reSilient coMputer archItectures and LIfE Sciences

Politecnico di Torino

Department of Control and Computer Engineering

SCHEDULING PERIODIC TASKS

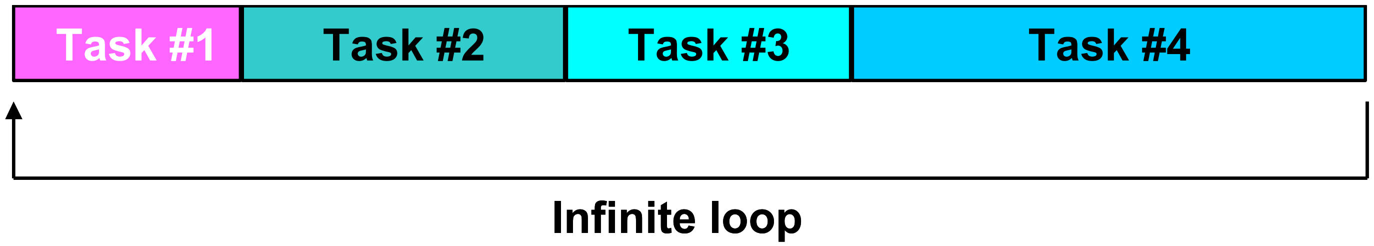
STEFANO DI CARLO

SCHEDULING OF PERIODIC TASKS

- The following algorithms are considered:
  - Timeline scheduling
  - Rate Monotonic (RM) scheduling
  - Earliest Deadline First (EDF) scheduling

TIMELINE SCHEDULING

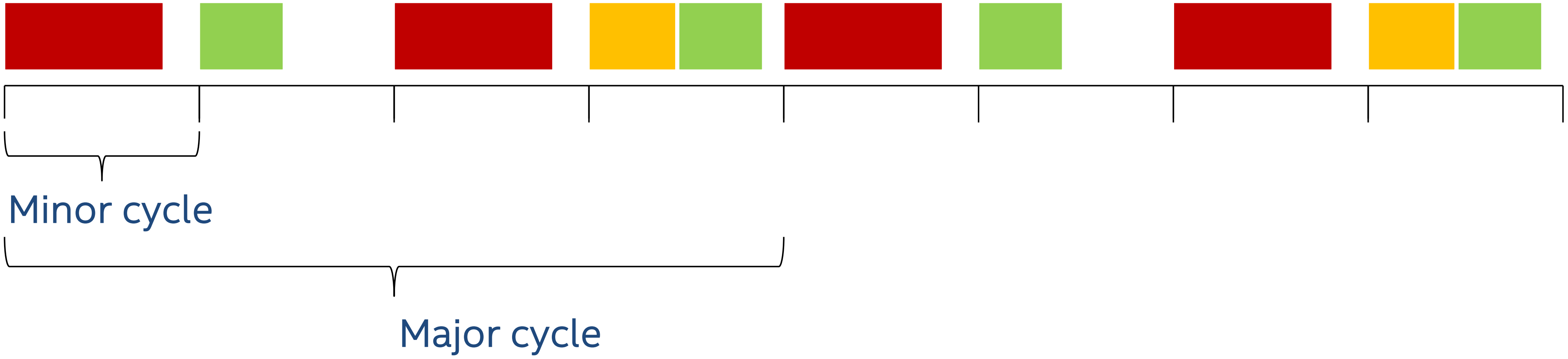
- Certain application does not require complex operating systems
  - Independent tasks executed in sequential fashion



  - No need for IPC, neither synchronization
  - Tasks cannot be interrupted → no preemption
    - Example: tasks manage acquisition of data using one shared resource (e.g., ADC)

TIMELINE SCHEDULING

- The temporal axis is divided into slots of equal length called <u>minor cycles</u>
- One or more tasks can be allocated for execution into minor cycles, in such a way to respect the frequencies derived from the application requirements
- A timer synchronizes the activation of the tasks at the beginning of each time slot
- A sequence of minor cycle repeated identically is called <u>major cycle</u>



TIMELINE SCHEDULING

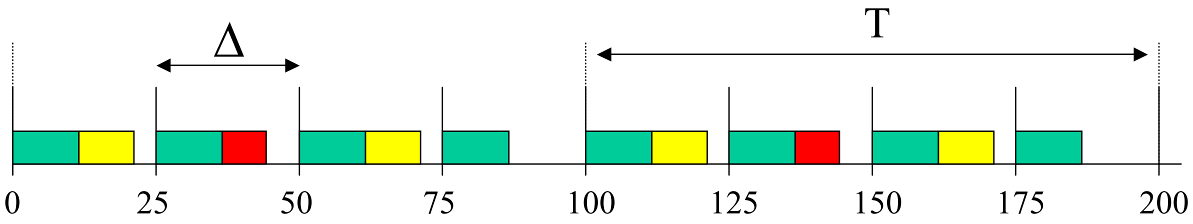
- Minor cycle
  - It is the greatest common divider of all the task periods
- Major cycle
  - It is the least common multiplier of all the task periods
- The scheduling is feasible if the sum of the WCET for the tasks in the minor cycle is at most equal to the minor cycle

- Example
  - Three tasks (A, B, C):
    - T_A = 25 ms (every 25 ms it must run)
    - T_B = 50 ms (every 50 ms it must run)
    - T_C = 100 ms (every 100 ms it must run

task	f	T
A	40 Hz	25 ms
B	20 Hz	50 ms
C	10 Hz	100 ms

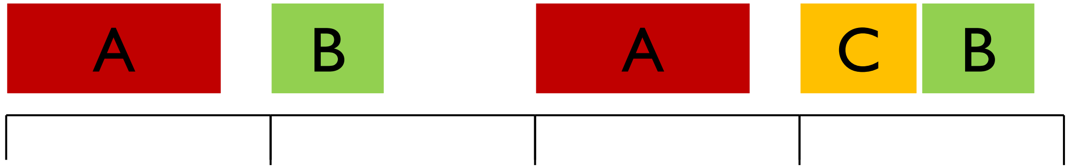
Δ = GCD (minor cycle)

T = lcm (major cycle)



TIMELINE SCHEDULING

- The implementation can be done very easily
  - Each task is coded as a function
  - Each minor cycle is implemented as a function that call each task allotted in the minor cycle
  - The major cycle is a endless loop that call each minor cycle function
  - The execution of the minor cycle function call is regulated by an interrupt timer programmed with the minor cycle duration




```
Minor_1()                    Major()
{                            {
 A();                          while(1)
}                              {
                                 Minor_1();
Minor_2()                        wait_timer();
{                                Minor_2();
 B();                            wait_timer();
}                                Minor_1();
                          wait_timer();
                          Minor_3();
                                 wait_timer();
Minor_3()                      }
{                             }
 C();
 B();
}
```

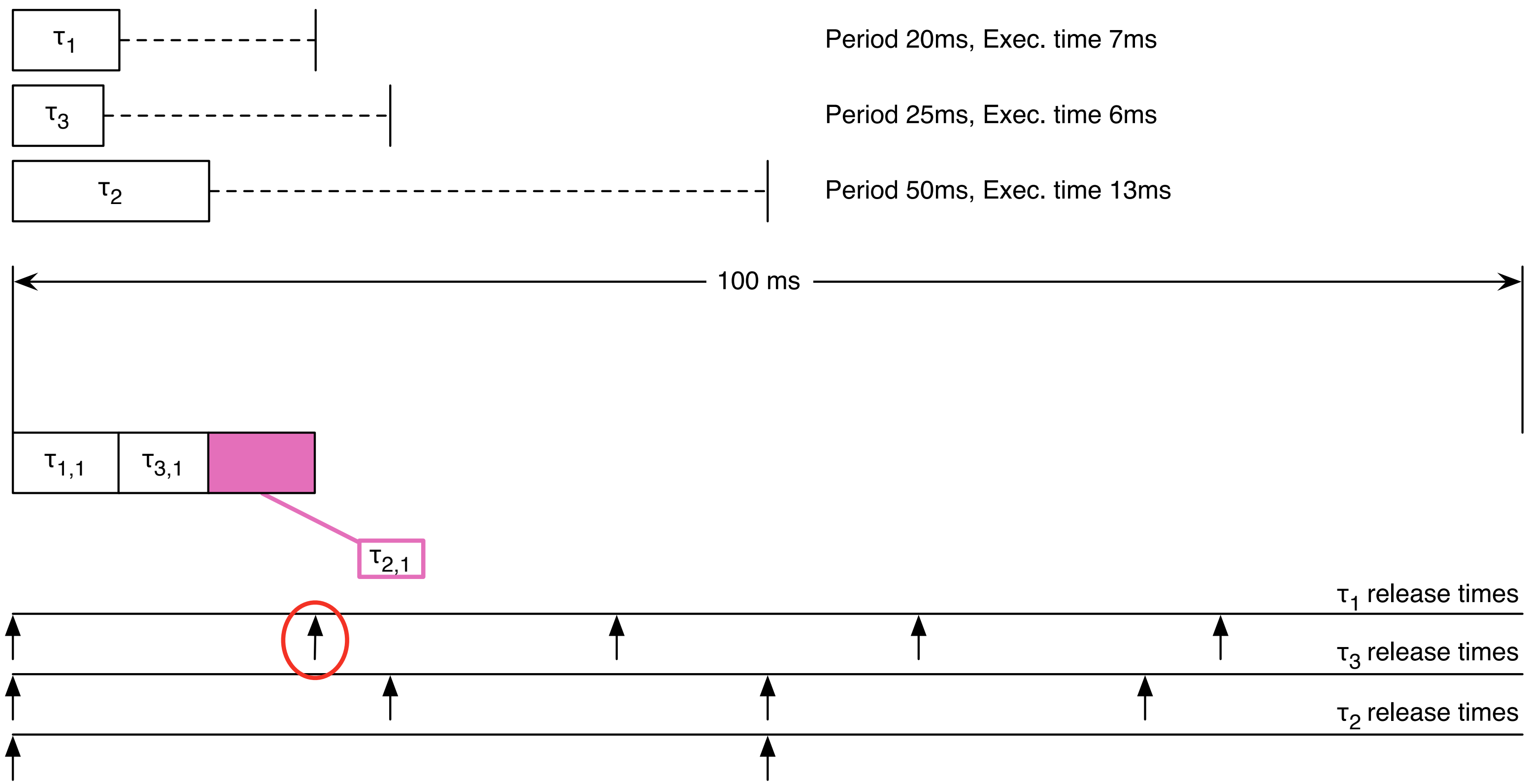
NON-HARMONIC PERIODS

- If the MCD is 1, it means the task periods are non-harmonic, meaning they don't share a common divisor greater than 1.
  - There are no simple, repeatable intervals where tasks will naturally align.
  - Task executions are less likely to overlap or synchronize over time.
- If the MCD is 1, the LCM could be quite large, requiring a longer major cycle to represent a complete, repeating schedule
- Impacts on Timing and Efficiency:
  - Complexity: A larger LCM increases the complexity of the schedule because the timeline needs to accommodate various tasks that might only synchronize after a long period.
  - Increased Context Switching: Tasks with very different periods will require more frequent context switches, possibly leading to overhead in the system.
  - Potential for Gaps in the Schedule: When tasks are non-harmonic and their MCD is 1, there might be periods where no tasks are executing, or some tasks may need to be scheduled more frequently to meet their deadlines, increasing the complexity of managing idle time.

RATE MONOTONIC SCHEDULING

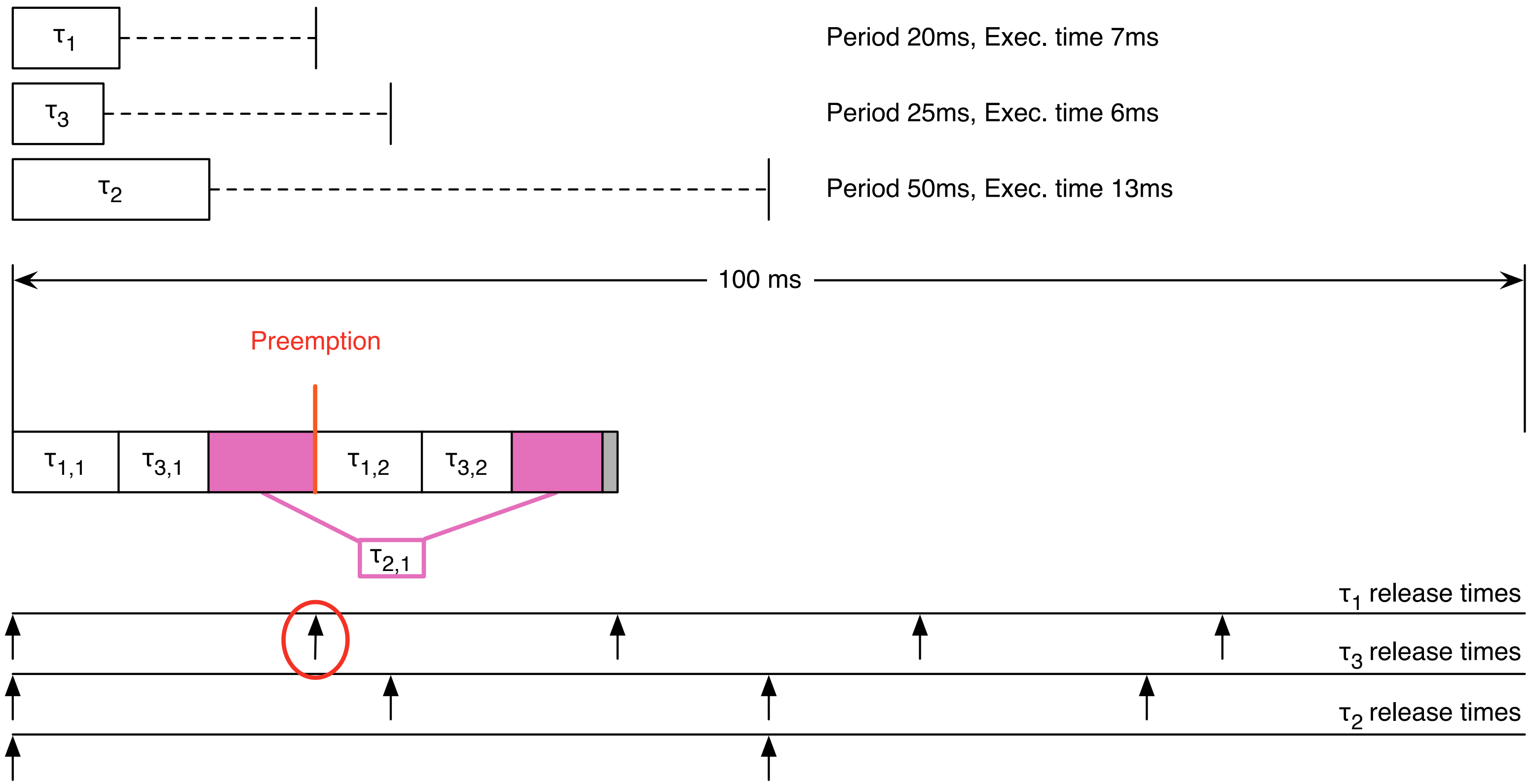
- Fixed priority scheduling
  - Each process has a fixed, static priority computed offline, before run-time
  - The ready processes are scheduled according to their priority
- Hypothesis
  - Basic process model (deadline=period)
  - Tasks have static priority
  - Scheduler is preemptive
  - One processor
- Scheduling algorithm
  - Each task is assigned a fixed priority that is inversely proportional to its period: the shorter the period, the higher the priority

EXAMPLE



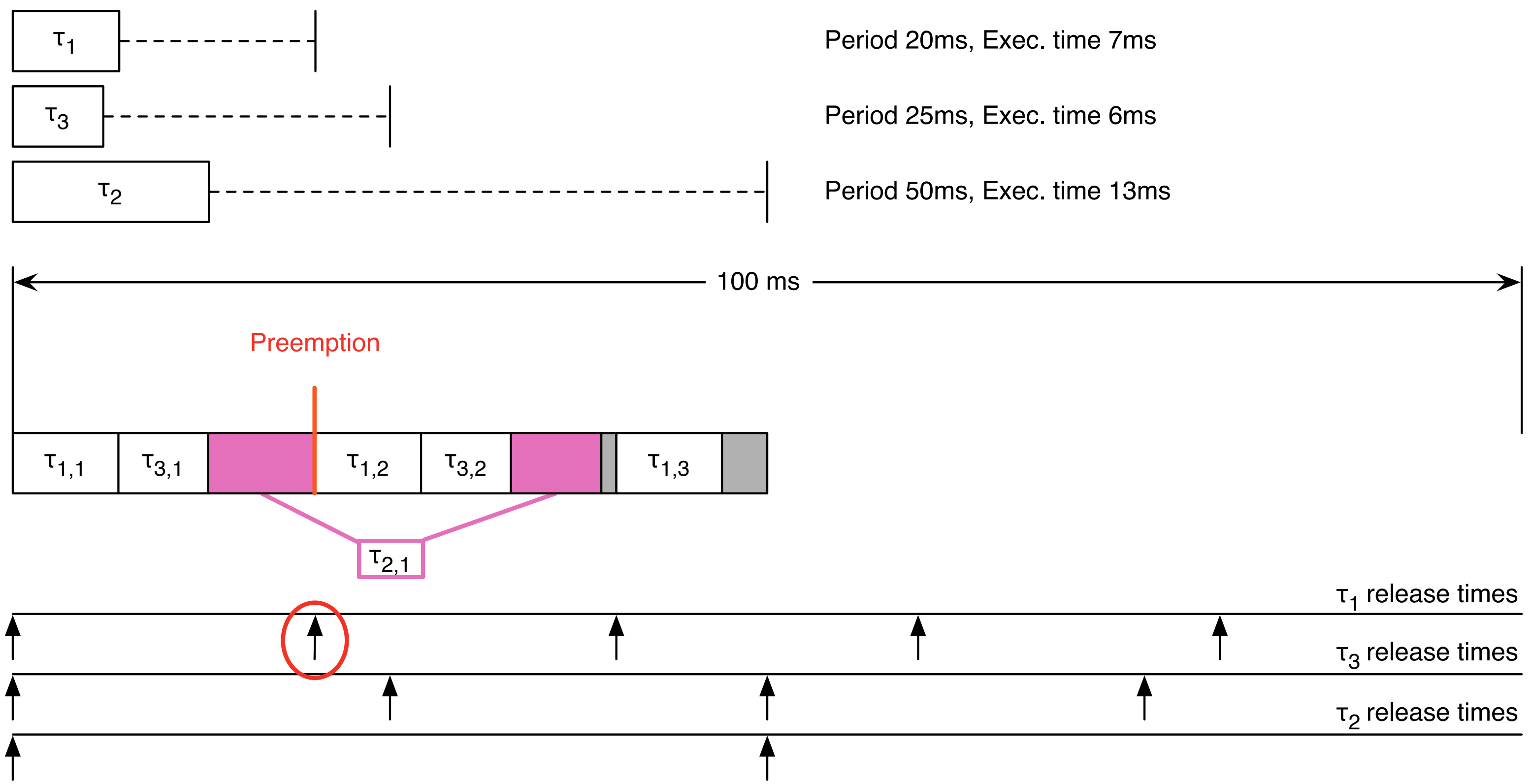
At $t = 0$, all tasks are ready: the first one to be executed is τ_1 then, at its completion, τ_3. At $t = 13$, τ_2 finally starts but, at $t = 20$, τ_1 is released again.

EXAMPLE



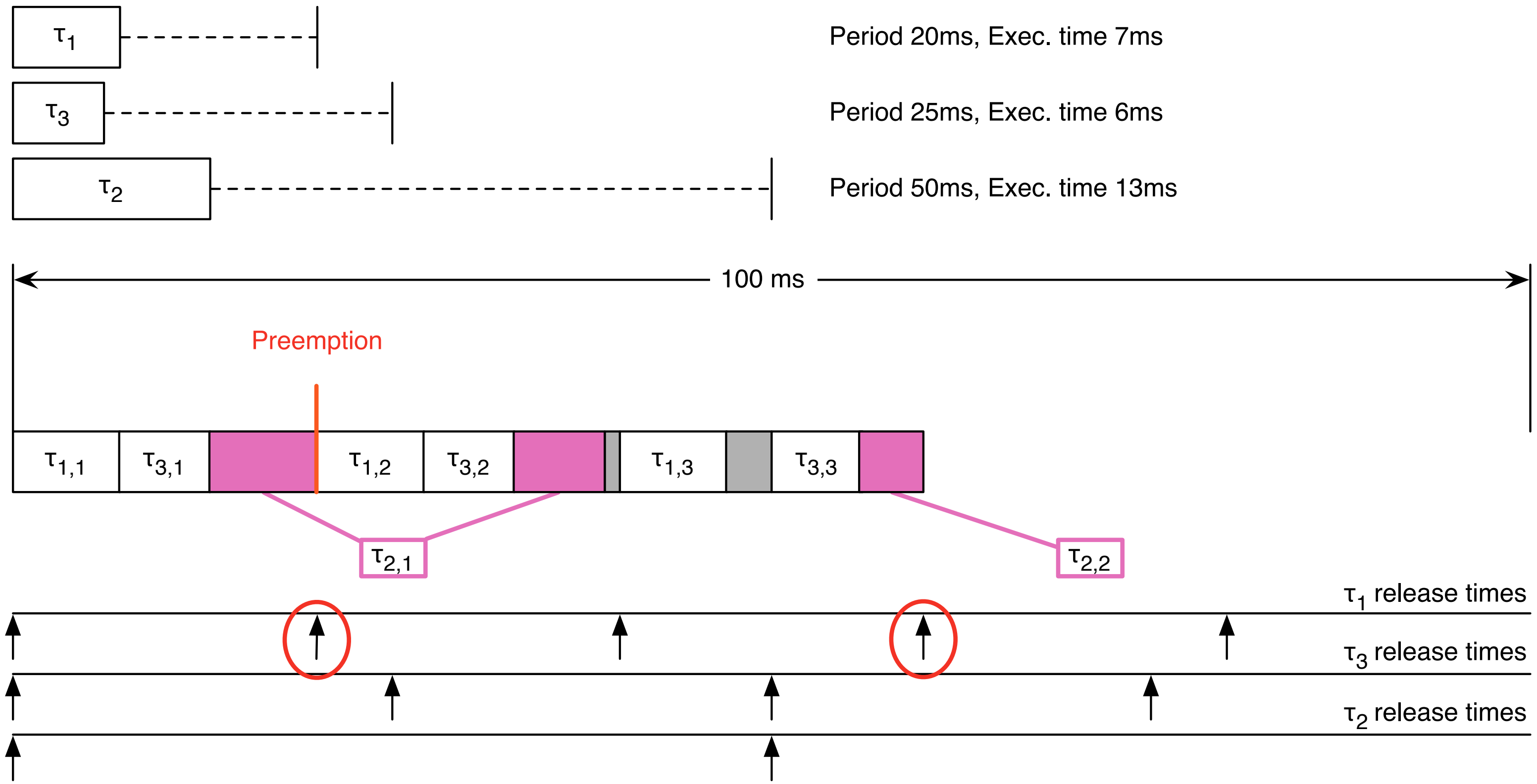
Hence, τ_2 is preempted in favor of τ_1. While τ_1 is executing, τ_3 is released, but this does not lead to a preemption: τ_3 is executed after τ_1 has finished. Finally, τ_2 is resumed and then completed at $t = 39$.

EXAMPLE



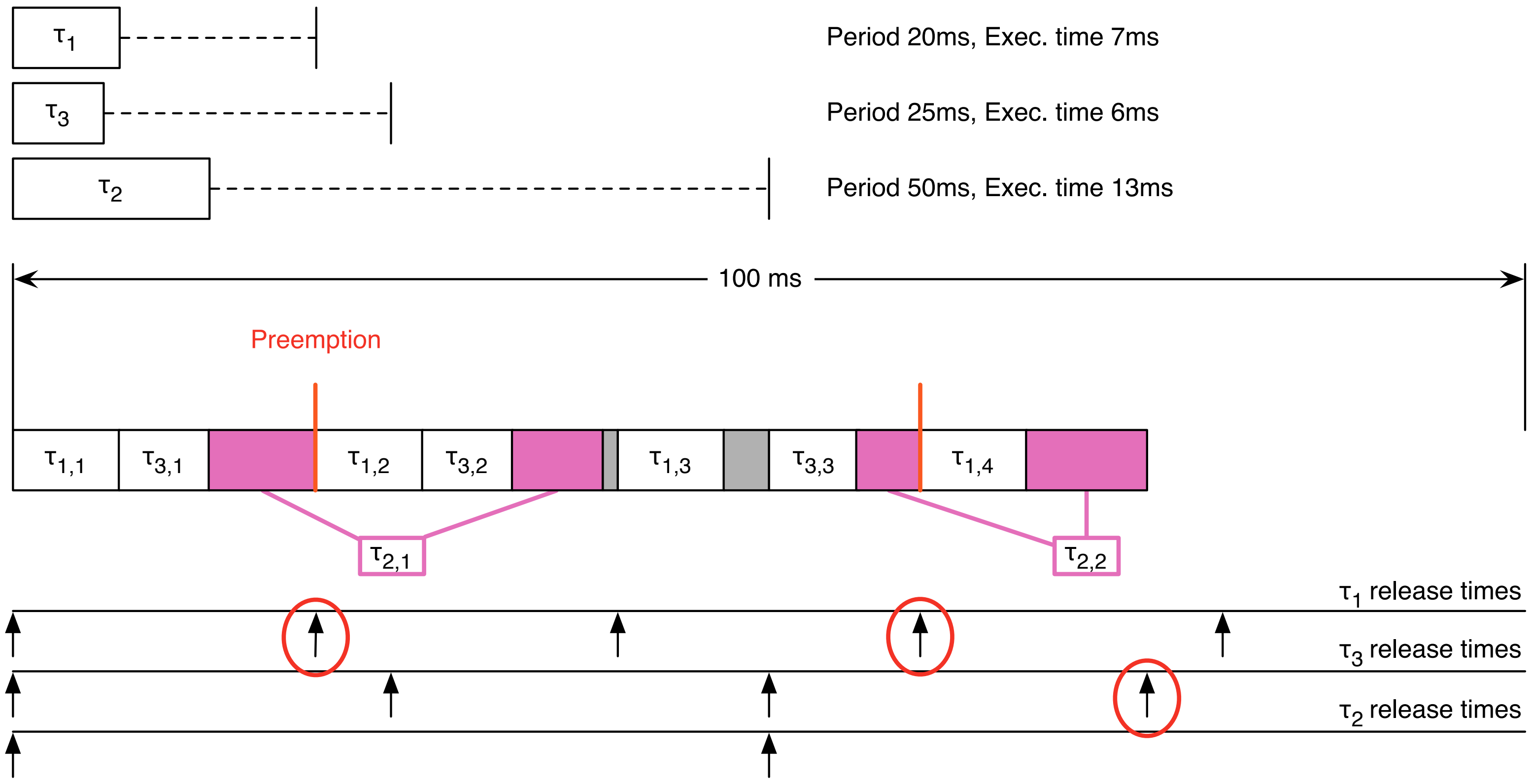
At $t = 40$, after 1 ms of idling, task τ_1 is released. Since it is the only *ready* task, it is executed immediately, and completes at $t = 47$.

EXAMPLE



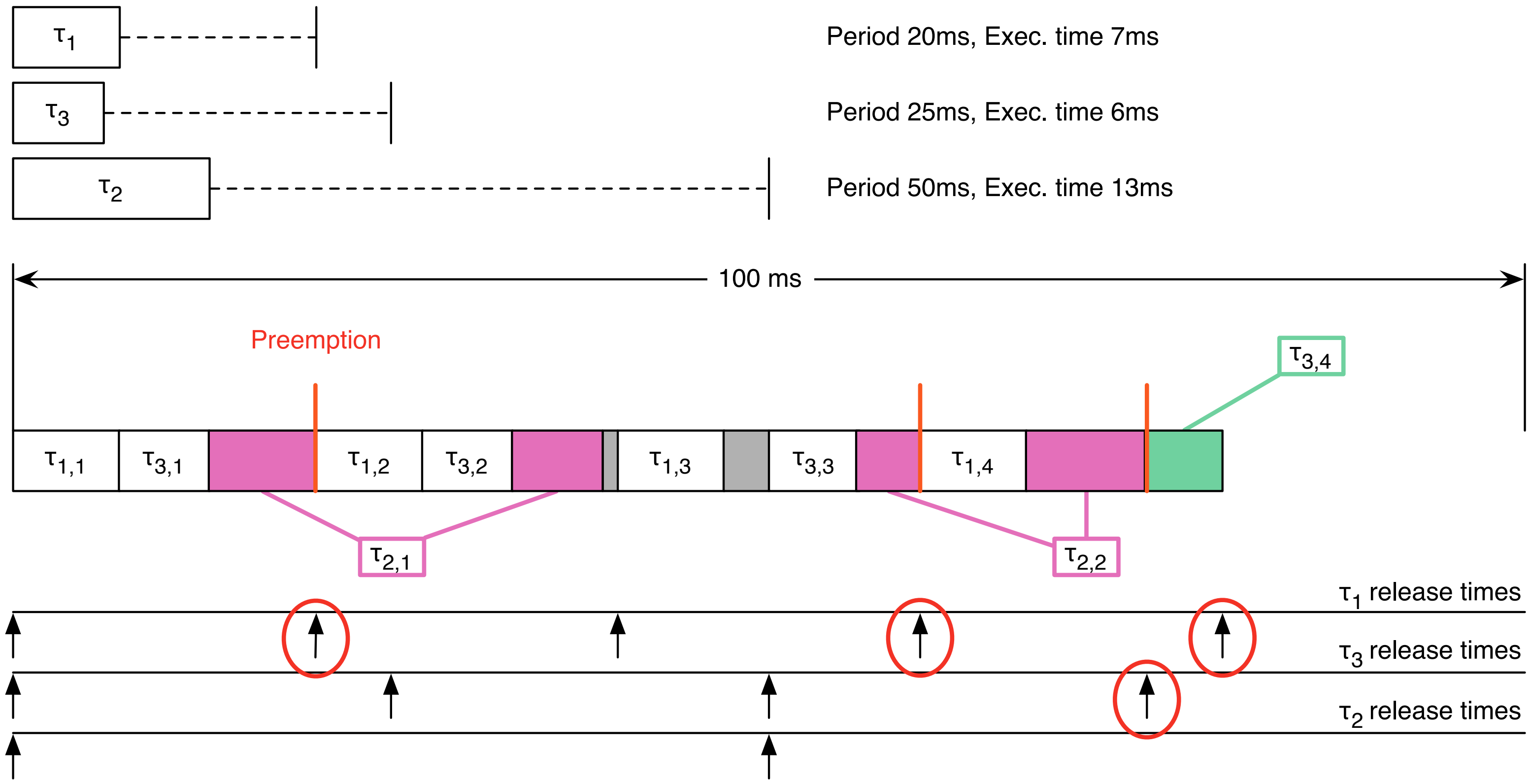
At $t = 50$, both τ_3 and τ_2 become *ready* simultaneously. τ_3 is run first, then τ_2 starts and runs for 4 ms. However, at $t = 60$, τ_1 is released again.

EXAMPLE



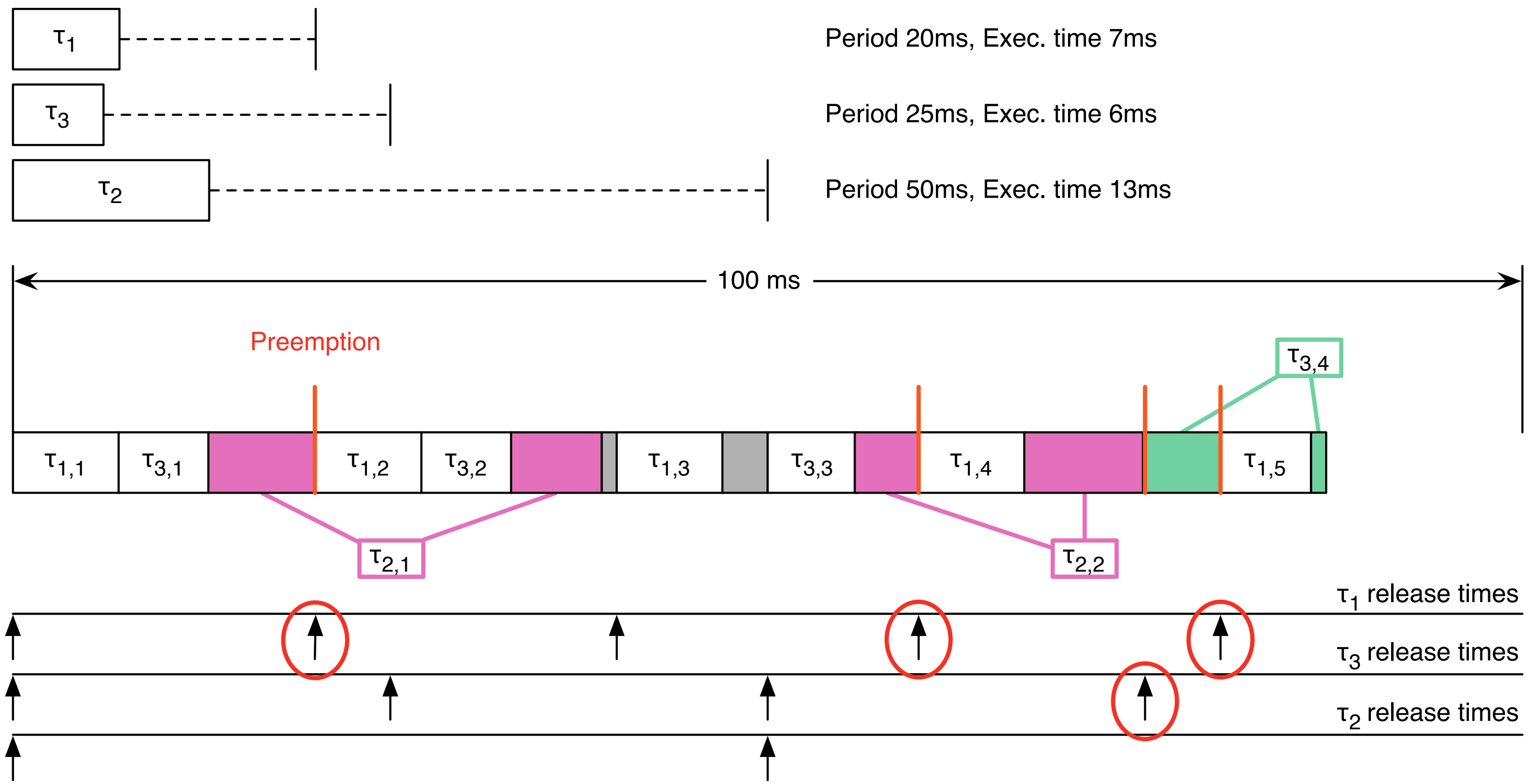
As before, this leads to the preemption of τ_2 and τ_1 runs to completion. Then, τ_2 is resumed and runs for 8 ms, until τ_3 is released.

EXAMPLE



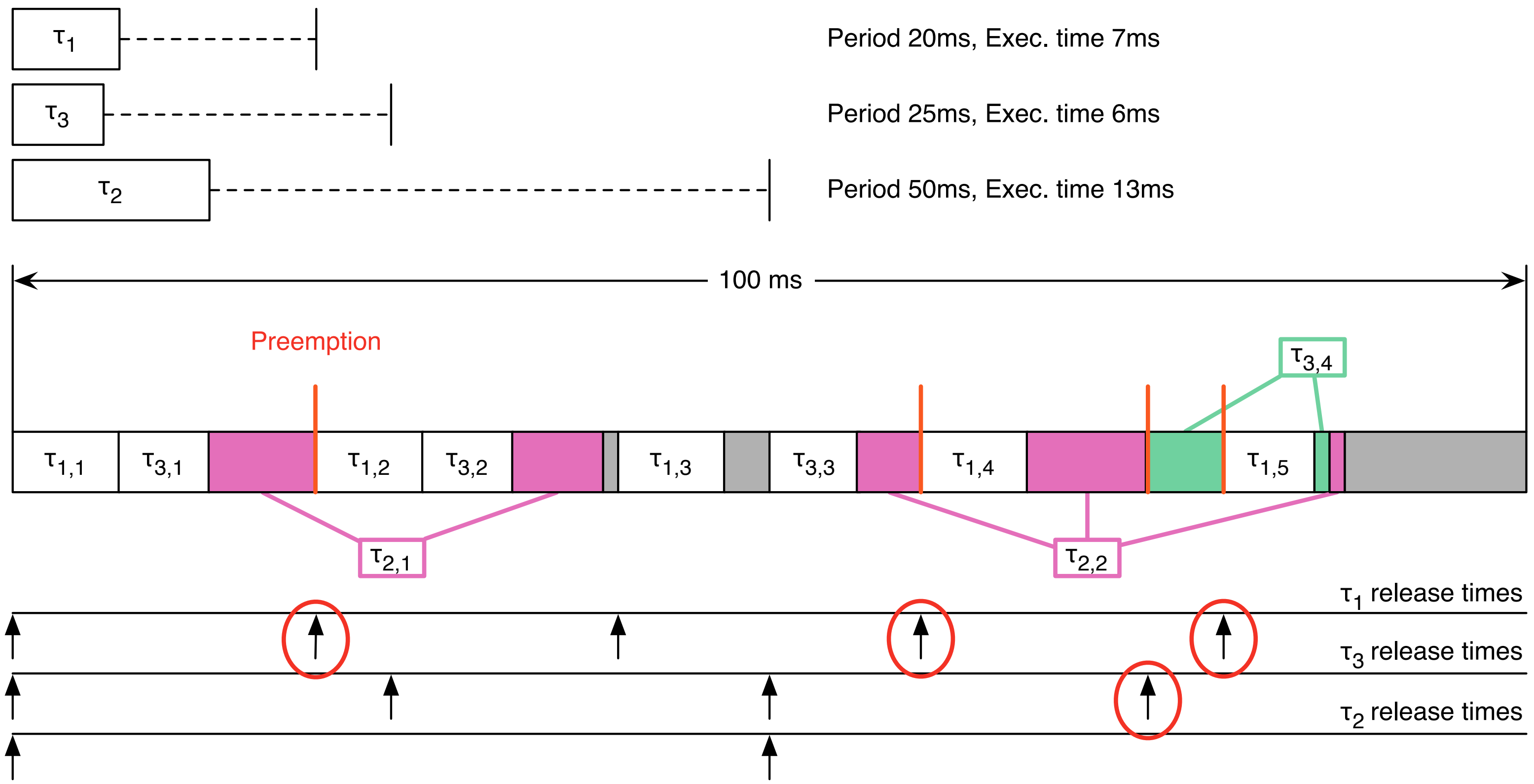
τ_2 is preempted again to run τ_3. The latter runs for 5 ms but at, $t = 80$, τ_1 is released for the fifth time.

EXAMPLE



τ_3 is preempted, too, to run τ_1. After the completion of τ_1, both τ_3 and τ_2 are *ready*. τ_3 runs for 1 ms, then completes.

EXAMPLE



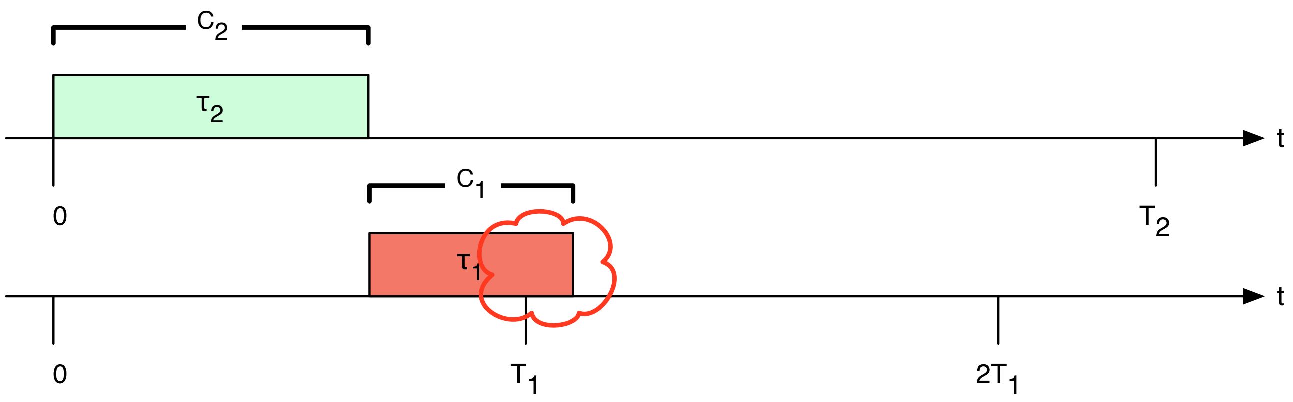
Finally, τ_2 runs and completes its execution cycle by consuming 1 ms of CPU time. After that, the system stays idle until $t = 100$, where the whole cycle starts again.

FEASIBILITY OF RM

- The relative deadline of a task is equal to its period: $D_i = T_i \ \forall$ i
- The absolute deadline is the time of its next release: $d_{i,j} = r_{i,j+1}$
- There is an overflow at time t if t is the deadline of a job that misses the deadline
- A scheduling is feasible for a given set of task if they are scheduled so that no overflows ever occur

FEASIBILITY OF RM

- Let us consider two tasks, τ_1 and τ_2, with $T_1 < T_2$
- If their priorities are not assigned according to RM, then τ_2 will have a priority higher than τ_1
- At a critical instant ($r_1=r_2=0$), their situation is



- The scheduling is feasible iff: $C_1+C_2 < T_1$

FEASIBILITY OF RMS

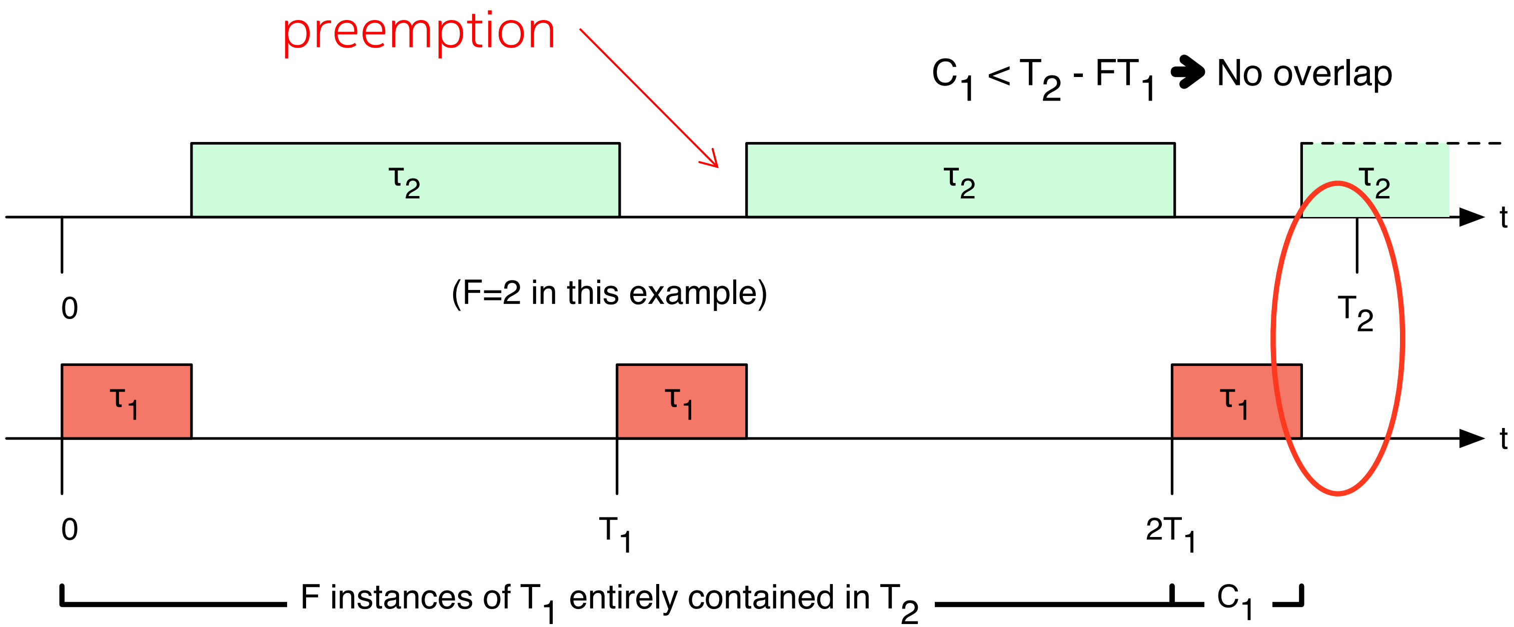
- If priorities are assigned according to RM, τ_1 will have a priority higher than τ_2
- Let F be the number of periods of τ_1 entirely contained in τ_2

$$F = \left\lfloor \frac{T_2}{T_1} \right\rfloor$$

- Two cases must be considered:
  - Execution time C_1 is "short enough" so that all the instances of τ_1 are completed before the next release of τ_2
  - Execution of the last instance of τ_1 overlaps the next release of τ_2

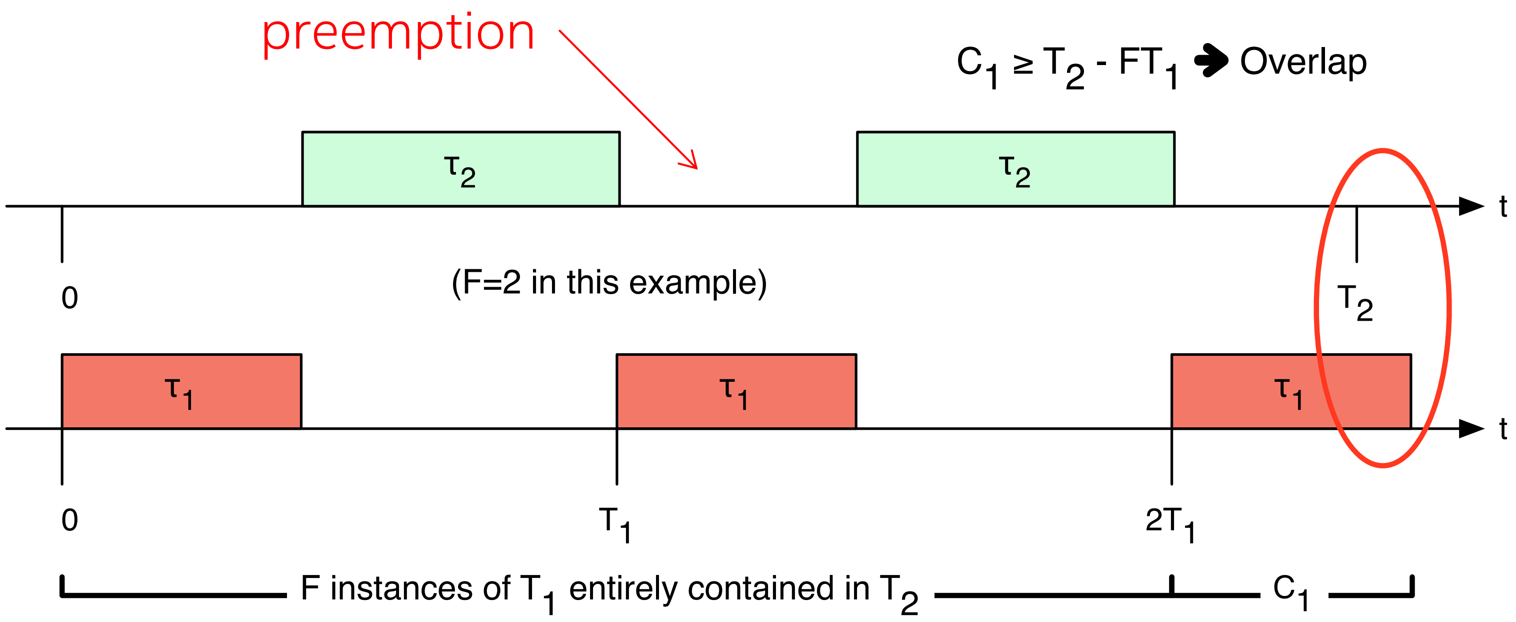
FEASIBILITY OF RM

- First case is feasible iff $(F+1)C_1 + C_2 \leq T_2$



FEASIBILITY OF RM

- Second case is feasible iff $FC_1 + C_2 \leq FT_1$



FEASIBILITY OF RM

- Given a set of two tasks τ_1 and τ_2 with $T_1 < T_2$
- If priorities are assigned according to RM, the scheduling is feasible iff:
  - $(F+1)C_1+C_2 \leq T_2$, when $C_1<T_2-FT_1$
  - $FC_1+C_2 \leq FT_1$, when $C_1 \geq T_2-FT_1$
- If priorities are assigned otherwise, the set is schedulable iff $C_1 + C_2 \leq T_1$

FEASIBILITY OF RM – SUFFICIENT CONDITION

- General criteria: let $\Gamma = \{\tau_1,\dots, \tau_n\}$ be a set of n periodic tasks, where each task τ_i is characterized by a processor utilization U_i

- Γ is schedulable with the RM if

$$\prod_{i=1}^{n}(U_i + 1) \leq 2$$

FEASIBILITY OF RM —NECESSARY CONDITION

```
Algorithm: DM_guarantee (Γ)
{       for (each τi∈Γ){
                I = 0;
                do {
                        R = I + Ci;
                        if (R > Di) return(UNSCHEDULABLE);
                        I = ∑j=1,…,(i-1)⌈R/Tj⌉ Cj;
                } while (I + Ci > R);
        }
        return(SCHEDULABLE);
}
```

Necessary & Sufficient

Assumption: Tasks are ordered according to their priorities:

$$m < n \Leftrightarrow D_m < D_n$$

EARLIEST DEADLINE FIRST SCHEDULING

- Dynamic priority scheduler
  - The ready tasks are executed in the order determined by their priority, which is computed at run-time
  - The priority assignment is dynamic, the same task may have different priorities at different time

- Hypothesis
  - Basic process model (deadline=period)
  - Tasks have **dynamic** priority
  - Scheduler is **preemptive**
  - One processor

- Scheduling algorithm
  - The EDF algorithm selects tasks according to their **absolute** deadlines. At each instant, the task with **earliest** deadline will receive **highest** priority

FEASIBILITY OF EDF

- Schedulability of periodic task set handled by EDF can be verified through the processor utilization factor

- A set of periodic tasks is schedulable with EDF if and only if

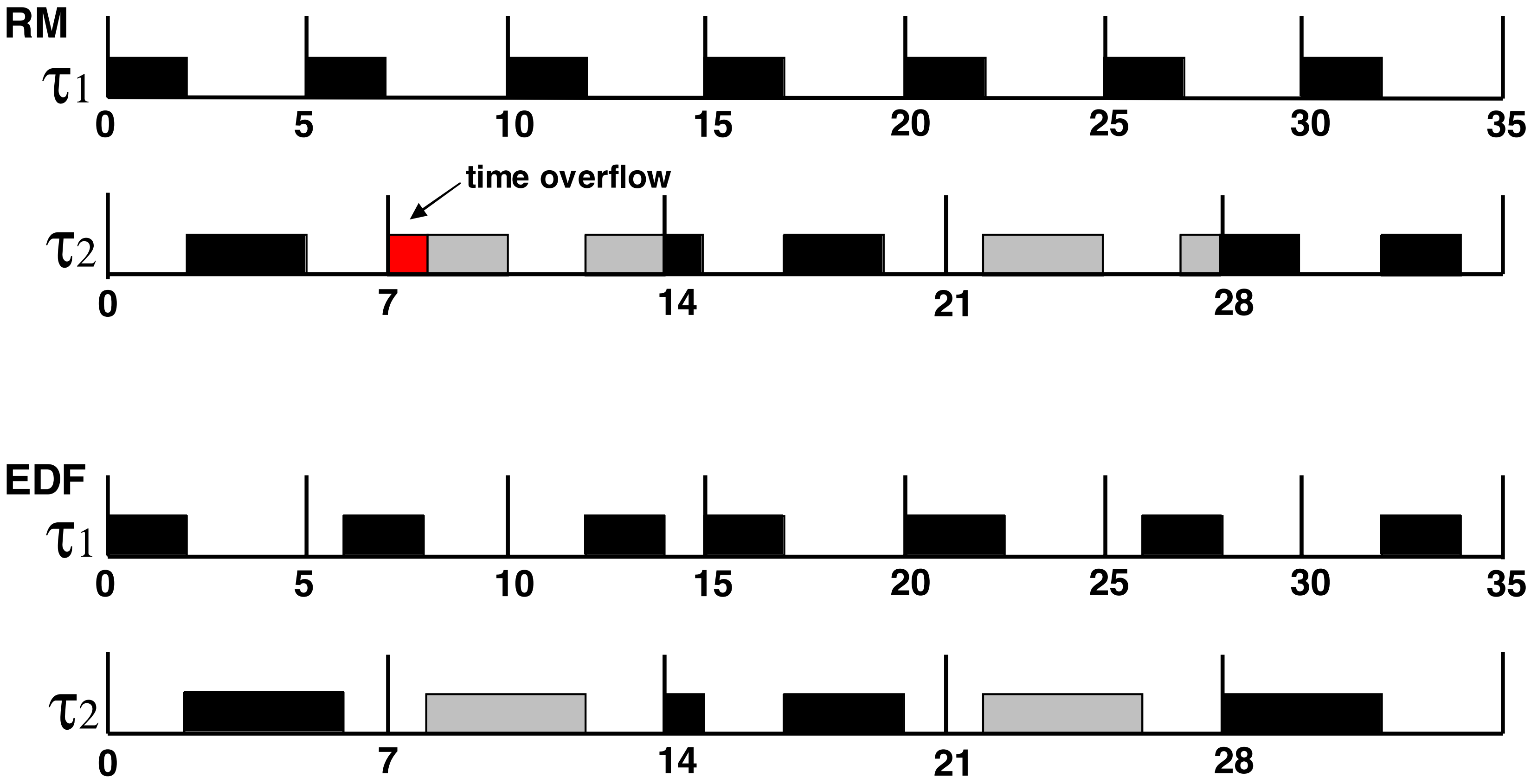
$$\sum_{i=1}^{n} \frac{C_i}{T_i} \leq 1$$

RM VS EDF

- RM is easier to implement than EDF, as priority is static
- EDF requires a more complex run-time system
- During overload situations, RM is easier to predict (lower-priority processes will miss deadlines first)
- EDF is less predictable, and can experience a domino effect in which a large number of tasks unnecessarily miss their deadline
- EDF is always able to exploit the full processor capacity, whereas RM in the worst case does not

RM VS EDF

- $C_1=2$, $T_1=5$, $C_2=4$, $T_2=7$

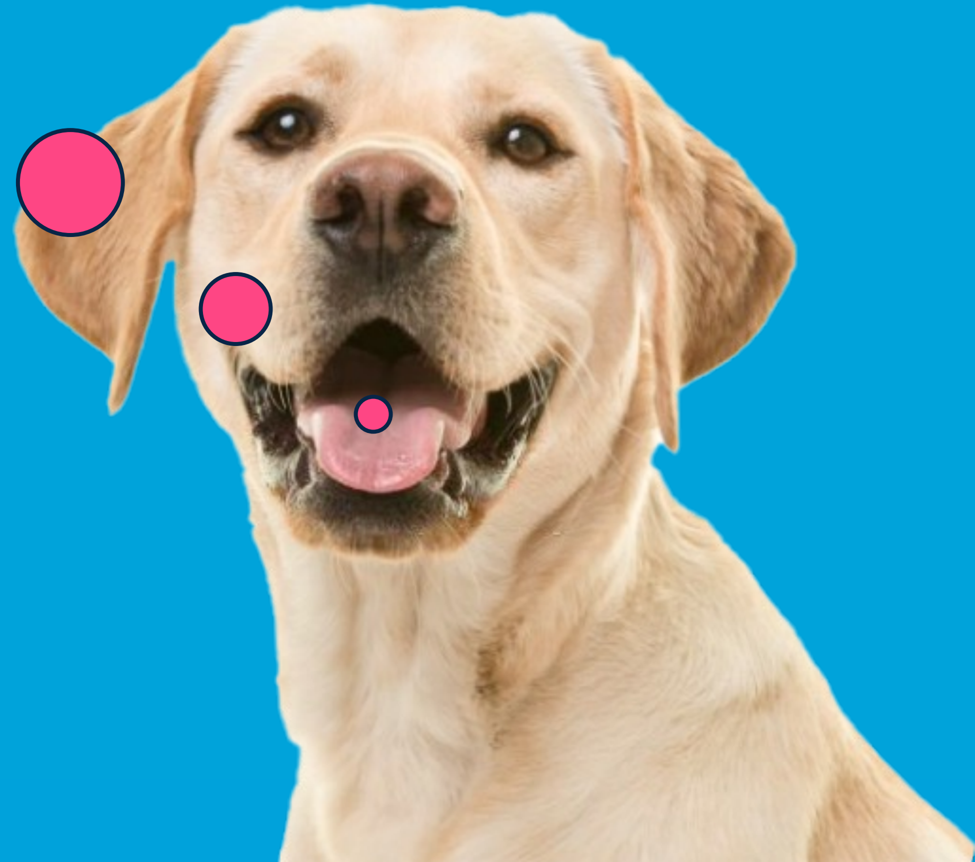


smilies
reSilient coMputer archItectures and LIfE Sciences

Politecnico di Torino
Department of Control and Computer Engineering

DAUIN

QUESTIONS?



THANK YOU!