MSE | MASTER OF SCIENCE IN ENGINEERING

MA_EmbReal

Scheduling Principles and Periodic Tasks

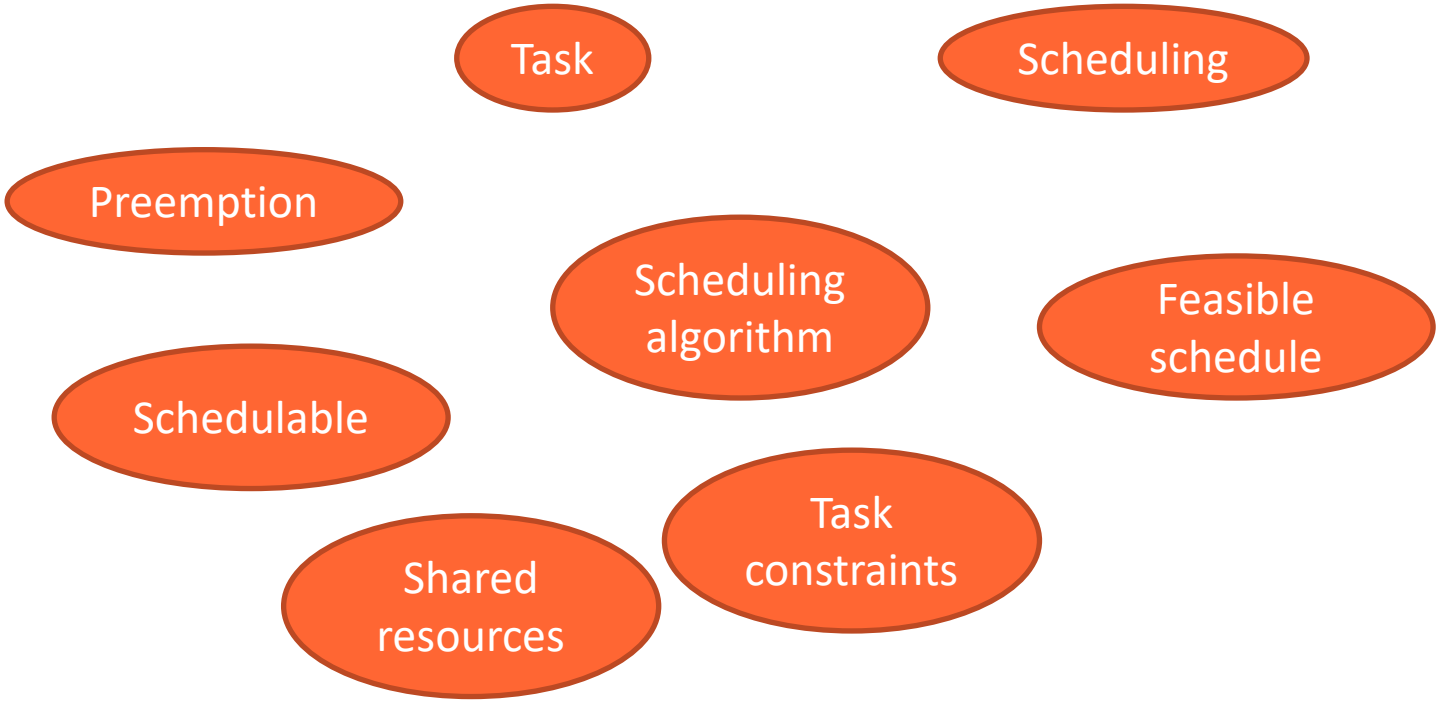
Version: 1.3

Serge Ayer – Luca Haab | 10.03.2025 | Cours MSE

Our mission

- Program with a mix of periodic / aperiodic tasks
  - Address first scheduling of periodic tasks
  - Add aperiodic tasks
  - Add dependencies among tasks
- Demonstrate that a schedule is feasible given a set of tasks with their constraints and dependencies
  - Use known bounds and elaborate a feasible schedule
  - Compute bounds for blocking times
- Use the appropriate scheduling algorithm in simulation and practice
- Implement a system that meets timing constraints
  - With functional safety concepts
  - With timing constraints watchdogs

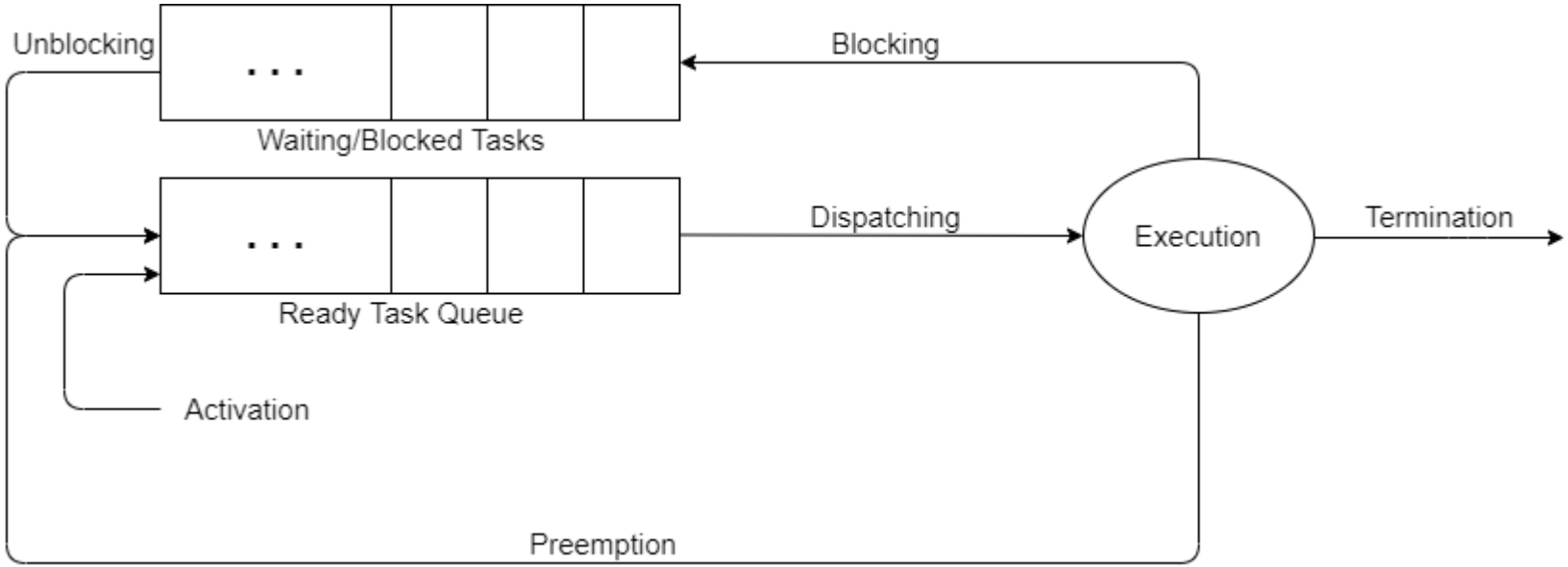
Important concepts



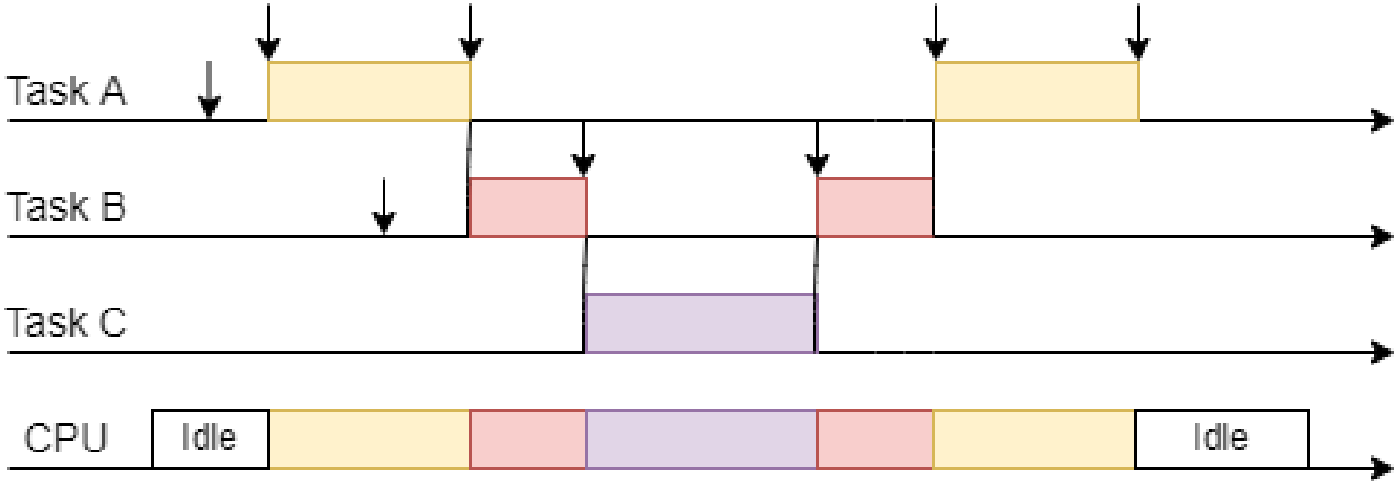
The Scheduling Problem

- Given
  - a set of processors,
  - a set of tasks,
    - with their precedence relations
    - with their timing constraints
  - a set of resources
- Assign the processors and resources to tasks in order to complete all tasks under the specified constraints.

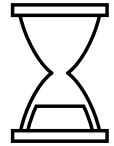
Scheduling of tasks

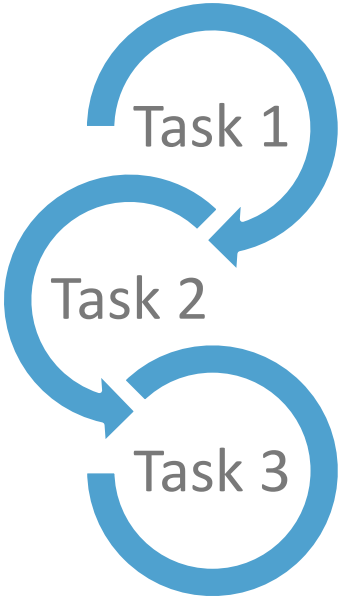


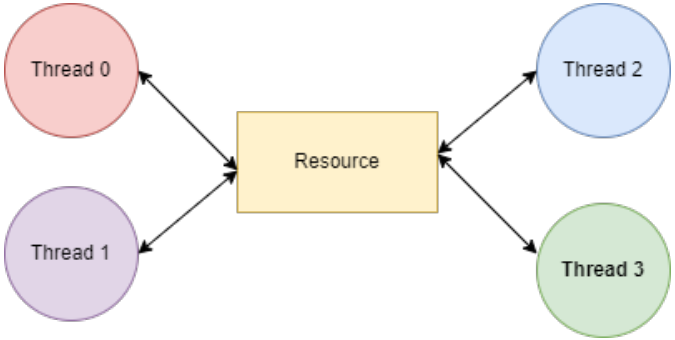
A schedule example



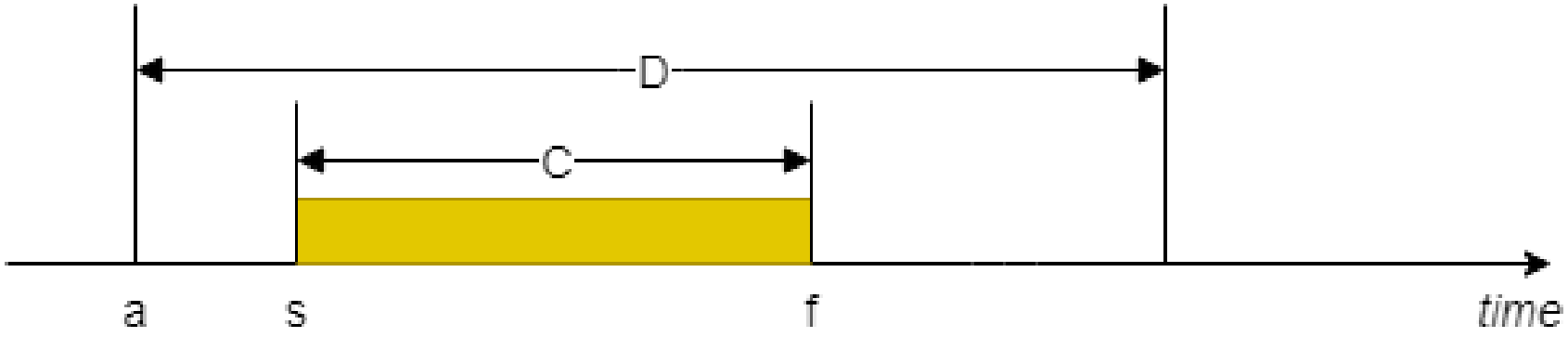
Task constraints



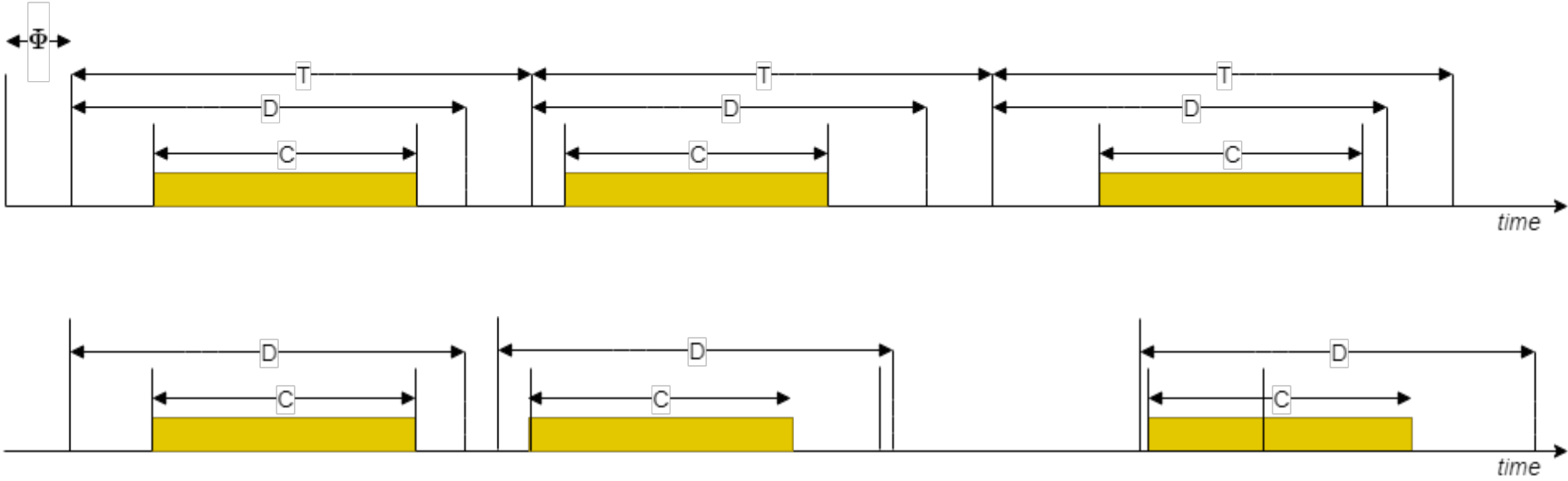




How are timing constraints specified?



Periodic vs aperiodic tasks



Periodic Tasks: Assumptions

- Algorithms are elaborated assuming a number of constraints
  - Tasks' periods T_i are constant for each task
  - The worst-case execution time of each task C_i is known and fixed
  - The deadline is the same as the period, $T_i = D_i$
  - Tasks are independent from each other
- Also, for the sake of simplicity
  - No overhead in the kernel (e.g. context switch)
  - No task suspends itself

Periodic Tasks: Definitions and Concepts

- As a consequence, a task i is entirely characterized by
  - Its period T_i
  - Its worst-case computation time C_i
  - Its phase ϕ_i or release time of its first instance
- The schedule repeats itself at the Hyperperiod / Major Cycle interval

  hyperperiod = $lcm(T_1, \dots, T_n)$
- Schedulability depends on the computational load
  - $U = \sum_{i=1}^{n} \frac{C_i}{T_i}$ (processor utilization factor)
  - U_{ub}: upper bound of U for schedulability
    - for a task set AND a scheduling algorithm
  - For schedulability, U cannot be > 1 !

Periodic Tasks: Scheduling Algorithms

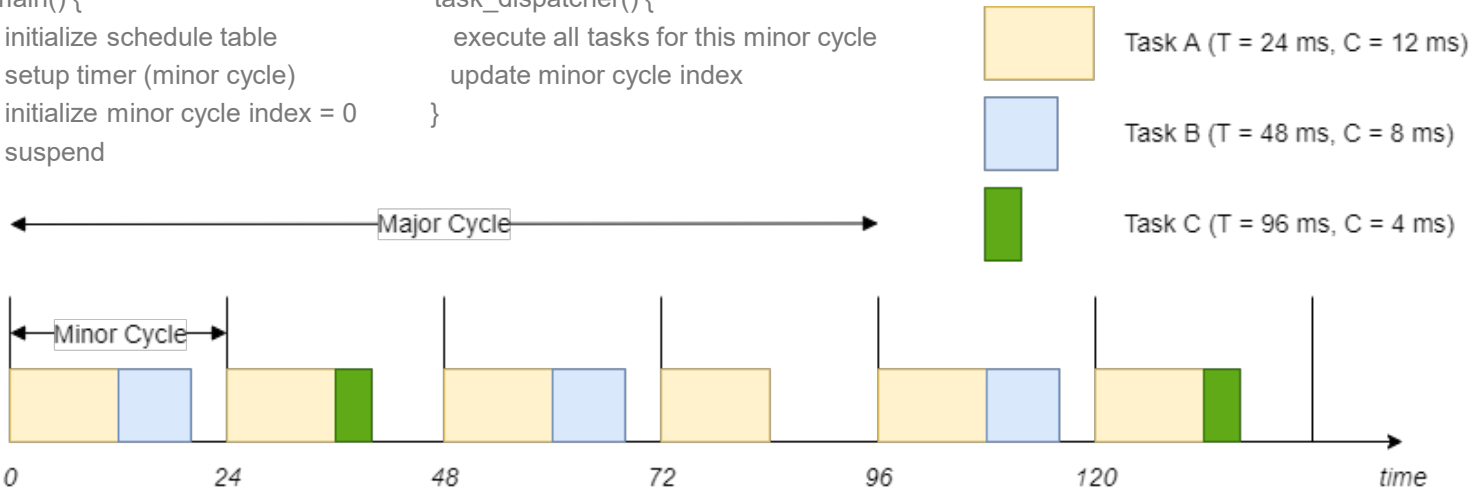
- A scheduling algorithm must guarantee that
  - each task is activated at the proper rate
  - each task is completed within its deadline
  - If T == D, then each task should execute once within its period
- Four basic most-known algorithms:
  - Time-Triggered Cyclic Executive (TTCE)
  - Rate Monotonic (RM)
  - Earliest Deadline First (EDF)
  - Deadline Monotonic (DM)

Time-Triggered Cyclic Executive

- Offline scheduling, without context switch
- Very often used for handling periodic tasks
- Method:
  - See exercise exercise on timeline cyclic scheduling
- Pseudo-code:

```
main() {
  initialize schedule table
  setup timer (minor cycle)
  initialize minor cycle index = 0
  suspend
}
```

```
task_dispatcher() {
  execute all tasks for this minor cycle
  update minor cycle index
}
```



Time-Triggered Cyclic Executive

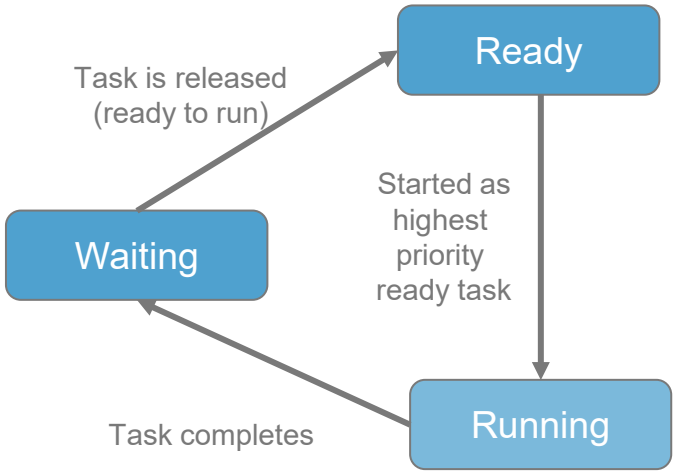
- Advantages
  - Very simple, does not even require an online scheduler
  - No overhead for context switch
  - Minimal jitter
- Disadvantages
  - Very fragile to overloads (domino effect)
  - Does not scale and adapt easily, even to minor changes
  - Difficult to handle aperiodic tasks efficiently

Dynamic scheduling

- Rather than applying a static order of tasks, allow task scheduling to be computed dynamically online:
  - Based on importance (priority) or any other criteria (e.g. task deadline, duration or creation time).
  - This also simplifies the creation of tasks with arbitrary rates.
- Scheduling based on task importance
  - Prioritization means that less important tasks don't delay more important ones.
- How often does the scheduler decide what to run?
  - Coarse grain: after a task finishes. It is non preemptive or Run-To-Completion (RTC)
  - Fine grain: at any time. It is preemptive – one task can preempt another less important task.

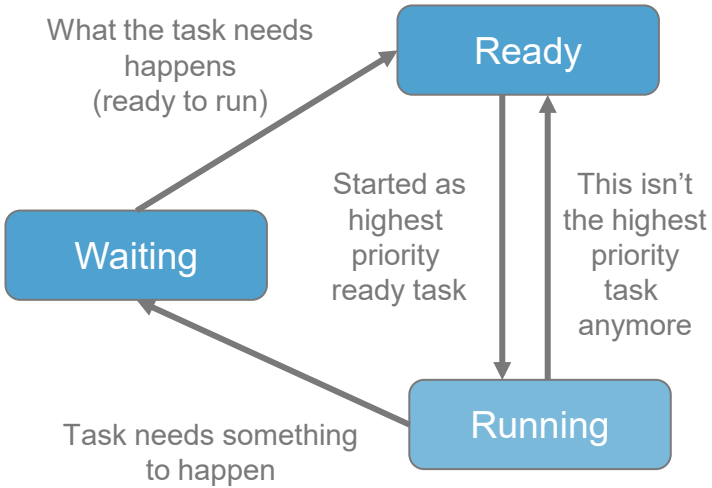
RTC: Task State and Scheduling Rules

- The Scheduler chooses among Ready tasks for execution based on priority
- Events can change a task state
- Scheduling rules:
  - If no task is ready, the scheduler sits in idle state.
  - If no task is running, the scheduler starts the highest priority ready task, if any.
  - Once started, a task runs until it completes (no preemption).
  - Completed tasks enter the waiting state until released again



Preemption: Task State and Scheduling Rules

- The Scheduler chooses among Ready tasks for execution based on priority
- Scheduling rules:
  - A task's activities may lead it to waiting (blocked)
  - A waiting task never gets the CPU. It must be signaled by an ISR or another task.
  - Only the scheduler moves tasks between ready and running



Preemptive Scheduling Algorithms (Periodic Tasks)

- Accepted constraints for RM and EDF
  - No resource sharing
  - D = T, periods are fixed, worst-time execution times are fixed
- Rate Monotonic Scheduling
  - Tasks with higher request rates/shorter periods have higher priorities.
  - Fixed periods means fixed priorities.
  - Is optimal among fixed-priority algorithms.

Preemptive Scheduling Algorithms (Periodic Tasks)

- Rate Monotonic Scheduling
  - Schedulability test: $\mathrm{U} = \sum_{i=1}^{n} \frac{C_i}{T_i} \leq \mathrm{n}(2^{\frac{1}{n}} - 1)$
  - Schedulability test (Hyperbolic bound): $\prod_{i=1}^{n}(U_i + 1) \leq 2$
  - Schedulability test conditions are sufficient conditions, not mandatory
  - The Hyperbolic bound is less pessimistic
  - The upper bound may also be relaxed when tasks' periods form harmonic subsets
    - An set of harmonic tasks is a set where any task's period is a multiple of all shorter periods
    - Example: $[T_1, T_2, T_3, T_4]$ = [1, 3, 6, 12]
    - The schedulability test becomes $\mathrm{U} = \sum_{i=1}^{n} \frac{C_i}{T_i} \leq \mathrm{K}(2^{\frac{1}{K}} - 1)$, where K is the number of task subsets
    - If K=1, the upper bound becomes 1.0, representing full utilization

Preemptive Scheduling Algorithms (Periodic Tasks)

- Earliest Deadline First
  - Tasks with earlier absolute deadlines will be executed at higher priorities.
  - Priorities are dynamic since absolute deadlines of periodic tasks vary over time.
  - Preemptive: the currently executed task is preempted whenever another periodic instance with an earlier deadline becomes active.
  - Schedulability test: $\mathrm{U} = \sum_{i=1}^{n} \frac{C_i}{T_i} \leq 1$
- Deadline monotonic
  - Both RM and EDF assume that the deadline is the end of the period
    - A task instance can be executed anytime within its period
  - Deadline monotonic is an extension of RM that releases this constraint
  - Like RM, it uses fixed priorities and the priority is inversely proportional to the deadline