redhat.

Deadline Scheduler

Open Issues

Daniel Bristot de Oliveira
Red Hat, Inc.

Who is Daniel?

redhat

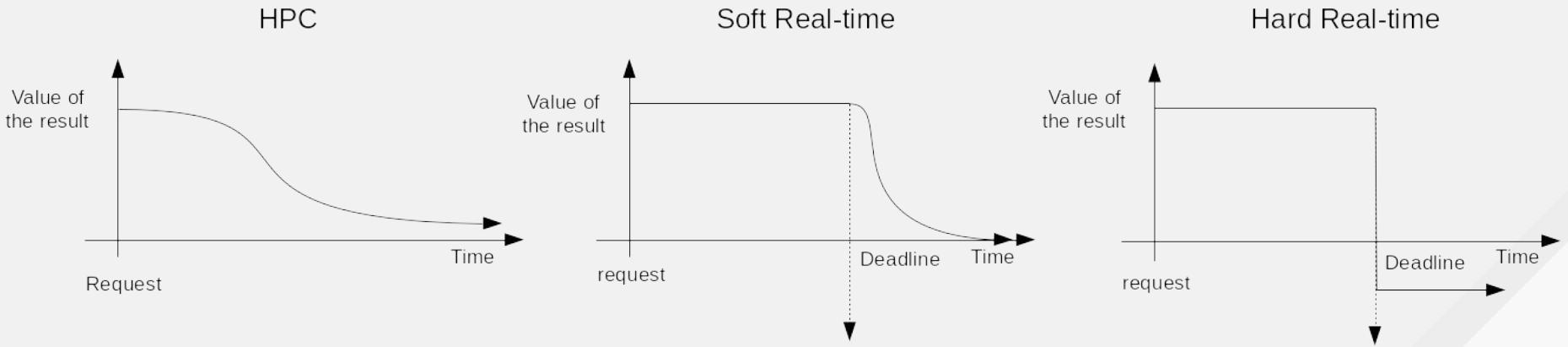
Real-time systems

Systems which deal with external events with timing constraints

- Real from real/external-world
- Time from timing constraints

The response of an event is correct if and only if:

- The logical response is correct
- It is produced within a deadline



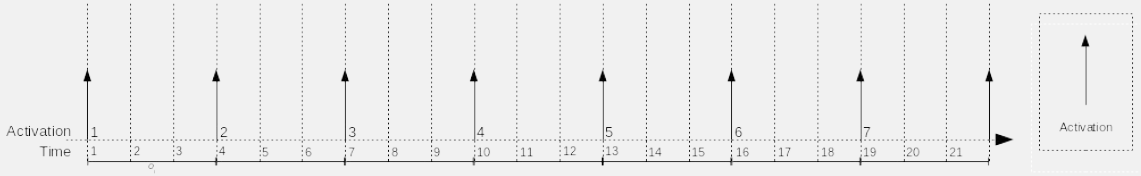
Real-time scheduler modeling

- A system is view as a “model”
  - A system is composed by a set of *n* tasks
  - A task is a set of infinity recurring jobs.
  - Each task is characterized by some parameters:
    - C or Q = WCET or Budget
    - T or P = Period or Minimum inter-arrival time
    - D = Deadline

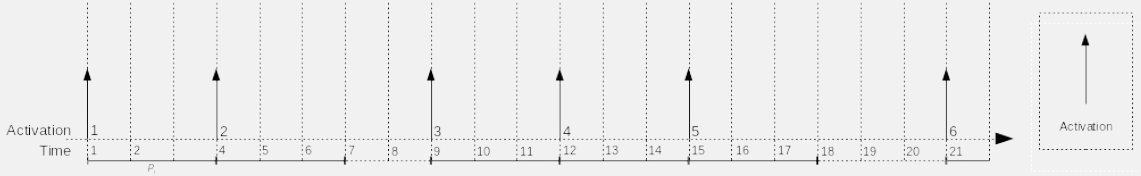
on Linux, DL tasks are characterized by:

- dl_period = Period [sporadic || periodic]
- dl_deadline = Relative deadline [by default == period ... but can be <]
- dl_runtime = Execution time;

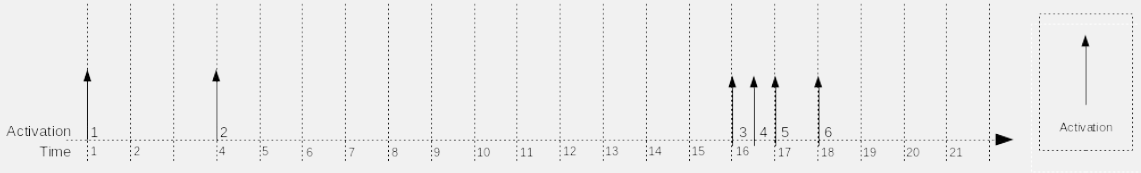
Regarding Period:



Periodic

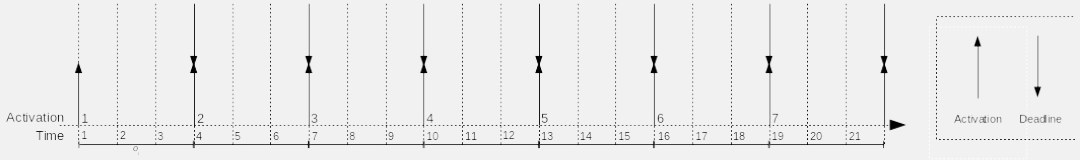


Sporadic

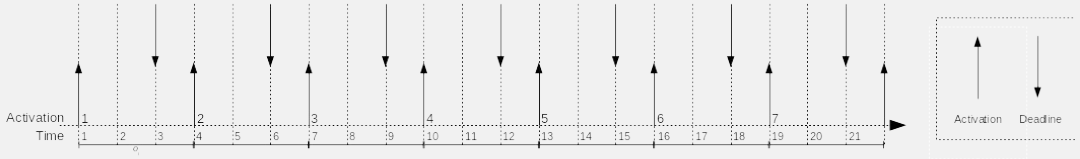


Aperiodic

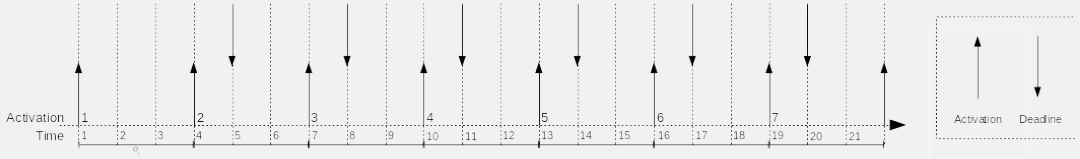
Regarding Deadline:



Implicit



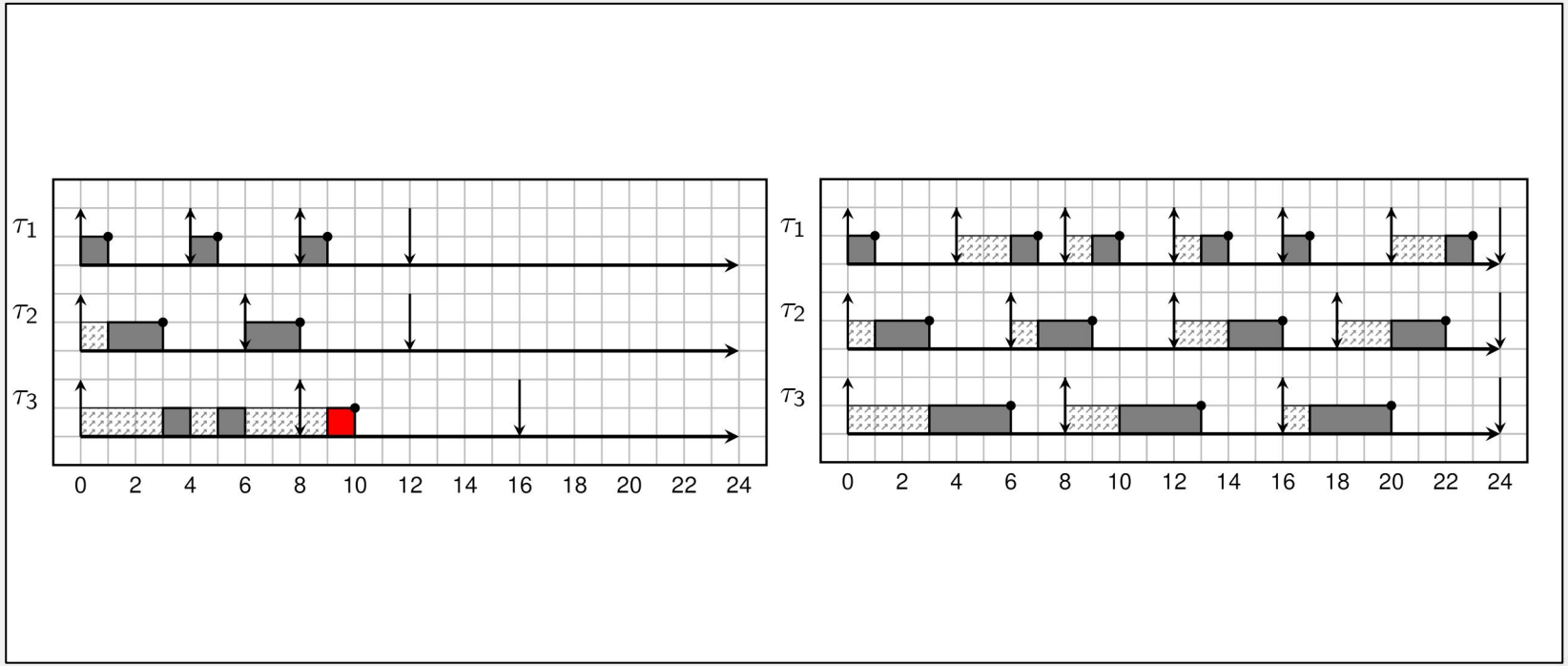
Constrained



Arbitrary

Why EDF scheduler?

Fixed Priority *versus* Deadline



EDF is optimal!

*Under optimal conditions

EDF is optimal (U<=1) with

- If tasks does not misbehave
- Job does not suspend (dequeue/enqueue) during an activation
- Implicit deadline (deadline == period)
- Uniprocessor

Note:
[U]tilization = C/T (or Q/P, runtime/period)
[D]ensity = C/D (or Q/D, runtime/deadline)

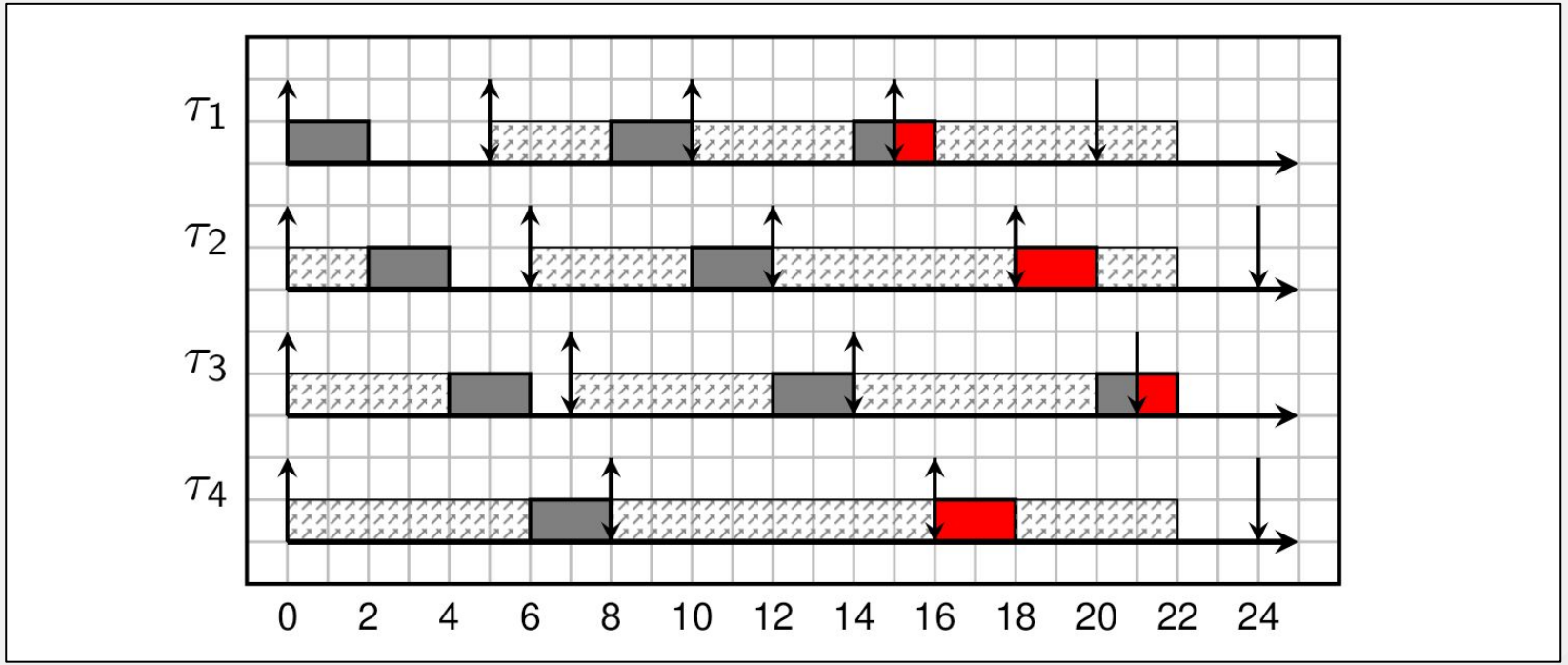
So, let’s explore each point!

What if a task runs longer than it said (C) it was suppose to run?

Or

What if the utilization goes higher than 100%?

The domino effect



To avoid the domino problem...

- Admission control to avoid overload:
  - The sum of the Utilization of all tasks cannot be higher than rt_period-rt_runtime/rt_period (by default 95%).
- CBS to avoid a misbehaving task to run more than runtime.

CBS: Constant Bandwidth Server

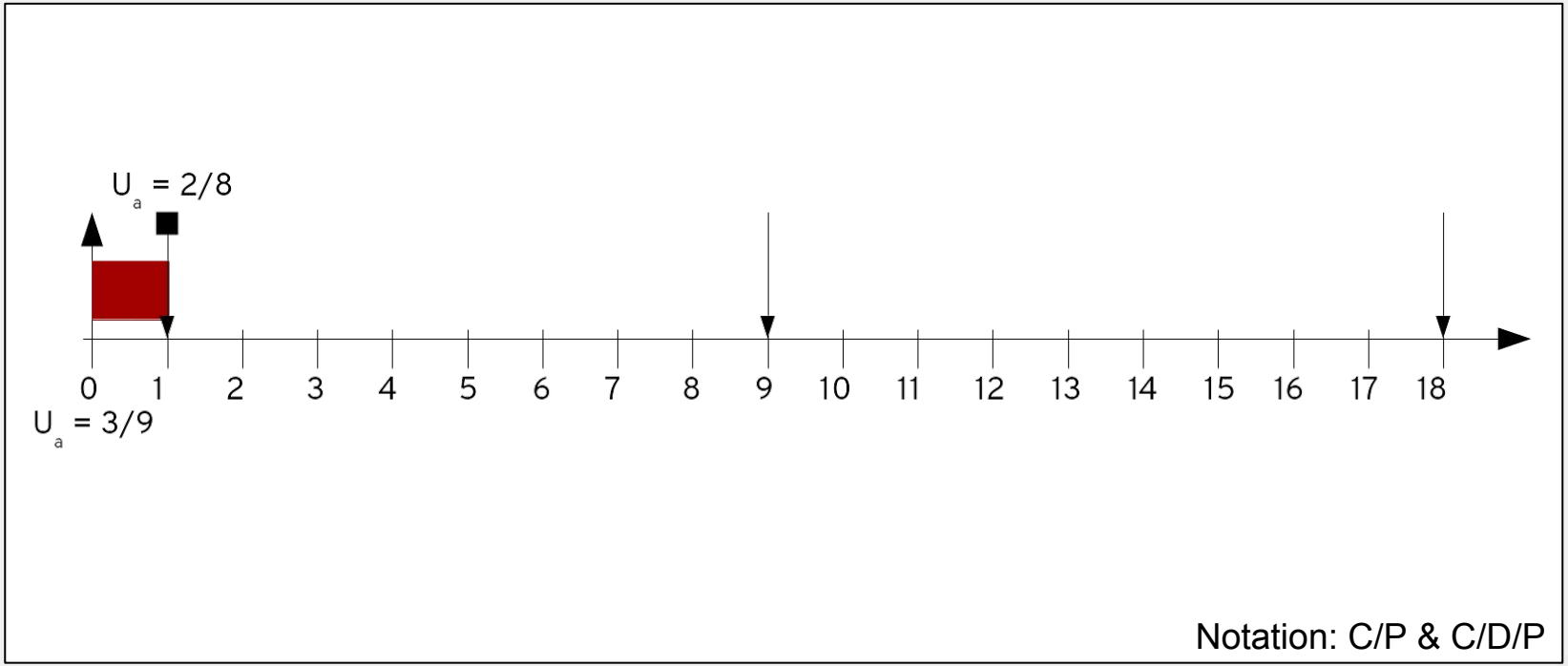
- Throttle a misbehaving task that uses more than allowed
- Try to provide **runtime** CPU time every **period.**
  - It relies on non-suspending tasks.

CBS & Suspending task

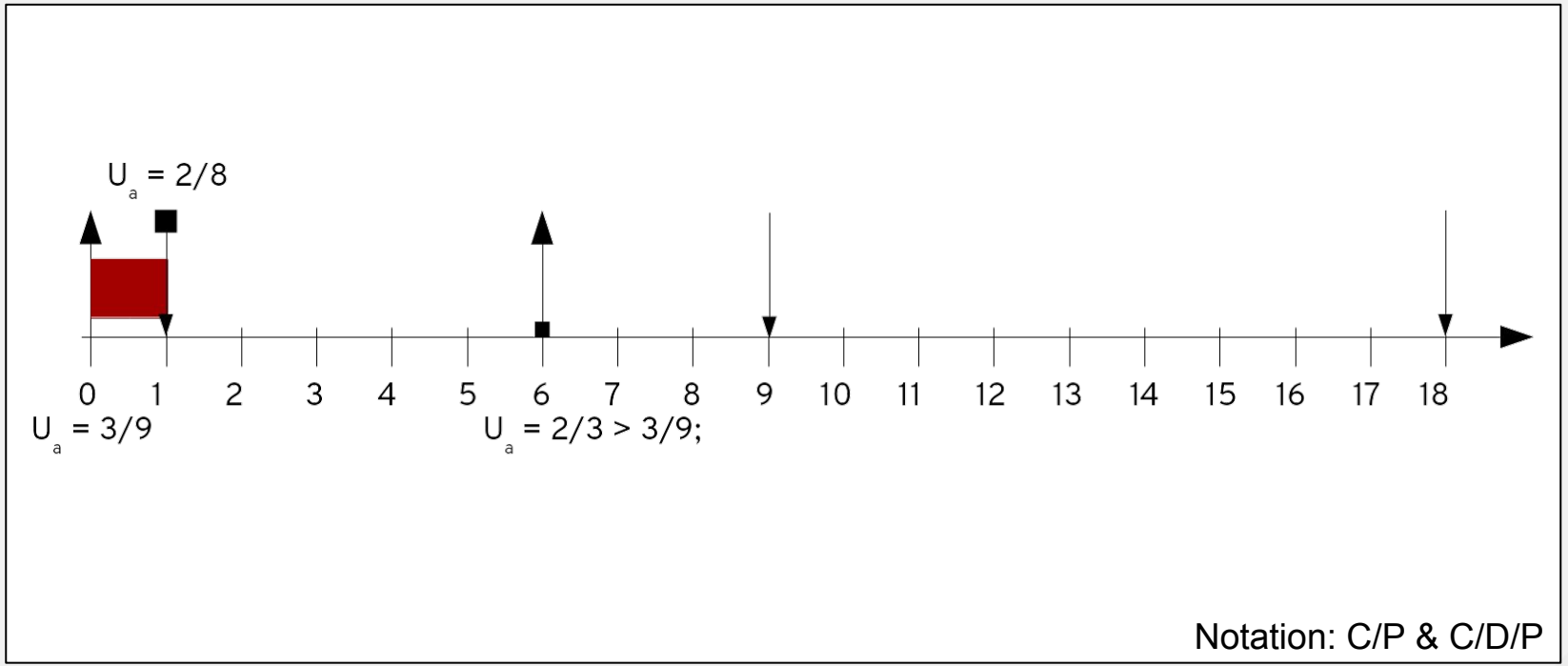
By assuming non-suspending tasks...

- It is implicitly assumed that, when queued, the **absolute U** of a task is bound to its **relative U** (U=runtime/period).
- In other words: The task will never overload the system.
- **If the task suspends/blocks, that might not be the case...**

For example, a task with U = 3/9 blocks with 2/8



Returning with U=2/3



CBS & Self-suspending tasks

- CBS wakeup rule (ensures that a task will not overload the system):

    If the **deadline** is in the **past**:

        new absolute **runtime** and absolute **deadline** is set.

    If the **deadline** is in the **future**:

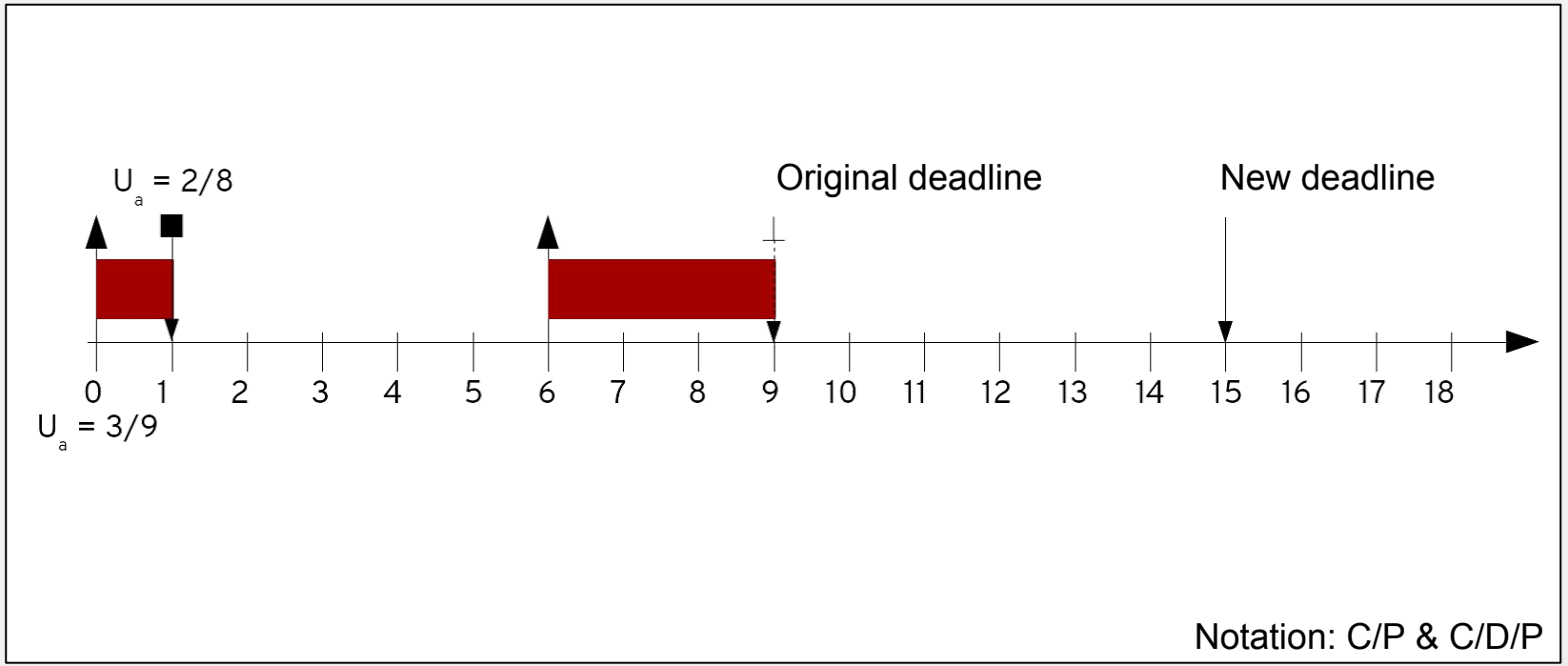
        If the **possible U** < **allowed U**

            Go ahead and run, my little reservation.

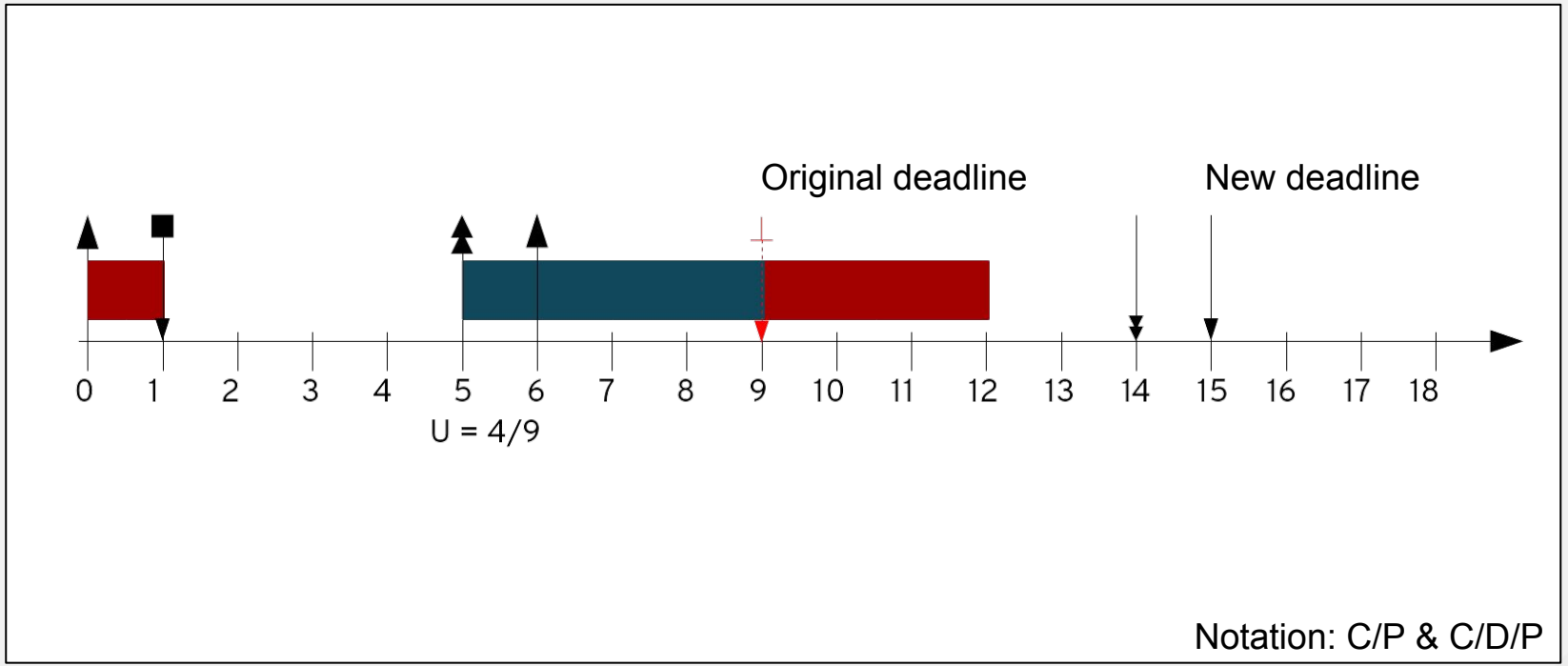
        else

            Reset runtime, set the new deadline

Replenish the runtime and reset period



In the presence of another deadline task...

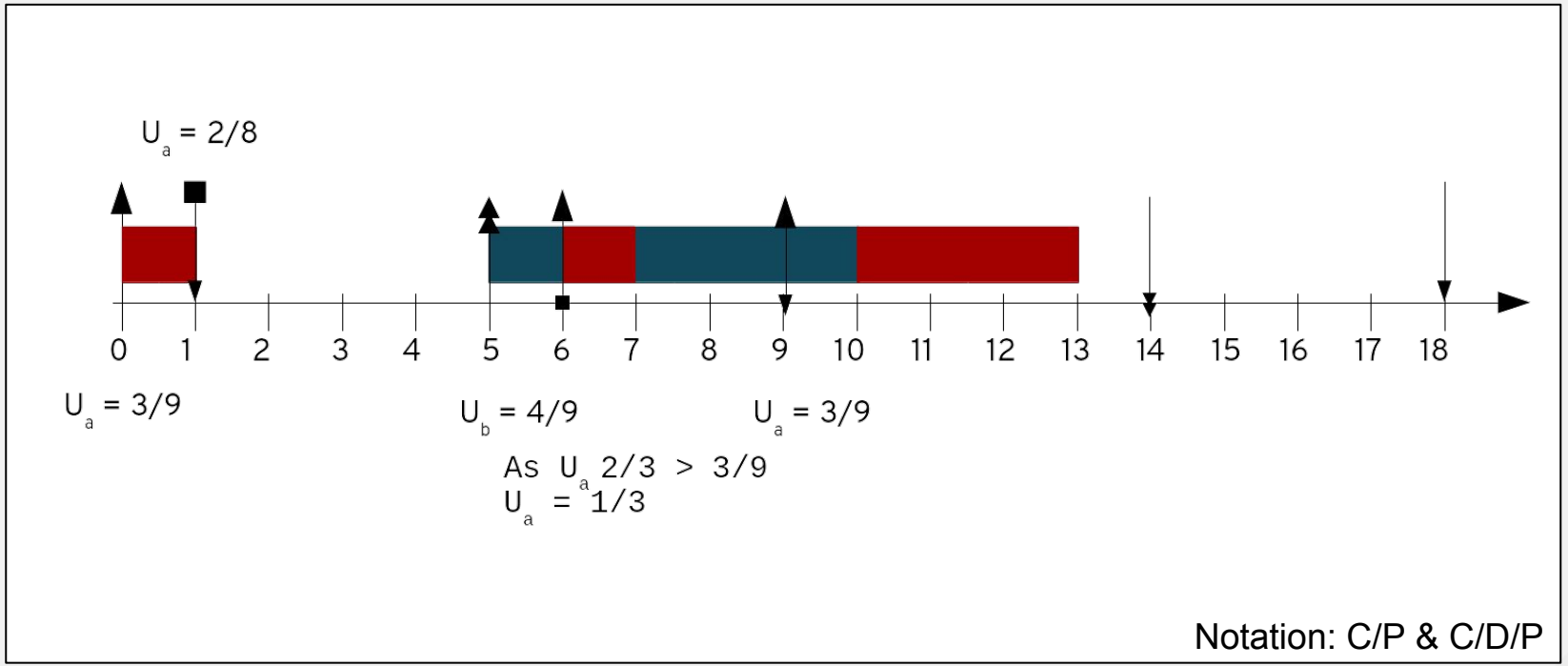


What do we care more,
having runtime/period after a wakeup
or try to make the deadline?

Revised CBS & Self-suspending tasks

- CBS wakeup rule (ensures that a task will not overload the system):
    - If the **deadline** is in the **past**:
        - new absolute **runtime** and absolute **deadline** is set.
    - If the deadline is in the future:
        - If the **absolute U** < **relative U**
            - Go ahead and run, my little CBS.
        - else
            - Truncate runtime, new runtime = (C / P) * laxity

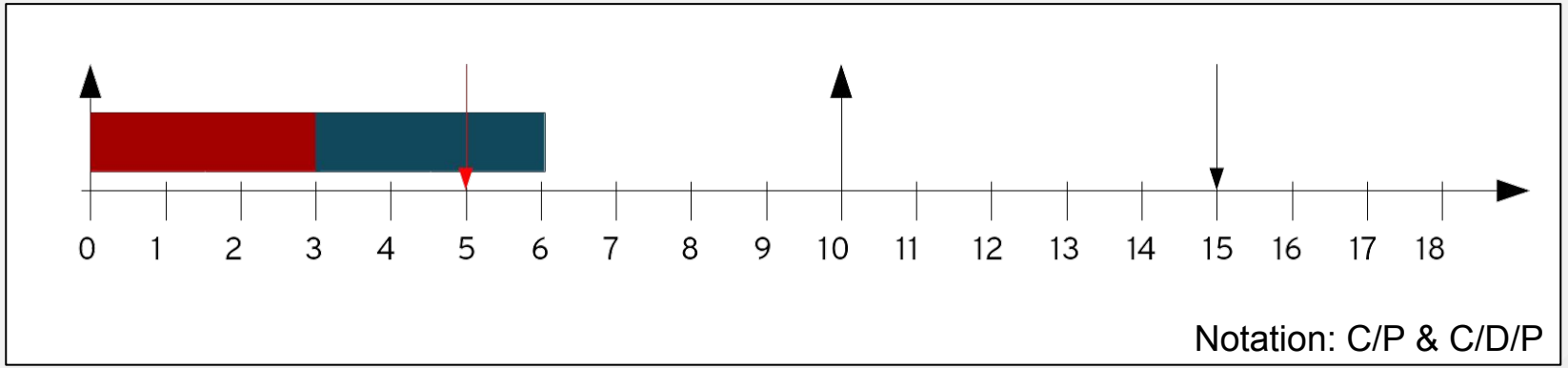
Using the revised CBS:



Should we consider using the revised CBS?

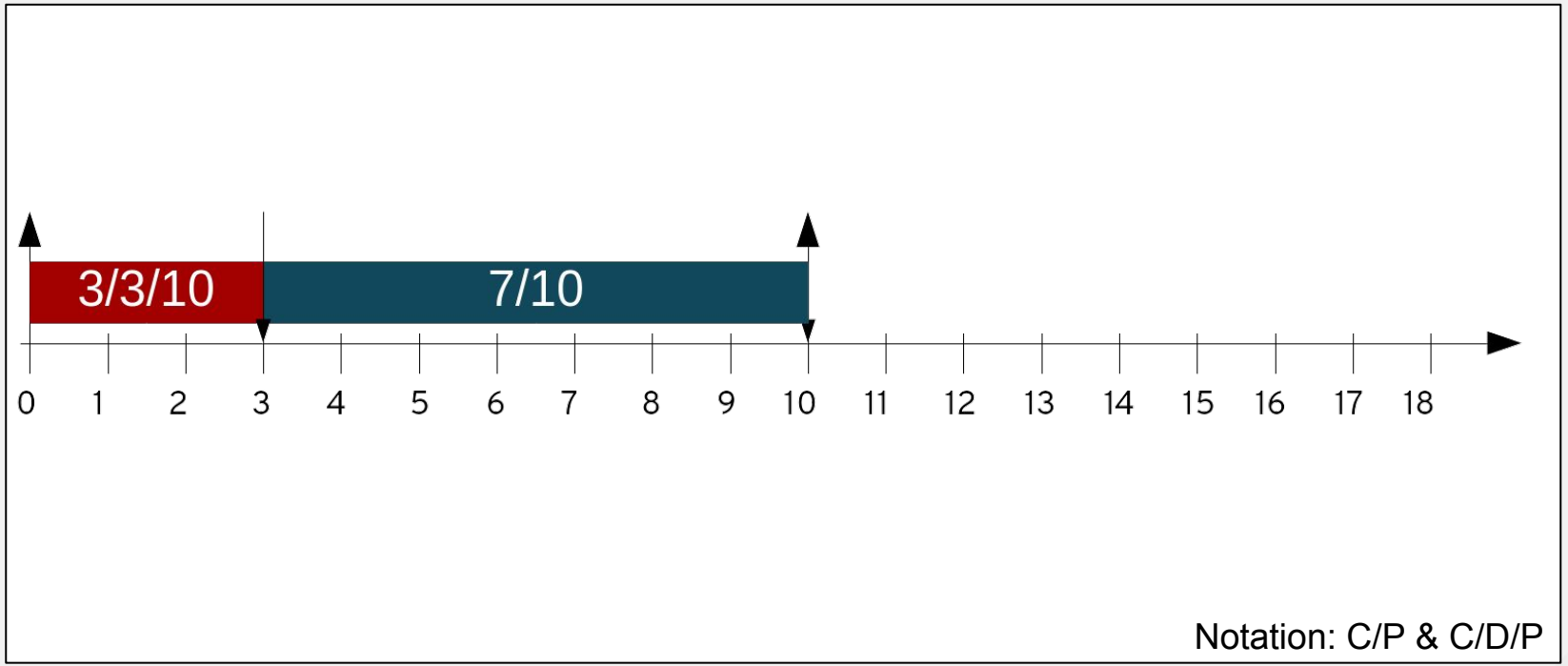
Constrained deadline

- Linux's deadline scheduler accepts task with deadline <= period.
- In the presence of an !implicit deadline task, the admission test is not valid to "guarantee" the deadline, even on single-core systems.
- For example, two tasks with 3/10 (60%) but deadline of 5:



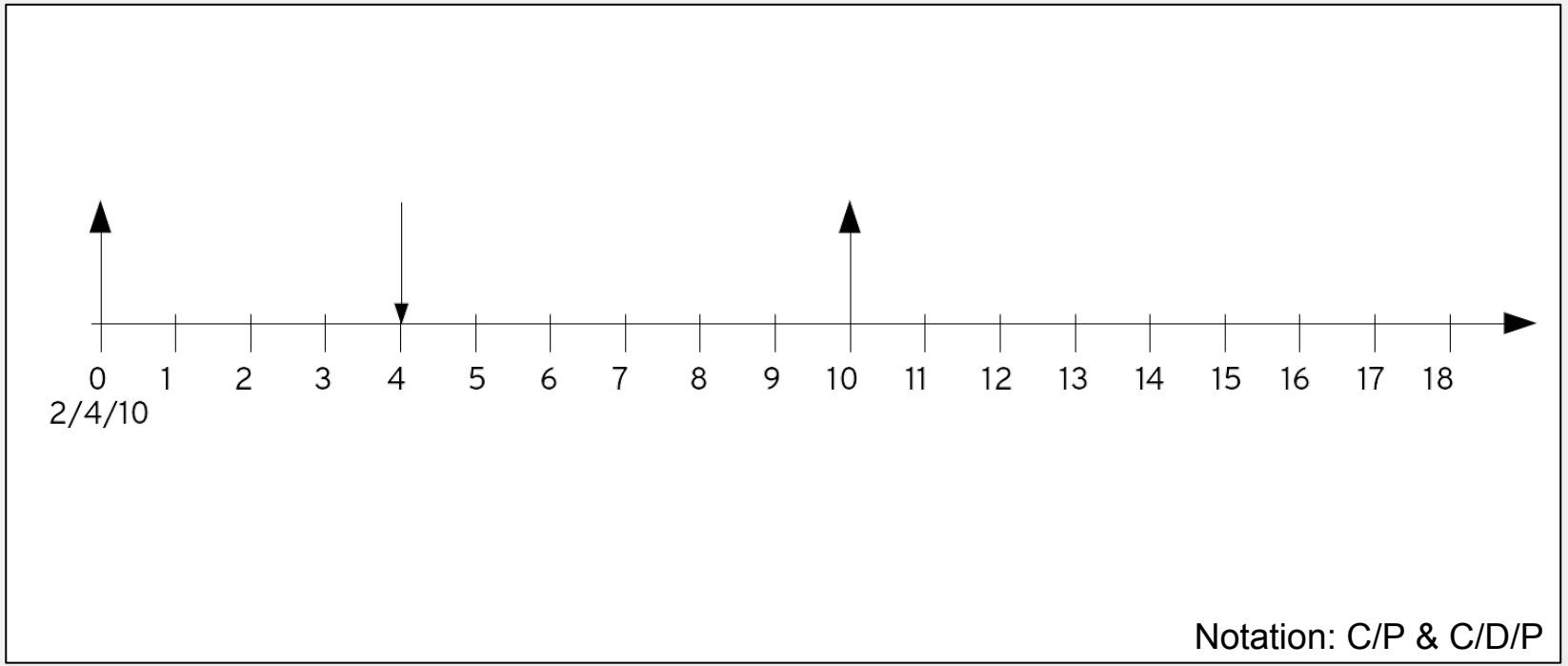
That is easy!
We should use runtime/deadline,
not runtime/period!

No, it is too pessimistic...

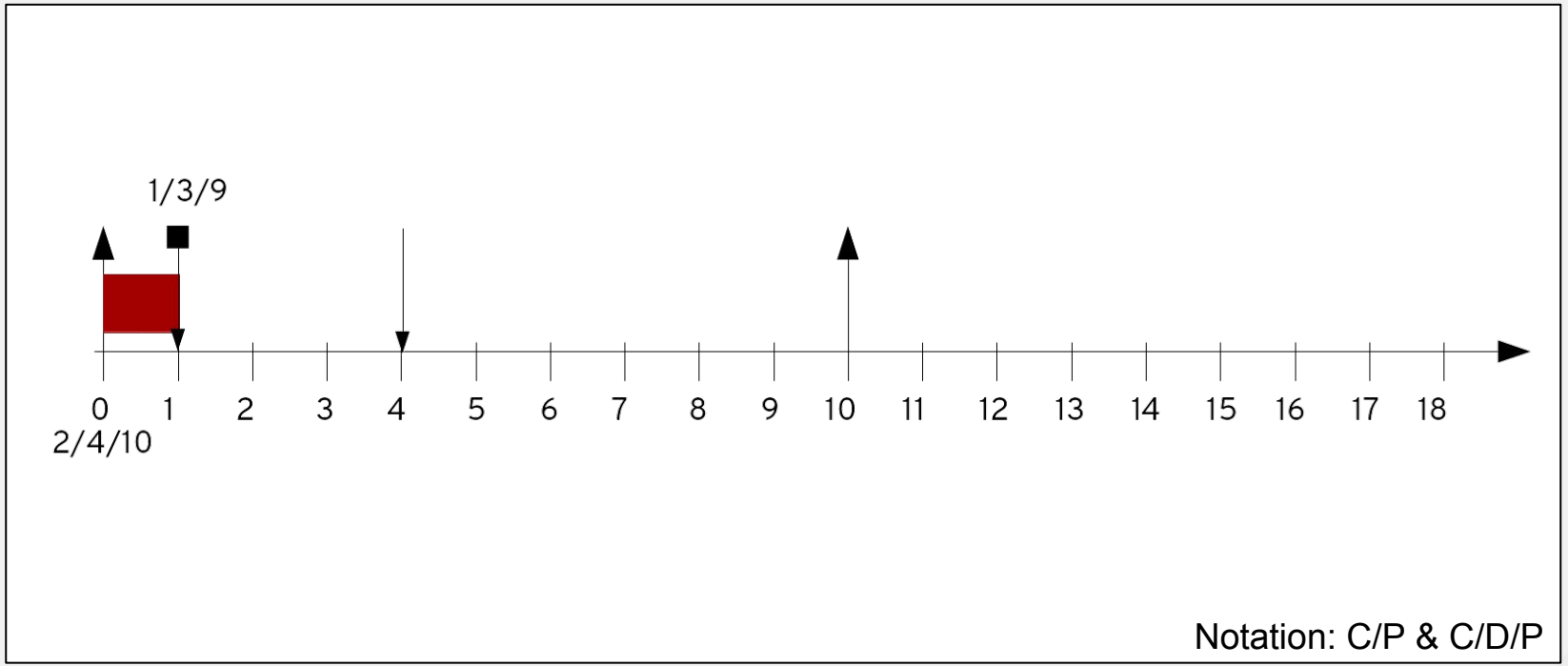


There is one case in which we decided to use it, with revised CBS...

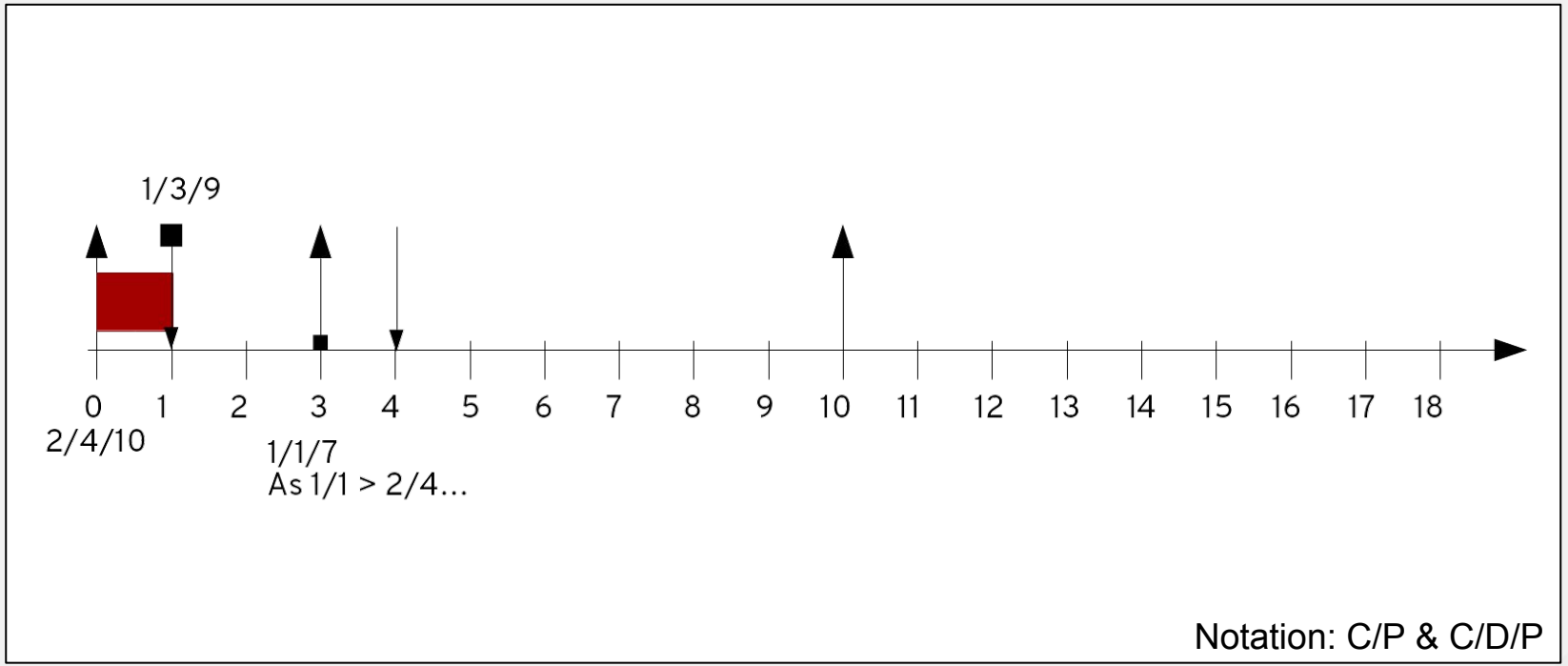
Self-suspending constrained deadline task



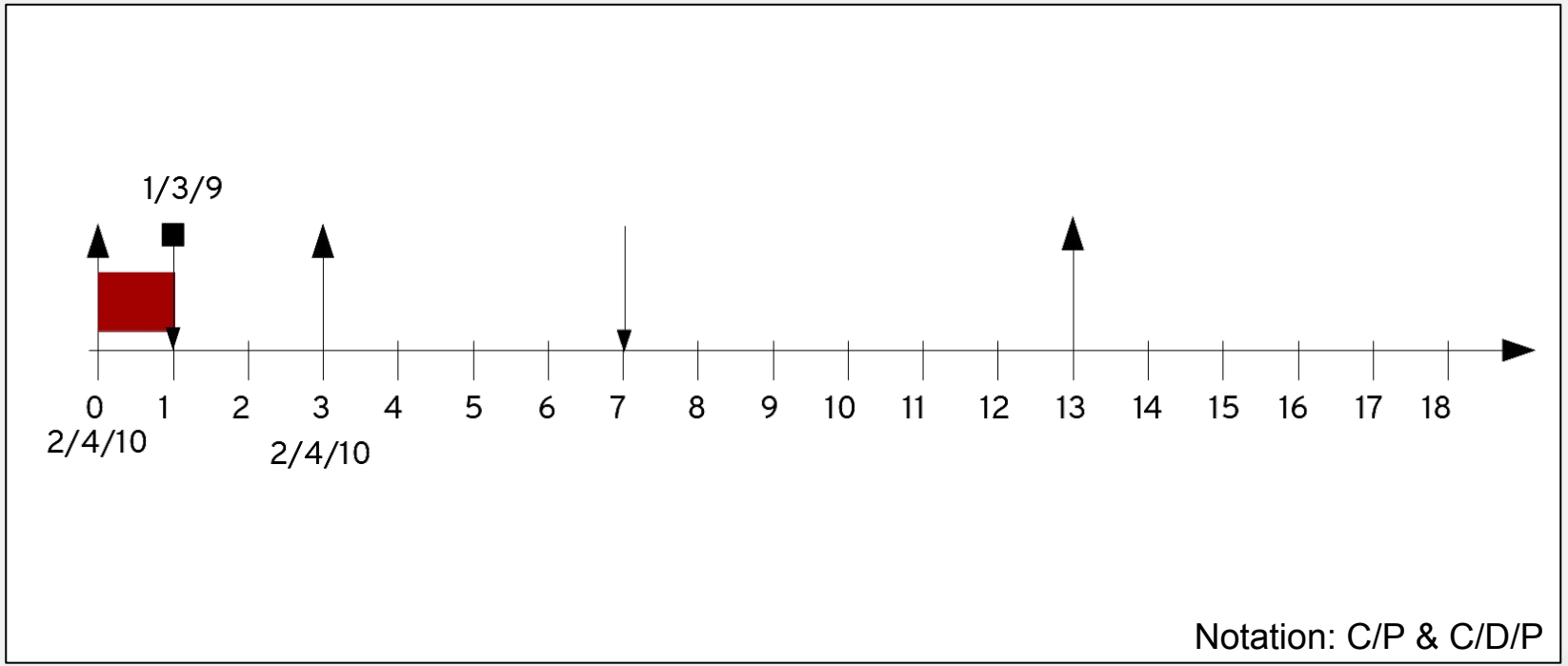
Self-suspending constrained deadline task



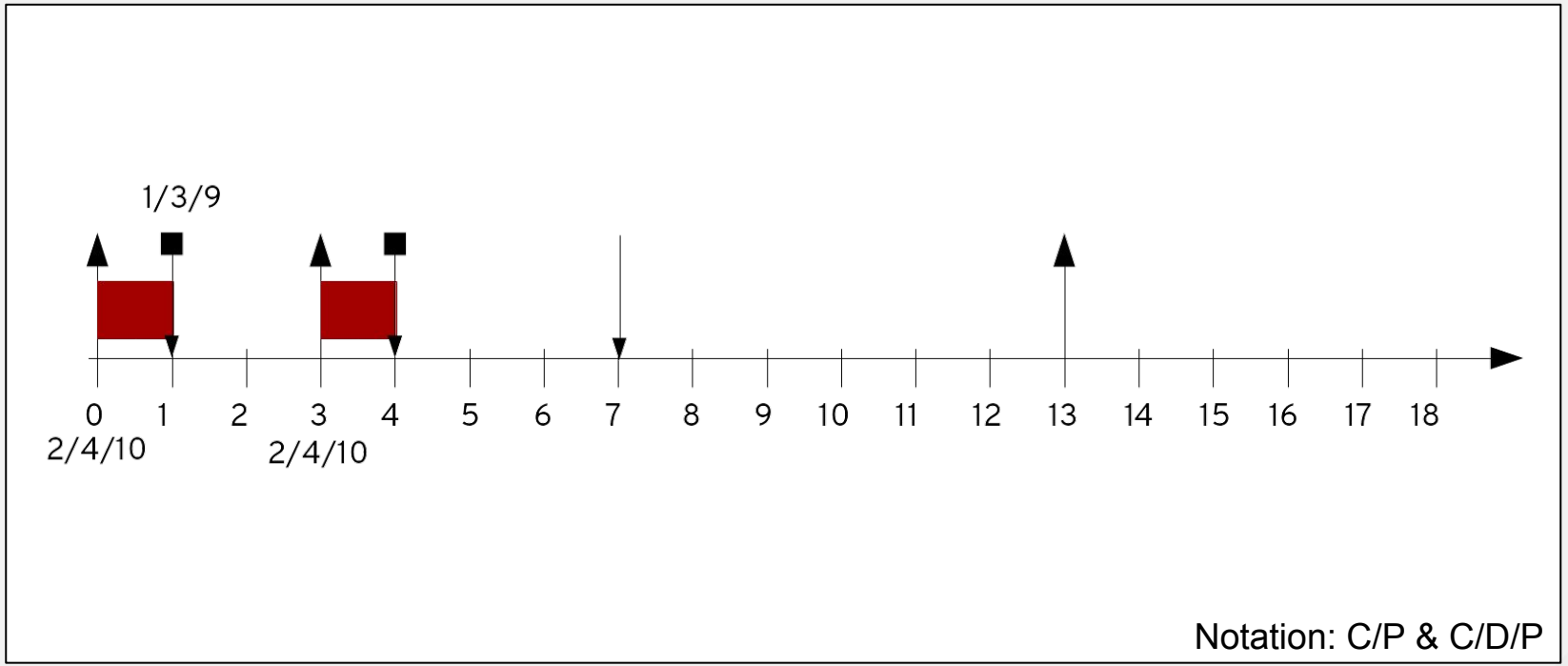
Self-suspending constrained deadline task



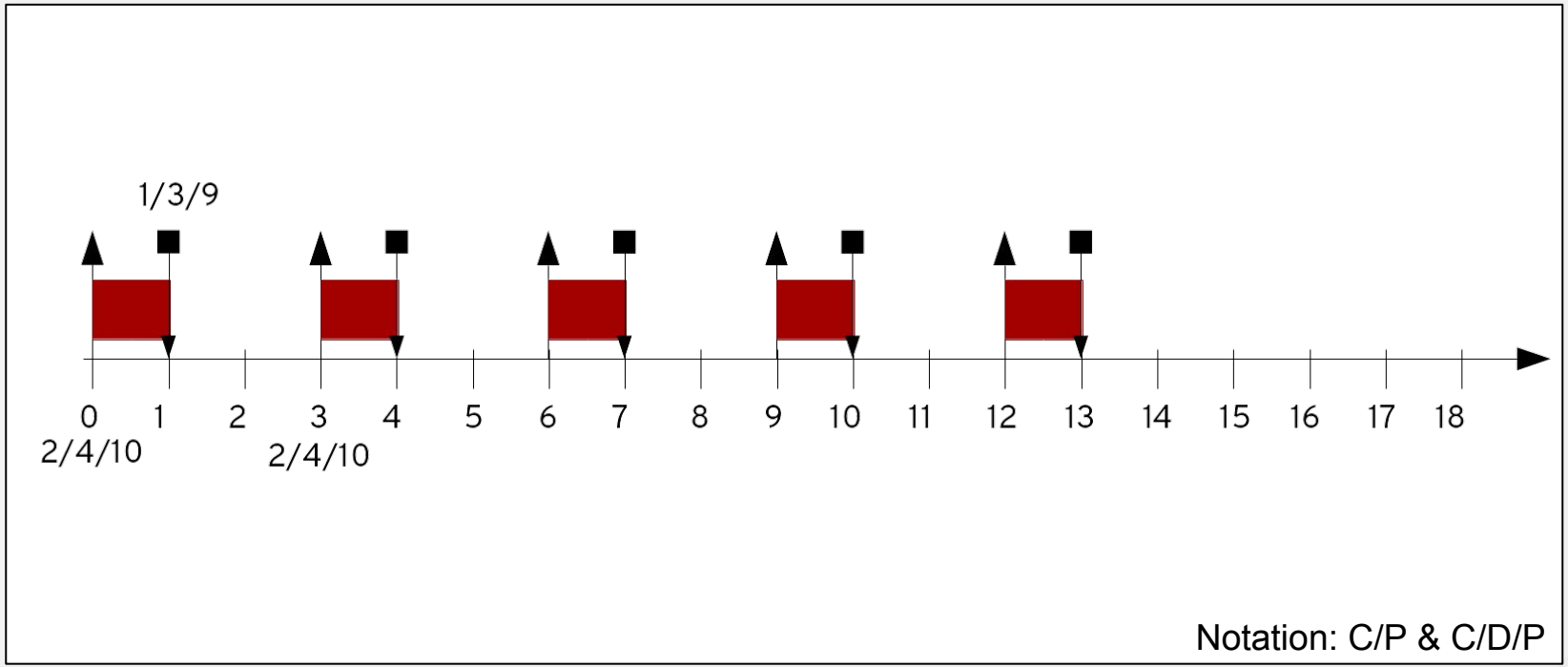
Self-suspending constrained deadline task



Self-suspending constrained deadline task



Self-suspending constrained deadline task



Revised CBS & Suspending & Constrained DL

- CBS wakeup rule (ensures that a task will not overload the system):

    If the **deadline** is in the **past**:

        If the next period is in the future:

            Throttle waiting the next period;

        else

            new absolute **runtime** and absolute **deadline** is set.

    If the deadline is in the future:

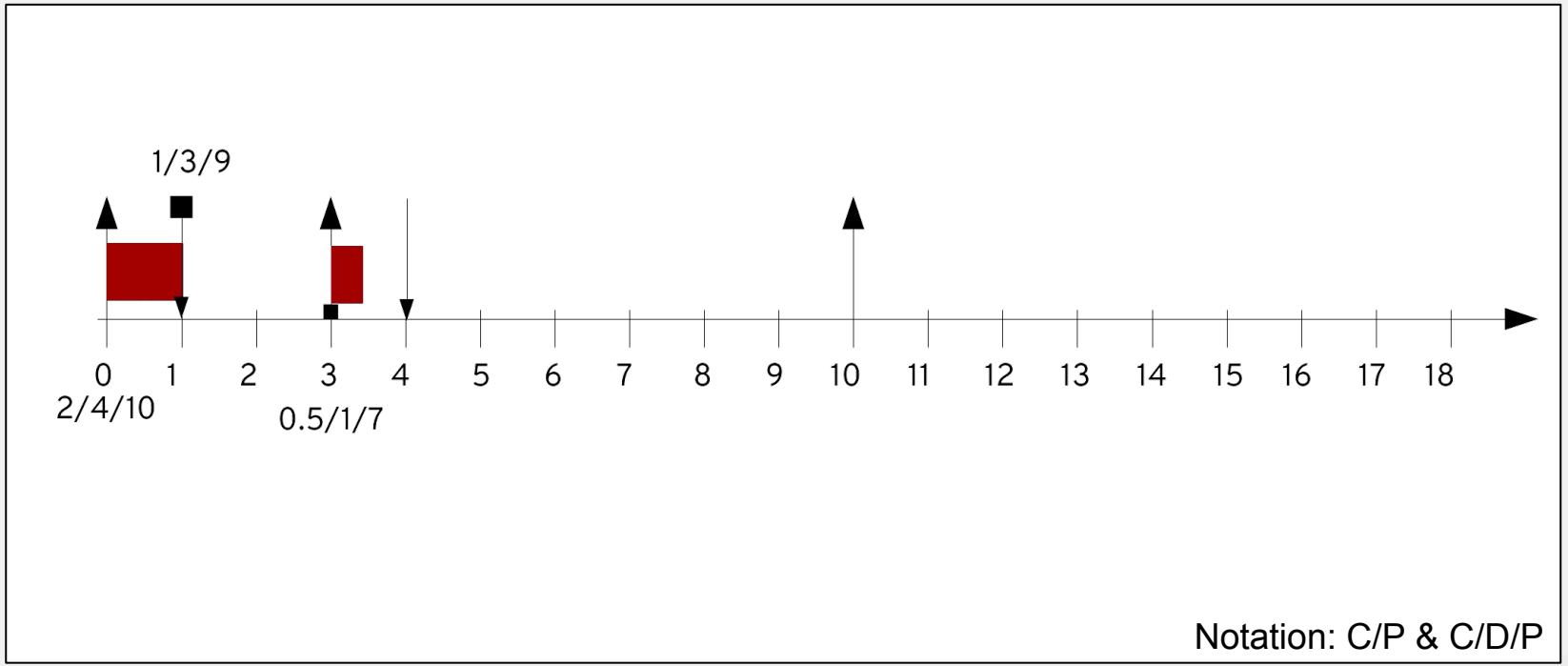
        If the **absolute Density** is < **relative Density**

            Go ahead and run, my little CBS.

        else

            Truncate runtime, new runtime = (C / D) * laxity

Self-suspending constrained deadline task



Mamma mia!
Things are confuse for deadline < period
& Self-suspending!?!?!?!

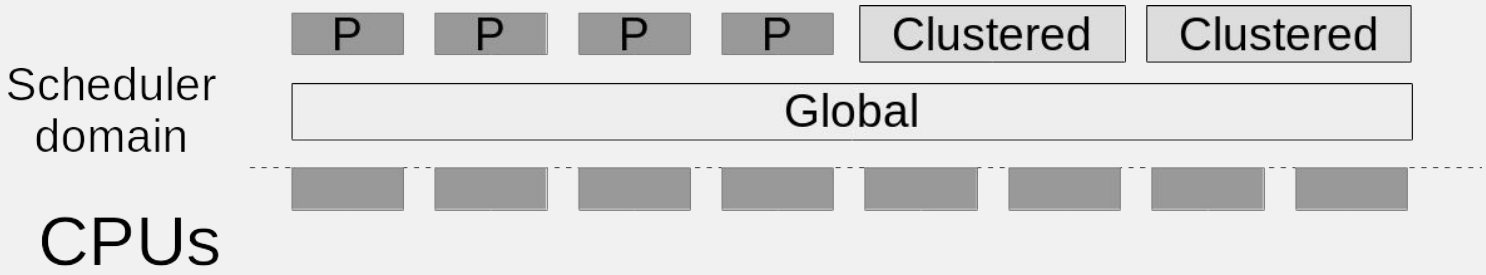
Suspending + constrained deadline
is a REAL open issue.

Let’s talk about multi-processor scheduling

Multi-processor scheduling

a scheduler can be classified as:

- **Partitioned:** When each scheduler manages a single CPU
- **Global**: When a single scheduler manages all M CPUs of the system
- **Clustered:** When a single scheduler manages a disjoint subset of the M CPUs
- a CPU cannot belong to two “domains”.



Let’s talk about global scheduling!

Global scheduling

Global scheduling adds a lot of anomalies.

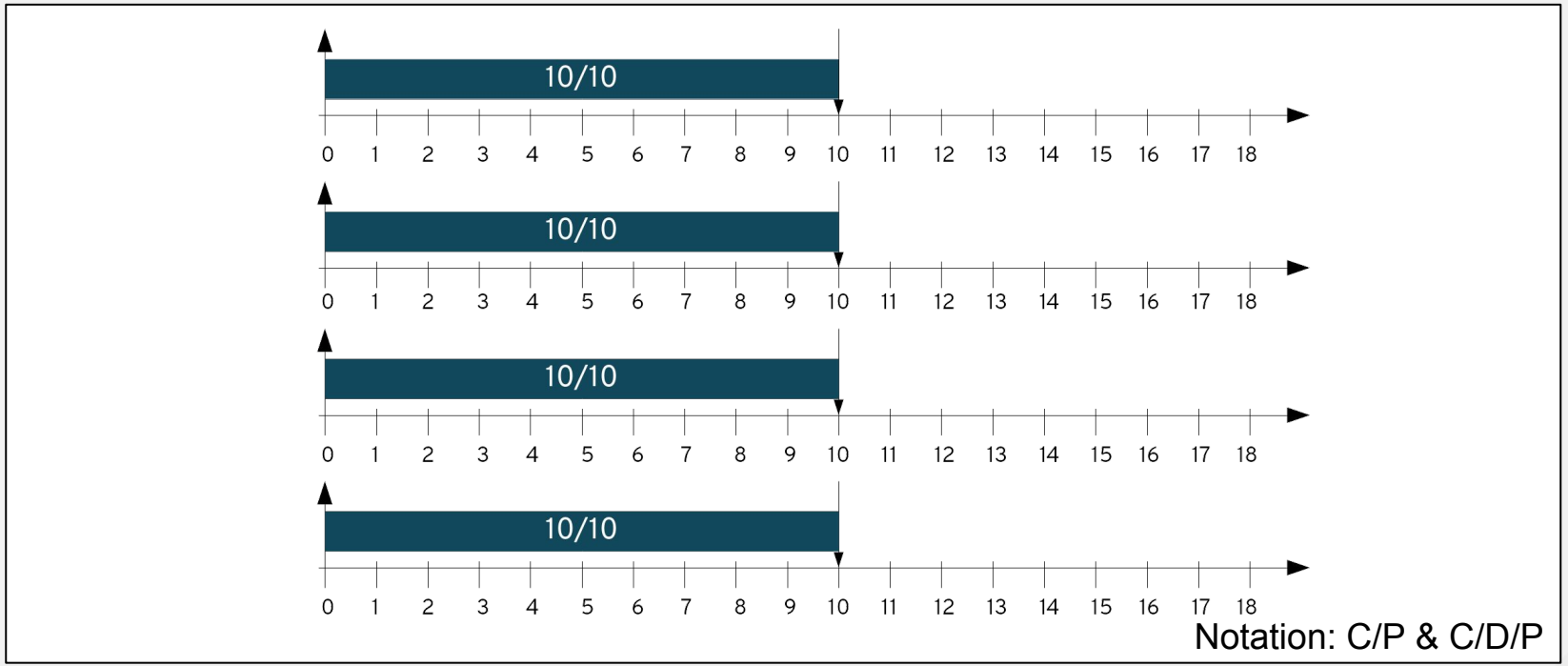
For instance, there is no critical instant.

- Release all tasks at same time is not the worst case anymore

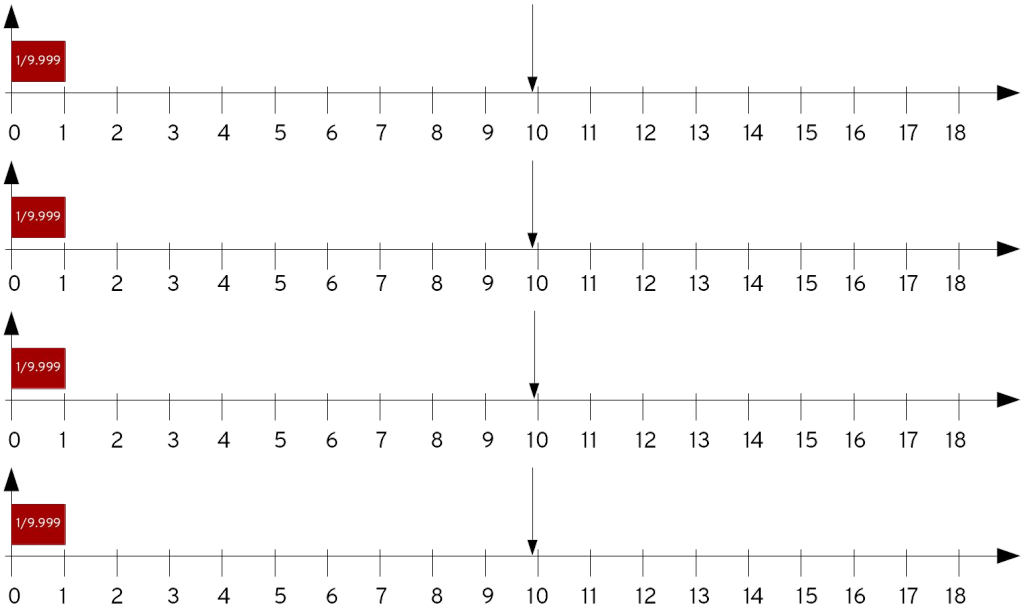
“Obvious things” are not obvious anymore:

- Reducing the load of a schedulable taskset does not turn guarantee the task set will still schedulable...

Dhall's effect

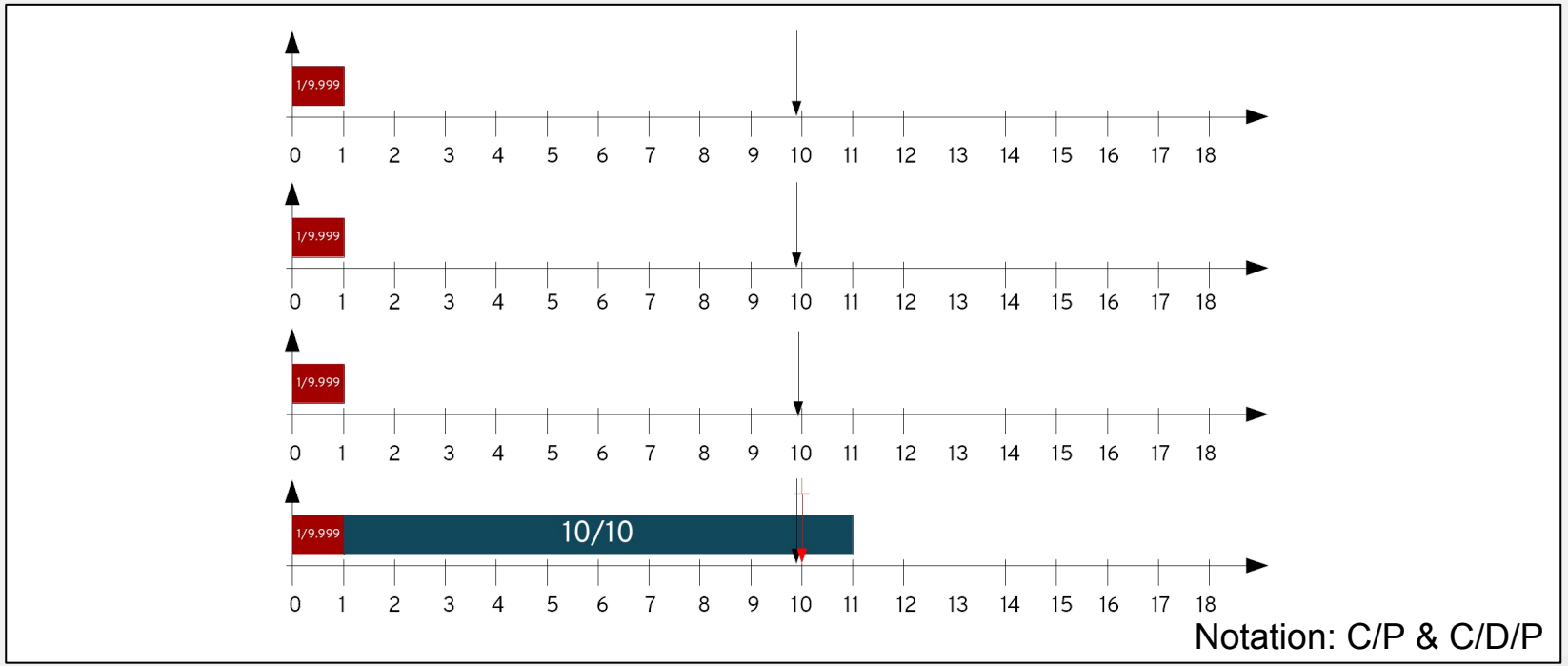


Reducing the load...



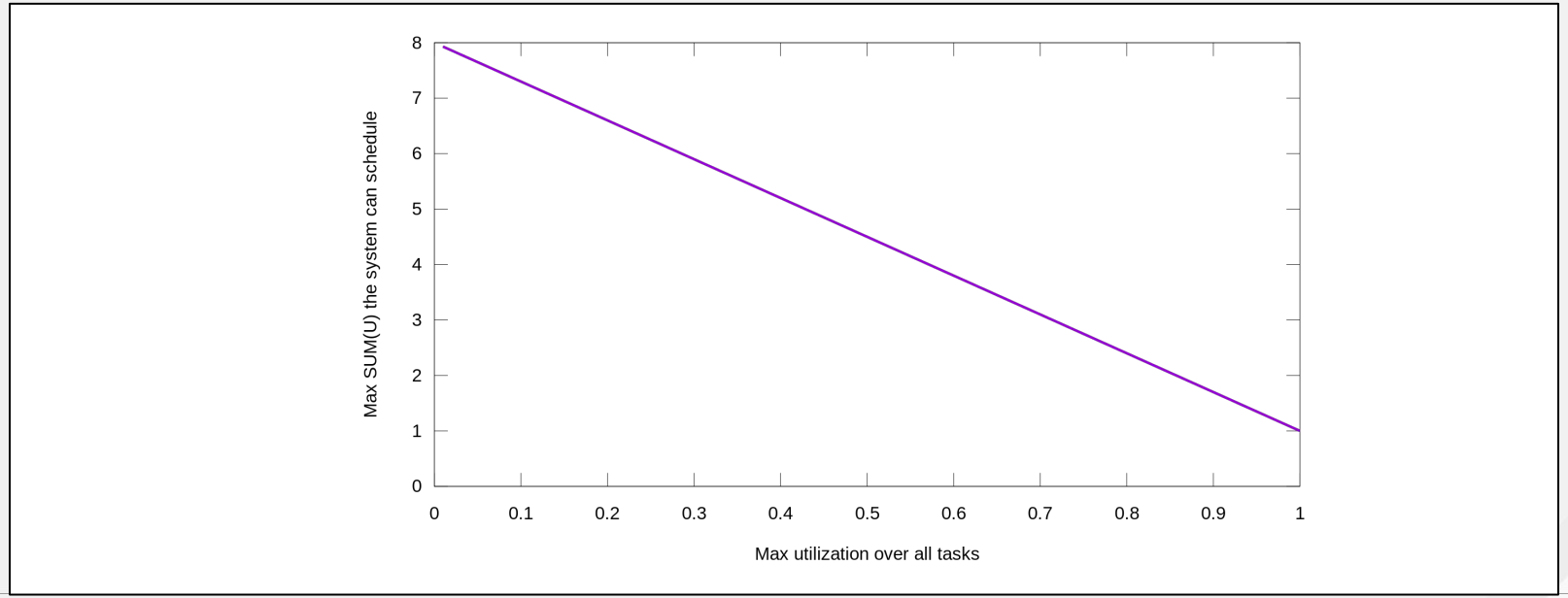
Notation: C/P & C/D/P

Increasing a little bit... BOOM!

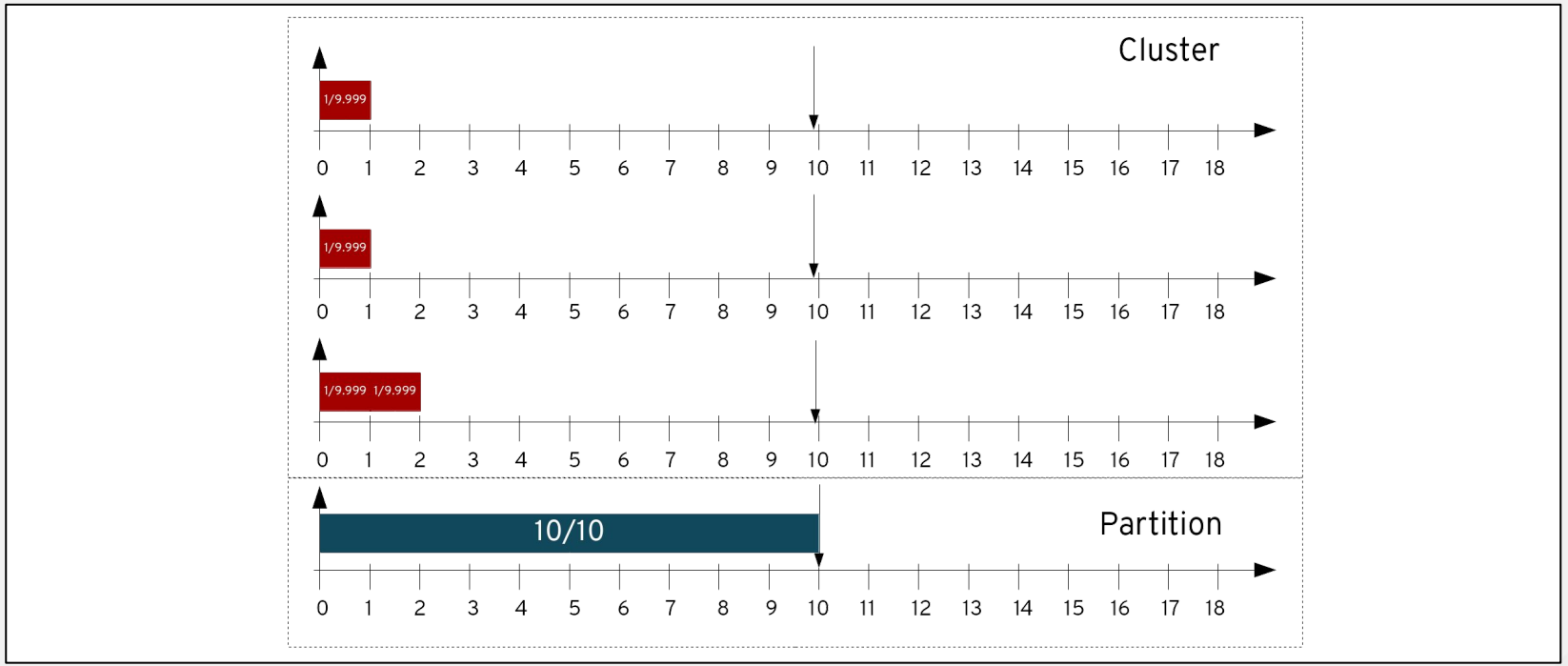


Taking Dhall's effect in account, an admission test would be:

- $\sum(U) <= M - (M - 1) * U_{max}$
- Where U_{max} is the highest U of all tasks



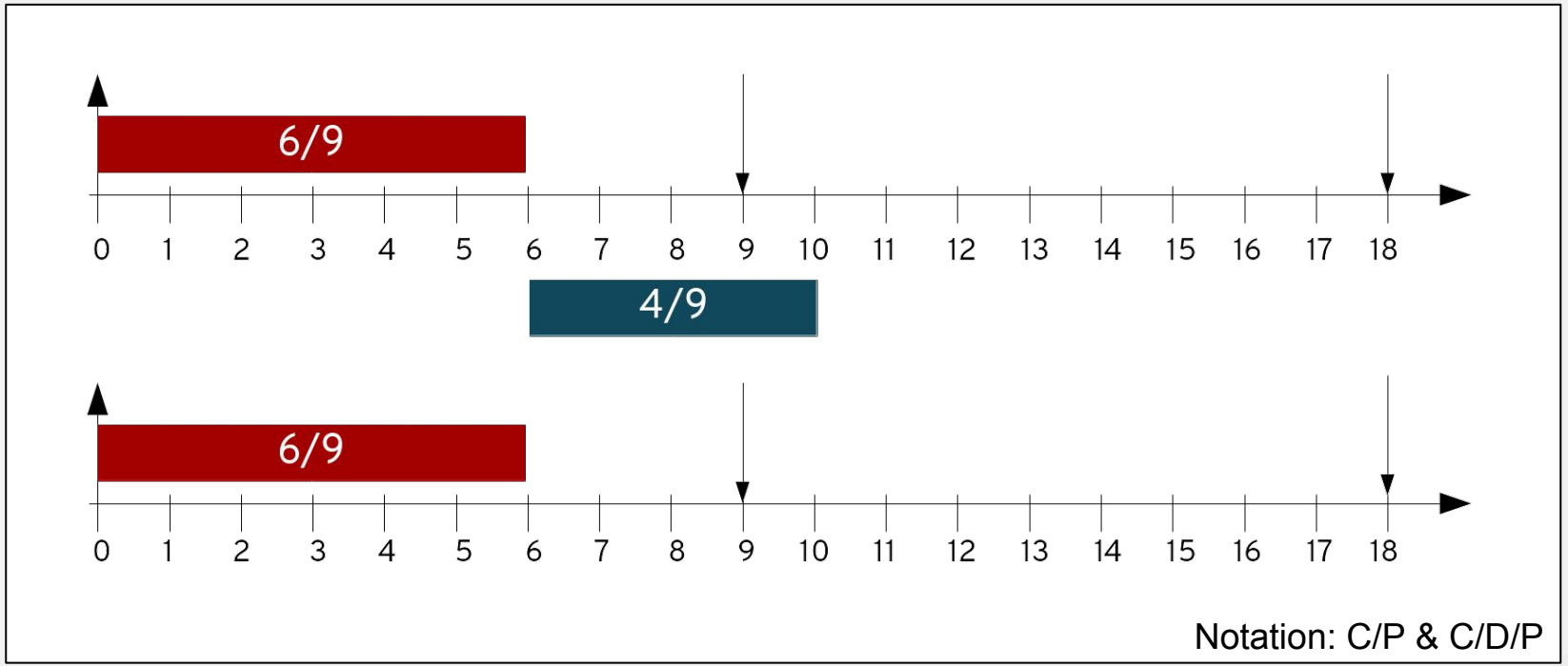
Solution: Partitioned + Clustered



What if those small tasks were per-cpu tasks?

So should we always use partitioned?

How about this scenario?

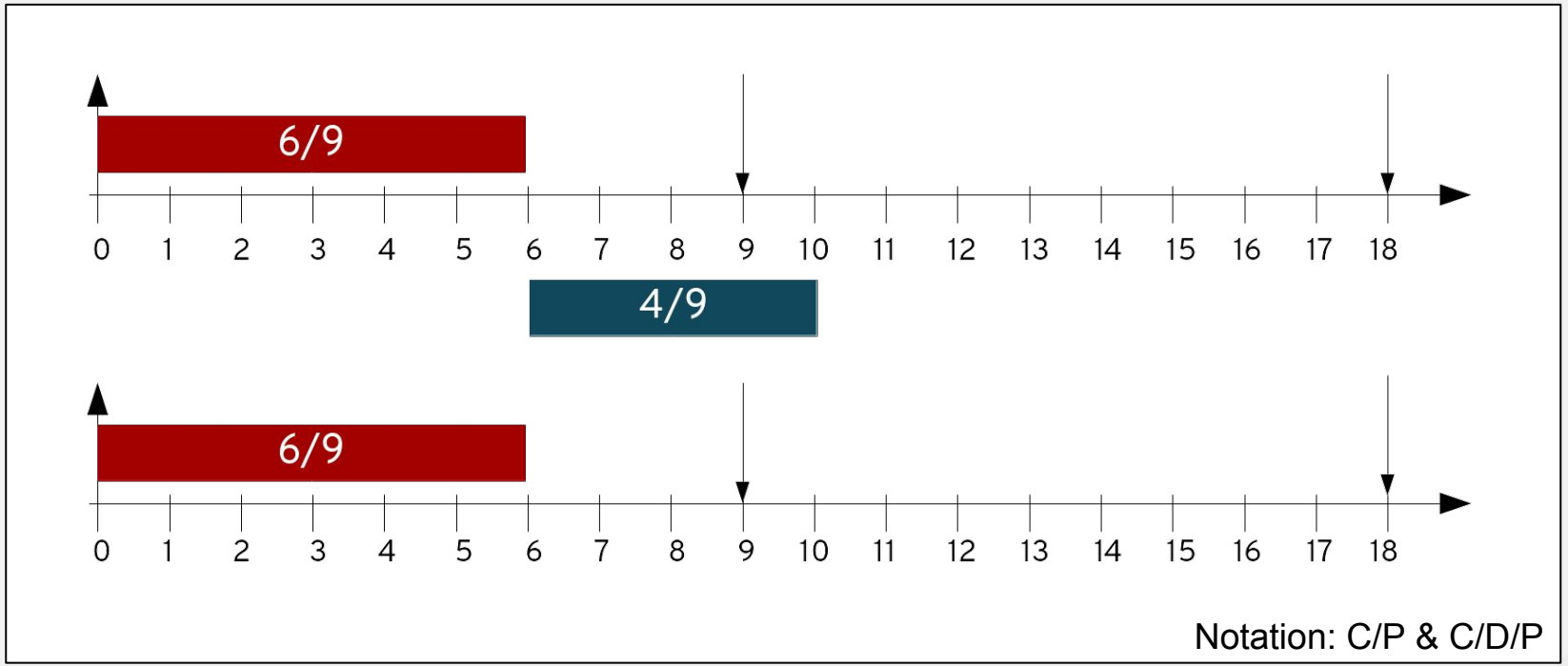


Neither partitioned nor global are optimal...

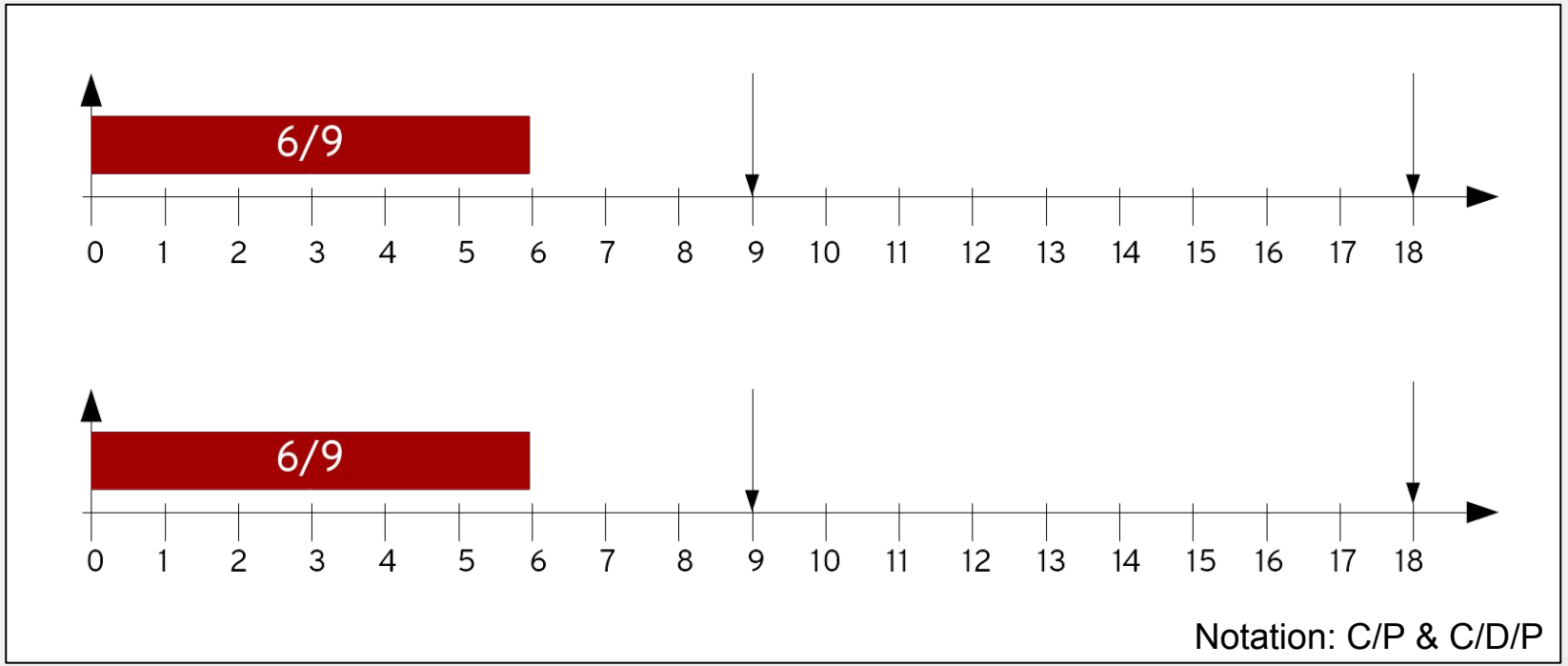
Is there anything else we could?

The word is: semi-partitioned

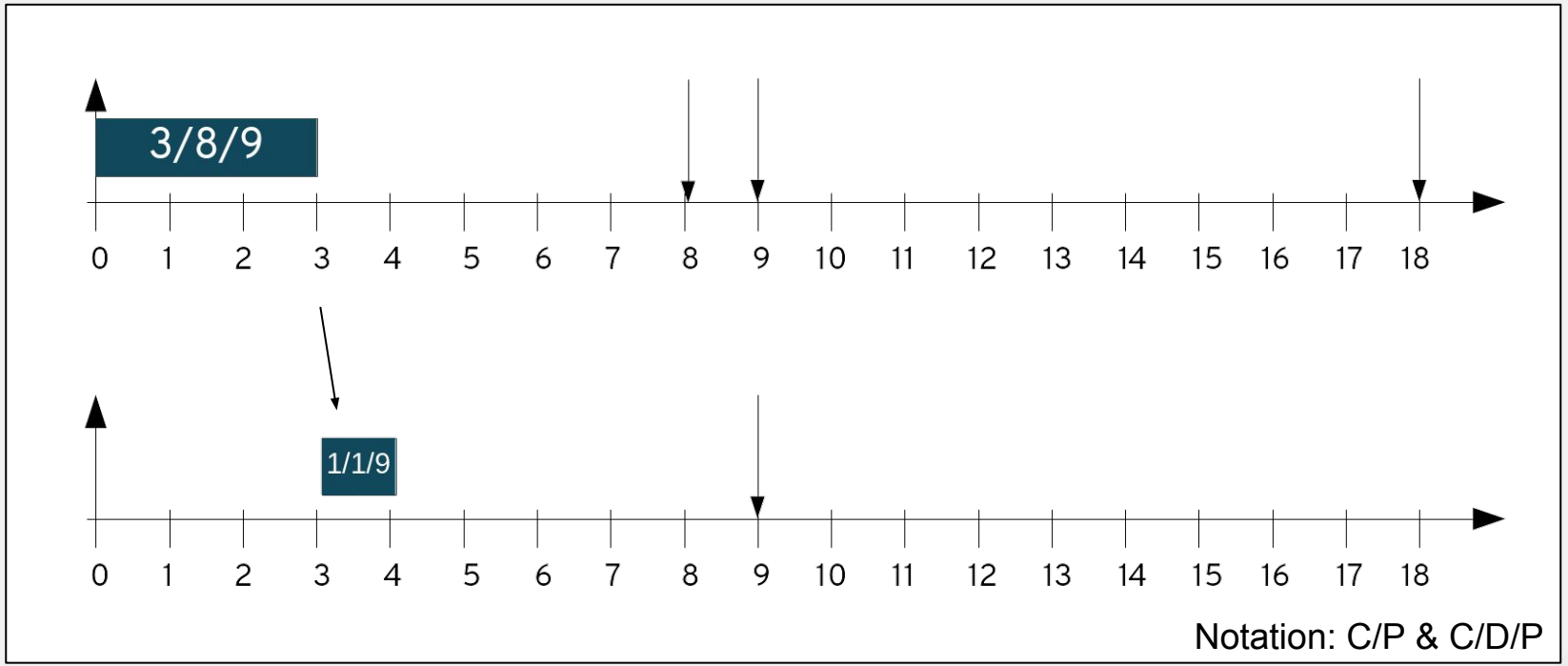
Let’s take this scenario:



Let’s pin some tasks:

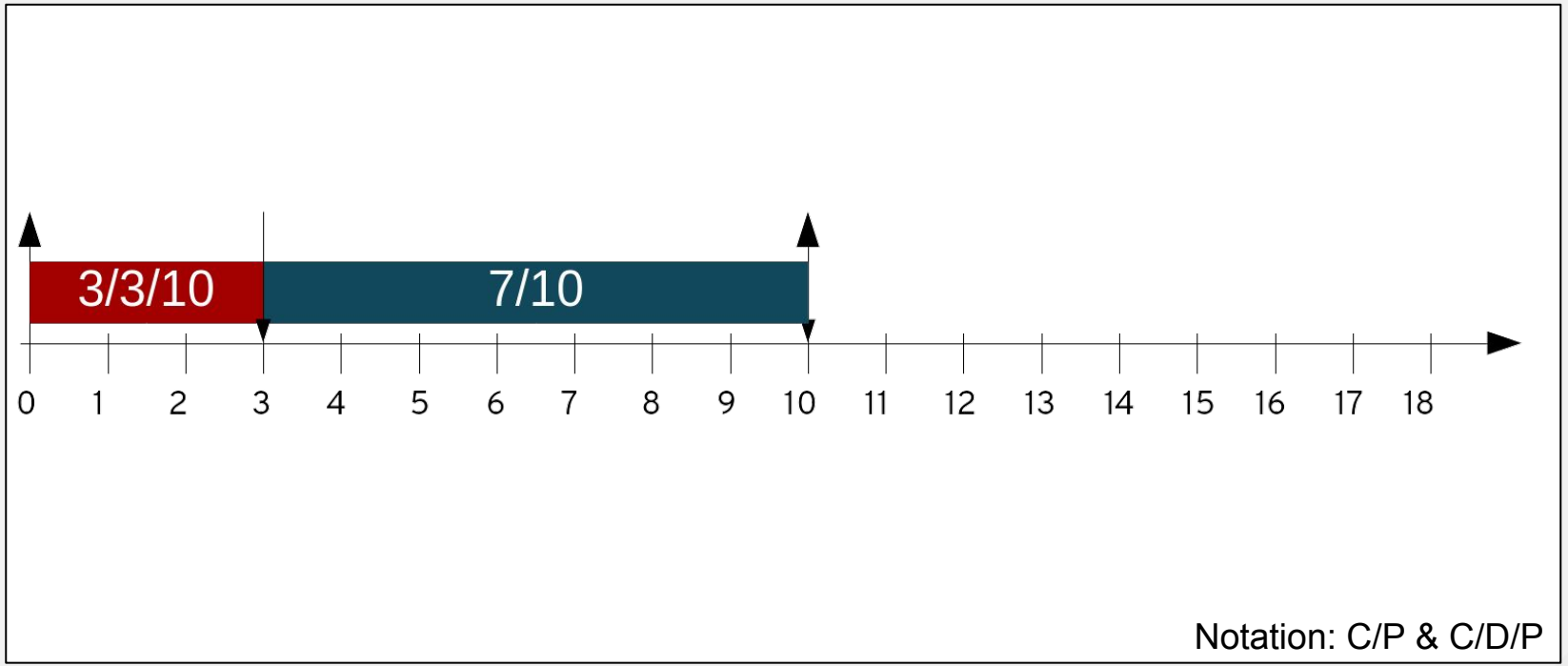


Then, we split the other one....

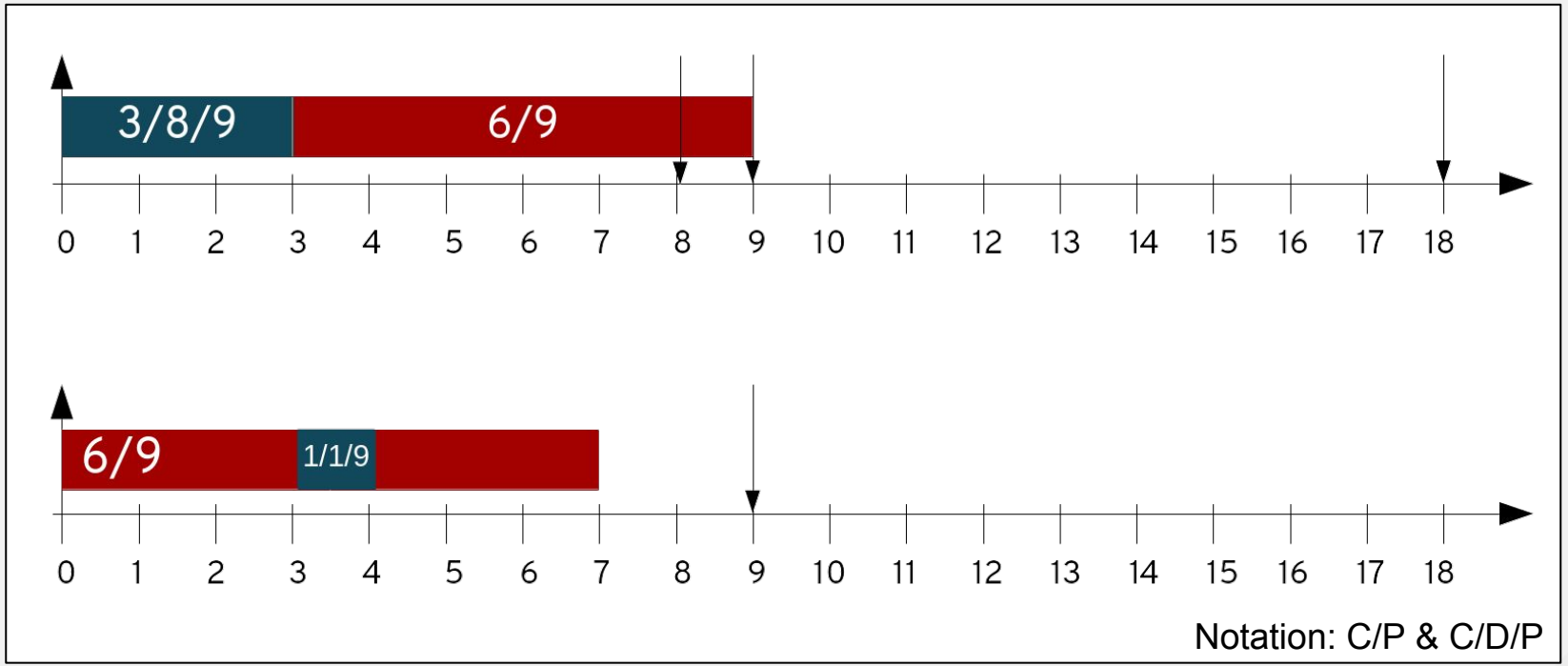


Hey hey hey! Didn’t you say constrained deadline tasks are a problem?

They are not always a problem:



And voilà!


3/8/9
6/9
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18
6/9
1/1/9
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18
Notation: C/P & C/D/P

How good is this idea?

B. Brandenburg and M. Gül

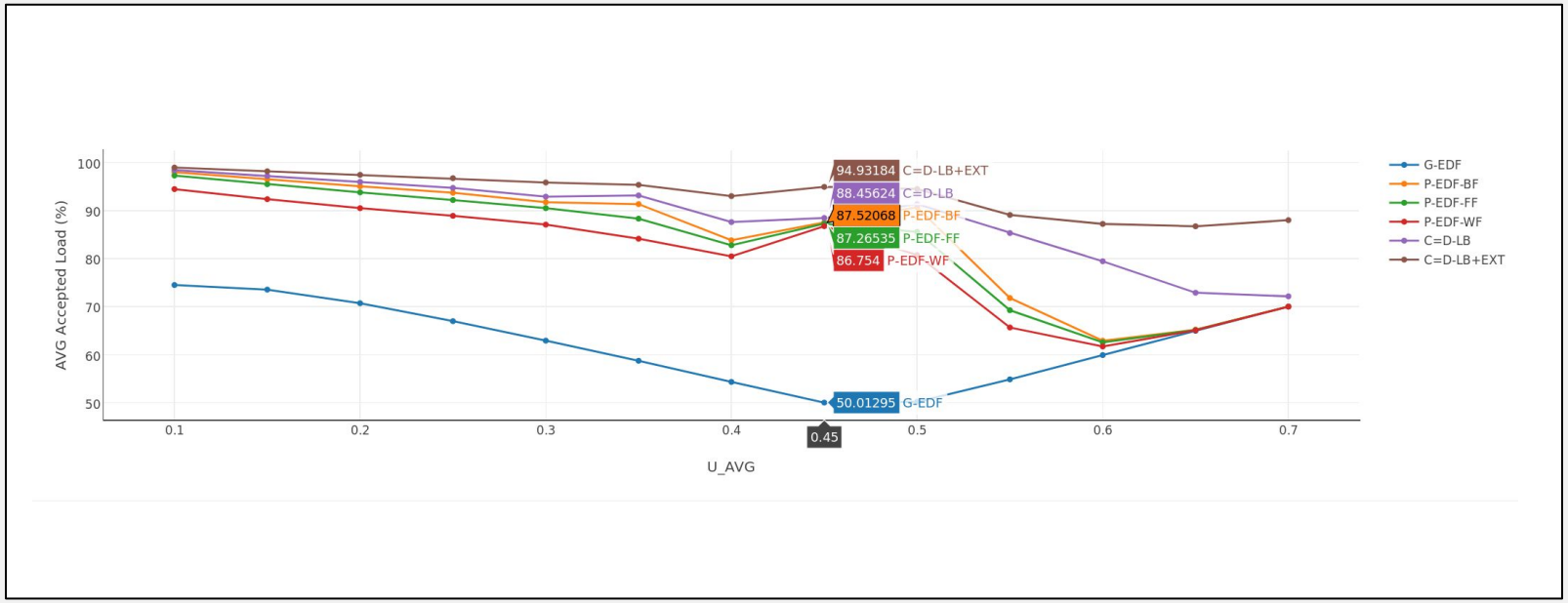
Global Scheduling Not Required: Simple, Near-Optimal Multiprocessor Real-Time Scheduling with Semi-Partitioned Reservations:

"*Empirically, near-optimal hard real-time schedulability — usually ≥99% schedulable utilization — can be achieved with simple, well-known and well-understood, low-overhead techniques (+ a few tweaks).*"

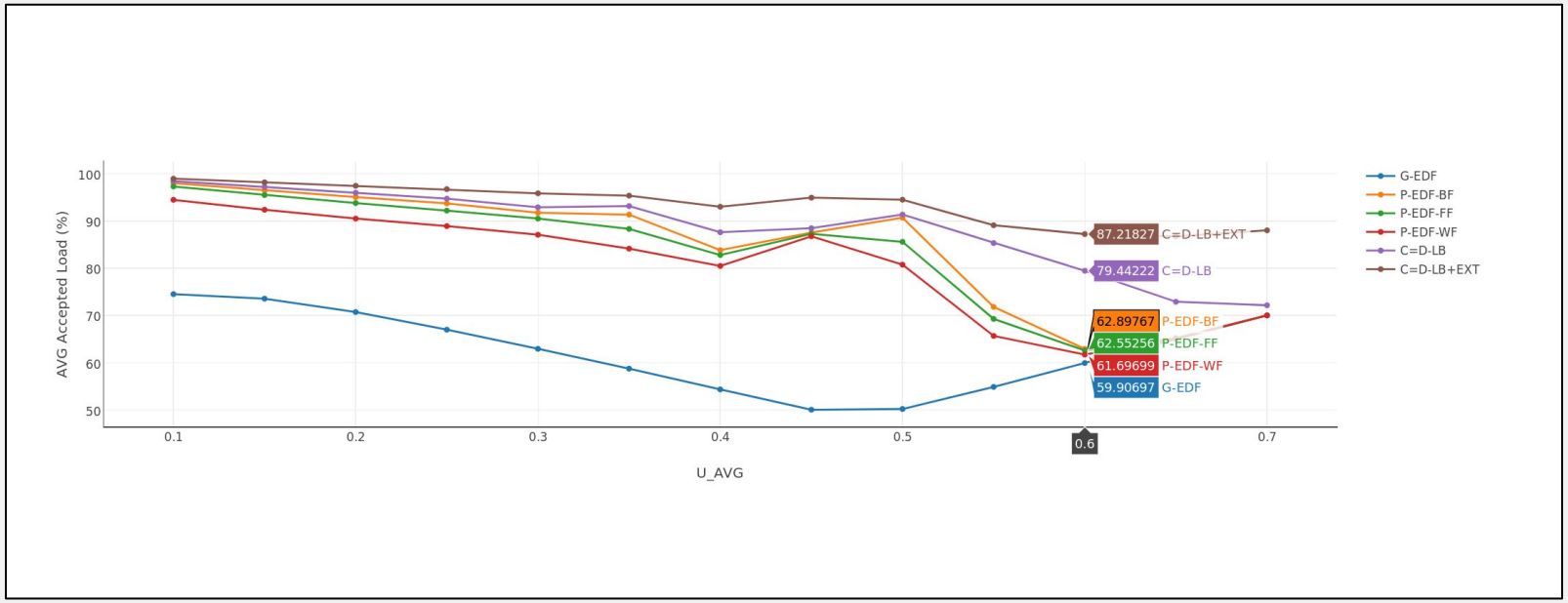
Daniel Casini, Alessandro Biondi, Giorgio Buttazzo

Semi-Partitioned Scheduling of Dynamic Real-Time Workload: A Practical Approach Based on Analysis-Driven Load Balancing.

Online semi-partitioned comparison:



Online semi-partitioned comparison:



Affinity! For almost free

Affinity for global scheduling is a problem

For semi-partitioned... it is not.

- Just one more input to the heuristics
- Possible to make a “per-cpu fake load” to reserve time for CFS
- DL Server to schedule CFS: Hierarchical scheduler
    - A re-implementation of RT Throttling:
        - [PATCH] sched/rt: RT_RUNTIME_GREED sched feature
        - https://lkml.org/lkml/2016/11/7/55

We still have arguments for another talk

But I am being throttled...

Questions?

Thank you! Obrigado! Grazie Mille!