Scheduling and Analys of Limited-Preemptive Modable Gang Tasks

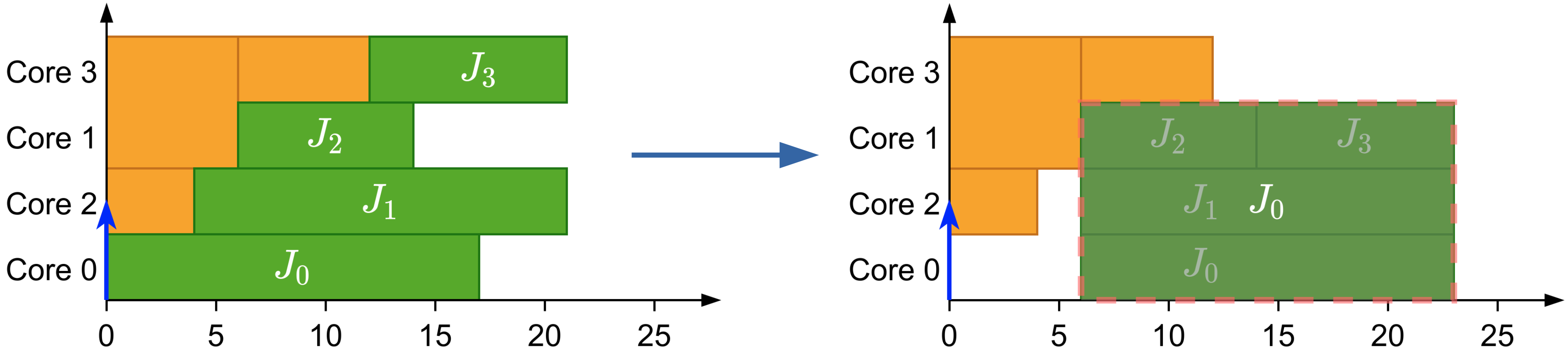
Joan Marcè i Igual Geoffrey Nelissen Mitra Nasri Paris Panagiotou

24th of February, 2020

TUDelft

What is gang?

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

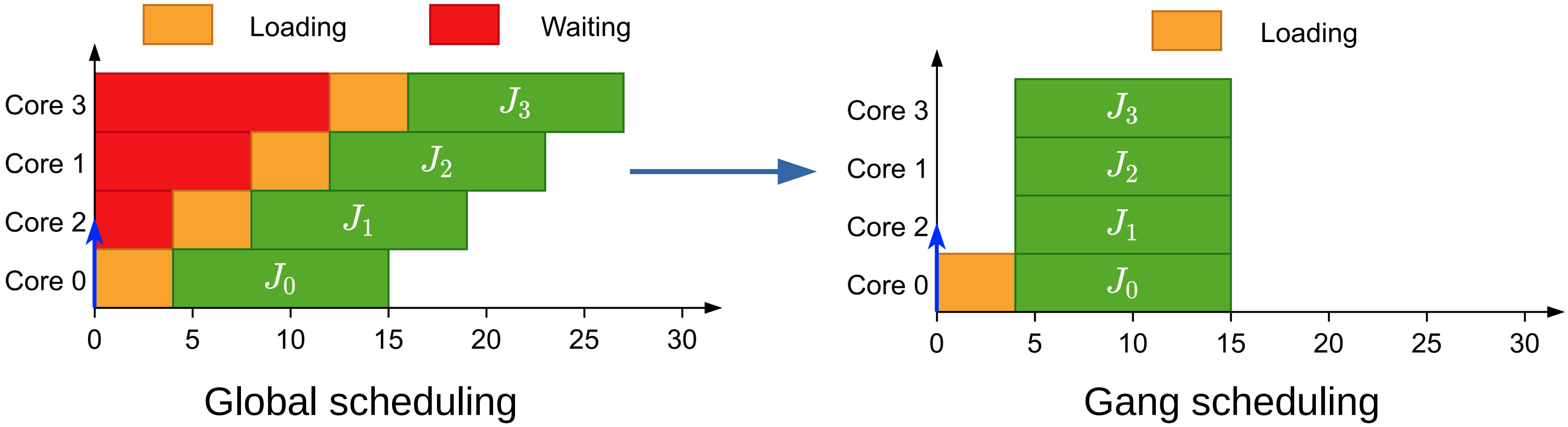


Global scheduling

Gang scheduling

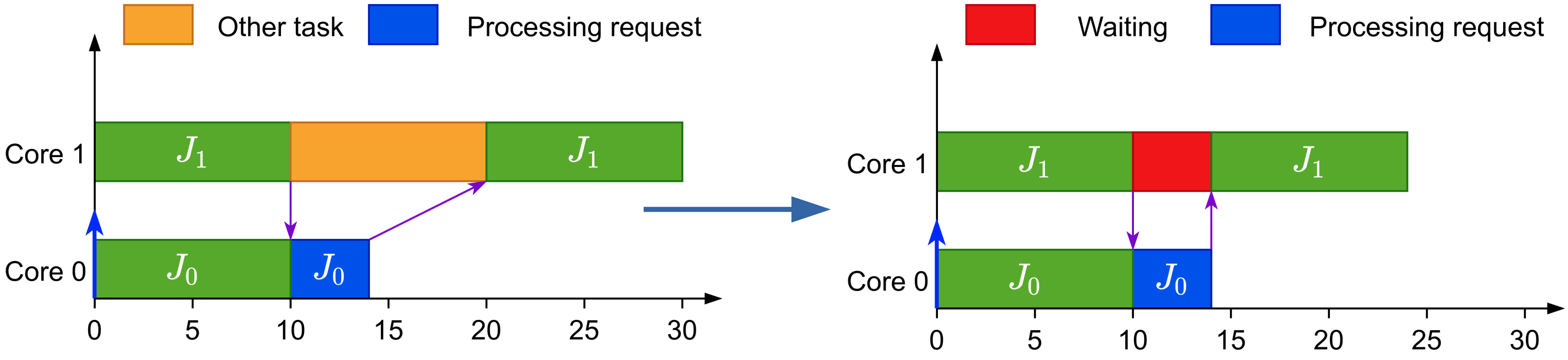
Why gang?

- Avoids overhead when loading initial data



Why gang?

- Avoids overhead when loading initial data
- Allows synchronization

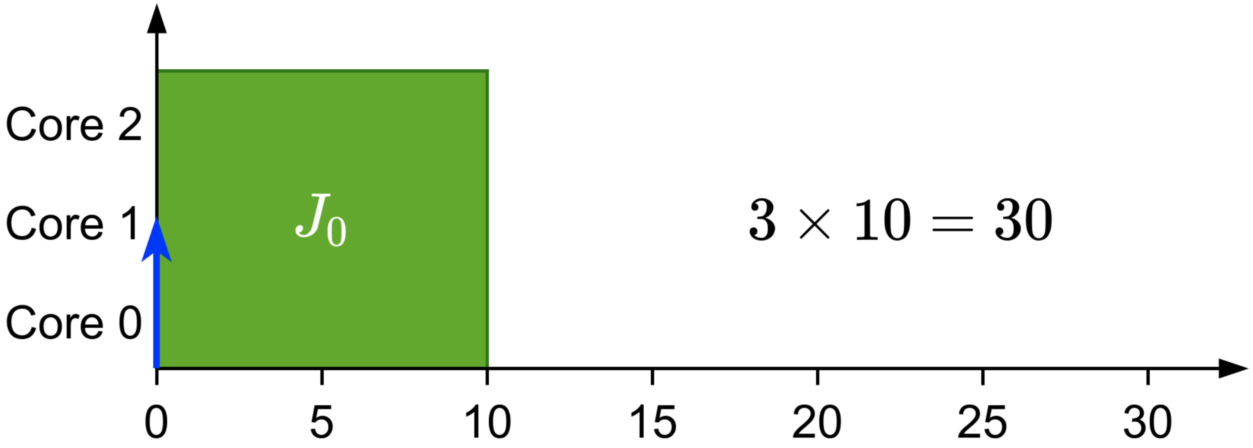


Global scheduling

Gang scheduling

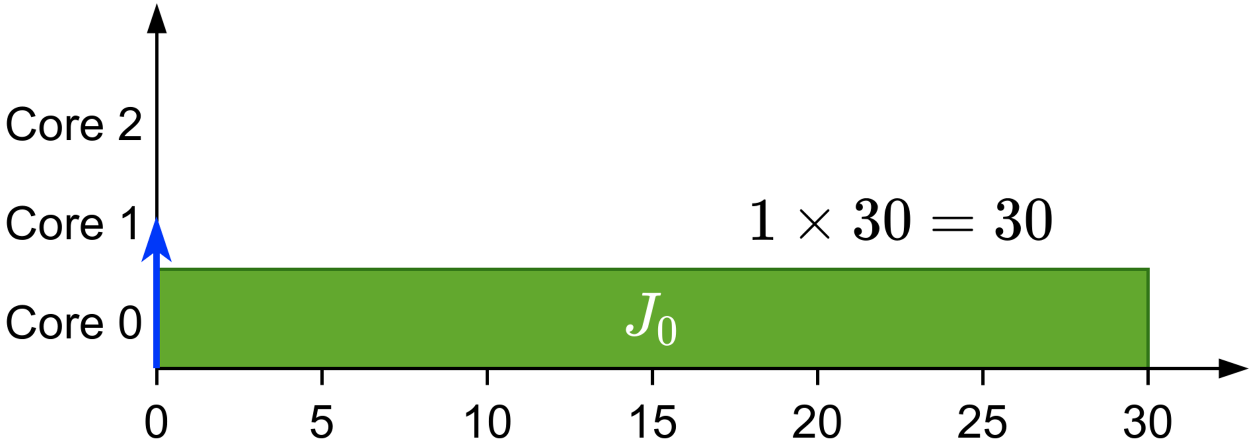
Types of gang

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



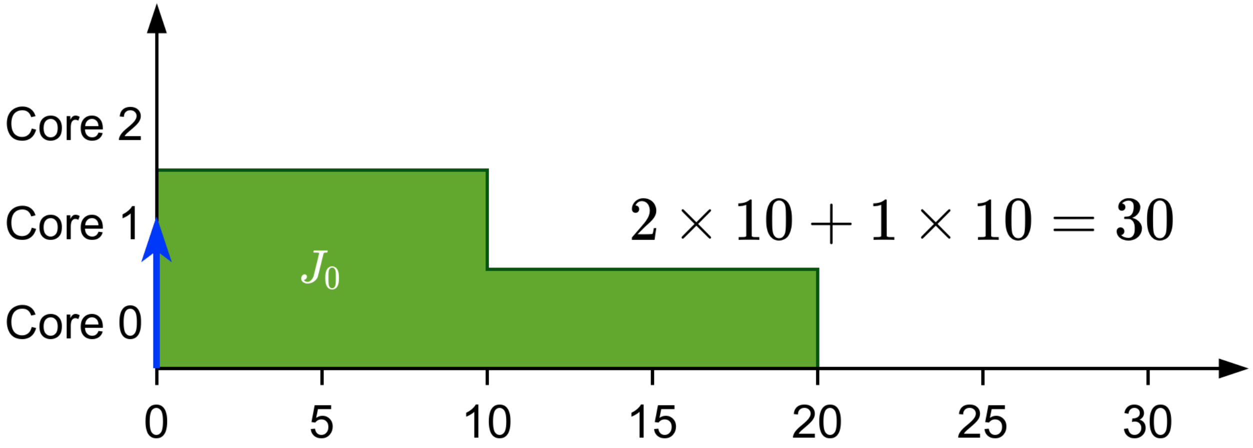
Types of gang

- **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



Previous work

- Introduced in the context of high-performance computing[1]
- In real-time:
  - For rigid tasks:
    - Job-Level Fixed-Priority is not predictable[2]
    - An optimal scheduler (DP-Fair) exists for preemptive tasks[3]
  - For moldable tasks
    - Global EDF has been adapted[4]
    - Preemptive scheduler that chooses cores to meet the deadline[5]

[1] Ousterhout, 1982

[2] Goossens et al., 2010

[3] Goossens et al., 2016

[4] Kato et al.,2009

[5] Berten et al., 2011

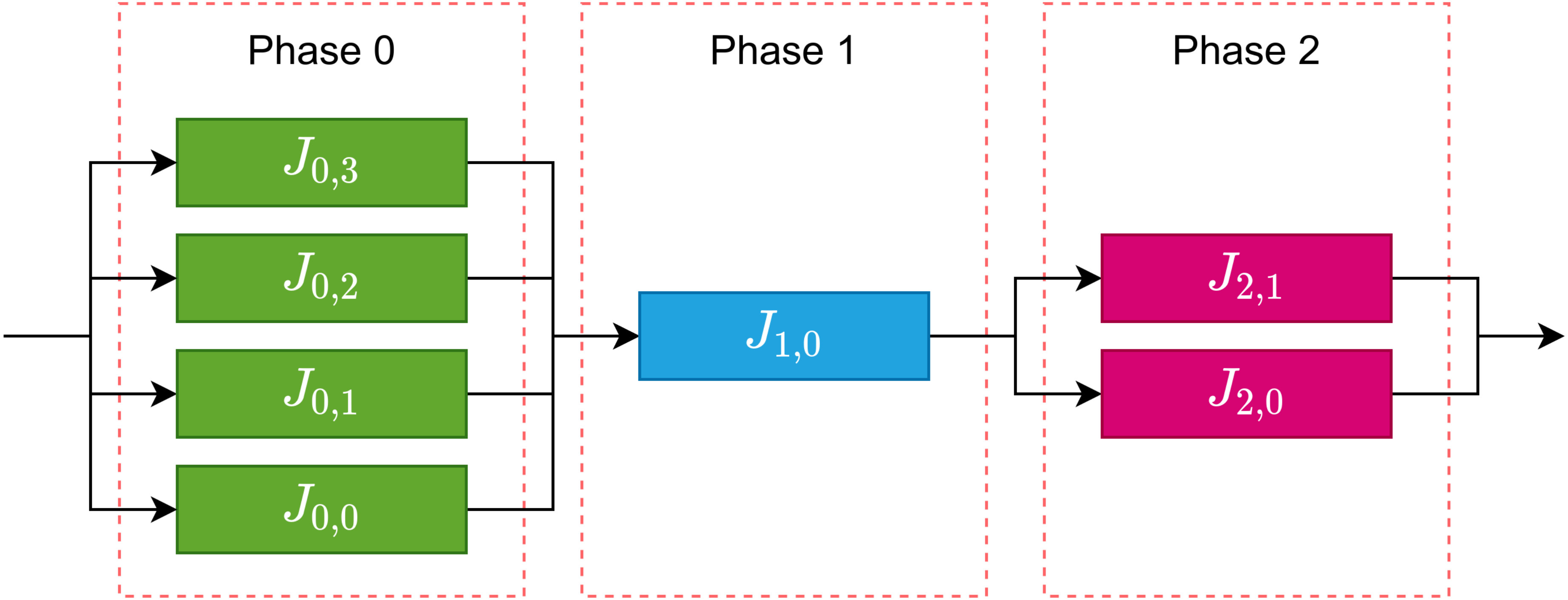
Previous work

- In real-time:
  - For malleable tasks:
    - An optimal preemptive scheduler, in terms of processors, has been proposed[6]
  - Bundled task-model[7]:
    - Preemptive rigid gang tasks
    - Tasks with precedence constraints modeled as a succession of “bundles”
    - Our limited-preemptive definition comes from here

[6] Collette et al., 2008

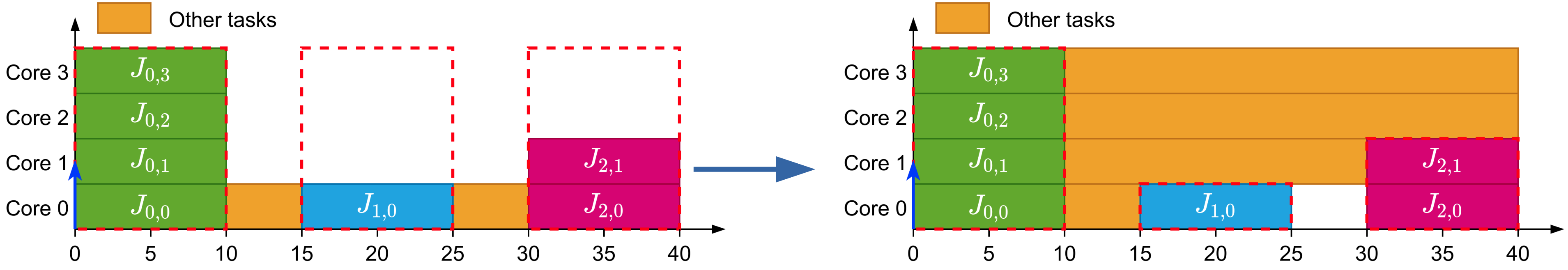
[7] Wasly et al., 2017

Limited-Preemptive



Limited-Preemptive

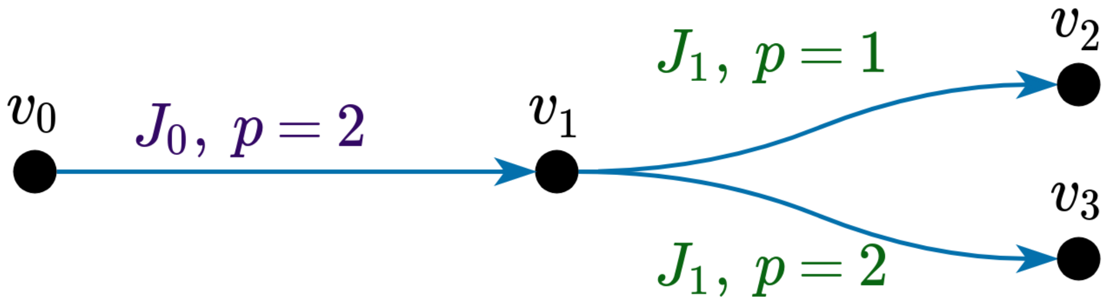
- Rigid gang could ask for cores that does not use
- Bundled[7] asks only the required cores but preemptions can happen inside a bundle
- LP only allows preemptions between bundles



[7] Wasly et al., 2017

Schedulability analysis

- Accurate and relatively fast analysis
- Based on the notion of Schedule Abstraction Graph
- Faster than an exact analysis
- Not as pessimistic as closed-form analyses



Our work

- We aim to extend schedulability analysis to moldable gang under the Job-Level Fixed Priority scheduler
  - Many different scenarios
  - Scheduler has to decide
    - When to release a job
    - How many cores to assign to this job
  - This could lead to state-space explosion

Analysis

- $A_p^{\min}$ Time at which we have p cores **possibly** available
- $A_p^{\max}$ Time at which we have p cores **certainly** available
- EST_i^p Earliest Start Time of job i with p cores
- LST_i^p Latest Start Time of job i with p cores
- EFT_i^p Earliest Finishing Time of job i with p cores
- LFT_i^p Latest Finishing Time of job i with p cores

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

Analysis

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

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

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

- Job cannot start with p cores after:
  - $p+1$ are avaible
  - A lower priority task can start
  - A higher priority task can start

Analysis

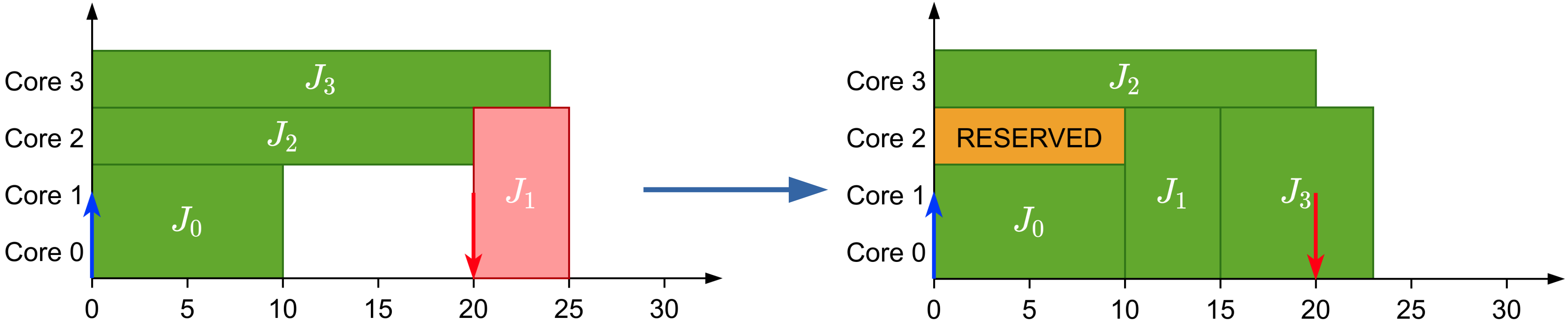
- Obtain EFT_i^p and LFT_i^p from job i

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

- Which allows us to compute $A_p^{\min}$ and $A_p^{\max}$

LPMRGS

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



Questions?