Scheduling Aperiodic and Sporadic Jobs

- Definitions

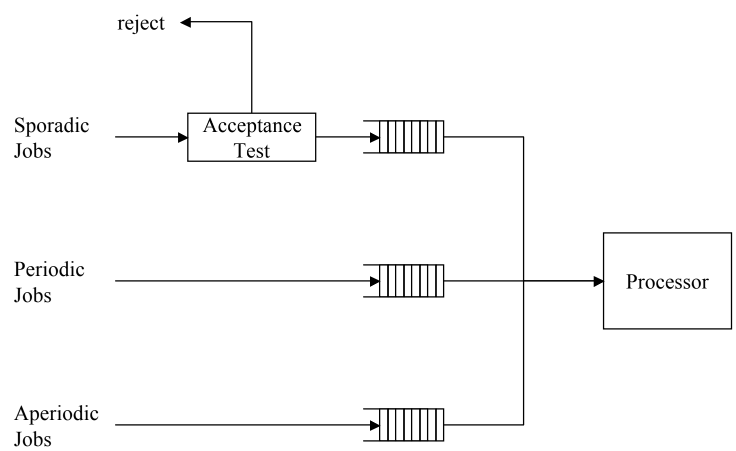
- Polling Server
- Deferrable Server
- Sporadic Server
- Generalized Processor Sharing
- Constant Utilization Server
- Total Bandwidth Server
- Preemptive Weighted Fair Queuing

Scheduling Aperiodic and Sporadic Jobs

- When variations in inter-release times and execution times are small:
  - can treat task as periodic task $T=(p_s, e_s)$, and schedule it accordingly.
- What about sporadic jobs? can arrive at any time
  execution times vary widely
  deadlines are unknown *a priori*?

- Given: n periodic tasks $T_1, \ldots, T_i = (p_i, e_i), \ldots, T_n$
  priority-driven scheduling algorithm
- We want to determine when to execute aperiodic and sporadic jobs, *i.e.,*
  - sporadic job: acceptance test
    scheduling of accepted job
  - aperiodic job: schedule job to complete ASAP.

Priority Queues for Periodic/Sporadic/Aperiodic Jobs



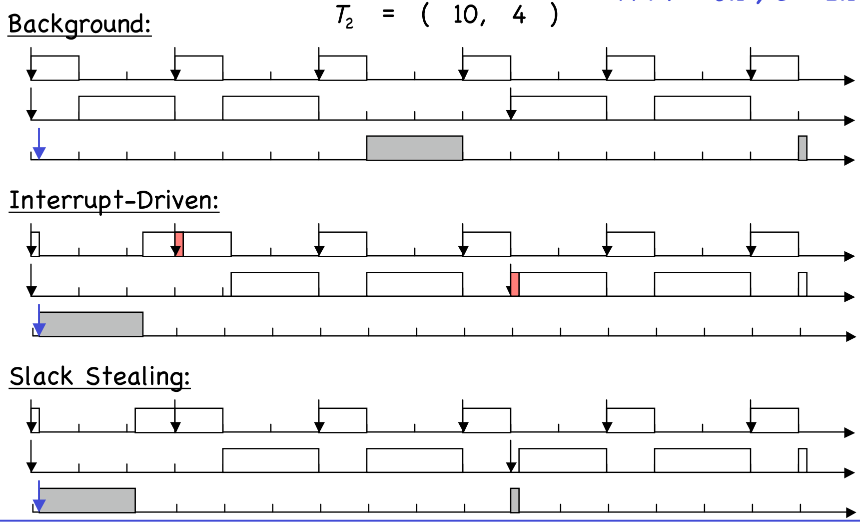
Background/Interrupt-Driven *vs.* Slack Stealing

- Background:
  - Aperiodic job queue has always lowest priority among all queues.
  - Periodic tasks and accepted jobs always meet deadlines.
  - Simple to implement.
  - Execution of aperiodic jobs may be unduly delayed.
- Interrupt-Driven:
  - Response time as short as possible.
  - Periodic tasks may miss some deadlines.
- Slack Stealing:
  - Postpone execution of periodic tasks only when it is safe to do so:
    - Well-suited for clock-driven environments.
    - What about priority-driven environments? (quite complicated)

Examples

$T_1 = (3, 1)$
$T_2 = (10, 4)$

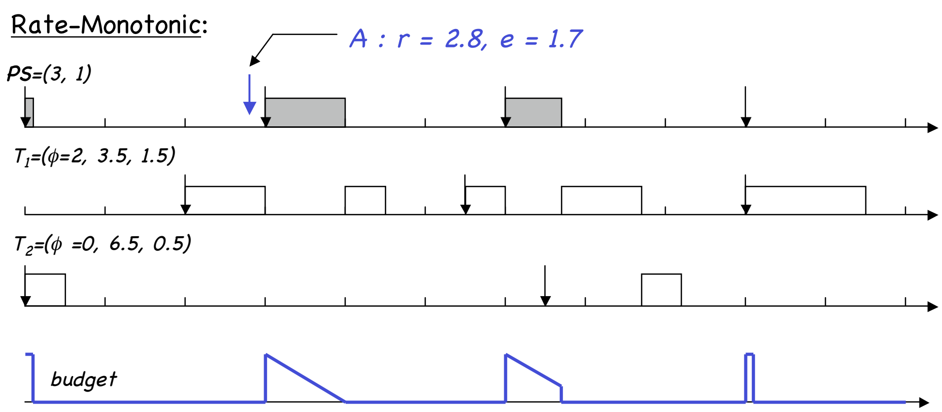
A : r = 0.1 , e = 2.1



Polled Execution *vs.* Bandwidth Preserving Servers

- Polling Server (p_s, e_s): scheduled as periodic task.
  - p_s : Poller ready for execution every p_s time units.
  - e_s : Upper bound on execution time.
- Terminology:
  - (Execution) budget: e_s
  - Replenishment: set budget to e_s at beginning of period.
  - Poller consumes budget at rate 1 while executing aperiodic jobs.
  - Poller exhausts budget whenever poller finds aperiodic queue empty.
  - Whenever the budget is exhausted, the scheduler removes the poller from periodic queue until replenished.
- Bandwidth-preserving server algorithms:
  - Improve upon polling approach
  - Use periodic servers
  - Are defined by consumption and replenishment rules.

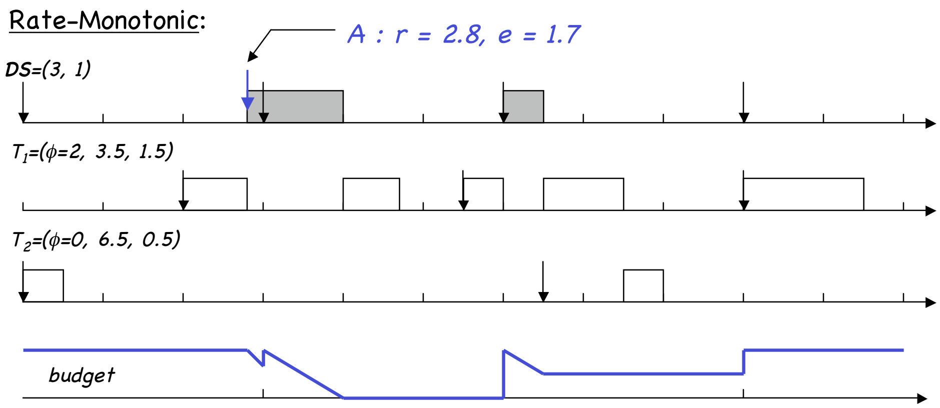
Example: Polling Server



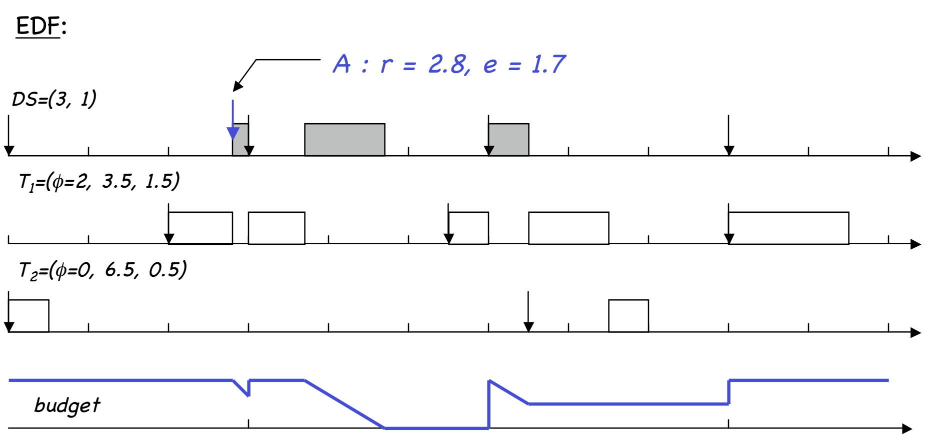
Deferrable Servers

- Rules:
  - Consumption: Execution budget consumed only when server executes.
  - Replenishment: Execution budget of server is set to e_s at each multiple of p_s.
- Preserves budget when no aperiodic job is ready.
- Any budget held prior to replenishment is lost (no carry-over).

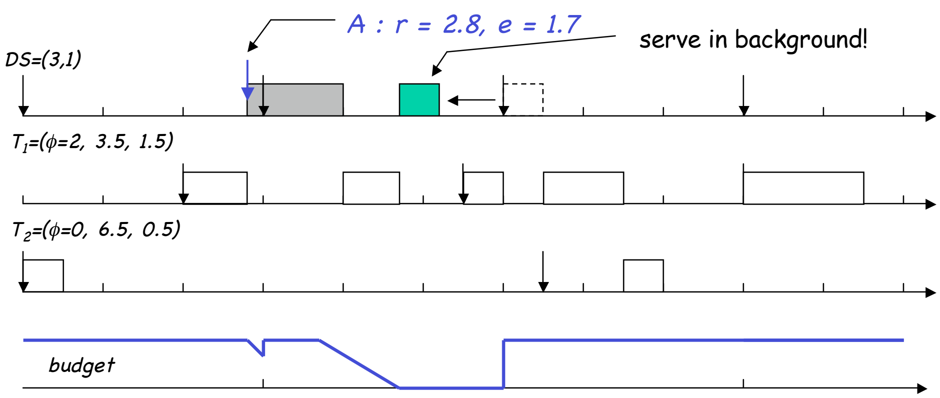
Example: Deferrable Server with RM



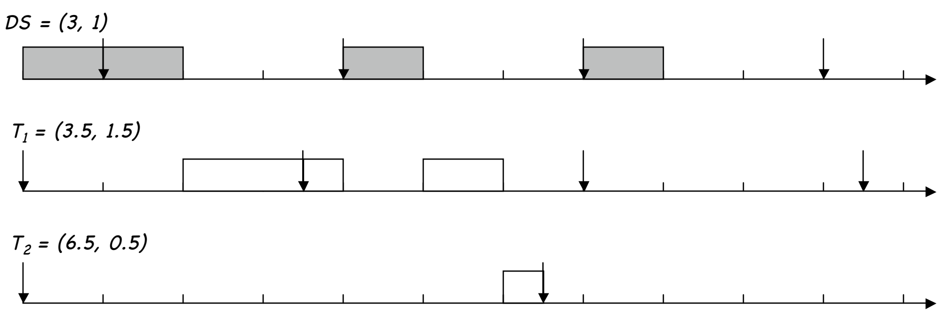
Example: Deferrable Server with EDF



Combination of Deferrable Server with Background Server



DS: Why not Increase the Budget?



Schedulability for Static-Priority Systems
(DS has highest priority)

- Lemma: In a static-priority periodic system with $D_i <= p_i$, with a deferrable server $T_{DS}(p_s, e_s)$ with highest priority, a critical instant for T_i happens when:
  (1) $r_{i,c} = t_0$ for some job $J_{i,c}$ in T_i.
  (2) jobs of higher-priority tasks are released at time t_0.
  (3) budget of (backlogged) server is e_s at time t_0.
  (4) next replenishment time is $t_0 + e_s$.

- Intuitively: Low-priority tasks suffer from a "back-to-back" hit by the deferrable server.

Time-Demand Analysis

$$w_i(t) = e_i + e_s + \lceil \frac{t - e_s}{p_s} \rceil e_s + \sum_{k=1}^{i-1} \lceil \frac{t}{p_k} \rceil e_k$$

- Schedulable Utilization:
  - Generally, no known schedulable utilization.
  - Only exception:
    - $p_s < p_1 < p_2 < \ldots < p_n < 2p_s$
    - $p_i = D_i$
    - rate-monotonic scheduling
    - $p_n > p_s + e_s$
  - For this case, the schedulable utilization is

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

Deferrable Servers and Arbitrary Static Priority

- Problem: Any budget that is not consumed at end of server period is lost.
- Maximum amount of time DS can consume depends on
  - Release time of all periodic jobs (with respect to replenishment times)
  - Execution times of all tasks.
- Upper bound on time demand for lower-priority tasks than *DS* :

$$w_i(t) \leq e_i + e_s + \lceil \frac{t - e_s}{p_s} \rceil e_s + \sum_{k=1}^{i-1} \lceil \frac{t}{p_k} \rceil e_k$$

- Multiple deferrable servers:
  - Time demand for task with priority lower than *m* *DS*'s:

$$w_i(t) \leq e_i + \sum_{q=1}^{m} \left(1 + \lceil \frac{t - e_{s,q}}{p_{s,q}} \rceil \right) e_{s,q} + \sum_{k=1}^{i-1} \lceil \frac{t}{p_k} \rceil e_k$$

Using the Schedulable Utilization:

- Assume that T_i has lower priority than server.
- $T_{DS}(p_s, e_s)$ behaves like a periodic task (p_s, e_s), except that it may execute for at most e_s additional time units during the interval $(r_{i,c}, r_{i,c}+D_i)$.

$$\sum_{k=1}^{i} u_k + u_s + \frac{e_s + b_i}{p_i} \leq U_X(i+1)$$

scheduling algorithm

- Example:

T_1	(3,	0.6)
T_{DS}	(4,	0.8)
T_2	(5,	0.5)
T_3	(7,	1.4)

- T_1: not affected by T_{DS}
- T_2: $\sum_{k=1}^{2} u_k + u_s + \frac{e_s}{p_2} = 0.66 \leq 0.7797 = U_{RM}(3)$
- T_3: $\sum_{k=1}^{3} u_k + u_s + \frac{e_s}{p_3} = 0.8143 \not\leq 0.757 = U_{RM}(4)$ no!

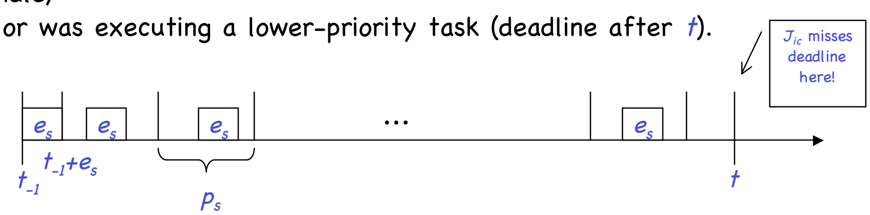
Schedulability for Deadline-Driven Systems

- Lemma: A periodic task T_i in a system of n independent, preemptive periodic tasks is schedulable with a *DS* with period p_s, execution time e_s, and utilization u_s, according to the EDF algorithm if

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

Proof

Proof:
- Let t be the deadline of some Job $J_{i,c}$.
- Let t_{-1} be the last point in time before t where either processor idle,
  or was executing a lower-priority task (deadline after t).



If J_{ic} misses deadline at time t, total amount of processor time consumed by Deferrable Server during interval $(t_{-1}, t]$ is bounded by :

$$e_s + \lfloor \frac{t - t_{-1} - e_s}{p_s} \rfloor e_s$$

Proof (II)

Proof:

- Time consumed by deferrable server:

$$w_{DS}(t-t_{-1}) = e_s + \lfloor \frac{t-t_{-1}-e_s}{p_s} \rfloor e_s \leq e_s + \frac{t-t_{-1}-e_s}{p_s} e_s = u_s(t-t_{-1}+p_s-e_s)$$

- Time consumed by Task T_k:

$$w_k(t-t_{-1}) = e_k \lfloor \frac{t-t_{-1}}{p_k} \rfloor \leq e_k \frac{t-t_{-1}}{p_k}$$

- We used "floor" instead of "ceiling" because last invocation has deadline after t.
- We miss deadline if we don't have enough time to finish by time t:

$$t-t_{-1} < \sum_{k=1}^{n} \frac{e_k}{p_k}(t-t_{-1}) + u_s(t-t_{-1}+p_s-e_s)$$

- Divide by $(t - t_{-1})$ and go from there.

Sporadic Servers

- Problem with Deferrable Server: $T_{DS}(p_s, e_s)$ may delay lower-priority jobs longer than periodic task $T(p_s, e_s)$.
- Sporadic Server (SS): Never use more time than the periodic task $T(p_s, e_s)$ with same parameters.
- If so, we can treat T_{SS} just as a periodic task.

Sporadic Servers in Static-Priority Systems

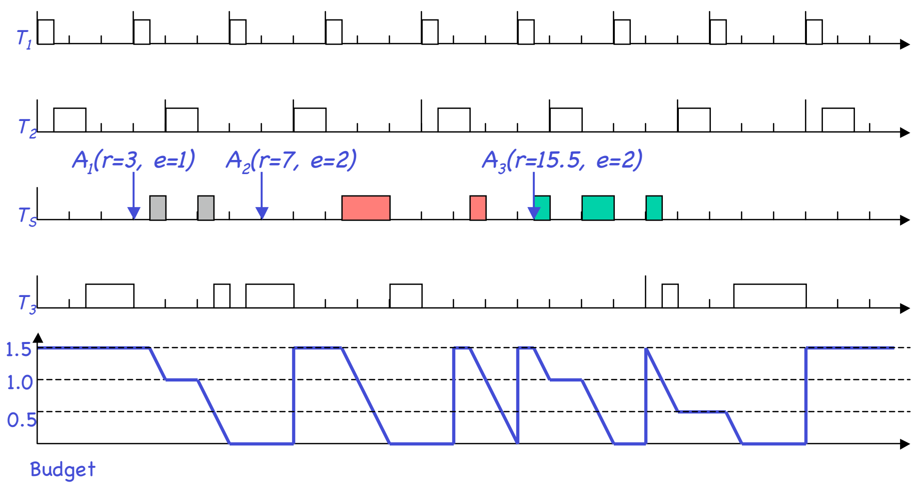
- Notation:
  - T : Task system with n tasks.
  - T_{SS} : Sporadic server, arbitrary priority.
  - T_H : Subset of T with higher priority than T_{SS}.
  - t_r : Latest replenishment time.
  - t_f : First instant after t_r at which server begins to execute.
  - t_e : *Effective* replenishment time.
- The scheduler determines t_e based on history and sets next replenishment time to $t_e + p_s$.

Simple Sporadic Server

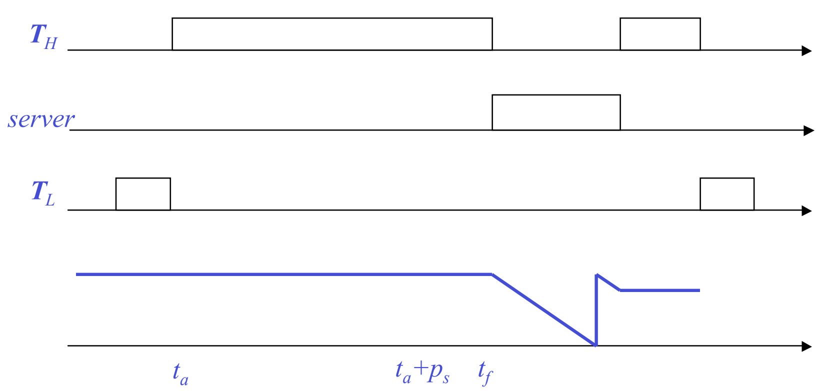
- Consumption Rules: The server's execution budget is consumed at the rate of one at any time t after t_r until the budget is exhausted whenever the following two conditions are true. When these conditions are not true, the server holds its budget:
  - *C1*: The server is executing.
  - *C2*: The server has executed since t_r and is suspended at the time t, and T_H is idle.
- Replenishment Rules:
- *R1*: The execution budget is set to e_s and the current time t_r is recorded initially when the system begins execution and each time when the budget is replenished.
- *R2*: The next budget replenishment time is determined at time t_f when the server first begins to execute since t_r. At time t_f, t_e is set to the latest time instant at which a lower-priority task executes in (t_r, t_f), and set to t_r if T_H is busy throughout this interval. 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 when the replenishment may be done sooner or later.
  - (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) The budget is replenished at time t whenever the system T has been idle before t and a periodic job is released at t.

Simple Sporadic Server: Example

T_1 = (3, 0.5) T_2 = (4, 1.0) T_3 = (19, 4.5) T_S = (5, 1.5)



A Situation where Rule 3a Applies



Informal Proof of Correctness

- "Correctness": The server never demands more time in any interval than corresponding periodic task $T_s = (p_s, e_s)$.
- For now: T has not been idle, and Rule *R3(b)* has never applied.
- We show that server "emulates" Task $T_s=(p_s,e_s)$.
- For this, we view replenishment time as nominal "release time" of server job.

- Rule *C1*: Each server job never executes for more than its budget e_s..
- Rule *C2*: Budget of idle sporadic server decreases as if server was executing.

⇒ Each server job only executes at times when a job of T_s would.

- On the other hand: *C2* means that server holds on to budget when
  - Job in T_H is executing. (obviously correct)
  - Server has not executed since t_r. (Actual release time can be later than nominal release time.)

Informal Proof of Correctness (cont.)

- Rules *R2* and *R3(a)* make sure that next replenishment time always set p_s time units later than effective release time t_e. The next effective release time is never earlier than next replenishment time.

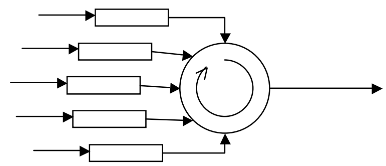
- *R2*: Make effective release time as early as possible.

- *R3(a)*: Emulates situation where job in T_s takes more time to complete than one period.

- *R3(b)*: Applicable only when busy interval of periodic task system ends, and a new one starts.
  Behavior of tasks in old busy period does not affect new busy period.
  This condition is already accounted for in schedulability analysis of T and T_s.

Emulating Generalized Processor Sharing

- Generalized Processor Sharing (bit-by-bit Round Robin):



- Timing isolation.
- Emulate GPS by (for example)
  - Constant Utilization Servers
  - Total Bandwidth Servers
  - Weighted-Fair-Queuing
- Structure: Run server algorithm on top of EDF scheduler.

Scheduling Sporadic Jobs with EDF

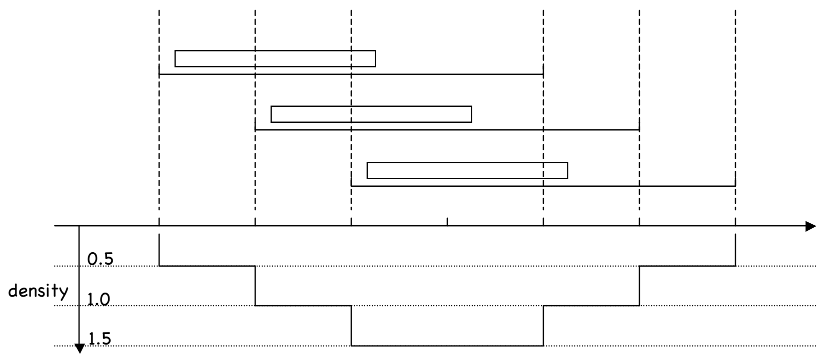
- **Definition**: *Density* of sporadic job J_i with release time r_i, maximum execution time e_i and deadline d_i:

$$density_i = e_i/(d_i - r_i).$$

- **Theorem**: A system of independent, preemptable sporadic jobs is schedulable according to EDF if the total density of all active jobs in the system is no greater than 1 at all times.

Scheduling Sporadic Jobs with EDF (cont)

- Theorem is not necessary!
- Example:



Scheduling Sporadic Jobs with EDF (cont)

- Sporadic task S_i as stream of sporadic jobs S_{i1}, S_{i2}, S_{i3}, ...
- Execution time of S_{ij} is e_{ij}.
- "Period" p_{ij} is time between invocation of S_{ij} and $S_{i(j+1)}$.

- Instantaneous utilization of sporadic job S_{ij} : e_{ij}/p_{ij}.
- Instantaneous utilization of sporadic task S_i : $u_i = max_j(e_{ij}/p_{ij})$.

- **Corollary:** A system of n independent, preemptable sporadic tasks, which is such that the relative deadline of every job is equal to its period, is schedulable on a processor according to the EDF algorithm if the total instantaneous utilization is equal or less to 1.

Constant Utilization Server Algorithm

- A Constant Utilization Server emulates a sporadic task with a constant instantaneous utilization.
- Consumption rule:
  - A server consumes its budget only when it executes.
- Replenishment rules (assume: server is allocated utilization u_s):

  R1 Initially, set $e_s := 0$ 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 $t >= d$, set $e_s := e$ and $d := t + e_s/u_s$.

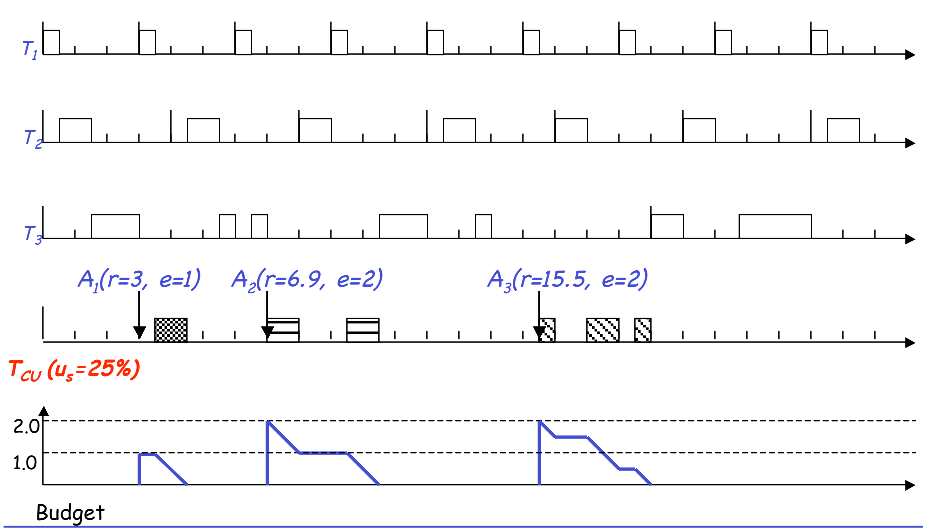
  R3 At the deadline d of the server,

  (a) if the server is backlogged, set $e_s := e$ and $d := d + e/u_s$

  (b) if the server is idle, do nothing.

Constant Utilization Server: Example

$T_1 = (3, 0.5)$ $T_2 = (4, 1.0)$ $T_3 = (19, 4.5)$

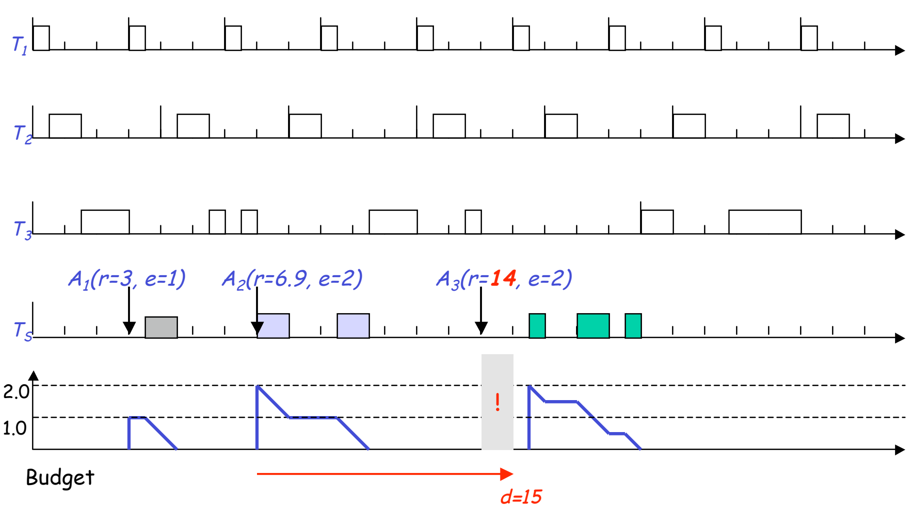


What about Unknown Execution Times?

- Assumption for constant utilization server: execution times of aperiodic jobs are known upon arrival.
  - ⇒ Restrictive.
- Possible solution: Assign fixed bandwidth to server:
  - fixed budget e_s
  - fixed period e_s/u_s
- Upon job completion of job with execution time $e < e_s$, reduce current deadline of server by $(e_s-e)/u_s$ before replenishing again.
- For execution time $e > e_s$, use more than one server period.

Problems with Constant Utilization Server: Unused Capacity

$T_1 = (3, 0.5)$ $T_2 = (4, 1.0)$ $T_3 = (19, 4.5)$



Total Bandwidth Server

- Allow server to use background time.
- Consumption rule:
  - A server consumes its budget only when it executes.
- Replenishment rules:

  R1: Initially, set e_s := 0 and d := 0.

  R2: When an aperiodic job with execution time e arrives at time t to an <u>empty</u> aperiodic job queue, set $d := max(d,t) + e/u_s$, and $e_s := e$.

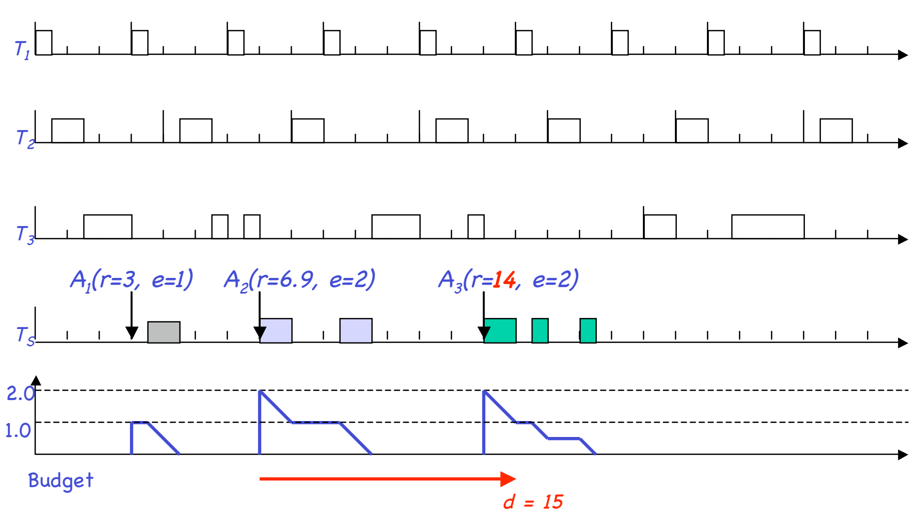
  R3: Upon completion of the current aperiodic job, remove job from queue.

  (a) if the server is backlogged, set $d := d + e/u_s$ and $e_s := e$

  (b) if the server is idle, do nothing.

Unused Capacity Eliminated with Total Bandwidth Server

$T_1 = (3, 0.5)$ $T_2 = (4, 1.0)$ $T_3 = (19, 4.5)$



Correctness of Total Bandwidth Server

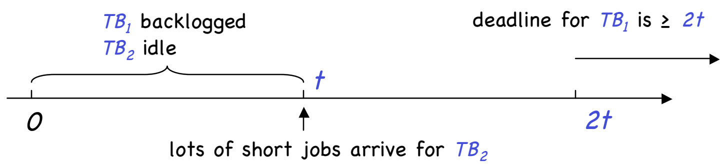
- Constant Utilization Server is correct.
- How does Total Bandwidth Server affect periodic tasks differently?
- Only interesting case:
  - Budget of Total Bandwidth Server replenished at time t before its deadline.
  - New deadline is $d' = d + e/u_s$.
- How does this affect the execution of periodic tasks?
  - Case 1: Current periodic job $J_{i,c}$ has deadline before d' ⇒ execution of periodic job is not affected.
  - Case 2: Current periodic job $J_{i,c}$ has deadline after d'
    - Case 2.1: Current periodic job $J_{i,c}$ is ready before time t ⇒ execution time demanded by Total Bandwidth Server from $r_{i,k}$ to d' is same as for Constant Utilization Server.
    - Case 2.2: Current periodic job $J_{i,c}$ is ready after time t ⇒ execution time demanded by Total Bandwidth Server from $r_{i,k}$ to d' is less than that of Constant Utilization Server.

Non-Preemptable Portions

- Non-preemptable portions either reduce schedulable utilization or introduce tardiness.
- **Definitions:**
  - $b_{max}(np)$ is maximum execution time of non-preemptable portions of periodic tasks and jobs executed by servers.
  - effective execution time of job executed by server: ratio of job execution time and server size.
  - D_{min} is minimum of all relative deadlines of periodic tasks and effective execution times of jobs executed by all servers in the system.
- **Corollary:** When a system of periodic tasks is scheduled with one or more total bandwidth and constant utilization server on the EDF basis, every periodic task and every server meets its deadline if the sum of the total density of the periodic tasks and the total size of all servers is no greater than $1-b_{max}(np)/D_{min}$.

Fairness and Starvation

- Total Bandwidth Server is not fair
- Example: TB_1 and TB_2 each of size 0.5
  - If both servers never idle, service is approximately equally shared among servers.
  - With idling servers, this is not always the case.



  - Processor time is allocated fairly during $(0, 2t)$, but not during $(t, 2t)$

Fairness & Starvation

Definition (**Fairness**):

$w_i(t_1, t_2)$ = total attained processor time for Server i during time interval (t_1, t_2).

$w_i(t_1, t_2)/u_i$ = normalized service.

Scheduler is fair during interval (t_1, t_2) if normalized service attained by all servers does not differ by more than a fairness_threshold *FR*.

- Ideally, *FR* is zero:

$$\frac{w_i(t_1,t_2)}{w_j(t_1,t_2)} = \frac{u_i}{u_j} \Longleftrightarrow w_i(t_1,t_2) = u_i(t_2 - t_1)$$

Eliminating Starvation

- Problem with Total Bandwidth server: When processing time available, allows to indefinitely put deadlines into the future.
- Constant Utilization server keeps deadlines "close":

  $d - t <= e_{i,max} / u_i$, where $e_{i,max}$ is max. execution time of jobs served by Server i.

- Replenishment rules for starvation-free Constant Utilization / Background server:
  - R1 - R3 : Same as Constant Utilization server.
  - R4 : Whenever busy interval ends, replenish budget of all backlogged servers.
- Note: Background time is not distributed to backlogged servers according to their size => starvation is eliminated, but not unfairness.

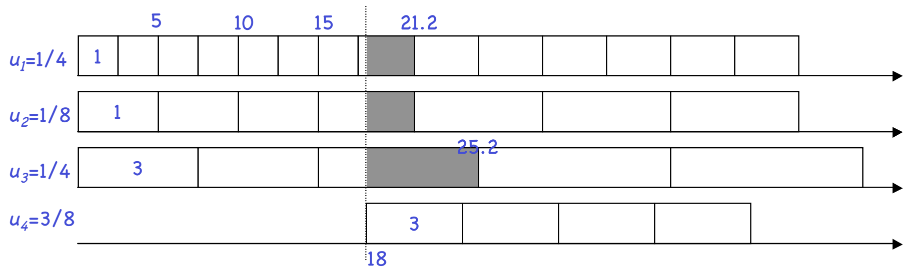
Processor Sharing

- Process a single atomic unit at a time per backlogged queue in round-robin fashion.
- If there are N backlogged servers, then each server receives exactly 1/N of the available capacity.
- Some terms:
  - $R(t)$ = number of rounds in the PS service discipline that have occurred up to time t
  - $N(t)$ = number of nonempty queues at time t
  - P_{ij} = job length of Job i in Server j
  - r_{ij} = arrival time of Job i in Server j
  - S_{ij} = value of $R(t)$ when Job i in Server j arrives
  - F_{ij} = value of $R(t)$ when Job i in Server j terminates
- Think of $R(t)$ as virtual time, which records the rate of service seen by job at head of queue in Server.

$$R'(t) = 1 / \max[1, N(t)]$$

Preemptive Weighted Fair-Queuing Algorithm

- Replenishment rules similar to Total Bandwidth server; except for computation of deadline at each replenishment time.
- pWFQ algorithm bounds fairness.
- Replenishment rules of pWFQ server make it emulate GPS server with same size.



- **Virtual Time**: Enqueue jobs in order of Finish Number: number of rounds for GPS server to exhaust budgets.

Rules for pWFQ

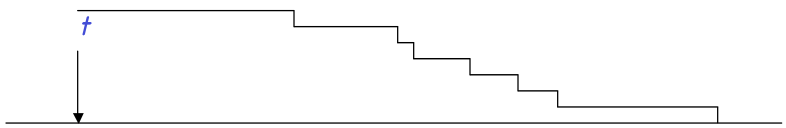
- *Scheduling Rule:* Assign priorities in order of increasing finish number.
- *Consumption Rule:* pWFQ server consumes budget only when it executes.
- *Initialization Rules:*
  - I1: When system is idle, $FN = 0$, $U_b = 0$, $t_{-1} = 0$. Budgets of all servers are zero.
  - I2: When first job arrives at time t with execution time e at some server FQ_k when system is idle:
    - (a) $t_{-1} := t$, and $U_b := U_b + u_k$, and
    - (b) set budget e_k of FQ_k to e and finish number $fn_k := e/u_k$.
- *Rules for updating Finish Times during System Busy Interval:*
  - R1: When job arrives at queue FQ_k while FQ_k is idle
    - (a) increment system finish number $FN := FN + (t-t_{-1})/U_b$
    - (b) $t_{-1} := t$, and $U_b := U_b + u_k$, and
    - (c) set budget e_k of FQ_k to e and its finish number $fn_k := FN + e/u_k$, enqueue server
  - R2: Whenever FQ_k completes job
    - (a) if server remains backlogged, set server budget e_k to e and increment its finish number: $fn_k := fn_k + e/u_k$.
    - (b) if server becomes idle, update U_b and FN as follows:

      $FN := FN + (t-t_{-1})/U_b$, $t_{-1} := t$, and $U_b := U_b - u_k$.

Scheduling Sporadic Tasks in EDF Systems

- Total density of periodic tasks is Δ .
- As long as total density of sporadic jobs does not exceed $1-\Delta$, all deadlines can be maintained by EDF.

- Acceptance Test:
  - Maintain list of time intervals and their densities:



  - Acceptance test then adds new density: