Response-Time Analysis for Non-Preemptive Global Scheduling with Spin Locks

Suhail Nogd

Supervised by: Dr. ir. Geoffrey Nelissen

Co-supervised by: Dr. Mitra Nasri

August 2020

Real-time systems

A system‘s correctness depends on

Logical correctness of results

AND

Timing correctness of results

Real-time systems

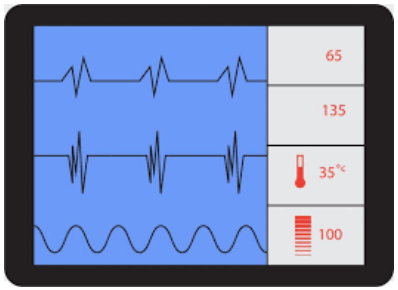
A system's correctness depends on

Logical correctness of results

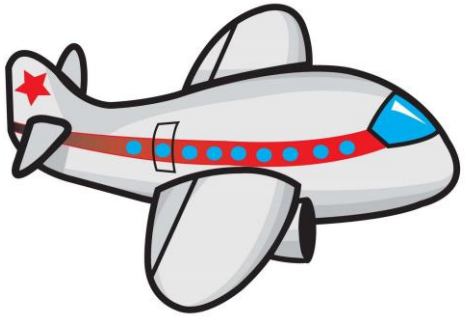
AND

Timing correctness of results

Automotive



Medical



Aircraft

Future real-time systems require high computing power

Future real-time systems require high computing power

Multiprocessor Platforms

- **Increased Computing Power**
- **Ability to exploit parallelism**

Future real-time systems require high computing power

Multiprocessor Platforms

- Increased Computing Power
- Ability to exploit parallelism

- Increased Complexity (Scheduling & Analysis

Multiprocessor scheduling

Partitioned

Tasks do not migrate

Migration

Multiprocessor scheduling

Task – A specific functionality of a system

↳ **Job –** An instance of a task

Partitioned

Tasks do not migrate

→

Migration

Multiprocessor scheduling

Task – A specific functionality of a system

↳ **Job –** An instance of a task

Partitioned	**Semi-Partitioned**
Tasks do not migrate	Some tasks can migrate

Migration

Multiprocessor scheduling

Task – A specific functionality of a system

↳ **Job –** An instance of a task

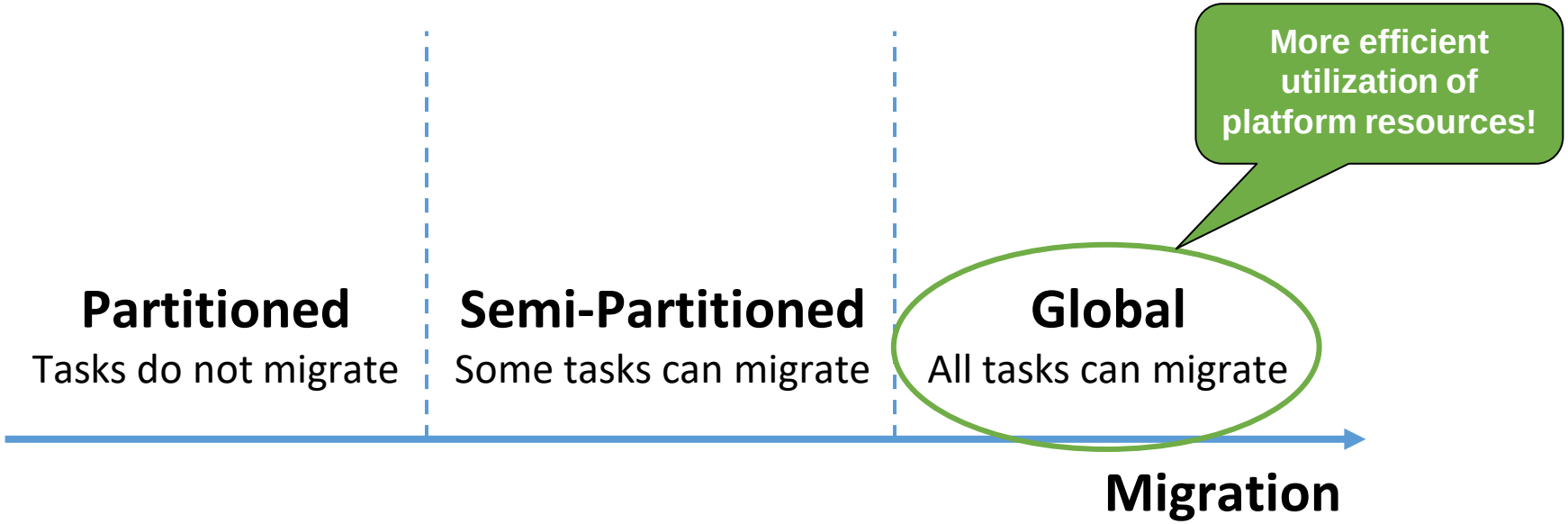
Partitioned	**Semi-Partitioned**	**Global**
Tasks do not migrate	Some tasks can migrate	All tasks can migrate

→ **Migration**

Multiprocessor scheduling

Task – A specific functionality of a system

↳ **Job –** An instance of a task



Global job-level fixed-priority (JLFP)

What is it?

A range of policies that assign jobs to processor(s) at specific time intervals to execute their workload

Global job-level fixed-priority (JLFP)

What is it?

A range of policies that assign jobs to processor(s) at specific time intervals to execute their workload

- *Earliest Deadline First (EDF)*

Global job-level fixed-priority (JLFP)

What is it?

A range of policies that assign jobs to processor(s) at specific time intervals to execute their workload

- *Earliest Deadline First (EDF)*
- *Fixed-Priority (FP)*

Shared resource

- **What is it?**
  - A resource that is requested by two or more tasks

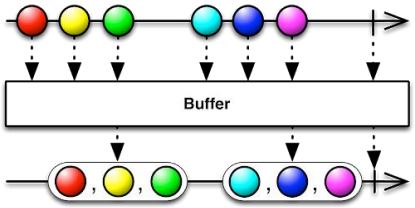
Shared resource

What is it?

- A resource that is requested by two or more tasks

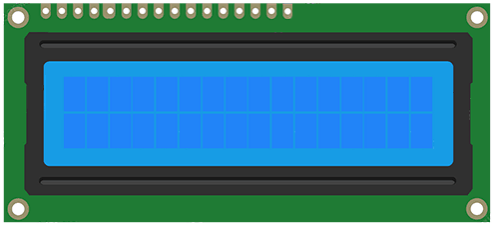
```
// global variable
int global = 5;
```

Global variable



Buffer

Timer



Display

Suspension-based locks vs. spin-based locks

Suspension-based locks
(i.e., yield core when blocked)

Spin-based locks
(i.e., busy-wait on core when blocked)

Suspension-based locks vs. spin-based locks

Suspension-based locks
(i.e., yield core when blocked)

- Blocked core used for useful executions

- Increased overhead
- Increased analysis complexity
- Increased OS support

Spin-based locks
(i.e., busy-wait on core when blocked)

Suspension-based locks vs. spin-based locks

Suspension-based locks
(i.e., yield core when blocked)

- Blocked core used for useful executions

- Increased overhead
- Increased analysis complexity
- Increased OS support

Spin-based locks
(i.e., busy-wait on core when blocked)

- Reduced analysis complexity
- Reduced OS support
- Reduced overhead

- Busy-waiting (core not used for useful computation)

Suspension-based locks vs. spin-based locks

Suspension-based locks
(i.e., yield core when blocked)

- Blocked core used for useful executions

- Increased overhead
- Increased analysis complexity
- Increased OS support

Spin-based locks
(i.e., busy-wait on core when blocked)

- Reduced analysis complexity
- Reduced OS support
- Reduced overhead

- Busy-waiting (core not used for useful computation)

Critical Section – Execution section that requires a shared resource

Suspension-based locks vs. spin-based locks

Suspension-based locks
(i.e., yield core when blocked)

- Blocked core used for useful executions
- More efficient for long critical sections

- Increased overhead
- Increased analysis complexity
- Increased OS support

Spin-based locks
(i.e., busy-wait on core when blocked)

- Reduced analysis complexity
- Reduced OS support
- Reduced overhead
- More efficient for short critical sections

- Busy-waiting (core not used for useful computation)

Critical Section – Execution section that requires a shared resource

Suspension-based locks vs. spin-based locks

Suspension-based locks
(i.e., yield core when blocked)

- Blocked core used for useful executions
- More efficient for long critical sections

- Increased overhead
- Increased analysis complexity
- Increased OS support

Spin-based locks
(i.e., busy-wait on core when blocked)

- Reduced analysis complexity
- Reduced OS support
- Reduced overhead
- More efficient for short critical sections

- Busy-waiting (core not used for useful computation)

Critical Section – Execution section that requires a shared resource

Suspension-based locks vs. spin-based locks

Suspension-based locks
(i.e., yield core when blocked)

- Blocked core used for useful executions
- More efficient for long critical sections

- Increased overhead
- Increased analysis complexity
- Increased OS support

Spin-based locks
(i.e., busy-wait on core when blocked)

- Reduced analysis complexity
- Reduced OS support
- Reduced overhead
- More efficient for short critical sections

- Busy-waiting (core not used for useful computation)

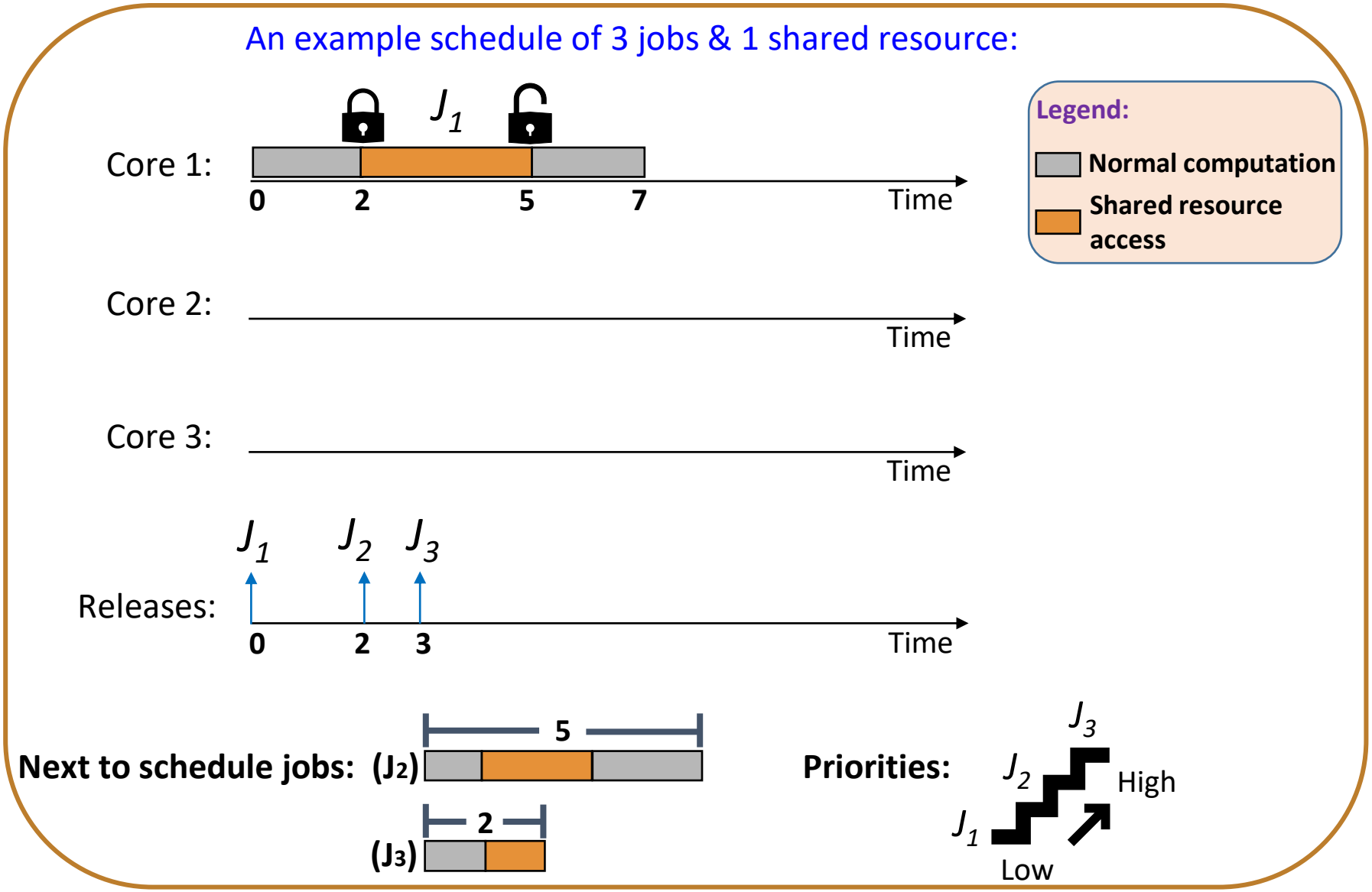
Critical Section – Execution section that requires a shared resource

FIFO-ordered spin locks

priority-ordered spin locks

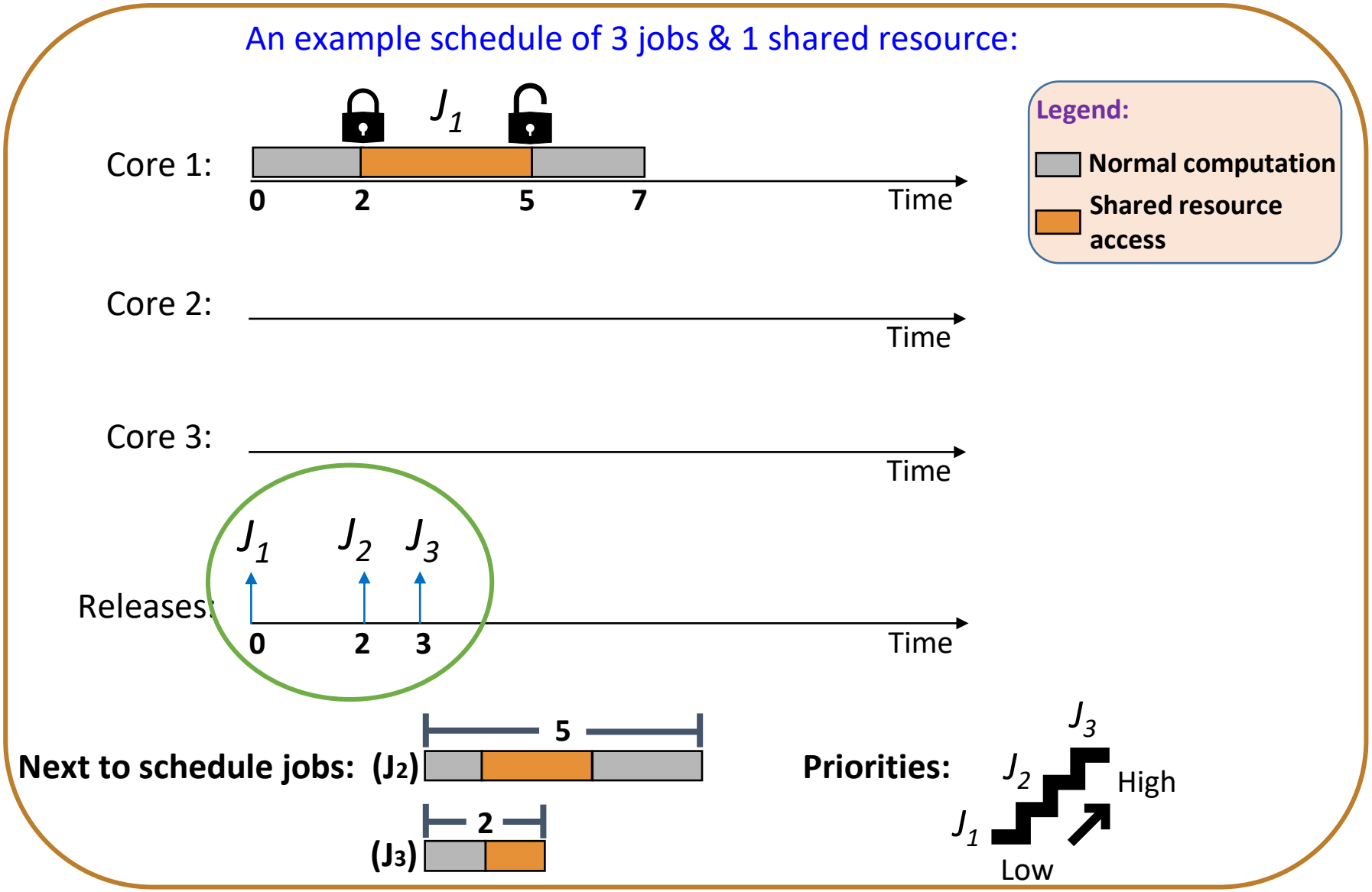
Spin locks

An example schedule of 3 jobs & 1 shared resource:



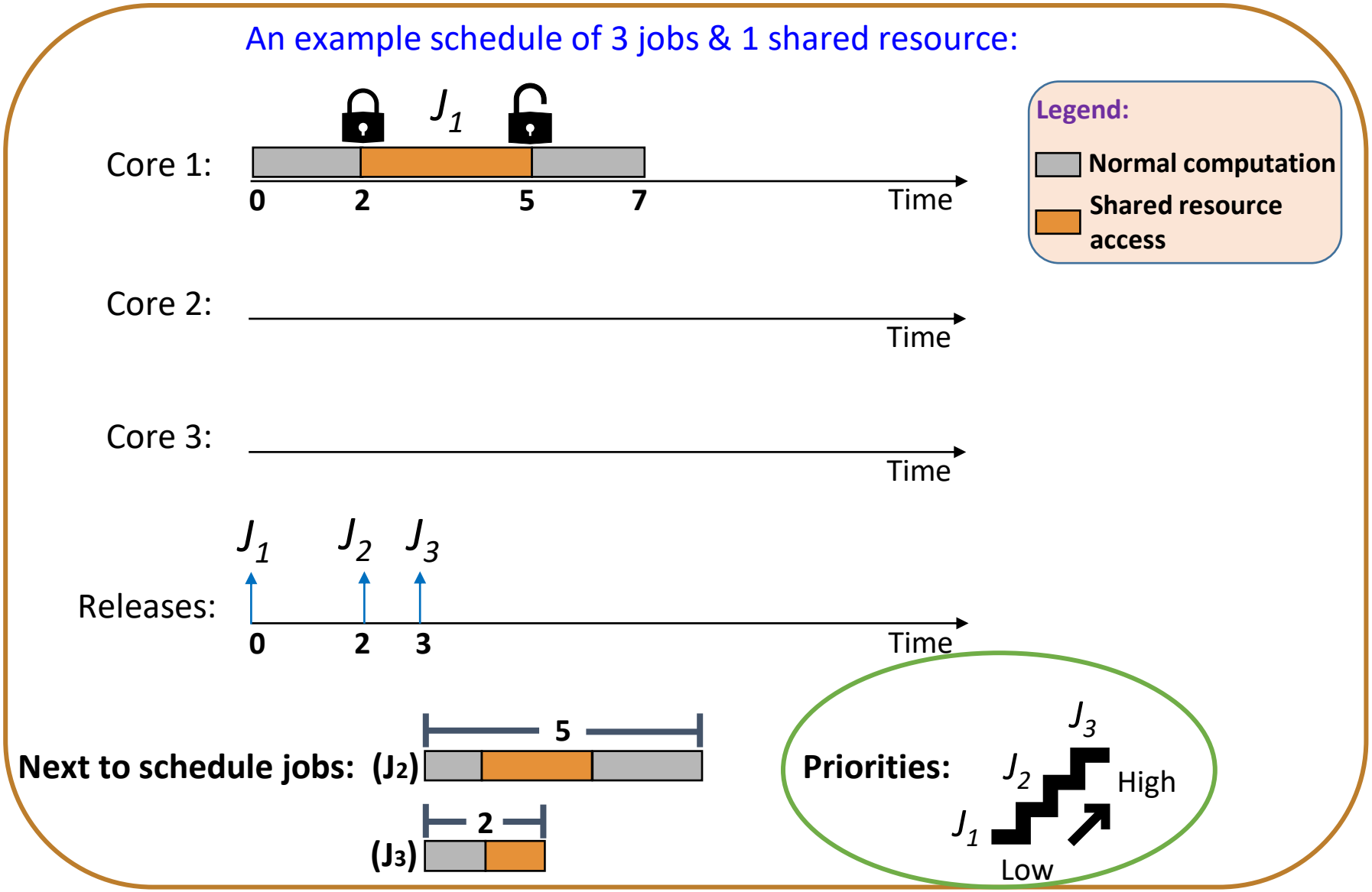
Spin locks

An example schedule of 3 jobs & 1 shared resource:



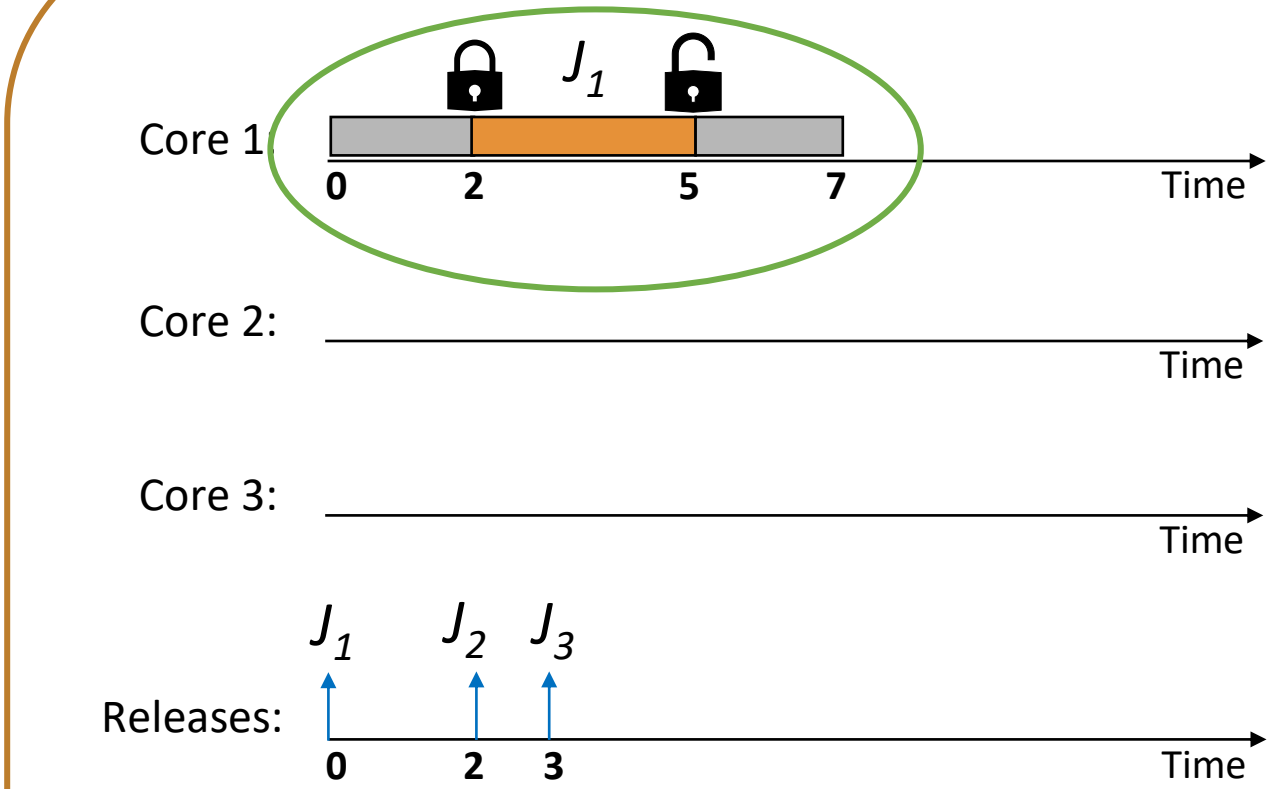
Spin locks

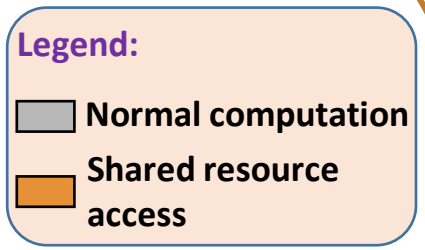
An example schedule of 3 jobs & 1 shared resource:

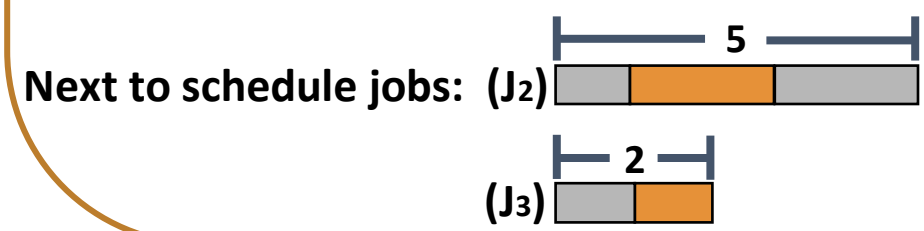


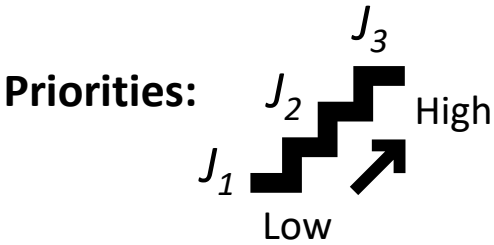
Spin locks

An example schedule of 3 jobs & 1 shared resource:



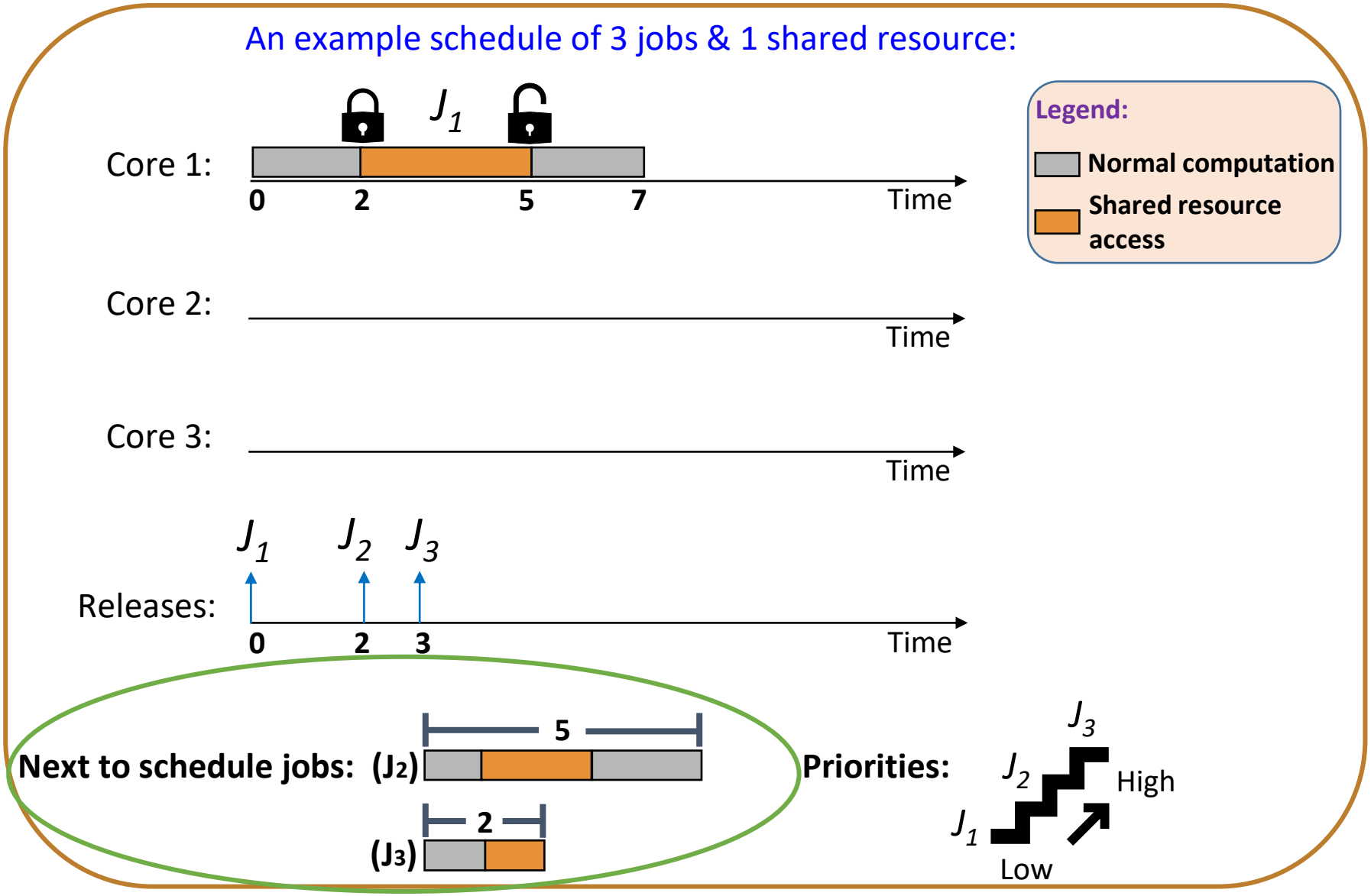






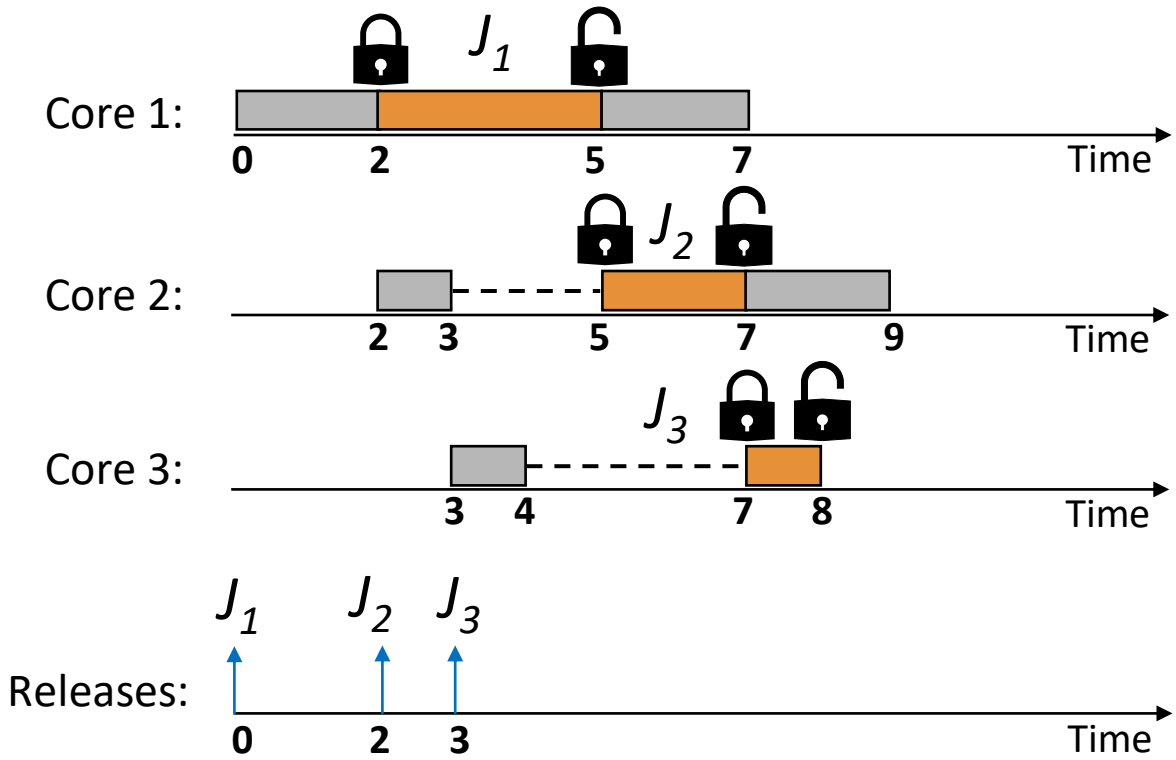
Spin locks

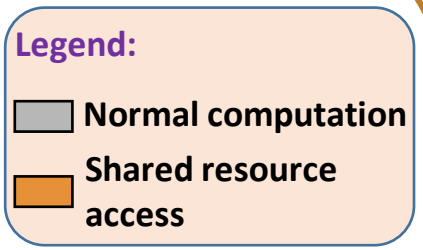
An example schedule of 3 jobs & 1 shared resource:



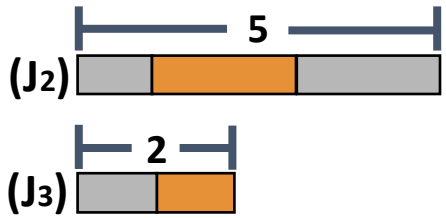
Fifo-ordered spin locks

An example schedule of 3 jobs & 1 shared resource:

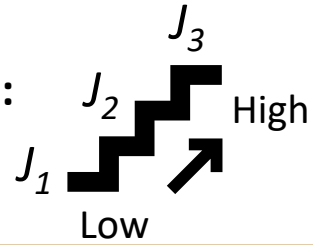




Next to schedule jobs:

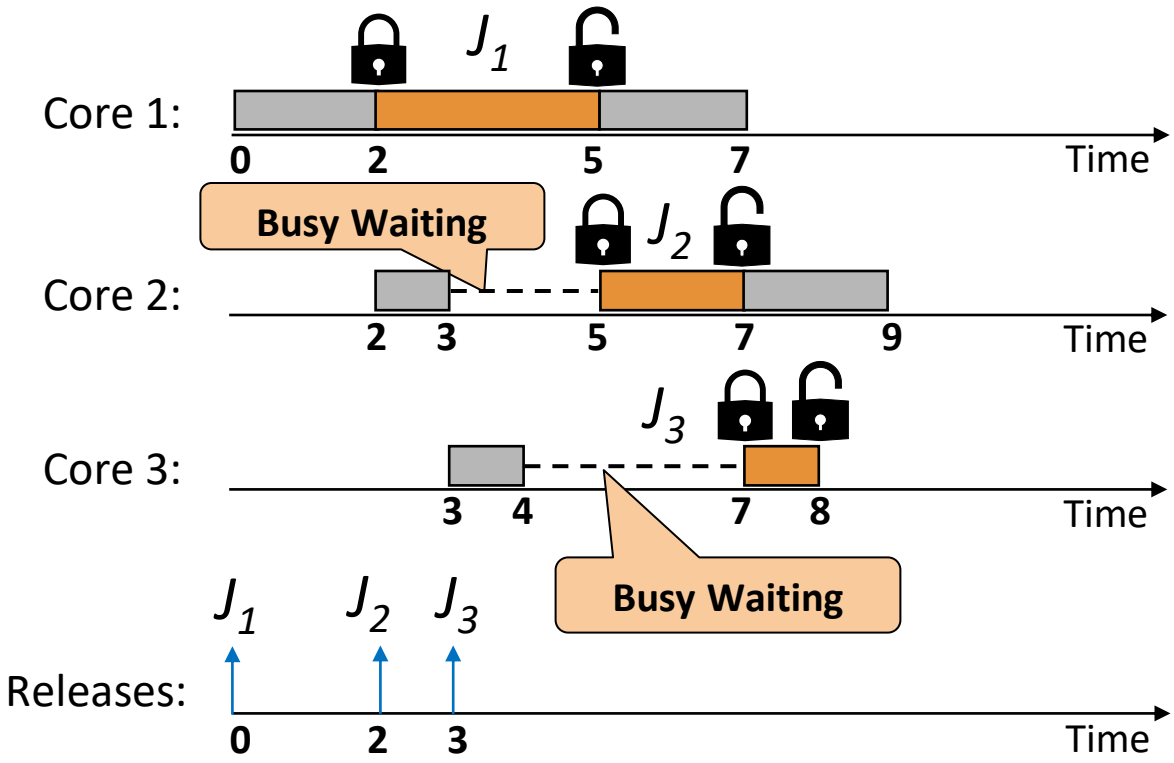


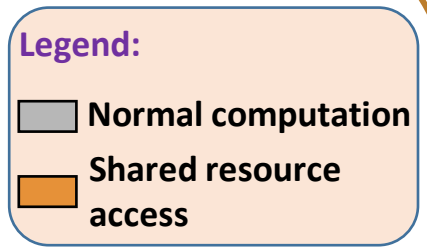
Priorities:



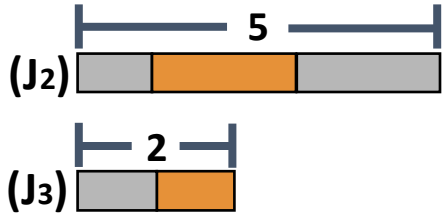
Fifo-ordered spin locks

An example schedule of 3 jobs & 1 shared resource:

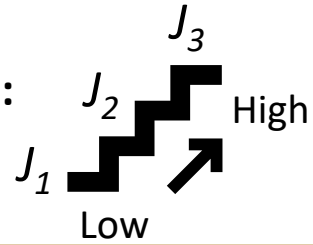




Next to schedule jobs:

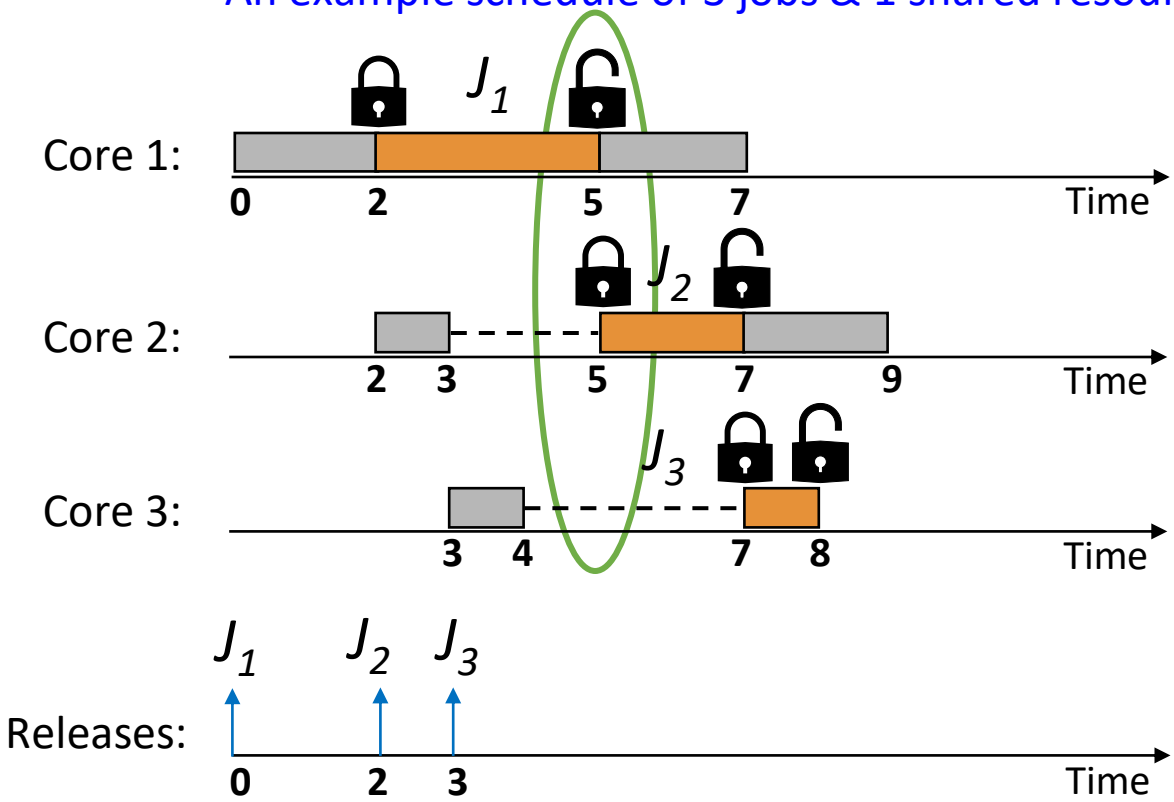


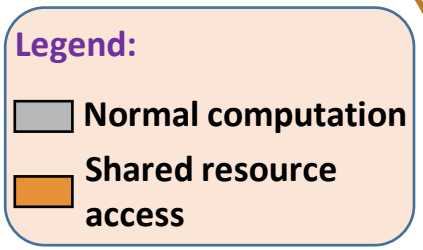
Priorities:



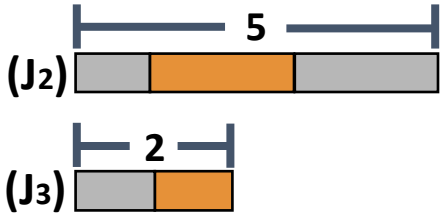
Fifo-ordered spin locks

An example schedule of 3 jobs & 1 shared resource:

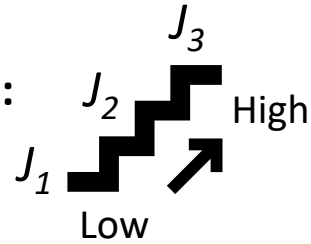




Next to schedule jobs:

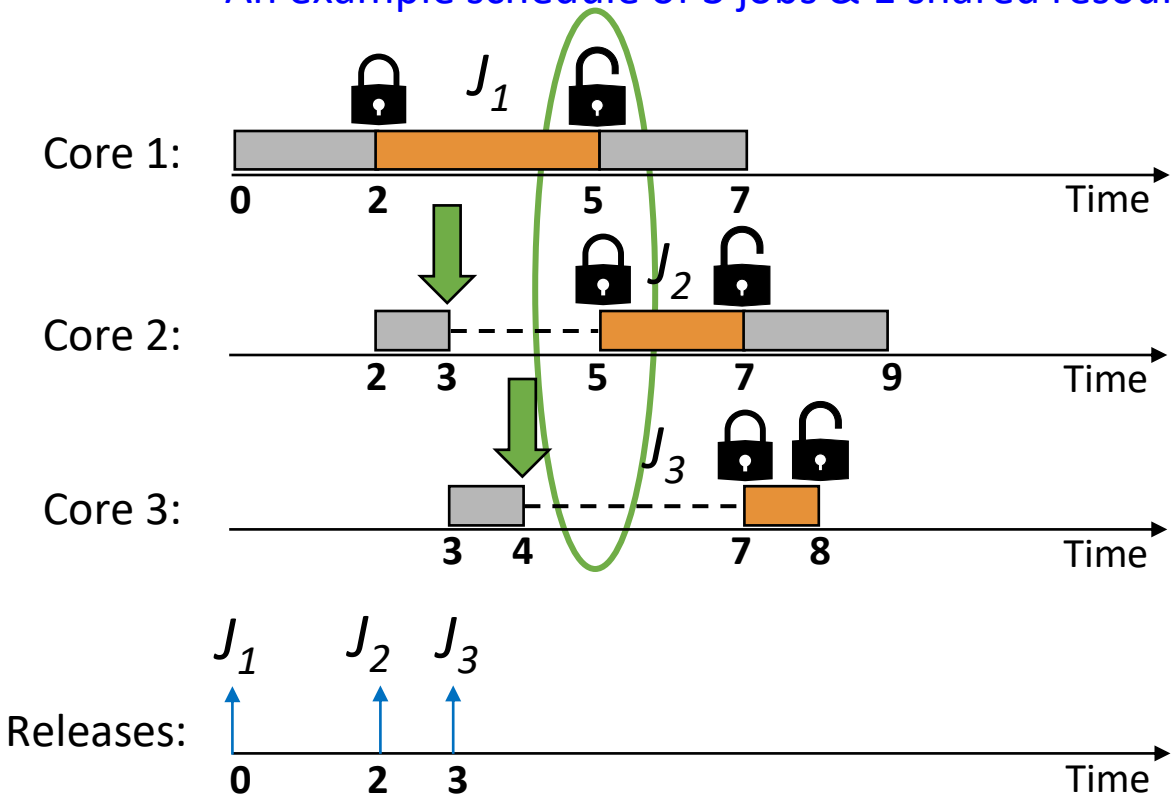


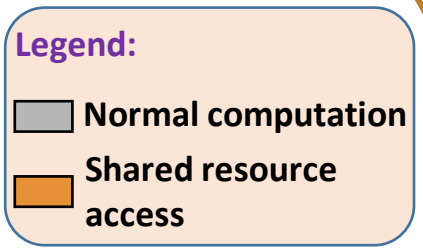
Priorities:



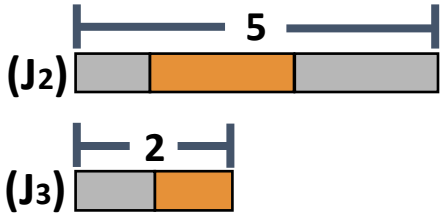
Fifo-ordered spin locks

An example schedule of 3 jobs & 1 shared resource:

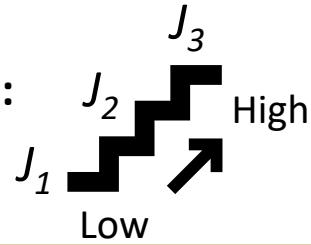




Next to schedule jobs:

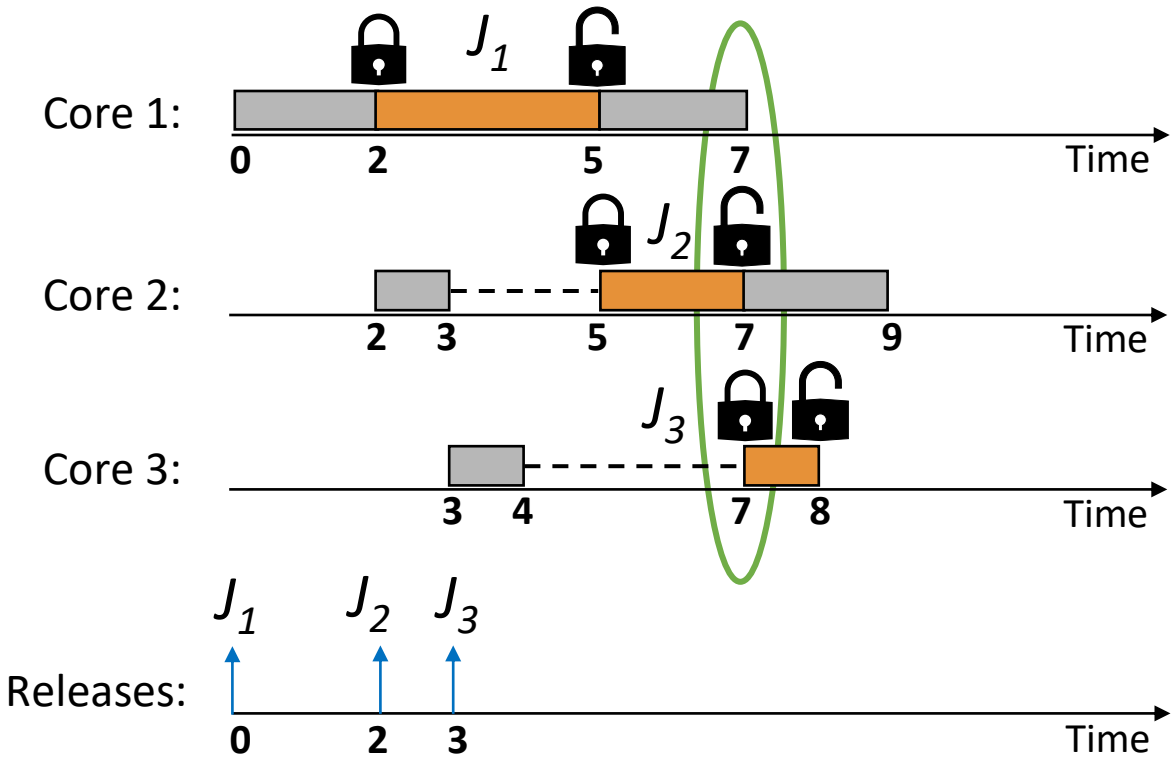


Priorities:



Fifo-ordered spin locks

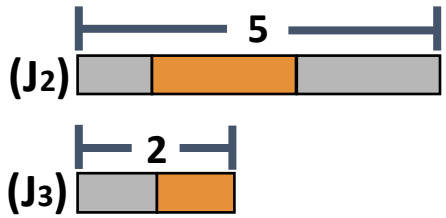
An example schedule of 3 jobs & 1 shared resource:



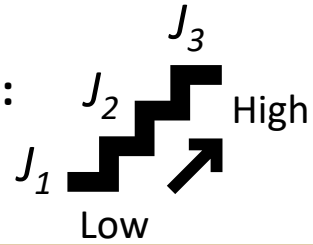
Legend:

- **Normal computation**
- **Shared resource access**

Next to schedule jobs:

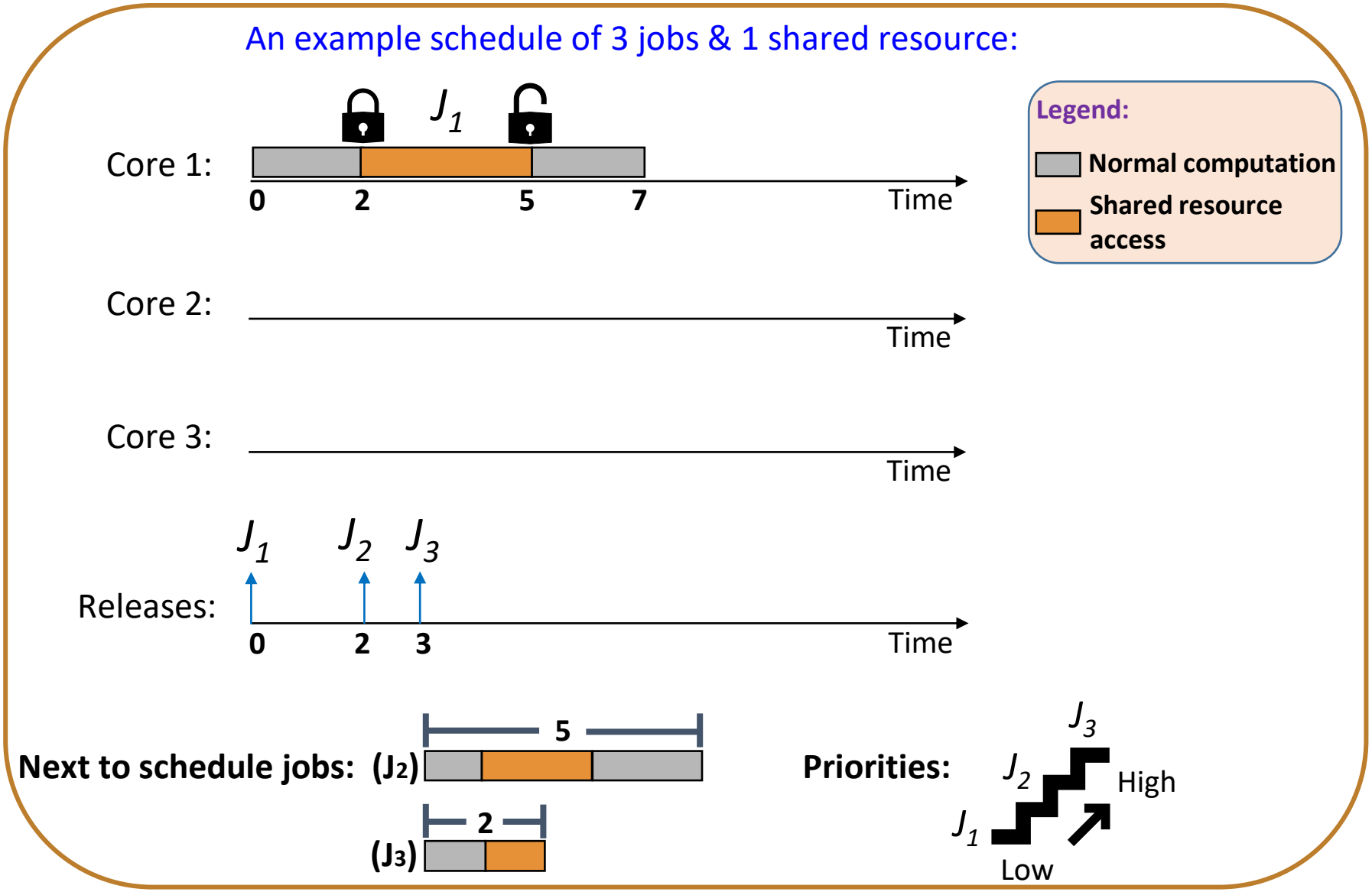


Priorities:



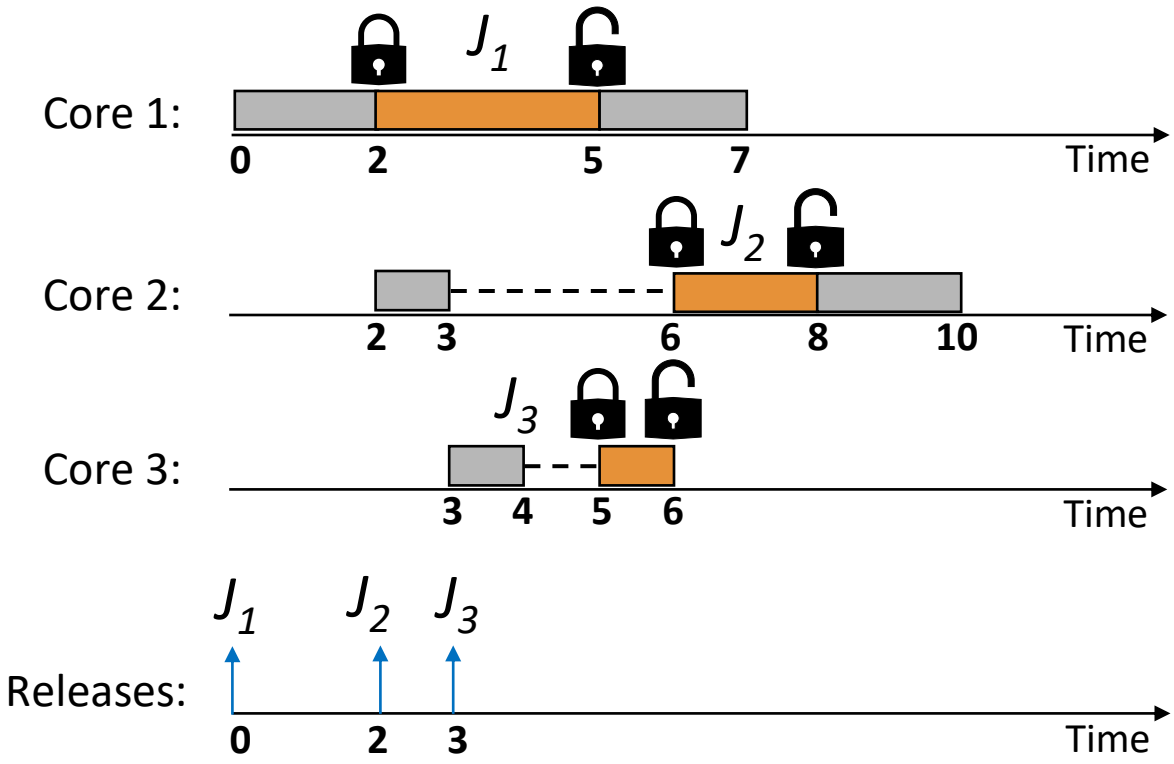
Priority-ordered spin locks

An example schedule of 3 jobs & 1 shared resource:



Priority-ordered spin locks

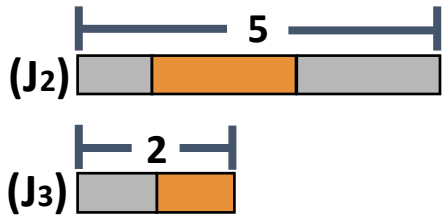
An example schedule of 3 jobs & 1 shared resource:



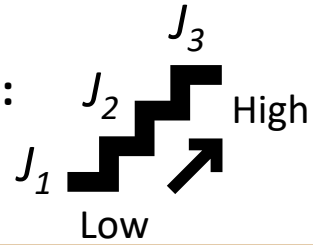
Legend:

- **Normal computation**
- **Shared resource access**

Next to schedule jobs:

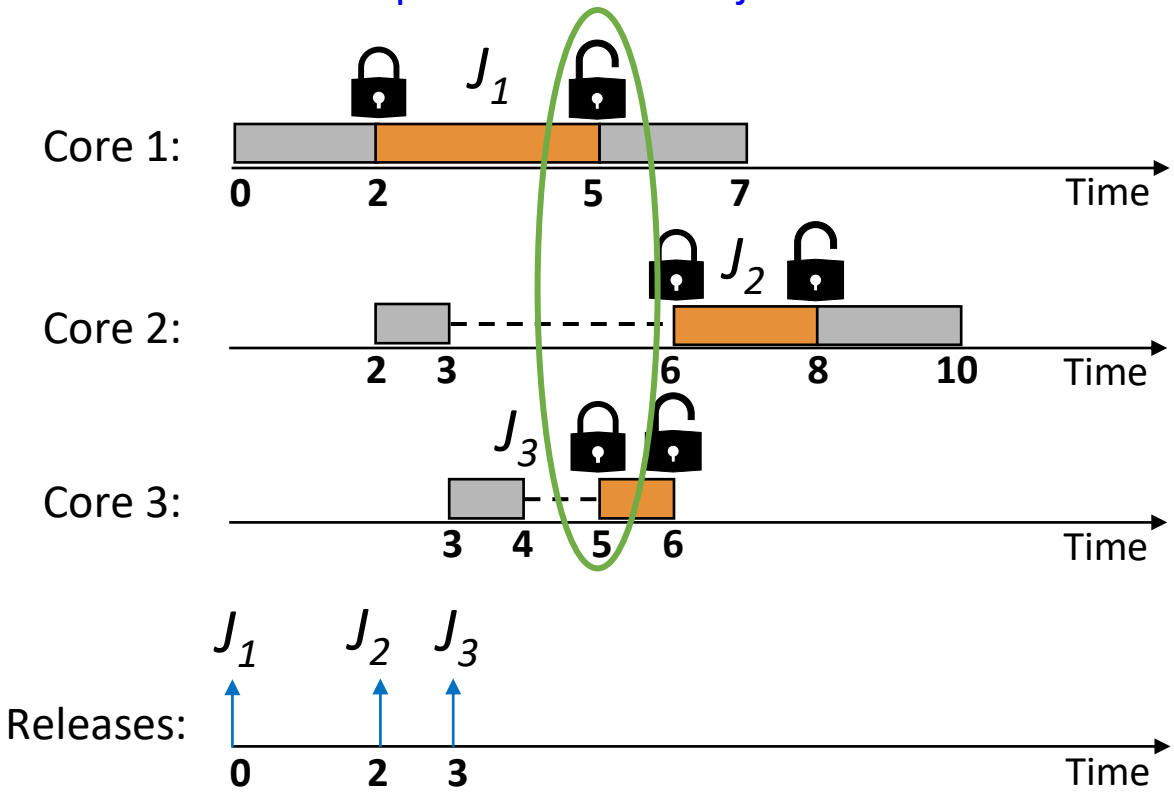


Priorities:



Priority-ordered spin locks

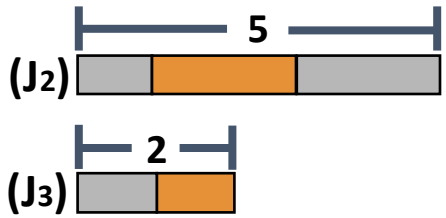
An example schedule of 3 jobs & 1 shared resource:



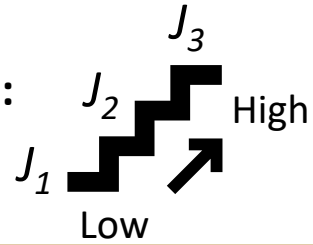
Legend:

- Normal computation
- Shared resource access

Next to schedule jobs:

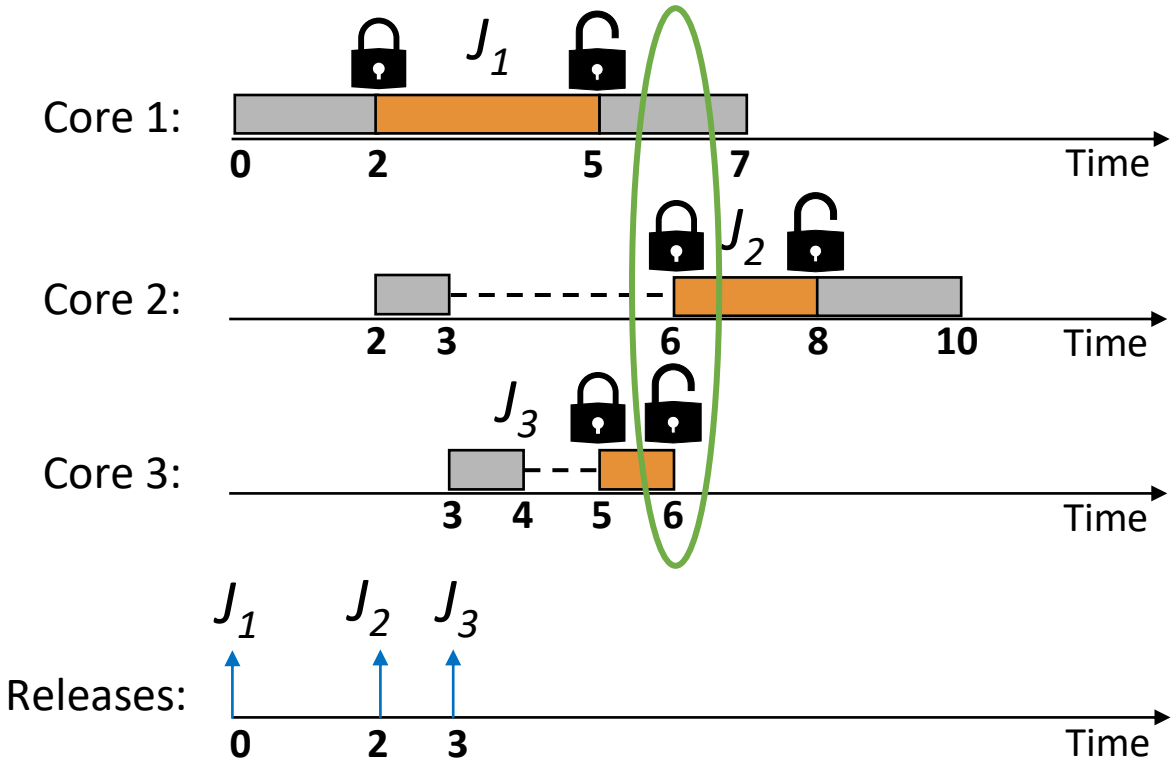


Priorities:



Priority-ordered spin locks

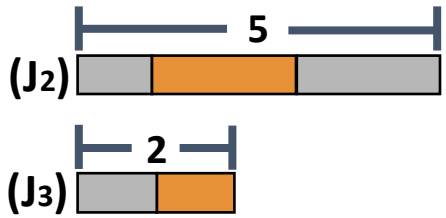
An example schedule of 3 jobs & 1 shared resource:



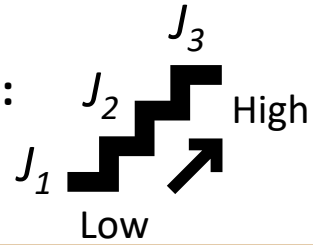
Legend:

- **Normal computation**
- **Shared resource access**

Next to schedule jobs:



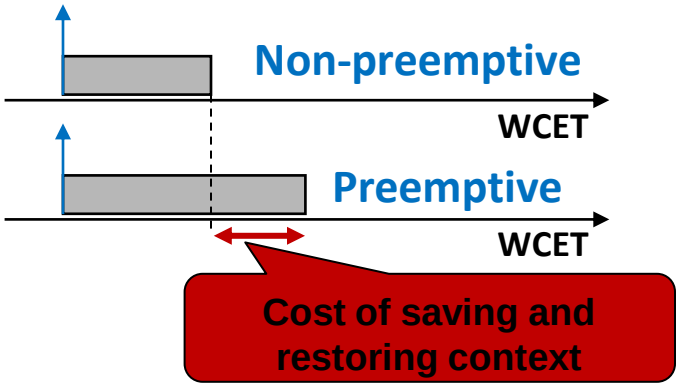
Priorities:



Non-preemptive scheduling

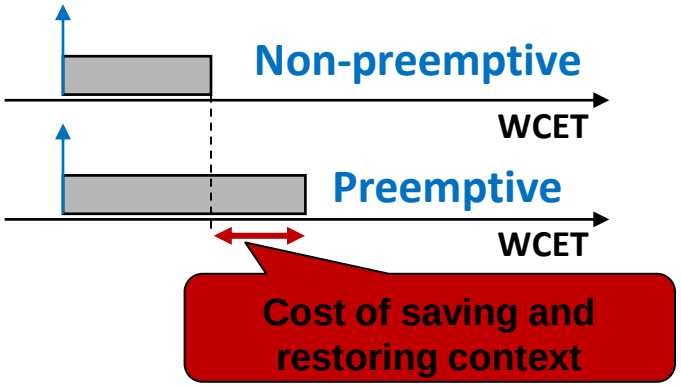
Non-preemptive scheduling

No added context switch cost



Non-preemptive scheduling

No added context switch cost

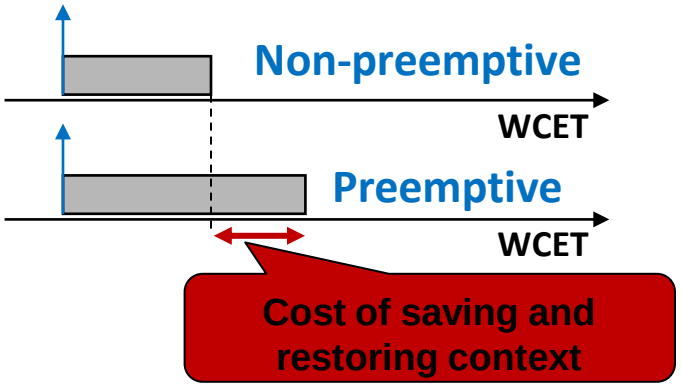


Less cache-related preemption delays

Fewer cache evictions

Non-preemptive scheduling

No added context switch cost



Less cache-related preemption delays

Fewer cache evictions

Improved timing predictability

➡ More accurate WCET estimation

Response-time analysis

- **What is it?**
  - Analysis that checks whether a given workload can be processed **within the specified timing requirements**

Response-time analysis

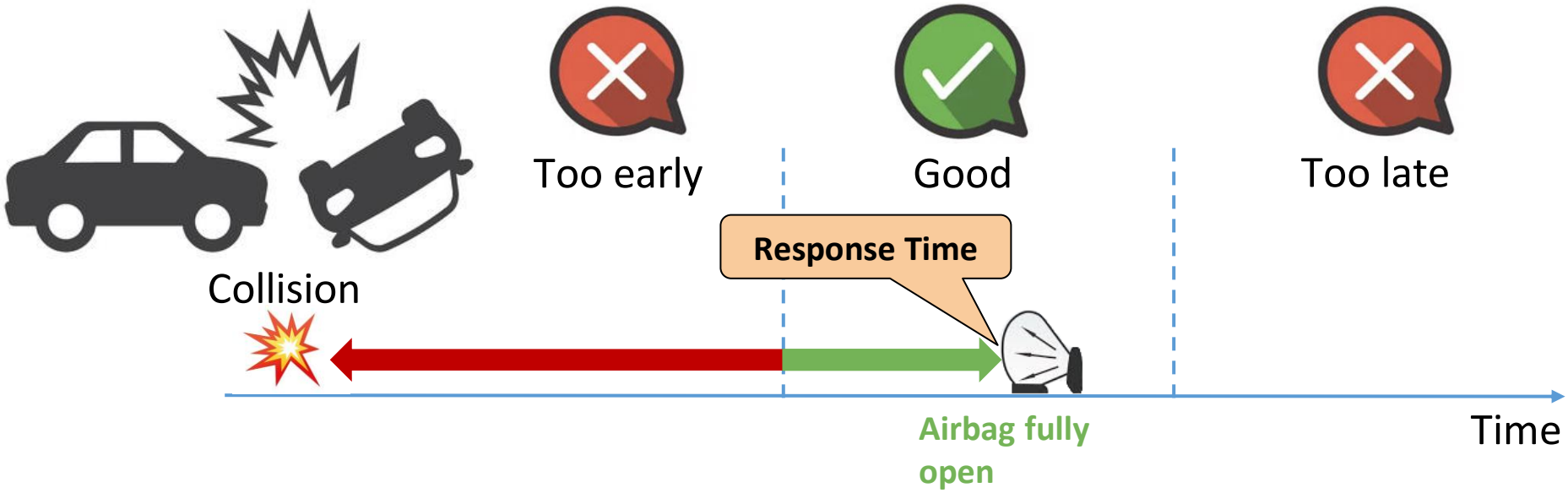
- **What is it?**
  - Analysis that checks whether a given workload can be processed **within the specified timing requirements**

- **Why is it important?**
  - Helps **predict** best- and worst-case timing behaviour of a system and allows to **validate** temporal requirements

Response-time analysis

- **What is it?**
  - Analysis that checks whether a given workload can be processed **within the specified timing constraints**

- **Why is it important?**
  - Helps **predict** best- and worst-case timing behaviour of a system and allows to **validate** temporal requirements



What analyses currently exist?

Blocking-aware schedulability analyses for global scheduling

Most interesting work

Blocking-aware schedulability analyses for global scheduling

Most interesting work

- Holistic blocking **spin-based** analysis **FMLP short**
- Holistic blocking **suspension-based** analysis **OMLP**

[1] Brandenburg, B.B., 2019. Multiprocessor real-time locking protocols: A systematic review. *arXiv preprint arXiv:1909.09600*.

Blocking-aware schedulability analyses for global scheduling

Most interesting work

- Holistic blocking spin-based analysis FMLP short
- Holistic blocking suspension-based analysis OMLP
- Linear-programming suspension-based analysis **FMLP long**
- Linear-programming suspension-based analysis **PIP**

[1] Brandenburg, B.B., 2019. Multiprocessor real-time locking protocols: A systematic review. *arXiv preprint arXiv:1909.09600*.

Blocking-aware schedulability analyses for global scheduling

Most interesting work

- Holistic blocking spin-based analysis FMLP short
- Holistic blocking suspension-based analysis OMLP
- Linear-programming suspension-based analysis FMLP long
- Linear-programming suspension-based analysis PIP

Preemptive

[1] Brandenburg, B.B., 2019. Multiprocessor real-time locking protocols: A systematic review. *arXiv preprint arXiv:1909.09600*.

Blocking-aware schedulability analyses for global scheduling

Most interesting work

- Holistic blocking spin-based analysis FMLP short
- Holistic blocking suspension-based analysis OMLP
- Linear-programming suspension-based analysis FMLP long
- Linear-programming suspension-based analysis PIP

} **Preemptive**

Analysis for non-preemptive execution and spin locks ?

[1] Brandenburg, B.B., 2019. Multiprocessor real-time locking protocols: A systematic review. *arXiv preprint arXiv:1909.09600*.

Blocking-aware schedulability analyses for global scheduling

Most interesting work

- Holistic blocking spin-based analysis FMLP short
- Holistic blocking suspension-based analysis OMLP
- Linear-programming suspension-based analysis FMLP long
- Linear-programming suspension-based analysis PIP

Preemptive

Analysis for non-preemptive execution and spin locks ?

[1] Brandenburg, B.B., 2019. Multiprocessor real-time locking protocols: A systematic review. *arXiv preprint arXiv:1909.09600*.

The problem in a nutshell

The problem in a nutshell

We obtain the best-case and worst-case response time

The problem in a nutshell

We obtain the best-case and worst-case response time

Workload model

Tasks/Jobs that share resources (Spin locks)

The problem in a nutshell

We obtain the best-case and worst-case response time

Workload model	Execution model
Tasks/Jobs that share resources (Spin locks)	Non-preemptive

The problem in a nutshell

We obtain the best-case and worst-case response time

Workload model	Execution model	Platform model
Tasks/Jobs that share resources (Spin locks)	Non-preemptive	Multicore (identical cores)

The problem in a nutshell

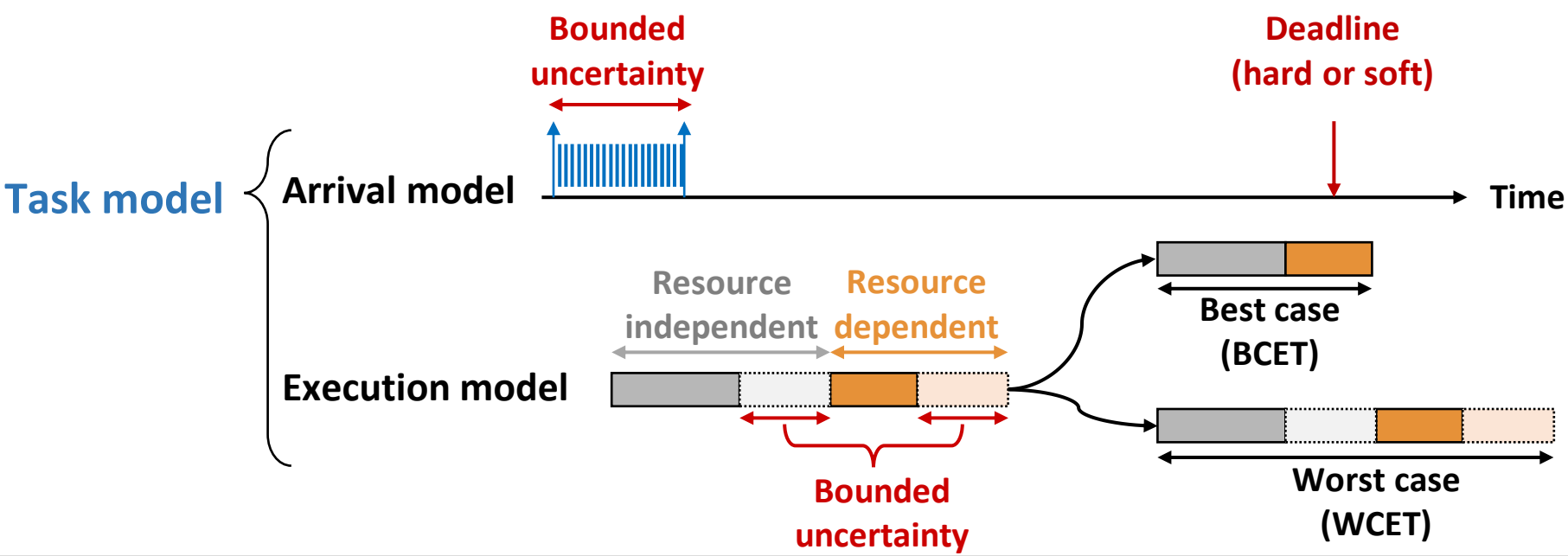
We obtain the best-case and worst-case response time

Workload model	Execution model	Platform model	Scheduler model
Tasks/Jobs that share resources (Spin locks)	Non-preemptive	Multicore (identical cores)	Global job-level fixed priority (JLFP)

The problem in a nutshell

We obtain the best-case and worst-case response time

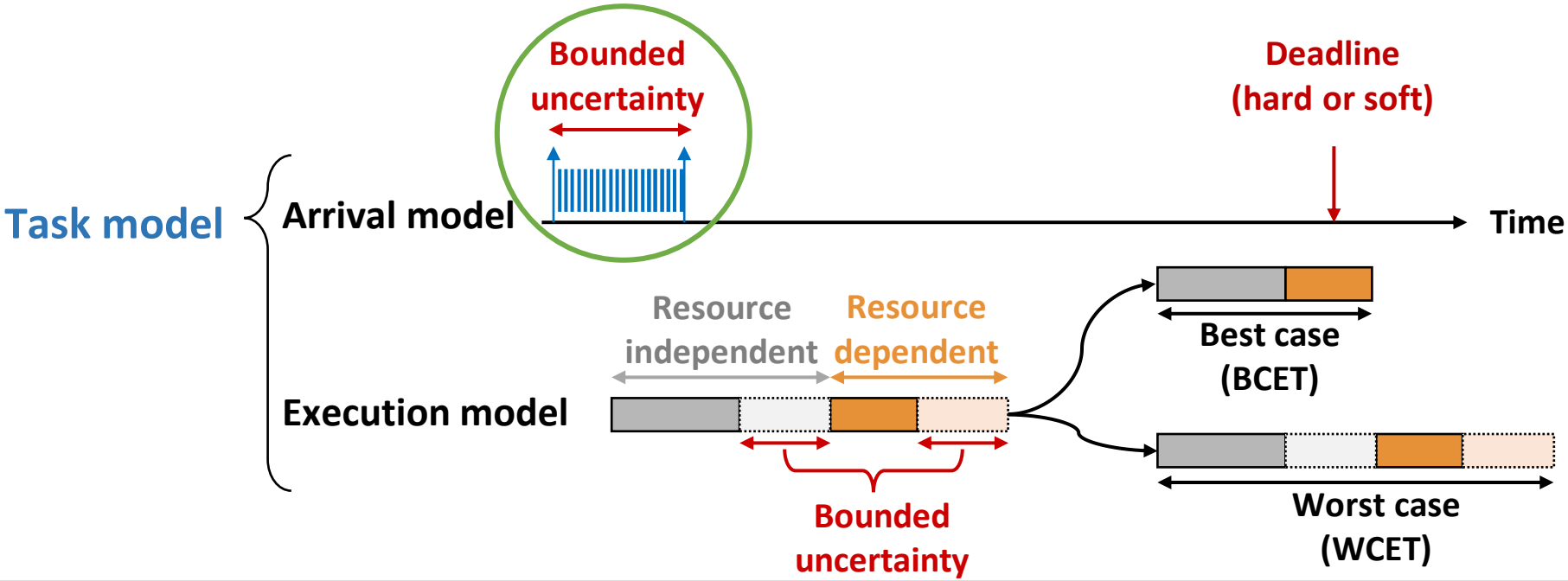
Workload model	Execution model	Platform model	Scheduler model
Tasks/Jobs that share resources (Spin locks)	Non-preemptive	Multicore (identical cores)	Global job-level fixed priority (JLFP)



The problem in a nutshell

We obtain the best-case and worst-case response time

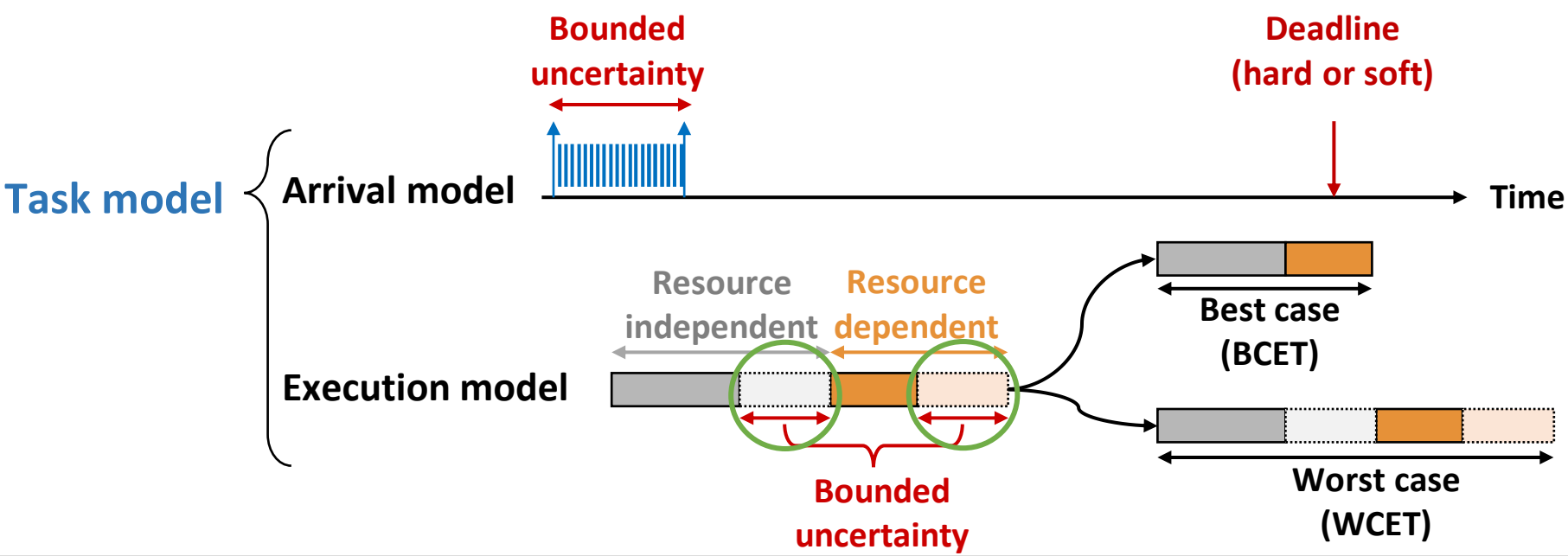
Workload model	Execution model	Platform model	Scheduler model
Tasks/Jobs that share resources (Spin locks)	Non-preemptive	Multicore (identical cores)	Global job-level fixed priority (JLFP)



The problem in a nutshell

We obtain the best-case and worst-case response time

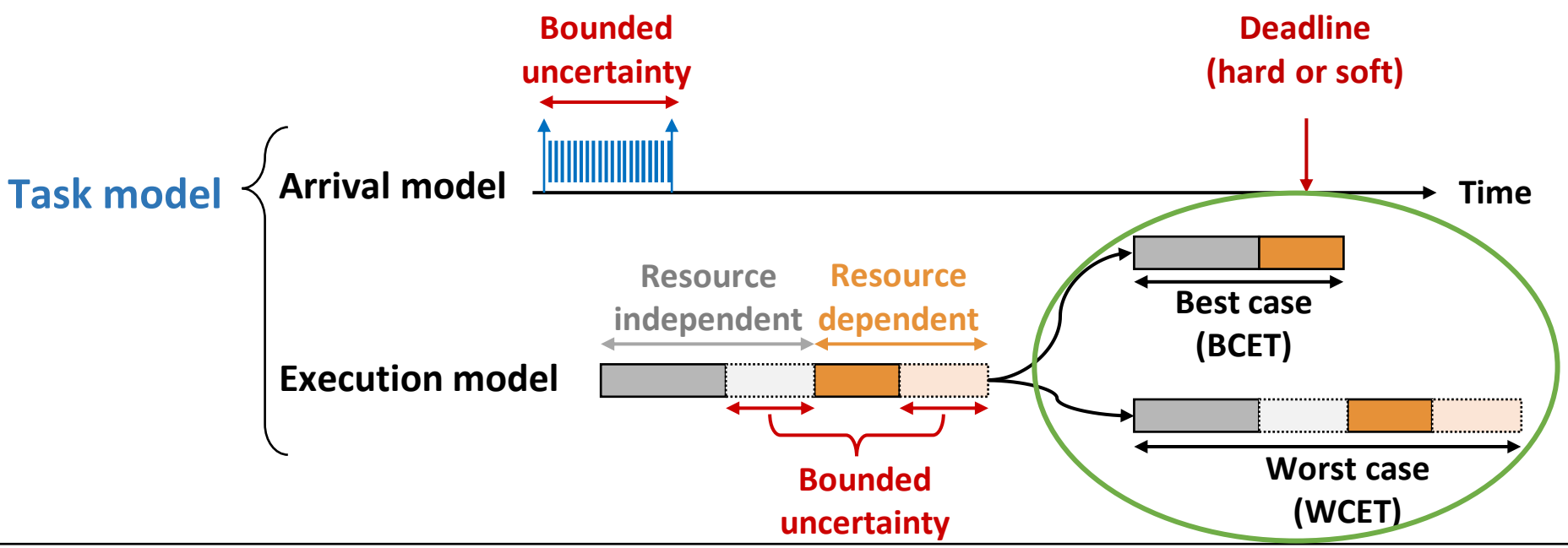
Workload model	Execution model	Platform model	Scheduler model
Tasks/Jobs that share resources (Spin locks)	Non-preemptive	Multicore (identical cores)	Global job-level fixed priority (JLFP)



The problem in a nutshell

We obtain the best-case and worst-case response time

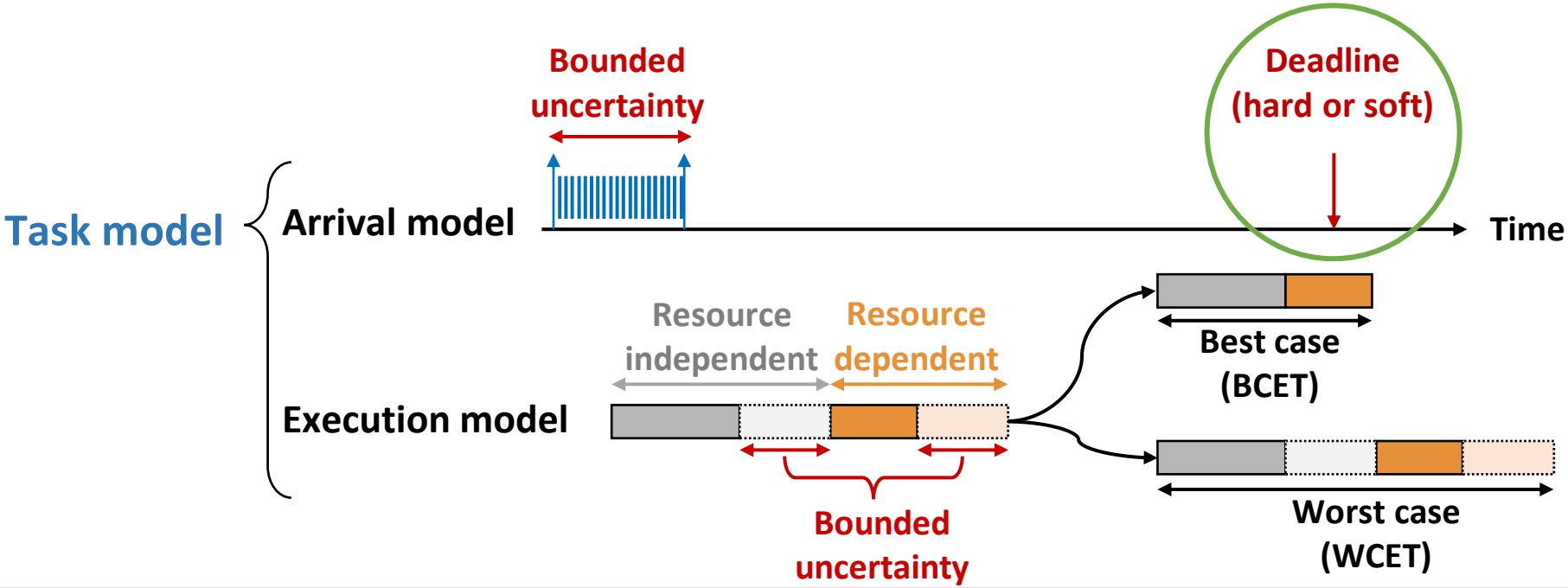
Workload model	Execution model	Platform model	Scheduler model
Tasks/Jobs that share resources (Spin locks)	Non-preemptive	Multicore (identical cores)	Global job-level fixed priority (JLFP)



The problem in a nutshell

We obtain the best-case and worst-case response time

Workload model	Execution model	Platform model	Scheduler model
Tasks/Jobs that share resources (Spin locks)	Non-preemptive	Multicore (identical cores)	Global job-level fixed priority (JLFP)



Contributions & achievements

Response-time analysis considering FIFO-ordered spin locks

Contributions & achievements

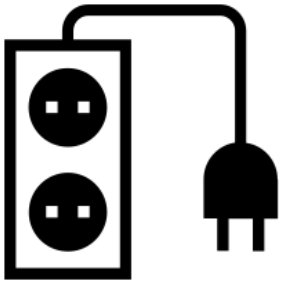
Response-time analysis considering FIFO-ordered spin locks

Response-time analysis considering priority-ordered spin locks

Contributions & achievements

Response-time analysis considering FIFO-ordered spin locks

Response-time analysis considering priority-ordered spin locks



- **Support for multi-unit resources & k-exclusion protocols**
- **Additional techniques to improve scalability**

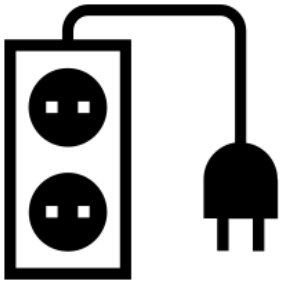
Contributions & achievements

Response-time analysis considering **FIFO-ordered spin locks**

→ Accepted and presented at **CAPITAL**

→ Full paper submitted to **RTSS**

Response-time analysis considering **priority-ordered spin locks**



- Support for **multi-unit resources & k-exclusion protocols**
- **Additional techniques** to improve scalability

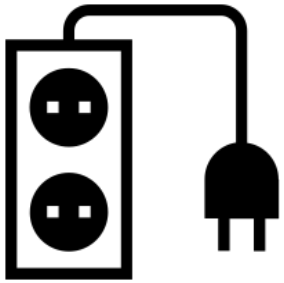
Contributions & achievements

Response-time analysis considering **FIFO-ordered spin locks**

→ Accepted and presented at **CAPITAL**

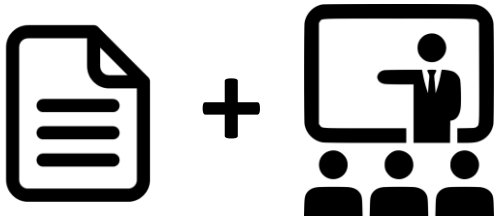
→ Full paper submitted to **RTSS**

Response-time analysis considering **priority-ordered spin locks**



- Support for **multi-unit resources & k-exclusion protocols**
- **Additional techniques** to improve scalability

Plan for future:



Schedulability analyses

Schedulability analyses

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

Schedulability analyses

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

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

Schedulability analyses

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

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

Response-time analysis using schedule abstraction

[ECRTS'19] Nasri, M., Nelissen, G. and Brandenburg, B.B., 2019, July. Response-time analysis of limited-preemptive parallel DAG tasks under global scheduling. In 31st Conference on Real-Time Systems (pp. 21-1).*arXiv:1909.09600*.

State of the art: schedule-abstraction-based analyses

[ECRTS'18]

Multiprocessor

Sufficient

Independent non-preemptive jobs/tasks

Global work-conserving job-level fixed-priority scheduling (JLFP)

[ECRTS'19]

Support for DAG tasks with precedence constraints

[ECRTS'18] Nasri, M., Nelissen, G. and Brandenburg, B.B., 2018, July. A response-time analysis for non-preemptive job sets under global scheduling. In *30th Euromicro Conference on Real-Time Systems* (pp. 9-1).

[ECRTS'19] Nasri, M., Nelissen, G. and Brandenburg, B.B., 2019, July. Response-time analysis of limited-preemptive parallel DAG tasks under global scheduling. In 31st Conference on Real-Time Systems (pp. 21-1).

State of the art: schedule-abstraction-based analyses

[ECRTS'18]

Multiprocessor

Sufficient

Independent non-preemptive jobs/tasks

Global work-conserving job-level fixed-priority scheduling (JLFP)

[ECRTS'19]

Support for DAG tasks with precedence constraints

[This work]

Support for shared resources with FIFO and priority-based spin locks

[ECRTS'18] Nasri, M., Nelissen, G. and Brandenburg, B.B., 2018, July. A response-time analysis for non-preemptive job sets under global scheduling. In *30th Euromicro Conference on Real-Time Systems* (pp. 9-1).

[ECRTS'19] Nasri, M., Nelissen, G. and Brandenburg, B.B., 2019, July. Response-time analysis of limited-preemptive parallel DAG tasks under global scheduling. In 31st Conference on Real-Time Systems (pp. 21-1).

Schedule-abstraction analysis

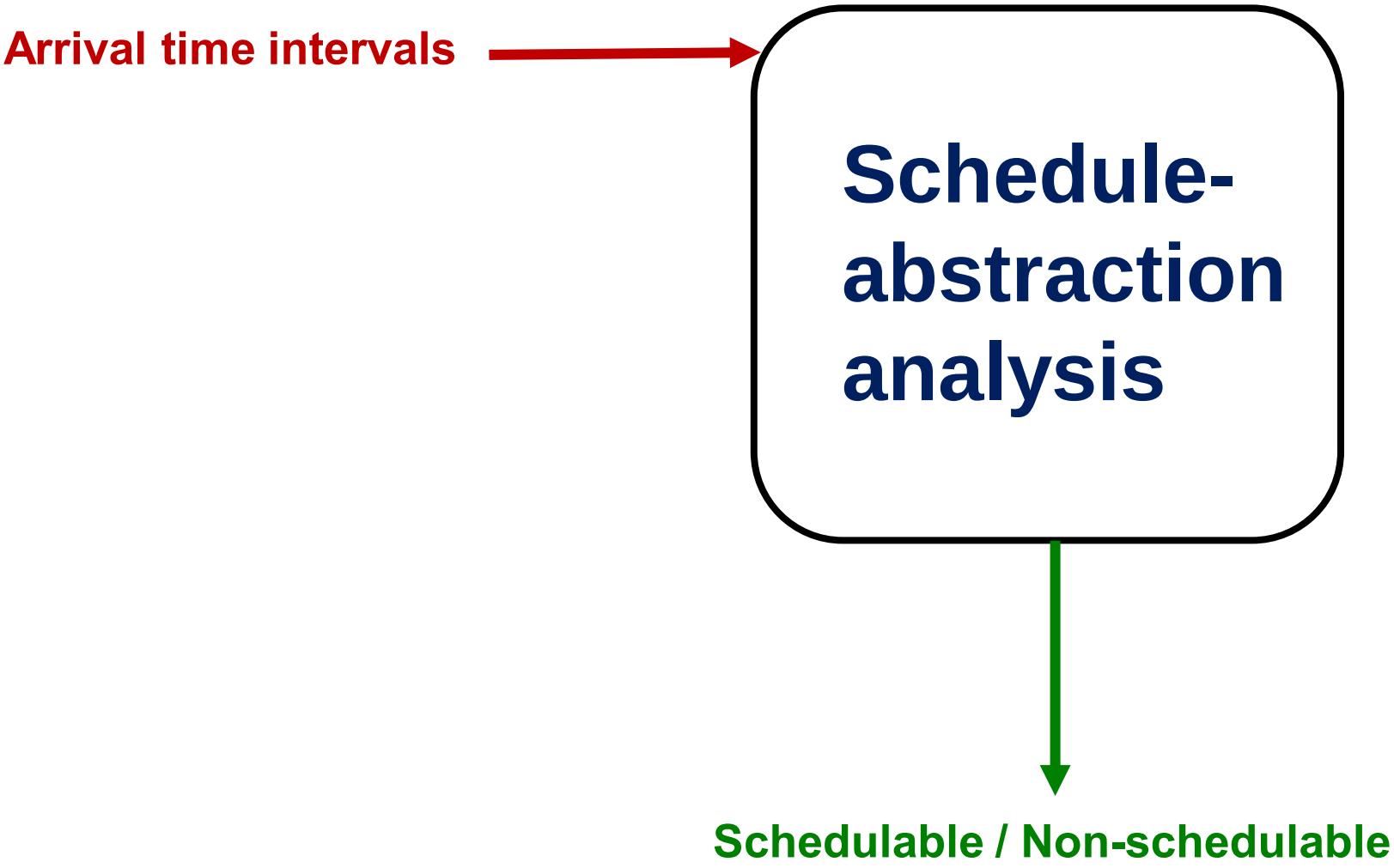
Schedule-abstraction graph input and output

Schedule-abstraction analysis

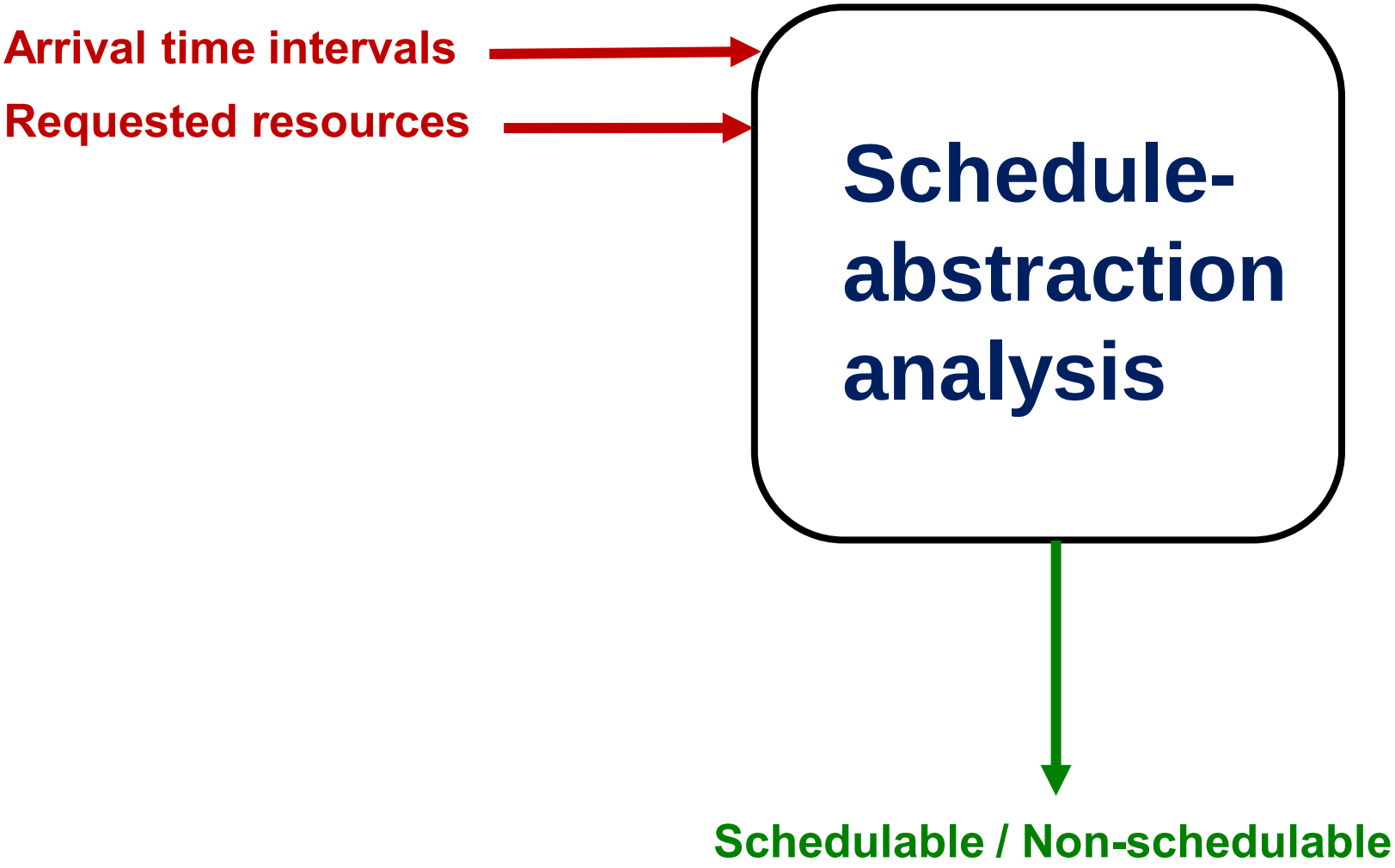
↓

Schedulable / Non-schedulable

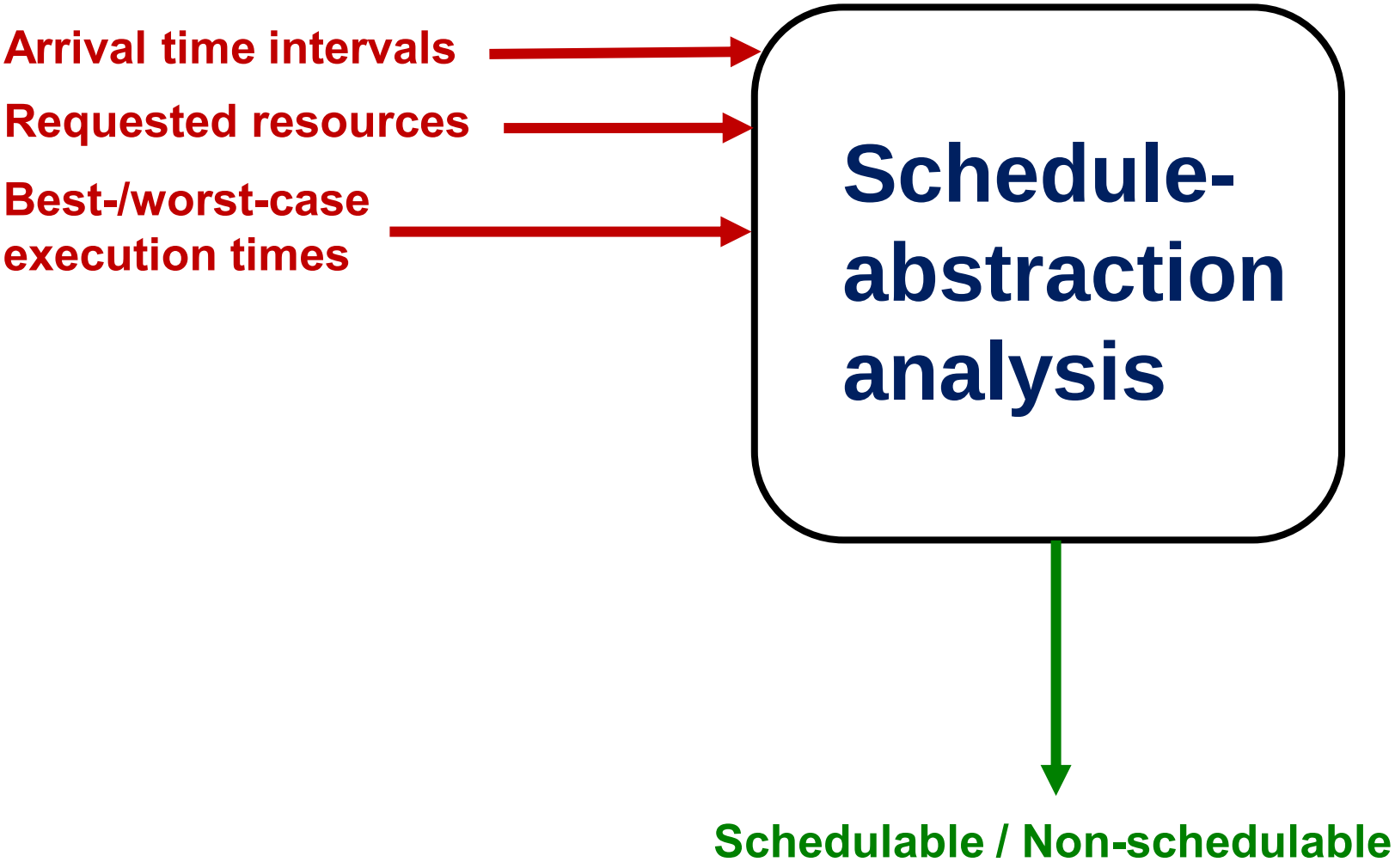
Schedule-abstraction graph input and output



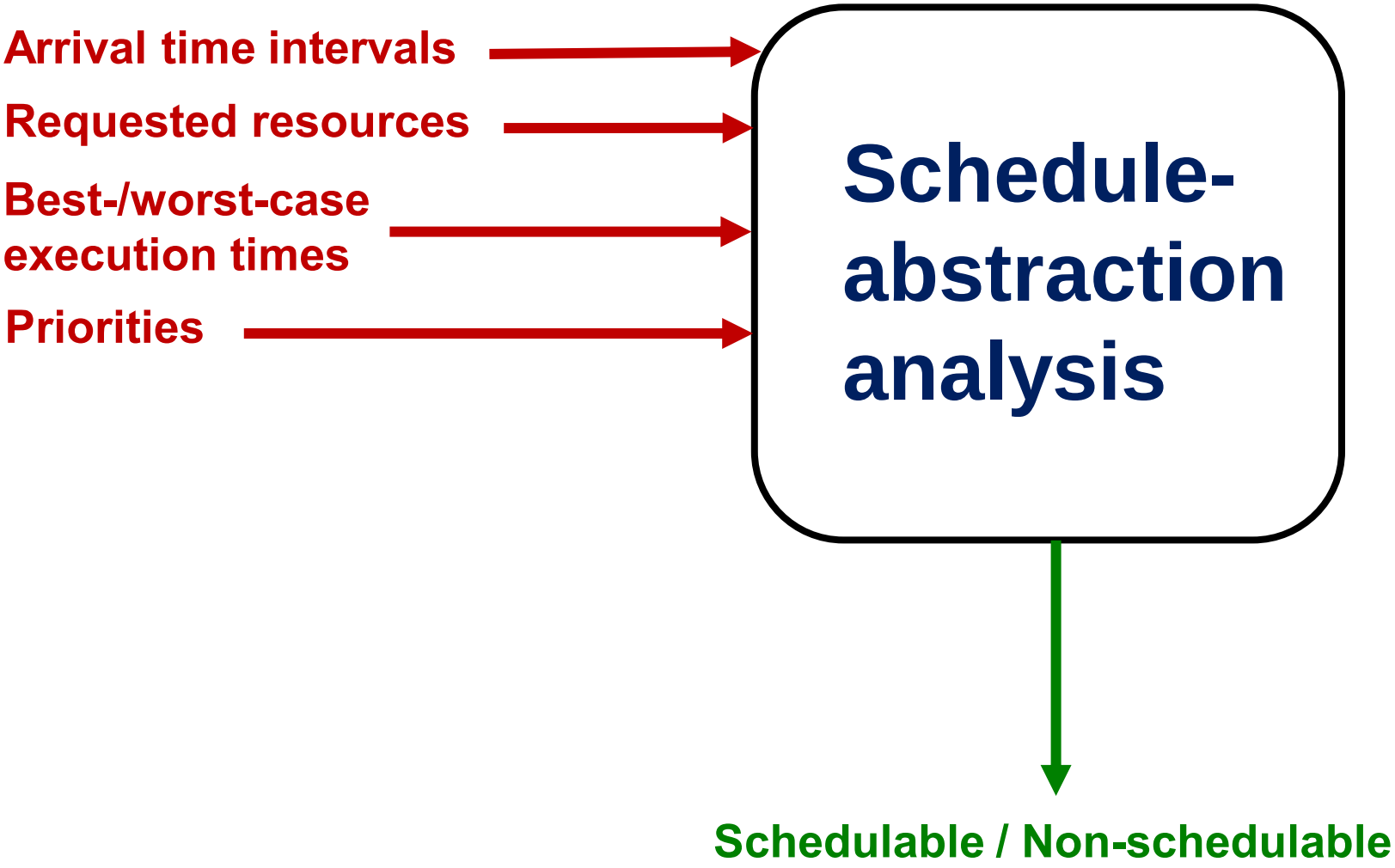
Schedule-abstraction graph input and output



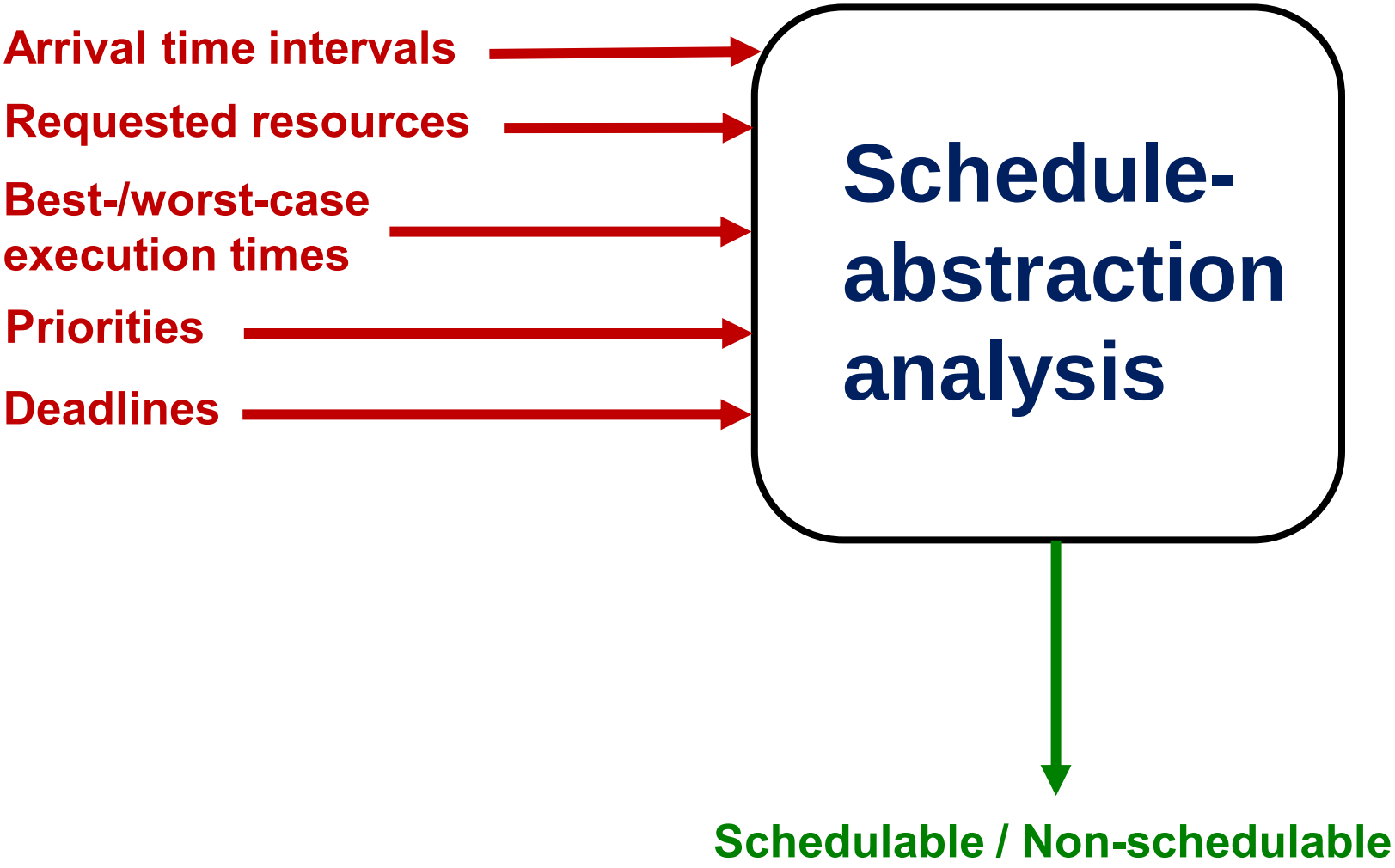
Schedule-abstraction graph input and output



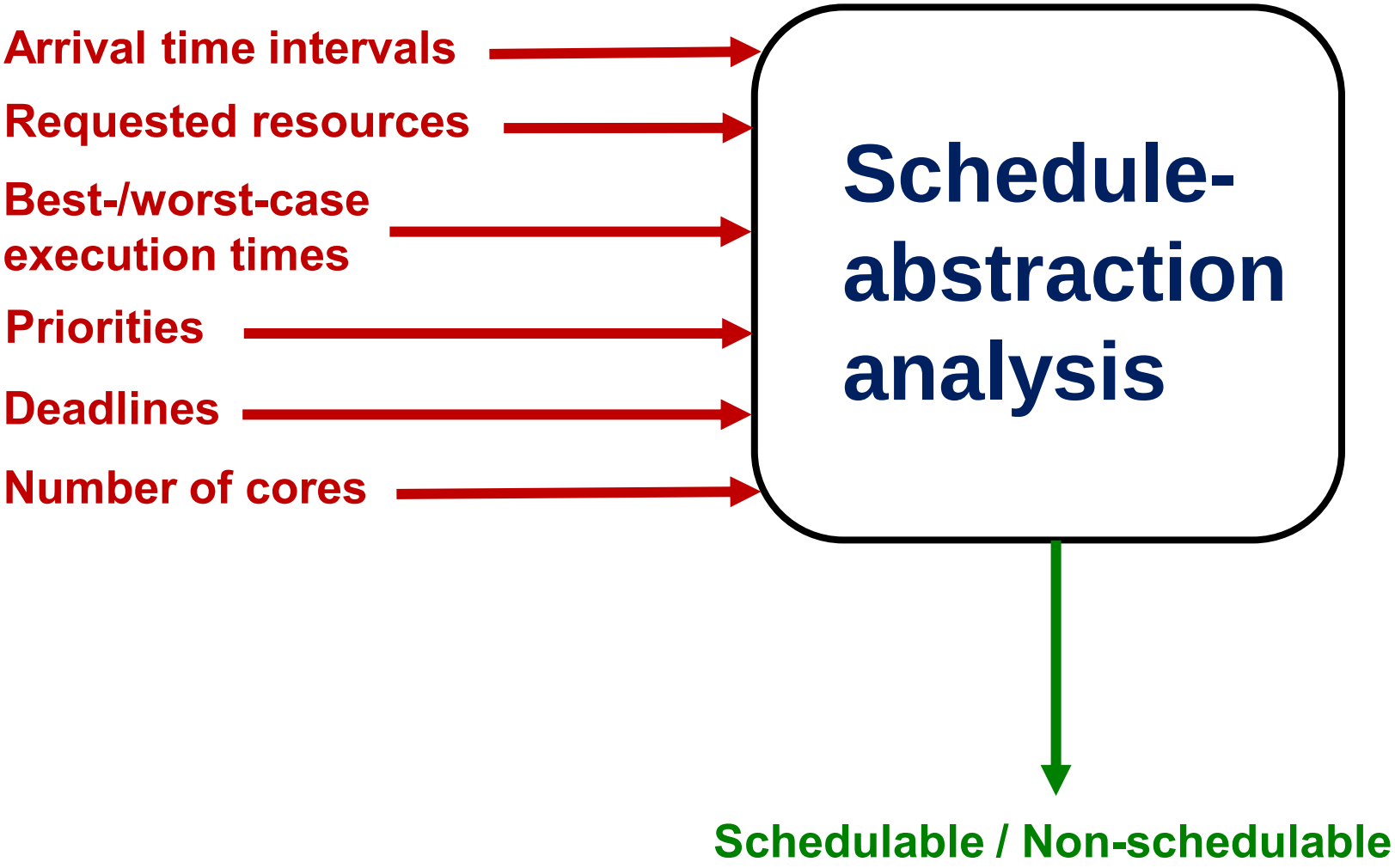
Schedule-abstraction graph input and output



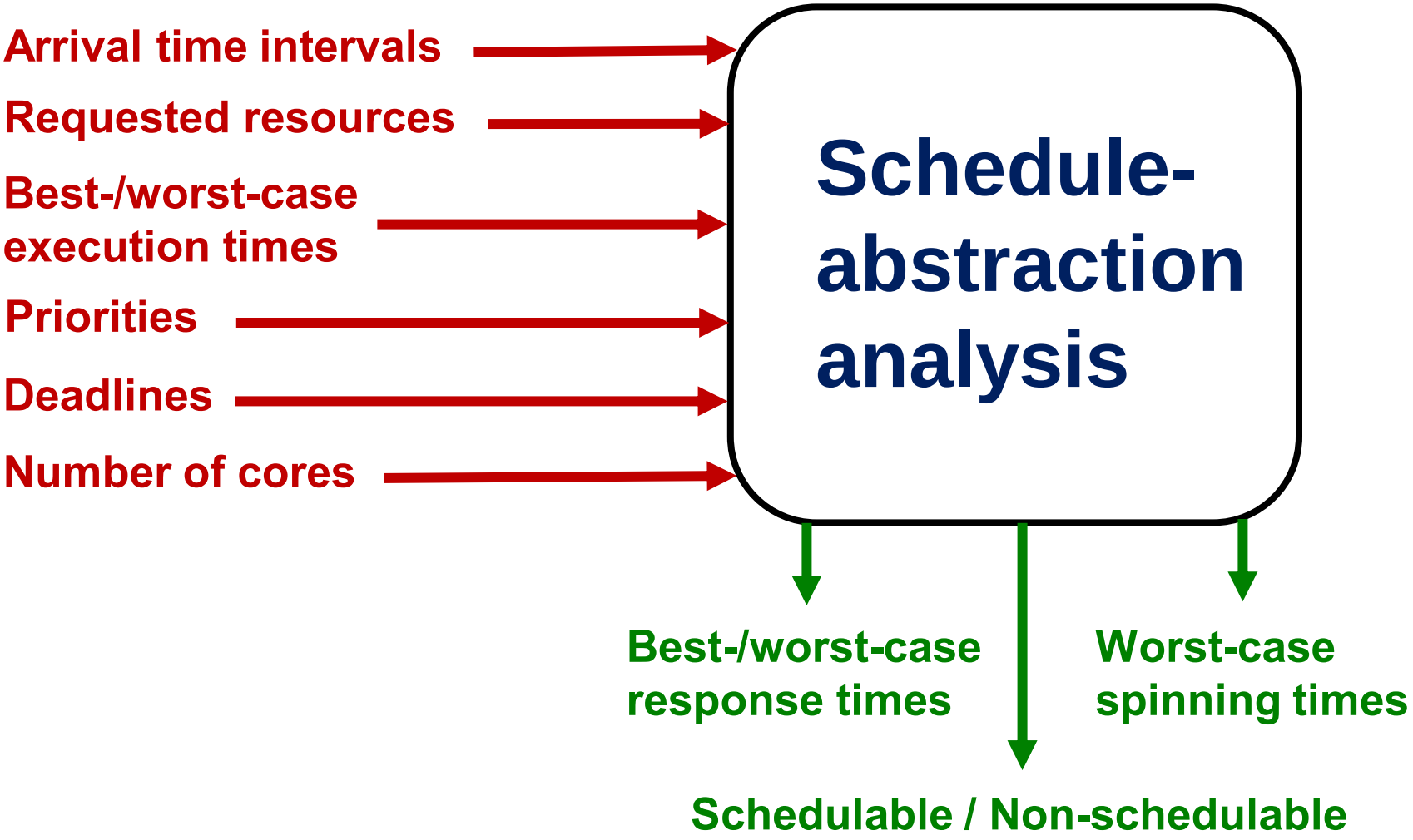
Schedule-abstraction graph input and output



Schedule-abstraction graph input and output

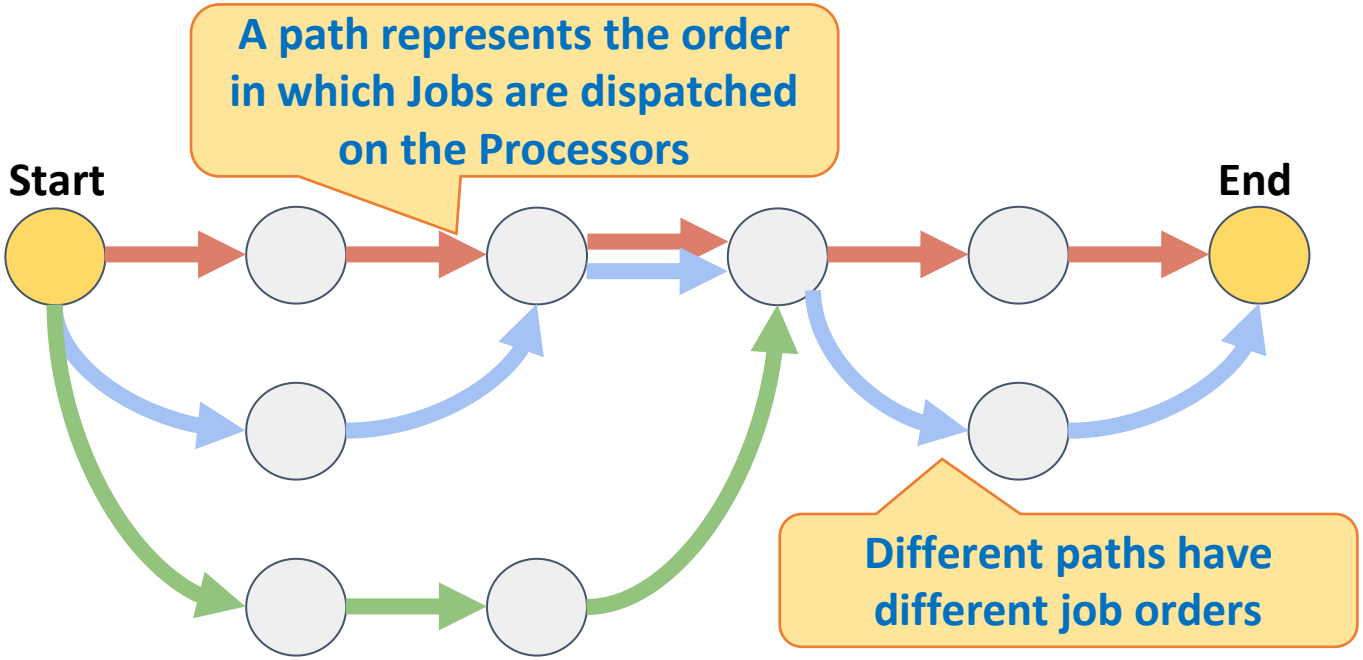


Schedule-abstraction graph input and output



Schedule-abstraction graph

A path reflects a sequence of job-dispatch decisions

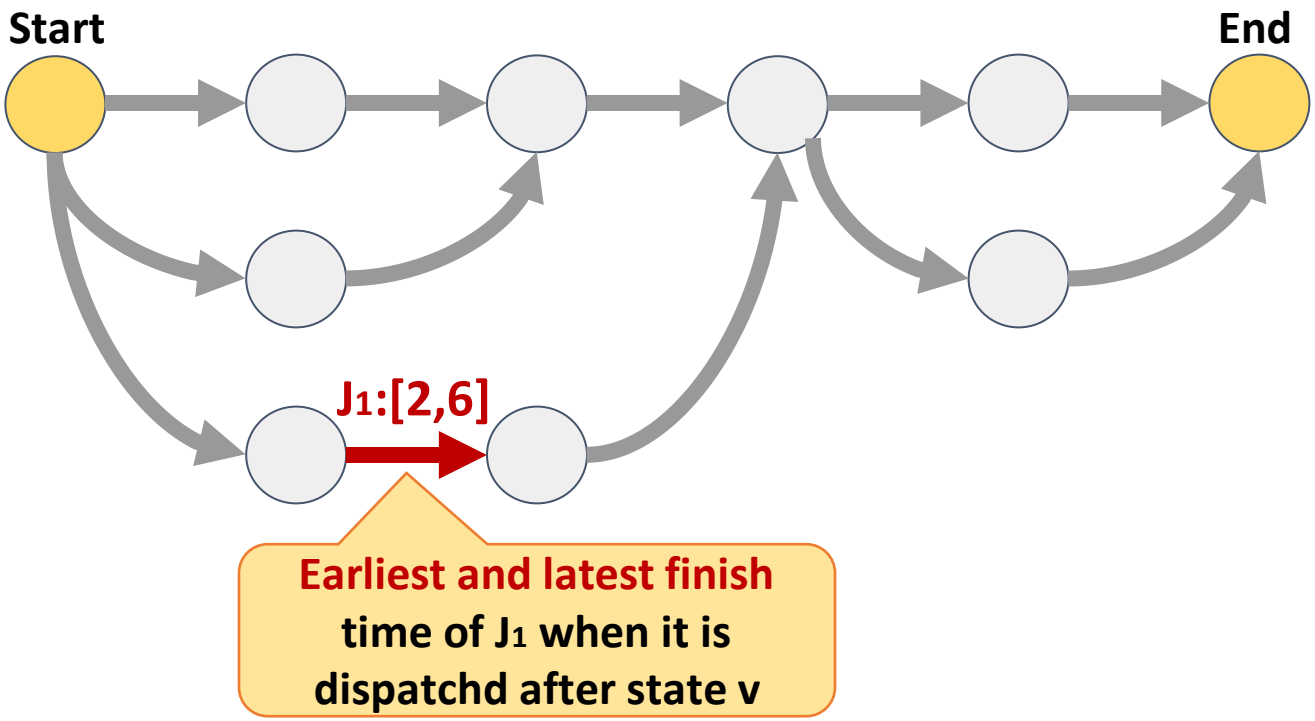


[ECRTS'19] Nasri, M., Nelissen, G. and Brandenburg, B.B., 2019, July. Response-time analysis of limited-preemptive parallel DAG tasks under global scheduling. In 31st Conference on Real-Time Systems (pp. 21-1).

Schedule-abstraction graph

A path reflects a sequence of job-dispatch decisions

A vertex abstracts a system state and an edge represents a dispatched job



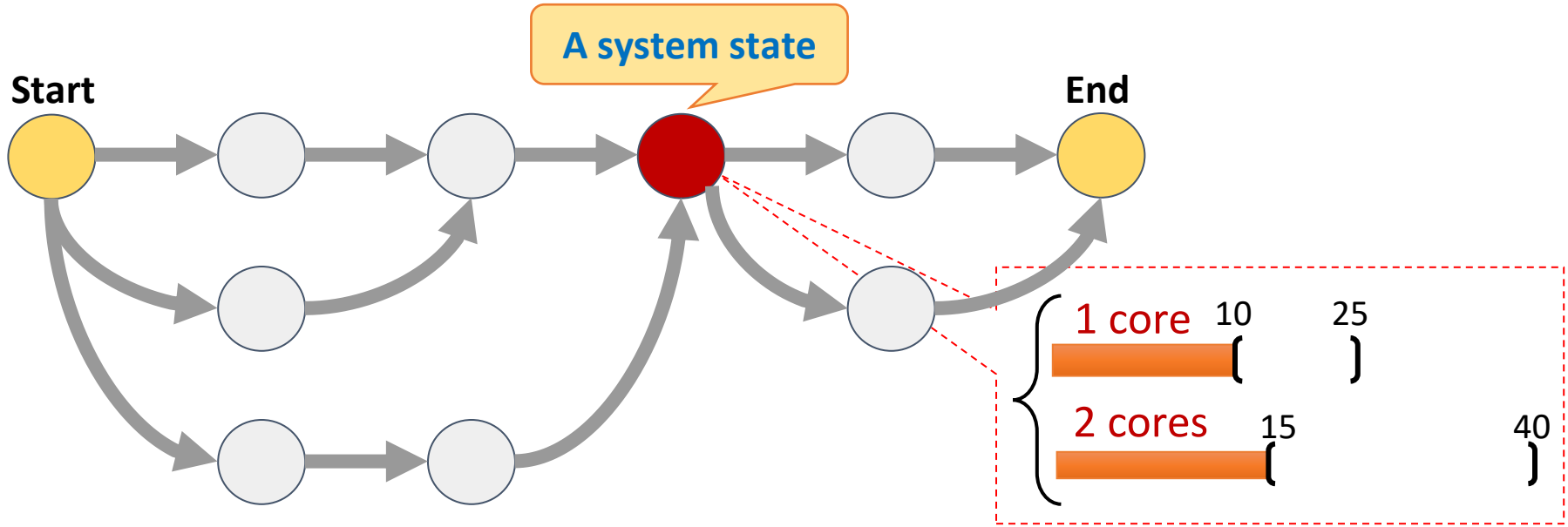
[ECRTS'19] Nasri, M., Nelissen, G. and Brandenburg, B.B., 2019, July. Response-time analysis of limited-preemptive parallel DAG tasks under global scheduling. In 31st Conference on Real-Time Systems (pp. 21-1).

Schedule-abstraction graph

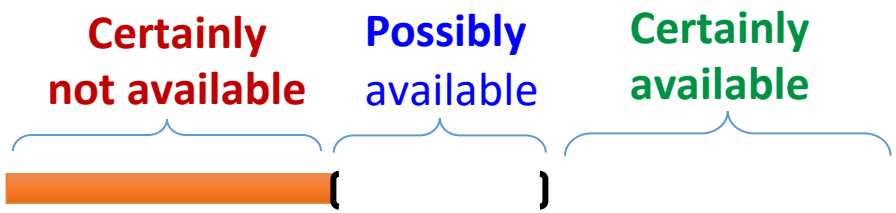
A path reflects a sequence of job-dispatch decisions

A vertex abstracts a system state and an edge represents a dispatched job

A state is labeled with the finish-time interval of any path reaching the state



Interpretation of an availability interval:



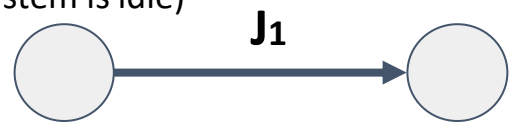
Building the schedule-abstraction graph

Building the graph
(a breadth-first method)

Repeat until every path includes all jobs

1. **Find the shortest path**
2. **For each not-yet-dispatched job that can be dispatched after the path:**
   - **2.1. Expand** (add a new vertex)

Initial State
(System is idle)



[ECRTS'19] Nasri, M., Nelissen, G. and Brandenburg, B.B., 2019, July. Response-time analysis of limited-preemptive parallel DAG tasks under global scheduling. In 31st Conference on Real-Time Systems (pp. 21-1).

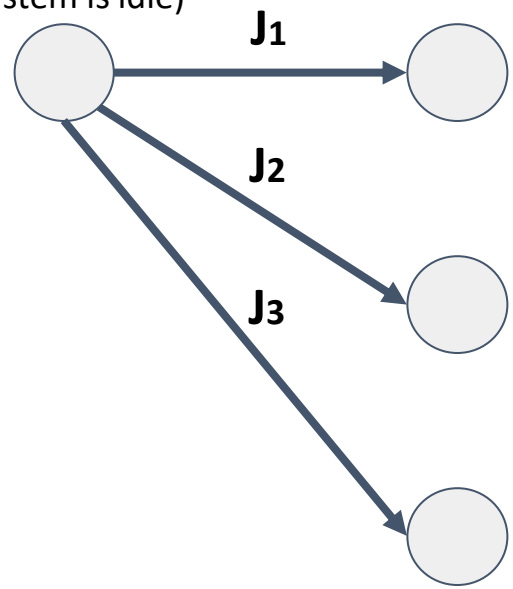
Building the schedule-abstraction graph

Building the graph
(a breadth-first method)

Repeat until every path includes all jobs

1. **Find the shortest path**
2. **For each not-yet-dispatched job that can be dispatched after the path:**
   - **2.1. Expand** (add a new vertex)

Initial State
(System is idle)



[ECRTS'19] Nasri, M., Nelissen, G. and Brandenburg, B.B., 2019, July. Response-time analysis of limited-preemptive parallel DAG tasks under global scheduling. In 31st Conference on Real-Time Systems (pp. 21-1).

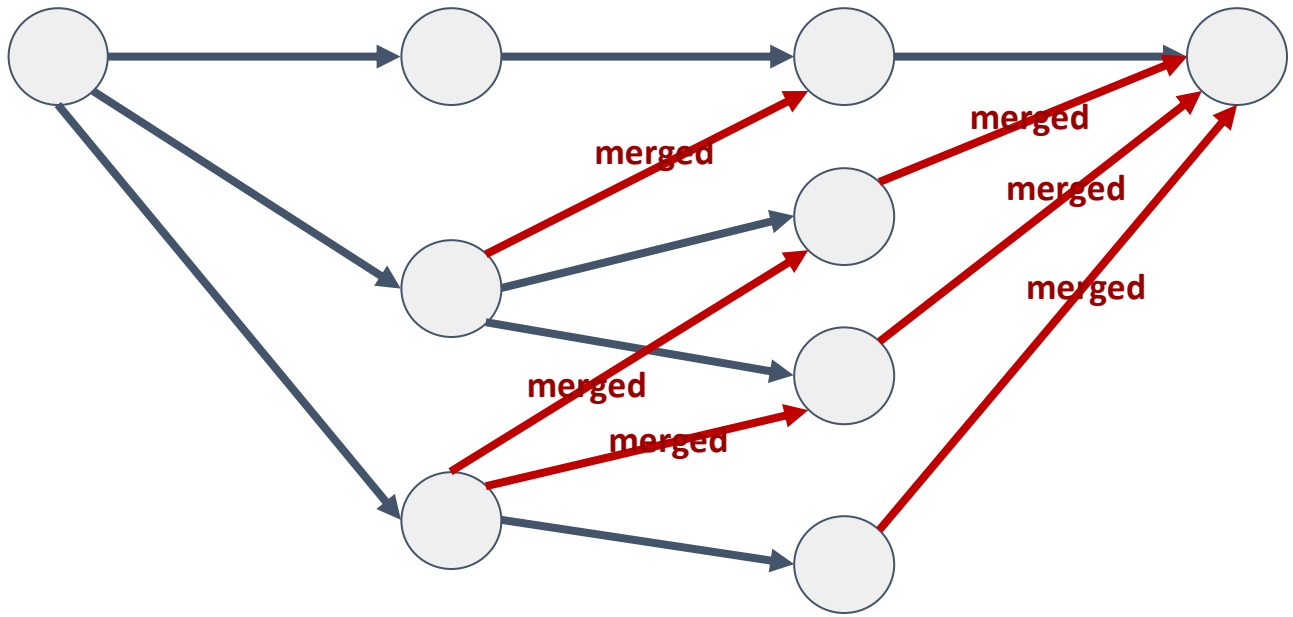
Building the schedule-abstraction graph

Building the graph
(a breadth-first method)

Repeat until every path includes all jobs

1. **Find the shortest path**
2. **For each not-yet-dispatched job that can be dispatched after the path:**
   - 2.1. **Expand** (add a new vertex)
   - 2.2. **Merge** (if possible, merge the new vertex with an existing vertex)

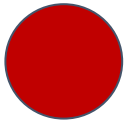
Initial State
(System is idle)



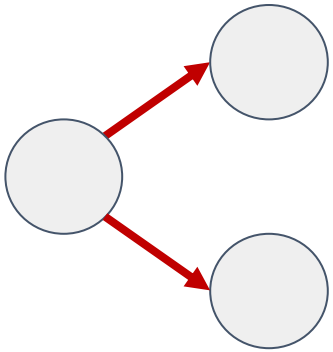
[ECRTS'19] Nasri, M., Nelissen, G. and Brandenburg, B.B., 2019, July. Response-time analysis of limited-preemptive parallel DAG tasks under global scheduling. In 31st Conference on Real-Time Systems (pp. 21-1).

Designing a schedule-abstraction-based analysis

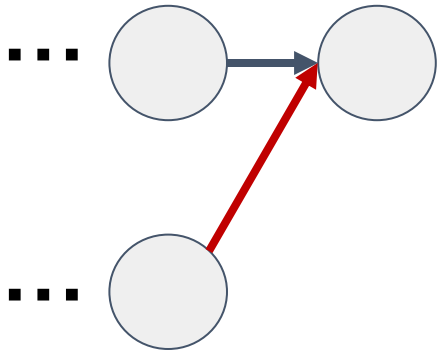
- **State representation**



- **Expansion rules**



- **Merge rules**



Agenda

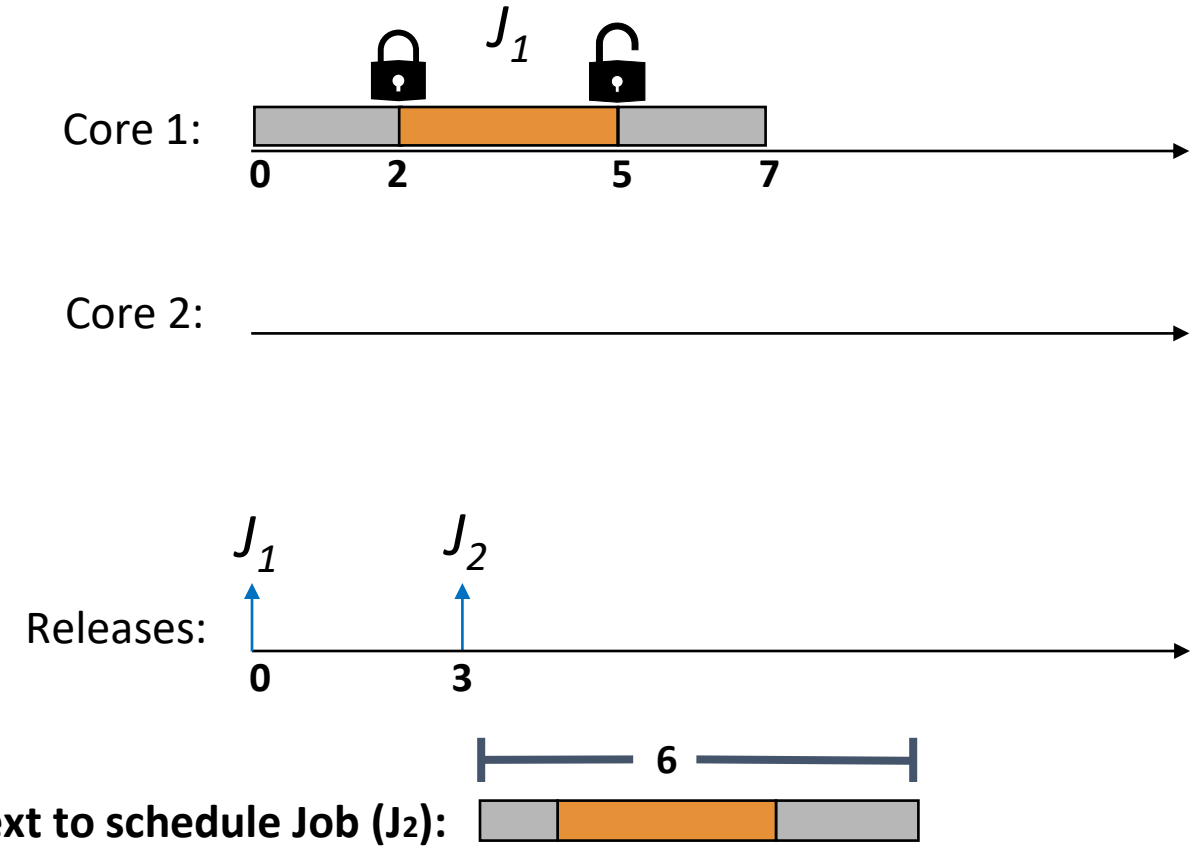
To do's:

- [ ] Design a system state representation
- [ ] Design expansion and merge rules


Schedule-abstraction graph
Challenges ?
Schedule-abstraction graph considering shared resource accesses

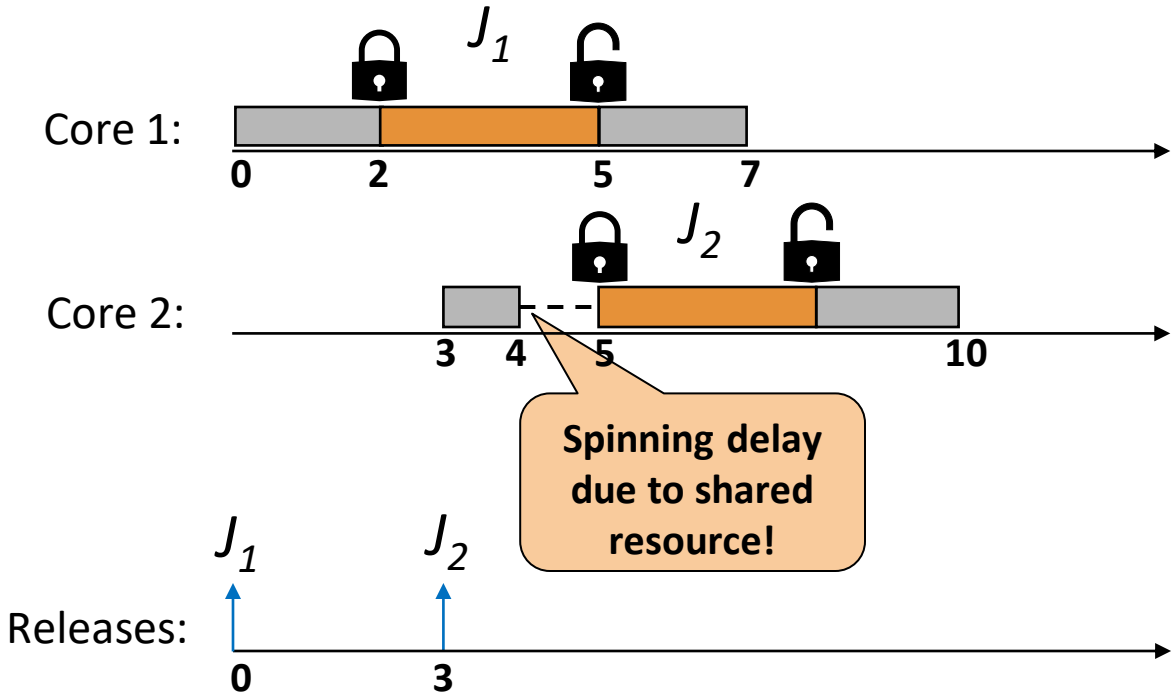
Handling shared resource

An example schedule of 2 jobs & 1 shared resource:



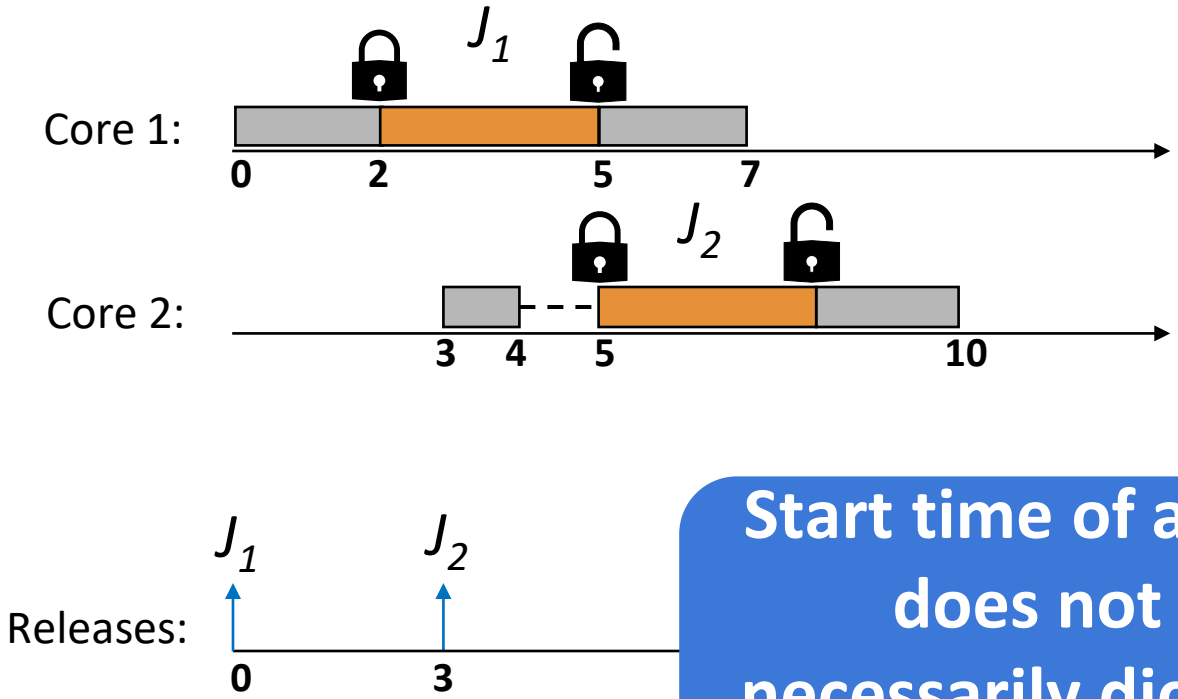
Handling shared resource

An example schedule of 2 jobs & 1 shared resource:



Handling shared resource

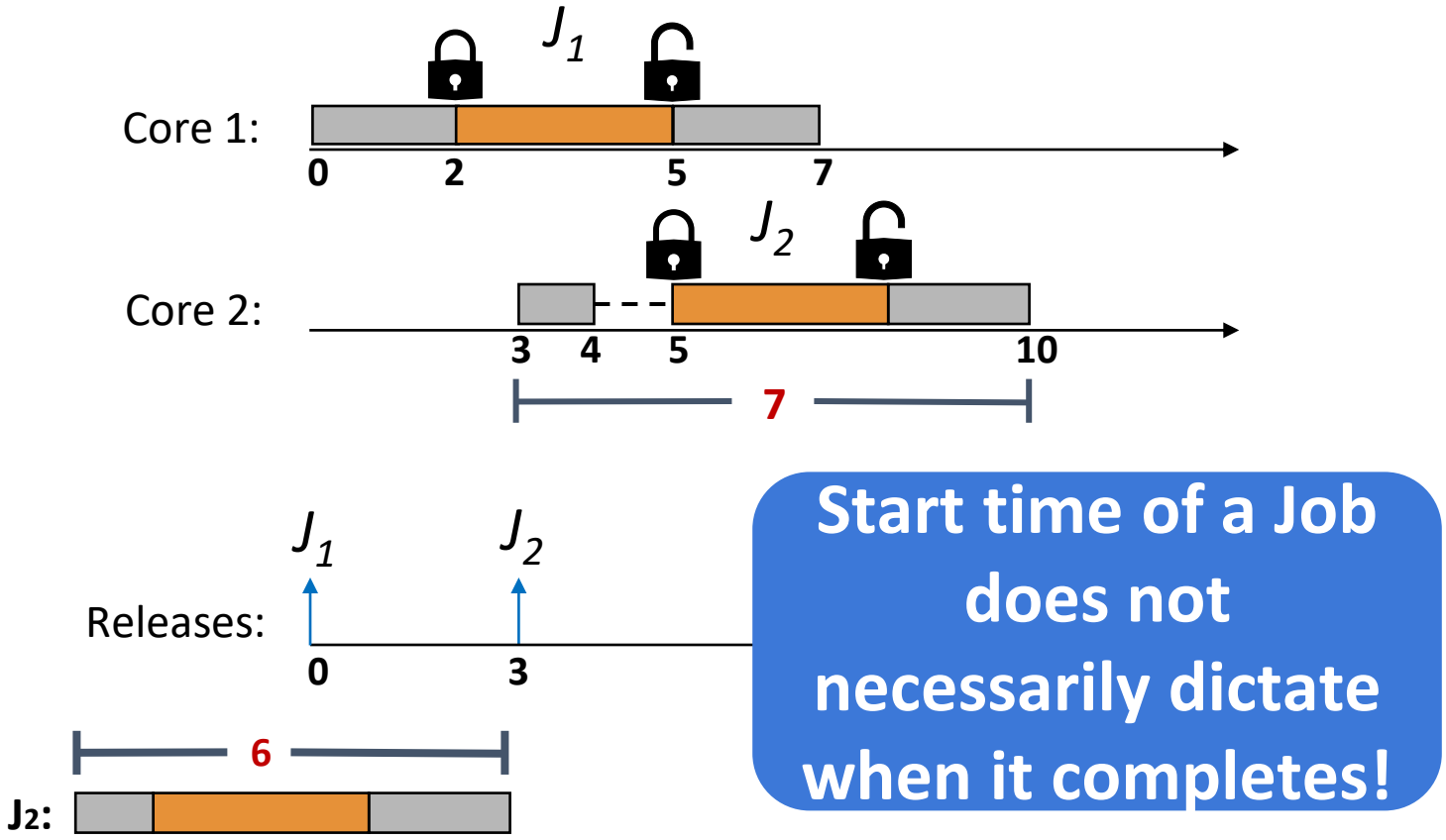
An example schedule of 2 jobs & 1 shared resource:



Start time of a Job does not necessarily dictate when it completes!

Handling shared resource

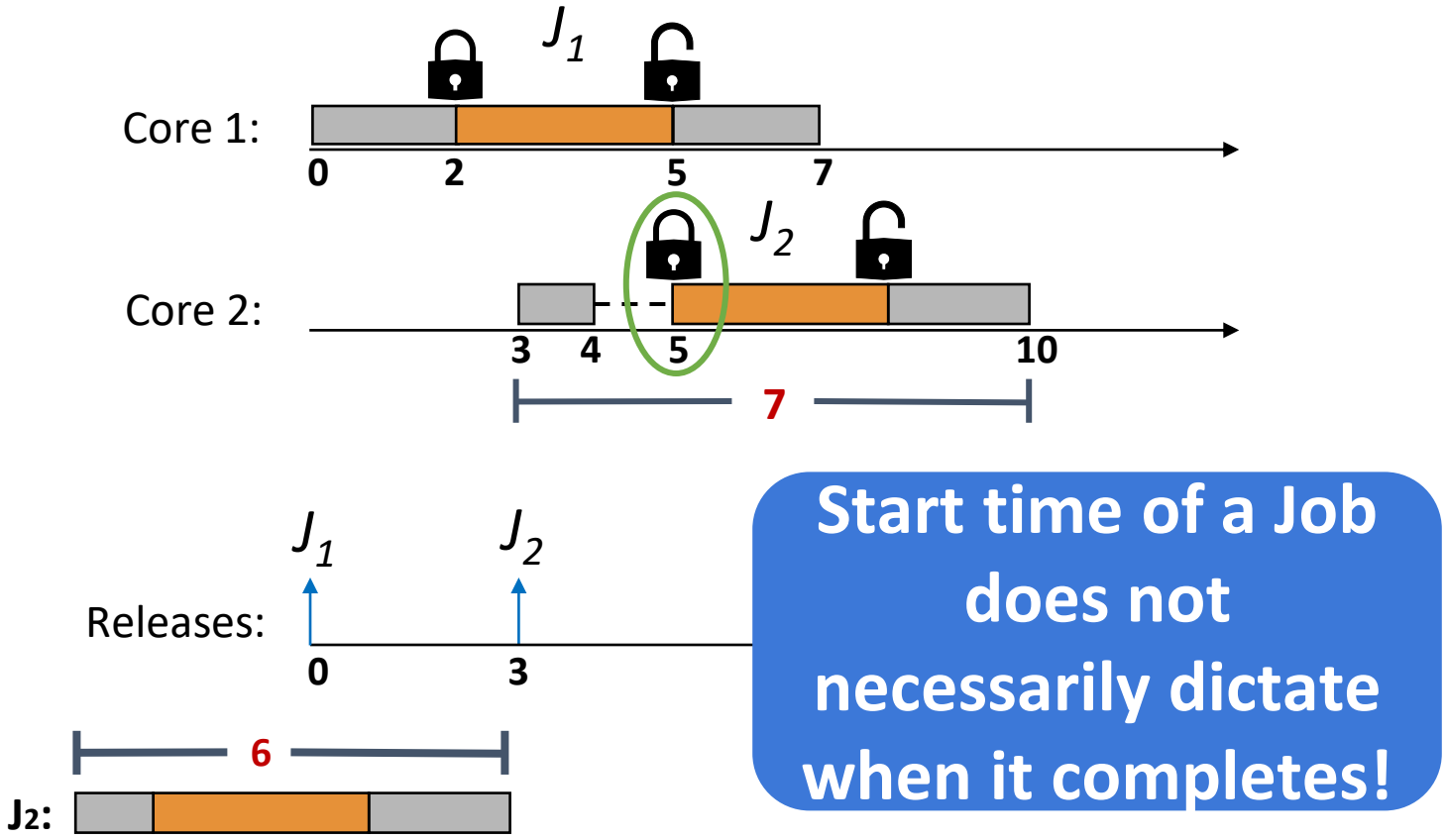
An example schedule of 2 jobs & 1 shared resource:



Start time of a Job does not necessarily dictate when it completes!

Handling shared resource

An example schedule of 2 jobs & 1 shared resource:

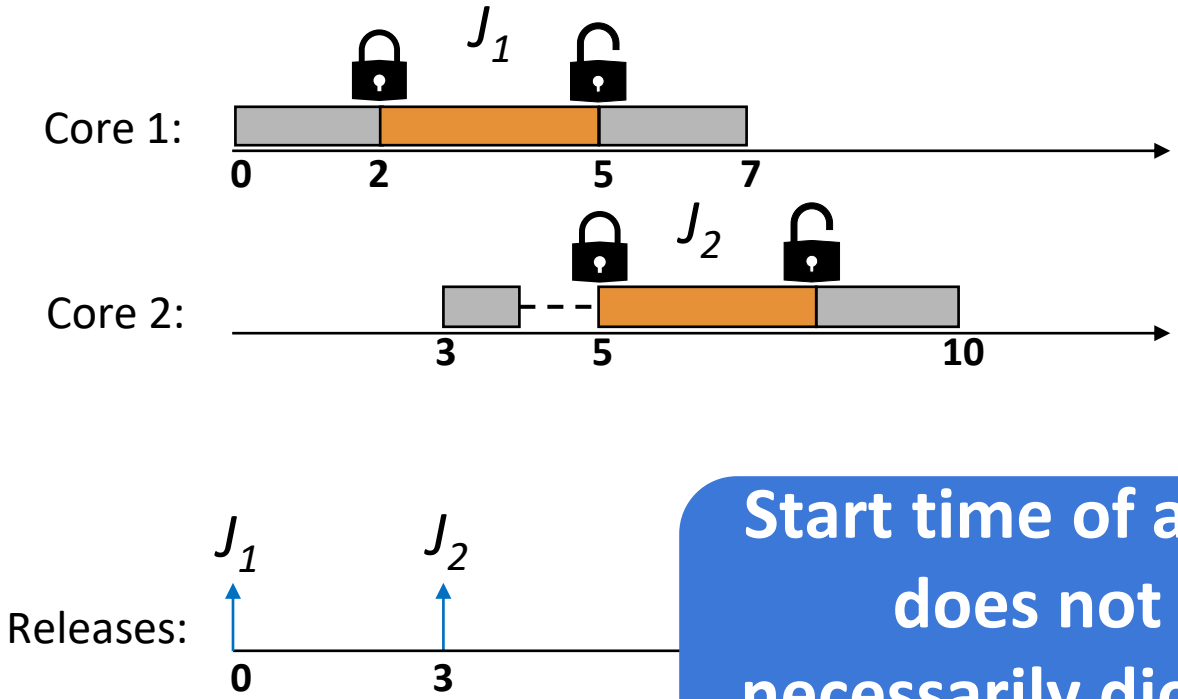


Start time of a Job does not necessarily dictate when it completes!

Handling shared resource

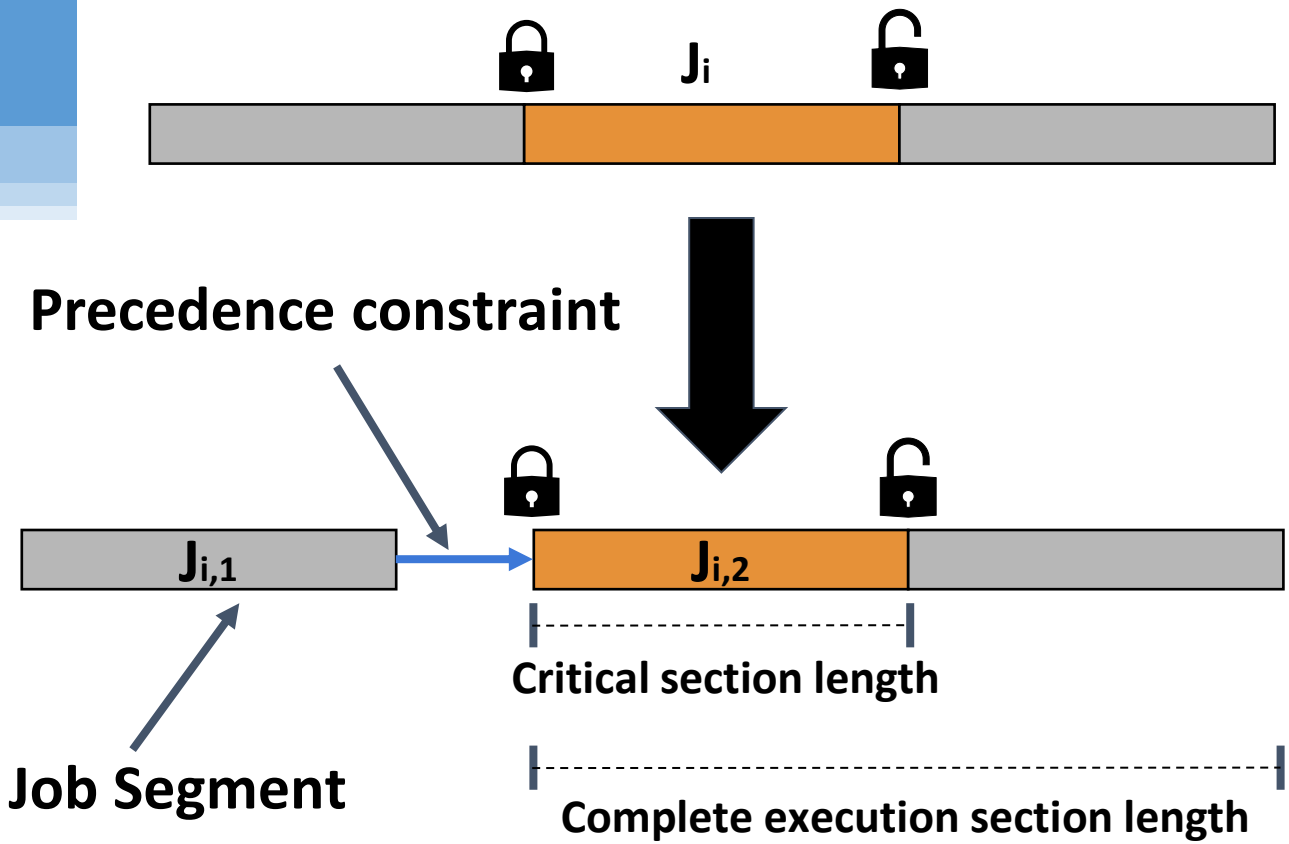
Keep track of shared resource availability!

An example schedule of 2 jobs & 1 shared resource:

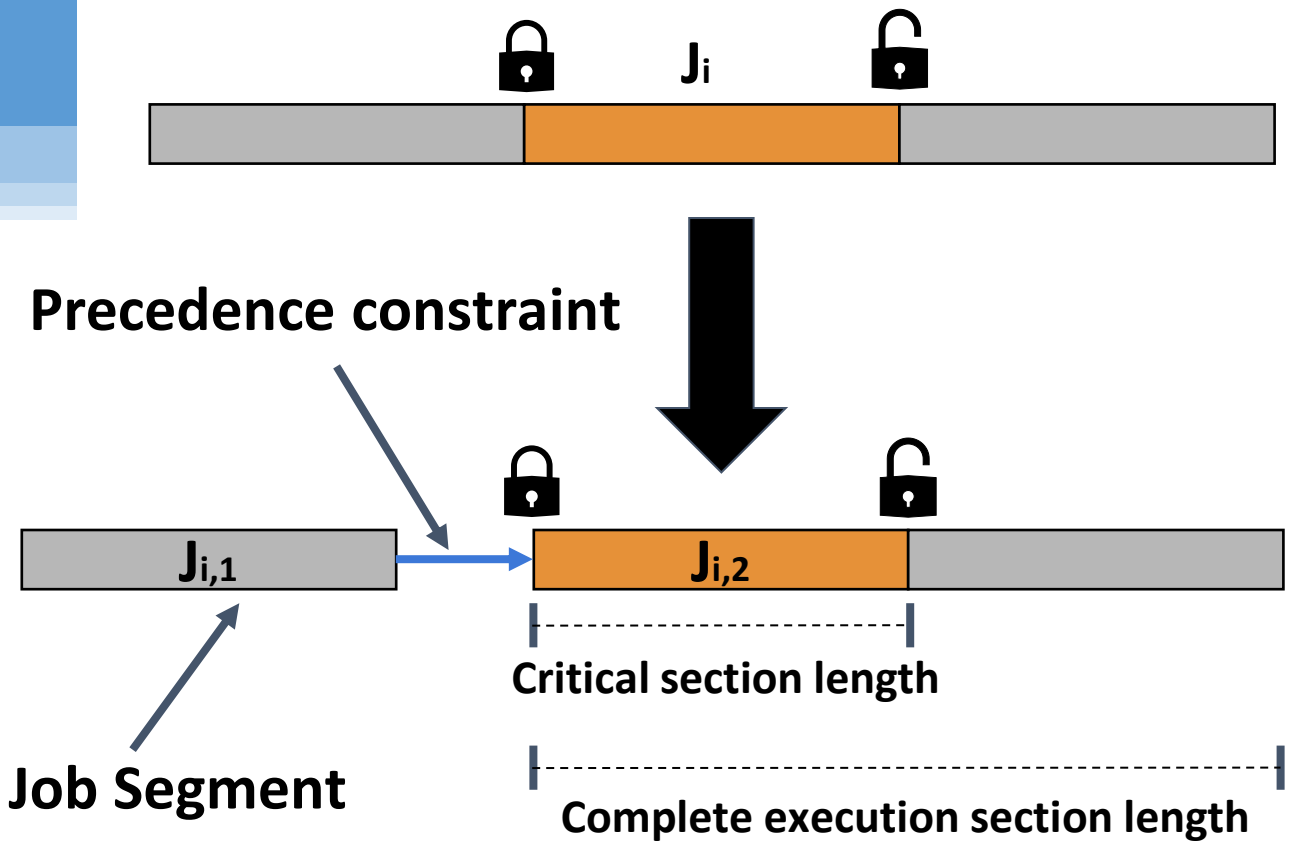


Start time of a Job does not necessarily dictate when it completes!

Split a job into segments



Split a job into segments

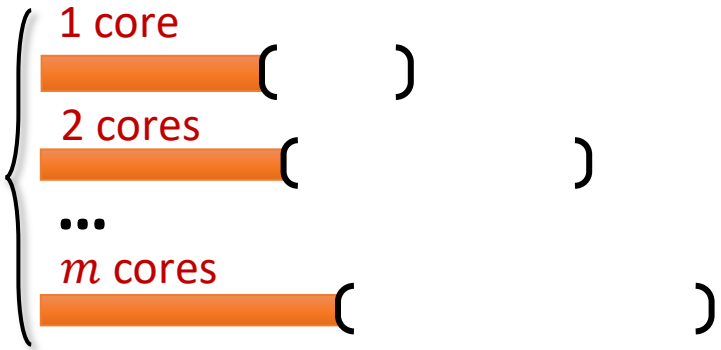


→ Schedule Abstraction Analysis operates on segment level

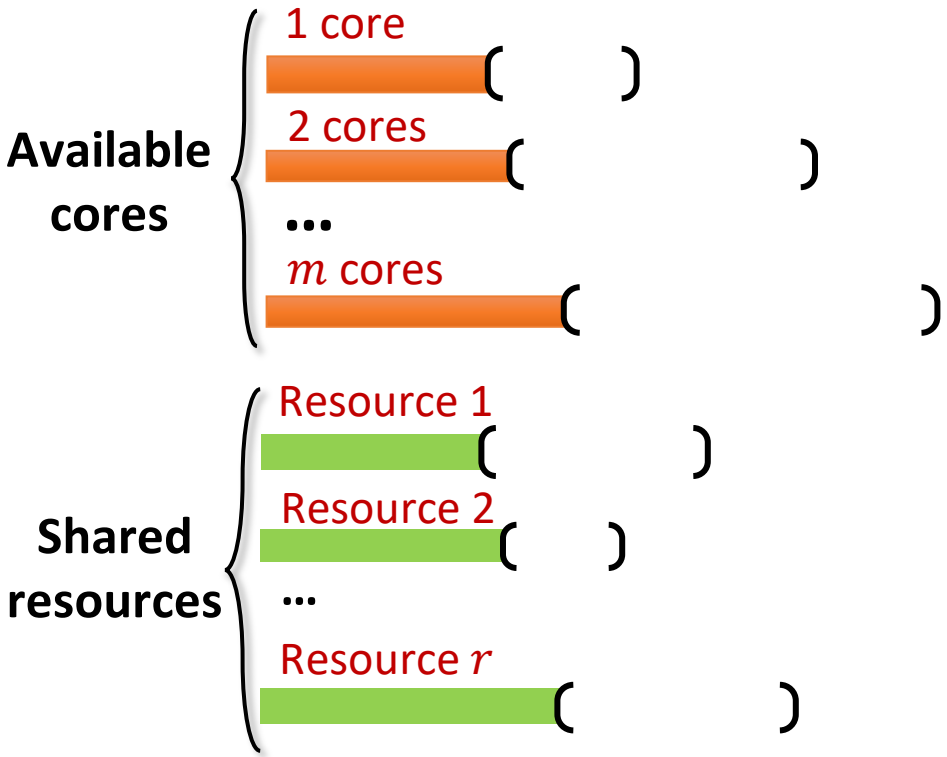
System state abstraction

[ECRTS'19]

Available cores

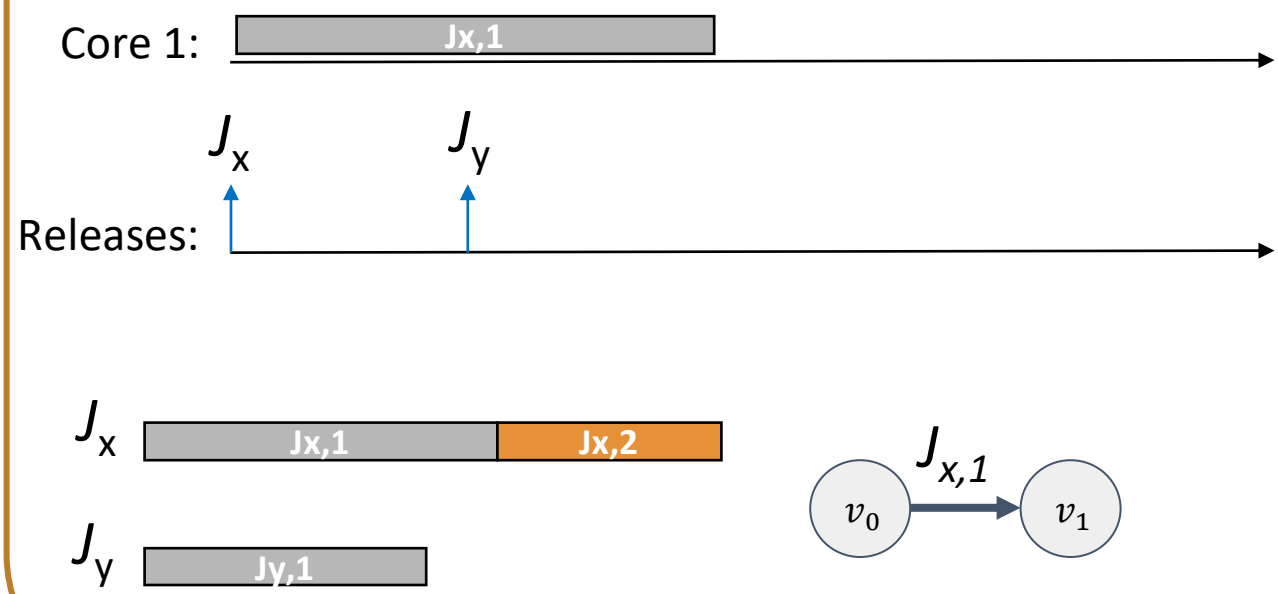


[This work]



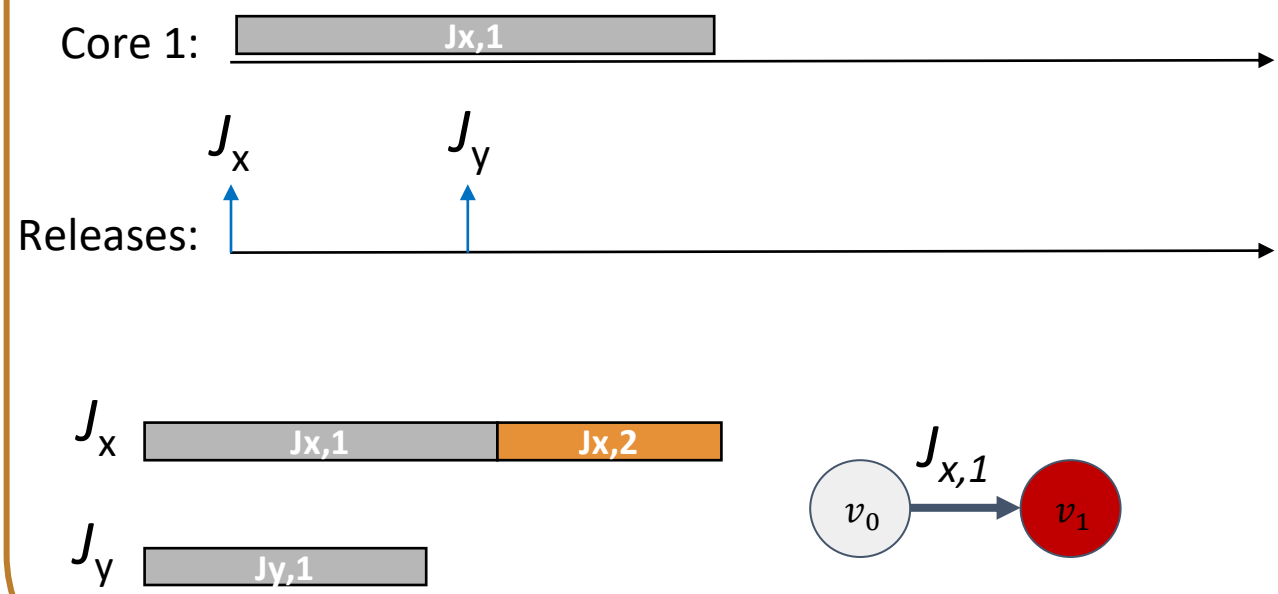
How to preserve non-preemptive execution?

An example schedule:



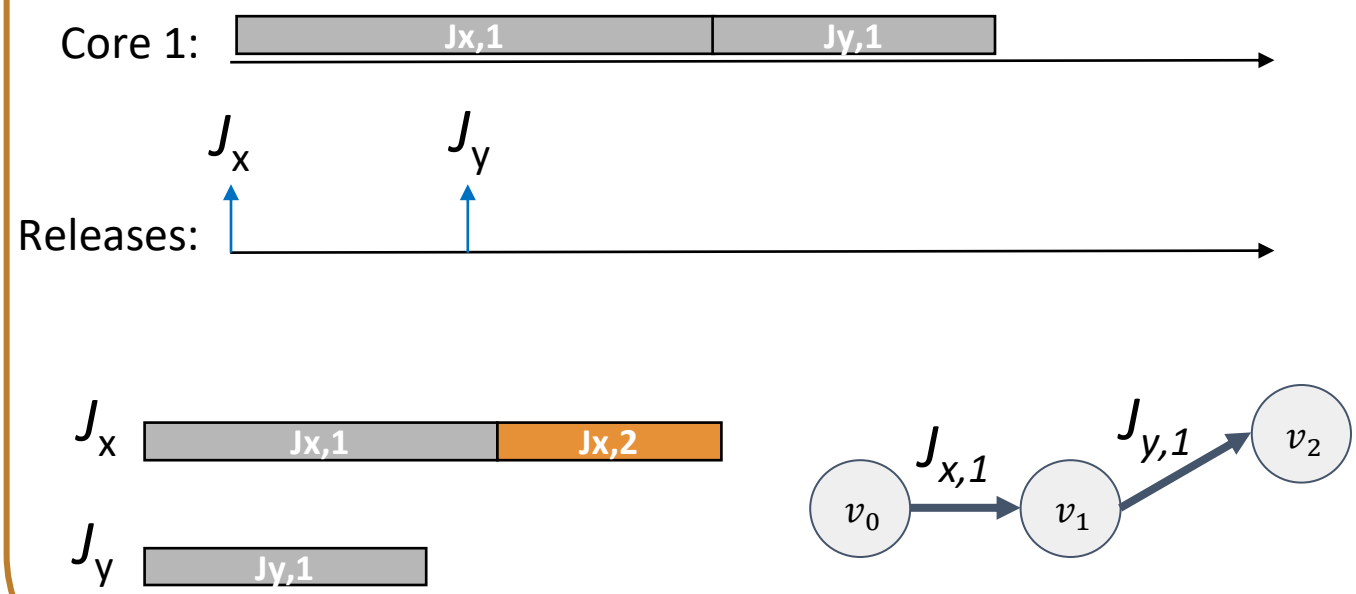
How to preserve non-preemptive execution?

An example schedule:



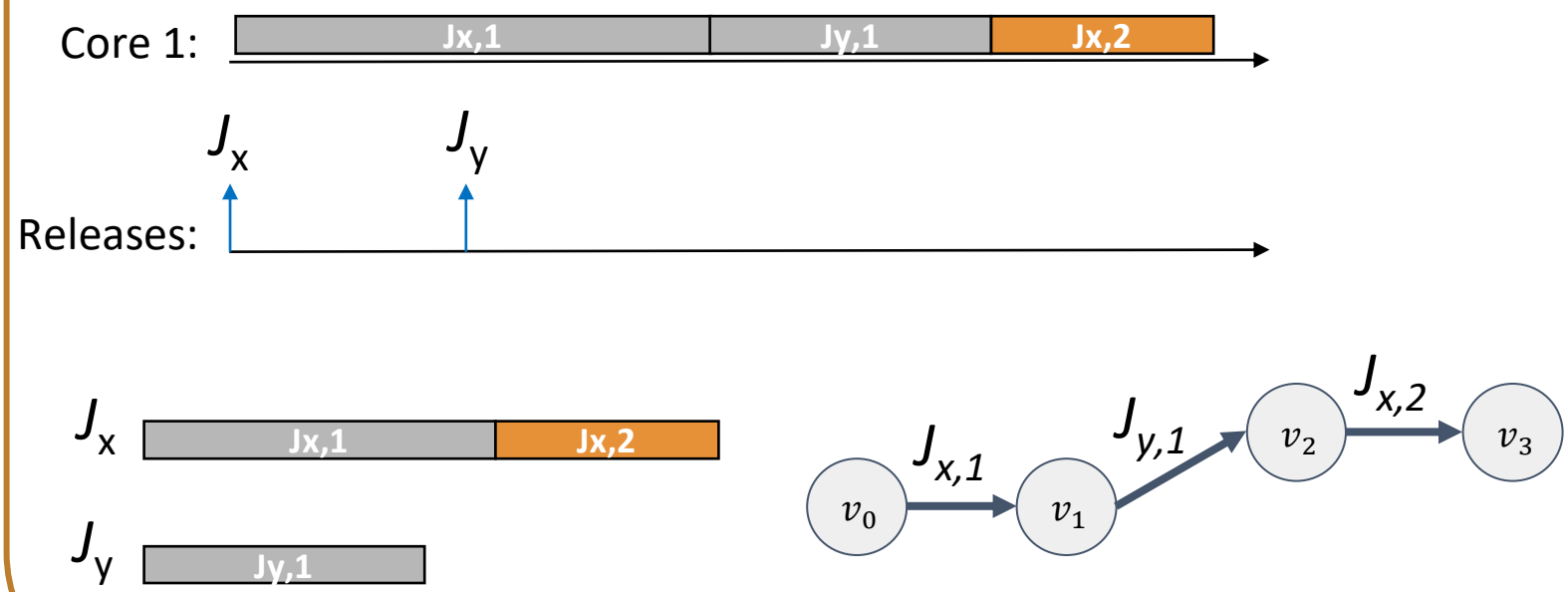
How to preserve non-preemptive execution?

An example schedule:



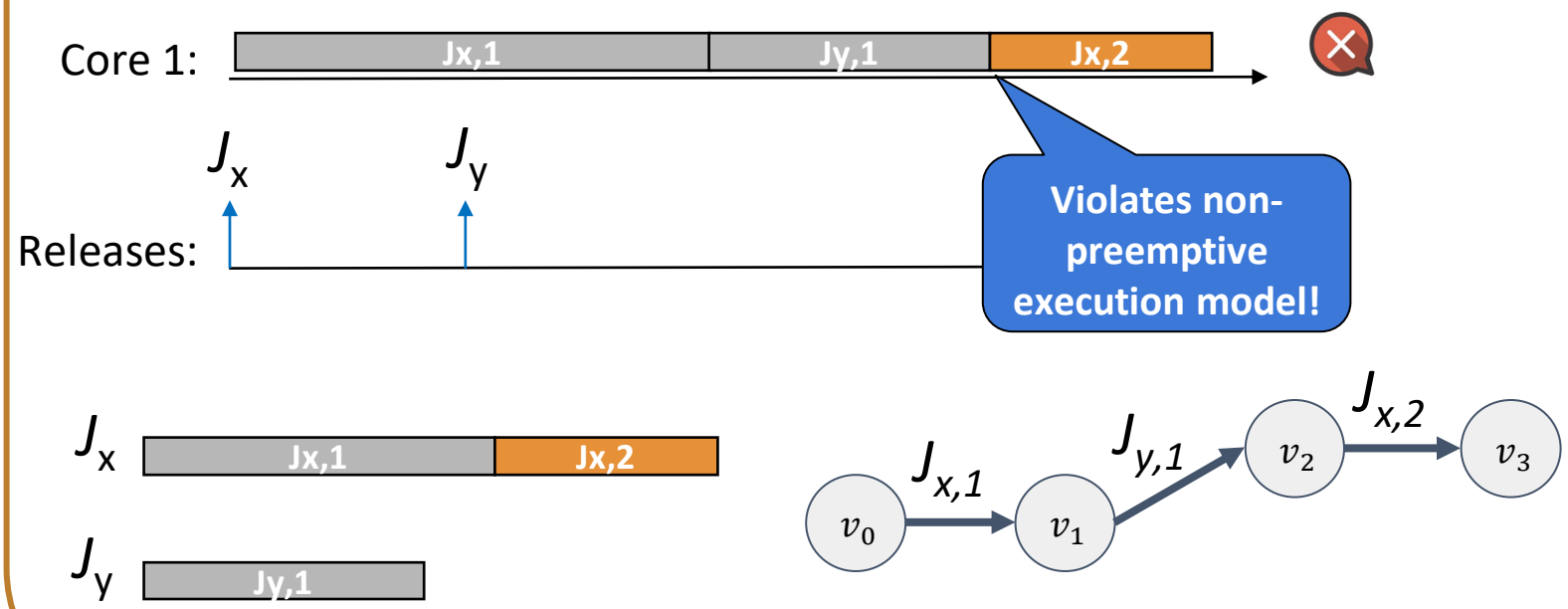
How to preserve non-preemptive execution?

An example schedule:



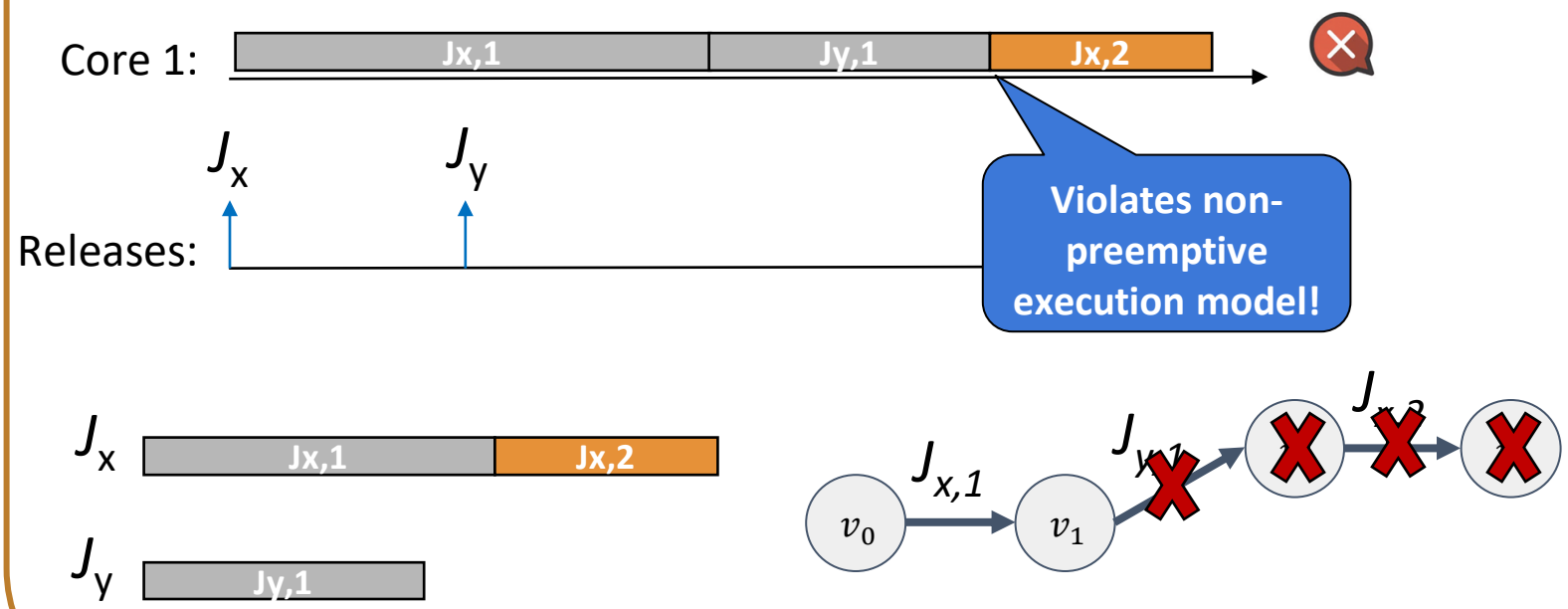
How to preserve non-preemptive execution?

An example schedule:



How to preserve non-preemptive execution?

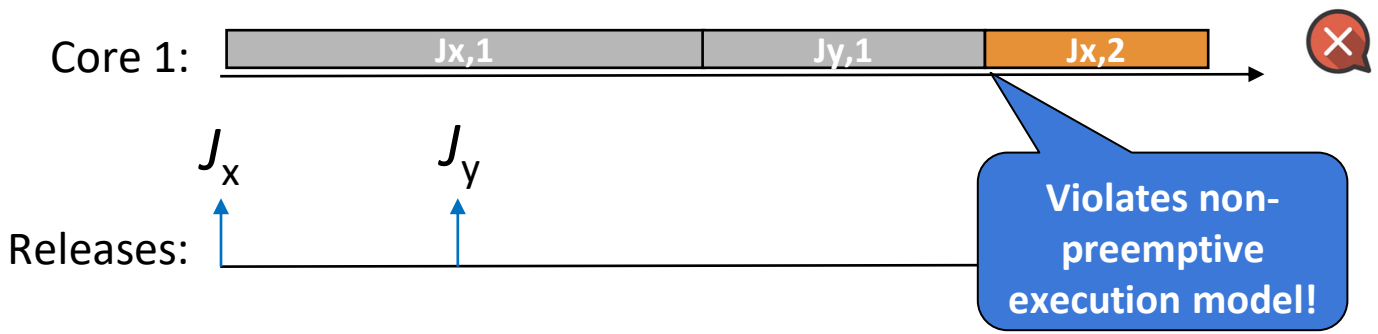
An example schedule:

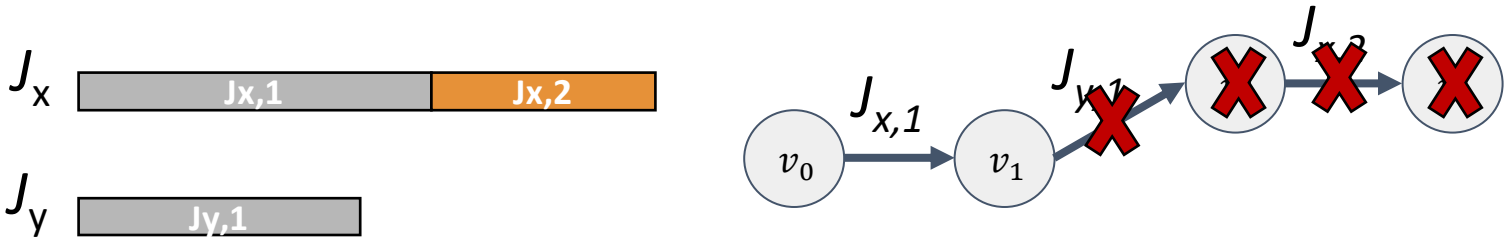


How to preserve non-preemptive execution?

Let jobs claim cores for their execution and release them after they finish

An example schedule:

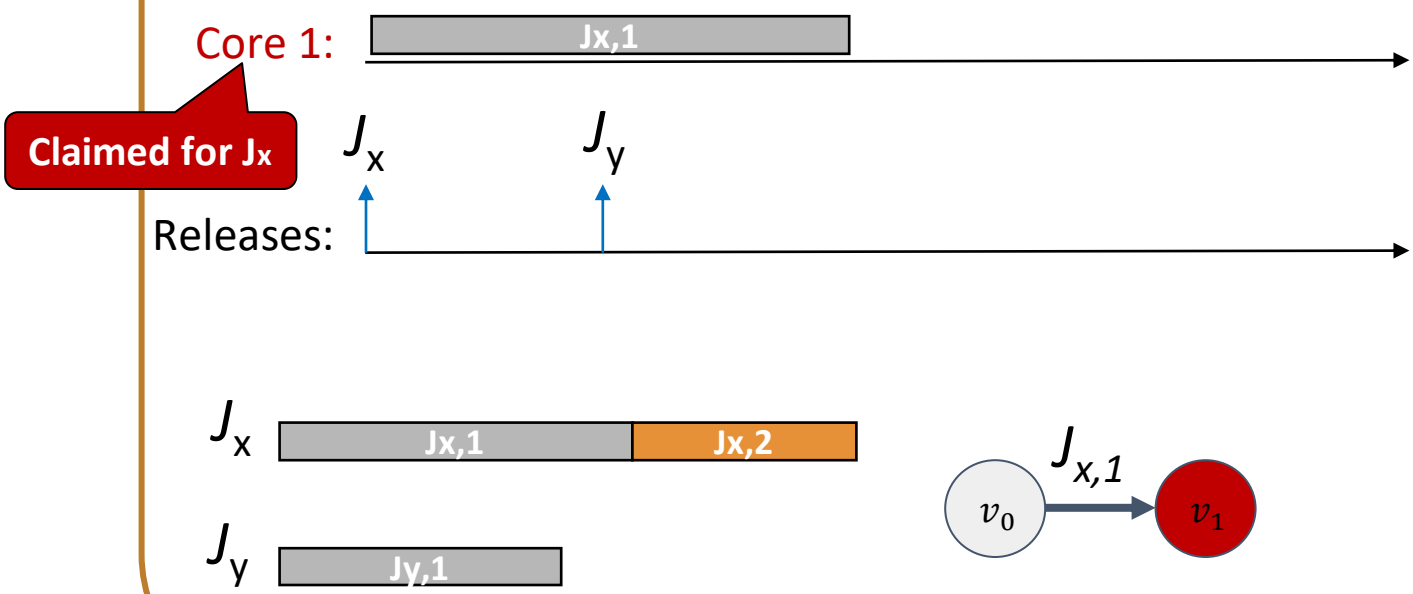




How to preserve non-preemptive execution?

Let jobs claim cores for their execution and release them after they finish

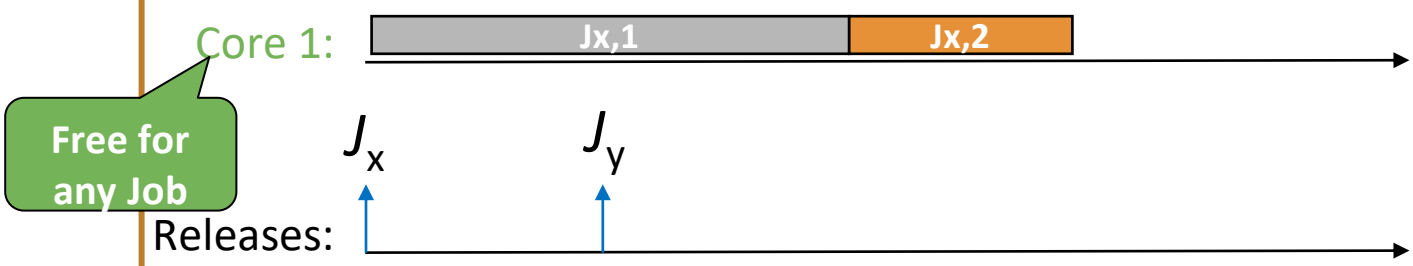
An example schedule:

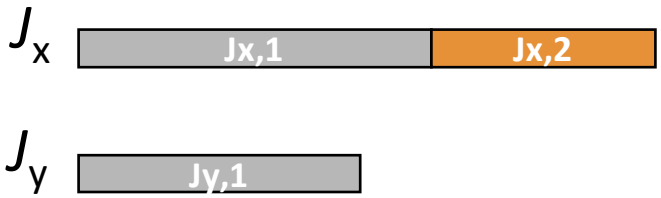


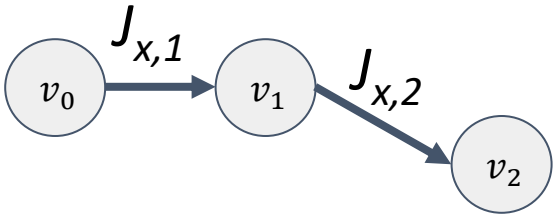
How to preserve non-preemptive execution?

Let jobs claim cores for their execution and release them after they finish

An example schedule:



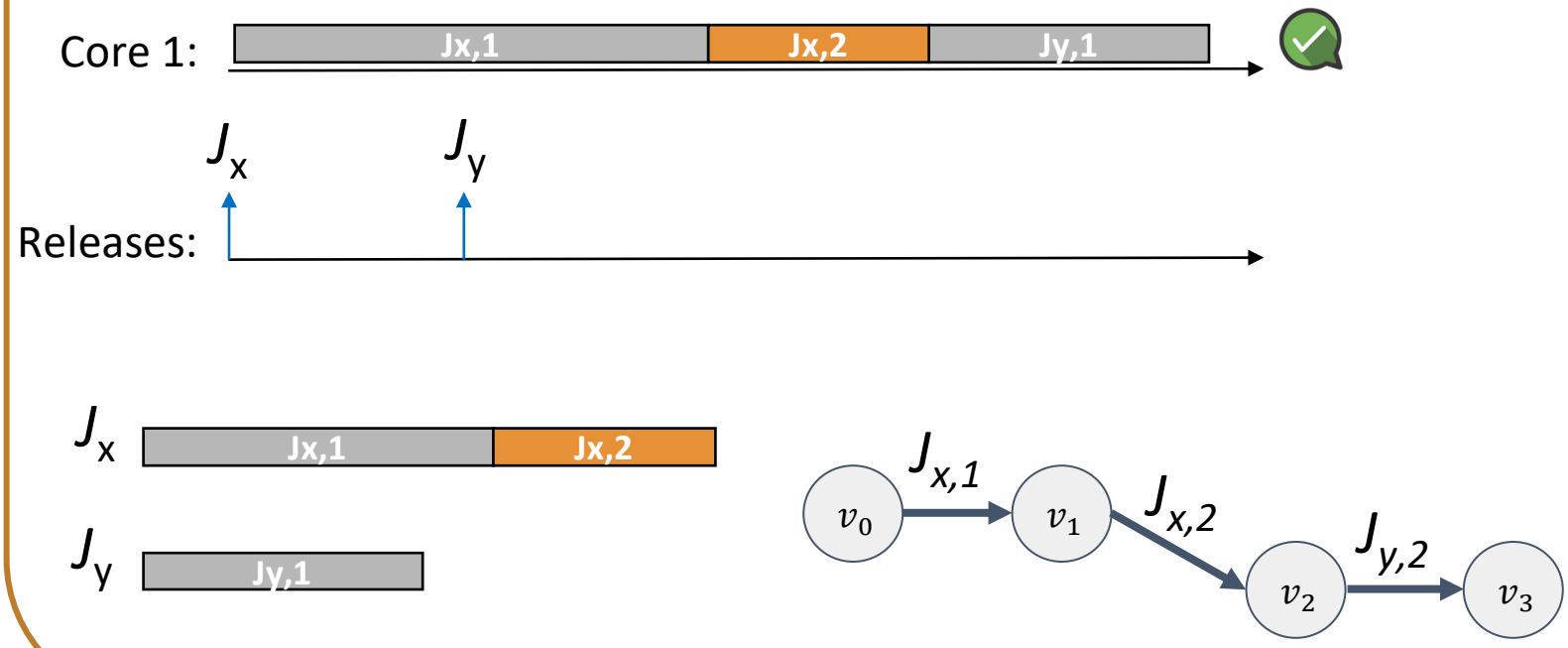




How to preserve non-preemptive execution?

Let jobs claim cores for their execution and release them after they finish

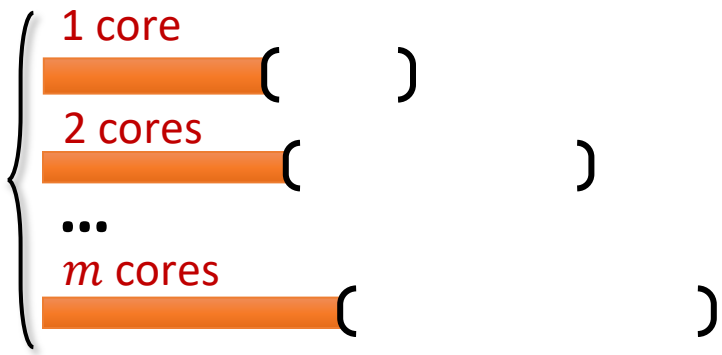
An example schedule:



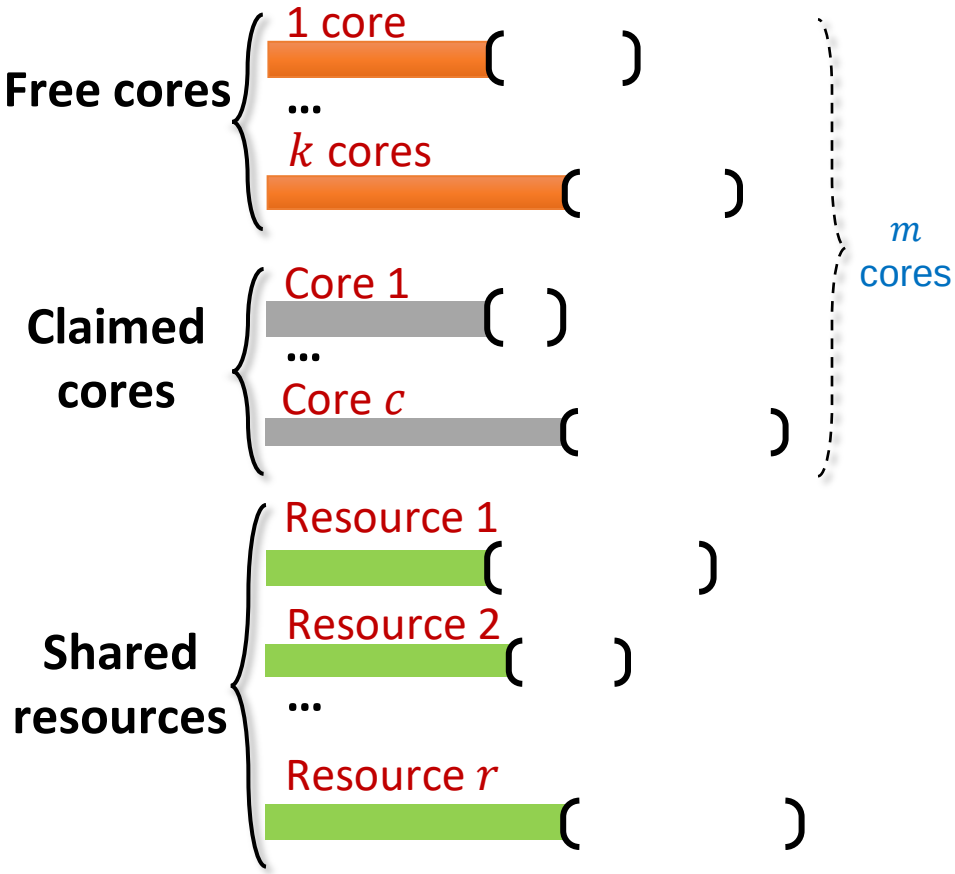
System state abstraction

[ECRTS'19]

Available cores



[This work]



Agenda

To do's:

- [x] ~~Design a system state representation~~
- [ ] Design expansion and merge rules

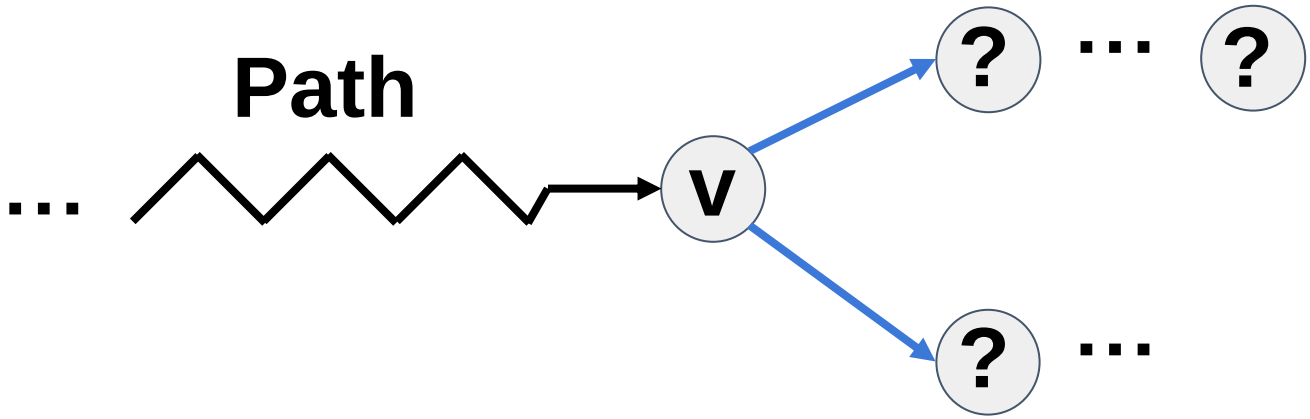
Expansion and merge rules

Expansion and merge rules to support the new system state abstraction and access to shared resource

Expansion and merge rules

Expansion and merge rules to support the new system state abstraction and access to shared resource

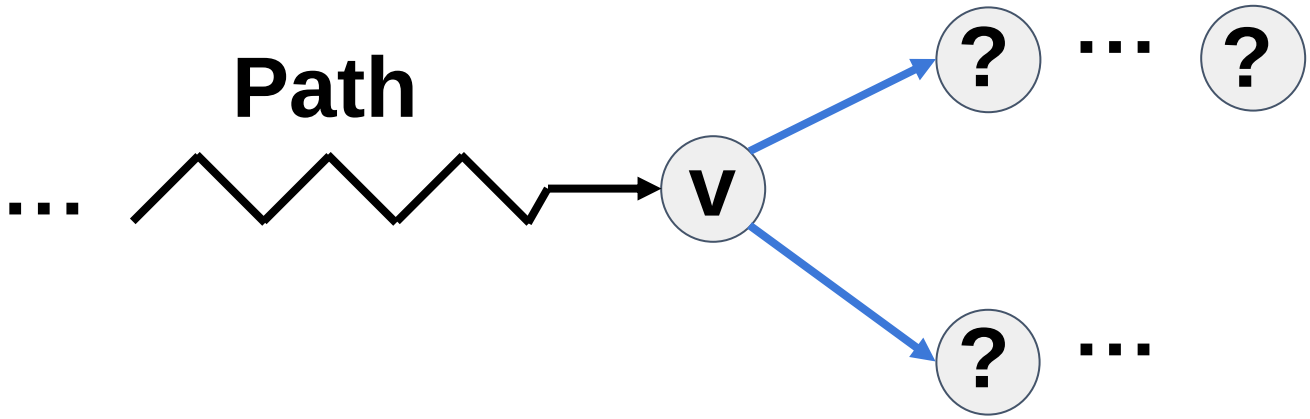
➡ **Potentially ready segments**



Expansion and merge rules

Expansion and merge rules to support the new system state abstraction and access to shared resource

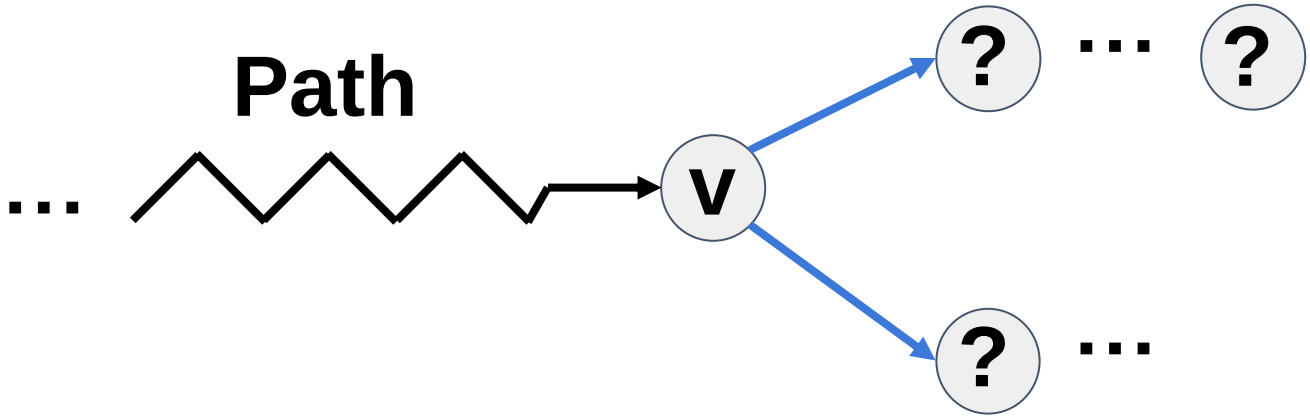
- **Potentially ready segments**
- **Earliest and latest start times**



Expansion and merge rules

Expansion and merge rules to support the new system state abstraction and access to shared resource

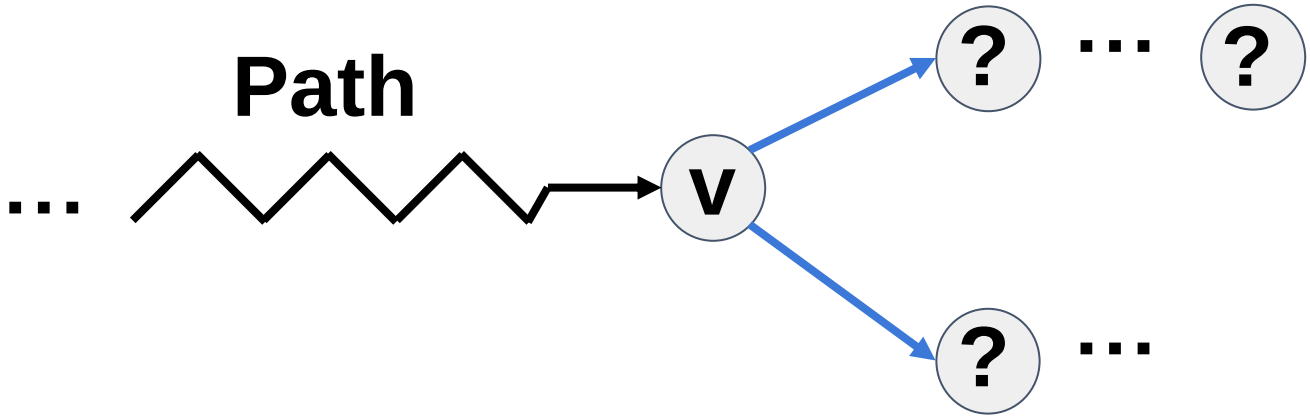
- **Potentially ready segments**
- **Earliest and latest start times**
- **Eligibility condition**



Expansion and merge rules

Expansion and merge rules to support the new system state abstraction and access to shared resource

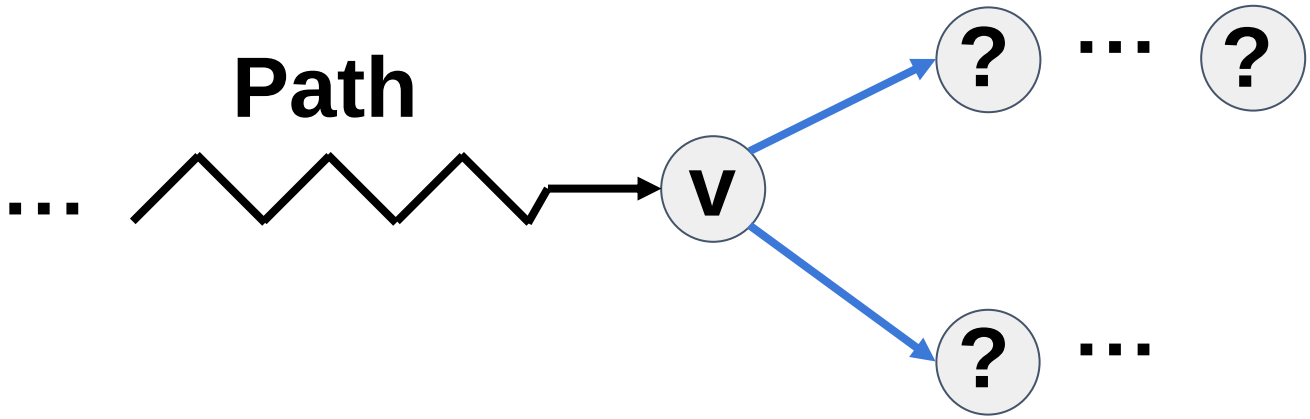
- **Potentially ready segments**
- **Earliest and latest start times**
- **Eligibility condition**
- **Earliest and latest finish times**



Expansion and merge rules

Expansion and merge rules to support the new system state abstraction and access to shared resource

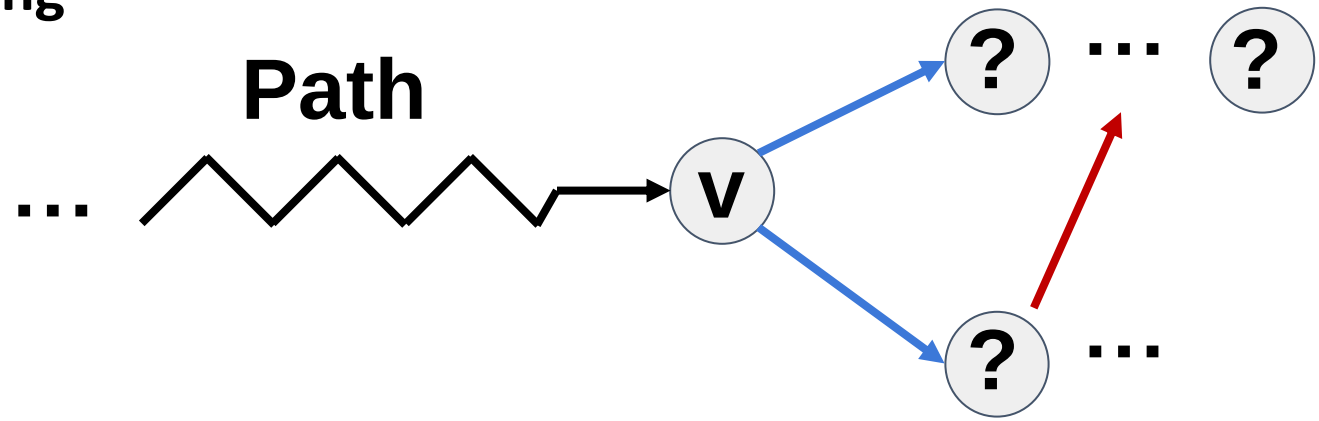
- Potentially ready segments
- Earliest and latest start times
- Eligibility condition
- Earliest and latest finish times
- Update availabilities



Expansion and merge rules

Expansion and merge rules to support the new system state abstraction and access to shared resource

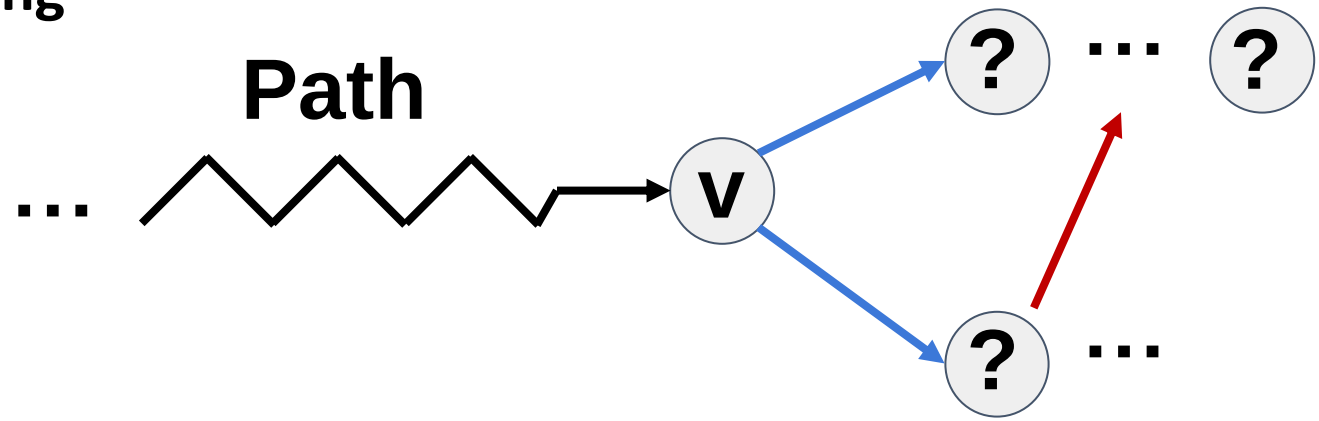
- Potentially ready segments
- Earliest and latest start times
- Eligibility condition
- Earliest and latest finish times
- Update availabilities
- Sound merging



Expansion and merge rules

Expansion and merge rules to support the new system state abstraction and access to shared resource

- Potentially ready segments
- Earliest and latest start times
- Eligibility condition
- Earliest and latest finish times
- Update availabilities
- Sound merging



Agenda

To do's:

- ~~Design a system state representation~~
- ~~Design expansion and merge rules~~

Evaluation

Evaluation baselines

Name	Type	Description
{EDF,FP}-SAG-SR-{FIFO,PRIO}	Scenario-aware analysis	Our approach

[1] B. B. Brandenburg. Scheduling and Locking in Multiprocessor Real-Time Operating Systems. PhD thesis, The University of North Carolina at Chapel Hill, 2011.

[2] M. Yang, A. Wieder, and B. B. Brandenburg. Global real-time semaphore protocols: A survey, unified analysis, and comparison. In 2015 IEEE Real-Time Systems Symposium, pages 1-12. IEEE, 2015.

Evaluation baselines

Name	Type	Description
{EDF,FP}-SAG-SR-{FIFO,PRIO}	Scenario-aware analysis	Our approach
{EDF,FP}-SAG-NO-BLOCKING	**---**	**Upper bound on attainable schedulability (NP+SAG)**
{EDF,FP}-SAG-INF-{FIFO,PRIO}	**Inflation-based analysis**	**SAG + inflation-based analysis**

[1] B. B. Brandenburg. Scheduling and Locking in Multiprocessor Real-Time Operating Systems. PhD thesis, The University of North Carolina at Chapel Hill, 2011.

[2] M. Yang, A. Wieder, and B. B. Brandenburg. Global real-time semaphore protocols: A survey, unified analysis, and comparison. In 2015 IEEE Real-Time Systems Symposium, pages 1-12. IEEE, 2015.

Evaluation baselines

Name	Type	Description
{EDF,FP}-SAG-SR-{FIFO,PRIO}	Scenario-aware analysis	Our approach
{EDF,FP}-SAG-NO-BLOCKING	---	Upper bound on attainable schedulability (NP+SAG)
{EDF,FP}-SAG-INF-{FIFO,PRIO}	Inflation-based analysis	SAG + inflation-based analysis
EDF-NO-BLOCKING	**---**	**Upper bound on attainable schedulability (P+EDF)**
FP-NO-BLOCKING	**---**	**Upper bound on attainable schedulability (P+FP)**

[1] B. B. Brandenburg. Scheduling and Locking in Multiprocessor Real-Time Operating Systems. PhD thesis, The University of North Carolina at Chapel Hill, 2011.
[2] M. Yang, A. Wieder, and B. B. Brandenburg. Global real-time semaphore protocols: A survey, unified analysis, and comparison. In 2015 IEEE Real-Time Systems Symposium, pages 1-12. IEEE, 2015.

Evaluation baselines

Name	Type	Description
{EDF,FP}-SAG-SR-{FIFO,PRIO}	Scenario-aware analysis	Our approach
{EDF,FP}-SAG-NO-BLOCKING	---	Upper bound on attainable schedulability (NP+SAG)
{EDF,FP}-SAG-INF-{FIFO,PRIO}	Inflation-based analysis	SAG + inflation-based analysis
EDF-NO-BLOCKING	---	Upper bound on attainable schedulability (P+EDF)
FP-NO-BLOCKING	---	Upper bound on attainable schedulability (P+FP)
{EDF,FP}-FMLP-SHORT	**Inflation-based analysis**	**Inflation-based analysis considering FMLP & short critical sections**
{EDF,FP}-OMLP	**Inflation-based analysis**	**Inflation-based analysis considering OMLP**

[1] B. B. Brandenburg. Scheduling and Locking in Multiprocessor Real-Time Operating Systems. PhD thesis, The University of North Carolina at Chapel Hill, 2011.
[2] M. Yang, A. Wieder, and B. B. Brandenburg. Global real-time semaphore protocols: A survey, unified analysis, and comparison. In 2015 IEEE Real-Time Systems Symposium, pages 1-12. IEEE, 2015.

Evaluation baselines

Name	Type	Description
{EDF,FP}-SAG-SR-{FIFO,PRIO}	Scenario-aware analysis	Our approach
{EDF,FP}-SAG-NO-BLOCKING	---	Upper bound on attainable schedulability (NP+SAG)
{EDF,FP}-SAG-INF-{FIFO,PRIO}	Inflation-based analysis	SAG + inflation-based analysis
EDF-NO-BLOCKING	---	Upper bound on attainable schedulability (P+EDF)
FP-NO-BLOCKING	---	Upper bound on attainable schedulability (P+FP)
{EDF,FP}-FMLP-SHORT	Inflation-based analysis	Inflation-based analysis considering FMLP & short critical sections
{EDF,FP}-OMLP	Inflation-based analysis	Inflation-based analysis considering OMLP
FP-FMLP-LONG	**LP-based analysis**	**Linear-programming-based analysis for FMLP & long critical sections**
FP-PIP	**LP-based analysis**	**Linear-programming-based analysis for PIP**

[1] B. B. Brandenburg. Scheduling and Locking in Multiprocessor Real-Time Operating Systems. PhD thesis, The University of North Carolina at Chapel Hill, 2011.
[2] M. Yang, A. Wieder, and B. B. Brandenburg. Global real-time semaphore protocols: A survey, unified analysis, and comparison. In 2015 IEEE Real-Time Systems Symposium, pages 1-12. IEEE, 2015.

Evaluation

Goal:

→ **Evaluate the accuracy of the analysis**

Evaluation

Goal:

→ **Evaluate the accuracy of the analysis**

Metric:

→ **Schedulability Ratio:** $\frac{Number\ of\ task\ sets\ deemed\ schedulable}{Number\ of\ total\ evaluated\ task\ sets}$

Evaluation

Goal:

→ **Evaluate the accuracy of the analysis**

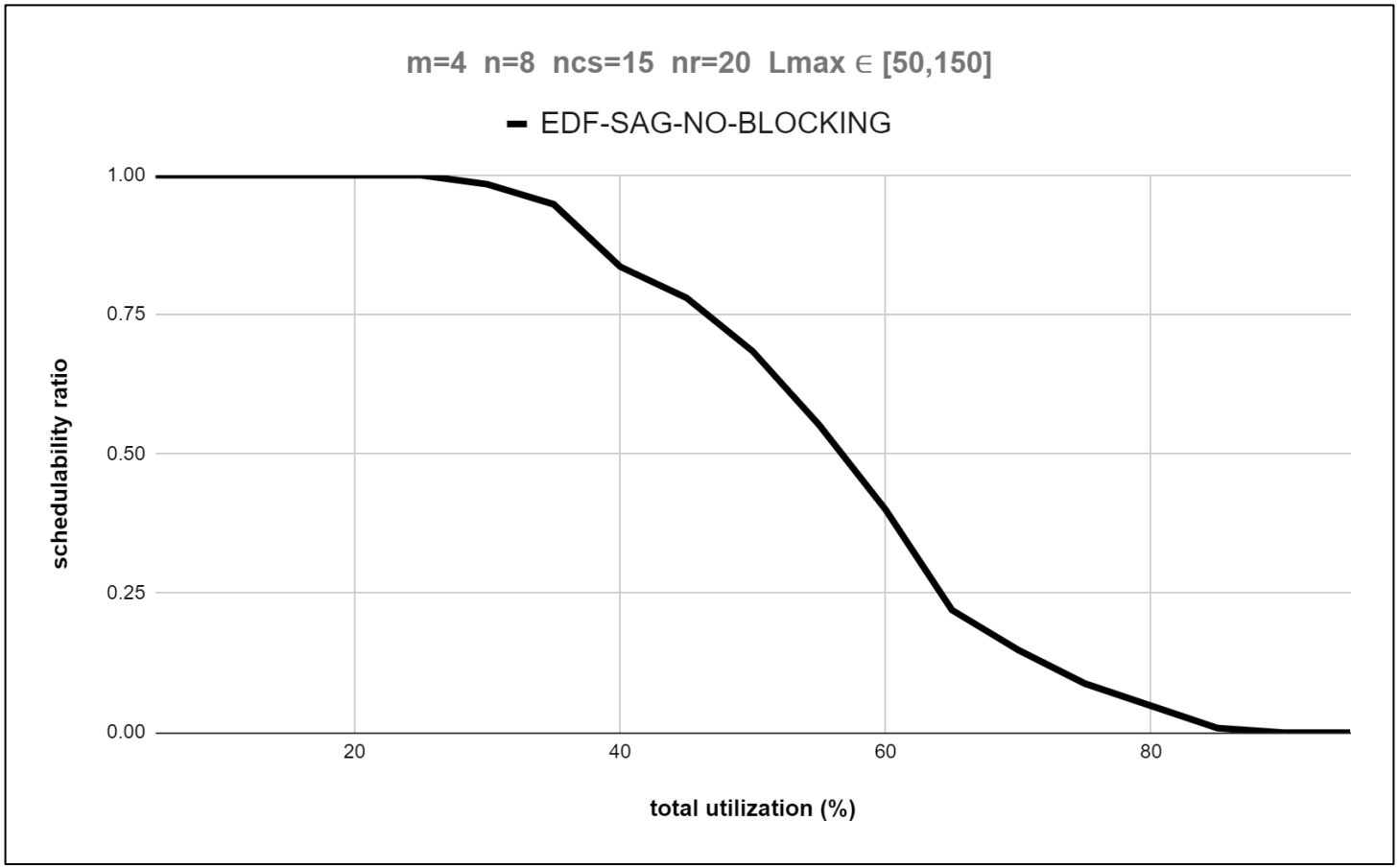
Metric:

→ **Schedulability Ratio:** $\frac{Number\ of\ task\ sets\ deemed\ schedulable}{Number\ of\ total\ evaluated\ task\ sets}$

Task sets:

→ **Randomly generated**

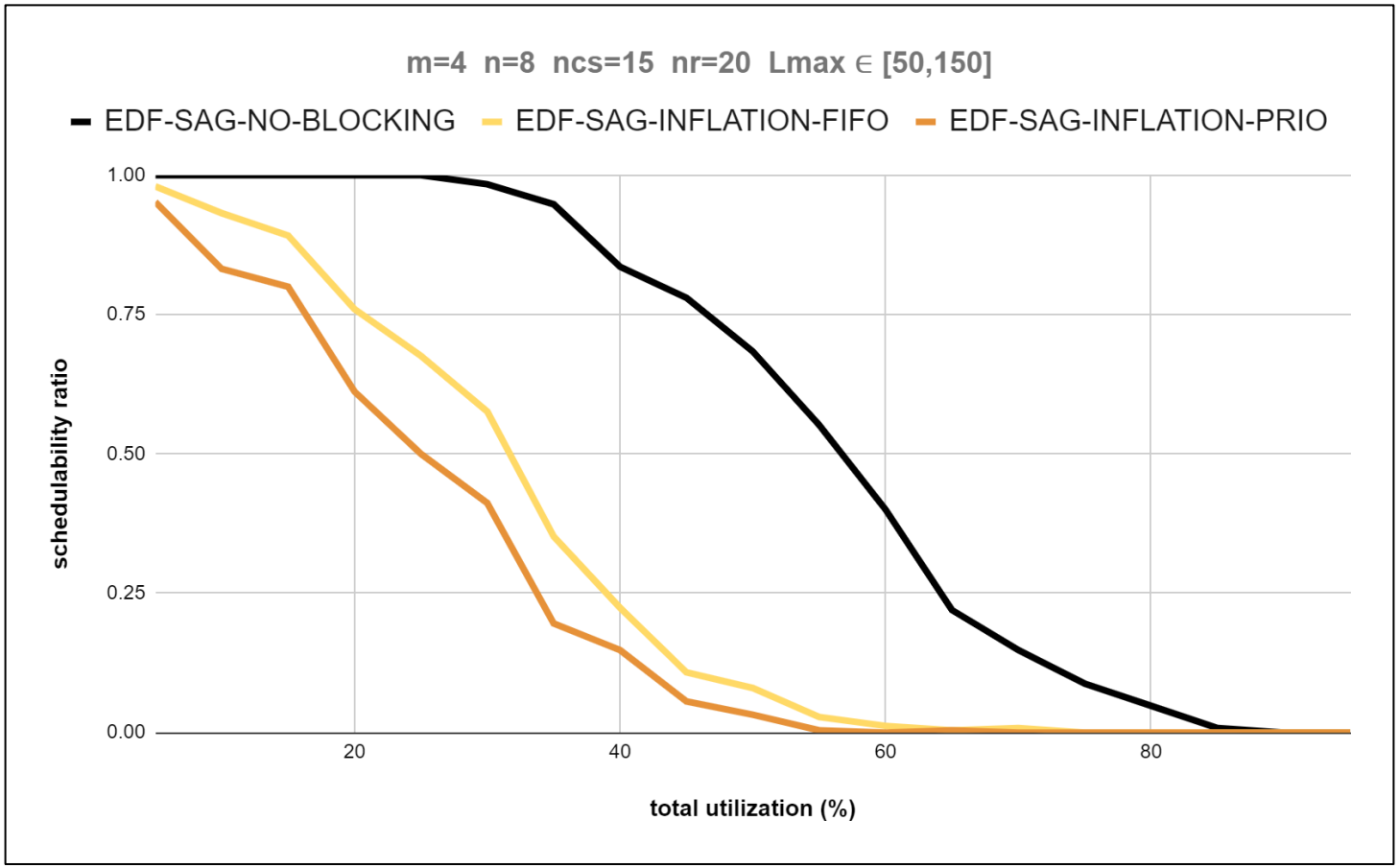
Results: SAG comparison



Legend:

- m: Number of cores
- n: Number of tasks
- n^{cs}: Maximum per task critical sections
- n^{r}: Number of shared resources
- L^{max}: Maximum length of critical sections

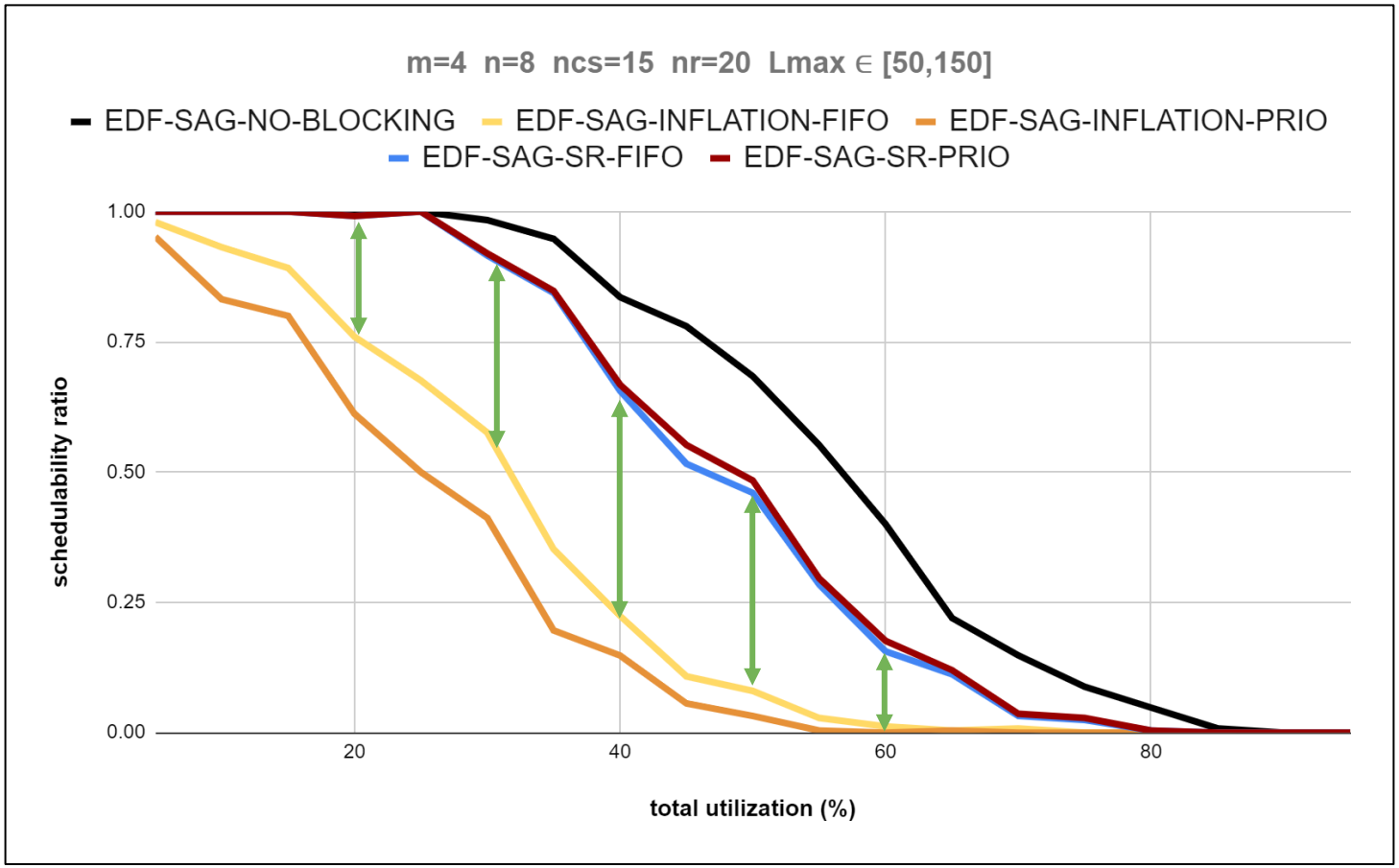
Results: SAG comparison



Legend:

- m: Number of cores
- n: Number of tasks
- n^{cs}: Maximum per task critical sections
- n^{r}: Number of shared resources
- L^{max}: Maximum length of critical sections

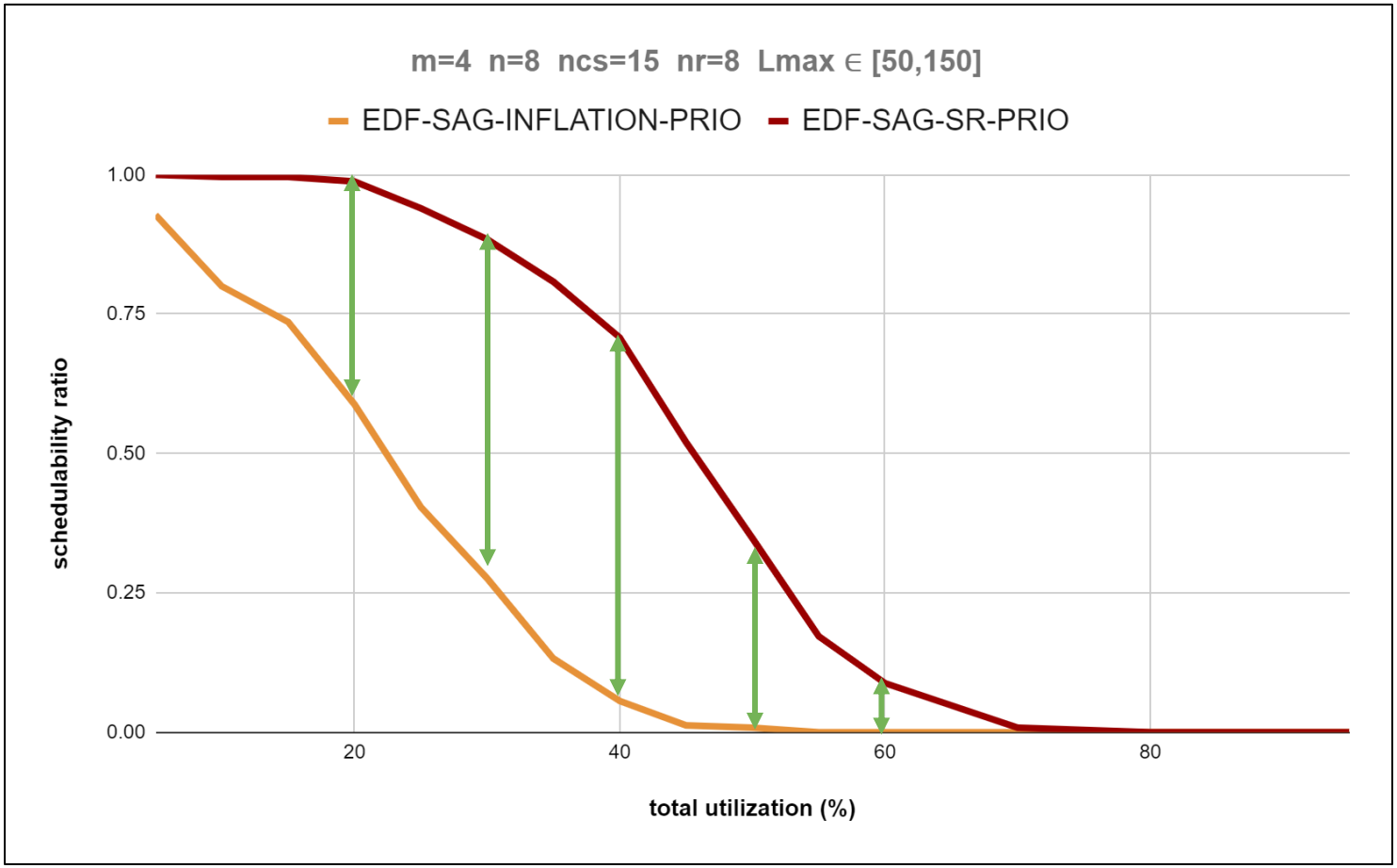
Results: SAG comparison



Legend:

- m: Number of cores
- n: Number of tasks
- n^{cs}: Maximum per task critical sections
- n^{r}: Number of shared resources
- L^{max}: Maximum length of critical sections

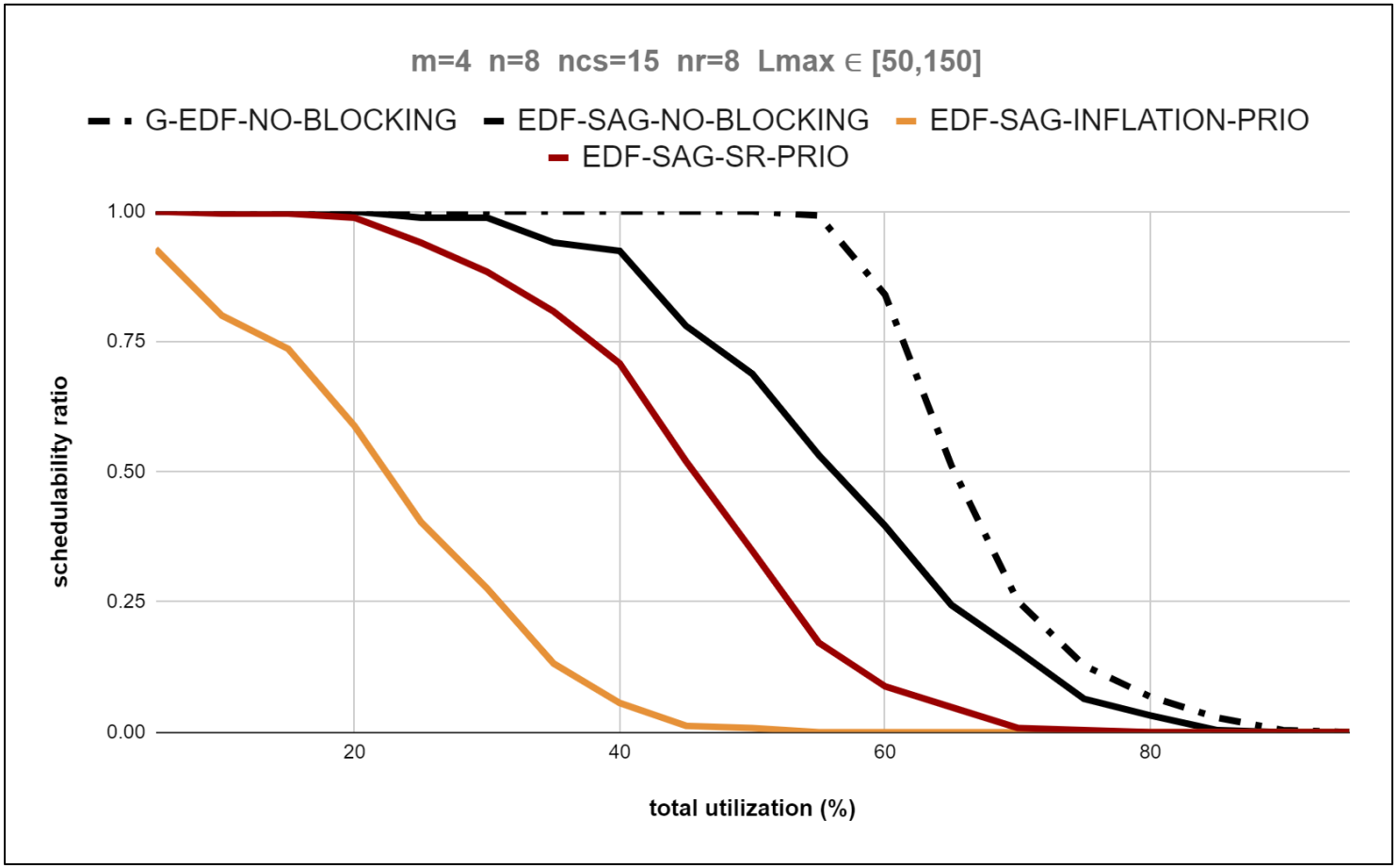
Results: EDF comparison



Legend:

- m: Number of cores
- n: Number of tasks
- n^{cs}: Maximum per task critical sections
- n^{r}: Number of shared resources
- L^{max}: Maximum length of critical sections

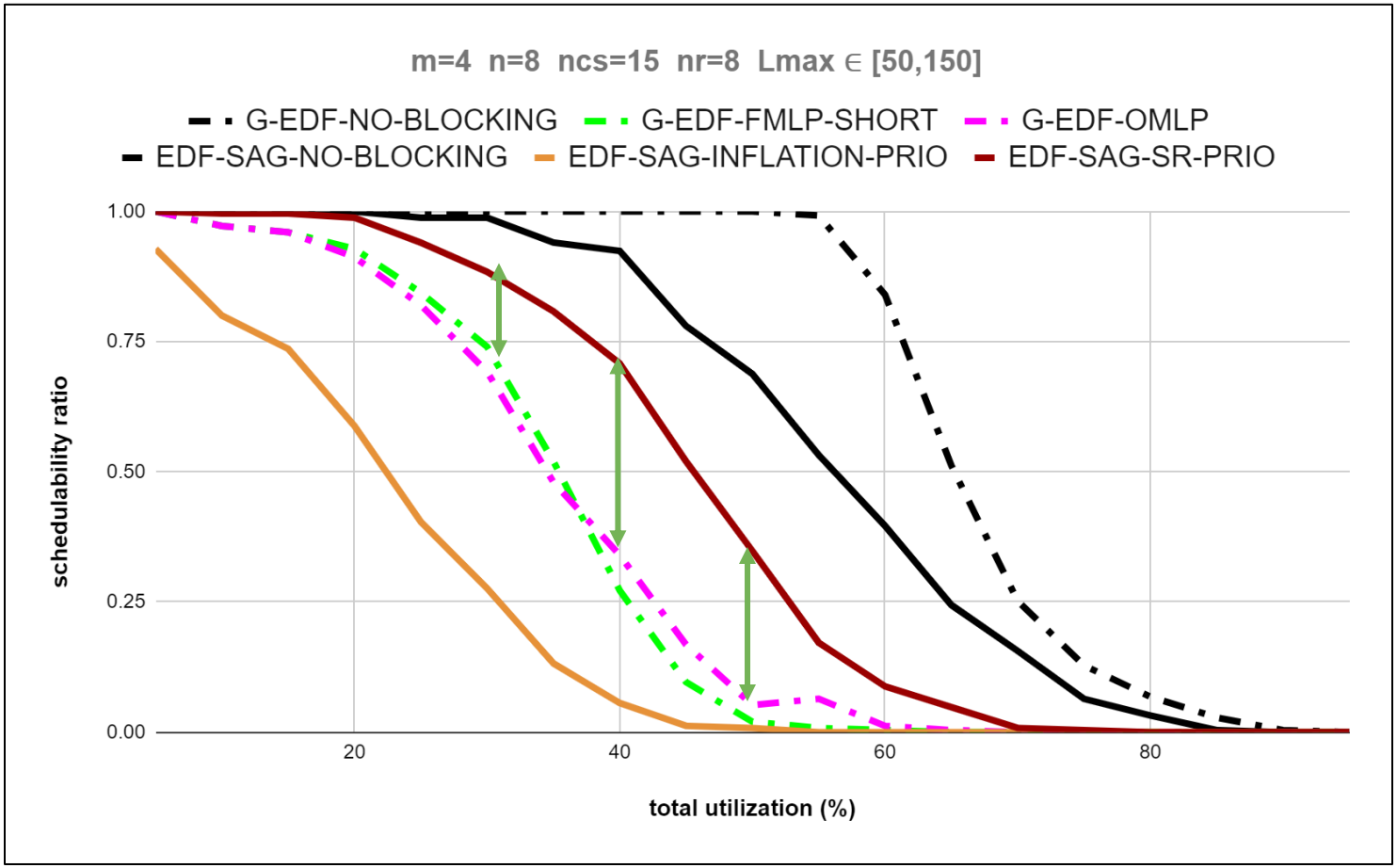
Results: EDF comparison



Legend:

- m: Number of cores
- n: Number of tasks
- n^{cs}: Maximum per task critical sections
- n^{r}: Number of shared resources
- L^{max}: Maximum length of critical sections

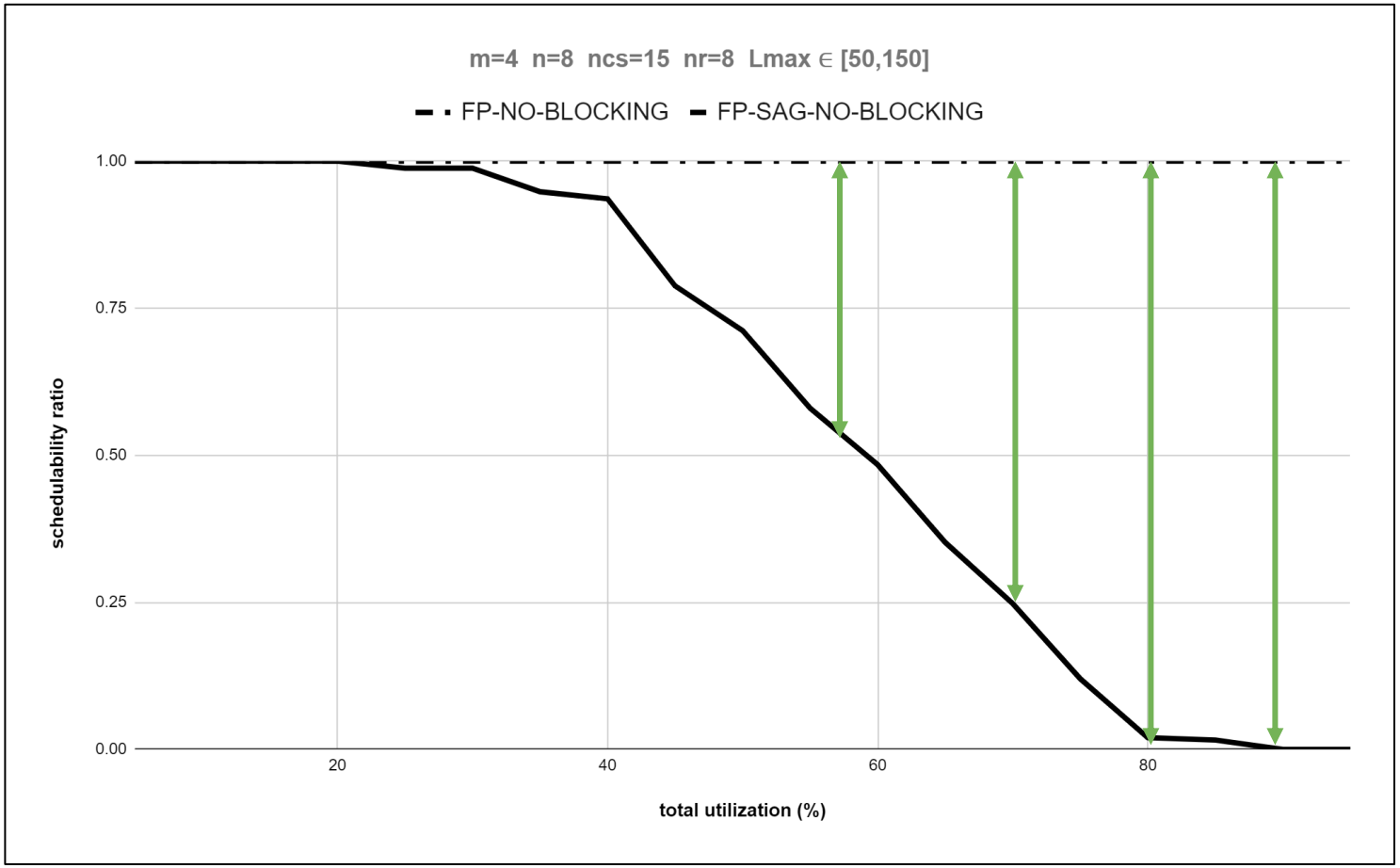
Results: EDF comparison



Legend:

- m: Number of cores
- n: Number of tasks
- n^{cs}: Maximum per task critical sections
- n^{r}: Number of shared resources
- L^{max}: Maximum length of critical sections

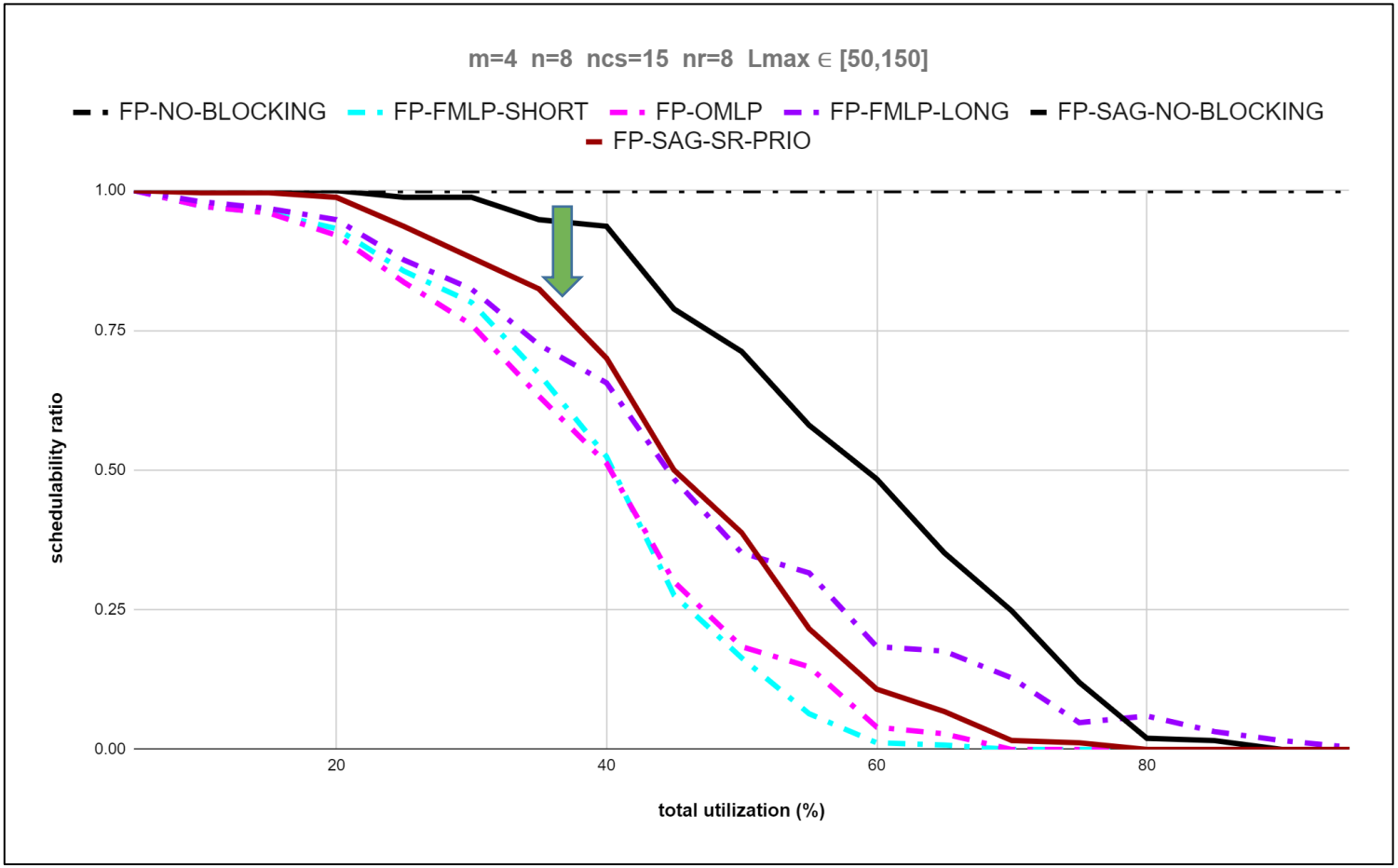
Results: FP comparison 1



Legend:

- m: Number of cores
- n: Number of tasks
- n^{cs}: Maximum per task critical sections
- n^{r}: Number of shared resources
- L^{max}: Maximum length of critical sections

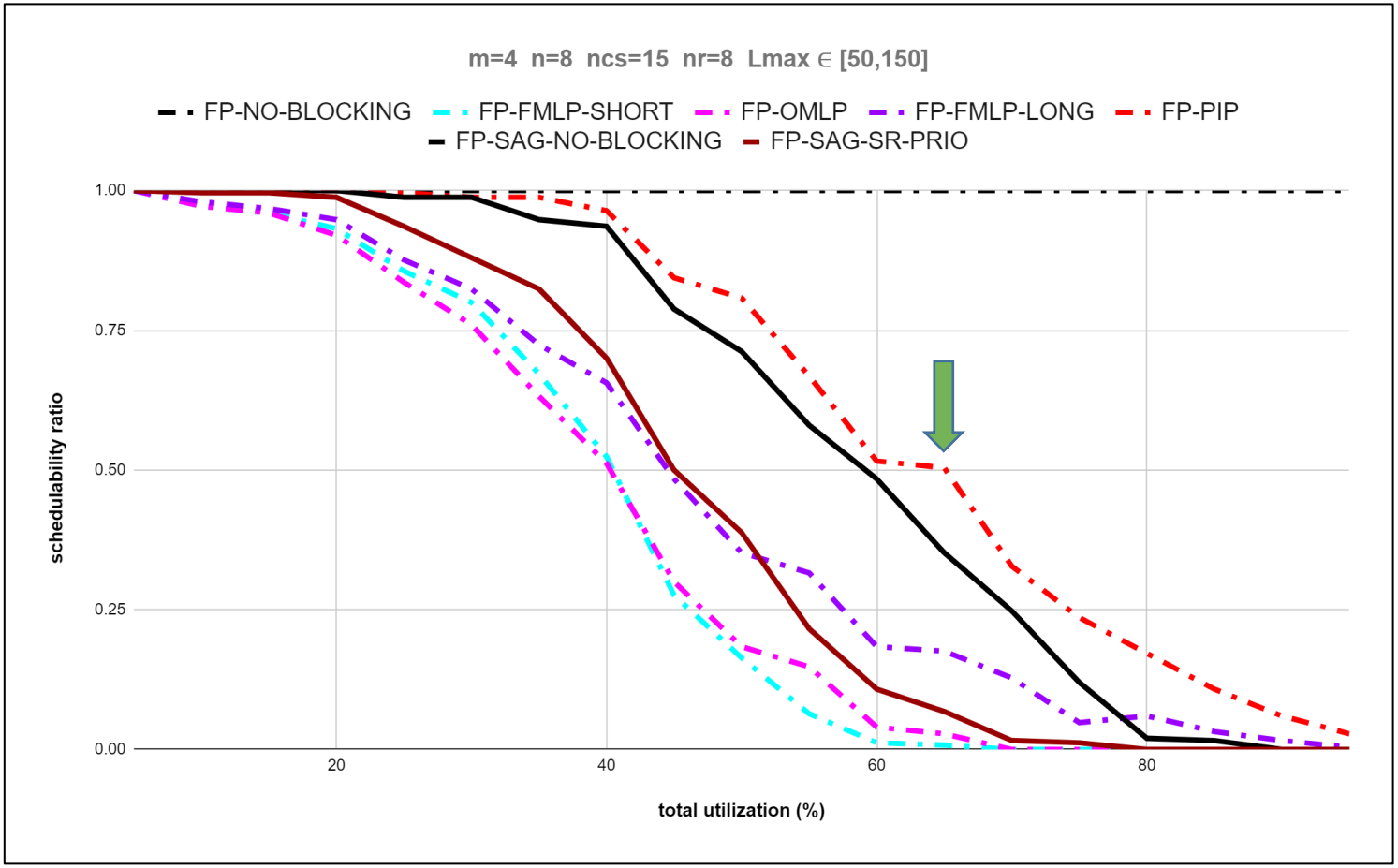
Results: FP comparison 1



Legend:

- m: Number of cores
- n: Number of tasks
- n^{cs}: Maximum per task critical sections
- n^{r}: Number of shared resources
- L^{max}: Maximum length of critical sections

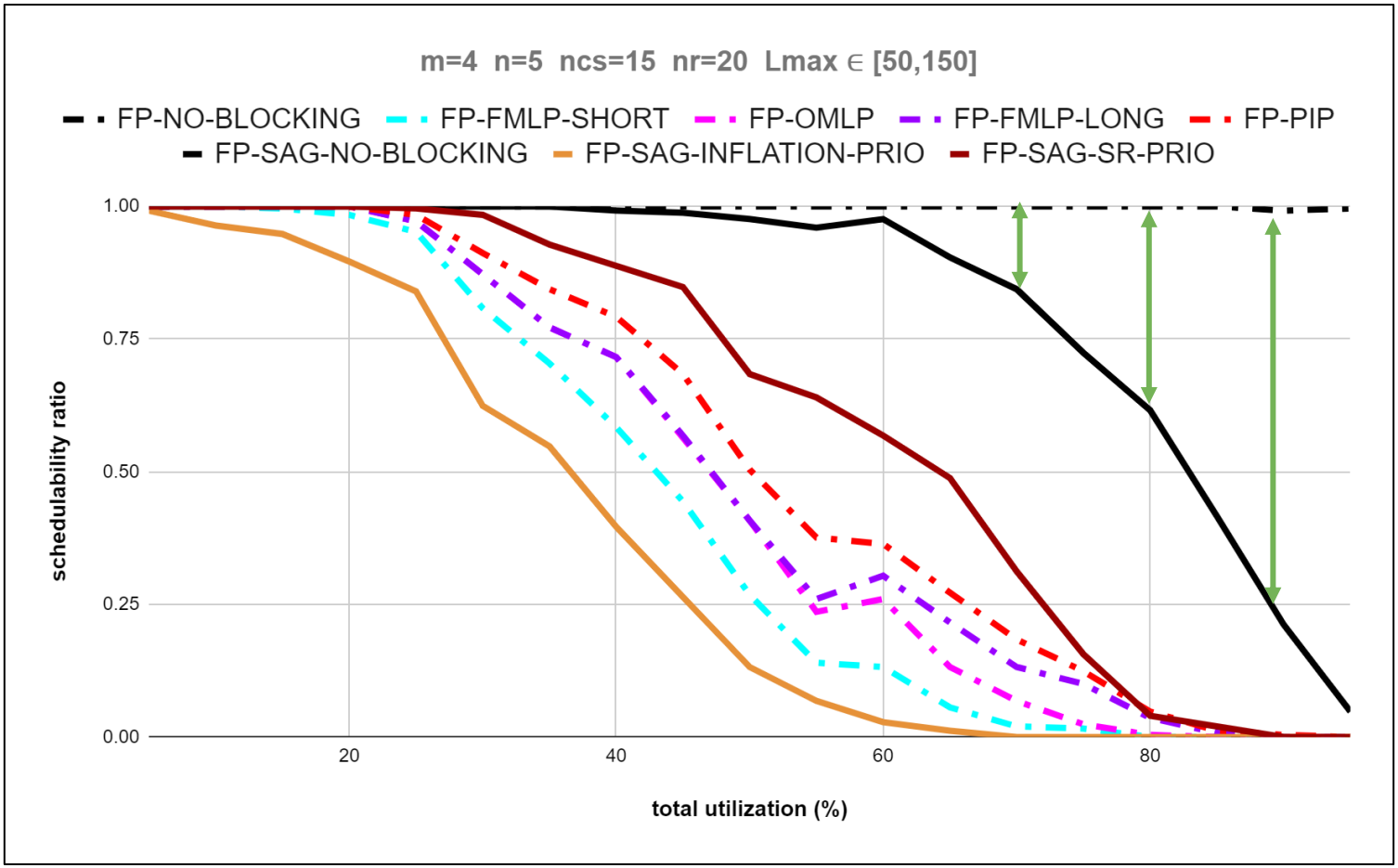
Results: FP comparison 1



Legend:

- m: Number of cores
- n: Number of tasks
- n^{cs}: Maximum per task critical sections
- n^{r}: Number of shared resources
- L^{max}: Maximum length of critical sections

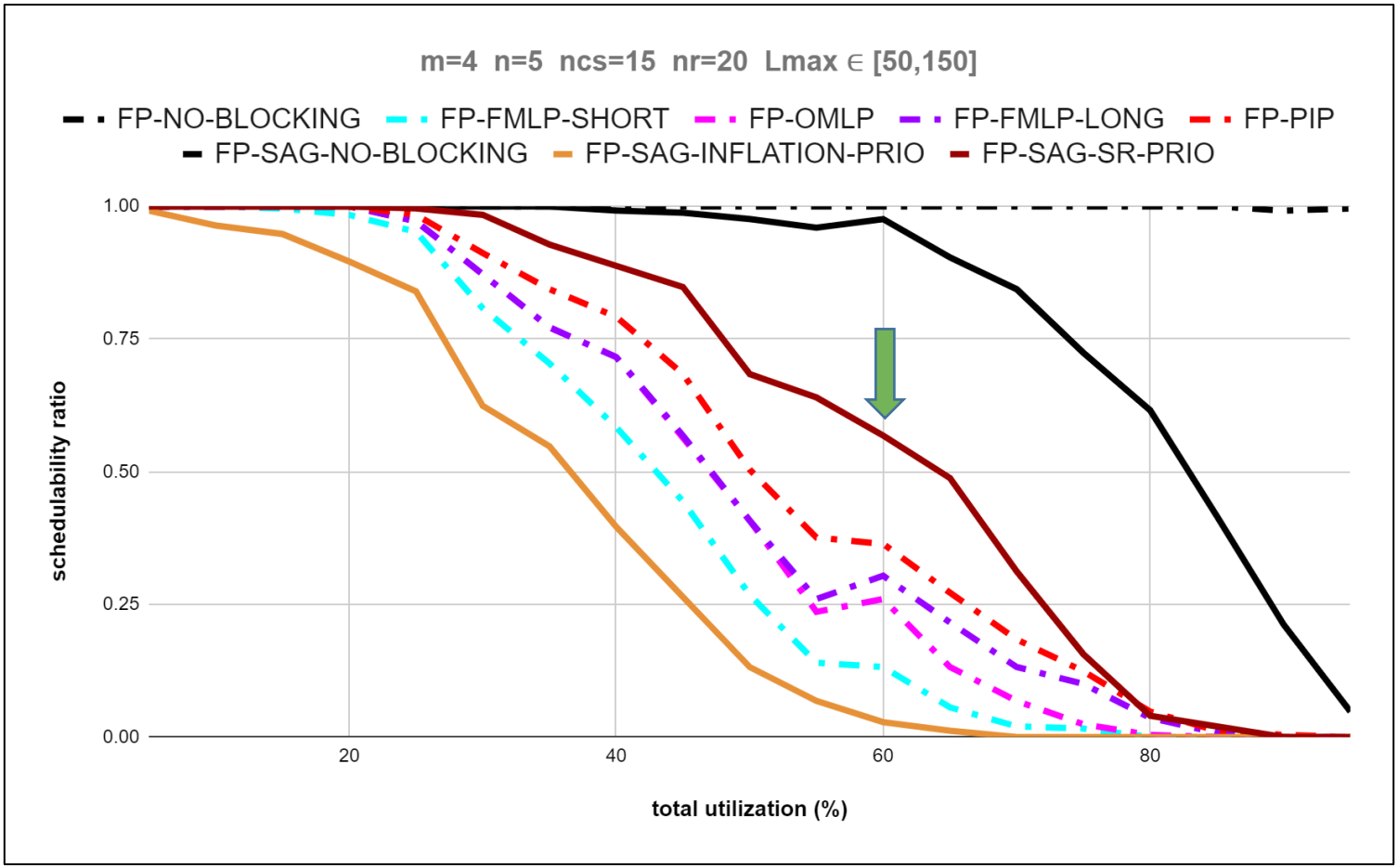
Results: FP comparison 2



Legend:

- m: Number of cores
- n: Number of tasks
- n^{cs}: Maximum per task critical sections
- n^{r}: Number of shared resources
- L^{max}: Maximum length of critical sections

Results: FP comparison 2



Legend:

- m: Number of cores
- n: Number of tasks
- n^{cs}: Maximum per task critical sections
- n^{r}: Number of shared resources
- L^{max}: Maximum length of critical sections

Extensions of the work

Partial order reduction

What is it?

→ Smart techniques to prune the graph

What is the goal?

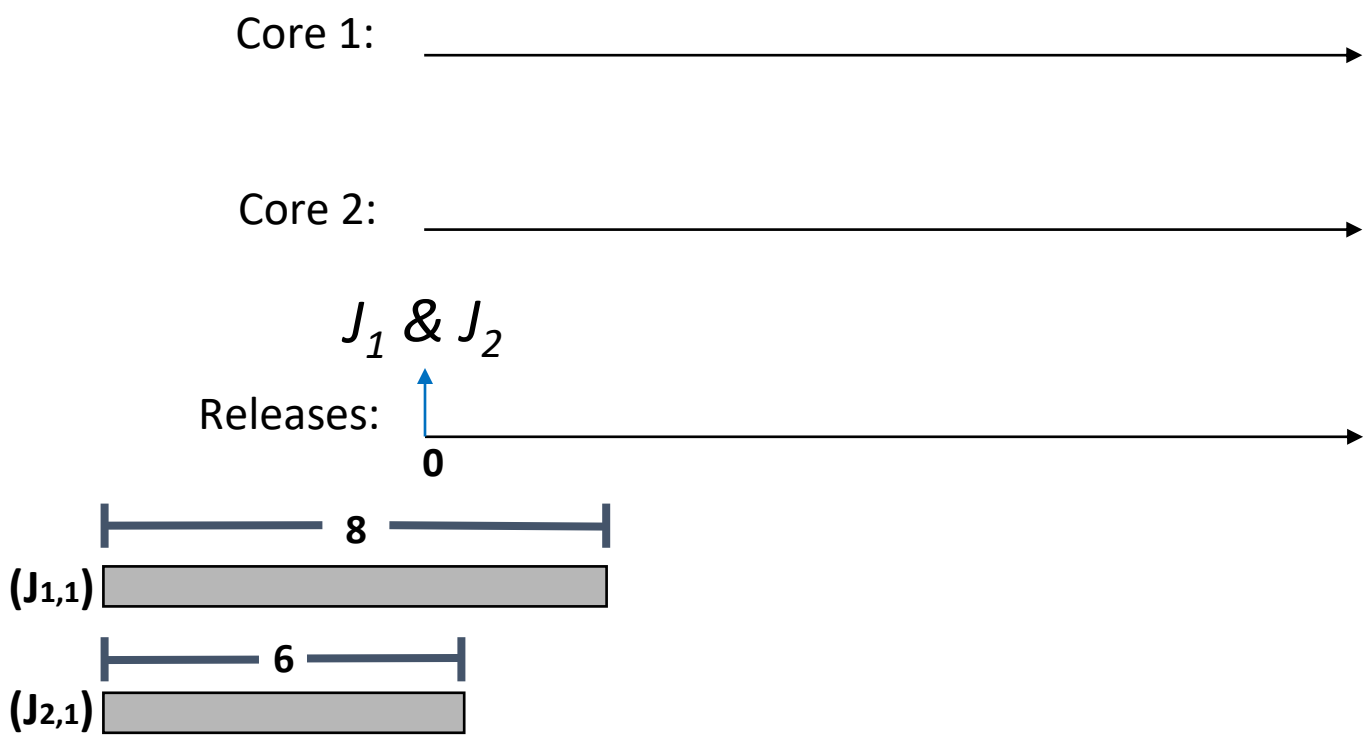
→ Slow down the growth of the graph to **improve scalability**

What is the key idea?

→ Find states where branching is redundant!

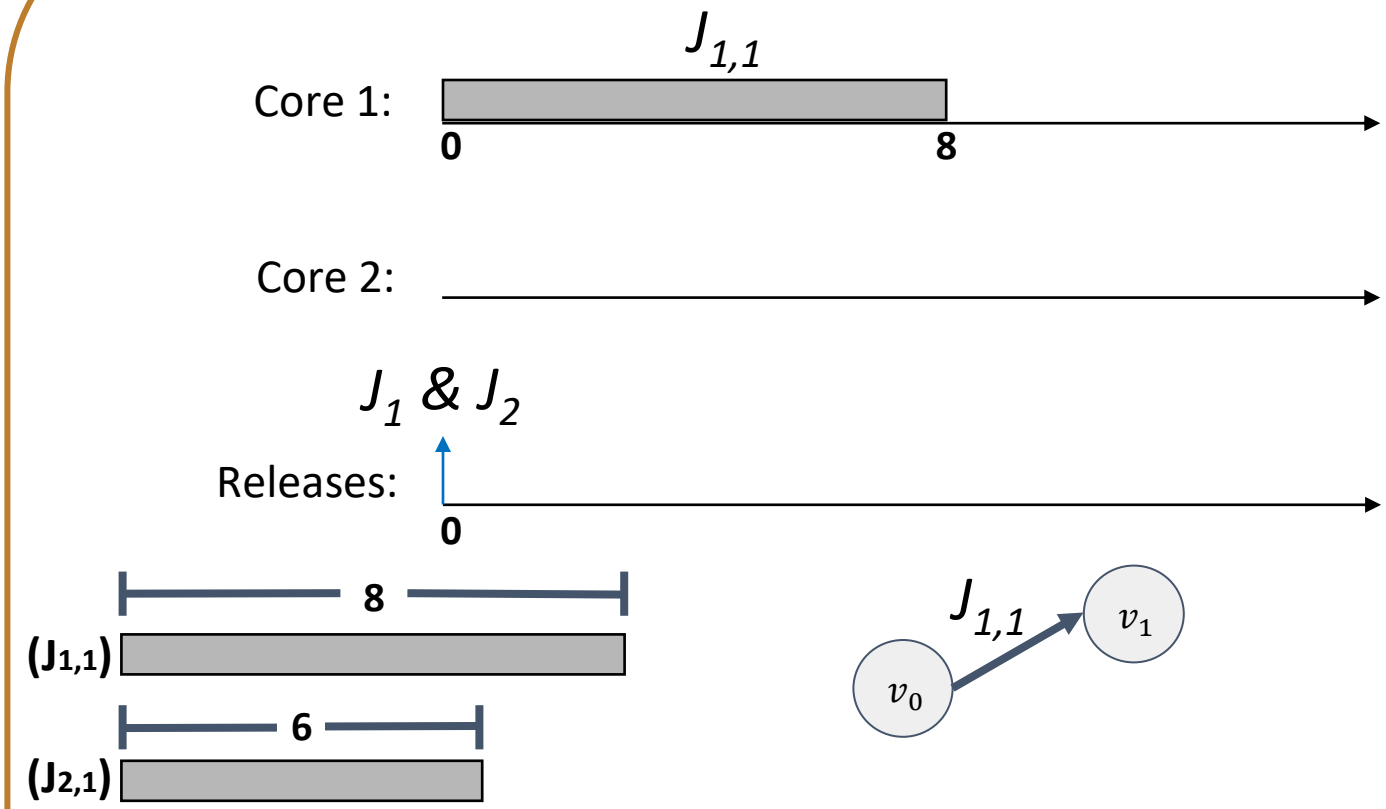
Partial order reduction

An example schedule:



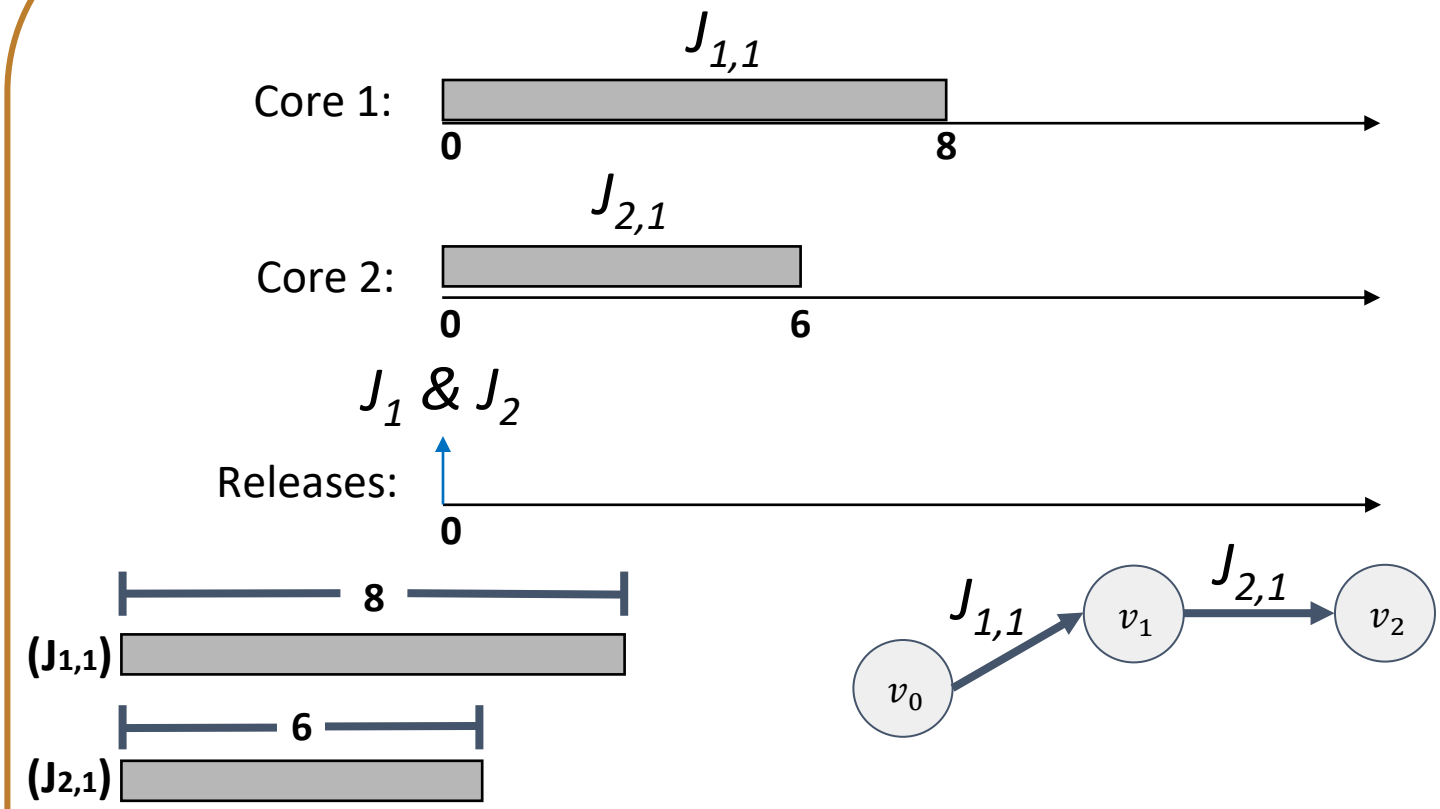
Partial order reduction

An example schedule:



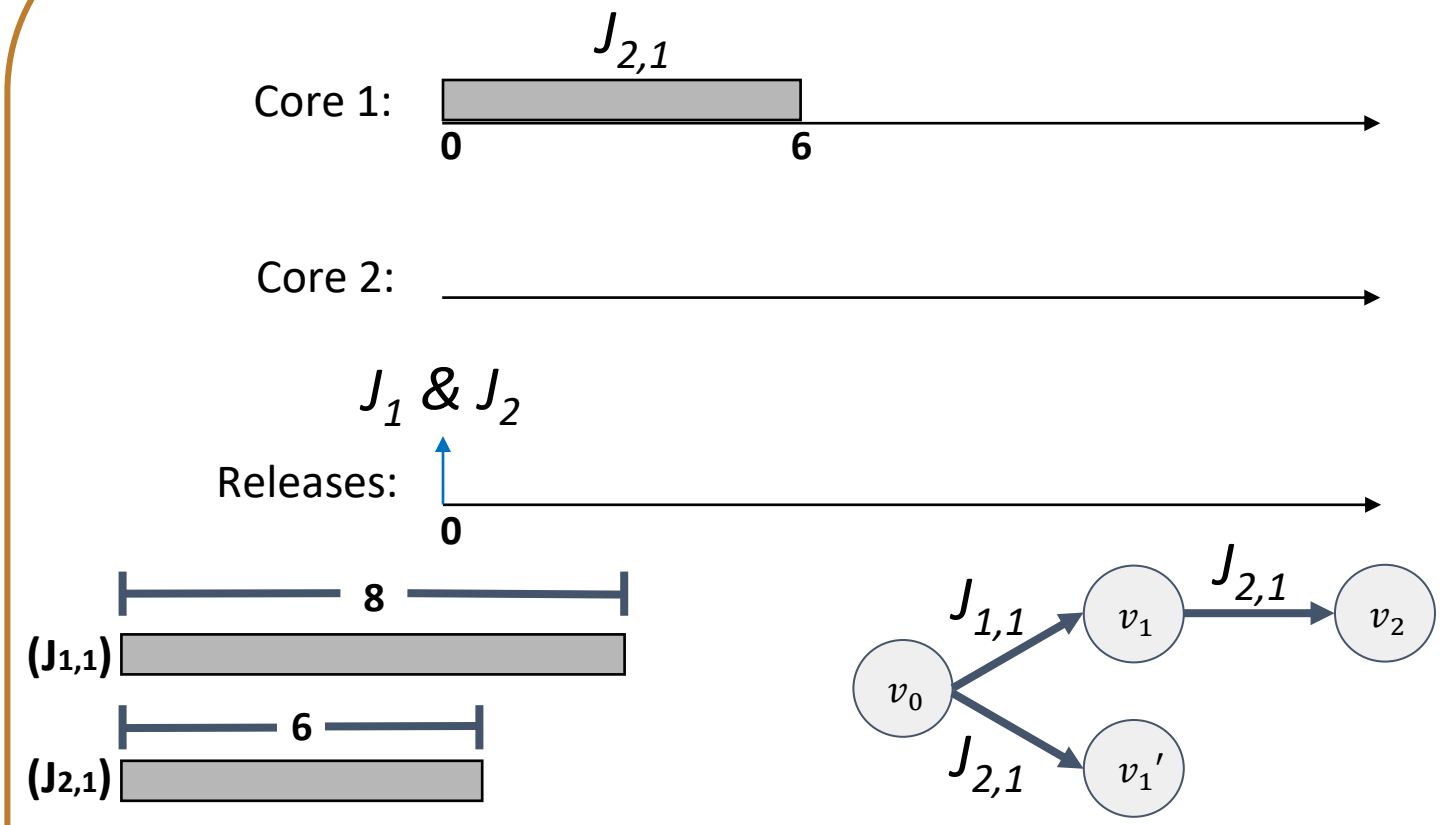
Partial order reduction

An example schedule:



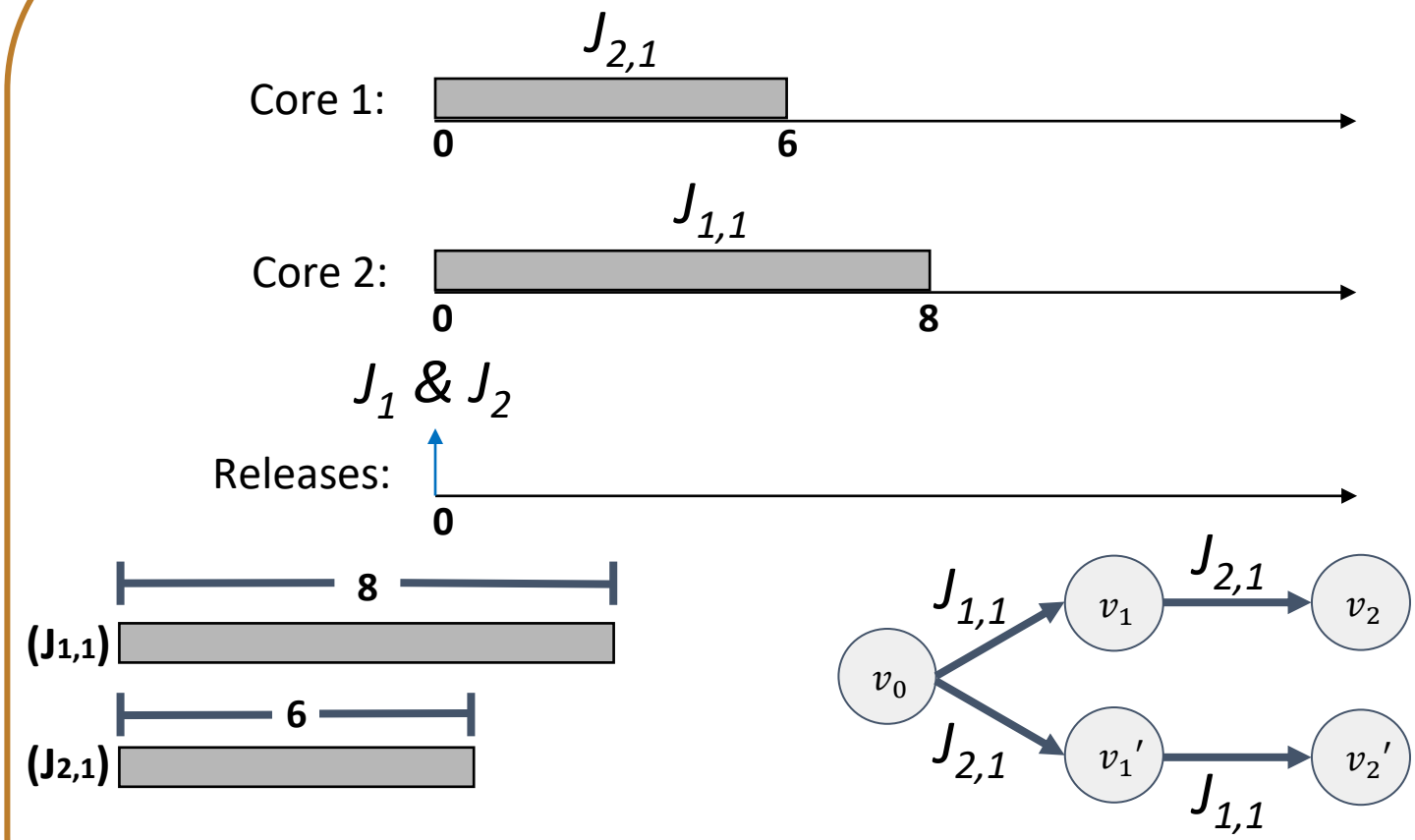
Partial order reduction

An example schedule:



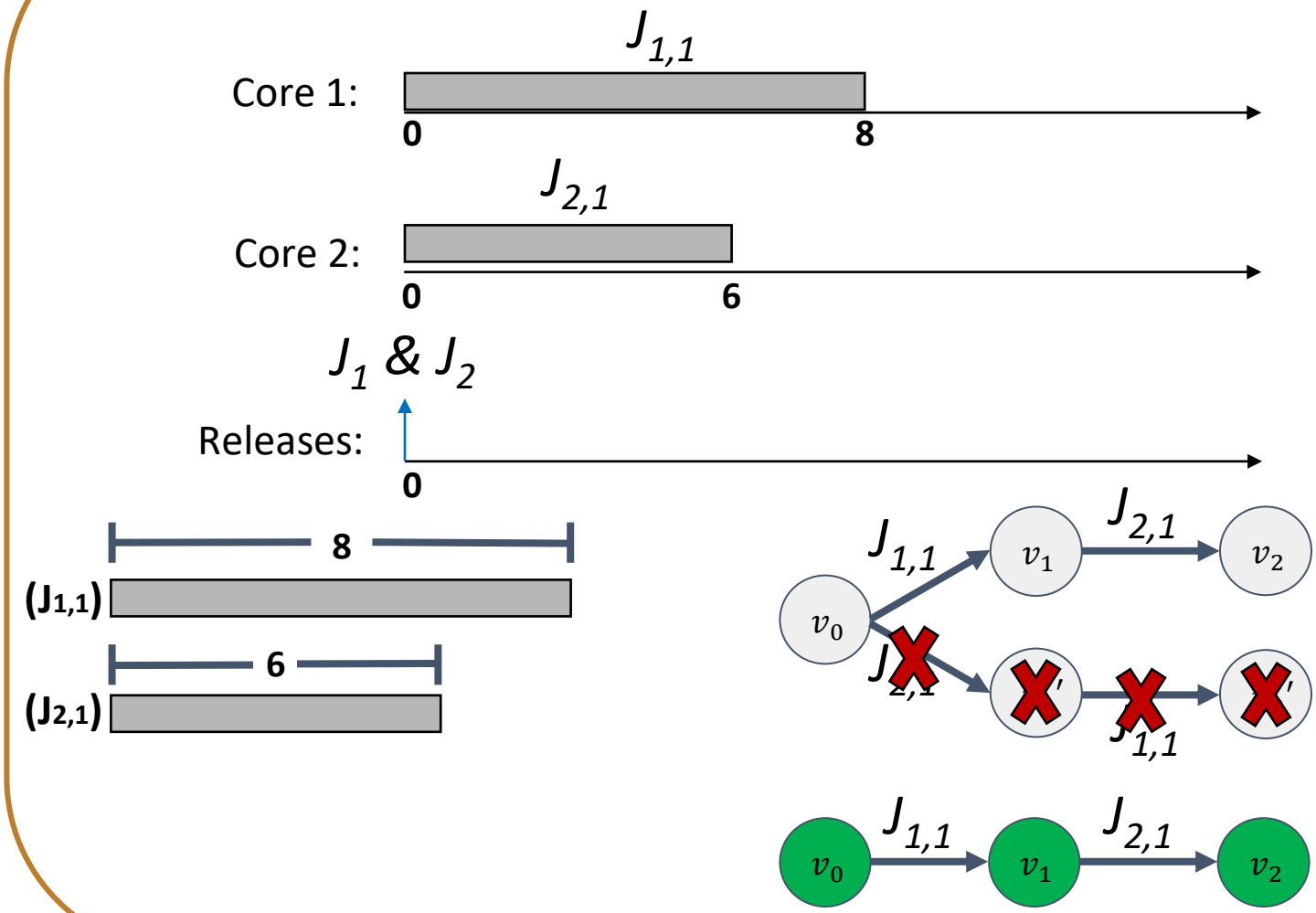
Partial order reduction

An example schedule:



Partial order reduction

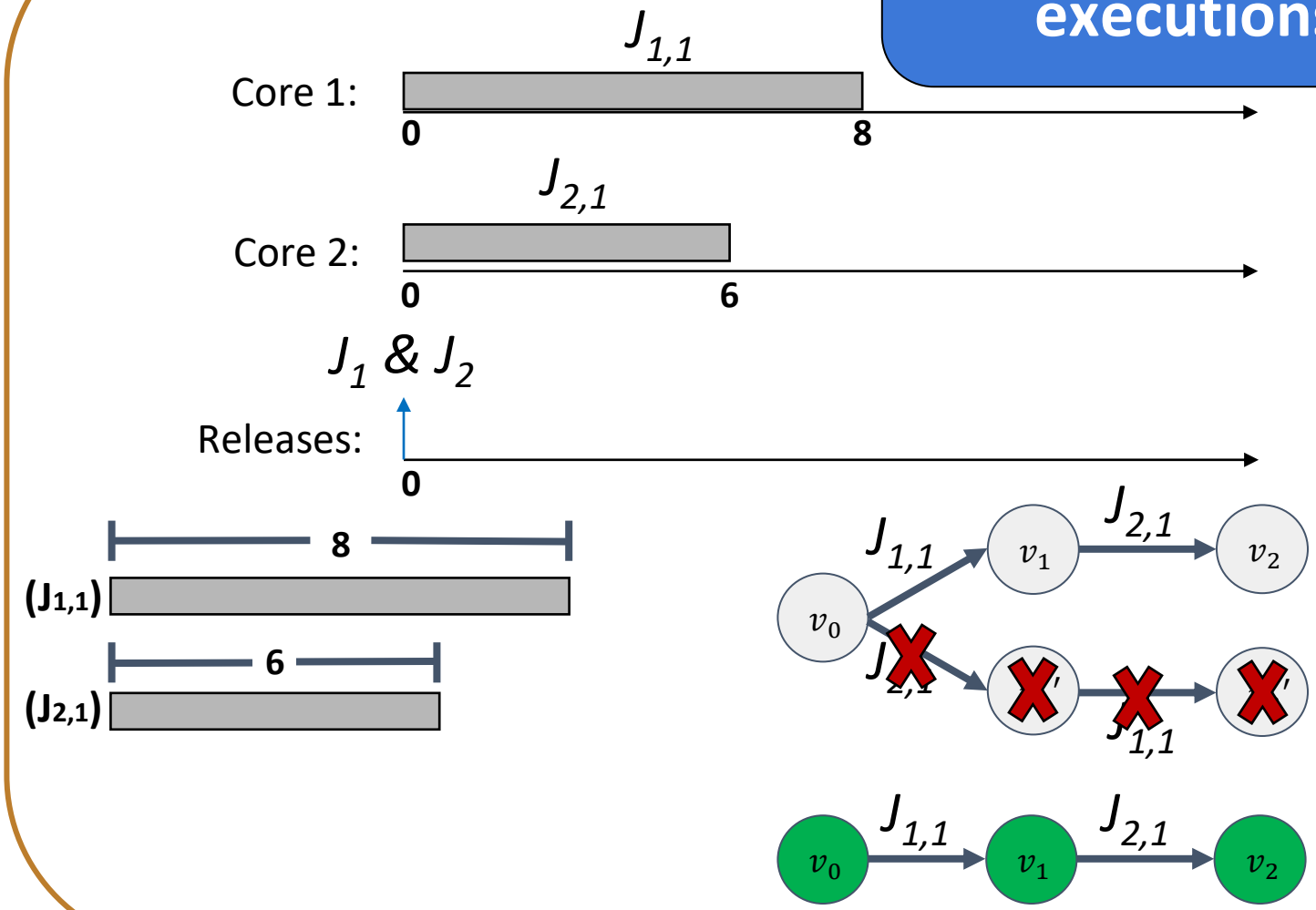
An example schedule:



Partial order reduction

As long as we do not affect other job executions!

An example schedule:



Partial order reduction

(1) Non-starting segments that don‘t access shared resources

→ Combined in workload representation

Partial order reduction

(1) Non-starting segments that don‘t access shared resources

→ Combined in workload representation

(2) Non-starting segments that access shared resources

Partial order reduction

(1) Non-starting segments that don‘t access shared resources

→ Combined in workload representation

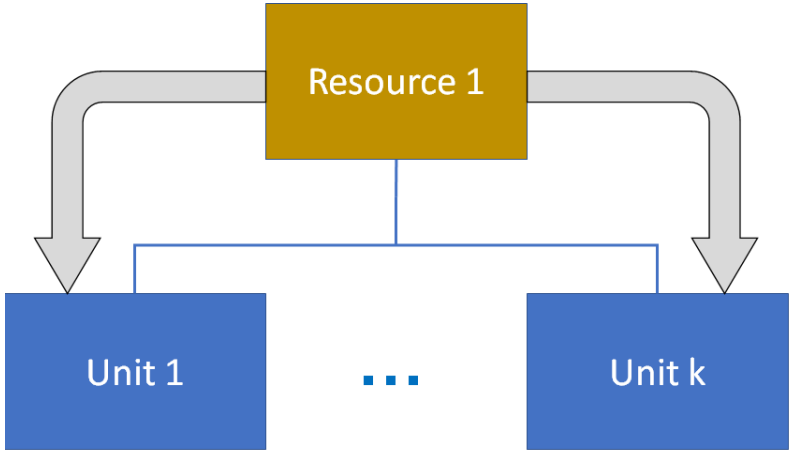
(2) Non-starting segments that access shared resources

(3) Starting segments that don‘t access shared resources

Multi-unit resources

What is it?

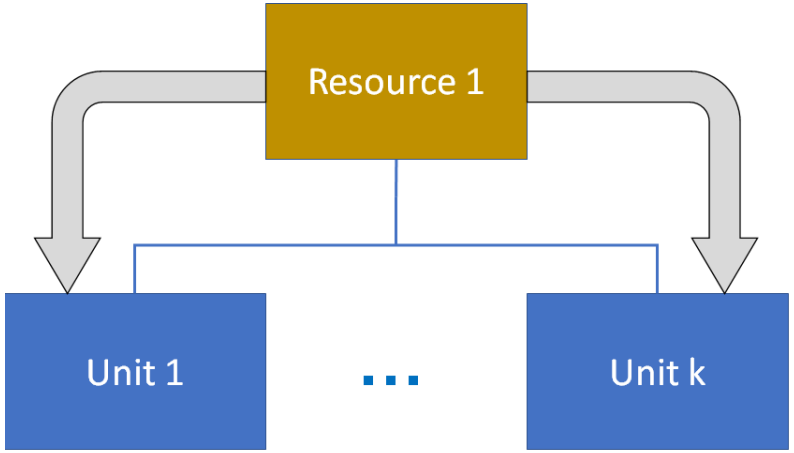
→ Shared resources that can be accessed by more than one task simulataneously



Multi-unit resources

What is it?

→ Shared resources that can be accessed by more than one task simulataneously



Why include it?

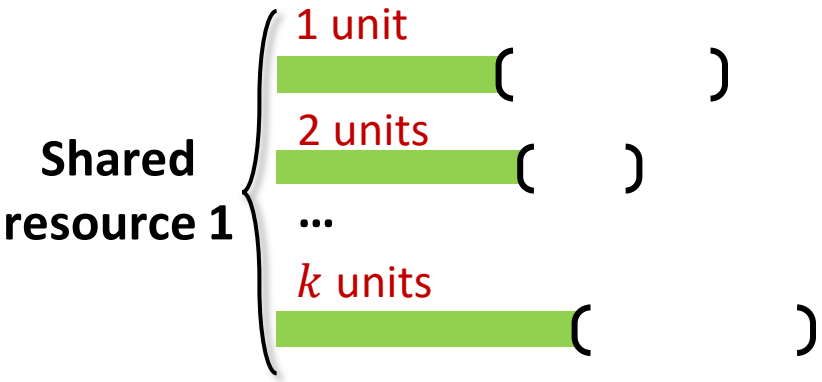
→ Common in real-time systems to find similar multiple units of the same resource

- *Communication channels*
- *I/O buffers*

Multi-unit resources

Idea:

→ Keep track when 1,2,…,k units of the same resource become(s) available



Summary & Conclusion

Summary

First blocking-aware response-time analysis for global non-preemptive systems that share resources via spin locks

Summary

First blocking-aware response-time analysis for global non-preemptive systems that share resources via spin locks

Extended the family of schedule-abstraction-based analysis to account for shared resources

Summary

First blocking-aware response-time analysis for global non-preemptive systems that share resources via spin locks

Extended the family of schedule-abstraction-based analysis to account for shared resources

Response-time analysis using schedule abstraction

- **+ New abstraction**

Summary

First blocking-aware response-time analysis for global non-preemptive systems that share resources via spin locks

Extended the family of schedule-abstraction-based analysis to account for shared resources

Response-time analysis using schedule abstraction

- **+ New abstraction**
- **+ Expansion and merge rules for shared resources supporting fifo- and priority-based spin locks**

Summary

First blocking-aware response-time analysis for global non-preemptive systems that share resources via spin locks

Extended the family of schedule-abstraction-based analysis to account for shared resources

Response-time analysis using schedule abstraction

- + New abstraction
- + Expansion and merge rules for shared resources supporting fifo- and priority-based spin locks
- + Extended the work via partial order reduction techniques and to support multi-unit resources

Conclusion

+ Our **scenario-aware SAG-based analysis** **significantly reduces pessimism** that exists in current analyses

Conclusion

- Our **scenario-aware SAG-based analysis** **significantly reduces pessimism** that exists in current analyses
- Our **scenario-aware SAG analysis** beats its inflation-based variant.

Conclusion

- Our **scenario-aware SAG-based analysis** **significantly reduces pessimism** that exists in current analyses
- Our **scenario-aware SAG analysis** beats its inflation-based variant
- Our **scenario-aware SAG analysis** **competes with and (sometimes even surpasses)** analyses considering a preemptive system

Conclusion

- Our **scenario-aware SAG-based analysis** **significantly reduces pessimism** that exists in current analyses
- Our **scenario-aware SAG analysis** beats its inflation-based variant
- Our **scenario-aware SAG analysis** **competes with and (sometimes even surpasses)** analyses considering a preemptive system
- **Opens further research paths** **in this direction**

Future work

- **Support for DAG tasks**
- **Support for nested locks**
- **Support for suspension-based locks**

Response-time analysis considering **FIFO-ordered spin locks**

→ **CAPITAL**

→ **RTSS**

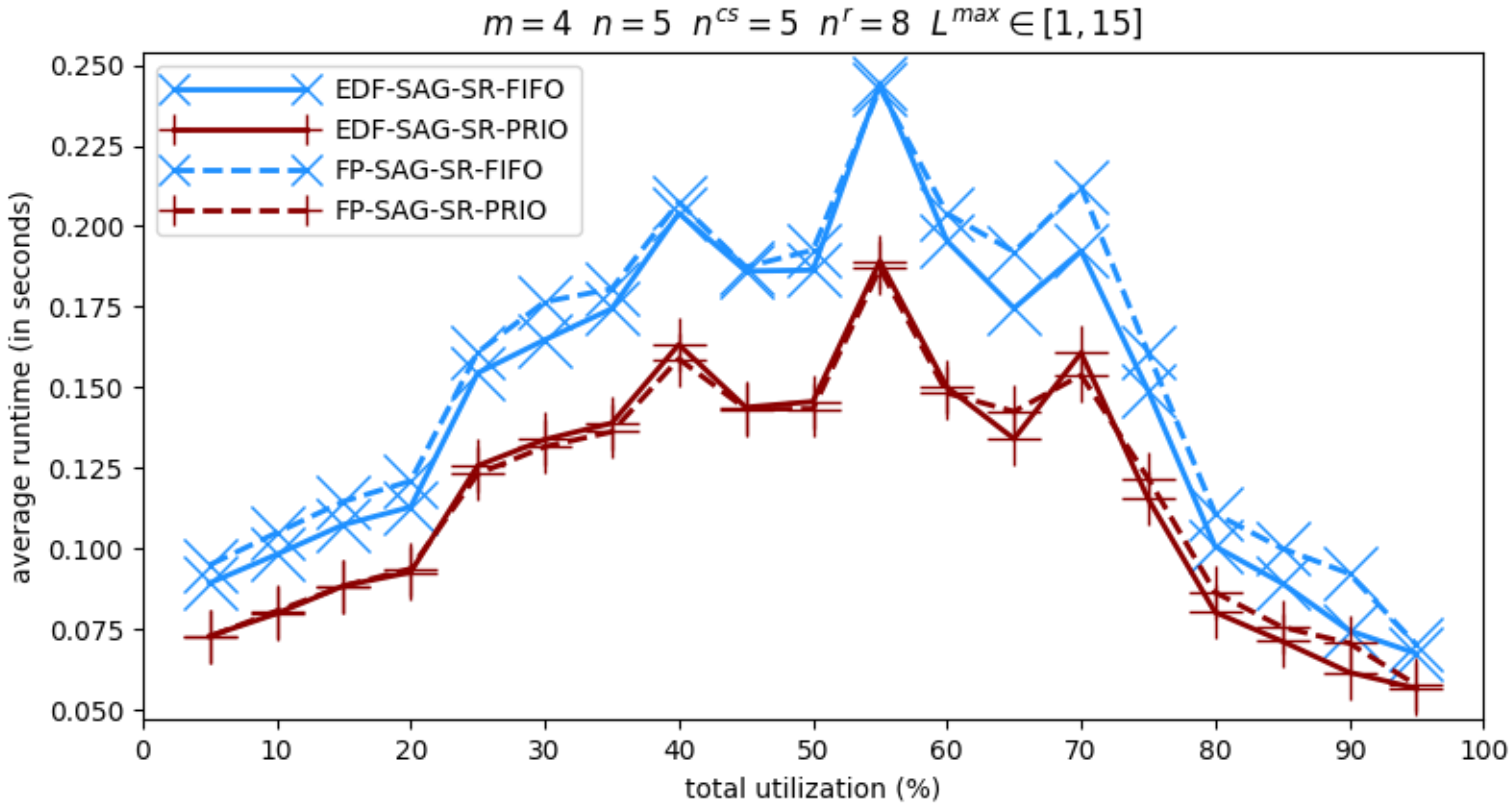
Response-time analysis considering **priority-ordered spin locks**

- Support for **multi-unit resources & k-exclusion protocols**
- **Additional techniques** to improve scalability

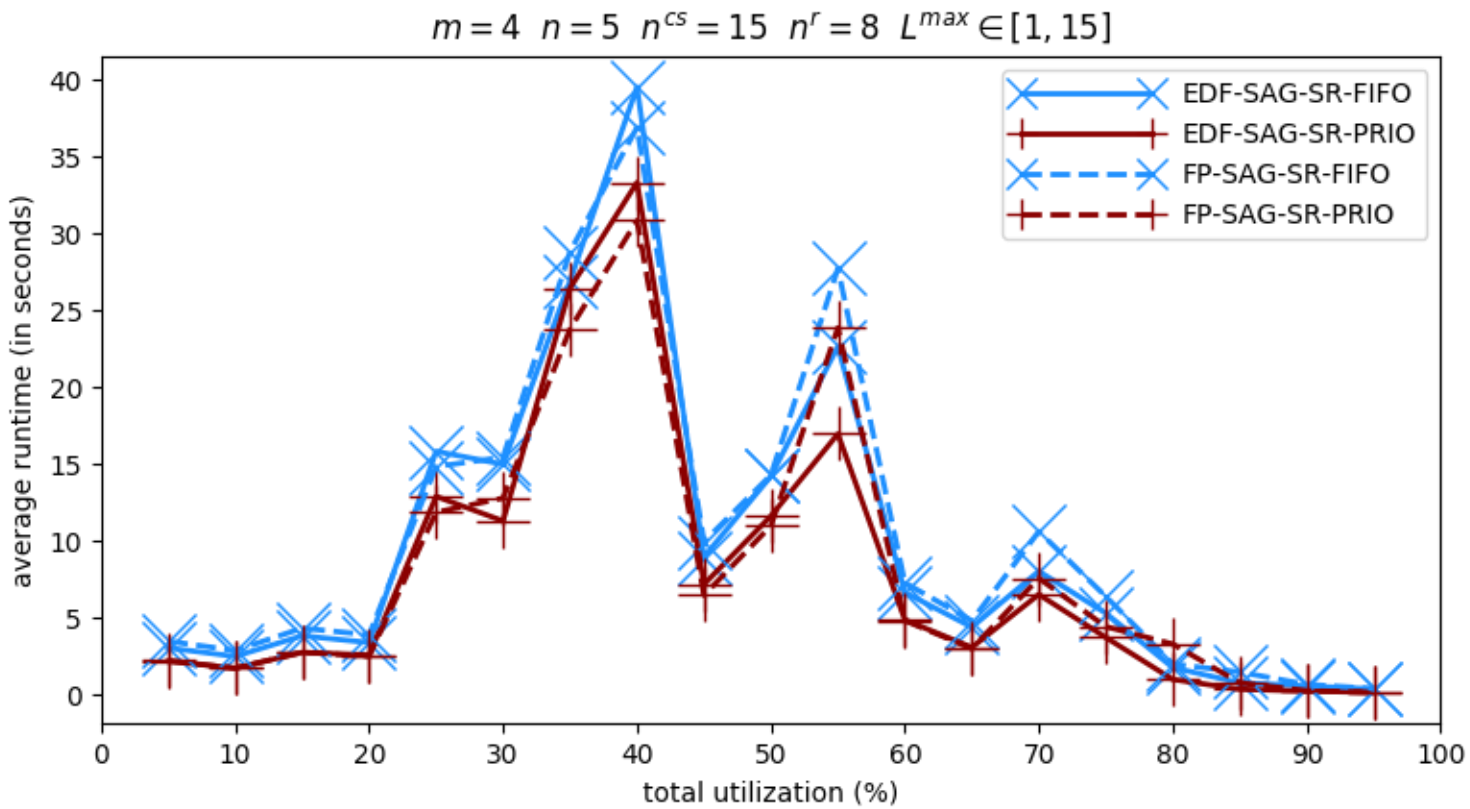
TU Delft

Backup slides

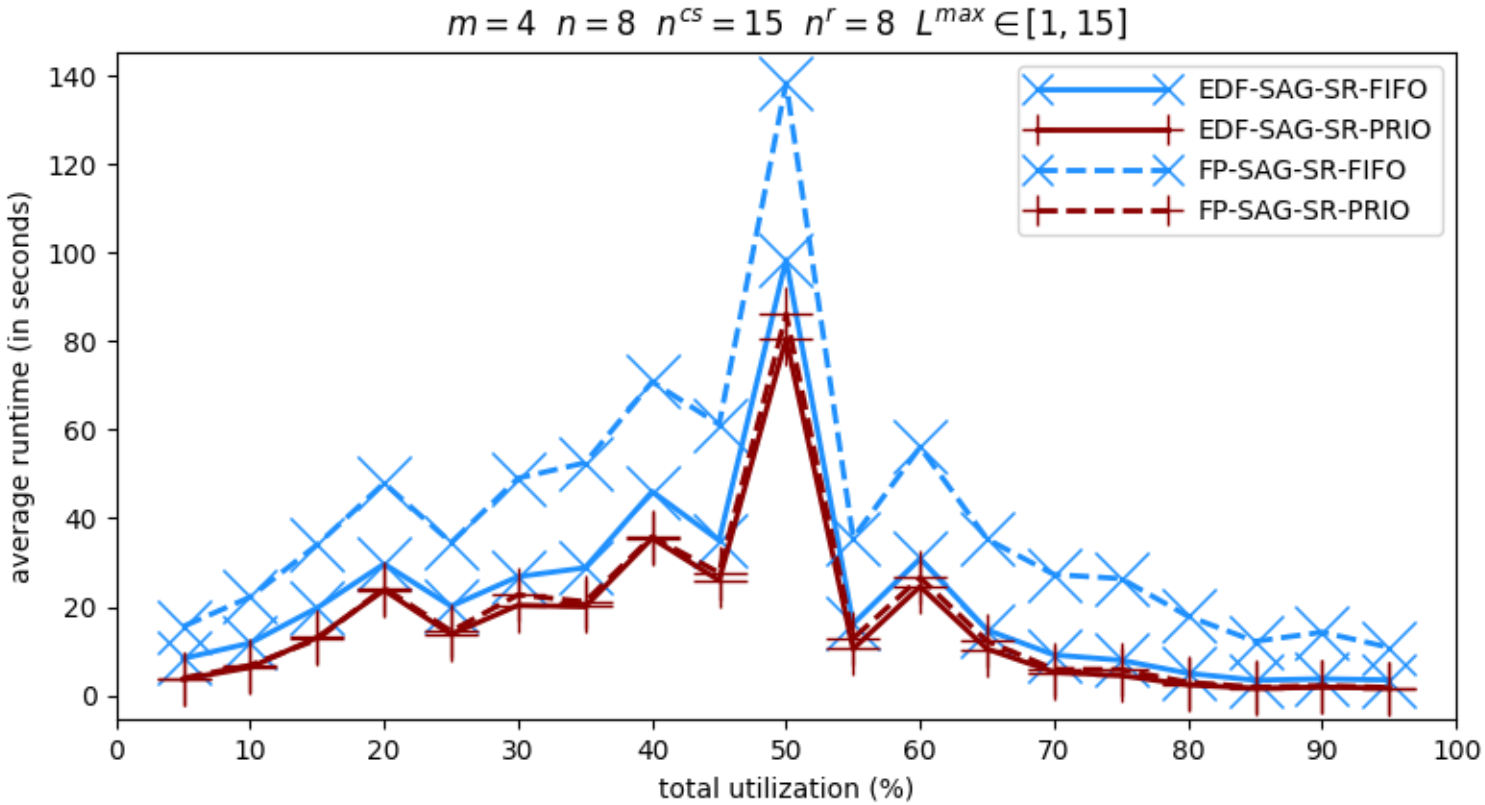
Runtime results



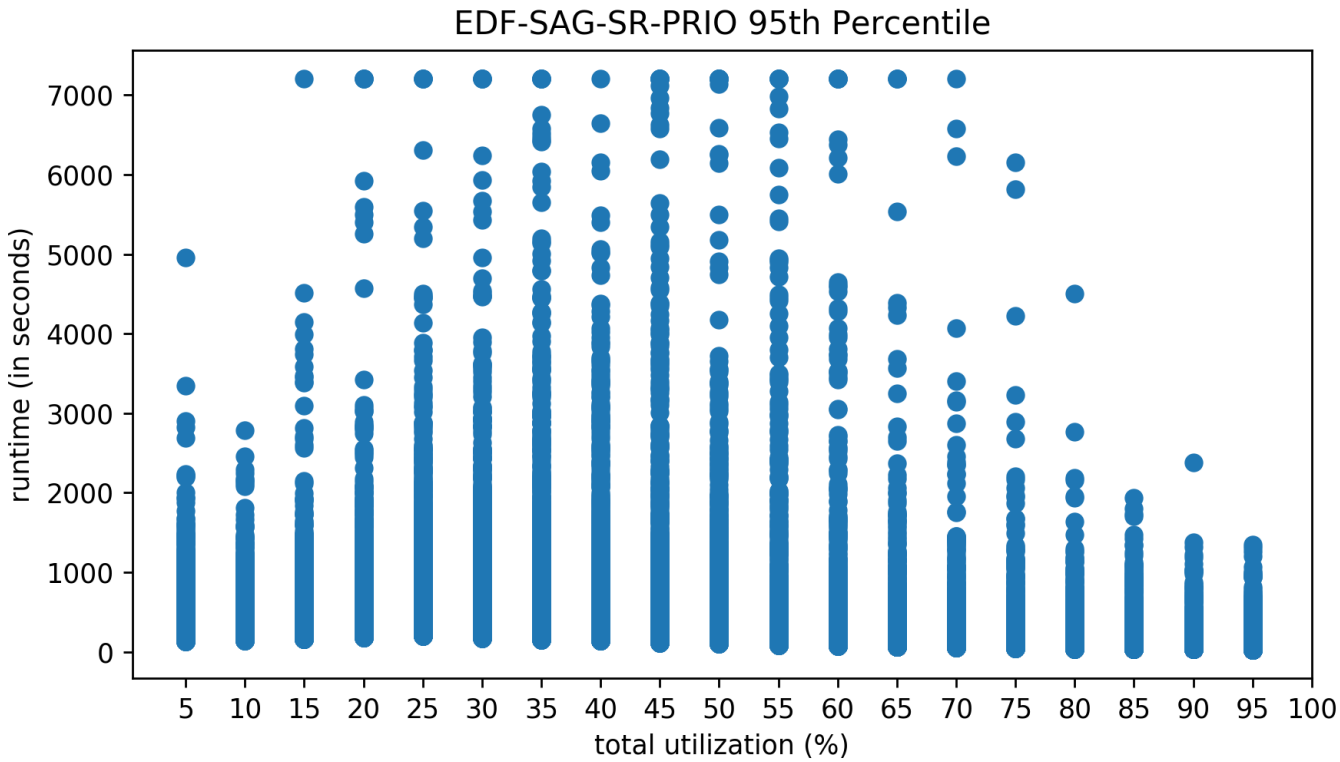
Runtime results



Runtime results



Runtime results



Runtime results