09_Overload Analysis

- Definitions and metrics of overloading situations.
- Case 1: Handling Transient Overloads due to aperiodic jobs
  - Overloads due to having more jobs than expected
- Impossibility results and how to assess the performance of an online algorithm in overload conditions.
- Approaches for handling such overloads and the RED algorithm.

- Case 2: Handling Transient Overloads due to Overruns
  - Transient overloads due to longer WCET
- Solution through resource reservation, e.g., via CBSs.
- Schedulability Analysis.
  - How to measure resource availability and how to check if the overloads can be accommodated/handled?
- How to adjust the servers' parameters.

1. Introduction

1.1. Reasons for Overloading

1. The system designer was **too optimistic.**
2. The designer was pragmatic but the **problem changed drastically.**
3. System **malfunctions and exceptions** occurred.

1.2. Wasy to measer overloads

- Intensity of Soft aperiodic tasks: $\rho = \lambda \, \bar{C}$

Load (traffic intensity) — Avg. job arrival rate — Avg. service time

- Utilization of Hard periodic tasks (D_i=T_i):

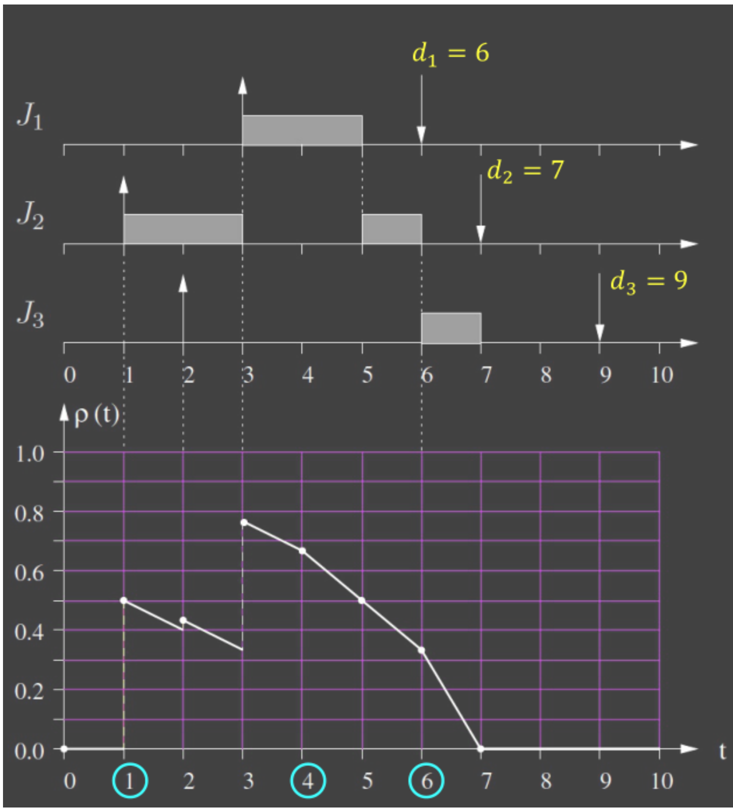
$$\rho = U = \sum_{i=1}^{n} \frac{C_i}{T_i}$$

Instantaneous Load for Hard Periodic Tasks

Compute the load in all intervals from current time t and each deadline (d_n) of all active jobs.

Partial load in $[t, d_n]$ due to first n jobs

$$\rho_n(t) = \sum_{d_k \le d_n} \frac{c_k(t)}{d_n - t} \qquad \rho(t) = \max_n \rho_n(t)$$



$$\rho_1(1) = 0$$

$$\rho_2(1) = \frac{2+1}{7-1} = \frac{3}{6}$$

$$\rho_3(1) = \frac{2+1}{9-1} = \frac{3}{8}$$

$$\rho_1(4) = \frac{1}{6-4} = \frac{1}{2}$$

$$\rho_2(4) = \frac{1+1}{7-4} = \frac{2}{3}$$

$$\rho_3(4) = \frac{1+1+1}{9-4} = \frac{3}{5}$$

$$\rho_1(6) - N/A$$

$$\rho_2(6) = \frac{0}{7-6} = 0$$

$$\rho_3(6) = \frac{1}{9-6} = \frac{1}{3} = 0.33$$

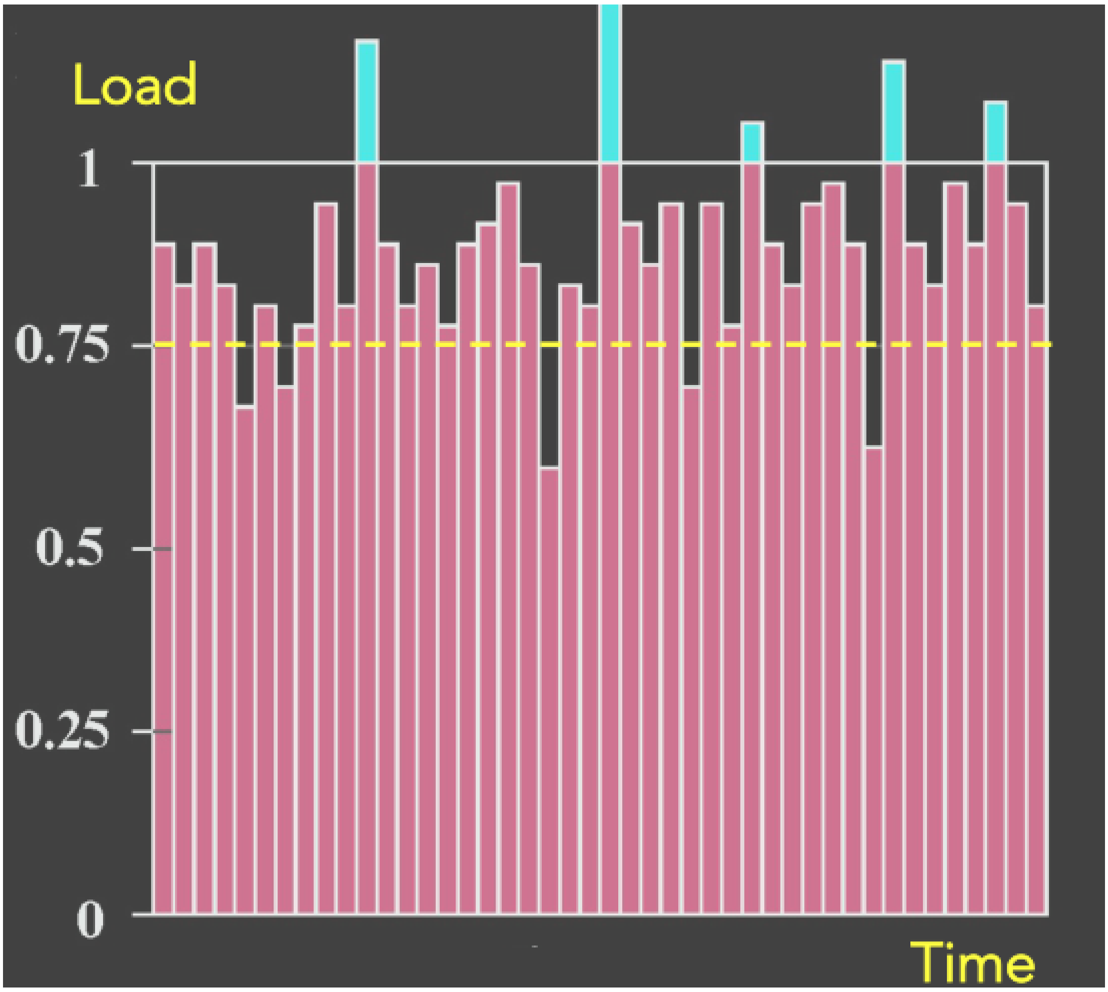
1.3. Overload Types

System designed under **worst-case assumptions**: Overloading is **avoided**

System designed under **average-case assumptions**: Overloading $\rho(t) > 1$

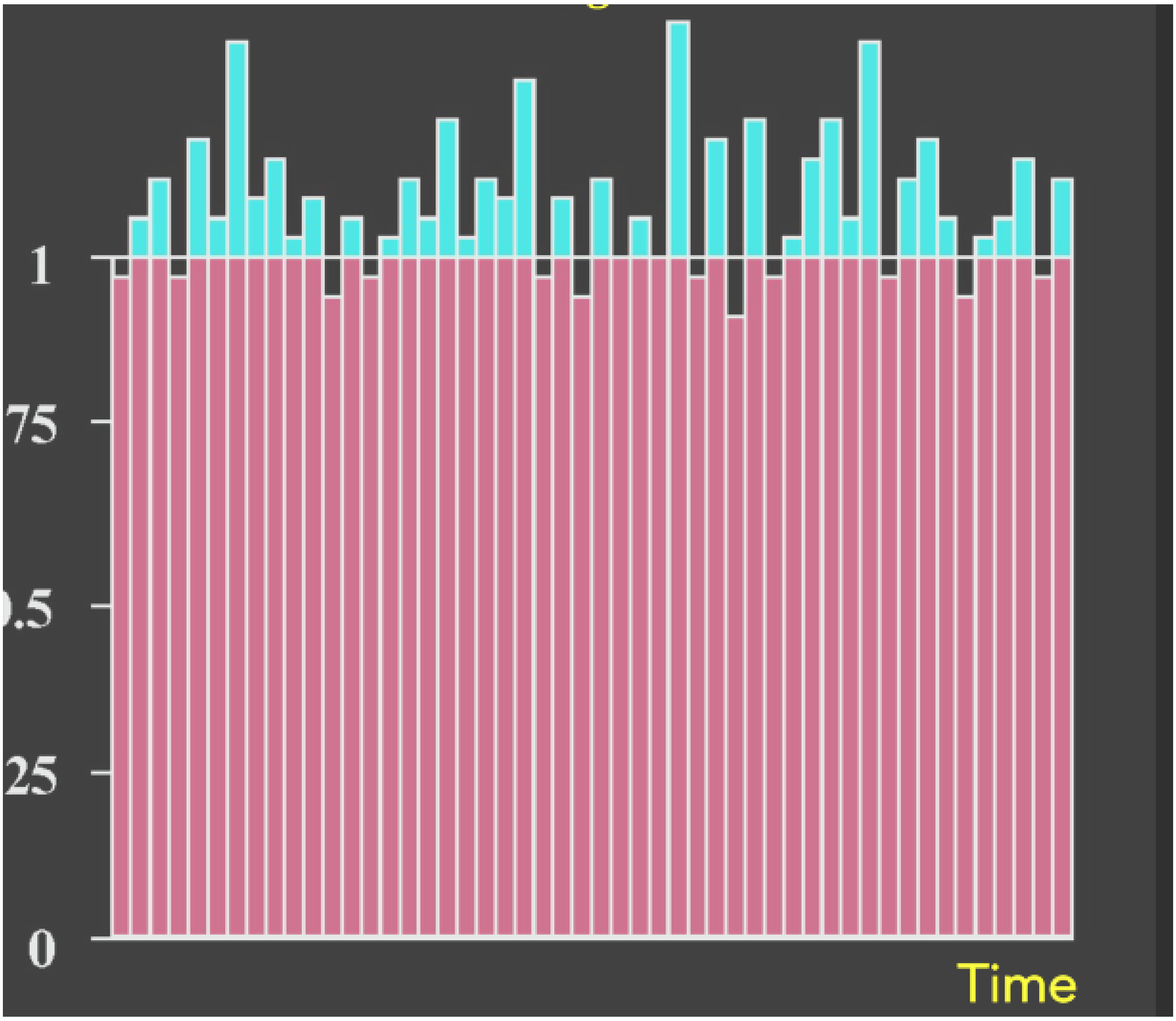
Transient Overload

$$\rho_{max} > 1 \quad \rho_{avg} < 1$$



Permanent Overload

$$\rho_{avg} > 1$$



1.4. Predictability vs Efficiency

Designing with **pessimistic assumptions**:

- **Highly predictable** but less efficient systems

Designing with **optimistic assumptions**:

- **Highly efficient** but less predictable systems.

1.5. Classes of Overloads

- Transient overloads due to aperiodic jobs
- Transient overloads due to task overruns.
- Permanent overloads in periodic task systems.

2. Handling Aperiodic Overloads

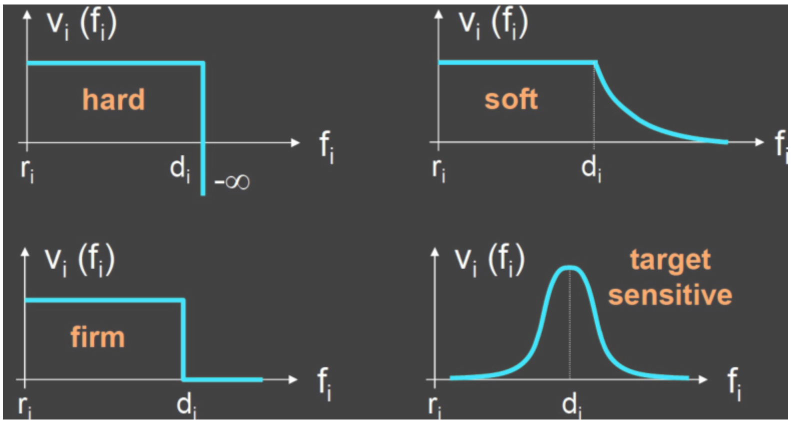
If we have more tasks than we can serve… there is no solution

So, we need to reject some tasks, the importnat thing is **How to decide which ones**

We should use **Value-based Scheduling**, in order to maximize the total system utility

e.g.

$v_i = V_i$	arbitrary constant
$v_i = C_i$	computation time
$v_i = V_i / C_i$	value density



2.1. Performance Evaluation

Aggregate Utility

$$\Gamma_{A}(\mathrm{T}) = \sum_{i=1}^{n} v_i\left(f_i\right)$$

Optimal Aggregate Utility

$$\Gamma^*(\ \mathrm{T}) = \max_{\mathrm{A}} \Gamma_{\mathrm{A}}(\mathrm{T})$$

It is the optimal value among all algorithm on the given set T

Impossibility Result Theorem 1

There is **no online algorithm** that can achieve, in a **guaranteed fashion**, the optimal solution $\mathit{\Gamma}^*$

- $\mathit{\Gamma}^A$ can only be maximized if we know the future.

Competitive Factor

An algorithm A has a **<u>competitive factor</u>** $\phi(A)$:, if it is **guaranteed** that, for any task set, it achieves:

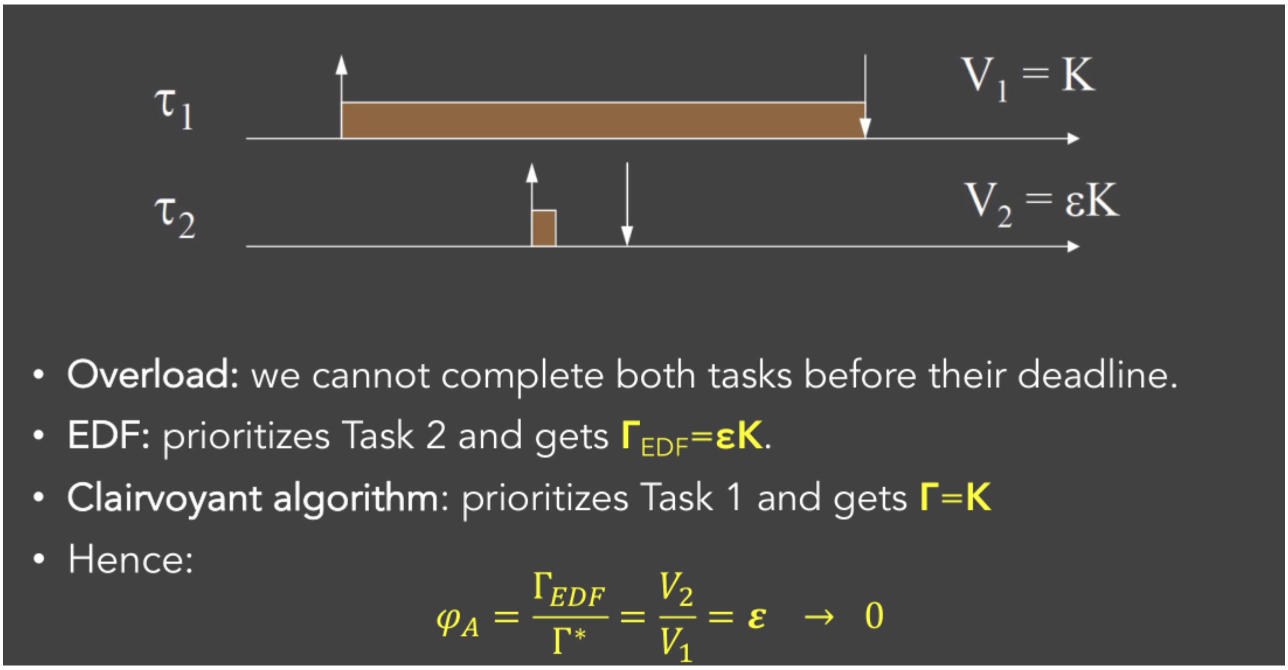
$$\Gamma_A \geq \varphi_A \Gamma^*$$

$$\varphi_A = \min_{\mathbb{T}} \frac{\Gamma_A(T)}{\Gamma^*(T)} \quad \varphi_A \in [0, 1]$$

For Γ^*, different T may has different different Algorithm, but they are all compared to algorithm A

Competitive Factor of EDF

$$\phi_{EDF} = 0$$



Impossibility results Theorem 2

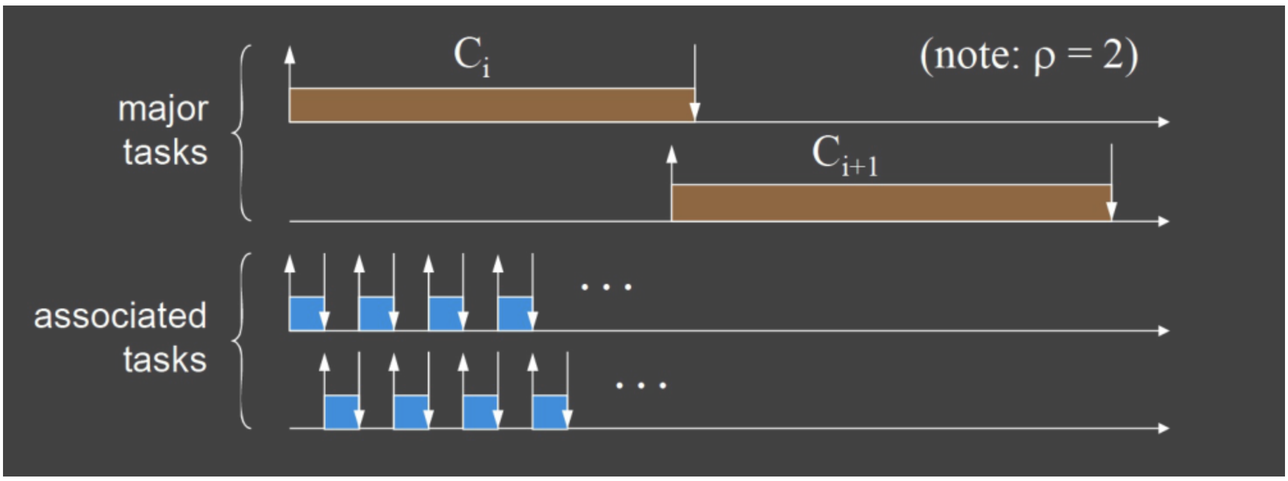
If $\rho \geq 2$ and $V_i = C_i, \quad \forall i$, then there **does not exist an online a lgorithm** that can achieve a competitive factor **greater than 0.25**

__**Proof**__: Adversary argument

assume that the algorithm “plays” against a sophisticated **adversary (clairvoyant scheduler)** which **creates the worst-case conditions** in terms of tasks sequence.

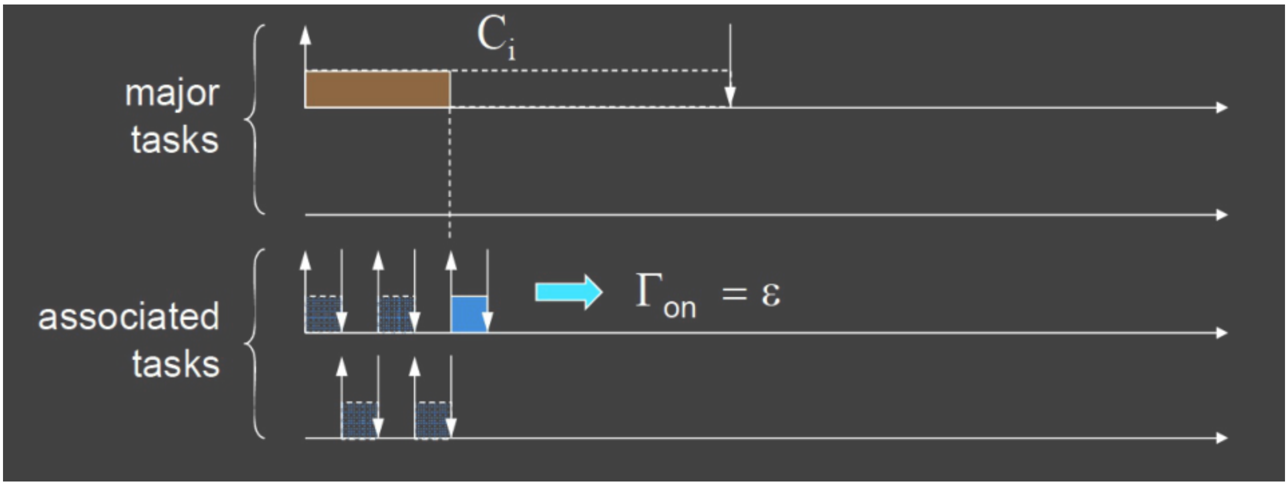
The adversary generates two types of tasks:

- Major taskss: $C_i = D_i$ and $r_{i+1} = d_i - \epsilon$
- Associated tasks: $C_i = \epsilon$ and $r_{i+1} = d_i$

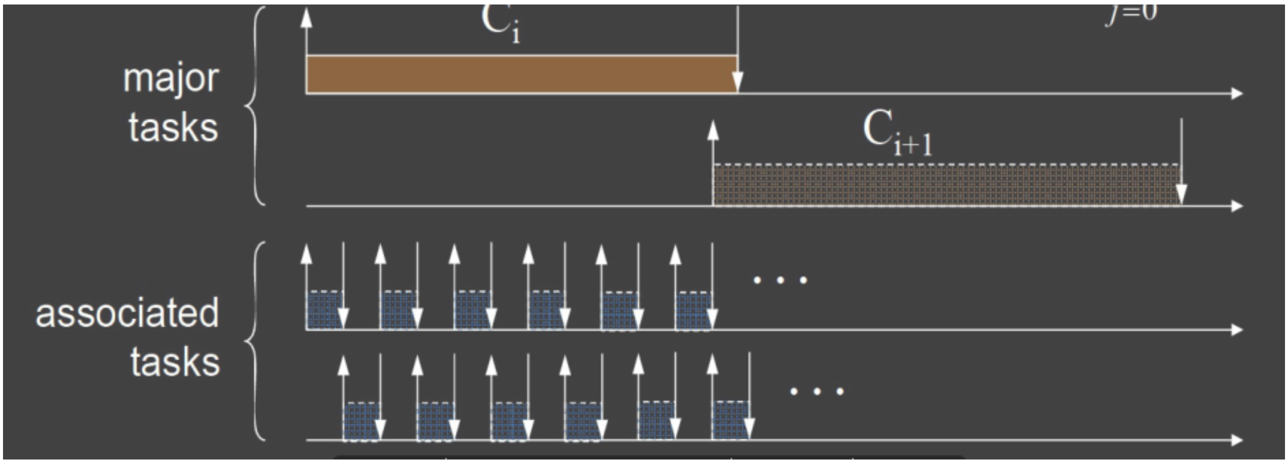


- If the player **aborts a major task** in favor of an associated task, the adversary **interrupts** the sequence of associated tasks.

- because $C_i = D_i$, even finish associated tasks, it cannot resume C_i, because it will miss deadline
- $\Gamma_{on} = \epsilon$



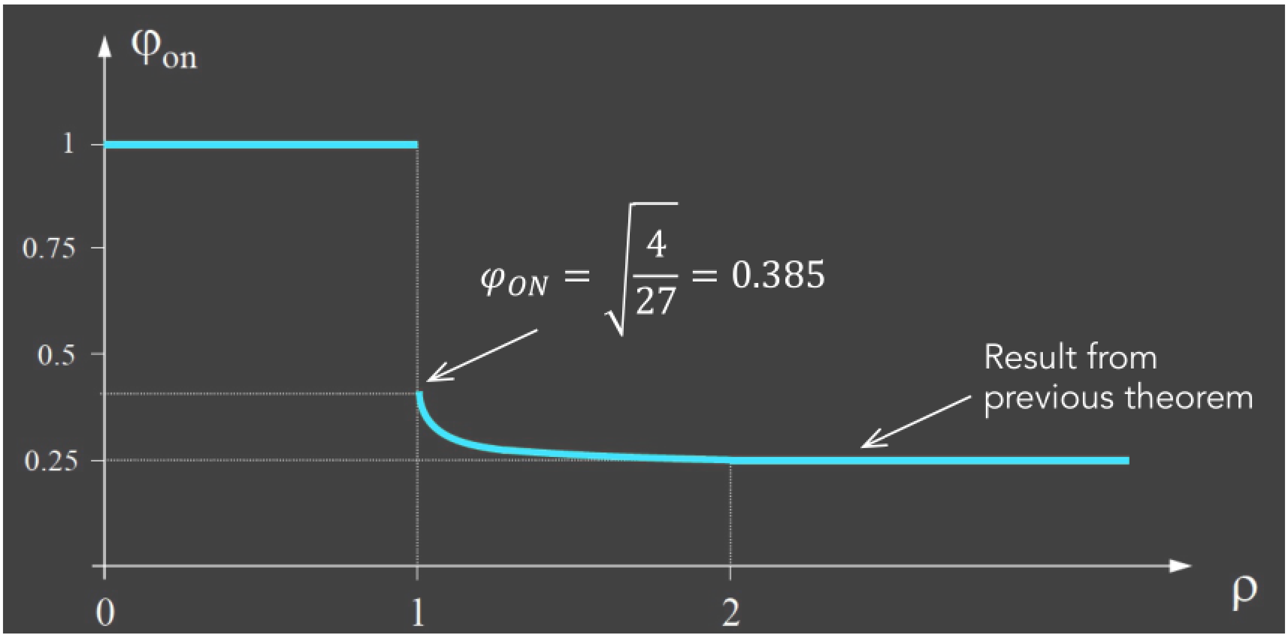
- If the player decides to complete Task i, the game terminates with the generation of Task i+1.
    - $\Gamma_{on} = C_i$
    - even if in the second part, it change to



A general Theorem

If $1 \leq \rho \geq 2$, then $\phi_{on} \leq p$, where p satisfies:

$$4[1 - (\rho - 1)p]^3 = 27p^2$$



2.2. Solution 1: Best-effort Scheduling

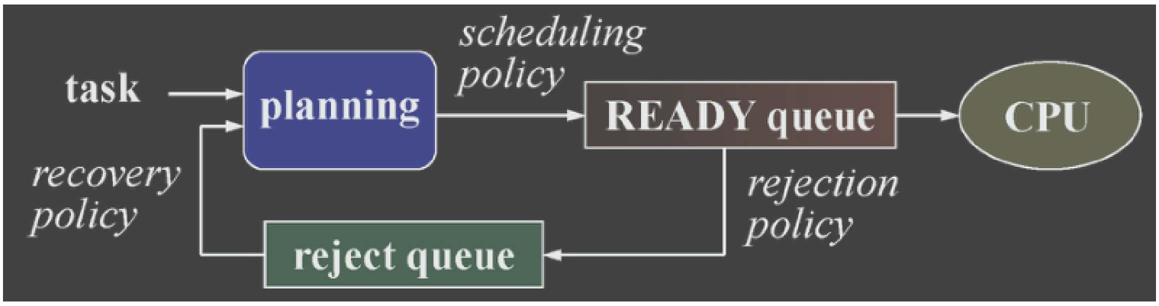
- **Admit all** incoming tasks/jobs;
- System operation is controlled by the scheduling (possibly, value-based) policy
- May cose **Domino Effect**

2.3. Solution 2: Admission-based Scheduling

- When a task is generated, we run a **schedulability test** and decide whether to admit it or not.
- This keeps the load below 1 and avoids domino effects.
- Low effieciency since we decide based on WCET

2.4. Solution 3: Robust Scheduling

- Tasks are scheduled by deadline and are rejected based on their value.
- In case of early completions, rejected tasks can be recovered by a reclaiming mechanism.



2.5. Example Solution 3: Robust EDF

Scheduling Policy: EDF

Rejection Policy: When an overload situation is detected, **reject the least value task** which can
bring the load below 1.

Recovery Policy: When there is enough spare time, re-accept the highest value task which
keeps the schedule still feasible.

Model of Robust EDF

$$J_i\left(C_i, D_i, M_i, V_i\right)$$

C_i : WCET

D_i: Relative Deadline

M_i: Flexible 2nd deadline (relative to D_i)

V_i: Task Value

Computation

using the Laxity values:

$$L_i = d_i - f_i \quad L_i = L_{i-1} + (d_i - d_{i-1}) - c_i(t)$$

$c_i(t)$: remaining computation time

Caculating the exceeding times

$$E_i = \max\left(0, -\left(L_i + M_i\right)\right) \quad E_{\max} = \max_{\mathrm{i}} \mathrm{E_i}$$

- always we suppose $M = 0$, then $E_i = 0$ means L_i is positive, not exceed deadline

Rejection:

reject the least-value task that can remove the overload.

- We can reject **1 or more tasks**; in the latter case, we need a more elaborate search to find the subset of tasks with the minimum value.

Algorithm: RED Acceptance Test
Input: A task set $\mathcal{J}$ with $\{C_i, D_i, V_i, M_i\}, \forall J_i \in \mathcal{J}$
Output: A schedulable task set
// Assumes deadlines are ordered by decreasing values

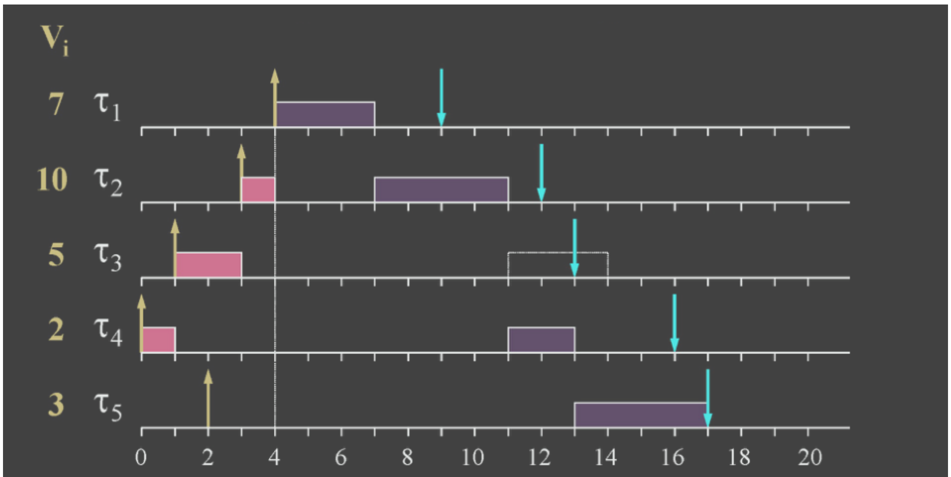
begin
    $E = 0;$ *// Maximum Exceeding Time*
    $L_0 = 0;$
    $d_0 = current\_time();$

    $J' = J \cup \{J_{new}\};$
    k = <position of J_{new} in the task set J'>;

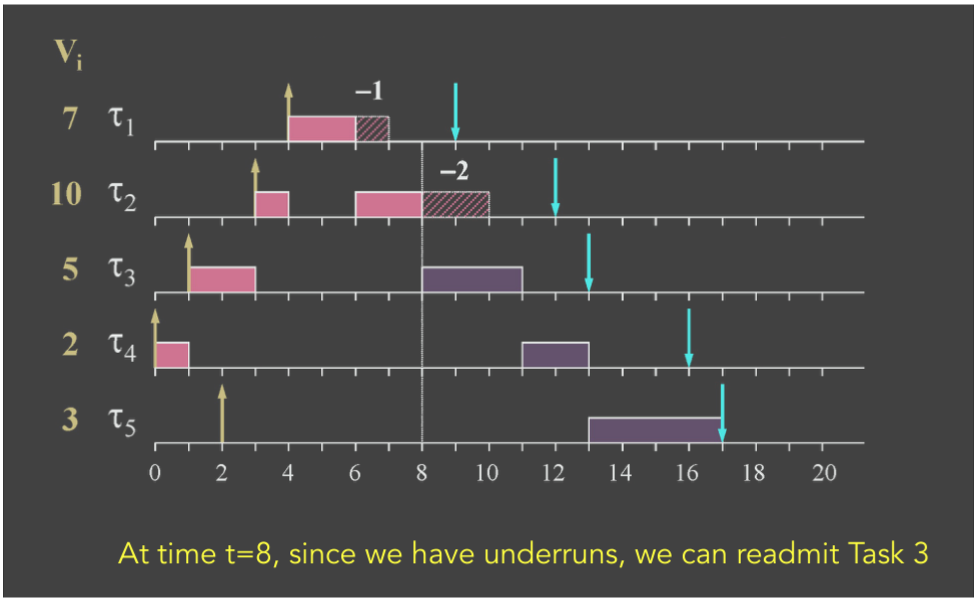
    for (each task J'_i such that $i \geq k$) **do**
        $L_i = L_{i-1} + (d_i - d_{i-1}) - c_i;$
        if $(L_i + M_i \; < \; -E)$ **then**   *// compute E_{max}*
            $E = -(L_i + M_i);$
        end
    end

    if $(E > 0)$ **then**
        <select a set J^* of least-value tasks to be rejected>;
        <reject all task in J^*>;
    end
end

Example



At time t=4, reject Task 3



3. Handeling Transient Overloads

Focus on: Overloads due to task overruns (even longer then WCET)

3.1. Solution: Resouce Reservation (RR)

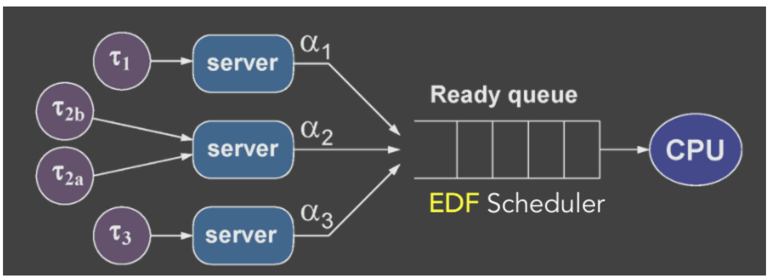
Reserve a fraction of the CPU for a set of tasks and prevent them from using more than that.

Main Idea

- For each task i, dedicate Q_i computation time every P_i time units.
- Thus, it is as if this is a **nice-behaving task with parameters** (Q_i, P_i).

Possible Implementation

RR can be enforced via a Constant Bandwidth Server (CBS)



Dimension RR Server and Schedulability Analysis

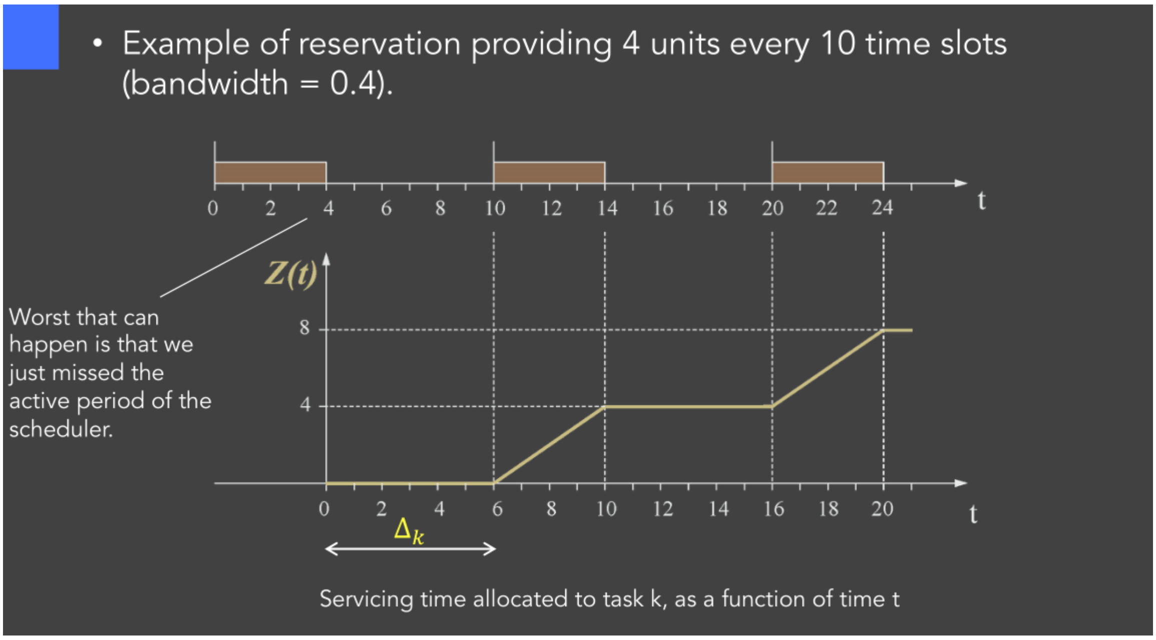
Want to know the **bandwidth** α_k for each task k

the **maximum delay** Δ_k that this resource is not available.

Supply Function

Given a reservation, the **supply function** $Z_k(t)$ is the **minimum amount of time provided by reservation k** in every time interval of length t ≥ 0

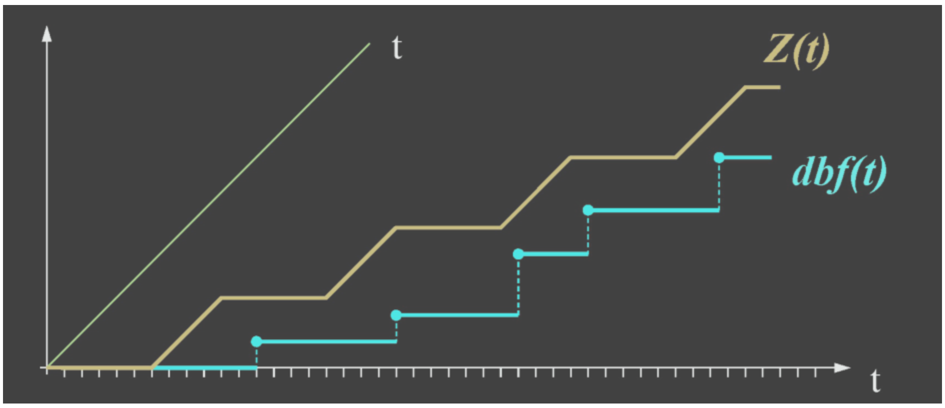
Example for Static Time Partition



Exact Test

The **processor demand criterion** (Sec. 4.6.1) can be reformulated as an exact test using the **demand bound function** (dbf):

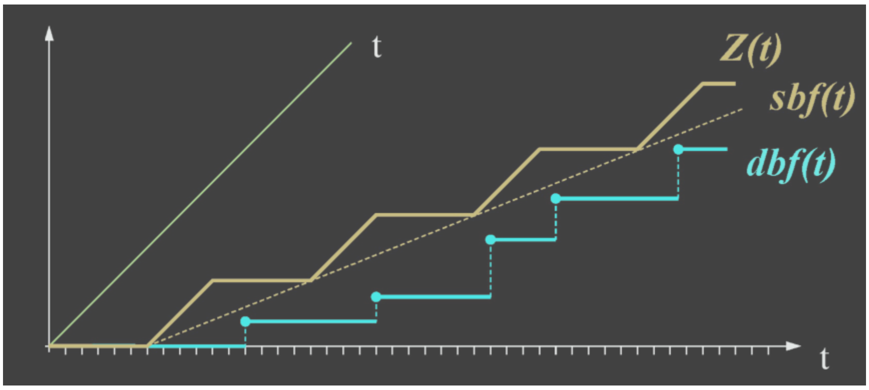
$$\forall t > 0, \quad dbf(t) \leq Z(t)$$



Sufficient Test

simpler (sufficient) test can be derived using **a lower bound of** $Z(t)$, namely the **supply bound function**:

$$\forall t > 0, \quad dbf(t) \leq sbf(t)$$



- Once the **bandwidth** and the delay are computed, a supply bound function can be expressed as follows

$$sbf(t) = \max\{0, a(t - \Delta)\}$$

a: bandwidth: the gradient

Δ: service delay

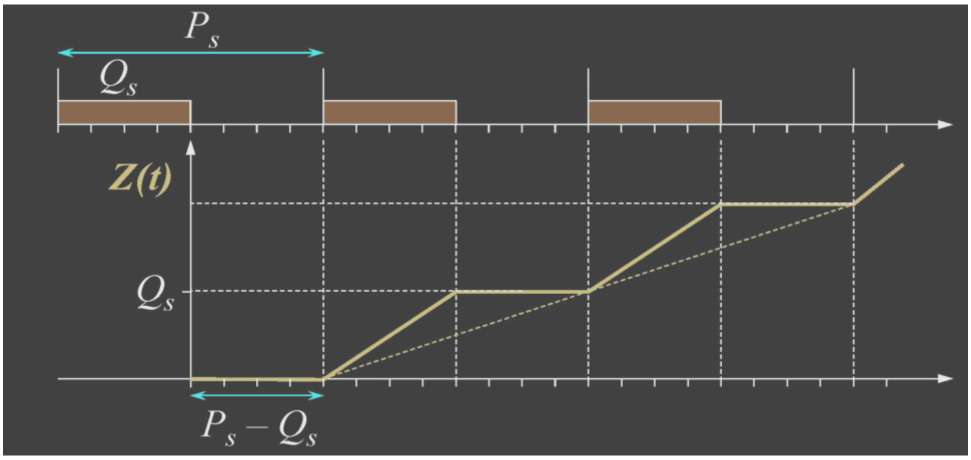
- For a given $Z(t)$, we can compute bandwidth and delay as:

$$a = \lim_{t \to \infty} \frac{Z(t)}{t} \qquad \Delta = \sup_{t \geq 0} \left\{ t - \frac{Z(t)}{a} \right\}$$

Examples for a Periodic Server:

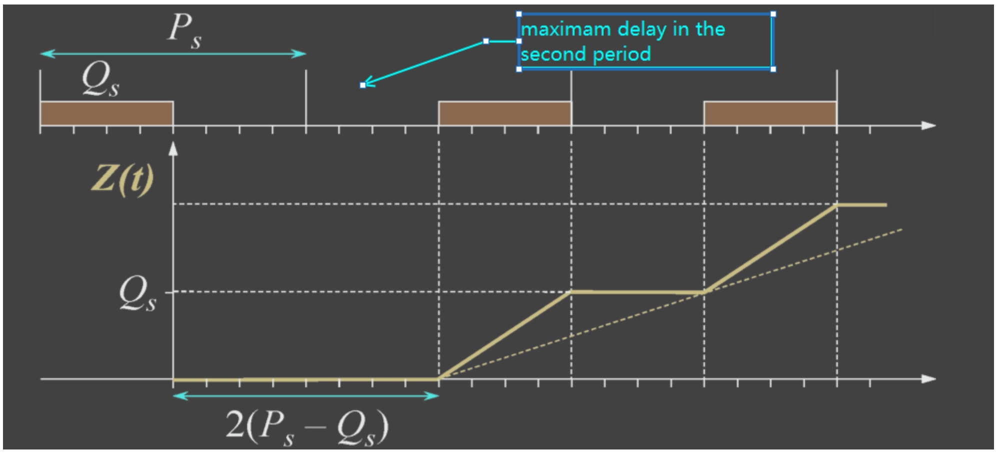
For a periodic server with budget Q_s and period P_s running at **the highest priority**, we get:

$$a = \frac{Q_S}{T_S} \qquad \Delta = P_S - Q_S$$



For a periodic server with budget Q_s and period P_s running at **unkown priority**, we get:

$$a = \frac{Q_S}{T_S} \quad \Delta = 2\left(P_S - Q_S\right)$$



Another Exact Test Scenarios

A set of **preemptive periodic tasks** with relative deadlines **less than or equal to periods**, can be scheduled by **EDF**, under a reservation characterized by a supply function $Z_k(t)$, **if and only if** $U_k < \alpha_k$ and:

$$\forall t > 0, \quad dbf(t) \leq Z_k(t)$$

3.2. Adjust the RR Server Parameters

haven't dimensioned properly each CBS?

- If Q_s is smaller than needed, the tasks will progress very slowly;
- If it is larger, then we will waste resources.

Solution 1:

transfer unused budgets across the servers.

Solution 2:

measure the actual task needs and **adjust their WCET**.

$$C_i' = \max_{k:Job \text{ of } \tau_i} \{e_{i,k}\} \quad U_i' = C_i'/T_i$$