University of Glasgow | School of Computing Science

Priority-driven Scheduling of Aperiodic and Sporadic Tasks (2)

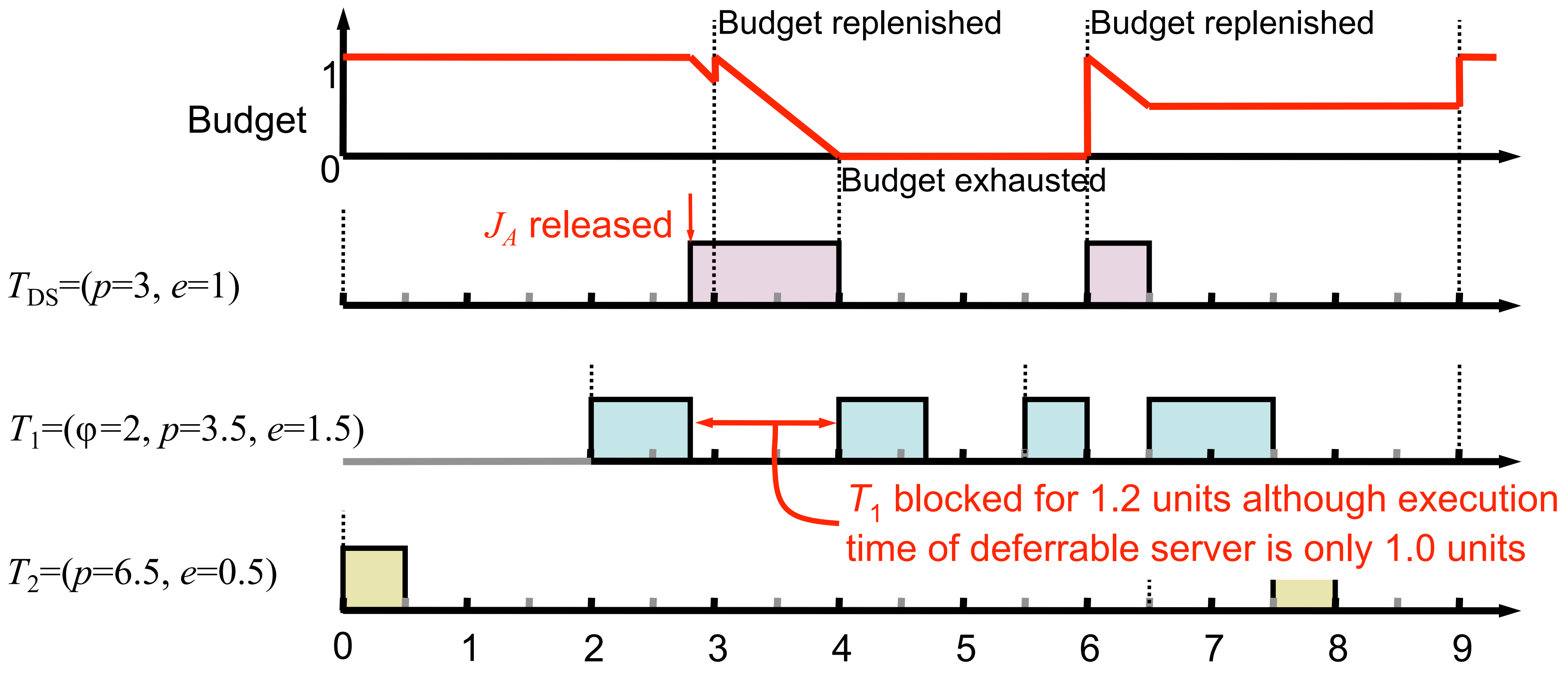
Advanced Operating Systems (M)
Lecture 7

Lecture Outline

- Scheduling aperiodic jobs (cont'd)
    - Sporadic servers
    - Constant utilisation servers
    - Total bandwidth servers
    - Weighted fair queuing servers
- Scheduling sporadic jobs

Limitations of Deferrable Servers

- Limitation of deferrable servers – they may delay lower-priority tasks for more time than a periodic task with the same period and execution time:



Sporadic Servers

- A sporadic server is designed to eliminate this limitation
  - A different type of bandwidth preserving server: several different sub-types
  - More complex consumption and replenishment rules ensure that a sporadic server with period p_S and budget e_S never demands more processor time than a periodic task with the same parameters

Simple Fixed-Priority Sporadic Server

- System, T, of independent preemptable periodic tasks and a sporadic server with parameters (p_s, e_s)
  - Fixed-priority scheduling; system can be scheduled if sporadic server behaves as a periodic task with parameters (p_s, e_s)
- Define:
  - T_H : the periodic tasks with higher priority than the server (may be empty)
  - t_r : the last time the server budget replenished
  - t_f : the first instant after t_r at which the server begins to execute
  - At any time t define:
    - $BEGIN$ as the start of the earliest busy interval in the most recent contiguous sequence of busy intervals of T_H starting before t (busy intervals are contiguous if the later one starts immediately the earlier one ends)
    - END as the end of the latest busy interval in this sequence if this interval ends before t; define $END = \infty$ if the interval ends after t

Simple Fixed-Priority Sporadic Server

- Consumption rule:
  - At any time $t \geq t_r$, if the server has budget and if either of the following two conditions is true, the budget is consumed at the rate of 1 per unit time:
    - C1: The server is executing
    - C2: The server has executed since t_r and $END < t$
  - When they are not true, the server holds its budget

- That is:
  - The server executes for no more time than it has execution budget
  - The server retains its budget if:
    - A higher-priority job is executing, or
    - It has not executed since t_r
  - Otherwise, the budget decreases when the server executes, or if it idles while it has budget

Simple Fixed-Priority Sporadic Server

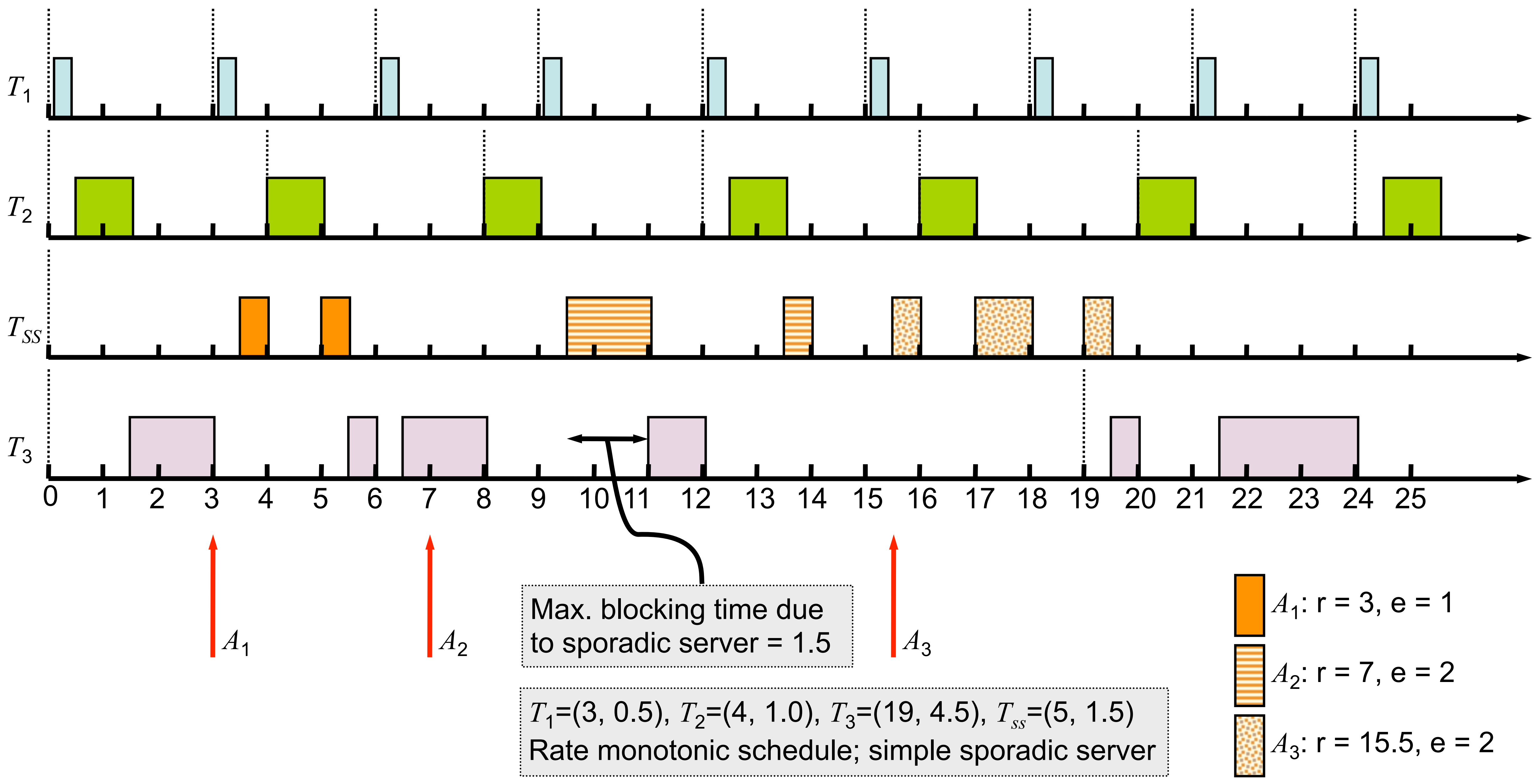
- Replenishment rules
  - R1: When system begins executing, and each time budget is replenished, set the budget to e_S and t_r = the current time.
  - R2: When server begins to execute (defined as time t_f)

```
if $END = t_f$ then
    $t_e = \max(t_r, BEGIN)$
else if $END < t_f$ then
    $t_e = t_f$
The next replenishment time is set to $t_e + p_S$
```

t_e = effective replenishment time

  - R3: budget replenished at the next replenishment time, unless:
    - If $t_e + p_S$ is earlier than t_f the budget is replenished as soon as it is exhausted
    - If T becomes idle before $t_e + p_S$, and becomes busy again at t_b, the budget is replenished at $\min(t_b, t_e + p_S)$

Example: Fixed-Priority Sporadic Server

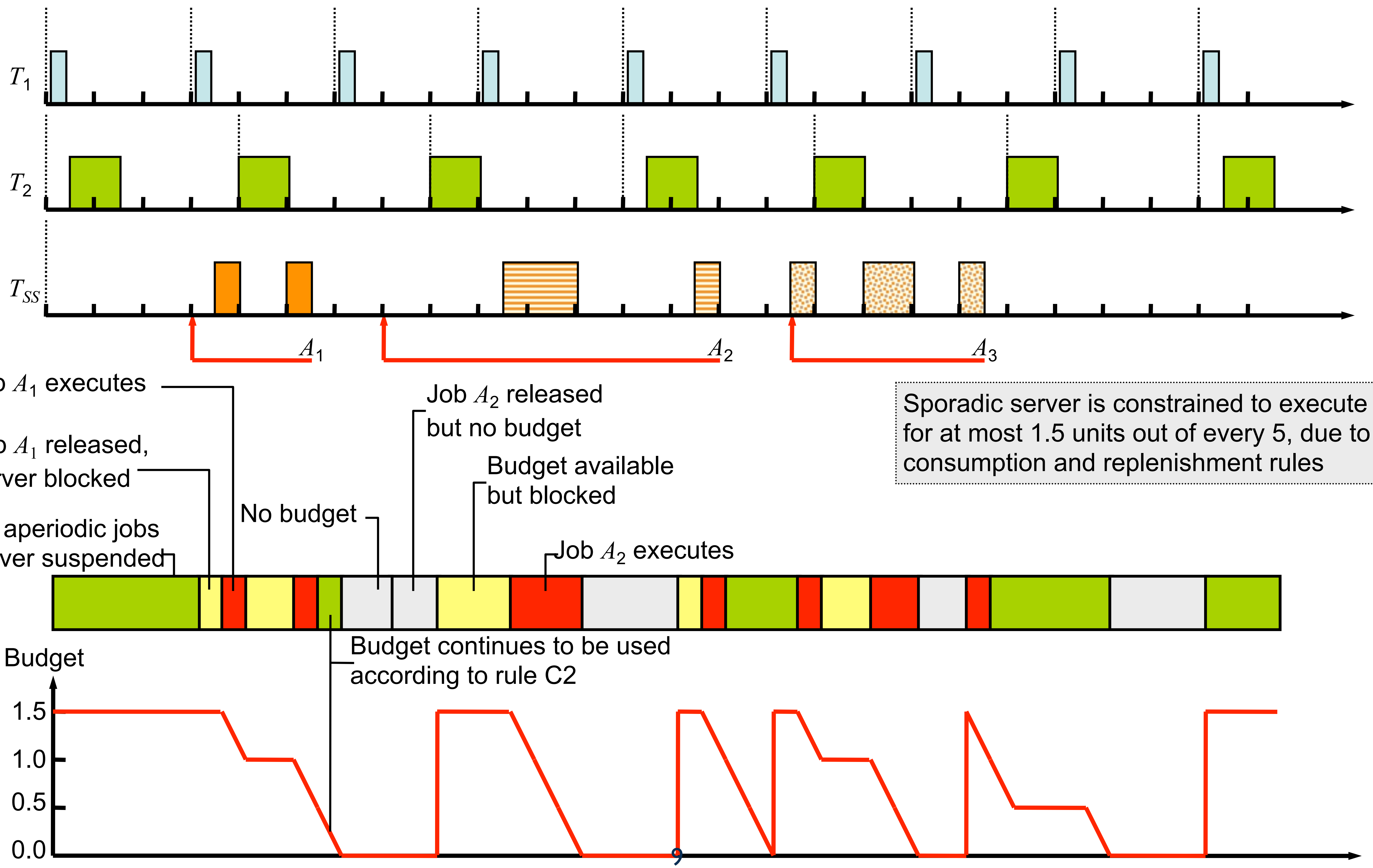


Max. blocking time due to sporadic server = 1.5

T_1=(3, 0.5), T_2=(4, 1.0), T_3=(19, 4.5), T_{ss}=(5, 1.5)
Rate monotonic schedule; simple sporadic server

- A_1: r = 3, e = 1
- A_2: r = 7, e = 2
- A_3: r = 15.5, e = 2

Example: Fixed-Priority Sporadic Server



T_1
T_2
T_{SS}
A_1
A_2
A_3
Job A_1 executes
Job A_1 released, server blocked
No aperiodic jobs server suspended
No budget
Job A_2 released but no budget
Budget available but blocked
Job A_2 executes
Budget continues to be used according to rule C2
Budget
1.5
1.0
0.5
0.0

Sporadic server is constrained to execute for at most 1.5 units out of every 5, due to consumption and replenishment rules

Simple Fixed-Priority Sporadic Server

- More complex than a polling server or a deferrable server, but much easier to prove the system can be scheduled

- Theorem: for the purpose of validating a schedule, you can treat a simple sporadic server (p_s, e_s) in a fixed-priority system exactly the same as any other periodic task T_i with $p_i = p_s$ and $e_i = e_s$
  - The actual inter-release times of the sporadic server will sometimes be greater than p_s, and their execution times less than e_s, but this does not affect correctness

Simple Dynamic-Priority Sporadic Server

- It is possible to define a simple sporadic server to operate in a dynamic-priority environment
  - E.g., when using EDF or LST scheduling
- Consumption and replenishment rules conceptually similar to those for a fixed-priority scheduler, with minor modifications that account for the difference in scheduling algorithm
  - Provides same scheduling guarantees as the simple sporadic server for fixed-priority schedulers
  - A simple sporadic server (p_s, e_s) in an EDF or LST system can be treated exactly the same as any other task T_i with $p_i = p_s$ and $e_i = e_s$ when testing whether the system can be scheduled

Other Bandwidth Preserving Servers

- Two other bandwidth preserving server algorithms often used for process scheduling:
    - Constant utilisation server
    - Total bandwidth server

- Used in virtual machine systems, to assign known fraction of processor time to some task
    - Aiming to provide fair sharing, timing isolation, or guaranteed throughput

Constant Utilisation Server

- Constant utilisation server reserves a fraction, $\tilde{u}_s$, of the processor time for execution of the server
- Like other bandwidth preserving servers, it has a budget and is defined in terms of consumption and replenishment rules
- When the budget is non-zero, the server is scheduled with other tasks on an EDF basis
  - The budget and deadline of the server are chosen such that the utilisation of the server is constant when it executes, and that it is always given enough budget to complete the job at the head of its queue each time its budget is replenished
  - The server never has any budget if it has no work to do

Constant Utilisation Server

- Consumption rule:
  - A constant utilisation server only consumes budget when it executes
- Replenishment rules:
  - Initially, budget $e_s = 0$ and deadline $d = 0$
  - When an aperiodic job with execution time e arrives at time t to an empty aperiodic job queue
    - If $t < d$, do nothing (⇒ server is busy; wait for it to become idle)
    - If $t \geq d$ then set $d = t + e/\tilde{u}_s$ and $e_s = e$
  - At the deadline d of the server
    - If the server is backlogged, set $d = d + e/\tilde{u}_s$ and $e_s = e$ (⇒ was busy when job arrived)
    - If the server is idle, do nothing
  - i.e., the server is always given enough budget to complete the job at the head of its queue, with known utilisation, when the budget is replenished

Total Bandwidth Server

- A total bandwidth server improves responsiveness by allowing a server to claim background time not used by the periodic tasks
  - Change the replenishment rules slightly, leave all else the same:
    - Initially, $e_s = 0$ and $d = 0$
    - When an aperiodic job with execution time e arrives at time t to an empty aperiodic job queue, set $d = \max(d, t) + e/\tilde{u}_s$ and $e_s = e$
    - When the server completes the current aperiodic job, the job is removed from the queue and, if the server is backlogged, set $d = d + e/\tilde{u}_s$ and $e_s = e$; if the server is idle, do nothing
  - Always ready for execution when backlogged
  - Assigns at least fraction $\tilde{u}_s$ of the processor to a task

Scheduling Sporadic Jobs

- Consider the problem of scheduling sporadic jobs alongside a system of periodic tasks and aperiodic jobs

- Recall the sporadic job scheduling problem:
  - Based on the execution time and deadline of each newly arrived sporadic job, decide whether to accept or reject the job
  - Accepting the job implies that the job will complete within its deadline, without causing any periodic task or previously accepted sporadic job to miss its deadline
  - Do not accept a sporadic job if cannot guarantee it will meet its deadline

Model for Scheduling Sporadic Jobs

- When sporadic jobs arrive, they are both accepted and scheduled in EDF order
    - In a dynamic-priority system, this is the natural order of execution
    - In a fixed-priority system, the sporadic jobs are executed by a bandwidth preserving server, which performs an acceptance test and runs the sporadic jobs in EDF order
    - In both cases, no new scheduling algorithm is required
- Definitions:
    - Sporadic jobs are denoted by $S_i(r_i, d_i, e_i)$ where r_i is the release time, d_i is the (absolute) deadline, and e_i is the maximum execution time
    - The density of a sporadic job $\Delta_i = e_i/(d_i - r_i)$
        - The total density of a system of n jobs is $\Delta = \Delta_1 + \Delta_2 + \ldots + \Delta_n$
    - The job is active during its feasible interval $(r_i, di]$

Sporadic Jobs in Dynamic-Priority Systems

- Theorem: A system of independent preemptable *sporadic* jobs can be scheduled using EDF if the total density of *all* active jobs in the system ≤ 1 at all times
    - This is the standard scheduling test for EDF systems, but including both periodic and sporadic jobs
    - This test uses the density since deadlines may not equal periods; hence it is a sufficient test, but not a necessary test
- What does this mean?
    - If we can bound the frequency with which sporadic jobs appear to the running system, we can guarantee that none are missed
    - Alternatively, when a sporadic job arrives, if we deduce that the total density would exceed 1 in its feasible interval, we reject the sporadic job (admission control)

Admission Control for Sporadic Jobs/EDF

- At time t there are n active sporadic jobs, stored in non-decreasing order of deadline
  - The deadlines partition the time from t to ∞ into $n + 1$ discrete intervals: $I_1, I_2, \ldots, I_{n+1}$
    - I_1 begins at t and ends at the earliest sporadic job deadline
    - For each $1 \leq k \leq n$, each interval I_{k+1} begins when the interval I_k ends, and ends at the next deadline in the list (or ∞ for I_{n+1})
  - The scheduler maintains the total density $\Delta_{s,k}$ of each interval I_k
- Let I_l be the interval containing the deadline d of the new sporadic job $S(t, d, e)$
  - The scheduler accepts the job if $\underbrace{\frac{e}{d-t}}_{\text{Density of new job}} + \Delta_{s,k} \leq 1 - \Delta$ for all $k = 1, 2, \ldots, l$
  - i.e. accept if the new sporadic job can be added, without increasing the density of any intervals past 1

Admission Control for Sporadic Jobs/EDF

- Notes:
  - This acceptance test is *not* optimal: a sporadic job may be rejected even though it could be scheduled (the result for the schedulable utilisation is based on the *density* and hence is sufficient but not necessary)
  - It is possible to derive a – much more complex – expression taking into account slack time, that is optimal. Unclear if the complexity is worthwhile.
  - This acceptance test assumes every sporadic job is ready for execution when released
    - If this is not the case, must modify the acceptance test to take into account the time when the jobs become ready, rather than their release time, when testing the intervals to see if their density exceeds 1

Sporadic Jobs in Fixed-Priority Systems

- Use a sporadic server to execute sporadic jobs in a fixed-priority system
  - The server (p_s, e_s) has budget e_s units every p_s units of time, so the scheduler can compute the least amount of time available to every sporadic job in the system
  - Assume that sporadic jobs ordered among themselves in EDF
  - When first sporadic job $S_1(t, d_{s,1}, e_{s,1})$ arrives, there is at least $\lfloor (d_{s,1} - t)/p_s \rfloor \cdot e_s$ units of processor time available to the server before the deadline of the job
    - $\lfloor (d_{s,1} - t)/p_s \rfloor$ = number of server periods available
  - Therefore it accepts S_1 if slack of job $\sigma_{s,1}(t) = \underbrace{\lfloor (d_{s,1} - t)/p_s \rfloor e_s}_{\text{Time available}} - \underbrace{e_{s,1}}_{\text{Execution time}} \geq 0$

[cont'd]

Sporadic Jobs in Fixed-Priority Systems

- To decide if a new job $S_i(t, d_{s,i}, e_{s,i})$ is acceptable when there are n sporadic jobs in the system, the scheduler first computes the slack $\sigma_{s,i}(t)$ of S_i:

$$\sigma_{s,i}(t) = \lfloor (d_{s,i} - t)/p_s \rfloor e_s - e_{s,i} - \sum_{d_{s,k} < d_{s,i}} (e_{s,k} - \xi_{s,k})$$

  where $\xi_{s,k}$ is the execution time of the completed part of the existing job S_k The job cannot be accepted if $\sigma_{s,i}(t) < 0$

  - As for $\sigma_{s,1}(t)$, but accounting for the already accepted sporadic jobs

- If $\sigma_{s,i}(t) \geq 0$, the scheduler then checks if any existing sporadic job S_k with deadline after $d_{s,i}$ may be adversely affected by the acceptance of S_i

  - Check if the slack $\sigma_{s,k}(t)$ for each S_k at the time is at least equal to the execution time $e_{s,i}$ of S_i (i.e., S_i is accepted if $\sigma_{s,k}(t) - e_{s,i} \geq 0$ for every existing sporadic job S_k with deadline ≥ $d_{s,i}$)

- This acceptance test for fixed-priority systems is more complex than that for dynamic-priority systems, but is still of reasonable time complexity to be implemented “on-line”

POSIX Scheduling API: Sporadic Server

- Hybrid sporadic/background server included in real time extensions to POSIX
  - Use the SCHED_SPORADIC scheduling policy

```
struct sched_param {
  int              sched_priority;
  int              sched_ss_low_priority;
  struct timespec  sched_ss_repl_period;
  struct timespec  sched_ss_init_budget;
};
```

  - When server has budget, runs at sched_priority, otherwise runs as a background server at sched_ss_low_priority
    - Set sched_ss_low_priority to be lower priority than real-time tasks, but possibly higher than other non-real-time tasks in the system
  - Also defines the replenishment period and the initial budget after replenishment
  - As usual with POSIX, applicable to fixed-priority systems only

Summary

- Have discussed further:
  - Scheduling aperiodic jobs (cont’d)
    - Sporadic servers
    - Constant utilisation servers
    - Total bandwidth servers
    - Weighted fair queuing servers
  - Scheduling sporadic jobs