Schedulability analysis of limited-preemptive moldable gang tasks

Joan Marcè i Igual

TU Delft

Daily Supervisor Geoffrey Nelissen

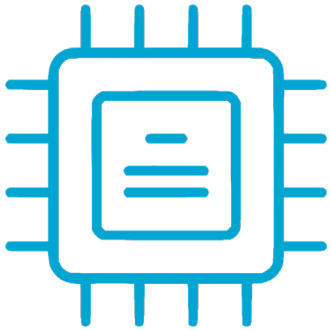
Co-supervisor Mitra Nasri

27th of August, 2020

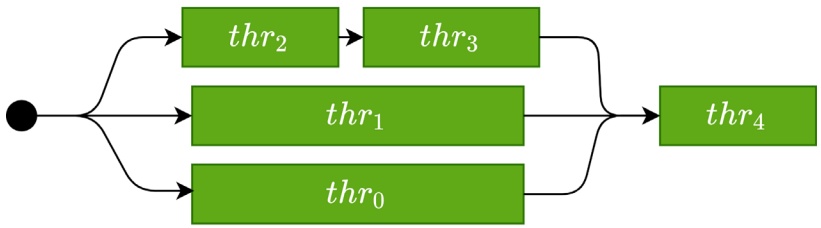
Real-time systems

- Systems of which correctness does depends not only on **logical** results but also on **timing constraints**
- Higher demand for computation resources
  - Multicore systems
  - GPUs
  - FPGAs



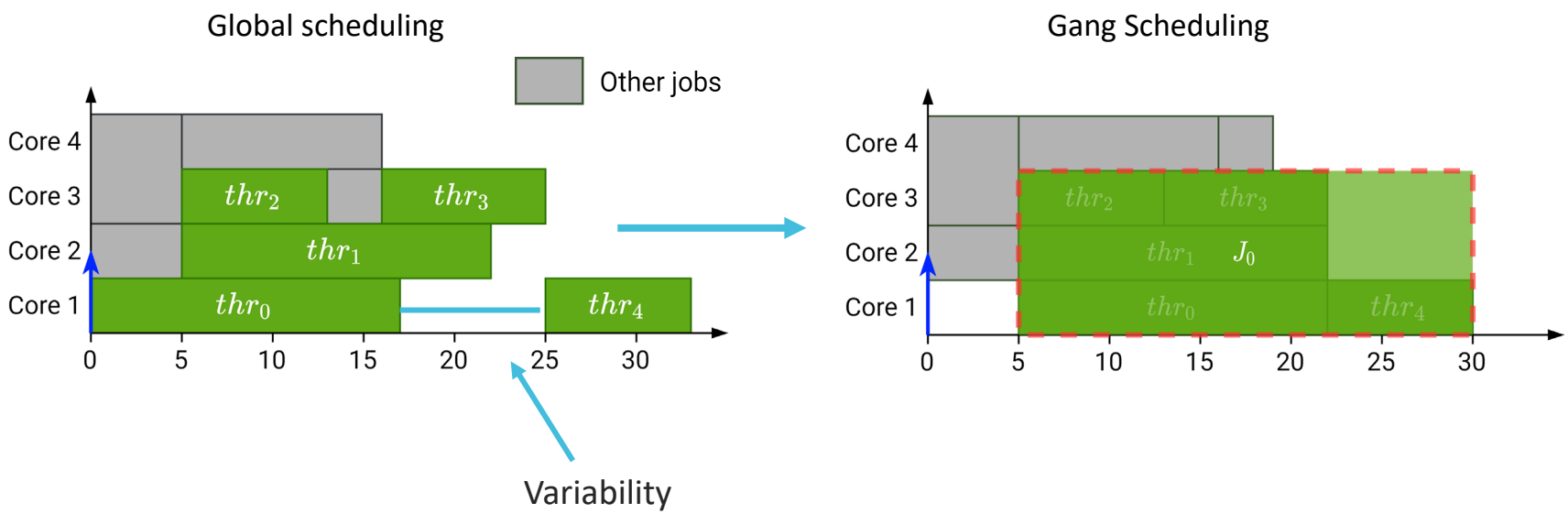


What is gang?



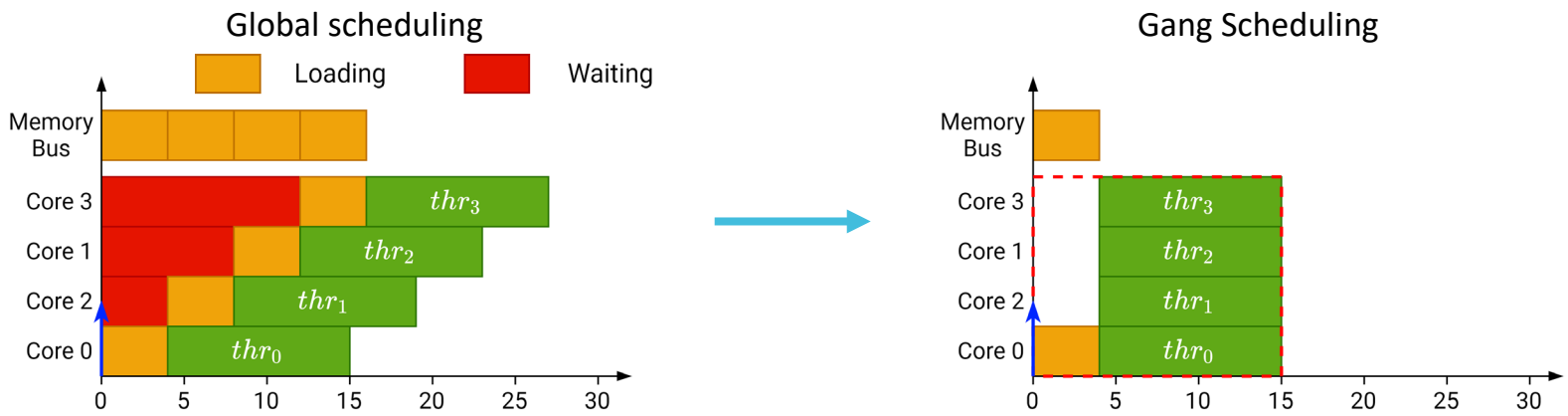
Parallel threads together as a “gang”

Execution does not start until there are enough cores



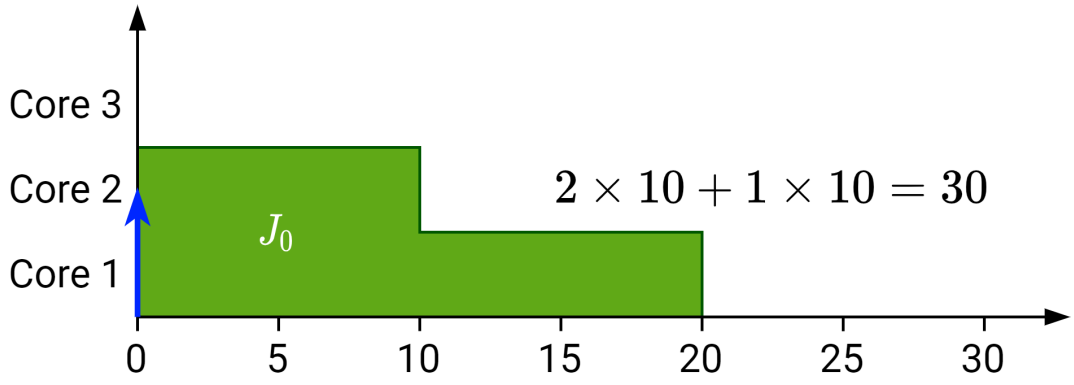
Why gang?

- More efficient synchronization
- Reduces variability in the execution
- Avoids overhead when loading initial data
- Shows its full potential when executed non-preemptively

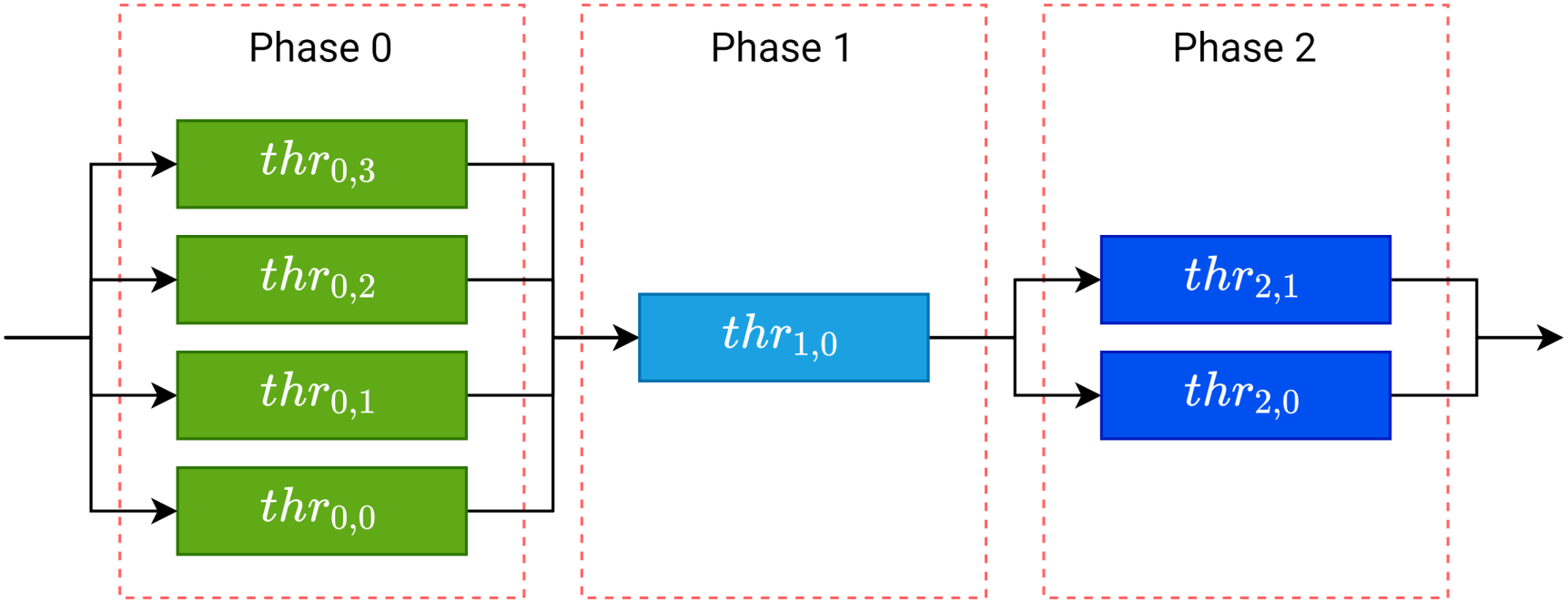


Types of gang

- **Rigid**: number of cores set by programmer → Blocking
- **Moldable**: number of cores assigned when job is dispatched → Flexibility
- **Malleable**: number of cores can change during runtime → Hard to implement

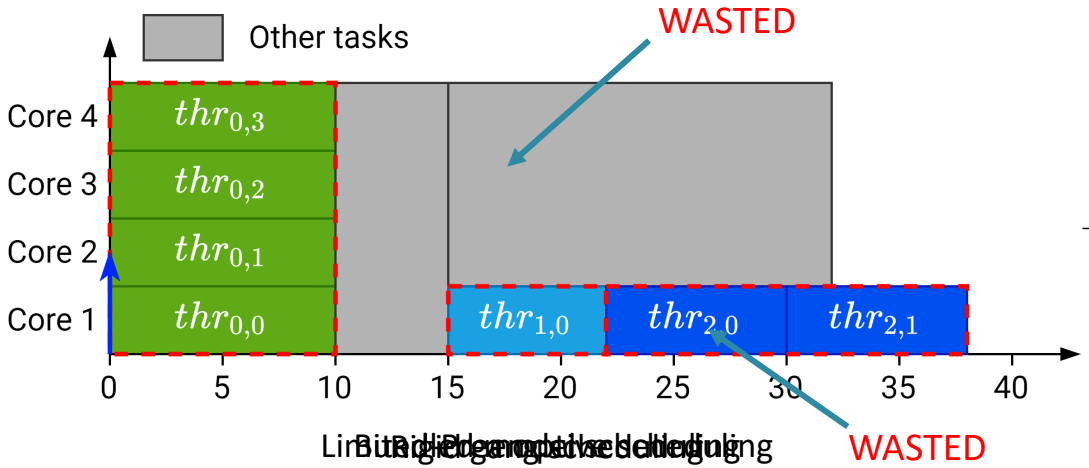


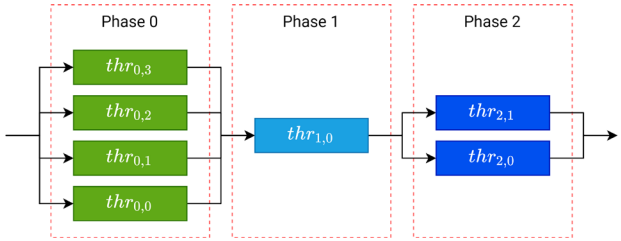
Bundled scheduling[1] vs limited-preemptive



Bundled scheduling[1] vs limited-preemptive

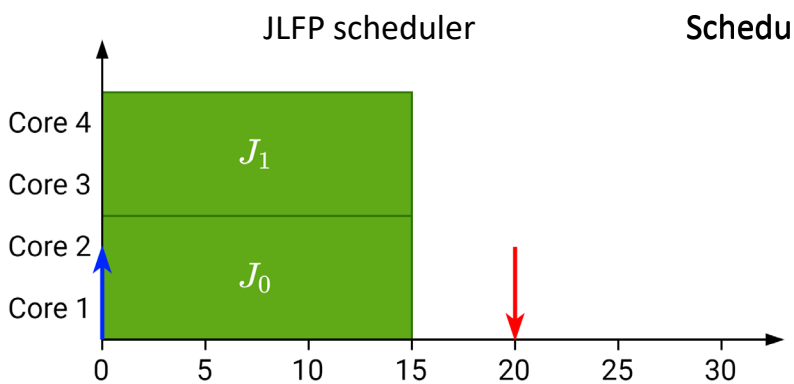
- Rigid gang reserves the whole block
- Bundled creates rigid blocks with dependencies
- Limited-Preemptive creates moldable blocks with dependencies





Job-level fixed-priority scheduling (JLFP) for gang

- Based on global JLFP scheduler
- Assigns maximum possible cores to a job
- We will solve this issue



Scheduling: J_0

	Priority	Min cores	Max cores	Deadline	Execution time
J_0	High	1	3	20	30, 15, 10
J_1	Low	2	2	20	15

Previous work

Introduced in high-performance computing in 1982[1]

Preemptive solutions

Schedulability tests

- Job-level fixed-priority[2]
- Earliest deadline first[3]

Schedulers

- Optimal for rigid gang (DP-Fair)[4]
- Moldable gang[5]

Non-preemptive solutions

Schedulability tests

- Earliest deadline first for rigid gang[6]

[1]Ousterhout, 1982
[2]Goossens et al., 2010
[3]Kato et al., 2009
[4]Goossens et al., 2016
[5]Berten et al., 2011
[6]Dong et al. 2019

Our work

Project goals

1. Design an accurate schedulability analysis for limited-preemptive moldable gang tasks

2. Evaluate the impact of the level of parallelism assigned to a job

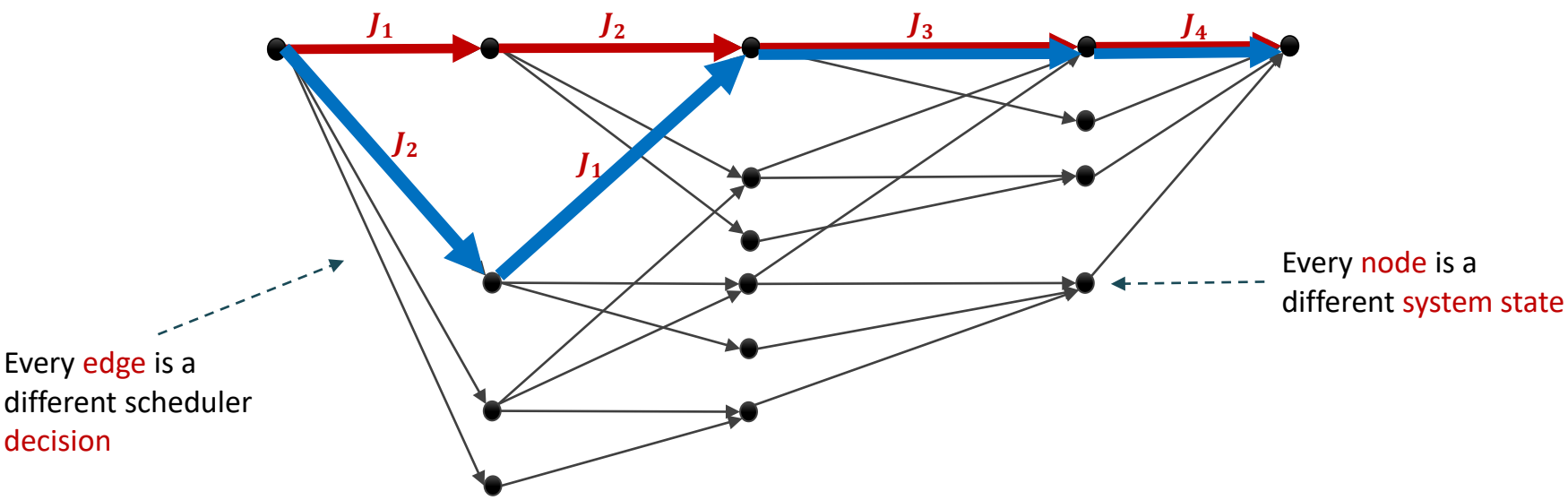
3. Propose a new scheduling algorithm to improve the schedulability of limited-preemptive moldable gang tasks

Agenda

- Gang schedulability analysis
- New scheduling policy

Schedule abstraction graph

- It is a technique that allows:
  - Search for all possible schedules
  - Aggregate “similar” schedules

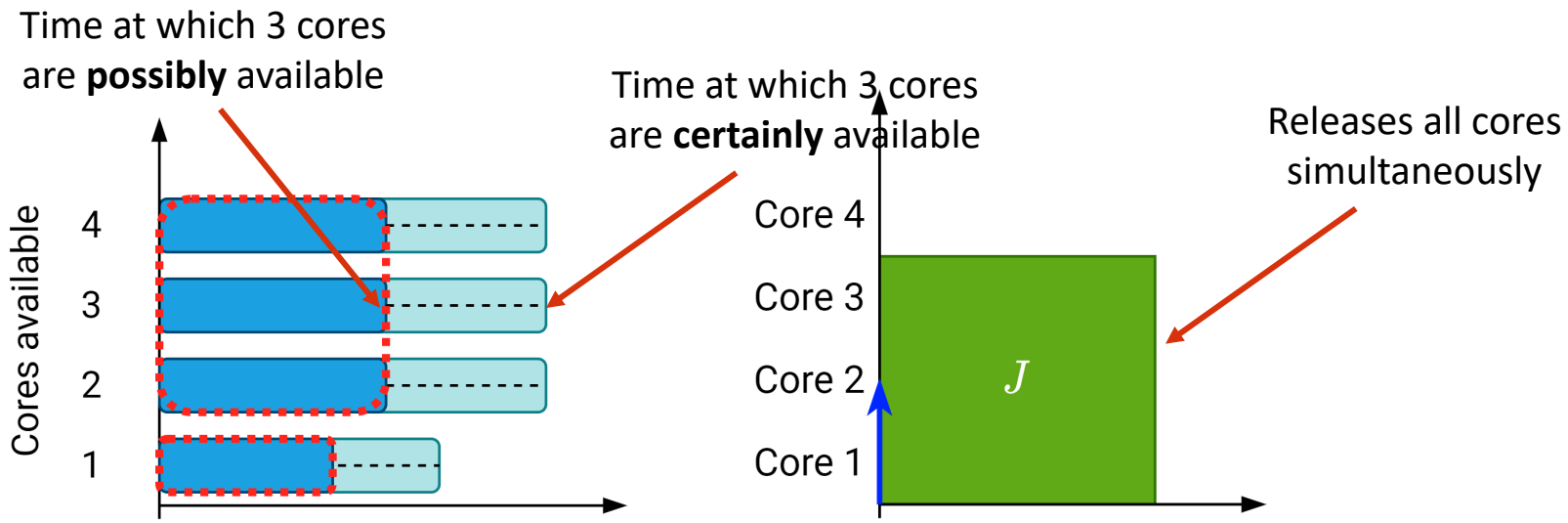


Using SAG for gang scheduling

- Introduce new system state representation
- Introduce new expansion rules
- Introduce new merge rules

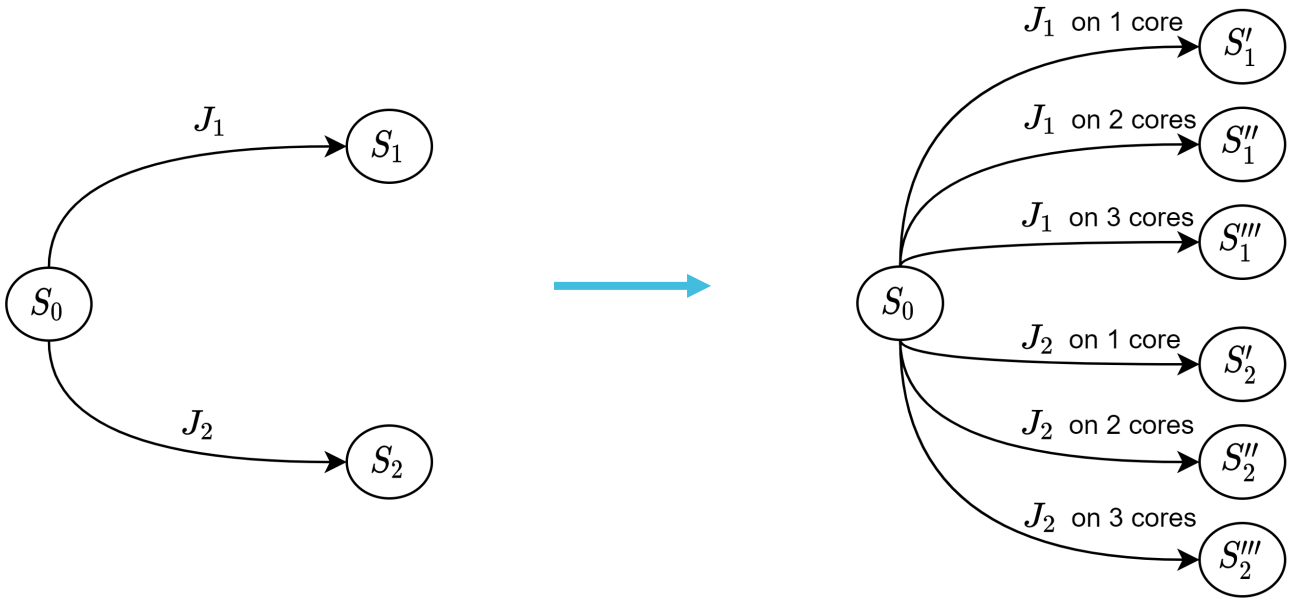
New system state representation

- Availabilities of cores → Know how many cores we have at time t
- Simultaneous groups of cores → Multiple cores released simultaneously
- Set of certainly running jobs → Jobs waiting for their predecessors



New SAG expansion rules

- Previously a state was created for every dispatchable job
- Now a state is created for every job and possible number of cores

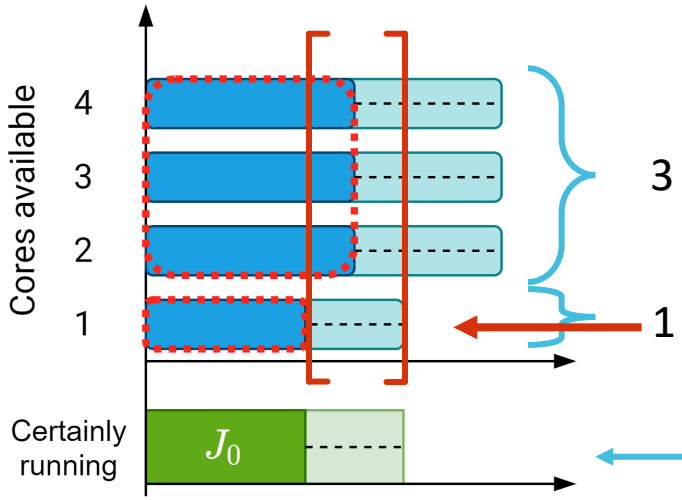


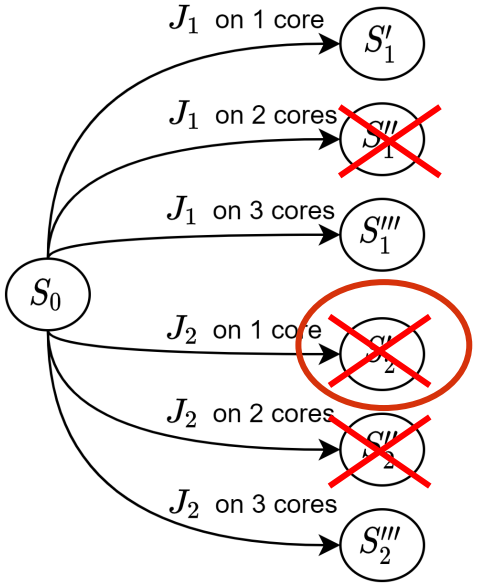
New SAG expansion rules

- Exploring more states is
  - Sound (does not accept unschedulable task sets)
  - Slower and more pessimistic

Try to minimize number of states that represent an impossible scenario

- Purge states by checking that
  - ✔ Enough cores available
  - ✔ More cores **not** available
  - ✔ Precedence constraints are respected





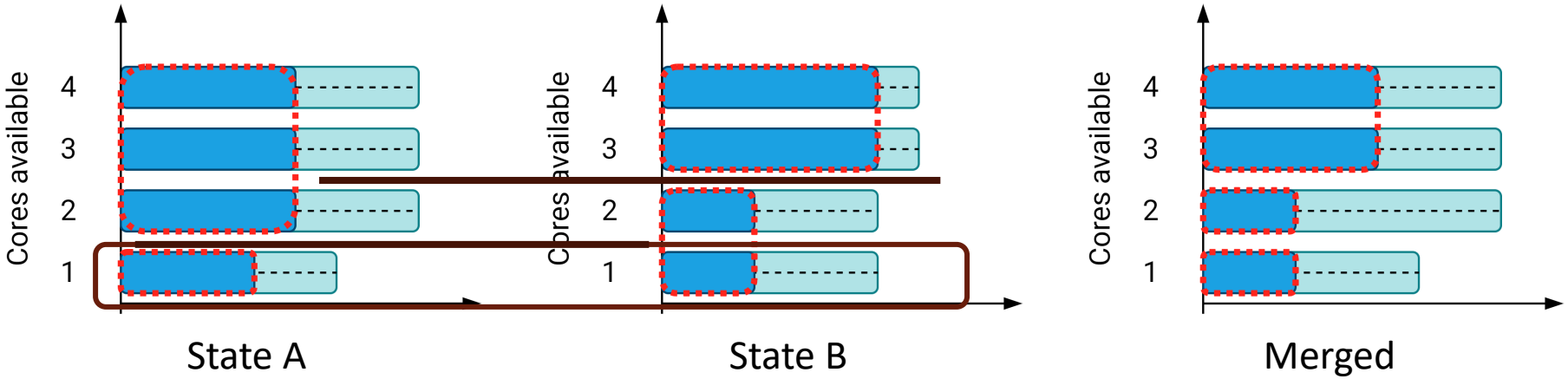
J_1 has higher priority

J_0 is predecessor of J_1

New SAG merge rules

- We have to merge
  - Availability times → Extend the intervals
  - Groups of cores → Break the groups into same size
  - Set of certainly running jobs → Keep only jobs running in both states

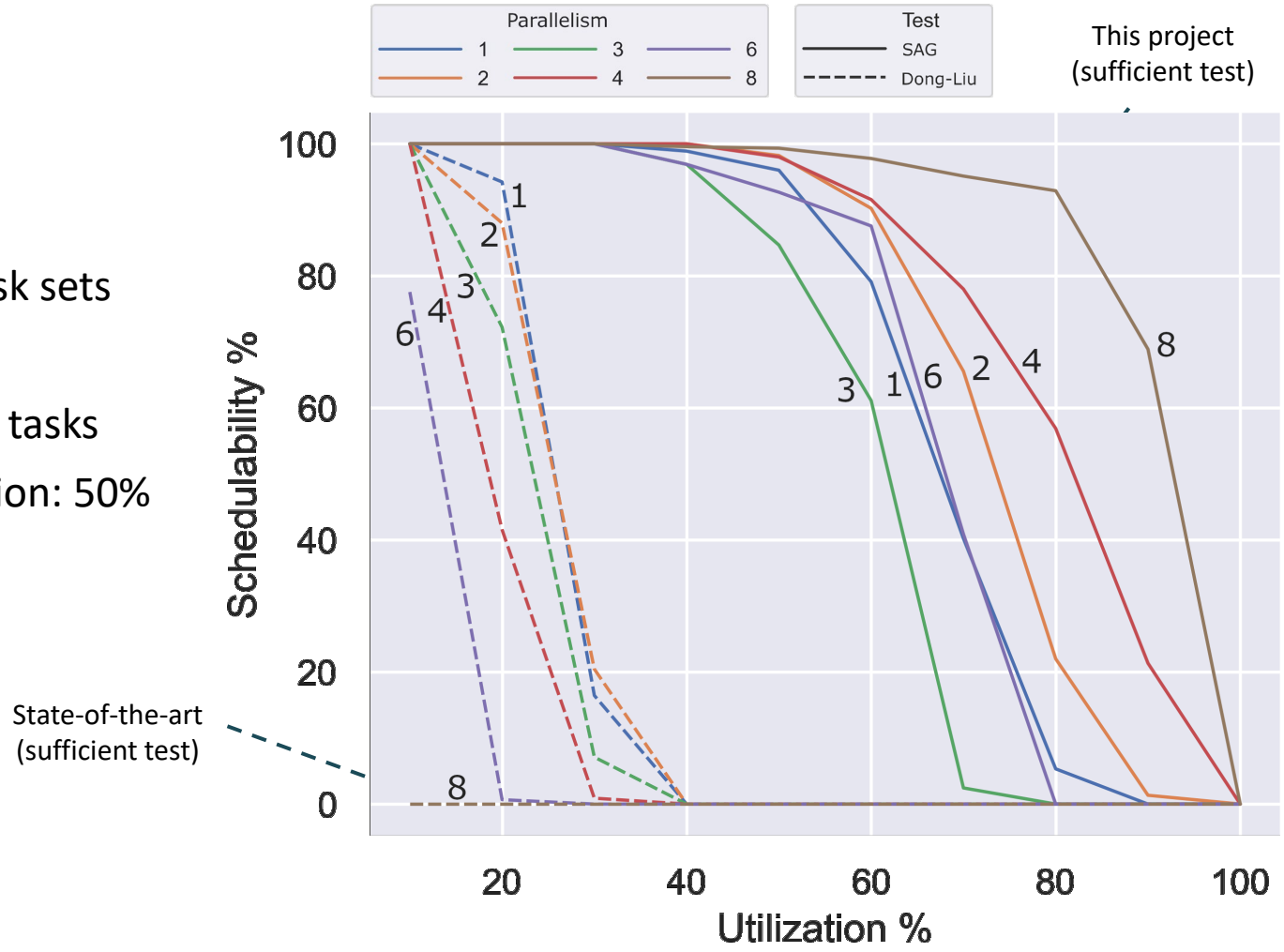
Reachable states from state A and B should also be reachable from the merged state



Our analysis results

Randomly generated task sets

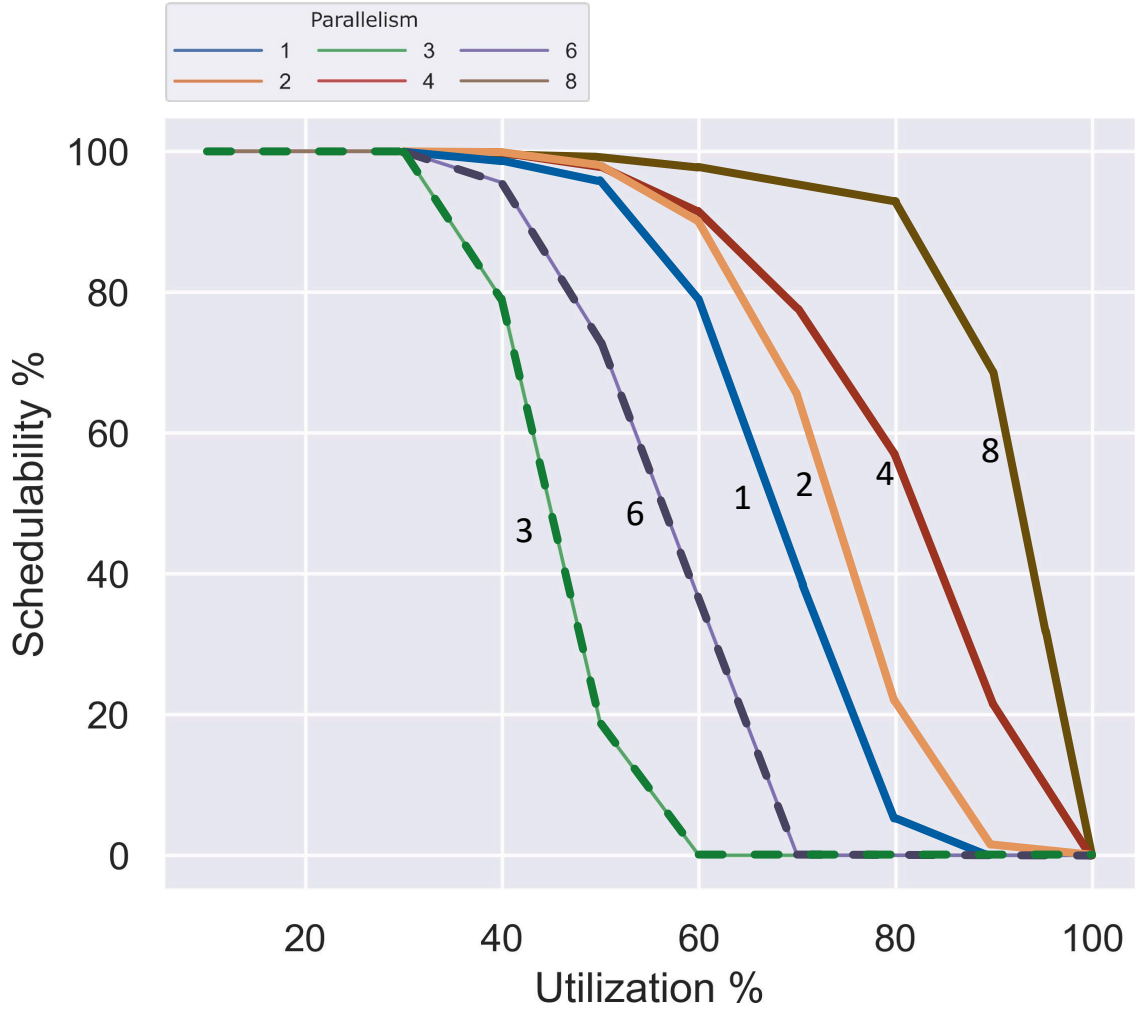
- System cores: 8
- System tasks: 20 rigid tasks
- Execution time variation: 50%
- Implicit deadlines
- Rate monotonic



Our analysis results

Randomly generated task sets

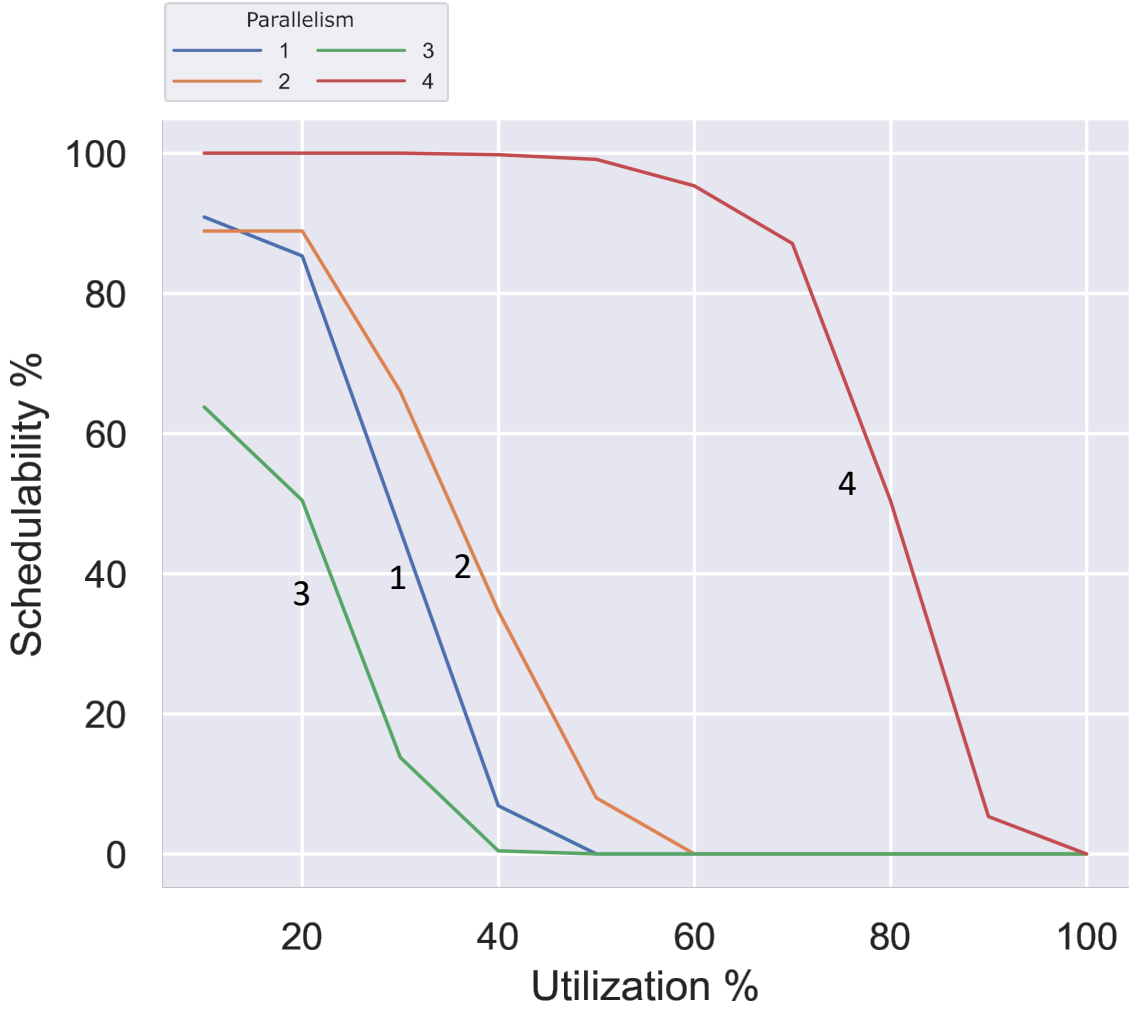
- System cores: 8
- System tasks: 20 moldable tasks
- Execution time variation: 50%
- Implicit deadlines
- Rate monotonic



Our analysis results

Randomly generated task sets

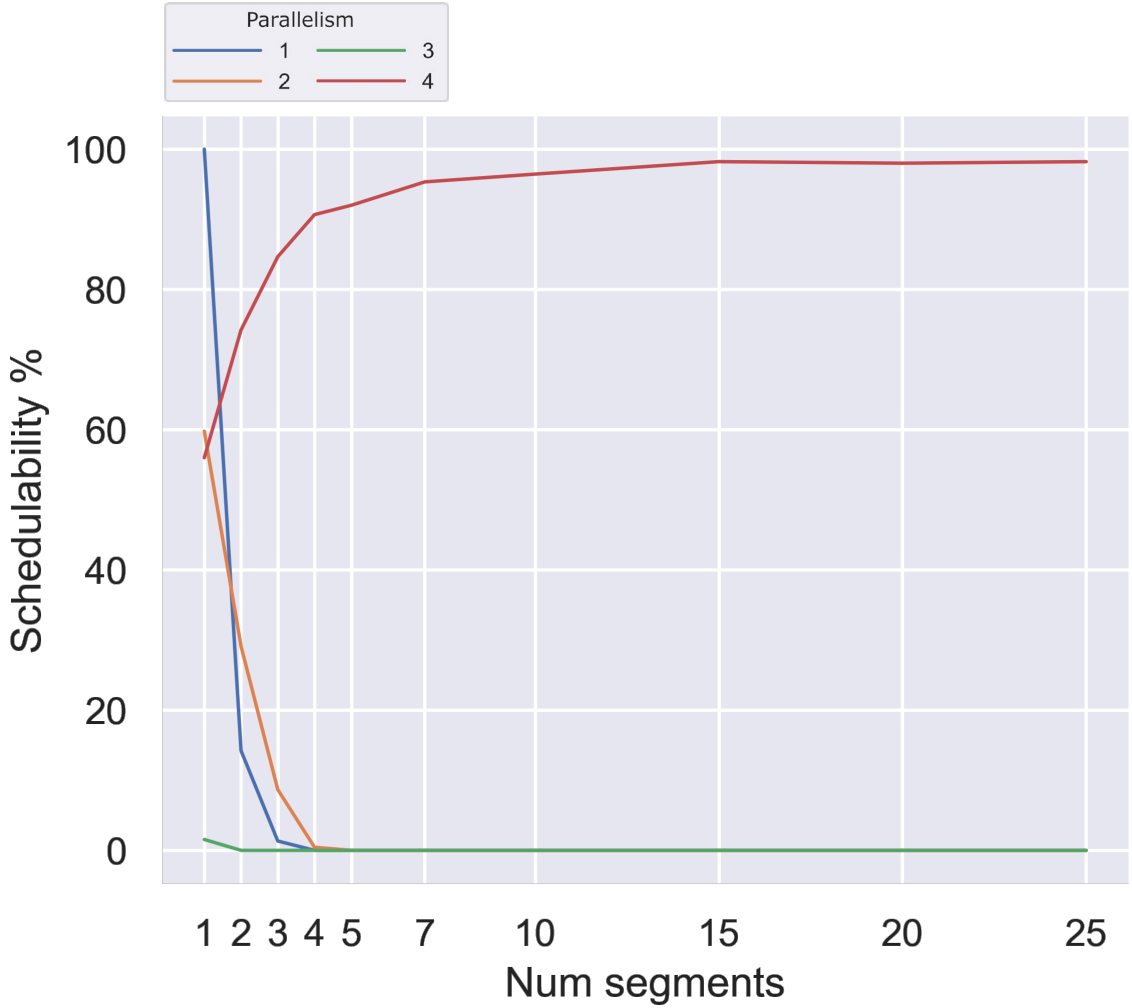
- System cores: 4
- System tasks:
  - 4 moldable tasks
  - 5 segments per task
- Execution time variation: 50%
- Implicit deadlines
- Rate monotonic



Our analysis results

Randomly generated task sets

- System cores: 4
- System tasks: 4 moldable tasks
- Execution time variation: 50%
- System utilization: 70%

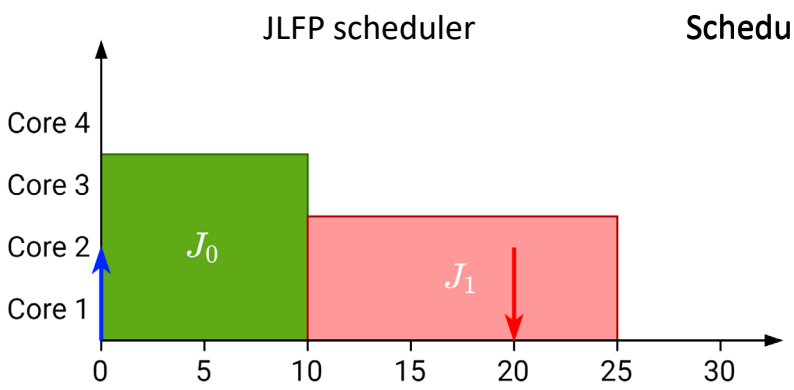


Agenda

- ✔ Gang schedulability analysis
- New scheduling policy

Core assignment

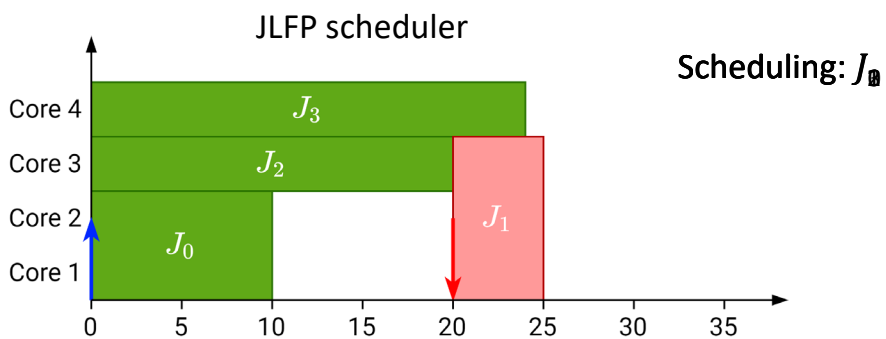
- JLFP assigns the maximum possible cores to a job
- Assign minimum number of cores that meet deadline



Scheduling: J_0

	Priority	Min cores	Max cores	Deadline	Execution time
J_0	High	1	3	20	30, 15, 10
J_1	Low	2	2	20	15

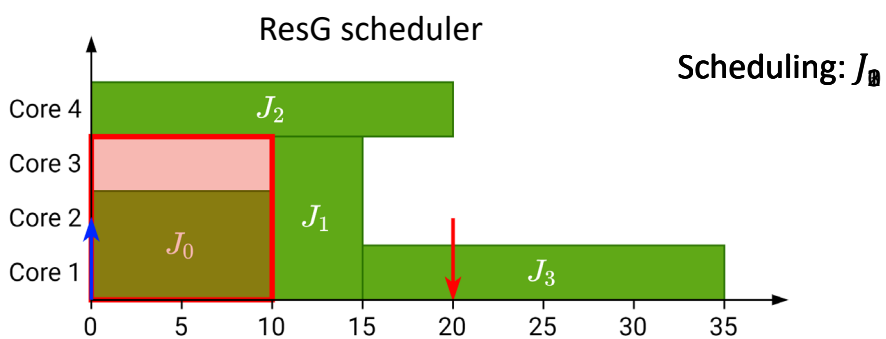
JLFP limitations with moldable gang



	Priority	Cores	Deadline	Execution time
J_0	High	2	∞	10
J_1	Mid-high	3	20	5
J_2	Mid-low	1	∞	20
J_3	Low	1	∞	20

Reservation-based gang scheduler

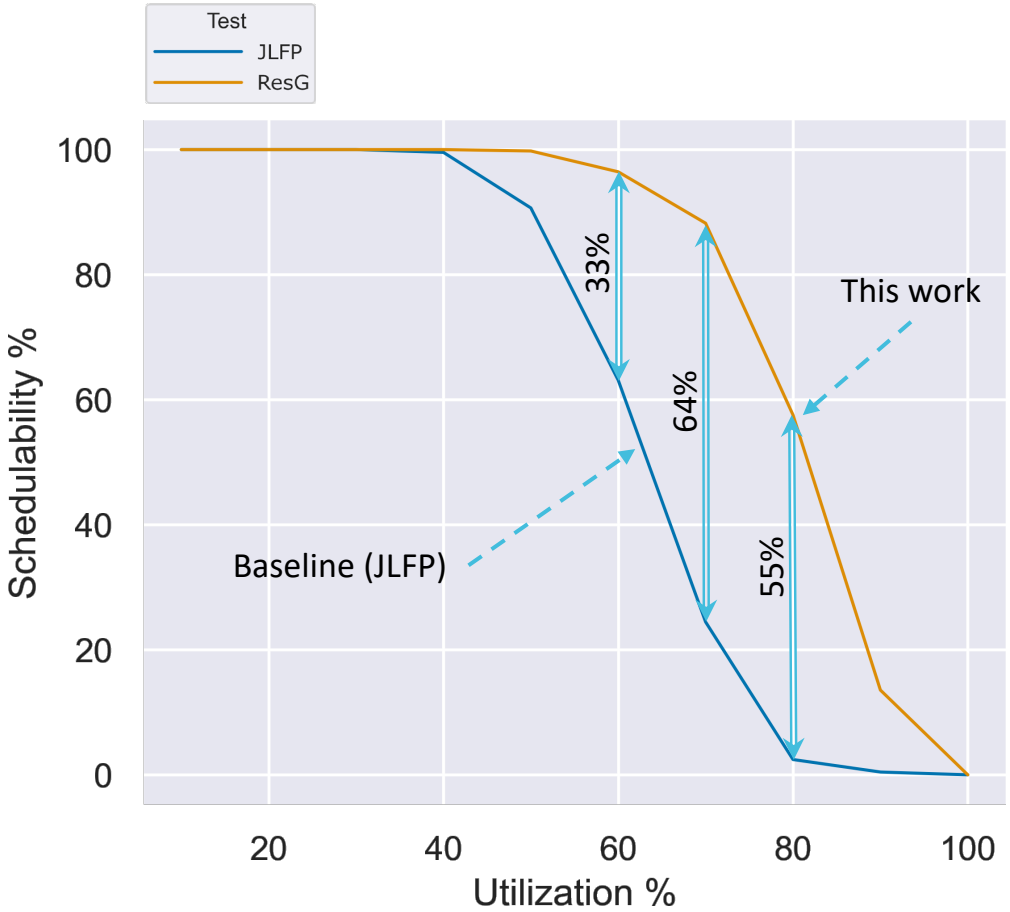
- Reservation-based
- Reserve cores of higher-priority tasks and distribute the remaining ones among lower priority tasks
- Non-work conserving scheduler



	Priority	Cores	Deadline	Execution time
J_0	High	2	∞	10
J_1	Mid-high	3	20	5
J_2	Mid-low	1	∞	20
J_3	Low	1	∞	20

Results JLFP vs ResG in simulator

- Evaluated in simulator
- Randomly generated task sets
  - System cores: 8
  - 20 moldable tasks
  - Implicit deadlines
  - Rate monotonic



Conclusions

- The maximum parallelism of a gang task has a big role in the schedulability
- With a better scheduling policy one can improve the schedulability of moldable gang tasks

Future work

- Further reduce sources of pessimism with precedence constraints
- Provide an SAG analysis for the new scheduling policy

Contributions

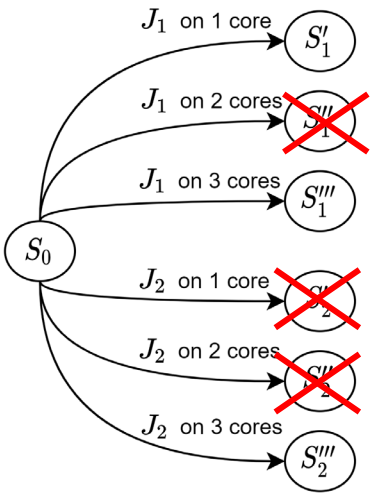
New schedulability analysis

Moldable gang tasks evaluation

Paper submitted to RTSS 2020 for the non-preemptive analysis

New scheduling policy

Presented in CAPITAL workshop



Thank you!