Sharif University of Technology
Department of Computer Science and Engineering

Lec. 4:
Fixed Priority Servers

Real-Time Computing

S. Safari
2023

Fixed Priority Servers

Introduction

- Many real-time control applications, require both types of periodic and aperiodic processes, which may also differ for their criticality.
- Typically, periodic tasks are time-driven and execute critical control activities with hard timing constraints aimed at guaranteeing regular activation rates.
- Aperiodic tasks are usually event-driven and may have hard, soft, or non-real-time requirements depending on the specific application.
- When dealing with hybrid task sets, the main objective of the kernel is to guarantee the schedulability of all critical tasks in worst-case conditions and provide good average response times for soft and non-real-time activities.

Introduction (2)

- Off-line guarantee of event-driven aperiodic tasks with critical timing constraints can be done only by making proper assumptions on the environment.
  - Assuming a maximum arrival rate for each critical event.
    - This implies that aperiodic tasks associated with critical events are characterized by a minimum interarrival time between consecutive instances, which bounds the aperiodic load.
- Aperiodic tasks characterized by a minimum interarrival time are called *sporadic*.
  - They are guaranteed under peak-load situations by assuming their maximum arrival rate.

Assumptions

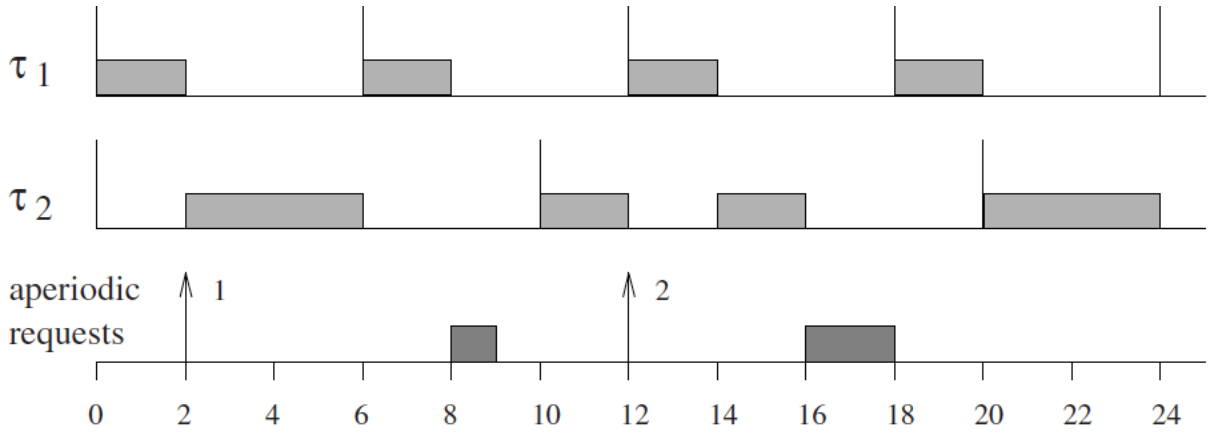
- Presenting a number of scheduling algorithms for handling hybrid task sets consisting of a subset of hard periodic tasks and a subset of soft aperiodic tasks.
- All algorithms rely on the following assumptions:
  - Periodic tasks are scheduled based on a fixed-priority assignment; namely, the Rate-Monotonic (RM) algorithm;
  - All periodic tasks start simultaneously at time $t=0$ and their relative deadlines are equal to their periods.
  - Arrival times of aperiodic requests are unknown.
  - When not explicitly specified, the minimum interarrival time of a sporadic task is assumed to be equal to its deadline.
  - All tasks are fully preemptable.

Background scheduling

- The simplest method to handle a set of soft aperiodic activities in the presence of periodic tasks is to schedule them in background.
  - When there are not periodic instances ready to execute.
- Major problem: The response time of aperiodic requests can be too long for certain applications.
  - Background scheduling can be adopted only when the aperiodic activities do not have stringent timing constraints and the periodic load is not high.

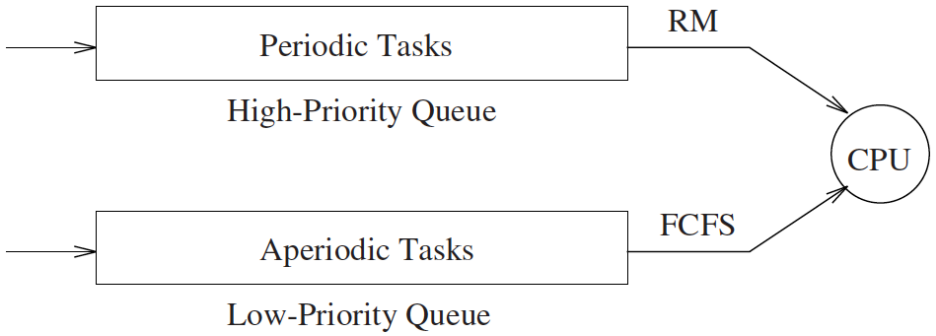
An Example of Background Scheduling

- An example in which two periodic tasks are scheduled by RM, while two aperiodic tasks are executed in background.
- Since the processor utilization factor of the periodic task set (U=0.73) is less than the least upper bound for two tasks ($U_{lub}(2)$=0.83), the periodic tasks are schedulable by RM.
- Note that the guarantee test does not change in the presence of aperiodic requests, since background scheduling does not influence the execution of periodic tasks.



Advantage of Background Scheduling

- The major advantage of background scheduling is its simplicity.
- Two queues are needed:
  - One (with a higher priority) dedicated to periodic tasks and the other (with a lower priority) reserved for aperiodic requests.
  - The two queueing strategies are independent and can be realized by different algorithms.
  - Tasks are taken from the aperiodic queue only when the periodic queue is empty.
  - The activation of a new periodic instance causes any aperiodic tasks to be immediately preempted.



Polling Server

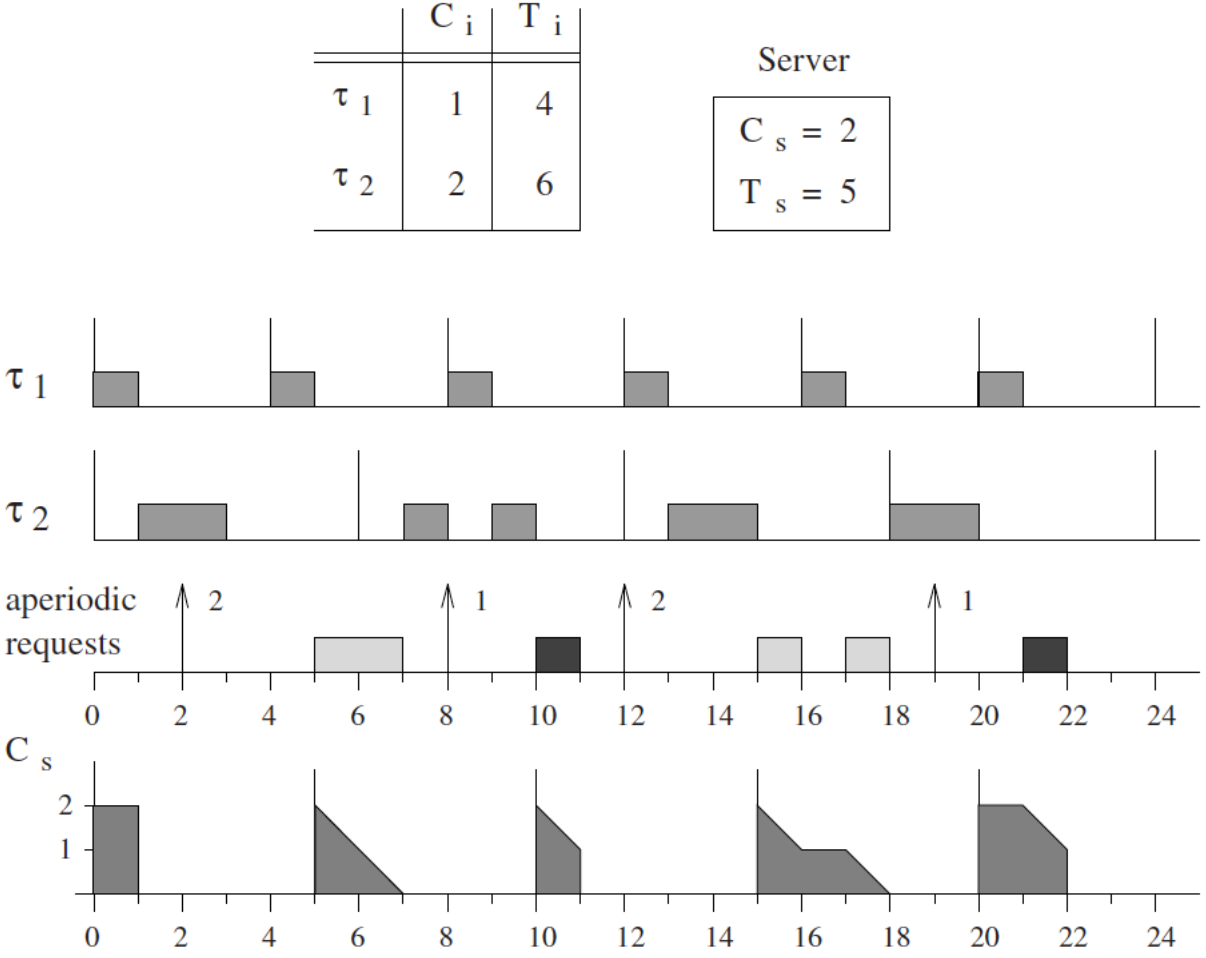
- The average response time of aperiodic tasks can be improved with respect to background scheduling through the use of a server.
  - A periodic task whose purpose is to service aperiodic requests as soon as possible.
- A server is characterized by a period T_s and a computation time C_s, called server capacity, or server budget.
- 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 budget.
- The ordering of aperiodic requests does not depend on the scheduling algorithm used for periodic tasks, and it can be done by arrival time, computation time, deadline, or any other parameter.

Polling Server (2)

- At regular intervals equal to the period T_s, *Polling Server* (PS) becomes active and serves the pending aperiodic requests within the limit of its capacity C_s.
- If no aperiodic requests are pending, PS suspends itself until the beginning of its next period, and the budget originally allocated for aperiodic service is discharged and given periodic tasks.
- If an aperiodic request arrives just after the server has suspended, it must wait until the beginning of the next period, when the server capacity is replenished at its full value.

An Example of Polling Server

- A Polling Server scheduled by RM.
- The aperiodic requests are reported on the third row, whereas the fourth row shows the server capacity as a function of time.
- Numbers beside the arrows indicate the computation times associated with the requests.



Schedulability Analysis of Polling Server

- Consider the problem of guaranteeing a set of hard periodic tasks in the presence of soft aperiodic tasks handled by a Polling Server.
  - In the worst case: A period equal to T_s and a computation time equal to C_s.
- Independently of the number of aperiodic tasks handled by the server, a maximum time equal to C_s is dedicated to aperiodic requests at each server period.
- As a consequence, the processor utilization factor of the Polling Server is $U_s = C_s/T_s$, and hence the schedulability of a periodic set with n tasks and utilization U_p can be guaranteed if:

$$U_p + U_s \leq U_{lub}(n+1).$$

Schedulability Analysis of Polling Server (2)

- If periodic tasks (including the server) are scheduled by RM, the schedulability test becomes:

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

- In general, in the presence of m servers, a set of n periodic tasks is schedulable by RM if:

$$U_p + \sum_{j=1}^{m} U_{s_j} \leq U_{lub}(n+m).$$

Schedulability Analysis of Polling Server (3)

- The least upper bound of utilization for a PS server with a highest priority and a set of periodic tasks that fully utilize the processor is:

$$U_{lub} = U_s + n\left[\left(\frac{2}{U_s+1}\right)^{1/n} - 1\right].$$

- Taking the limit of last Equation as n $\rightarrow \infty$, we find the worst-case bound as a function of U_s to be given by:

$$\lim_{n\to\infty} U_{lub} = U_s + \ln(K) = U_s + \ln\left(\frac{2}{U_s+1}\right).$$

- Thus, given a set of n periodic tasks and a Polling Server with utilization factors U_p and U_s, respectively, the schedulability of the periodic task set is guaranteed under RM if:

$$U_p + U_s \leq U_s + n\left(K^{1/n} - 1\right); \qquad U_p \leq n\left[\left(\frac{2}{U_s+1}\right)^{1/n} - 1\right].$$

Schedulability Analysis of Polling Server (4)

- The response time of a periodic task τ_i in the presence of a Polling Server at the highest priority can be found as the smallest integer satisfying the following recurrent relation:

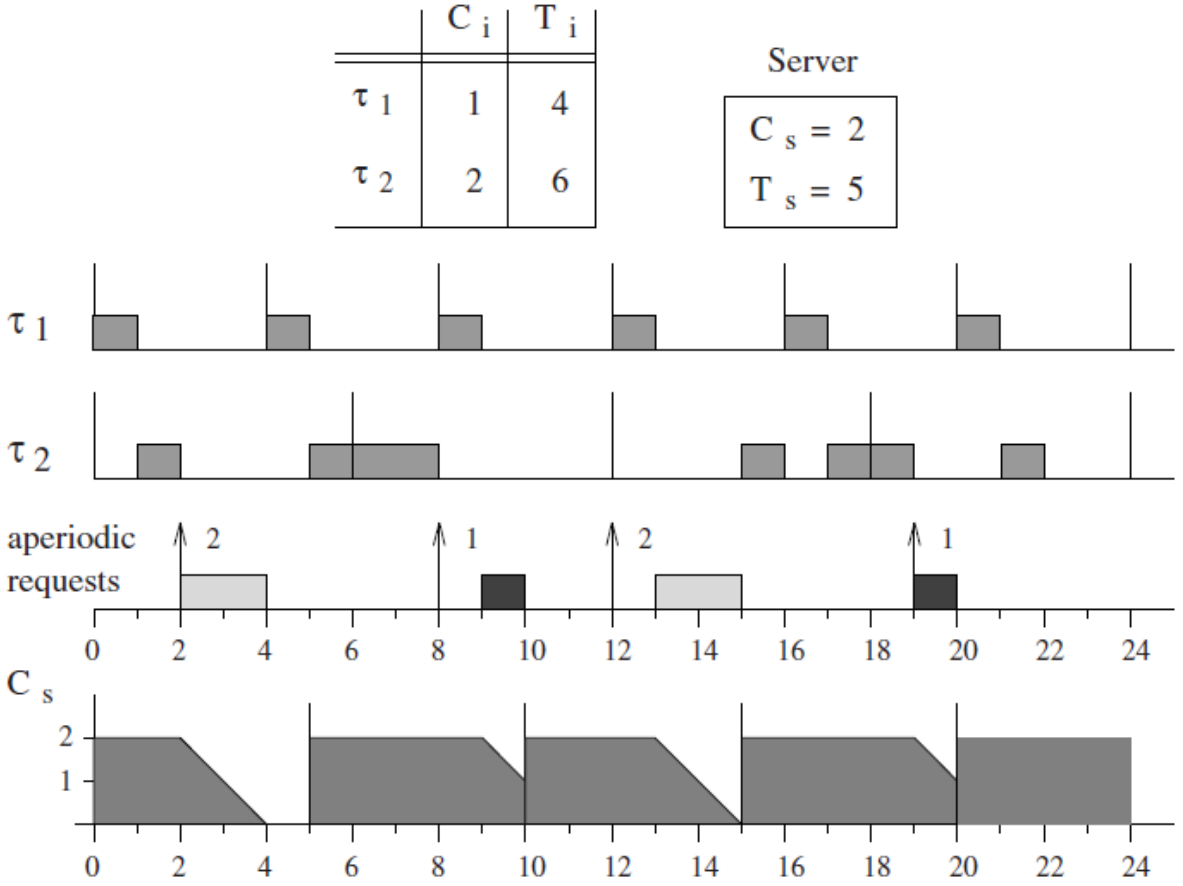
$$R_i = C_i + \left\lceil \frac{R_i}{T_s} \right\rceil C_s + \sum_{j=1}^{i-1} \left\lceil \frac{R_i}{T_j} \right\rceil C_j .$$

Deferrable Server

- The *Deferrable Server* (DS) algorithm is a service technique introduced by Lehoczky, Sha, and Strosnider [LSS87, SLS95] to improve the average response time of aperiodic requests with respect to polling service.
- As the Polling Server, the DS algorithm creates a periodic task (usually having a high priority) for servicing aperiodic requests.
- Unlike polling, DS preserves its capacity if no requests are pending upon the invocation of the server.
- The capacity is maintained until the end of the period.
  - Aperiodic requests can be serviced at the same server's priority at anytime, as long as the capacity has not been exhausted.
  - At the beginning of any server period, the capacity is replenished at its full value.

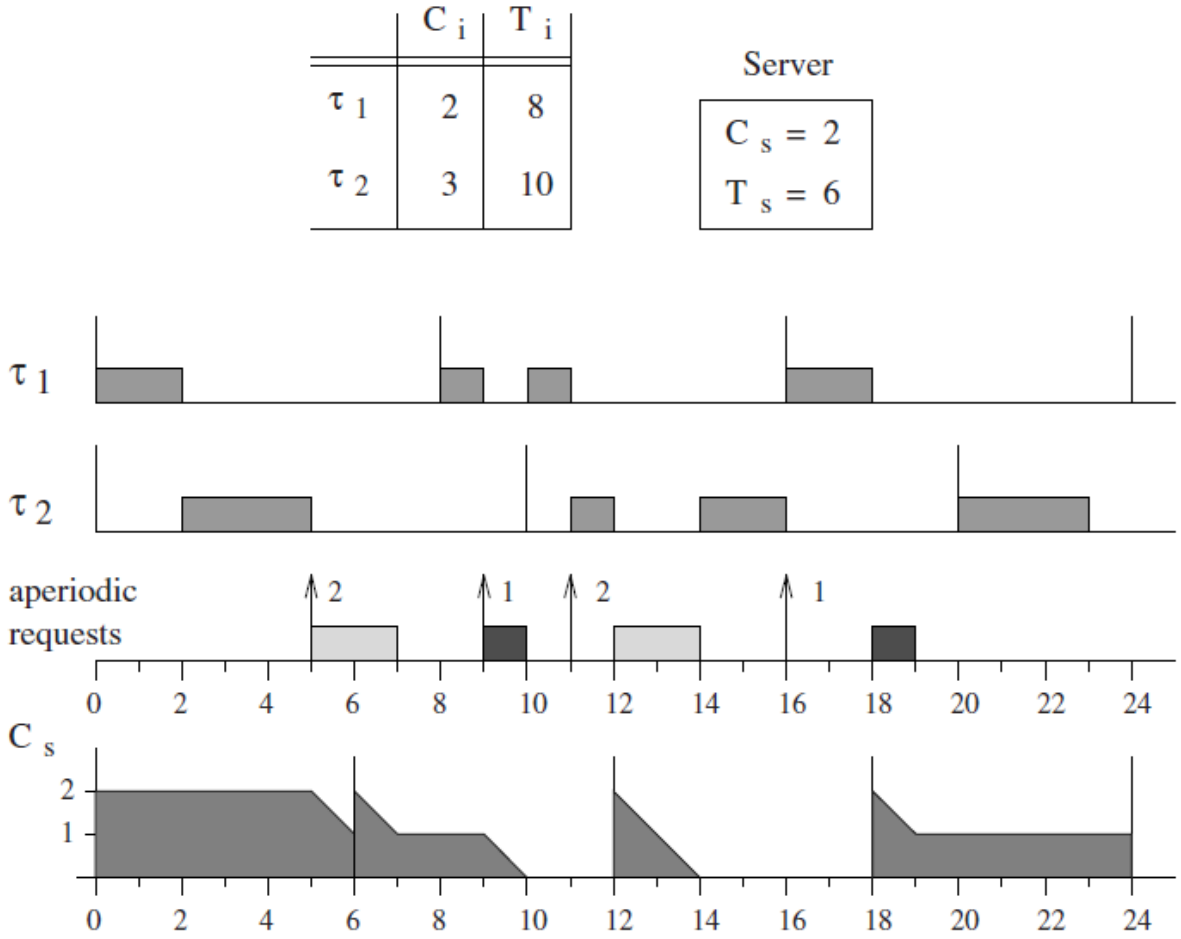
An Example of Deferrable Server

- DS provides much better aperiodic responsiveness than polling, since it preserves the capacity until it is needed.



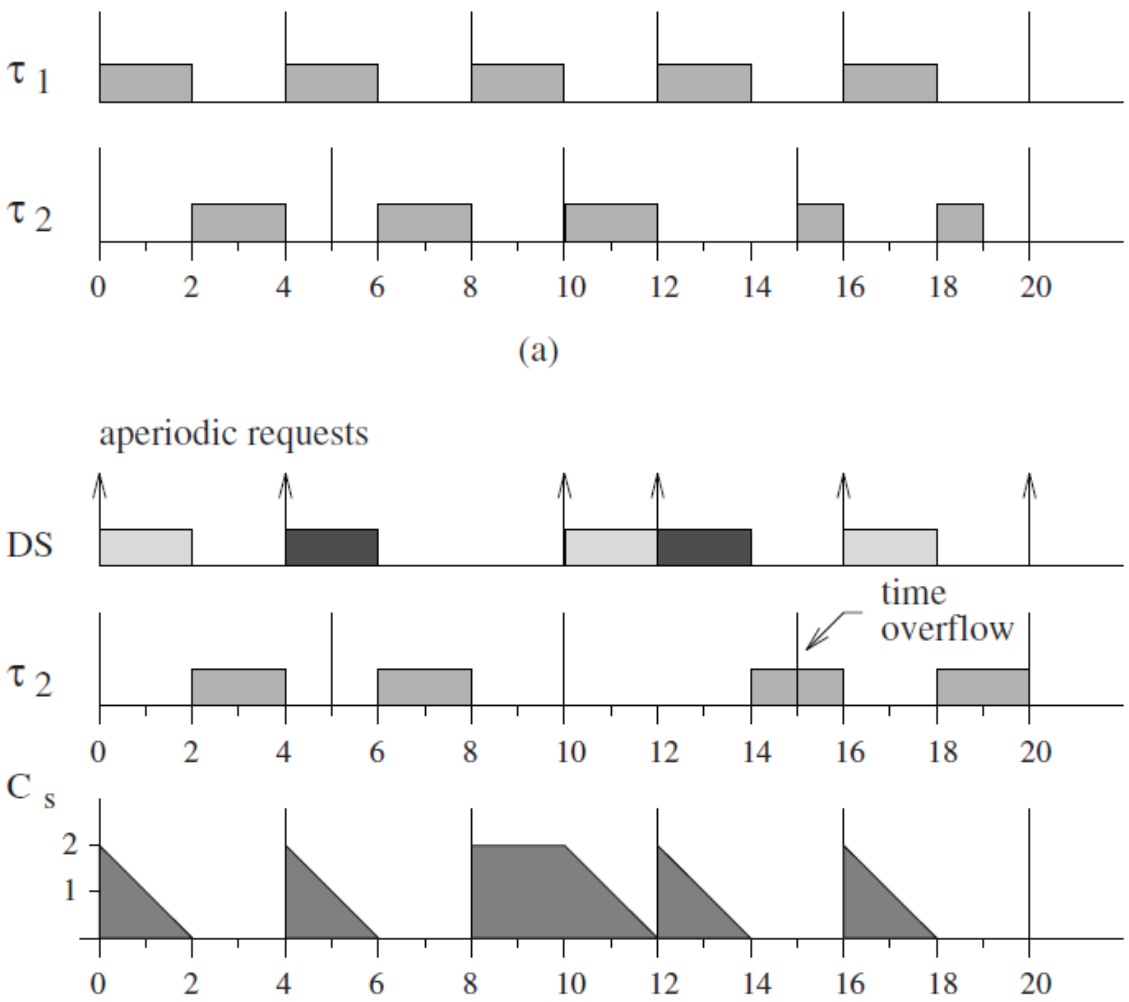
Another Example of Deferrable Server

- Shorter response times can be achieved by creating a Deferrable Server having the highest priority among the periodic tasks.
- The second aperiodic request preempts task τ_1, being $C_s>0$ and $T_s<T_1$, and it entirely consumes the capacity at time $t=10$.
- When the third request arrives at time $t=11$, the capacity is zero; hence, its service is delayed until the beginning of the next server period.



Schedulability Analysis of Deferrable Server

- Any schedulability analysis related to the Rate-Monotonic algorithm has been done on the implicit assumption that a periodic task cannot suspend itself, but must execute whenever it is the highest-priority task ready to run (assumption A5 in Lecture 3).
- It is easy to see that the Deferrable Server violates this basic assumption.



Schedulability Analysis of Deferrable Server (2)

- The least upper bound of utilization for a DS with a highest priority and a set of periodic tasks that fully utilize the processor is:

$$U_{lub} = U_s + n\left[\left(\frac{U_s+2}{2U_s+1}\right)^{1/n} - 1\right].$$

- Taking the limit of last Equation as n $\rightarrow \infty$, we find the worst-case bound as a function of U_s to be given by:

$$\lim_{n\to\infty} U_{lub} = U_s + \ln\left(\frac{U_s+2}{2U_s+1}\right).$$

- Thus, given a set of n periodic tasks and a Deferrable Server with utilization factors U_p and U_s, respectively, the schedulability of the periodic task set is guaranteed under RM if:

$$U_p \leq n\left(K^{1/n} - 1\right). \qquad K = \frac{U_s+2}{2U_s+1},$$

Priority Exchange

- The *Priority Exchange* (PE) algorithm is a scheduling scheme introduced by Lehoczky, Sha, and Strosnider [LSS87] for servicing a set of soft aperiodic requests along with a set of hard periodic tasks.
- With respect to DS, PE has a slightly worse performance in terms of aperiodic responsiveness but provides a better schedulability bound for the periodic task set.
- Like DS, the PE algorithm uses a periodic server (usually at a high priority) for servicing aperiodic requests.
- It differs from DS in the manner in which the capacity is preserved.
- Unlike DS, PE preserves its high-priority capacity by exchanging it for the execution time of a lower-priority periodic task.

An Example of Priority Exchange

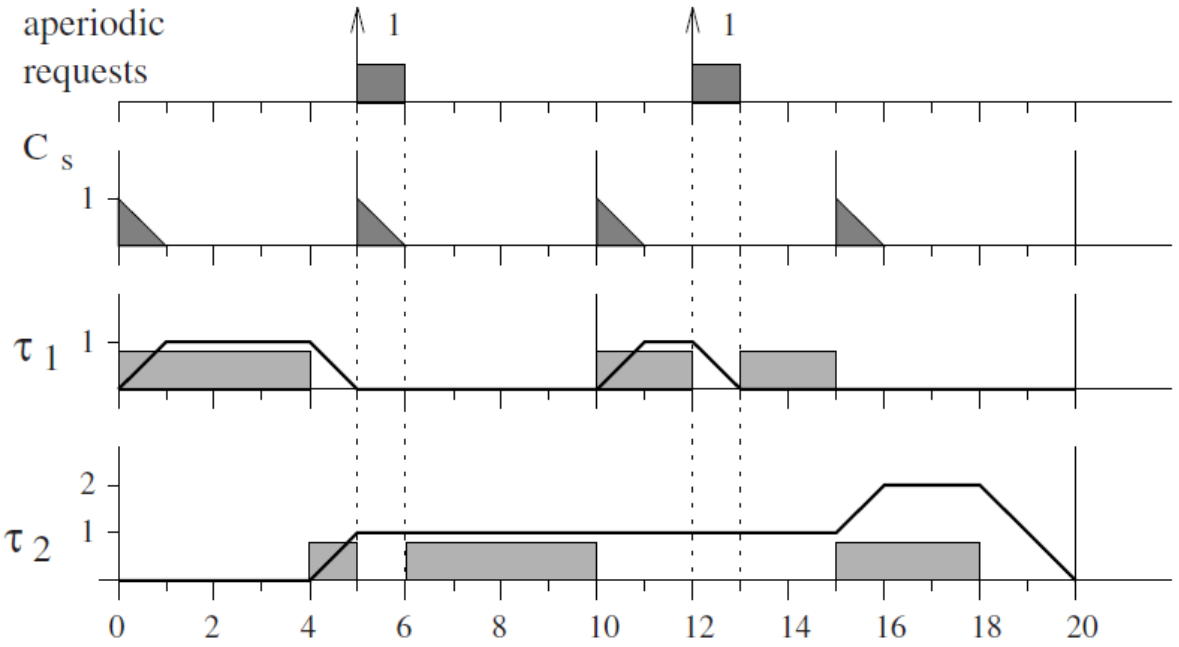
- An example of aperiodic scheduling using the PE
- At time t=15, a new high-priority replenishment takes place, but the capacity is exchanged with the execution time of τ_2.

	C_i	T_i
τ_1	4	10
τ_2	8	20

Server

$C_s = 1$

$T_s = 5$



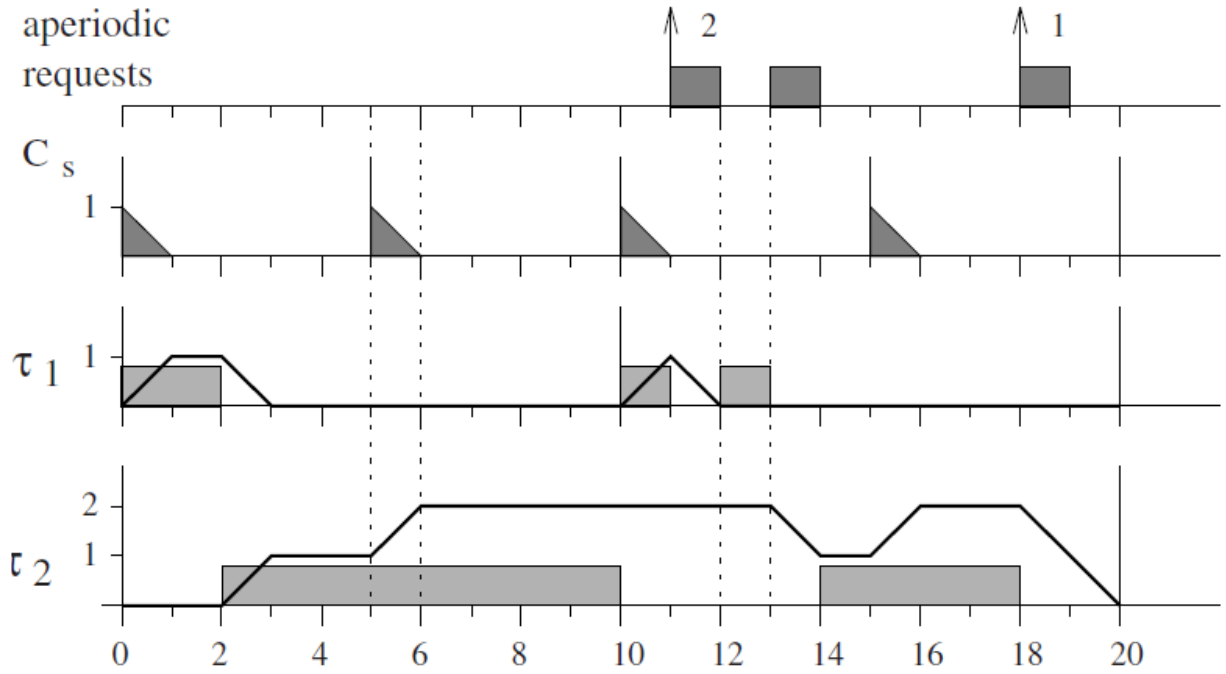
Another Example of Priority Exchange

	C_i	T_i
τ_1	2	10
τ_2	12	20

Server

$C_s = 1$

$T_s = 5$



Schedulability Analysis of Priority Exchange

- Considering that, in the worst case, a PE server behaves as a periodic task, the schedulability bound for a set of periodic tasks running along with a Priority Exchange server is the same as the one derived for the Polling Server. Hence, assuming that PE is the highest-priority task in the system, we have:

$$U_{lub} = U_s + n\left(K^{1/n} - 1\right). \qquad K = \frac{2}{U_s + 1}.$$

- Thus, given a set of *n* periodic tasks and a Priority Exchange server with utilization factors *Up* and *Us*, respectively, the schedulability of the periodic task set is guaranteed under RM if:

$$U_p \leq n\left[\left(\frac{2}{U_s + 1}\right)^{1/n} - 1\right].$$

Sporadic Server

- The *Sporadic Server* (SS) algorithm is another technique, proposed by Sprunt, Sha, and Lehoczky [SSL89], which allows the enhancement of the average response time of aperiodic tasks without degrading the utilization bound of the periodic task set.
- The SS algorithm creates a high-priority task for servicing aperiodic requests and, like DS, preserves the server capacity at its high-priority level until an aperiodic request occurs.
- SS differs from DS in the way it replenishes its capacity. Whereas DS and PE periodically replenish their capacity to full value at the beginning of each server period, SS replenishes its capacity only after it has been consumed by aperiodic task execution.

Sporadic Server (2)

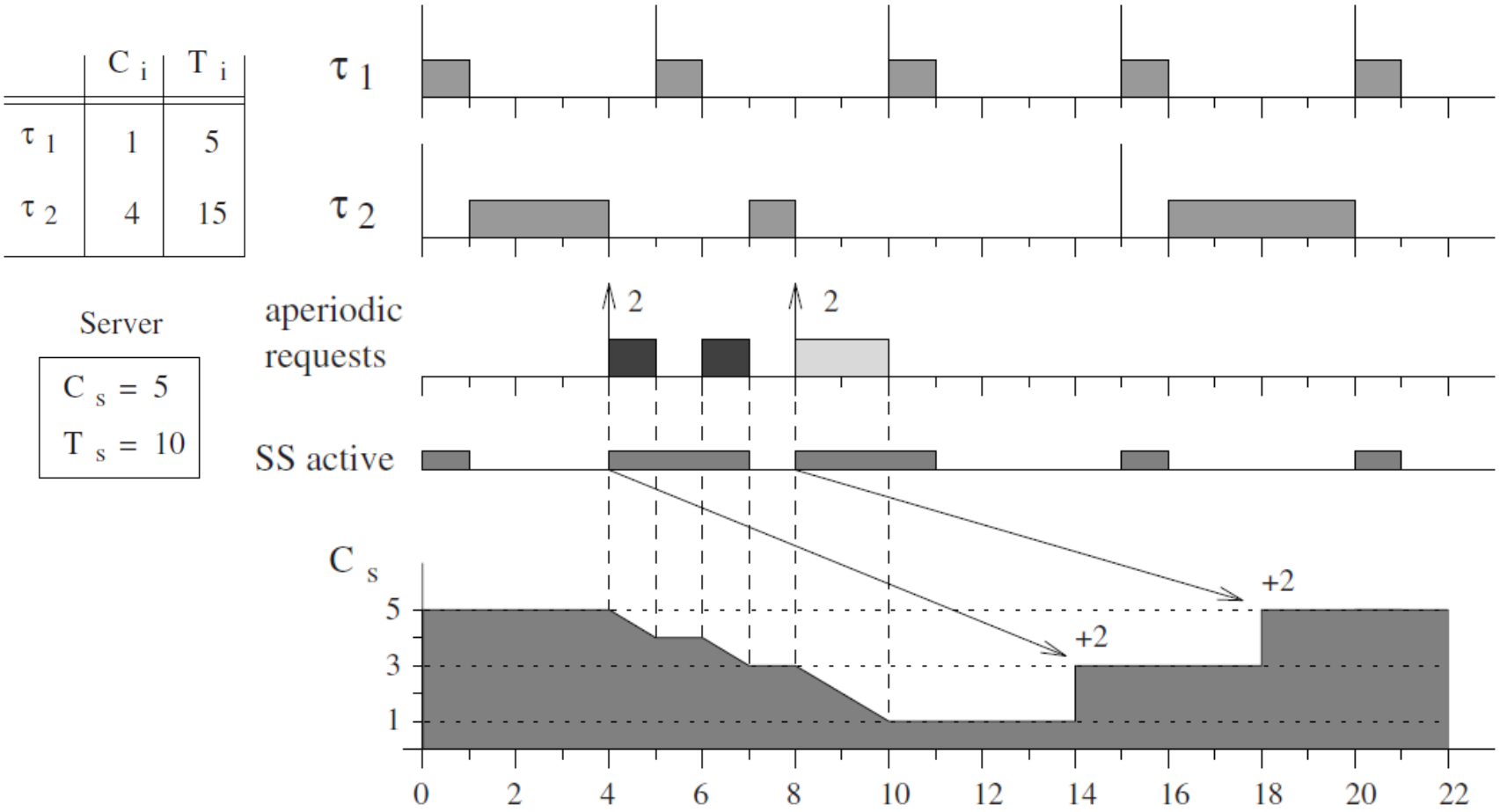
- In order to simplify the description of the replenishment method used by SS, the following terms are defined:

P_{exe}	It denotes the priority level of the task that is currently executing.
P_s	It denotes the priority level associated with SS.
Active	SS is said to be *active* when $P_{exe} \geq P_s$.
Idle	SS is said to be *idle* when $P_{exe} < P_s$.
RT	It denotes the *replenishment time* at which the SS capacity will be replenished.
RA	It denotes the *replenishment amount* that will be added to the capacity at time RT.

Sporadic Server (3)

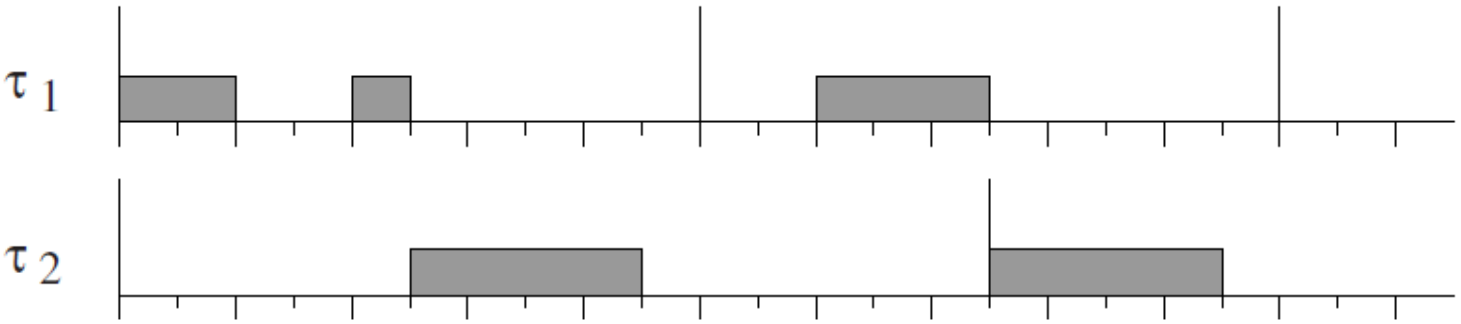
- Using this terminology, the capacity C_s consumed by aperiodic requests is replenished according to the following rule:
  - The replenishment time RT is set as soon as SS becomes active and $C_s > 0$. Let t_A be such a time. The value of RT is set equal to t_A plus the server period ($RT = t_A + T_s$).
  - The replenishment amount RA to be done at time RT is computed when SS becomes idle or C_s has been exhausted. Let t_I be such a time. The value of RA is set equal to the capacity consumed within the interval $[t_A, t_I]$.

An Example of Sporadic Server



Another Example of Sporadic Server

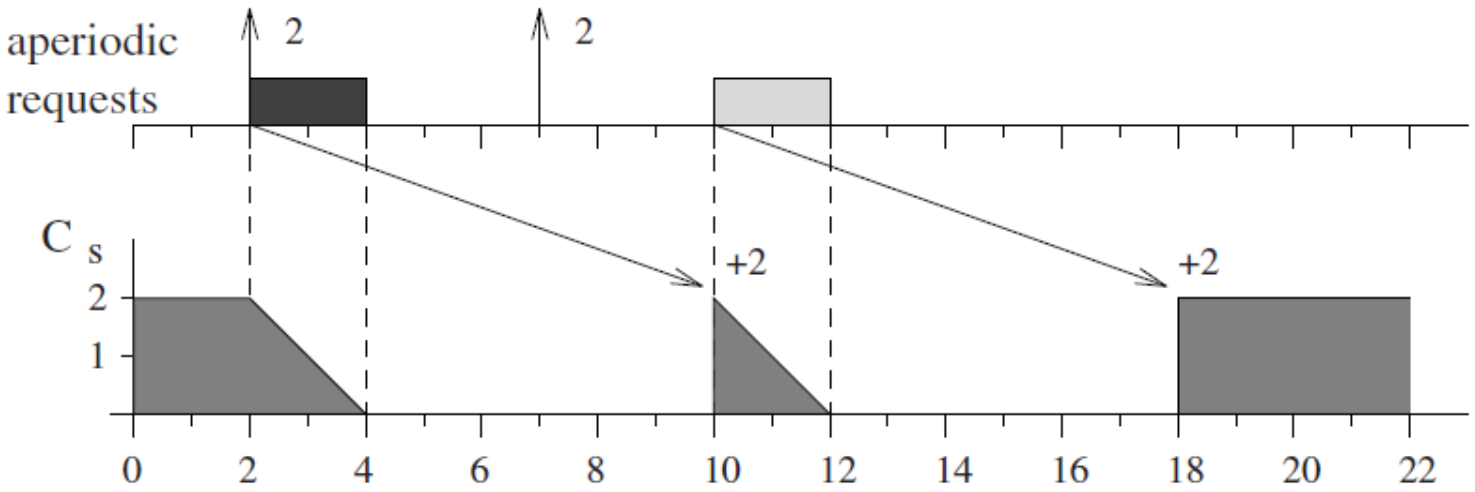
	C_i	T_i
τ_1	3	10
τ_2	4	15



Server

$C_s = 2$

$T_s = 8$



Slack Stealing

- The *Slack Stealing* algorithm is another aperiodic service technique which offers substantial improvements in response time over the previous service methods (PE, DS, and SS).
- Unlike PE, DS, and SS, the Slack Stealing algorithm does not create a periodic server for aperiodic task service.
  - It creates a passive task, referred to as the *Slack Stealer*, which attempts to make time for servicing aperiodic tasks by "stealing" all the processing time it can from the periodic tasks without causing their deadlines to be missed.
  - This is equivalent to stealing slack from the periodic tasks.
- We recall that if $c_i(t)$ is the remaining computation time at time t, the slack of a task τ_i is:

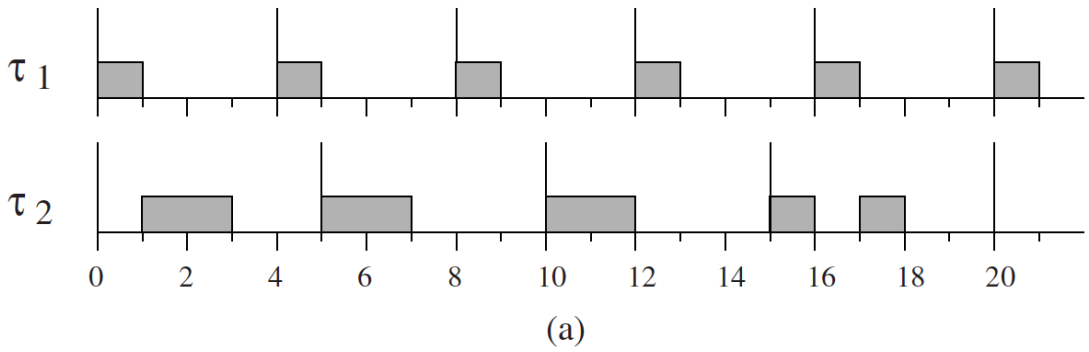
$$slack_i(t) = d_i - t - c_i(t).$$

Slack Stealing (2)

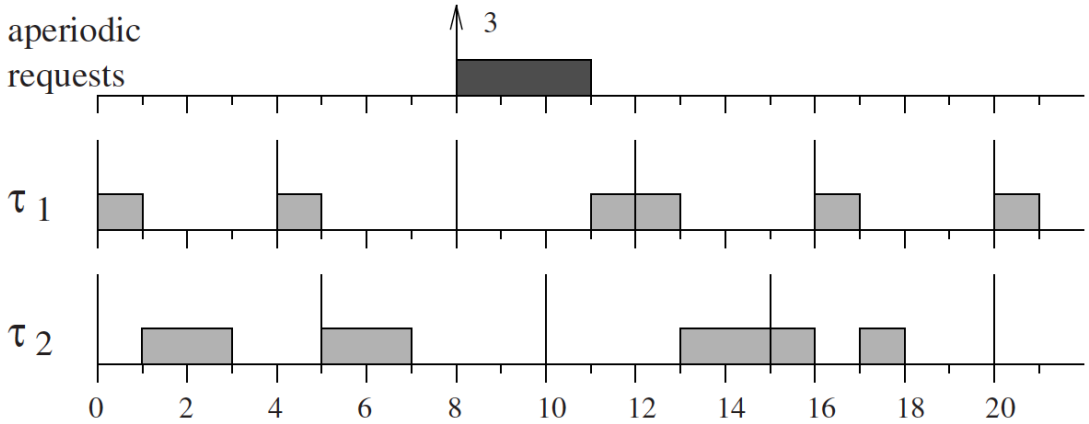
- The main idea behind slack stealing is that, typically, there is no benefit in early completion of the periodic tasks.
- When an aperiodic request arrives, the Slack Stealer steals all the available slack from periodic tasks and uses it to execute aperiodic requests as soon as possible.
- If no aperiodic requests are pending, periodic tasks are normally scheduled by RM.

An Example of Slack Stealing

- Example of Slack Stealer behavior:
  - a: when no aperiodic requests are pending;
  - b: when an aperiodic request of three units arrives at time $t = 8$.



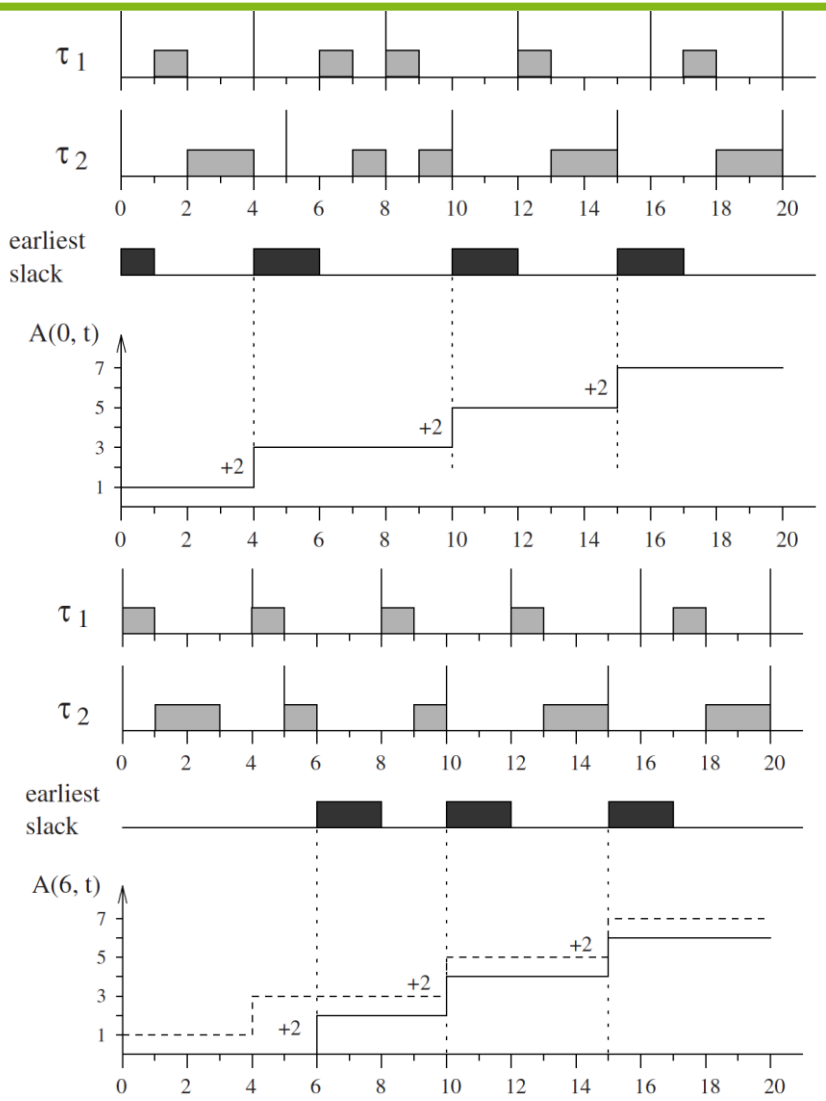
(a)



(b)

Slack Stealing Challenges

- In order to schedule an aperiodic request $J_a(r_a, C_a)$ according to the Slack Stealing algorithm, we need to determine the earliest time t such that at least C_a units of slack are available in $[r_a, t]$.
- The computation of the slack is carried out through the use of a slack function $A(s, t)$, which returns the maximum amount of computation time that can be assigned to aperiodic requests in the interval $[s, t]$ without compromising the schedulability of periodic tasks.



Evaluation Summary of Fixed-Priority Servers

	performance	computational complexity	memory requirement	implementation complexity
Background Service	poor	excellent	excellent	excellent
Polling Server	poor	excellent	excellent	excellent
Deferrable Server	good	excellent	excellent	excellent
Priority Exchange	good	good	good	good
Sporadic Server	good	good	good	good
Slack Stealer	excellent	poor	poor	poor

Summary of Fixed-Priority Servers

- Introduction to fixed priority servers and some assumptions
- Different scheduling methods of fixed-priority servers
  - Background
  - Polling server
  - Deferrable server
  - Priority exchange
  - Sporadic server
  - Slack stealer
- Evaluation of fixed-priority servers