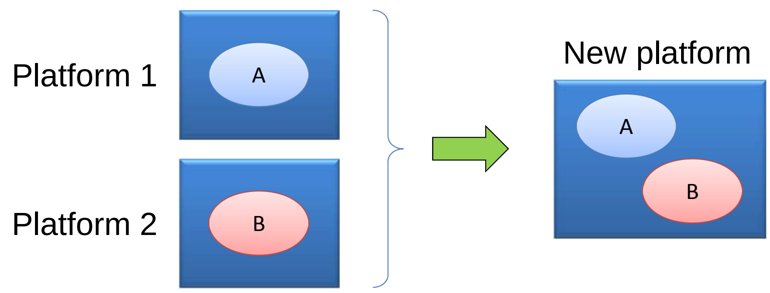
Reducing Execution Interference

Giorgio Buttazzo
g.buttazzo@sssup.it

Real-Time Systems Laboratory
Scuola Superiore Sant'Anna

The problem

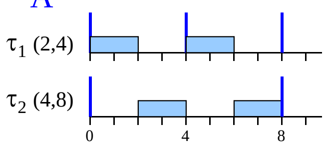


Consider two applications, A and B, independently developed on dedicated platforms.

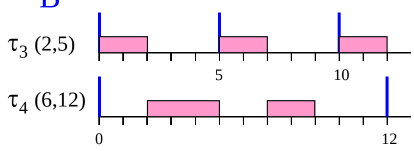
How can we **guarantee** them when they are concurrently executed in the same platform?

Example with fixed priorities

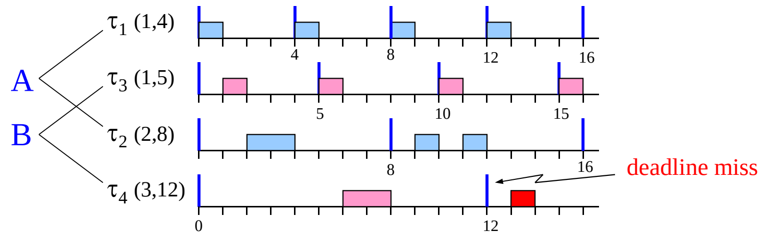
RM schedule (S = 1)



RM schedule (S = 1)



A+B **RM schedule (S = 2)** $\Rightarrow$ assuming $C'_i = \frac{C_i}{s}$



Example with EDF

Application A

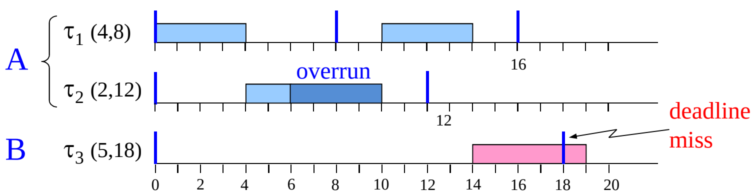
A $\left\{ \begin{array}{l} \tau_1\ (4,8) \\ \tau_2\ (2,12) \end{array} \right.$ $U_A = \frac{1}{2} + \frac{1}{6} = \frac{2}{3}$

Application B

B $\tau_3\ (5,18)$ $U_B = \frac{5}{18}$

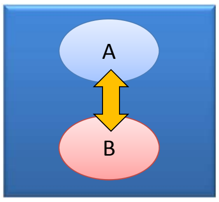
$$U_{TOT} = \frac{2}{3} + \frac{5}{18} = \frac{17}{18} < 1$$

A+B **EDF schedule**



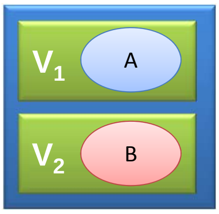
Resource partitioning

Clearly, when multiple applications execute on the same platform, they compete for the same resources and may delay each others.



What we need is a mechanism able to partition the processor in two subsystems (virtual processors), each dedicated to a single application.

In this way, an overrun occurring in an application does not propagate to the others, but only affects the application itself.

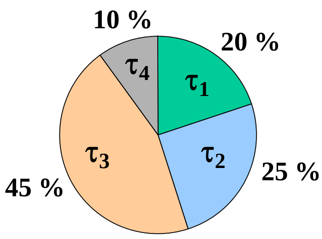


Resource Reservation

In general, what we really need is to have:

Resource partitioning

Each application receives a fraction $\alpha_i < 1$ of the processor sufficient to meet its execution requirements.

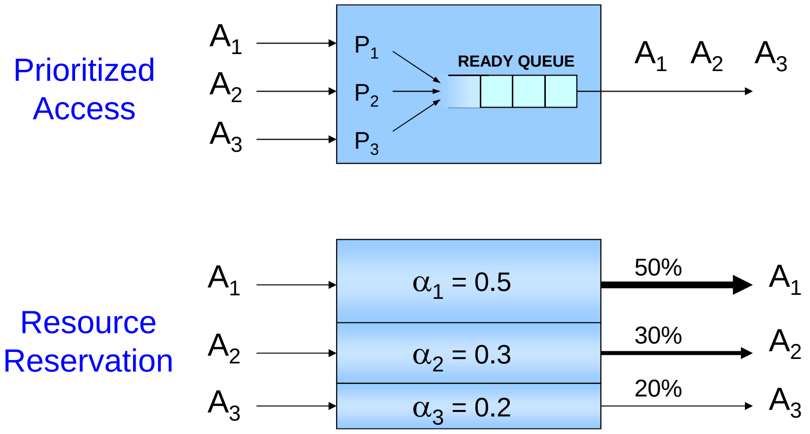


Enforcement mechanism

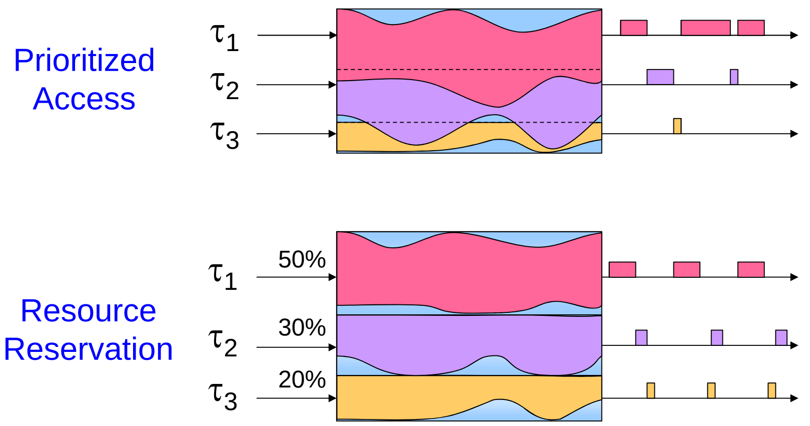
A mechanism that prevents an application to consume more than its reserved fraction.

In this way, the application executes as it were executing alone on a slower processor with speed α_i.

Priorities vs. Reservations



Priorities vs. Reservations



Priorities vs. Reservations

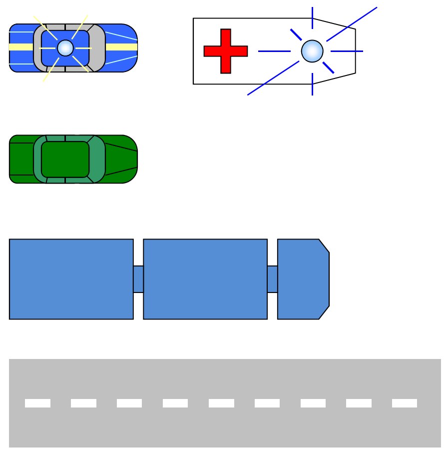
Tasks ⇒ vehicles ***Resource ⇒ road***

High priority

Medium priority

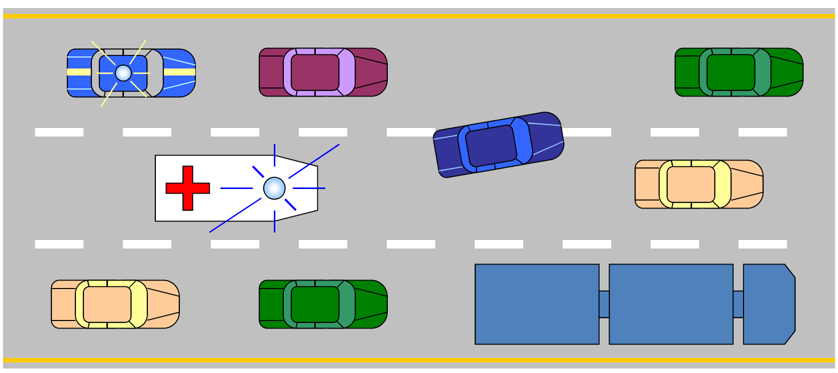
Low priority

Shared resource



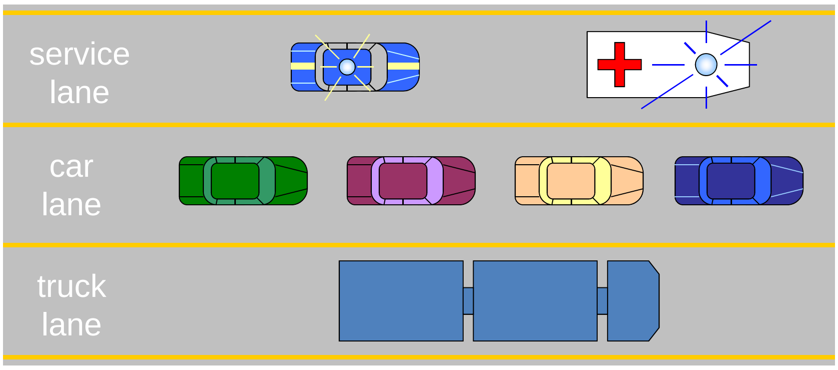
Prioritized approach

Problems in overload conditions



Resource Reservation

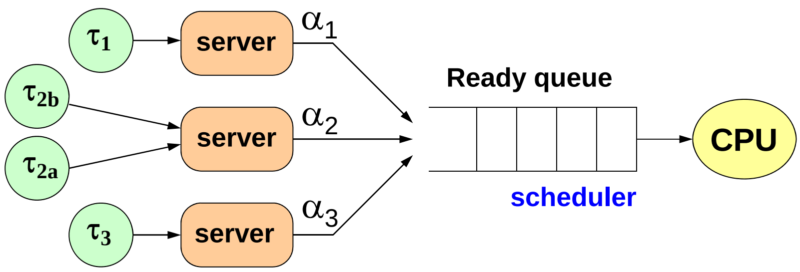
Each type of vehicle has a reserved lane



Benefits of Res. Reservation

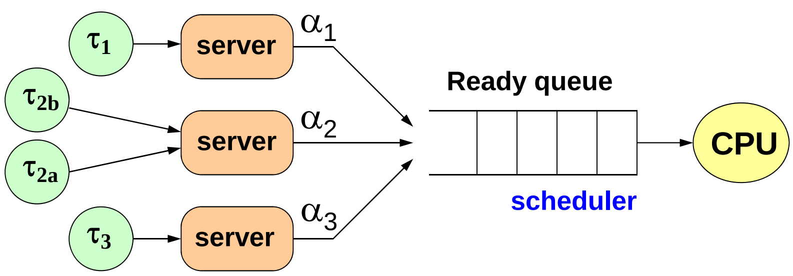
1. Resource allocation is easier than priority mapping.
2. It provides temporal isolation: overruns occurring in a reservation do not affect other tasks in the system.
   - Important for modularity and scalability
3. Simpler schedulability analysis:
   - Response times only depends on the application demand and the amount of reserved resource.
4. Easier probabilistic approach

Implementing Reservations



	scheduler	server
Fixed priorities	**RM/DM**	**Sporadic Server**
Dynamic priorities	**EDF**	**CBS**

Guaranteeing Reservations



If a processor is partitioned into n reservations, we must have that:

$$\sum_{i=1}^{n} \alpha_i \leq U_{\text{lub}}^{A}$$

where A is the adopted scheduling algorithm.

Hard vs. Soft reservations

SOFT reservation

It guarantees that the served application receives at least a budget Q every P.

HARD reservation

It guarantees that the served application receives at most a budget Q every P.

Constant Bandwidth Server

- It assigns deadlines to tasks as the TBS, but keeps track of job executions through a budget mechanism.
- When the budget is exhausted it is immediately replenished, but the deadline is postponed to keep the demand constant.

CBS parameters

Maximum budget:	Q_s		assigned by the user
Server period:	T_s		
Server bandwidth:	$U_s = Q_s/T_s$		
Current budget:	q_s	(initialized to 0)	maintained by the server
Server deadline:	d_s	(initialized to 0)	

16

Basic CBS rules

Arrival of job J_k at time r_k $\Rightarrow$ assign d_s

if $(r_k + q_s/U_s \leq d_s)$ **then** recycle (q_s, d_s)

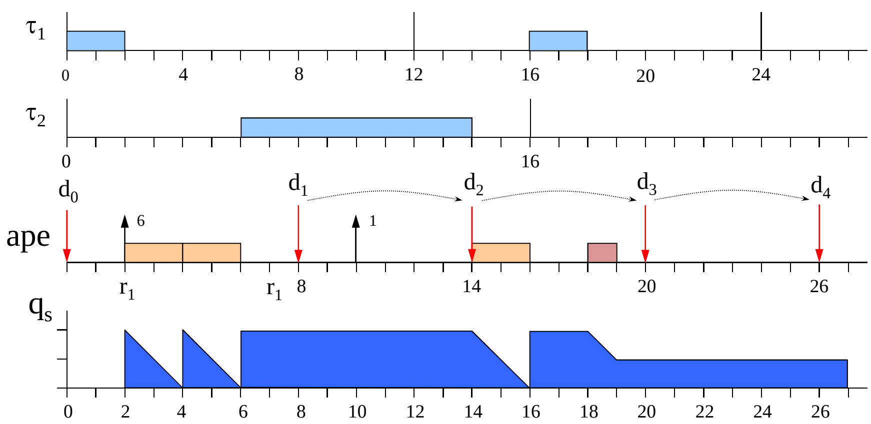
else $\begin{cases} d_s = r_k + T_s \\ q_s = Q_s \end{cases}$

Budget exhausted $\Rightarrow$ postpone d_s

$\begin{cases} d_s = d_s + T_s \\ q_s = Q_s \end{cases}$

The server remains active

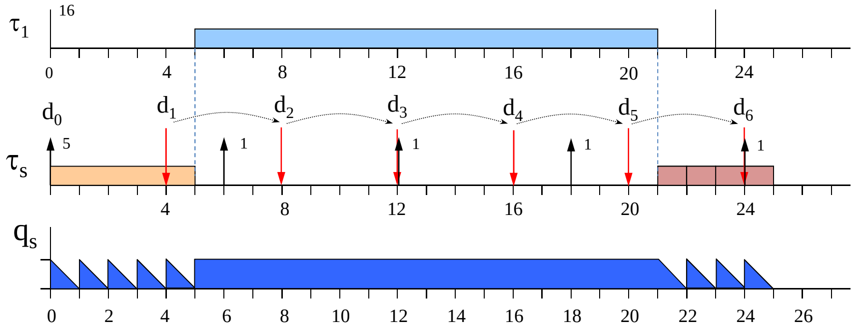
Example of Soft CBS



CBS: $Q_s = 2$, $T_s = 6$

Problem of Soft CBS

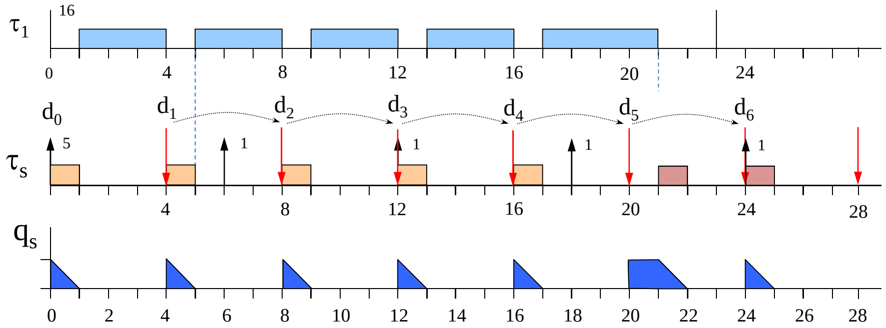
If more bandwidth is allocated during idle intervals, the server deadline goes far away (deadline aging) and the executions tend to become less regular.



CBS: $Q_s = 1$, $T_s = 4$

Example with Hard CBS

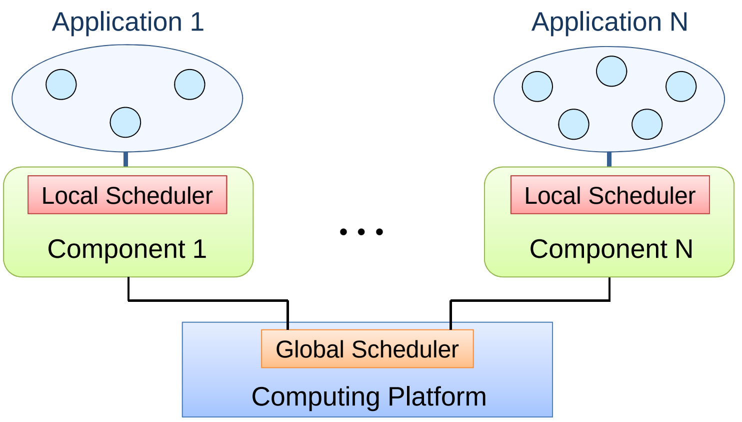
The deadline aging problem can be avoided by suspending the server when $q_s = 0$ and recharging the budget at d_s:



CBS: $Q_s = 1$, $T_s = 4$

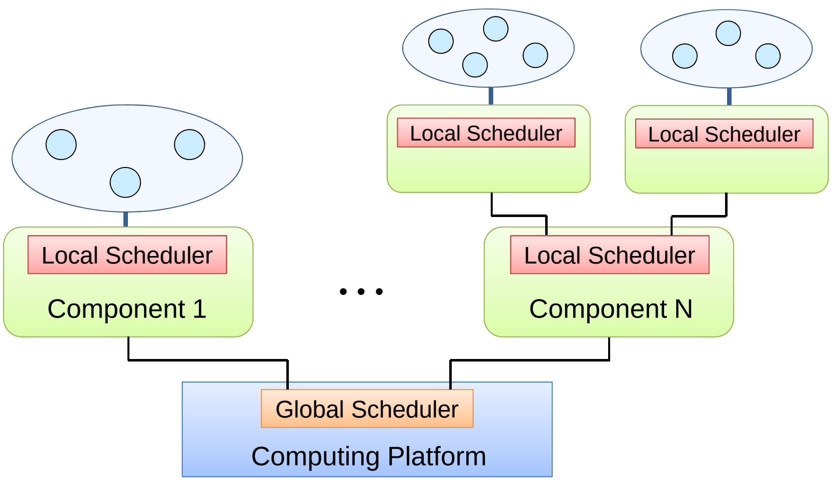
Hierarchical scheduling

Resource reservation can be used to develop hierarchical systems, where each component is implemented within a reservation:



Hierarchical scheduling

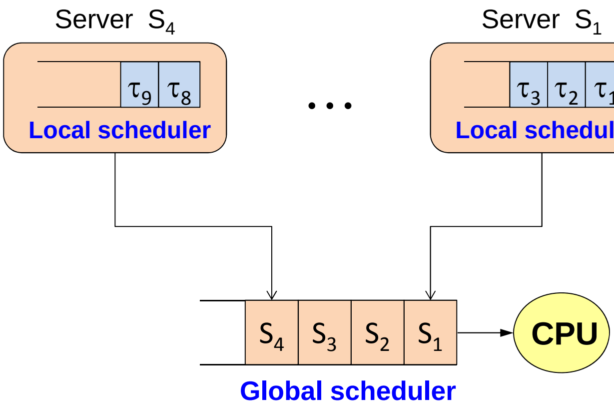
In general, a component can also be divided in other sub-components



Hierarchical scheduling

Global scheduler: the one managing the system ready queue

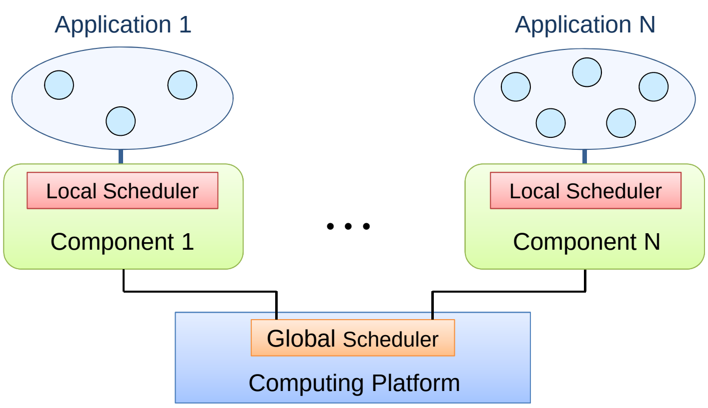
Local schedulers: the ones managing the servers queues



Hierarchical Analysis

Global analysis: Servers must be schedulable by the global scheduler running on the physical platform.

Local analysis: Applications must be scheduled by the local schedulers running on the components.



Analysis under RR

Under EDF, the analysis of an application within a reservation is done through the Processor Demand Criterion:

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

Under Fixed Priority Systems (FPS), the analysis is done through the Workload Analysis:

$$\forall i = 1, \ldots, n \quad \exists\, t \in (0, D_i] : W_i(t) \leq t$$

The difference is that in an interval of length t the processor is only partially available.

Analysis under RR

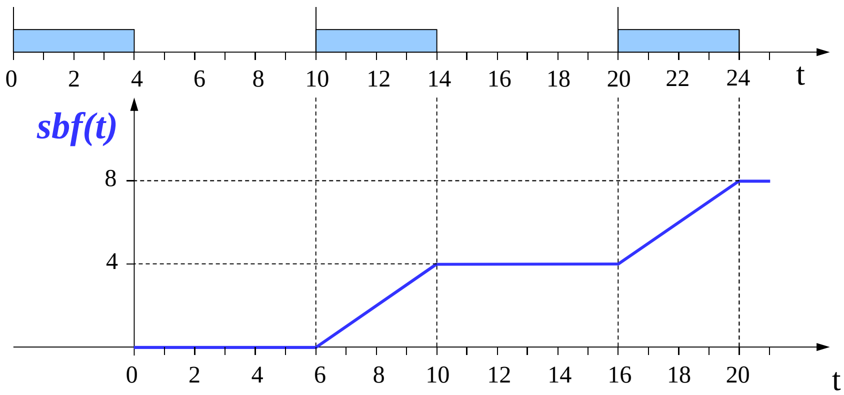
To describe the time available in a reservation, we need to identify, for any interval $[0, t]$, the minimum time allocated in the worst-case situation.

Supply bound function sbf(t):

minimum amount of time available in reservation R_k in every time interval of length t.

Example: Static time partition

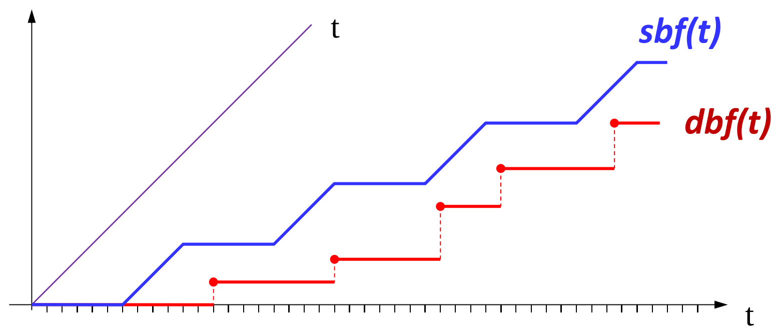
Example of reservation providing 4 units every 10 (bandwidth = 0.4).



Analysis under RR

Hence the Processor Demand Criterion can be reformulated as follows:

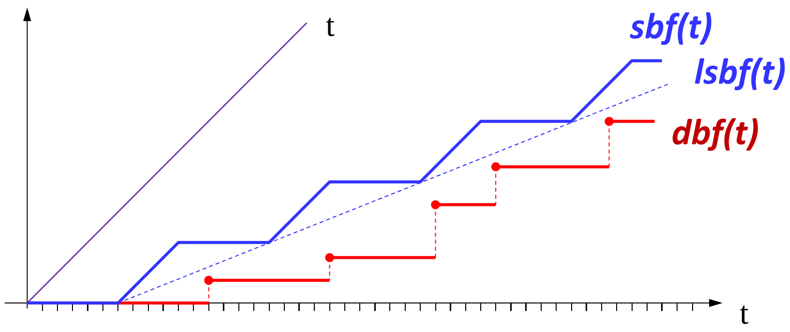
$$\forall t > 0, \quad dbf(t) \leq sbf(t)$$



Analysis under RR

A simpler sufficient test, can be derived by replacing ***sbf(t)*** with a lower bound, called linear supply bound function ***lsbf(t)***:

$$\forall t > 0, \quad dbf(t) \;\le\; lsbf(t)$$

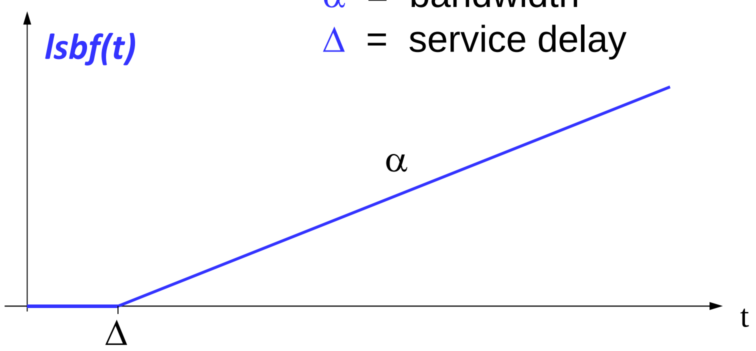


Supply bound function

A linear supply bound function has the following form:

$$lsbf(t) = \max\{0, \alpha(t - \Delta)\}$$

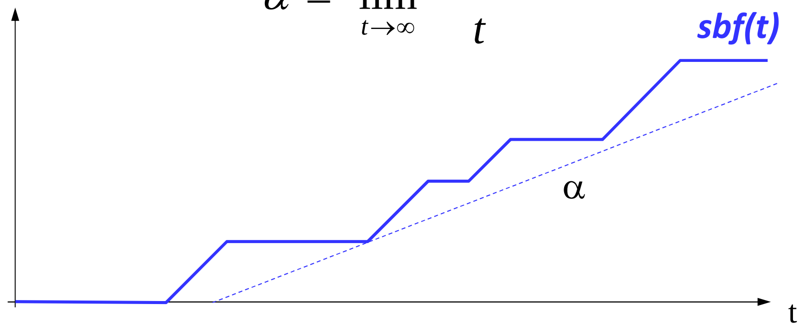
α = bandwidth
Δ = service delay



Deriving α and Δ

Given a generic supply function *sbf*(*t*), the bandwidth α is the equivalent slope computed for long intervals:

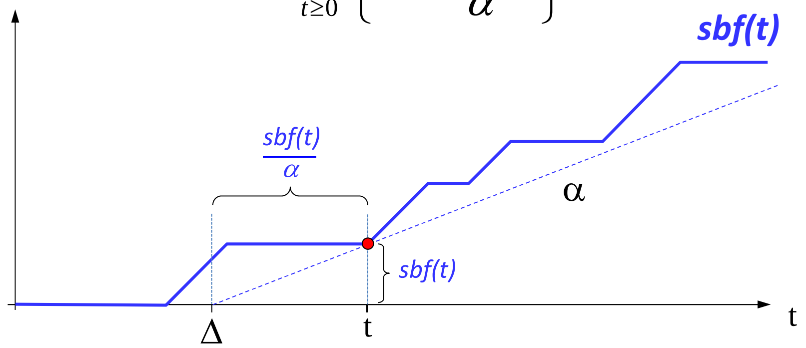
$$\alpha = \lim_{t \to \infty} \frac{sbf(t)}{t}$$



Deriving α and Δ

While the delay Δ is the highest intersection with the time axis of the line of slope α touching the ***sbf(t)***:

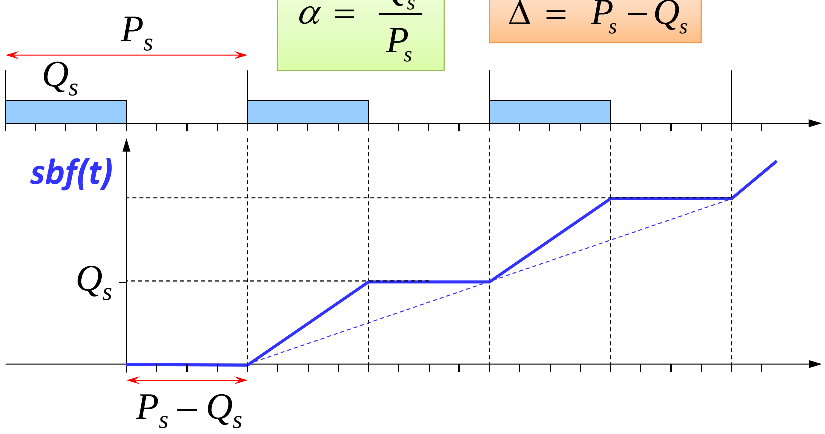
$$\Delta = \sup_{t \geq 0} \left\{ t - \frac{sbf(t)}{\alpha} \right\}$$



Example: Periodic Server

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

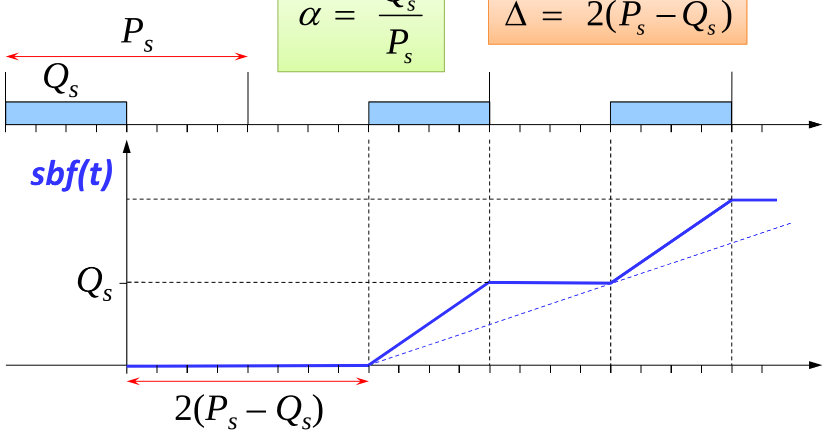
$$\alpha = \frac{Q_s}{P_s} \qquad \Delta = P_s - Q_s$$



Example: Periodic Server

For a periodic server with budget Q_s and period P_s running at unknown priority, we have:

$$\alpha = \frac{Q_s}{P_s} \qquad \Delta = 2(P_s - Q_s)$$



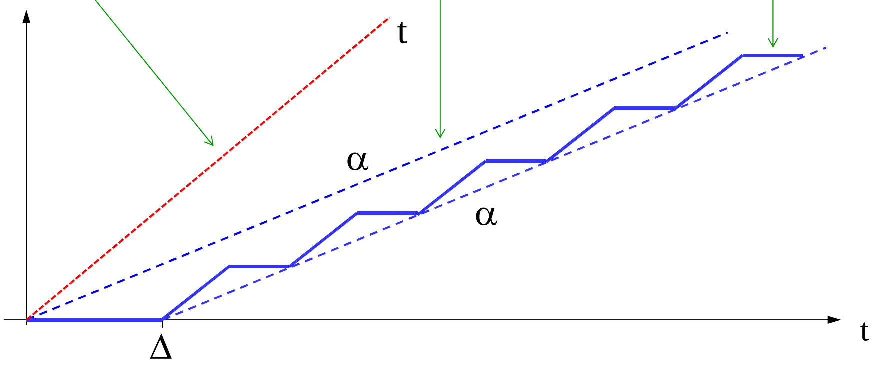
Observation

It is worth comparing the following supply functions:

Supply function of a full processor

Supply function of a fluid partition with bandwidth α

Supply function of a real partition with bandwidth α of a periodic server Q,P

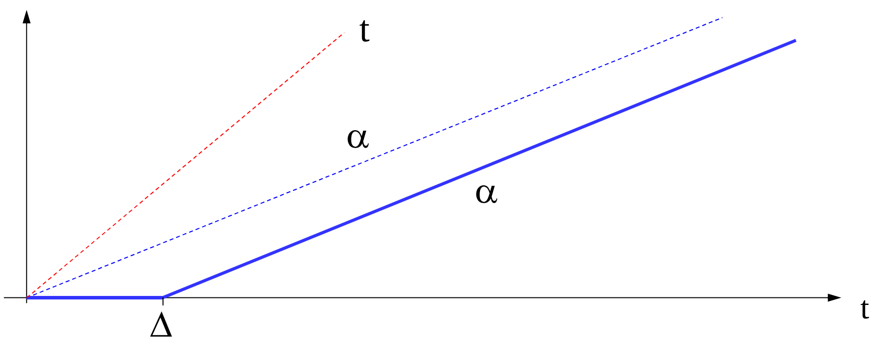


Observation

In a periodic server with bandwidth α, we have that:

$$\begin{cases} Q = \alpha P \\ \Delta = 2(P - Q) = 2P(1-\alpha) \end{cases}$$

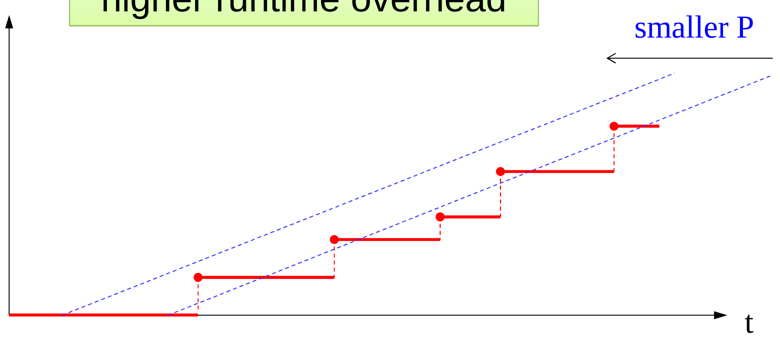
The delay Δ is proportional to the server period P



Observation

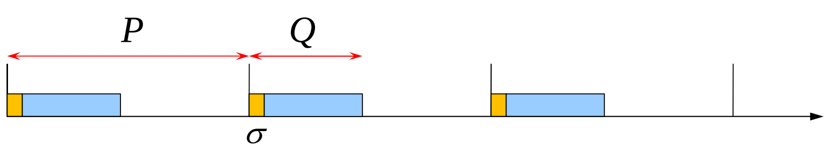
Note that, for a given bandwidth α, reducing P reduces the delay Δ and improves schedulability, tending to a fluid reservation, but …

smaller periods generates higher runtime overhead



Taking overhead into account

If σ is the context switch overhead, we have:



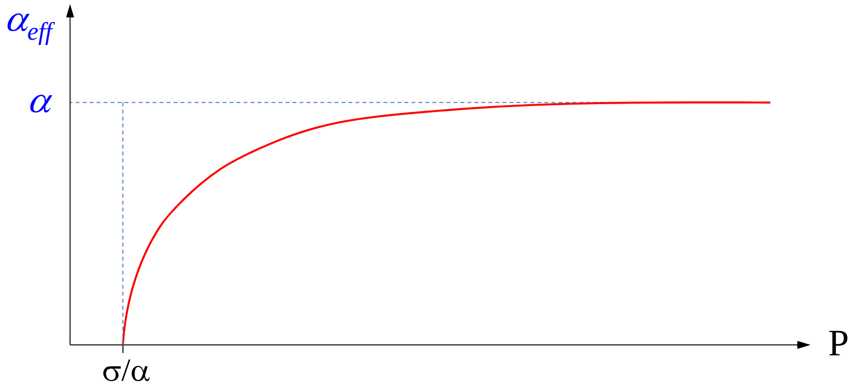
Allocated bandwidth: $\alpha = \frac{Q}{P}$ $\quad \Delta = 2P(1-\alpha)$

Effective bandwidth: $\alpha_{eff} = \frac{Q-\sigma}{P} = \alpha - \frac{\sigma}{P}$

$\alpha_{eff} > 0$ ➡ $P_{\min} = \frac{\sigma}{\alpha}$ ➡ $\Delta_{\min} = 2P_{\min}(1-\alpha) = 2\sigma\left(\frac{1-\alpha}{\alpha}\right)$

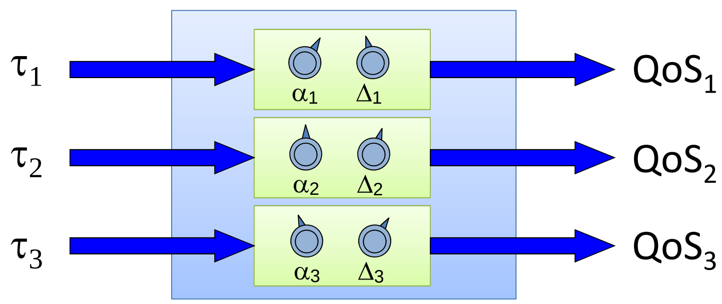
Effective bandwidth

Hence reducing the server period P reduces the delay Δ, but also reduces the effective bandwidth α_{eff} that can be exploited:

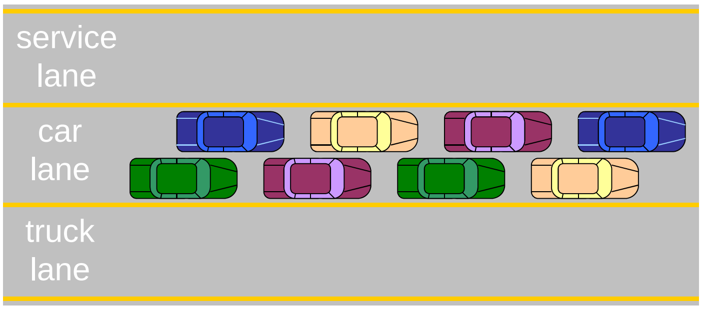


Reservation interface

Note that the (α, Δ) parameters offer an alternative interface, which is independent of the implementation mechanism (static partitions or Q-P server):



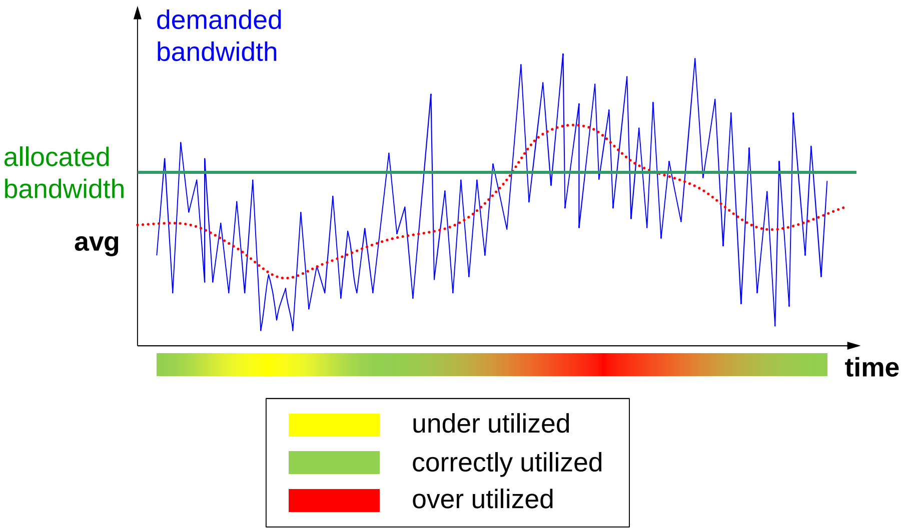
Avoiding resource wastes



A reservation can be wasted because of

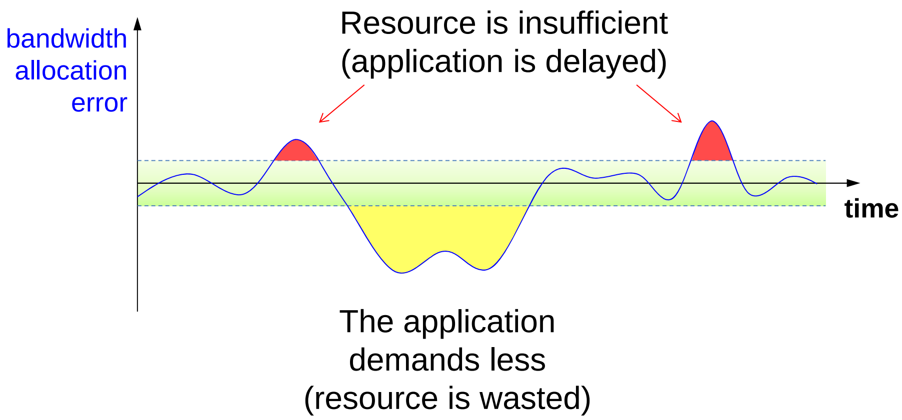
- occasional load reduction due to early terminations
- wrong bandwidth allocation at design time
- application changes due to environmental changes

Example of sporadic waste



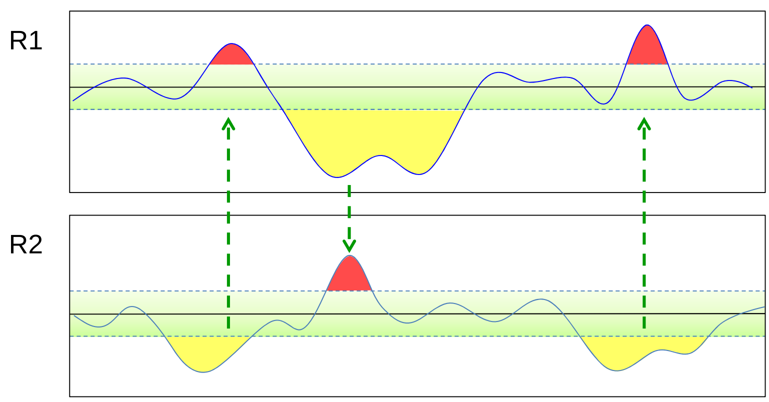
Bandwidth allocation error

Bandwidth allocation error = Demanded Bandwidth − Allocated Bandwidth

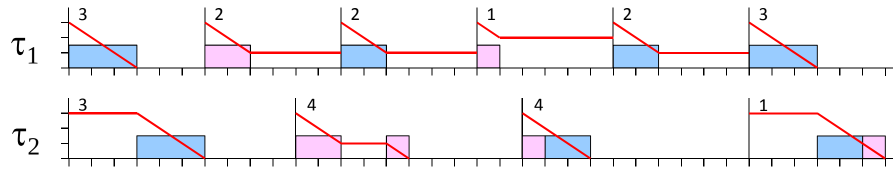


Resource Reclaiming

Unused bandwidth in a reservation can be used to satisfy extra occasional needs in another reservation:



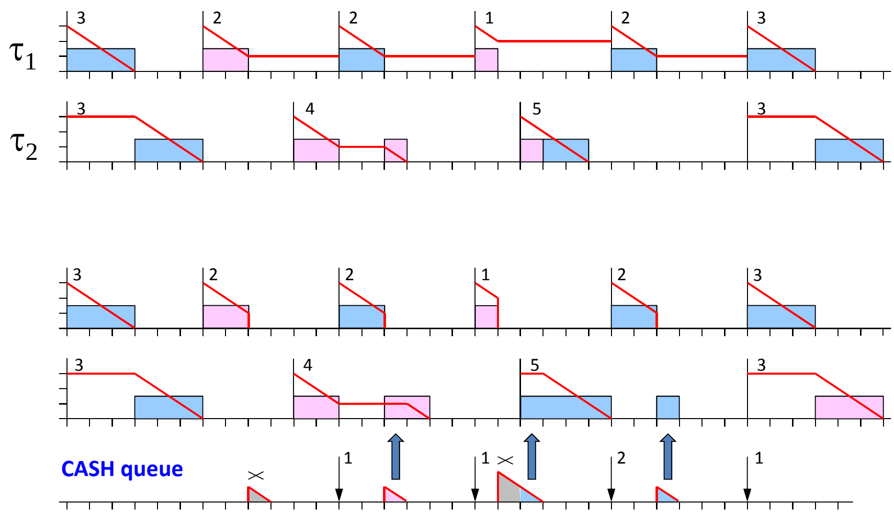
Budget reclaiming



CASH: Capacity Sharing algorithm

- When a job finishes and $q_s > 0$, the residual budget is put in a global queue of spare capacities (the CASH queue), with a deadline equal to the server deadline.
- A server first uses the capacity in the CASH queue with the earliest deadline $d_q \leq d_s$, otherwise q_s is used.
- Idle times consume the capacity in the CASH queue with the earliest deadline.

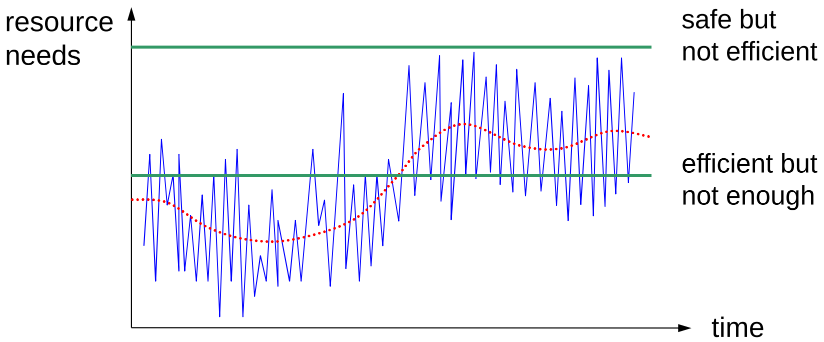
Capacity Sharing algorithm



Handling wrong allocations

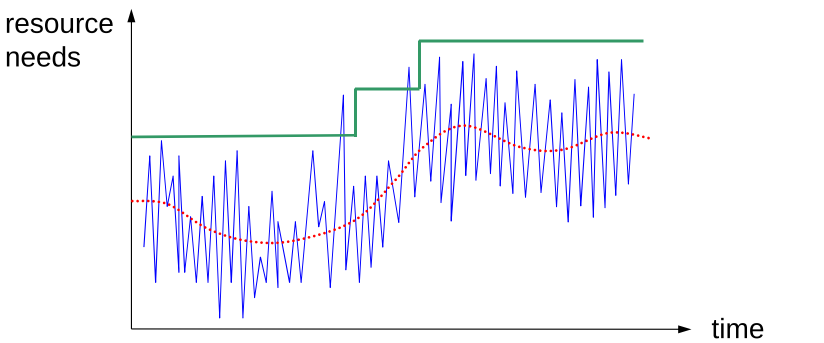
There are situations in which the reservation error is caused by a wrong bandwidth allocation:

➡ any reservation is not appropriate



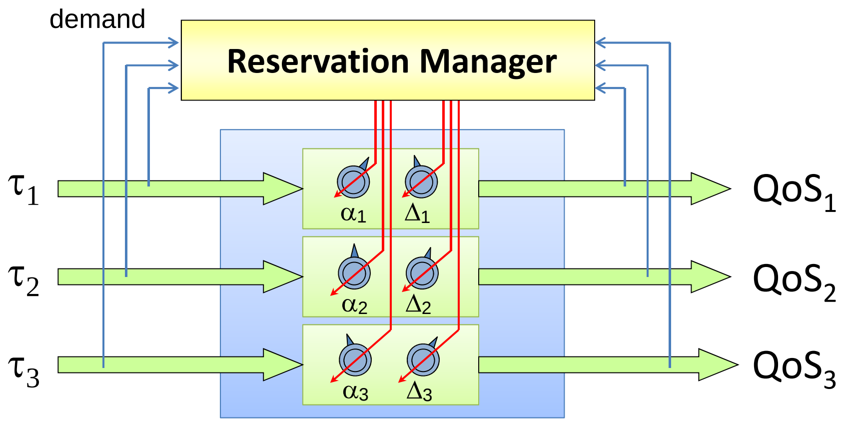
Need for adaptivity

In these cases, the only solution is to design the system to be adaptive so that reservations can be changed based on runtime requirements:



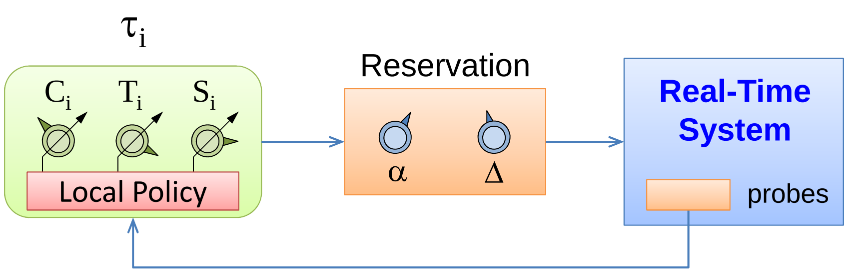
Adaptive QoS Management

Reservation parameters can be changed at run time by a Reservation Manager (Global adaptation):



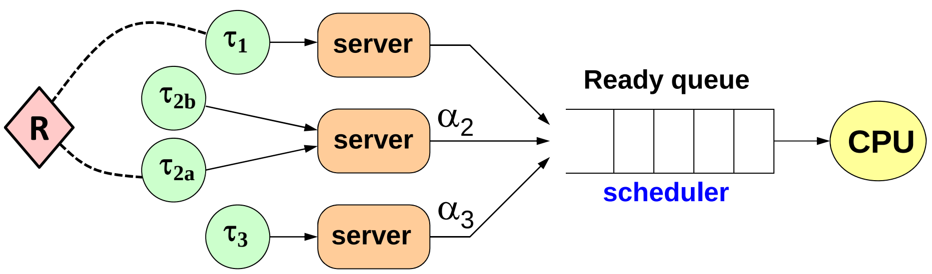
Local adaptation

A local adaptation approach is also possible for a task to comply with the assigned reservation:



Resource Reservation under Resource Sharing

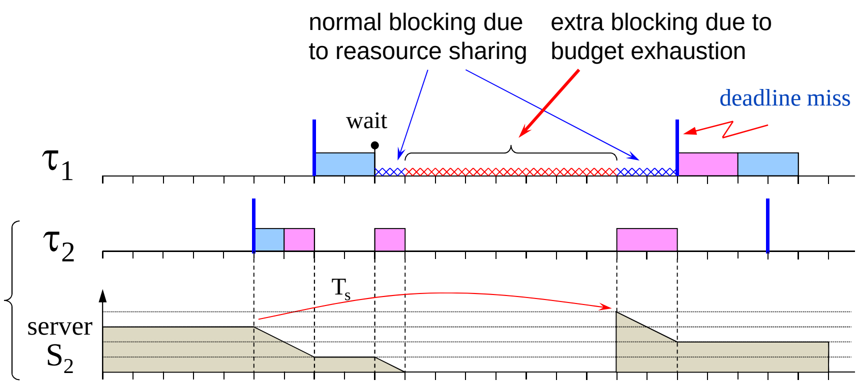
Shared Resources



Tasks are usually not independent as they share resources (e.g., data structures, peripherals, common memory areas).

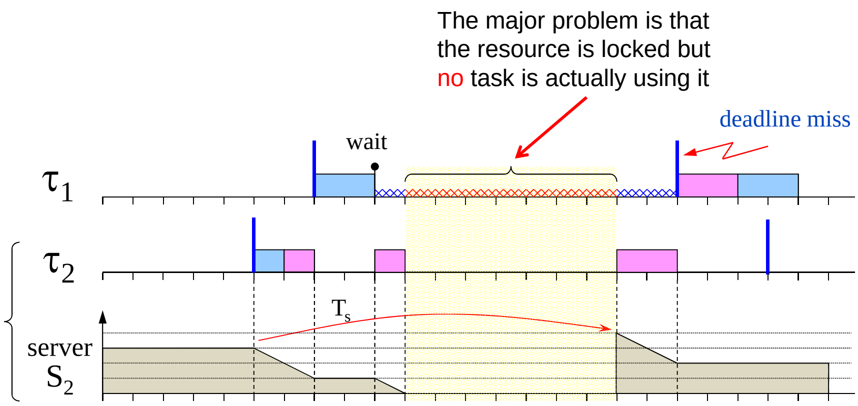
Problems with Reservations

- Resource sharing may break isolation:



Problems with Reservations

- Resource sharing may break isolation:



Possible approaches

Reactive approaches

Let the budget finishes and react with a given strategy:

- Overrun
  - Without payback
  - With payback
- Proxy execution (BWI)

Proactive approaches

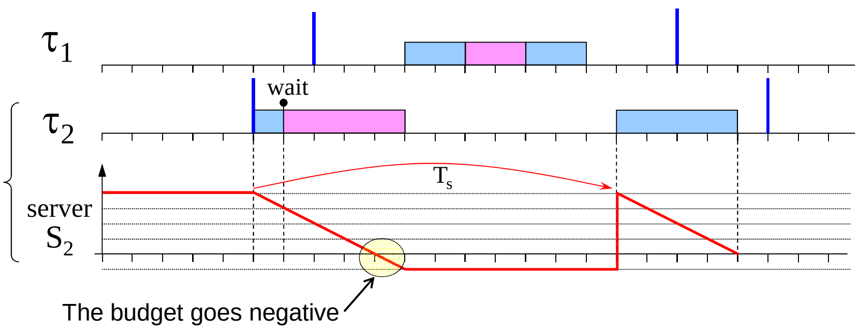
Prevent the budget to finish inside a critical section:

- Check and wait (SIRAP)
- Check and recharge (BROE)

Overrun without Payback

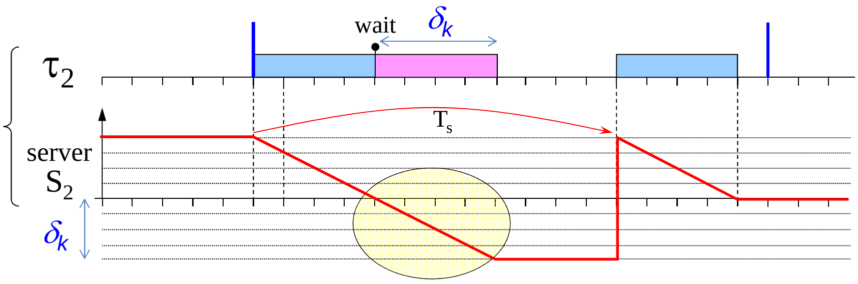
When the budget exhausts inside a critical section, do nothing.

Isolation is broken!



Overrun without Payback

Let δ_k be the length of the critical section to be entered.
In the worst-case the server consumes $Q_s + \delta_k$ budget units

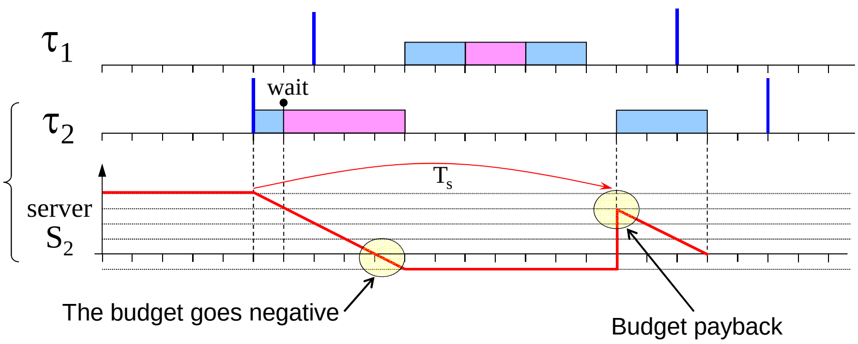


Overrun with Payback

When the budget exhausts inside a critical section, do nothing. Payback at the next budget replenishment.

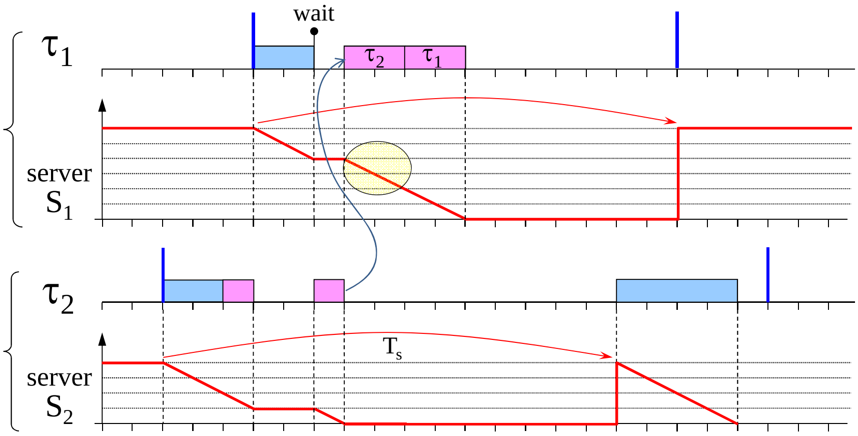
Isolation is broken!

Note that the worst-case bandwidth consumption does not change



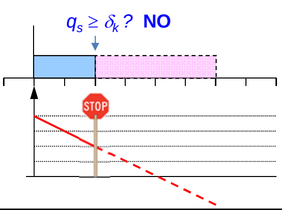
Proxy Execution

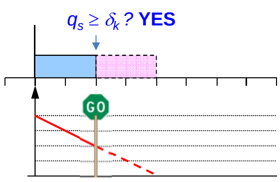
When the budget exhausts inside a critical section, inherit the bandwidth of another server



Proactive Approaches

- Let δ_k be the length of the critical section to be entered, and q_s be the budget of the server at the lock time;
- Proactive approaches are based on a budget check <u>before</u> locking the resource (i.e., $q_s \geq \delta_k$?);
- The scheduler requires the knowledge of δ_k at run-time.



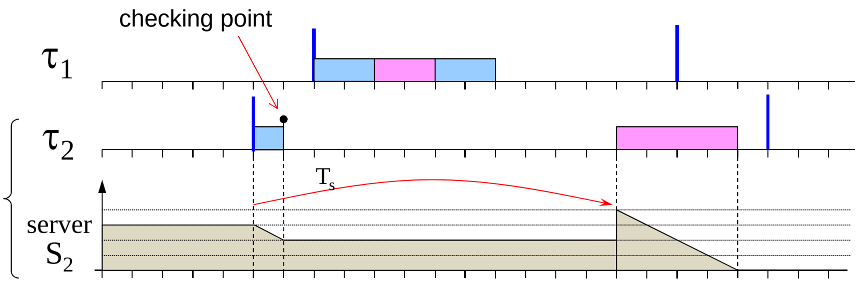


SIRAP

Check and wait

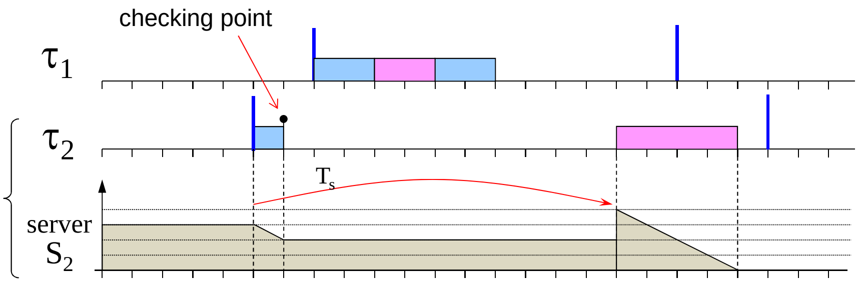
If ($q_s \geq \delta_k$) then enter, else wait for the next replenishment.

Note that off-line we must guarantee that $Q_s \geq \max\{\delta_k\}$.



SIRAP

- Penalizes the response-time of the task wishing to access the resource;
- Potentially inserts idle-time (unused budget).

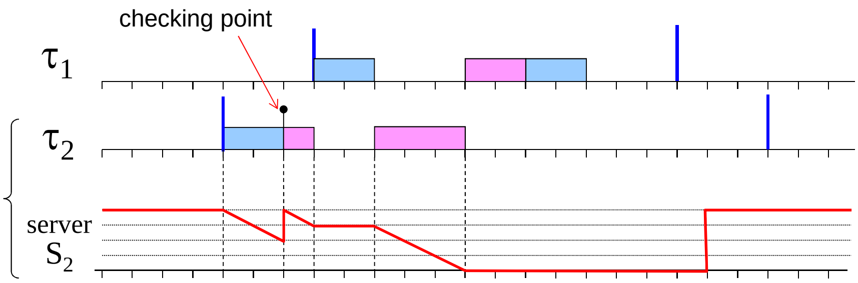


BROE

Check and recharge

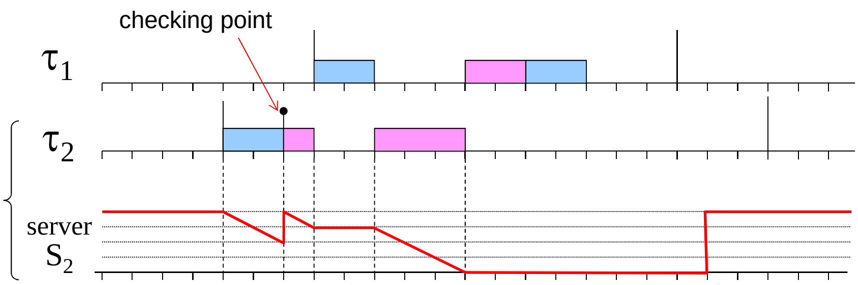
If ($q_s \geq \delta_k$) then enter, else recharge the budget at full value and proportionally postpone the server deadline.

Note that off-line we must guarantee that $Q_s \geq \max\{\delta_k\}$.



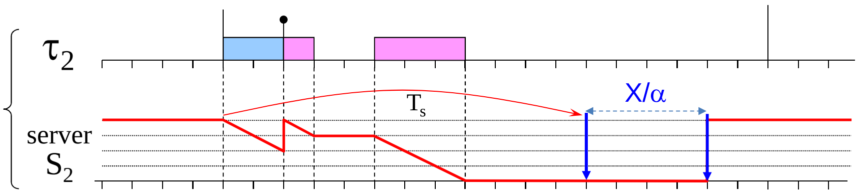
BROE

- Performs better than SIRAP in most situations;
- BROE works only with EDF-scheduled reservation servers.



BROE

- BROE is designed to guarantee a bounded-delay partition (α, Δ).
- A budget recharge of X time units reflects as a proportional deadline shift of X/α



BROE

Note that a deadline shift of X/α guarantees that the server never consumes a bandwidth higher than α, provided that

$$\alpha_i + \sum_{j \neq i} \alpha_j \leq 1$$

In fact, since $D = \frac{Q}{\alpha}$

The deadline increment ΔD that guarantees a bandwidth α with a budget $(Q + x)$ can be found by imposing:

$$\frac{Q + x}{D + \Delta D} = \alpha \qquad \text{thus:} \qquad \Delta D = D - \frac{Q + x}{\alpha} = D - \frac{Q}{\alpha} + \frac{x}{\alpha} = \frac{x}{\alpha}$$

BROE

BROE Design Goals

Overcome to the problem of budget depletion inside critical sections

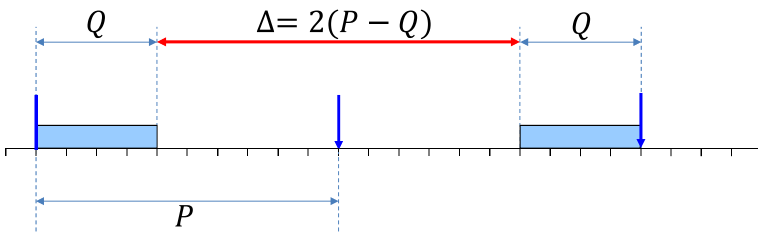
- Avoiding budget overruns;
- Ensuring bandwidth isolation (i.e., each server must consume no more than $\alpha = \frac{Q}{P}$ of the processor bandwidth);
- Guaranteeing a bounded-delay partition to the served tasks.

BROE: bandwidth guarantee

- When the budget is not enough to complete the critical section, BROE performs a full budget replenishment;
- To contain the server bandwidth, the budget replenishment must be reflected in a proportional deadline postponement
- To bound the service delay, the server must be suspended until a proper time.

BROE: bounded-delay

- To guarantee real-time workload executing upon a reservation server, the server must ensure a bounded-delay service

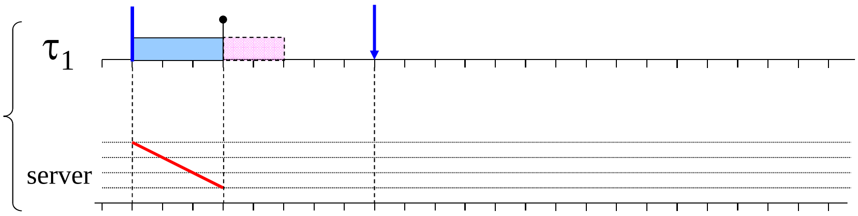


BROE: bounded-delay

- The budget replenishment and the corresponding deadline postponement can easily result in a violation of the worst-case delay $\Delta = 2(P - Q)$, if not properly handled!

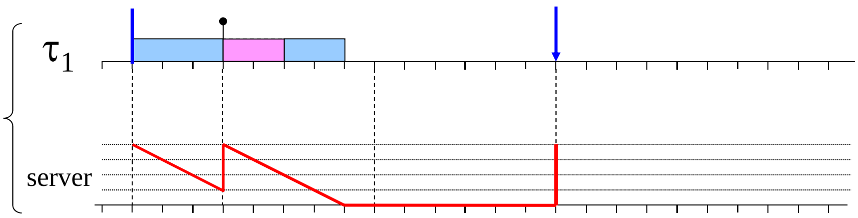
BROE: bounded-delay

- Consider a BROE server with Q=4 and P=8
- τ_1 accesses a resource having $\delta = 2$



BROE: bounded-delay

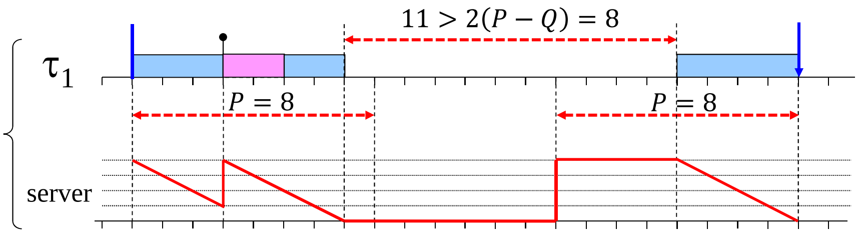
- Consider a BROE server with Q=4 and P=8
- τ_1 accesses a resource having $\delta = 2$



BROE: bounded-delay

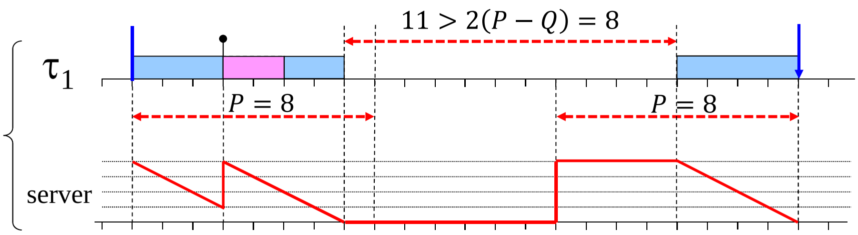
- Consider a BROE server with Q=4 and P=8
- τ_1 accesses a resource having $\delta = 2$
- The worst-case delay $\Delta = 2(P - Q)$ is violated!

The worst-case the delay can be potentially unbounded!



BROE: bounded-delay

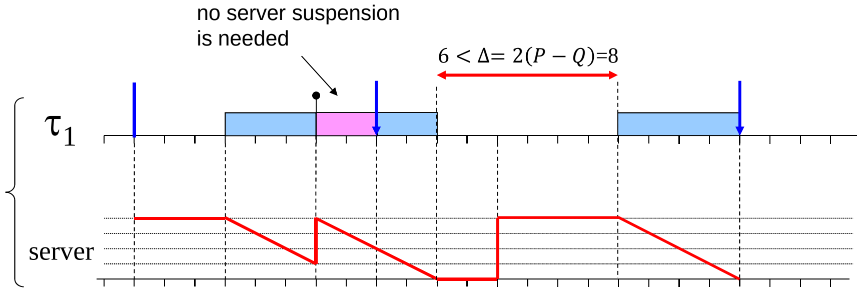
- How to solve this problem?
- The idea is to prevent the server to execute "*too earlier*" with respect to its deadline, after a budget replenishment



BROE: bounded-delay

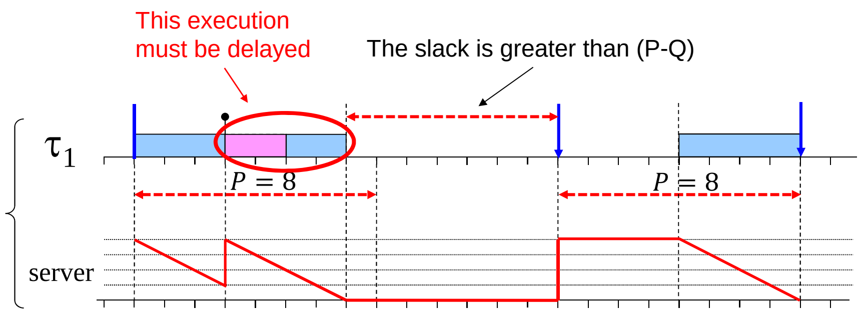
- If the server is “not executing too earlier”, it is not possible to violate the worst-case delay Δ

Depending on the execution state, BROE decides to suspend the server or not



BROE: bounded-delay

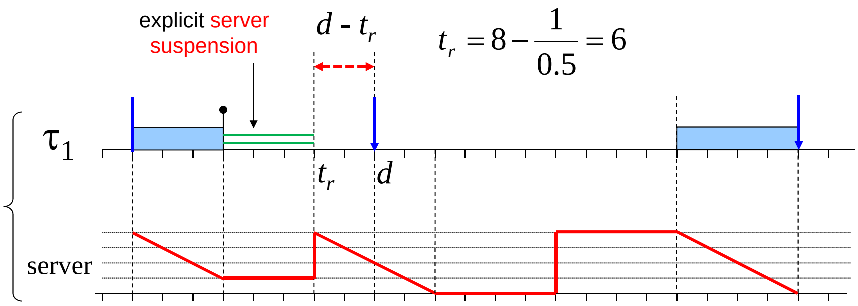
- For how long the server must be suspended?



BROE: bounded-delay

How to compute time t_r such that the bandwidth in $[t_r, d]$ is exactly α?

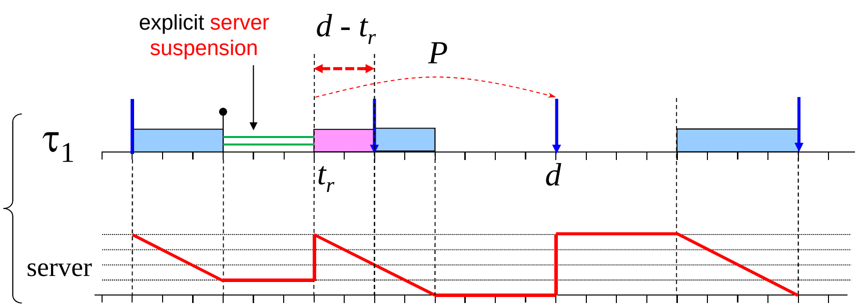
$$\frac{q(t)}{d - t_r} = \alpha \Rightarrow t_r = d - \frac{q(t)}{\alpha}$$



BROE: bounded-delay

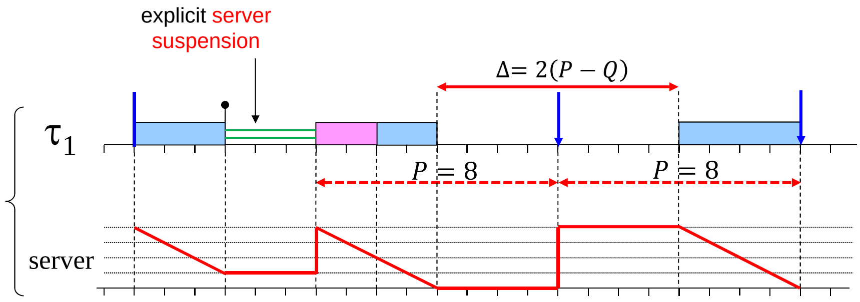
How to compute time t_r such that the bandwidth in $[t_r, d]$ is exactly α?

$$\frac{q(t)}{d - t_r} = \alpha \Rightarrow t_r = d - \frac{q(t)}{\alpha}$$



BROE: bounded-delay

Note that, thanks to the suspension, the worst-case service delay is still $\Delta = 2(P - Q)$:



BROE: goals

BROE Design Goals

Overcome to the problem of budget depletion inside critical sections

- Avoiding budget overruns;
- Ensuring bandwidth isolation (i.e., each server must consume no more than $\alpha = \frac{Q}{P}$ of the processor bandwidth);
- Guaranteeing a bounded-delay partition to the served tasks.

BROE: rules

BROE Resource Access Policy

Consider a BROE server having budget Q and period P. The current budget at time t is denoted as $q(t)$.

When a task wishes to access a resource R_k of length δ_k at time t:

- **if** $q(t) \geq \delta_k$, enter the critical section (there is enough budget);
- **else** compute a recharging time $t_r = d - \frac{q(t)}{\alpha}$
  - If $t < t_r$, the server is suspended until time t_r, the budget is replenished to Q and the deadline is shifted to $d = t_r + P$
  - Otherwise, the budget is immediately replenished to Q and $d = t_r + P$

BROE: constraints

- The BROE resource access policy can work **only with EDF** due to the proportional deadline shift. The support for FP is currently an open problem;
- To perform the budget check, BROE requires the specification of the **worst-case holding time** for the shared resources;
- BROE is intrinsically designed for the worst-case: the budget check can cause a scheduling decision that could be unnecessary.

BROE: recap

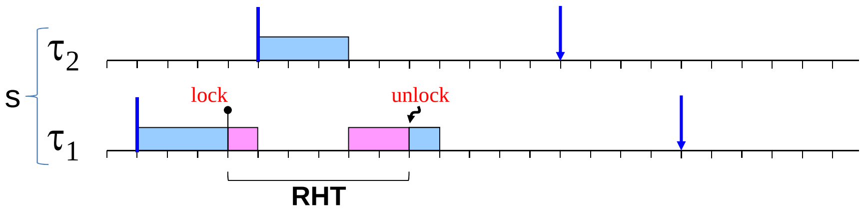
- The BROE server is a scheduling mechanism providing resource reservation including the support for shared resources
  - Hard reservation implementing the Hard-CBS algorithm;
  - Resource access protocol that guarantees both bandwidth isolation and bounded-delay to the served application.

Resource Holding Time

- In general, the BROE budget check has to be performed using the Resource Holding Time (RHT) of a shared resource;
- **RHT** = *budget consumed from the lock of a resource until its unlock*

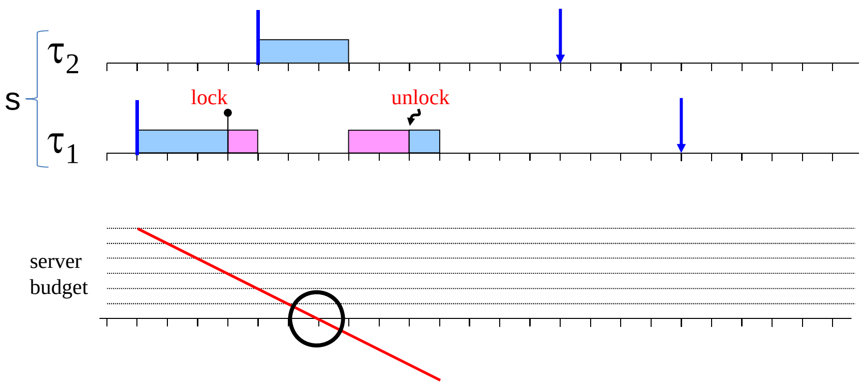
Resource Holding Time

- In general, the BROE budget check has to be performed using the Resource Holding Time (RHT) of a shared resource;
- **RHT** = *budget consumed from the lock of a resource until its unlock*



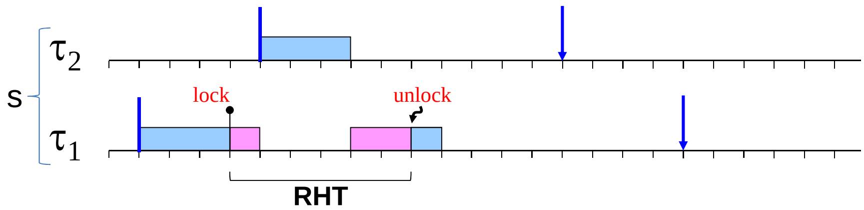
Resource Holding Time

- Interference from high-priority task has to be accounted in the budget consumed when a resource is locked



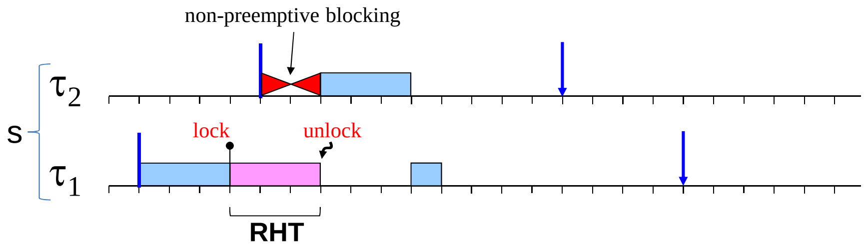
Resource Holding Time

- **RHT** = *Critical Section WCET* + *Worst-case Interference*
- The interference is caused by the task preemptions



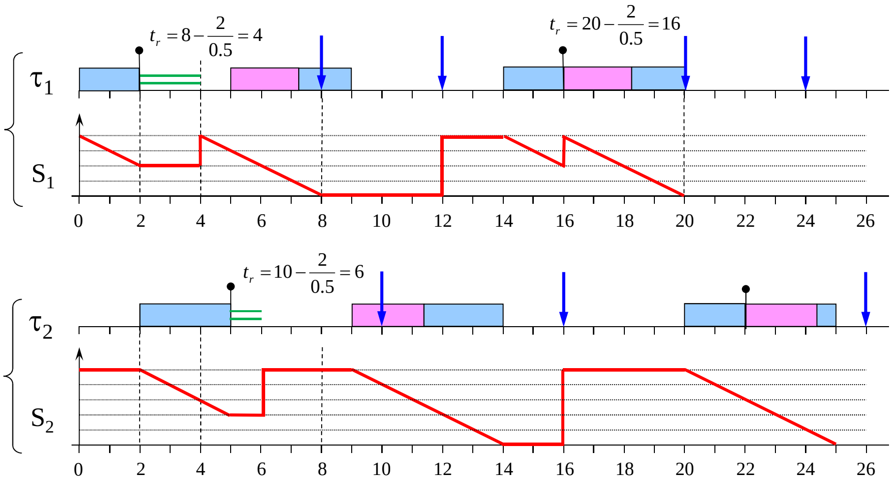
Resource Holding Time

- If resources are accessed in a non-preemptive manner, the RHT is equal to the worst-case critical section length;
- Trade-off: lower threshold for the budget check, but greater task blocking due to non-preemptive blocking



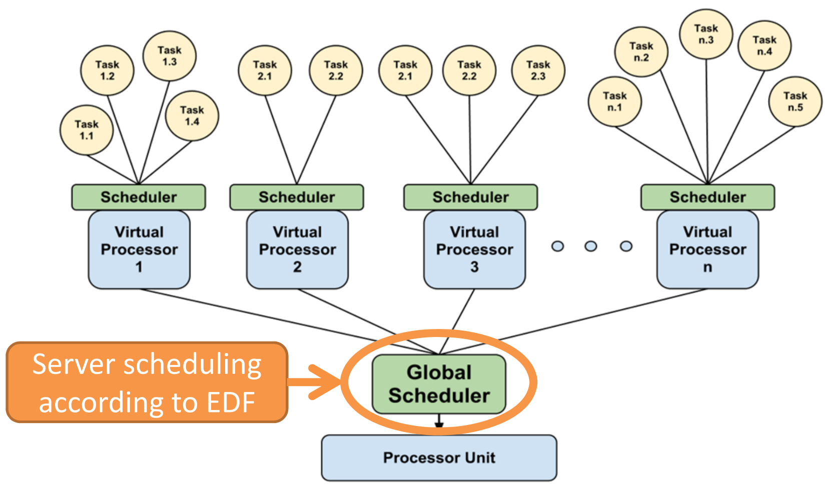
BROE: example

Consider 2 BROE servers: ($Q_1 = 4$, $P_1 = 8$) ($Q_2 = 5$, $P_2 = 10$)



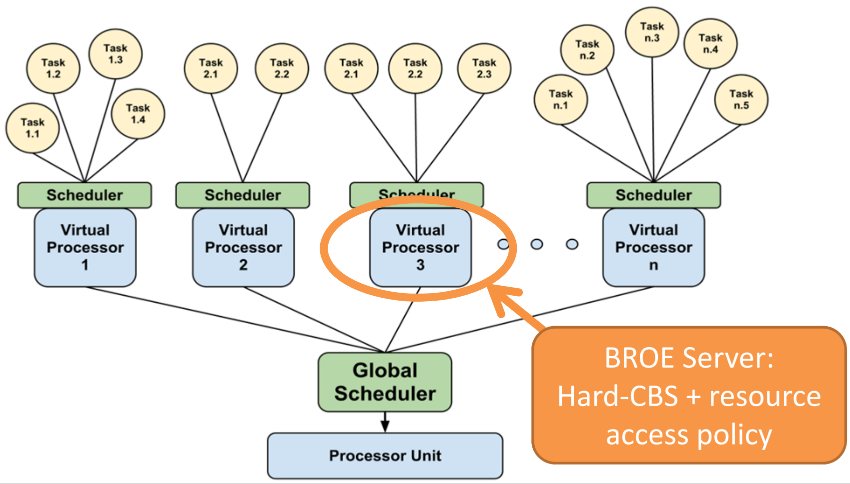
Implementation Issues

- **Goal**: Implementation of a two-level Hierarchical Scheduling Framework using the BROE algorithm.



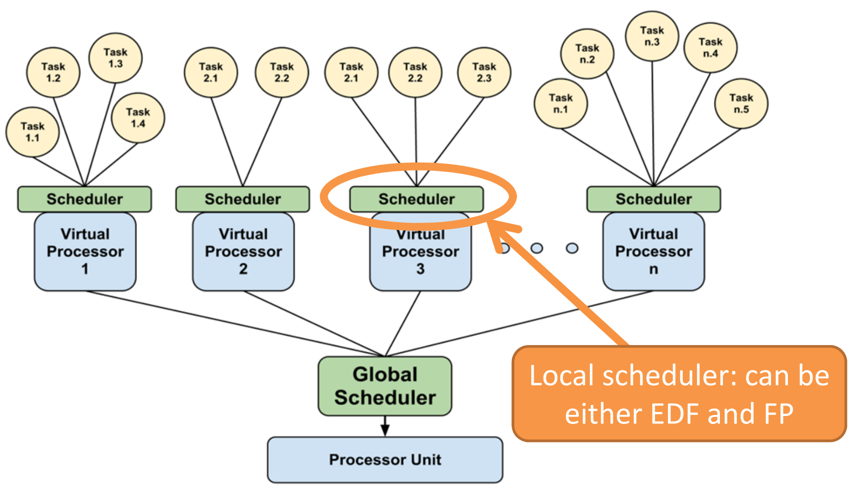
Implementation Issues

- **Goal**: Implementation of a two-level Hierarchical Scheduling Framework using the BROE algorithm.



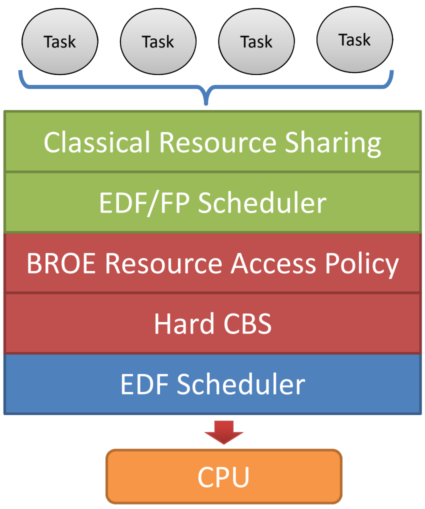
Implementation Issues

- **Goal**: Implementation of a two-level Hierarchical Scheduling Framework using the BROE algorithm.



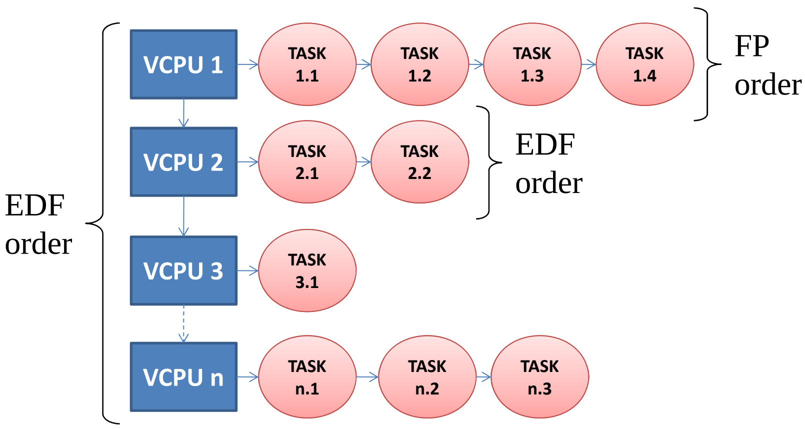
Implementation Issues

- Multi-layer scheduling infrastructure



Implementation Issues

- Ready queue structure



Implementation Issues

- OS with tick: the kernel comes into operation periodically, even if there are no scheduling events to be handled;
- OS tick-less: the kernel come into operation only when is needed, i.e., in correspondence of scheduling events.
- *Example*: budget management for reservation
- We look at tick-less RTOS implementation on small microcontrollers.

Implementation Issues

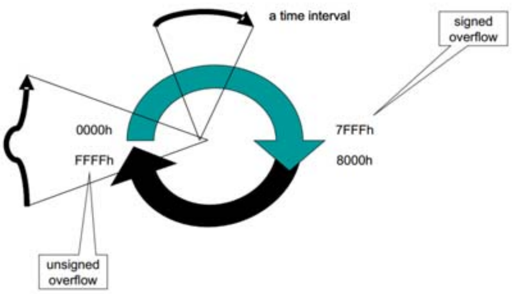
- EDF scheduling implementation needs a timing reference having both
  - High-resolution;
  - Long life-time (to handle absolute deadlines).

It Requires 64-bit data structure for time representation

- Dealing with 64-bit data structures in small microcontrollers imposes a significant overhead in the scheduler implementation.

Implementation Issues

- Circular timer: avoid an absolute timing reference. The notion of time is relative with respect to a free running timer.
- Let T the lifetime of the free running timer.
- It is possible to handle temporal events having a maximum spread of T/2.



Implementation Issues

- Consider two events e_1 and e_2.
- Let $t(e_1)$ be the absolute time of an event, and $r(e_1)$ its relative representation by using the circular timer.
- To compare two events having $|t(e_1) - t(e_2)| < T/2$
  - If $(r(e_1) - r(e_2)) > 0$ then $t(e_1) > t(e_2)$
  - If $(r(e_1) - r(e_2)) < 0$ then $t(e_1) < t(e_2)$
  - If $(r(e_1) - r(e_2)) == 0$ then $t(e_1) = t(e_2)$

Implementation Issues

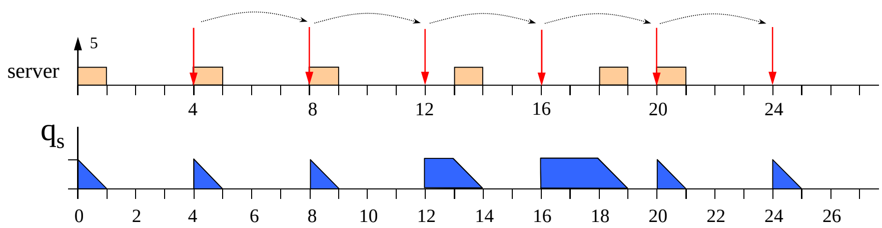
- Warning: a relative representation becomes inconsistent after T/2!
- Inactive servers: It is necessary to perform a periodic check of inconsistent deadlines;
- A special timer has to be reserved for that job.

The implementation of EDF requires 2 timers:

- Free running timer
- Periodic timer for deadline consistency

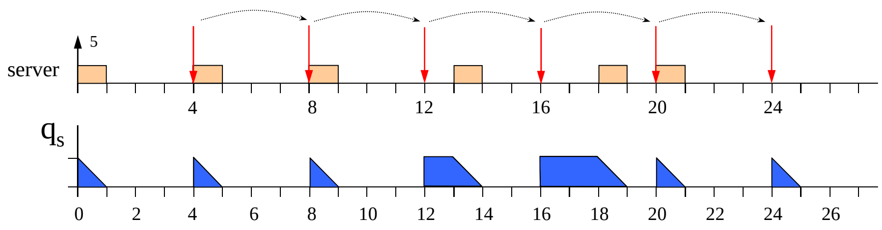
Implementation Issues

- **Hard-CBS Server**: its implementation requires to manage two main operations
  - Budget enforcement;
  - Budget recharge.



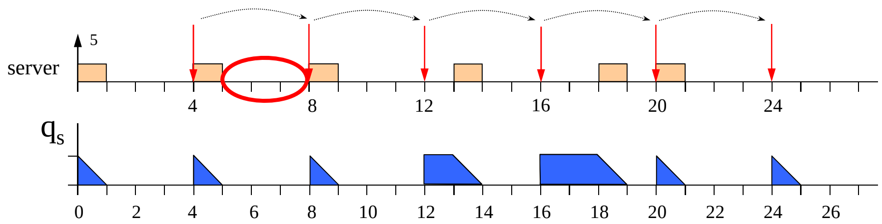
Implementation Issues

- **Budget enforcement**: when then server starts to execute at time t, set up an one-shot timer with the current budget $q(t)$.
- If a preemption occurs, the timer is reconfigured; otherwise, it will fire to notify a budget exhaustion.



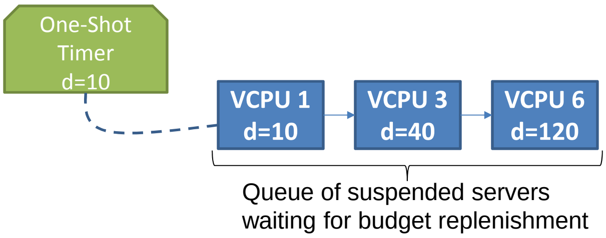
Implementation Issues

- **Budget recharge**: when a server exhaust its budget, it has to be *suspended* until its deadline, where the budget will be recharged.
- A deadline-ordered queue of suspended server has to be provided. Another one-shot timer triggers the budget recharge event for the first server in the queue.



Implementation Issues

- **Budget recharge**: when a server exhaust its budget, it has to be *suspended* until its deadline, where the budget will be recharged.
- A deadline-ordered queue of suspended server has to be provided. Another one-shot timer triggers the budget recharge event for the first server in the queue.



Implementation Issues

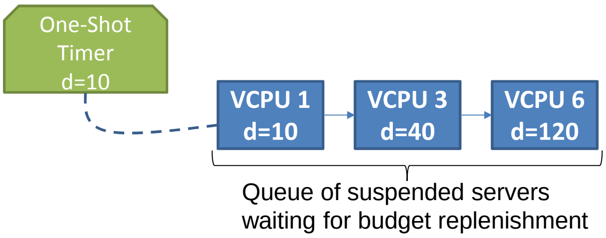
- **Hard-CBS Server**: its implementation requires to manage two main operations
  - Budget enforcement;
  - Budget recharge.

The implementation of the Hard CBS requires 2 timers:

- One-shot timer for budget enforcement
- One-shot timer for budget recharge

Implementation Issues

- **BROE server suspension**: can be implemented exploiting the budget recharge queue
  - *“If $t < t_r$, the server is suspended until time t_r”*



Implementation Issues

- **BROE server suspension**: can be implemented exploiting the budget recharge queue
  - *“If $t < t_r$, the server is suspended until time t_r”*