A "Simple" Fixed-Priority Sporadic Server

- Consider a system T of N independent preemptable periodic tasks, plus a single sporadic server task with parameters (p_s, e_s)
  - Tasks are scheduled using a fixed-priority algorithm; system schedulable if we assume (p_s, e_s) behaves as a standard periodic task

- Definitions:
  - T_H is the subset of periodic tasks with higher priorities than the server
    - That subset may be *idle* when no job in T_H is ready for execution, or *busy*
  - Define t_r as the last time the server budget replenished
  - Define t_f as 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

A "Simple" Fixed-Priority Sporadic Server

- Consumption rule:
  - At any time t after t_r, if the server has budget and if either of the following two conditions is true, the server's 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

A "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 $t_e + p_S$.

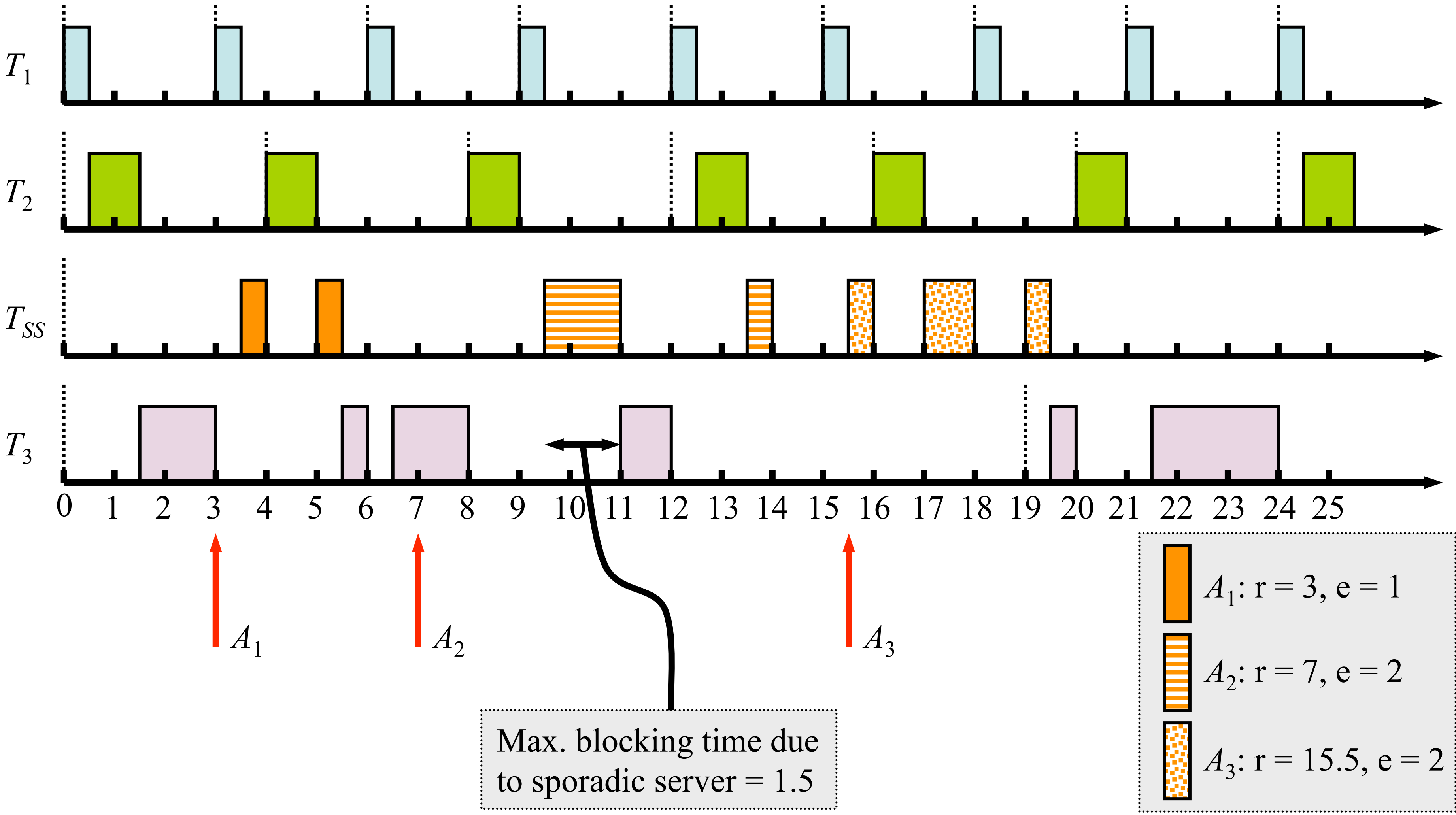
t_e = the *effective replenishment time*

R3: The next replenishment occurs at the next replenishment time ($= t_e + p_S$), except under the following conditions:

(a) If $t_e + p_S$ is earlier than t_f the budget is replenished as soon as it is exhausted

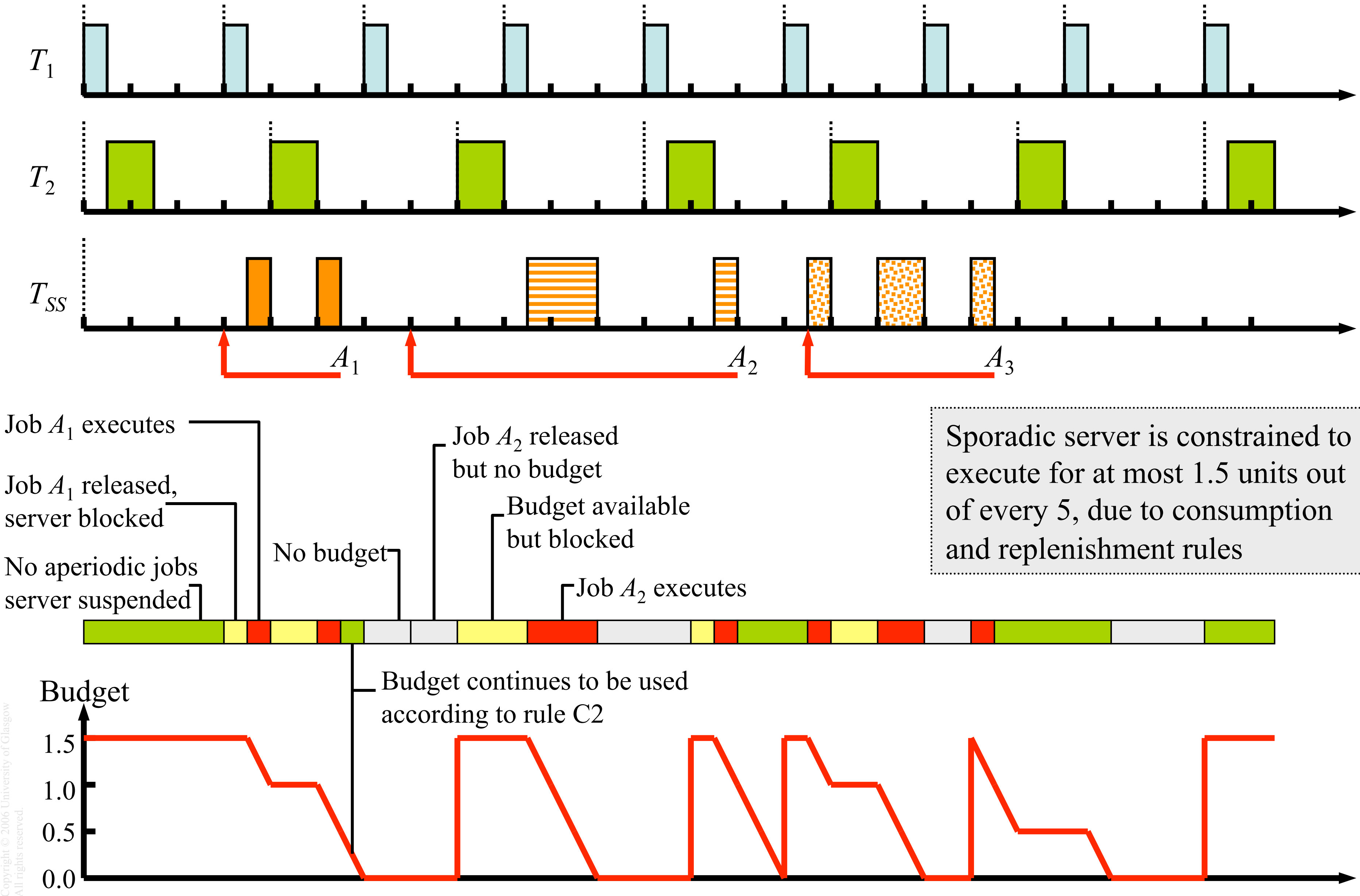
(b) If T becomes idle before $t_e + p_S$, and becomes busy again at t_b, the budget is replenished at $\min(t_b, t_e + p_S)$

Example: Fixed-Priority Sporadic Server



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: Fixed-Priority Sporadic Server



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

Constant Utilization Server

- Consumption rule:
  - A constant utilization 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 utilization, when the budget is replenished

Total Bandwidth Server

- A constant utilization server gives a known fraction of processor capacity to a task; but cannot claim unused capacity to complete the task earlier
- 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

Weighted Fair Queuing Server

- Aim of the constant utilization and total bandwidth servers is to assign some fraction of processor capacity to a task
- When assigning capacity there is the issue of *fairness*:
  - A scheduling algorithm is *fair* within any particular time interval if the fraction of processor time in the interval attained by each backlogged server is proportional to the server size
    - Not only do all tasks meet their deadline, but they all make continual progress according to their share of the processor, no *starvation*
  - Constant utilization and total bandwidth servers are fair on the long term, but can diverge significantly from fair shares in the short term
    - Total bandwidth server partly by design, since it uses background time, but also has fairness issues when there is no spare background time

- As we discuss in lecture 16, the *weighted fair queuing* algorithm can also be used to share processor time between servers, and is designed to ensure fairness in allocations

Deferrable Server

- The simplest bandwidth-preserving server
  - Improves response time of aperiodic jobs, compared to polling server

- Consumption rule:
  - The budget is consumed at the rate of one per unit time whenever the server executes
  - Unused budget is retained throughout the period, to be used whenever there are aperiodic jobs to execute
    - Instead of discarding the budget if no aperiodic job to execute at start of period, keep in the hope a job arrives
- Replenishment rule:
  - The budget is set to e_S at multiples of the period
    - i.e. time instants $k \cdot p_S$, for $k = 0, 1, 2, \ldots$
  - Note: the server is not allowed to carry over budget from period to period

Fixed- and Dynamic-Priority Algorithms

- A priority-driven scheduler is an on-line scheduler
  - It does *not* pre-compute a schedule of tasks/jobs: instead assigns priorities to jobs when released, places them on a run queue in priority order
  - When pre-emption is allowed, a scheduling decision is made whenever a job is released or completed
  - At each scheduling decision time, the scheduler updates the run queues and executes the job at the head of the queue

- Jobs in a task may be assigned the same priority (*task level fixed-priority*) or different priorities (*task level dynamic-priority*)
- The priority of each job is usually fixed (*job level fixed-priority*); but some systems can vary the priority of a job after it has started (*job level dynamic-priority*)
  - Job level dynamic-priority usually very inefficient