Response-Time Analysis of Limited-Preemptive Parallel DAG Tasks Under Global Scheduling


Mitra Nasri


Geoffrey Nelissen


Björn Brandenburg

TUDelft

CISTER
Research Centre in
Real-Time & Embedded
Computing Systems

MAX PLANCK INSTITUTE
FOR SOFTWARE SYSTEMS

Our work in a nutshell

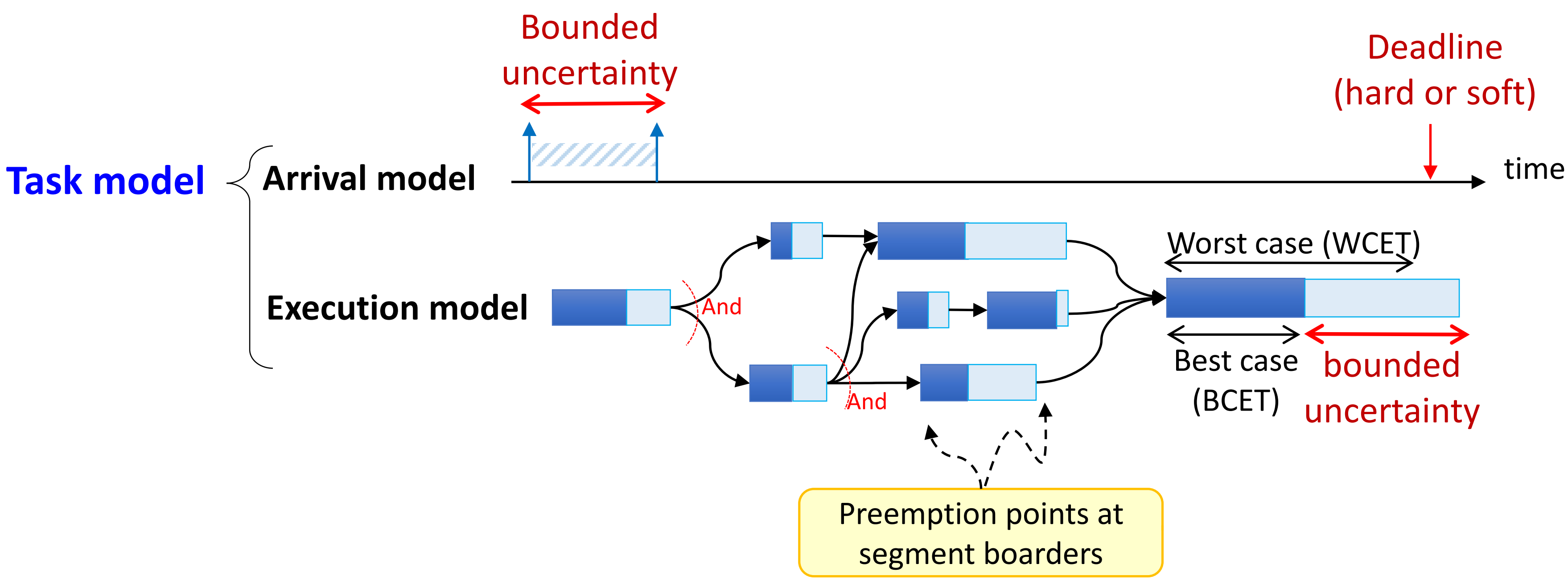
We obtain the worst-case and best-case response time

Workload model: **Parallel DAG tasks (or job sets)**

Platform model: **Multicore (identical cores)**

Scheduler model: **Global job-level fixed-priority (JLFP)**

Execution model: **Limited preemptive (fixed-preemption points)**



Our work in a nutshell

We obtain the worst-case and best-case response time

Workload model

Parallel DAG tasks (or job sets)

Platform model

Multicore (identical cores)

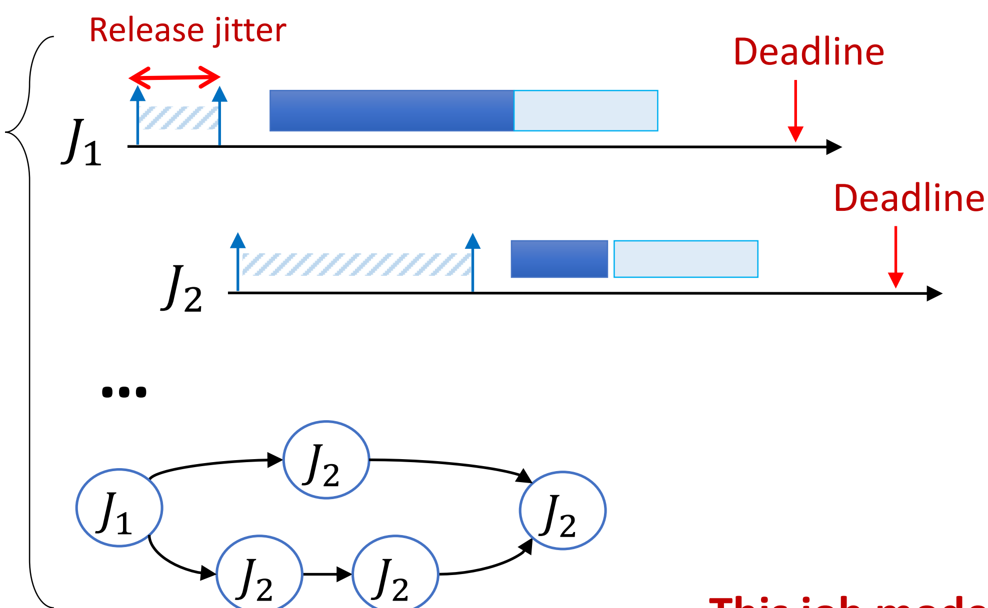
Scheduler model

Global job-level fixed-priority (JLFP)

Execution model

Limited preemptive (fixed-preemption points)

Job model



Examples:

- Transactions
- Multi-frame tasks
- Periodic DAG tasks
- ...

This job model supports bounded non-deterministic arrivals, but not sporadic tasks (un-bounded non-deterministic arrivals)

State of the art

Closed-form analyses
(e.g., problem-window analysis)

- Fast

- Pessimistic
- Hard to extend

$$R_i^{(0)} = C_i + \sum_{j=1}^{i-1} C_j$$

$$R_i^{(k)} = C_i + \sum_{j=1}^{i-1} \left\lceil \frac{R_i^{(k-1)}}{T_j} \right\rceil C_j$$

ons of lower-priority tasks. A response-time AG-based task-set with a limited-preemptive rity scheduler is computed by iterating the following equation until a fixed point is reached, starting with $R_k = len(G_k) + \frac{1}{m}(vol(G_k) - len(G_k))$:

$$R_k \leftarrow len(G_k) + \frac{1}{m}(vol(G_k) - len(G_k) + I_k^{hp} + I_k^{lp}) \quad (1)$$

State of the art

Closed-form analyses (e.g., problem-window analysis)

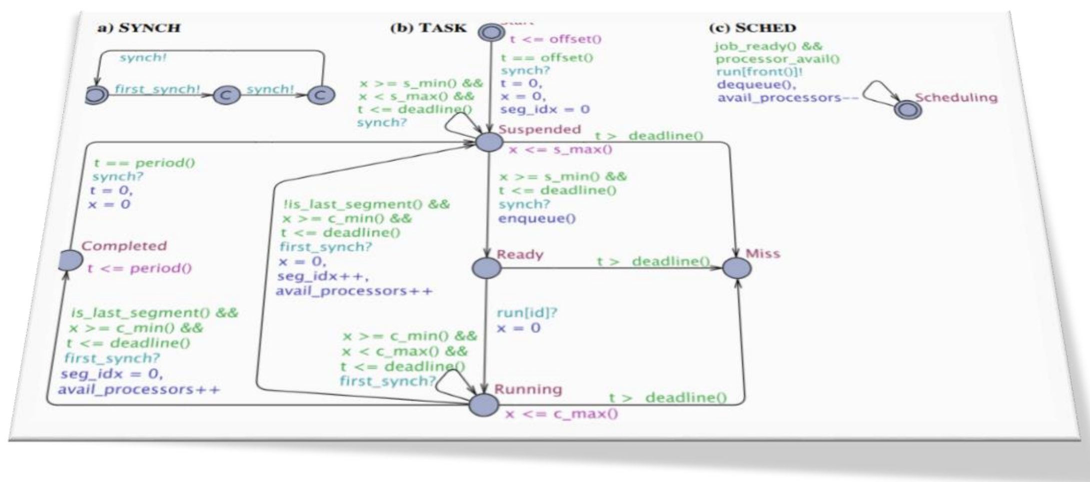
- Fast

- Pessimistic
- Hard to extend

Exact tests in generic formal verification tools (e.g., UPPAAL)

- Accurate
- Easy to extend

- Not scalable



State of the art

Closed-form analyses
(e.g., problem-window analysis)

- **Fast**

- **Pessimistic**
- **Hard to extend**

Exact tests in generic formal verification tools (e.g., UPPAAL)

- **Accurate**
- **Easy to extend**

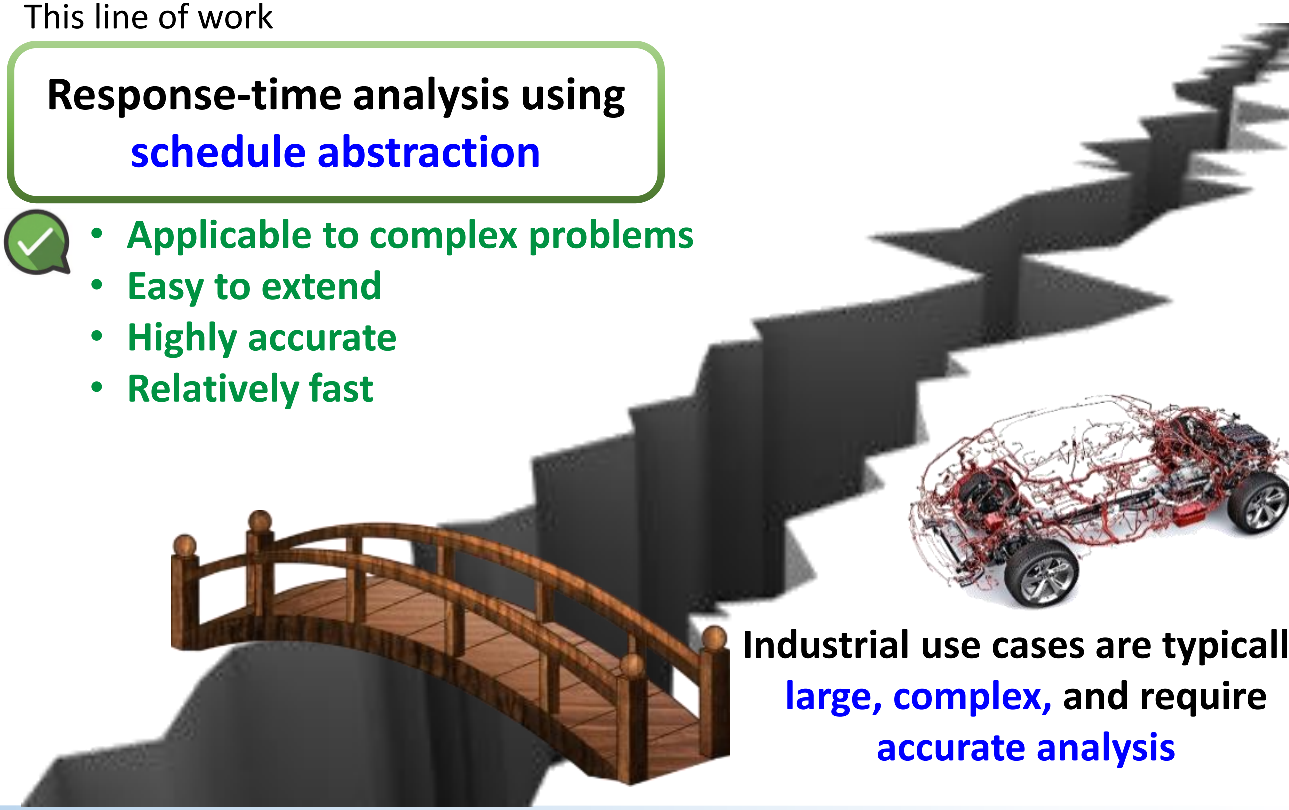
- **Not scalable**

Idea: explore all possible schedules

This line of work

Response-time analysis using schedule abstraction

- **Applicable to complex problems**
- **Easy to extend**
- **Highly accurate**
- **Relatively fast**



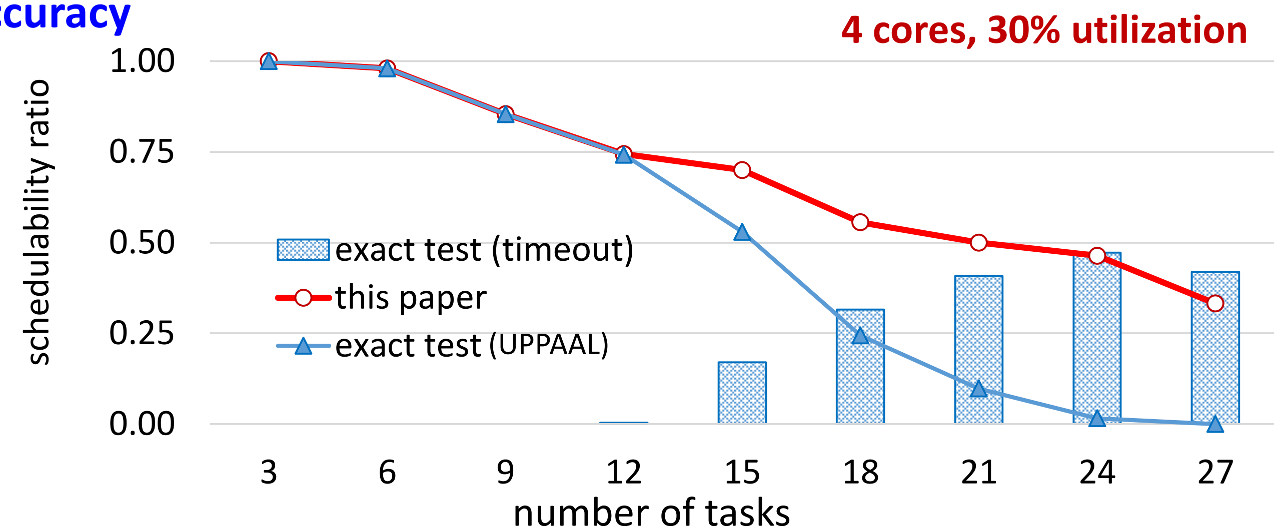
Industrial use cases are typically large, complex, and require accurate analysis

State of the art: comparison

Sequential periodic tasks
(global FP scheduling)

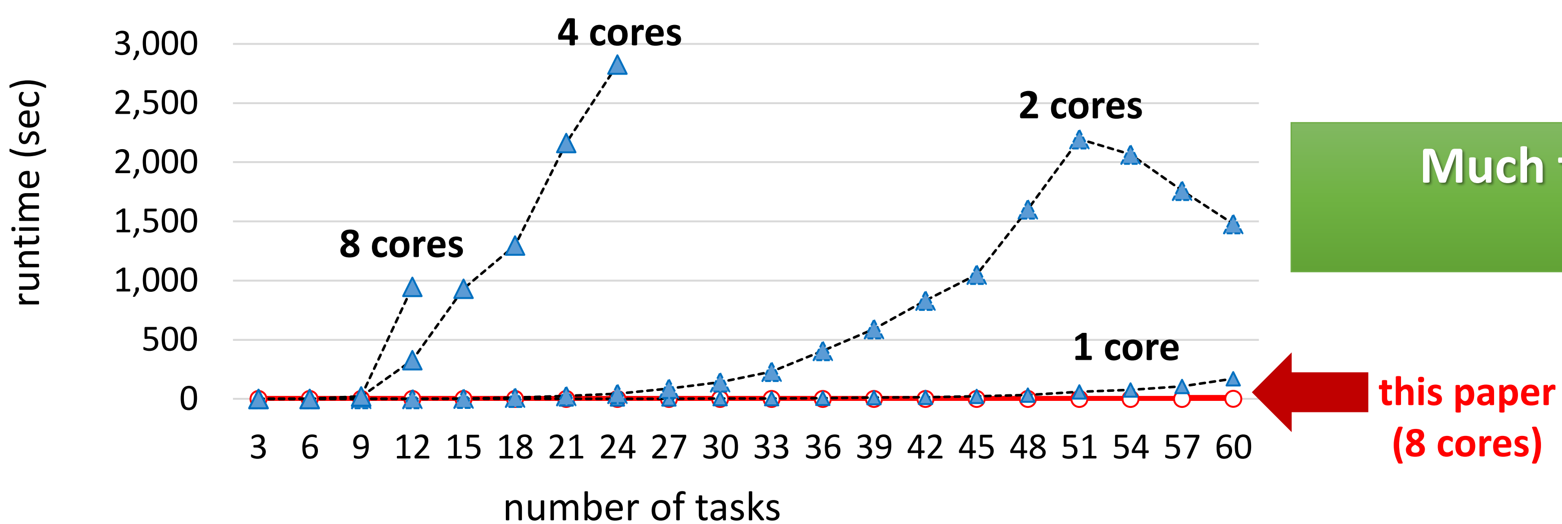
Experiment on sequential periodic tasks

Accuracy



Almost as accurate as the exact test

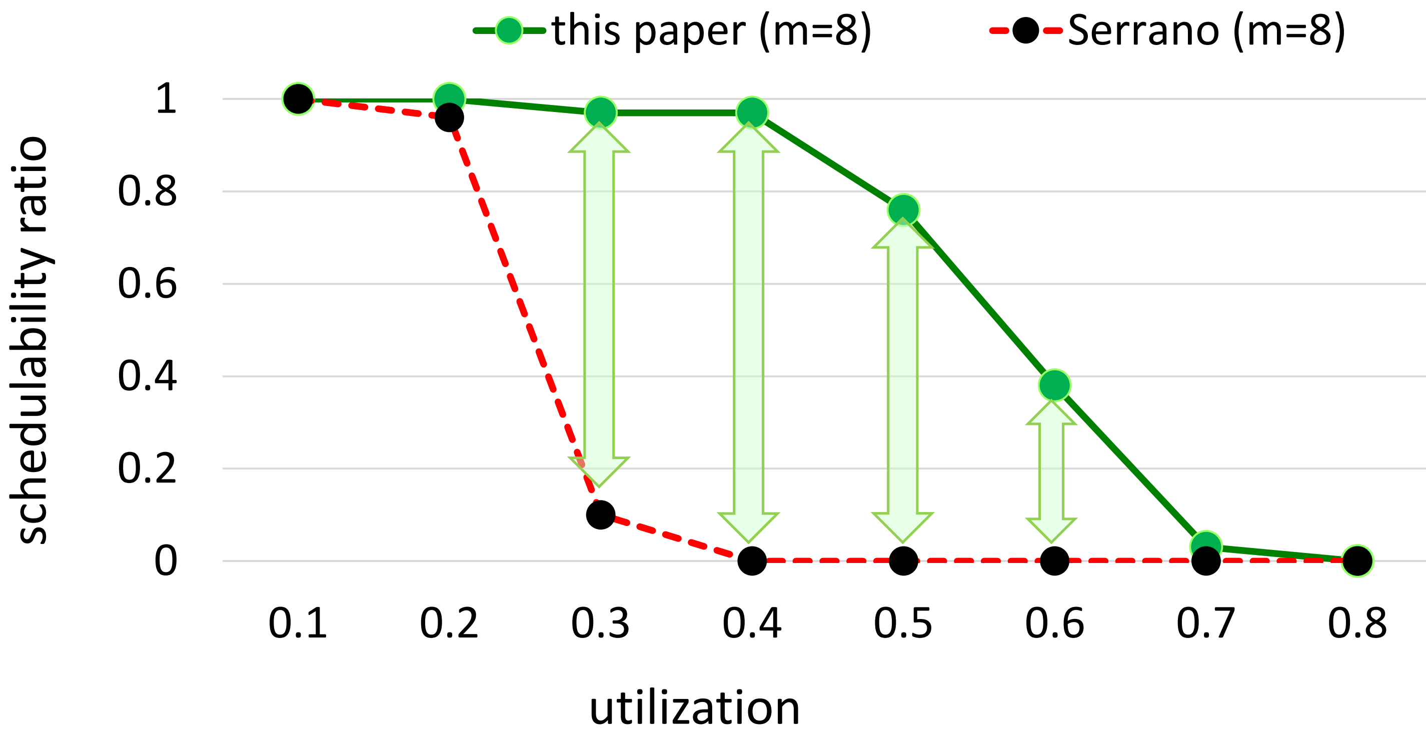
Runtime



Much faster

State of the art: comparison

Effectiveness (for parallel DAG tasks)



Much less pessimistic than the closed-form analysis

State of the art: schedule-abstraction-based analyses

[RTSS’17]

Uniprocessor	Independent non-preemptive jobs/tasks	Work-conserving and **non-work-conserving** job-level fixed-priority scheduling (JLFP)
Exact		

[ECRTS’18]

Multiprocessor	Independent non-preemptive jobs/tasks	**Global** work-conserving job-level fixed-priority scheduling (JLFP)
Sufficient		

[this work]

Multiprocessor	**Non-preemptive jobs/DAG tasks with precedence constraints**	**Global** work-conserving job-level fixed-priority scheduling (JLFP)
Sufficient		A new system abstraction (more scalable)

[RTSS’17] M. Nasri and B. Brandenburg, "An Exact and Sustainable Analysis of Non-Preemptive Scheduling”.
[ECRTS’18] M. Nasri, G. Nelissen, and B. Brandenburg, “A Response-Time Analysis for Non-Preemptive Job Sets under Global Scheduling”.

Agenda

- Schedule-abstraction-based analysis
- Supporting precedence constraints
  - Challenges
  - A new abstraction
- Evaluation
- Conclusion and future work

Response-time analysis using schedule abstraction

Highlights

A sound analysis must consider
all possible execution scenarios
(i.e., combination of release times and execution times)

Due to scheduling anomalies

Observation

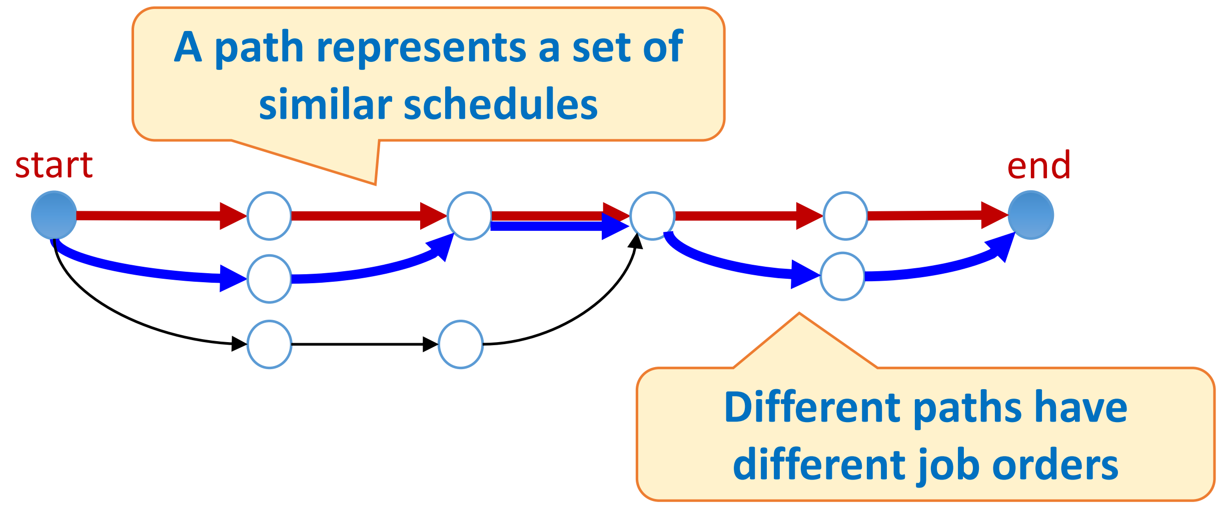
There are fewer permissible
job orderings than schedules

Solution

Use job-ordering abstraction to analyze schedulability
by building a graph that represents all possible schedules

Response-time analysis using schedule-abstraction graphs

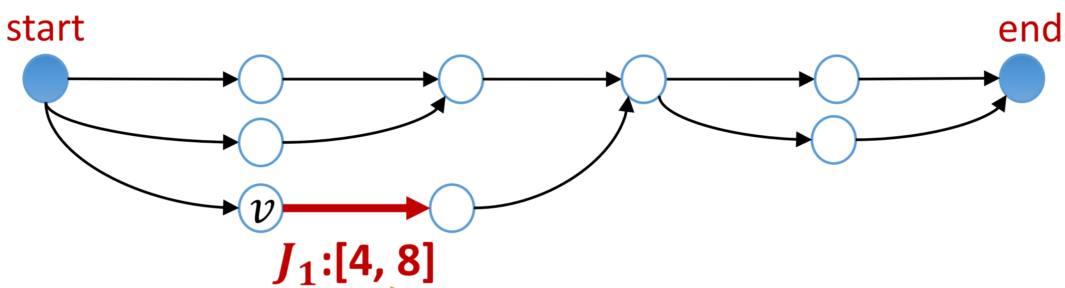
A **path** aggregates all schedules with the same job ordering



Response-time analysis using schedule-abstraction graphs

A **path** aggregates all schedules with the same job ordering

A **vertex** abstracts a system state and an **edge** represents a dispatched job



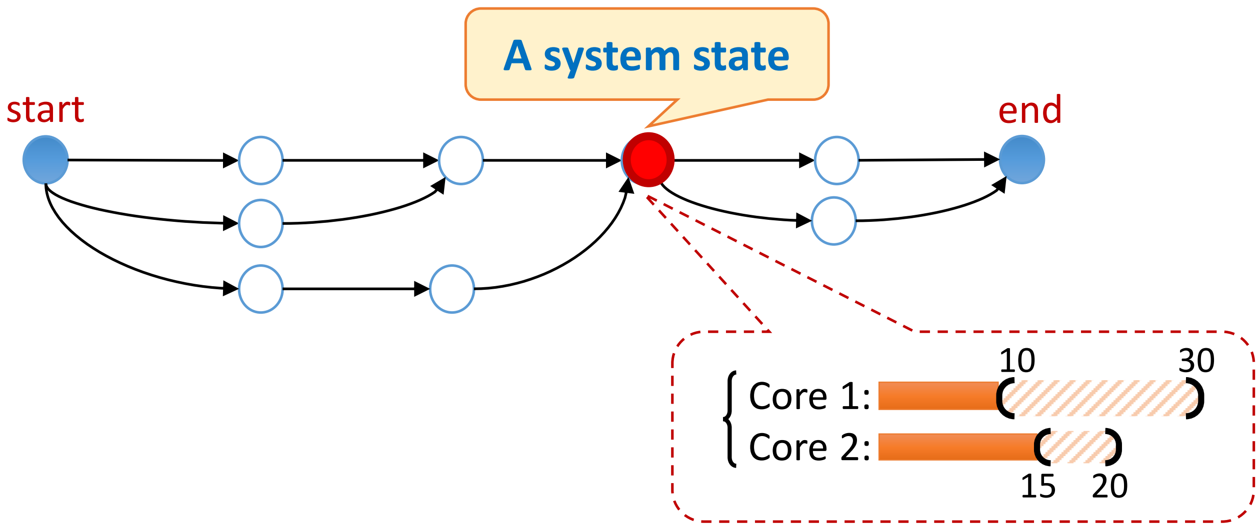
Earliest and latest finish time of J_1 when it is dispatched after state v

Response-time analysis using schedule-abstraction graphs

A **path** aggregates all schedules with the same job ordering

A **vertex** abstracts a system state and an **edge** represents a dispatched job

A **state** is labeled with the **finish-time interval** of any path reaching the state



Interpretation of an uncertainty interval:

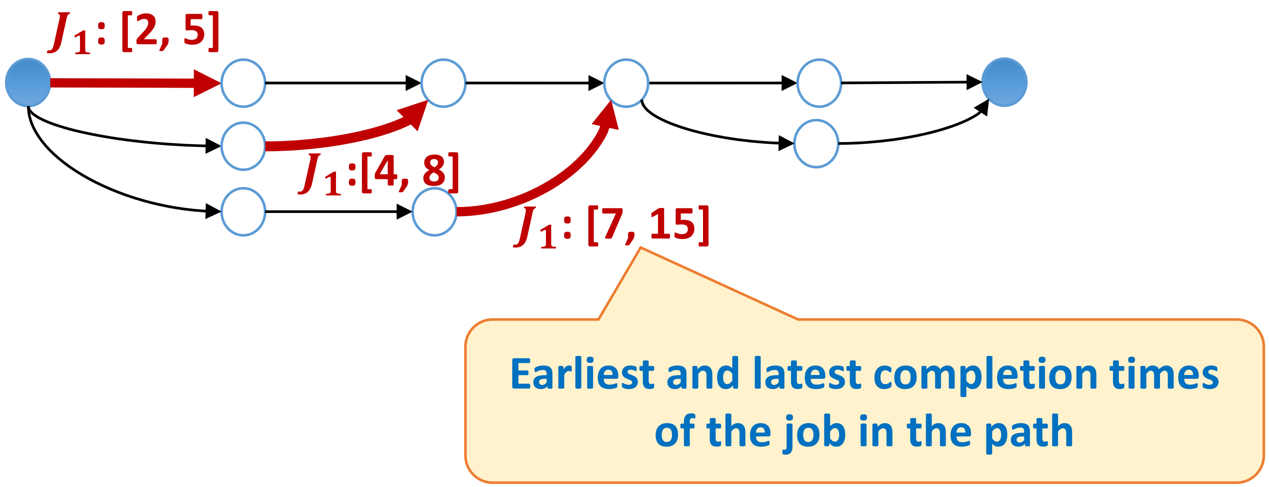
Response-time analysis using schedule-abstraction graphs

A path aggregates all schedules with the same job ordering

A vertex abstracts a system state and an edge represents a dispatched job

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

Obtaining the response time:



Best-case response time = **min** {completion times of the job} = **2**

Worst-case response time = **max** {completion times of the job} = **15**

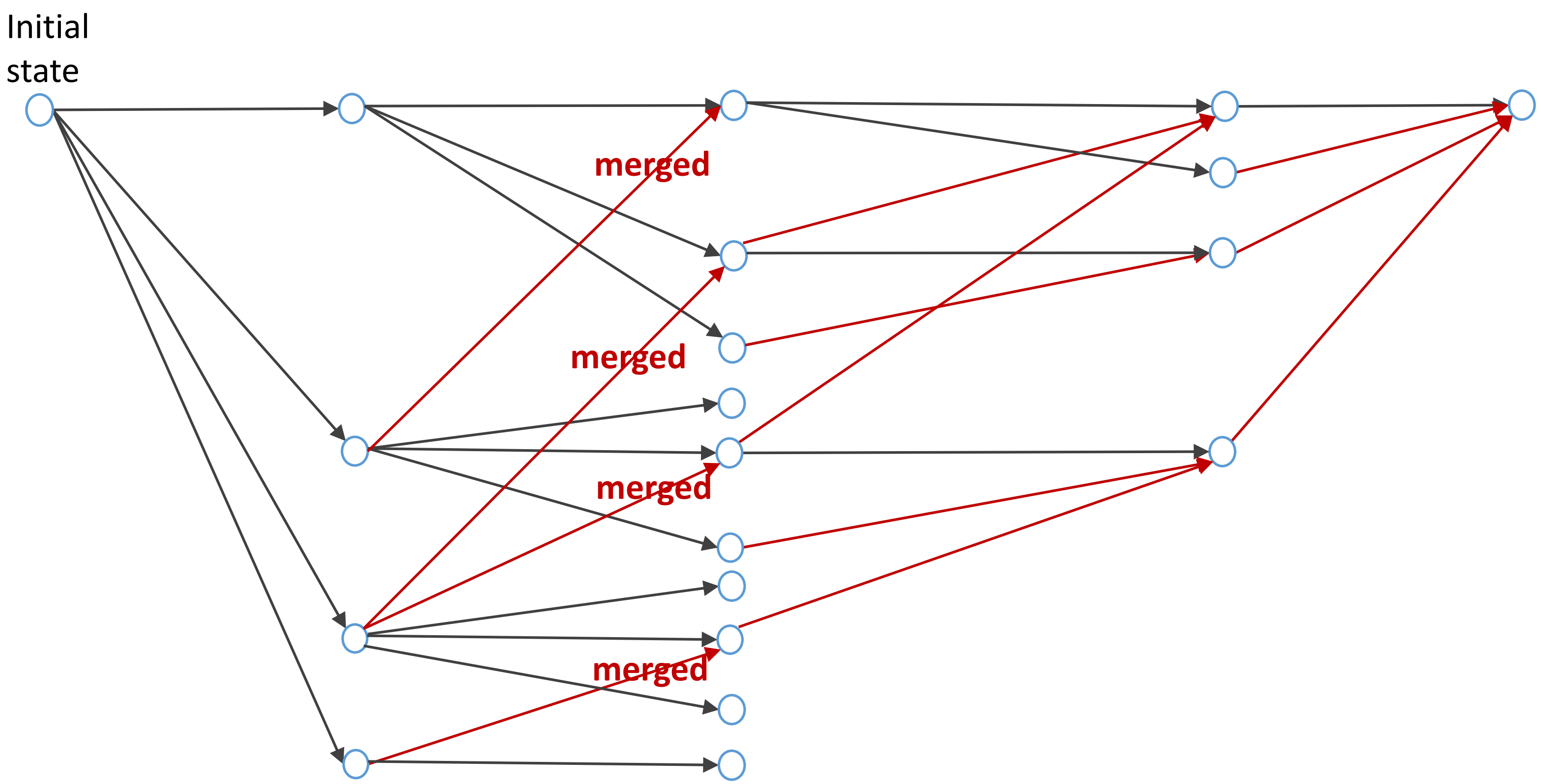
Building the schedule-abstraction graph

Building the graph
(a breadth-first method)

Repeat until every path includes all jobs

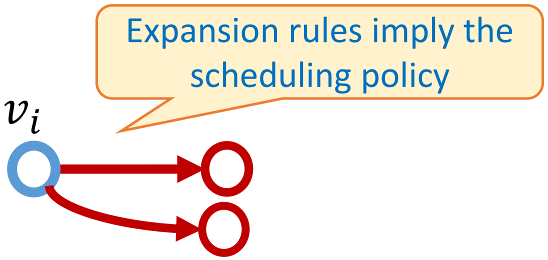
1. **Find the shortest path**
2. **For each not-yet-dispatched job that can be dispatched after the path:**
   - **2.1. Expand** (add a new vertex)
   - **2.2. Merge** (if possible, merge the new vertex with an existing vertex)

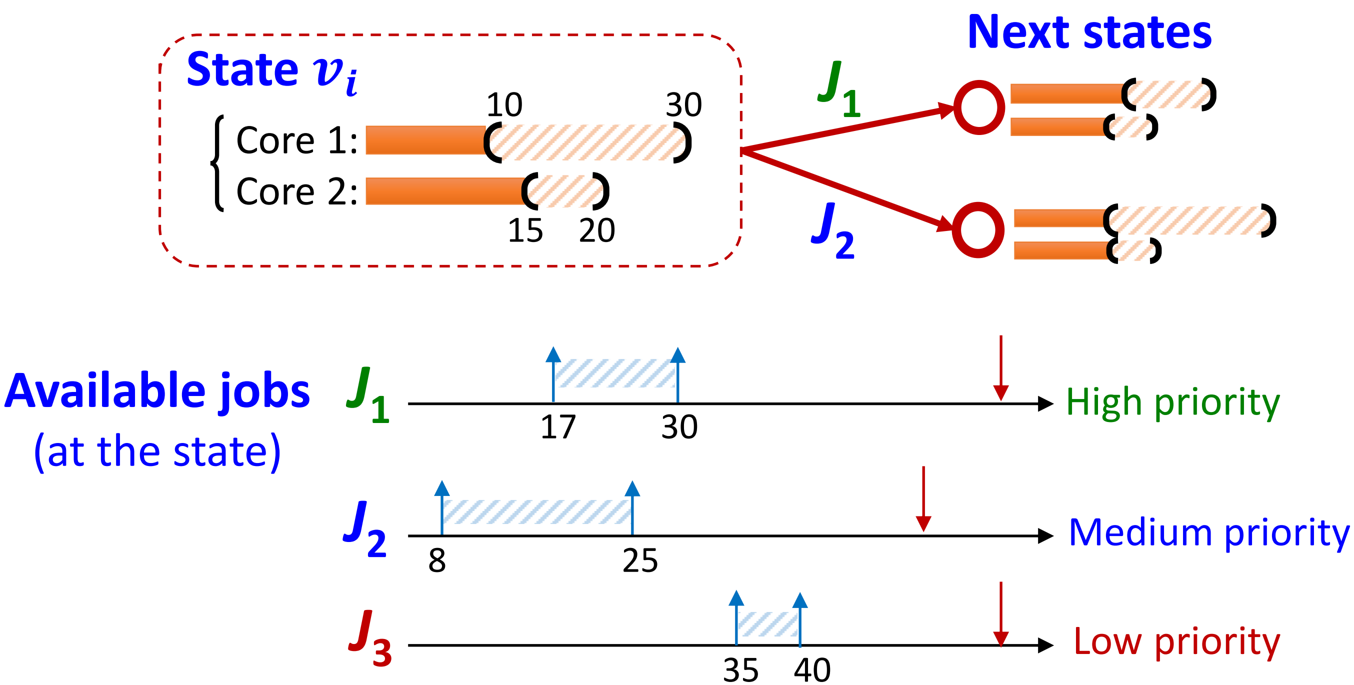
System is idle and
no job has been scheduled



Building the schedule-abstraction graph

Expanding a vertex:
(reasoning on uncertainty intervals)





How to use schedule-abstraction graphs to solve a new problem?

What is encoded by an edge?
What is encoded by a state?

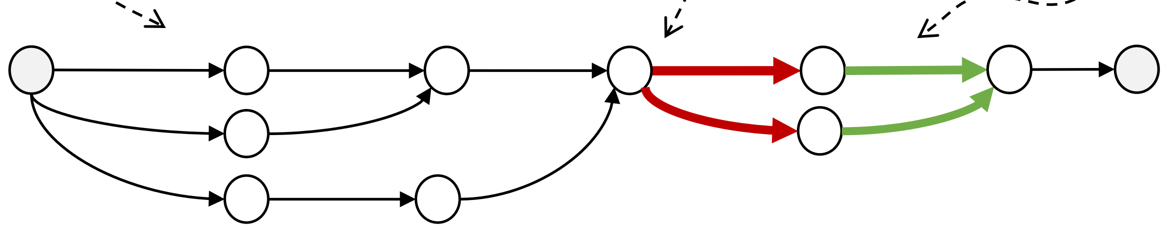
How to create
new states?

How to identify
similar states?

Define the state abstraction

Define the expansion rules

Define merging rules



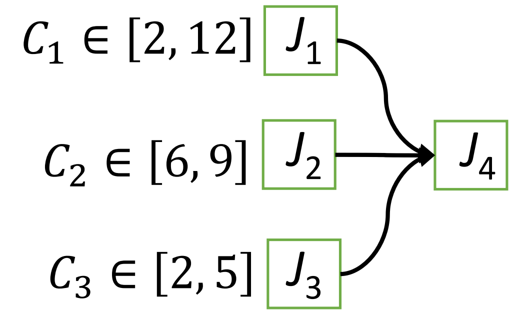
And then, prove soundness

“the expansion rules must cover all possible schedules of the job set”

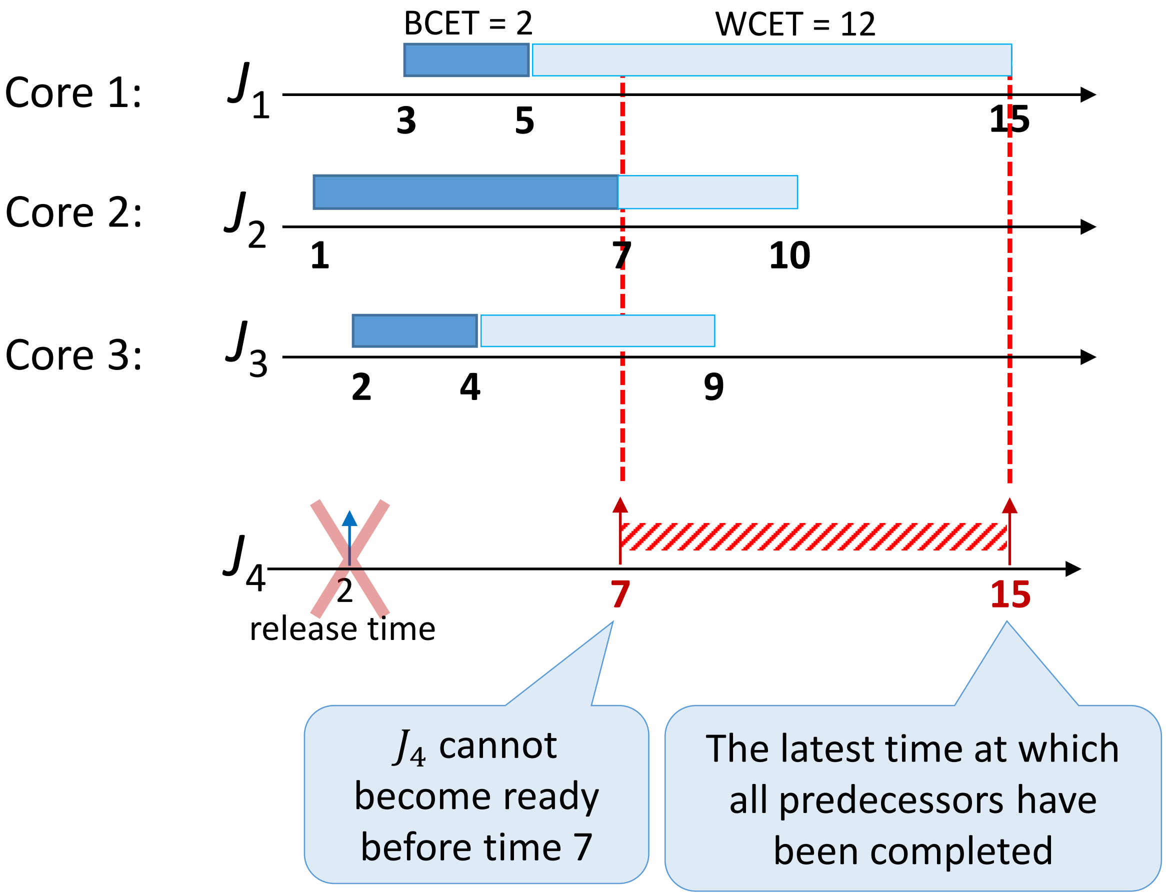
Agenda

- ~~Schedule-abstraction based analysis~~
- Supporting precedence constraints
  - Challenges
  - A new abstraction
- Evaluation
- Conclusion and future work

Handling precedence constraints

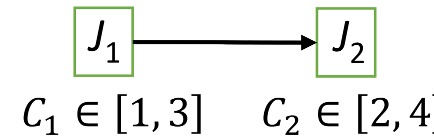


An example schedule:

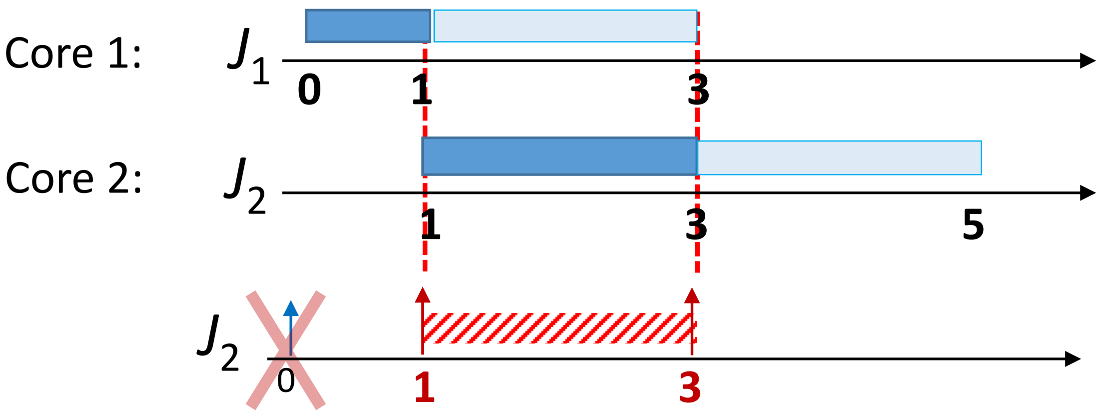


Is that enough?

Challenge 1: modeling precedence constraint as release jitter may cover impossible scenarios (→ pessimism)

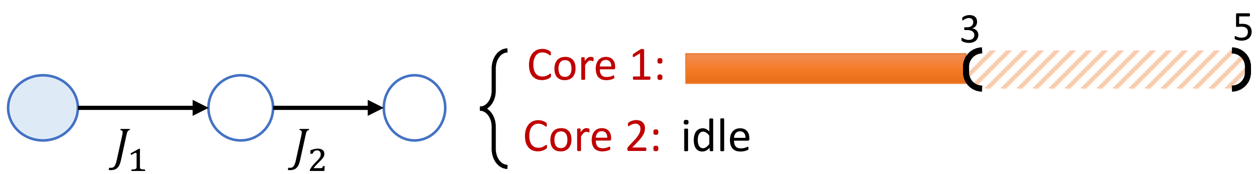


An example schedule:

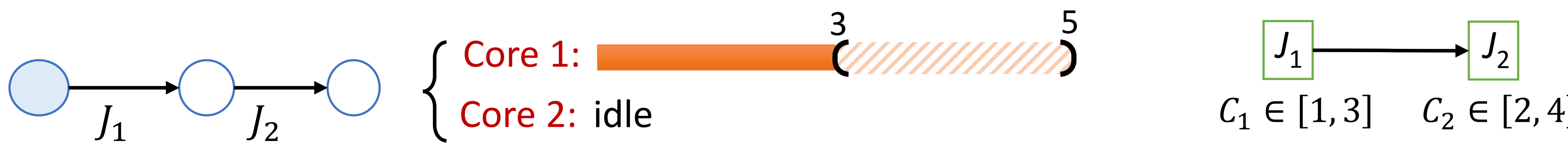


Is there a scenario at which two cores are busy at any time in the interval [1, 3]?

No! because J_2 can start its execution only if J_1 has finished



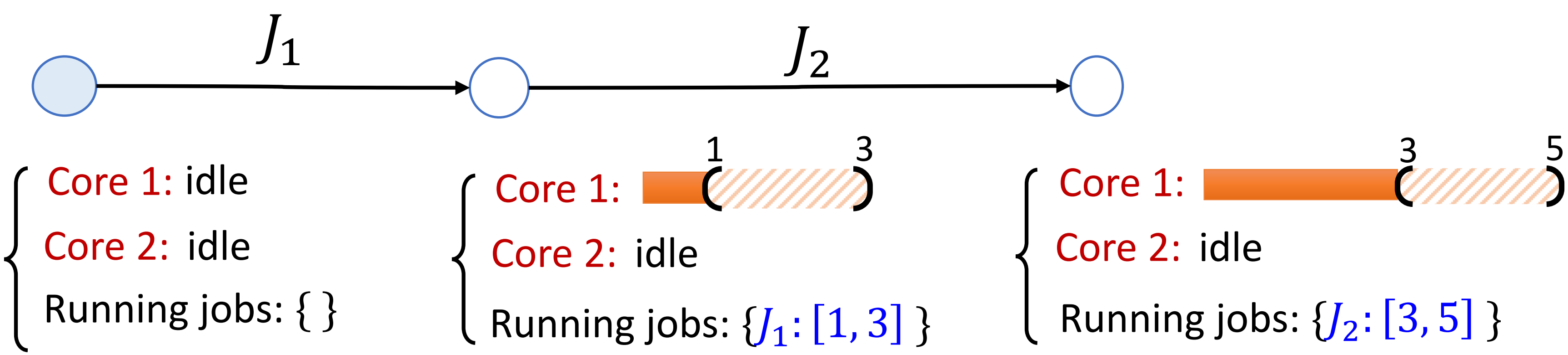
Challenge 1's solution: keep track of running jobs in a state



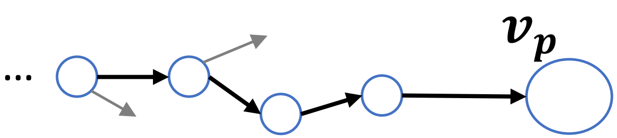
Maintain a set S of certainly running jobs and their completion time intervals

When scheduling J_i: Remove all its predecessors from S

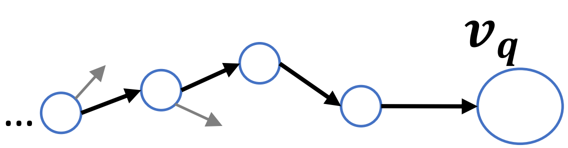
Update availability of cores considering the removal of J_i's predecessors



Challenge 2: Updating certainly running jobs after the merge phase

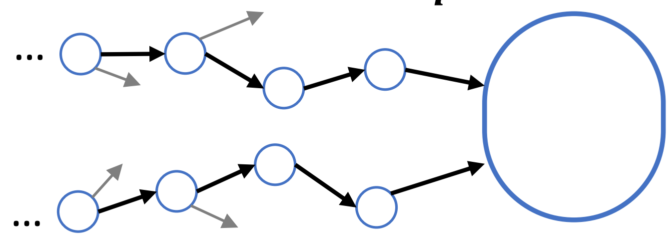


$$S = \begin{Bmatrix} J_1: [2, 5], \\ J_2: [7, 10] \end{Bmatrix}$$



$$S = \begin{Bmatrix} J_1: [4, 8], \\ J_3: [3, 9] \end{Bmatrix}$$

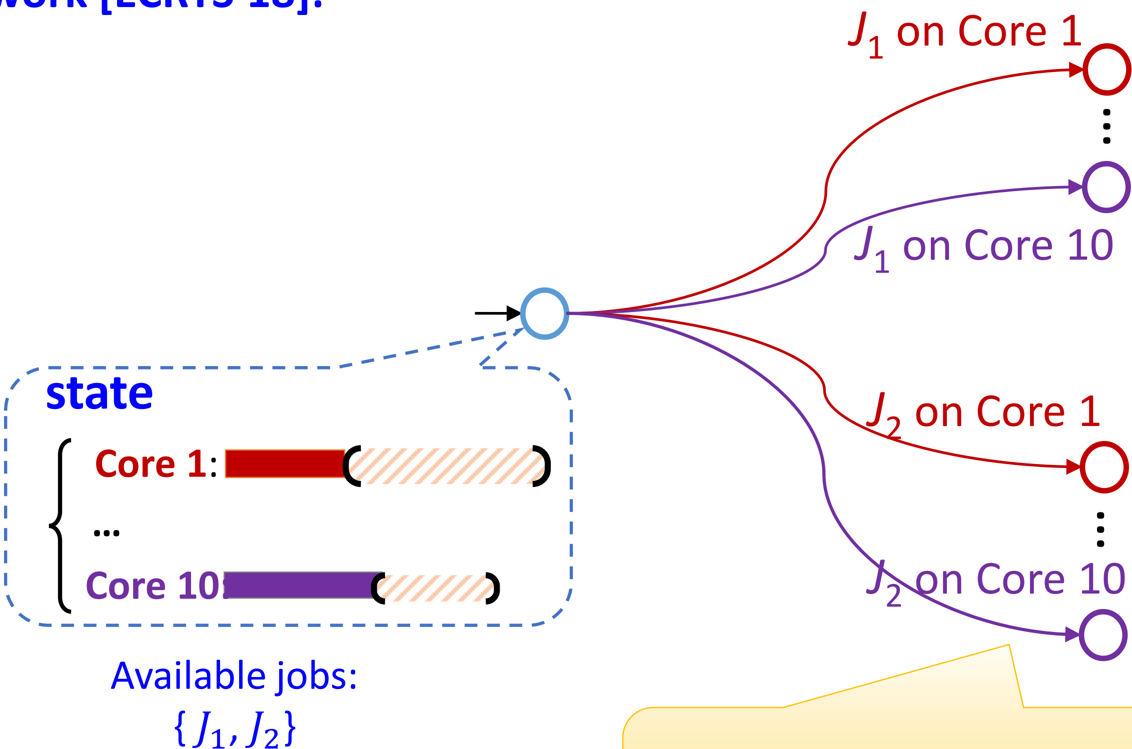
v_p merged with v_q



$$S = \{J_1: [2, 8]\}$$

Challenge 3: improving the scalability (with a new state abstraction)

Prior work [ECRTS'18]:



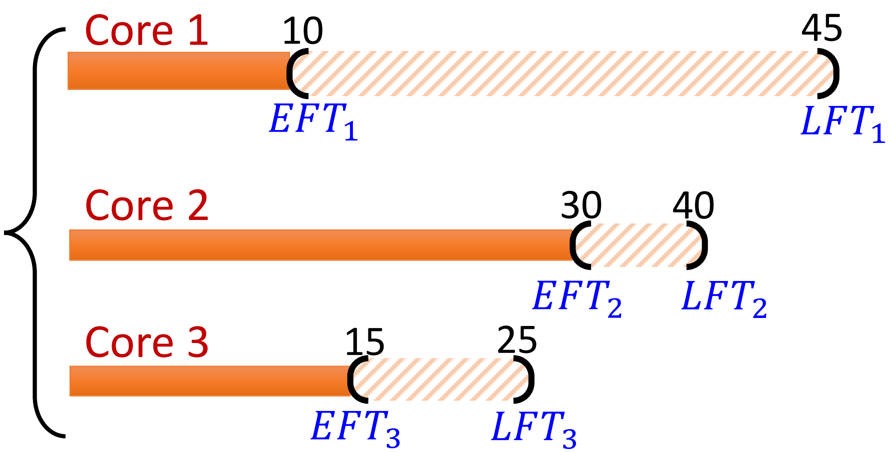
Assume that these jobs can be scheduled on either of the cores

Symmetry increases <u>the cost of</u> "expansion phase"

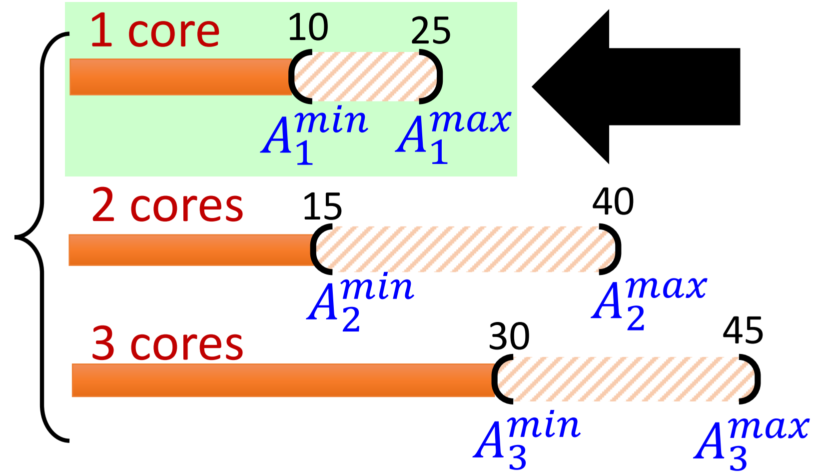
Note: these vertices will likely "merge" during the merge phase anyway

Our new state abstraction

[ECRTS'18]



This work



How does it help?

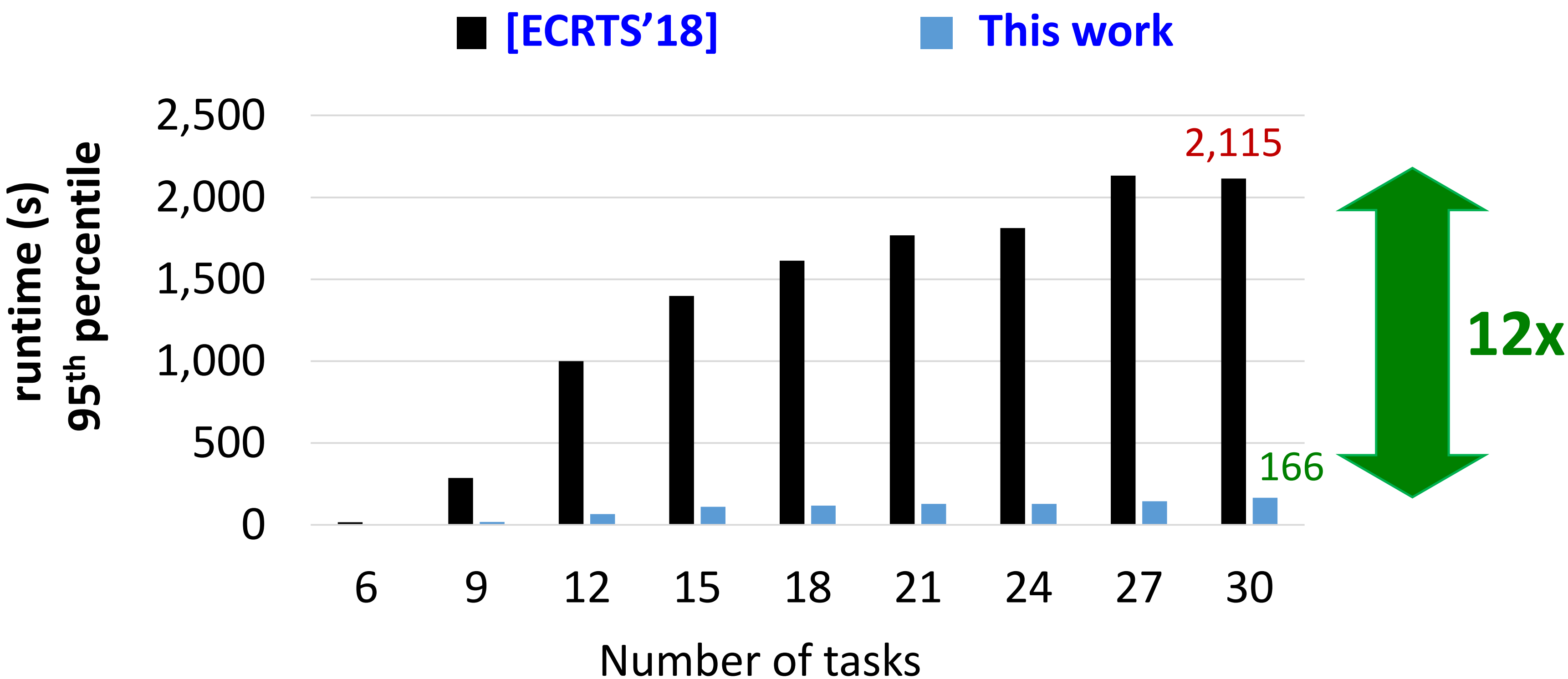
When a **new job** is dispatched, it only affects the **first core availability interval**

because there is no need to expand all combination of jobs and cores!

EFT_i: earliest finish time of the i^{th} core
LFT_i: latest finish time of the i^{th} core
A_i^{min}: earliest availability time of i cores
A_i^{max}: latest availability time of i cores

Evaluating the effect of the new abstraction

Non-preemptive periodic tasks, 4 cores, utilization = 70%



Agenda

- ~~Schedule-abstraction based analysis~~
- ~~Supporting precedence constraints~~
  - ~~Challenges~~
  - ~~A new abstraction~~
- Evaluations
- Conclusion and future work

Experiment setup

Experiment platform

- Multi-threaded C++ program.
  We parallelized the breadth-first exploration of the schedule-abstraction graph using Intel's open-source **Thread Building Blocks (TBB) library**.

- A cluster of machines each equipped with 256 GiB RAM and Intel Xeon E5-2667 v2 processors clocked at 3.3 GHz.

- We report the CPU time of all of the threads together as the runtime of the analysis

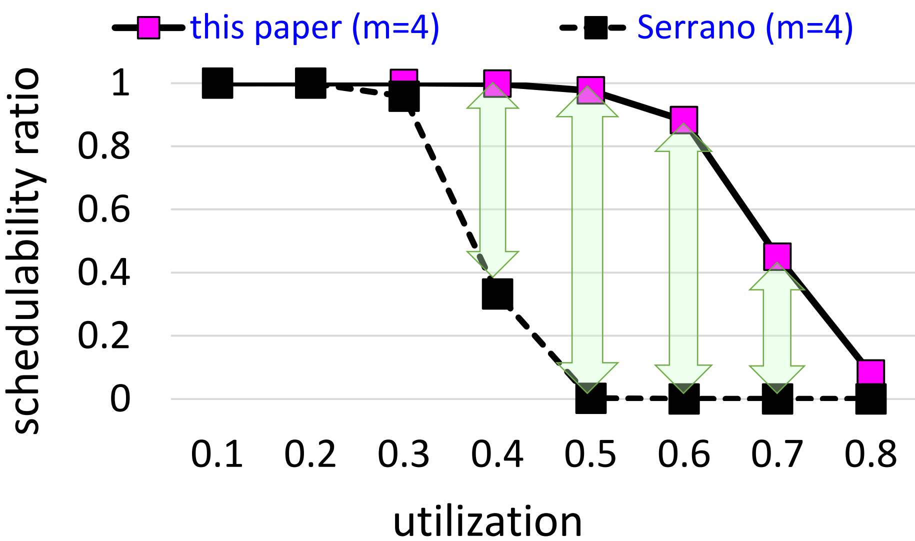
DAG tasks:

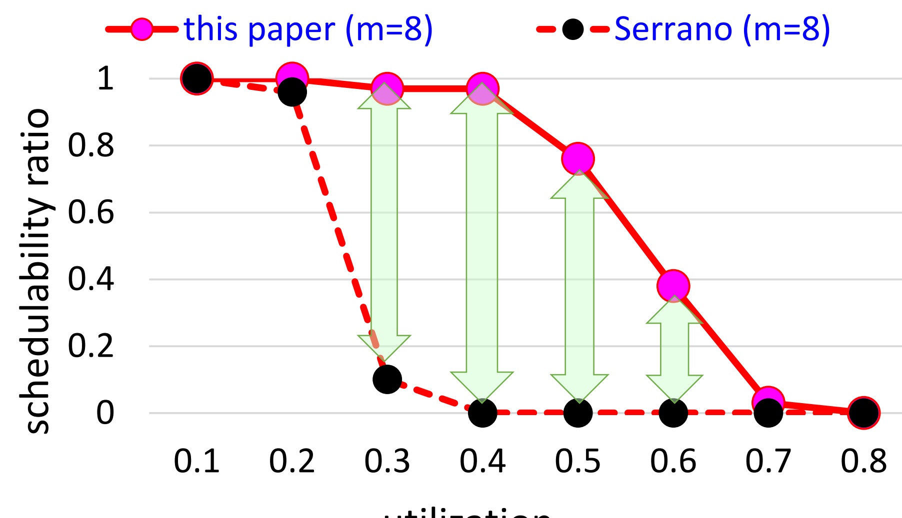
- Periods in [500, 100000]
- Utilization of a task: uUniFast

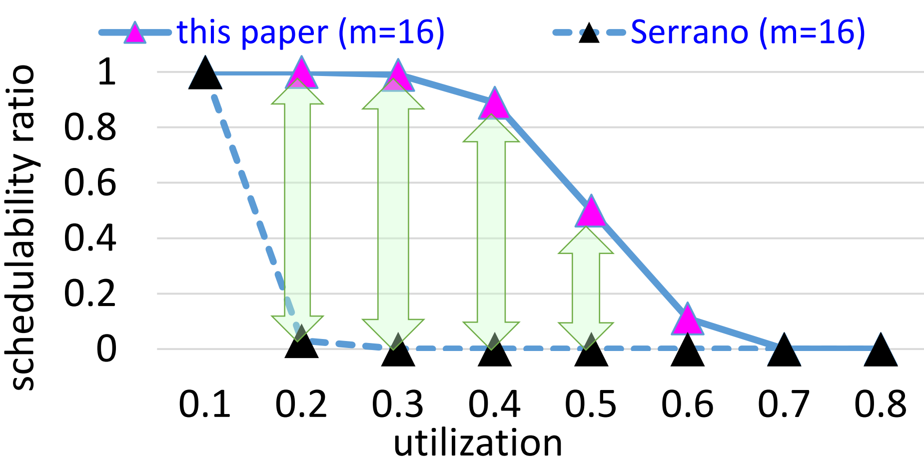
- Series-parallel DAGs with nested fork-joins generated with the method from [Cassini 2018, Serrano 2017, Melani 2015, Peng 2014]
  - Maximum nodes in a DAG: 50
  - Maximum length of the critical path: 10
  - Maximum nested branches: 3

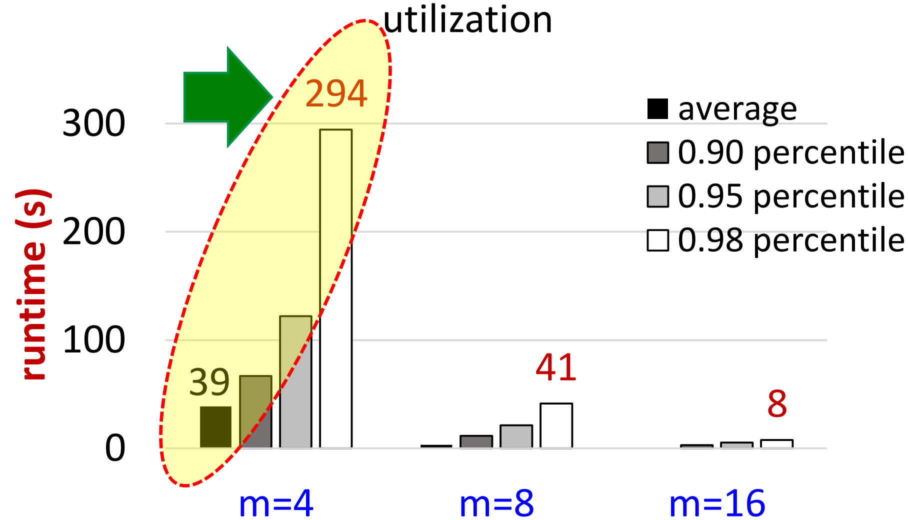
Parallel DAG tasks

10 parallel random DAG tasks







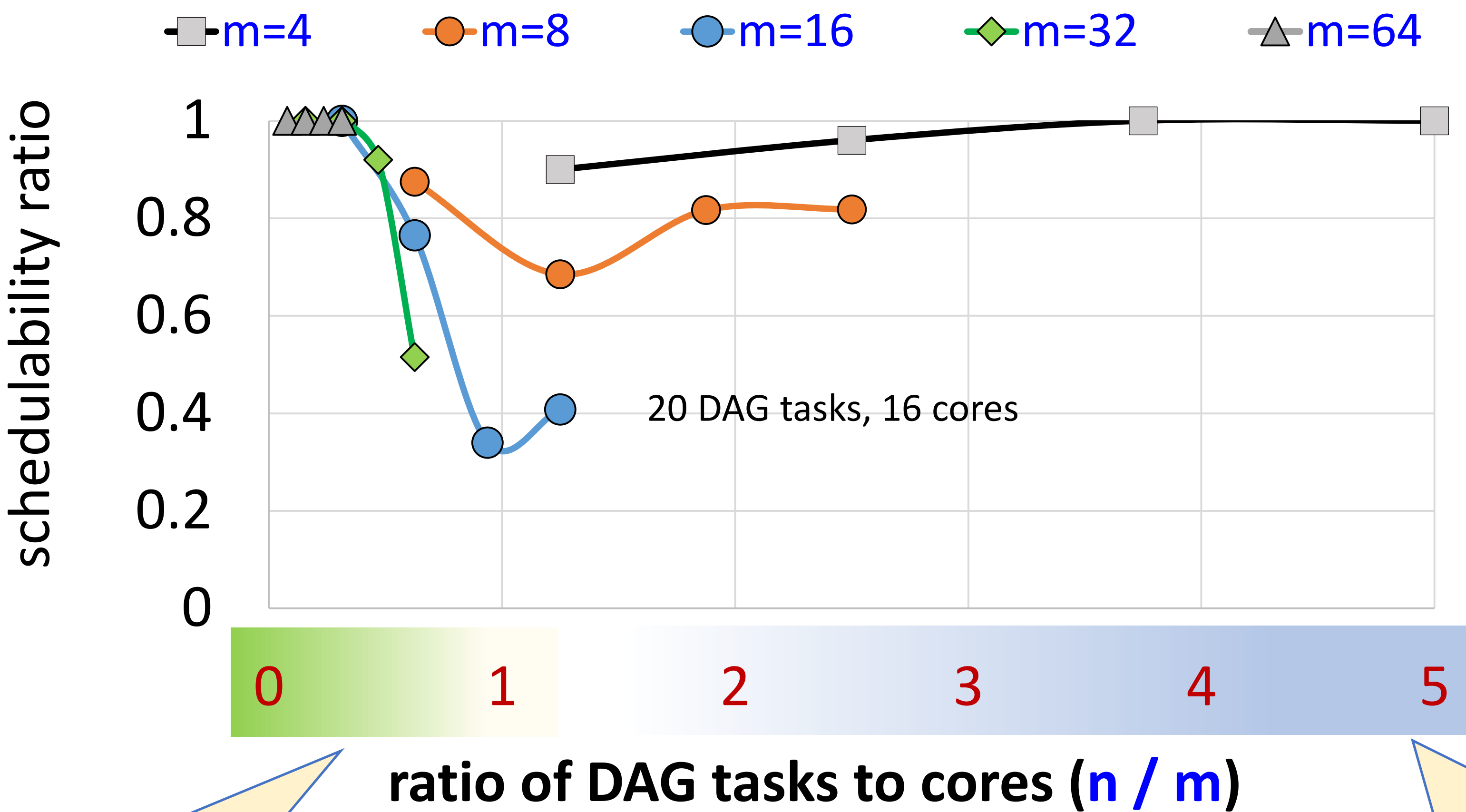


M. Serrano, A. Melani, S. Kehr, M. Bertogna, and E. Quiñones, “An Analysis of Lazy and Eager Limited Preemption Approaches under DAG-Based Global Fixed Priority Scheduling”, ISORC, 2017.

DAG tasks: varying cores (m) and tasks (n)

U=50%

Scalability experiment

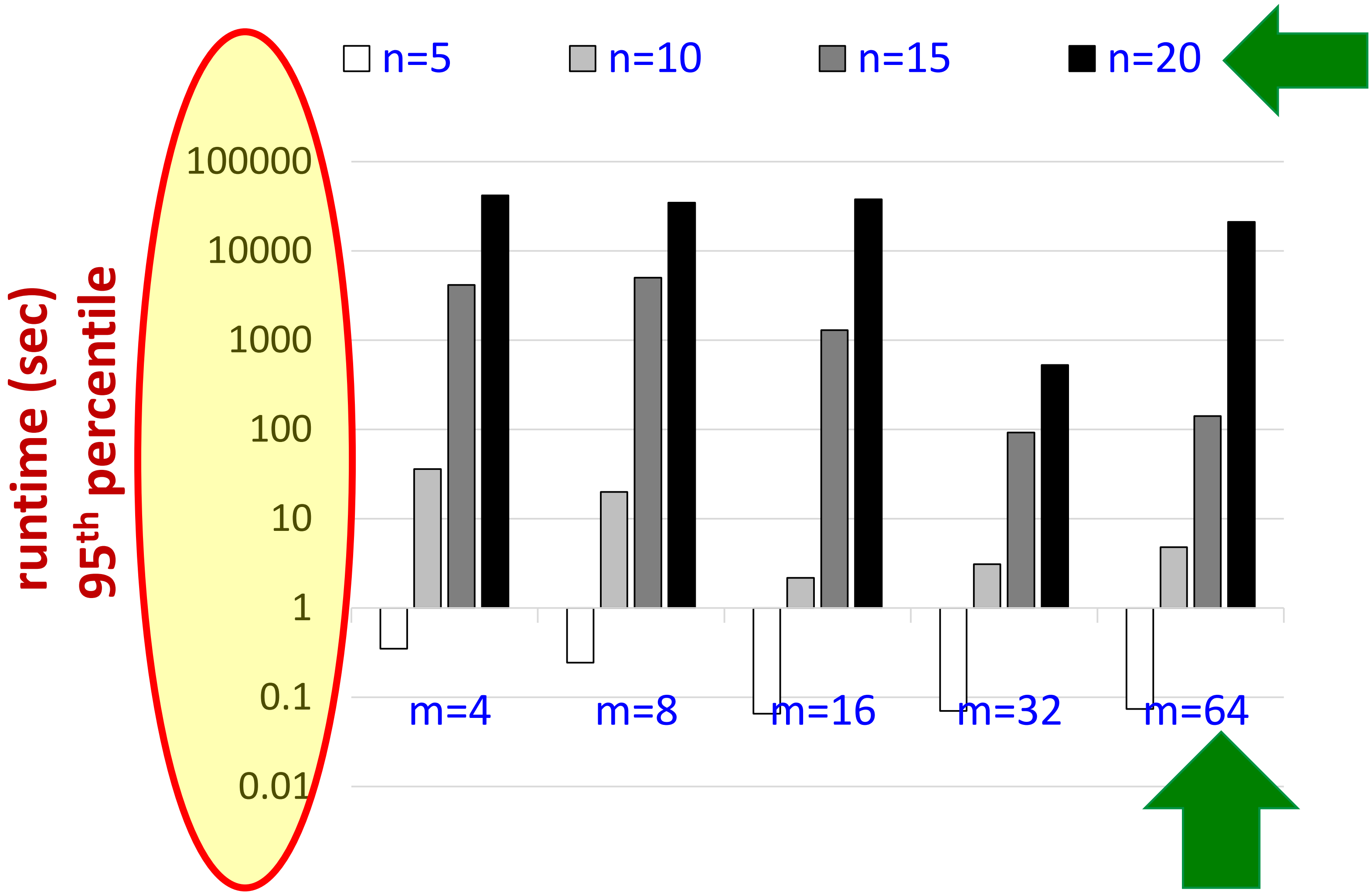


More cores than tasks: higher parallelism

More tasks than cores: each task has smaller utilization

DAG tasks: varying cores (m) and tasks (n)

U=50%



Conclusions and future directions

Conclusion

Response-time analysis using schedule abstraction

+ **New abstraction**

  + **Expansion rules to support precedence constraints**

Results: achieving **high accuracy** (similar to UPPAAL) while being able to **scale to practically relevant** system sizes ($n \leq 20, m \leq 64$)

Mitra Nasri, Geoffrey Nelissen, Björn Brandenburg

TU Delft

CISTER
Research Centre in Real-Time & Embedded Computing Systems

MAX PLANCK INSTITUTE FOR SOFTWARE SYSTEMS

This work

Multiprocessor | **Parallel DAG tasks**

Better state abstraction | **Global** job-level fixed-priority scheduling (JLFP)

Future work

near future → in a few years → eventually

near future
- **Preemptive** execution
- Partial-order reduction
- **Heterogeneous platforms**
- **Self-suspending tasks**

in a few years
- **Sporadic tasks**
- Gang scheduling
- **Shared resources**
- Co-running tasks

eventually
- Dynamic schedulers
- **Combine the framework with timing analysis tools**

The framework is open source. You can find that on the authors' page.

Thank you