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 Periodic Tasks

Overview

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

	Deadline equals period	Deadline smaller than period
static priority	RM (rate-monotonic)	DM (deadline-monotonic)
dynamic priority	EDF	EDF*

Model of Periodic Tasks

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

- Definitions:

Γ : denotes a set of periodic tasks

τ_i : denotes a periodic task

$\tau_{i,j}$: denotes the jth instance of task i

$r_{i,j}, s_{i,j}, f_{i,j}, d_{i,j}$: denote the release time, start time, finishing time, absolute deadline of the jth instance of task i

Φ_i : denotes the phase of task i (release time of its first instance)

D_i : denotes the relative deadline of task i

T_i : denotes the period of task i

Model of Periodic Tasks

- *The following hypotheses are assumed on the tasks:*
  - The instances of a periodic task are *regularly activated at a constant rate*. The interval T_i between two consecutive activations is called period. The release times satisfy

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

  - All instances have the *same worst case execution time* C_i
  - All instances of a periodic task have the *same relative deadline* D_i. Therefore, the absolute deadlines satisfy

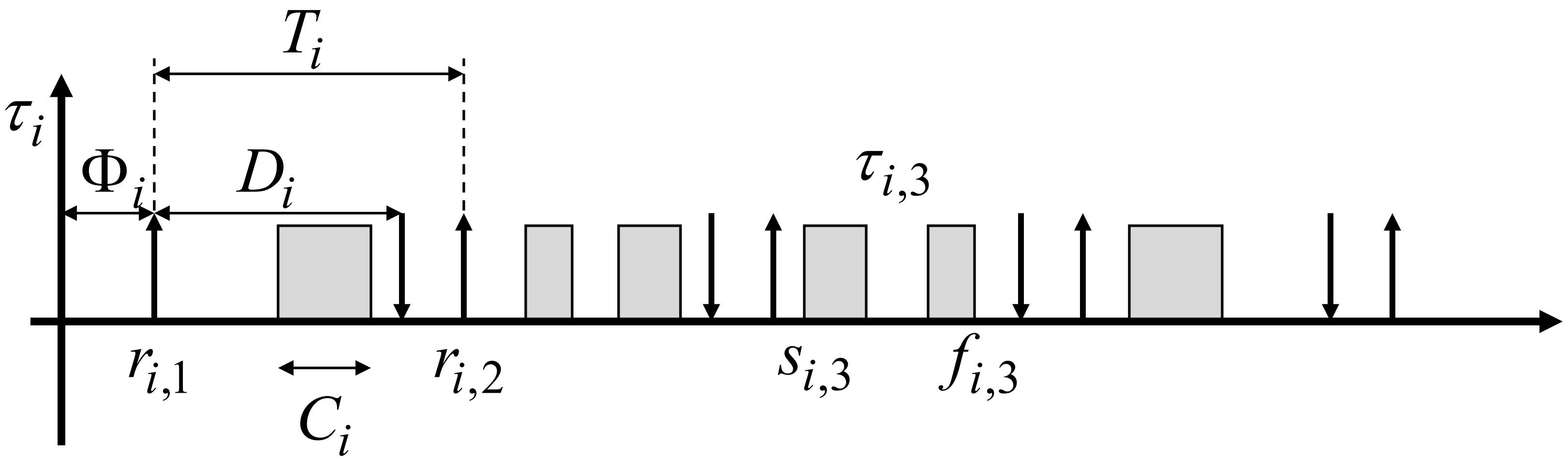
$$d_{i,j} = \Phi_i + (j-1)T_i + D_i$$

  - Often, the relative deadline equals the period $D_i = T_i$ (*implicit deadline*), and therefore

$$d_{i,j} = \Phi_i + jT_i$$

Model of Periodic Tasks

- *The following hypotheses are assumed on the tasks (continued):*
  - All periodic tasks are *independent*; that is, there are no precedence relations and no resource constraints.
  - *No task can suspend itself*, for example on I/O operations.
  - All tasks are *released as soon as they arrive*.
  - All *overheads* in the OS kernel are assumed to be *zero*.
  - *Example:*



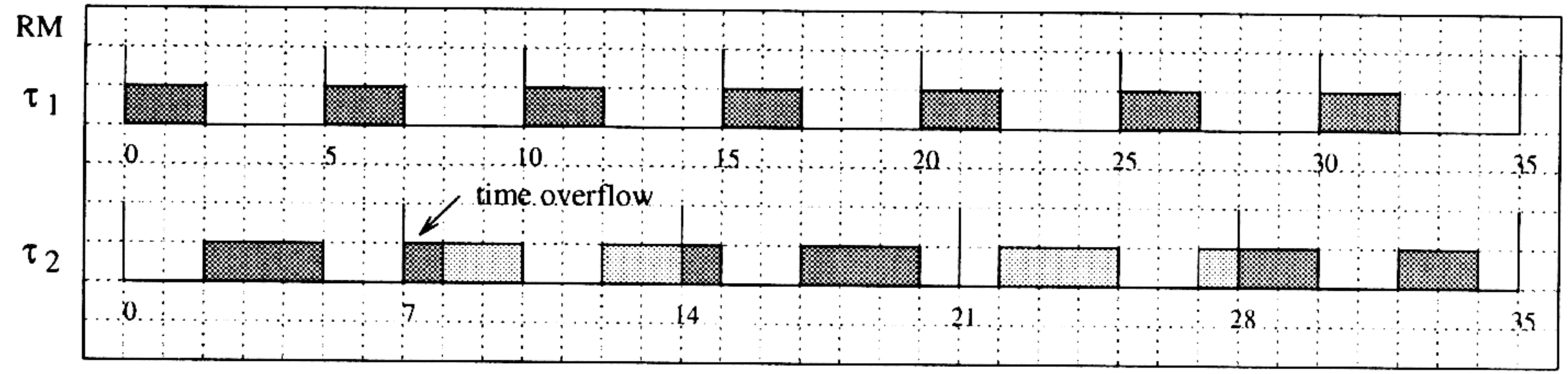
Rate Monotonic Scheduling (RM)

- *Assumptions:*
  - Task priorities are assigned to tasks before execution and do not change over time (static priority assignment).
  - RM is intrinsically preemptive: the currently executing job is preempted by a job of a task with higher priority.
  - Deadlines equal the periods $D_i = T_i$.

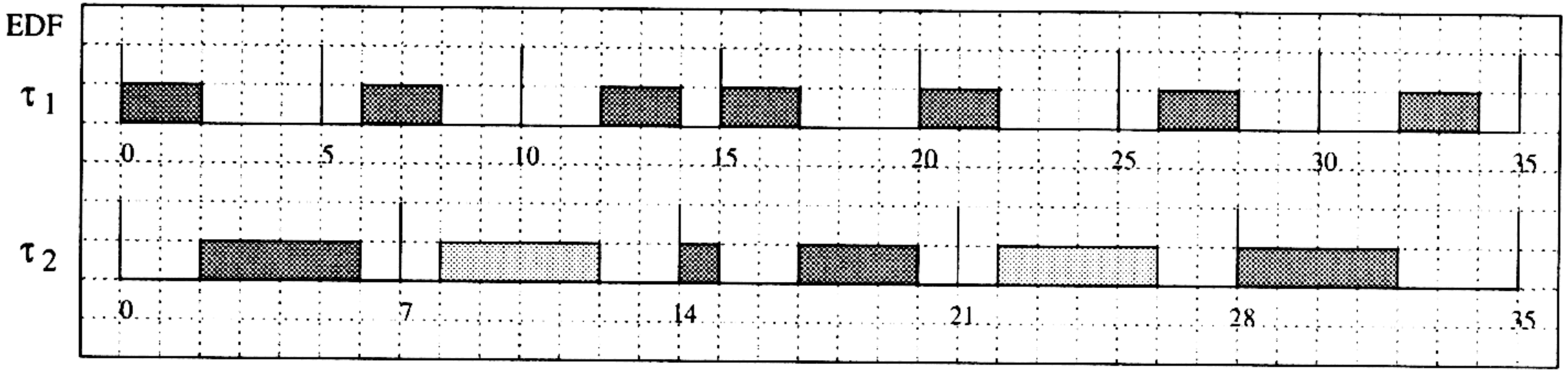
Rate-Monotonic Scheduling Algorithm: Each task is assigned a priority. Tasks with higher request rates (that is with shorter periods) will have higher priorities. Jobs of tasks with higher priority interrupt jobs of tasks with lower priority.

Periodic Tasks

Example: 2 tasks, deadlines = periods, utilization = 97%



(a)



(b)

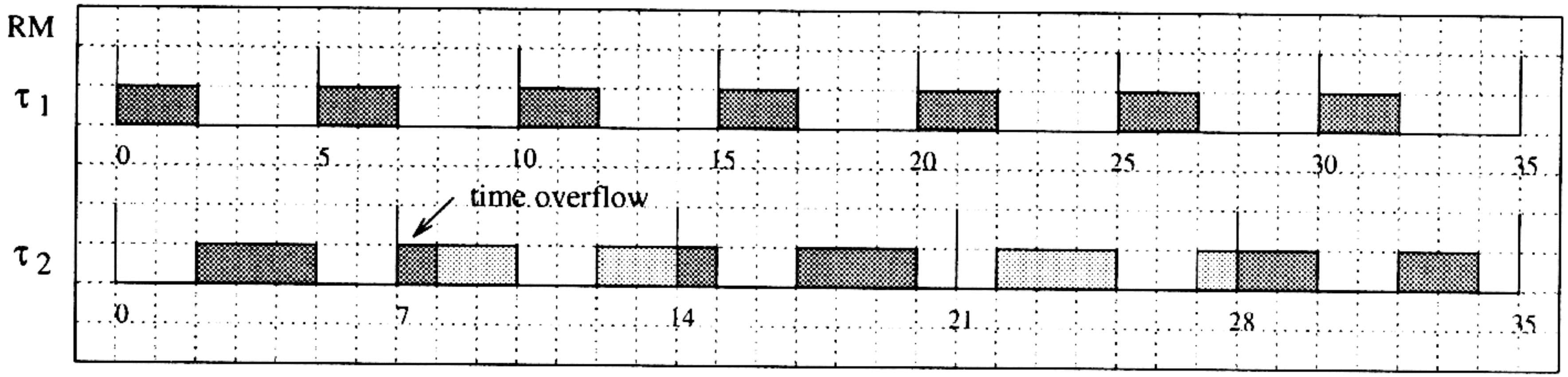
Rate Monotonic Scheduling (RM)

Optimality: 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.

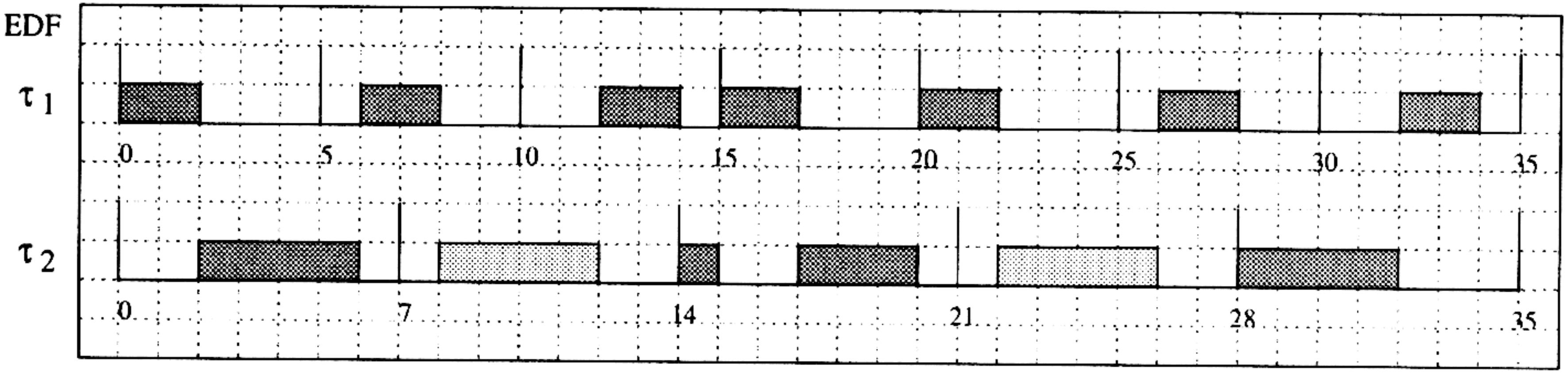
- The *proof* is done by considering several cases that may occur, but the main ideas are as follows:
  - A *critical instant* for any task occurs whenever the task is released simultaneously with all higher priority tasks. The tasks schedulability can easily be checked at their critical instants. If all tasks are feasible at their critical instant, then the task set is schedulable in any other condition.
  - Show that, given two periodic tasks, if the schedule is feasible by an arbitrary priority assignment, then it is also feasible by RM.
  - Extend the result to a set of n periodic tasks.

Periodic Tasks

Example: 2 tasks, deadlines = periods, utilization = 97%



(a)



(b)

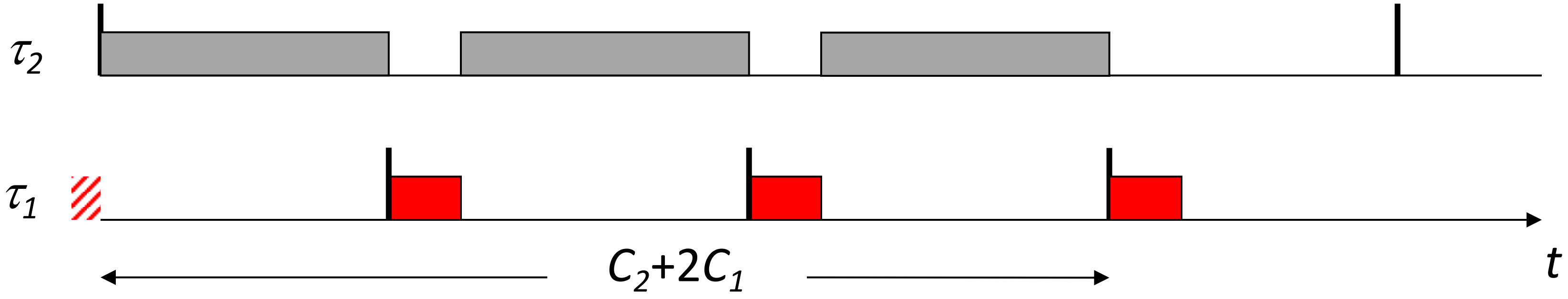
Proof of Critical Instance

Definition: A critical instant of a task is the time at which the release of a job will produce the largest response time.

Lemma: For any task, the critical instant occurs if a job is simultaneously released with all higher priority jobs.

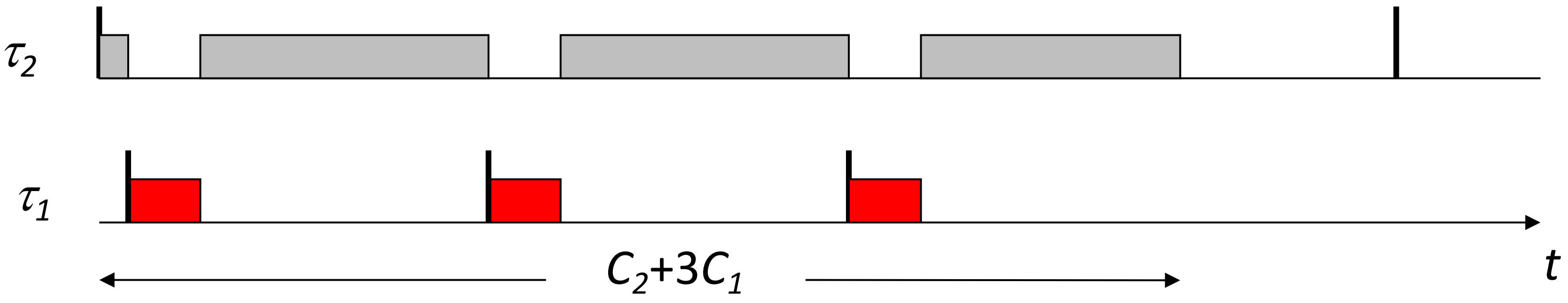
Proof sketch: Start with 2 tasks τ_1 and τ_2 .

Response time of a job of τ_2 is delayed by jobs of τ_1 of higher priority:



Proof of Critical Instance

Delay may increase if τ_1 starts earlier:



Maximum delay achieved if τ_2 and τ_1 start simultaneously.

Repeating the argument for all higher priority tasks of some task τ_2 :

> The worst case response time of a job occurs when it is released simultaneously with all higher-priority jobs.

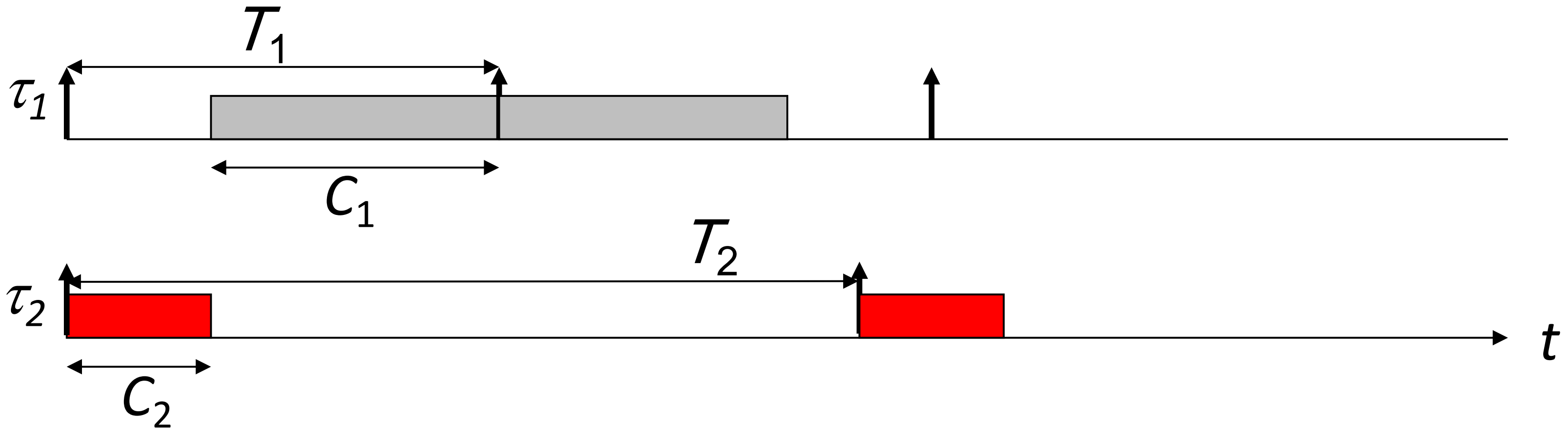
Proof of RM Optimality (2 Tasks)

We have two tasks τ_1, τ_2 with periods $T_1 < T_2$.

Define $F = \lfloor T_2/T_1 \rfloor$: the number of periods of τ_1 **fully** contained in T_2

Consider two cases A and B:

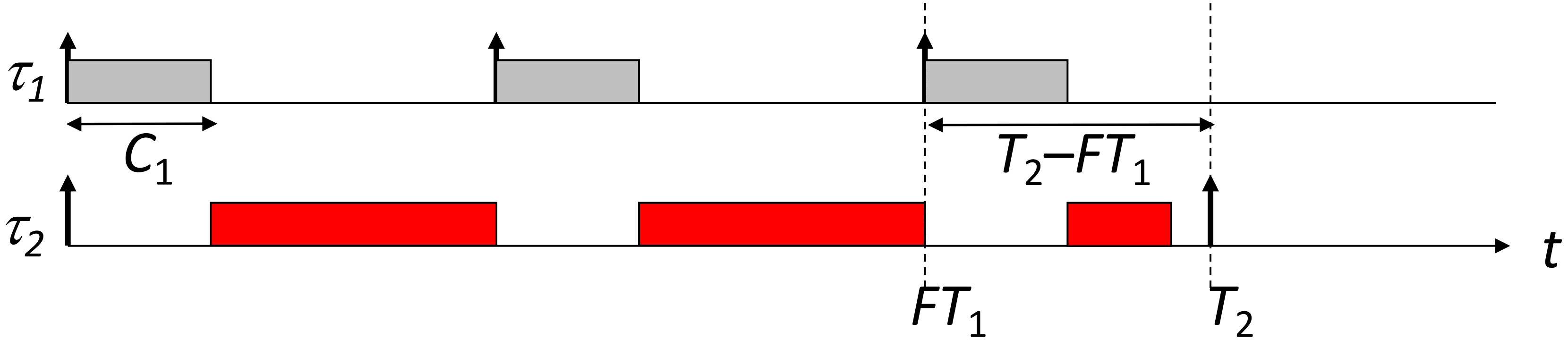
Case A: Assume RM is not used → prio(τ_2) is highest:



Schedule is feasible if $C_1 + C_2 \leq T_1$ and $C_2 \leq T_2$ (A)

Proof of RM Optimality (2 Tasks)

Case B: Assume RM **is** used → prio(τ_1) is highest:



Schedulable is feasible if

$$FC_1 + C_2 + \min(T_2 - FT_1, C_1) \leq T_2 \text{ and } C_1 \leq T_1 \qquad \text{(B)}$$

We need to show that (A) $\Rightarrow$ (B): $C_1 + C_2 \leq T_1 \Rightarrow C_1 \leq T_1$

$$C_1 + C_2 \leq T_1 \Rightarrow FC_1 + C_2 \leq FC_1 + FC_2 \leq F\,T_1 \Rightarrow$$

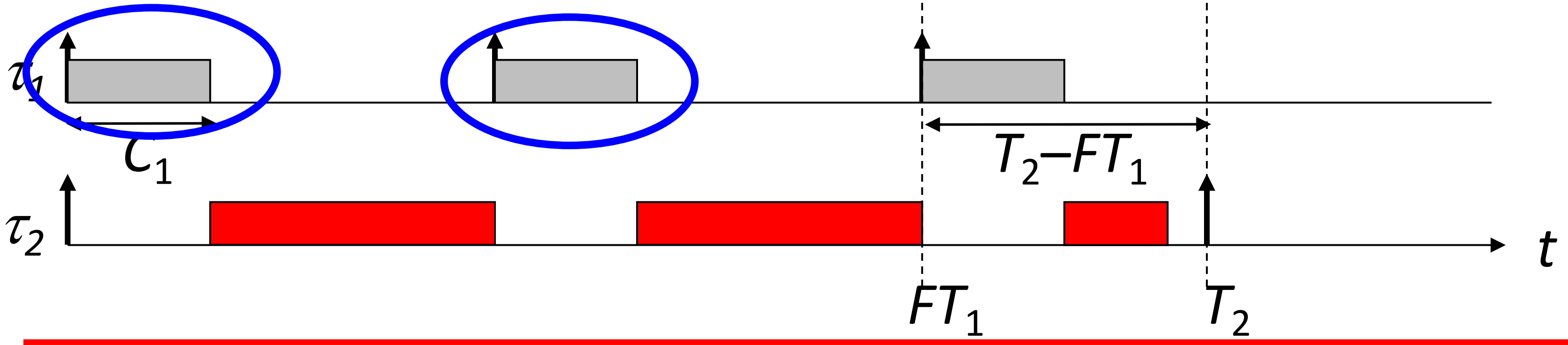
$$FC_1 + C_2 + \min(T_2 - FT_1, C_1) \leq F\,T_1 + \min(T_2 - FT_1, C_1) \leq \min(T_2, C_1 + FT_1) \leq T_2$$

Given tasks τ_1 and τ_2 with $T_1 < T_2$, then if the schedule is feasible by an arbitrary fixed priority assignment, it is also feasible by RM.

Proof of RM Optimality (2 Tasks)

Proof of RM Optimality (2 Tasks)

Case B: Assume RM **is** used → prio(τ_1) is highest:



Schedulable is feasible if

$$FC_1 + C_2 + \min(T_2 - FT_1, C_1) \leq T_2 \text{ and } C_1 \leq T_1 \qquad \text{(B)}$$

We need to show that (A) $\Rightarrow$ (B): $C_1 + C_2 \leq T_1 \Rightarrow C_1 \leq T_1$

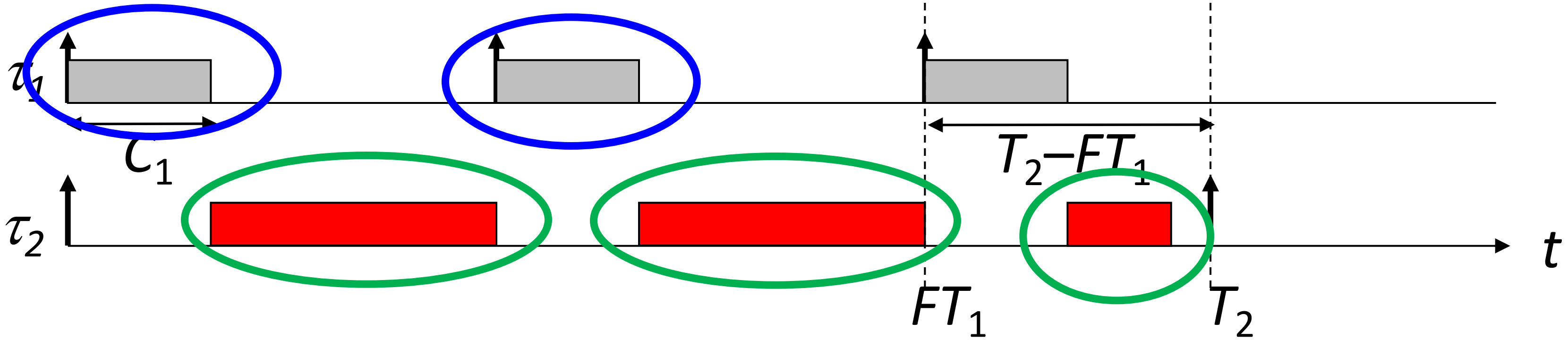
$$C_1 + C_2 \leq T_1 \Rightarrow FC_1 + C_2 \leq FC_1 + FC_2 \leq F\,T_1 \Rightarrow$$

$$FC_1 + C_2 + \min(T_2 - FT_1, C_1) \leq F\,T_1 + \min(T_2 - FT_1, C_1) \leq \min(T_2, C_1 + FT_1) \leq T_2$$

Given tasks τ_1 and τ_2 with $T_1 < T_2$, then if the schedule is feasible by an arbitrary fixed priority assignment, it is also feasible by RM.

Proof of RM Optimality (2 Tasks)

Case B: Assume RM **is** used → prio(τ_1) is highest:



Schedulable is feasible if

$$FC_1+C_2+\min(T_2-FT_1,\ C_1) \le T_2 \text{ and } C_1 \le T_1 \qquad \text{(B)}$$

We need to show that (A) $\Rightarrow$ (B): $C_1+C_2 \le T_1 \Rightarrow C_1 \le T_1$

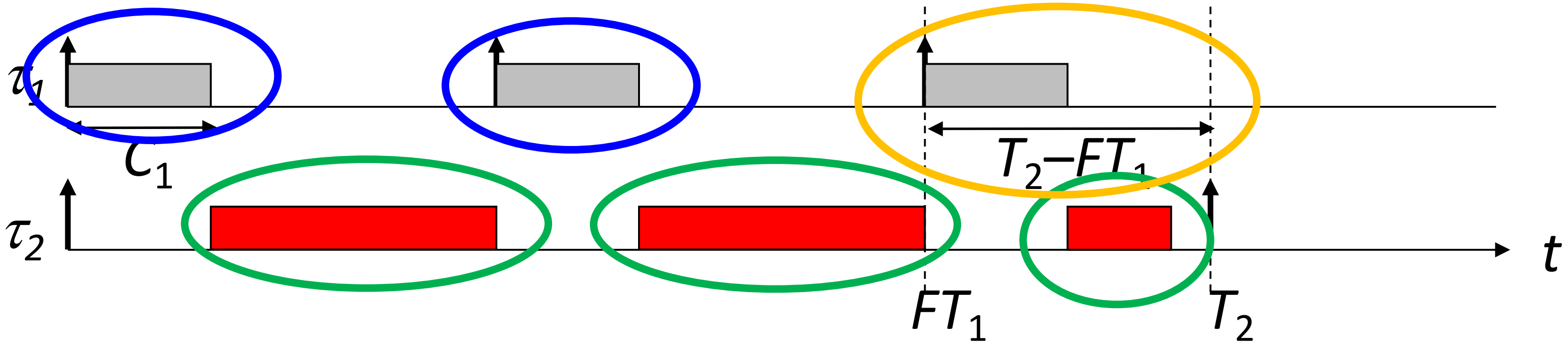
$$C_1+C_2 \le T_1 \Rightarrow FC_1+C_2 \le FC_1+FC_2 \le FT_1 \Rightarrow$$

$$FC_1+C_2+\min(T_2-FT_1,\ C_1) \le FT_1 + \min(T_2-FT_1,\ C_1) \le \min(T_2,\ C_1+FT_1) \le T_2$$

Given tasks τ_1 and τ_2 with $T_1 < T_2$, then if the schedule is feasible by an arbitrary fixed priority assignment, it is also feasible by RM.

Proof of RM Optimality (2 Tasks)

Case B: Assume RM **is** used → prio(τ_1) is highest:



Schedulable is feasible if

$FC_1+C_2+\min(T_2-FT_1, C_1) \leq T_2$ and $C_1 \leq T_1$ (B)

We need to show that (A) $\Rightarrow$ (B): $C_1+C_2 \leq T_1 \Rightarrow C_1 \leq T_1$

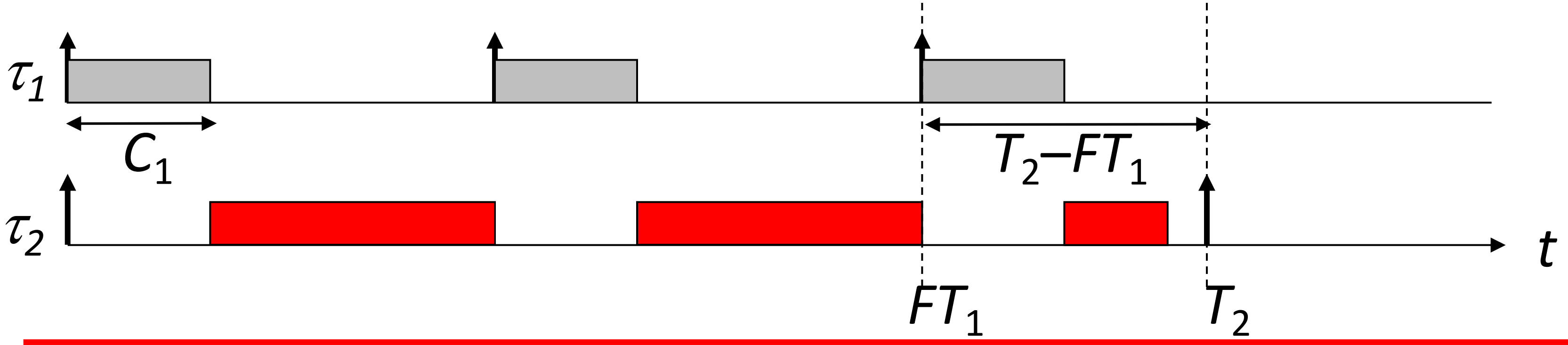
$C_1+C_2 \leq T_1 \Rightarrow FC_1+C_2 \leq FC_1+FC_2 \leq F\,T_1 \Rightarrow$

$FC_1+C_2+\min(T_2-FT_1, C_1) \leq F\,T_1 + \min(T_2-FT_1, C_1) \leq \min(T_2, C_1+FT_1) \leq T_2$

Given tasks τ_1 and τ_2 with $T_1 < T_2$, then if the schedule is feasible by an arbitrary fixed priority assignment, it is also feasible by RM.

Proof of RM Optimality (2 Tasks)

Case B: Assume RM **is** used → prio(τ_1) is highest:



Schedulable is feasible if

$$FC_1 + C_2 + \min(T_2 - FT_1, C_1) \leq T_2 \text{ and } C_1 \leq T_1 \qquad \text{(B)}$$

We need to show that (A) $\Rightarrow$ (B): $C_1 + C_2 \leq T_1 \Rightarrow C_1 \leq T_1$

$$C_1 + C_2 \leq T_1 \Rightarrow FC_1 + C_2 \leq FC_1 + FC_2 \leq FT_1 \Rightarrow$$

$$FC_1 + C_2 + \min(T_2 - FT_1, C_1) \leq FT_1 + \min(T_2 - FT_1, C_1) \leq \min(T_2, C_1 + FT_1) \leq T_2$$

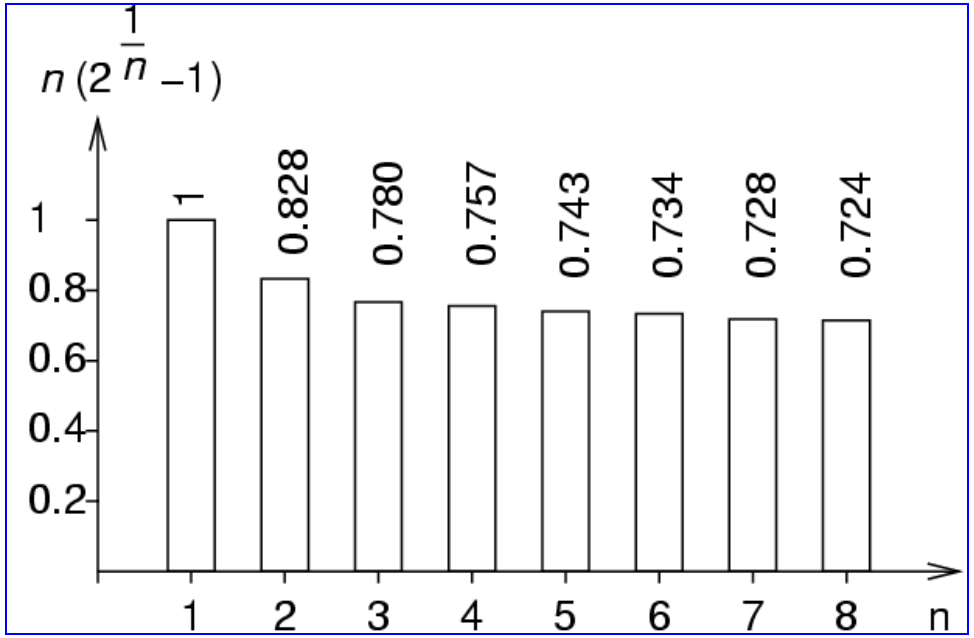
Given tasks τ_1 and τ_2 with $T_1 < T_2$, then if the schedule is feasible by an arbitrary fixed priority assignment, it is also feasible by RM.

Admittance Test

Rate Monotonic Scheduling (RM)

Schedulability analysis: A set of periodic tasks is schedulable with RM if

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



This condition is sufficient but not necessary.

The term $U = \sum_{i=1}^{n} \frac{C_i}{T_i}$ denotes the *processor utilization factor U* which is the fraction of processor time spent in the execution of the task set.

Proof of Utilization Bound (2 Tasks)

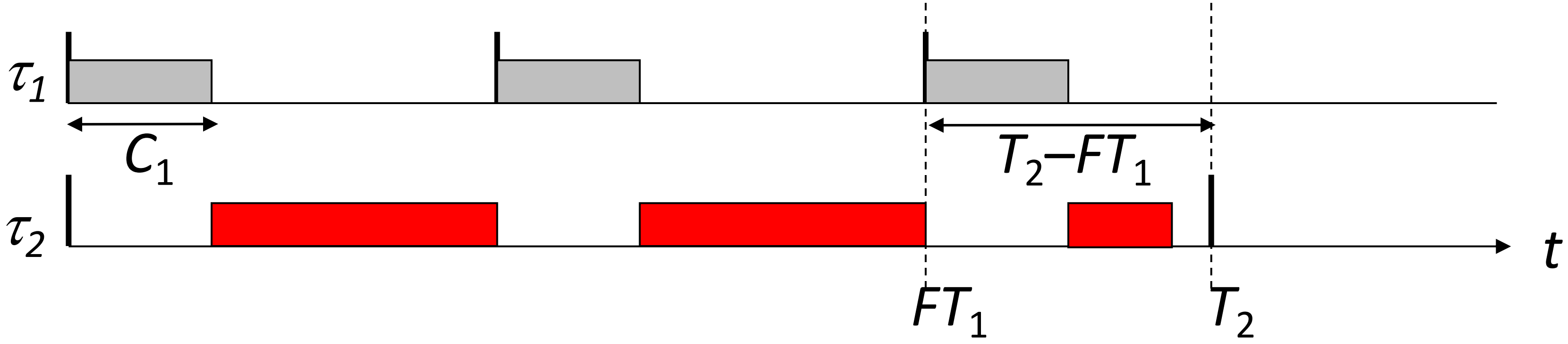
We have two tasks τ_1, τ_2 with periods $T_1 < T_2$.
Define $F = \lfloor T_2/T_1 \rfloor$: number of periods of τ_1 **fully** contained in T_2

Proof Concept: Compute upper bound on utilization U such that the task set is still schedulable:

- assign priorities according to RM;
- compute upper bound U_{up} by increasing the computation time C_2 to just meet the deadline of τ_2; we will determine this limit of C_2 using the results of the RM optimality proof.
- minimize upper bound with respect to other task parameters in order to find the utilization below which the system is definitely schedulable.

Proof of Utilization Bound (2 Tasks)

As before:



Schedulable if $FC_1 + C_2 + \min(T_2 – FT_1, C_1) \leq T_2$ and $C_1 \leq T_1$

Utilization:

$$U = \frac{C_1}{T_1} + \frac{C_2}{T_2} = \frac{C_1}{T_1} + \frac{T_2 - FC_1 - \min\{T_2 - FT_1, C_1\}}{T_2}$$

$$= 1 + \frac{C_1(T_2 - FT_1) - T_1 \min\{T_2 - FT_1, C_1\}}{T_1 T_2}$$

Proof of Utilization Bound (2 Tasks)

$$U = \frac{C_1}{T_1} + \frac{C_2}{T_2} = \frac{C_1}{T_1} + \frac{T_2 - FC_1 - \min\{T_2 - FT_1, C_1\}}{T_2}$$

$$= 1 + \frac{C_1(T_2 - FT_1) - T_1 \min\{T_2 - FT_1, C_1\}}{T_1 T_2}$$

Proof of Utilization Bound (2 Tasks)

Minimize utilization bound w.r.t C_1:

- If $C_1 \leq T_2 – FT_1$ then U decreases with increasing C_1
- If $T_2 – FT_1 \leq C_1$ then U decreases with decreasing C_1
- Therefore, minimum U is obtained with $C_1 = T_2 – FT_1$:

$$U = 1 + \frac{(T_2 - FT_1)^2 - T_1(T_2 - FT_1)\}}{T_1 T_2}$$

$$= 1 + \frac{T_1}{T_2}((\frac{T_2}{T_1} - F)^2 - (\frac{T_2}{T_1} - F))$$

We now need to minimize w.r.t. $G = T_2/T_1$ where $F = \lfloor T_2/T_1 \rfloor$ and $T_1 < T_2$. As F is integer, we first suppose that it is independent of $G = T_2/T_1$. Then we obtain

$$U = \frac{T_1}{T_2}((\frac{T_2}{T_1} - F)^2 + F)) = \frac{(G-F)^2 + F}{G}$$

Proof of Utilization Bound (2 Tasks)

Minimizing *U* with respect to *G* yields

$$2G(G-F)-(G-F)^2-F=G^2-(F^2+F)=0$$

If we set $F = 1$, then we obtain

$$G=\frac{T_2}{T_1}=\sqrt{2} \qquad \boxed{U=2(\sqrt{2}-1)}$$

It can easily be checked, that all other integer values for *F* lead to a larger upper bound on the utilization.

Deadline Monotonic Scheduling (DM)

- Assumptions are as in rate monotonic scheduling, but *deadlines may be smaller than the period*, i.e.

$$C_i \leq D_i \leq T_i$$

Algorithm: Each task is assigned a priority. Tasks with smaller relative deadlines will have higher priorities. Jobs with higher priority interrupt jobs with lower priority.

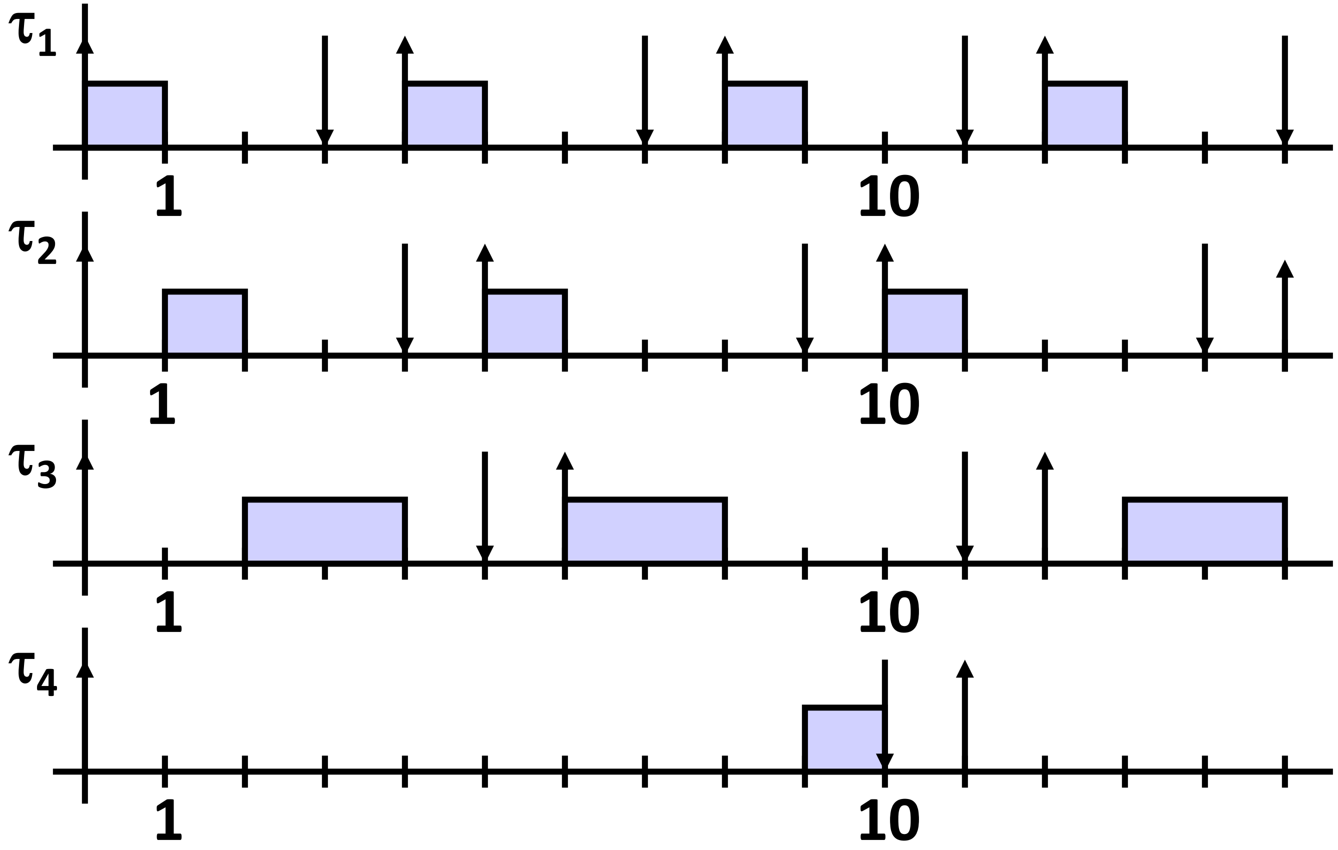
- *Schedulability Analysis:* A set of periodic tasks is schedulable with DM if

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

This condition is sufficient but not necessary (in general).

Deadline Monotonic Scheduling (DM) - Example

$U = 0.874$ $$\sum_{i=1}^{n} \frac{C_i}{D_i} = 1.08 > n\left(2^{1/n} - 1\right) = 0.757$$



Deadline Monotonic Scheduling (DM)

There is also a *necessary and sufficient schedulability test* which is computationally more involved. It is based on the following observations:

- The *worst-case processor demand* occurs when all tasks are released simultaneously; that is, at their critical instances.
- For each task *i*, the sum of its processing time and the *interference* imposed by higher priority tasks must be less than or equal to D_i .
- A measure of the *worst case interference* for task 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 $m < n \Leftrightarrow D_m < D_n$:

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

Deadline Monotonic Scheduling (DM)

- The *longest response time* R_i of a job of a periodic task i is computed, at the critical instant, as the sum of its computation time and the interference due to preemption by higher priority tasks:

$$R_i = C_i + I_i$$

- Hence, the schedulability test needs to compute the smallest R_i that satisfies

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

for all tasks i. Then, $R_i \leq D_i$ must hold for all tasks i.

- It can be shown that this *condition is necessary and sufficient*.

Deadline Monotonic Scheduling (DM)

The longest response times R_i of the periodic tasks *i* can be computed iteratively by the following algorithm:

```
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);
}
```

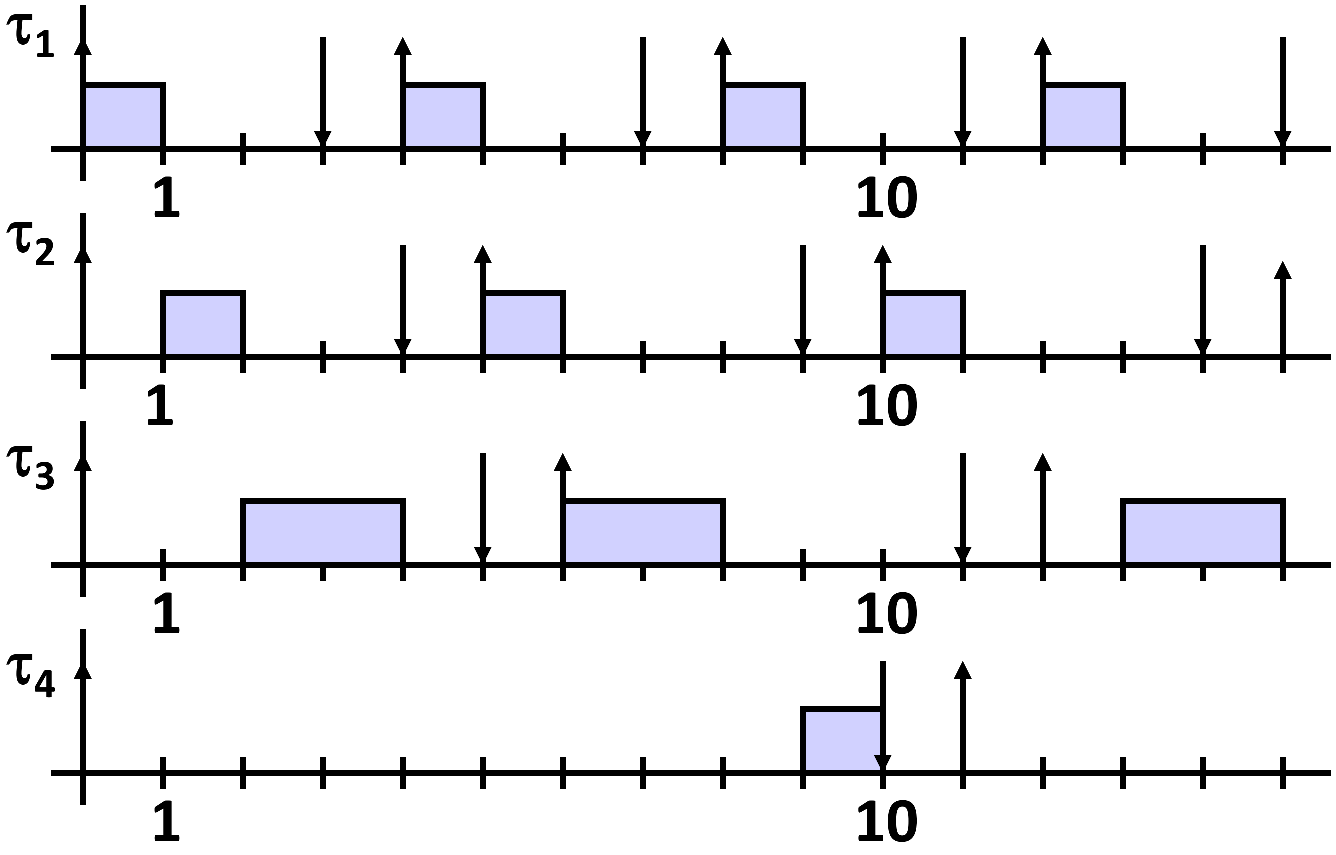
DM Example

Example:

- Task 1: $C_1 = 1; T_1 = 4; D_1 = 3$
- Task 2: $C_2 = 1; T_2 = 5; D_2 = 4$
- Task 3: $C_3 = 2; T_3 = 6; D_3 = 5$
- Task 4: $C_4 = 1; T_4 = 11; D_4 = 10$
- Algorithm for the schedulability test for task 4:
  - Step 0: $R_4 = 1$
  - Step 1: $R_4 = 5$
  - Step 2: $R_4 = 6$
  - Step 3: $R_4 = 7$
  - Step 4: $R_4 = 9$
  - Step 5: $R_4 = 10$

DM Example

$U = 0.874$ $\sum_{i=1}^{n} \frac{C_i}{D_i} = 1.08 > n\left(2^{1/n} - 1\right) = 0.757$



EDF Scheduling (earliest deadline first)

- *Assumptions:*
  - dynamic priority assignment
  - intrinsically preemptive

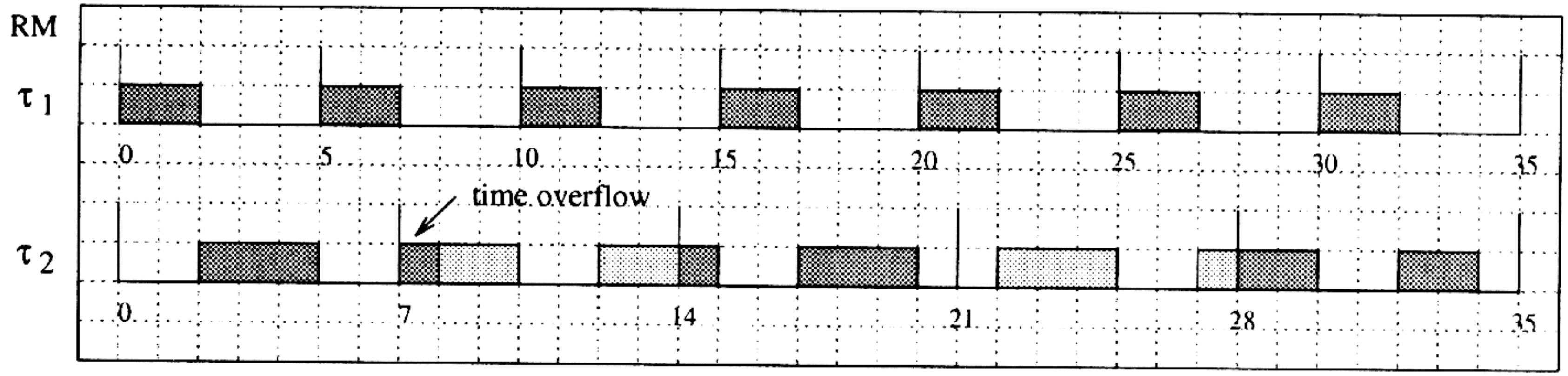
- *Algorithm:* The currently executing task is preempted whenever another periodic instance with earlier deadline becomes active.

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

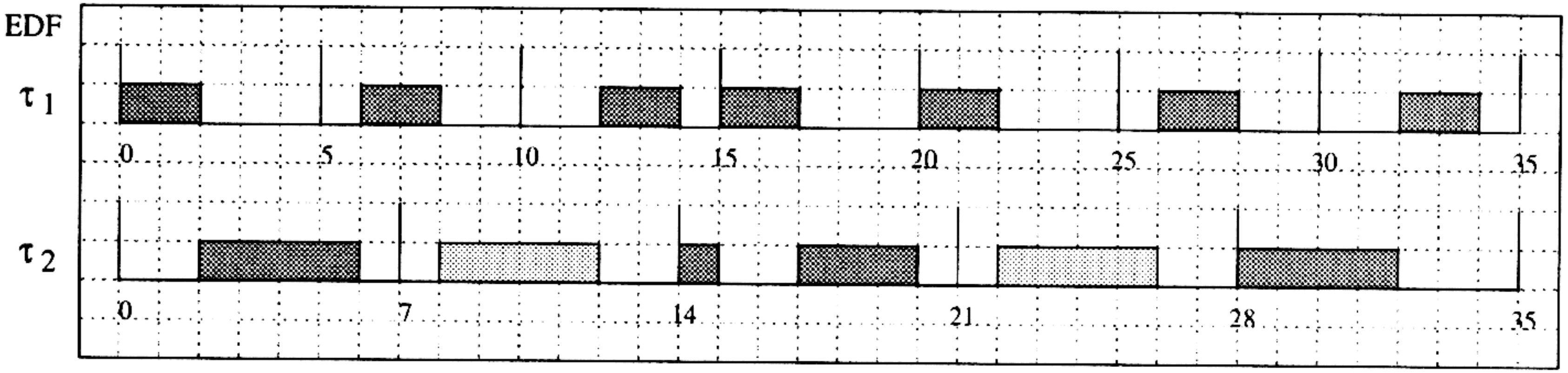
- *Optimality:* No other algorithm can schedule a set of periodic tasks if the set that can not be scheduled by EDF.
- The proof is simple and follows that of the aperiodic case.

Periodic Tasks

Example: 2 tasks, deadlines = periods, utilization = 97%



(a)



(b)

EDF Scheduling

A *necessary and sufficient schedulability test* for $D_i = T_i$:

A set of periodic tasks is schedulable with EDF if and only if $\sum_{i=1}^{n} \frac{C_i}{T_i} = U \leq 1$

The term $U = \sum_{i=1}^{n} \frac{C_i}{T_i}$ denotes the *average processor utilization.*

EDF Scheduling

- If the utilization satisfies $U > 1$, then there is no valid schedule: The total demand of computation time in interval $T = T_1 \cdot T_2 \cdot \ldots \cdot T_n$ is

$$\sum_{i=1}^{n} \frac{C_i}{T_i} T = UT > T$$

  and therefore, it exceeds the available processor time in this interval.

- If the utilization satisfies $U \leq 1$, then there is a valid schedule.

  We will proof this fact by contradiction: Assume that deadline is missed at some time t_2 . Then we will show that the utilization was larger than 1.

EDF Scheduling

- *If the deadline was missed* at t_2 then define t_1 as a time before t_2 such that (a) the processor is continuously busy in $[t_1, t_2]$ and (b) the processor only executes tasks that have their arrival time AND their deadline in $[t_1, t_2]$.
- *Why does such a time t_1 exist?* We find such a t_1 by starting at t_2 and going backwards in time, always ensuring that the processor only executed tasks that have their deadline before or at t_2 :
  - Because of EDF, the processor will be busy shortly before t_2 and it executes on the task that has deadline at t_2.
  - Suppose that we reach a time such that shortly before the processor works on a task with deadline after t_2 or the processor is idle, then we found t_1: We know that there is no execution on a task with deadline after t_2 .
    - But it could be in principle, that a task that arrived before t_1 is executing in $[t_1, t_2]$.
    - If the processor is idle before t_1, then this is clearly not possible due to EDF (the processor is not idle, if there is a ready task).
    - If the processor is not idle before t_1, this is not possible as well. Due to EDF, the processor will always work on the task with the closest deadline and therefore, once starting with a task with deadline after t_2 all task with deadlines before t_2 are finished.

EDF Scheduling

- Within the interval $[t_1, t_2]$ the total *computation time demanded* by the periodic tasks is bounded by

$$C_p(t_1,t_2) = \sum_{i=1}^{n} \left\lfloor \frac{t_2 - t_1}{T_i} \right\rfloor C_i \leq \sum_{i=1}^{n} \frac{t_2 - t_1}{T_i} C_i = (t_2 - t_1) U$$

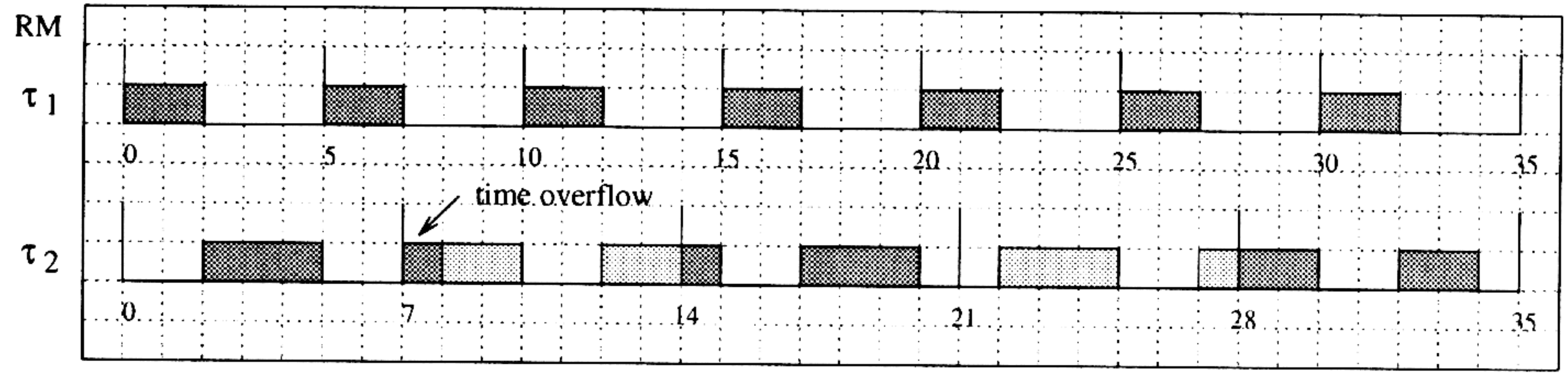
number of complete periods of task i in the interval

- Since the deadline at time t_2 is missed, we must have:

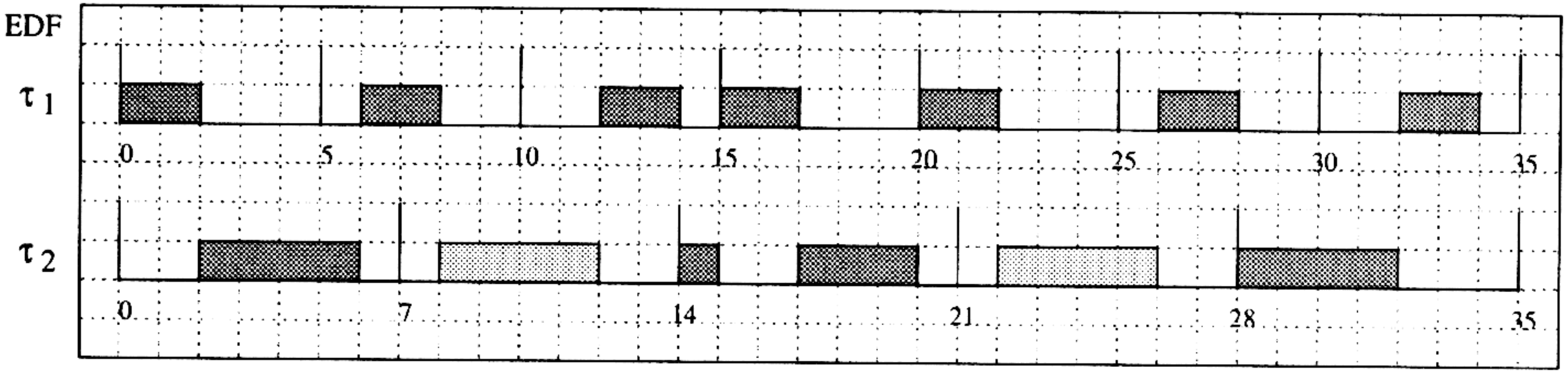
$$t_2 - t_1 < C_p(t_1, t_2) \leq (t_2 - t_1) U \;\Rightarrow\; U > 1$$

Periodic Task Scheduling

Example: 2 tasks, deadlines = periods, utilization = 97%



(a)



(b)

Real-Time Scheduling of Mixed Task Sets

Problem of Mixed Task Sets

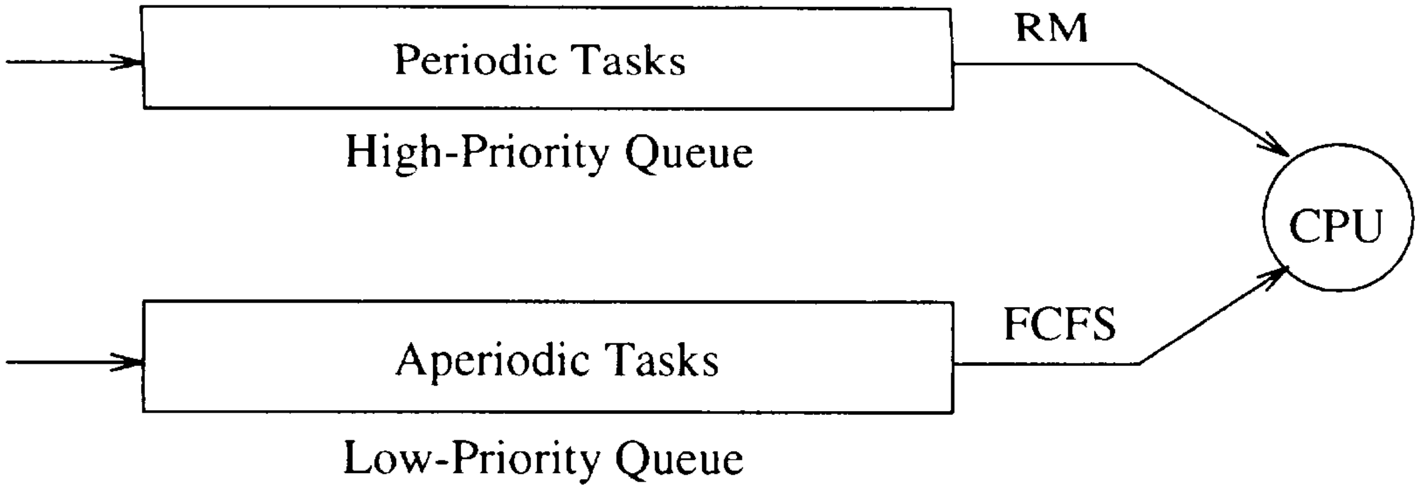
In many applications, there are aperiodic as well as periodic tasks.

- *Periodic tasks: time-driven*, execute critical control activities with hard timing constraints aimed at guaranteeing regular activation rates.
- *Aperiodic tasks: event-driven*, may have hard, soft, non-real-time requirements depending on the specific application.
- *Sporadic tasks:* Offline guarantee of event-driven aperiodic tasks with critical timing constraints can be done only by making proper assumptions on the environment; that is by assuming a *maximum arrival rate* for each critical event. Aperiodic tasks characterized by a minimum interarrival time are called sporadic.

Background Scheduling

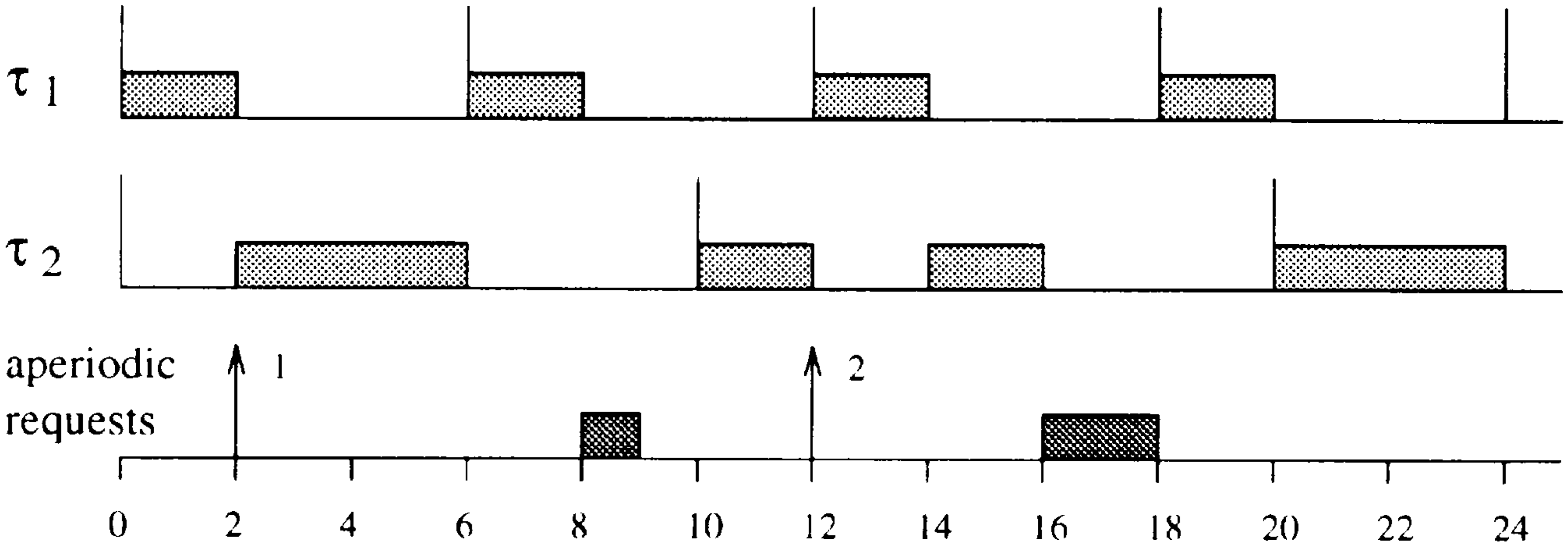
Background scheduling is a simple solution for RM and EDF:

- Processing of aperiodic tasks in the background, i.e. execute if there are no pending periodic requests.
- Periodic tasks are not affected.
- Response of aperiodic tasks may be prohibitively long and there is no possibility to assign a higher priority to them.
- Example:



Background Scheduling

Example (rate monotonic periodic schedule):



Rate-Monotonic Polling Server

- ***Idea:*** Introduce an artificial periodic task whose purpose is to service aperiodic requests as soon as possible (therefore, “server”).
- Function of *polling server (PS)*
  - At regular intervals equal to T_s, a PS task is instantiated. When it has the highest current priority, it serves any pending aperiodic requests within the limit of its capacity C_s.
  - If no aperiodic requests are pending, PS suspends itself until the beginning of the next period and the time originally allocated for aperiodic service is not preserved for aperiodic execution.
  - Its priority (period!) can be chosen to match the response time requirement for the aperiodic tasks.
- *Disadvantage:* If an aperiodic requests arrives just after the server has suspended, it must wait until the beginning of the next polling period.

Rate-Monotonic Polling Server

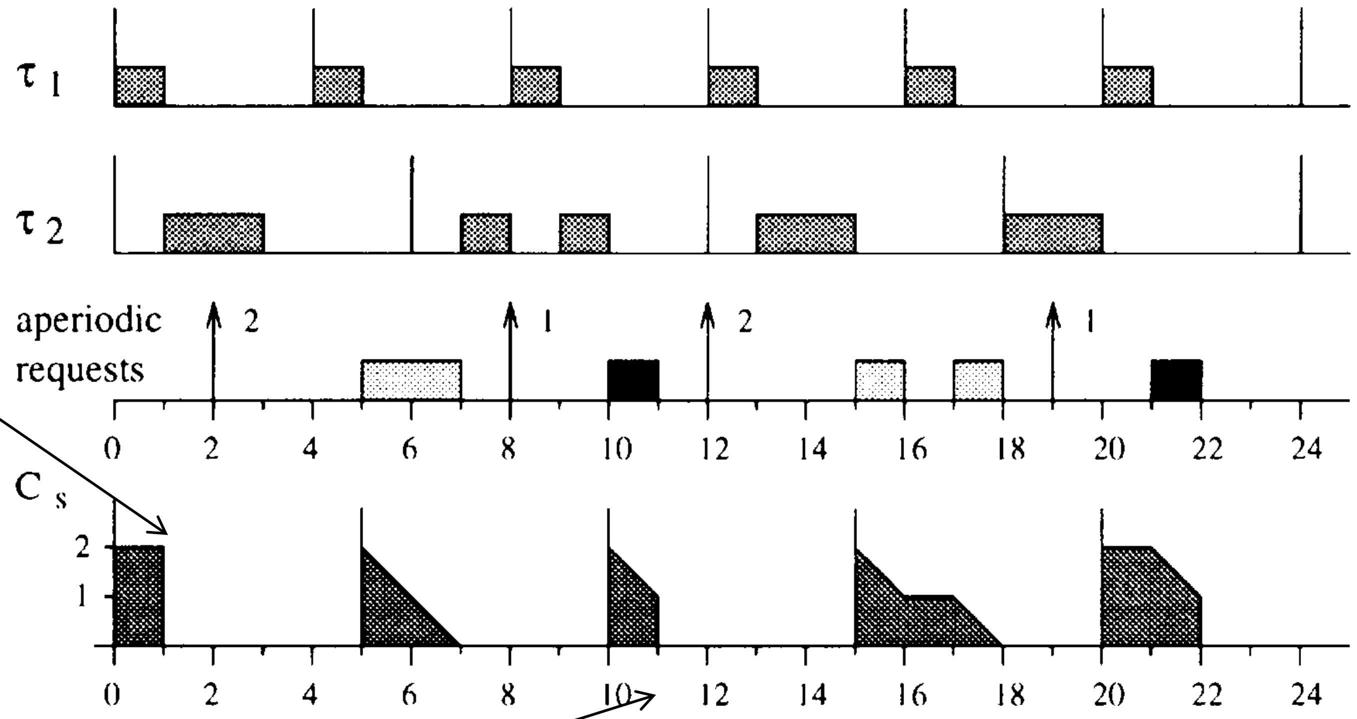
Example:

	C_i	T_i
τ_1	1	4
τ_2	2	6

Server

$C_s = 2$
$T_s = 5$

server has current highest priority and checks the queue of tasks



remaining budget is lost

Rate-Monotonic Polling Server

Schedulability analysis of periodic tasks:

- The interference by a server task is the same as the one introduced by an equivalent periodic task in rate-monotonic fixed-priority scheduling.
- A set of periodic tasks and a server task can be executed within their deadlines if

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

- Again, this test is sufficient but not necessary.

Rate-Monotonic Polling Server

Guarantee the response time of aperiodic requests:

- *Assumption:* An aperiodic task is finished before a new aperiodic request arrives.
  - Computation time C_a, deadline D_a
  - Sufficient schedulability test:

If the server task has the highest priority there is a necessary test also.

$$(1 + \left\lceil \frac{C_a}{C_s} \right\rceil) T_s \le D_a$$

The aperiodic task arrives shortly after the activation of the server task.

Maximal number of necessary server periods.

EDF – Total Bandwidth Server

Total Bandwidth Server:

- When the kth aperiodic request arrives at time $t = r_k$, it receives a deadline

$$d_k = \max(r_k, d_{k-1}) + \frac{C_k}{U_s}$$

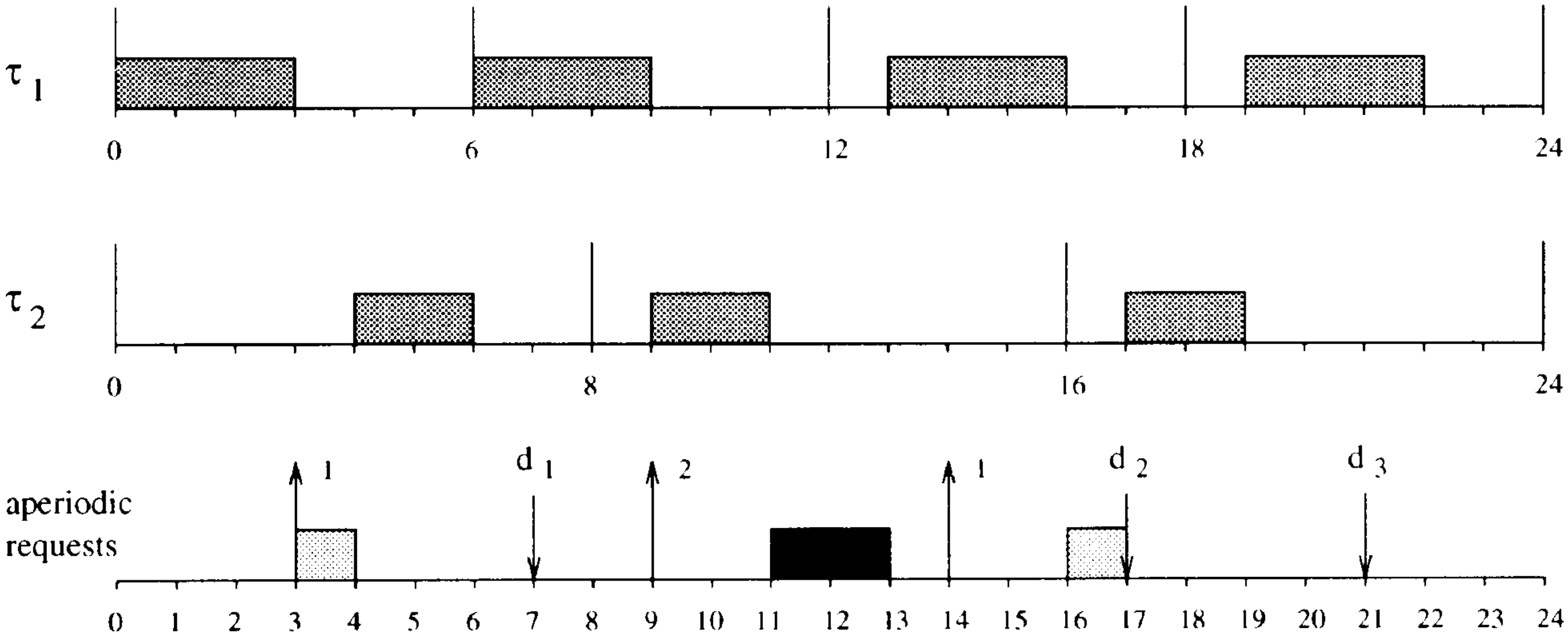
  where C_k is the execution time of the request and U_s is the server utilization factor (that is, its bandwidth). By definition, $d_0 = 0$.

- Once a deadline is assigned, the request is inserted into the ready queue of the system as any other periodic instance.

EDF – Total Bandwidth Server

Example:

$$U_p = 0.75,\ \ U_s = 0.25,\ U_p + U_s = 1$$



EDF – Total Bandwidth Server

Schedulability test:

Given a set of *n* periodic tasks with processor utilization U_p and a total bandwidth server with utilization U_s, the whole set is schedulable by EDF if and only if

$$U_p + U_s \leq 1$$

Proof:

- In each interval of time $[t_1, t_2]$, if C_{ape} is the total execution time demanded by aperiodic requests arrived at t_1 or later and served with deadlines less or equal to t_2, then

$$C_{ape} \leq (t_2 - t_1) U_s$$

EDF – Total Bandwidth Server

If this has been proven, the proof of the schedulability test follows closely that of the periodic case.

Proof of lemma:

$$C_{ape} = \sum_{k=k_1}^{k_2} C_k$$

$$= U_s \sum_{k=k_1}^{k_2} (d_k - \max(r_k, d_{k-1}))$$

$$\leq U_s \left(d_{k_2} - \max(r_{k_1}, d_{k_1 - 1})\right)$$

$$\leq U_s (t_2 - t_1)$$