Spring 2009 -Real-Time Systems

http://www.neu-rtes.org/courses/spring2009/

Chapter 4
Aperiodic Scheduling

Real-Time Embedded Systems Laboratory

Northeastern University

Mixed Tasksets

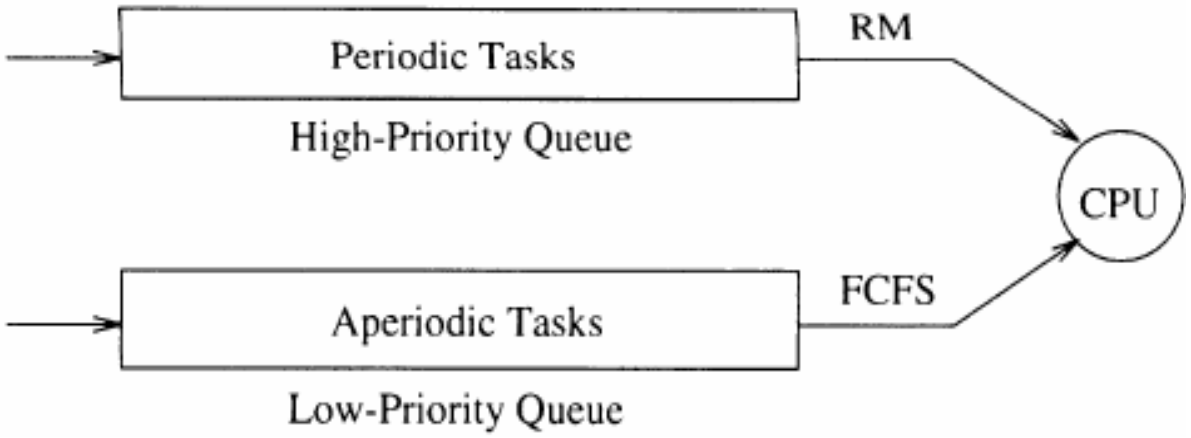
- In many applications, there are as well aperiodic as periodic tasks.
- ***Periodic tasks***: time-driven, execute critical control activities with hard timing constraints aimed at guaranteeing regular activation rates.
- ***Aperiodic tasks***: event-driven, may have hard, soft, non real-time requirements depending on the specific application.
- ***Sporadic tasks***: Aperiodic tasks characterized by a minimum interarrival time are called *sporadic*.

SOFT aperiodic tasks

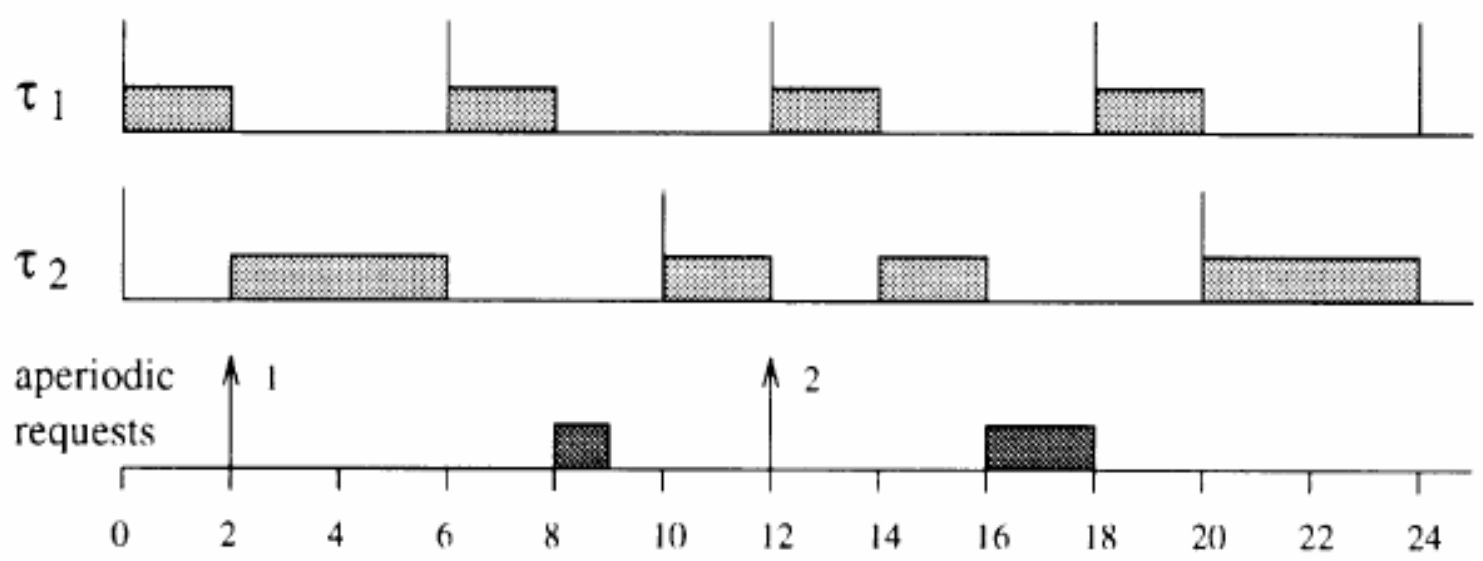
- • Aperiodic tasks with **SOFT** deadlines should be executed as soon as possible, but without jeopardizing HARD tasks.
- • We may be interested in
  - → minimizing the average response time
  - → performing an on-line guarantee

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.



Example



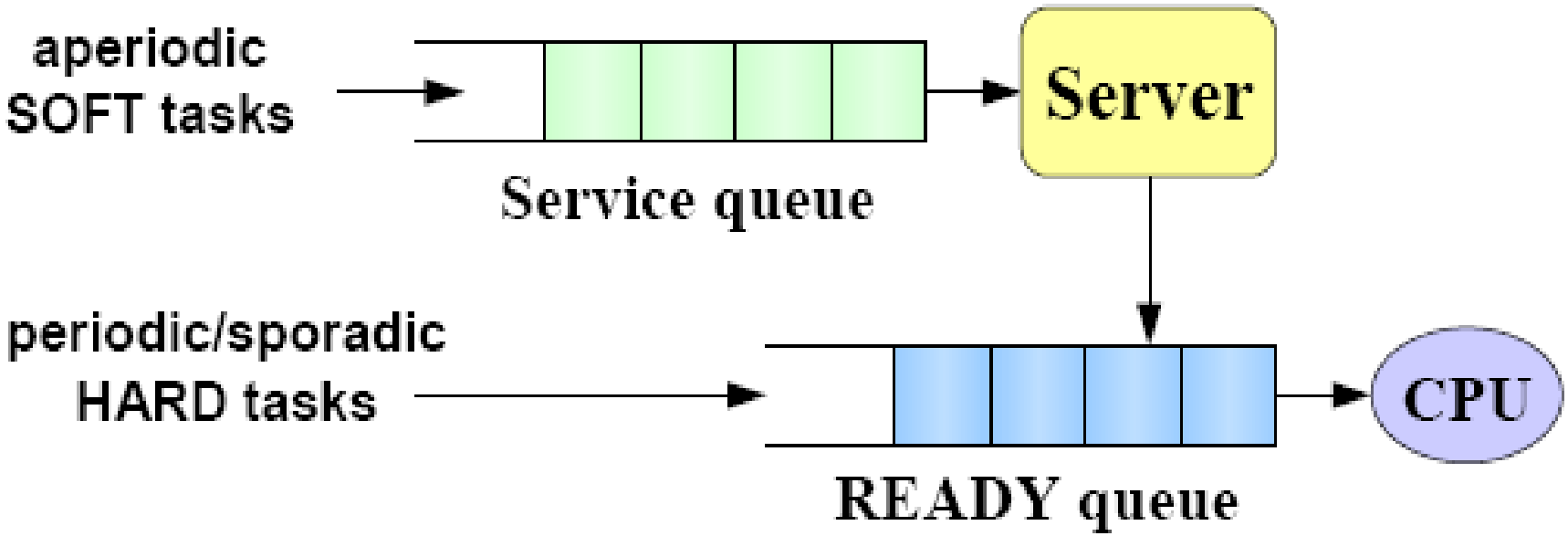
Polling Server

- ***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 and a computation time .
  - 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.

Terms

- A periodic server (p_s, e_s) is defined partially by its period p_s and execution time e_s. The parameter e_s is called the *execution budget* (or simply *budget*) of the server. The ratio $u_s = e_s / p_s$ is the *size* of the server.
- *Backlogged*: whenever the aperiodic job queue is nonempty
- *Eligible* (*i.e., ready*): *when it is backlogged and has budget* (i.e., its budget is nonzero).

Aperiodic service queue



- The server is scheduled as any periodic task.
- Priority ties are broken in favor of the server.
- Aperiodic tasks can be selected using an arbitrary queueing discipline.

Polling Server

Function of polling server (PS)

- At regular intervals equal to T_s , PS becomes active and 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.

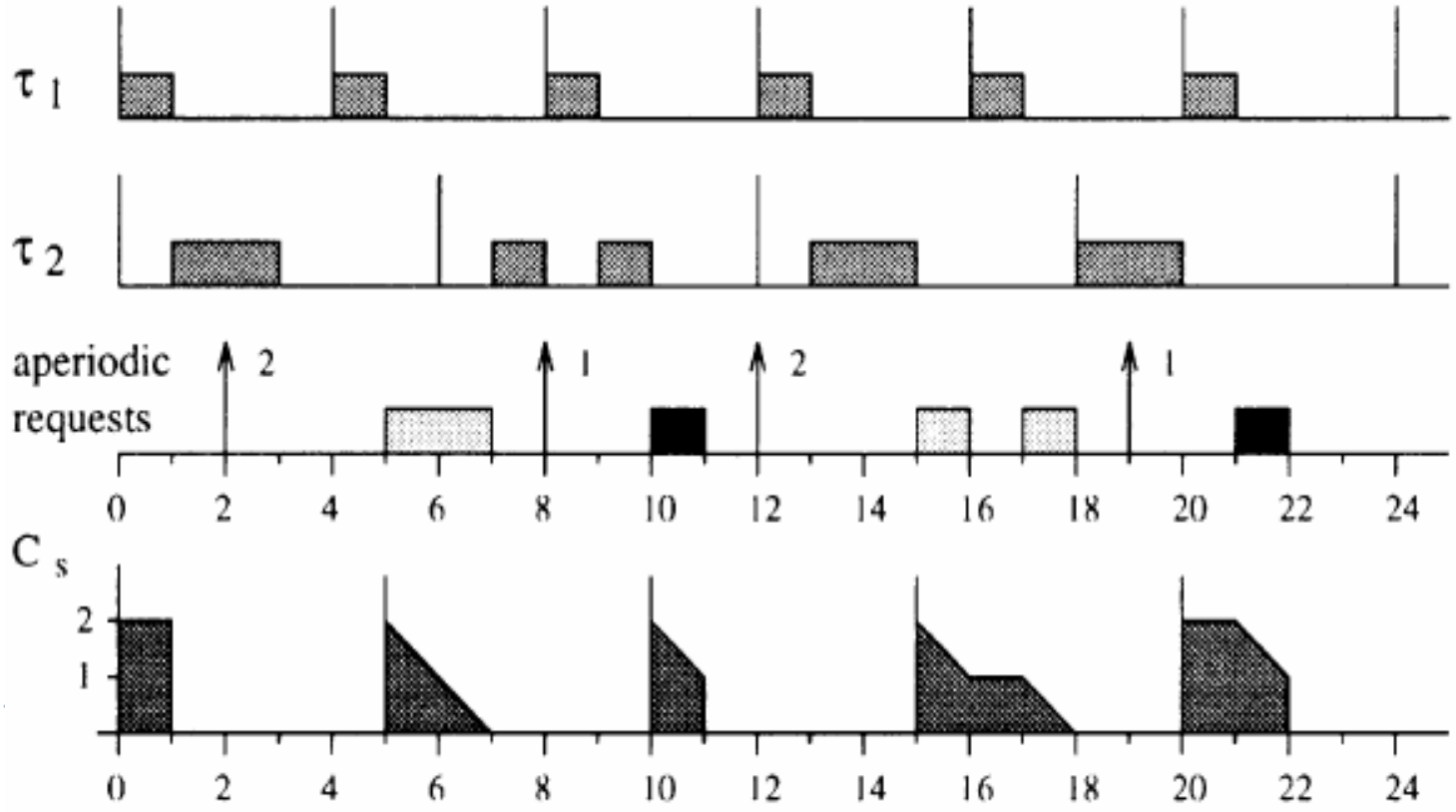
Example

	C_i	T_i
τ_1	1	4
τ_2	2	6

Server

$C_s = 2$

$T_s = 5$



Polling Server Schedulability

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

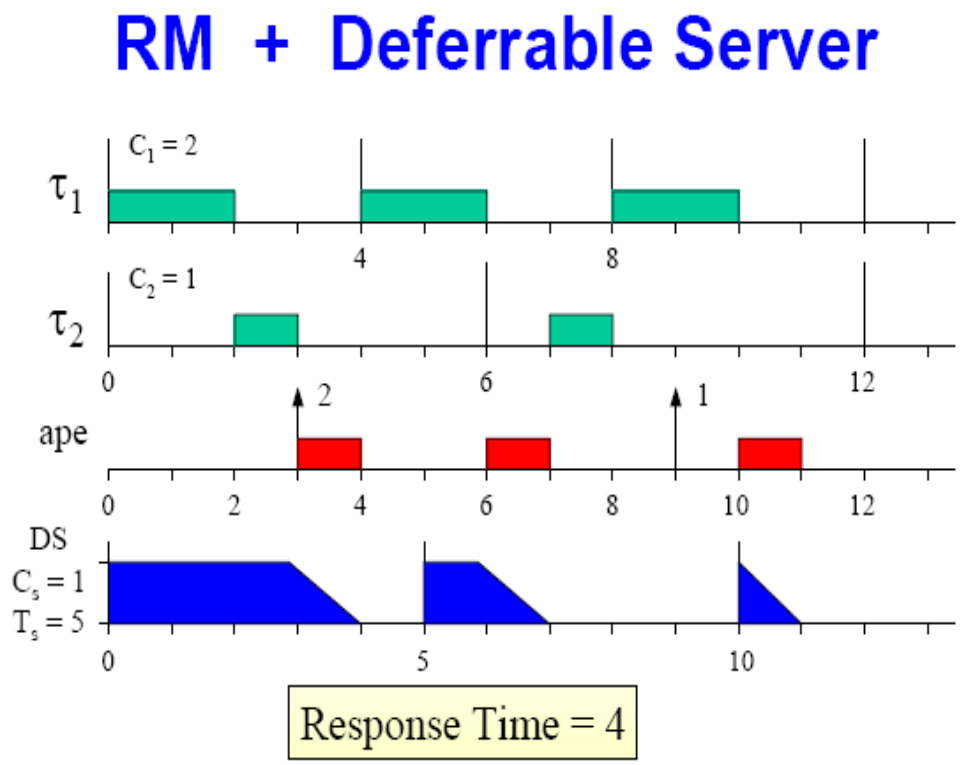
$$\frac{C_s}{T_s} + \sum_{i=1}^{n} \frac{C_i}{T_i} \leq (n+1)\left(2^{1/(n+1)} - 1\right)$$

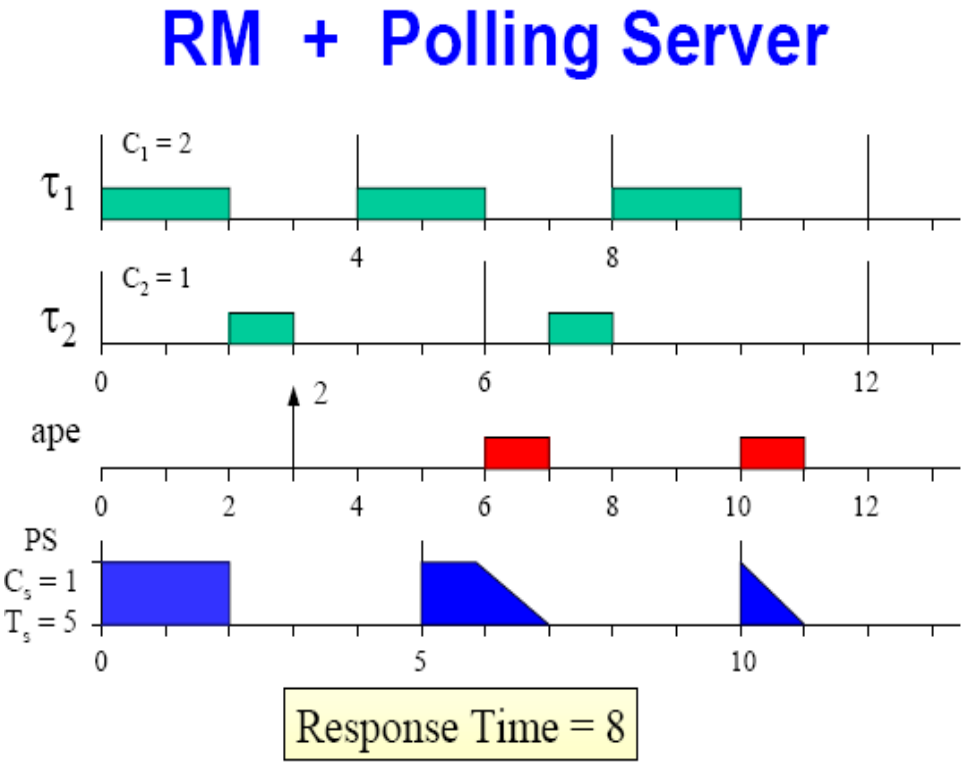
- Again, this test is sufficient but not necessary.

Deferrable Server

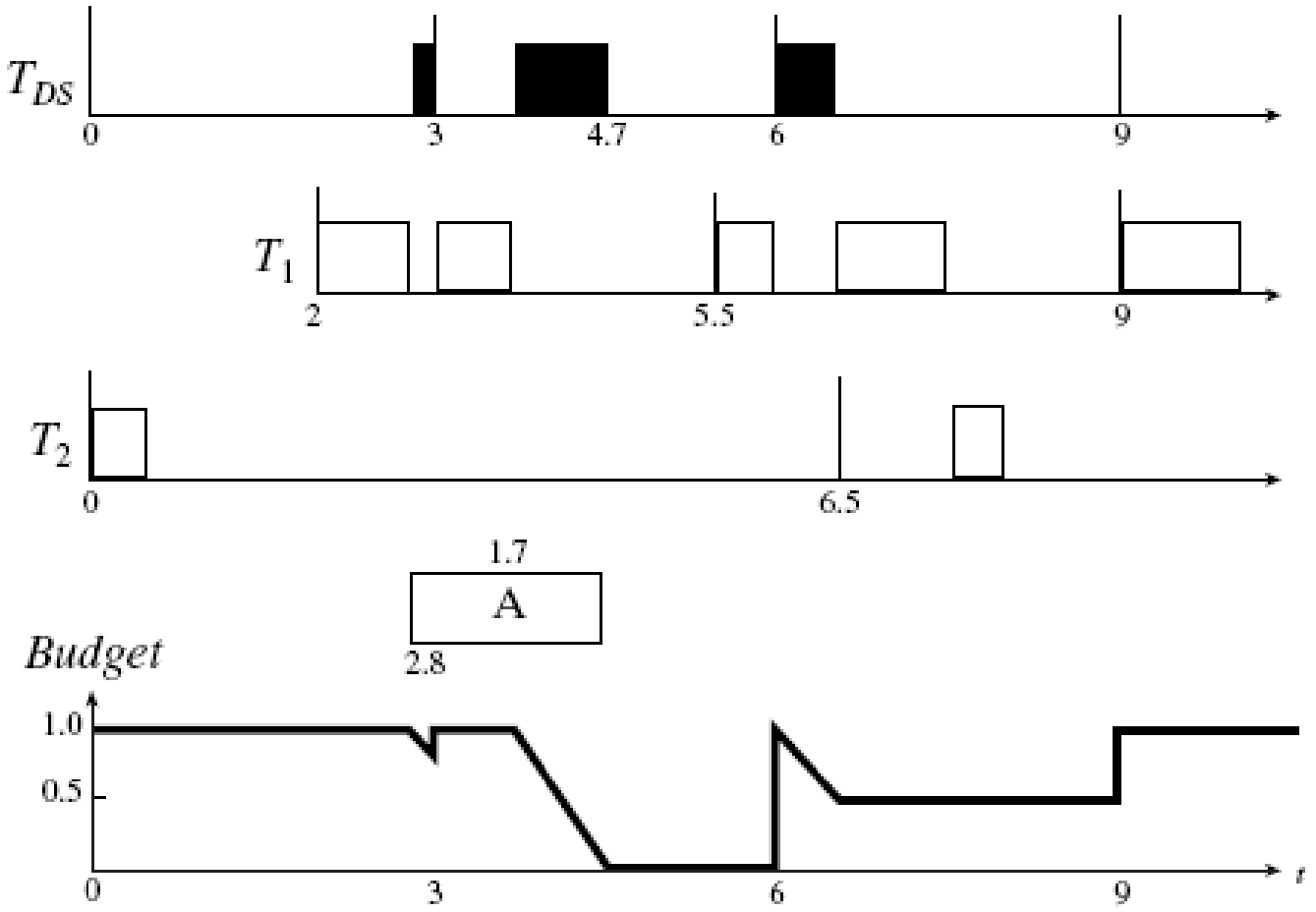
- • Is similar to the PS, but the budget is not discharged if there are no pending requests.
  - *Consumption Rule*
  - The execution budget of the server is consumed at the rate of one per unit time whenever the server executes.
  - *Replenishment Rule*
  - The execution budget of the server is set to *es* at time instants *kpk*, for *k* = 0, 1, 2, ...
- • Keeping the budget improves responsiveness, but decreases the utilization bound.

Deferrable Server





EDF+Deferrable Server

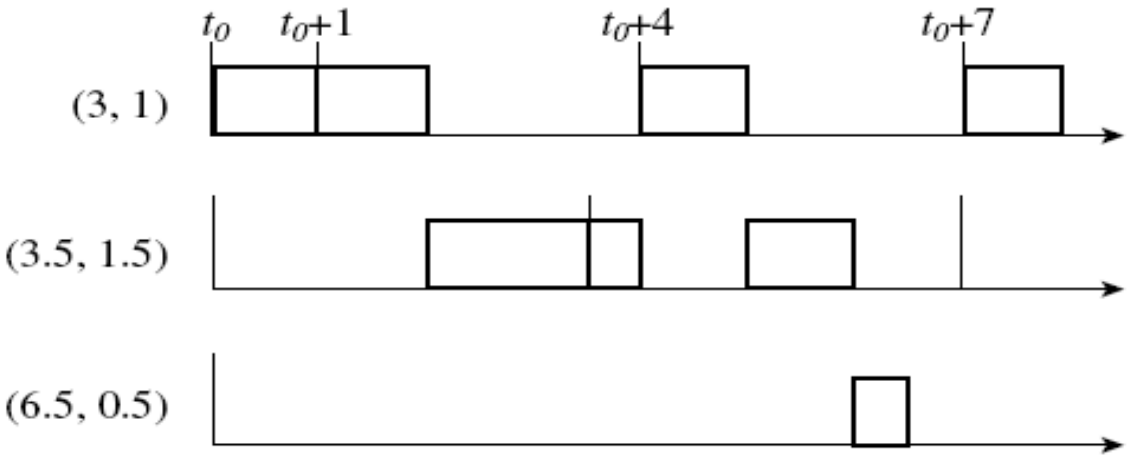


Example illustrating the operations of a deferrable server: *(*T_{DS} *= (*3, 1*)*, *T*1 *= (*2, 3.5, 1.5*)*, and *T*2 *= (*6.5, 0.5*)*.

Deferrable Server

- The responsiveness of the system can be further improved if we combine the use of a deferrable server with background execution.
- This server is scheduled whenever the budget of the deferrable server has been exhausted and none of the periodic tasks is ready for execution.

Schedulability of Fixed-Priority Systems Containing DS



The factor limiting the budget of a deferrable server

THEOREM Consider a system of n independent, preemptable periodic tasks whose periods satisfy the inequalities $p_s < p_1 < p_2 < \cdots < p_n < 2p_s$ and $p_n > p_s + e_s$ and whose relative deadlines are equal to their respective periods. This system is schedulable rate-monotonically with a deferrable server (p_s, e_s) if their total utilization is less than or equal to

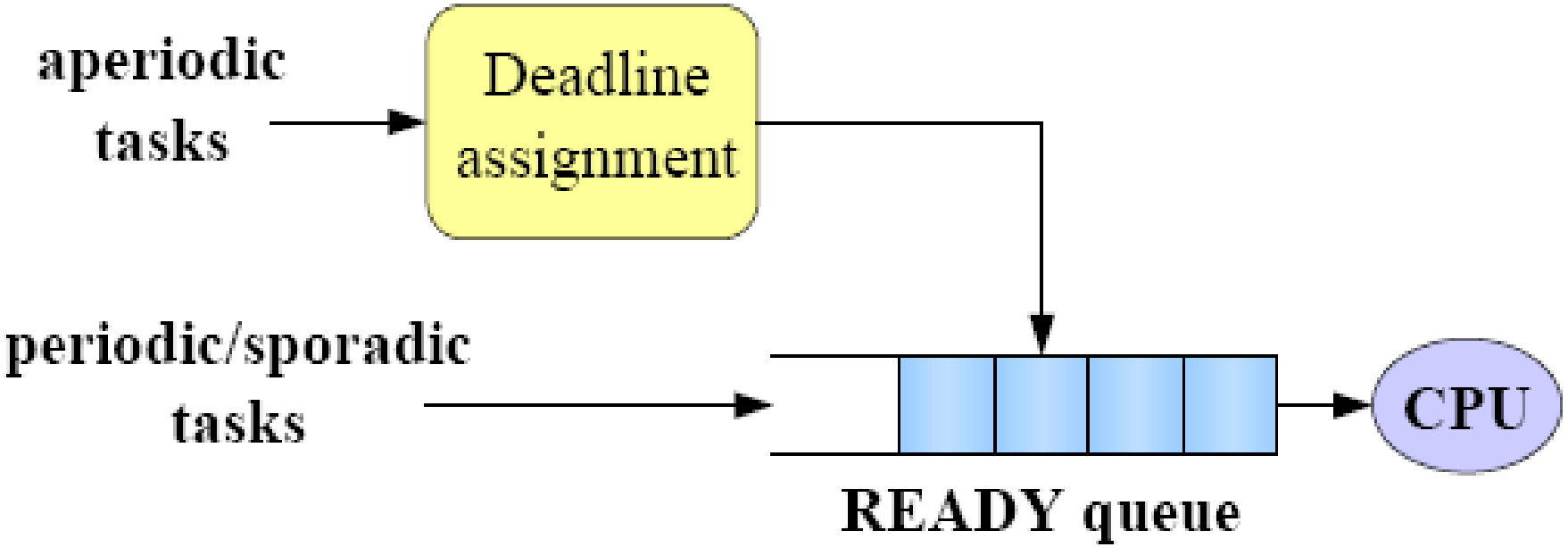
$$U_{RM/DS}(n) = (n-1)\left[\left(\frac{u_s+2}{u_s+1}\right)^{1/(n-1)} - 1\right]$$

where u_s is the utilization e_s/p_s of the server.

Total Bandwidth Server

- • It is a dynamic priority server, used along with EDF.
- • Each aperiodic request is assigned a deadline so that the server demand does not exceed a given bandwidth Us .
- • Aperiodic jobs are inserted in the ready queue and scheduled together with the HARD tasks.

The TBS mechanism



- Deadlines ties are broken in favor of the server.
- Periodic tasks are guaranteed *if and only if*

$$U_p + U_s \leq 1$$

Deadline assignment rule

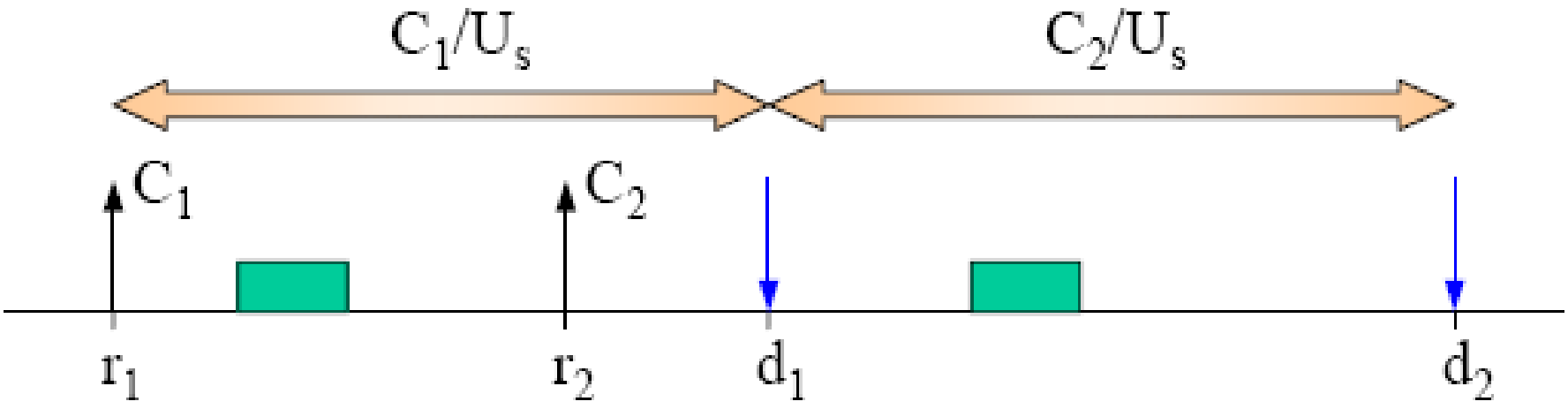
- Deadline has to be assigned not to jeopardize periodic tasks.
- A safe relative deadline is equal to the minimum period that can be assigned to a new periodic task with utilization U_s:

$$U_s = C_k / T_k \quad \Rightarrow \quad T_k = d_k - r_k = C_k / U_s$$

- Hence, the absolute deadline can be set as:

$$d_k = r_k + C_k / U_s$$

Deadline assignment rule



- To keep track of the bandwidth assigned to previous jobs, d_k must be computed as:

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

Replenishment Rules of a Total Bandwidth Server of size $\tilde{u}_s$

R1 Initially, $e_s = 0$ and $d = 0$.

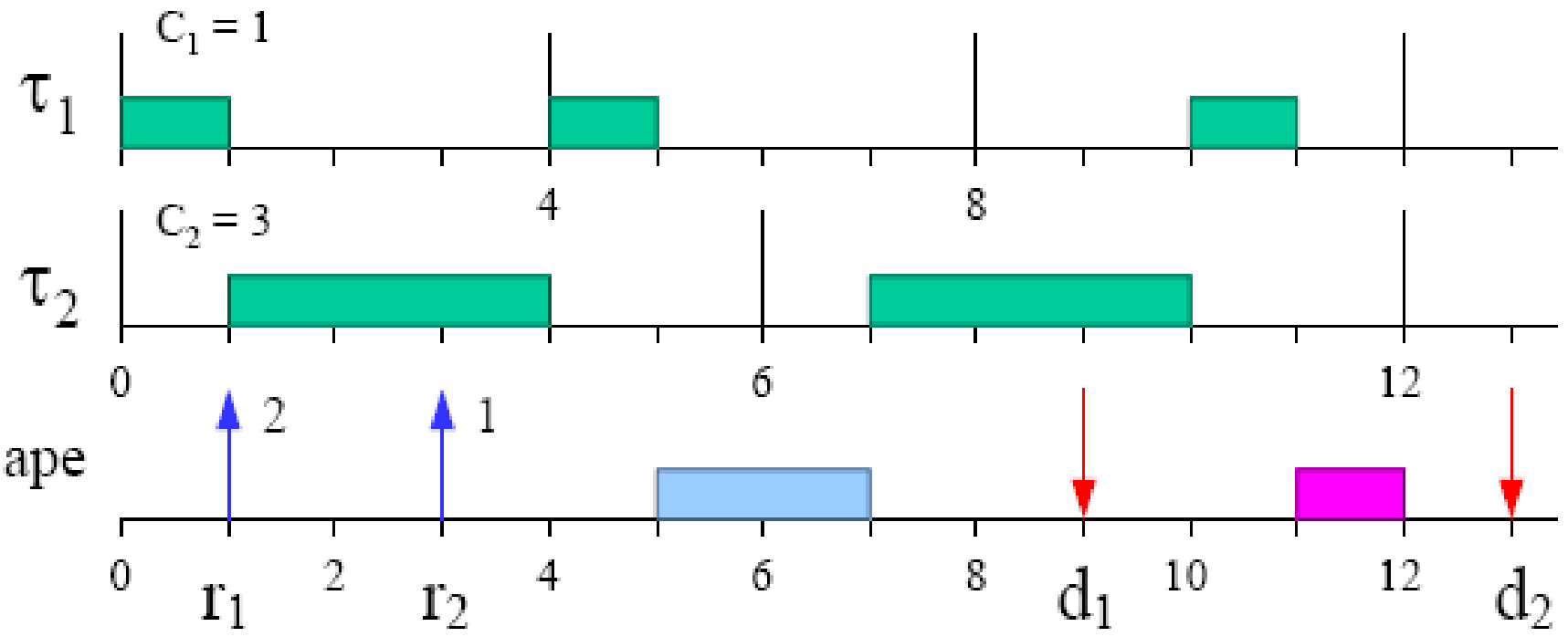
R2 When an aperiodic job with execution time e arrives at time t to an empty aperiodic job queue, set d to $\max(d, t) + e/\tilde{u}_s$ and $e_s = e$.

R3 When the server completes the current aperiodic job, the job is removed from its queue.

(a) If the server is backlogged, the server deadline is set to $d + e/\tilde{u}_s$, and $e_s = e$.

(b) If the server is idle, do nothing.

EDF + TBS schedule

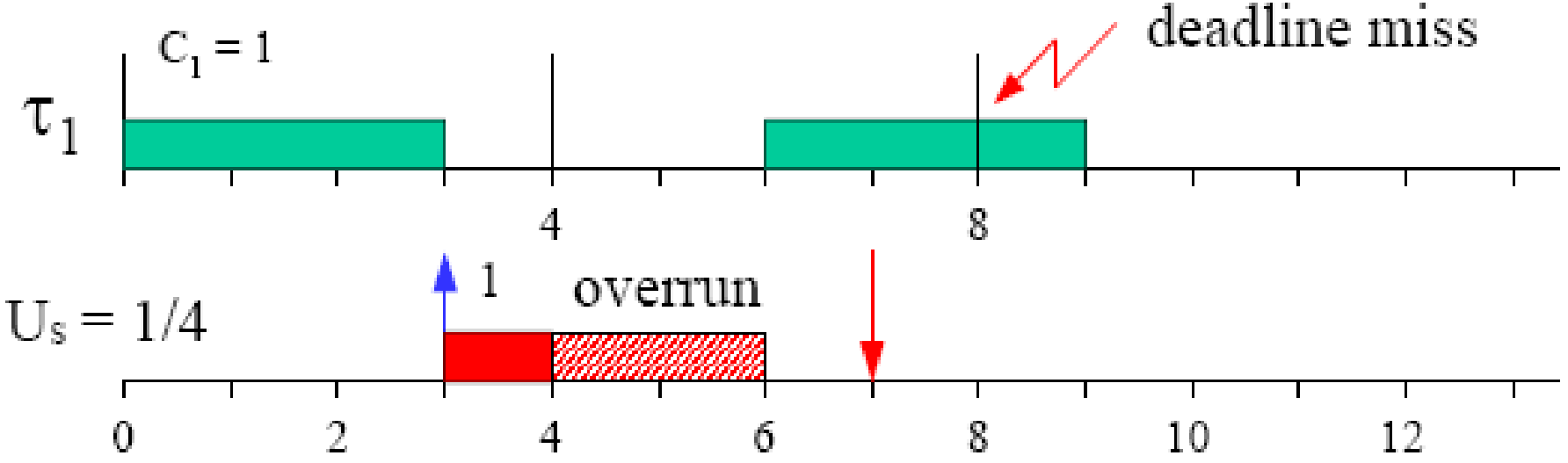


$$U_s = 1 - U_p = 1/4$$

$$\begin{cases} d_1 = r_1 + C_1 / U_s = 1 + 2 \cdot 4 = 9 \\ d_2 = \max(r_2, d_1) + C_2 / U_s = 9 + 1 \cdot 4 = 13 \end{cases}$$

Problems with the TBS

- Without a budget management, there is no protection against execution overruns.
- If a job executes more than expected, hard tasks could miss their deadlines.

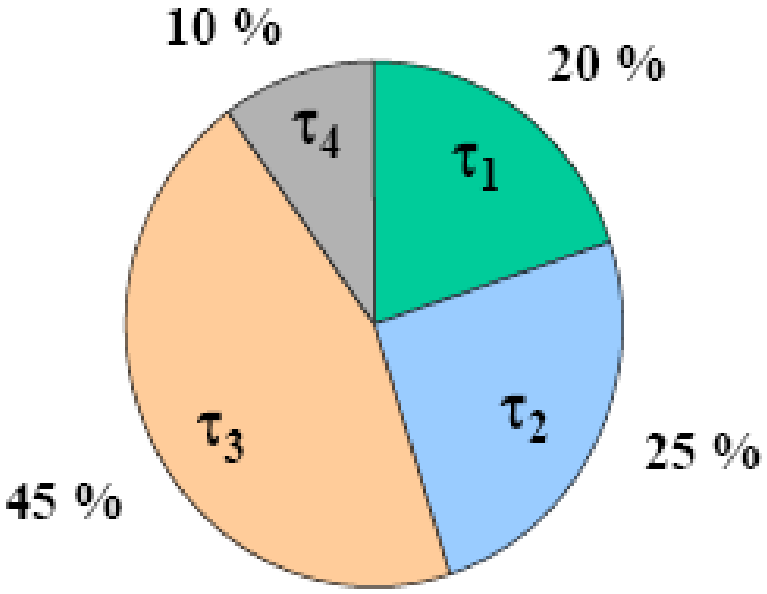


Solution: task isolation

- In the presence of overruns, only the faulty task should be delayed.
- Each task τ_i should not demand more than its declared utilization ($U_i = C_i/T_i$).
- If a task executes more than expected, its priority should be decreased (i.e., its deadline postponed).

Bandwidth partitioning

- Ideally, each task should be assigned a given bandwidth and never demand more.



Questions

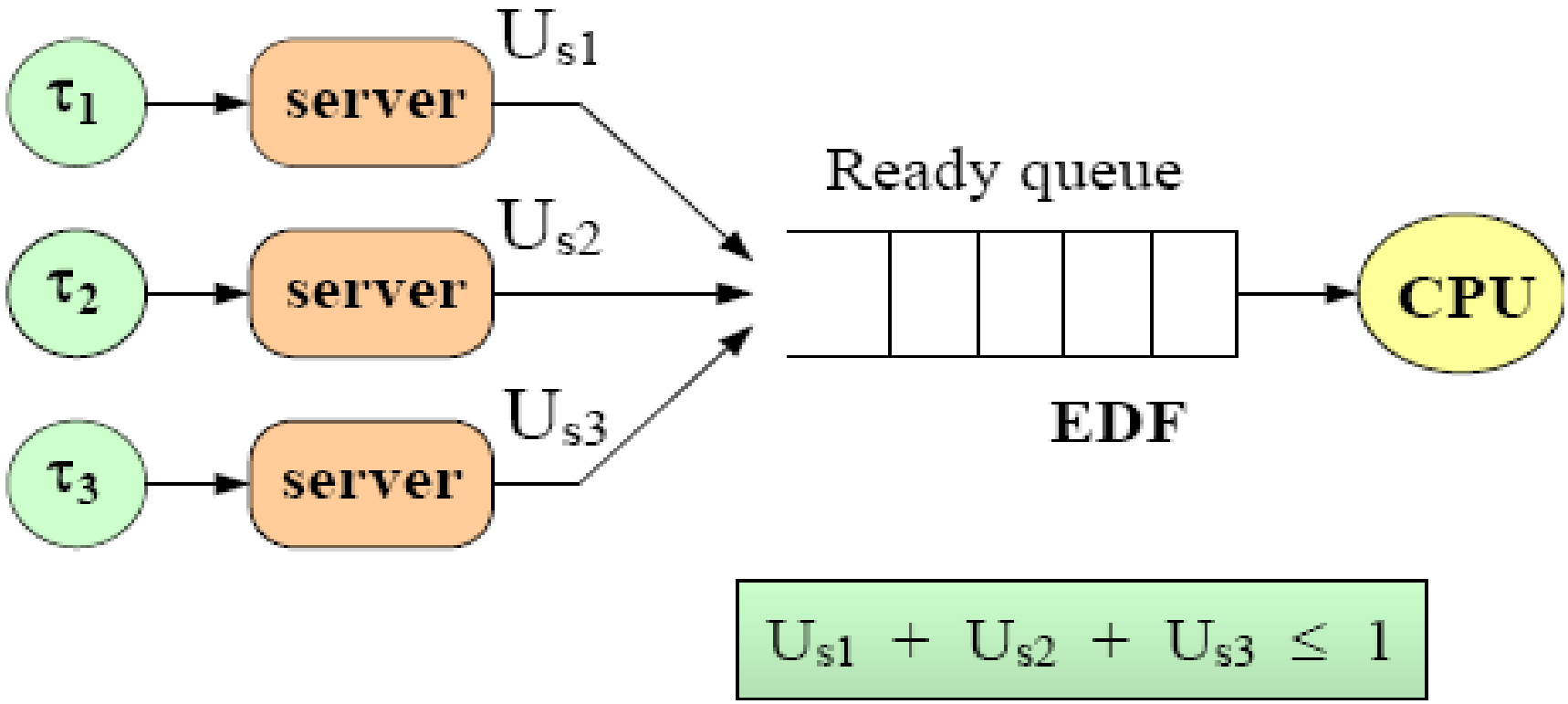
- **What do we do if a task overruns?**
  - Only that task should be delayed.

- **Consequences**
  - if the task is hard => exception
  - if the task is soft => QoS degradation

Achieving isolation

- Isolation among tasks can be achieved through a **bandwidth reservation**.
- Each task is managed by a dedicated server having bandwidth U_s .
- The server assigns priorities (or deadlines) to tasks so that they do not exceed the reserved bandwidth.

Implementation



•**More details in:** *Garder,M.K;Liu,J.W.S. Performance of Algorithms for Scheduling Real-Time Systems with Overrun and Overload. In EMRTS 1999*

Constant Bandwidth Server (CBS)

- It assigns deadlines to tasks as the TBS, but keeps track of job executions through a budget mechanism.
- When the budget is exhausted it is immediately replenished, but the deadline is postponed to keep the demand constant.

Replenishment Rules of a Constant Utilization Server of Size $\tilde{u}_s$

R1 Initially, $e_s = 0$, and $d = 0$.

R2 When an aperiodic job with execution time e arrives at time t to an empty aperiodic job queue,

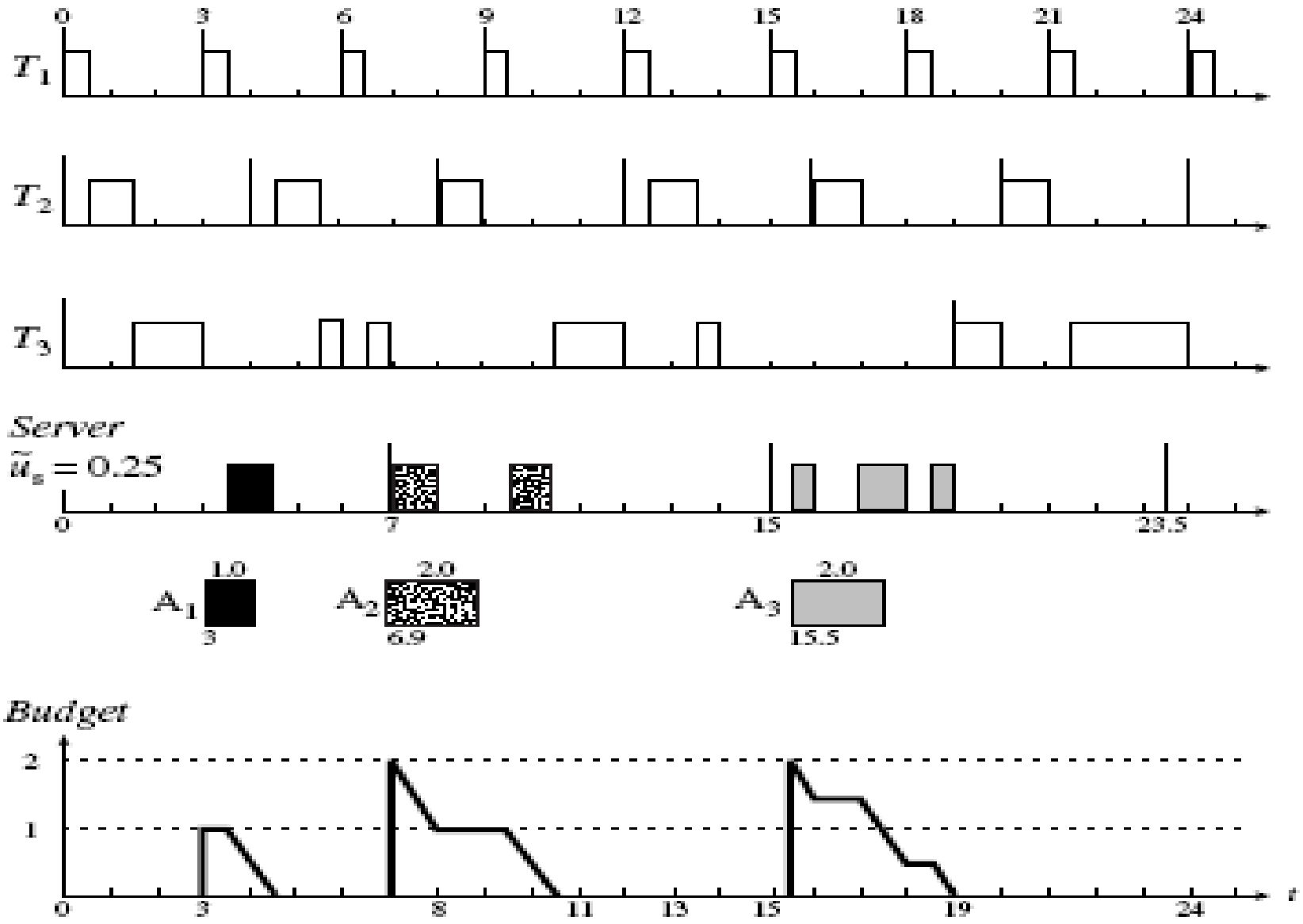
(a) if $t < d$, do nothing;

(b) if $t \geq d$, $d = t + e/\tilde{u}_s$, and $e_s = e$.

R3 At the deadline d of the server,

(a) if the server is backlogged, set the server deadline to $d + e/\tilde{u}_s$ and $e_s = e$;

(b) if the server is idle, do nothing.



$T1 = (4, 0.5)$, $T2 = (4, 1.0)$, $T3 = (19, 4.5)$.

Comparison with TBS&CBS

- For a given set of aperiodic jobs and server size, both kinds of servers have the same sequence of deadlines, but the budget of a total bandwidth server may be replenished earlier than that of a constant utilization server.
- In the above example, this means that the server's budget is replenished at 6.9 and, if *A*3 were to arrive at 14, at 14, and the deadline of the server is 15 and 23.5, respectively, *A*3 would be completed at time 17.5 if it were executed by a total bandwidth server but would be completed at 19 by a constant bandwidth server.

Recommended Readings

- Jane W.S. Liu. *Real-Time Systems*, 2002.
- LUI SHA, et al. *Real Time Scheduling Theory: A Historical Perspective. In Real-Time Systems 2004.*
- Garder,M.K;Liu,J.W.S. *Performance of Algorithms for Scheduling Real-Time Systems with Overrun and Overload. In EMRTS 1999.*