Low-Overhead Non-Preemptive Scheduling of Real-Time Tasks upon Multiprocessor Platforms

An extensive attempt on time-triggered systems

Eghonghon-aye *Eigbe*

Supervised by: Mitra Nasri

TUDelft

July 2020

Background

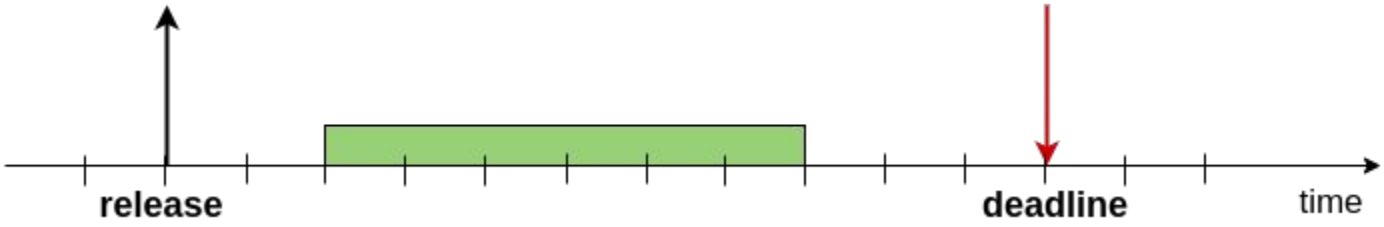
Real-time system

A system whose correctness depends on

Logical correctness of results

AND

Timing correctness of results



Future real-time systems require a lot of computing power

Enter multiprocessor platforms

↓

Increased computing power

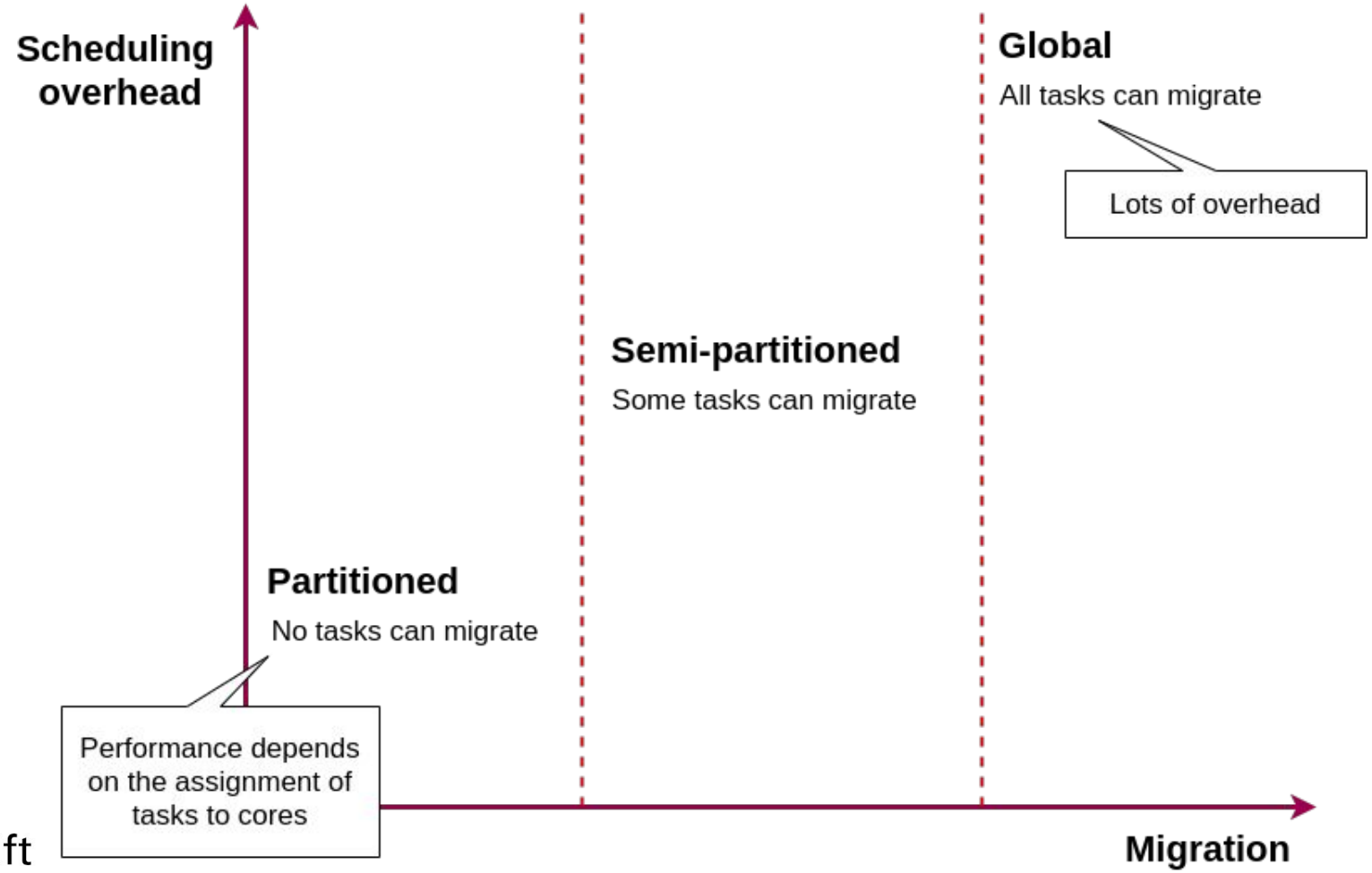
Ability to exploit parallelism

Increased scheduling complexity

TU Delft

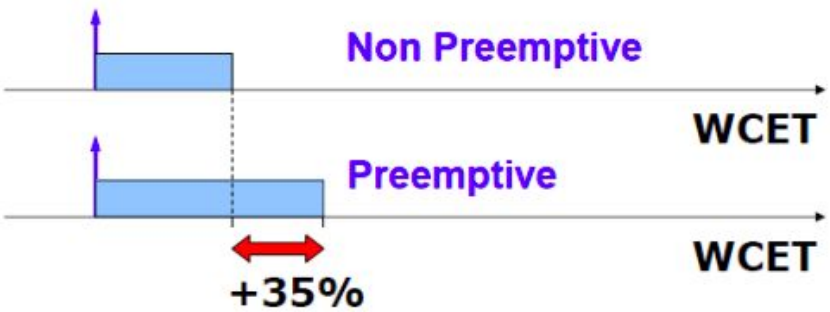
Classification of multiprocessor schedulers

Periodic **tasks** are system functions that periodically release instances (**jobs**) all through the lifetime of the system



Why non-preemptive scheduling?

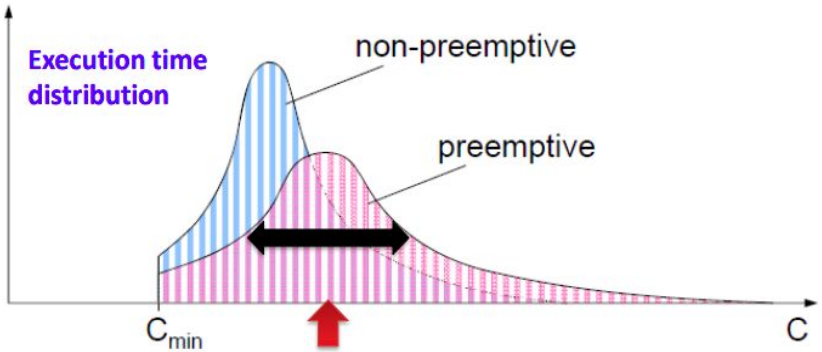
No added context switch cost



Less cache-related preemption delay (CRPD)

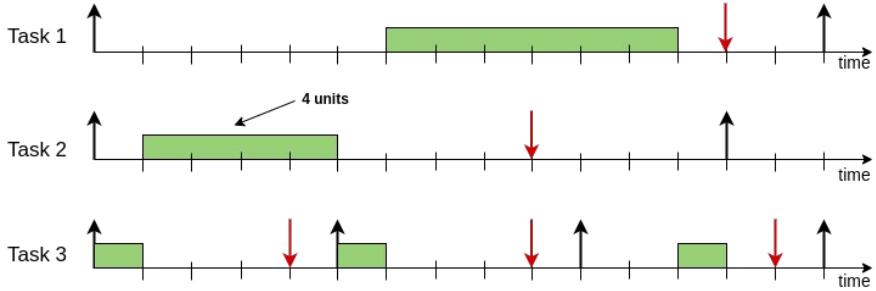
Fewer cache evictions

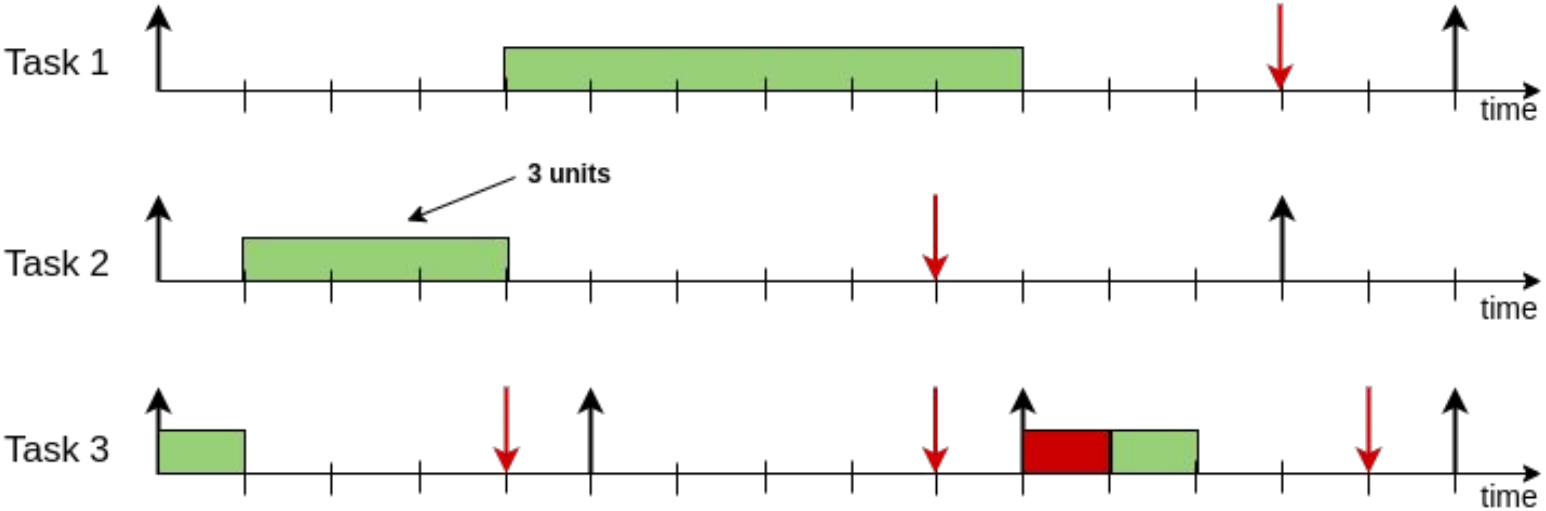
Less variation in WCET



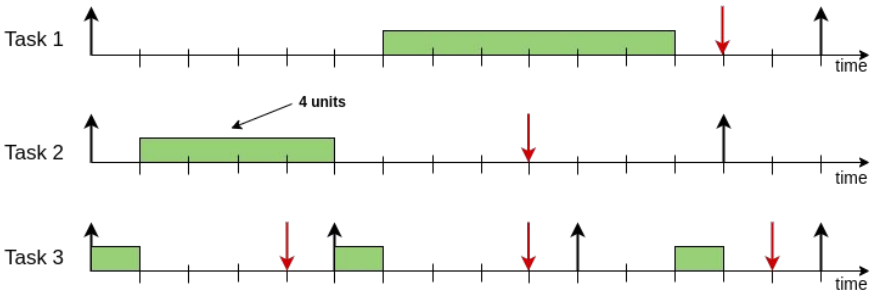
[1]T. P. Baker, “Stack-based Scheduling for Real-time Processes,” Real-Time Systems, vol. 3, no. 1, pp. 67–99, 1991.
[2] C.-G. Lee, J. Hahn, Y.-M. Seo, S. Min, R. Ha, S. Hong, C. Park, M. Lee, and C. Kim, “Analysis of cache-related preemption delay in fixed-priority preemptive scheduling,” IEEE Transactions on Computers, vol. 47, no. 6, pp. 700– 713, 1998

Non-preemptive scheduling anomalies

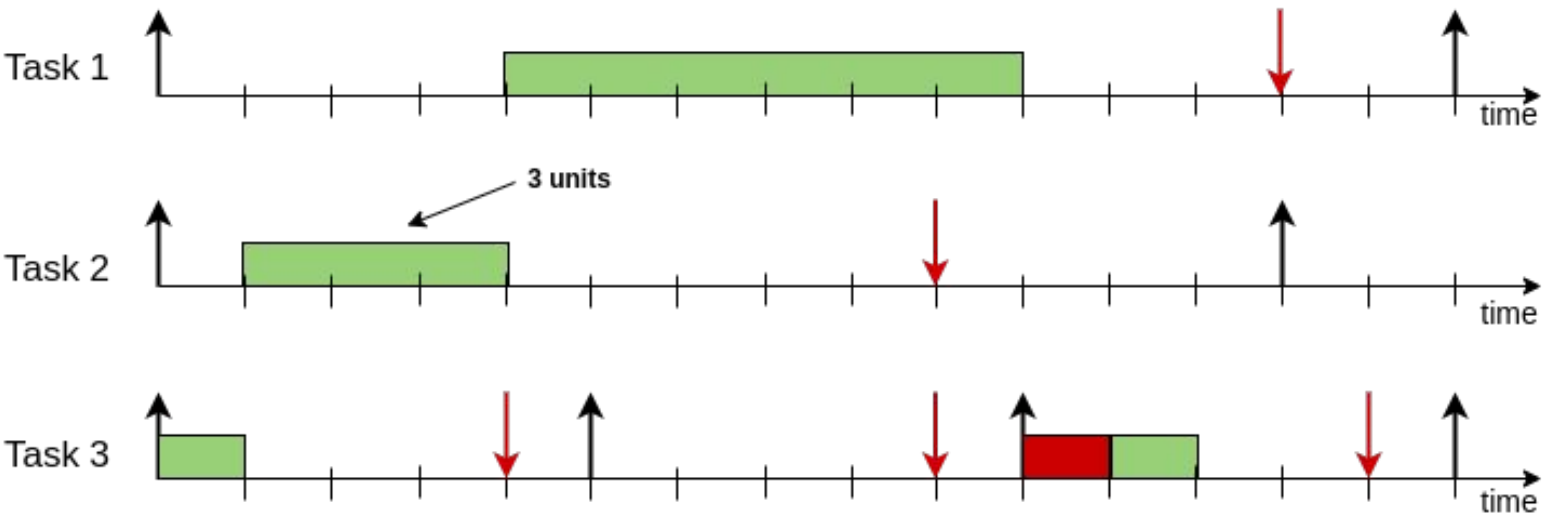




Non-preemptive scheduling anomalies



Executing for less than WCET can lead to deadline misses



Sustainable scheduling as a cure for some anomalies

Padding computation to ensure WCET always

Using a sustainable online policy

Manipulate offsets and release times to recreate any offline schedule

↓

FIFO with Offset Tuning

TUDelft

Nasri, Mitra, Robert I. Davis, and Björn B. Brandenburg. "FIFO with offsets: High schedulability with low overheads." IEEE, 2018.

Thesis Roadmap

Thesis Roadmap

What is the true limit of partitioning?

A necessary test for partition-ability

What if partitioning fails?

A semi-partitioning scheme

Can this improve parallel task scheduling?

Decompose the parallel task into sequential parts

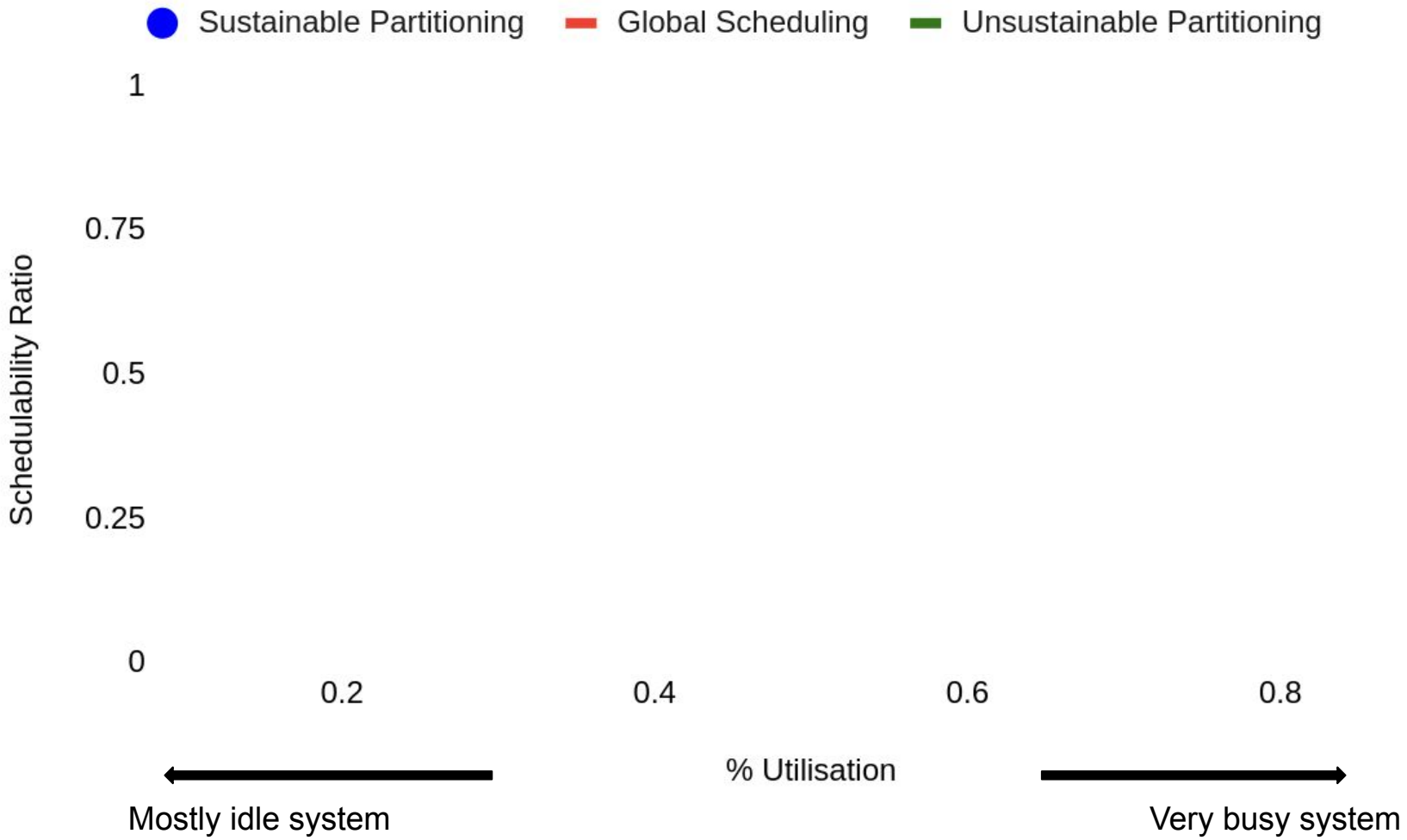
How efficient are our algorithms at runtime?

Extend FIFO-OT to multiprocessor platforms

TU Delft

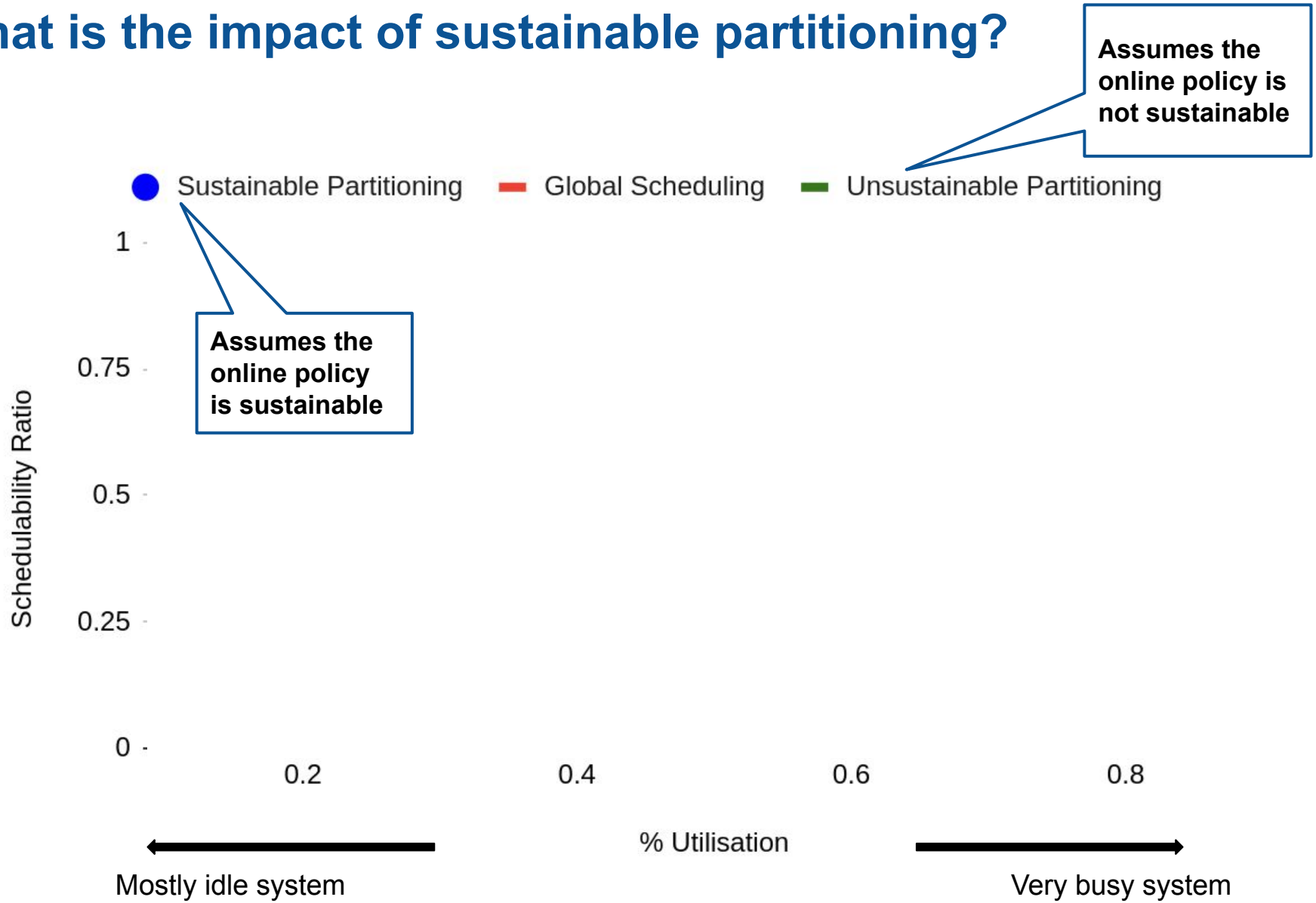
What is the true limit of partitioning?

What is the impact of sustainable partitioning?



TU Delft

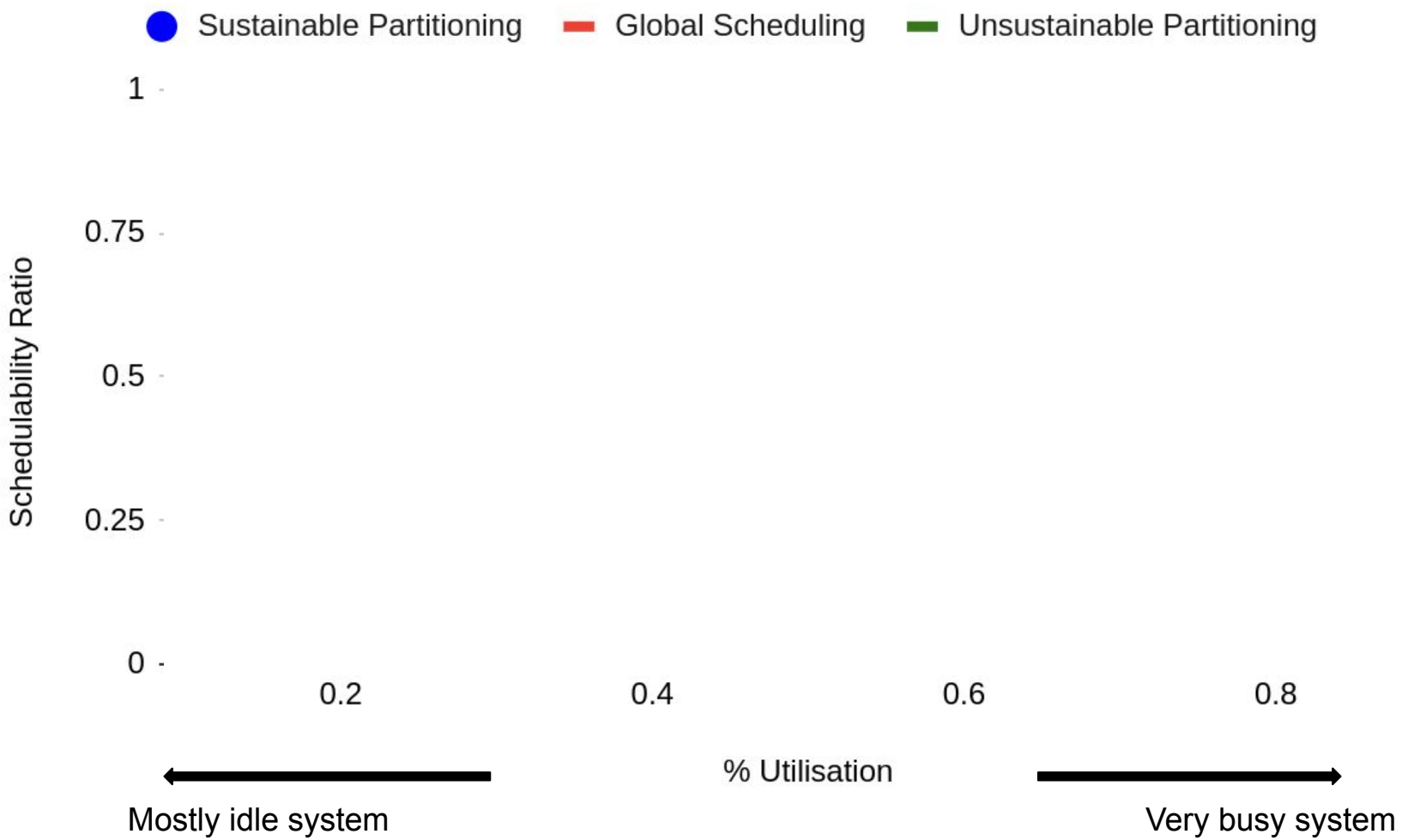
What is the impact of sustainable partitioning?



TUDelft

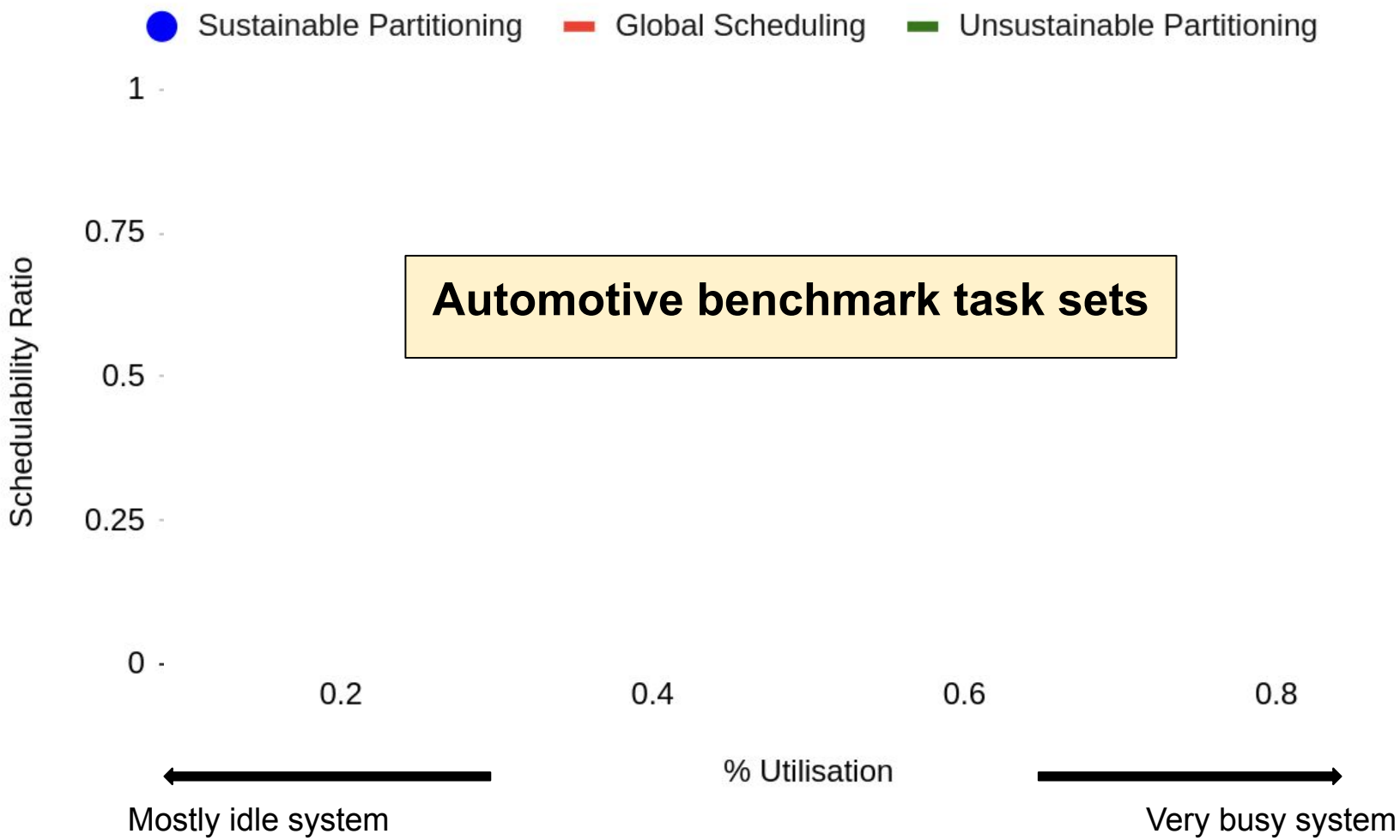
What is the impact of sustainable partitioning?

No non-preemptive semi-partitioning policy (to the best of our knowledge)



TU Delft

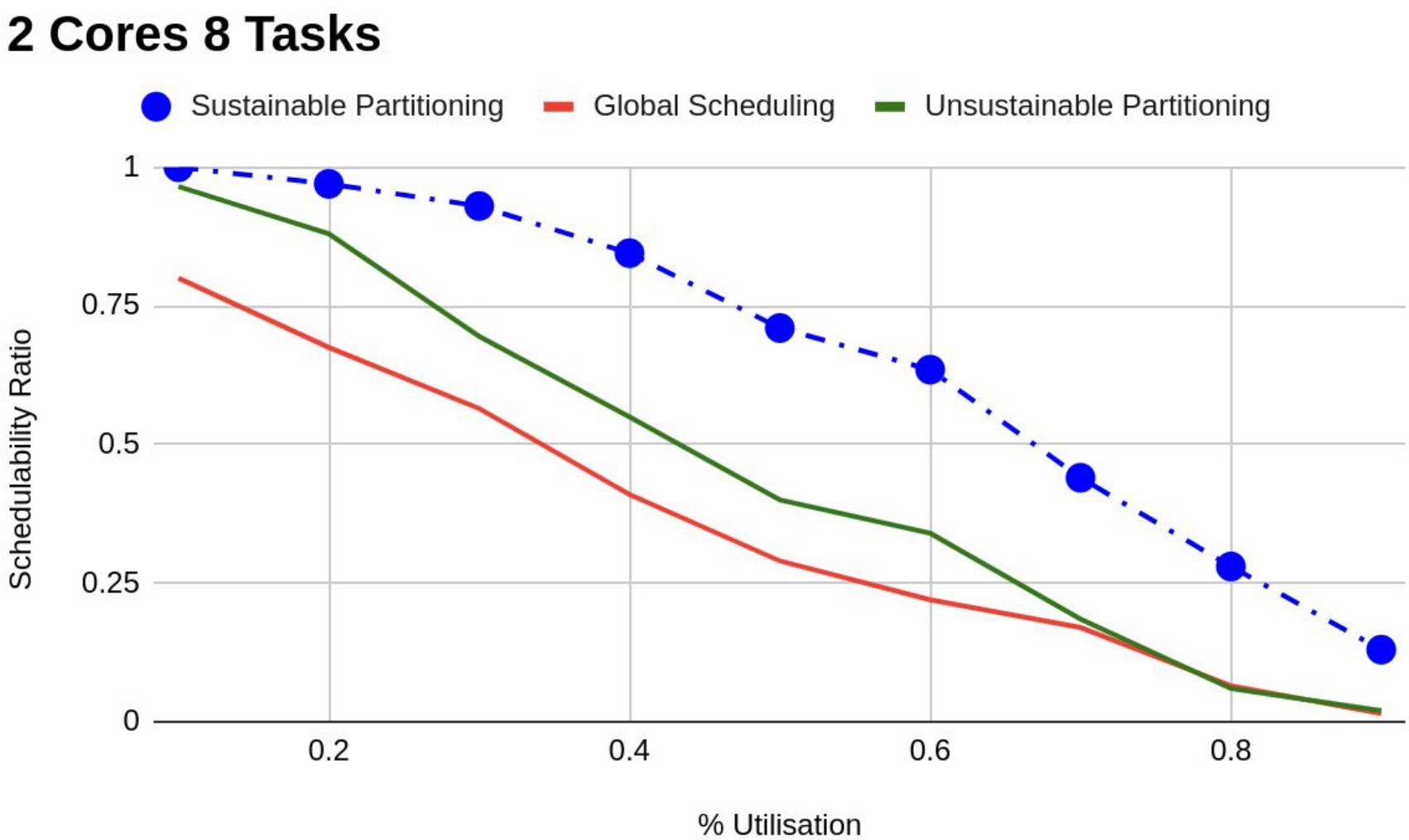
What is the impact of sustainable partitioning?



Automotive benchmark task sets

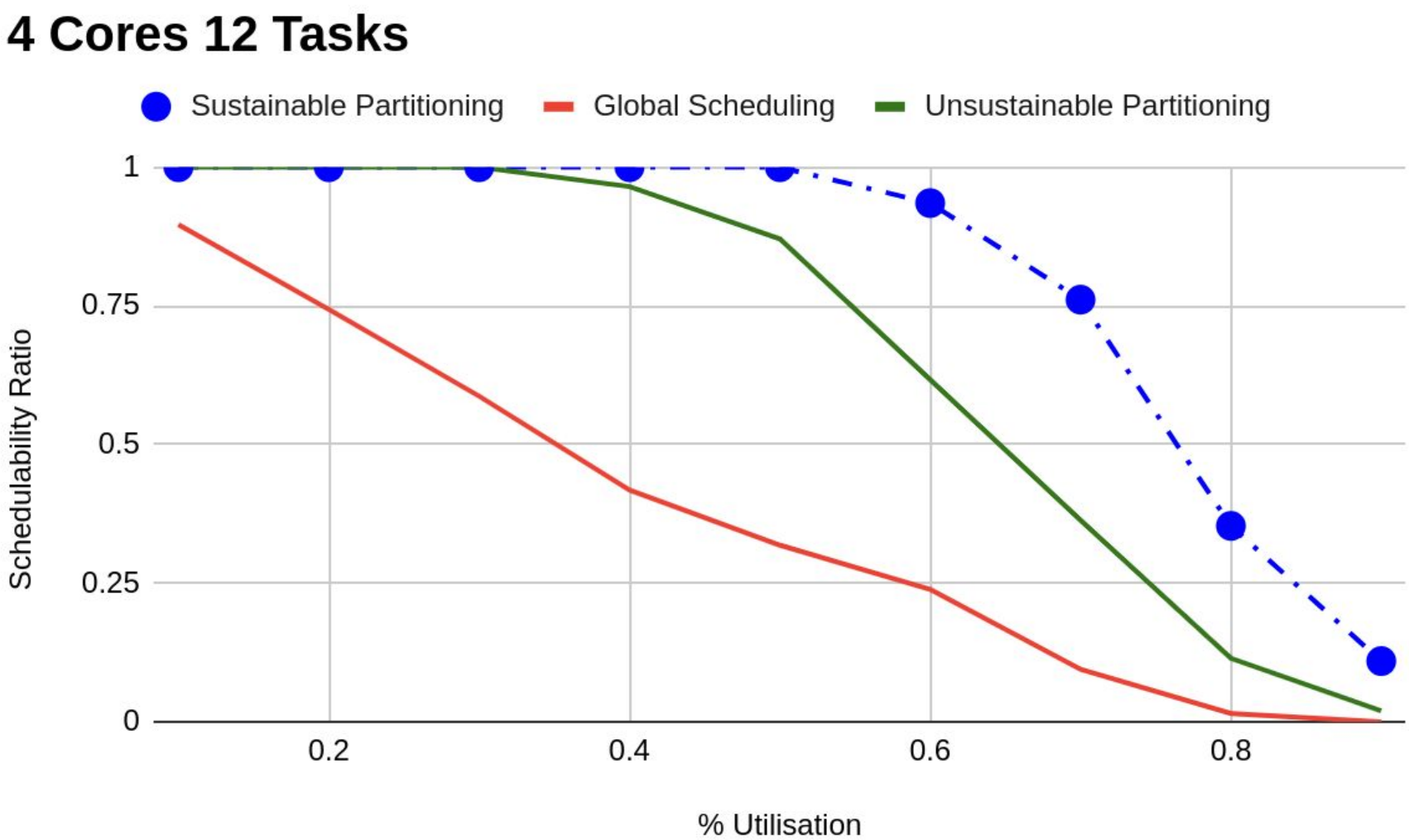
Kramer Simon, Ziegenbein Dirk and Hamann Arne, “Real World Automotive Benchmarks for Free”, WATERS, 2015

What is the impact of sustainable partitioning?



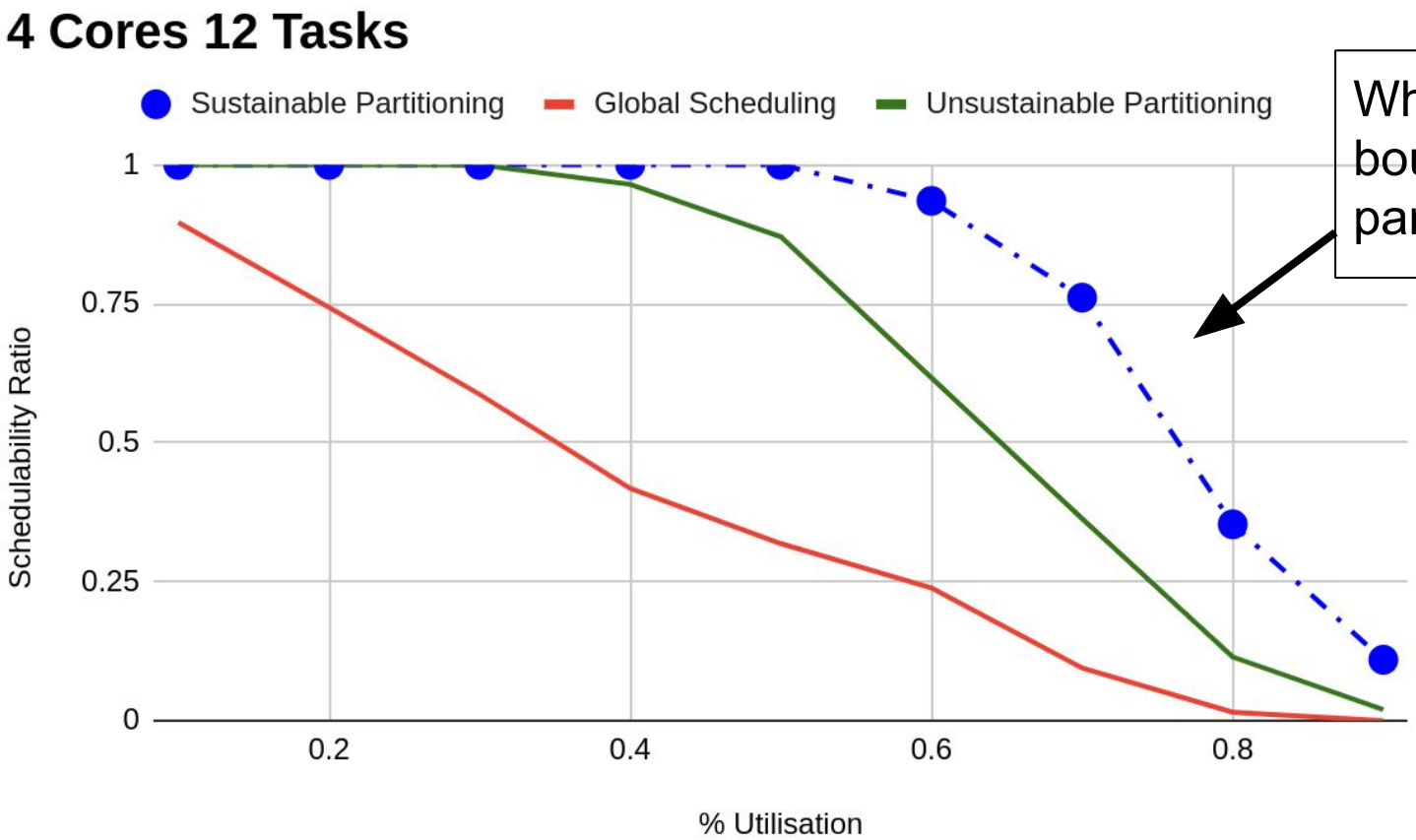
TU Delft

What is the impact of sustainable partitioning?



TU Delft

Why do we need a necessary test for partition-ability?



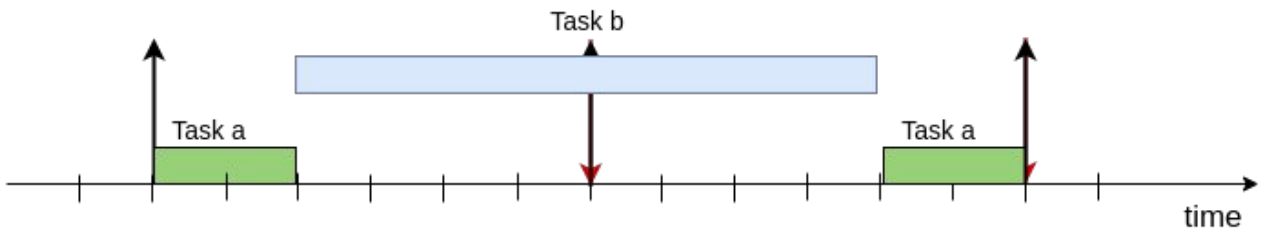
What is the upper bound on partition-ability?

TU Delft

Building a necessary test for partition-ability

- **Recognize conflicts**
  - Conflicts are subsets of tasks that cannot go on the same core

- **Do a pairwise search through the task set**
  - Pairs of tasks must not be able to completely block each other

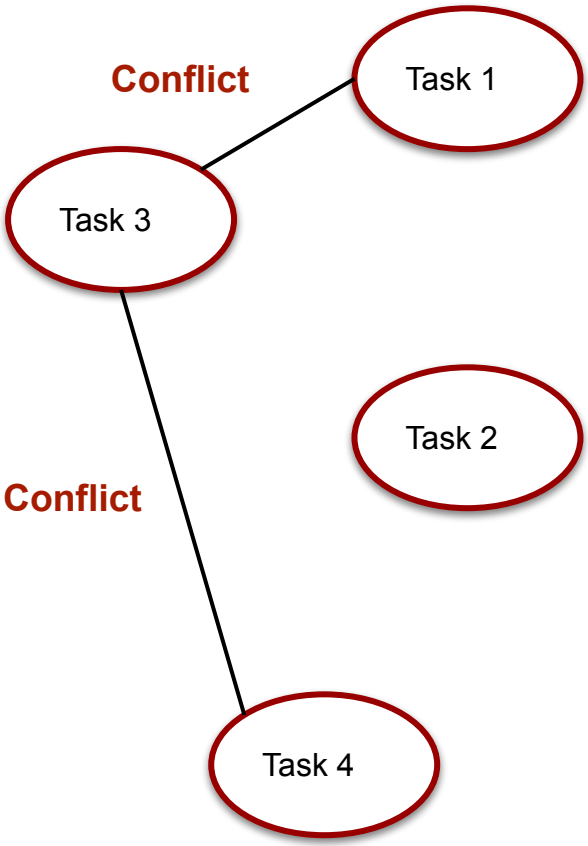


  - If they can, there is a conflict

TU Delft

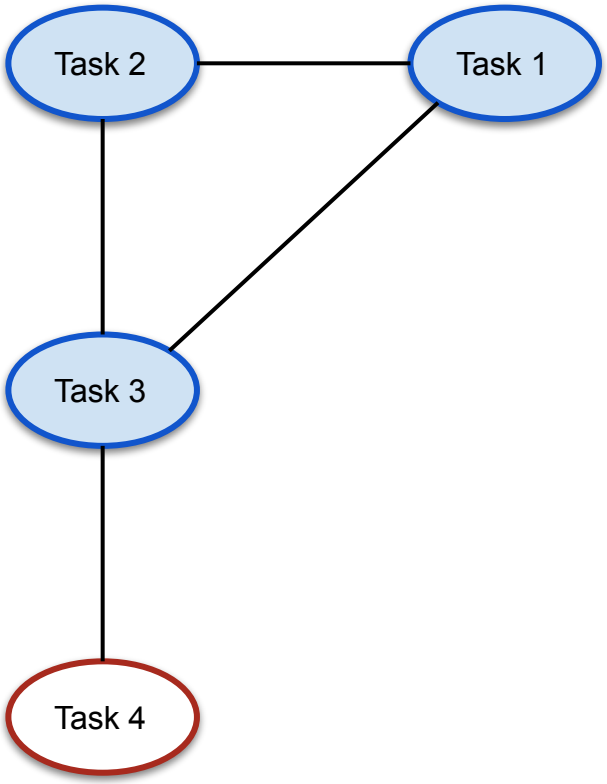
Conflict graph model of a task set

- Each node is a task
- Each edge represents a conflict between two tasks

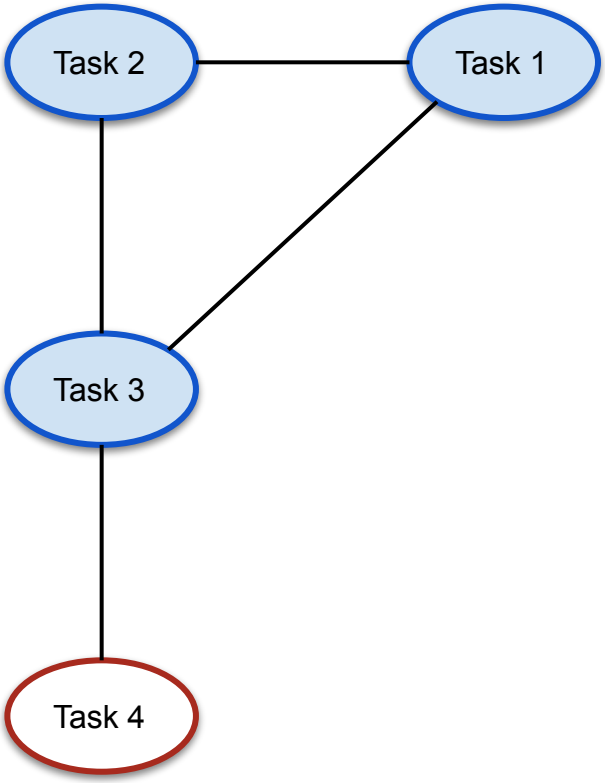


TU Delft

What can cliques tell us?

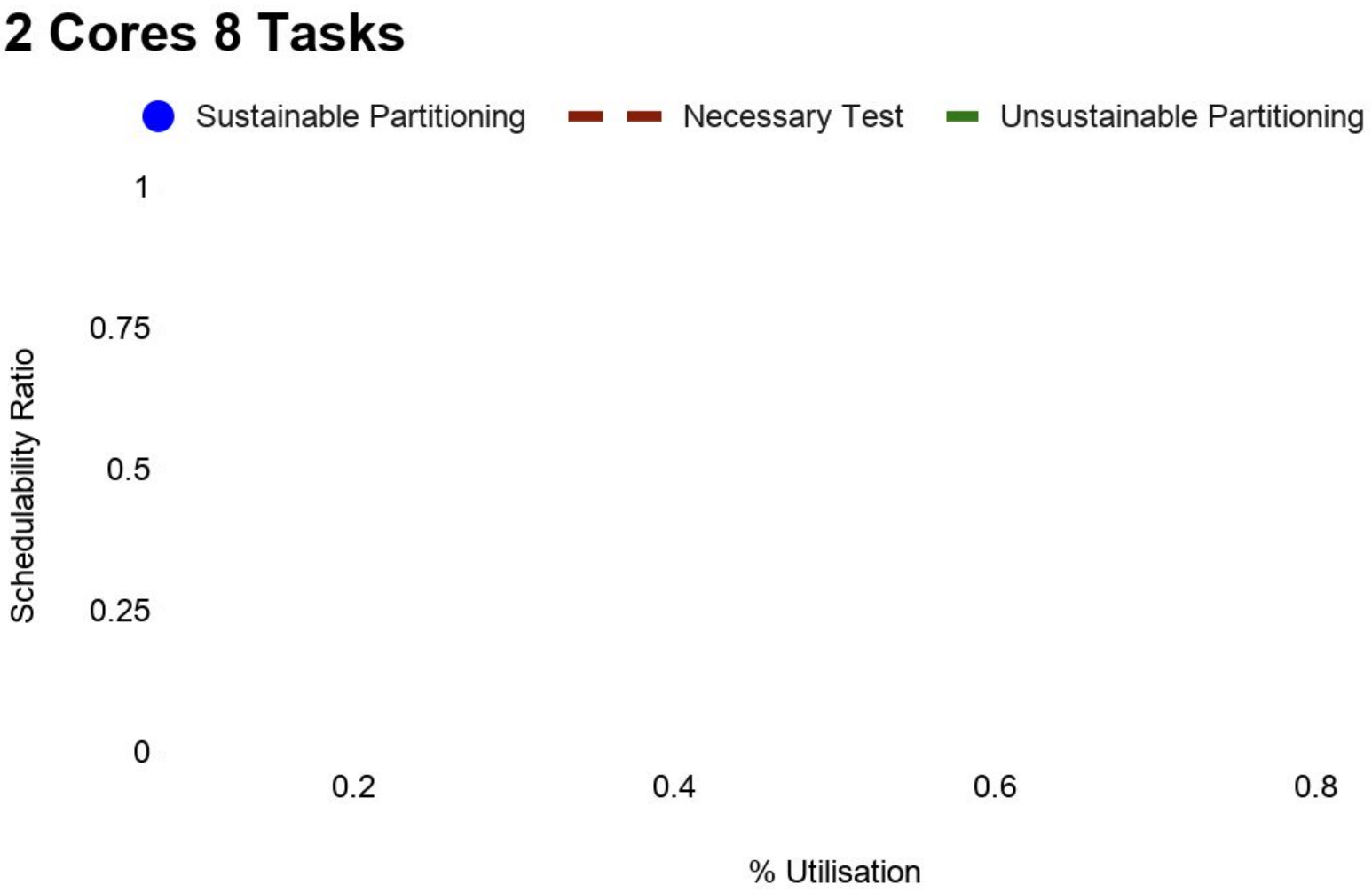


A necessary test



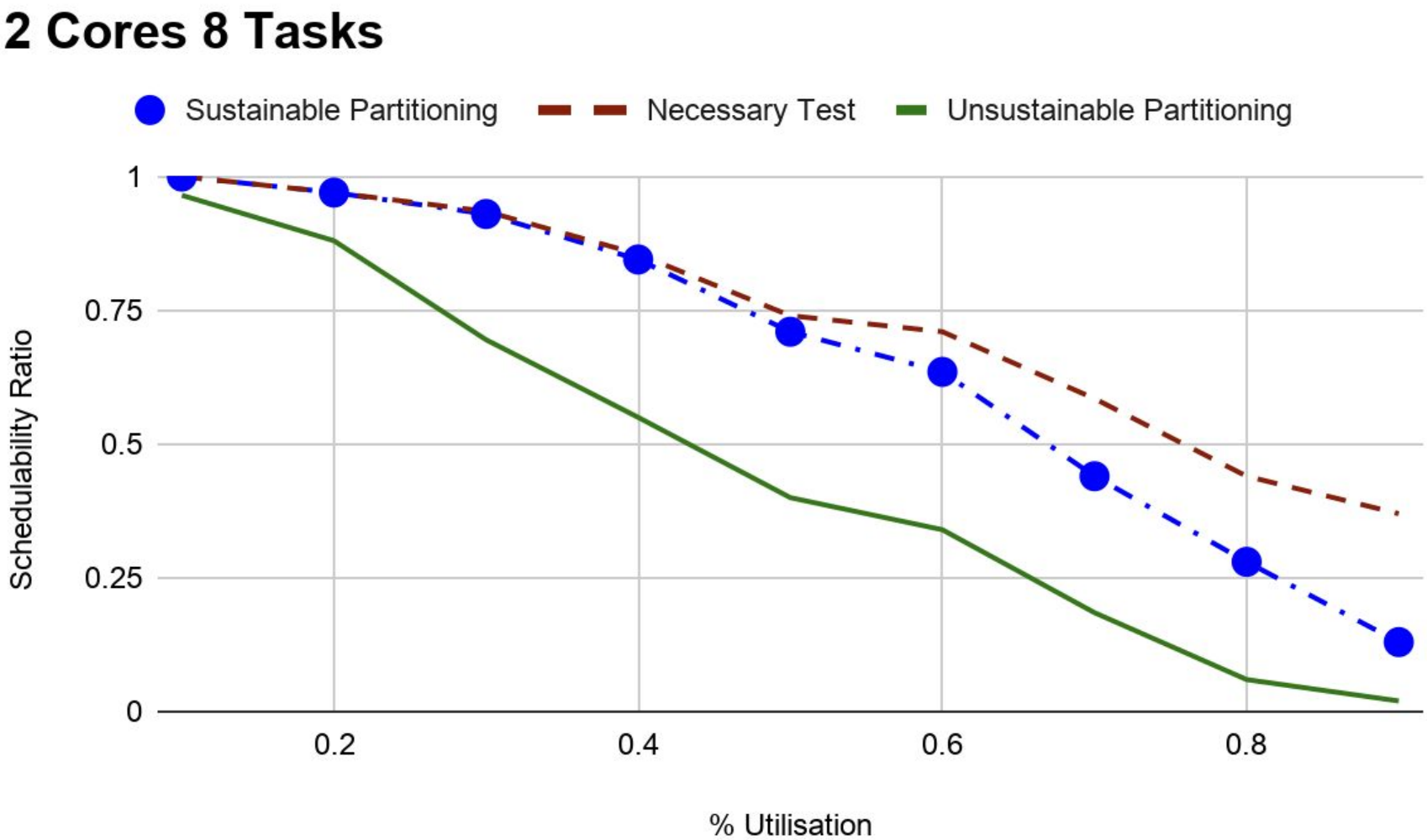
The size of the largest clique is the minimum number of cores required to partition the task set

How do we perform against the upper bound?



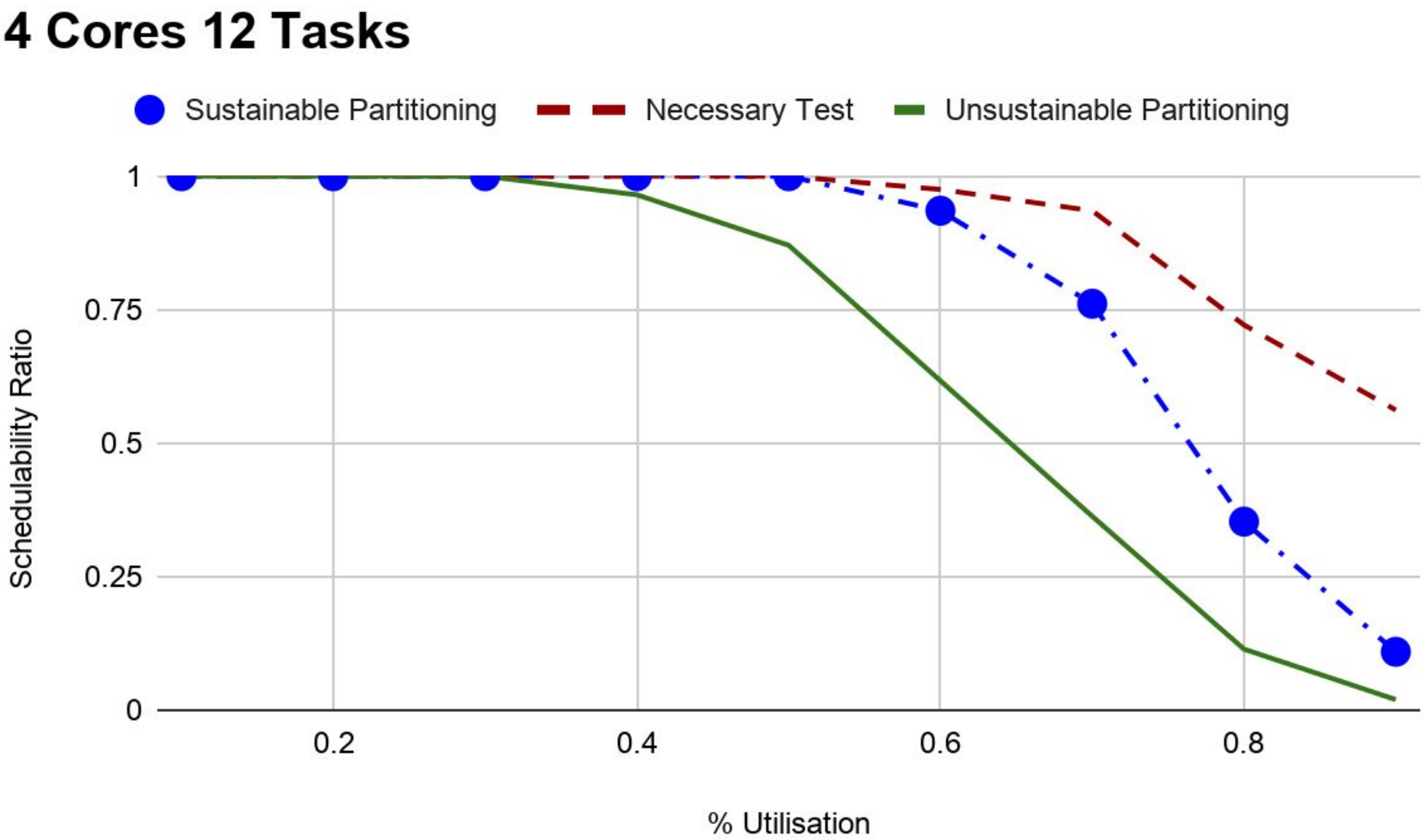
TU Delft

How do we perform against the upper bound?



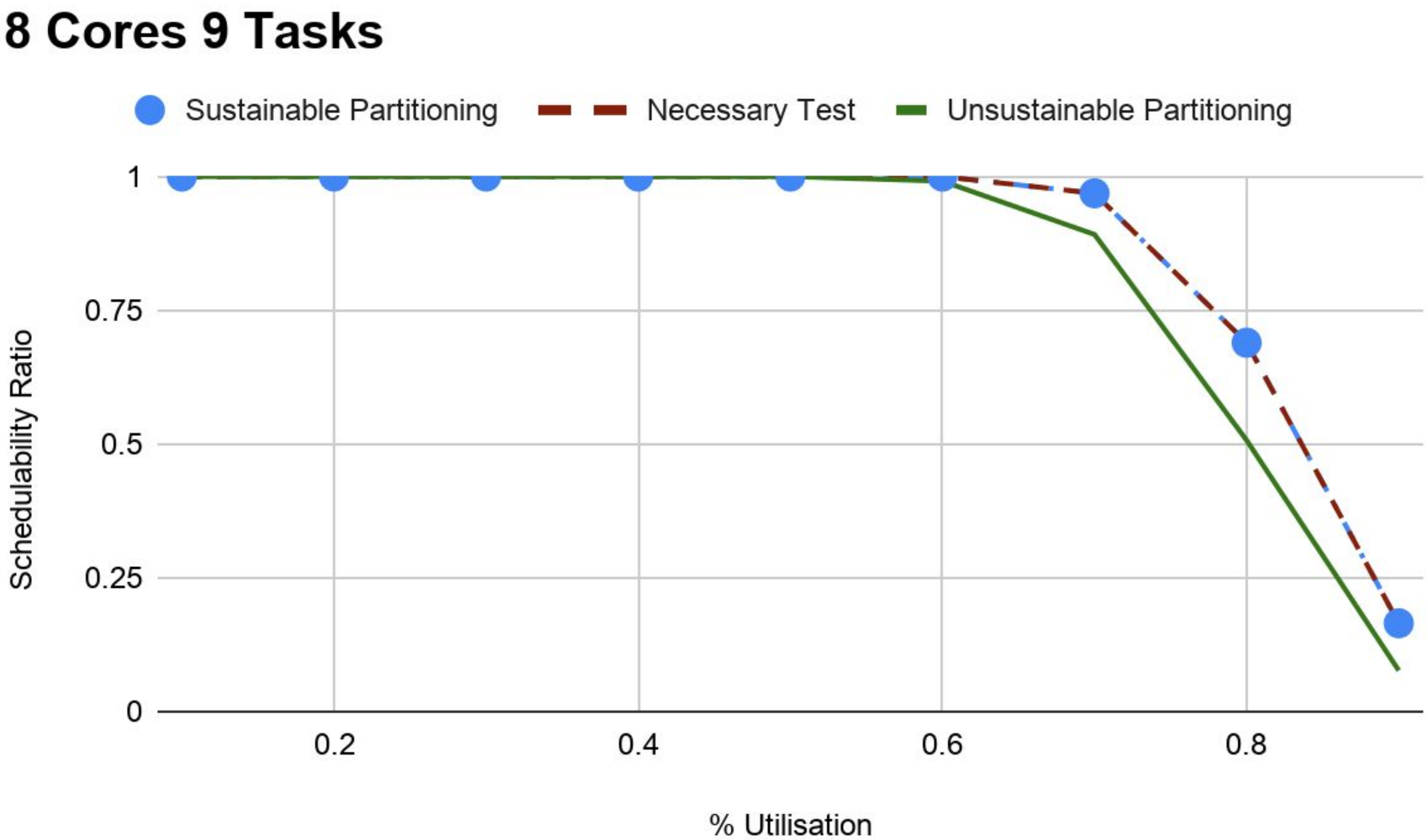
TU Delft

How do we perform against the upper bound?



TU Delft

How do we perform against the upper bound?

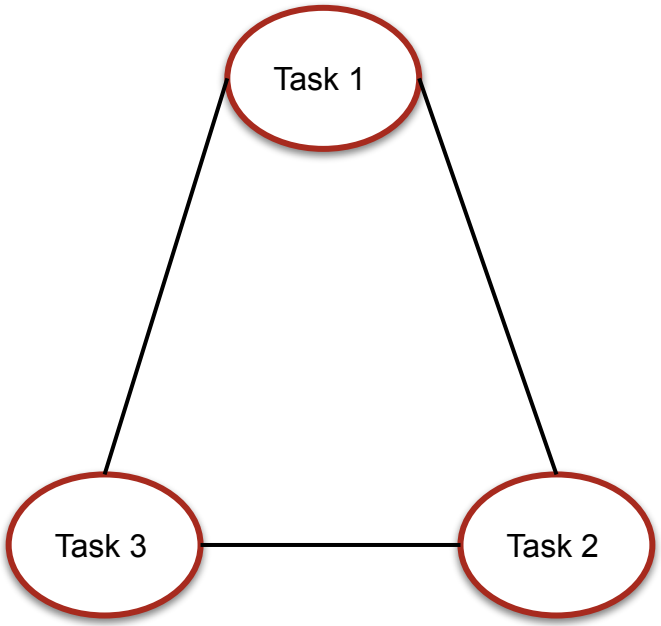


TU Delft

What if partitioning fails?

Case Study

Task Number	Period T	Utilisation U	Execution Time C
1	100	0.65	65
2	100	0.45	45
3	20	0.4	8

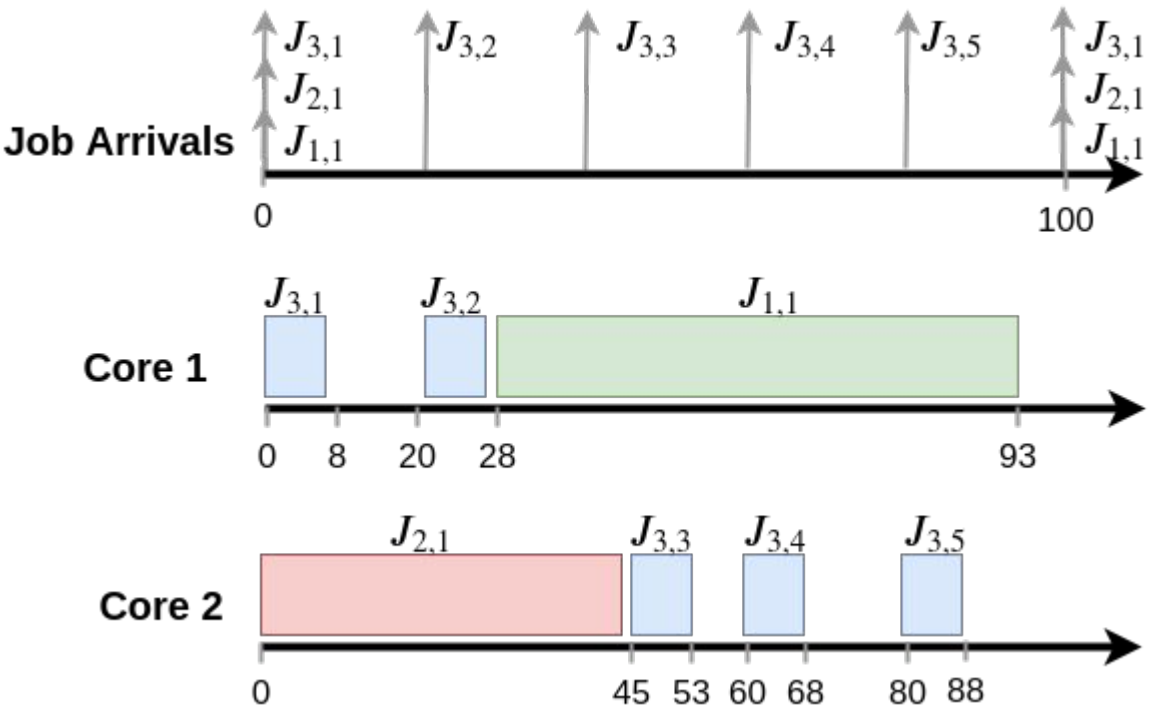


Conflict Graph

Not partition-able on 2 cores

Case Study

Task Number	Period	Utilisation	Execution Time
1	100	0.65	65
2	100	0.45	45
3	20	0.4	8



Schedulable on 2 cores

Our approach to semi-partitioned scheduling

Partitioning heuristics work by progressively fitting **tasks** onto cores

↓

But semi partitioning allows some migration

↓

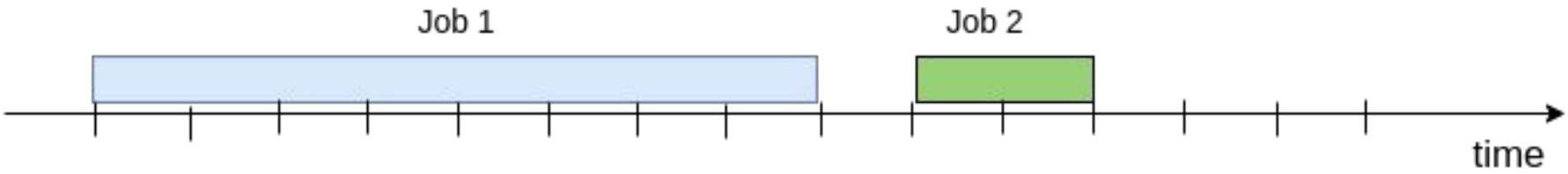
So we progressively fit **jobs**

↓

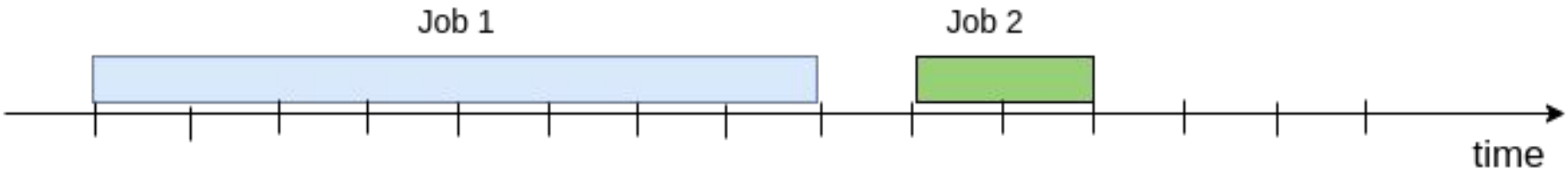
Largest objects first principle

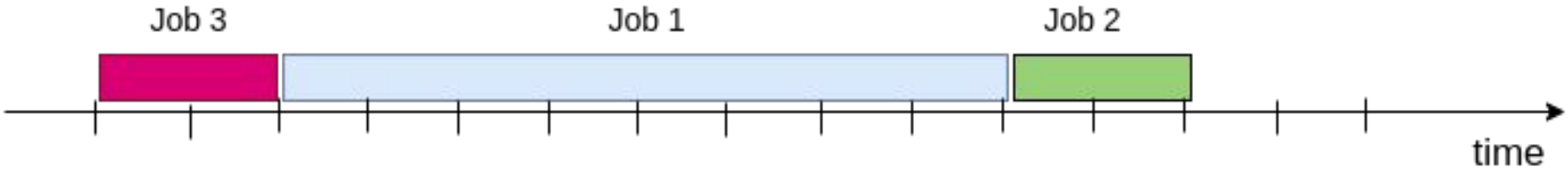
TU Delft

Liquid-path scheduling



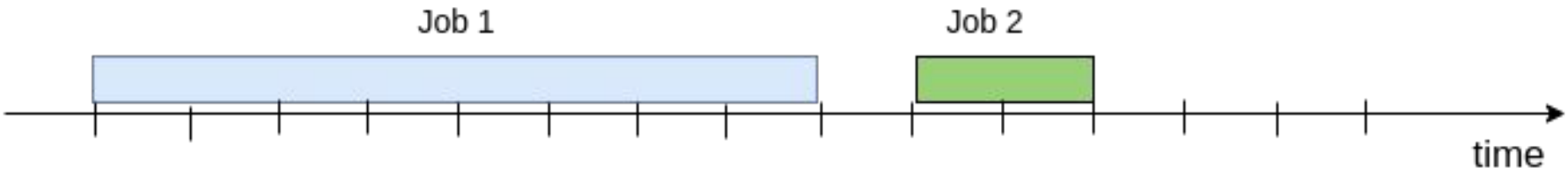
Liquid-path scheduling

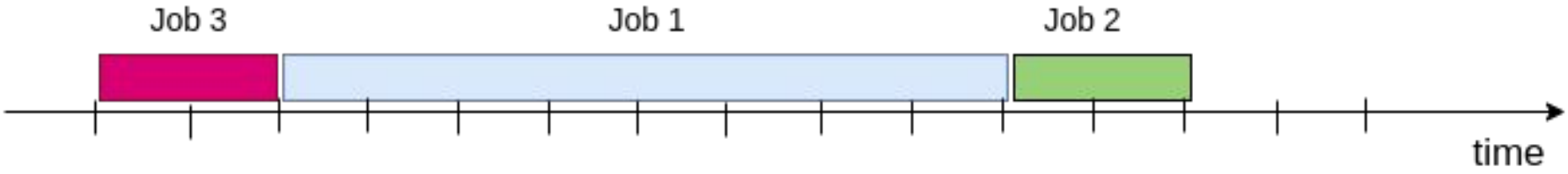




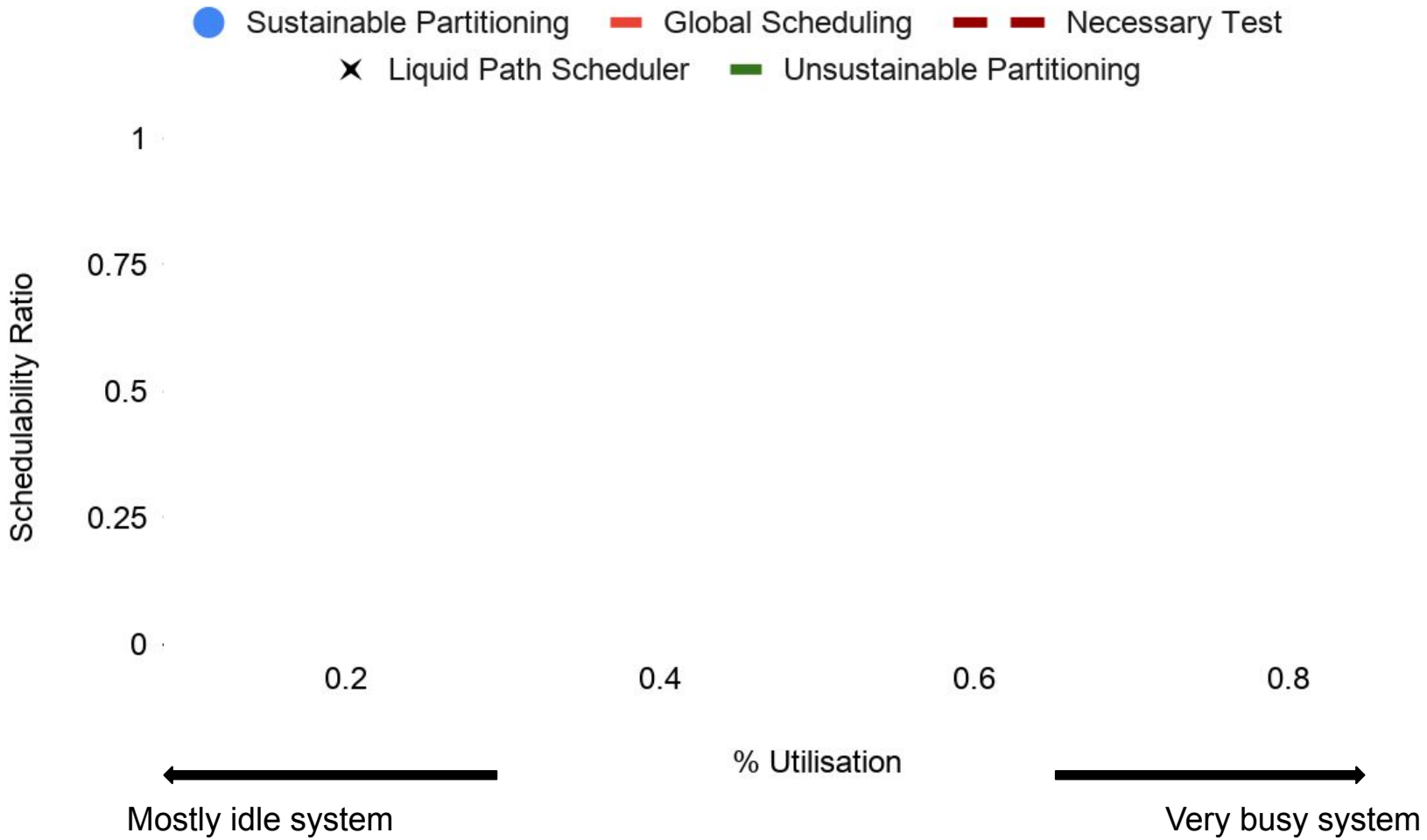
Liquid-path scheduling

Do this till we have placed all jobs for one hyperperiod (after this the whole system repeats itself)



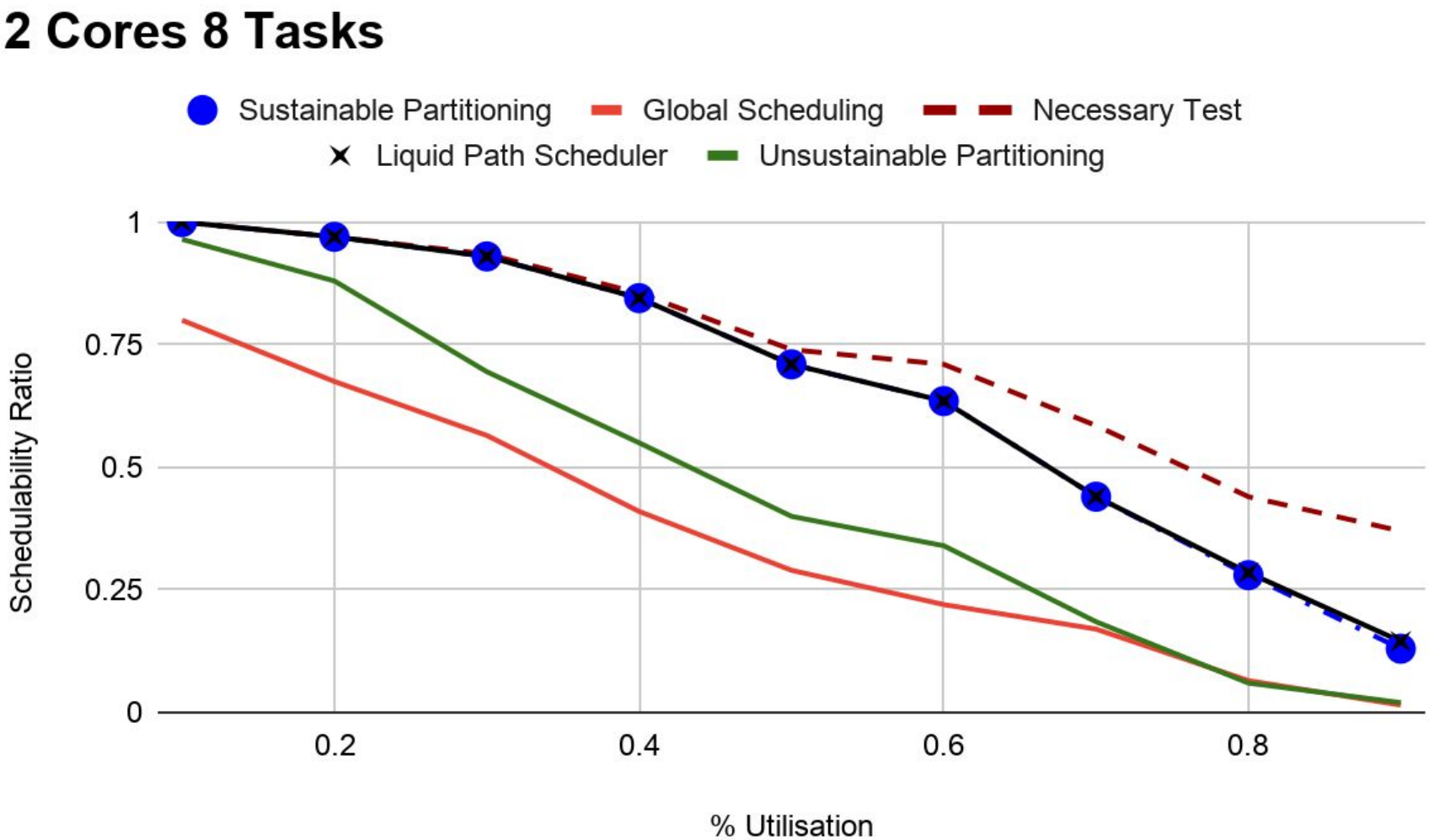


Evaluation



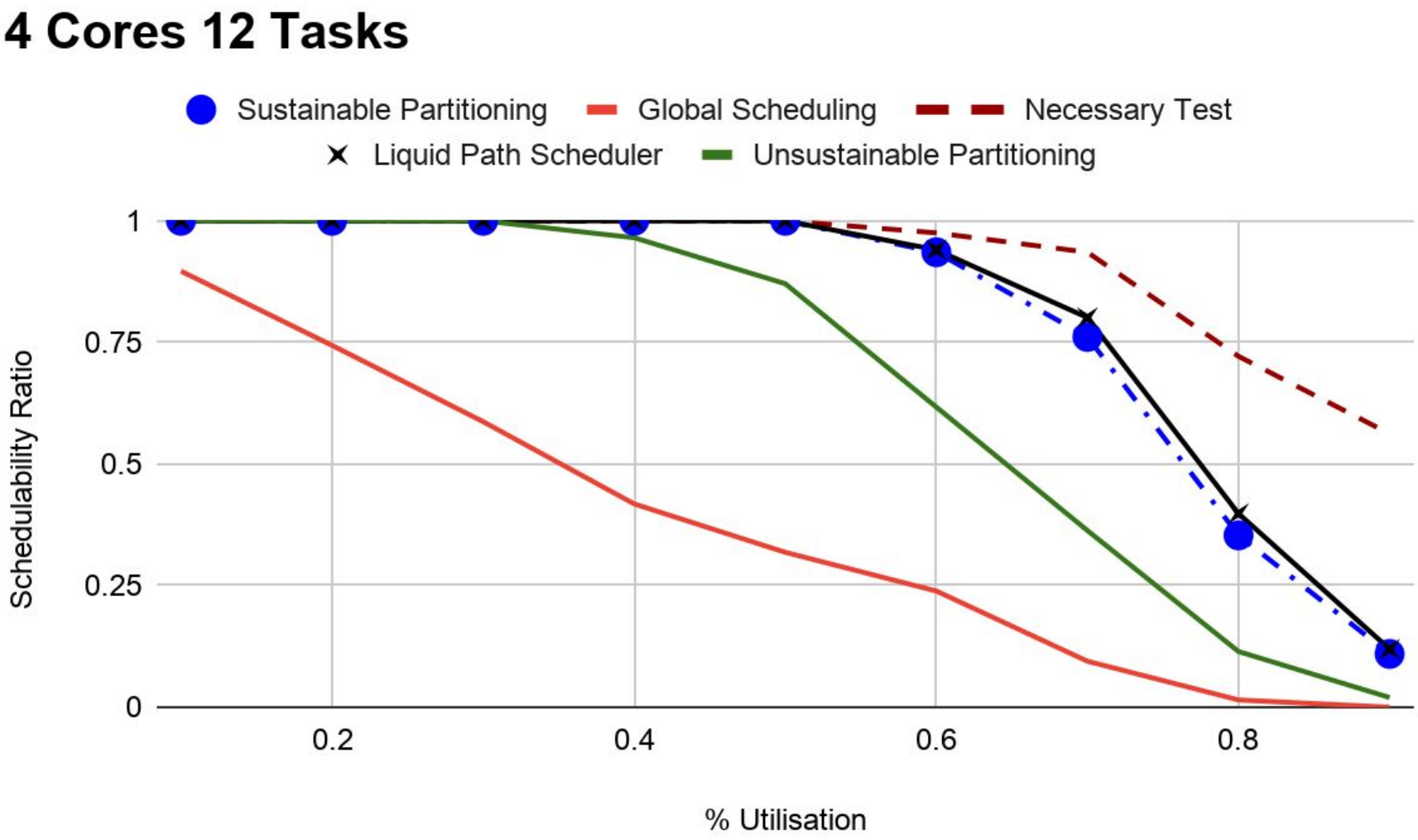
TUDelft

Evaluation



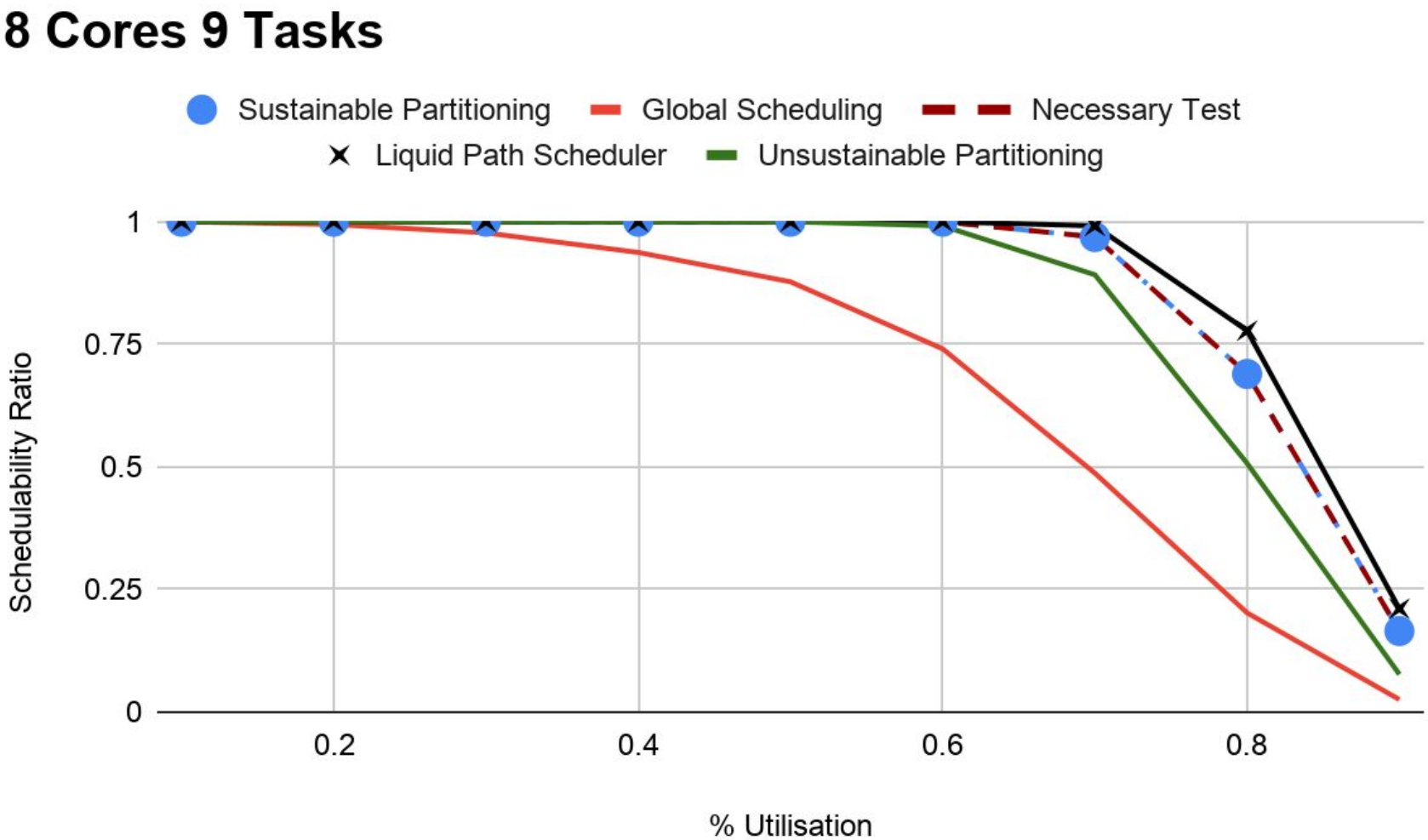
TU Delft

Evaluation



TU Delft

Un-partitionable task sets are schedulable

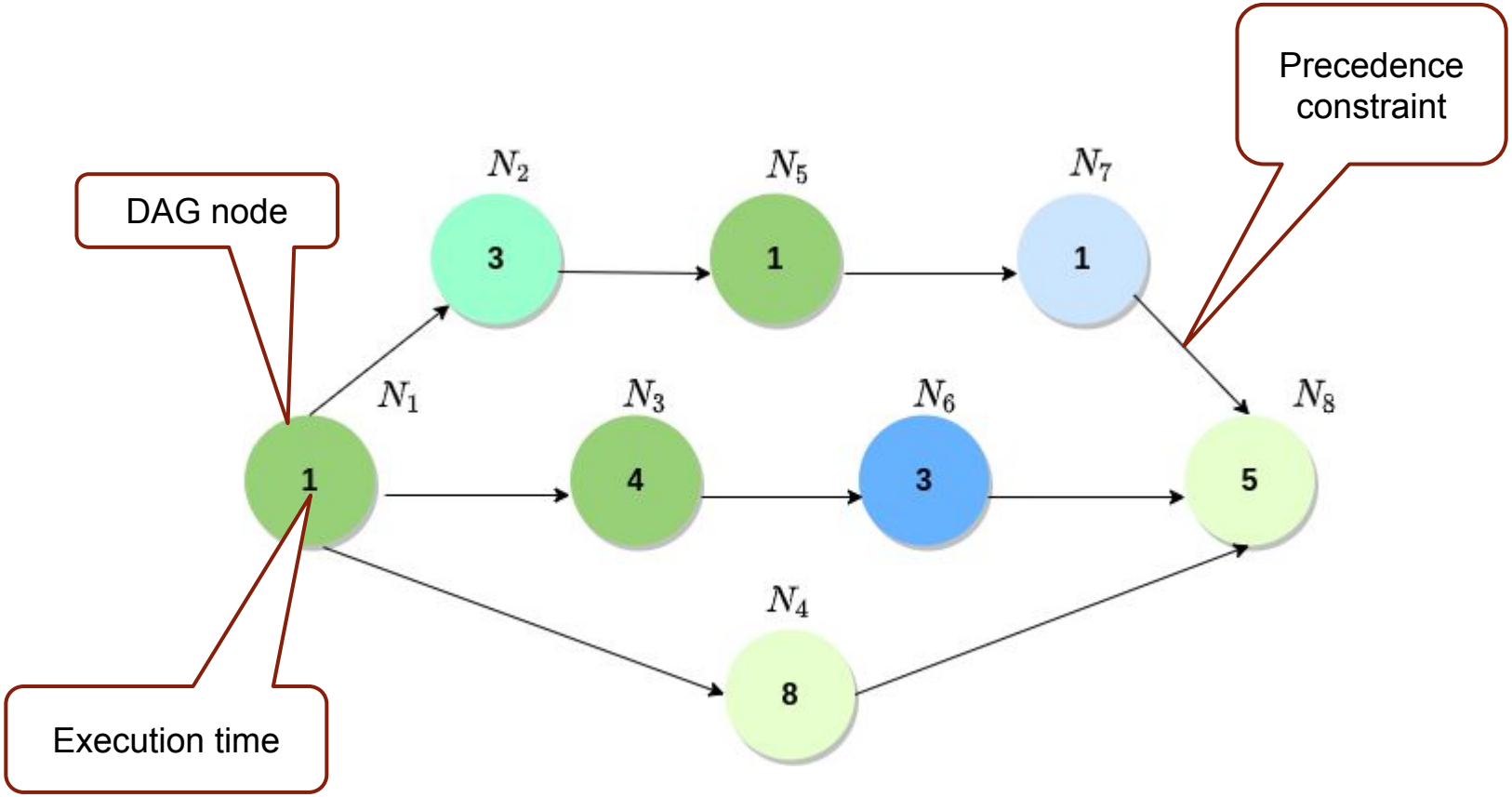


TU Delft

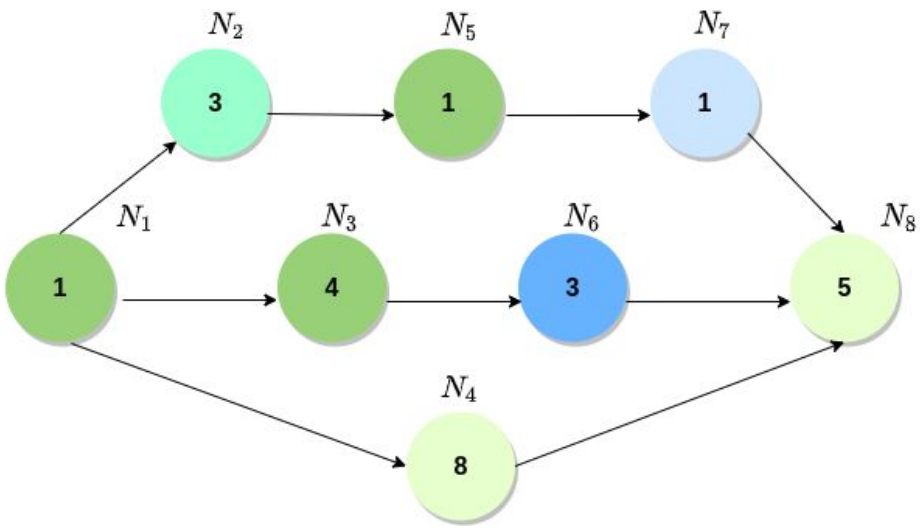
Can this improve parallel task scheduling?

TU Delft

The DAG model



DAG scheduling

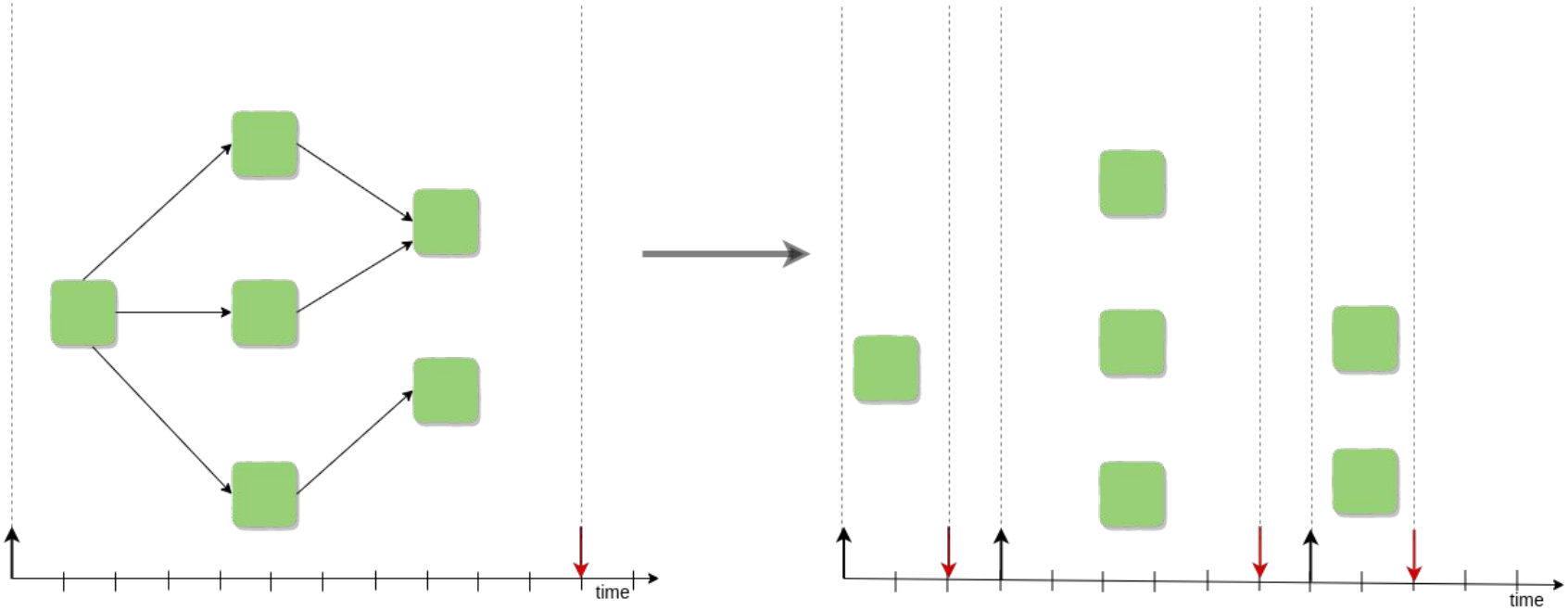


Directly schedule on a precedence aware scheduler

OR

Decompose the DAG into a set of sequential tasks and then schedule those

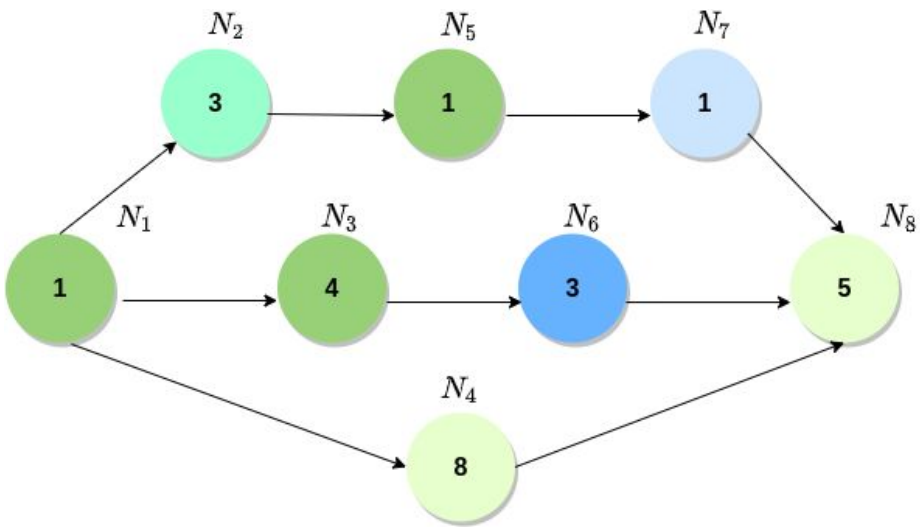
We choose decomposition!



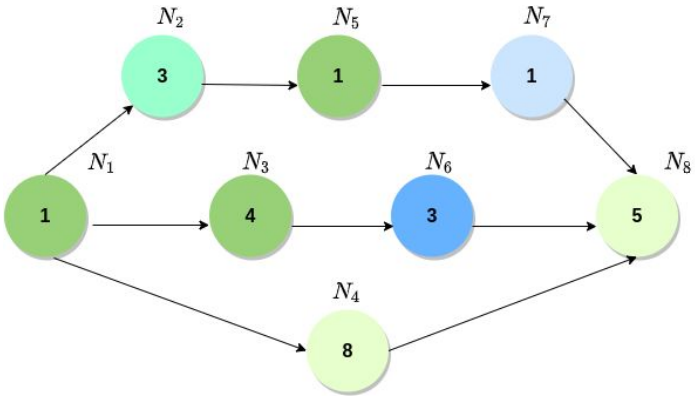
- Reduced complexity of the scheduler
- Sequential analysis applies
- Performance is highly dependent on how well we decompose

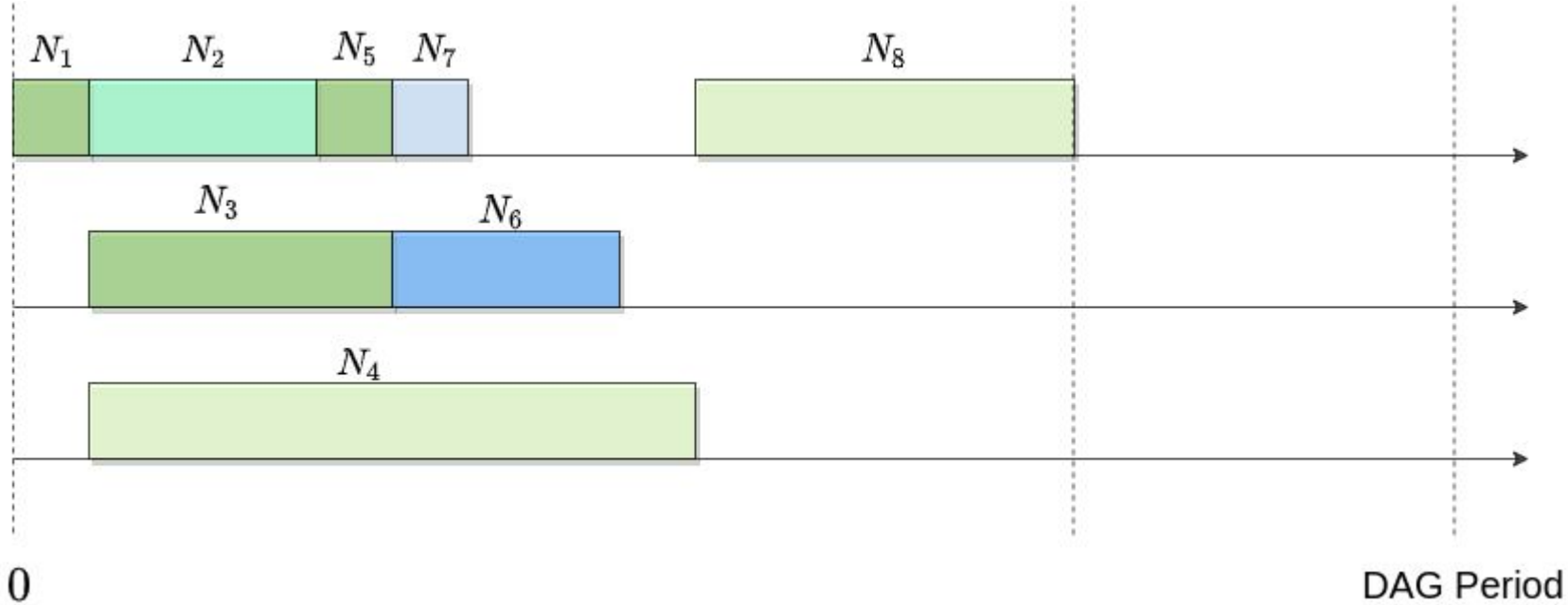
TU Delft

Our decomposition strategy

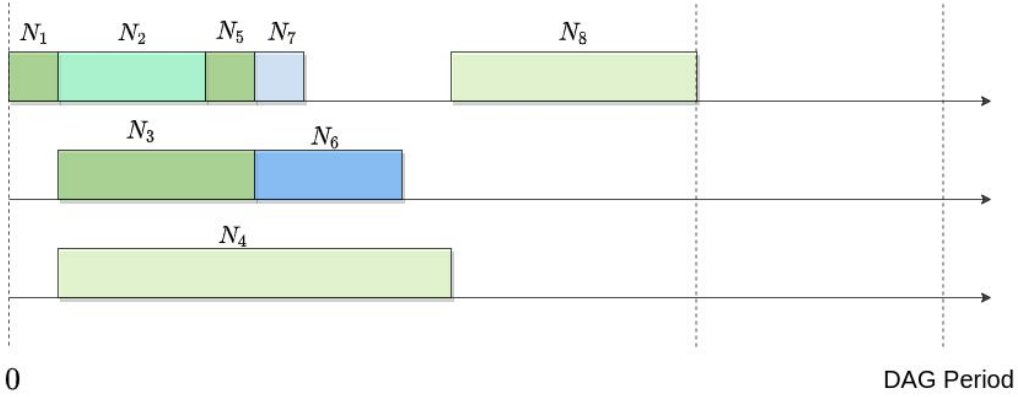


The ideal schedule

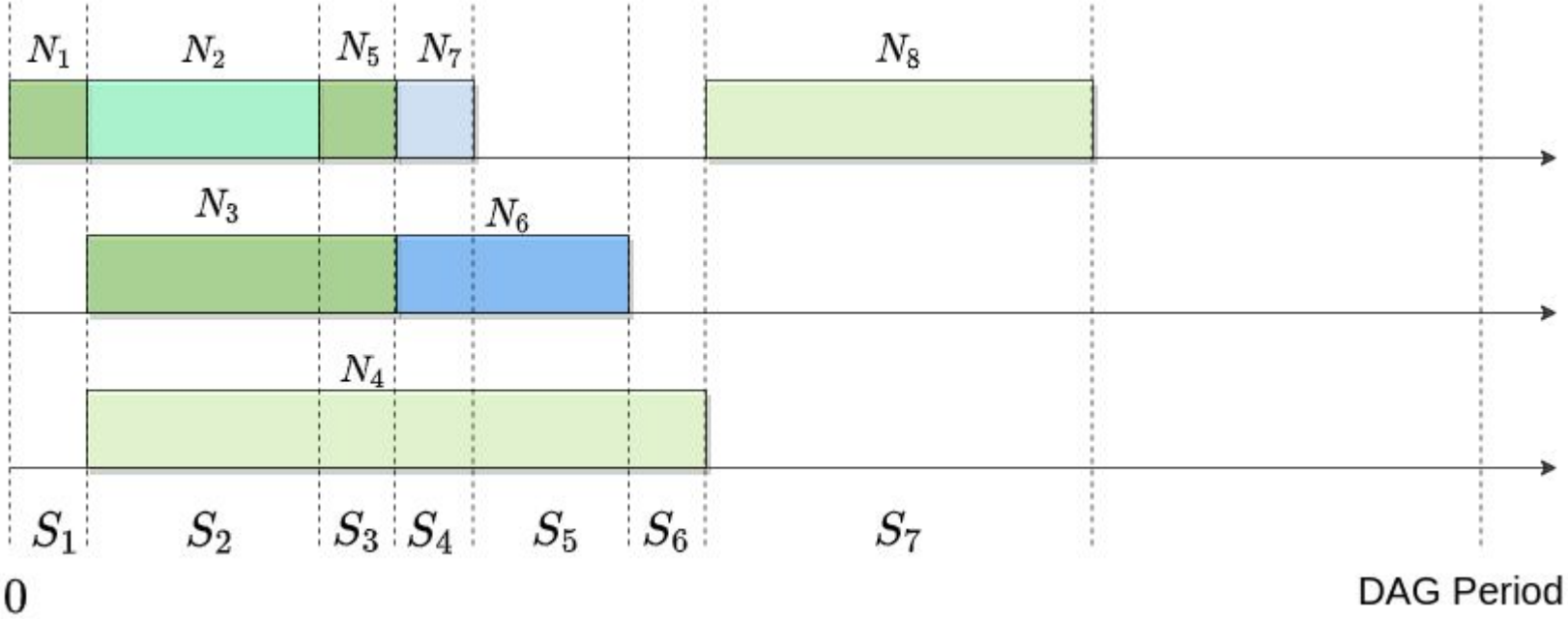




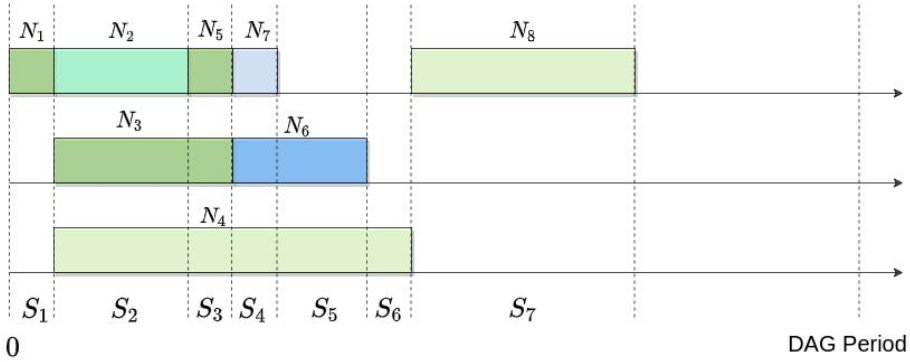
Sectioning the ideal schedule



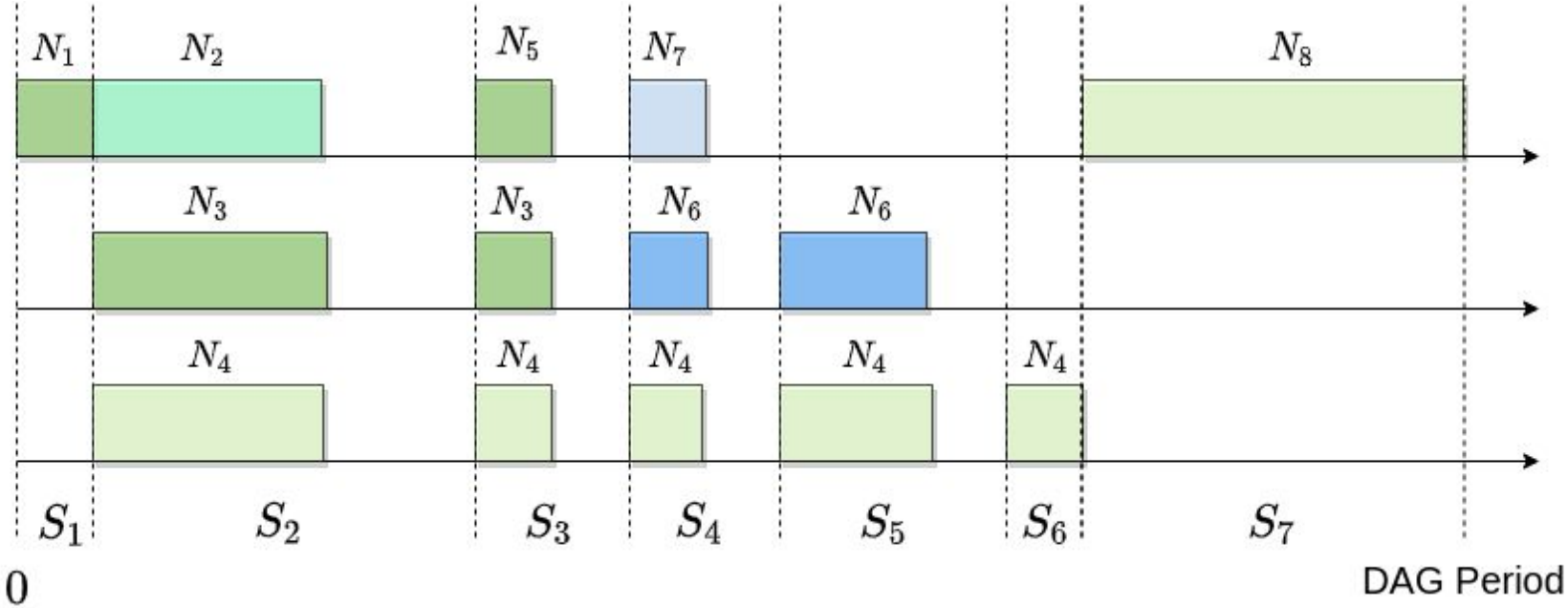
- **Separate ideal schedule into sections**



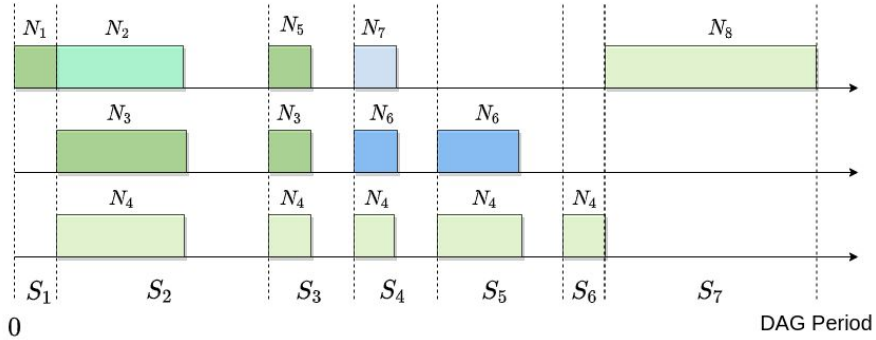
Assigning deadlines



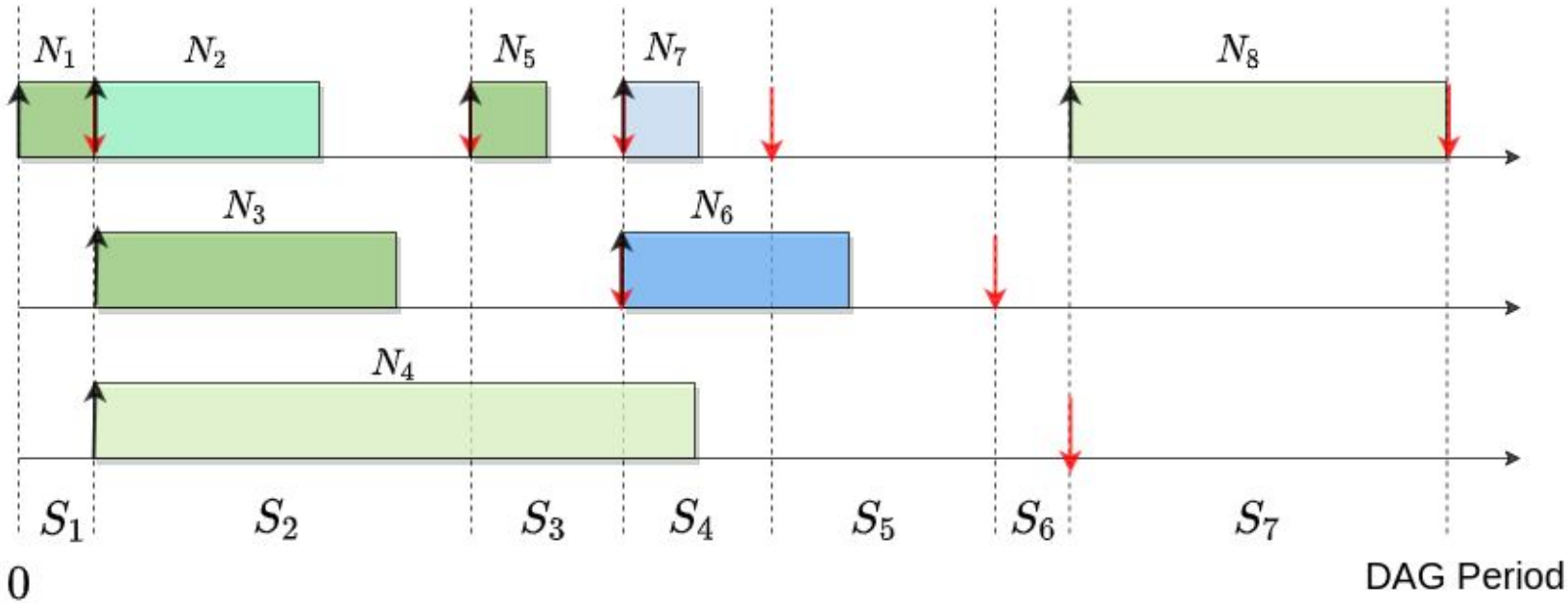
- **Assign deadlines and stretch schedule**



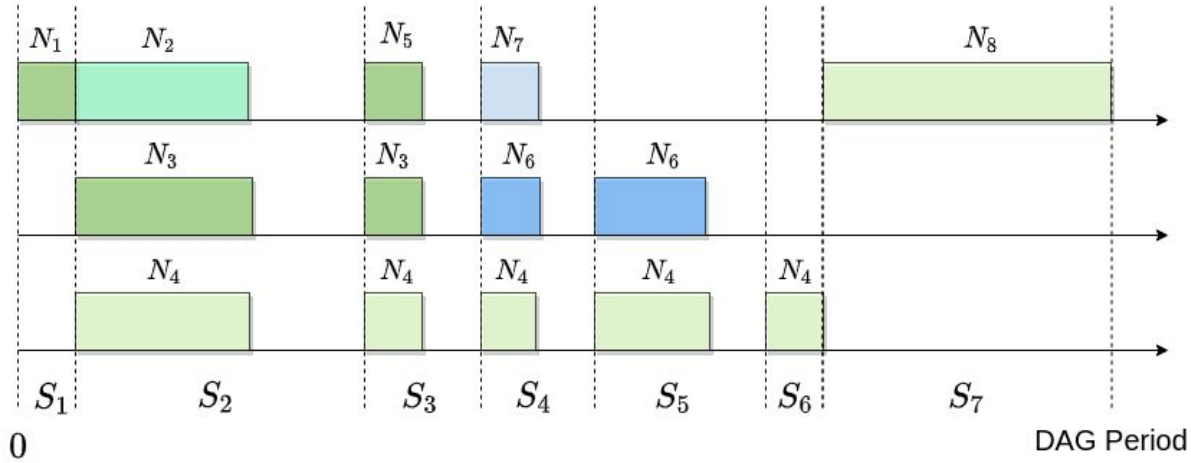
Restitching nodes



- **Re-stitch nodes for non-preemptive execution**

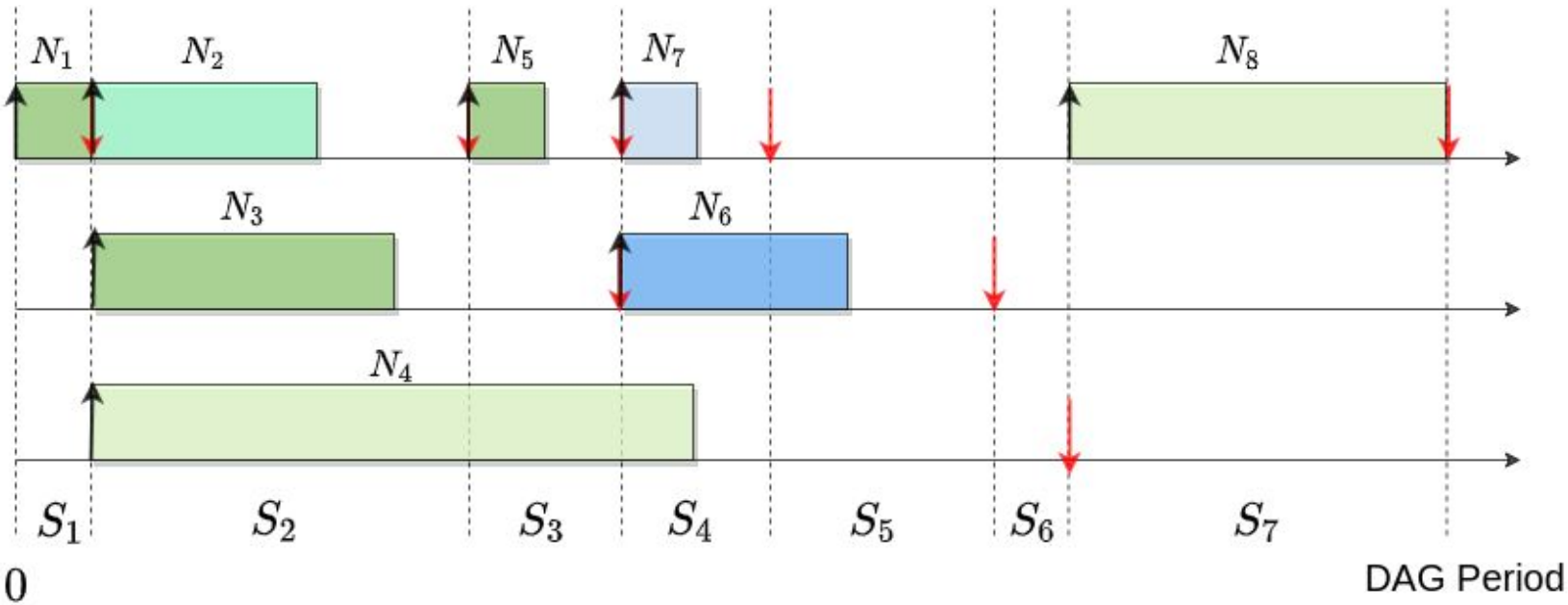


How do we assign deadlines?

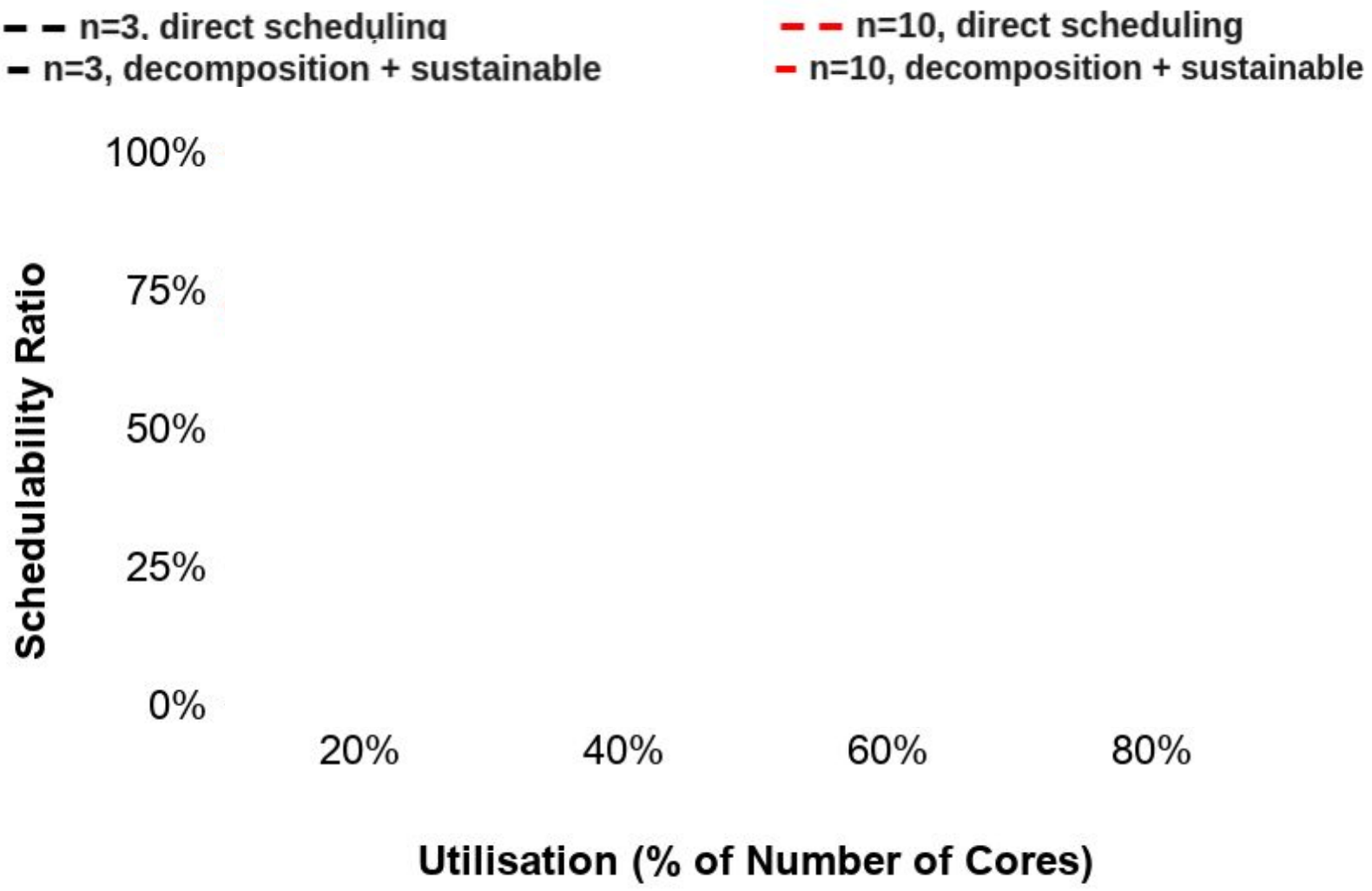


- **Give sections with more parallelism more slack**
- **Give sections with larger execution times more slack**
- **Attempt to keep density of threads less than 1**
- **Attempt to keep density of sections less than *m***

Scheduling decomposed DAG tasks

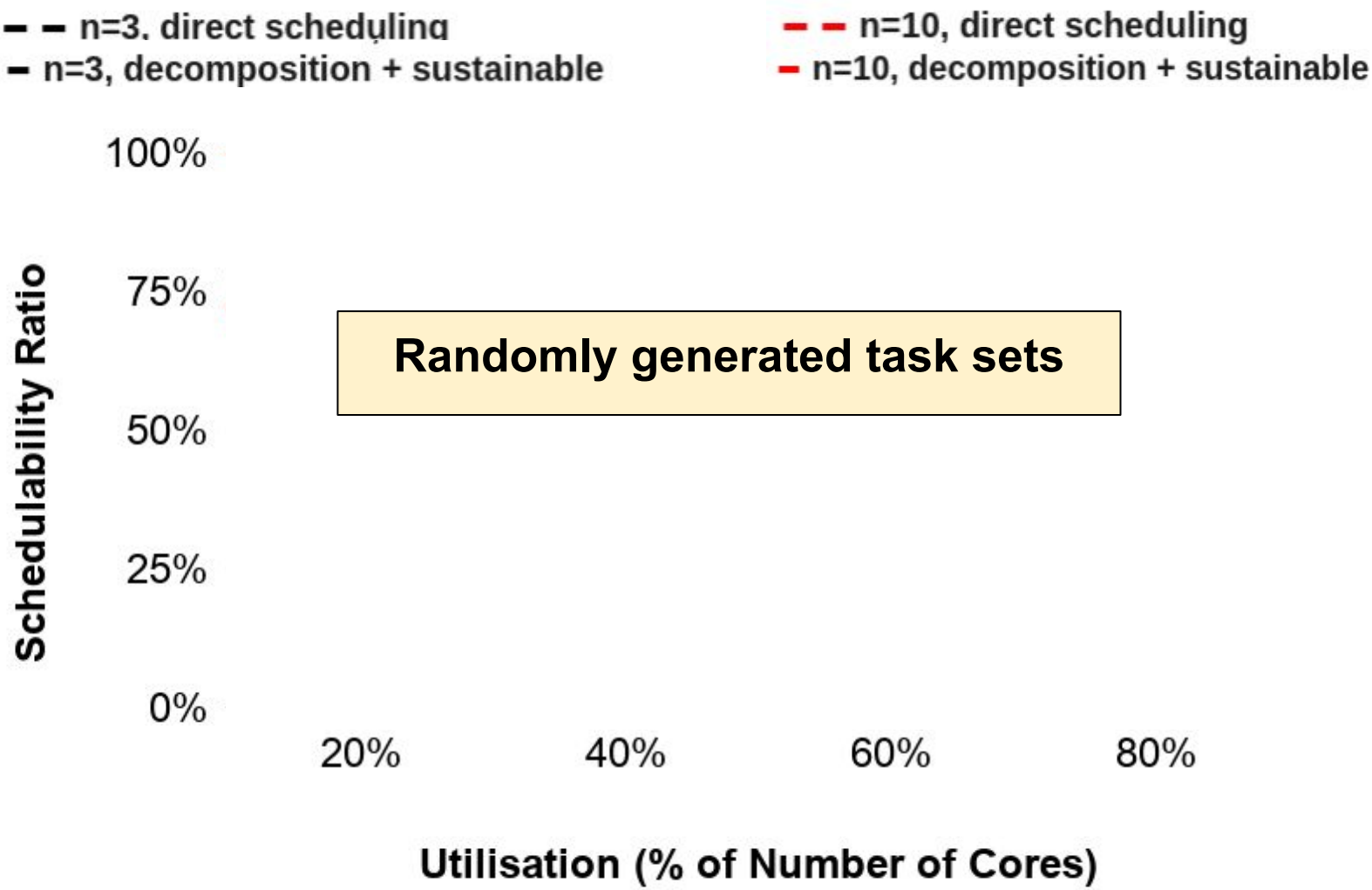


Evaluation



TU Delft

Evaluation

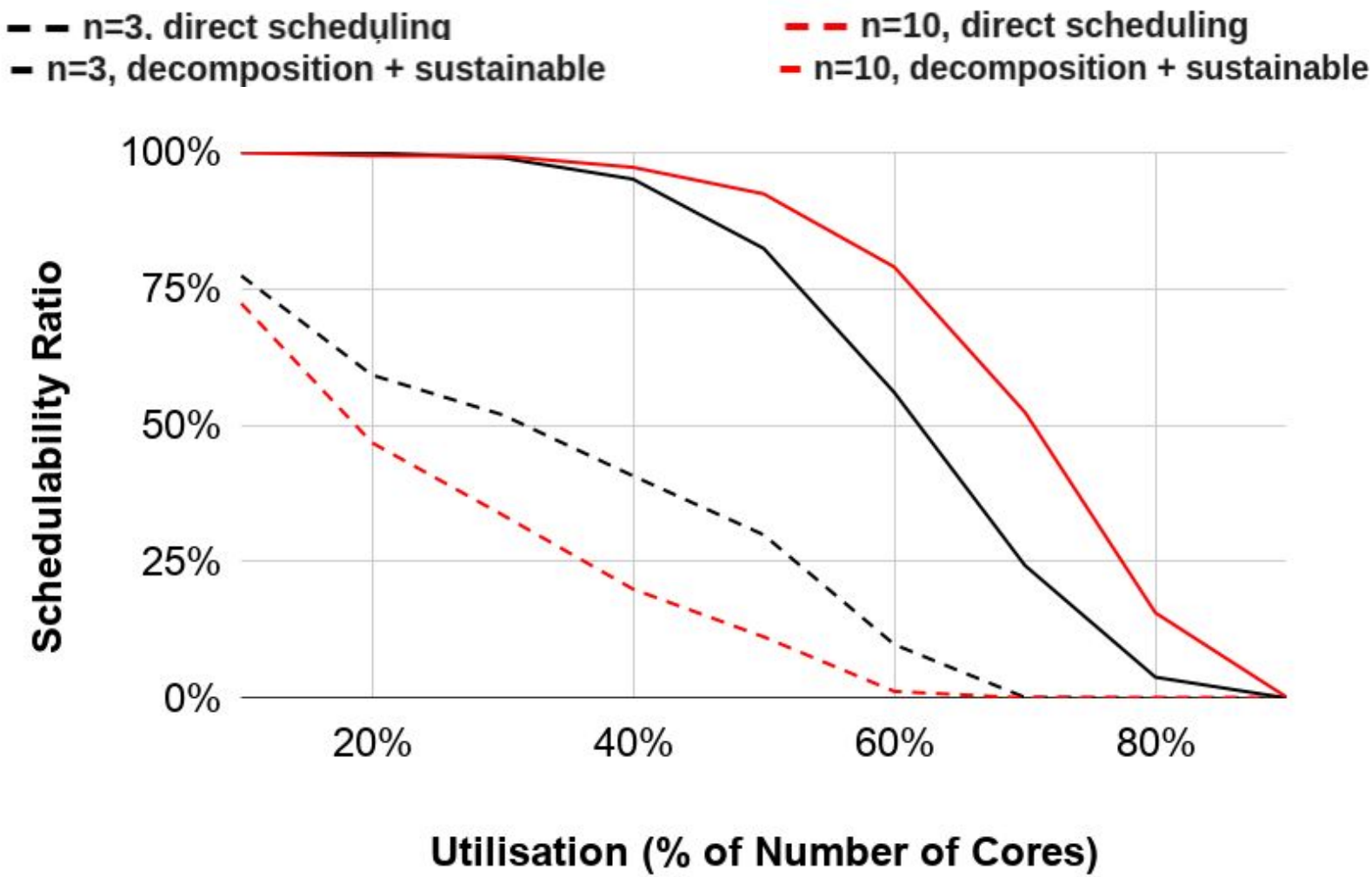


Randomly generated task sets

Maria A Serrano, Alessandra Melani, Marko Bertogna, and Eduardo Quiñones. “Response-time analysis of dag tasks under fixed priority scheduling with limited preemptions.” DATE, 2016.

Decomposition outperforms direct scheduling

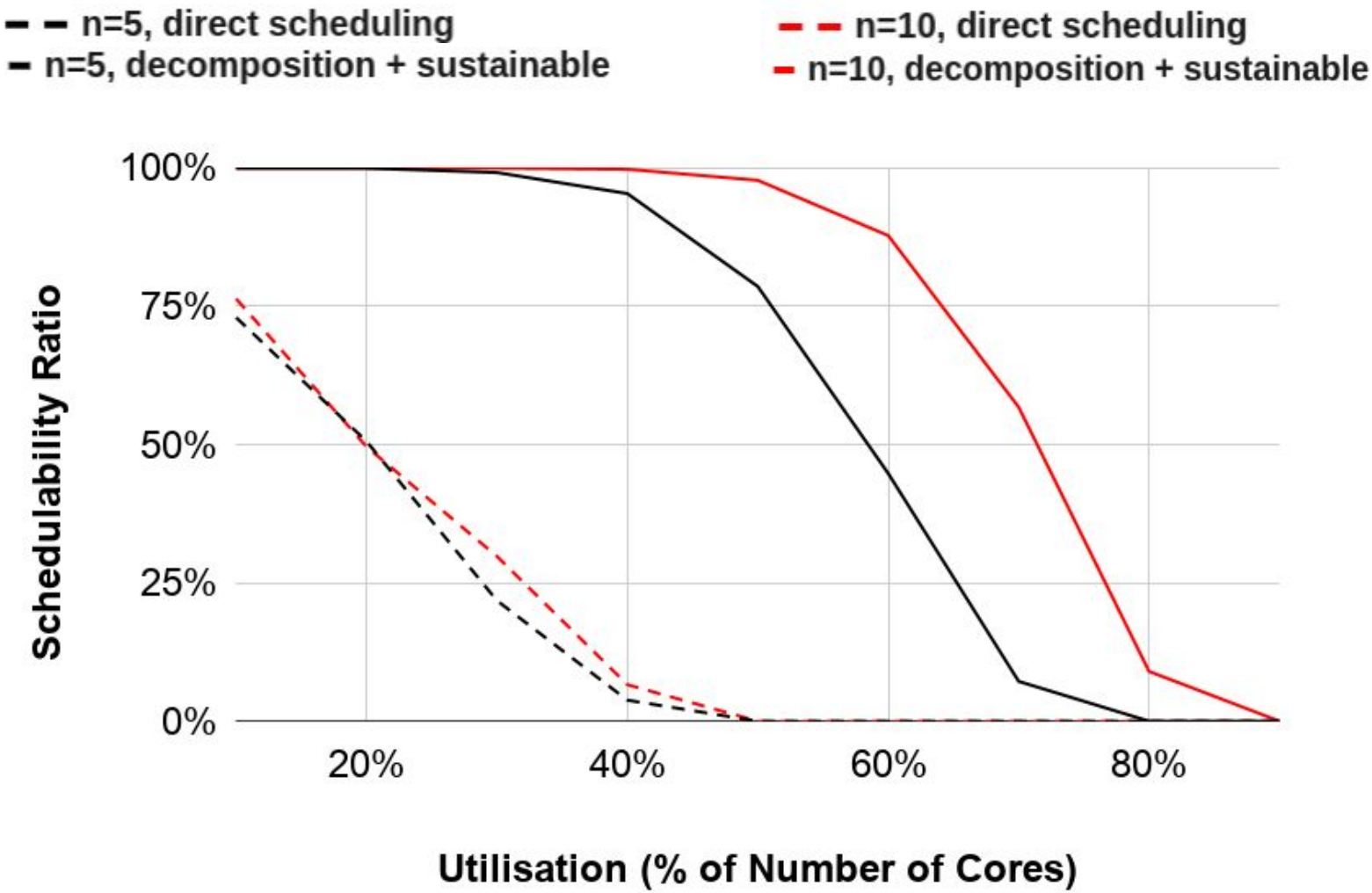
4 Cores



TU Delft

Decomposition outperforms direct scheduling

8 Cores



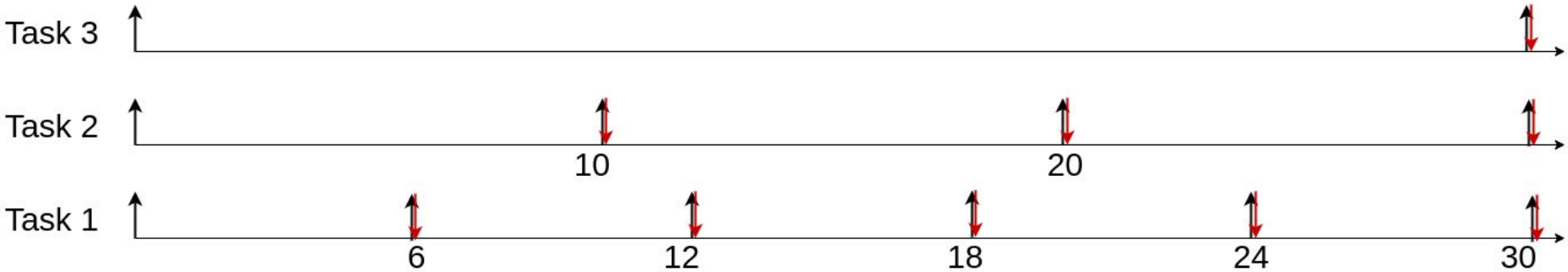
TU Delft

A sustainable low-overhead scheduler

TU Delft

FIFO as a sustainable scheduler

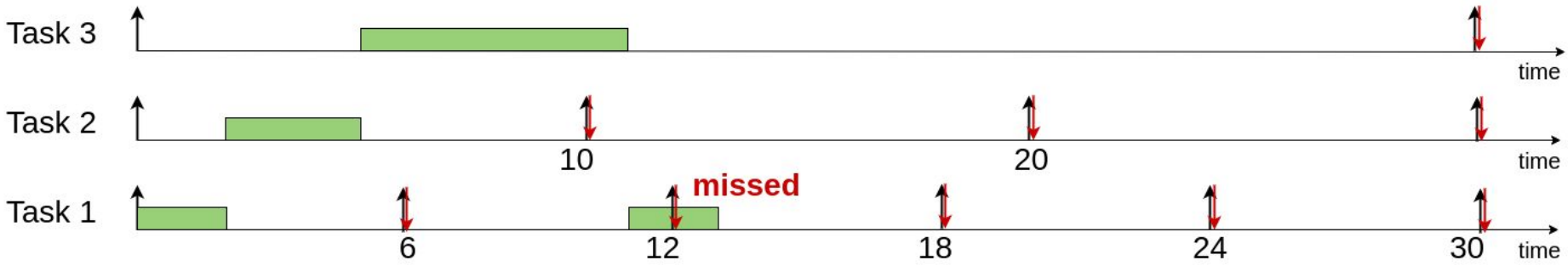
Task Number	Period	Execution Time
1	30	6
2	10	3
3	6	2



FIFO order of task releases

Task 1	**Task 2**	**Task 3**	**Task 1**	**Task 2**	**Task 1**	**Task 1**	**Task 2**	**Task 1**

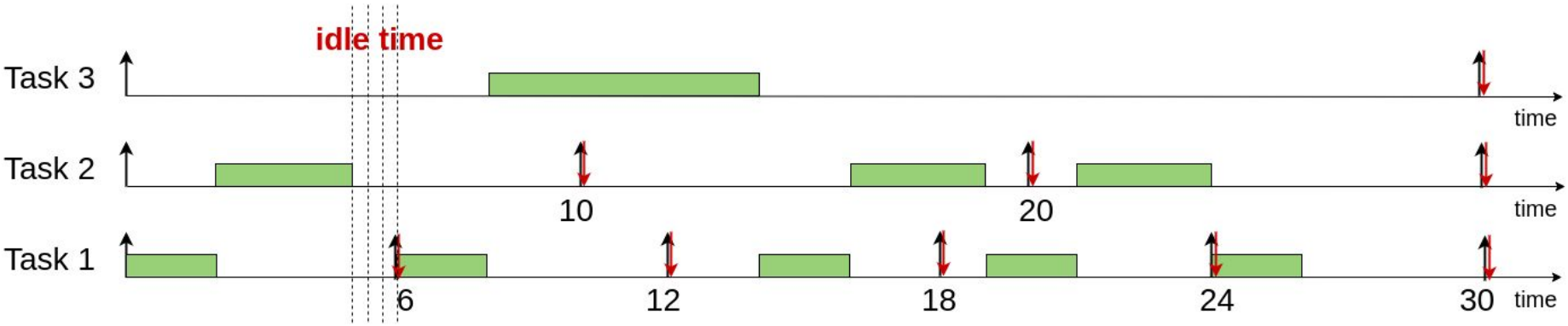
FIFO has low schedulability



FIFO order of task releases

Task 1	Task 2	Task 3	Task 1	Task 2	Task 1	Task 1	Task 2	Task 1

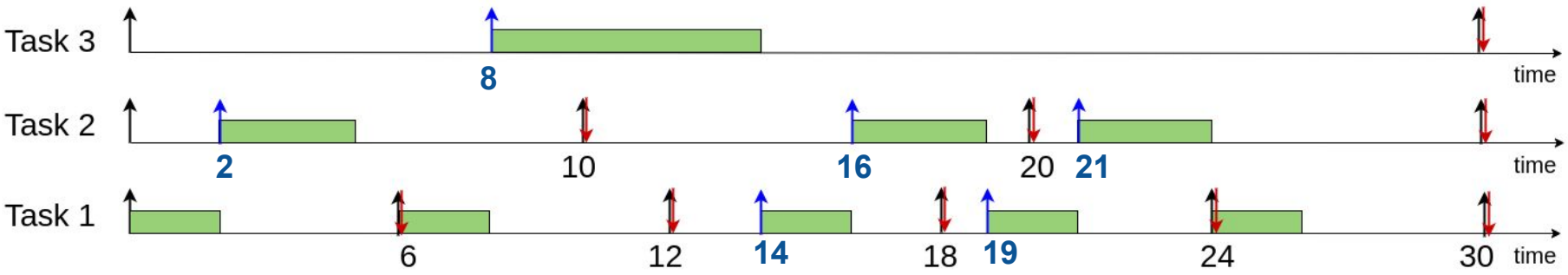
Recreating an offline schedule at runtime



Modified	Task 1	Task 2	Task 1	Task 3	Task 1	Task 2	Task 1	Task 2	Task 1
Original	Task 1	Task 2	Task 3	Task 1	Task 2	Task 1	Task 1	Task 2	Task 1

TU Delft

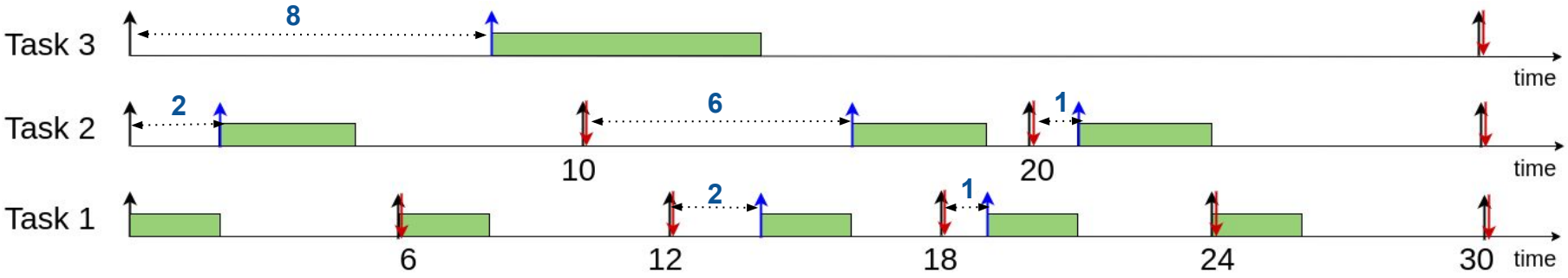
Building an offset table



Modified	Task 1	Task 2	Task 1	Task 3	Task 1	Task 2	Task 1	Task 2	Task 1
Original	Task 1	Task 2	Task 3	Task 1	Task 2	Task 1	Task 1	Task 2	Task 1

TU Delft

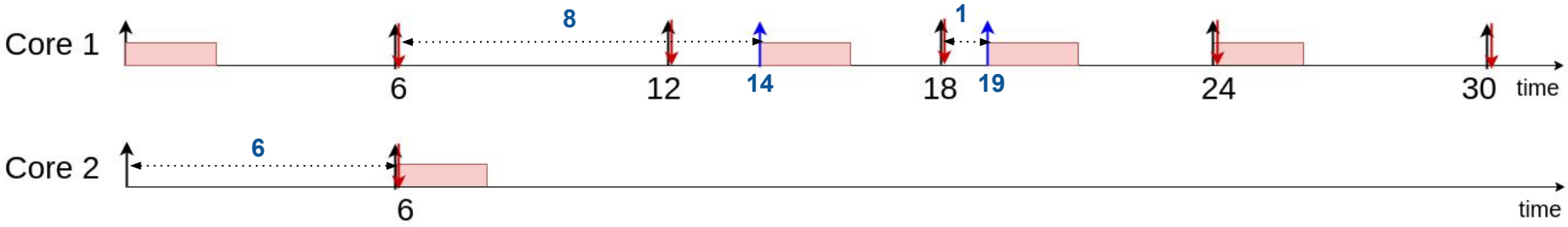
Building an offset table



Modified	Task 1	Task 2	Task 1	Task 3	Task 1	Task 2	Task 1	Task 2	Task 1
Original	Task 1	Task 2	Task 3	Task 1	Task 2	Task 1	Task 1	Task 2	Task 1

TU Delft

Building a multiprocessor offset table



- **Mimic partitioning**
  - Only look from the perspective of one core at a time
- **Handle migration**
  - Make it possible to have offsets larger than the period
  - This makes the core skip the jobs that don't belong to it

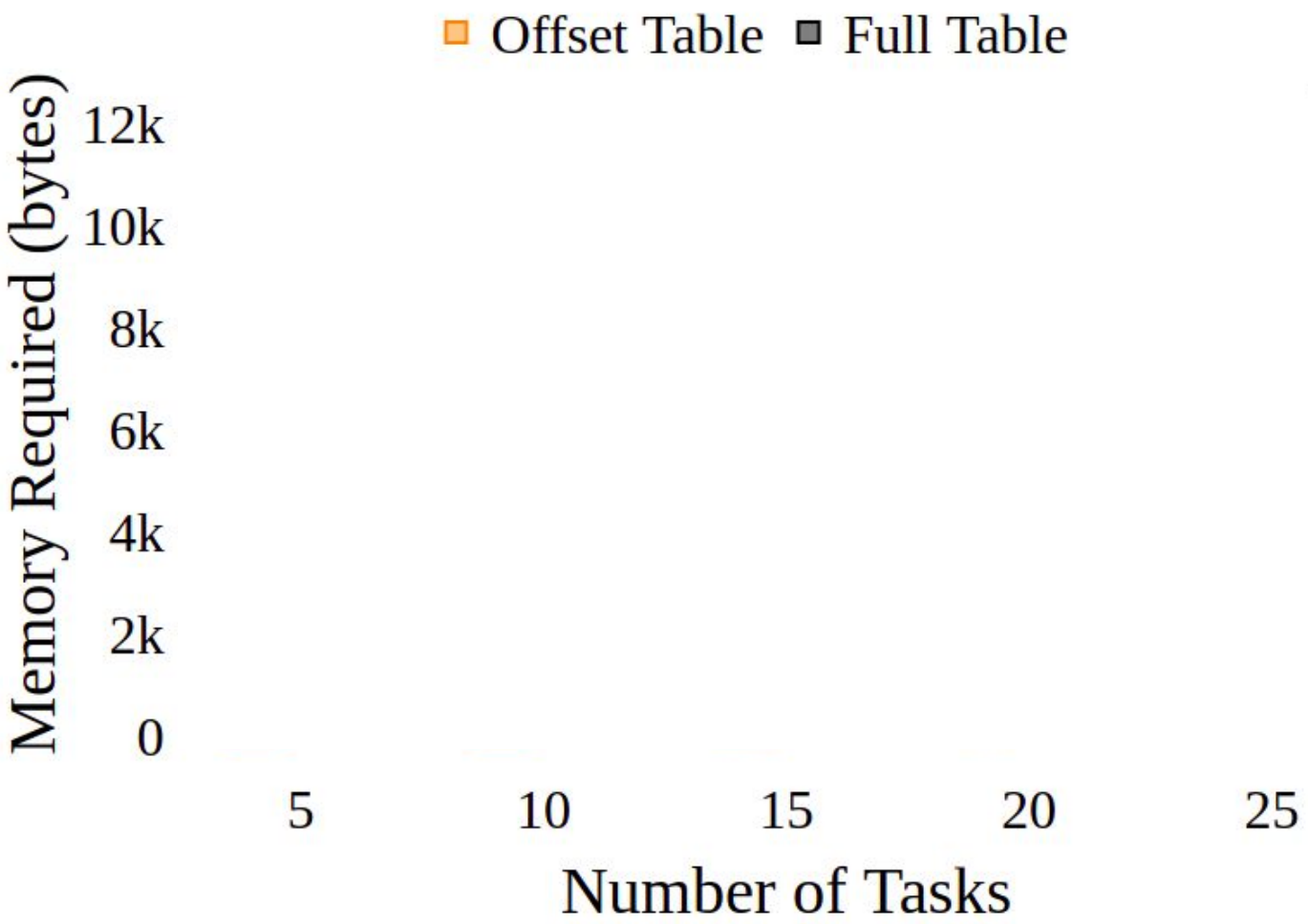
Implementation of multiprocessor FIFO-OT

- **Chosen platform**
  - Raspberry pi 3
  - 4 Cores
  - Open source enough for bare metal OS

- **Measure how much overhead is incurred**
  - In terms of storage cost
  - In terms of processing time of the scheduler

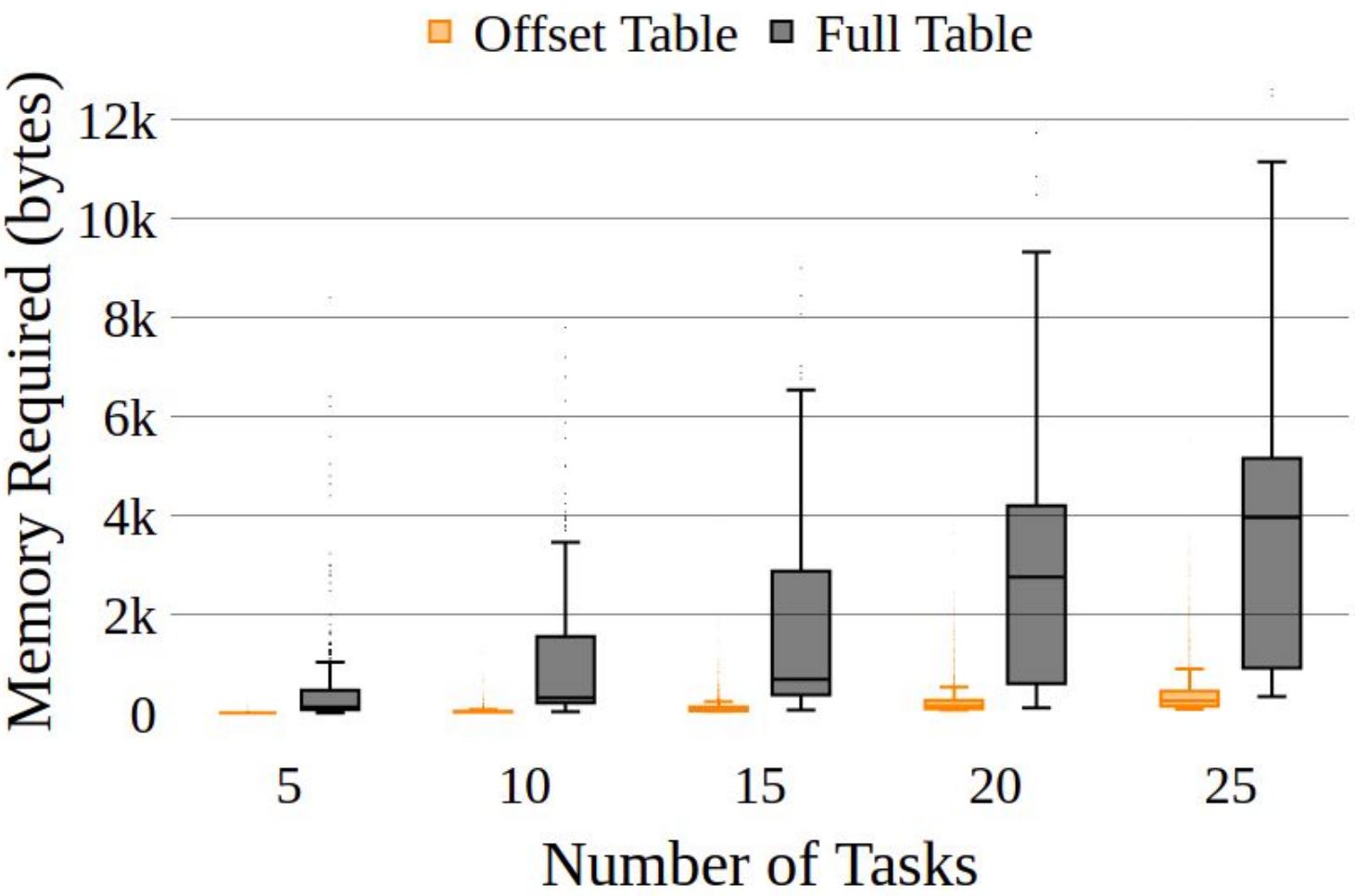
TU Delft

Memory consumption



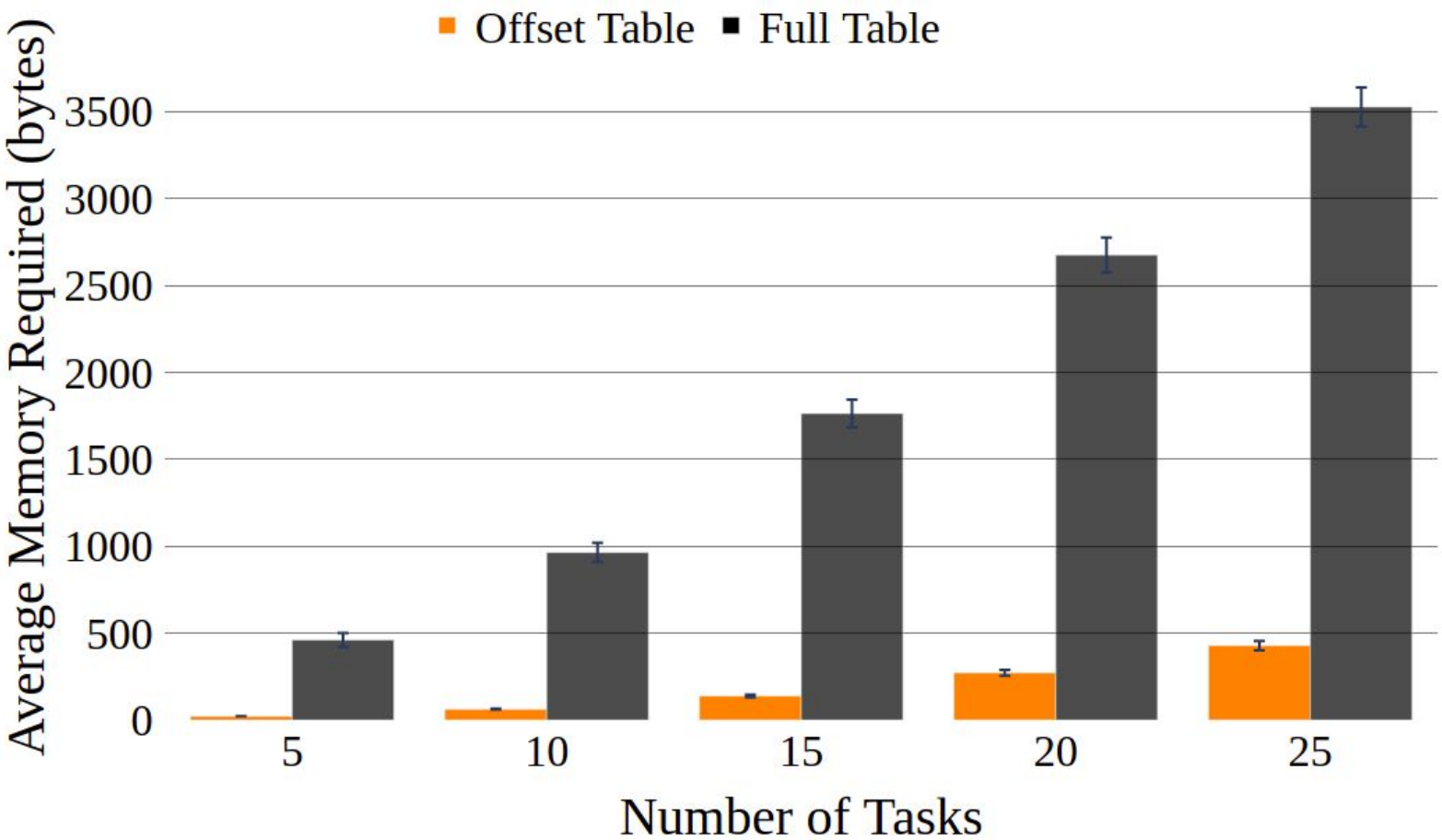
TU Delft

Memory consumption



TU Delft

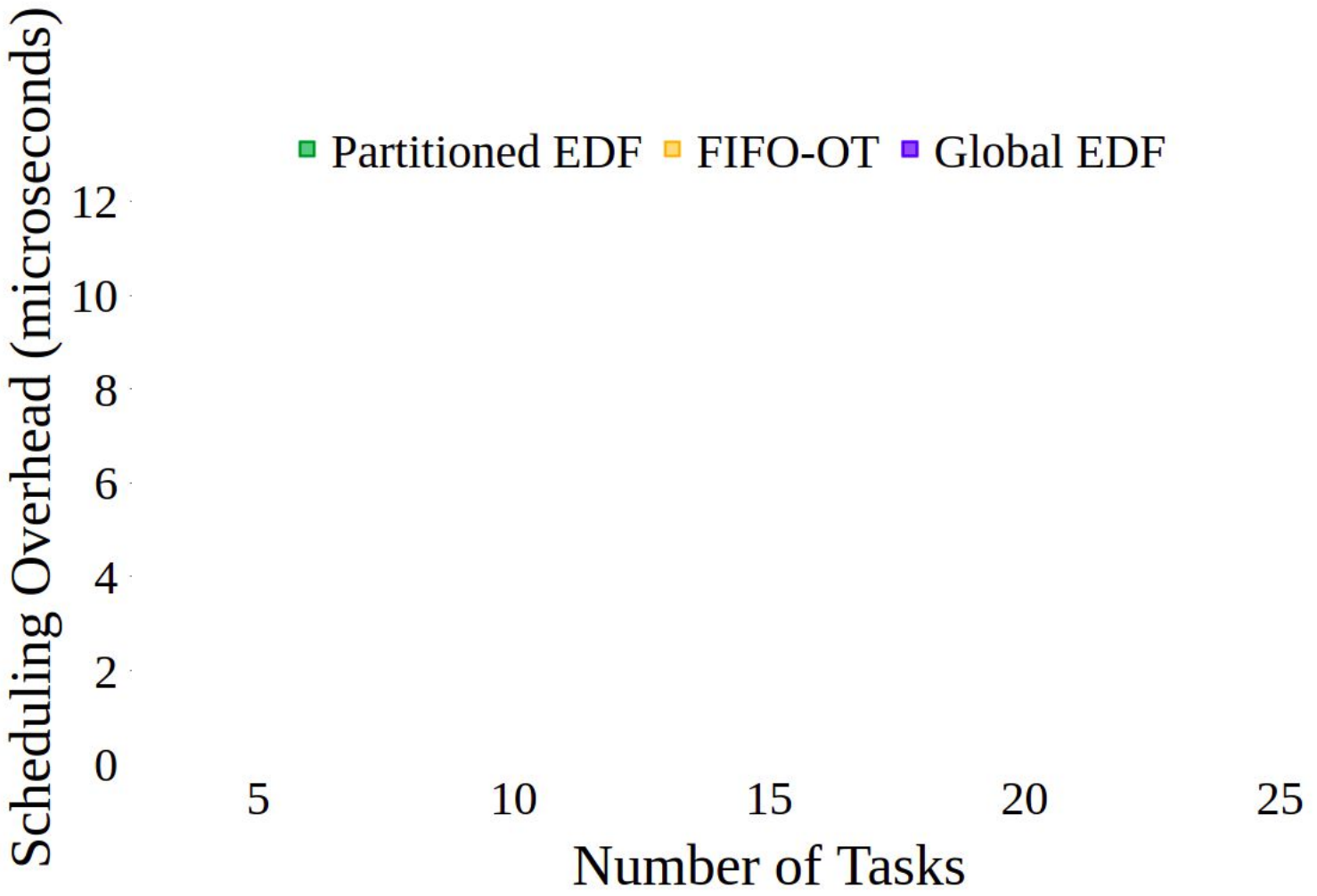
Memory consumption



Error bars represent 95% confidence intervals

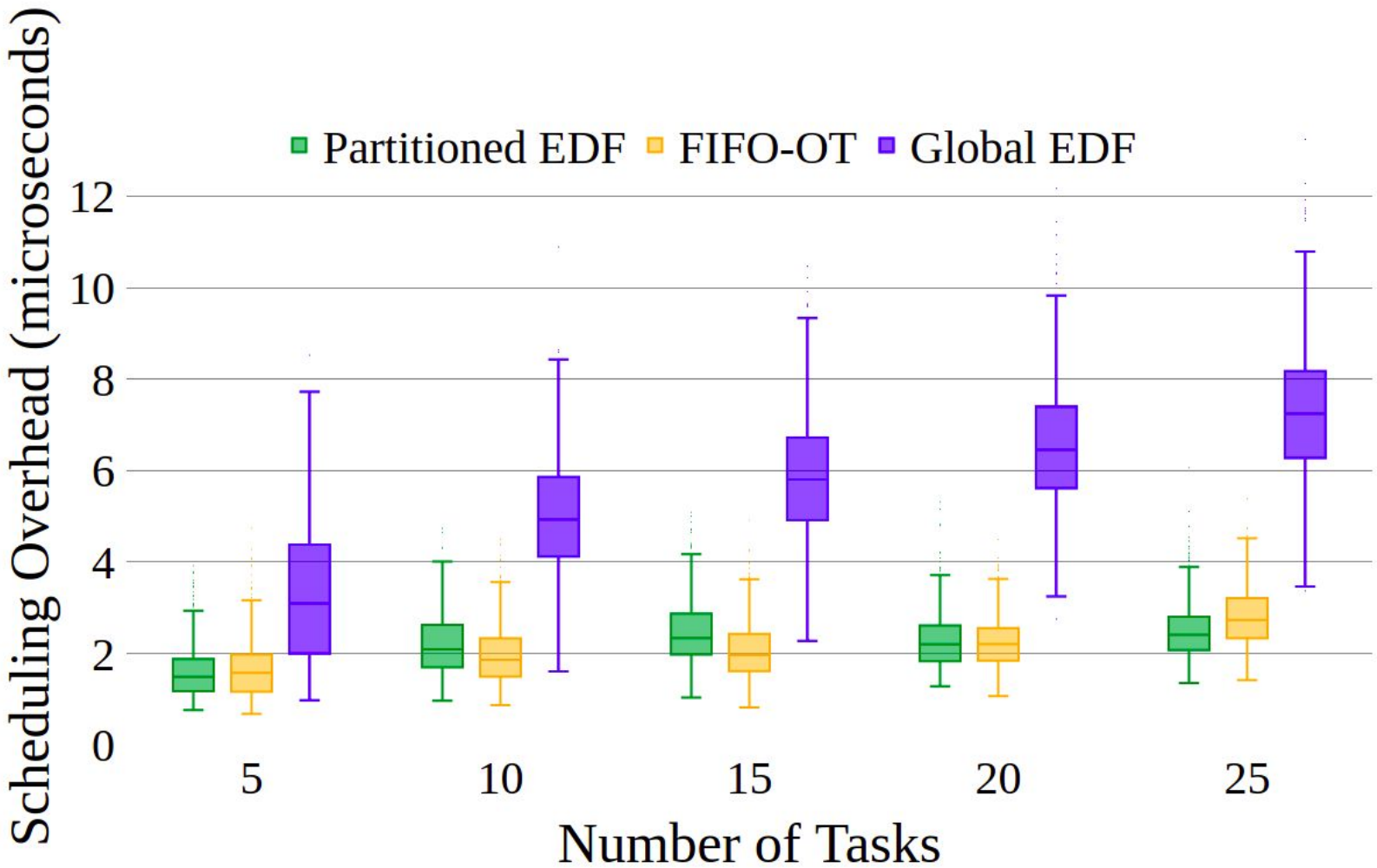
TU Delft

Scheduling overhead



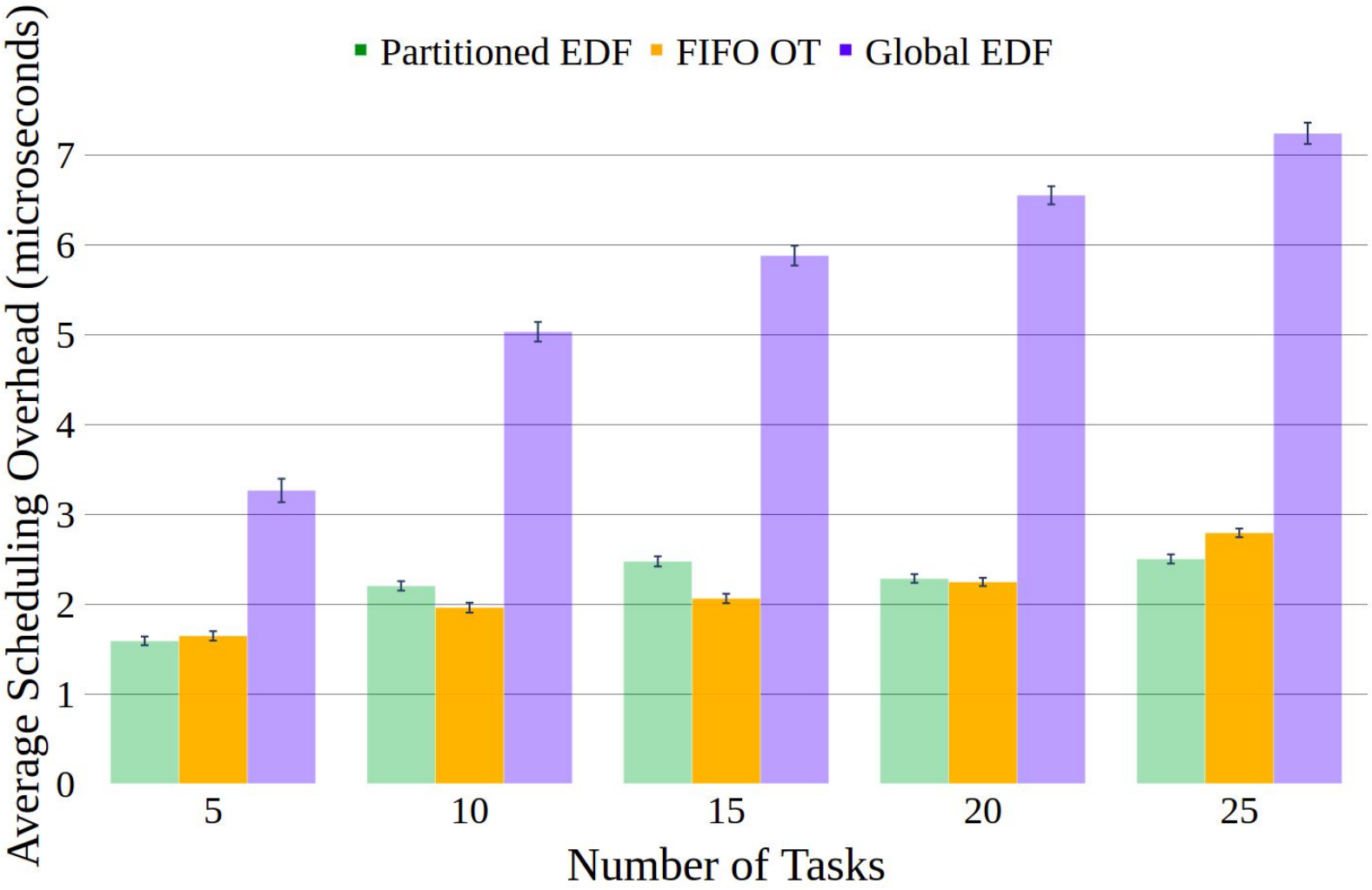
TU Delft

Scheduling overhead



TU Delft

Scheduling overhead



Error bars represent 95% confidence intervals

TU Delft

Summary

Summary

Thesis Roadmap

What is the true limit of partitioning?

A necessary test for partition-ability

What if partitioning fails?

A semi-partitioning scheme

Can this improve parallel task scheduling?

Decompose the parallel task into sequential parts

How efficient are our algorithms at runtime?

Extend FIFO-OT to multiprocessor platforms

TU Delft

Conclusions

- **Sustainable partitioning** or **semi-partitioning** as a strategy gives **higher schedulability ratios** with **smaller runtime overheads.**

- The limits of partitioning depends on the amount of conflicts present in the task set.

- **Sustainable partitioning of decomposed DAG tasks** was found to surpass direct scheduling in all examined cases.

- **Sustainable multiprocessor scheduling** is achievable with **scheduling overhead of a partitioned solution** and using **90% less memory** than a table scheduler on average.

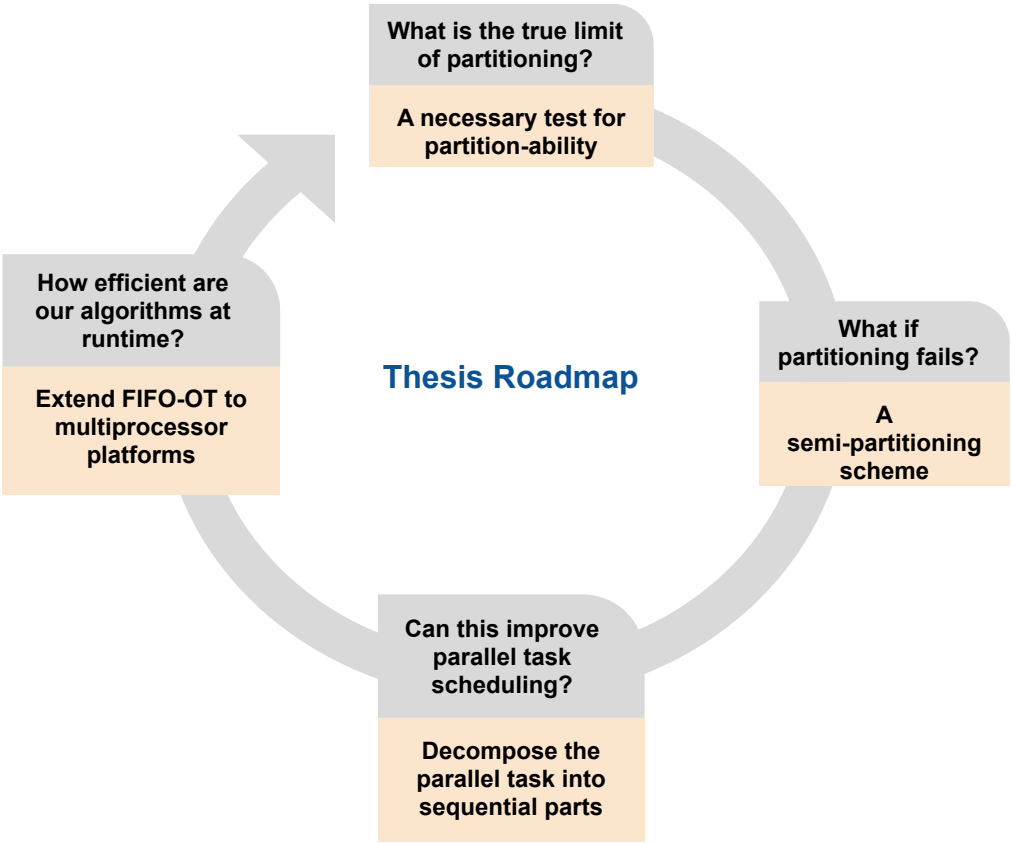
TU Delft

Future work

- Handle sporadic task sets
- Corunner aware partitioning
- A test for task set feasibility

TU Delft

THANK YOU



TUDelft