

The logo of SUPSI (Scuola Universitaria Professionale della Svizzera Italiana) is a light gray square with a faint grid pattern.

SUPSI

Real-time Operating Systems

Operating Systems

Amos Brocco, Lecturer & Researcher

Objectives

- Study the basic operation of real-time systems
- Study some basic scheduling algorithms for real-time systems

►► Browsing

- Get a rapid overview.

► Reading

- Read it and try to understand the concepts.


📖 Studying

- Read in depth, understand the concepts as well as the principles behind the concepts.

You are also encouraged to try out (compile and run) code examples!




Realtime scheduling

- (RTOS – Real Time Operating System)
- What's a real-time operating system?
 - It's a multitasking operating system for applications with real-time requirements 
- In a real-time application the correctness depends not only on the **logical result** but also on the **time** those results are given.





Characteristics of a real-time system

- A real-time system must be **predictable**:
 - Must satisfy scheduling time constraints: there should be a schedule that satisfies such constraints (in such case we say that the system is **schedulable**) 
 - If the process pool is dynamic, it must verify that scheduling constraints are always fulfilled




Real-time system design

- **Event driven design:**
 - Priority scheduling
 - Tasks are executed in response to events
 - Higher priority tasks pre-empt lower priority ones 
- **Time sharing design:**
 - Round robin scheduling 
 - "More deterministic"



Tasks

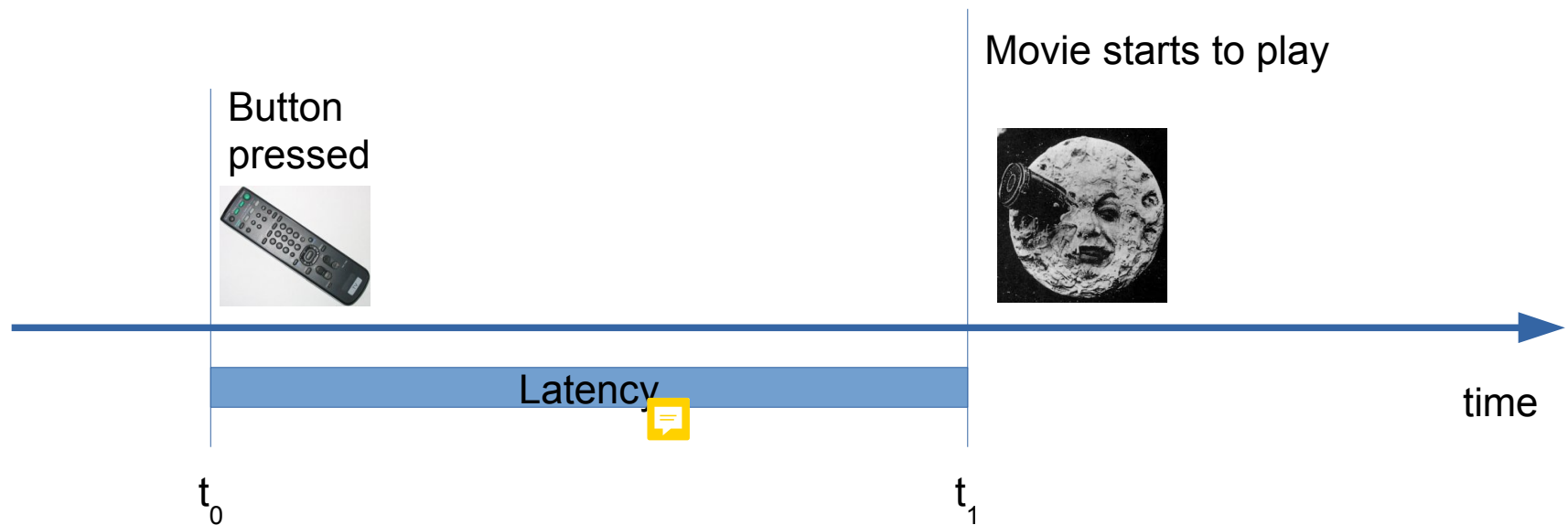
- **Event triggered**
 - Tasks that are executed in response to an (external) event (**aperiodic** tasks) 
Example: the user pushes a button, a valve is opened
- **Time triggered**
 - Tasks are executed at a certain time or at regular intervals (**periodic** tasks)
Example: read the sensor value every t seconds



Latency

- Time between the occurring of an event (*stimulus*) and the response by the software

Example: a DVD Player





Sources of latency

- **Hardware:**
 - Interrupt generation
 - Electric signal propagation
- **Software:**
 - Interrupt handling routines
 - Scheduling
 - Context switching
 - User application
 - Swapping



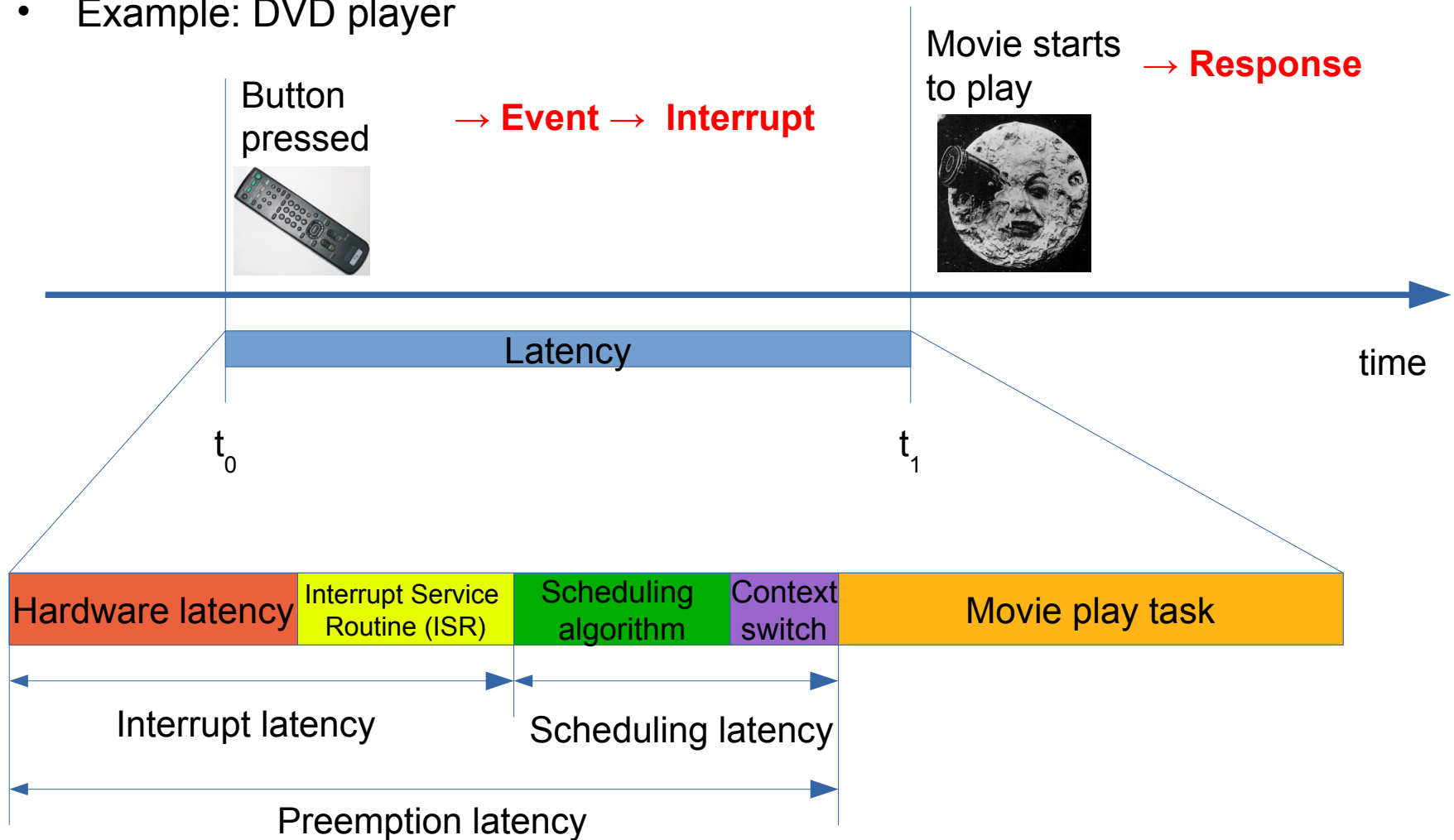
Preemption

- In an multitasking operating system some other task may be running when an event occurs: **preemption** is necessary.
- Preemption means the **ability of the operating system to preempt** or stop a **currently scheduled task** in order to **execute a higher priority task**.
- Event ► Interrupt ► Interrupt handling ► Preemption of current task ► Task Switching ► Response



Preemption latency

- Example: DVD player





Real-time requirements

- **Hard Real-Time**
 - Tasks **must** be completed within a deadline.
 - Failure to meet a deadline cannot be tolerated: a response coming too late is not only useless but may also be **dangerous**.
 - Examples: critical systems supervision and control (airplanes autopilot, nuclear reactors control,...)





Real-time requirements

- **Soft Real-Time**
 - Tasks must be completed within a deadline, but exceptions are tolerated. Task is allowed to finish behind the deadline.
 - Missing of a deadline does not produce serious or disastrous consequences
 - Examples: DVD Player (missing a deadline may just cause a degradation in the image, slow downs, fewer frames-per-second)



Real-time requirements

- **Firm Real-Time**



- Missing a deadline is not dangerous: a response coming too late is just useless.
- Task is not allowed to continue after the deadline: if the task has not finished the result is discarded.
- Examples: videoconferencing (live audio-video transmission)



Schedulability

- **Schedulability** is achieved when deadlines are met for each task → important role played by the **scheduling policy/algorithm**
- Soft real-time:
 - Priority scheduling
- Hard real-time:
 - Dynamic scheduling
 - Static scheduling



Priority scheduling

- Each task is scheduled according to its priority
 - Priorities can be adjusted at run-time
- Provides "best-effort" performances, but...
 - The outcome of the scheduling is **unpredictable**



Dynamic scheduling

- Tasks can be added or removed dynamically to the scheduling queue: the **scheduling order** is determined **at runtime**
 - The system must ensure the validity of the queue at each change (*"is the system still schedulable?"*)
- This approach is **flexible** but...
 - It is hard to ensure correctness





Static scheduling

- The number of tasks and the scheduling order are defined *a priori*: at runtime, scheduling is simple and takes little time, but...
- The system **is not flexible**
 - Limited to periodic (time triggered) tasks
 - Cannot add or remove new tasks
 - Changes require an off-line re-evaluation of the schedule
- ... but its behavior is easier to model and understand



Study of some periodic scheduling algorithms

- We consider periodic tasks t_i with hard time constraints (*deadlines*) 
- Tasks do not depend on each other and have a **known execution time** $X(t_i)$
- Each task is **scheduled periodically** with period P_i , with a **deadline interval** $D(t_i)$ (i.e. it has at most $D(t_i)$ time to complete each execution) 

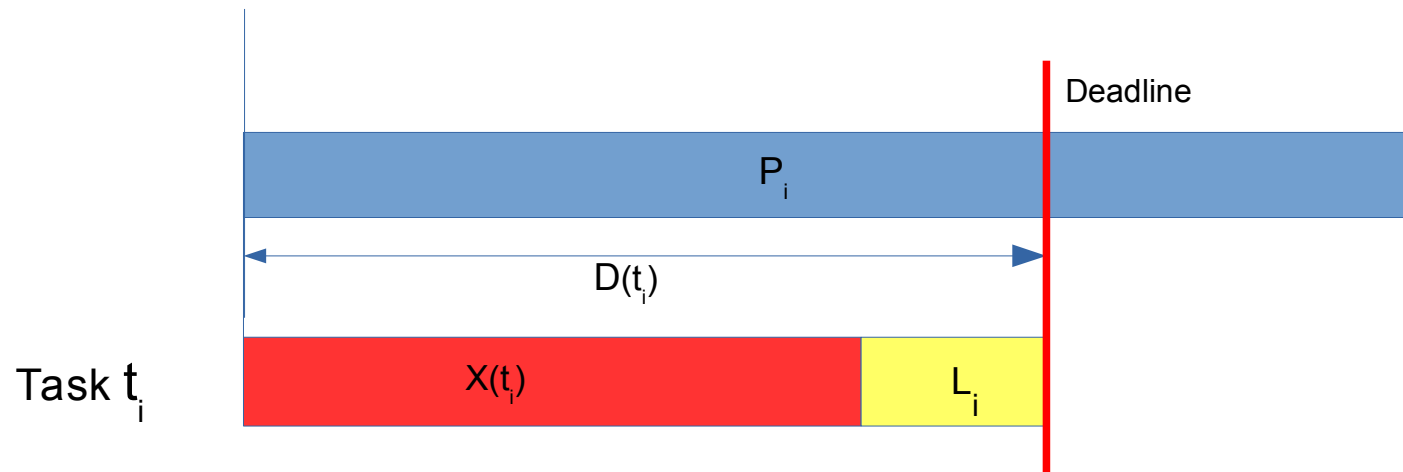


Study of some periodic scheduling algorithms



- We define the **slack** or **laxity time** L_i as

$$L_i = D(t_i) - X(t_i)$$





Schedulability condition

- If we consider $\mathbf{D(t_i)} = \mathbf{P_i}$ we can determine if the set of tasks is schedulable on m processors/cores by computing the **accumulated utilization u** as:

$$u = \sum_i^n X(t_i) / P_i$$



- If the system has m processors/cores we have schedulability if $\mathbf{u \leq m}$



Rate Monotonic Scheduling (RMS)

- Static priority scheduling:
 - Each task has an a priori (at design time) priority
- The rate monotonic scheduling algorithm (RMS) assigns fixed priorities to periodic tasks in order to maximize their "schedulability"
- Basic idea: *"assign the priority of each task according to its period, so that **the shorter the period the higher the priority**"*



Rate Monotonic Scheduling (RMS): assumptions

- Assumptions:
 - Tasks are independent and periodic
 - The deadline interval has the same length as the scheduling period ($D_i = P_i$)
 - The execution time is known and constant
 - Context switch latency is negligible

Accumulated utilization for n tasks

$$u = \sum_i^n X(t_i)/P_i \leq n(2^{1/n} - 1)$$



Schedulability condition for RMS



Optimality of RMS

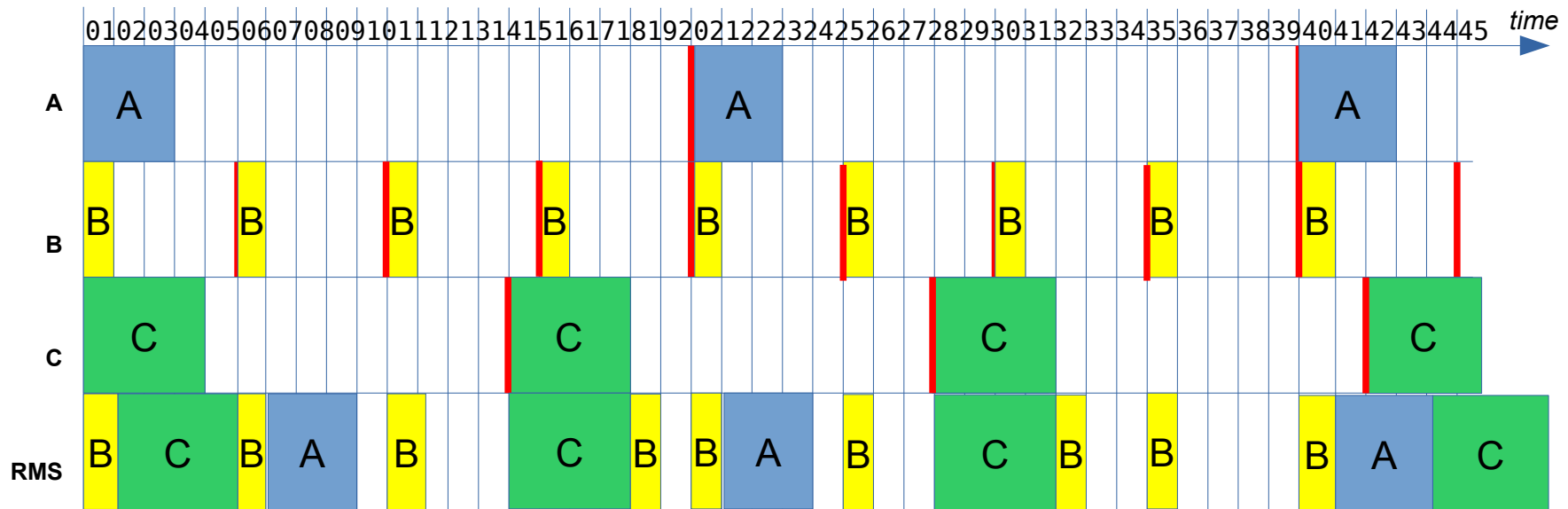
- RMA is the **optimal static-priority algorithm**:
 - If a task set cannot be scheduled using the RMA algorithm, it cannot be scheduled using any other static-priority algorithm.
- Worst case bound for 1 CPU and n tasks: $W_s = n(2^{1/n} - 1)$
 $n \rightarrow \infty, W_s \rightarrow 69\%$

Scheduling example with RMS (1)

- 3 periodic tasks: A (run time 3), B (1), C (4)
- Periods: A = 20, B = 5, C = 14
- Priority: A = $1/20$, B = $1/5$, C = $1/14$ (order B,C,A)
- $U = 3/20 + 1/5 + 4/14 = \sim 0.63 = 63\%$

The shorter the period, the higher the priority

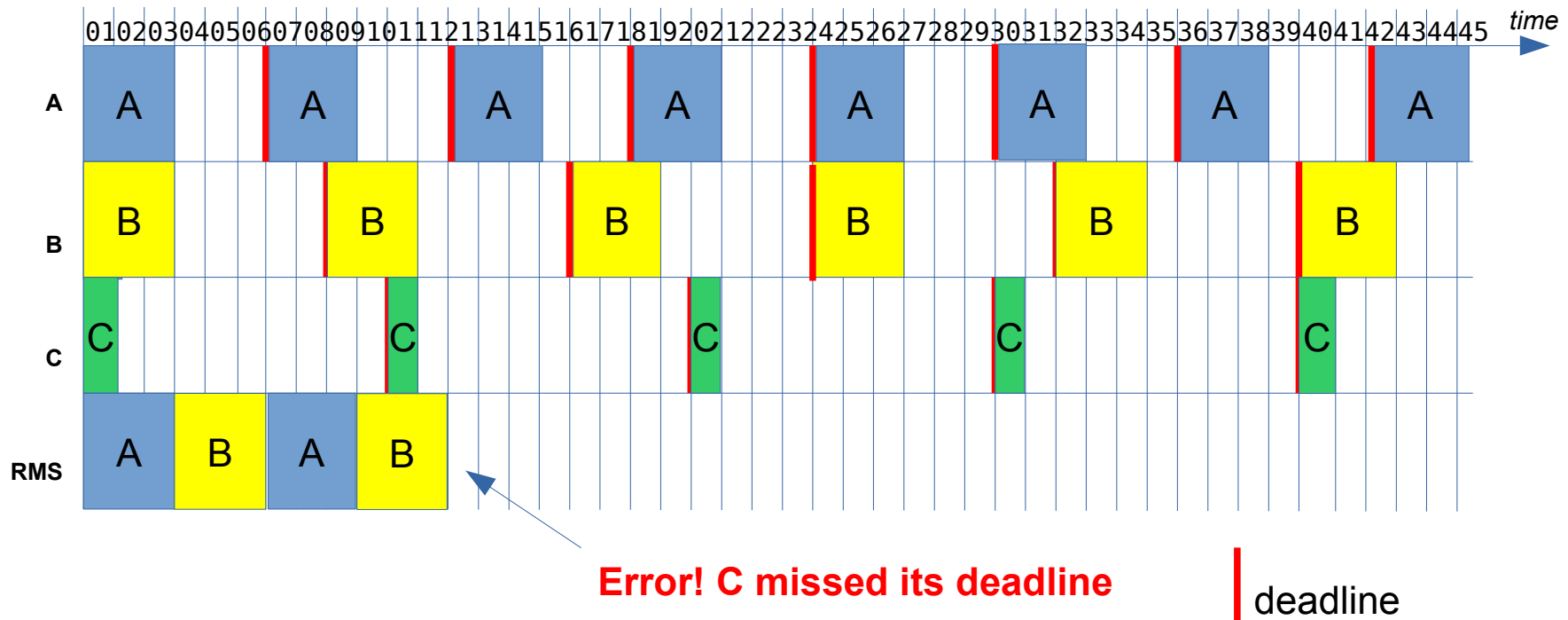
≤ 1 i.e. schedulable on 1 CPU,
 ≤ 0.77 i.e. schedulable with RMS
 $(3 * (2^{1/3} - 1)) = 0.77$



deadline

Scheduling example with RMS (2)

- 3 periodic tasks: A (3) ,B (3) ,C (1)
- Periods: A = 6, B = 8, C = 10
- Priority: A = 1/6, B = 1/8, C = 1/10 (order A,B,C)
- $U = 3 / 6 + 3 / 8 + 1 / 10 = \sim 0.97 = 97\%$ ≤ 1 i.e. schedulable on 1 CPU,
 > 0.77 i.e. **NOT SCHEDULABLE** with RMS





Scheduling Earliest Deadline First (EDF)

- Dynamic priority scheduling
- **Task with the earliest deadline has highest priority** → priorities change during execution
- Utilization bound is 100%: deadlines are guaranteed to be met if the CPU utilization is at most 100%
 - When the system is overloaded, the scheduling becomes unpredictable
- Also works for non-periodic tasks

Scheduling example with EDF

- 3 periodic tasks: A (3) ,B (3) ,C (1)
- Periods: A = 6, B = 8, C = 10
- $U = 3 / 6 + 3 / 8 + 1 / 10 = \sim 0.97 = 97\%$ ≤ 1 i.e. Schedulable with 1 CPU

