Semi-Partitioned Scheduling of Dynamic Real-Time Workload

Daniel Casini, Alessandro Biondi, and Giorgio Buttazzo
Scuola Superiore Sant'Anna

ReTiS Laboratory

Pisa, Italy

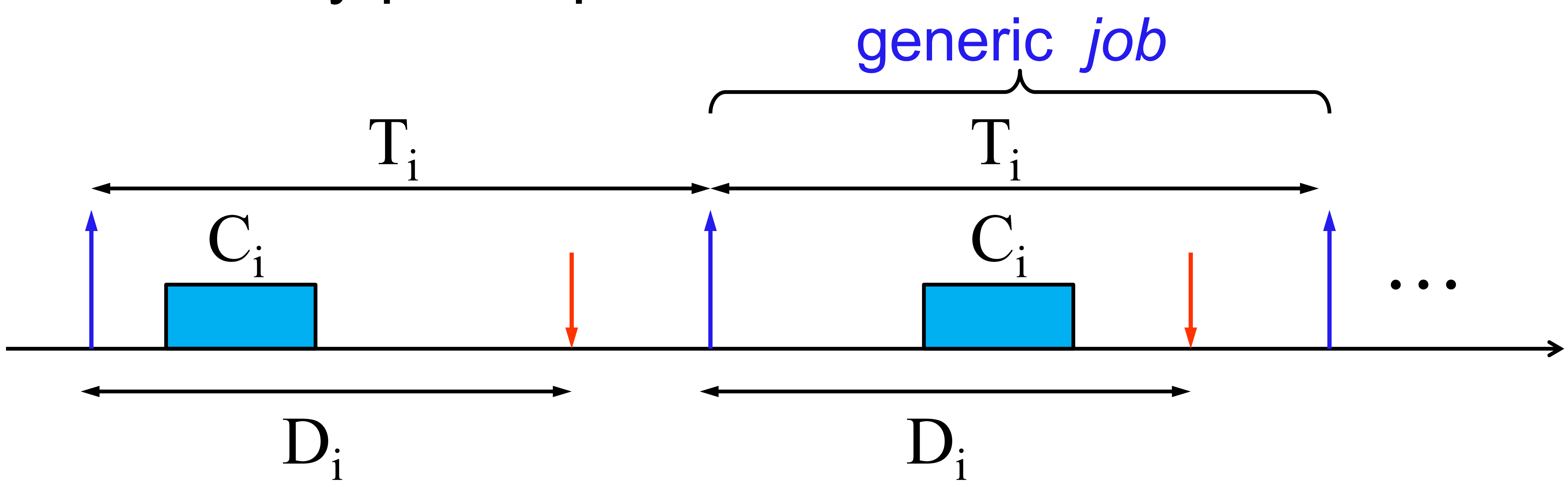
Task model

Real-time workload consists of a set of cyclic tasks, each characterized by:

Task utilization $U_i = \frac{C_i}{T_i}$

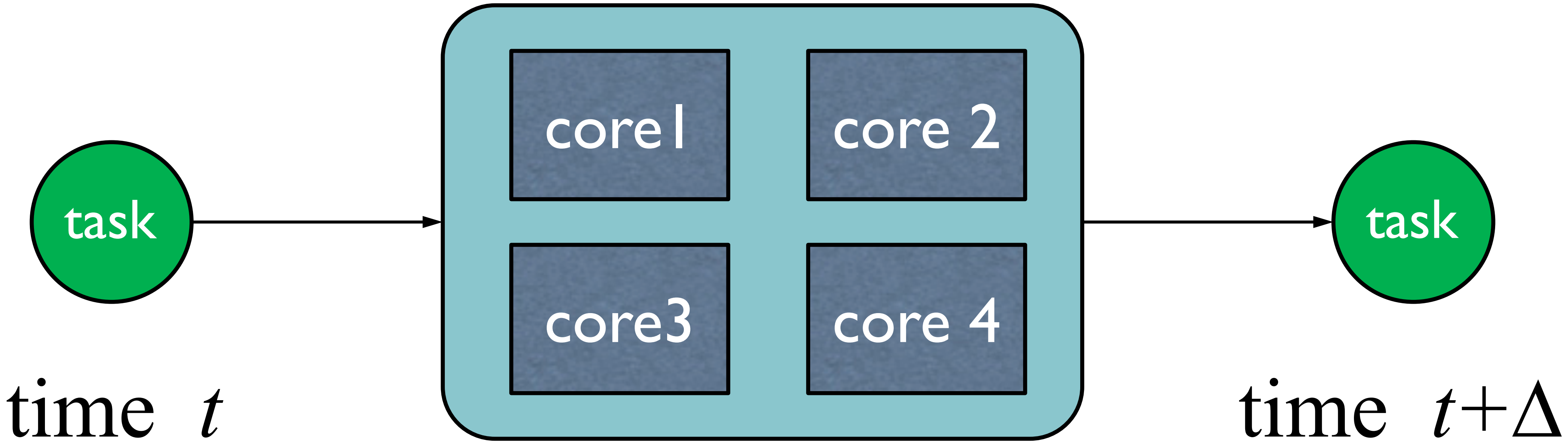
C_i	worst-case computation time
T_i	activation perdiod
D_i	relative deadline

- Each task generates an infinite sequence of instances (jobs), activated periodically or sporadically
- Jobs are fully preemptive



Dynamic real-time workload

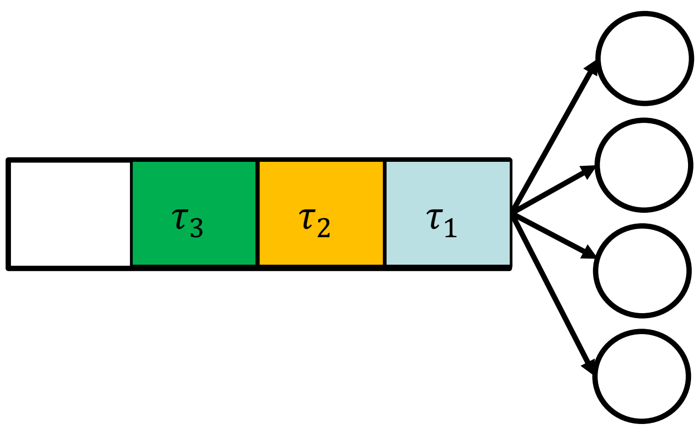
Real-time tasks can **join** and **leave** the system **at runtime**:

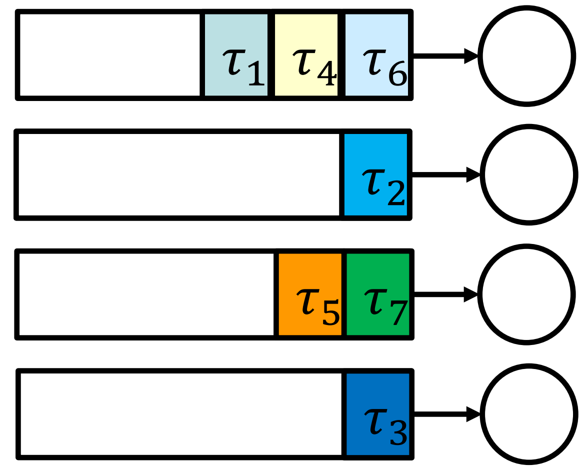


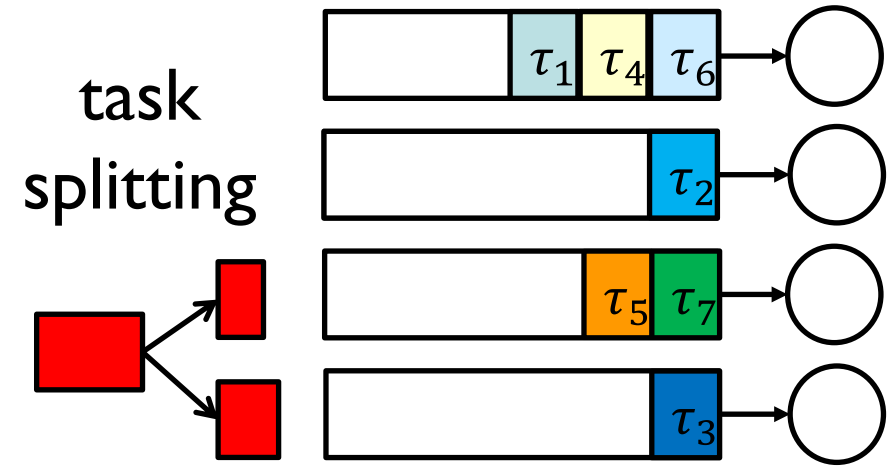
No **a-priori knowledge** of the workload

- Cloud computing, multimedia, real-time databases, …

Multiprocessor Scheduling





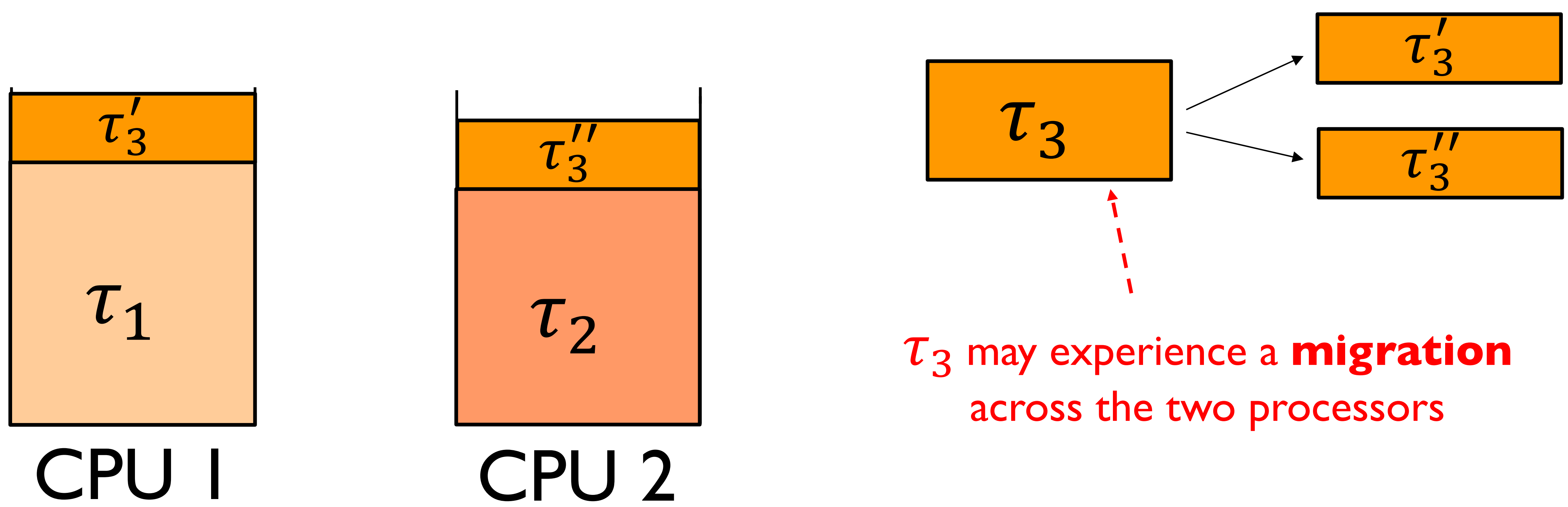


Global Scheduling	Partitioned Scheduling	Semi-Partitioned Scheduling
Full migration	No migration	Only some tasks migrate
✓ Auto. load balance	✗ No load balance	✓ Load balance
✓ High efficiency	✗ Low efficiency	✓ High efficiency
✗ High overhead	✓ Low overhead	✓ Low overhead
✗ Difficult to analyze	✓ Easy to analyze	✓ Easy to analyze

Semi-Partitioned Scheduling

Anderson et al. (2005)

- Builds upon partitioned scheduling
- Tasks that do not fit in a processor are **split** into **sub-tasks**



C=D Splitting

Burns et al. (2010)

- Task is split into multiple chunks, with the first n-1 chunks at zero-laxity (C = D)

Original task

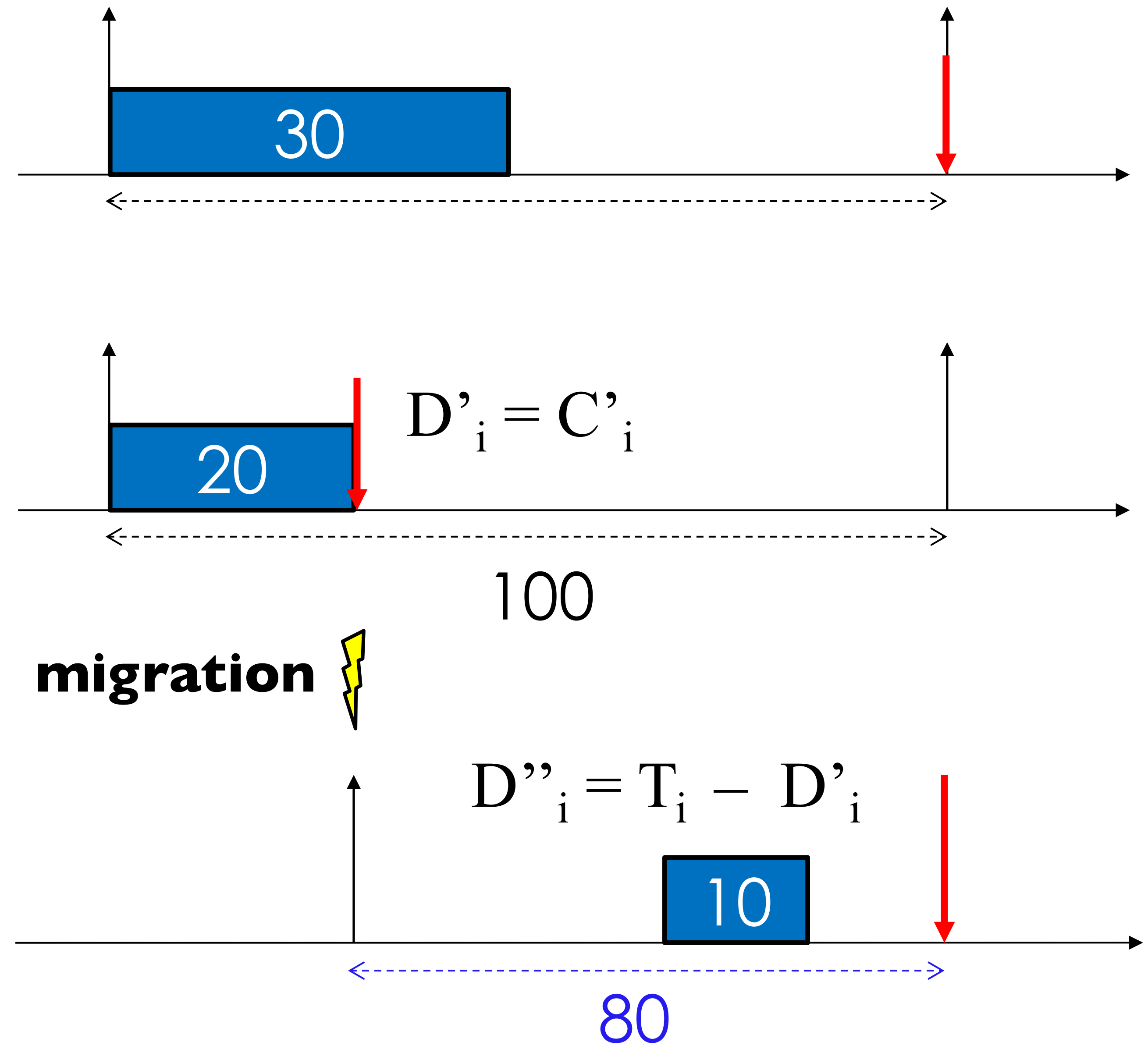
$\tau_3 = (30, 100, 100)$

Zero-laxity chunk

$\tau'_3 = (20, 20, 100)$

Last chunk

$\tau''_3 = (10, 80, 100)$



A very important result

Brandenburg and Gül (2016)

"Global Scheduling Not Required"

Empirically, near-optimal schedulability (99%+) achieved with simple, well-known and low-overhead techniques

- Based on C=D Semi-Partitioned Scheduling
- Performance achieved by applying multiple clever heuristics (off-line)

Conceived for **static** workload

Semi-Partitioned Scheduling

- ✔ More predictable execution
- ✔ Reuse of results for uniprocessors
- ✔ Excellent worst-case performance
- ✔ Low overhead
- ✘ A-priori knowledge of the workload
- ✘ High complexity for optimal splitting

HOW TO MAINTAIN THE BENEFITS OF SEMI-PARTITIONED SCHEDULING WITHOUT REQUIRING ANY OFF-LINE PHASE?

How to partition and split tasks online?

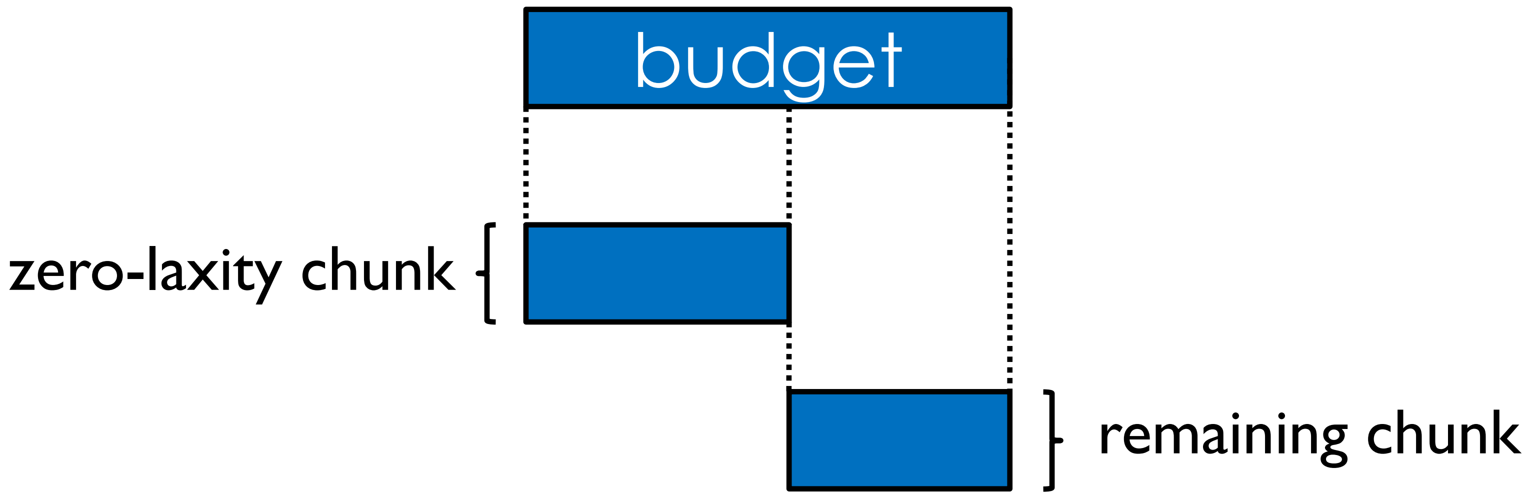
This work

- This work considers dynamic workload consisting of reservations (budget, period)

  This model is compliant with Linux (SCHED_DEADLINE), hence usable in billions of devices around the world

- The workload is executed under C=D Semi-Partitioned Scheduling

  - Budget splitting

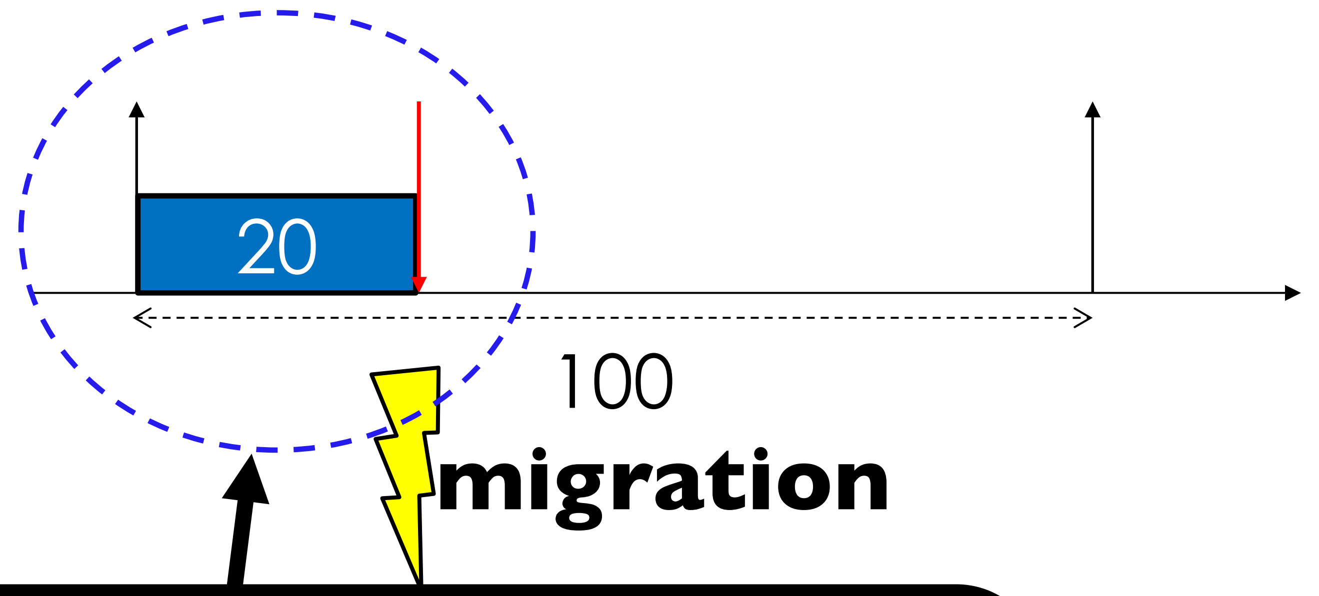


C=D Budget Splitting

$\tau = (\text{budget} = 30, \text{period} = 100)$

to be split

$\tau' = (20, 20, 100)$



$\tau'' =$

How to find a **safe** zero-laxity budget?

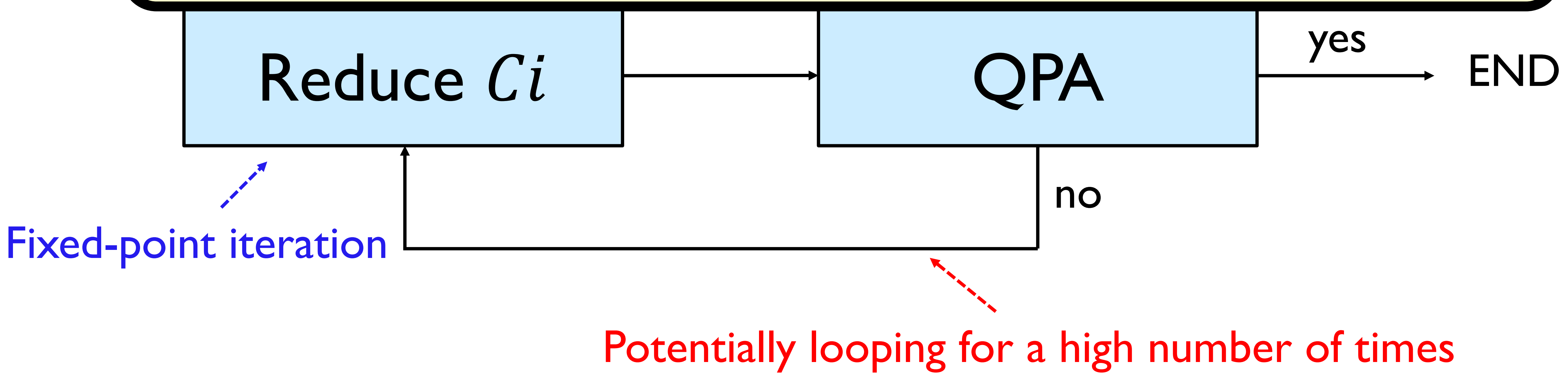
How to find the zero-laxity budget?

Burns et al. (2010)

- Iterative process based on **QPA** (*Quick Processor-demand Analysis*) with high complexity (no bound provided by the authors)

- Also used by Brandenburg and Gül (2016)

Unsuitable to be performed **online**!



Our approach: approximated C=D

Main goal: Compute a safe bound for the zero-laxity budget in linear time

- In this work we proposed an approximate method based on solving a system of inequalities

Constants depending on static task-set parameters

$$\begin{cases} C' = D' \leq K_1 \\ \cdots \\ C' = D' \leq K_N \end{cases} \Rightarrow C' = \min(K_1, \ldots, K_N)$$

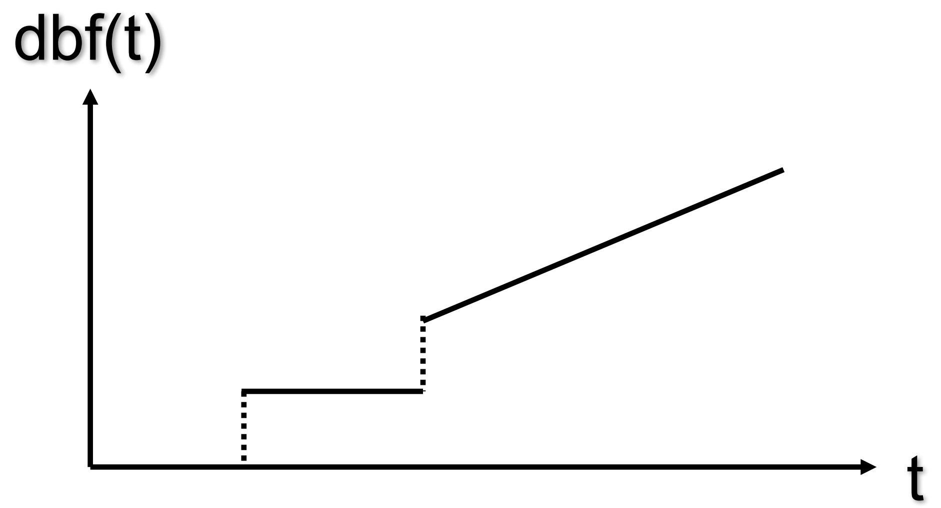
order of number of tasks

Our approach: approximated C=D

How have we achieved the closed-form formulation?

- Approach based on approximate demand-bound functions

  Some of them similar to those proposed by *Fisher et al.* (2006)



- + <u>theorems</u> to obtain a closed-form formulation

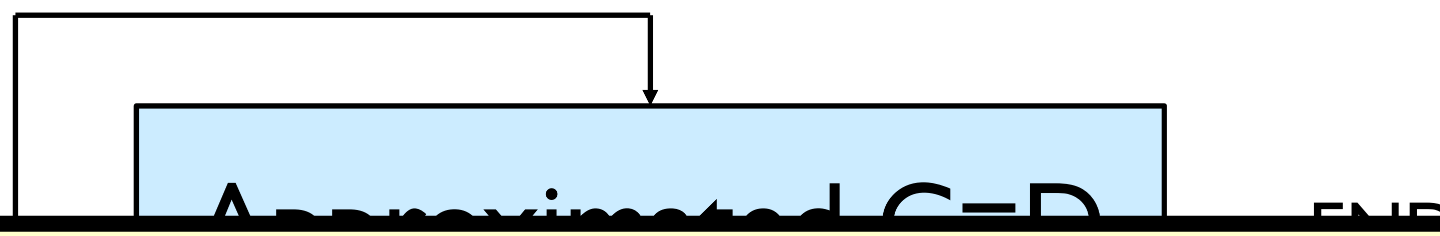
  The derivation of the closed-form solution has been also mechanized with the Wolfram Mathematica tool

Approximated C=D: Extensions

The approximation can be improved by:

- **Extension 1:** Iterative algorithm that refines the bound

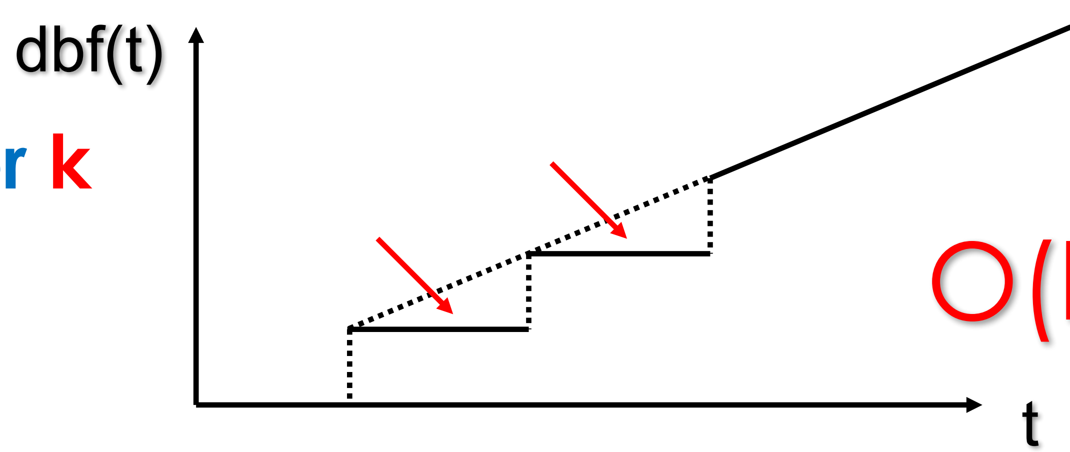
Repeats for a fixed



We found that **significant improvements** can be achieved with just **two iterations**

- **Extension 2:** Refinement on the precisions of the approximate dbfs

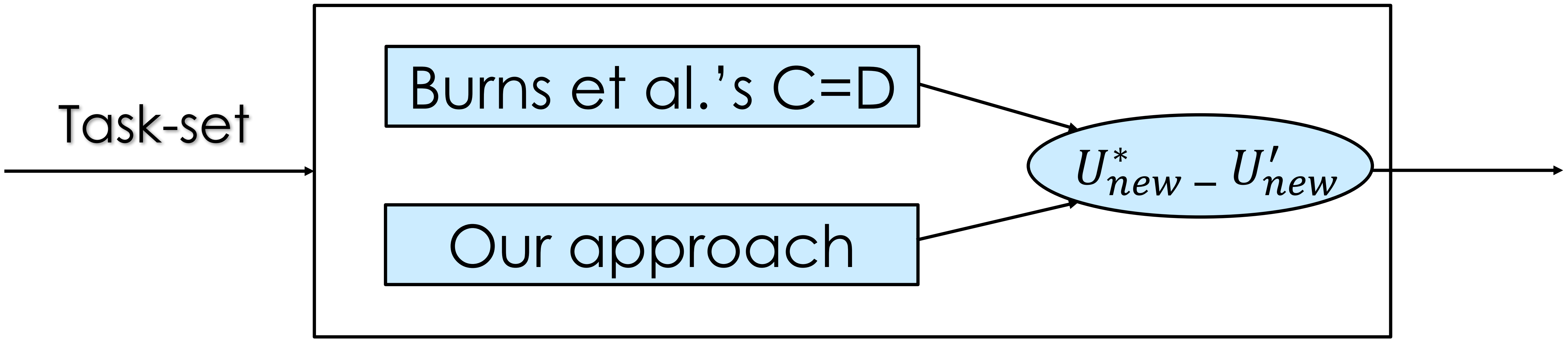
Add a fixed number k of discontinuities



O(k*n)

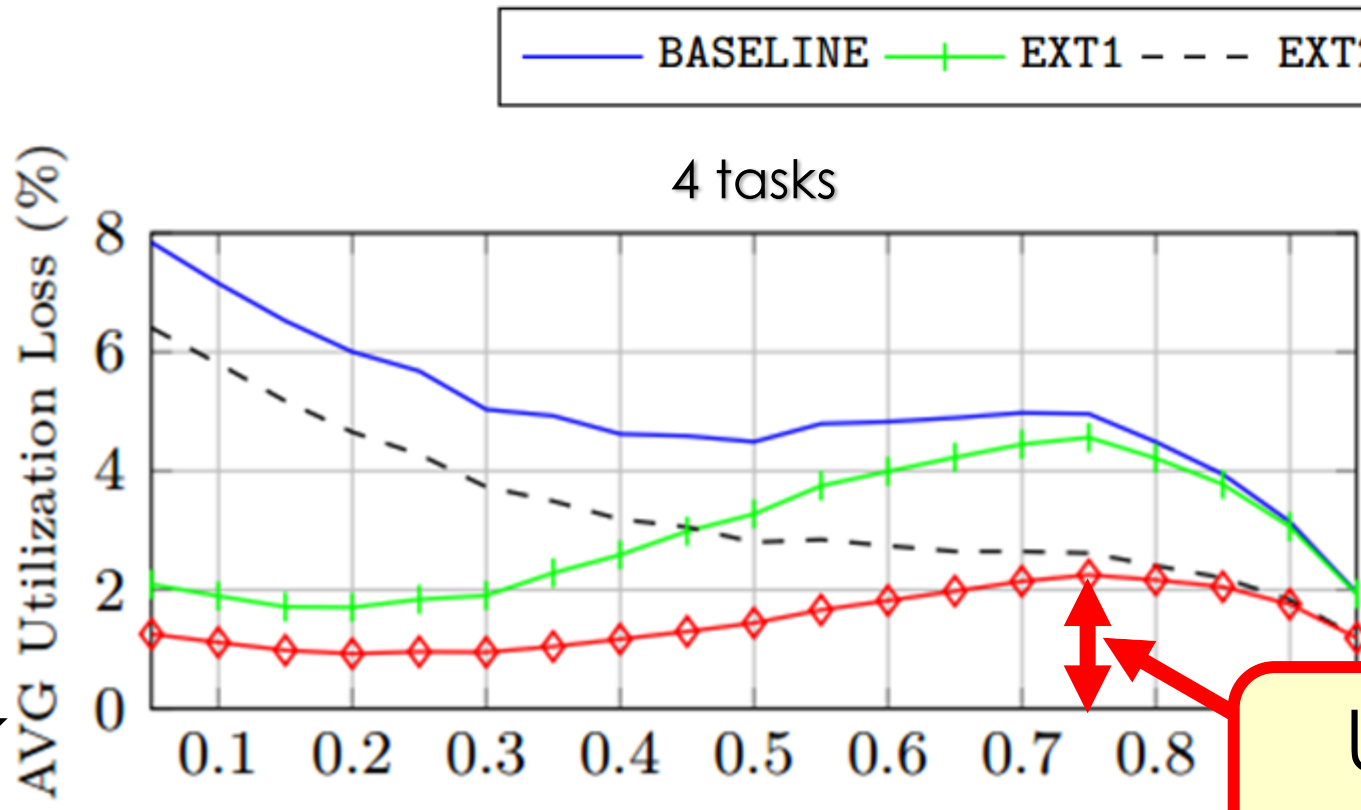
Experimental Study

Measure the utilization loss introduced by our approach with respect to the (exact) Burns et al.'s algorithm



Tested almost 2 Million of task sets over wide range of parameters

Representative Results



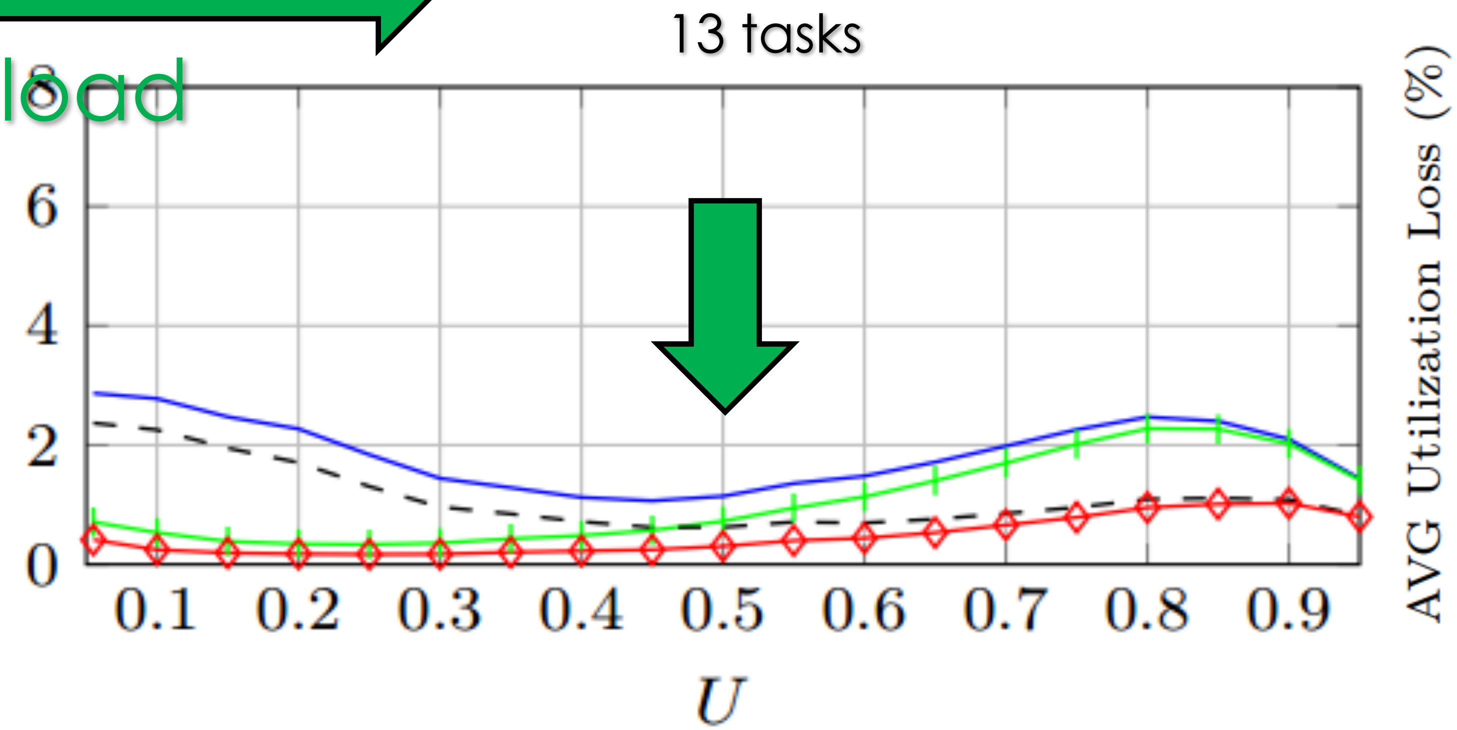
Extension 1 is effective for low utilization values

Extension 2 is effective for high utilization values

Utilization loss **~2%** w.r.t. the exact algorithm

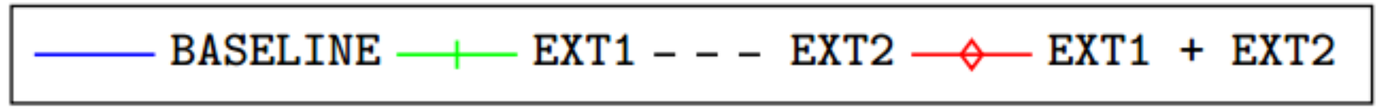
The lower the better

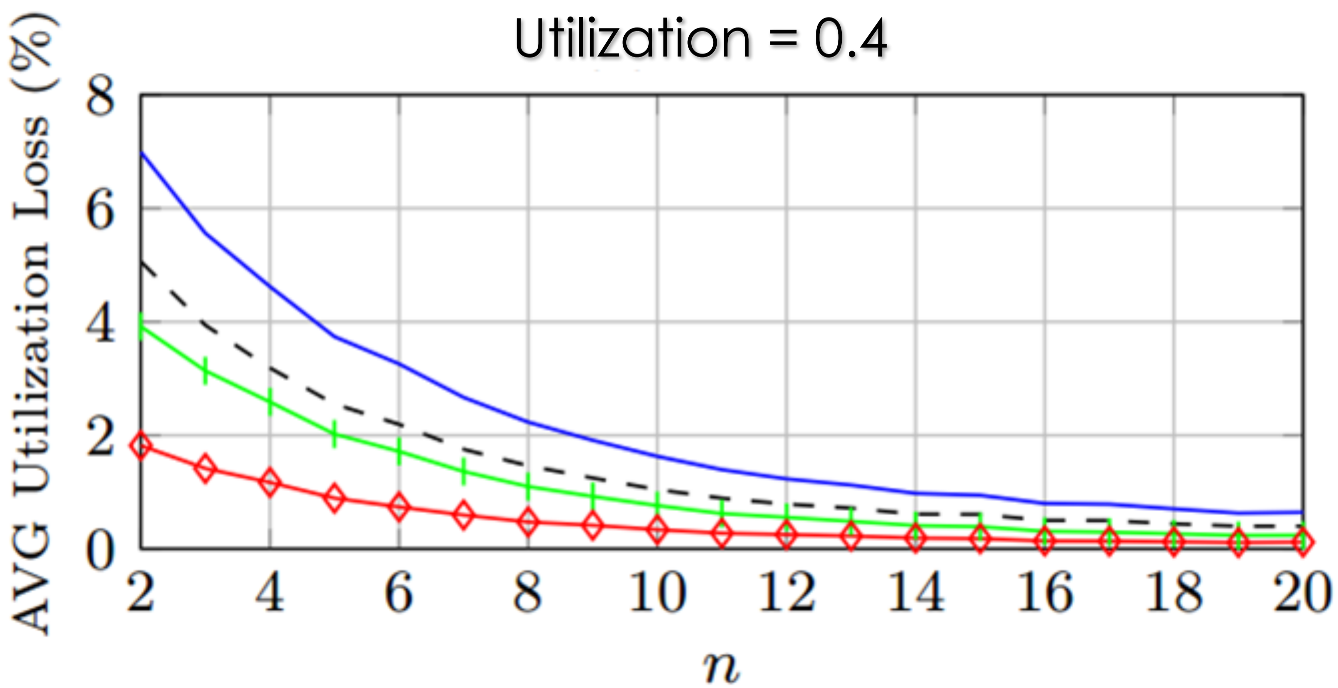
Increasing CPU load



The average utilization loss decreases as the number of tasks increases

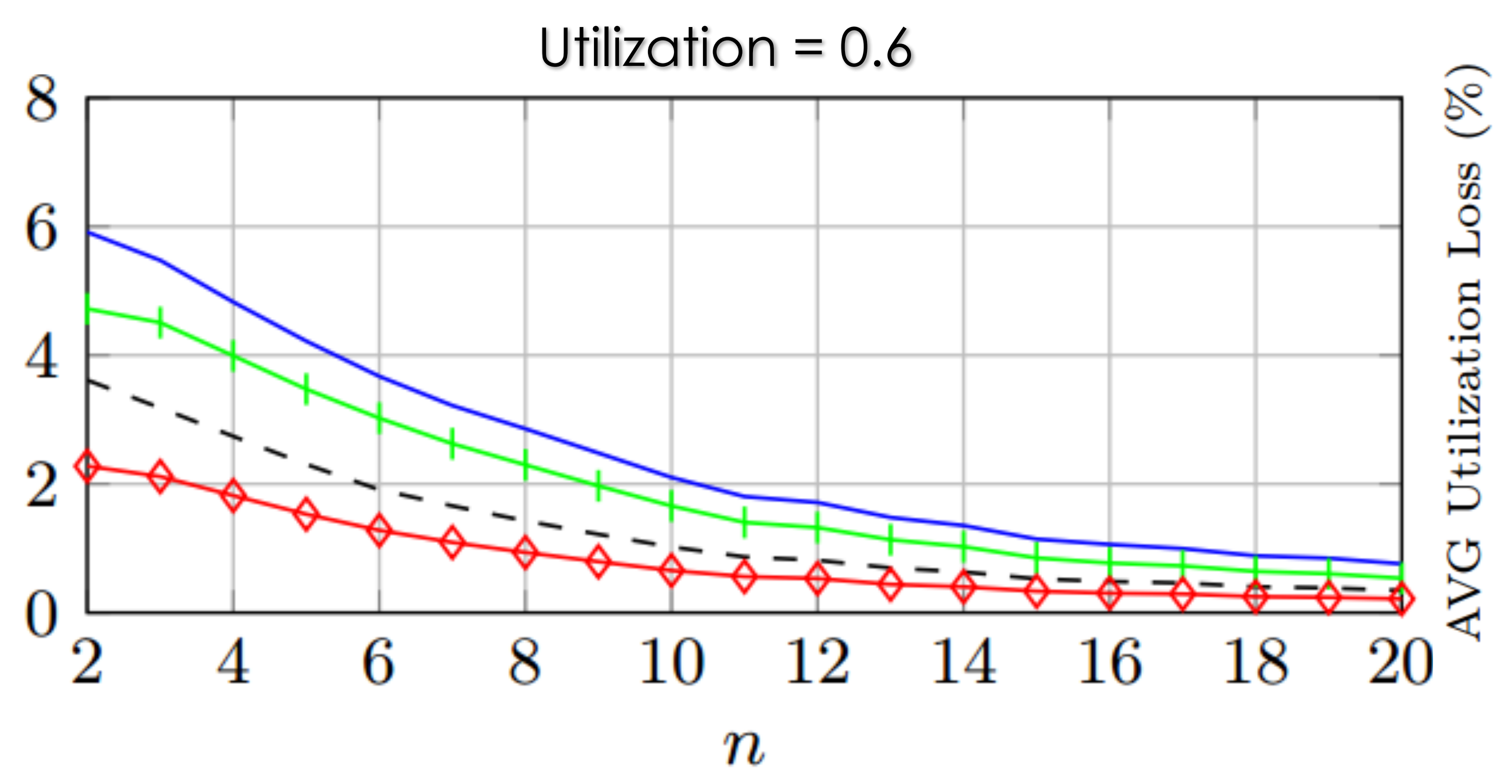
Representative Results





Utilization loss of the baseline approach reaches **very low** values for n > 12

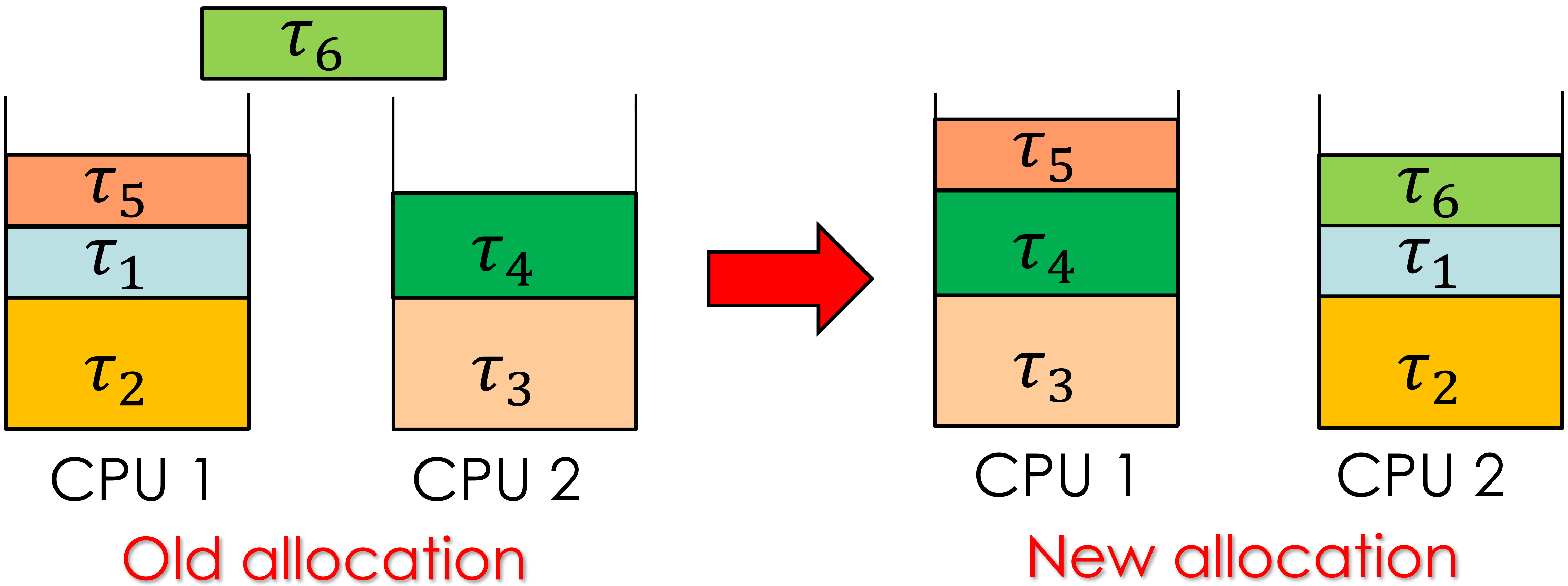
Same trend observed for all utilization values



HOW TO APPLY ON-LINE SEMI-PARTITIONING TO PERFORM LOAD BALACING?

Why do not use classical approaches?

- Existing task-placement algorithms for semi-partitioning would require **reallocating** many tasks (they were conceived for static workload)



Impracticable to be performed on-line:
the previous allocation cannot be ignored!

The problem

How to achieve high schedulability performance with

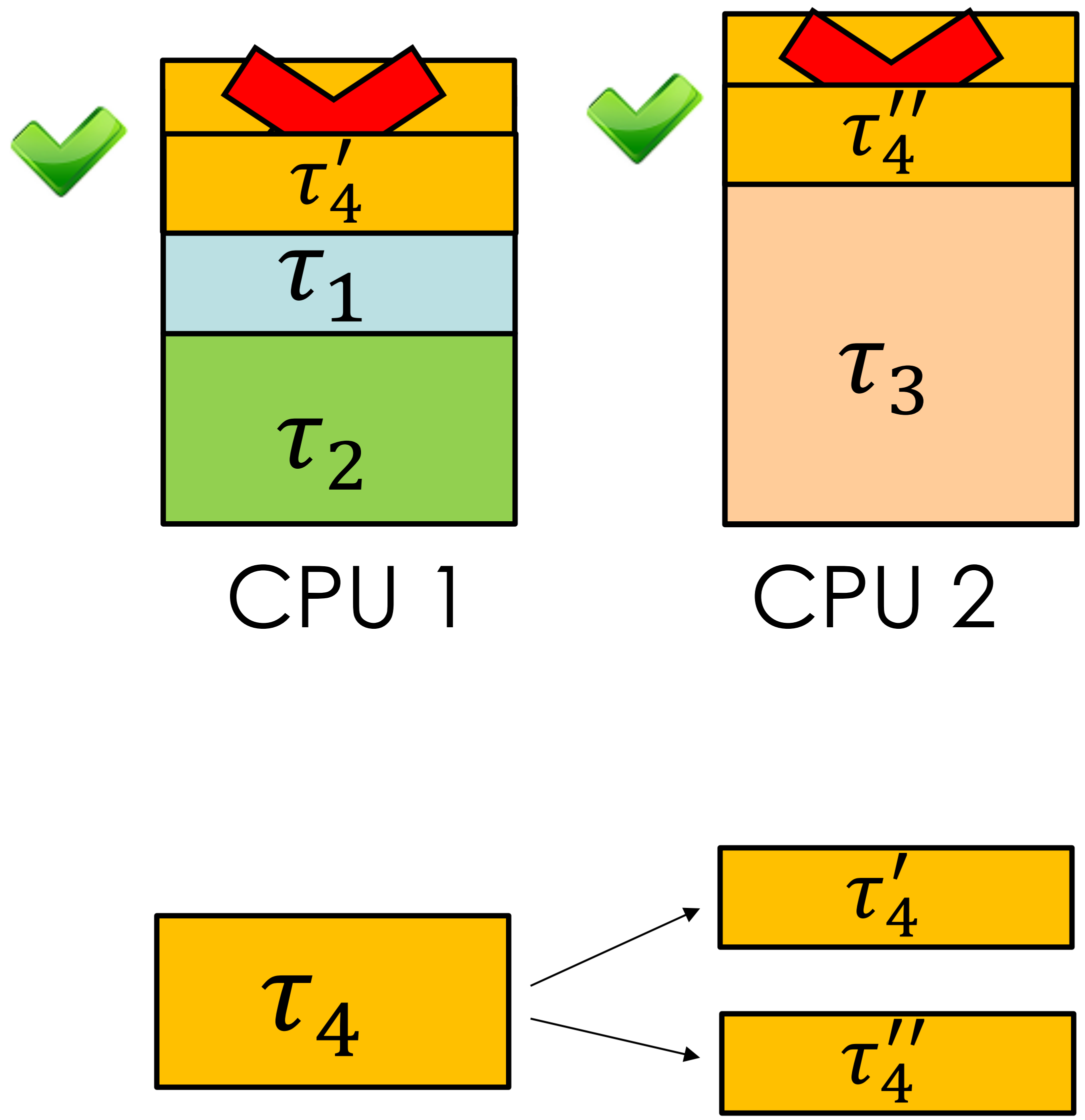
- a very limited number of re-allocations; and
- keeping the mechanism as simple as possible?

Focus on **practical applicability**

Proposed approach

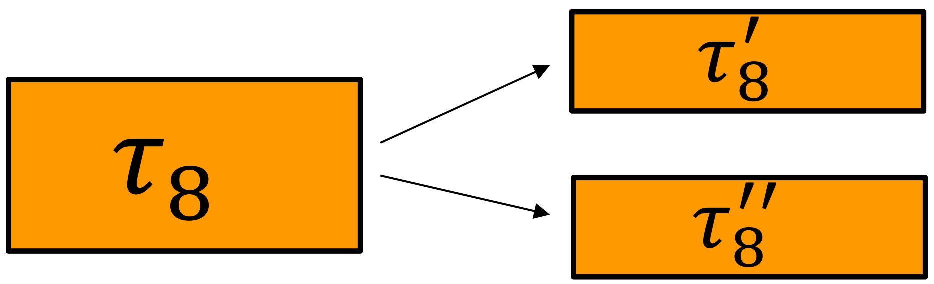
First try a simple bin packing heuristic (e.g., first-fit)

If **not** schedulable, **try to split**



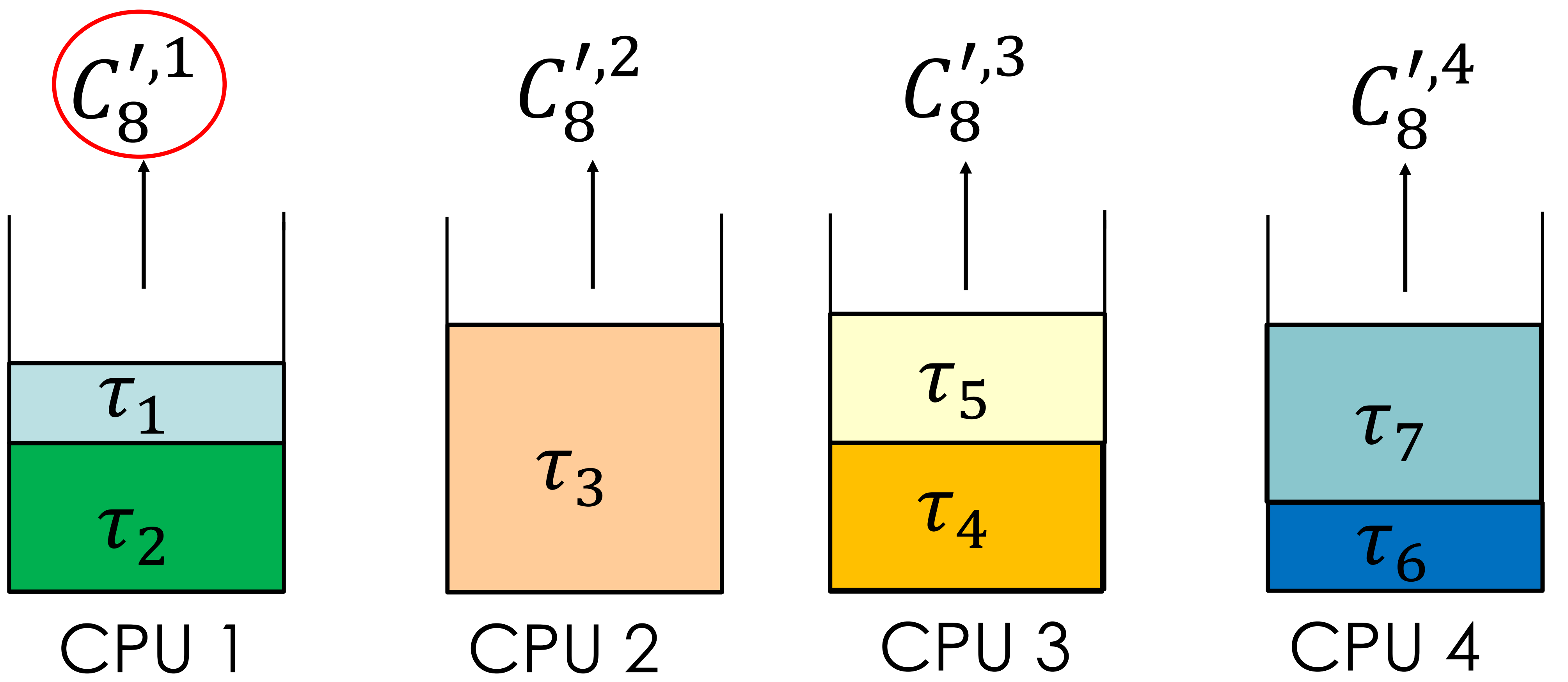
Proposed approach

- How to split?



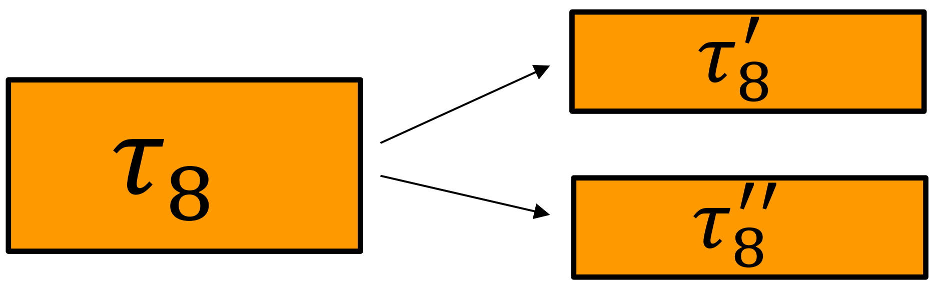
take the maximum zero-laxity budget across the processors

$$\max C_8'$$



Proposed approach

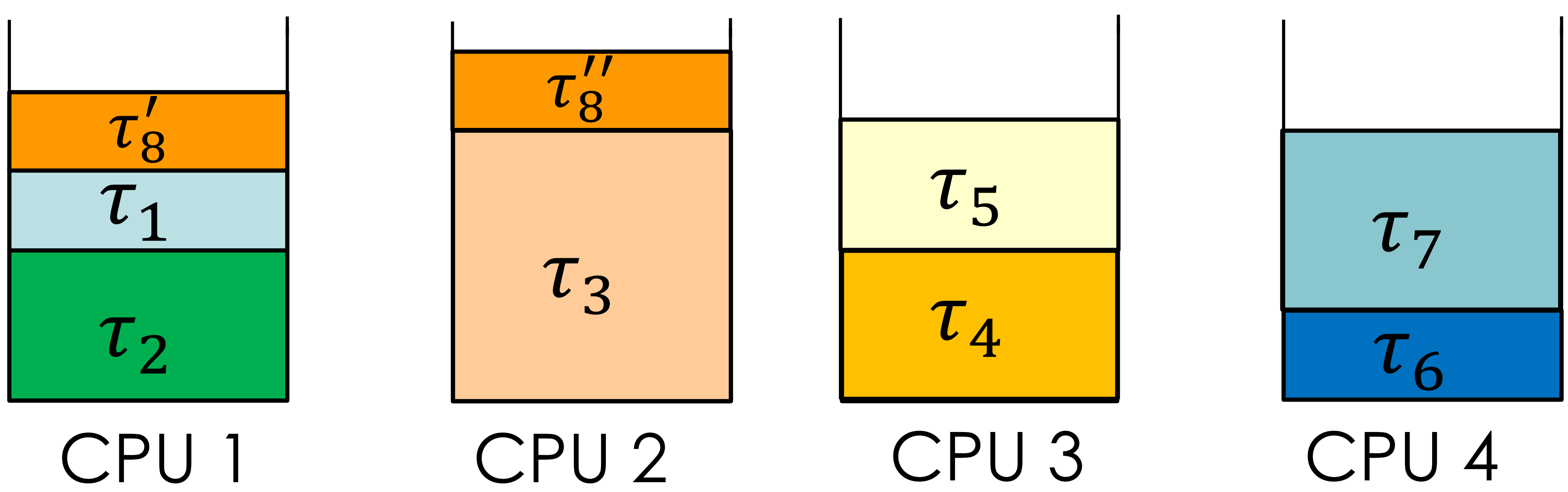
- Admission of a new reservation



1) Allocate the zero-laxity part according to the previous rule

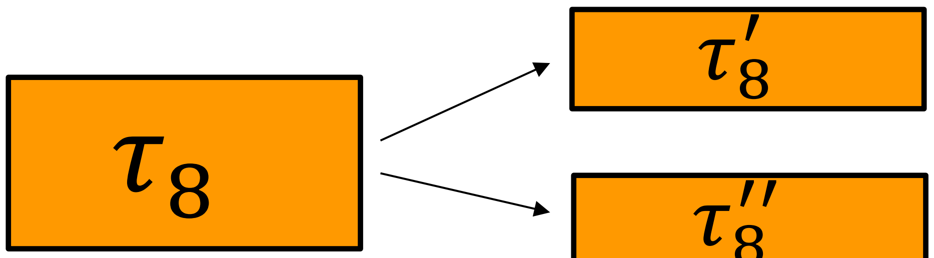
2) Allocate the remaining part using a bin-packing heuristics

$O(m * n^{MAX})$



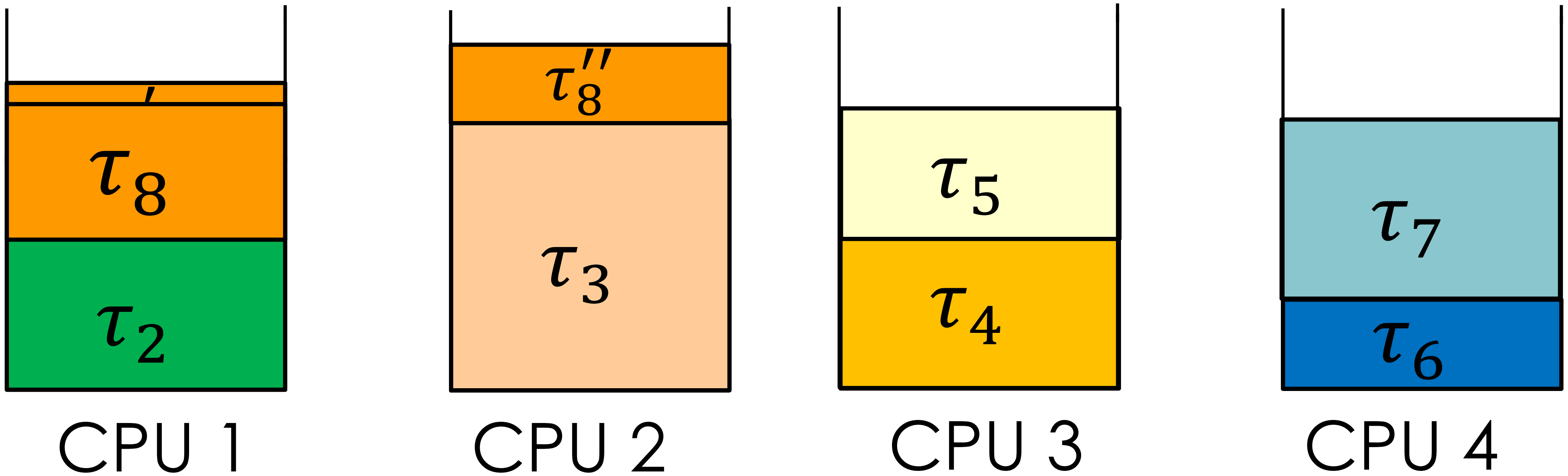
Proposed approach

- Exit of a reservation



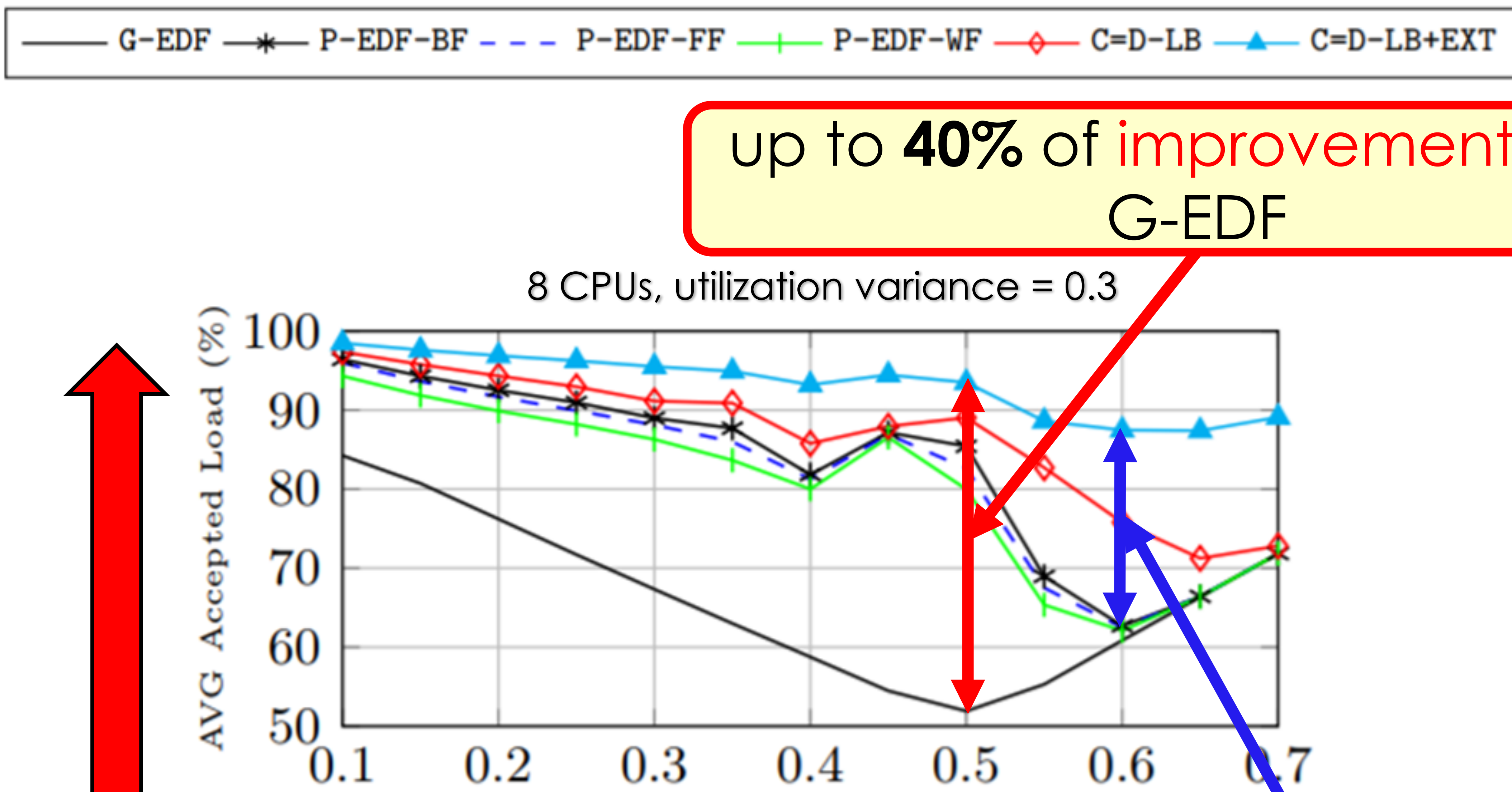
Try to recompact split reservations to favor the admission of future workload

$O(n^{MAX})$



Recall: a property of C=D Scheduling is that there can be at most m split tasks

Experiments



up to **40%** of improvement over G-EDF

up to **25%** of improvement over P-EDF

The higher the better

Increasing average task utilization

Conclusions

- We proposed a linear-time method for computing an approximation of the C=D splitting algorithm
- The approximation algorithm has been used to develop load-balacing mechanisms
- Two large-scale experimental studies have been conducted:
  - The splitting algorithm showed an average utilization loss < 3%
  - The Load Balancing mechanisms allow keeping the system load >87% with improvements up to 40% over G-EDF and up to 25% to P-EDF

Future Work

- Finding better heuristics for load balancing
- Ad-hoc mechanism for handling scheduling transients
- Support for elastic reservation to favor the admission of new workload
- Synchronization issues
- Implementation in a real-time operating systems (e.g., Linux under SCHED_DEADLINE)

Thank you!

Daniel Casini
daniel.casini@sssup.it