Fixed-Priority Scheduling (FPS)

- This is the most widely used approach
- Each task has a fixed priority which is determined offline (pre-run-time)
- The tasks are executed in the order determined by their priority (subject to periods)
  - Schedule is actually only decided during execution (on-line)
  - Ready tasks are ordered by priority, the first to execute is the one with higher priority
- The model can be extended to several tasks having the same priority, however, it is usual to consider all tasks have different priorities

Fixed-Priority Scheduling (FPS)

- Advantages
  - Simple algorithm with low overhead - O(n) complexity
  - Easy to scale to many tasks
  - Easy to accommodate tasks with dissimilar periods (and sporadics)
  - Under overloads only low priority tasks may become unschedulable
- Disadvantages
  - High-priority tasks may starve low priority ones
  - More runtime complexity and overhead than cyclic executive

Fixed-Priority Scheduling (FPS)

- In real-time systems, the "priority" of a task is derived from its temporal requirements, not its importance to the correct functioning or integrity of the system
  - Two basic approaches exist (proven to be optimal) to assign priorities:
    - Rate Monotonic (RM): the smallest the period, the highest the priority
    - Deadline Monotonic (DM): the smallest the (relative) deadline, the highest the priority
    - When deadlines equal periods, both are the same
  - Priorities can also be assigned based on other criteria (e.g. importance or criticality)
    - No guarantees of optimality, may be inefficient

Fixed-Priority Scheduling (FPS)

- FPS Example

	C (ms)	T (ms)	D (ms)	Rate Monotonic Priority	Deadline Monotonic Priority
Control	20	60	40	High (3)	Med (2)
Alarm	5	70	20	Med (2)	High (3)
Logger	50	100	100	Low (1)	Low (1)

Deadline Monotonic is optimal if Deadlines are different from Periods

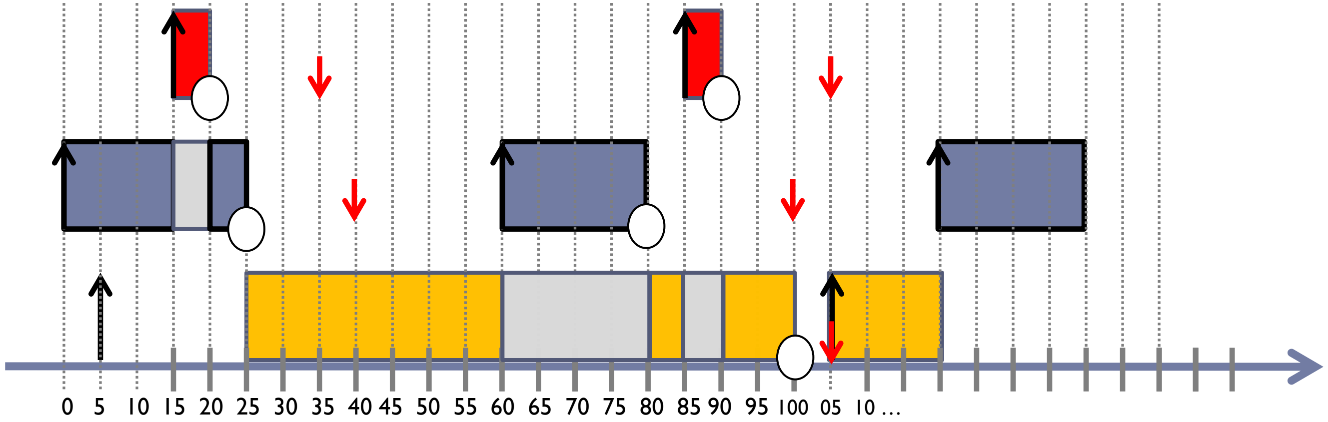
They are only different on assigning priorities – during execution the rule is always the same

Note: some OSes use priorities inversed – the lowest the highest priority

Fixed-Priority Scheduling (FPS)

- FPS Example

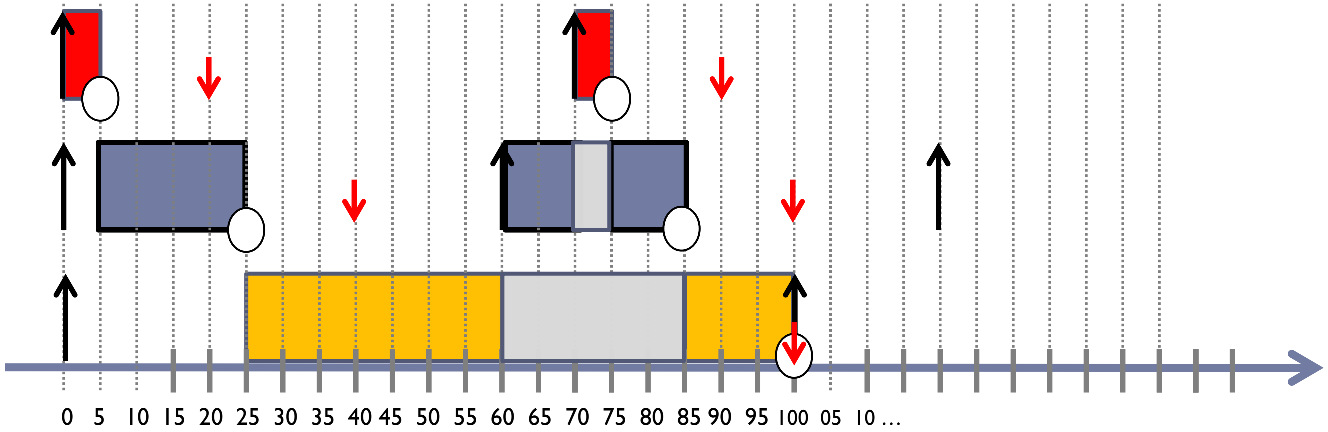
	C (ms)	T (ms)	D (ms)	Priority	Arrival instant
Control	20	60	40	2	0
Alarm	5	70	20	3	15
Logger	50	100	100	1	5



Fixed-Priority Scheduling (FPS)

- FPS Example

	C (ms)	T (ms)	D (ms)	Priority	Arrival instant
Control	20	60	40	2	0
Alarm	5	70	20	3	0
Logger	50	100	100	1	0

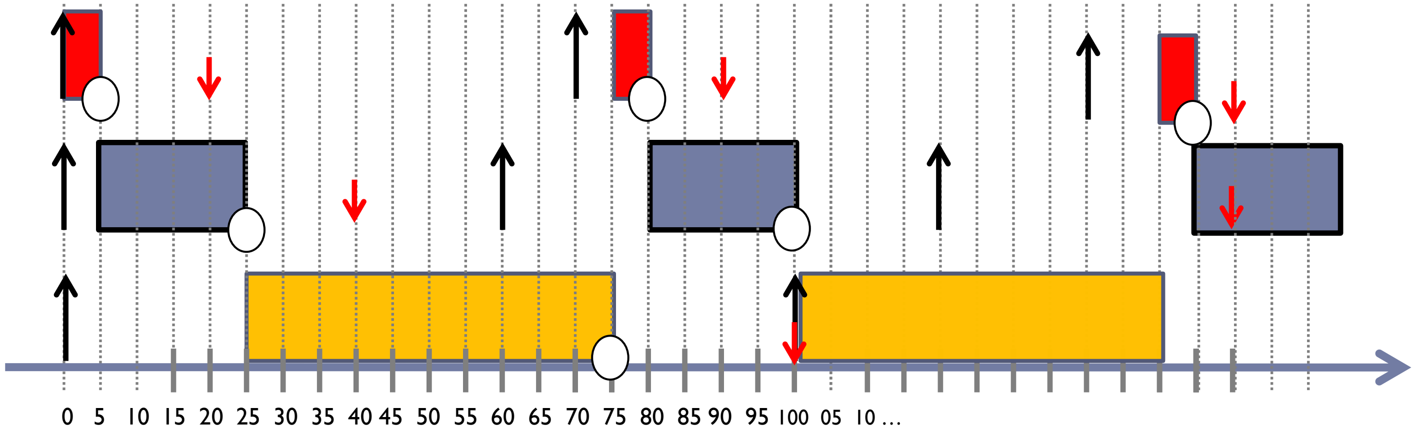


When there are no blocking factors (resource sharing or non-pre-emption), the worst-case behaviour is when all arrive at zero (critical instant)

Fixed-Priority Scheduling (FPS)

- FPS Example

	C (ms)	T (ms)	D (ms)	Priority	Arrival instant
Control	20	60	40	2	0
Alarm	5	70	20	3	0
Logger	50	100	100	1	0



If it was non-pre-emptive

Fixed-Priority Scheduling (FPS)

- Analysing schedulability
  - Whatever the scheduling algorithm used, it is necessary to verify if the system is schedulable, that is, all tasks will meet their deadlines
  - Even if decisions are made at runtime, the goal is that the guarantees can be provided off-line
  - Basically, $\forall i \Rightarrow \mathrm{R}i \leq \mathrm{D}i$
  - There are mainly two types of tests to determine schedulability
    - Based on the utilization of the processor
    - Based on calculating all response times

Fixed-Priority Scheduling (FPS)

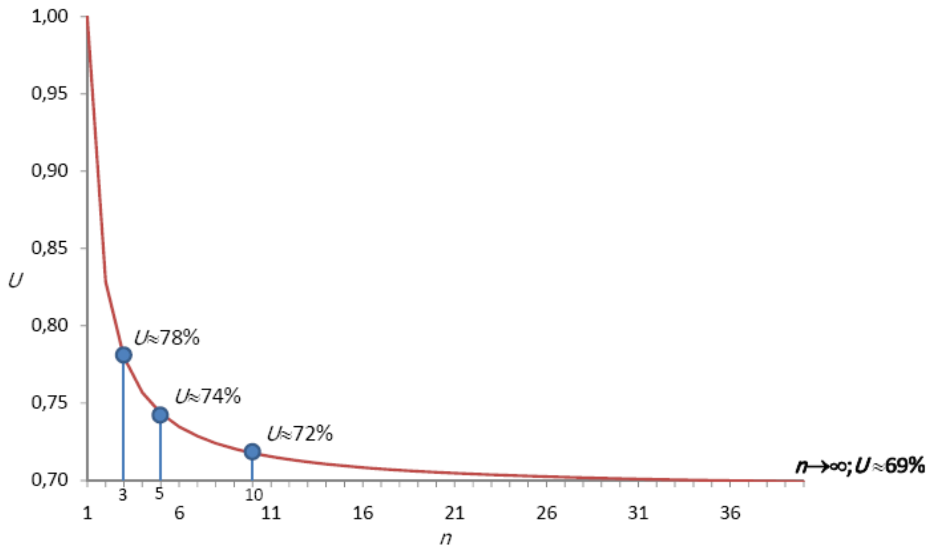
- Utilization based tests
  - In the simpler cases
    - Pre-emptive systems
    - When Deadlines equal Periods
  - The utilization of the system is the sum of each task utilization

$$U = \sum U_i = \sum \frac{C_i}{T_i}$$

    - If $U > 1 \Rightarrow$ The system is never schedulable, since CPU utilization is higher than 100%
  - We will mostly deal with the simpler utilization-based tests
    - Can become very pessimistic
    - There are more complex tests, with tighter results

Fixed-Priority Scheduling (FPS)

- Utilization based tests
  - For pre-emptive FPS (D=T)
    - If $U > 1 \Rightarrow$ The system is never schedulable, since CPU utilization is higher than 100%
    - If $U \leq n\left(2^{1/n} - 1\right) \Rightarrow$ The system is schedulable
    - Between these values no conclusion can be made
      - This is a necessary, but not sufficient test



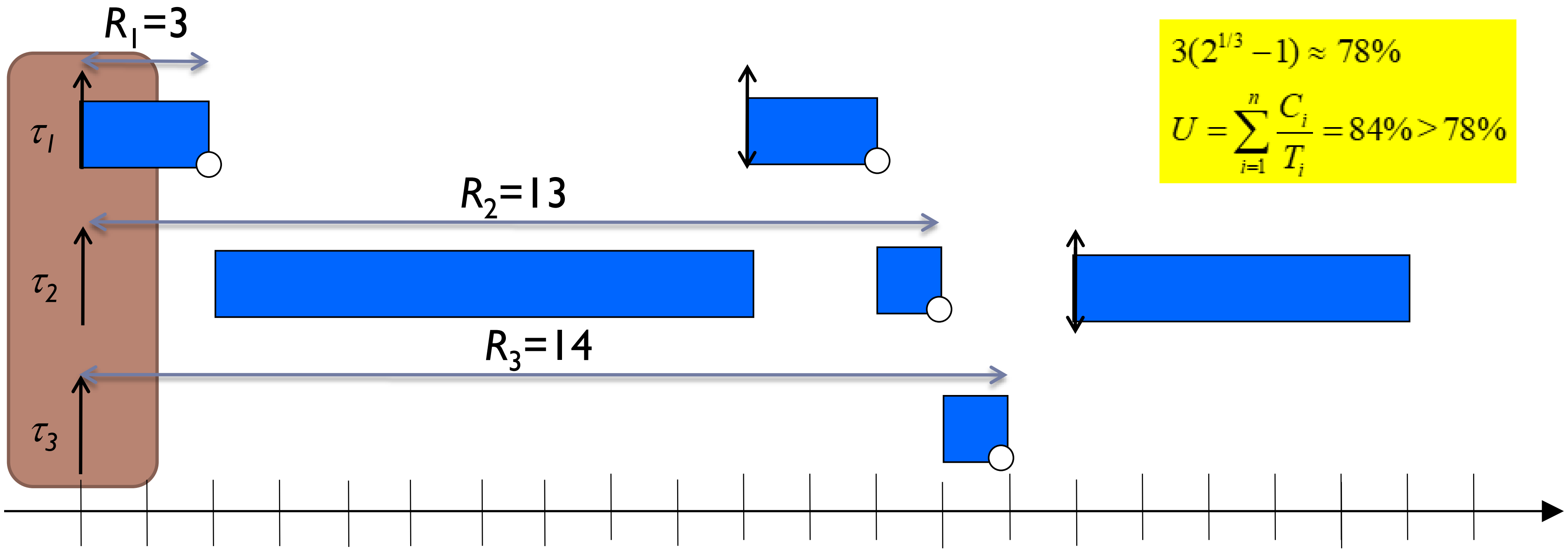
Fixed-Priority Scheduling (FPS)

- Utilization based tests
  - For pre-emptive FPS (D=T)

	C (ms)	T=D(ms)	U(Ci/Ti)
τ_1	2	10	0,20
τ_2	9	15	0,60
τ_3	1	25	0,04
		$U\ (\Sigma\ C_i\ /\ T_i\ )=$	84%

$$3(2^{1/3}-1) \approx 78\%$$

$$U = \sum_{i=1}^{n} \frac{C_i}{T_i} = 84\% > 78\%$$



Fixed-Priority Scheduling (FPS)

- Non-preemptive case
  - Problem with utilization-based tests: blocking time has to be taken into account for every task individually
    - Every task has to be tested individually
    - Tests become pessimistic and more complex
  - A contrained deadline taskset is schedulable if, for every task τ_i, the following holds

$$\left(\frac{C_i + B_i}{D_i} + 1\right) \prod_{\tau_j \in hp(\tau_i)} \left(U_j + 1\right) \leq 2$$

Fixed-Priority Scheduling (FPS)

- Exercise: check schedulability of taskset (for preemptive)

	C (ms)	T (ms)	D (ms)	
τ_1	5	50	50	
τ_2	2	10	10	
τ_3	3	7	7	

Fixed-Priority Scheduling (FPS)

- Exercise: what is the guaranteed minimum feasible period (for preemptive)

	C (ms)	T = D (ms)
τ_1	?	?
τ_2	2	10
τ_3	3	7

```
task body Task_t1 is
  Next_Time :  Time;
  Task_A_Period : Time_Span := …;
begin
  Next_Time := Clock;
  loop
    for i in 1..10 loop                          -- 10µs
      input := read_input ();                    -- 30µs
      if input = -1 then                         -- 30µs
         cont := cont + 1;                       -- 10µs
         send_msg (“Error”);                     -- 100µs
       end if;
       if i = 2 then                             -- 10µs
         send_msg (input);                       -- 100µs
       end if;
    end loop;

    Next_Time := Next_Time + Task_A_Period;      -- 10µs
    delay until Next_Time;                       -- considered negligible
  end loop;
end Task_t1;
```

Fixed-Priority Scheduling (FPS)

- Response-time analysis
  - Simple utilization based tests are only applicable to a simple model (pre-emptive, D=T)
    - These tests are also pessimistic (sufficient but not necessary)
      - There are systems that fail in the test but that nevertheless are schedulable
    - Extension to constrained deadline systems (D < T) is easy by using density instead of utilization

$$U^* = \sum \frac{C_i}{D_i}$$

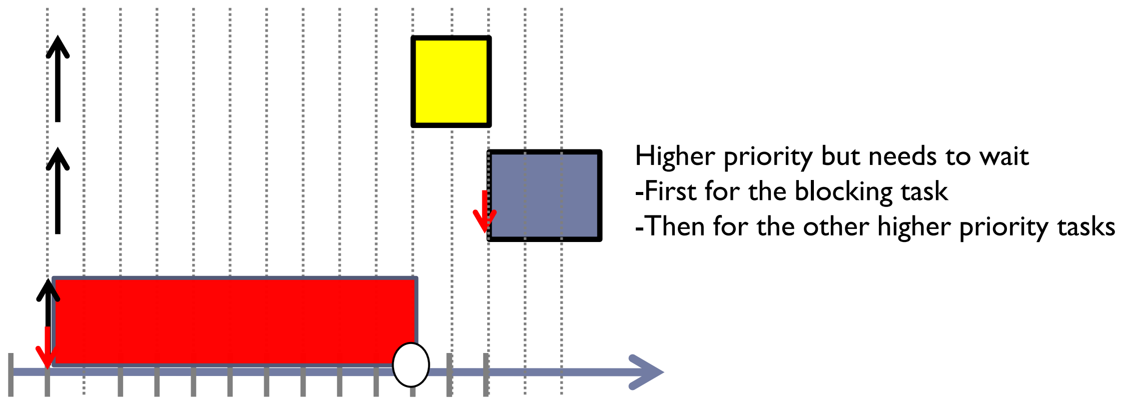
    - Other more complex utilization based tests exist, but are neither simple nor efficient

  - The optimal necessary and sufficient test is to derive all tasks' response times and compare to the deadline

$$\forall i \Rightarrow R_i \leq D_i$$

Fixed-Priority Scheduling (FPS)

- Response-time analysis
  - To derive the response times it is necessary to determine the critical instant
    - We have seen that in a non-blocking pre-emptive system this is all tasks released at t=0
    - In the other cases, the critical instant is different per task
      - E.g. for non-preemptive fixed priority it is the instant where just before the task (and all others of higher priority) being released, the maximum blocking task in the system gets the CPU

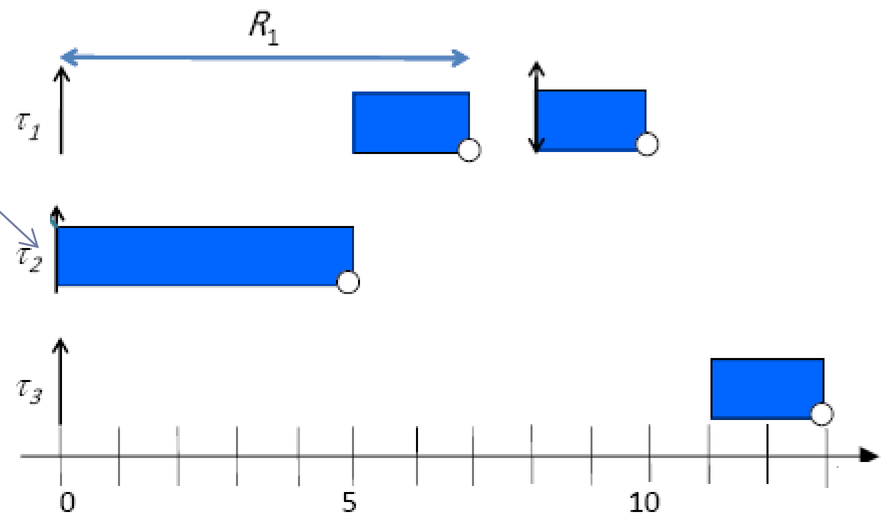


Fixed-Priority Scheduling (FPS)

- Response-time analysis
  - Example for non preemptive
    - For task t_1

	C(ms)	T=D(ms)
τ_1	2	8
τ_2	5	130
τ_3	2	140

t_2 just released

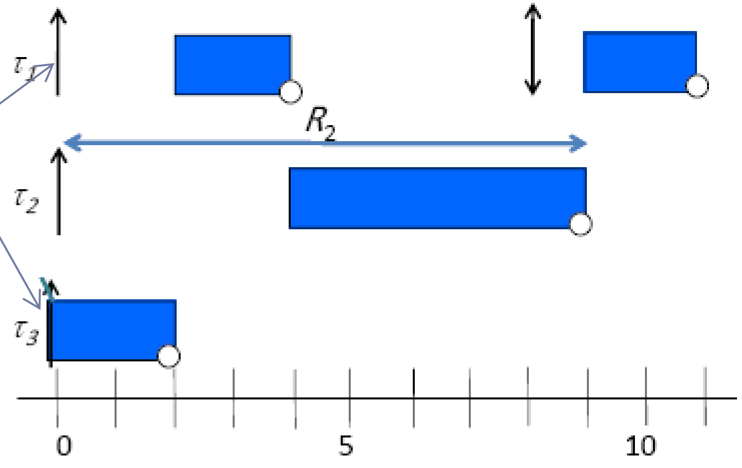


Fixed-Priority Scheduling (FPS)

- Response-time analysis
  - Example for non preemptive
    - For task *t2*

	C(ms)	T=D(ms)
τ_1	2	8
τ_2	5	130
τ_3	2	140

t_3 just released and t_l simultaneously

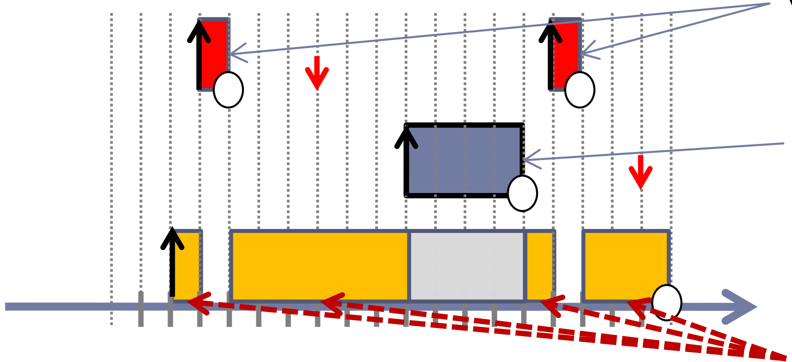


Fixed-Priority Scheduling (FPS)

- Response-time analysis
  - The time graph can be used to calculate response-times
    - However, it can be very complex and error-prone
  - There are methods to do that
    - For all scheduling methods
  - Basically, we have to iteratively determine how many higher priority tasks occur while the task we are calculating executes

Fixed-Priority Scheduling (FPS)

- Response-time analysis (preemptive)



- Basically, there are $\left\lceil \frac{R_3}{T_1} \right\rceil$ executions of t_1 and $\left\lceil \frac{R_3}{T_2} \right\rceil$ execution of t_2 during the execution of t_3

- Interference of t_1 is $\left\lceil \frac{R_3}{T_1} \right\rceil * C_1$ **+** Interference of t_2 is $\left\lceil \frac{R_3}{T_2} \right\rceil * C_2$ **+** C_3

$$R_3 = C_3 + \left\lceil \frac{R_3}{T_1} \right\rceil * C_1 + \left\lceil \frac{R_3}{T_2} \right\rceil * C_2$$

Fixed-Priority Scheduling (FPS)

- Response-time analysis (preemptive)

$$R_3 = C_3 + \left\lceil \frac{R_3}{T_1} \right\rceil * C_1 + \left\lceil \frac{R_3}{T_2} \right\rceil * C_2$$

$$\Downarrow$$

$$R_i = C_i + \sum_{j \in hp(i)} \left\lceil \frac{R_i}{T_j} \right\rceil * C_j$$

The same term (R_i) is on both sides of the equation

We have to solve it with recurrence (fixed-point iterative technique)

hp(i) is the set of tasks with higher priority than task i

Fixed-Priority Scheduling (FPS)

- Response-time analysis (preemptive)

The first iteration is response time equals execution time

$$R_i^0 = C_i$$

Then the equation is iteratively applied

$$R_i^{n+1} = C_i + \sum_{j \in hp(i)} \left\lceil \frac{R_i^n}{T_j} \right\rceil * C_j$$

It stops when it converges

$$R_i^{n+1} = R_i^n$$

or when response times are greater than deadlines

$$R_i^{n+1} > D_i$$

Fixed-Priority Scheduling (FPS)

- Response-time analysis (preemptive)
  - Example

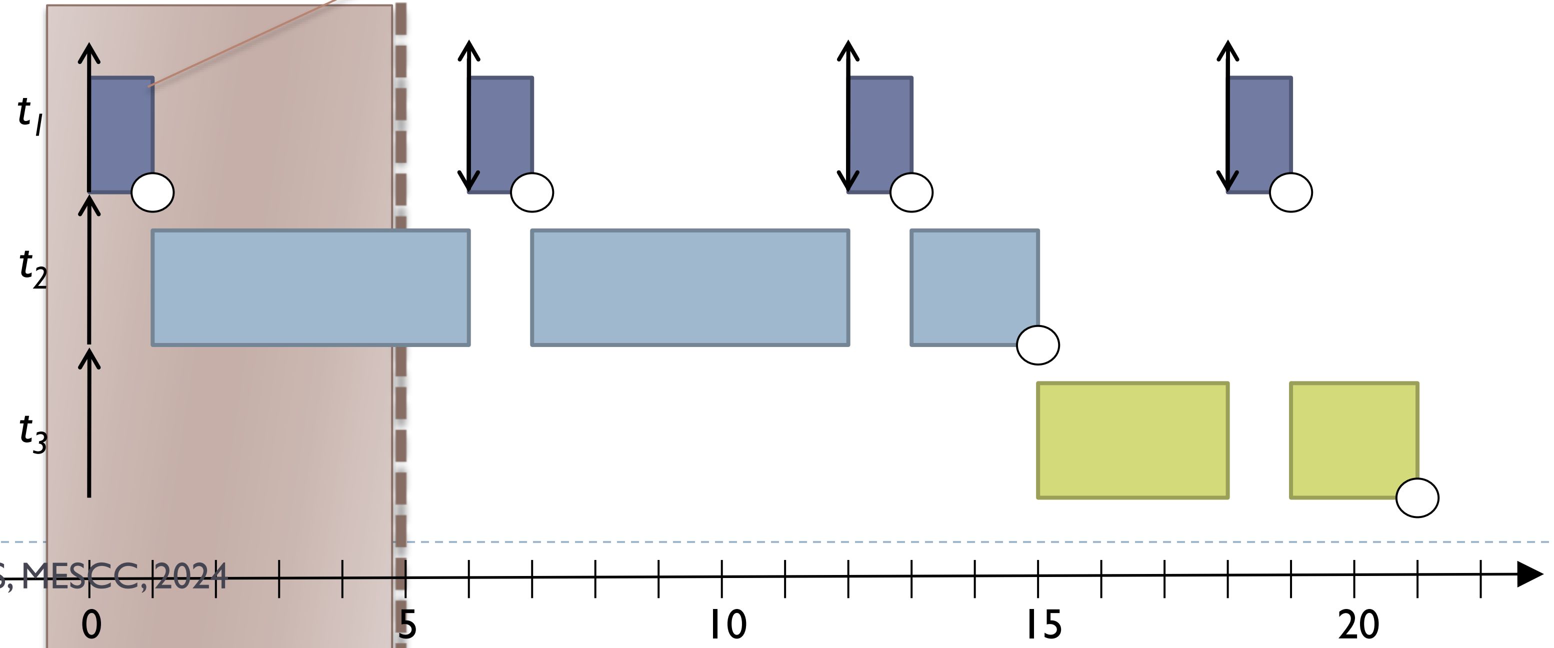
	C (ms)	T=D (ms)
τ_1	1	6
τ_2	12	130
τ_3	5	140

$$R_3^0 = C_3 = 5$$

$$R_3^1 = C_3 + \left\lceil \frac{5}{6} \right\rceil \times C_1$$

Time interval to analyse

Add one job of t_1



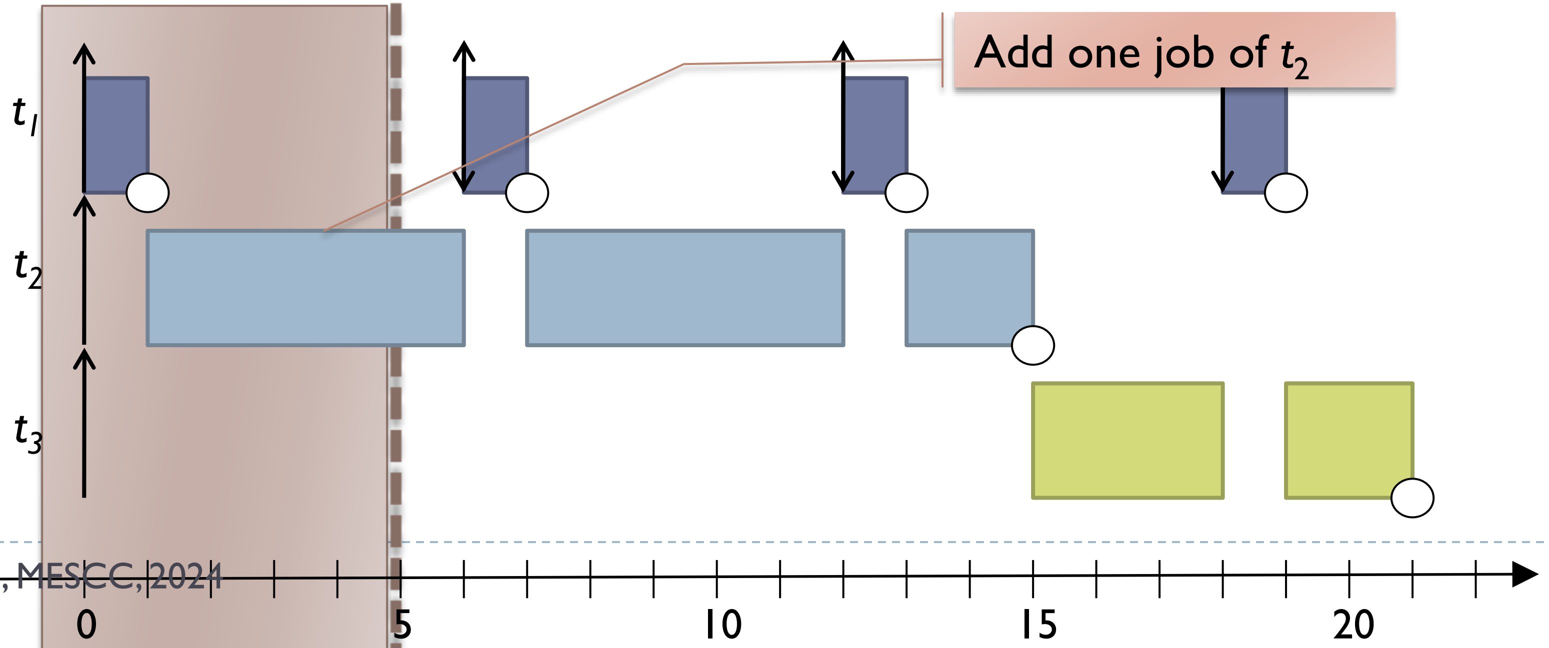
Fixed-Priority Scheduling (FPS)

- Response-time analysis (preemptive)
  - Example

	C (ms)	T=D (ms)
τ_1	1	6
τ_2	12	130
τ_3	5	140

$$R_3^0 = C_3 = 5$$

$$R_3^1 = C_3 + \left\lceil \frac{5}{6} \right\rceil \times C_1 + \left\lceil \frac{5}{130} \right\rceil \times C_2 = 5 + 1 \times 1 + 1 \times 12 = 18$$

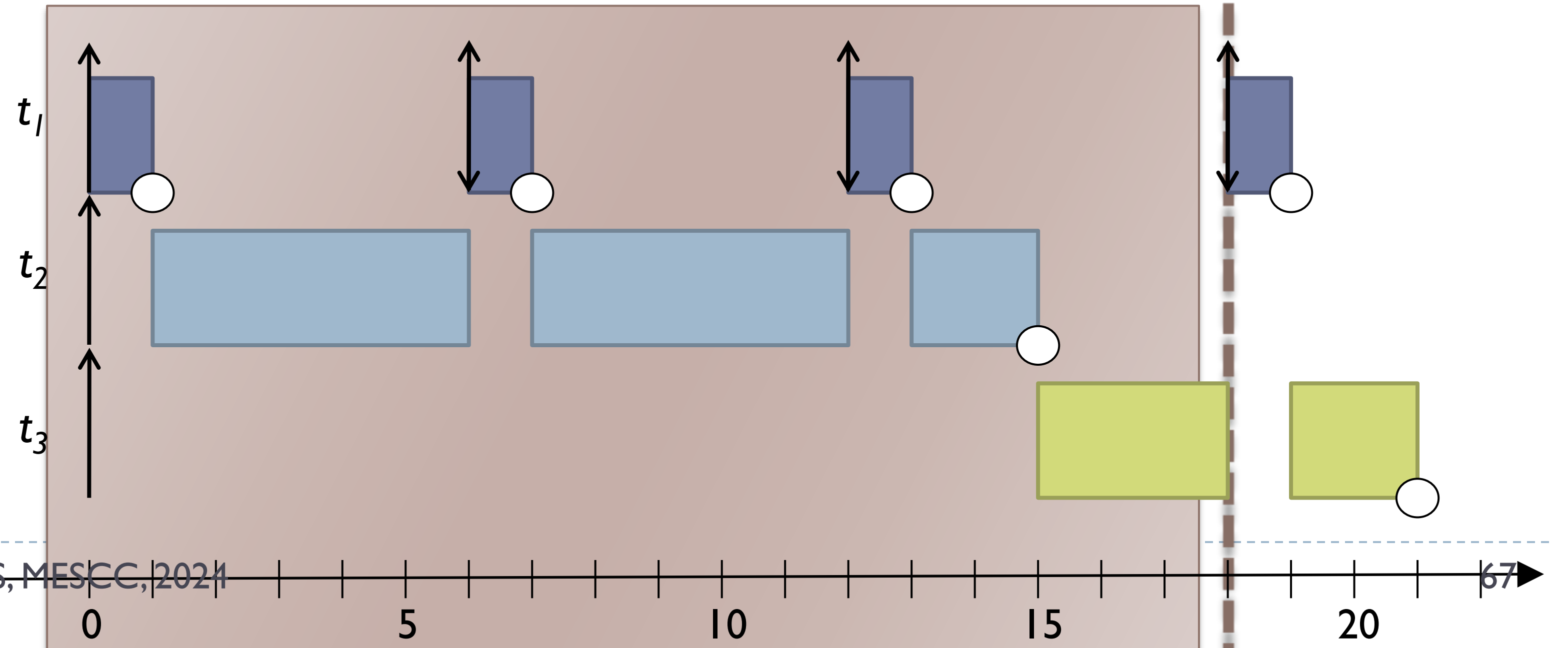


Fixed-Priority Scheduling (FPS)

- Response-time analysis (preemptive)
  - Example

$$R_3^1 = 18$$

	C (ms)	T=D (ms)
τ_1	1	6
τ_2	12	130
τ_3	5	140

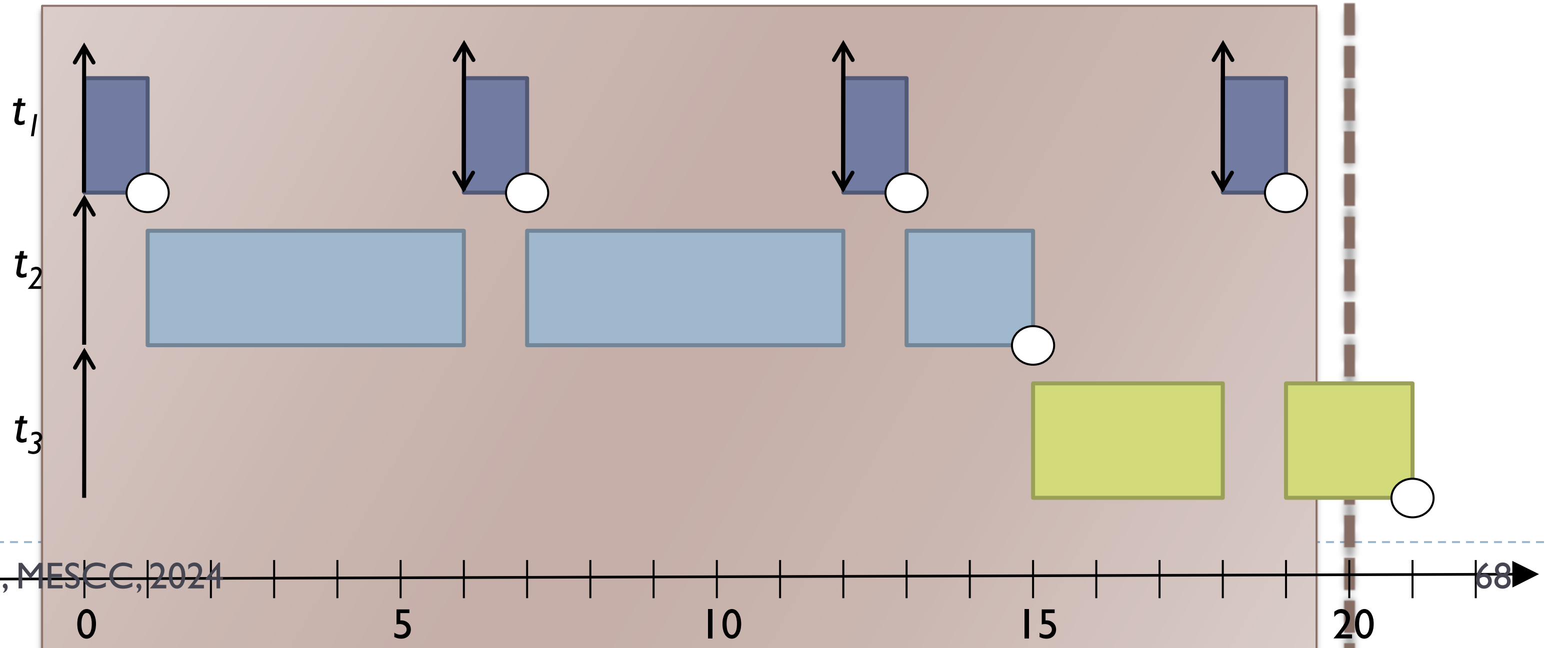


Fixed-Priority Scheduling (FPS)

- Response-time analysis (preemptive)
  - Example

	C (ms)	T=D (ms)
τ_1	1	6
τ_2	12	130
τ_3	5	140

$$R_3^2 = 20$$

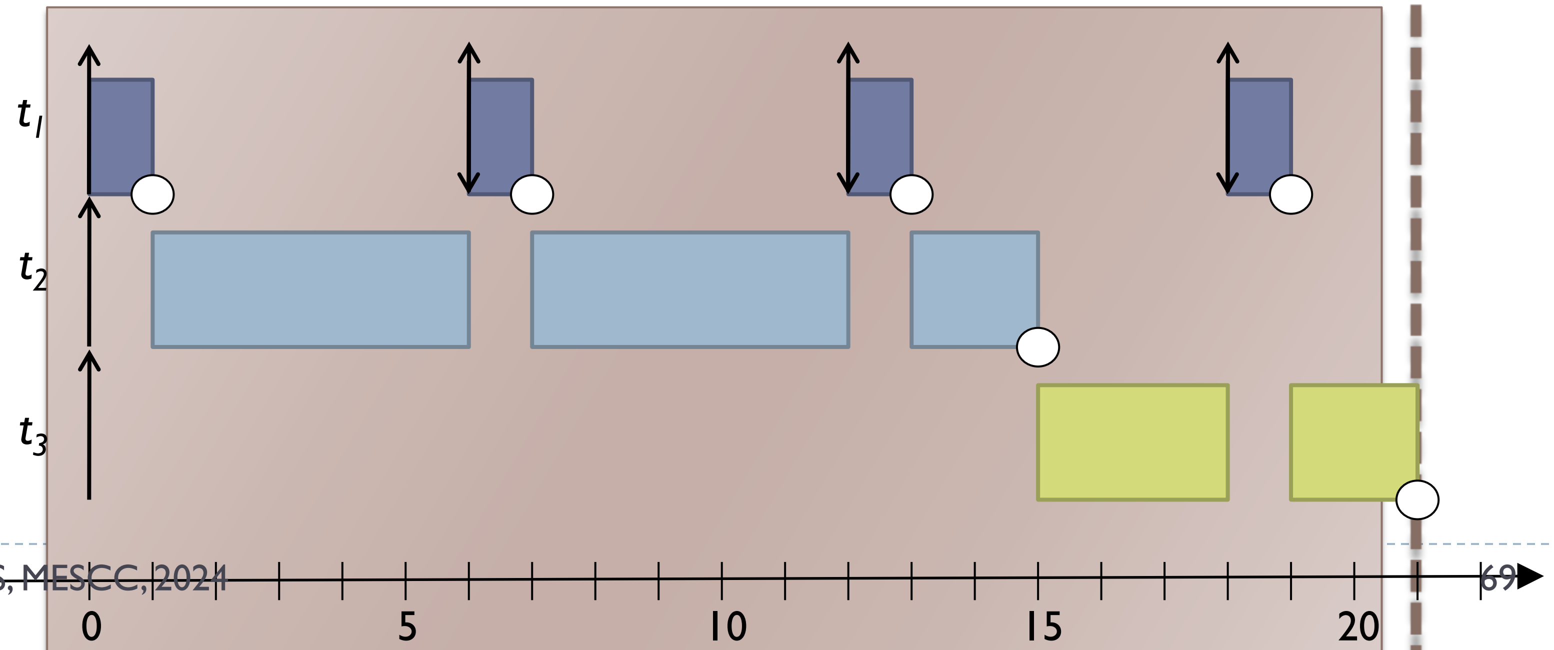


Fixed-Priority Scheduling (FPS)

- Response-time analysis (preemptive)
  - Example

$$R_3^3 = 21$$

	C (ms)	T=D (ms)
τ_1	1	6
τ_2	12	130
τ_3	5	140



Fixed-Priority Scheduling (FPS)

- Response-time analysis (preemptive)
  - Example

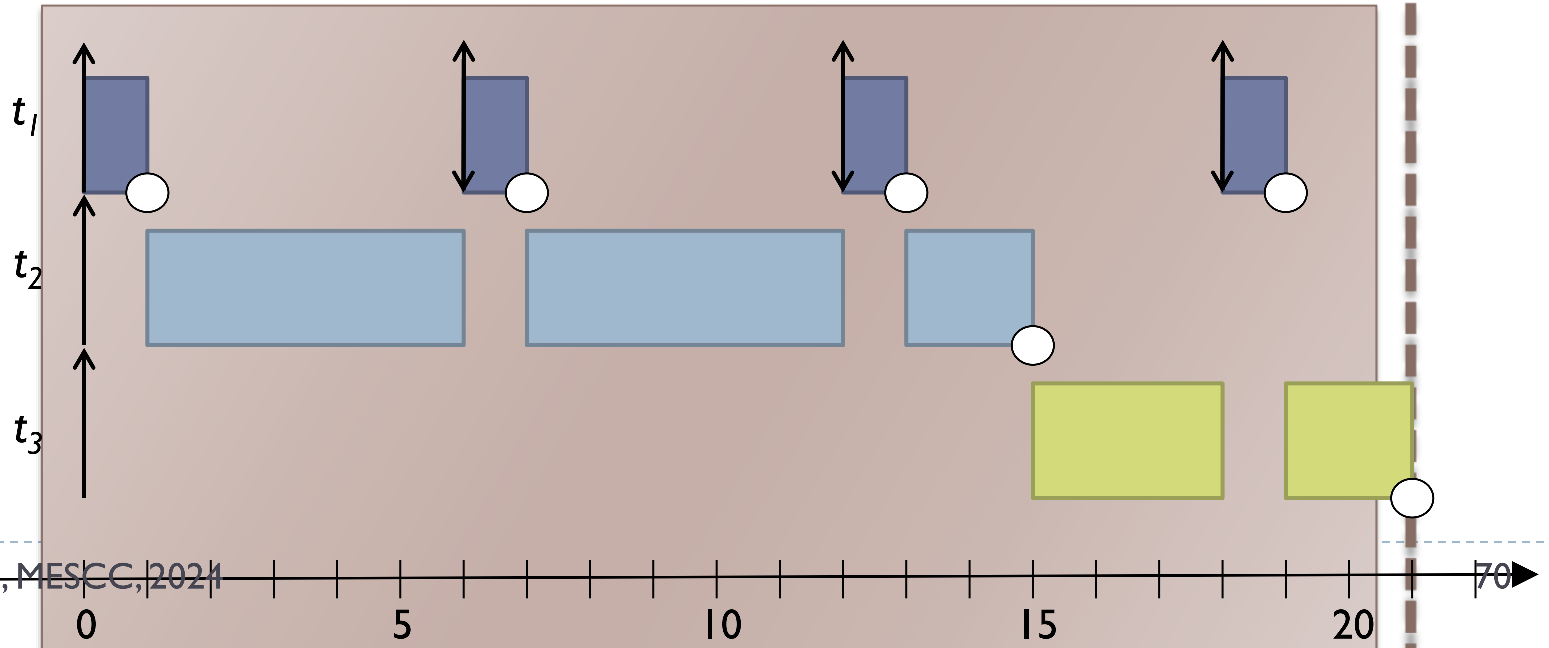
Converged:

R_3 = 21 ms

	C (ms)	T=D (ms)
τ_1	1	6
τ_2	12	130
τ_3	5	140

$$R_3^3 = 21$$

$$R_3^4 = C_3 + \left\lceil \frac{21}{6} \right\rceil \times C_1 + \left\lceil \frac{21}{130} \right\rceil \times C_2 = 5 + 4 \times 1 + 1 \times 12 = 21$$



Fixed-Priority Scheduling (FPS)

- Response-time analysis (preemptive)
  - Example

$$R_3^0 = C_3 = 5$$

$$R_3^1 = C_3 + \left\lceil \frac{5}{6} \right\rceil \times C_1 + \left\lceil \frac{5}{130} \right\rceil \times C_2 = 5 + 1 \times 1 + 1 \times 12 = 18$$

$$R_3^2 = C_3 + \left\lceil \frac{18}{6} \right\rceil \times C_1 + \left\lceil \frac{18}{130} \right\rceil \times C_2 = 5 + 3 \times 1 + 1 \times 12 = 20$$

$$R_3^3 = C_3 + \left\lceil \frac{20}{6} \right\rceil \times C_1 + \left\lceil \frac{20}{130} \right\rceil \times C_2 = 5 + 4 \times 1 + 1 \times 12 = 21$$

$$R_3^4 = C_3 + \left\lceil \frac{21}{6} \right\rceil \times C_1 + \left\lceil \frac{21}{130} \right\rceil \times C_2 = 5 + 4 \times 1 + 1 \times 12 = 21$$

Fixed-Priority Scheduling (FPS)

- Response-time analysis for non-preemptive
  - In this case, higher-priority tasks only interfere if they arrive **before** a task t_i starts
    - Iteration in the equation is only with Interference
  - However we need to add blocking (B_i) of a lower priority task
    - A task t_i will, in the worst-case, need to wait for the "biggest" of the lower-priority tasks:
  - When the task starts, executes until the end without preemptions

  - Response-time:

$$B_i = \max_{j \in lp(i)}(C_j) \Rightarrow \begin{aligned} I_i^0 &= B_i \\ I_i^{n+1} &= B_i + \sum_{j \in hp(i)} \left\lceil \frac{I_i^n}{T_j} \right\rceil * C_j \end{aligned} \Rightarrow R_i = I_i + C_i$$

Fixed-Priority Scheduling (FPS)

- Exercise: check schedulability of taskset (preemptive/non-preemptive)

	C (ms)	T (ms)	D (ms)	Rate Monotonic Priority	Deadline Monotonic Priority
Control	20	60	40	3	2
Alarm	5	70	20	2	3
Logger	50	100	100	1	1

Ceiling vs floor

- What happens when we have simultaneous events
  - The celling function of zero is zero
  - However, if two tasks appear at the same time, they cannot execute both
  - After the celling function being used for many years, it was demonstrated that it was wrong and in fact it should be floor plus one:

$$R_i = C_i + \sum_{j \in hp(i)} \left(\left\lfloor \frac{R_i}{T_i} \right\rfloor + 1 \right) * C_j$$

  - This applies to all schedulability analysis equations, although the community still uses mostly the ceiling function

Fixed-Priority Scheduling (FPS)

- Exercise
  - Consider a taskset scheduled with Rate Monotonic (RM)
  - Schedule and context switch overhead = 0.1 ms
  - Determine graphically the response time for the tasks, under preemptive and non-preemptive

	C (ms)	T=D (ms)
τ_1	25	70
τ_2	10	60
τ_3	20	80