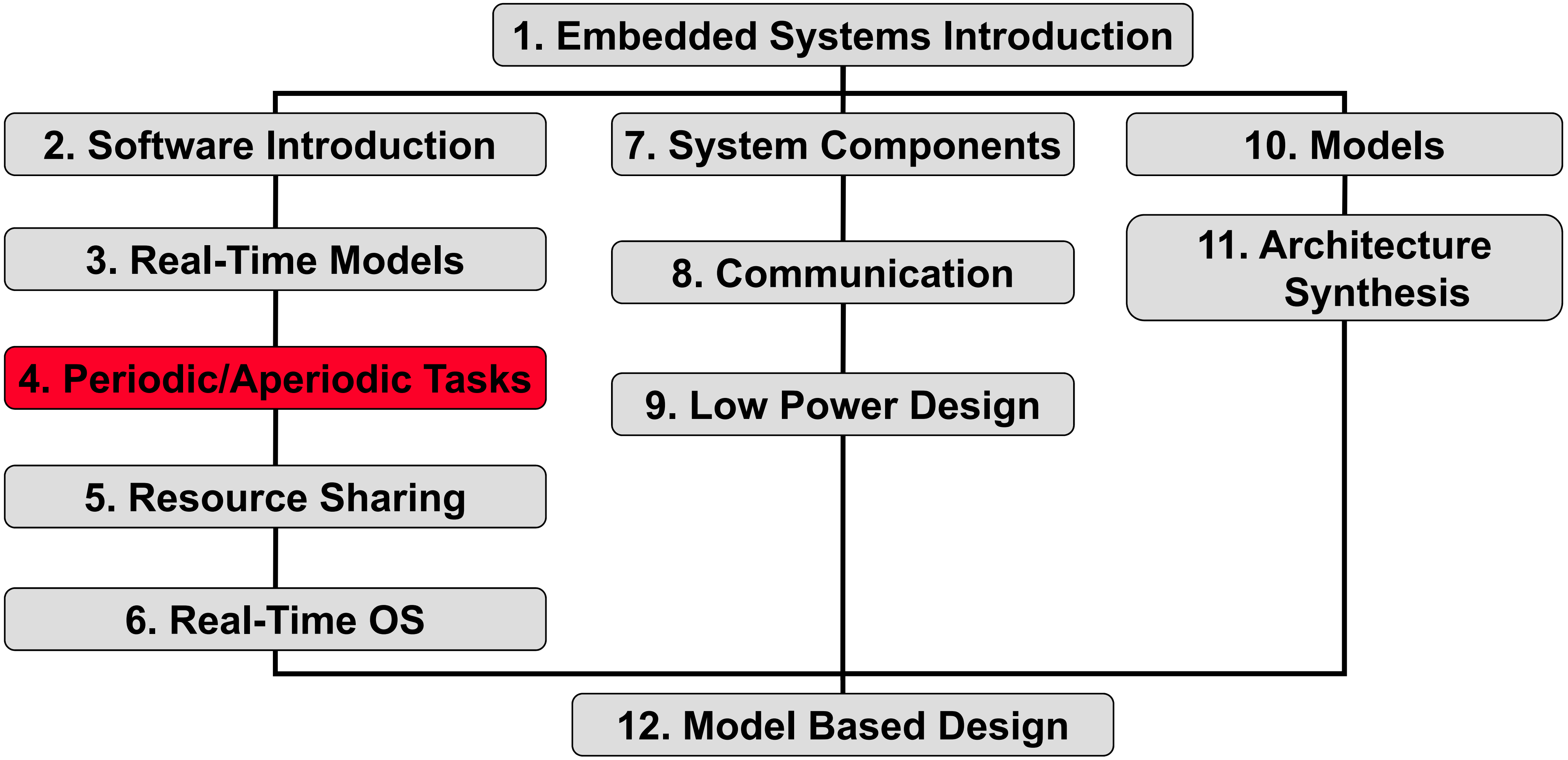
Embedded Systems

4. Aperiodic and Periodic Tasks

Lothar Thiele

Contents of Course



Software and Programming

Processing and Communication

Hardware

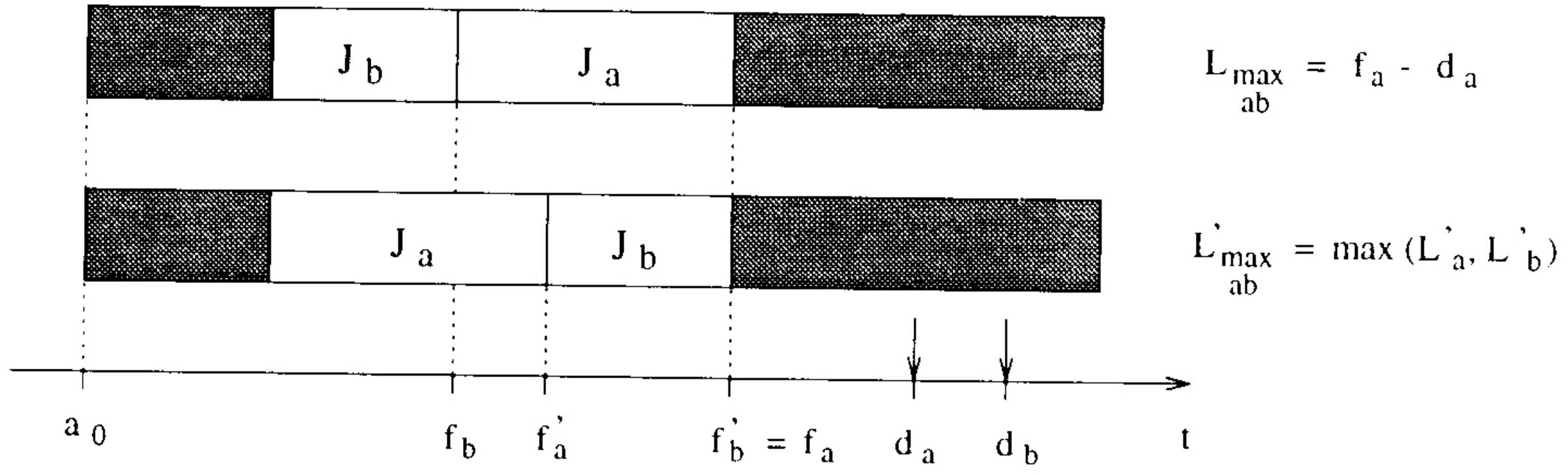
Overview

- Scheduling of ***aperiodic tasks*** with real-time constraints:
  - Table with some known algorithms:

	Equal arrival times non preemptive	Arbitrary arrival times preemptive
Independent tasks	EDD (Jackson)	EDF (Horn)
Dependent tasks	LDF (Lawler)	EDF* (Chetto)

Earliest Deadline Due (EDD)

- ***Jackson's rule***: Given a set of ***n*** tasks. Processing in order of non-decreasing deadlines is optimal with respect to minimizing the maximum lateness.
- ***Proof concept***:



J_b
J_a
J_a
J_b
$L_{\max_{ab}} = f_a - d_a$
$L'_{\max_{ab}} = \max(L'_a, L'_b)$
a_0
f_b
f'_a
$f'_b = f_a$
d_a
d_b
t

if $(L'_a \geq L'_b)$ then $L'_{\max_{ab}} = f'_a - d_a < f_a - d_a$

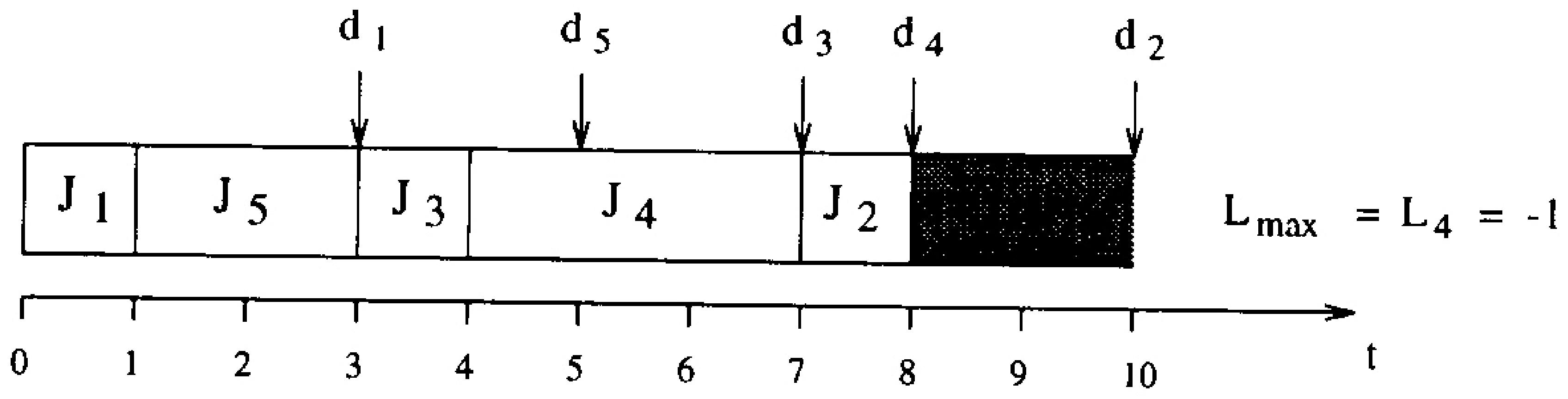
if $(L'_a \leq L'_b)$ then $L'_{\max_{ab}} = f'_b - d_b < f_a - d_a$

in both cases: $L'_{\max_{ab}} < L_{\max_{ab}}$

Earliest Deadline Due (EDD)

- ***Example 1***:

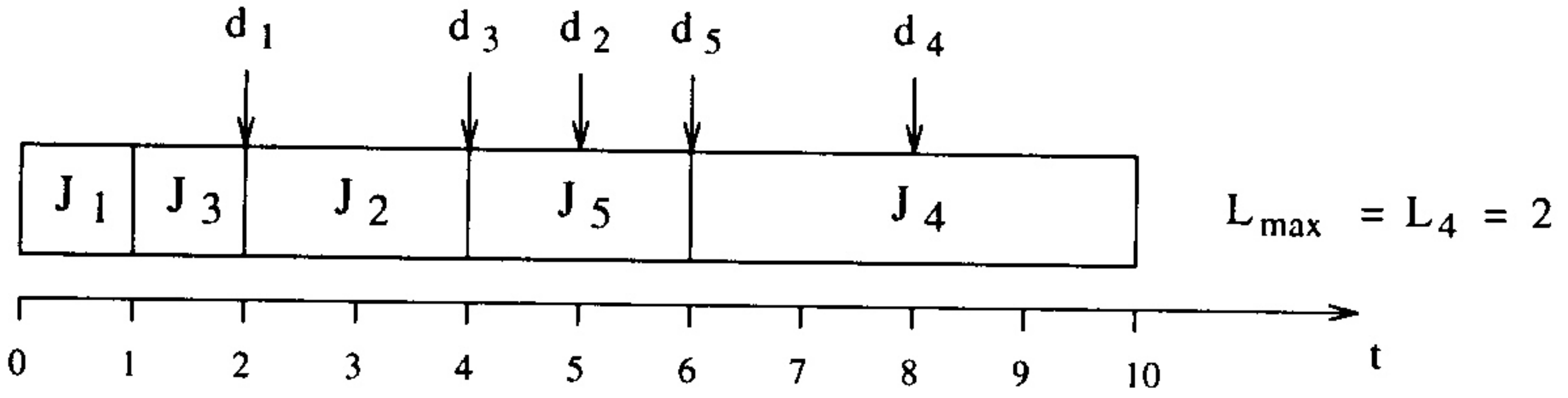
	J_1	J_2	J_3	J_4	J_5
C_i	1	1	1	3	2
d_i	3	10	7	8	5



Earliest Deadline Due (EDD)

- ***Example 2***:

	J_1	J_2	J_3	J_4	J_5
C_i	1	2	1	4	2
d_i	2	5	4	8	6



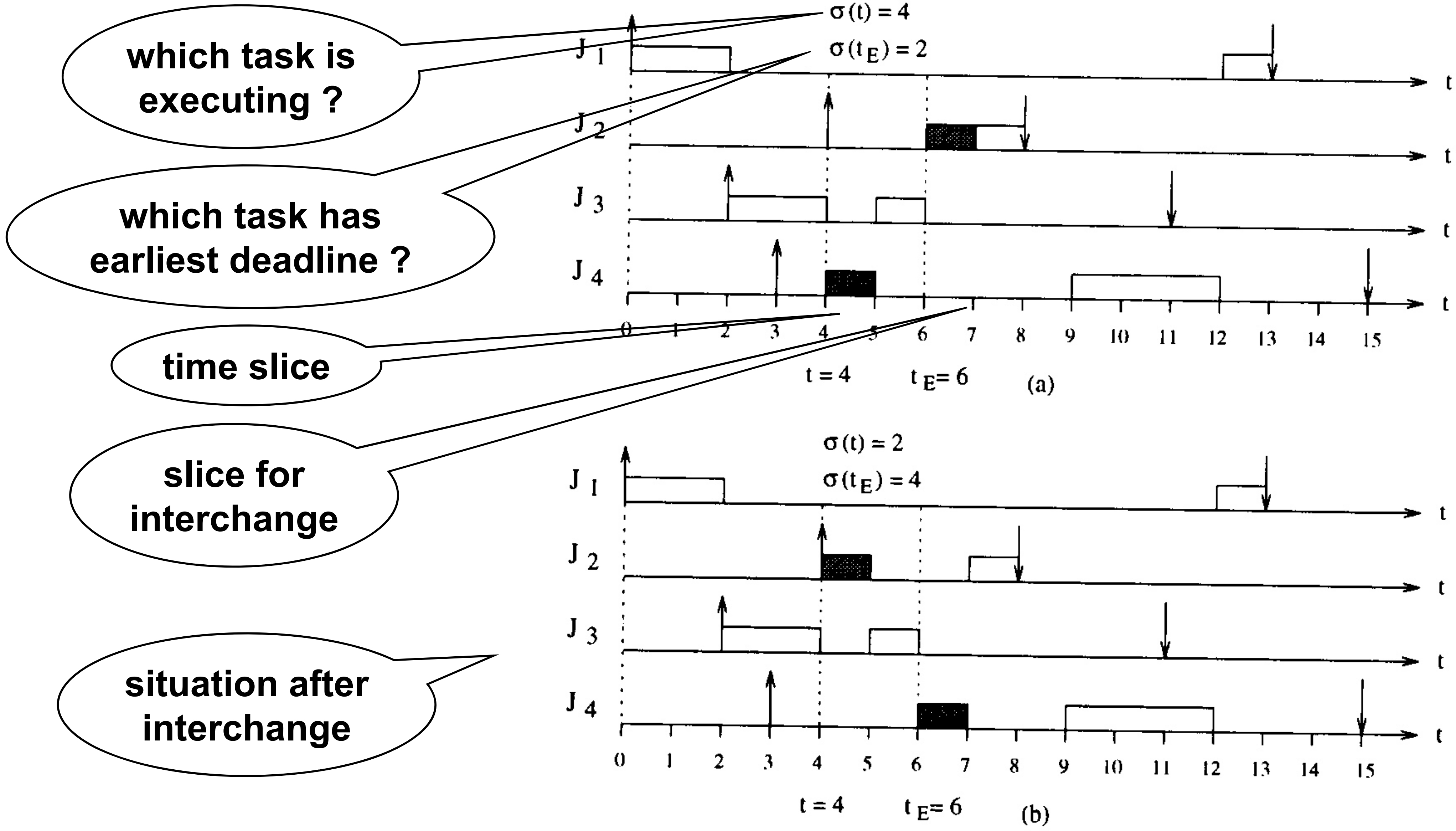
Earliest Deadline First (EDF)

- ***Horn's rule***: Given a set of n independent tasks with arbitrary arrival times, any algorithm that at any instant executes the task with the earliest absolute deadline among the ready tasks is optimal with respect to minimizing the maximum lateness.
- ***Concept of proof***: For each time interval $[t, t+1)$ it is verified, whether the actual running task is the one with the earliest absolute deadline. If this is not the case, the task with the earliest absolute deadline is executed in this interval instead. This operation cannot increase the maximum lateness.

Earliest Deadline First (EDF)

- Used quantities and terms:
  - $\sigma(t)$ identifies the task executing in the slice $[t, t+1)$
  - $E(t)$ identifies the ready task that, at time t, has the earliest deadline
  - $t_E(t)$ is the time $(\geq t)$ at which the next slice of task $E(t)$ begins its execution in the current schedule

Earliest Deadline First (EDF)



Earliest Deadline First (EDF)

remaining worst-case execution time of task k

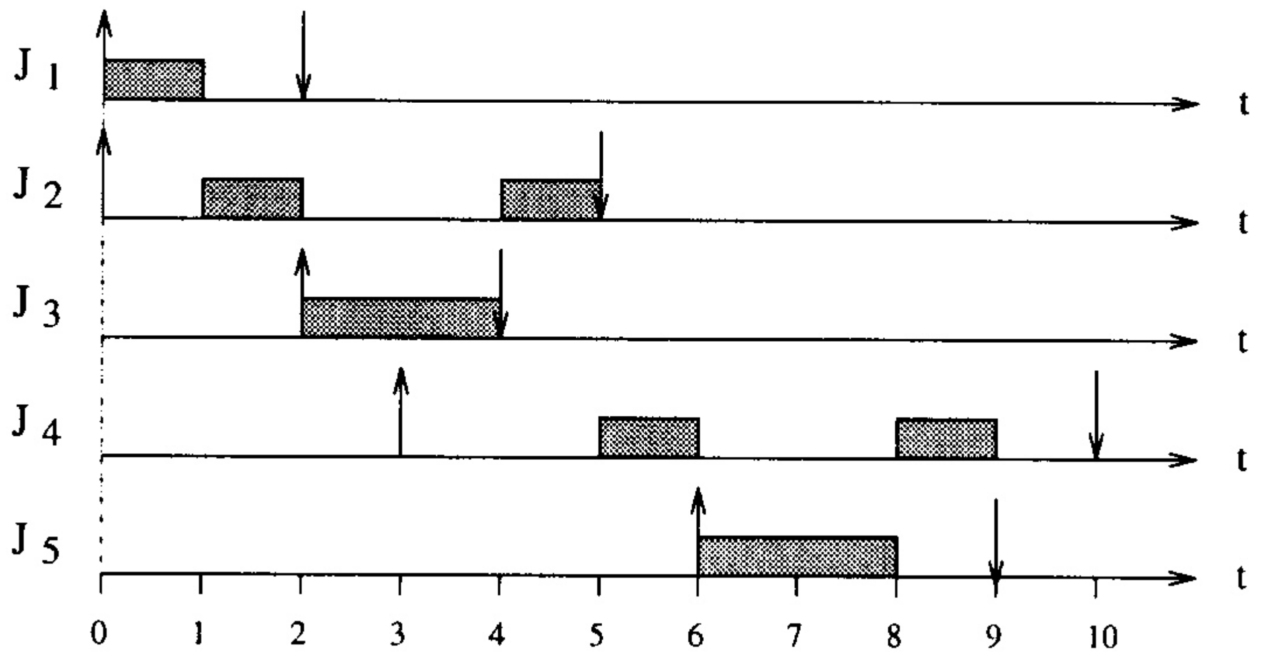
- ***Guarantee***:
  - worst case finishing time of task i: $f_i = t + \sum_{k=1}^{i} c_k(t)$
  - EDF guarantee condition: $\forall i = 1,\dots,n \quad t + \sum_{k=1}^{i} c_k(t) \le d_i$
  - algorithm:

```
Algorithm: EDF_guarantee (J, $J_{new}$)
{       J‘=J∪{$J_{new}$};  /* ordered by deadline */
        t = current_time();
        $f_0$ = t;
        for (each $J_i$∈J‘) {
                $f_i$ = $f_{i-1}$ + $c_i$(t);
                if ($f_i$ > $d_i$) return(INFEASIBLE);
        }
        return(FEASIBLE);
}
```

Earliest Deadline First (EDF)

- *Example*:

	J_1	J_2	J_3	J_4	J_5
a_i	0	0	2	3	6
C_i	1	2	2	2	2
d_i	2	5	4	10	9



Earliest Deadline First (EDF*)

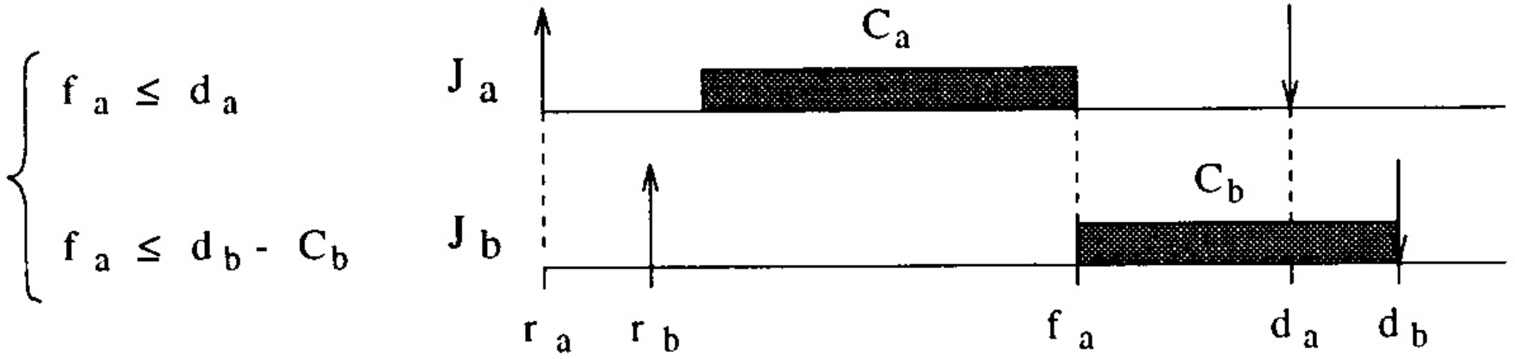
- The problem of scheduling a set of *n* ***tasks with precedence constraints*** (concurrent activation) can be solved in polynomial time complexity if tasks are preemptable.
- The ***EDF* algorithm*** determines a feasible schedule in the case of tasks with precedence constraints if there exists one.

- By the modification it is guaranteed that if there exists a valid schedule at all then
  - a task starts execution not earlier than its release time and not earlier than the finishing times of its predecessors (a task cannot preempt any predecessor)
  - all tasks finish their execution within their deadlines

Earliest Deadline First (EDF*)

- ***Modification of deadlines***:
  - Task must finish the execution time within its deadline
  - Task must not finish the execution later than the maximum start time of its successor

task b depends on task a: $J_a \rightarrow J_b$

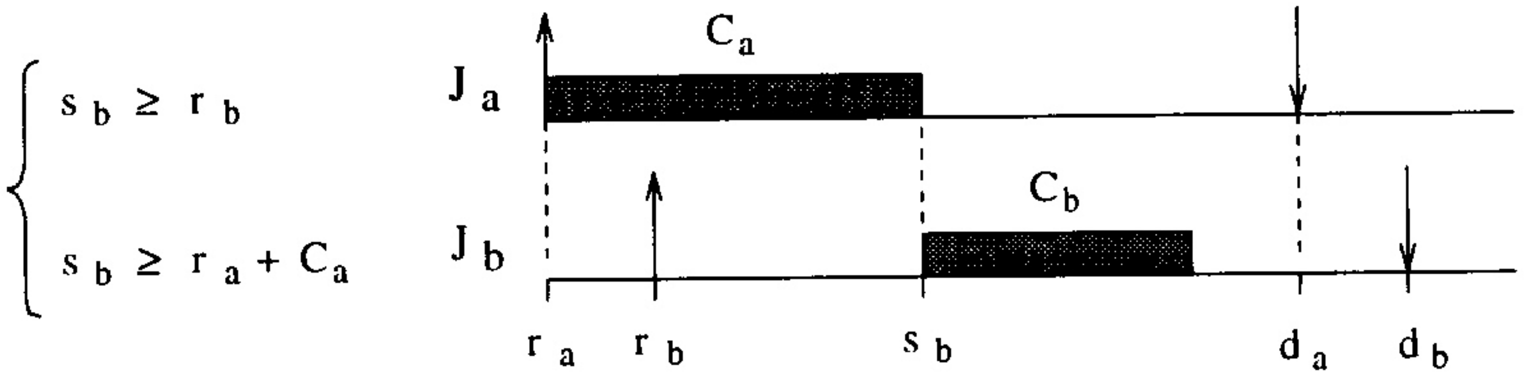


- Solution: $d_i{}^* = \min\left(d_i, \min\left(d_j{}^* - C_j : J_i \rightarrow J_j\right)\right)$

Earliest Deadline First (EDF*)

- ***Modification of release times***:
  - Task must start the execution not earlier than its release time.
  - Task must not start the execution earlier than the minimum finishing time of its predecessor.

task b depends on task a: $J_a \rightarrow J_b$



- Solution: $r_j^* = \max\left(r_j, \max\left(r_i^* + C_i : J_i \rightarrow J_j\right)\right)$

Earliest Deadline First (EDF*)

- *Algorithm* for modification of release times:
  1. For any initial node of the precedence graph set $r_i^* = r_i$
  2. Select a task ***j*** such that its release time has not been modified but the release times of all immediate predecessors ***i*** have been modified. If no such task exists, exit.
  3. Set $r_j^* = \max\left(r_j, \max\left(r_i^* + C_i : J_i \rightarrow J_j\right)\right)$
  4. Return to step 2

- *Algorithm* for modification of deadlines:
  1. For any terminal node of the precedence graph set $d_i^* = d_i$
  2. Select a task ***i*** such that its deadline has not been modified but the deadlines of all immediate successors ***j*** have been modified. If no such task exists, exit.
  3. Set $d_i^* = \min\left(d_i, \min\left(d_j^* - C_j : J_i \rightarrow J_j\right)\right)$
  4. Return to step 2

Earliest Deadline First (EDF*)

- *Proof concept*
  - Show that if there exists a feasible schedule for the modified task set under EDF then the original task set is also schedulable (ignoring precedence relations). To this end, show that the original task set meets the timing constraints also. This can be done by using $d_i^* \le d_i$, $r_i^* \ge r_i$.
  - Show the reverse also.
  - In addition, show that the precedence relations in the original task set are not violated. In particular, show that
    - a task cannot start before its predecessor and
    - a task cannot preempt its predecessor.

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 servoing, control loops, action planning and system monitoring. When a control 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 generic periodic task
  $\tau_{i,j}$: denotes the *j*th instance of task *i*
  $r_{i,j}, s_{i,j}, f_{i,j}, d_{i,j}$:
  denotes the release time, start time, finishing time, absolute deadline of the *j*th instance of task *i*
  Φ_i : phase of task *i* (release time of its first instance)
  D_i : relative deadline 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

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

Model of Periodic Tasks

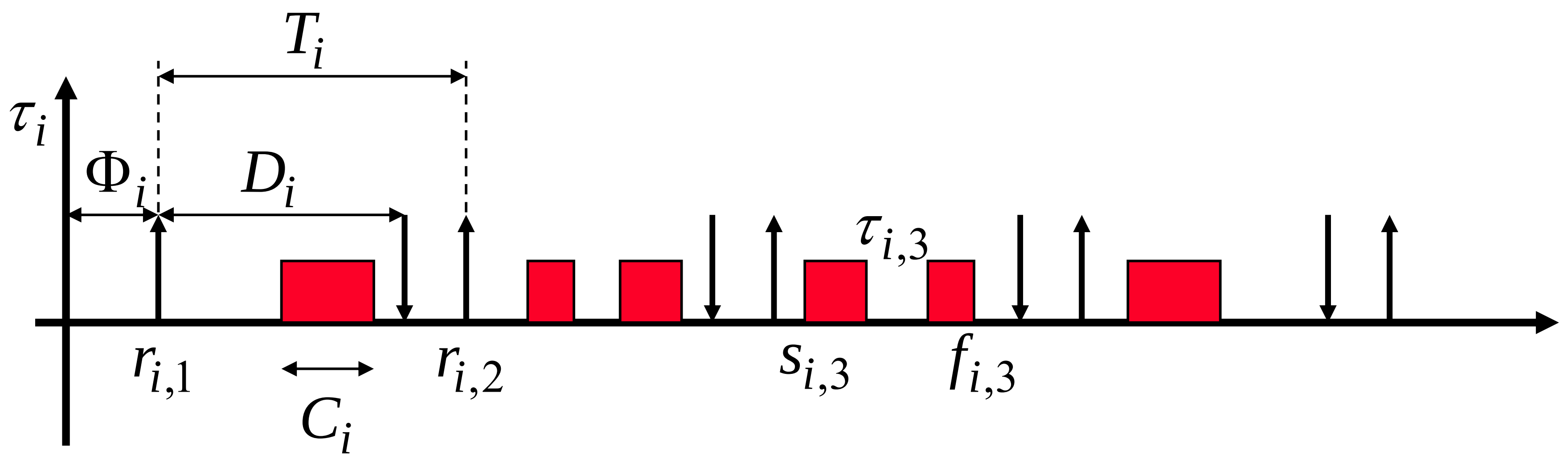
- The following ***hypotheses*** are assumed on the tasks cont':
  - Often, the relative deadline equals the period $D_i = T_i$ and therefore

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

  - 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.

Model of Periodic Tasks

- ***Example***:



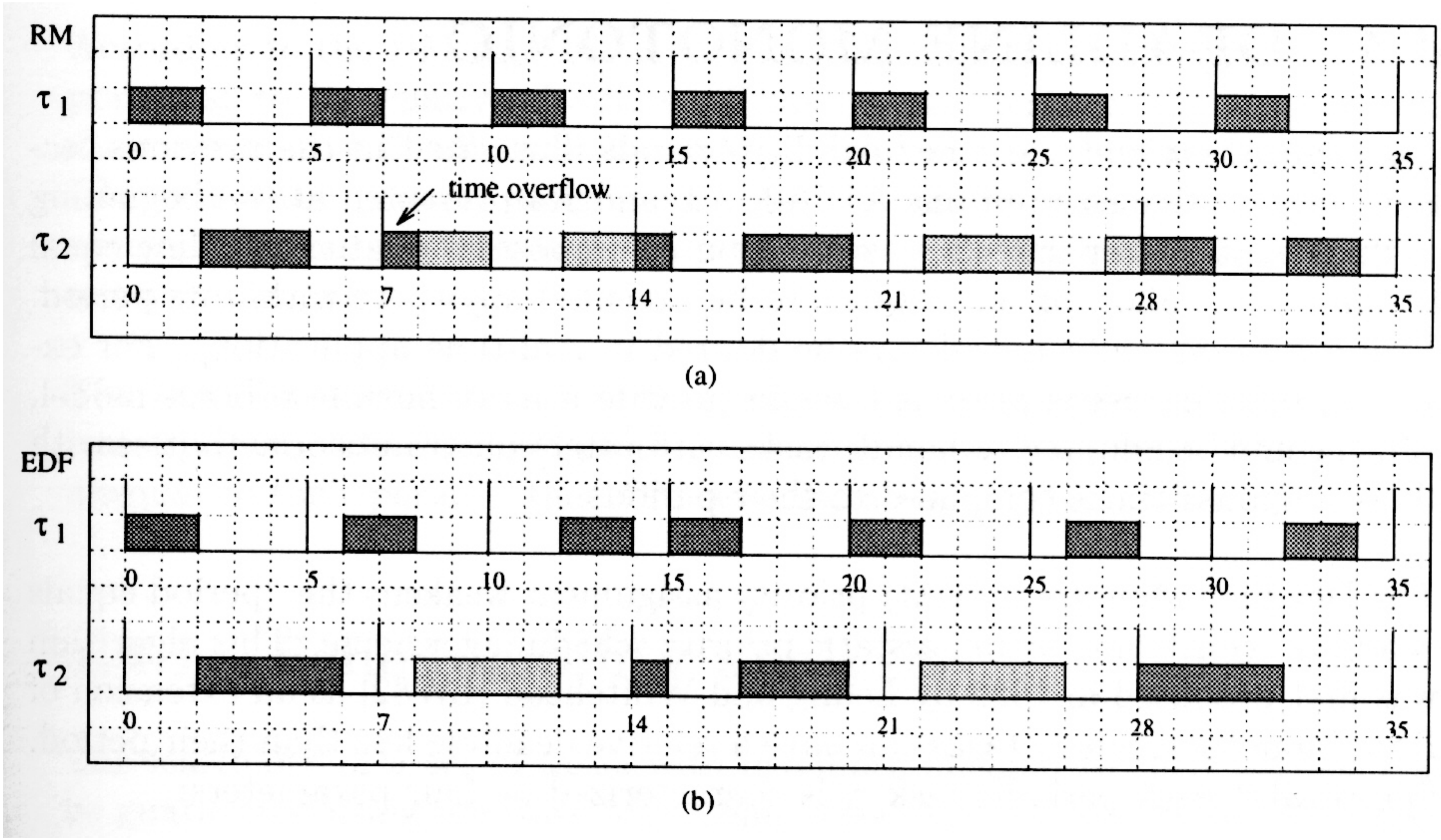
Rate Monotonic Scheduling (RM)

- ***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 task is preempted by a task with higher priority.
  - ***Deadlines*** equal the periods $D_i = T_i$.

- ***Algorithm***: Each task is assigned a priority. Tasks with higher request rates (that is with shorter periods) will have higher priorities. Tasks with higher priority interrupt tasks with lower priority.

Periodic Tasks

- ***Example***: 2 tasks, deadline = periods, ***U* = 97%**



Rate Monotonic Scheduling (RM)

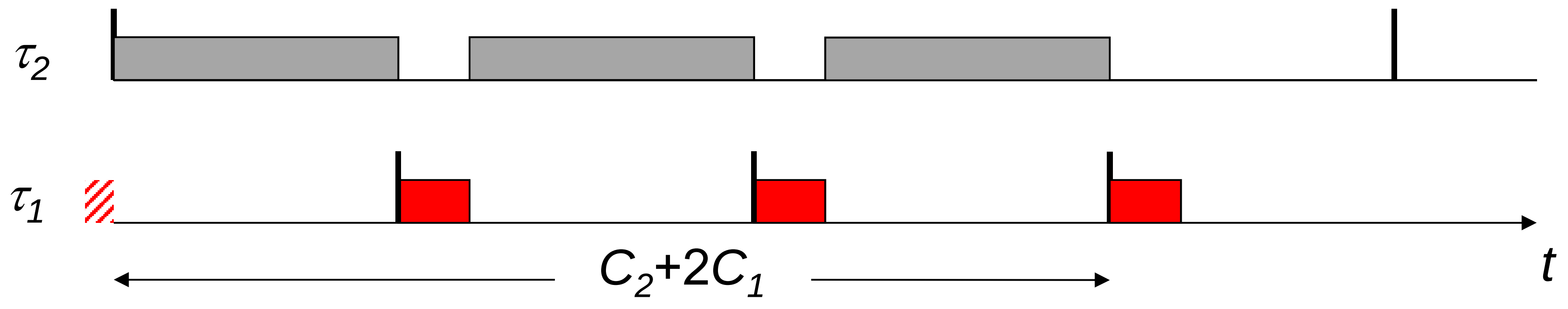
- ***Optimality***: RM is optimal among all fixed-priority assignments in the sense that not other fixed-priority algorithm can schedule a task set that cannot be scheduled by RM.
- The ***proof*** is done by considering several cases that may occur, but the main ideas are as follows:
  - ***A critical 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 instances. If all tasks are feasible at their critical instants, 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.

Proof of Critical Instance

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

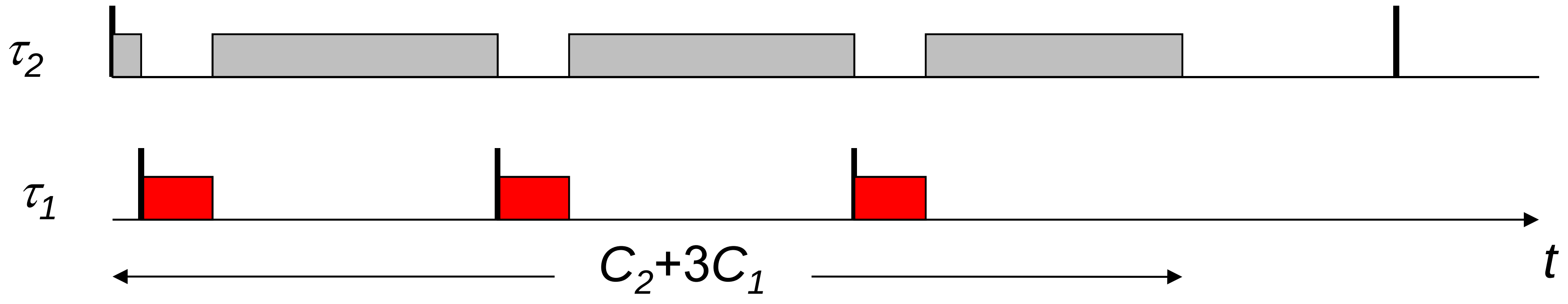
Lemma: For any task, the **critical instant** occurs if that task is simultaneously released with all higher priority tasks.

Proof sketch: Start with 2 tasks τ_1 and τ_2 .
Response time of τ_2 is delayed by tasks 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 task occurs when it is released simultaneously with all higher-priority tasks.

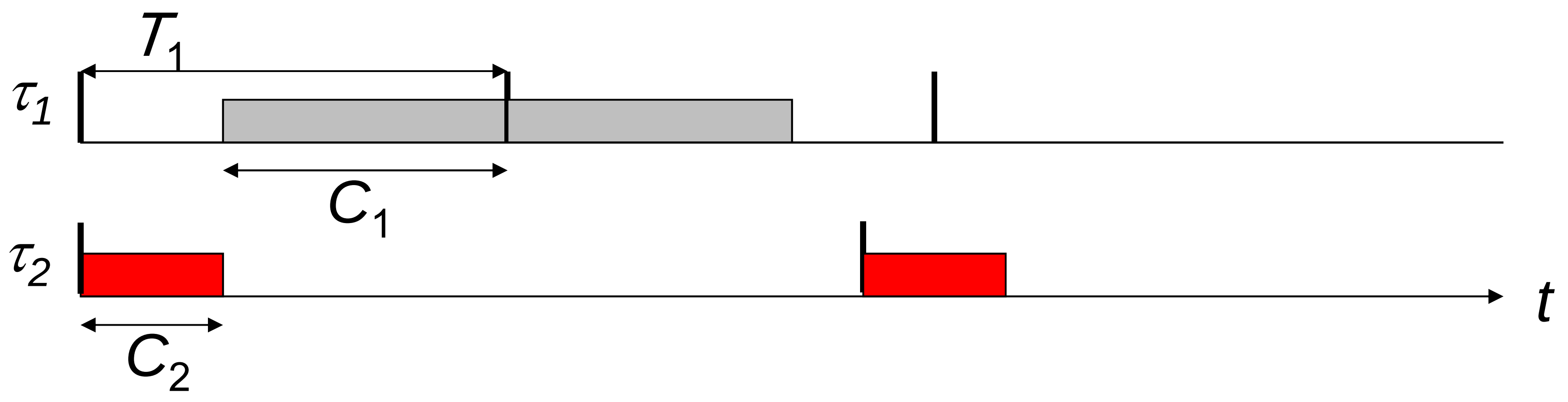
Proof of RM Optimality (2 Tasks)

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

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

Consider two cases A and B:

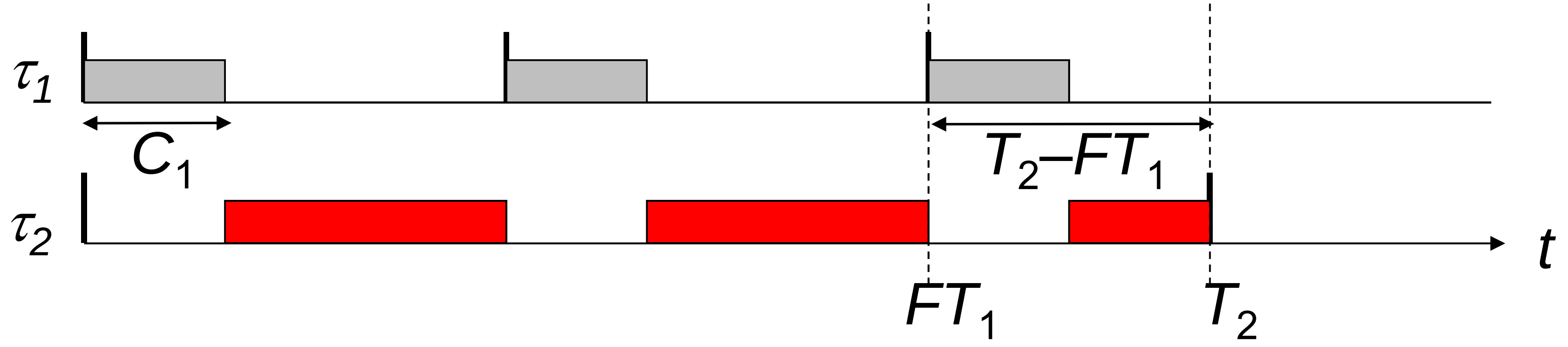
A: Assume RM is **not** used → prio(τ_2) is highest:



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

Proof of RM Optimality (2 Tasks)

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



Schedulable 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) ⇒ (B): $C_1 + C_2 \le T_1 \Rightarrow C_1 \le T_1$

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

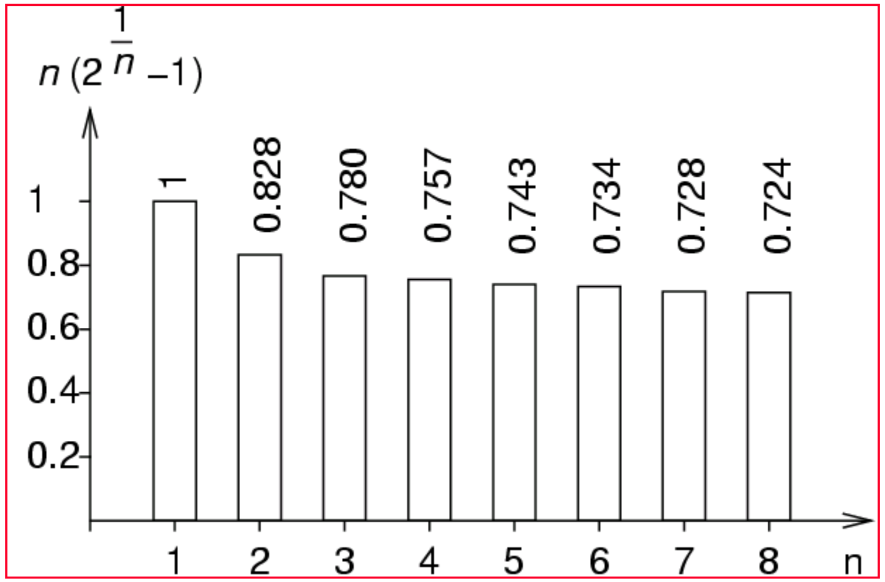
$$FC_1 + C_2 + \min(T_2 – FT_1, C_1) \le FT_1 + \min(T_2 – FT_1, C_1) \le \min(T_2, C_1 + FT_1) \le \mathrm{T}_2$$

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.

Rate Monotonic Scheduling (RM)

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

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



  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)

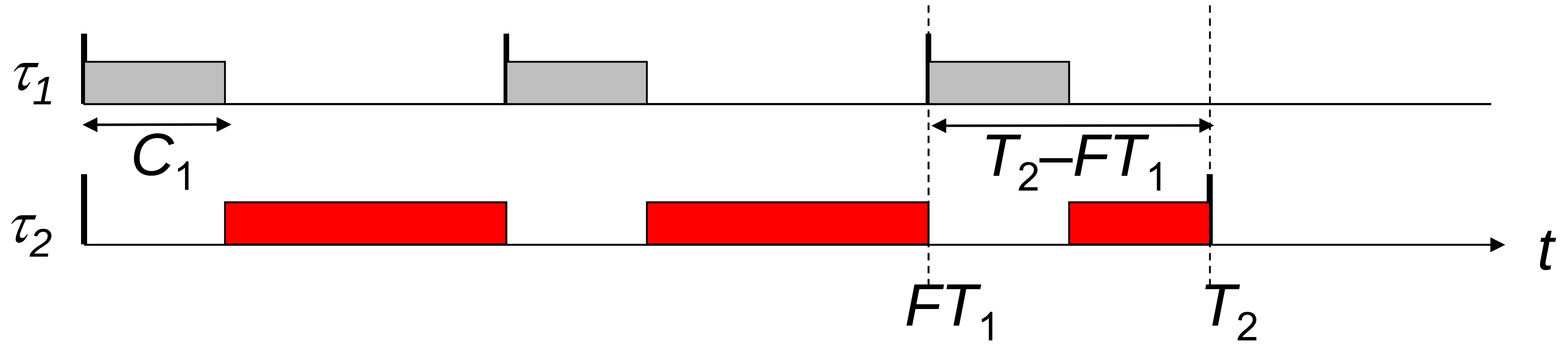
We have two tasks τ_1, τ_2 with periods $T_1 < T_2$.

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

- Proof procedure: 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 setting computation times to fully utilize processor (C_2 adjusted to fully utilize processor);
  - minimize upper bound with respect to other task parameters.

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)

- 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 $G = T_2/T_1$. We obtain

Proof of Utilization Bound (2 Tasks)

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

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 periodic, 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. Tasks with higher priority interrupt tasks 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)

- 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 (preemption) 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 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, $R_i \le D_i$ 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 ($\Gamma$)
{       for (each $\tau_i \in \Gamma$){
                I = 0;
                do {
                        R = I + $C_i$;
                        if (R > $D_i$) return(UNSCHEDULABLE);
                        I = $\sum^{j=1,\dots,(i-1)} \lceil R/T_j \rceil C_j$;
                } while (I + $C_i$ > R);
        }
        return(SCHEDULABLE);
}
```

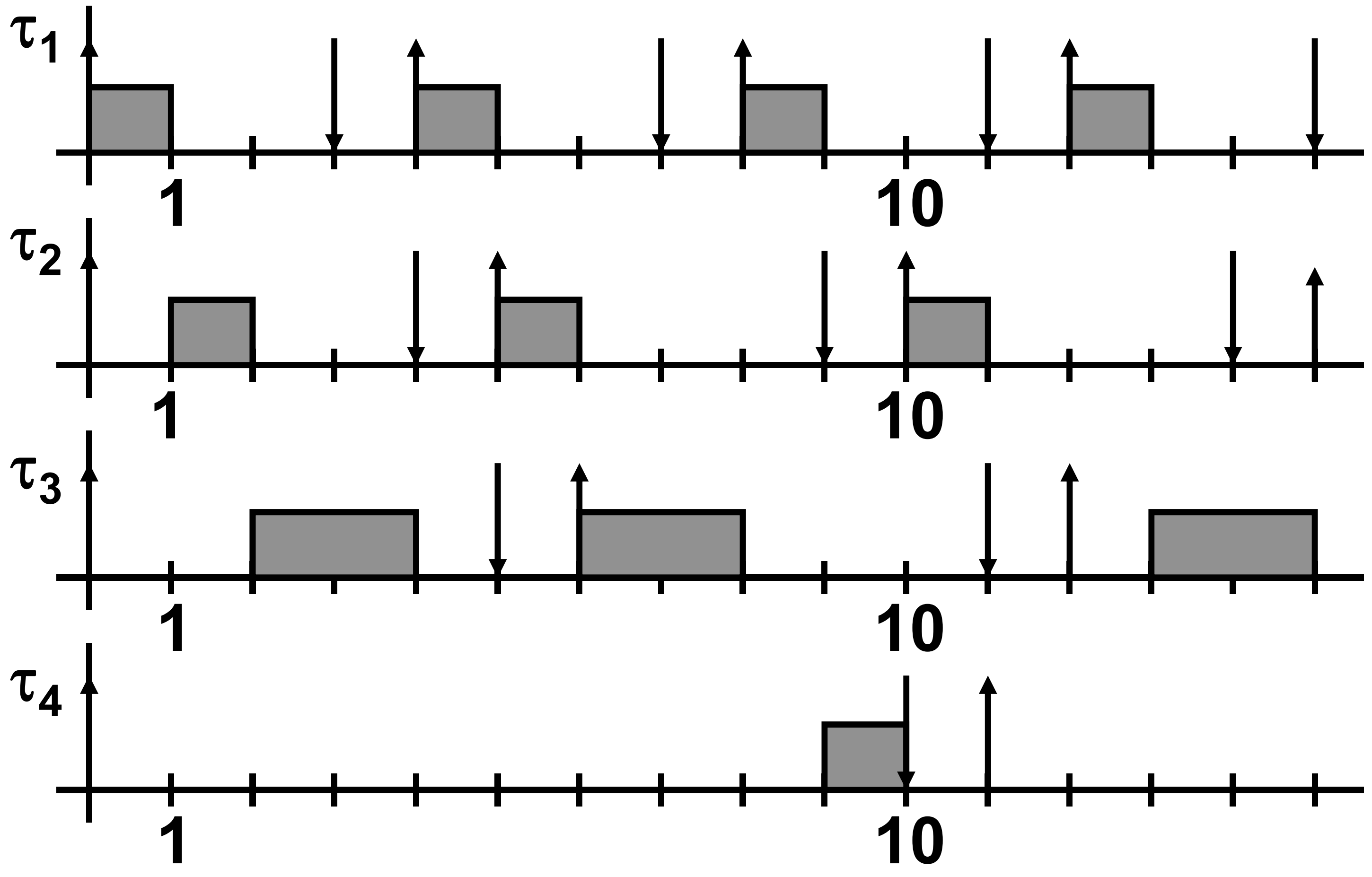
DM Example

- ***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 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: $R_4 = 10$

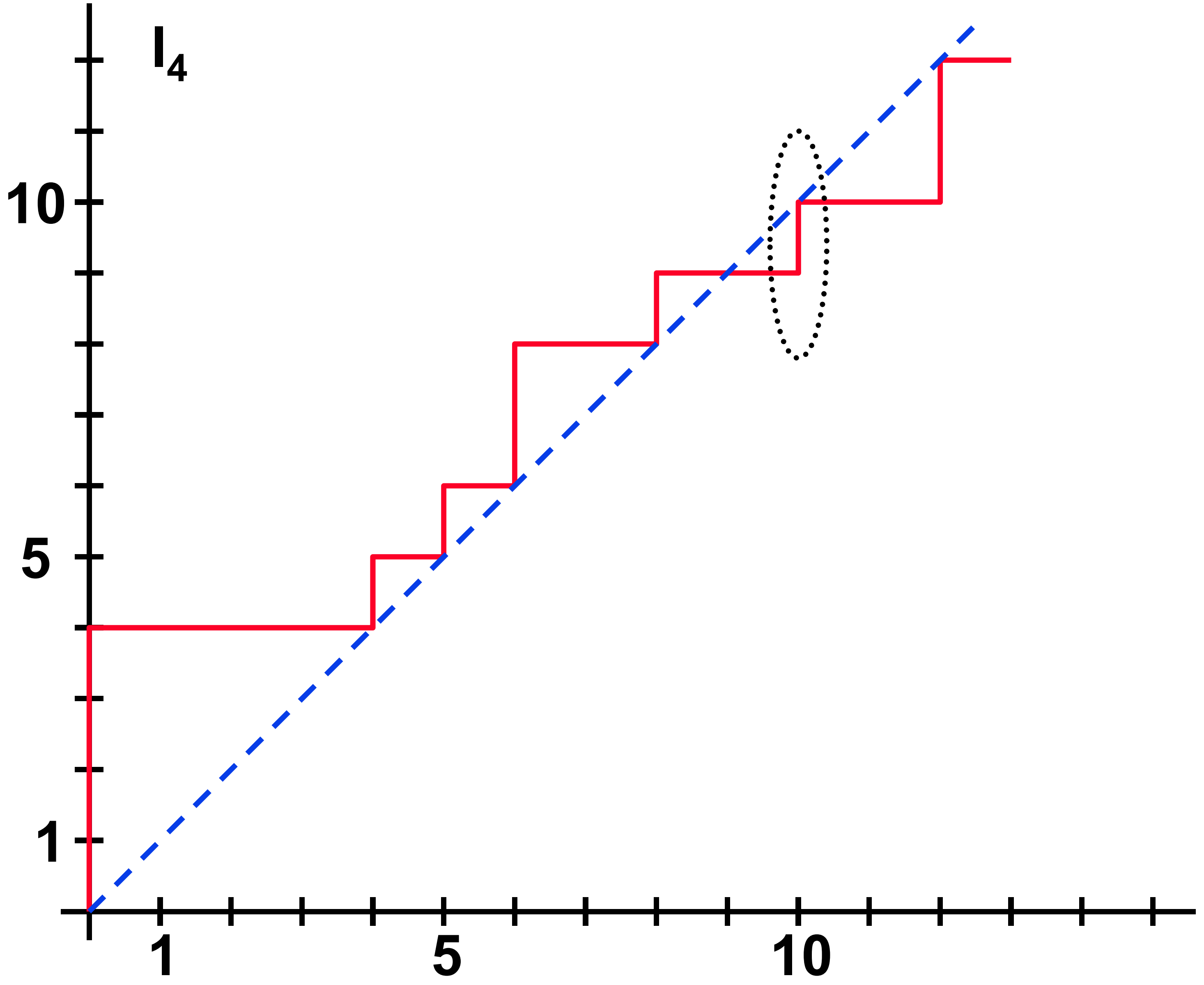
DM Example

$U = 0.874$

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



DM Example



EDF Scheduling (earliest deadline first)

- ***Assumptions***:
  - dynamic priority assignment
  - intrinsically preemptive
  - $D_i \le T_i$
- ***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.

EDF Scheduling

- A necessary and sufficient ***schedulability test*** if $D_i = T_i$:
  - 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.

- If the utilization satisfies $U \leq 1$, then there is a valid schedule.
  - We will proof this 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 the maximal time before t_2 where
  - the processor is continuously busy in $[t_1, t_2]$ and
  - the processor only executes tasks that have their arrival time AND 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 when the processor gets idle, then we found t_1: There is a task arrival at t_1 and the task queue is empty shortly before.
    - Suppose that we reach a time such that shortly before the processor works on a task with deadline after t_2, then we also found t_1: Because of EDF, all tasks the processor processed in $[t_1, t_2]$ arrived at or after t_1 (otherwise, the processor would not have operated before t_1 on a task with deadline after t_2).

EDF Scheduling

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

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

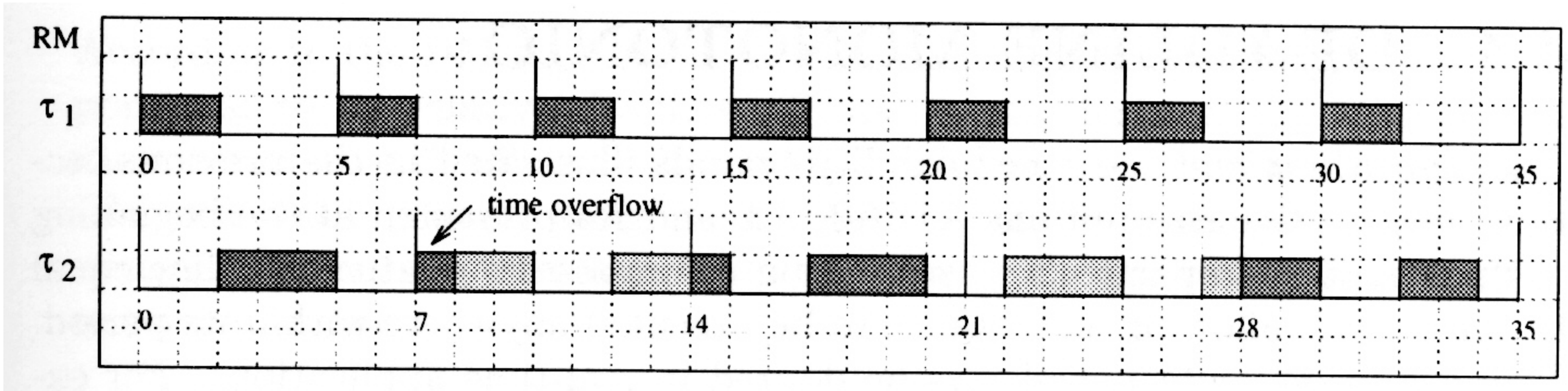
number of complete periods of task I in the interval

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

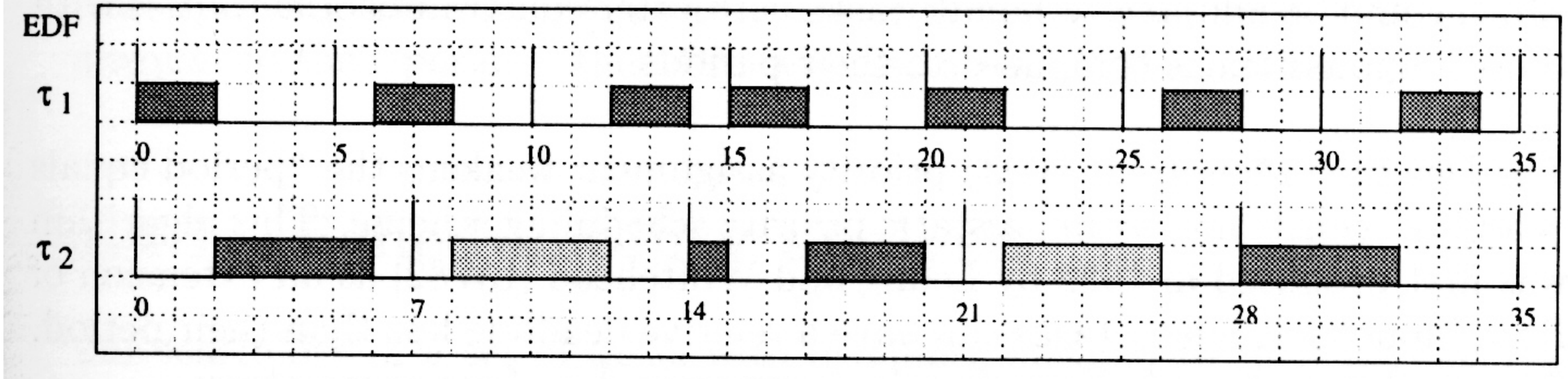
$$t_2 - t_1 < C_p(t_1,t_2) \le (t_2 - t_1)U \;\Rightarrow\; U > 1$$

Periodic Tasks

- *Example*: 2 tasks, deadline = periods, ***U* = 97%**



(a)



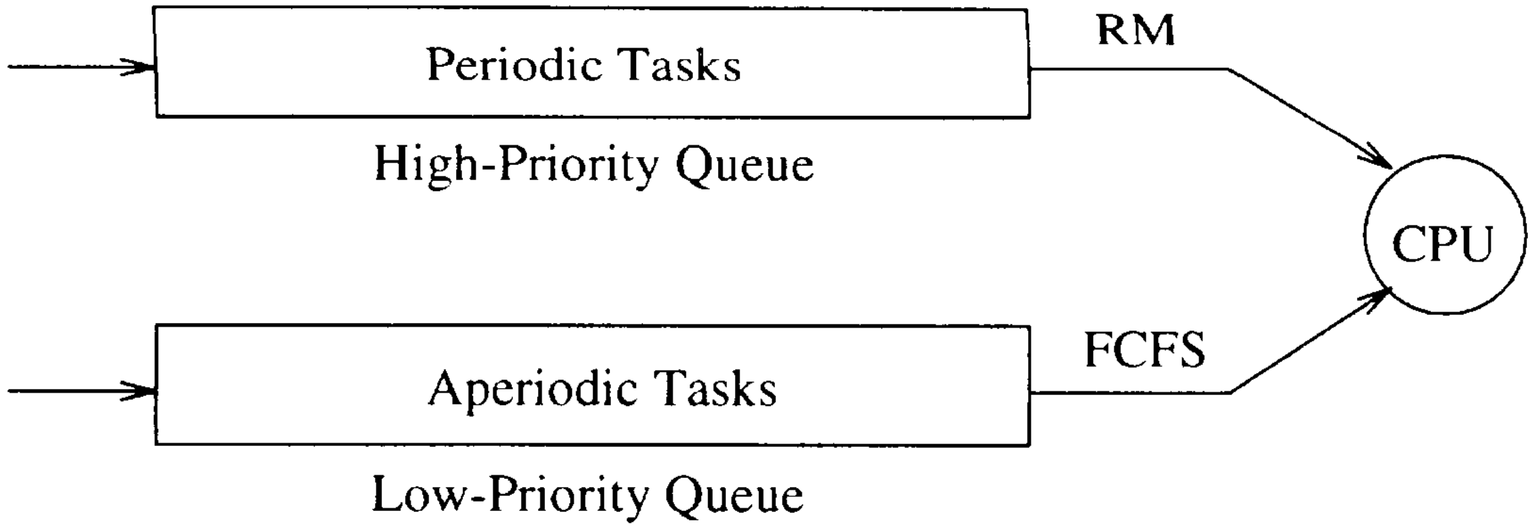
(b)

Problem of Mixed Task Sets

- In many applications, there are as well aperiodic 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***.

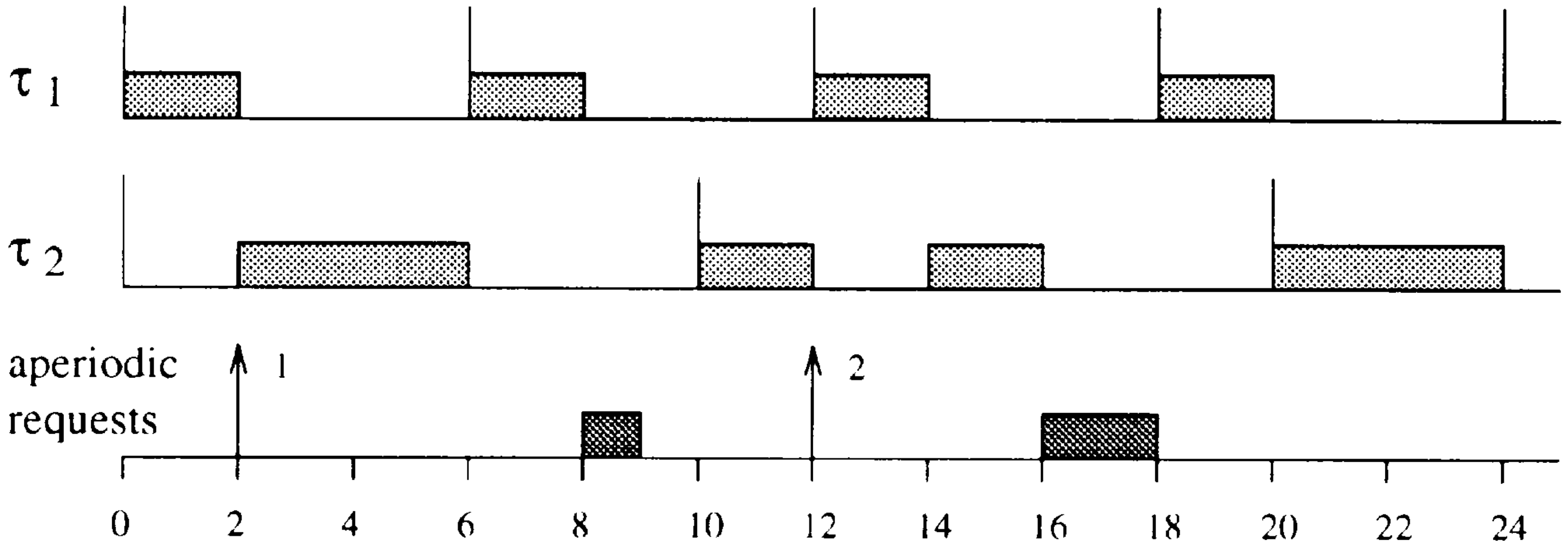
Background Scheduling

- ***Simple solution*** for RM and EDF scheduling of periodic tasks:
  - Processing of aperiodic tasks in the background, i.e. if there are no periodic request.
  - 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.



Background Scheduling

- *Example* (rate monotonic periodic schedule):



RM - Polling Server

- ***Idea***: Introduce an ***artificial periodic task*** whose purpose is to service aperiodic requests as soon as possible (therefore, “server”).
  - Like any periodic task, a server is characterized by a period T_s and a computation time C_s.
  - The server is scheduled with the same algorithm used for the periodic tasks and, once active, it serves the aperiodic requests within the limit of its server capacity.
  - Its priority (period!) can be chosen to match the response time requirement for the aperiodic tasks.

RM - Polling 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***.
- ***Disadvantage***: If an aperiodic requests arrives just after the server has suspended, it must wait until the beginning of the next polling period.

RM - Polling Server

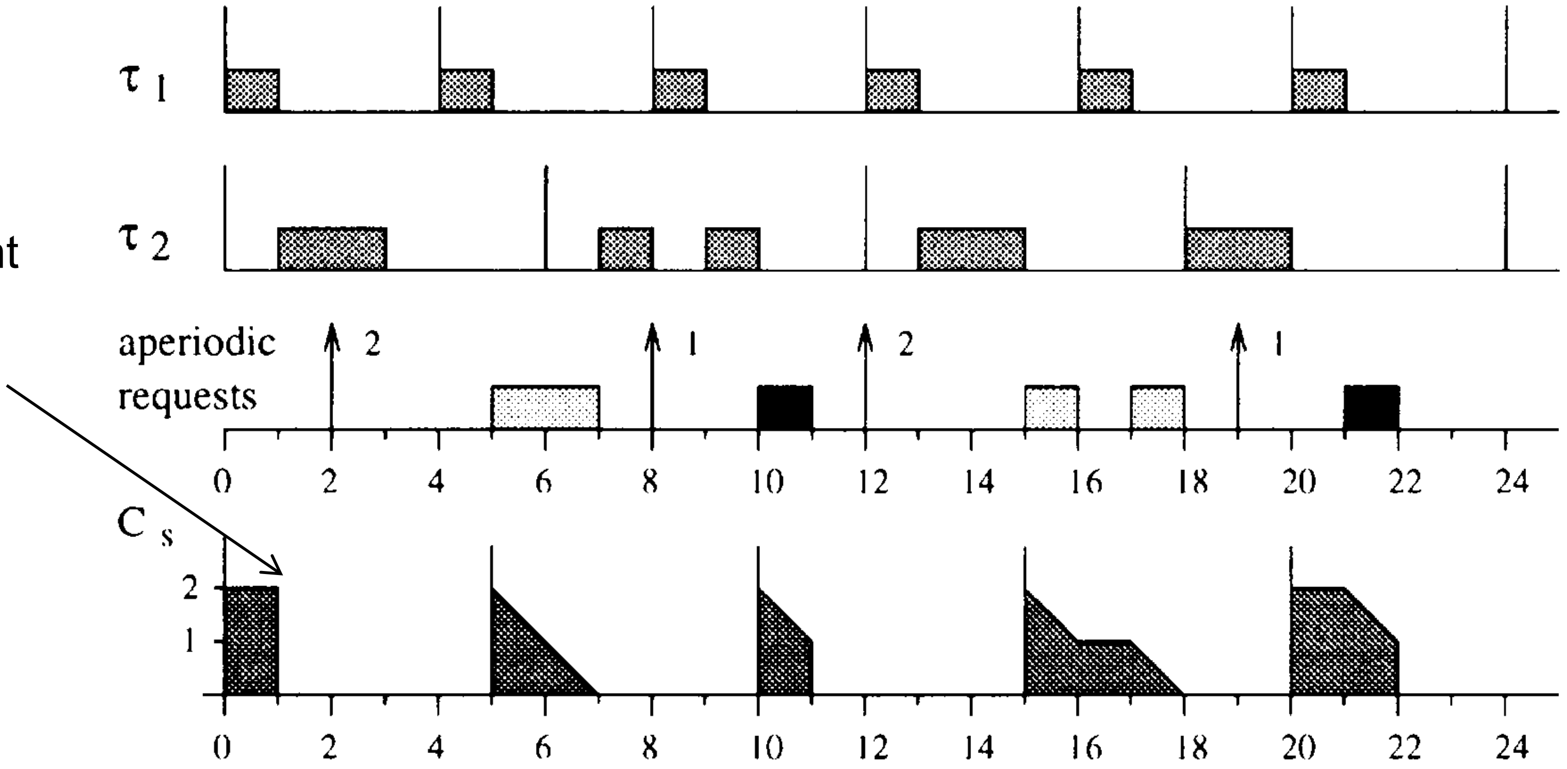
- *Example*

	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



RM - Polling Server

- ***Schedulability analysis*** of periodic tasks
  - As in the case of RM as the interference by a server task is the same as the one introduced by an equivalent periodic task.
  - 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.

RM - Polling Server

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

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

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

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

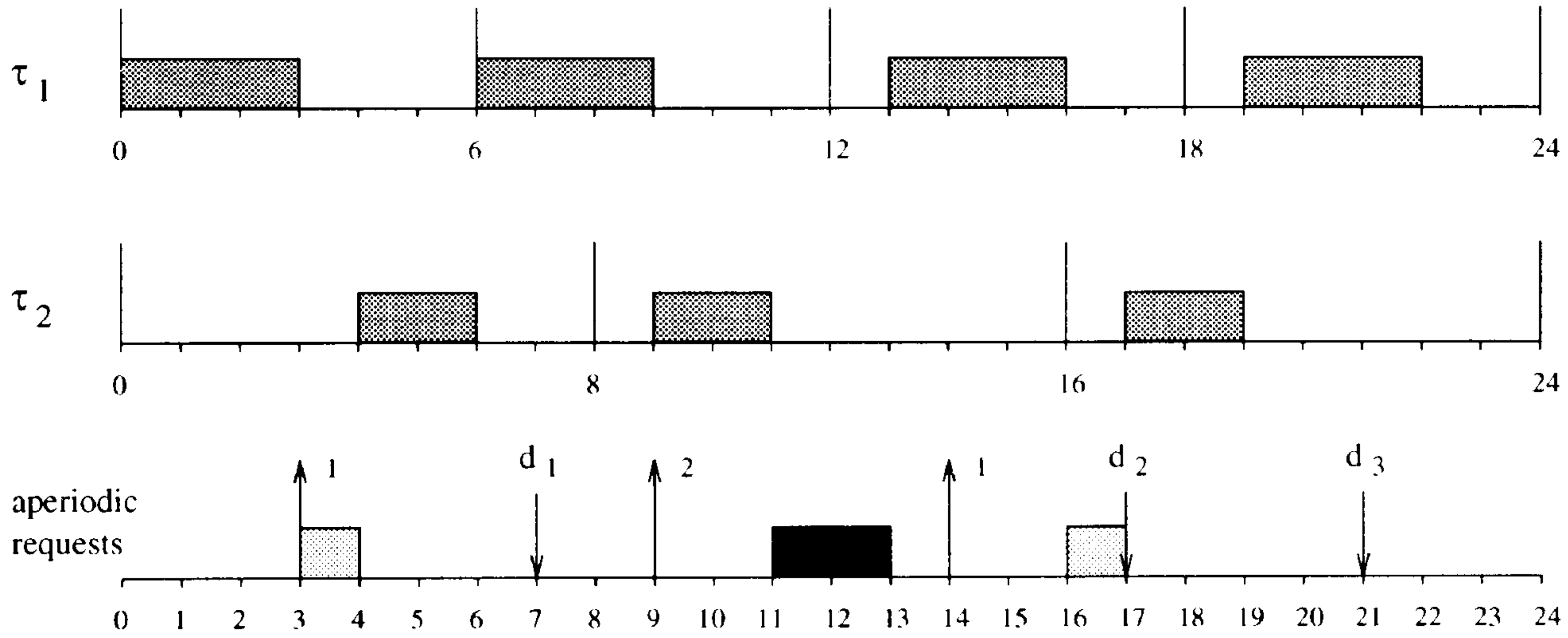
Maximal number of necessary server periods.

EDF – Total Bandwidth Server

- ***Total Bandwidth Server***:
  - When the *k*th 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

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

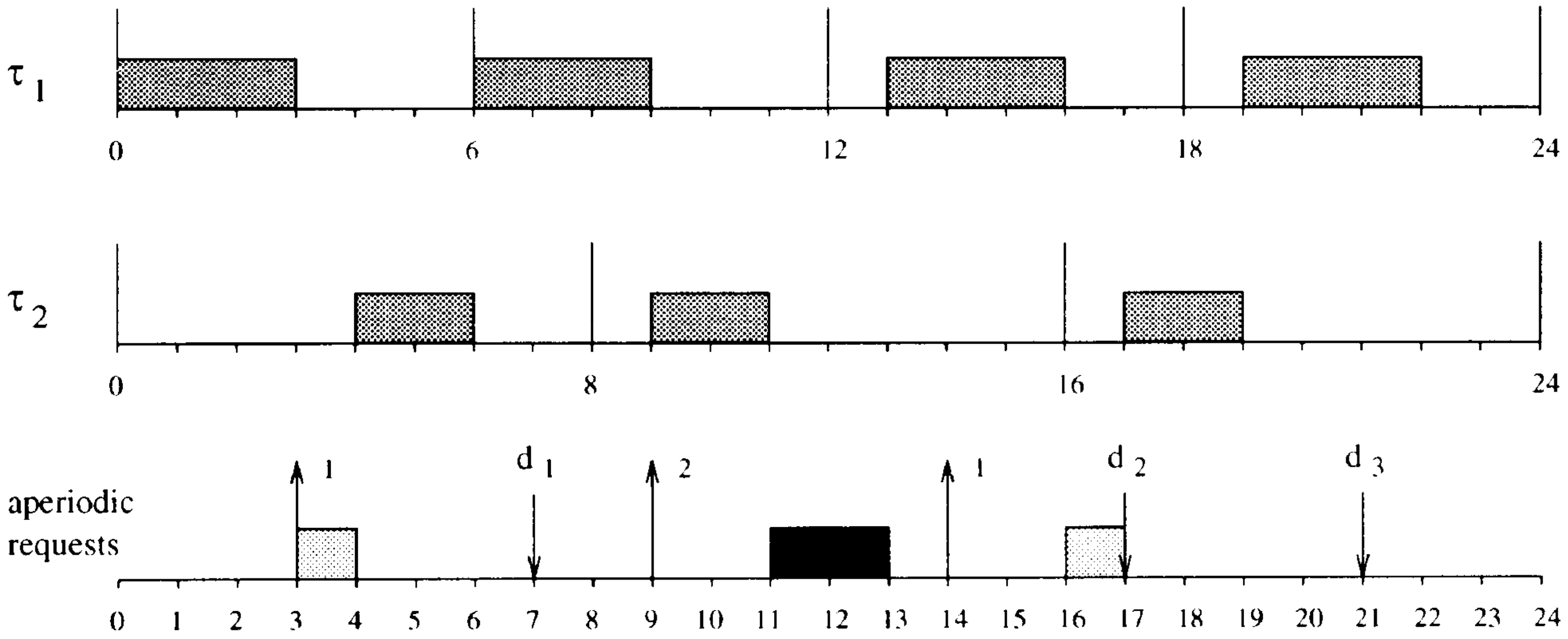
EDF – Total Bandwidth Server

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

$$
\begin{aligned}
C_{ape} &= \sum_{k=k_1}^{k_2} C_k \\
&= U_s \sum_{k=k_1}^{k_2} (d_k - \max(r_k, d_{k-1})) \\
&\le U_s \left(d_{k_2} - \max(r_{k_1}, d_{k_1-1})\right) \\
&\le U_s (t_2 - t_1)
\end{aligned}
$$

EDF – Total Bandwidth Server

- ***Example***: $U_p = 0.75, \;\; U_s = 0.25, \;\; U_p + U_s = 1$


τ_1
τ_2
aperiodic requests
d_1
d_2
d_3