Introduction to Embedded Systems

6. Real-Time Scheduling

Prof. Dr. Marco Zimmerling

NETWORKED EMBEDDED SYSTEMS LAB

UNI FREIBURG

Where we are …

Software

Hardware

1. Introduction to Embedded Systems
2. Software Development
3. Hardware-Software Interface
4. Programming Paradigms
5. Embedded Operating Systems
6. Real-time Scheduling
7. Shared Resources
8. Hardware Components
9. Power and Energy
10. Architecture Synthesis

Hardware-Software

Real-Time Scheduling of Aperiodic Tasks

Overview Aperiodic Task Scheduling

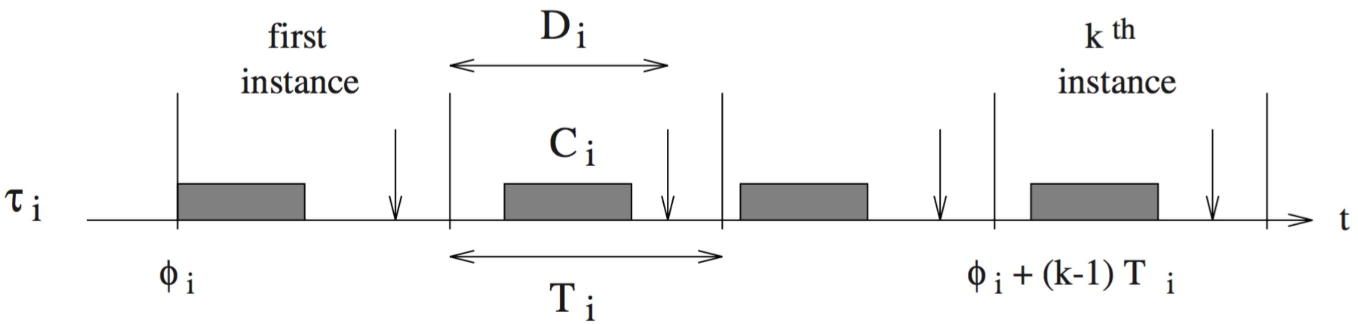
Scheduling of *aperiodic tasks* with real-time constraints:

- Table with some known algorithms:

		Equal arrival times non preemptive	Arbitrary arrival times preemptive
(*independent* = without precedence constraints)	**Independent tasks**	EDD (Jackson)	EDF (Horn)
(*dependent* = with precedence constraints)	**Dependent tasks**	LDF (Lawler)	EDF* (Chetto)

Real-Time Scheduling of Periodic Tasks

Periodic Tasks (1)



- Γ : set of periodic tasks
- τ_i : periodic task
- $\tau_{i,j}$: j-th instance of task τ_i
- $r_{i,j}, s_{i,j}, f_{i,j}, d_{i,j}$: release time, start time, finishing time, and absolute deadline of the j-th instance of task τ_i
- Φ_i : phase of task τ_i (release time of first instance)
- C_i : worst-case execution time of task τ_i
- T_i : period of task τ_i
- D_i : relative deadline of task τ_i
- The release times are given by $r_{i,j} = \Phi_i + (j-1)T_i$ and the absolute deadlines are given by $d_{i,j} = r_{i,j} + D_i = \Phi_i + (j-1)T_i + D_i$. If we have $D_i = T_i$ (*implicit deadline*), then the absolute deadlines are given by $d_{i,j} = \Phi_i + jT_i$.

Periodic Tasks (2)

- *Examples*: sensory data acquisition, low-level actuation, control loops, action planning, and system monitoring
- When an *application* features several concurrent periodic tasks with individual timing constraints, the OS has to guarantee that each periodic task τ_i is regularly activated at its proper *rate* $1/T_i$ and is completed within its deadline.

- Assumptions:
  - All periodic tasks are *independent*; that is, there are no precedence relations and no resource constraints.
  - *Tasks cannot suspend themselves*, for example, during I/O operations.
  - All *overheads* in the OS kernel are assumed to be *zero*.

Overview

Table of some known *preemptive scheduling algorithms for periodic tasks*:

	$D_i = T_i$	$D_i \leq T_i$
	Deadline equals period	Deadline smaller than period
static priority	RM (rate-monotonic)	DM (deadline-monotonic)
dynamic priority	EDF	EDF

Rate-Monotonic (RM) Scheduling

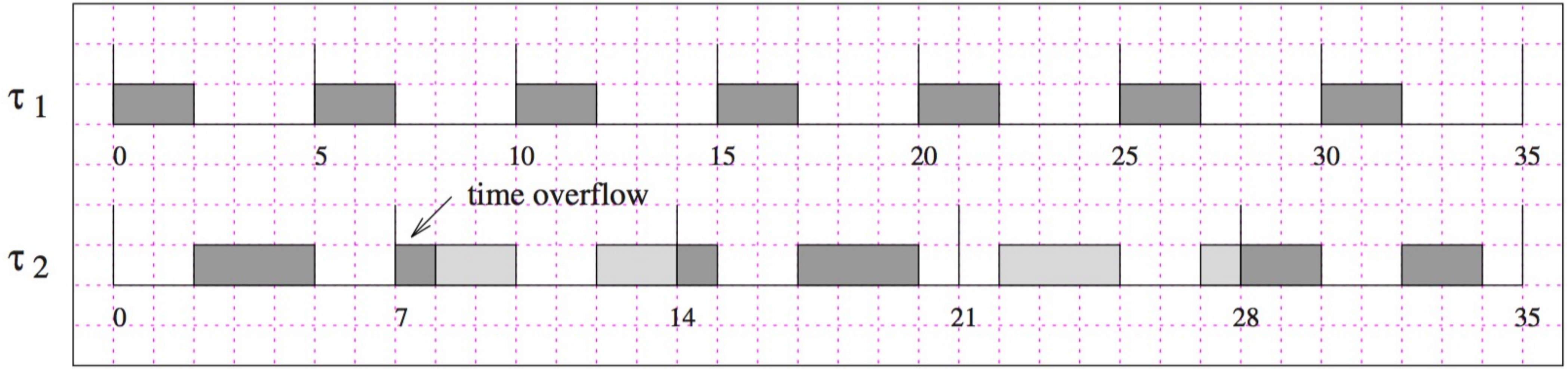
- Scheduling of periodic tasks τ_i with *implicit deadlines*
  - All tasks have relative deadlines equal to their periods (i.e., $C_i \leq D_i = T_i$)
  - Every task has a priority. The priorities are assigned to the tasks before execution and do not change over time. This is called a static or *fixed priority assignment*.
  - The currently executing task instance is *preempted* by an instance of a task with a higher priority. In the following, we will just say “task” instead of “task instance.”

- *RM scheduling rule*: Given a set of n independent periodic tasks with implicit deadlines, assign a fixed priority to each task such that tasks with higher request rates (i.e., with shorter periods) have higher priorities. RM is optimal among all fixed-priority assignments in the sense that no other fixed-priority algorithm can schedule a task set that cannot be scheduled by RM.

Example: Infeasible Schedule Produced by RM

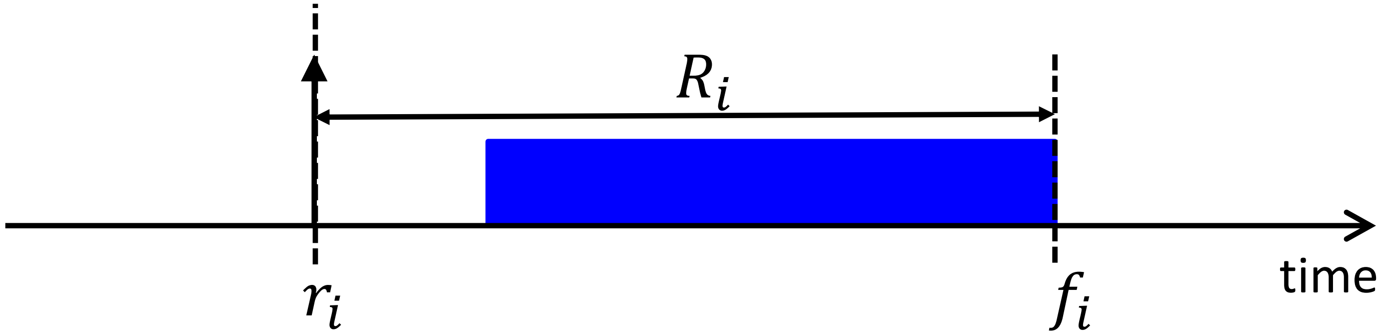
- Two periodic tasks τ_1 and τ_2 with
  - Phases $\Phi_1 = \Phi_2 = 0$
  - Periods $T_1 = 5$ and $T_2 = 7$
  - Worst-case execution times $C_1 = 2$ and $C_2 = 4$

- Schedule produced by RM, where *τ_1 has higher priority than τ_2* since $T_1 < T_2$:



Critical Instant

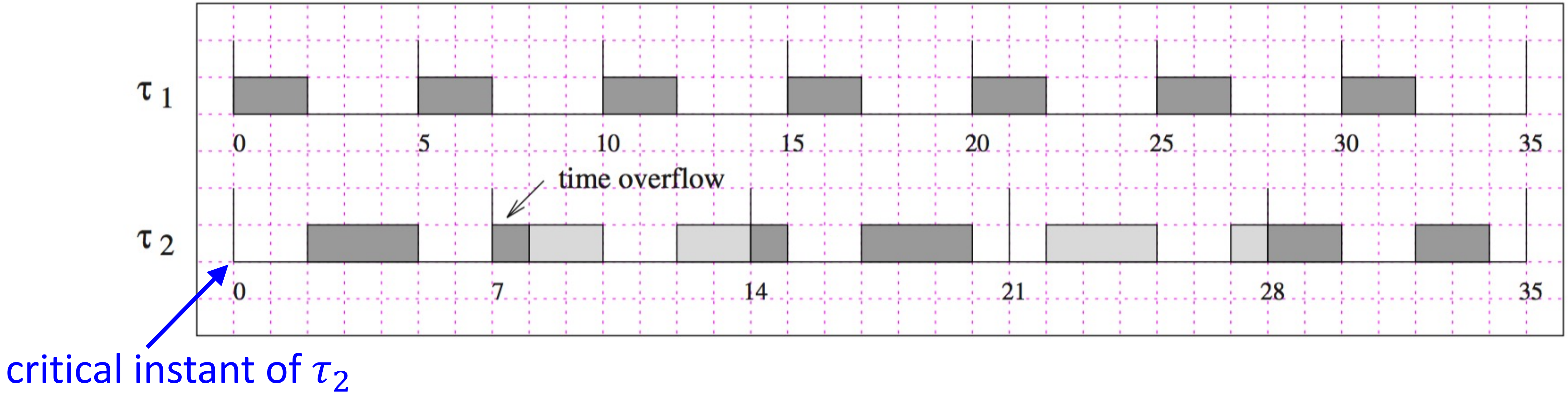
- RM is optimal among all fixed-priority assignments in the sense that no other fixed-priority algorithm can schedule a task set that cannot be scheduled by RM.
- To proof the optimality of RM, we need the concept of a *critical instant*.
- *Definition*: A critical instant of a task is the release time r_i that produces the largest response time R_i, that is, the largest difference between release time r_i and finishing time f_i.
- *Lemma*: For any task, a critical instant occurs whenever the task is released simultaneously with all higher-priority tasks.



Example: Infeasible Schedule Produced by RM

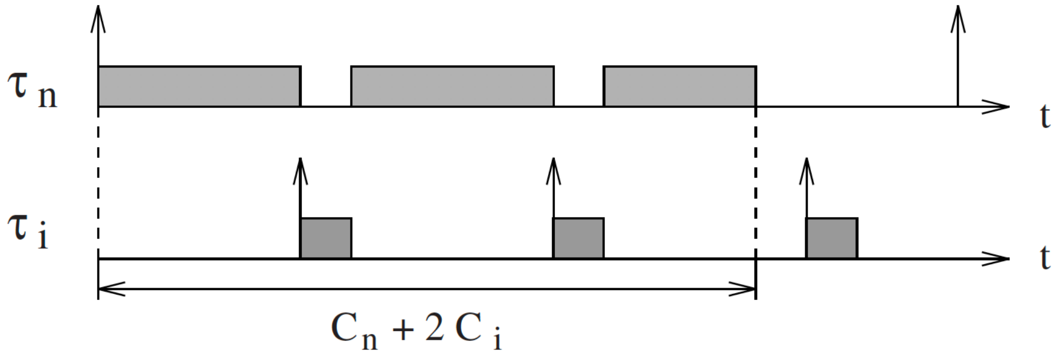
- Two periodic tasks τ_1 and τ_2 with
  - Phases $\Phi_1 = \Phi_2 = 0$
  - Periods $T_1 = 5$ and $T_2 = 7$
  - Worst-case execution times $C_1 = 2$ and $C_2 = 4$

- Schedule produced by RM, where τ_1 *has higher priority than* τ_2 since $T_1 < T_2$:

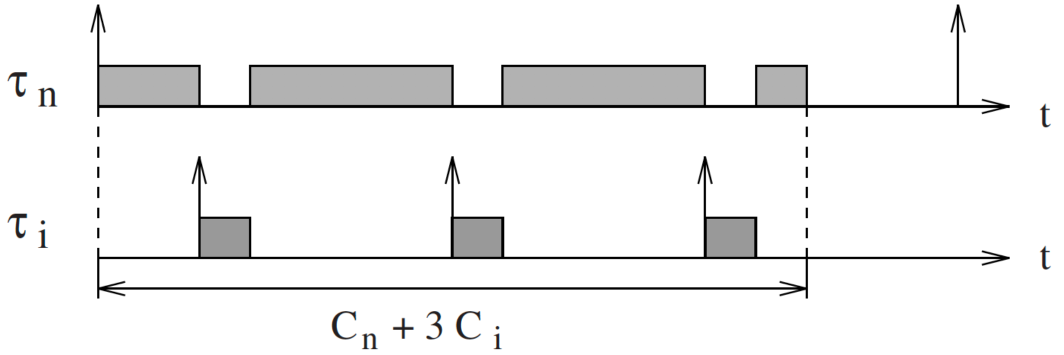


Critical Instant: Proof Sketch

- Let $\Gamma = \{\tau_1, \tau_2, \dots, \tau_n\}$ be a set of periodic tasks ordered by increasing periods. Thus, task τ_n has the longest period and, according to RM, the lowest priority.
- The response time of τ_n is delayed by interference of τ_i with higher priority.



- The response time of τ_n may increase if τ_i is released earlier.



- Thus, the response time is largest if tasks τ_n and τ_i are released simultaneously.
- Repeating the argument for all higher-priority tasks proves the lemma.

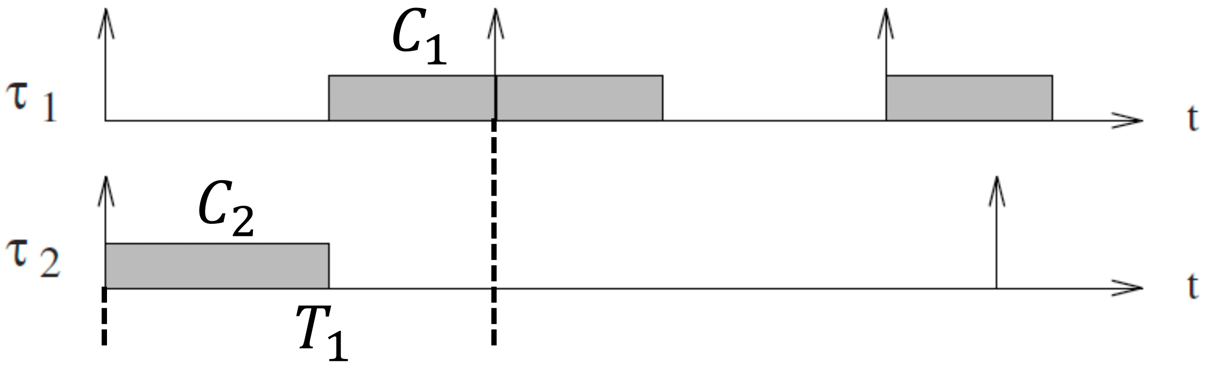
Optimality of RM: Proof Overview

- We just proved that, for any task, a critical instant occurs whenever the task is released simultaneously with all higher-priority tasks.
- This means that the schedulability of tasks can easily be checked at their critical instants: If all tasks are feasible at their critical instants (i.e., the largest response time does not exceed the deadline for every task), then the task set is schedulable in any other condition.

- Based on this result, the optimality of RM can be shown in two steps:
  1. Show that, given a set with two periodic tasks, if the task set is schedulable by an arbitrary priority assignment, then it is also schedulable by RM.
  2. Extend the result to a set of n periodic tasks.
- In the following, we will only prove the first step.

Optimality of RM: Proof of First Step (1)

- We have $\Gamma = \{\tau_1, \tau_2\}$ with $T_1 < T_2$.
- If priorities are *not* assigned according to RM, then τ_2 has the highest priority.

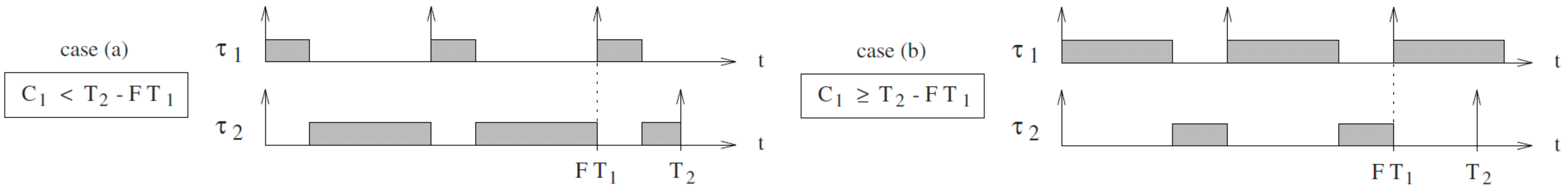


- Looking at the critical instant of τ_1, where it is simultaneously released with all higher-priority tasks (here this is only τ_2), we find that the task set is schedulable if

$$C_1 + C_2 \leq T_1$$

Optimality of RM: Proof of First Step (2)

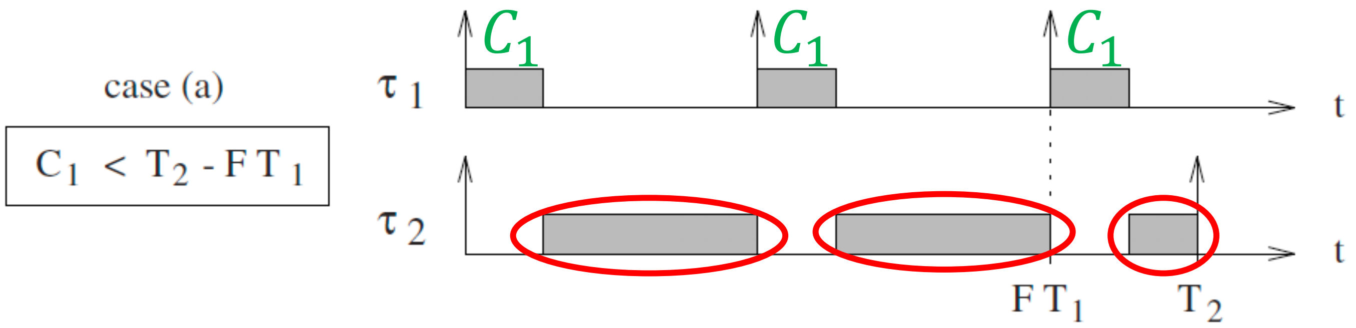
- If priorities *are* assigned according to RM, then τ_1 has the highest priority.
- Two cases must be considered to guarantee a feasible schedule, where $F = \lfloor T_2/T_1 \rfloor$ is the number of periods of τ_1 that are fully contained in T_2.
  - *Case (a)*: The computation time of τ_1, when synchronously activated with τ_2, is short enough so that all its requests are completed before the second request of τ_2.
  - *Case (b)*: The computation time of τ_1, when synchronously activated with τ_2, is long enough to overlap with the second request of τ_2



- We must show that in either case the condition for schedulability without RM (see previous slide) implies the condition for schedulability with RM (see next slides).

Optimality of RM: Proof of First Step (3)

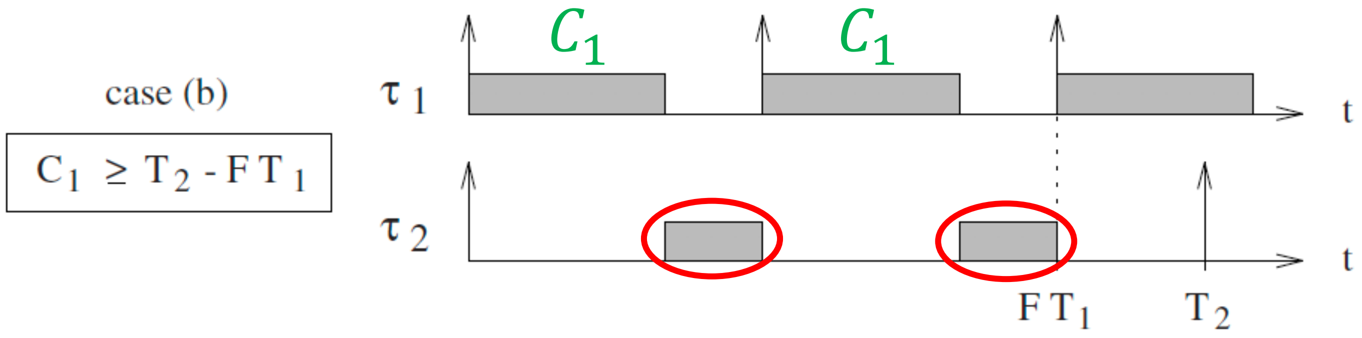
- In *case (a)* the task set is schedulable
  if $(F+1)C_1 + C_2 \leq T_2$



- Thus, we have to show that $C_1 + C_2 \leq T_1 \Longrightarrow (F+1)C_1 + C_2 \leq T_2$
- We begin with: $C_1 + C_2 \leq T_1$
  Multiply both sides with F: $FC_1 + FC_2 \leq FT_1$
  Since $F \geq 1$ we can write: $FC_1 + C_2 \leq FC_1 + FC_2 \leq FT_1$
  Add C_1 to each member: $(F+1)C_1 + C_2 \leq FT_1 + C_1$
  Since in case (a) $C_1 < T_2 - FT_1$: $(F+1)C_1 + C_2 \leq FT_1 + C_1 < T_2$

Optimality of RM: Proof of First Step (4)

- In *case (b)* the task set is schedulable
  if $FC_1 + C_2 \leq FT_1$



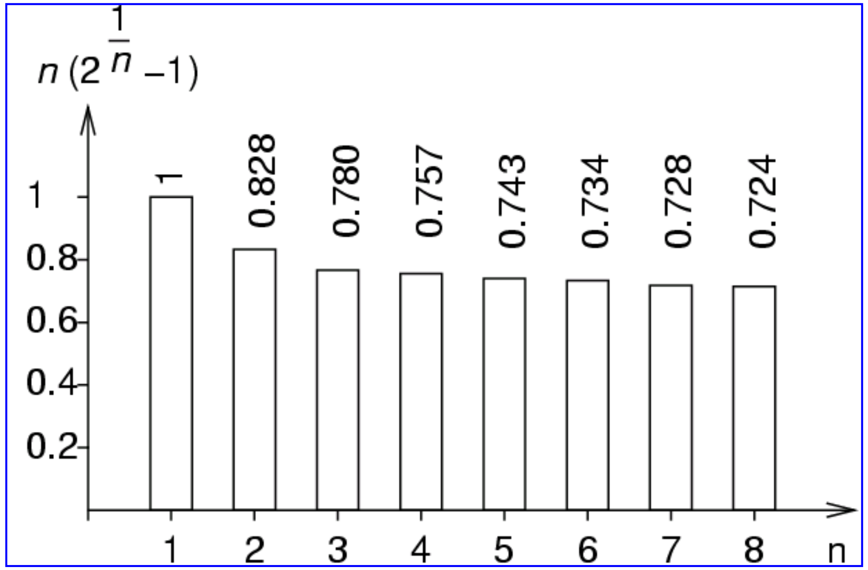
- Thus, we have to show that $C_1 + C_2 \leq T_1 \Longrightarrow FC_1 + C_2 \leq FT_1$
- We begin with: $C_1 + C_2 \leq T_1$
  Multiply both sides with F: $FC_1 + FC_2 \leq FT_1$
  Since $F \geq 1$ we can write: $FC_1 + C_2 \leq FC_1 + FC_2 \leq FT_1$

- *Summary*: Both cases combined, we have shown that, given two periodic tasks τ_1 and τ_2 with $T_1 < T_2$, if the schedule is feasible with an arbitrary priority assignment, then it is also feasible with RM. In other words, RM is optimal.

RM Schedulability Test

- A set of n periodic real-time tasks is schedulable using RM if

$$\sum_{i=1}^{n} C_i/T_i \leq n\left(2^{1/n} - 1\right)$$

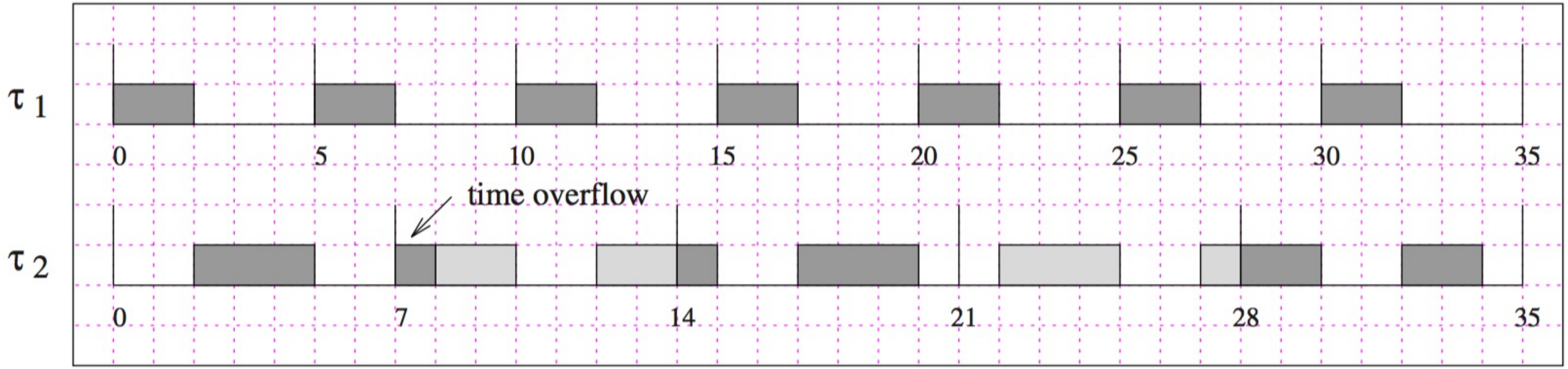


- This condition is *sufficient but not necessary*. That is, if the above condition holds for a given task set, then this task set is definitely schedulable using RM. However, if the above condition does not hold, then the task set may or may not be schedulable using RM.

- The term $\sum_{i=1}^{n} C_i/T_i$ is called the *processor utilization U* of a set of n periodic real-time tasks. It denotes the fraction of time the processor spends executing the task set (i.e., the "computational load" induced by the task set).

Example Revisited: Processor Utilization

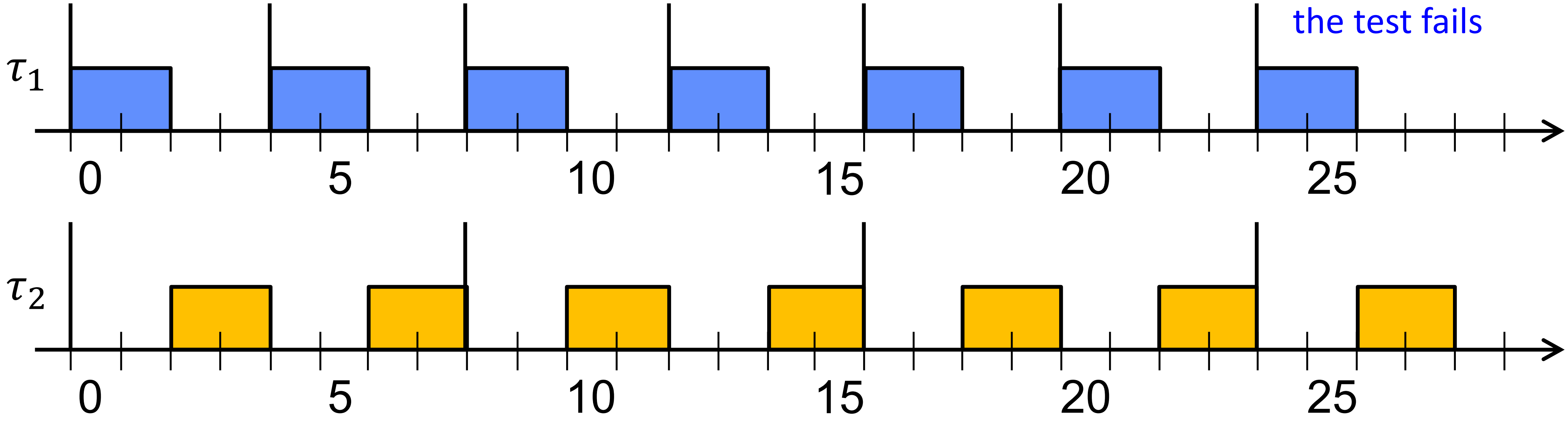
- Two periodic tasks τ_1 and τ_2 with
  - Phases $\Phi_1 = \Phi_2 = 0$
  - Periods $T_1 = 5$ and $T_2 = 7$
  - Worst-case execution times $C_1 = 2$ and $C_2 = 4$
- *Processor utilization U* $= \sum_{i=1}^{n} C_i/T_i = \frac{2}{5} + \frac{4}{7} \approx 0.97 > 2\left(2^{1/2} - 1\right) \approx 0.83$



Another Example: Sufficiency of RM Schedulability Test

- Two periodic tasks τ_1 and τ_2 with
  - Phases $\Phi_1 = \Phi_2 = 0$
  - Periods $T_1 = 4$ and $T_2 = 8$, thus task τ_1 has higher priority than task τ_2
  - Worst-case execution times $C_1 = 2$ and $C_2 = 4$
- *Processor utilization U* $= \sum_{i=1}^{n} C_i/T_i = \frac{2}{4} + \frac{4}{8} = 1 > 2(2^{1/2} - 1) \approx 0.83$

Schedulable although the test fails



Deadline-Monotonic (DM) Scheduling

- Scheduling of periodic tasks τ_i where the relative deadlines may be smaller than the corresponding period (i.e., $C_i \leq D_i \leq T_i$). All other properties are as for RM:
  - Priorities are assigned to tasks before execution and do not change over time.
  - Currently executing task is preempted by higher-priority task.
- *DM scheduling rule:* Given a set of n independent periodic tasks τ_i with $D_i \leq T_i$, assign a fixed priority to each task such that tasks with shorter relative deadlines D_i have higher priorities. Thus, at any instant, the task with the shortest relative deadlines is executed. DM is optimal in the sense that if a task set is schedulable by some fixed-priority assignment, then it is also schedulable by DM.
- *Schedulability test*: A set of n periodic real-time tasks is schedulable using DM if

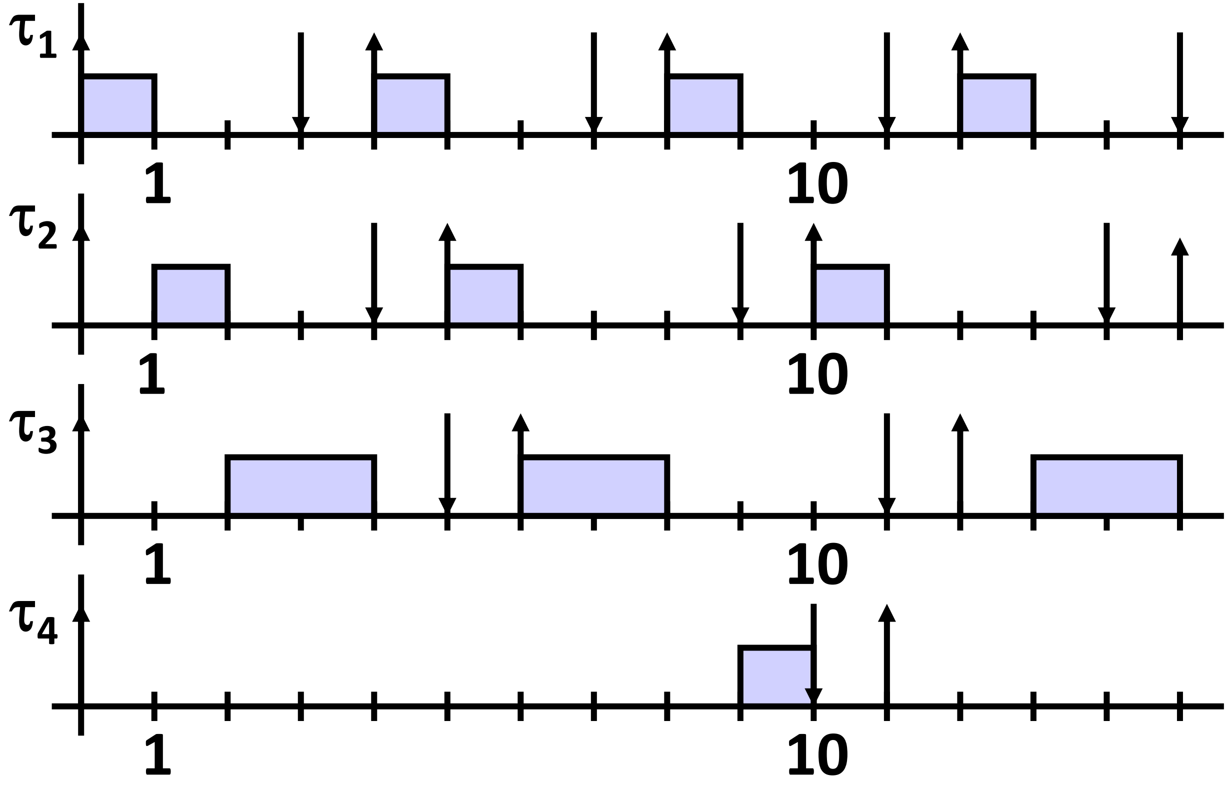
$$\sum_{i=1}^{n} C_i/D_i \leq n\left(2^{1/n} - 1\right)$$

  This condition is *sufficient but not necessary*.

Example: Sufficiency of DM Schedulability Test

- Four tasks are given:

τ_i	Φ_i	T_i	D_i	C_i
τ_1	0	4	3	1
τ_2	0	5	4	1
τ_3	0	6	5	2
τ_4	0	11	10	1



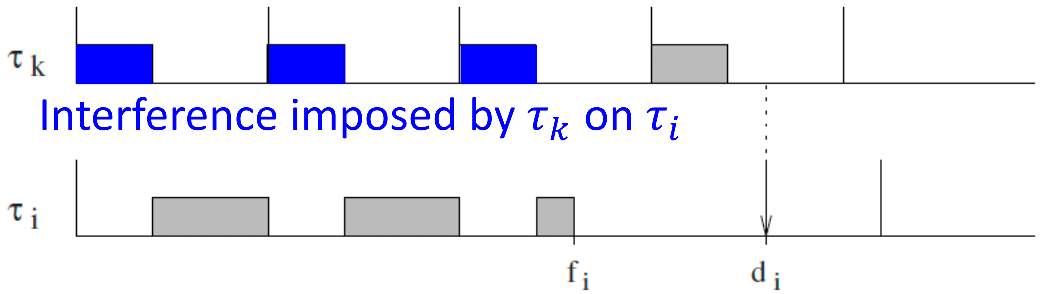
- Processor utilization: $U = \sum_{i=1}^{n} C_i/T_i = \frac{1}{4} + \frac{1}{5} + \frac{2}{6} + \frac{1}{11} \approx 0.87$

Schedulable although the test fails

- *Schedulability test*: $\sum_{i=1}^{n} C_i/D_i = \frac{1}{3} + \frac{1}{4} + \frac{2}{5} + \frac{1}{10} \approx 1.08 > 4\left(2^{1/4} - 1\right) \approx 0.76$

DM Necessary and Sufficient Schedulability Test (1)

- This computationally more involved test is based on the following observations:
  - The *worst-case processor demand* occurs when all tasks are released simultaneously, that is, at their critical instants.
  - To be schedulable, for each task τ_i in the set, the sum of its processing time and the *interference* imposed by all higher-priority tasks must be less than or equal to D_i.



  - The *worst-case interference* I_i for task τ_i can be computed as the sum of the processing times of all higher-priority tasks released before some time t, where tasks are ordered according to $\tau_m < \tau_n \Leftrightarrow D_m < D_n$:

$$I_i = \sum_{j=1}^{i-1} \left\lceil \frac{t}{T_j} \right\rceil C_j$$

DM Necessary and Sufficient Schedulability Test (2)

- The *longest response time R_i* of a periodic task τ_i is computed, at the critical instant, as the sum of its worst-case execution time C_i and the interference I_i due to preemption by higher-priority tasks:

$$R_i = C_i + I_i$$

- Hence, the schedulability test needs to compute, for all tasks τ_i, the smallest R_i that satisfies the following equality:

$$R_i = C_i + \sum_{j=1}^{i-1} \left\lceil \frac{R_i}{T_j} \right\rceil C_j$$

- Then, for a task set to be schedulable, $R_i \leq D_i$ must hold for all tasks τ_i.
- It can be shown that this *condition is necessary and sufficient*.

DM Necessary and Sufficient Schedulability Test: Algorithm

```
DM_guarantee (Γ) {
    for (each τ_i ∈ Γ) {
        I_i = Σ_{k=1}^{i-1} C_k;
        do {
            R_i = I_i + C_i;
            if (R_i > D_i) return(UNSCHEDULABLE);
            I_i = Σ_{k=1}^{i-1} ⌈R_i / T_k⌉ C_k;
        } while (I_i + C_i > R_i);
    }
    return(SCHEDULABLE);
}
```

DM_guarantee (Γ) {

    for (each $\tau_i \in \Gamma$) {

        $I_i = \sum_{k=1}^{i-1} C_k;$

        do {

            $R_i = I_i + C_i;$

            if $(R_i > D_i)$ return(UNSCHEDULABLE);

            $I_i = \sum_{k=1}^{i-1} \left\lceil \frac{R_i}{T_k} \right\rceil C_k;$

        } **while** $(I_i + C_i > R_i);$

    }

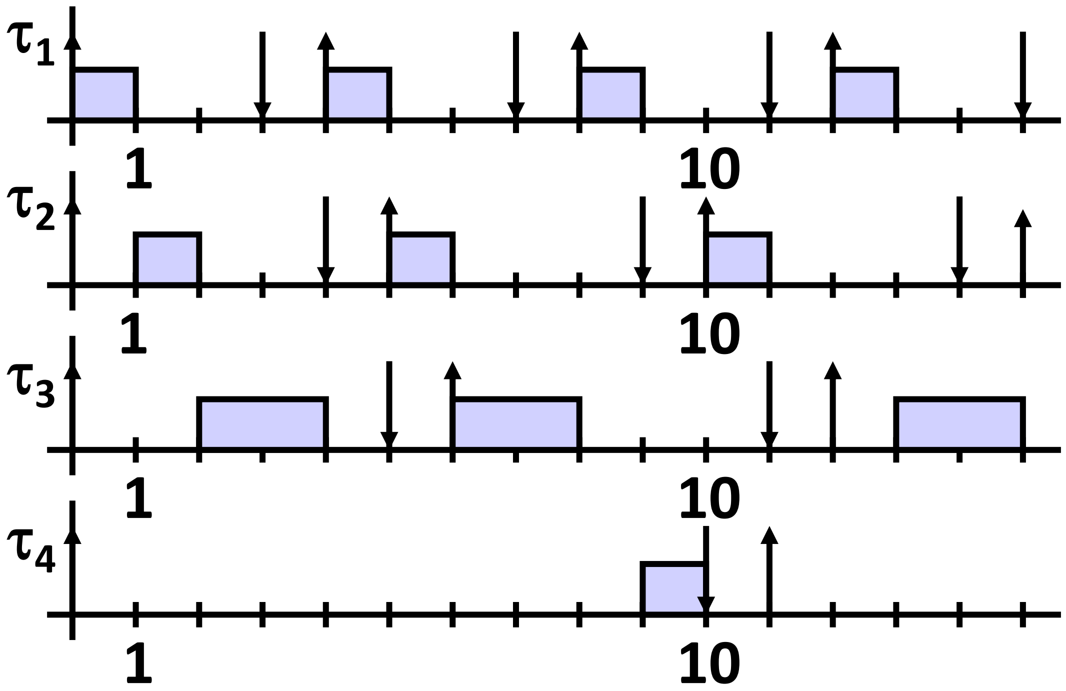
    return(SCHEDULABLE);

}

DM Necessary and Sufficient Schedulability Test: Example

- Four tasks are given:

τ_i	Φ_i	T_i	D_i	C_i
τ_1	0	4	3	1
τ_2	0	5	4	1
τ_3	0	6	5	2
τ_4	0	11	10	1



- What is R_4?

- Step 0: $R_4 = \sum_{i=1}^{4} C_i = \ldots$, $R_4 > D_4$?, $I_4 = \sum_{i=1}^{3} \left\lceil \frac{R_4}{T_i} \right\rceil = \cdots$, $I_4 + C_4 > R_4$?

- Step 1: $R_4 = I_4 + C_4 = \cdots$, $R_4 > D_4$?, $I_4 = \sum_{i=1}^{3} \left\lceil \frac{R_4}{T_i} \right\rceil = \cdots$, $I_4 + C_4 > R_4$?

- Step 2: …

DM Necessary and Sufficient Schedulability Test: Example

- Step 0: $R_4 = 5, I_4 = 5, I_4 + C_4 > R_4$
- Step 1: $R_4 = 6, I_4 = 6, I_4 + C_4 > R_4$
- Step 2: $R_4 = 7, I_4 = 8, I_4 + C_4 > R_4$
- Step 3: $R_4 = 9, I_4 = 9, I_4 + C_4 > R_4$
- Step 4: $R_4 = 10, I_4 = 9, I_4 + C_4 = R_4$

- This means that task τ_4 finishes at $R_4 = 10$, which can also be seen by looking at the schedule on the previous slides. Since $R_4 \leq D_4$, τ_4 is schedulable.
- If, like in this example, $R_i \leq D_i$ for all tasks τ_i in the task set, we can conclude that the task set is schedulable by DM.

Earliest Deadline First (EDF)

- As before, we consider preemptive scheduling of periodic tasks τ_i.
- *EDF algorithm*: A dynamic priority is assigned to each task such that tasks with earlier absolute deadlines have higher priorities. The currently executing task is preempted whenever a task with earlier absolute deadline becomes active. EDF is optimal in the sense that no other algorithm can schedule a set of periodic real-time tasks that cannot be scheduled by EDF.

- Using EDF the priorities are assigned *dynamically*, because the absolute deadline $d_{i,j}$ of a periodic task τ_i depends on the j-th instance that is currently active

$$d_{i,j} = \Phi_i + (j-1)T_i + D_i$$

- Instead, using RM (or DM) the priorities are *fixed*, because these are determined based on the periods T_i (or deadlines D_i) which do not change over time.

EDF Schedulability Test for Implicit Deadlines

- A set of n periodic real-time tasks, where $D_i = T_i$ for all tasks τ_i, is schedulable using EDF if and only if

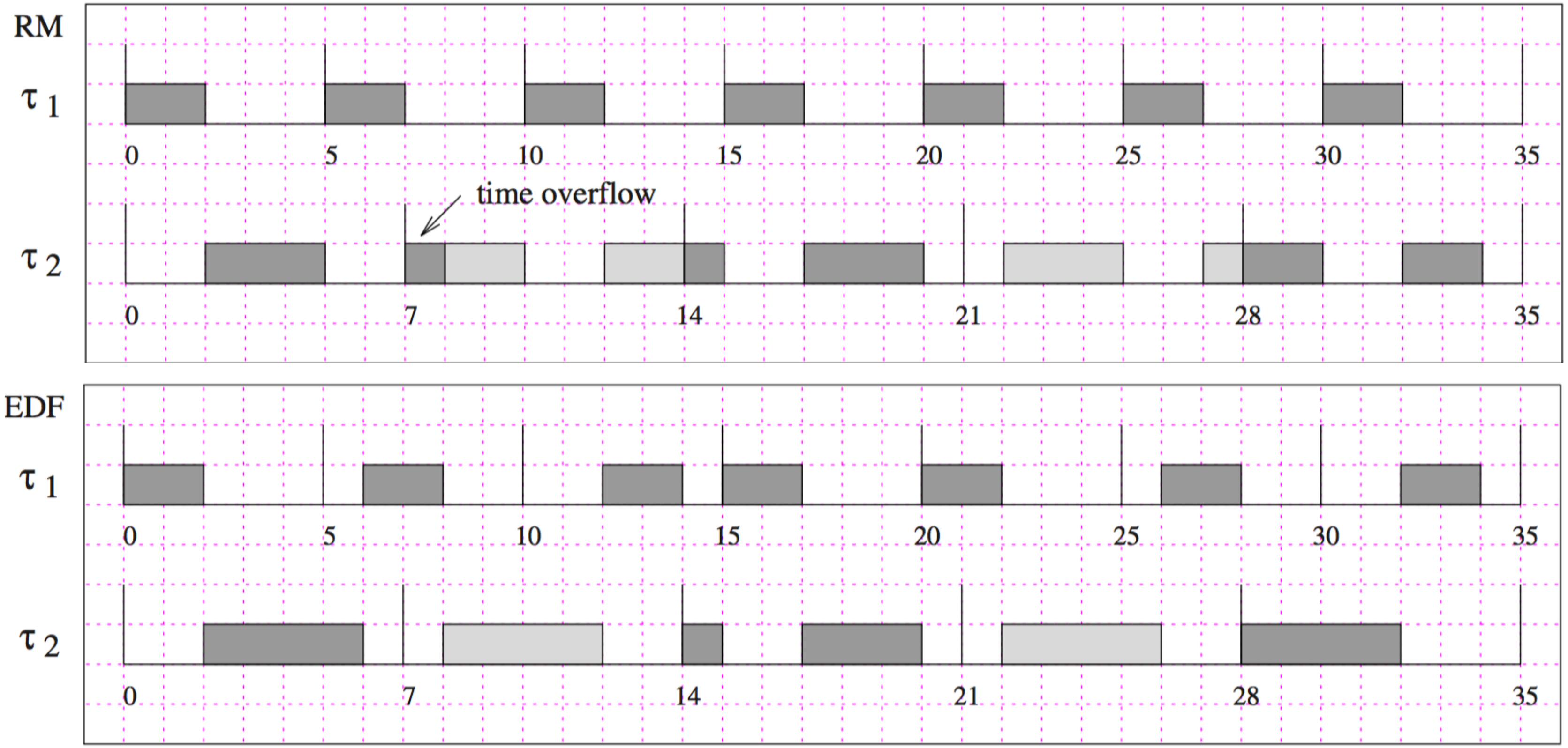
$$\sum_{i=1}^{n} C_i/T_i = U \leq 1$$

- This condition is both *necessary and sufficient*.

- Proof sketch:
  1. If the processor utilization satisfies $U > 1$, then there exists no valid schedule. This is because the total demand in the time interval $T = T_1 \cdot T_2 \cdot \cdots \cdot T_n$ is $\sum_{i=1}^{n} \frac{C_i}{T_i} T = UT > T$, which exceeds the available processor time in this interval.
  2. If the processor utilization satisfies $U \leq 1$, then there exists a valid schedule. We can prove this by contradiction: Assume that a deadline miss occurs at some time, then we can show that the processor utilization before this time exceeds 1.

Example: RM versus EDF

- Two periodic tasks τ_1 and τ_2 with
  - Phases $\Phi_1 = \Phi_2 = 0$
  - Periods $T_1 = 5$ and $T_2 = 7$
  - Worst-case execution times $C_1 = 2$ and $C_2 = 4$



$U \approx 0.97 > 0.83$

$U \approx 0.97 < 1$

No deadline miss and fewer preemptions due to dynamic priority assignment