Mixing Real-Time and Non-Real-Time

Resource Reservation, Temporal Isolation

Michal Sojka

Czech Technical University in Prague,
FEE and CIIRC

December 4, 2024

Some slides are taken from lectures by Steve Goddard and James H. Anderson

Outline

Outline

Terminology & Goals

Open system A real-time system where a set of tasks and their parameters is not known at design time.
Example: Future cars

Temporal isolation A property guaranteeing that task timing is not influenced by behavior of other tasks. *Resource reservation* is typically a part of solution for temporal isolation.

Resource reservation A way of guaranteeing availability of active resources (e.g. CPU) to tasks even when we do not know what else runs in our system.

Terminology & Goals

Open system	A real-time system where a set of tasks and their parameters is not known at design time. Example: Future cars
Temporal isolation	A property guaranteeing that task timing is not influenced by behavior of other tasks. *Resource reservation* is typically a part of solution for temporal isolation.
Resource reservation	A way of guaranteeing availability of active resources (e.g. CPU) to tasks even when we do not know what else runs in our system.

Goals

- Resource reservation mechanisms at real-time scheduling level.
- Hardware-level problems (e.g. sharing of last-level cache in a multi-core system) are out of scope.

Mixing RT and Non-RT Tasks in Priority-Driven Systems

Chapter 7 of Liu

- We discussed mixing real-time and non-real-time (aperiodic) jobs in cyclic scheduling (Slack stealing)
- We now address the same issue in on-line schedulers
- We first consider two straightforward scheduling algorithms for periodic and aperiodic jobs.
- Then we look at a class of algorithms called bandwidth-preserving servers that schedule aperiodic jobs in a real-time system.

Periodic and Aperiodic Tasks

Review of the terminology used by Liu

- **Periodic task:** T_i is specified by (ϕ_i, p_i, e_i, D_i).
    - p_i is the minimum time between job releases.
    - Previous notation: Task τ_i is specified by (ϕ_i, T_i, C_i, D_i).
- **Aperiodic tasks:** non-real-time
    - Released at arbitrary times.
    - Have no deadline and e_i is unspecified.
- We assume periodic and aperiodic tasks are independent of each other.

Outline

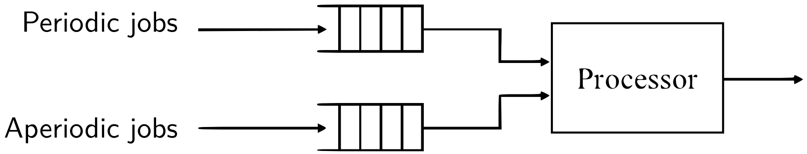
Correct and Optimal Schedules in mixed job systems

Terminology

- 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?

Background Scheduling

- Periodic jobs are scheduled using any priority-driven scheduling algorithm.
- Aperiodic 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 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.

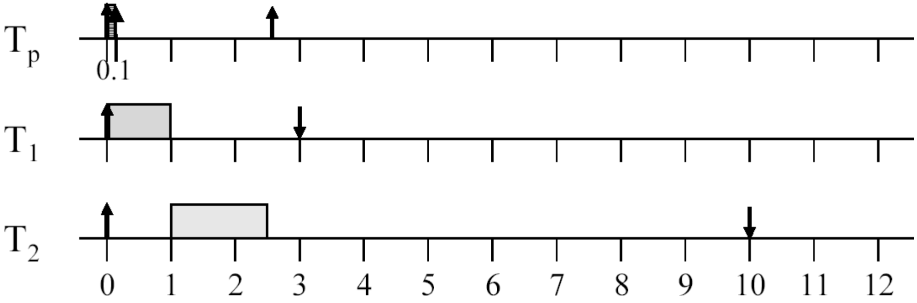
Simple Periodic Server

(Liu calls this a Polling server or the Poller)

- Periodic jobs are scheduled using any priority-driven scheduling algorithm.
- Aperiodic are executed by a special periodic server:
  - The periodic server is a periodic task $T_p = (p_s, e_s)$.
    - 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 T_p 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, $T_1 = (3, 1)$, $T_2 = (10, 4)$, and a periodic server $T_p = (p_s, e_s) = (2.5, 0.5)$. Assume an aperiodic job J_a arrives at $t = 0.1$ with and execution time of $e_a = 0.8$.

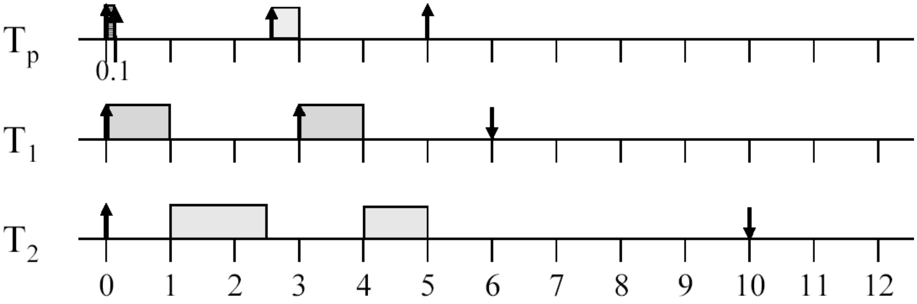


The periodic server cannot execute the job that arrives at time 0.1 since it was suspended at time 0 because the aperiodic job queue was empty.

Periodic Server with RM Scheduling

Example Schedule

Two tasks, $T_1 = (3, 1)$, $T_2 = (10, 4)$, and a periodic server $T_p = (p_s, e_s) = (2.5, 0.5)$. Assume an aperiodic job J_a arrives at $t = 0.1$ with and execution time of $e_a = 0.8$.

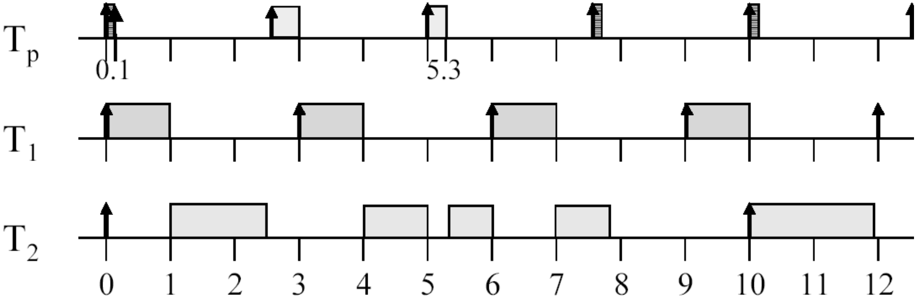


The periodic server executes job J_a until it exhausts its budget.

Periodic Server with RM Scheduling

Example Schedule

Two tasks, $T_1 = (3, 1)$, $T_2 = (10, 4)$, and a periodic server $T_p = (p_s, e_s) = (2.5, 0.5)$. Assume an aperiodic job J_a arrives at $t = 0.1$ with and execution time of $e_a = 0.8$.



The response time of the aperiodic job J_a is 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 ($p_s - \epsilon$ 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!

Bandwidth-Preserving Servers
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.

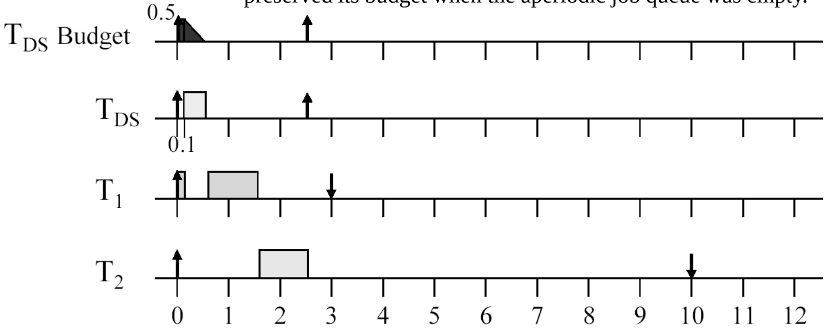
Deferrable Server (DS)

- Let the task $T_{DS} = (p_s, e_s)$ be 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 kp_s, for $k \in \{0, 1, ...\}$.
  - 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 Schedule: Two tasks, $T_1 = (3, 1)$, $T_2 = (10, 4)$, and a periodic server $T_{DS} = (p_s, e_s) = (2.5, 0.5)$. Assume an aperiodic job J_a arrives at $t = 0.1$ with execution time of $e_a = 0.8$.

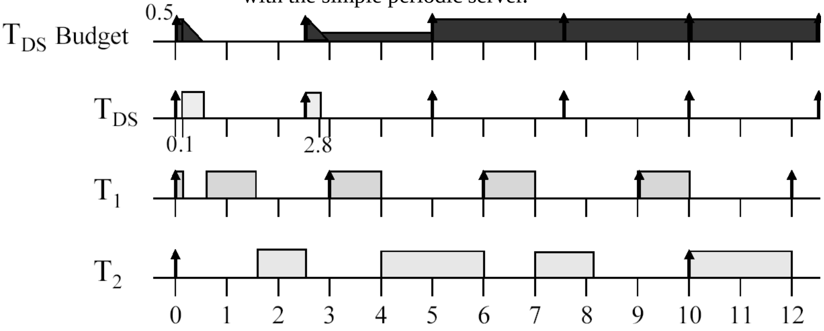
The DS can execute the job that arrives at time 0.1 since it preserved its budget when the aperiodic job queue was empty.



DS with RM Scheduling

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

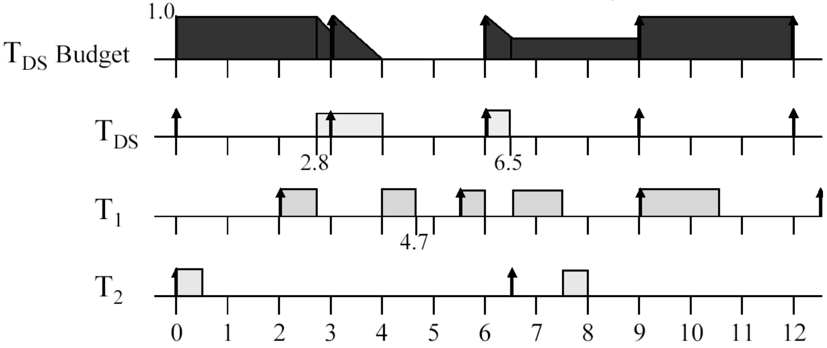
The response time of the aperiodic job J_a is **2.7**. It was **5.2** with the simple periodic server.



DS with RM Scheduling

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

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



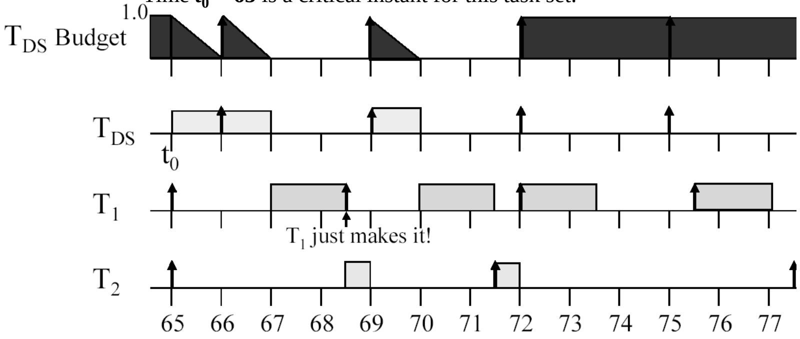
DS with Background Scheduling

- 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: Two tasks, $T_1 = (2, 3.5, 1.5)$, $T_2 = (6.5, 0.5)$, and a periodic server $T_{DS} = (p_s, e_s) = (3, 1)$. Assume an aperiodic job J_a arrives at $t = 65$ with 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 Punch Line

- 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.
- We want the server to behave as plain periodic task.
    - Solution: Sporadic Server, but it's complex!

Sporadic Servers (SS)

- Sporadic Servers (SS) were designed to overcome the additional 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 $\boldsymbol{T_S = (p_s, e_s)}$ 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.

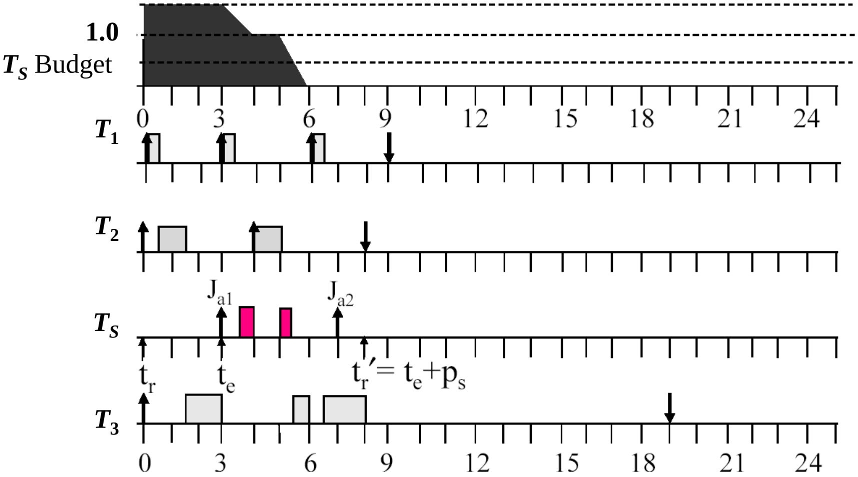
Simple SS in a Fixed-Priority System

- **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** $< t \Rightarrow \mathbf{T}_H$ 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** $= t_f$, $t_e = \max(t_r, \mathbf{BEGIN})$.
    - 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 $\mathbf{T}$ is idle before t_r' and then begins a new busy interval at t_b, then the budget is replenished at $\min(t_r', t_b)$.

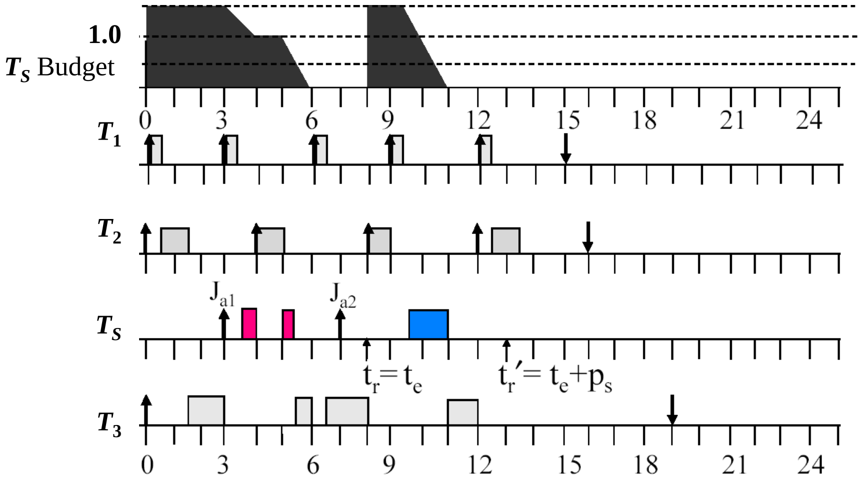
Simple SS with RM Scheduling

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



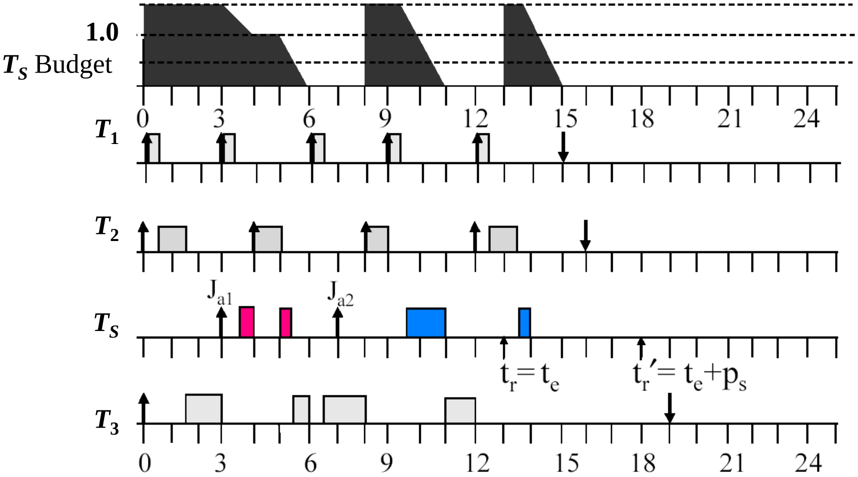
Simple SS with RM Scheduling

<u>Example schedule:</u> ***T_1* = (3,0.5), *T_2* = (4,1), *T_3* = (19,4.5),** and ***T_S* = (5,1.5).** Assume an 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**.



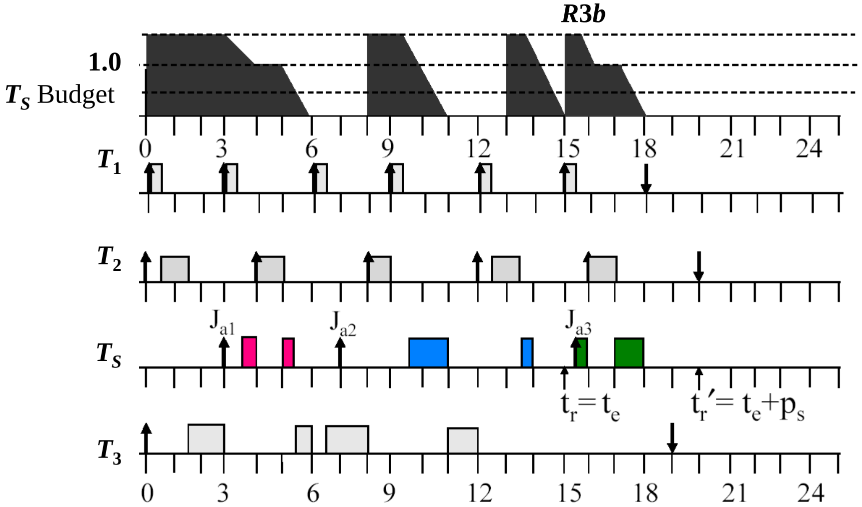
Simple SS with RM Scheduling

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



Simple SS with RM Scheduling

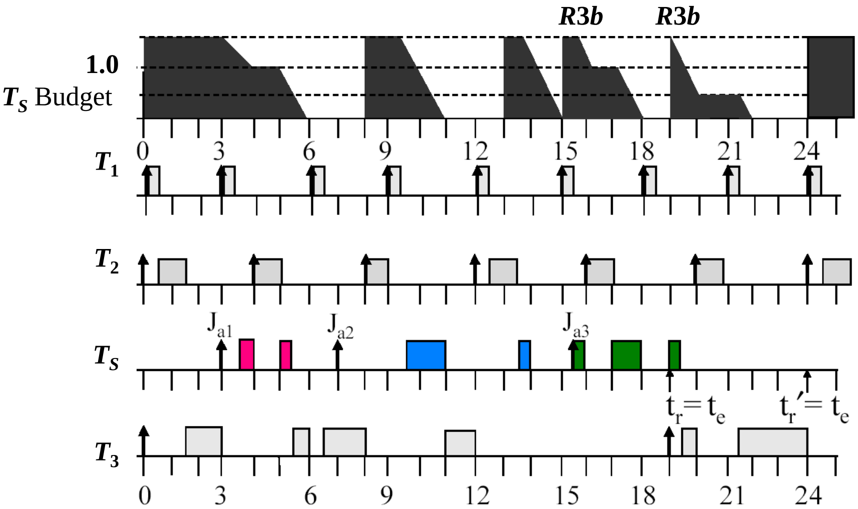
<u>Example schedule:</u> ***T*$_1$ = (3,0.5), *T*$_2$ = (4,1), *T*$_3$ = (19,4.5),** and ***T*$_S$ = (5,1.5).** Assume an aperiodic job ***J*$_{a1}$** arrives at ***t*$_1$ = 3** with ***e*$_{a1}$ = 1**, ***J*$_{a2}$** arrives at ***t*$_2$ = 7** with ***e*$_{a1}$ = 2**, and ***J*$_{a3}$** arrives at ***t*$_3$ = 15.5** with ***e*$_{a3}$ = 2**.



Simple SS with RM Scheduling

<u>Example schedule:</u> $\boldsymbol{T_1 = (3,0.5)}$, $\boldsymbol{T_2 = (4,1)}$, $\boldsymbol{T_3 = (19,4.5)}$, and $\boldsymbol{T_S = (5,1.5)}$.

Assume an aperiodic job $\boldsymbol{J_{a1}}$ arrives at $\boldsymbol{t_1 = 3}$ with $\boldsymbol{e_{a1} = 1}$, $\boldsymbol{J_{a2}}$ arrives at $\boldsymbol{t_2 = 7}$ with $\boldsymbol{e_{a1} = 2}$, and $\boldsymbol{J_{a3}}$ arrives at $\boldsymbol{t_3 = 15.5}$ with $\boldsymbol{e_{a3} = 2}$.



Correctness of Simple SS

- The Simple SS behaves exactly as a “real-world” sporadic task except when Rule **R3*b*** is applied.
- Rule **R3*b*** 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 new busy interval).
  - Thus Rule **R3*b*** replenishes the Simple SS at this instant since it will not affect schedulability!

Enhancements to the 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 $\boldsymbol{e_s}$.
- Replenishment Rules are the same as for the Simple SS except Rule **R3*b***.
  - The SBS budget is replenished at the beginning of each idle interval of **T**. $\boldsymbol{t_r}$ is set at the end of the idle interval.

Other Enhancements to the 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 $\boldsymbol{e_s}$ 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 $(\boldsymbol{p_s}, \boldsymbol{e_{s,k}})$ such that $\Sigma\, \boldsymbol{e_{s,k}} = \boldsymbol{e_s}$.

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 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. **R3*b***).

SpSL Rules R2 and R3

- Replenishment rules **R2** and **R3** from the Simple SS:

R2 At time $\boldsymbol{t_f}$,

- if **END** $= \boldsymbol{t_f}$, $\boldsymbol{t_e} = \mathbf{max}(\boldsymbol{t_r}, \mathbf{BEGIN})$.
- if **END** $< \boldsymbol{t_f}$, $\boldsymbol{t_e} = \boldsymbol{t_f}$.

The next scheduled replenishment time is $\boldsymbol{t_r' = t_e + p_s}$.

R3 The next replenishment occurs at $\boldsymbol{t_r'}$ except

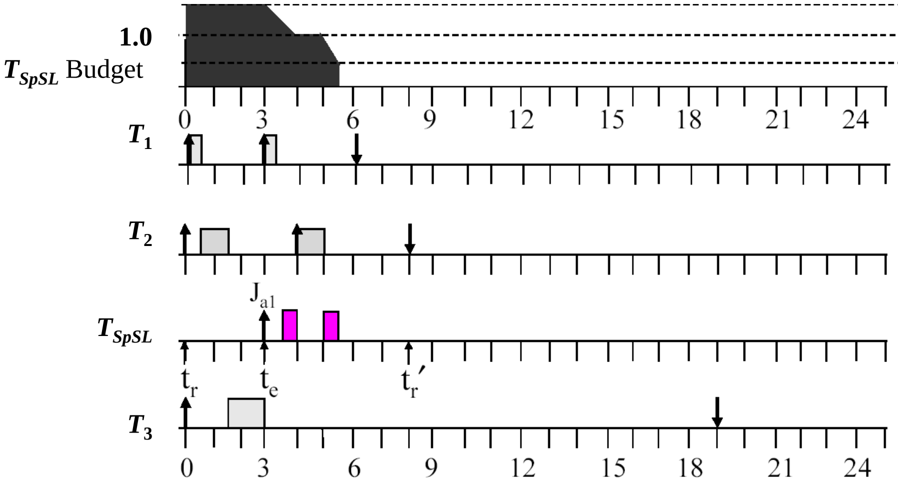
(a) If $\boldsymbol{t_r' < t_f}$, then the budget is replenished as soon as it is exhausted.

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

- Notice that when **R3*b*** applies, all budget chunks will be combined into a single budget again.

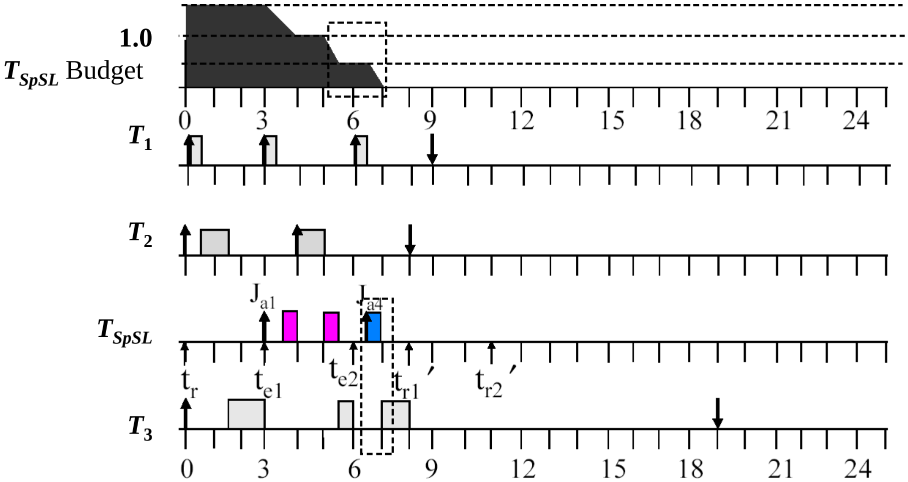
SpSL with RM Scheduling

Same task set: ***T_1* = (3,0.5), *T_2* = (4,1), *T_3* = (19,4.5),** and ***T_{SpSL}* = (5,1.5).** Assume an aperiodic job ***J_{a1}*** arrives at ***t_1* = 3** with ***e_{a1}* = 1**, ***J_{a2}*** arrives at ***t_2* = 7** with ***e_{a1}* = 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**



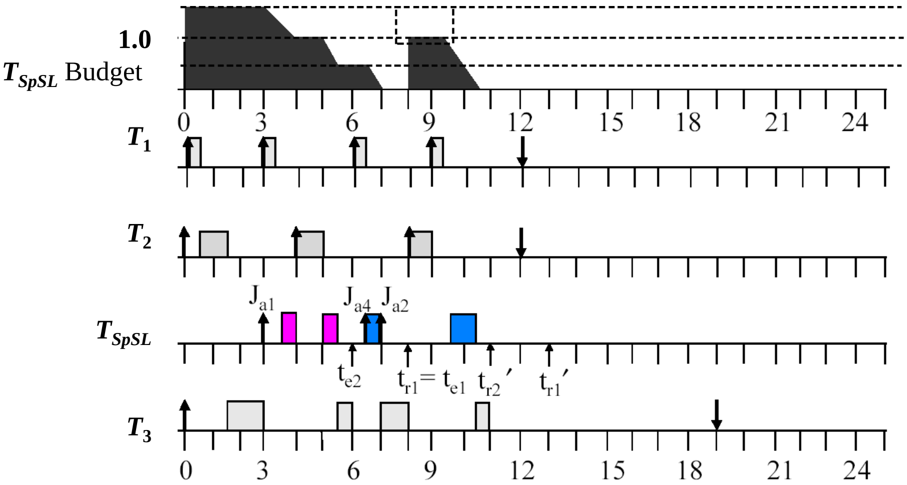
SpSL with RM Scheduling

<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 an aperiodic job J_{a1} arrives at t_1 **= 3** with e_{a1} **= 1**, J_{a2} arrives at t_2 **= 7** with e_{a1} **= 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**



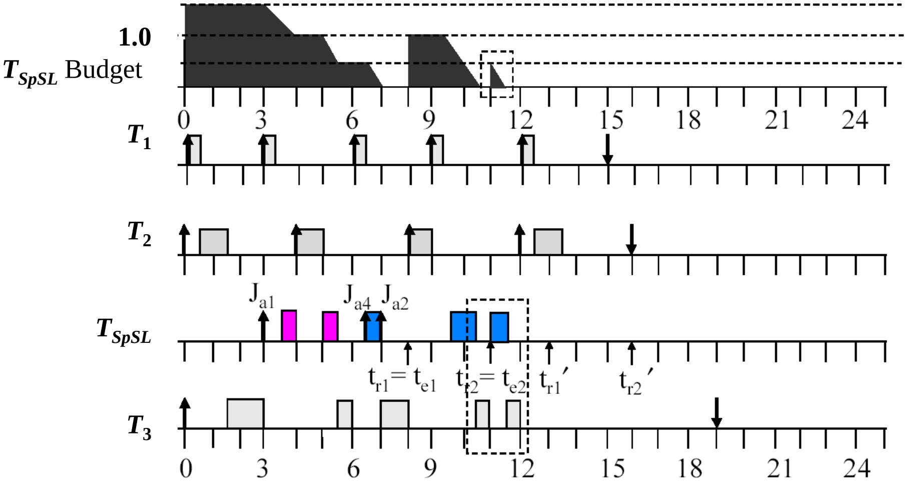
SpSL with RM Scheduling

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



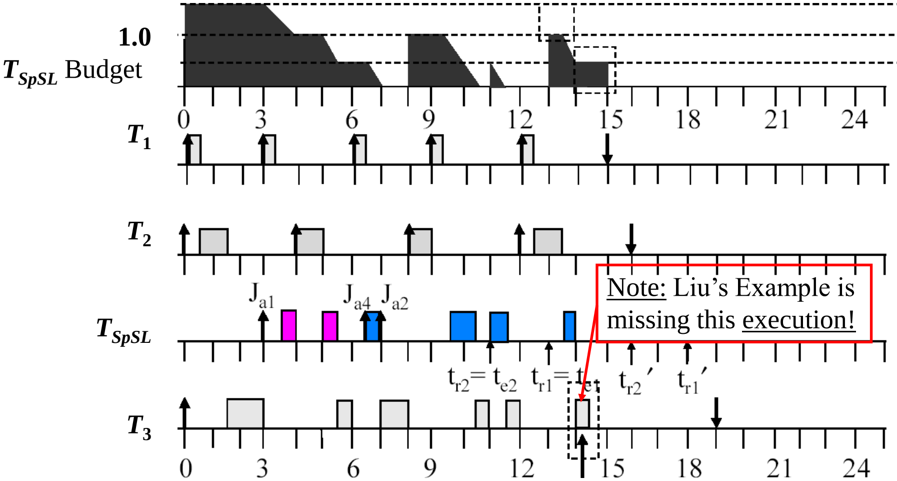
SpSL with RM Scheduling

Same task set: T_1 **= (3,0.5),** T_2 **= (4,1),** T_3 **= (19,4.5),** and T_{SpSL} **= (5,1.5).** Assume an aperiodic job J_{a1} arrives at t_1 **= 3** with e_{a1} **= 1**, J_{a2} arrives at t_2 **= 7** with e_{a1} **= 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

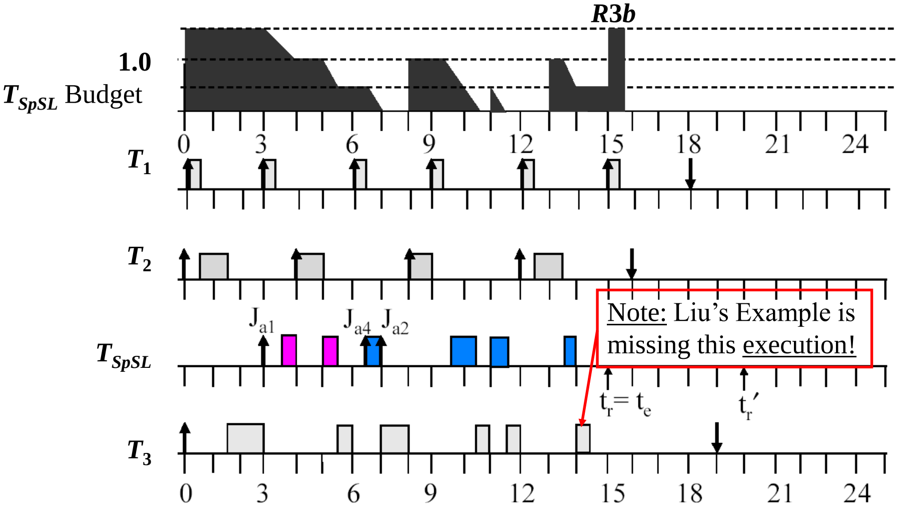
Same task set: $\boldsymbol{T_1 = (3,0.5)}$, $\boldsymbol{T_2 = (4,1)}$, $\boldsymbol{T_3 = (19,4.5)}$, and $\boldsymbol{T_{SpSL} = (5,1.5)}$. Assume an aperiodic job $\boldsymbol{J_{a1}}$ arrives at $\boldsymbol{t_1 = 3}$ with $\boldsymbol{e_{a1} = 1}$, $\boldsymbol{J_{a2}}$ arrives at $\boldsymbol{t_2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



Note: Liu's Example is missing this execution!

SpSL with RM Scheduling

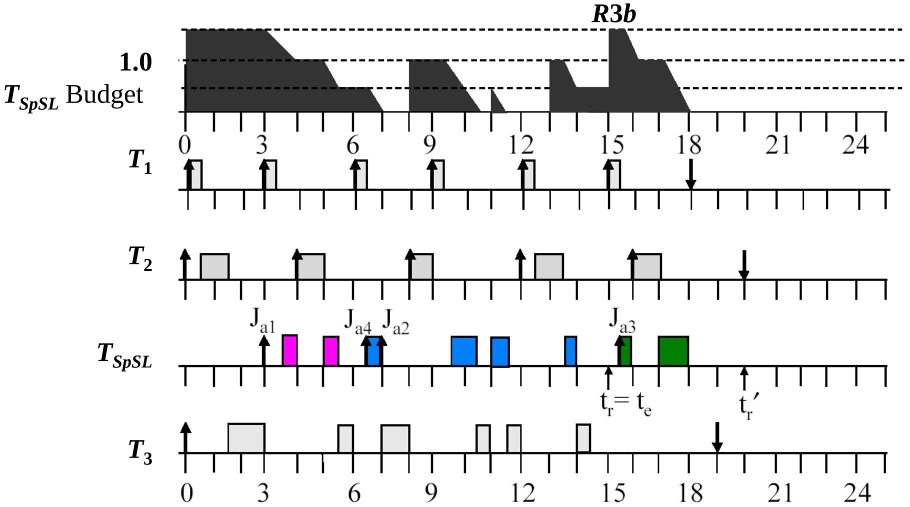
Same task set: $\boldsymbol{T_1 = (3,0.5)}$, $\boldsymbol{T_2 = (4,1)}$, $\boldsymbol{T_3 = (19,4.5)}$, and $\boldsymbol{T_{SpSL} = (5,1.5)}$. Assume an aperiodic job J_{a1} arrives at $\boldsymbol{t_1 = 3}$ with $\boldsymbol{e_{a1} = 1}$, J_{a2} arrives at $\boldsymbol{t_2 = 7}$ with $\boldsymbol{e_{a1}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



Note: Liu's Example is missing this execution!

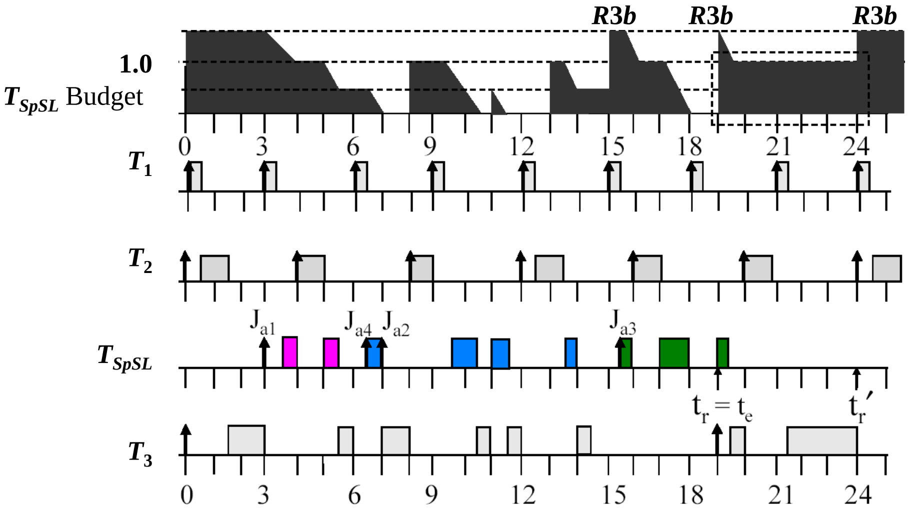
SpSL with RM Scheduling

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



SpSL with RM Scheduling

<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 an aperiodic job ***J***$_{a1}$ arrives at ***t***$_1$ **= 3** with ***e***$_{a1}$ **= 1**, ***J***$_{a2}$ arrives at ***t***$_2$ **= 7** with ***e***$_{a1}$ **= 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 Liu for details.

Outline

Aperiodic Job Servers for Deadline-Driven Systems

- The Total Bandwidth Server (TBS) was created by Spuri and Butazzo (RTSS ‘94) to schedule
  - aperiodic task whose arrival time was unknown but
  - whose worst-case execution time (***wcet***) was known.
  - A trivial admission control algorithm used with the TBS can also schedule sporadic jobs (aperiodic jobs whose ***wcet*** and deadline is known).
- The constant utilization server was created by Deng, Liu, and Sun (Euromicro Workshop ‘97) to schedule
  - aperiodic task whose arrival time was unknown but
  - whose ***wcet*** and deadline was known.
  - However, they also wanted to schedule aperiodic jobs with no deadlines and whose ***wcet*** was unknown.
  - This server is almost the same as the TBS.

Aperiodic Job Servers for Deadline-Driven Systems

- Liu’s presentation of the constant utilization server and the TBS is poorly motivated and nearly unintelligible.
- You should read the papers, they make more sense than the material in the book.
- We will first cover the TBS and then the constant utilization server since it may be easier to understand the value of the constant utilization server when they are presented in this order.

Total Bandwidth Server (TBS)

- One way to reduce the response time of aperiodic jobs whose ***wcet*** is known in a deadline-driven system is to
  - allocate a fixed (maximum) percentage, $\boldsymbol{U_S}$, of the processor to the serve aperiodic jobs, and
  - make sure the aperiodic load never exceeds this maximum utilization value.
  - When an aperiodic job comes in, assign it a deadline such that the demand created by all of the aperiodic jobs in any feasible interval never exceeds the maximum utilization allocated to aperiodic jobs.
- This approach is the main idea behind the TBS.

  Note: We use $\boldsymbol{U_S}$ to denote the server size (as Spuri and Buttazzo do) rather than $\tilde{u}_s$ as Liu does.

TBS in Deadline-Driven Systems

- Let TB be a TBS of size $\boldsymbol{U_S}$ in a task system **T** scheduled with EDF. Thus, the server is allocated $\boldsymbol{U_S}$ percent of the total processor bandwidth.
- The server is ready for execution only when it is backlogged.
  - In other words: the server is only suspended when it is idle.
- **Consumption Rule:** When executing, TB consumes its budget at the rate of one time unit per unit of execution until the budget is exhausted.
- **Replenishment Rule:**

R1 Initially, the execution budget $\boldsymbol{e_s = 0}$ and the server's deadline $\boldsymbol{d_s = 0}$.

R2 At time $\boldsymbol{t}$ when aperiodic job $\boldsymbol{J_i}$ with execution time $\boldsymbol{e_i}$ arrives and the server is idle, set $\boldsymbol{d_s = \max(d_s, t) + e_i/U_S}$ and $\boldsymbol{e_s = e_i}$. If the server is backlogged, no nothing.

R3 When the server completes the currently executing aperiodic job, $\boldsymbol{J_i}$,

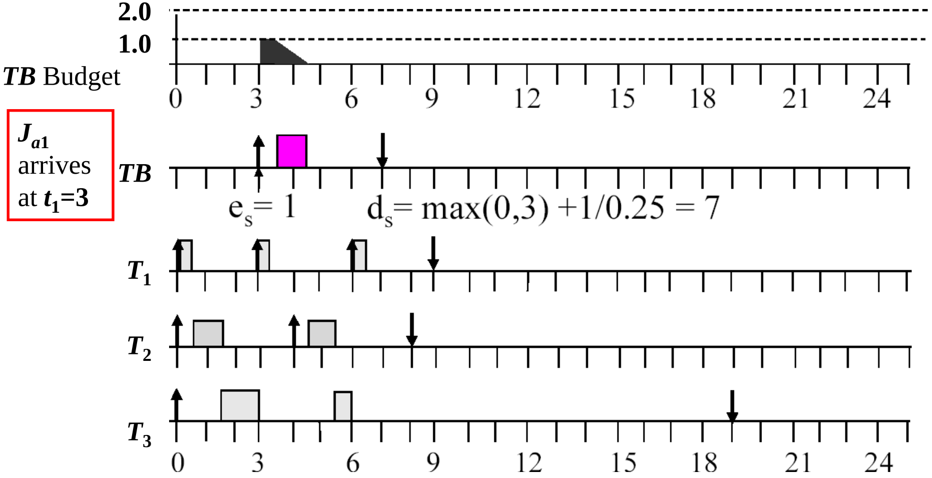
(a) If the server is backlogged, the server deadline is set to $\boldsymbol{d_s = d_s + e_i /U_S}$ and $\boldsymbol{e_s = e_i}$.

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

TBS with EDF Scheduling

<u>Same task set:</u> $\boldsymbol{T_1}$ **= (3,0.5),** $\boldsymbol{T_2}$ **= (4,1),** $\boldsymbol{T_3}$ **= (19,4.5),** but $\boldsymbol{U_S}$ **= 0.25.**
Assume an aperiodic job $\boldsymbol{J_{a1}}$ arrives at $\boldsymbol{t_1}$ **= 3** with $\boldsymbol{e_{a1}}$ **= 1**, $\boldsymbol{J_{a2}}$ arrives at $\boldsymbol{t_2}$ **= 6.9** with $\boldsymbol{e_{a1}}$ **= 2**, and $\boldsymbol{J_{a3}}$ arrives at $\boldsymbol{t_3}$ **= 14** with $\boldsymbol{e_{a3}}$ **= 2**.

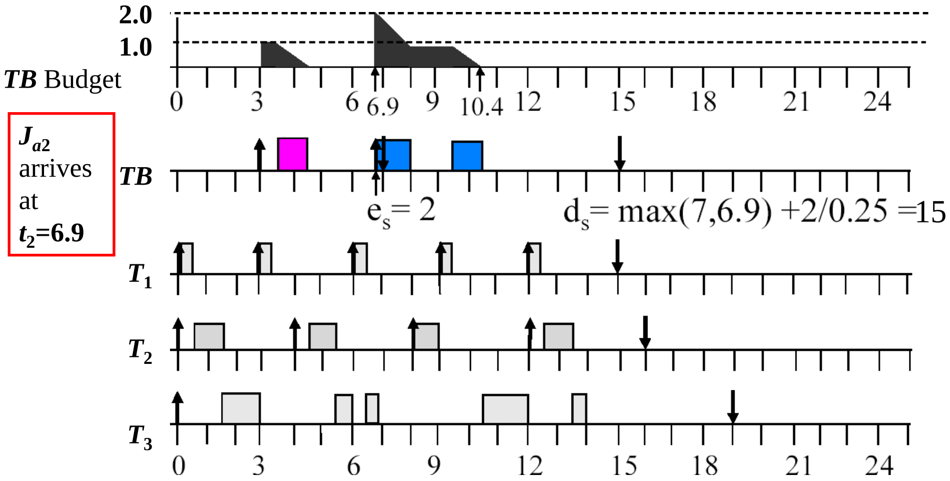
$\boldsymbol{J_{a1}}$ arrives at $\boldsymbol{t_1}$**=3**



TBS with EDF Scheduling

Same task set: T_1 **= (3,0.5),** T_2 **= (4,1),** T_3 **= (19,4.5),** but U_S **= 0.25.** Assume an aperiodic job J_{a1} arrives at t_1 **= 3** with e_{a1} **= 1**, J_{a2} arrives at t_2 **= 6.9** with e_{a1} **= 2**, and J_{a3} arrives at t_3 **= 14** with e_{a3} **= 2**.

J_{a2} arrives at t_2**=6.9**

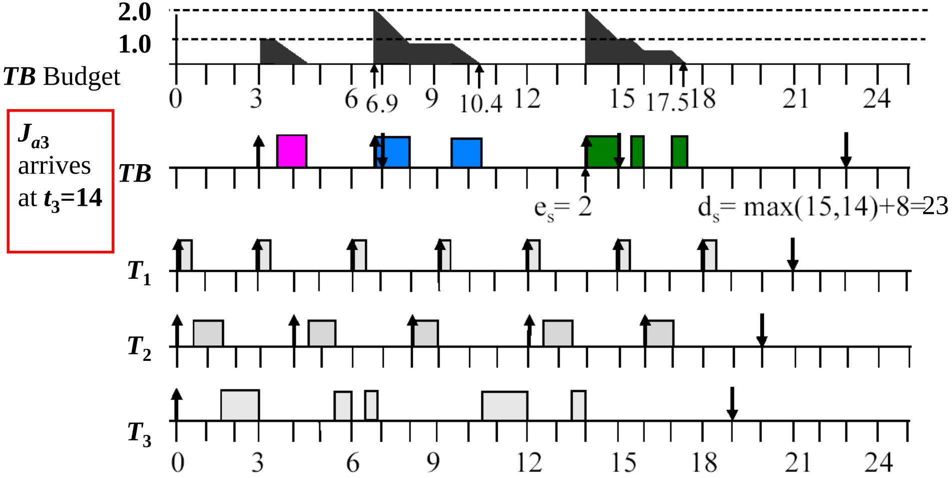


TBS with EDF Scheduling

Same task set: T_1 **= (3,0.5),** T_2 **= (4,1),** T_3 **= (19,4.5),** but U_S **= 0.25.**

Assume an aperiodic job J_{a1} arrives at t_1 **= 3** with e_{a1} **= 1**, J_{a2} arrives at t_2 **= 6.9** with e_{a1} **= 2**, and J_{a3} arrives at t_3 **= 14** with e_{a3} **= 2**.

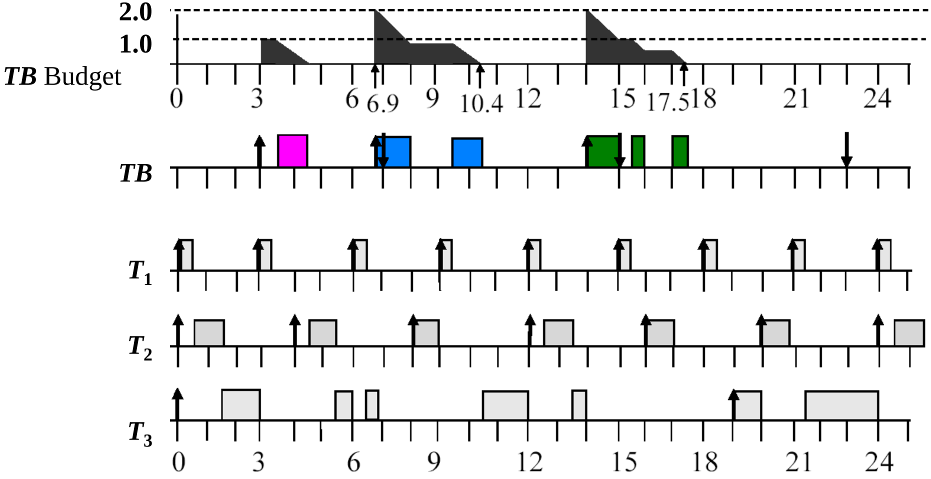
J_{a3} arrives at t_3**=14**



TBS with EDF Scheduling

<u>Same task set:</u> $\boldsymbol{T_1 = (3,0.5)}$, $\boldsymbol{T_2 = (4,1)}$, $\boldsymbol{T_3 = (19,4.5)}$, but $\boldsymbol{U_S = 0.25}$.

Assume an aperiodic job $\boldsymbol{J_{a1}}$ arrives at $\boldsymbol{t_1 = 3}$ with $\boldsymbol{e_{a1} = 1}$, $\boldsymbol{J_{a2}}$ arrives at $\boldsymbol{t_2 = 6.9}$ with $\boldsymbol{e_{a1} = 2}$, and $\boldsymbol{J_{a3}}$ arrives at $\boldsymbol{t_3 = 14}$ with $\boldsymbol{e_{a3} = 2}$.



Sporadic jobs and the TBS

- Recall that a sporadic job is like an aperiodic job except it has a hard deadline.
  - Assume each sporadic job J_i has a release time r_i, ***wcet*** e_i, and a relative deadline D_i: $J_i = (r_i, e_i, D_i)$
- The TBS assigns deadlines such that the (absolute) deadline d_i assigned any job J_i is
  $d_i = \max(r_i, d_i - 1) + e_i/U_S$ where $d_0=1$.
- Thus the TBS can guarantee the deadlines of all accepted sporadic jobs if it only accepts job J_{i+1} if $r_{i+1} + D_{i+1} \leq d_{i+1}$ and rejects it otherwise.

Periodic, Aperiodic, and Sporadic Jobs

- In some systems, we may want to support all three types of jobs: periodic, aperiodic, and sporadic.
- The TBS can do this by accepting all aperiodic jobs and only accepting a sporadic job if its deadline can be met.
- But there is no value in completing sporadic jobs before their deadline, which the TBS will do.
- Thus Deng, Liu, and Sun created the constant utilization server (CUS).
  - Note: We can schedule sporadic jobs with the CUS and aperiodic jobs with the TBS in the same system.

CUS in Deadline-Driven Systems

- Let ***CU*** be a ***CUS*** of size U_S in a task system **T** scheduled with EDF.
- As with the TBS, the server is ready for execution only when it is backlogged.
  - Thus, the server is only suspended when the server is idle.
- **Consumption Rule:** When executing, ***CU*** consumes its budget at the rate of one time unit per unit of execution until the budget is exhausted.
- **Replenishment Rule:**

  R1 Initially, the execution budget $e_s = 0$ and the server's deadline $d_s = 0$.

  R2 At time t when aperiodic job J_i with execution time e_i arrives and the server is idle

  (a) If $t < d_s$, do nothing;

  (b) If $t \geq d_s$, $d_s = t + e_i/U_S$ and $e_s = e_i$.

  R3 At the deadline d_s of the server,

  (a) If the server is backlogged, update the server deadline and budget:

  $$d_s = d_s + e_i / U_S \text{ and } e_s = e_i \; .$$

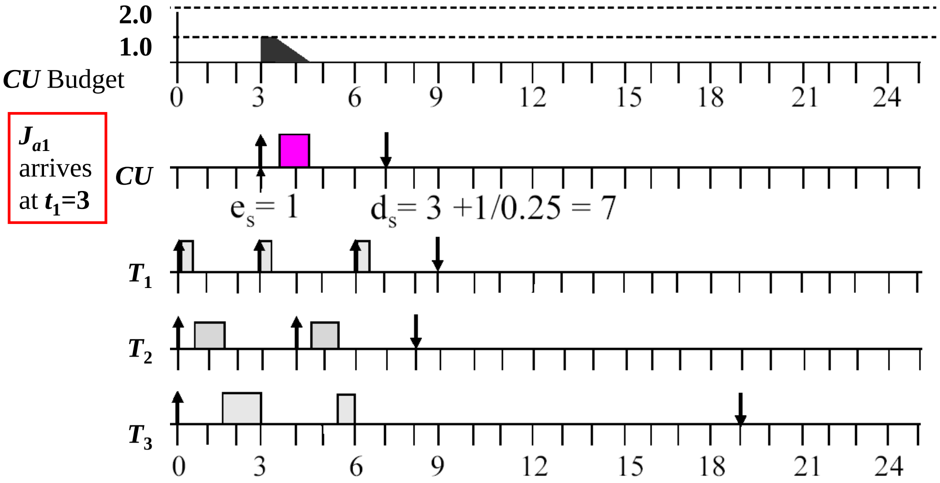
  (b) If the server is idle, do nothing.

CUS with EDF Scheduling

<u>Same task set:</u> $\boldsymbol{T_1 = (3,0.5)}$, $\boldsymbol{T_2 = (4,1)}$, $\boldsymbol{T_3 = (19,4.5)}$, and $\boldsymbol{U_S = 0.25}$.

Assume an aperiodic job $\boldsymbol{J_{a1}}$ arrives at $\boldsymbol{t_1 = 3}$ with $\boldsymbol{e_{a1} = 1}$, $\boldsymbol{J_{a2}}$ arrives at $\boldsymbol{t_2 = 6.9}$ with $\boldsymbol{e_{a1} = 2}$, and $\boldsymbol{J_{a3}}$ arrives at $\boldsymbol{t_3 = 14}$ with $\boldsymbol{e_{a3} = 2}$.

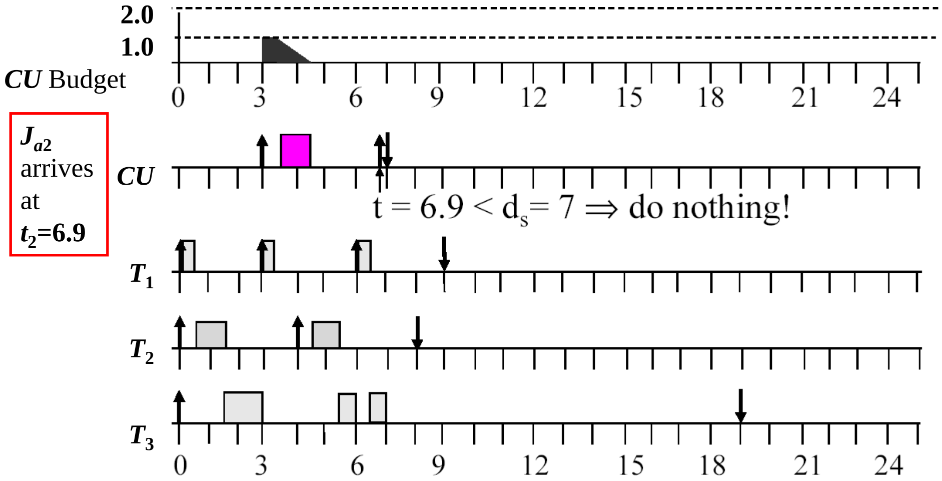
$\boldsymbol{J_{a1}}$ arrives at $\boldsymbol{t_1}$=3



CUS with EDF Scheduling

Same task set: T_1 **= (3,0.5),** T_2 **= (4,1),** T_3 **= (19,4.5),** and U_S **= 0.25.**
Assume an aperiodic job J_{a1} arrives at $t_1 = 3$ with $e_{a1} = 1$, J_{a2} arrives at $t_2 = 6.9$ with $e_{a1} = 2$, and J_{a3} arrives at $t_3 = 14$ with $e_{a3} = 2$.

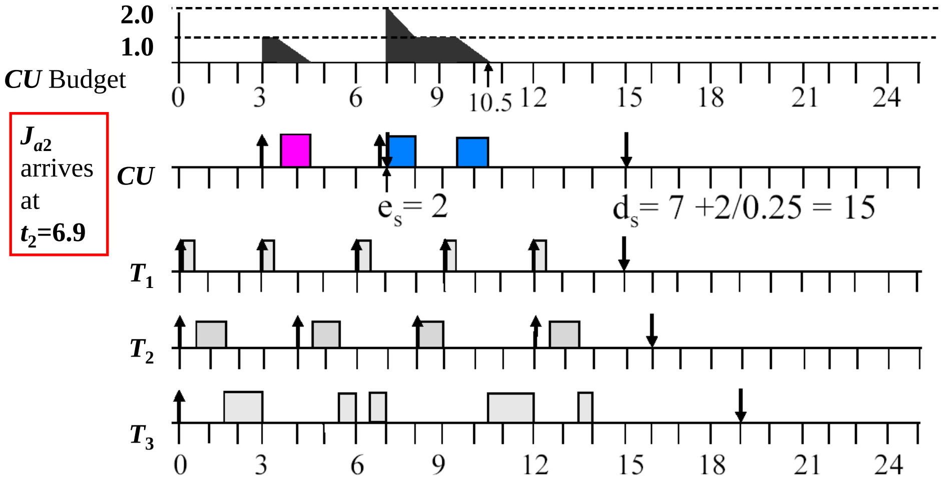
J_{a2} arrives at t_2=6.9



CUS with EDF Scheduling

<u>Same task set:</u> T_1 **= (3,0.5),** T_2 **= (4,1),** T_3 **= (19,4.5),** and U_S **= 0.25.** Assume an aperiodic job J_{a1} arrives at t_1 **= 3** with e_{a1} **= 1**, J_{a2} arrives at t_2 **= 6.9** with e_{a1} **= 2**, and J_{a3} arrives at t_3 **= 14** with e_{a3} **= 2**.

J_{a2} arrives at t_2**=6.9**

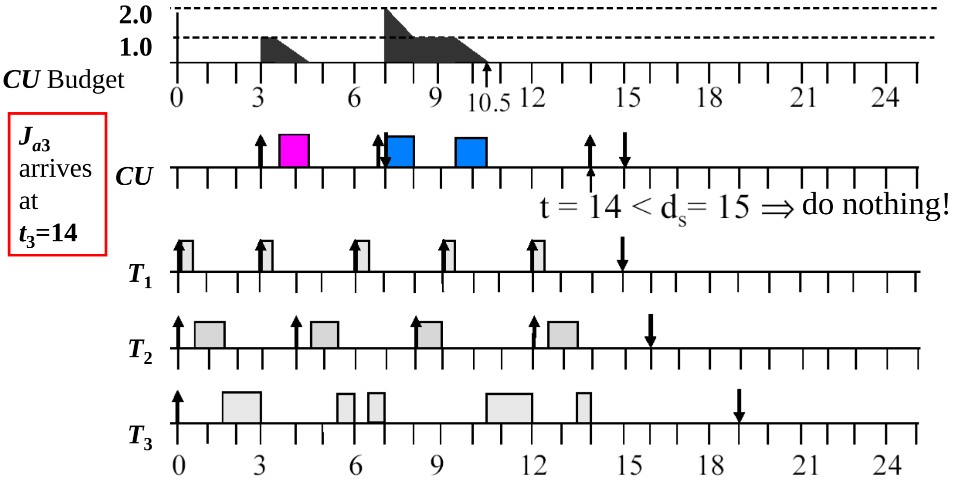


CUS with EDF Scheduling

<u>Same task set:</u> $\boldsymbol{T_1}$ **= (3,0.5),** $\boldsymbol{T_2}$ **= (4,1),** $\boldsymbol{T_3}$ **= (19,4.5),** and $\boldsymbol{U_S}$ **= 0.25.**

Assume an aperiodic job $\boldsymbol{J_{a1}}$ arrives at $\boldsymbol{t_1}$ **= 3** with $\boldsymbol{e_{a1}}$ **= 1**, $\boldsymbol{J_{a2}}$ arrives at $\boldsymbol{t_2}$ **= 6.9** with $\boldsymbol{e_{a1}}$ **= 2**, and $\boldsymbol{J_{a3}}$ arrives at $\boldsymbol{t_3}$ **= 14** with $\boldsymbol{e_{a3}}$ **= 2**.

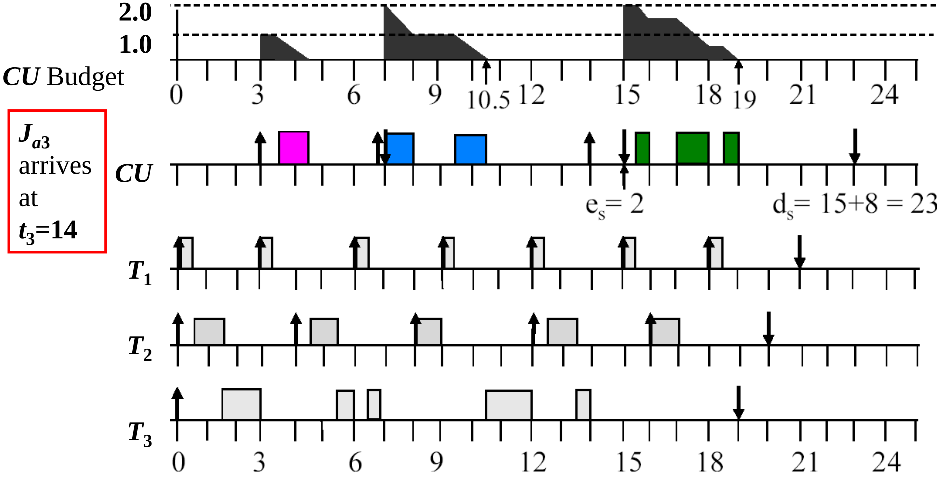
$\boldsymbol{J_{a3}}$ arrives at $\boldsymbol{t_3}$**=14**



CUS with EDF Scheduling

<u>Same task set:</u> T_1 **= (3,0.5),** T_2 **= (4,1),** T_3 **= (19,4.5),** and U_S **= 0.25.**
Assume an aperiodic job J_{a1} arrives at t_1 **= 3** with e_{a1} **= 1**, J_{a2} arrives at t_2 **= 6.9** with e_{a1} **= 2**, and J_{a3} arrives at t_3 **= 14** with e_{a3} **= 2**.

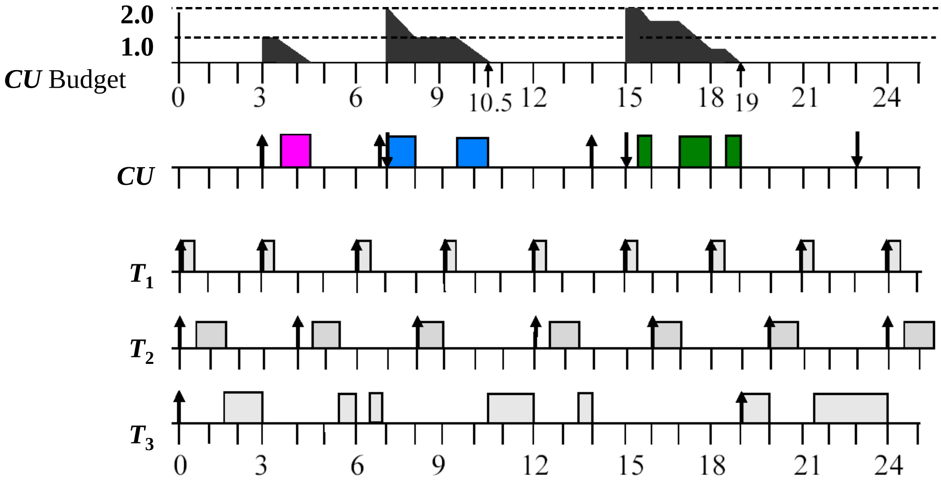
J_{a3} arrives at t_3**=14**



CUS with EDF Scheduling

<u>Same task set:</u> $\boldsymbol{T_1 = (3,0.5)}$, $\boldsymbol{T_2 = (4,1)}$, $\boldsymbol{T_3 = (19,4.5)}$, and $\boldsymbol{U_S = 0.25}$.

Assume an aperiodic job $\boldsymbol{J_{a1}}$ arrives at $\boldsymbol{t_1 = 3}$ with $\boldsymbol{e_{a1} = 1}$, $\boldsymbol{J_{a2}}$ arrives at $\boldsymbol{t_2 = 6.9}$ with $\boldsymbol{e_{a1} = 2}$, and $\boldsymbol{J_{a3}}$ arrives at $\boldsymbol{t_3 = 14}$ with $\boldsymbol{e_{a3} = 2}$.



Comments on the CUS

- Deadlines and replenish amounts are the same in both servers.
- The main difference between the CUS and the TBS is that the CUS never replenishes the server's budget early.
- Thus, the TBS actually yields better average response times for aperiodic jobs (and it was created before the CUS…)
- Moreover, it would appear that the TBS may be able to accept more sporadic jobs than the CUS.
- The value of the CUS is not clear, and Liu does a terrible job arguing for it!

TBS and CUS Summary

- Both servers can be modified to support the case when the ***wcet*** of aperiodic jobs is unkown:
  - fix the execution budget at some value es and assume the server has a period of $\boldsymbol{e_s/U_S}$.
- Both servers can be modified to "reclaim unused resources" when the actual execution time $\boldsymbol{e}$ is less than the ***wcet*** $\boldsymbol{e_s}$ that we assumed:
  - reduce the current deadline of the server by $\boldsymbol{(e_s - e)/U_S}$ units before replenishing the budget.
- We can have multiple TBS/CUS servers as long as the total processor utilization/density is not greater than 1.

Readings

Liu’s book:

- Chapter 7, Section 1: Introduction.
- Chapter 7, Section 2: Deferrable servers.
- Chapter 7, Section 3: Sporadic servers.
- Chapter 7, Section 4: Constant utilization and total bandwidth servers. (We will skip weighted fair queuing, since we are covering proportional-share scheduling, which is similar.)

Other notes

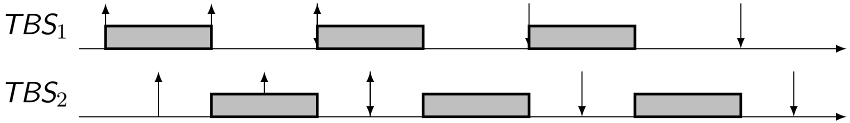
- It is possible to remove the "restriction" of knowing execution times of aperiodic jobs. How?
- Non-preemptible version of TBS is called virtual clock algorithm and is often used in scheduling of packets in switched networks.

Fairness and Starvation

- **Fairness** of the scheduling algorithm:
  The fraction time of processor time in the interval attained by each server that is backlogged throughout the interval is proportional to the server size.
- For many applications (e.g., data transmission in switched networks), fairness is important.
- TBS is unfair! See examples on next slides.
- CUS is fair.
- Fair version of TBS is called *weighted fair-queueing*.

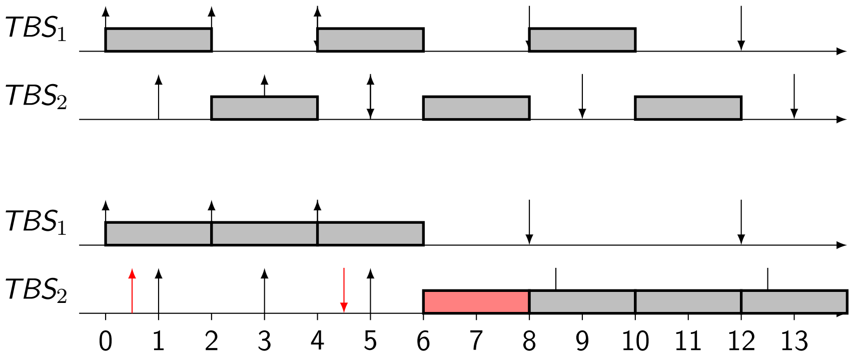
Fairness of TBS

$U_{TBS_1} = 0.5,\ U_{TBS_2} = 0.5,\ e_i = 2$



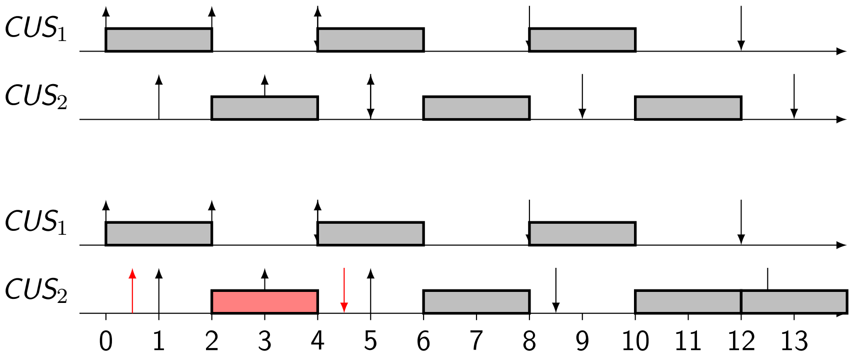
Fairness of TBS

$U_{TBS_1} = 0.5,\ U_{TBS_2} = 0.5,\ e_i = 2$



Fairness of CUS

$U_{CUS_1} = 0.5,\ U_{CUS_2} = 0.5,\ e_i = 2$



Constant Bandwidth Server

- Implemented in Linux as SCHED_DEADLINE scheduling policy.
- Abeni and Buttazzo, 1998
- Similar to TBS and CUS, but limits the task execution even in case of task overruns (i.e., estimated WCET $<$ actual WCET)
  - Usage: one task per server (TBS and CUS can serve multiple different tasks)
- designed for multimedia applications
  - sporadic (hard) tasks
  - soft tasks: mean execution, interarrival times, not fixed
  - periodic tasks
- assign maximum bandwidth of CPU to each soft task
- handles overload of aperiodics
  - limited by assigned bandwidth
  - might slow down the task, but not impair other tasks
- EDF based