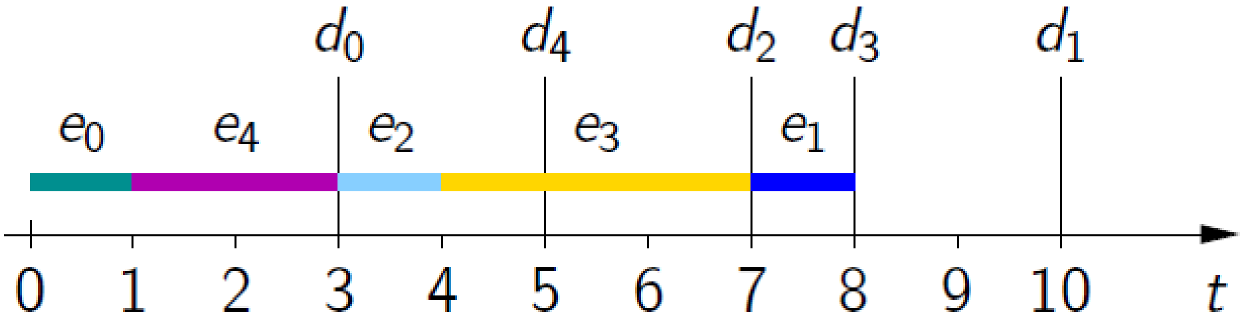
Case 1: $f'_a - d_a > f'_b - d_b$.
Then: $L'_{max} \leq f'_a - d_a = f_b - d_a \leq L_{max}$
(because: $d_a > d_b$).

Case 2: $f'_a - d_a \leq f'_b - d_b$.
Then: $L'_{max} \leq f'_b - d_b \leq L_{max}$
(because: $f'_b < f_b$).

Theorem: $L'_{max} \leq L_{max}$.
Hence, S' is no worse than S.

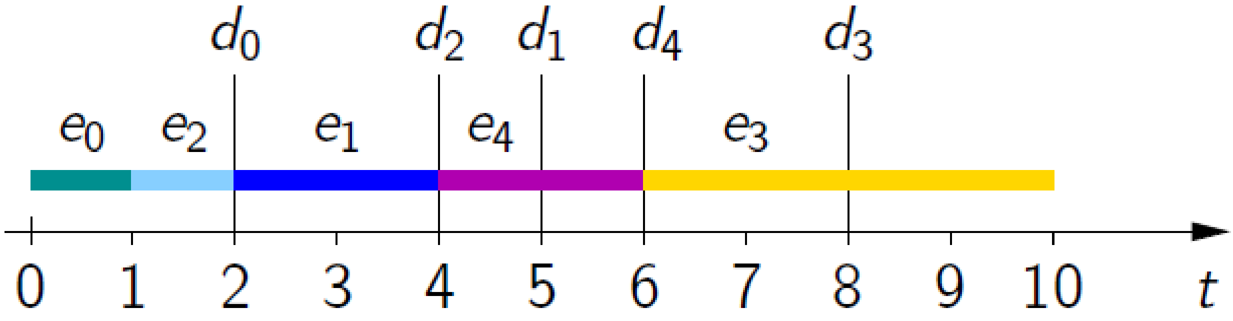
EDD Examples

	e_i	d_i
τ_0	1	3
τ_1	1	10
τ_2	1	7
τ_3	3	8
τ_4	2	5



$$L_{max} = L_3 = -1$$

	e_i	d_i
τ_0	1	2
τ_1	2	5
τ_2	1	4
τ_3	4	8
τ_4	2	6



$$L_{max} = L_3 = 2$$

Deadline Driven Scheduling:
2. Horn’s algorithm: EDF (1974)

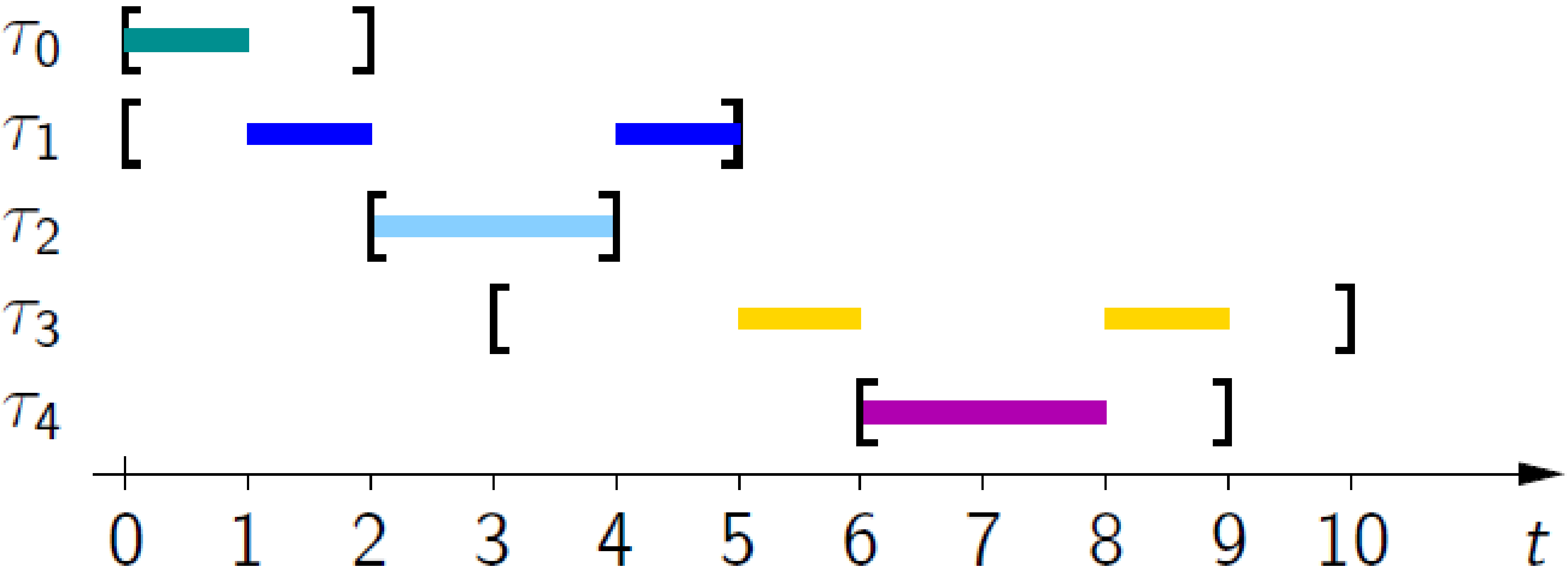
- Extend EDD by allowing tasks to “arrive” (become ready) at any time.
- Earliest deadline first (EDF) (aka Horn’s):

Theorem: 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. minimising the maximum lateness.

- Proof uses a similar interchange argument.
- Minimum maximum lateness ⇒ optimal wrt to feasibility.
- Any feasible schedule will give you a negative number, but this will give you the largest negative number

Horn, W. A. (1974). Some simple scheduling algorithms. Naval Research Logistics Quarterly, 21(1), 177-185.

EDF Examples



	r_i	e_i	d_i
τ_0	0	1	2
τ_1	0	2	5
τ_2	2	2	4
τ_3	3	2	10
τ_4	6	2	9

EDF scheduling applies to both periodic and aperiodic tasks. For periodic tasks, the processor utilization can be 100 %:

Using EDF for Periodic Tasks

- The EDF algorithm can be applied to periodic tasks as well as aperiodic tasks.
- It is a dynamic priority scheduling algorithm. For periodic tasks, a task might receive a different priority in each period.
  - Simplest use: Deadline is the end of the period.
  - Alternative use: Separately specify deadline (relative to the period start time) and period.
- An EDF schedule for *n* periodic tasks is feasible if and only if [Liu and Layland, 1973]:

$$\mu \leq 1$$

Comparison of EDF and RMS

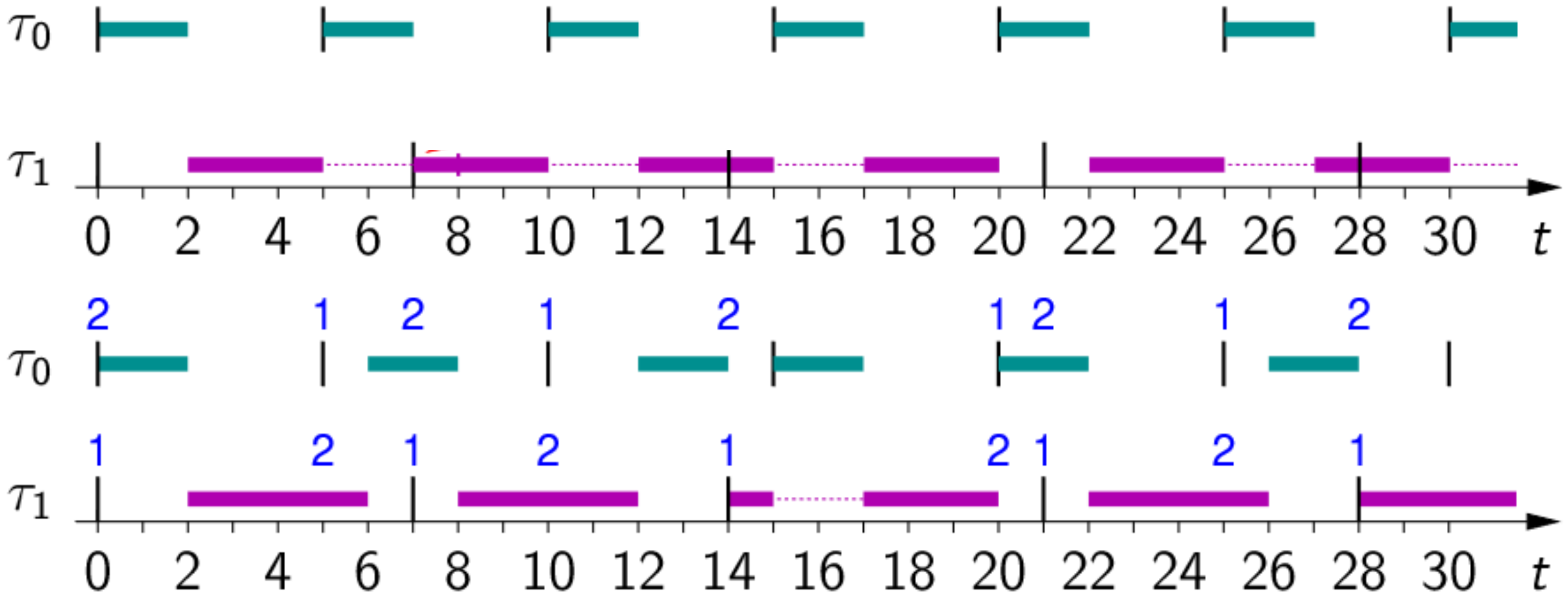
- Favouring RMS
  - Scheduling decisions are simpler (fixed priorities vs. the dynamic priorities required by EDF. EDF scheduler must maintain a list of ready tasks that is sorted by priority.)
- Favouring EDF
  - Since EDF is optimal w.r.t. maximum lateness, it is also optimal w.r.t. feasibility. RMS 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 utilisation 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.

EDF vs RMS Example

- Periodic task τ_0 with $e_0 = 2$ and $p_0 = 5$, and periodic task τ_1 with $e_1 = 4$ and $p_1 = 7$. Processor utilization:

$$\mu = \frac{2}{5} + \frac{4}{7} \approx 0.97 \geq 2(\sqrt{2} - 1) = 0.828$$

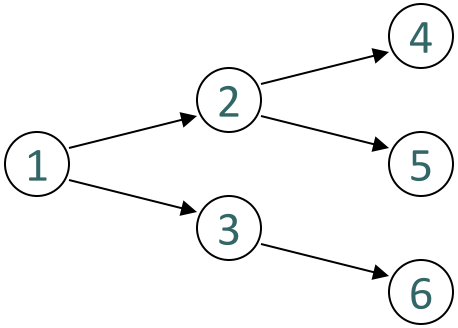
- RMS is not guaranteed to be feasible.



RMS is not

EDF is.

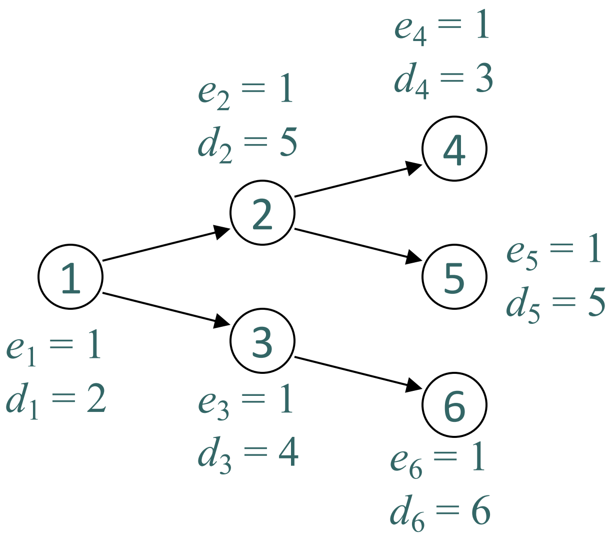
Precedence Constraints

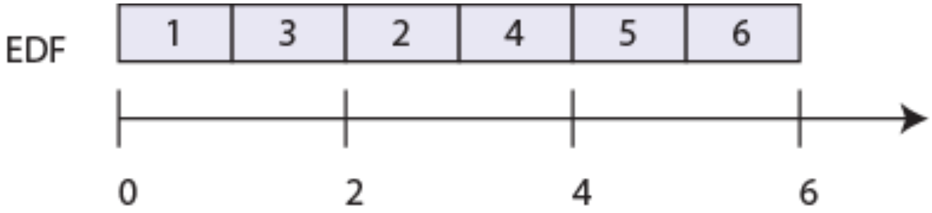


DAG, showing that task 1 must complete before tasks 2 and 3 can be started, etc.

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

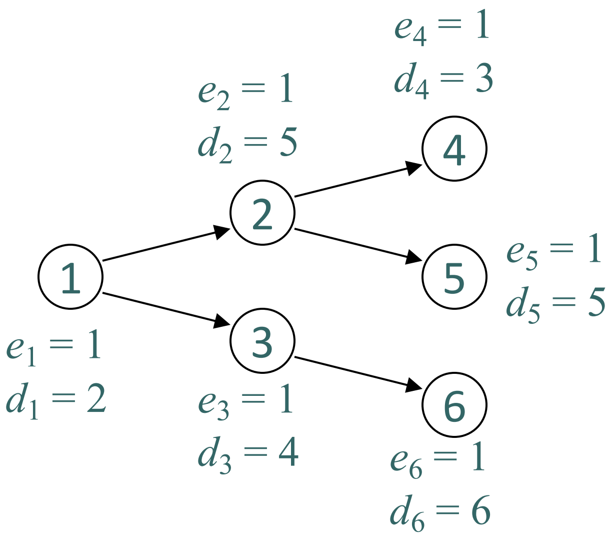
Example: EDF Schedule

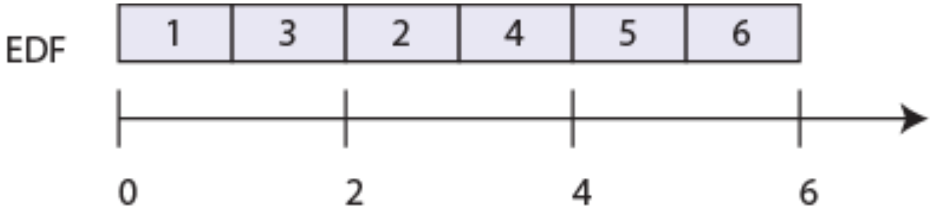




- Is this feasible?

EDF is not optimal under precedence constraints



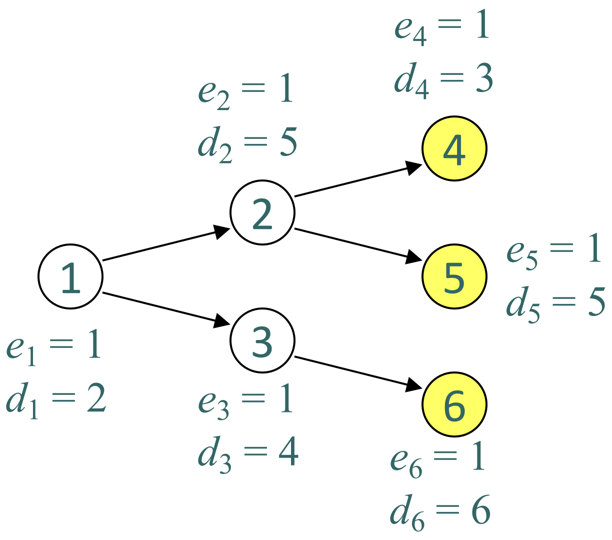


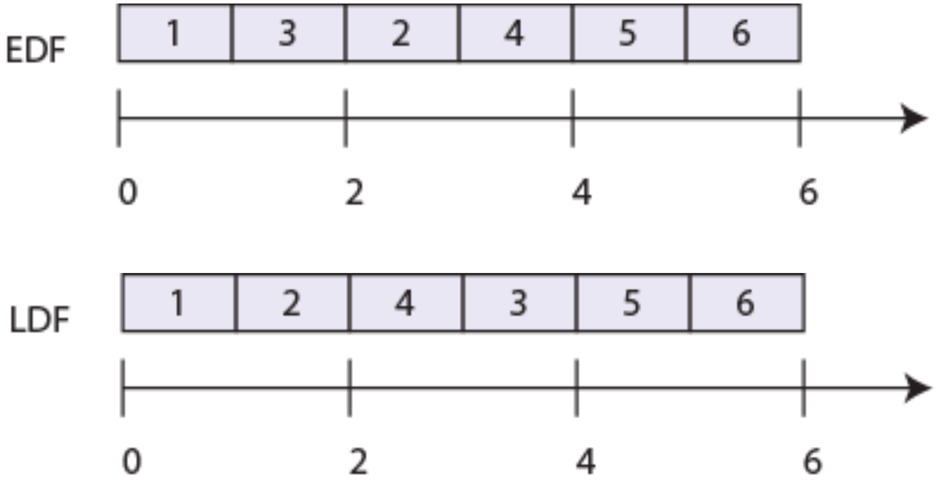
- The EDF schedule chooses task 3 at time 1 because it has an earlier deadline. This choice results in task 4 missing its deadline.

- Is there a feasible schedule?

Latest Deadline First (LDF)

(Lawler, 1973)



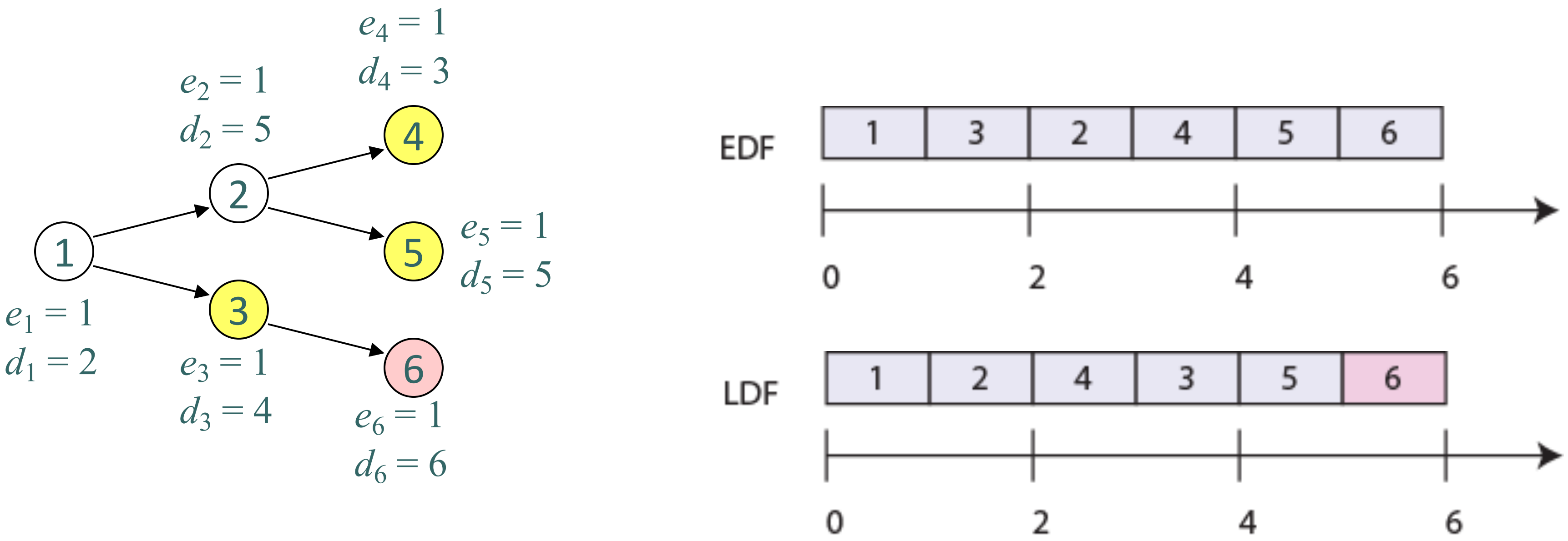


- The LDF scheduling strategy builds a schedule backwards. Given a DAG, choose the leaf node with the latest deadline to be scheduled last, and work backwards.

Lawler, Eugene L. "Optimal sequencing of a single machine subject to precedence constraints." Management science 19.5 (1973): 544-546.

Latest Deadline First (LDF)

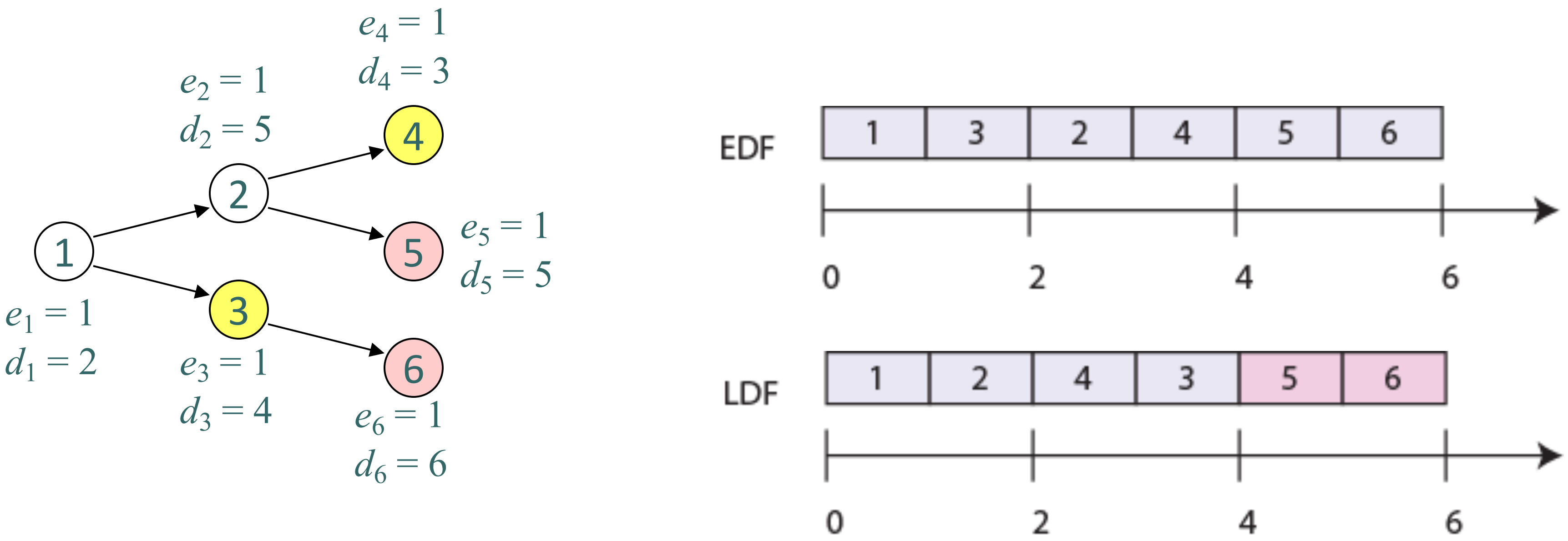
(Lawler, 1973)



- 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)

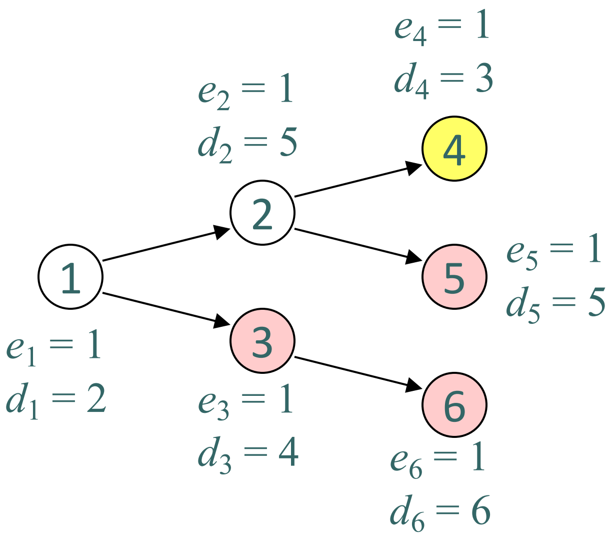
(Lawler, 1973)

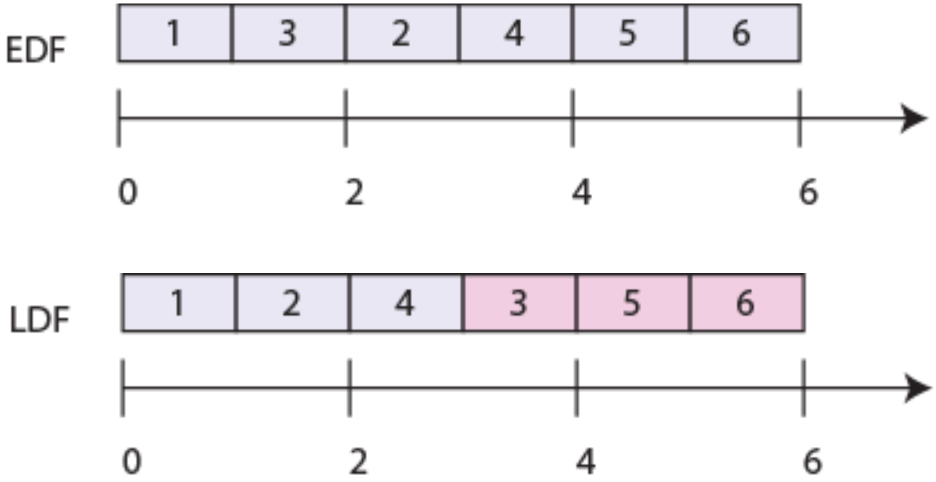


- The LDF scheduling strategy builds a schedule backwards. Given a DAG, choose the leaf node with the latest deadline to be scheduled last, and work backwards.

Latest Deadline First (LDF)

(Lawler, 1973)

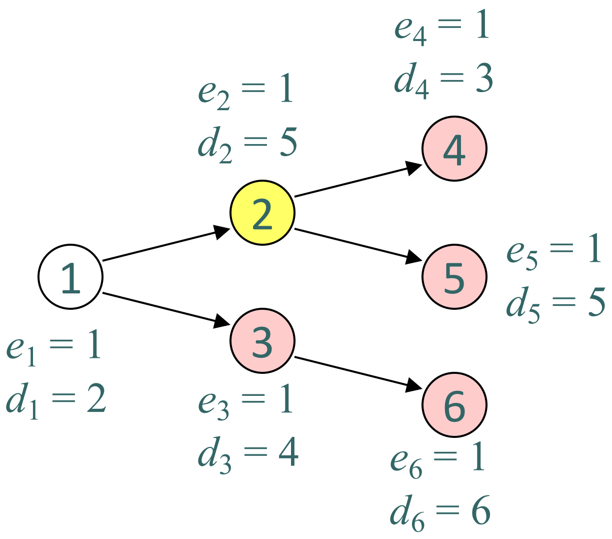


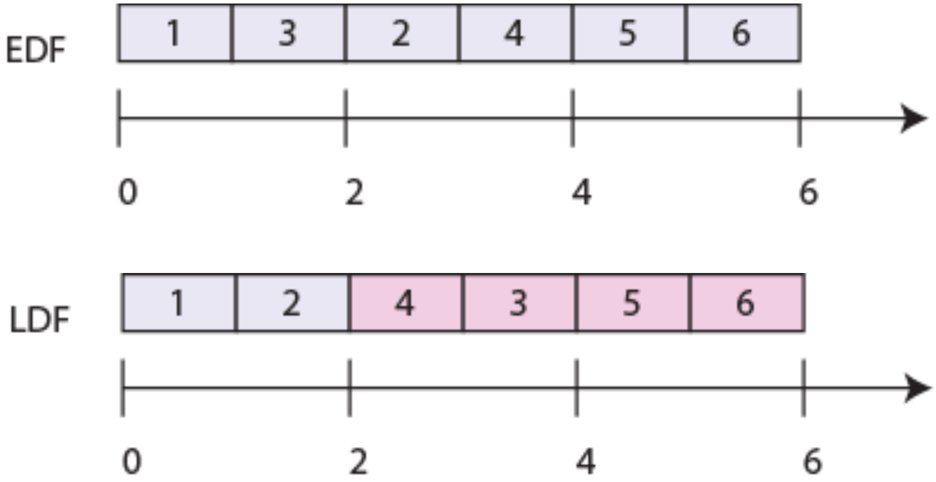


- The LDF scheduling strategy builds a schedule backwards. Given a DAG, choose the leaf node with the latest deadline to be scheduled last, and work backwards.

Latest Deadline First (LDF)

(Lawler, 1973)

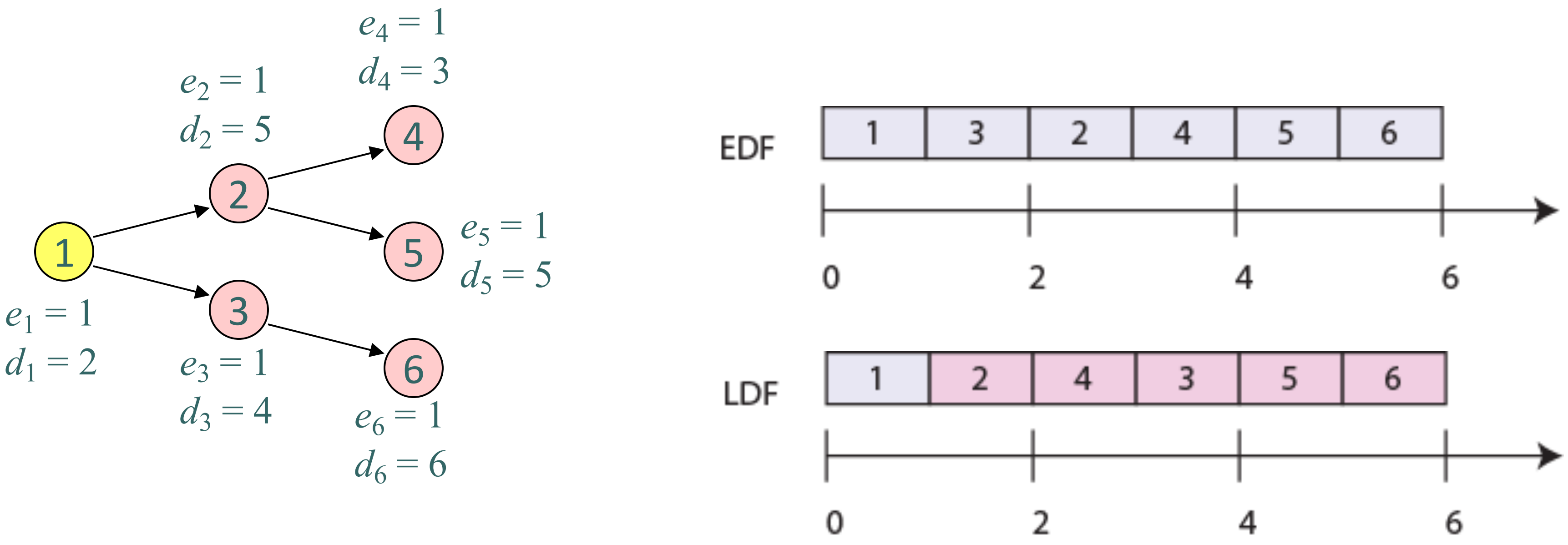




- The LDF scheduling strategy builds a schedule backwards. Given a DAG, choose the leaf node with the latest deadline to be scheduled last, and work backwards.

Latest Deadline First (LDF)

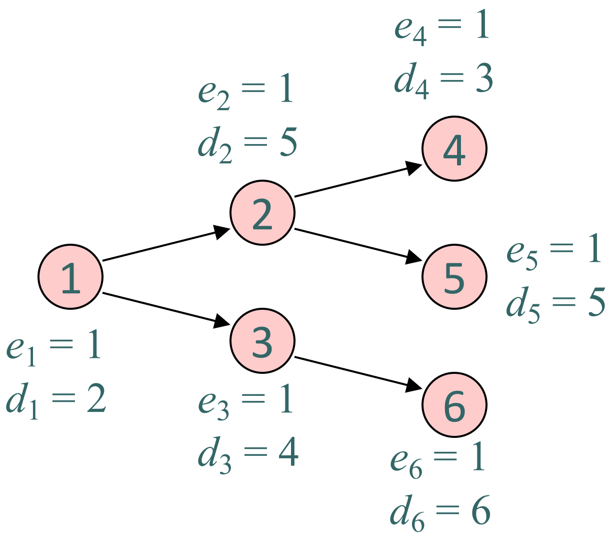
(Lawler, 1973)

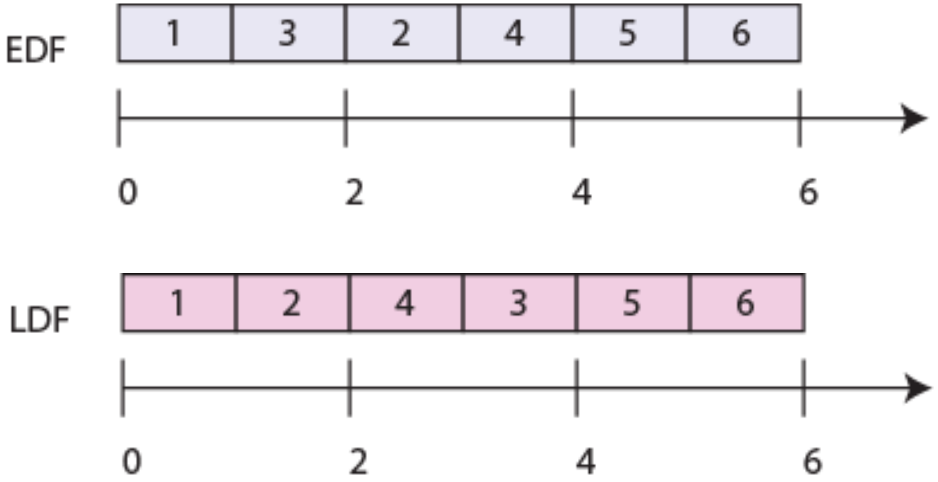


- The LDF scheduling strategy builds a schedule backwards. Given a DAG, choose the leaf node with the latest deadline to be scheduled last, and work backwards.

Latest Deadline First (LDF)

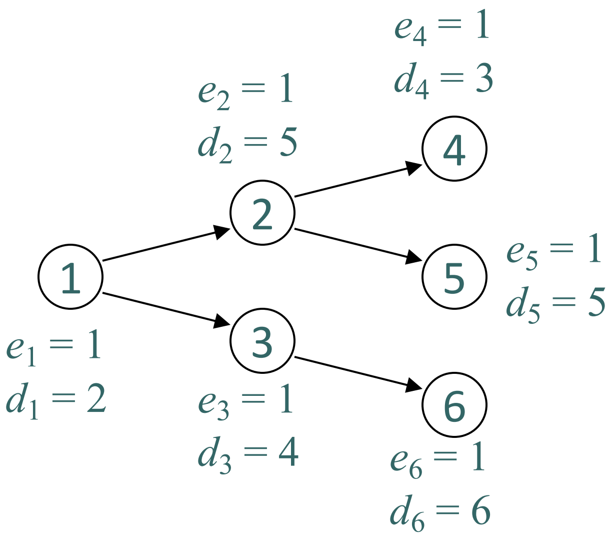
(Lawler, 1973)

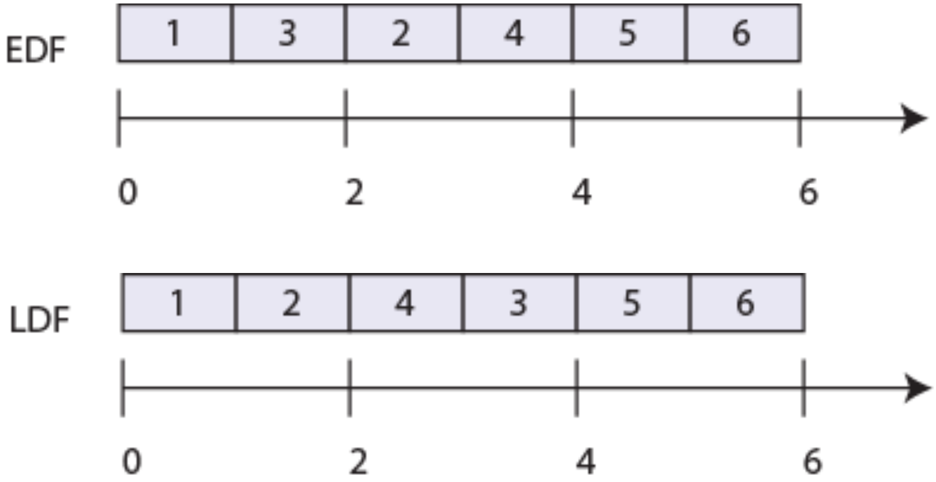




- The LDF scheduling strategy builds a schedule backwards. Given a DAG, choose the leaf node with the latest deadline to be scheduled last, and work backwards.

LDF is optimal under precedence constraints





The LDF schedule shown at the bottom respects all precedences and meets all deadlines.

→ Also minimises maximum lateness

Latest Deadline First (LDF)

(Lawler, 1973)

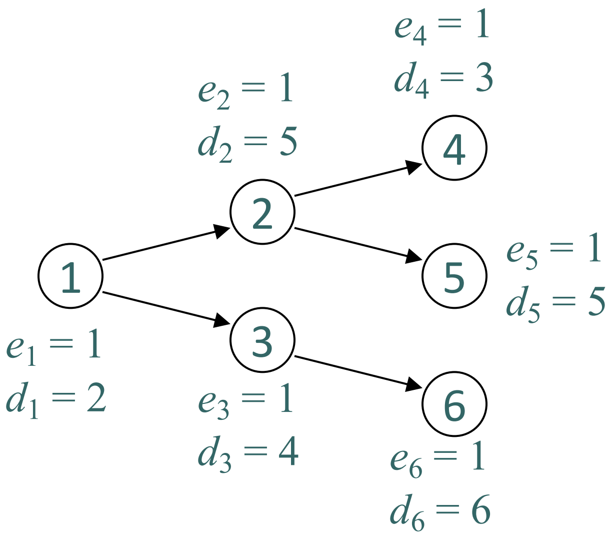
- LDF is optimal in the sense that it minimizes the maximum lateness.

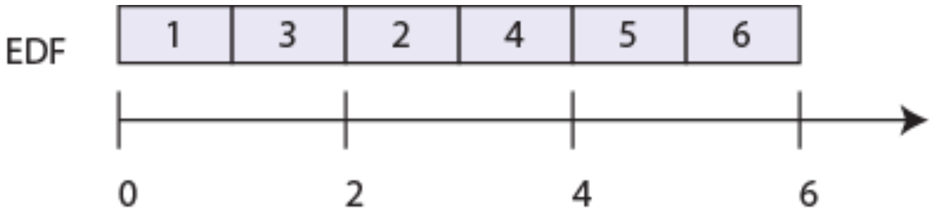
- It does not require preemption. (We'll see that EDF can be made to work with preemption.)

- However, it requires that all tasks be available and their precedences known before any task is executed.
- In other words, it does not support arrival of tasks.
- There is a simple modification of EDF that addresses this issue.

EDF with Precedences (EDF*)

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





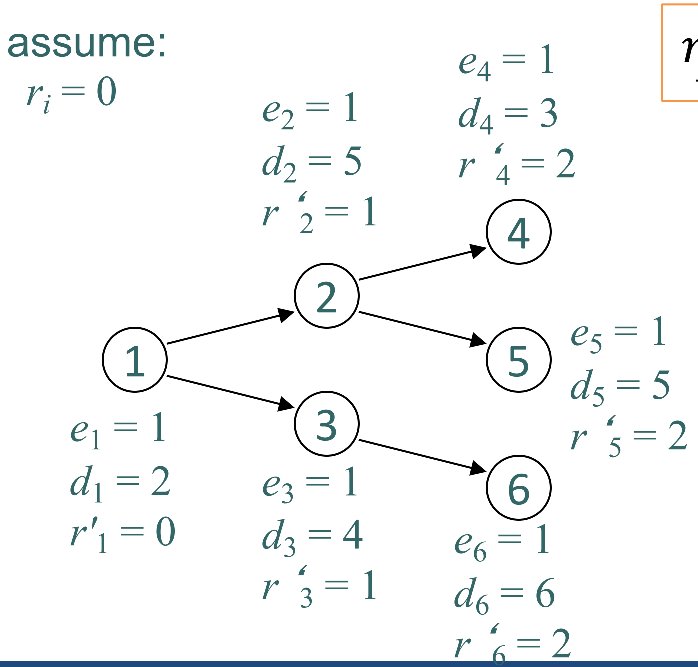
Chetto, H., Silly, M., & Bouchentouf, T. (1990). Dynamic scheduling of real-time tasks under precedence constraints. Real-Time Systems, 2(3), 181-194.

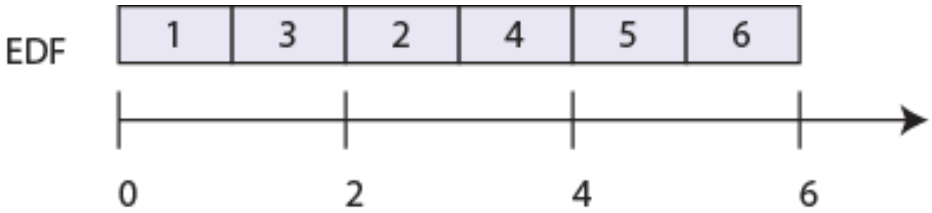
Recall that for the tasks at the left, EDF yields the schedule above, where task 4 misses its deadline.

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:

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



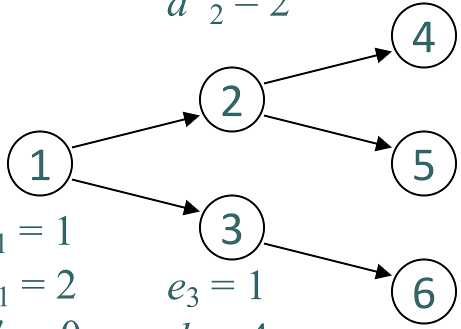


EDF with Precedences
Modifying deadlines

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

$$\boxed{d_i' = \min(\, d_i, d_j' - e_j)}$$

assume:
$r_i = 0$



$e_1 = 1$
$d_1 = 2$
$r'_1 = 0$
$d'_1 = 1$

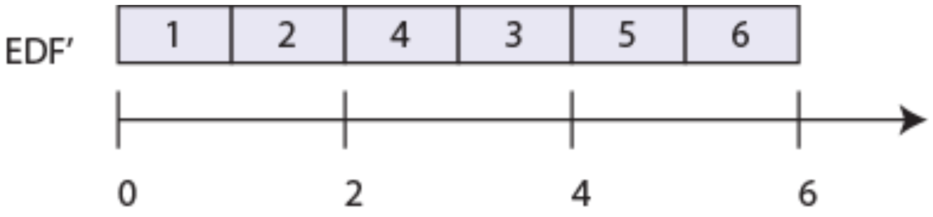
$e_2 = 1$
$d_2 = 5$
$r'_2 = 1$
$d'_2 = 2$

$e_3 = 1$
$d_3 = 4$
$r'_3 = 1$
$d'_3 = 4$

$e_4 = 1$
$d_4 = 3$
$r'_4 = 2$
$d'_4 = 3$

$e_5 = 1$
$d_5 = 5$
$r'_5 = 2$
$d'_5 = 5$

$e_6 = 1$
$d_6 = 6$
$r'_6 = 2$
$d'_6 = 6$



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

Optimality

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

Conclusion

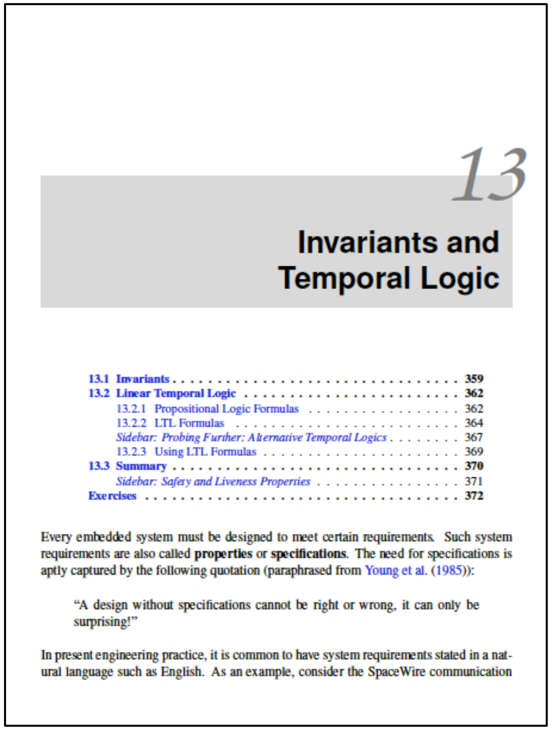
- Timing behavior under all known task scheduling strategies is brittle. Small changes can have big (and unexpected) consequences.
  - Priority Inversion and Deadlock

Performance improvement at a local level, may result in performance degradation at a global level.

- Known as the non-monotonic property of multi-processor schedules.
- Such phenomena are common whenever we interconnect ***dynamical systems*** together.
- Since execution times are so hard to predict, such brittleness can result in unexpected system failures.

Things to do …

- Read Chapter 13
- Assignment 2 deadline is Sept 16



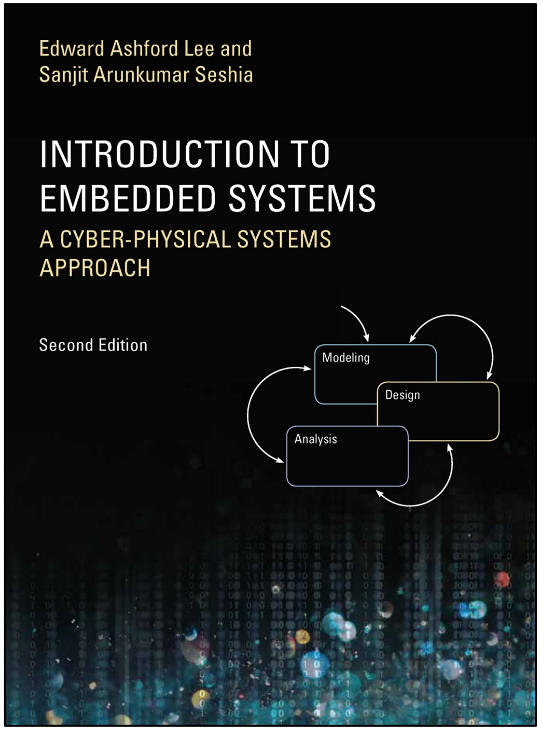
13

Invariants and Temporal Logic

Every embedded system must be designed to meet certain requirements. Such system requirements are also called **properties** or **specifications**. The need for specifications is aptly captured by the following quotation (paraphrased from Young et al. (1985)):

> "A design without specifications cannot be right or wrong, it can only be surprising!"

In present engineering practice, it is common to have system requirements stated in a natural language such as English. As an example, consider the SpaceWire communication



Edward Ashford Lee and
Sanjit Arunkumar Seshia

INTRODUCTION TO EMBEDDED SYSTEMS

A CYBER-PHYSICAL SYSTEMS APPROACH

Second Edition

Next Lecture

- Invariants and temporal logic