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

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

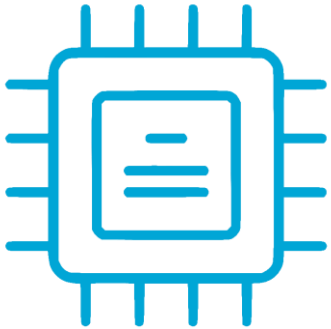
- Systems of which correctness does depends not only on **logical** results but also on **timing constraints**

Real-time systems

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

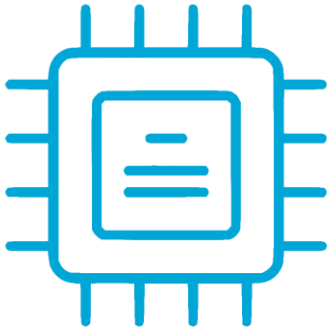
Real-time systems

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



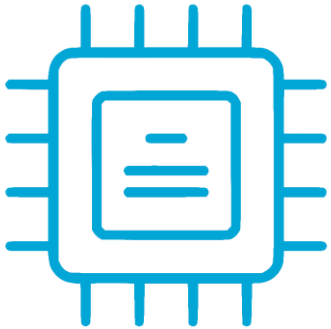
Real-time systems

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



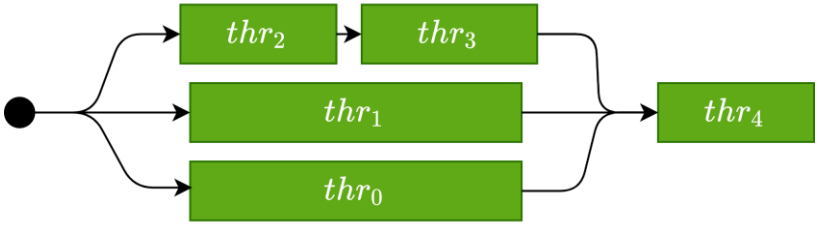
Real-time systems

- 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

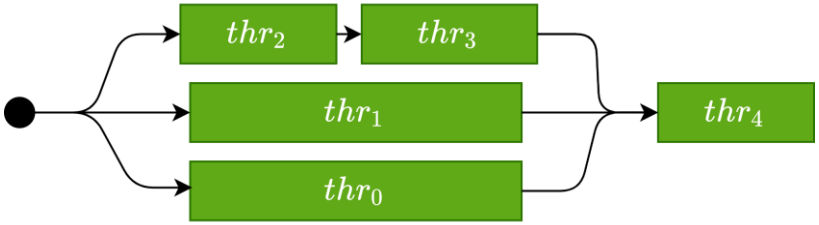


What is gang?

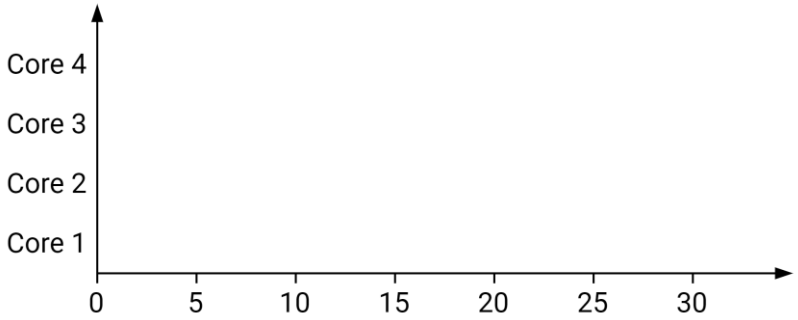
What is gang?



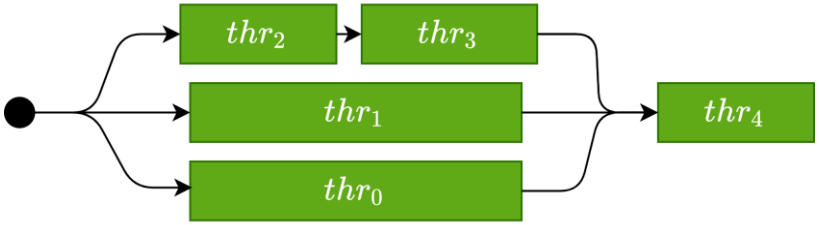
What is gang?

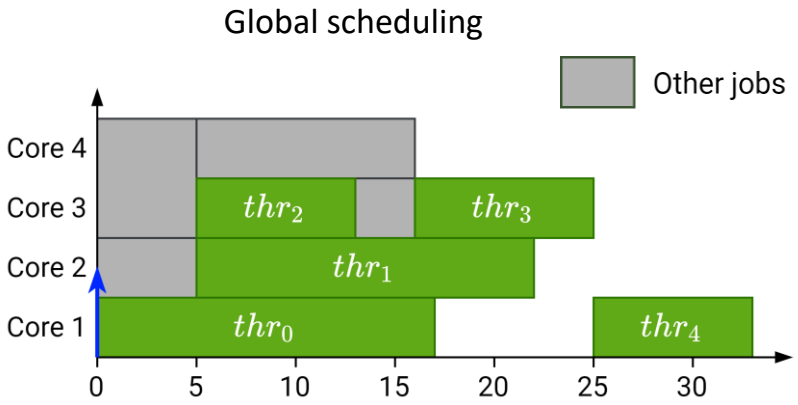


Global scheduling

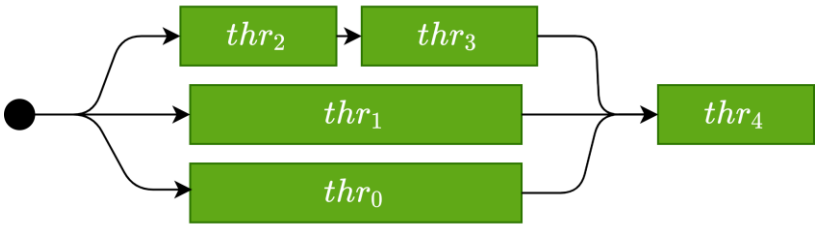


What is gang?


thr_2
thr_3
thr_1
thr_4
thr_0

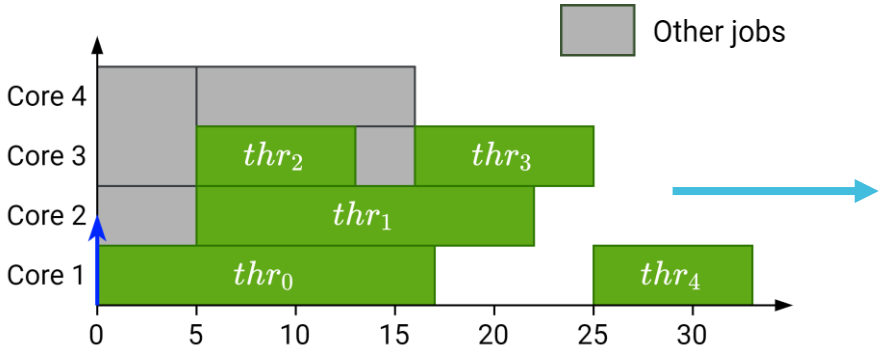

Global scheduling
Other jobs
Core 4
Core 3
Core 2
Core 1
thr_2
thr_3
thr_1
thr_0
thr_4
0
5
10
15
20
25
30

What is gang?

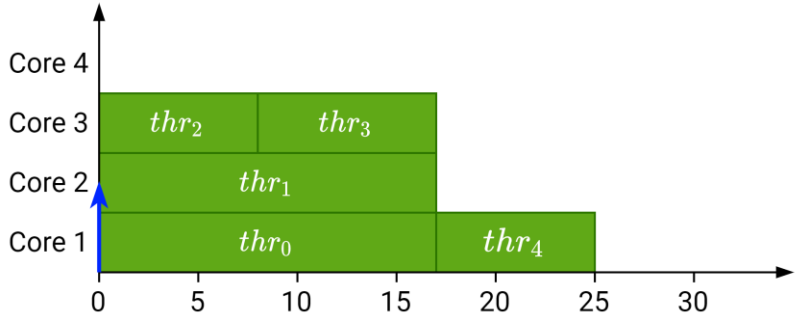


Parallel threads together as a “gang”

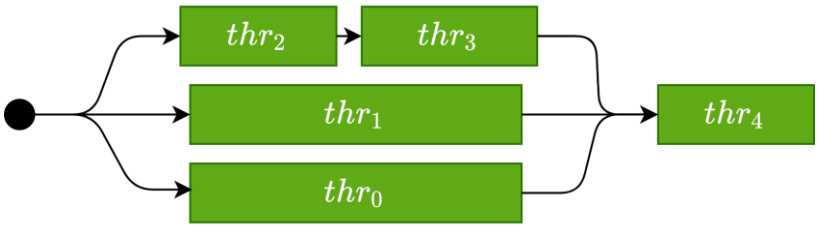
Global scheduling



Gang Scheduling



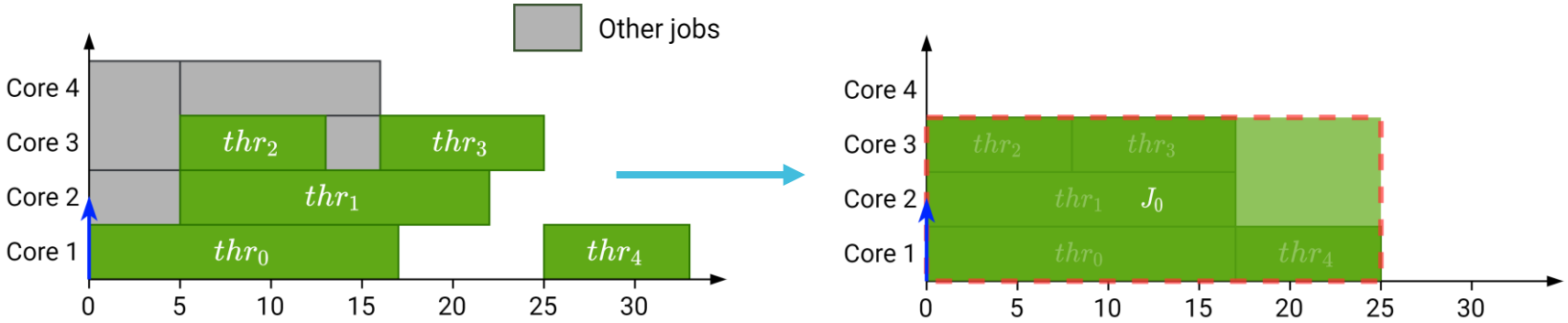
What is gang?



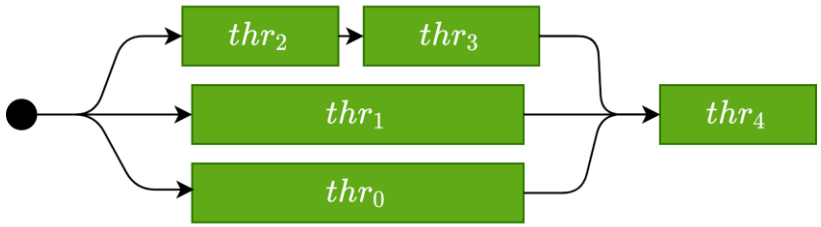
Parallel threads together as a “gang”

Global scheduling

Gang Scheduling



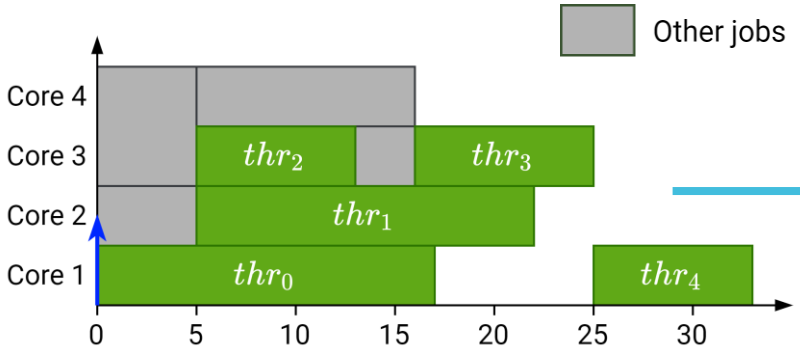
What is gang?



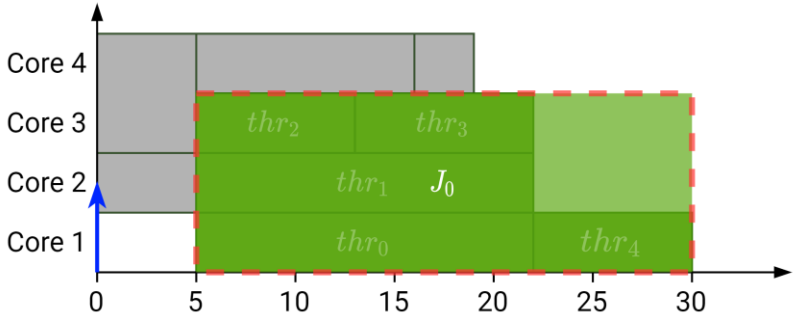
Parallel threads together as a “gang”

Execution does not start until there are enough cores

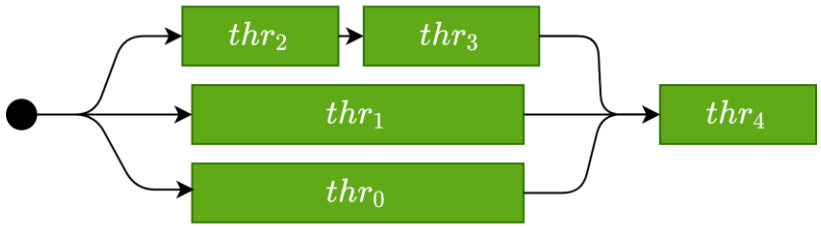
Global scheduling



Gang Scheduling



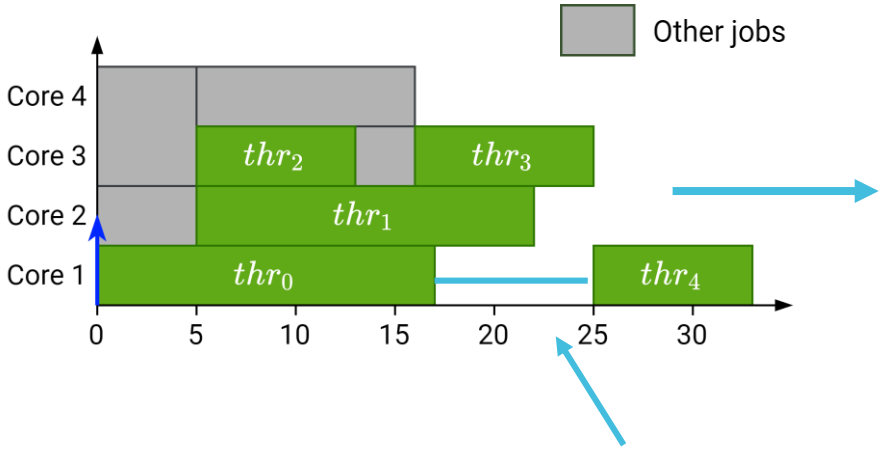
What is gang?



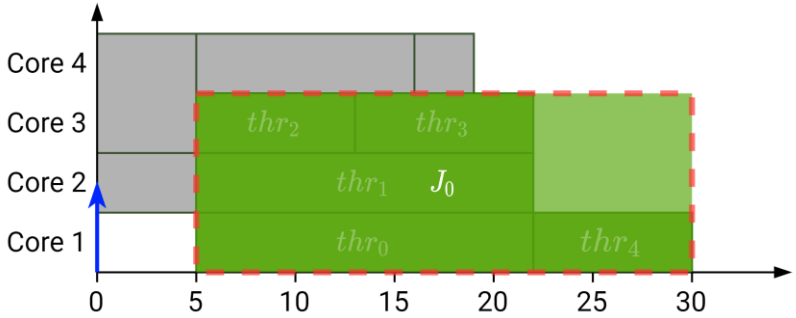
Parallel threads together as a “gang”

Execution does not start until there are enough cores

Global scheduling



Gang Scheduling



Why gang?

Why gang?

- More efficient synchronization

Why gang?

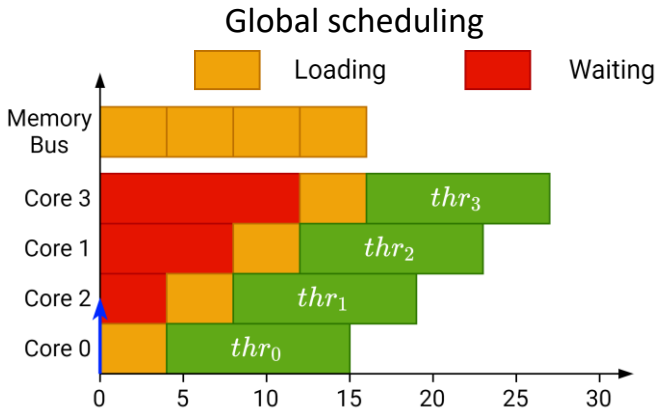
- More efficient synchronization
- Reduces variability in the execution

Why gang?

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

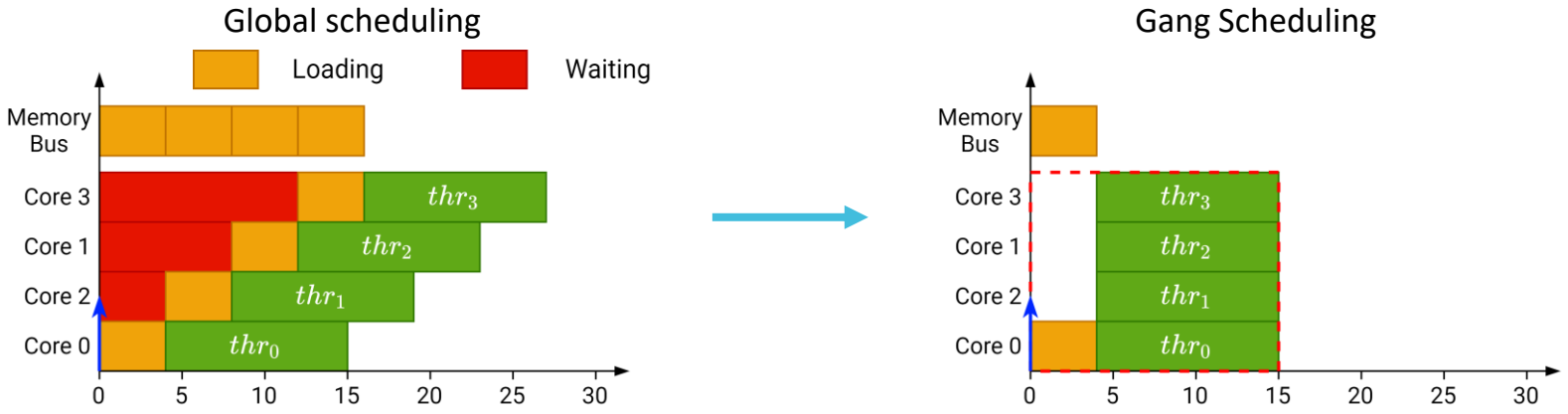
Why gang?

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



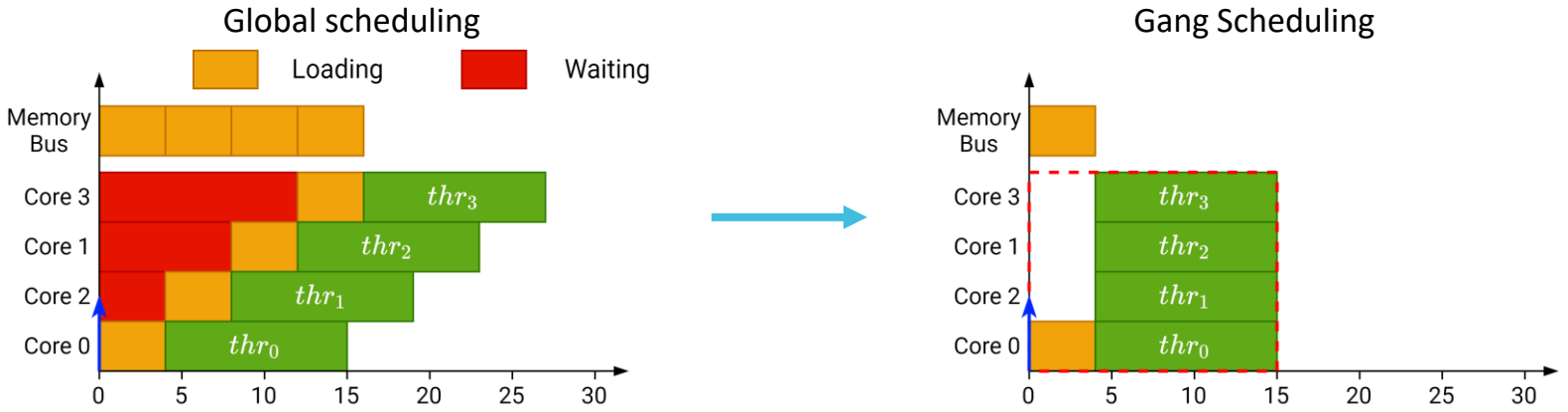
Why gang?

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



Why gang?

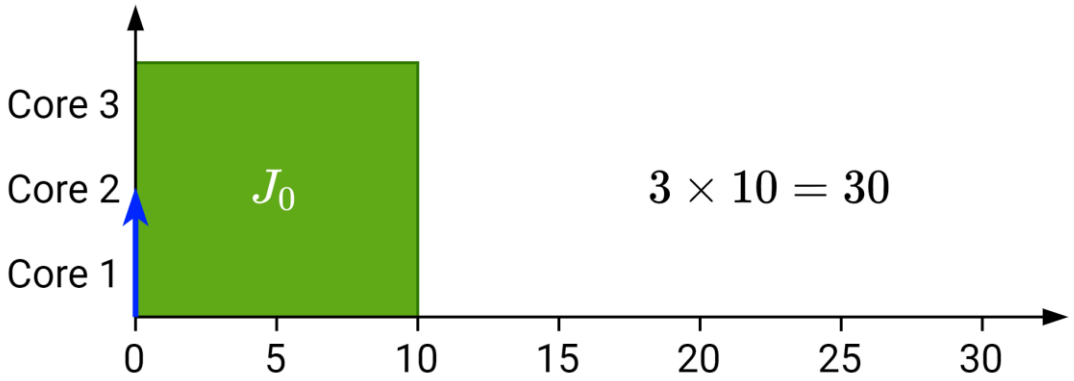
- More efficient synchronization
- Reduces variability in the execution
- Avoids overhead when loading initial data
- Shows its full potential when executed non-preemptively



Types of gang

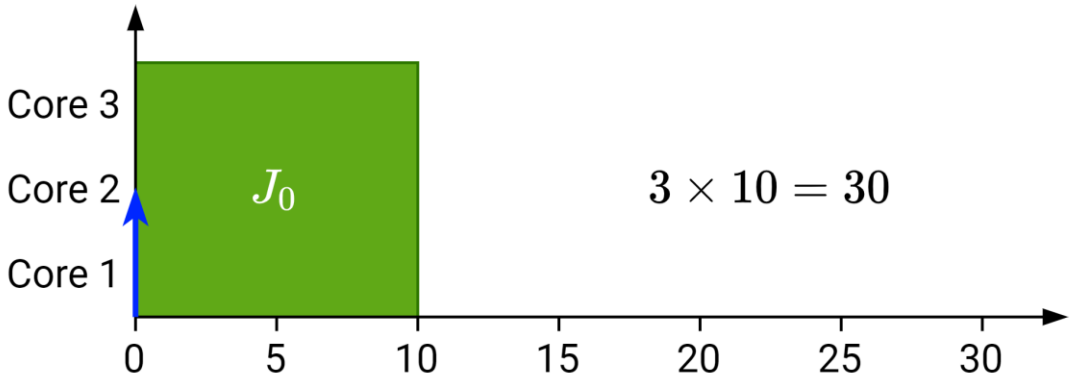
Types of gang

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



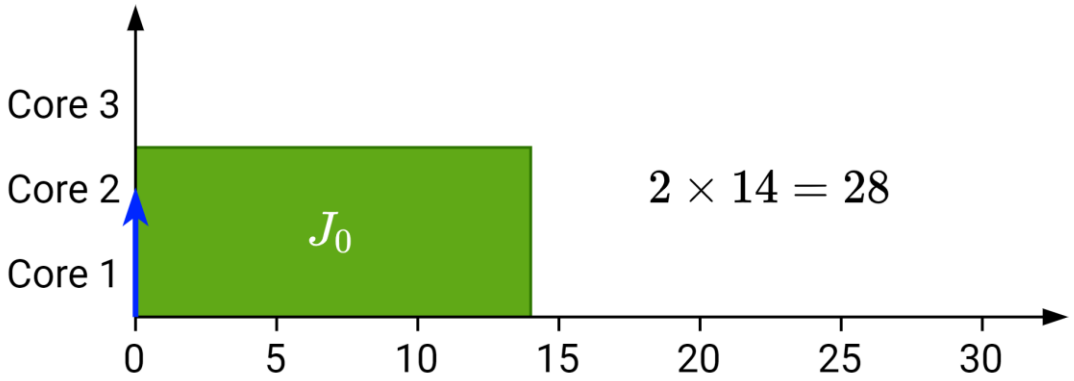
Types of gang

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



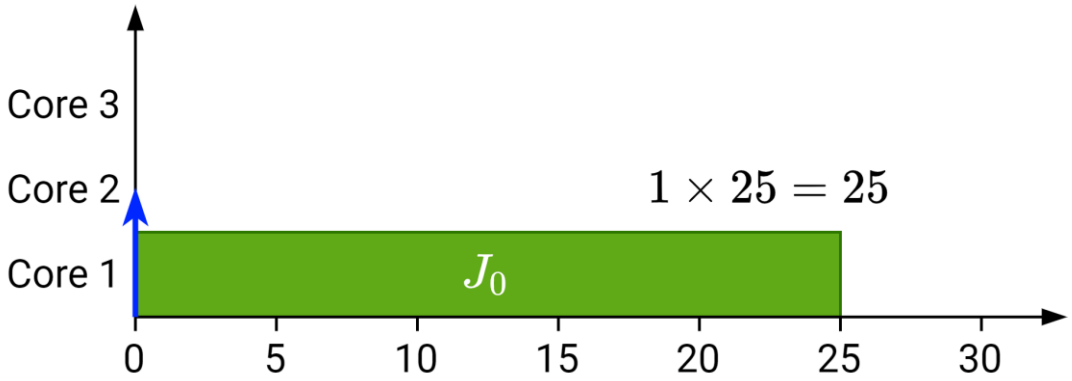
Types of gang

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



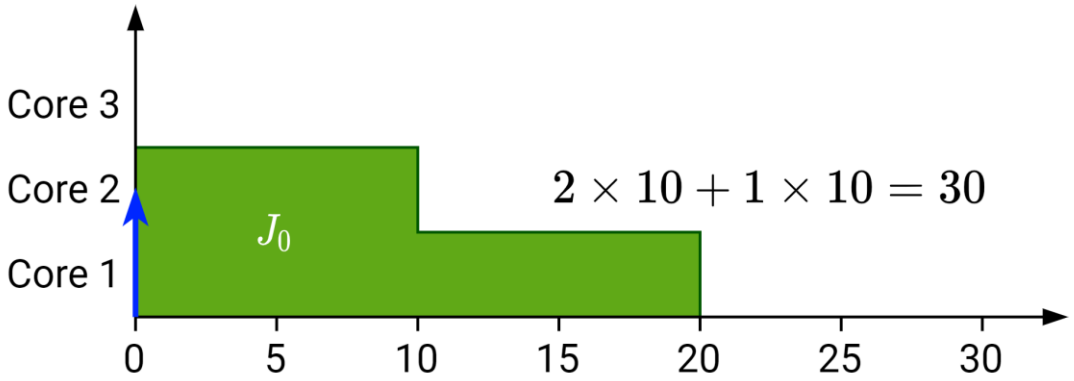
Types of gang

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



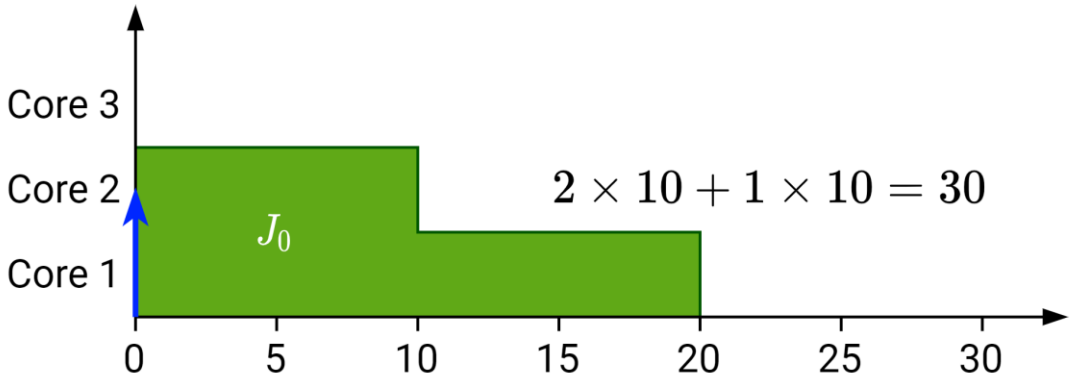
Types of gang

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



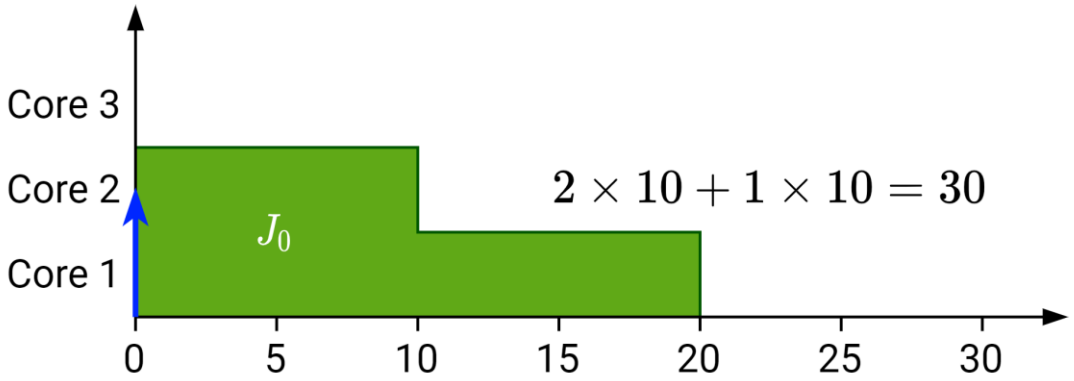
Types of gang

- **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



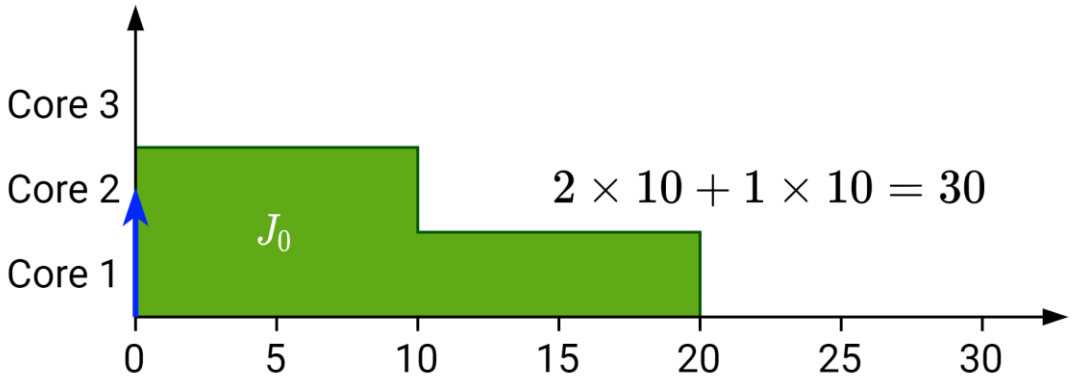
Types of gang

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



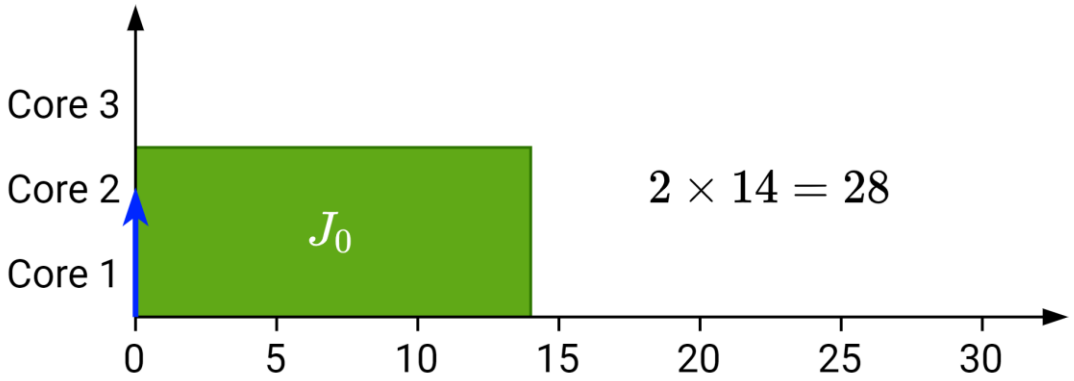
Types of gang

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

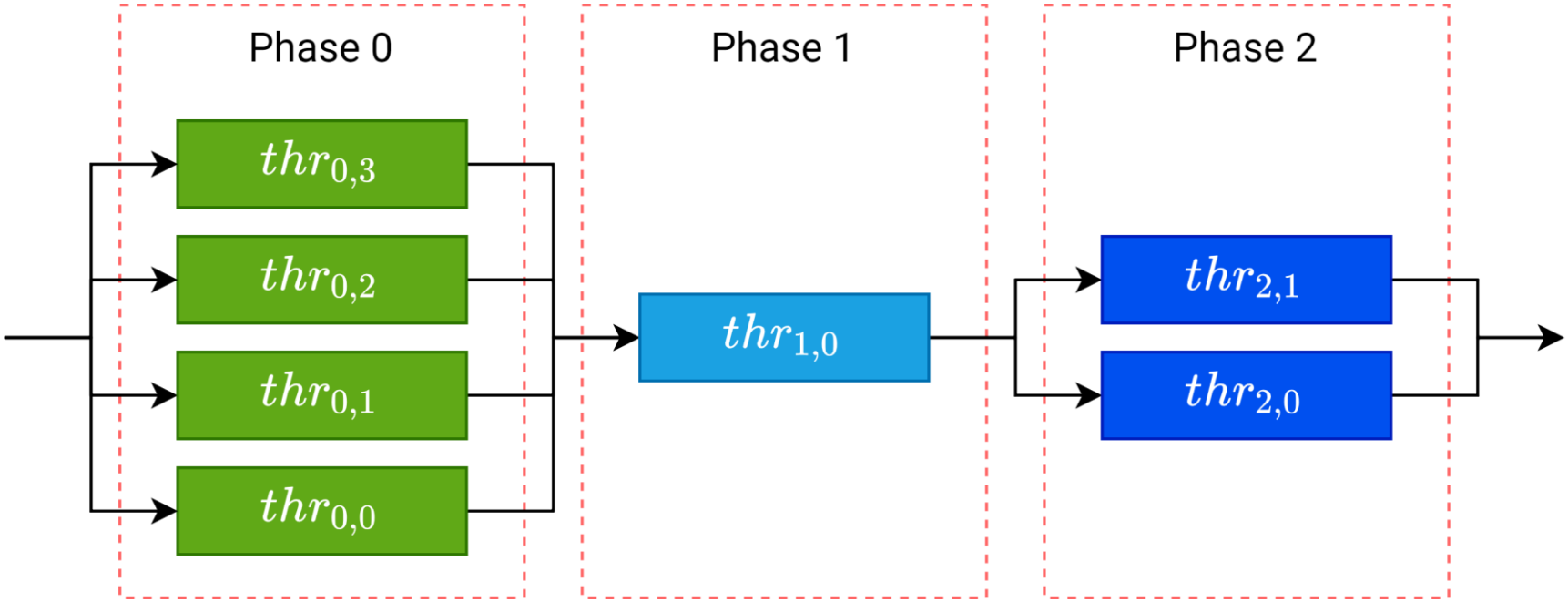


Types of gang

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

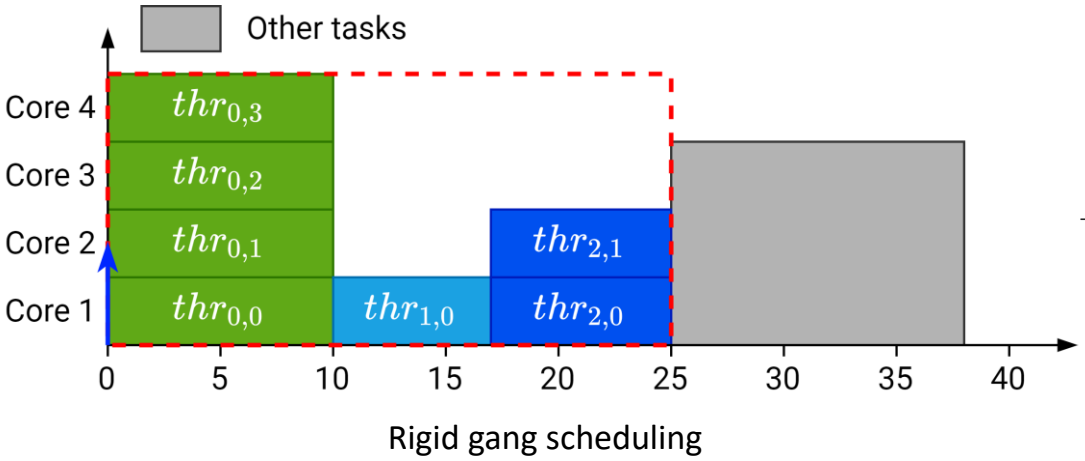


Bundled scheduling[1] vs limited-preemptive

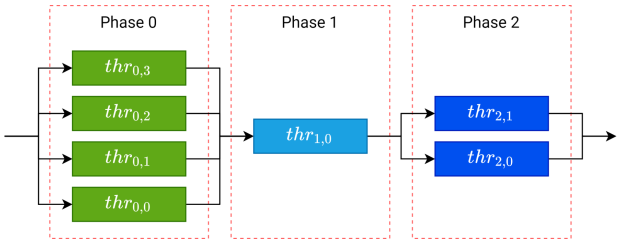


Bundled scheduling[1] vs limited-preemptive

- Rigid gang reserves the whole block

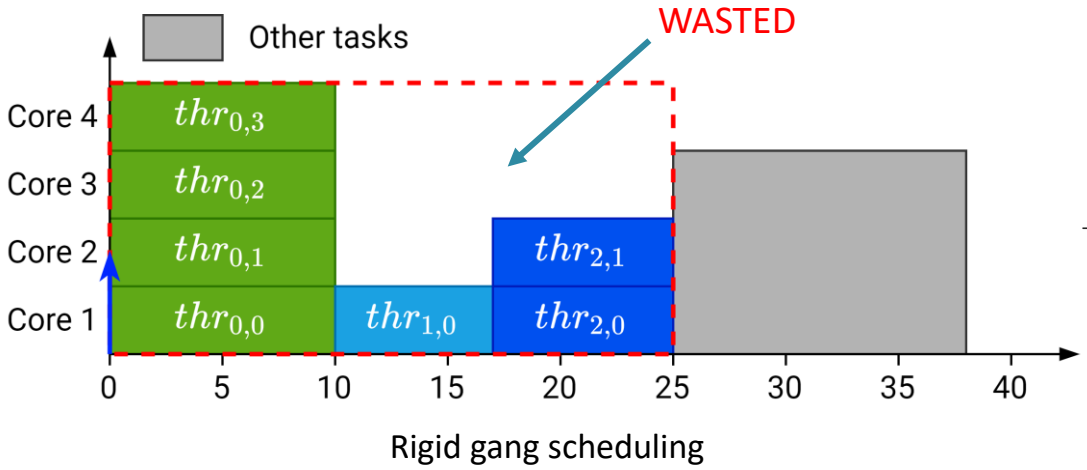


Rigid gang scheduling

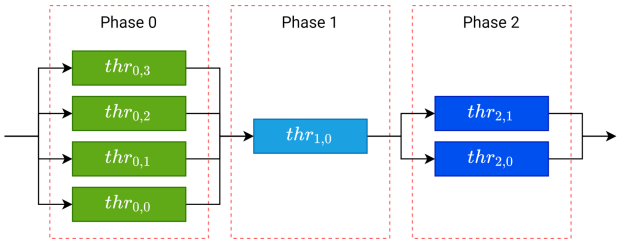


Bundled scheduling[1] vs limited-preemptive

- Rigid gang reserves the whole block

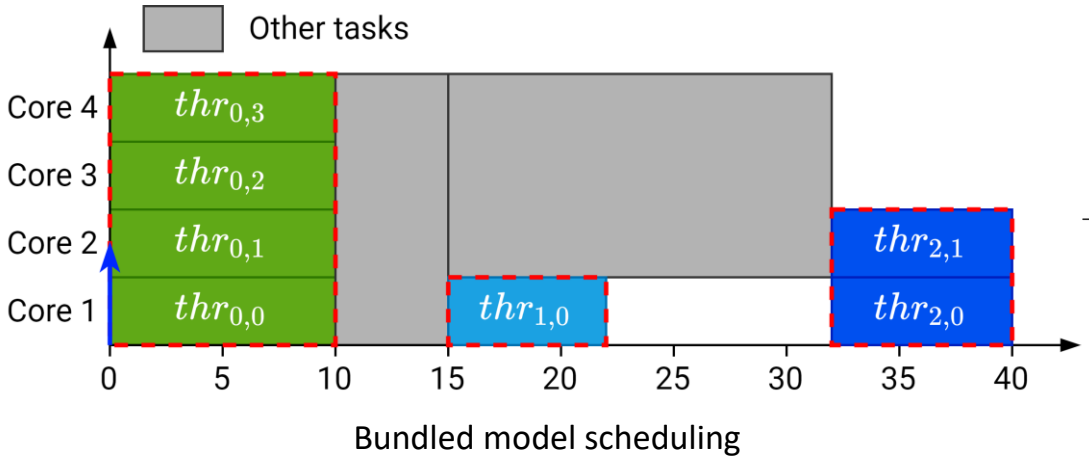


Rigid gang scheduling

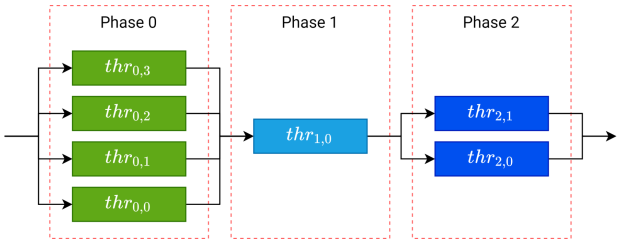


Bundled scheduling[1] vs limited-preemptive

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

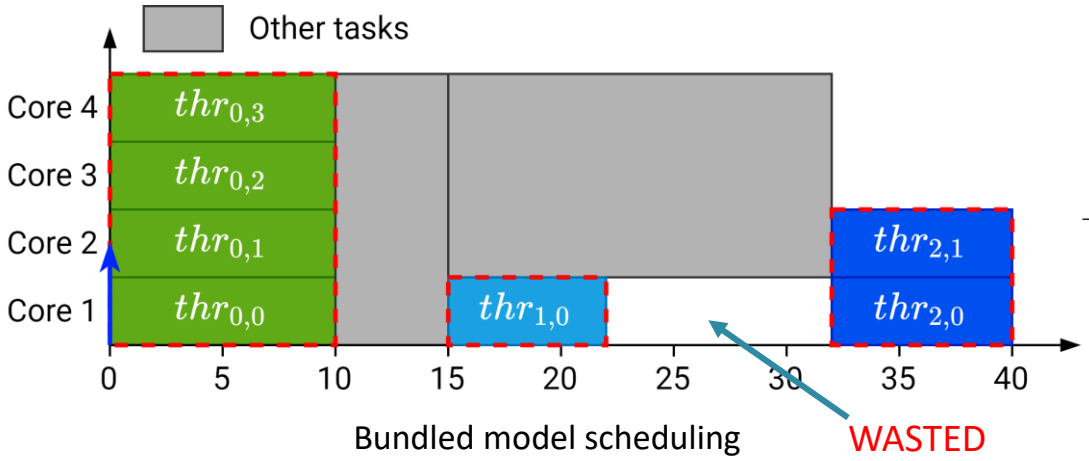


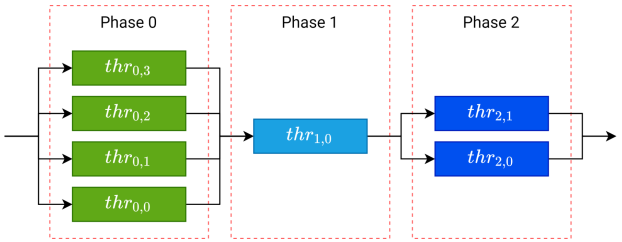
Bundled model scheduling



Bundled scheduling[1] vs limited-preemptive

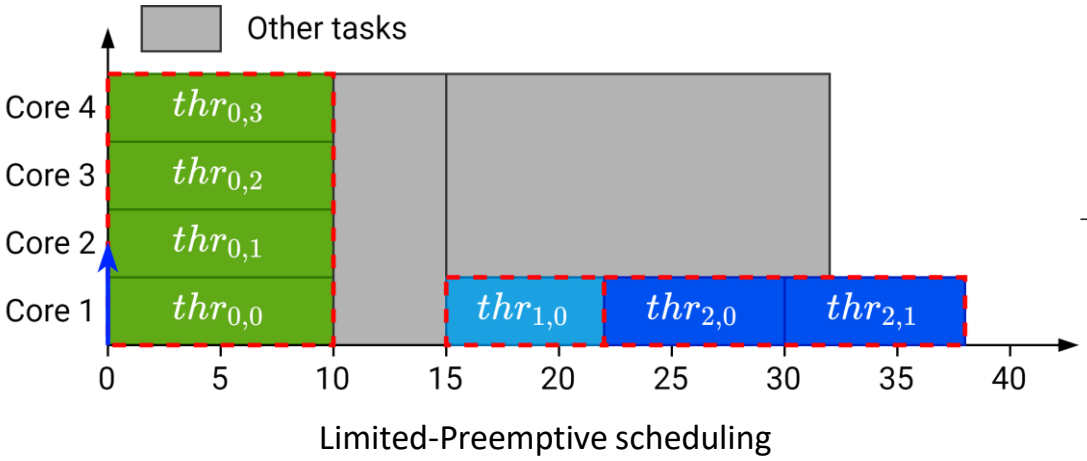
- Rigid gang reserves the whole block
- Bundled creates rigid blocks with dependencies



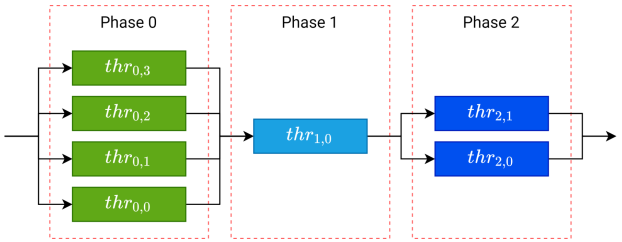


Bundled scheduling[1] vs limited-preemptive

- 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

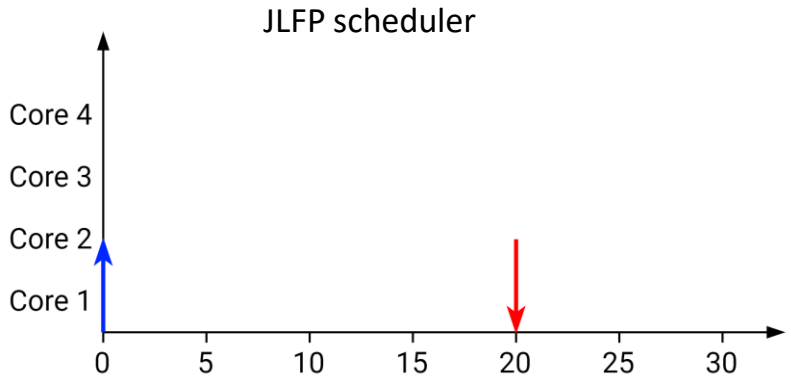
- Based on global JLFP scheduler

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

- Based on global JLFP scheduler
- Assigns maximum possible cores to a job

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

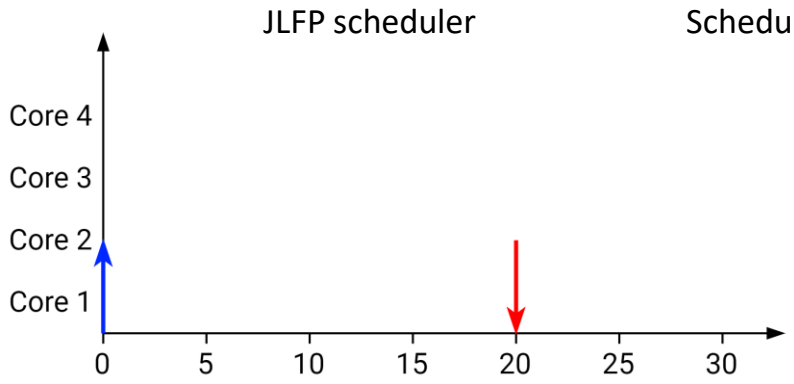
- Based on global JLFP scheduler
- Assigns maximum possible cores to a job



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

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

- Based on global JLFP scheduler
- Assigns maximum possible cores to a job

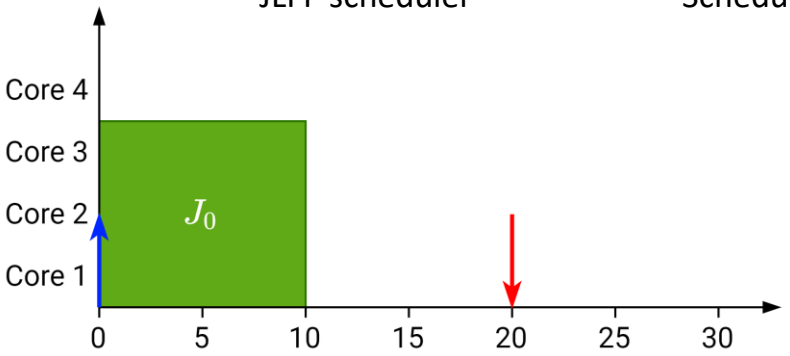


	Priority	Min cores	Max cores	Deadline	Execution time
J_0	High	1	3	20	30, 15, 10
J_1	Low	2	2	20	15

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

- Based on global JLFP scheduler
- Assigns maximum possible cores to a job

JLFP scheduler



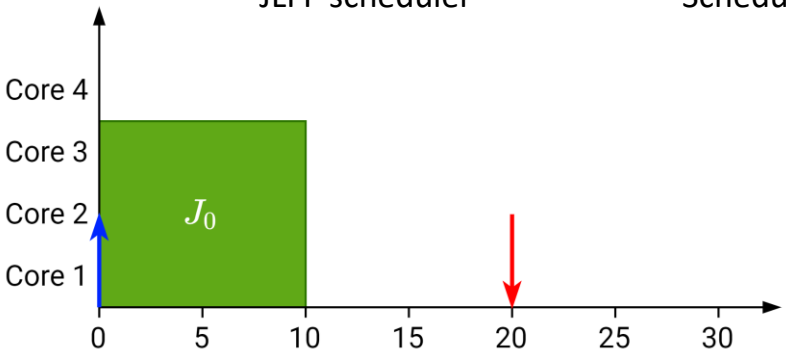
Scheduling: J_0

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

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

- Based on global JLFP scheduler
- Assigns maximum possible cores to a job

JLFP scheduler



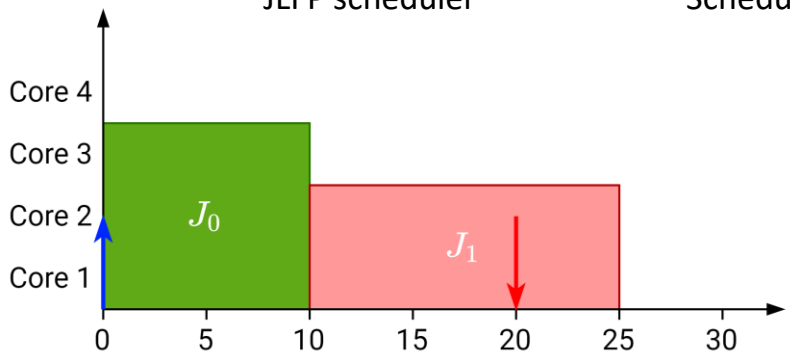
Scheduling: J_1

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

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

- Based on global JLFP scheduler
- Assigns maximum possible cores to a job

JLFP scheduler

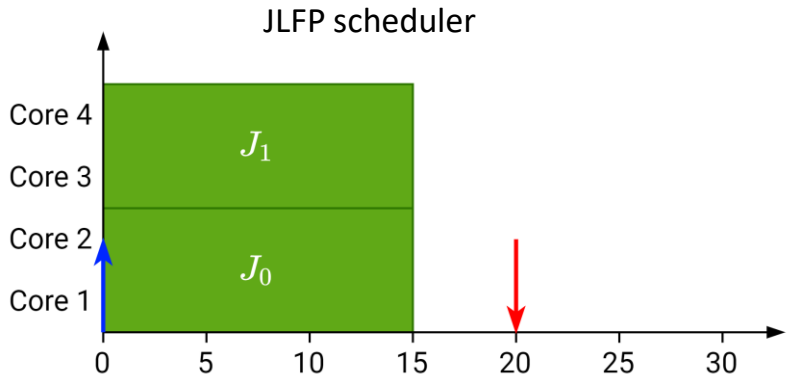


Scheduling: J_1

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

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

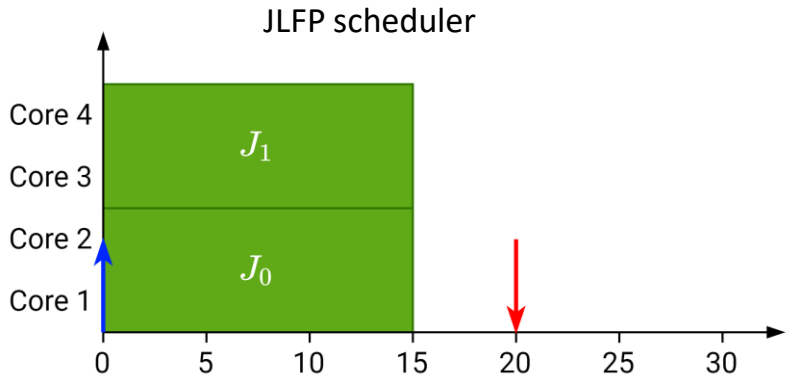
- Based on global JLFP scheduler
- Assigns maximum possible cores to a job



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

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



	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

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]

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

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 a job

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

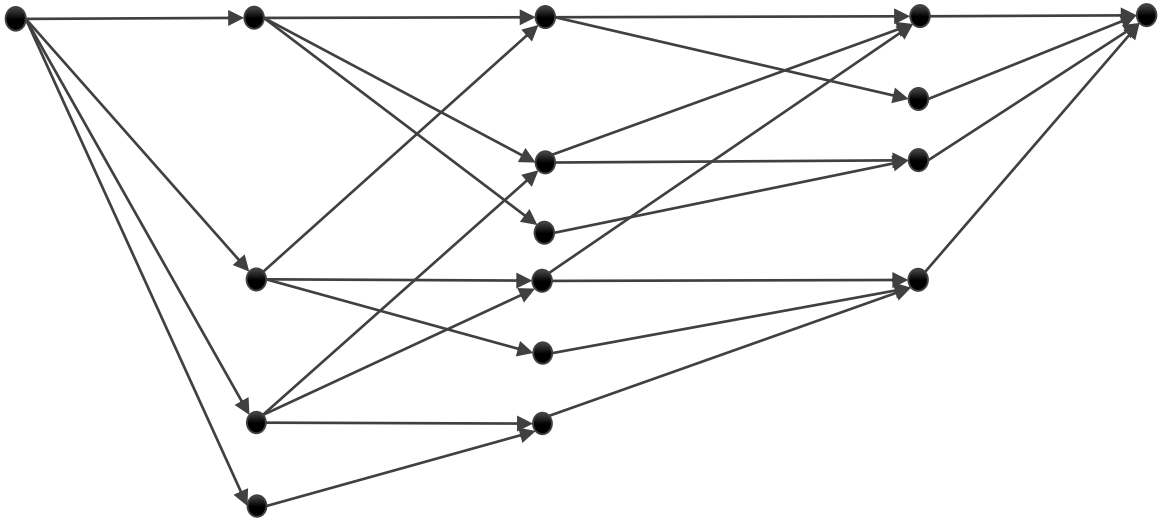
3. Propose a new scheduling algorithm to improve the schedulability of limited-preemptive moldable gang tasks

Agenda

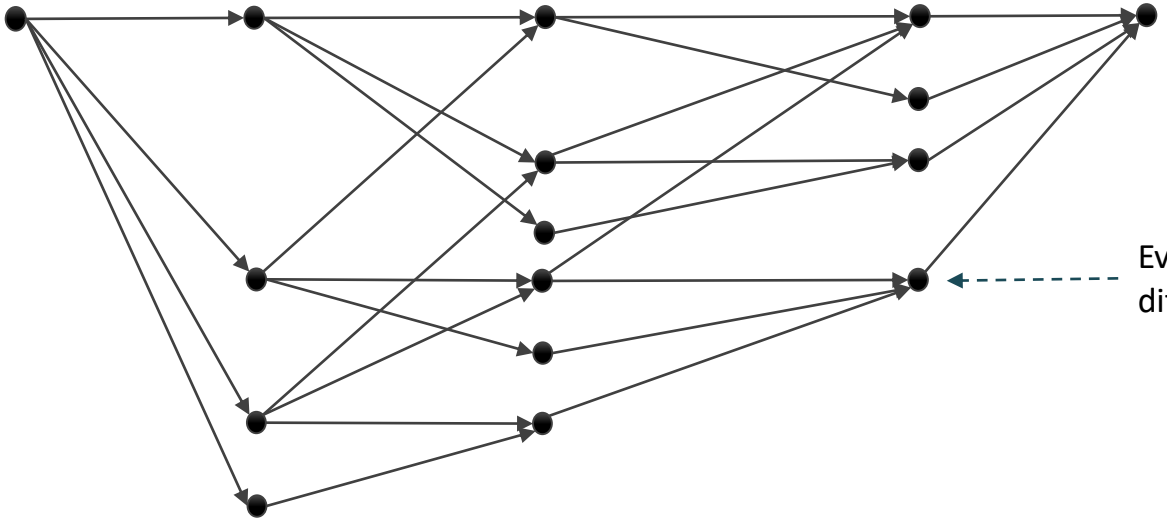
- Gang schedulability analysis
- New scheduling policy

Schedule abstraction graph

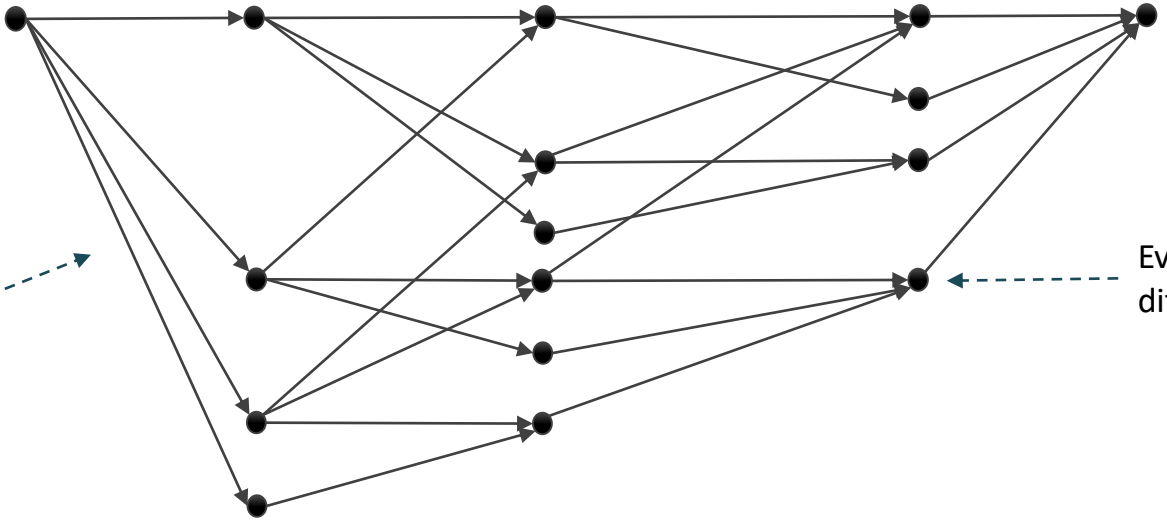
Schedule abstraction graph



Schedule abstraction graph



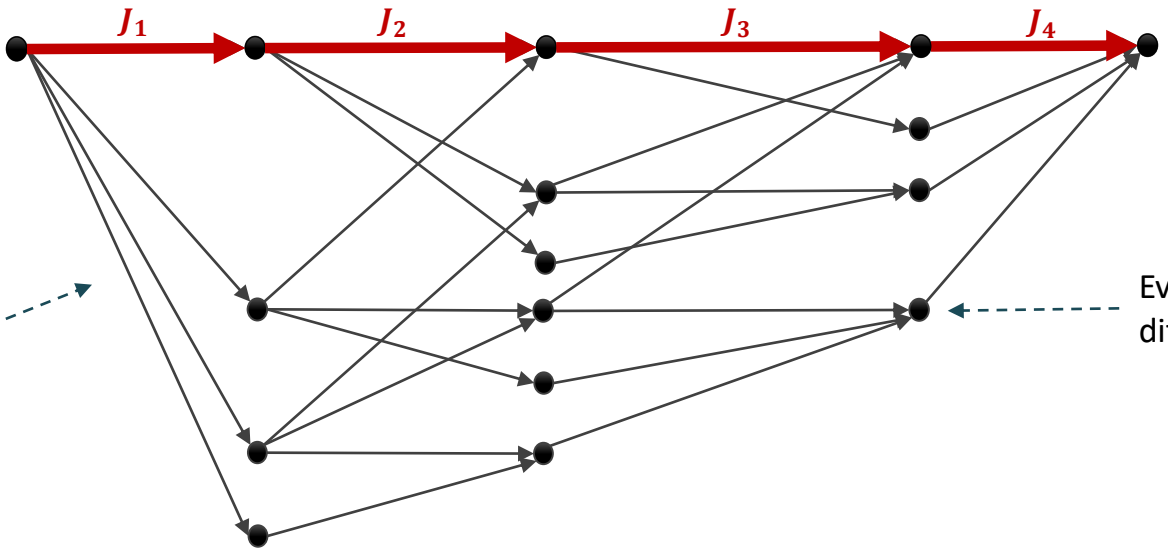
Schedule abstraction graph



Every edge is a different scheduler decision

Every node is a different system state

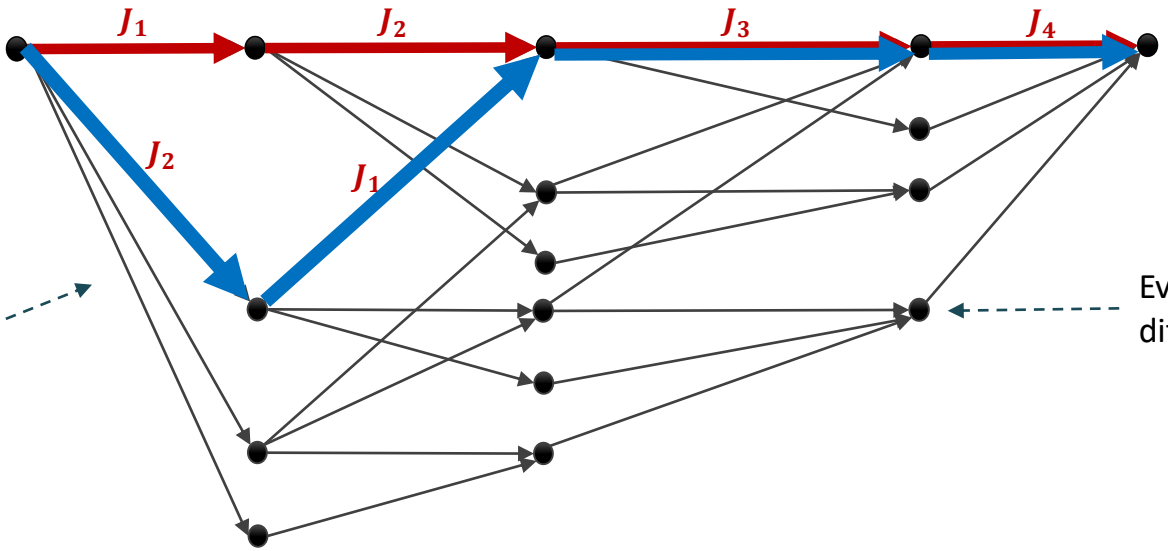
Schedule abstraction graph



Every edge is a different scheduler decision

Every node is a different system state

Schedule abstraction graph

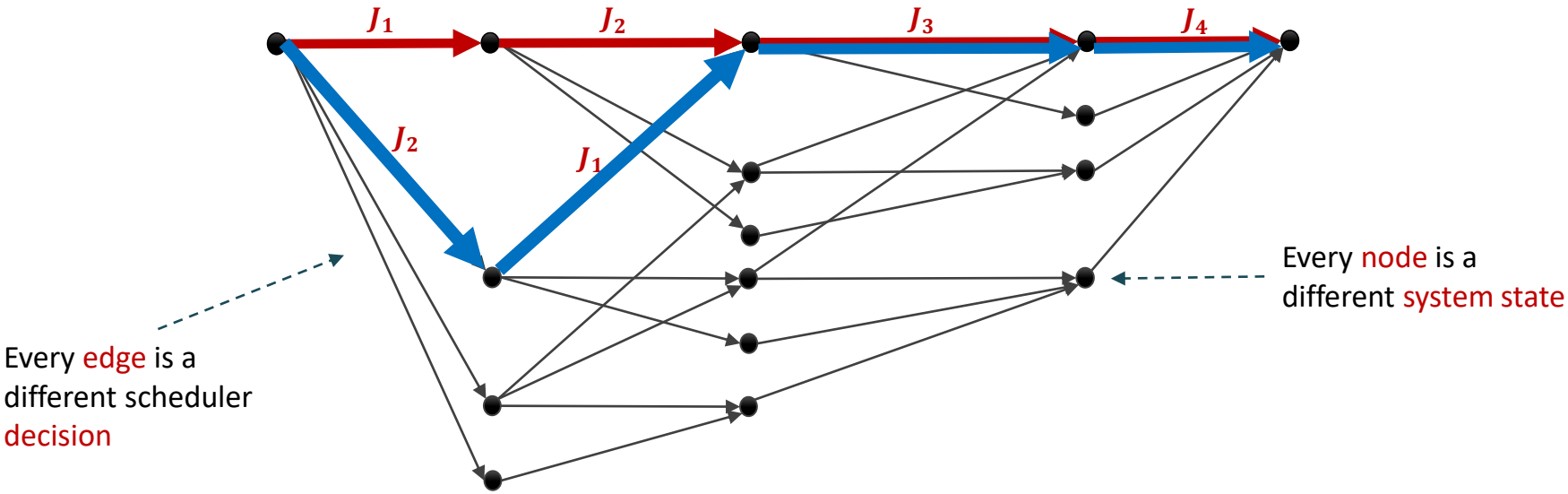


Every edge is a different scheduler decision

Every node is a different system state

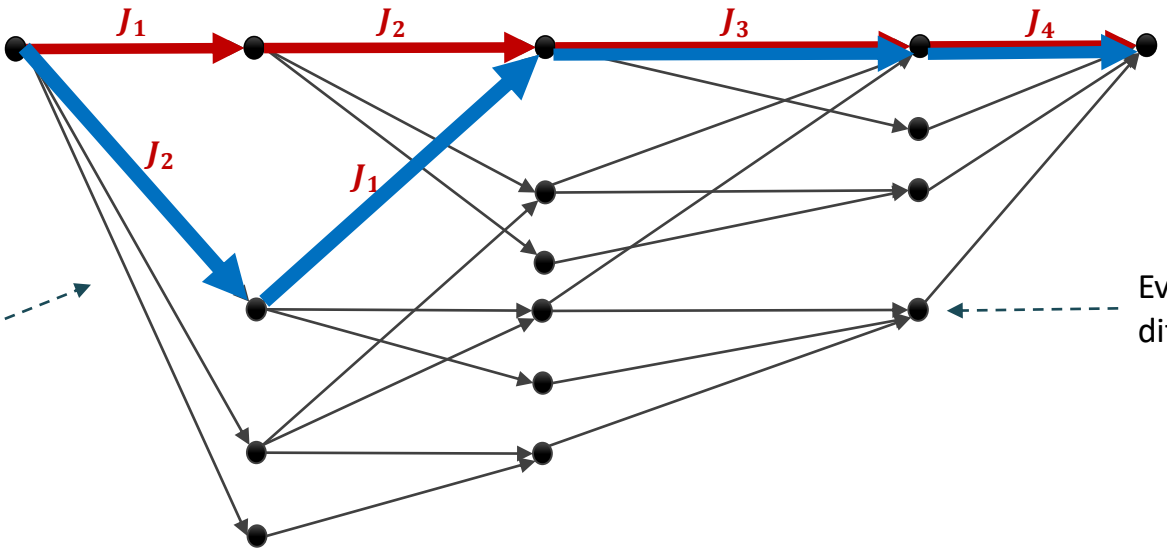
Schedule abstraction graph

- It is a technique that allows:



Schedule abstraction graph

- It is a technique that allows:
  - Search for all possible schedules

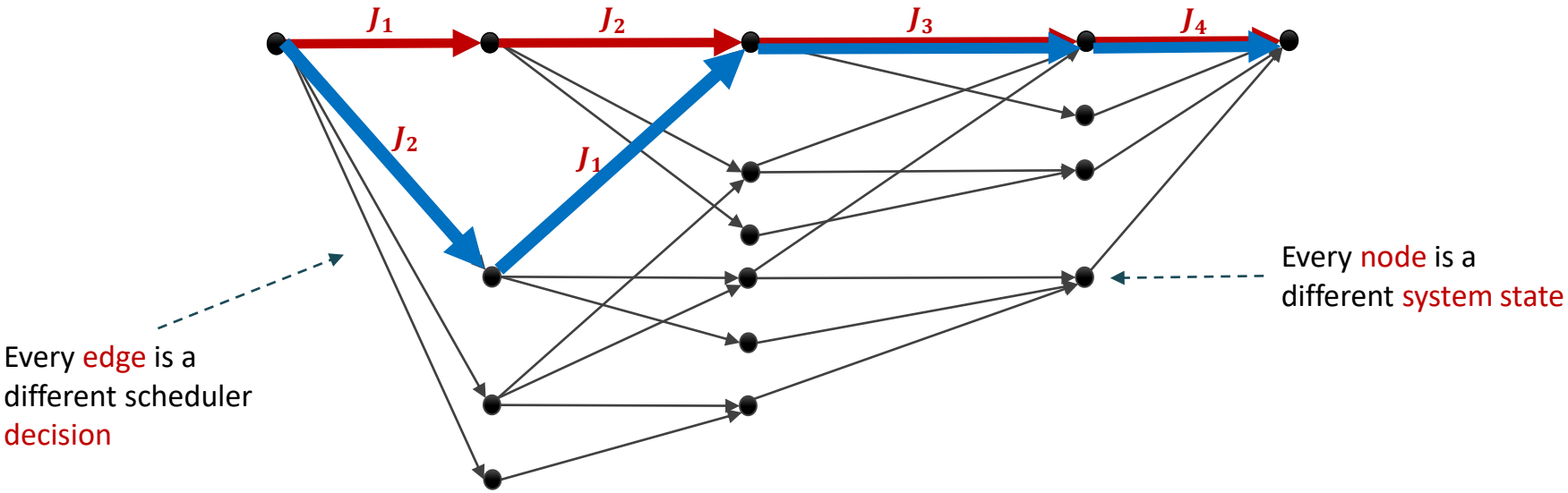


Every edge is a different scheduler decision

Every node is a different system state

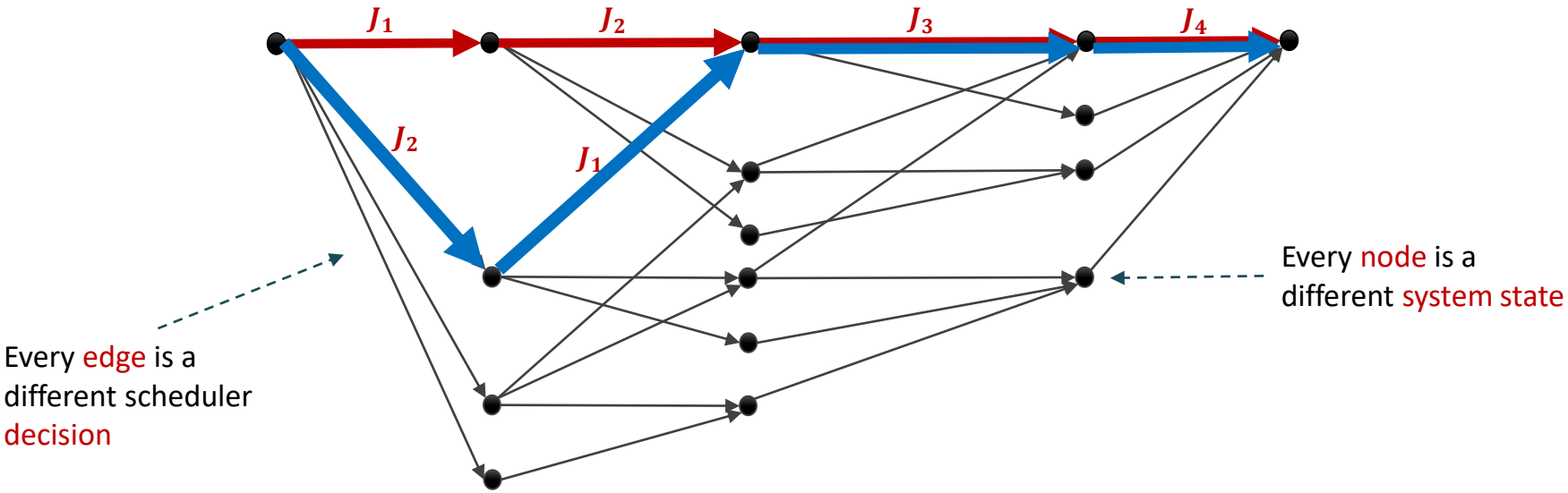
Schedule abstraction graph

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



Schedule abstraction graph

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



Using SAG for gang scheduling

Using SAG for gang scheduling

- Introduce new system state representation

Using SAG for gang scheduling

- Introduce new system state representation
- Introduce new expansion rules

Using SAG for gang scheduling

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

New system state representation

New system state representation

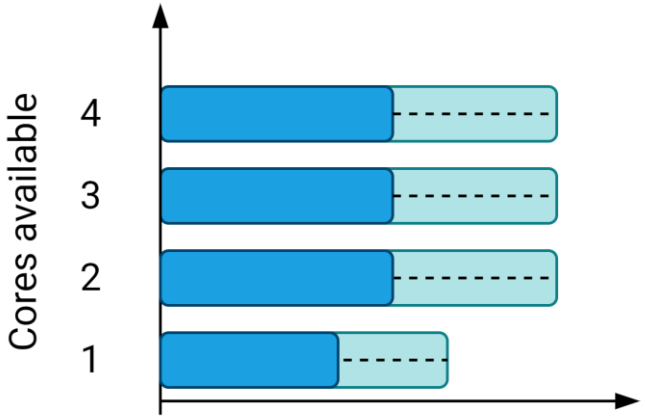
- Availabilities of cores

New system state representation

- Availabilities of cores → Know how many cores we have at time *t*

New system state representation

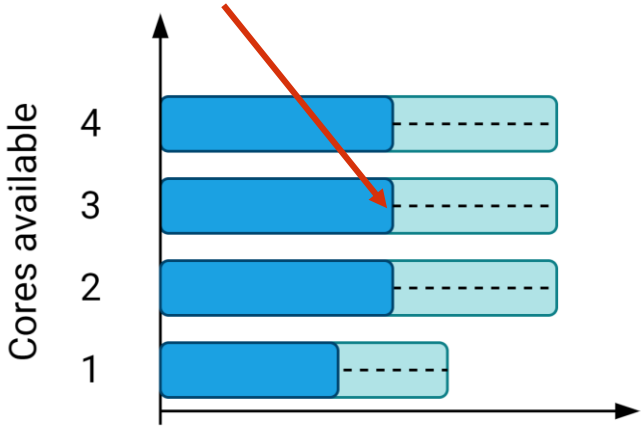
- Availabilities of cores ⟶ Know how many cores we have at time *t*



New system state representation

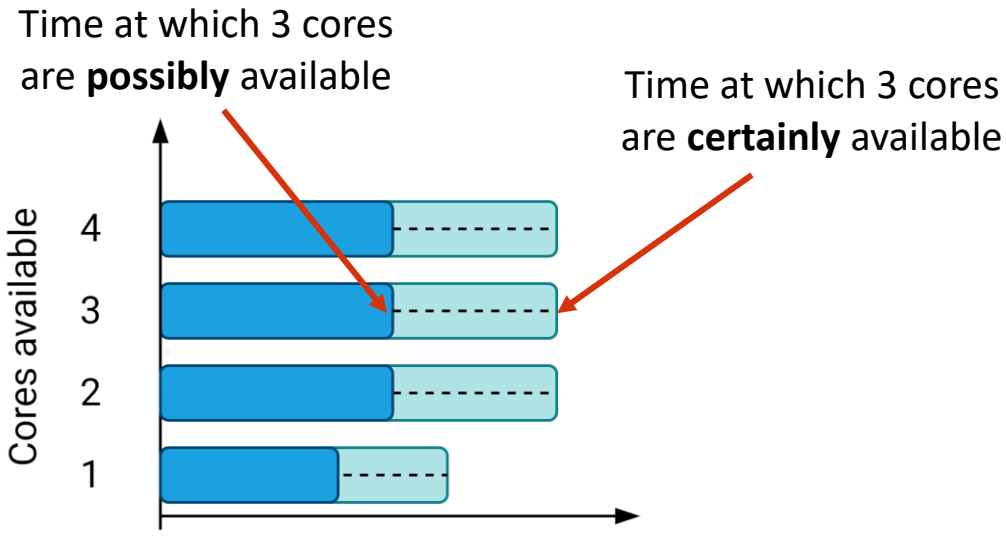
- Availabilities of cores ⟶ Know how many cores we have at time *t*

Time at which 3 cores are **possibly** available



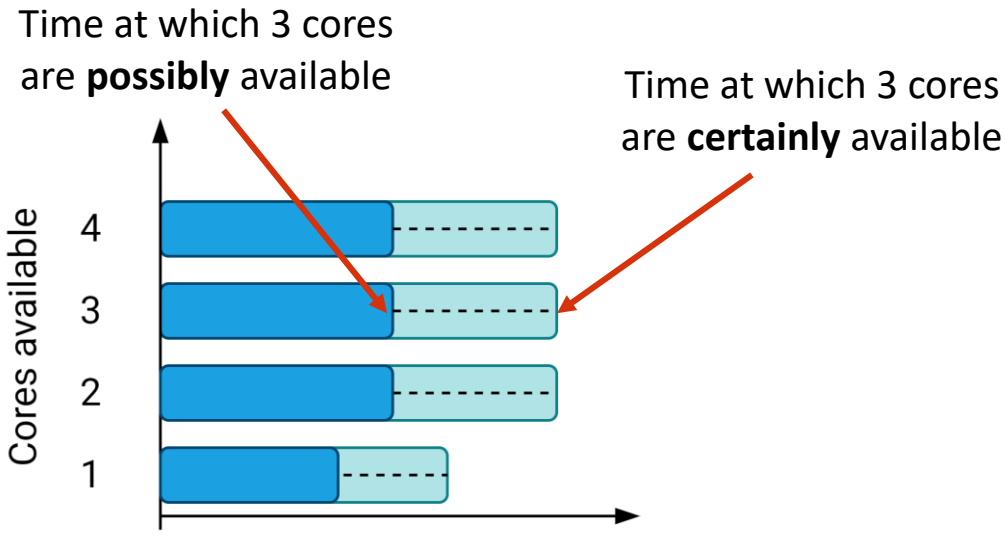
New system state representation

- Availabilities of cores ⟶ Know how many cores we have at time t



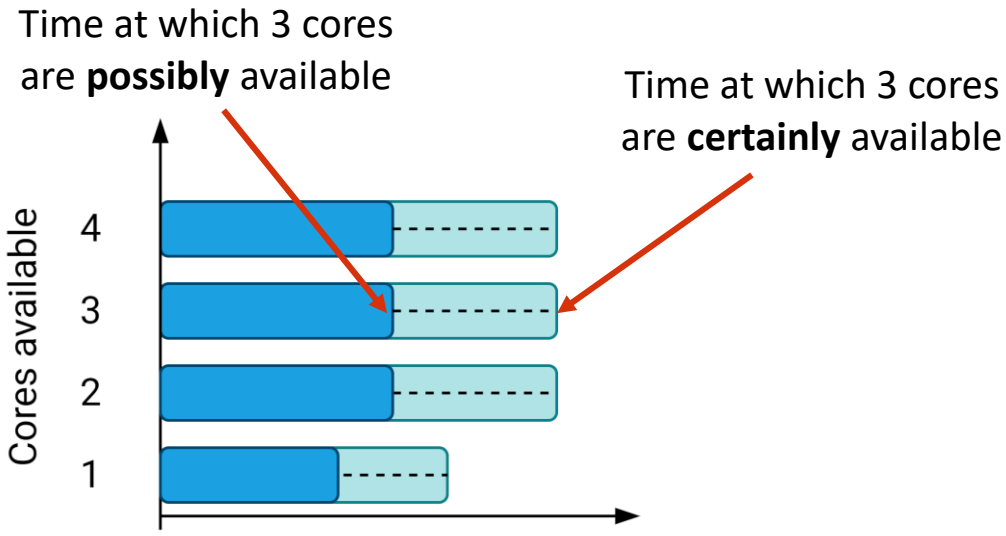
New system state representation

- Availabilities of cores → Know how many cores we have at time t
- Simultaneous groups of cores



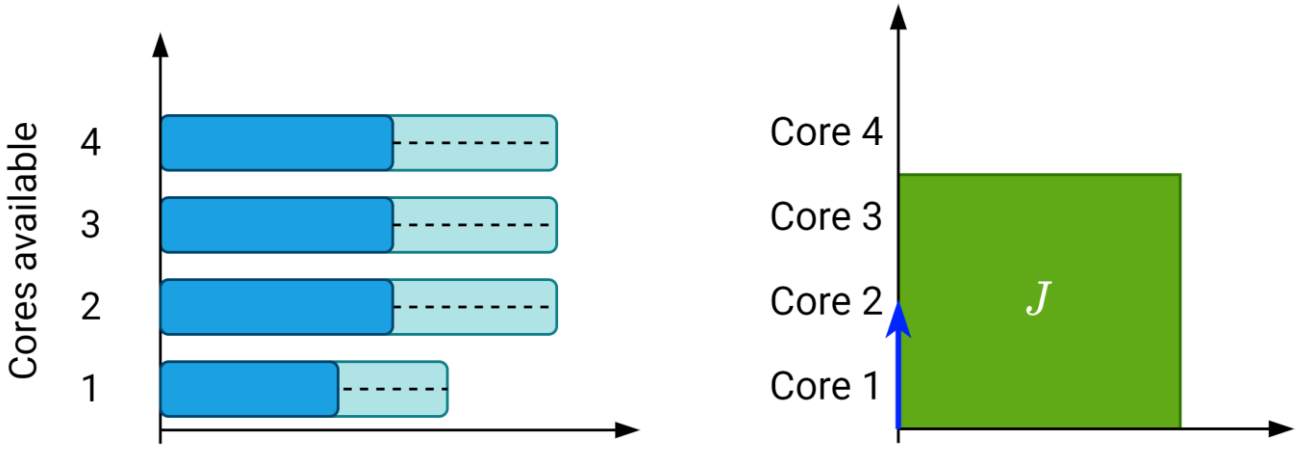
New system state representation

- Availabilities of cores → Know how many cores we have at time t
- Simultaneous groups of cores → Multiple cores released simultaneously



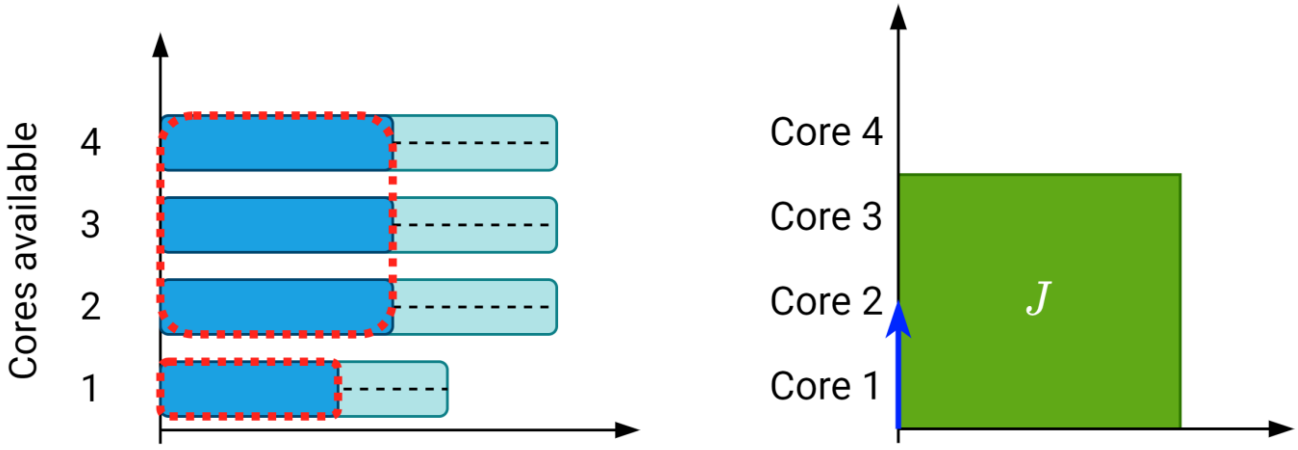
New system state representation

- Availabilities of cores → Know how many cores we have at time t
- Simultaneous groups of cores → Multiple cores released simultaneously



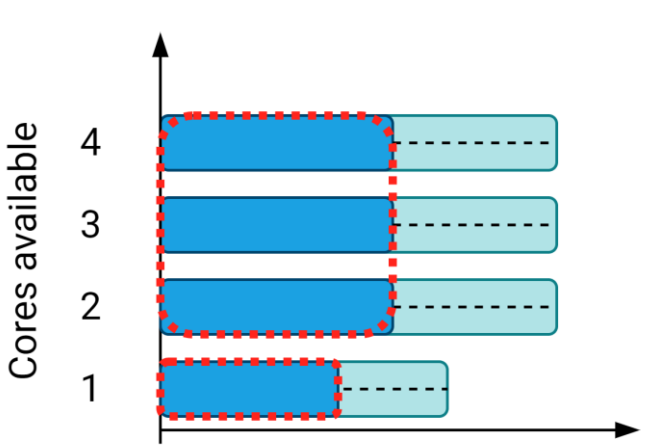
New system state representation

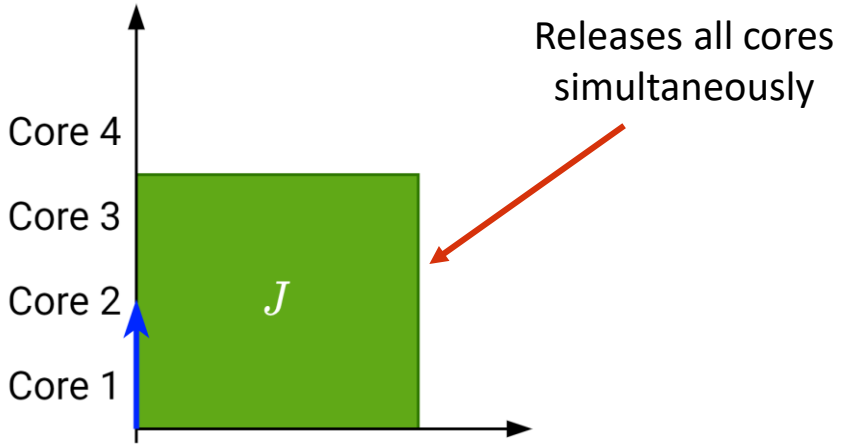
- Availabilities of cores → Know how many cores we have at time t
- Simultaneous groups of cores → Multiple cores released simultaneously



New system state representation

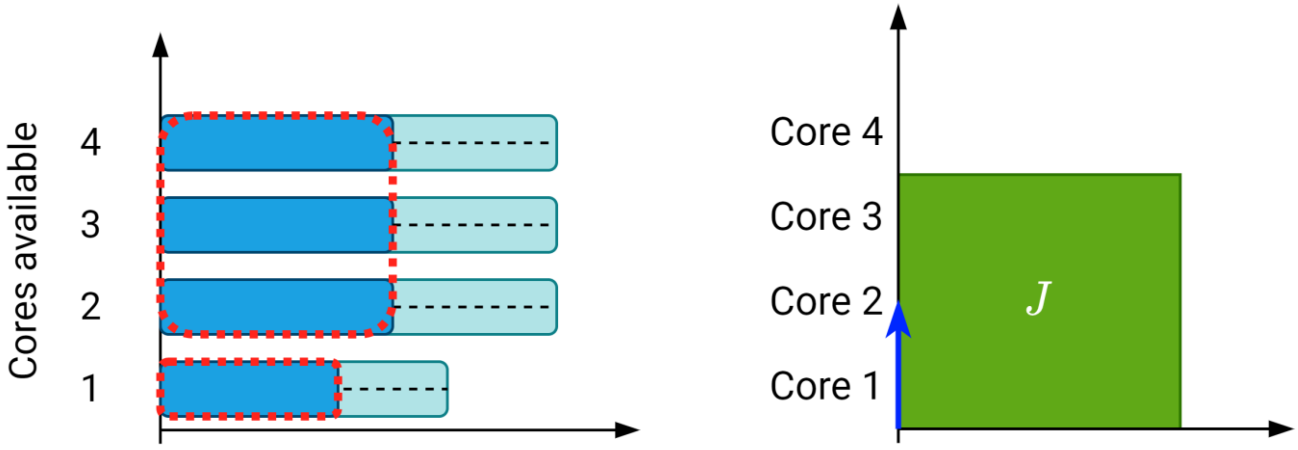
- Availabilities of cores → Know how many cores we have at time t
- Simultaneous groups of cores → Multiple cores released simultaneously





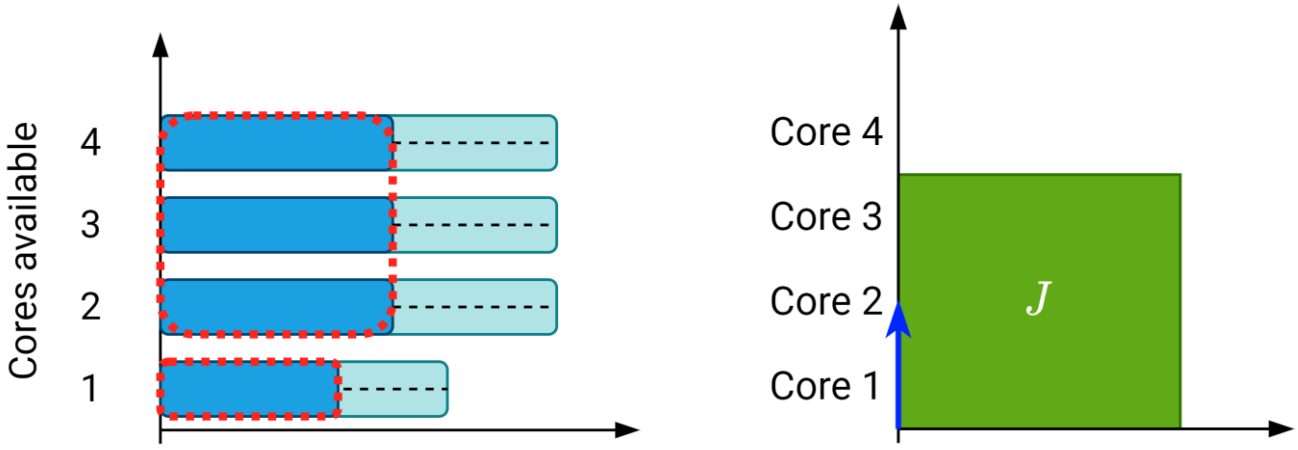
New system state representation

- 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



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

New SAG expansion rules

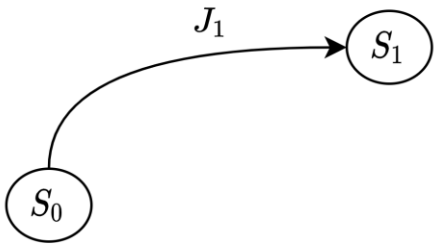
- Previously a state was created for every dispatchable job

New SAG expansion rules

- Previously a state was created for every dispatchable job

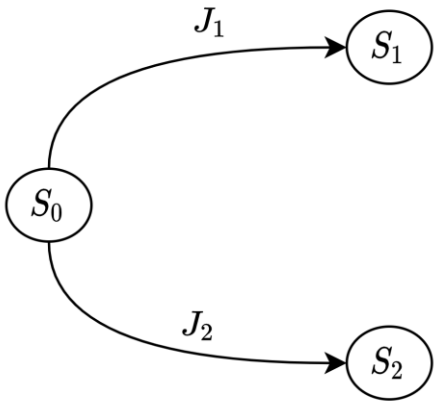
New SAG expansion rules

- Previously a state was created for every dispatchable job



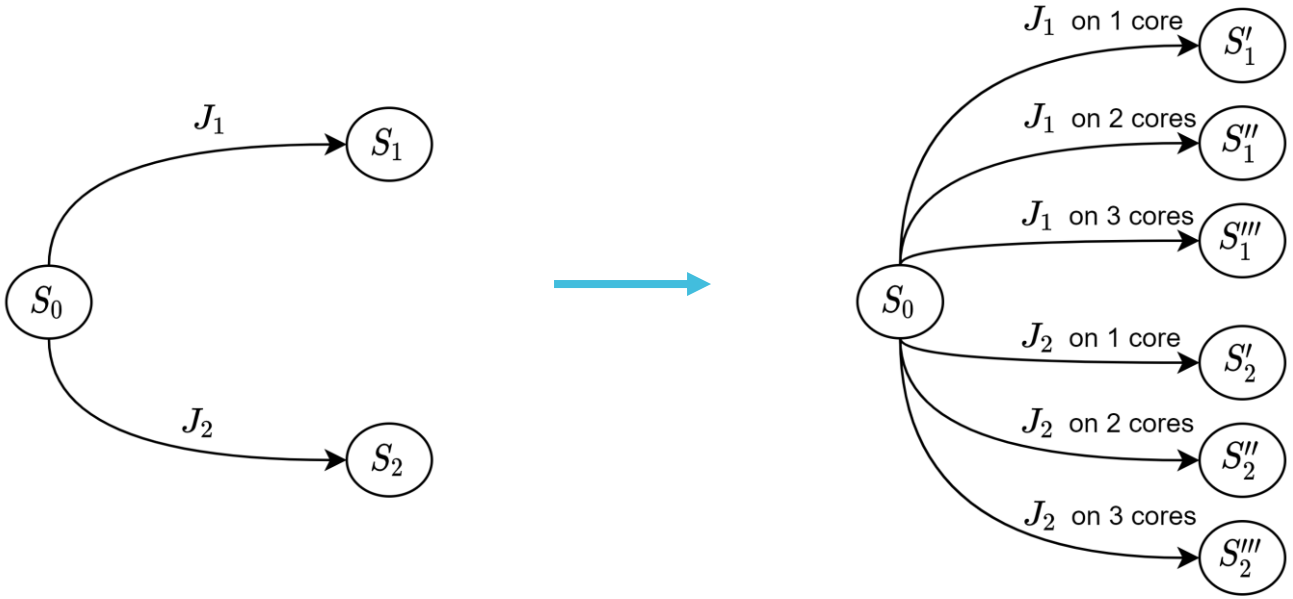
New SAG expansion rules

- Previously a state was created for every dispatchable job

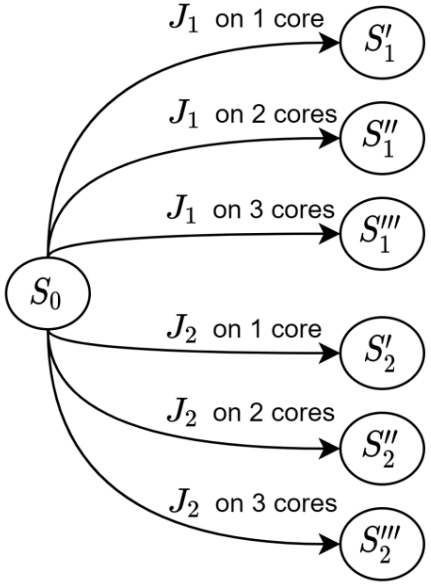


New SAG expansion rules

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

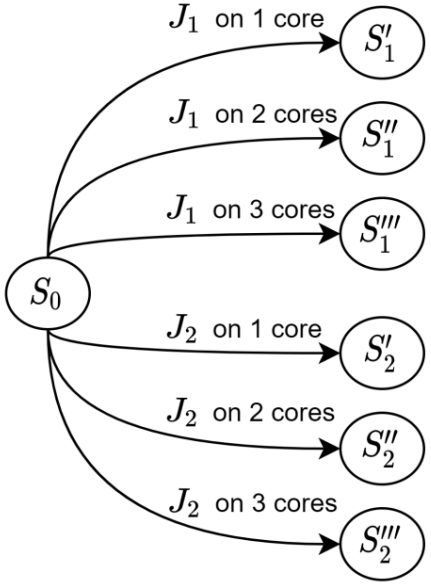


New SAG expansion rules



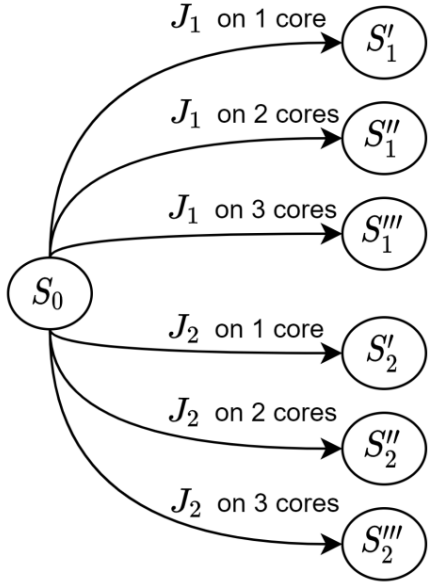
New SAG expansion rules

- Exploring more states is



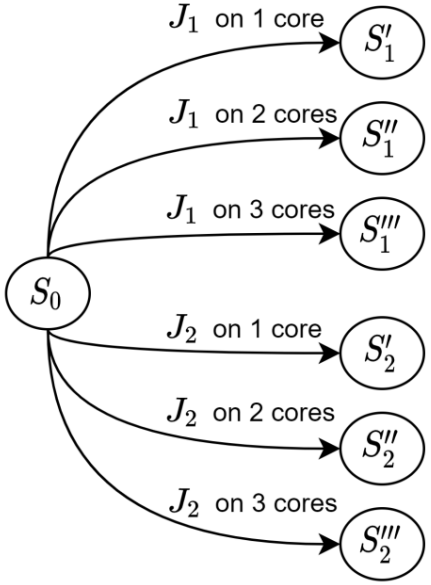
New SAG expansion rules

- Exploring more states is
  - 👍 Sound (does not accept unschedulable task sets)



New SAG expansion rules

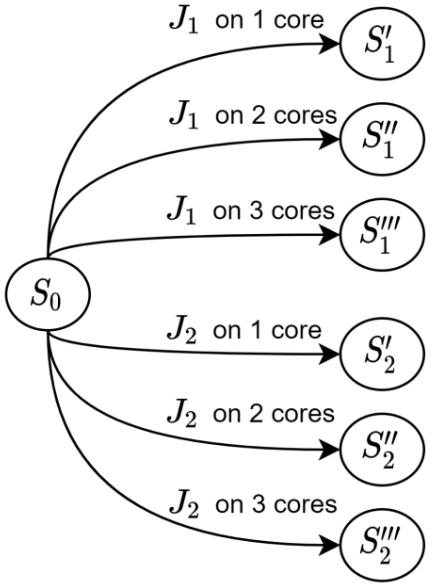
- Exploring more states is
  - 👍 Sound (does not accept unschedulable task sets)
  - 👎 Slower and more pessimistic



New SAG expansion rules

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

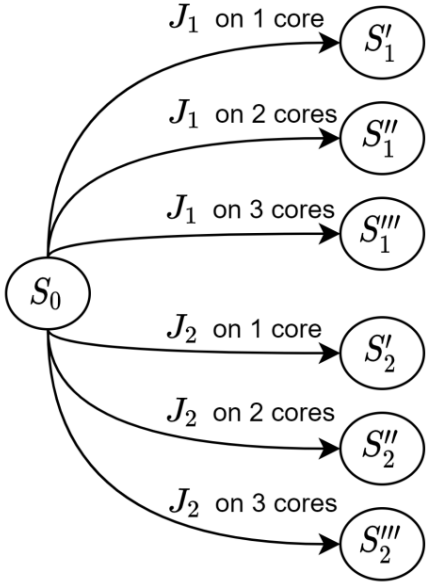
Try to minimize number of states that represent an impossible scenario



New SAG expansion rules

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

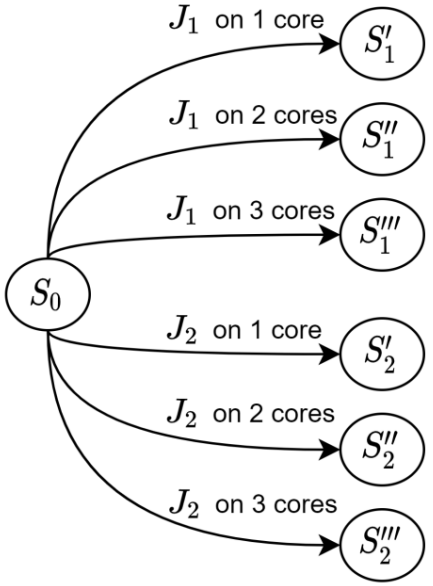
Try to minimize number of states that represent an impossible scenario



New SAG expansion rules

- Exploring more states is
  - Sound (does not accept unschedulable task sets)
  - Slower and more pessimistic
- Purge states by checking that
  - Enough cores available

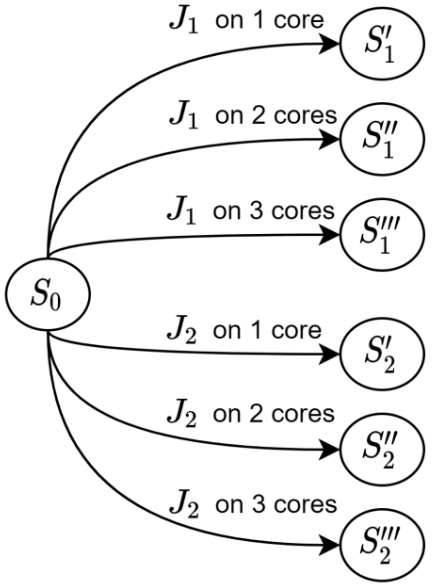
Try to minimize number of states that represent an impossible scenario



New SAG expansion rules

- Exploring more states is
  - 👍 Sound (does not accept unschedulable task sets)
  - 👎 Slower and more pessimistic
- Purge states by checking that
  - Enough cores available
  - More cores **not** available

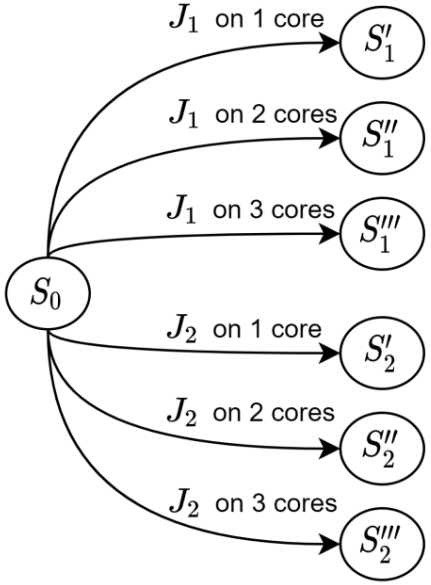
Try to minimize number of states that represent an impossible scenario



New SAG expansion rules

- Exploring more states is
  - 👍 Sound (does not accept unschedulable task sets)
  - 👎 Slower and more pessimistic
- Purge states by checking that
  - Enough cores available
  - More cores **not** available
  - Precedence constraints are respected

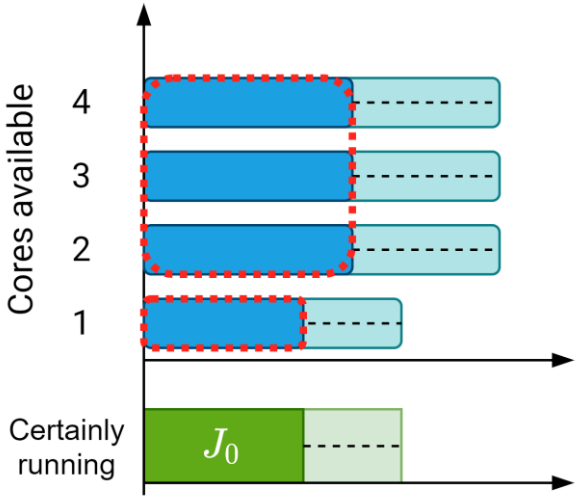
Try to minimize number of states that represent an impossible scenario

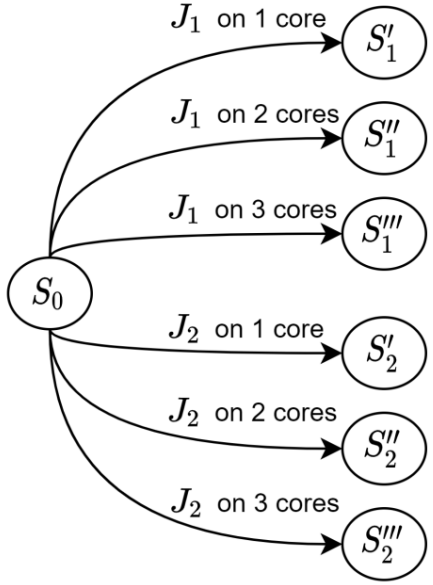


New SAG expansion rules

- Exploring more states is
  - 👍 Sound (does not accept unschedulable task sets)
  - 👎 Slower and more pessimistic
- Purge states by checking that
  - Enough cores available
  - More cores **not** available
  - Precedence constraints are respected

Try to minimize number of states that represent an impossible scenario

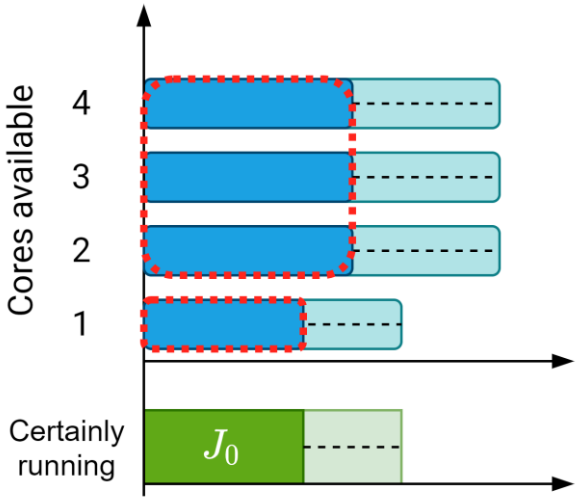


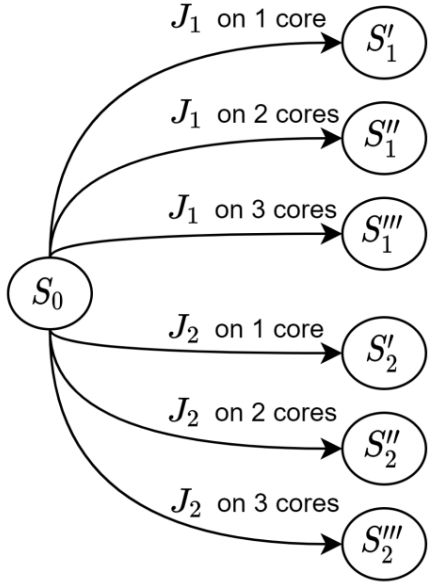


New SAG expansion rules

- Exploring more states is
  - 👍 Sound (does not accept unschedulable task sets)
  - 👎 Slower and more pessimistic
- Purge states by checking that
  - Enough cores available
  - More cores **not** available
  - Precedence constraints are respected

Try to minimize number of states that represent an impossible scenario





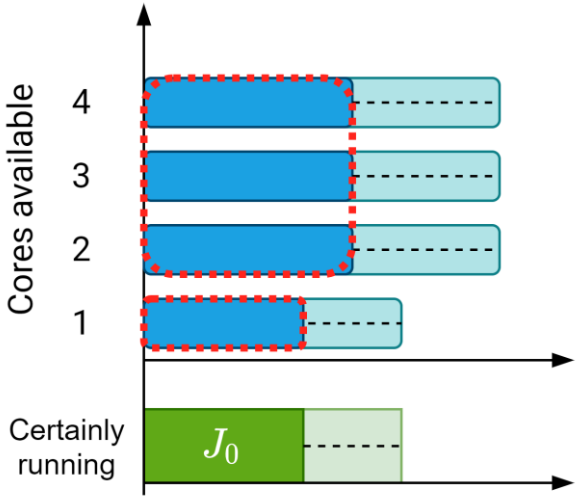
J_1 has higher priority

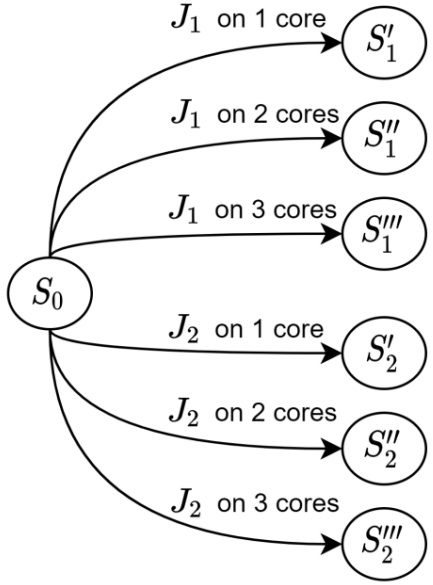
New SAG expansion rules

- 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





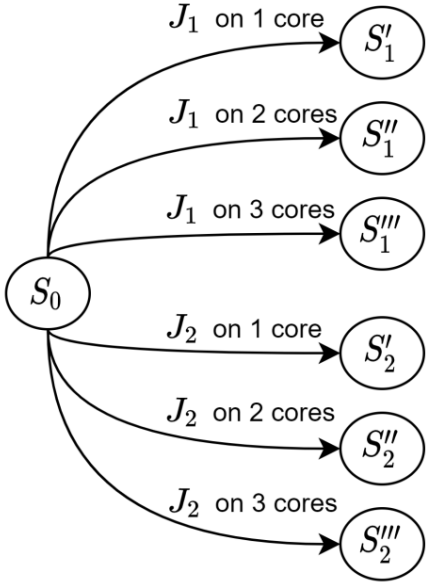
J_1 has higher priority

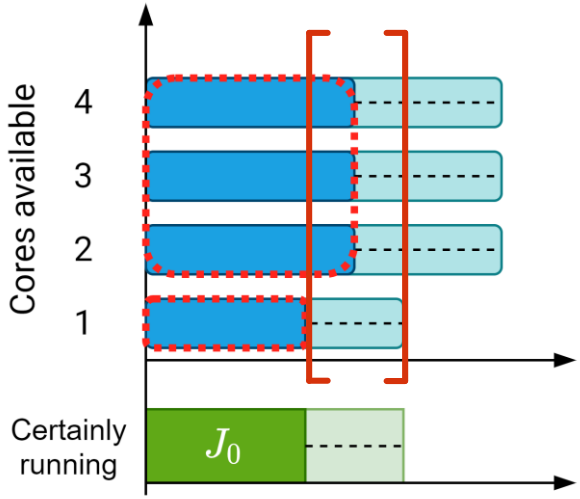
J_0 is predecessor of J_1

New SAG expansion rules

- Exploring more states is
  - Sound (does not accept unschedulable task sets)
  - Slower and more pessimistic
- Purge states by checking that
  - Enough cores available
  - More cores **not** available
  - Precedence constraints are respected

Try to minimize number of states that represent an impossible scenario





J_1 has higher priority

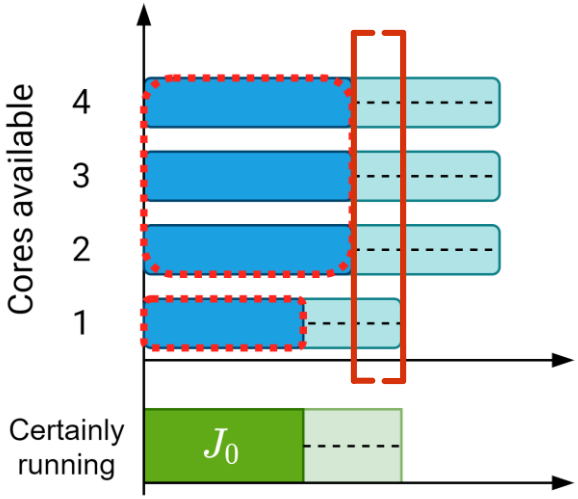
J_0 is predecessor of J_1

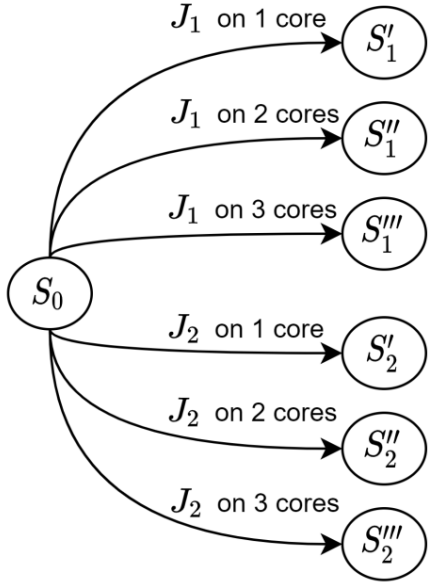
New SAG expansion rules

- 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

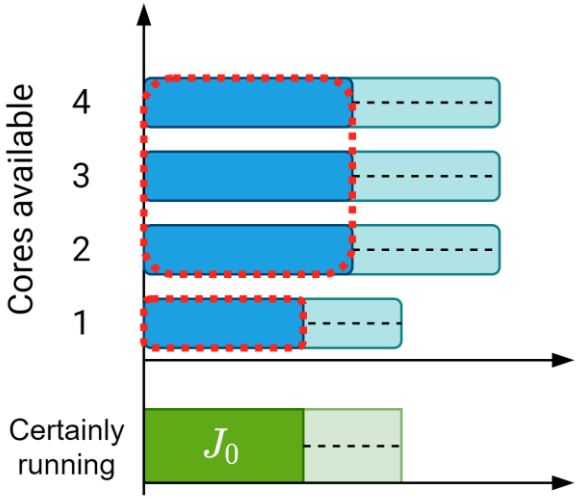
J_0 is predecessor of J_1

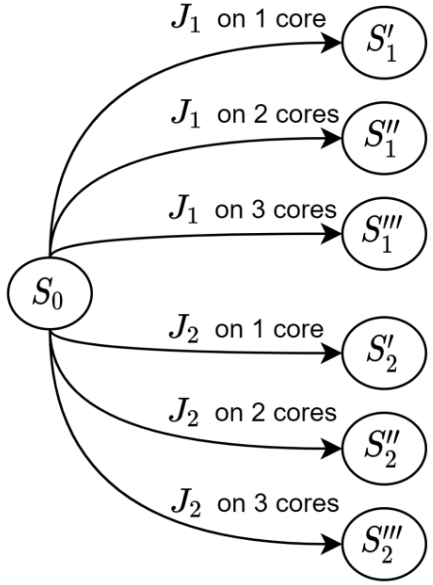
New SAG expansion rules

- 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

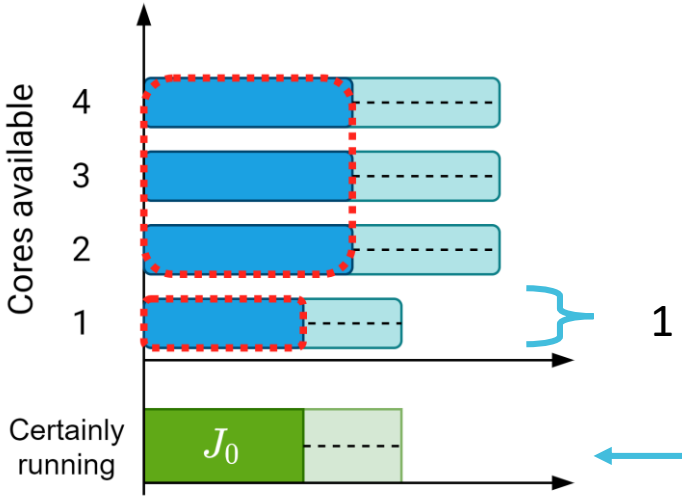
J_0 is predecessor of J_1

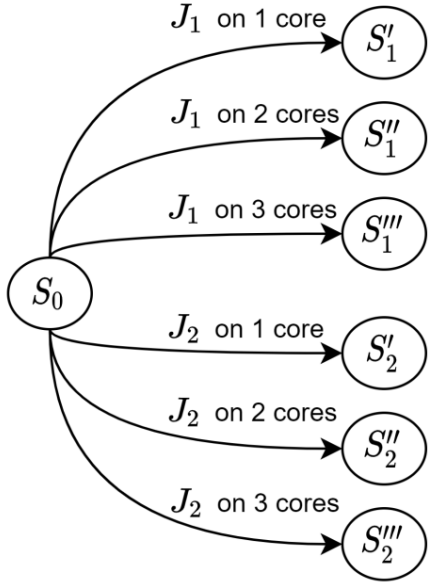
New SAG expansion rules

- 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

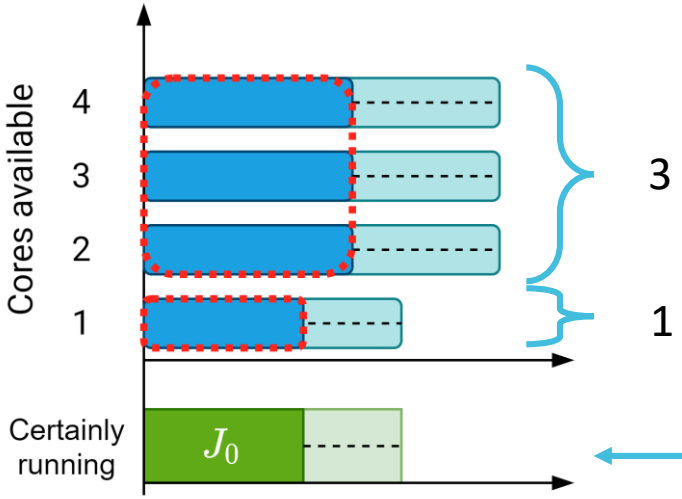
J_0 is predecessor of J_1

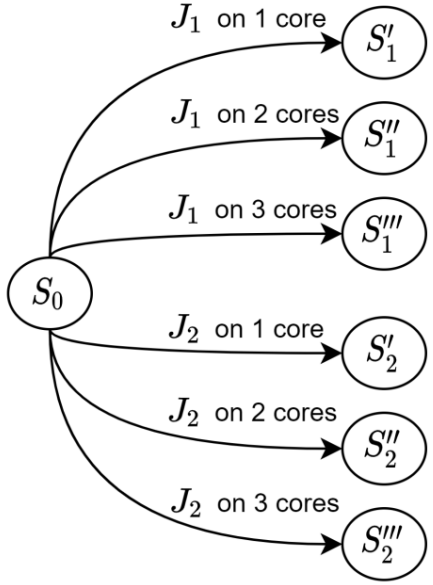
New SAG expansion rules

- 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

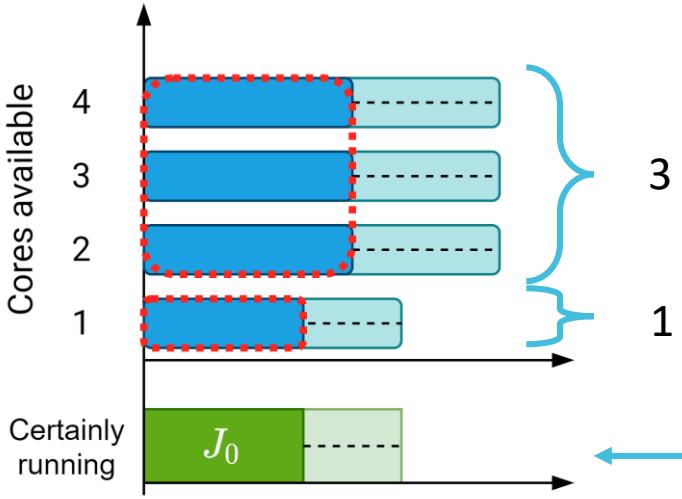
J_0 is predecessor of J_1

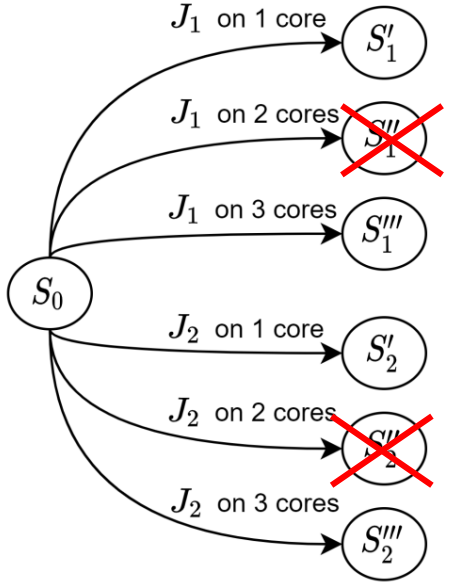
New SAG expansion rules

- 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

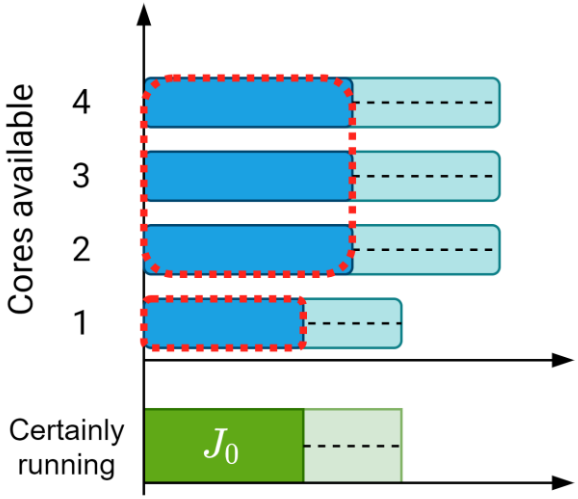
J_0 is predecessor of J_1

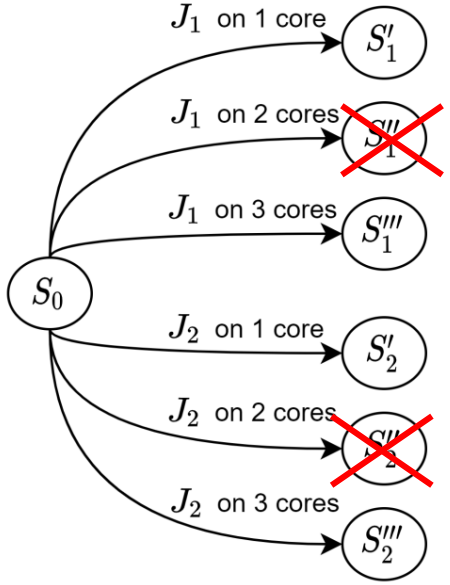
New SAG expansion rules

- 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

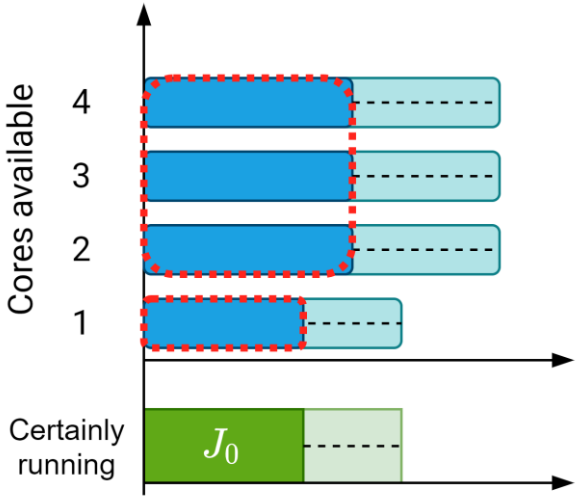
J_0 is predecessor of J_1

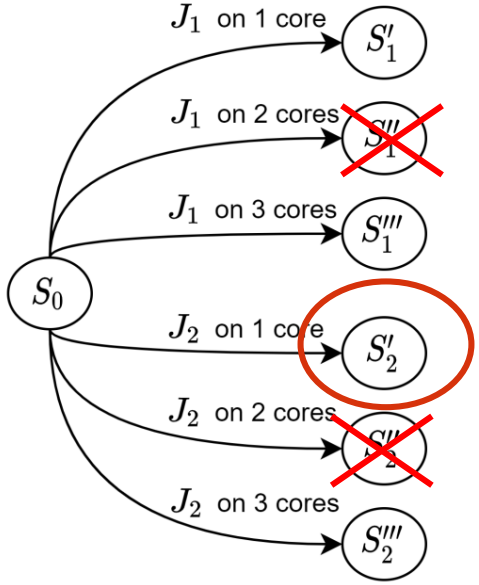
New SAG expansion rules

- 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

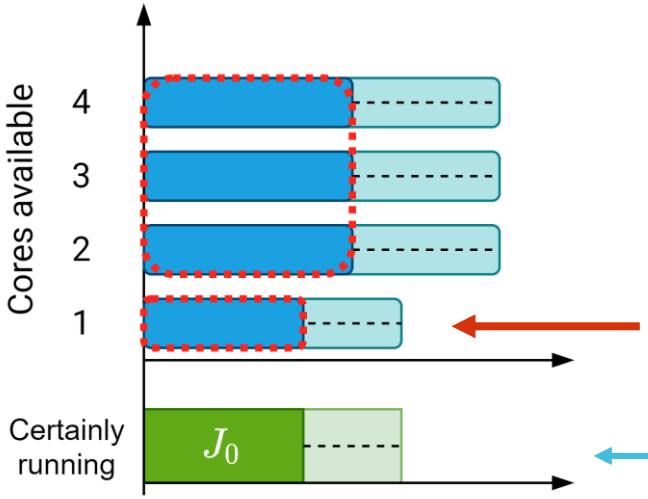
J_0 is predecessor of J_1

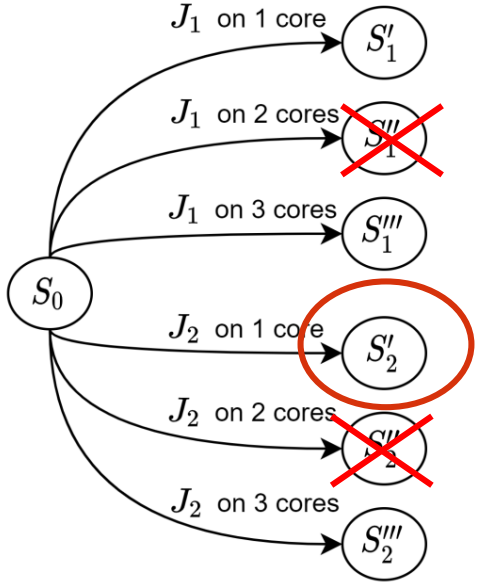
New SAG expansion rules

- 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

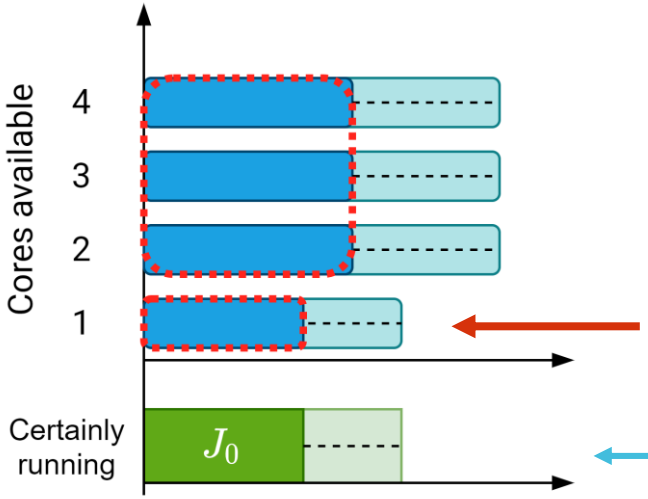
J_0 is predecessor of J_1

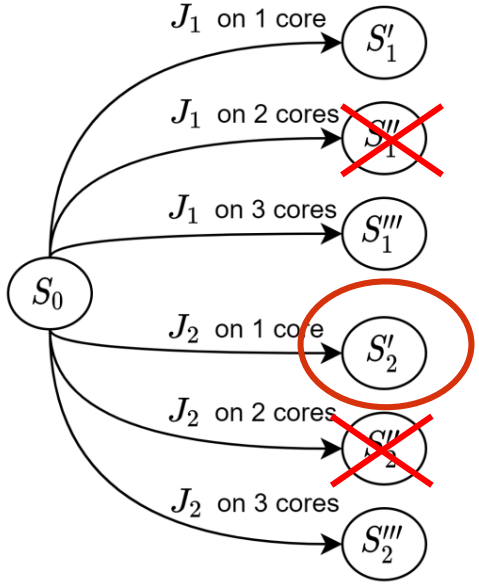
New SAG expansion rules

- 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

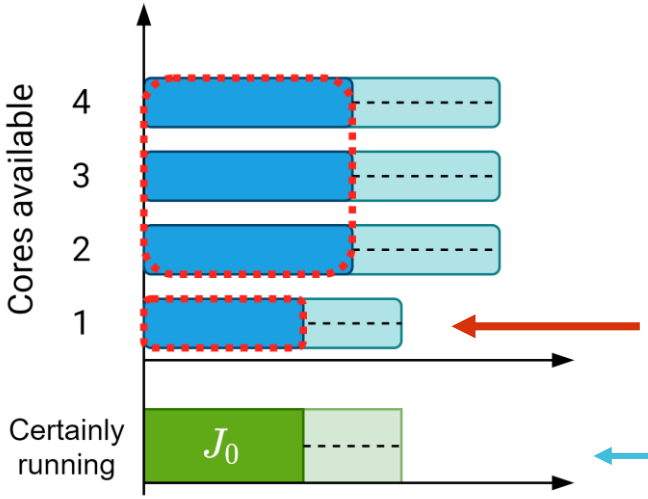
J_0 is predecessor of J_1

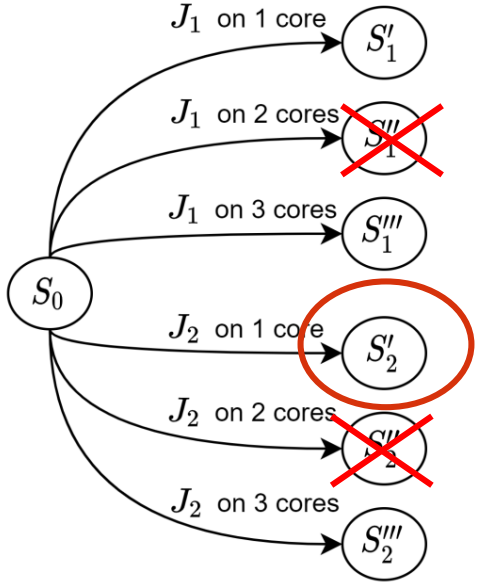
New SAG expansion rules

- 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

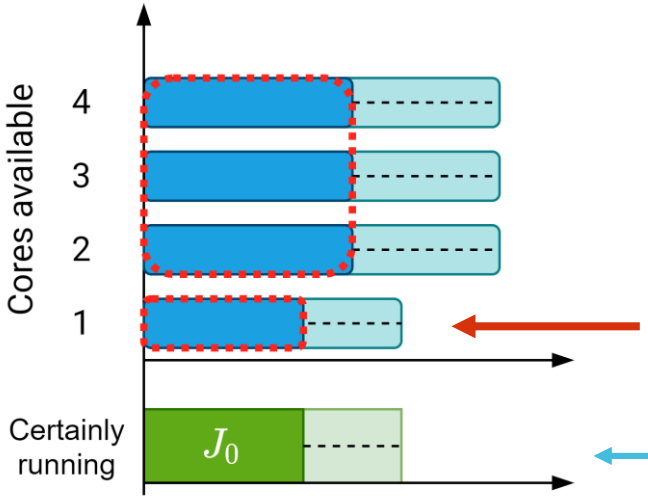
J_0 is predecessor of J_1

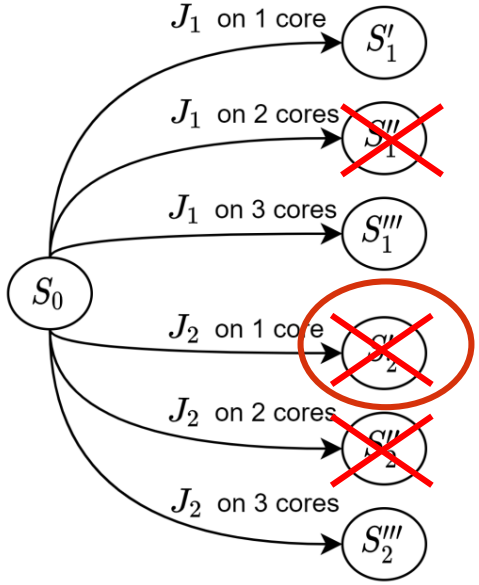
New SAG expansion rules

- 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

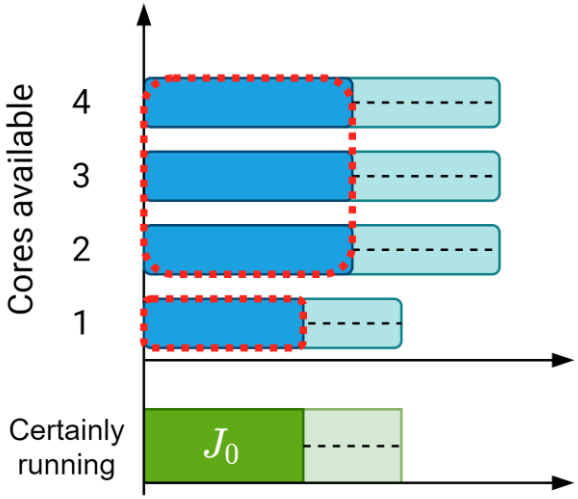
J_0 is predecessor of J_1

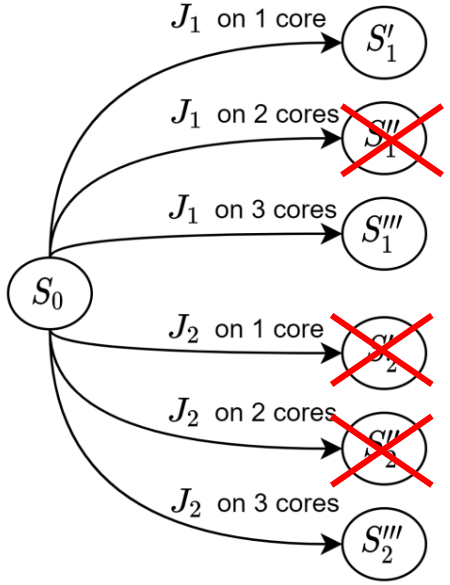
New SAG expansion rules

- 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

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

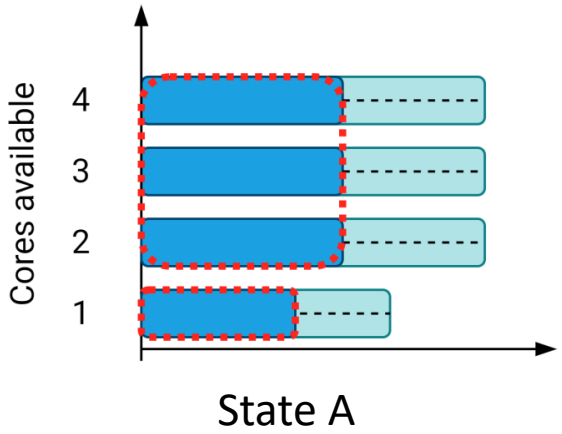
- We have to merge
  - Availability times
  - Groups of cores
  - Set of certainly running jobs

New SAG merge rules

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

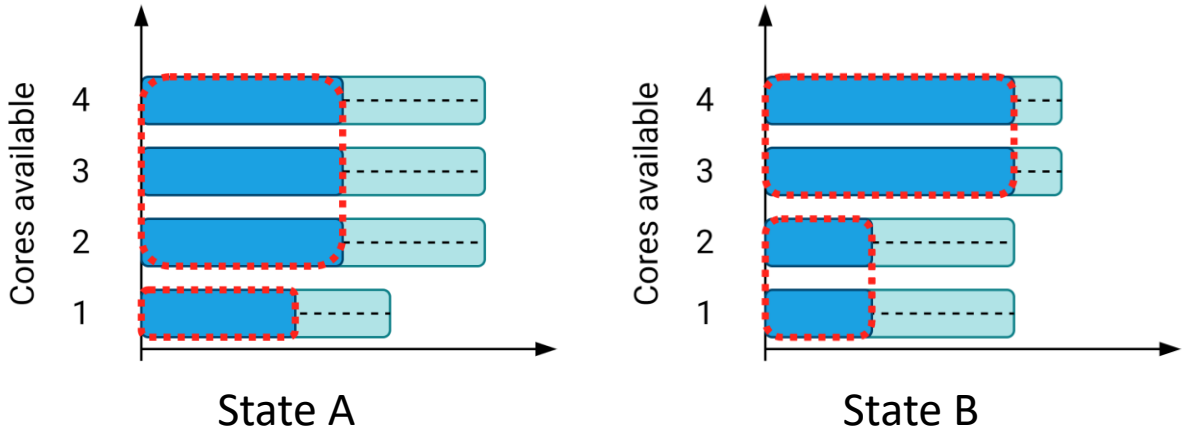
New SAG merge rules

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



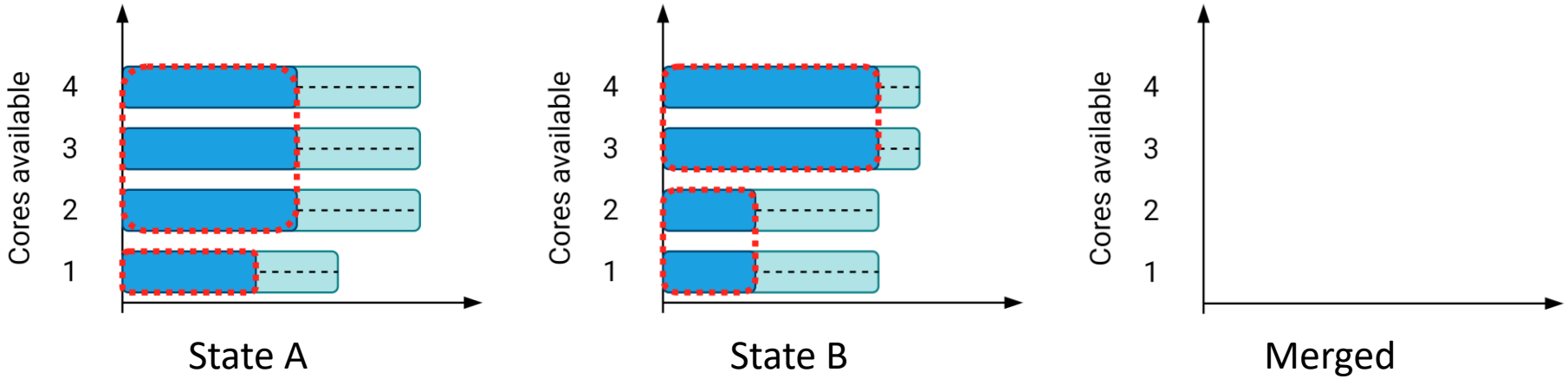
New SAG merge rules

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



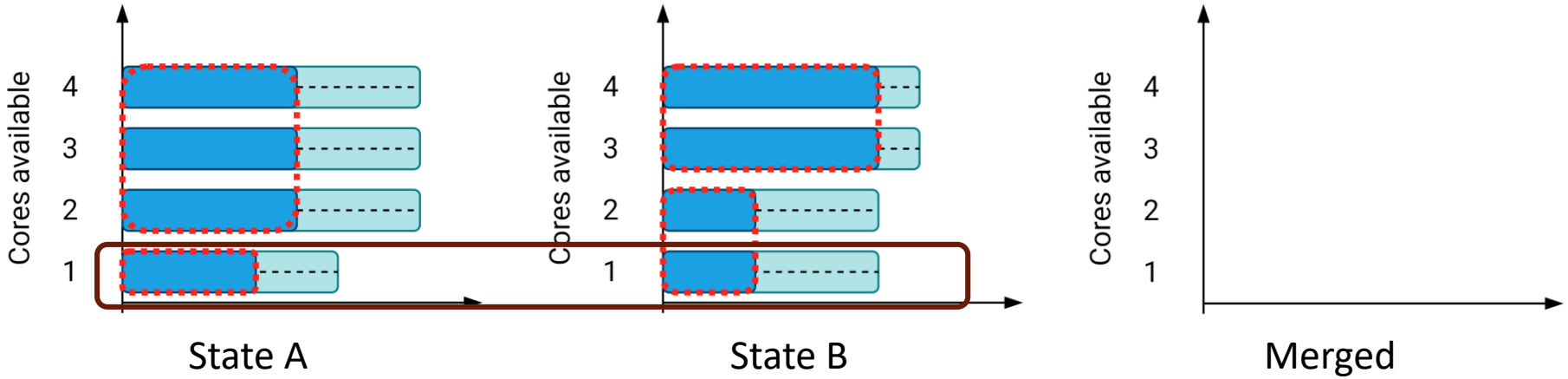
New SAG merge rules

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



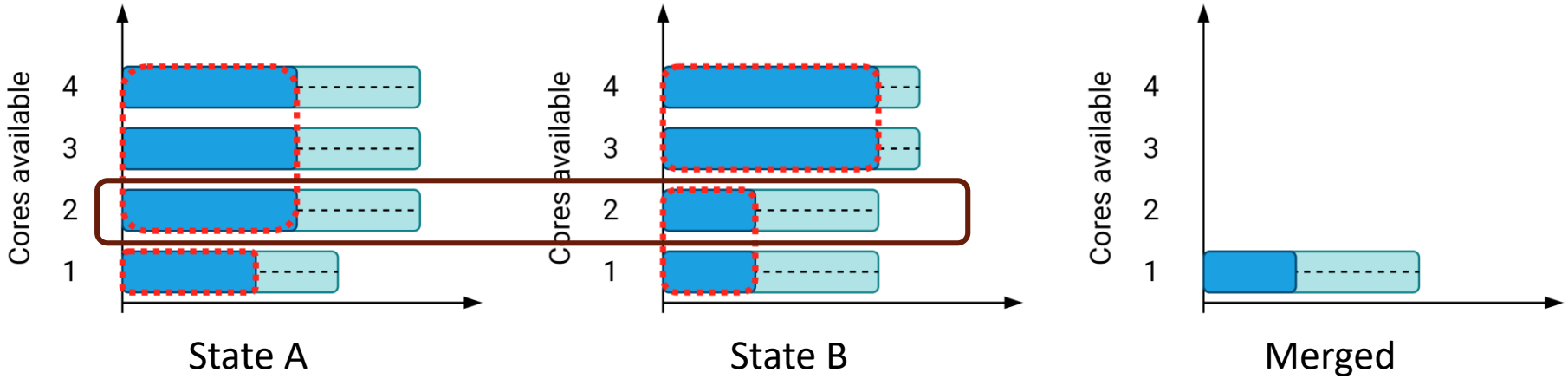
New SAG merge rules

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



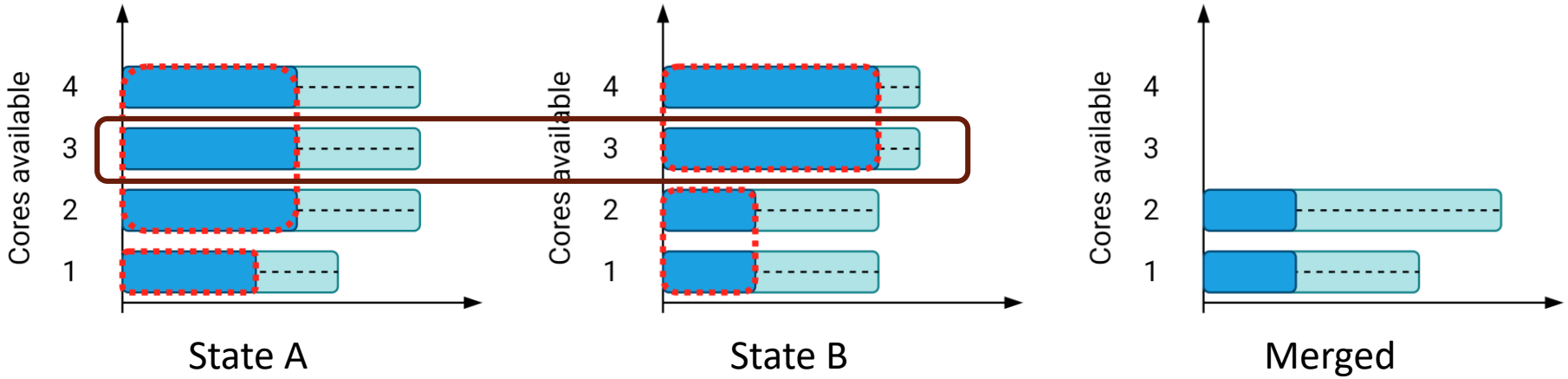
New SAG merge rules

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



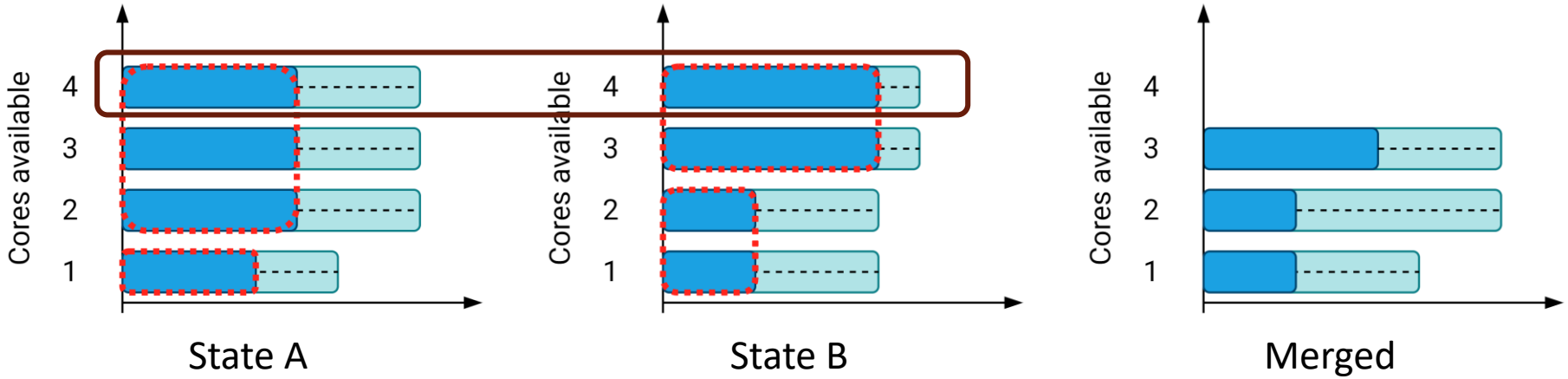
New SAG merge rules

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



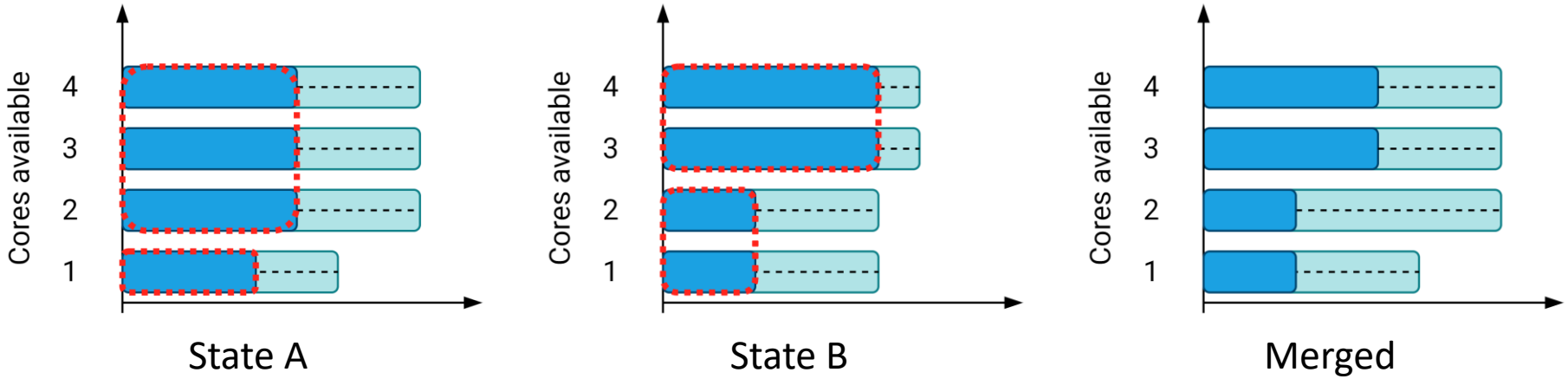
New SAG merge rules

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



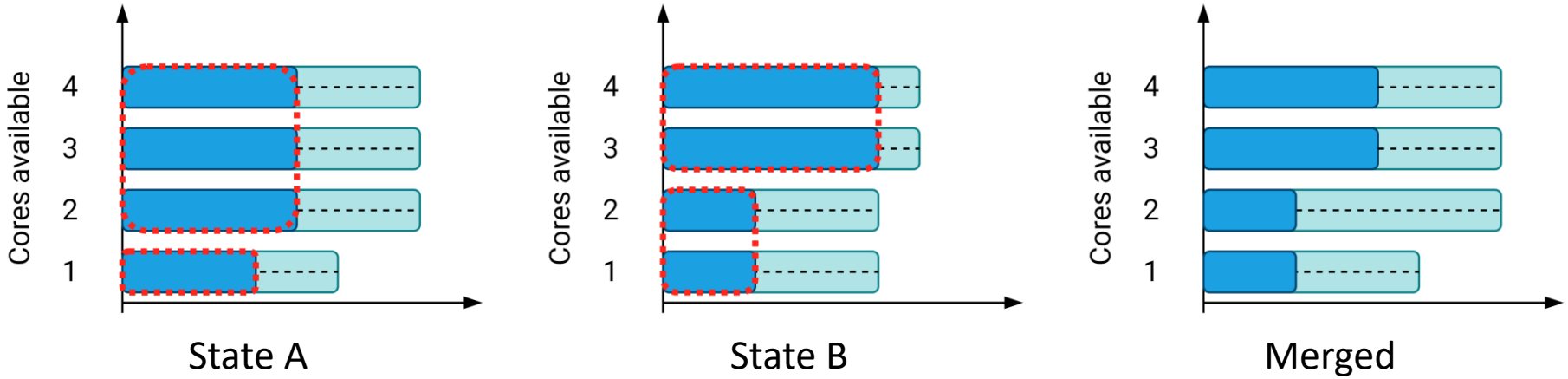
New SAG merge rules

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



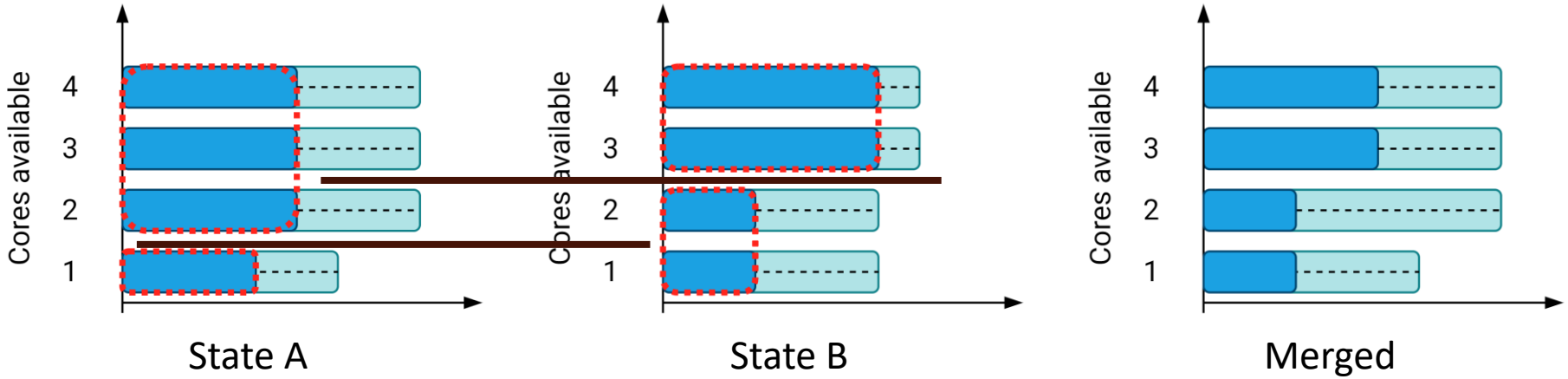
New SAG merge rules

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



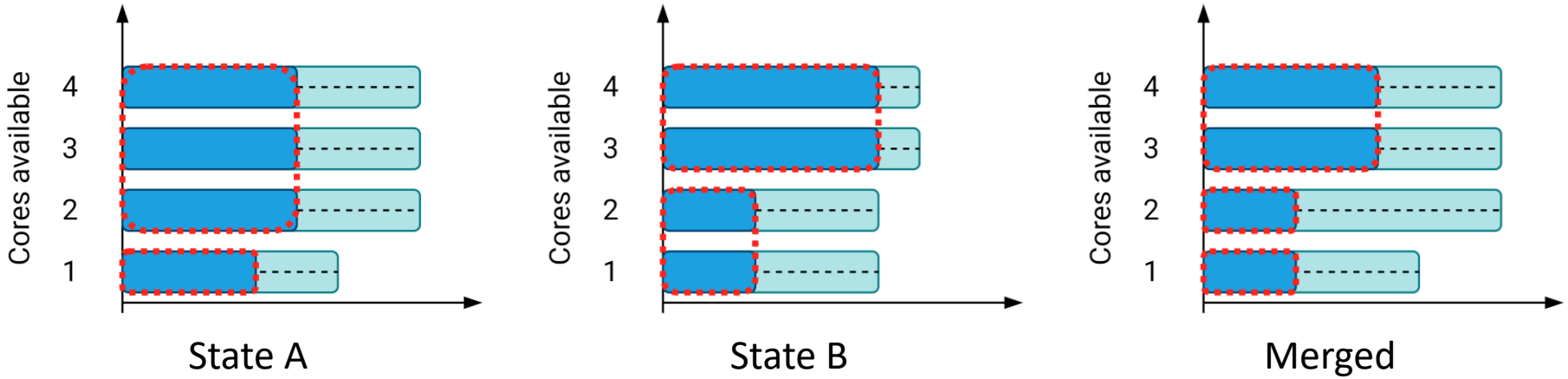
New SAG merge rules

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



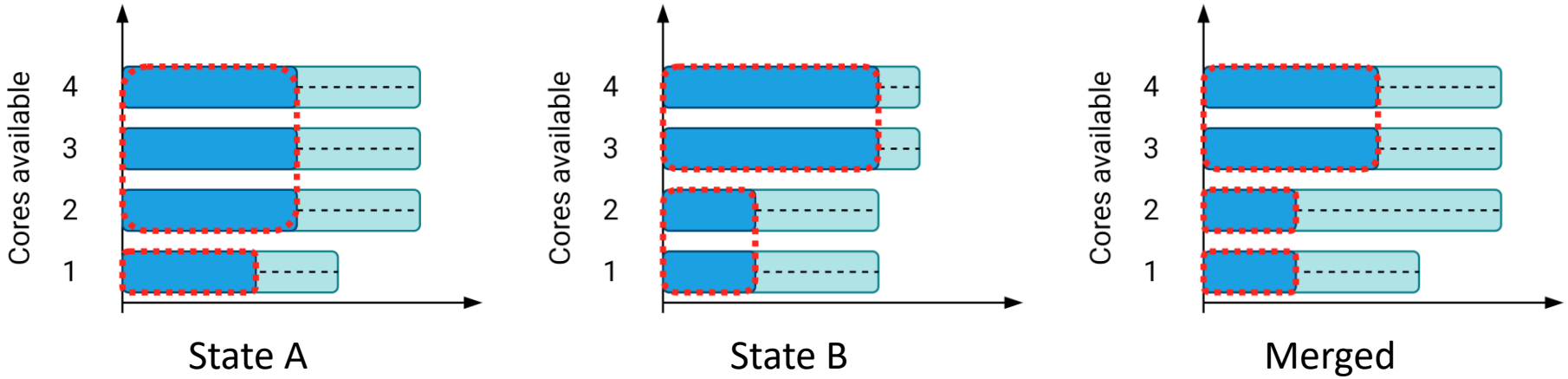
New SAG merge rules

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



New SAG merge rules

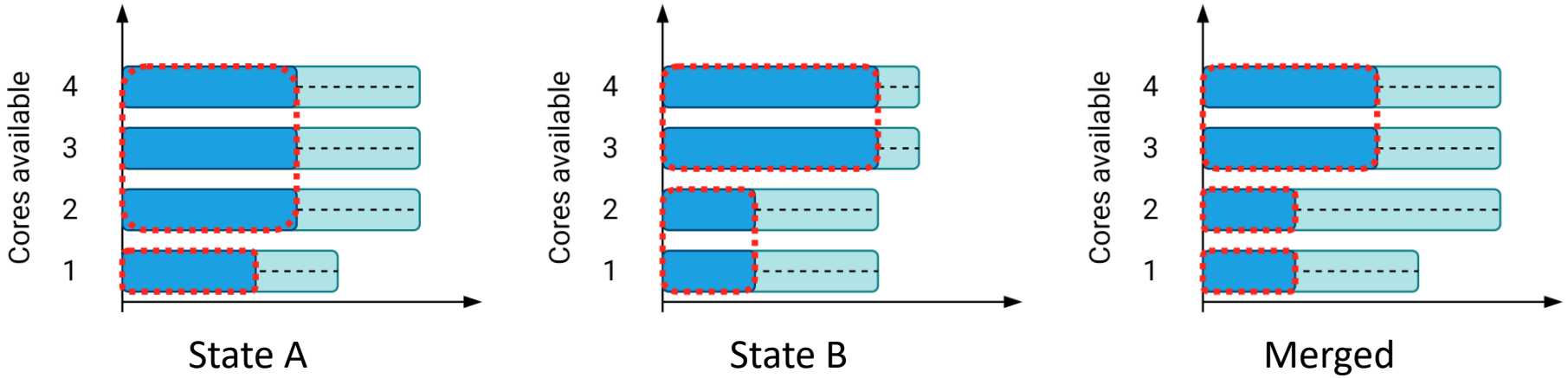
- We have to merge
  - Availability times → Extend the intervals
  - Groups of cores → Break the groups into same size
  - Set of certainly running jobs → Keep only jobs running in both states



New SAG merge rules

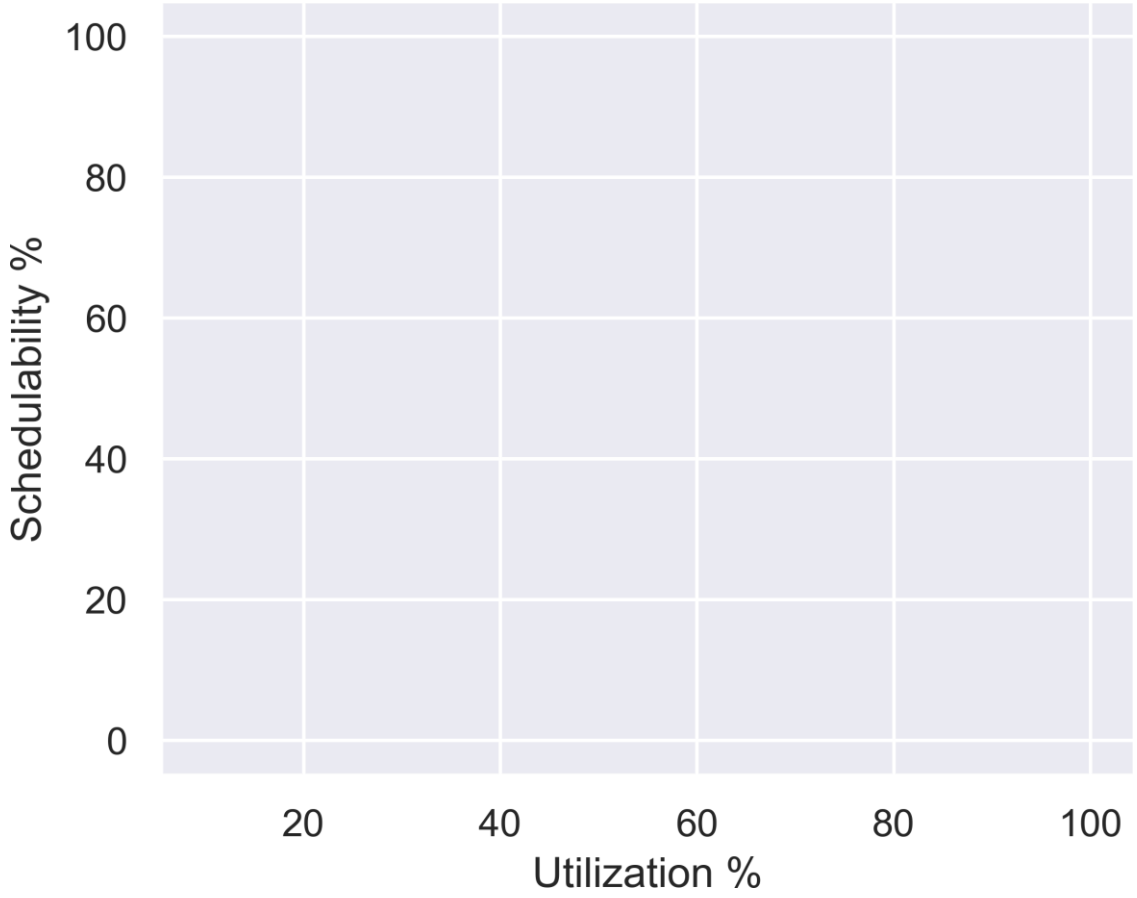
- We have to merge
  - Availability times → Extend the intervals
  - Groups of cores → Break the groups into same size
  - Set of certainly running jobs → Keep only jobs running in both states

Reachable states from merged state are all the states that can be reached from state A and state B



Our analysis results

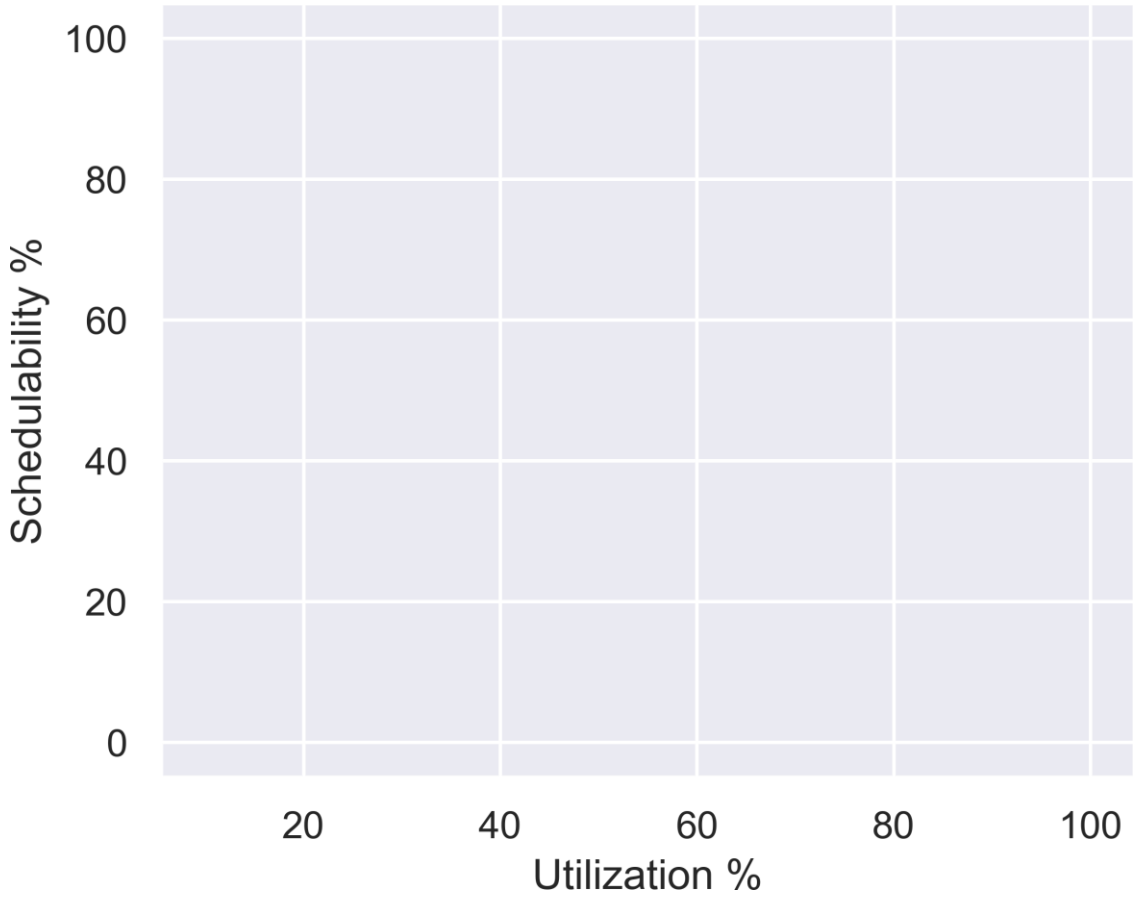
Our analysis results



Our analysis results

Randomly generated task sets

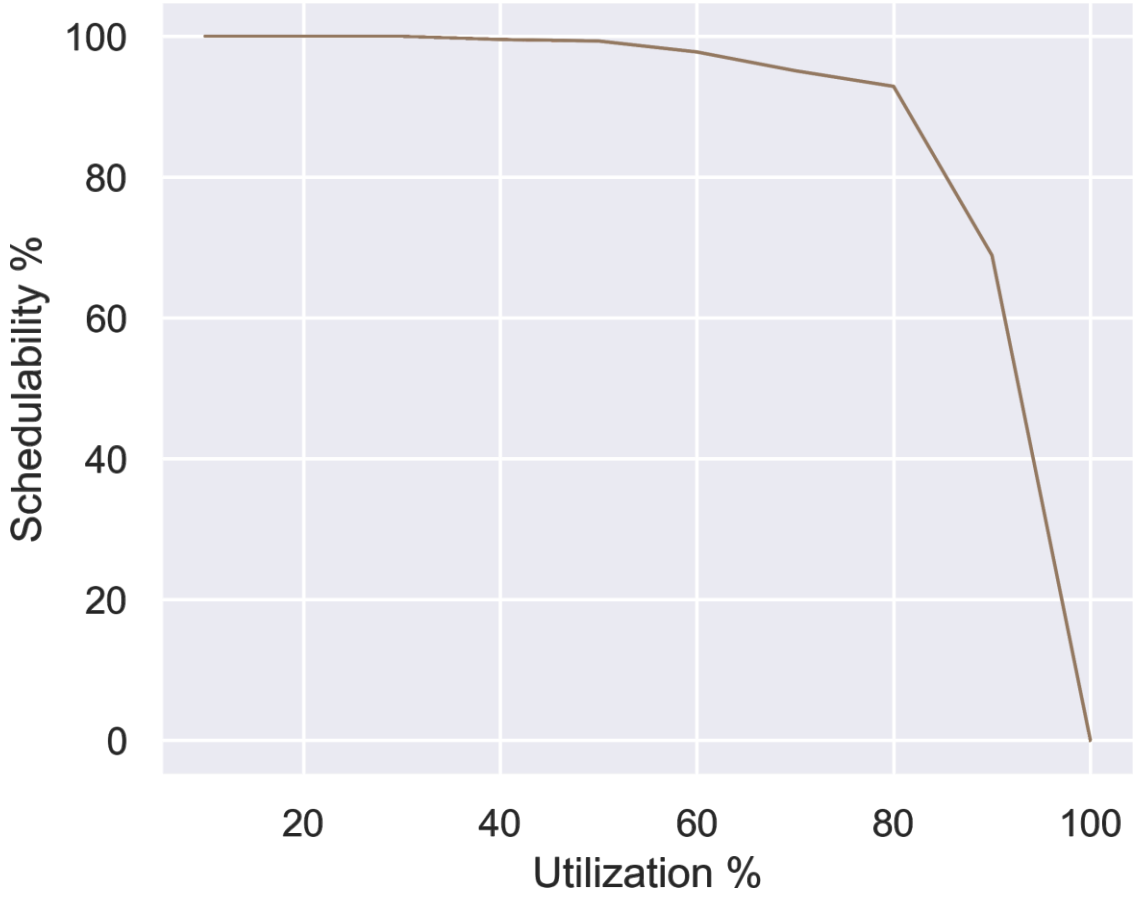
- System cores: 8
- System tasks: 20 rigid tasks
- Execution time variation: 50%
- Implicit deadlines
- Rate monotonic



Our analysis results

Randomly generated task sets

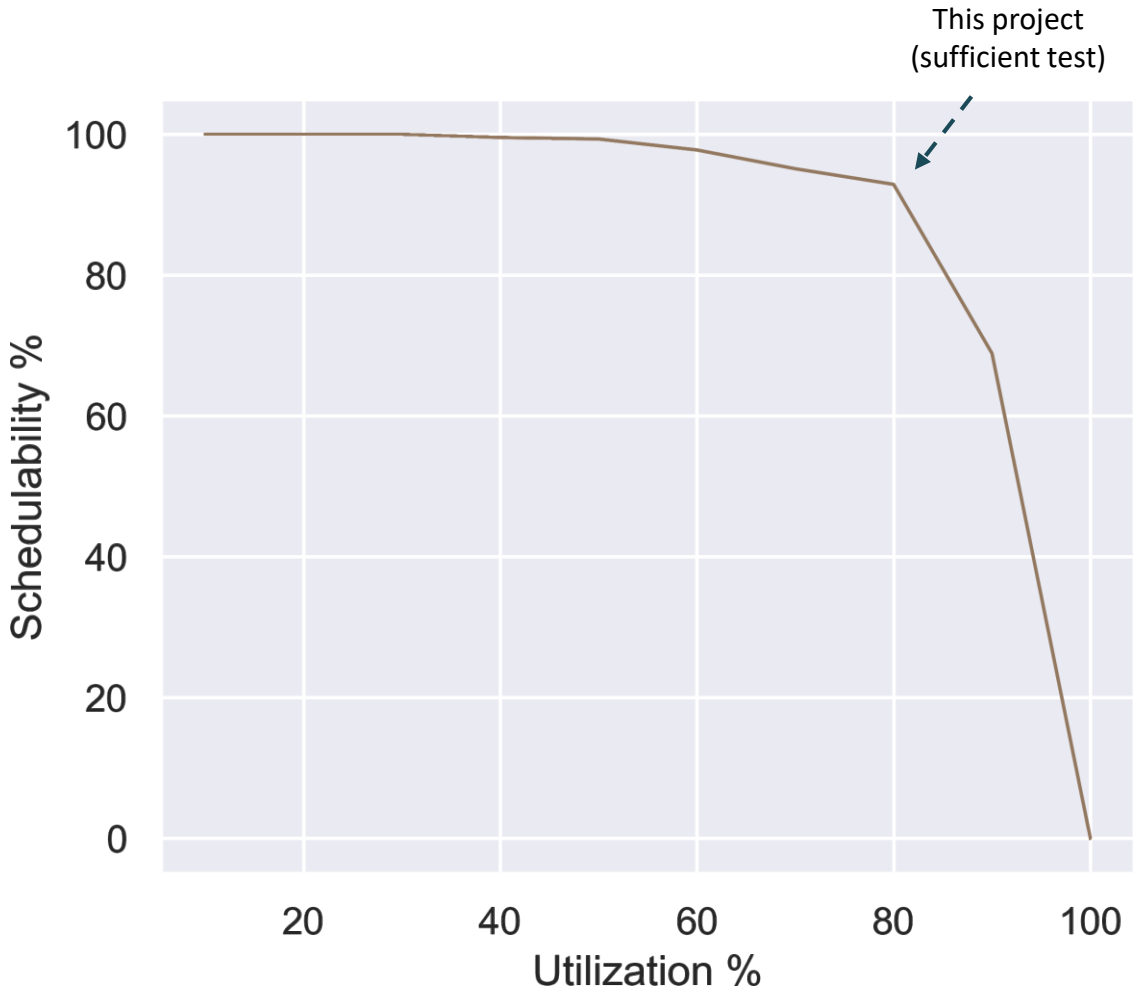
- System cores: 8
- System tasks: 20 rigid tasks
- Execution time variation: 50%
- Implicit deadlines
- Rate monotonic



Our analysis results

Randomly generated task sets

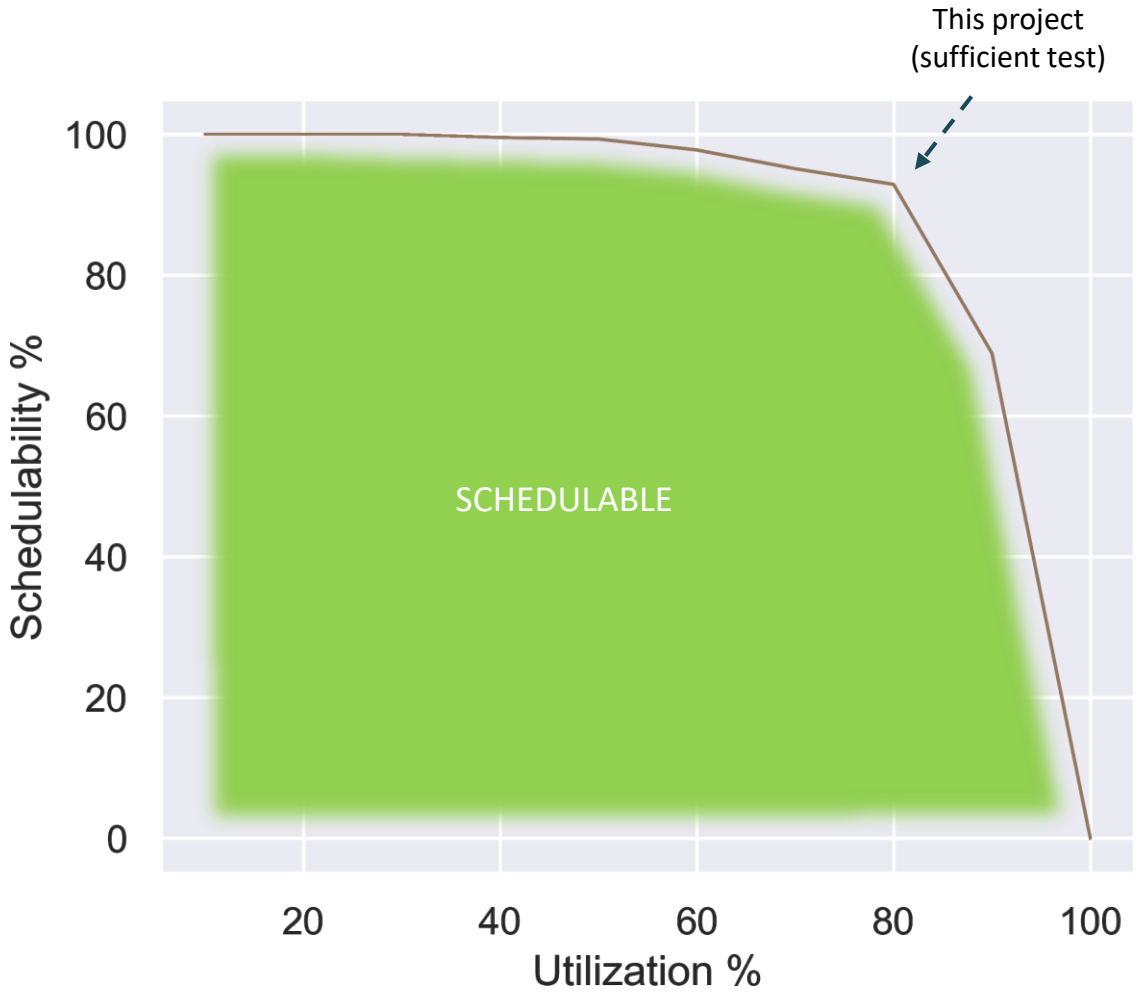
- System cores: 8
- System tasks: 20 rigid tasks
- Execution time variation: 50%
- Implicit deadlines
- Rate monotonic



Our analysis results

Randomly generated task sets

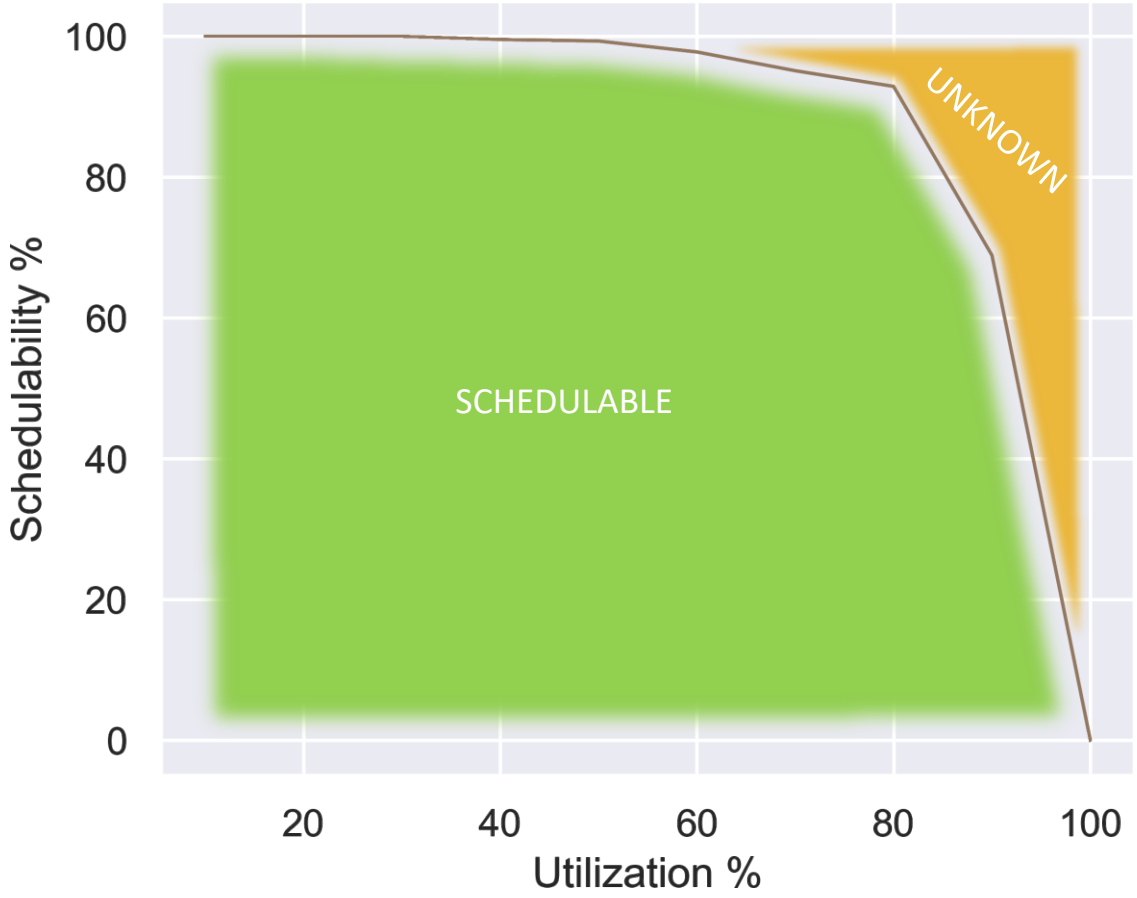
- System cores: 8
- System tasks: 20 rigid tasks
- Execution time variation: 50%
- Implicit deadlines
- Rate monotonic



Our analysis results

Randomly generated task sets

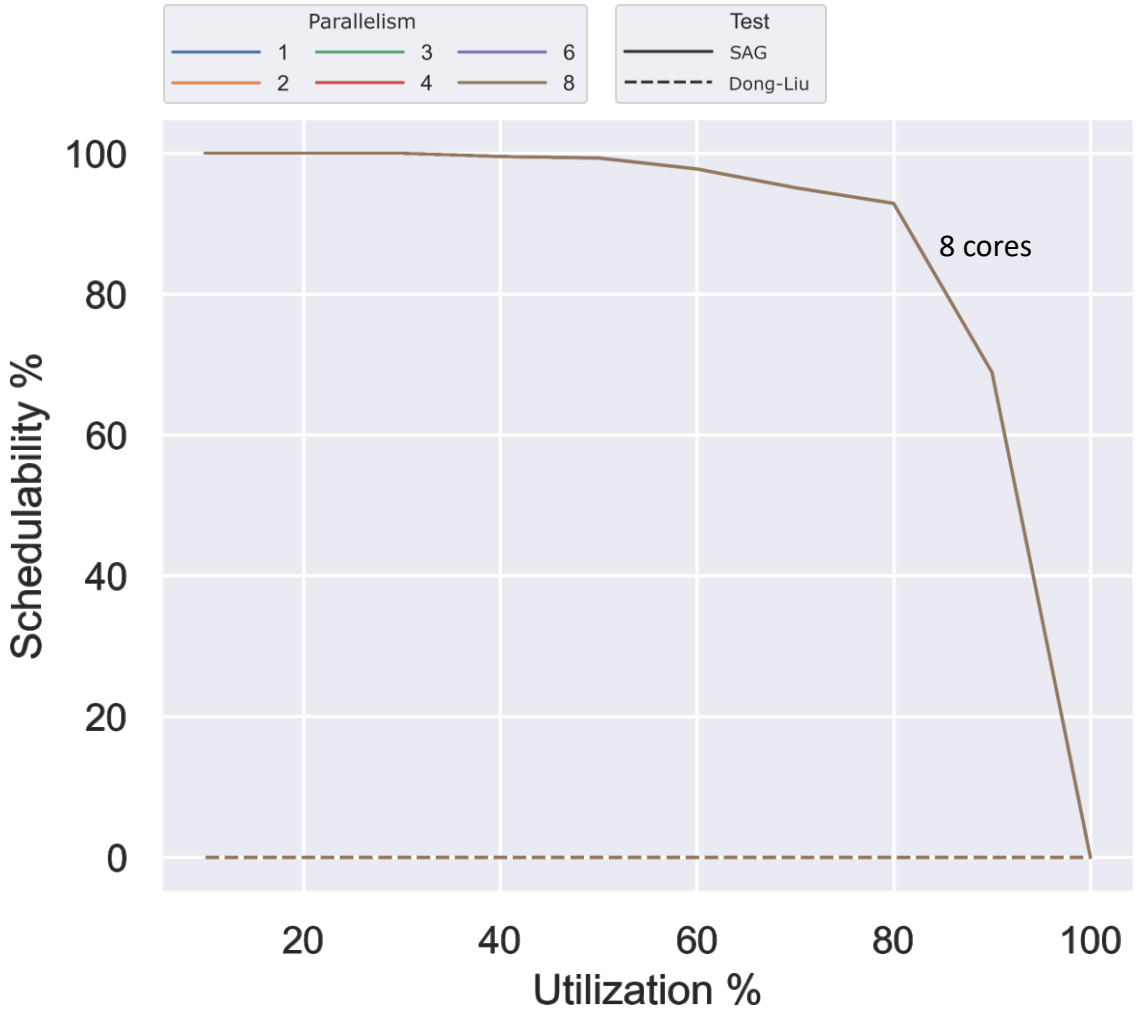
- System cores: 8
- System tasks: 20 rigid tasks
- Execution time variation: 50%
- Implicit deadlines
- Rate monotonic



Our analysis results

Randomly generated task sets

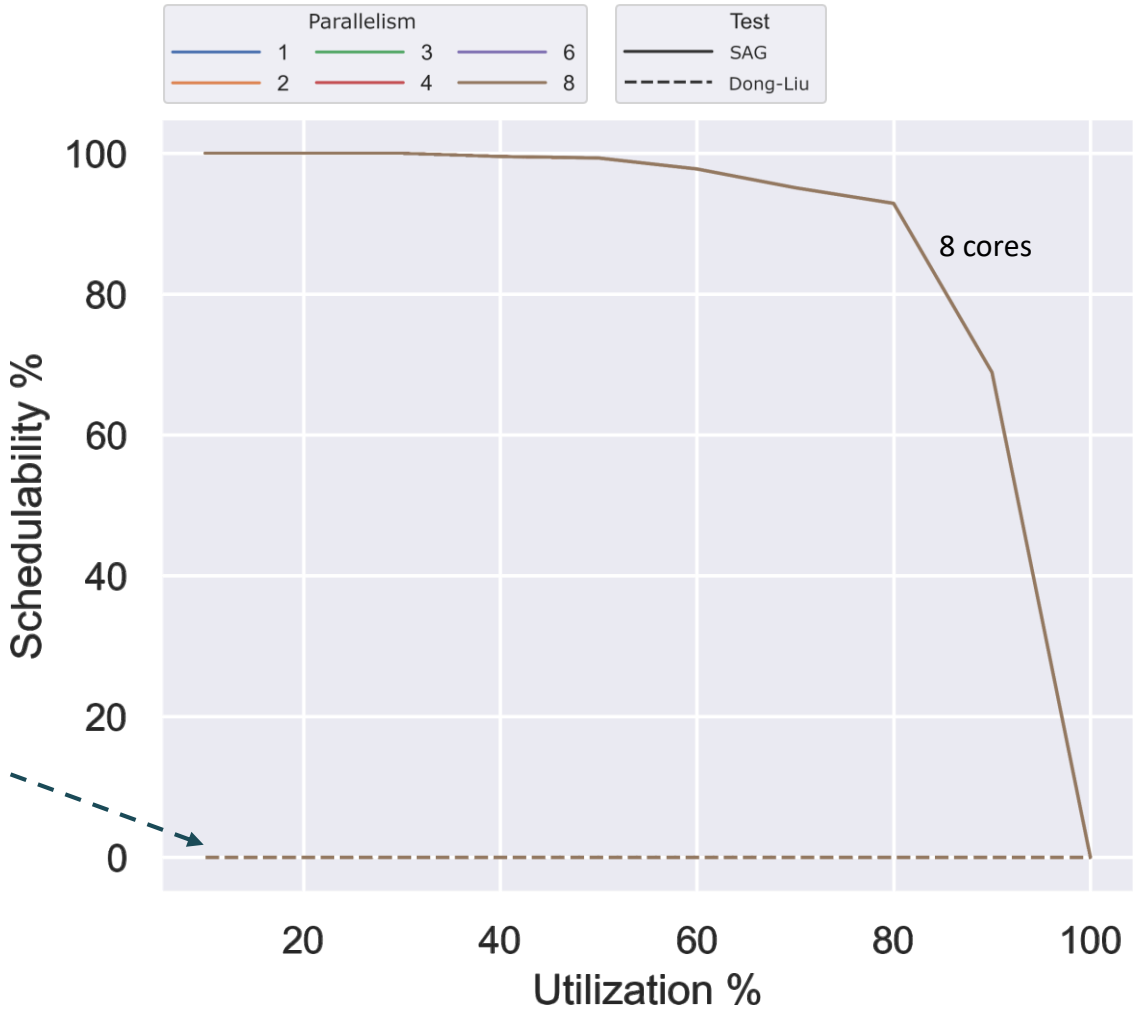
- System cores: 8
- System tasks: 20 rigid tasks
- Execution time variation: 50%
- Implicit deadlines
- Rate monotonic



Our analysis results

Randomly generated task sets

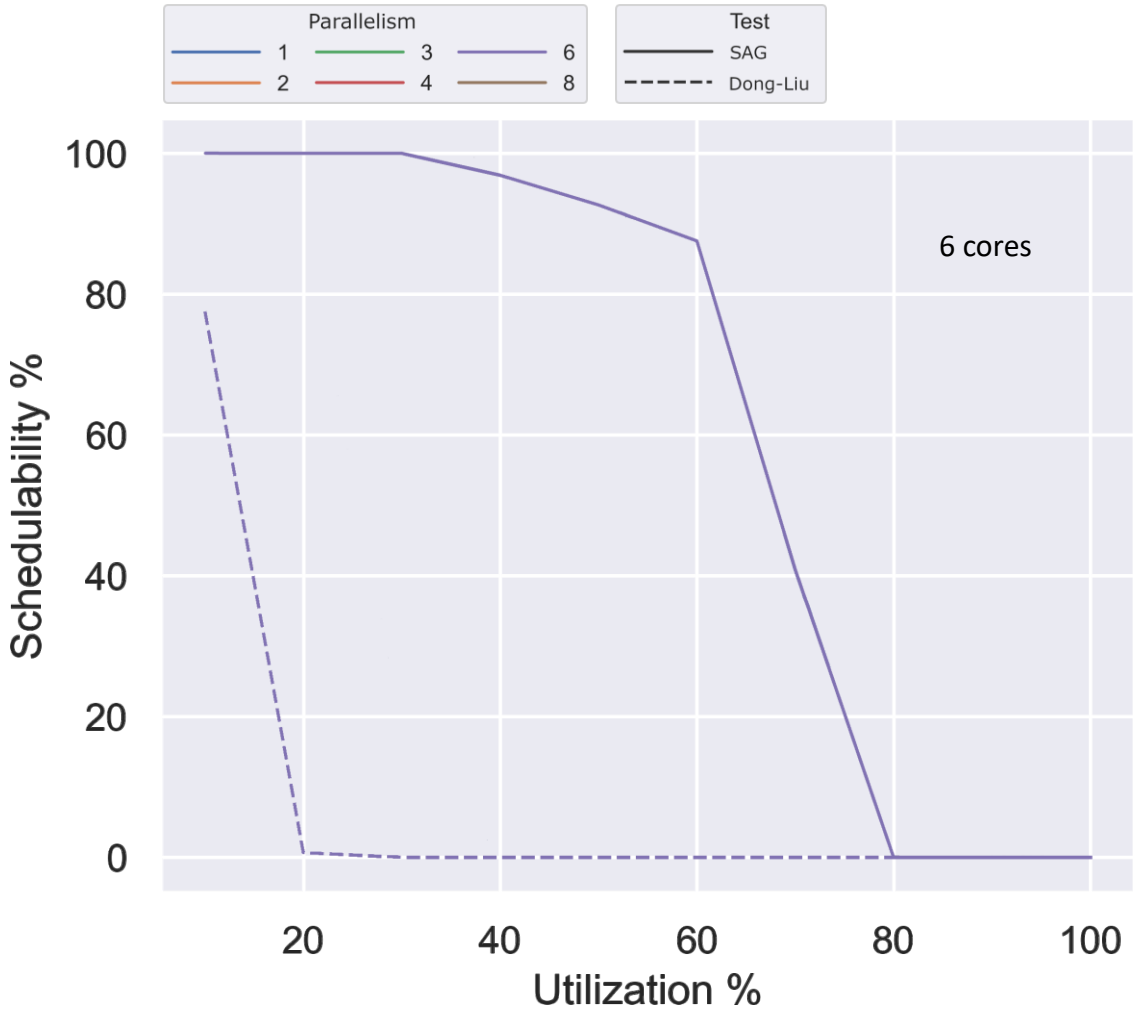
- System cores: 8
- System tasks: 20 rigid tasks
- Execution time variation: 50%
- Implicit deadlines
- Rate monotonic



Our analysis results

Randomly generated task sets

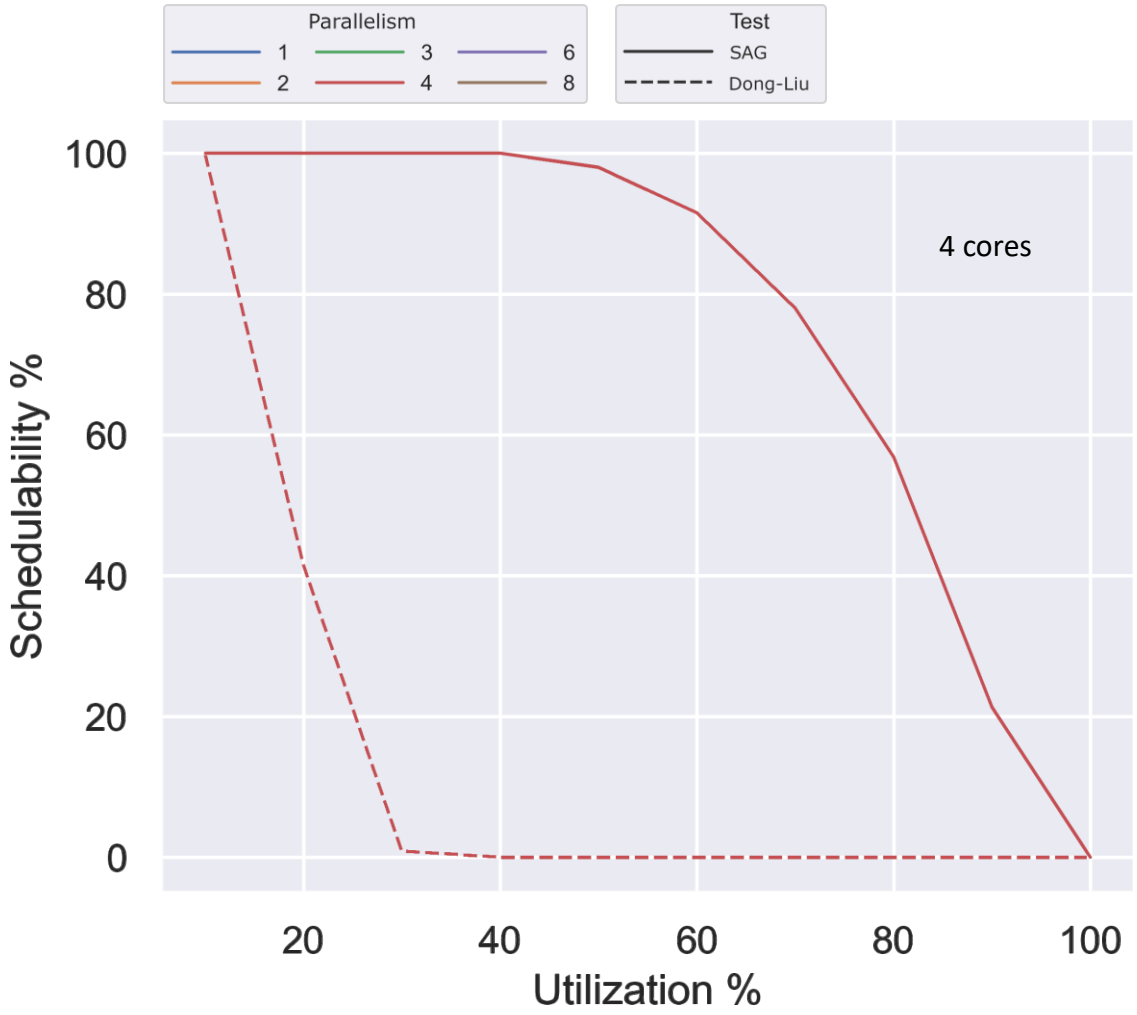
- System cores: 8
- System tasks: 20 rigid tasks
- Execution time variation: 50%
- Implicit deadlines
- Rate monotonic



Our analysis results

Randomly generated task sets

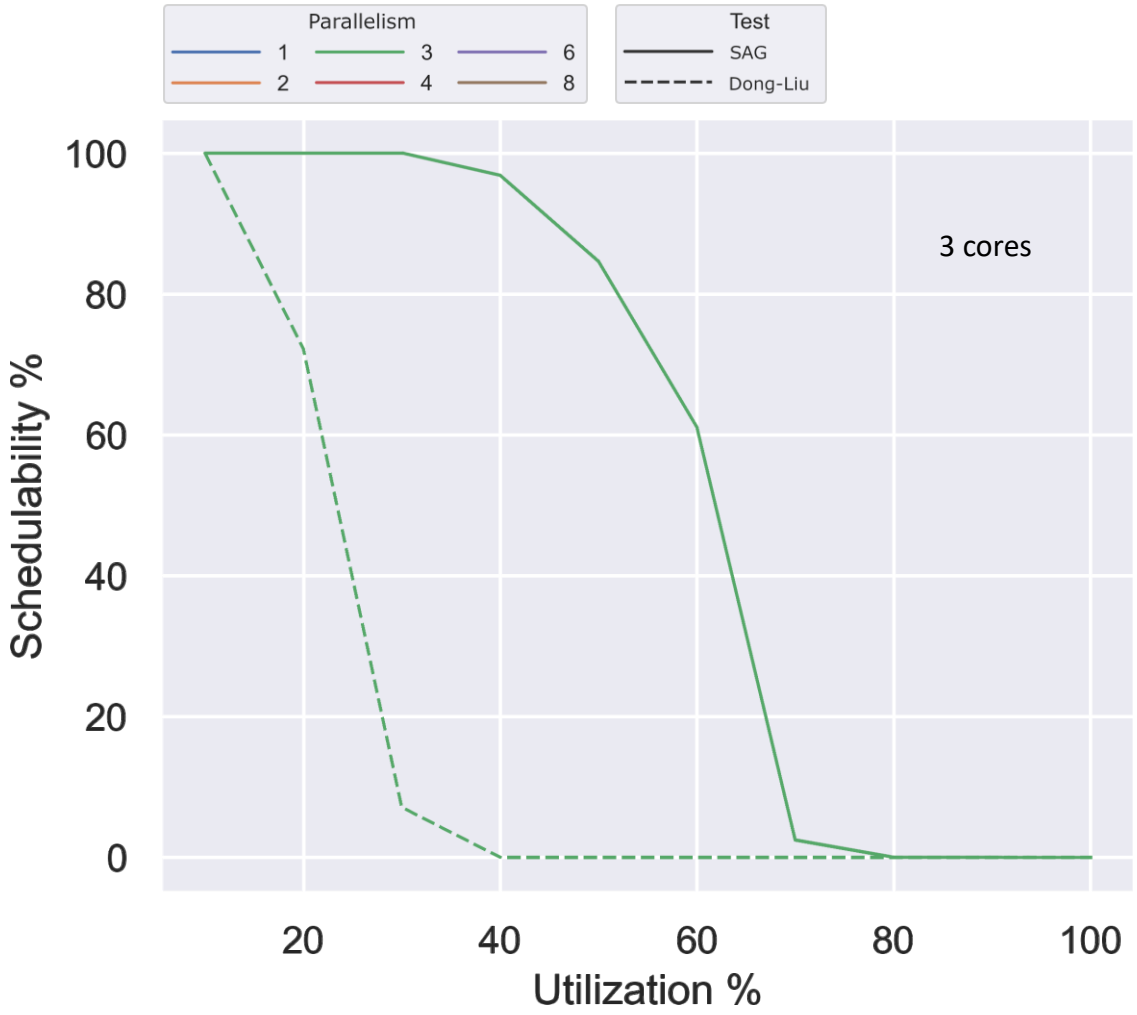
- System cores: 8
- System tasks: 20 rigid tasks
- Execution time variation: 50%
- Implicit deadlines
- Rate monotonic



Our analysis results

Randomly generated task sets

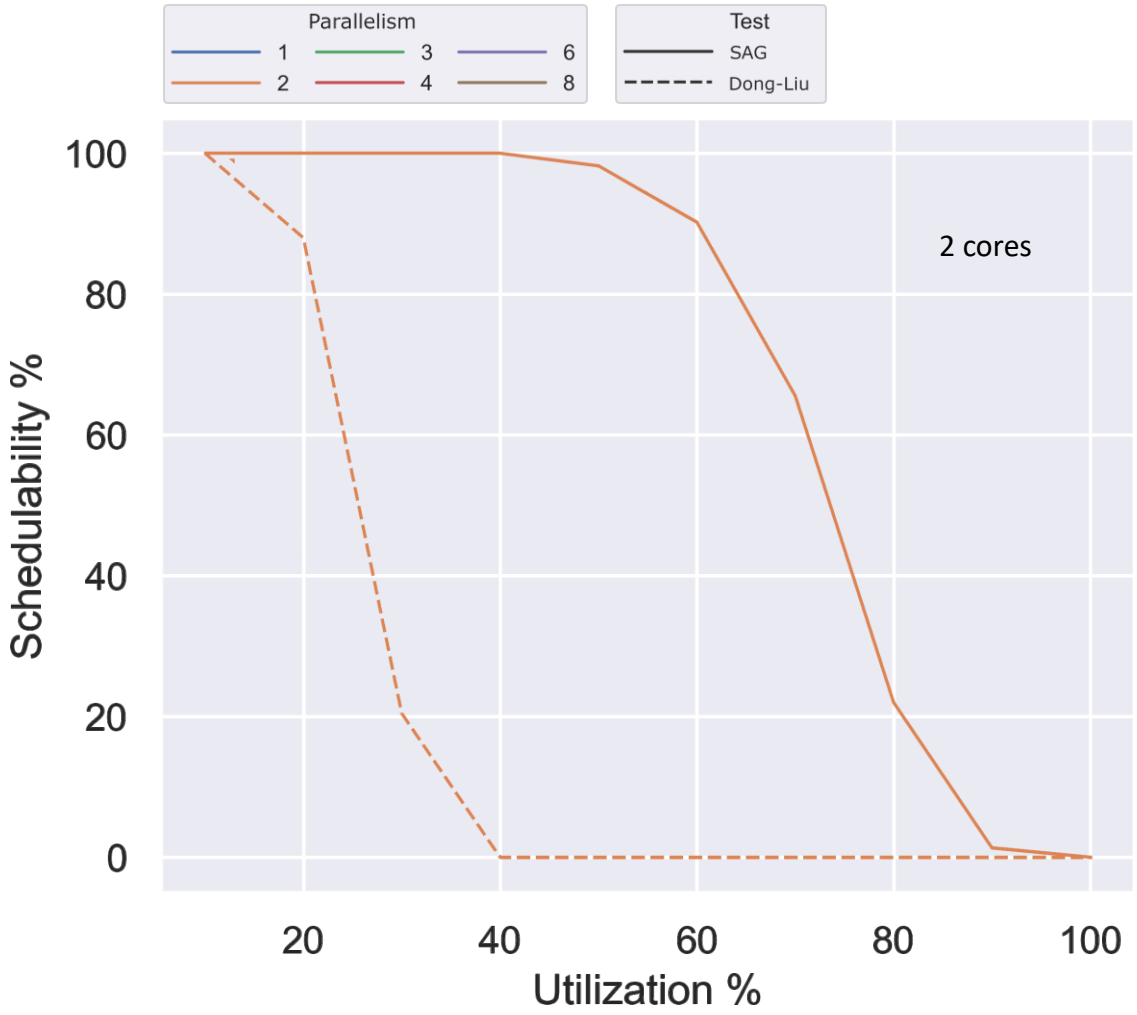
- System cores: 8
- System tasks: 20 rigid tasks
- Execution time variation: 50%
- Implicit deadlines
- Rate monotonic



Our analysis results

Randomly generated task sets

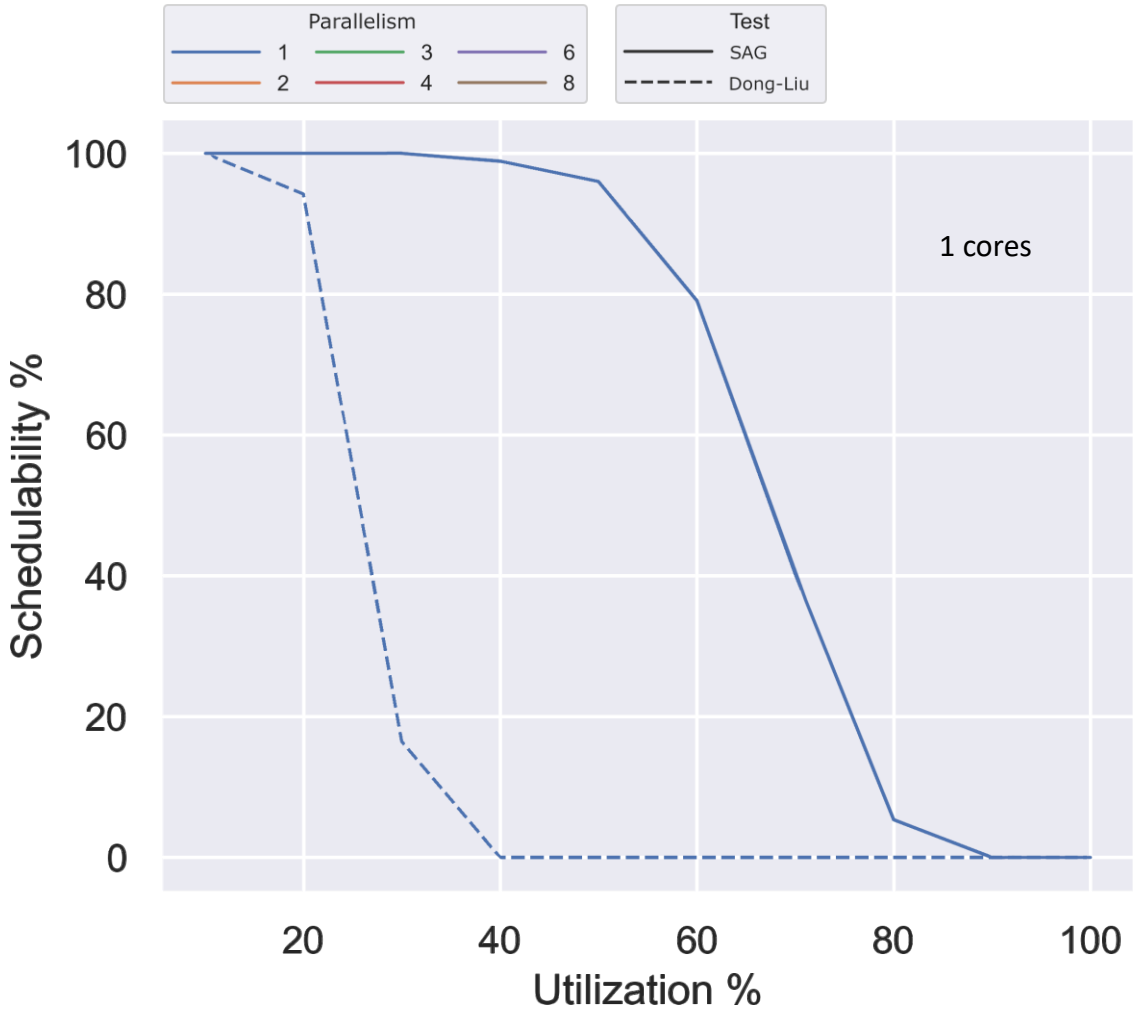
- System cores: 8
- System tasks: 20 rigid tasks
- Execution time variation: 50%
- Implicit deadlines
- Rate monotonic



Our analysis results

Randomly generated task sets

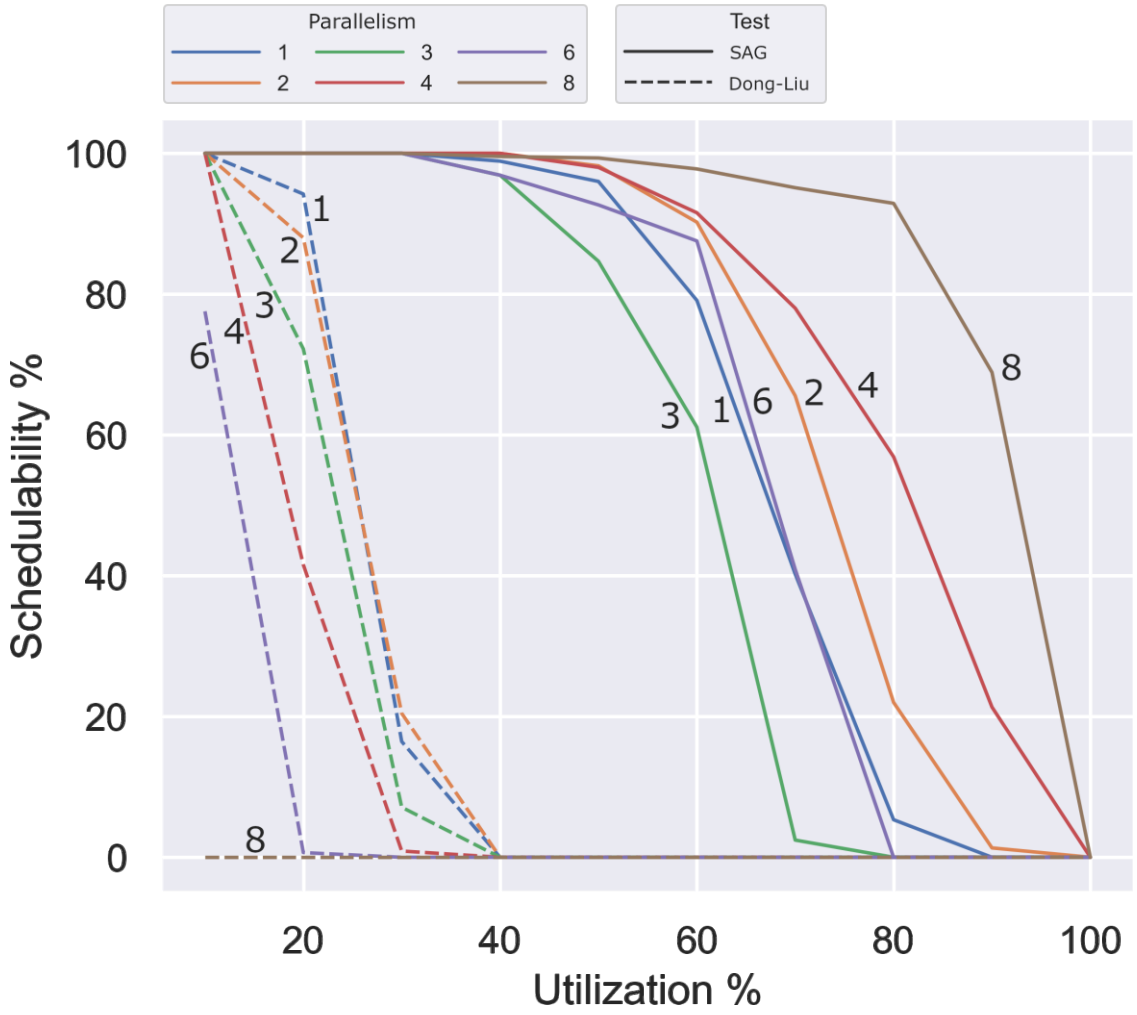
- System cores: 8
- System tasks: 20 rigid tasks
- Execution time variation: 50%
- Implicit deadlines
- Rate monotonic



Our analysis results

Randomly generated task sets

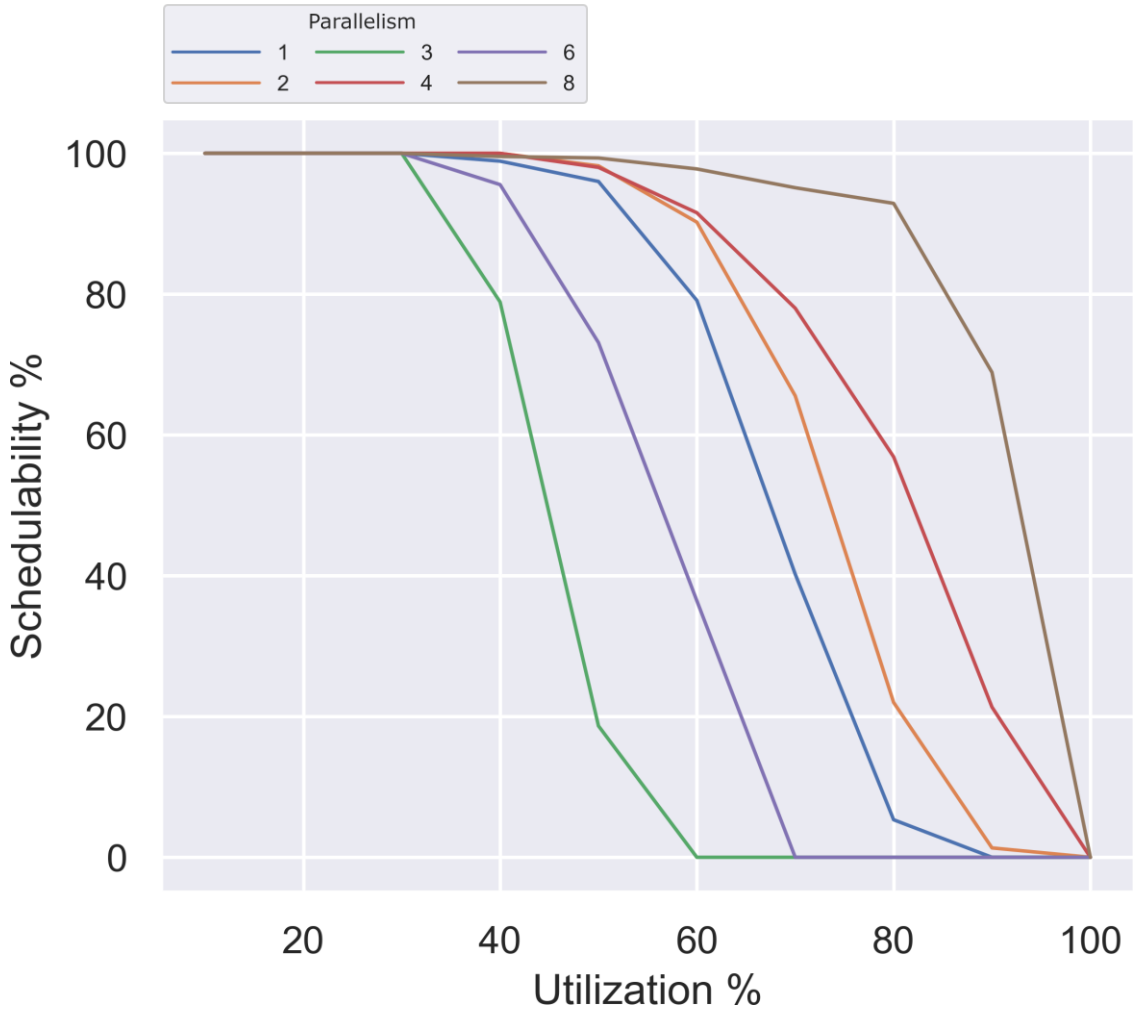
- System cores: 8
- System tasks: 20 rigid tasks
- Execution time variation: 50%
- Implicit deadlines
- Rate monotonic



Our analysis results

Randomly generated task sets

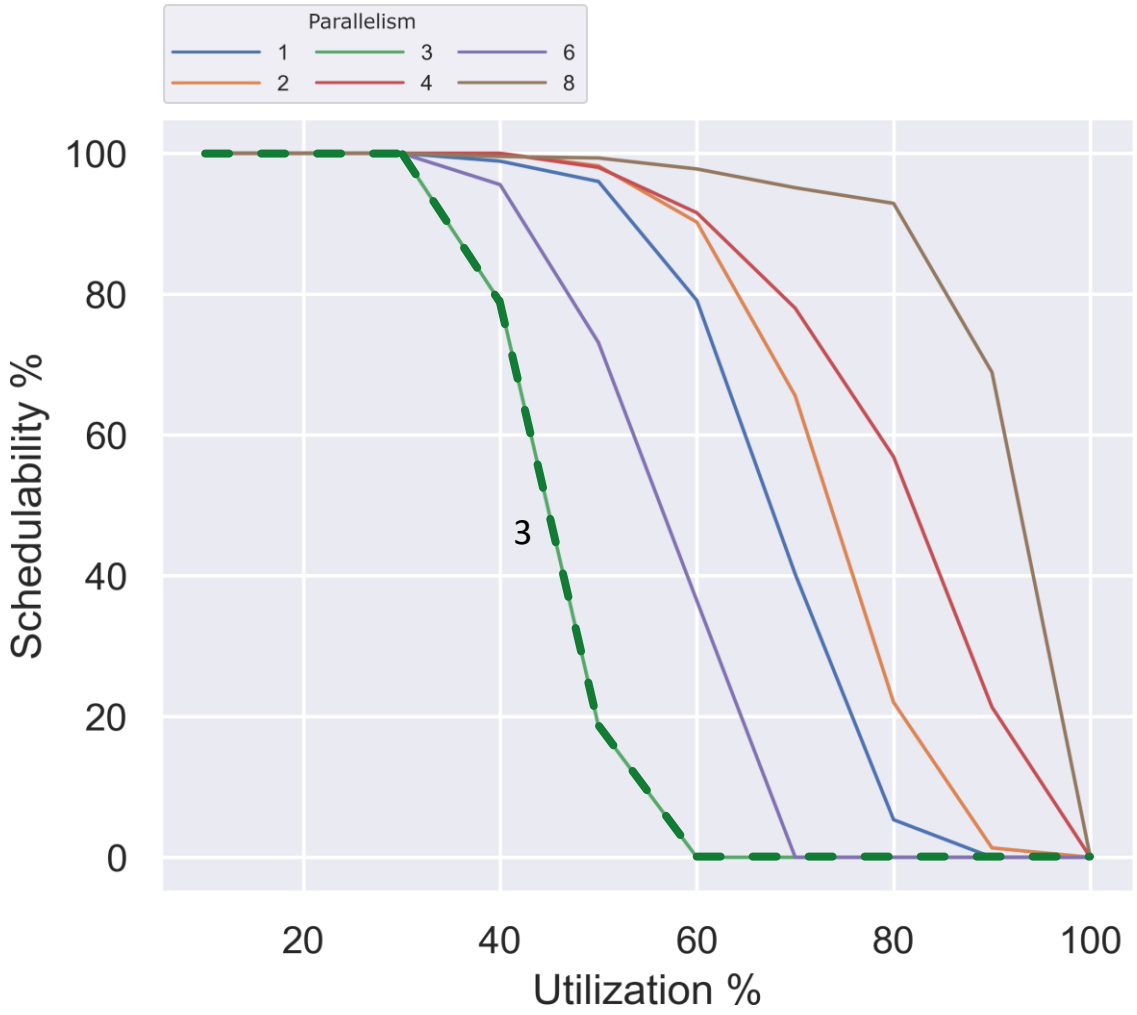
- System cores: 8
- System tasks: 20 moldable tasks
- Execution time variation: 50%
- Implicit deadlines
- Rate monotonic



Our analysis results

Randomly generated task sets

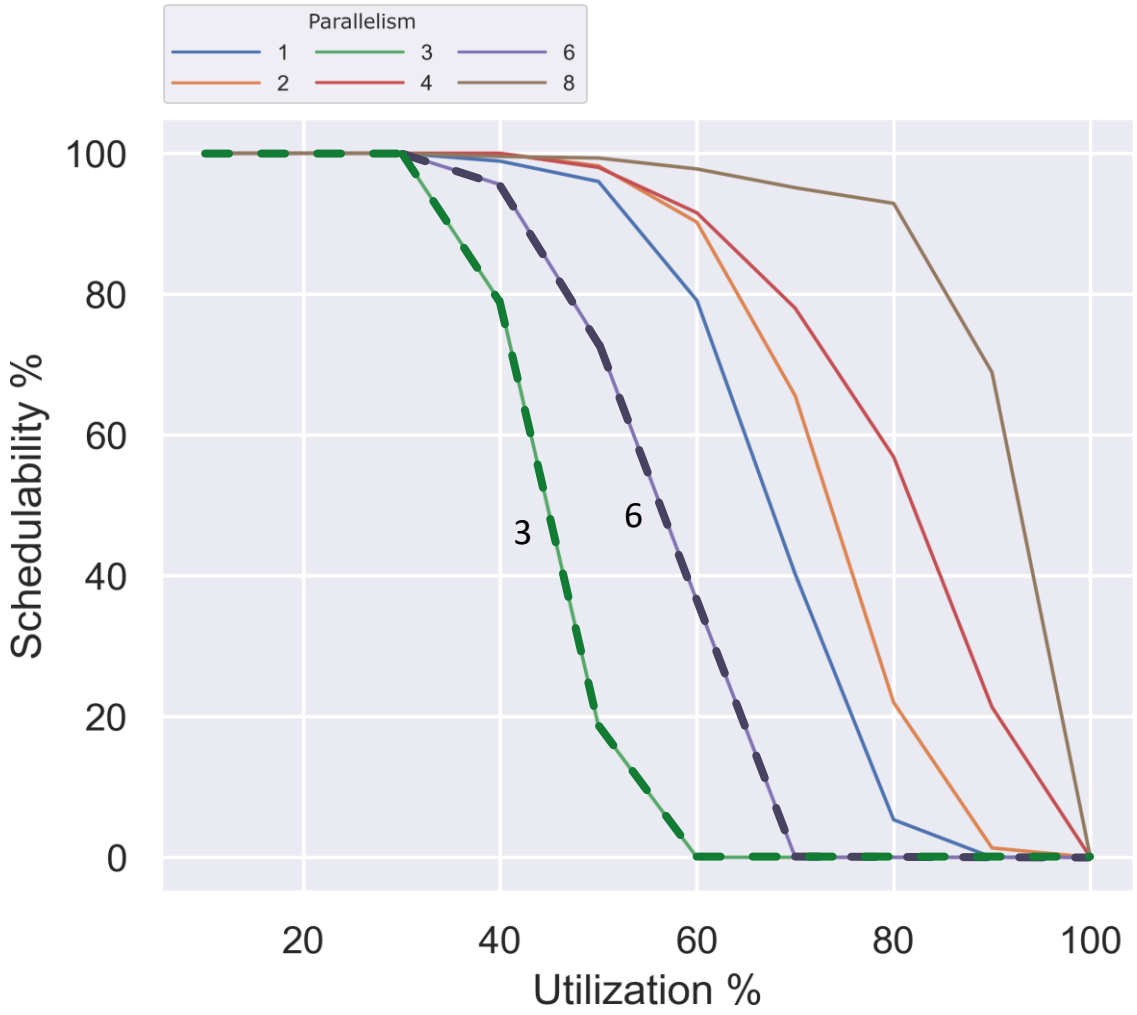
- System cores: 8
- System tasks: 20 moldable tasks
- Execution time variation: 50%
- Implicit deadlines
- Rate monotonic



Our analysis results

Randomly generated task sets

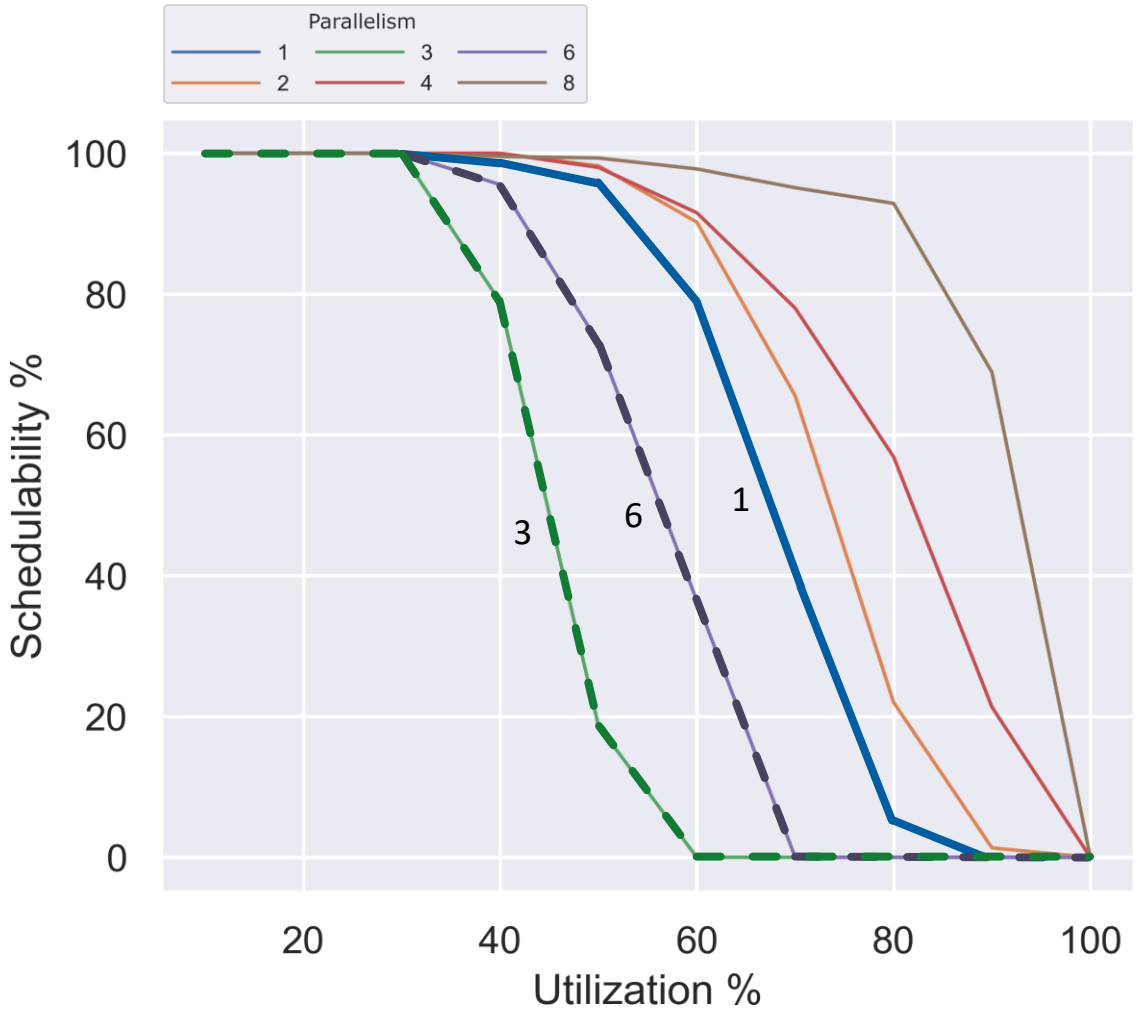
- System cores: 8
- System tasks: 20 moldable tasks
- Execution time variation: 50%
- Implicit deadlines
- Rate monotonic



Our analysis results

Randomly generated task sets

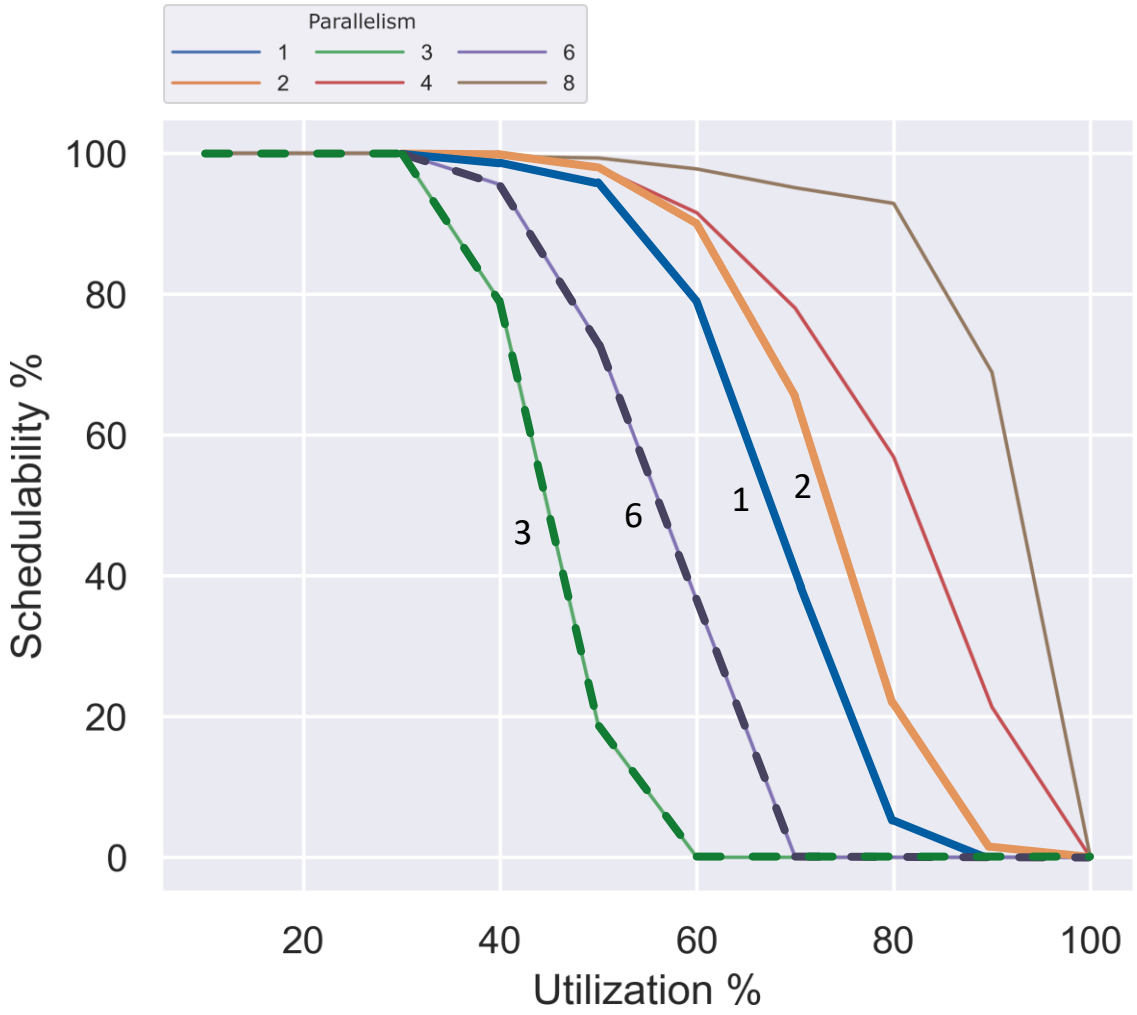
- System cores: 8
- System tasks: 20 moldable tasks
- Execution time variation: 50%
- Implicit deadlines
- Rate monotonic



Our analysis results

Randomly generated task sets

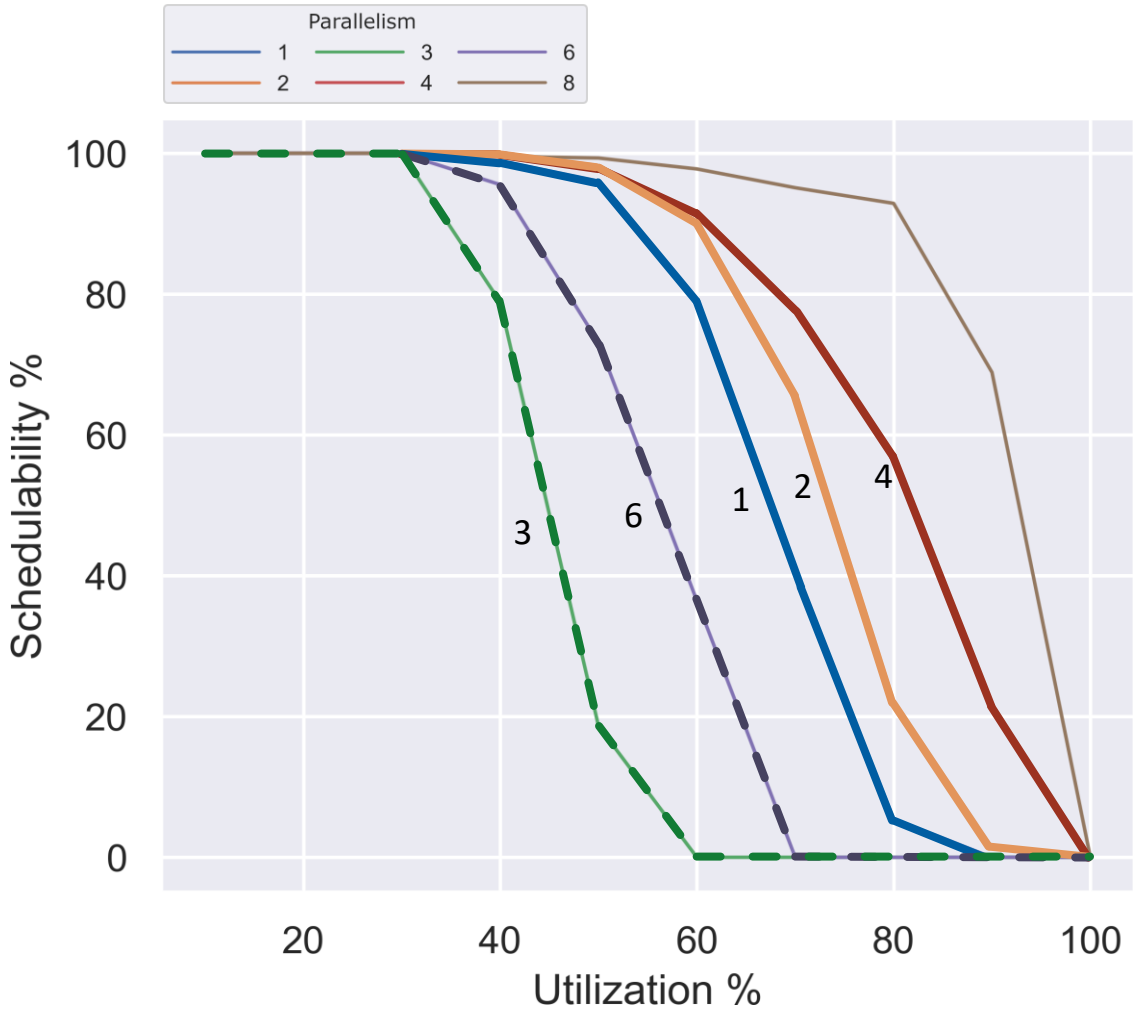
- System cores: 8
- System tasks: 20 moldable tasks
- Execution time variation: 50%
- Implicit deadlines
- Rate monotonic



Our analysis results

Randomly generated task sets

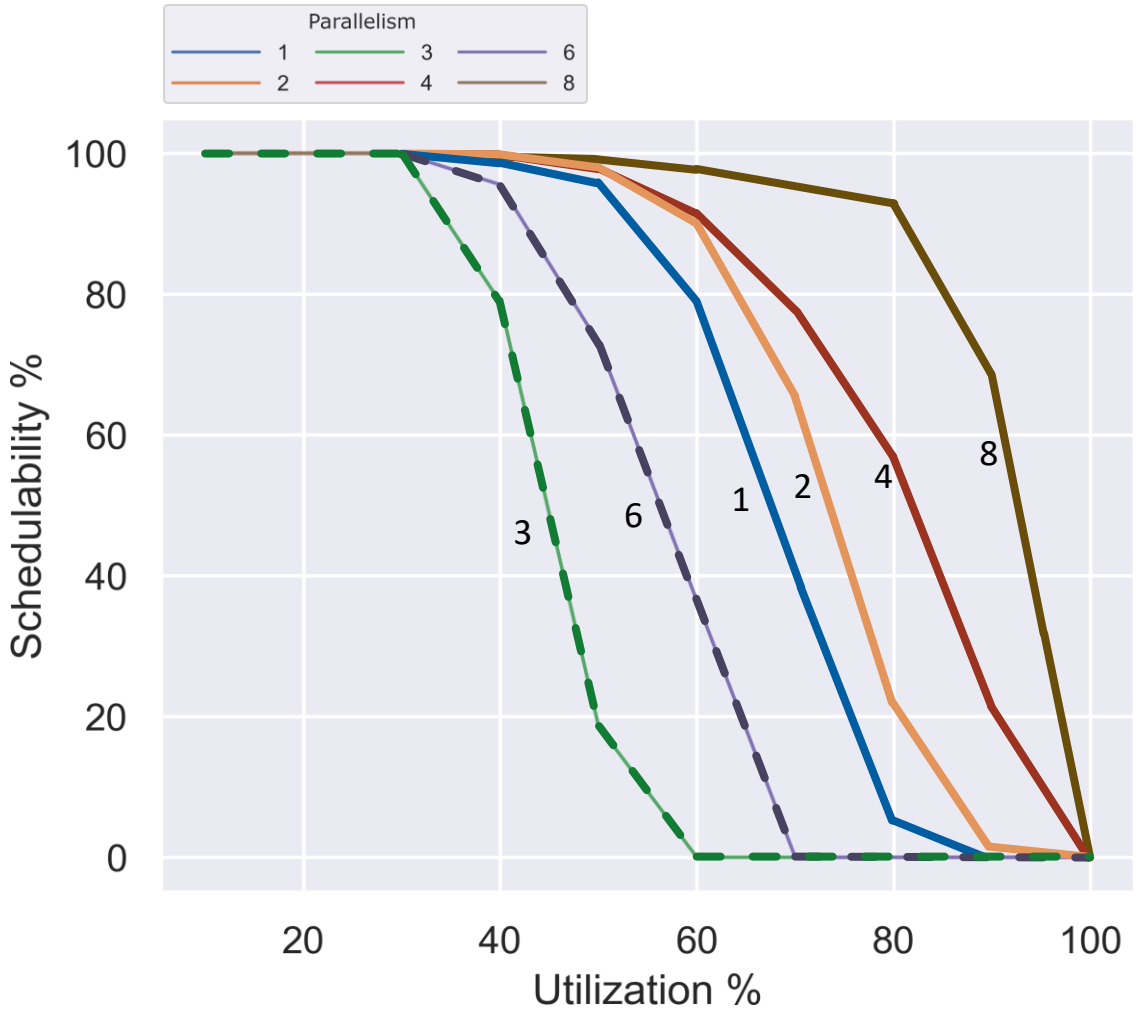
- System cores: 8
- System tasks: 20 moldable tasks
- Execution time variation: 50%
- Implicit deadlines
- Rate monotonic



Our analysis results

Randomly generated task sets

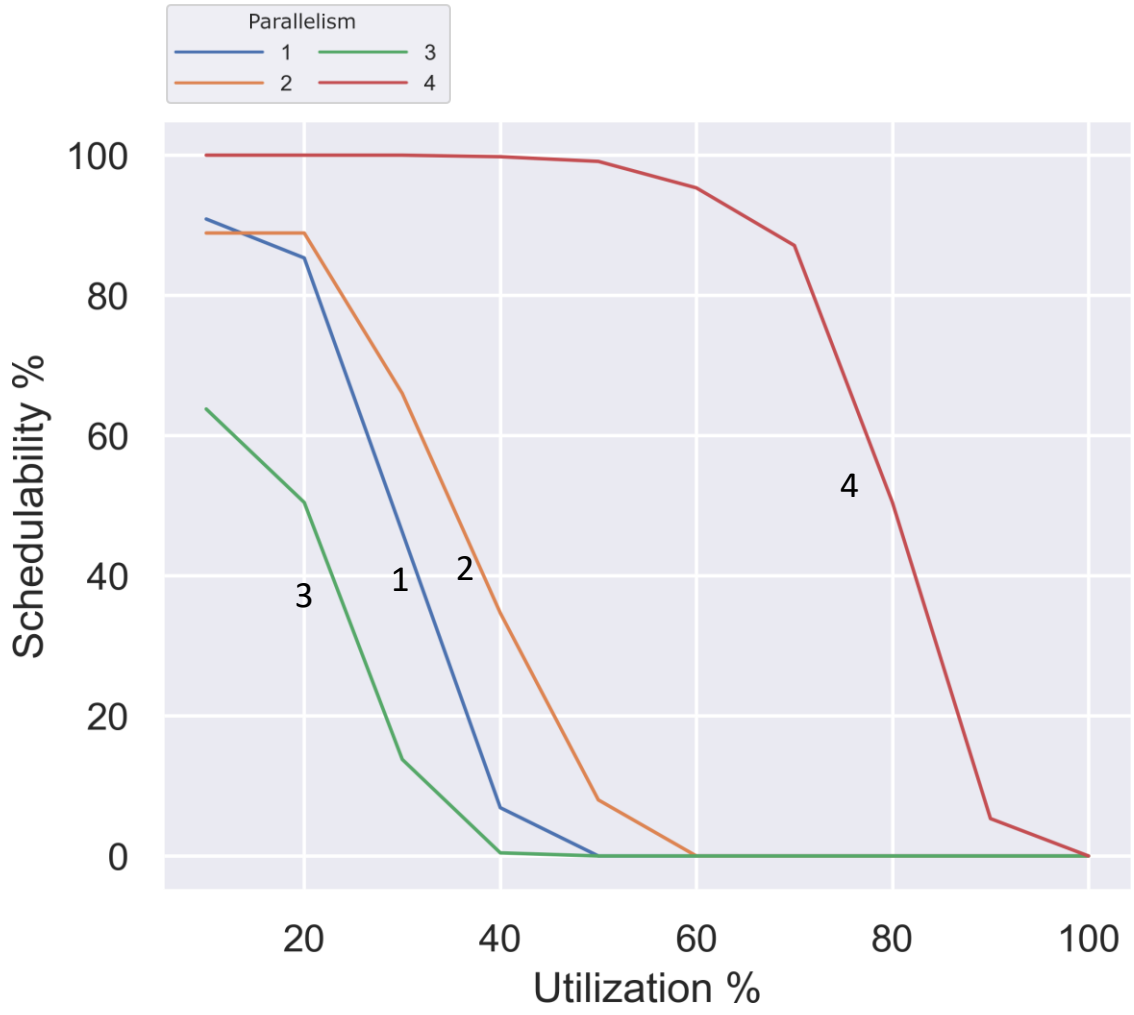
- System cores: 8
- System tasks: 20 moldable tasks
- Execution time variation: 50%
- Implicit deadlines
- Rate monotonic



Our analysis results

Randomly generated task sets

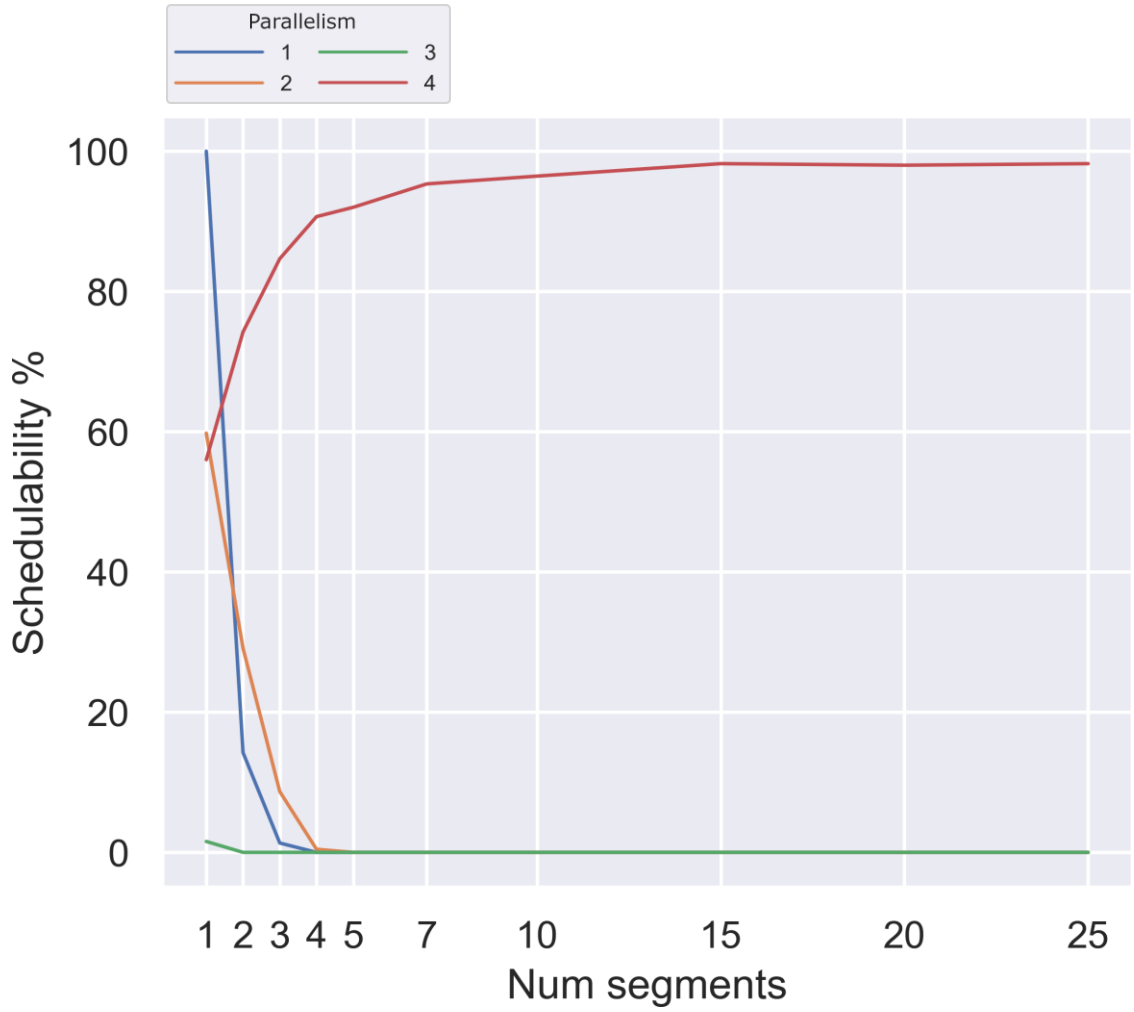
- System cores: 4
- System tasks:
  - 4 moldable tasks
  - 5 segments per task
- Execution time variation: 50%
- Implicit deadlines
- Rate monotonic



Our analysis results

Randomly generated task sets

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



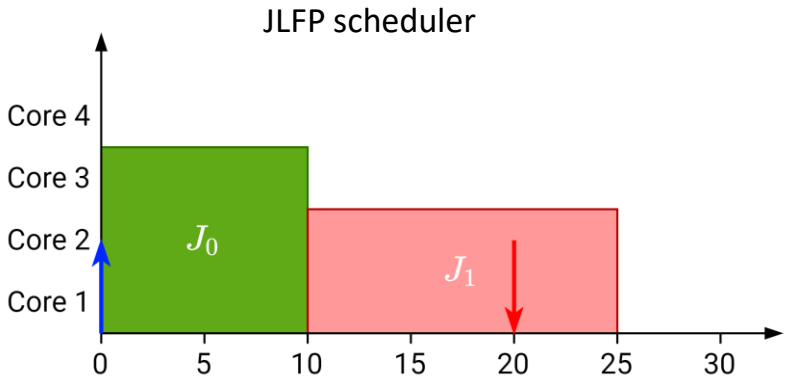
Agenda

- ✔ Gang schedulability analysis
- New scheduling policy

Core assignment

Core assignment

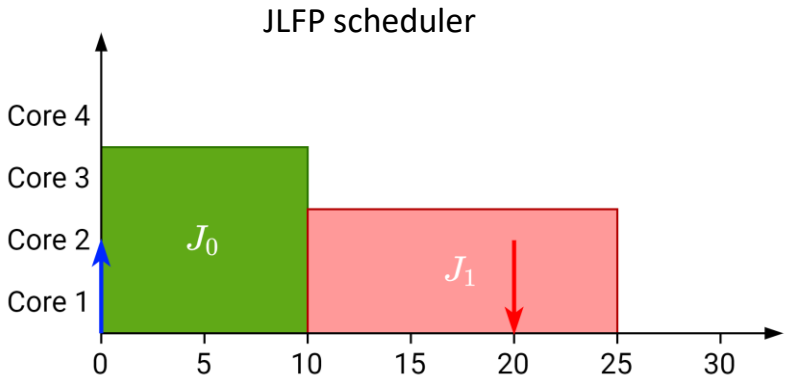
- JLFP assigns the maximum possible cores to a job



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

Core assignment

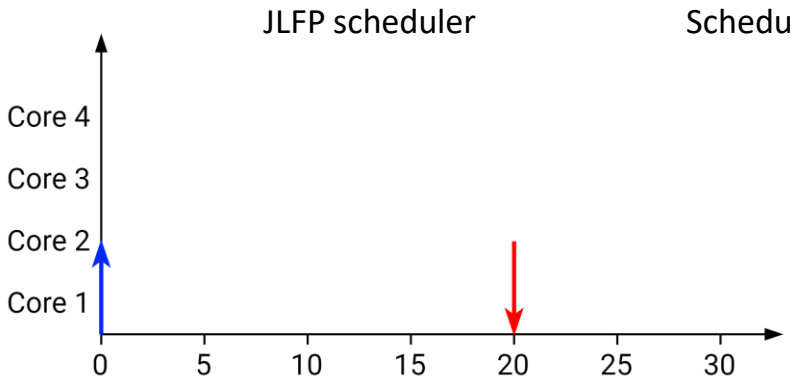
- JLFP assigns the maximum possible cores to a job
- Assign minimum number of cores that meet deadline



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

Core assignment

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



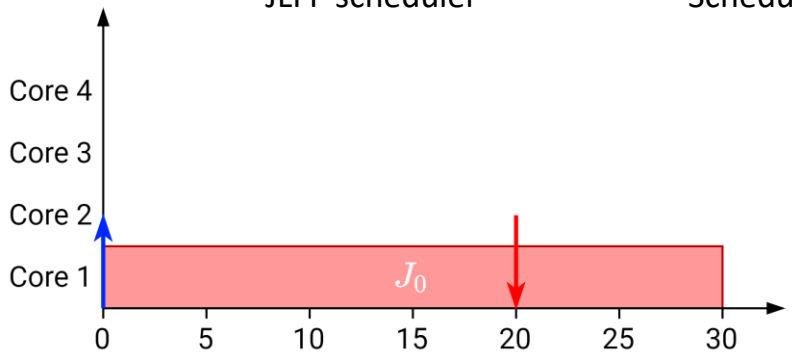
	Priority	Min cores	Max cores	Deadline	Execution time
J_0	High	1	3	20	30, 15, 10
J_1	Low	2	2	20	15

Core assignment

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

JLFP scheduler

Scheduling: J_0

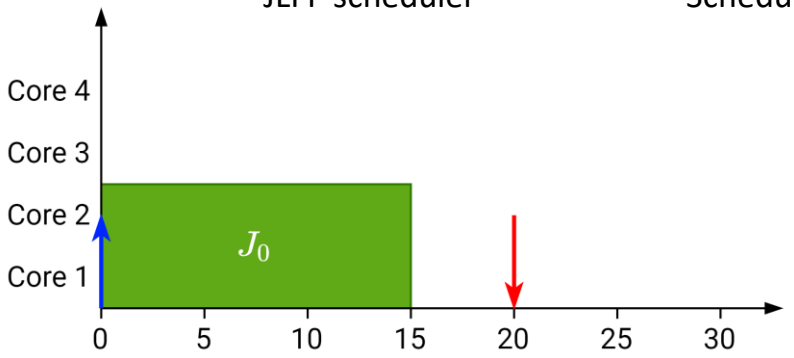


	Priority	Min cores	Max cores	Deadline	Execution time
J_0	High	1	3	20	30, 15, 10
J_1	Low	2	2	20	15

Core assignment

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

JLFP scheduler

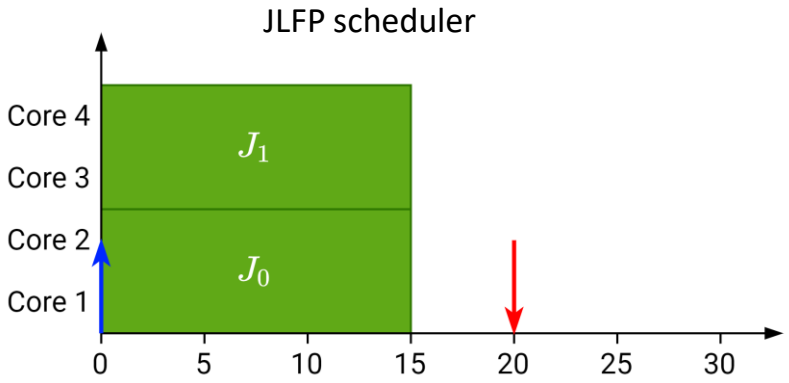


Scheduling: J_1

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

Core assignment

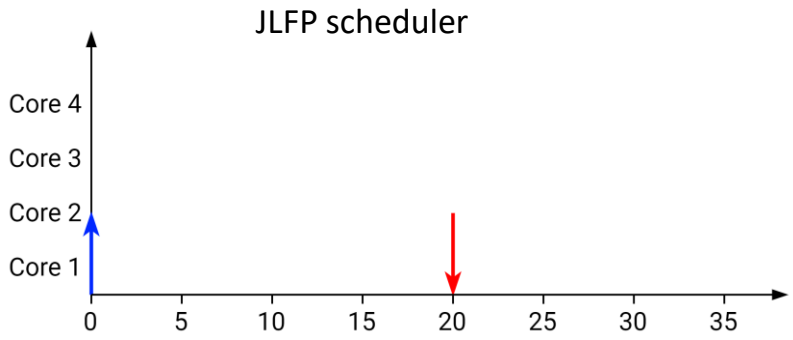
- JLFP assigns the maximum possible cores to a job
- Assign minimum number of cores that meet deadline



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

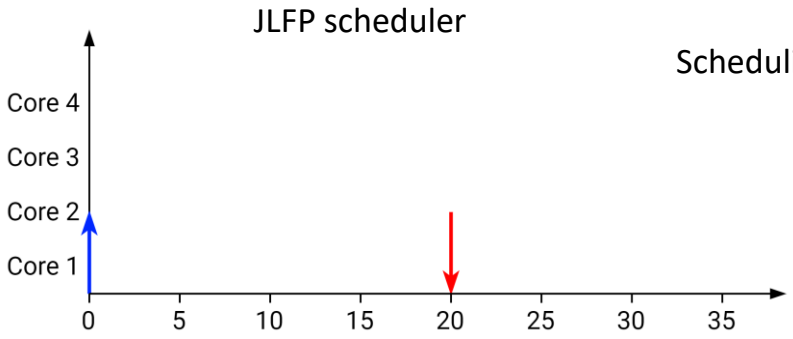
JLFP limitations with moldable gang

JLFP limitations with moldable gang



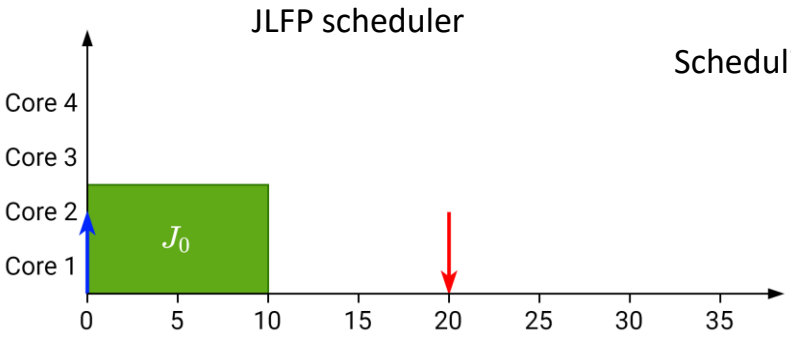
	Priority	Cores	Deadline	Execution time
J_0	High	2	∞	10
J_1	Mid-high	3	20	5
J_2	Mid-low	1	∞	20
J_3	Low	1	∞	20

JLFP limitations with moldable gang



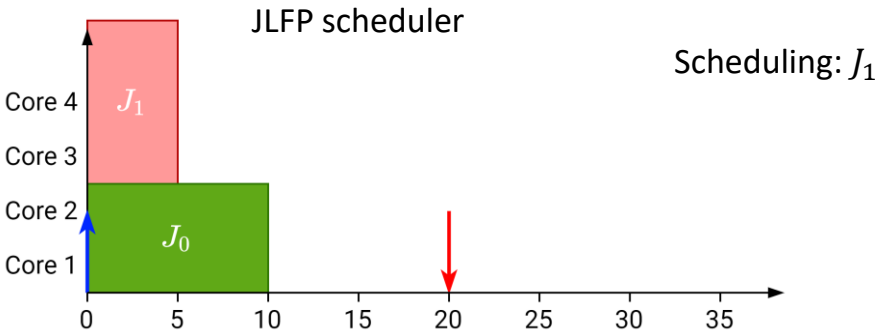
	Priority	Cores	Deadline	Execution time
J_0	High	2	∞	10
J_1	Mid-high	3	20	5
J_2	Mid-low	1	∞	20
J_3	Low	1	∞	20

JLFP limitations with moldable gang



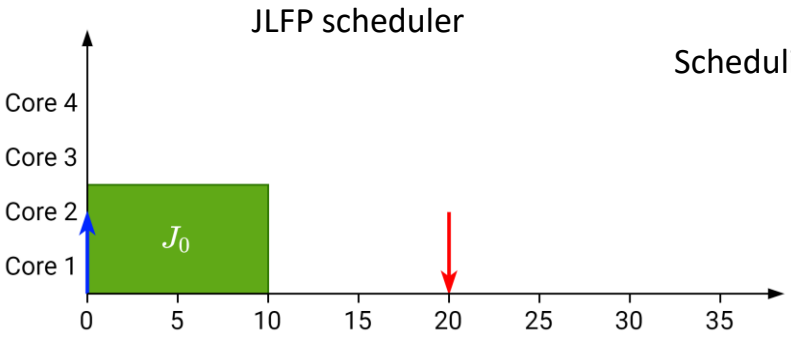
	Priority	Cores	Deadline	Execution time
J_0	High	2	∞	10
J_1	Mid-high	3	20	5
J_2	Mid-low	1	∞	20
J_3	Low	1	∞	20

JLFP limitations with moldable gang



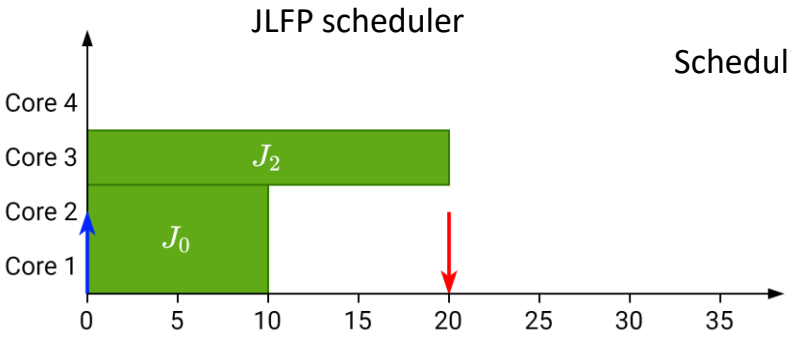
	Priority	Cores	Deadline	Execution time
J_0	High	2	∞	10
J_1	Mid-high	3	20	5
J_2	Mid-low	1	∞	20
J_3	Low	1	∞	20

JLFP limitations with moldable gang



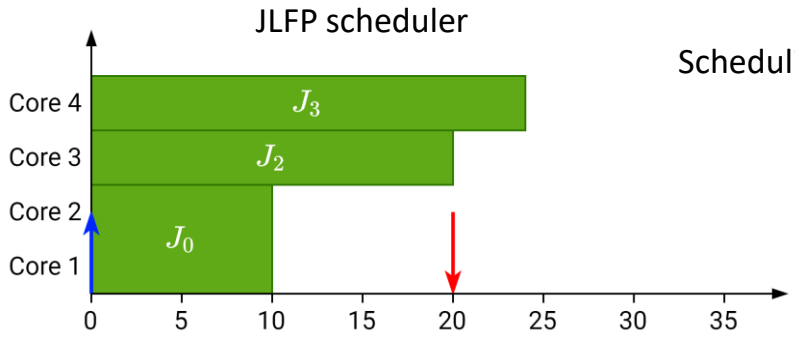
	Priority	Cores	Deadline	Execution time
J_0	High	2	∞	10
J_1	Mid-high	3	20	5
J_2	Mid-low	1	∞	20
J_3	Low	1	∞	20

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

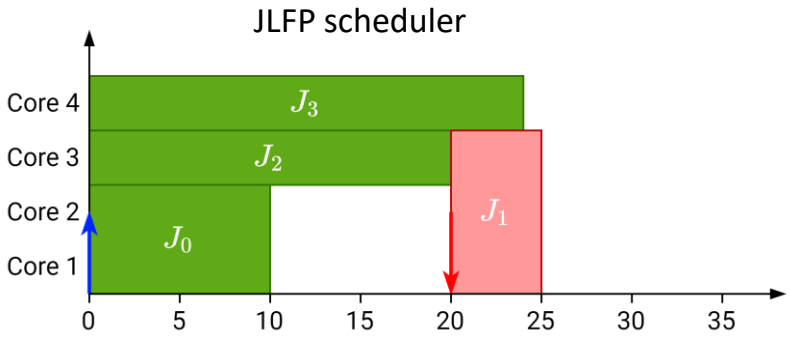
JLFP limitations with moldable gang



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

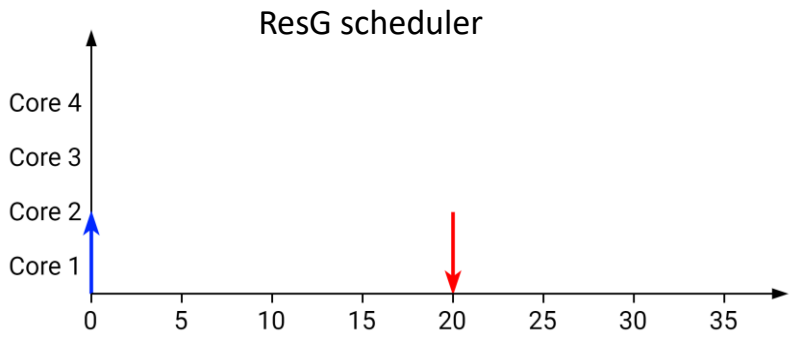
JLFP limitations with moldable gang



	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

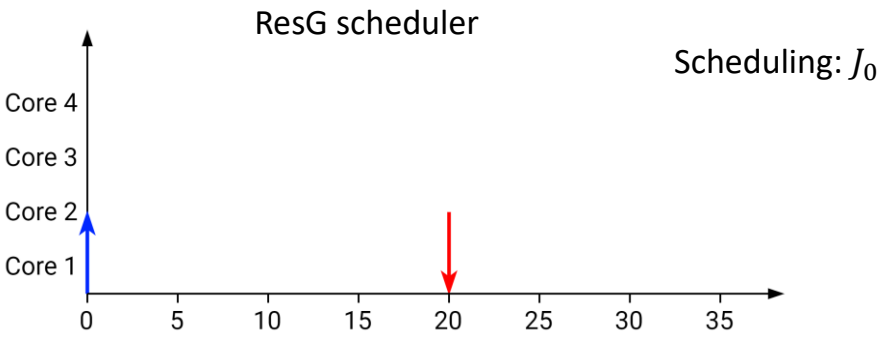
- Reservation-based



	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

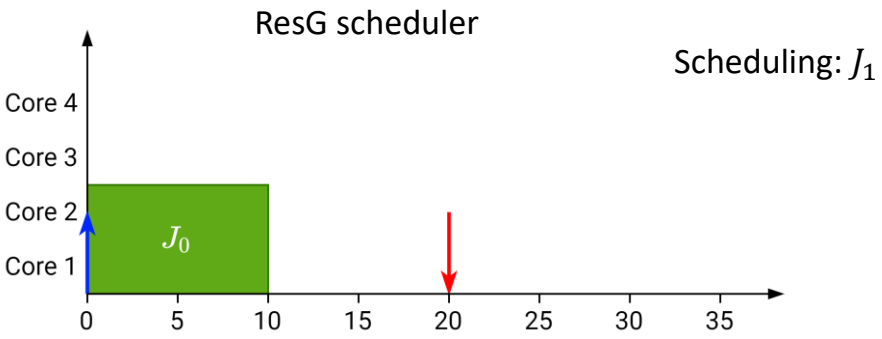
- Reservation-based



	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

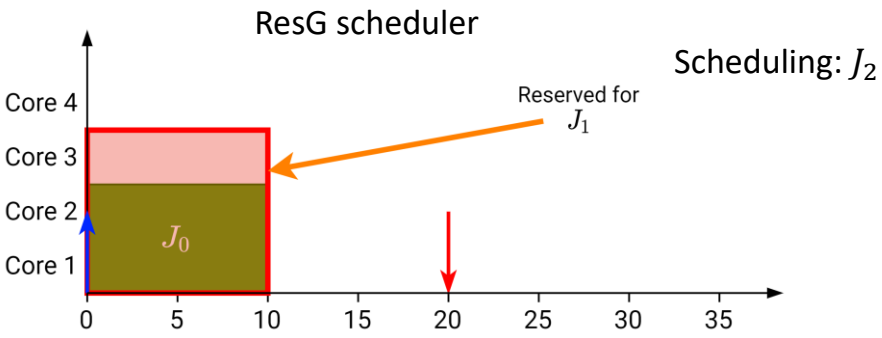
- Reservation-based



	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

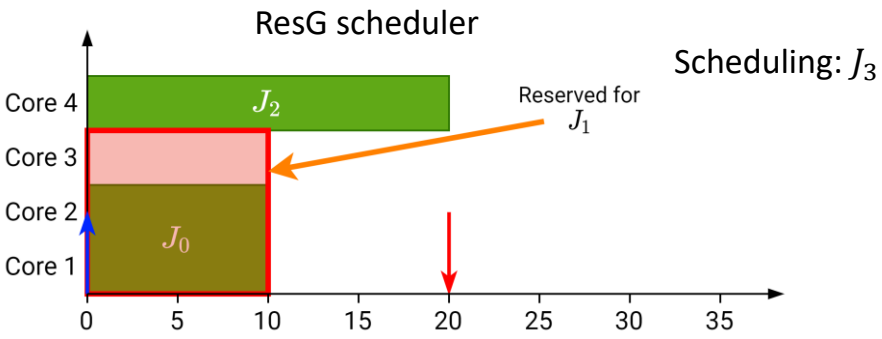
- Reservation-based
- Reserve cores of higher-priority tasks and distribute the remaining ones among lower priority tasks



	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

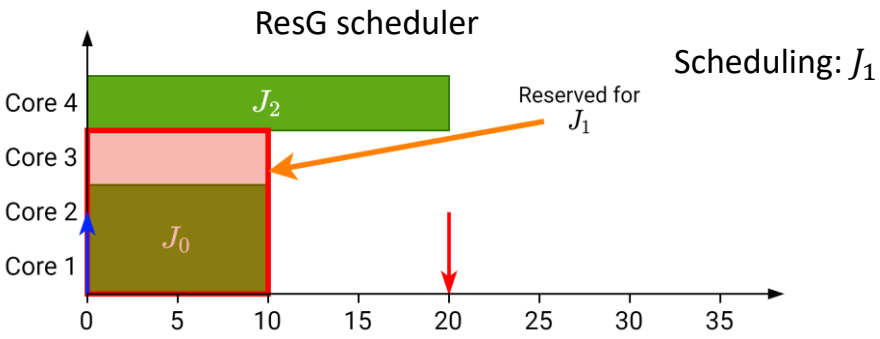
- Reservation-based
- Reserve cores of higher-priority tasks and distribute the remaining ones among lower priority tasks



	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

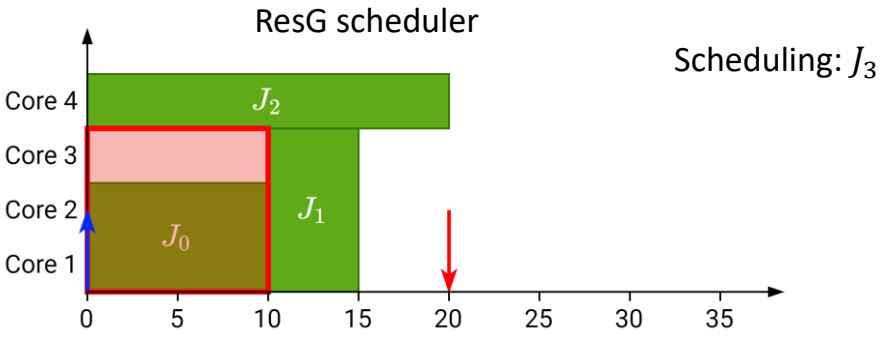
- Reservation-based
- Reserve cores of higher-priority tasks and distribute the remaining ones among lower priority tasks



	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

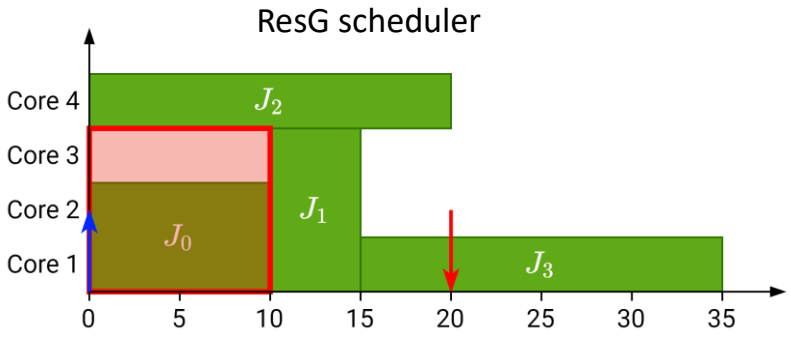
- Reservation-based
- Reserve cores of higher-priority tasks and distribute the remaining ones among lower priority tasks



	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

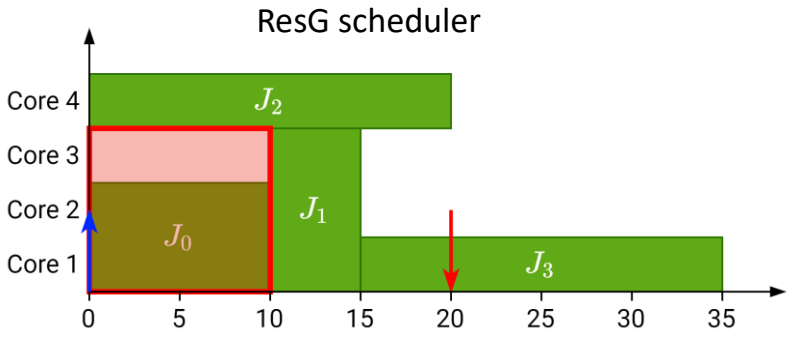
- Reservation-based
- Reserve cores of higher-priority tasks and distribute the remaining ones among lower priority tasks



	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

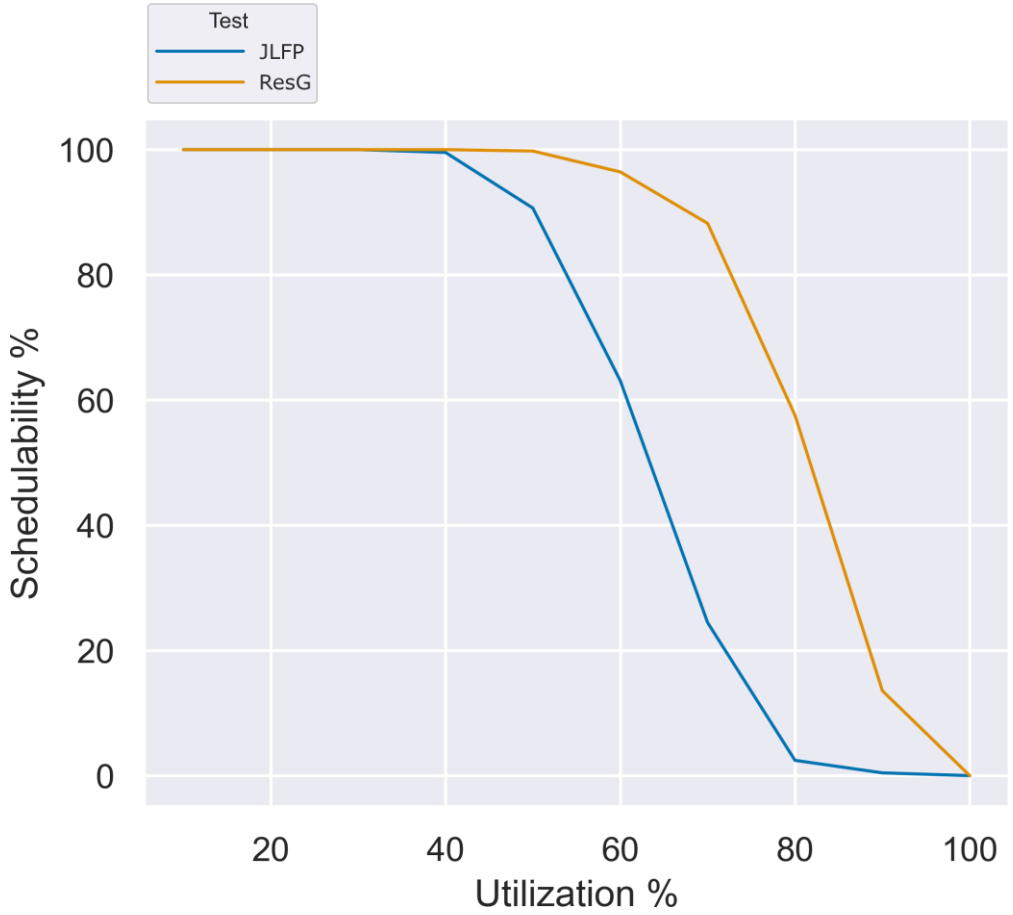
- Evaluated in simulator

Results JLFP vs ResG in simulator

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

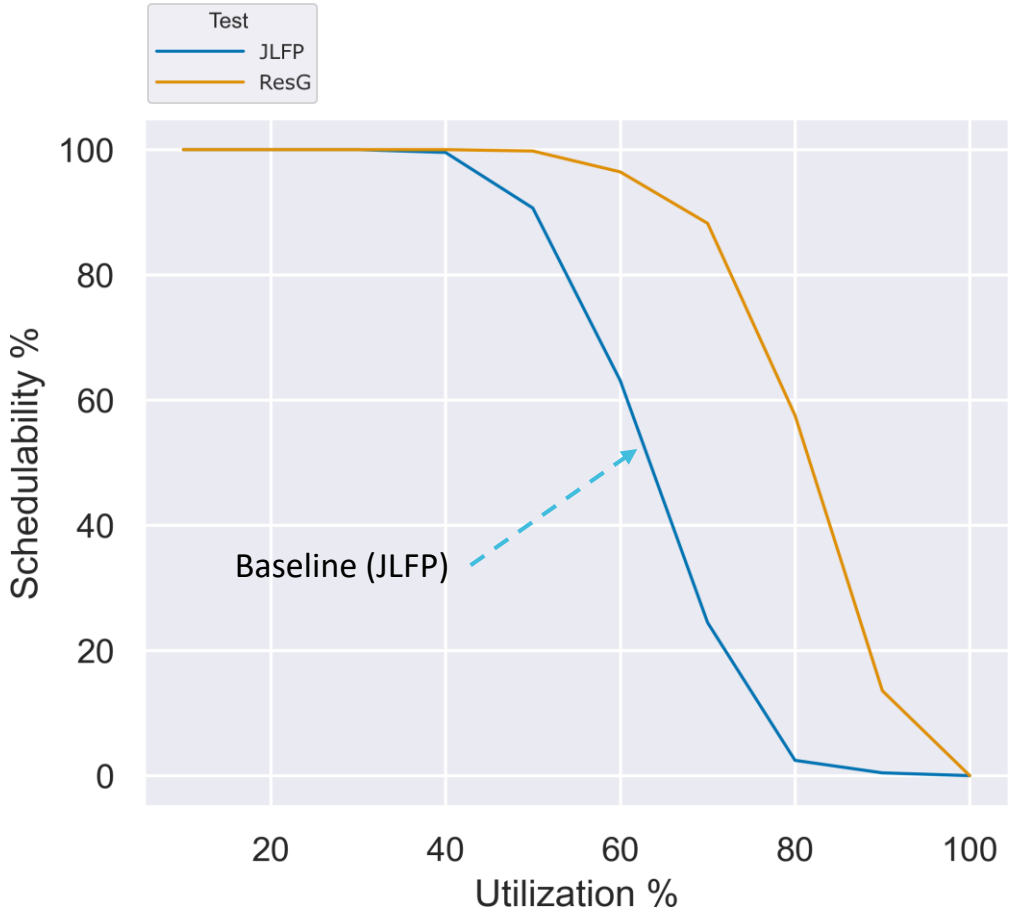
Results JLFP vs ResG in simulator

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



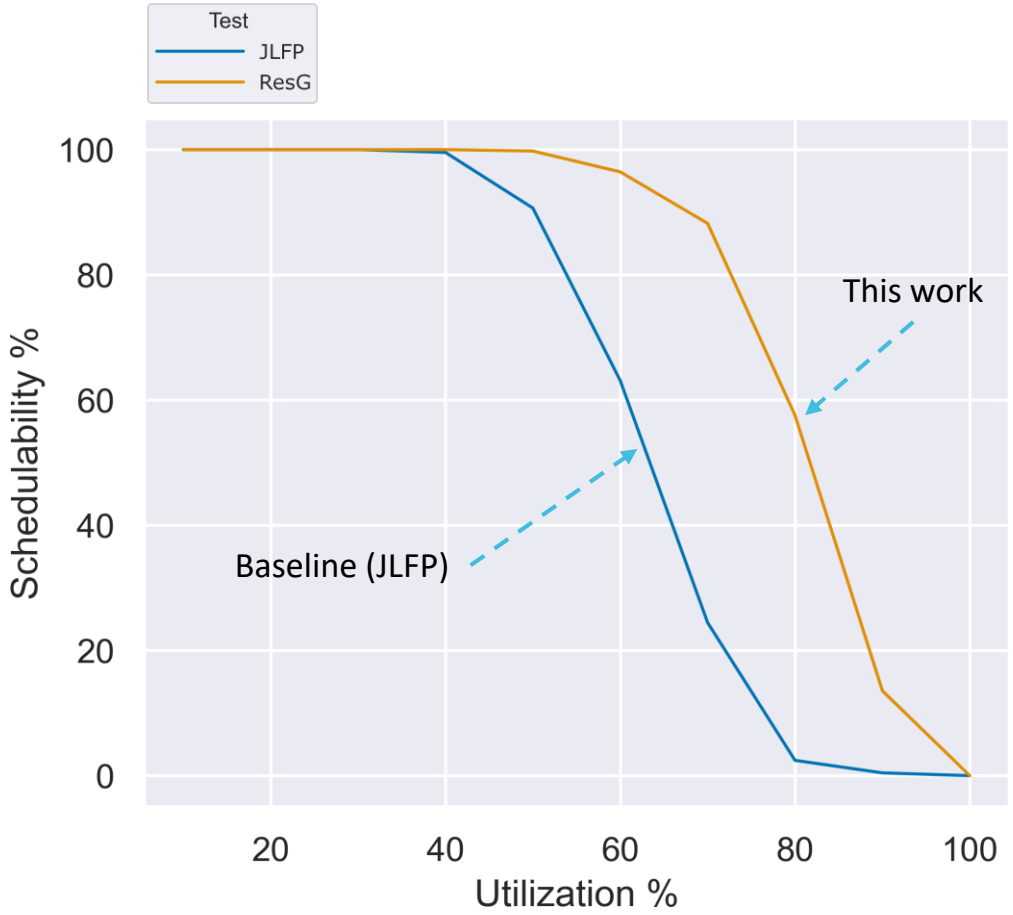
Results JLFP vs ResG in simulator

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



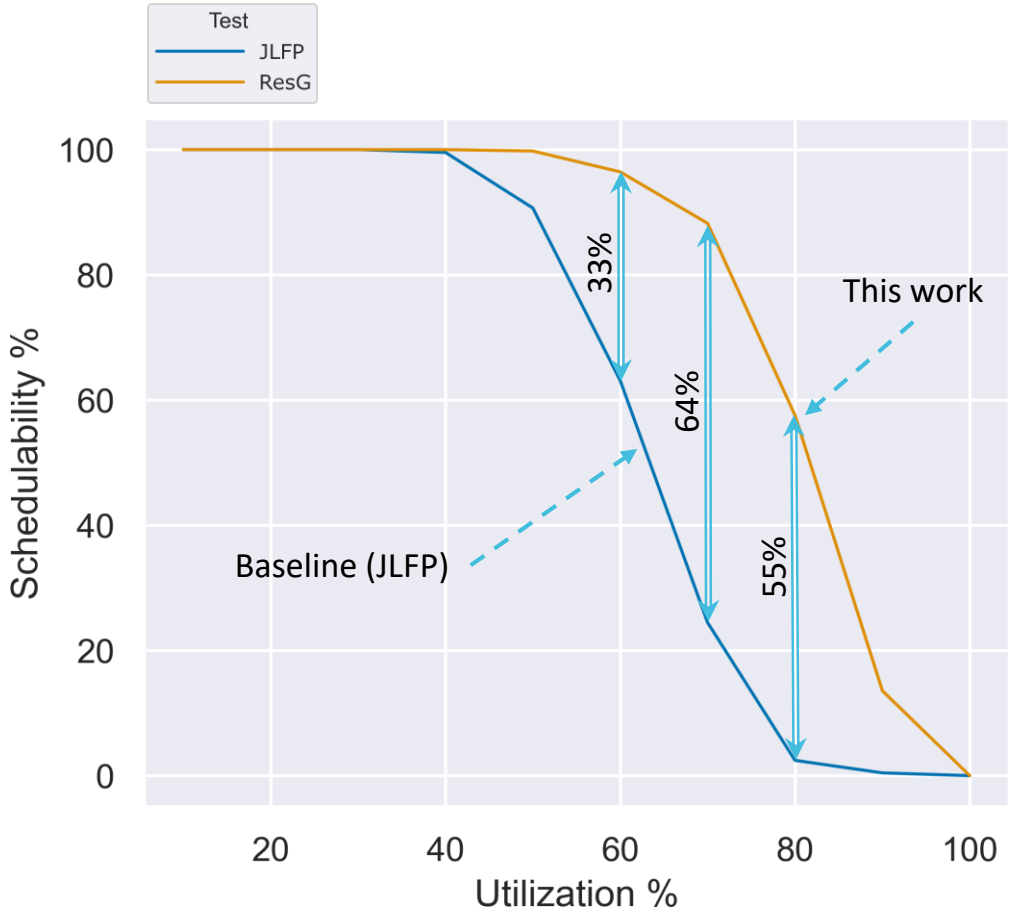
Results JLFP vs ResG in simulator

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



Results JLFP vs ResG in simulator

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



Conclusions

Conclusions

- The maximum parallelism of a gang task has a big role in the schedulability

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

Future work

- Further reduce sources of pessimism with precedence constraints

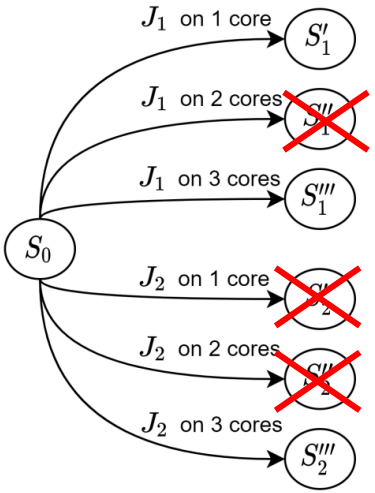
Future work

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

Contributions

Contributions

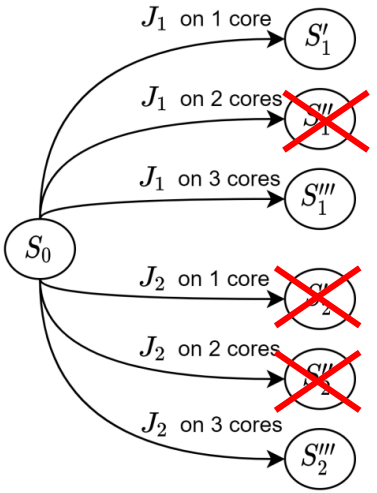
New schedulability analysis



Contributions

New schedulability analysis

Moldable gang tasks evaluation

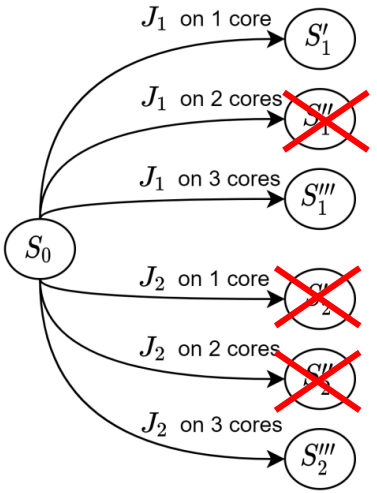


Contributions

New schedulability analysis

Moldable gang tasks evaluation

Paper submitted to RTSS 2020 for the non-preemptive analysis



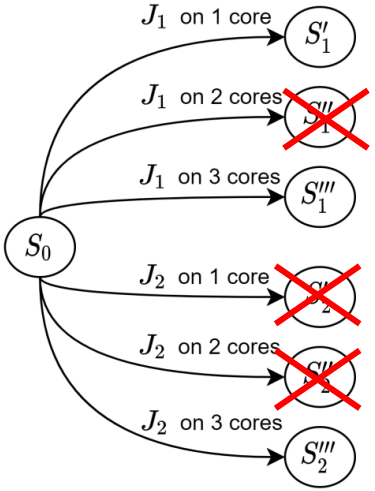
Contributions

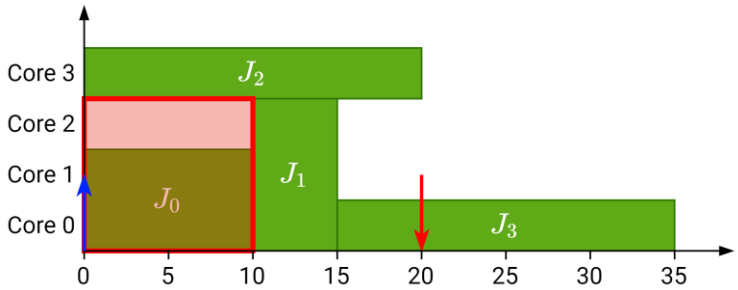
New schedulability analysis

Moldable gang tasks evaluation

New scheduling policy

Paper submitted to RTSS 2020 for the non-preemptive analysis





Contributions

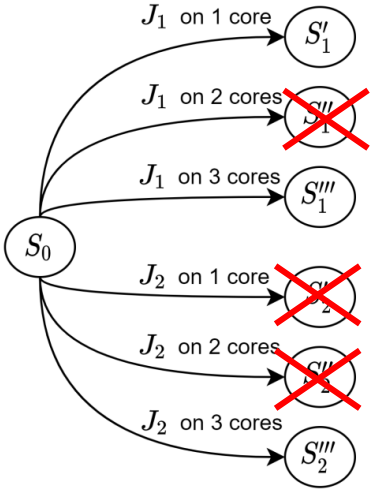
New schedulability analysis

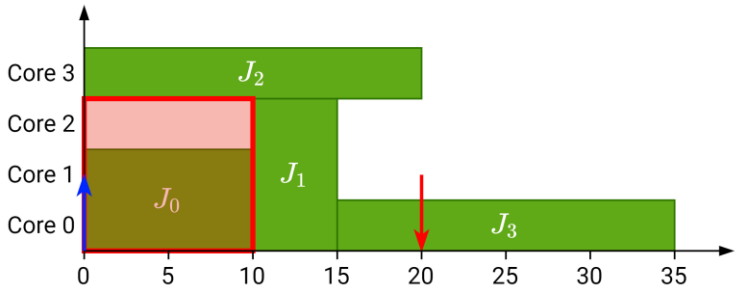
Moldable gang tasks evaluation

New scheduling policy

Paper submitted to RTSS 2020 for the non-preemptive analysis

Presented in CAPITAL workshop





Contributions

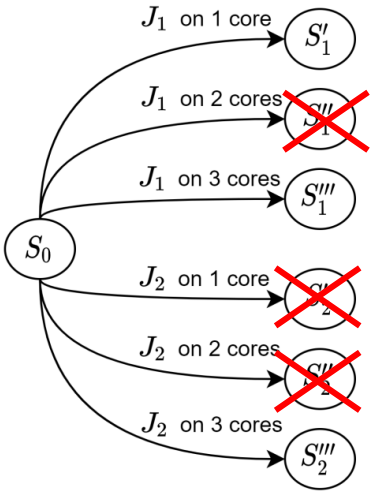
New schedulability analysis

Moldable gang tasks evaluation

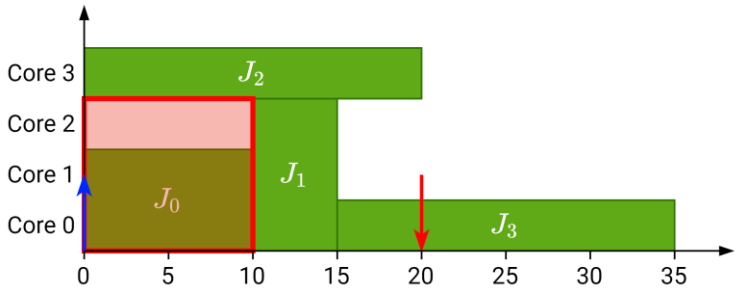
New scheduling policy

Paper submitted to RTSS 2020 for the non-preemptive analysis

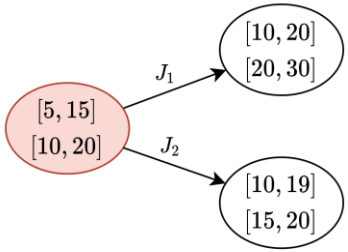
Presented in CAPITAL workshop



Thank you!

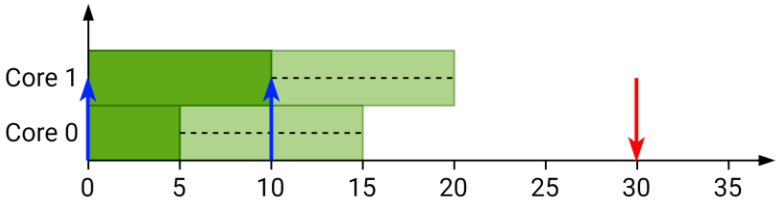


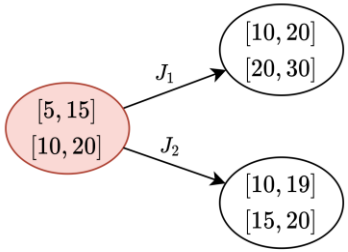
How does SAG work?



How does SAG work?

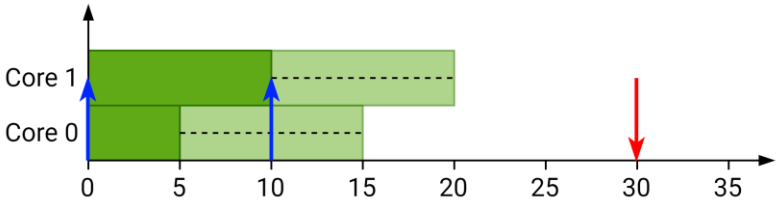
J_i	C_i^{min}	C_i^{max}	r_i	d_i	P_i
J_1	10	15	10	30	1
J_2	5	5	0	100	2

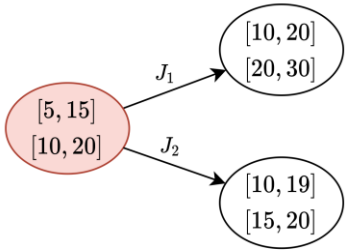




How does SAG work?

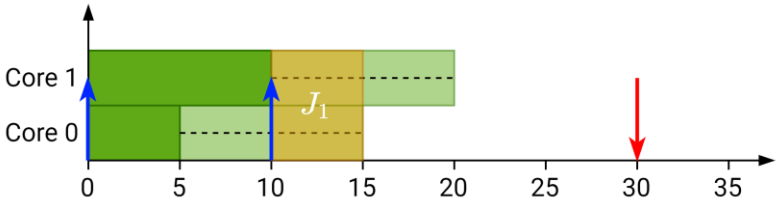
J_i	C_i^{min}	C_i^{max}	r_i	d_i	P_i
J_1	10	15	10	30	1
J_2	5	5	0	100	2

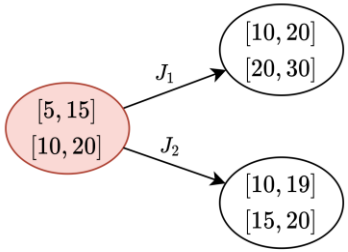




How does SAG work?

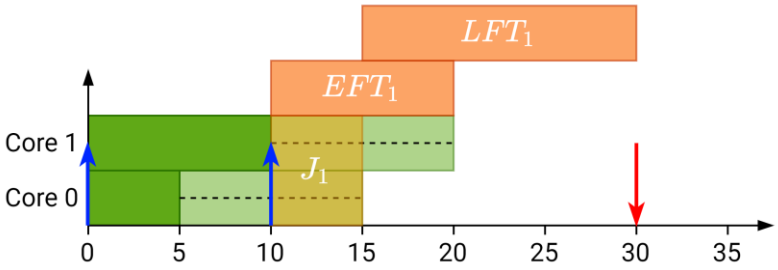
J_i	C_i^{min}	C_i^{max}	r_i	d_i	P_i
J_1	10	15	10	30	1
J_2	5	5	0	100	2

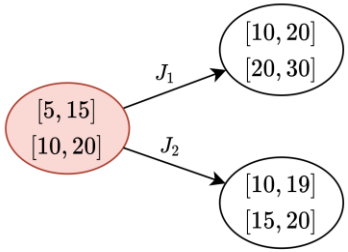




How does SAG work?

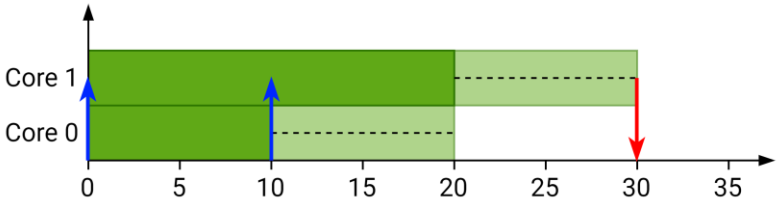
J_i	C_i^{min}	C_i^{max}	r_i	d_i	P_i
J_1	10	15	10	30	1
J_2	5	5	0	100	2

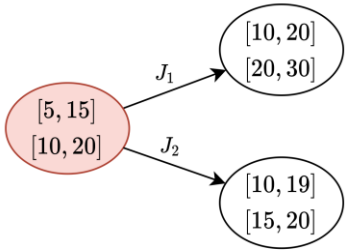




How does SAG work?

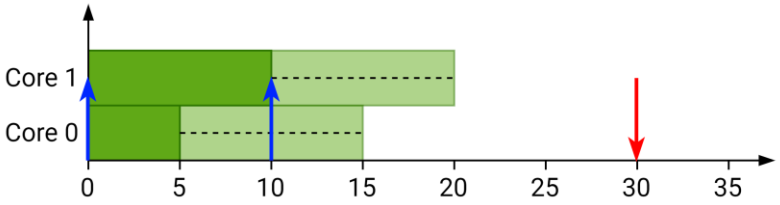
J_i	C_i^{min}	C_i^{max}	r_i	d_i	P_i
J_1	10	15	10	30	1
J_2	5	5	0	00	2

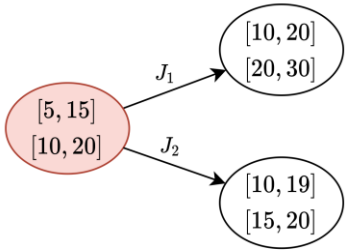




How does SAG work?

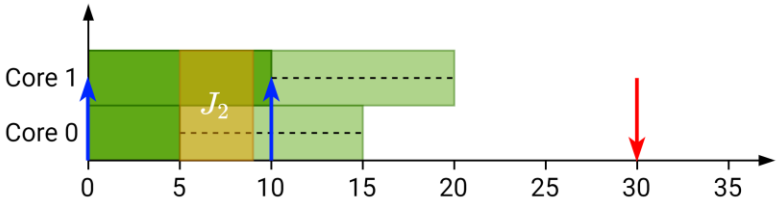
J_i	C_i^{min}	C_i^{max}	r_i	d_i	P_i
J_1	10	15	10	30	1
J_2	5	5	0	00	2

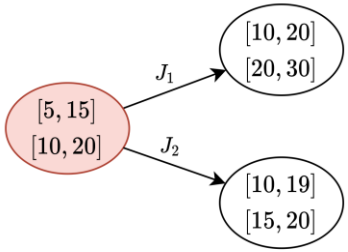




How does SAG work?

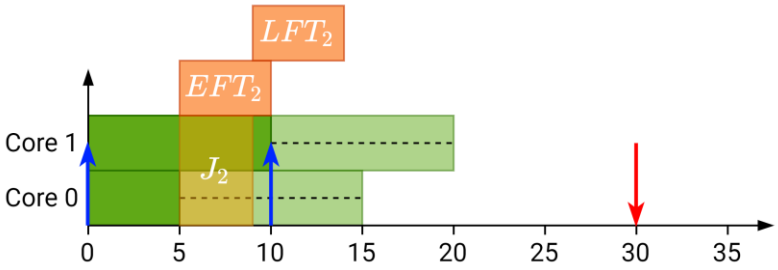
J_i	C_i^{min}	C_i^{max}	r_i	d_i	P_i
J_1	10	15	10	30	1
J_2	5	5	0	00	2

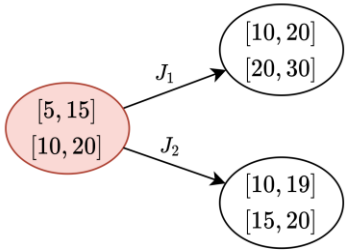




How does SAG work?

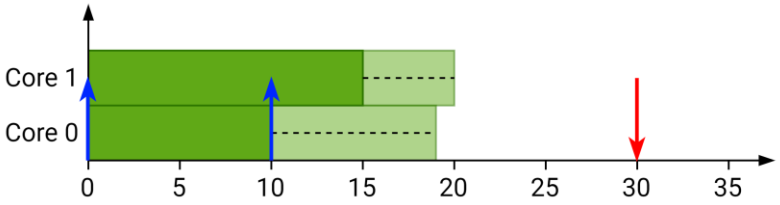
J_i	C_i^{min}	C_i^{max}	r_i	d_i	P_i
J_1	10	15	10	30	1
J_2	5	5	0	00	2

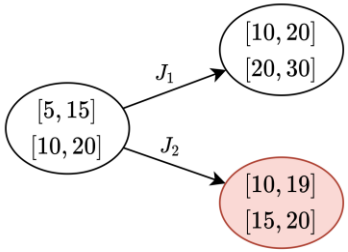




How does SAG work?

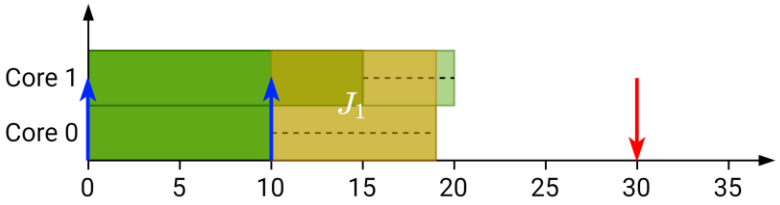
J_i	C_i^{min}	C_i^{max}	r_i	d_i	P_i
J_1	10	15	10	30	1
J_2	5	5	0	00	2

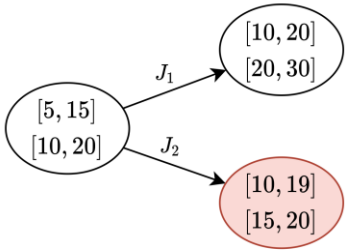




How does SAG work?

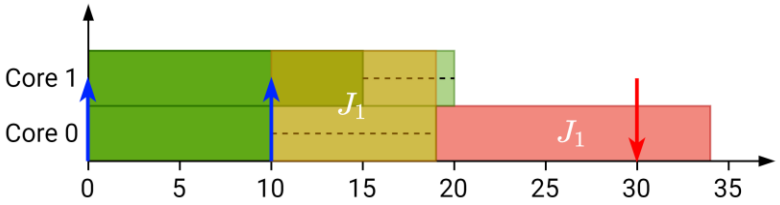
J_i	C_i^{min}	C_i^{max}	r_i	d_i	P_i
J_1	10	15	10	30	1
J_2	5	5	0	00	2

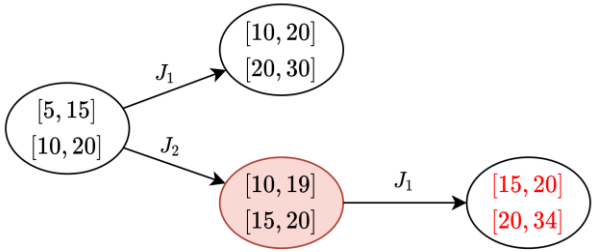




How does SAG work?

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

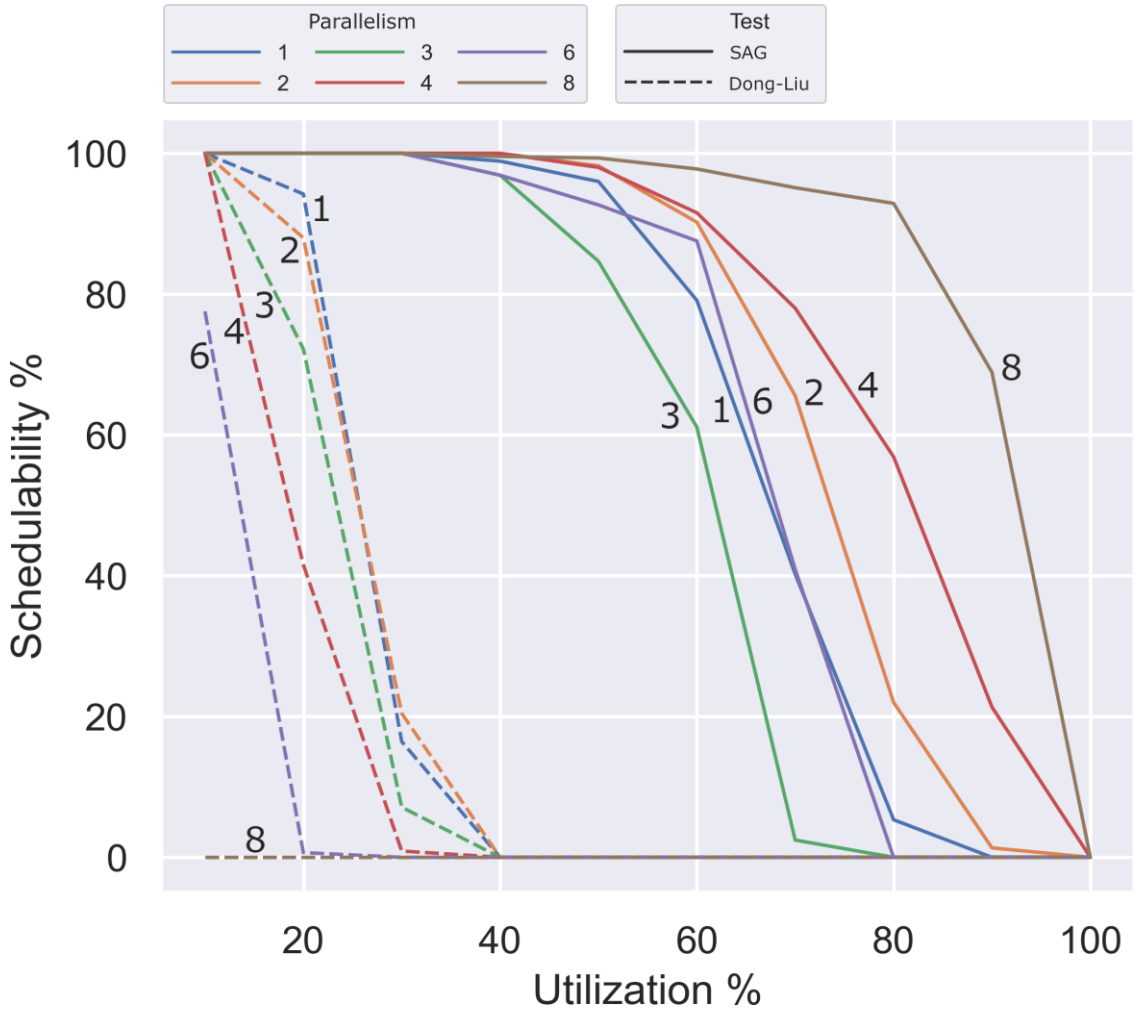




Runtime analysis

Randomly generated task sets

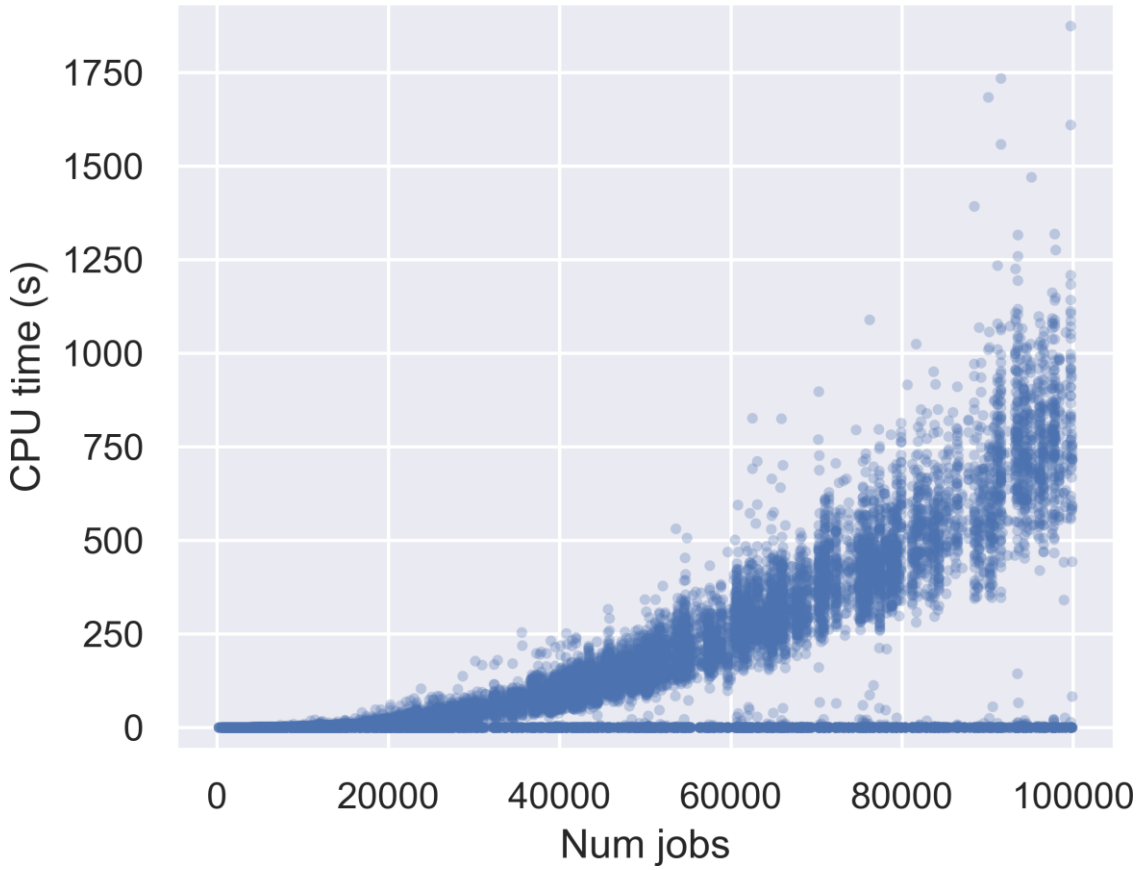
- System cores: 8
- System tasks: 20 rigid tasks
- Execution time variation: 50%
- Implicit deadlines
- Rate monotonic



Runtime analysis

Randomly generated task sets

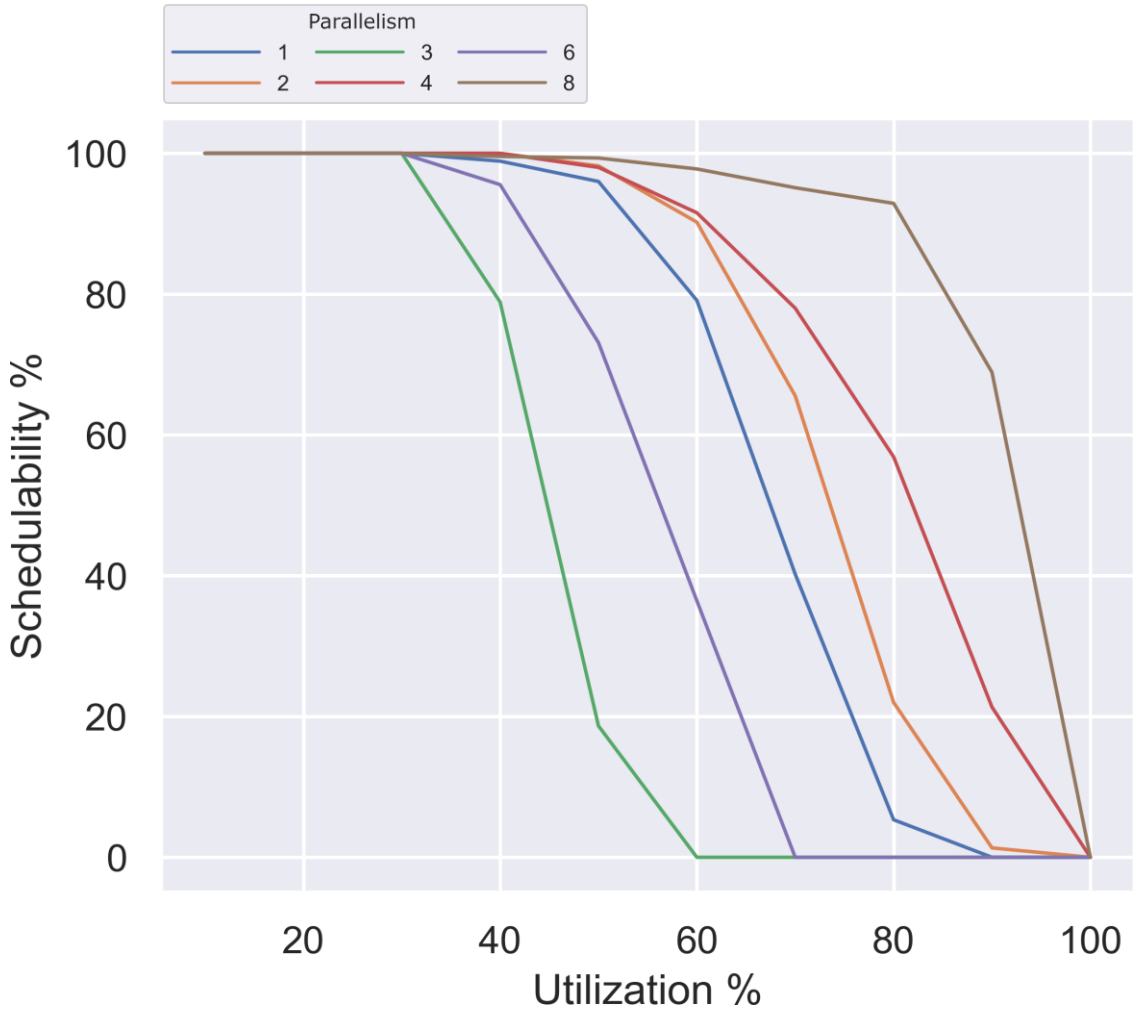
- System cores: 8
- System tasks: 20 rigid tasks
- Execution time variation: 50%
- Implicit deadlines
- Rate monotonic



Runtime analysis

Randomly generated task sets

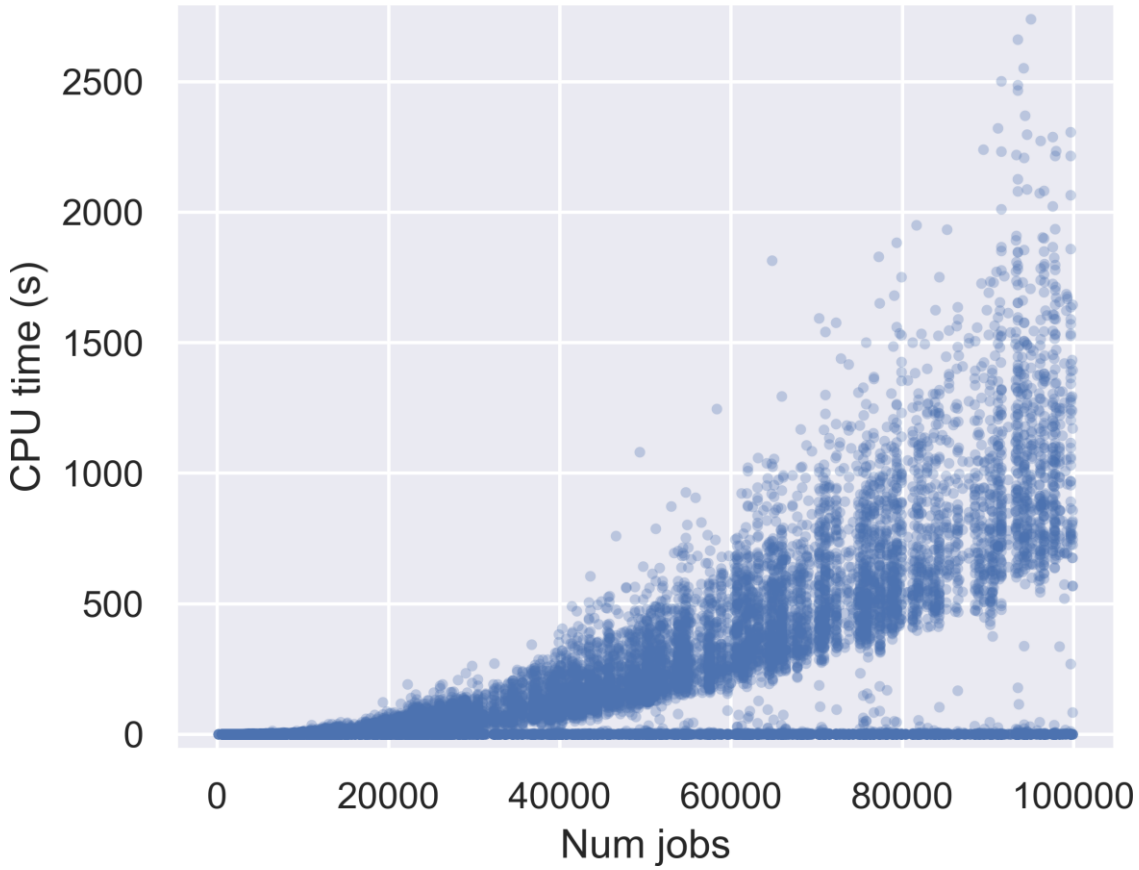
- System cores: 8
- System tasks: 20 moldable tasks
- Execution time variation: 50%
- Implicit deadlines
- Rate monotonic



Runtime analysis

Randomly generated task sets

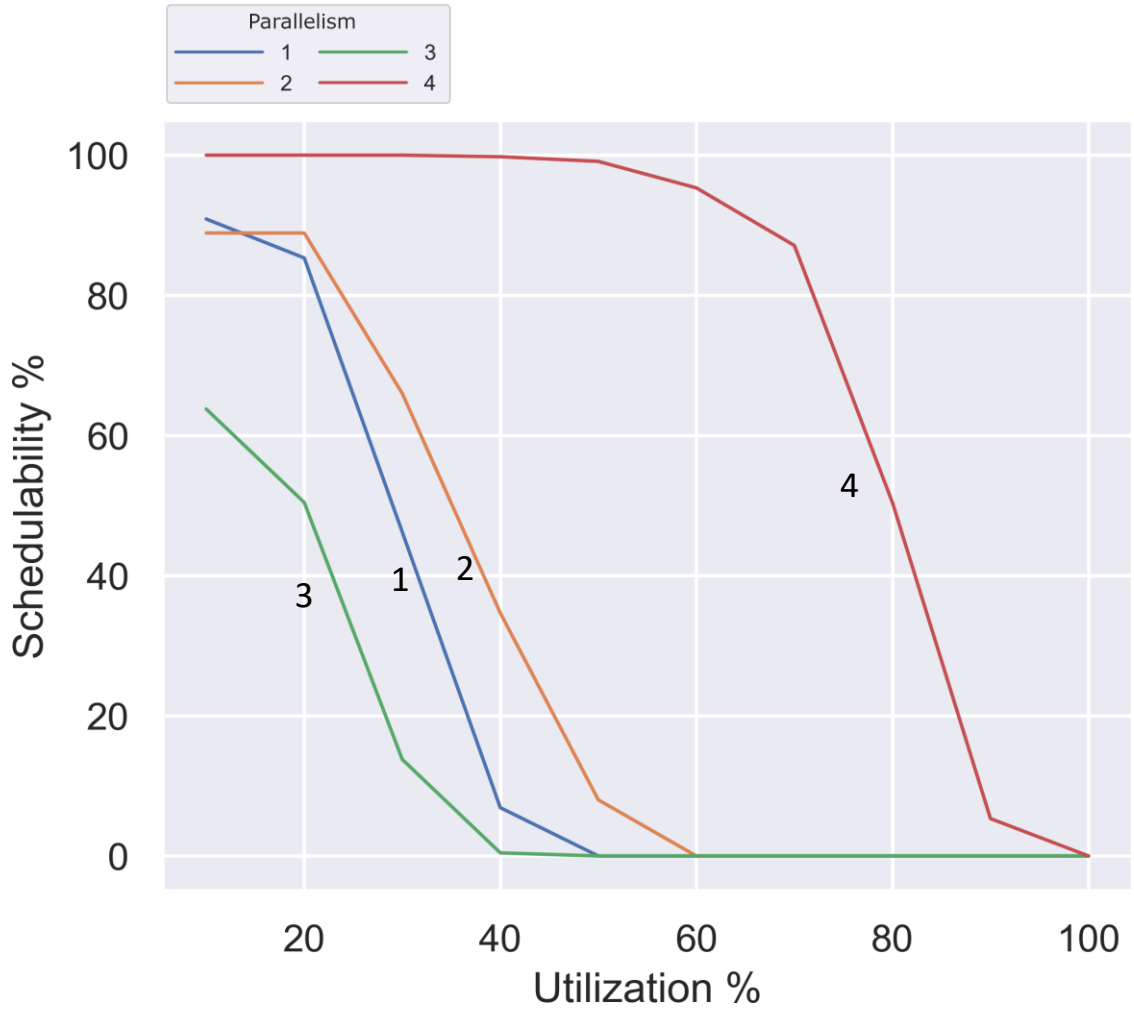
- System cores: 8
- System tasks: 20 moldable tasks
- Execution time variation: 50%
- Implicit deadlines
- Rate monotonic



Our analysis results

Randomly generated task sets

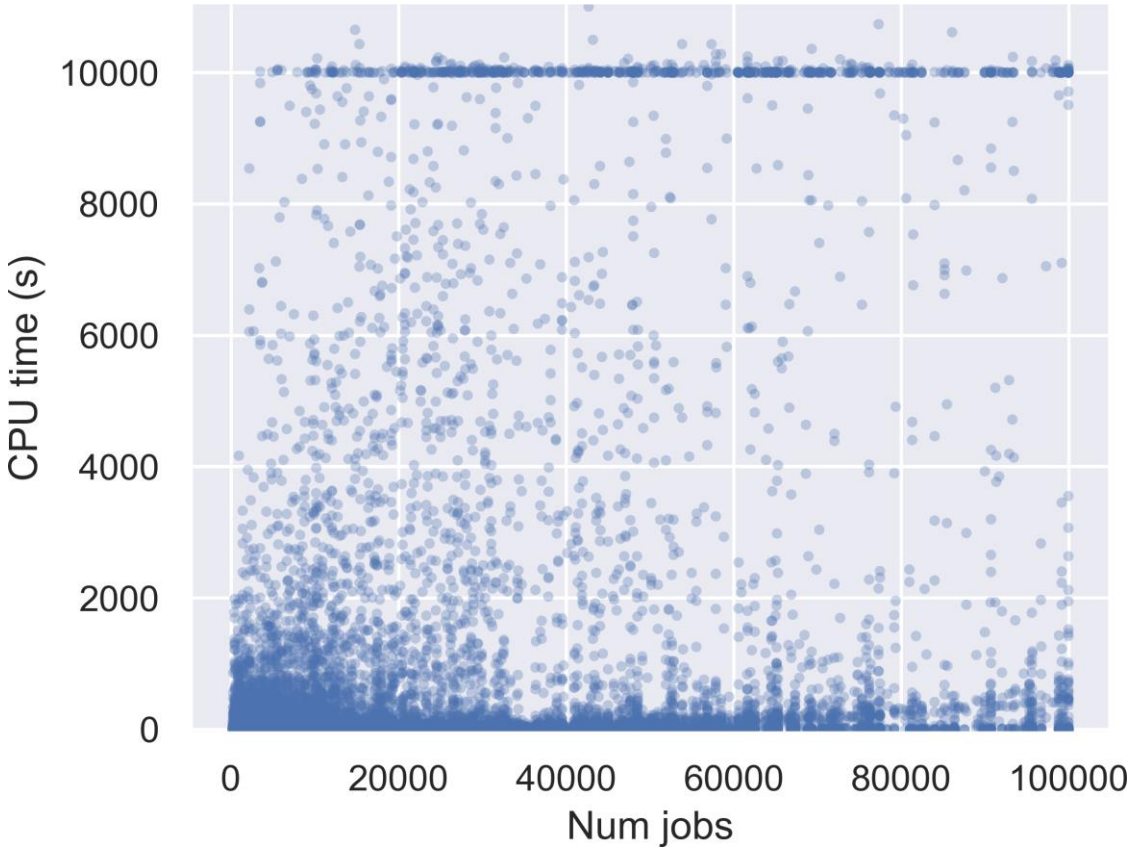
- System cores: 4
- System tasks:
  - 4 moldable tasks
  - 5 segments per task
- Execution time variation: 50%
- Implicit deadlines
- Rate monotonic



Our analysis results

Randomly generated task sets

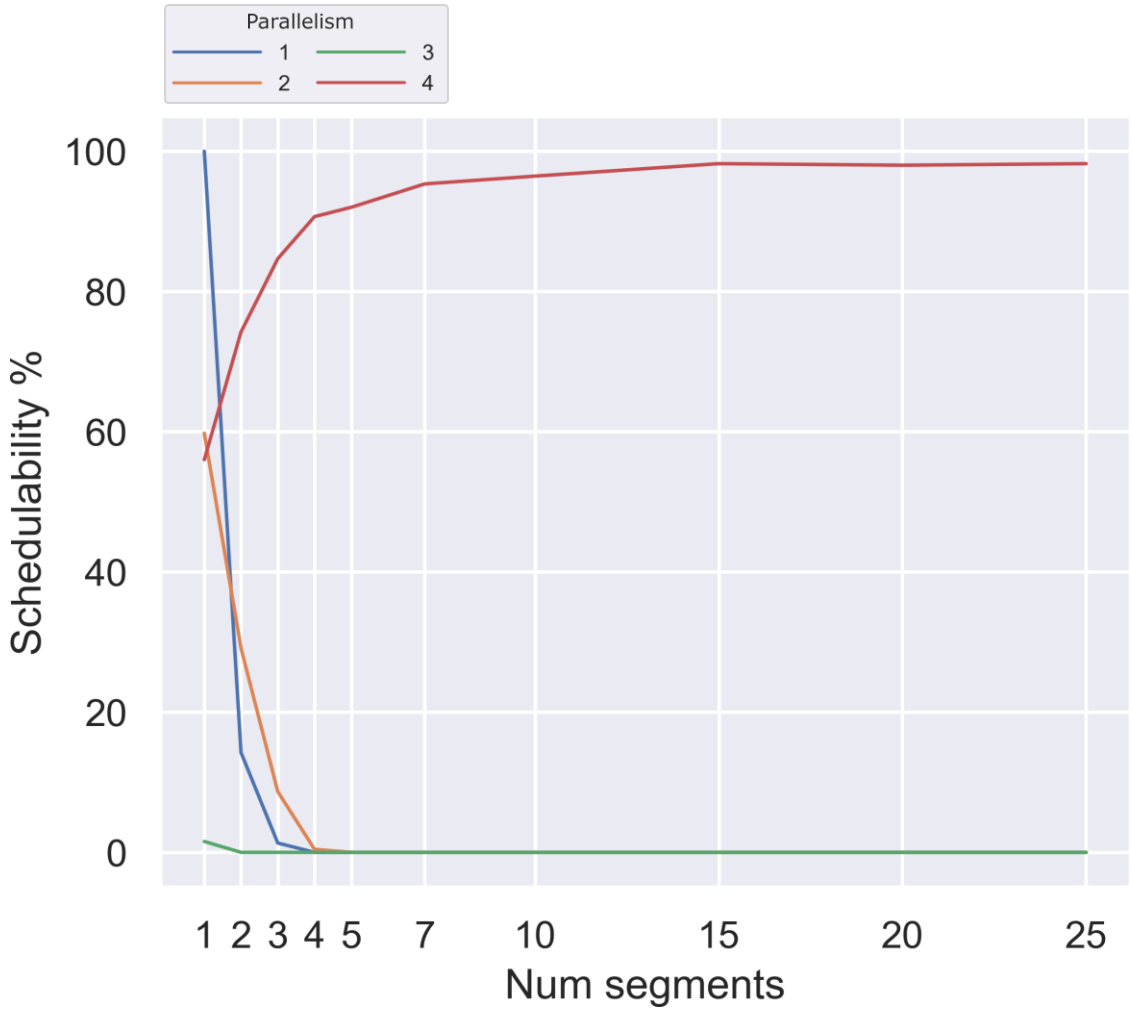
- System cores: 4
- System tasks:
  - 4 moldable tasks
  - 5 segments per task
- Execution time variation: 50%
- Implicit deadlines
- Rate monotonic



Our analysis results

Randomly generated task sets

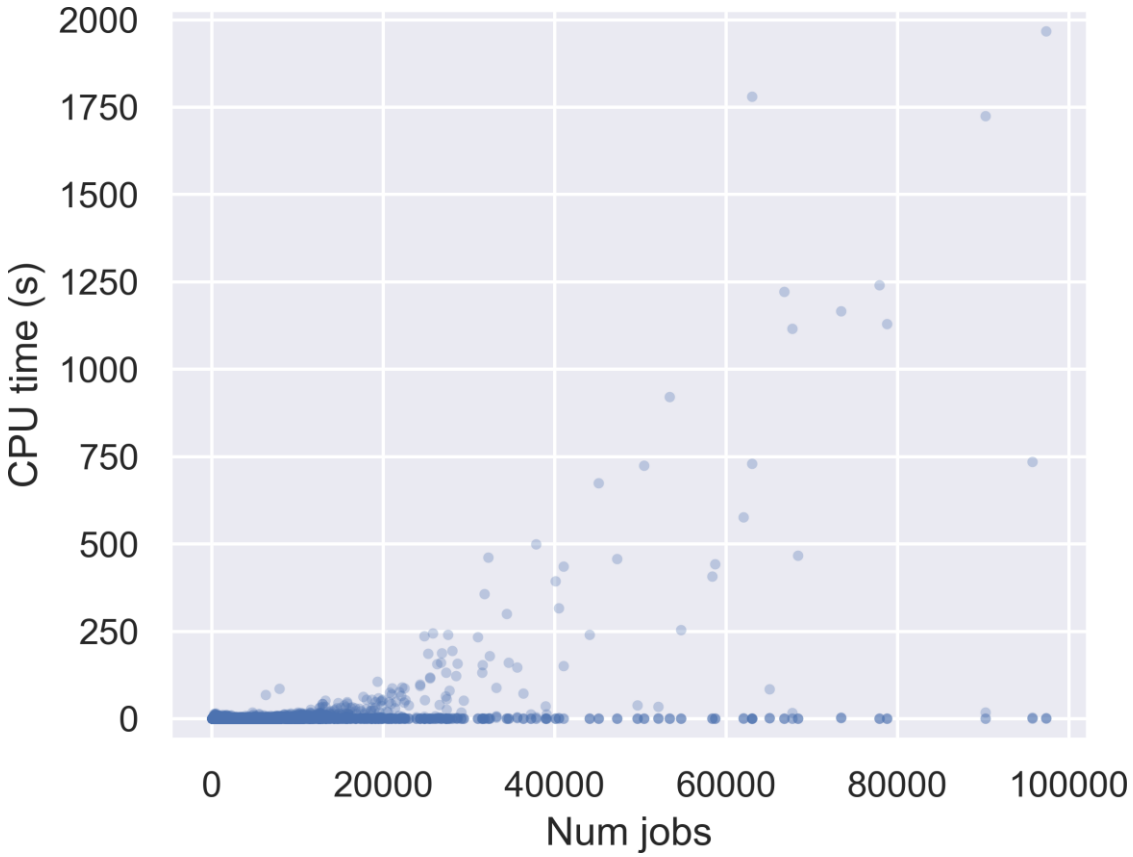
- System cores: 4
- System tasks: 4 moldable tasks
- Execution time variation: 50%
- System utilization: 70%



Our analysis results

Randomly generated task sets

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



Computational complexity

Computational complexity

- Number of states

Computational complexity

- Number of states
  - Possible job orderings: $O(n!)$

Computational complexity

- Number of states
  - Possible job orderings: $O(n!)$
  - Possible cores used by job: $O(m)$

Computational complexity

- Number of states
  - Possible job orderings: $O(n!)$
  - Possible cores used by job: $O(m)$
- Time to generate each state

Computational complexity

- Number of states
  - Possible job orderings: $O(n!)$
  - Possible cores used by job: $O(m)$
- Time to generate each state
  - Time to check other jobs: $O(n)$

Computational complexity

- Number of states
  - Possible job orderings: $O(n!)$
  - Possible cores used by job: $O(m)$
- Time to generate each state
  - Time to check other jobs: $O(n)$
  - Time to check cores available: $O(m^2)$