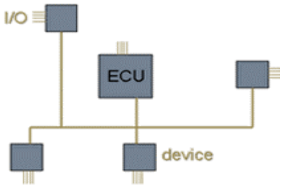
Towards an **Efficient** and **Accurate** Schedulability Analysis for Real-Time Cyber-Physical Systems

Mitra Nasri

Assistant professor
Delft University of Technology

Embracing future challenges

Back then



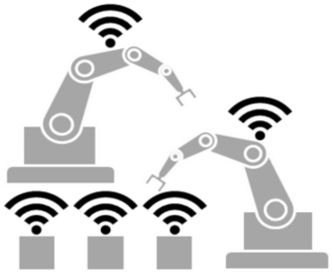
A few computing nodes
and control loops

Simple hardware and
software **architecture**

Liu and Layland task model was a
relevant thing for those systems

Simple, predictable, and easier-to-analyze computing models

Now (and future)



Complex (and usually parallelizable) application workloads running on heterogeneous multi- and many-core platforms

Intensive I/O accesses

Use of hardware accelerators
(GPUs, FPGA, DSP, co-processors, etc.)

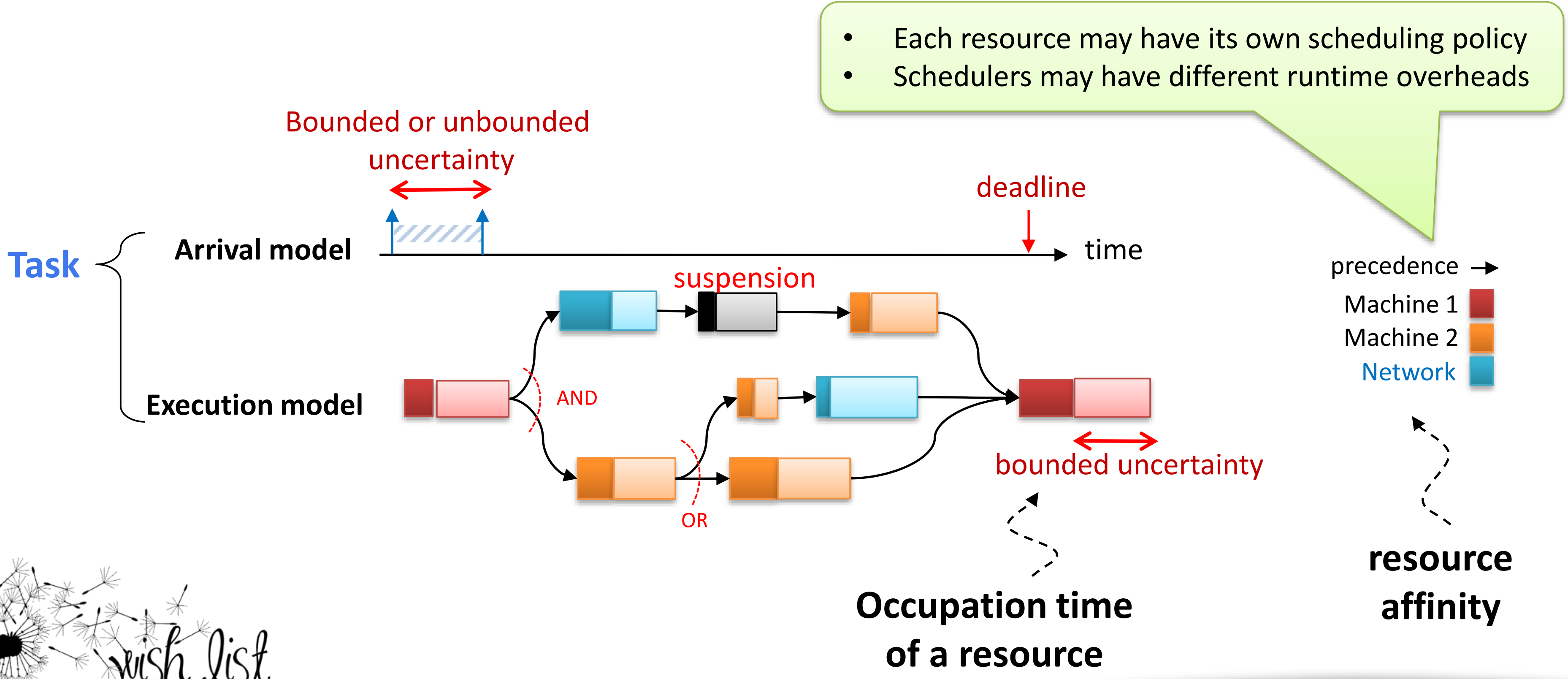
Computation offloading
(to the cloud, edge, etc.)

Complex, less predictable, and harder-to-analyze computing models

A wish list

Obtain the worst-case and best-case <u>response time</u>

Parallel heterogeneous DAG tasks with conditional branches



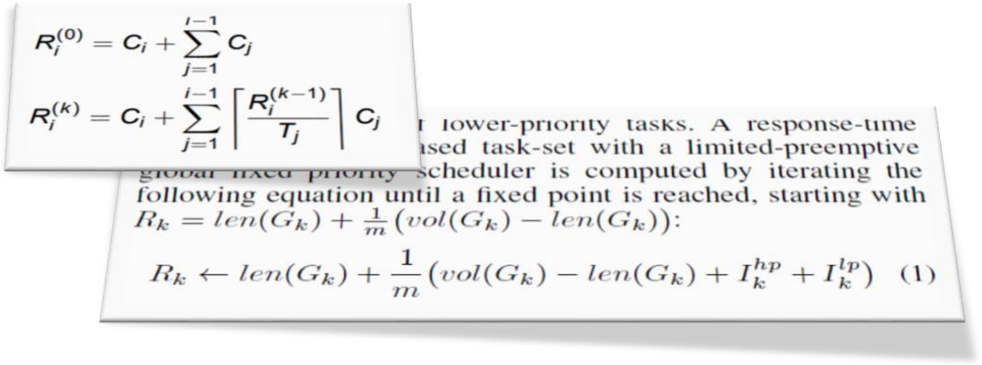
- Each resource may have its own scheduling policy
- Schedulers may have different runtime overheads

State of the art

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

✓
- **Fast**

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



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

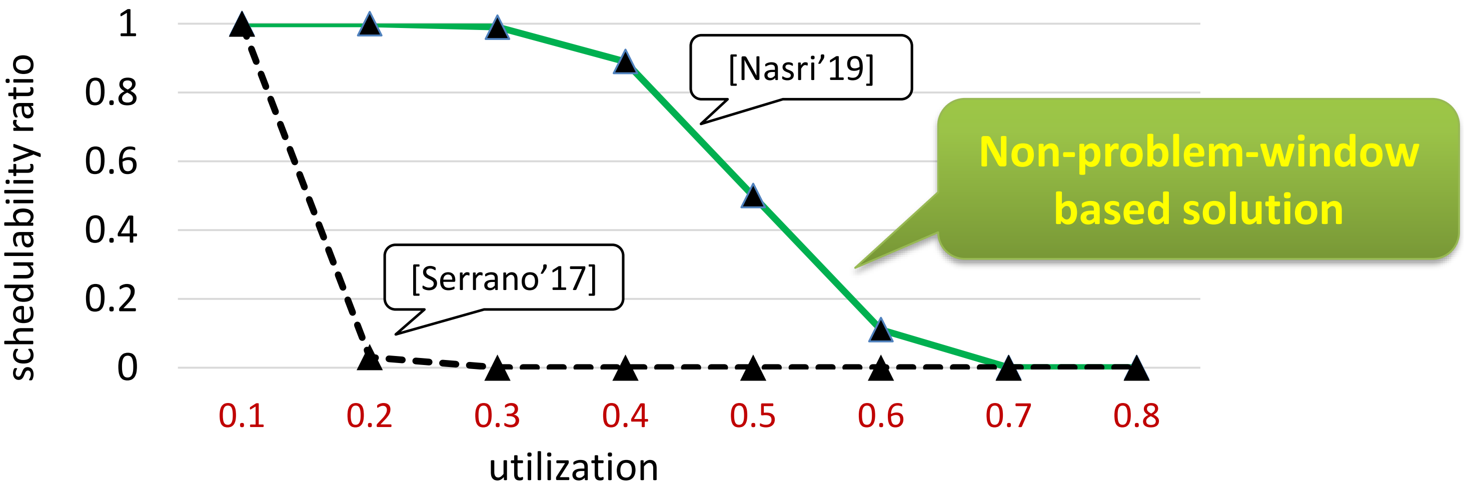
$$R_i^{(k)} = C_i + \sum_{j=1}^{i-1} \left\lceil \frac{R_i^{(k-1)}}{T_j} \right\rceil C_j$$

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

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

Experiment:

10 limited-preemptive parallel DAG tasks scheduled by global FP on 16 cores



Non-problem-window based solution

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.

Mitra Nasri, Geoffrey Nelissen, and Björn B. Brandenburg, "Response-Time Analysis of Limited-Preemptive Parallel DAG Tasks under Global Scheduling", the Euromicro Conference on Real-Time Systems (ECRTS), 2019, pp. 21:1-21:23.

State of the art

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

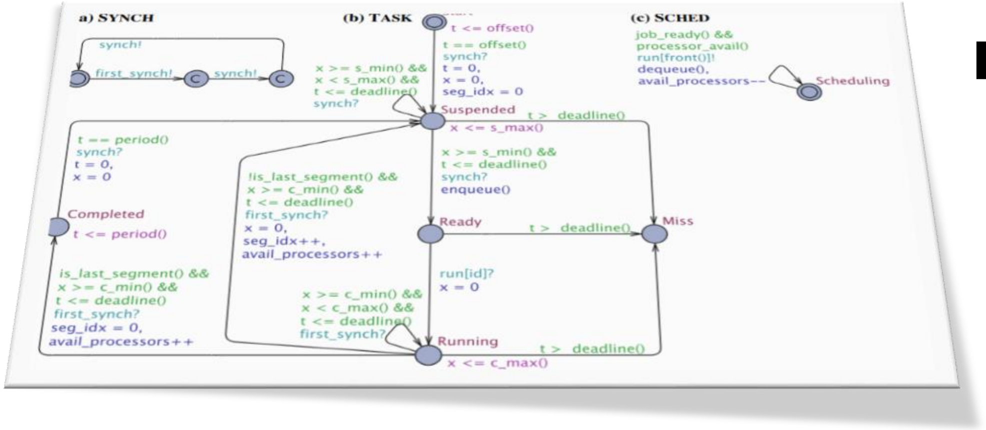
✓
- Fast

✗
- Pessimistic
- Hard to extend

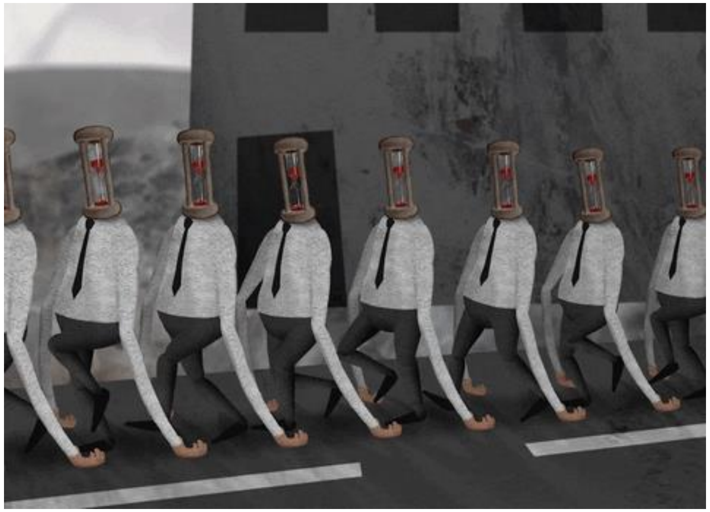
Exact tests in generic formal verification tools (e.g., UPPAAL)

✓
- Accurate
- Easy to extend

✗
- Not scalable



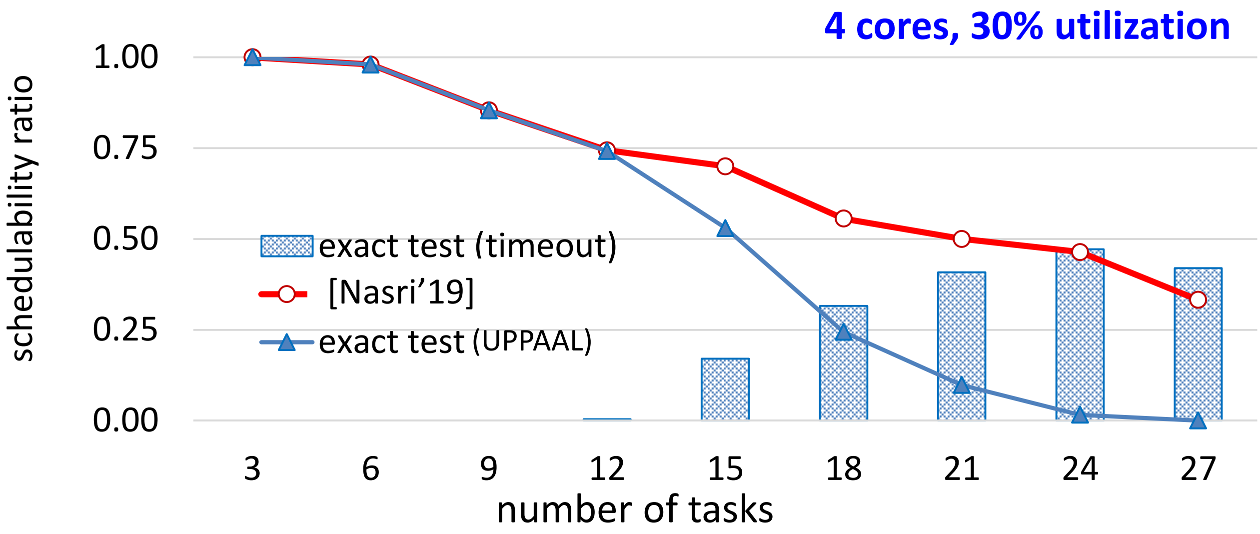
The "tool" does all the labor
(to find the worst case)

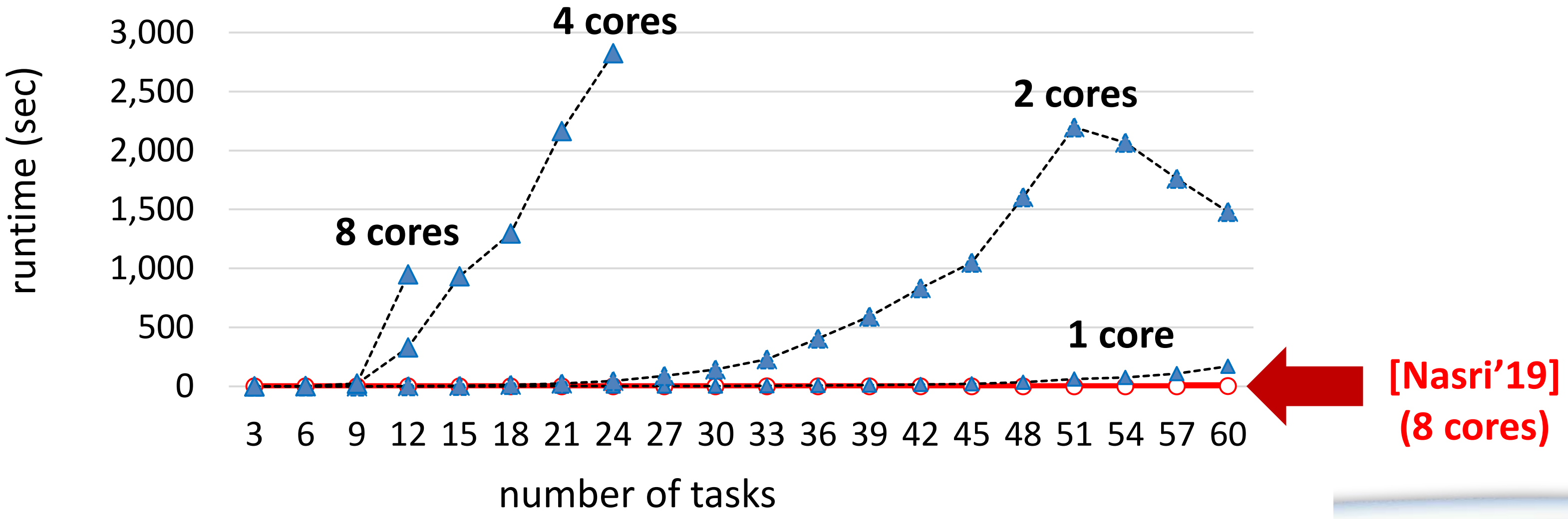


Generic verification tools are very slow and do not scale to reasonable problem sizes

Results from formal verification-based analyses

Sequential non-preemptive periodic tasks (scheduled by global FP)





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

Our new line of work

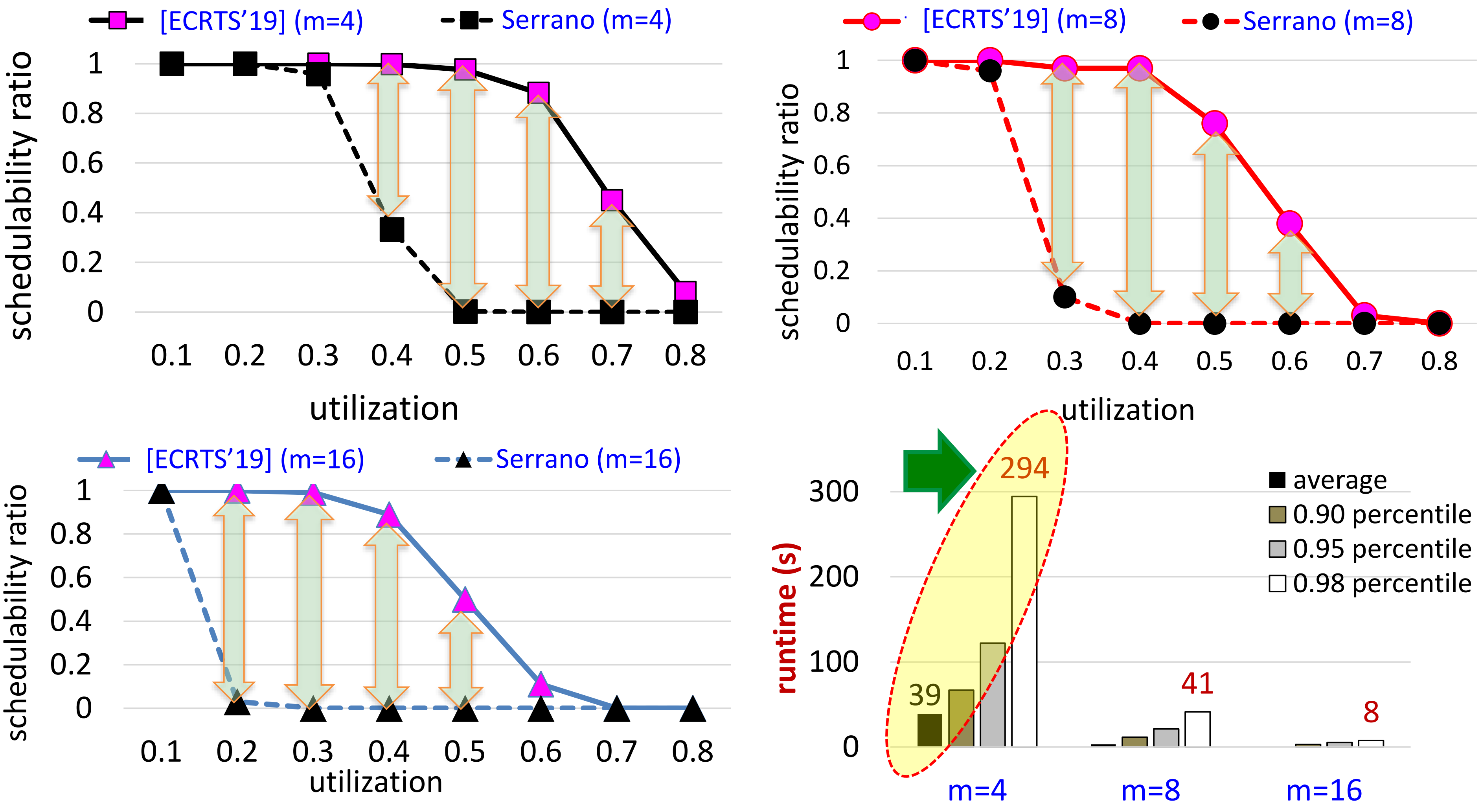
Response-time analysis using schedule abstraction

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

Idea: efficiently explore the space of all possible schedules

Some results on 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.

Response-time analysis using schedule abstraction

An example: the problem of global non-preemptive scheduling

Obtain the **worst-case** and **best-case response time**

Workload model

Non-preemptive job sets

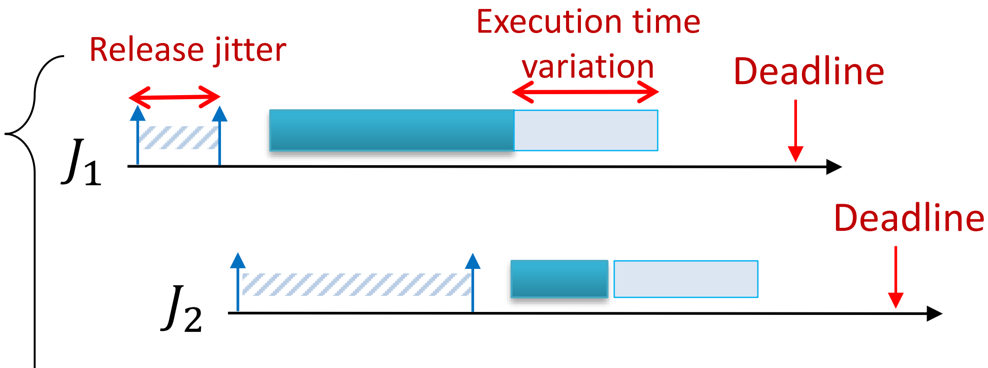
Platform model

Multicore (identical cores)

Scheduler model

Global job-level fixed-priority (JLFP)

Job model



…

The job set is provided for an **observation window**, e.g., a hyperperiod.

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

Solution 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

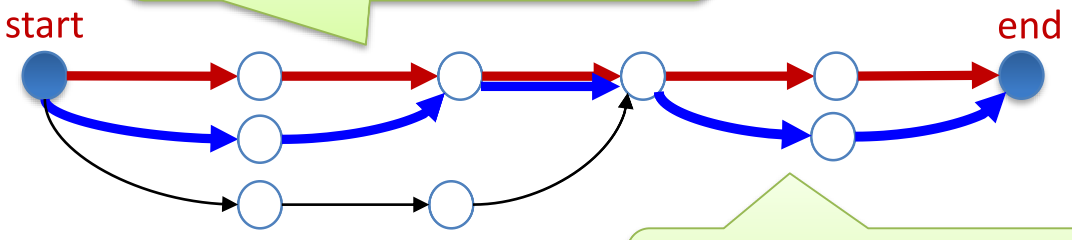
Solution

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

A path represents a set of similar schedules

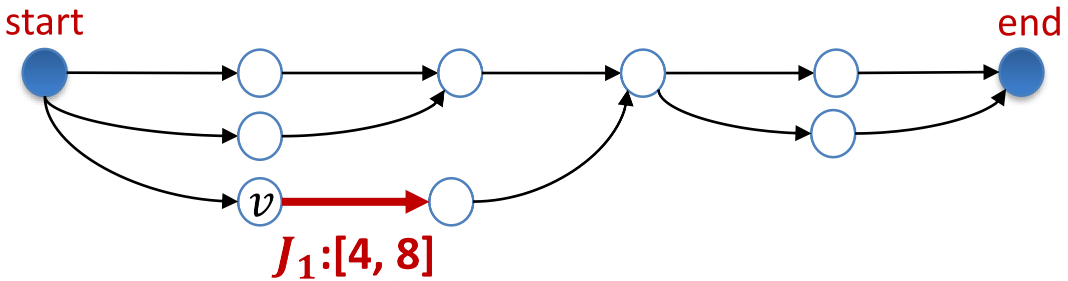


Different paths have different job orders

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



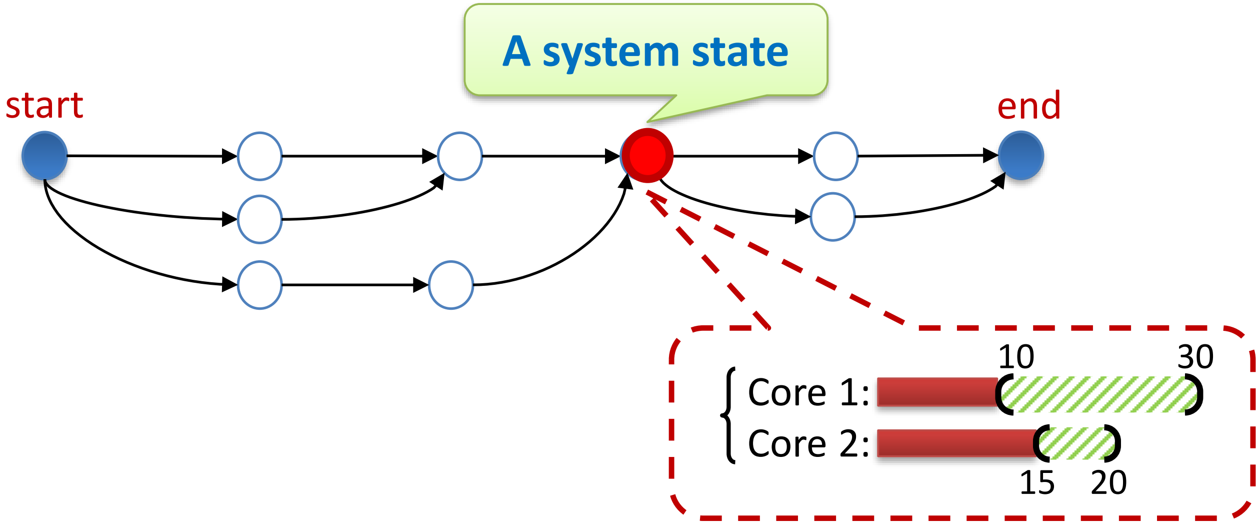
Earliest and latest finish time of J_1 when it is dispatched after state v

Response-time analysis using schedule-abstraction graphs

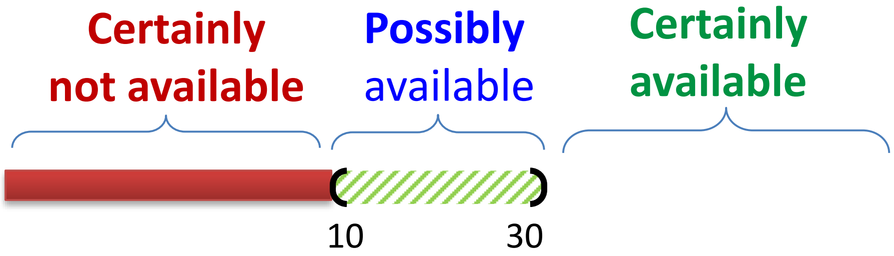
A **path** aggregates all schedules with the same job ordering

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:



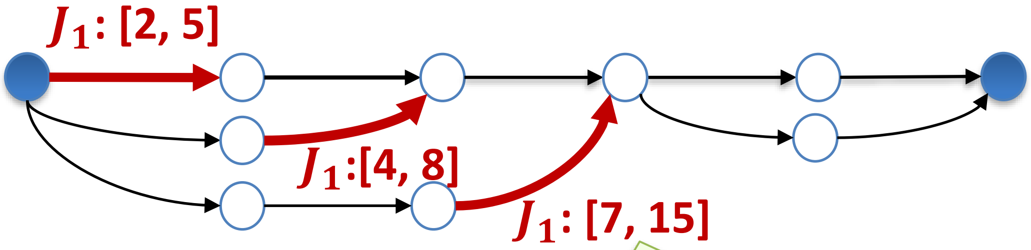
Response-time analysis using schedule-abstraction graphs

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:



Earliest and latest completion times of the job in the path

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

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

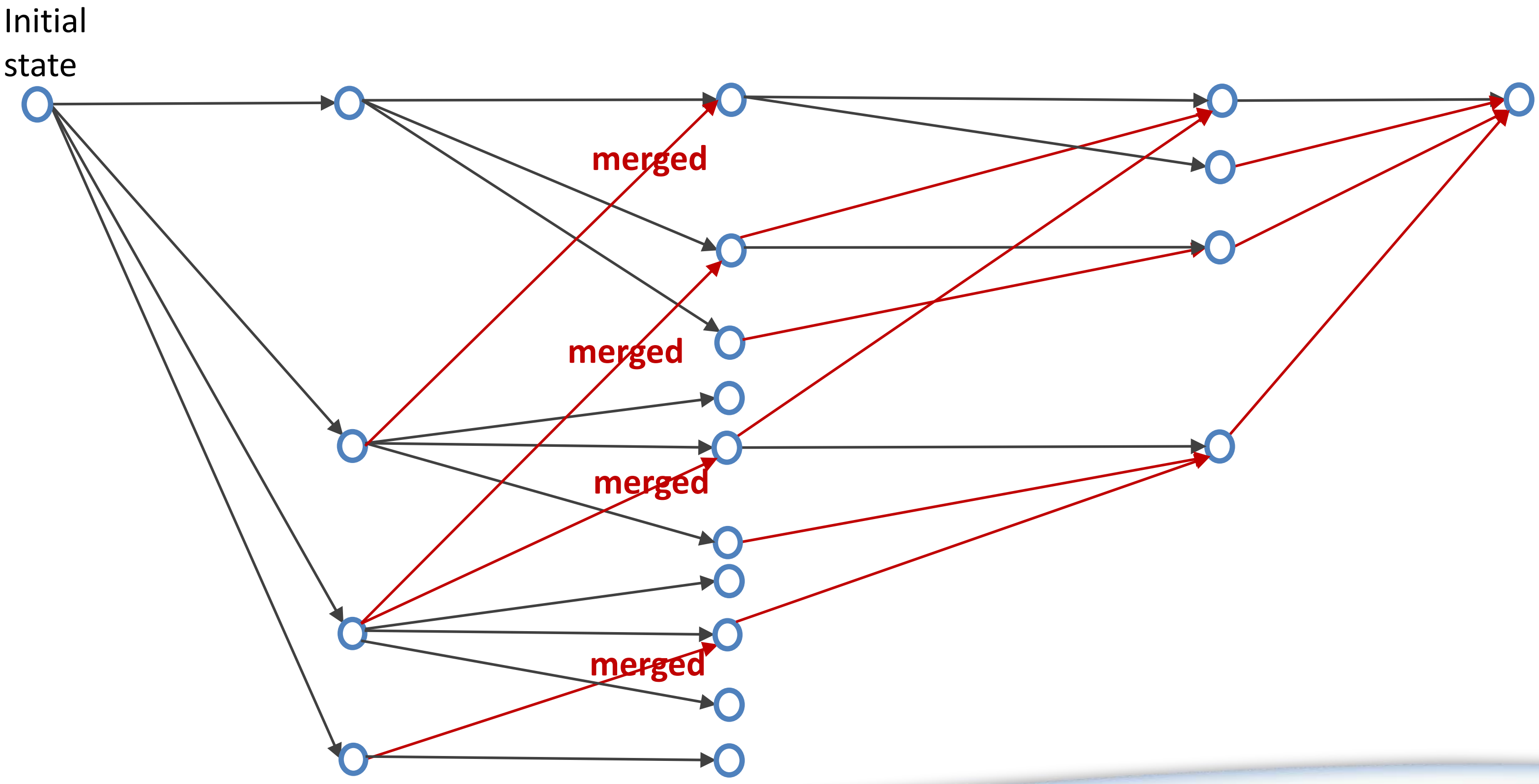
Building the schedule-abstraction graph

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)

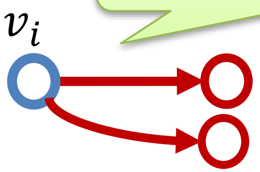
System is idle and
no job has been scheduled

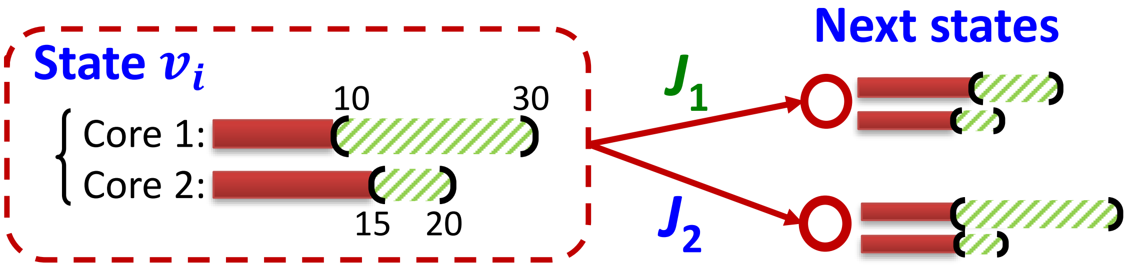


Building the schedule-abstraction graph

Expanding a vertex:
(**reasoning on uncertainty intervals**)

Expansion rules imply the scheduling policy



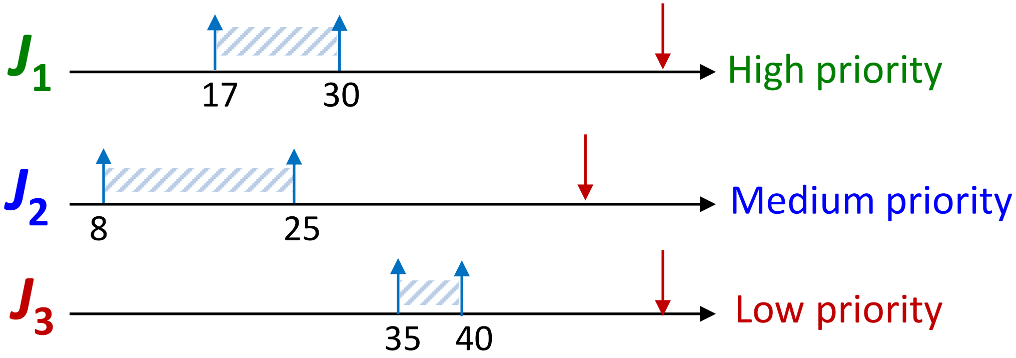


State v_i

Core 1:
Core 2:

Next states

Available jobs
(at the state)



High priority

Medium priority

Low priority

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

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

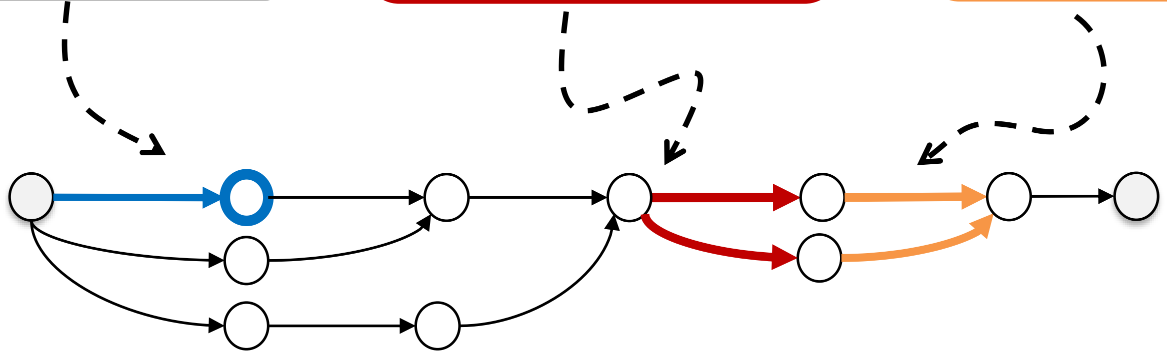
How to create
new states?

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”

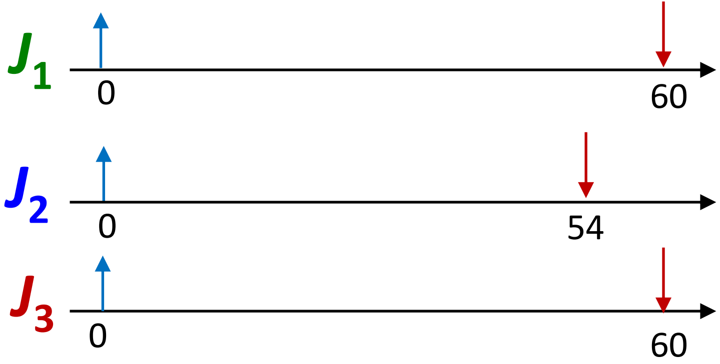
Challenges

https://www.globallanguageservices.co.uk/30-days-of-language-challenges/

Challenge: handling release jitter

No jitter

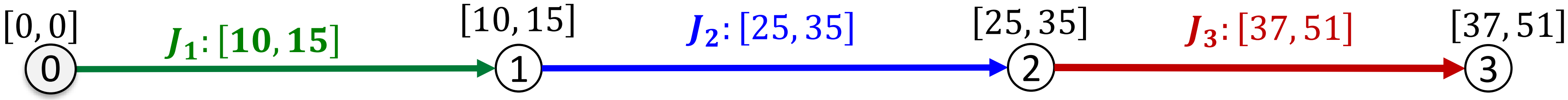
uniprocessor



High priority $C_1 \in [10, 15]$

Medium priority $C_2 \in [15, 20]$

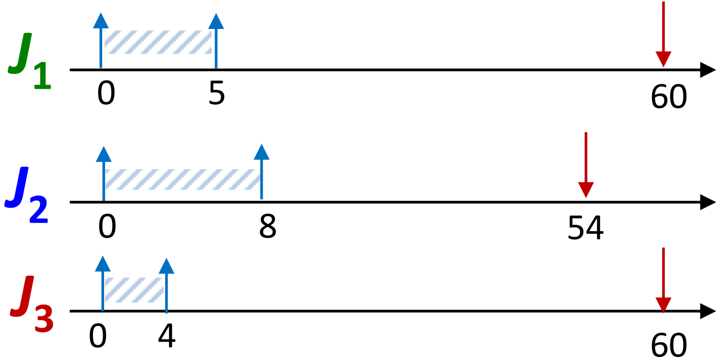
Low priority $C_3 \in [12, 16]$



Challenge: handling release jitter

Small jitter

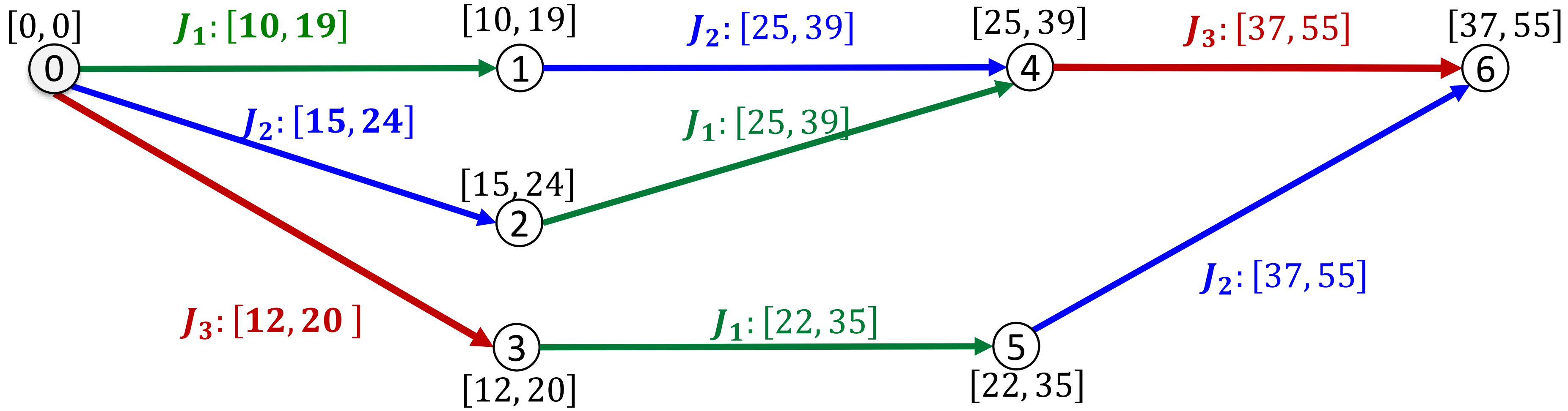
uniprocessor



High priority $C_1 \in [10, 15]$

Medium priority $C_2 \in [15, 20]$

Low priority $C_3 \in [12, 16]$

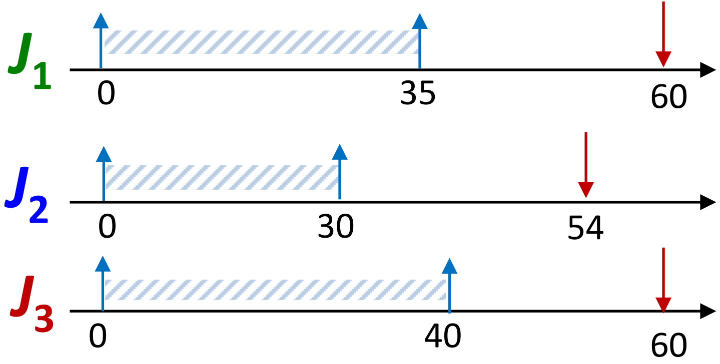


Challenge: handling release jitter

Large release jitter (or sporadic release) may result in a combinatorial state space

uniprocessor

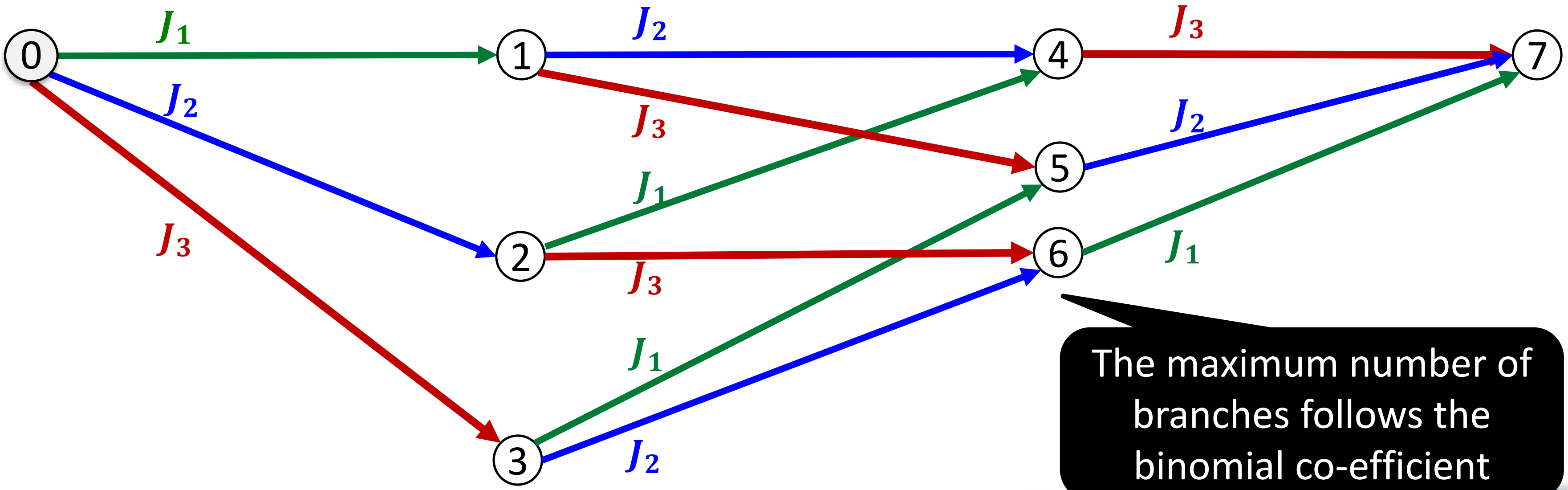
Larger jitter



High priority $C_1 \in [10, 15]$

Medium priority $C_2 \in [15, 20]$

Low priority $C_3 \in [12, 16]$



The maximum number of branches follows the binomial co-efficient

Challenge: handling release jitter

Large release jitter (or sporadic release) may result in a combinatorial state space

Potential solutions

Partial-order reduction

Avoid exploring paths that do not contribute to the worst-case scenario.

Use approximation to derive the worst-case completion time of the remaining jobs in that path

Challenges: handling release jitter

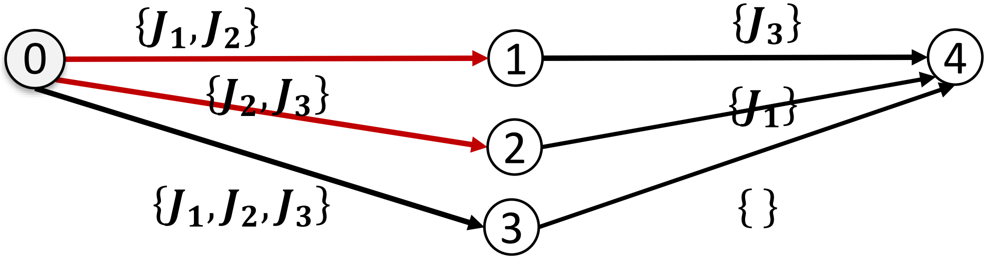
Large release jitter may result in a combinatorial state space

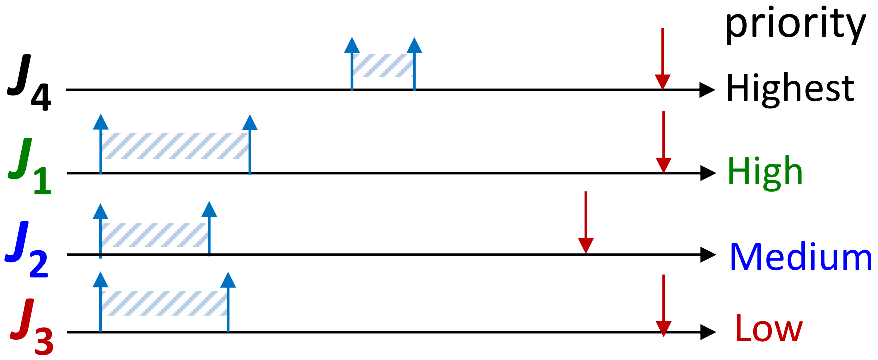
Potential solutions

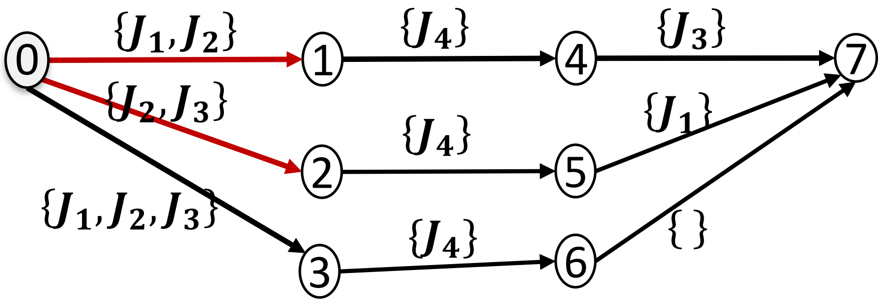
Partial-order reduction

Batch processing
(rather than processing a single job at a time)

Derive expansion rules for a set of jobs







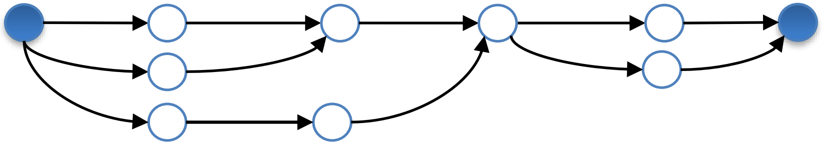
Challenges: handling release jitter

Large release jitter may result in a combinatorial state space

Potential solutions

- Partial-order reduction
- Batch processing
- **Using memorization (to avoid exploring previously seen patterns)**

What else?



Thank you.