Inovace tohoto kurzu byla v roce 2011/12 podpořena projektem CZ.2.17/3.1.00/33274 financovaným Evropským sociálním fondem a Magistrátem hl. m. Prahy.

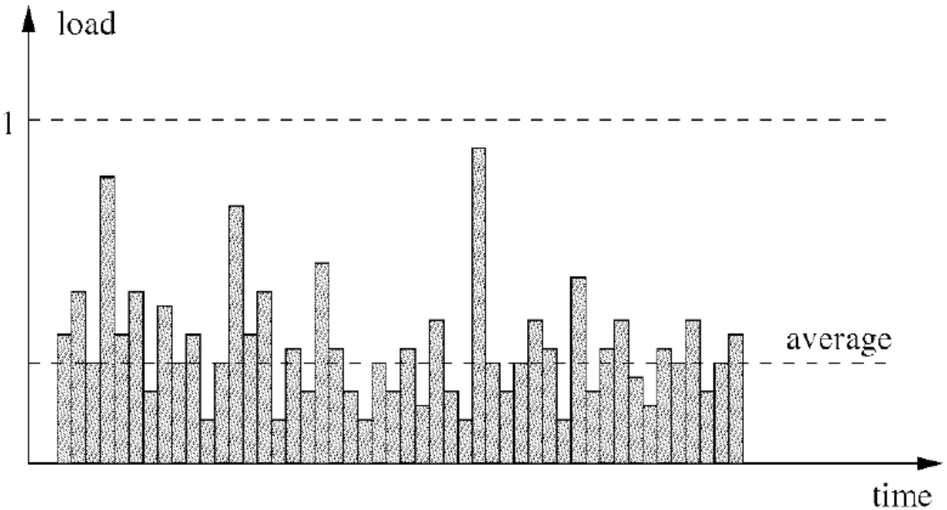
Evropský sociální fond
Praha & EU: Investujeme do vaší budoucnosti

Embedded and Real-Time Systems

Soft Real-Time Systems

Periodic tasks

- Many tasks do not require hard-real time approach
  - When deadline overruns are infrequent and/or acceptable
  - Hard-real time scheduling may lead to resource waste



Underloaded system with low average resource usage

System with transient overloads but high average resource usage

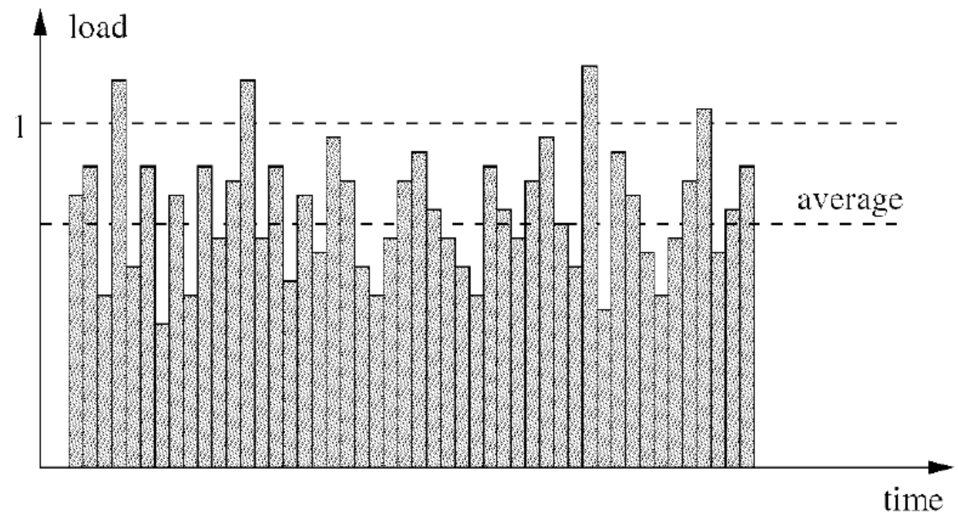


Figure taken from Buttazzo, G. et al: Soft Real-Time Systems

Characterizing soft real-time tasks

- Based on utility functions that states the value of the result based on the time it is delivered

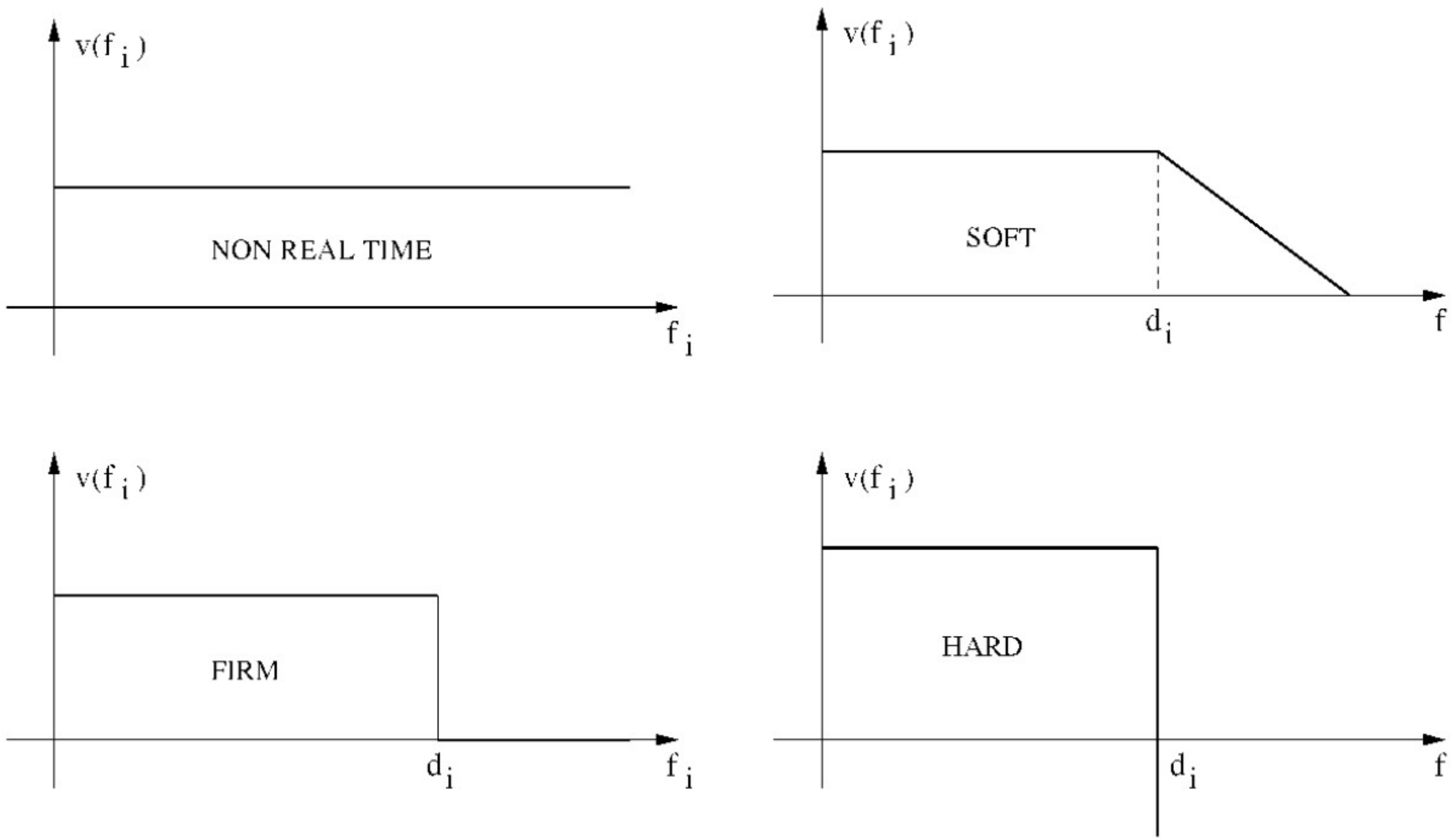


Figure taken from Buttazzo, G. et al: Soft Real-Time Systems

Soft real-time systems

- We do not optimize for deadlines, but for the delivered value

- Cumulative value: $$\Gamma_A = \sum_{i=1}^{n} v(f_i)$$

Overload conditions

- The system may thus experience load greater than 1

- Load defined as:

$$\rho(t) = \max_i \rho_i(t)$$

$$\rho_i(t) = \frac{\sum_{d_k \le d_i} c_k(t)}{d_i - t}$$

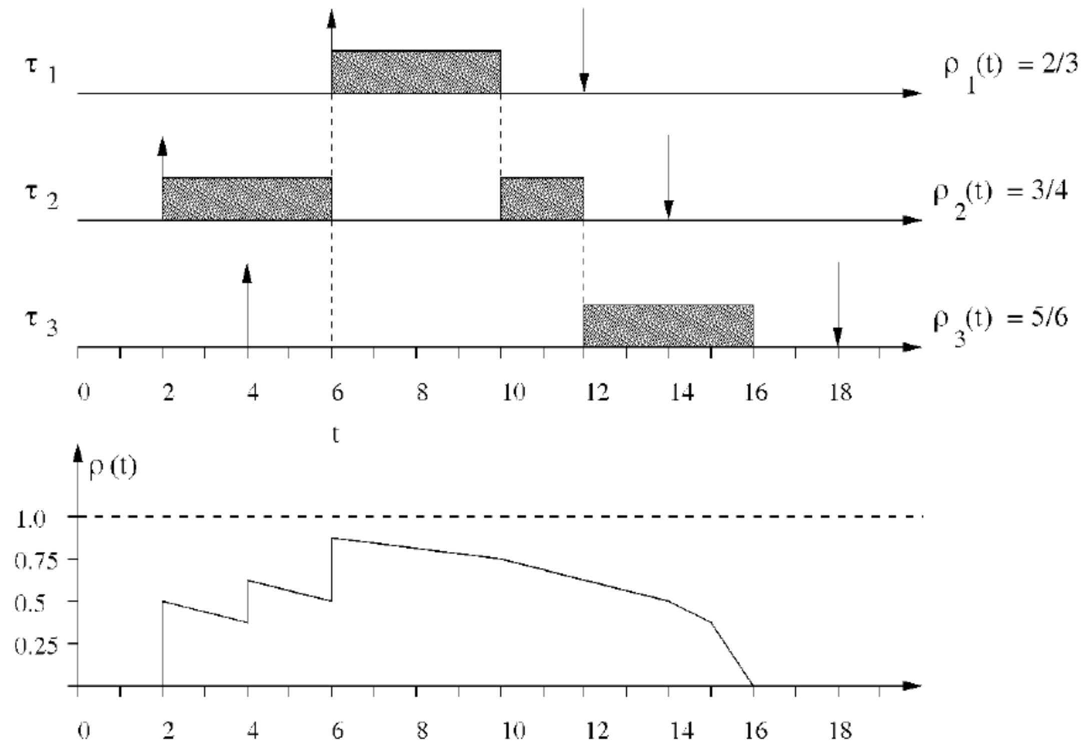


Figure taken from Buttazzo, G. et al: Soft Real-Time Systems

Handling overloads

- Classical algorithms do not cope well with overloads (e.g. EDF)

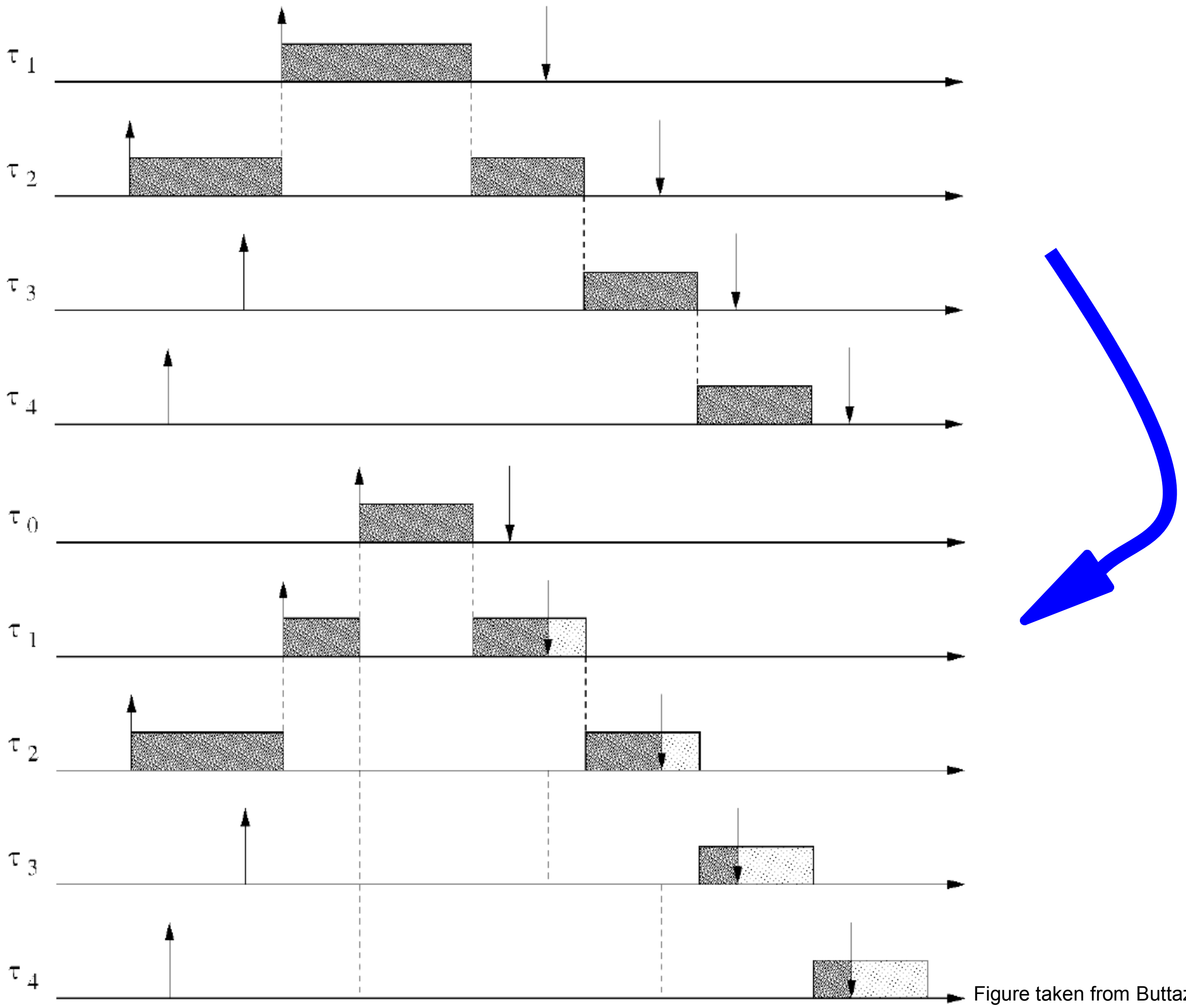
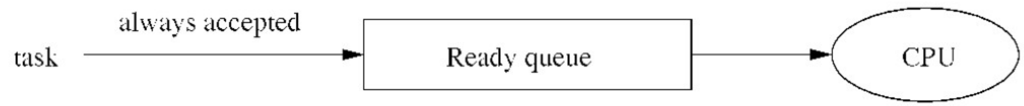


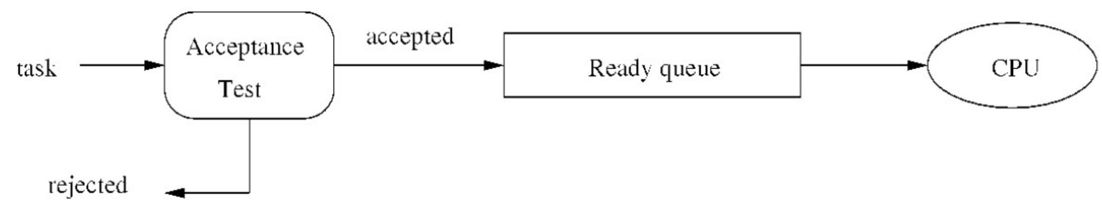
Figure taken from Buttazzo, G. et al: Soft Real-Time Systems

Handling overloads

- Strategies to handle overloads
  - Best effort scheduling
    - No prediction for overload conditions



  - Simple Admission control
    - Incoming task may be rejected



  - Robust scheduling
    - Incoming task may cause rejection of existing tasks

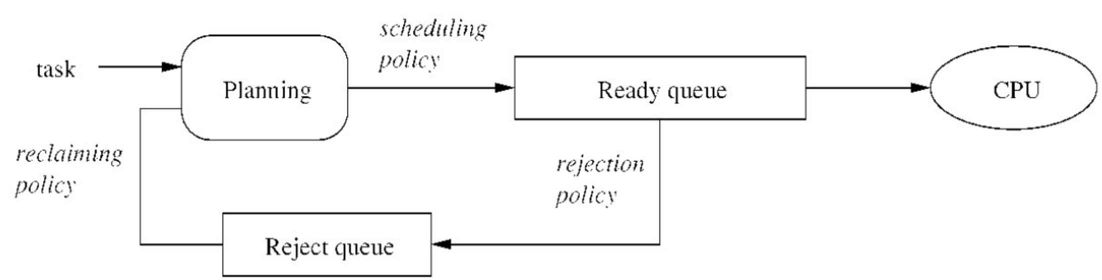


Figure taken from Buttazzo, G. et al: Soft Real-Time Systems

Robust Earliest Deadline (RED)

- Robust scheduling algorithm
- Each task has
  - worst-case execution time (C_i)
  - relative deadline (D_i)
  - deadline tolerance (M_i)
  - importance value (V_i)

- Tasks are scheduled according to deadlines and accepted based on secondary deadlines (i.e. increased by deadline tolerance)

Robust Earliest Deadline

- RED computes residual laxities $L_i = d_i - f_i$
  This can be computed in $O(n)$

- Then computes maximum exceeding time:

$$E_{max} = \max_i(E_i)$$

$$E_i = \max(0, -(L_i + M_i))$$

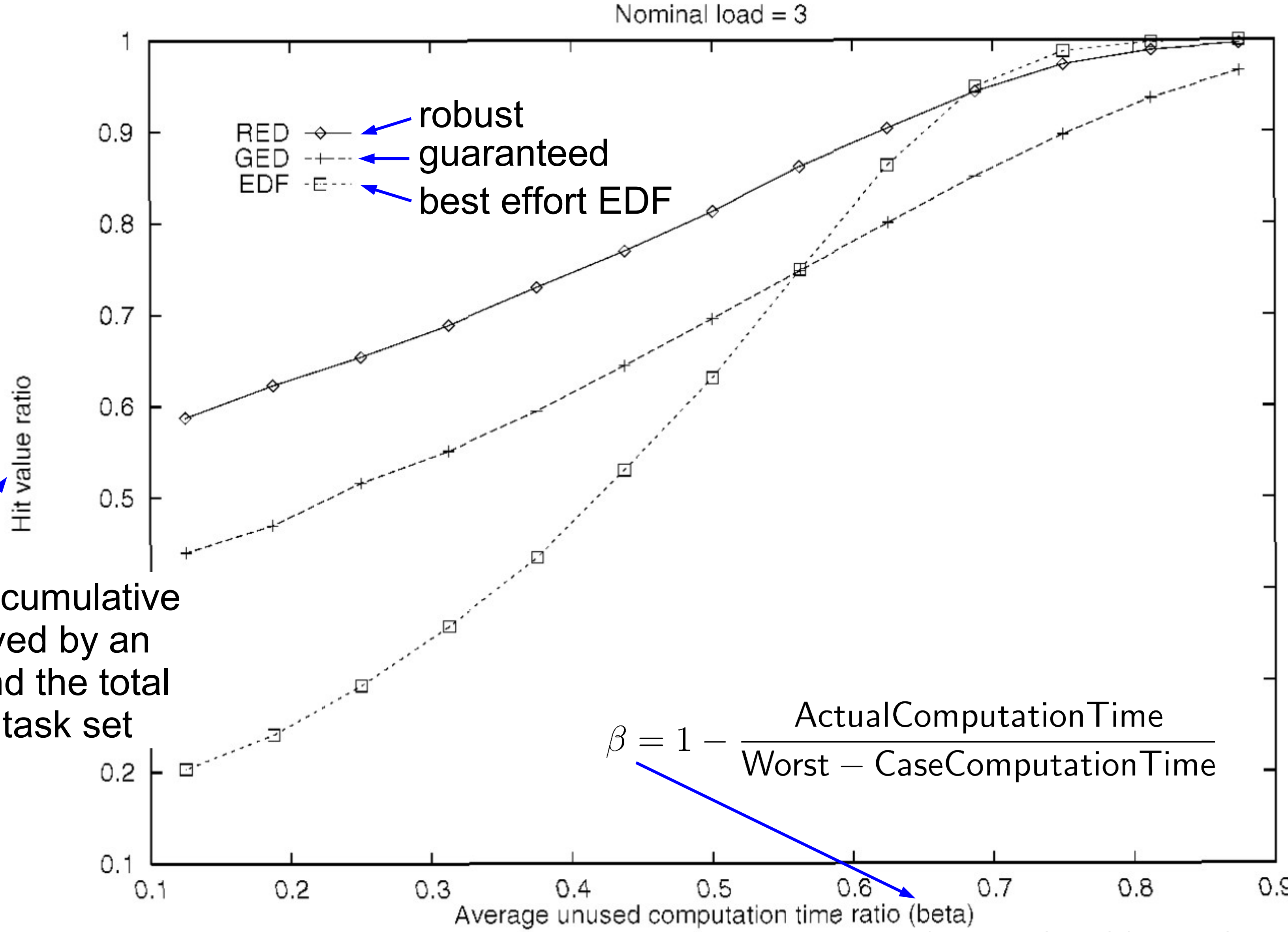
- This gives a clue how much time is needed. Then RED selectes some tasks (e. g. least valued the rejection of can solve the overload) and rejects them.

RED – Resource reclaiming

- RED keeps the rejected tasks in a special queue and re-accepts them when some task finishes before its WCET

- Only tasks with positive laxity are re-accepted
- Those with negative laxity are discarded from the queue

RED – Performance evaluation



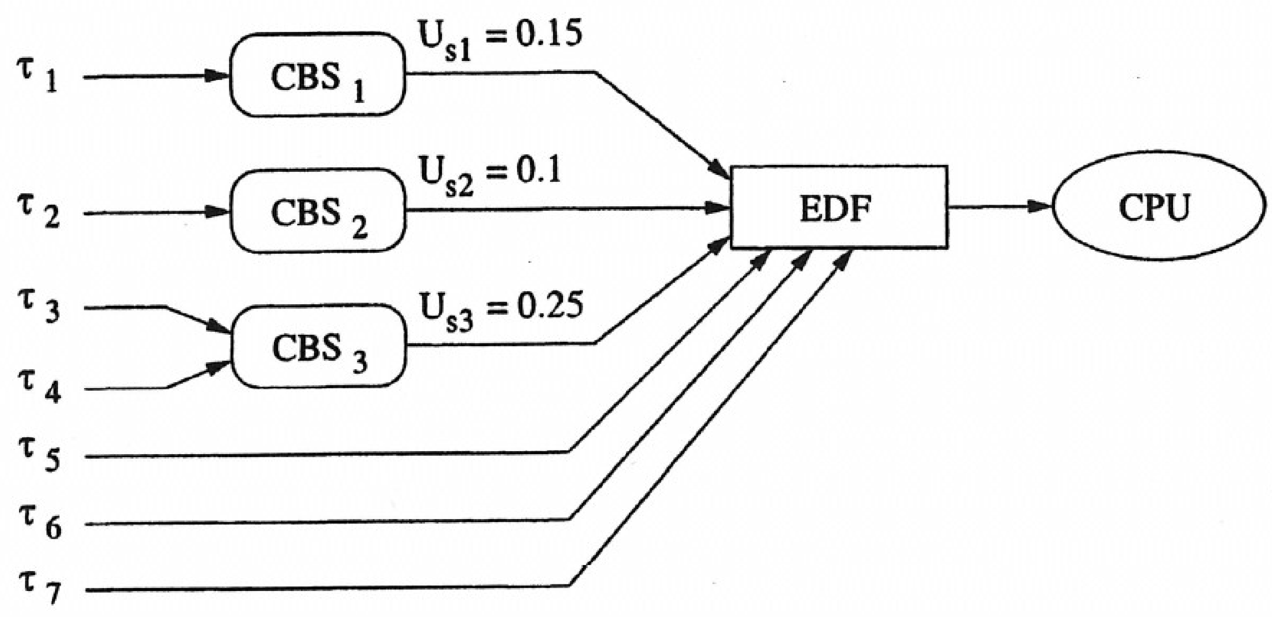
Ratio of the cumulative value achieved by an algorithm and the total value of the task set

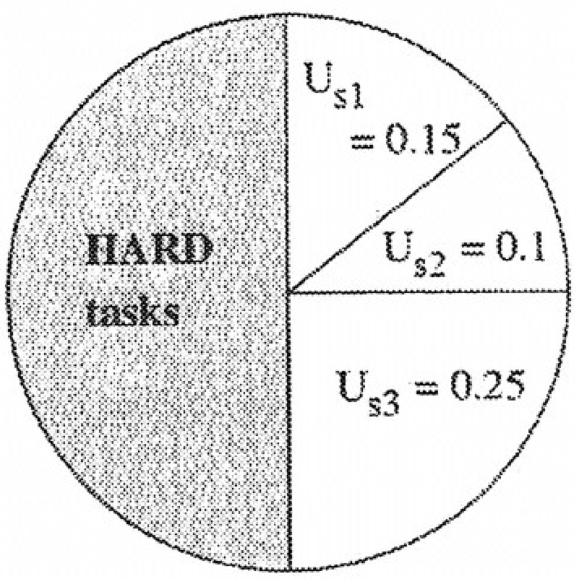
$$\beta = 1 - \frac{\text{ActualComputationTime}}{\text{Worst} - \text{CaseComputationTime}}$$

Figure taken from Buttazzo, G. et al: Soft Real-Time Systems

Enforcing temporal protection

- A simple way of enforcing temporal protection is to use constant bandwidth servers for tasks, which are allowed to overrun





Performance degradation methods

- In this approach, overloads are not solved by rejecting tasks but by degradation of tasks

- Service adaptation
  - Load is controlled by reducing the computation times

- Job skipping
  - Load is reduced by aborting entire task instances

- Period adaptation
  - Load is reduced by relaxing timing constraints

Service adaptation

- Each task has two parts
  - Mandatory subtask M_i
    - Must be completed
  - Optional subtask O_i
    - Comes after the mandatory part
    - May be aborted
    - Corresponds to precising the results, etc.

Imprecise schedule

Task	r_i	d_i	C_i	m_i	o_i
τ_1	0	6	4	2	2
τ_2	2	7	4	1	3
τ_3	4	10	5	2	3
τ_4	12	15	3	1	2
τ_5	6	20	8	5	3

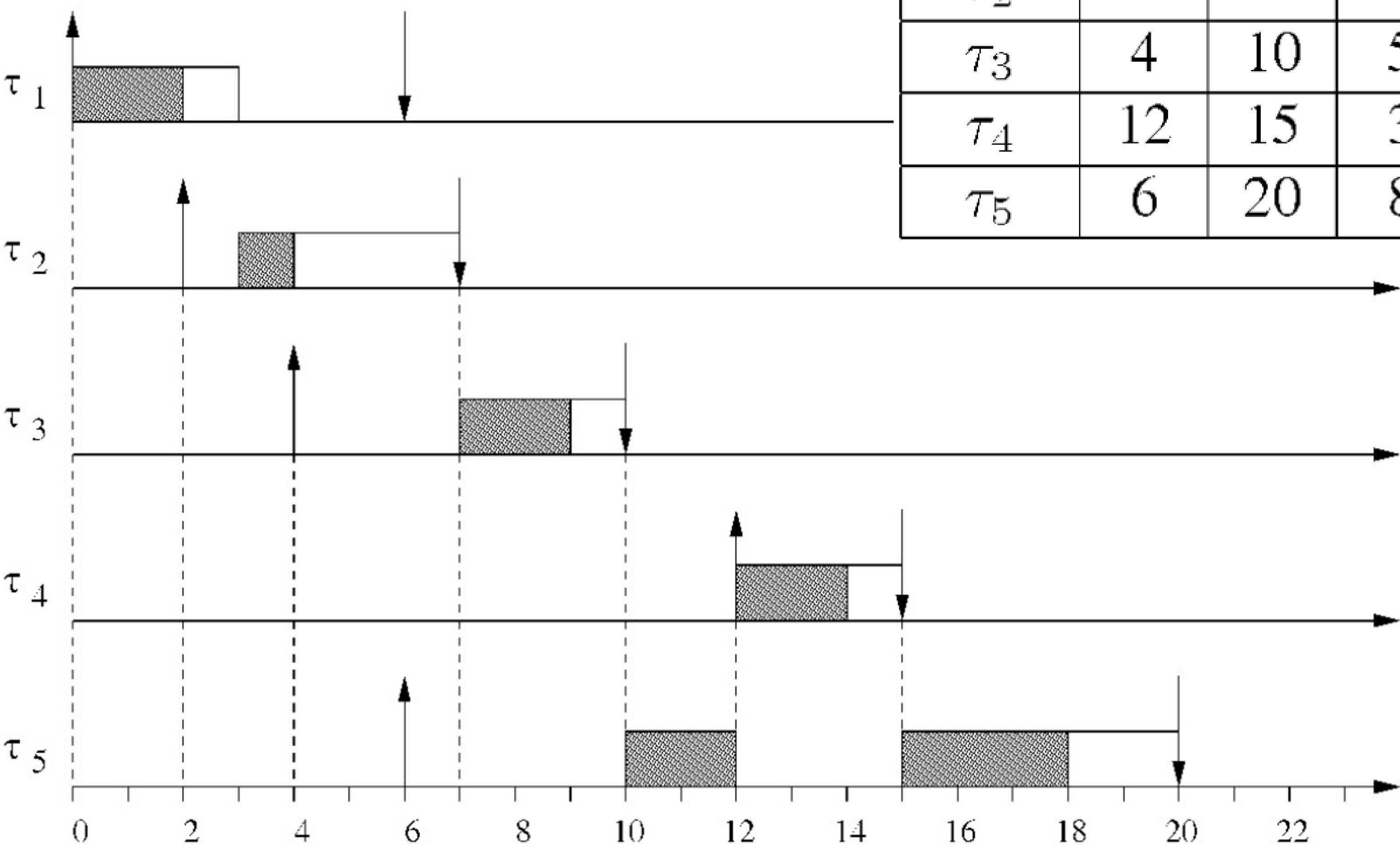


Figure taken from Buttazzo, G. et al: Soft Real-Time Systems

Service adaptation

- Hard real-time tasks have optional part empty
- We can define the error: $\epsilon_i = o_i - \sigma_i$
  - σ_i is the time really allocated to subtask O_i
- and average error: $\bar{\epsilon} = \sum_{i=1}^{n} w_i \epsilon_i$
  - w_i is the importance of the task
- If tasks cannot be degraded in this way, it is still possible to have several implementations of a task from which, the scheduler may choose

Job skipping

- Each task has a skip parameter
  - Tells after how many instances one may skip one task
  - Skip parameter of infinity means hard task

Task	Task1	Task2	Task3
Computation	1	2	5
Period	3	4	12
Skip Parameter	4	3	∞
U_p	1.25		

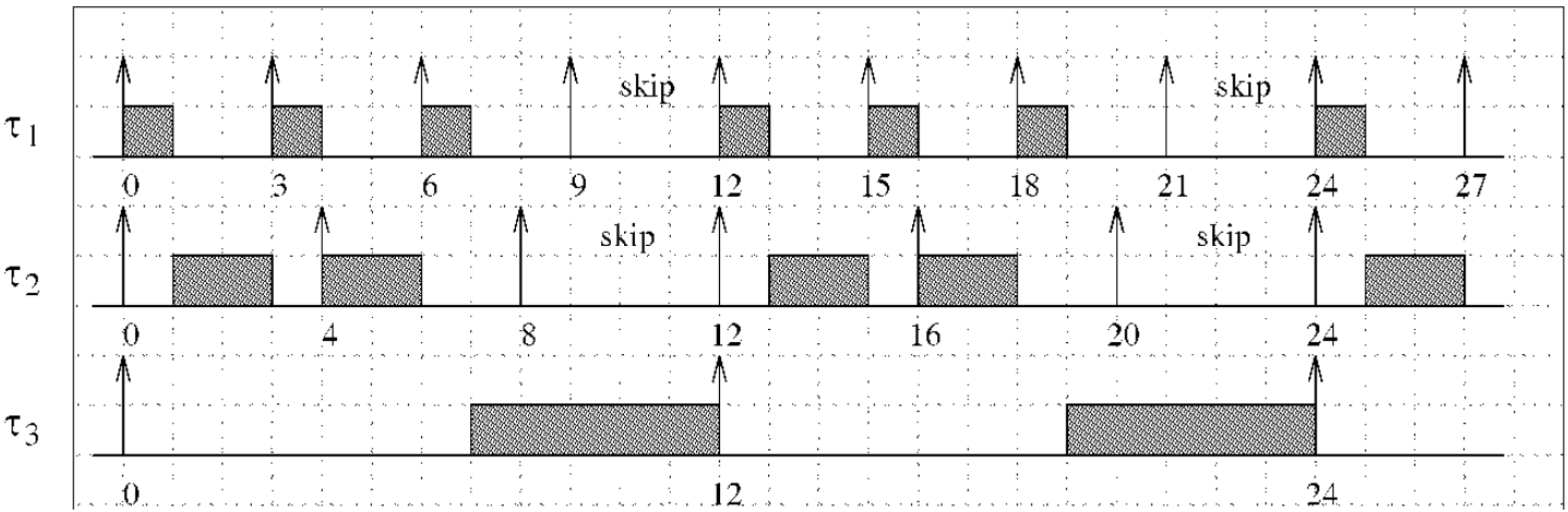


Figure taken from Buttazzo, G. et al: Soft Real-Time Systems

Job skipping

- Instances of tasks divided to:
  - Red instances – must complete before its deadline
  - Blue instances – can be aborted at any time

  - If a blue instance is skipped, then next s-1 instances must be red
  - If a blue instance is completed, the next instance is also blue

- Algorithms under EDF
  - Red tasks only
  - Blue when possible (blue scheduled when there are no ready red jobs to execute)

Schedulability of skippable tasks

- Given set $\Gamma = T_i(p_i, c_i, s_i)$ of n periodic tasks that allow skips an equivalent processor utilization factor can be defined as:

$$U_p^* = \max_{L \geq 0} \{ \frac{\sum_i D(i, [0, L])}{L} \}$$

where

$$D(i, [0, L]) = (\lfloor \frac{L}{p_i} \rfloor - \lfloor \frac{L}{p_i s_i} \rfloor) c_i$$

- A set Γ of skippable periodic tasks, which are deeply-red, is schedulable if an only if

$$U_p^* \leq 1$$

Schedulability of skippable tasks

- Deeply red means that all the tasks are synchronously activated and the first $s_i - 1$ instances of each task are red.
- This is kind of the worst case of the schedule

Spare capacity in skippable schedule

- Given a set of periodic tasks that allow skip with equivalent utilization U_p^* and a set of soft aperiodic tasks handled by a server with utilizaiton factor U_s, the hybrid set is schedulable by RTO or BWP if:

$$U_p^* + U_s \leq 1$$

Spare capacity in skippable schedule

Task	Task1	Task2
Computation	2	2
Period	3	5
Skip Parameter	2	∞
U_p	1.07	
U_p^*	0.8	
$1 - U_p^*$	0.2	

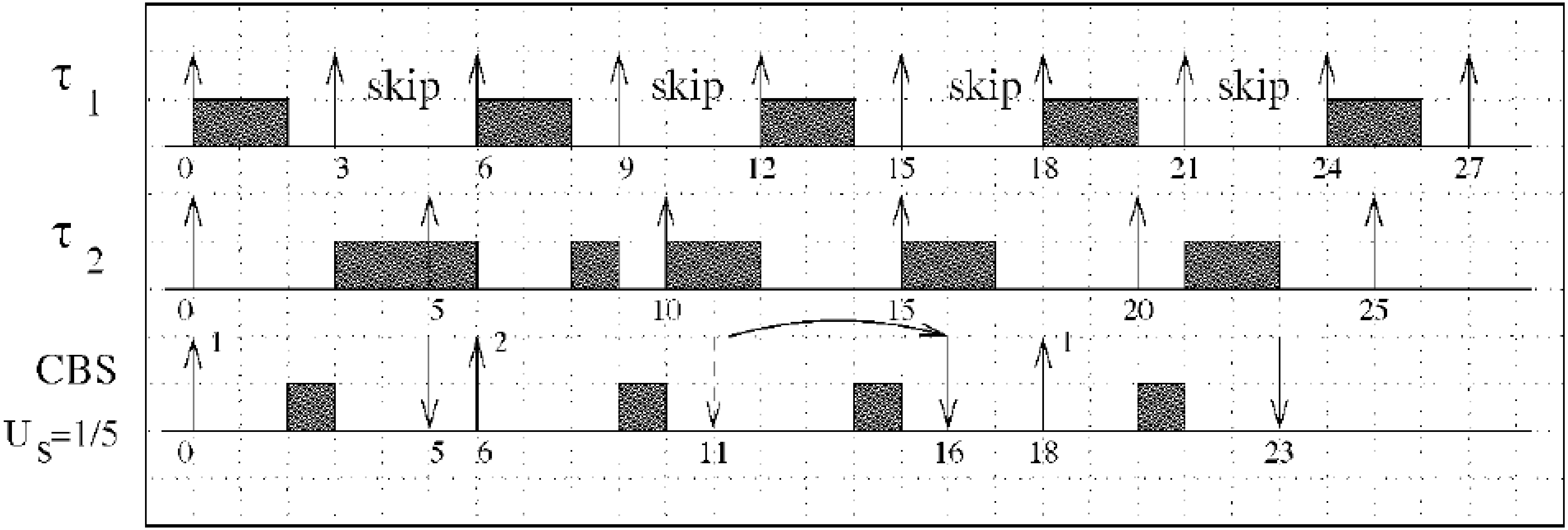


Figure taken from Buttazzo, G. et al: Soft Real-Time Systems

Period adaptation – Elastic model

- Tasks have nominal period T_{i_0} ,
  maximum period T_{i_max} and elastic coefficient E_i

- Task period may be stretched up to the maximum period
- The bigger the elastic coefficient, the more voluntary is the task to stretch its period

- The idea behind is that task utilization is like a spring, so we compress the task utilization
  - This has to be done iteratively due to period length constraints

Task compression

Algorithm Task_compress(Γ, U_d) {

$U_0 = \sum_{i=1}^{n} C_i/T_{i_0}$;
$U_{min} = \sum_{i=1}^{n} C_i/T_{i_{max}}$;
if ($U_d < U_{min}$) return INFEASIBLE;

do {

$U_f = U_{v_0} = E_v = 0$;
for ($each\ \tau_i$) {
  if (($E_i == 0$) or ($T_i == T_{i_{max}}$))
    $U_f = U_f + U_{i_{min}}$;
  else {
    $E_v = E_v + E_i$;
    $U_{v_0} = U_{v_0} + U_{i_0}$
  }
}

$ok = 1$;
for ($each\ \tau_i \in \Gamma_v$) {
  if (($E_i > 0$) and ($T_i < T_{i_{max}}$)) {
    $U_i = U_{i_0} - (U_{v_0} - U_d + U_f)E_i/E_v$;
    $T_i = C_i/U_i$;
    if ($T_i > T_{i_{max}}$) {
      $T_i = T_{i_{max}}$;
      $ok = 0$;
    }
  }
}

} **while** ($ok == 0$);
return FEASIBLE;
}