Aperiodic Servers

Real-Time Industrial Systems

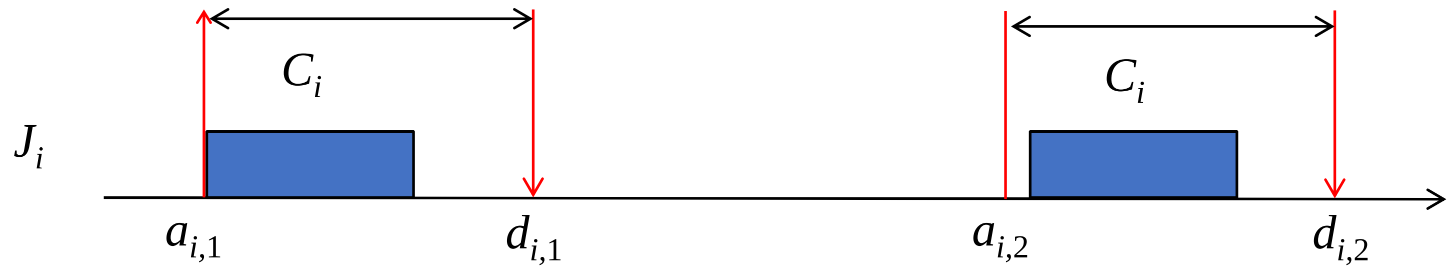
Marcello Cinque

Roadmap

- Scheduling of aperiodic tasks
- Aperiodic servers:
  - Static solutions
  - Dynamic solutions

- References:
  - Giorgio Buttazzo: "Hard real-time computing systems: Predictable Scheduling Algorithms and Applications", Third Edition, Springer, 2011

Aperiodic and sporadic task model



- Aperiodic tasks: $a_{i,k+1} > a_{i,k}$
- Sporadic tasks: $a_{i,k+1} > a_{i,k} + \Delta_i$
  - Aperiodic tasks where jobs are separated by a minimum inter-arrival time Δ_i

Siamo sicuri che non si ripresenteranno almeno per un Delta time, informazione utilissima per la verifica della fesability

Scheduling of aperiodic real-time tasks

- If alone in the system
  - if sporadic, they can be assigned a fixed priority inversely proportional to the deadline (as if scheduled with DM, using FP-based schedulability analysis)
  - otherwise they can be scheduled with EDF, adding an **on-line admission check** (or **warranty test**):

$$\forall\, i = 1, \dots, n \quad t + \sum_{k=1}^{i} c_k(t) \le d_i$$

- And if not alone in the system?

Mixed scheduling

- Systems with a mixture of tasks
  - Periodic, hard
  - Aperiodic, hard or soft

- The problem is to guarantee periodic tasks while not delaying aperiodic tasks excessively
  - Soft aperiodic tasks should not be penalized
  - Hard aperiodic tasks must be scheduled in order to meet their deadlines

- Background scheduling for aperiodic tasks could be used
  - But it cannot guarantee aperiodic tasks deadlines!

Aperiodic tasks guarantees

- Schedulability guarantee can be requested to individual instances or to the entire aperiodic task (all istances)
- Instance-based guarantee must be assessed on-line, when the instance arrives
- Task-based guarantee must be assessed off-line, before the task starts
  - In such a case, it is necessary to assume a minimum interarrival time between the instances → sporadic task model

Aperiodic Servers

- A periodic process (server) dedicated to the service of aperiodic requests
  - In order to have a budget available to guarantee aperiodic task execution together with periodic tasks
    In pratica uno spazio computazionale, una frazione di tempo riservata ai task aperiodici

- An aperiodic server is characterized by a period T_s and a capacity or budget C_s

Lo possiamo vedere come un task periodico di periodo Ts e costo Cs nel quale eseguire i task aperiodici

Polling Server (PS)

- Every T_s the Polling Server is activated to serve pending aperiodic requests
  - To the requests it is given a maximum execution time equal to C_s

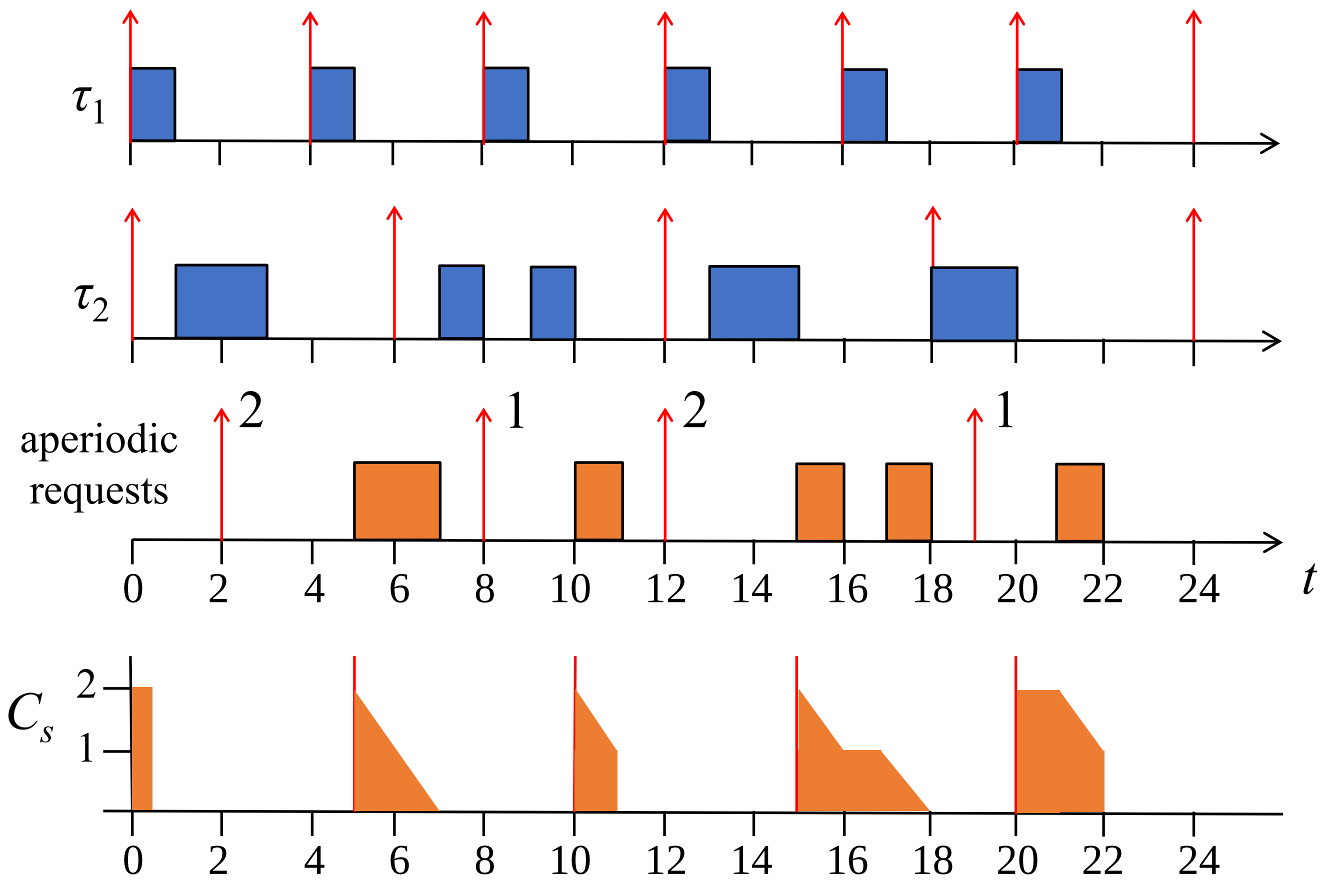
- If no requests are pending, the server is suspendend until the next period
  - If an aperiodic task arrives when the server is idle, it will be served at the next period

- It can work with either RM or EDF (so either static or dynamic)

Polling Server with Rate Monotonic

	τ_1	τ_2
C_i	1	2
T_i	4	6

Server	
C_s	2
T_s	5



In questo caso si assegna una priorità anche al server con le stesso regole di RM, il Server avrà priorità compresa tra Tau1 e Tau2 4<5<6

PS: guarantee test for periodic tasks

- In the worst case, the PS behaves as a periodic task with execution time C_s and period T_s
- We can thus reuse the Liu and Layland test extended to n+1 taks:

A questo punto si applicano le stesse regole di RM considerando anche Cs/Ts

$$\underbrace{\sum_{i=1}^{n}\frac{C_i}{T_i}}_{\substack{U_p \\ \text{Periodic} \\ \text{Utilization}}} + \underbrace{\frac{C_s}{T_s}}_{\substack{U_s \\ \text{Aperiodic} \\ \text{Bandwidth}}} \le (n+1)[2^{1/(n+1)}-1]$$

PS: guarantee test for periodic tasks and hard sporadic tasks

- In case of a mixed task set with:
  - n_p periodic hard tasks,
  - n_s sporadici hard tasks, with minimum interarrival time set to the relative deadline,
  - n_a aperiodici soft tasks, managed by a single PS,
- the guarantee test can be:

$$\sum_{i=1}^{n_p} \frac{C_i}{T_i} + \sum_{i=1}^{n_s} \frac{C_i}{D_i} + U_s \leq U_{lub}(n_p + n_s + 1)$$

lub stands for Least Upper Bound

PS: guarantee test for aperiodic hard tasks

- In this case, the test must be executed on-line, when the aperiodic task is activated
- Considering an aperiodic task J_a with execution time C_a and deadline D_a:
  - If $C_a \leq C_s$, then the aperiodic task will be completed within two activations of the PS, hence a guarantee test can be:

Sicuro in 2 tempi perchè nel caso peggiore il task mi arrivera un microsecondo dopo l'attivazione ma sono sicuro chhe alla prossima attivazione lo terminerò

Allora il test di fattibilità risulta: $2T_s \leq D_a$

  - For arbitrary execution times, the test becomes:

$$T_s + \left\lceil \frac{C_a}{C_s} \right\rceil T_s \leq D_a$$

Deferrable Server (DS)

- As the PS, it is a periodic task that serve aperiodic requests
- Differently from the PS, it keeps its capacity even when no aperiodic requests are present at its activation

  in questo caso quindi if Ca<=Cs mi basta un solo tempo perchè non mi metto a dormire ma aspetto

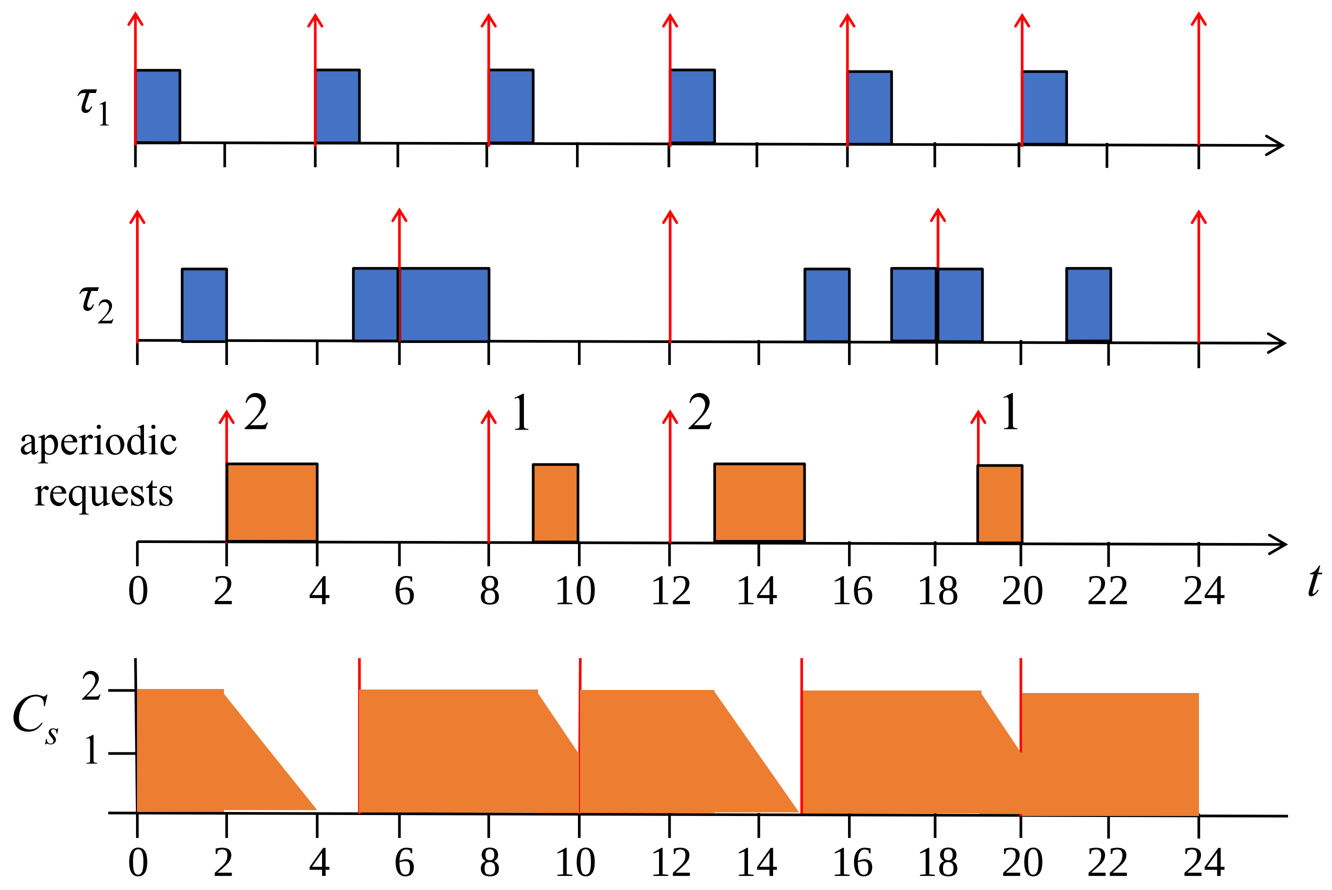
- So, the DS improves the serving time of aperiodic requests
  - Since the capacity is kept during the whole period, aperiodic requests can be served as soon as they arrive (according to the priority of the server) as long as there is still capacity

- It is a **static** solution, based on RM

DS with RM

	τ_1	τ_2
C_i	1	2
T_i	4	6

Server	
C_s	2
T_s	5

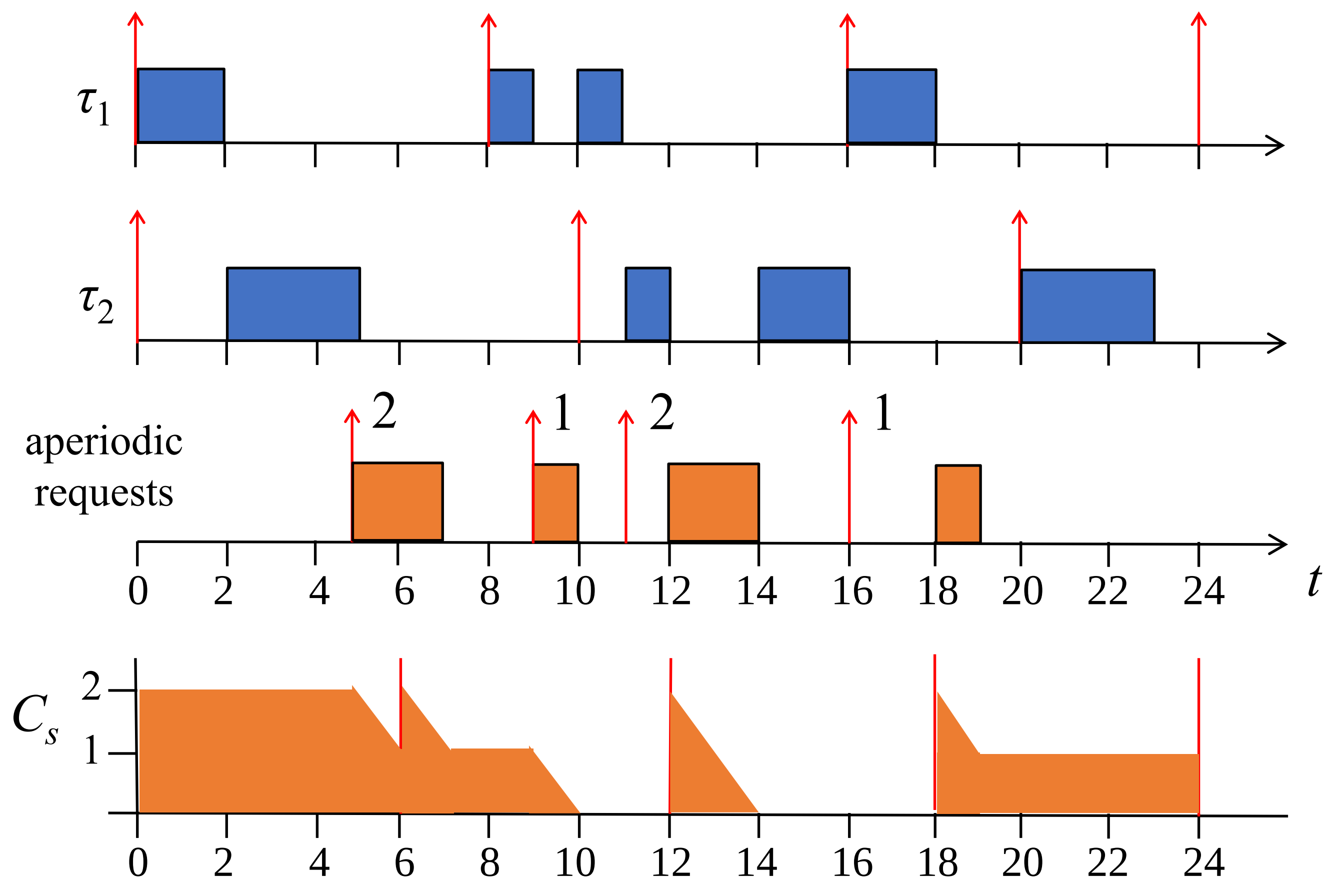


Mantemgo la capacità finchè non mi arriva l'arancione,

DS with RM (maximum priority server)

	τ_1	τ_2
C_i	2	3
T_i	8	10

Server	
C_s	2
T_s	6



A differenza del Polling Server dove siamo sicuri che di occupare la banda per Ctempo ad ogni intervallo, con il Defferbale Server possiamo anche occupare il 100% di banda (restiamo in attesa con Capacità massima e serviamo immediatamente il task senza schedularlo nel prossimo intervalo) - uso più banda di quella che realmente mi serve

DS: schedulability test

- The behavior of a DS is not equivalent to a periodic task, as, in some situations and for limited time intervals, it can consume more CPU bandwidth than a normal periodic task

- So, the classical schedulability test cannot be used for the DS, and the following can be adopted when the DS is the "task" with maximum priority:

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

  - where U_p is the utilization factor of periodic tasks and U_s is the DS bandwidth

Solutions based on CPU Bandwidth management

- To avoid the interference problem of the DS, other servers are based on the management of CPU bandwidth, trying to keep the CPU utilization of aperiodic requests always within the budget U_s

Us= Cs/Ts

- Examples are:
  - Sporadic Server (static)
  - Dynamic Sporadic Server (dynamic)
  - Total Bandwidth Server (dynamic)
  - Constant Bandwidth Server (dynamic)

Sporadic Server (SS)

- It is a static bandwidth (or capacity) manager (works with RM)
- Differently from DS, the capacity is not refilled regularly at each period, but within a period from the instant it has been consumed

- It can be demonstrated that the SS behavior is equivalent to a periodic task with period T_s and capacity C_s scheduled with a static priority P_s

Sporadic Server rules

per calcolare la priorità
utilizziamo la regola RM

- The SS is *active* if the running task has priority $p \geq P_s$, *idle* otherwise

- If, at time t the server becomes active and there is capacity:
  - The capacity consumed until the SS becomes idle or Cs = 0 will be replenished after a period T_s from time t
  - In other terms, defined $\boldsymbol{RT}$ as the *Replenishment Time*, it is:

$$RT = t + T_s$$

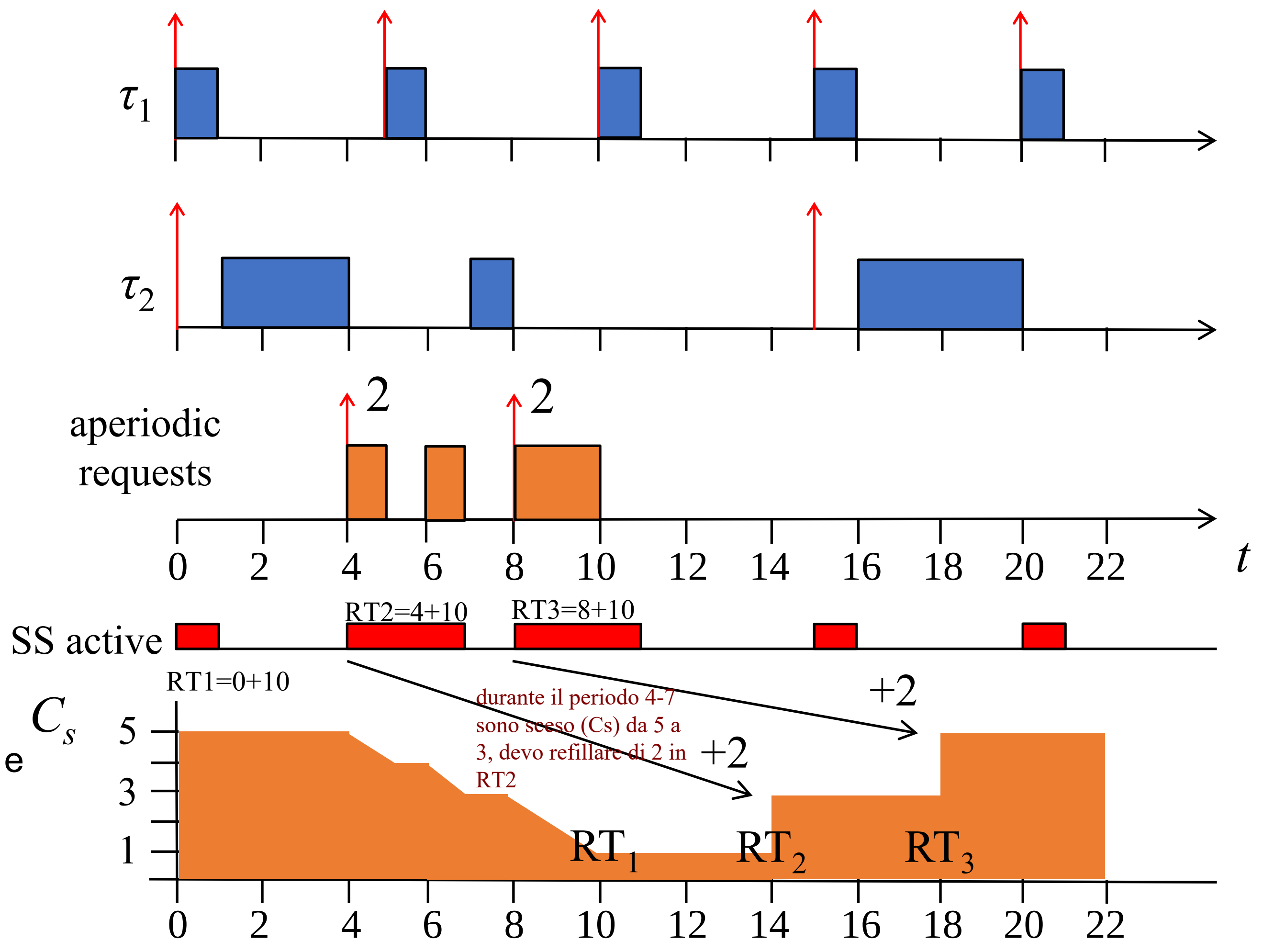
Molto più difficile da implementare rispetto a Polling e Defferable Server

SS: example

	τ_1	τ_2
C_i	1	4
T_i	5	15

Server	
C_s	5
T_s	10

In Rt1 non ricarico nulla perchè nel primo punto rosso non ho consumato. In RT2 ricarico quanto consumato tra 4 e 7, ovvero 2 e così via



durante il periodo 4-7 sono sceso (Cs) da 5 a 3, devo refillare di 2 in RT2

Dynamic Sporadic Server (DSS)

- Dynamic variant of the SS where the RT coincides with the server deadline *ds* and tasks (including the DSS) are scheduled with EDF
- If U_s is the server bandwidth, where $U_s = C_s / T_s$, and U_p is the utilization of periodic tasks, then the total periodic task set plus the DSS is schedulable with EDF iff:

$$\boxed{U_p + U_s \leq 1}$$

Il Budget (C/T) dei task periodici pìu quello del
Servere deve essere minore uguale di 1

Total Bandwidth Server (TBS)

- Dynamic solution
- Adapts the service to the characteristics of the aperiodic request:
  - Close deadline if the requested calculation is low
  - Far deadline if the requested calculation is high
- In detail, when the k-th aperiodic request arrives a time $t = r_k$, it receives a "*personalized*" deadline and it is scheduled with EDF:

Ck la stimiamo per ogni aperiodico

rk quando la richiesta arriva

$$d_k = \max(r_k , d_{k-1}) + C_k / U_s$$

Problem:
What if the estimate of C_k is wrong??

- U_s is the *server bandwidth*. At start: $d_0 = 0$

Constant Bandwidth Server (CBS)[1]

- Dynamic solution
- It introduces a protection mechanism that assures that the server bandwidth $U_s = C_s \,/\, T_s$ is never overconsumed, independently from aperiodic requests computation times C_k
- The server state is represented by
  - c_s: current budget at time t (initially 0)
  - d_s: current server deadline (intially 0)
- The same guarantee test of DSS applies to CBS and TBS $U_p + U_s \leq 1$

1) L. Abeni and G. Buttazzo, «Integrating multimedia applications in hard real-time systems», Proc. of the IEEE Real-Time Systems Symposium, 1998

CBS rules

1. If a new aperiodic request arrives when another request is served, it is simply enqueued (with FIFO policy)
2. If a new aperiodic request arrives at time t when when the server is idle

   garantisco di non sforare Us
   - If $0 < c_s \leq (d_s - t)U_s$ it is served with the current server parameters
   - Else, the budget is refilled, and the deadline delayed: $c_s = C_s$ and $d_s = t + T_s$
3. When a request is completed, the server picks-up the next from its queue, if present, and it is served with the current server parameters
4. When the budget is exhausted ($c_s = 0$) it is always refilled ($c_s = C_s$) and the deadline delayed ($d_s = d_s + T_s$)

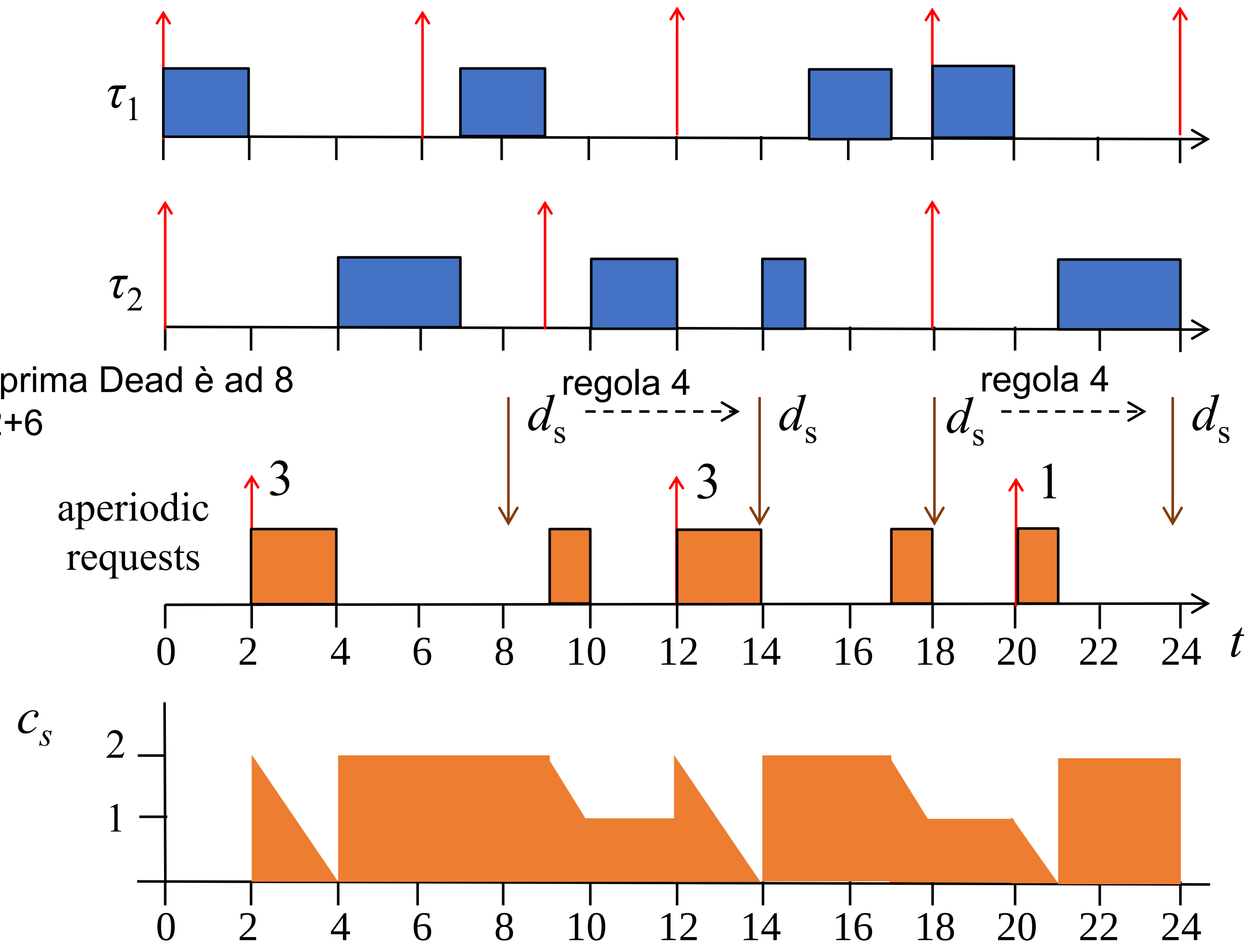
Non abbiamo una priorità fissa come prima, ogni volta devo guardare le Deadline

CBS: example

	τ_1	τ_2
C_i	2	3
T_i	6	9

Server	
C_s	2
T_s	6

inizialmente la prima Dead è ad 8 data da t+Ts=2+6



Regola 2
Cs(attuale)<= (ds-t)Us
1 <= (14-12)*(2/6)=0,66

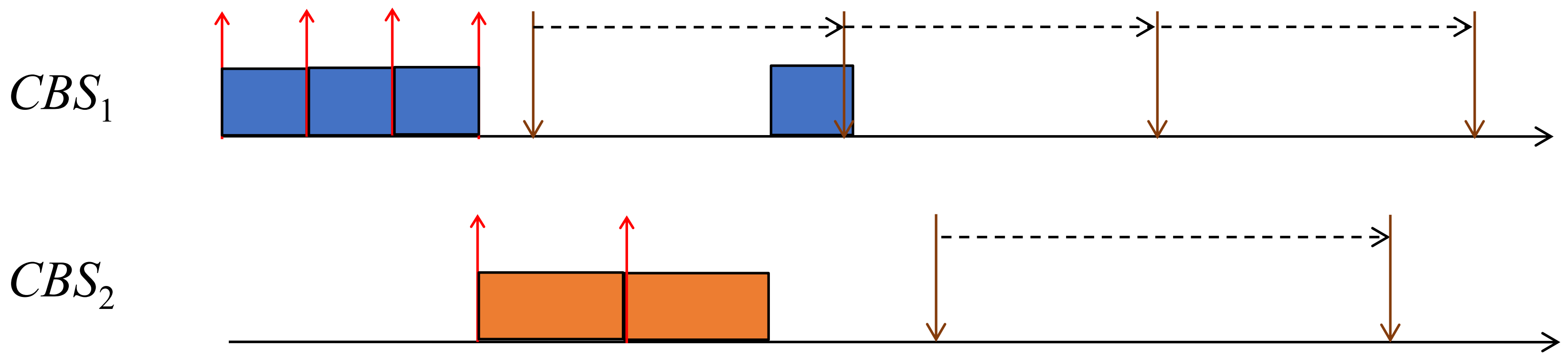
Hard and soft reservations

- A server is said to implement a ***hard reservation*** if, when the server budget is depleted, the server is suspended until the next replenishment time.
  - e.g.: PS, DS, SS, DSS

- A server is said to implement a ***soft reservation*** if, when the server budget is depleted, it is immediately replenished, so that the server remains always active (*backlogged*)
  - e.g.: TBS, CBS

The deadline aging problem

- Soft reservation servers, and so the CBS, may suffer of the deadline aging problem



The deadline aging problem

- The problem causes the amount of execution granted by a server to depend on the activations and periods of other servers or of periodic tasks

Ad esempio in Polling Servere davamo come upepr bound 2periodi

- This makes not possible to provide <u>an upper bound on the service delay</u> on aperiodic requests in any time interval
  - Not possible to provide analytical guarantees to aperiodic tasks deadlines
  - If a critical aperiodic task arrives at the beginning of a long black-out period, it would inevitably miss its deadline!
- So, the CBS needs to be modified if hard reservations are needed
  - Can provide hard schedulability only if it serves a sporadic task J_i with:
    $C_i \leq C_s$ and $D_i \geq T_s$

Hard CBS[2]

- Rules no. 2 and 4 of the CBS are modified to enforce hard reservations:

2. If a new aperiodic request arrives at time t when when the server is idle, a replenishment time is computed as $RT = d_s - c_s / U_s$
   a. if $t < RT$, the server is suspended until RT. At time RT, the server returns active, the budget is refilled, and the deadline delayed: $c_s = C_s$ and $d_s = RT + T_s$
   b. otherwise the budget is immediately replenished to C_s and $d_s = t + T_s$

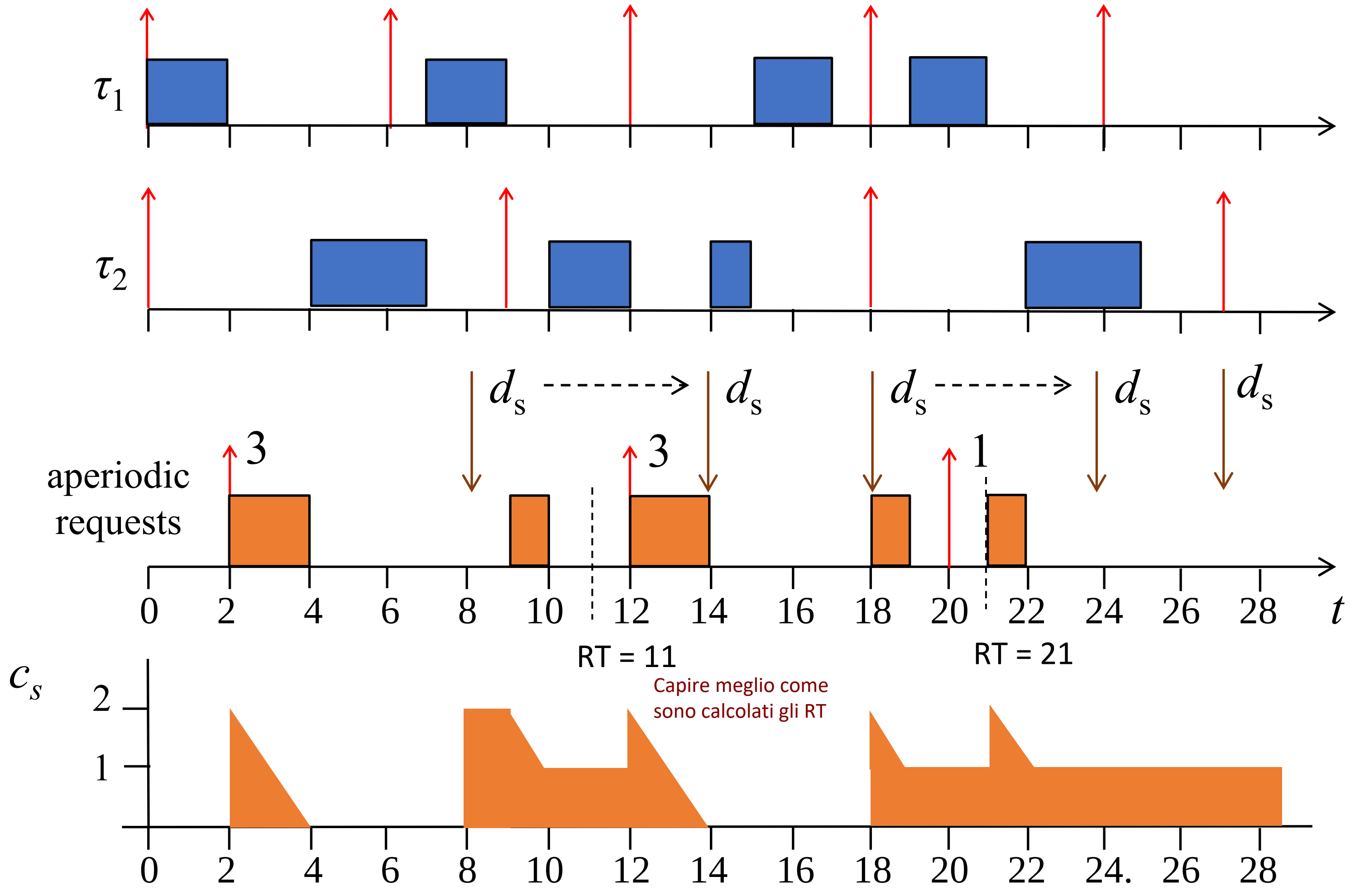
4. When the budget is exhausted ($c_s = 0$) the server is suspended until d_s. At time d_s it is refilled ($c_s = C_s$) and the deadline delayed ($d_s = d_s + T_s$)

2) A. Biondi, A. Melani, M, Bertogna, «Hard Constant Bandwidth Server: Comprehensive formulation and Critical Scenarios», Proc. of the 9th IEEE International Symposium on Industrial Embedded Systems (SIES 2014)

Hard CBS: example

	τ_1	τ_2
C_i	2	3
T_i	6	9

Server	
C_s	2
T_s	6



The deadline aging problem… solved

- Since the server is suspended until the next deadline, excessive requests are turned into a hard (predictable) periodic behavior