CIS 721 - Real-Time Systems

Lecture 15: Mixing Real-Time and Non-Real-Time in Priority Driven Systems (cont.)

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

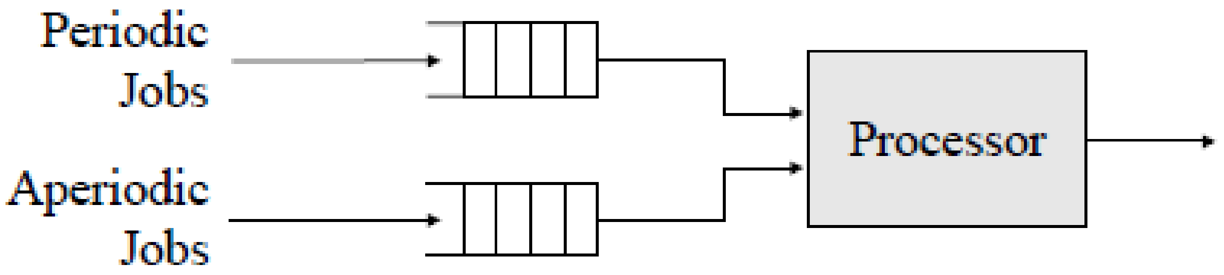
- **Periodic task:** T_i is specified by (φ_i, p_i, e_i, D_i)
- **Aperiodic tasks:** non-real-time
  - Released at arbitrary times.
  - Have no deadline and e_i is unspecified.
- We assume that periodic and aperiodic tasks are independent of each other.

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)

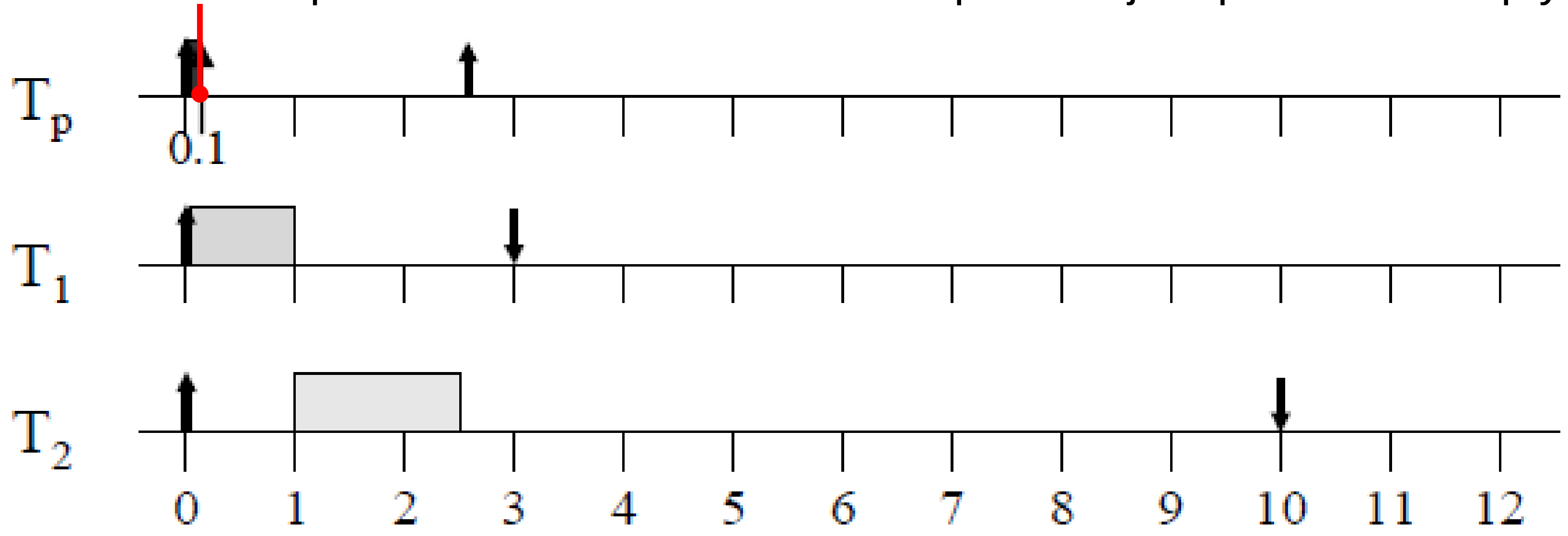
- Periodic jobs are scheduled using any priority-driven scheduling algorithm.
- Aperiodic jobs are scheduled and executed in the background:
  - Aperiodic jobs are executed only when there is no periodic job ready to execute.
  - Simple to implement and always produces correct schedules. **The lowest 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.

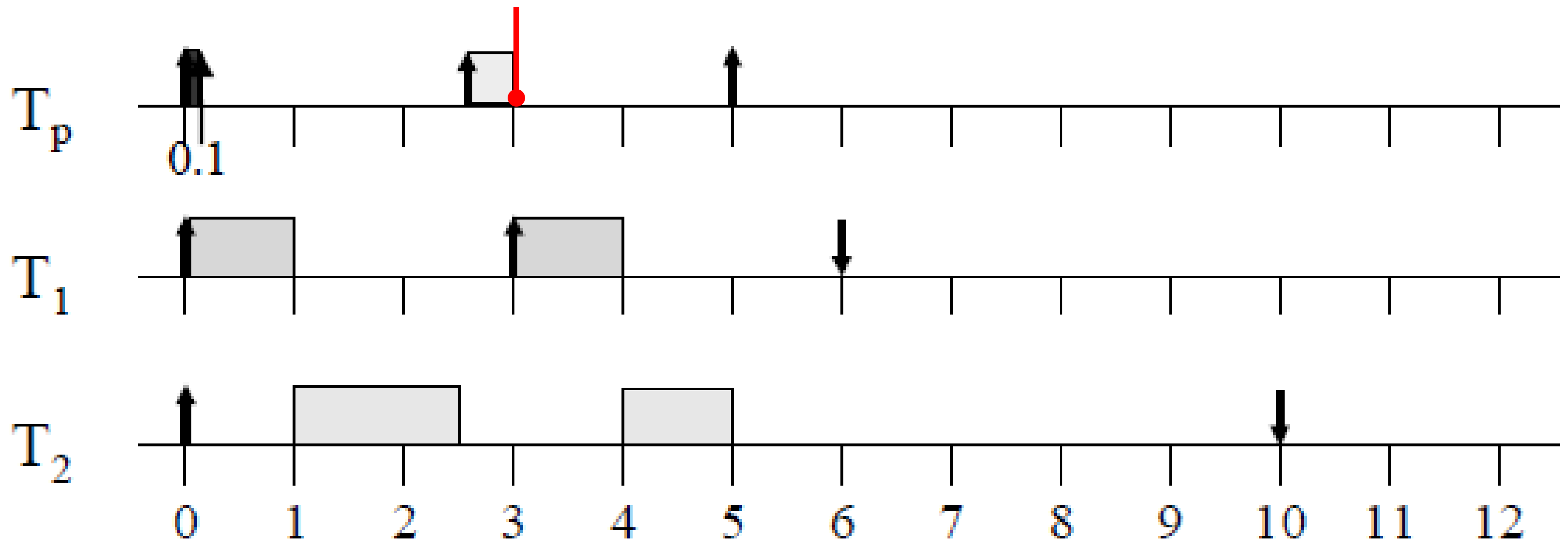
Periodic Server with RM Scheduling

- **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.



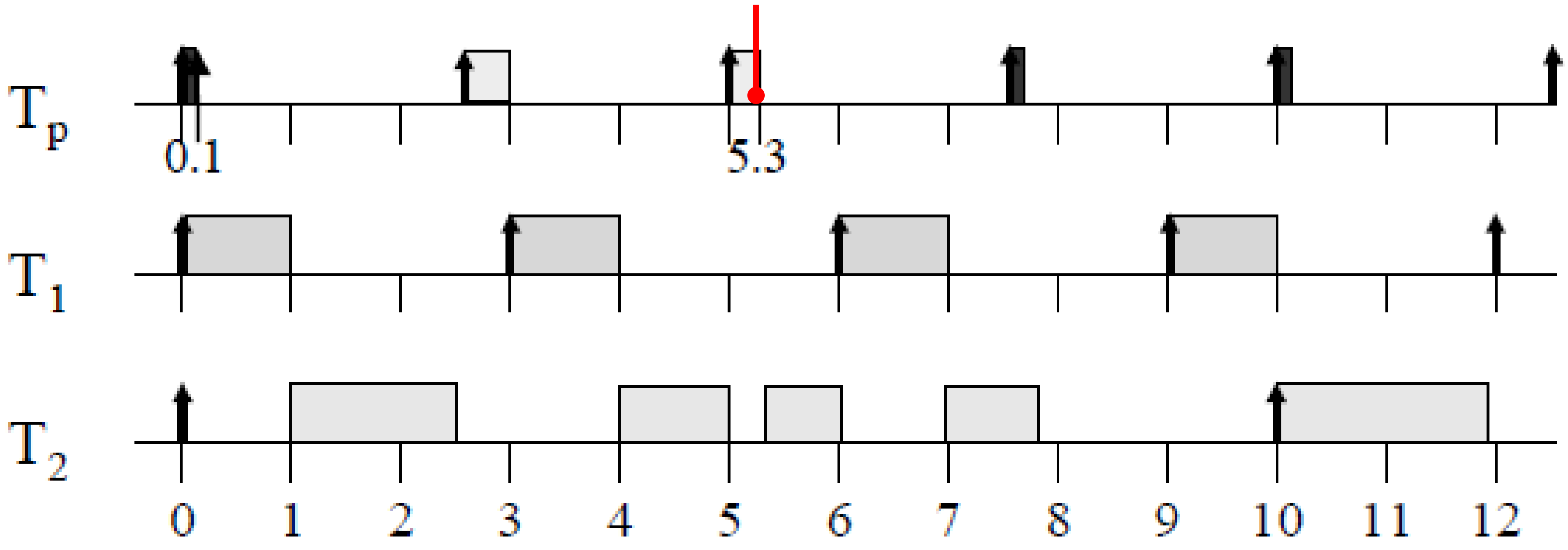
Example (cont.)

- 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.

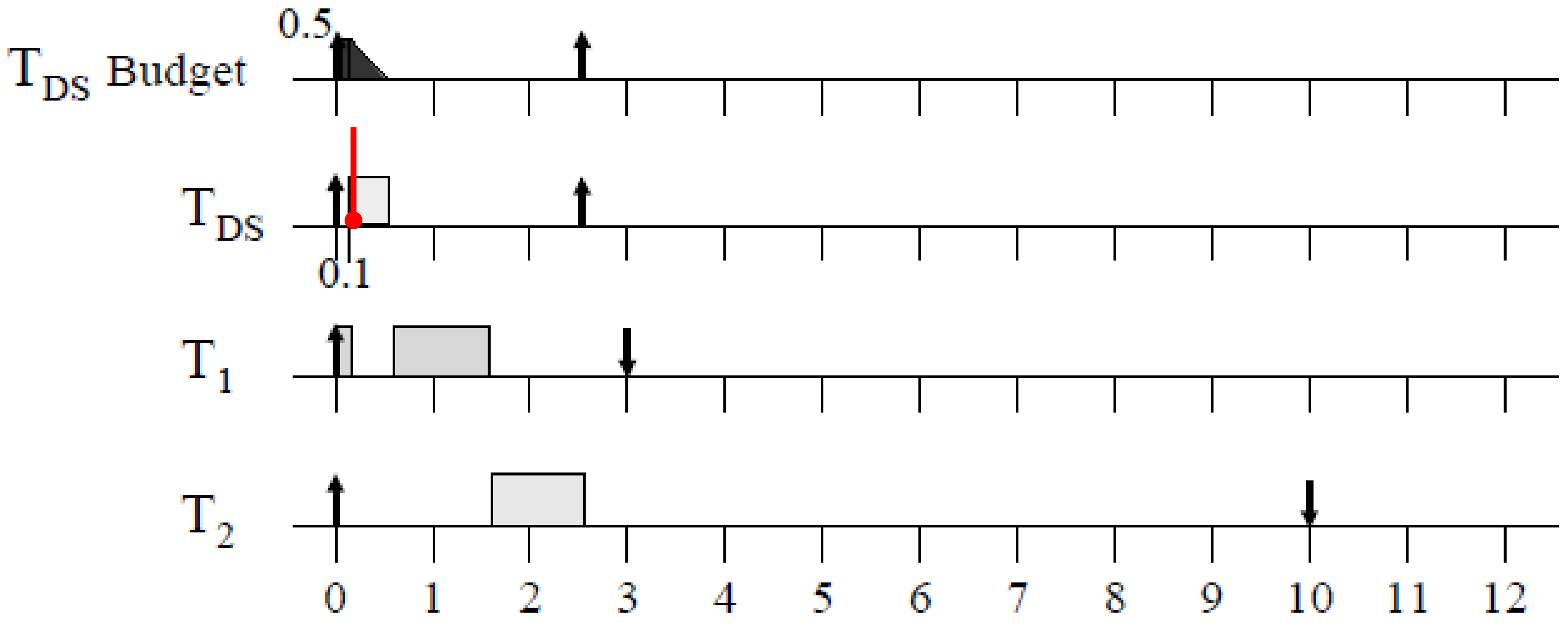
Deferrable Server (DS)

- 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

<u>Example Schedule:</u> 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).

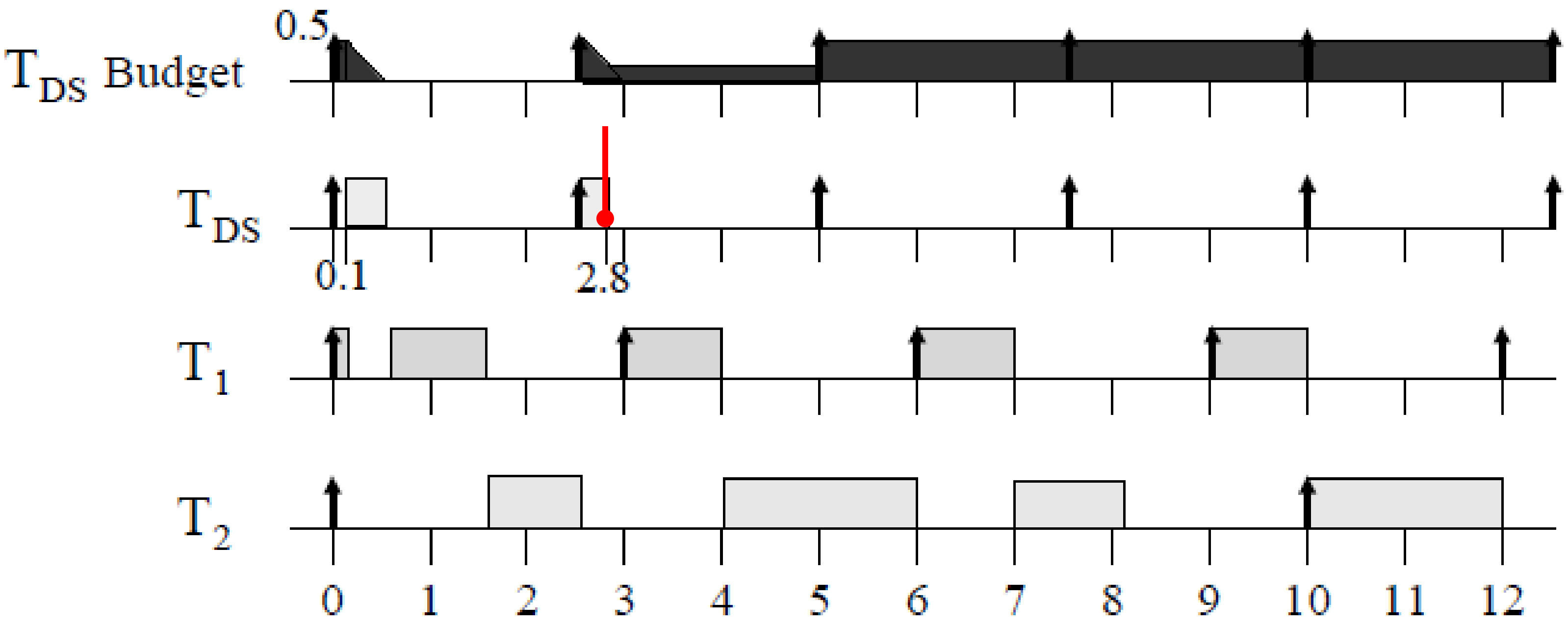
The DS <u>can</u> execute the job that arrives at time 0.1 since it preserved its budget when the aperiodic job queue was empty.



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).

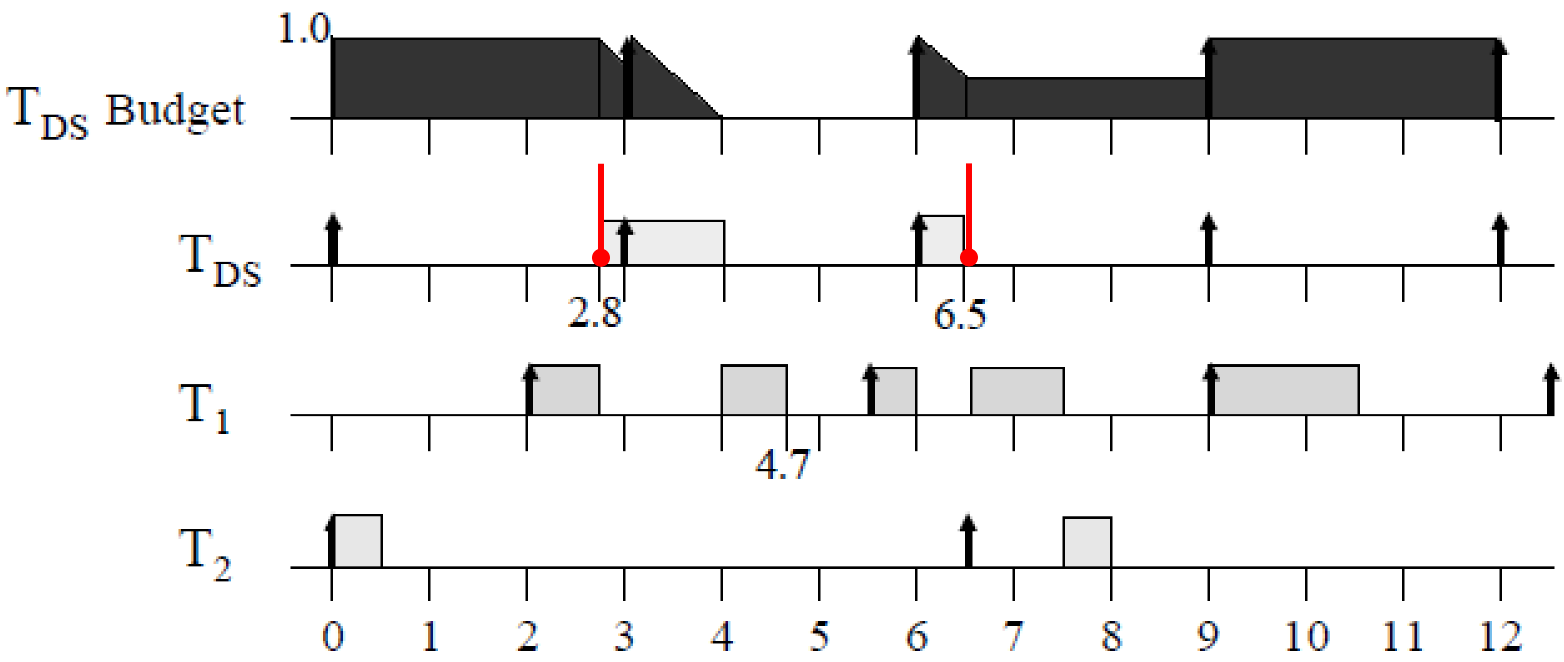
The response time of the aperiodic job J_a is 2.7
It was 5.2 with the simple periodic server.



Another Example w/ RM Scheduling

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 $e_a = 1.7$.

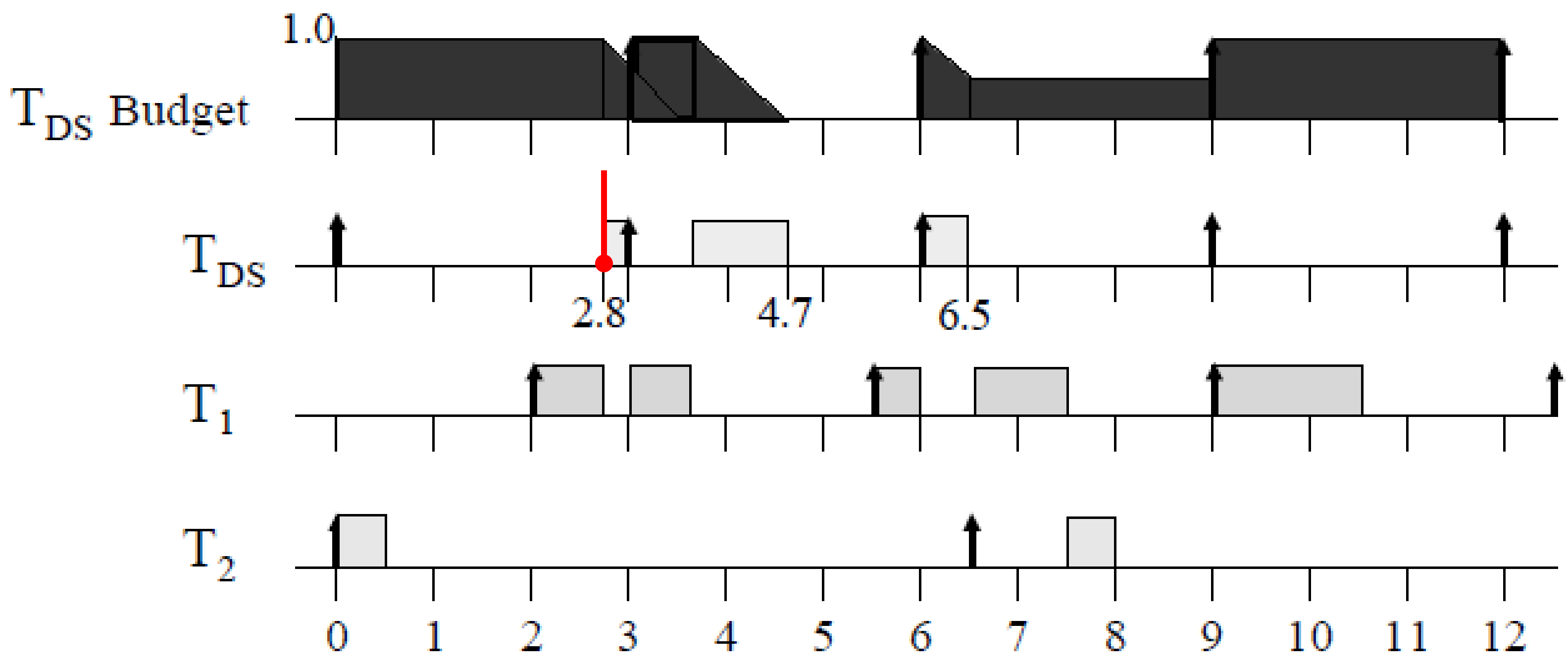
The response time of the aperiodic job J_a is 3.7.



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 $e_a = 1.7$.

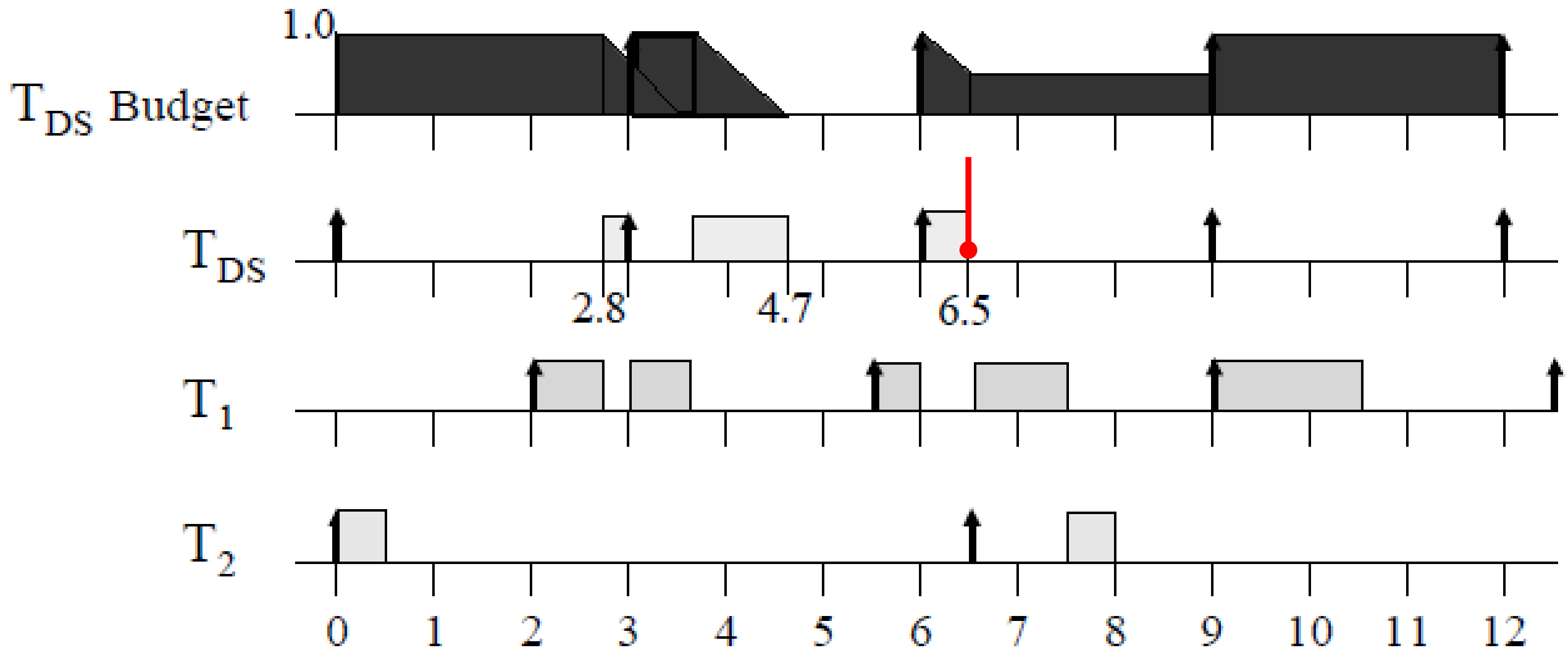
The response time of the aperiodic job J_a is still 3.7.



DS with EDF vs Background Scheduling

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.

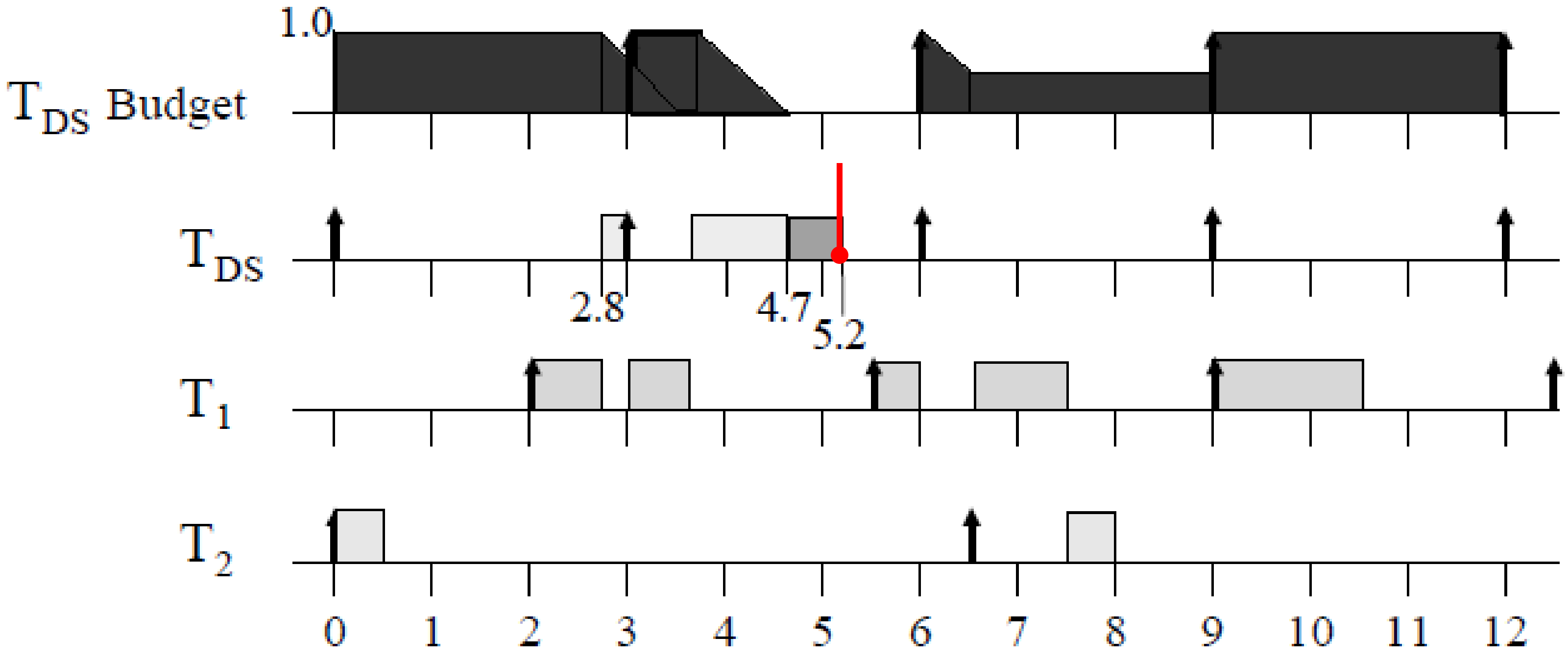


DS with EDF vs **Background Scheduling**

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

Same Task Set: 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.



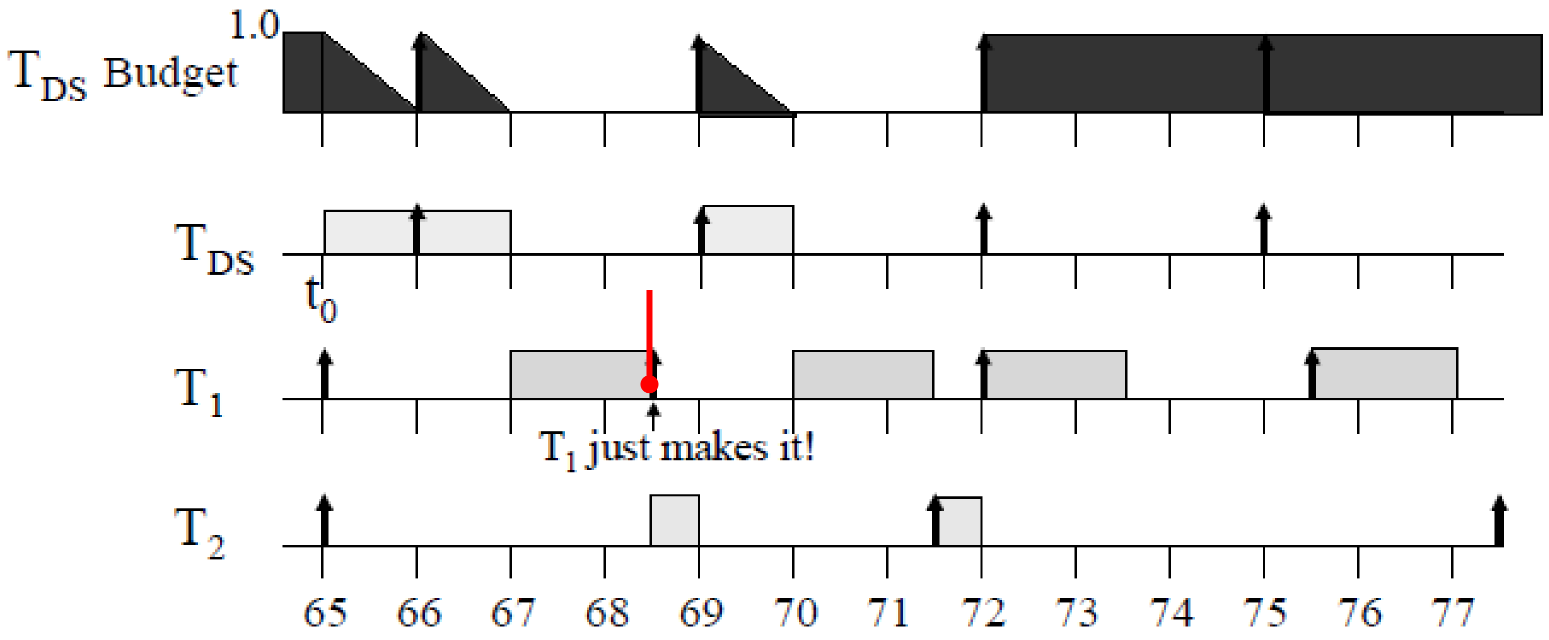
DS with Background Scheduling

- We can also combine background scheduling of the deferrable server with RM.
- 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.



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.

Sporadic Servers (SS)

- We will look at two SS for fixed-priority systems and one for deadline-driven systems.
- They differ in complexity (and thus overhead) due to different consumption and replenishment rules.
- We assume, as with a DS, that the scheduler monitors the execution budget of the SS.
- However, in all cases schedulability conditions remain unchanged from an equivalent system without an SS.

Simple SS in a Fixed-Priority System

First some (new) terms:

- Let **T** be a set of *n* independent, preemptable periodic tasks.
- The (arbitrary) priority of the server T_S in **T** is π_s.
- $\mathbf{T_H}$ is the subset of tasks that have higher priority than T_S.
- **T** ($\mathbf{T_H}$) is idle when no job in **T** ($\mathbf{T_H}$) is eligible for execution.
  - **T** ($\mathbf{T_H}$) is busy when it is not idle.
- Let BEGIN be the instant in time when $\mathbf{T_H}$ transitions from idle to busy, and END be the instant in time when it becomes idle again (or infinity if $\mathbf{T_H}$ is still busy) .
  - The interval (BEGIN, END] is a busy interval.
- t_r is the last replenishment time of T_S.
- t_r' is the next scheduled replenishment time of T_S.
- t_e is the *effective* replenishment time of T_S.
- t_f is the first instant after t_r at which T_S begins to execute.

Simple SS in a Fixed-Priority System

- Consumption Rule: at any time t after t_r, T_S consumes its budget at the rate of one time unit per unit of execution until the budget is exhausted when either

  C1 T_S is executing, or

  C2 T_S has executed since t_r and END < t. (END < t $\Rightarrow$ T_H is currently idle.)

- Replenishment Rule: t_r is set to the current time whenever the execution budget is replenished with e_s time units by the scheduler.

  R1 Initially, $t_r = t_e = 0$ and $t_r' = p_s$ (assuming the system starts at time 0).

  R2 At time t_f,

  - if END = t_f, $t_e = \max(t_r, \text{BEGIN})$.
  - If END < t_f, $t_e = t_f$.

  The next scheduled replenishment time is $t_r' = t_e + p_s$.

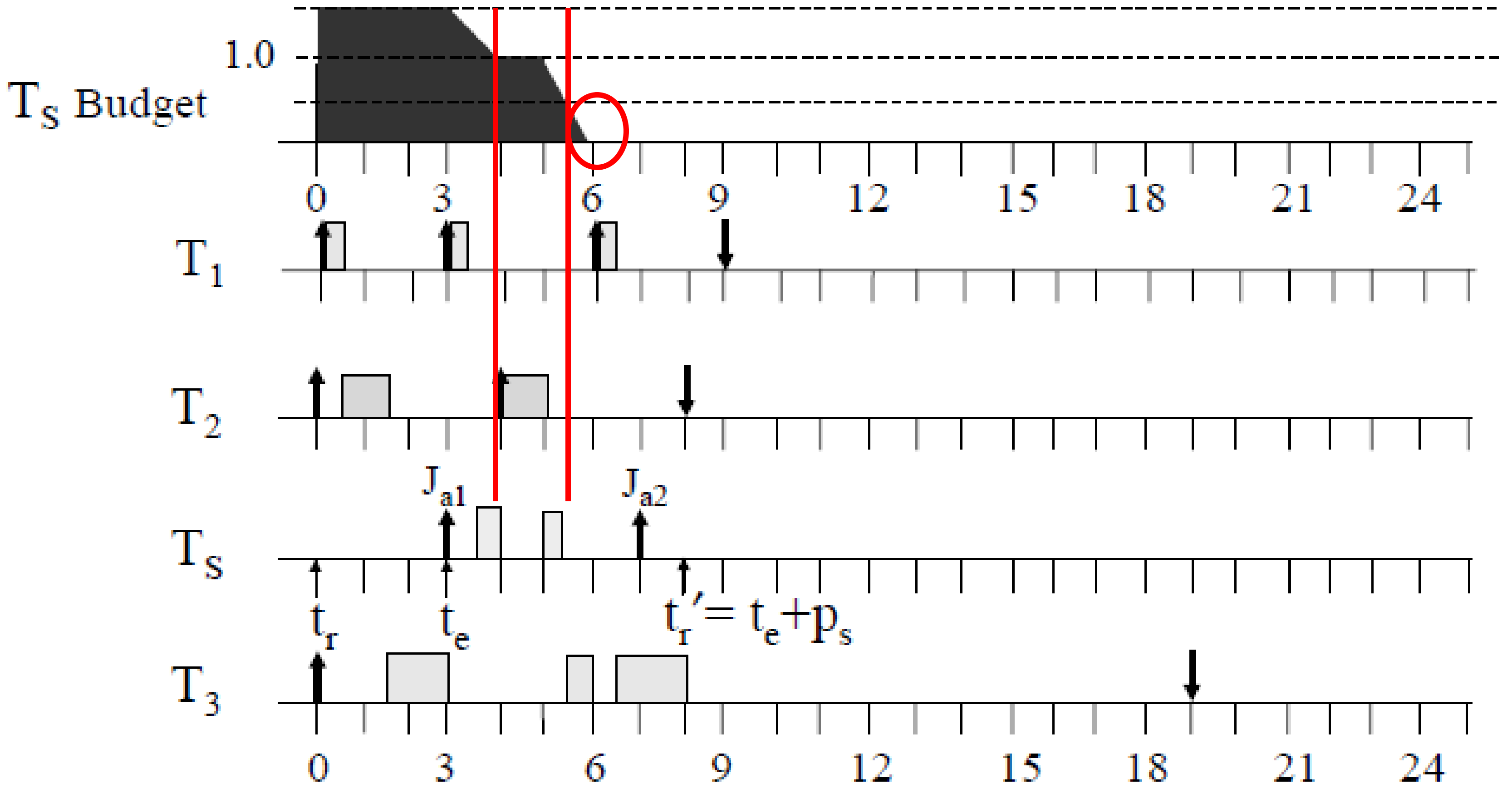
  R3 The next replenishment occurs at t_r' except

  (a) If $t_r' < t_f$, then the budget is replenished as soon as it is exhausted.

  (b) If **T** is idle before t_r' and then begins a new busy interval at t_b, then the budget is replenished at $\min(t_r', t_b)$.

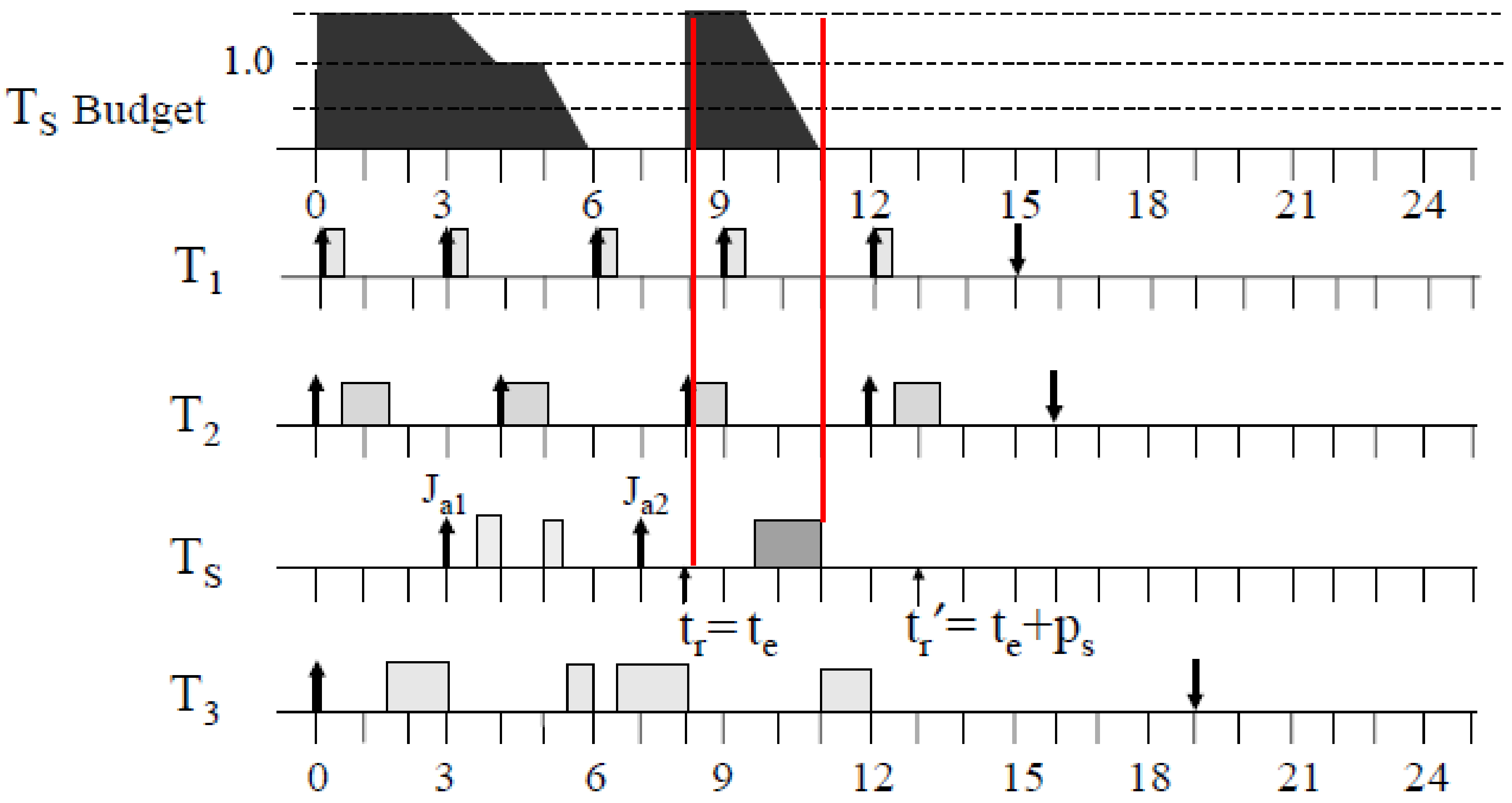
Simple SS with RM Scheduling

Example Schedule: $T_1 = (3,0.5)$, $T_2 = (4,1)$, $T_3 = (19,4.5)$, and $T_S = (5,1.5)$. Assume aperiodic job J_{a1} arrives at $t_1 = 3$ with $e_{a1} = 1$, J_{a2} arrives at $t_2 = 7$ with $e_{a2} = 2$, and J_{a3} arrives at $t_3 = 15.5$ with $e_{a3} = 2$.



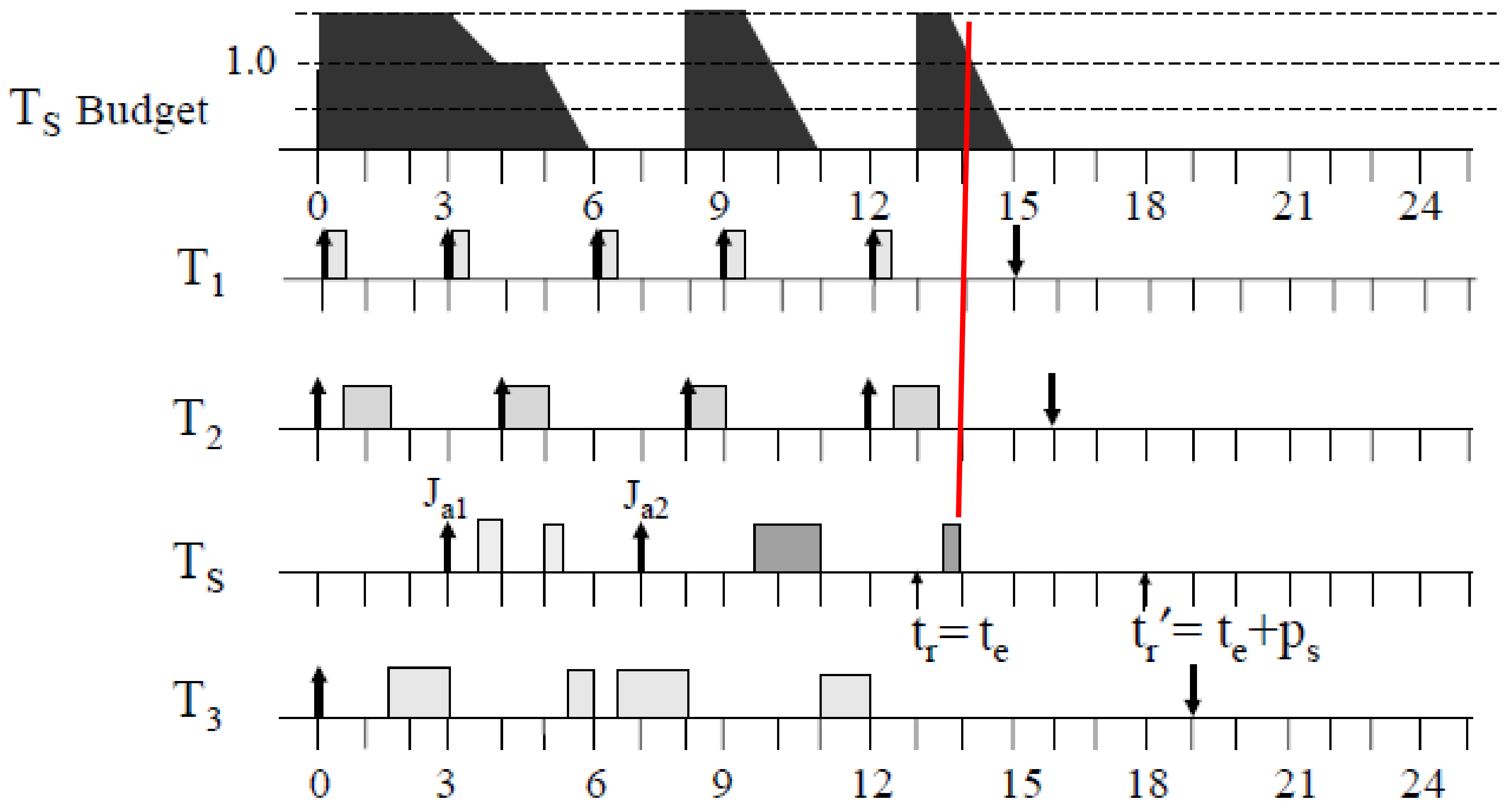
Simple SS with RM Scheduling (cont.)

Example Schedule: $T_1 = (3,0.5)$, $T_2 = (4,1)$, $T_3 = (19,4.5)$, and $T_S = (5,1.5)$. Assume aperiodic job J_{a1} arrives at $t_1 = 3$ with $e_{a1} = 1$, J_{a2} arrives at $t_2 = 7$ with $e_{a2} = 2$, and J_{a3} arrives at $t_3 = 15.5$ with $e_{a3} = 2$.



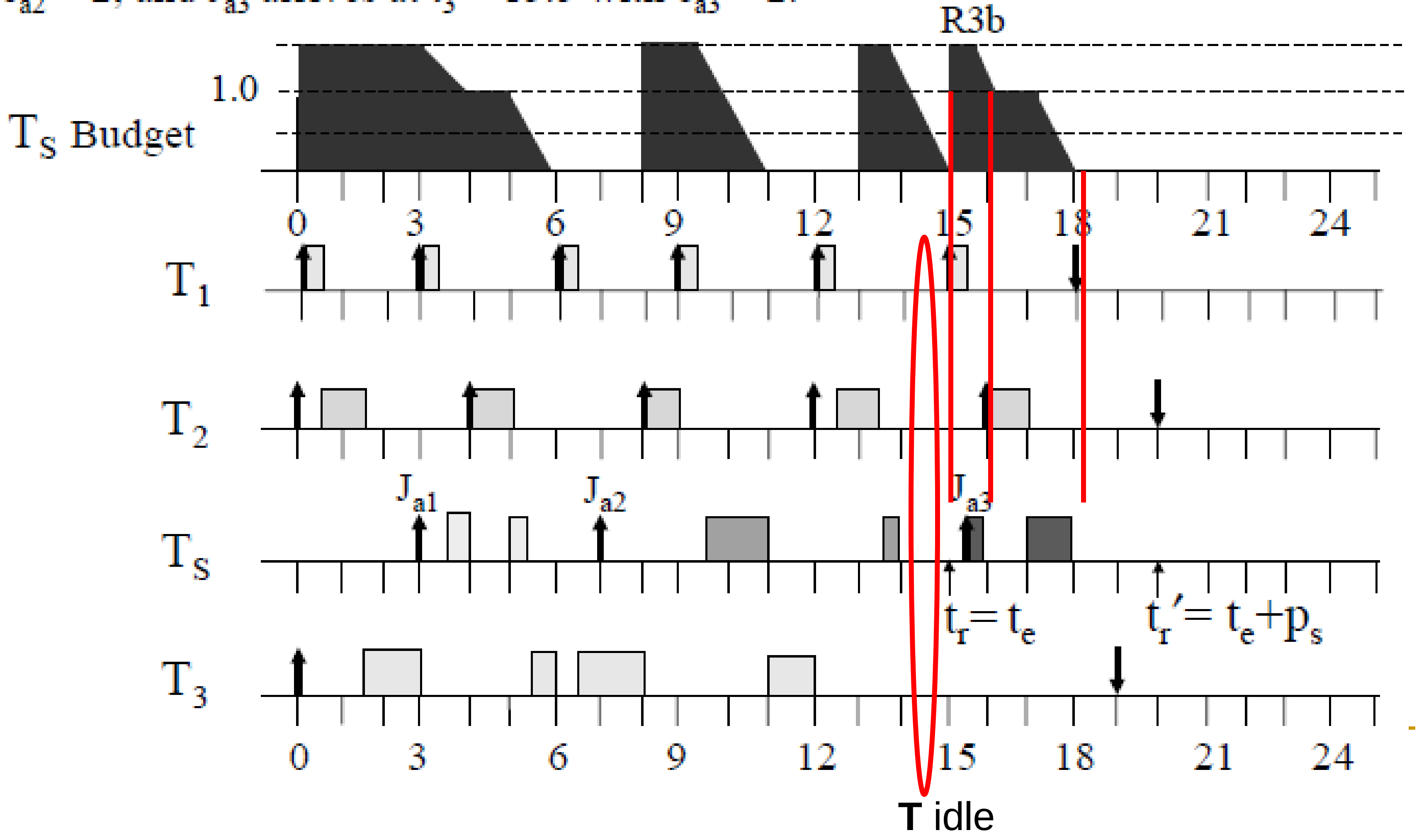
Simple SS with RM Scheduling (cont.)

Example Schedule: $T_1 = (3, 0.5)$, $T_2 = (4, 1)$, $T_3 = (19, 4.5)$, and $T_S = (5, 1.5)$. Assume aperiodic job J_{a1} arrives at $t_1 = 3$ with $e_{a1} = 1$, J_{a2} arrives at $t_2 = 7$ with $e_{a2} = 2$, and J_{a3} arrives at $t_3 = 15.5$ with $e_{a3} = 2$.



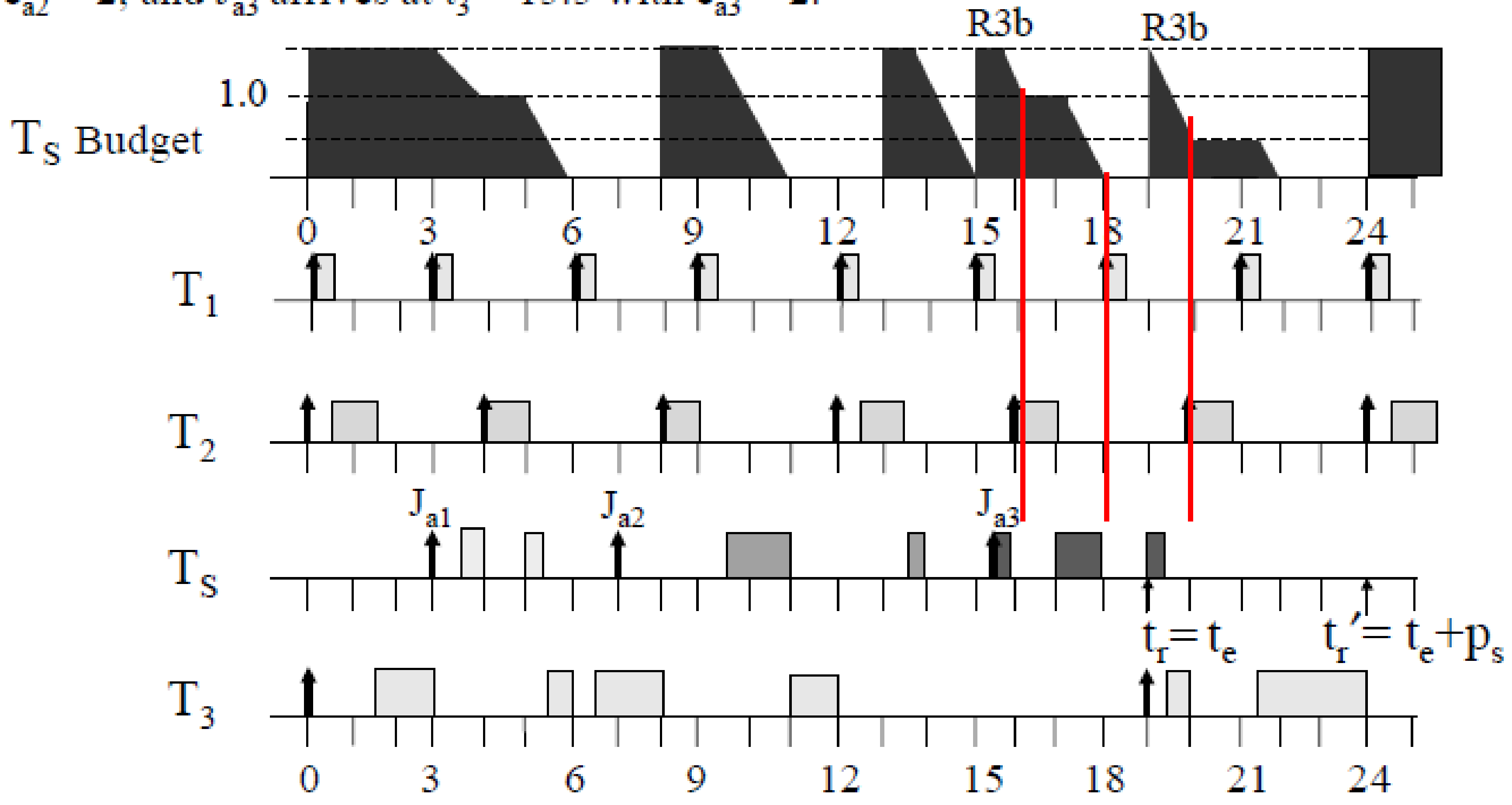
Simple SS with RM Scheduling (cont.)

Example Schedule: $T_1 = (3,0.5)$, $T_2 = (4,1)$, $T_3 = (19,4.5)$, and $T_S = (5,1.5)$. Assume aperiodic job J_{a1} arrives at $t_1 = 3$ with $e_{a1} = 1$, J_{a2} arrives at $t_2 = 7$ with $e_{a2} = 2$, and J_{a3} arrives at $t_3 = 15.5$ with $e_{a3} = 2$.



Simple SS with RM Scheduling (cont.)

Example Schedule: $T_1 = (3, 0.5)$, $T_2 = (4, 1)$, $T_3 = (19, 4.5)$, and $T_S = (5, 1.5)$. Assume aperiodic job J_{a1} arrives at $t_1 = 3$ with $e_{a1} = 1$, J_{a2} arrives at $t_2 = 7$ with $e_{a2} = 2$, and J_{a3} arrives at $t_3 = 15.5$ with $e_{a3} = 2$.



Correctness of Simple SS

- The Simple SS behaves exactly as a “real-world” sporadic task except when Rule R3b is applied.
- Rule R3b takes advantage of the schedulability test for a fixed-priority periodic (“real-world” sporadic) task set **T**.
  - We know that if the system **T** transitions from an idle state to a busy interval, all jobs will make their deadlines--even if they are all released at the same instant (at the start of the busy interval).
  - Thus, Rule R3b replenishes the Simple SS at this instant since it will not affect schedulability!

Enhancements to Simple SS

- We can improve response times of aperiodic jobs by combining the Background Server with the Simple SS to create a Sporadic/Background Server (SBS).
- Consumption Rules are the same as for the Simple SS except when the task system **T** is idle.
  - As long as T is idle, the execution budget stays at e_s.
- Replenishment Rules are the same as for the Simple SS except Rule R3b.
  - The SBS budget is replenished at the beginning of each idle interval of T; t_r is set at the end of the idle interval.

Other Enhancements to Simple SS

- We can also improve response times of aperiodic jobs by replenishing the server's execution budget in small chunks during its period rather than with a single replenishment of e_s time units at the end.
  - This adds to the complexity of the consumption and replenishment rules, of course.
- Sprunt, Sha, and Lehoczky proposed such a server:
  - The SpSL sporadic server preserves unconsumed chunks of budget whenever possible and replenishes the consumed chunks as soon as possible.
  - Thus, it emulates several periodic tasks with parameters $(p_s, e_{s,k})$ such that $\Sigma\, e_{s,k} = e_s$.

SpSL in a Fixed-Priority System

- Breaking of Execution Budget into Chunks:

  B1 Initially, the budget = e_s and $t_r = 0$. There is only one chunk of budget.

  B2 Whenever the server is suspended, the current budget e, if not exhausted, is broken up into two chunks.

  - The first chunk is the portion that was consumed during the last server busy interval, e_1.
    - Its next replenishment time, t_{r1}', is the same as the original chunk's: t_r'. The replenishment amount will be e_1.
  - The second chunk is the remaining budget, e_2.
    - Its last replenishment time is tentatively set to $t_{r2} = t_e$, which will be reset if this budget is used before t_{r1}'. Otherwise, the two chunks will be combined into one budget at time t_{r1}'.

- Consumption Rules:

  C1 The server consumes budgets (when there is more than one budget chunk) in the order of their last replenishment times. That is, the budget with smallest t_r is consumed first.

  C2 The server consumes its budget only when it executes.

- Replenishment Rules: The next replenishment time of each chunk of budget is set according to rules R2 and R3 of the simple SS. The budget chunks are combined whenever they are replenished at the same time (e.g. R3b).

SpSL Rules R2 and R3

- Replenishment rules R2 and R3 from the Simple SS:

  R2 At time t_f,

  - if END = t_f, $t_e = \max(t_r, \text{BEGIN})$.
  - If END < t_f, $t_e = t_f$.

  The next scheduled replenishment time is $t_r' = t_e + p_s$.

  R3 The next replenishment occurs at t_r' except

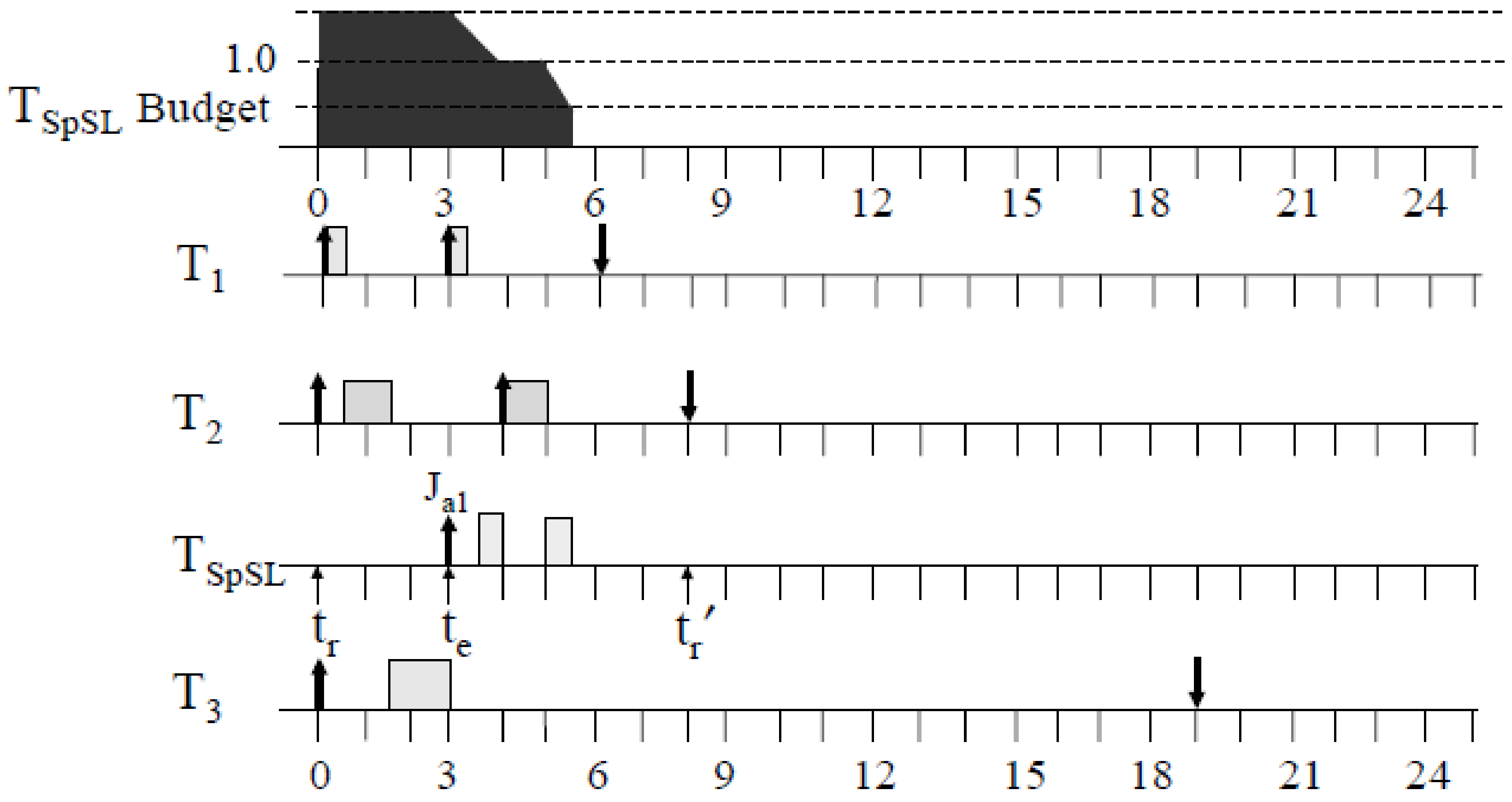
  (a) If $t_r' < t_f$, then the budget is replenished as soon as it is exhausted.

  (b) If **T** is idle before t_r' and then begins a new busy interval at t_b, then the budget is replenished at $\min(t_r', t_b)$.

- Notice that when R3b applies, all budget chunks will be combined into a single budget again.

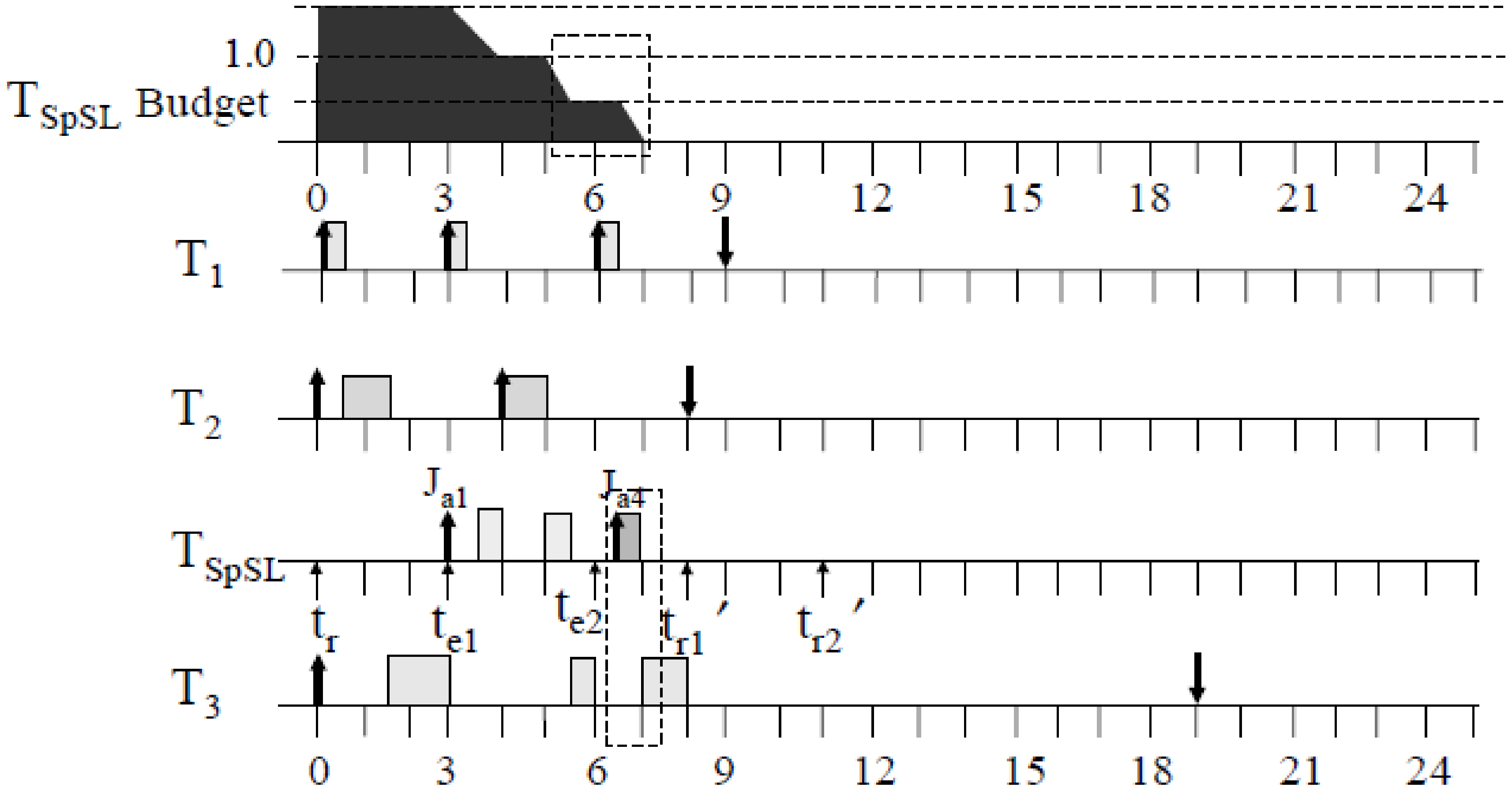
SpSL with RM Scheduling Example

Same Task Set: $T_1 = (3,0.5)$, $T_2 = (4,1)$, $T_3 = (19,4.5)$, and $T_{SpSL} = (5,1.5)$. Assume aperiodic job J_{a1} arrives at $t_1 = 3$ with $e_{a1} = 1$, J_{a2} arrives at $t_2 = 7$ with $e_{a2} = 2$, and J_{a3} arrives at $t_3 = 15.5$ with $e_{a3} = 2$. Plus job J_{a4} arrives at $t_4 = 6.5$ with $e_{a4} = 0.5$



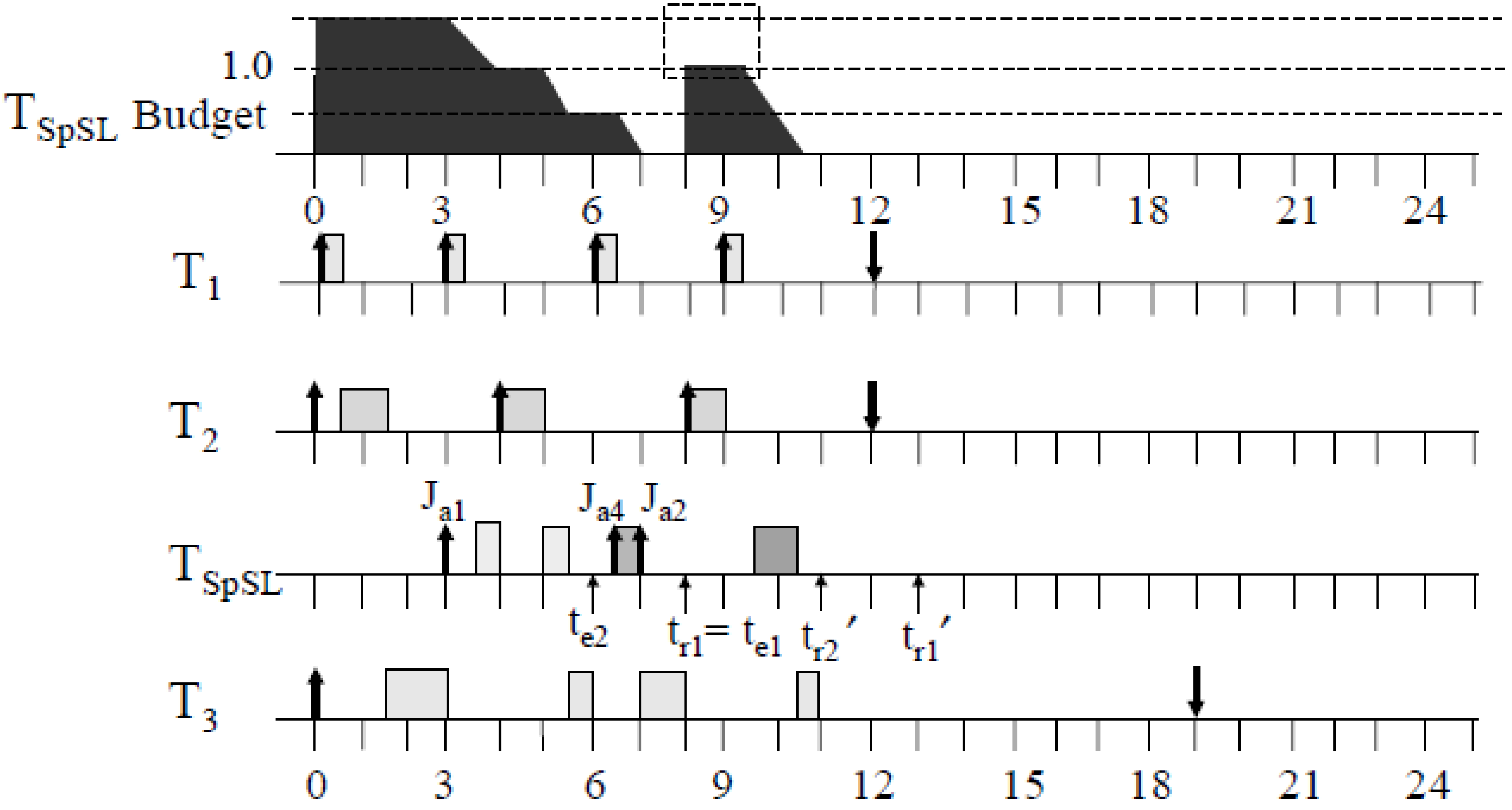
SpSL with RM Scheduling Example (cont.)

Same Task Set: $T_1 = (3,0.5)$, $T_2 = (4,1)$, $T_3 = (19,4.5)$, and $T_{SpSL} = (5,1.5)$. Assume aperiodic job J_{a1} arrives at $t_1 = 3$ with $e_{a1} = 1$, J_{a2} arrives at $t_2 = 7$ with $e_{a2} = 2$, and J_{a3} arrives at $t_3 = 15.5$ with $e_{a3} = 2$. Plus job J_{a4} arrives at $t_4 = 6.5$ with $e_{a4} = 0.5$



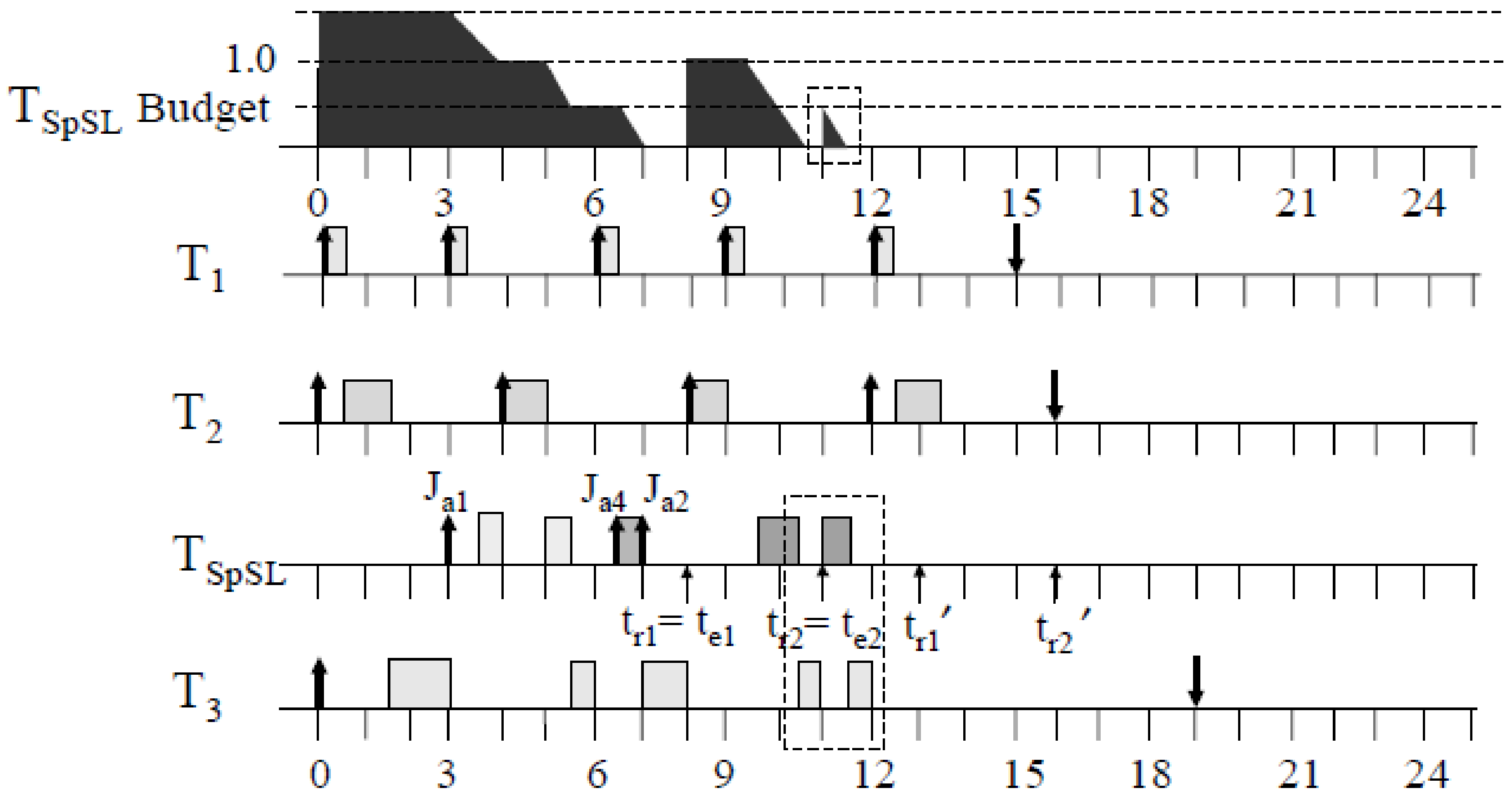
SpSL with RM Scheduling Example (cont.)

Same Task Set: $T_1 = (3,0.5)$, $T_2 = (4,1)$, $T_3 = (19,4.5)$, and $T_{SpSL} = (5,1.5)$. Assume aperiodic job J_{a1} arrives at $t_1 = 3$ with $e_{a1} = 1$, J_{a2} arrives at $t_2 = 7$ with $e_{a2} = 2$, and J_{a3} arrives at $t_3 = 15.5$ with $e_{a3} = 2$. Plus job J_{a4} arrives at $t_4 = 6.5$ with $e_{a4} = 0.5$



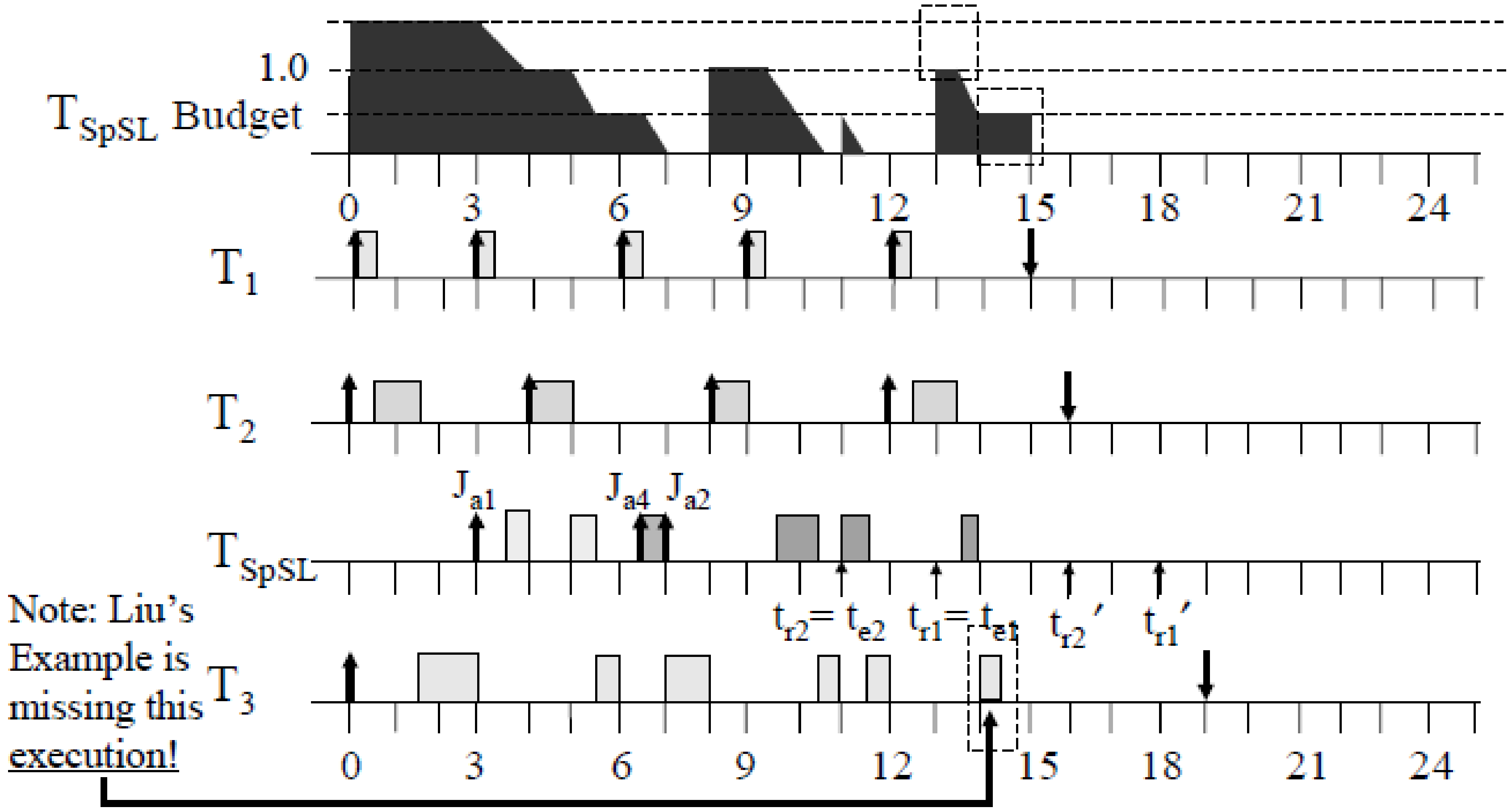
SpSL with RM Scheduling Example (cont.)

Same Task Set: $T_1 = (3,0.5)$, $T_2 = (4,1)$, $T_3 = (19,4.5)$, and $T_{SpSL} = (5,1.5)$. Assume aperiodic job J_{a1} arrives at $t_1 = 3$ with $e_{a1} = 1$, J_{a2} arrives at $t_2 = 7$ with $e_{a2} = 2$, and J_{a3} arrives at $t_3 = 15.5$ with $e_{a3} = 2$. Plus job J_{a4} arrives at $t_4 = 6.5$ with $e_{a4} = 0.5$



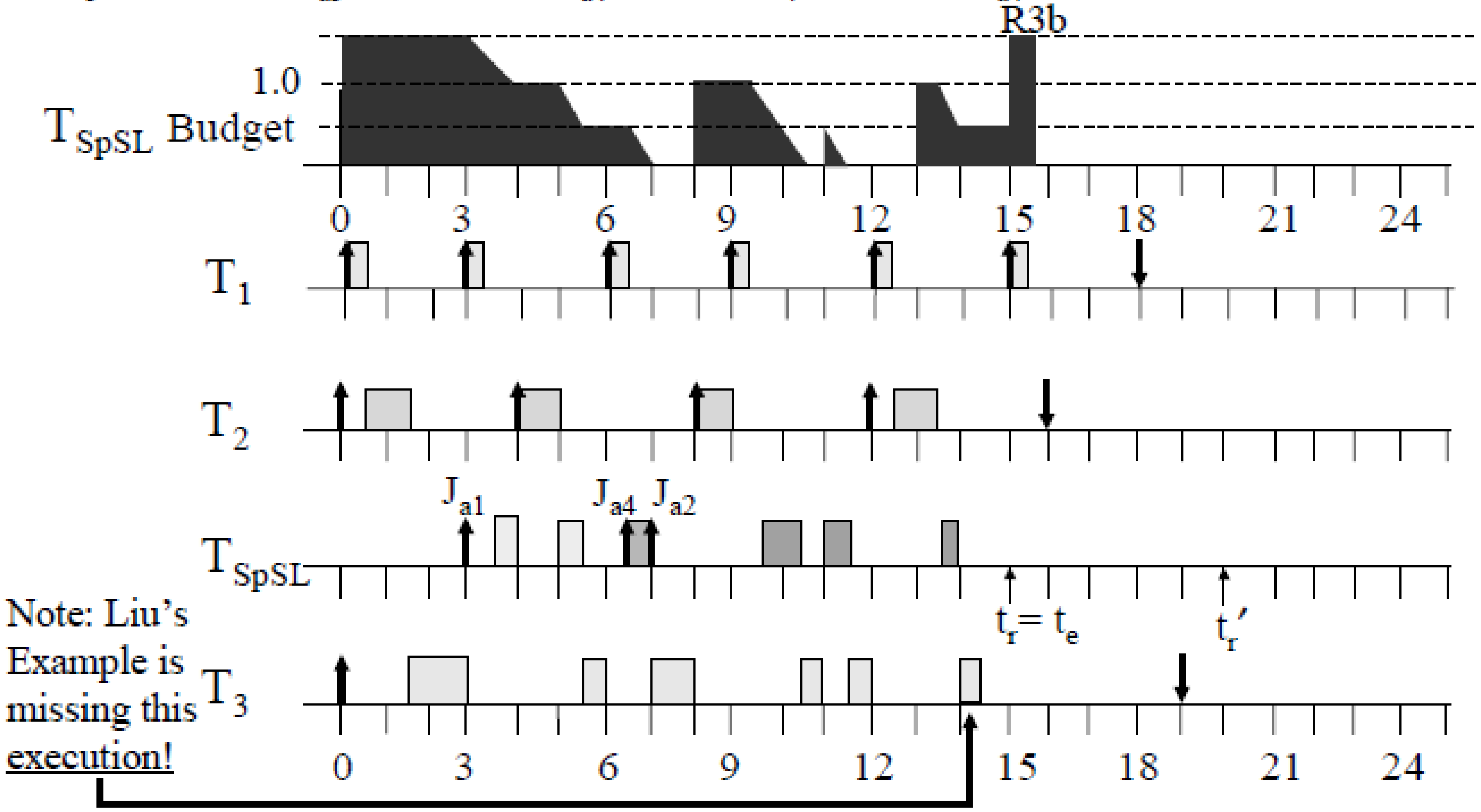
SpSL with RM Scheduling Example (cont.)

<u>Same Task Set:</u> $T_1 = (3,0.5)$, $T_2 = (4,1)$, $T_3 = (19,4.5)$, and $T_{SpSL} = (5,1.5)$. Assume aperiodic job J_{a1} arrives at $t_1 = 3$ with $e_{a1} = 1$, J_{a2} arrives at $t_2 = 7$ with $e_{a2} = 2$, and J_{a3} arrives at $t_3 = 15.5$ with $e_{a3} = 2$. Plus job J_{a4} arrives at $t_4 = 6.5$ with $e_{a4} = 0.5$



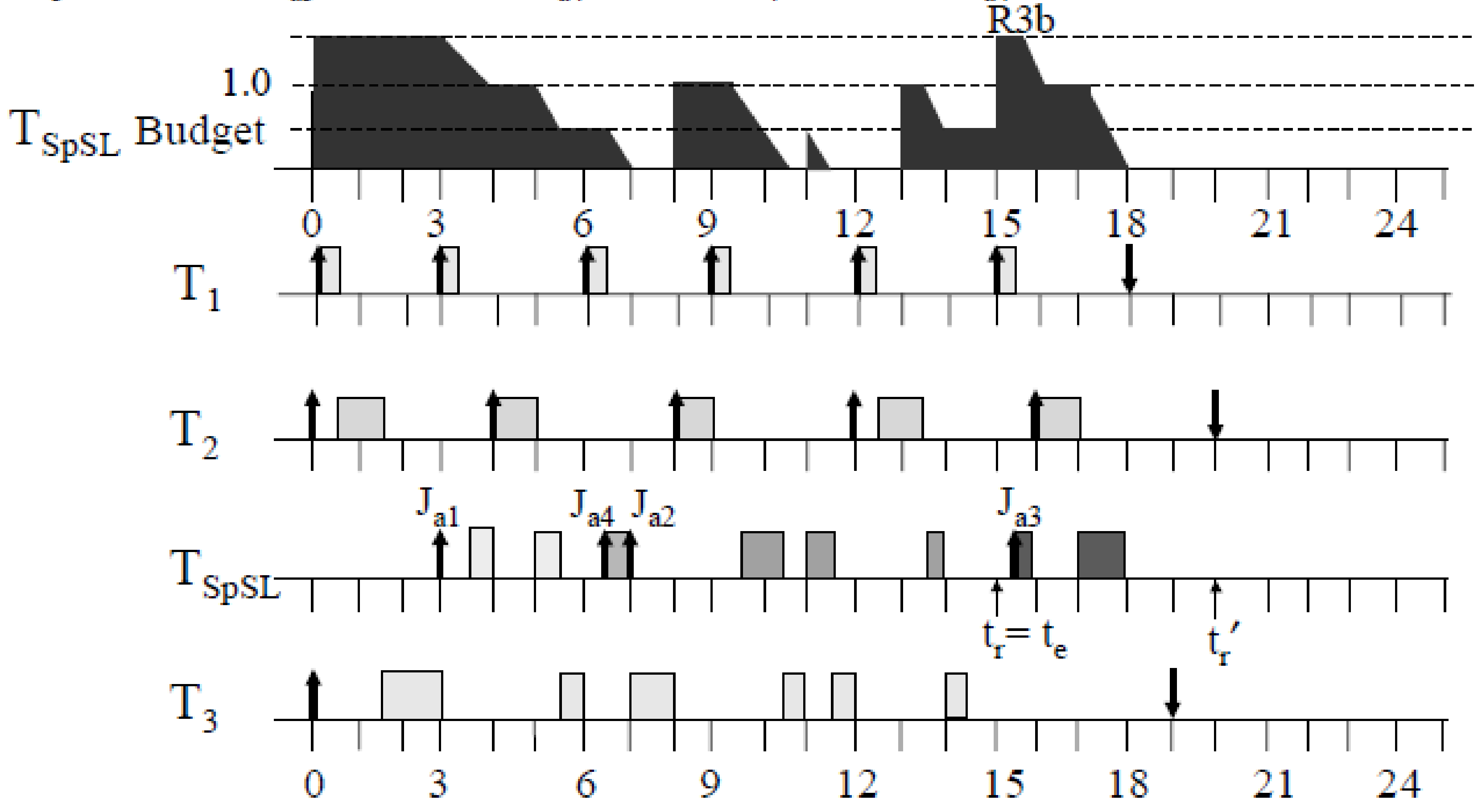
SpSL with RM Scheduling Example (cont.)

Same Task Set: $T_1 = (3,0.5)$, $T_2 = (4,1)$, $T_3 = (19,4.5)$, and $T_{SpSL} = (5,1.5)$. Assume aperiodic job J_{a1} arrives at $t_1 = 3$ with $e_{a1} = 1$, J_{a2} arrives at $t_2 = 7$ with $e_{a2} = 2$, and J_{a3} arrives at $t_3 = 15.5$ with $e_{a3} = 2$. Plus job J_{a4} arrives at $t_4 = 6.5$ with $e_{a4} = 0.5$



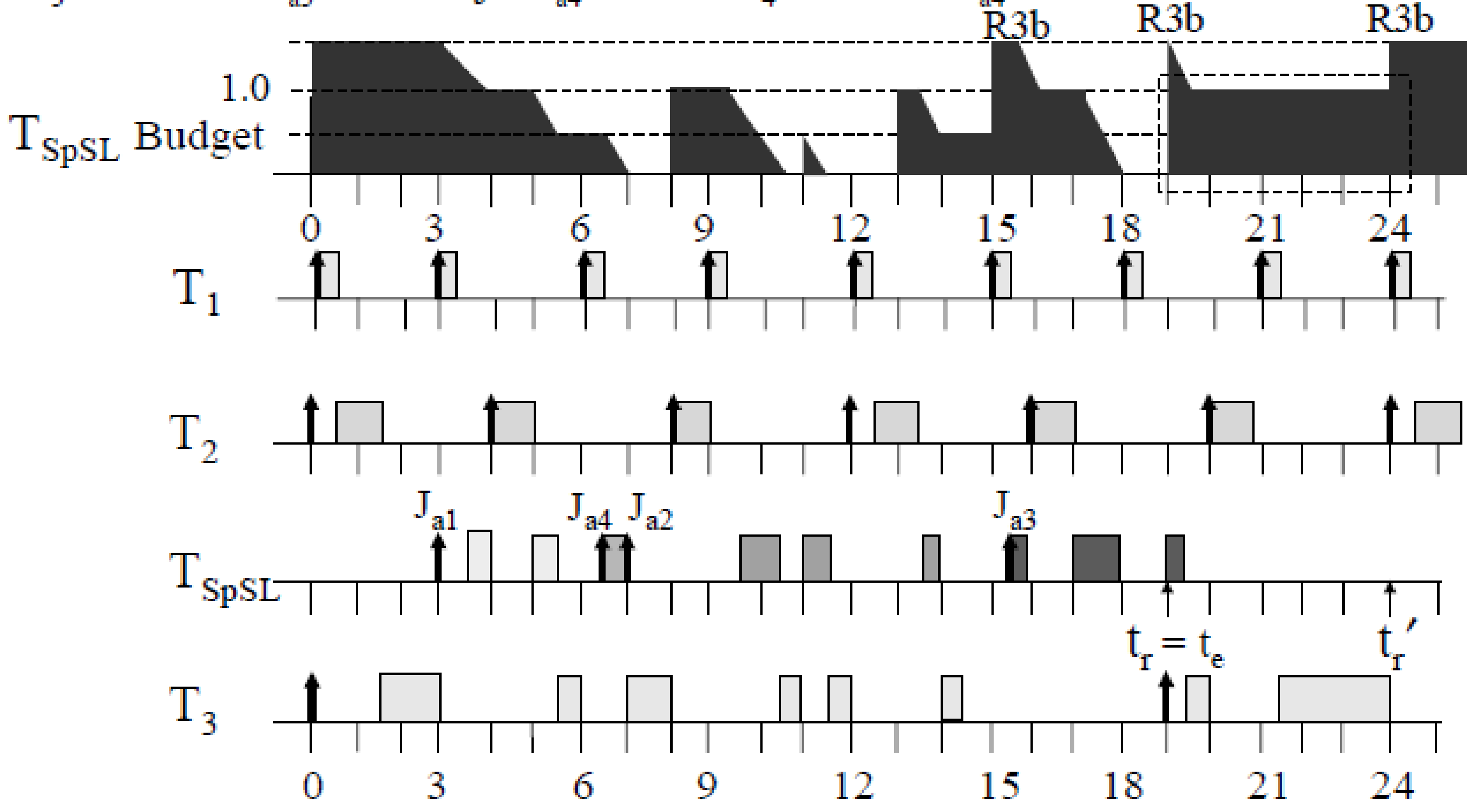
SpSL with RM Scheduling Example (cont.)

Same Task Set: $T_1 = (3,0.5)$, $T_2 = (4,1)$, $T_3 = (19,4.5)$, and $T_{SpSL} = (5,1.5)$. Assume aperiodic job J_{a1} arrives at $t_1 = 3$ with $e_{a1} = 1$, J_{a2} arrives at $t_2 = 7$ with $e_{a2} = 2$, and J_{a3} arrives at $t_3 = 15.5$ with $e_{a3} = 2$. Plus job J_{a4} arrives at $t_4 = 6.5$ with $e_{a4} = 0.5$



SpSL with RM Scheduling Example (cont.)

Same Task Set: $T_1 = (3,0.5)$, $T_2 = (4,1)$, $T_3 = (19,4.5)$, and $T_{SpSL} = (5,1.5)$. Assume aperiodic job J_{a1} arrives at $t_1 = 3$ with $e_{a1} = 1$, J_{a2} arrives at $t_2 = 7$ with $e_{a2} = 2$, and J_{a3} arrives at $t_3 = 15.5$ with $e_{a3} = 2$. Plus job J_{a4} arrives at $t_4 = 6.5$ with $e_{a4} = 0.5$



Enhancing the SpSL Server

- We can further improve response times of aperiodic jobs by combining the SpSL with the Background Server to create a SpSL/Background Server.
  - Try to write precisely the consumption and replenishment rules for this server!
- We can also enhance the SpSL by using a technique called Priority Exchanges.
  - When the server has no work, it trades time with an executing lower priority task.
  - See the textbook (Liu) for details.