THE LINUX SCHEDULER: A DECADE OF WASTED CORES

Jean-Pierre Lozi
jplozi@unice.fr
Université Nice Sophia Antipolis

Baptiste Lepers
baptiste.lepers@epfl.ch
ÉCOLE POLYTECHNIQUE FÉDÉRALE DE LAUSANNE

Fabien Gaud
me@fabiengaud.net
COHO DATA

Alexandra Fedorova
sasha@ece.ubc.ca

Justin Funston
jfunston@ece.ubc.ca
UBC

Vivien Quéma
vivien.quema@imag.fr
Grenoble INP Ensimag

INTRODUCTION

- Take a machine with a lot of cores (64 in our case)

INTRODUCTION

- Take a machine with a lot of cores (64 in our case)
- **Run two CPU-intensive processes in two terminals (e.g. R scripts):**

```
R < script.R --nosave &
R < script.R --nosave &
```

INTRODUCTION

- Take a machine with a lot of cores (64 in our case)
- **Run two CPU-intensive processes in two terminals (e.g. R scripts):**

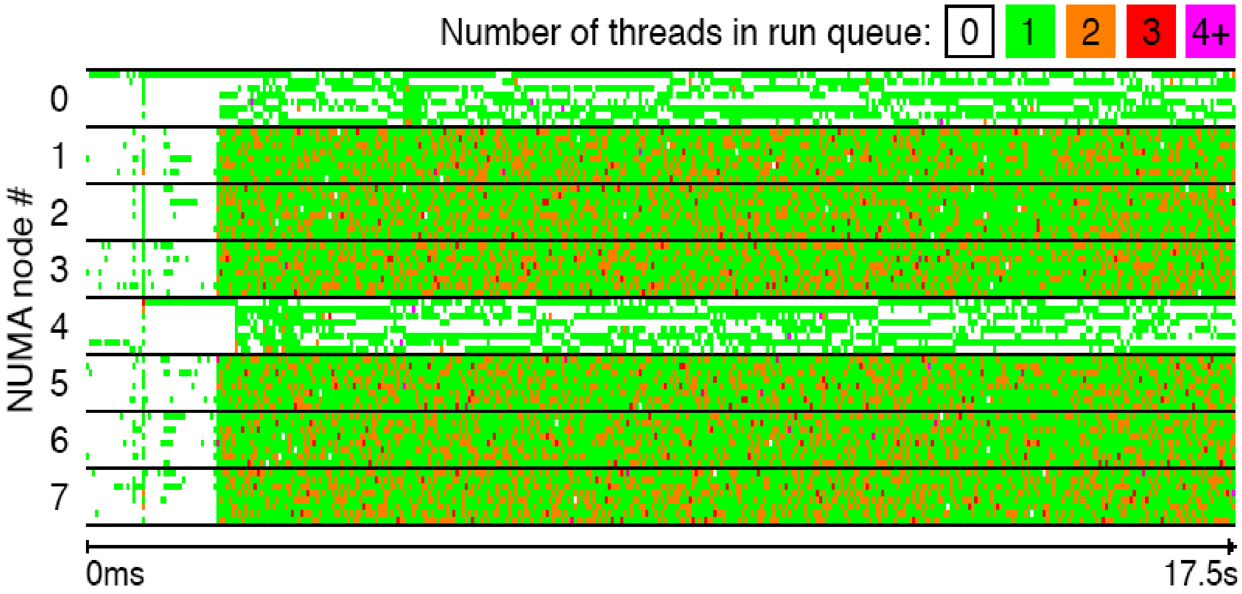
```
R < script.R --nosave &
R < script.R --nosave &
```

- **Compile your kernel in a third terminal:**

```
make –j 62 kernel
```

INTRODUCTION

- Take a machine with a lot of cores (64 in our case)
- **Run two CPU-intensive processes in two terminals (e.g. R scripts):**
  R < script.R --nosave &
  R < script.R --nosave &
- **Compile your kernel in a third terminal:**
  make –j 62 kernel
- **Here is what might happen:**



INTRODUCTION

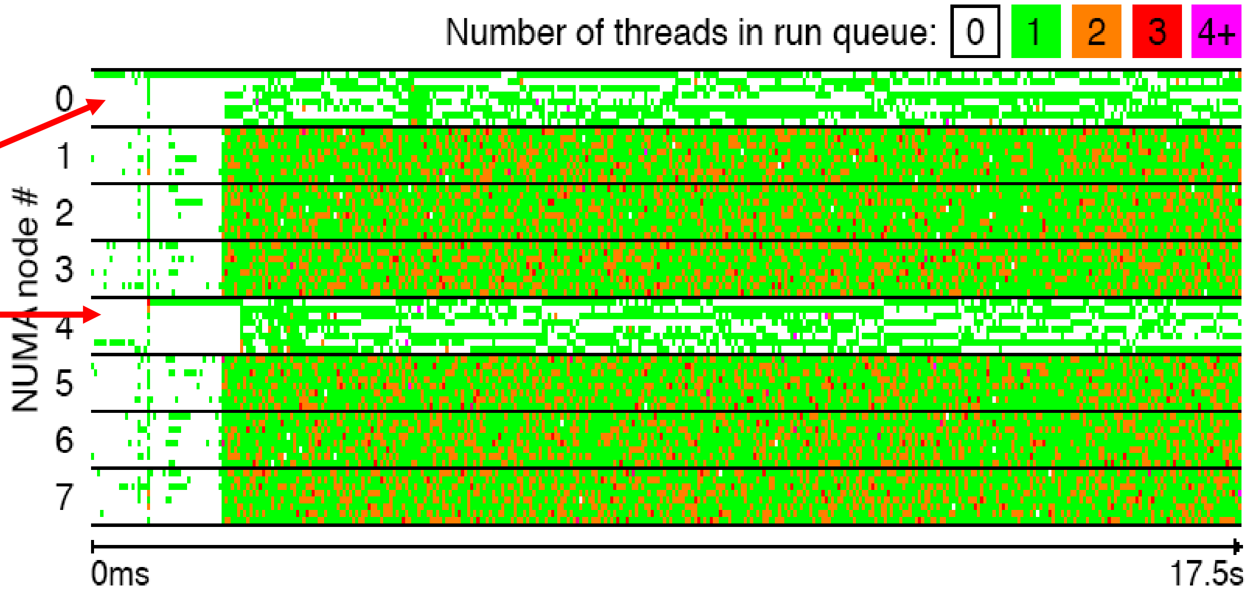
- Take a machine with a lot of cores (64 in our case)
- **Run two CPU-intensive processes in two terminals (e.g. R scripts):**

R < script.R --nosave &
R < script.R --nosave &

- **Compile your kernel in a third terminal:**

make –j 62 kernel

- **Here is what might happen:**
  - **Two NUMA nodes with many idle cores (white)**



INTRODUCTION

- Take a machine with a lot of cores (64 in our case)

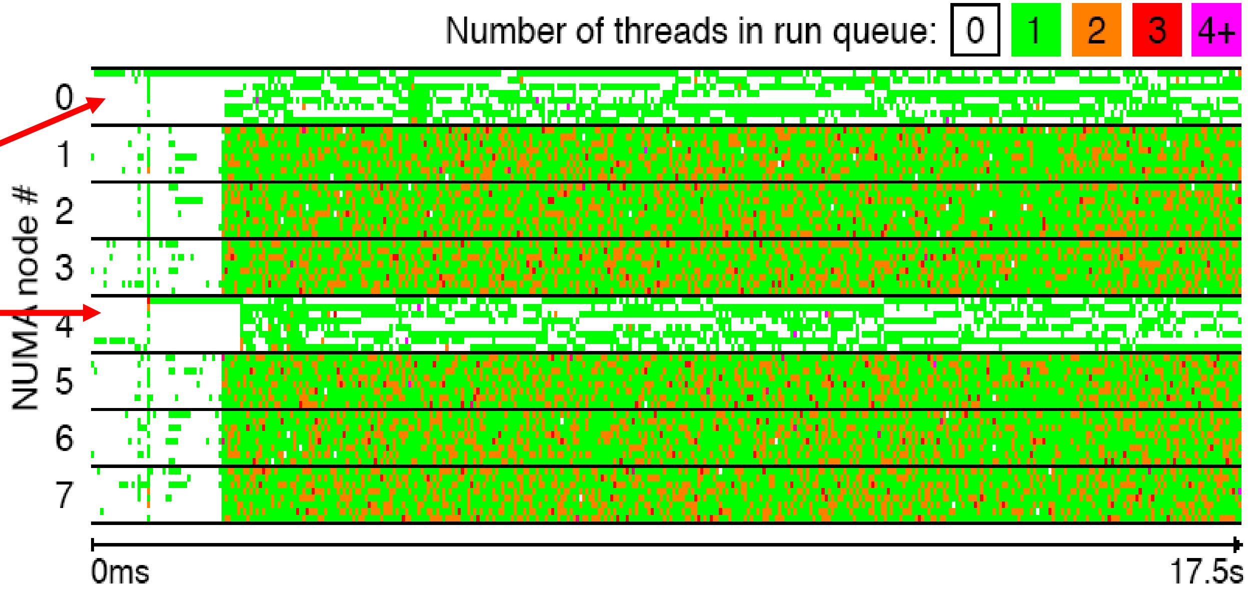
- **Run two CPU-intensive processes in two terminals (e.g. R scripts):**

R < script.R --nosave &
R < script.R --nosave &

- **Compile your kernel in a third terminal:**

make –j 62 kernel

- **Here is what might happen:**
  - **Two NUMA nodes with many idle cores (white)**
  - **Other NUMA nodes with many overloaded cores (orange, red)**



INTRODUCTION

Performance degradation: 14% for the make process!

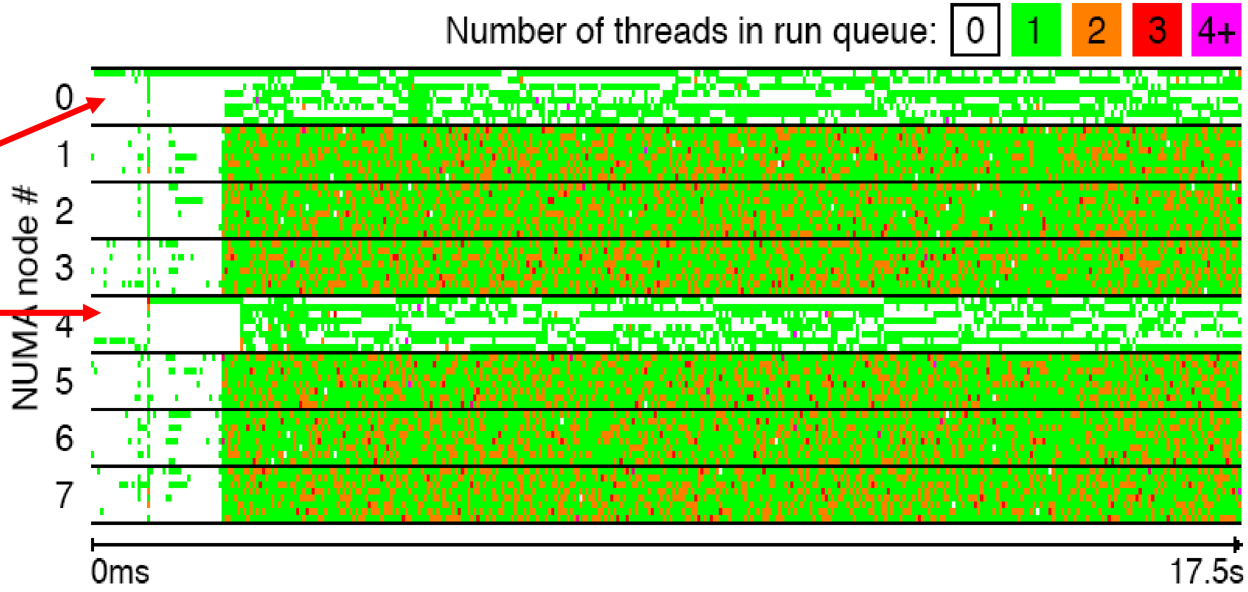
- Take a machine with a lot of cores (64 in our case)
- **Run two CPU-intensive processes in two terminals (e.g. R scripts):**

R < script.R --nosave &
R < script.R --nosave &

- **Compile your kernel in a third terminal:**

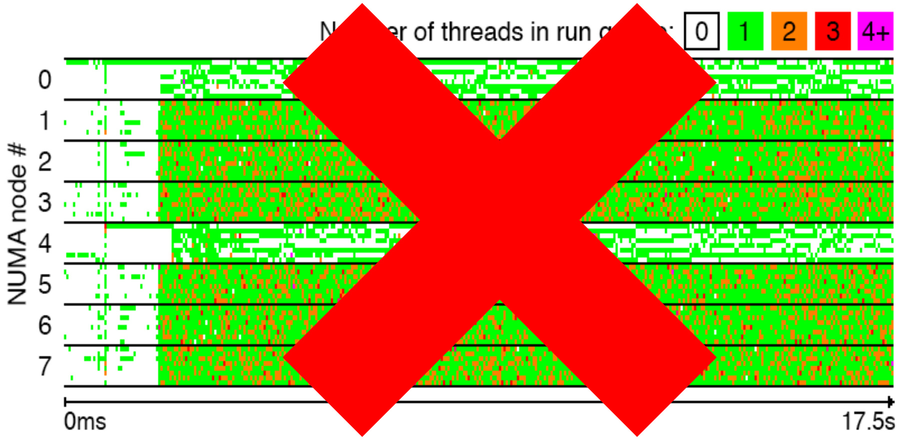
make –j 62 kernel

- **Here is what might happen:**
  - **Two NUMA nodes with many idle cores (white)**
  - **Other NUMA nodes with many overloaded cores (orange, red)**



INTRODUCTION

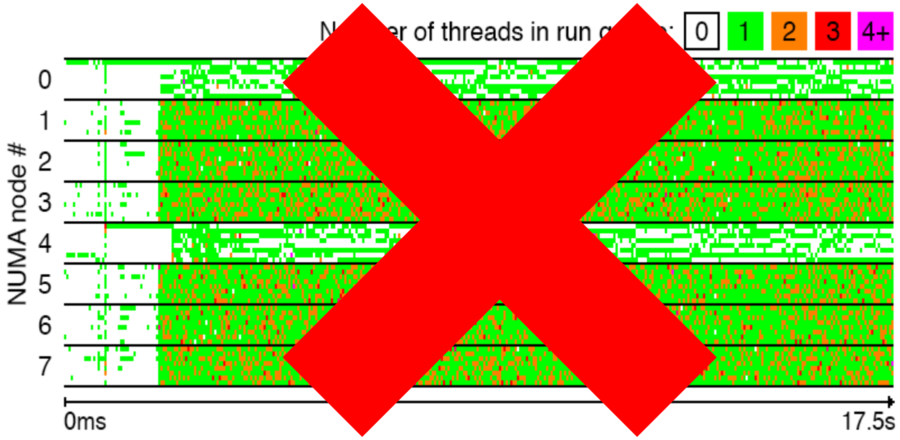
- General-purpose schedulers aim to be work-conserving on multicore architectures



INTRODUCTION

- General-purpose schedulers aim to be work-conserving on multicore architectures
- **Basic invariant:** no idle cores if some cores have several threads in their runqueues
  - *Can actually happen, but only in transient situations!*

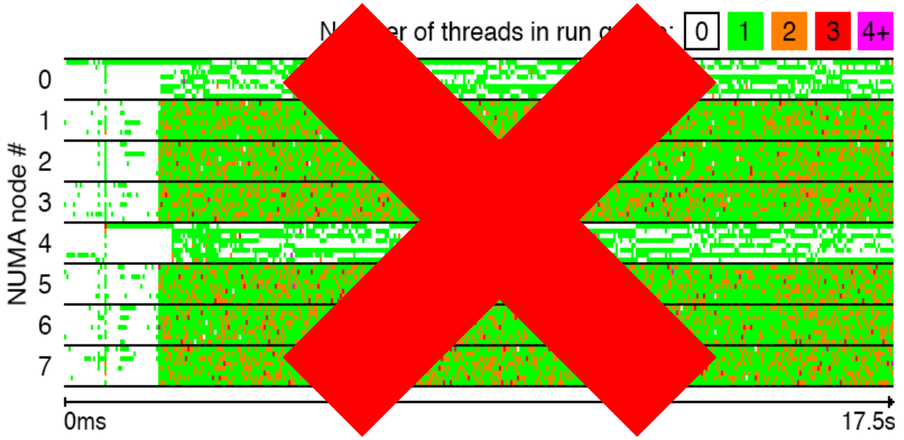
INTRODUCTION



- General-purpose schedulers aim to be work-conserving on multicore architectures
- **Basic invariant:** no idle cores if some cores have several threads in their runqueues
  - *Can actually happen, but only in transient situations!*

We found four major bugs that break this invariant in the Linux scheduler (CFS)!

INTRODUCTION

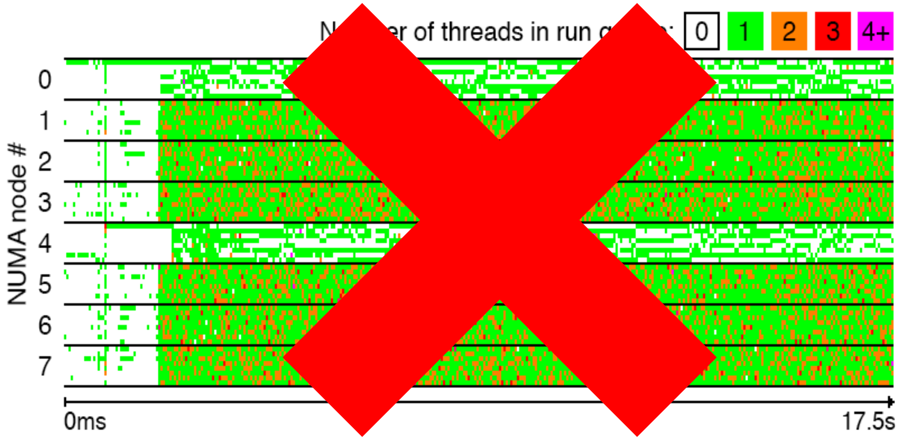


- General-purpose schedulers aim to be work-conserving on multicore architectures
- **Basic invariant:** no idle cores if some cores have several threads in their runqueues
  - *Can actually happen, but only in transient situations!*

We found four major bugs that break this invariant in the Linux scheduler (CFS)!

- **This talk:** presentation of the CFS scheduler + issues we found + discussion

INTRODUCTION



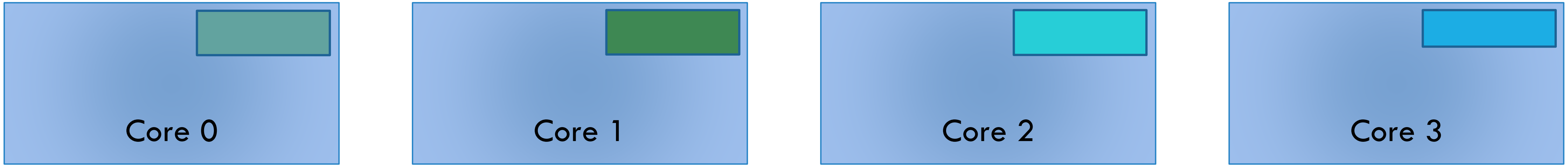
- General-purpose schedulers aim to be work-conserving on multicore architectures
- **Basic invariant:** no idle cores if some cores have several threads in their runqueues
  - *Can actually happen, but only in transient situations!*

We found four major bugs that break this invariant in the Linux scheduler (CFS)!

- **This talk:** presentation of the CFS scheduler + issues we found + discussion

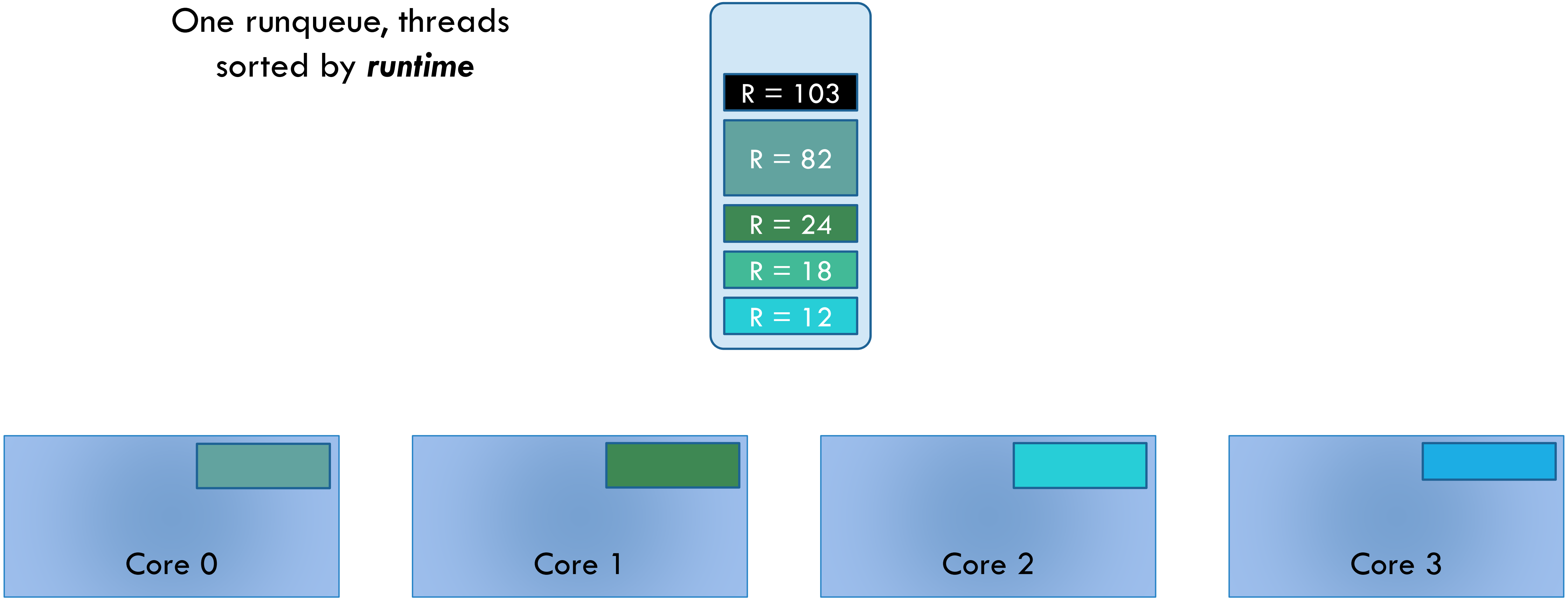
Disclaimer: this is a motivation paper!
Don't expect a solved problem ☺

THE COMPLETELY FAIR SCHEDULER (CFS): CONCEPT



THE COMPLETELY FAIR SCHEDULER (CFS): CONCEPT

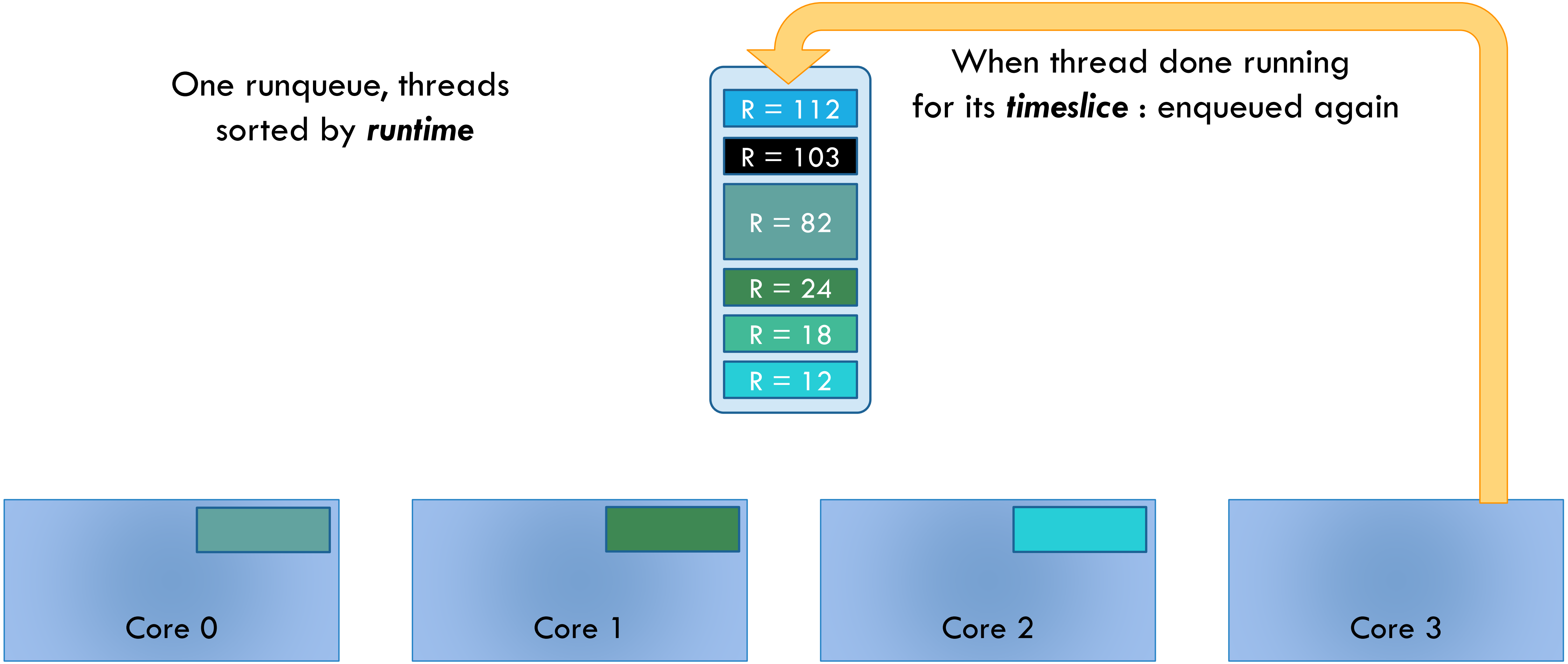
One runqueue, threads sorted by ***runtime***



THE COMPLETELY FAIR SCHEDULER (CFS): CONCEPT

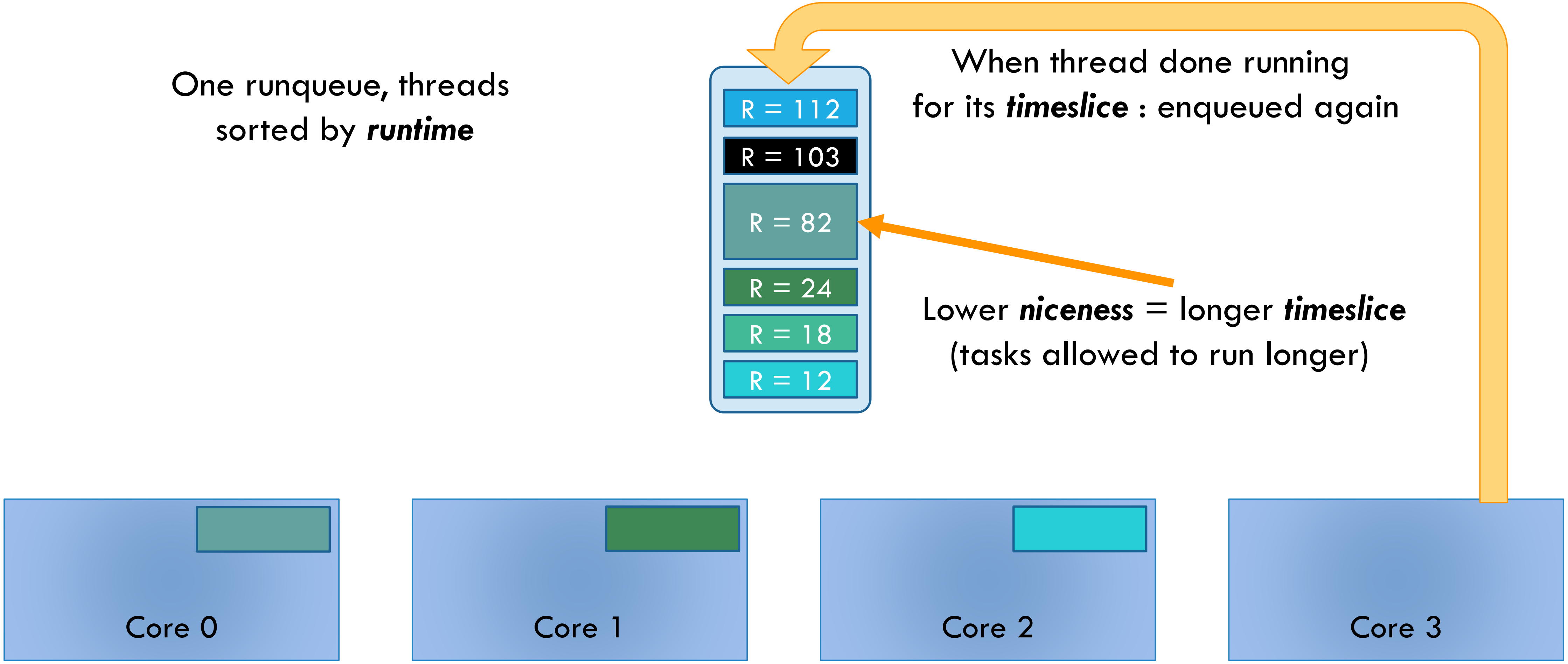
One runqueue, threads sorted by ***runtime***

When thread done running for its ***timeslice*** : enqueued again



THE COMPLETELY FAIR SCHEDULER (CFS): CONCEPT

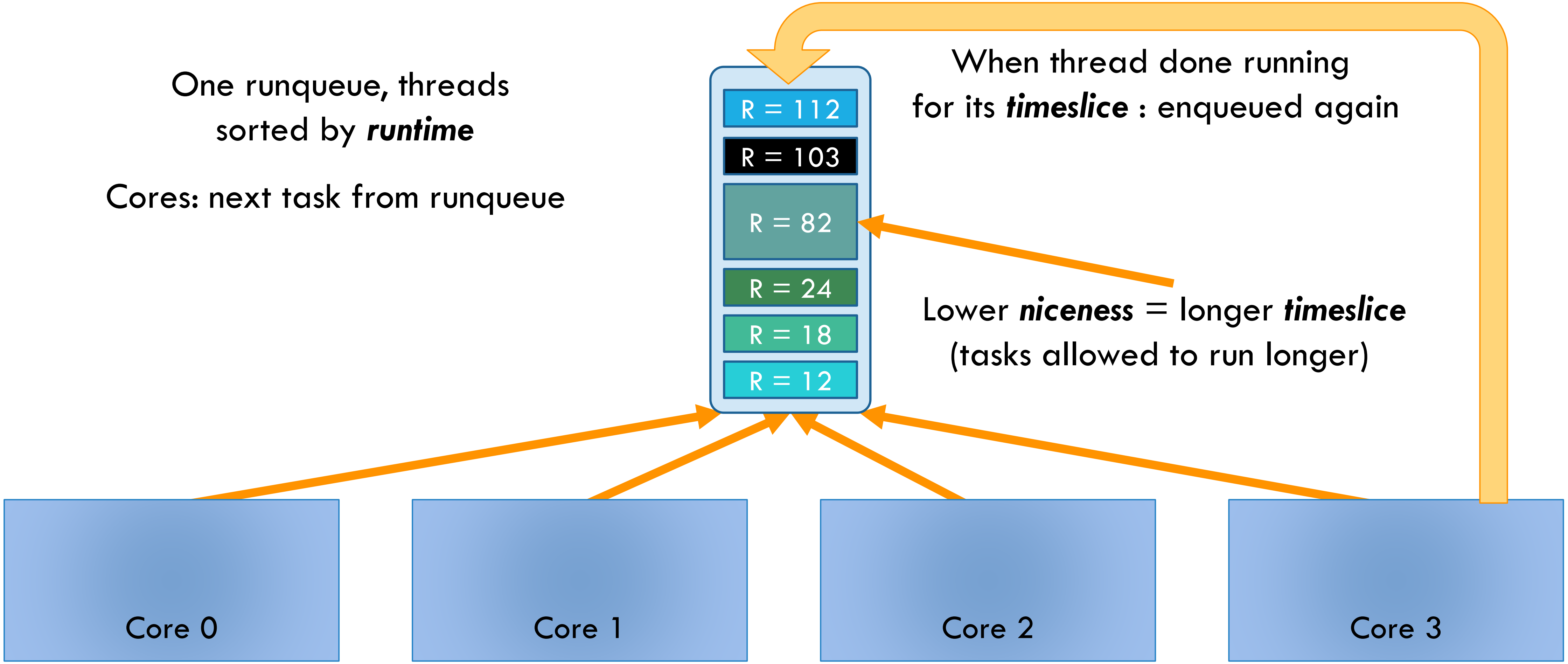
One runqueue, threads sorted by ***runtime***



THE COMPLETELY FAIR SCHEDULER (CFS): CONCEPT

One runqueue, threads sorted by ***runtime***

Cores: next task from runqueue

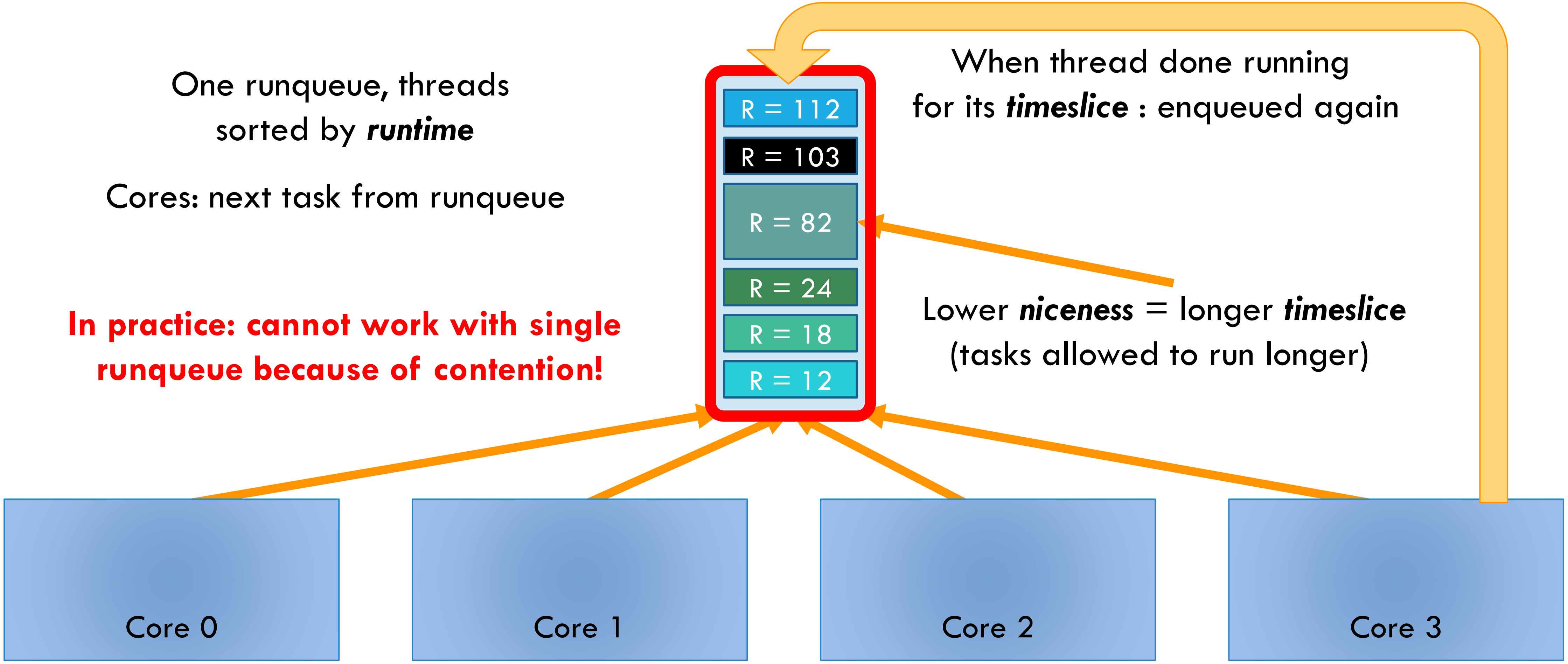


THE COMPLETELY FAIR SCHEDULER (CFS): CONCEPT

One runqueue, threads sorted by ***runtime***

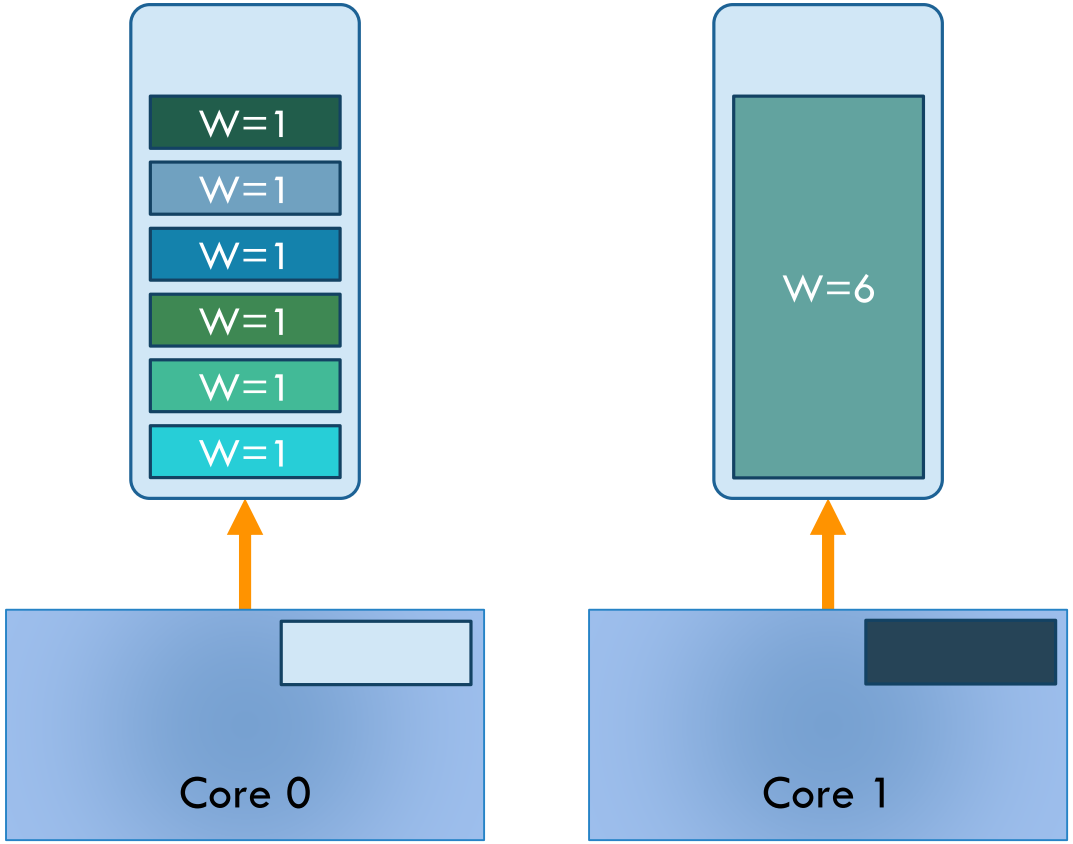
Cores: next task from runqueue

In practice: cannot work with single runqueue because of contention!



CFS: IN PRACTICE

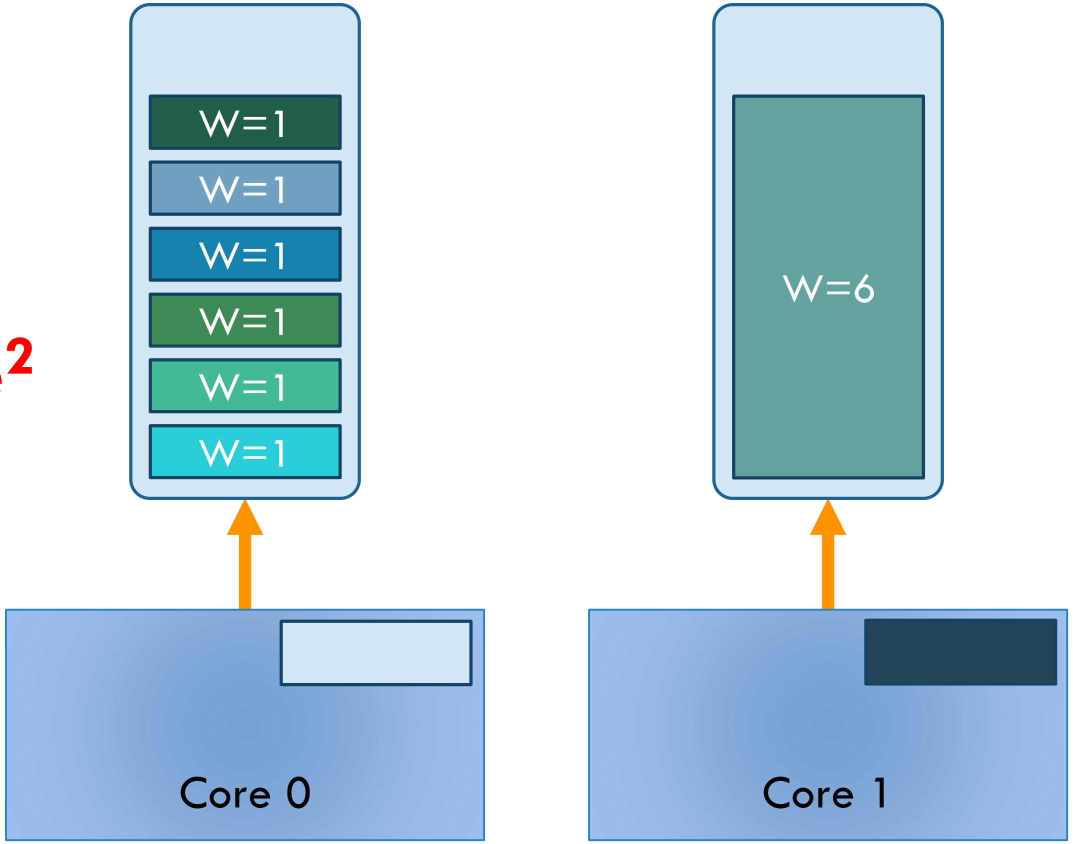
- **One runqueue per core** to avoid contention



CFS: IN PRACTICE

- **One runqueue per core** to avoid contention
- **CFS** **periodically** **balances "loads":**

load(task) = weight[1] x % cpu use[2]

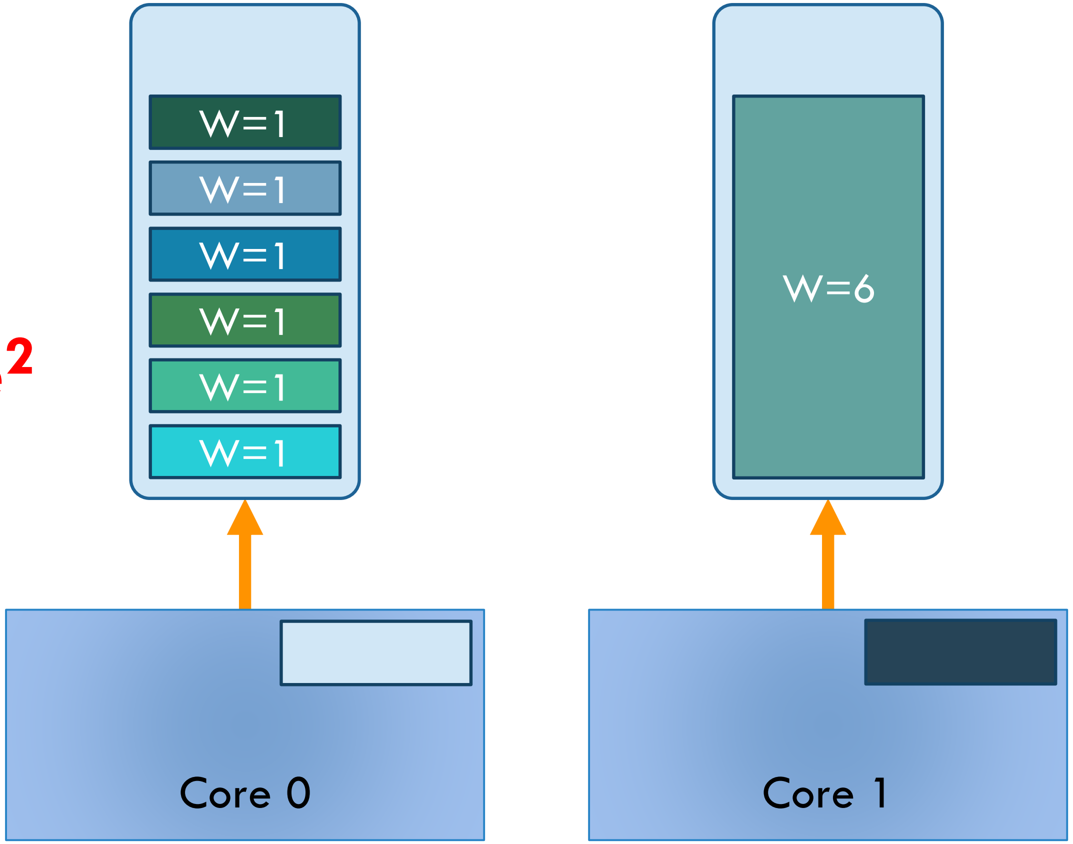


CFS: IN PRACTICE

- **One runqueue per core** to avoid contention
- **CFS** **periodically** **balances “loads”:**

load(task) = weight[1] x % cpu use[2]

[1] Lower niceness = higher weight



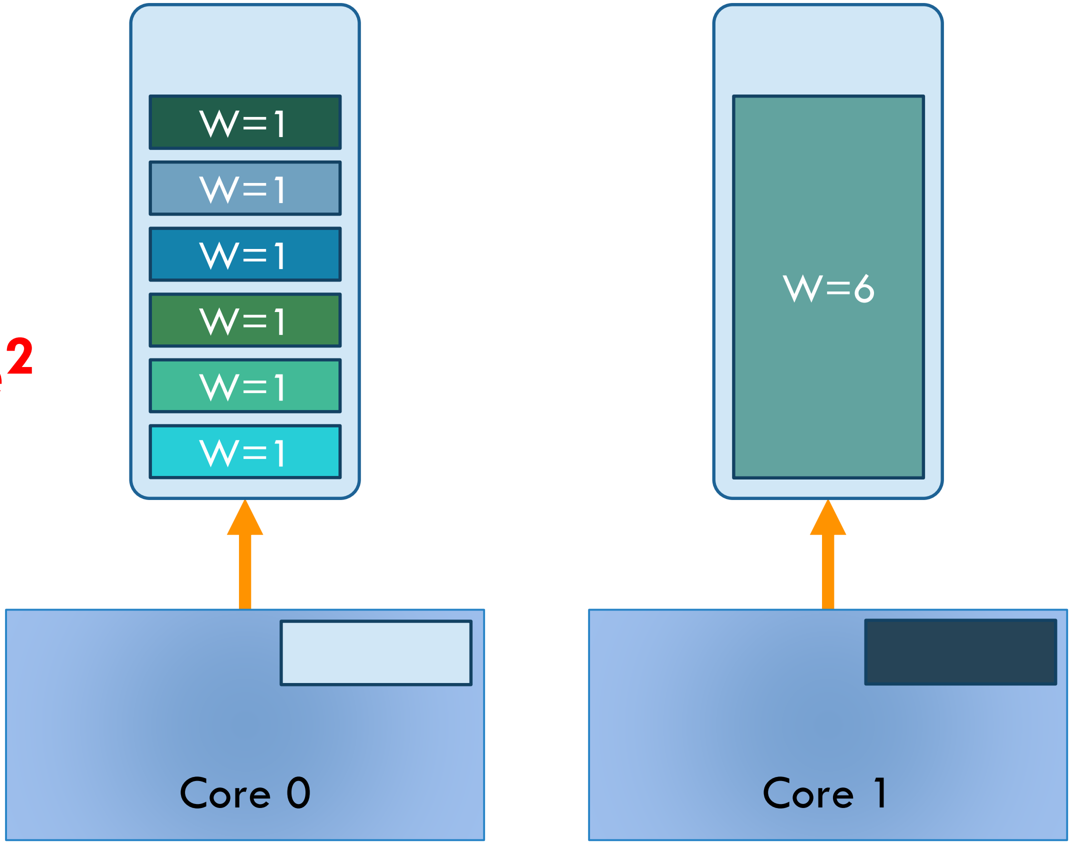
CFS: IN PRACTICE

- **One runqueue per core** to avoid contention
- **CFS periodically balances "loads":**

load(task) = weight[1] x % cpu use[2]

[1] Lower niceness = higher weight

[2] Prevent high-priority thread from taking whole CPU just to sleep



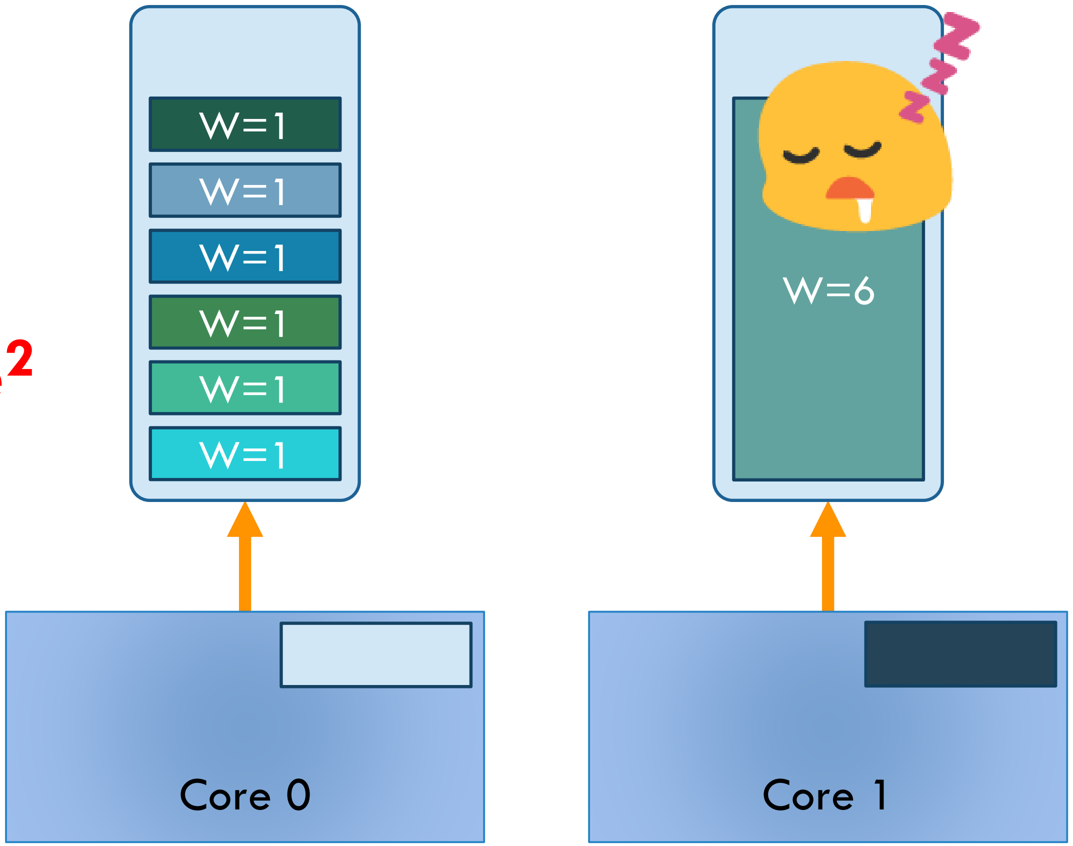
CFS: IN PRACTICE

- **One runqueue per core** to avoid contention
- **CFS** **periodically** **balances "loads":**

load(task) = weight[1] x % cpu use[2]

[1] Lower niceness = higher weight

[2] Prevent high-priority thread from taking whole CPU just to sleep



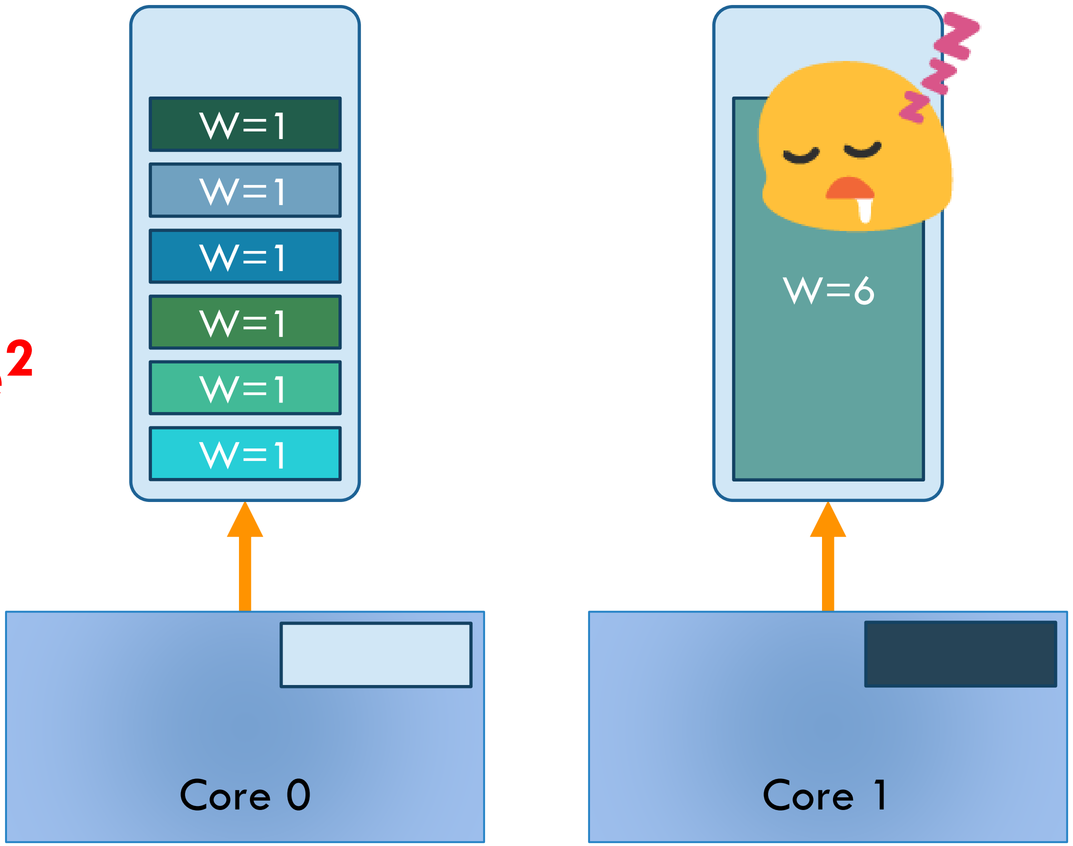
CFS: IN PRACTICE

- **One runqueue per core** to avoid contention
- **CFS periodically balances "loads":**

load(task) = weight[1] x % cpu use[2]

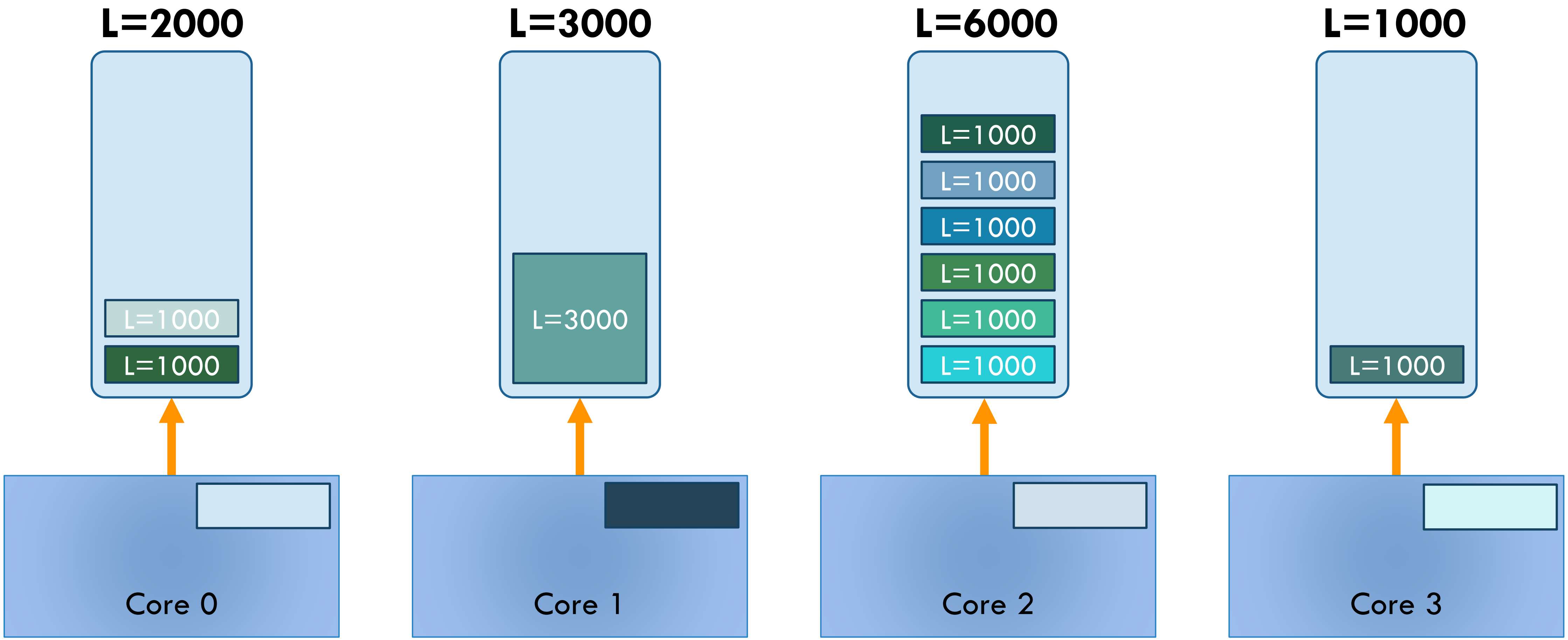
[1] Lower niceness = higher weight

[2] Prevent high-priority thread from taking whole CPU just to sleep

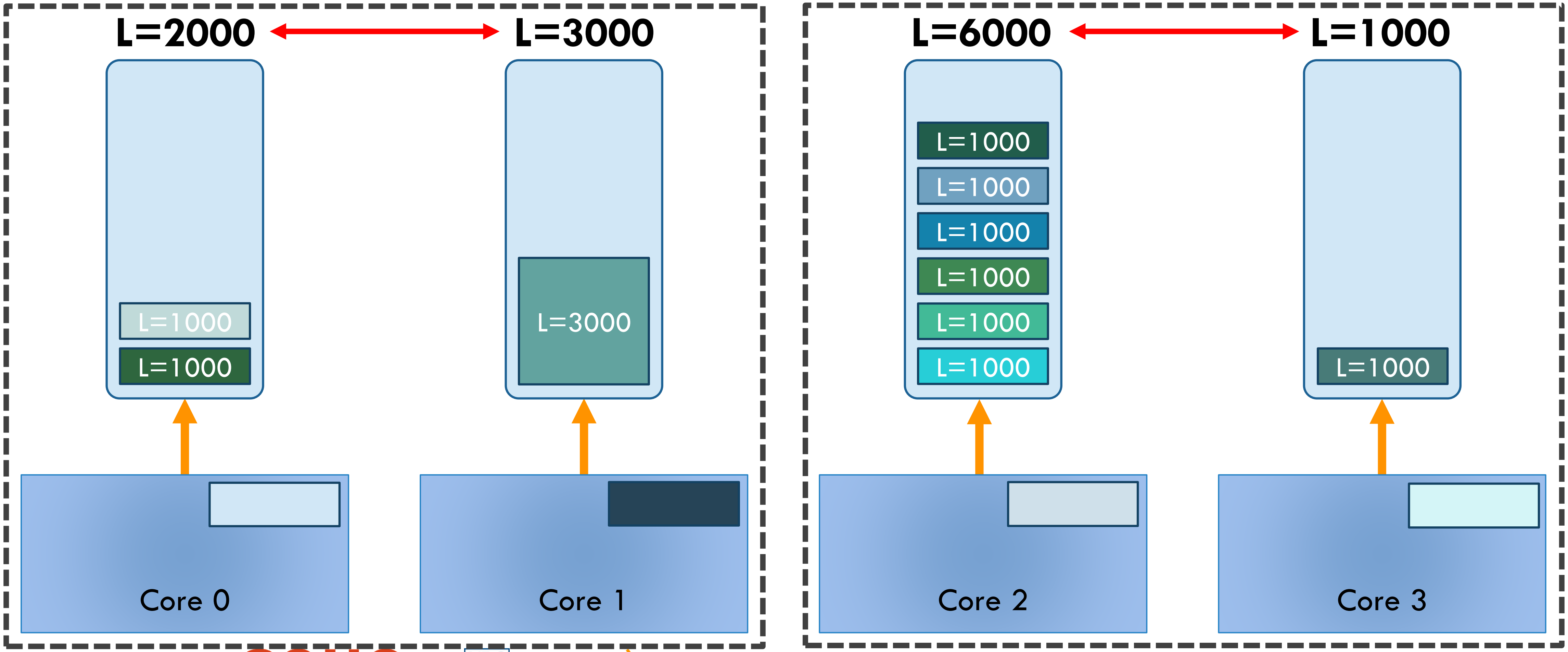


- **Since there can be many cores: hierarchical approach!**

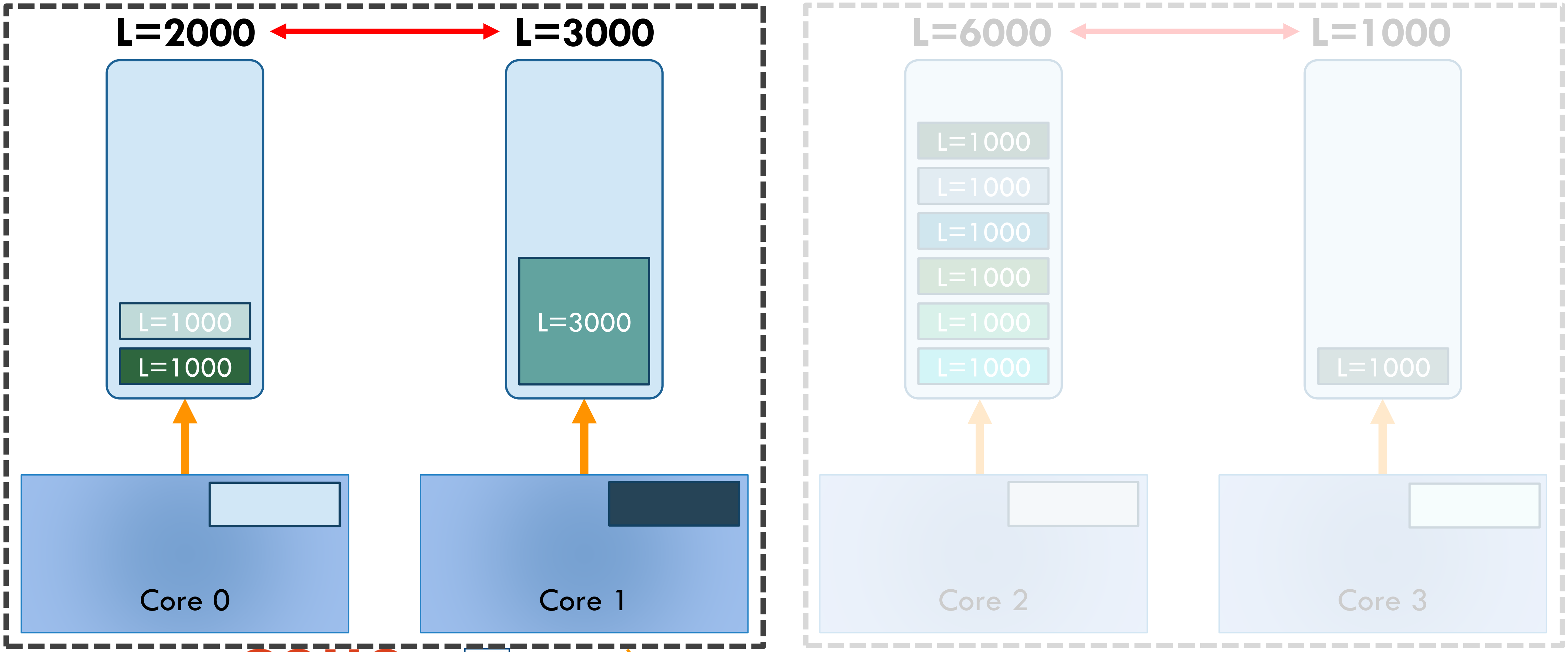
CFS: BALANCING THE LOAD



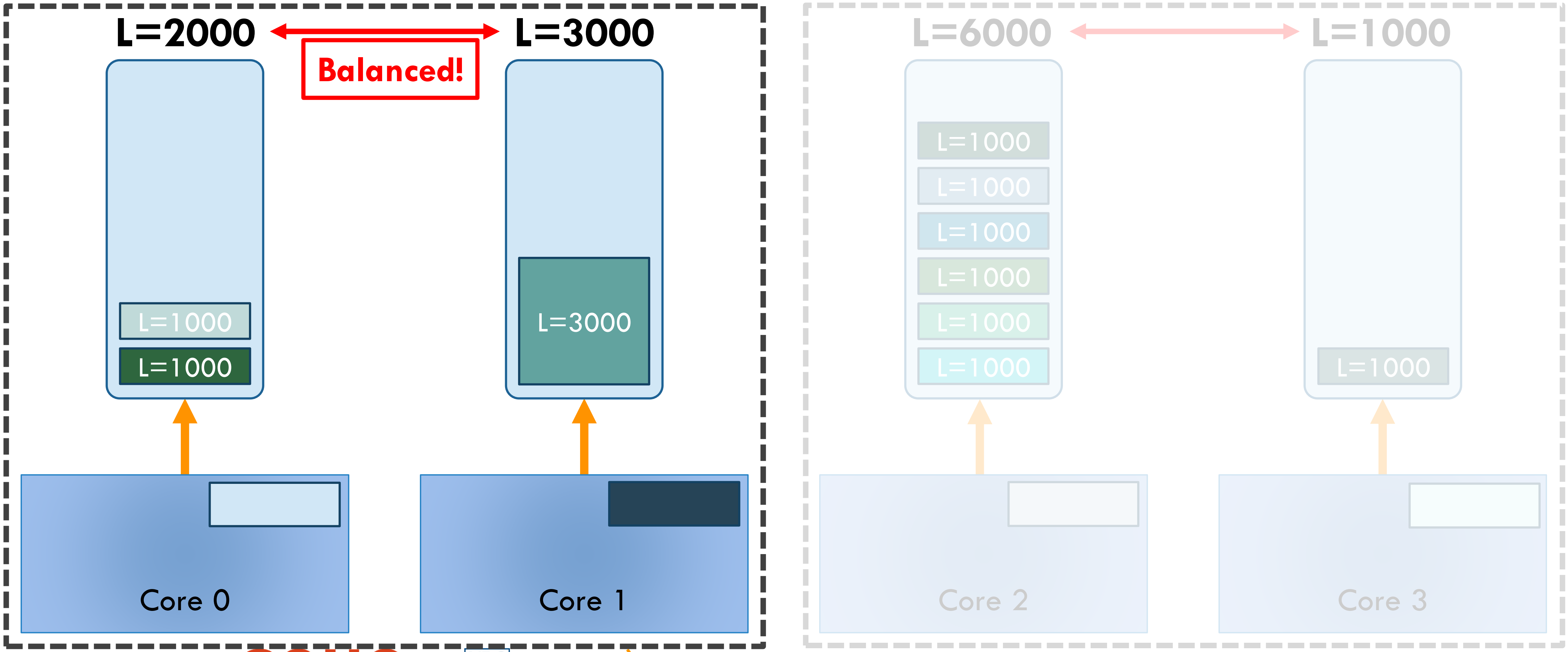
CFS: BALANCING THE LOAD



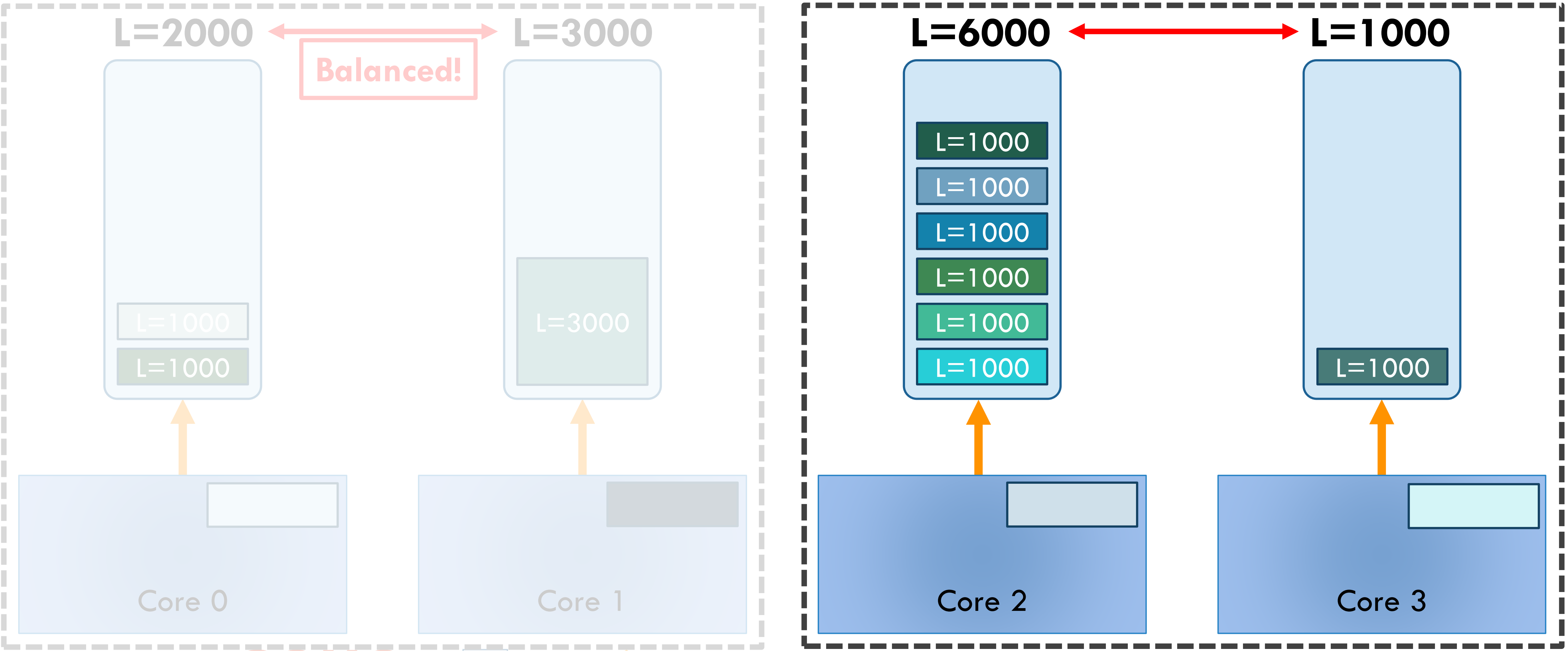
CFS: BALANCING THE LOAD



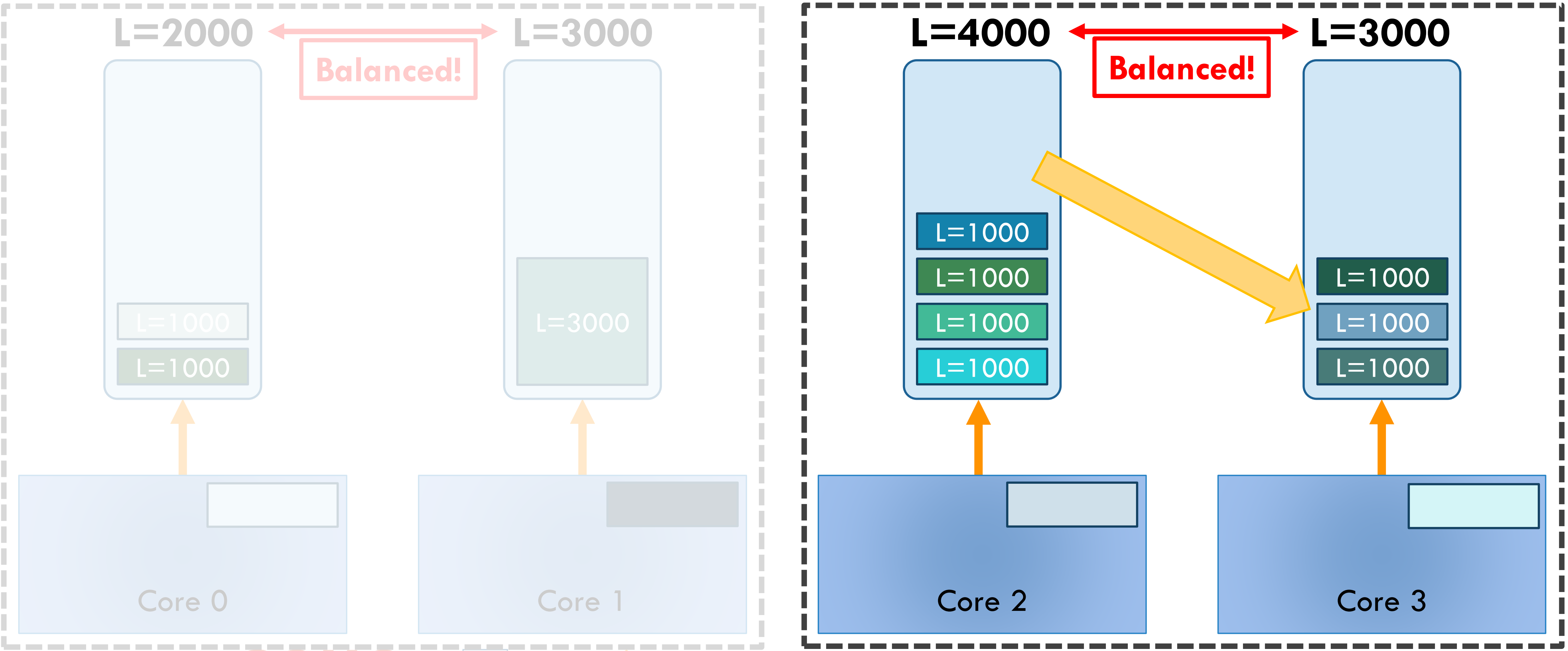
CFS: BALANCING THE LOAD



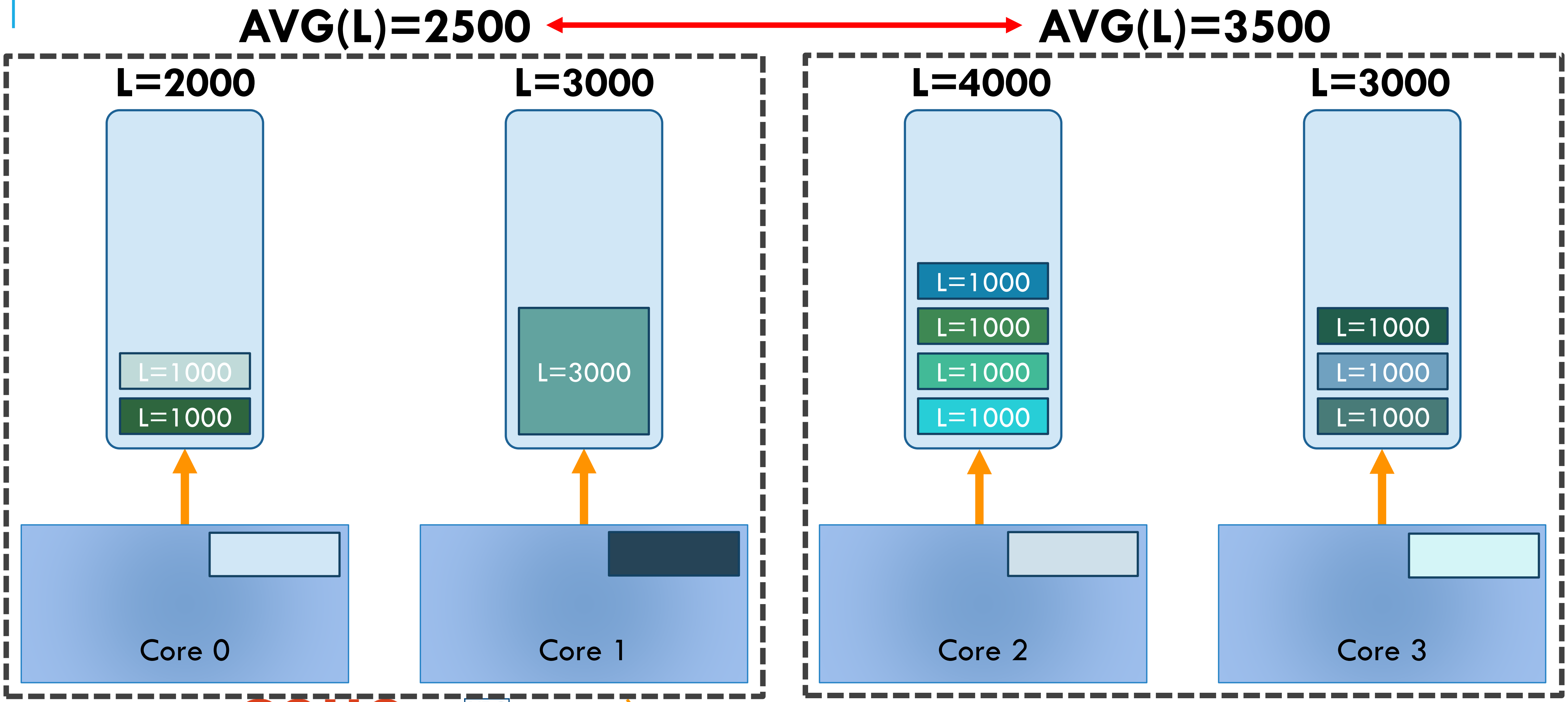
CFS: BALANCING THE LOAD



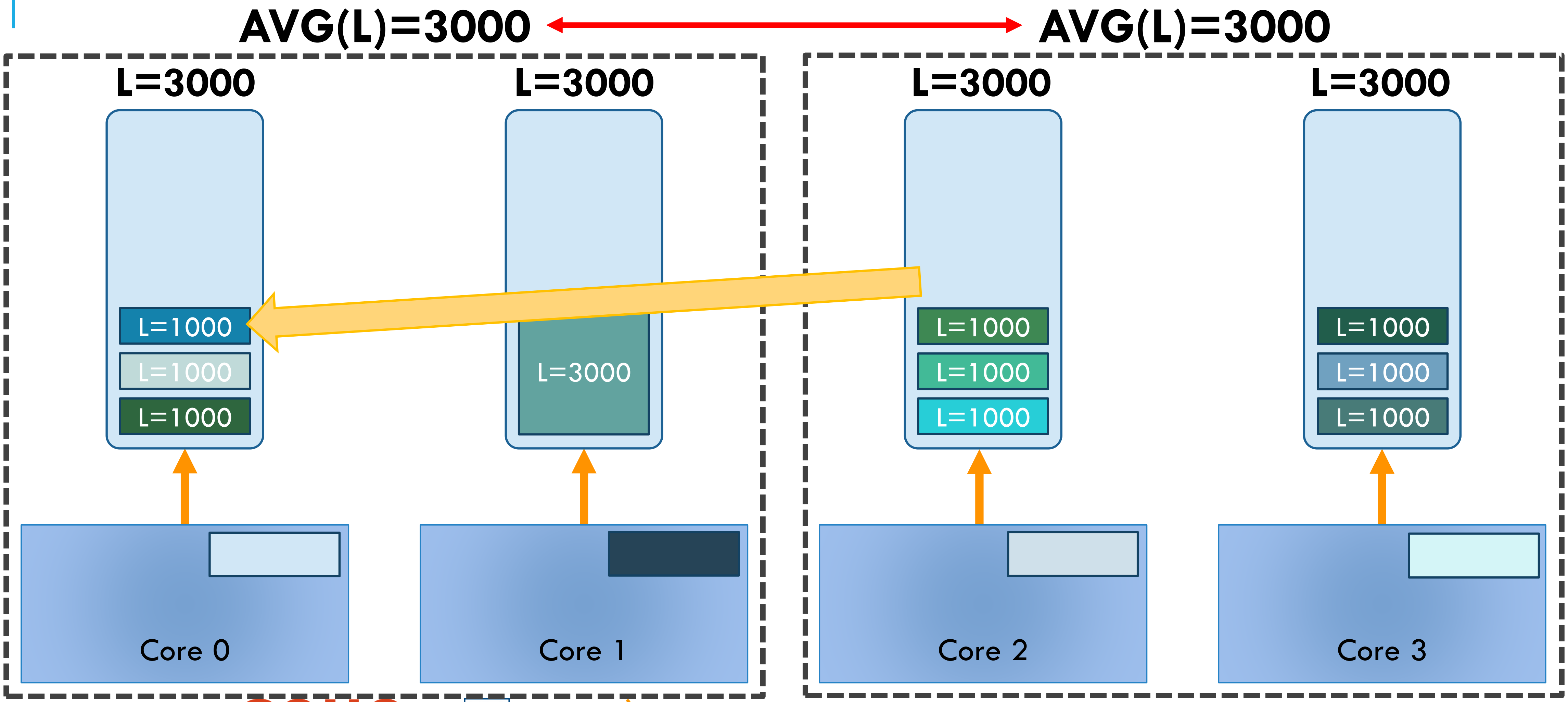
CFS: BALANCING THE LOAD



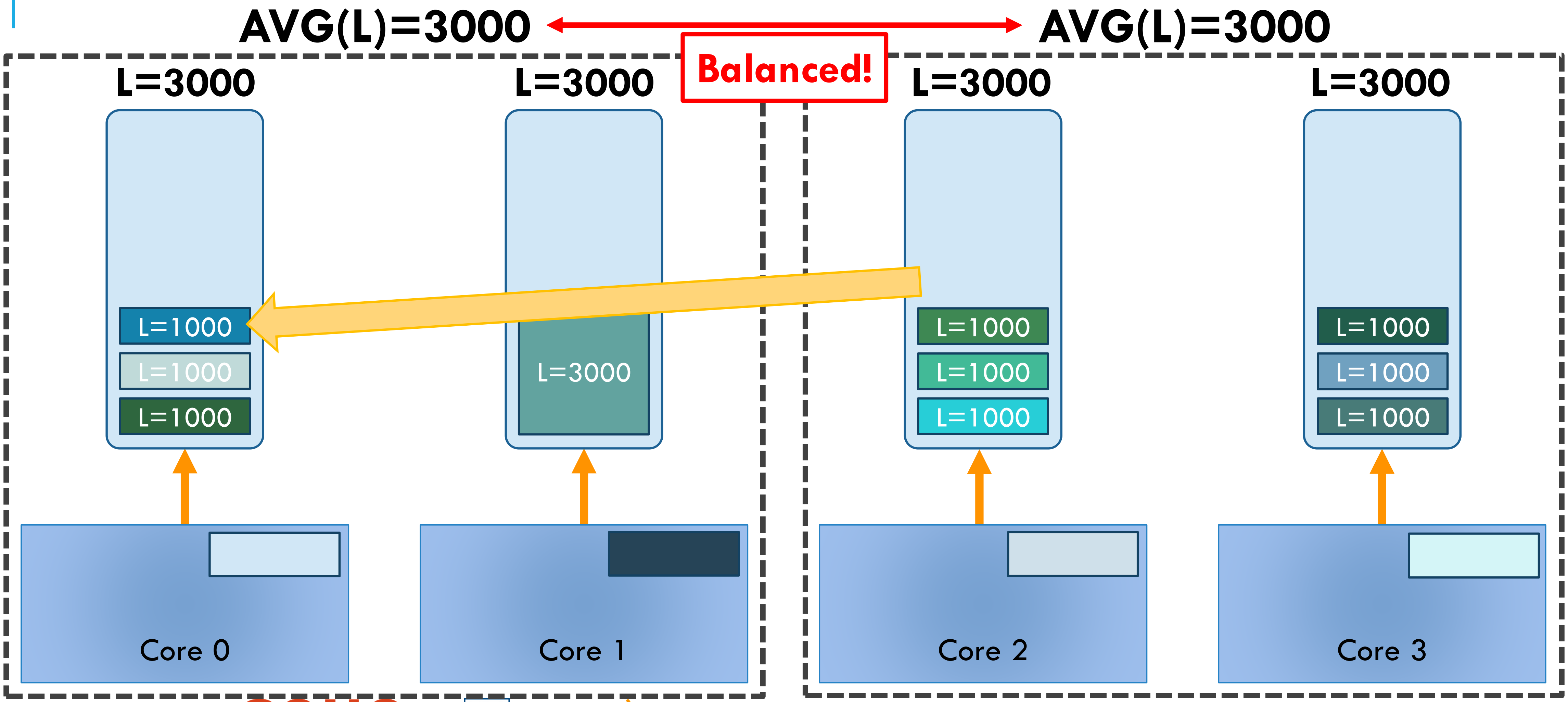
CFS: BALANCING THE LOAD



CFS: BALANCING THE LOAD



CFS: BALANCING THE LOAD



CFS: BALANCING THE LOAD

- Load calculations are actually more complicated, use more heuristics

CFS: BALANCING THE LOAD

- Load calculations are actually more complicated, use more heuristics
- **One of them aims to increase fairness between "sessions"**

CFS: BALANCING THE LOAD

- Load calculations are actually more complicated, use more heuristics
- **One of them aims to increase fairness between "sessions"**
  - **Idea:** ensure a `tty` cannot eat up all resources by spawning many threads

CFS: BALANCING THE LOAD

- Load calculations are actually more complicated, use more heuristics
- **One of them aims to increase fairness between "sessions"**
  - **Idea:** ensure a `tty` cannot eat up all resources by spawning many threads

L=1000

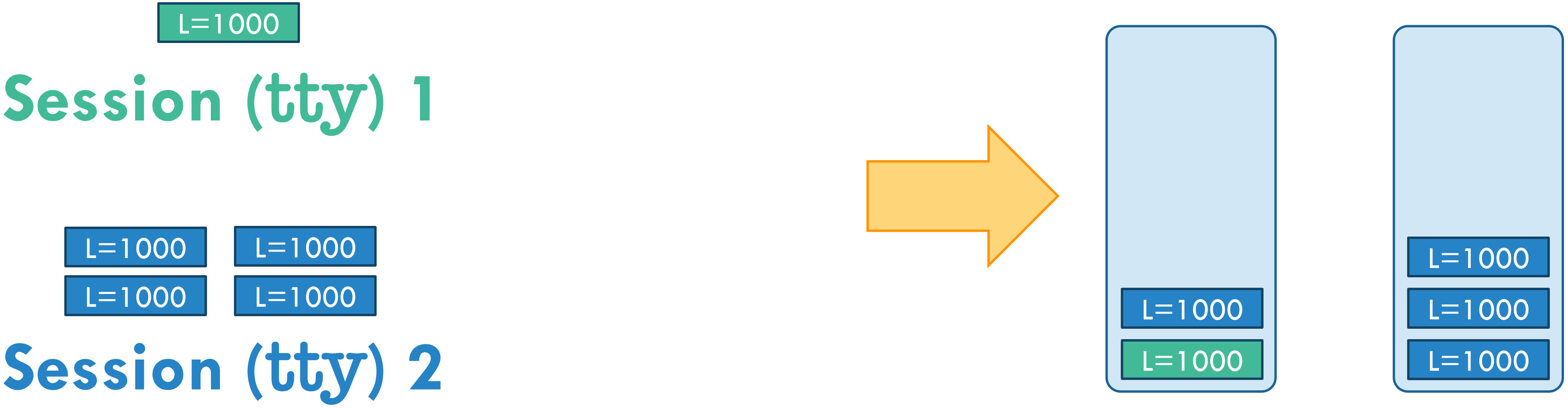
Session (tty) 1

L=1000 L=1000
L=1000 L=1000

Session (tty) 2

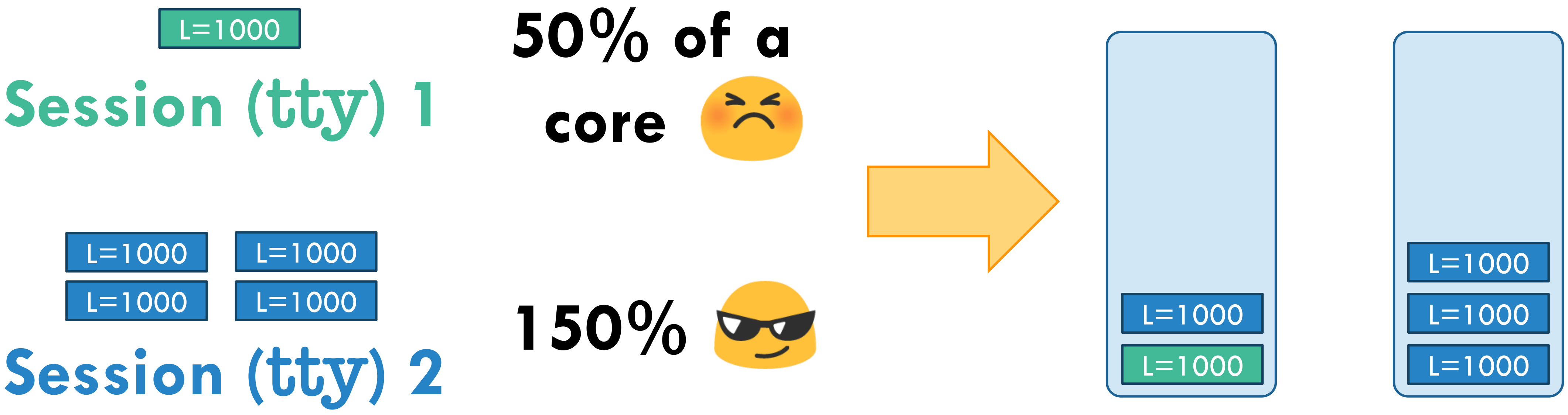
CFS: BALANCING THE LOAD

- Load calculations are actually more complicated, use more heuristics
- **One of them aims to increase fairness between "sessions"**
  - **Idea:** ensure a `tty` cannot eat up all resources by spawning many threads



CFS: BALANCING THE LOAD

- Load calculations are actually more complicated, use more heuristics
- **One of them aims to increase fairness between "sessions"**
  - **Idea:** ensure a `tty` cannot eat up all resources by spawning many threads



CFS: BALANCING THE LOAD

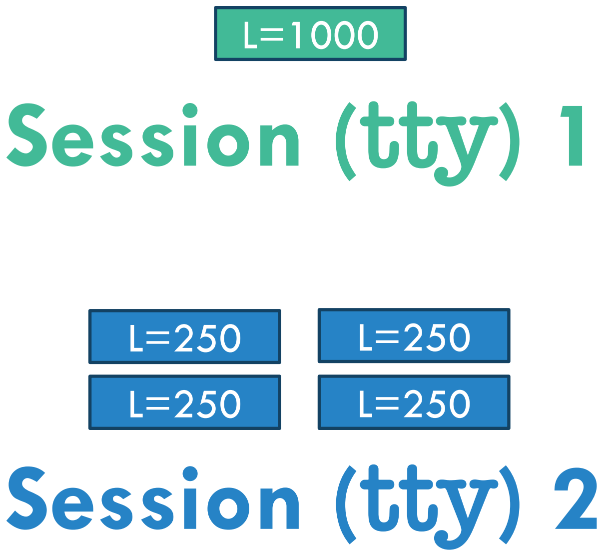
- Load calculations are actually more complicated, use more heuristics
- **One of them aims to increase fairness between "sessions"**
  - **Idea:** ensure a `tty` cannot eat up all resources by spawning many threads

CFS: BALANCING THE LOAD

- Load calculations are actually more complicated, use more heuristics
- **One of them aims to increase fairness between "sessions"**
  - **Solution:** divide the load of a task by the number of threads in its `tty`!

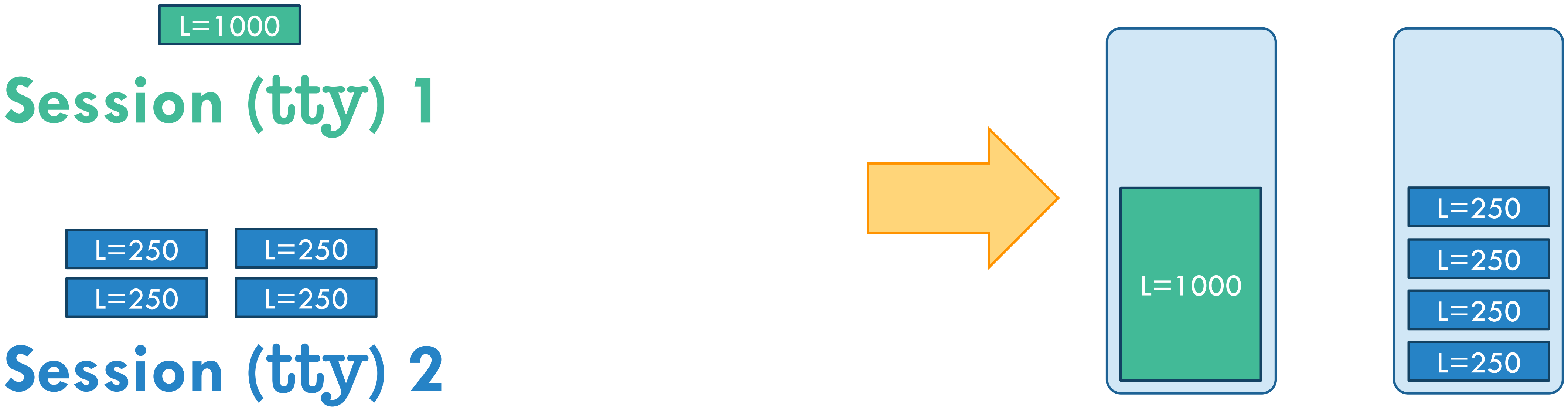
CFS: BALANCING THE LOAD

- Load calculations are actually more complicated, use more heuristics
- **One of them aims to increase fairness between "sessions"**
  - **Solution:** divide the load of a task by the number of threads in its tty!



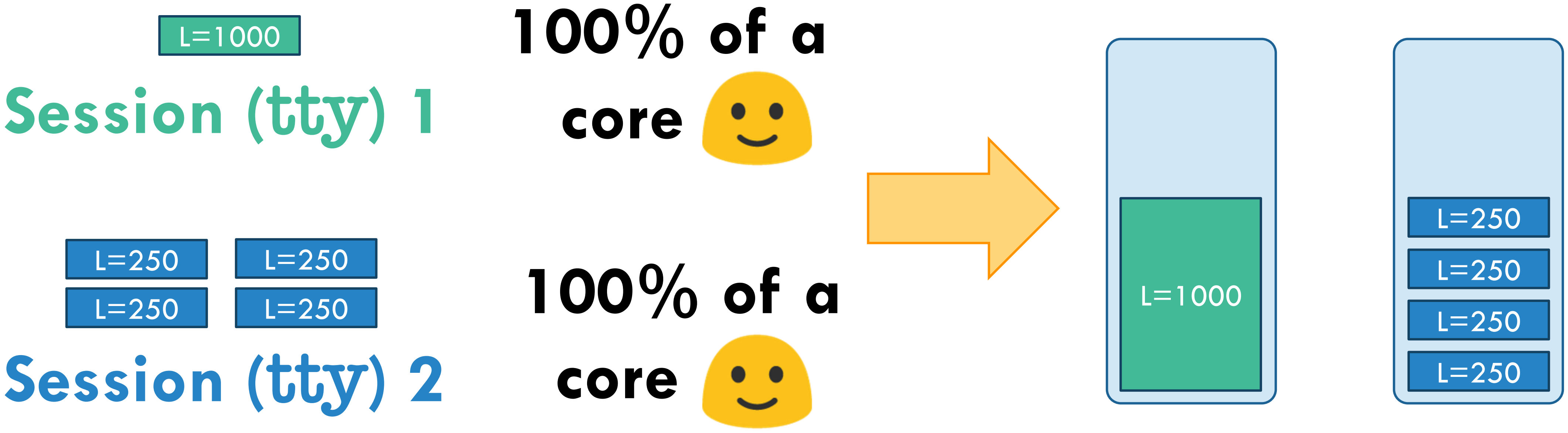
CFS: BALANCING THE LOAD

- Load calculations are actually more complicated, use more heuristics
- **One of them aims to increase fairness between "sessions"**
  - **Solution:** divide the load of a task by the number of threads in its tty!



CFS: BALANCING THE LOAD

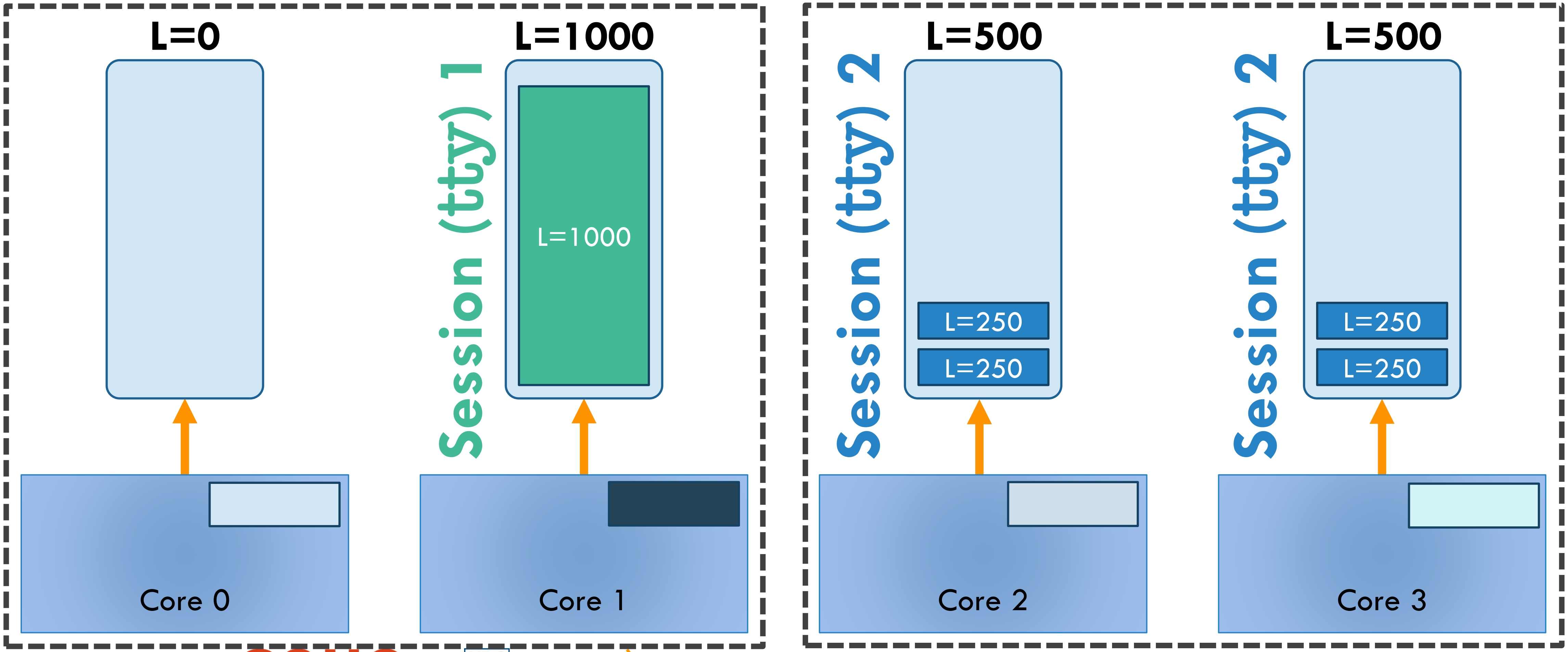
- Load calculations are actually more complicated, use more heuristics
- **One of them aims to increase fairness between "sessions"**
  - **Solution:** divide the load of a task by the number of threads in its tty!



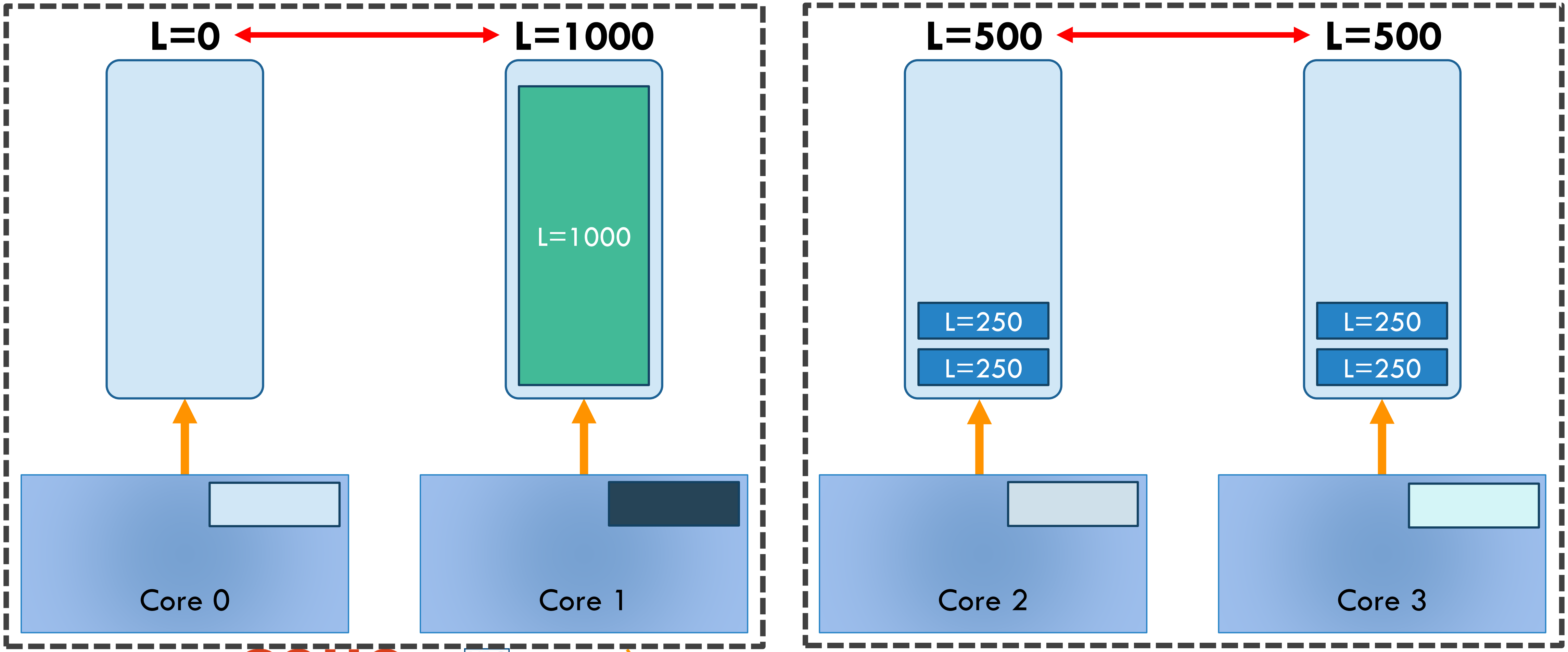
CFS: BALANCING THE LOAD

- Load calculations are actually more complicated, use more heuristics
- **One of them aims to increase fairness between "sessions"**
  - **Solution:** divide the load of a task by the number of threads in its tty!

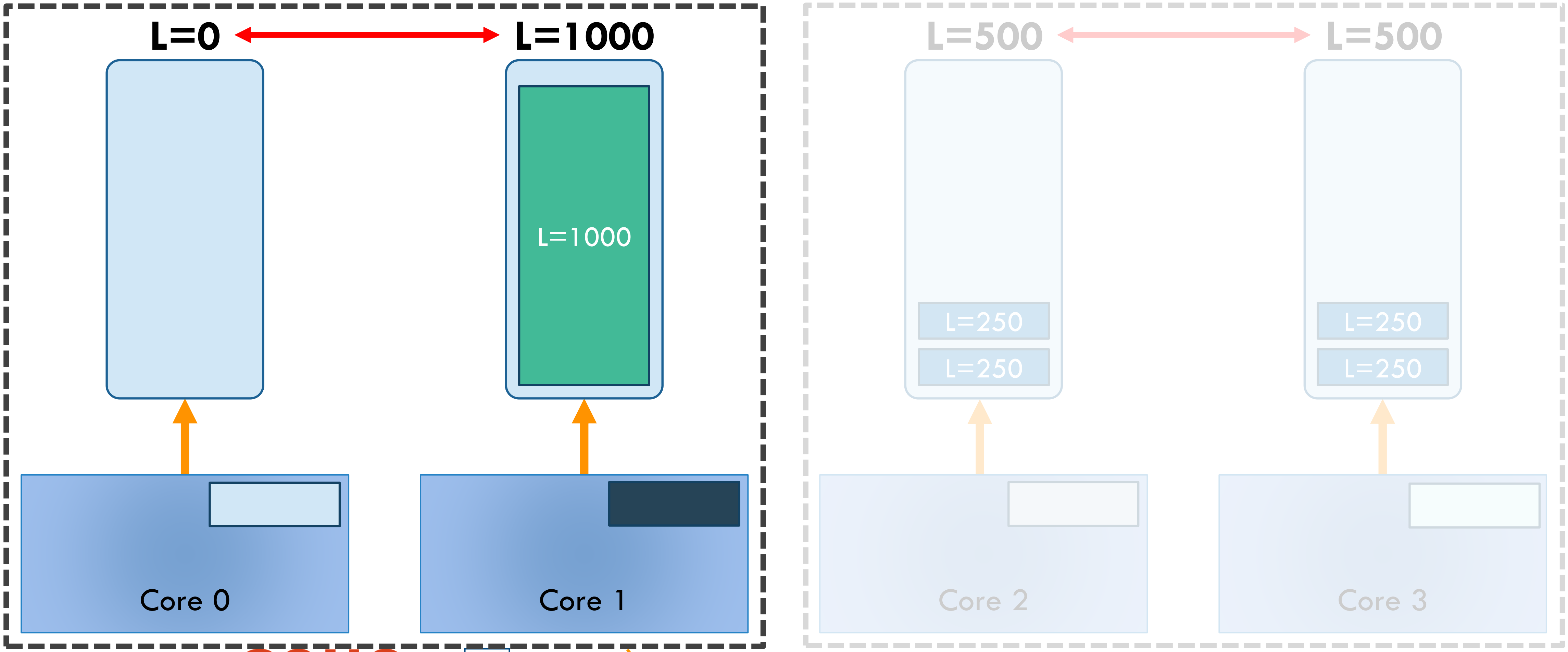
CFS: BALANCING THE LOAD: BUG #1



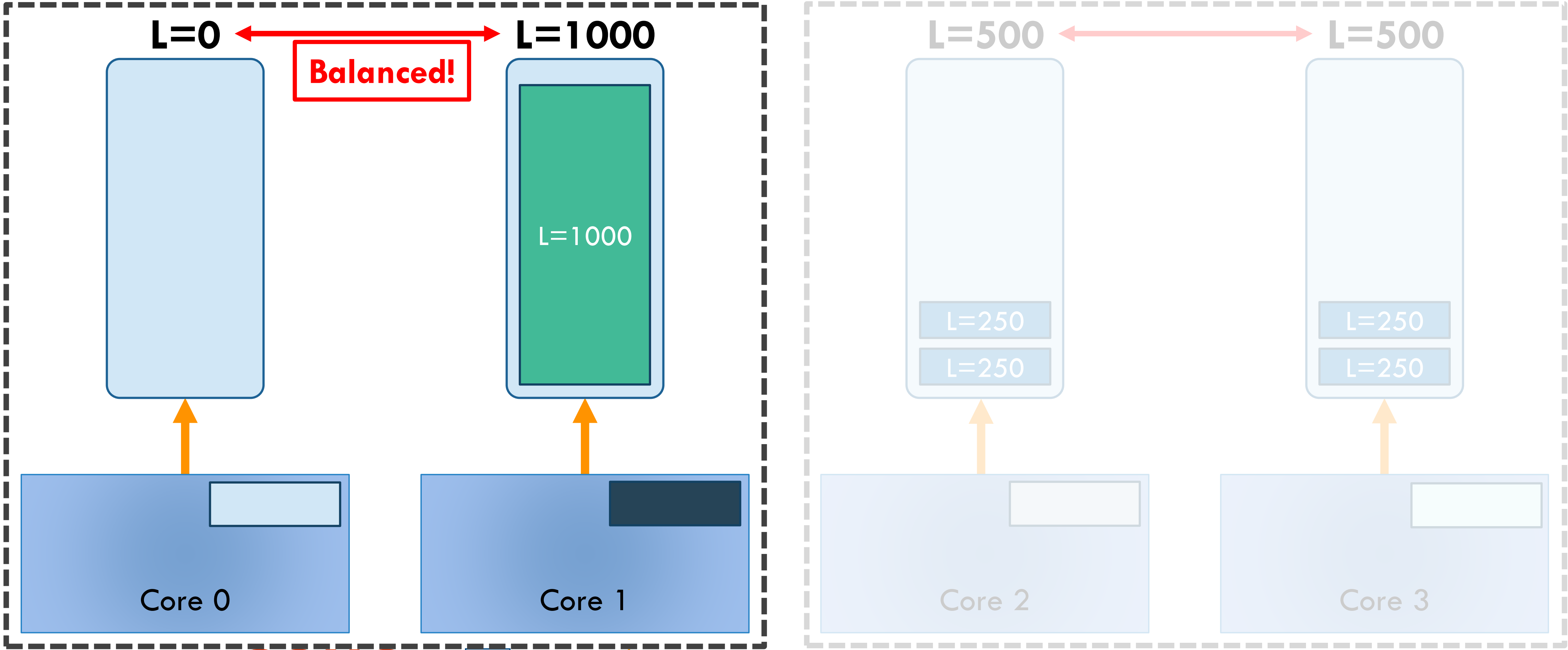
CFS: BALANCING THE LOAD: BUG #1



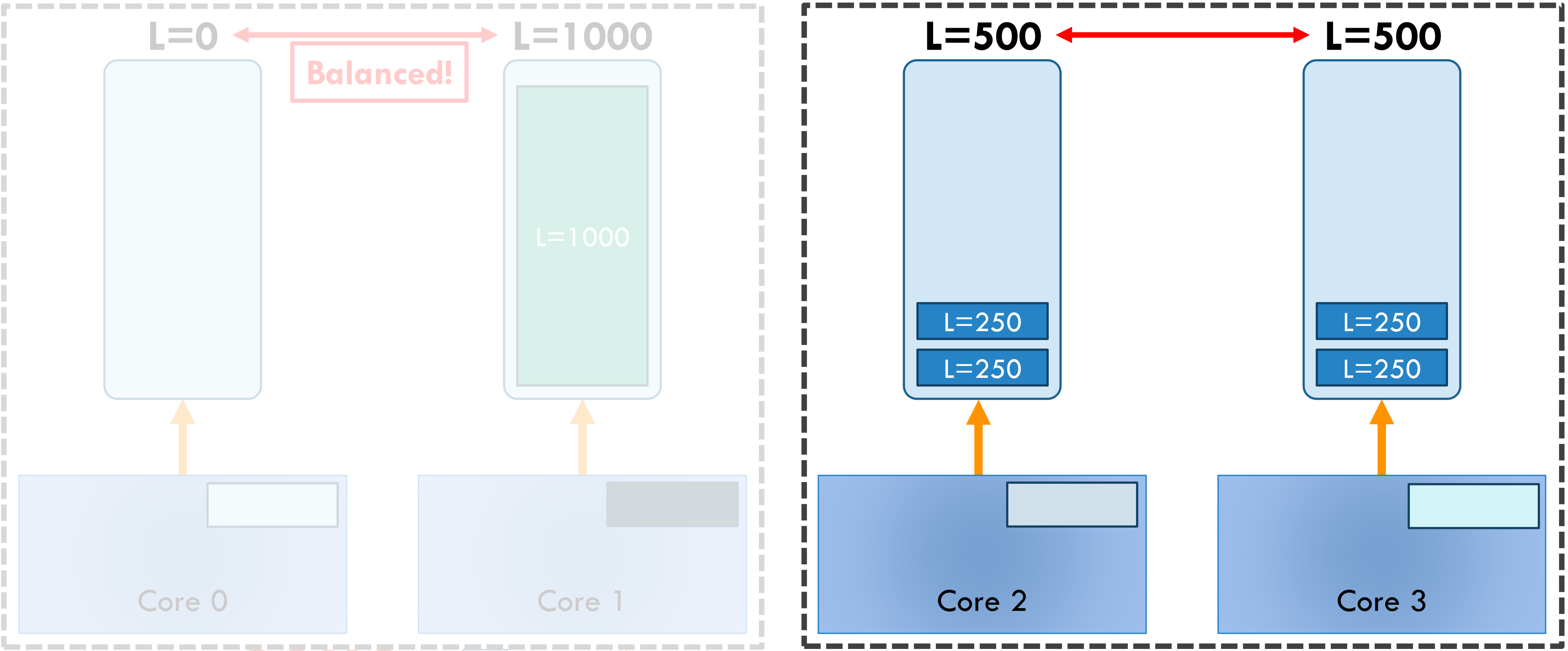
CFS: BALANCING THE LOAD: BUG #1



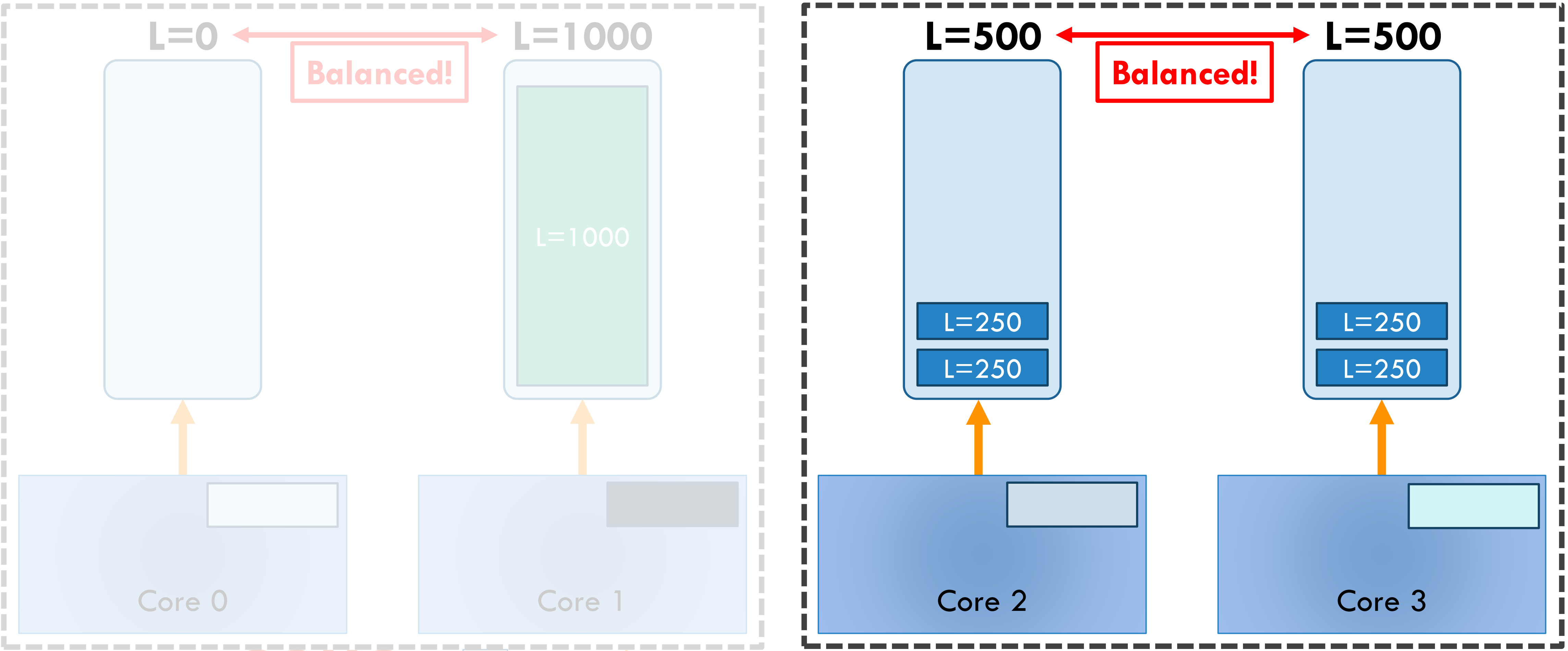
CFS: BALANCING THE LOAD: BUG #1



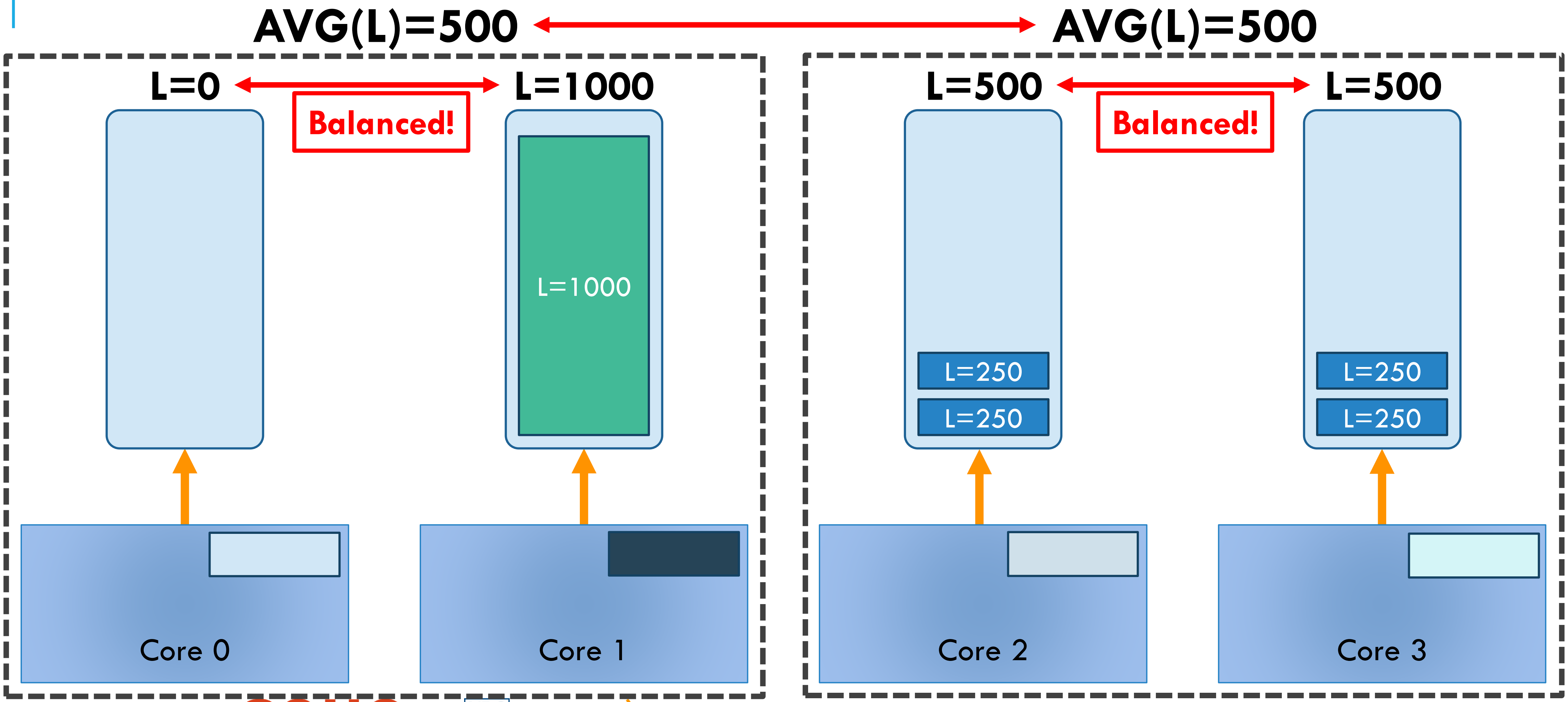
CFS: BALANCING THE LOAD: BUG #1



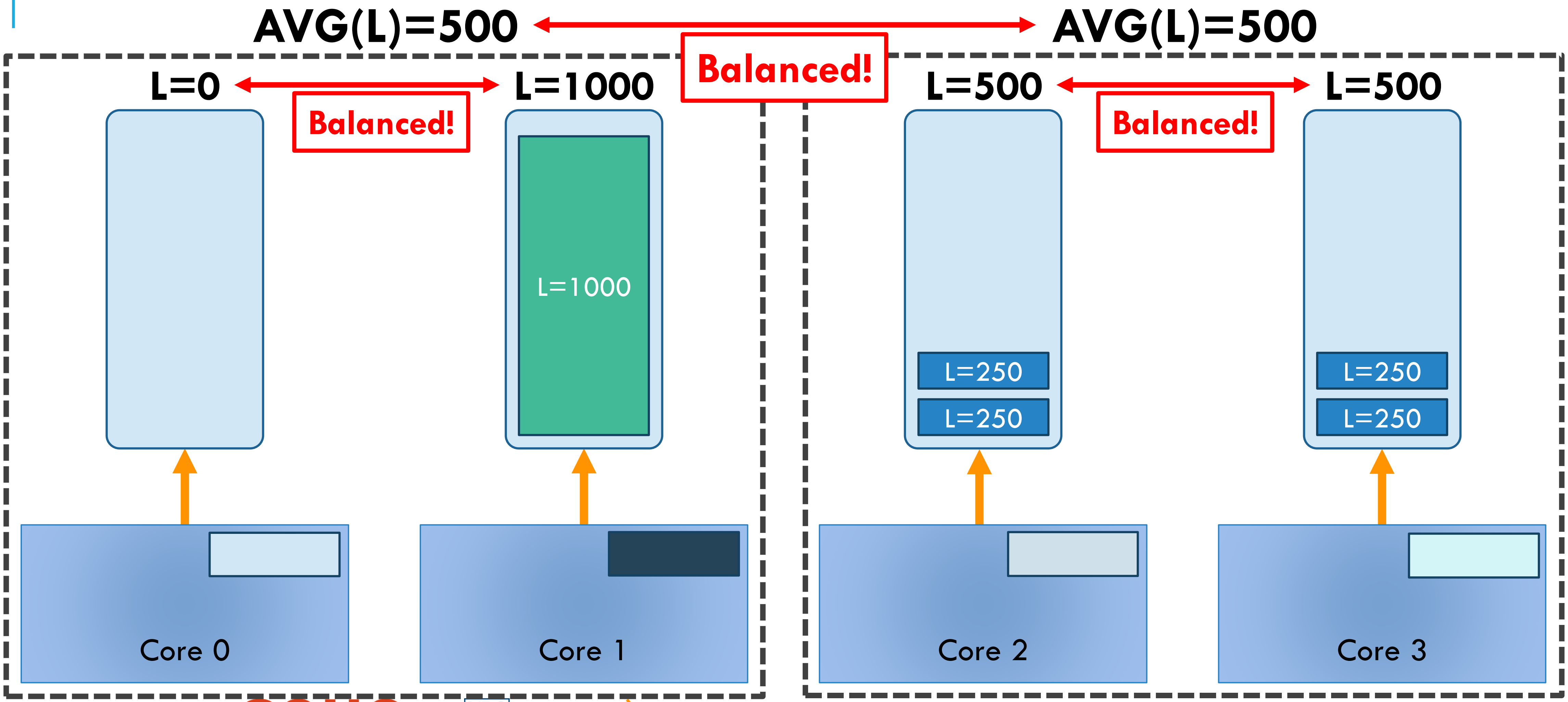
CFS: BALANCING THE LOAD: BUG #1



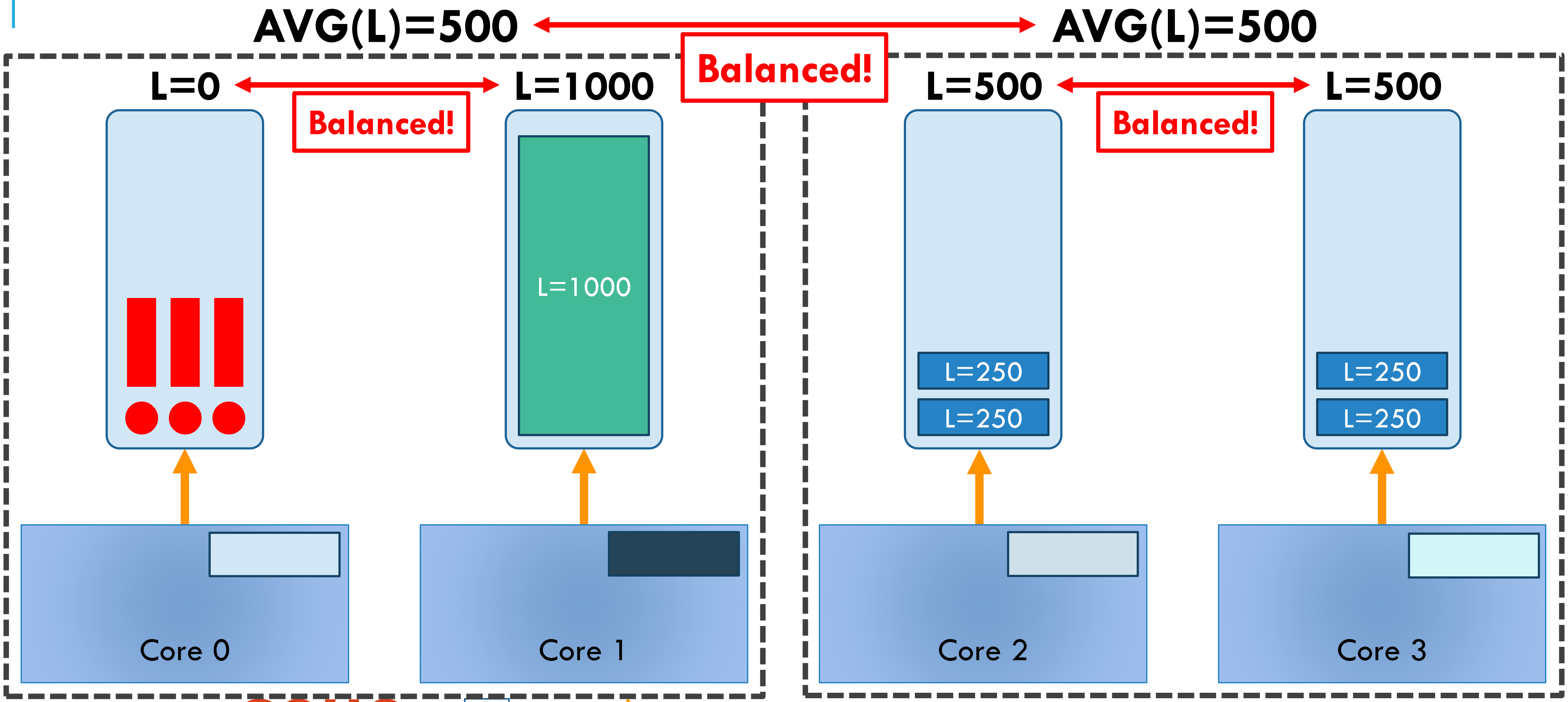
CFS: BALANCING THE LOAD: BUG #1



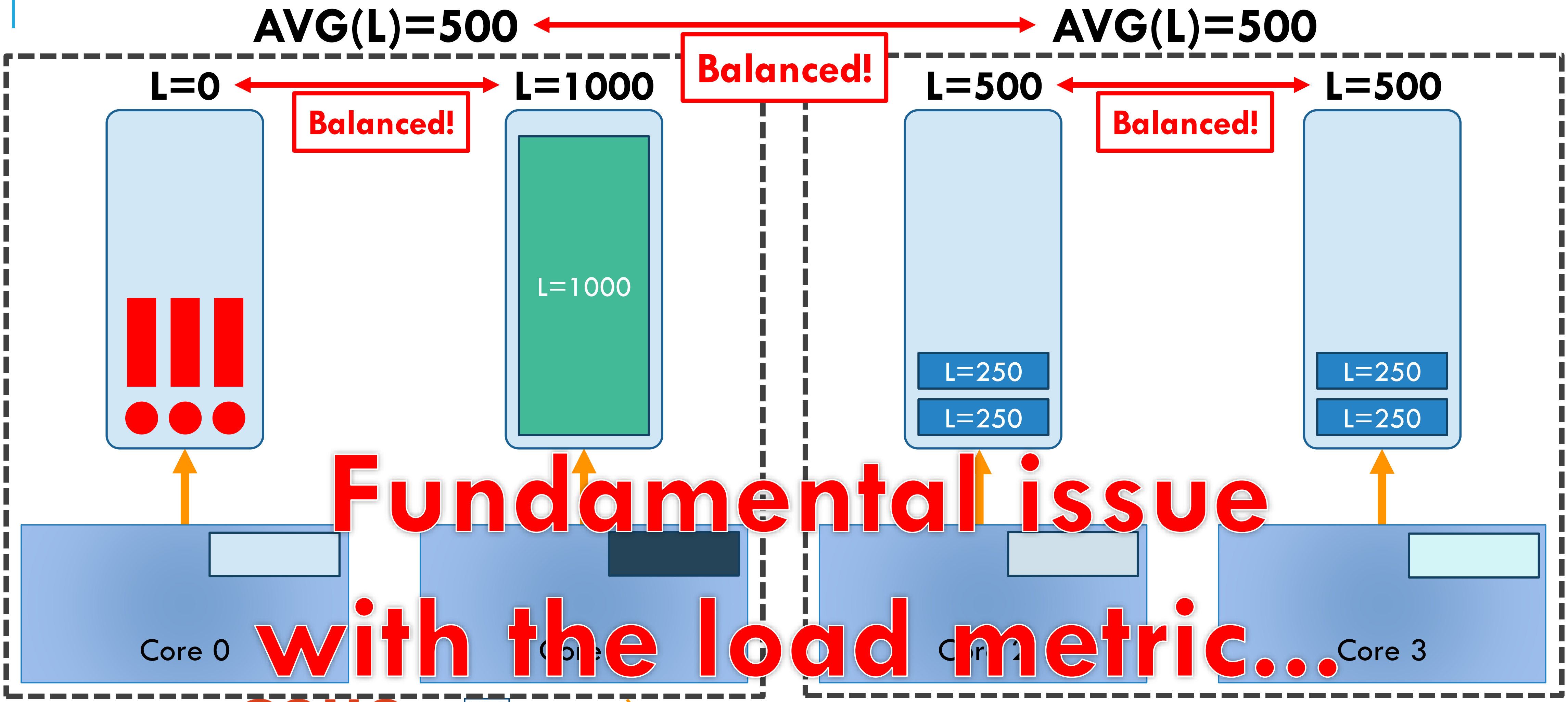
CFS: BALANCING THE LOAD: BUG #1



CFS: BALANCING THE LOAD: BUG #1

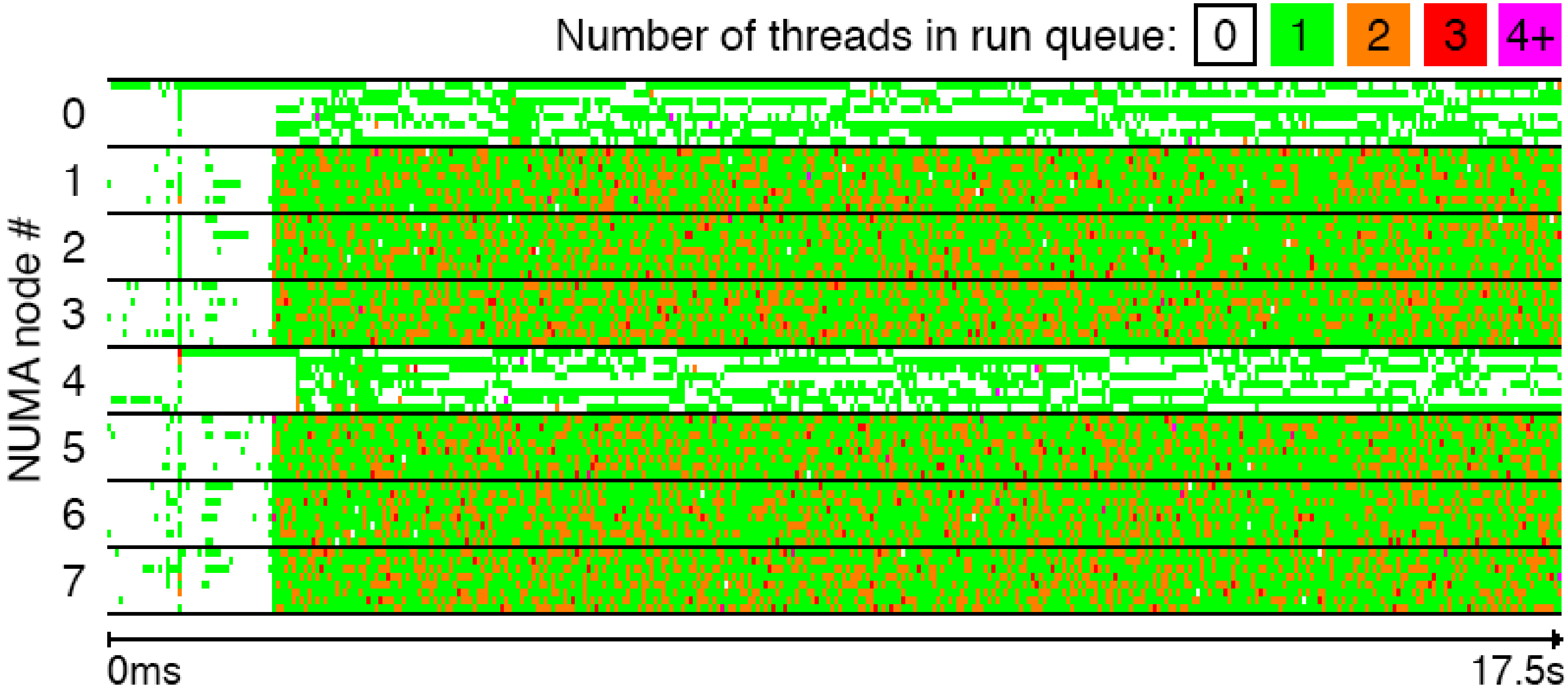


CFS: BALANCING THE LOAD: BUG #1



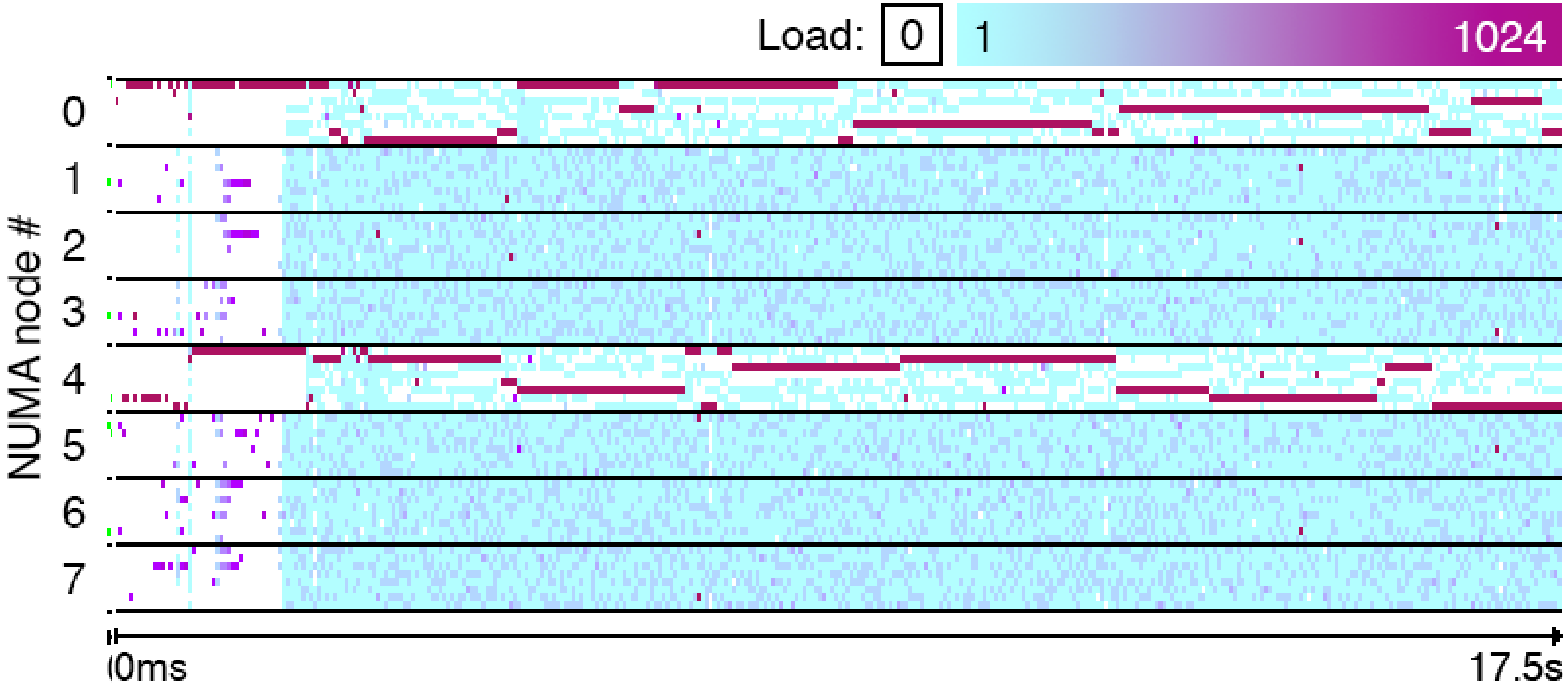
CFS: BALANCING THE LOAD: BUG #1

- **This was our bug!**



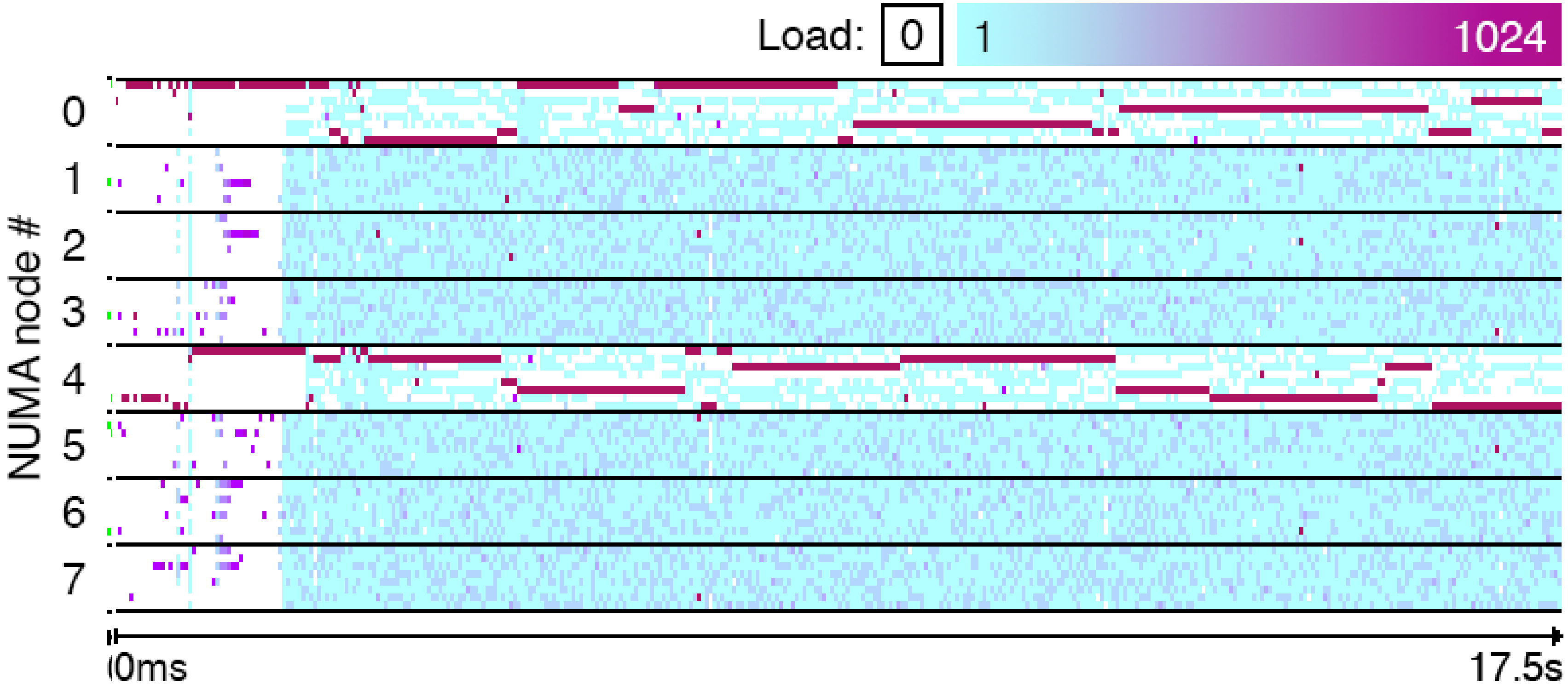
CFS: BALANCING THE LOAD: BUG #1

- **This was our bug!**



CFS: BALANCING THE LOAD: BUG #1

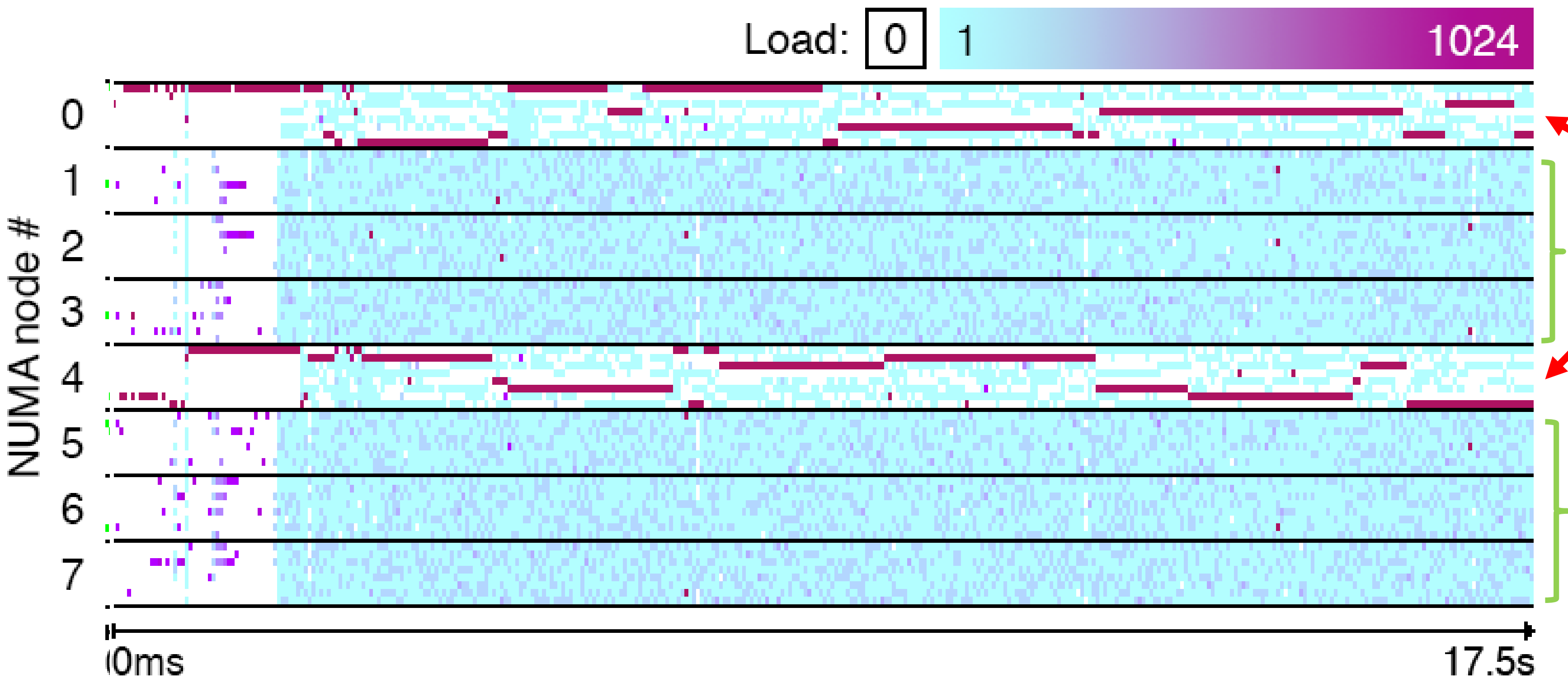
- **This was our bug!**



Load 1 = avg(R thread with high load + a few make threads with low load)

CFS: BALANCING THE LOAD: BUG #1

- **This was our bug!**

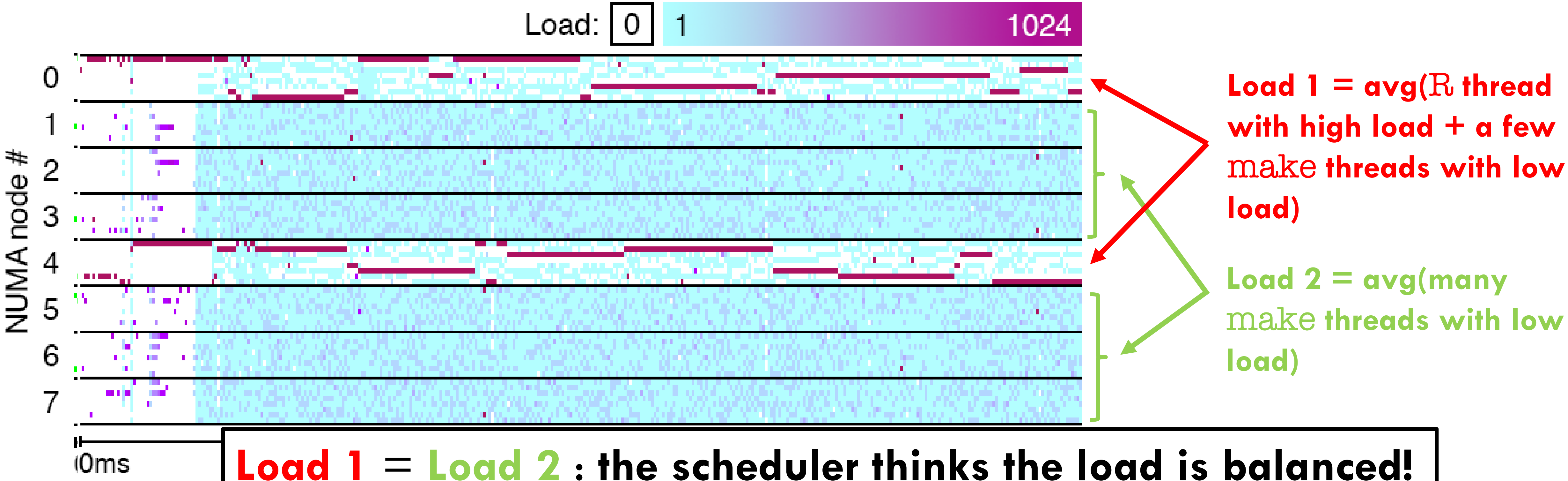


Load 1 = avg(R thread with high load + a few make threads with low load)

Load 2 = avg(many make threads with low load)

CFS: BALANCING THE LOAD: BUG #1

- **This was our bug!**



Load 1 = Load 2 : the scheduler thinks the load is balanced!

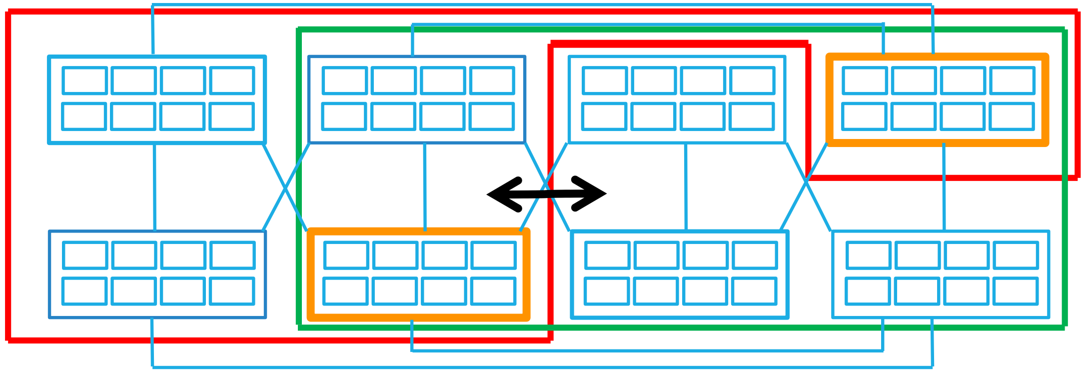
MORE BUGS: THE HIERARCHY

- **We saw load balancing hierarchical:** cores, pairs of cores, dies, CPUs, NUMA nodes…

MORE BUGS: THE HIERARCHY

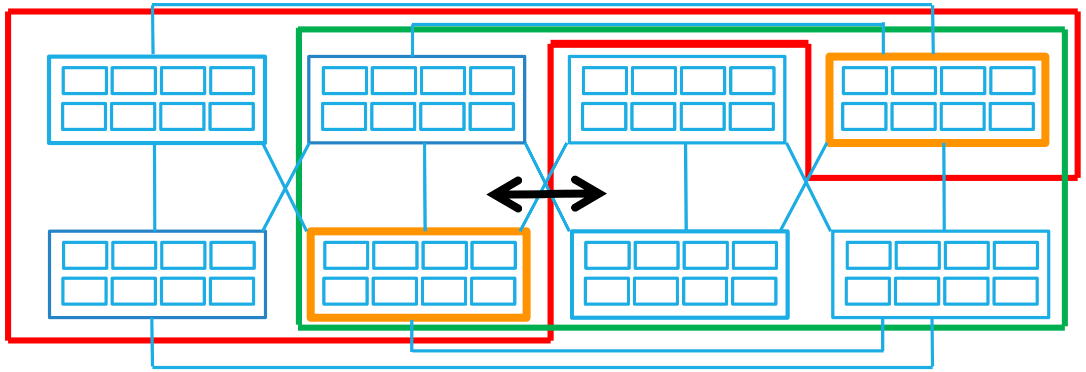
- **We saw load balancing hierarchical:** cores, pairs of cores, dies, CPUs, NUMA nodes…
- **Bug #2: on complex machines, hierarchy built incorrectly!**

MORE BUGS: THE HIERARCHY

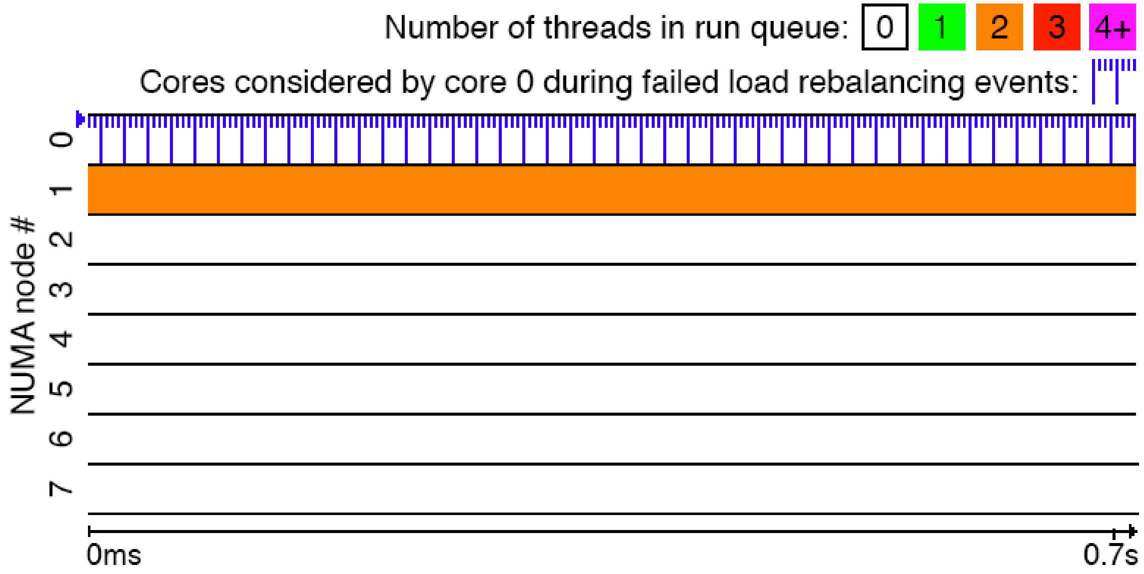


- **We saw load balancing hierarchical:** cores, pairs of cores, dies, CPUs, NUMA nodes…
- **Bug #2: on complex machines, hierarchy built incorrectly!**
  - **Intuition:** at the last level, groups in the hierarchy “not disjoint”

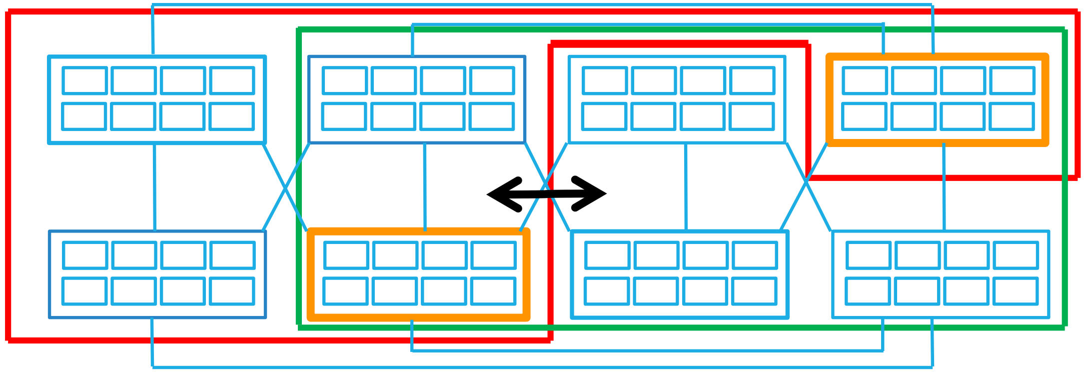
MORE BUGS: THE HIERARCHY



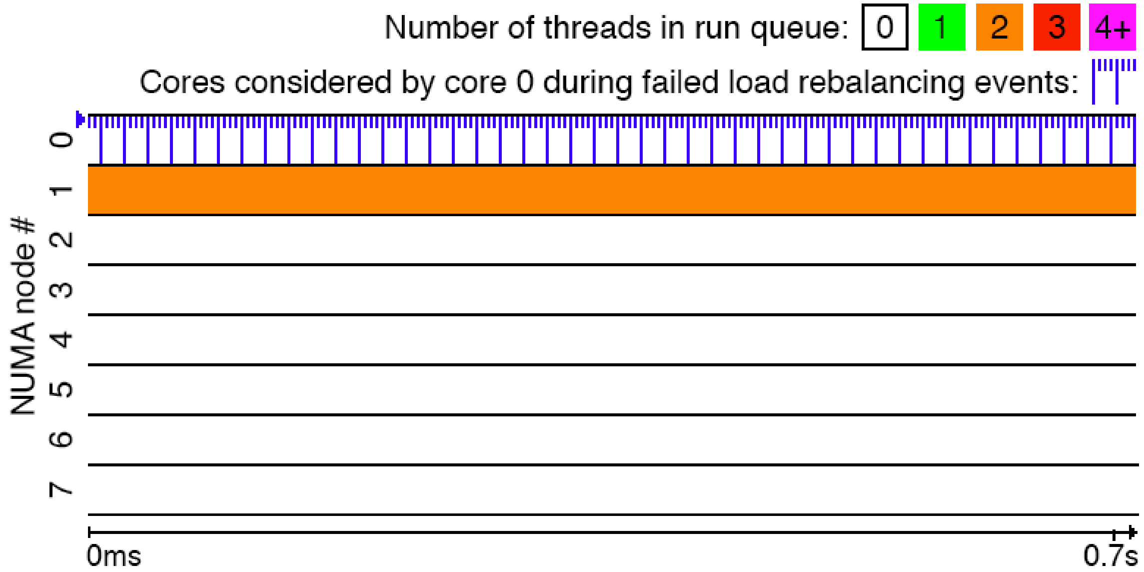
- **We saw load balancing hierarchical:** cores, pairs of cores, dies, CPUs, NUMA nodes…
- **Bug #2: on complex machines, hierarchy built incorrectly!**
  - **Intuition:** at the last level, groups in the hierarchy "not disjoint"
  - **Can break load balancing:** whole application running on a single node!



MORE BUGS: THE HIERARCHY



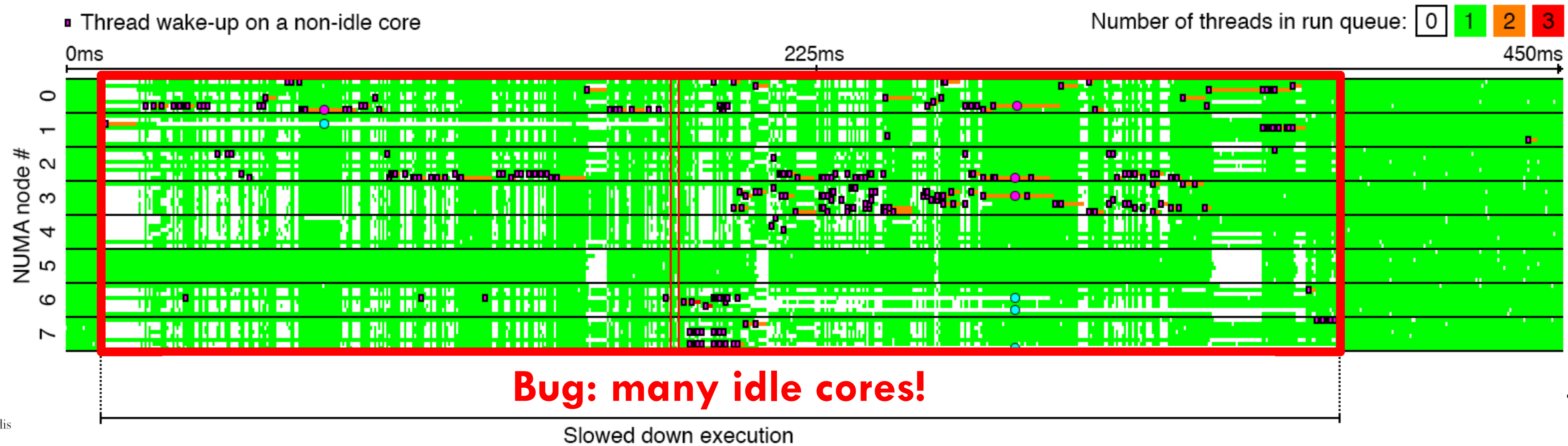
- **We saw load balancing hierarchical:** cores, pairs of cores, dies, CPUs, NUMA nodes…
- **Bug #2: on complex machines, hierarchy built incorrectly!**
  - **Intuition:** at the last level, groups in the hierarchy “not disjoint”
  - **Can break load balancing:** whole application running on a single node!



- **Bug #3: disabling/reenabling a core breaks the hierarchy completely**

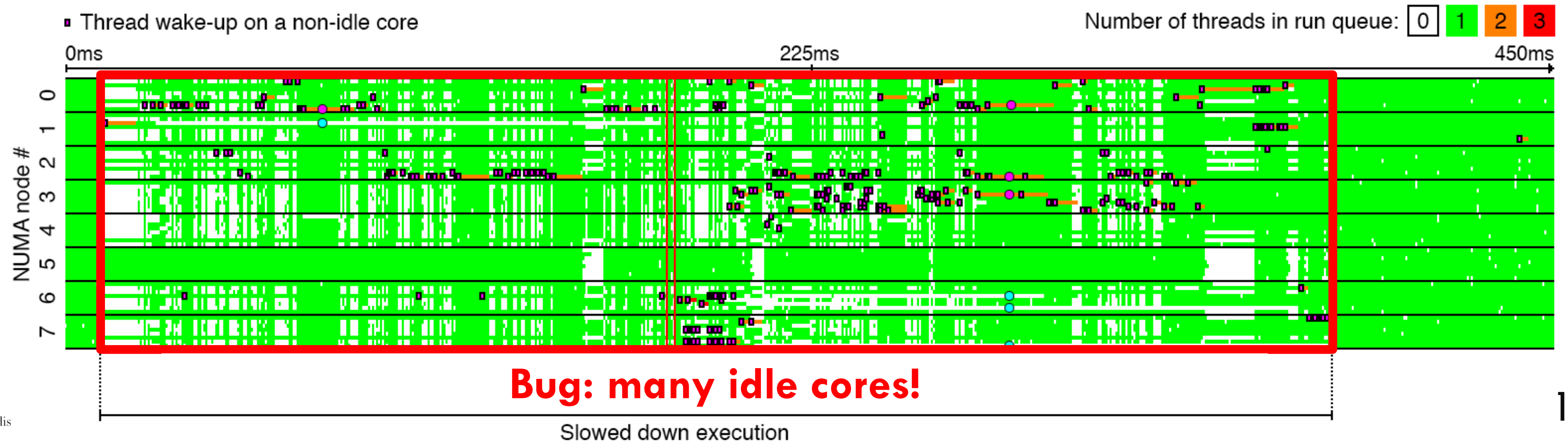
MORE BUGS: WAKEUPS

- **Bug #4: slow phases with idle cores with popular commercial database + TPC-H**



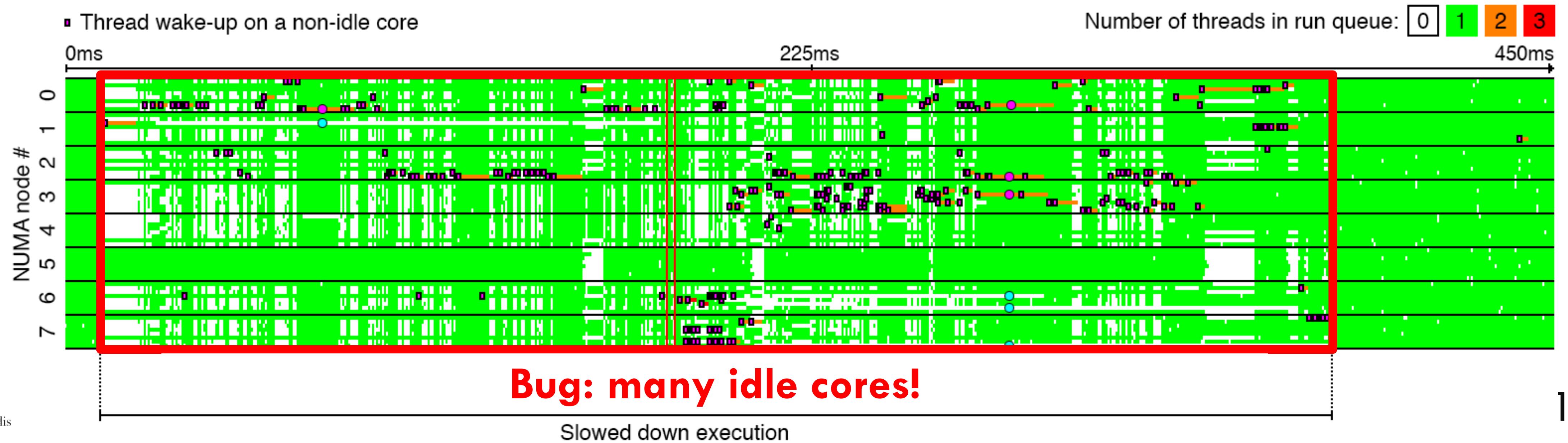
MORE BUGS: WAKEUPS

- **Bug #4: slow phases with idle cores with popular commercial database + TPC-H**
  - In addition to periodic load balancing, threads pick where they wake up



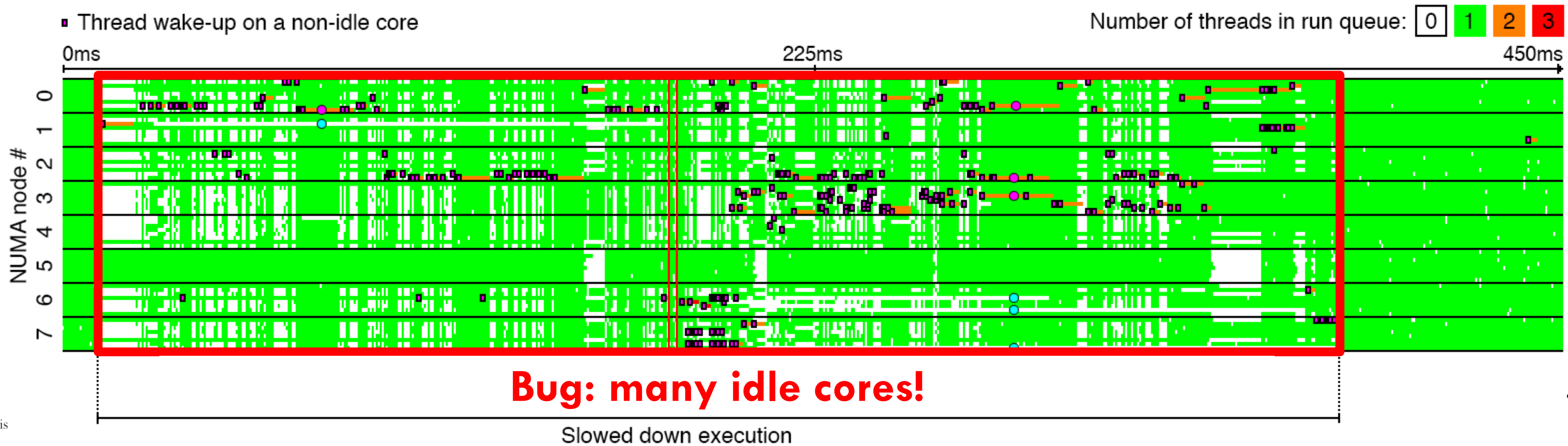
MORE BUGS: WAKEUPS

- **Bug #4: slow phases with idle cores with popular commercial database + TPC-H**
  - In addition to periodic load balancing, threads pick where they wake up
  - Only local CPU cores considered for wakeup due to locality "optimization"



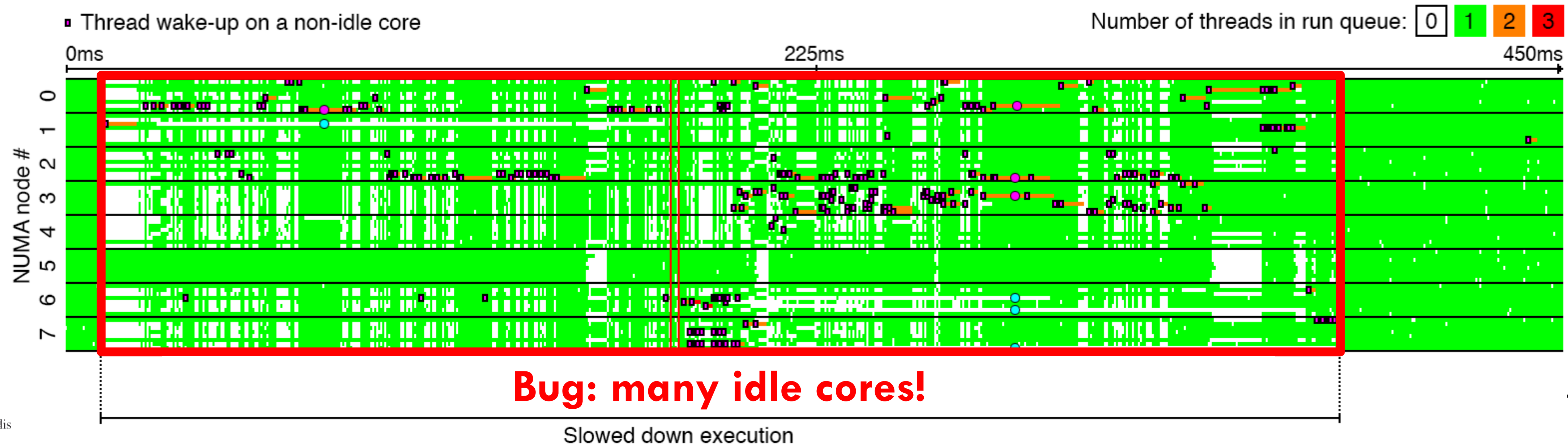
MORE BUGS: WAKEUPS

- **Bug #4: slow phases with idle cores with popular commercial database + TPC-H**
  - In addition to periodic load balancing, threads pick where they wake up
  - Only local CPU cores considered for wakeup due to locality "optimization"
  - **Intuition:** periodic load balancing global, wakeup balancing local



MORE BUGS: WAKEUPS

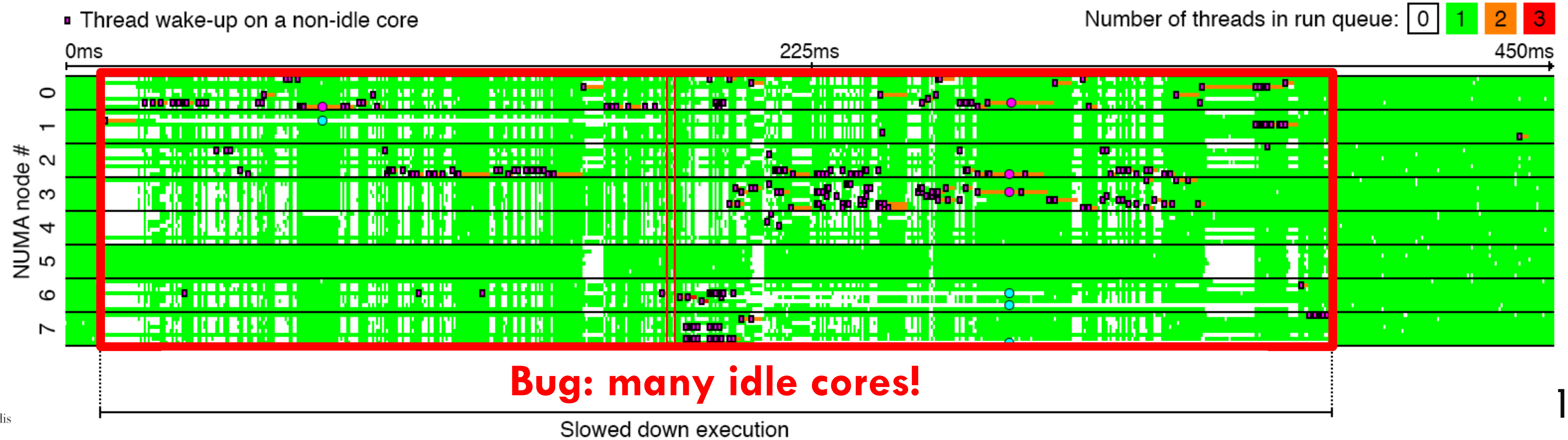
- **Bug #4: slow phases with idle cores with popular commercial database + TPC-H**
  - In addition to periodic load balancing, threads pick where they wake up
  - Only local CPU cores considered for wakeup due to locality "optimization"
  - **Intuition:** periodic load balancing global, wakeup balancing local
    - **One makes mistakes the other cannot fix!**



MORE BUGS: WAKEUPS

Performance degradation: 13-24%!

- **Bug #4: slow phases with idle cores with popular commercial database + TPC-H**
  - In addition to periodic load balancing, threads pick where they wake up
  - Only local CPU cores considered for wakeup due to locality "optimization"
  - **Intuition:** periodic load balancing global, wakeup balancing local
    - **One makes mistakes the other cannot fix!**



DISCUSSION: HOW DID WE COME TO THIS?

- Scheduling (as in dividing CPU cycles among theads) often thought to be a solved problem

DISCUSSION: HOW DID WE COME TO THIS?

- Scheduling (as in dividing CPU cycles among theads) often thought to be a solved problem
- **To recap, on Linux, CFS works like this:**
  - It periodically balances, using a metric named ***load*,**

DISCUSSION: HOW DID WE COME TO THIS?

- Scheduling (as in dividing CPU cycles among theads) often thought to be a solved problem
- **To recap, on Linux, CFS works like this:**
  - It periodically balances, using a metric named ***load,***
  - threads among groups of cores in a ***hierarchy***.

DISCUSSION: HOW DID WE COME TO THIS?

- Scheduling (as in dividing CPU cycles among theads) often thought to be a solved problem
- **To recap, on Linux, CFS works like this:**
  - It periodically balances, using a metric named ***load,***
  - threads among groups of cores in a ***hierarchy***.
  - In addition to this, threads ***balance the load by selecting core where to wake up***.

DISCUSSION: HOW DID WE COME TO THIS?

- Scheduling (as in dividing CPU cycles among theads) often thought to be a solved problem
- **To recap, on Linux, CFS works like this:**
  - It periodically balances, using a metric named ***load,***
  - ↑ **Fundamental issue here...** *appeared with `tty`-balancing heuristic for multithreaded apps*
  - threads among groups of cores in a ***hierarchy***.

  - In addition to this, threads ***balance the load by selecting core where to wake up***.

DISCUSSION: HOW DID WE COME TO THIS?

- Scheduling (as in dividing CPU cycles among theads) often thought to be a solved problem
- **To recap, on Linux, CFS works like this:**
  - It periodically balances, using a metric named ***load,***

  ↑ **Fundamental issue here...** *appeared with `tty`-balancing heuristic for multithreaded apps*
  - threads among groups of cores in a ***hierarchy***.

  ↑ **Fundamental issue here...** *added with support of complex NUMA hierarchies*
  - In addition to this, threads ***balance the load by selecting core where to wake up***.

DISCUSSION: HOW DID WE COME TO THIS?

- Scheduling (as in dividing CPU cycles among theads) often thought to be a solved problem
- **To recap, on Linux, CFS works like this:**
  - It periodically balances, using a metric named ***load,***
  - ↑ **Fundamental issue here...** *appeared with `tty`-balancing heuristic for multithreaded apps*
  - threads among groups of cores in a ***hierarchy*.**
  - ↑ **Fundamental issue here...** *added with support of complex NUMA hierarchies*
  - In addition to this, threads ***balance the load by selecting core where to wake up*.**
  - ↑ **Fundamental issue here...** *added with locality optimization for multicore architectures*

DISCUSSION: HOW DID WE COME TO THIS?

- Scheduling (as in dividing CPU cycles among theads) often thought to be a solved problem
- **To recap, on Linux, CFS works like this:**
  - It periodically balances, using a metric named ***load***,
  - ↑ **Fundamental issue here...** *appeared with `tty`-balancing heuristic for multithreaded apps*
  - threads among groups of cores in a ***hierarchy***.
  - ↑ **Fundamental issue here...** *added with support of complex NUMA hierarchies*
  - In addition to this, threads ***balance the load by selecting core where to wake up***.
  - ↑ **Fundamental issue here...** *added with locality optimization for multicore architectures*

CFS was simple...

then became complex/broken when needed to support new hardware/uses!

DISCUSSION: WHERE DO WE GO FROM HERE?

- **Linux scheduler keeps evolving, different algorithms, new heuristics…**
  - *Hardware evolves fast, won't get any better!*

DISCUSSION: WHERE DO WE GO FROM HERE?

- **Linux scheduler keeps evolving, different algorithms, new heuristics…**
  - *Hardware evolves fast, won't get any better!*

We *need* a *safe* way to keep up with future hardware/uses!

DISCUSSION: WHERE DO WE GO FROM HERE?

- **Linux scheduler keeps evolving, different algorithms, new heuristics…**
  - *Hardware evolves fast, won't get any better!*

We *need* a *safe* way to keep up with future hardware/uses!

- **Code testing**
  - No clear fault (no crash, no deadlock, etc.), existing tools don't target these bugs

DISCUSSION: WHERE DO WE GO FROM HERE?

- **Linux scheduler keeps evolving, different algorithms, new heuristics…**
  - *Hardware evolves fast, won't get any better!*

We *need* a *safe* way to keep up with future hardware/uses!

- **Code testing**
  - No clear fault (no crash, no deadlock, etc.), existing tools don't target these bugs
- **Performance regression**
  - Usually done with 1 app on a machine to avoid interactions: insufficient coverage

DISCUSSION: WHERE DO WE GO FROM HERE?

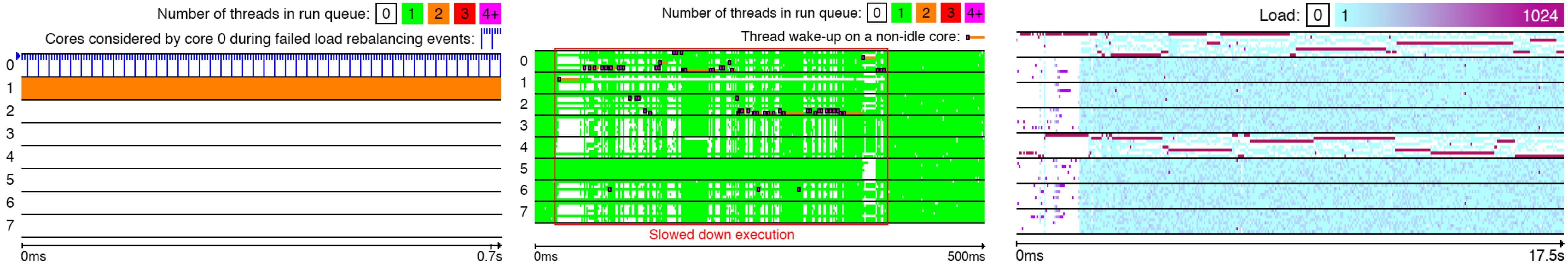
- **Linux scheduler keeps evolving, different algorithms, new heuristics…**
  - *Hardware evolves fast, won't get any better!*

We *need* a *safe* way to keep up with future hardware/uses!

- **Code testing**
  - No clear fault (no crash, no deadlock, etc.), existing tools don't target these bugs
- **Performance regression**
  - Usually done with 1 app on a machine to avoid interactions: insufficient coverage
- **Model checking, formal proofs**
  - Complex, parallel code: so far, nobody knows how to do it…

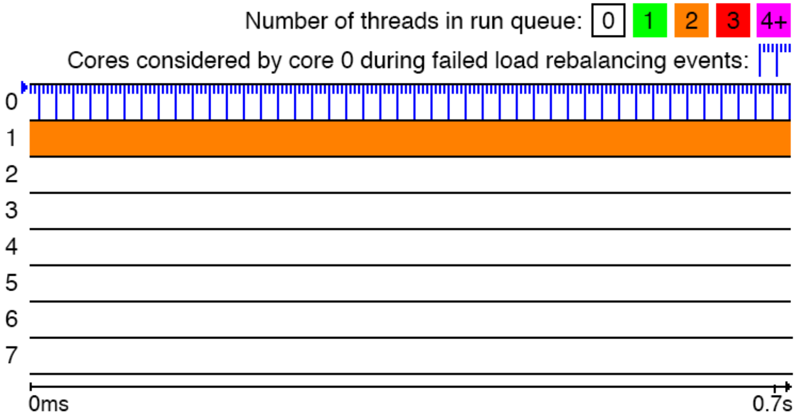
DISCUSSION: WHERE DO WE GO FROM HERE?

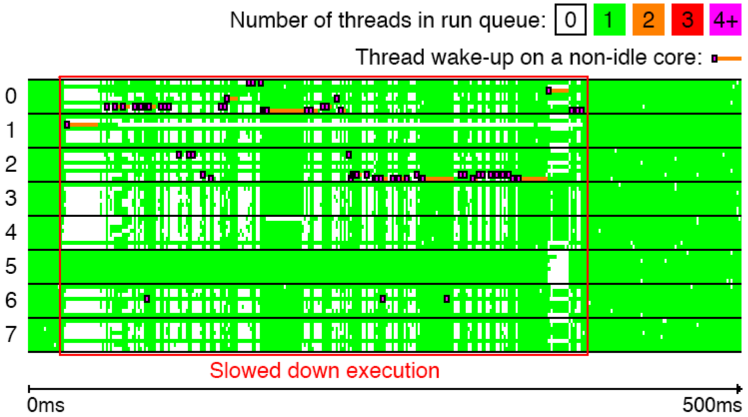
- **What worked for us:** *sanity checker* detects invariant violations to find bugs

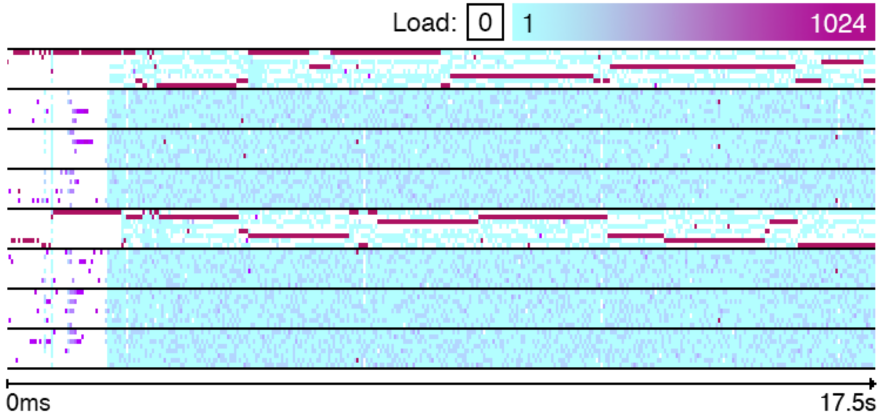


DISCUSSION: WHERE DO WE GO FROM HERE?

- **What worked for us:** *sanity checker* detects invariant violations to find bugs
- **Idea:** detect suspicious situations, monitor them and produce report if they last



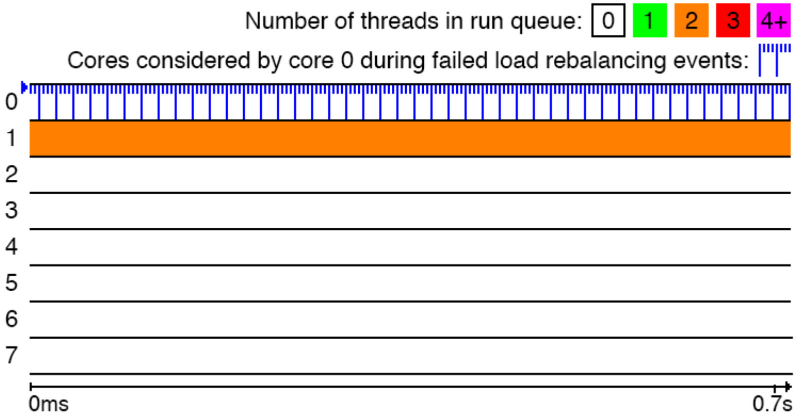


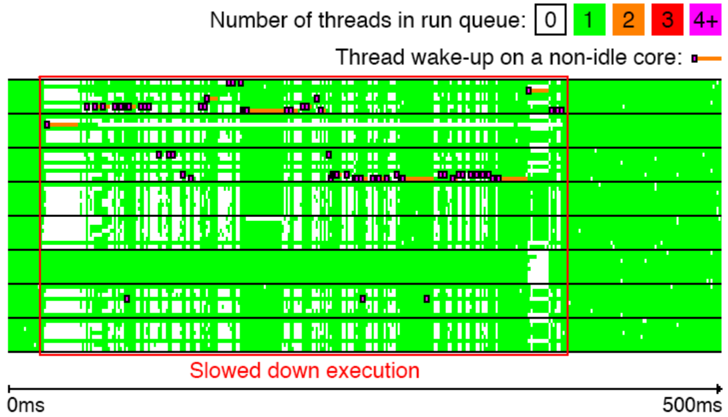


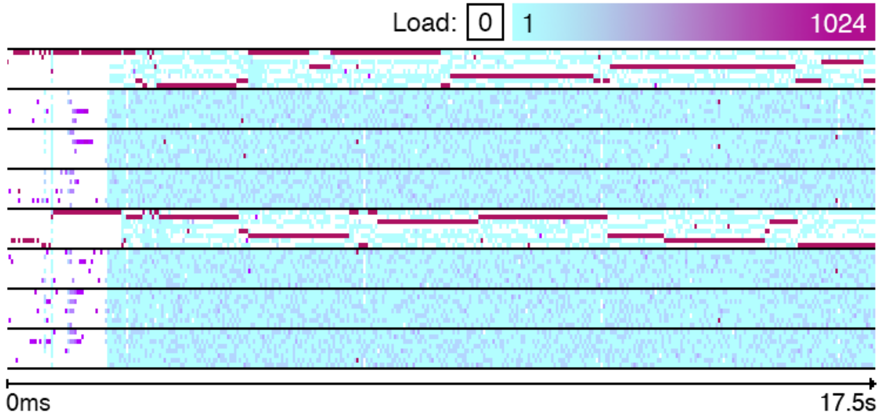
Université Nice Sophia Antipolis | EPFL ÉCOLE POLYTECHNIQUE FÉDÉRALE DE LAUSANNE | COHO DATA | UBC | Grenoble INP Ensimag

DISCUSSION: WHERE DO WE GO FROM HERE?

- **What worked for us:** *sanity checker* detects invariant violations to find bugs
- **Idea:** detect suspicious situations, monitor them and produce report if they last
- ***All bugs presented here detected with sanity checker!***

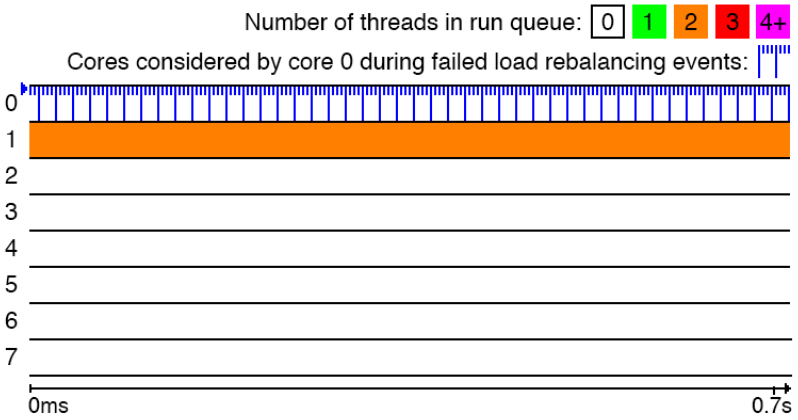


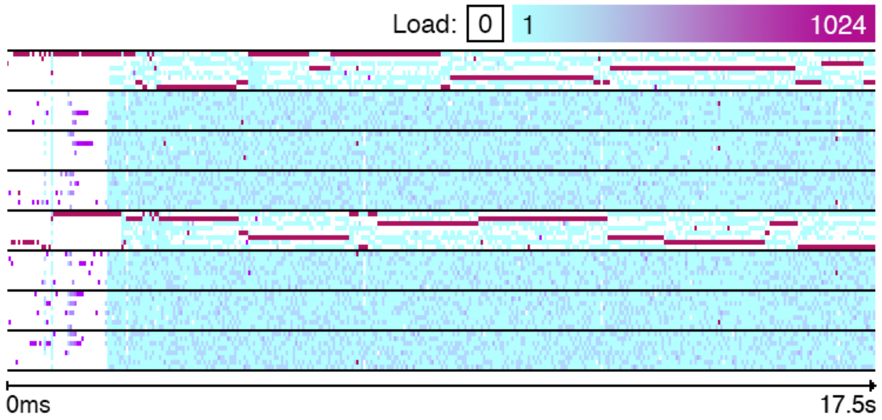




DISCUSSION: WHERE DO WE GO FROM HERE?

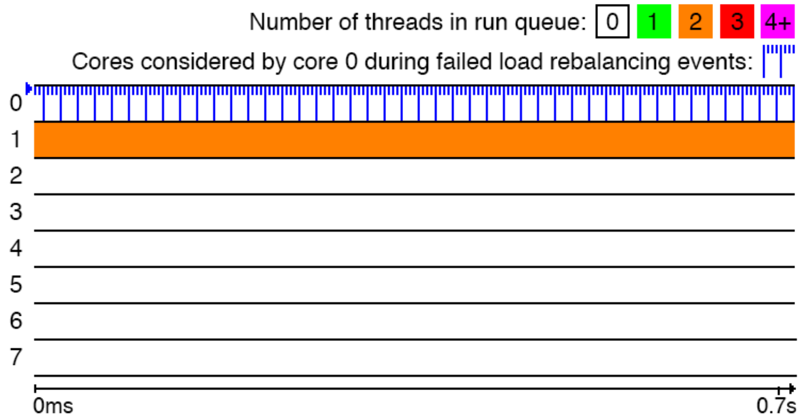
- **What worked for us:** *sanity checker* detects invariant violations to find bugs
- **Idea:** detect suspicious situations, monitor them and produce report if they last
- ***All bugs presented here detected with sanity checker!***
- **Our experience:** exact traces are *necessary* to understand complex scheduling problems

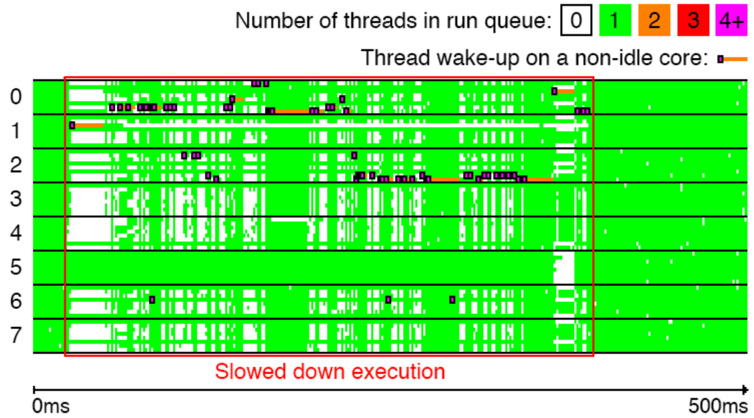


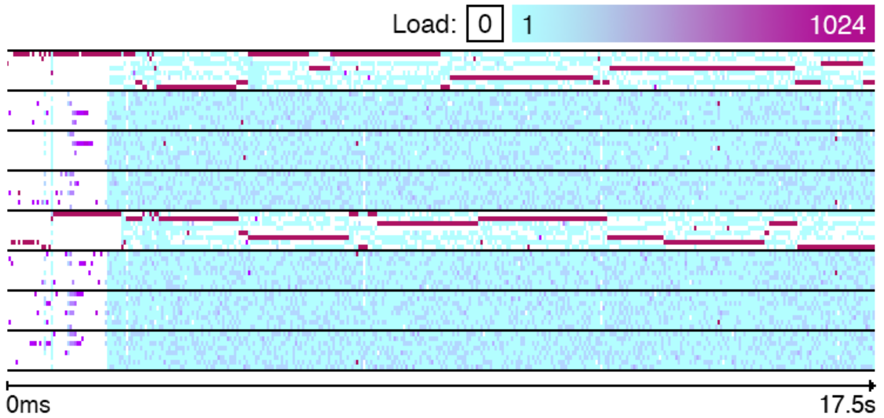


DISCUSSION: WHERE DO WE GO FROM HERE?

- **What worked for us:** *sanity checker* detects invariant violations to find bugs
- **Idea:** detect suspicious situations, monitor them and produce report if they last
- ***All bugs presented here detected with sanity checker!***
- **Our experience:** exact traces are *necessary* to understand complex scheduling problems
- Custom visual tool show all scheduling events / migrations / considered cores / load…







DISCUSSION: FIXING THE SCHEDULER POSSIBLE?

- **Basic fixes for the bugs we analyzed:**
  - **Bug #1:** minimum load instead of average (may be less stable!)
  - **Bugs #2-#3 :** building the hierarchy differently (seems to always work!)
  - **Bug #4:** wake up on cores idle for longest time (may be bad for energy!)

DISCUSSION: FIXING THE SCHEDULER POSSIBLE?

- **Basic fixes for the bugs we analyzed:**
  - **Bug #1:** minimum load instead of average (may be less stable!)
  - **Bugs #2-#3 :** building the hierarchy differently (seems to always work!)
  - **Bug #4:** wake up on cores idle for longest time (may be bad for energy!)
- **Fixes not perfect, hard to ensure they never worsen performance**
  - Linux scheduler too complex, many competing heuristics added empirically!
  - Hard to guess the effect of one change...

DISCUSSION: FIXING THE SCHEDULER POSSIBLE?

- **Basic fixes for the bugs we analyzed:**
  - **Bug #1:** minimum load instead of average (may be less stable!)
  - **Bugs #2-#3 :** building the hierarchy differently (seems to always work!)
  - **Bug #4:** wake up on cores idle for longest time (may be bad for energy!)
- **Fixes not perfect, hard to ensure they never worsen performance**
  - Linux scheduler too complex, many competing heuristics added empirically!
  - Hard to guess the effect of one change...
- **Efficient redesign of the scheduler possible?**
  - We envision scheduler with *isolated* modules each trying to optimize one variable...

DISCUSSION: FIXING THE SCHEDULER POSSIBLE?

- **Basic fixes for the bugs we analyzed:**
  - **Bug #1:** minimum load instead of average (may be less stable!)
  - **Bugs #2-#3 :** building the hierarchy differently (seems to always work!)
  - **Bug #4:** wake up on cores idle for longest time (may be bad for energy!)
- **Fixes not perfect, hard to ensure they never worsen performance**
  - Linux scheduler too complex, many competing heuristics added empirically!
  - Hard to guess the effect of one change...
- **Efficient redesign of the scheduler possible?**
  - We envision scheduler with *isolated* modules each trying to optimize one variable...
  - How do you make them all work together? **Complex, open problem!**

CONCLUSION

- Scheduling (as in dividing CPU cycles among theads) often thought to be a solved problem

CONCLUSION

- Scheduling (as in dividing CPU cycles among theads) often thought to be a solved problem
- **Analysis:** fundamental issues (added incrementally), even basic invariant violated!

CONCLUSION

- Scheduling (as in dividing CPU cycles among theads) often thought to be a solved problem
- **Analysis:** fundamental issues (added incrementally), even basic invariant violated!
- **Proposed pragmatic detection approach (*sanity checker* + traces):** helpful

Université Nice Sophia Antipolis · ÉCOLE POLYTECHNIQUE FÉDÉRALE DE LAUSANNE · COHO DATA · UBC · Grenoble INP Ensimag

CONCLUSION

- Scheduling (as in dividing CPU cycles among theads) often thought to be a solved problem
- **Analysis:** fundamental issues (added incrementally), even basic invariant violated!
- **Proposed pragmatic detection approach (*sanity checker* + traces):** helpful
- **Proposed fixes:** not always satisfactory

CONCLUSION

- Scheduling (as in dividing CPU cycles among theads) often thought to be a solved problem
- **Analysis:** fundamental issues (added incrementally), even basic invariant violated!
- **Proposed pragmatic detection approach (*sanity checker* + traces):** helpful
- **Proposed fixes:** not always satisfactory

Open problem: how do we ensure the scheduler works/evolves correctly ?

New design? New techniques involving testing/performance regression/proofs/...?

CONCLUSION

- Scheduling (as in dividing CPU cycles among theads) often thought to be a solved problem
- **Analysis:** fundamental issues (added incrementally), even basic invariant violated!
- **Proposed pragmatic detection approach (*sanity checker* + traces):** helpful
- **Proposed fixes:** not always satisfactory

Open problem: how do we ensure the scheduler works/evolves correctly ?

New design? New techniques involving testing/performance regression/proofs/...?

Your next paper ☺