CIS 721 - Real-Time Systems

Lecture 14: Mixing Real-Time and Non-Real-Time in Priority Driven Systems

Mitch Neilsen
neilsen@ksu.edu

Outline

- See Chapter 7 of Liu's text
- We discussed mixing real-time and non-real-time (aperiodic) jobs in static cyclic schedules
- We now address the same issue in priority-driven systems.
- First, we consider two straightforward scheduling algorithms for periodic and aperiodic jobs.
- Then, we look at a class of algorithms called **bandwidth-preserving algorithms** to schedule aperiodic jobs in a real-time system.

Periodic and Aperiodic Tasks

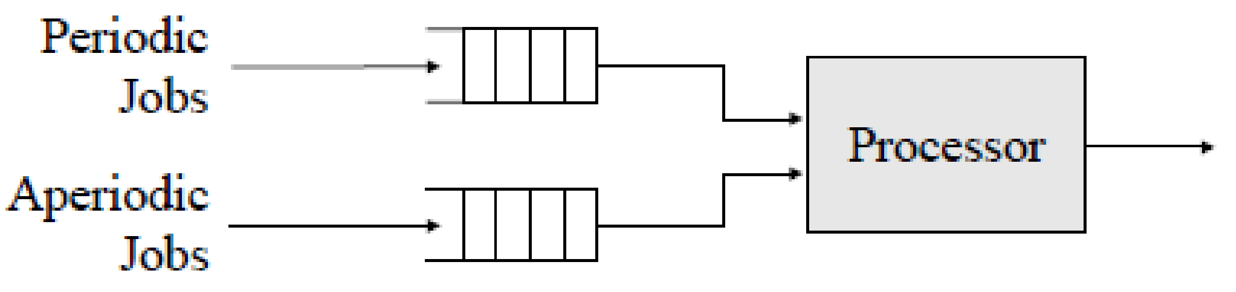
- **Periodic task:** T_i is specified by (φ_i, p_i, e_i, D_i)
- **Aperiodic tasks:** non-real-time
  - Released at arbitrary times.
  - Have no deadline and e_i may be unspecified.
- We assume that periodic and aperiodic tasks are independent of each other.

Correct and Optimal Schedules

- A **correct schedule** never results in a deadline being missed by periodic tasks.
- A **correct scheduling algorithm** only produces correct schedules.
- An **optimal aperiodic job scheduling algorithm** minimizes either
  - the response time of the aperiodic job at the head of the queue, or
  - the average response time of all aperiodic jobs.

Scheduling Mixed Jobs

- We assume there are separate job queues for real-time (periodic) and non-real-time (aperiodic) jobs.
- How do we minimize response time for aperiodic jobs without impacting periodic jobs?



Background Scheduling (BS)

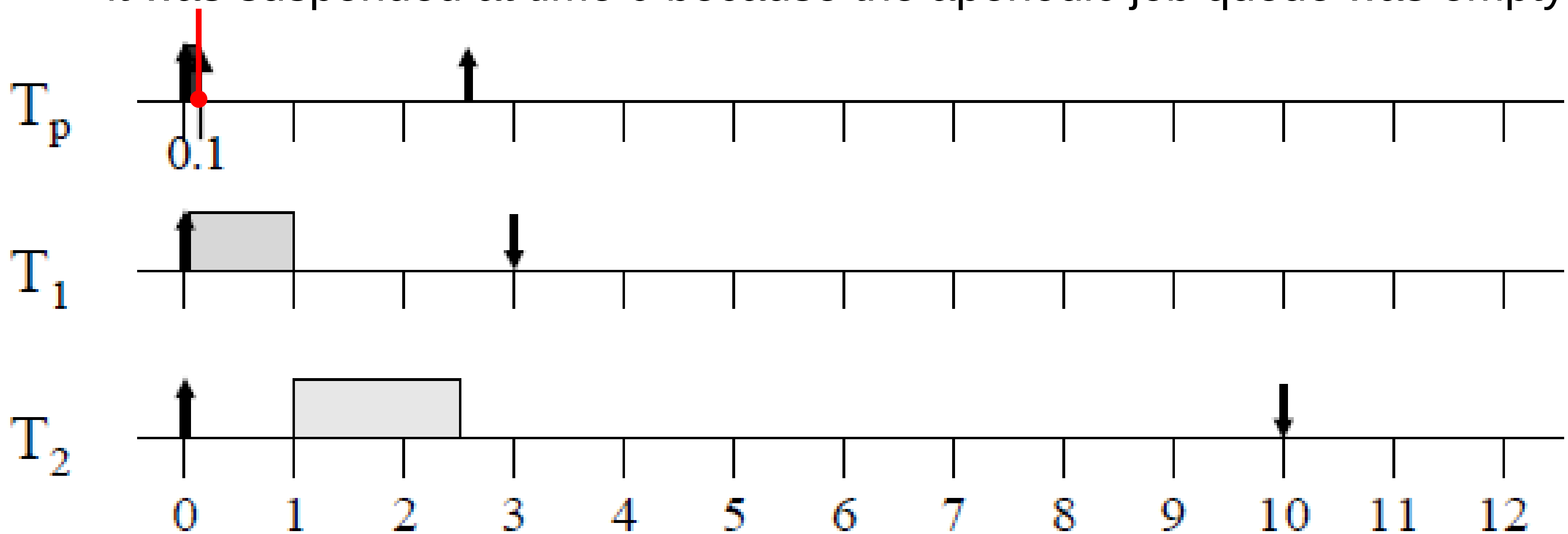
- Periodic jobs are scheduled using any priority-driven scheduling algorithm.
- Aperiodic jobs are scheduled and executed in the background:
  - Aperiodic jobs are executed only when there is no periodic job ready to execute.
  - Simple to implement and always produces correct schedules. **The lowest priority task simply executes jobs from the aperiodic job queue.**
  - We can improve response times without jeopardizing deadlines by using a **slack stealing algorithm** to delay the execution of periodic jobs as long as possible.
    - This is the same thing we did with cyclic executives.
    - However, it is very expensive (in terms of overhead) to implement slack-stealing in priority-driven systems.

Simple Periodic (Polling) Server

- Periodic jobs are scheduled using any priority-driven scheduling algorithm.
- Aperiodic jobs are executed by a special periodic server:
  - The periodic server is a periodic task Tp=(p_s, e_s).
    - e_s is called the execution budget of the server.
    - The ratio $u_s = e_s/p_s$ is the size of the server.
  - Suspends as soon as the aperiodic queue is empty or Tp has executed for e_s time units (which ever comes first).
    - This is called exhausting its execution budget.
  - Once suspended, it cannot execute again until the start of the next period.
    - That is, the execution budget is replenished (reset to e_s time units) at the start of each period.
    - Thus, the start of each period is called the replenishment time for the simple periodic server.

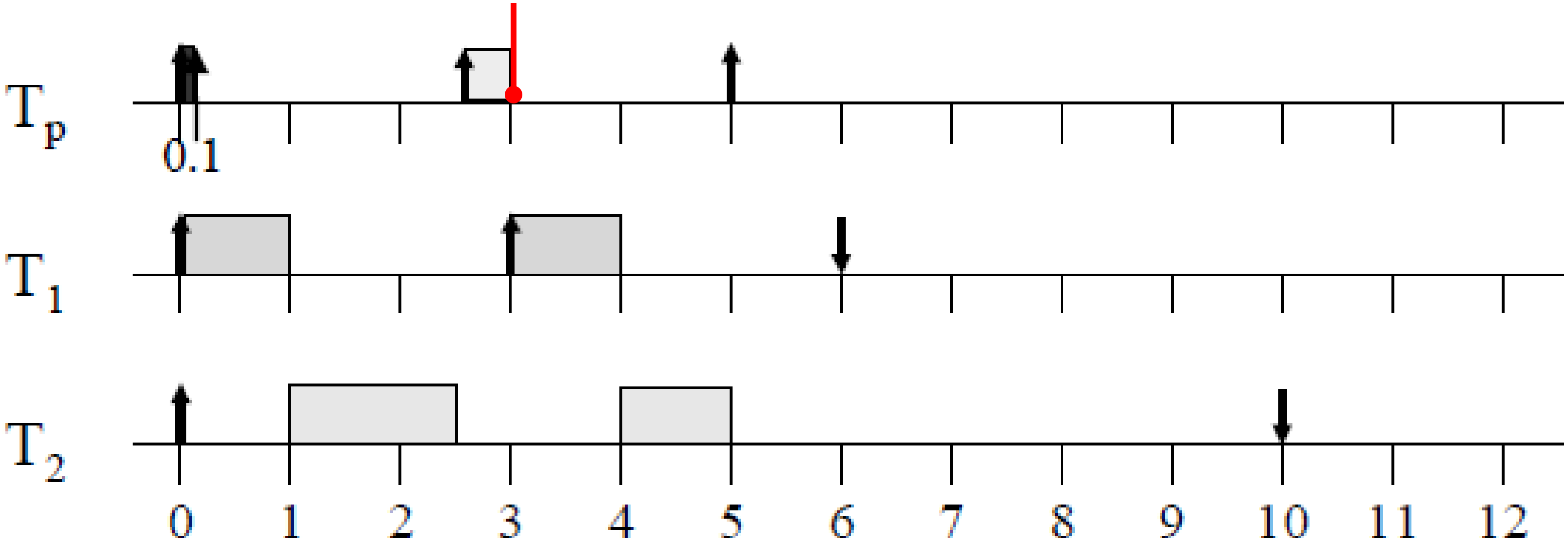
Periodic Server with RM Scheduling

- **Example Schedule:** Two tasks, T1 = (3,1), T2 = (10,4), and a periodic server Tp = (2.5,0.5). Assume an aperiodic job Ja arrives at t = 0.1 with and execution time of e_a = 0.8.
  - The periodic server cannot execute the job that arrives at time 0.1 since it was suspended at time 0 because the aperiodic job queue was empty.



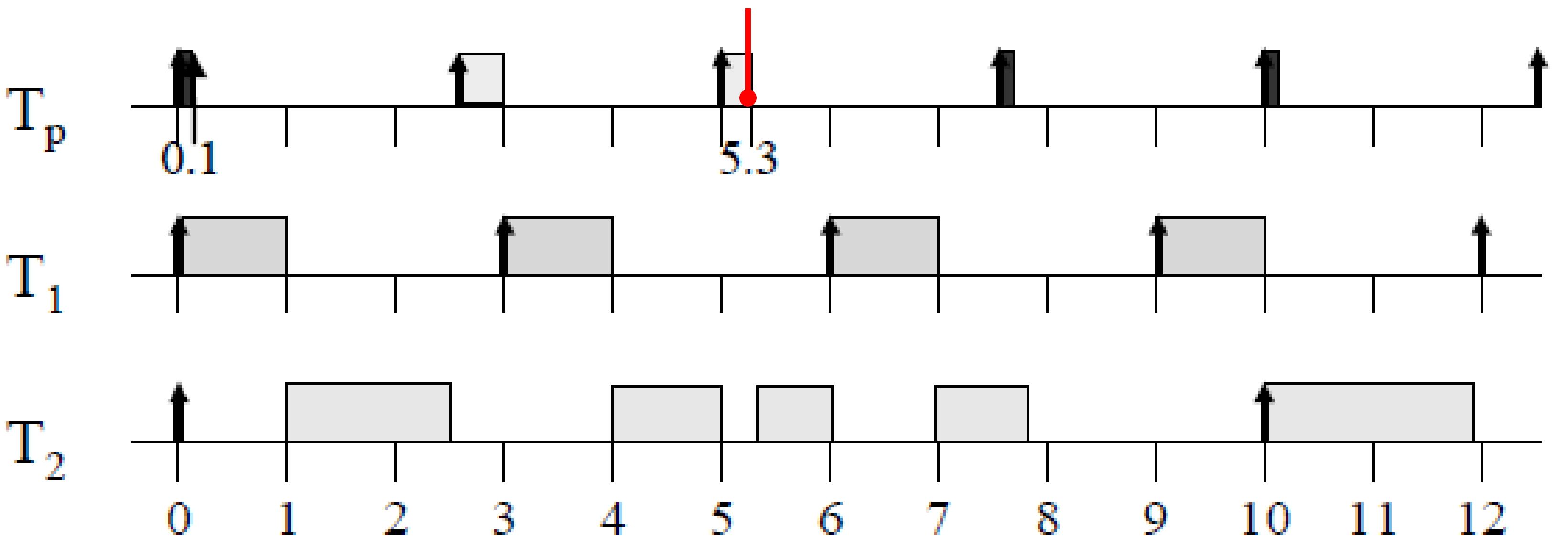
Example (cont.)

- The Periodic Server executes its job Ja starting at 2.5, up until its exhausts its budget at time 3.



Example (cont.)

- It finishes executing in the next period from time 5.0 to 5.3.
- So the response time of Ja is 5.2; that is, from 0.1 to 5.3.



Improving the Periodic Server

- The problem with the periodic server is that it exhausts its execution budget whenever the aperiodic job queue is empty.
  - If an aperiodic job arrives ε time units after the start of the period, it must wait until the start of the next period (p_s – ε time units) before it can begin execution.
- We would like to preserve the execution budget of the polling server and use it later in the period to shorten the response time of aperiodic jobs:
  - **Bandwidth-Preserving Servers** do just this!

Bandwith-Preserving Servers

- More terminology:
  - The periodic server is **backlogged** whenever the aperiodic job queue is nonempty or the server is executing a job.
  - The server is **idle** whenever it is not backlogged.
  - The server is **eligible** for execution when it is backlogged and has an execution budget (greater than zero).
  - When the server executes, it consumes its execution budget at the rate of one time unit per unit of execution.
  - Depending on the type of periodic server, it may also consume all or a portion of its execution budget when it is idle: the simple periodic server consumed all of its execution budget when the server was idle.

Bandwidth-Preserving Servers

- **Bandwidth-preserving servers** differ in their replenishment times and how they preserve their execution budget when idle.
- We assume the scheduler tracks the consumption of the server's execution budget and suspends the server when the budget is exhausted or the server becomes idle.
- The scheduler replenishes the servers execution budget at the appropriate replenishment times, as specified by the type of bandwidth-preserving periodic server.
- The server is only eligible for execution when it is backlogged and its execution budget is non-zero.

Four Bandwidth-Preserving Servers

- Deferrable Servers (1987)
  - Oldest and simplest of the bandwidth-preserving servers.
  - Static-priority algorithms by Lehoczky, Sha, and Strosnider.
  - Deadline-driven algorithms by Ghazalie and Baker (1995).
- Sporadic Servers (1989)
  - Static-priority algorithms by Sprunt, Sha, and Lehoczky.
  - Deadline-driven algorithms by Ghazalie and Baker (1995).
- Total Bandwidth Servers (1994, 1995)
  - Deadline-driven algorithms by Spuri and Buttazzo.
- Constant Utilization Servers (1997)
  - Deadline-driven algorithms by Deng, Liu, and Sun.

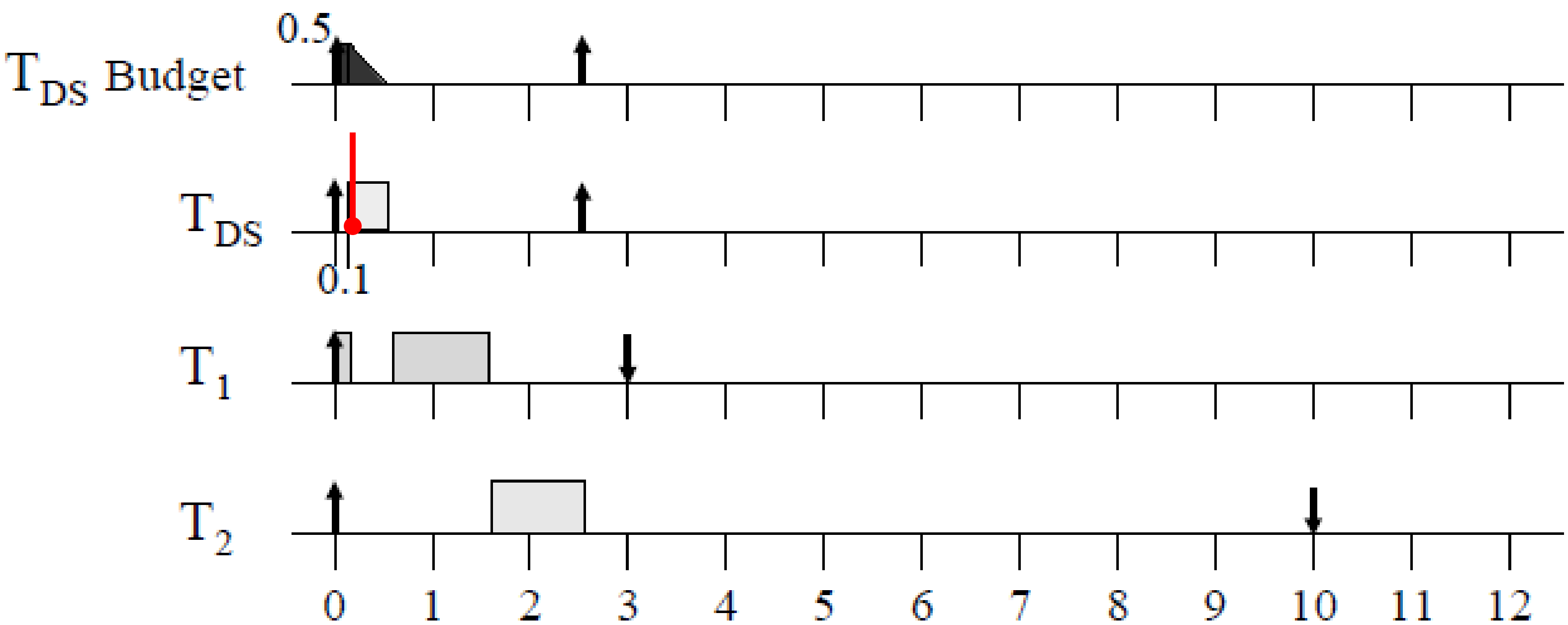
Deferrable Server (DS)

- Let the task $T_{DS} = (p_s, e_s)$ denote a **deferrable server**.
- Consumption Rule:
  - The execution budget is consumed at the rate of one time unit per unit of execution.
- Replenishment Rule:
  - The execution budget is set to e_s at time instants $k*p_s$, for $k>=0$.
  - Note: Unused execution budget cannot be carried over to the next period.
- The scheduler treats the deferrable server as a periodic task that may suspend itself during execution (i.e., when the aperiodic queue is empty).

DS with RM Scheduling Example

Example Schedule: Same two tasks, $T_1 = (3,1)$, $T_2 = (10,4)$, and deferrable server $T_{DS} = (2.5,0.5)$. Assume an aperiodic job J_a arrives at time t = 0.1 with and execution time of $e_a = 0.8$ (again).

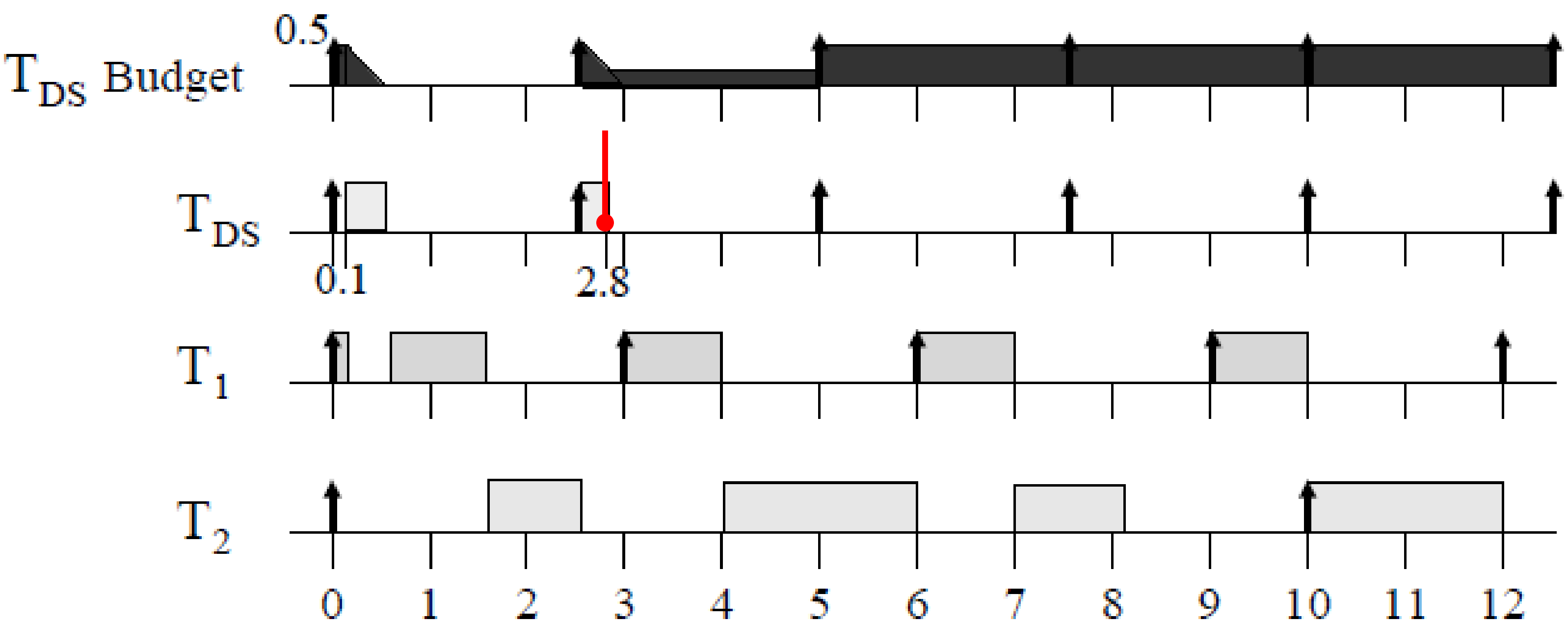
The DS can execute the job that arrives at time 0.1 since it preserved its budget when the aperiodic job queue was empty.



DS with RM Scheduling Example (cont.)

Example Schedule: Same two tasks, $T_1 = (3,1)$, $T_2 = (10,4)$, and deferrable server $T_{DS} = (2.5,0.5)$. Assume an aperiodic job J_a arrives at time $t = 0.1$ with and execution time of $e_a = 0.8$ (again).

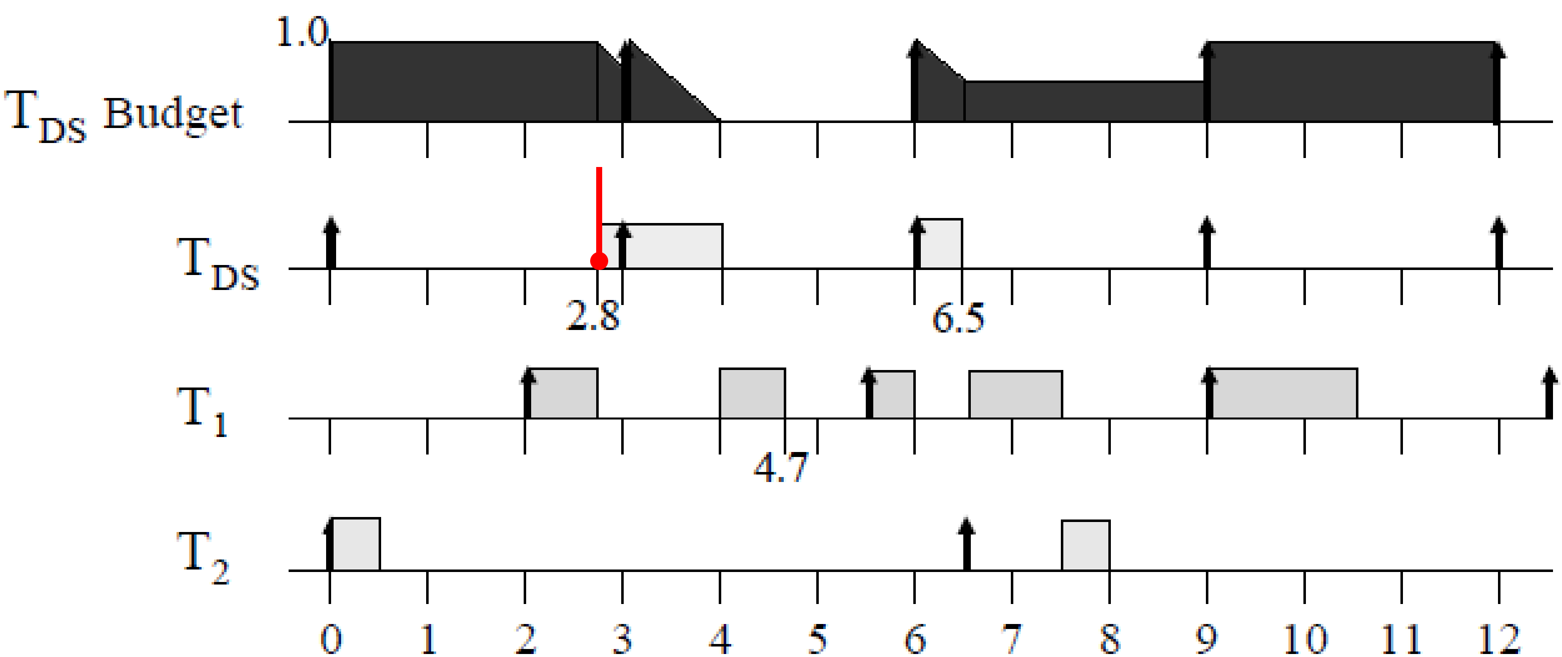
The response time of the aperiodic job J_a is 2.7
It was 5.2 with the simple periodic server.



Another Example

Another Example: Two tasks, $T_1 = (2,3.5,1.5)$, $T_2 = (6.5,0.5)$, and a deferrable server $T_{DS} = (3,1)$. Assume an aperiodic job J_a arrives at time $t = 2.8$ with and execution time of $e_a = 1.7$.

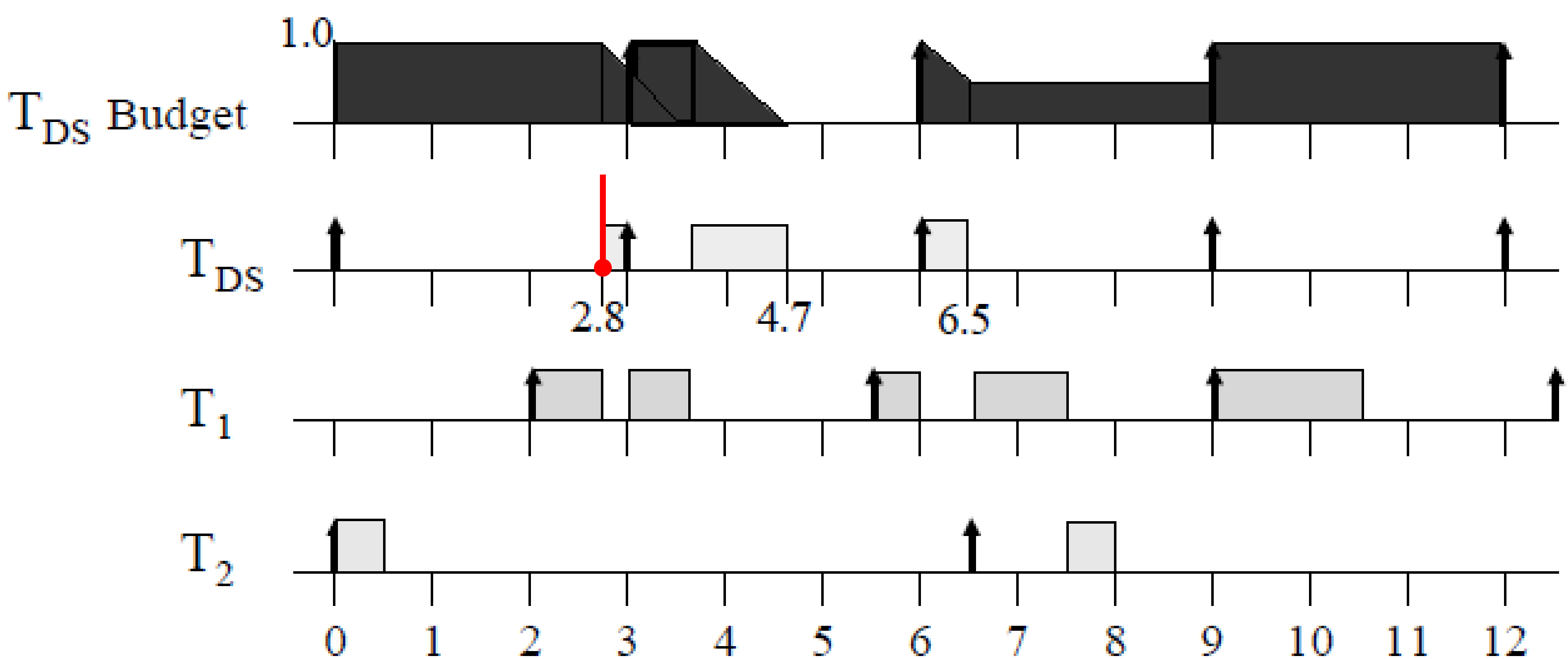
The response time of the aperiodic job J_a is 3.7.



DS with EDF Scheduling Example

Same Task Set: Two tasks, $T_1 = (2, 3.5, 1.5)$, $T_2 = (6.5, 0.5)$, and a deferrable server $T_{DS} = (3, 1)$. Assume an aperiodic job J_a arrives at time t = 2.8 with and execution time of $e_a = 1.7$.

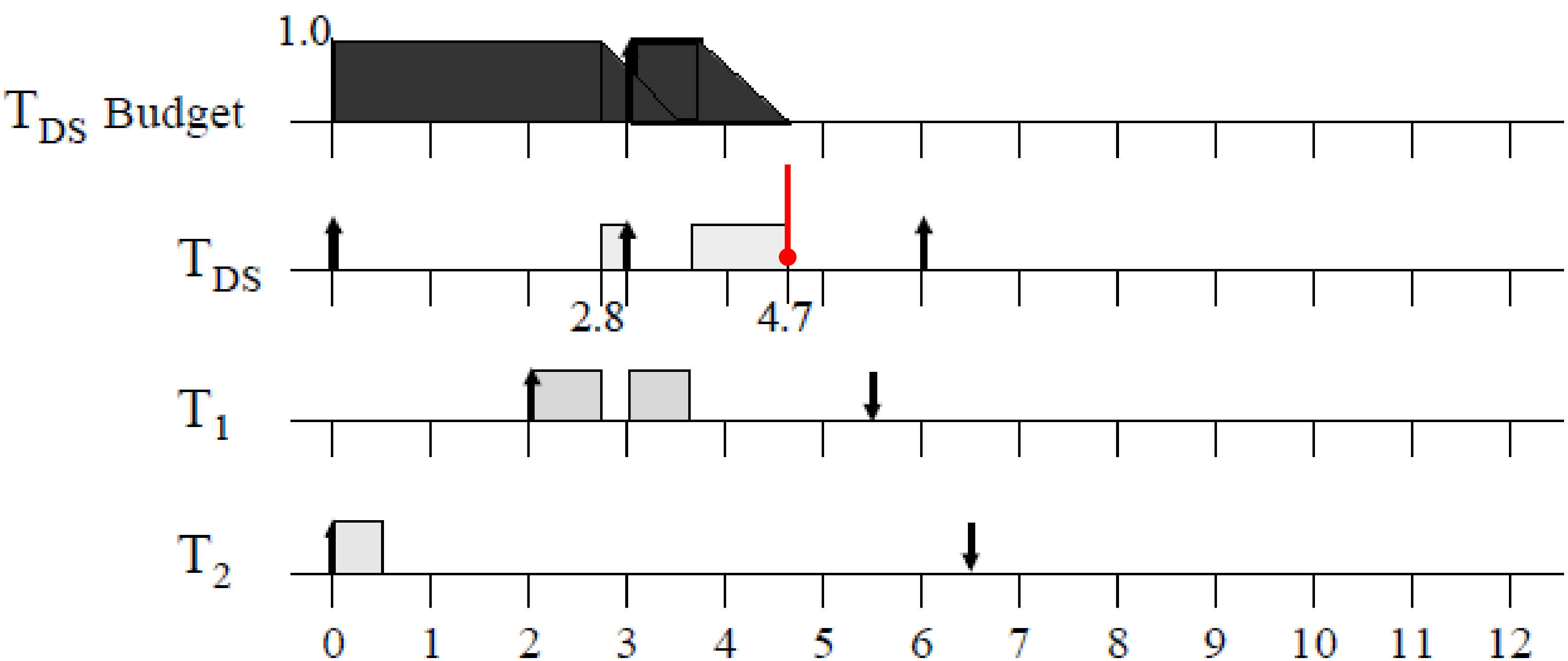
The response time of the aperiodic job J_a is still 3.7.



DS with EDF vs Background Scheduling

<u>Same Task Set:</u> Two tasks, $T_1 = (2,3.5,1.5)$, $T_2 = (6.5,0.5)$, and $T_{DS} = (3,1)$ with background scheduling. Assume an aperiodic job J_a arrives at time t = 2.8 with and execution time of $e_a = 1.7$.

The DS exhausts its budget at time 4.7...

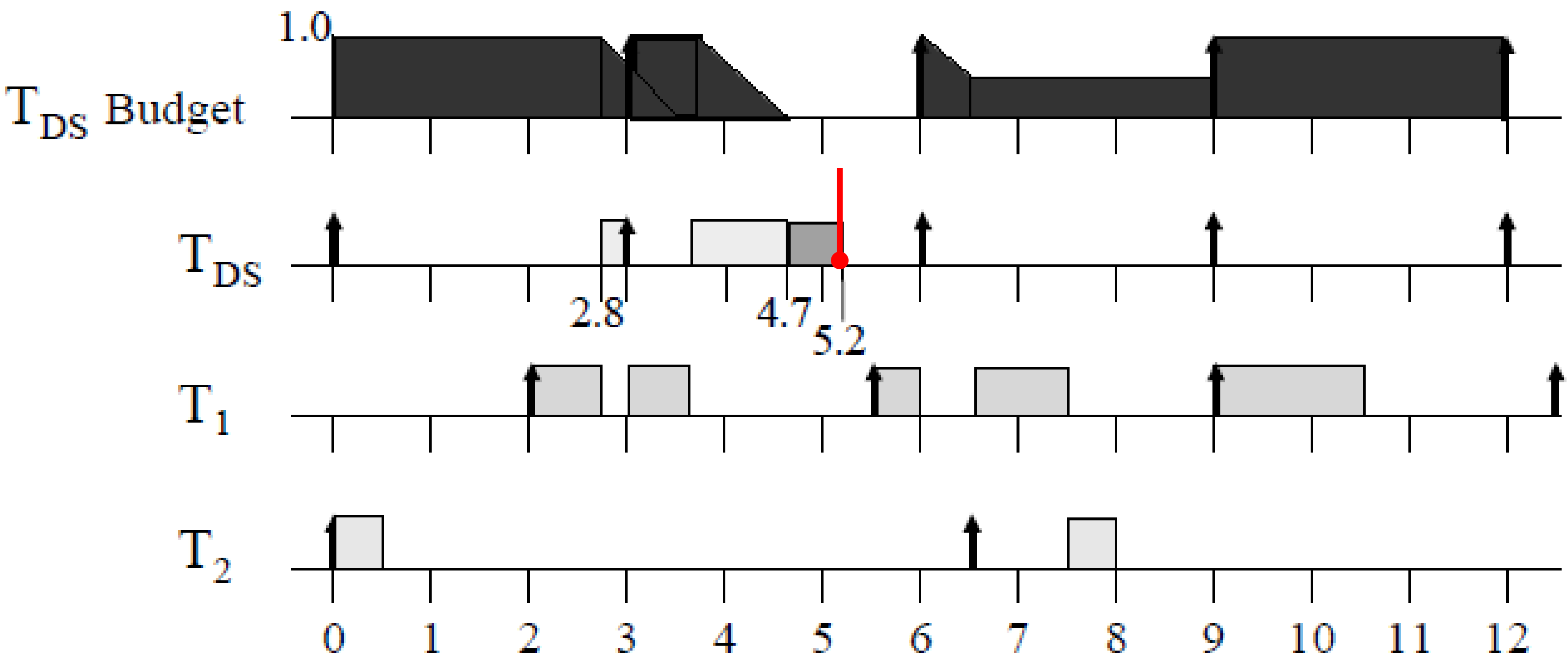


DS with EDF vs **Background Scheduling**

[recall, Background Scheduling = schedule when no periodic tasks are ready]

<u>Same Task Set:</u> Two tasks, $T_1 = (2,3.5,1.5)$, $T_2 = (6.5,0.5)$, and $T_{DS} = (3,1)$ with background scheduling. Assume an aperiodic job J_a arrives at time $t = 2.8$ with and execution time of $e_a = 1.7$.

However, using background scheduling, the response time of the aperiodic job J_a is reduced to 2.4.



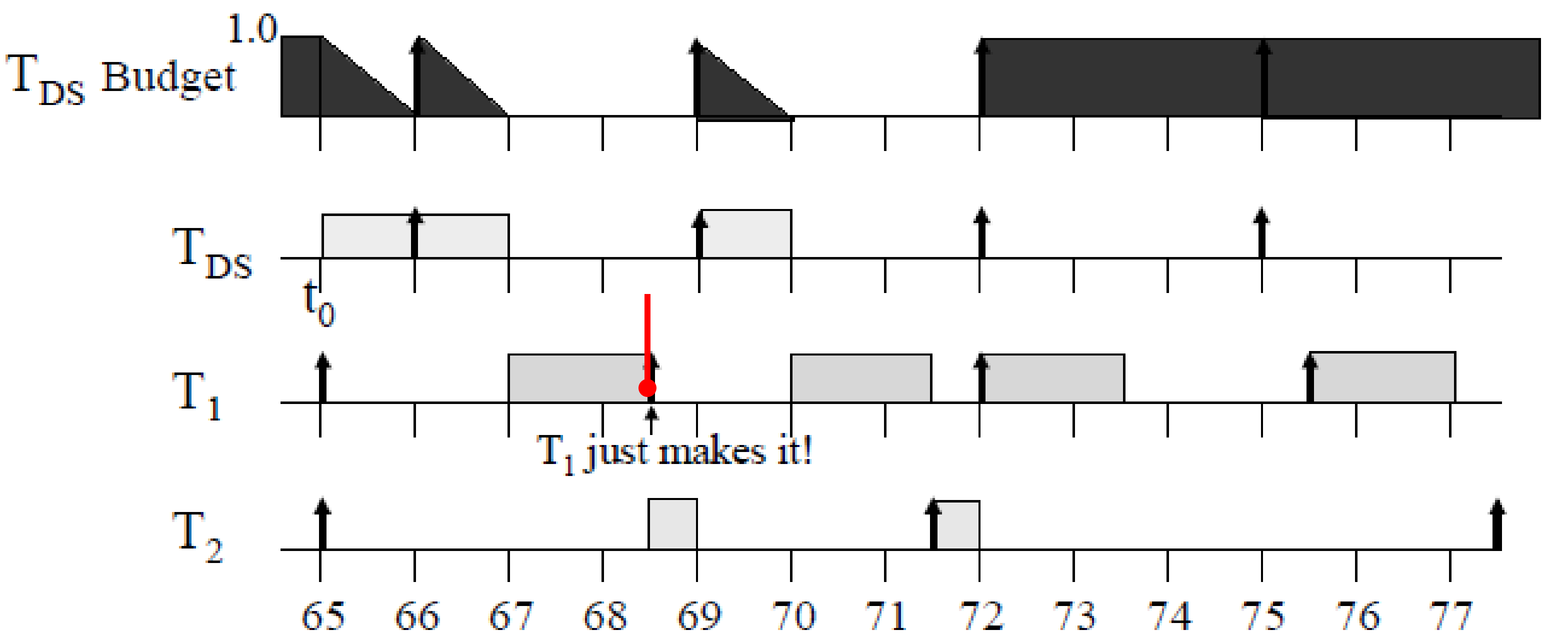
DS with Background Scheduling

- We can also combine background scheduling of the deferrable server with RM.
  - For the deferrable server example task set, the response time doesn’t change. Why?
- Why complicate things by adding background scheduling of the deferrable server?
- Why not just give the deferrable scheduler a larger execution budget? See the next slide!

DS with RM Scheduling Revisited

Modified Example: Same two tasks, $T_1 = (2,3.5,1.5)$, $T_2 = (6.5,0.5)$, and deferrable server $T_{DS} = (3,1)$. Assume an aperiodic job J_a arrives at time $t_0 = 65$ with and execution time of $e_a = 3$.

A larger execution budget for T_{DS} would result in T_1 missing a deadline.
Time $t_0 = 65$ is a critical instant for this task set.



Schedulability and DS

- There are no known necessary and sufficient schedulability conditions for task sets that contain a DS with arbitrary priority. We will see why shortly.
- However, we can extend TDA and Generalized TDA to yield necessary and sufficient schedulability tests when the DS is the highest priority task in a periodic (real world sporadic) task set.
- We start with a critical instant lemma for systems with a DS.

Critical Instants in a Fixed Priority System with a Deferrable Server (DS)

Lemma 7-1: [Lehoczky, Sha, and Strosnider] In a fixed-priority system in which the relative deadline of every independent, preemptable periodic task is no greater than its period and there is a deferrable server (p_s, e_s) with the highest priority among all tasks, a critical instant of every periodic task T_i occurs at time t_0 when all of the following are true.

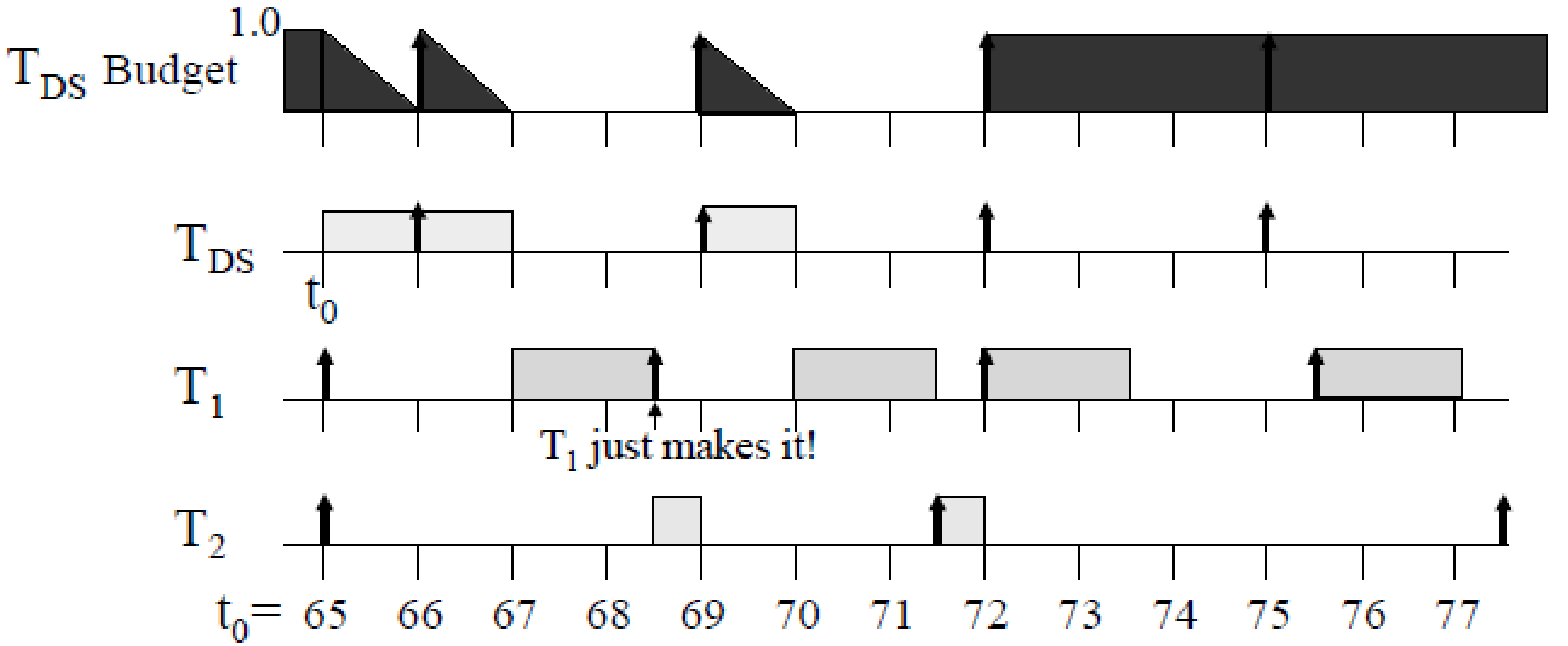
1. One of its jobs $J_{i,c}$ is released at t_0.
2. A job in every higher-priority task is release at the same time.
3. The budget of the server is e_s at t_0, one or more aperiodic jobs are released at t_0, and they keep the server backlogged hereafter.
4. The next replenishment time of the server is $t_0 + e_s$.

Observations on Lemma 7.1

- The Proof of Lemma 7.1 is a straightforward extension of the proof we gave for Theorem 6.5. Convince yourself of this!
- Note: We are not saying that $T_{DS}, T_1, \ldots, T_i$ will all necessarily release jobs at the same time, but if this does happen, we are claiming that the time of release will be a critical instant for Ti.
- We can use the critical instant t_0, defined by Lemma 7.1, to derive necessary and sufficient conditions for the schedulability of a task set when the DS has highest priority.
- First, lets take a look at a processor demand anomaly created by the bandwidth preserving DS.

Observations on Lemma 7.1 (cont.)

All four conditions of Lemma 7.1 hold in the last example:



Notice that the processor demand created by the DS in an interval from [65,68.5] is twice what it would be if it were an ordinary periodic task! This is because we preserve the bandwidth of the DS.

Response Time Analysis with a DS

Observation: Cases (1) and (2) of Lemma 7.1 define a critical instant for any fixed-priority task set. When cases (3) and (4) of Lemma 7.1 are true, the processor demand created by the DS in an interval of length *t* can be at most

$$e_s + \left\lceil \frac{t - e_s}{p_s} \right\rceil e_s \qquad (*)$$

Thus, TDA and Generalized TDA with blocking terms can be extended to systems with a DS that executes at the highest priority. The TDA function becomes:

$$\boxed{w_i(t) = e_i + b_i + e_s + \left\lceil \frac{t - e_s}{p_s} \right\rceil e_s + \sum_{k=1}^{i-1} \left\lceil \frac{t}{p_k} \right\rceil \cdot e_k \quad \text{for } 0 < t \le \min(D_i, p_i)}$$

DS with Highest Fixed Priority

- When the DS is the highest priority process in a fixed-priority system:
  - It may be able to execute an extra e_s time units more than a normal periodic task in the feasible interval of task T_i, as expressed in Equation (*) and the modified $w_i(t)$.
- Thus, Response Time Analysis, using the modified $w_i(t)$ provides a necessary and sufficient condition for fixed-priority systems with one DS executing at the **highest** priority.

DS with Arbitrary Fixed Priority

- When the DS is not the highest priority process:
  - It may not be able to execute the extra e_s time units expressed in Equation (*) and the modified $w_i(t)$.
  - However, the time-demand function of a task Ti with lower priority than an arbitrary-priority DS is bounded from above by the modified $w_i(t)$.
- Thus, Response Time Analysis provides a sufficient (but not necessary) condition for fixed-priority systems with one **arbitrary**-priority DS.

Multiple Arbitrary Fixed-Priority DS

We may want to differentiate aperiodic jobs by executing them at different priorities. To do this, we use multiple DS with different priorities and task parameters ($p_{s,k}$,$e_{s,k}$).

The TDA and Generalized TDA with blocking terms can be further extended to these systems. Specifically, the time demand function $w_i(t)$ of a periodic task T_i with a lower priority than *m* DS becomes:

$$\boxed{w_i(t) = e_i + b_i + \sum_{k=1}^{m}\left(1 + \left\lceil \frac{t - e_{s,k}}{p_{s,k}} \right\rceil\right) e_{s,k} + \sum_{k=1}^{i-1}\left\lceil \frac{t}{p_k} \right\rceil \cdot e_k \quad \text{for } 0 < t \le \min(D_i, p_i)}$$

Schedulable Utilization with Fixed-Priority DS

- We now look at utilization-based scheduling tests for fixed-priority systems with one DS.
- There are no known necessary and sufficient schedulable utilization conditions for fixedpriority systems with a DS.
- However, there does exist a sufficient condition for RM when the DS has the shortest period plus some other conditions...

RM Schedulable Utilization with a DS

<u>Theorem 7.2</u> Consider a system of n independent, preemptable periodic tasks whose periods satisfy the inequalities $p_s < p_1 < p_2 < \ldots < p_n < 2p_s$ 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.

<u>Proof:</u> Similar to Thm 6.11 and left as an exercise!

Note that this is only a **<u>sufficient</u>** schedulability test.

RM Schedulable Utilization with a DS and Arbitrary Periods

Observe: If $p_i < p_s$, then task T_i is unaffected by the DS. If $p_i > p_s$, then it may be blocked an extra e_s time units in a feasible interval.

Theorem: Consider a system of *n* independent, preemptable periodic tasks whose relative deadlines are equal to their respective periods. Task T_i with $p_i > p_s$ is schedulable rate monotonically with a deferrable server (p_s, e_s) if

$$U_i + u_s + \frac{e_s + b_i}{p_i} \le U_{RM}(i+1)$$

where u_s is the utilization e_s/p_s of the server, U_i is the total utilization of the tasks $T_1 \ldots T_i$, and b_i is the blocking time encountered by task T_i from lower priority tasks.

Schedulability of a Deadline-Driven System with a DS

- In fixed-priority systems, the DS behaves like a periodic task (p_s, e_s) except that it could execute an extra amount of time (at most e_s time units) in the feasible interval of any lower priority job.
- In a deadline-driven system, the DS can execute at most e_s time units in the feasible interval of any job (under certain conditions).
- We present a sufficient (but not necessary) schedulability condition for the EDF algorithm.
- First, a bound on the processor demand created by a DS.

Bounding the Demand of a DS in an EDF System

- An interval $(a, b]$ is **post-idle** if either $a = 0$, or if no job with a deadline in the interval $(a+1, b]$ executes in the interval $(a–1, a]$.
  - The implication of this definition is that all jobs with deadlines in $(a+1, b]$ are “idle” during the interval $(a–1, a]$ in the sense that all jobs released before time a with deadlines in $(a+1, b]$ have completed execution before time a (*i.e.*, either the processor is idle in $(a–1, a]$ or a job with deadline at time a or after time b executes in $(a–1, a]$).
- The following lemma gives us a simple upper bound for the processor demand in a post-idle interval of length L.

Maximum Demand of a DS

Lemma: The maximum demand $w_{DS}(L)$ of a DS= (p_s, e_s) during a post-idle interval of length L in an EDF scheduled system of *n* independent, preemptable periodic tasks is bounded such that

$$w_{DS}(L) \leq u_s(L + p_s - e_s)$$

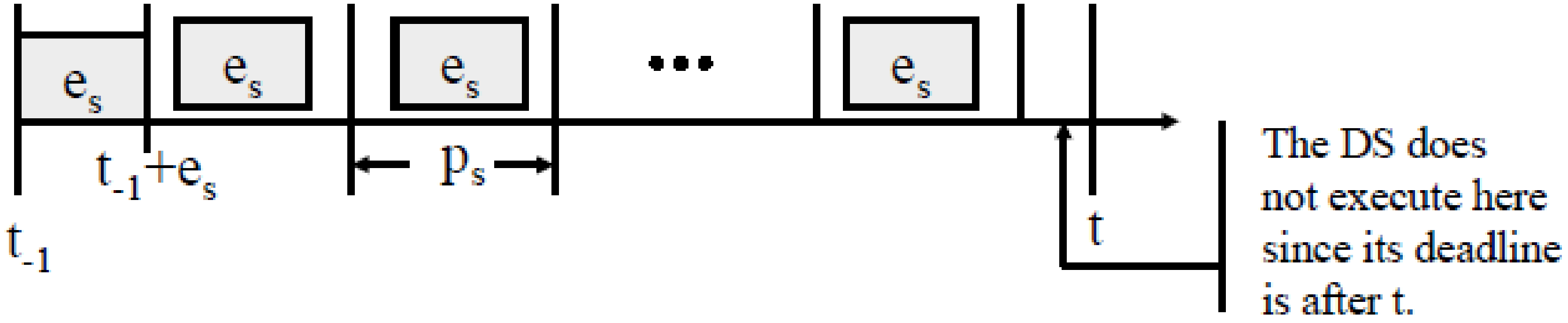
where u_s is the utilization e_s/p_s of the server.

Proof: Let $(t_{-1}, t]$ be a post-idle interval. The maximum demand for DS occurs when

1. At time t_{-1}, its budget is equal to e_s and the server's deadline (and budget replenished time) is $t_{-1} + e_s$.
2. One or more aperiodic jobs arrive at t_{-1} and the DS is backlogged until at least time t.
3. The server's deadline $t_{-1} + e_s$ is earlier than the deadlines of all the periodic jobs that are ready for execution in the interval $(t_{-1}, t_{-1} + e_s]$.

Proof (cont.)

Maximum demand created by DS in a post-idle interval under EDF:



Observe that under these conditions, the maximum demand created by DS in the post-idle interval $(t_{-1},t]$ is at most

$$e_s + \left\lfloor \frac{t-(t_{-1}+e_s)}{p_s} \right\rfloor e_s = e_s + \left\lfloor \frac{t-t_{-1}-e_s}{p_s} \right\rfloor e_s$$

Proof (cont.)

Thus, $$w_{DS}(t-t_{-1}) \le e_s + \left\lfloor \frac{t-t_{-1}-e_s}{p_s} \right\rfloor e_s \le e_s + \frac{t-t_{-1}-e_s}{p_s} e_s$$

$$= \frac{p_s}{p_s} e_s + \frac{t-t_{-1}-e_s}{p_s} e_s = p_s u_s + (t-t_{-1}-e_s) u_s$$

$$= u_s (p_s + (t-t_{-1}-e_s))$$

$$= u_s (t-t_{-1}+p_s-e_s)$$

Since $(t_{-1}, t]$ is a post-idle interval of length $L = t - t_{-1}$,

$$w_{DS}(L) \le u_s (L + p_s - e_s).$$

Schedulability with a DS

- Combining this result with Theorem 6.5, we obtain a Theorem 7.3 by Ghazalie and Baker:

Theorem 7.3: A periodic task T_i in a system of n independent, preemptable periodic tasks is schedulable with a DS= (p_s, e_s) according to the EDF algorithm if

$$\sum_{k=1}^{n} \frac{e_k}{\min(D_k, p_k)} + u_s\left(1 + \frac{p_s - e_s}{D_i}\right) \leq 1 \qquad (7.5)$$

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

Proof of Theorem 7.3

Suppose Equation (7.5) holds for task T_i but a deadline is missed. Let t_d be the earliest point in time at which a deadline is missed and t_{-1} be the start of last post-idle interval that includes time t_d. Thus, a deadline is missed in the post-idle interval $(t_{-1}, t_d]$.

From Theorem 6.2 and the previous lemma, the demand in this interval is at most

$$\sum_{k=1}^{n} \left\lfloor \frac{t_d - t_i}{\min(D_k, p_k)} \right\rfloor e_k + u_s (t_d - t_i + p_s - e_s)$$

Because a deadline is missed at t_d, demand over $(t_{-1}, t_d]$ exceeds $t_d - t_{-1}$. Thus, we have

$$t_d - t_i < \sum_{k=1}^{n} \left\lfloor \frac{t_d - t_i}{\min(D_k, p_k)} \right\rfloor e_k + u_s (t_d - t_i + p_s - e_s)$$

$$\leq \sum_{k=1}^{n} \frac{t_d - t_i}{\min(D_k, p_k)} e_k + u_s (t_d - t_i + p_s - e_s)$$

Proof (cont.)

Dividing both sides by $(t_d - t_{-1})$, we get

$$1 < \sum_{k=1}^{n} \frac{e_k}{\min(D_k, p_k)} + \frac{u_s(t_d - t_i + p_s - e_s)}{t_d - t_i}$$

$$= \sum_{k=1}^{n} \frac{e_k}{\min(D_k, p_k)} + u_s\left(1 + \frac{p_s - e_s}{t_d - t_i}\right)$$

$$\leq \sum_{k=1}^{n} \frac{e_k}{\min(D_k, p_k)} + u_s\left(1 + \frac{p_s - e_s}{D_i}\right)$$

Since $D_i \leq (t_d - t_{-1})$.

This contradicts our assumption that Equation (7.5) holds.

Multiple DS

We may want to differentiate aperiodic jobs by executing them at different priorities. To do this in a deadline-driven system, we (again) use multiple DS with different priorities and task parameters $(p_{s,k}, e_{s,k})$.

Corollary: A periodic task T_i in a system of n independent, preemptable periodic tasks is schedulable with m a DS= $(p_{s,k}, e_{s,k})$ according to the EDF algorithm if

$$\sum_{k=1}^{n} \frac{e_k}{\min(D_k, p_k)} + \sum_{k=1}^{m} u_{s,k} \left(1 + \frac{p_{s,k} - e_{s,k}}{D_i} \right) \leq 1$$

where $u_{s,k}$ is the utilization $e_{s,k}/p_{s,k}$ of server k.

The proof is left as an exercise.

DS Summary

- In both fixed-priority and deadline-driven systems, we see that the DS behaves like a periodic task with parameters (p_s, e_s) except it may execute an additional amount of time in the feasible interval of any lower priority job.
- This is because, the bandwidth-preserving conditions result in a scheduling algorithm that is non-work-conserving with respect to a normal periodic task.

Sporadic Servers

- Sporadic Servers (SS) were designed to overcome the blocking time a DS may impose on lower priority jobs.
- All sporadic servers are bandwidth preserving, but the consumption and replenishment rules ensure that a SS, specified a $T_S = (p_s, e_s)$ never creates more demand than a periodic ("real-world" sporadic) task with the same task parameters.
- Thus, schedulability of a system with a SS is determined exactly as a system without a SS.