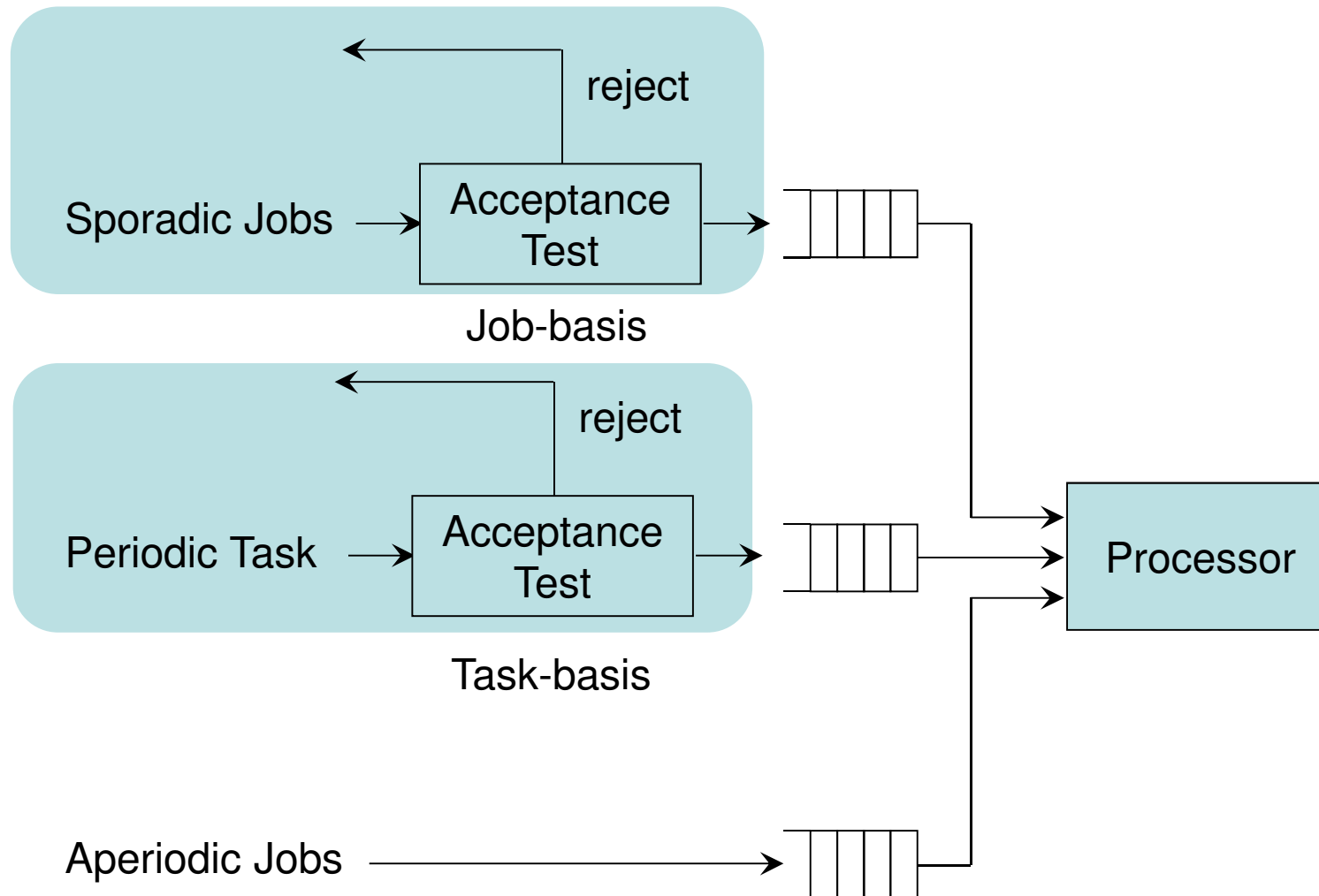


# Aperiodic Job Scheduling

Real-Time and Embedded Operating Systems

Prof. Li-Pin Chang  
ESSLab@NYCU



# 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 them if their deadlines can be satisfied
    - Otherwise, reject them

# Aperiodic and Sporadic Jobs

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

# Aperiodic and Sporadic Jobs

- Optimality
  - Aperiodic jobs
    - Minimal job response times
  - Sporadic jobs
    - No accepted jobs miss their deadlines

# 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 good response
    - Schedulability of periodic tasks must not be affected
  - How to handle sporadic jobs
    - Employ the mechanisms proposed for aperiodic jobs
    - Test whether deadlines of sporadic jobs can be met

# Part A: Handling Aperiodic Jobs

Brute-Force Methods & Server Design



# Handling Aperiodic Jobs

- Approaches

- Background execution
  - Improvement: slack stealing
- Interrupt-driven execution
  - Improvement: slack stealing

} Brute-force methods

- Polled execution

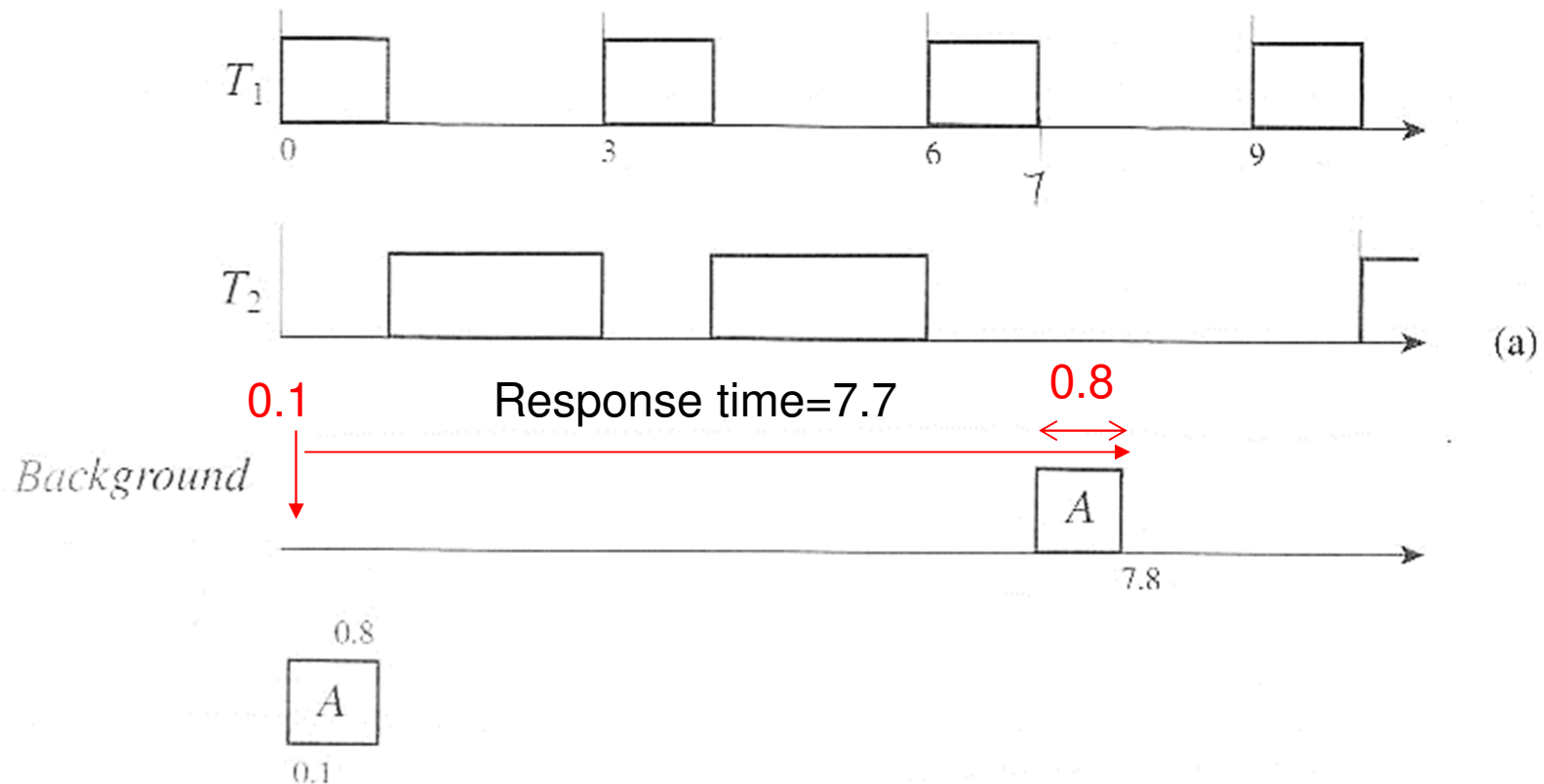
- Improvement: Bandwidth-preserving servers

} server

# Brute-Force Method-1

- Background execution
  - Handle aperiodic jobs whenever there is no periodic jobs to execute
    - Simple
    - Always produce a correct schedule
    - Poor response time (of aperiodic jobs)

# Background Execution

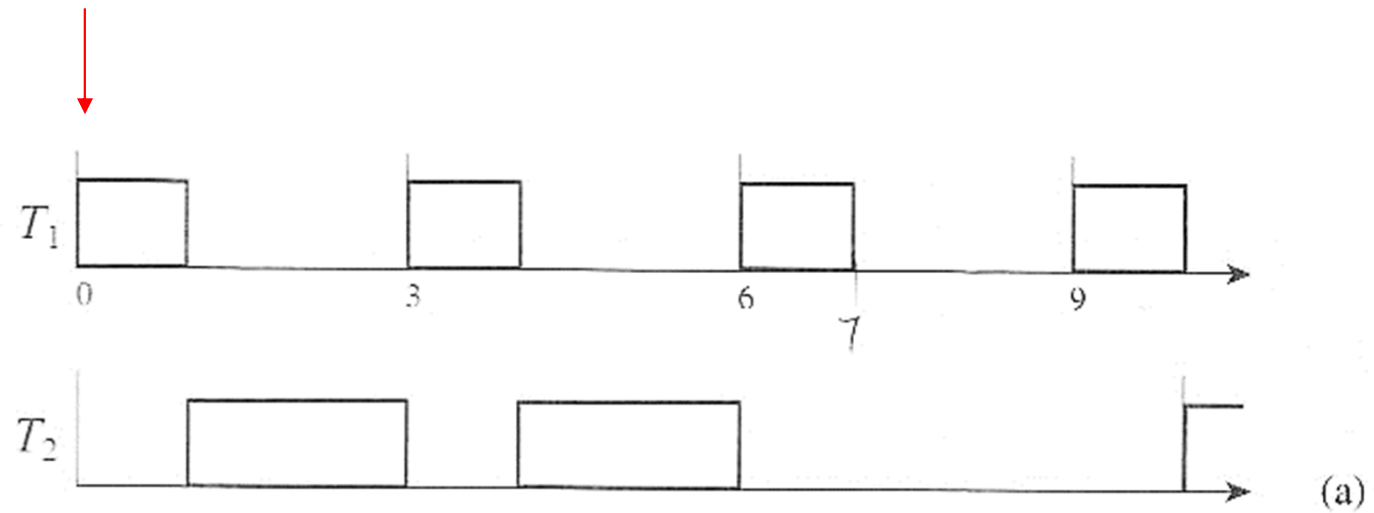


# Brute-Force Method-2

- Interrupt execution
  - Opposite to background execution
  - On arrivals, aperiodic jobs immediately interrupts the currently running periodic job
  - Fastest response time
  - May damage the schedulability of periodic jobs

# Interrupt Execution

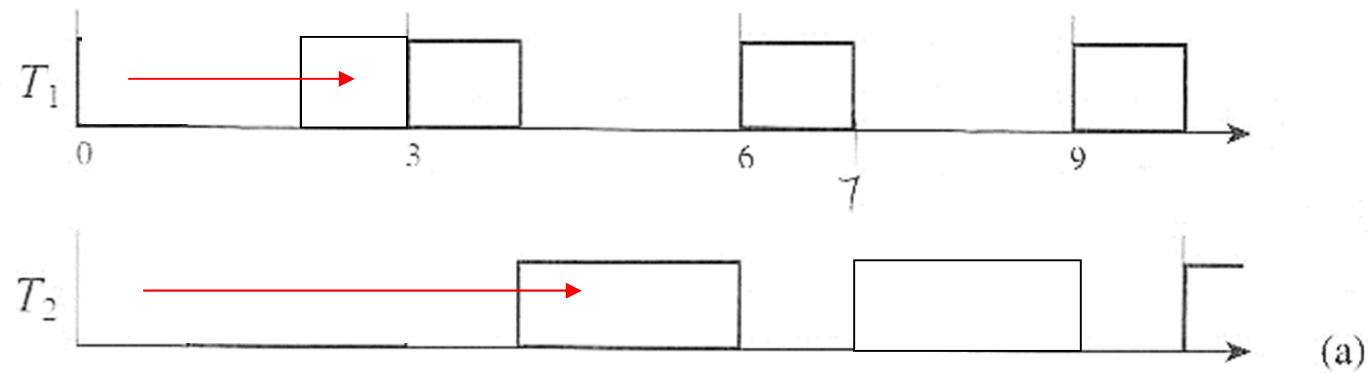
Arrival = 0.1  
If execution time > 2...



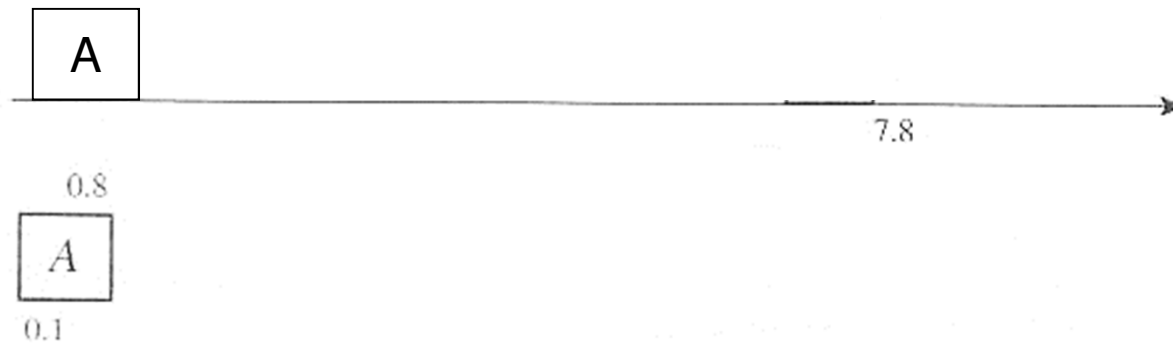
# Improvement

- Slack stealing
  - To postpone periodic jobs when it is safe to do so
  - May be easily implemented in LSF algorithm, but hard to implement in EDF or RM

# Slack Stealing



*Background*

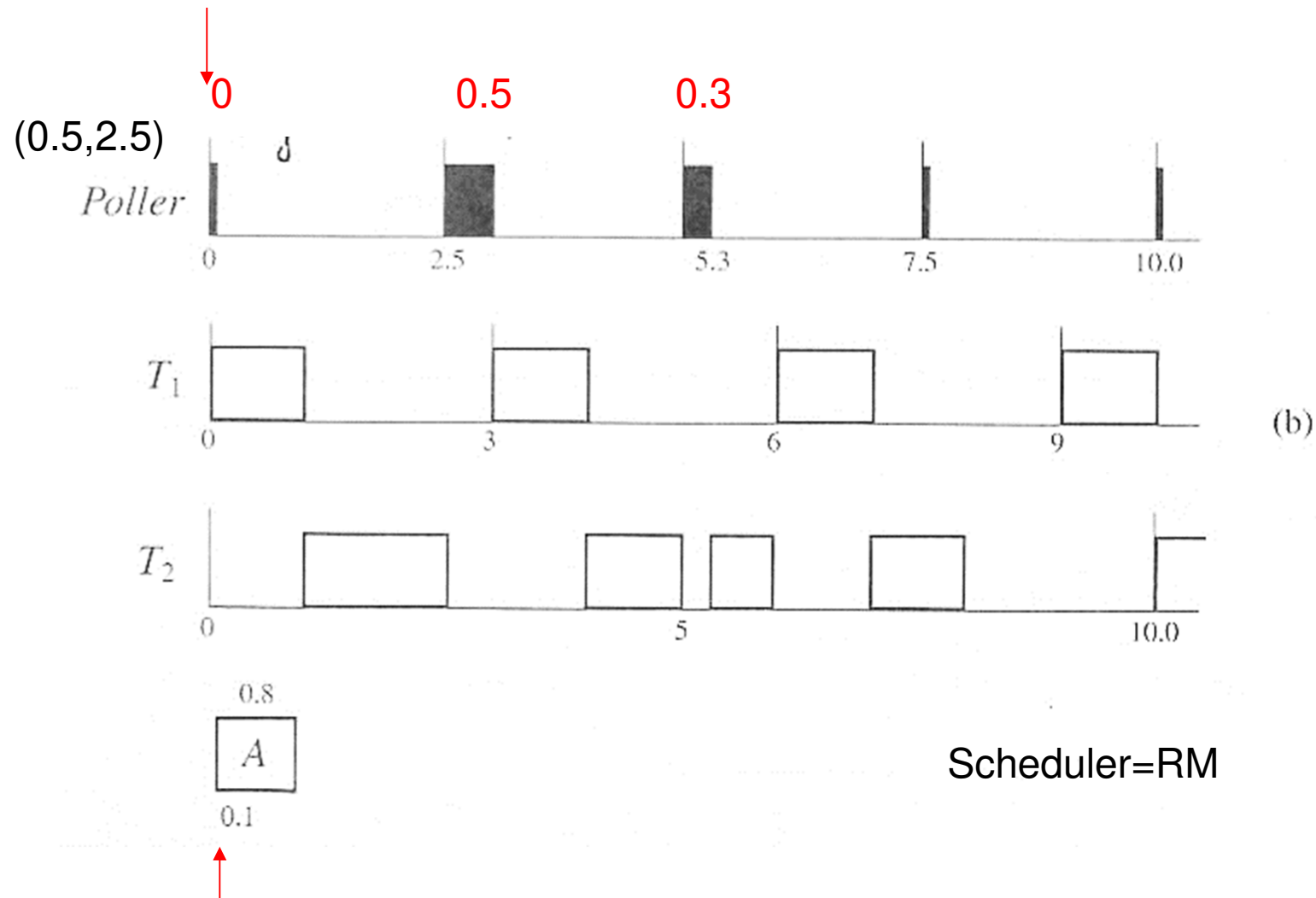


# A Primitive Server Design

- Polled execution
  - A purely periodic task (polling server) that serves 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 loses all its budget at time 0, since there is no ready aperiodic jobs



Aperiodic job A arrives at time 0.1

# Polling Server

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

# Server Terminology

- Terminology (1/3)
  - Server
    - A task that serves aperiodic/sporadic jobs
    - Characterized by  $(c,p)$  in terms of schedulability
      - A server is *not* necessarily a periodic task
  - $c$ : Server budget
    - A server is ready for execution if its budget is non-zero
  - $c/p$ : Server size
    - A fraction of CPU time reserved for the server

# Server Terminology

- 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


# Server Terminology

- Terminology (3/3)
  - Budget consumption rules
    - Define how budget is consumed when serving 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 were a periodic task
    - When budget is exhausted, the server is suspended immediately

# Advanced Server Designs

- Bandwidth-**preserving** servers
  - Deferrable servers
  - Sporadic servers
  - Constant utilization servers, total bandwidth servers, weighted fair-queuing servers
- These servers aim at improving responsiveness by
  - preserving budget as late as possible and
  - replenishing budget as early as possible

# Bandwidth-Preserving Servers

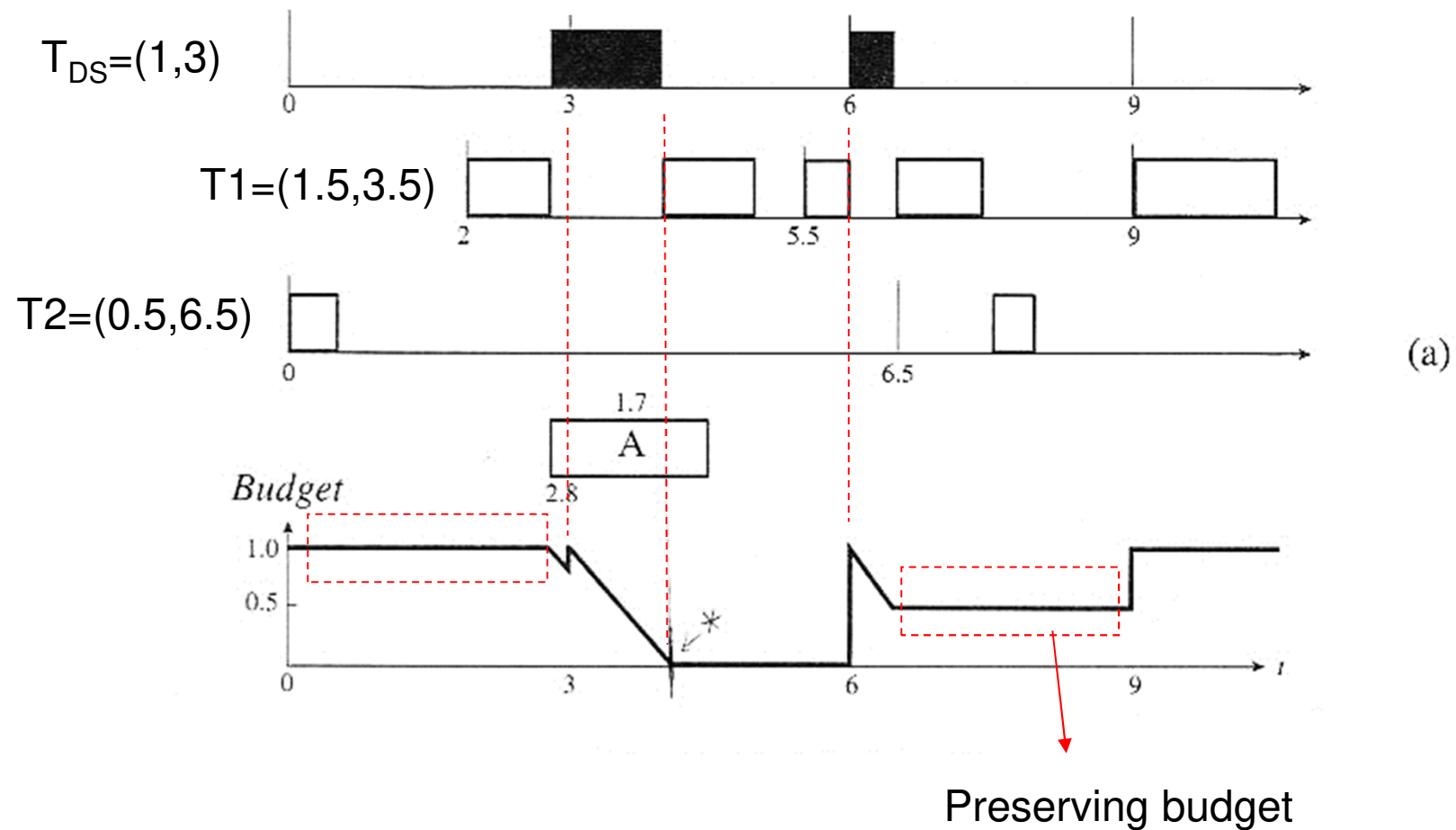
- Deferrable servers 
- Sporadic servers (RM only)
- Priority-exchange servers
- Constant utilization servers and total bandwidth server (EDF only)

# Deferrable Servers

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

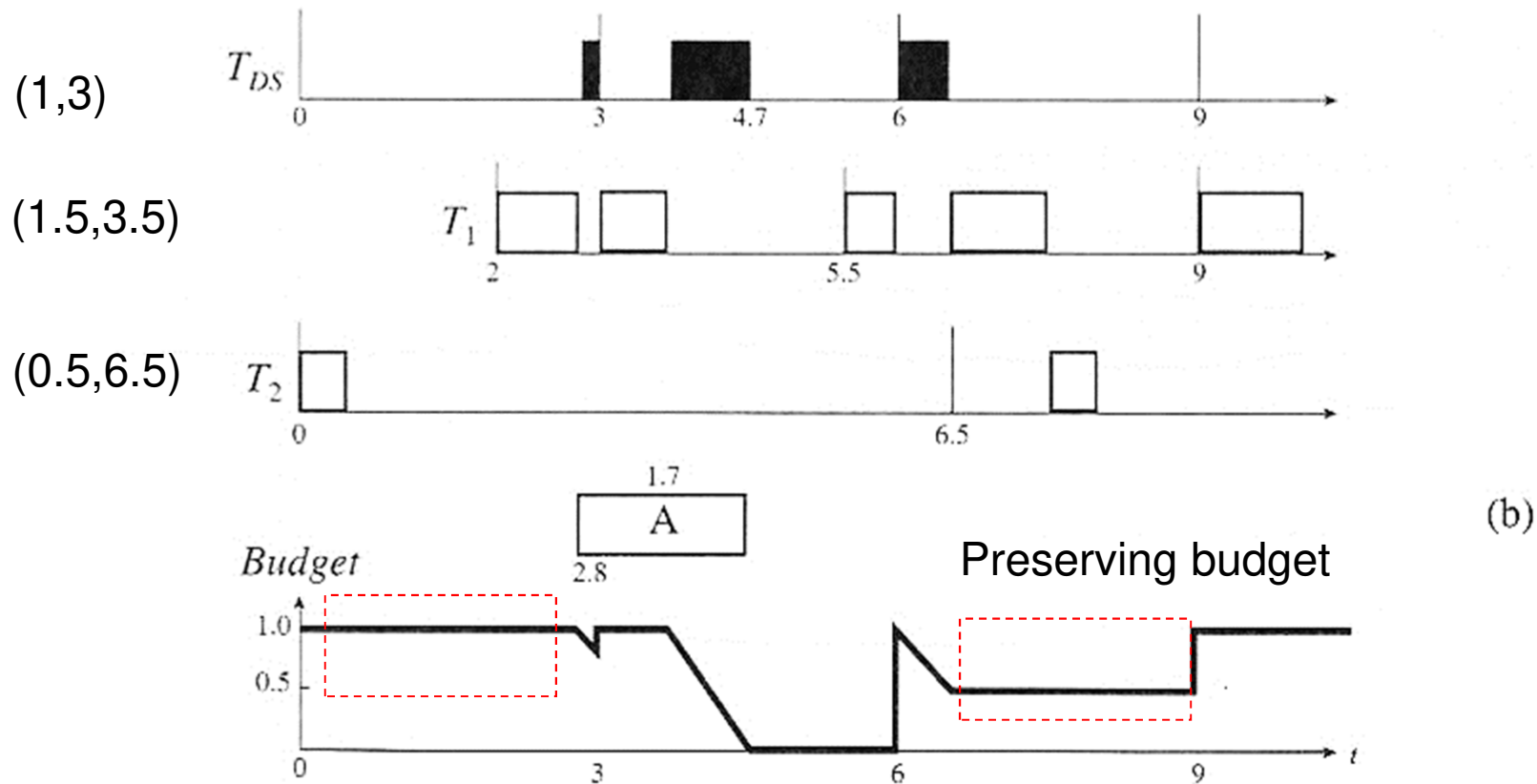


# Deferrable Servers



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

# Deferrable Servers



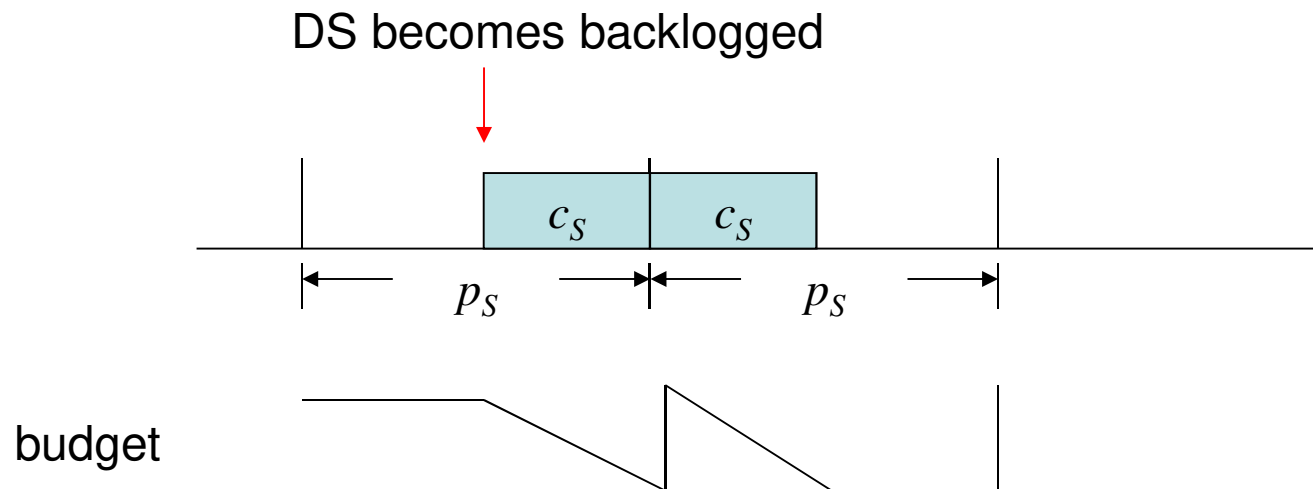
Deadline-driven (EDF): the deadline of a server job is the next replenishment time

# Deferrable Servers

- Because budget is preserved, an aperiodic job may conflict with periodic jobs **at any time**
  - Differently, jobs of a purely periodic task is ready at the beginning of every period
- Does the task critical instant exist in the presence of a deferrable server?

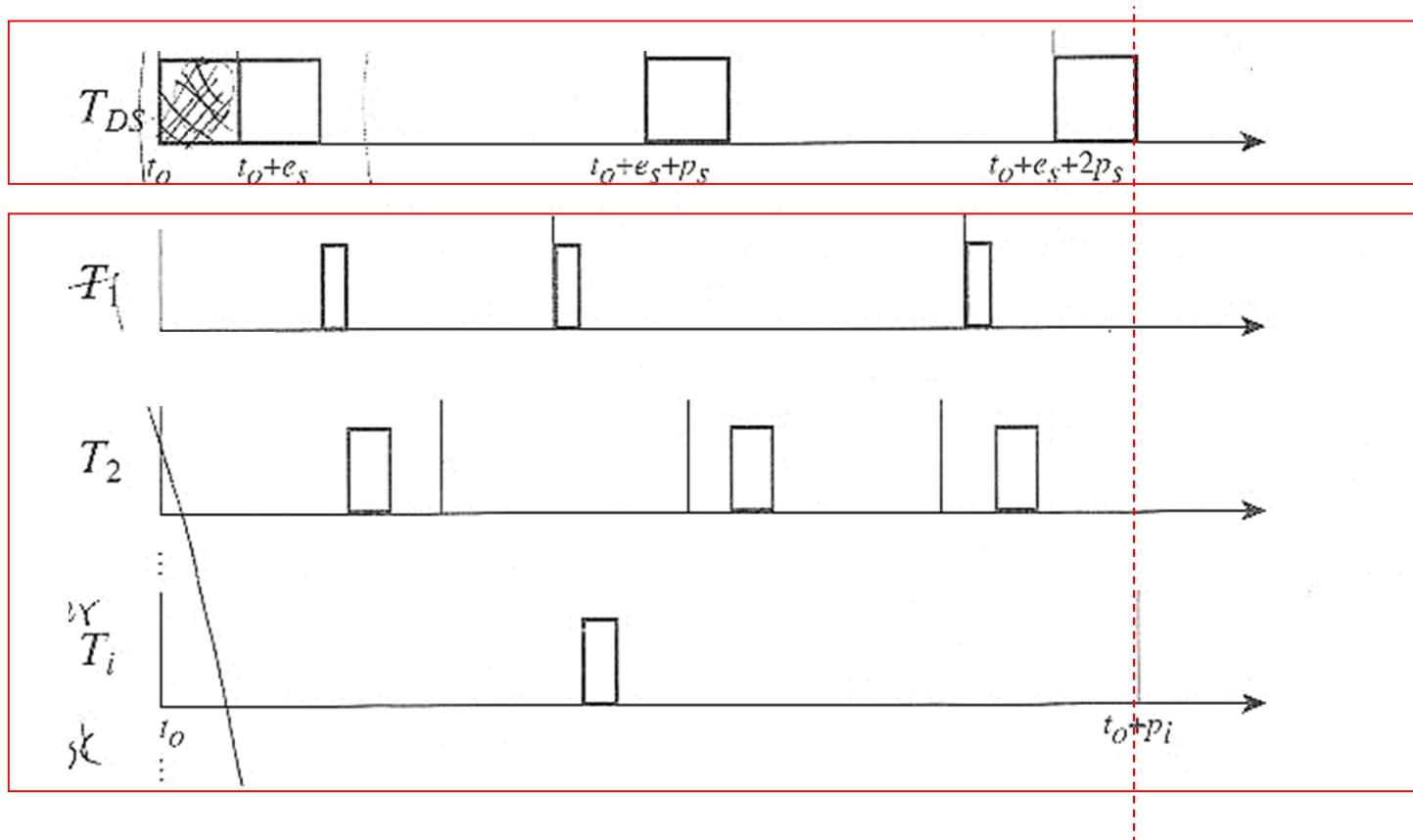
# Deferrable Servers

- The below scenario shows when a deferrable server interferes low-priority tasks the most (aka double hit)



# Deferrable Servers

- Critical instance of a periodic task  $T_i$ 
  - Here, the deferrable server  $T_{DS}$  has the highest priority



# Deferrable Servers

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

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

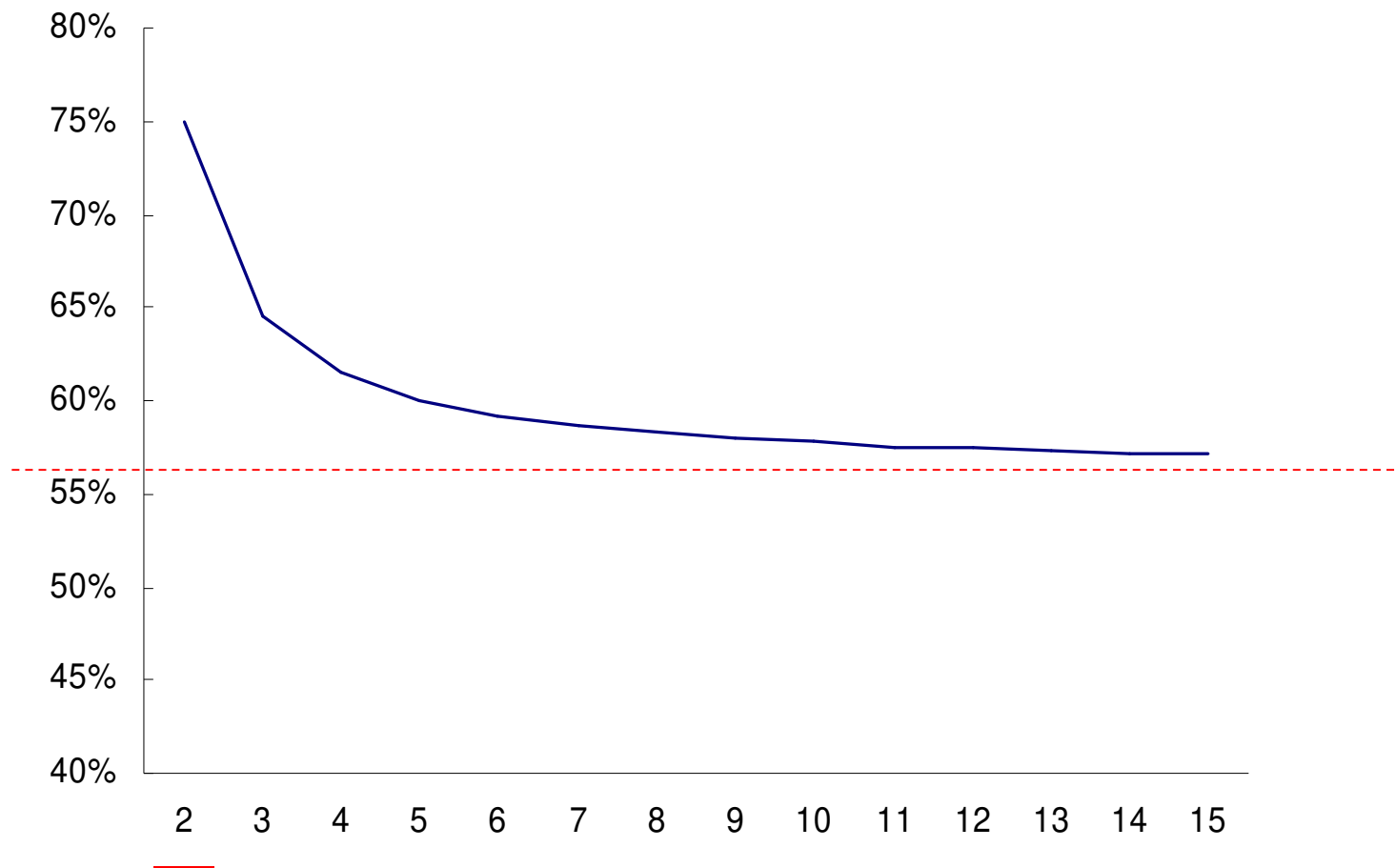
$$R_{i,j} = c_i + \boxed{c_s + \left\lceil \frac{R_{i,j-1} - c_s}{p_s} \right\rceil \times c_s} + \sum_{k=1}^{i-1} \left\lceil \frac{R_{i,j-1}}{p_k} \right\rceil \times c_k$$

# Deferrable Servers

- Response time analysis is accurate, but can we use the utilization bound as a quick test?
- Bad news:
  - There is no known utilization bound if the priority of the deferrable server is arbitrary
- Good news (not really):
  - Consider a system of  $n$  independent, preemptible periodic tasks and  $p_s < p_1 < \dots < p_n < 2p_s$  and  $p_n > p_s + c_s$

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

[Lehoczky87] 31



Utilization bound when  $U_s = 1/3$

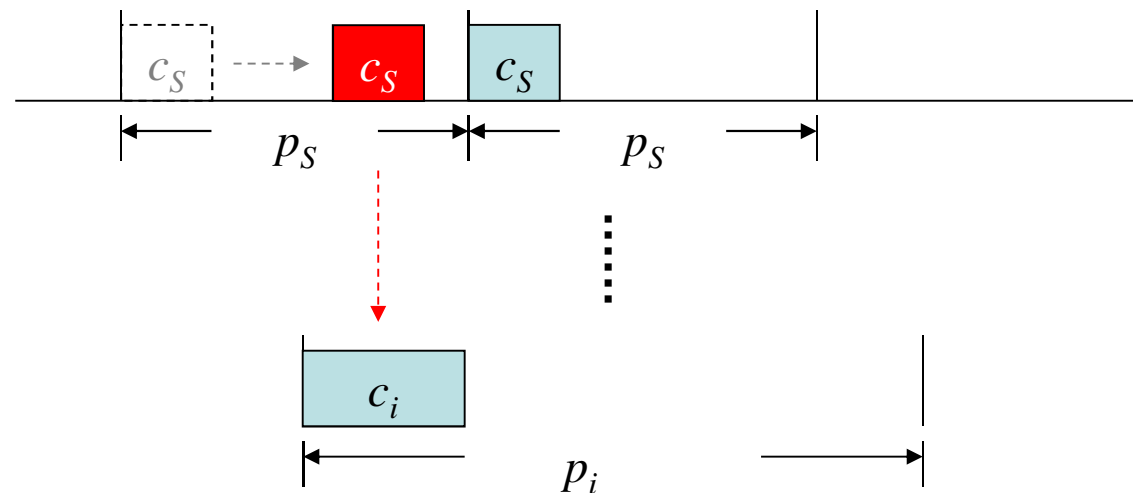


# Deferrable Servers

- Yet another good news:
  - The Liu-and-Layland utilization bound is still useful here, if we take the interference from the server into consideration

# Deferrable Server

- A high-priority deferrable server  $T_{DS}$  may interfere a low-priority task  $T_i$  by **up to**  $C_S$  units of time
  - The red  $c_s$  is not supposed to be there, if  $T_{ds}$  is a periodic task...



# Deferrable Server

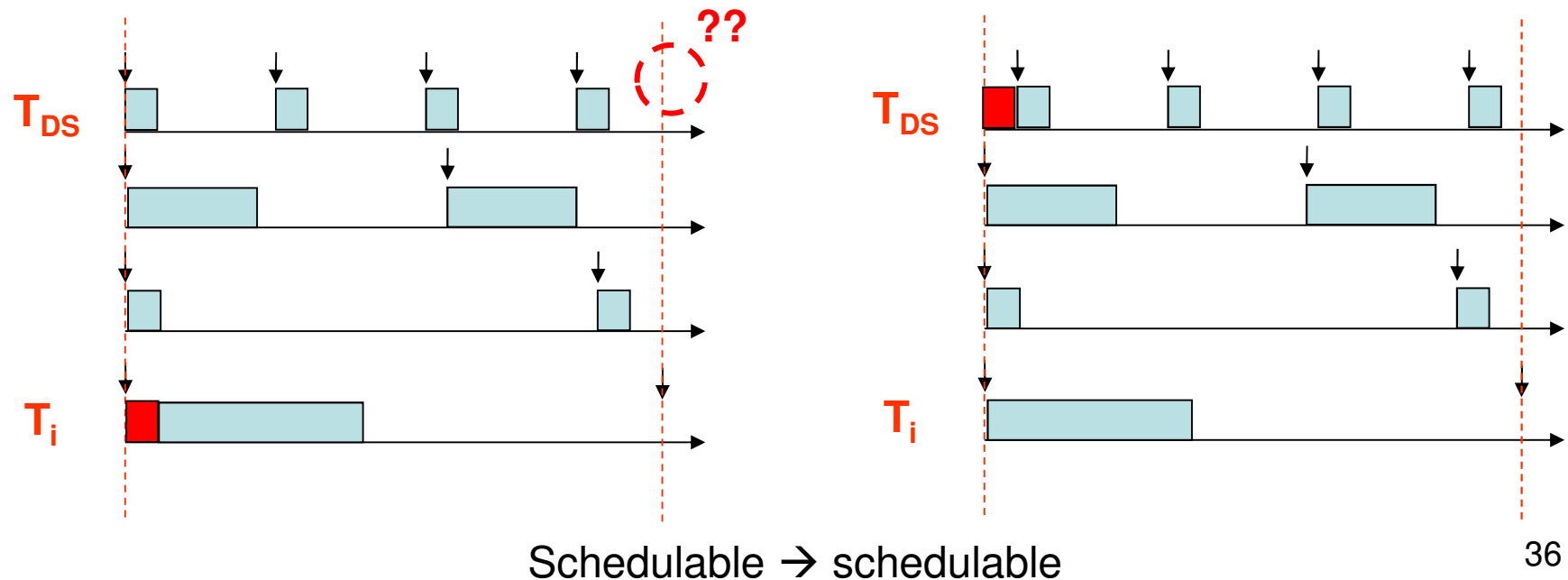
- Task set  $\{T_1, \dots, T_m, T_{DS}, T_{m+2}, \dots, T_n\}$ 
  - $\{T_1, \dots, T_m, T_{DS}\}$  is schedulable if  $\sum c_i/p_i \leq U(m)$
  - For  $i=(m+2) \dots n$ , Task  $T_i$  is schedulable if

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

- 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 treat the additional  $c_s$  as “extra computation time” of  $T_i$ ?



# Deferrable Servers

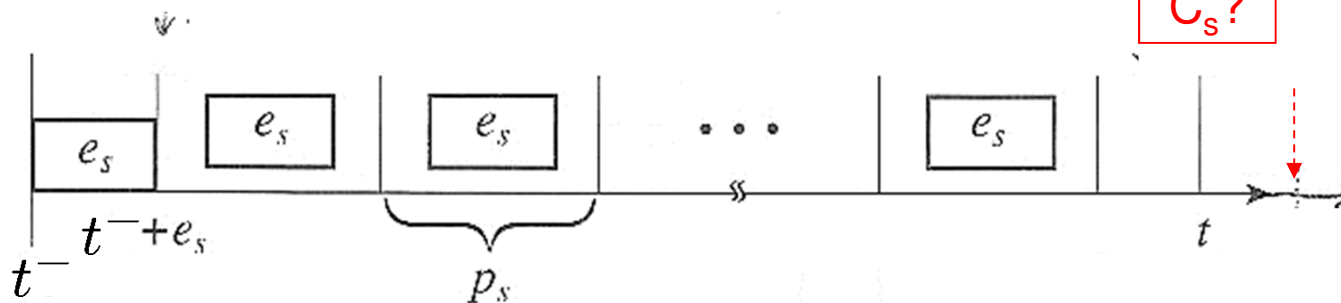
- Schedulability of a deadline-driven system with a deferrable server
  - Consider task set  $\{T_1, \dots, T_n\} \cup \{(c_s, p_s)\}$ 
    - Let  $t$  be the deadline of a job of a periodic task
    - Let  $t^- (<t)$  be the latest time instant when the CPU is idle or is executing jobs with deadline later than  $t$ 
      - In  $(t^-, t]$ , only jobs whose deadline  $<t$  are executed
    - We are interested in the maximum computation time in  $(t^-, t]$  demanded by periodic tasks and the deferrable server

# Deferrable Servers

- Deadline-driven systems
  - The maximum CPU time demanded by the deferrable server in time interval  $(t^-, t]$  is

$$c_s + \left\lfloor \frac{t - t^- - c_s}{p_s} \right\rfloor c_s$$

The last job cannot affect jobs in  $(t^-, t]$  because its deadline is later than  $t$



The maximum occurs when the term inside the floor function is an integer

# Deferrable Servers

- Deadline-driven systems
  - The maximum CPU time demanded by periodic jobs is

$$\sum_{k=1}^n \left\lfloor \frac{t - t^-}{p_k} \right\rfloor c_k$$

Similarly, the maximum happens when the term inside the floor function is a integer

# Deferrable Servers

- If the taskset is unschedulable, then the total CPU time demanded by periodic jobs and aperiodic jobs exceeds  $t - t^-$

$$t - t^- < c_s + \left( \frac{t - t^- - c_s}{p_s} \right) c_s + \sum_{k=1}^n \frac{c_k}{p_k} (t - t^-)$$

- Dividing both sides of the inequality and then Inverting the above statement, we have the theorem:



# Deferrable Servers

- A deadline-driven system with a deferrable server is schedulable **if**


$$\sum_{k=1}^n \frac{c_k}{p_k} + u_s \left( 1 + \frac{p_s - c_s}{p_s} \right) \leq 1$$

[Ghazalie95]

# Deferrable Servers

- RM
  - RTA
  - [Lehoczky87]
  - Revised U test
- EDF
  - [Ghazalie95]
- Deferrable servers has been adopted in Ada 2005 for real-time and concurrent programming

# Bandwidth-Preserving Servers

- Deferrable servers
- Sporadic servers (RM only) 
- Priority-exchange servers
- Constant utilization servers and total bandwidth server (EDF only)

# Sporadic Servers

- Deferrable servers have simple rules for budget consumption and replenishment, but they impose extra delays on periodic tasks
  - Double hits in priority-driven and deadline-driven scheduling
- Schedulability tests become quite complicated

# Sporadic Servers

- Sporadic servers are designed to improve upon deferrable servers
  - Schedulability test for periodic systems with sporadic servers is very simple
  - However, sporadic servers have complicated consumption and replenishment rules

# Sporadic Servers

- Design guideline: a sporadic server never demand processor time more than a periodic task  $(c,p)$  **in any time interval**
  - Sporadic servers can be characterized by the corresponding periodic tasks in terms of schedulability

# Sporadic Servers

- A sporadic server can be characterized by  $(c,p)$ 
  - Again, notice that a sporadic server is not a periodic task!
- Different designs of sporadic servers exist, and they aim at
  - preserving server budget as long as possible, and
  - replenishing server budget as early as possible

# Sporadic Servers

- Simple sporadic servers in fixed-priority systems

- $t_r$  denotes the latest (actual) replenishment time.
- $t_f$  denotes the first instant after  $t_r$  at which the server begins to execute.
- $t_e$  denotes the latest *effective replenishment time*.
- At any time  $t$ , *BEGIN* is the beginning instant of the earliest busy interval among the latest contiguous sequence of busy intervals of the higher-priority subsystem  $\mathbf{T}_H$  that started before  $t$ . (Two busy intervals are contiguous if the later one begins immediately after the earlier one ends.)
- *END* is the end of the latest busy interval in the above defined sequence if this interval ends before  $t$  and equal to infinity if the interval ends after  $t$ .



# Sporadic Servers

- Consumption rules

**C1** The server is executing.

**C2** The server has executed since  $t_r$  and  $END < t$

- *Replenishment Rules of Simple Fixed-Priority Sporadic Server:*

**R1** Initially when the system begins execution and each time when the budget is replenished, the execution budget =  $e_s$ , and  $t_r$  = the current time.

**R2** At time  $t_f$ , if  $END = t_f$ ,  $t_e = \max(t_r, BEGIN)$ . If  $END < t_f$ ,  $t_e = t_f$ . The next replenishment time is set at  $t_e + p_s$ .

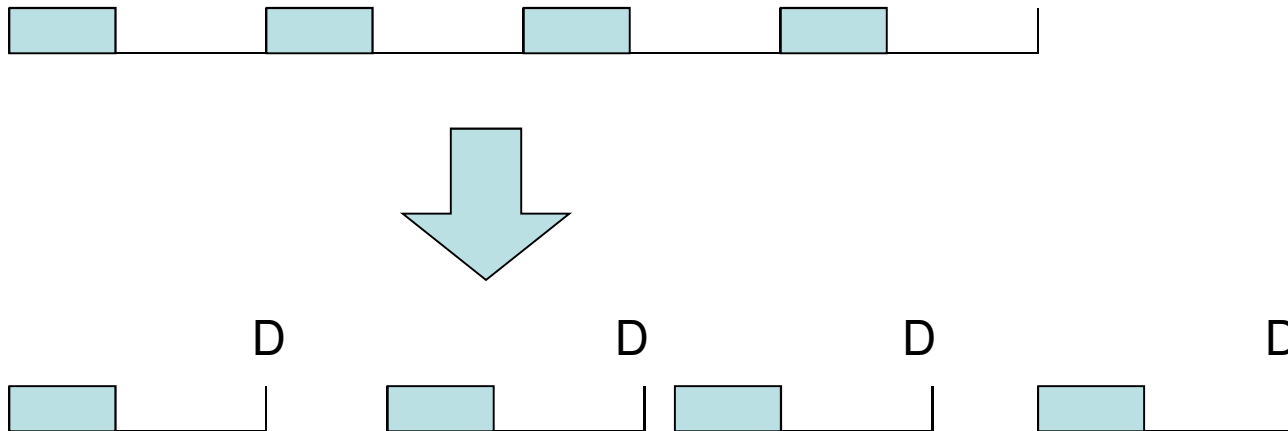
**R3** The next replenishment occurs at the next replenishment time, except under the following conditions. Under these conditions, replenishment is done at times stated below.

(a) If the next replenishment time  $t_e + p_s$  is earlier than  $t_f$ , the budget is replenished as soon as it is exhausted.

(b) If the system **T** becomes idle before the next replenishment time  $t_e + p_s$  and becomes busy again at  $t_b$ , the budget is replenished at  $\min(t_e + p_s, t_b)$ .

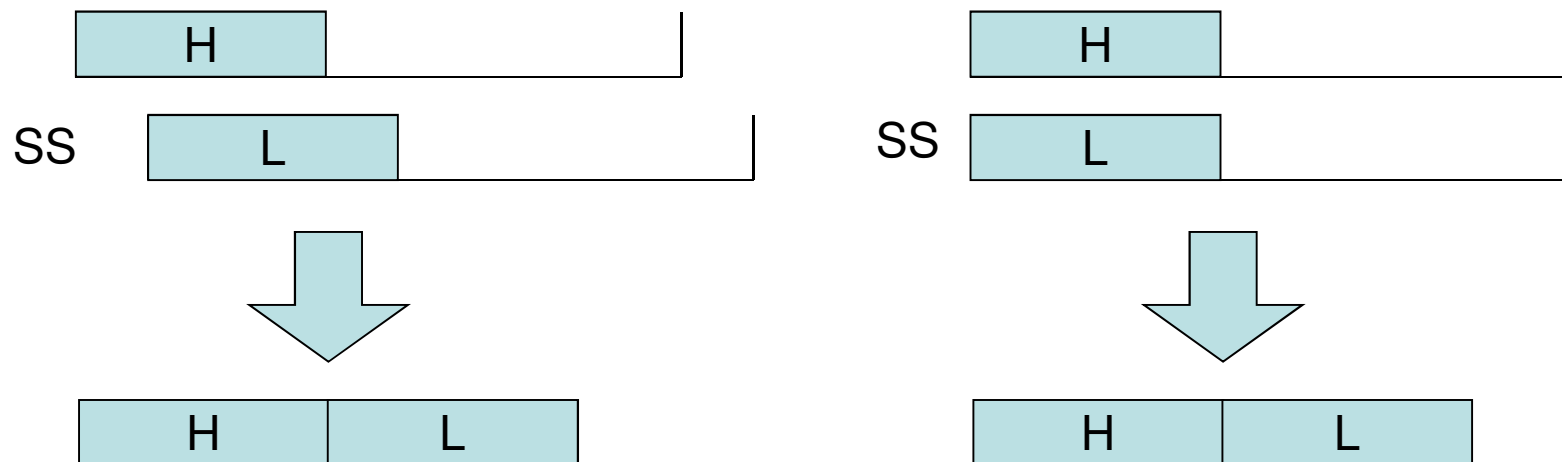
# Simple Sporadic Servers

- **Fact A:** if a periodic task is schedulable in a fixed-priority system, then all its jobs are schedulable if their inter-arrival times are not shorter than its task period
  - Help preserve the budget as late as possible



# Simple Sporadic Servers

- **Fact B:** when a low-priority job is **delayed** by a high-priority job, the produced schedule is the same as that produced by the two jobs arriving at the same time
  - Help replenish the budget as early as possible



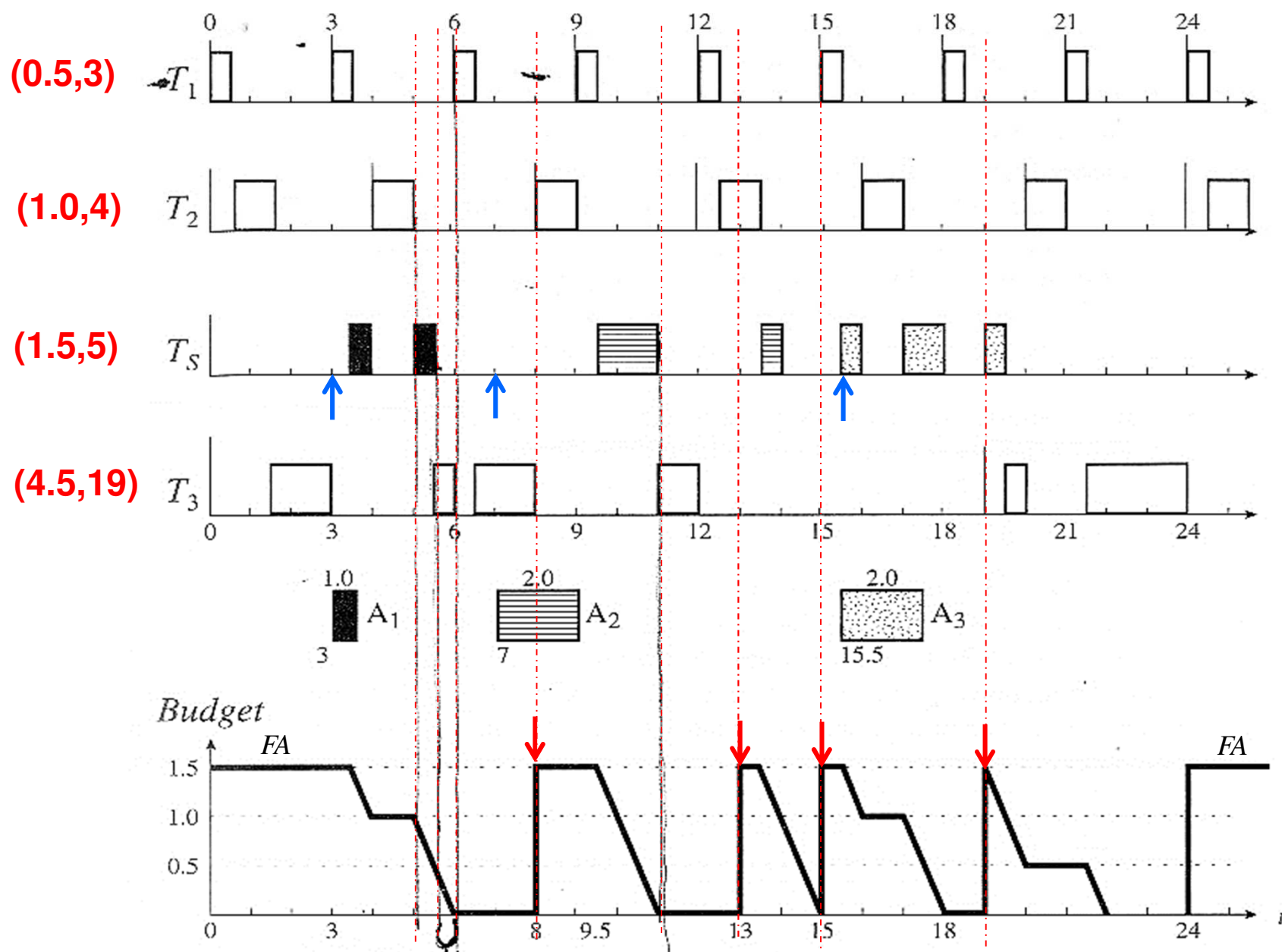


FIGURE 7-8 Example illustrating the operations of a simple sporadic server:  $T_1 = (3, 0.5)$ ,  $T_2 = (4, 1.0)$ ,  $T_3 = (19, 4.5)$ ,  $T_s = (5, 1.5)$ .

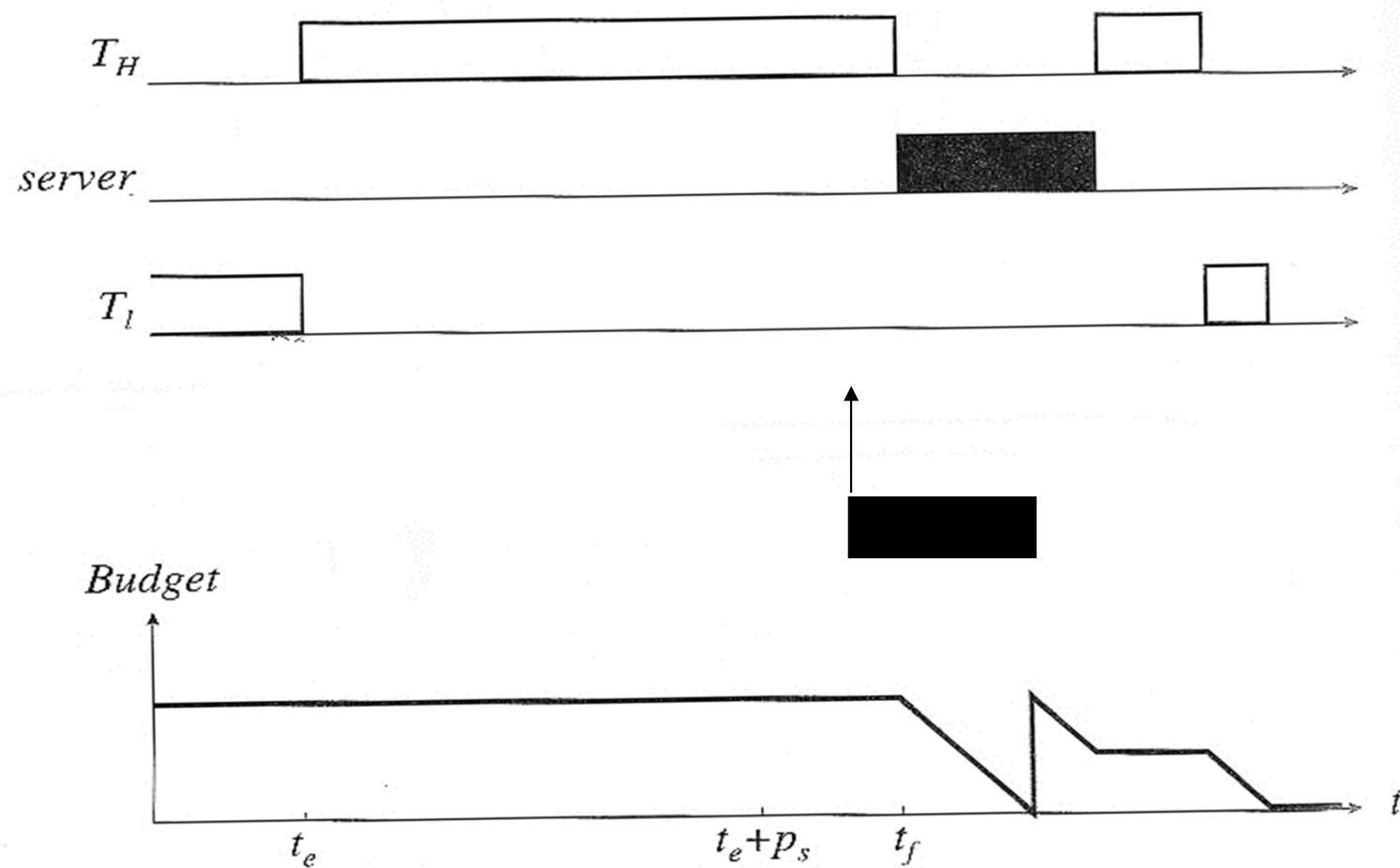
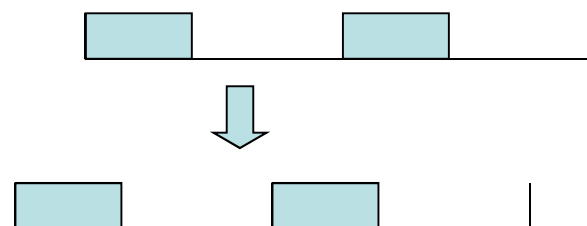


FIGURE 7-9 A situation where rule R3a applies.



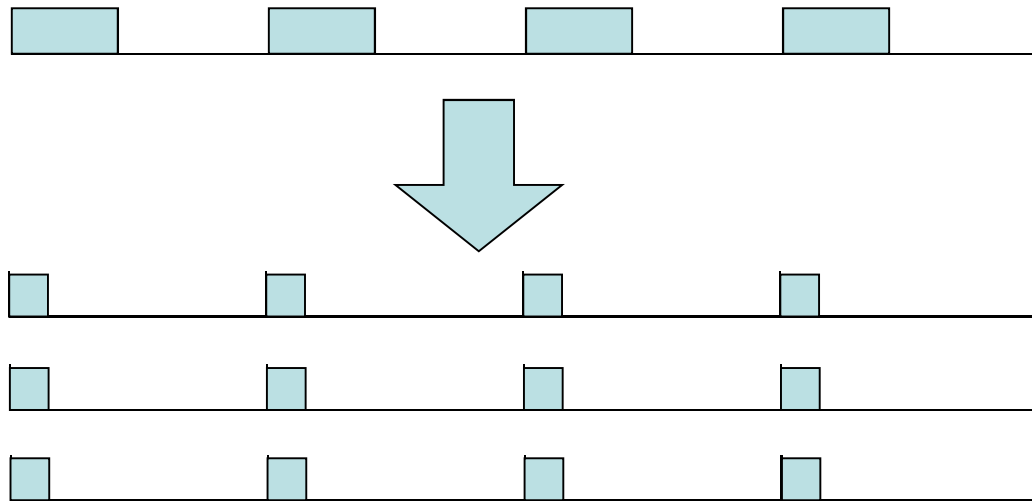
# Sporadic Servers

- An idle simple sporadic server consumes its budget as all high-priority tasks are idle
  - Actually no aperiodic jobs are serviced, the consumed budget is wasted!
  - For further improvement, a sporadic server could be split into chunks [Sprunt 1989]
    - One “chunk” emulates a smaller sporadic server with the same “period”

# Sporadic Servers

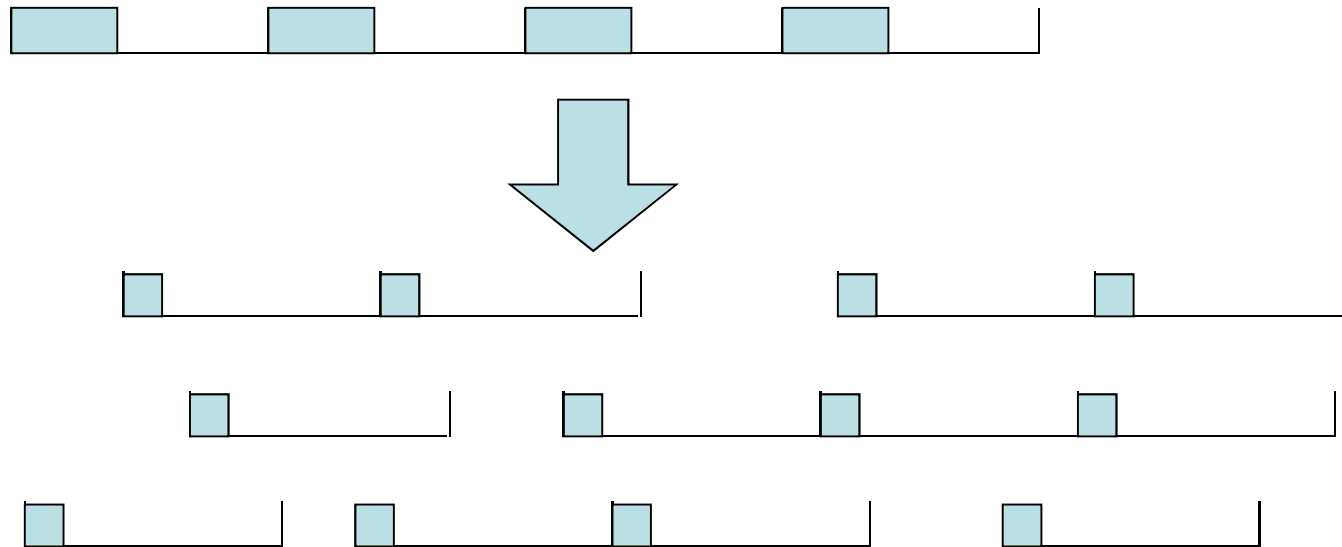
- **Fact C:** Let  $T$  be a periodic system. If  $\{(c, p)\} \cup T$  is schedulable then  $\{(c_1, p), (c_2, p), \dots, (c_n, p)\} \cup T$  is also schedulable, provided that

$$\sum_{i=1}^n c_i = c$$



# Sporadic Servers

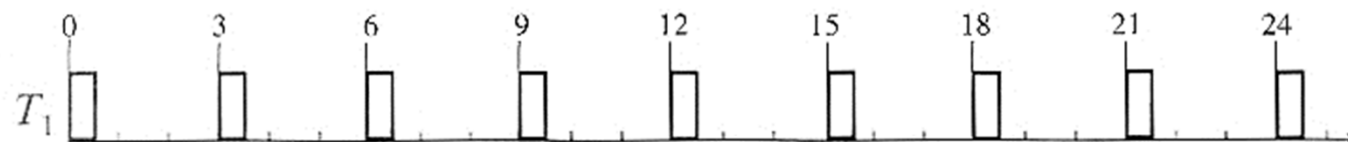
- Combining Fact A and Fact C



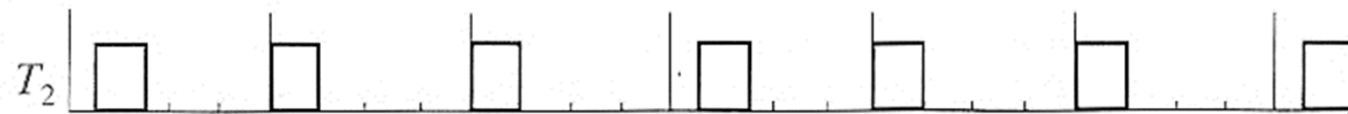
Chunks can be split and merged whenever needed



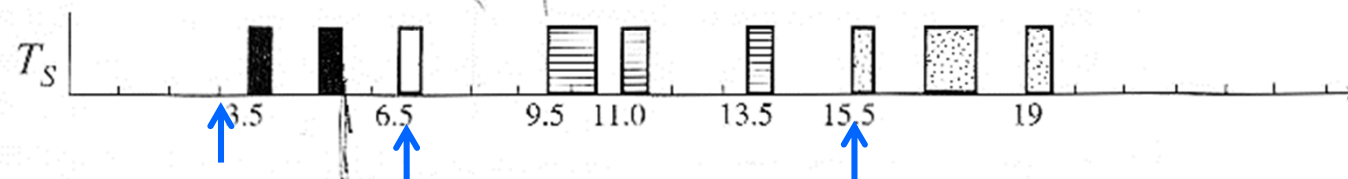
(0.5,3)



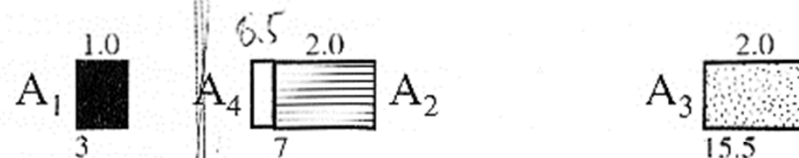
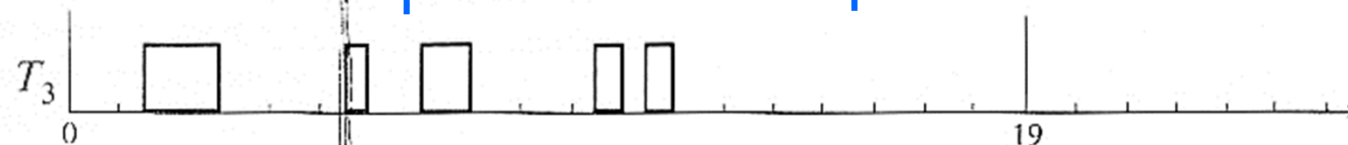
(1.0,4)



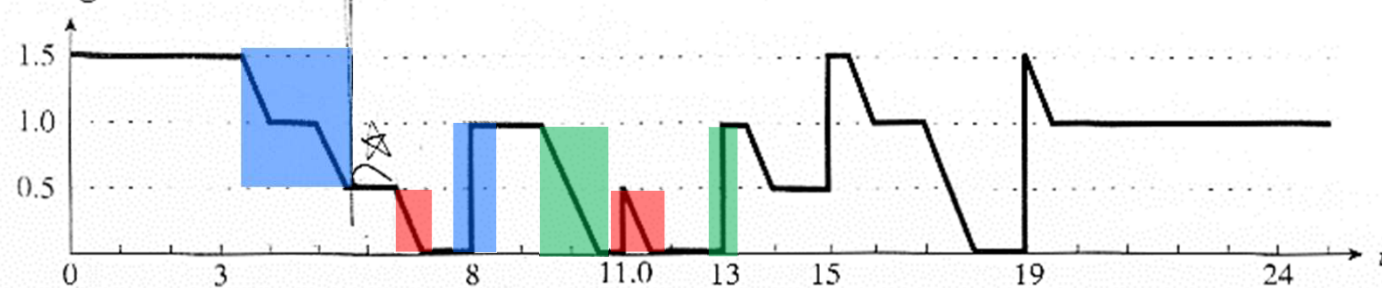
(1.5,5)



(4.5,19)



Budget



[Sprunt]

FIGURE 7-11 Example illustrating SpSL sporadic servers:  $T_1 = (3, 0.5)$ ,  $T_2 = (4, 1.0)$ ,  $T_3 = (19, 4.5)$ ,  $T_S = (5, 1.5)$

# Sporadic Servers

- Response times in simple Sporadic Servers
  - A1:5.5, A2: 14, A3: 19.5
- Response times in Sprunt Sporadic Servers
  - A1: 5.5, A2: 14, A3: 19.5, **A4: 7**

# Sporadic Servers

- Note that previous slides about sporadic servers are for fixed-priority systems
- In deadline-driven systems, both the concepts of simple sporadic servers [Liu] and “chunk” sporadic servers [Ghazalie] are still applicable
  - They are very complicated though
  - However, we have better, simpler solutions for deadline-driven scheduling!

# Sporadic Servers

- Sporadic servers have been adopted by real-time POSIX standard
  - RTLinux
  - RTAI


# POSIX specification on SS

- How to use
  - Call `sched_setscheduler(pid, SCHED_SPORADIC, ...)`
  - Applicable to fixed-priority systems only
  - Also set an foreground priority `sched_priority` and a background priority
- Operation
  - The server runs at foreground priority if there is budget but switches to background priority otherwise.
  - Budget is replenished by the scheduler on periods

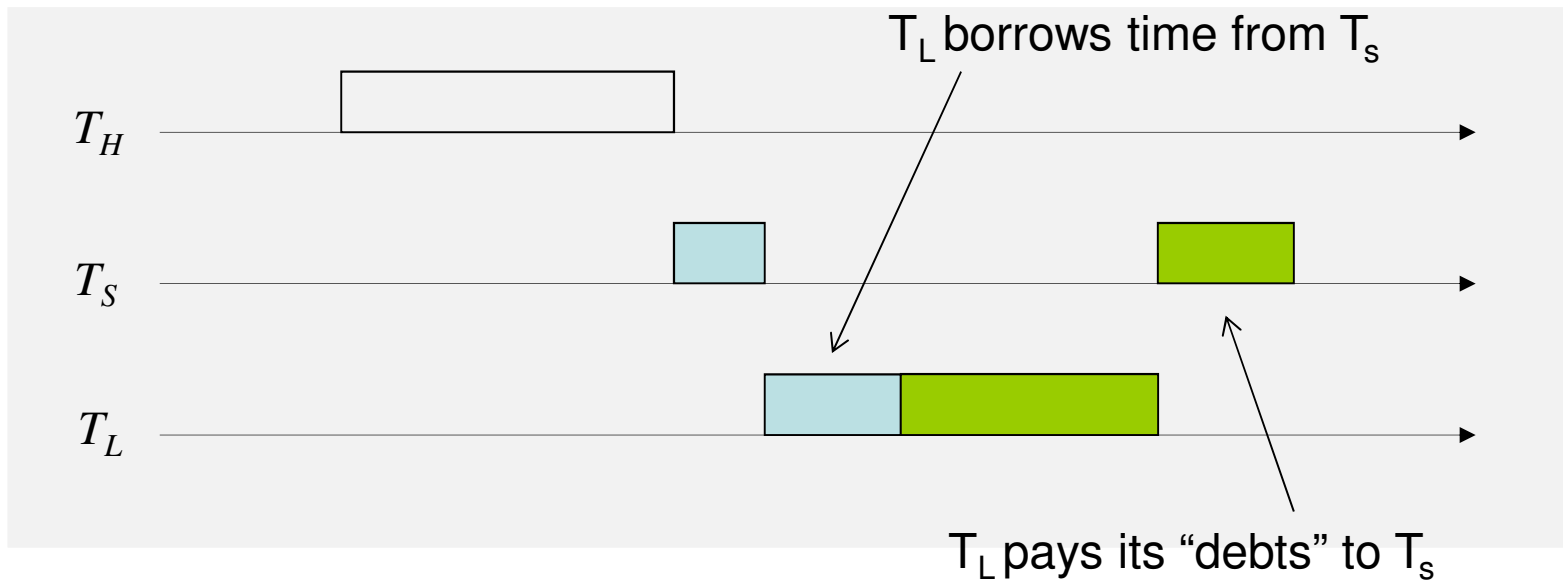
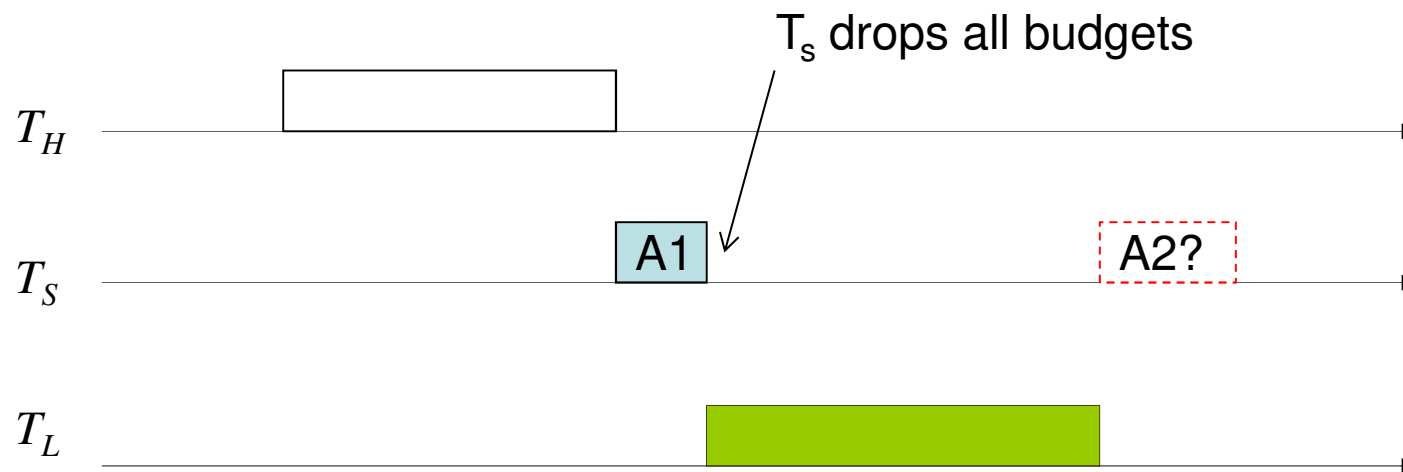
# Sporadic Servers

- Discussion
  - DS: simple cons/repl rules, complicated schedulability test
  - SS: simple schedulability test, complicated cons/repl rules
  - If you are a system engineer, which one will you take?

# Bandwidth-Preserving Servers

- Deferrable servers
- Sporadic servers
- Priority-exchange servers 
- Constant utilization servers and total bandwidth server

# Priority-Exchange Servers






# Priority-Exchange Servers

- Arithmetically simple, but extremely hard to implement
  - May require hardware acceleration
- We shall then see the elegance and beauty of bandwidth-preserving servers for deadline-driven systems

# Bandwidth-Preserving Servers

- Deferrable servers
- Sporadic servers (RM only)
- Priority-exchange servers
- Constant utilization servers and total bandwidth server (EDF only) 

# Constant Utilization Servers

- Let a sporadic job is characterized by arrival time  $a$ , execution time  $c$ , deadline  $d$ 
  - Define its density as  $c/(d - a)$
- *Theorem*: A system of independent periodic jobs is schedulable by EDF if the total density of tasks is no larger than 1 at any time instant

# Constant Utilization Servers

- A sporadic task is of a stream of sporadic jobs



- Given any time point  $t$  that falls between the  $a_i$  and  $d_i$  of a sporadic job  $J_i$ ,
  - $c_i/(d_i - a_i)$  is referred to as the **instantaneous utilization (density)** contributed by job  $J_i$  at time instant  $t$

# Constant Utilization Servers

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

# Constant Utilization Servers

- Consumption rules is simple: A server consumes its budget when it is serving jobs
- Replenishment rules:

*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

- As suggested by the name, the instantaneous utilization of every sporadic job equals to the server size
- CUS replenishes its budget at job deadlines



- **Pitfall:** A sporadic job may have two “deadlines”:
  - The deadline comes with the job:  $d_a$
  - The deadline assigned by CUS algorithm:  $d_b$
  - Must satisfy  $d_b \leq d_a$

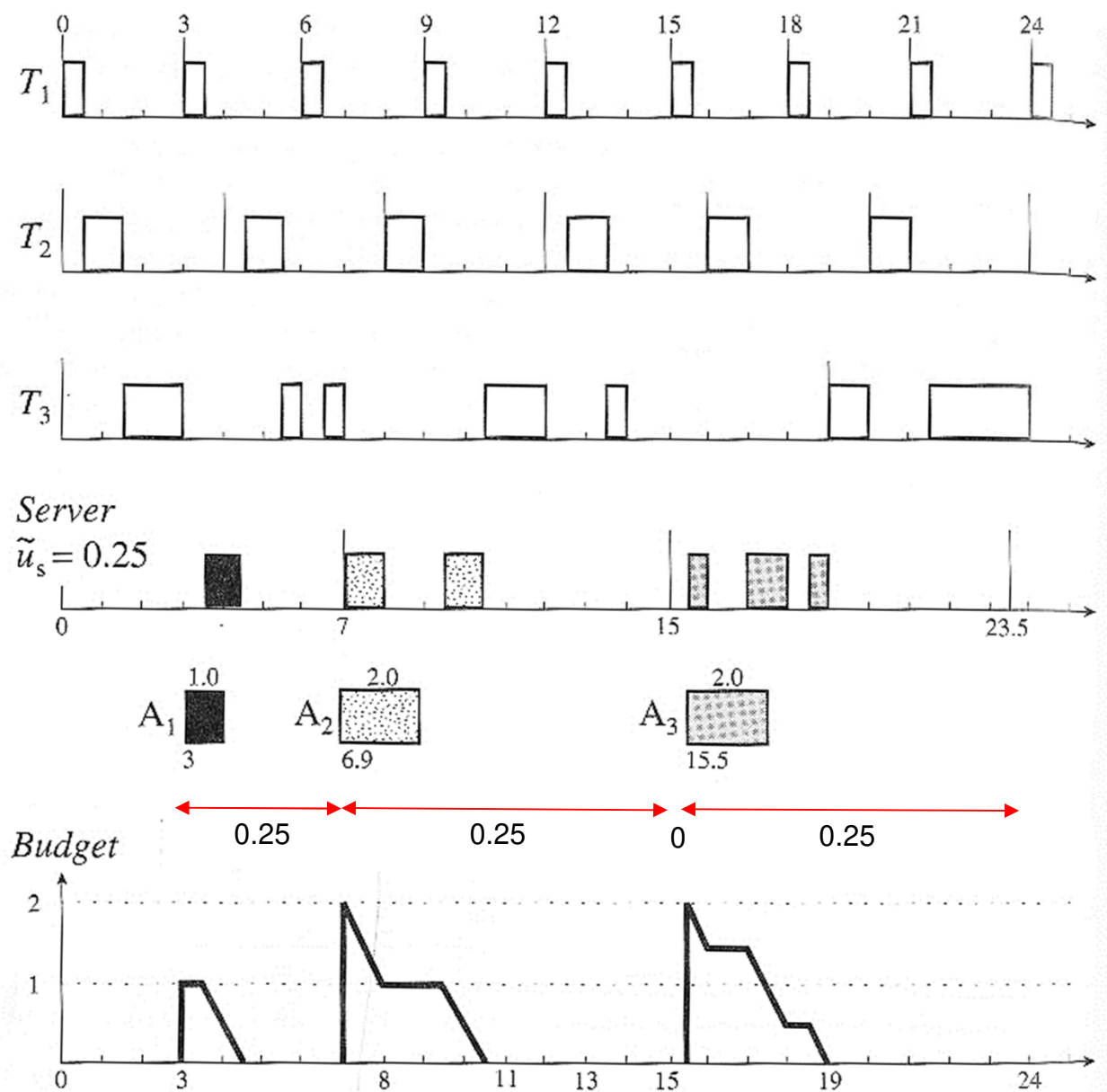


FIGURE 7-13 Example illustrating the operations of constant utilization server:  $T_1 = (4, 0.5)$ ,  $T_2 = (4, 1.0)$ ,  $T_3 = (19, 4.5)$ .



# Total Bandwidth Servers

- A CUS replenishes its budget **not earlier** than job deadlines
  - A newly arriving job has to wait until the deadline of the latest aperiodic job, even if the server is currently idle
    - A2 in the last example
- Is early replenishment possible?
  - Yes: total bandwidth servers

# Total Bandwidth Servers

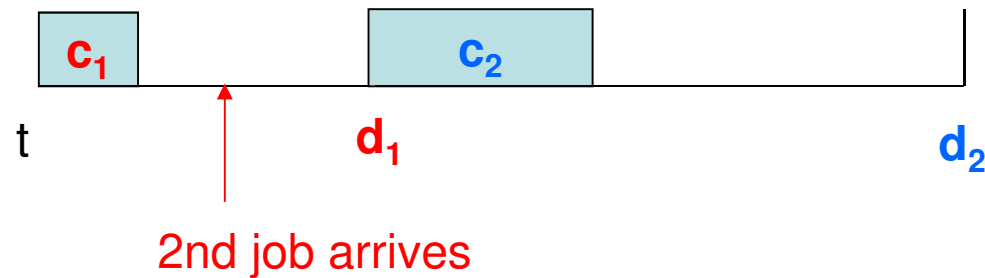
- Consumption rules: the same as those of CUS
- Replenishment rules:

*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!

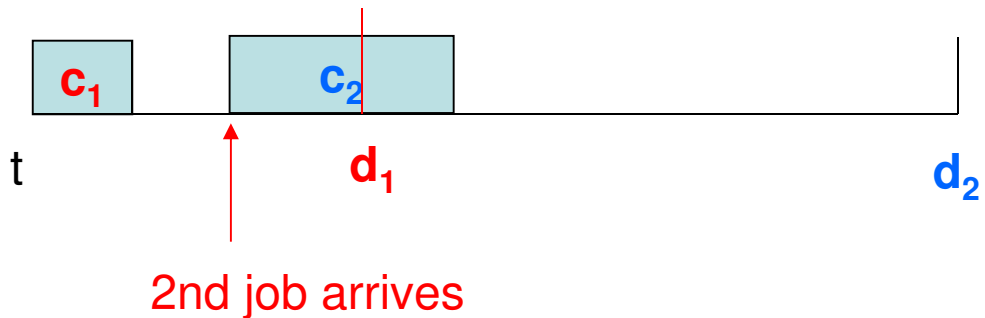
# Total Bandwidth Servers



CUS:

$$\frac{c_1}{d_1 - t} = \frac{c_2}{d_2 - d_1} = u_s$$

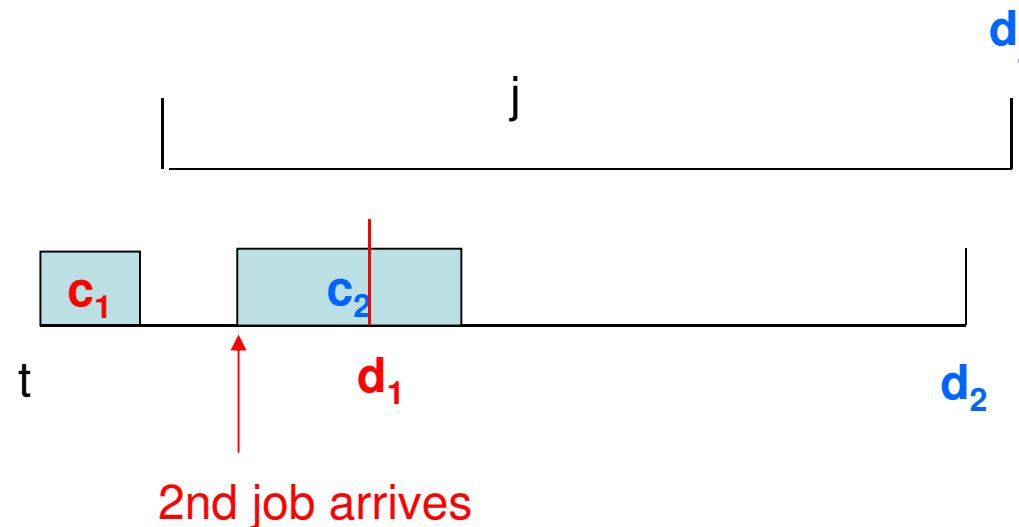
TBS:



Budget is replenished on the arrival of the 2nd job, and the job is assigned to a deadline  $d_2 = d_1 + c_2/u_s$

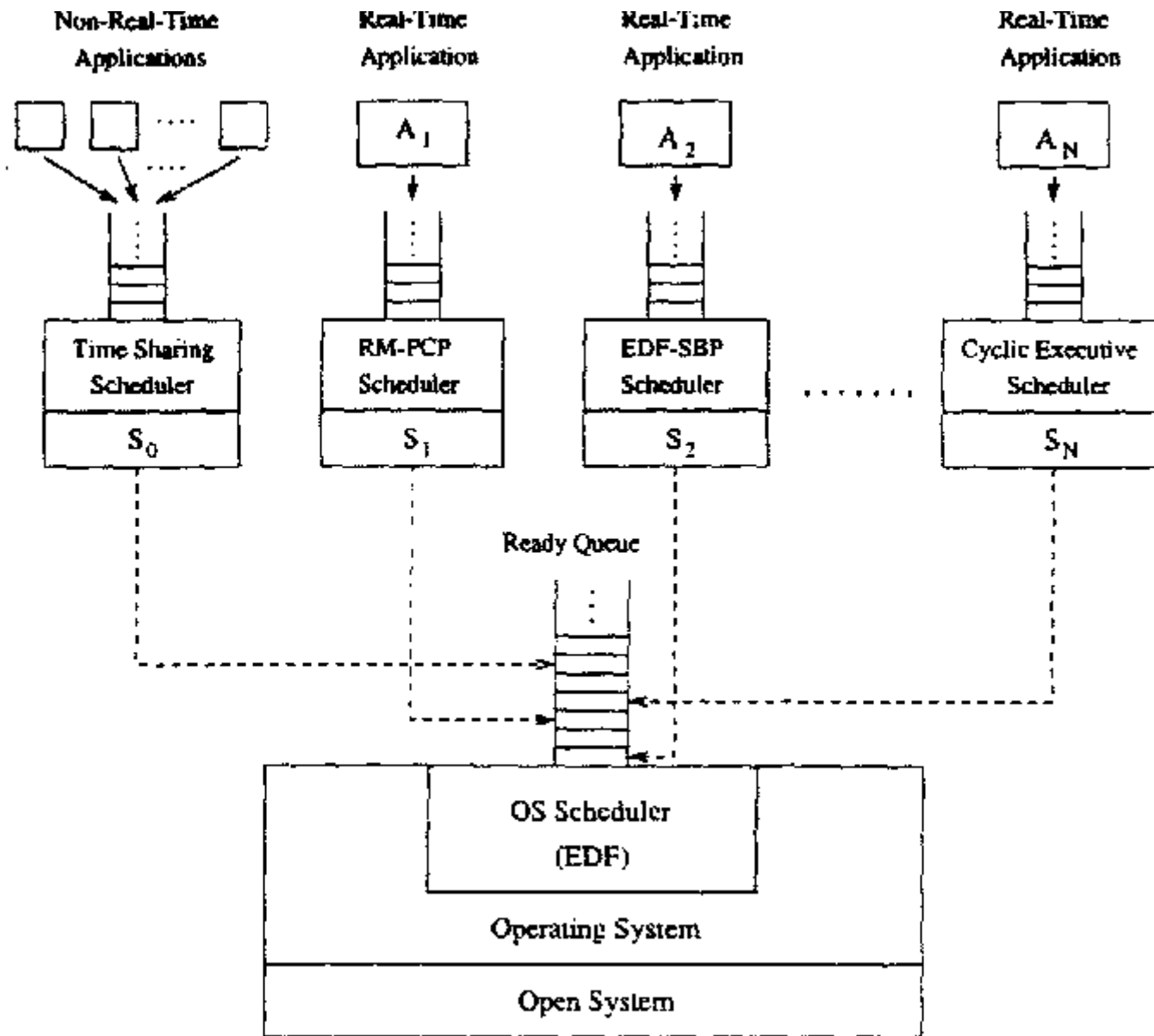
# Total Bandwidth Servers

- Correctness of TBS:
  - Jobs affected by the early replenishment
    - Only those  $d_j > d_2$
    - What about those  $d_j < d_2$  or  $d_j < d_1$ ?



# Total Bandwidth Servers

- CUS/TBS are two good approaches for CPU bandwidth reservation
  - Service-level agreement, SLA
  - Quality of service, QoS
- Aperiodic jobs serviced by different servers would have an illusion that they are serviced by different CPUs with reduced power
  - This concept extends to **open system**, which can be applied to virtual machine scheduling



# Part B: Checking Deadlines for Sporadic Jobs

# Scheduling Sporadic Jobs

- So far we discussed many techniques to serve aperiodic jobs in periodic systems
- While serving aperiodic jobs, a server has to check the deadlines of sporadic jobs
  - A server rejects an incoming sporadic job if it finds the sporadic deadline unsatisfiable



# Scheduling Sporadic Jobs: Deadline-driven systems (EDF)

- CUS/TBS
  - Compare
    - (d1) the deadline of the sporadic job
    - (d2) the deadline assigned by server algorithm
  - If  $d1 \leq d2$ , then accept

# Scheduling Sporadic Jobs: Priority-Driven Systems (RM)

- The concept of sporadic server:
  - “the server  $(c_s, p_s)$  receives at most  $c_s$  units of time every  $p_s$  units of time”
- The first sporadic job  $S_1(r, c_{s,1}, d_{s,1})$  is accepted if

$$\rho_{s,1}(t) = \left\lfloor \frac{(d_{s,1} - r)}{p_s} \right\rfloor c_s - c_{s,1} \geq 0$$

# Scheduling Sporadic Jobs: Priority-Driven Systems (RM)

- Acceptance test of job  $S_i(r, c_{s,i}, d_{s,i})$

$$\rho_{s,i}(t) =$$

$$\left\lfloor \frac{(d_{s,i} - r)}{p_s} \right\rfloor c_s - c_{s,i} - \sum_{d_{s,k} < d_{s,i}} (c_{s,k} - \epsilon_{s,k}) \geq 0$$

- $c_{s,k} - \epsilon_{s,k}$  stands for the remaining execution time of sporadic job  $S_k$
- If succeed, proceed to the next step

# Scheduling Sporadic Jobs: Priority-Driven Systems (RM)

- Check if  $\rho_{s,k} \geq 0$  for  $S_k$ , whose deadline is after  $d_{s,i}$  and may be adversely affected by the acceptance of  $S_i$
- If all tests succeed,  $S_i$  is accepted

# Summary

- Serving aperiodic/sporadic jobs in periodic system
  - Priority-driven systems (DS, SS)
  - Deadline-driven systems (DS, TBS, CUS)
- Server designs are based on periodic task emulation
  - Can budget be retained as late as possible?
  - Can budget be replenished as early as possible?
- Aperiodic jobs are served in best-effort fashion
- Testing sporadic deadlines before admitting sporadic jobs