MAX PLANCK INSTITUTE
FOR SOFTWARE SYSTEMS

An Exact and Sustainable Analysis of Non-Preemptive Scheduling

Mitra Nasri*   Björn B. Brandenburg

Max Planck Institute for Software Systems (MPI-SWS)
Germany

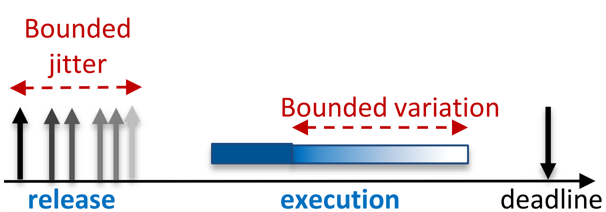
RTSS, December 2017

Our work in a nutshell

"An exact and sustainable schedulability analysis for non-preemptive scheduling"

Our analysis

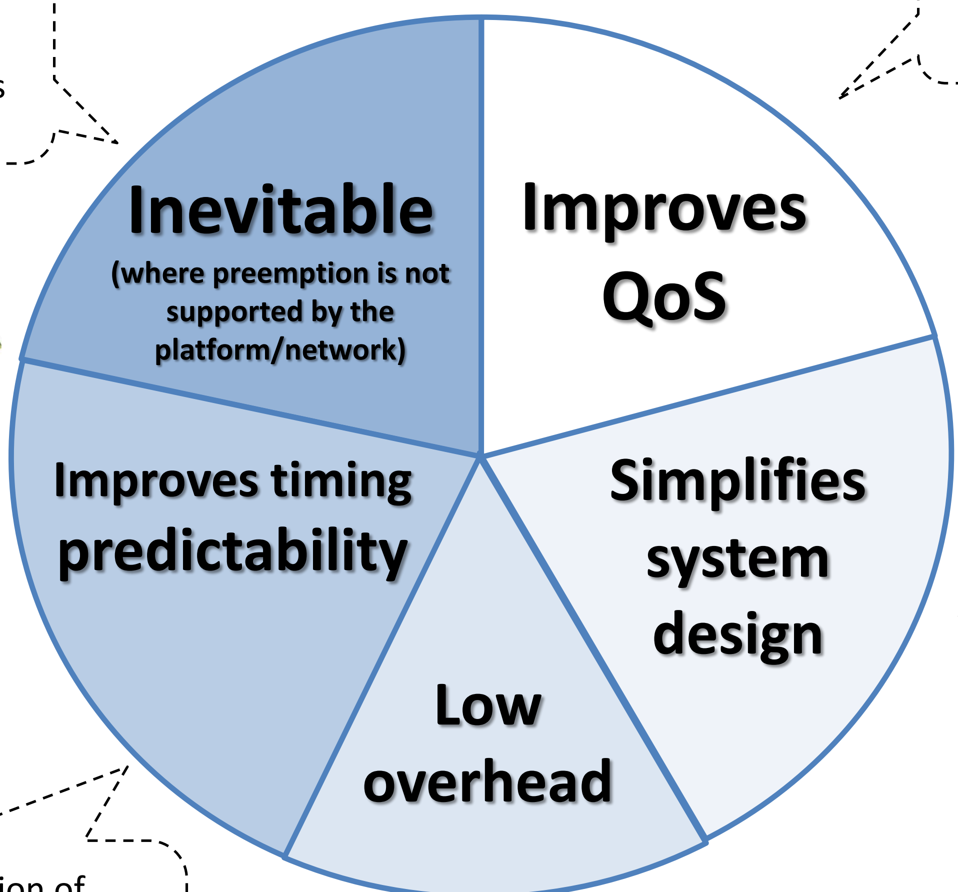
- **Non-preemptive jobs (or periodic tasks)**
- **Hard or soft timing constraints**
- **Uniprocessor**
- **Release jitter**
- **Execution time variation**
- **Fixed-job priority scheduling algorithms**
  - EDF
  - Fixed priority
- **A wide class of non-work conserving scheduling algorithms**
  - CW-EDF [Nasri16]
  - Precautious-RM [Nasri14]
- **Exact schedulability analysis**
- **Exact best-case and worst-case response time**
  - And tardiness bound



Why non-preemptive scheduling?

Examples
- GPU device
- Hardware accelerators
- CAN bus

- **Control systems** are sensitive to I/O delay and preemptions



- Simpler resource management policies
- Grants exclusive resource access

- A more accurate estimation of **worst-case execution-time (WCET)**
- More predictable cache

- Reduces context switches
- Avoids intra-task **cache-related preemption delays** **(CRPD)**

Why do we need a new analysis?

Most of NPS policies are not sustainable
(w.r.t. execution time variation, release jitter, etc.)
→ **Simulation-based schedulability tests cannot be used**

Schedulability analyses for sporadic tasks
[Jeffay91, Tindel94, Davis07]
⇒ **Pessimistic** for **periodic** tasks
⇒ **Not applicable** to arbitrary **job sets**
→ **Existing analyses are not enough**

Existing schedulability analyses based on model checking, timed automata, abstraction refinements, etc.
[Sun97, Baker07, Guan07, Bonifaci10, Burmyakov15, Stigge15]
⇒ **Not very scalable**
→ **Existing analyses are not efficient**

Many non-work-conserving scheduling algorithms do NOT have a schedulability analysis yet
→ **No solution yet**

What do we want?

An efficient, exact, general

schedulability analysis

THAT includes

a wide class of scheduling algorithms and task models

Image is from http://theyoungprofessionalgroup.com

Agenda

- **Main idea:**

Searching all possible schedules efficiently and accurately

- Constructing the search graph
- Evaluation
- Conclusion

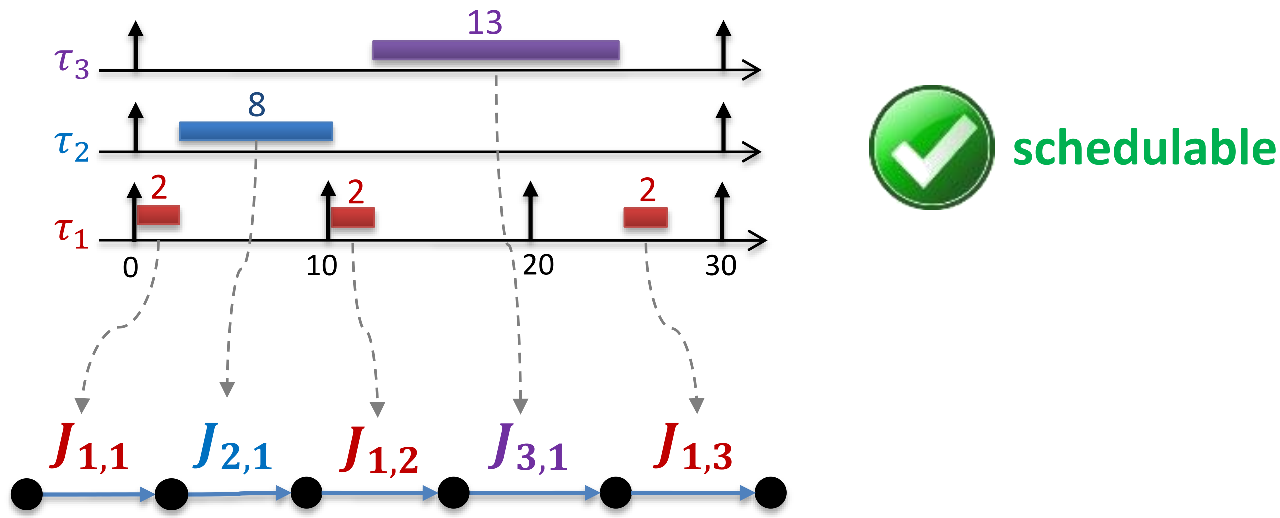
Basic scenario: no runtime variation in the workload

Task	Period	Execution time
τ_3	30	13
τ_2	30	8
τ_1	10	2

Non-preemptive fixed-priority scheduling

One schedule

One job ordering

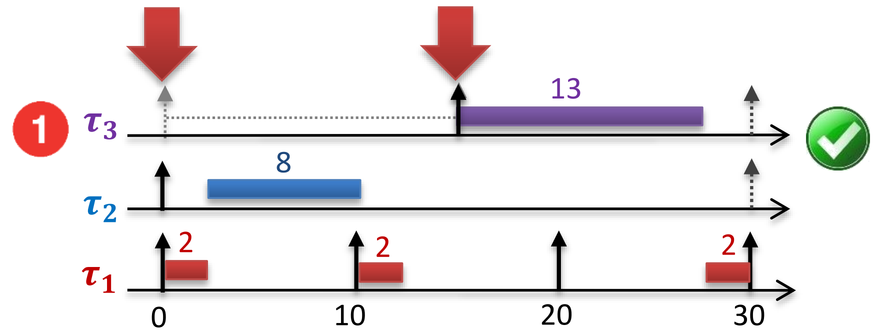


Both existing tests for sporadic tasks reject this task set [Jeffay91, Davis07]

Values are integer.
Scheduling algorithm: Non-preemptive fixed-priority (NP-FP)
A schedule is an assignment of execution intervals to the jobs.

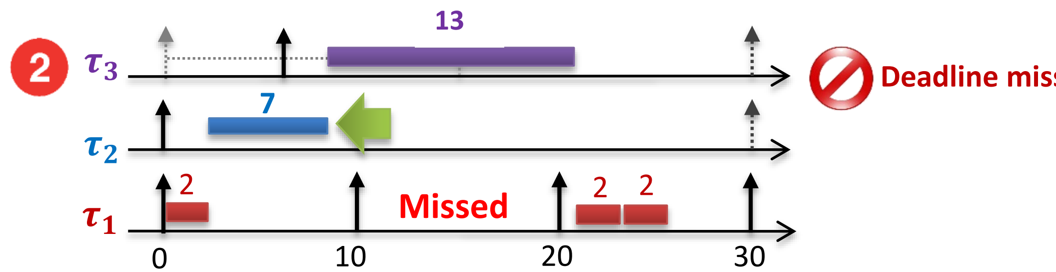
Scenario: execution time variation and release jitter

Task	Period	Execution time Min Max	Release jitter
τ_3	30	[3, 13]	15
τ_2	30	[7, 8]	0
τ_1	10	[1, 2]	0

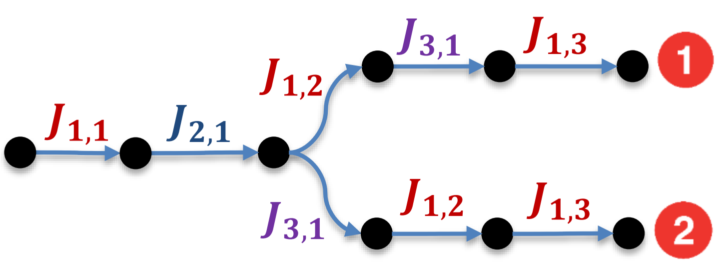


More than 100
different schedules

Not schedulable



Only two
different job orderings



A graph of job orderings

Values are <u>integer</u>.
Scheduling algorithm: NP-FP
A <u>schedule</u> is an assignment of execution intervals to the tasks.

Challenges

Due to scheduling anomalies

For an exact analysis, we need to consider **all possible execution scenarios**

Observation

There are fewer permissible job orderings than schedules

Research question

Is there a way to use job-ordering abstraction to analyze schedulability?

How to abstract schedules in a graph of job orderings?

How to efficiently find all job orderings?

How to identify timing violations in the resulting graph?

Abstracting schedules in a graph of job orderings

Requirement

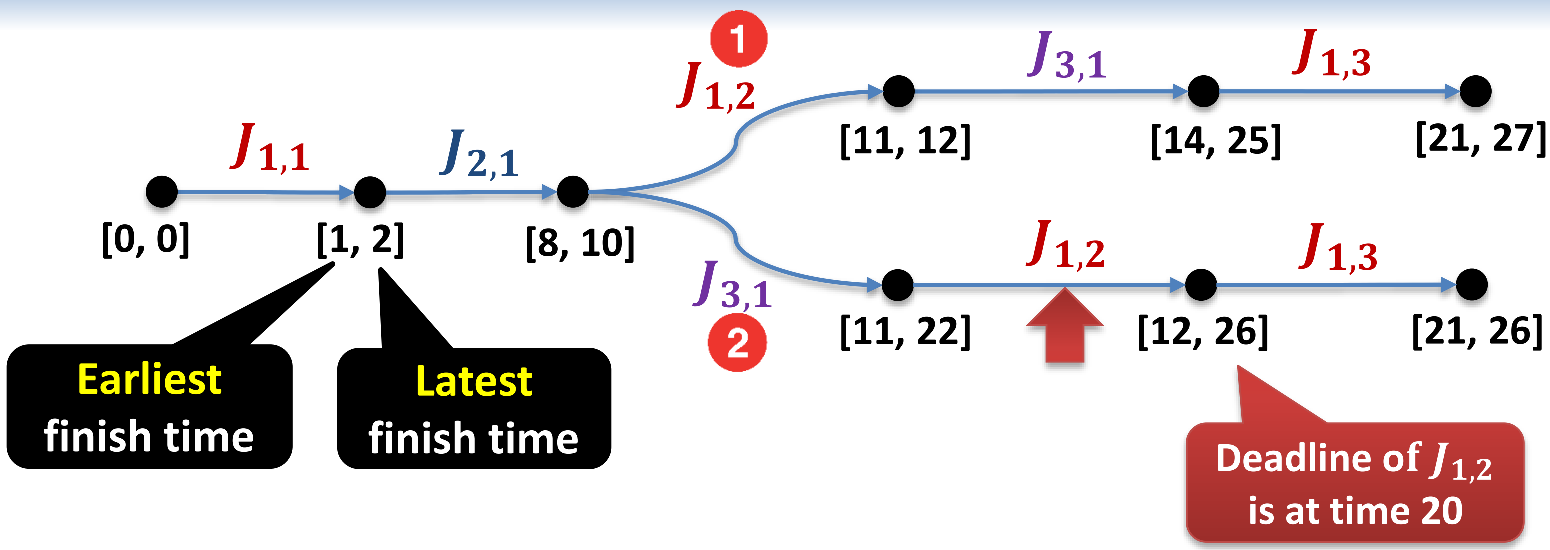
Knowing when a job misses its deadline

Solution

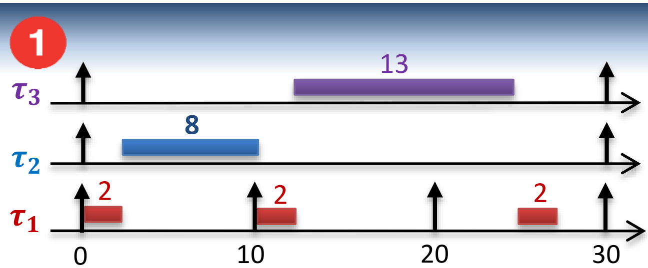
Encode the **earliest** and **latest finish time** of a job

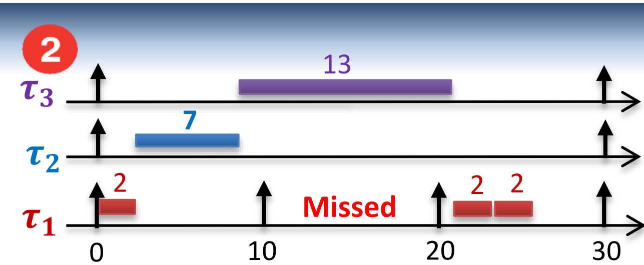
Verification of schedulability

Check if the **latest finish time** is not larger than the deadline



Each path shows a job ordering



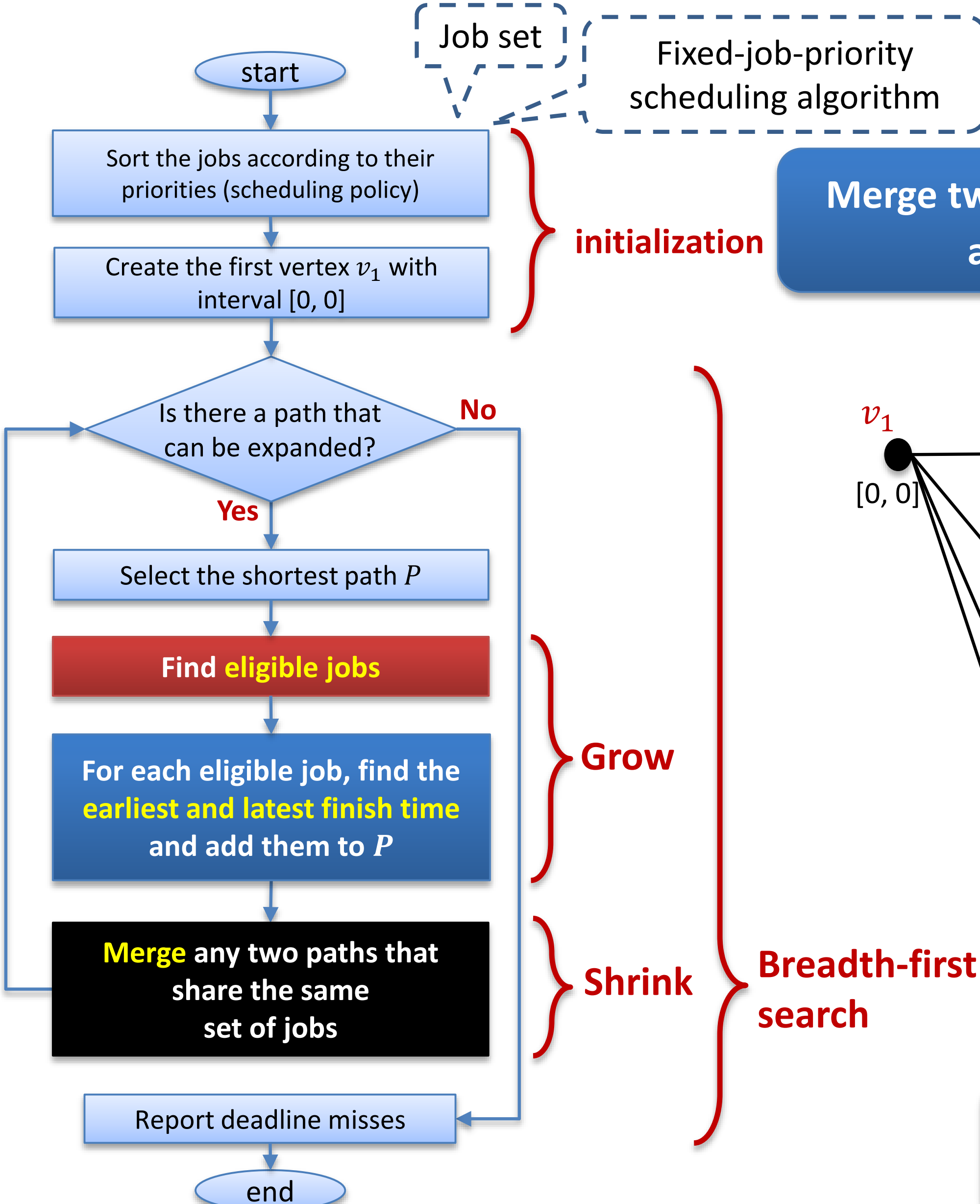


Task	Period	Execution time Min	Execution time Max	Jitter
τ_3	10	[3,	13]	15
τ_2	30	[7,	8]	0
τ_1	30	[1,	2]	0

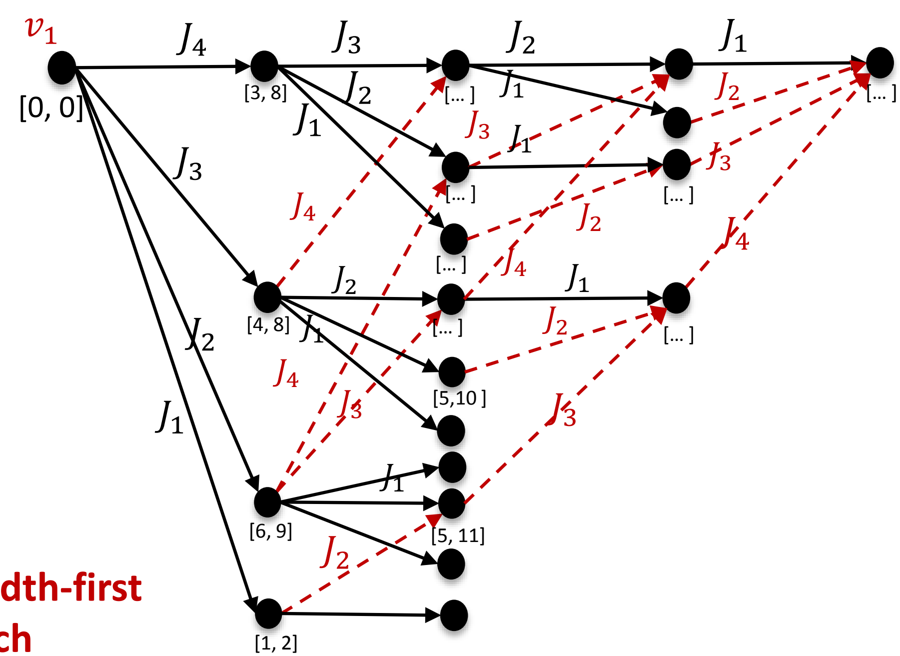
Agenda

- Main idea: Searching all possible execution scenarios efficiently and accurately
- **Constructing the search graph**
- Evaluation
- Conclusion

Constructing the search graph

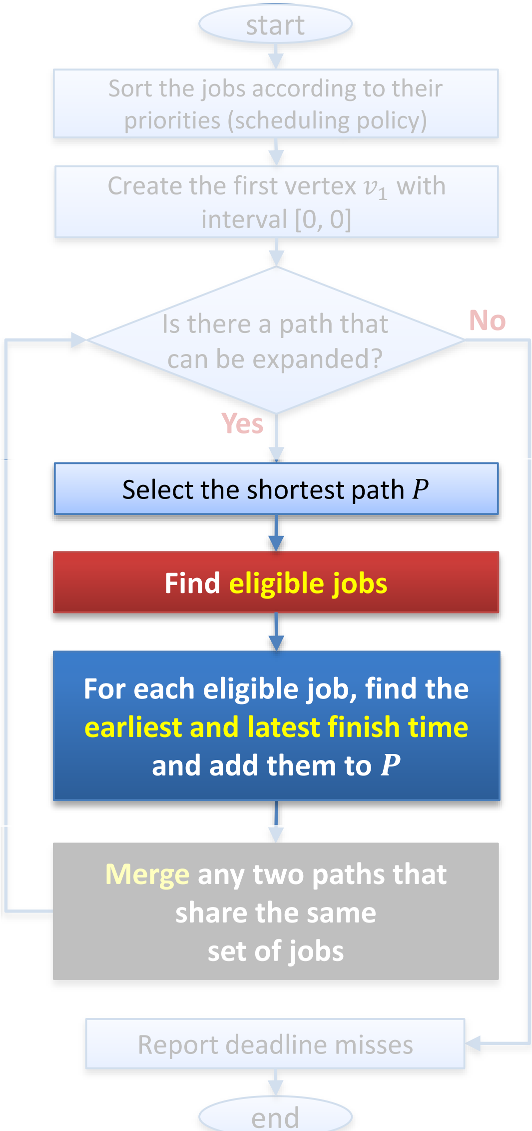


Merge two paths if they have the same set of jobs and their final intervals intersect

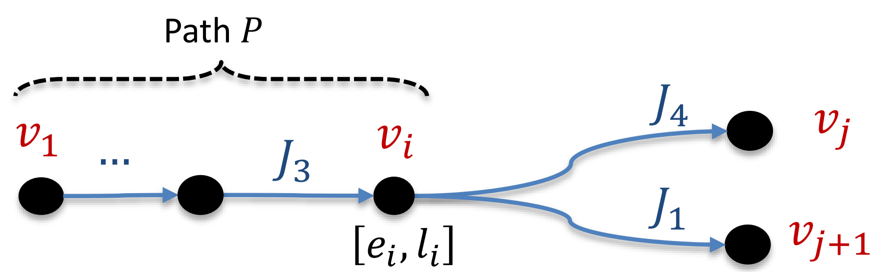


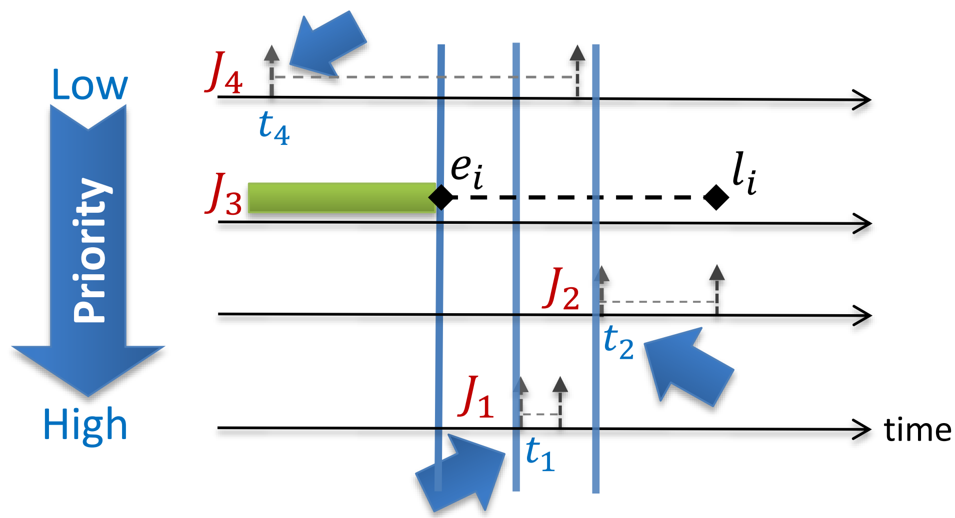
The graph grows more slowly

Growing the graph



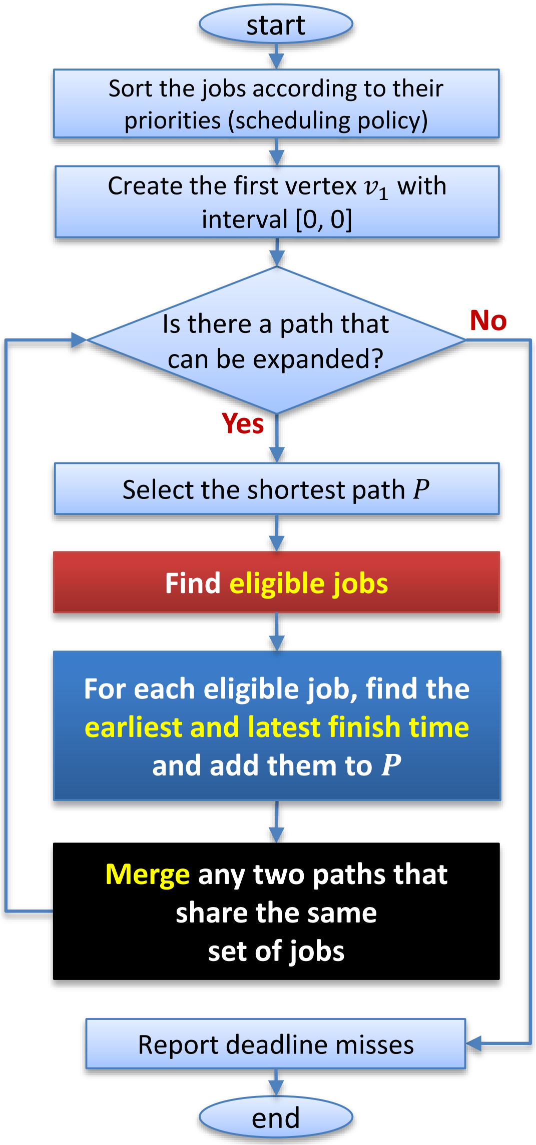
An **eligible job** for path P is a job that can be scheduled after P in **at least one execution scenario**





e_i = the earliest finish time of path P
l_i = the latest finish time of path P

Requirements of an <u>exact</u> analysis



"Eligibility conditions" are <u>necessary</u> and <u>sufficient</u>

The "final interval" of each is exact:
For any time t in the interval, there must be an execution scenario that ends at t

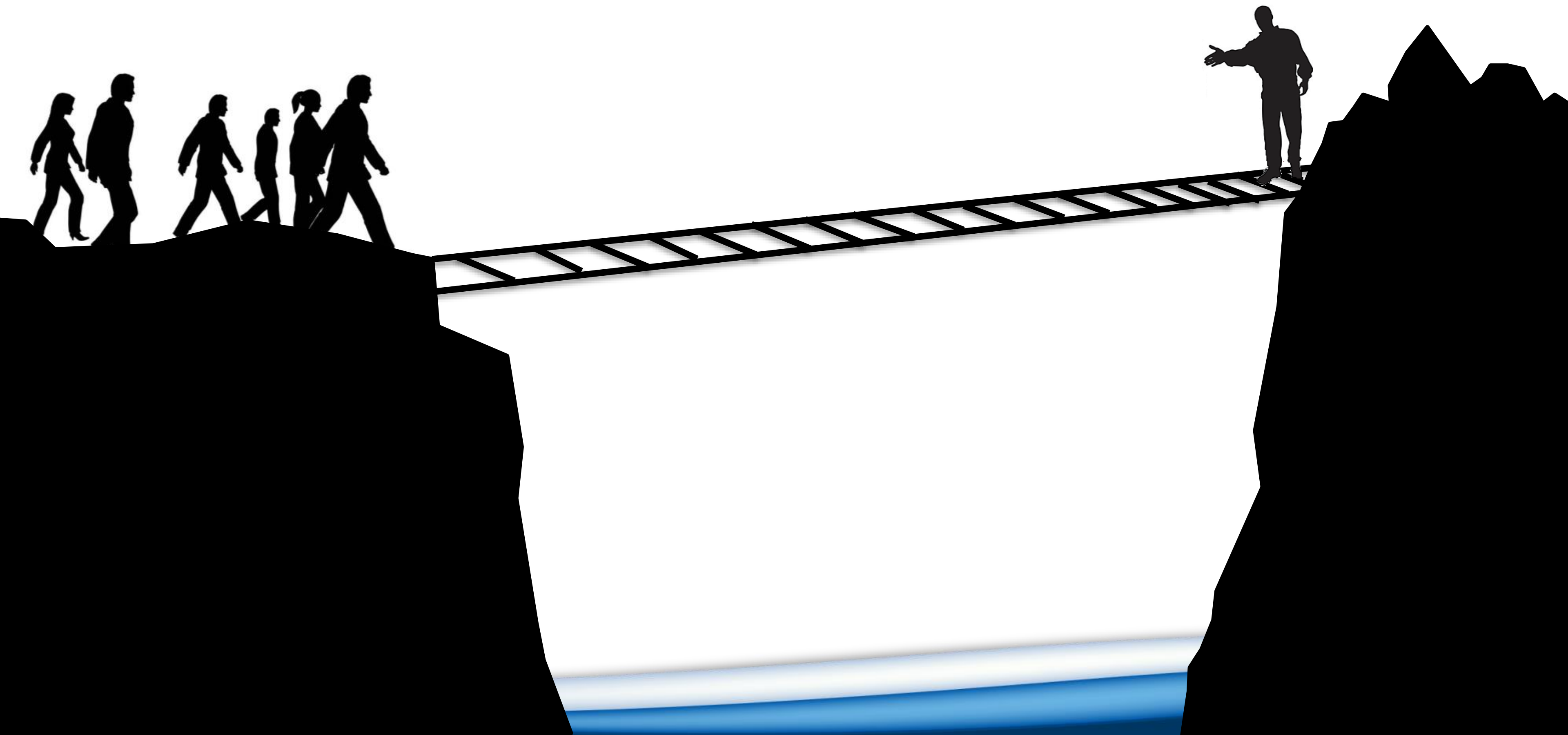
Final intervals remain "exact" after merging process

In our work, we have proved these properties for

- Fixed-job-priority scheduling algorithms
- Tasks with release jitter and execution time variation
- Hard and soft timing constraints
- Work-conserving and non-work-conserving scheduling algorithms

How to apply the analysis to a new system or algorithm?

1. Define eligibility conditions
2. Define how to obtain the final intervals
3. Prove the aforementioned properties



Agenda

- Main idea: Searching all possible execution scenarios efficiently and accurately
- Constructing the search graph
- **Evaluation**
- Conclusion

Image from https://searchengineland.com/answer-box-experiment-journey-known-unknown-factors-270948

Main questions

- Is our analysis **effective**?
  - Does it actually improve the accuracy of schedulability analysis?
  - What is our **achievement** for **non-work-conserving** scheduling policies?

- Is our analysis **efficient**?
  - How fast is the analysis?

Evaluation setup

	Automotive benchmark task sets [Kramer15]	Synthetic task sets
No jitter		
Small jitter (up to 100 microseconds)		
Large jitter (up to 20% of the period)		

Automotive benchmark task sets [Kramer15]

- **Variable parameter**: **utilization**
- Generate **runnables** according to [Kramer15] until the given utilization is reached
- Pack a random number of runnables together to build a **task**
- **Up to 30 tasks per task set**

To evaluate the effectiveness
in a realistic setup and different utilization values

Synthetic task sets

- **Variable parameter**: maximum number of **jobs in a hyperperiod**
- Periods are from [1, 1000]ms with log-uniform distribution
- Up to 50% runtime variation in the execution time
- 10 tasks per task set

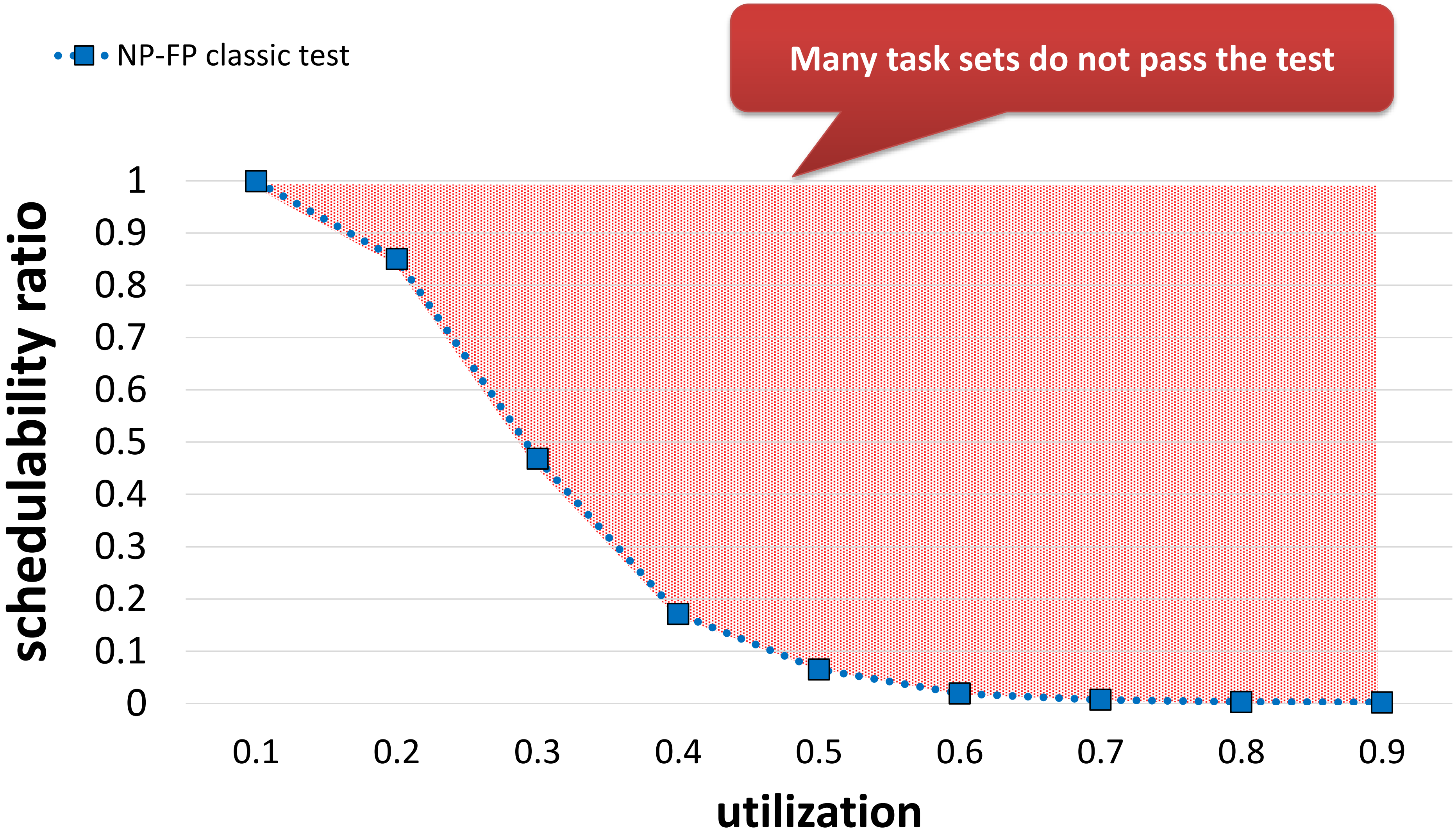
To evaluate the efficiency
when there are a large number of jobs

[Kramer15] S. Kramer, D. Ziegenbein, and A. Hamann, "Real world automotive benchmark for free," in WATERS, 2015.

Note: only task sets that pass the necessary schedulability condition of non-preemptive scheduling were considered.

How effective is our schedulability analysis?

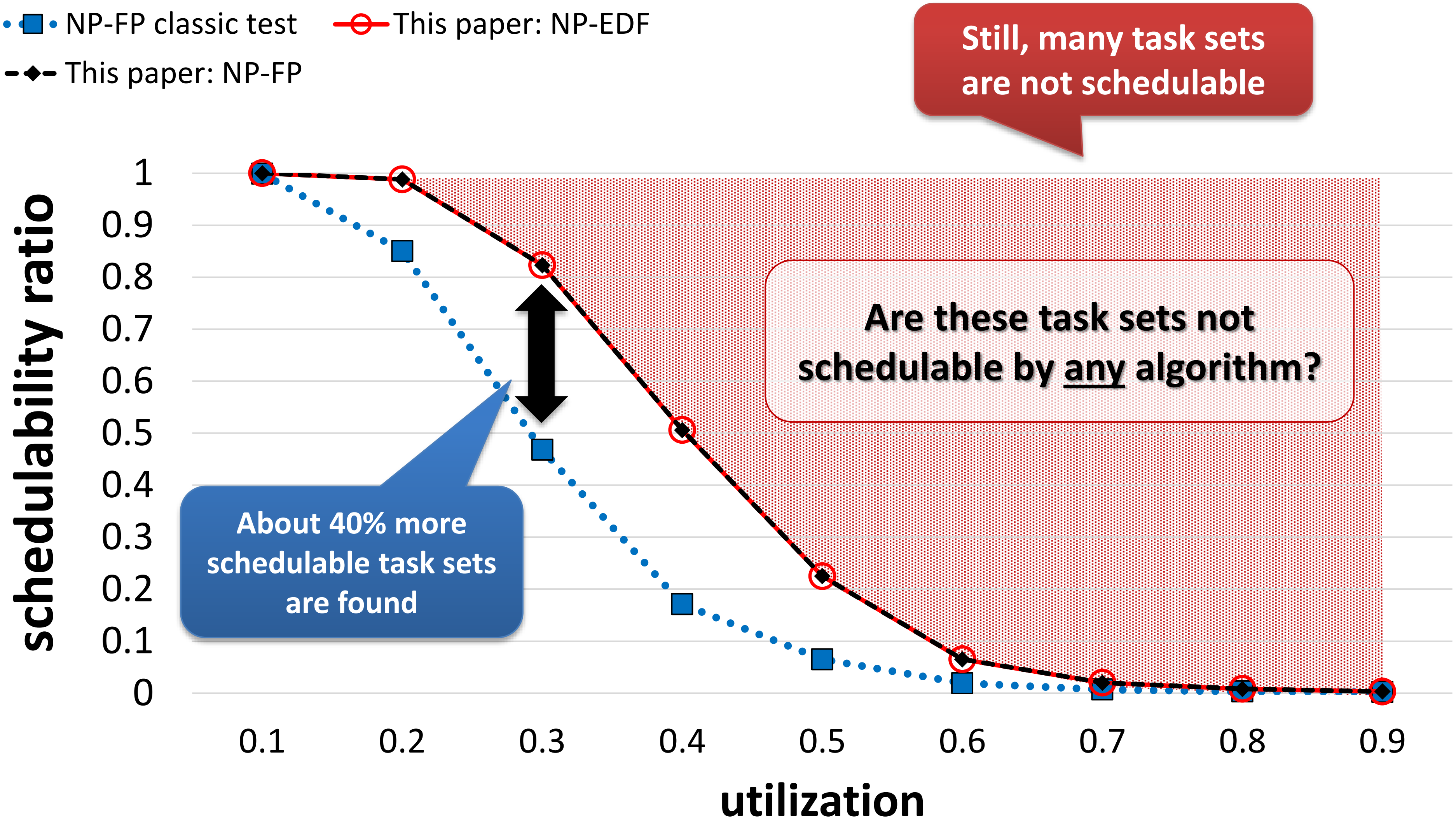
Automotive benchmark, no jitter



Task sets in this experiment have up to 35 tasks and 3500 jobs

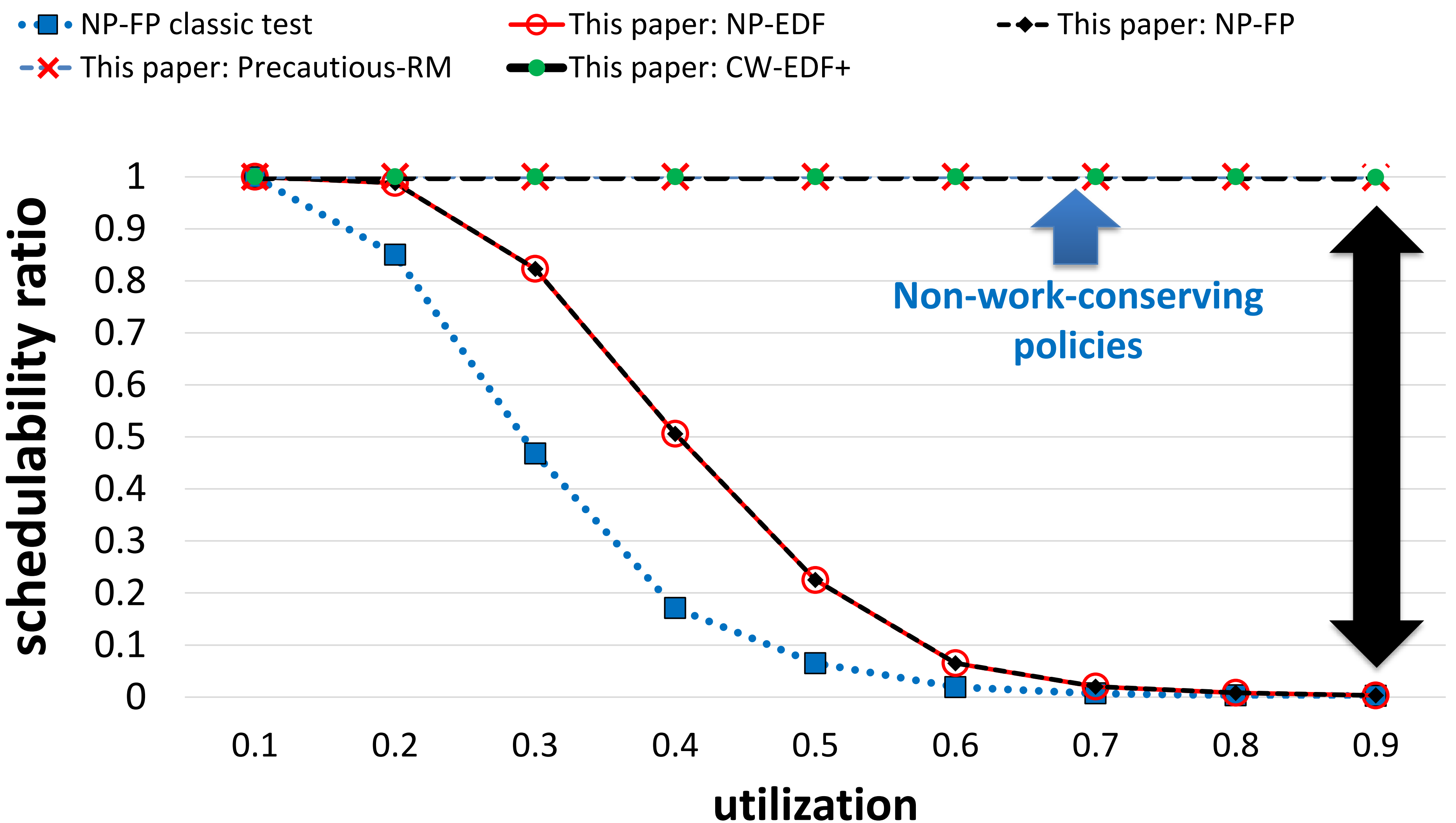
How effective is our schedulability analysis?

Automotive benchmark, no jitter



How effective is our schedulability analysis?

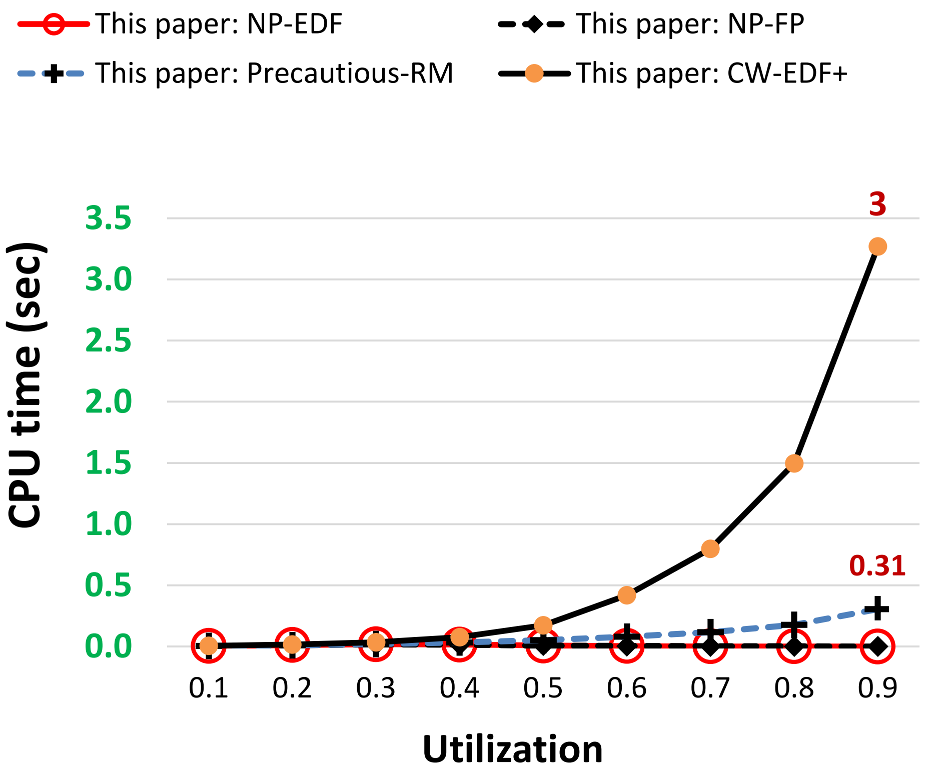
Automotive benchmark, no jitter

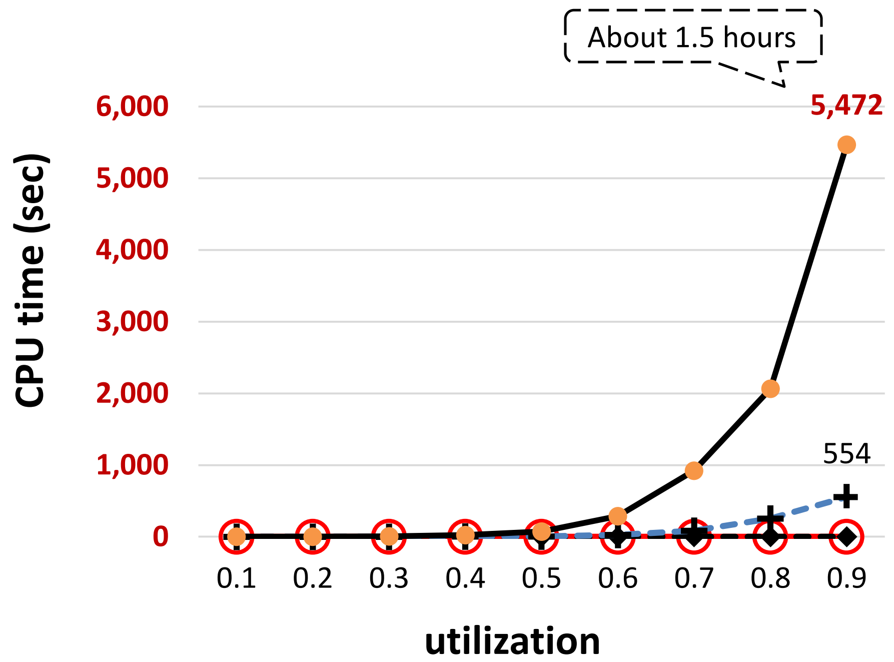


How efficient is our schedulability analysis?

Automotive benchmark

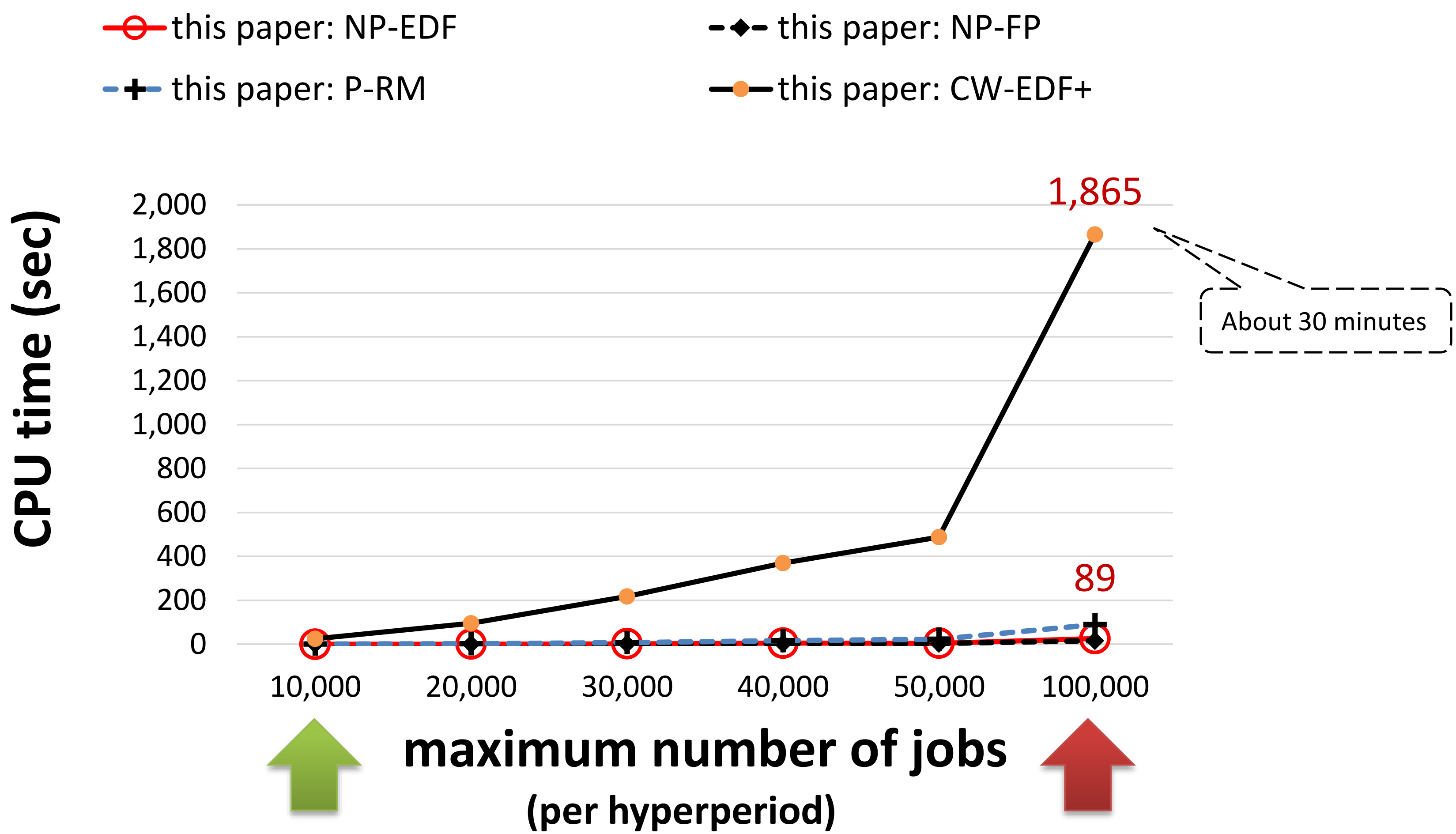
No jitter | **Large jitter**





How efficient is our schedulability analysis?

Synthetic tasks
Small jitter



Agenda

- Main idea: Searching all possible execution scenarios efficiently and accurately
- Constructing the search graph
- Evaluation
- **Conclusion**

Image from http://zworth.net/

Conclusion

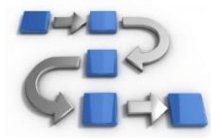
Goal

An efficient, exact, and general schedulability analysis for a wide class of scheduling algorithms

Solution

Constructing a precise abstraction of all possible schedules

Method

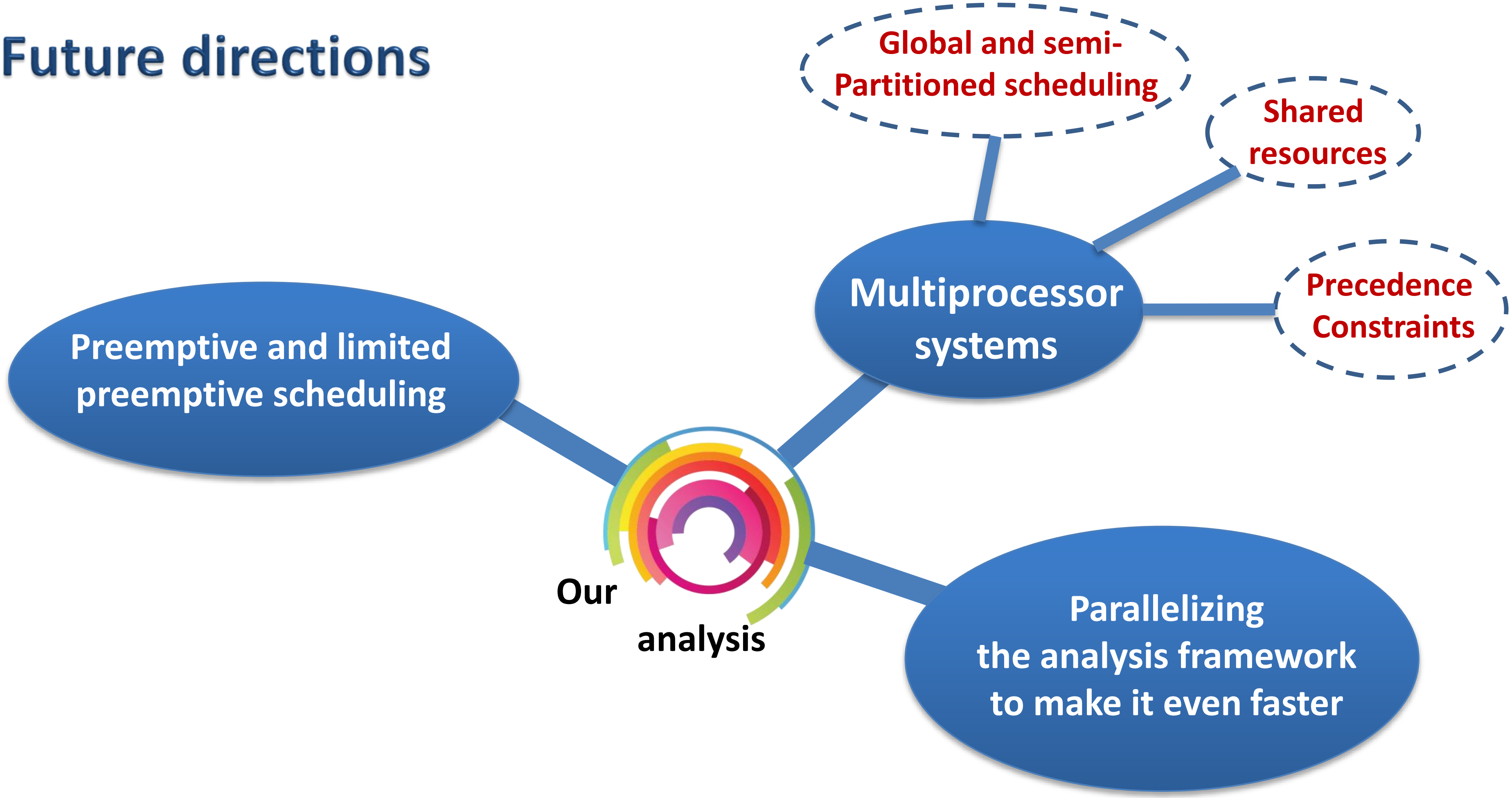


Building a schedule-abstraction graph based on job ordering

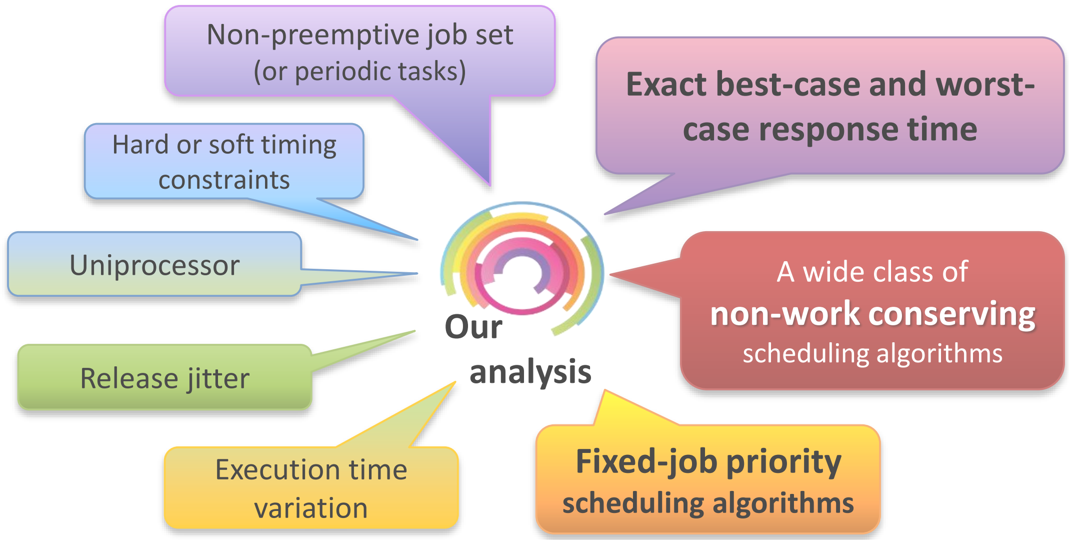
Key idea

An efficient merge technique to defer the state-space explosion

Future directions



MAX PLANCK INSTITUTE
FOR SOFTWARE SYSTEMS



Source code available at
https://people.mpi-sws.org/~bbb/papers/details/rtss17/index.html

Thank you