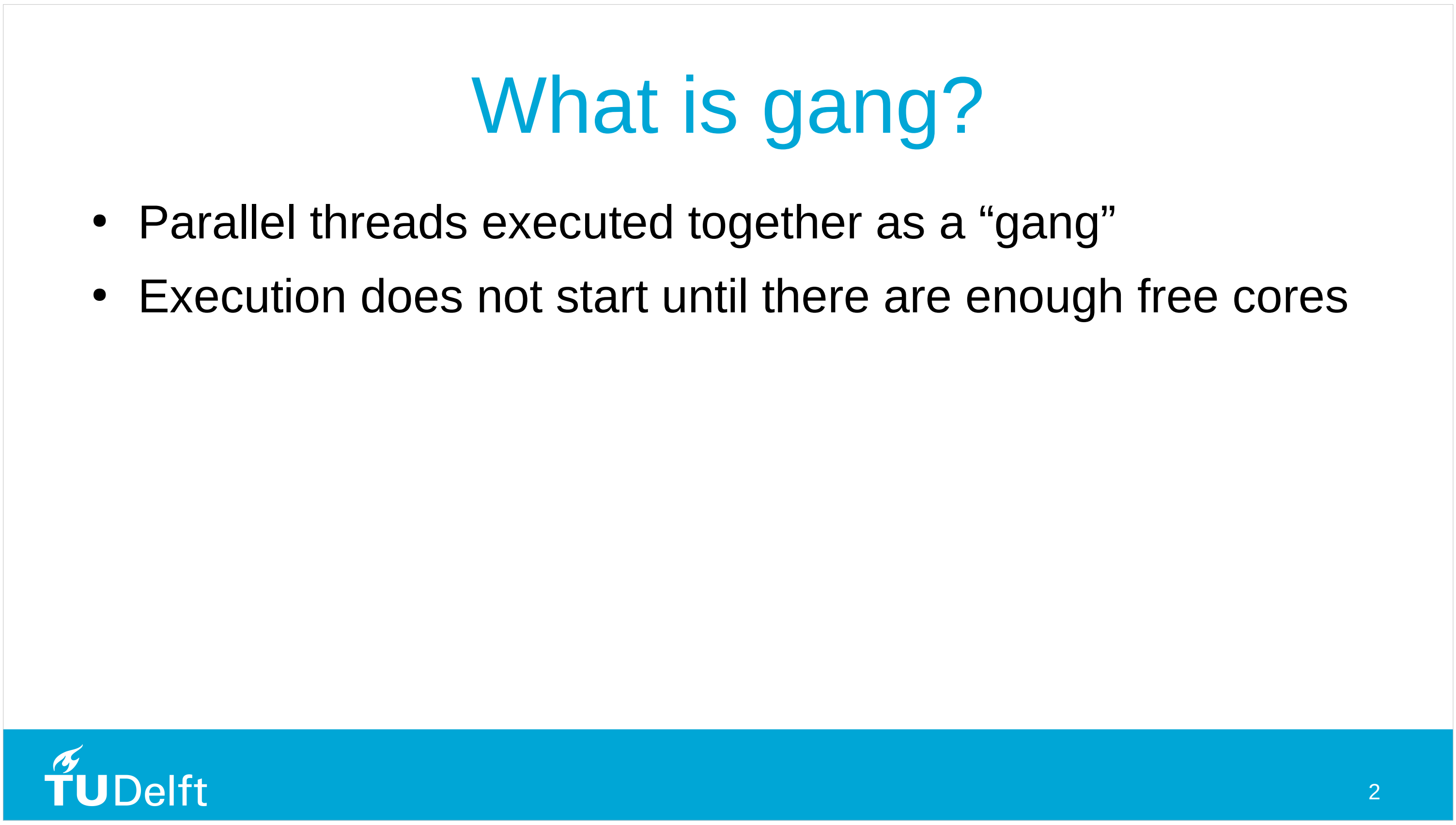
Scheduling and Analys of Limited-Preemptive Modable Gang Tasks

Joan Marcè i Igual Geoffrey Nelissen Mitra Nasri Paris Panagiotou

24th of February, 2020



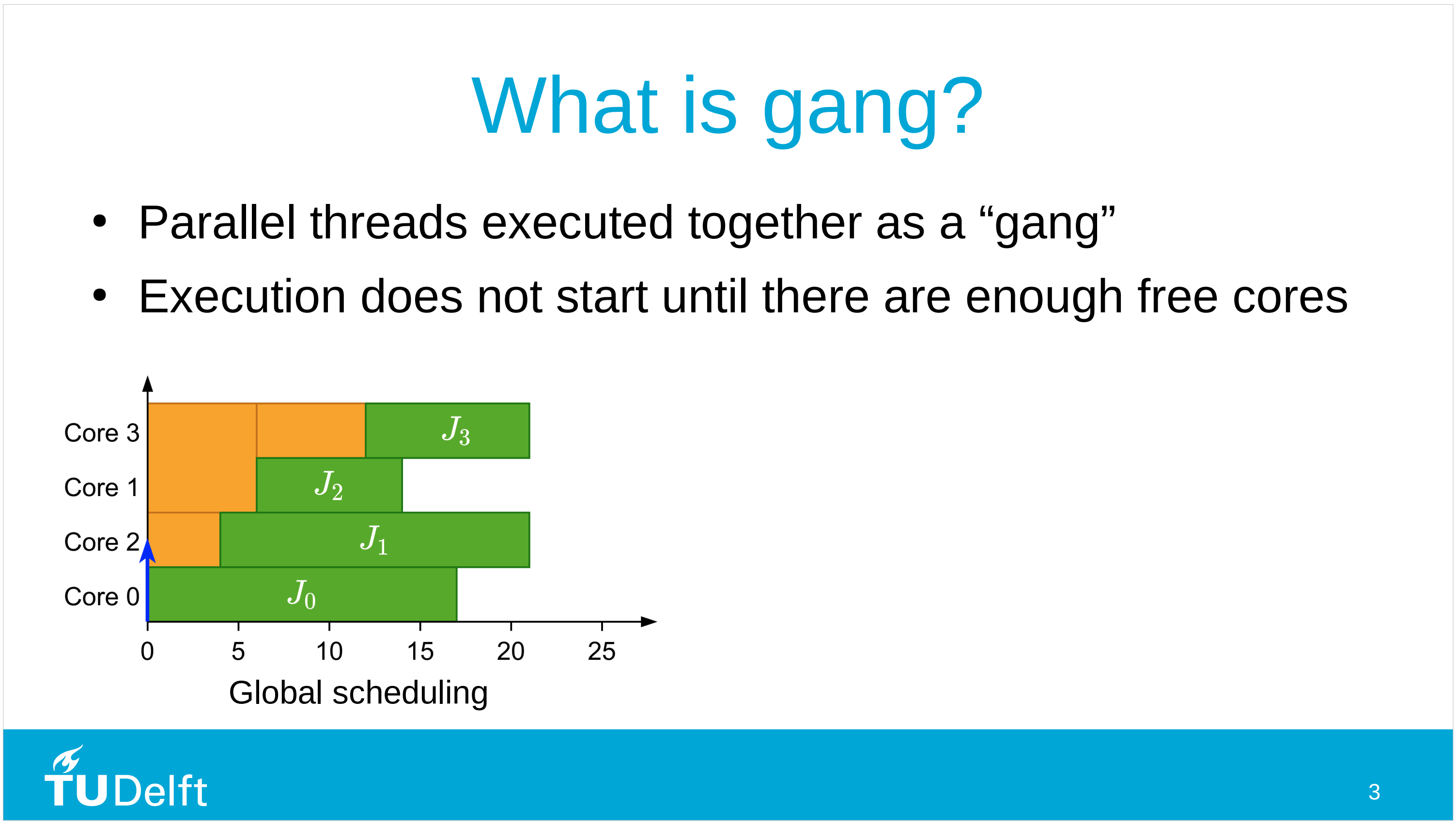
What is gang?

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

First of all, let’s explain what gang scheduling is.

It’s the execution of multiple parallel threads together as a “gang”. In these threads the execution does not start until there are enough cores to execute them together.
Let’s see an example:<click>

1.We have these jobs assigned to different cores.
2.If we release them as a gang job then we should pack them like a single task. Obtaining the following group. <click>
3.Then,<click> we have obtained the gang task result of merging the jobs



What is gang?

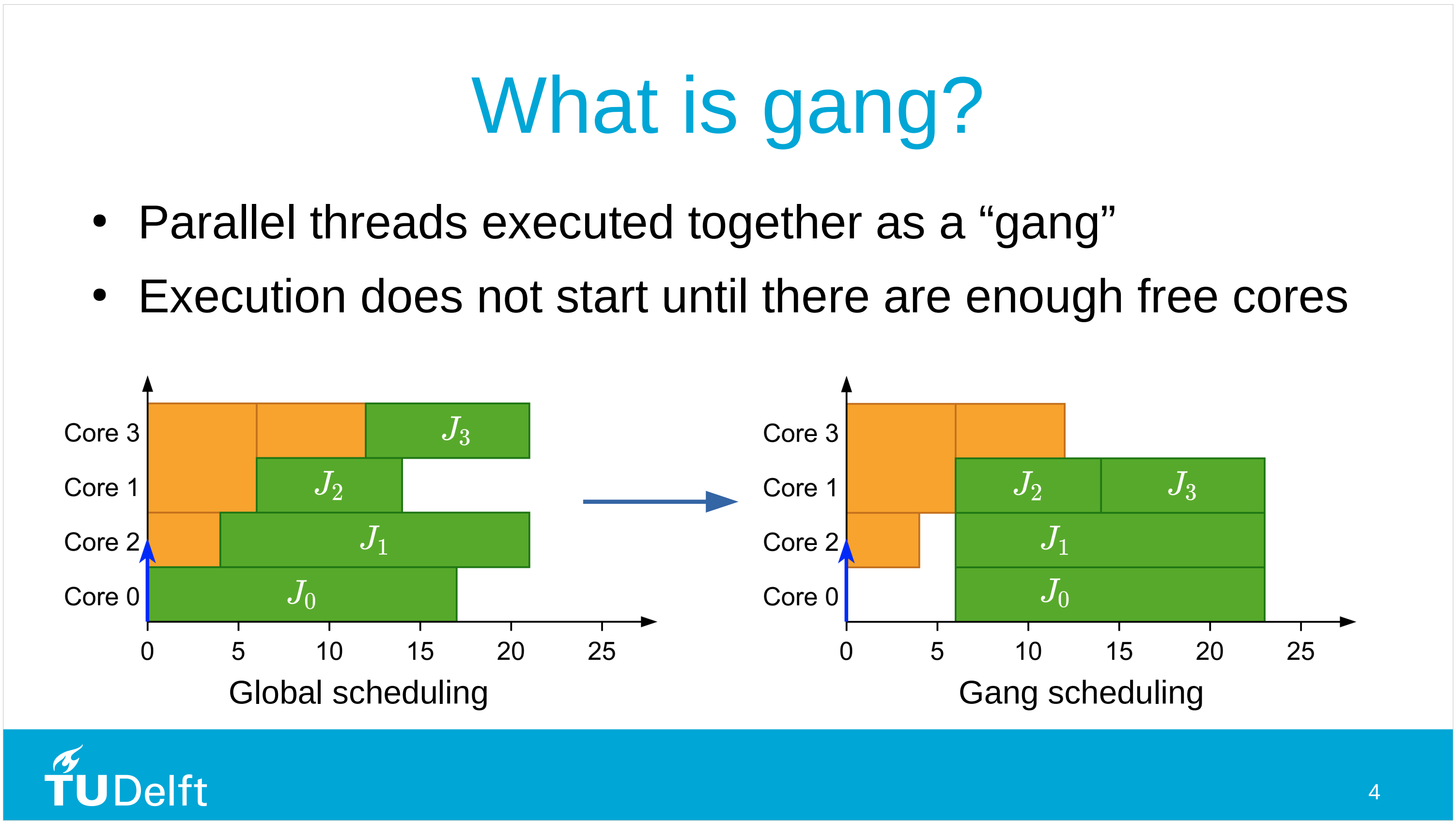
- Parallel threads executed together as a “gang”
- Execution does not start until there are enough free cores

First of all, let’s explain what gang scheduling is.

It’s the execution of multiple parallel threads together as a “gang”. In these threads the execution does not start until there are enough cores to execute them together.

Let’s see an example:<click>

1. We have these jobs assigned to different cores.
2. If we release them as a gang job then we should pack them like a single task. Obtaining the following group. <click>
3. Then,<click> we have obtained the gang task result of merging the jobs



What is gang?

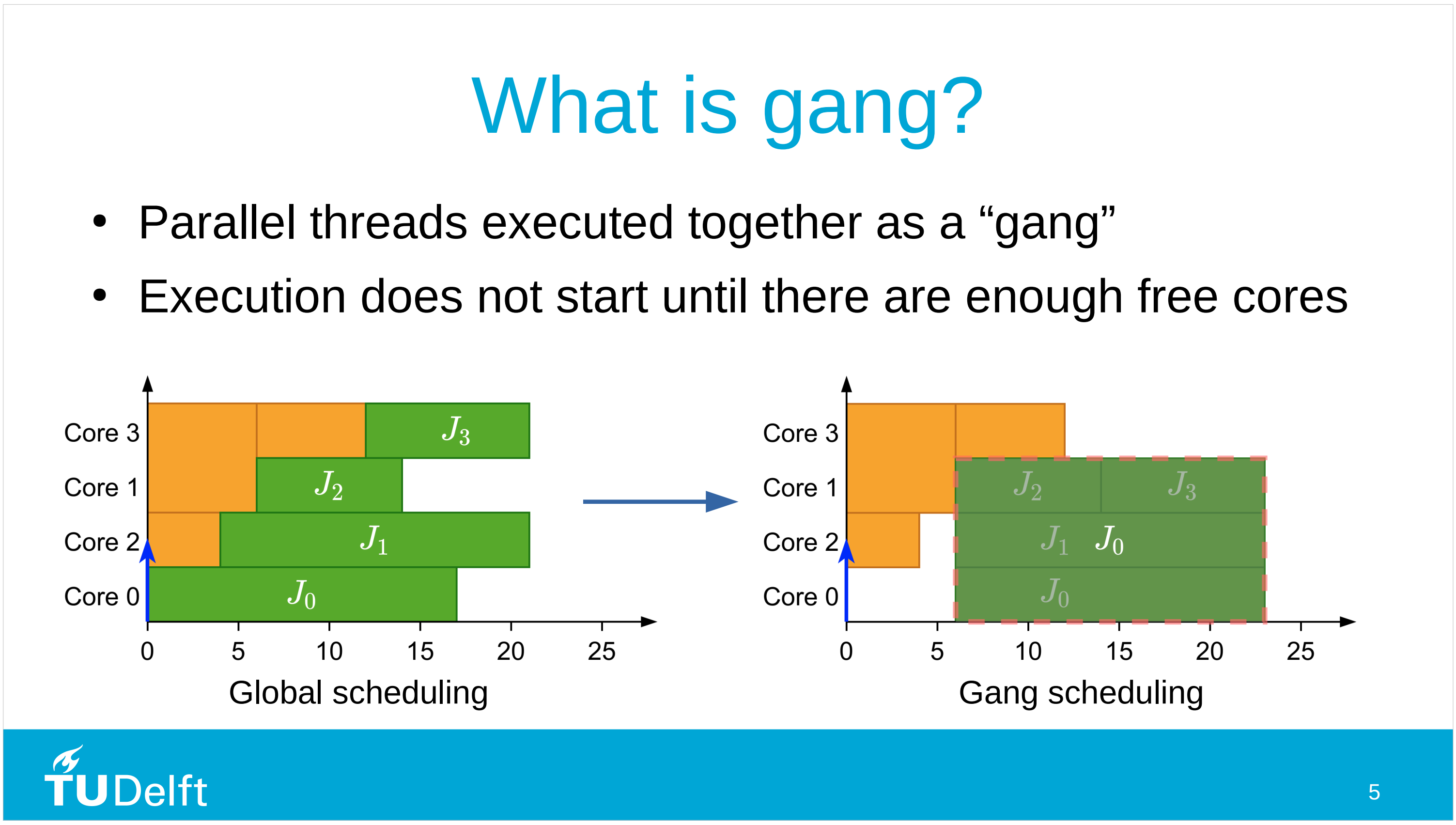
- Parallel threads executed together as a “gang”
- Execution does not start until there are enough free cores

First of all, let’s explain what gang scheduling is.

It’s the execution of multiple parallel threads together as a “gang”. In these threads the execution does not start until there are enough cores to execute them together.

Let’s see an example:<click>

1. We have these jobs assigned to different cores.
2. If we release them as a gang job then we should pack them like a single task. Obtaining the following group. <click>
3. Then,<click> we have obtained the gang task result of merging the jobs



What is gang?

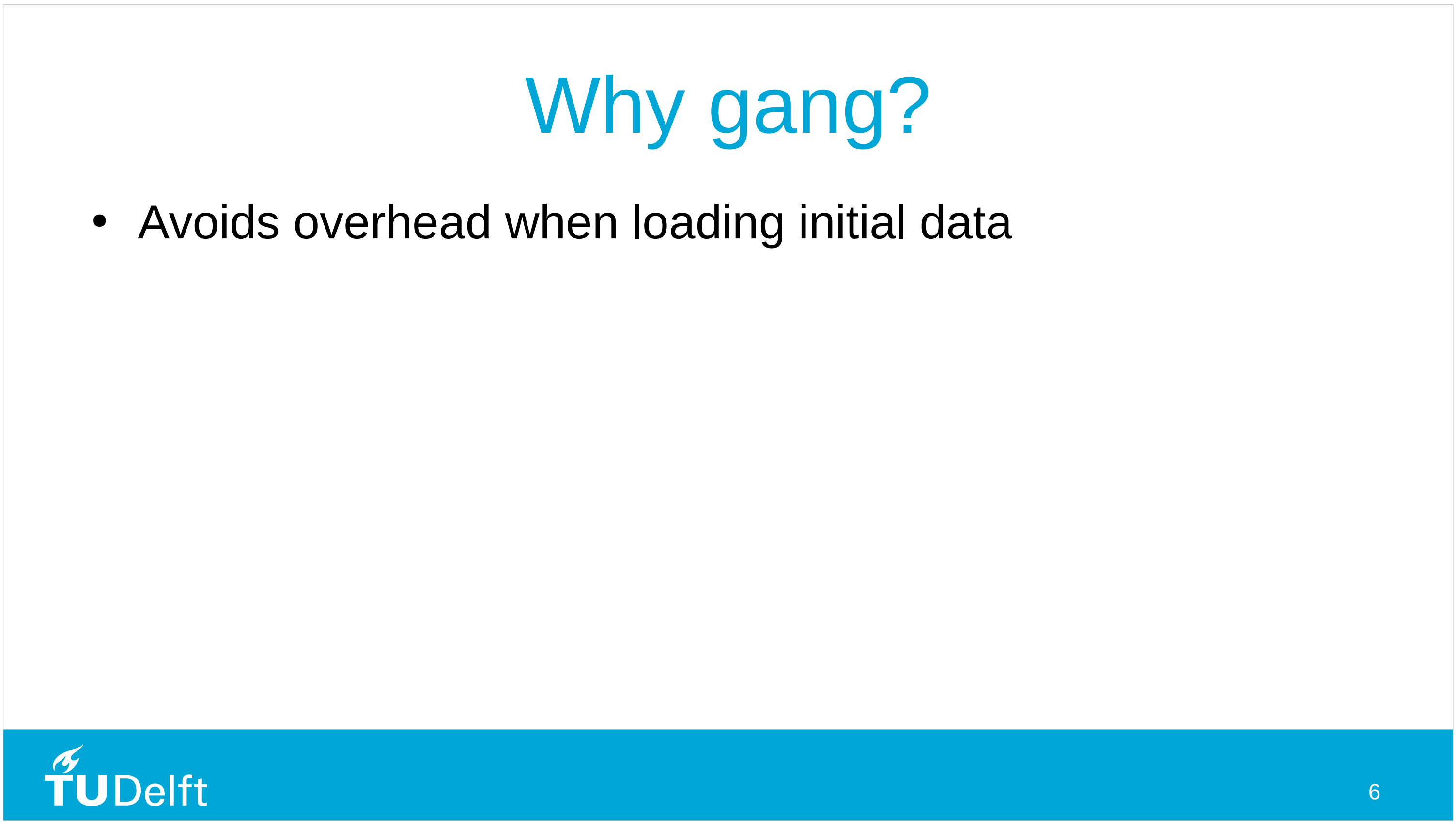
- Parallel threads executed together as a “gang”
- Execution does not start until there are enough free cores

First of all, let’s explain what gang scheduling is.

It’s the execution of multiple parallel threads together as a “gang”. In these threads the execution does not start until there are enough cores to execute them together.

Let’s see an example:<click>

1. We have these jobs assigned to different cores.
2. If we release them as a gang job then we should pack them like a single task. Obtaining the following group. <click>
3. Then,<click> we have obtained the gang task result of merging the jobs



Why gang?

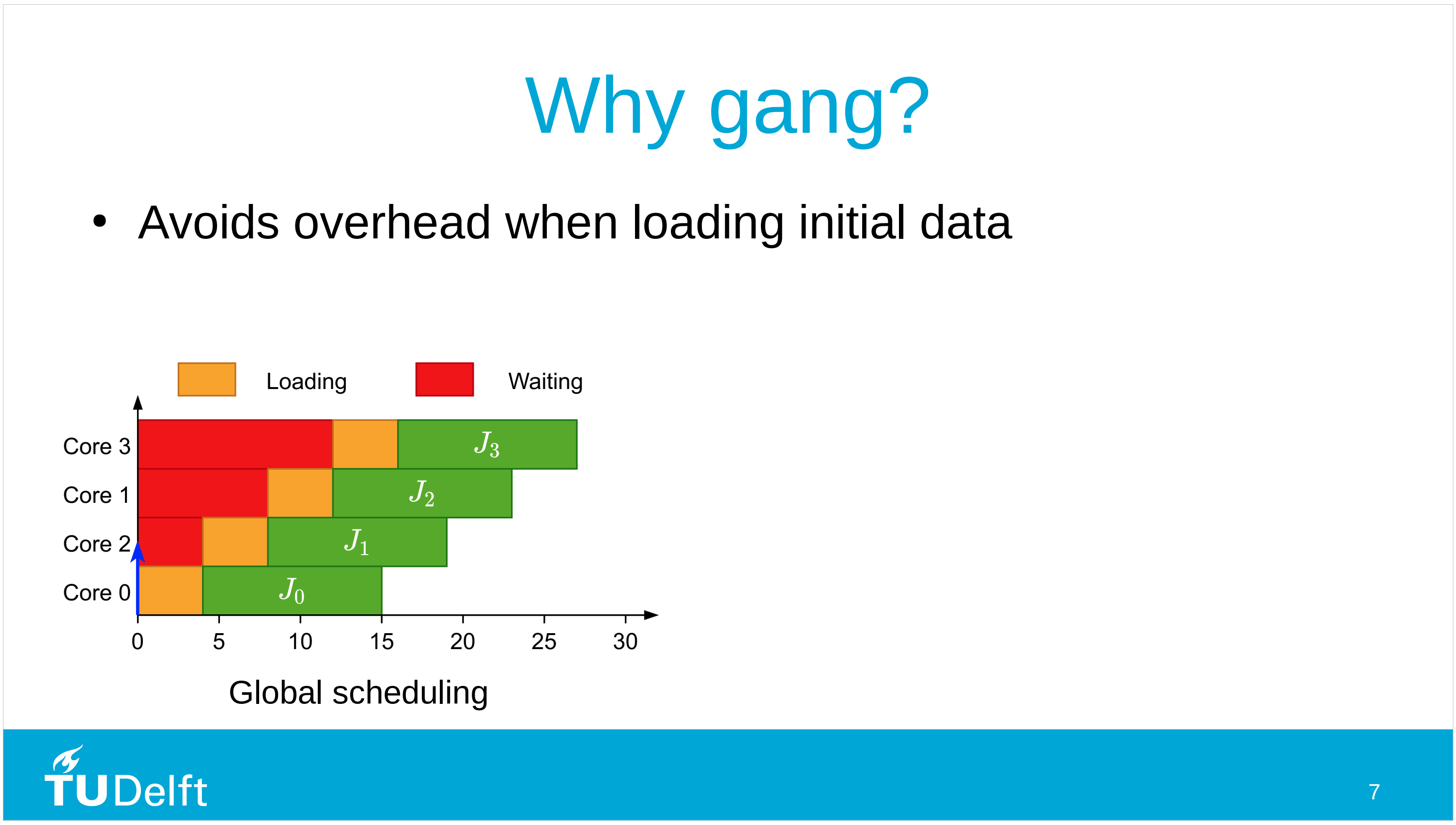
- Avoids overhead when loading initial data

So then, why do we need gang scheduling?

Well, it helps in reducing overhead when loading data initially.

<click>
In this example all these threads have been scheduled at the same time but have to wait until the previous one have finished loading.

On the contrary<click>, if we know that all these threads are starting at the same time we can load the data once for all the threads at the same time which is what we would do with gang scheduling.



Why gang?

- Avoids overhead when loading initial data

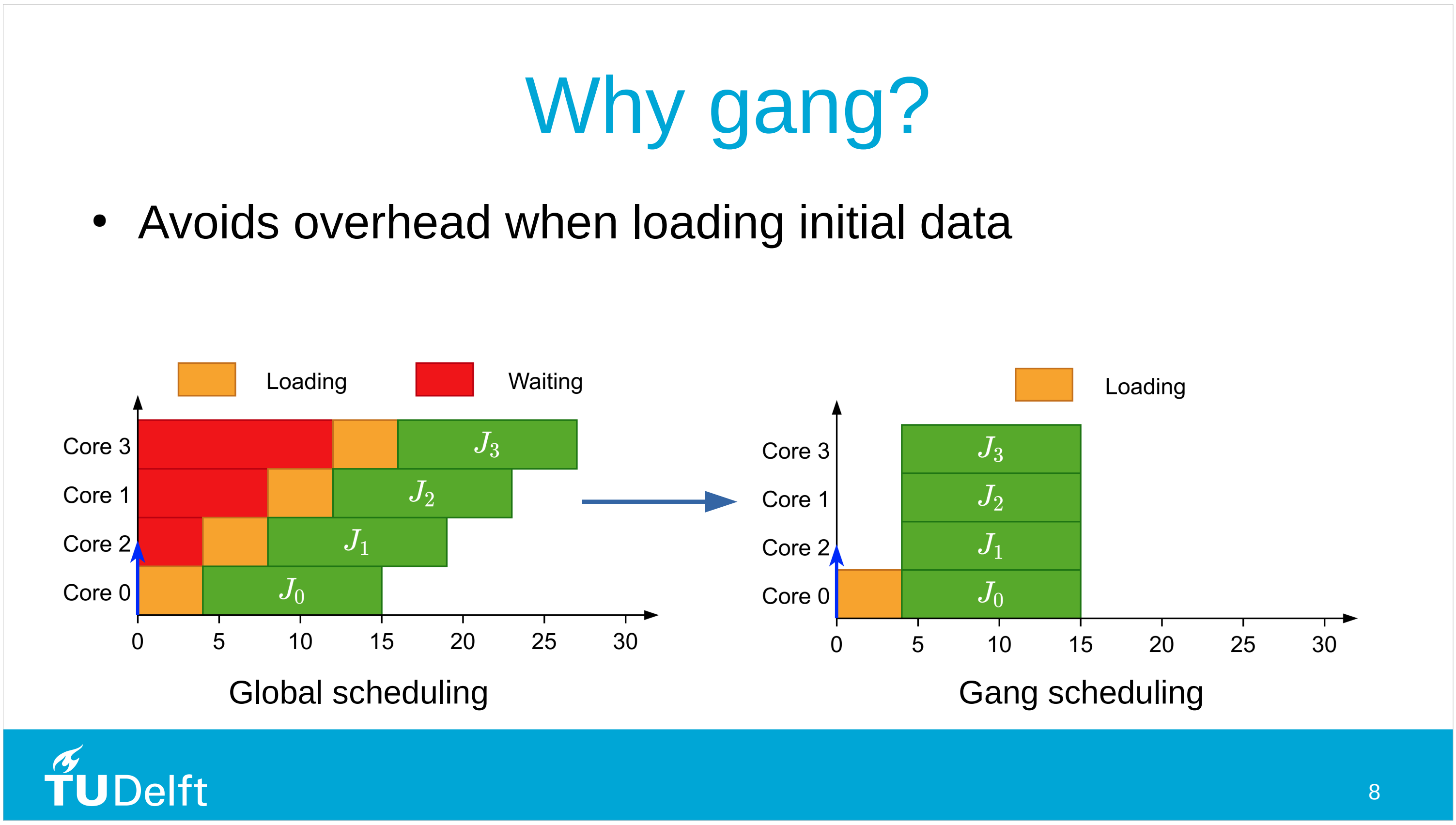
Global scheduling

So then, why do we need gang scheduling?

Well, it helps in reducing overhead when loading data initially.

<click>
In this example all these threads have been scheduled at the same time but have to wait until the previous one have finished loading.

On the contrary<click>, if we know that all these threads are starting at the same time we can load the data once for all the threads at the same time which is what we would do with gang scheduling.



Why gang?

- Avoids overhead when loading initial data

So then, why do we need gang scheduling?

Well, it helps in reducing overhead when loading data initially.

<click>
In this example all these threads have been scheduled at the same time but have to wait until the previous one have finished loading.

On the contrary<click>, if we know that all these threads are starting at the same time we can load the data once for all the threads at the same time which is what we would do with gang scheduling.

Why gang?

- Avoids overhead when loading initial data
- Allows synchronization

Global scheduling

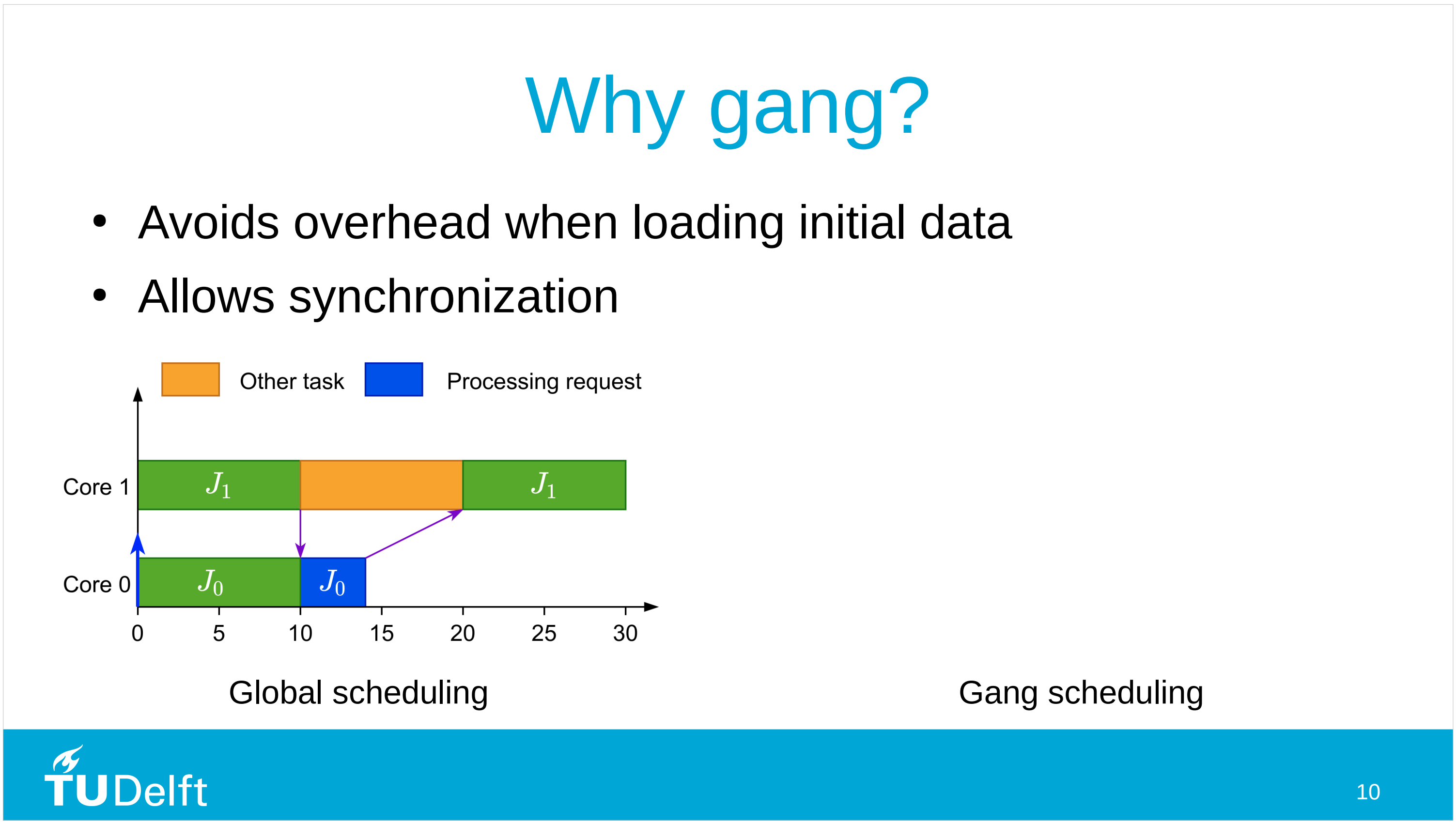
Gang scheduling

TUDelft

Additionally it helps in synchronization situations, like in the following example<click>

Here there are two jobs and job 1 sends some data to job 0 that is processed and sent back. In this case another task has been scheduled after job 1 sent the data and after the other task finishes then job 1 can continue its execution

On the other hand if we use gang<click> the job would wait for job 0 to finish the processing part and return the results as no other task would be allowed to execute while job 1 is waiting.



Why gang?

- Avoids overhead when loading initial data
- Allows synchronization

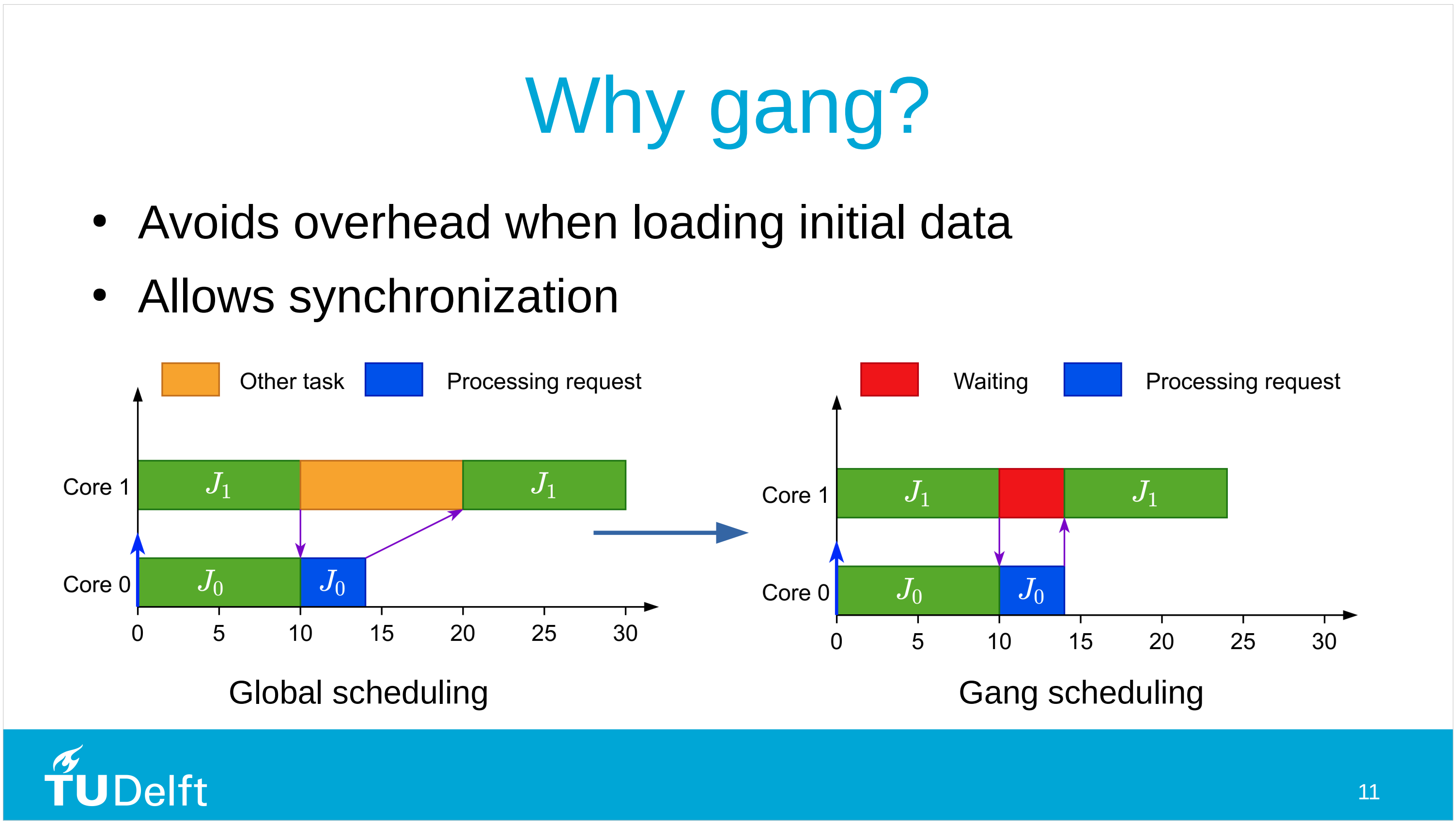
Global scheduling

Gang scheduling

Additionally it helps in synchronization situations, like in the following example<click>

Here there are two jobs and job 1 sends some data to job 0 that is processed and sent back. In this case another task has been scheduled after job 1 sent the data and after the other task finishes then job 1 can continue its execution

On the other hand if we use gang<click> the job would wait for job 0 to finish the processing part and return the results as no other task would be allowed to execute while job 1 is waiting.



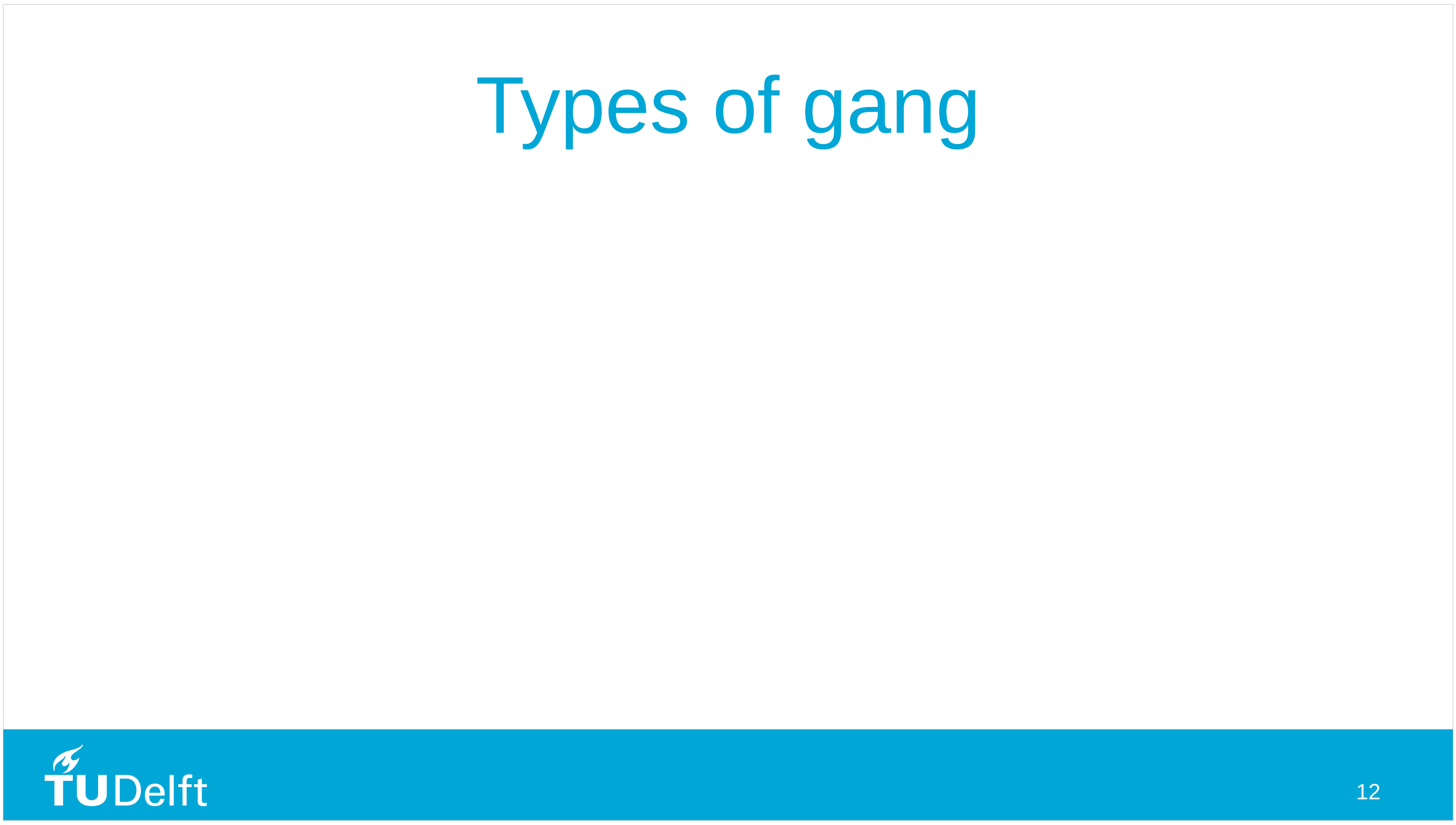
Why gang?

- Avoids overhead when loading initial data
- Allows synchronization

Additionally it helps in synchronization situations, like in the following example<click>

Here there are two jobs and job 1 sends some data to job 0 that is processed and sent back. In this case another task has been scheduled after job 1 sent the data and after the other task finishes then job 1 can continue its execution

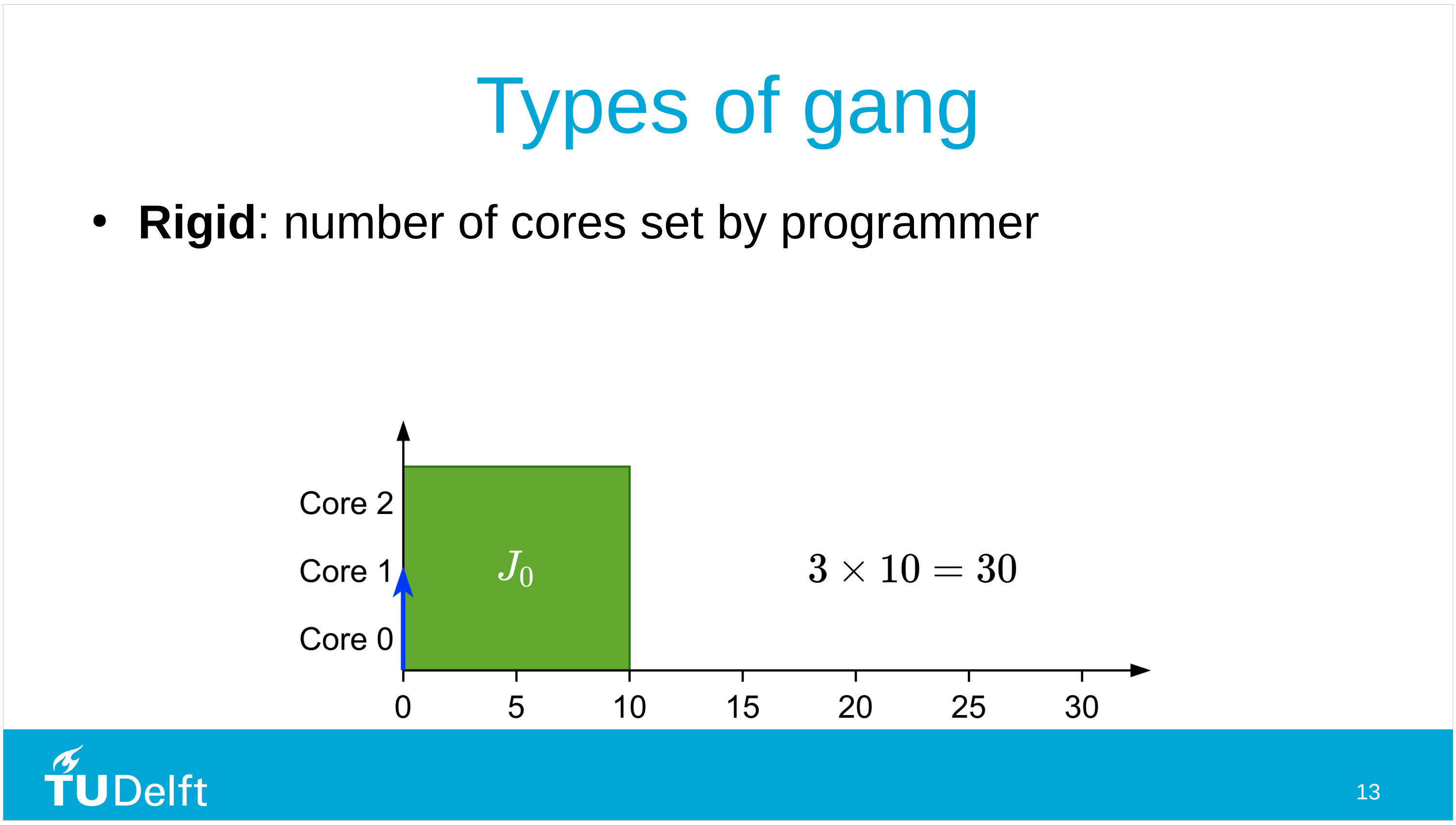
On the other hand if we use gang<click> the job would wait for job 0 to finish the processing part and return the results as no other task would be allowed to execute while job 1 is waiting.



Types of gang

When talking about gang scheduling, we can have three types:

<click>Rigid gang, where the number of cores is set by the programmer and fixed during execution. In this example this job executes with 3 cores and requires 10 time units to compute

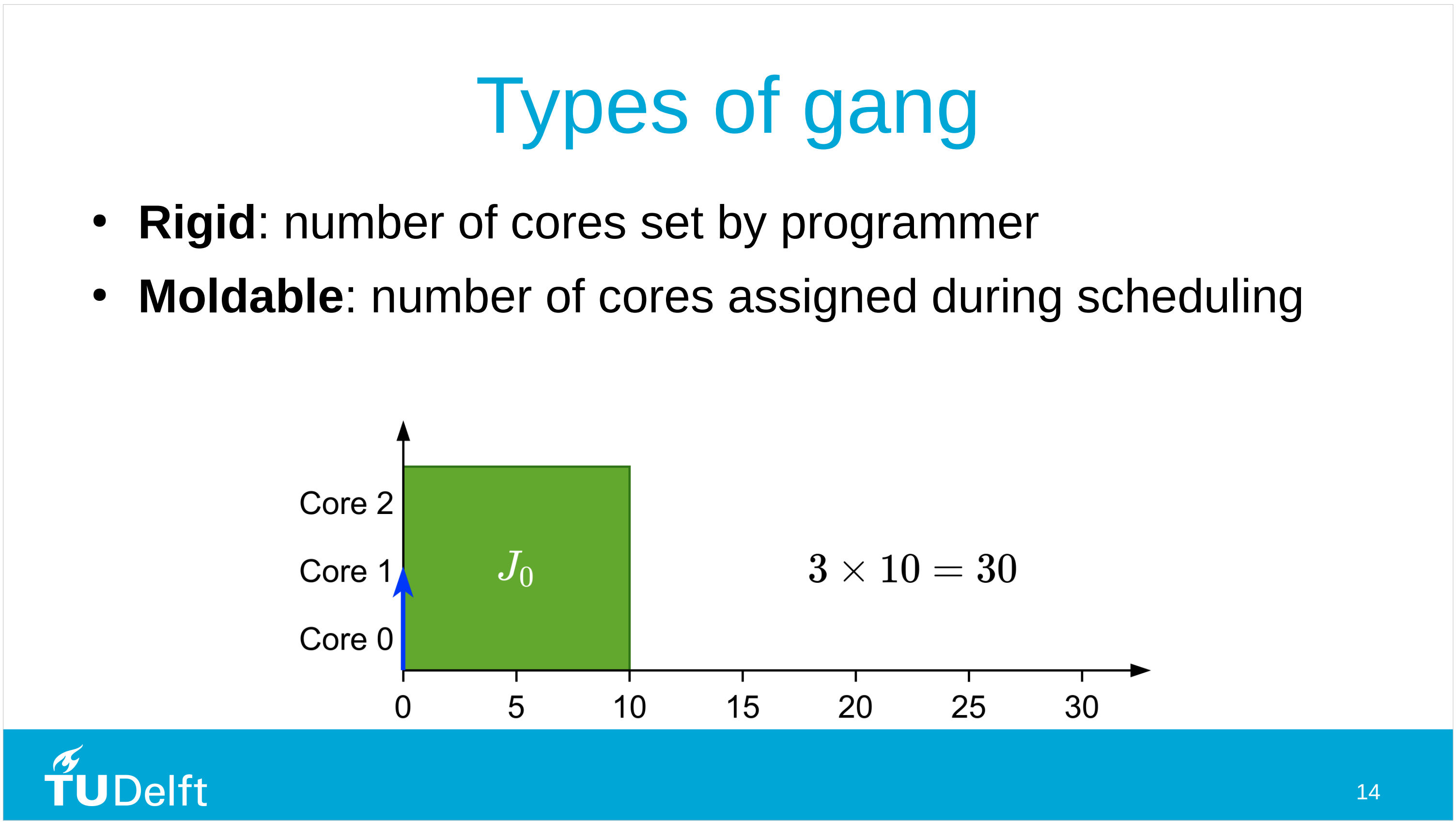


Types of gang

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

When talking about gang scheduling, we can have three types:

<click>Rigid gang, where the number of cores is set by the programmer and fixed during execution. In this example this job executes with 3 cores and requires 10 time units to compute

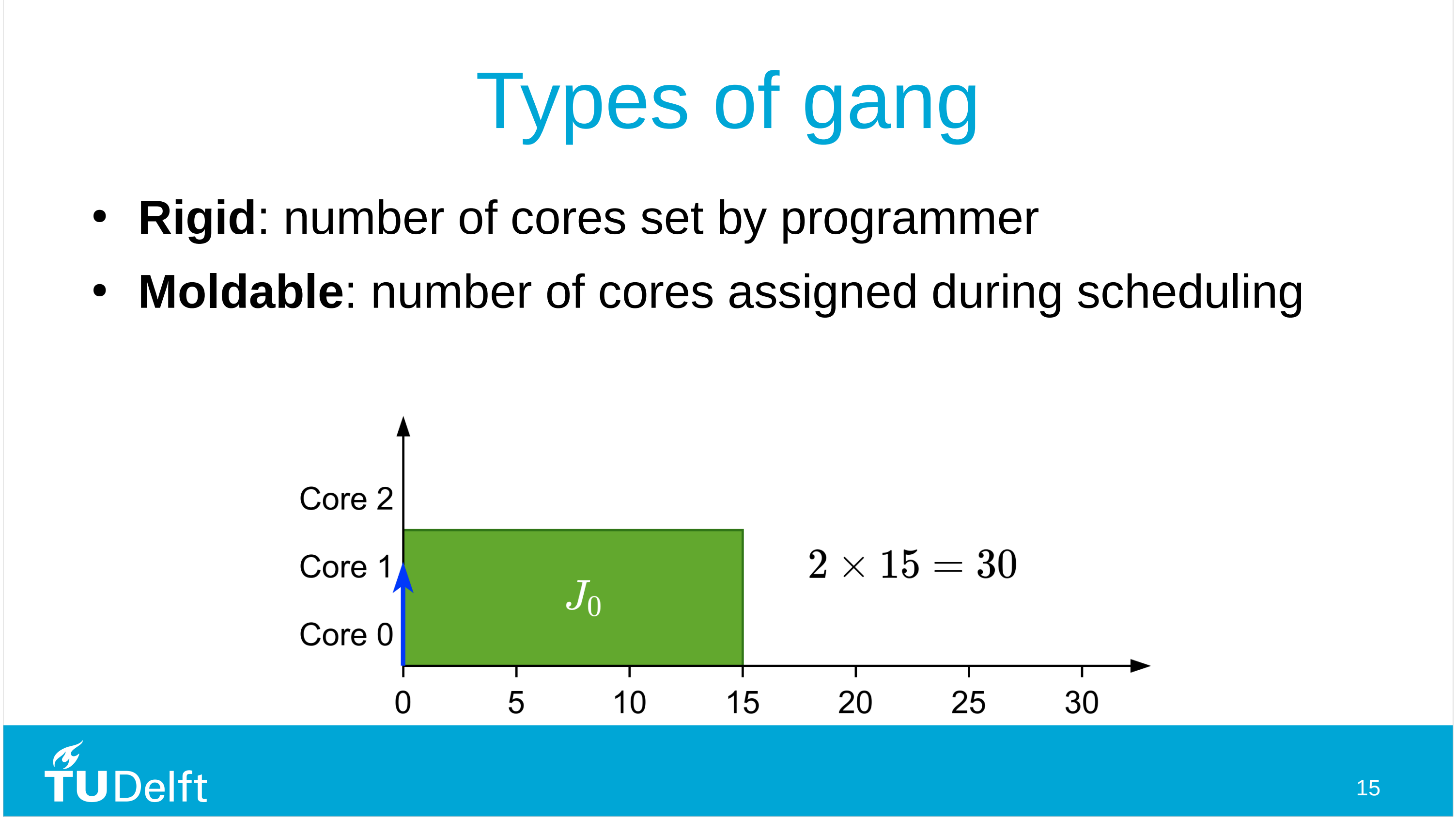


Types of gang

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

Moldable gang, where the number of cores can be in a range of values and the actual chosen value is decided when scheduling the task

In the example now the job can also execute <click> in two cores and <click> in one core. While also increasing the execution time.

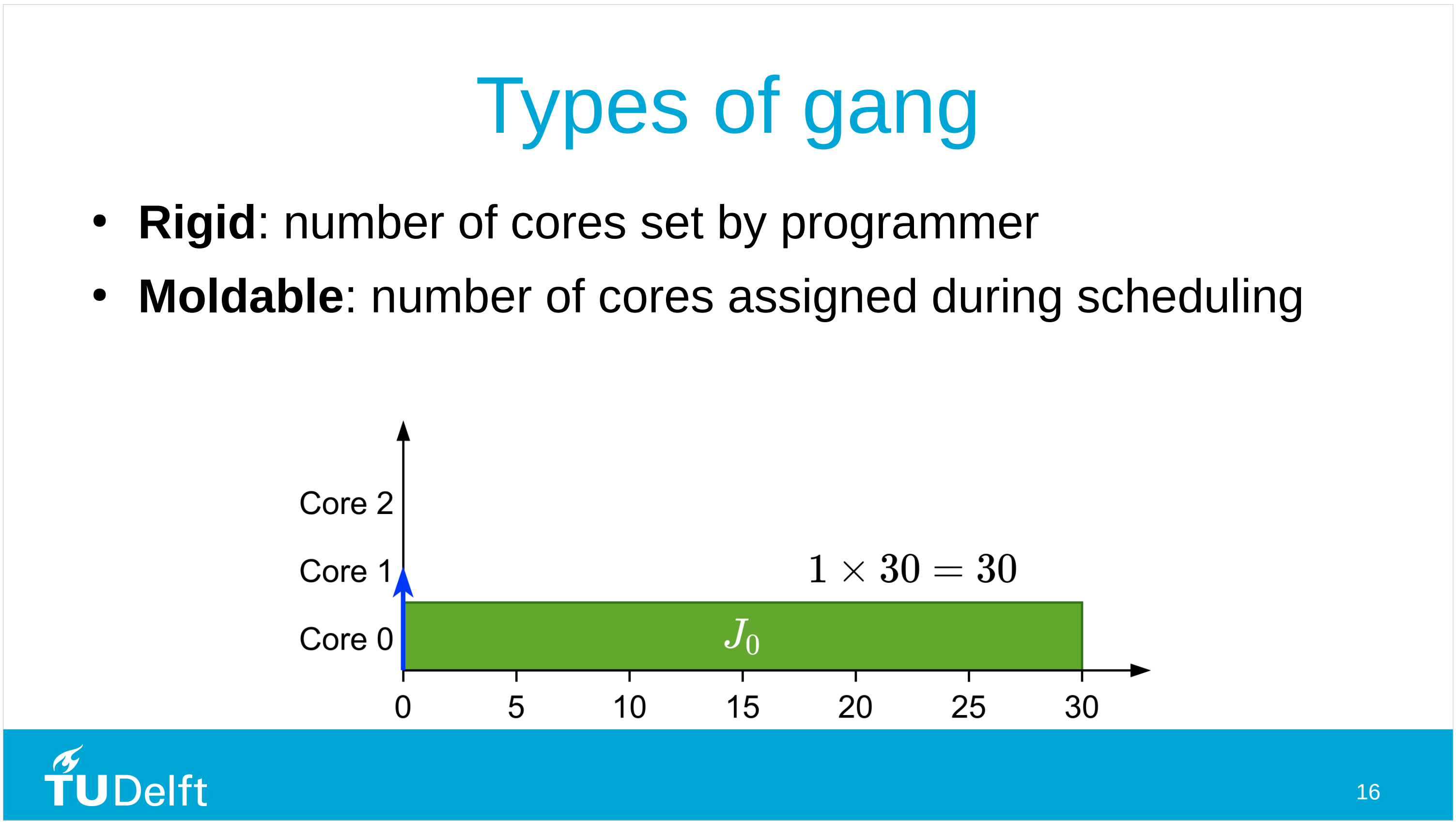


Types of gang

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

Moldable gang, where the number of cores can be in a range of values and the actual chosen value is decided when scheduling the task

In the example now the job can also execute <click> in two cores and <click> in one core. While also increasing the execution time.

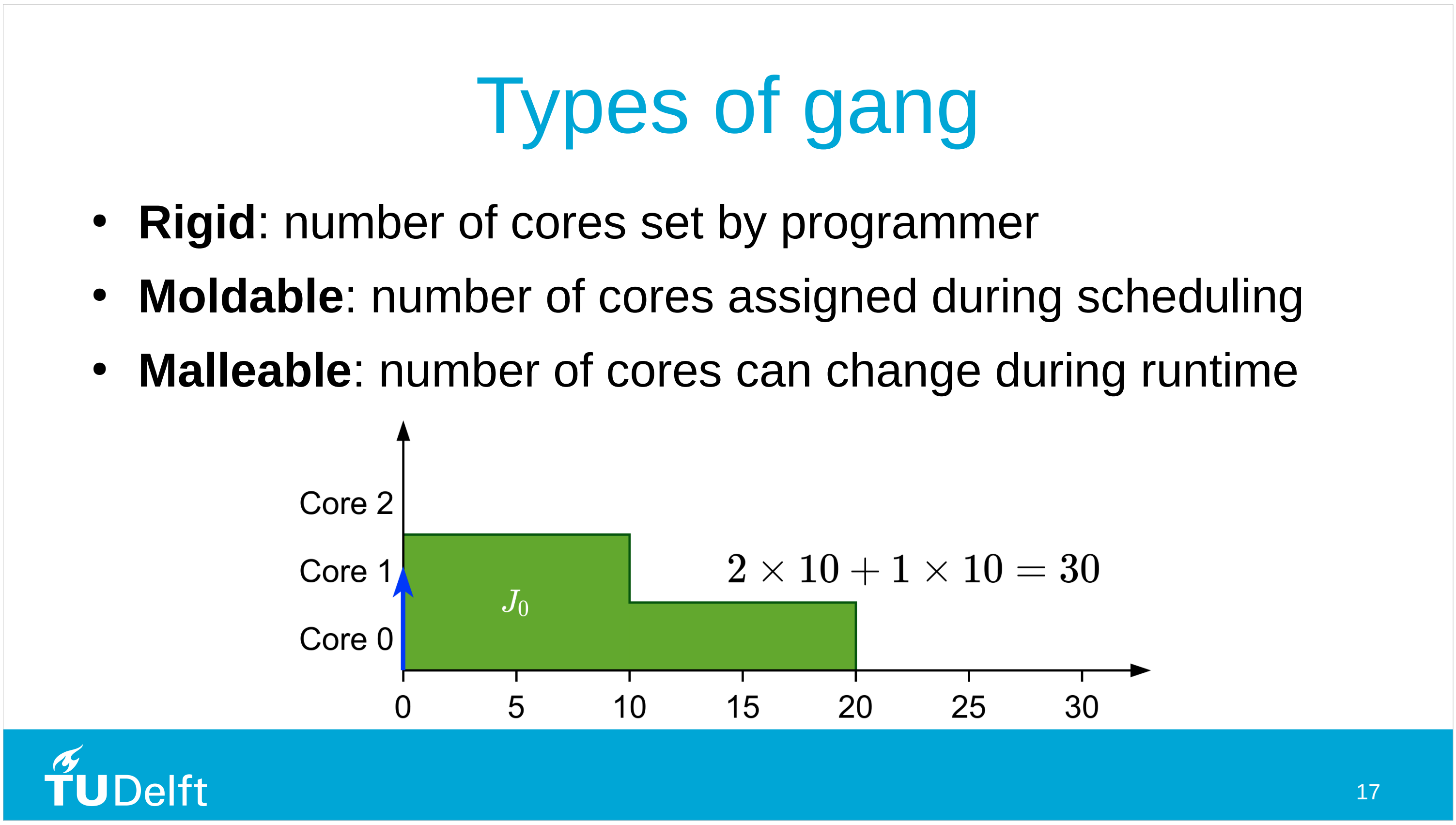


Types of gang

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

Moldable gang, where the number of cores can be in a range of values and the actual chosen value is decided when scheduling the task

In the example now the job can also execute <click> in two cores and <click> in one core. While also increasing the execution time.

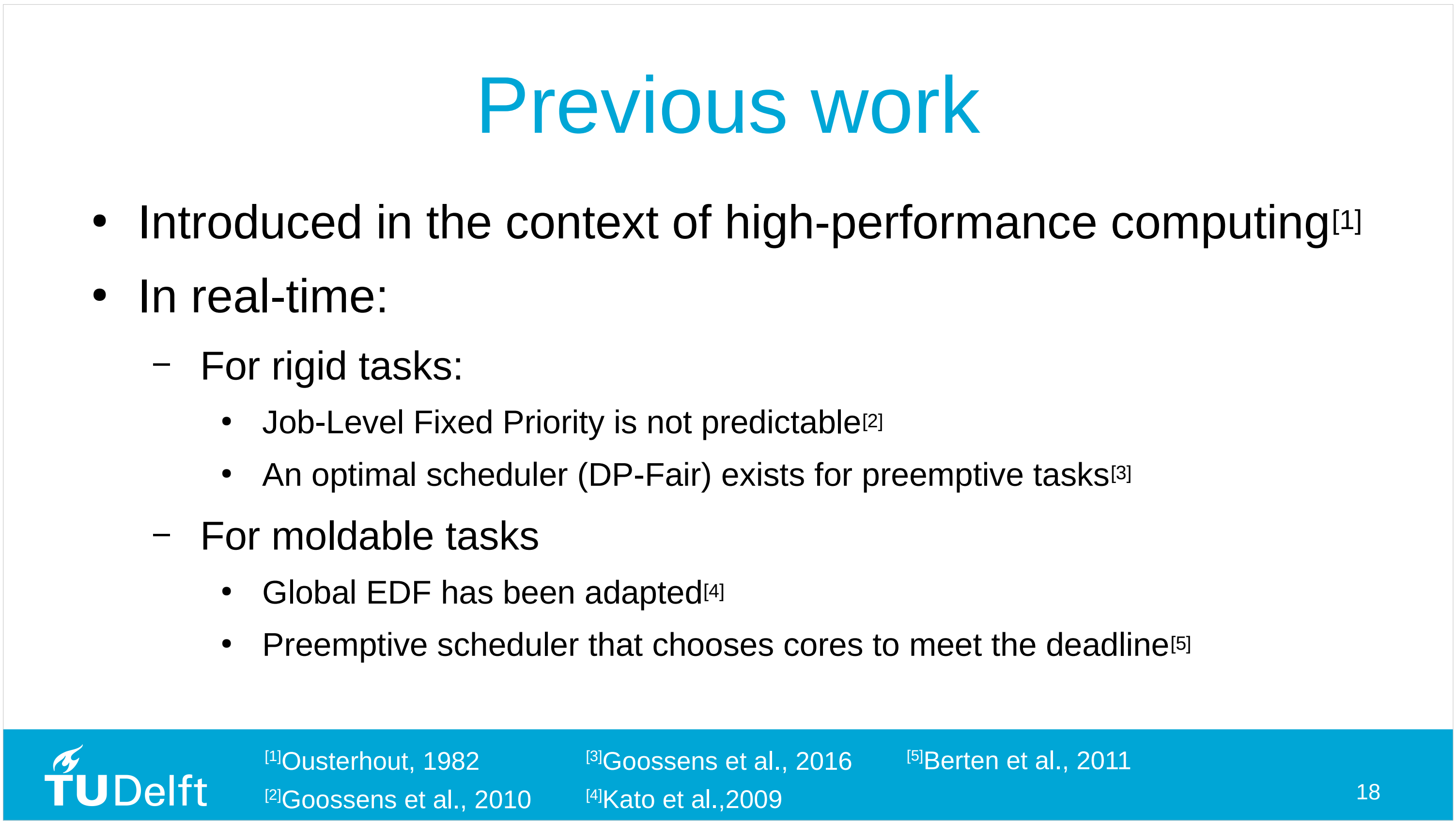


Types of gang

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

Finally there is malleable gang, where the number of cores can vary during the execution of the job and in the example the job starts executing with 2 cores and then at time 10 it switches to one core.

This work focuses on moldable gang



Previous work

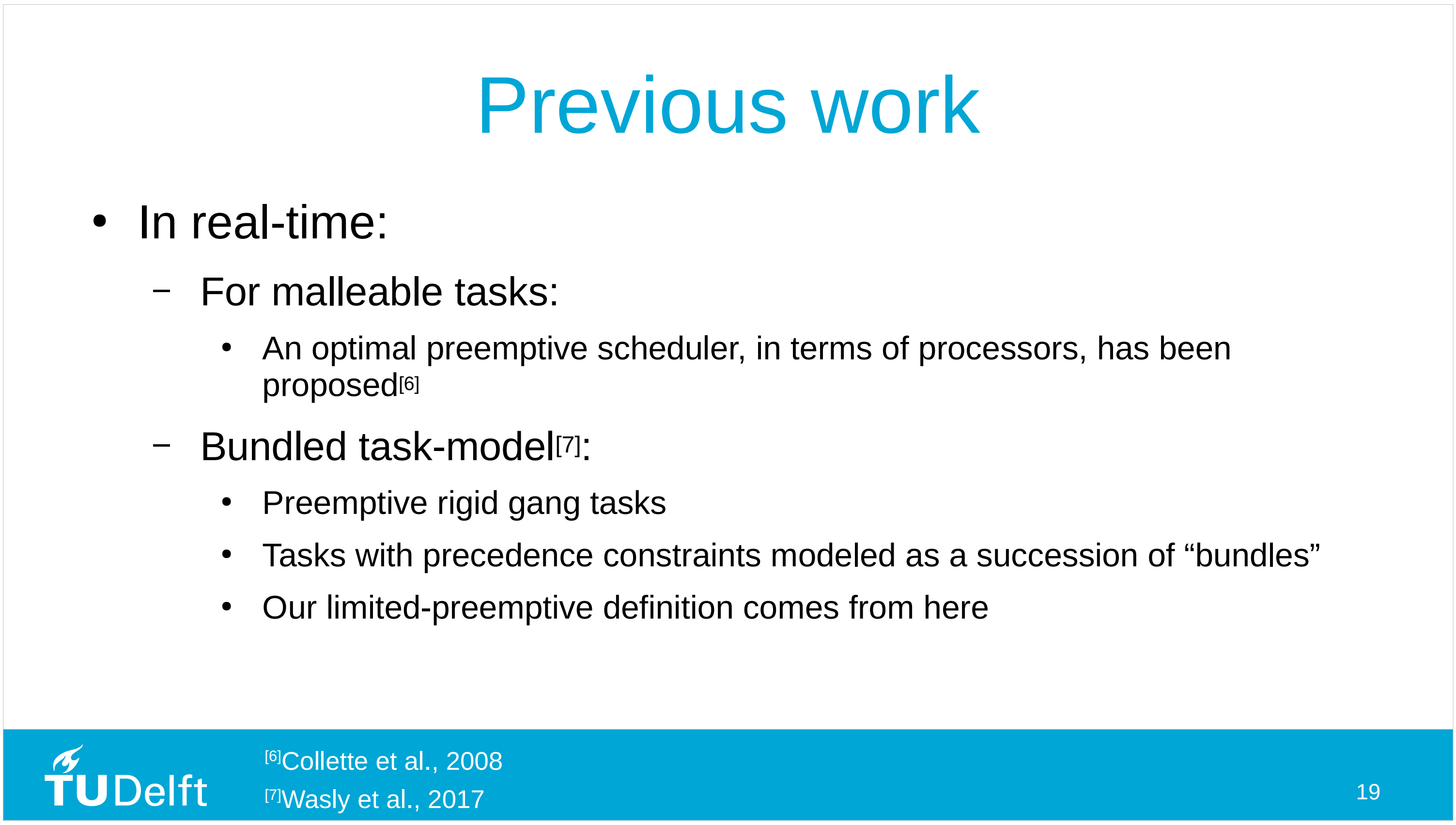
- Introduced in the context of high-performance computing[1]
- In real-time:
  - For rigid tasks:
    - Job-Level Fixed Priority is not predictable[2]
    - An optimal scheduler (DP-Fair) exists for preemptive tasks[3]
  - For moldable tasks
    - Global EDF has been adapted[4]
    - Preemptive scheduler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

Alright, so what has been done previously in gang scheduling? <click>First, the gang task model was introduced in high-performance computing as a way to reduce synchronization overhead and optimize access to shared data

<click>From a real-time viewpoint the results are rather limited.

- <click>For rigid gang taks it has been proven that the Job-Level Fixed-Priority scheduler is not predictable and an optimal scheduler was proposed, although it requires a high number of preemptions.
- <click>For moldable gang tasks, the global EDF scheduler has been adapted and studied. Additionally a preemptive scheduler was proposed that tries to be smart and choose the appropiate cores in order to meet deadlines.



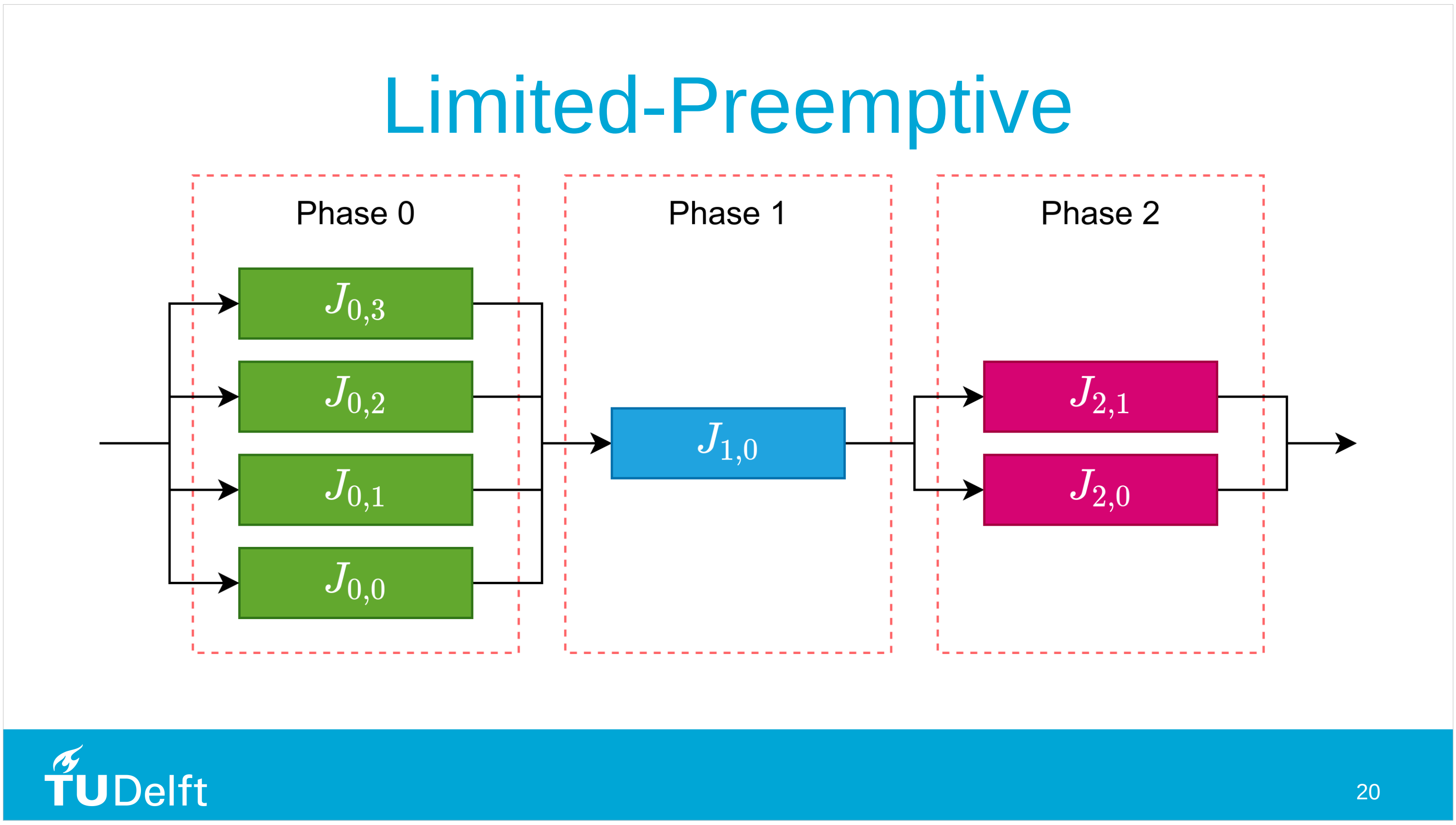
Previous work

- 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

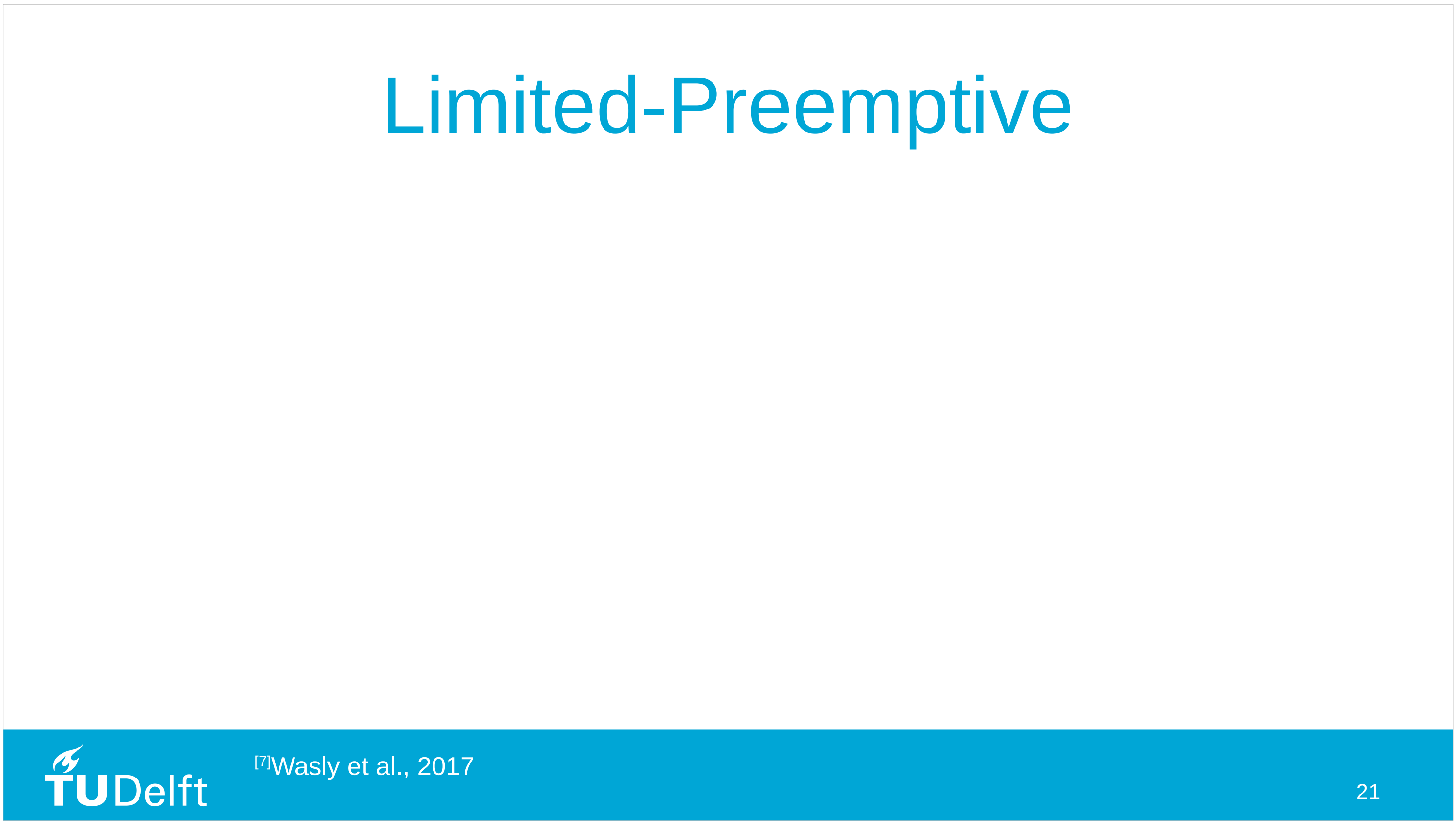
[7]Wasly et al., 2017

- For malleable taks an optimal, in terms of processors, preemptive scheduler has been proposed. It was one of the first works of gang scheduling with real-time analysis
- Additionally there is the bundled task model that works with preemptive rigid gang tasks and models the tasks with precedence constraints as a succession of “bundles”. The limited-preemptive definition that we use for our work comes from this definition.



Limited-Preemptive

Let’s see an example to understand the limited-preemptive definition. Here we have a job that is divided in three phases with dependencies between them.



Limited-Preemptive

[7]Wasly et al., 2017

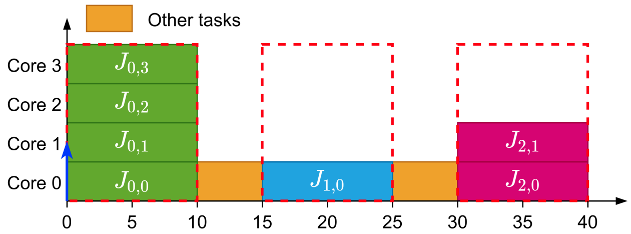
If we try to schedule them with rigid gang we could obtain the following schedule were the gang task is taking cores that does not use.<click>

That’s why we have the bundled task model where each bundle only asks for the cores it requires but it allows preemptions to happen inside the bundle

<click>And here is where our limited-preemptive definition comes in where the preemptions are only allowed between bundles and not inside them.

Limited-Preemptive

- Rigid gang could ask for cores that does not use

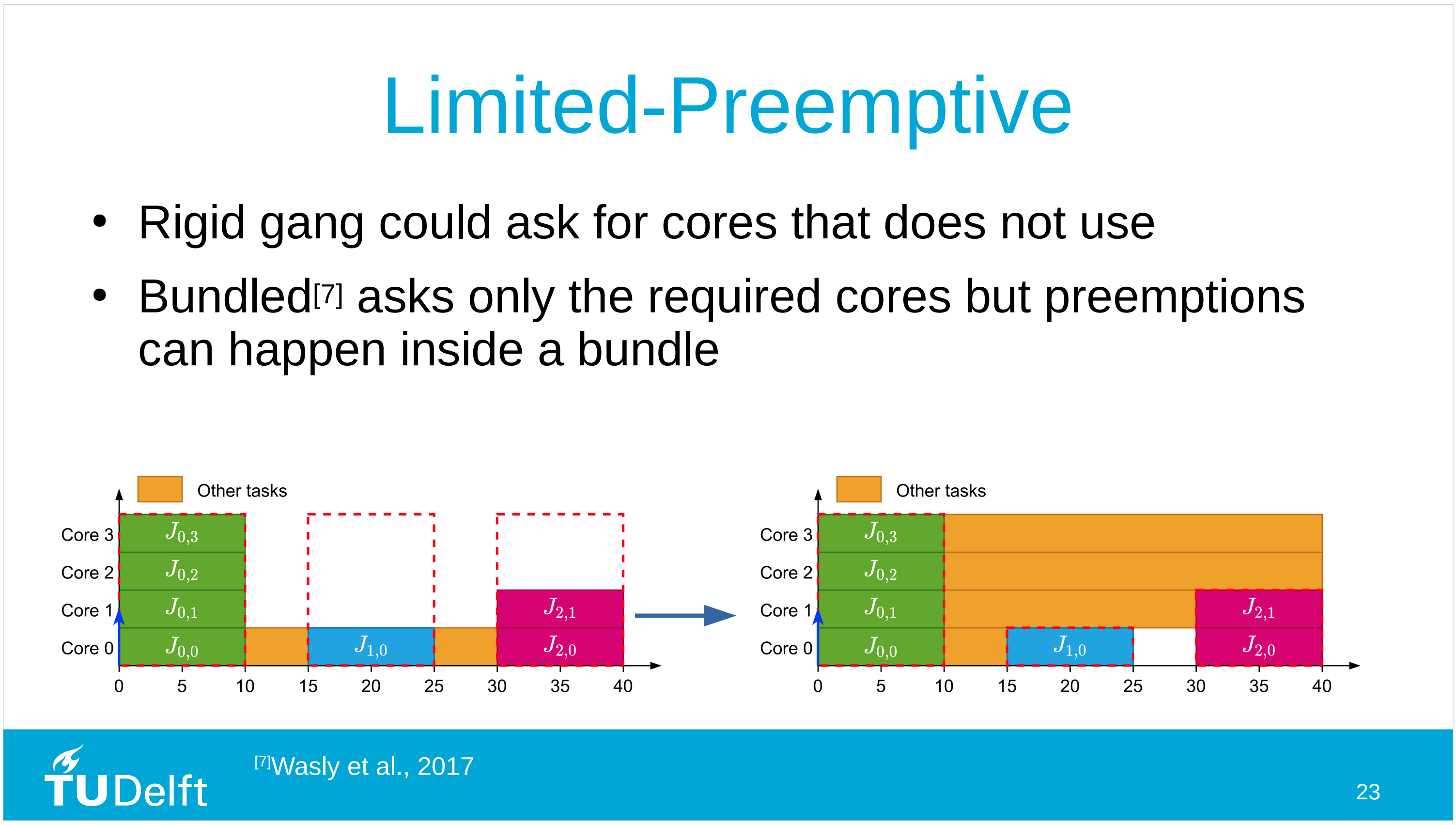


[7]Wasly et al., 2017

If we try to schedule them with rigid gang we could obtain the following schedule were the gang task is taking cores that does not use.<click>

That’s why we have the bundled task model where each bundle only asks for the cores it requires but it allows preemptions to happen inside the bundle

<click>And here is where our limited-preemptive definition comes in where the preemptions are only allowed between bundles and not inside them.



Limited-Preemptive

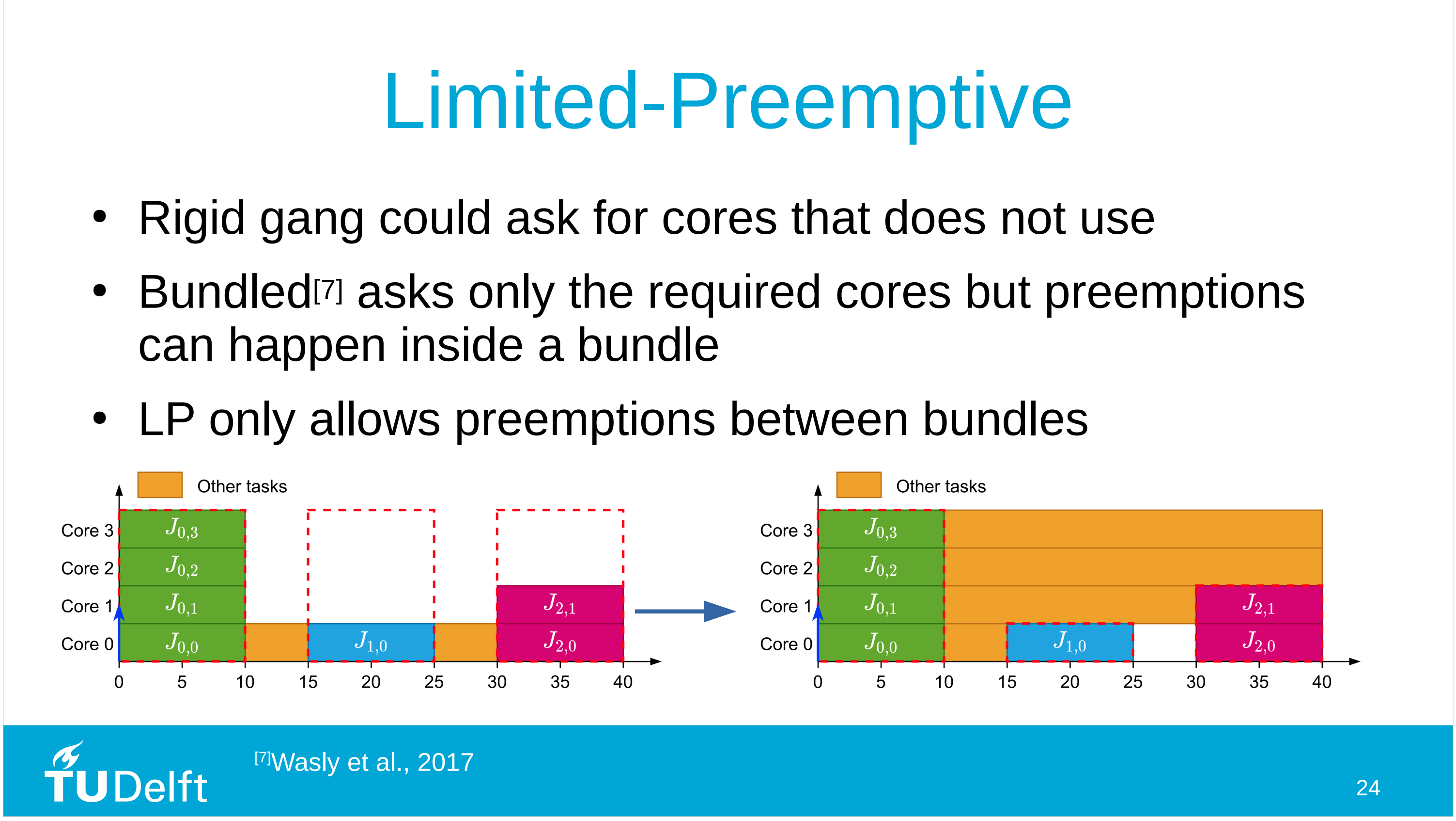
- Rigid gang could ask for cores that does not use
- Bundled[7] asks only the required cores but preemptions can happen inside a bundle

[7]Wasly et al., 2017

If we try to schedule them with rigid gang we could obtain the following schedule were the gang task is taking cores that does not use.<click>

That’s why we have the bundled task model where each bundle only asks for the cores it requires but it allows preemptions to happen inside the bundle

<click>And here is where our limited-preemptive definition comes in where the preemptions are only allowed between bundles and not inside them.



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

If we try to schedule them with rigid gang we could obtain the following schedule were the gang task is taking cores that does not use.<click>

That’s why we have the bundled task model where each bundle only asks for the cores it requires but it allows preemptions to happen inside the bundle

<click>And here is where our limited-preemptive definition comes in where the preemptions are only allowed between bundles and not inside them.

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
  - Scheduler has to decide
    - When to release a job
    - How many cores to assign to this job

LPMRGS