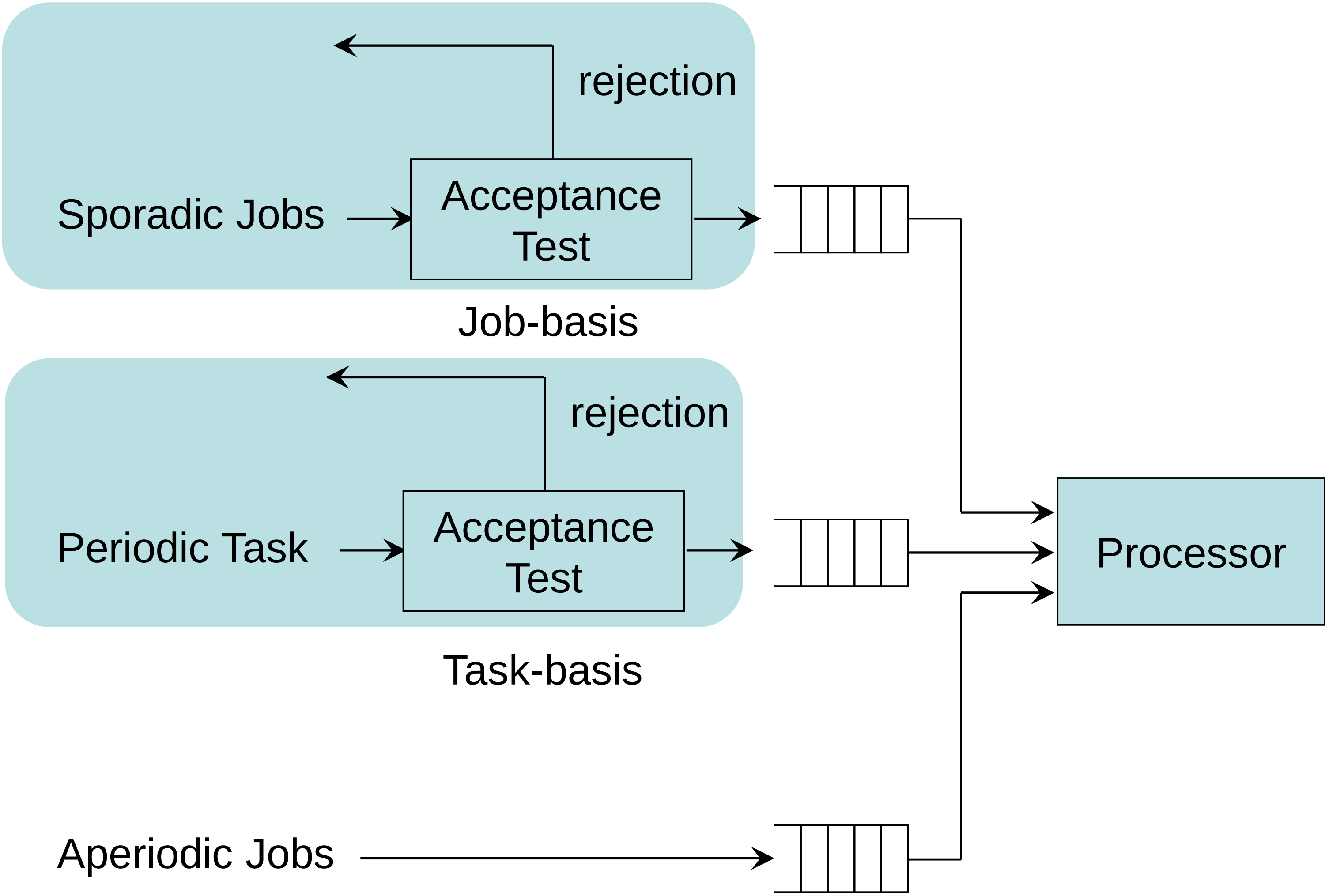
Aperiodic and Sporadic Job Scheduling

Embedded OS Implementation

Prof. Ya-Shu Chen
National Taiwan University
of Science and Technology


rejection
Sporadic Jobs
Acceptance Test
Job-basis
rejection
Periodic Task
Acceptance Test
Task-basis
Processor
Aperiodic Jobs

Aperiodic and Sporadic Jobs

- Definition
  - Jobs of aperiodic and sporadic tasks
    - Inter-arrival times are arbitrary
  - Aperiodic Jobs
    - Assigned to either no deadlines or soft deadlines
  - Sporadic Jobs
    - Assigned to hard deadlines

Aperiodic and Sporadic Jobs

- Handling of jobs
  - Aperiodic jobs
    - In a best-effort fashion
    - Accept anyway
  - Sporadic jobs
    - Accept jobs if deadline satisfaction is guaranteed
    - Otherwise, reject jobs

Aperiodic and Sporadic Jobs

- Correctness
  - All accepted sporadic jobs and periodic jobs meet their respective deadlines

Aperiodic and Sporadic Jobs

- Optimality
  - Aperiodic jobs
    - Minimizing the average response time of jobs
  - Sporadic jobs
    - Accept jobs if and only if they can be scheduled by some means

Aperiodic and Sporadic Jobs

- What we are going to talk about are…
  - How to handle aperiodic jobs
    - Reserve a portion of CPU power for aperiodic jobs in fixed-priority and deadline-driven systems
    - Try to deliver high responsiveness
    - Schedulability of periodic tasks must not be affected
  - How to handle sporadic jobs
    - Employ mechanism proposed for the above
    - Test whether deadlines of sporadic jobs can be met

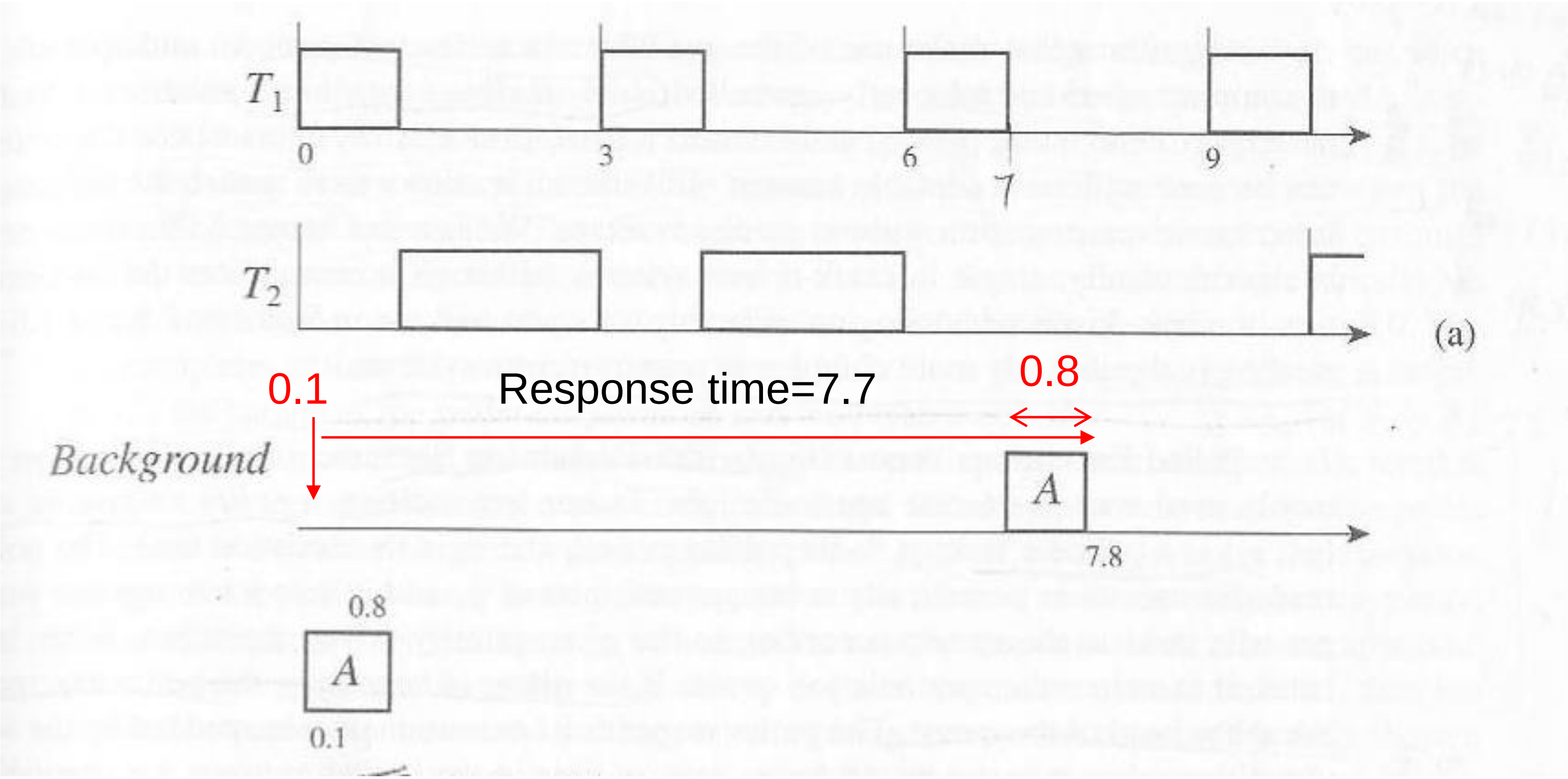
Handling Aperiodic Jobs

- Approaches
  - Background execution
    - Improvement: slack stealing
  - Interrupt-driven execution
    - Improvement: slack stealing
  - Polled execution
    - Improvement: Bandwidth-preserving servers

Handling Aperiodic Jobs

- Background execution
  - Handle aperiodic jobs whenever there is no periodic jobs to execute
    - Extremely simple
    - Always produce correct schedule
    - Poor response time

Background Execution

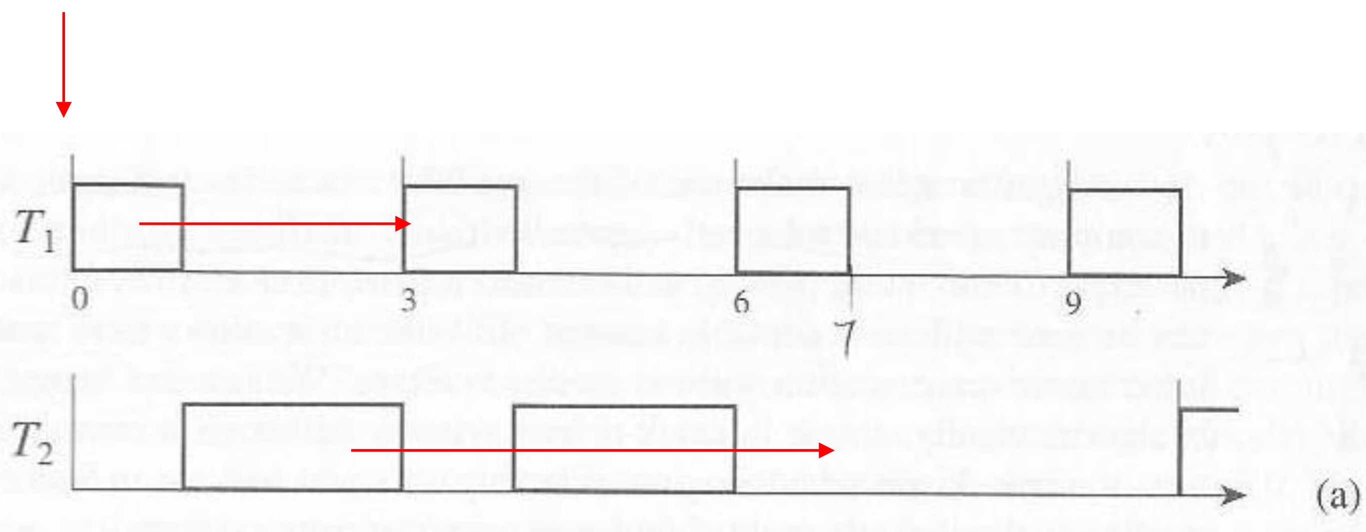


Handling Aperiodic Jobs

- Interrupt execution
  - An obvious extension to background execution
  - On arrivals, aperiodic jobs immediately interrupts the execution of any periodic jobs
  - Fastest response time
  - Potentially damage the schedulability of periodic jobs

Interrupt Execution

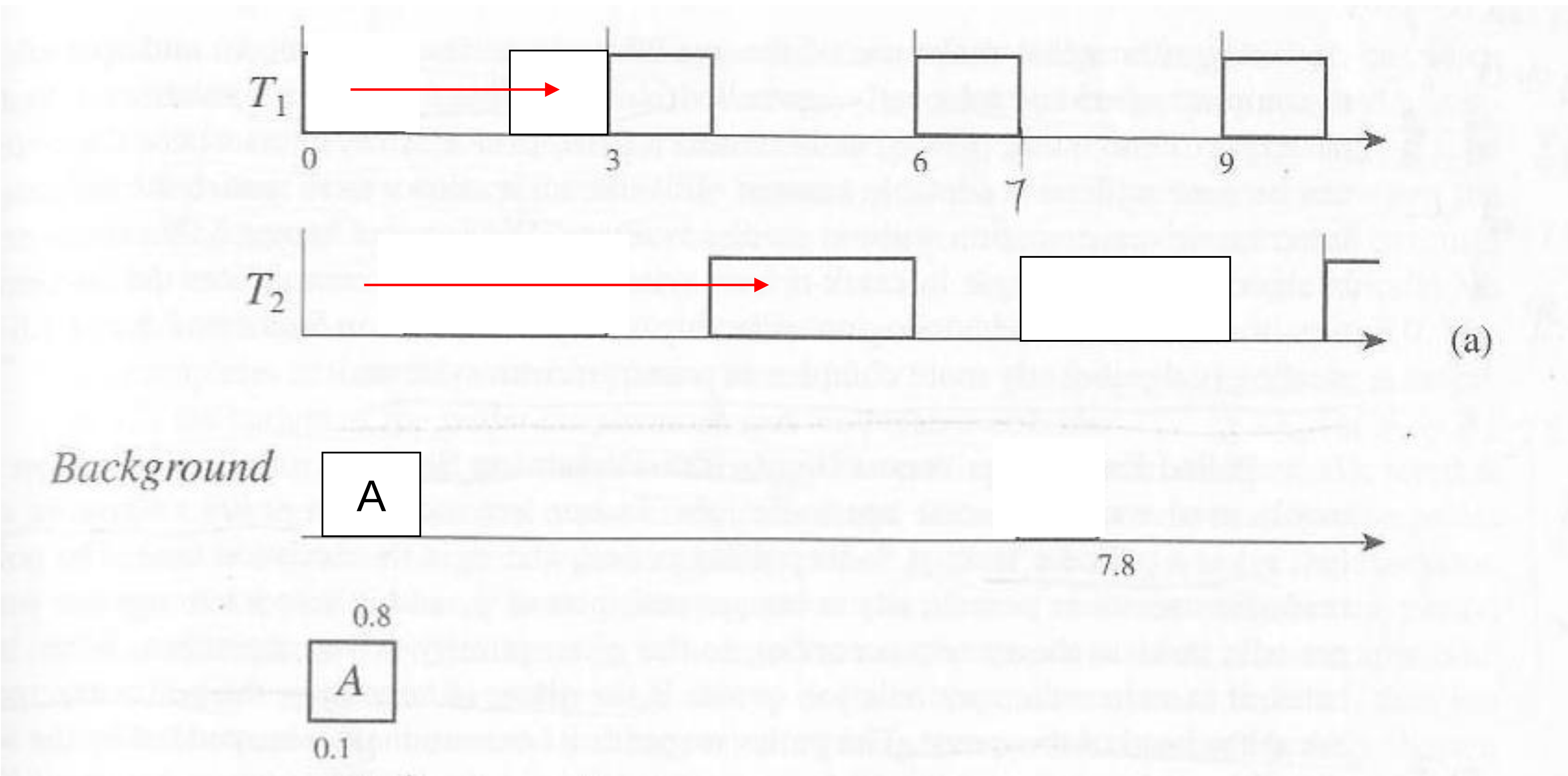
Execution time=2.1 *



Handling Aperiodic Jobs

- Slack stealing
  - To postpone periodic jobs when it is safe to do so
  - Algorithmically simple, but hard to implement

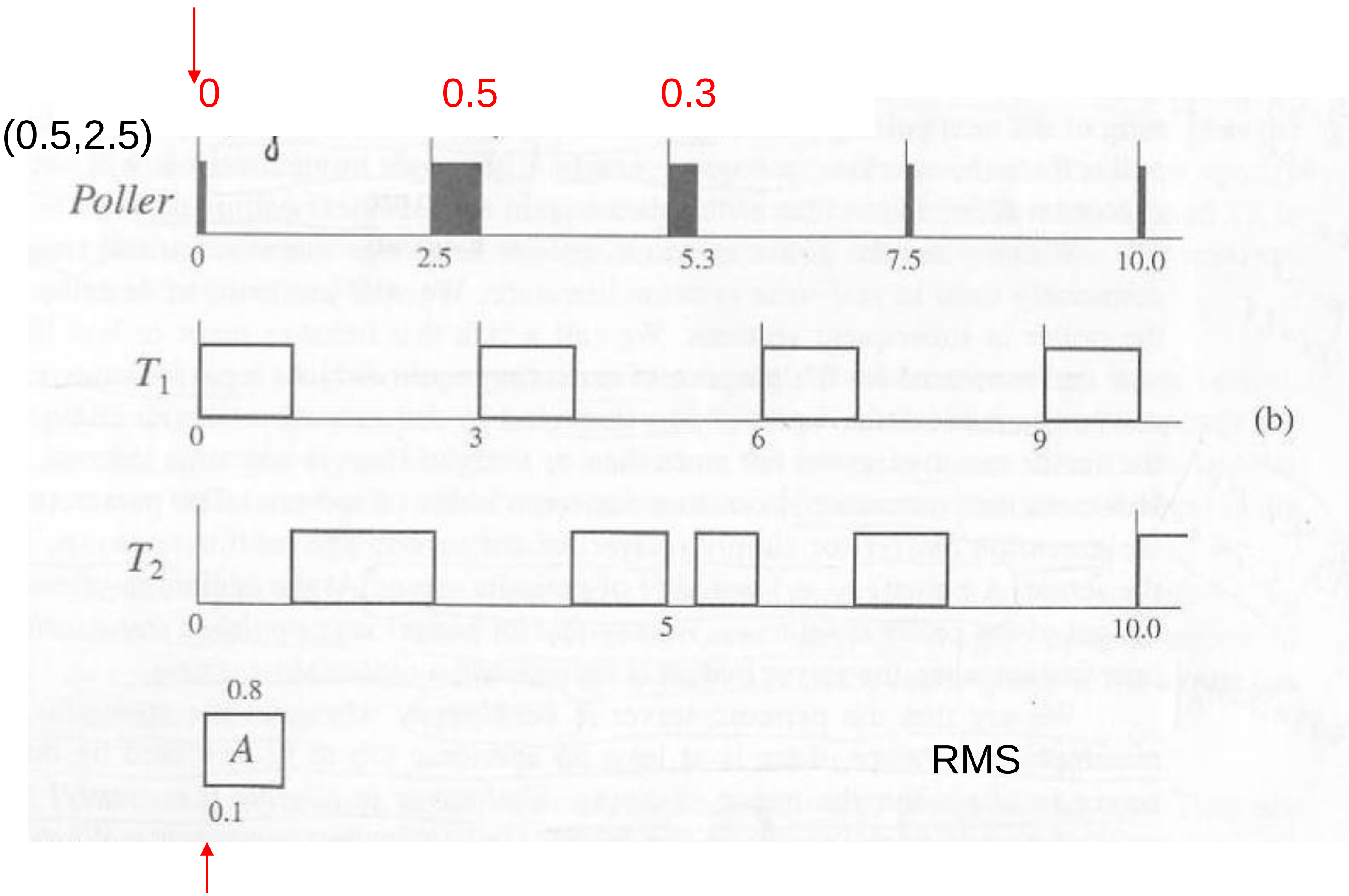
Slack Stealing



Handling Aperiodic Jobs

- Polled execution
  - A purely periodic task (polling server) to serve a queue of aperiodic jobs
  - When the polling server gains control of the CPU, it services aperiodic jobs in the queue
  - If the queue becomes empty, the polling server suspends immediately
    - The queue is not checked until the next polling period

The polling server drops all its budget at time 0 since there is no ready aperiodic jobs



Aperiodic job A arrives at time 0.1

Handling Aperiodic Jobs

- Polling servers are easy to implement and to analyze
- However, a polling server lost all its budget once the queue is empty
  - If we can *preserve* the residual budget, aperiodic jobs can be serviced and their response time can be shortened

Handling Aperiodic Jobs

- Bandwidth-preserving servers
  - Deferrable servers
  - Sporadic servers
  - Constant utilization servers, total bandwidth servers
- These servers aim at improving responsiveness by means of:
  - Preserving budget whenever possible
  - Aggressively replenish

Handling Aperiodic Jobs

- Terminology (1/3)
  - Periodic jobs
    - Jobs of periodic tasks
  - Server
    - A task servicing sporadic jobs
    - Can be either periodic servers or sporadic servers
    - Characterized by (c,p)
      - Don't confuse with a periodic task, a server does not stands for a periodic task
    - c→budget, c/p→server size

Handling Aperiodic Jobs

- Terminology (2/3)
  - Backlogged
    - A server is backlogged if there are some ready jobs in its queue
  - Eligible
    - A server is eligible if it has budget and is backlogged

Handling Aperiodic Jobs

- Terminology (3/3)
  - Budget consumption rules
    - Define how budget is consumed when servicing jobs
  - Budget replenishment rules
    - Define how budget is replenished
  - Server execution
    - When a server is eligible, it is scheduled with other periodic tasks as if it was a periodic task
    - When budget is exhausted, the server is suspended immediately

Deferrable Servers

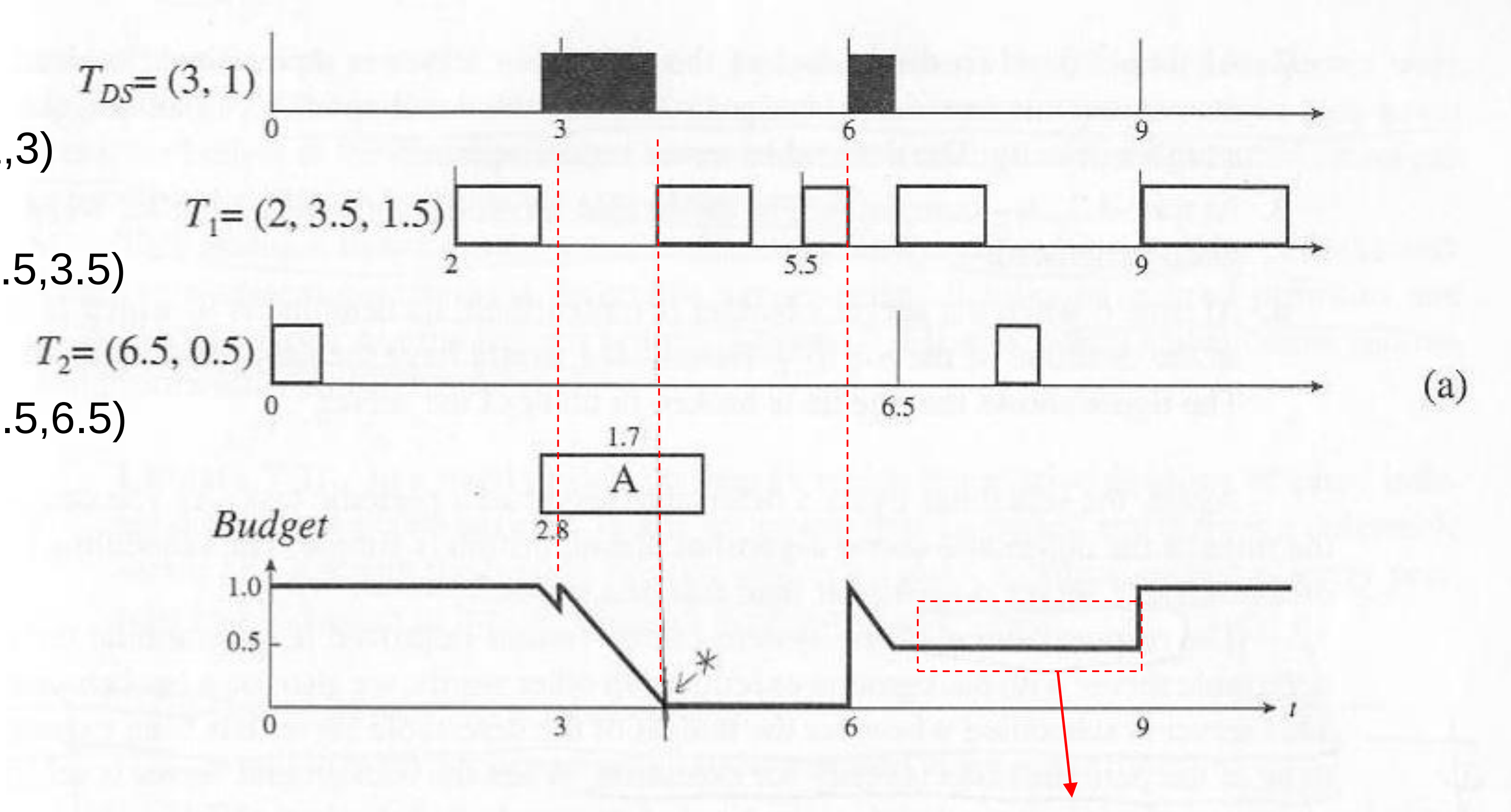
- Let a deferrable server be denoted by (c_s, p_s)
- Consumption rules:
  - Budget is consumed at rate of 1 whenever the server services aperiodic jobs
- Replenishment rules:
  - Budget is set to c_s at $k*p_s$, for $k=0,1,2,\ldots$

Deferrable Servers

(1,3)

(1.5,3.5)

(0.5,6.5)



Preserving budget

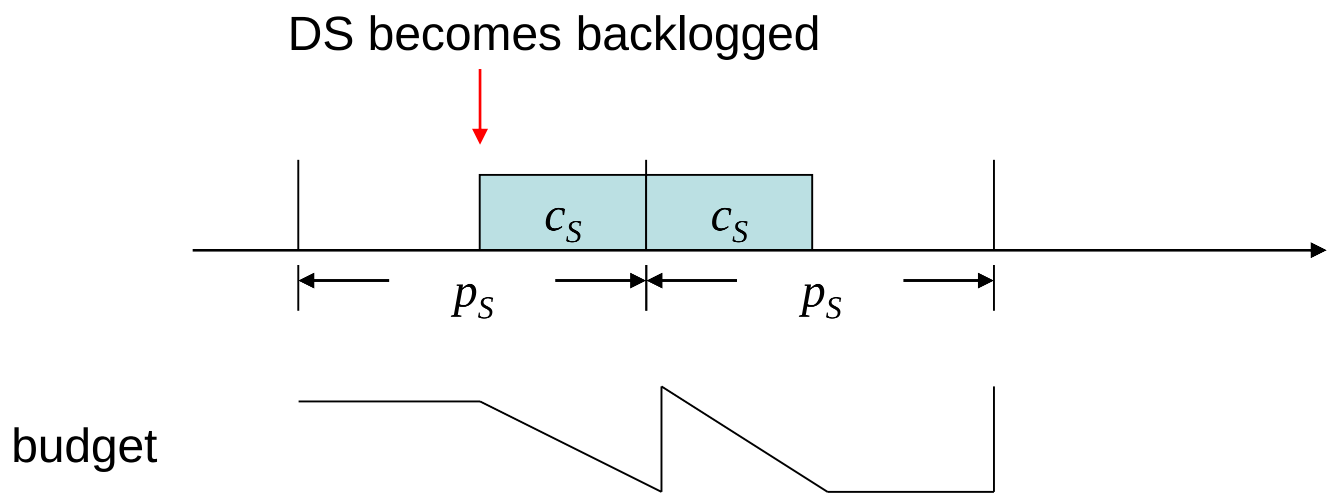
Fixed-priority: priority $T_{DS} \rightarrow T_1 \rightarrow T_2$

Deferrable Servers

- Because budget is preserved, aperiodic jobs may conflict with any periodic jobs at any time
  - Differently, jobs of a purely periodic task are ready at the beginning of every period
- Does critical instants for tasks of a fixed-priority system in which there is a deferrable server exists?

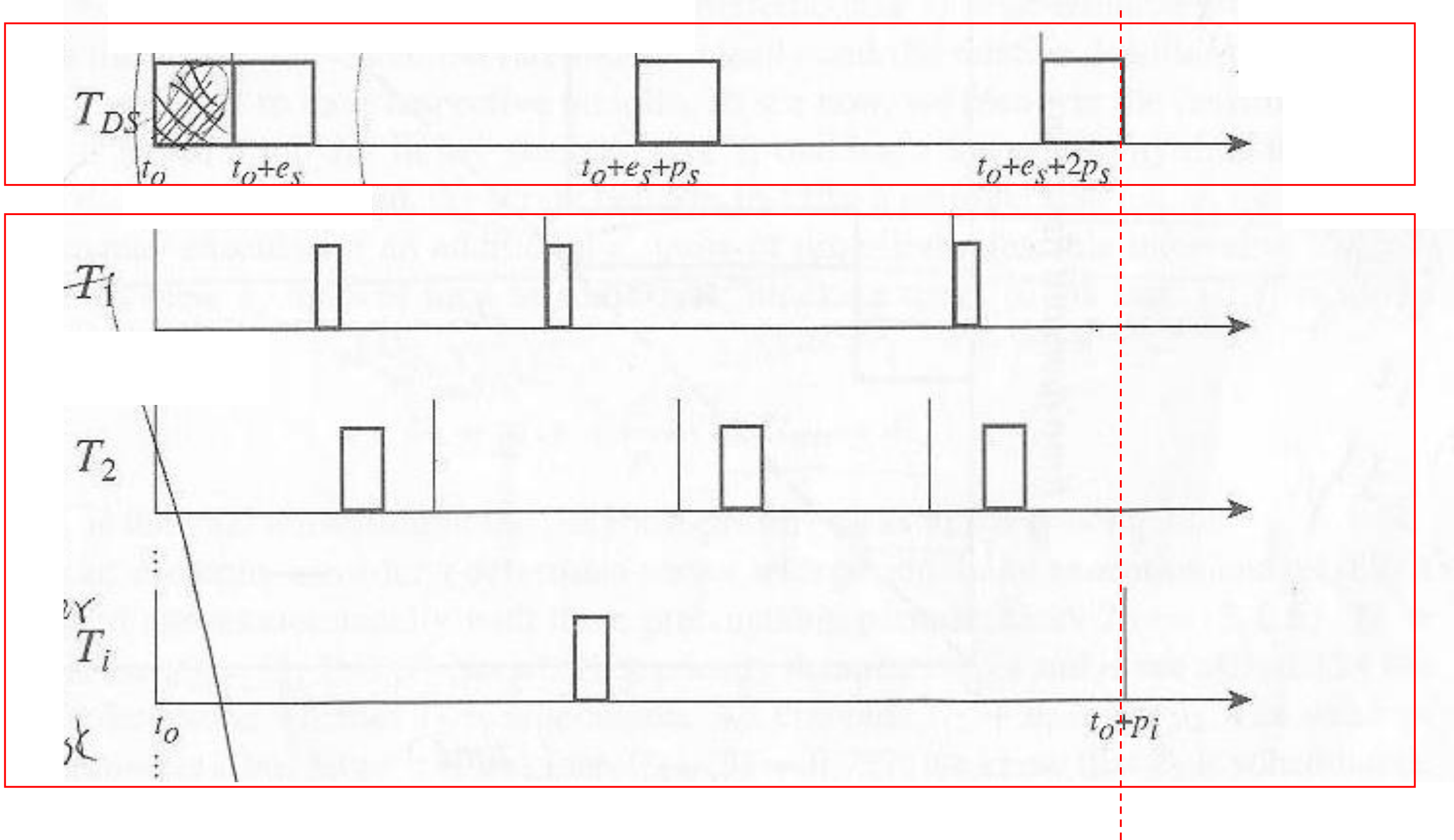
Deferrable Servers

- The below figure shows a scenario in which a deferrable server interferes low-priority tasks most



Deferrable Servers

- *Theorem*: Critical instant of task T_i if a deferrable server is of the highest priority



Deferrable Servers

- Response time analysis
  - For each i, task T_i is schedulable if $R_{i,j}$ converges no later than P_i

$$R_{i,0} = c_i$$

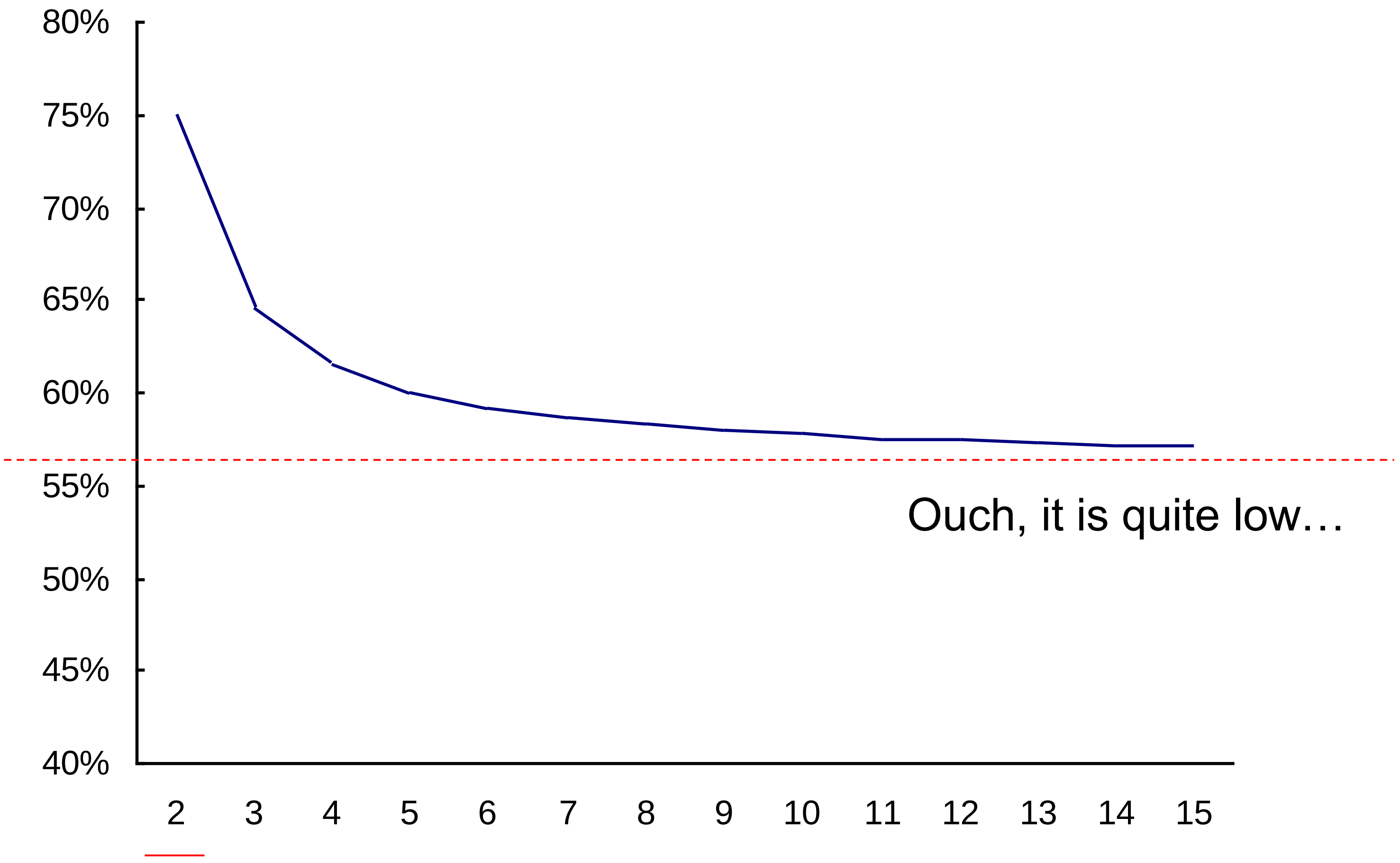
$$R_{i,j} = c_i + c_s + \left\lceil \frac{R_{i,j-1} - c_s}{p_s} \right\rceil \times c_s + \sum_{k=1}^{i-1} \left\lceil \frac{R_{i,j-1}}{p_k} \right\rceil \times c_k$$

Deferrable Servers

- Response time analysis is accurate, but can we have a polynomial time test like the utilization-bound test?
- Bad news:
  - There is no know utilization bound if a deferrable server is of an arbitrary priority
- Good news:
  - Consider a system of n independent, preemptable periodic tasks and $p_s<p_1<\ldots<p_n<2p_s$ and $p_n>p_s+c_s$

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

[Lehoczky87]



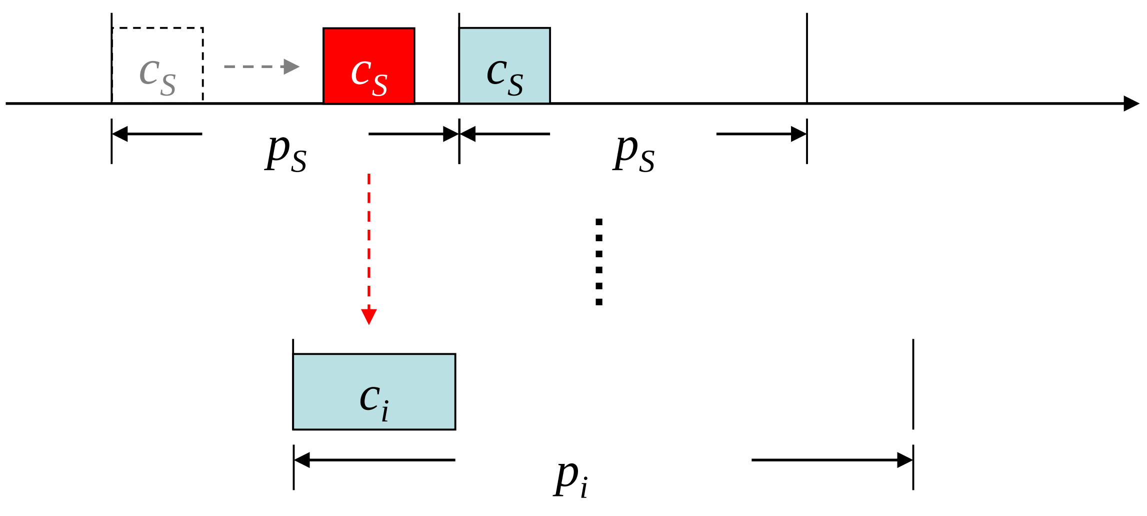
Utilization bound when $U_s = 1/3$

Deferrable Servers

- Yet another good news:
  - We can still make use of the old utilization bound if some conservative considerations are taken

Deferrable Server

- If the deferrable server T_{DS} is of an arbitrary priority, it may impose up to one additional C_s on feasible intervals of jobs of some low-priority task Ti



Deferrable Server

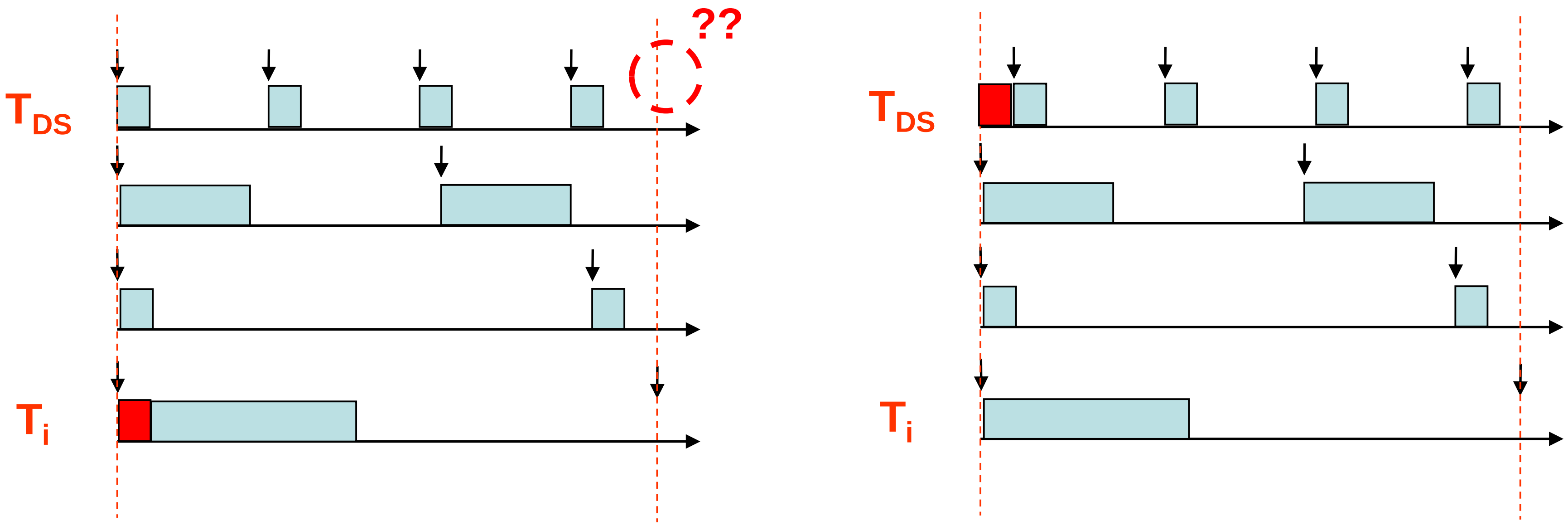
- Task set $\{T_1,\ldots,T_m,T_{DS},T_{m+2},\ldots T_n\}$
  - $\{T_1,\ldots,T_m\}$ is schedulable if $\sum ci/pi \leqq U(m)$
  - For *i=m+1~n*, the reminder of the task set is schedulable if

$$\sum_{j=1}^{i} \frac{c_j}{p_j} + \frac{c_s}{p_i} \leq U(i)$$

  - Let's try $T_1=(0.6,3)$, $T_{DS}=(0.8,4)$, $T_3=(0.5,5)$, $T_4=(1.4,7)$

Deferrable Servers

- Why can we treat the additional c_s as "extra blocking time" of T_i?



Schedulable → schedulable

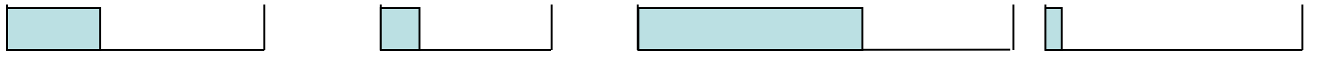
Constant Utilization Servers

- Let a sporadic job is characterized by arrival time a, execution time c, deadline d
  - Let its density be defined as $c/(d-a)$

- *Theorem*: A system of independent sporadic jobs is schedulable if the total density of jobs is no larger than 1 at any time instant

Constant Utilization Servers

- A sporadic task is of a stream of sporadic jobs



- Let a_i be the inter-arrival time between the i-th sporadic job and the (i+1)-th sporadic job, c_i be its computation time damand
- c_i/a_i is referred to the instantaneous utilization of the i-th job

Constant Utilization Servers

- Assumption:
  - All periodic and sporadic jobs must be completed by the arrival of their next successive jobs

- *Corollary*: A periodic system and a collection of sporadic tasks are schedulable by EDF if the sum of utilization of the former and instantaneous utilization of the latter is no greater than 1 at any time

Constant Utilization Servers

- Consumption rules is simple: A server consumes its budget only when it executes
- Replenishment rules: as follows

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

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

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

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

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

R3 At the deadline d of the server,

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

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

[Spuri 96]

Constant Utilization Servers

- The instantaneous utilization of every sporadic job equals to the server size, as is its name
- Budget is replenished only on the deadlines of jobs

Pitfall: A sporadic job may have two "deadlines":

•The deadline comes with the job→da

•The deadline assigned by CUS algorithm→db

The job might not be fulfilled in time if db is later than da

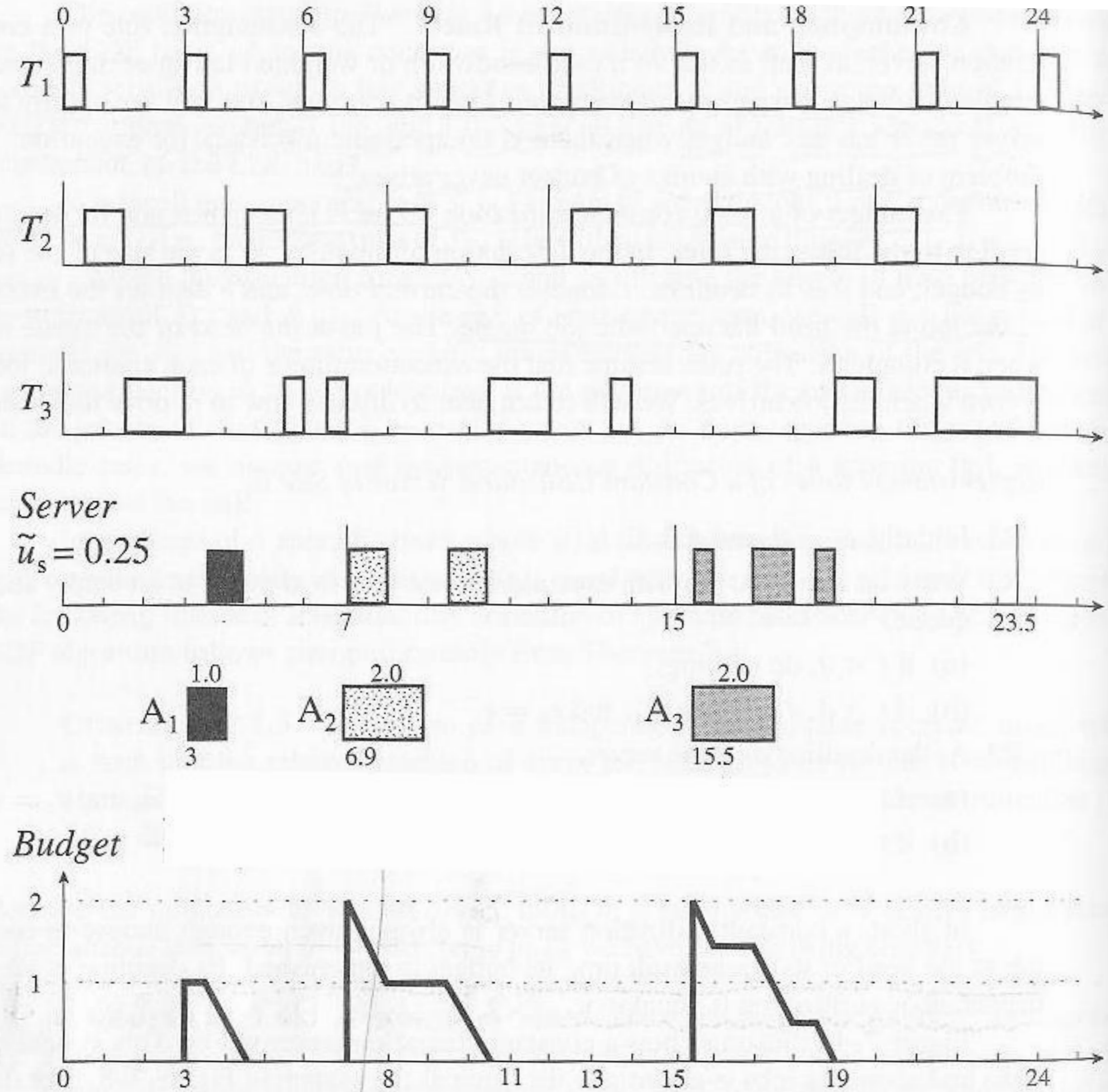


FIGURE 7–13 Example illustrating the operations of constant utilization server: $T_1 = (3, 0.5)$, $T_2 = (4, 1.0)$, $T_3 = (19, 4.5)$.

Total Bandwidth Servers

- CUS replenish budget on deadlines of jobs only
  - Aperiodic jobs arrives before an upcoming server deadline will be pended until budget is replenished at the deadline

- Is it possible to advance the replenishment?
  - Yes: total bandwidth servers

Total Bandwidth Servers

- Consumption rules: the same as those of CUS
- Replenishment rules:

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

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

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

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

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

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

Instantly replenish server budget!

[Deng 97]

Summary

- Serving aperiodic/sporadic jobs in periodic system
  - Priority-driven systems (DS)
  - Deadline-driven systems (CUS)
- One major approach for server design is to emulate a periodic task
  - Budget should be retain as long as possible
  - Budget should be replenished as soon as possible
- Aperiodic jobs are served in best-effort fashion
- Acceptance of sporadic jobs should be tested