Aperiodic Task Scheduling

Gerhard Fohler
Mälardalen University, Sweden
gerhard.fohler@mdh.se

Non Periodic Tasks

So far periodic events and tasks
what about others?

- **Sporadic** (aperiodic, but minimum interarrival time)
  - worst case: all sporadic tasks arrive with highest frequency (with minimum time between arrivals)
  - all other arrival patterns less demanding
  - if we can schedule worst case, we can schedule all other
  - worst case - minimum interarrival time - like periodic task

  assume sporadic tasks as periodic for schedulability test

- **Aperiodic** (no limitations on arrival times known)
- **soft:** without deadline
  not much to do from scheduling view
- **firm:** with deadline
  (worst case execution time needs to be known as well)
  usually “all or nothing” semantic:
  when we start task, we want that it runs until completion; else we don’t start

Background Services

Fixed priority scheduling, rate monotonic

What is the minimum we can do for aperiodic tasks in a periodically scheduled system?

Background service: execute aperiodic tasks when no periodic ones are executing

- no disturbance of periodic tasks (and their feasibility)
- simple run-time mechanisms
  - queue for periodics
  - queue for aperiodics - FCFS
- no guarantees

Polling Server

- Background service lives from “left overs” of periodic tasks, without guarantees
- If enough idle time, ok
- long response times, although faster service possible
- How can we provide that at least a certain amount of processing goes to aperiodic tasks?

Server task

periodic task, whose purpose is to service aperiodic requests as soon as possible

- period T_s
- computation time C_s is called **capacity** of the server

Polling server algorithm

- at periods T_s server becomes active and serves aperiodic requests with its capacity C_s
- no aperiodic activities - not execute, waits for next period, capacity lost
- based on rate monotonic

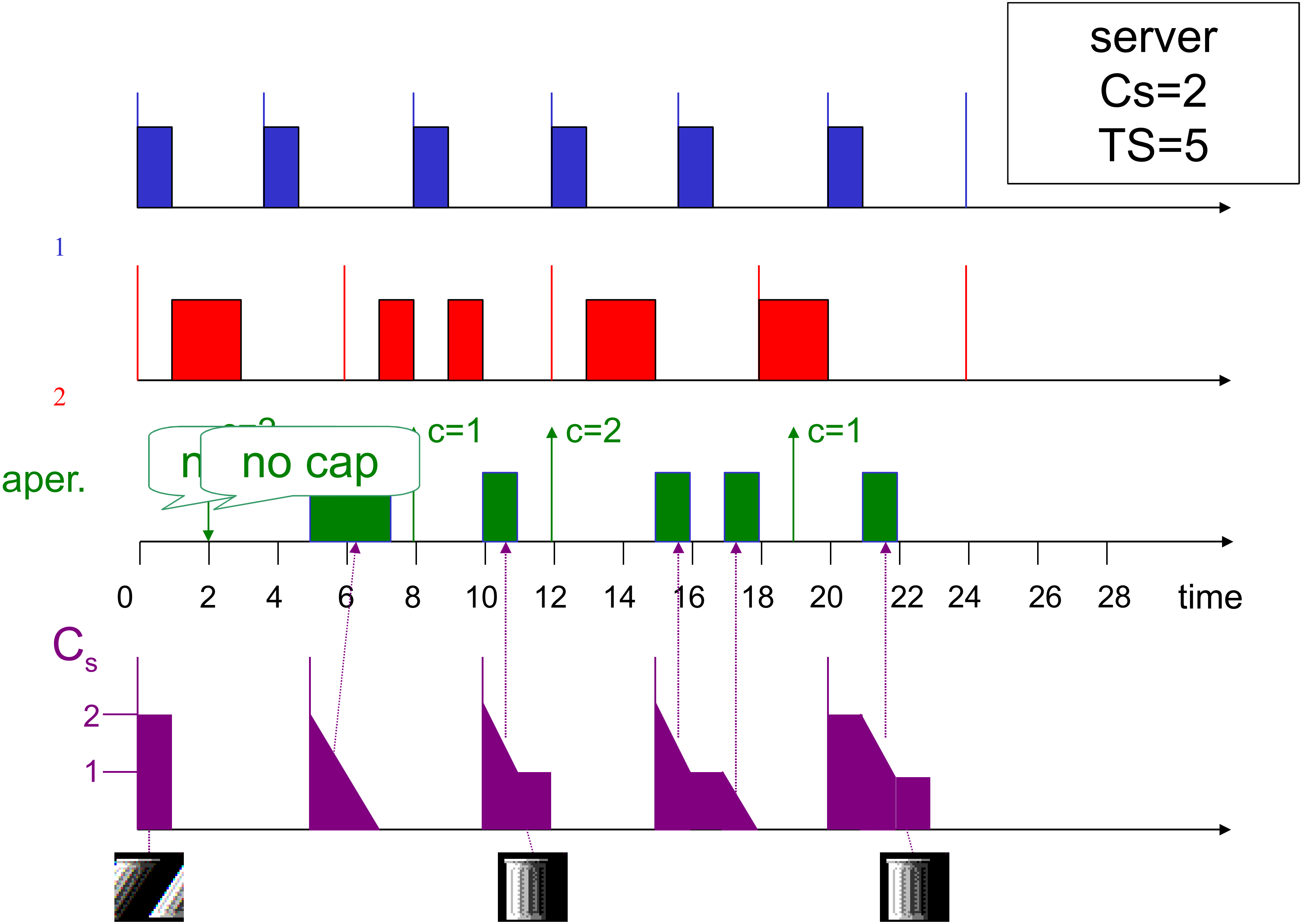
Lehozcky, Sha, Strosnider, Sprunt 1987, 1989

Example Polling Server

- two tasks $_1$, $_2$

	C_i	T_i
$_1$	1	4
$_2$	2	6

- server
  C_s=2
  T_S=5


server
Cs=2
TS=5
1
2
aper.
no cap
c=1
c=2
c=1
0 2 4 6 8 10 12 14 16 18 20 22 24 26 28
time
C_s
2
1

Aperiodic Guarantee

Aperiodic guarantee

hard aperiodic task T_a, C_a, D_a

worst case:

- aperiodic request misses the server task
- has to wait until next instance
- if $C_a <= C_s$, aperiodic request completed within two server periods (one for waiting, one for executing)
  $2*T_s <= D_a$
- arbitrary execution times:
  $T_s + \quad C_a/C_s \quad T_s <= D_a$

average response time not very good!

Further FPS Server Algorithms

- **Deferrable Server**(Lehozczky, Sha, Strosnider 1987, 1995)
  - lower bound for periodic tasks
- **Priority exchange**
  - (Lehozczky, Sha, Strosnider 1987)
- **Sporadic server**
  - Sprunt, Sha, Lehozcky 1989
  - replenishes capacity only after aperiodic execution
- **optimum algorithm**
  - does not exist!
  - Tia, Liu, Shankar 1995
  - proof that with static priority assignment, no algorithm exists to minimize response time

Dynamic Priority Servers

- EDF based
- **Dynamic priority exchange server**
  - Spuri, Buttazzo 1994, 1996
  - like rate monotonic priority exchange, but for EDF
- **Dynamic sporadic server**
  - Spuri, Buttazzo 1994, 1996
- **Earliest deadline late server**
  - Chetto, Chetto 1989

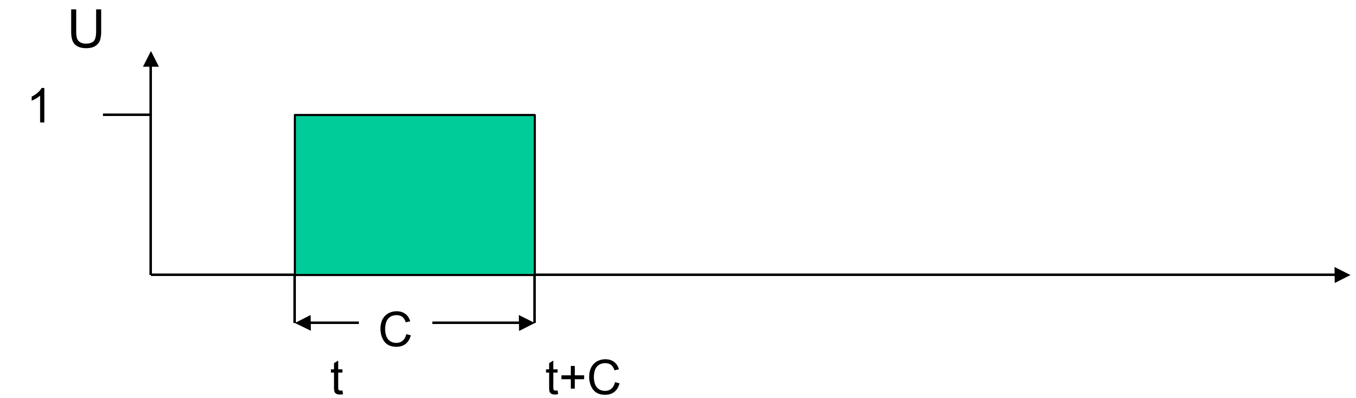
Total bandwidth server

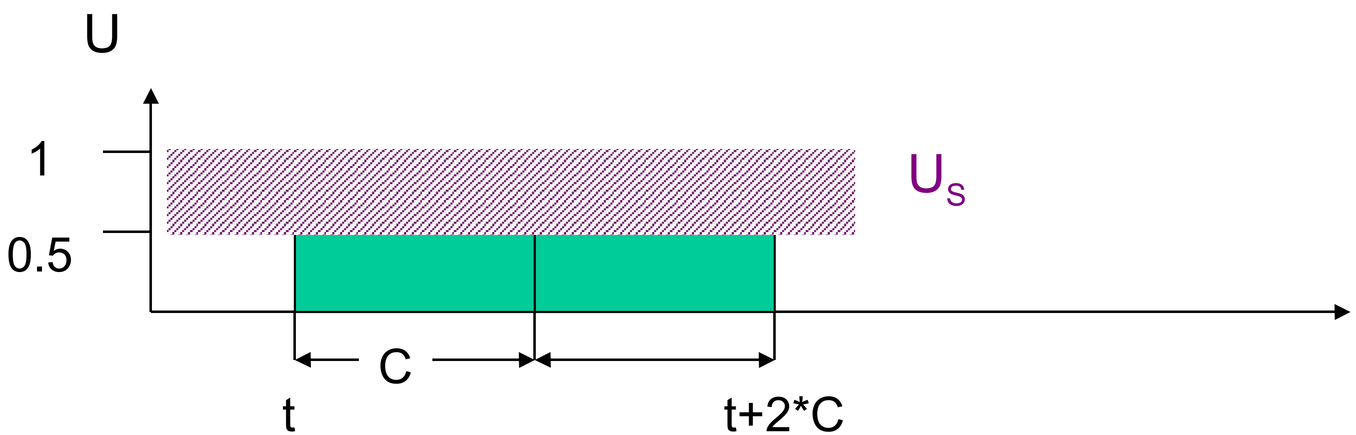
- Spuri, Buttazzo 1994, 1996
- response time dependent on server period:
  - shorter periods have shorter response times
  - but higher overhead
- how else shorter response times?
  - change the deadline of the aperiodic to earlier time (its EDF here, so it will get serviced earlier)
  - but make sure that total load of aperiodics does not exceed maximum value (bandwidth) U_s

How can we calculate minimum deadline for U_S?

assume we have all CPU for us:

dl = C





k^{th} aperiodic request

- arrival time r_k
- computation time c_k
- deadline d_k
- server utilization U_S

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

$d_0 = 0$

- uses all bandwidth of server
- very simple run-time mechanism
- no extra server task

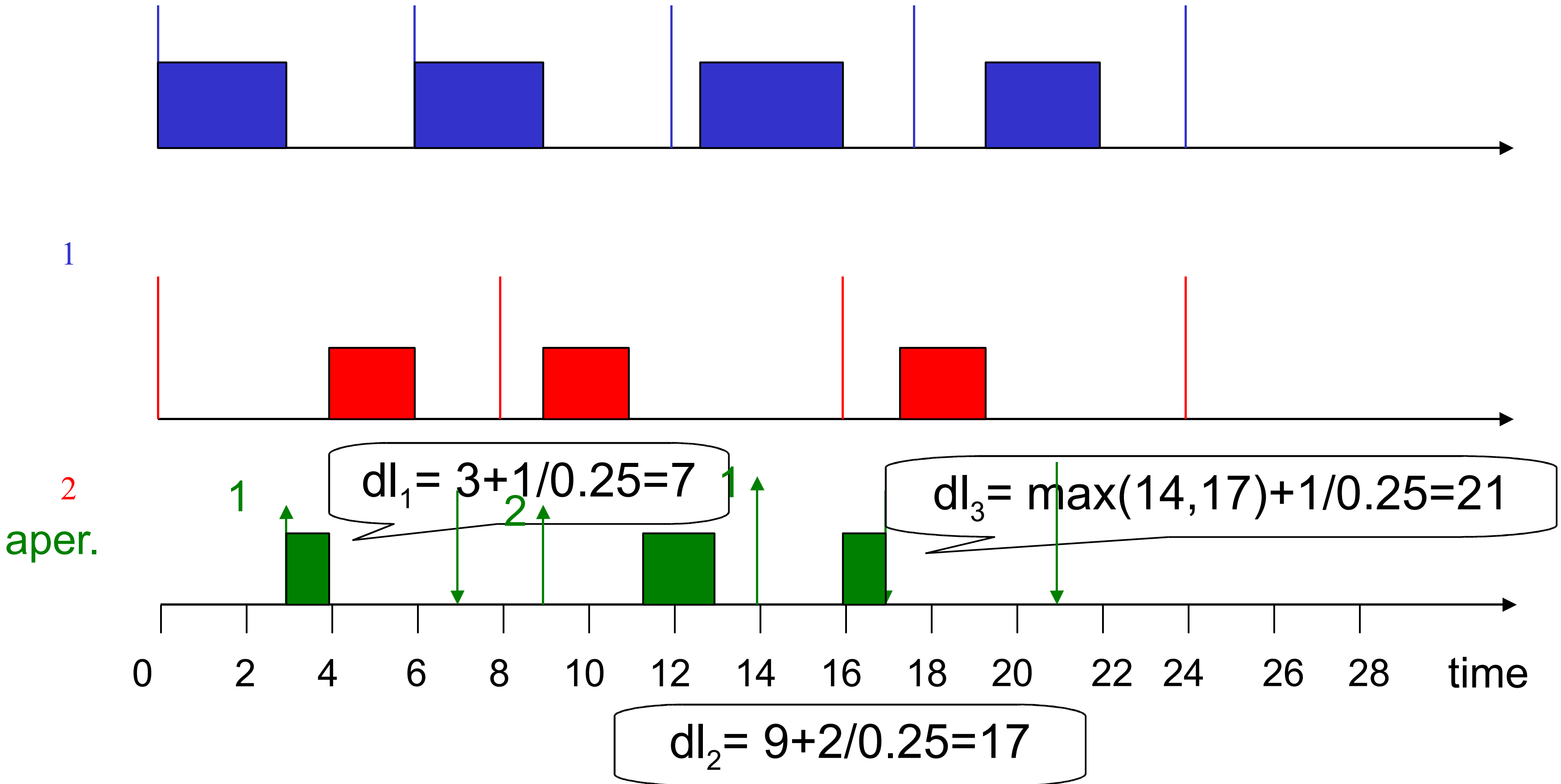
schedulability

$U_p + U_s <= 1$

Sum of periodic load and bandwidth of server less or equal 1.

Example Total Bandwidth Server

- periodic tasks $_{1}$(3,6), $_{2}$(2,8)
- TBS $U_s = 1 - U_p = 0.25$

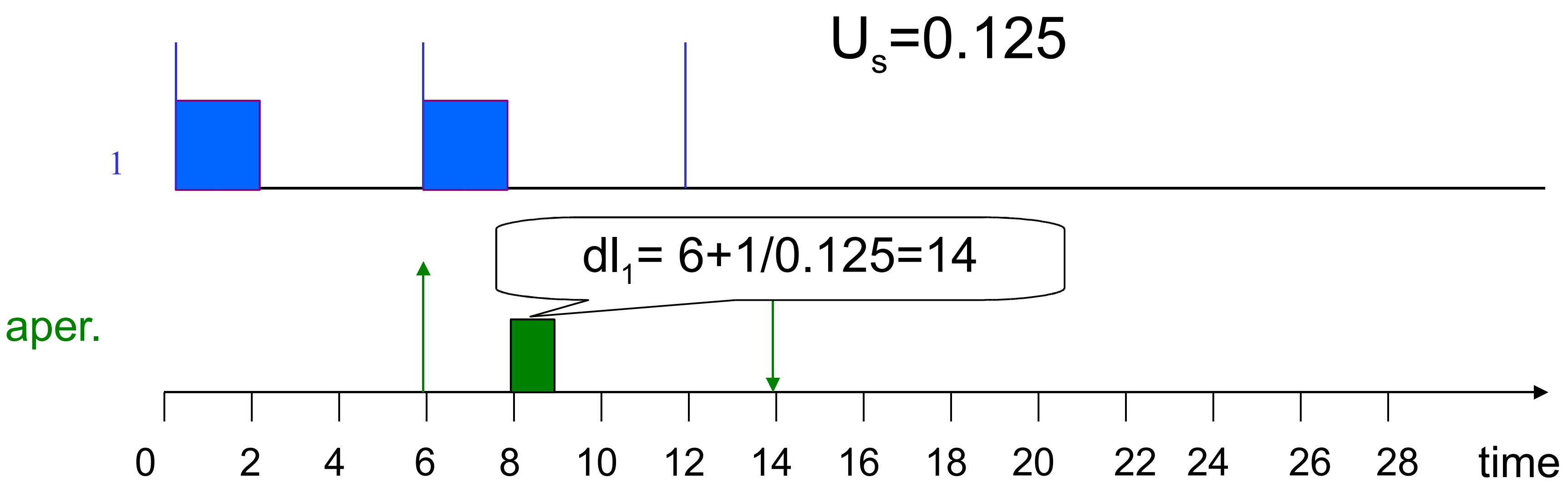


Total Bandwith Server - Comments

- based on
  - U_s not actual periodic load
  - worst case c

Total Bandwith Server - Comments

- TBS assigns deadlines based on maximum U_s (not actual load)
  $d_k = \max(r_k, d_{k-1}) + C_k/U_s$, $d_0 = 0$



TB*

- Buttazzo, Sensini - 1997
- assigns deadlines d_k first according to TBS
- then shortens, as much as periodics allow
  - new $d'_k = f_k$ …finishing time according to EDF schedule, including periodics
  - apply recursively
  - maintains schedulability, since order maintained
- complexity, many steps

Constant Bandwidth Server

- Abeni and Buttazzo, 1998
- designed for multimedia applications
  - sporadic (hard) tasks
  - soft tasks: mean execution, interarrival times, not fixed
  - periodic tasks
- assign maximum bandwidth of CPU to each soft task
- handles overload of aperiodics
  - limited by assigned bandwidth
  - might slow down, but not impair effect other tasks
- EDF based

CBS Definitions

- task $_i$
  - sequence of jobs $J_{i,j}$
  - $r_{i,j}$ … request, arrival time of the j[th] job of task $_I$
- hard task
  - (C_i,T_i)
    - C_i worst case execution time
    - T_i minimum interarrival time
    - deadline equal to next period: $d_{i,j} = r_{i,j} + T_i$
- soft task
  - (C_i,T_i)
    - C_i *mean* execution time
    - T_i *desired* interarrival time
    - soft deadline equal to next period: $d_{i,j} = r_{i,j} + T_i$

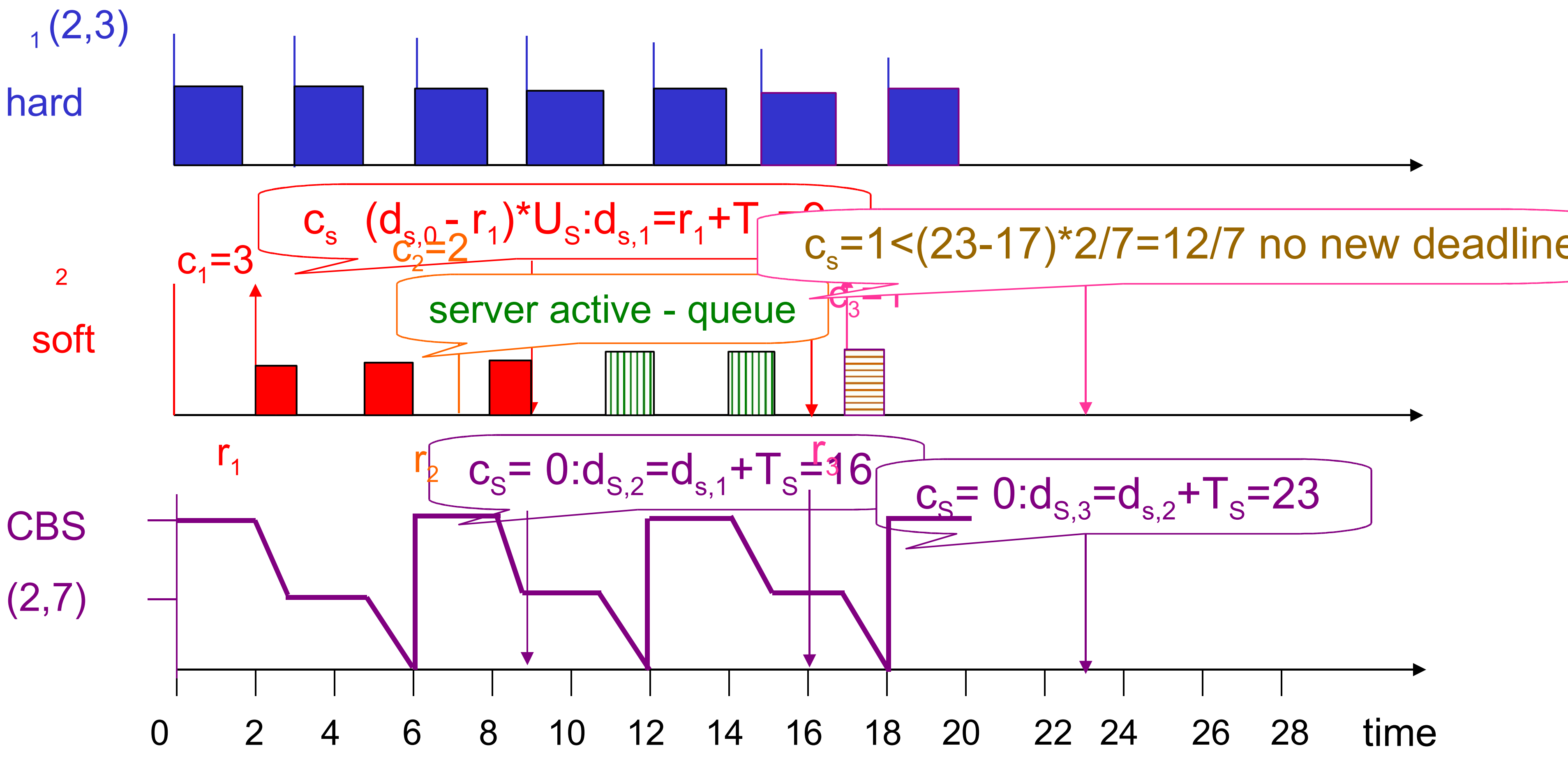
- c_S … <u>budget</u>
- (Q_S,T_S)
  - Q_S … maximum budget
  - T_S … period of server
- $U_S = Q_S/T_S$ … *<u>server bandwidth</u>*
- $d_{S,k}$ … deadline associated to server
  - initial $d_{S,0} =0$

- job $J_{i,j}$ <u>comes in</u>, is served, <u>assigned dynamic deadline $d_{i,j}$</u> equal to current server deadline $d_{S,k}$
  - job executes, server budget c_s decreased

- $c_S=0$:
  - <u>budget recharged</u> to maximum Q_S
  - new server deadline: $d_{s,k+1}=d_{s,k}+T_S$

- $J_{i,j}$ arrives, CBS active (jobs pending): put in queue
- $J_{i,j}$ arrives, CBS idle:
  - ~~c_S _ $(d_{s,k} - r_{i,j})*U_S$~~:
    - <u>new deadline</u> $d_{s,k+1} = r_{i,j} + T_S$
    - c_S recharged to Q_S
  - else
    - job served with last server deadline $d_{s,k}$
- job finishes: next job in queue
- at any time, job assigned last deadline generated by server

Example CBS



1 (2,3)
hard
2
soft
c1=3
cs (ds,0 - r1)*US:ds,1=r1+T
c2=2
cs=1<(23-17)*2/7=12/7 no new deadline
server active - queue
c3=1
r1
r2
r3
cS= 0:dS,2=ds,1+TS=16
cS= 0:dS,3=ds,2+TS=23
CBS
(2,7)
0 2 4 6 8 10 12 14 16 18 20 22 24 26 28
time

- limits impact “harm” by ill behaved aperiodics, e.g., exec time overrun
- various improvements
  - several servers
  - capacity exchange
  - feedback control
  - …..

Articles

- TBS:
  Spuri, Buttazzo
  “Efficient Aperiodic Service under Earliest Deadline Scheduling”
  Proceedings of the 15th IEEE Real-Time System Symposium (RTSS 94), Portorico, pp. 2-21, December 1994
- CBS:
  L. Abeni and G. Buttazzo, "Integrating Multimedia Applications in Hard Real-Time Systems", Proceedings of the IEEE Real-Time Systems Symposium, Madrid, Spain, pp. 4-13, December 1998.

Schedulability Analysis

First show that aperiodic load executed not exceeds U_S of server

Lemma: *In each interval of time [t_1,t_2], if C_{ape} is the total execution time demanded by aperiodic requests arrived at t_1 or later and served with deadlines less than or equal to t_2, then*

$$C_{ape} \quad (t_2 - t_1)\, U_S$$

Proof: by definition:

$$C_{ape} \qquad\qquad C_k$$

$$t_1 \quad r_k, d_k \quad t_2$$

- TB* uses periodic interference…can now calculate it

- (formulae for completeness only)
  $I_f(t, d_k^s) =$

$$\overset{n}{\underset{i\ \ 1}{}}\ \max\ \ 0,\ \frac{d_k^s \quad next\_r_i(t)}{T_i} \quad 1\ \ C_i$$

next_r$_i$(t)…time at which next instance of I after t starts

TBS assigns deadlines in increasing order,

therefore there must exist two aperiodic requests with indeces k_1 and k_2 such that

$$\overset{}{\underset{t_1 \quad r_k, d_k \quad t_2}{}} C_k \qquad \overset{k_2}{\underset{k \quad k_1}{}} C_k$$

$$C_{ape} \qquad \overset{k_2}{\underset{k \quad k_1}{}} C_k \qquad \overset{k_2}{\underset{k \quad k_1}{}} [d_k \quad \max(r_k, d_{k \quad 1})] * U_S$$

$$[d_{k_2} \quad \max(r_{k_1}, d_{k_1 \quad 1})] * U_S$$

□ $$(t_2 \quad t_1) * U_S$$

Proof main result:

Theorem: Given a set of n periodic tasks with processor utilization U_p and a TBS with processor utilization of U_S, the whole set is schedulable by EDF if and only if
$U_p + U_S$ 1

Proof: If:

- assume $U_p + U_S$ 1 <u>plus</u> overflow at time t
- overflow preceded by continuous utilization
- from a point t' on (t'< t), only instances of tasks ready at t' or later *and* having deadlines less than or equal to t are run
- C total execution time demanded by these instances
- since there is overflow at t: t - t' < C

- we also know that

$$C \qquad {}_{i \;\; 1}^{\;n} \quad \frac{t \quad t'}{T_i} * C_i \qquad C_{ape}$$

$${}_{i \;\; 1}^{\;n} \quad \frac{t \quad t'}{T_i} * C_i \qquad (t \quad t') * U_S$$

$$(t \quad t') * (U_p \quad U_S)$$

it follows: $U_p + U_S > 1$ …# contradiction

- **only if:**
- assume aperiodic request enters periodically with period T_S and execution time $C_S=T_S U_S$, then server behaves like periodic task
- total utilization of processor is then U_p+U_S
- if task set schedulable: U_P+U_S 1

□