University of Glasgow | School of Computing Science

Real-time Scheduling of Aperiodic and Sporadic Tasks (1)

Advanced Operating Systems
Lecture 4

Lecture Outline

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

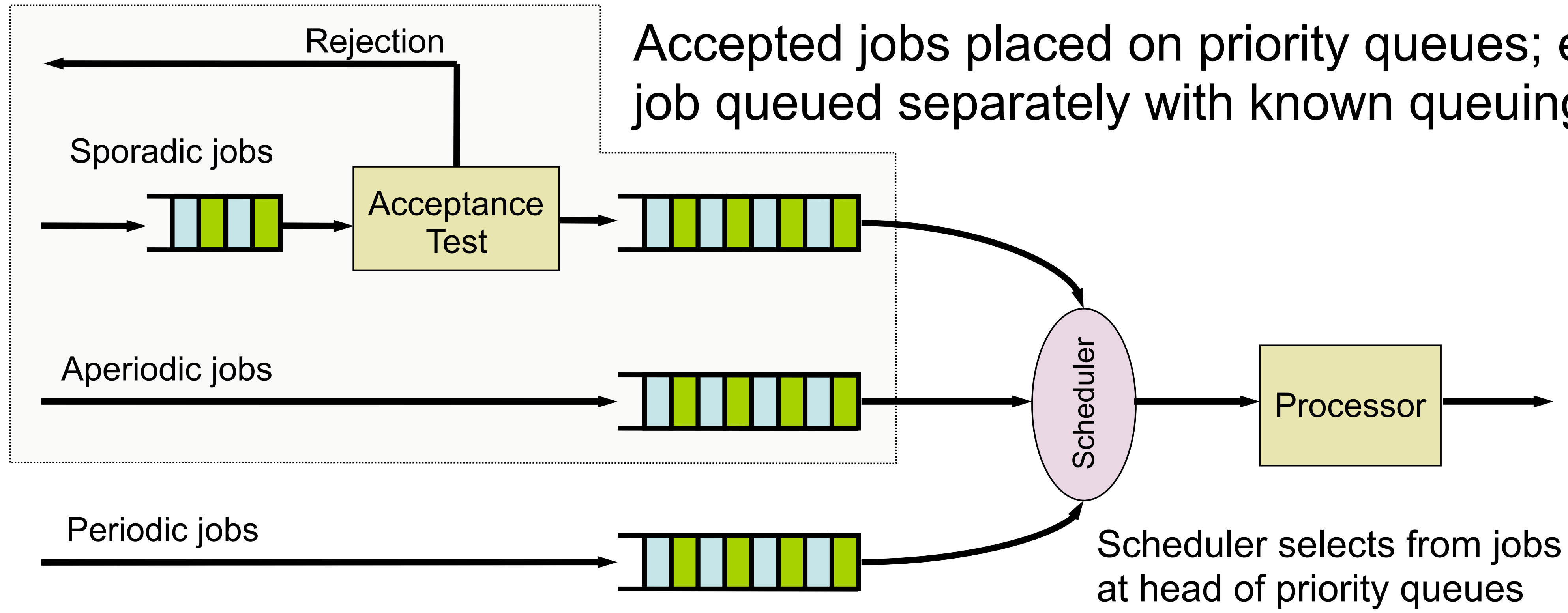
Aperiodic Tasks

- Recall: if jobs have unpredictable release times, a task is termed *aperiodic*
- Problem is to schedule their jobs without disrupting correctness of the system
- Aperiodic jobs are always accepted

Sporadic Tasks

- Recall: a *sporadic* task is an aperiodic task where jobs have deadlines once released
- 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

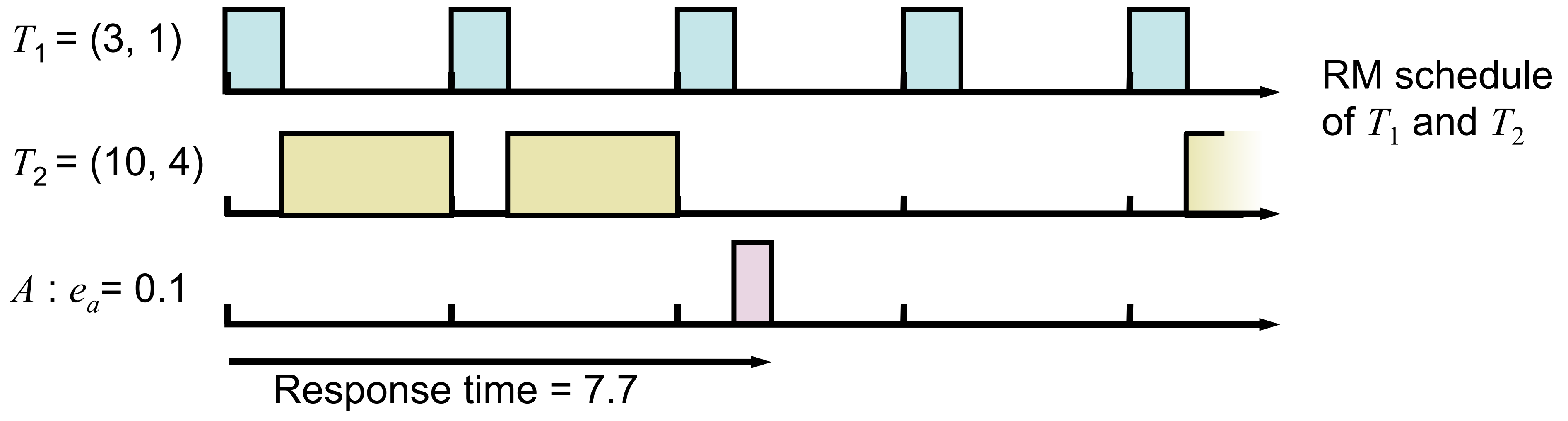
Scheduling Aperiodic Jobs

- Consider the simple case: scheduling aperiodic jobs along with a system of periodic jobs
    - Ignore sporadic jobs for now

- Two basic approaches:
    - Background execution
    - Execution using a periodic server

Background Execution of Aperiodic Jobs

- Aperiodic jobs are scheduled and executed only at times when there are no periodic or sporadic jobs ready for execution
  - Clearly produces *correct* schedules; extremely simple to implement in clock-driven and priority-driven schedulers
  - Not *optimal* since it is almost guaranteed to delay execution of aperiodic jobs in favour of periodic and sporadic jobs, giving unduly long response times for the aperiodic jobs



Periodic Servers

- 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*

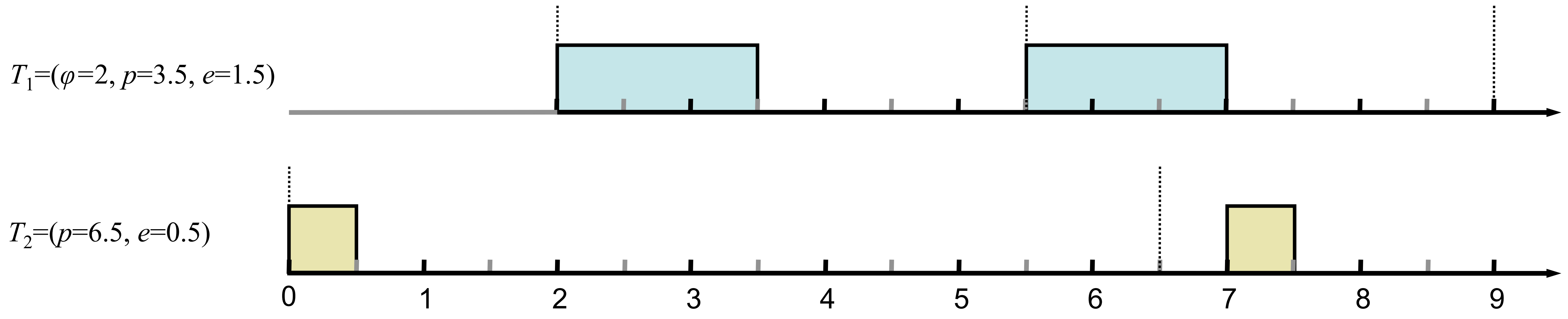
Periodic Servers: Polling

- A common way to schedule aperiodic jobs is using a *polling server* – simplest periodic server
    - A periodic server $T_{ps} = (p_s, e_s)$ is scheduled
    - When executed, it examines the aperiodic job queue:
        - If an aperiodic job is in the queue, it is executed for up to e_s time units
        - If the aperiodic queue is empty when polled, the server suspends and gives up it's budget
        - The budget is replenished to e_s every p_s time units (and doesn't carry over to next period)
- Simple to prove correctness – treat as periodic task
    - For clock-driven or fixed-priority systems; dynamic-priority systems might suffer scheduling anomalies
    - Gives slow response for tasks that arrive just after the polling instant

Periodic Servers: Deferrable

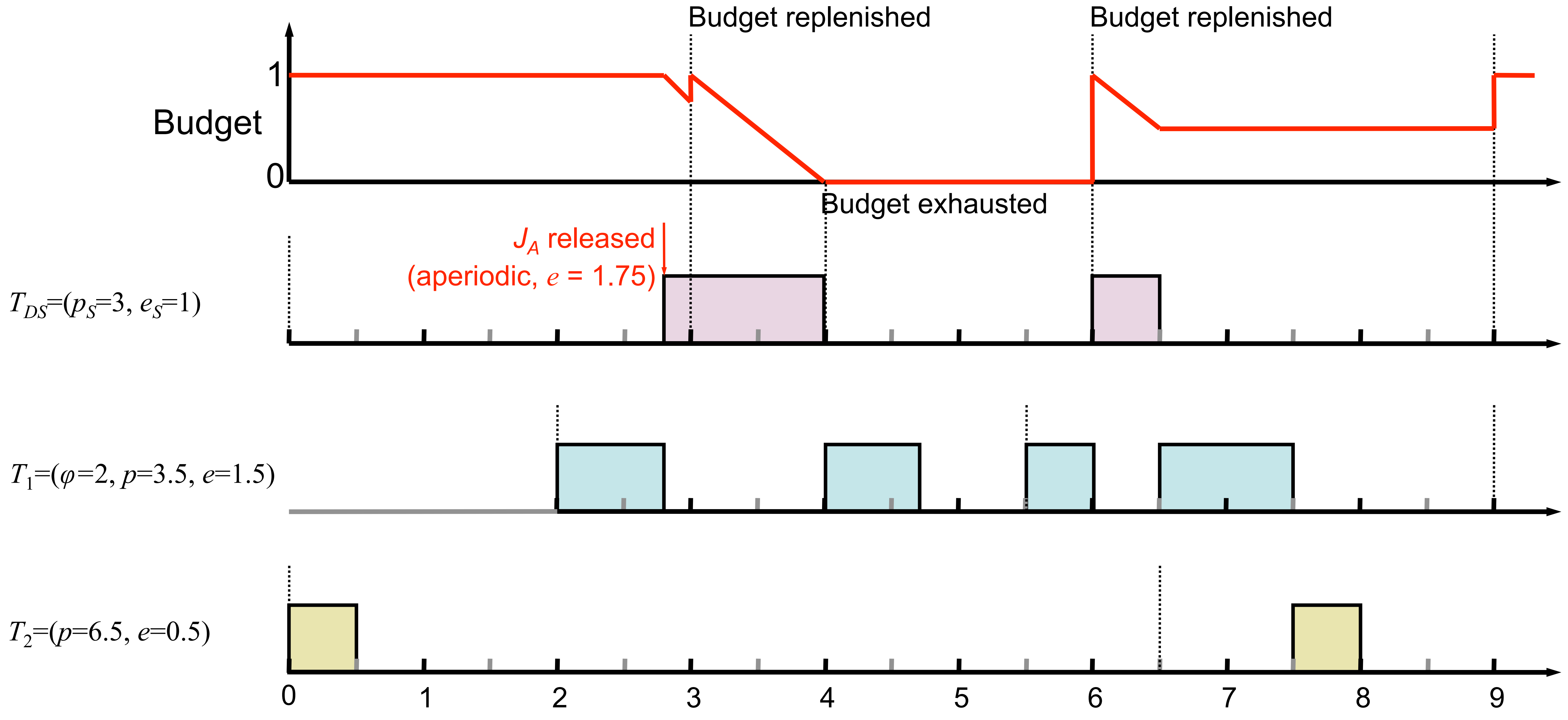
- An alternative periodic server
- Budget consumption and replenishment:
  - The budget is consumed when the server executes
  - Once scheduled, unused budget is retained throughout the period, to be used whenever there are aperiodic jobs to execute – i.e., if a job misses the polling instant, it can be scheduled later in the period
  - The budget is replenished to e_S at multiples of the period, but cannot carry over budget from period to period
- Improves response time of aperiodic jobs

Deferrable Server: Example with RM



Periodic tasks T_1 and T_2 are scheduled according to the rate monotonic algorithm

Deferrable Server: Example with RM



Add the deferrable server, scheduled according to the rate monotonic priority, but with the budget consumption and replenishment rules affecting its execution time

The deferrable server is usually run at highest priority, but this is not strictly required

Deferrable Server: Scheduling in RM (1)

- Maximum utilisation test fails
  - Utilisation varies depending on arrival times of jobs executed by server
  - Use time demand analysis based on critical instants to determine if the system can be scheduled
- Finding the critical instants:
  - Assume a fixed-priority system T in which $D_i \leq p_i \; \forall \; i$ scheduled with a deferrable server (p_S, e_S) that has the highest priority among all tasks
  - A critical instant of every periodic tasks T_i occurs at a time t_0 when all of the following are true:
    - One of its jobs $J_{i,c}$ is released at t_0
    - A job in every higher-priority periodic task is released at t_0
    - The budget of the server is e_S at t_0, one or more aperiodic jobs are released at t_0, and they keep the server backlogged hereafter
    - The next replenishment time of the server is $t_0 + e_S$

Deferrable Server: Scheduling in RM (2)

- The definition of critical instant is identical to that for the periodic tasks without the deferrable server + the worst-case requirements for the server

- The time-demand function is: $$w_i(t) = \underbrace{e_i}_{\text{Execution time of job } J_i} + \underbrace{\sum_{k=1}^{i-1} \left\lceil \frac{t}{p_k} \right\rceil e_k}_{\text{Execution time of higher priority jobs started during this interval}} + \underbrace{e_s + \left\lceil \frac{t - e_s}{p_s} \right\rceil e_s}_{\text{Execution time of deferrable server}}$$

- To determine whether the task T_i is can be schedule, we simply have to check whether $w_i(t) \leq t$ for some $t \leq D_i$
- Remember, this is a sufficient condition, not necessary – i.e., if this condition is not true, the schedule might still be correct

Deferrable Server: Scheduling in RM (3)

- In general, no maximum utilisation test for a fixed-priority system with a deferrable server
  - One special case: a system of n independent, preemptable periodic tasks, whose periods satisfy $p_s < p_1 < p_2 < \ldots < p_n < 2p_s$ and $p_n > p_s + e_s$, where the relative deadlines equal their respective periods, can be scheduled rate-monotonically with a deferrable server provided $U < U_{RM/DS}(n)$ where:

$$U_{RM/DS}(n) = (n-1)\left\lfloor \left(\frac{u_s+2}{u_s+1}\right)^{\frac{1}{(n-1)}} - 1 \right\rfloor$$

Deferrable Server: Scheduling in EDF

- It is easier to reason about the schedulability of a deadline-driven system with a deferrable server
  - The deadline of a deferrable server is its next replenishment time
  - A periodic task T_i in a system of N independent, preemptable, periodic tasks is schedulable with a deferrable server with period p_S, execution budget e_S and utilization u_S, according to the EDF algorithm if:

$$\sum_{k=1}^{N} \frac{e_k}{\min(D_k, p_k)} + u_s\left(1 + \frac{p_s - e_s}{D_i}\right) \leq 1$$

  - Must be calculated for each task in the system, since D_i included

  - Example: tasks T_1=(3, 0.6), T_2=(5.0, 0.5), T_3=(7, 1.4) scheduled with a deferrable server p_s=4, e_s=0.8
  - The left-hand side of the above inequality is 0.913, 0.828 and 0.792 respectively; hence the three tasks are schedulable

Summary

- Aperiodic and sporadic tasks
- System model – acceptance tests
- Approaches to scheduling aperiodic tasks
    - Background execution
    - Polling server – simple; inefficient
    - Deferrable server – more efficient; too complex to prove correctness for RM; effective for EDF systems