Sharif University of Technology
Department of Computer Science and Engineering

Lec. 6:
Dynamic Priority Servers

Real-Time Computing

S. Safari

2023

Dynamic Priority Servers

Introduction

- Solving the problem of scheduling soft aperiodic tasks and hard periodic tasks under dynamic priority assignments.
- Different service methods are introduced, the objective of which is to reduce the average response time of aperiodic requests without compromising the schedulability of hard periodic tasks.
- Periodic tasks are scheduled by the Earliest Deadline First (EDF) algorithm.
- With respect to fixed-priority assignments, dynamic scheduling algorithms are characterized by higher schedulability bounds, which allow the processor to be better utilized, increase the size of aperiodic servers, and enhance aperiodic responsiveness.

Assumptions

- All periodic tasks $\tau_i : i = 1, ..., n$ have hard deadlines and their schedulability must be guaranteed offline.
- All aperiodic tasks $J_i : i = 1, ..., m$ do not have deadlines and must be scheduled as soon as possible, but without jeopardizing the schedulability of the periodic tasks.
- Each periodic task τ_i has a period T_i, a computation time C_i, and a relative deadline D_i equal to its period.
- All periodic tasks are simultaneously activated at time $t=0$.
- Each aperiodic task has a known computation time but an unknown arrival time.

Dynamic Priority Exchange Server

- The *Dynamic Priority Exchange* (DPE) server is an aperiodic service technique proposed by Spuri and Buttazzo [SB94, SB96] that can be viewed as an extension of the Priority Exchange server [LSS87], adapted to work with a deadline-based scheduling algorithm.
- The main idea of the algorithm is to let the server trade its runtime with the runtime of lower-priority periodic tasks (under EDF this means a longer deadline) in case there are no aperiodic requests pending.
- The server runtime is only exchanged with periodic tasks but never wasted (unless there are idle times).
- It is simply preserved, even if at a lower priority, and it can be later reclaimed when aperiodic requests enter the system.

Dynamic Priority Exchange Server (2)

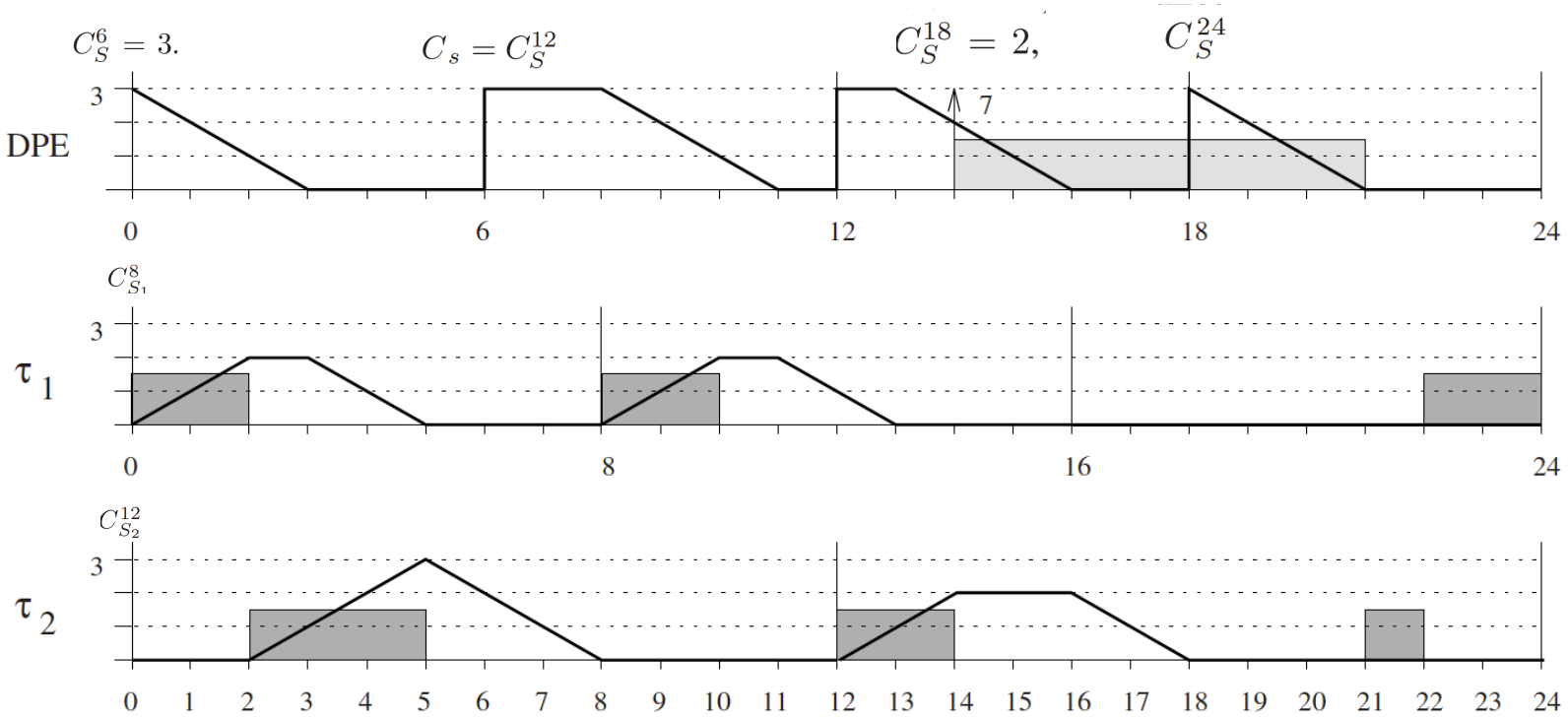
- The algorithm is defined as follows:
  - The DPE server has a specified period T_s and a capacity C_s.
  - At the beginning of each period, the server's aperiodic capacity is set to C_S^d, where d is the deadline of the current server period.
  - Each deadline d associated to the instances (completed or not) of the ith periodic task has an aperiodic capacity, $C_{S_i}^d$, initially set to 0.
  - Aperiodic capacities (those greater than 0) receive priorities according to their deadlines and the EDF algorithm, like all the periodic task instances (ties are broken in favor of capacities; that is, aperiodic requests).

Dynamic Priority Exchange Server (3)

- Whenever the highest-priority entity in the system is an aperiodic capacity of C units of time the following happens:
  - – if there are aperiodic requests in the system, these are served until they complete or the capacity is exhausted (each request consumes a capacity equal to its execution time);
  - – if there are no aperiodic requests pending, the periodic task having the shortest deadline is executed; a capacity equal to the length of the execution is added to the aperiodic capacity of the task deadline and is subtracted from C (that is, the deadlines of the highest-priority capacity and the periodic task are exchanged);
  - – if neither aperiodic requests nor periodic task instances are pending, there is an idle time and the capacity C is consumed until, at most, it is exhausted.

An Example of Dynamic Priority Exchange Server

- Two periodic tasks, τ_1 and τ_2, with periods T_1=8 and T_2=12 and worst-case execution times C_1=2 and C_2=3, and a DPE server with period T_s=6 and capacity C_s=3, are present in the system.



Dynamic Sporadic Server

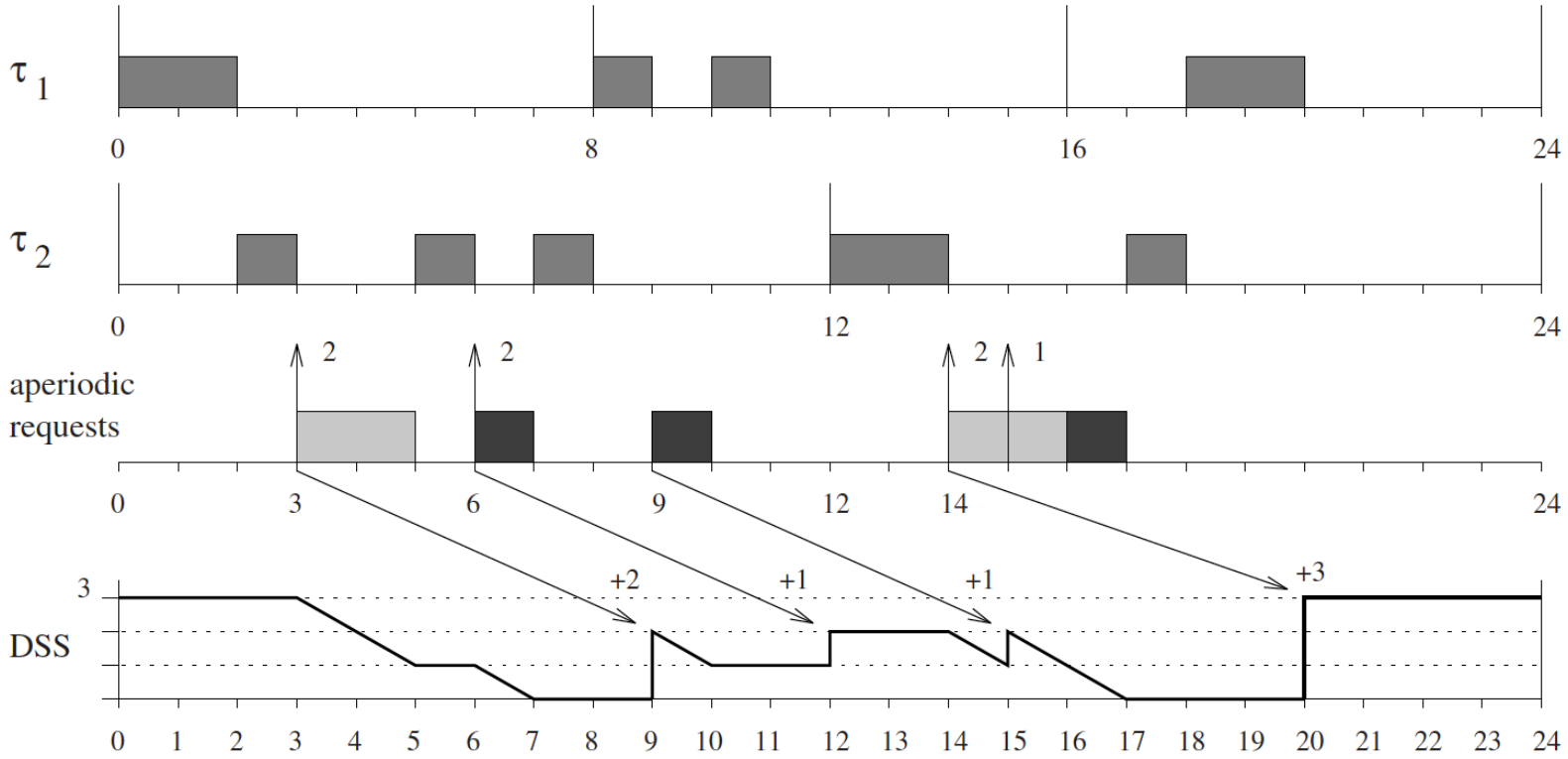
- The *Dynamic Sporadic Server* (DSS) is an aperiodic service strategy proposed by Spuri and Buttazzo [SB94, SB96] that extends the Sporadic Server [SSL89] to work under a dynamic EDF scheduler.
- DSS is characterized by a period T_s and a capacity C_s, which is preserved for possible aperiodic requests.
- Unlike other server algorithms, however, the capacity is not replenished at its full value at the beginning of each server period but only when it has been consumed.
- The times at which the replenishments occur are chosen according to a replenishment rule, which allows the system to achieve full processor utilization.

Dynamic Sporadic Server (2)

- The main difference between the classical SS and its dynamic version consists in the way the priority is assigned to the server.
- Whereas SS has a fixed priority according to the RM, DSS has a dynamic priority assigned through a suitable deadline.
- The rules of deadline assignment and the capacity replenishment:
  - When the server is created, its capacity C_s is initialized at its maximum value.
  - The next replenishment time RT and the current server deadline d_s are set as soon as $C_s>0$ and there is an aperiodic request pending. If t_A is such a time, then $RT=d_s=t_A + T_s$.
  - The replenishment amount RA to be done at time RT is computed when the last aperiodic request is completed or C_s has been exhausted. If t_I is such a time, then RA is set equal to the capacity consumed within the interval $[t_A, t_I]$.

An Example of Dynamic Sporadic Server

- An EDF schedule obtained on a task set consisting of two periodic tasks with periods T_1=8, T_2=12 and execution times C_1=2, C_2=3, and a DSS with period T_s=6 and capacity C_s=3.



Total Bandwidth Server

- Looking at the characteristics of the Sporadic Server algorithm, it can be easily seen that, when the server has a long period, the execution of the aperiodic requests can be delayed significantly.
  - This is due to the fact that when the period is long, the server is always scheduled with a far deadline.
  - This is regardless of the aperiodic execution times.

Total Bandwidth Server (2)

- There are two possible approaches to reduce the aperiodic response times.
  1. Using a Sporadic Server with a shorter period.
     - Increases the run-time overhead of the algorithm because, to keep the server utilization constant, the capacity has to be reduced proportionally, but this causes more frequent replenishments and increases the number of context switches with the periodic tasks.
  2. Assigning a possible earlier deadline to each aperiodic request. The assignment must be done in such a way that the overall processor utilization of the aperiodic load never exceeds a specified maximum value U_s.

Total Bandwidth Server (3)

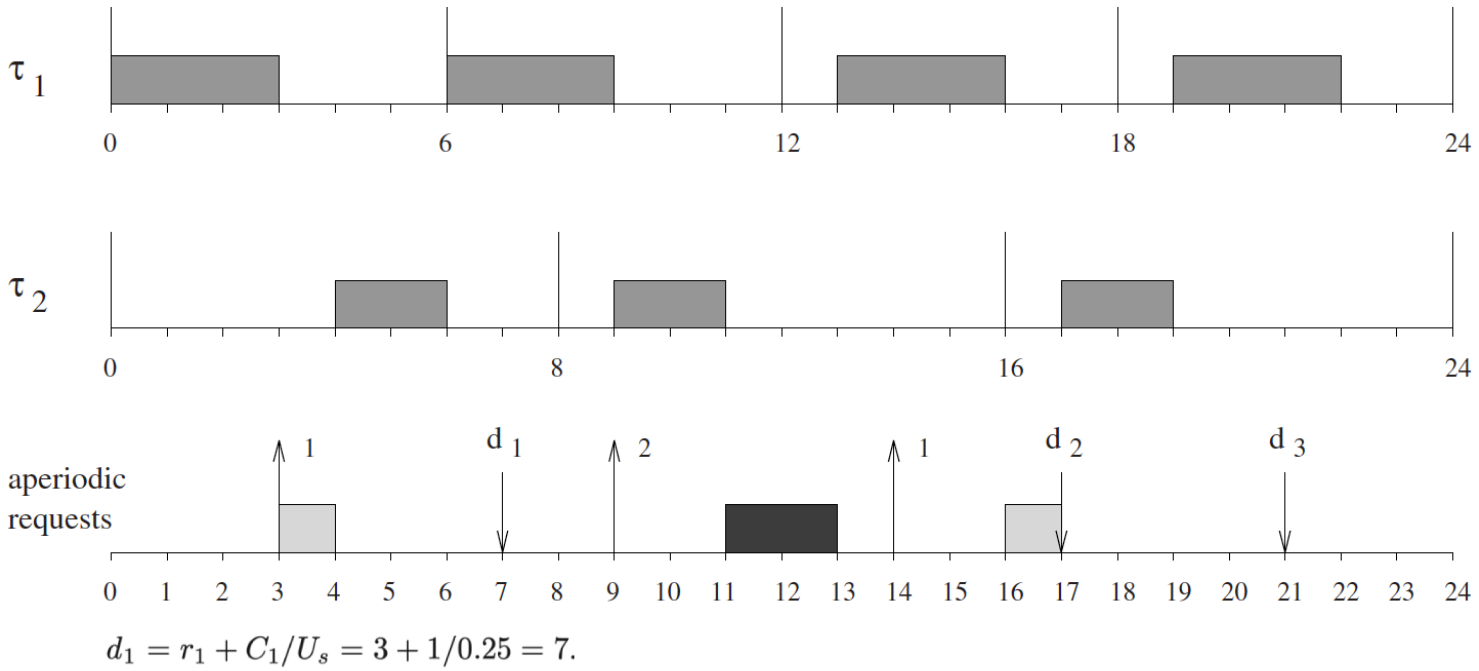
- The *Total Bandwidth Server* (TBS) is a simple and efficient aperiodic service mechanism proposed by Spuri and Buttazzo [SB94, SB96].
- The name of the server comes from the fact that, each time an aperiodic request enters the system, the total bandwidth of the server is immediately assigned to it, whenever possible.
- 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 $d_0=0$. Note that in the deadline assignment rule the bandwidth allocated to previous aperiodic requests is considered through the deadline d_{k-1}.

An Example of Total Bandwidth Server

- An EDF schedule obtained on a task set consisting of two periodic tasks with periods T_1=8, T_2=12 and execution times C_1=3, C_2=2, and a TBS with the utilization $U_s = 1 - U_p = 0.25$.



$d_1 = r_1 + C_1/U_s = 3 + 1/0.25 = 7.$

Earliest Deadline Late Server

- Using the available slack of periodic tasks for advancing the execution of aperiodic requests is the basic principle adopted by the EDL server [SB94, SB96].
- This aperiodic service algorithm can be viewed as a dynamic version of the Slack Stealing algorithm [LRT92].
- The definition of the EDL server makes use of some results presented by Chetto and Chetto [CC89].
  - Two complementary versions of EDF – namely, EDS and EDL – are proposed.
  - Under EDS the active tasks are processed as soon as possible, whereas under EDL they are processed as late as possible.

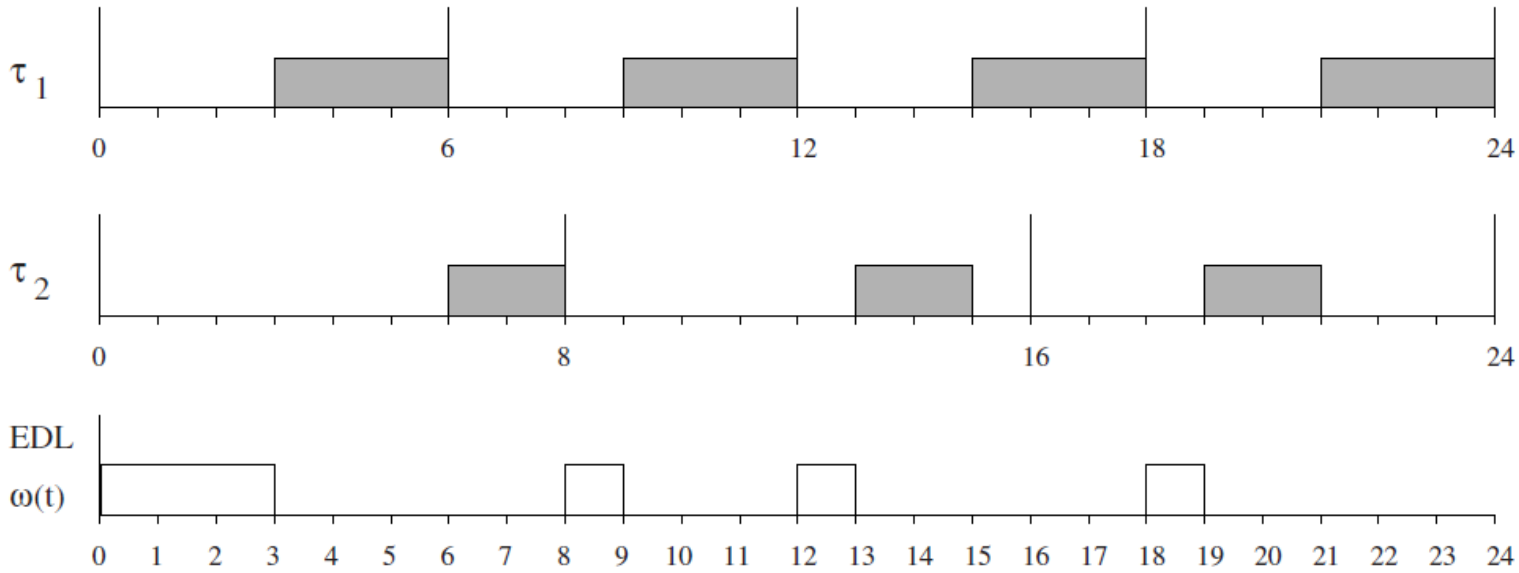
Earliest Deadline Late Server (2)

- An important property of EDL is that in any interval [0, t] it guarantees the maximum available idle time.
- This result is used to build an acceptance test for aperiodic tasks with hard deadlines, while here it is used to build an optimal server mechanism for soft aperiodic activities.
- To simplify the description of the EDL server, $\omega_J^A(t)$ denotes the following *availability function*, defined for a scheduling algorithm A and a task set J:

$$\omega_J^A(t) = \begin{cases} 1 & \text{if the processor is idle at } t \\ 0 & \text{otherwise.} \end{cases}$$

Availability Function under EDL

- The integral of $\omega_J^A(t)$ on an interval of time $[t_1, t_2]$ is denoted by $\Omega_J^A(t_1, t_2)$ and gives the total idle time in the specified interval.



The result of Optimality of EDL

- **Theorem (Chetto and Chetto).** Let J be any aperiodic task set and A any preemptive scheduling algorithm. For any instant t:

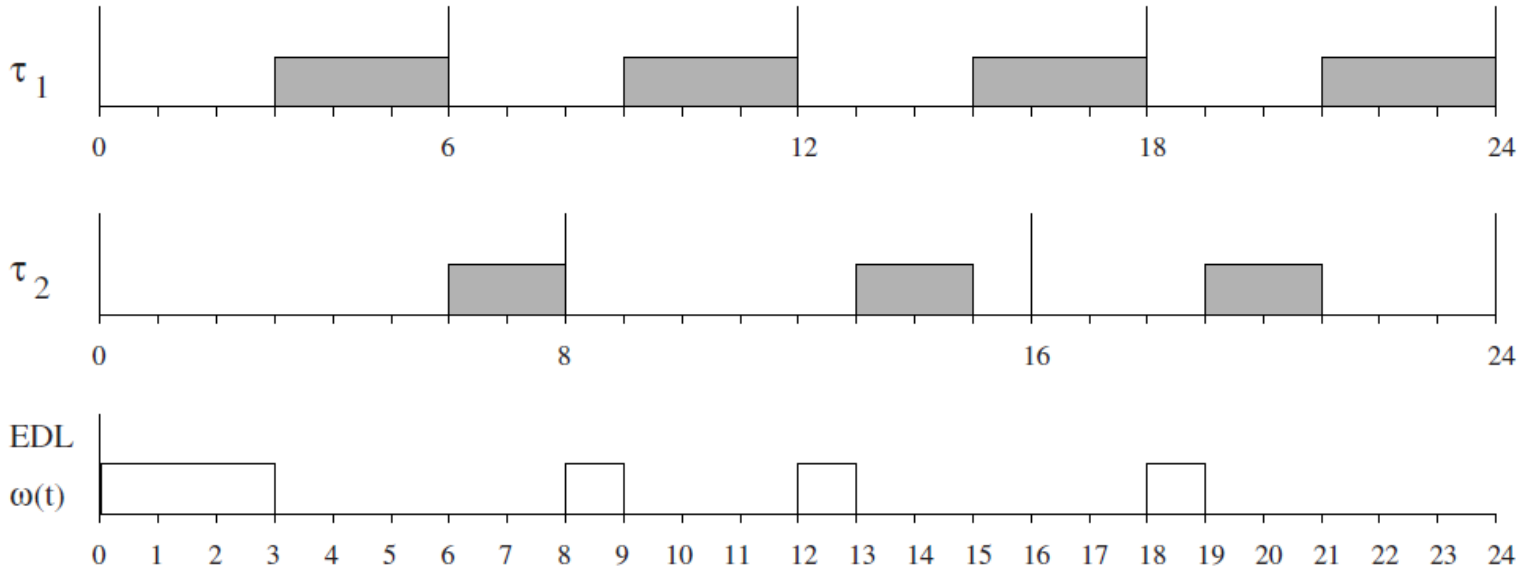
$$\Omega_J^{\mathrm{EDL}}(0,t) \geq \Omega_J^A(0,t).$$

- This result allows to develop an optimal server using the idle times of an EDL scheduler.
- In particular, given a periodic task set J, the function $\omega_J^A(t)$, which is periodic with *hyperperiod* H=lcm(T_1, ..., T_n), can be represented by means of two arrays.
  - The first, $E = (e_0, e_1, \ldots, e_p)$, represents the times at which idle times occur, while the second, $D = (\Delta_0, \Delta_1, \ldots, \Delta_p)$, represents the lengths of these idle times.

The result of Optimality of EDL (2)

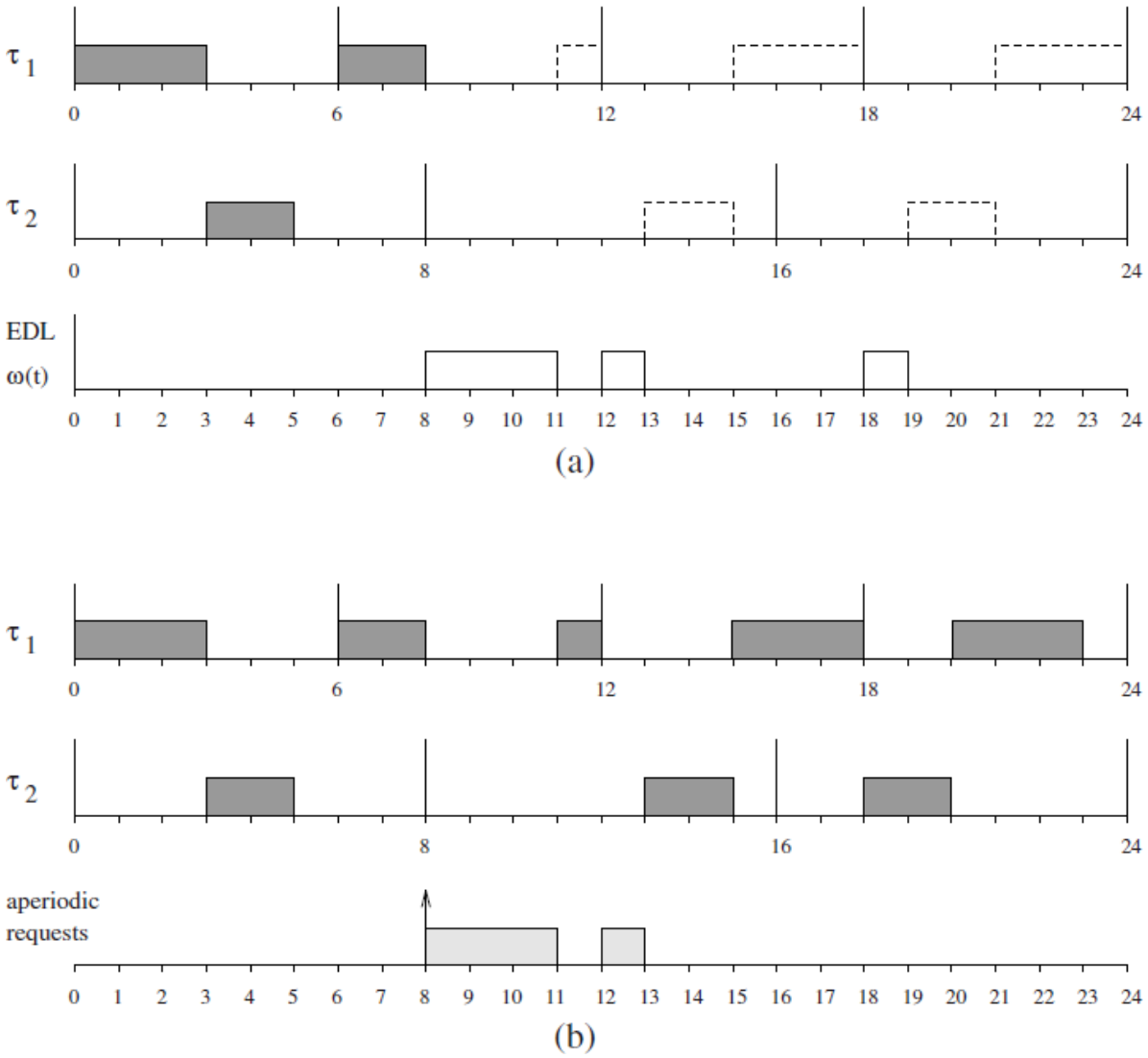
- The two arrays for the example in slide 50 are shown in the following Table. Note that idle times occur only after the release of a periodic task instance.

i	0	1	2	3
e_i	0	8	12	18
Δ_i	3	1	1	1



An Example of the EDL Service Mechanism

- The idle times of an EDL schedule are recomputed using the current periodic tasks.
- The response times achieved by this method are optimal, so they cannot be reduced further.



Improved Priority Exchange Server

- Although optimal, the EDL server has too much overhead to be considered practical.
- The heavy computation of the idle times can be avoided by using the mechanism of priority exchanges.
- With this mechanism, in fact, the system can easily keep track of the time advanced to periodic tasks and possibly reclaim it at the right priority level.
- The idle times of the EDL algorithm can be pre-computed offline and the server can use them to schedule aperiodic requests, when there are any, or to advance the execution of periodic tasks.
- The idle time advanced can be saved as aperiodic capacity at the priority levels of the periodic tasks executed.

Improved Priority Exchange Server (2)

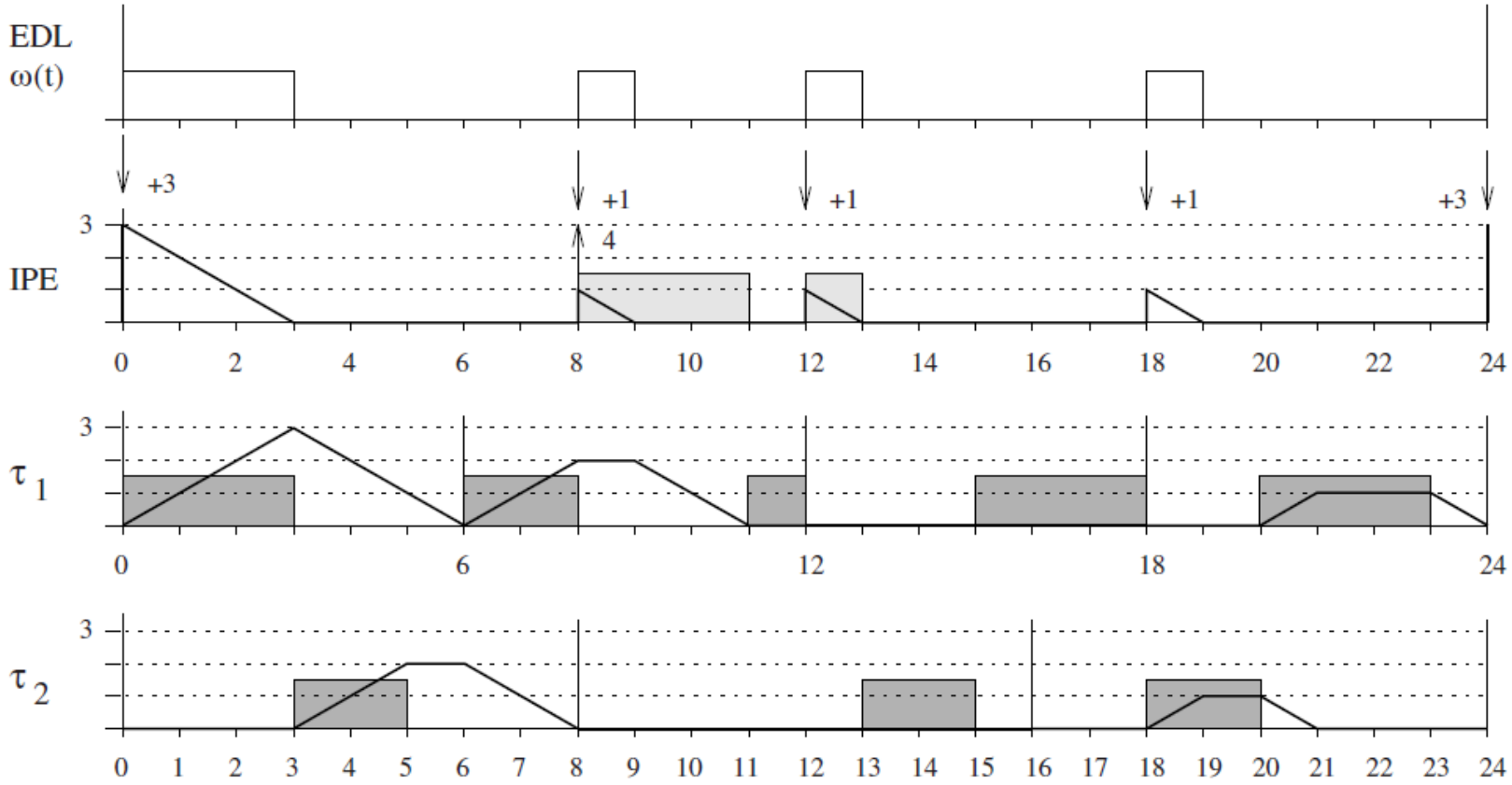
- The *Priority Exchange Server* (DPE) server is modified using the idle times of an EDL scheduler.
- There are two main advantages in this approach.
  1. A far more efficient replenishment policy is achieved for the server.
  2. The resulting server is no longer periodic, and it can always run at the highest priority in the system.

Improved Priority Exchange Server (3)

- The IPE server is thus defined in the following way:
  - The IPE server has an aperiodic capacity, initially set to 0.
  - At each instant $t=e_i+kH$, with $0 \leq i \leq p$ and $k \geq 0$, a replenishment of Δ_i units of time is scheduled for the server capacity; that is, at time $t = e_0$ the server will receive Δ_0 units of time (the two arrays E and D have been defined in the previous method).
  - The server priority is always the highest in the system, regardless of any other deadline.
  - All other rules of IPE (aperiodic requests and periodic instances executions, exchange and consumption of capacities) are the same as for a DPE server.

An Example of Improved Priority Exchange

- Note that the server replenishments are set according to the function $\omega_{\tau}^{\mathrm{EDL}}$.



Improving TBS

- The deadline assignment rule used by the TBS algorithm is a simple and efficient technique for servicing aperiodic requests in a hard real-time periodic environment.
- At the cost of a slightly higher complexity, such a rule can be modified to enhance aperiodic responsiveness.
- The key idea is to shorten the deadline assigned by the TBS as much as possible, still maintaining the periodic tasks schedulable [BS99].
- If d_k is the deadline assigned to an aperiodic request by the TBS, a new deadline d'_k can be set at the estimated worst-case finishing time f_k of that request, scheduled by EDF with deadline d_k.

Improving TBS (2)

- The following lemma shows that setting the new deadline d'_k at the current estimated worst-case finishing time does not jeopardize schedulability:
- *Lemma. Let σ be a feasible schedule of task set T, in which an aperiodic job J_k is assigned a deadline d_k, and let f_k be the finishing time of J_k in σ. If d_k is substituted with $d'_k=f_k$, then the new schedule σ' produced by EDF is still feasible.*
- The process of shortening the deadline can be applied recursively to each new deadline, until no further improvement is possible, given that the schedulability of the periodic task set must be preserved.
- If d_k^s is the deadline assigned to the aperiodic request J_k at step s and f_k^s is the corresponding finishing time in the current EDF schedule (achieved with d_k^s), the new deadline d_k^{s+1} is set at time f_k^s.
- At each step, the schedulability of the task set is guaranteed by the presented lemma.

Improving TBS (3)

- The algorithm stops either when $d_k^s = d_k^{s-1}$ or after a maximum number of steps defined by the system designer for bounding the complexity.
- The exact evaluation of f_k^s would require the development of the entire schedule up to the finishing time of request J_k, scheduled with d_k^s.
- There is no need to evaluate the exact value of f_k^s to shorten the deadline. Rather, the following upper bound can be used:

$$\tilde{f}_k^s = t + C_k^a + I_p(t, d_k^s),$$

- where t is the current time (corresponding to the release time r_k of request J_k or to the completion time of the previous request), C_k^a is the worst-case computation time required by J_k, and $I_p(t,\ d_k^s)$ is the interference on J_k due to the periodic instances in the interval $[t, d_k^s)$.
- $\tilde{f}_k^s$ is an upper bound for f_k^s because it identifies the time at which J_k and all the periodic instances with deadline less than d_k^s end to execute.
- Hence, $f_k^s \leq \tilde{f}_k^s$.

Improving TBS (4)

- The periodic interference $I_p(t, d_k^s)$ can be expressed as the sum of two terms, $\mathrm{I_a}(t, d_k^s)$ and $\mathrm{I_f}(t, d_k^s)$, where $\mathrm{I_a}(t, d_k^s)$ is the interference due to the currently active periodic instances with deadlines less than d_k^s, and $\mathrm{I_f}(t, d_k^s)$ is the future interference due to the periodic instances activated after time t with deadline before d_k^s. Hence,

$$I_a(t, d_k^s) = \sum_{\tau_i \text{ active},\ d_i < d_k^s} c_i(t)$$

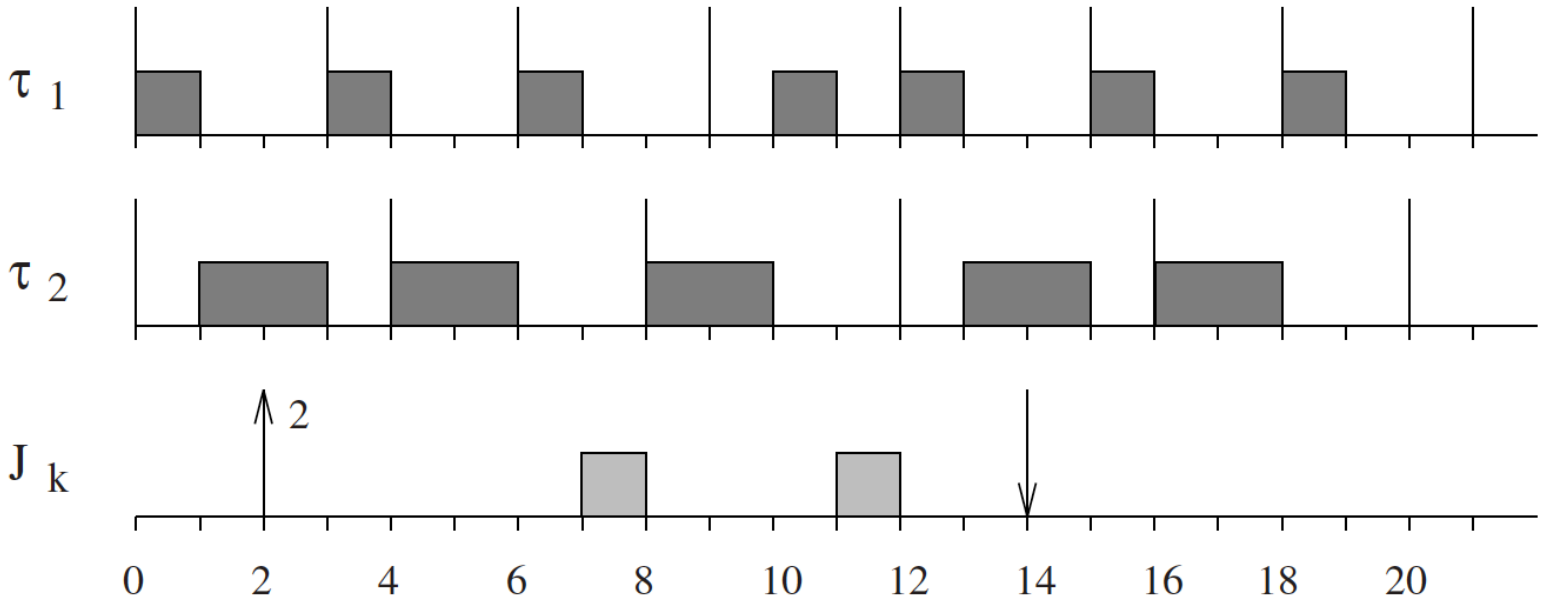
$$I_f(t, d_k^s) = \sum_{i=1}^{n} \max\left(0, \left\lceil \frac{d_k^s - next\_r_i(t)}{T_i} \right\rceil - 1\right) C_i,$$

- where $next\_ri(t)$ identifies the time greater than t at which the next periodic instance of task τ_i will be activated. If periodic tasks are synchronously activated at time zero, then:

$$next\_r_i(t) = \left\lceil \frac{t}{T_i} \right\rceil T_i.$$

An Example of Improving TBS

- Two periodic tasks and a single aperiodic job.
- U_p=10/12=5/6 and U_s=2/12=1/6.
- When the aperiodic request arrives at time t=2, it receives a deadline $d_k^0 = r_k + C_k^a/U_s$=14, according to the TBS algorithm.



An Example of Improving TBS (2)

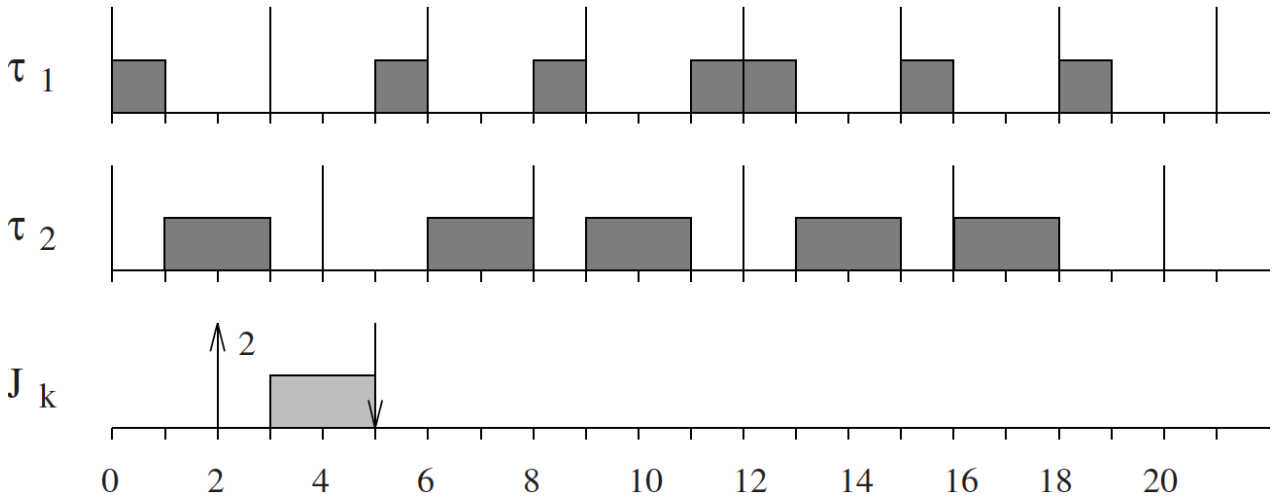
- By applying equations in slide 29, we have:

$$I_a(2,14) = c_2(2) = 1$$
$$I_f(2,14) = 3C_1 + 2C_2 = 7,$$

- And, by Equation in slide 28:

$$d_k^1 = \tilde{f}_k^0 = t + C_k^a + I_a + I_f = 12.$$

$step$	d_k^s	f_k^s
0	14	12
1	12	9
2	9	8
3	8	6
4	6	5
5	5	5



Evaluation Summary of Dynamic Priority Servers

	performance	computational complexity	memory requirement	implementation complexity
DPE				
DSS				
TBS				
EDL				
IPE				
TB*				

Summary of Dynamic Priority Servers

- Introduction to Dynamic Priority Servers
- Different scheduling methods of Dynamic Priority Servers
  - Dynamic Priority Exchange Server
  - Dynamic Sporadic Server
  - Total Bandwidth Server
  - Earliest Deadline Late Server
  - Improved Priority Exchange Server
  - Improving TBS
- Evaluation of Dynamic Priority Servers