Servers for aperiodic tasks

Principles for server design
Explaining servers through example

Aperiodic tasks

- How do we deal with aperiodic tasks?
- Why?
  - Critical, but occasional, operations that require immediate attention
  - Occasional events that need to completed soon, but periodic tasks are more important and need to meet their deadlines; aperiodic tasks do not have hard deadlines
  - Examples: system mode changes, activity logs, garbage collection

Mixing periodic and aperiodic tasks

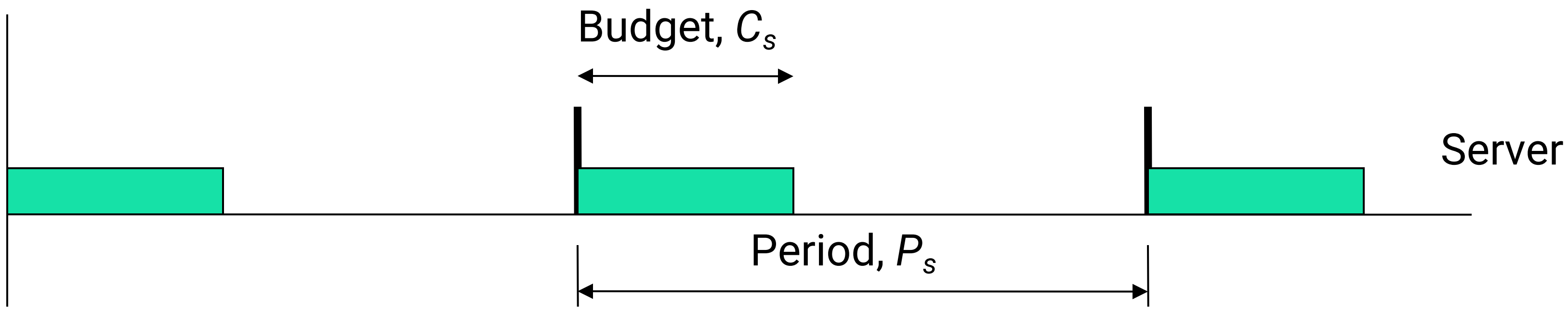
- **Question:** how to execute aperiodic tasks without violating schedulability guarantees given to periodic tasks?
- And in a static-priority environment

Mixing periodic and aperiodic tasks

- **Question:** how to execute aperiodic tasks without violating schedulability guarantees given to periodic tasks?
- And in a static-priority environment
- Easy approach: schedule aperiodic tasks at the lowest priority level
    - In the “background”
    - **Problem:** Extremely poor performance for aperiodic tasks; periodic tasks can be delayed as long as they do not miss their deadlines

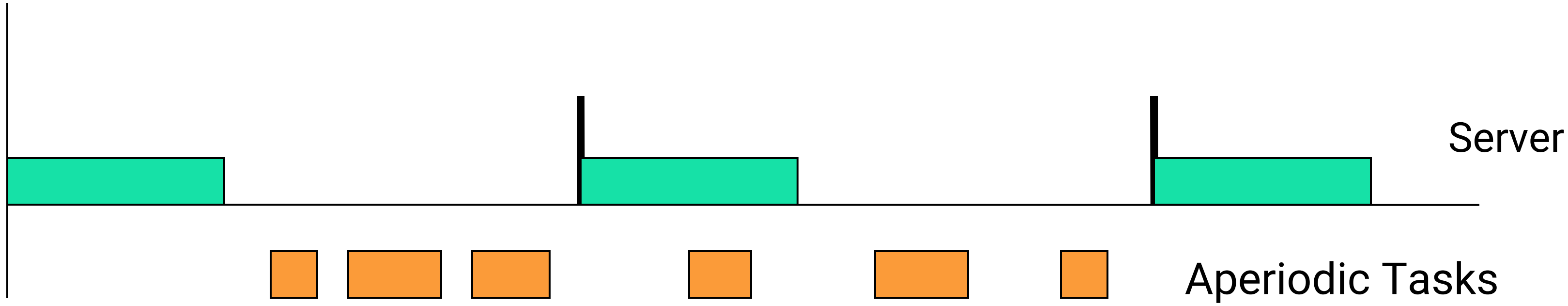
Server-based systems

- Periodically invoke a service task ("server") to execute aperiodic tasks
- The server is modeled as a periodic task and can be included in schedulability analysis
- Allocate the server a computation budget C_s and a period P_s
- The server can serve aperiodic tasks until the budget expires; the budget can be replenished every period
- **Many choices:** Servers have different flavors depending on the details of when they are invoked, what priority they have, and when budgets are replenished



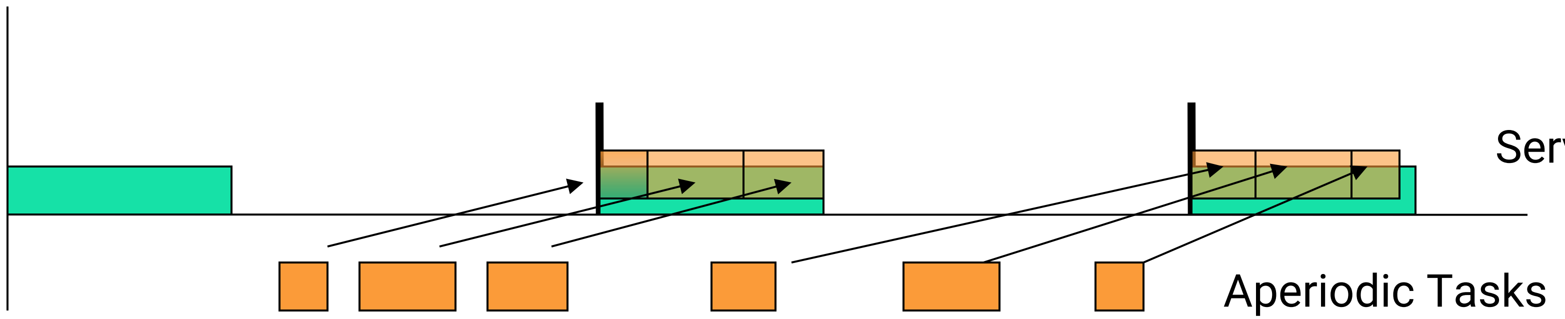
Server-based systems

- Periodically invoke a service task ("server") to execute aperiodic tasks
- The server is modeled as a periodic task and can be included in schedulability analysis
- Allocate the server a computation budget C_s and a period P_s
- The server can serve aperiodic tasks until the budget expires; the budget can be replenished every period
- **Many choices:** Servers have different flavors depending on the details of when they are invoked, what priority they have, and when budgets are replenished



Server-based systems

- Periodically invoke a service task (“server”) to execute aperiodic tasks
- The server is modeled as a periodic task and can be included in schedulability analysis
- Allocate the server a computation budget C_s and a period P_s
- The server can serve aperiodic tasks until the budget expires; the budget can be replenished every period
- **Many choices:** Servers have different flavors depending on the details of when they are invoked, what priority they have, and when budgets are replenished



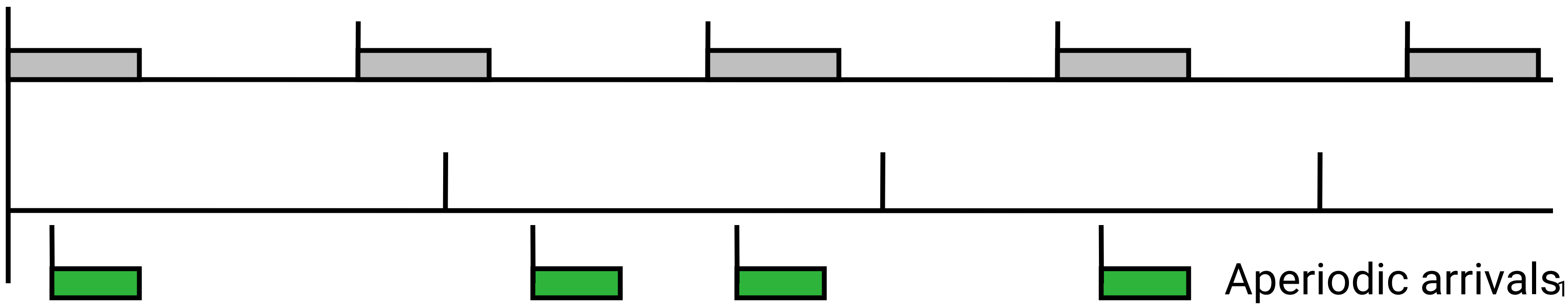
Polling Server

- Runs as a periodic task (priority set according to policy)
- Aperiodic arrivals are queued until the server task is invoked
- A server is said to be **active** if its budget is > 0 and is either:
  - Running: it is the highest priority task and is servicing an aperiodic request, or
  - In ready queue: has been preempted by a higher priority task while servicing an aperiodic request or arrived while a higher priority task is running
- otherwise it is **idle** (in the waiting queue)
- When the server is invoked it serves the aperiodic queue until it is empty or until the budget expires then *suspends itself*
  - **Suspends itself**: removed from ready queue and placed in waiting queue → becomes *idle*
- If server is idle, then it is returned to ready queue in the event of its next arrival (period, which is the next replenishment time) and then a scheduling decision is made

Server is treated as a regular periodic task in schedulability analysis

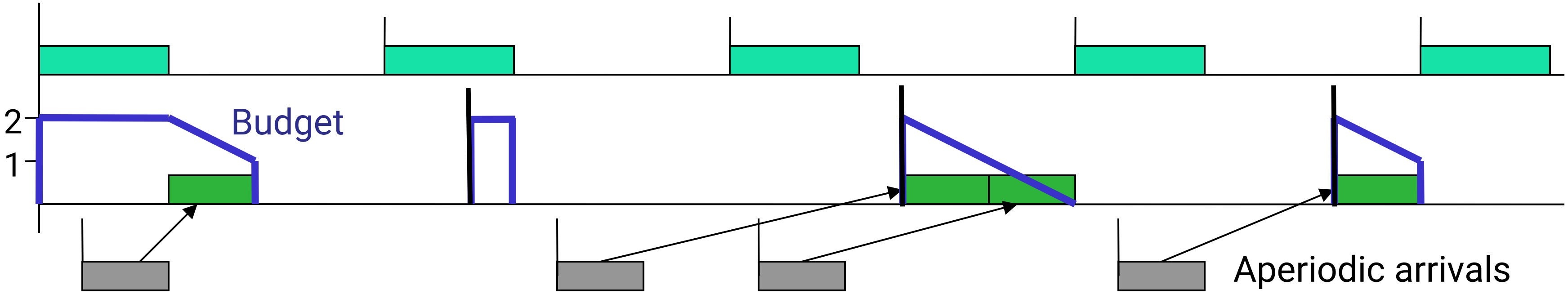
Example of a Polling Server

- Polling server:
  - Period $P_s = 5$
  - Budget $C_s = 2$
- Periodic task
  - $P = 4$
  - $C = 1.5$
- All aperiodic arrivals have $C=1$



Example of a Polling Server

- Polling server:
  - Period P_s = 5
  - Budget C_s = 2
- Periodic task
  - P = 4
  - C = 1.5
- All aperiodic arrivals have $C=1$



Polling server

Server is just another periodic task → **Under RM:** periodic taskset is feasible if

$$\sum_{i=1}^{n} \frac{C_i}{P_i} + \frac{C_s}{P_s} \leqslant (n+1)\left[2^{1/(n+1)} - 1\right]$$

With more careful analysis and assuming T_s is the *highest* priority task

$$U_p \leqslant n\left[\left(\frac{2}{U_s+1}\right)^{1/n} - 1\right]$$

$$U_p = \sum_{i=1}^{n} \frac{C_i}{P_i} \qquad U_s = \frac{C_s}{P_s}$$

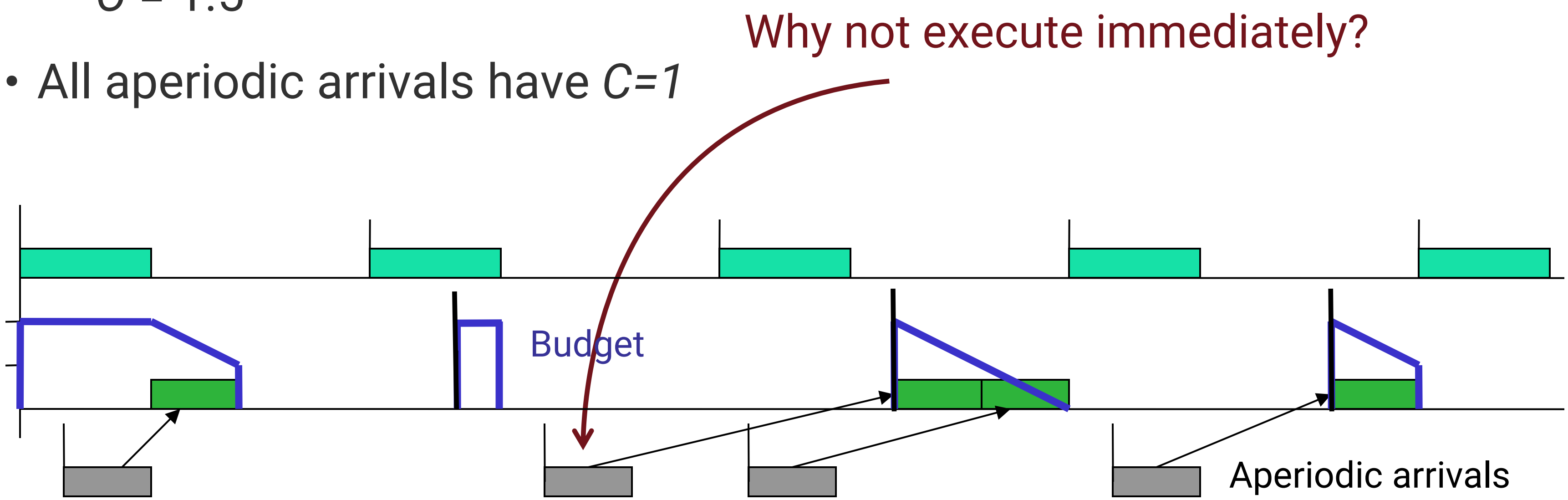
Replicate LL bound analysis for RM

How to dimension a polling server?

- How do we set C_s and P_s in such a way to guarantee feasibility?
- Hyperbolic bound extends to $\prod_{i=1}^{n}(1 + U_i) \leq \frac{2}{U_s+1}$
- Let $Y = \prod_{i=1}^{n}(1 + U_i)$
- Then the maximum possible server utilization U_s for feasibility is $U_s^{\max} := \frac{2-Y}{Y}$
- What period (priority in RM) to assign to server?
- For any assigned period P_s, $C_s = P_s \times U_s^{\max}$

Example of a Polling Server

- Polling server:
  - Period P_s = 5
  - Budget C_s = 2
- Periodic task
  - P = 4
  - C = 1.5
- All aperiodic arrivals have $C=1$

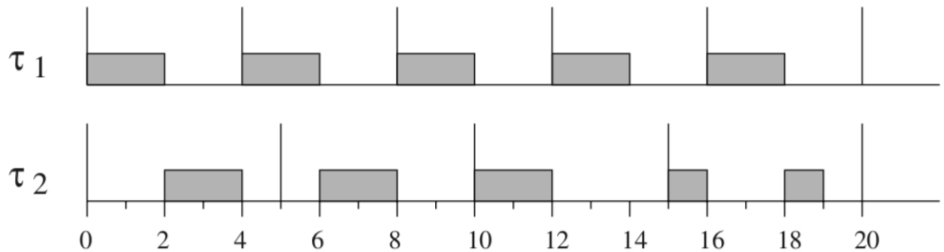


Deferrable server

- Unlike polling server, *preserves its budget* if no aperiodic requests are pending upon its invocation
- Schedules aperiodic jobs that arrive later during its period until budget exhausted
  - If DS is idle and still has budget when an aperiodic request arrives, DS is removed from waiting queue and is inserted into ready queue
- **Replenishment rule:** like polling server, budget set to C_s at its next arrival

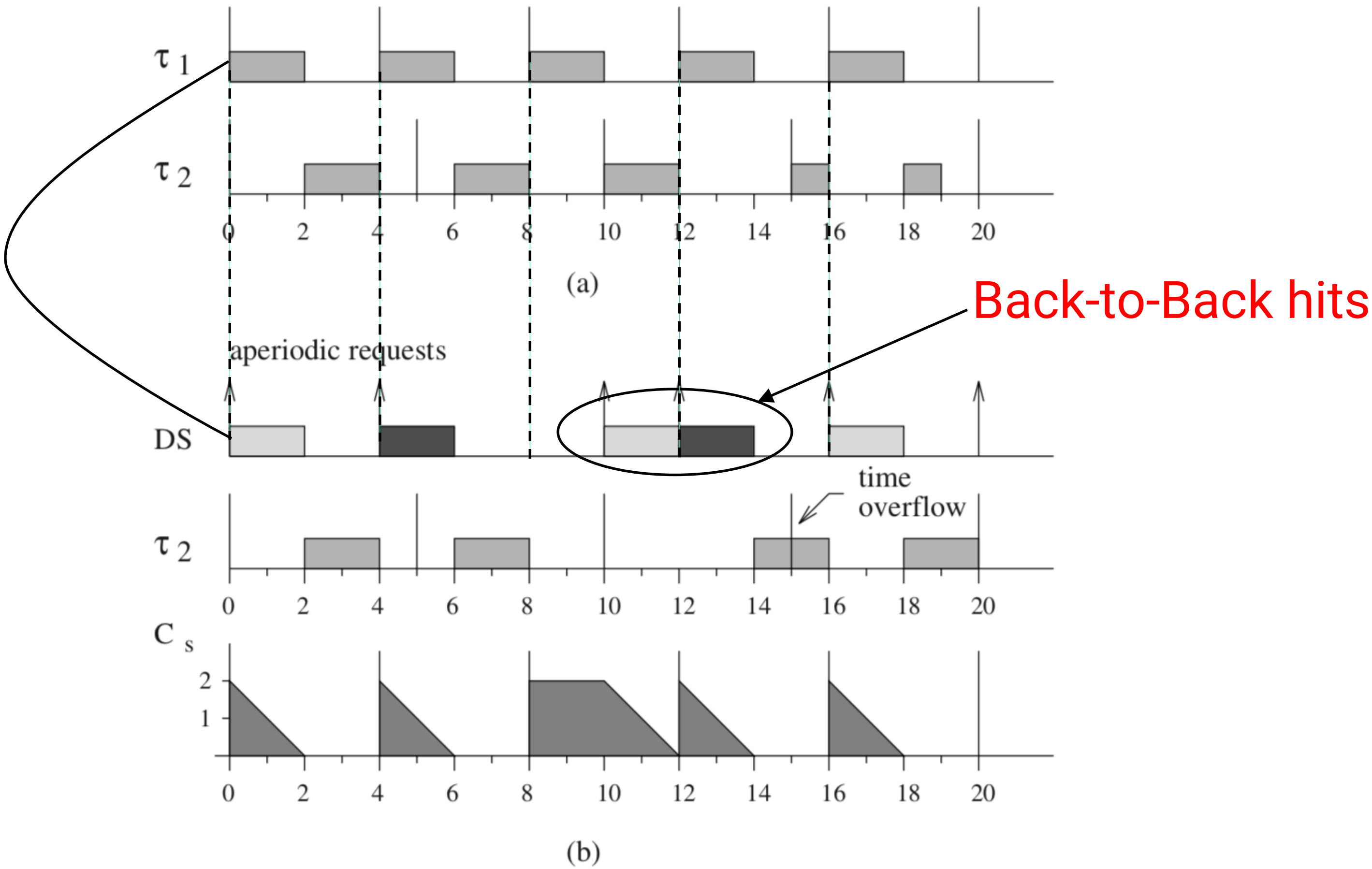
DS cannot be treated as a periodic task in schedulability analysis. Why?

Deferrable server: ***Not*** a regular periodic task



(a)

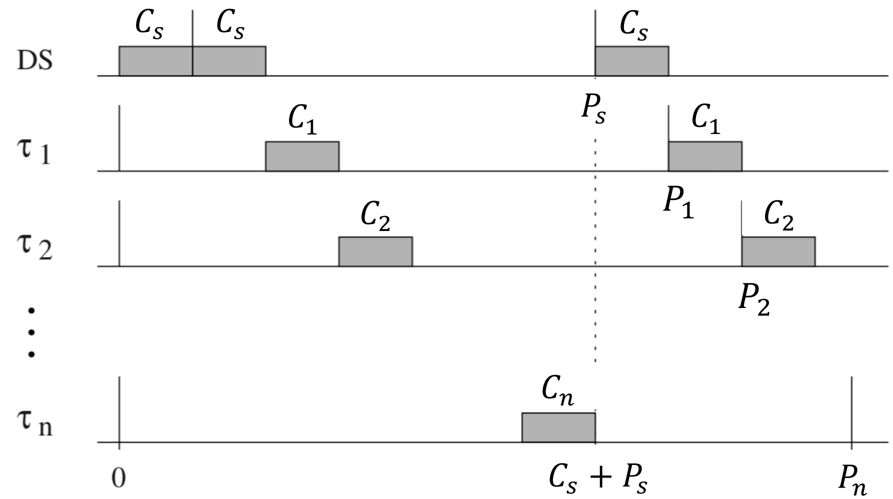
Deferrable server: *Not* a regular periodic task



Invasive nature because it preserves capacity
Execution might be ***deferred*** till late in period → not how periodic tasks behave!
Not present in PS → if no period task available then PS suspends itself until next period

Deferrable server: Schedulable utilization

- In general, complex schedulable utilization expressions and conditions
- Example:
  - Server is highest priority task
  - $P_s < P_1 < P_2 < \cdots < P_n < 2P_s$
  - $P_n > P_s + C_s$
  - $P_i = D_i$ (implicit-deadline)
  - rate-monotonic scheduling



Worst case conditions for DS

$$\text{Schedulable utilization: } U_p \leqslant n\left[\left(\frac{U_s+2}{2U_s+1}\right)^{1/n} - 1\right]$$

Sporadic Server

- Fixes the invasive nature of DS
- More complex consumption and replenishment rules ensure that a sporadic server with period P_s and budget C_s never demands more processor time than a periodic task with the same parameters
- Allows server to be treated as a periodic task in schedulability analysis!

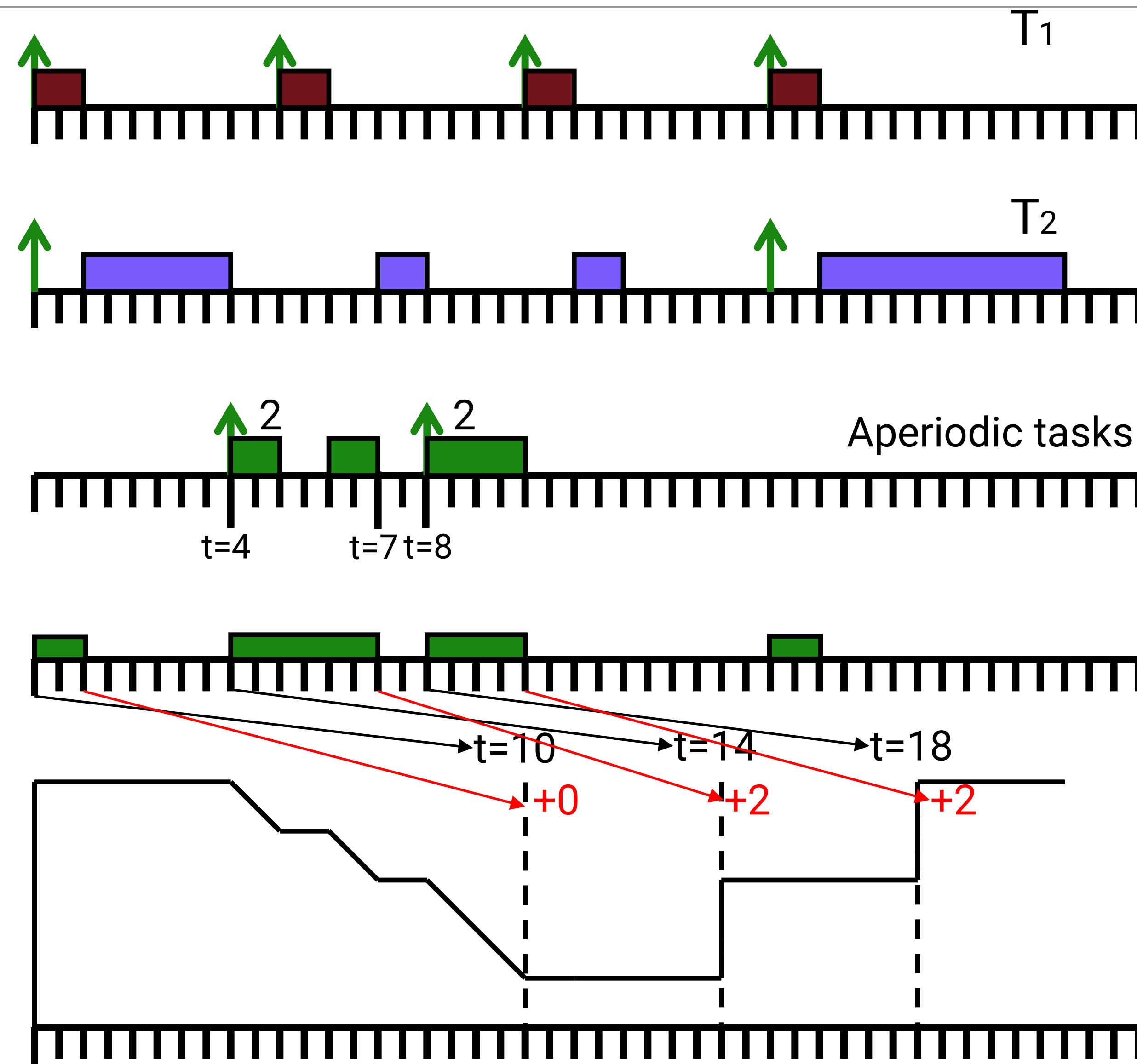
Sporadic Server

- Every time the server becomes active, say at t_A, it sets replenishment time one period into the future, $t_A + P_s$ (but does not decide on replenishment amount).
- When the server becomes idle, say at t_I , set replenishment amount to capacity consumed in $[t_A, t_I]$
- If an aperiodic request arrives when budget is zero, server does **not** become active and aperiodic request will have to wait till the next replenishment epoch

Sporadic Server: Example

Two periodic tasks
T_1: (P_1=5, C_1=1)
T_2: (P_2=15, C_2=5)

Sporadic Server
P_s=10, C_s=5

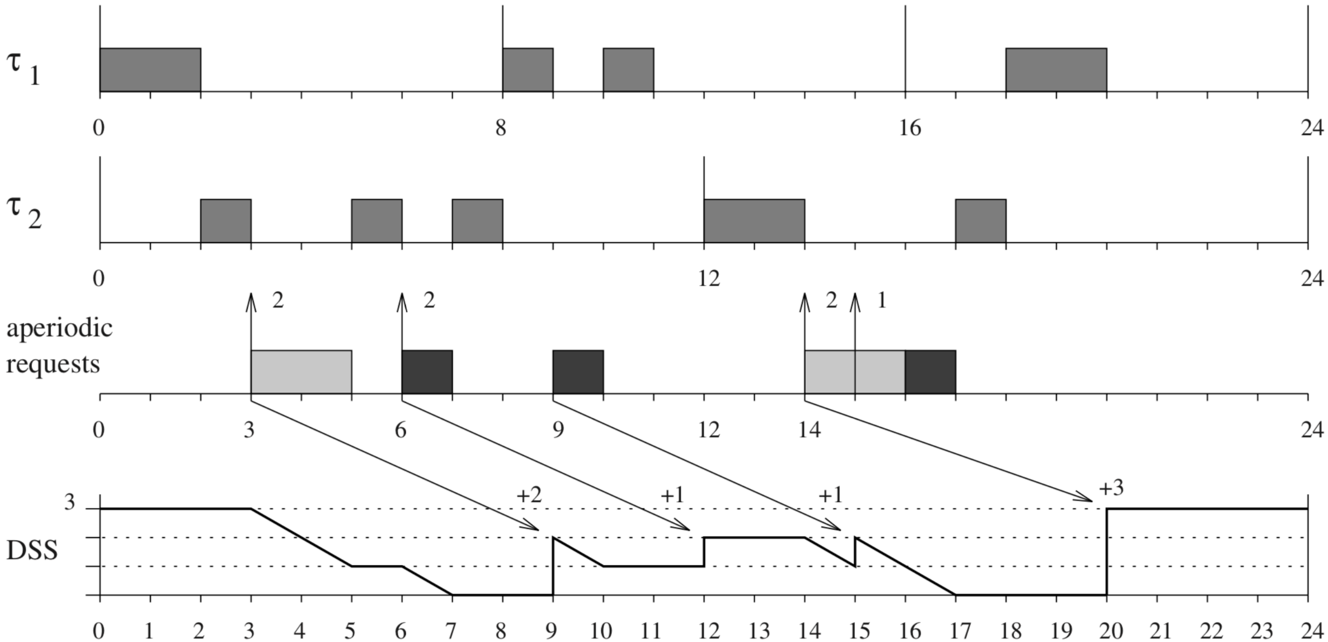


Dynamic priority aperiodic task servers

Dynamic sporadic server

- In general, dynamic servers are based on assigning the server suitable time-varying deadlines so that the processor is fully utilized.
- When the server is created its capacity C_s is initialized.
- When server is idle and there is a pending aperiodic task and $C_s > 0$, server becomes "active"
    - Set a replenishment time one period into the future (deadline)
    - At time t_A, $d_s := t_A + P_s$ becomes the *absolute* deadline of the server task
- When the server becomes inactive set the replenishment amount as the capacity consumed

Example for dynamic sporadic server



Two periodic tasks

T1: (*P1=8, C1=2*)

T2: (*P2=12, C2=3*)

Dynamic sporadic server

Ps=6

Cs=3

Ties among tasks are always resolved in favor of the *server:*

- Increases responsiveness of aperiodic requests without jeopardizing feasibility of hard tasks

- When the server is created its capacity C_s is initialized.
- When there is a pending aperiodic task and $C_s > 0$, server becomes "active"
  - Set a replenishment time one period into the future (absolute deadline)
- When the server becomes inactive set the replenishment amount as the capacity consumed

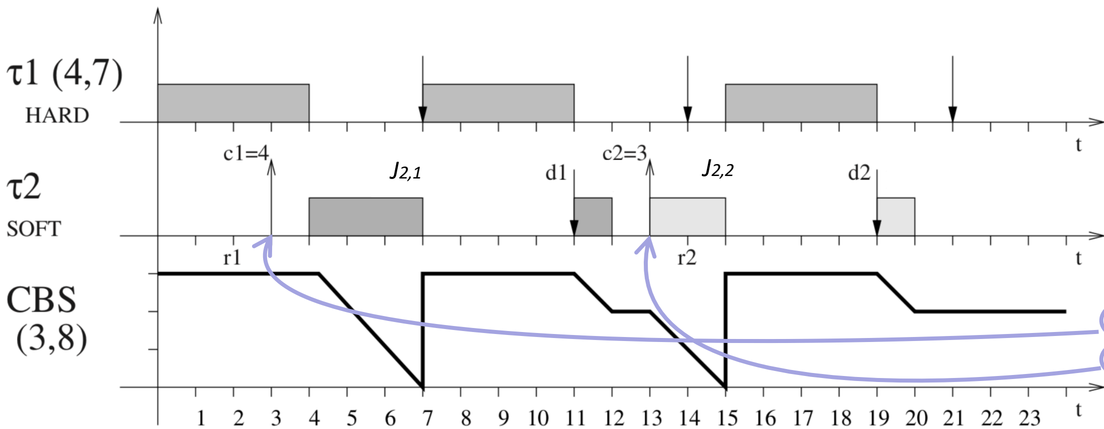
Dynamic sporadic server: Problems?

- When server period is long, execution of aperiodic requests is significantly delayed
  - Longer period ⇒ server scheduled with farther absolute deadlines (policy is EDF)
- **Possible solution**: Reduce period of server while keeping U_S constant
  - To keep U_S constant while decreasing period, capacity should decrease
  - **Consequences**: Excessive run-time overhead
    - More frequent replenishments
    - Increased context switches with periodic tasks
- **Another approach**: change the way server deadlines are assigned
  - Assign server earlier deadlines in such a way that U_S is never exceeded (to preserve temporal isolation of hard tasks and not interfere with them)

The Constant Bandwidth Server (CBS)

- Server has a maximum budget Q_s and a period P_s
- The server is said to be **active** if jobs are pending, otherwise it is **idle** [much simpler definitions]
- When an aperiodic job arrives it inherits the server deadline, d_s
- When an aperiodic job executes the server budget is decreased by the same amount
- **When the budget is zero it is recharged to Q_s and deadline d_s is increased by P_s**
- **When an aperiodic request arrives at time t and the server is idle:**
  - **If remaining budget $C_s(t) > (d_s - t)U_s$, the deadline is advanced to $t + P_s$**
- The main advantage of the CBS is that it can deal with overruns -- when jobs exceed their estimated computation times

Example for CBS



Maximum budget

One hard periodic task
T_1: (P_1=7, C_1=4)

One soft task served by a CBS
P_s=8, Q_s=3

$J_{2,1}$ has computation time *4* and arrives at *t=3*
$J_{2,2}$ has computation time *3* and arrives at *t=13*

Current budget

The first instance of Task 2 ($J_{2,1}$) arrives at *t=3*. At *t*=3, d_s=8 and C_s=3.
C_s = 3 > $(d_s\text{-}t)U_s$ = 15/8. Therefore the server deadline is set to 3+8=11.

At *t=7*, the budget is exhausted so the new deadline is set to 11+8=19 and the budget replenished. At *t*=12, $J_{2,1}$ is complete.

At *t=13*, $J_{2,2}$ is released. C_s = 2 < $(d_s\text{-}t)U_s$ = 9/4. $J_{2,2}$ starts executing with deadline 19.

At *t=15*, the budget is exhausted. The new deadline of 19+8=27 is assigned to the server and the budget is reset to 3. $J_{2,2}$ completes at *t*=20.

Principles of server design

- It is simple enough to represent the servers as periodic tasks
- So, why so many rules?
  - We want to reduce the response times for aperiodic tasks
  - Avoid the problems with the polling server: retain unused budget
  - If we want to retain the budget
    - When does it expire?
      - If a server has budget 2 and deadline 5, it cannot have a budget of 2 when t=4; there is only one unit of computation remaining but a budget of 2
      - We can not make the operating system do too much work. It only schedules by priority or deadline and does not verify if the deadline has expired or not.

Principles of server design

- It is simple enough to represent the servers as periodic tasks
- So, why so many rules?
    - We want to reduce the response times for aperiodic tasks
    - Avoid the problems with the polling server: retain unused budget
    - If we want to retain the budget
        - When does it expire?
        - When does it increase?
            - If we consume a portion of the budget, when do we restore it?
        - We cannot allow the server to use more than the allotted fraction of the processor: if the server has a utilization of 0.4, it can not use more than 2 units of time every 5 units (or 4 in every 10, 8 in every 20, ...)
        - How can we implement these easily? [The polling server is easy to implement.]

Summarizing aperiodic servers

- Quite a few aperiodic server mechanisms
- The difference between these schemes concerns performance and complexity (implementation, memory etc.)
- CBS is used most often in the dynamic priority case: reasonable performance and easy implementation
    - Officially implemented in the Linux kernel since 3.14

Lecture summary

- Aperiodic task servers
  - Static priorities
    - Polling Server
    - Sporadic Server
  - Dynamic priorities
    - Dynamic Sporadic Server
    - Constant Bandwidth Server