MAX PLANCK INSTITUTE
FOR SOFTWARE SYSTEMS

MAX-PLANCK-GESELLSCHAFT

CISTER
Research Centre in
Real-Time & Embedded
Computing Systems

A **Response-Time Analysis** for **Non-Preemptive Job Sets** under **Global Scheduling**

Mitra Nasri Geoffrey Nelissen Björn B. Brandenburg

ECRTS 2018

Our work in a nutshell

Applicable to
- Irregular release patterns
- Bursty releases
- And periodic tasks with/without offset

Multiprocessor platform

Non-preemptive job sets

Hard or soft timing constraints

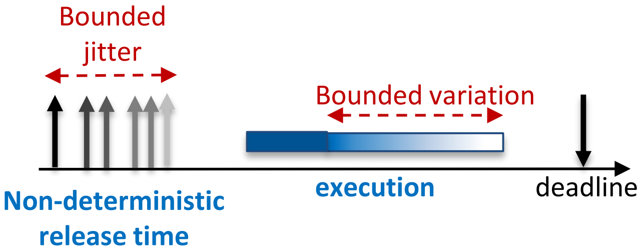
Release jitter

Execution time variation

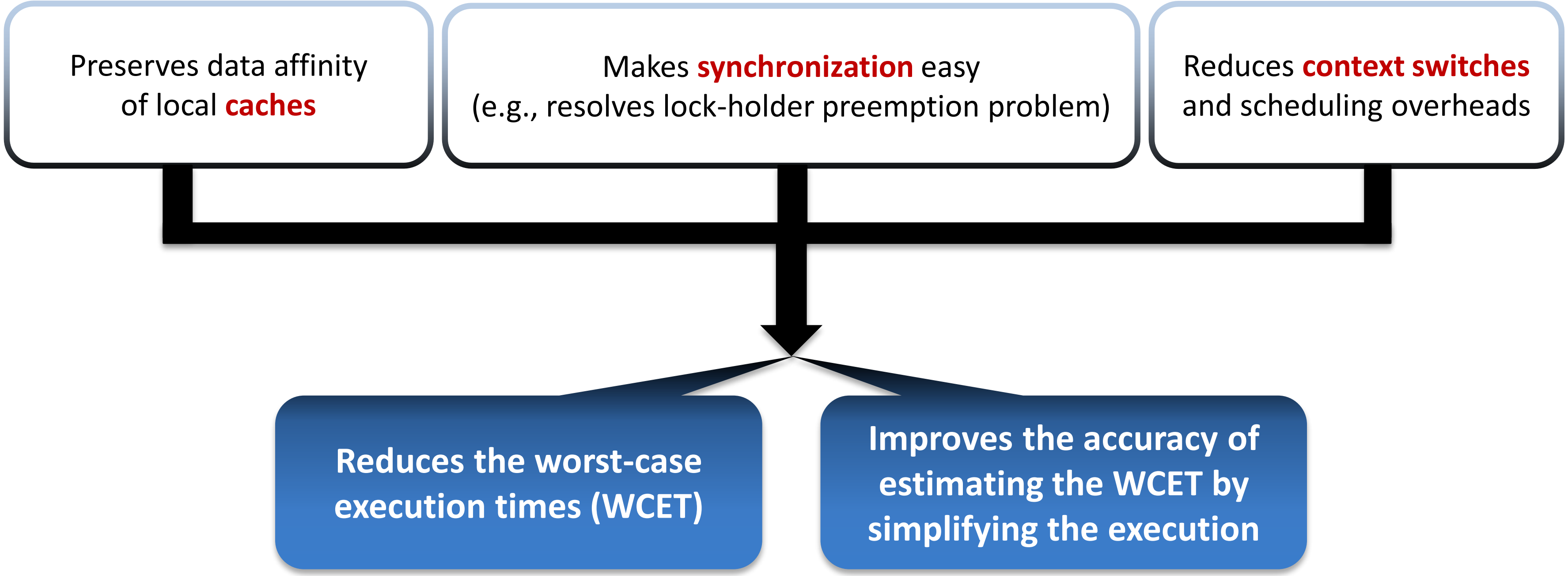
Per job best-case and worst-case response time (BCRT and WCRT)

Global job-level fixed-priority (JLFP) work-conserving scheduling policies

Global EDF
Global fixed priority
Global RM
…



Why non-preemptive scheduling for multiprocessor platforms?



Preserves data affinity of local **caches**

Makes **synchronization** easy (e.g., resolves lock-holder preemption problem)

Reduces **context switches** and scheduling overheads

Reduces the worst-case execution times (WCET)

Improves the accuracy of estimating the WCET by simplifying the execution

Hence, it can be used to make multiprocessor platforms more (time) predictable

State of the art on global non-preemptive scheduling

Schedulability of global JLFP non-preemptive policies for

Finite job sets*	Periodic tasks with offset	Synchronous periodic tasks	Sporadic tasks
open problem ✓	**open problem** Analysis od sporadic tasks is applicable but very pessimistic ✓	**open problem** Analysis od sporadic tasks is applicable but very pessimistic ✓	**Exact analysis** **Open problem** **Sufficient analyses** • Global fixed-priority [Baruah06, Guan08, Guan11, Lee14, Lee17] • Global EDF [Baruah06, Guan08] • General work-conserving policy [Baruah06, Guan08]

We derive a response-time bound for these cases

Applicable to

- Irregular release patterns
- Bursty releases
- Frame-based tasks
- …

* In finite job sets, each job is known by its release time, release jitter, best-case and worst-case execution times, and deadline

JLFP: job-level fixed-priority

Contributions

A response-time analysis

for **a wide class** of **global scheduling** policies

based on searching the space of possible schedules

We use and extend the notion of schedule-abstraction graphs [RTSS'17]

(recently introduced to analyze uniprocessor non-preemptive scheduling)

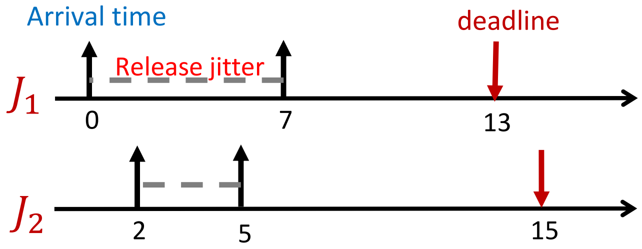
[RTSS'17] Mitra Nasri and Björn B. Brandenburg, "An Exact and Sustainable Analysis of Non-Preemptive Scheduling", RTSS, 2017, pp. 1-12.

Schedule-Abstraction Graphs

(definition, usage, and construction)

Key challenges in the schedulability analysis of job sets

(with non-deterministic parameters)



Arrival time [Audsley93]

Latest release time

Best-case execution time (BCET)

Worst-case execution time (WCET)

Job	**Release time**		**Deadline**	**Execution time**		**Priority**
	Min	**Max**		**Min**	**Max**	
J_1	0	7	13	1	5	high
J_2	2	5	15	2	5	low

Since there is no periodicity assumption about job releases, finding a worst-case scenario is fundamentally hard

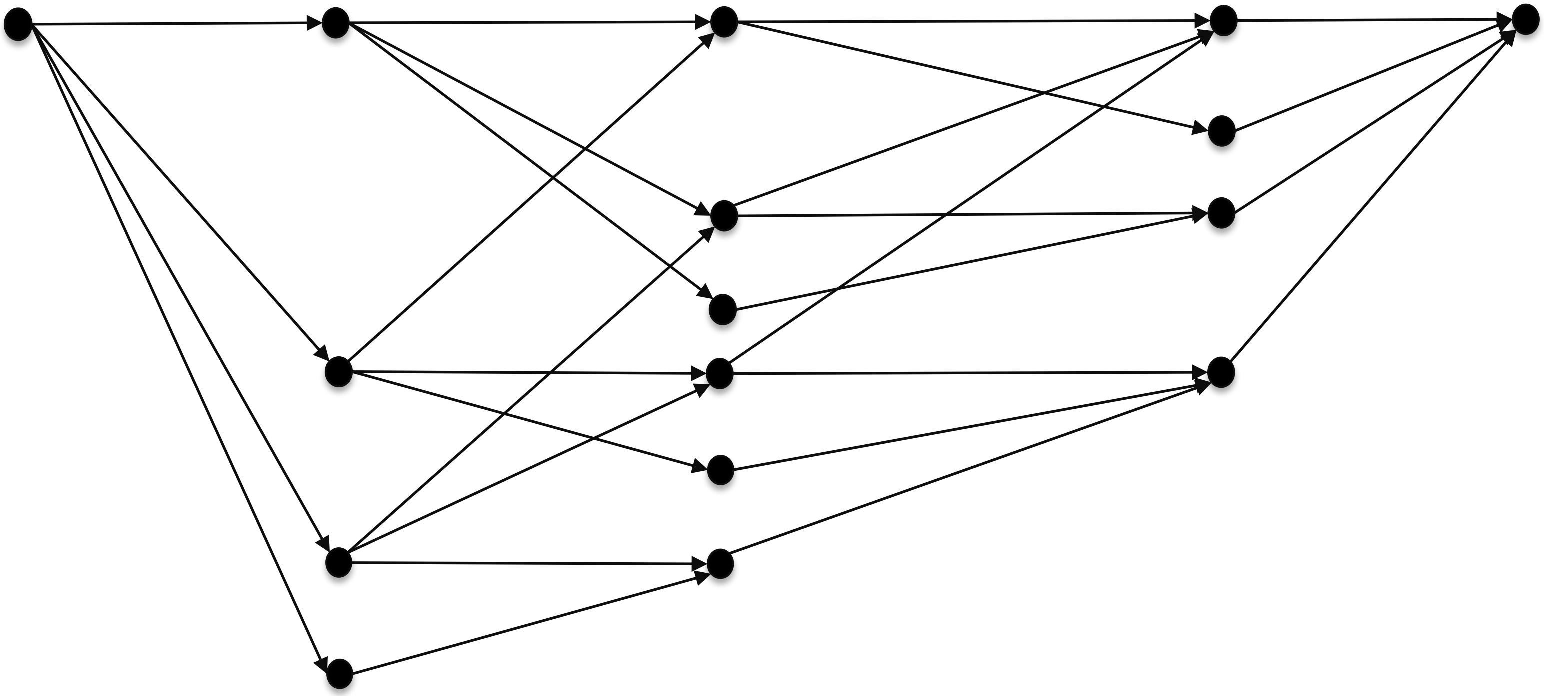
Naively enumerating all possible combinations of release times and execution times (a.k.a. execution scenarios) **is not practical**

[Audsley'93] Neil Audsley, Alan Burns, Mike Richardson, Ken Tindell, and Andy J. Wellings. Applying new scheduling theory to static priority preemptive scheduling. Software Engineering Journal, 1993.

What is a schedule-abstraction graph?

“**schedule-abstraction graph**” [RTSS’17] is a technique that allows us to **aggregate “similar”** schedules while searching for all possible schedules

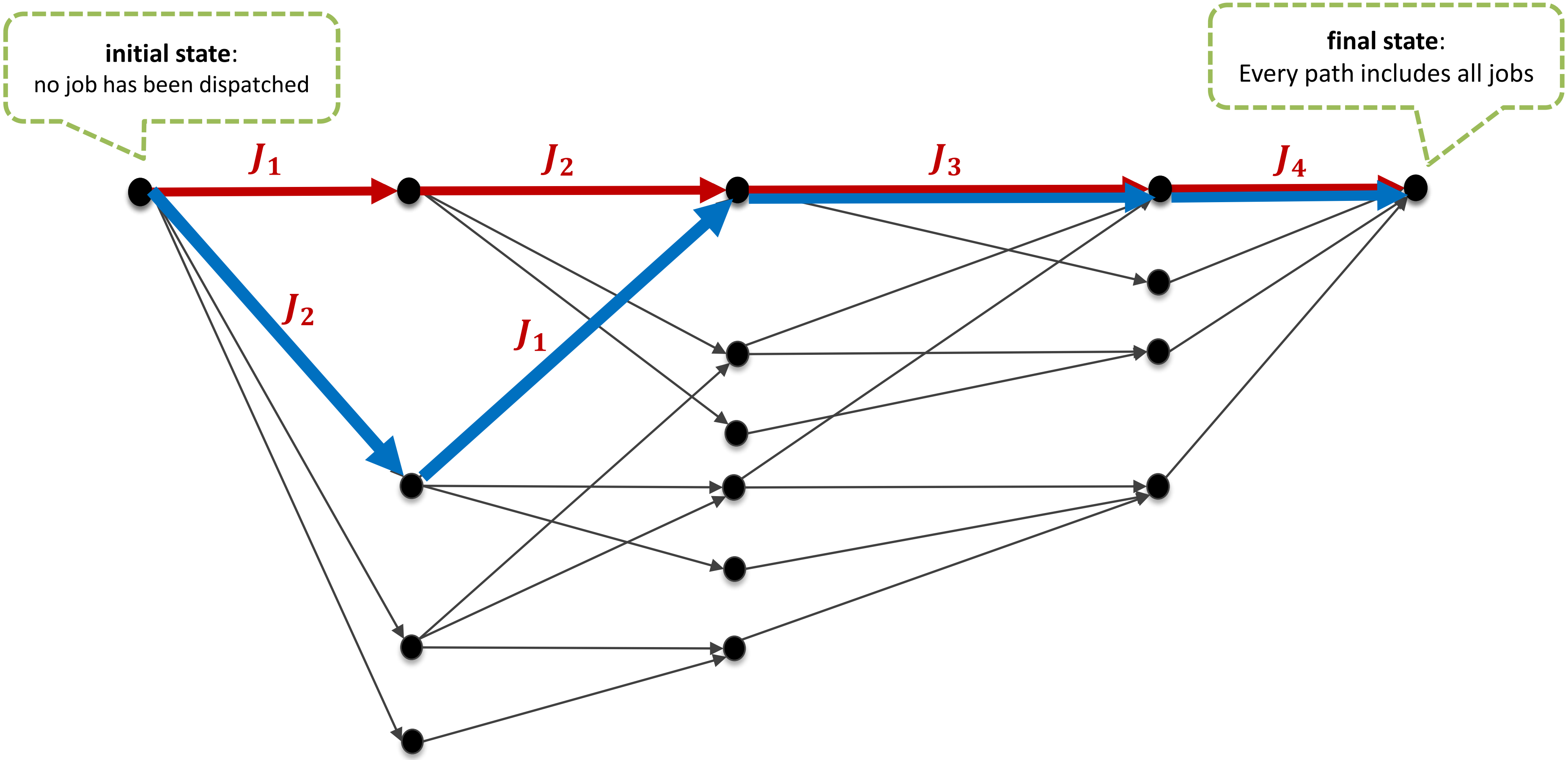
Hence, it reduces the search space



[RTSS’17] Mitra Nasri and Björn B. Brandenburg, "An Exact and Sustainable Analysis of Non-Preemptive Scheduling”, RTSS, 2017, pp. 1-12.

What is a schedule-abstraction graph?

A path in the graph represents an ordered set of dispatched jobs

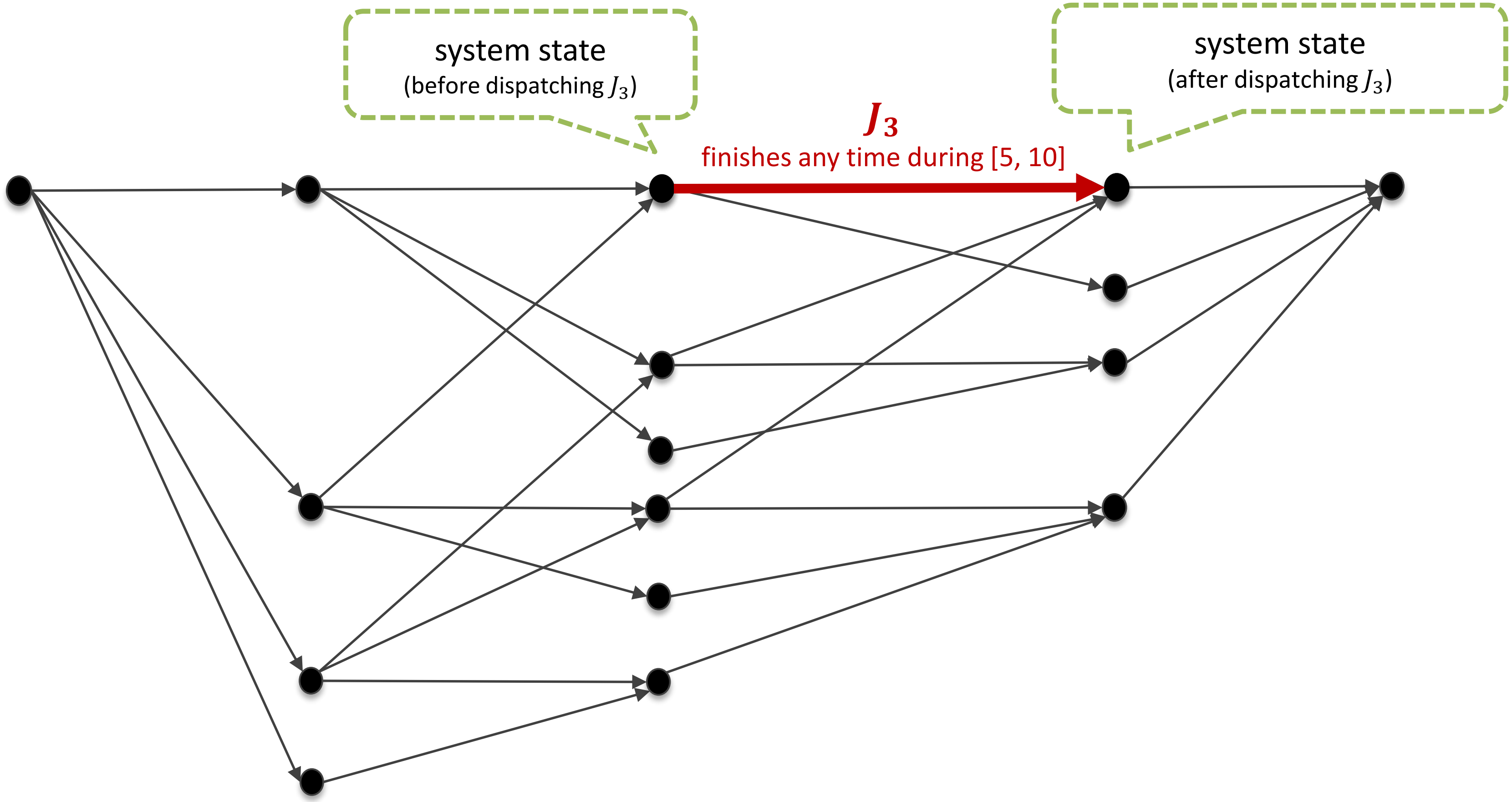


[RTSS’17] Mitra Nasri and Björn B. Brandenburg, "An Exact and Sustainable Analysis of Non-Preemptive Scheduling”, RTSS, 2017, pp. 1-12.

What is a schedule-abstraction graph?

A path in the graph represents an ordered set of dispatched jobs

A vertex abstracts a system state
An edge abstracts a dispatched job



system state
(before dispatching J_3)

system state
(after dispatching J_3)

J_3
finishes any time during [5, 10]

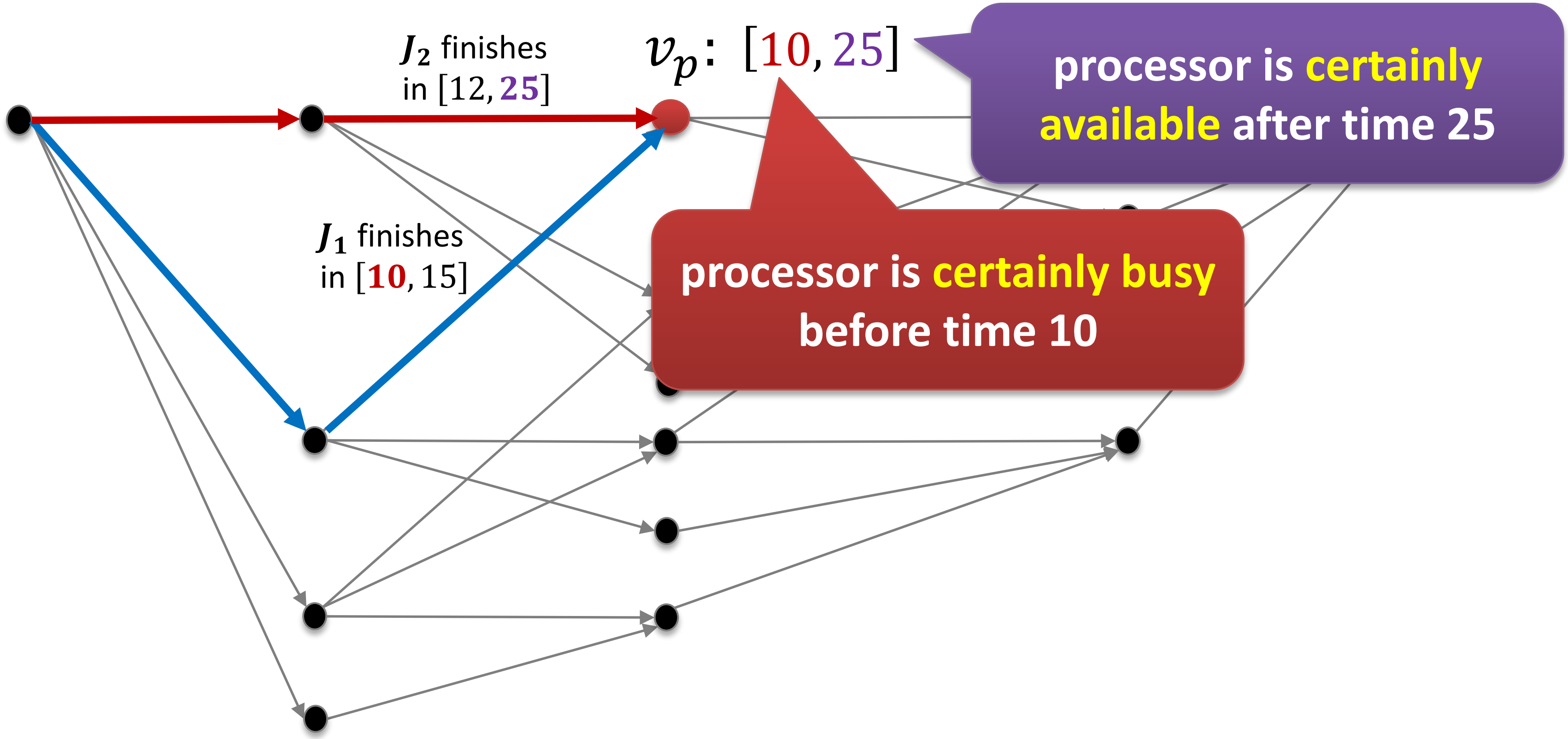
[RTSS'17] Mitra Nasri and Björn B. Brandenburg, "An Exact and Sustainable Analysis of Non-Preemptive Scheduling", RTSS, 2017, pp. 1-12.

What is a schedule-abstraction graph?

A path in the graph represents an ordered set of dispatched jobs

A vertex abstracts a system state
An edge abstracts a dispatched job

A state represents the finish-time interval of any path reaching that state



processor is certainly available after time 25

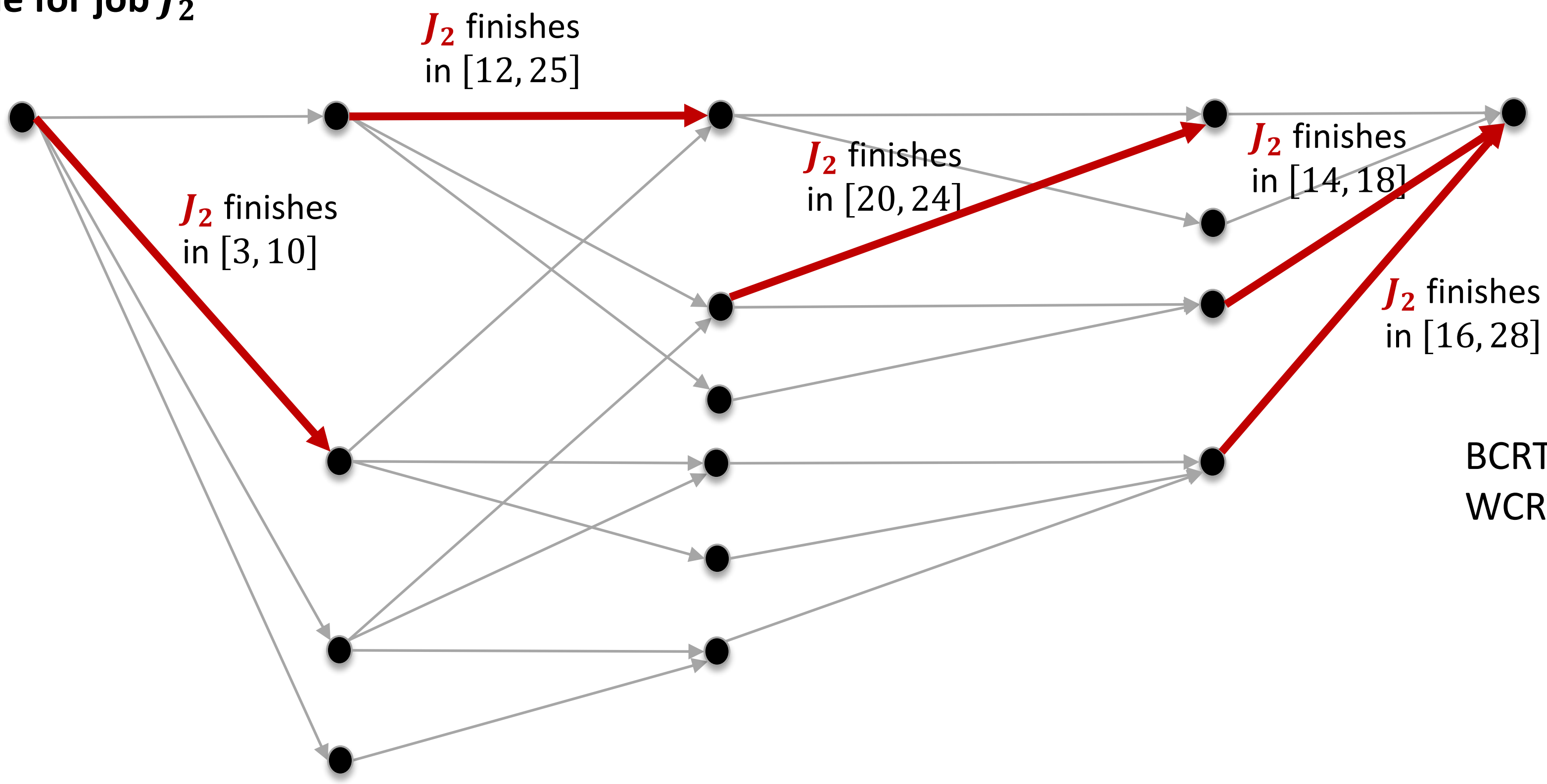
processor is certainly busy before time 10

[RTSS’17] Mitra Nasri and Björn B. Brandenburg, "An Exact and Sustainable Analysis of Non-Preemptive Scheduling”, RTSS, 2017, pp. 1-12.

How to use a schedule-abstraction graph?

The worst-case (best-case) response time of a job J_i is its largest (smallest) finish time among all edges whose label is J_i

Example for job J_2



J_2 finishes in $[12, 25]$

J_2 finishes in $[3, 10]$

J_2 finishes in $[20, 24]$

J_2 finishes in $[14, 18]$

J_2 finishes in $[16, 28]$

BCRT = 3
WCRT = 28

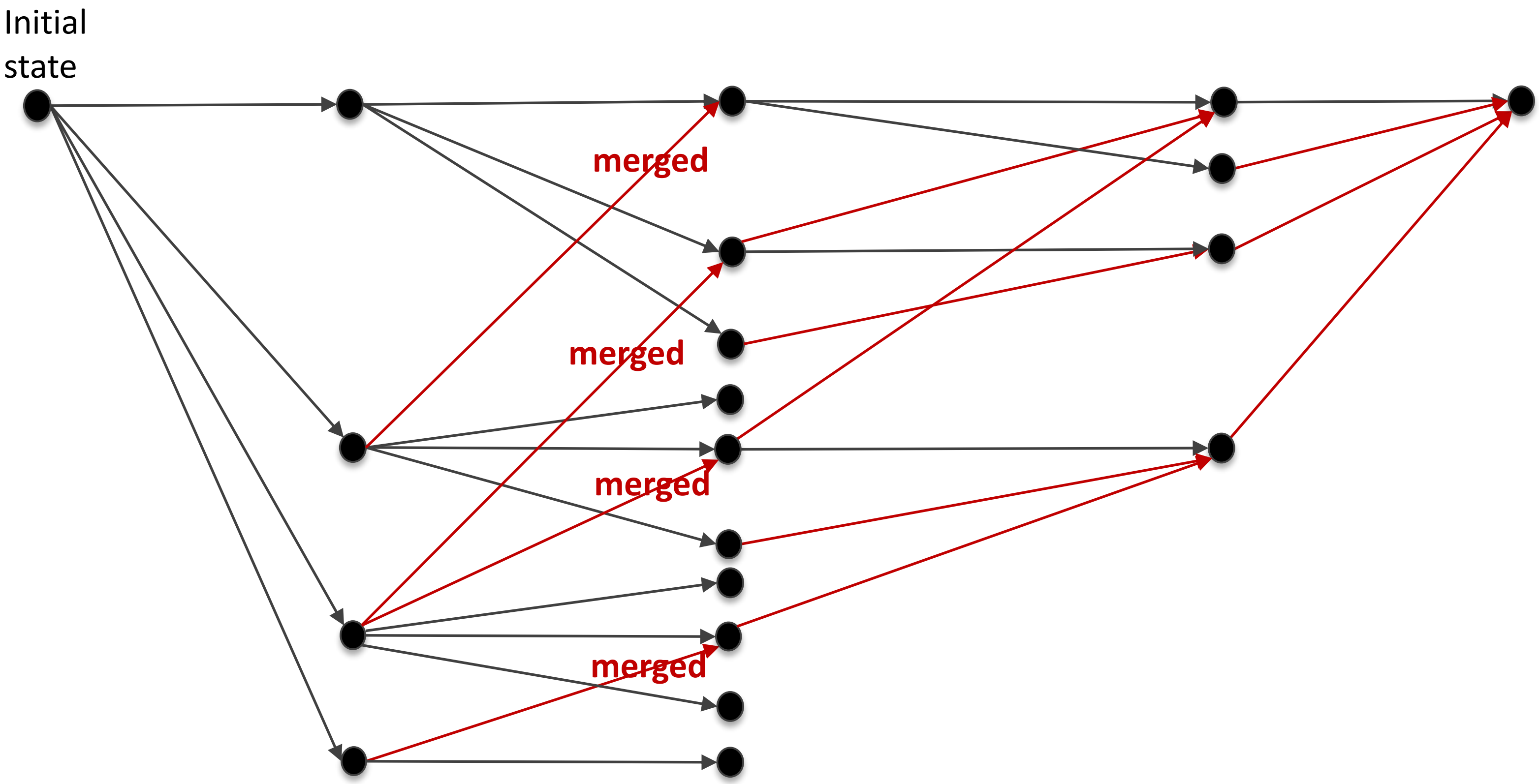
[RTSS’17] Mitra Nasri and Björn B. Brandenburg, "An Exact and Sustainable Analysis of Non-Preemptive Scheduling”, RTSS, 2017, pp. 1-12.

How to build a schedule-abstraction graphs?

[RTSS'17] used a breadth-first strategy

Repeat until every path includes all jobs

1. Find the shortest path
2. For each not-dispatched job that can be dispatched after the path:
   - 2.1. Expand (add a new vertex)
   - 2.2. Merge (if possible, merge the new vertex with an existing vertex)



[RTSS'17] Mitra Nasri and Björn B. Brandenburg, "An Exact and Sustainable Analysis of Non-Preemptive Scheduling", RTSS, 2017, pp. 1-12.

Schedule-Abstraction Graphs

for Global Scheduling Policies

(this work)

Overview of the solution

Goal: **define and build** a schedule-abstraction graph for global scheduling policies

SYSTEM ABSTRACTION
(What is the system state?
What is on the edges?)

EXPANSION RULES
(How to select jobs that can be dispatched
"next" by the scheduling policy at any state?)

MERGING RULES
(When and how to merge
two states?)

In the talk

In the paper

Our prior work in [RTSS'17] **was for uniprocessor system**

Its state definition and expansion and merging rules
are not applicable to multiprocessor scheduling

[RTSS'17] Mitra Nasri and Björn B. Brandenburg, "An Exact and Sustainable Analysis of Non-Preemptive Scheduling", RTSS, 2017, pp. 1-12.

Definition of state

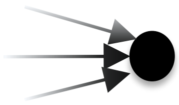
The earliest finish time of the job running on this core

The latest finish time of the job running on this core

$$v_i = \begin{cases} \varphi_1 : [EFT_1, LFT_1] \\ \varphi_2 : [EFT_2, LFT_2] \\ \dots \\ \varphi_m : [EFT_m, LFT_m] \end{cases}$$

One interval for each of the m cores

Example:

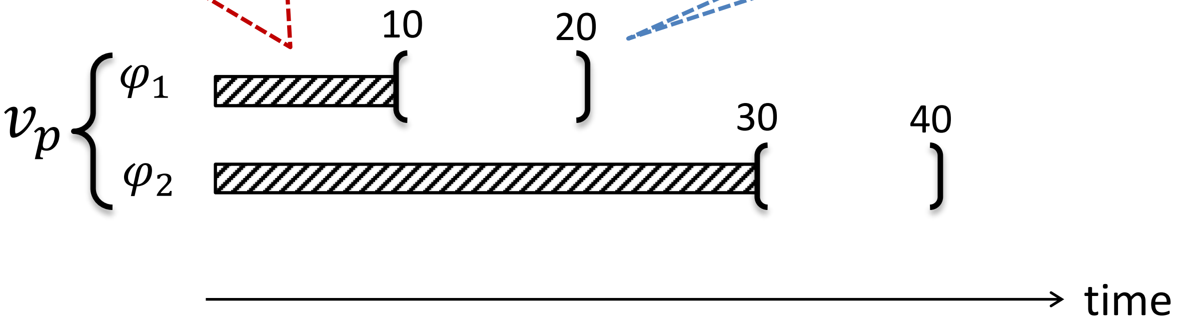


$$v_p : \begin{cases} \varphi_1 : [10, 20] \\ \varphi_2 : [30, 40] \end{cases}$$

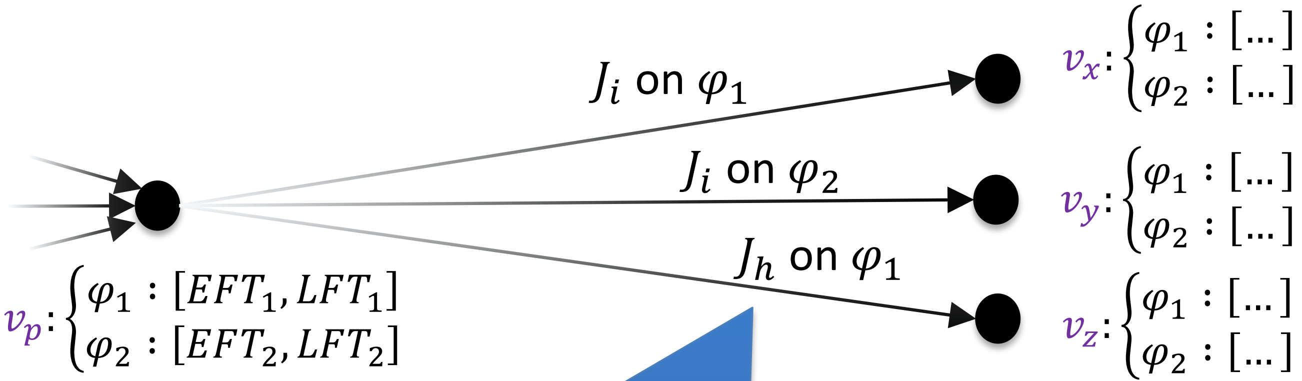
Core φ_1 is possibly available from time 10

Core φ_1 is certainly available from time 20

Core φ_1 is certainly not available before time 10



Definition of expansion rules (for global multiprocessor scheduling)


J_i on φ_1
J_i on φ_2
J_h on φ_1
v_p: $\begin{cases} \varphi_1 : [EFT_1, LFT_1] \\ \varphi_2 : [EFT_2, LFT_2] \end{cases}$
v_x: $\begin{cases} \varphi_1 : [\dots] \\ \varphi_2 : [\dots] \end{cases}$
v_y: $\begin{cases} \varphi_1 : [\dots] \\ \varphi_2 : [\dots] \end{cases}$
v_z: $\begin{cases} \varphi_1 : [\dots] \\ \varphi_2 : [\dots] \end{cases}$

(eligible jobs)
Which jobs **may possibly** be dispatched “next” on each of the cores?

What is the **new state**?

Rule 1: work-conserving scheduler

If at time t there is a certainly released job and a certainly available core, a job will be dispatched at time t.

Rule 2: job-level fixed-priority scheduler

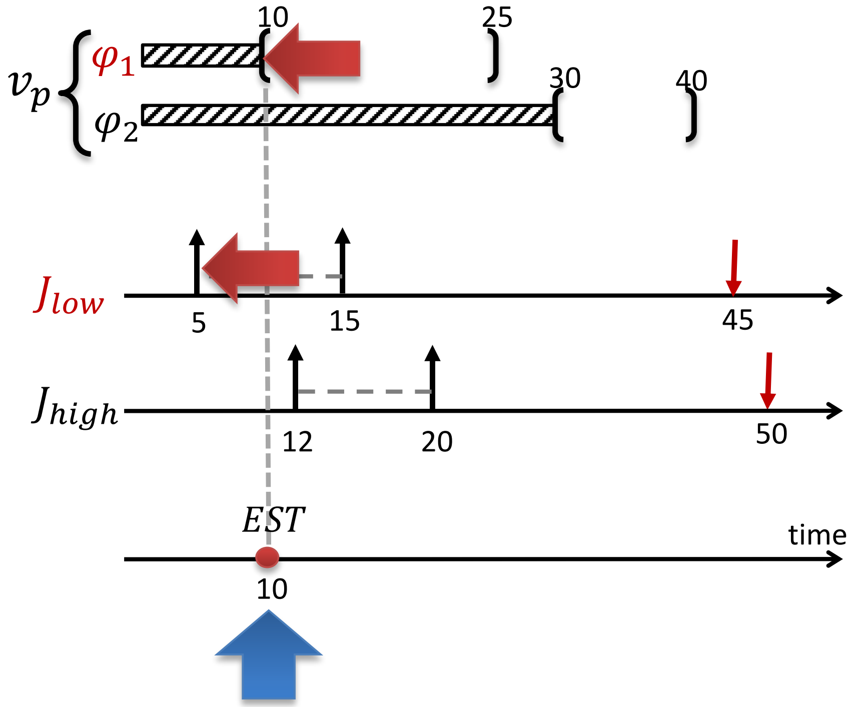
A lower priority job cannot be dispatched as soon as a higher-priority job is certainly released and not yet scheduled.

Finding “eligible” jobs

For each not-scheduled job J_i on each core φ_k

1. Find the **earliest start time** (EST) of J_i on φ_k
2. Find the **latest start time** (LST) of J_i on any core for a **work-conserving** and **JLFP policy**
3. If **EST $\leq$ LST** then add an edge for job J_i dispatched on core φ_k

Example: is J_{low} eligible on each core φ_1?



earliest start time

Job	Release time Min	 Max	Deadline	Execution time Min	 Max	Priority
J_{low}	5	15	50	2	15	low
J_{high}	12	20	45	1	10	high

Finding “eligible” jobs

For each not-scheduled job J_i on each core φ_k

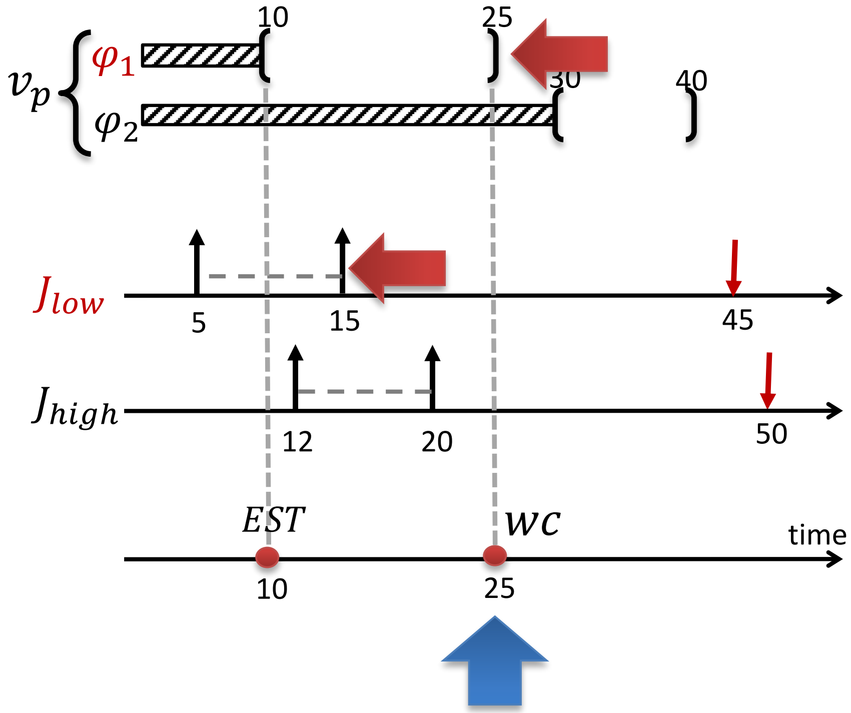
1. Find the **earliest start time** (EST) of J_i on φ_k

2. Find the **latest start time** (LST) of J_i on any core for a **work-conserving** and **JLFP policy**

3. If **EST $\leq$ LST** then add an edge for job J_i dispatched on core φ_k

Merging rules and other details in the paper…

Example: is J_{low} eligible on each core φ_1?



work-conserving policy

Job	Release time		Deadline	Execution time		Priority
	Min	Max		Min	Max	
J_{low}	5	15	50	2	15	low
J_{high}	12	20	45	1	10	high

Empirical Evaluation

Main questions

How much the proposed analysis improves schedulability over the state of the art?

Which state of the art?

- For most cases that we cover, there is no prior test.
- So we compare against sporadic tests

Does the proposed analysis scale (in terms of runtime) **to practical workload sizes?**

Evaluation setup

Baseline tests (designed for sporadic tasks)

Baruah-EDF [Baruah'06]	for Global-EDF
Guan-Test1-WC [Guan'11]	for general work-conserving scheduling policies
Guan-Test2-FP [Guan'11]	for Global-FP
Lee-FP [Lee'17]	for Global-FP

We used rate-monotonic priorities for all fixed-priority policies

Periodic task set generation

- Periods randomly chosen from [10000, 100000]μs with log-uniform distribution
- Utilizations are obtained from RandFixSum
- Release jitter options: {no jitter, small jitter of 100μs}
- BCET = 0.1 · WCET
- A task set with more than 100000 jobs per hyperperiod is discarded

Experiment platform

- Intel Xeon E7-8857 v2 processor
- 3 GHz clock speed and 1.5 TiB RAM

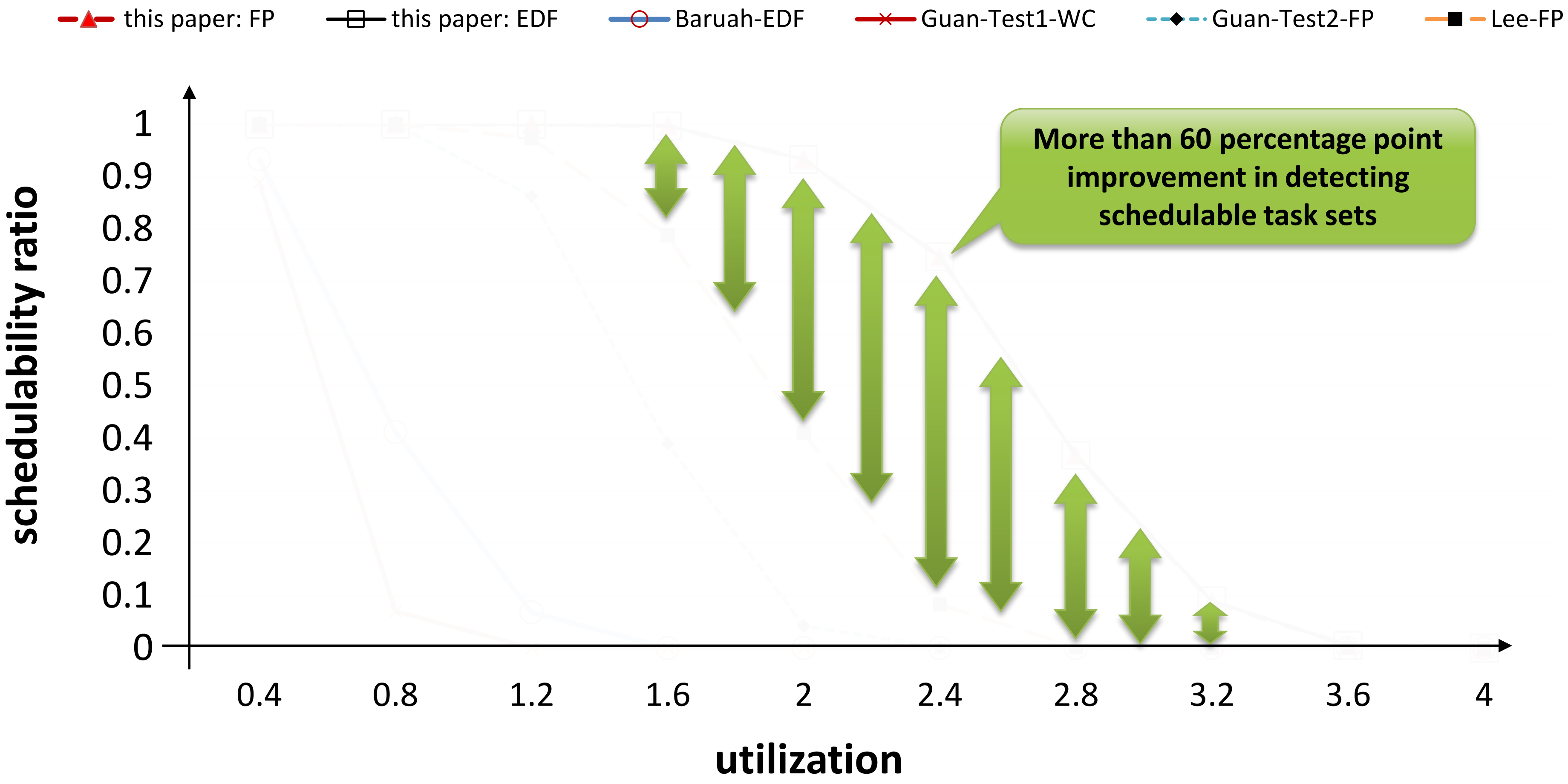
[Baruah'06] Sanjoy Baruah, Samarjit Chakraborty. Schedulability analysis of non-preemptive recurring real-time tasks, IPDPS, 2006.

[Guan'11] Nan Guan, Wang Yi, Qingxu Deng, Zonghua Gu, and Ge Yu. Schedulability analysis for non-preemptive fixed-priority multiprocessor scheduling, JSA, 2011.

[Lee'17] Jinkyu Lee. Improved schedulability analysis using carry-in limitation for non-preemptive fixed-priority multiprocessor scheduling, TC, 2017.

Schedulability improvements

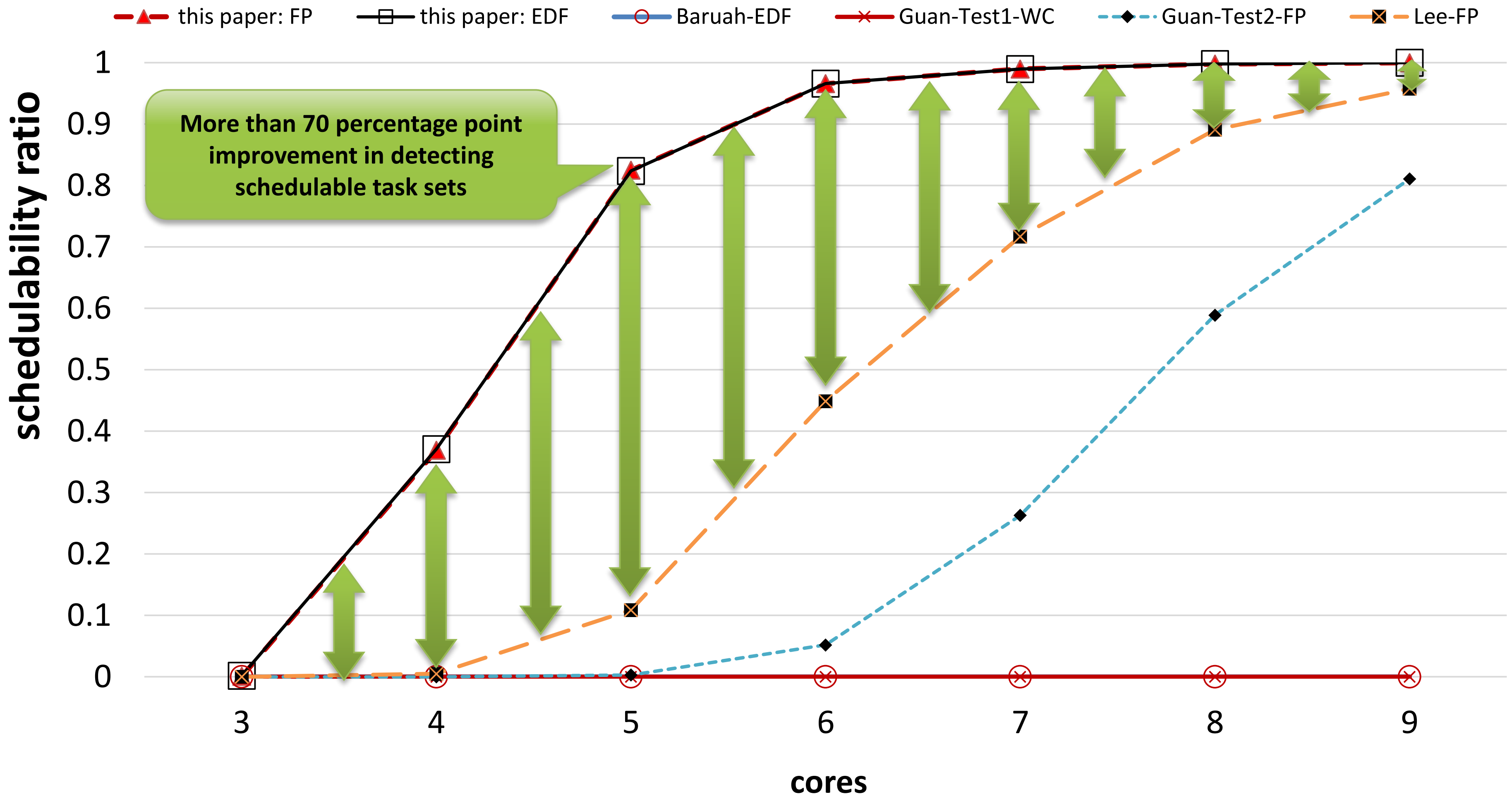
10 tasks, 4 cores, varying utilization



More than 60 percentage point improvement in detecting schedulable task sets

Schedulability improvements

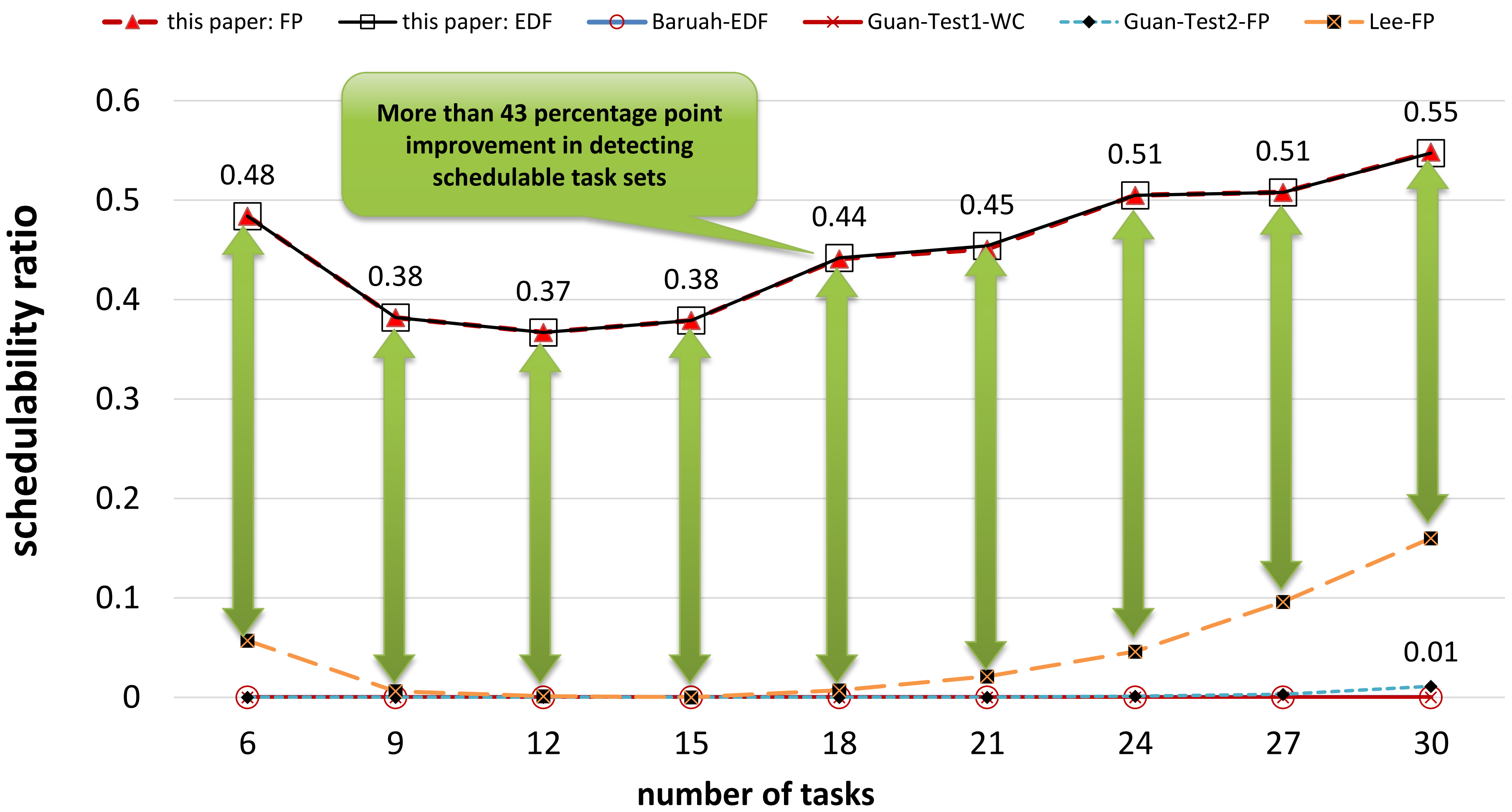
10 tasks, U = 2.8, varying number of cores



More than 70 percentage point improvement in detecting schedulable task sets

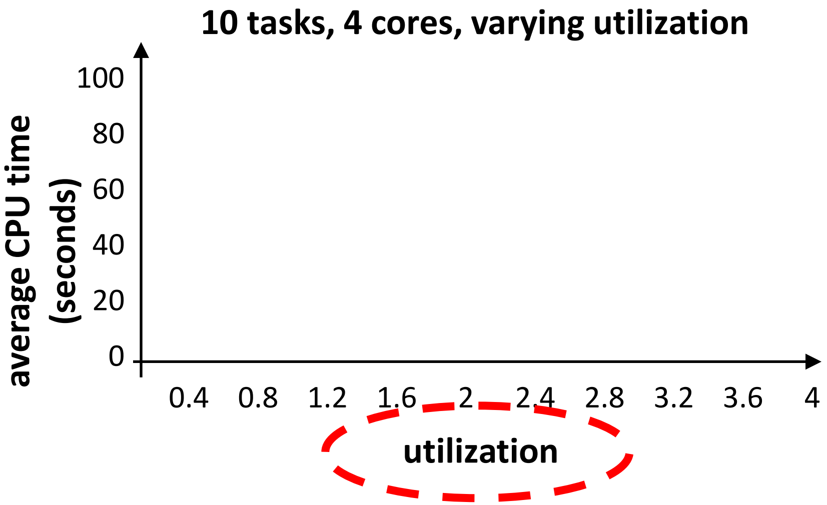
Schedulability improvements

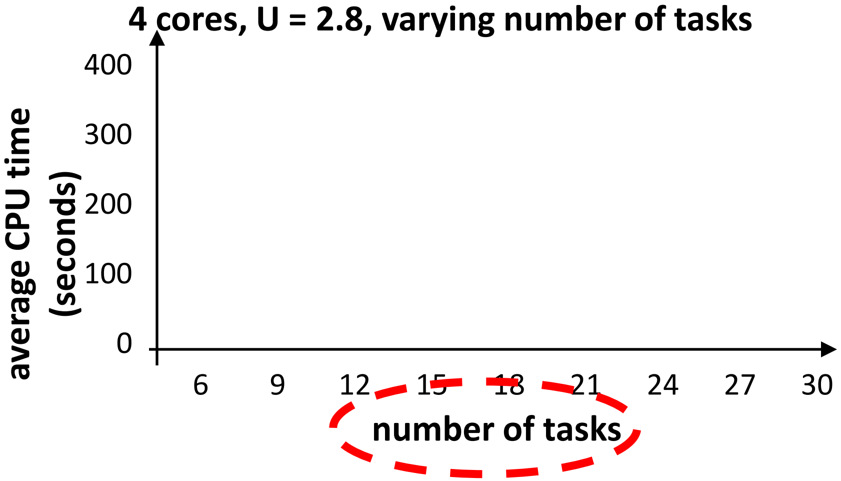
4 cores, U = 2.8, varying number of tasks

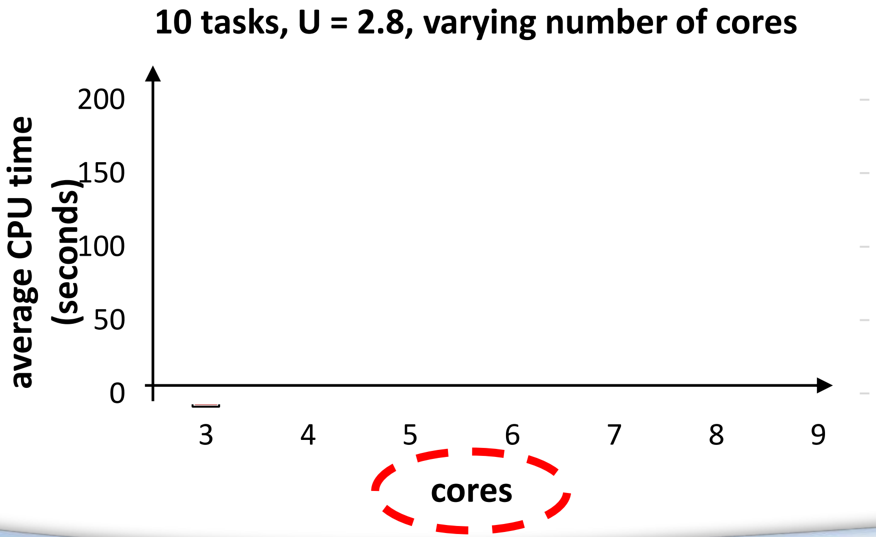


More than 43 percentage point improvement in detecting schedulable task sets

Runtime of the analysis

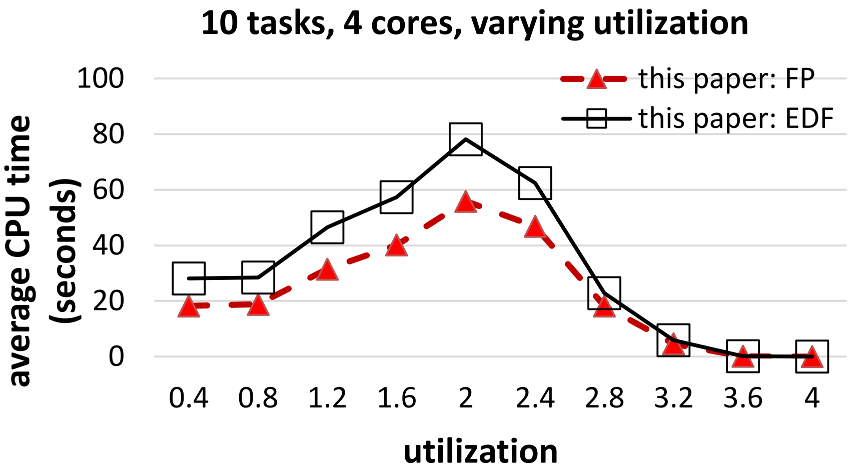


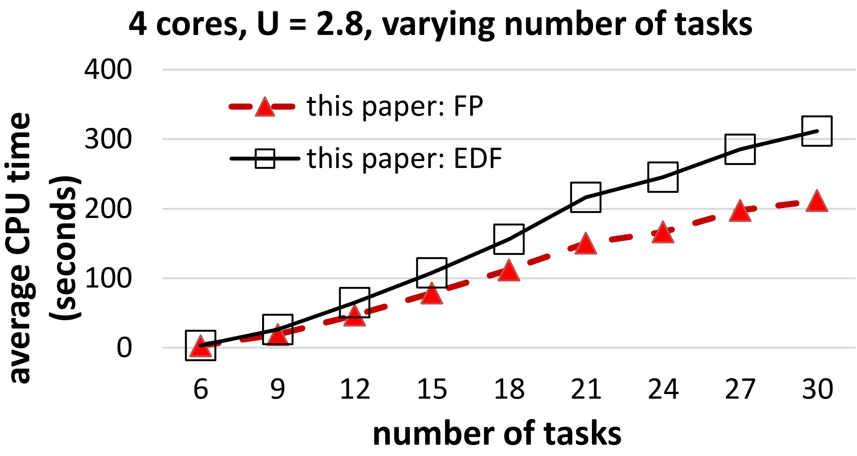


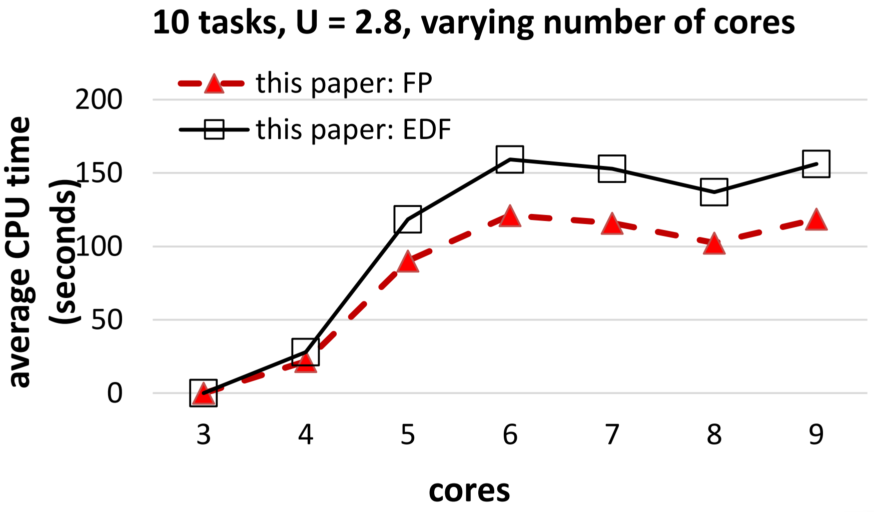


- Experiment performed on Intel Xeon E7-8857 v2 processor 3 GHz clock speed and 1.5 TiB RAM
- A single-threaded implementation

Runtime of the analysis







The analysis has acceptable runtime for small- and medium-sized workloads

Conclusions and future directions

Summary

Goal

A response time analysis for non-preemptive job sets scheduled by global JLFP policies

Solution

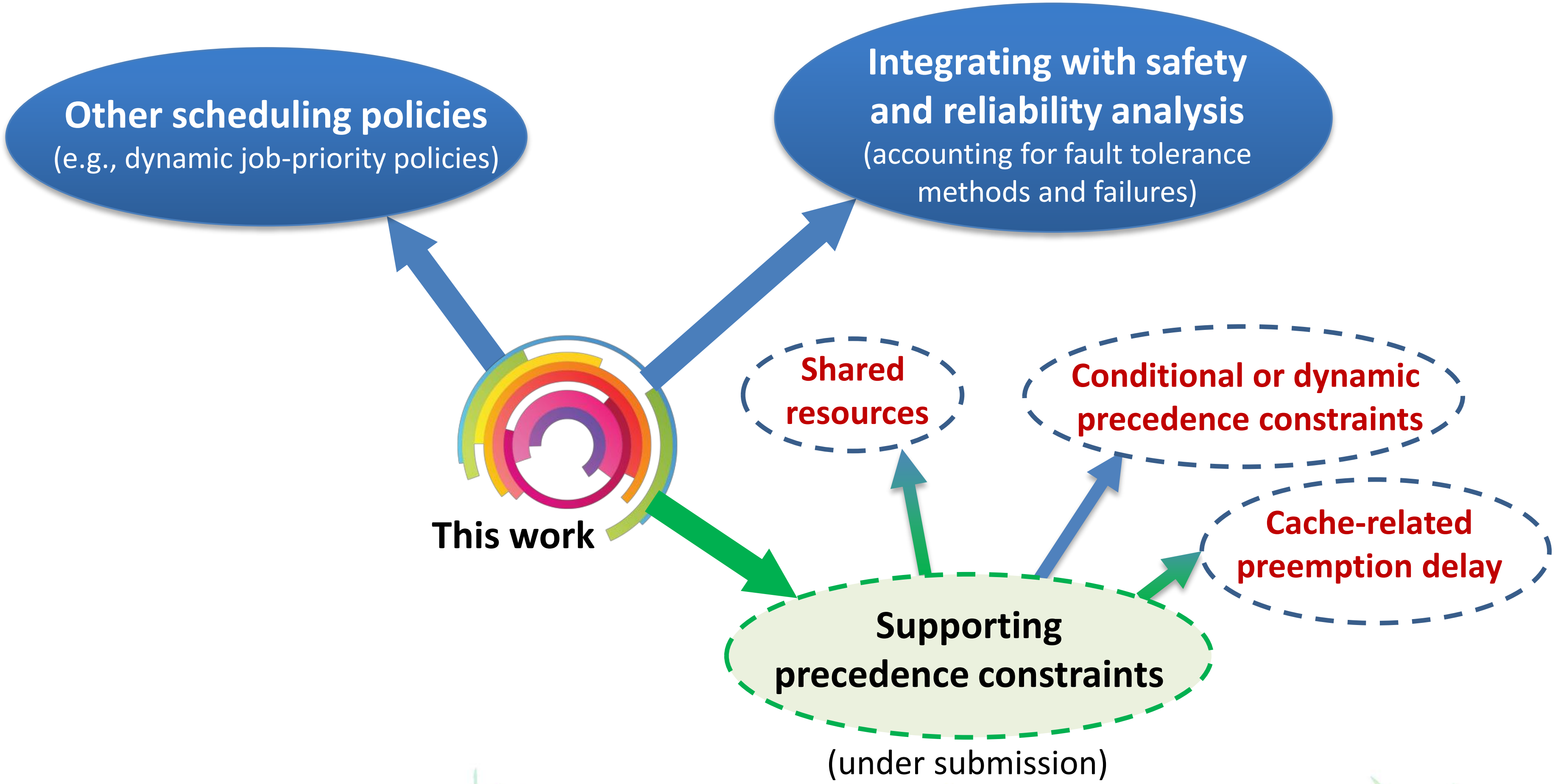
We introduced a schedule-abstraction graph for global multiprocessor scheduling

What did we get?

Up to 70 percentage point improvement in schedulability ratio

(w.r.t. the baseline analyses for sporadic tasks)

Road map and future directions



Other scheduling policies
(e.g., dynamic job-priority policies)

Integrating with safety and reliability analysis
(accounting for fault tolerance methods and failures)

This work

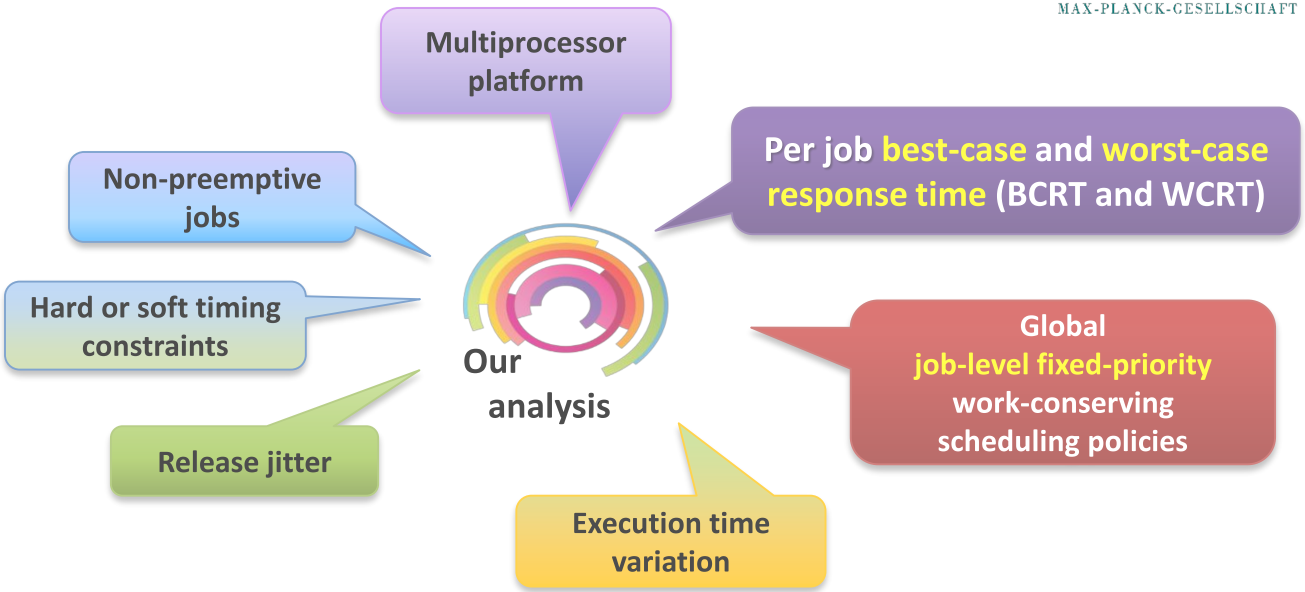
Shared resources

Conditional or dynamic precedence constraints

Cache-related preemption delay

Supporting precedence constraints

(under submission)



Our analysis

Multiprocessor platform

Per job best-case and worst-case response time (BCRT and WCRT)

Non-preemptive jobs

Hard or soft timing constraints

Release jitter

Global job-level fixed-priority work-conserving scheduling policies

Execution time variation

- **Mitra Nasri**
- Geoffrey Nelissen
- Björn B. Brandenburg

Thank you