University of Glasgow

School of Computing Science

60 YEARS OF COMPUTING AT GLASGOW

# Real-time Systems: Scheduling Aperiodic and Sporadic Tasks

Advanced Operating Systems (M)

Lecture 16

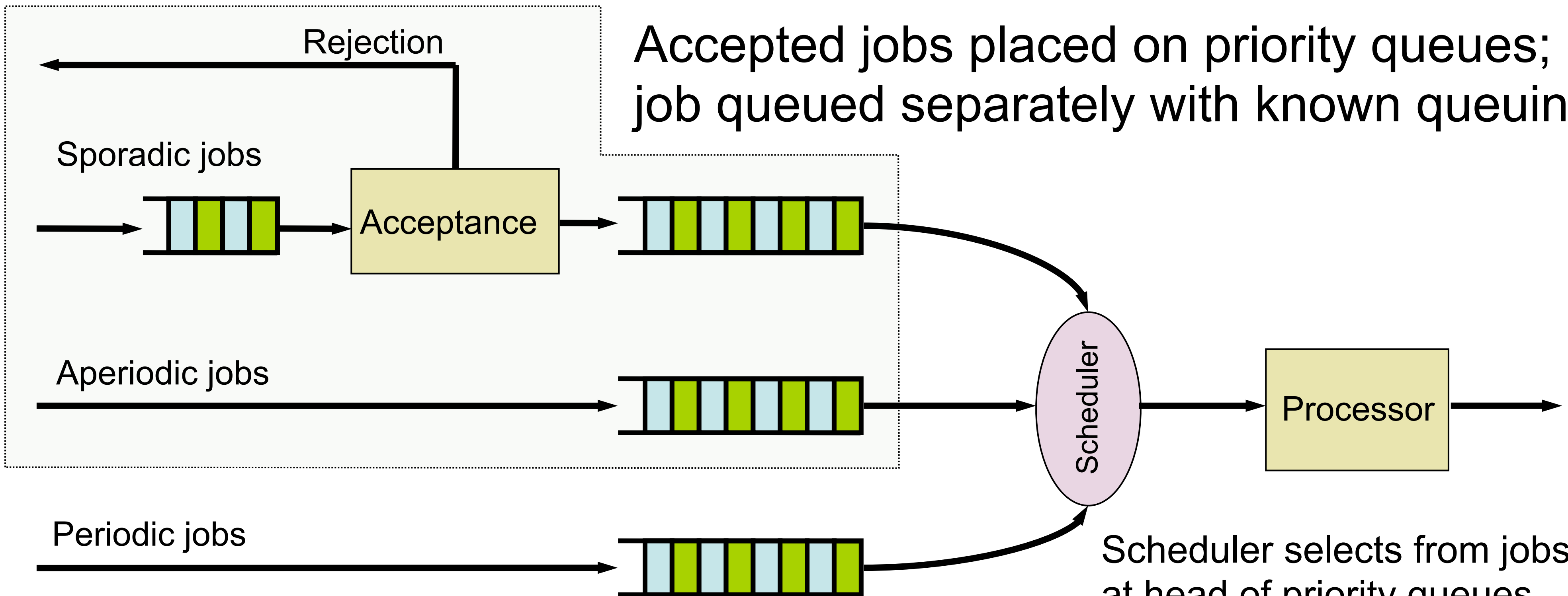
# Lecture Outline

- Aperiodic and sporadic tasks
  - System model
  - Acceptance test concept
- Scheduling aperiodic and sporadic jobs
  - Background execution
  - Sporadic server

# Aperiodic and Sporadic Tasks

- The jobs in an *aperiodic task* have unpredictable release times, but no deadline
  - Problem: schedule the jobs without disrupting correctness of the system
  - Aperiodic jobs are always accepted
- A *sporadic task* is an aperiodic task with a deadline
  - Cannot guarantee systems with sporadic tasks are correct without bounding release or execution time
  - 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; the remainder of the system can still be scheduled
  - If accepted, schedule their jobs without disrupting correctness of rest of the system

# System Model




Accepted jobs placed on priority queues; each type of job queued separately with known queuing discipline

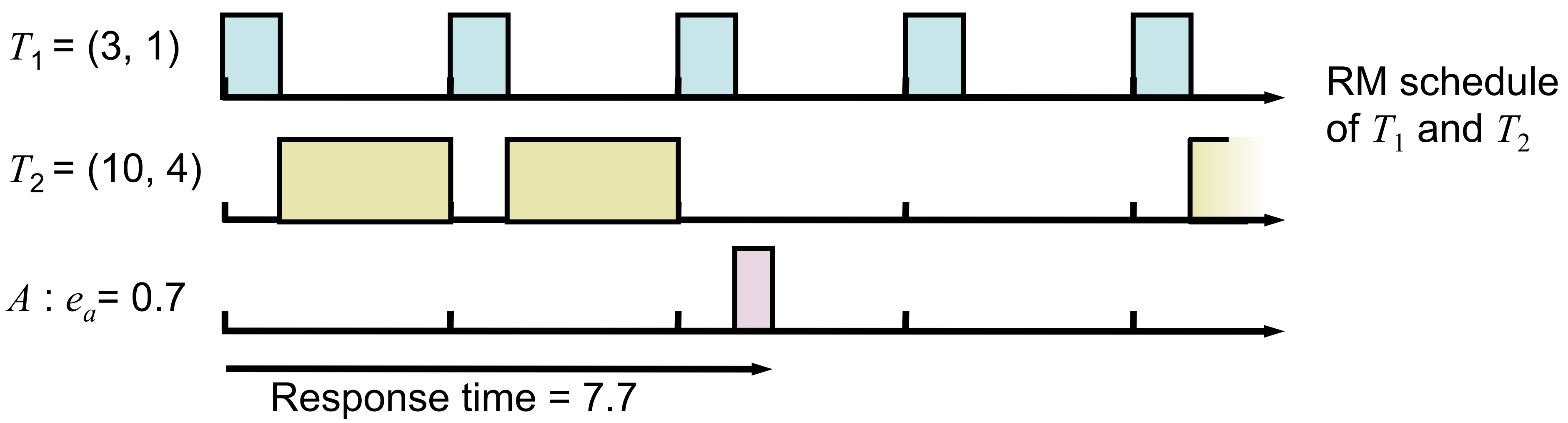
Scheduler selects from jobs at head of priority queues

Single processor; independent, preemptable, periodic tasks can be scheduled in absence of aperiodic and sporadic jobs

Aperiodic and sporadic jobs are preemptable and independent

# Scheduling Aperiodic Jobs

- Run aperiodic jobs as lowest priority in the system: only run if no periodic or sporadic jobs available
- Clearly produces *correct* schedules, and extremely simple to implement
- Response times poor; acceptable as no deadlines



# Scheduling Sporadic Jobs

- How to schedule sporadic jobs alongside a system of periodic tasks and aperiodic jobs?
  - 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
- Requires a *sporadic server* to execute the jobs

# Periodic Servers

- A sporadic server is a particular kind of *periodic server*
- A periodic server is a task that behaves much like a periodic task, but created to execute aperiodic jobs
    - A periodic server, $T_{ps} = (p_{ps}, e_{ps})$ never executes for more than $e_{ps}$ units of time within each period $p_{ps}$
        - The budget of the server is $e_{ps}$
        - Budget consumed when the server is executing, and replenished periodically
    - A periodic server is backlogged if the aperiodic job queue is nonempty
    - A periodic server is scheduled as any other periodic task, except it only executes when scheduled and when it is backlogged and has non-zero budget

# The 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:
    - $\mathrm{BEGIN}$ 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)
    - $\mathrm{END}$ as the end of the latest busy interval in this sequence if this interval ends before $t$; define $\mathrm{END} = \infty$ if the interval ends after $t$

# The 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 $\mathrm{END} < t$
  - When they are not true, the server holds its budget

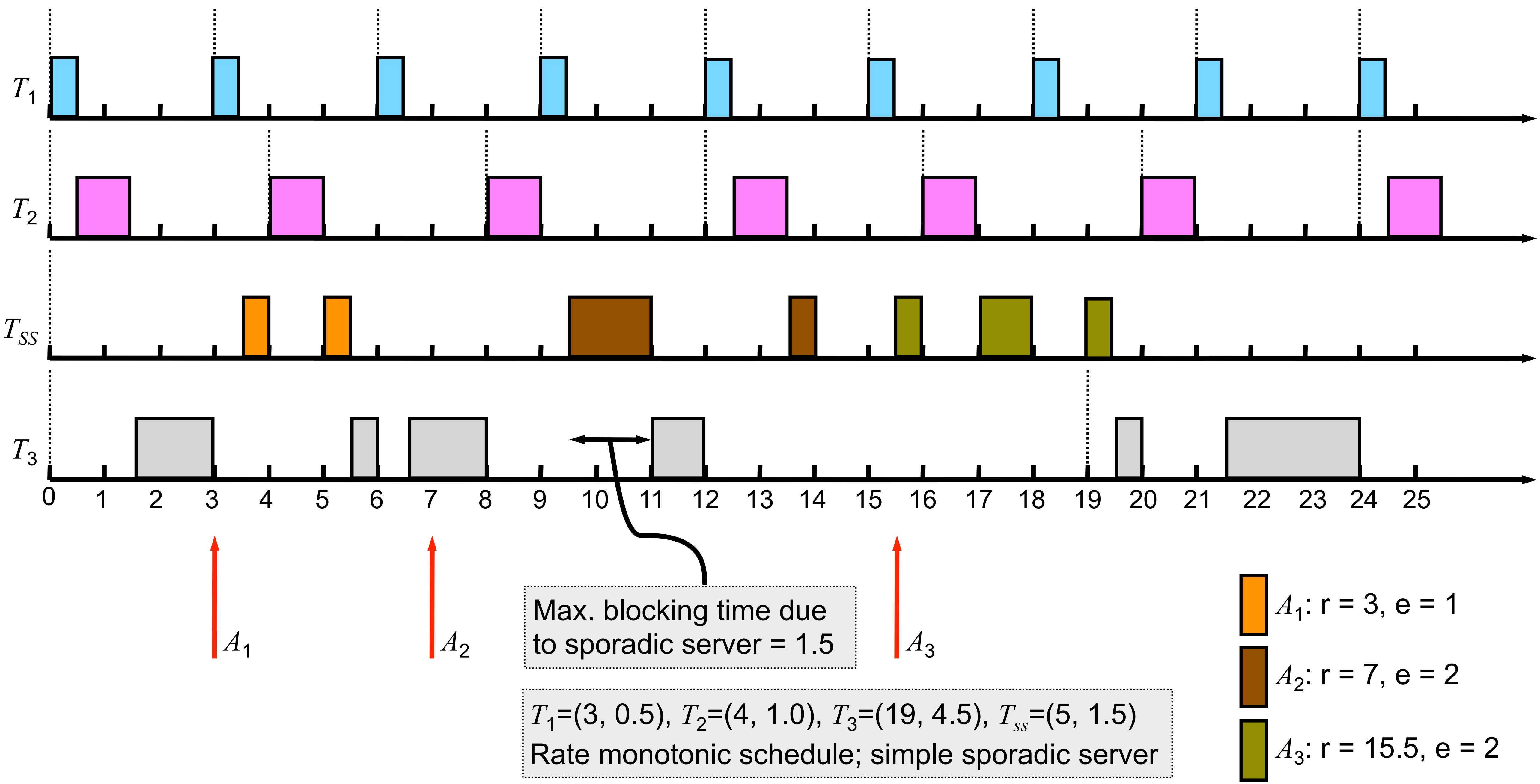
- That is:
  - The server executes for no more time than it has execution budget
  - The server retains its budget if:
    - A higher-priority job is executing, or
    - It has not executed since $t_r$
  - Otherwise, the budget decreases when the server executes, or if it idles while it has budget

# The 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 $\text{END} = t_f$ then
    $t_e = \max(t_r, \text{BEGIN})$
    else if $\text{END} < t_f$ then
    $t_e = t_f$
    The next replenishment time is set to $t_e + p_S$
  - 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 $\min(t_b, t_e + p_S)$

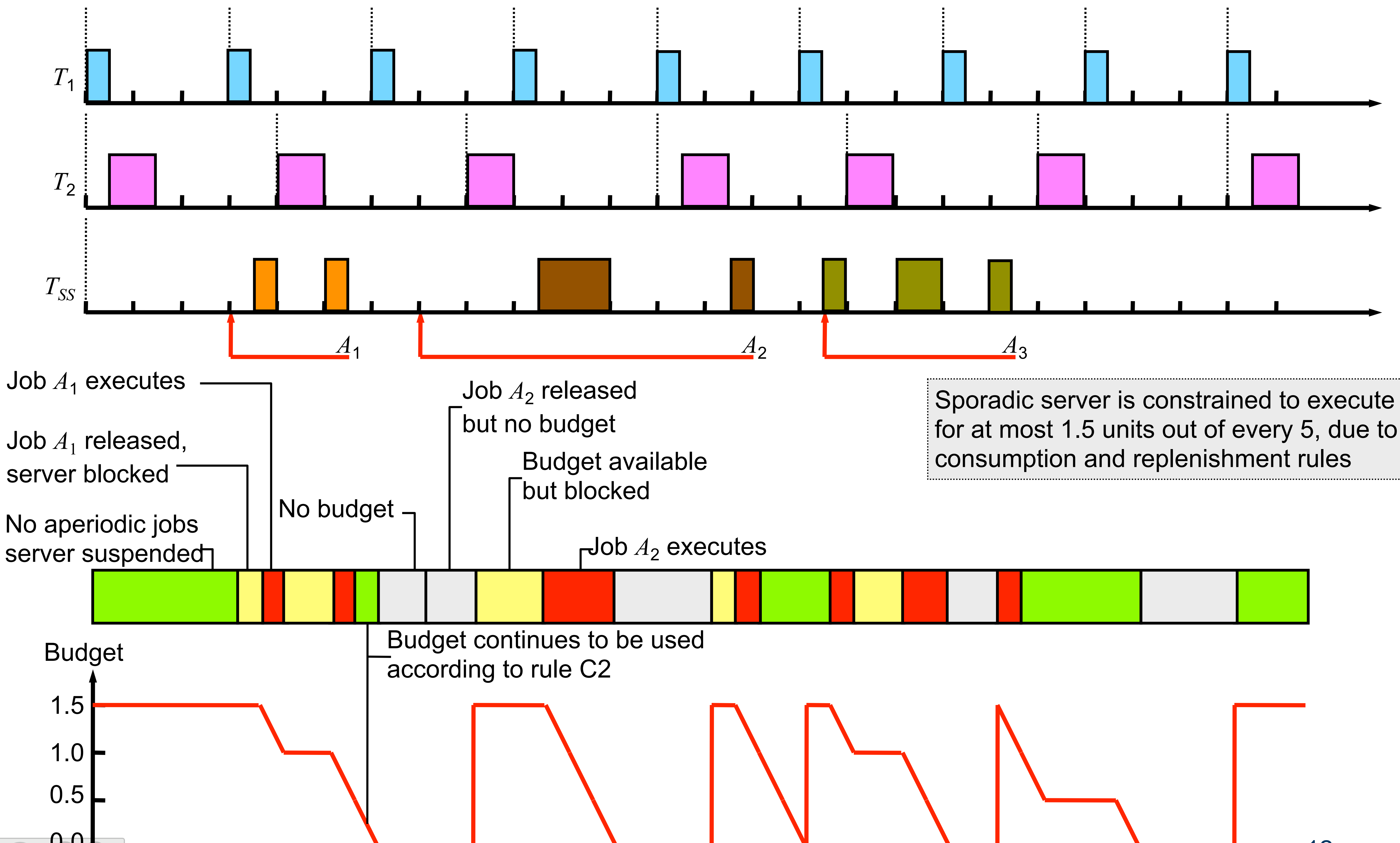
$t_e$ = effective replenishment time

# Example




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

# Example



$T_1$
$T_2$
$T_{SS}$
$A_1$
$A_2$
$A_3$
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
Sporadic server is constrained to execute for at most 1.5 units out of every 5, due to consumption and replenishment rules
Budget continues to be used according to rule C2
Budget
1.5
1.0
0.5
0.0

# Correctness of Schedule

- 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 is sometimes greater than $p_s$, and their execution times less than $e_s$, but this does not affect correctness

# Scheduling Sporadic Jobs

- How to schedule 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 periodic server that 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 $(r_i, d_i]$

# 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 $\infty$ 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 $\infty$ 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 $\frac{e}{d-t} + \Delta_{s,k} \leq 1 - \Delta$ for all $k = 1, 2, \ldots, l$ where $\Delta$ is density of periodic jobs

    Density of new job
  - 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 maximum 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 $\sigma_{s,1}(t) = \lfloor (d_{s,1} - t)/p_s \rfloor \, e_s - e_{s,1} \geq 0$

Time available: $\lfloor (d_{s,1} - t)/p_s \rfloor \, e_s$

Execution time: $e_{s,1}$

[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 $\geq d_{s,i}$)
- 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"

# Practical Usage

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

- Use of sporadic server for scheduling aperiodic tasks – complex to implement, easy to prove correctness
- Scheduling sporadic tasks
  - In EDF systems – density test for correctness
  - In RM systems using sporadic server – complex rules for correctness, but intuition of behaviour straight-forward