Schedulability analysis of limited-preemptive moldable gang tasks

Joan Marcè i Igual

TUDelft

Daily Supervisor Geoffrey Nelissen

Co-supervisor Mitra Nasri

27th of August, 2020

Real-time systems

Real-time systems

- Systems of which correctness does depends not only on **logical** results but also on **timing constraints**

Real-time systems

- Systems of which correctness does depends not only on **logical** results but also on **timing constraints**

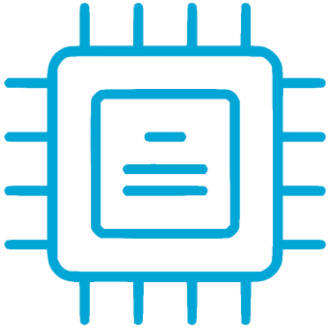
Real-time systems

- Systems of which correctness does depends not only on **logical** results but also on **timing constraints**

Real-time systems

- Systems of which correctness does depends not only on **logical** results but also on **timing constraints**

- Multicore systems



Definitions

Definitions

- Task
  - A functionality of the system

Definitions

- Task
  - A functionality of the system
- Job
  - Instance of a task

Definitions

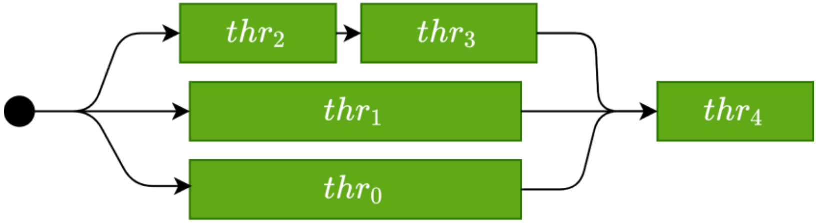
- Task
  - A functionality of the system
- Job
  - Instance of a task
- Schedule
  - A particular assignment of jobs to the processors and time intervals

Definitions

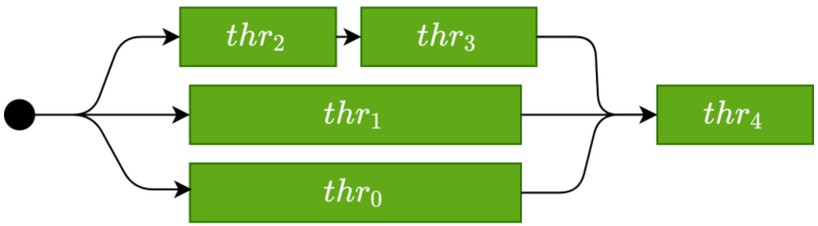
- Task
  - A functionality of the system
- Job
  - Instance of a task
- Schedule
  - A particular assignment of jobs to the processors and time intervals
- Scheduling policy
  - Algorithm that produces a schedule
  - FIFO, Round-Robin, JLFP, EDF

What is gang?

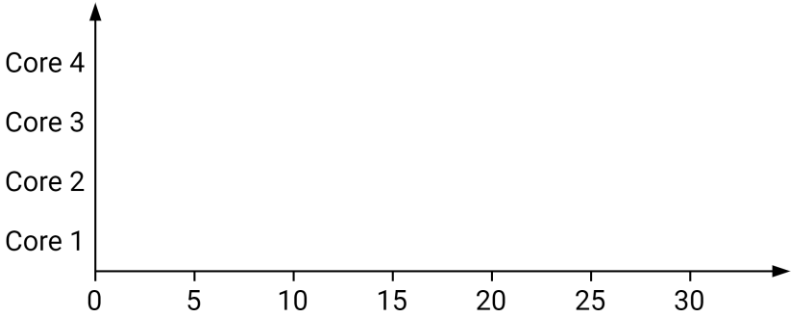
What is gang?



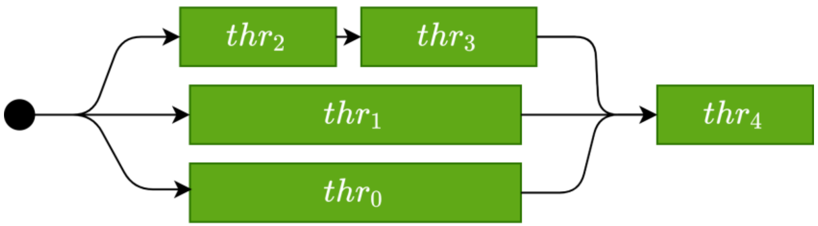
What is gang?



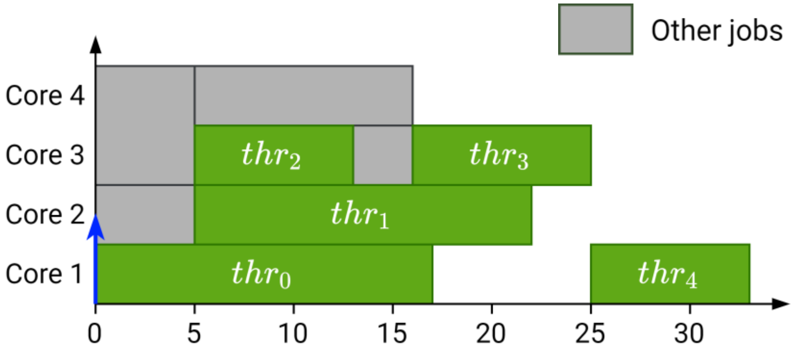
Global scheduling



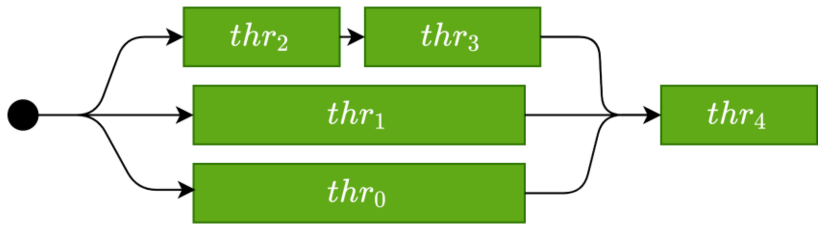
What is gang?



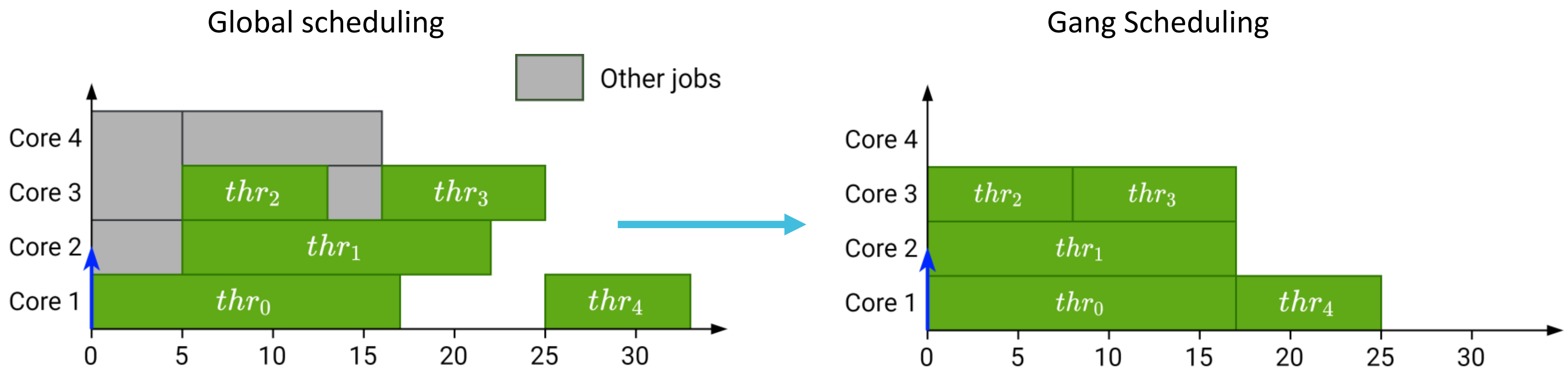
Global scheduling



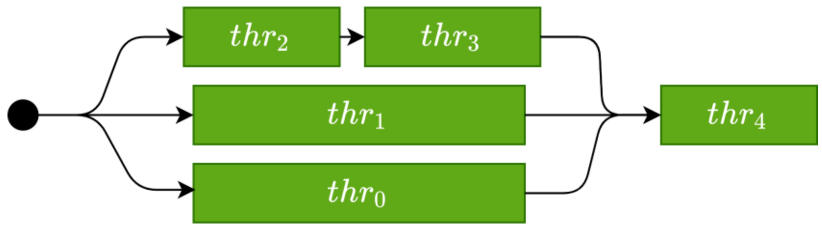
What is gang?



Parallel threads together as a "gang"

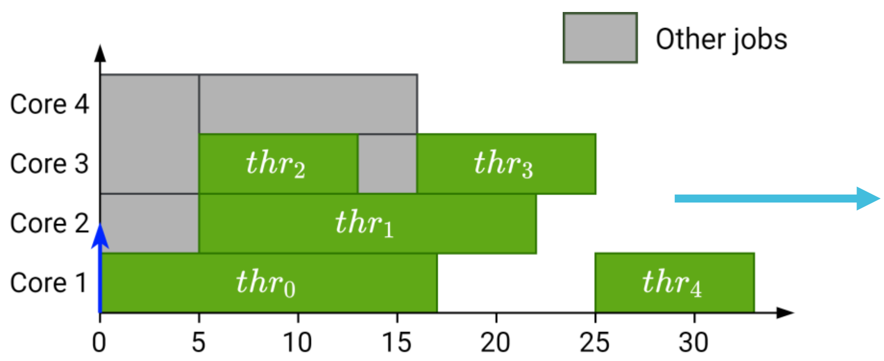


What is gang?

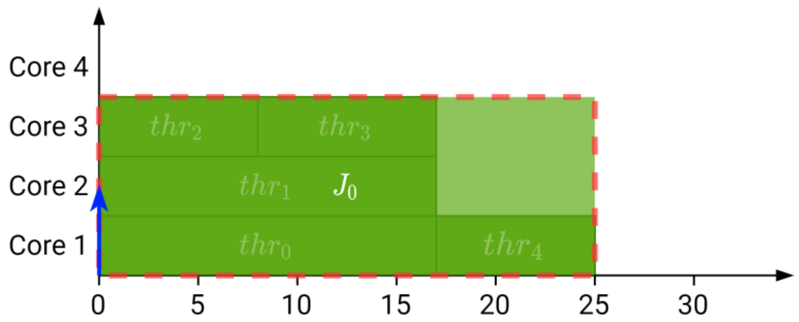


Parallel threads together as a “gang”

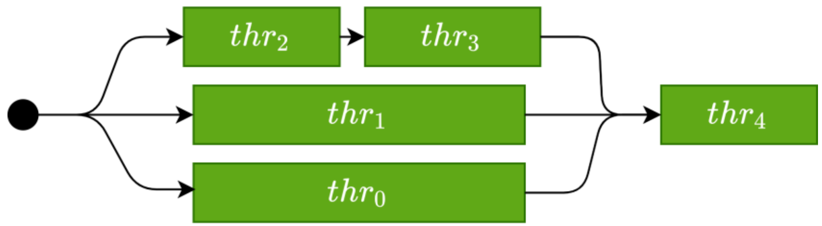
Global scheduling



Gang Scheduling

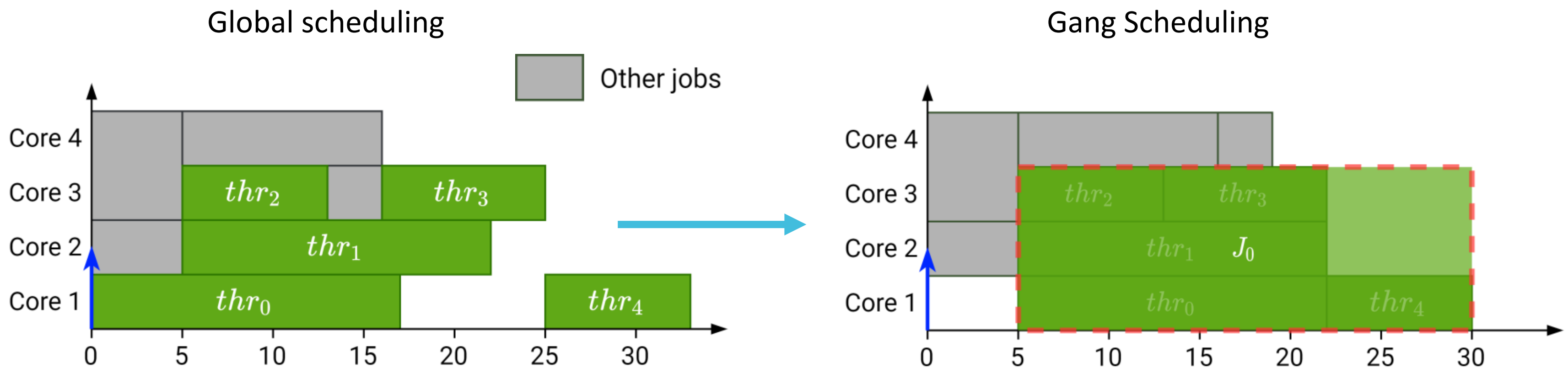


What is gang?

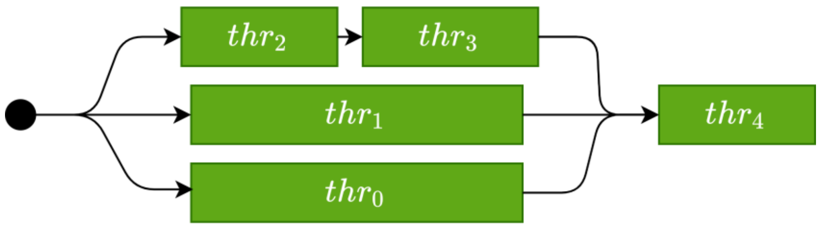


Parallel threads together as a "gang"

Execution does not start until there are enough cores



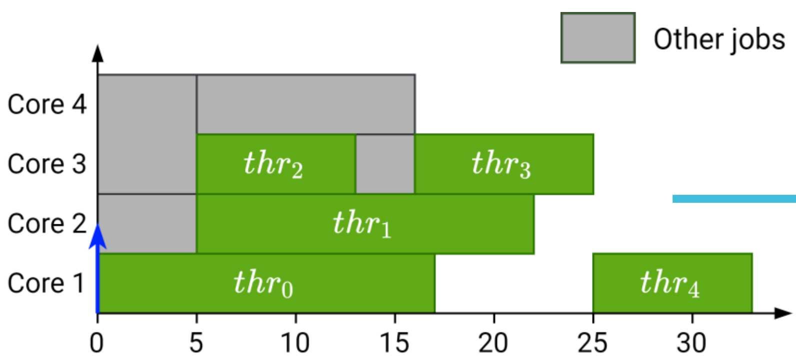
What is gang?



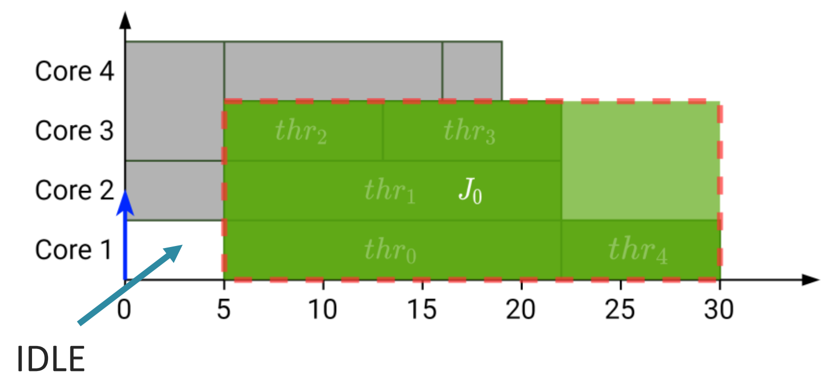
Parallel threads together as a “gang”

Execution does not start until there are enough cores

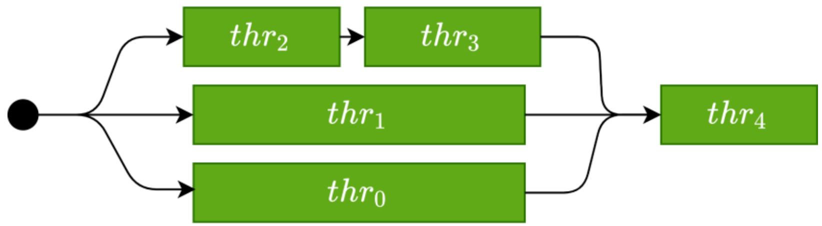
Global scheduling



Gang Scheduling

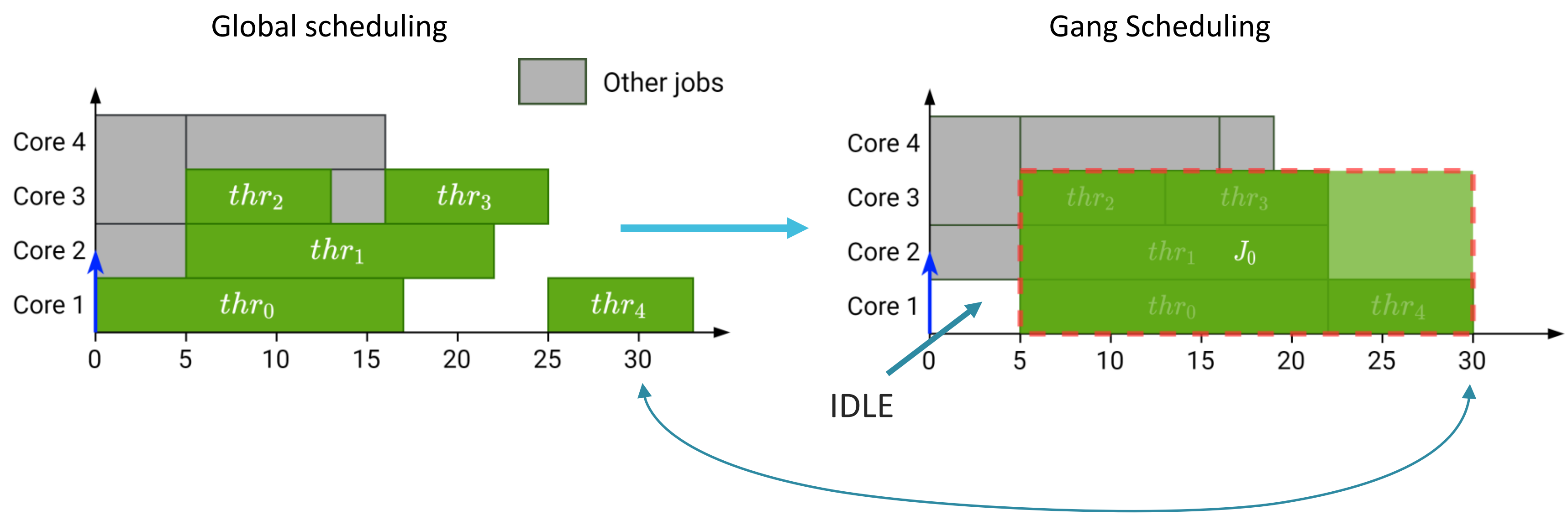


What is gang?



Parallel threads together as a “gang”

Execution does not start until there are enough cores



Why gang?

Why gang?

- More efficient synchronization

Why gang?

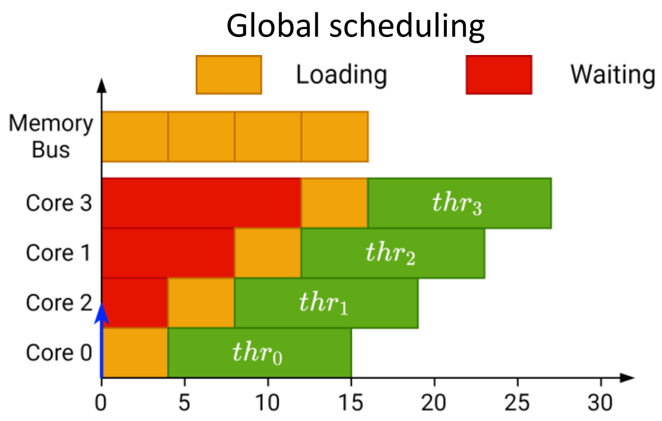
- More efficient synchronization
- Reduces variability in the execution

Why gang?

- More efficient synchronization
- Reduces variability in the execution
- Avoids overhead when loading initial data

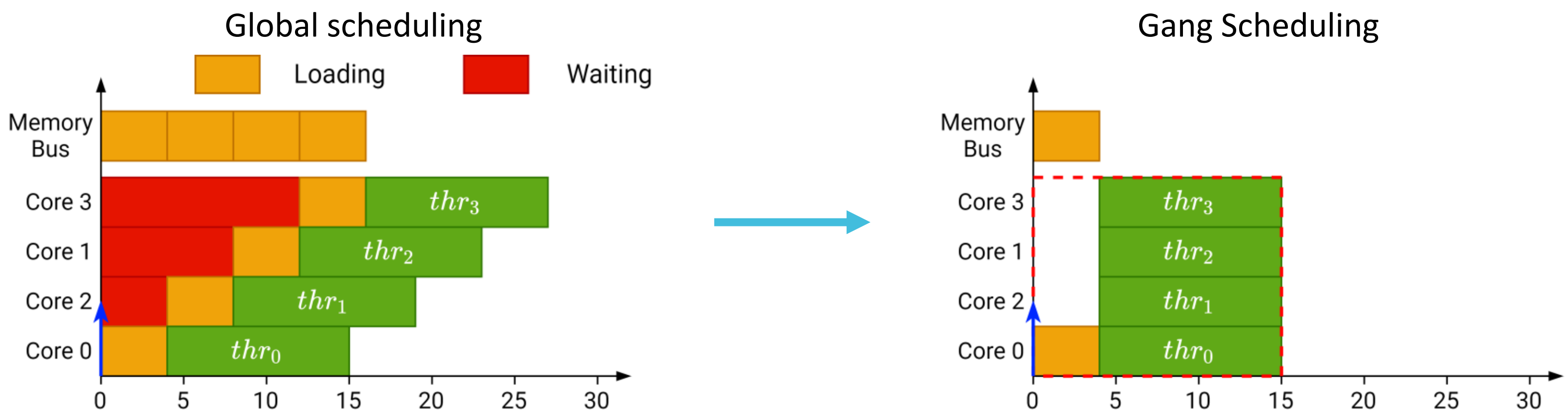
Why gang?

- More efficient synchronization
- Reduces variability in the execution
- Avoids overhead when loading initial data



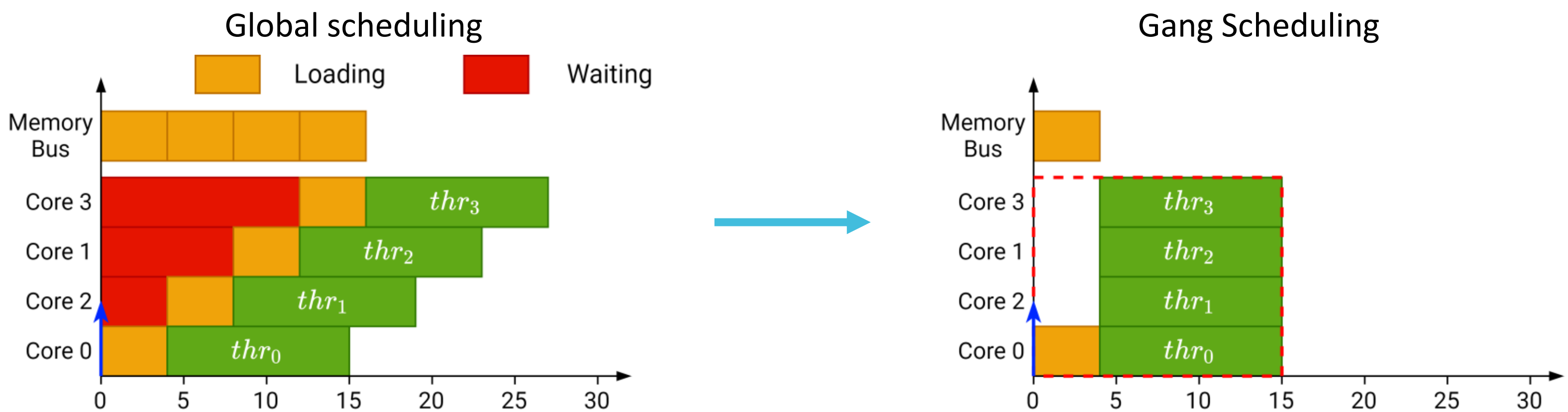
Why gang?

- More efficient synchronization
- Reduces variability in the execution
- Avoids overhead when loading initial data



Why gang?

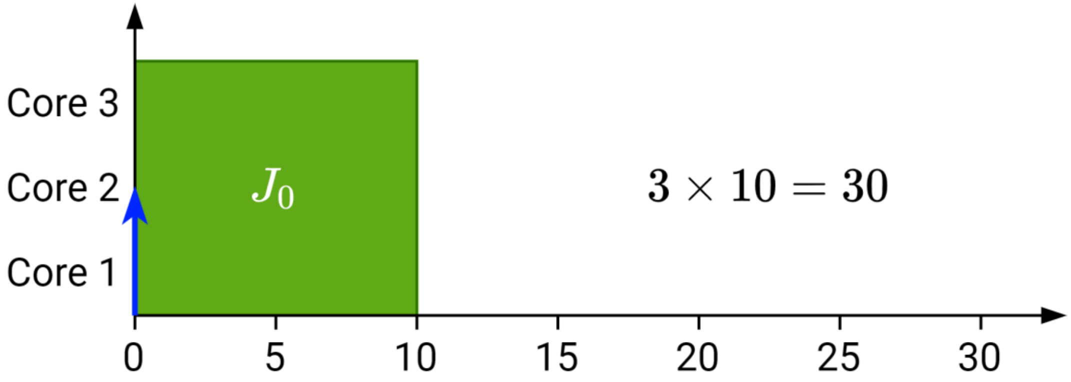
- More efficient synchronization
- Reduces variability in the execution
- Avoids overhead when loading initial data
- Shows its full potential when executed non-preemptively



Types of gang

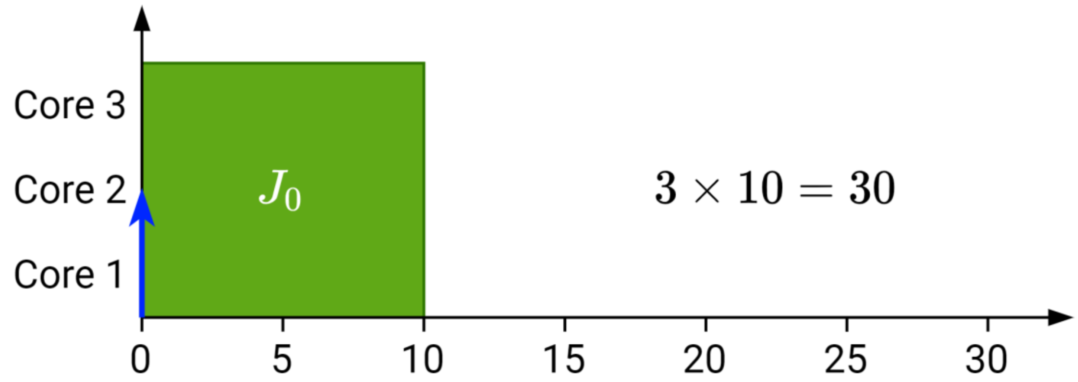
Types of gang

- **Rigid**: number of cores set by programmer



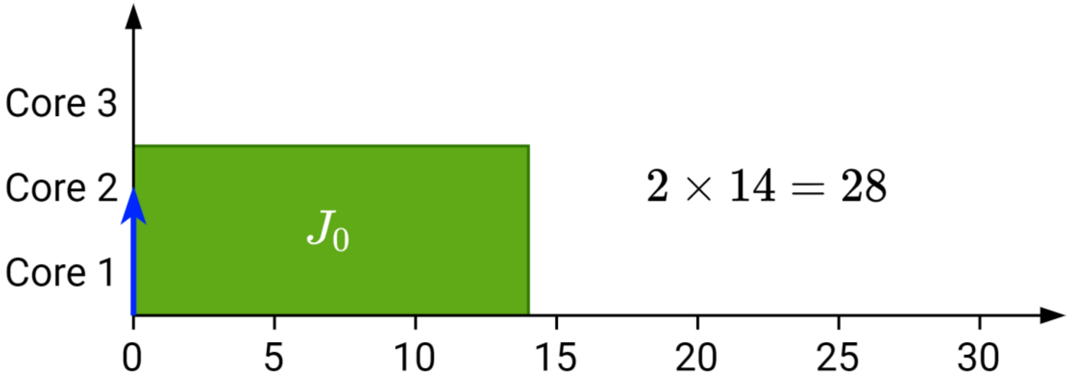
Types of gang

- **Rigid**: number of cores set by programmer
- **Moldable**: number of cores assigned when job is dispatched



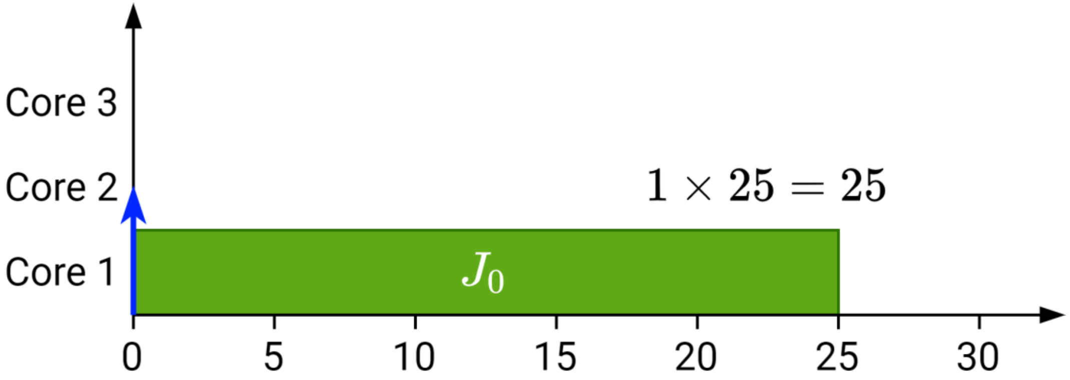
Types of gang

- **Rigid**: number of cores set by programmer
- **Moldable**: number of cores assigned when job is dispatched



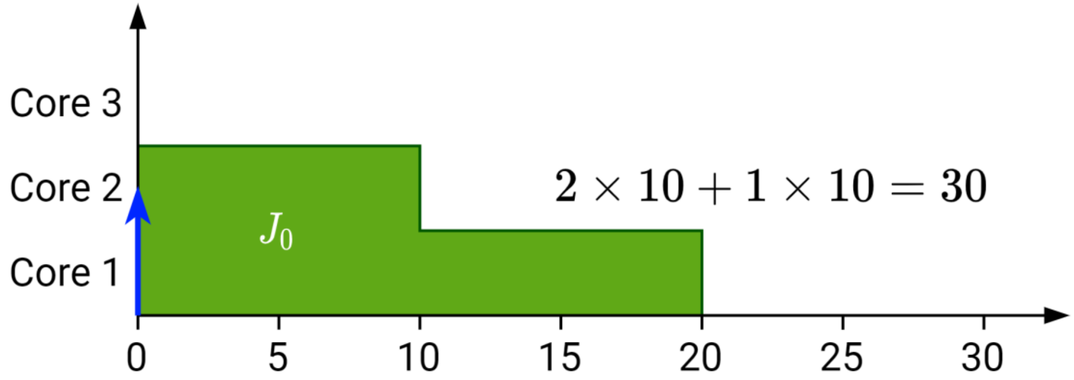
Types of gang

- **Rigid**: number of cores set by programmer
- **Moldable**: number of cores assigned when job is dispatched



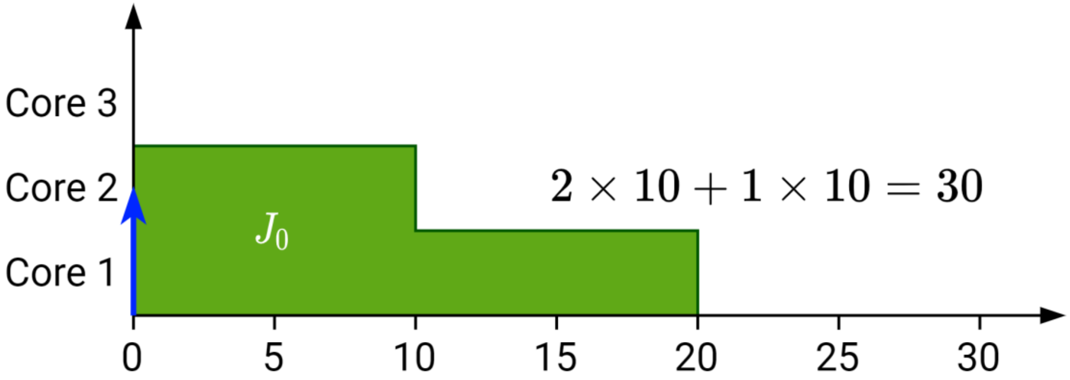
Types of gang

- **Rigid**: number of cores set by programmer
- **Moldable**: number of cores assigned when job is dispatched
- **Malleable**: number of cores can change during runtime



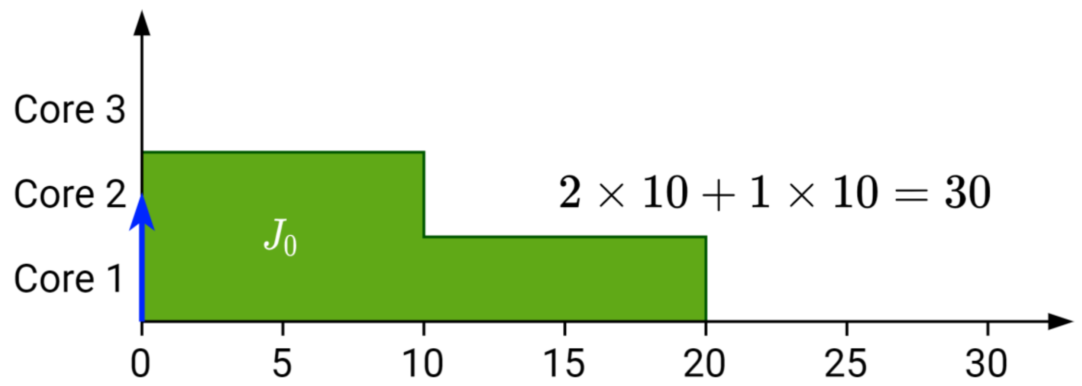
Types of gang

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



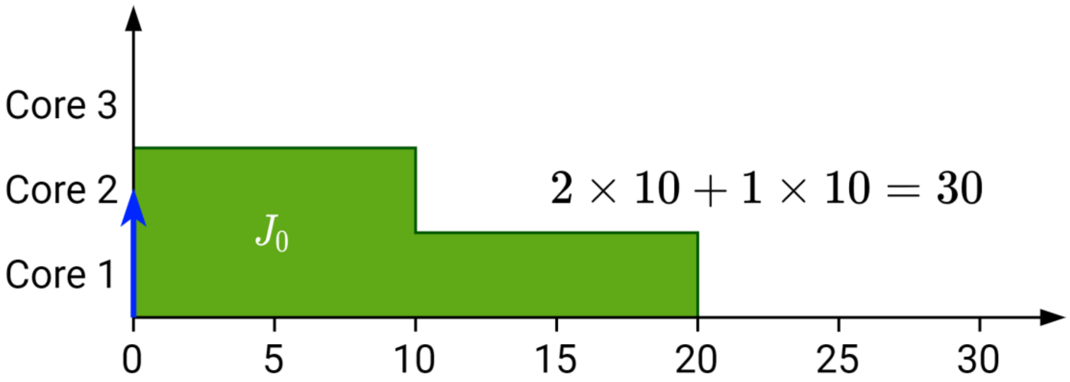
Types of gang

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



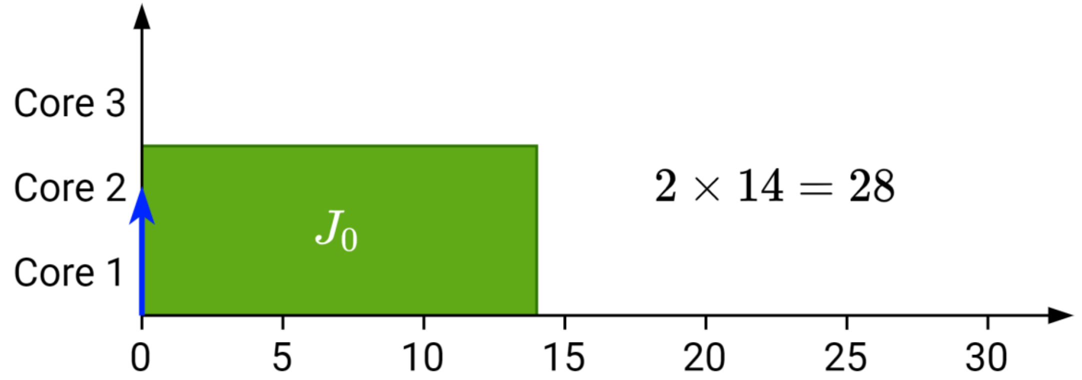
Types of gang

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

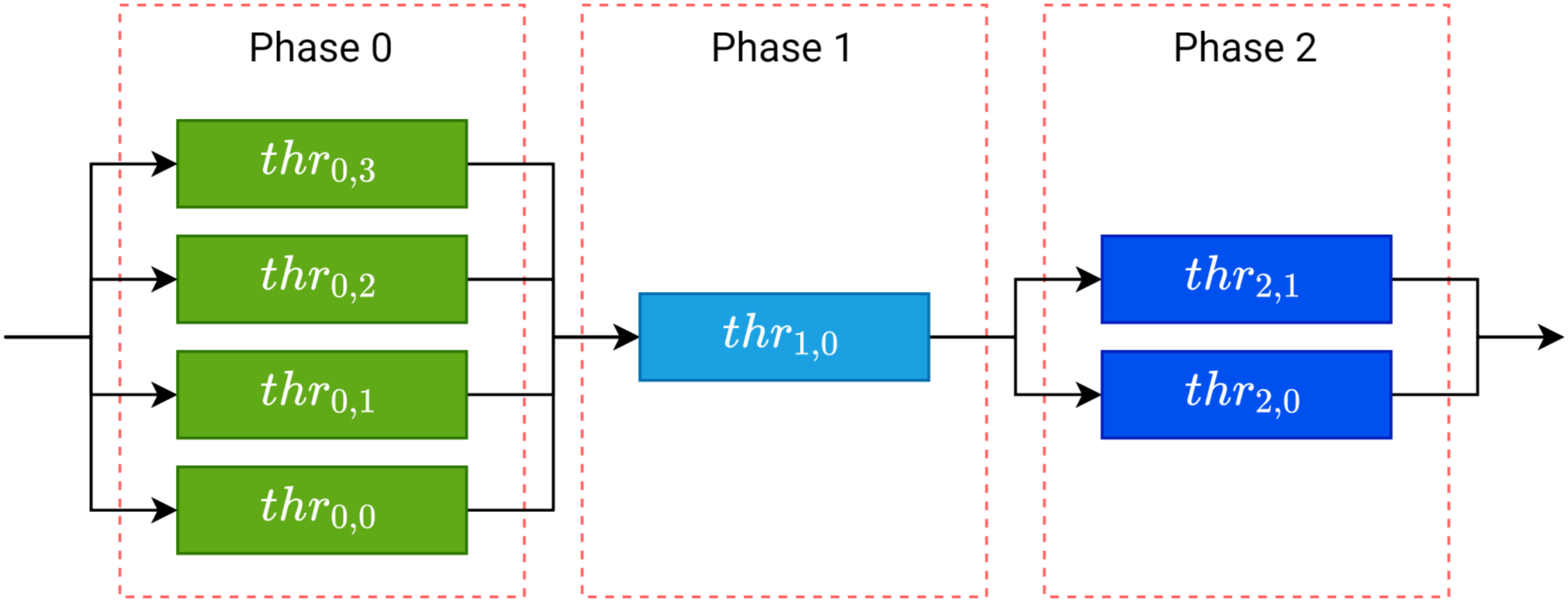


Types of gang

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

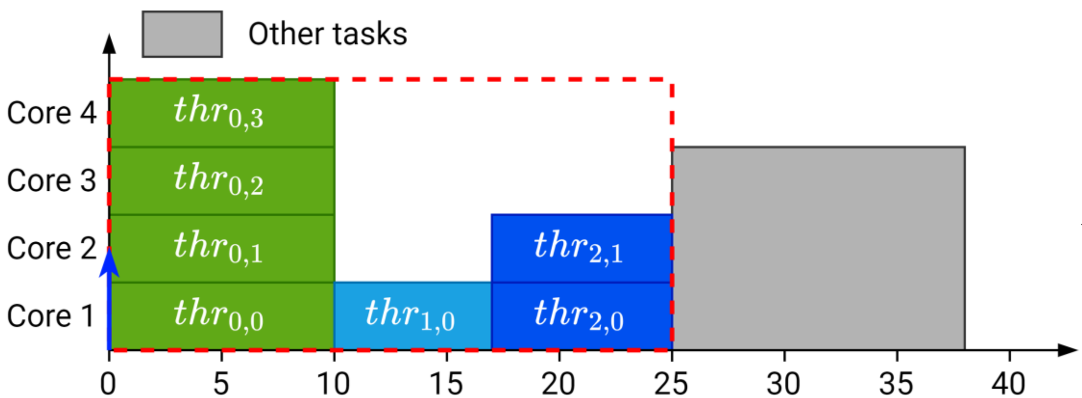


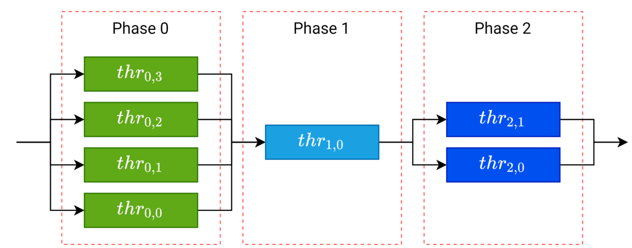
Bundled scheduling[1] vs limited-preemptive



Bundled scheduling[1] vs limited-preemptive

- Rigid gang reserves the whole block

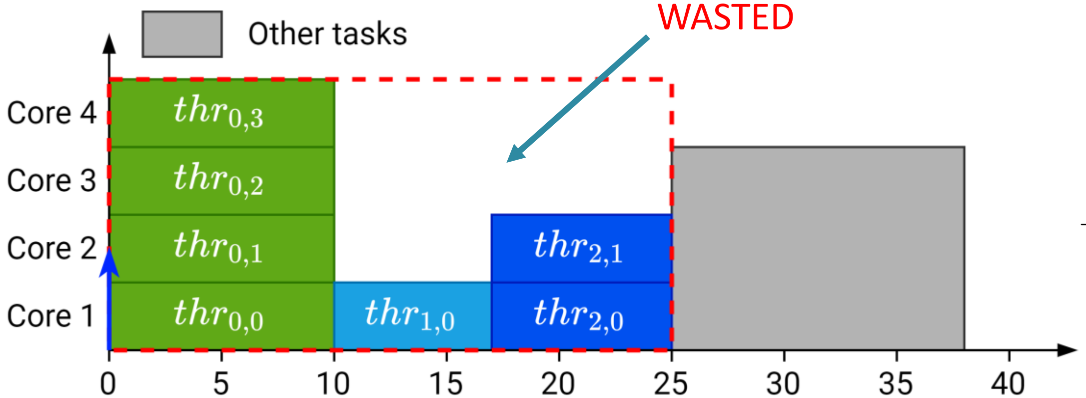




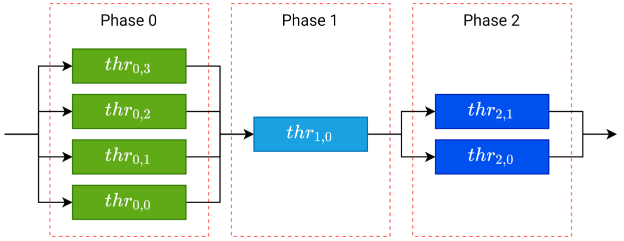
Rigid gang scheduling

Bundled scheduling[1] vs limited-preemptive

- Rigid gang reserves the whole block

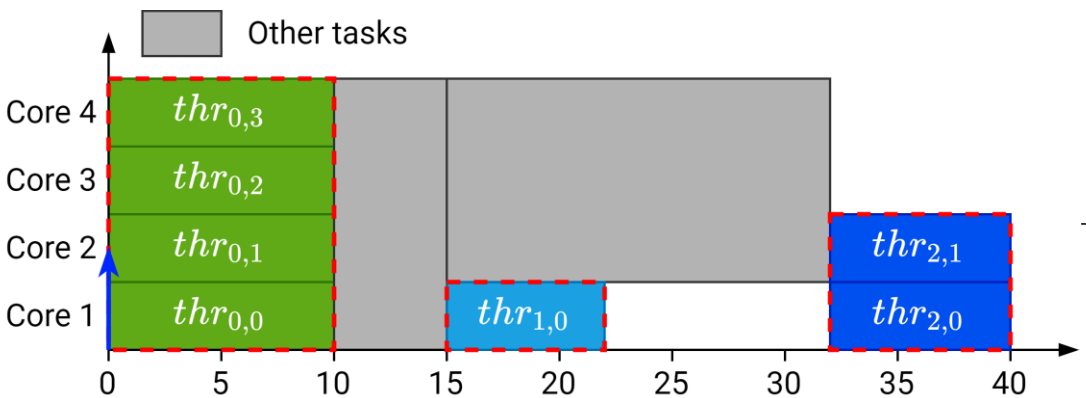


Rigid gang scheduling

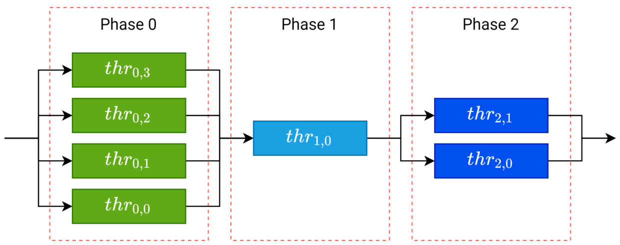


Bundled scheduling[1] vs limited-preemptive

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

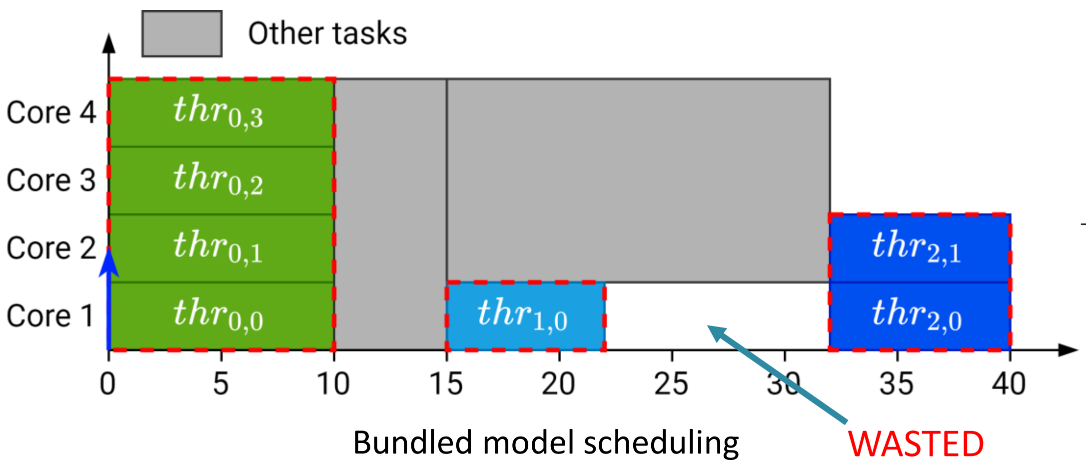


Bundled model scheduling

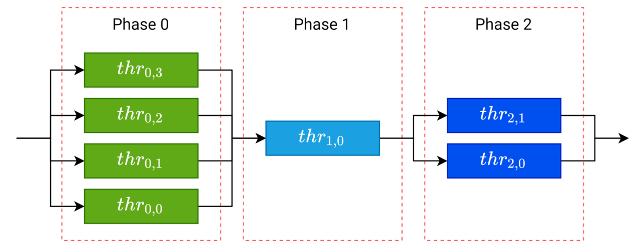


Bundled scheduling[1] vs limited-preemptive

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

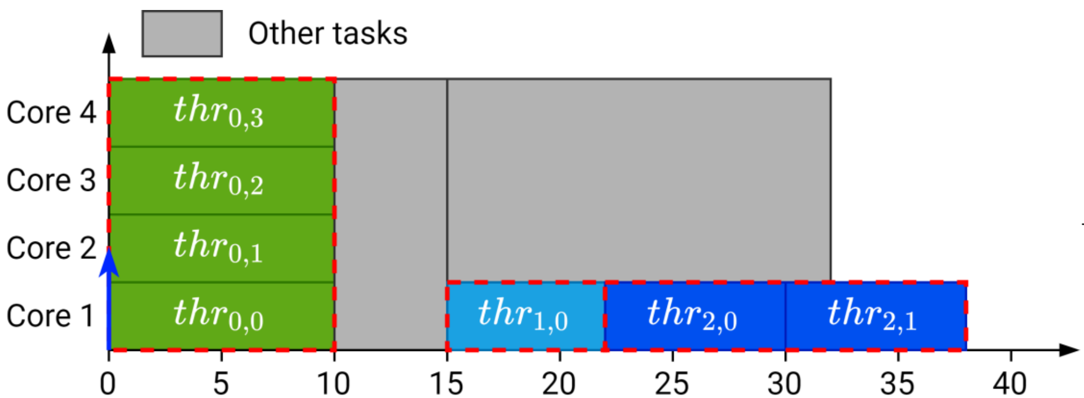


Bundled model scheduling

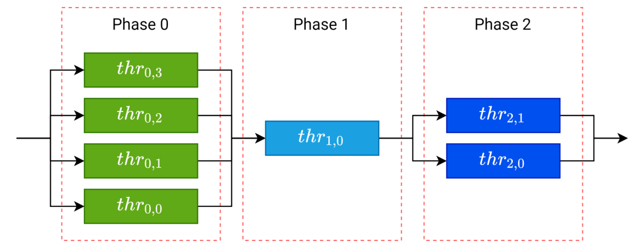


Bundled scheduling[1] vs limited-preemptive

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



Limited-Preemptive scheduling



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

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

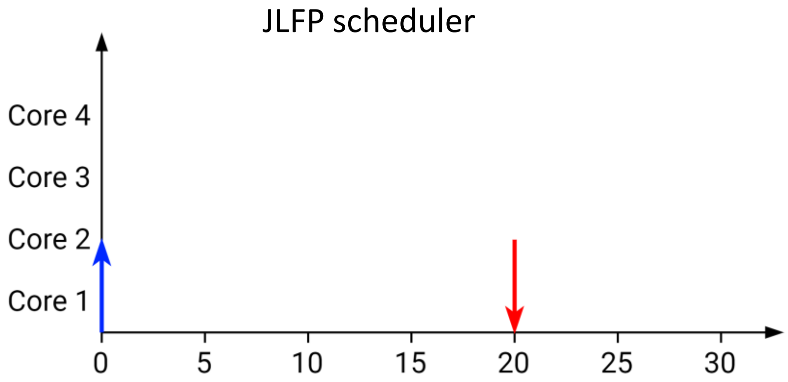
- Based on global JLFP scheduler

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

- Based on global JLFP scheduler
- Assigns maximum number of available cores to a job between the range of cores that the job allows

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

- Based on global JLFP scheduler
- Assigns maximum number of available cores to a job between the range of cores that the job allows



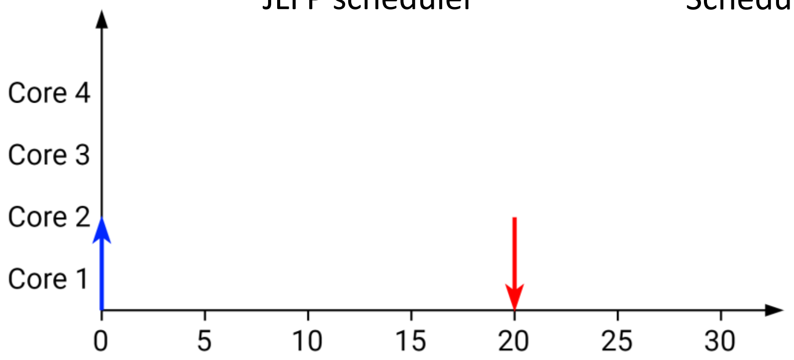
	Priority	Min cores	Max cores	Deadline	Execution time
J_0	High	2	3	∞	15, 10
J_1	Low	2	2	20	15

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

- Based on global JLFP scheduler
- Assigns maximum number of available cores to a job between the range of cores that the job allows

JLFP scheduler

Scheduling: J_0



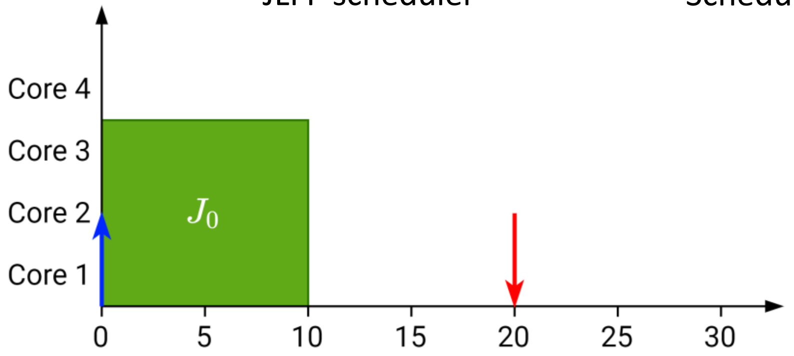
	Priority	Min cores	Max cores	Deadline	Execution time
J_0	High	2	3	∞	15, 10
J_1	Low	2	2	20	15

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

- Based on global JLFP scheduler
- Assigns maximum number of available cores to a job between the range of cores that the job allows

JLFP scheduler

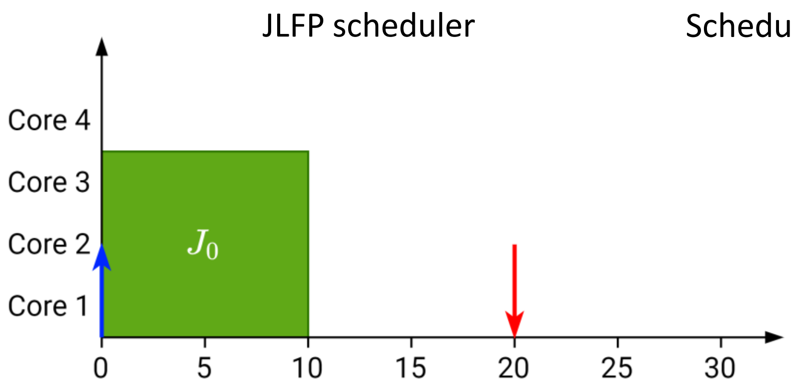
Scheduling: J_0



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

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

- Based on global JLFP scheduler
- Assigns maximum number of available cores to a job between the range of cores that the job allows

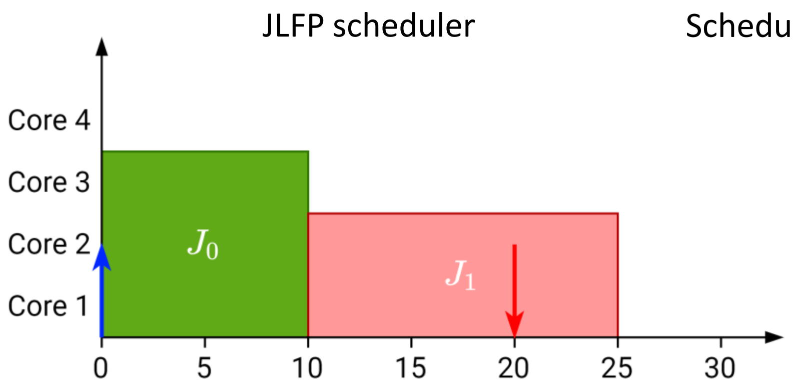


Scheduling: J_1

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

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

- Based on global JLFP scheduler
- Assigns maximum number of available cores to a job between the range of cores that the job allows

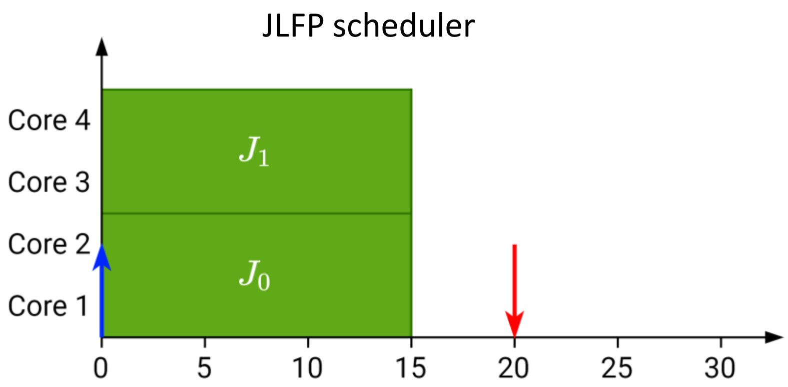


Scheduling: J_1

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

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

- Based on global JLFP scheduler
- Assigns maximum number of available cores to a job between the range of cores that the job allows



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

Previous work

Previous work

Introduced in high-performance computing in 1982[1]

[1]Ousterhout, 1982

Previous work

Introduced in high-performance computing in 1982[1]

Preemptive solutions

[1]Ousterhout, 1982

Previous work

Introduced in high-performance computing in 1982[1]

Preemptive solutions

Schedulability tests

[1]Ousterhout, 1982

Previous work

Introduced in high-performance computing in 1982[1]

Preemptive solutions

Schedulability tests

- Job-level fixed-priority[2]

[1]Ousterhout, 1982

[2]Goossens et al., 2010

Previous work

Introduced in high-performance computing in 1982[1]

Preemptive solutions

Schedulability tests

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

[1]Ousterhout, 1982

[2]Goossens et al., 2010

[3]Kato et al., 2009

Previous work

Introduced in high-performance computing in 1982[1]

Preemptive solutions

Schedulability tests

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

Schedulers

[1]Ousterhout, 1982

[2]Goossens et al., 2010

[3]Kato et al., 2009

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]

[1]Ousterhout, 1982
[2]Goossens et al., 2010
[3]Kato et al., 2009
[4]Goossens et al., 2016

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]

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

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

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

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

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

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

Project goals

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

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 the jobs

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 the jobs

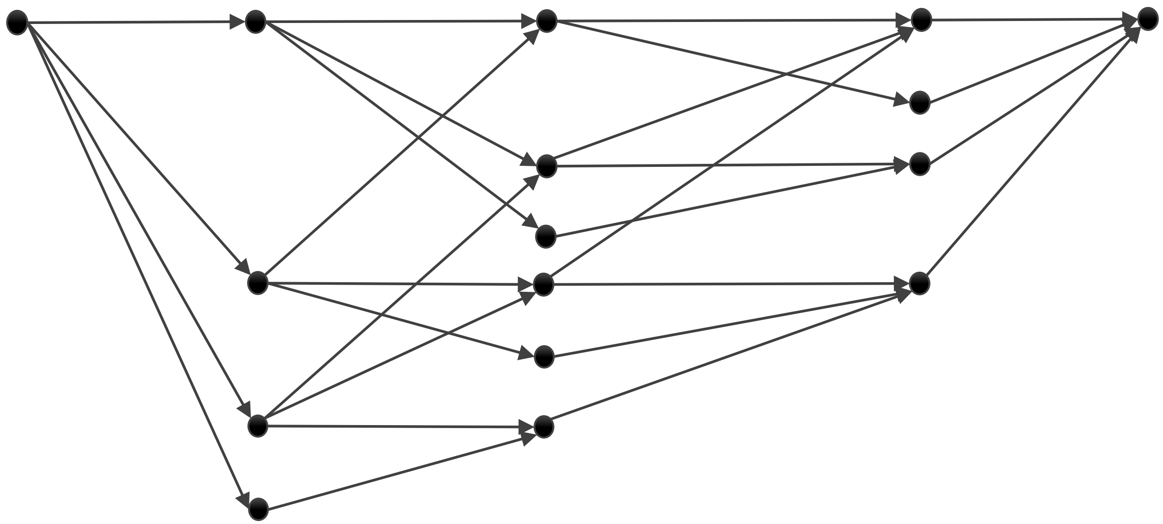
3. Propose a new scheduling algorithm to improve the schedulability of limited-preemptive moldable gang tasks

Agenda

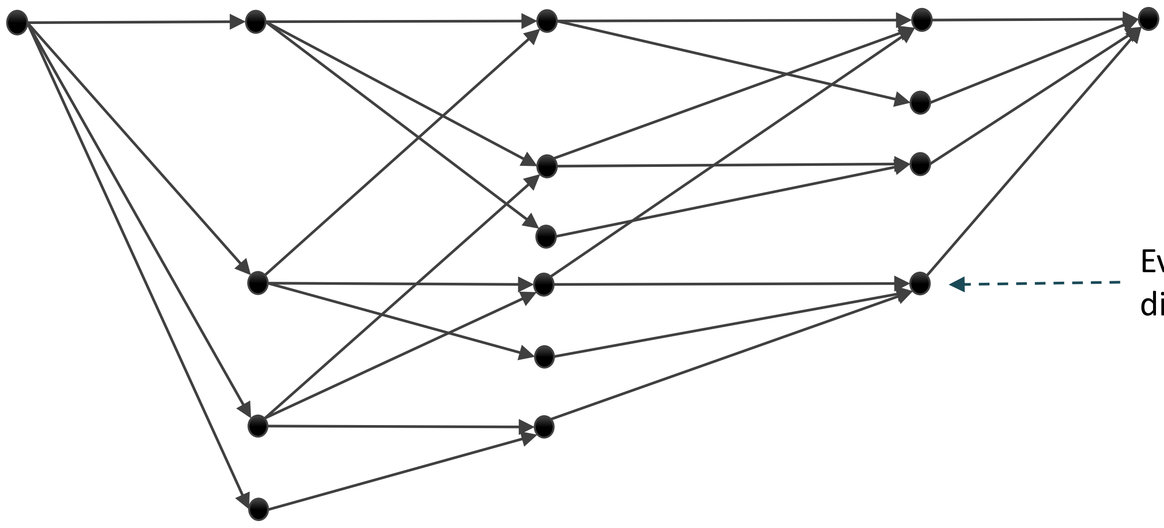
- Gang schedulability analysis
- New scheduling policy

Schedule abstraction graph

Schedule abstraction graph

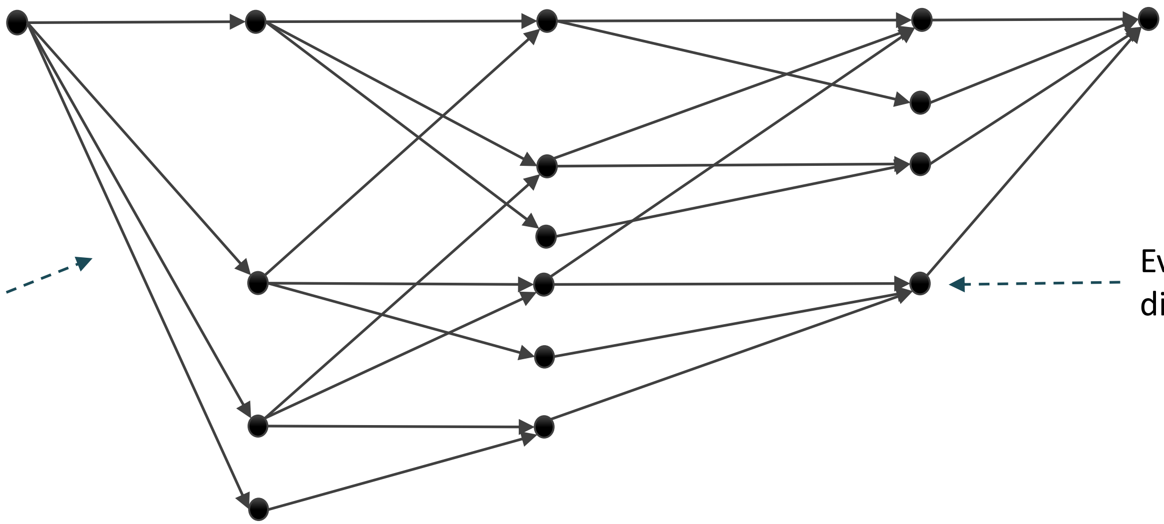


Schedule abstraction graph



Every node is a different system state

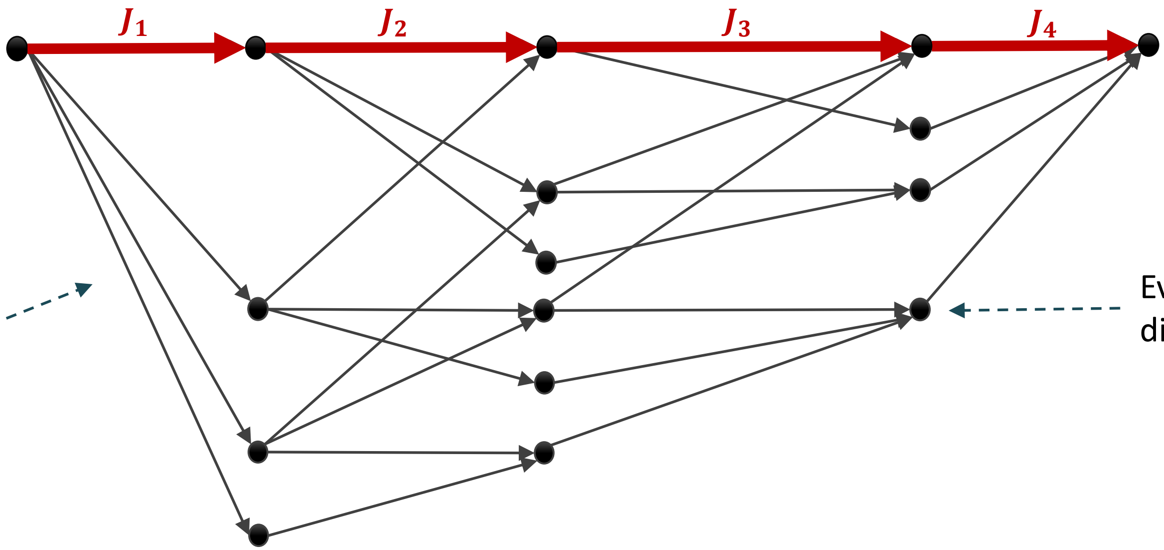
Schedule abstraction graph



Every edge is a different scheduler decision

Every node is a different system state

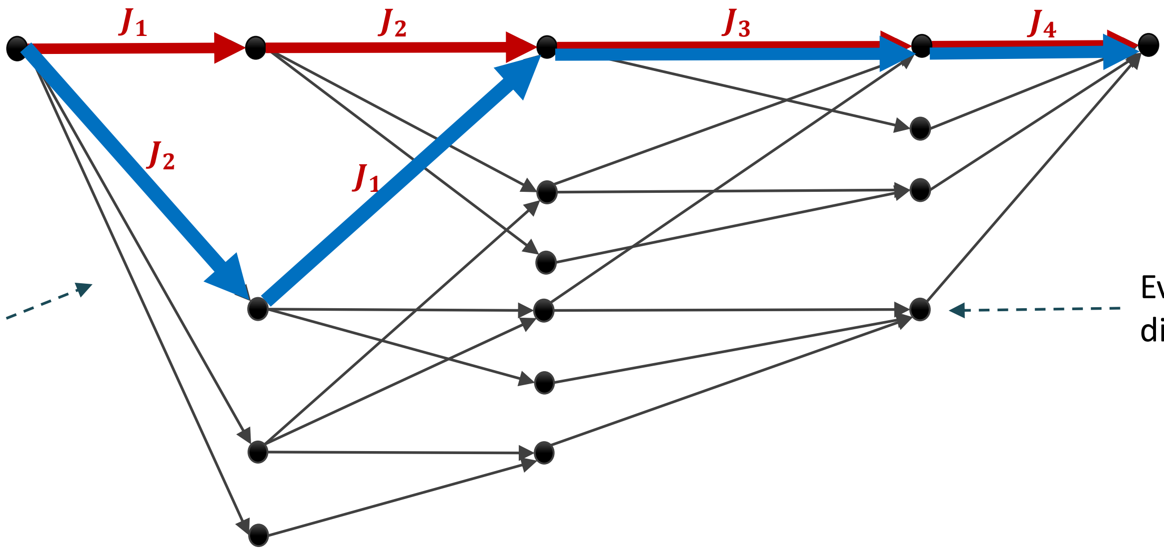
Schedule abstraction graph



Every node is a different system state

Every edge is a different scheduler decision

Schedule abstraction graph

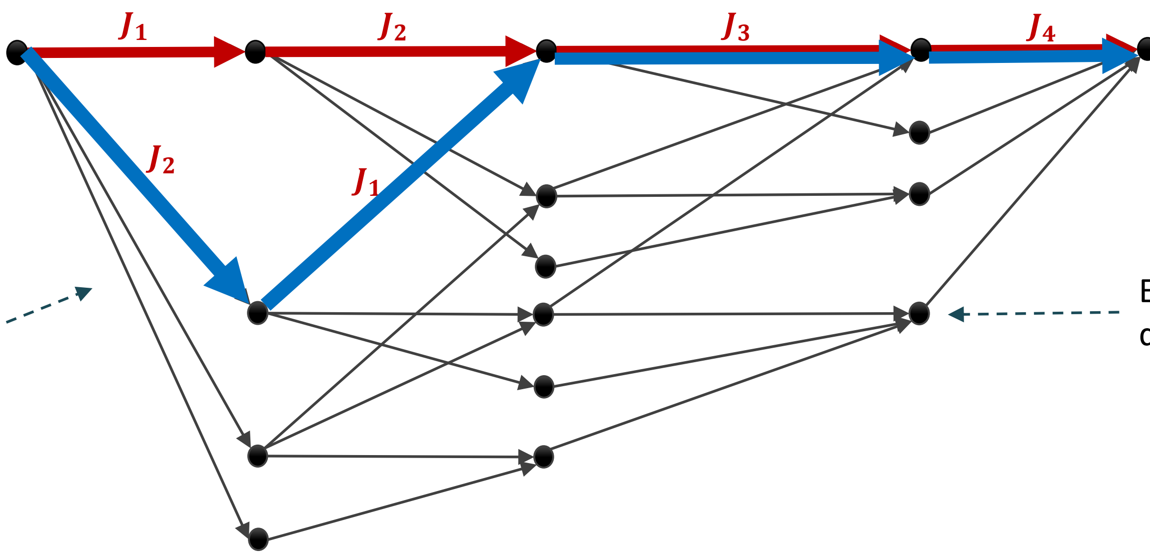


Every edge is a different scheduler decision

Every node is a different system state

Schedule abstraction graph

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



Every edge is a different scheduler decision

Every node is a different system state

Extending SAG

Extending SAG

- Update system state representation

Extending SAG

- Update system state representation
- Update expansion rules

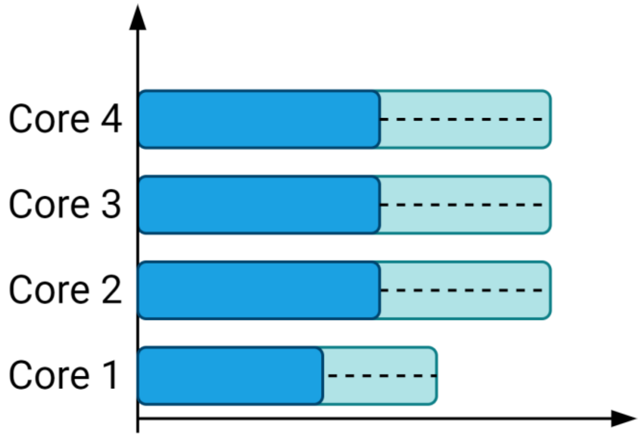
Extending SAG

- Update system state representation
- Update expansion rules
- Update merge rules

New state representation

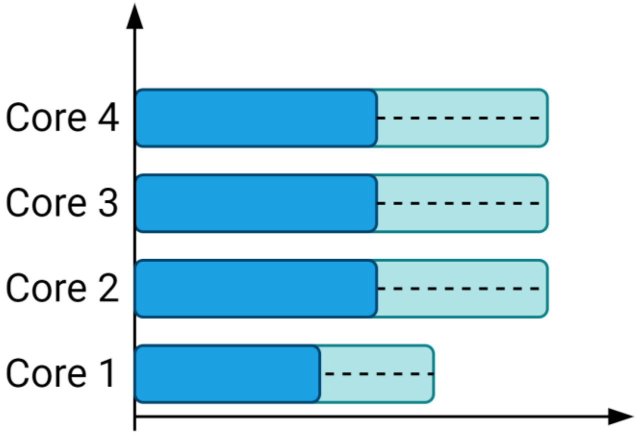
New state representation

- We already had core availabilities



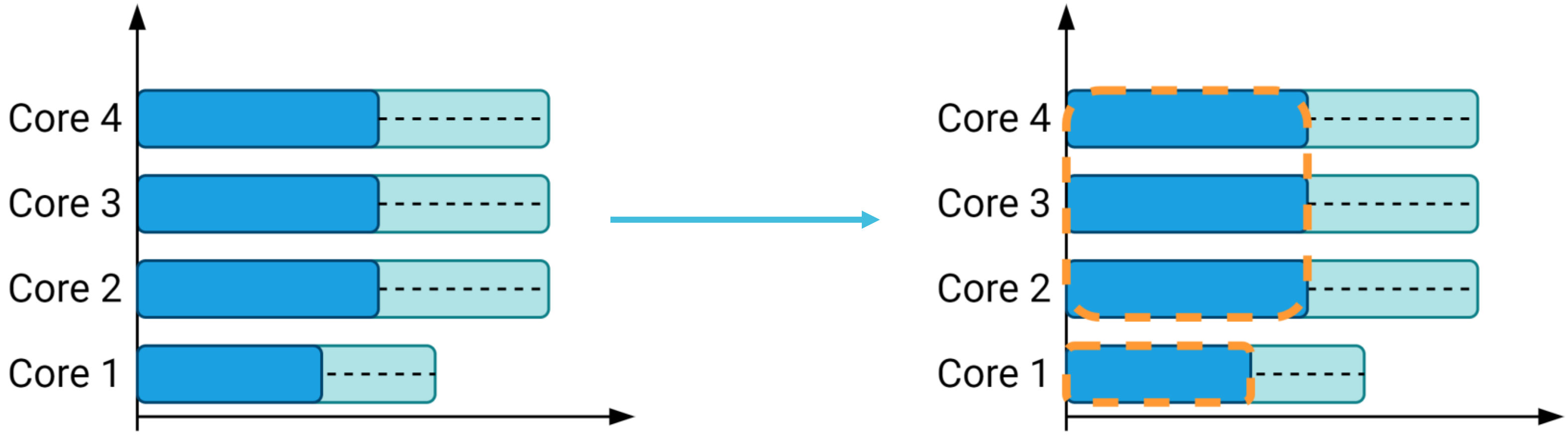
New state representation

- We already had core availabilities
- We need to keep track of



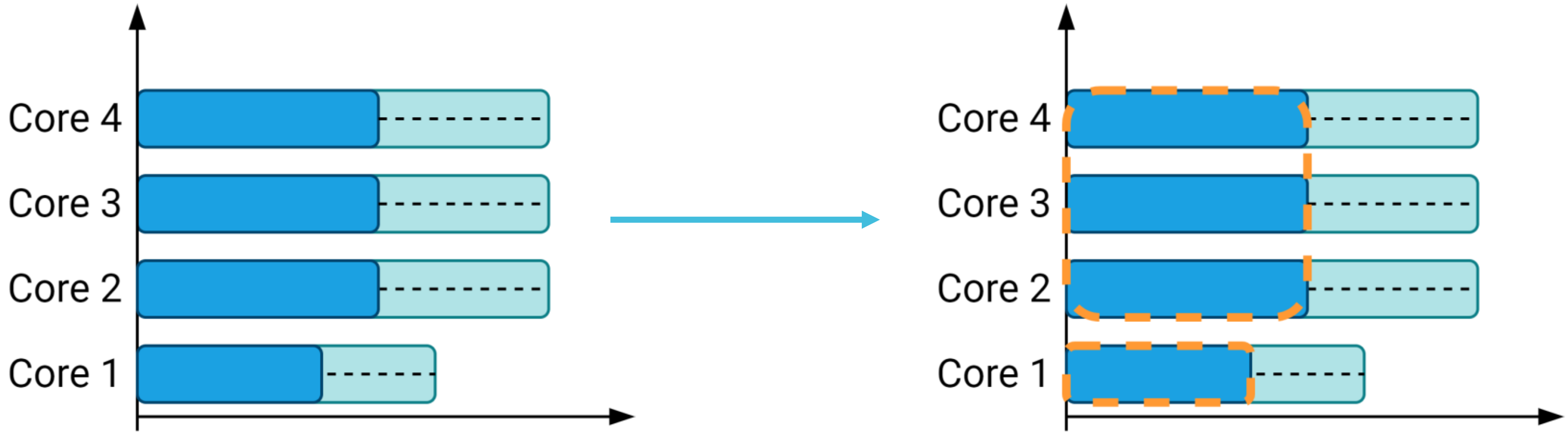
New state representation

- We already had core availabilities
- We need to keep track of
  - Groups of cores



New state representation

- We already had core availabilities
- We need to keep track of
  - Groups of cores
  - Certainly running jobs



New SAG expansion rules

New SAG expansion rules

- Previously a state was created for every schedulable job

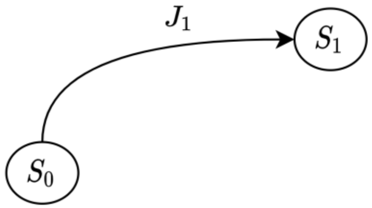
TUDelft

New SAG expansion rules

- Previously a state was created for every schedulable job

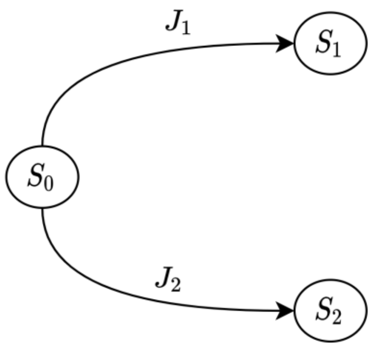
New SAG expansion rules

- Previously a state was created for every schedulable job



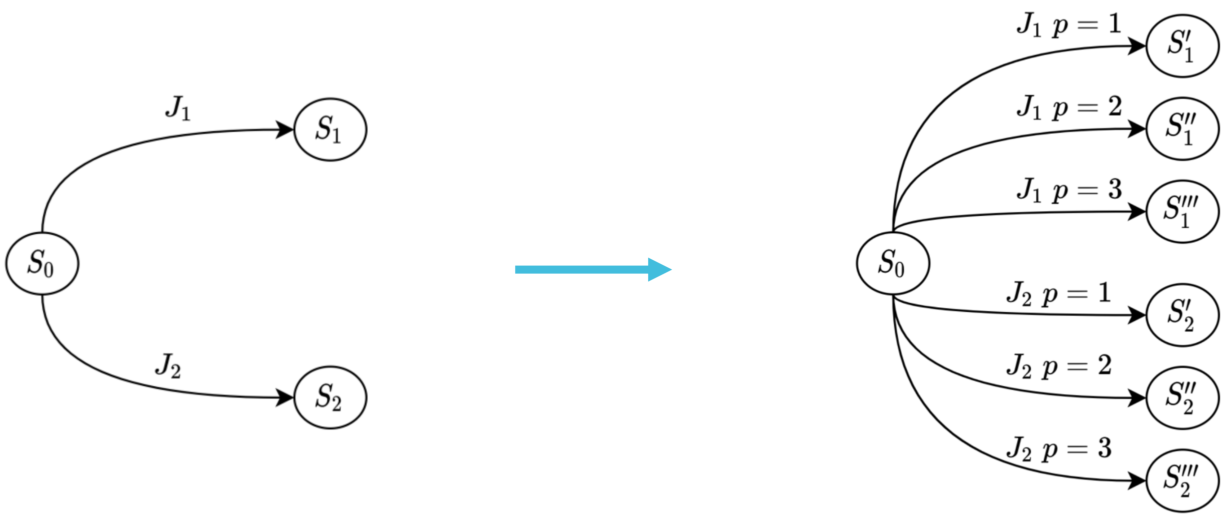
New SAG expansion rules

- Previously a state was created for every schedulable job



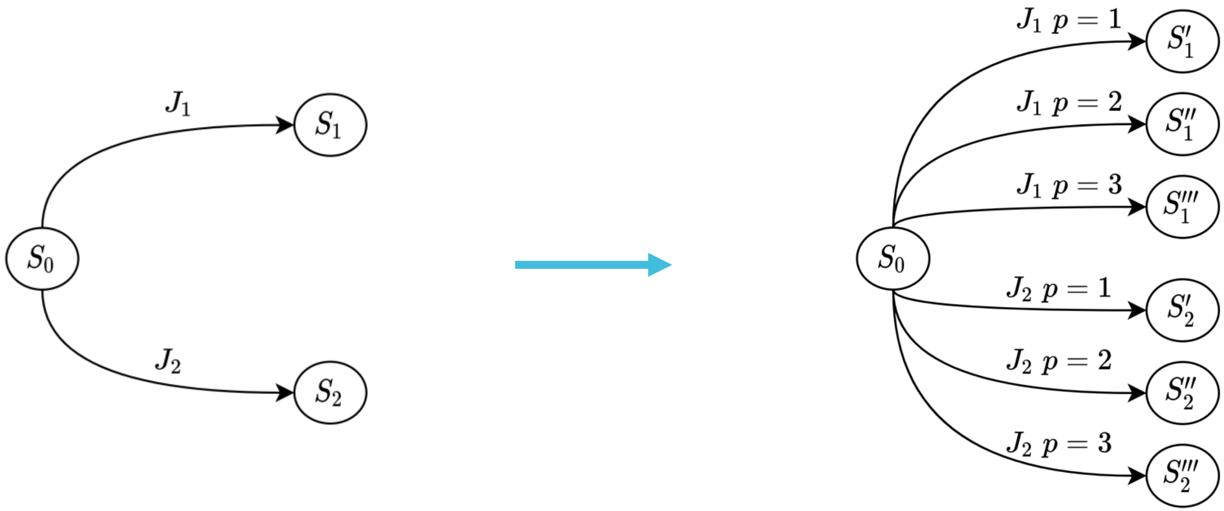
New SAG expansion rules

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

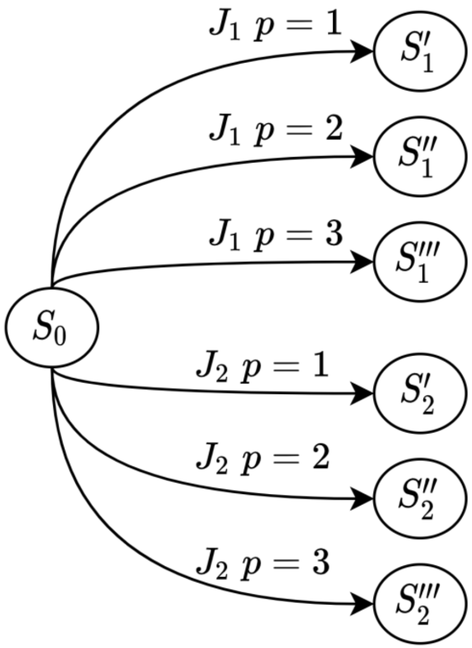


New SAG expansion rules

- Previously a state was created for every schedulable job
- Now a state is created for every job and possible number of cores
- Can stimulate state-space explosion

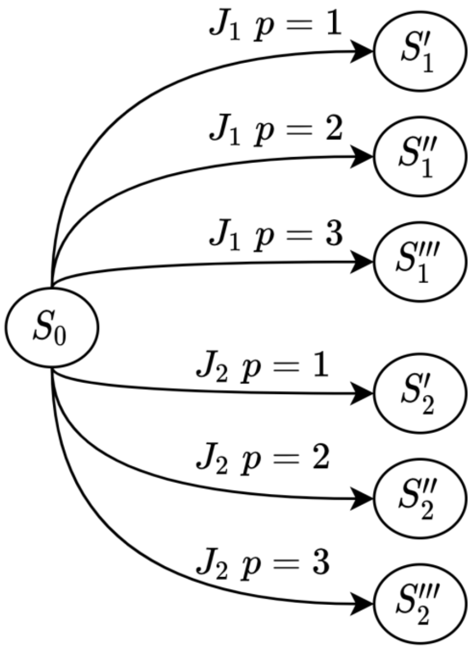


New SAG expansion rules



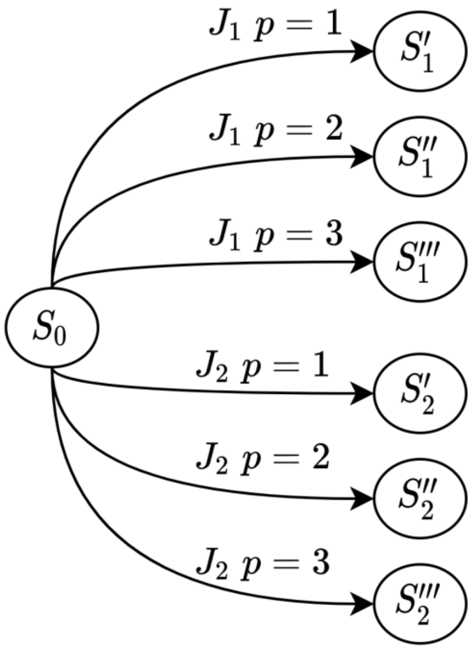
New SAG expansion rules

- Purge states by checking that



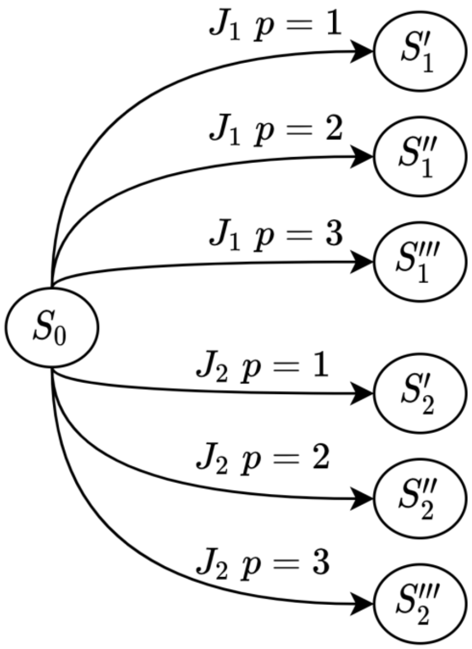
New SAG expansion rules

- Purge states by checking that
  - p cores available
  -



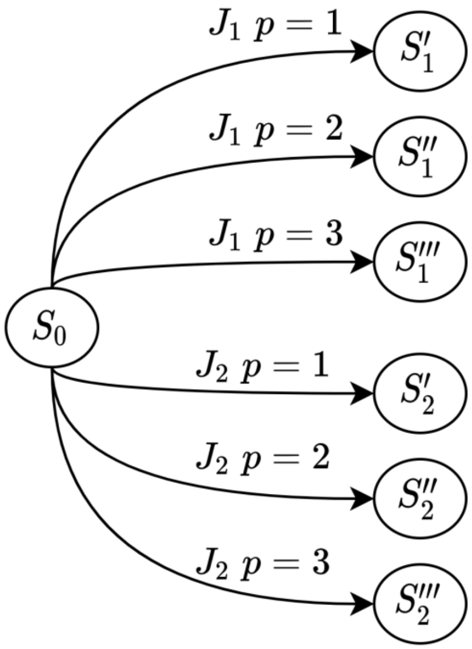
New SAG expansion rules

- Purge states by checking that
  - p cores available
  - More cores **not** available



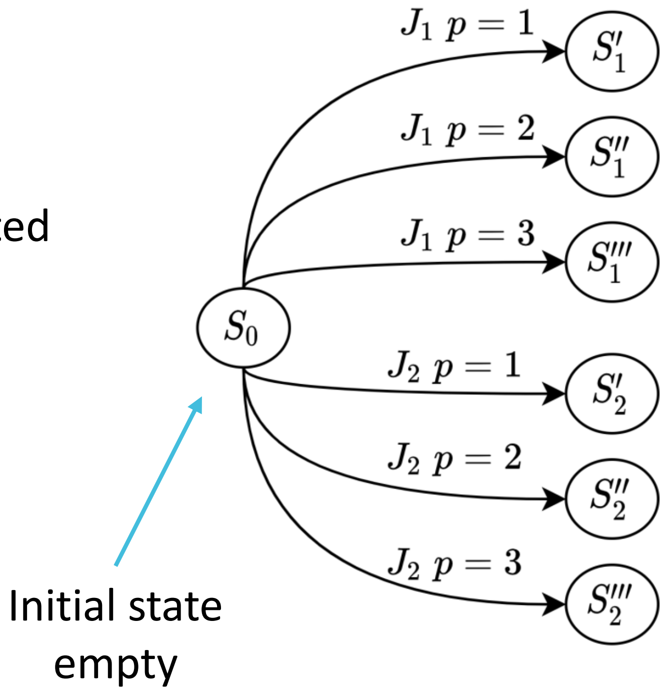
New SAG expansion rules

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



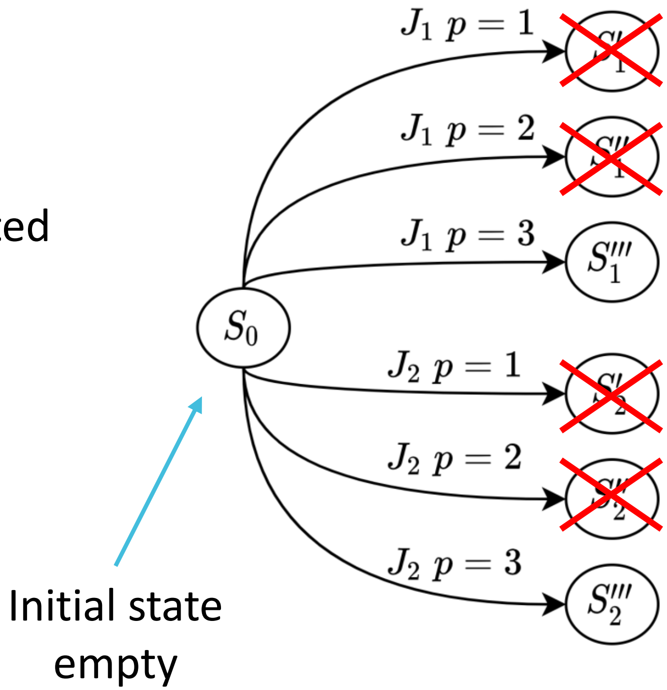
New SAG expansion rules

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



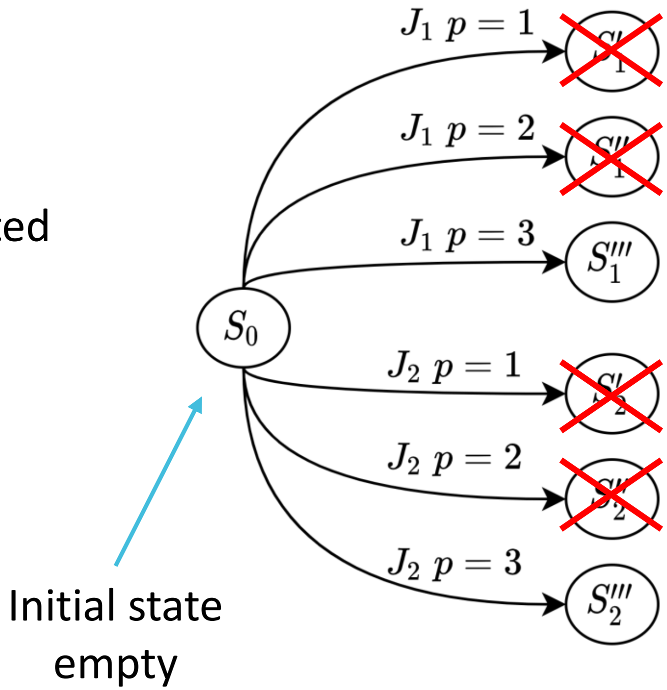
New SAG expansion rules

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



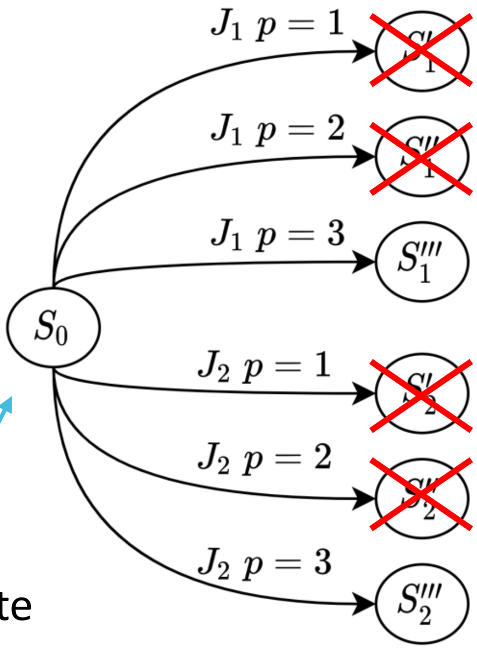
New SAG expansion rules

- Purge states by checking that
  - p cores available
  - More cores **not** available
  - Precedence constraints are respected
- Exploring more states



New SAG expansion rules

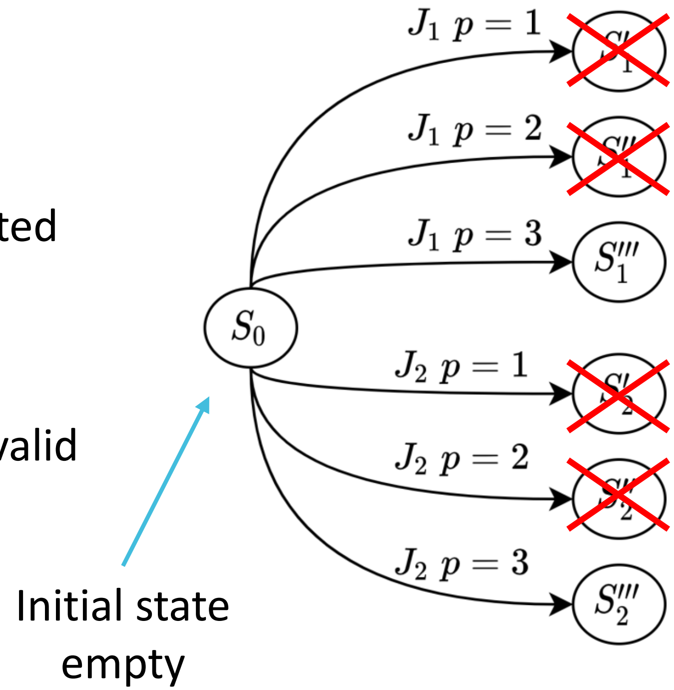
- Purge states by checking that
  - p cores available
  - More cores **not** available
  - Precedence constraints are respected
- Exploring more states
  - Is safe, does not make analysis invalid



Initial state
empty

New SAG expansion rules

- Purge states by checking that
  - p cores available
  - More cores **not** available
  - Precedence constraints are respected
- Exploring more states
  - 👍 Is safe, does not make analysis invalid
  - 👎 Slower and more pessimistic



New SAG merge rules

New SAG merge rules

- We have to merge

New SAG merge rules

- We have to merge
  - Availability times

New SAG merge rules

- We have to merge
  - Availability times
  - Groups of cores

New SAG merge rules

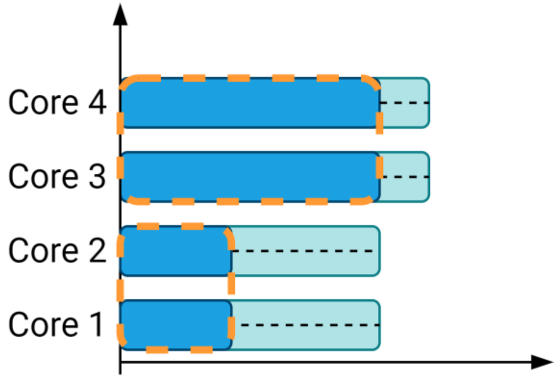
- We have to merge
  - Availability times
  - Groups of cores
  - Certainly running jobs

New SAG merge rules

- We have to merge
  - Availability times → Extend the intervals
  - Groups of cores
  - Certainly running jobs

New SAG merge rules

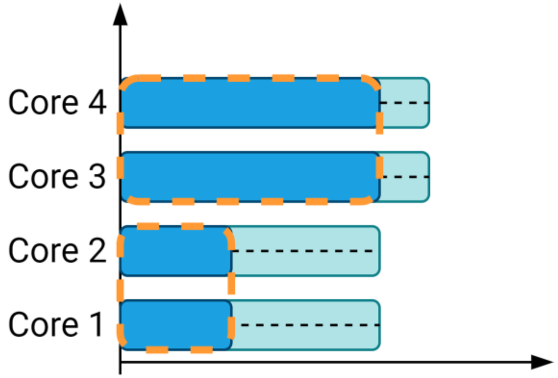
- We have to merge
  - Availability times → Extend the intervals
  - Groups of cores
  - Certainly running jobs



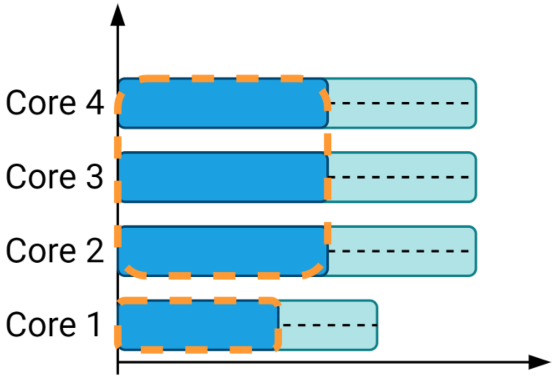
Sate A

New SAG merge rules

- We have to merge
  - Availability times → Extend the intervals
  - Groups of cores
  - Certainly running jobs



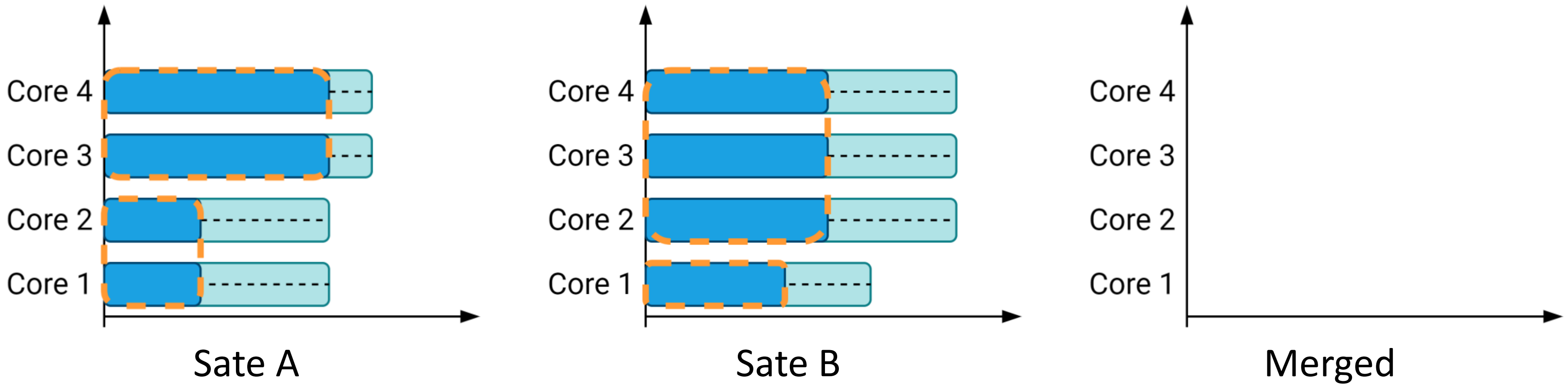
Sate A



Sate B

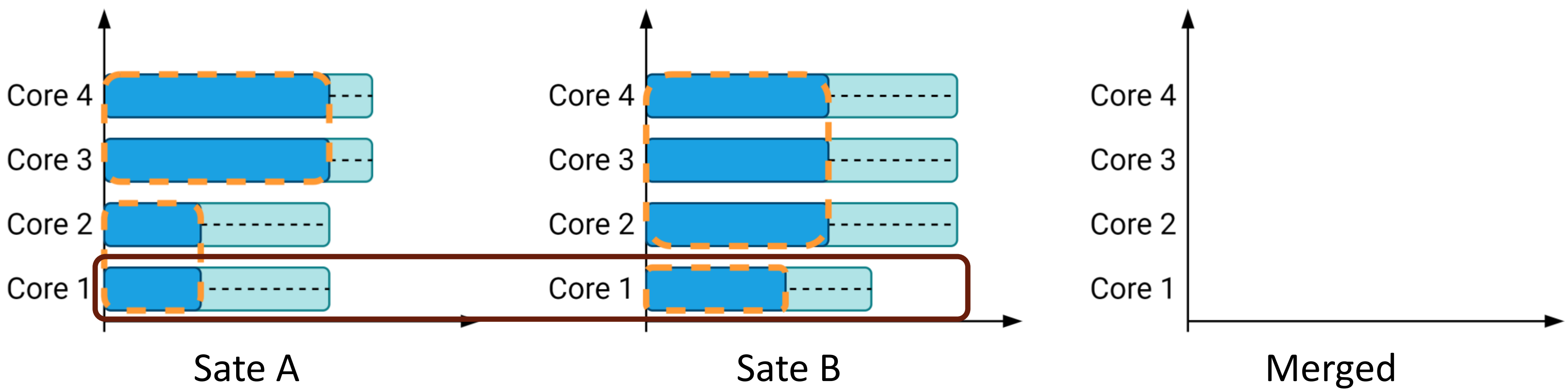
New SAG merge rules

- We have to merge
  - Availability times → Extend the intervals
  - Groups of cores
  - Certainly running jobs



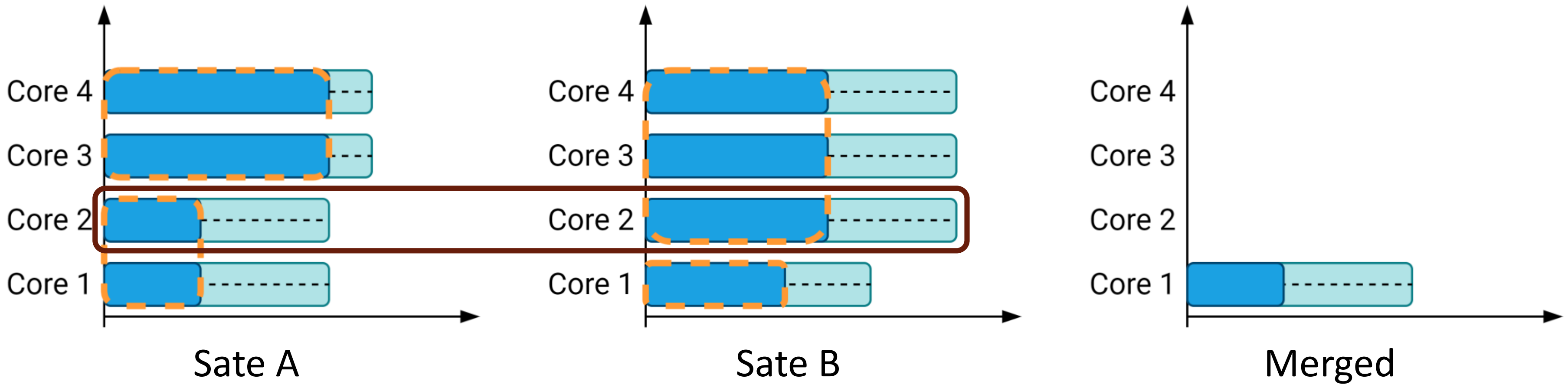
New SAG merge rules

- We have to merge
  - Availability times → Extend the intervals
  - Groups of cores
  - Certainly running jobs



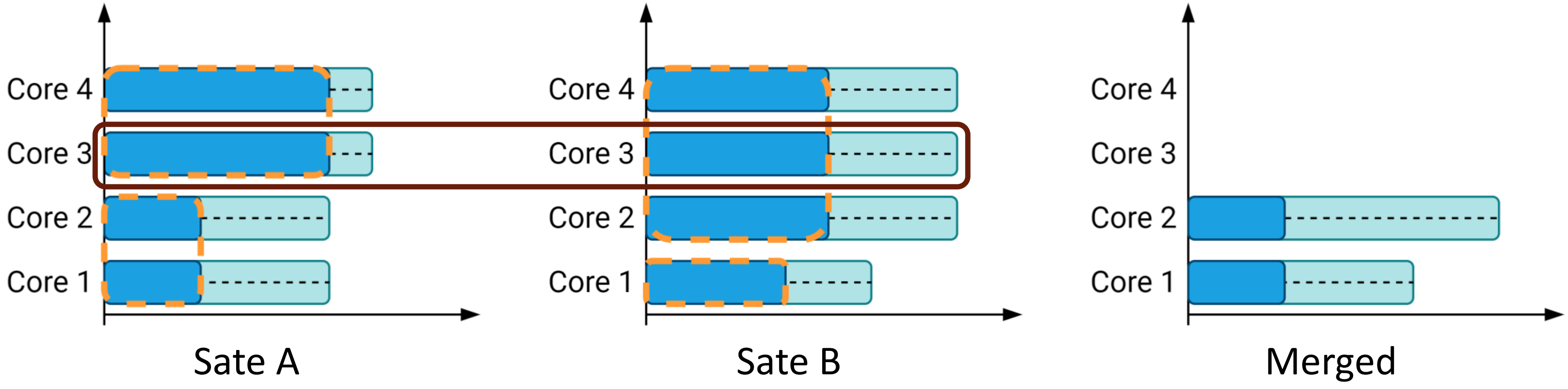
New SAG merge rules

- We have to merge
  - Availability times → Extend the intervals
  - Groups of cores
  - Certainly running jobs



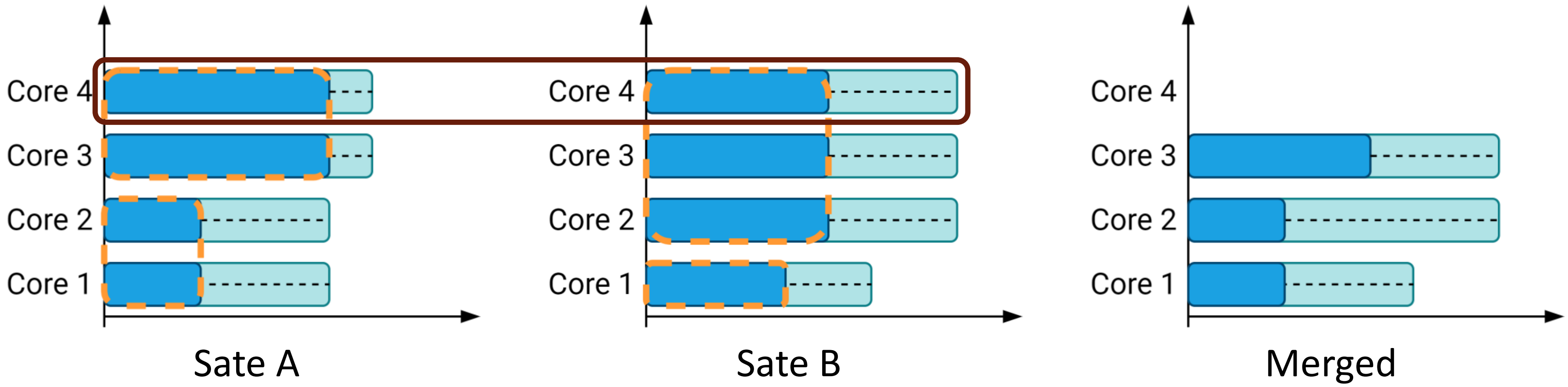
New SAG merge rules

- We have to merge
  - Availability times → Extend the intervals
  - Groups of cores
  - Certainly running jobs



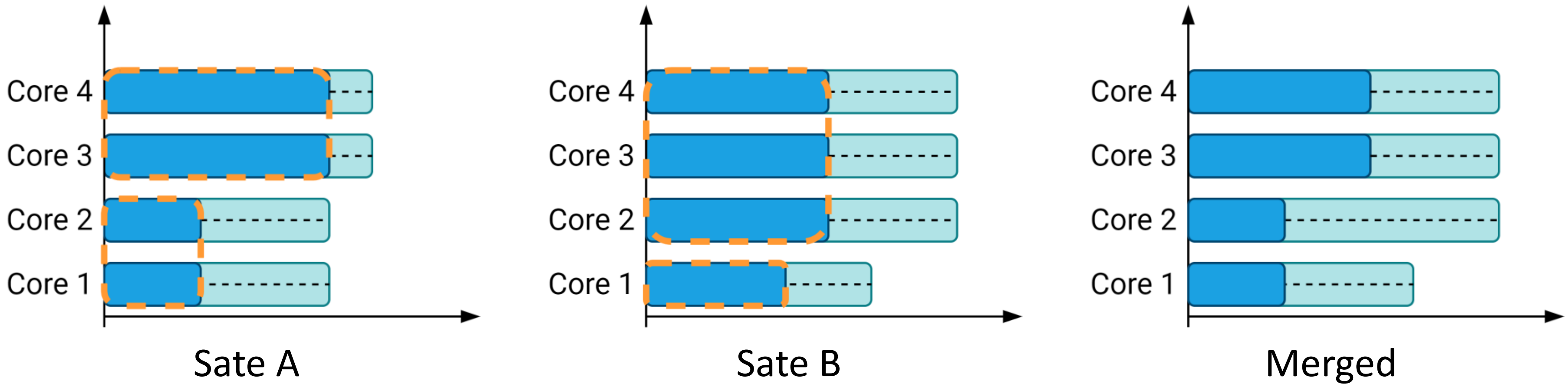
New SAG merge rules

- We have to merge
  - Availability times → Extend the intervals
  - Groups of cores
  - Certainly running jobs



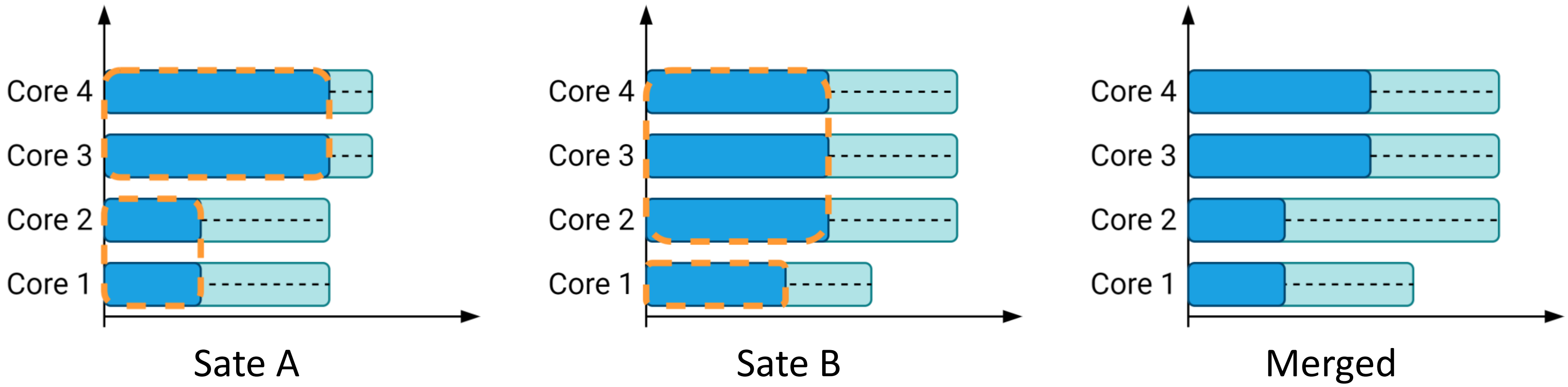
New SAG merge rules

- We have to merge
  - Availability times → Extend the intervals
  - Groups of cores
  - Certainly running jobs



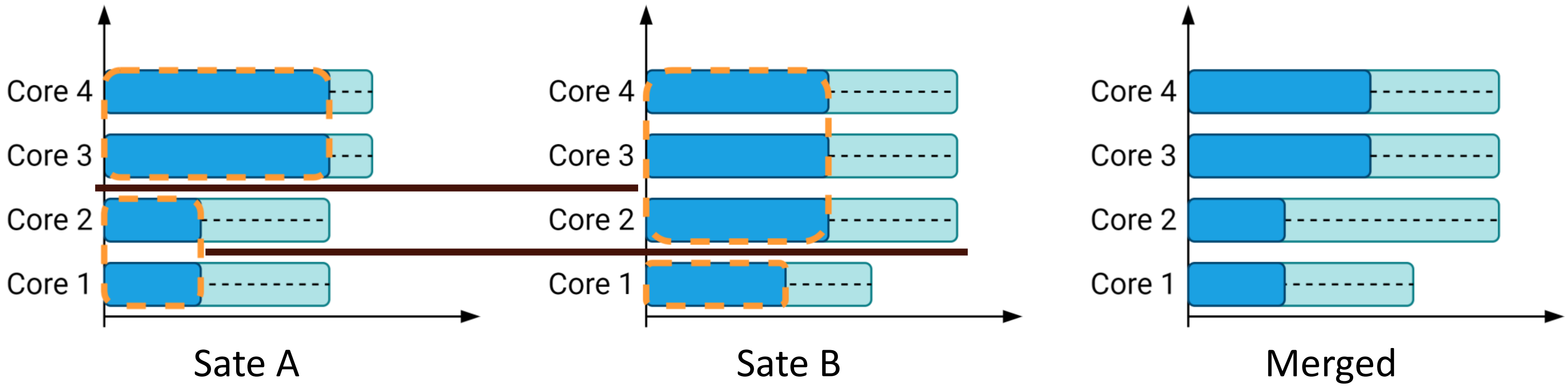
New SAG merge rules

- We have to merge
  - Availability times → Extend the intervals
  - Groups of cores → Break the groups into same size
  - Certainly running jobs



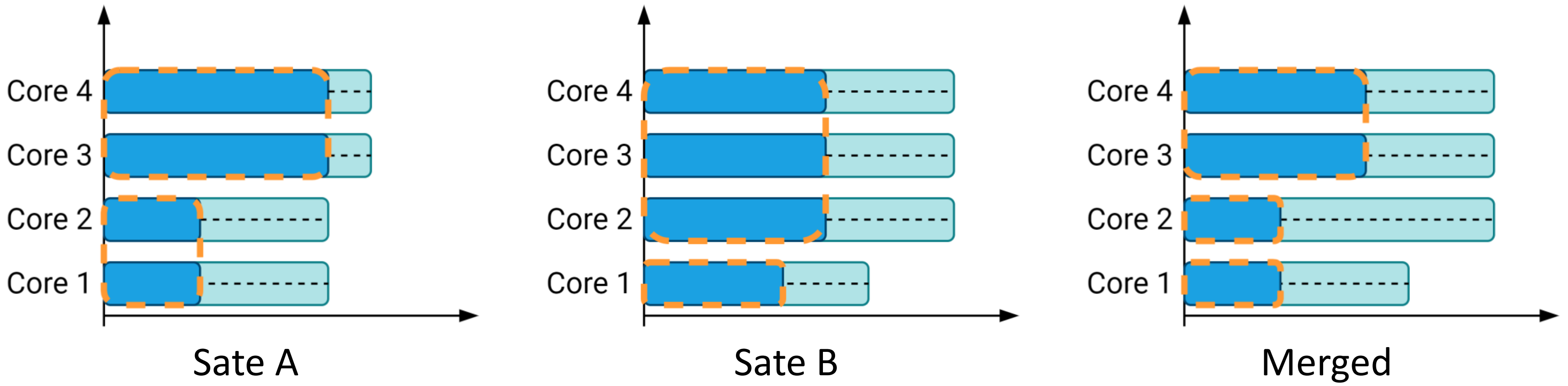
New SAG merge rules

- We have to merge
  - Availability times → Extend the intervals
  - Groups of cores → Break the groups into same size
  - Certainly running jobs



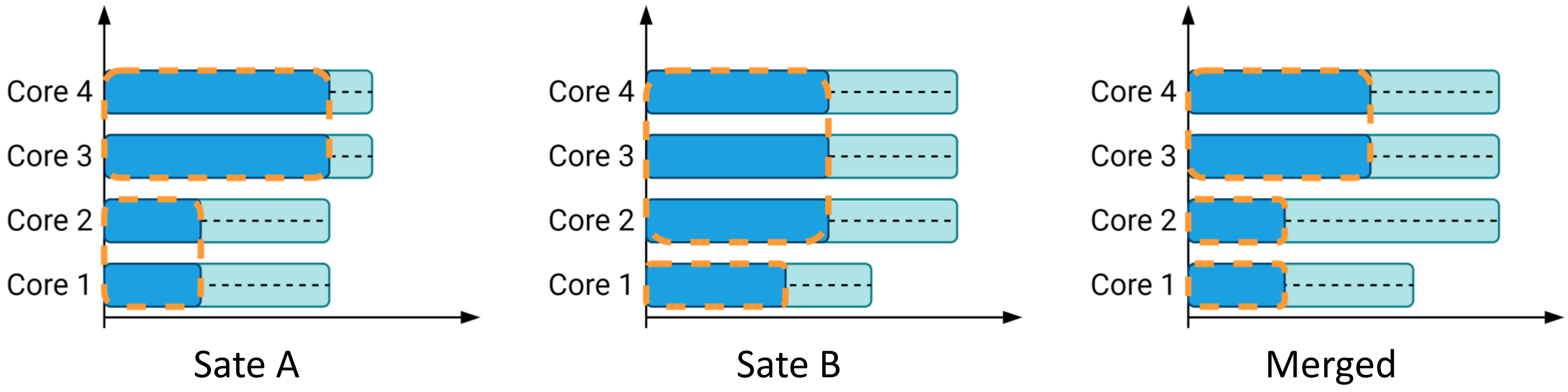
New SAG merge rules

- We have to merge
  - Availability times → Extend the intervals
  - Groups of cores → Break the groups into same size
  - Certainly running jobs



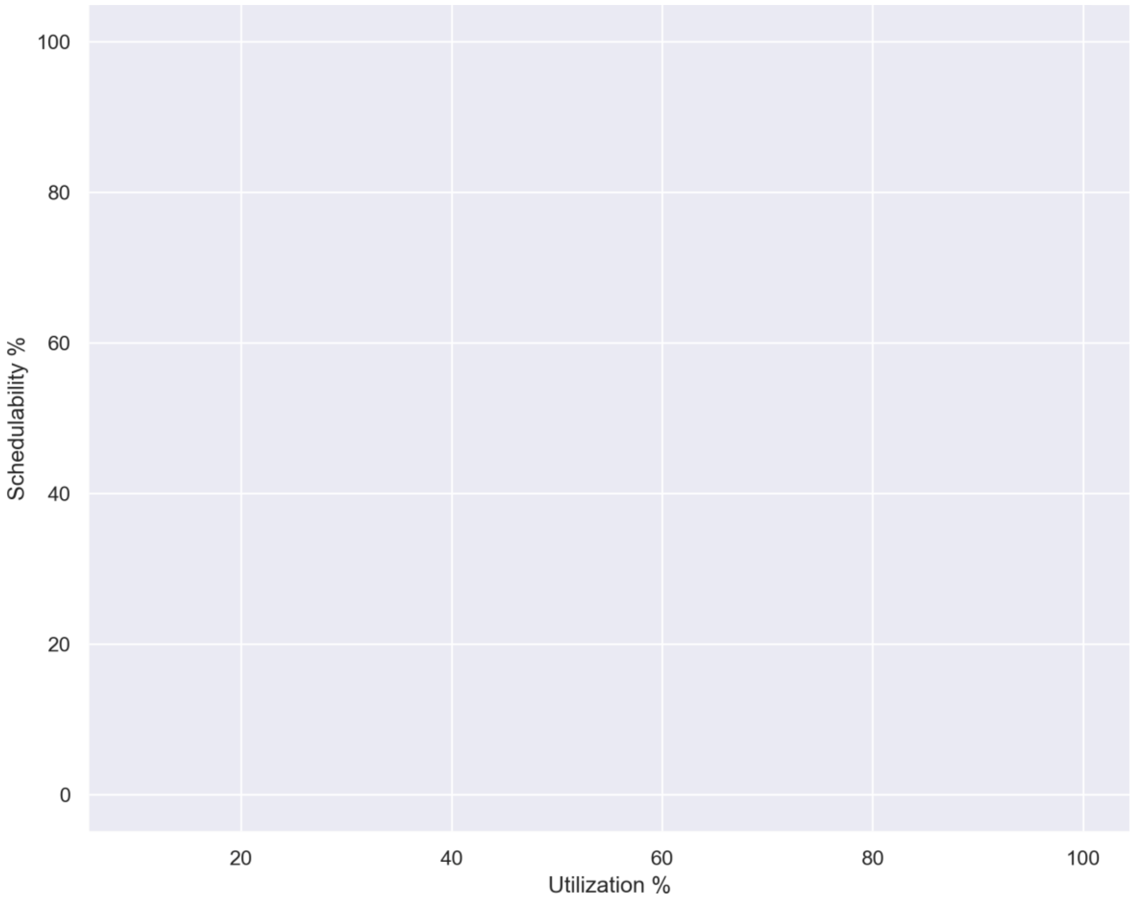
New SAG merge rules

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



Our analysis results

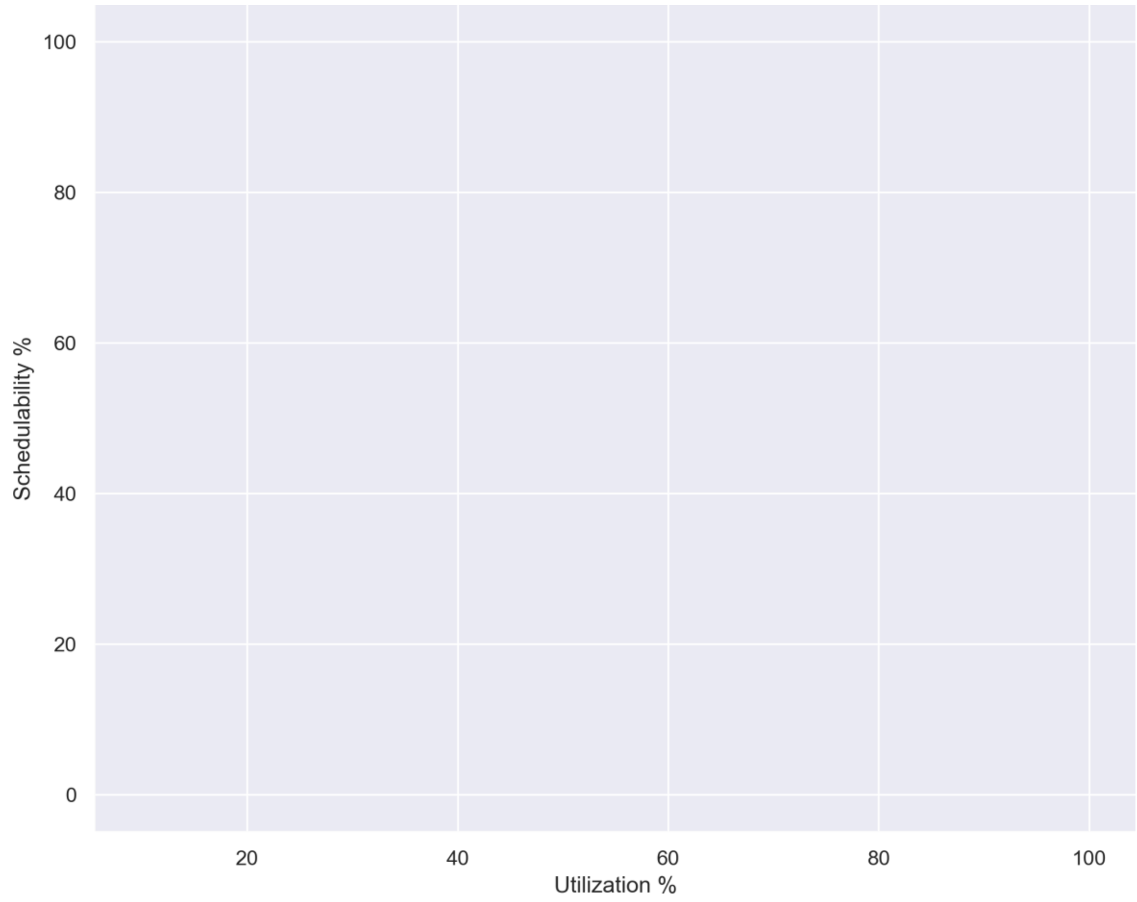
Our analysis results



Our analysis results

Randomly generated task sets

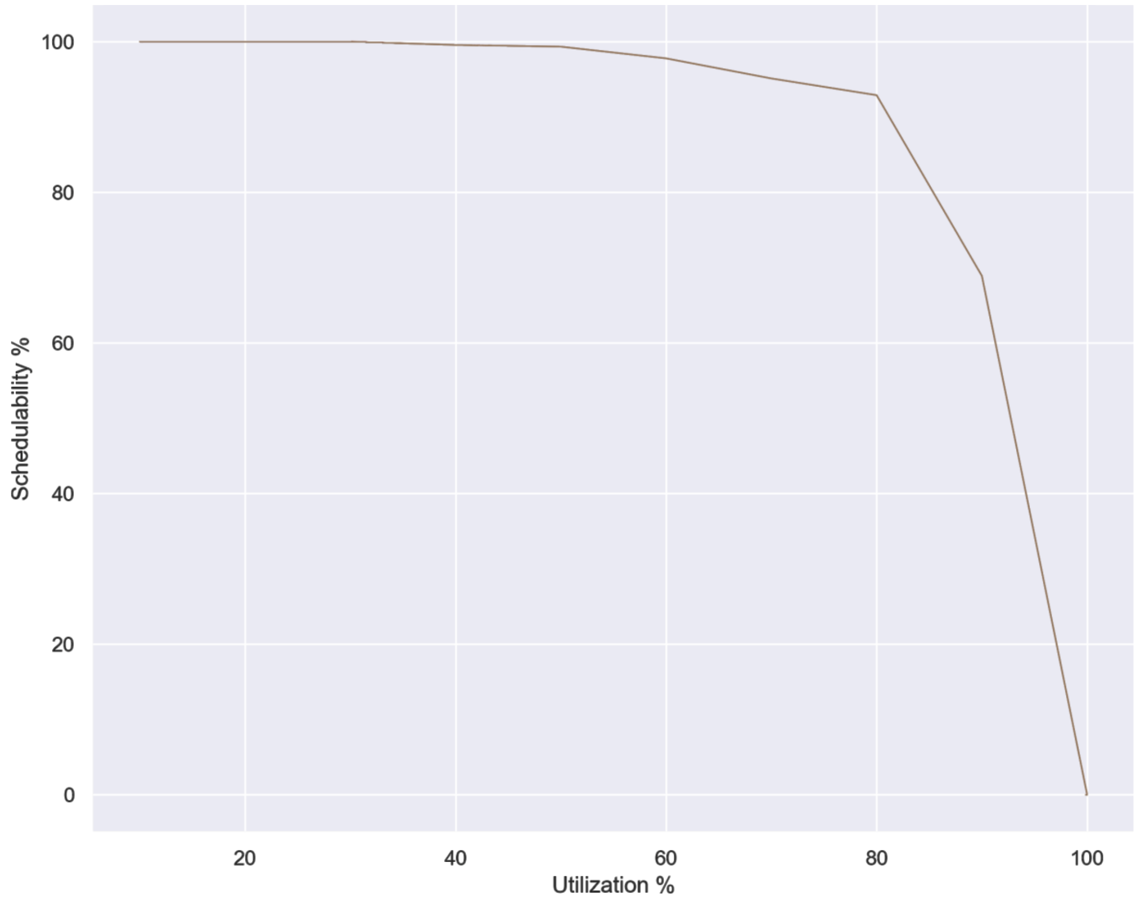
- System processors: 8
- System tasks: 20 rigid tasks
- Execution time variation: 50%



Our analysis results

Randomly generated task sets

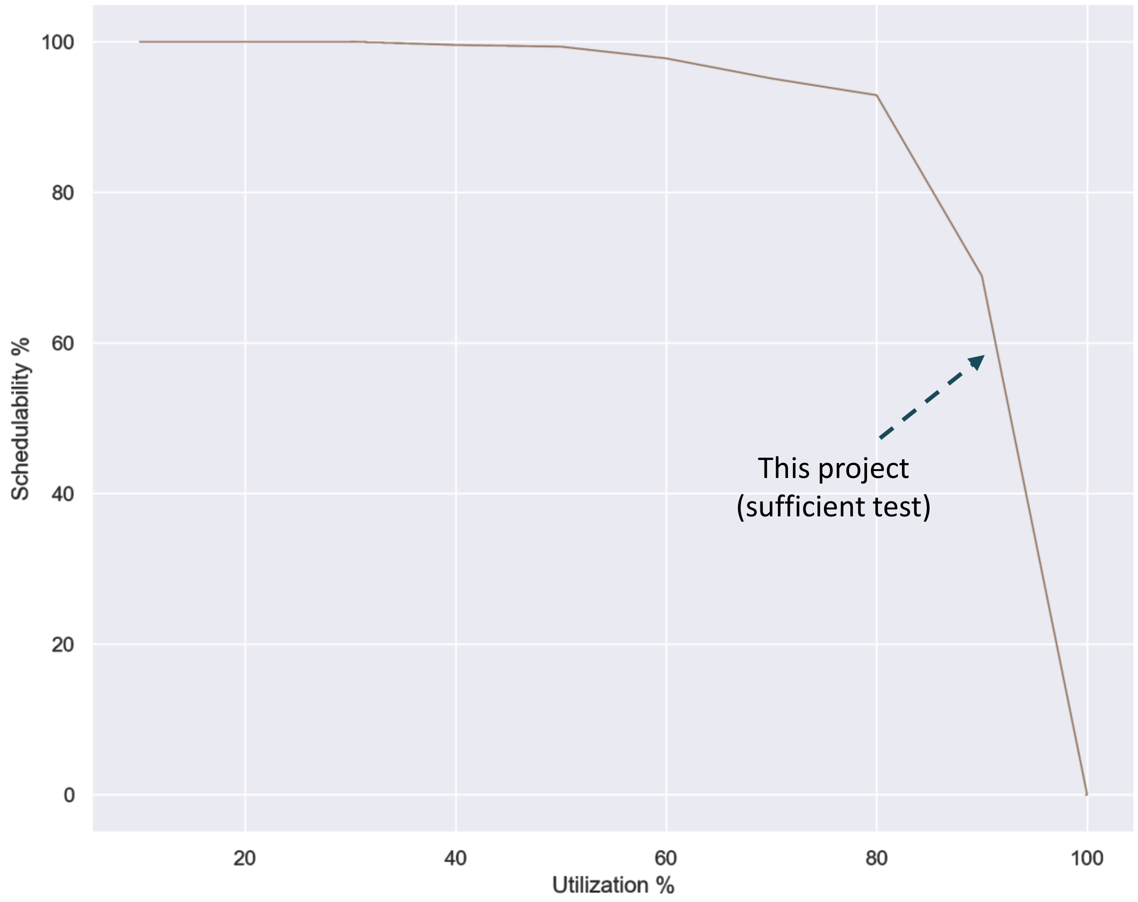
- System processors: 8
- System tasks: 20 rigid tasks
- Execution time variation: 50%



Our analysis results

Randomly generated task sets

- System processors: 8
- System tasks: 20 rigid tasks
- Execution time variation: 50%

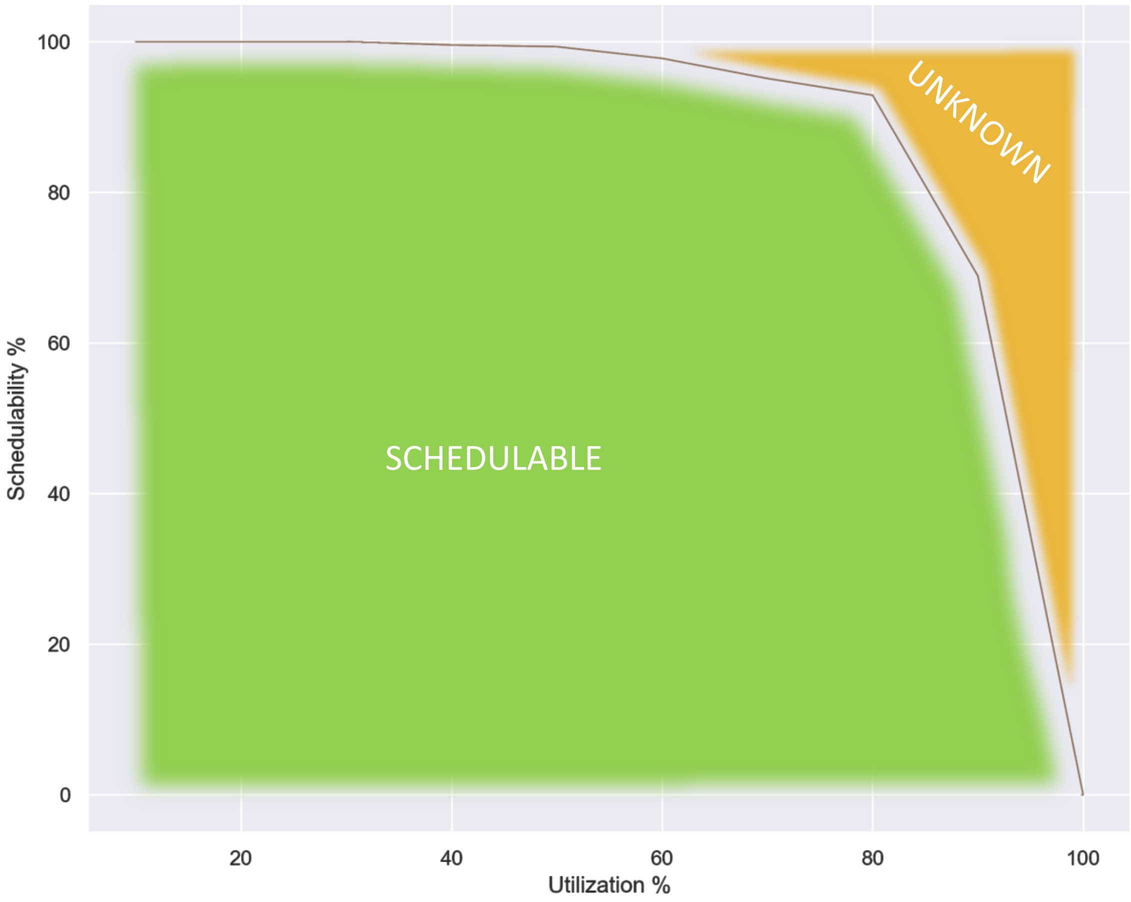


TUDelft

Our analysis results

Randomly generated task sets

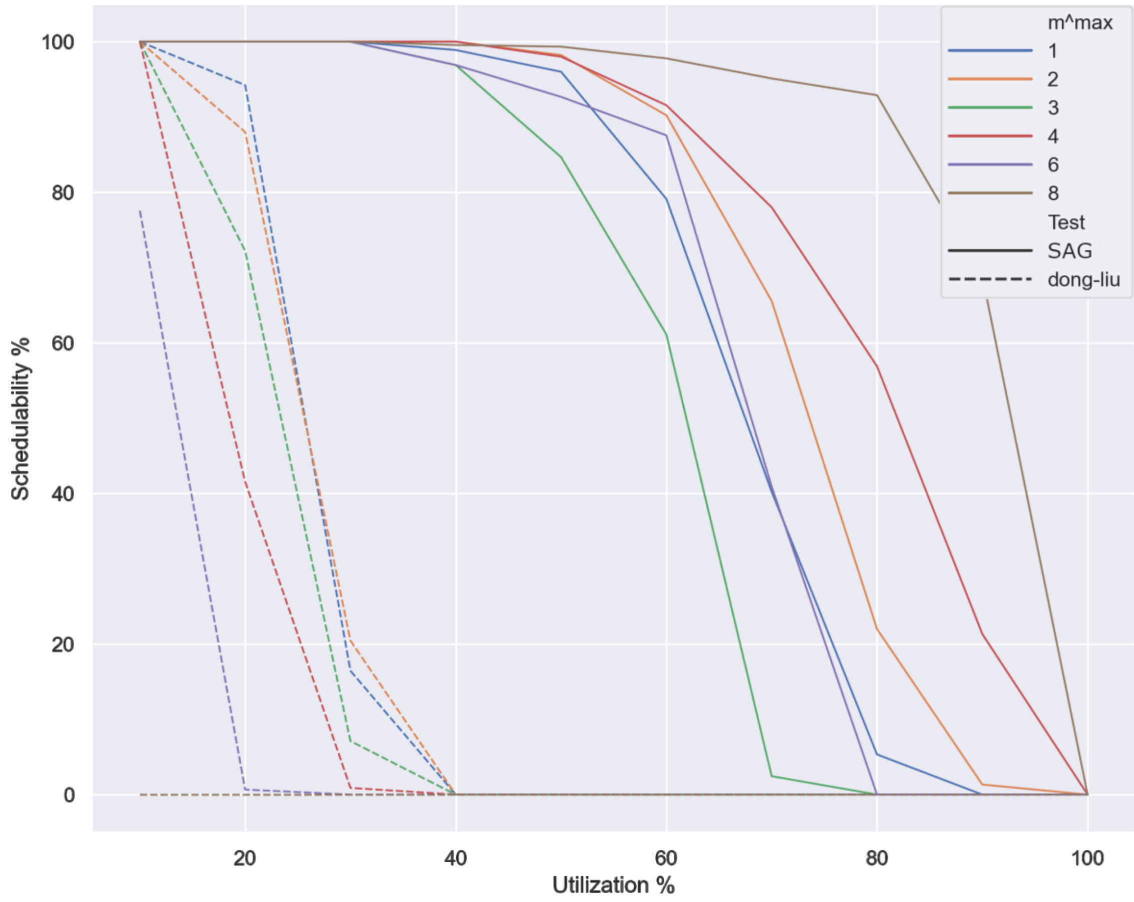
- System processors: 8
- System tasks: 20 rigid tasks
- Execution time variation: 50%



Our analysis results

Randomly generated task sets

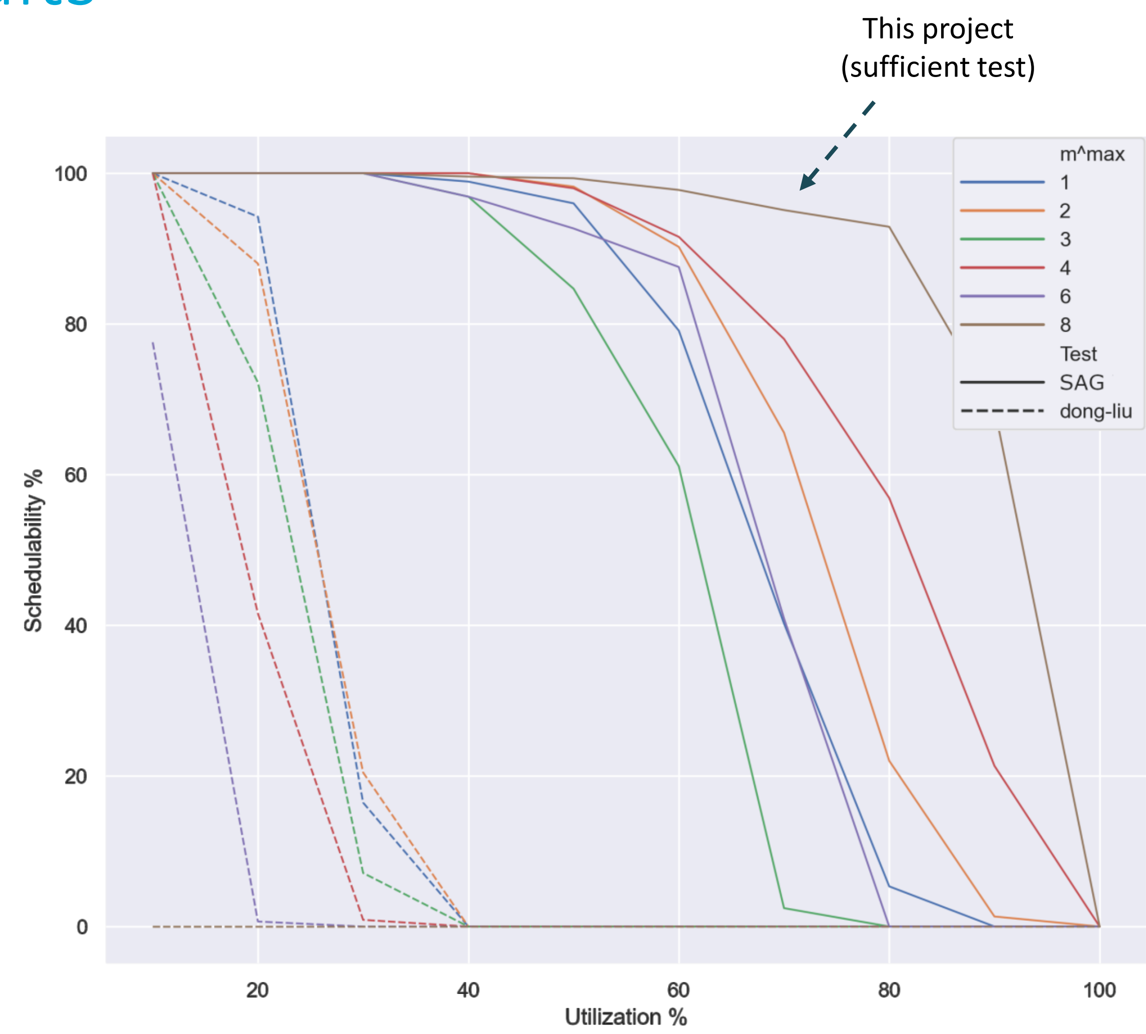
- System processors: 8
- System tasks: 20 rigid tasks
- Execution time variation: 50%



Our analysis results

Randomly generated task sets

- System processors: 8
- System tasks: 20 rigid tasks
- Execution time variation: 50%

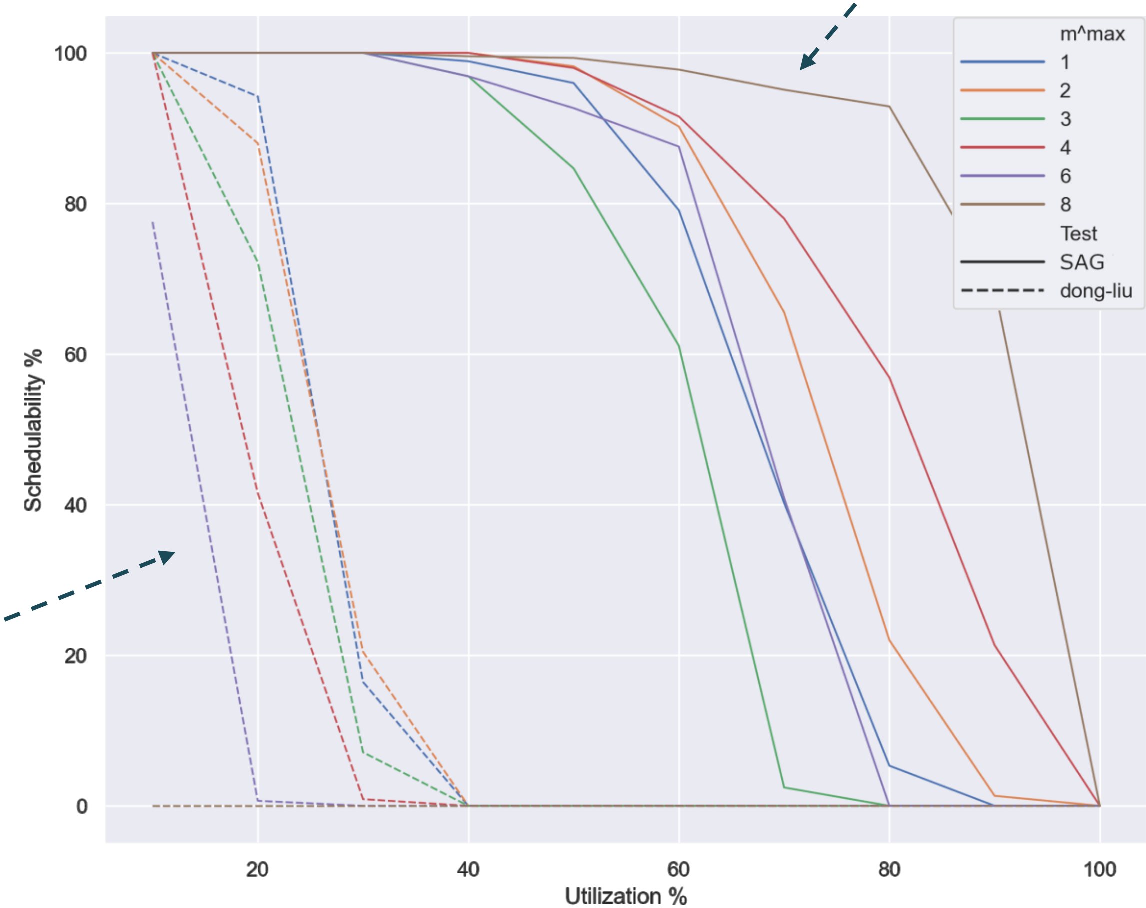


Our analysis results

Randomly generated task sets

- System processors: 8
- System tasks: 20 rigid tasks
- Execution time variation: 50%

This project
(sufficient test)

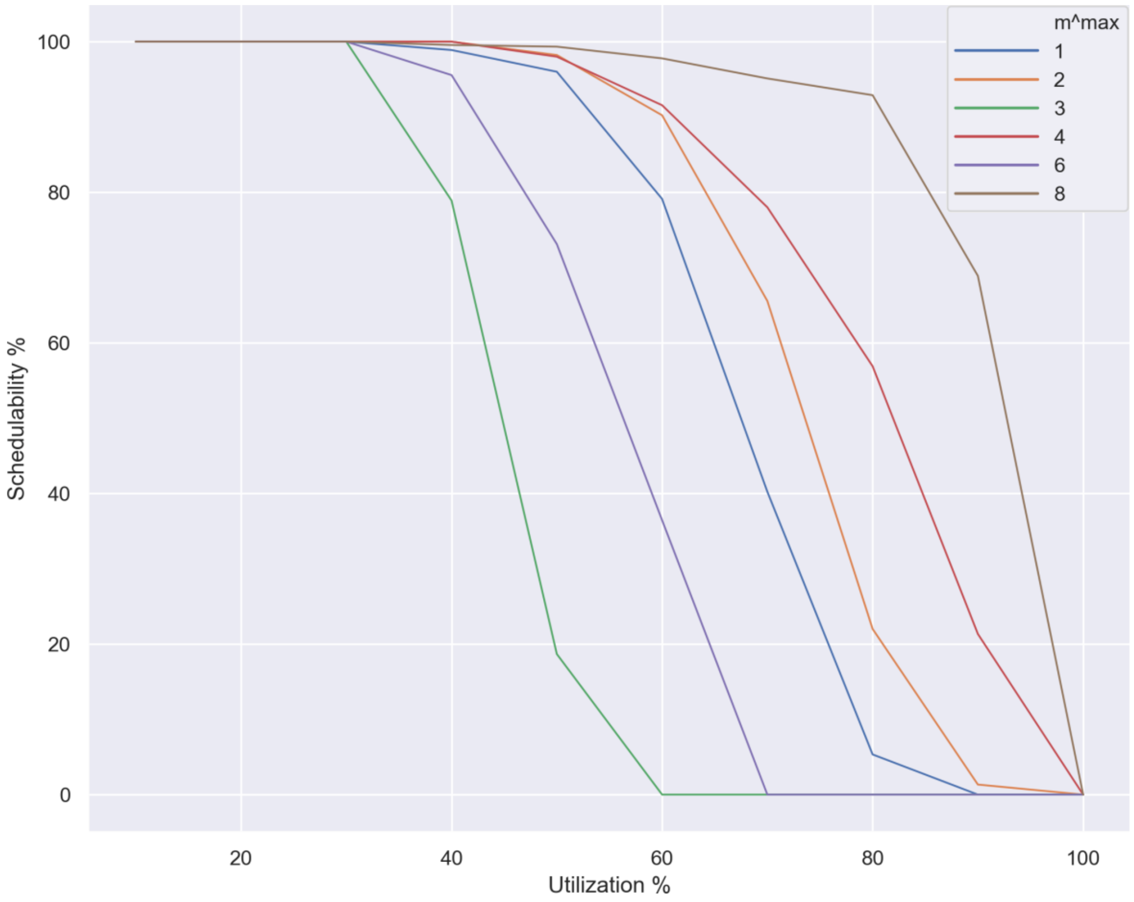


State-of-the-art
(sufficient test)

Our analysis results

Randomly generated task sets

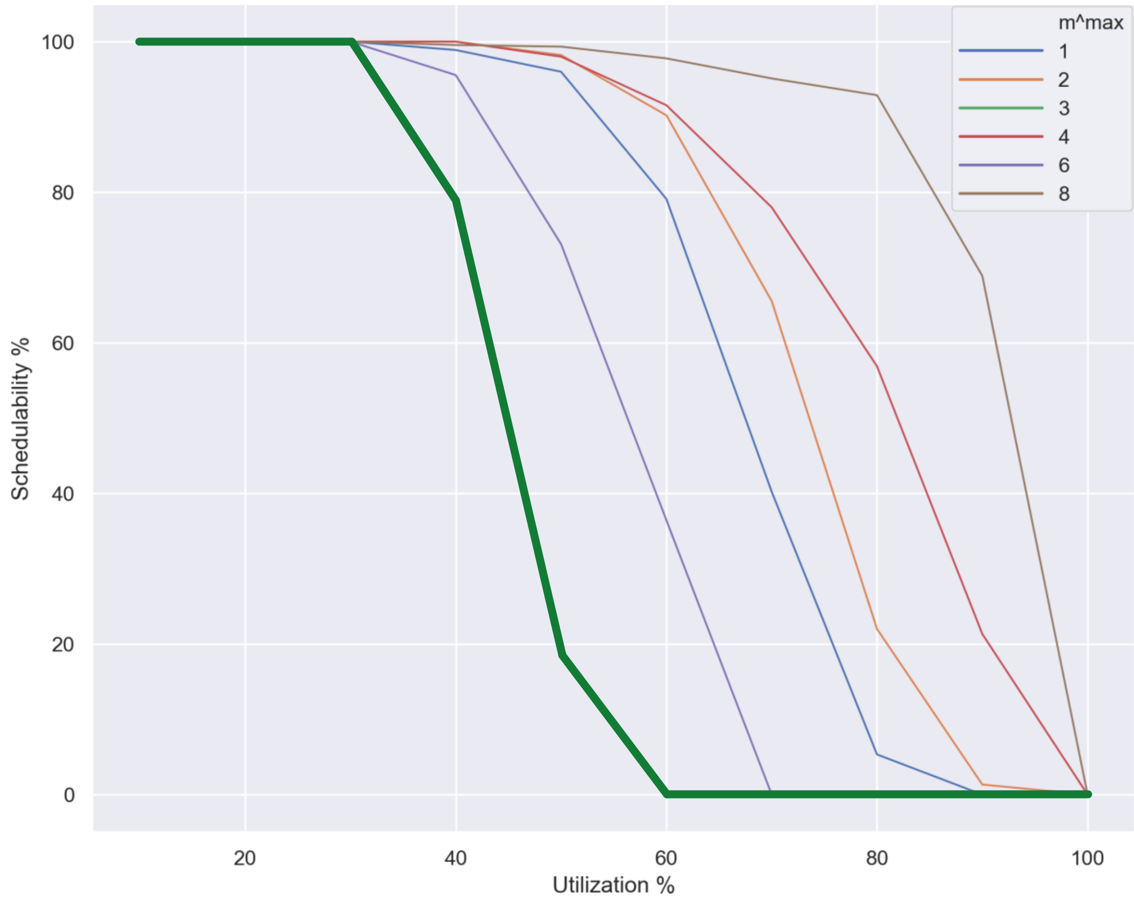
- System processors: 8
- System tasks: 20 moldable tasks
- Execution time variation: 50%



Our analysis results

Randomly generated task sets

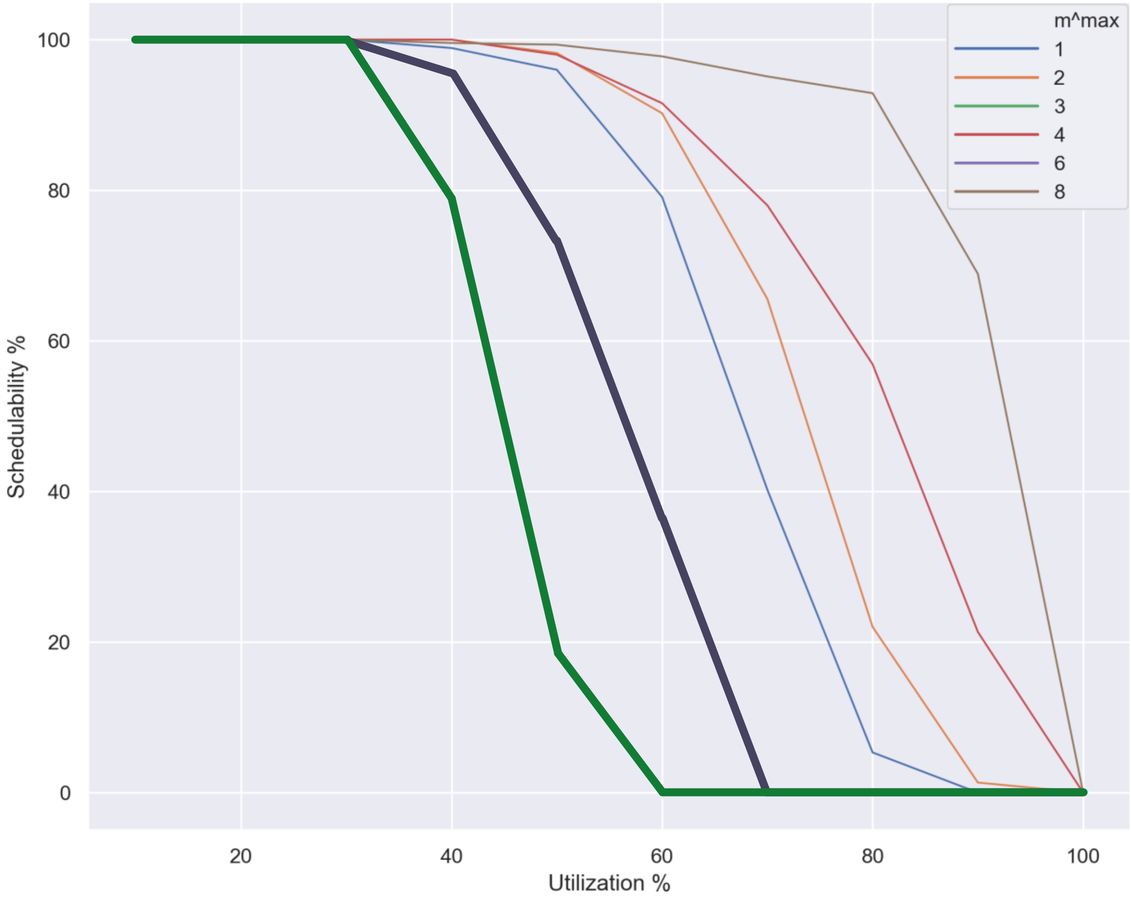
- System processors: 8
- System tasks: 20 moldable tasks
- Execution time variation: 50%



Our analysis results

Randomly generated task sets

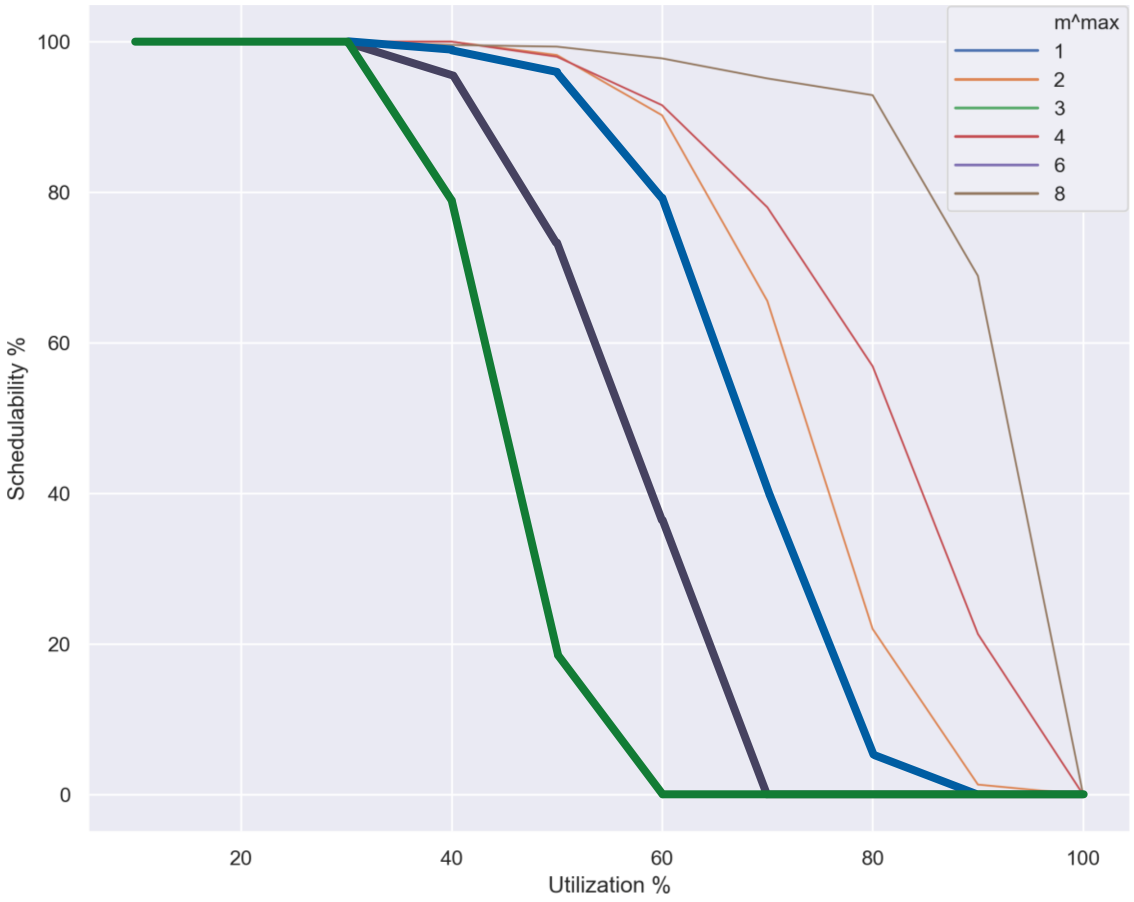
- System processors: 8
- System tasks: 20 moldable tasks
- Execution time variation: 50%



Our analysis results

Randomly generated task sets

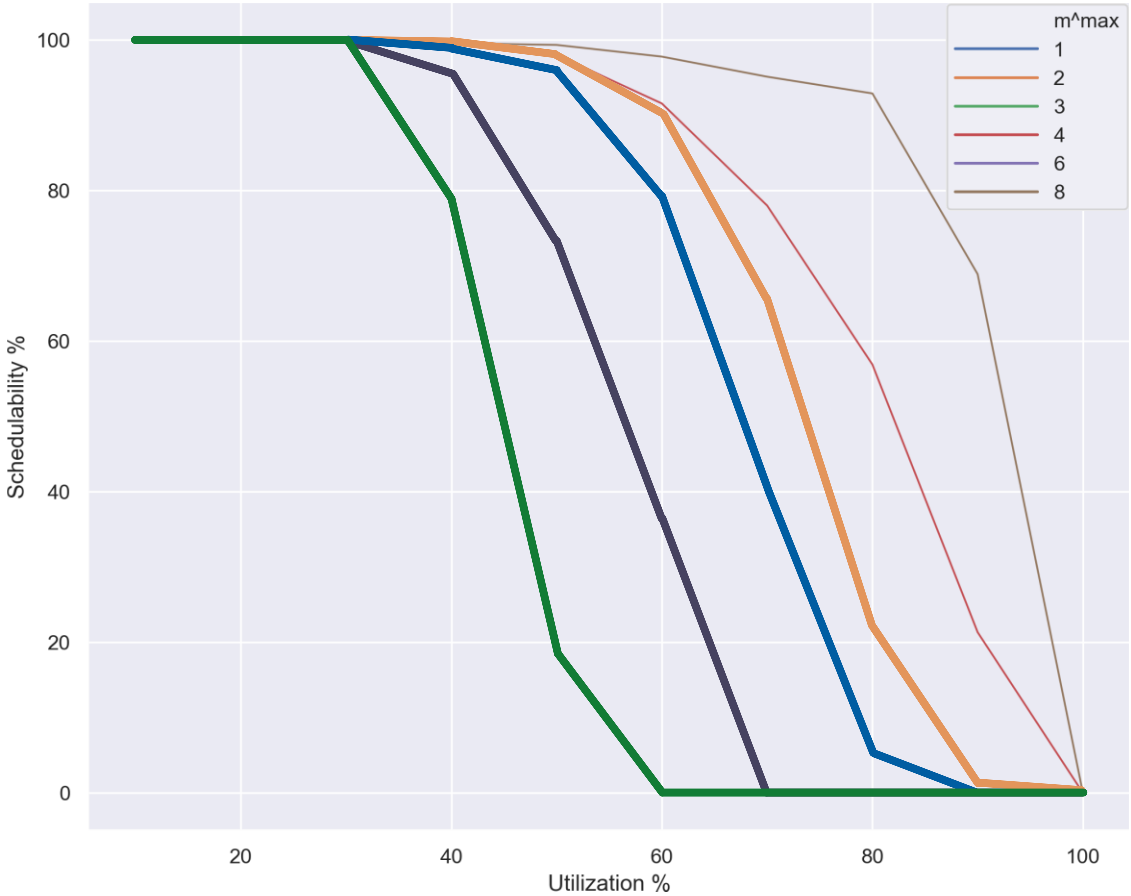
- System processors: 8
- System tasks: 20 moldable tasks
- Execution time variation: 50%



Our analysis results

Randomly generated task sets

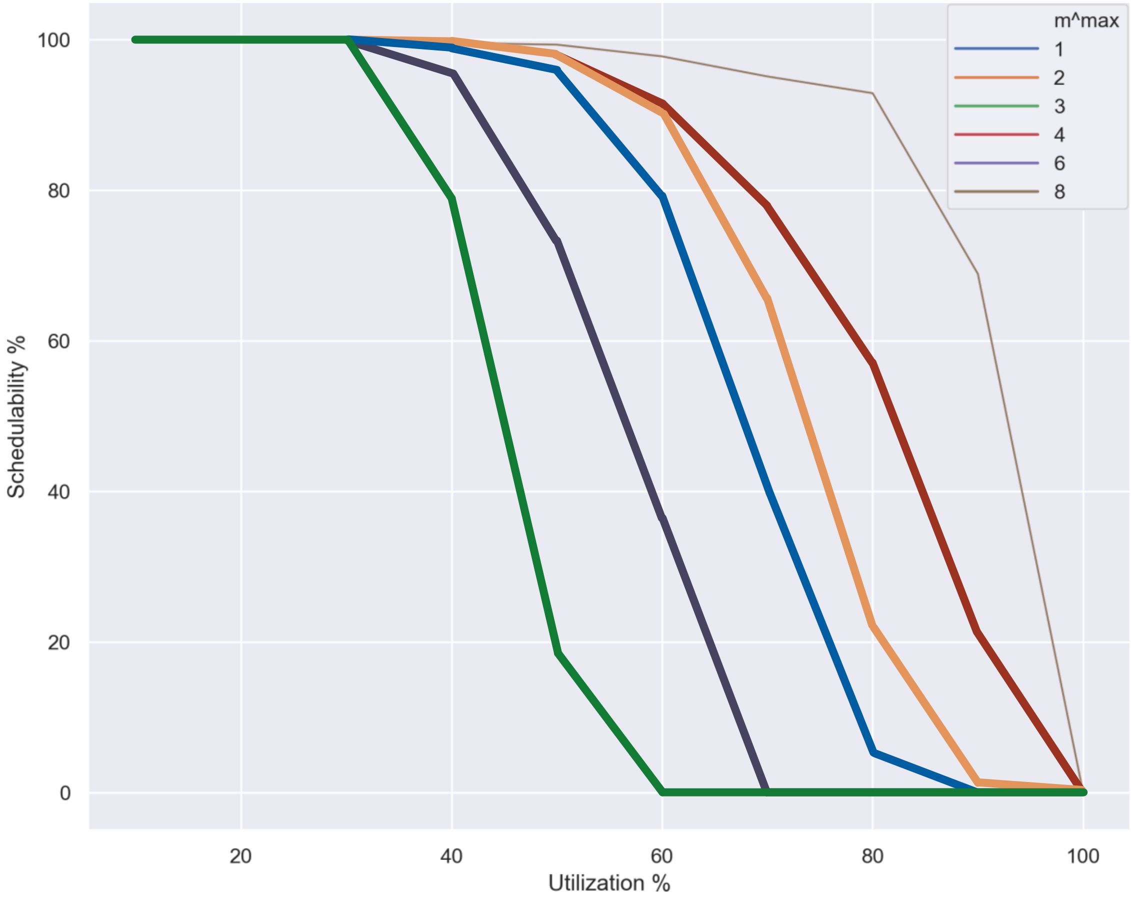
- System processors: 8
- System tasks: 20 moldable tasks
- Execution time variation: 50%



Our analysis results

Randomly generated task sets

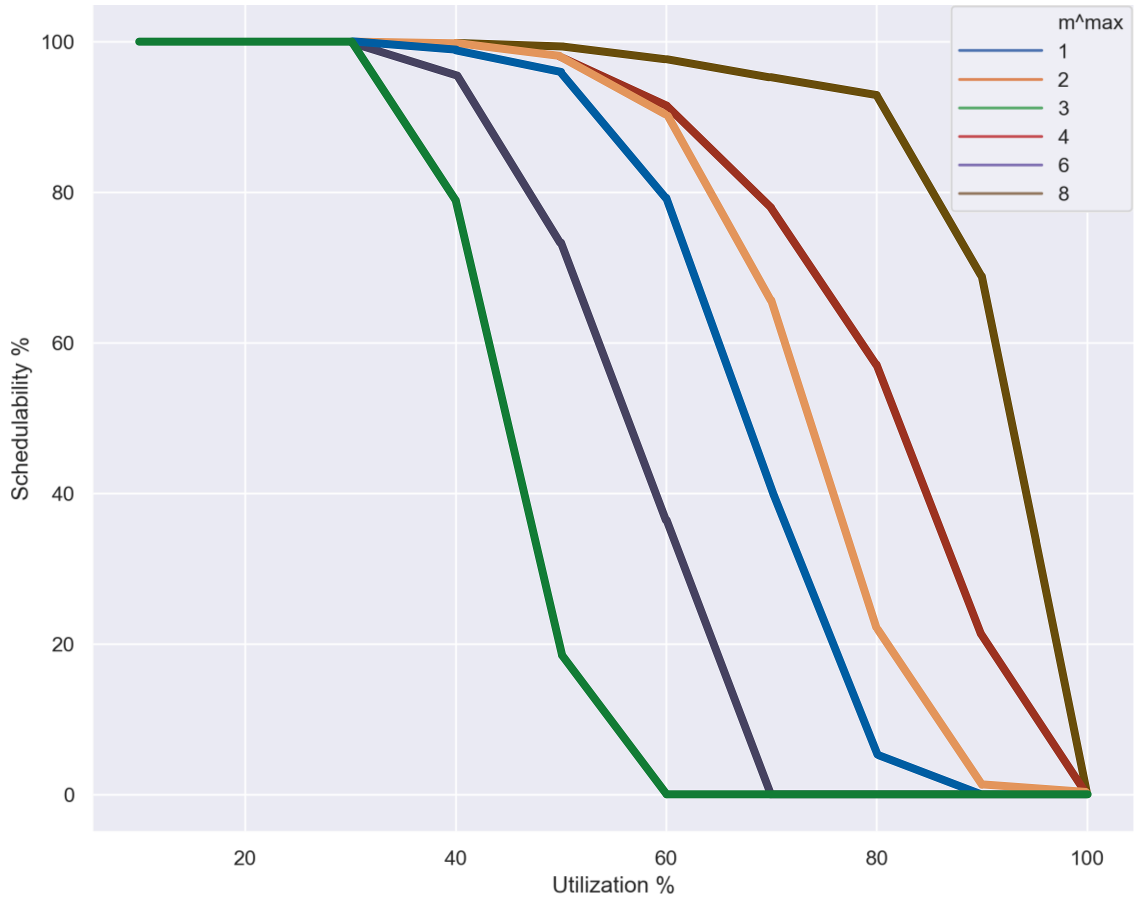
- System processors: 8
- System tasks: 20 moldable tasks
- Execution time variation: 50%



Our analysis results

Randomly generated task sets

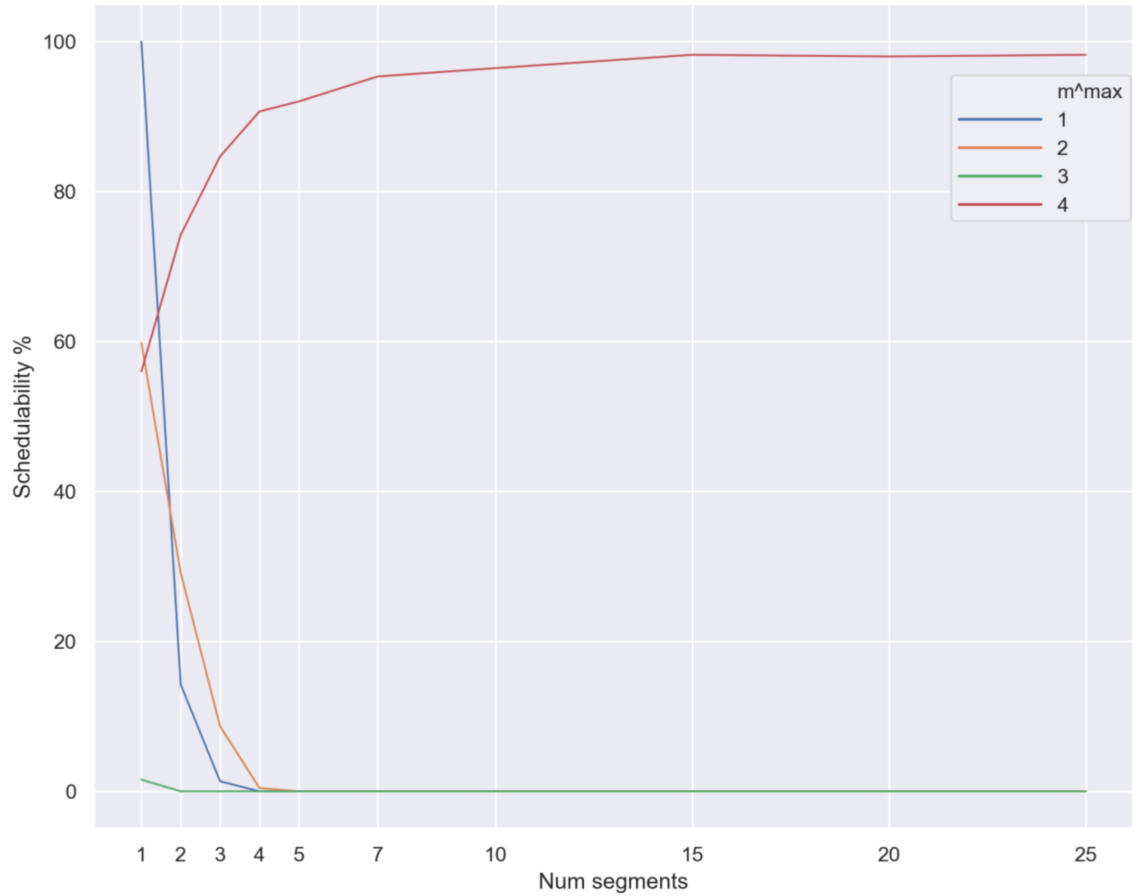
- System processors: 8
- System tasks: 20 moldable tasks
- Execution time variation: 50%



Our analysis results

Randomly generated task sets

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



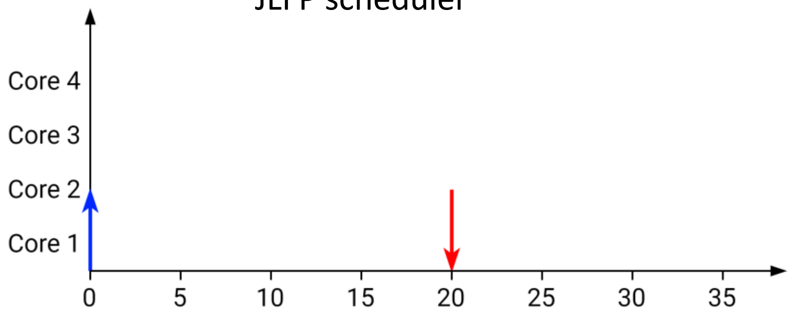
Agenda

- ~~Gang schedulability analysis~~
- New scheduling policy

JLFP limitations with moldable gang

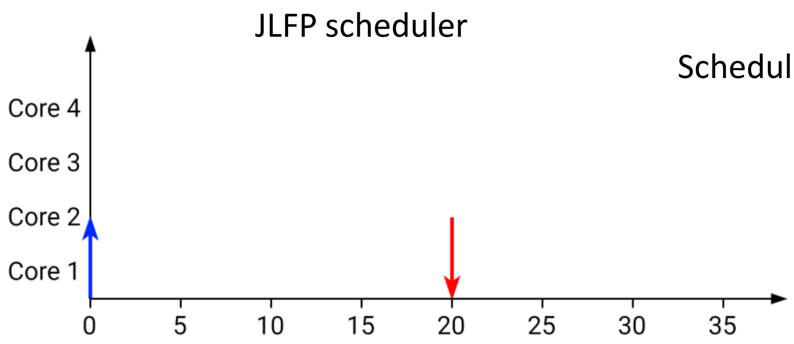
JLFP limitations with moldable gang

JLFP 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

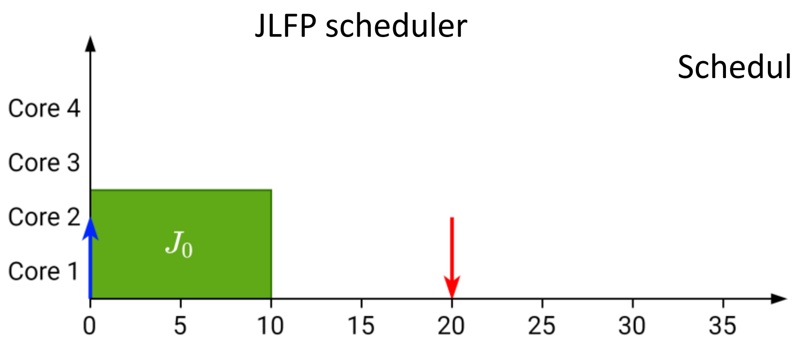
JLFP limitations with moldable gang



Scheduling: J_0

	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

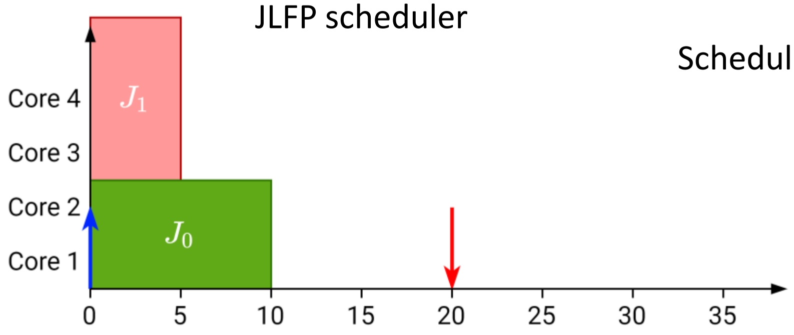
JLFP limitations with moldable gang



Scheduling: J_1

	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

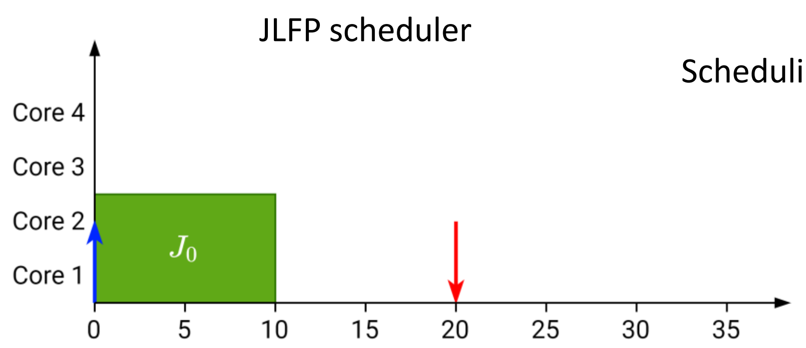
JLFP limitations with moldable gang



Scheduling: J_1

	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

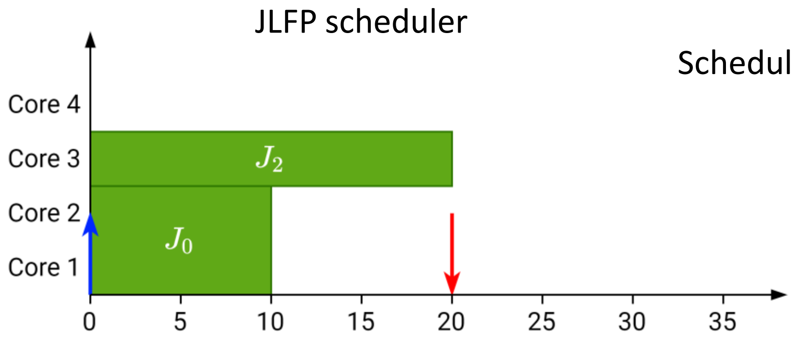
JLFP limitations with moldable gang



Scheduling: J_2

	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

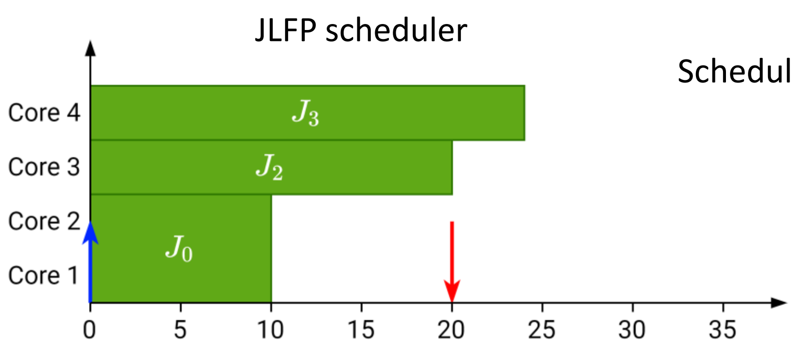
JLFP limitations with moldable gang



Scheduling: J_3

	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

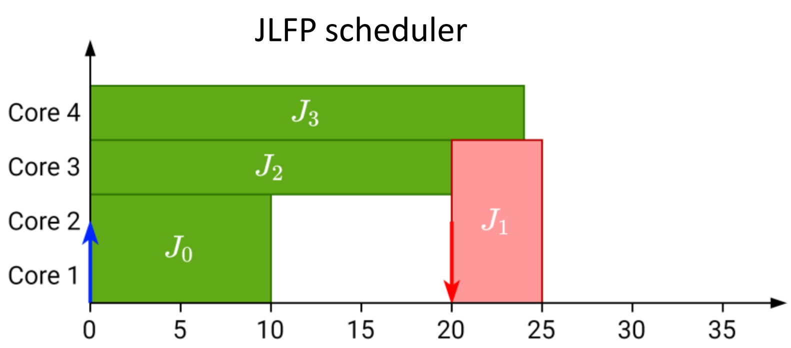
JLFP limitations with moldable gang



Scheduling: J_1

	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

JLFP limitations with moldable gang

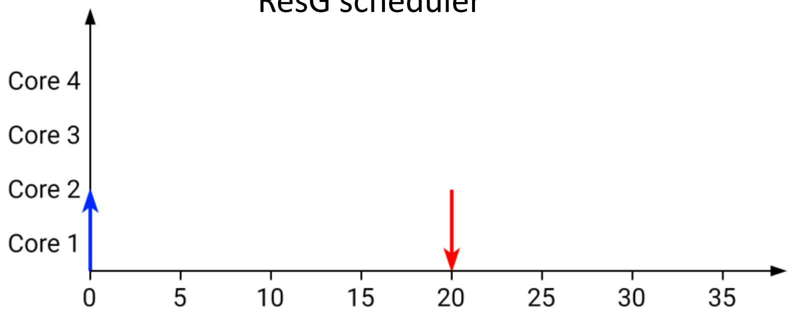


	Priority	Cores	Deadline	Execution time
J_0	High	2	∞	10
J_1	Mid-high	3	20	5
J_2	Mid-low	1	∞	20
J_3	Low	1	∞	20

Reservation-based gang scheduler

- Reservation-based

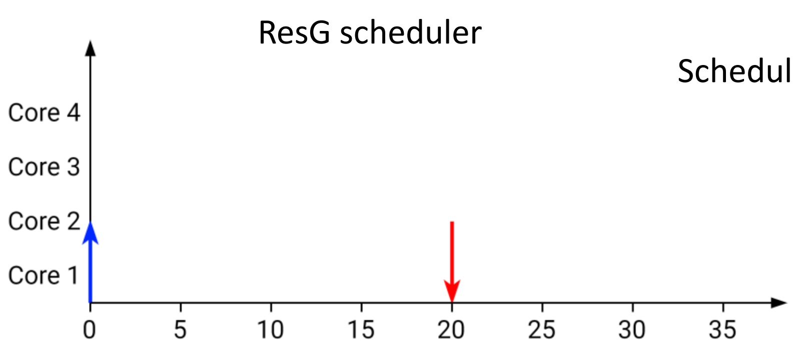
ResG scheduler



	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

Reservation-based gang scheduler

- Reservation-based

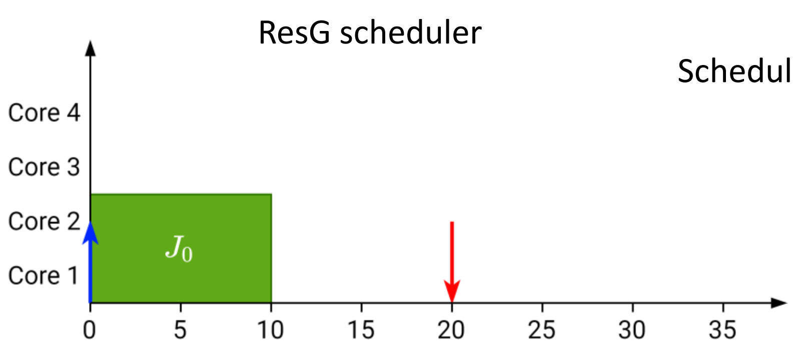


Scheduling: J_0

	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

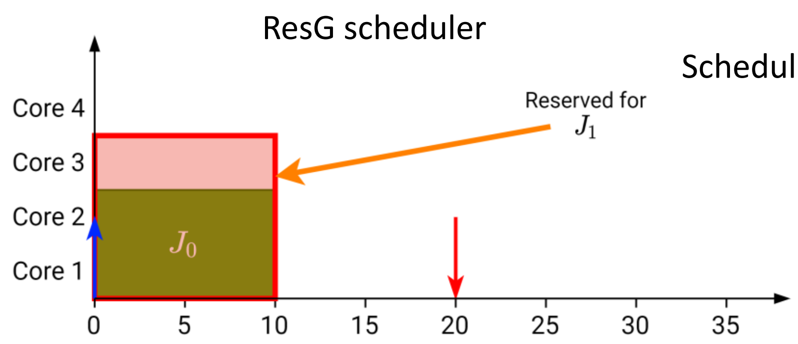


Scheduling: J_1

	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

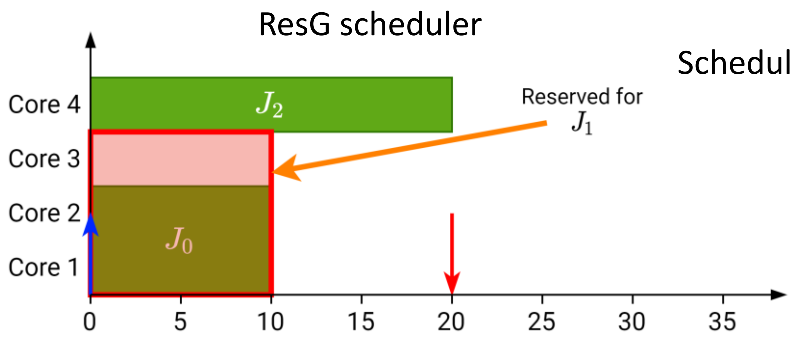


Scheduling: J_2

	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

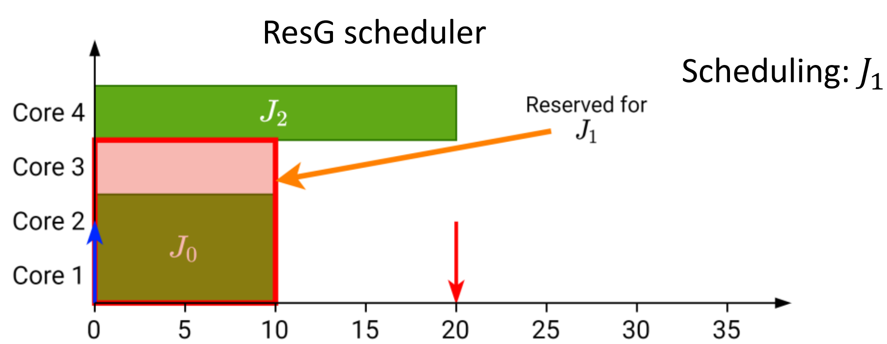


Scheduling: J_3

	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

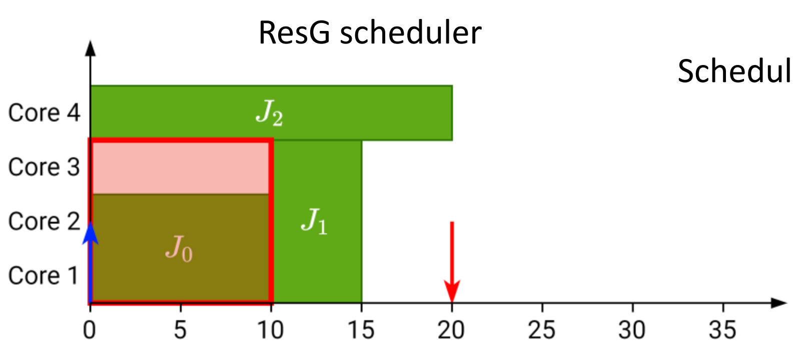
- Reservation-based
- Reserve cores of higher-priority tasks and distribute the remaining ones among lower priority tasks



	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

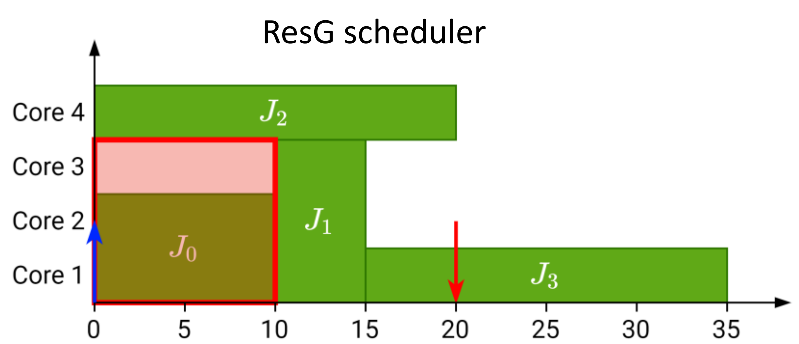


Scheduling: J_3

	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

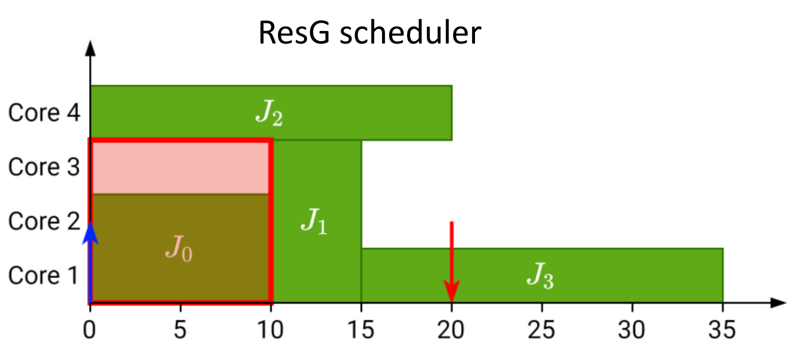
- Reservation-based
- Reserve cores of higher-priority tasks and distribute the remaining ones among lower priority tasks



	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

Results JLFP vs ResG in simulator

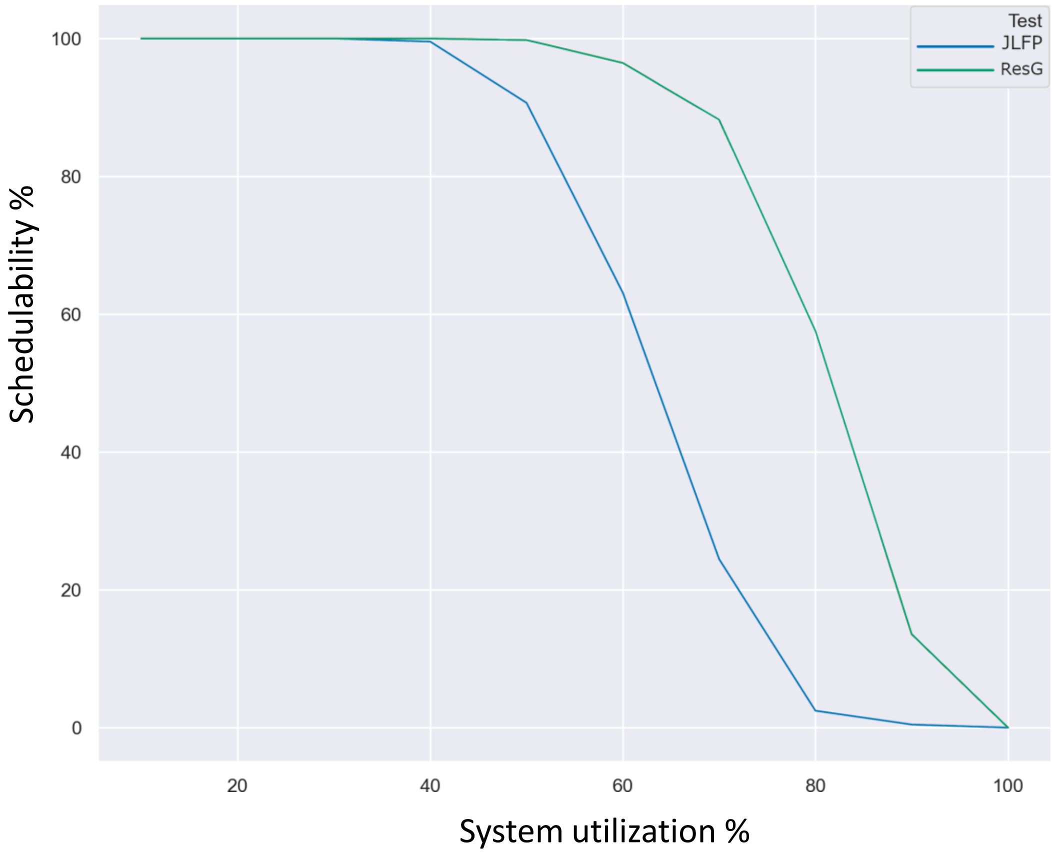
- Evaluated in simulator

Results JLFP vs ResG in simulator

- Evaluated in simulator
- Randomly generated task sets

Results JLFP vs ResG in simulator

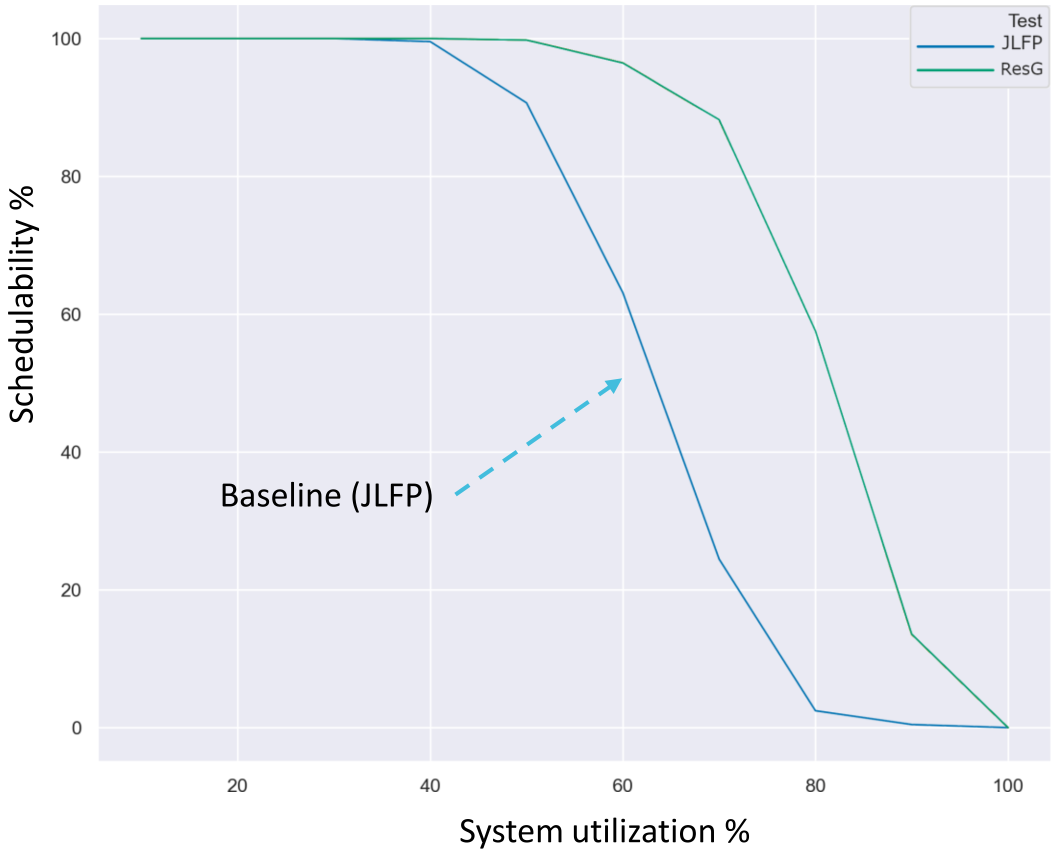
- Evaluated in simulator
- Randomly generated task sets



TUDelft

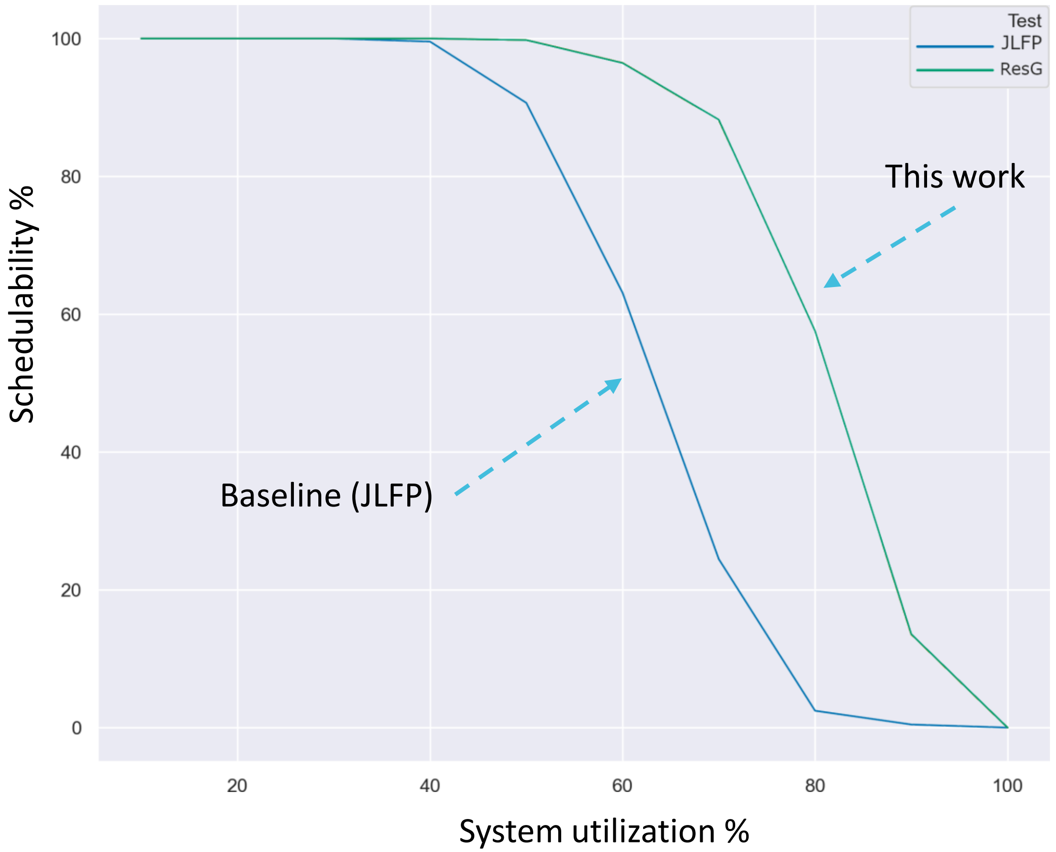
Results JLFP vs ResG in simulator

- Evaluated in simulator
- Randomly generated task sets



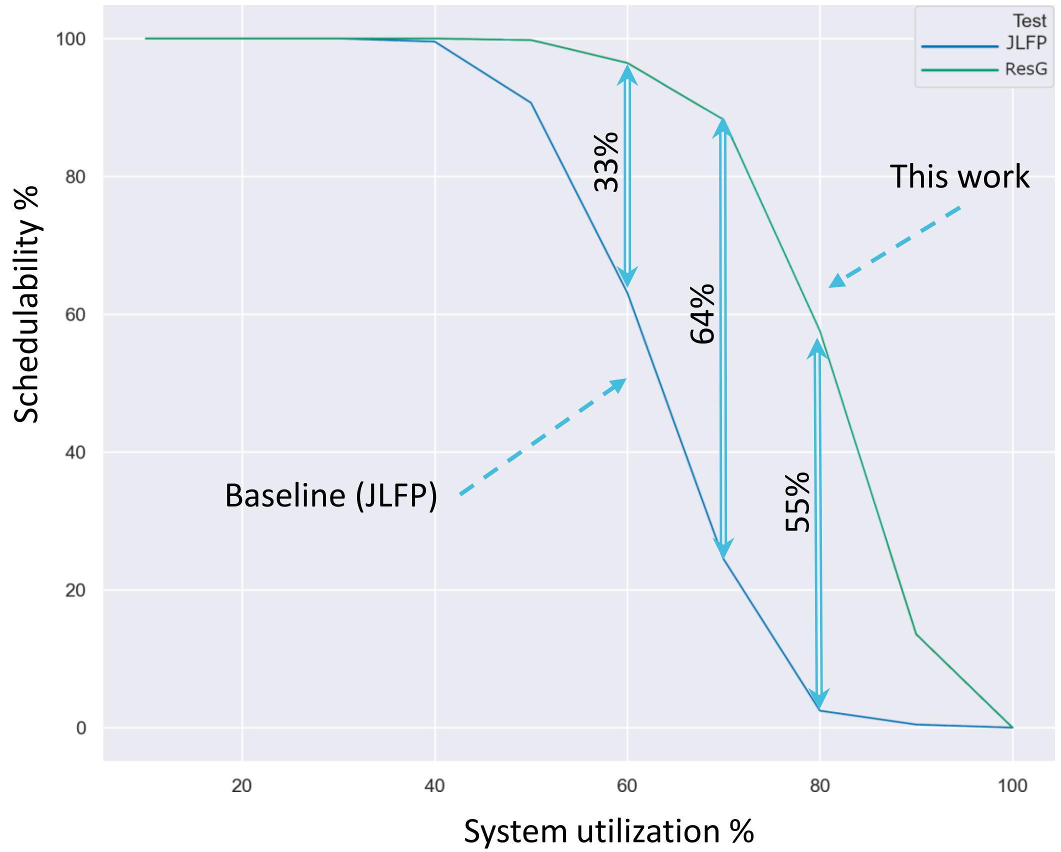
Results JLFP vs ResG in simulator

- Evaluated in simulator
- Randomly generated task sets



Results JLFP vs ResG in simulator

- Evaluated in simulator
- Randomly generated task sets



Conclusions

Conclusions

- With a better scheduling policy one can improve the schedulability of moldable gang tasks

Summary

Summary

- A new analysis for gang tasks using SAG has been defined

Summary

- A new analysis for gang tasks using SAG has been defined
- A new scheduling policy that uses gang moldable properties has been created

Next steps

Next steps

- Further reduce sources of pessimism

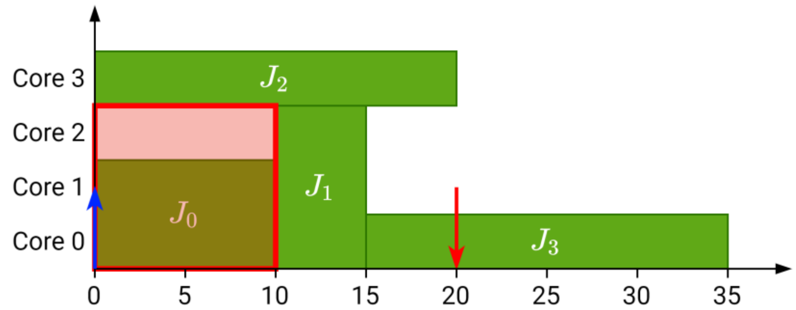
Next steps

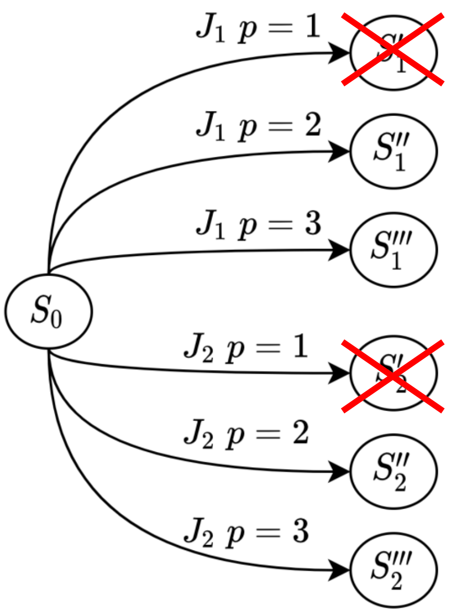
- Further reduce sources of pessimism
- Provide analysis for ResG scheduler and respective proofs

Next steps

- Further reduce sources of pessimism
- Provide analysis for ResG scheduler and respective proofs
- Thorough evaluation of results using SURFSara cluster

Questions?

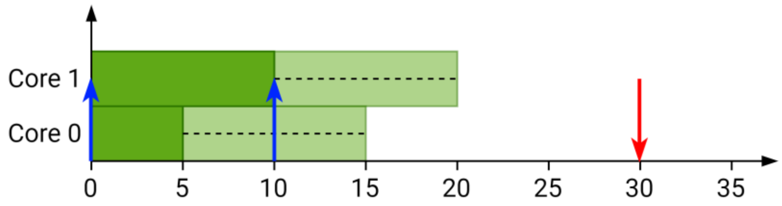

Core 3
Core 2
Core 1
Core 0
J_2
J_1
J_0
J_3
0
5
10
15
20
25
30
35


$J_1\ p = 1$
$J_1\ p = 2$
$J_1\ p = 3$
$J_2\ p = 1$
$J_2\ p = 2$
$J_2\ p = 3$
S_0
S_1''
S_1'''
S_2''
S_2'''

How does SAG work?

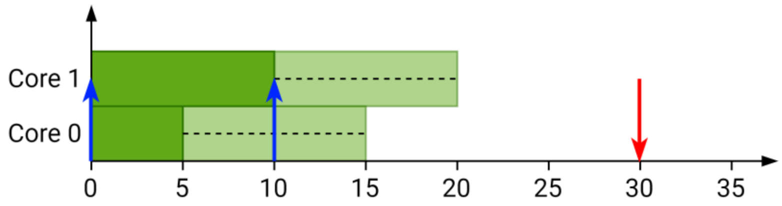
How does SAG work?

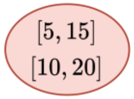
J_i	C_i^{min}	C_i^{max}	r_i	d_i	P_i
J_1	10	15	10	30	1
J_2	5	5	0	100	2



How does SAG work?

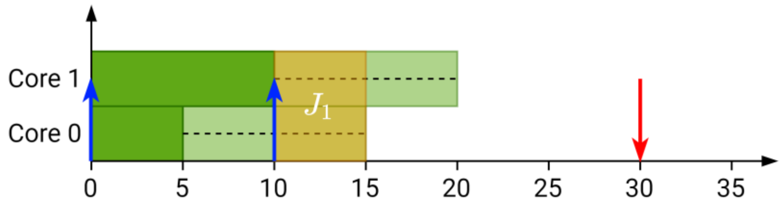
J_i	C_i^{min}	C_i^{max}	r_i	d_i	P_i
J_1	10	15	10	30	1
J_2	5	5	0	100	2

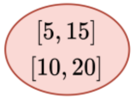




How does SAG work?

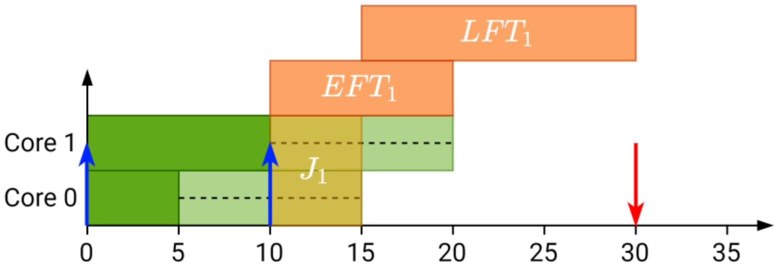
J_i	C_i^{min}	C_i^{max}	r_i	d_i	P_i
J_1	10	15	10	30	1
J_2	5	5	0	100	2

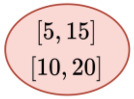




How does SAG work?

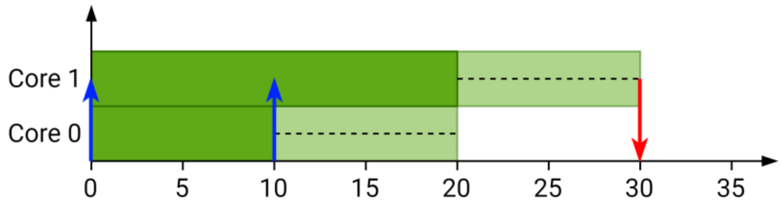
J_i	C_i^{min}	C_i^{max}	r_i	d_i	P_i
J_1	10	15	10	30	1
J_2	5	5	0	100	2

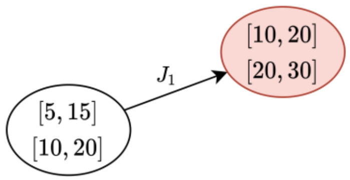




How does SAG work?

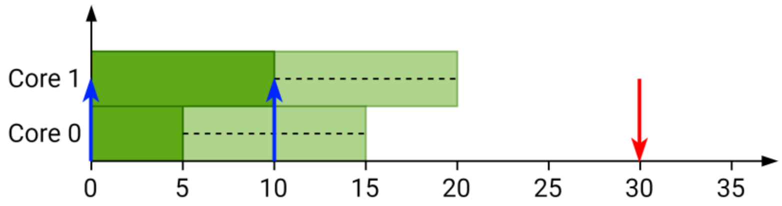
J_i	C_i^{min}	C_i^{max}	r_i	d_i	P_i
J_1	10	15	10	30	1
J_2	5	5	0	00	2

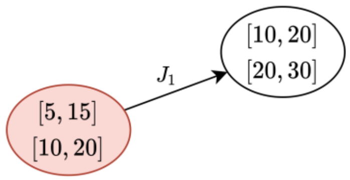




How does SAG work?

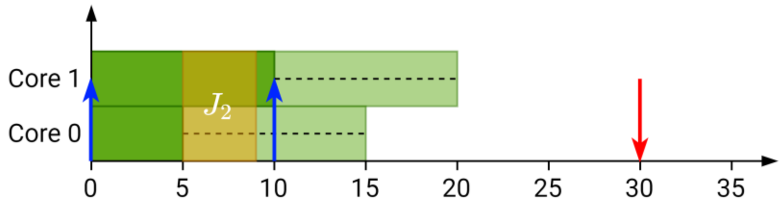
J_i	C_i^{min}	C_i^{max}	r_i	d_i	P_i
J_1	10	15	10	30	1
J_2	5	5	0	00	2

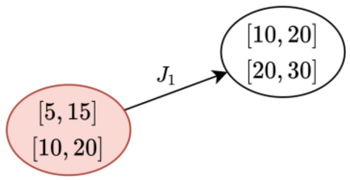




How does SAG work?

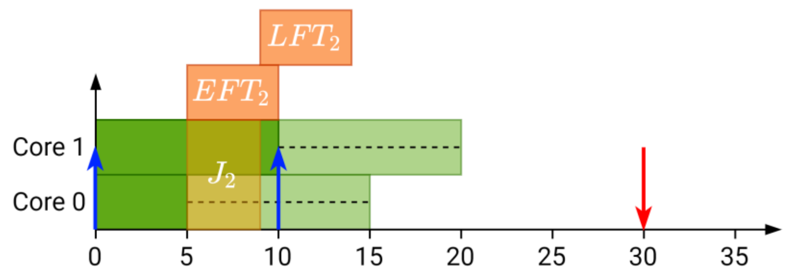
J_i	C_i^{min}	C_i^{max}	r_i	d_i	P_i
J_1	10	15	10	30	1
J_2	5	5	0	00	2

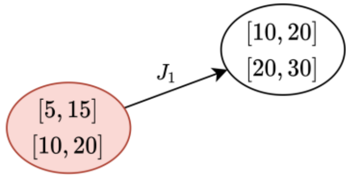




How does SAG work?

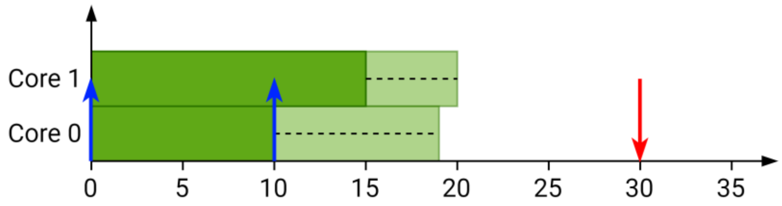
J_i	C_i^{min}	C_i^{max}	r_i	d_i	P_i
J_1	10	15	10	30	1
J_2	5	5	0	00	2

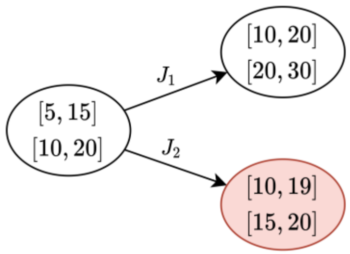




How does SAG work?

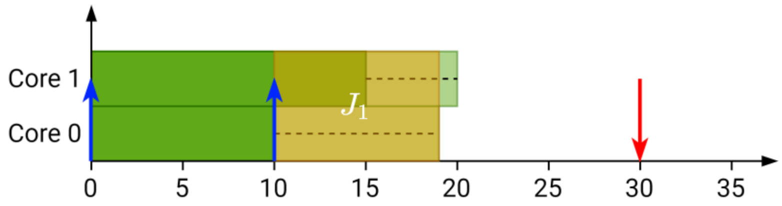
J_i	C_i^{min}	C_i^{max}	r_i	d_i	P_i
J_1	10	15	10	30	1
J_2	5	5	0	00	2

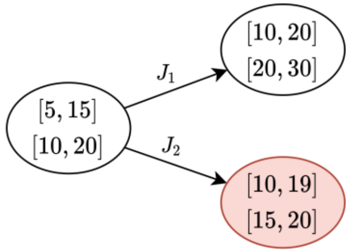




How does SAG work?

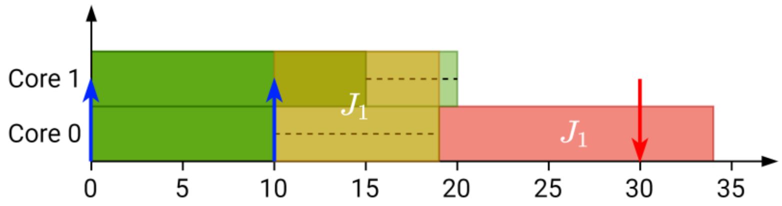
J_i	C_i^{min}	C_i^{max}	r_i	d_i	P_i
J_1	10	15	10	30	1
J_2	5	5	0	00	2

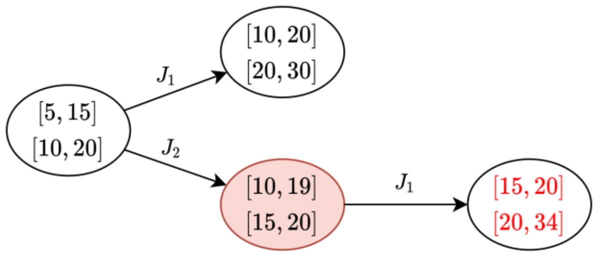




How does SAG work?

J_i	C_i^{min}	C_i^{max}	r_i	d_i	P_i
J_1	10	15	10	30	1
J_2	5	5	0	00	2





SAG analysis changes for gang

SAG analysis changes for gang

$$EST_i = \max\{R_i^{\min}, A_1^{\min}\}$$

$$LST_i = \min\{t_{wc}, t_{high} - 1\}$$

$$t_{wc} = \max\left\{A_1^{\max}, \overset{\infty}{\min}\{R_x^{\max} \mid J_x \in \mathcal{R}^p\}\right\}$$

$$t_{high} = \overset{\infty}{\min}\{th_x(J_i) \mid J_x \in \mathcal{R}^p \wedge p_x < p_i\}$$

$$th_x(J_i) = \max\Big\{r_x^{\max}, \overset{0}{\max}\{LFT_y^* \mid J_y \in pred(J_x) \setminus pred(J_i)\}\Big\}$$

SAG analysis changes for gang

$$EST_i = \max\{R_i^{\min}, A_1^{\min}\}$$

$$LST_i = \min\{t_{wc}, t_{high} - 1\}$$

$$t_{wc} = \max\left\{A_1^{\max}, \overset{\infty}{\min}\{R_x^{\max} \mid J_x \in \mathcal{R}^p\}\right\}$$

$$t_{high} = \overset{\infty}{\min}\{th_x(J_i) \mid J_x \in \mathcal{R}^p \wedge p_x < p_i\}$$

$$th_x(J_i) = \max\left\{r_x^{\max}, \overset{0}{\max}\{LFT_y^* \mid J_y \in pred(J_x) \setminus pred(J_i)\}\right\}$$

$$EST_i^p = \max\{R_i^{\min}, t_{gang}\}$$

$$LST_i^p = \min\{t_{avail}, t_{wc}, t_{high} - 1\}$$

$$t_{wc} = \min_{J_j \in \mathcal{R}^v}\left\{\max\left\{R_j^{\max}, A_{m_j^{\min}}^{\max}\right\}\right\}$$

$$t_{high} = \overset{\infty}{\min_{J_j \in \{hp_i \cap \mathcal{R}^v\}}}\left\{th_x(J_i, J_j), \overset{0}{\max}\{LFT_y^* \mid J_y \in pred(J_j) \setminus pred(J_i)\}\right\}$$

$$t_h(J_i, J_j) = \begin{cases} r_j^{\max} & \text{if } m_j^{\min} \leq p \\ \max\left\{r_j^{\max}, A_{m_j^{\min}}^{\max}\right\} & \text{otherwise} \end{cases}$$

$$t_{gang} = \begin{cases} A_p^{\min} & \text{if } p = m_i^{\max} \\ A_p^{exact} & \text{otherwise} \end{cases} \qquad t_{avail} = \begin{cases} A_{p+1}^{\max} - 1 & \text{if} p < m_i^{\max} \\ +\infty & \text{otherwise} \end{cases}$$

SAG analysis changes for gang

$$EST_i = \max\{R_i^{\min}, A_1^{\min}\}$$

$$LST_i = \min\{t_{wc}, t_{high} - 1\}$$

$$t_{wc} = \max\left\{A_1^{\max}, \overset{\infty}{\min}\{R_x^{\max} \mid J_x \in \mathcal{R}^p\}\right\}$$

$$t_{high} = \overset{\infty}{\min}\{th_x(J_i) \mid J_x \in \mathcal{R}^p \wedge p_x < p_i\}$$

$$th_x(J_i) = \max\left\{r_x^{\max}, \overset{0}{\max}\{LFT_y^* \mid J_y \in pred(J_x) \setminus pred(J_i)\}\right\}$$

Check if execution with p cores is possible

$$EST_i^p = \max\{R_i^{\min}, t_{gang}\}$$

$$LST_i^p = \min\{t_{avail}, t_{wc}, t_{high} - 1\}$$

$$t_{wc} = \min_{J_j \in \mathcal{R}^v}\left\{\max\left\{R_j^{\max}, A_{m_j^{\min}}^{\max}\right\}\right\}$$

$$t_{high} = \overset{\infty}{\min_{J_j \in \{hp_i \cap \mathcal{R}^v\}}}\left\{th_x(J_i, J_j), \overset{0}{\max}\{LFT_y^* \mid J_y \in pred(J_j) \setminus pred(J_i)\}\right\}$$

$$t_h(J_i, J_j) = \begin{cases} r_j^{\max} & \text{if } m_j^{\min} \le p \\ \max\left\{r_j^{\max}, A_{m_j^{\min}}^{\max}\right\} & \text{otherwise} \end{cases}$$

$$t_{gang} = \begin{cases} A_p^{\min} & \text{if } p = m_i^{\max} \\ A_p^{exact} & \text{otherwise} \end{cases} \qquad t_{avail} = \begin{cases} A_{p+1}^{\max} - 1 & \text{if} p < m_i^{\max} \\ +\infty & \text{otherwise} \end{cases}$$

SAG analysis changes for gang

p cores available

$$EST_i = \max\{R_i^{\min}, A_1^{\min}\}$$

$$LST_i = \min\{t_{wc}, t_{high} - 1\}$$

$$t_{wc} = \max\left\{A_1^{\max}, \overset{\infty}{\min}\{R_x^{\max} \mid J_x \in \mathcal{R}^p\}\right\}$$

$$t_{high} = \overset{\infty}{\min}\{th_x(J_i) \mid J_x \in \mathcal{R}^p \wedge p_x < p_i\}$$

$$th_x(J_i) = \max\left\{r_x^{\max}, \overset{0}{\max}\{LFT_y^* \mid J_y \in pred(J_x) \setminus pred(J_i)\}\right\}$$

$$EST_i^p = \max\{R_i^{\min}, t_{gang}\}$$

$$LST_i^p = \min\{t_{avail}, t_{wc}, t_{high} - 1\}$$

$$t_{wc} = \min_{J_j \in \mathcal{R}^v}\left\{\max\left\{R_j^{\max}, A_{m_j^{\min}}^{\max}\right\}\right\}$$

$$t_{high} = \overset{\infty}{\min_{J_j \in \{hp_i \cap \mathcal{R}^v\}}}\left\{th_x(J_i, J_j), \overset{0}{\max}\{LFT_y^* \mid J_y \in pred(J_j) \setminus pred(J_i)\}\right\}$$

$$t_h(J_i, J_j) = \begin{cases} r_j^{\max} & \text{if } m_j^{\min} \le p \\ \max\left\{r_j^{\max}, A_{m_j^{\min}}^{\max}\right\} & \text{otherwise} \end{cases}$$

$$t_{gang} = \begin{cases} A_p^{\min} & \text{if } p = m_i^{\max} \\ A_p^{exact} & \text{otherwise} \end{cases}$$

$$t_{avail} = \begin{cases} A_{p+1}^{\max} - 1 & \text{if} p < m_i^{\max} \\ +\infty & \text{otherwise} \end{cases}$$

TUDelft

SAG analysis changes for gang

$$EST_i = \max\{R_i^{\min}, A_1^{\min}\}$$

$$LST_i = \min\{t_{wc}, t_{high} - 1\}$$

$$t_{wc} = \max\left\{A_1^{\max}, \overset{\infty}{\min}\{R_x^{\max} \mid J_x \in \mathcal{R}^p\}\right\}$$

$$t_{high} = \overset{\infty}{\min}\{th_x(J_i) \mid J_x \in \mathcal{R}^p \wedge p_x < p_i\}$$

$$th_x(J_i) = \max\left\{r_x^{\max}, \overset{0}{\max}\{LFT_y^* \mid J_y \in pred(J_x) \setminus pred(J_i)\}\right\}$$

$$EST_i^p = \max\{R_i^{\min}, t_{gang}\}$$

$$LST_i^p = \min\{t_{avail}, t_{wc}, t_{high} - 1\}$$

$p + 1$ cores **not** available

$$t_{wc} = \min_{J_j \in \mathcal{R}^v}\left\{\max\left\{R_j^{\max}, A_{m_j^{\min}}^{\max}\right\}\right\}$$

$$t_{high} = \overset{\infty}{\min_{J_j \in \{hp_i \cap \mathcal{R}^v\}}}\left\{th_x(J_i, J_j), \overset{0}{\max}\{LFT_y^* \mid J_y \in pred(J_j) \setminus pred(J_i)\}\right\}$$

$$th(J_i, J_j) = \begin{cases} r_j^{\max} & \text{if } m_j^{\min} \le p \\ \max\left\{r_j^{\max}, A_{m_j^{\min}}^{\max}\right\} & \text{otherwise} \end{cases}$$

$$t_{gang} = \begin{cases} A_p^{\min} & \text{if } p = m_i^{\max} \\ A_p^{exact} & \text{otherwise} \end{cases}$$

$$t_{avail} = \begin{cases} A_{p+1}^{\max} - 1 & \text{if} p < m_i^{\max} \\ +\infty & \text{otherwise} \end{cases}$$