Scheduling and Analysis of Limited-Preemptive Malleable Gang Taks

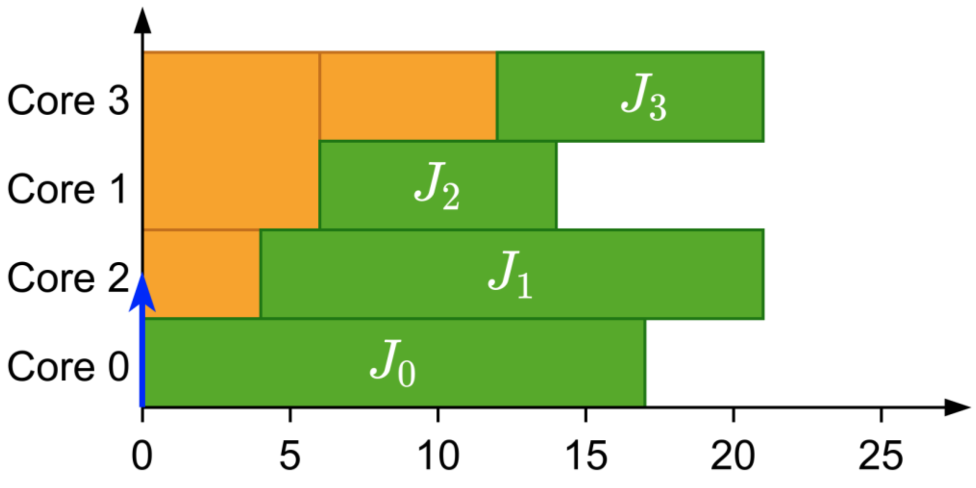
Joan Marcè i Igual Geoffrey Nelissen Mitra Nasri Paris Panagiotou

24th of February, 2020

TUDelft

What is gang?

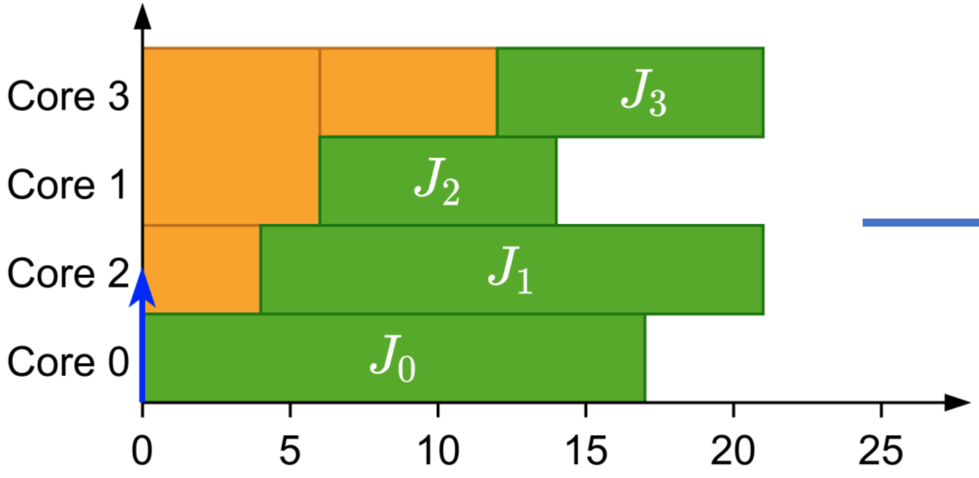
What is gang?



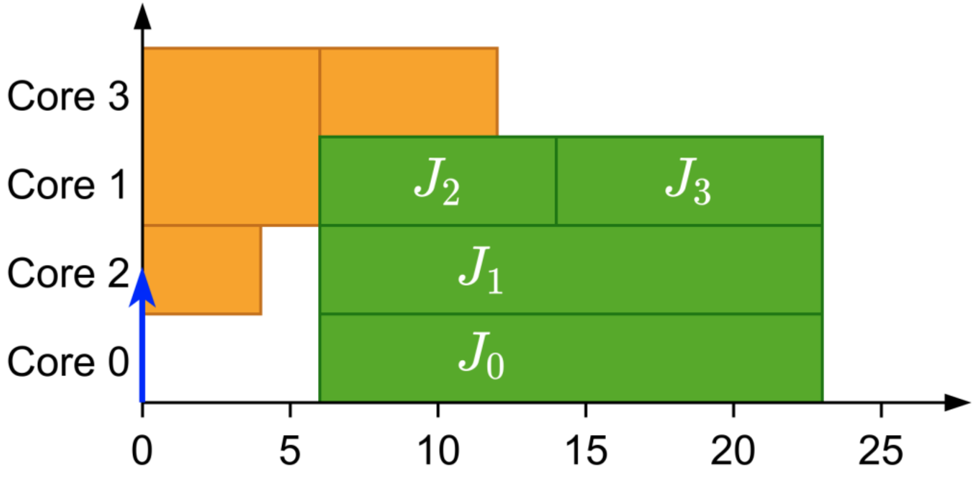
Global scheduling

What is gang?

- Parallel threads executed together as a “gang”



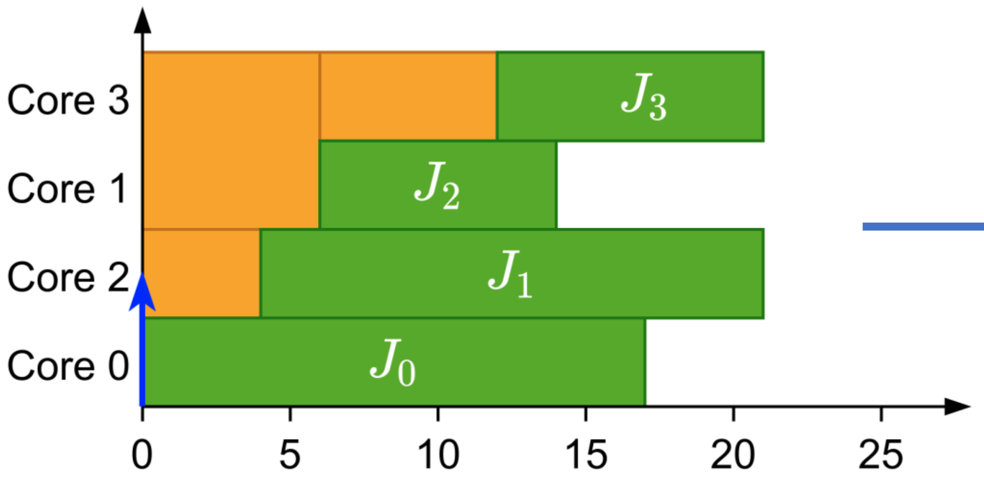
Global scheduling



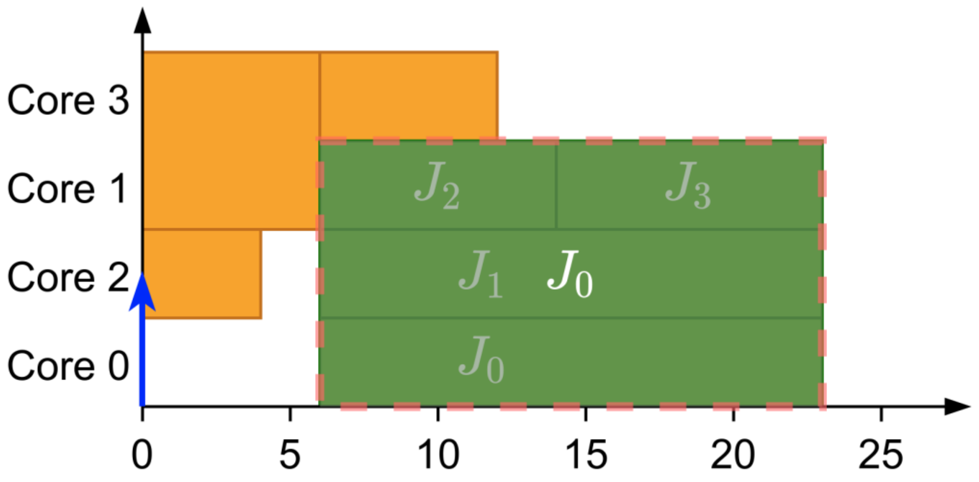
Gang Scheduling

What is gang?

- Parallel threads executed together as a “gang”
- Execution does not start until there are enough free cores



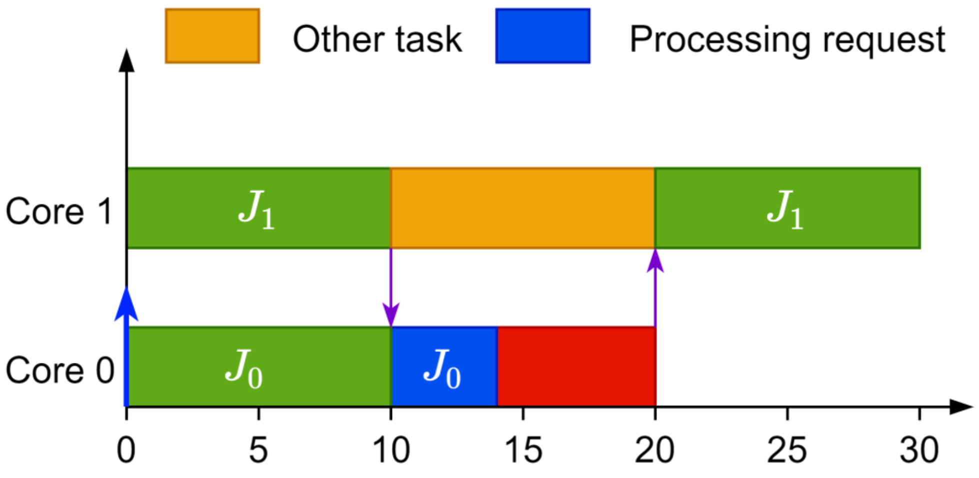
Global scheduling



Gang Scheduling

Why gang?

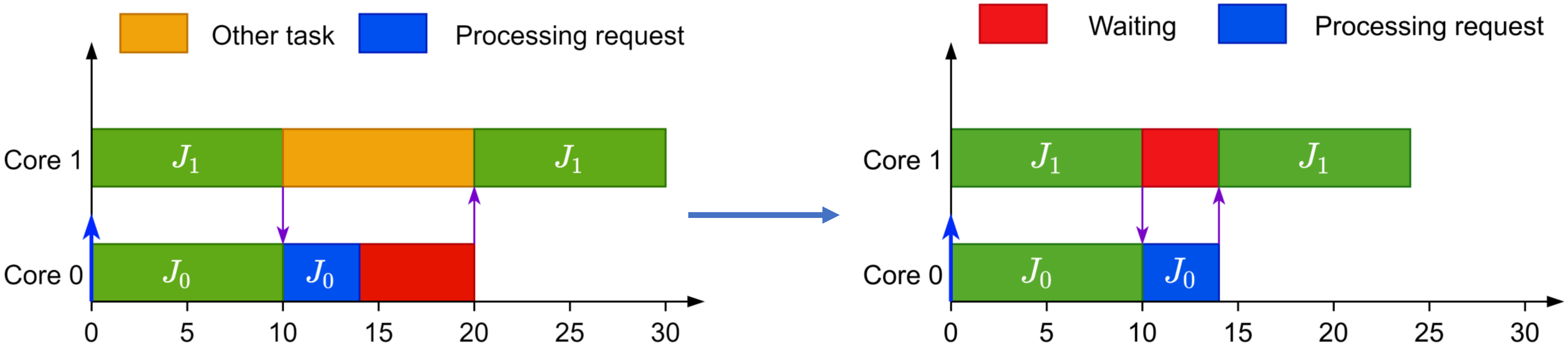
Why gang?



Global scheduling

Why gang?

- Efficient synchronization



Global scheduling

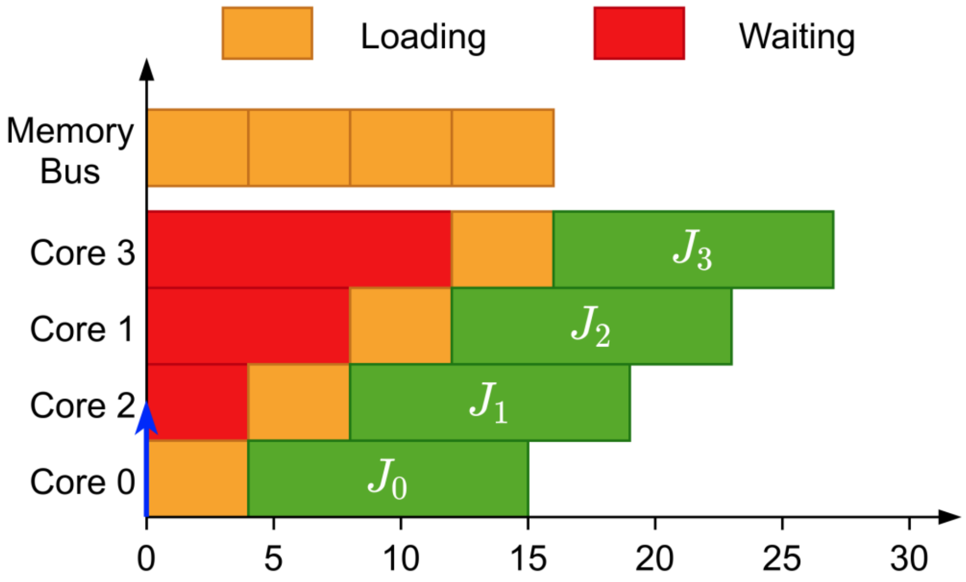
Gang Scheduling

Why gang?

- Efficient synchronization

Why gang?

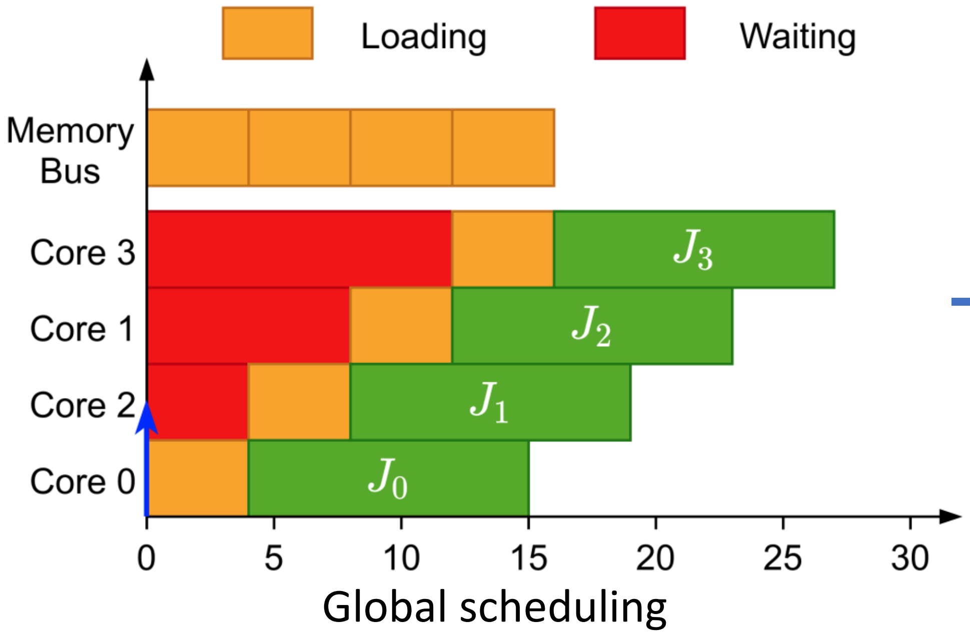
- Efficient synchronization

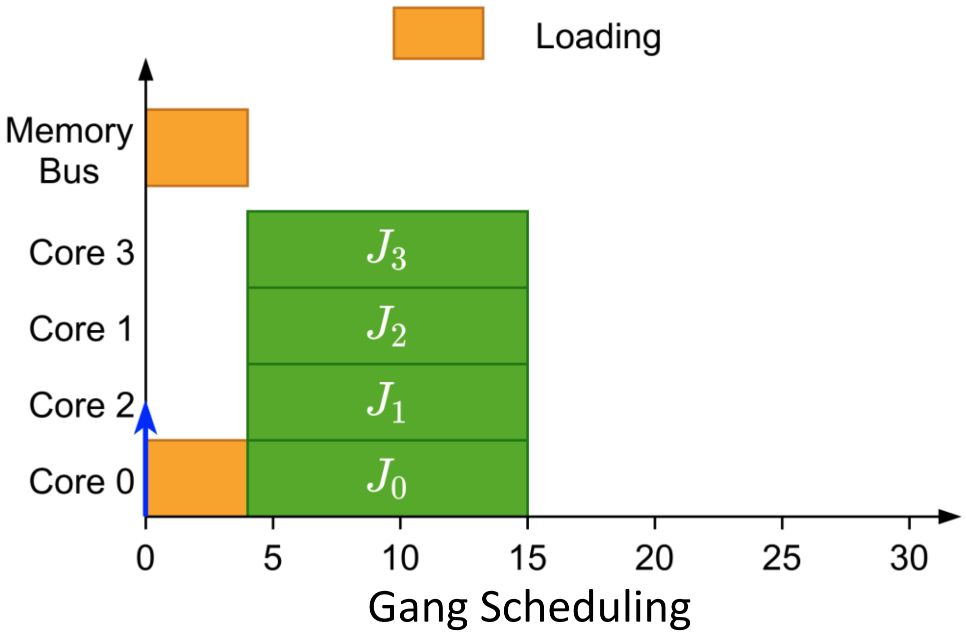


Global scheduling

Why gang?

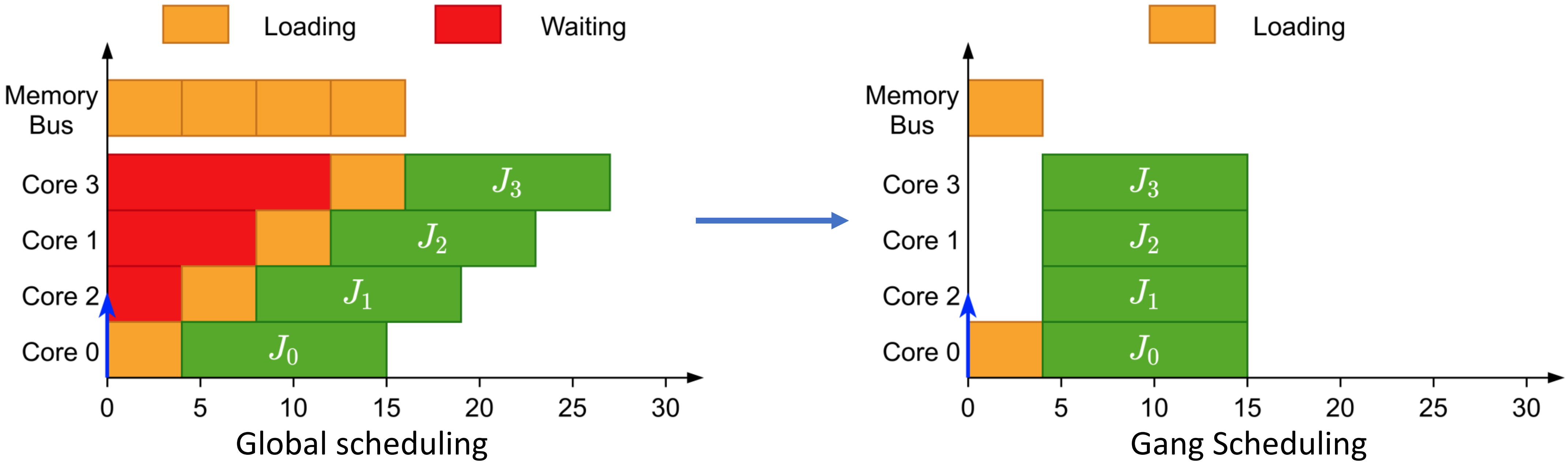
- Efficient synchronization





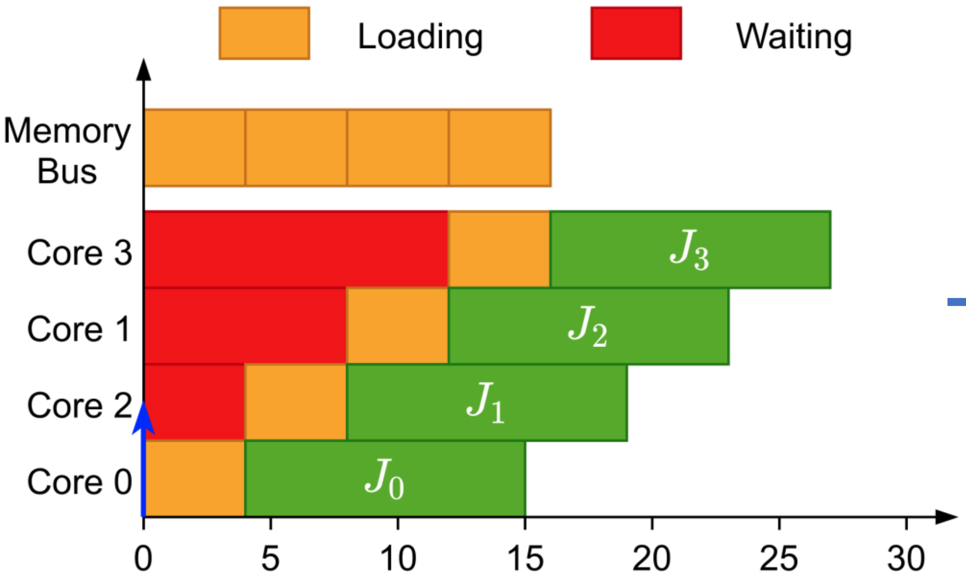
Why gang?

- Efficient synchronization
- Avoids overhead when loading initial data

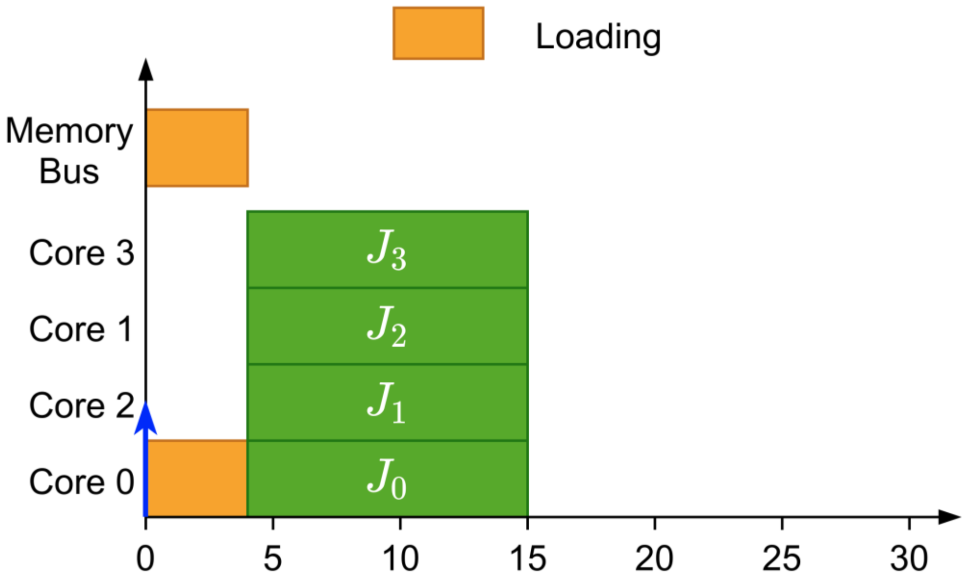


Why gang?

- Efficient synchronization
- Avoids overhead when loading initial data
- Shows its full potential when executed non-preemptively



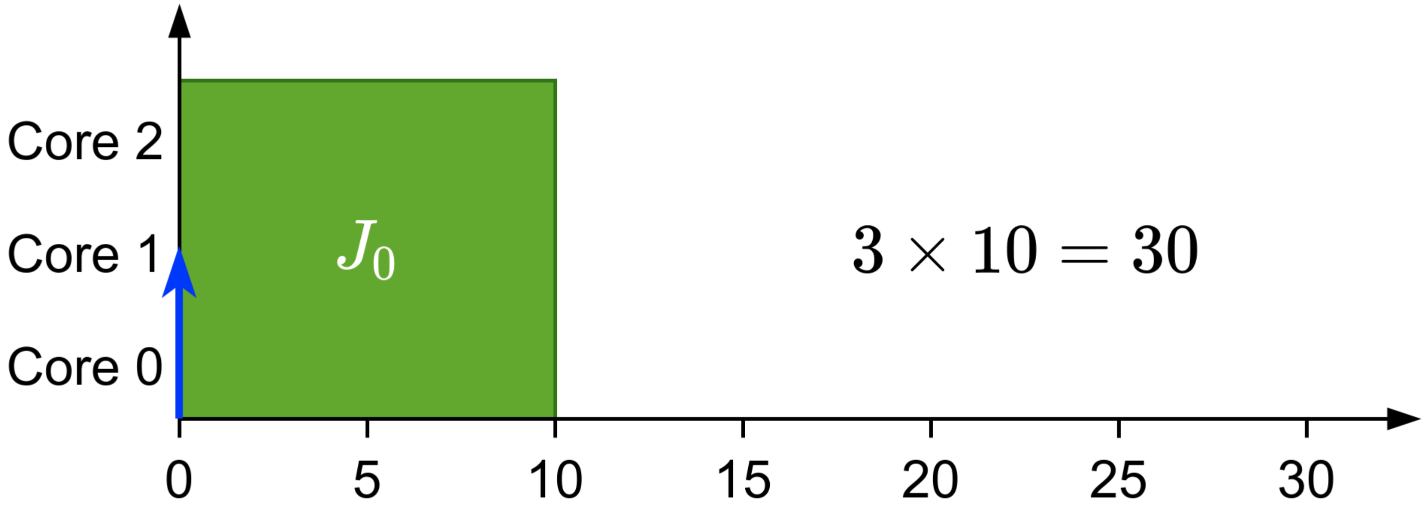
Global scheduling



Gang Scheduling

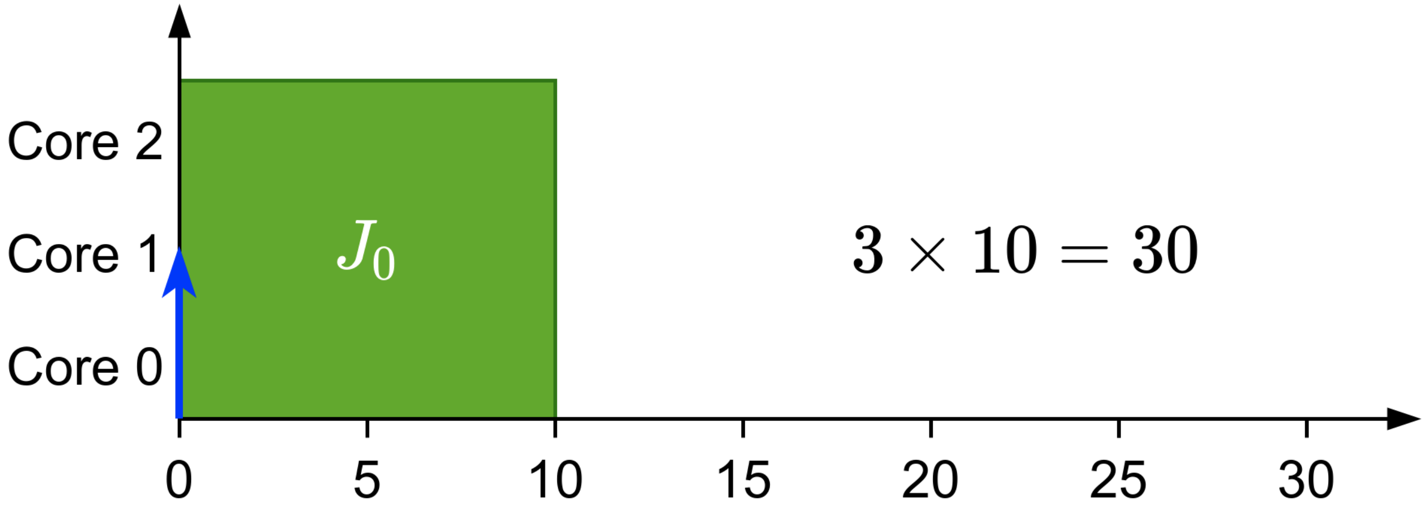
Types of gang

Types of gang



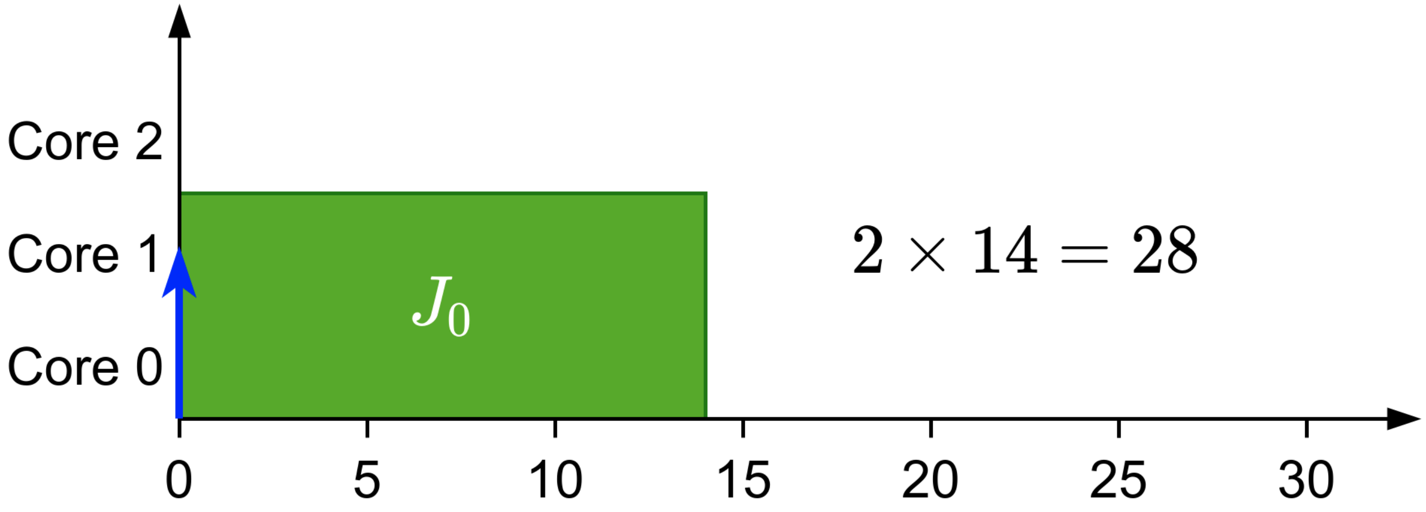
Types of gang

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



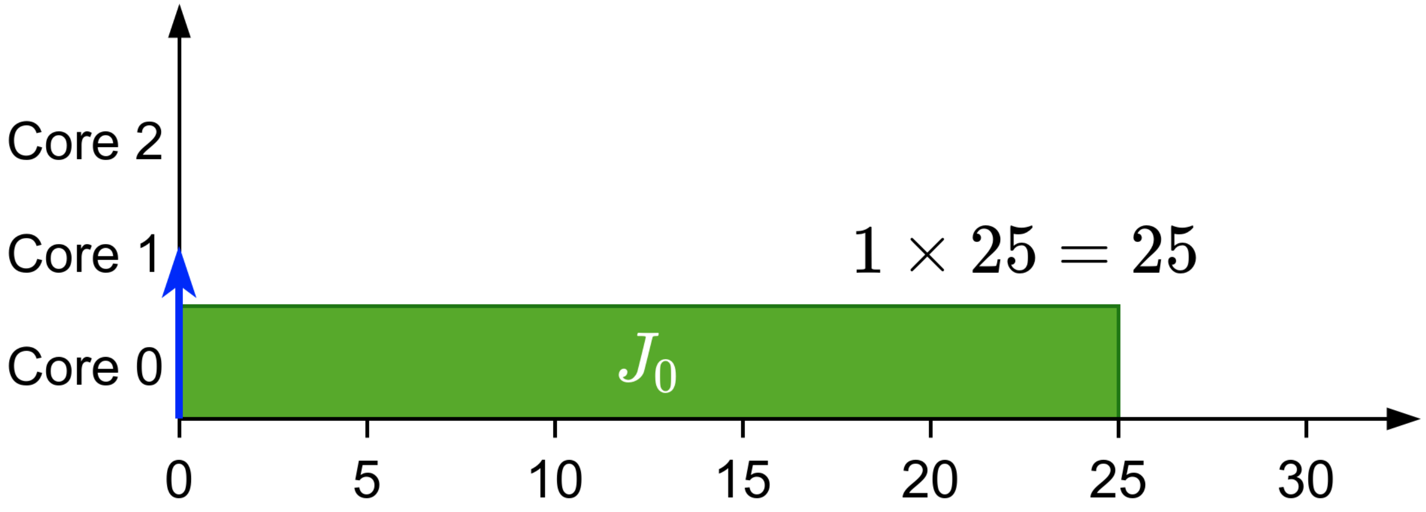
Types of gang

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



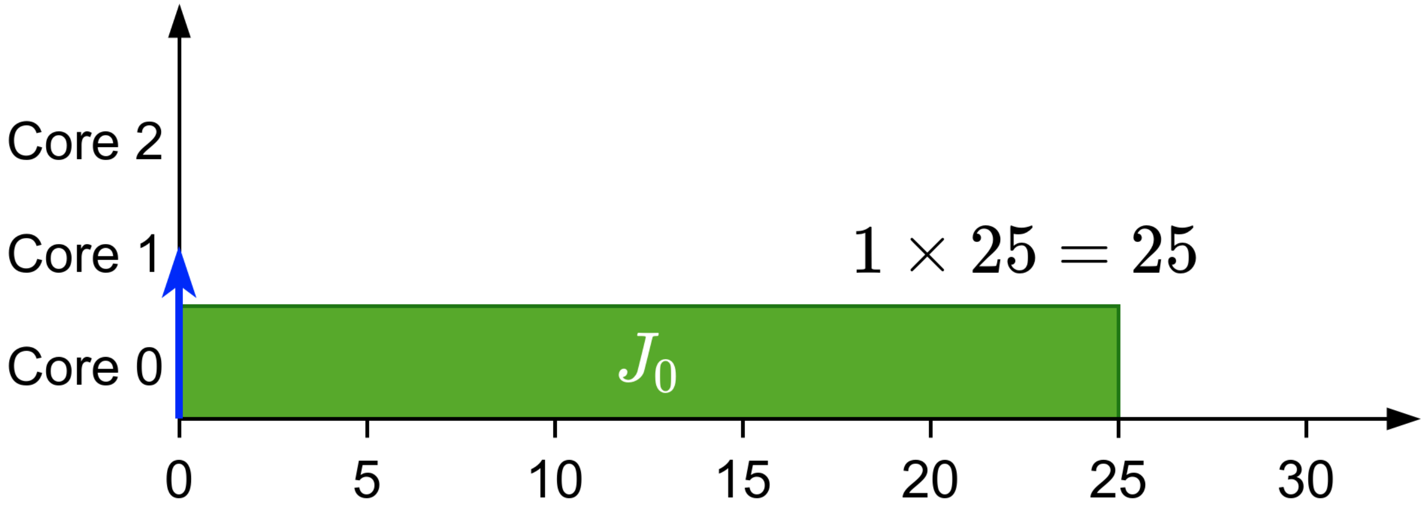
Types of gang

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



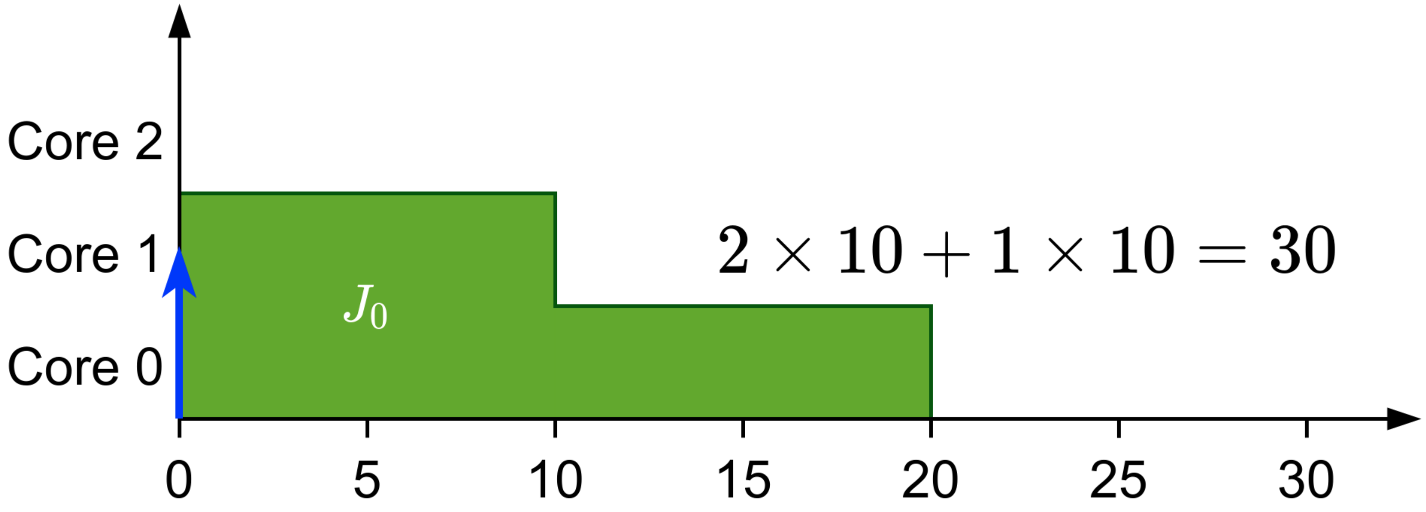
Types of gang

- **Rigid**: number of cores set by programmer
- **Moldable**: number of cores assigned during scheduling



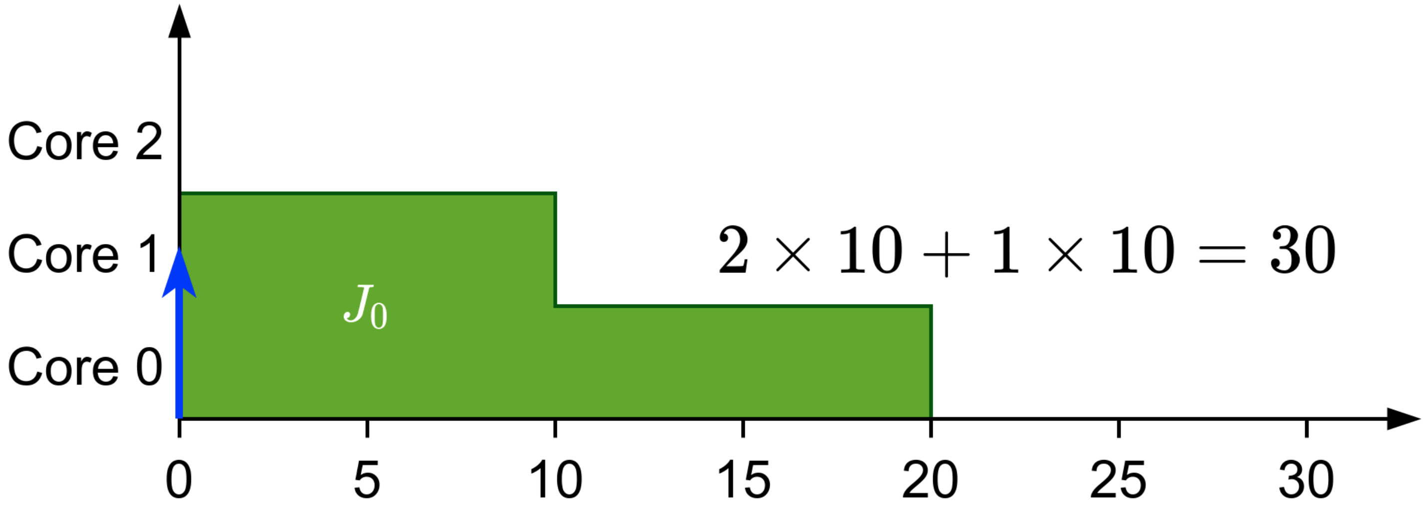
Types of gang

- **Rigid**: number of cores set by programmer
- **Moldable**: number of cores assigned during scheduling
- **Malleable**: Number of cores can change during runtime



Types of gang

- **Rigid**: number of cores set by programmer
- **Moldable**: number of cores assigned during scheduling
- **Malleable**: Number of cores can change during runtime



Previous work

Previous work

- Introduced in the context of high-performance computing[1]

[1]Ousterhout, 1982

Previous work

- Introduced in the context of high-performance computing[1]
- In real-time:

[1]Ousterhout, 1982

Previous work

- Introduced in the context of high-performance computing[1]
- In real-time:
  - We know that JLFP scheduler is not predictable/sustainable[2]

[1]Ousterhout, 1982
[2]Goossens et al., 2010

Previous work

- Introduced in the context of high-performance computing[1]
- In real-time:
  - We know that JLFP scheduler is not predictable/sustainable[2]
  - Most of the work is focused in fully-preemptive solutions:
    - Optimal scheduler for rigid gang (DP-Fair)[3]
    - Moldable scheduler[4]

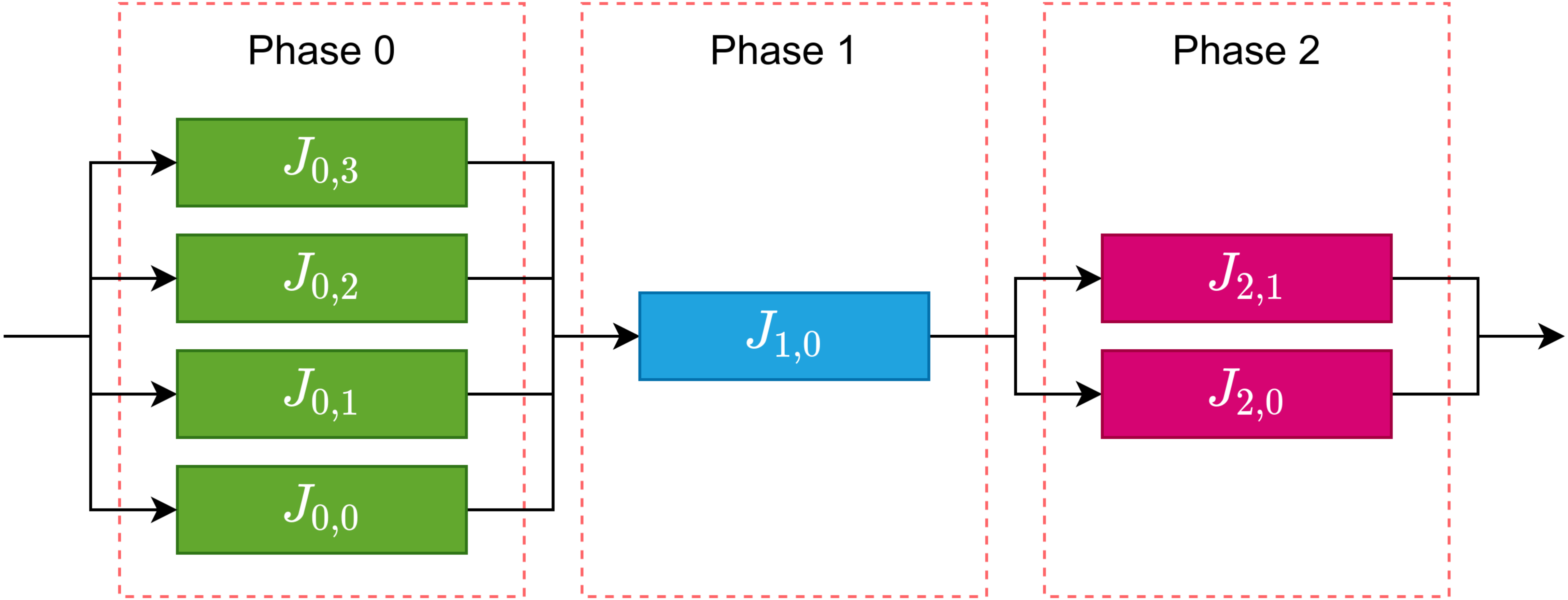
[1]Ousterhout, 1982
[2]Goossens et al., 2010
[3]Goossens et al., 2016
[4]Berten et al., 2011

Previous work

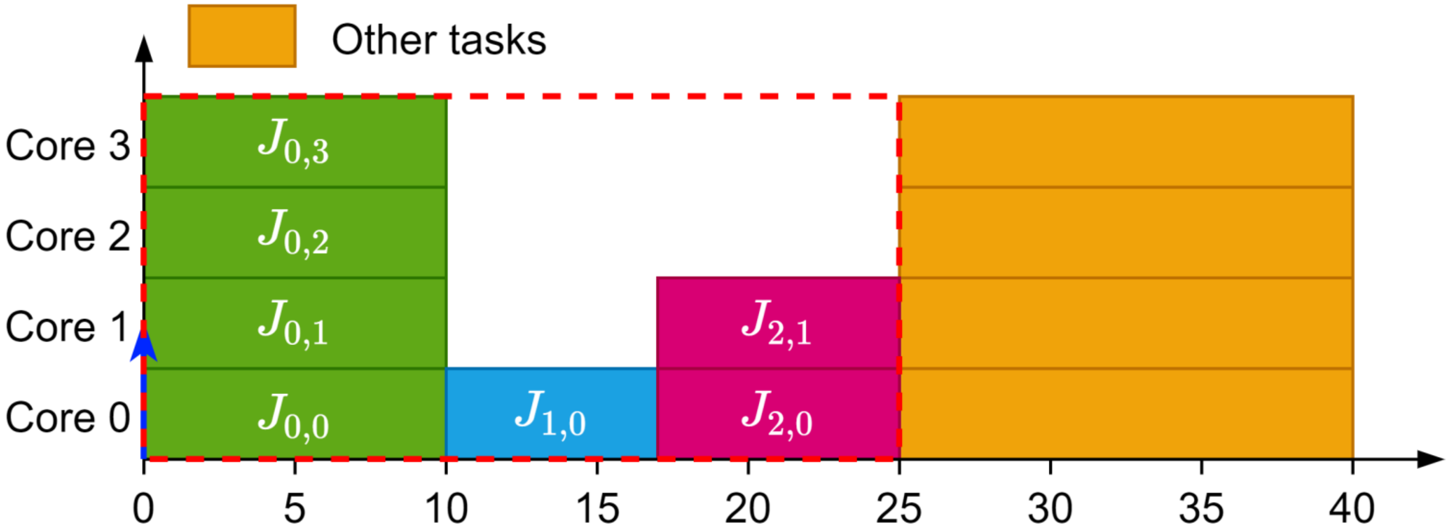
- Introduced in the context of high-performance computing[1]
- In real-time:
  - We know that JLFP scheduler is not predictable/sustainable[2]
  - Most of the work is focused in fully-preemptive solutions:
    - Optimal scheduler for rigid gang (DP-Fair)[3]
    - Moldable scheduler[4]
  - Bundled scheduling[5]
    - Tasks with precedence constraints modelled as a succession of bundles
    - Our limited-preemptive definition comes from here

[1]Ousterhout, 1982
[2]Goossens et al., 2010
[3]Goossens et al., 2016
[4]Berten et al., 2011
[4]Wasly et al., 2017

Summarizing



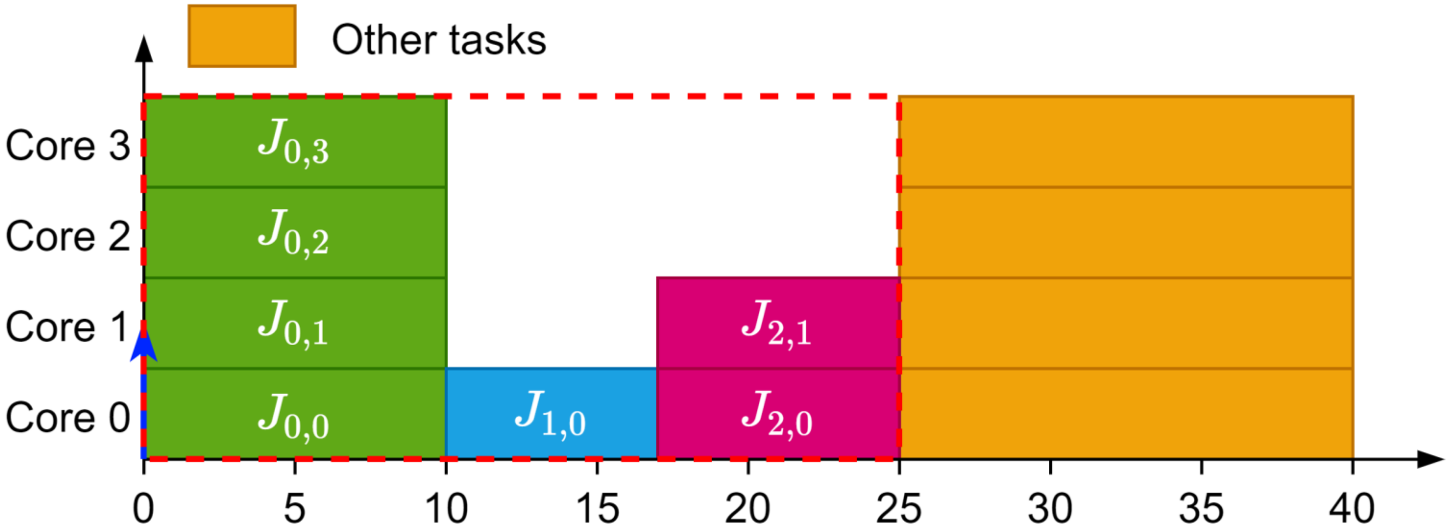
Summarizing



Rigid gang scheduling

Summarizing

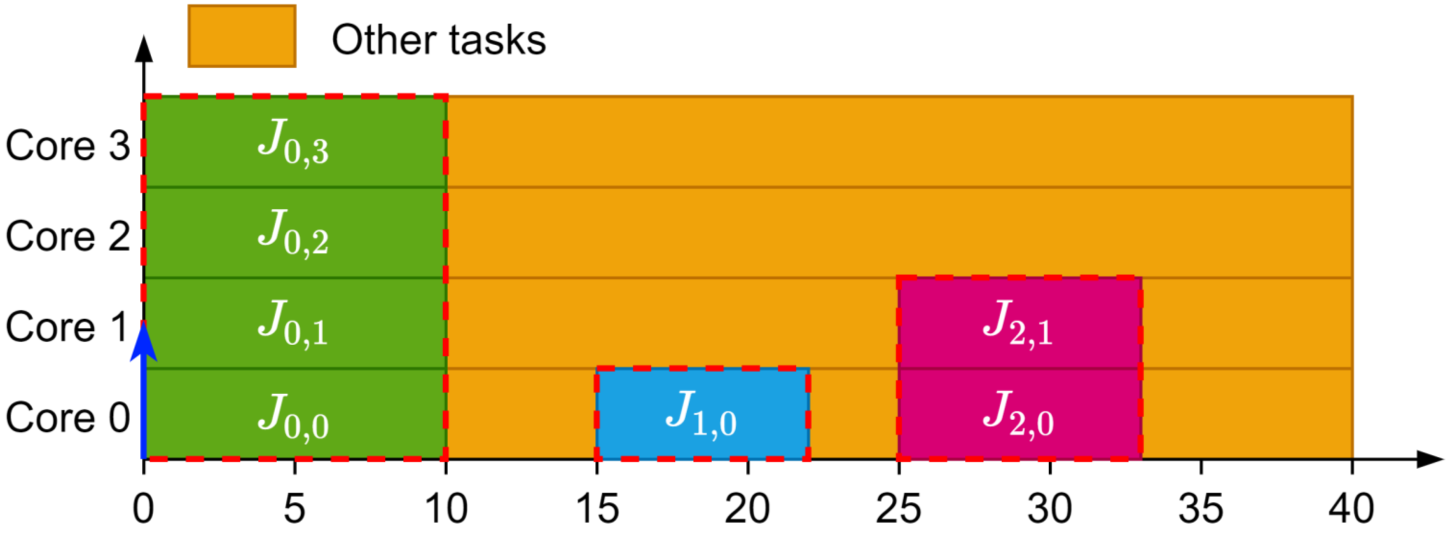
- Rigid gang reserves the whole block



Rigid gang scheduling

Summarizing

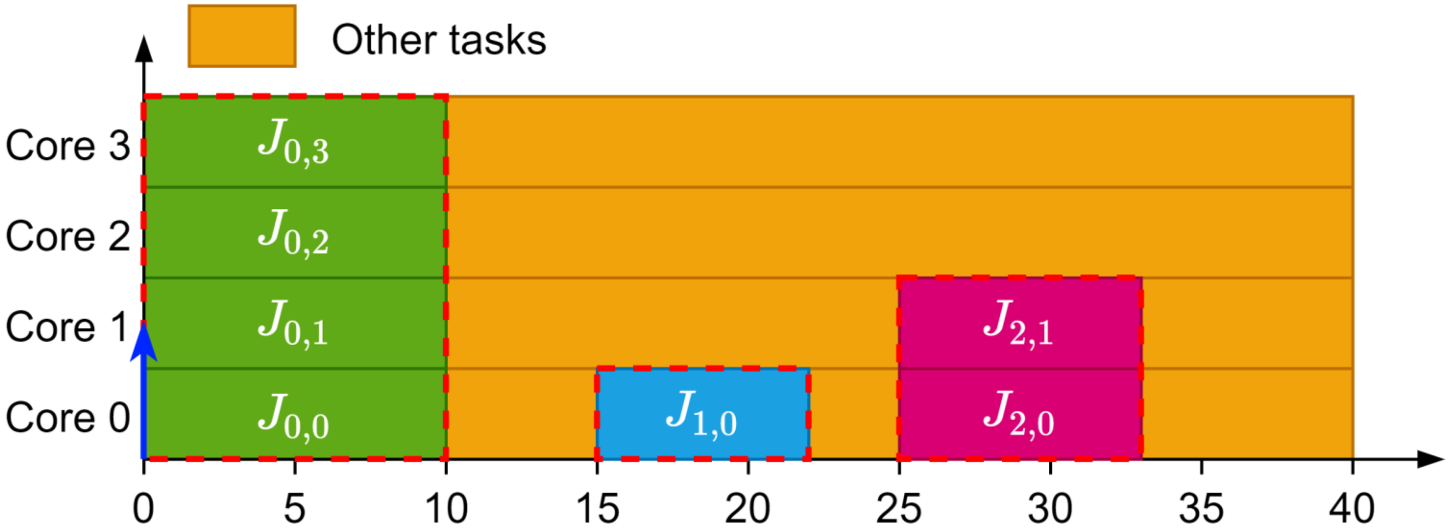
- Rigid gang reserves the whole block



Bundled model scheduling

Summarizing

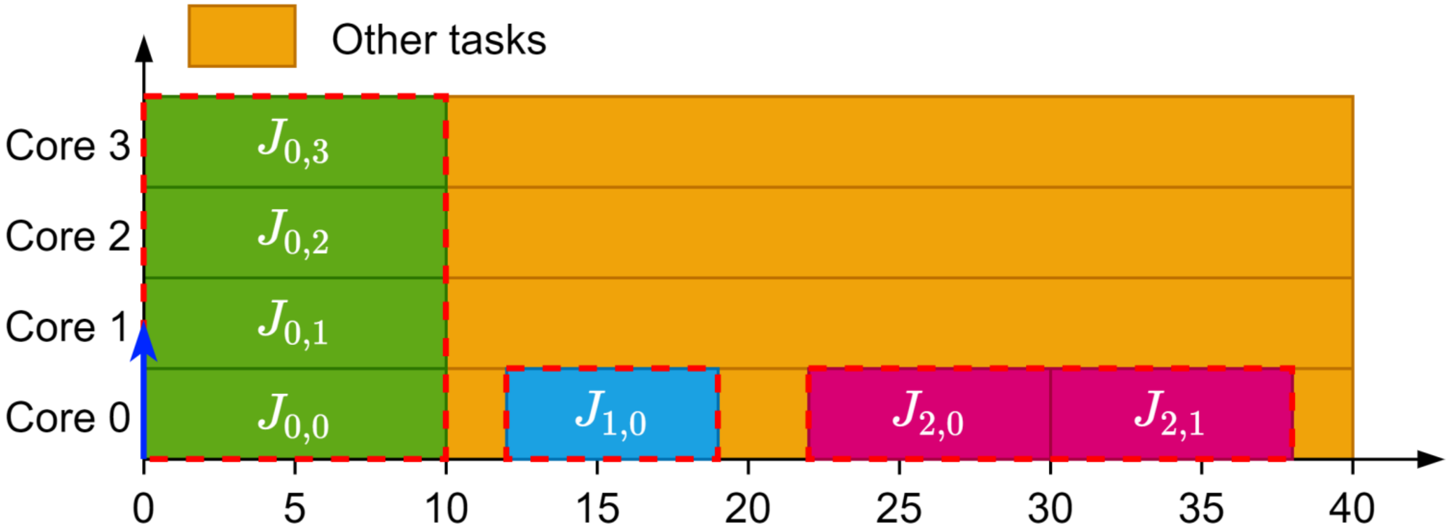
- Rigid gang reserves the whole block
- Bundled creates multiple rigid blocks with depencencies



Bundled model scheduling

Summarizing

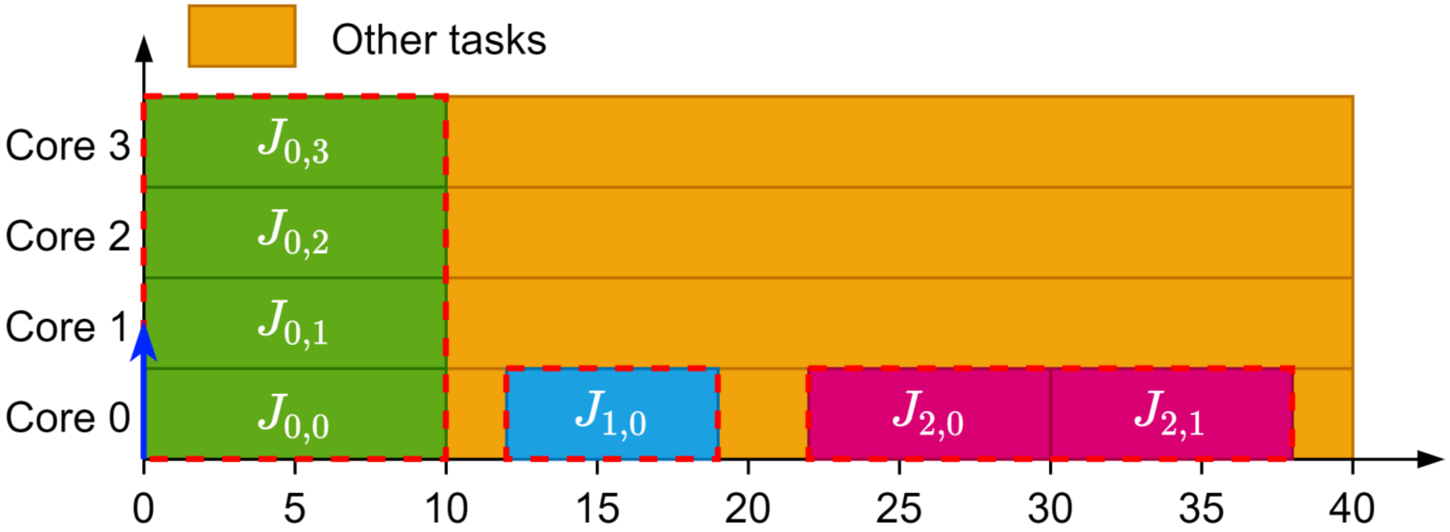
- Rigid gang reserves the whole block
- Bundled creates multiple rigid blocks with depencencies



Limited-Preemptive scheduling

Summarizing

- Rigid gang reserves the whole block
- Bundled creates multiple rigid blocks with depencencies
- Limited-Preemptive tries to schedule these blocks in a moldable way



Limited-Preemptive scheduling

Our work

Project goals

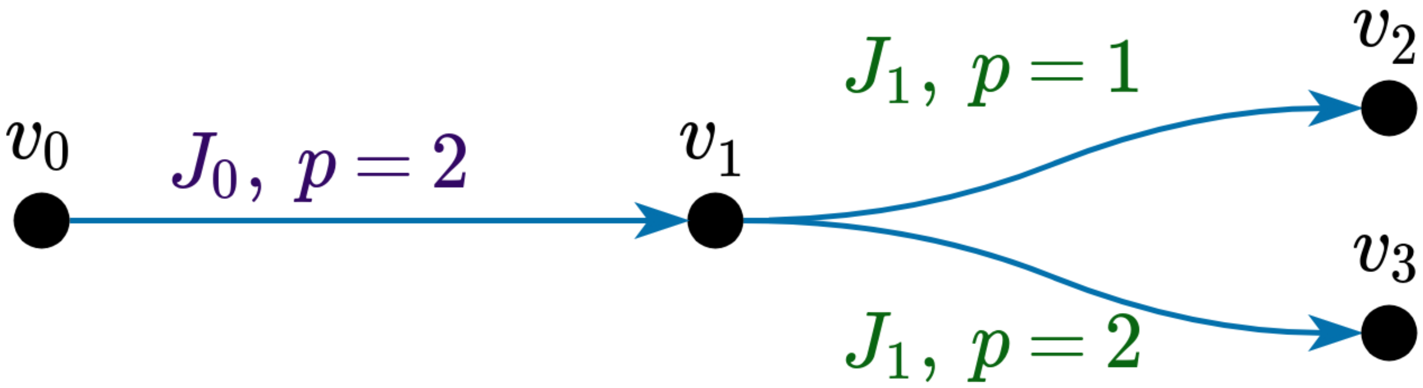
Project goals

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

Project goals

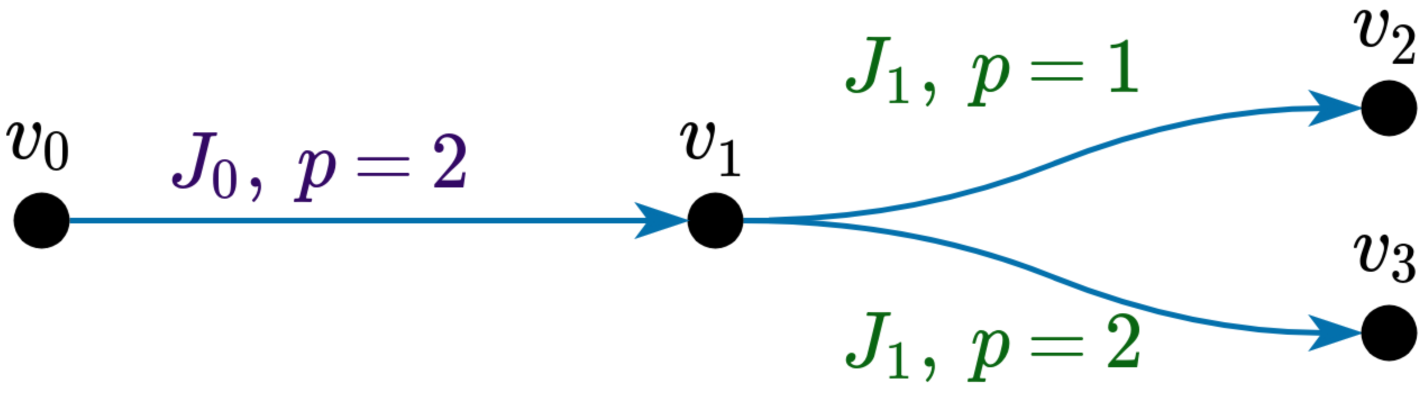
- Design an accurate schedulability analysis for limited-preemptive moldable gang tasks
- Propose a new scheduling algorithm to improve the schedulability of limited-preemptive moldable gang tasks

Schedule Abstraction Graph



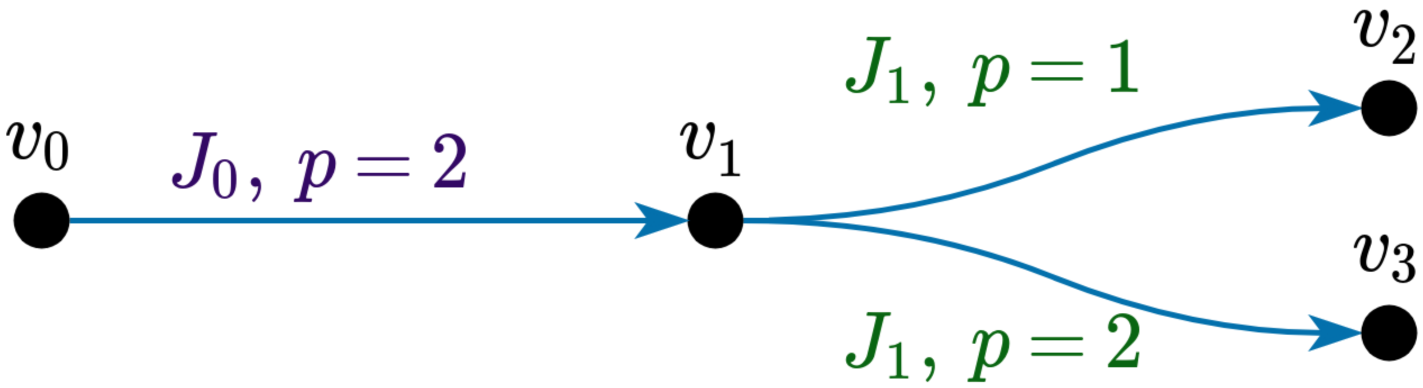
Schedule Abstraction Graph

- Accurate and relatively fast analysis
  - Faster than an exact analysis
  - Not as pessimistic as closed-form analyses



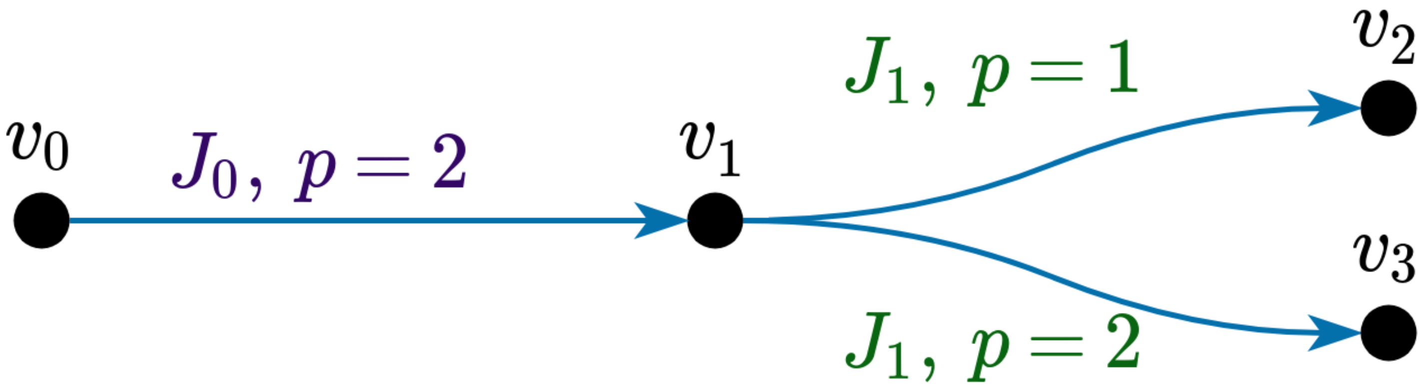
Schedule Abstraction Graph

- Accurate and relatively fast analysis
  - Faster than an exact analysis
  - Not as pessimistic as closed-form analyses
- Models scheduler decisions



Schedule Abstraction Graph

- Accurate and relatively fast analysis
  - Faster than an exact analysis
  - Not as pessimistic as closed-form analyses
- Models scheduler decisions
- Encodes core availability after every transition



Job-Level Fixed Priority Scheduler for Gang

- Based on Global JLFP scheduler
- Work conserving scheduler
- Job with highest priority goes first
- Assigns maximum cores available between $s_i^{\min}$ and $s_i^{\max}$

Difficulties related to SAG

- We have to consider all scenarios.
- The scheduler has to decide:
  - When to release a job
  - How many cores to assign to this job

Analysis

Analysis

- $A_c^{\min}$ time at which we have c cores possibly available
- $A_c^{\max}$ time at which we have c cores certainly available

Analysis

- $A_c^{\min}$ time at which we have c cores possibly available
- $A_c^{\max}$ time at which we have c cores certainly available

- EST_i Earliest Start Time
- LST_i Latest Start Time
- EFT_i Earliest Finishing Time
- LFT_i Latest Finishing Time

Analysis

- $A_c^{\min}$ time at which we have c cores possibly available
- $A_c^{\max}$ time at which we have c cores certainly available

- EST_i Earliest Start Time
- LST_i Latest Start Time
- EFT_i Earliest Finishing Time
- LFT_i Latest Finishing Time

$$EST_i \leq LST_i$$

Analysis

- $A_c^{\min}$ time at which we have c cores possibly available
- $A_c^{\max}$ time at which we have c cores certainly available

- EST_i^p Earliest Start Time
- LST_i^p Latest Start Time
- EFT_i^p Earliest Finishing Time
- LFT_i^p Latest Finishing Time

$$EST_i^p \leq LST_i^p$$

Analysis

$$EST_i^p = \max\{r_i^{\min}, A_p^{\min}\}$$

- Job cannot start before
  - Being released
  - Enough cores are available

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

- Job cannot start with p cores after:
  - $p + 1$ cores are available as JLFP would schedule it with $p + 1$ cores
  - A lower priority task is ready because JLFP is work-conserving
  - A higher priority task is ready

Analysis

- Obtain EFT_i^p and LFT_i^p from:

$$EFT_i^p = EST_i^p + c_i^{\min}(p)$$
$$LFT_i^p = LST_i^p + c_i^{\max}(p)$$

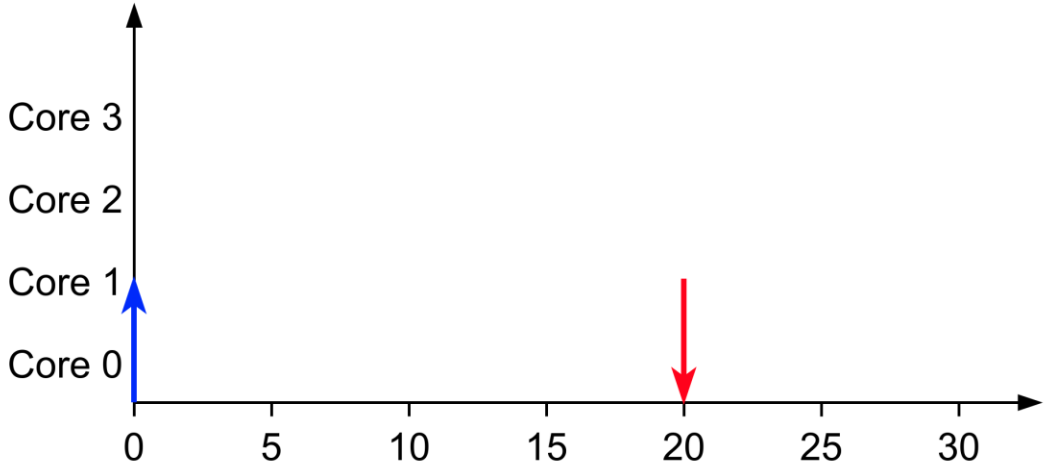
- And compute new $A_c^{\min}$ and $A_c^{\max}$

New scheduler

New Scheduler

New Scheduler

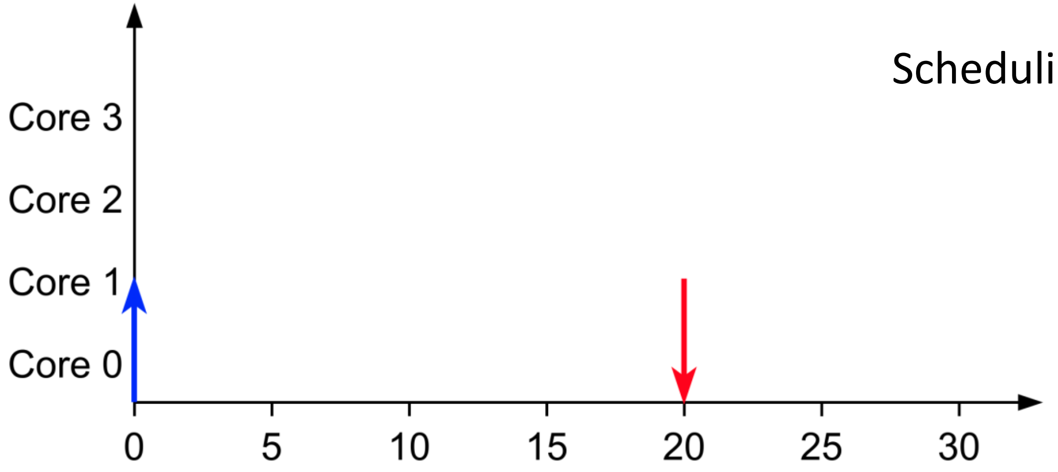
JLFP scheduler



	$s_j^{\min}$	$s_j^{\max}$	d_i	$c_i(v)$
J_0	2	2	100	10
J_1	3	3	20	5
J_2	1	1	100	20
J_3	1	3	100	24, 13, 20

New Scheduler

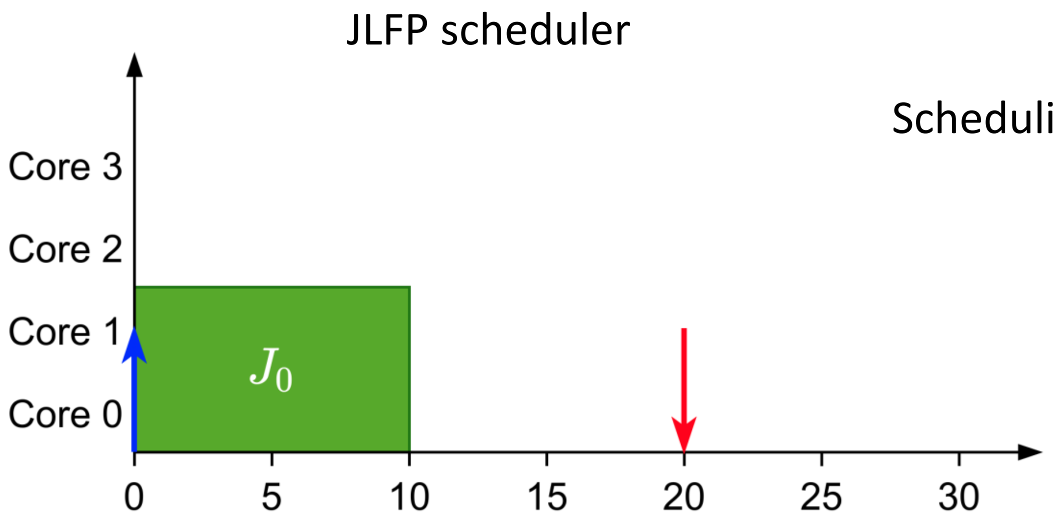
JLFP scheduler



Scheduling: J_0

	$s_j^{\min}$	$s_j^{\max}$	d_i	$c_i(v)$
J_0	2	2	100	10
J_1	3	3	20	5
J_2	1	1	100	20
J_3	1	3	100	24, 13, 20

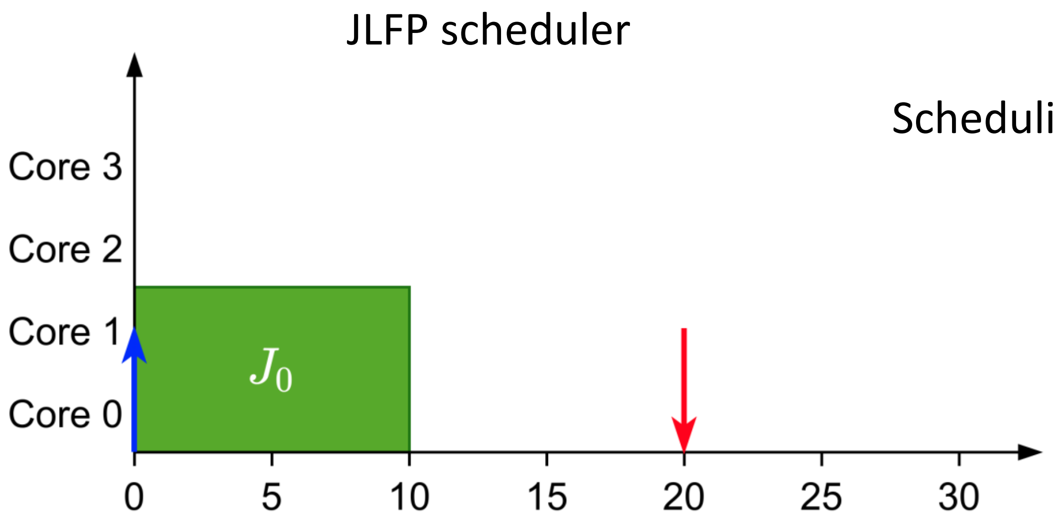
New Scheduler



Scheduling: J_1

	$s_j^{\min}$	$s_j^{\max}$	d_i	$c_i(v)$
J_0	2	2	100	10
J_1	3	3	20	5
J_2	1	1	100	20
J_3	1	3	100	24, 13, 20

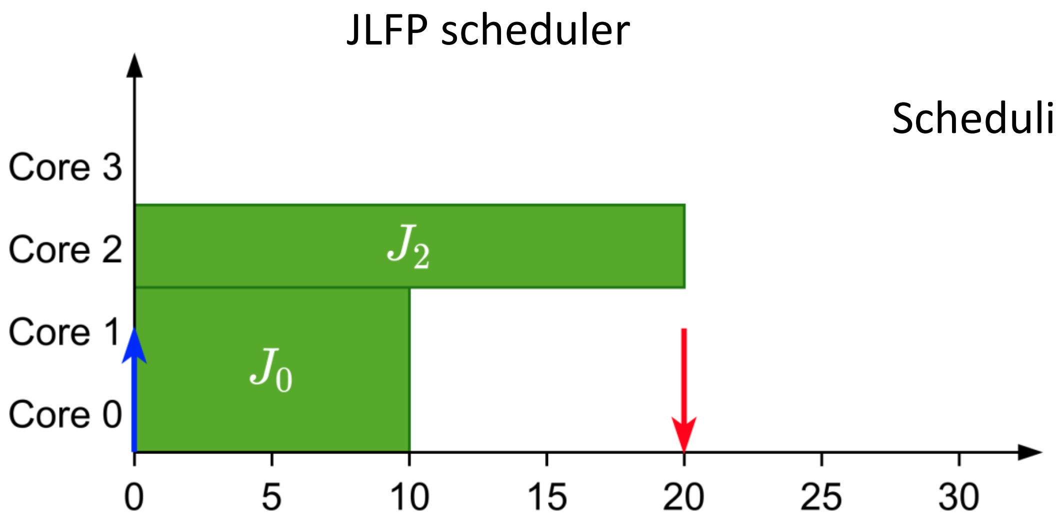
New Scheduler



Scheduling: J_2

	$s_j^{\min}$	$s_j^{\max}$	d_i	$c_i(v)$
J_0	2	2	100	10
J_1	3	3	20	5
J_2	1	1	100	20
J_3	1	3	100	24, 13, 20

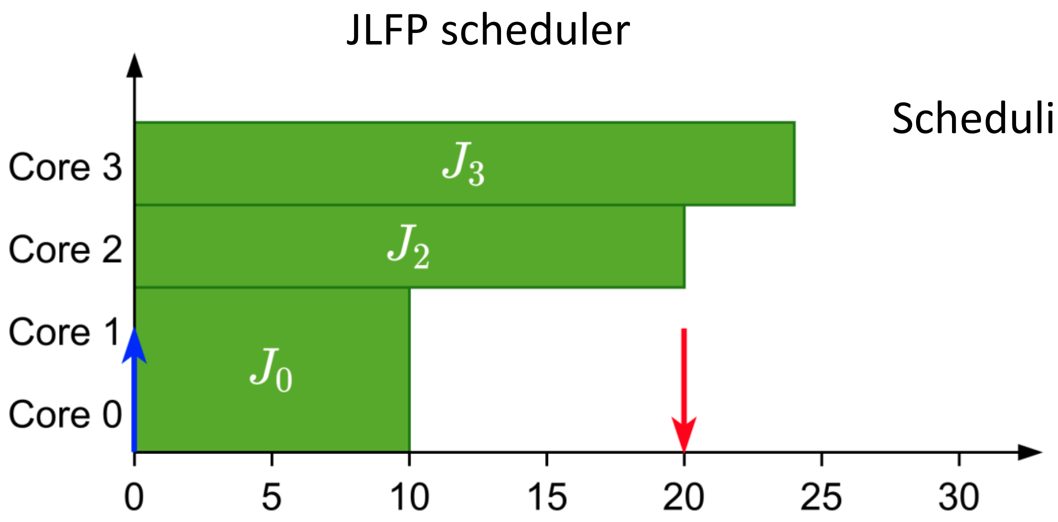
New Scheduler



Scheduling: J_3

	$s_j^{\min}$	$s_j^{\max}$	d_i	$c_i(v)$
J_0	2	2	100	10
J_1	3	3	20	5
J_2	1	1	100	20
J_3	1	3	100	24, 13, 20

New Scheduler

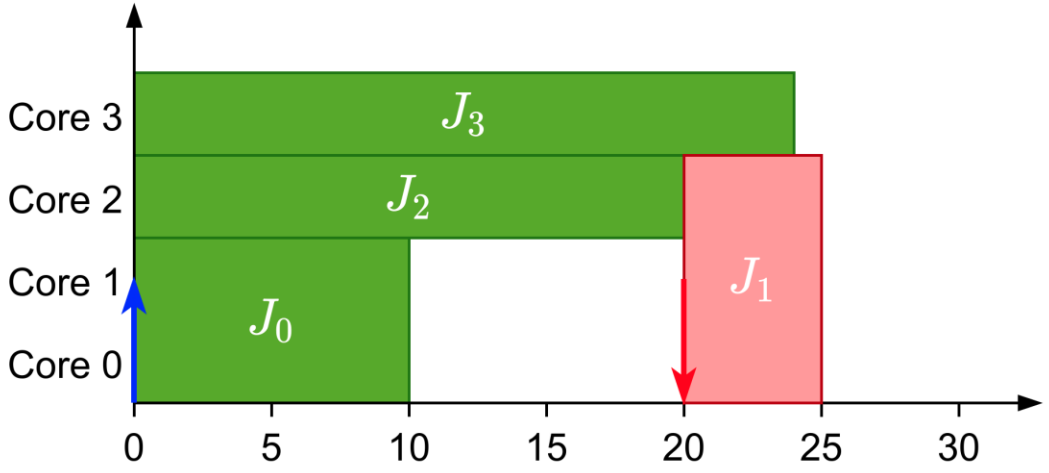


Scheduling: J_1

	$s_j^{\min}$	$s_j^{\max}$	d_i	$c_i(v)$
J_0	2	2	100	10
J_1	3	3	20	5
J_2	1	1	100	20
J_3	1	3	100	24, 13, 20

New Scheduler

JLFP scheduler



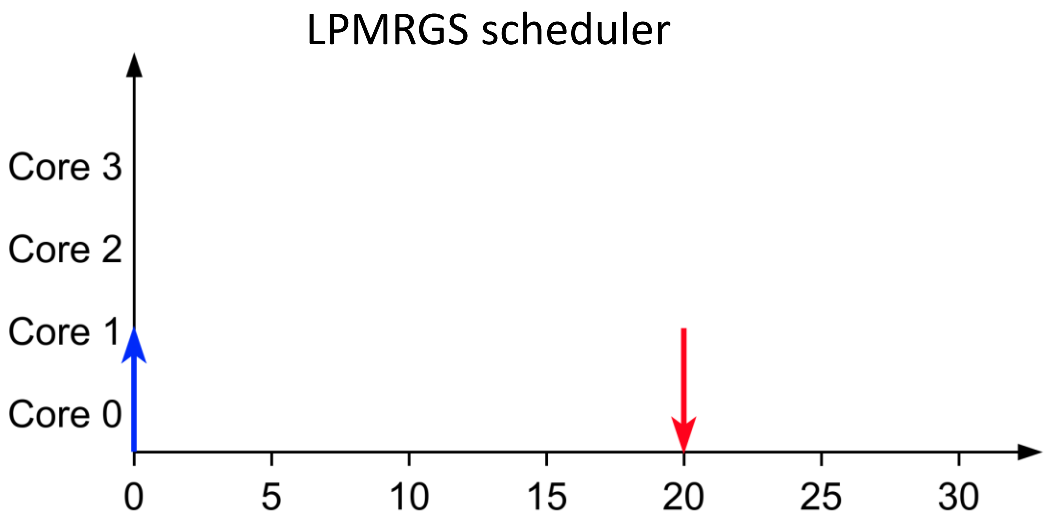
	$s_j^{\min}$	$s_j^{\max}$	d_i	$c_i(v)$
J_0	2	2	100	10
J_1	3	3	20	5
J_2	1	1	100	20
J_3	1	3	100	24, 13, 20

New Scheduler

- Limited-Preemptive Moldable Reservation Gang Scheduler

New Scheduler

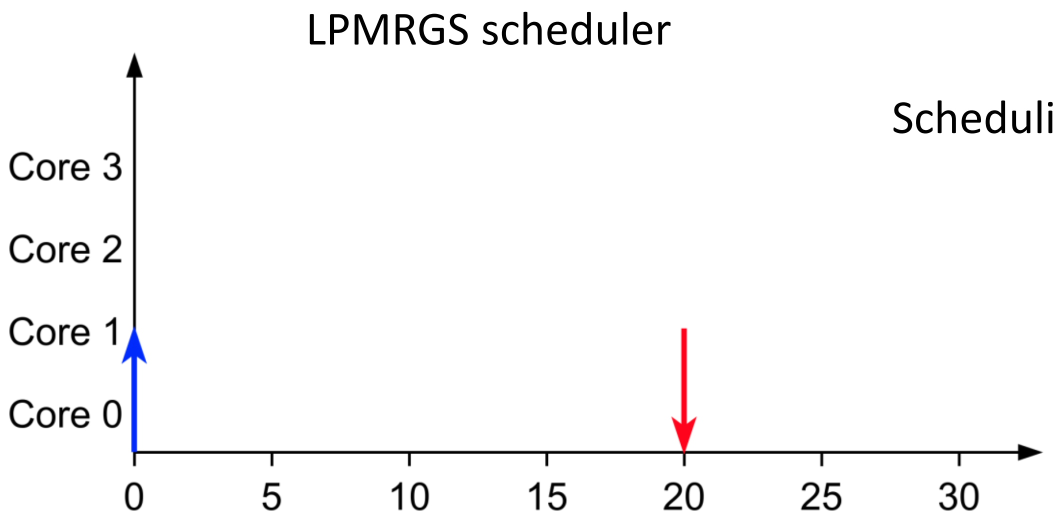
- Limited-Preemptive Moldable Reservation Gang Scheduler



	$s_j^{\min}$	$s_j^{\max}$	d_i	$c_i(v)$
J_0	2	2	100	10
J_1	3	3	20	5
J_2	1	1	100	20
J_3	1	3	100	24, 13, 20

New Scheduler

- Limited-Preemptive Moldable Reservation Gang Scheduler

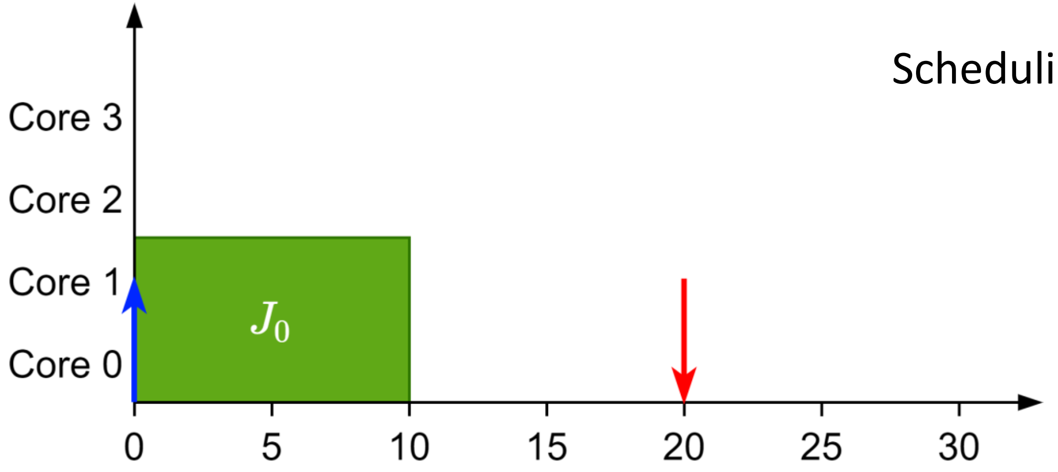


Scheduling: J_0

	$s_j^{\min}$	$s_j^{\max}$	d_i	$c_i(v)$
J_0	2	2	100	10
J_1	3	3	20	5
J_2	1	1	100	20
J_3	1	3	100	24, 13, 20

New Scheduler

- Limited-Preemptive Moldable Reservation Gang Scheduler

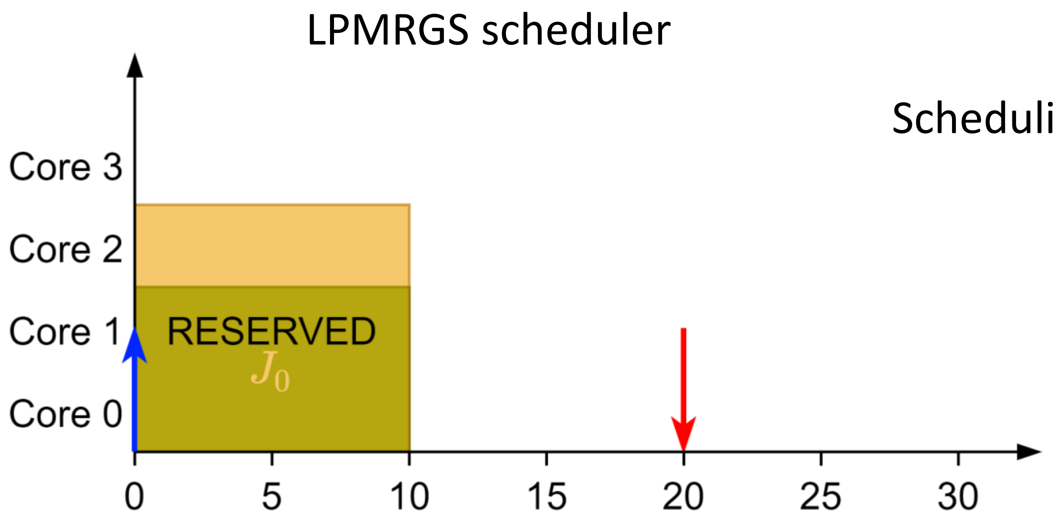


Scheduling: J_1

	$s_j^{\min}$	$s_j^{\max}$	d_i	$c_i(v)$
J_0	2	2	100	10
J_1	3	3	20	5
J_2	1	1	100	20
J_3	1	3	100	24, 13, 20

New Scheduler

- Limited-Preemptive Moldable Reservation Gang Scheduler

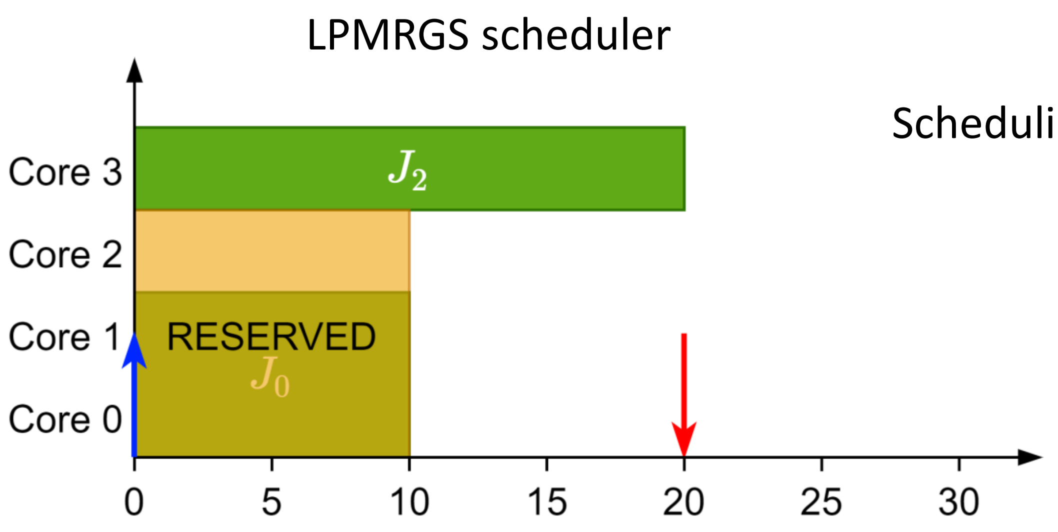


Scheduling: J_2

	$s_j^{\min}$	$s_j^{\max}$	d_i	$c_i(v)$
J_0	2	2	100	10
J_1	3	3	20	5
J_2	1	1	100	20
J_3	1	3	100	24, 13, 20

New Scheduler

- Limited-Preemptive Moldable Reservation Gang Scheduler

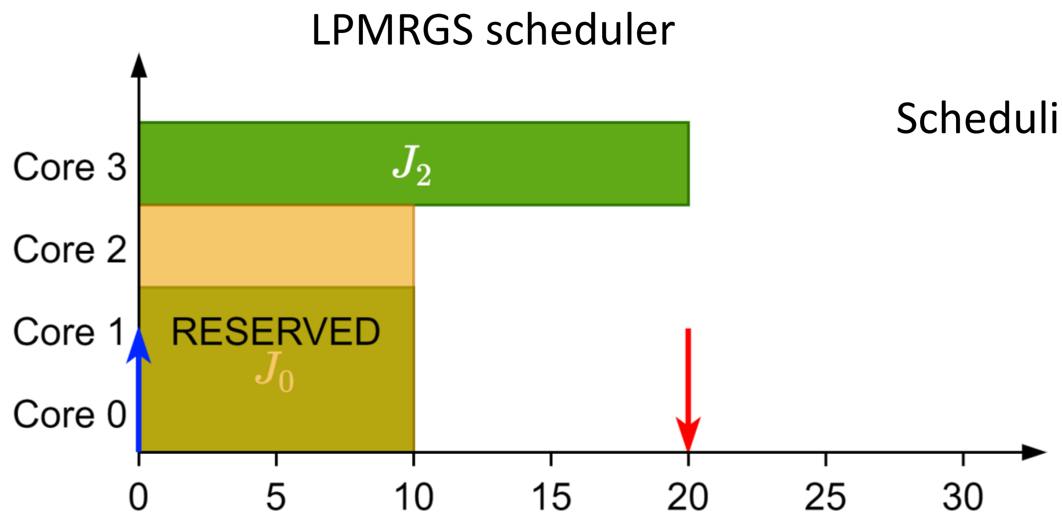


Scheduling: J_3

	$s_j^{\min}$	$s_j^{\max}$	d_i	$c_i(v)$
J_0	2	2	100	10
J_1	3	3	20	5
J_2	1	1	100	20
J_3	1	3	100	24, 13, 20

New Scheduler

- Limited-Preemptive Moldable Reservation Gang Scheduler



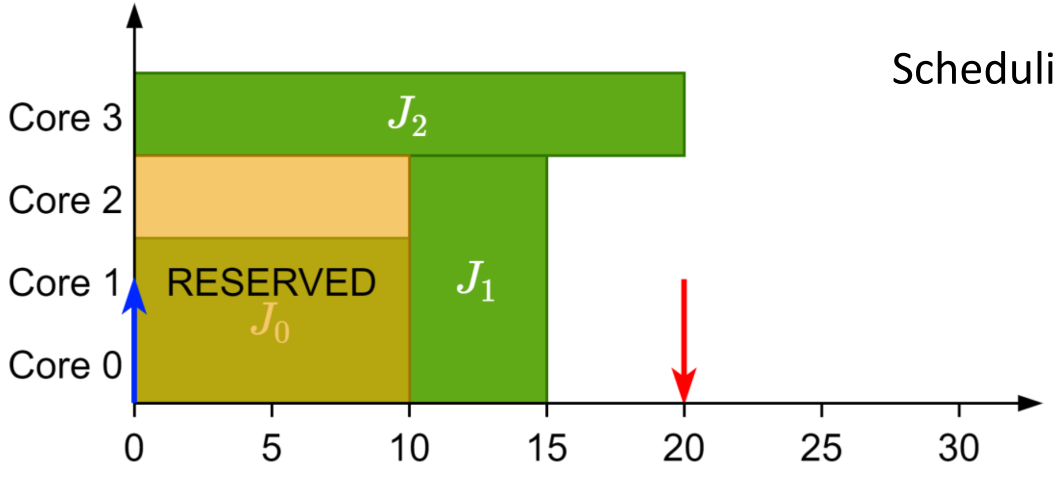
Scheduling: J_1

	$s_j^{\min}$	$s_j^{\max}$	d_i	$c_i(v)$
J_0	2	2	100	10
J_1	3	3	20	5
J_2	1	1	100	20
J_3	1	3	100	24, 13, 20

New Scheduler

- Limited-Preemptive Moldable Reservation Gang Scheduler

LPMRGS scheduler

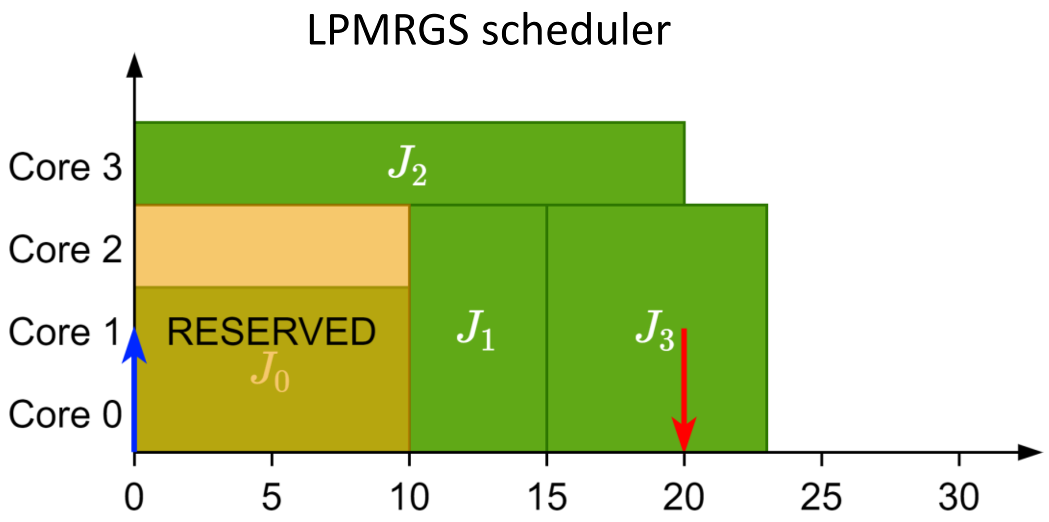


Scheduling: J_3

	$s_j^{\min}$	$s_j^{\max}$	d_i	$c_i(v)$
J_0	2	2	100	10
J_1	3	3	20	5
J_2	1	1	100	20
J_3	1	3	100	24, 13, 20

New Scheduler

- Limited-Preemptive Moldable Reservation Gang Scheduler

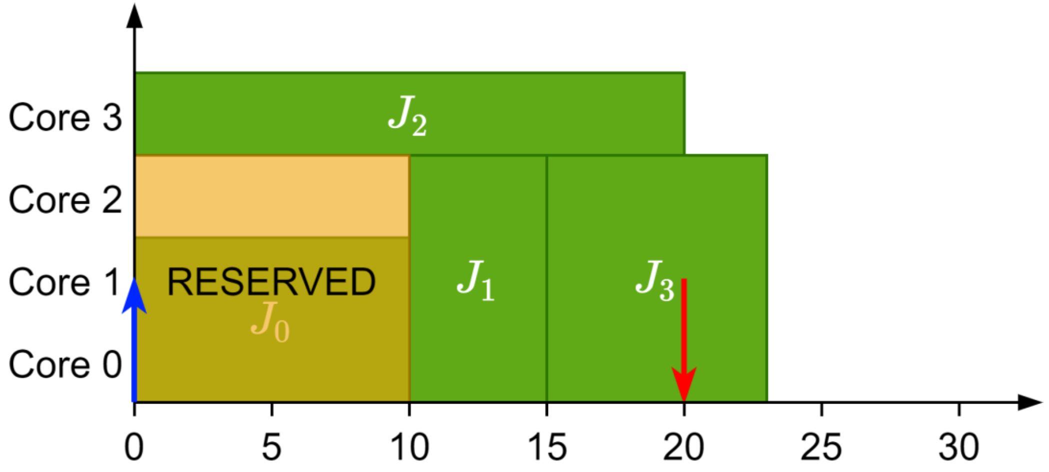


	$s_j^{\min}$	$s_j^{\max}$	d_i	$c_i(v)$
J_0	2	2	100	10
J_1	3	3	20	5
J_2	1	1	100	20
J_3	1	3	100	24, 13, 20

New Scheduler

- Limited-Preemptive Moldable Reservation Gang Scheduler
- Non-work conserving scheduler
- Reserve cores of higher-priority tasks and distribute the remaining ones among lower priority tasks

LPMRGS scheduler



	$s_j^{\min}$	$s_j^{\max}$	d_i	$c_i(v)$
J_0	2	2	100	10
J_1	3	3	20	5
J_2	1	1	100	20
J_3	1	3	100	24, 13, 20

Results

Questions?