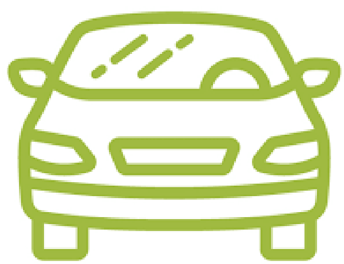
Schedulability analysis of globally scheduled preemptive applications

Srinidhi Srinivasan

Supervisor: Dr. Geoffrey Nelissen
Co-supervisor: Dr. Mitra Nasri

Safety critical systems

Safety-critical systems → failure could result in catastrophic damage



Even under the worst case scenarios, the timing constraints of the system are met.

Worst-case response time (WCRT) < Deadline ✓ → Schedulable

What is the problem being tackled?

Workload Model

Periodic preemptive tasks

Platform Model

Multiprocessor platform

Scheduler Model

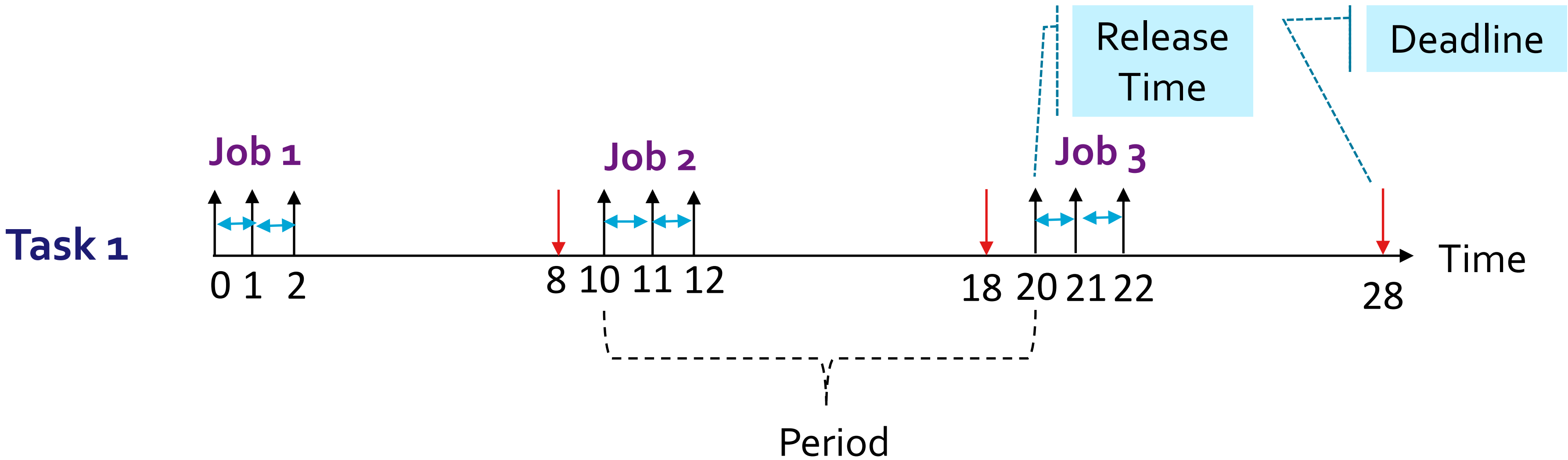
Global job-level-fixed-priority

How to find the worst-case response time (WCRT) ?

What are tasks?

Tasks ⟶ A task is a piece of code that implements one of the system functionalities .

Task ID	Execution time variation [Min,Max]	Release jitter	Period	Deadline	Offset
1	[4,5]	1	10	8	1



Jobs in an observation window

Job 1

Job 2

Job 3

.

.

.

Job n

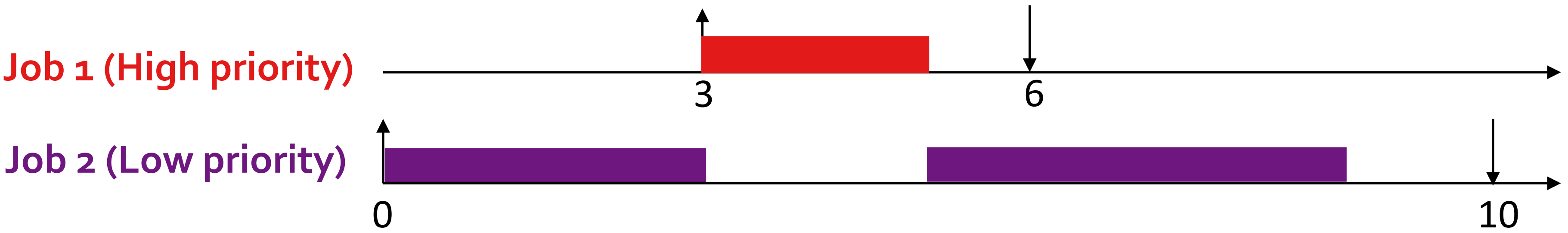
Observation window

Smallest duration of time after which the schedules formed by these jobs repeat themselves

Hyperperiod –
lowest common multiple of the periods of all tasks

What are preemptive tasks/jobs?

A preemptive task/ job → Execution can be interrupted



Global job level fixed priority scheduler

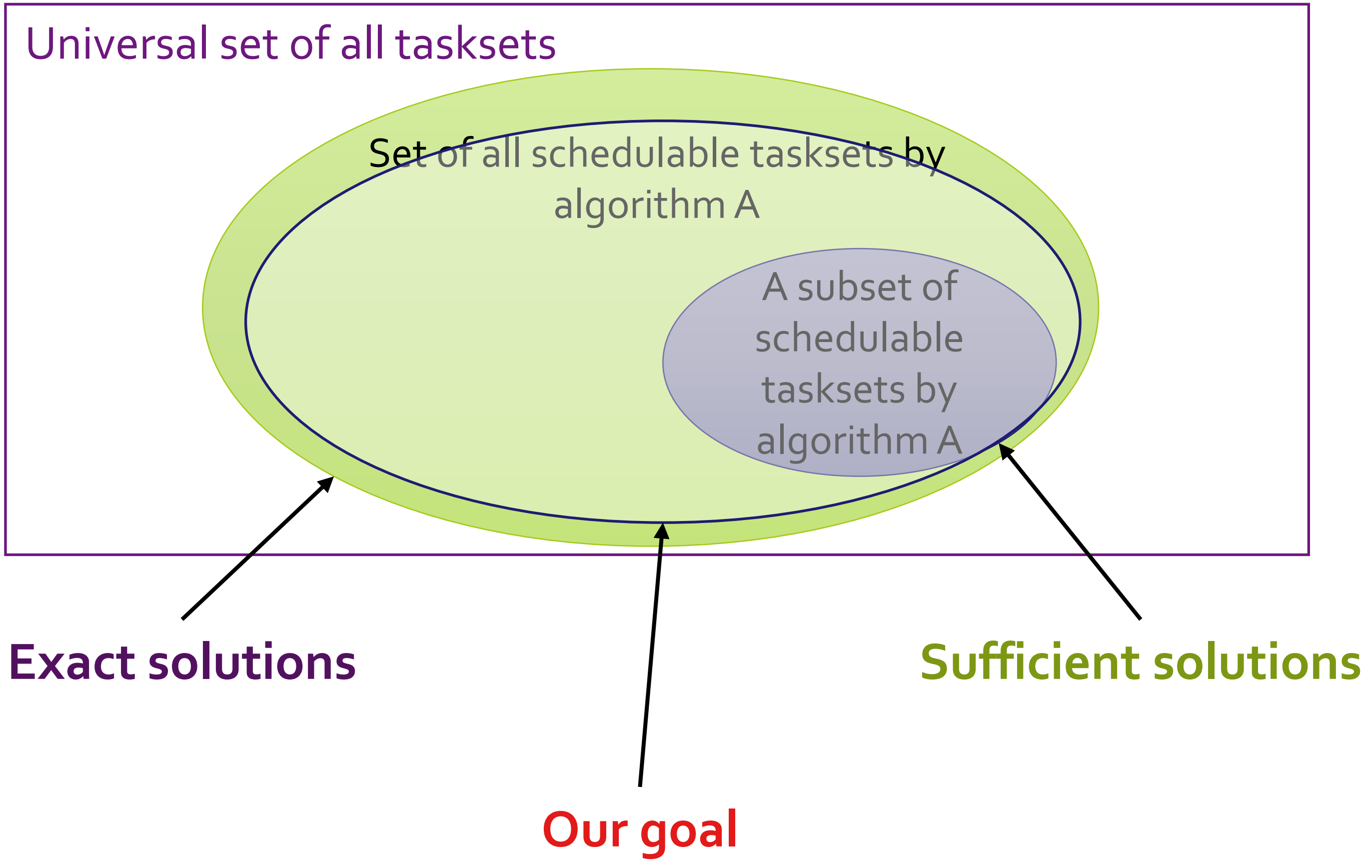
Job level fixed priority scheduler →

- Priority is assigned to each job based on parameters like period or deadline
- Many algorithms follow such a scheduler:
  - Rate-Monotonic
  - Deadline-Monotonic
  - Earliest Deadline First
  - etc.

What is the goal?

Build a schedulability analysis technique
for preemptive tasks
scheduled by a job-level fixed priority scheduler
on a multiprocessor platform

Sufficient and exact solutions



State of the art – Multiprocessor case

Brute-force determination of multiprocessor schedulability for sets of sporadic hard-deadline tasks

T. P. Baker and M. Cirinei , 2007

+ Exact by using finite state machines

– Only if period of task is 3,4 or 5

– State space explosion occurs very soon

A. Burmyakov, E. Bini, and E. Tovar, "An exact schedulability test for global FP using state space pruning"

V. Bonifaci and A. Marchetti-Spaccamela, "Feasibility analysis of sporadic real-time multiprocessor task systems"

} **Scalability issues**

State of the art – Multiprocessor case

Uniprocessor platform with preemptive tasks

Response-Time Analysis (RTA) (Audsley et al., 1995)

Multiprocessor platform with preemptive tasks

- Marko Bertogna and Michele Cirinei. 2007. "Response-time analysis for globally scheduled symmetric multiprocessor platforms"
- Nan Guan, Martin Stigge, Wang Yi, and Ge Yu. 2009. "New response time bounds for fixed priority multiprocessor scheduling"
- Many more…

\+ Can handle multiple types of tasksets

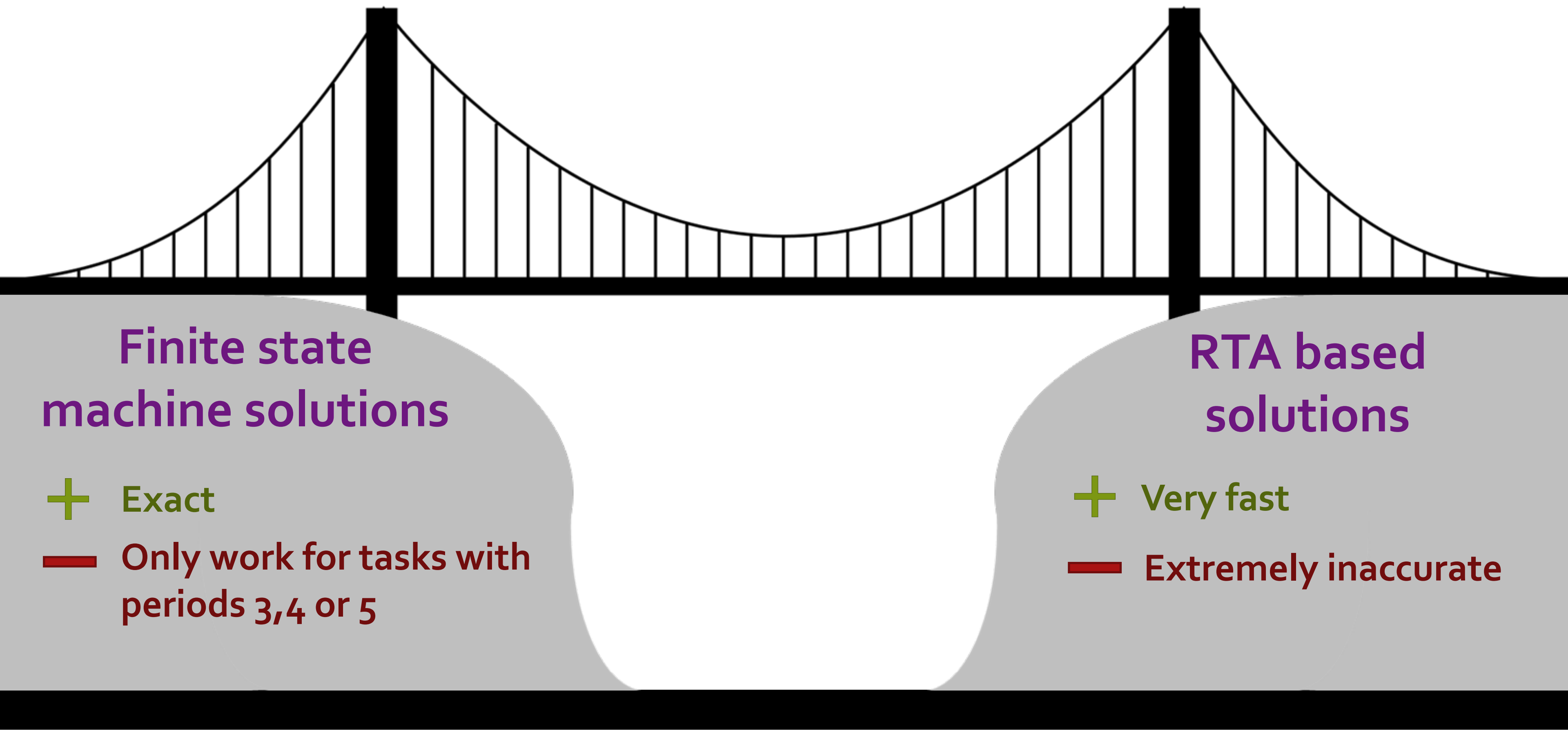
\+ Very fast in analysis

– A sufficient solution

– Highly inaccurate

Build a solution that bridges the gap

- ✓ Accurate
- ✓ Ability to scale to realistic system sizes



Schedule abstraction graph[1]

What is it?

- Non-preemptive jobsets
- Considers job orderings of schedules to build a schedule abstraction graph

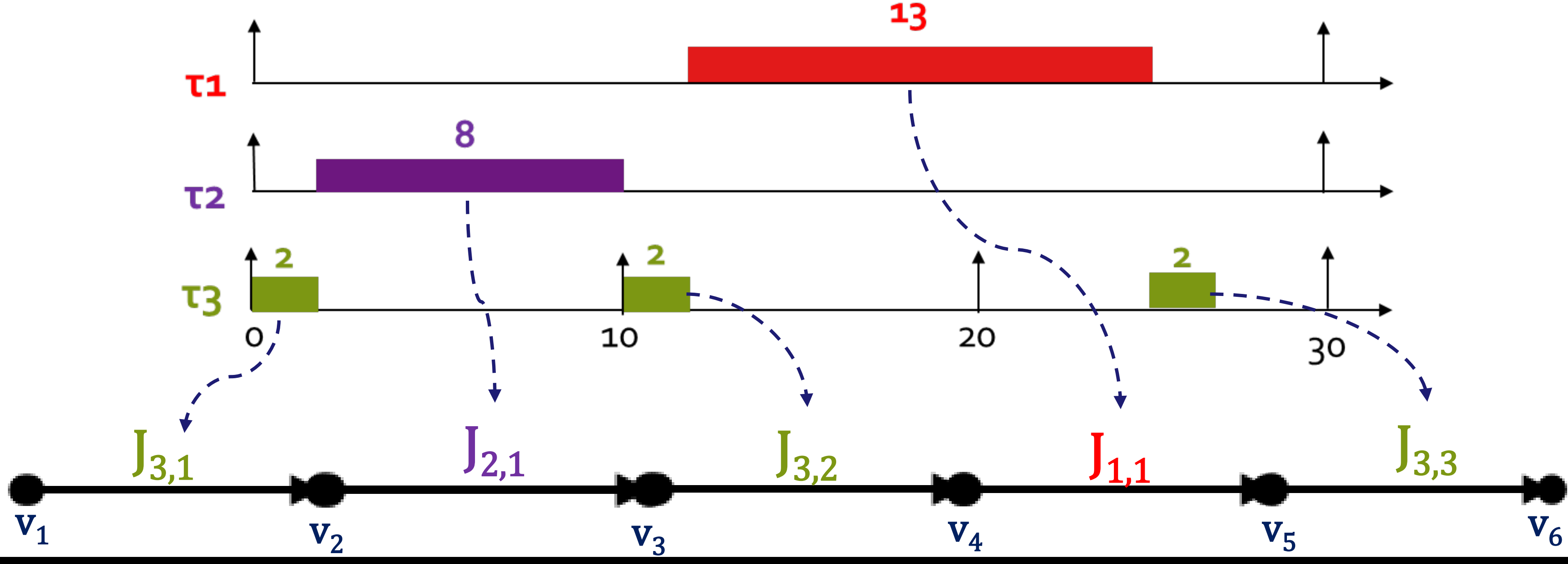
Why does it provide hope?

- **New technique**
- Performs **simulation of events** instead of simulation of time
- Proven to behave much **better than RTA based solutions** for non-preemptive tasks

[1] M. Nasri and B. B. Brandenburg, "An Exact and Sustainable Analysis of Non-preemptive Scheduling," *2017 IEEE Real-Time Systems Symposium (RTSS)*, Paris, 2017, pp. 12-23

Schedule abstraction graph for non-preemptive (Example)

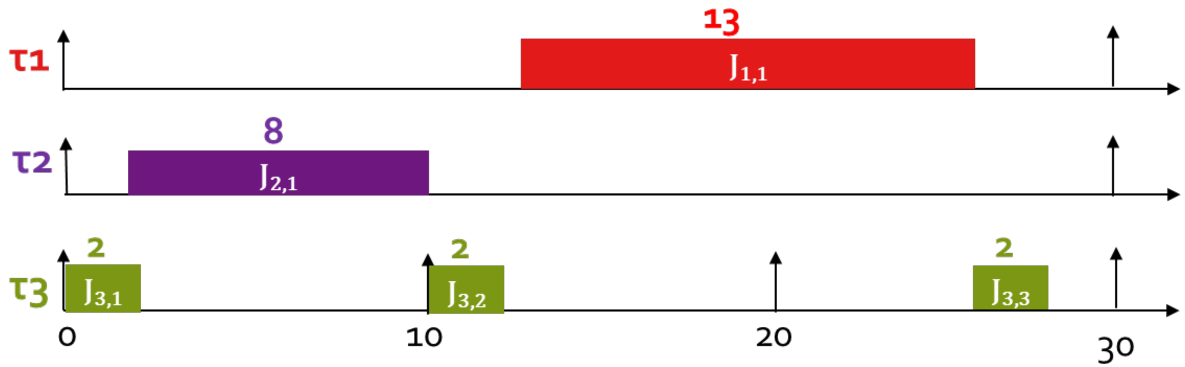
Task	Period	Execution time [Min,Max]
τ1	30	[3,13]
τ2	30	[7,8]
τ3	10	[1,2]

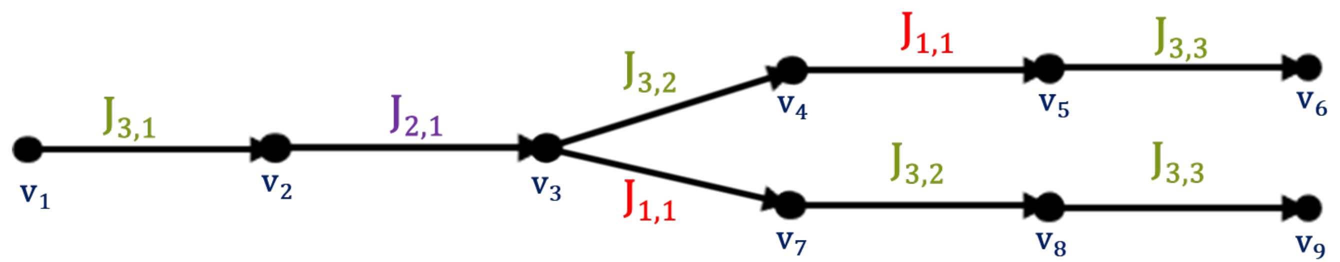


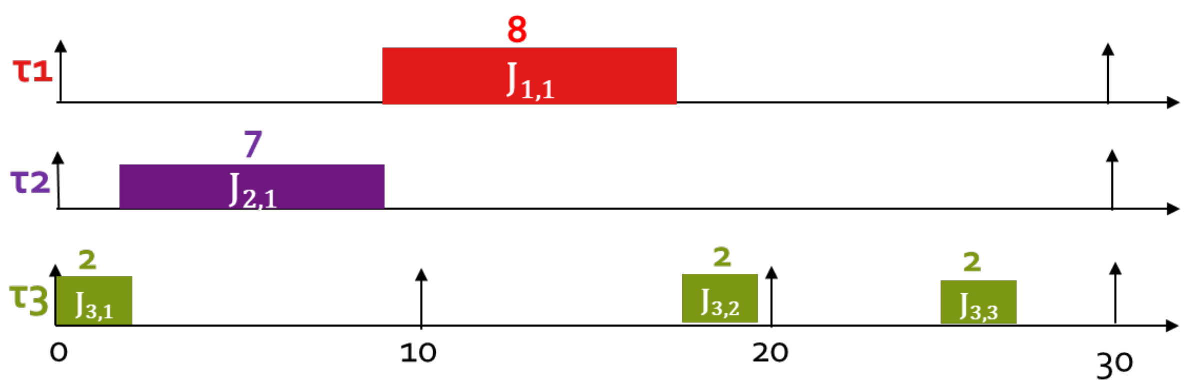
TUDelft

Schedule abstraction graph for non-preemptive (Example)

Task	Execution Time	Priority
τ1	[3,13]	Low
τ2	[7,8]	Medium
τ3	[1,2]	High

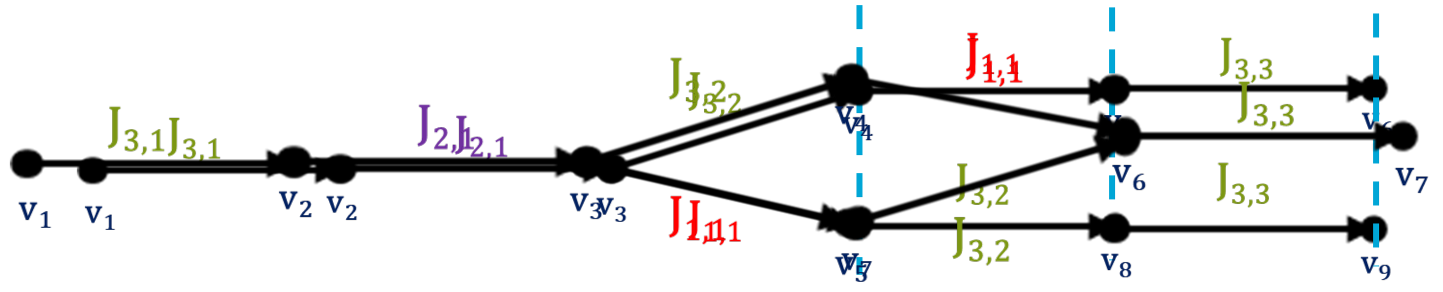


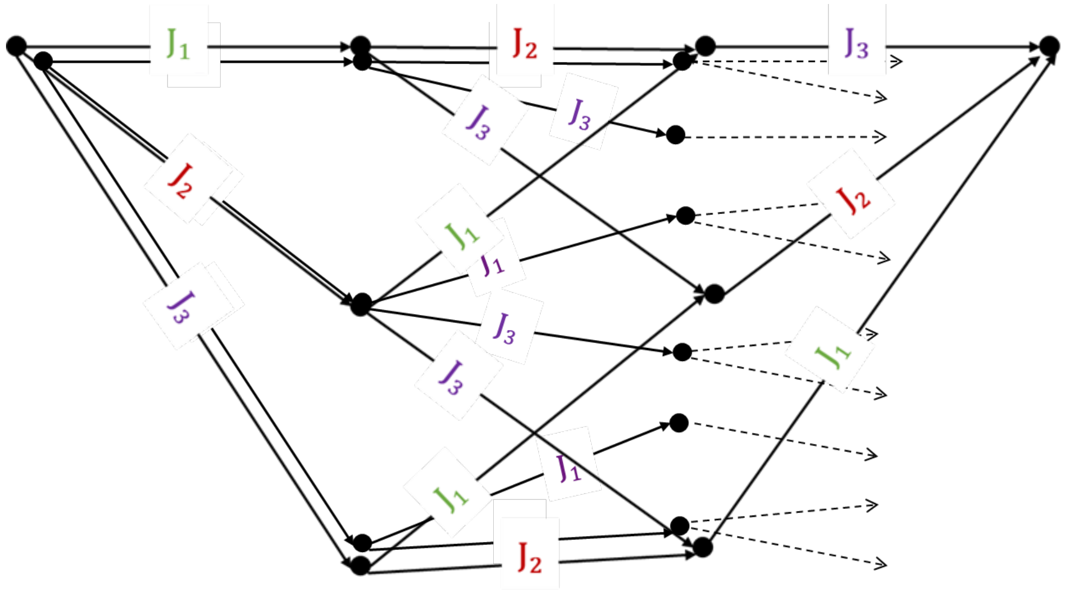




TUDelft

Schedule abstraction graph for non-preemptive (Example)





Why are preemptive tasks difficult to analyze?

Find worst-case response time (WCRT)

Non-preemptive tasks

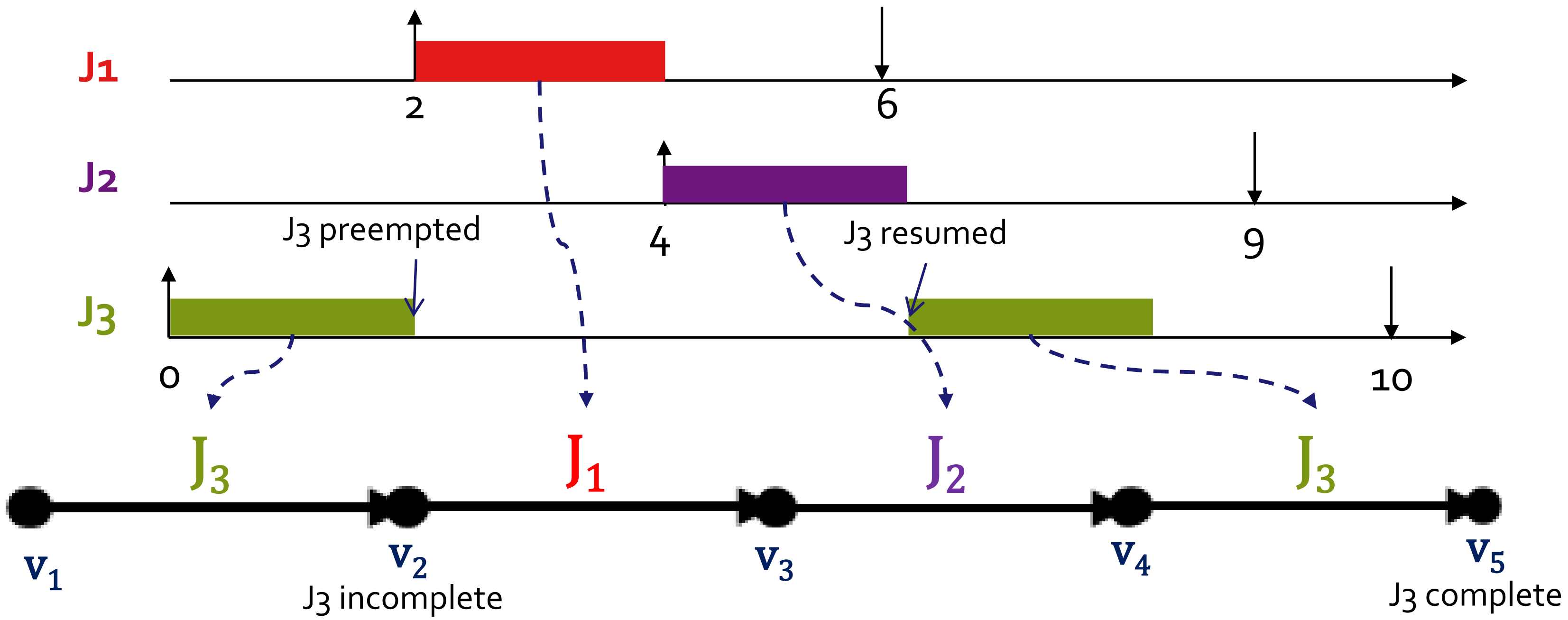
- Calculate when the job can start executing
- WCRT = Start time + Execution time

Preemptive tasks

- Calculate when the job can start executing
- **Keep track of all the preemptions that the job undergoes**
- Determine the WCRT

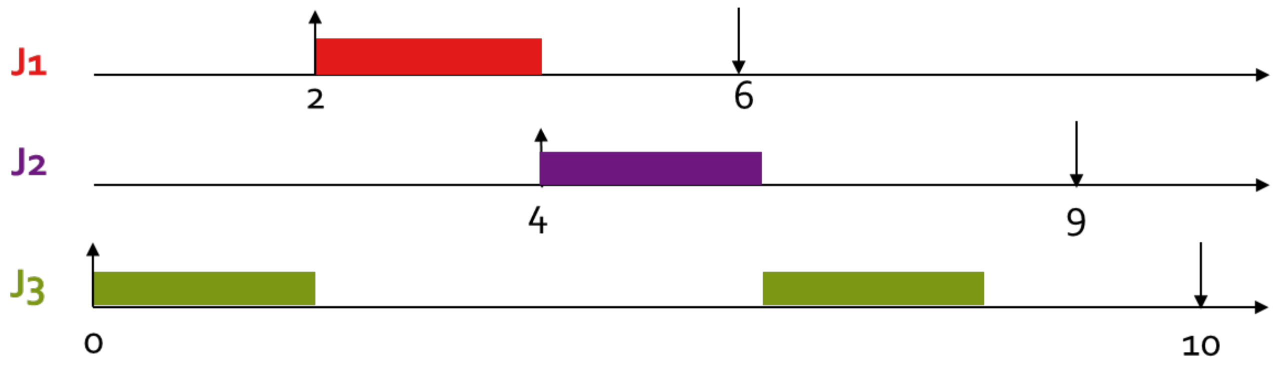
Naïve extension

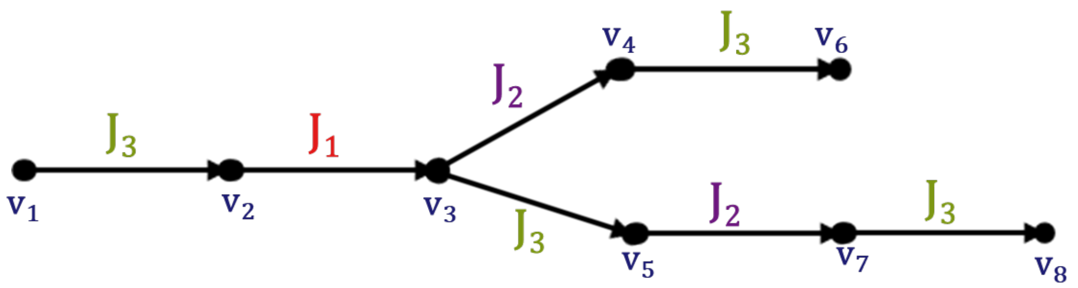
Job	Release Time [Min,Max]	Deadline	Execution Time	Priority
J1	[2,2]	6	2	High
J2	[4,5]	9	2	Medium
J3	[0,0]	10	4	Low



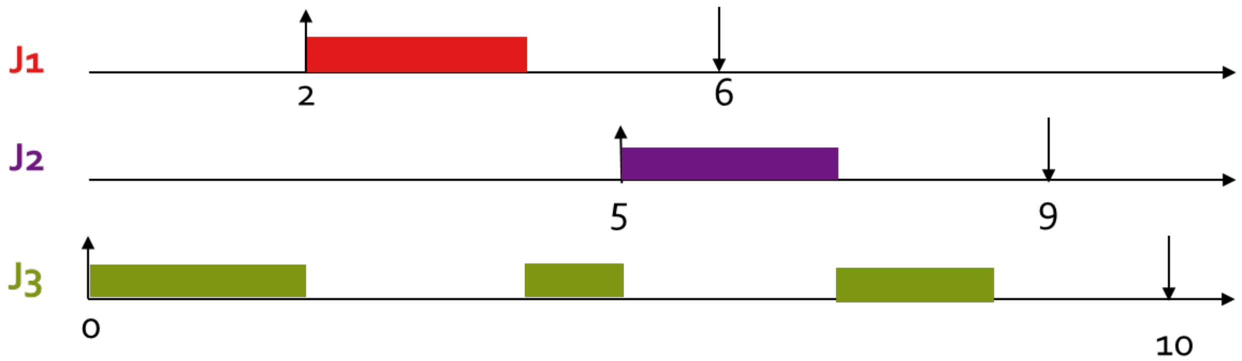
Issues with the naïve extension

When **J2** releases at 4

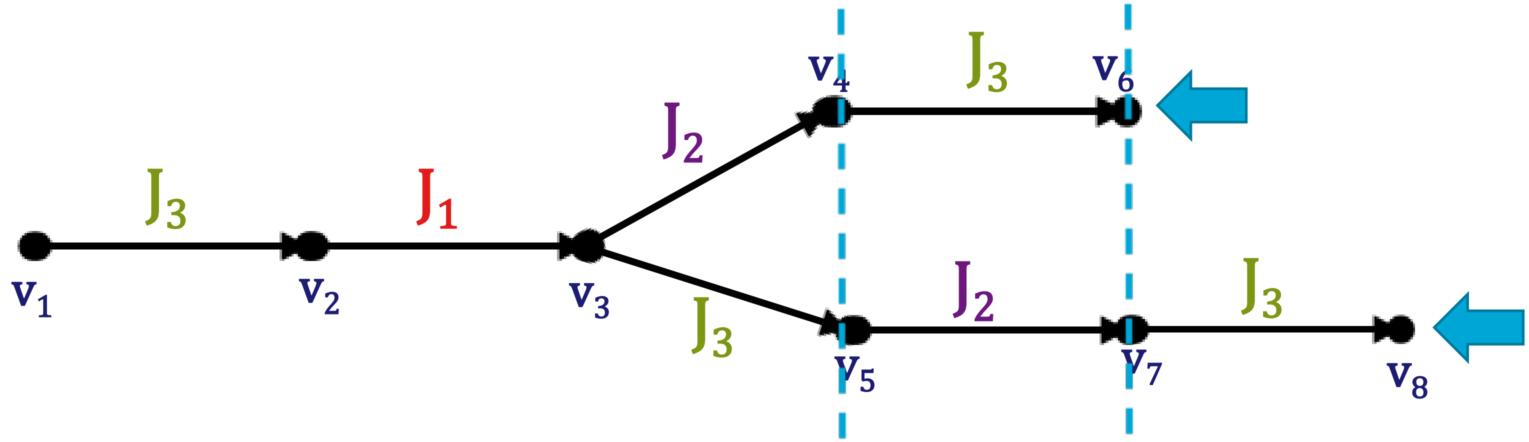




When **J2** releases at 5



Issues with the naïve extension



Not being able to merge → State space explosion → Build a new solution with changed semantics

Our solution

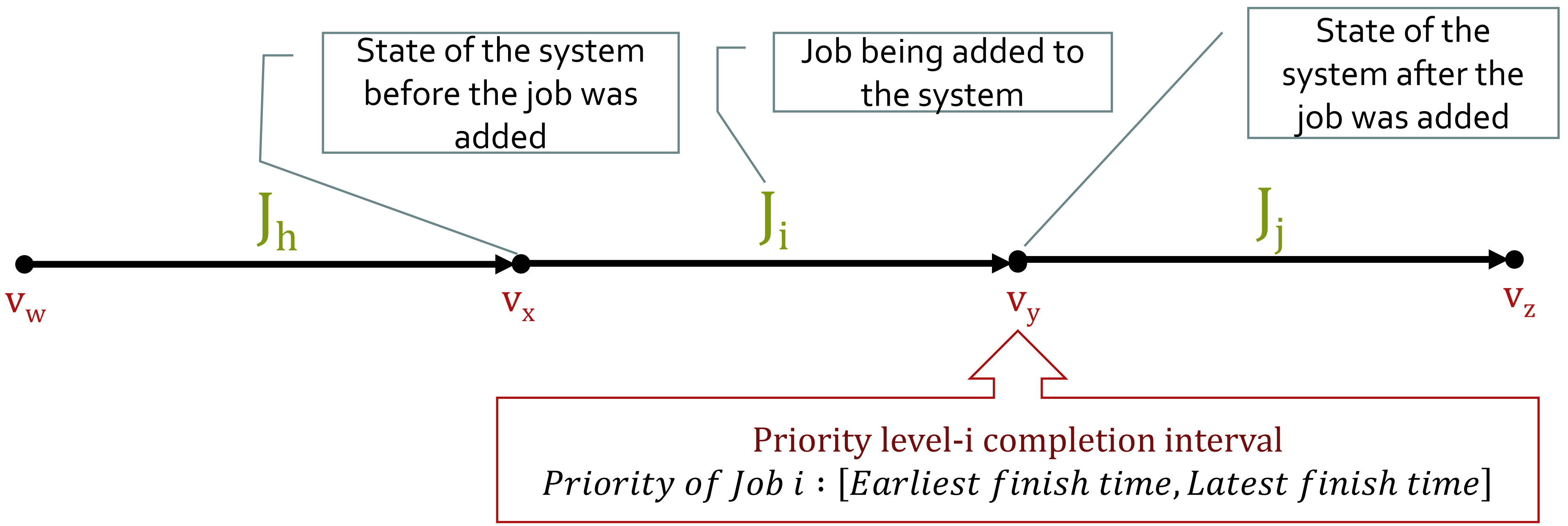
Preemptive tasks on a **uniprocessor** platform

→

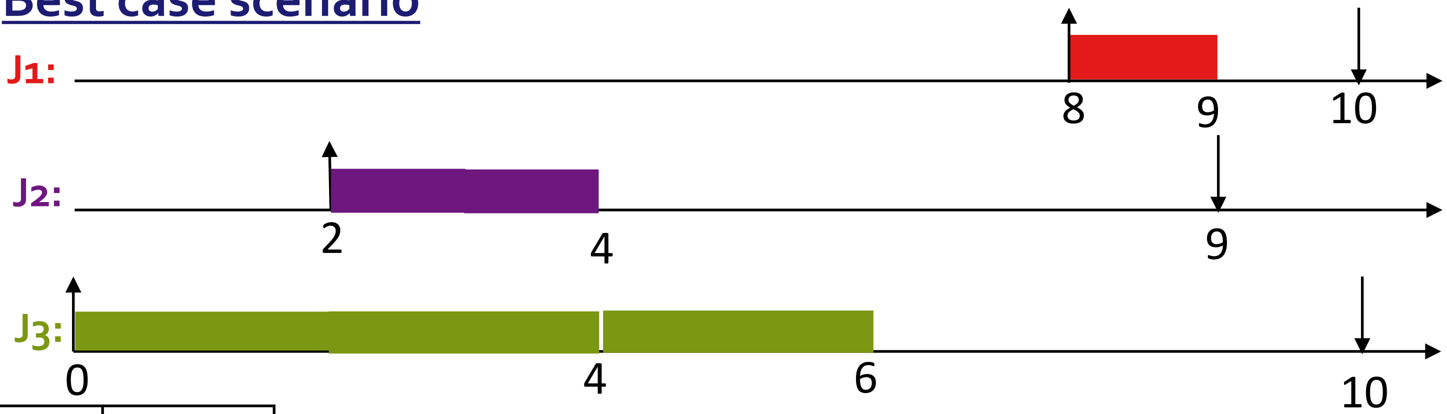
Extend to **multiprocessor** platform

Our solution - Uniprocessor

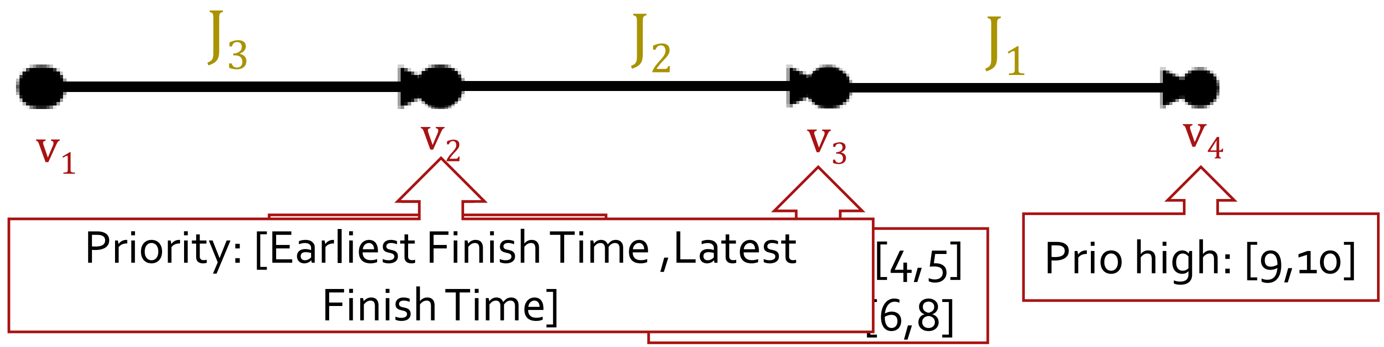
A taskset → Find jobs in a observation window → Sort jobs based on earliest release and add jobs onto the graph → Calculate worst-case response time from the graph



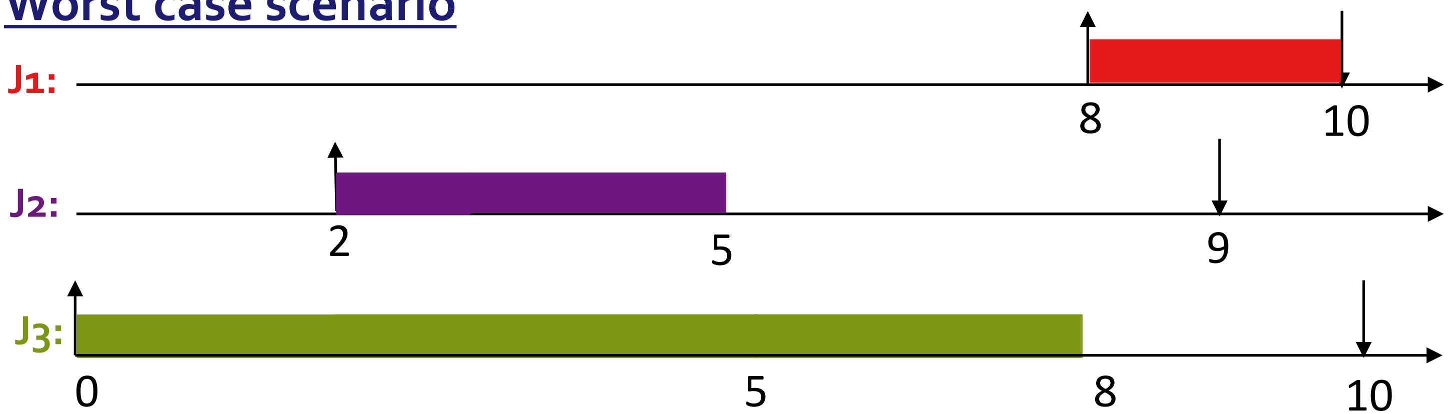
Best case scenario

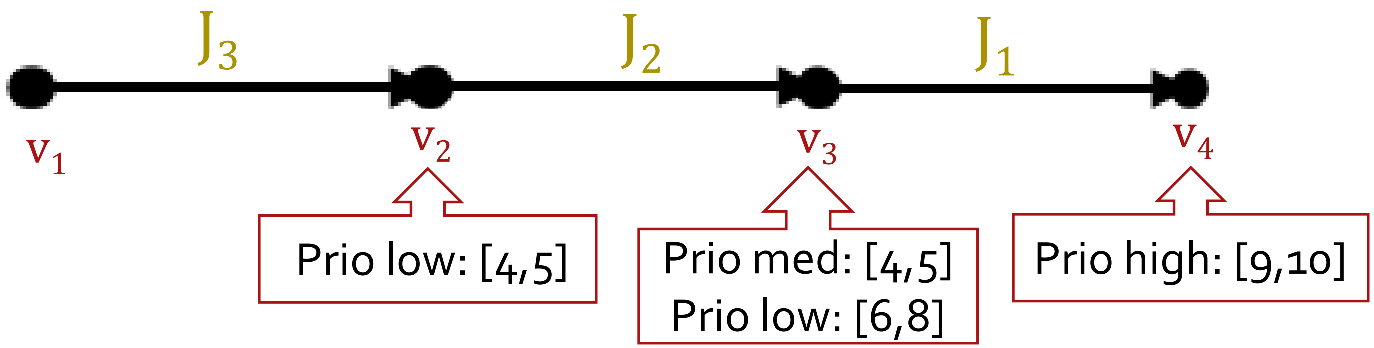


Job	Execution Time	Priority
J1	[1,2]	High
J2	[2,3]	Medium
J3	[4,5]	Low



Worst case scenario

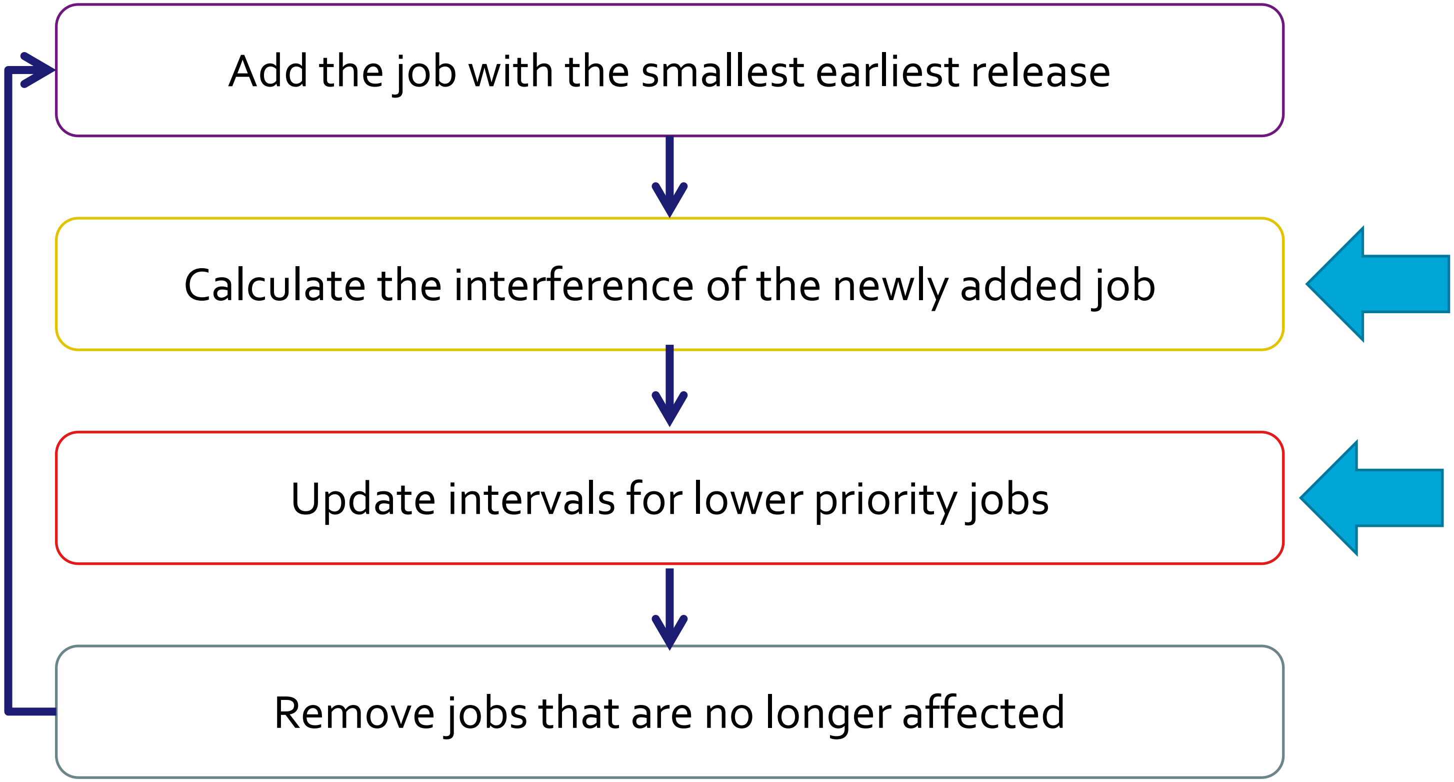




Jobs	Priority	Latest finish time	Deadline
J1	High	<=	10
J2	Medium	<=	9
J3	Low	<=	10

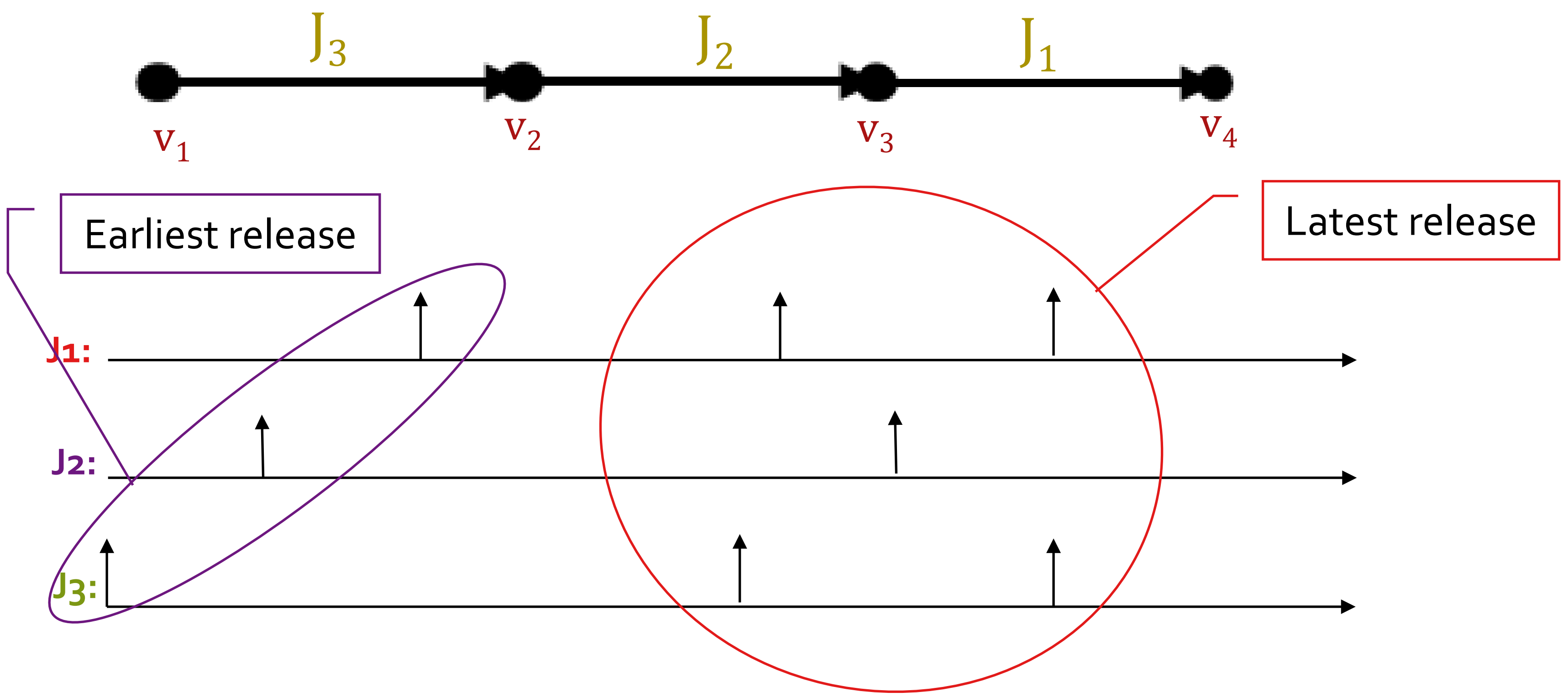
Taskset → **Schedulable** ✓

Analytical view of our solution



Calculating maximum interference

Calculating maximum interference → Depends on latest release



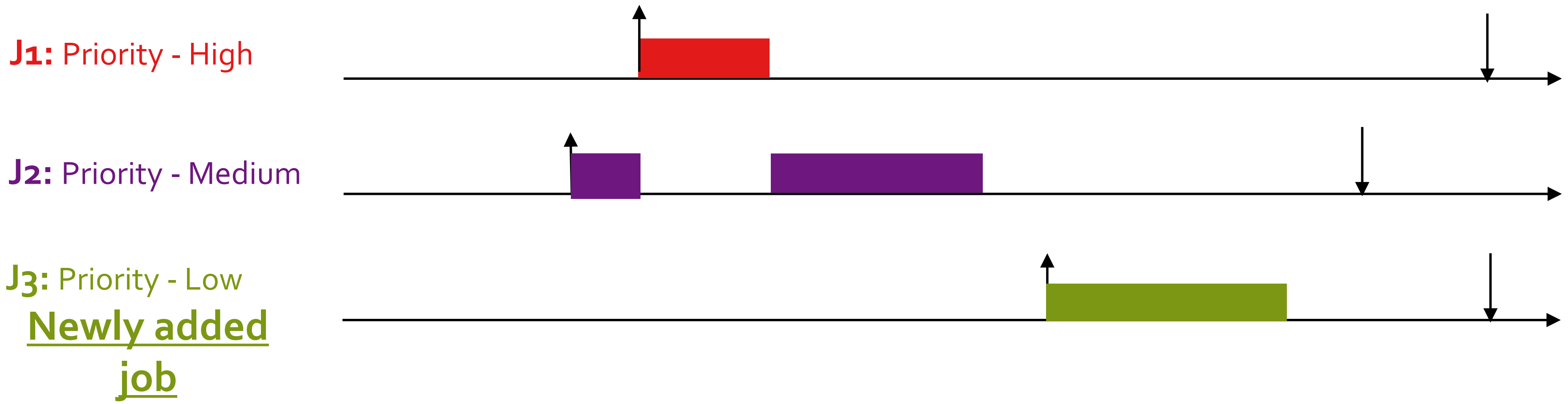
Analytical view of our solution

1. Add the job with the smallest earliest release
2. Calculate the interference of the newly added job
3. Update intervals for lower priority jobs
4. Remove jobs that are no longer affected

Adding a job

Scenario 1:

When latest release of J3 is after the latest finish of all the higher priority jobs

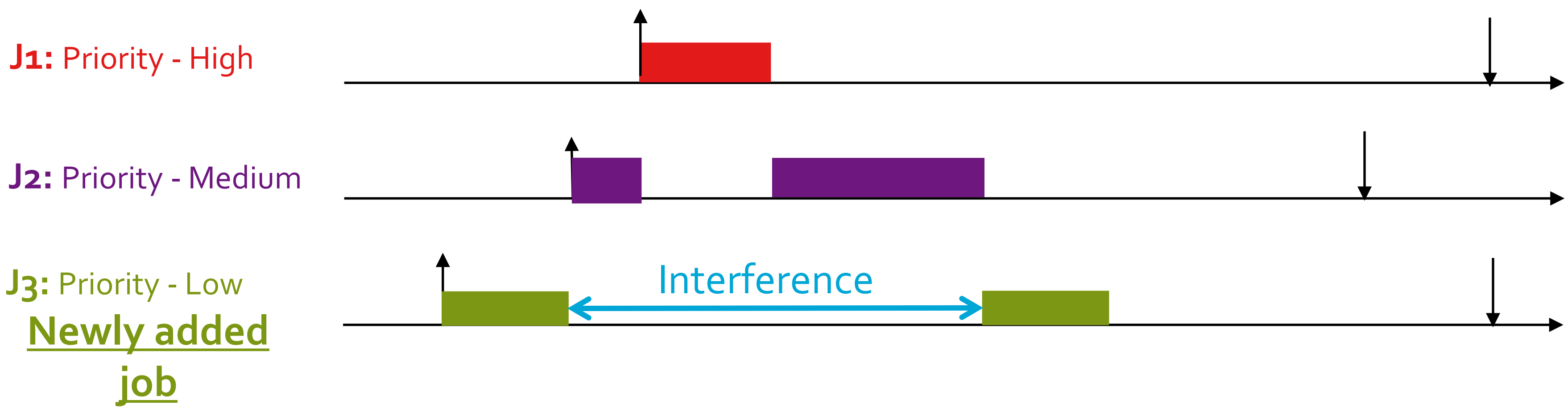


Interference(J3) = 0

Adding a job

Scenario 2:

When latest release of J3 is before the latest release of all the higher priority jobs



Interference(J3) = Sum of largest execution time of J1 and J2

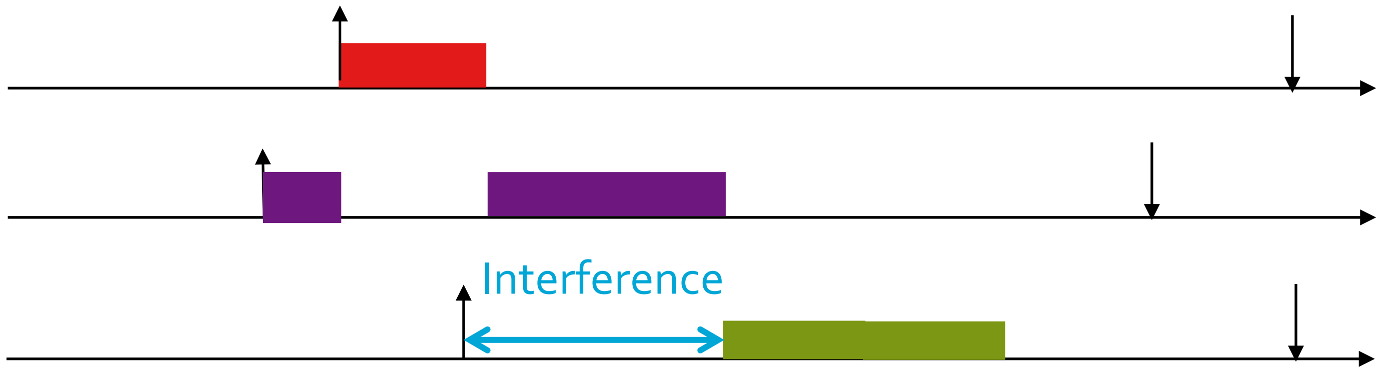
Adding a job

Scenario 3:
In all other scenarios

J1: Priority - High

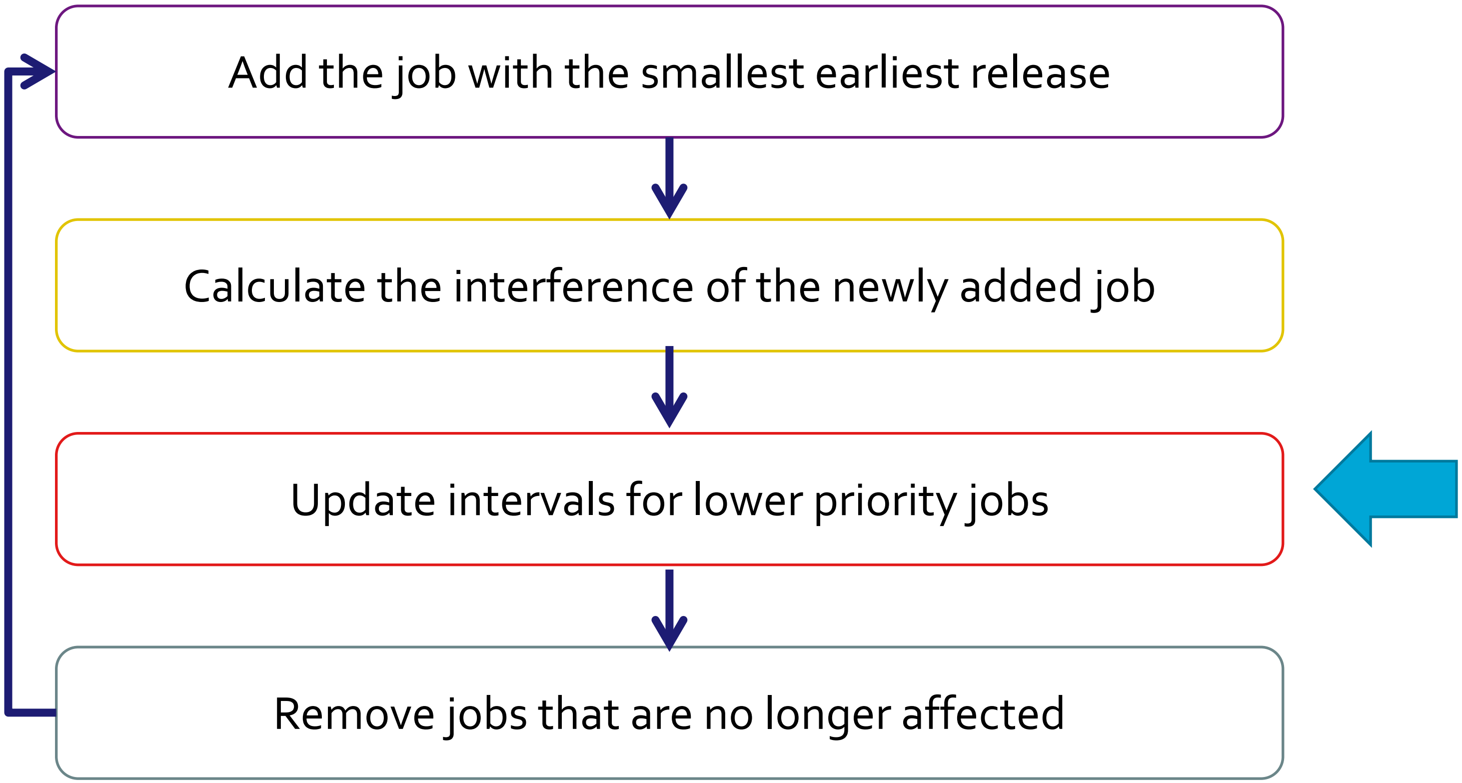
J2: Priority - Medium

J3: Priority - Low
Newly added job



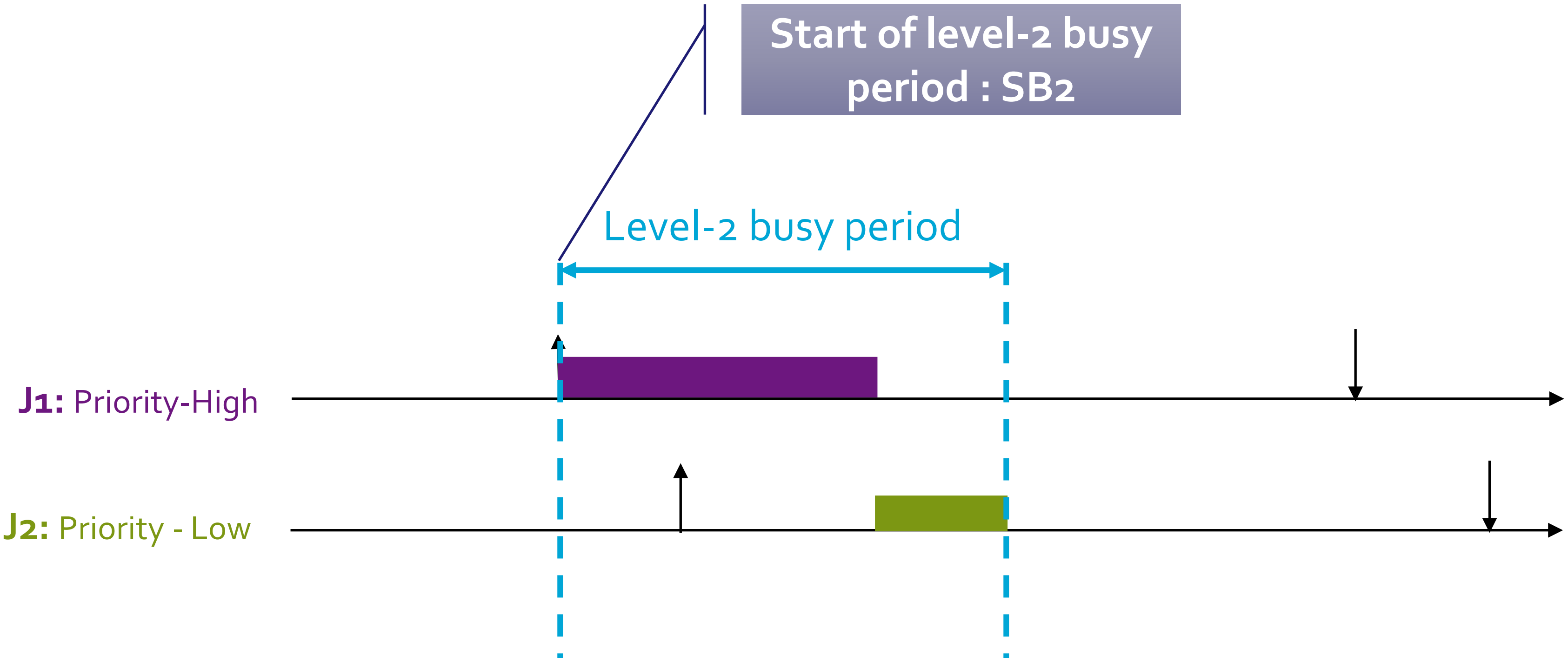
Interference(J3) = LFT(J2) – Latest release(J3)

Analytical view of our solution



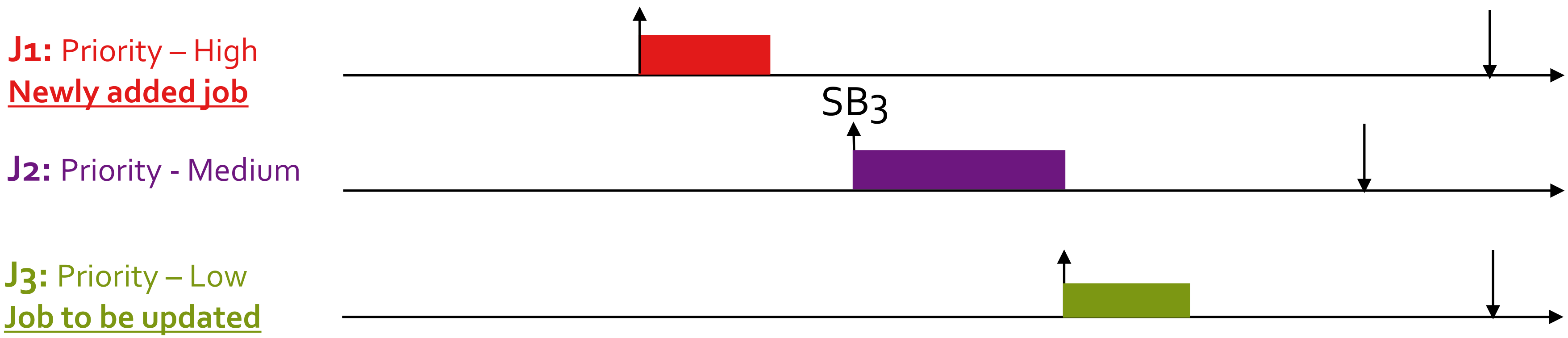
Busy period

Level-i busy period → Interval of time during which only jobs with priority higher or equal job J_i execute.



Updating a job

Scenario 1:
When SB3 > finish time of newly added job

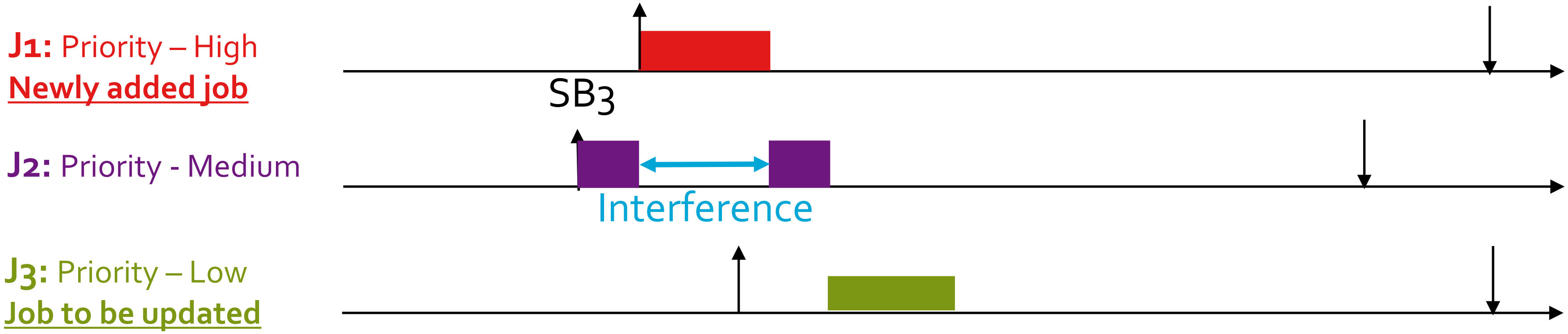


Interference(J3,J1) = 0

Updating a job

Scenario 2:

When SB3 < release time of newly added job



Interference(J3,J1) = Maximum execution time(J1)

Updating a job

Scenario 3:
In all other cases

J1: Priority – High
Newly added job

J2: Priority - Medium

J3: Priority – Low
Job to be updated

Interference(J3,J1) = LFT(J1) – SB3

Experiments - Uniprocessor

Tasksets

- Synthetic tasksets were generated using the Emberson and Davis Tool [1]
- Tasksets properties:
  - Ranged from 3 tasks to 18 tasks per taskset
  - Periods were either uniform and loguniform

State of the art

- **Response time analysis** for **uniprocessor** systems and **preemptive** tasks.

How does our solution perform in terms of accuracy for a given taskset in comparison to the state of the art?

[1] Techniques For The Synthesis Of Multiprocessor Tasksets, Emberson et al. 2010

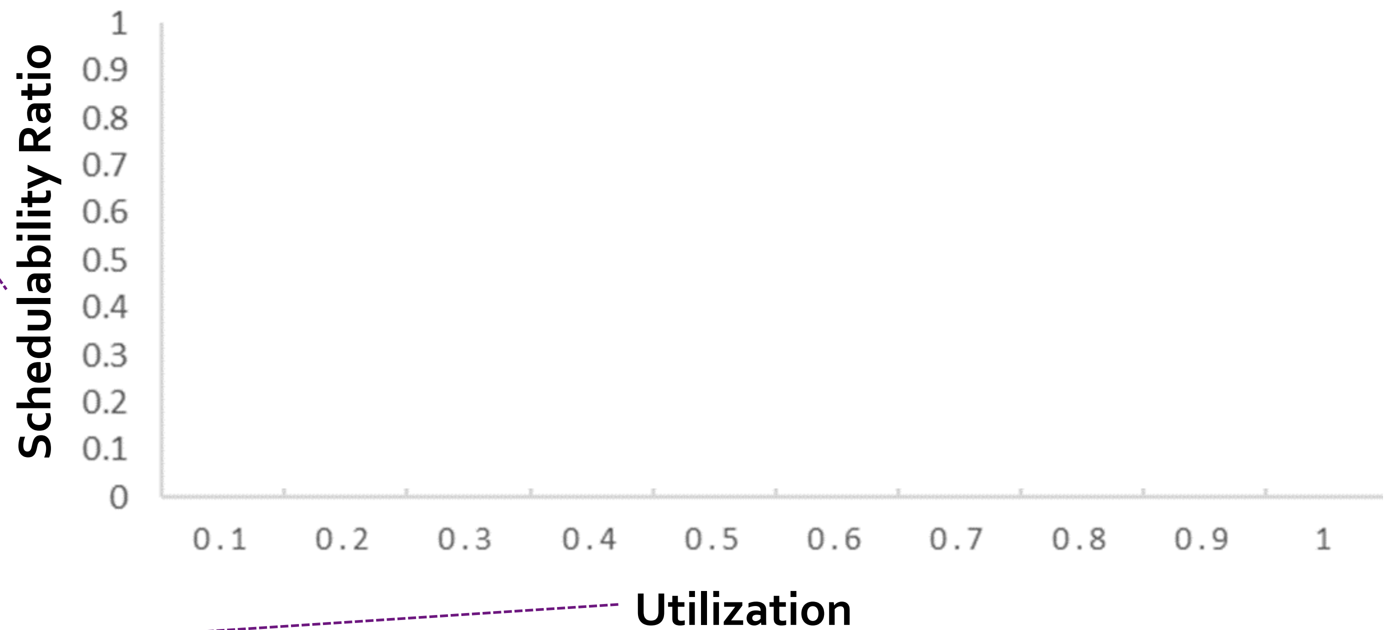
State of the art – Uniprocessor

Model	Methods	Type
~~Jitter~~ ~~Offset~~	• Response time analysis - Audsley RTA[1] • Park test[2] • Distance constrained taskset algorithm (DCT)[3]	Exact Sufficient Sufficient
~~Jitter~~ Offset	• Scheduling of hard real-time periodic systems with various kinds of deadline and offset constraints - Goossens[4]	Exact
Jitter Offset	• Calculating exact worst-case response times for static priority scheduled tasks with offsets and jitter – Redell et al.[5]	Exact

[1] Audsley et al., 1995
[2] Park et al., 2014
[3] Díaz-Ramíres et al., 2013
[4] Goossens, 1999
[5] Redell et al., 2002

Evaluation

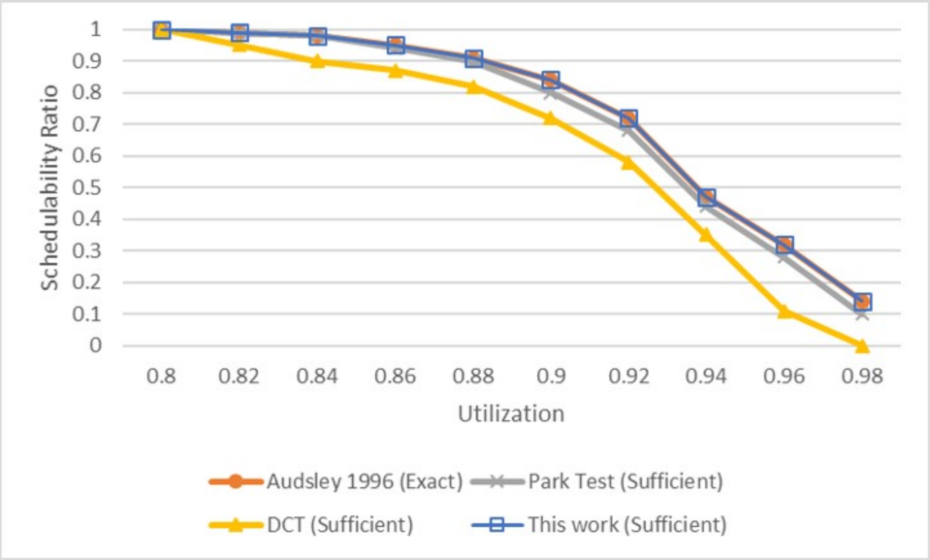
Number of tasksets that are schedulable
Total number of tasksets



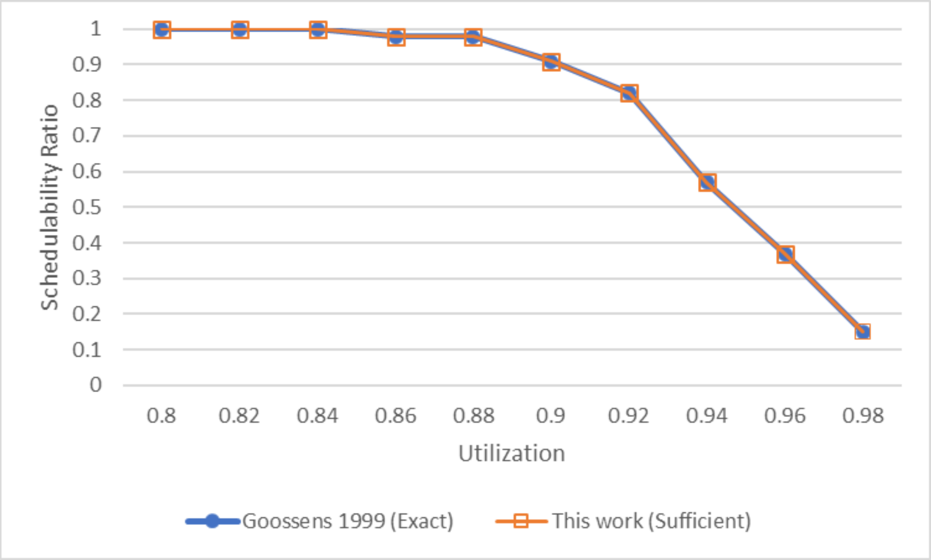
Total workload per unit time

The higher the schedulability ratio, the better

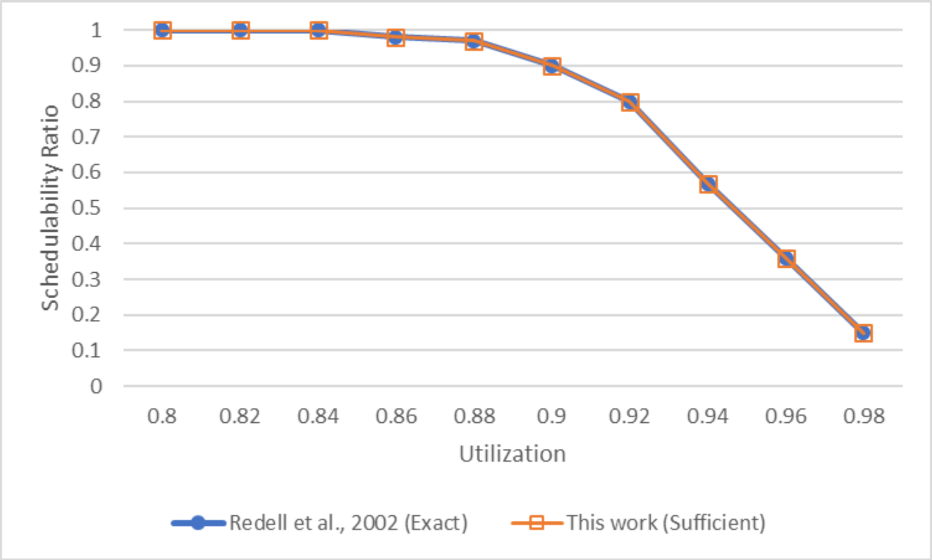
NO Jitter, NO Offset



NO Jitter but with offset



With both jitter and offset



Our solution overlaps with the exact solution in each case !

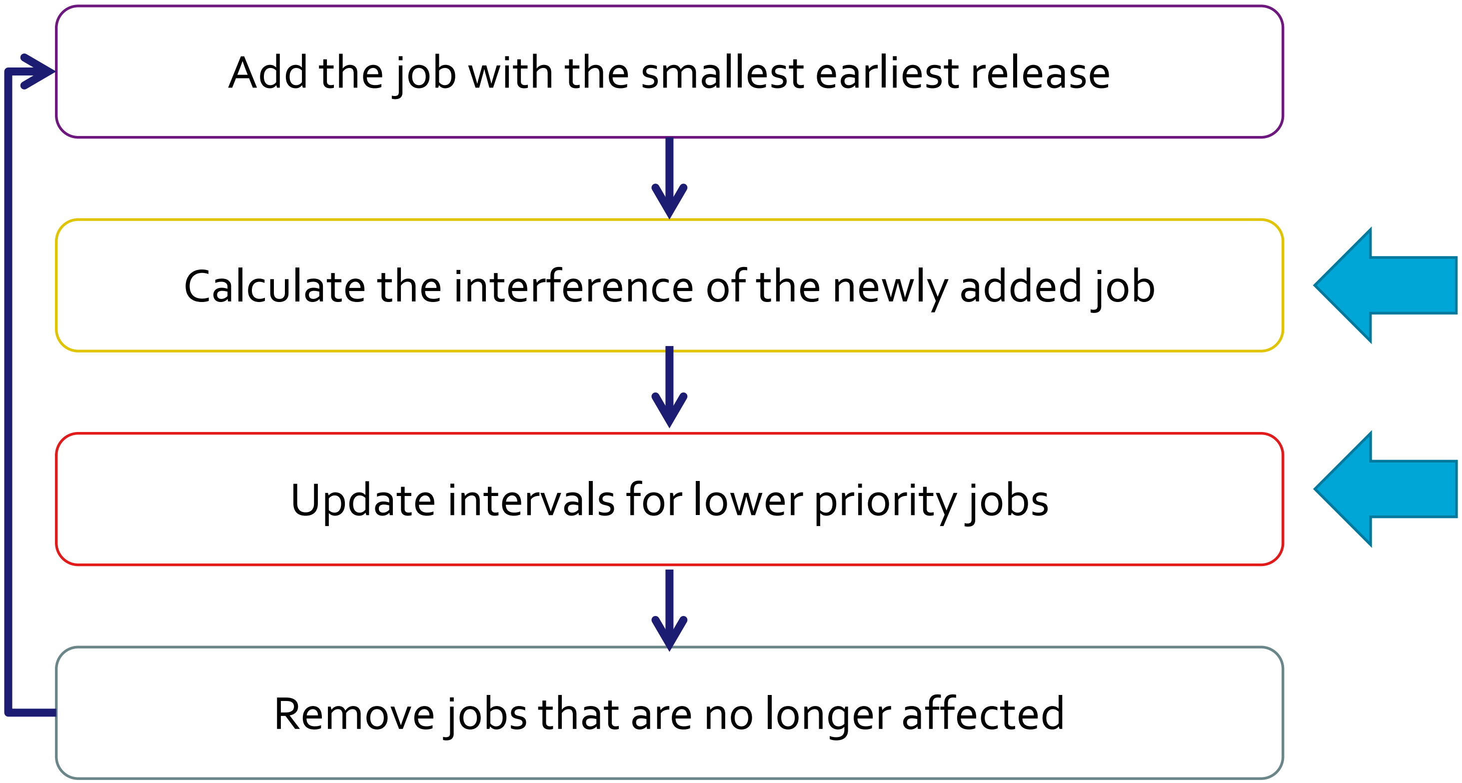
Taskset Size: 9 Tasks   Period Properties: Loguniform

Take away from the uniprocessor solution

Our solution performs with **high accuracy** for preemptive tasks on a **uniprocessor** system while **avoiding state space explosion**

Provides hope to build an accurate analysis for preemptive tasks on a multiprocessor system

Extension to multiprocessor platform



Extension to multiprocessor platform

While adding a job – Interference is calculated in the same way as for uniprocessor

While updating a job – Interference is further simplified than in the uniprocessor platform as the busy period concept was not extended

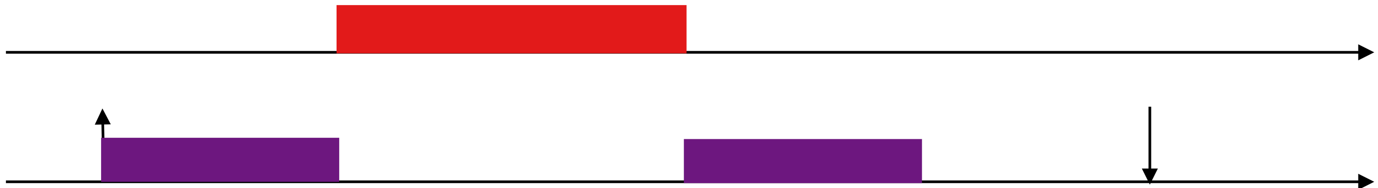
For a multiprocessor platform with 'm' cores:

In both cases,

$$\text{Interference on multiprocessor platform} = \frac{\text{Interference on a single processor}}{\text{Number of cores(m)}}$$

Amount of interference faced : 9 units

Job that faces 9 units of interference



Scenario 1: When the job is scheduled on a single core processor

Processor :

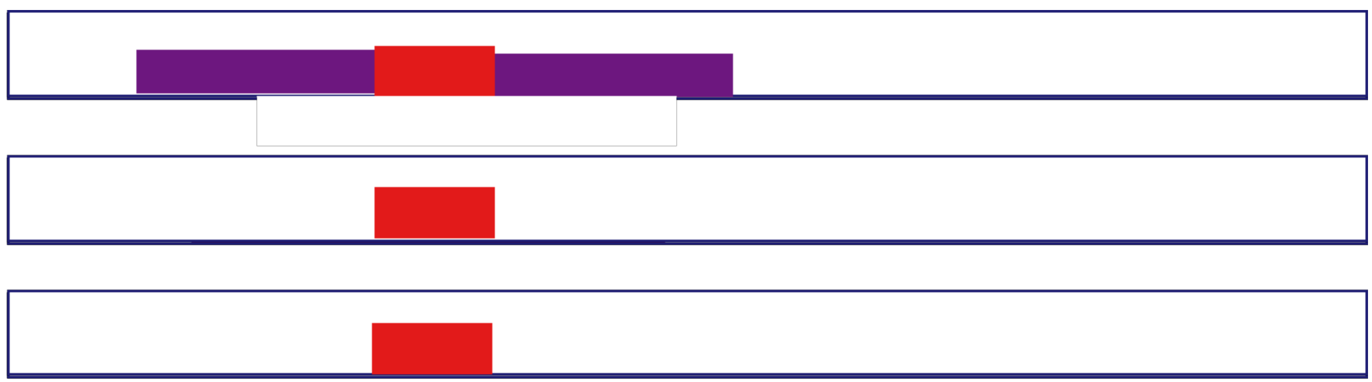
9 units

Scenario 2: When the job is scheduled on a multi core processor

Processor 1 :

Processor 2 :

Processor 3 :



3 units

Experiments - Multiprocessor

Tasksets

- Synthetic tasksets were generated using the Emberson and Davis Tool [1]
- Tasksets properties:
  - Number of cores (m): 2 to 8
  - Number of tasks ranged from 1*(m) to 5*(m)
  - Periods were loguniform

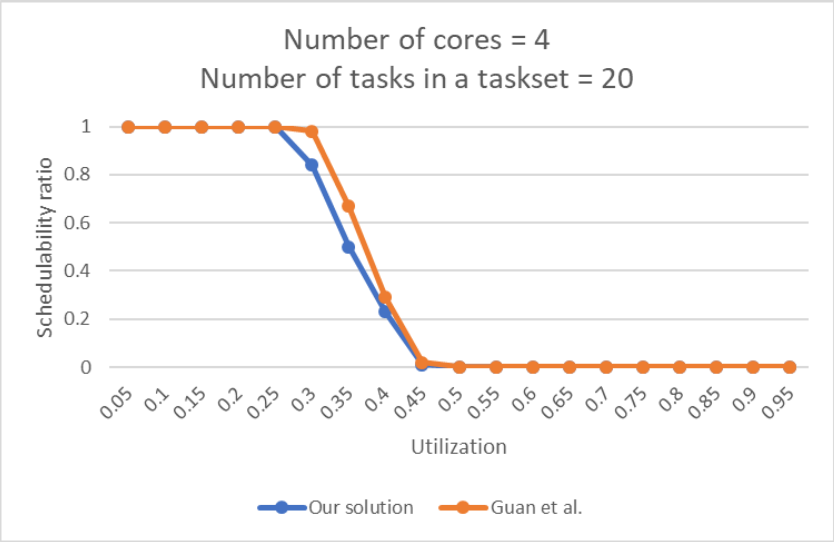
State of the art

- **Response time analysis (RTA)** for **multiprocessor** systems and **preemptive** tasks. The paper by Guan et.al [2] was chosen

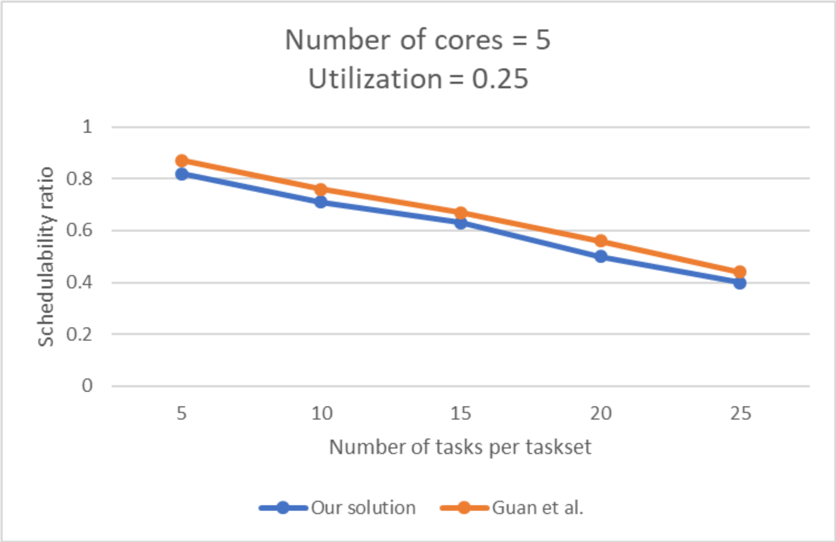
How does our solution perform in terms of accuracy for a given taskset in comparison to the state of the art?

[1] Techniques For The Synthesis Of Multiprocessor Tasksets, Emberson et al. 2010
[2] N. Guan, M. Stigge, W. Yi, and G. Yu. New response time bounds for fixed priority multiprocessor scheduling. In 2009 30th IEEE Real-Time Systems Symposium, pages 387–397, 2009

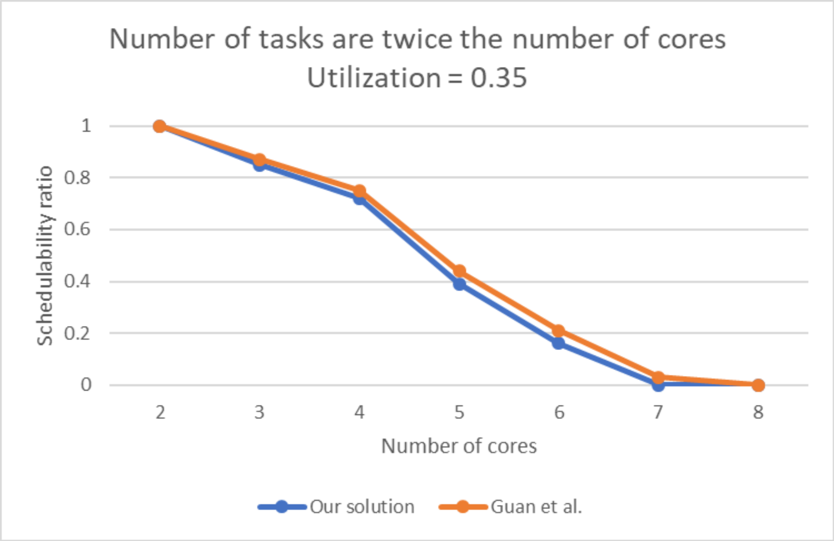
Varying utilization



Varying number of tasks



Varying number of cores



Our solution performs slightly worse than the state of the art

Conclusion

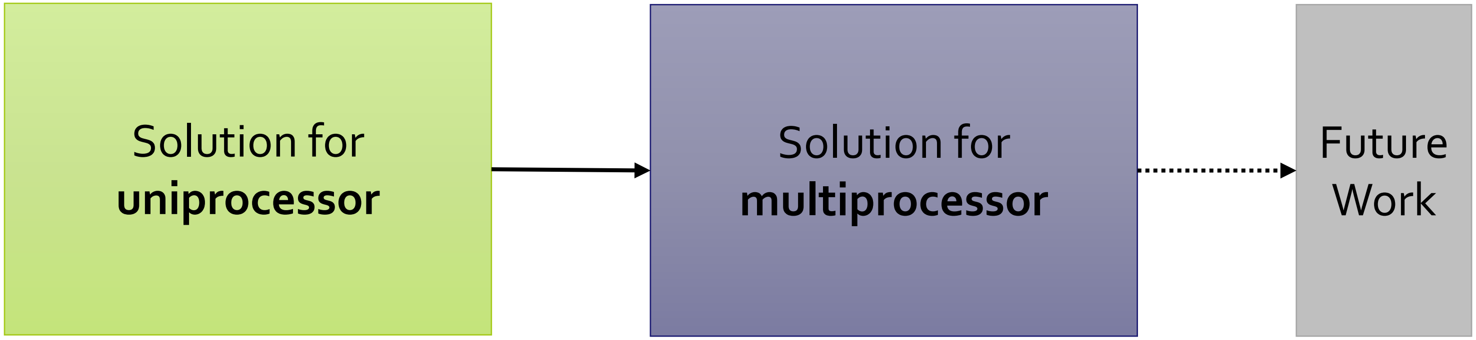
Our solution performs with **high accuracy** on a **uniprocessor system** while **avoiding state space explosion**

Pessimism is introduced when extending to the multiprocessor case

This method has never been applied to this problem before, and hence more research will help bring down the pessimism

Summary

Thesis timeline:



Thank You!