Multiprocessor scheduling, part 2

Scheduling theory

Pontus Ekberg

Uppsala University

2018-10-03

Platform assumptions

Assumptions

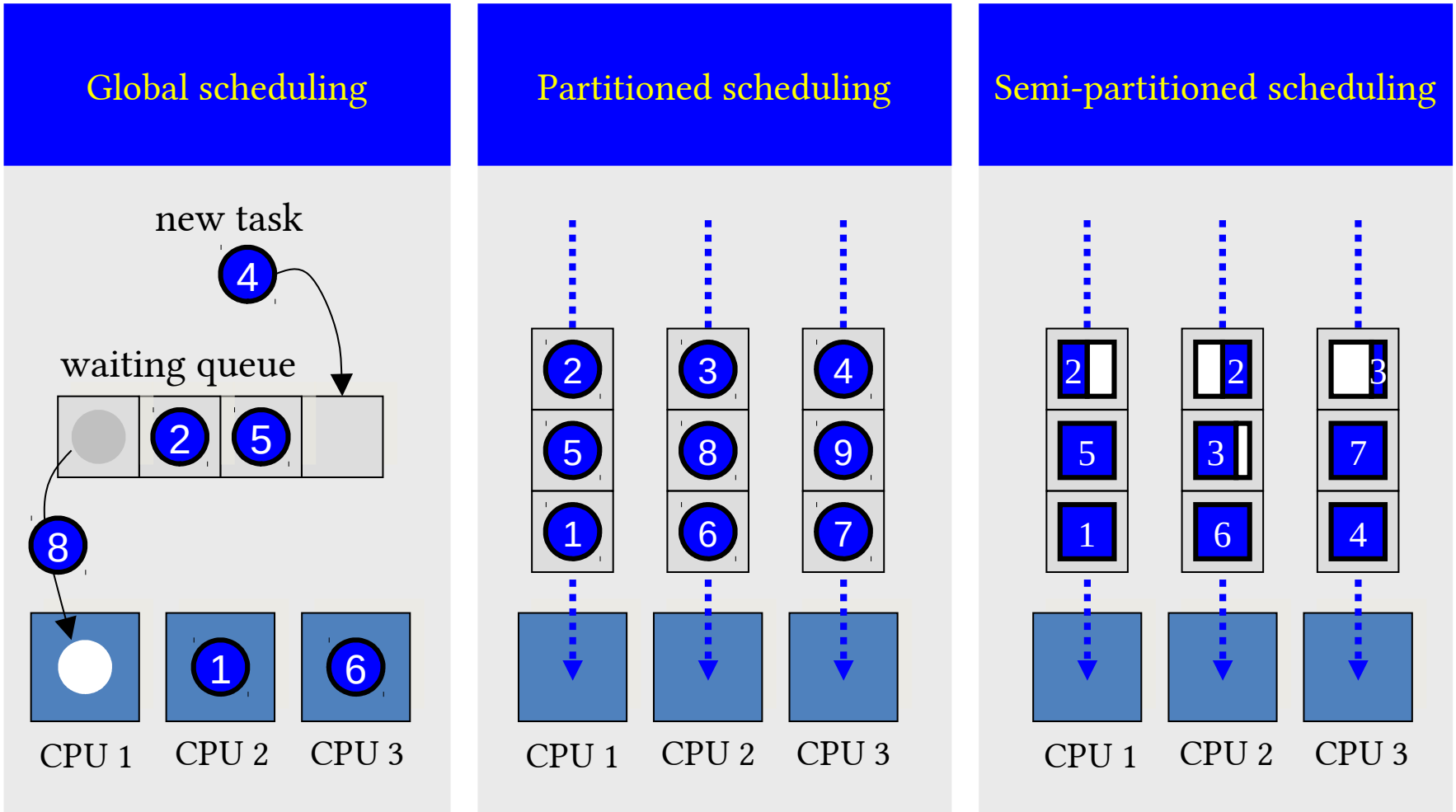
- We have m *independent* processors, where $m > 1$
  - No interference between processors/cores!
  - In principle achievable by carefully selecting hardware and middleware
  - Otherwise it is hopefully a good approximation
- Processors are *identical*
  - Any job can be executed on any processor, with the same WCET
  - Other possible models are *uniform* and *heterogeneous*

Task model assumptions

Assumptions

- Code is *sequential*
  - No parallelism allowed inside the individual jobs
  - *But:* more expressive task models do permit intra-task parallelism
- We will consider ordinary *sporadic tasks*
  - That are preemptive
  - Don't share mutually exclusive resources
  - *But:* some techniques generalize nicely to other task models

Main classification of multiprocessor schedulers



Partitioned scheduling

Partitioned scheduling

=

Partitioning strategy

+

Uniprocessor scheduling

Partitioned scheduling

Partitioned scheduling

=

Partitioning strategy

+

Uniprocessor scheduling

Well understood!

Partitioned scheduling

Partitioned scheduling

=

Partitioning strategy — ?

+

Uniprocessor scheduling

Well understood!

Partitioning

Example: Implicit deadline sporadic tasks with EDF on 4 cores

Partitioning

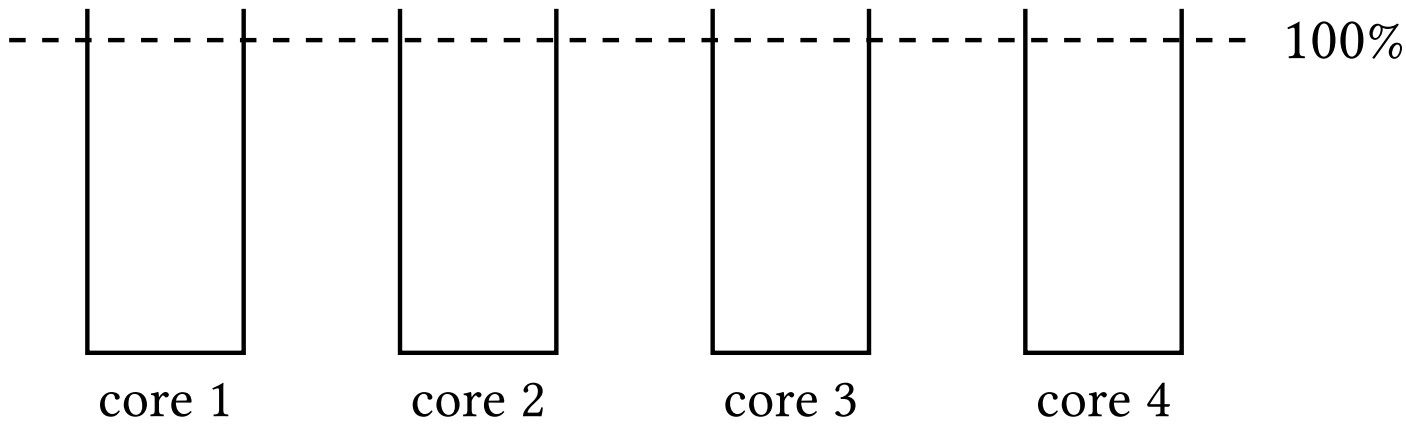
Example: Implicit deadline sporadic tasks with EDF on 4 cores

Recall: They are EDF-schedulable on a uniprocessor iff $\sum_i \mathrm{U}(\tau_i) \leqslant 1$

Partitioning

Example: Implicit deadline sporadic tasks with EDF on 4 cores

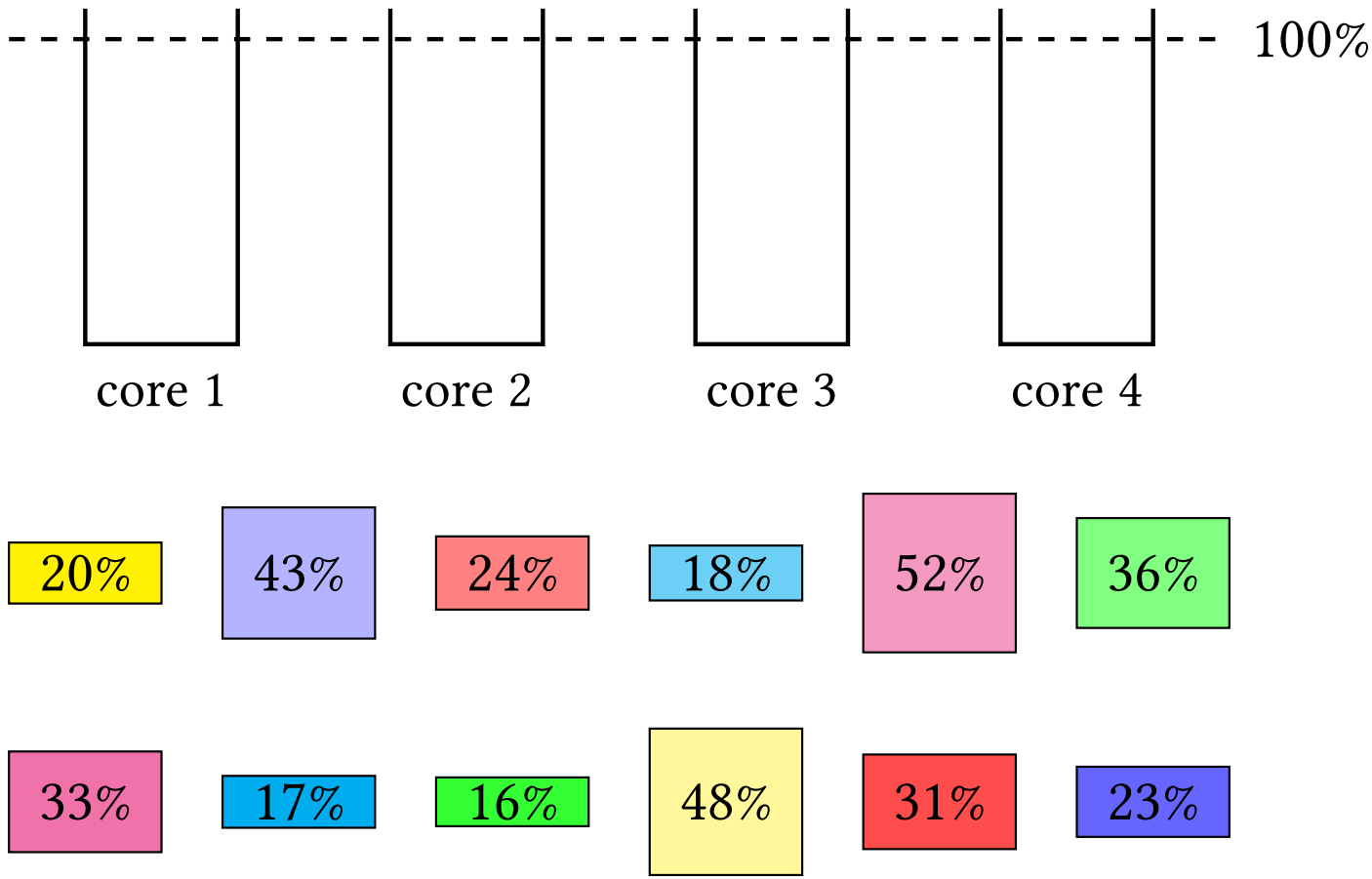
Recall: They are EDF-schedulable on a uniprocessor iff $\sum_i \mathrm{U}(\tau_i) \leqslant 1$



Partitioning

Example: Implicit deadline sporadic tasks with EDF on 4 cores

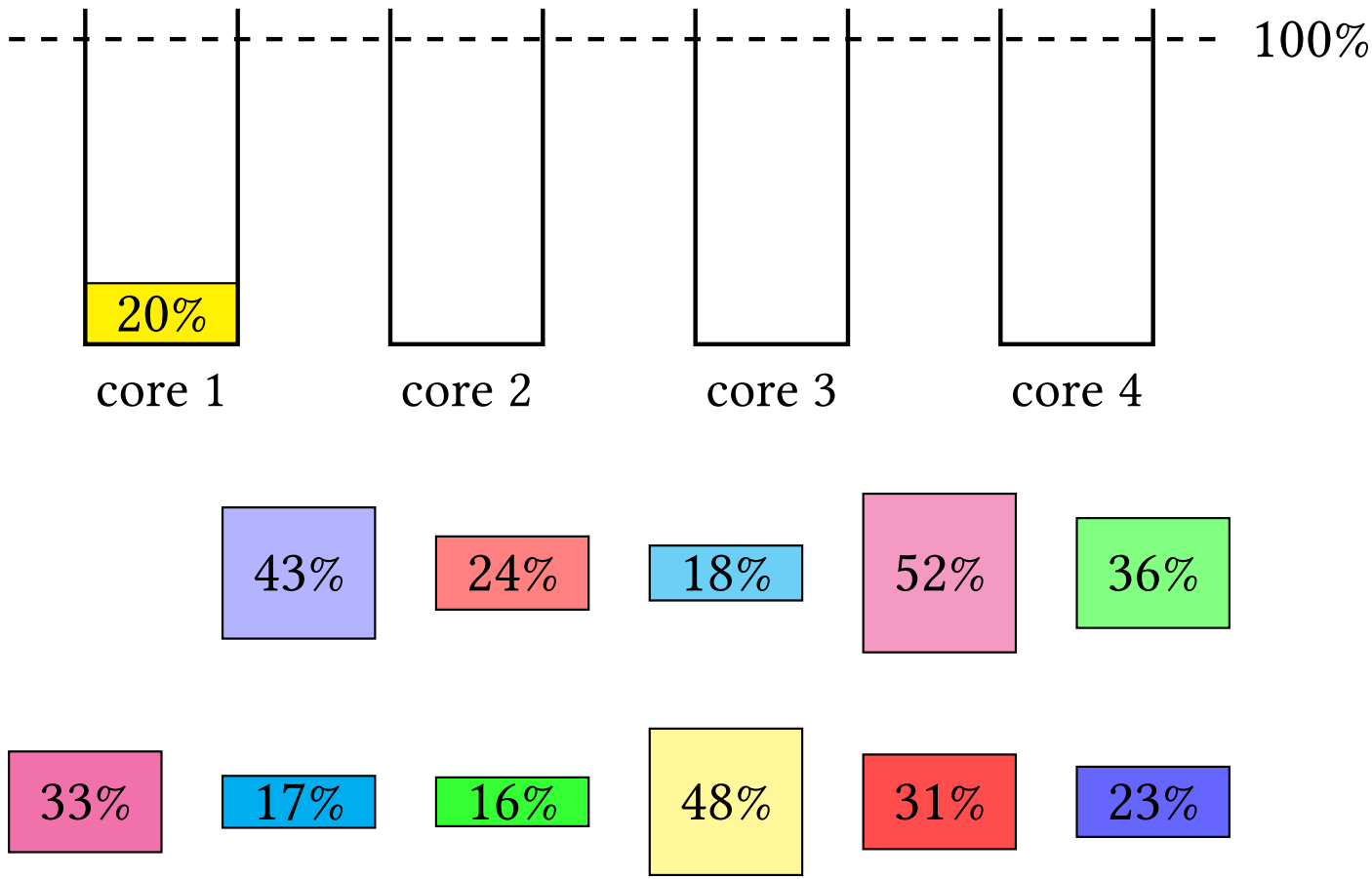
Recall: They are EDF-schedulable on a uniprocessor iff $\sum_i \mathrm{U}(\tau_i) \leqslant 1$



Partitioning

Example: Implicit deadline sporadic tasks with EDF on 4 cores

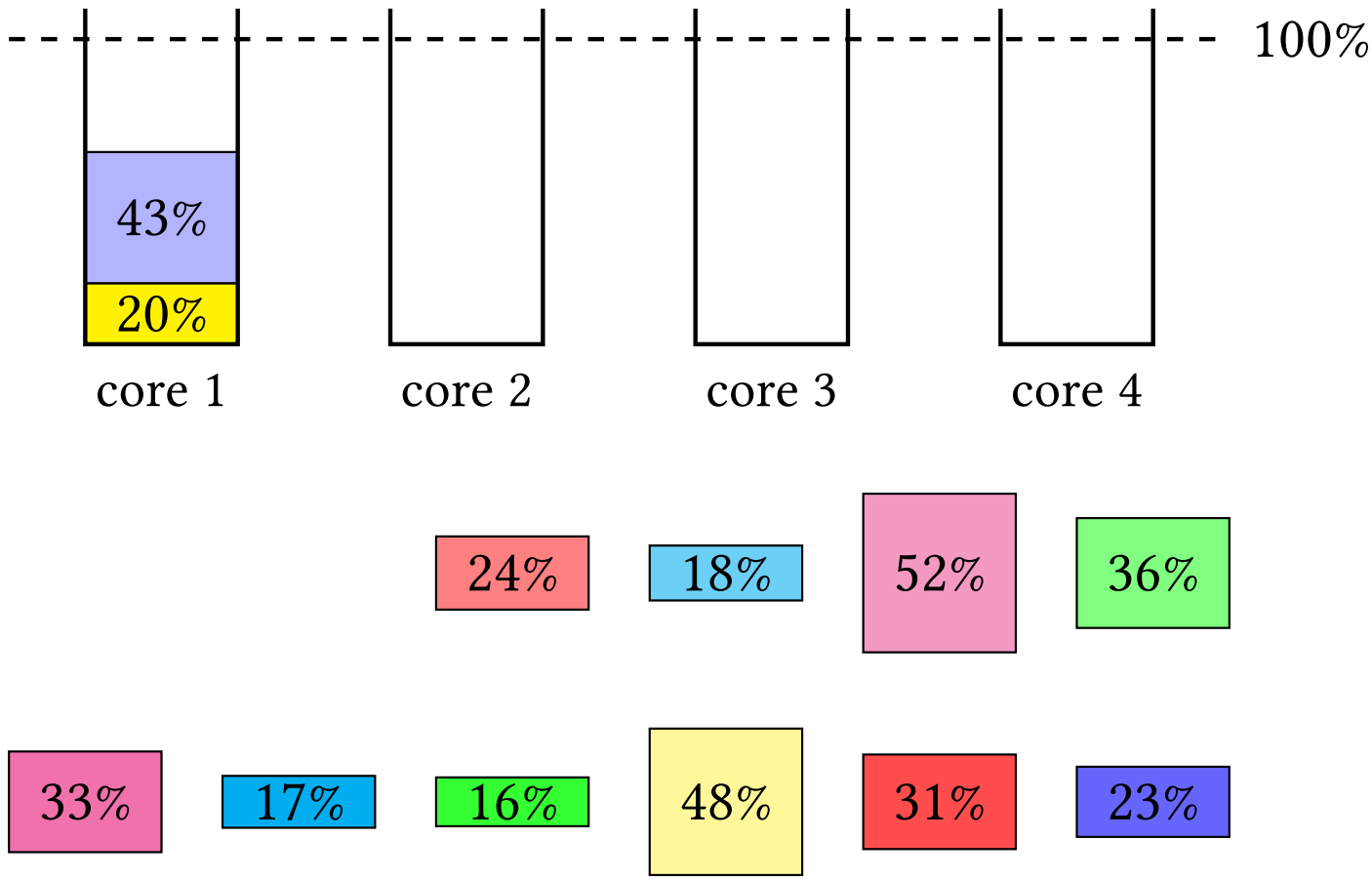
Recall: They are EDF-schedulable on a uniprocessor iff $\sum_i \mathrm{U}(\tau_i) \leqslant 1$



Partitioning

Example: Implicit deadline sporadic tasks with EDF on 4 cores

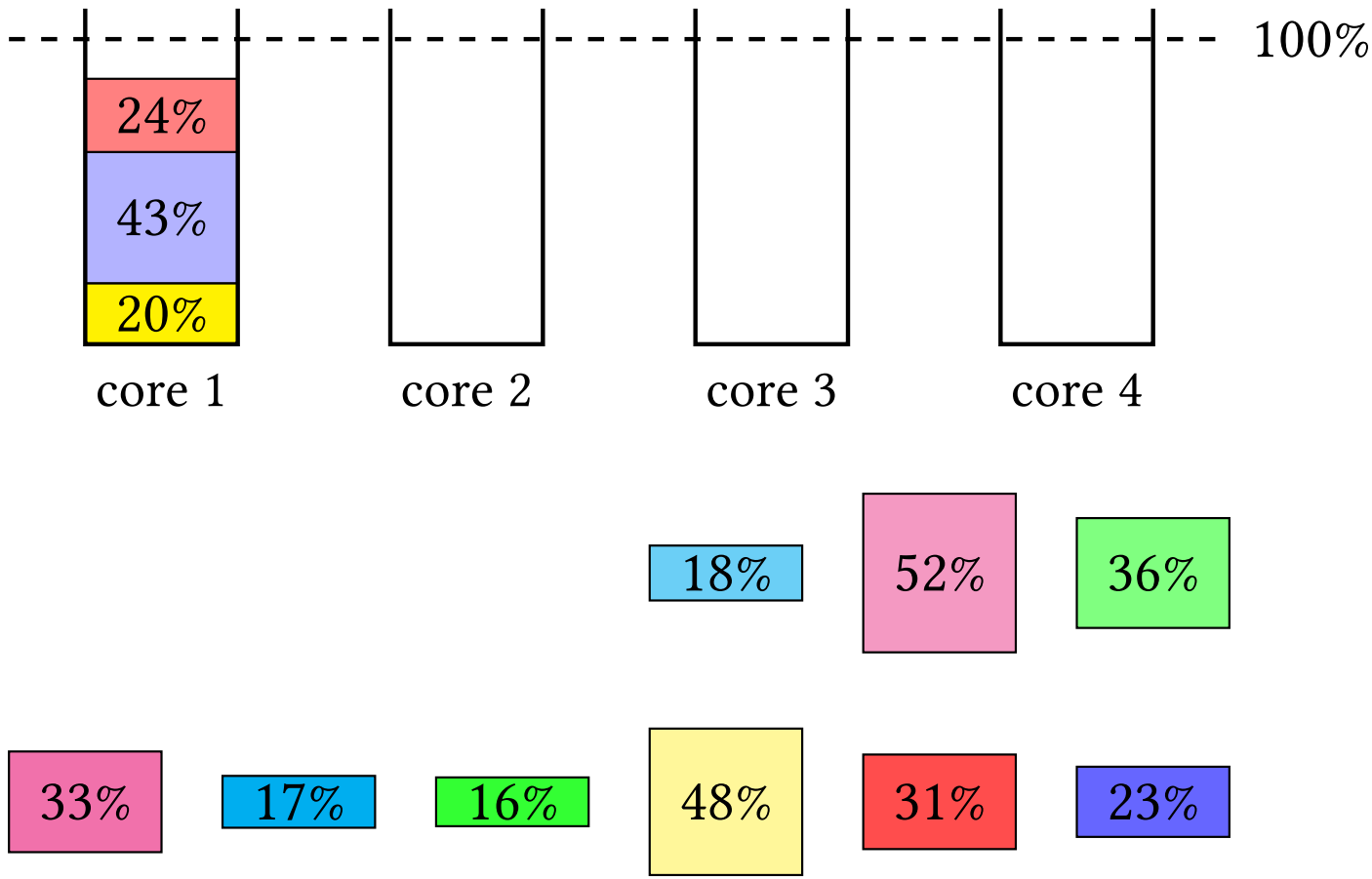
Recall: They are EDF-schedulable on a uniprocessor iff $\sum_i \mathrm{U}(\tau_i) \leqslant 1$



Partitioning

Example: Implicit deadline sporadic tasks with EDF on 4 cores

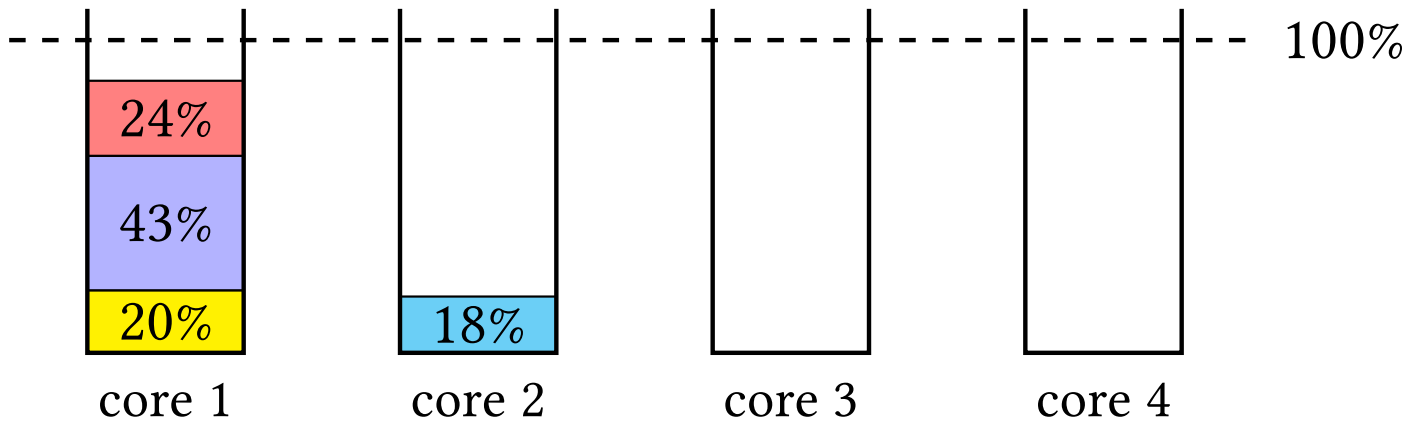
Recall: They are EDF-schedulable on a uniprocessor iff $\sum_i \mathrm{U}(\tau_i) \leqslant 1$

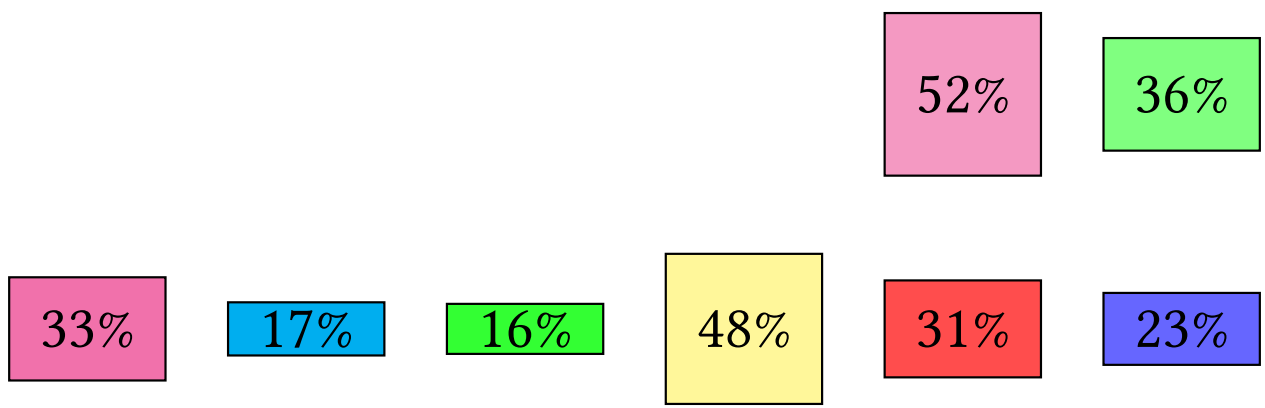


Partitioning

Example: Implicit deadline sporadic tasks with EDF on 4 cores

Recall: They are EDF-schedulable on a uniprocessor iff $\sum_i \mathrm{U}(\tau_i) \leqslant 1$

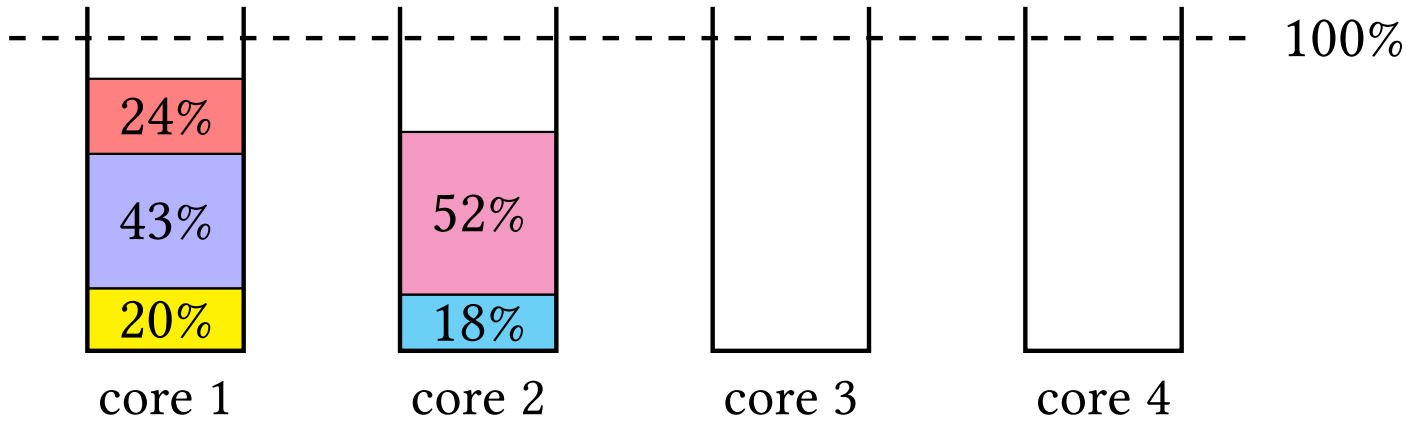


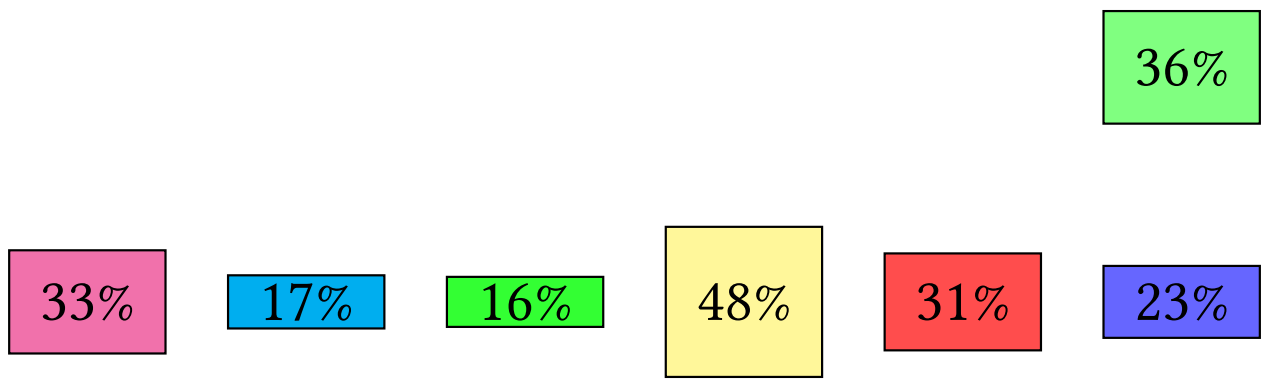


Partitioning

Example: Implicit deadline sporadic tasks with EDF on 4 cores

Recall: They are EDF-schedulable on a uniprocessor iff $\sum_i \mathrm{U}(\tau_i) \leqslant 1$

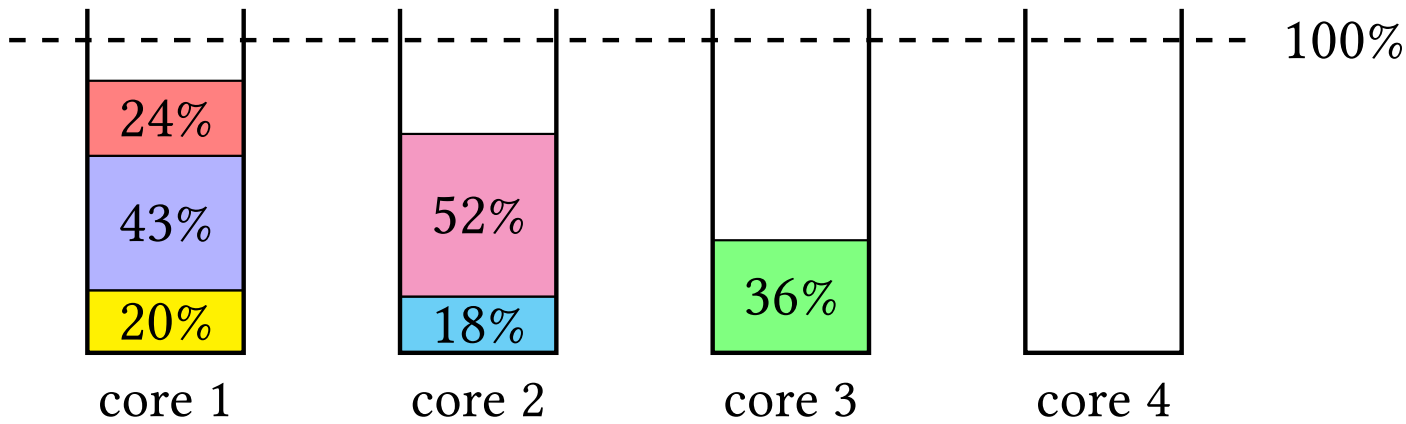


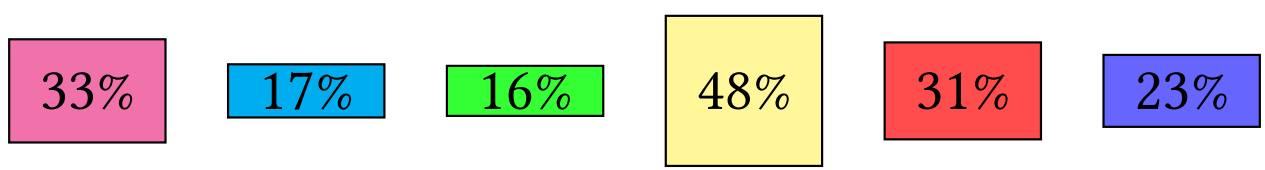


Partitioning

Example: Implicit deadline sporadic tasks with EDF on 4 cores

Recall: They are EDF-schedulable on a uniprocessor iff $\sum_i \mathrm{U}(\tau_i) \leqslant 1$

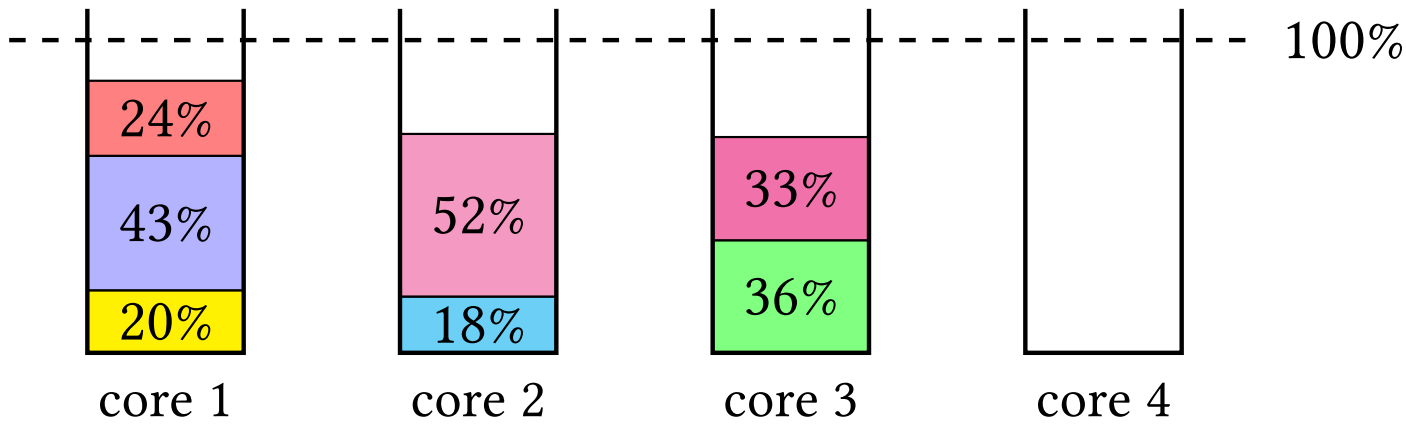


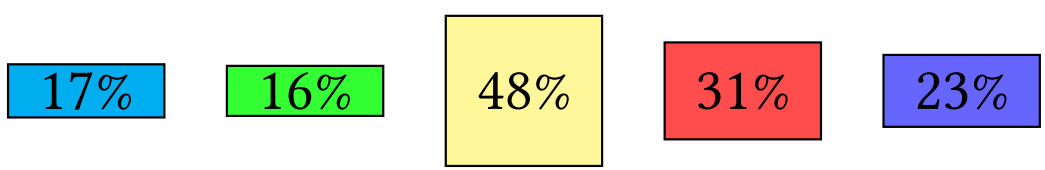


Partitioning

Example: Implicit deadline sporadic tasks with EDF on 4 cores

Recall: They are EDF-schedulable on a uniprocessor iff $\sum_i \mathrm{U}(\tau_i) \leqslant 1$

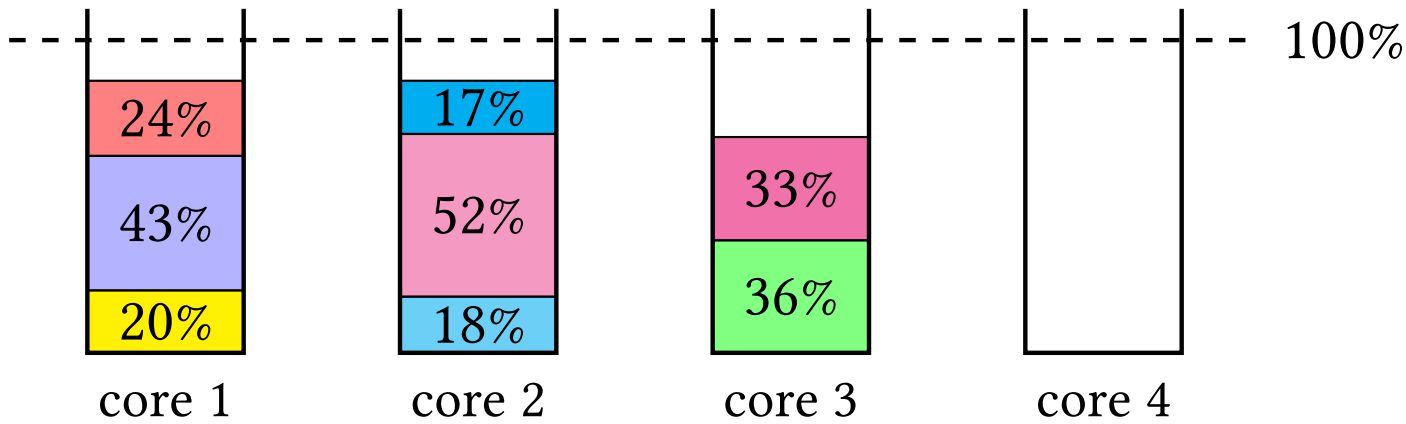


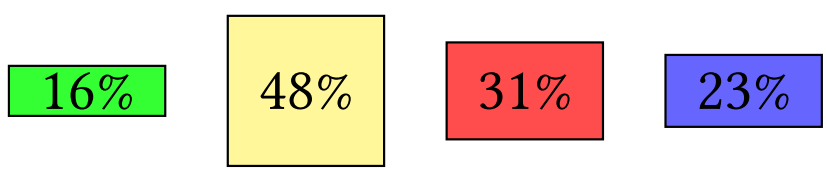


Partitioning

Example: Implicit deadline sporadic tasks with EDF on 4 cores

Recall: They are EDF-schedulable on a uniprocessor iff $\sum_i \mathrm{U}(\tau_i) \leqslant 1$

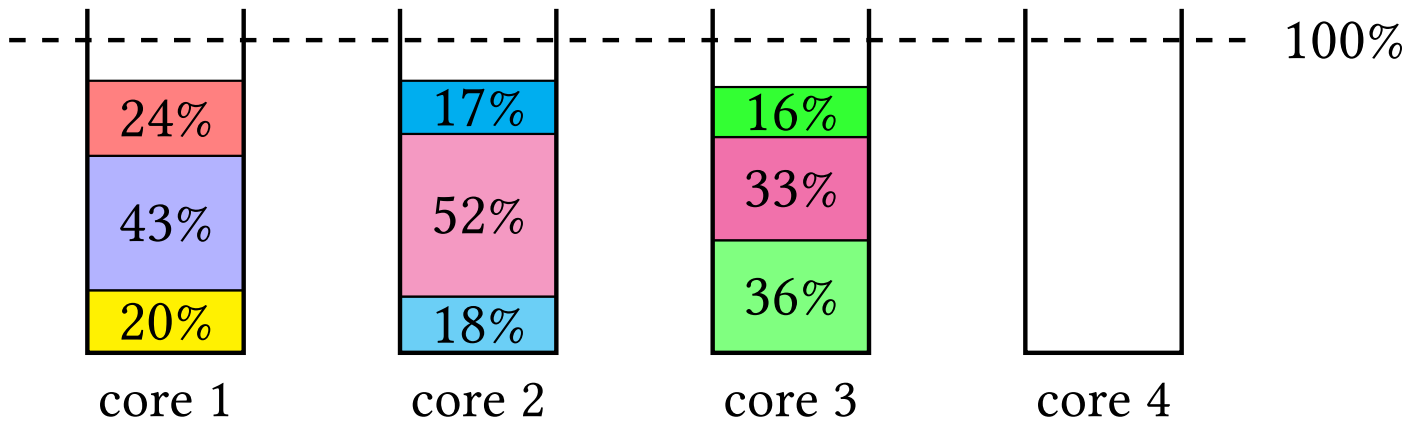




Partitioning

Example: Implicit deadline sporadic tasks with EDF on 4 cores

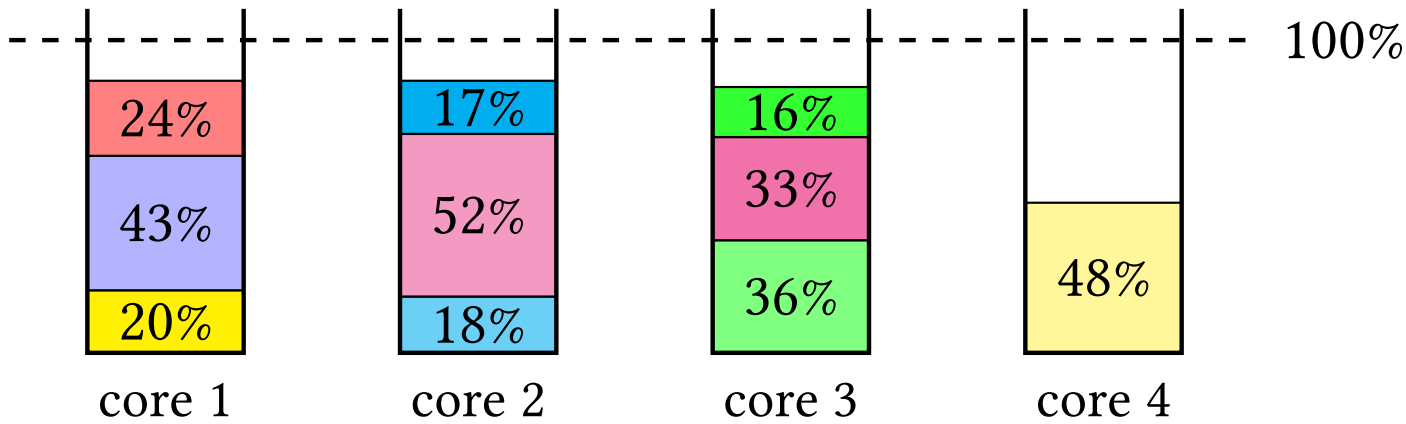
Recall: They are EDF-schedulable on a uniprocessor iff $\sum_i \mathrm{U}(\tau_i) \leqslant 1$

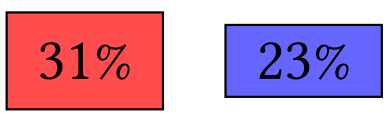


Partitioning

Example: Implicit deadline sporadic tasks with EDF on 4 cores

Recall: They are EDF-schedulable on a uniprocessor iff $\sum_i \mathrm{U}(\tau_i) \leqslant 1$

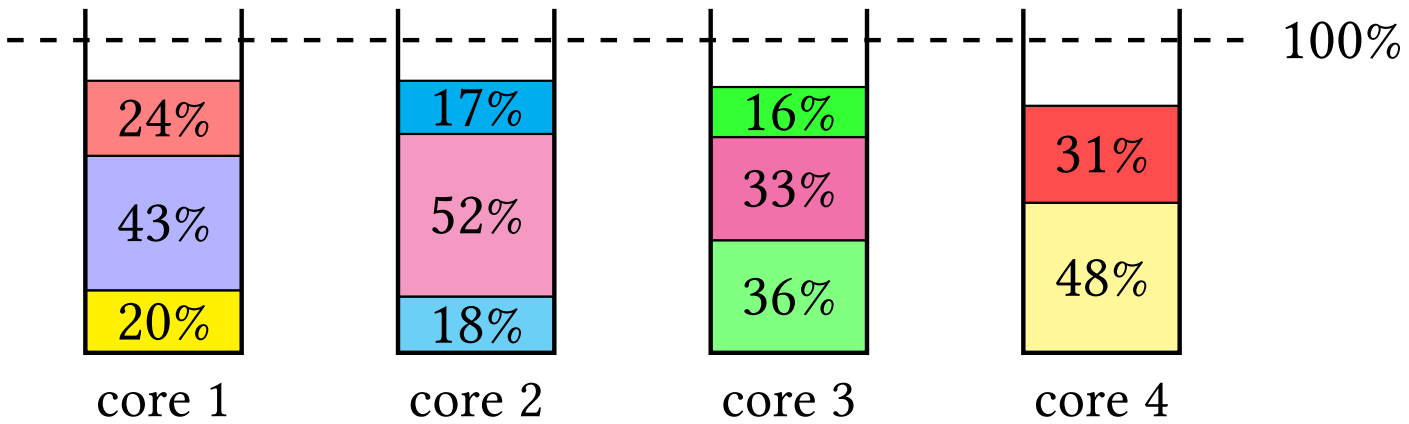




Partitioning

Example: Implicit deadline sporadic tasks with EDF on 4 cores

Recall: They are EDF-schedulable on a uniprocessor iff $\sum_i \mathrm{U}(\tau_i) \leqslant 1$

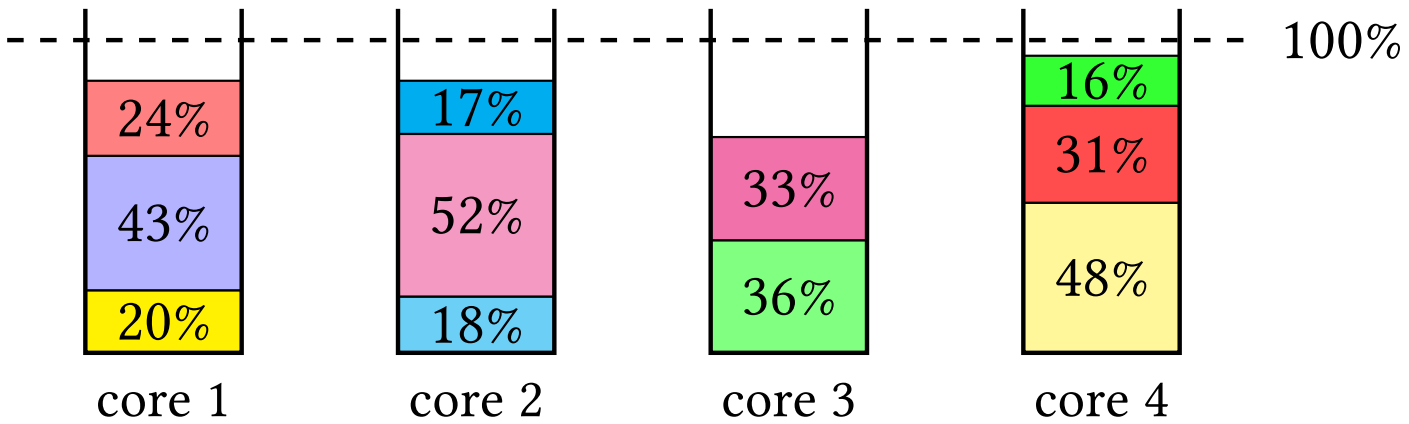


23%

Partitioning

Example: Implicit deadline sporadic tasks with EDF on 4 cores

Recall: They are EDF-schedulable on a uniprocessor iff $\sum_i \mathrm{U}(\tau_i) \leqslant 1$

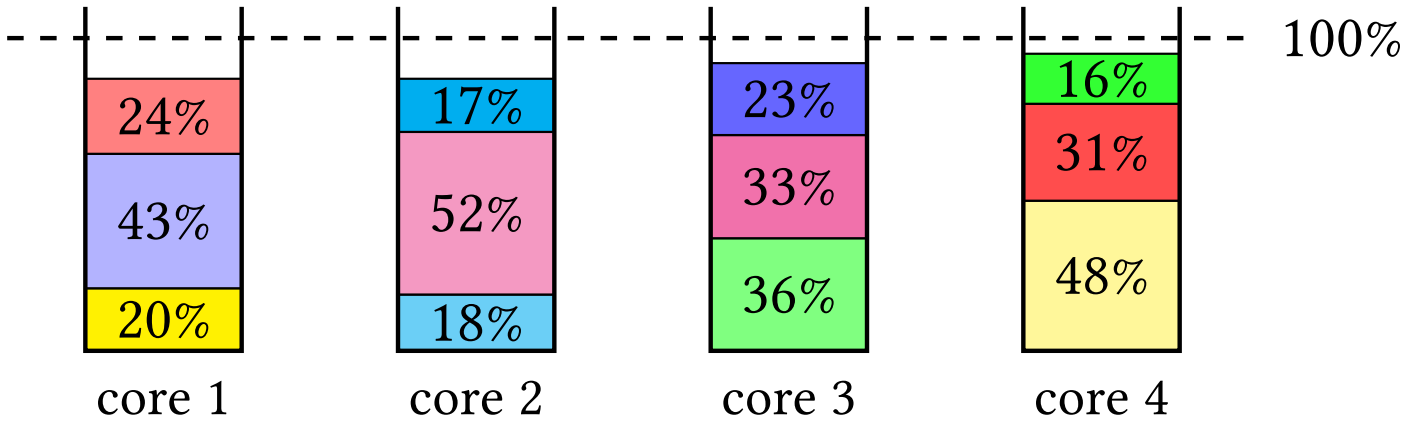


23%

Partitioning

Example: Implicit deadline sporadic tasks with EDF on 4 cores

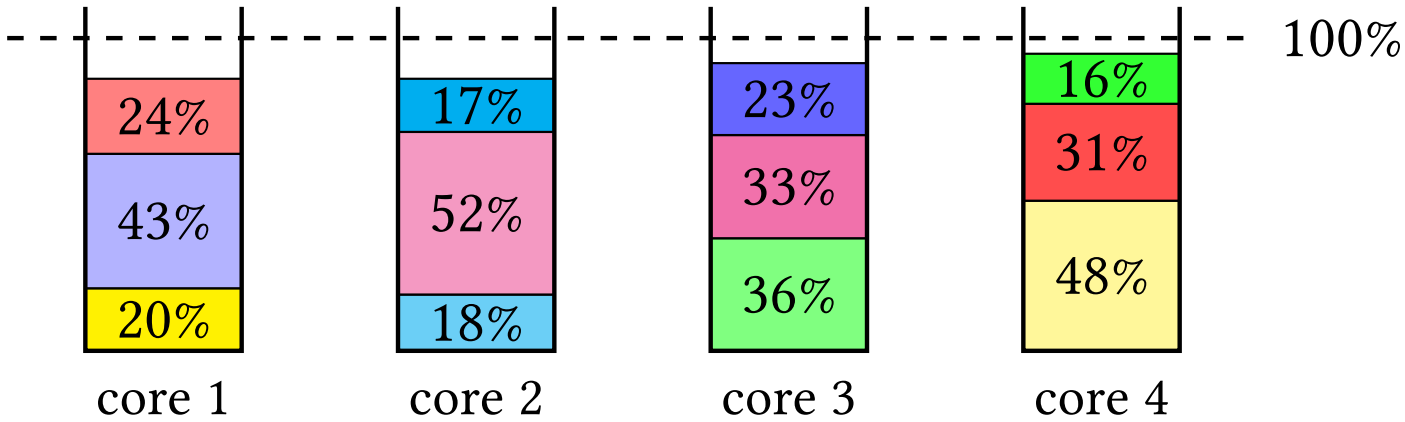
Recall: They are EDF-schedulable on a uniprocessor iff $\sum_i \mathrm{U}(\tau_i) \leqslant 1$



Partitioning

Example: Implicit deadline sporadic tasks with EDF on 4 cores

Recall: They are EDF-schedulable on a uniprocessor iff $\sum_i \mathrm{U}(\tau_i) \leqslant 1$



Bin Packing!

(Famously NP-hard)

How good is partitioned scheduling?

Fact

A set $\mathcal{T}$ of implicit-deadline tasks is schedulable by EDF on 1 preemptive processor iff $U(\mathcal{T}) \leqslant 1$.

Question

Is a set $\mathcal{T}$ of implicit-deadline tasks schedulable by partitioned EDF on m preemptive processors iff $U(\mathcal{T}) \leqslant m$?

How good is partitioned scheduling?

Fact

A set $\mathcal{T}$ of implicit-deadline tasks is schedulable by EDF on 1 preemptive processor iff $\mathrm{U}(\mathcal{T}) \leqslant 1$.

Question

Is a set $\mathcal{T}$ of implicit-deadline tasks schedulable by partitioned EDF on m preemptive processors iff $\mathrm{U}(\mathcal{T}) \leqslant m$?

No!

A utilization bound

Theorem (López et al., 2000)

A set $\mathcal{T}$ of implicit-deadline sporadic tasks is schedulable by partitioned EDF on m preemptive processors if

$$\mathrm{U}(\mathcal{T}) \leqslant \frac{m+1}{2}.$$

A utilization bound

Theorem (López et al., 2000)

A set $\mathcal{T}$ of implicit-deadline sporadic tasks is schedulable by partitioned EDF on m preemptive processors if

$$U(\mathcal{T}) \leqslant \frac{m+1}{2}.$$

- This is only a sufficient condition if $m > 1$. (Why?)

A utilization bound

Theorem (López et al., 2000)

A set $\mathcal{T}$ of implicit-deadline sporadic tasks is schedulable by partitioned EDF on m preemptive processors if

$$U(\mathcal{T}) \leqslant \frac{m+1}{2}.$$

- This is only a sufficient condition if $m > 1$. (Why?)
- There exists no better utilization bound. (Proof on blackboard)

A utilization bound

Theorem (López et al., 2000)

A set $\mathcal{T}$ of implicit-deadline sporadic tasks is schedulable by partitioned EDF on m preemptive processors if

$$U(\mathcal{T}) \leqslant \frac{m+1}{2}.$$

- This is only a sufficient condition if $m > 1$. (Why?)
- There exists no better utilization bound. (Proof on blackboard)
- If the condition holds, we can always find a valid partitioning with a simple heuristic (for example, *First-Fit* as tried on previous slide).

Improvements still possible?

Definition

Let $U_{max}(\mathcal{T})$ be the largest utilization of any task in $\mathcal{T}$, and let

$$\beta = \left\lfloor \frac{1}{U_{max}(\mathcal{T})} \right\rfloor .$$

Theorem (López et al., 2000)

A set $\mathcal{T}$ of implicit-deadline sporadic tasks is schedulable by partitioned EDF on m preemptive processors if

$$U(\mathcal{T}) \leqslant \frac{\beta m + 1}{\beta + 1}.$$

Partitioning in other settings

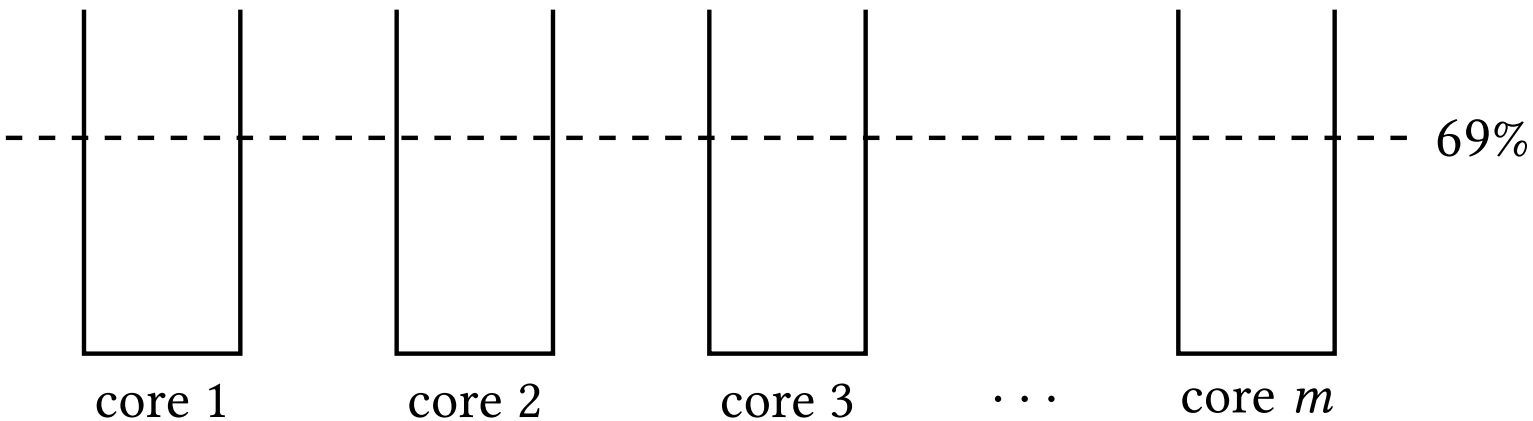
The same general idea applies for other task models and schedulers:

1. Make a partitioning
2. Evaluate the (uniprocessor) schedulability of each partition
3. If this fails for some partition, go back to 1 or give up

Partitioning in other settings

The same general idea applies for other task models and schedulers:

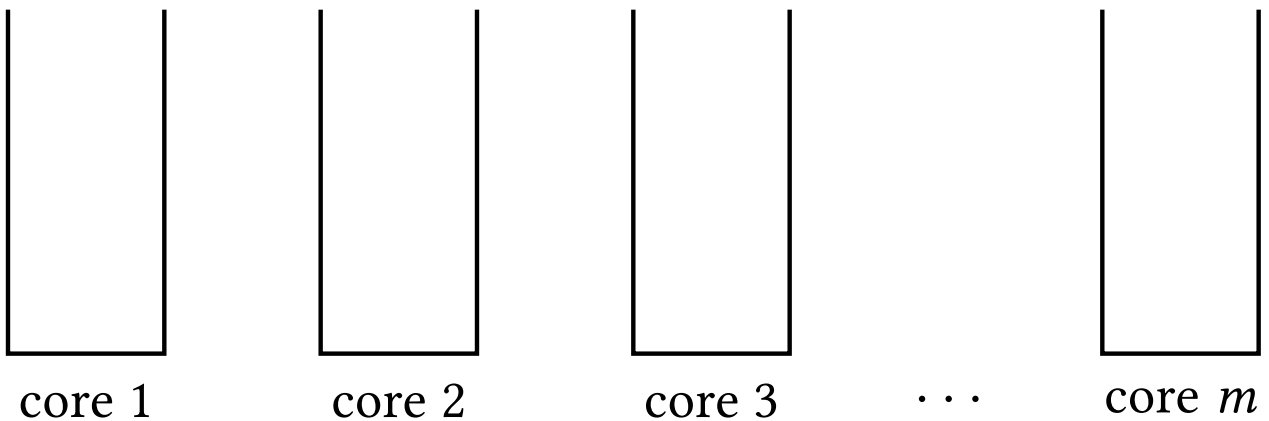
1. Make a partitioning
2. Evaluate the (uniprocessor) schedulability of each partition
3. If this fails for some partition, go back to 1 or give up



Partitioning in other settings

The same general idea applies for other task models and schedulers:

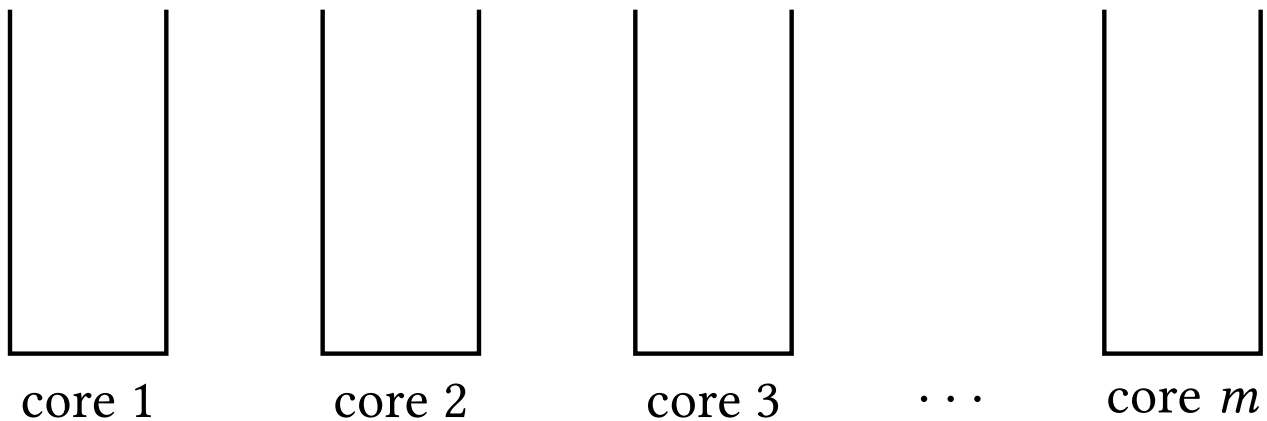
1. Make a partitioning
2. Evaluate the (uniprocessor) schedulability of each partition
3. If this fails for some partition, go back to 1 or give up



Partitioning in other settings

The same general idea applies for other task models and schedulers:

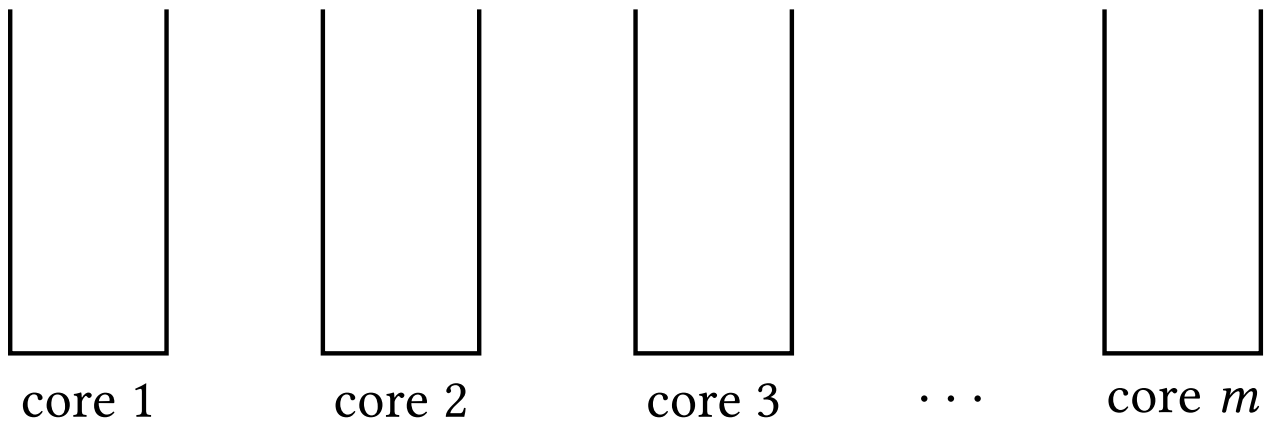
1. Make a partitioning
2. Evaluate the (uniprocessor) schedulability of each partition
3. If this fails for some partition, go back to 1 or give up



Note: We don't need to use the same scheduler on every processor.

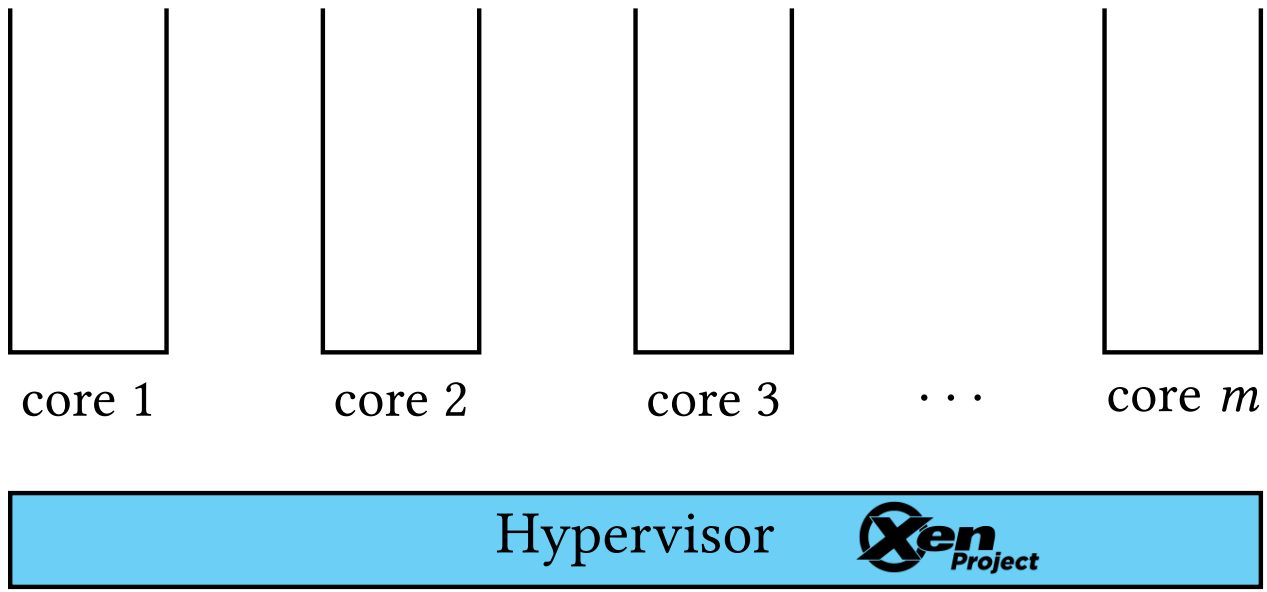
Partitioning for existing subsystems

We could also assign different cores to existing (sub-)systems and control them via a *hypervisor*. For example:



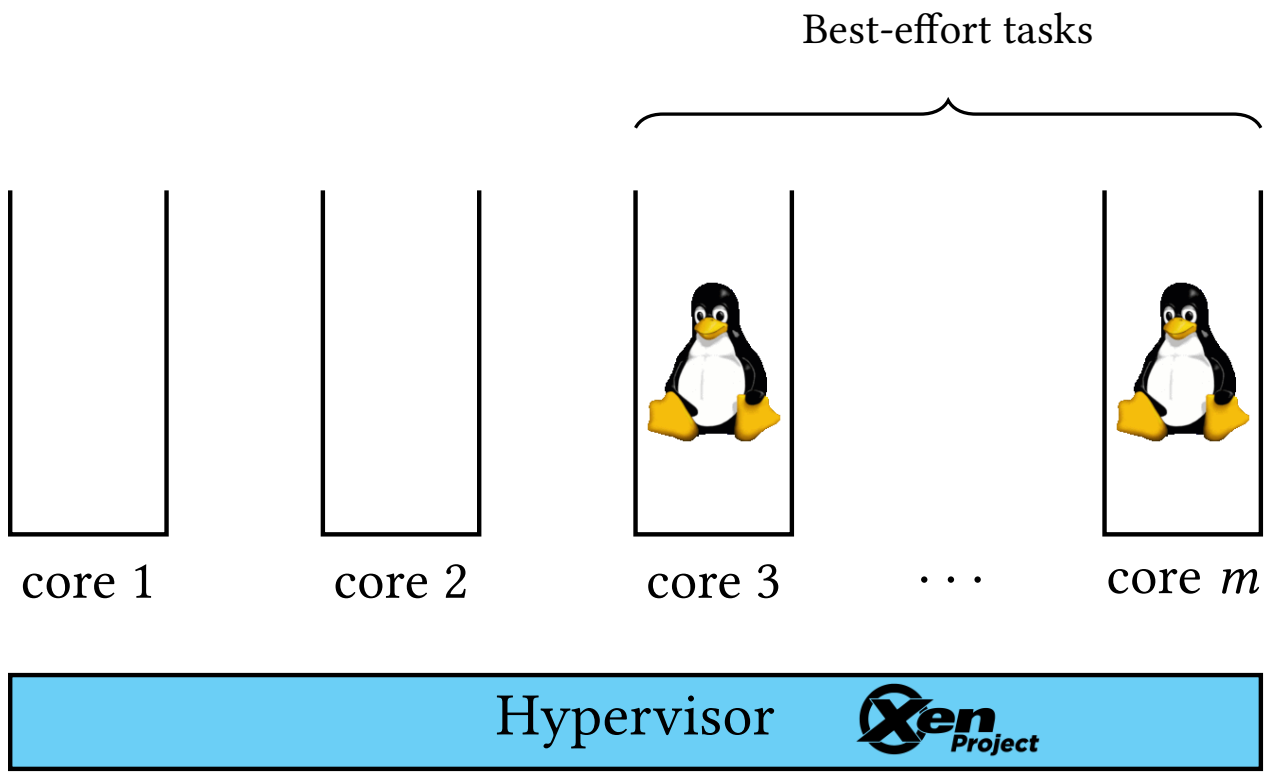
Partitioning for existing subsystems

We could also assign different cores to existing (sub-)systems and control them via a *hypervisor*. For example:



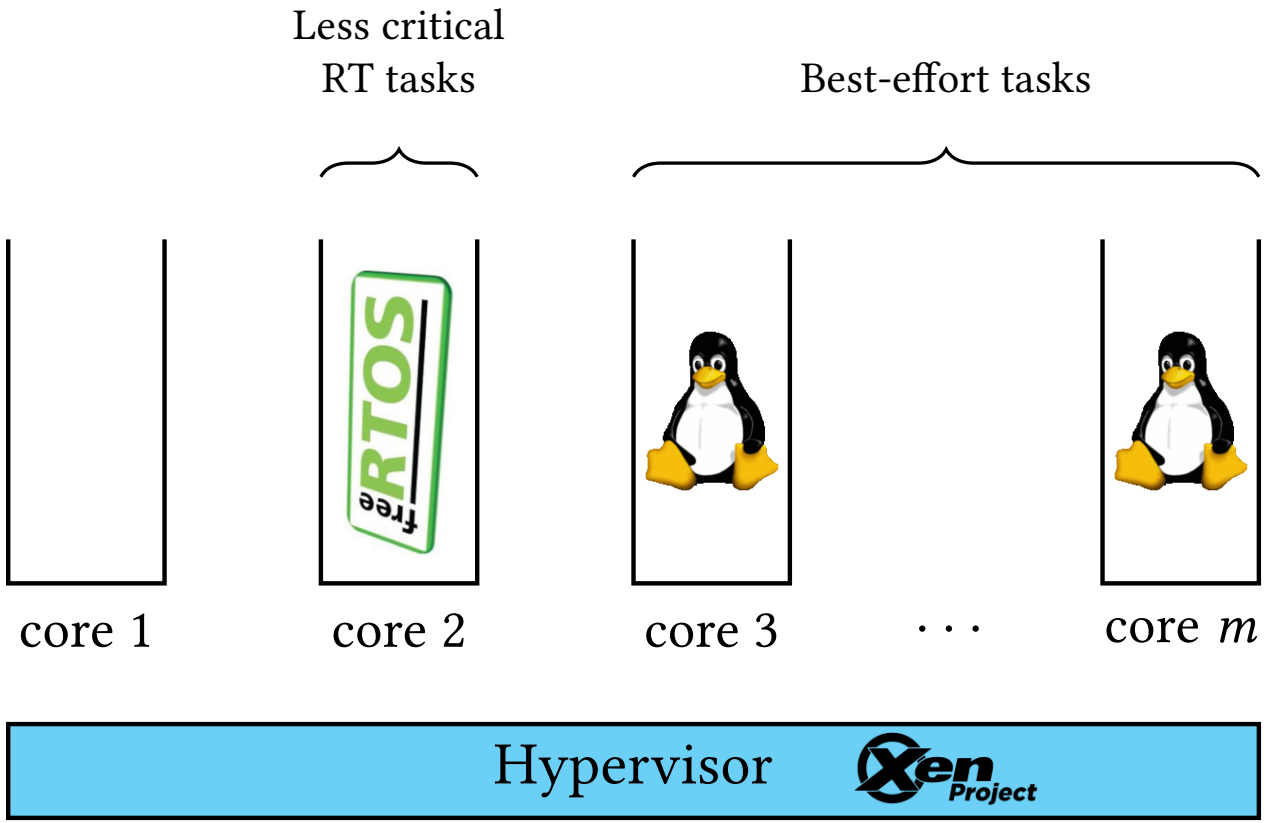
Partitioning for existing subsystems

We could also assign different cores to existing (sub-)systems and control them via a *hypervisor*. For example:



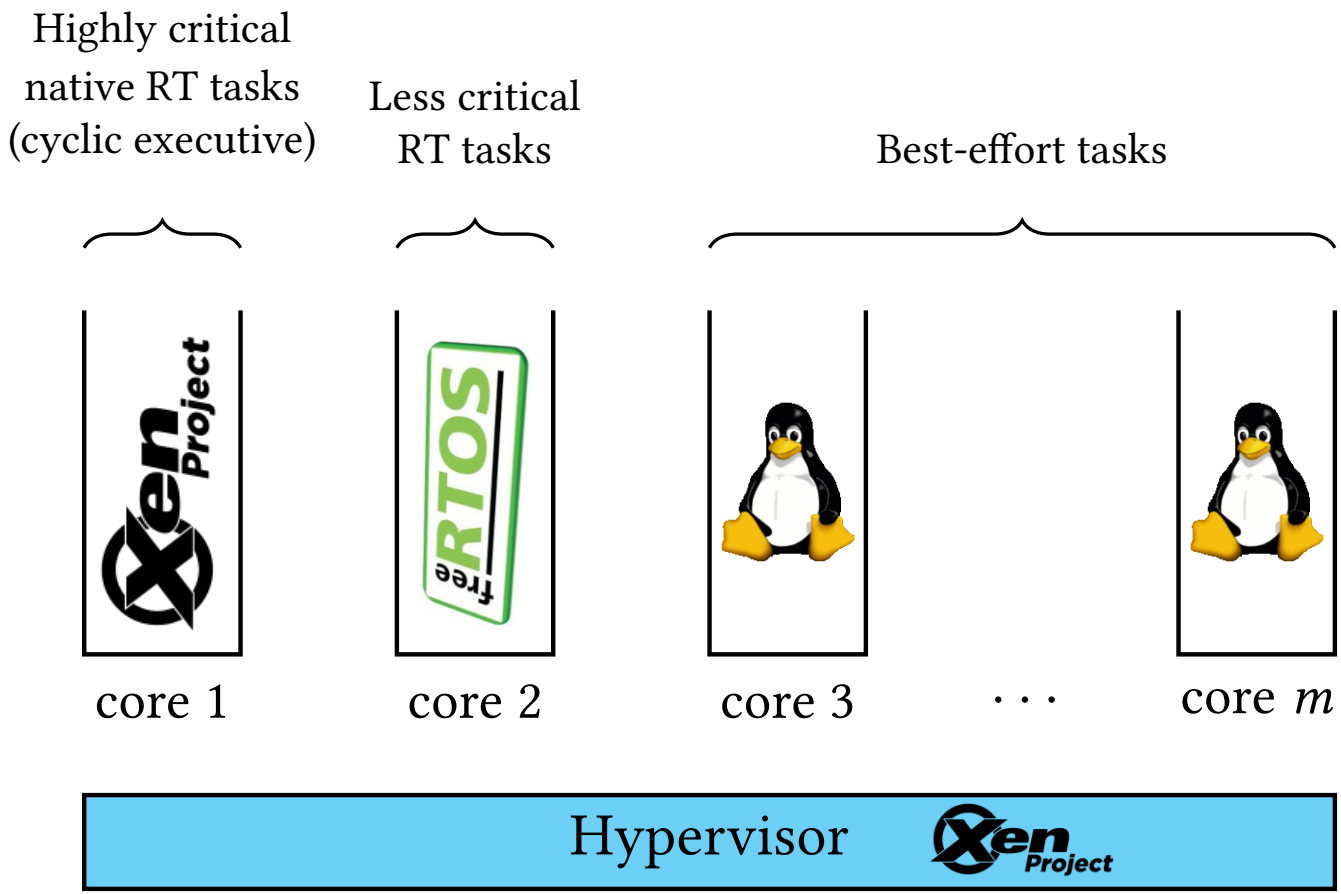
Partitioning for existing subsystems

We could also assign different cores to existing (sub-)systems and control them via a *hypervisor*. For example:



Partitioning for existing subsystems

We could also assign different cores to existing (sub-)systems and control them via a *hypervisor*. For example:



Global scheduling

With global scheduling, different jobs from the same task may execute on different processors. A single job may even *migrate* between processors.

Global scheduling

With global scheduling, different jobs from the same task may execute on different processors. A single job may even *migrate* between processors.

- Global scheduling is in principle more powerful than partitioned scheduling. (Why?)

Global scheduling

With global scheduling, different jobs from the same task may execute on different processors. A single job may even *migrate* between processors.

- Global scheduling is in principle more powerful than partitioned scheduling. (Why?)
- It is often easy to define a "global" version of single processor scheduling algorithms:

  Global EDF (G-EDF): Run the m jobs with the earliest deadlines
  Global FP (G-FP): Run the m jobs with the highest priorities

Global scheduling

With global scheduling, different jobs from the same task may execute on different processors. A single job may even *migrate* between processors.

- Global scheduling is in principle more powerful than partitioned scheduling. (Why?)
- It is often easy to define a "global" version of single processor scheduling algorithms:

  Global EDF (G-EDF): Run the m jobs with the earliest deadlines
  Global FP (G-FP): Run the m jobs with the highest priorities
- Exact schedulability tests are usually *highly intractable*.

Global EDF

Utilization bound

A set $\mathcal{T}$ of implicit-deadline sporadic tasks is schedulable by Global EDF on m preemptive processors if

$$\mathrm{U}(\mathcal{T}) \leqslant 1.$$

Global EDF

Utilization bound

A set $\mathcal{T}$ of implicit-deadline sporadic tasks is schedulable by Global EDF on m preemptive processors if

$$U(\mathcal{T}) \leqslant 1.$$

No better than with a single processor!

But this is only sufficient

No larger bound is valid (Proof on blackboard)

Global EDF—improved utilization bound

Utilization bound, improved

A set $\mathcal{T}$ of implicit-deadline sporadic tasks is schedulable by Global EDF on m preemptive processors if

$$\mathrm{U}(\mathcal{T}) \leqslant m - (m-1)\mathrm{U}_{\max}(\mathcal{T}).$$

Again, it is easier to schedule "light" tasks.

Global FP?

Global Fixed-Priority has similar (but slightly worse) utilization bounds. Details omitted here.

An optimal algorithm for implicit-deadline tasks!

Good news

It is possible to achieve *optimal* scheduling of implicit-deadline sporadic tasks on m processors, with the utilization bound

$$\mathrm{U}(\mathcal{T}) \leqslant m.$$

An optimal algorithm for implicit-deadline tasks!

Good news

It is possible to achieve *optimal* scheduling of implicit-deadline sporadic tasks on m processors, with the utilization bound

$$\mathrm{U}(\mathcal{T}) \leqslant m.$$

Bad news

It is utterly impractical.

Proportionate fairness

Rough idea

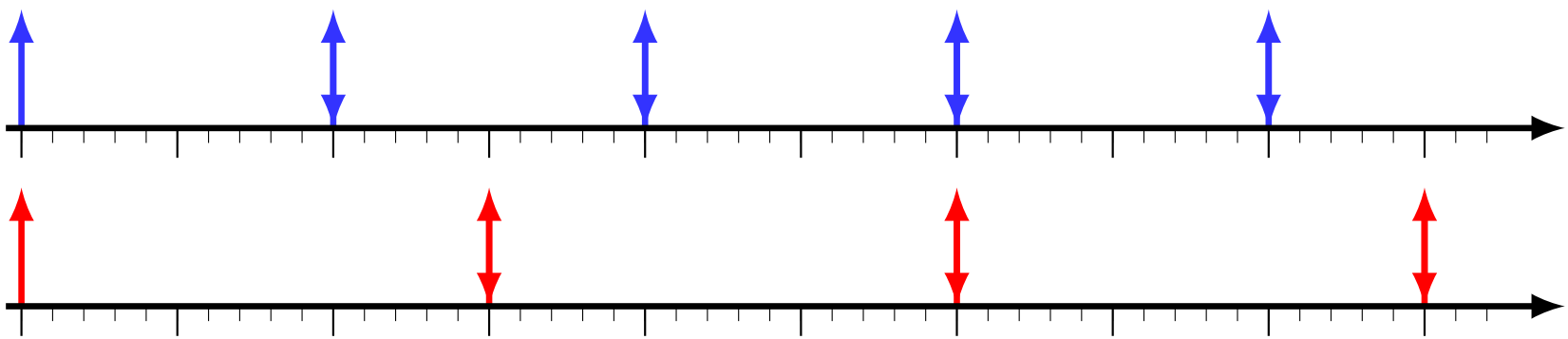
In every time interval of length t, execute task τ_i for a total of $t \cdot \mathrm{U}(\tau_i)$ time units.

Proportionate fairness

Rough idea

In every time interval of length t, execute task τ_i for a total of $t \cdot \mathrm{U}(\tau_i)$ time units.

This is approximated by splitting the execution of the jobs into tiny pieces and distributing them evenly.

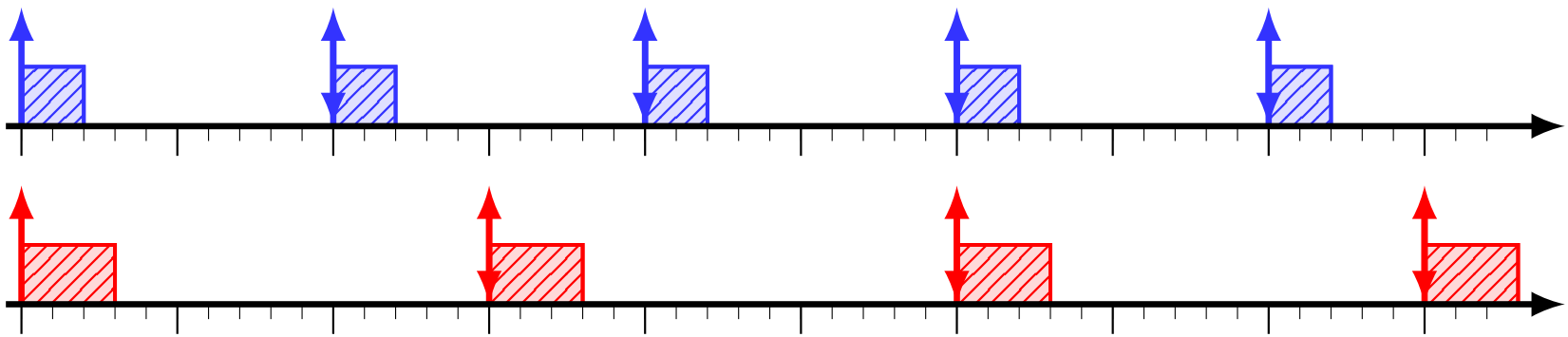


Proportionate fairness

Rough idea

In every time interval of length t, execute task τ_i for a total of $t \cdot \mathrm{U}(\tau_i)$ time units.

This is approximated by splitting the execution of the jobs into tiny pieces and distributing them evenly.

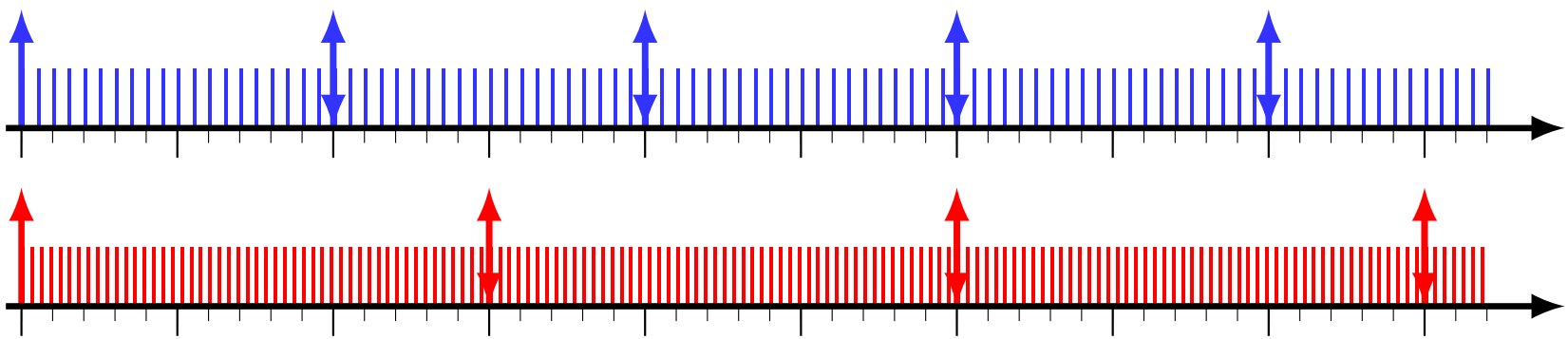


Proportionate fairness

Rough idea

In every time interval of length t, execute task τ_i for a total of $t \cdot \mathrm{U}(\tau_i)$ time units.

This is approximated by splitting the execution of the jobs into tiny pieces and distributing them evenly.

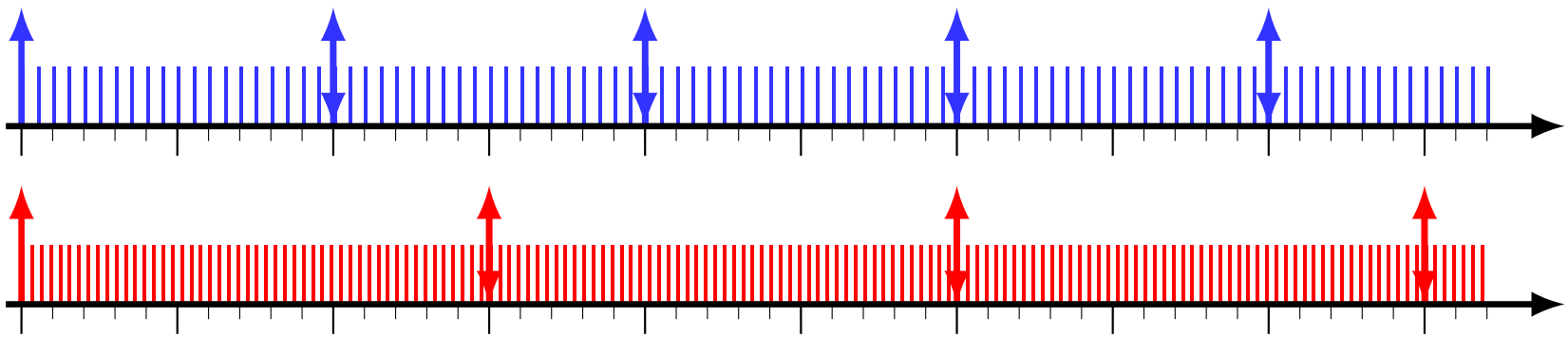


Proportionate fairness

Rough idea

In every time interval of length t, execute task τ_i for a total of $t \cdot \mathrm{U}(\tau_i)$ time units.

This is approximated by splitting the execution of the jobs into tiny pieces and distributing them evenly.



Now it is easy to construct a schedule if the overall rate does not exceed the processing capacity, i.e., if $\mathrm{U}(\mathcal{T}) \leqslant m$.

Why is it impractical?

Why is it impractical?

This approach creates a *huge* number of preemptions and migrations. In reality, they are not for free.

Why is it impractical?

This approach creates a *huge* number of preemptions and migrations. In reality, they are not for free.

Question

How many preemptions can EDF or FP create?

Why is it impractical?

This approach creates a *huge* number of preemptions and migrations. In reality, they are not for free.

Question

How many preemptions can EDF or FP create?

At most one per job. (Why?)

Some main points: Partitioned vs. Global

Partitioned:

- \+ KISS-compatible
- \+ Reuses mature results for single processors
- \+ No migrations
- − Non-optimal
- − Can utilize only 50% of the available resources in the worst case (but more on average)

Global:

- \+ In principle more powerful
- \+ Better at load-balancing and average latency
- − Simple algorithms (G-EDF, G-FP etc.) work poorly for meeting hard deadlines
- − Higher overheads
- − Exact tests are often intractable

Taking the best from both: semi-partitioning

Semi-partitioning

Semi-partitioned scheduling is roughly to

1. partition as many tasks as possible, then
2. *split* the remaining tasks onto several partitions.

Taking the best from both: semi-partitioning

Semi-partitioning

Semi-partitioned scheduling is roughly to

1. partition as many tasks as possible, then
2. *split* the remaining tasks onto several partitions.

We would like the splitting to be done in such a way that ordinary uniprocessor scheduling and analysis can be applied to each processor.

One way for EDF: The $C = D$ approach

To "partition" the task on processors $P_1, \ldots, P_m$ using the $C = D$ approach:

1. Set $k = 1$
2. Put as many tasks as possible on processor P_k
3. Take some task $\tau = (C, D, T)$ that did not fit, and split it into two "tasks"
   - $\tau^1 = (C', C', T)$
   - $\tau^2 = (C - C', D - C', T)$

   such that C' is maximized and τ^1 can fit on P_k.
4. Put τ^1 on P_k and τ^2 on P_{k+1}.
5. Set $k = k + 1$. Repeat from ?? unless $k = m$.
6. Try to put the remaining tasks on P_m.

One way for EDF: The $C = D$ approach

- Read more in *Partitioned EDF Scheduling for Multiprocessors using a C=D Scheme* (Burns et al., 2010)
- Check out this paper for great empirical results using this method (+ some extra tweaks): *Global Scheduling Not Required: Simple, Near-Optimal Multiprocessor Real-Time Scheduling with Semi-Partitioned Reservations* (Brandenburg and Gül, 2016)

Semi-partitioning with fixed-priority?

Recall

A task set $\mathcal{T}$ with implicit deadlines is schedulable with RM priority ordering on a single processor if

$$\mathrm{U}(\mathcal{T}) \leqslant \ln(2) \approx 0.69.$$

There exists a semi-partitioned FP algorithm for implicit-deadline sporadic tasks that achieves a perfect scaling to m processors:

$$\mathrm{U}(\mathcal{T}) \leqslant m \cdot \ln(2)$$

- Read more in *Fixed-Priority Multiprocessor Scheduling with Liu & Layland's Utilization Bound* (Guan et al., 2010)