RÉPUBLIQUE FRANÇAISE
Liberté
Égalité
Fraternité

Inria

SOUTENANCE D'HABILITATION À DIRIGER DES RECHERCHES

EFFICIENT TASK SCHEDULING, SYNCHRONIZATION, AND GRAPH ANALYTICS ON MULTICORE ARCHITECTURES

Jean-Pierre Lozi
Inria, Paris, France

Overview

- Selected works from ~12 years of research, incl. 5 years in the private sector (Oracle)

Overview

- Selected works from ~12 years of research, incl. 5 years in the private sector (Oracle)
- **Focus:**

Improving the performance of CPU-/memory-intensive applications
such as graph engines on modern hardware

Overview

- Selected works from ~12 years of research, incl. 5 years in the private sector (Oracle)
- **Focus:**

  Improving the performance of CPU-/memory-intensive applications such as graph engines on modern hardware

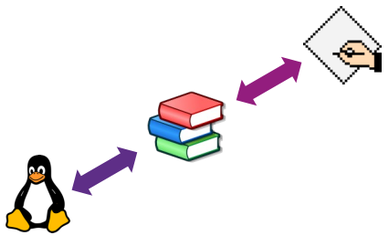
- **The hard truth:** Improving real-world software stacks is messy!

Overview

- Selected works from ~12 years of research, incl. 5 years in the private sector (Oracle)
- **Focus:**

  Improving the performance of CPU-/memory-intensive applications such as graph engines on modern hardware

- **The hard truth:** Improving real-world software stacks is messy!
- **Bottlenecks in all layers:** In the kernel, libraries, and applications

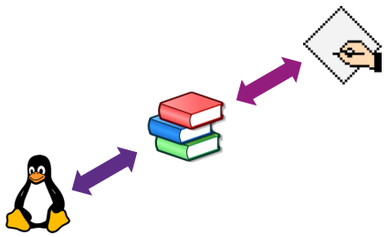


Overview

- Selected works from ~12 years of research, incl. 5 years in the private sector (Oracle)
- **Focus:**

  Improving the performance of CPU-/memory-intensive applications such as graph engines on modern hardware

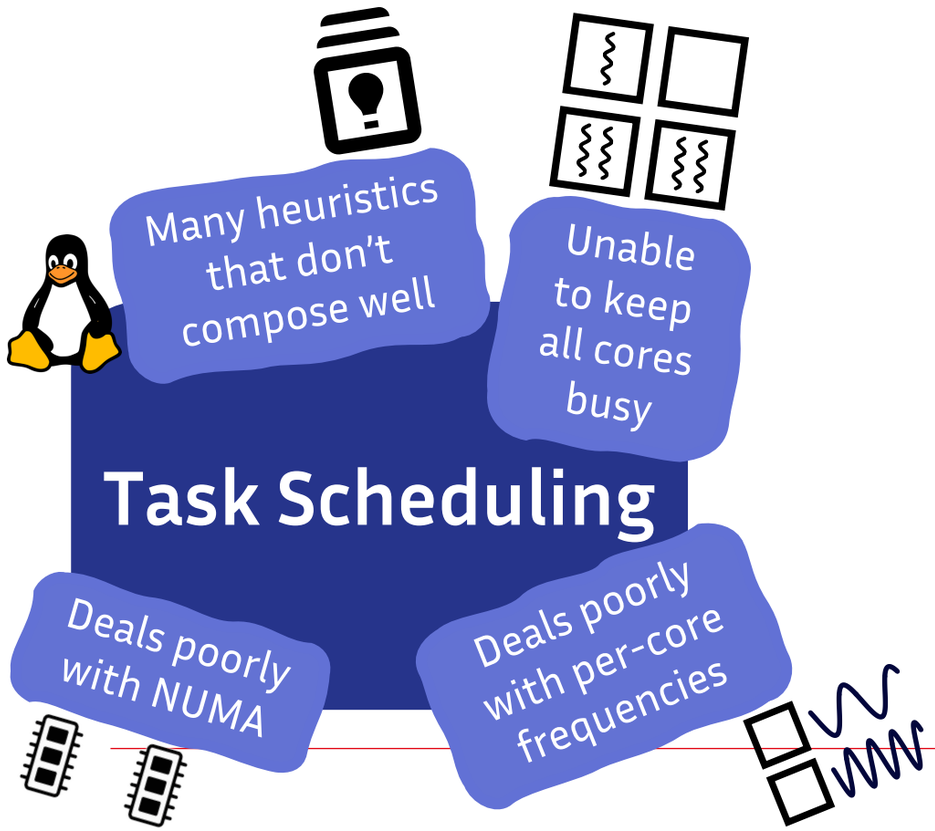
- **The hard truth:** Improving real-world software stacks is messy!
- **Bottlenecks in all layers:** In the kernel, libraries, and applications
- **No silver bullet !**
  - Some optimization techniques shared
  - But custom solutions needed for each bottleneck



Bottlenecks Across the Software Stack

Task Scheduling

Bottlenecks Across the Software Stack

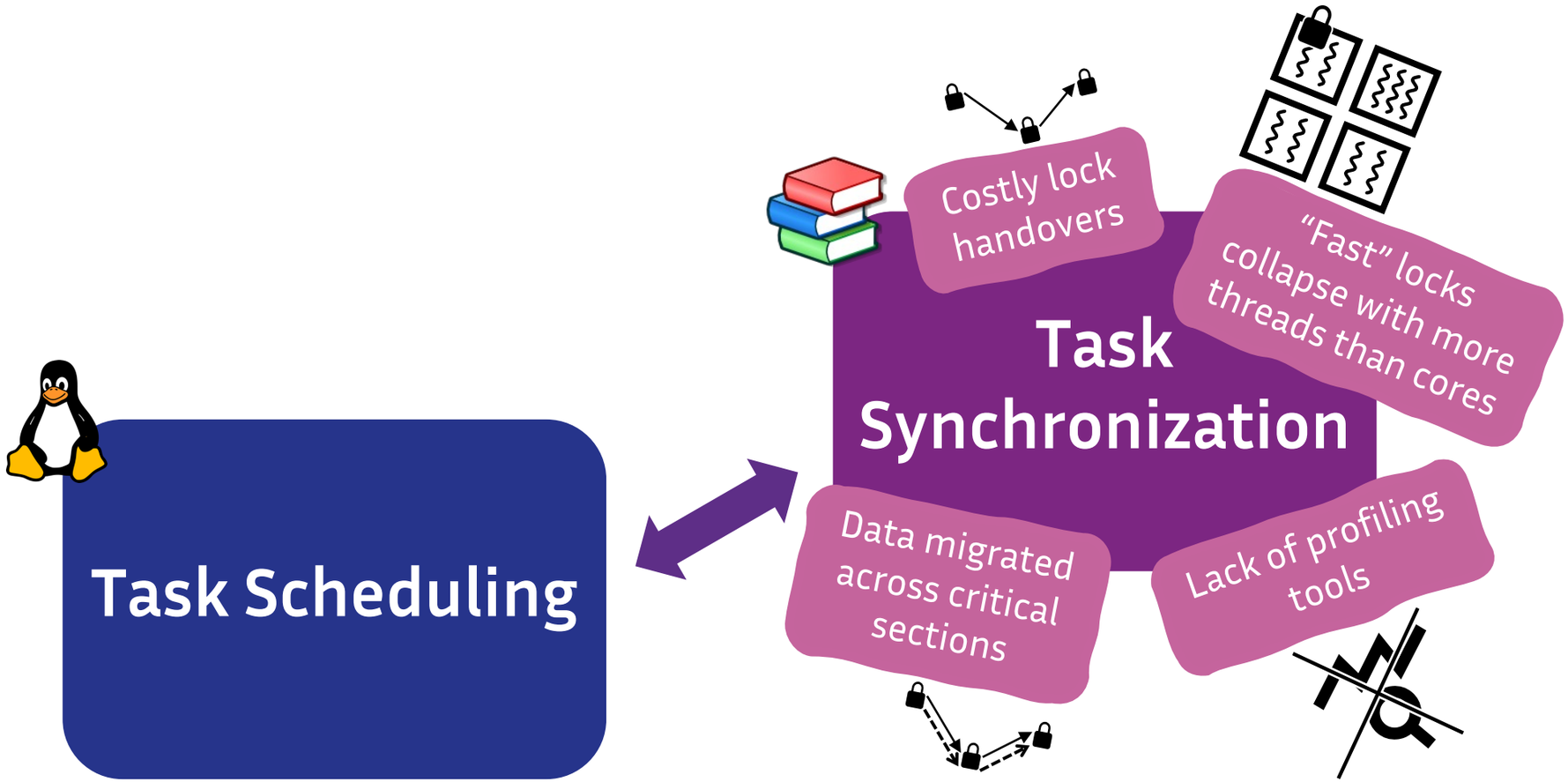


Bottlenecks Across the Software Stack

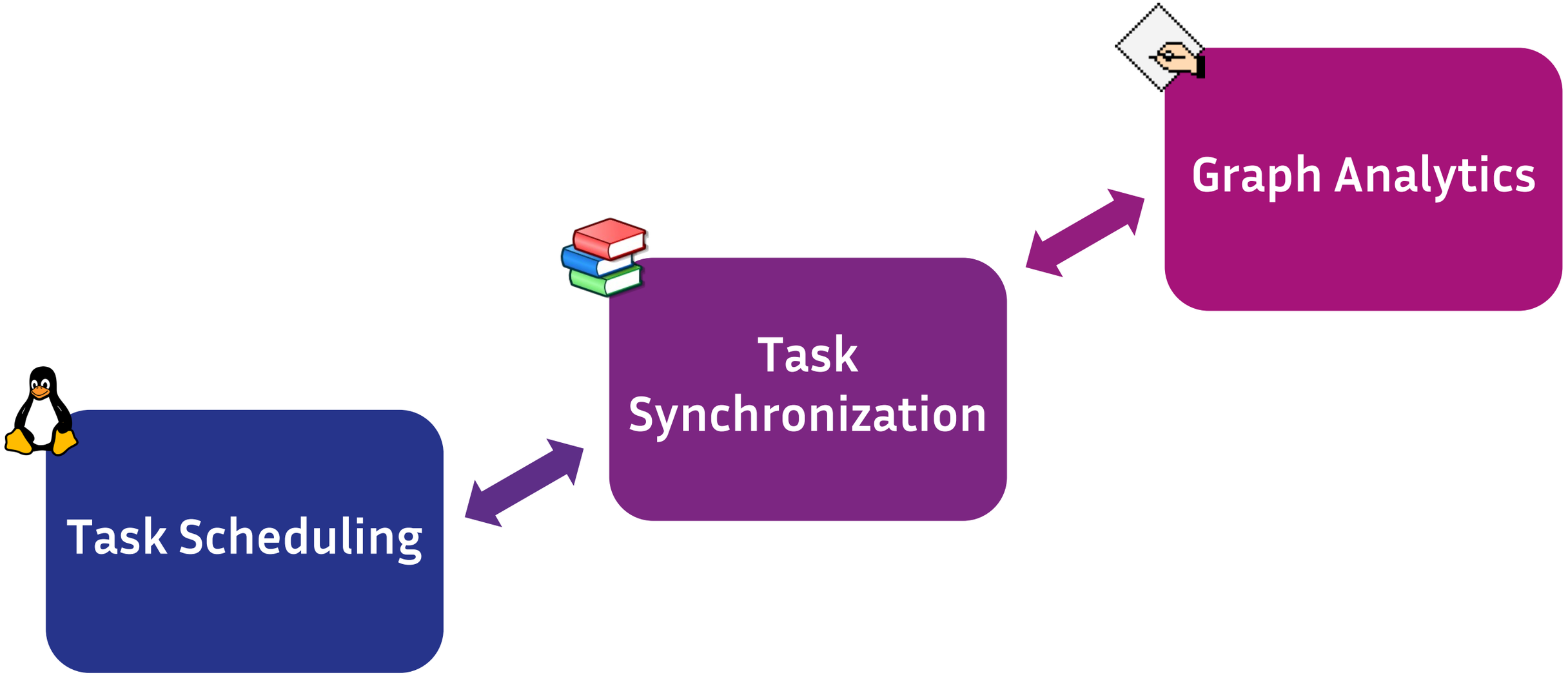
Task Synchronization

Task Scheduling

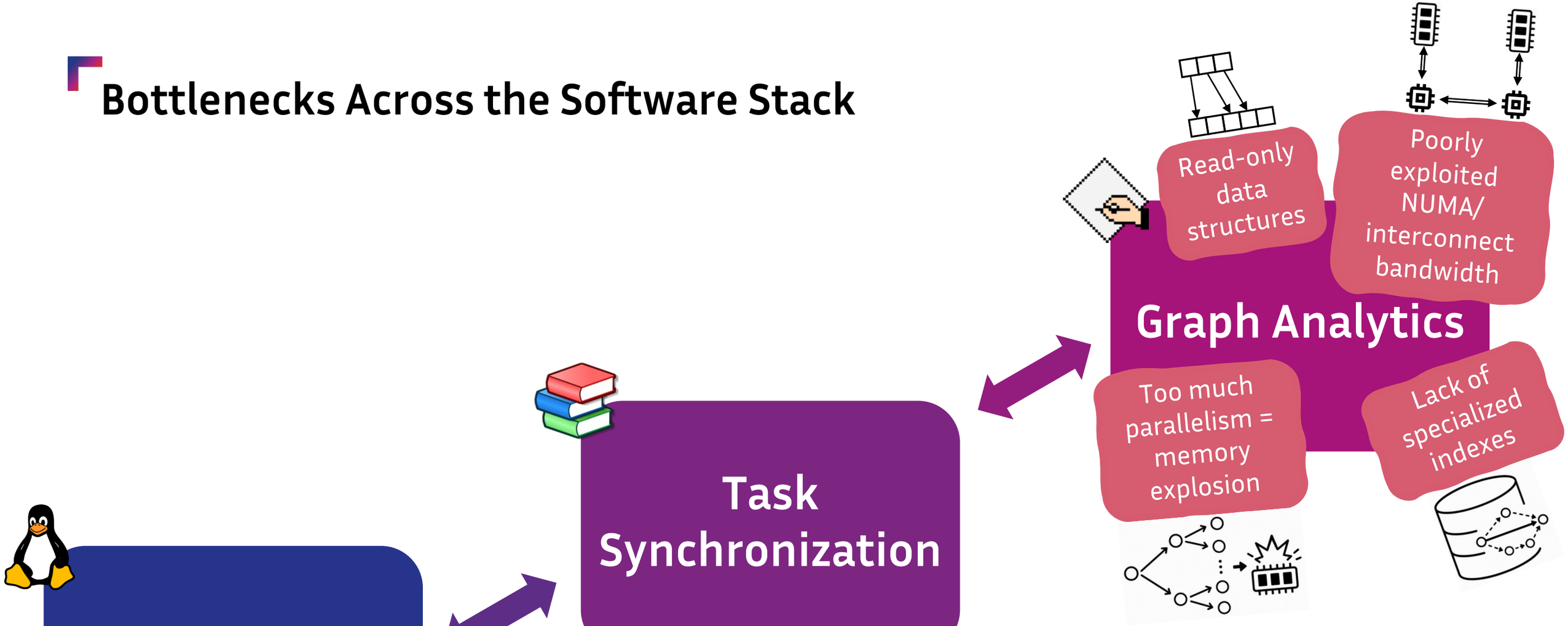
Bottlenecks Across the Software Stack



Bottlenecks Across the Software Stack



Bottlenecks Across the Software Stack



Presented Works

Presented Works

Presented Works

Presented Works

Presented Works

1. Task Scheduling: "Wasted Cores" [EuroSys '16]

- Four bugs showing the **lack of work conservation** in the Linux scheduler
  - Went unnoticed for many years!

1. Task Scheduling: "Wasted Cores" [EuroSys '16]

- Four bugs showing the **lack of work conservation** in the Linux scheduler
  - Went unnoticed for many years!

Work conservation invariant: ∃ idle core ⇒
∄ core with several tasks to run

1. Task Scheduling: "Wasted Cores" [EuroSys '16]

- Four bugs showing the **lack of work conservation** in the Linux scheduler
  - Went unnoticed for many years!

Work conservation invariant: ∃ idle core ⇒
∄ core with several tasks to run

tasks in run queue: 0 1 2

1. Task Scheduling: "Wasted Cores" [EuroSys '16]

- Four bugs showing the **lack of work conservation** in the Linux scheduler
  - Went unnoticed for many years!

Work conservation invariant: ∃ idle core ⇒ ∄ core with several tasks to run

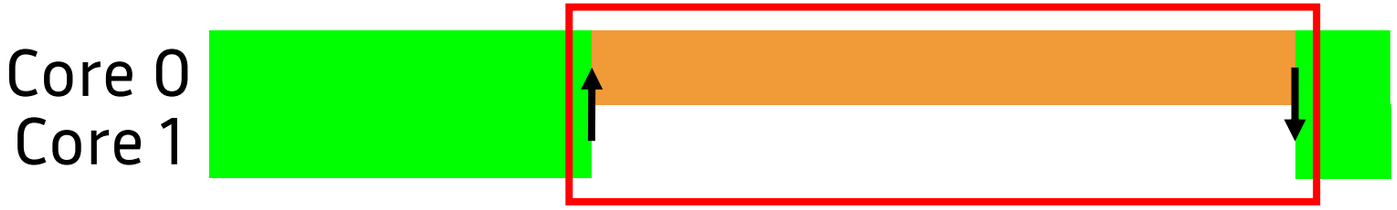
tasks in run queue: 0 1 2

Work conservation maintained 🙂

1. Task Scheduling: "Wasted Cores" [EuroSys '16]

- Four bugs showing the **lack of work conservation** in the Linux scheduler
  - Went unnoticed for many years!

Work conservation invariant: ∃ idle core ⇒ ∄ core with several tasks to run



Unfortunate task migration = work conservation not maintained ☹

tasks in run queue: 0 1 2

1. Task Scheduling: "Wasted Cores" [EuroSys '16]

- Four bugs showing the **lack of work conservation** in the Linux scheduler
  - Went unnoticed for many years!

Work conservation invariant: ∃ idle core ⇒ ∄ core with several tasks to run

tasks in run queue: 0 1 2

- **Why?**

1. Task Scheduling: "Wasted Cores" [EuroSys '16]

- Four bugs showing the **lack of work conservation** in the Linux scheduler
  - Went unnoticed for many years!

Work conservation invariant: ∃ idle core ⇒ ∄ core with several tasks to run

tasks in run queue: 0 1 2

- **Why?**
  - Complex temporal/spatial problem: needs heuristics to make local decisions

1. Task Scheduling: "Wasted Cores" [EuroSys '16]

- Four bugs showing the **lack of work conservation** in the Linux scheduler
  - Went unnoticed for many years!

Work conservation invariant: ∃ idle core ⇒ ∄ core with several tasks to run

tasks in run queue: 0 1 2

- **Why?**
  - Complex temporal/spatial problem: needs heuristics to make local decisions
  - More adjustments for new hardware (e.g., NUMA) and use cases (e.g., containers)

1. Task Scheduling: "Wasted Cores" [EuroSys '16]

- Four bugs showing the **lack of work conservation** in the Linux scheduler
  - Went unnoticed for many years!

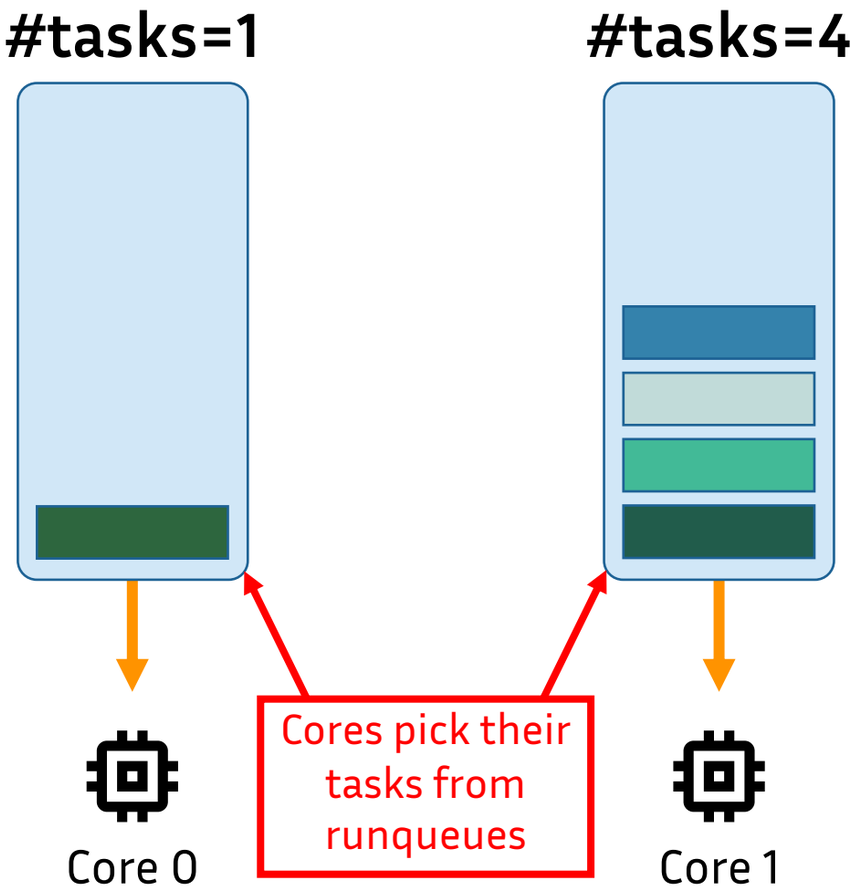
Work conservation invariant: ∃ idle core ⇒ ∄ core with several tasks to run

tasks in run queue: 0 1 2

- **Why?**
  - Complex temporal/spatial problem: needs heuristics to make local decisions
  - More adjustments for new hardware (e.g., NUMA) and use cases (e.g., containers)
  - These heuristics/adjustments often don't compose well!

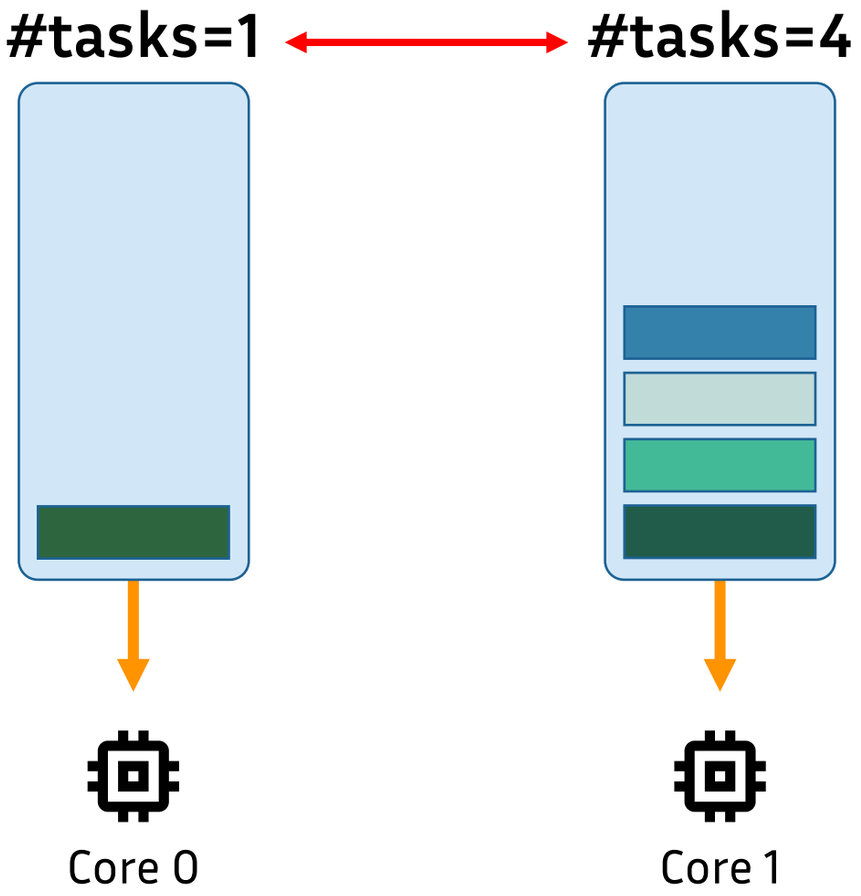
1. Task Scheduling: "Wasted Cores" [EuroSys '16]

Background: Load balancing in Linux (2016, simplified)



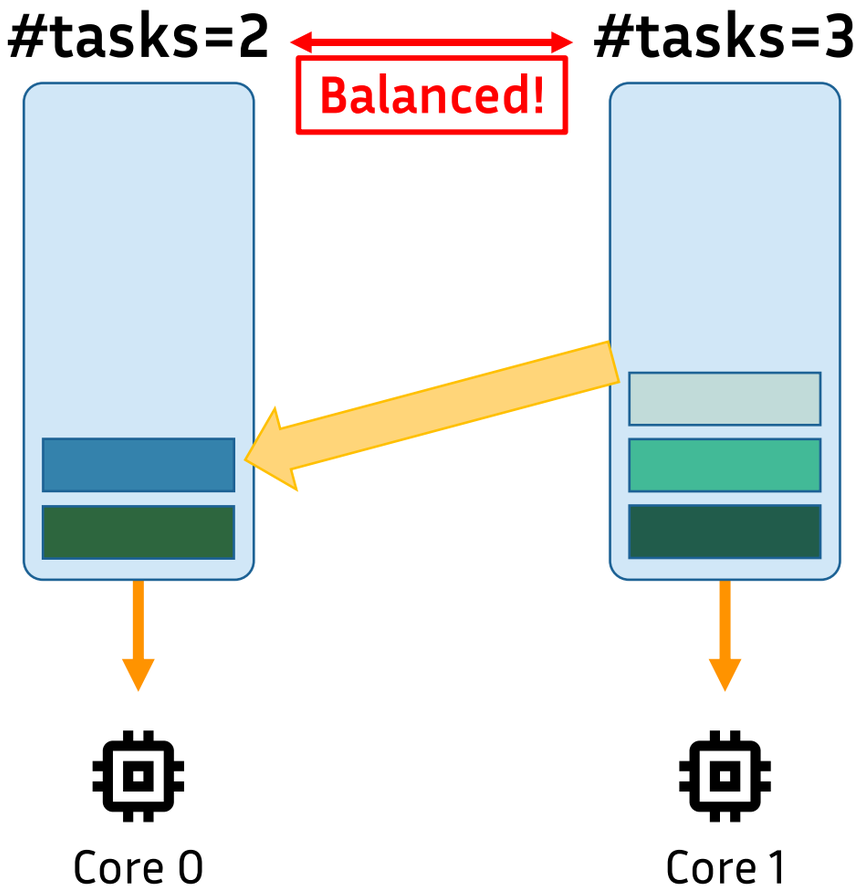
1. Task Scheduling: "Wasted Cores" [EuroSys '16]

Background: Load balancing in Linux (2016, simplified)



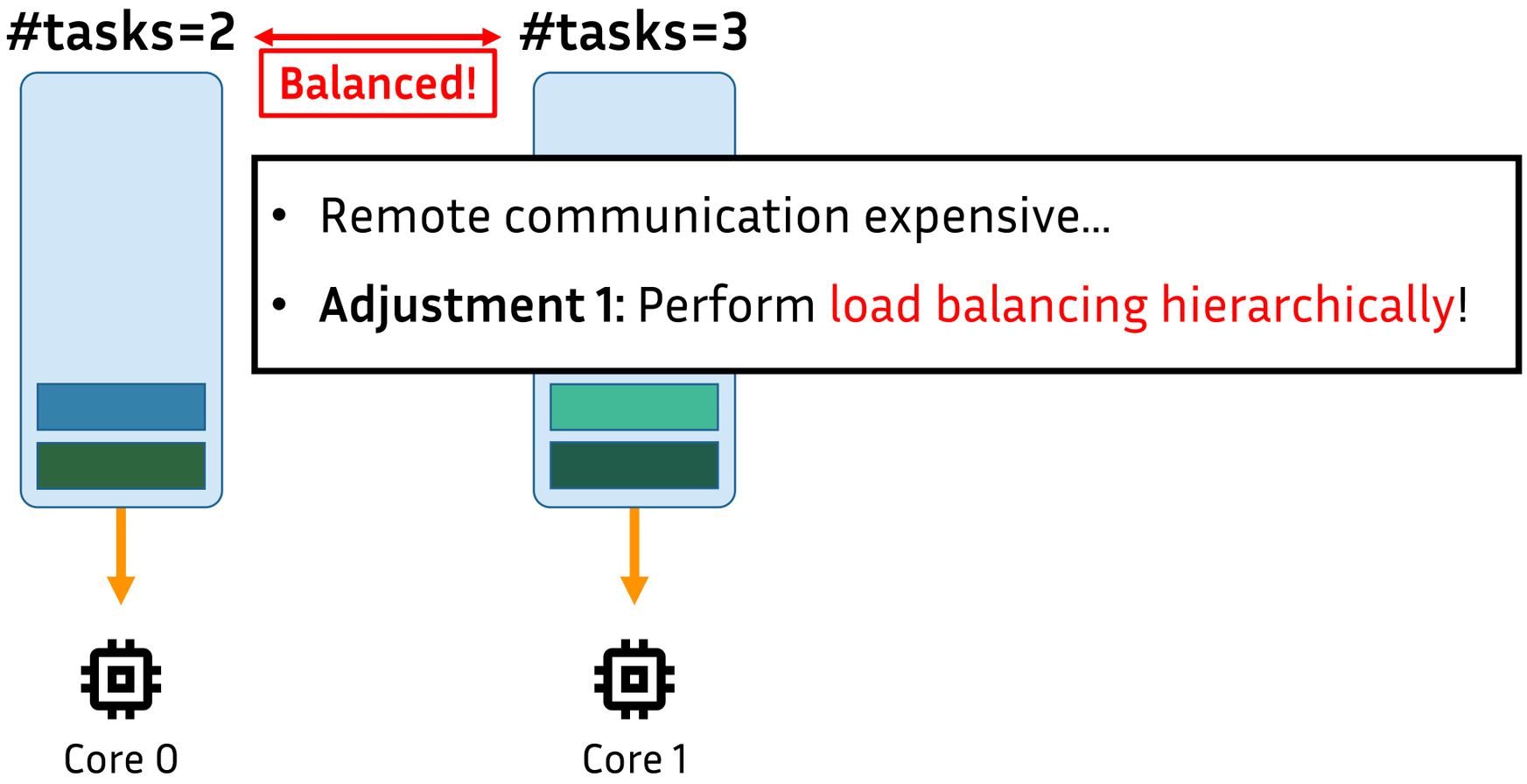
1. Task Scheduling: "Wasted Cores" [EuroSys '16]

Background: Load balancing in Linux (2016, simplified)



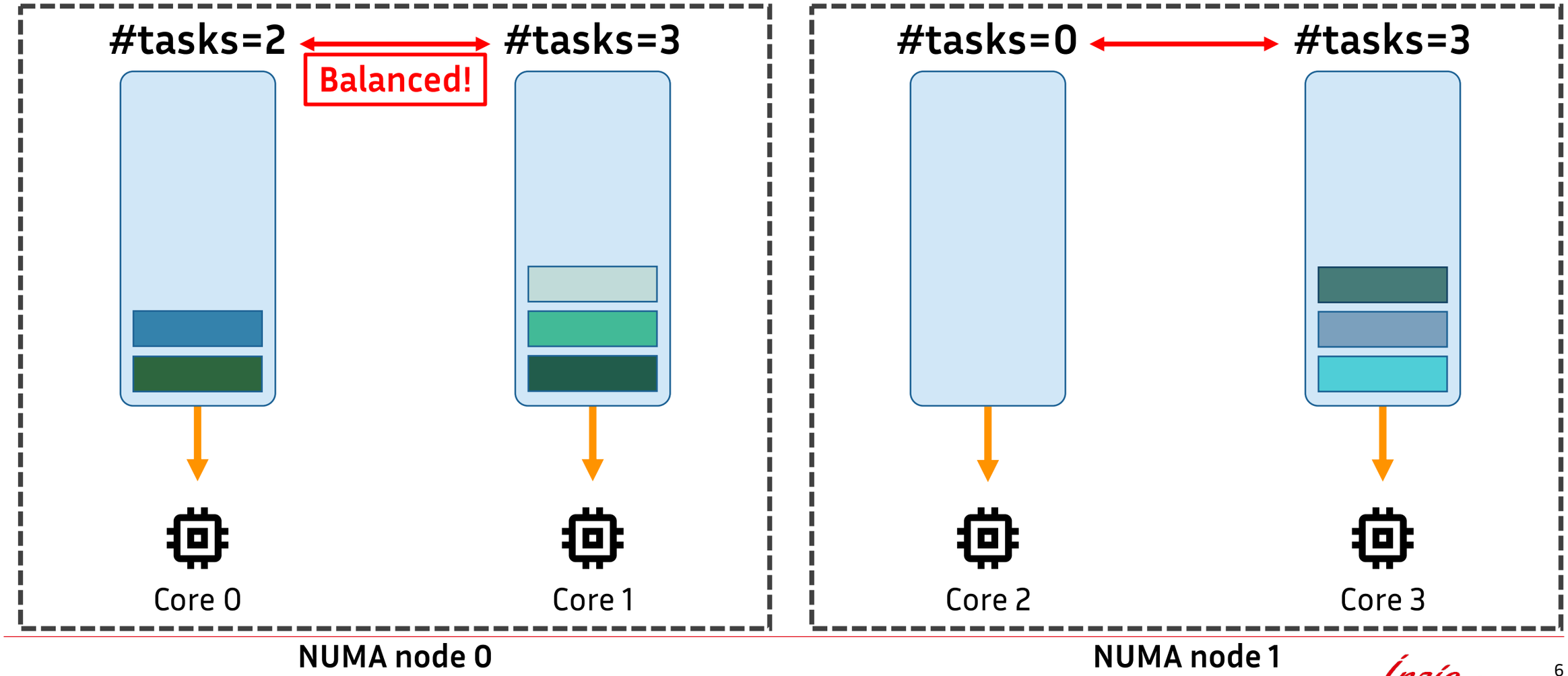
1. Task Scheduling: "Wasted Cores" [EuroSys '16]

Background: Load balancing in Linux (2016, simplified)



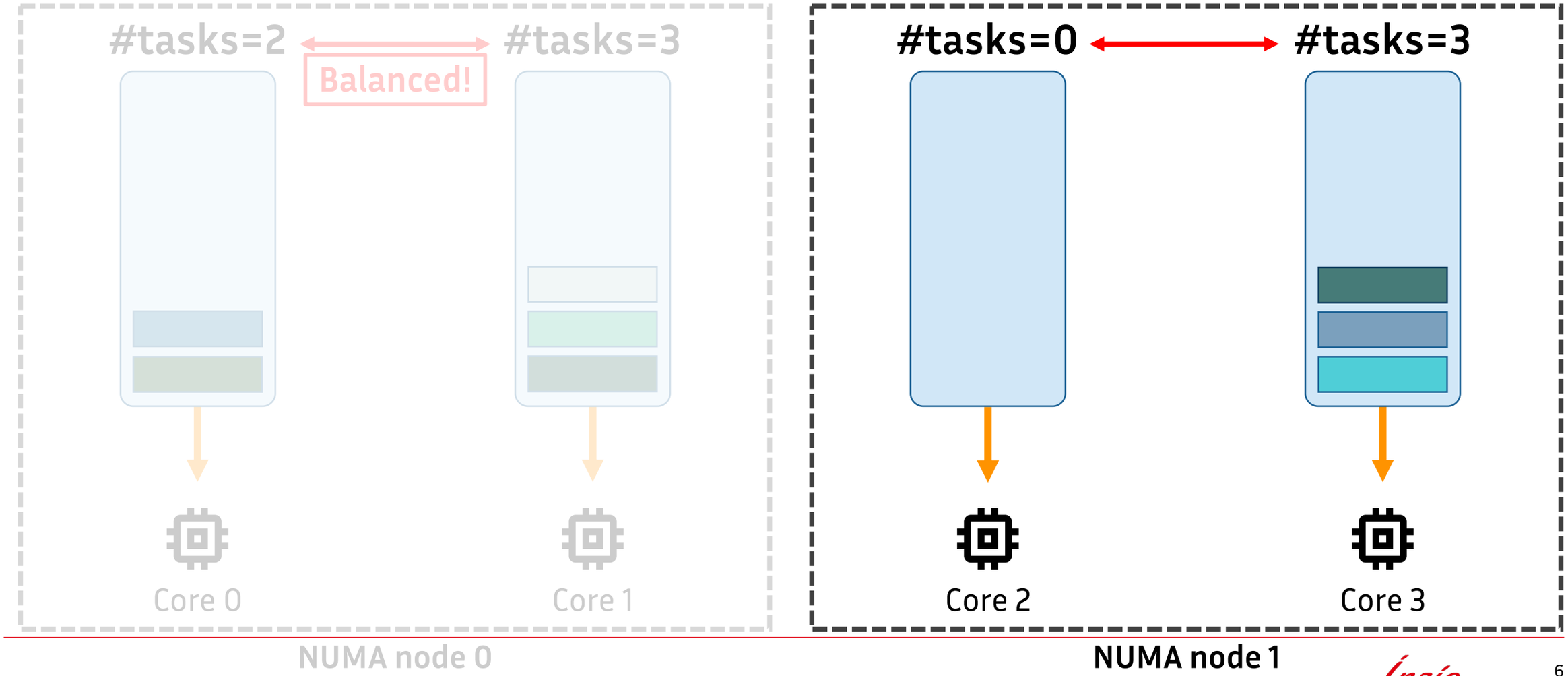
1. Task Scheduling: "Wasted Cores" [EuroSys '16]

Background: Load balancing in Linux (2016, simplified)



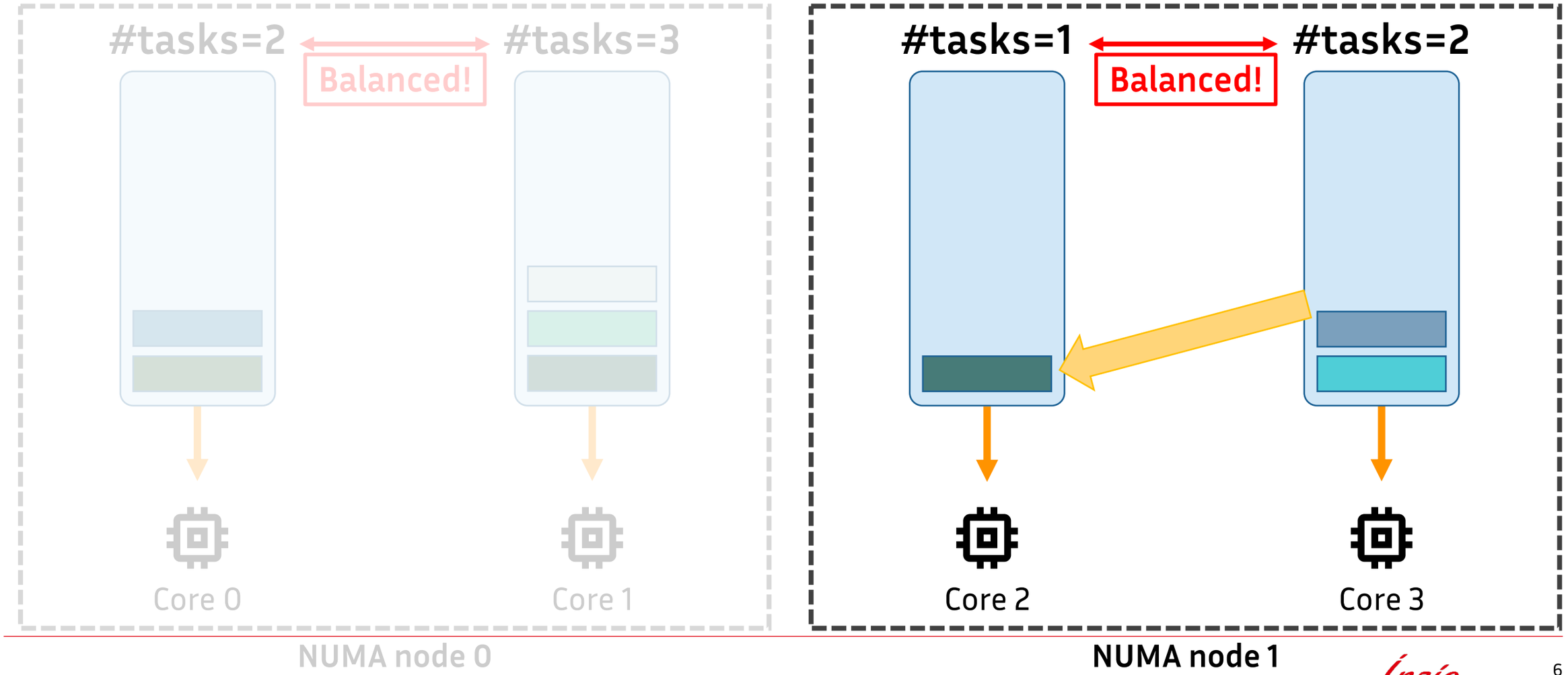
1. Task Scheduling: "Wasted Cores" [EuroSys '16]

Background: Load balancing in Linux (2016, simplified)



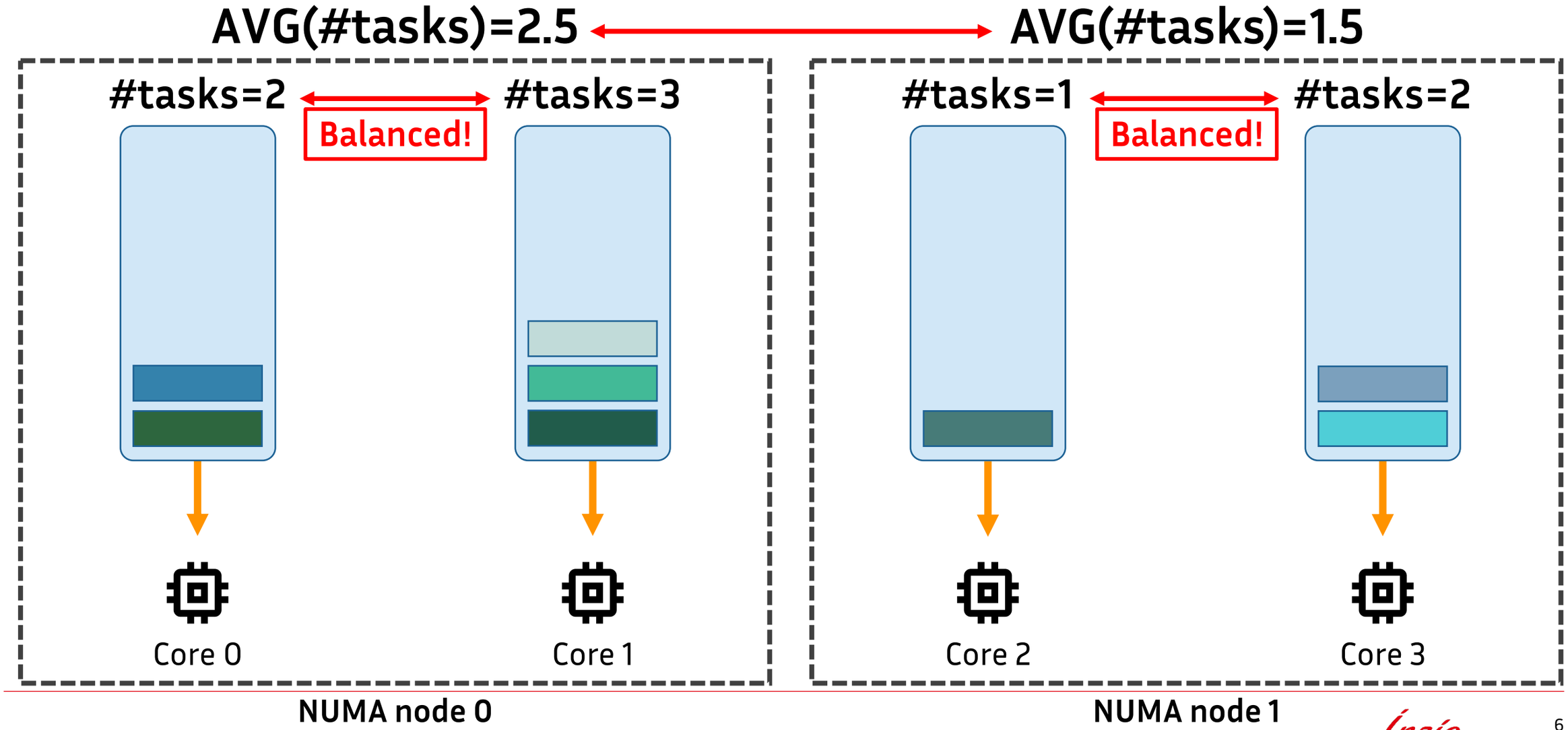
1. Task Scheduling: "Wasted Cores" [EuroSys '16]

Background: Load balancing in Linux (2016, simplified)



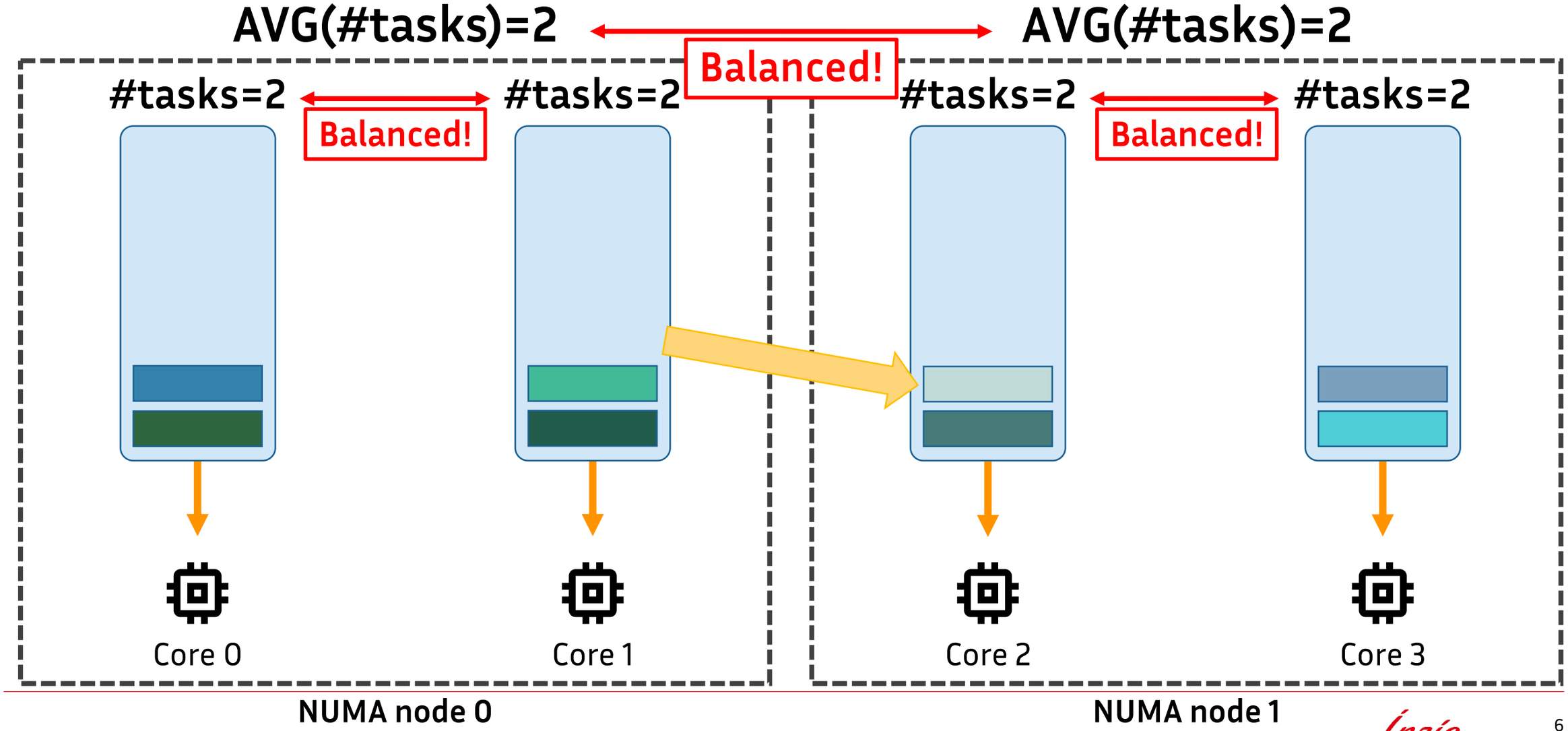
1. Task Scheduling: "Wasted Cores" [EuroSys '16]

Background: Load balancing in Linux (2016, simplified)



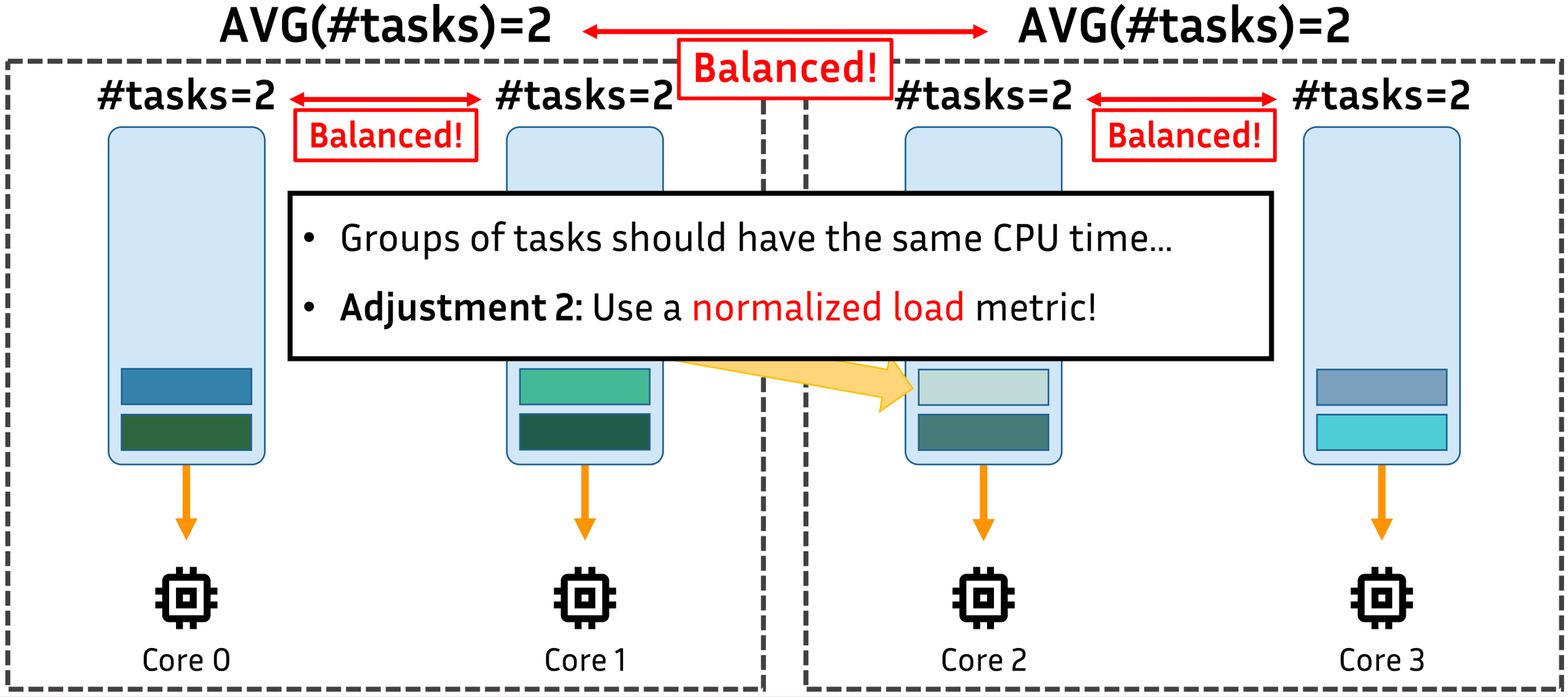
1. Task Scheduling: "Wasted Cores" [EuroSys '16]

Background: Load balancing in Linux (2016, simplified)



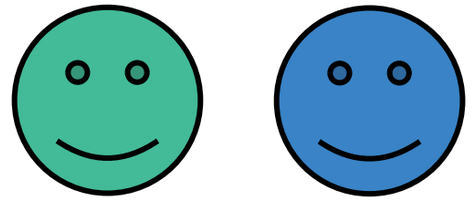
1. Task Scheduling: "Wasted Cores" [EuroSys '16]

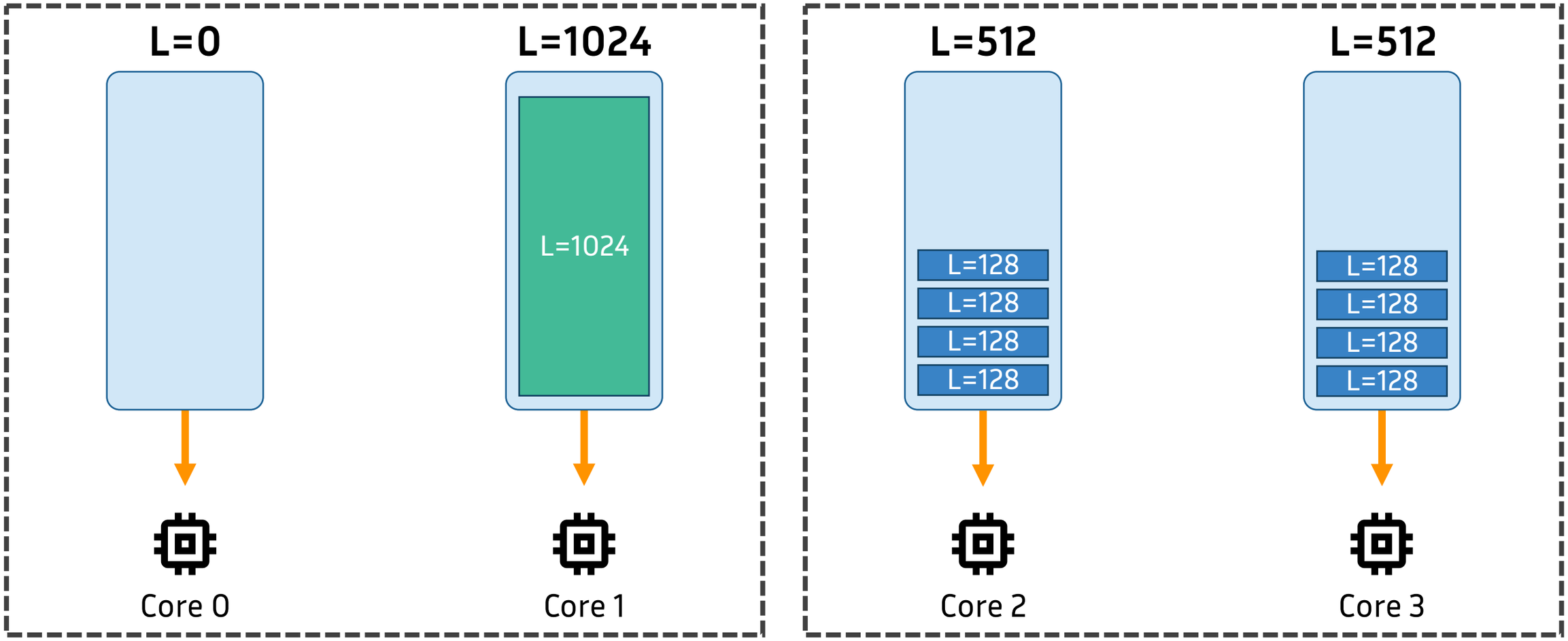
Background: Load balancing in Linux (2016, simplified)



1. Task Scheduling: "Wasted Cores" [EuroSys '16]

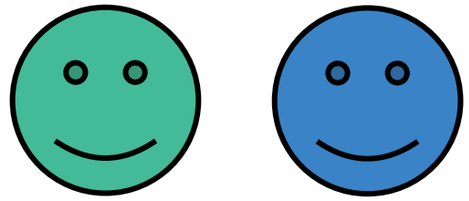
Bug #1: "Group Imbalance"

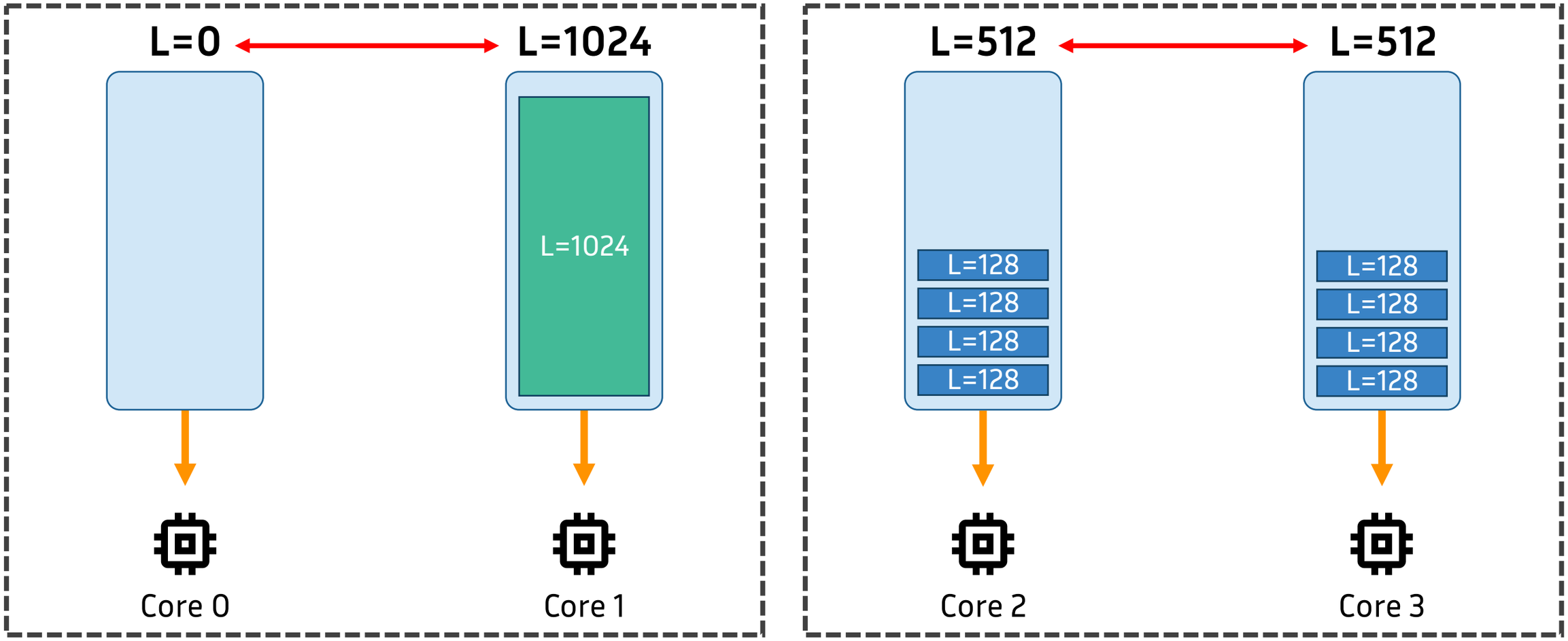




1. Task Scheduling: "Wasted Cores" [EuroSys '16]

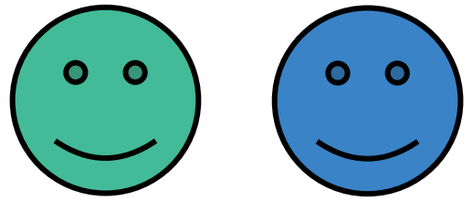
Bug #1: "Group Imbalance"

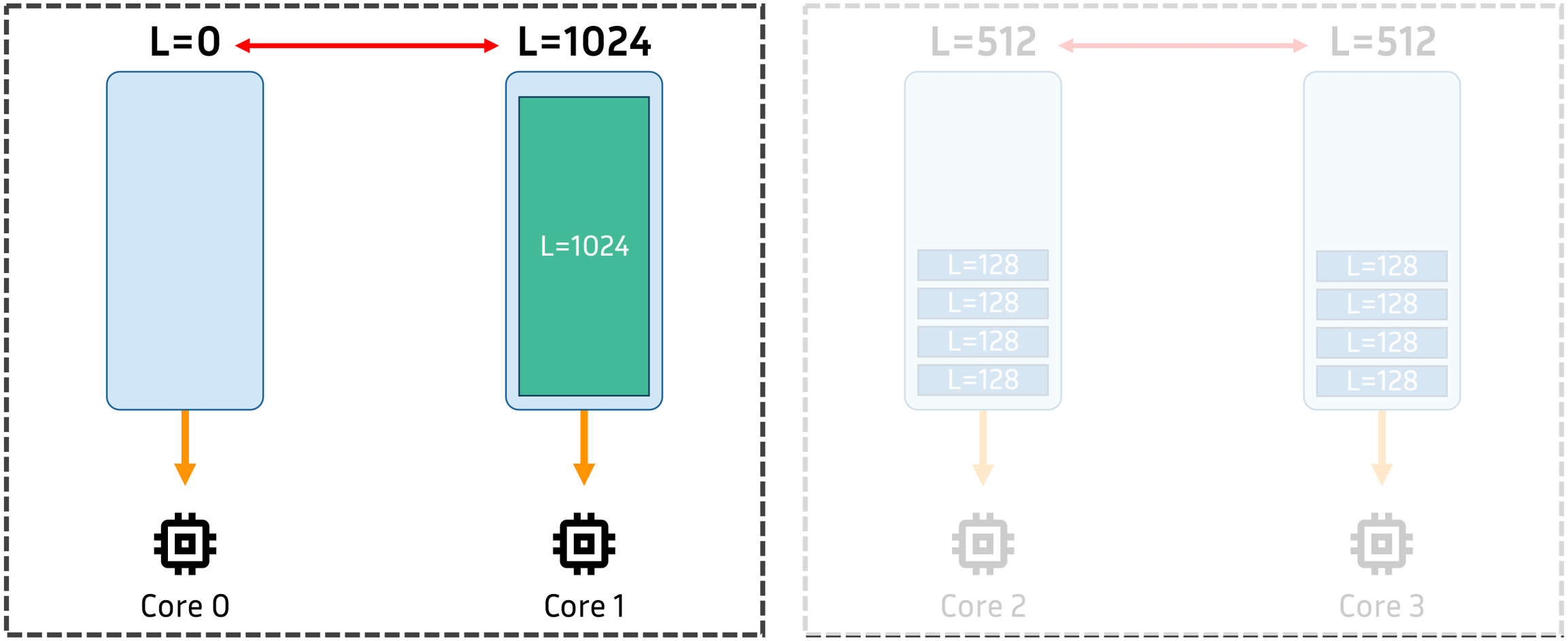




1. Task Scheduling: "Wasted Cores" [EuroSys '16]

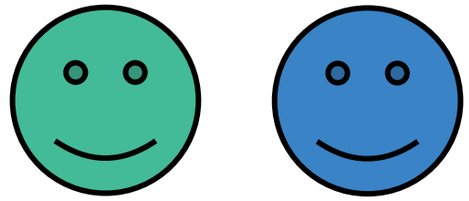
Bug #1: "Group Imbalance"

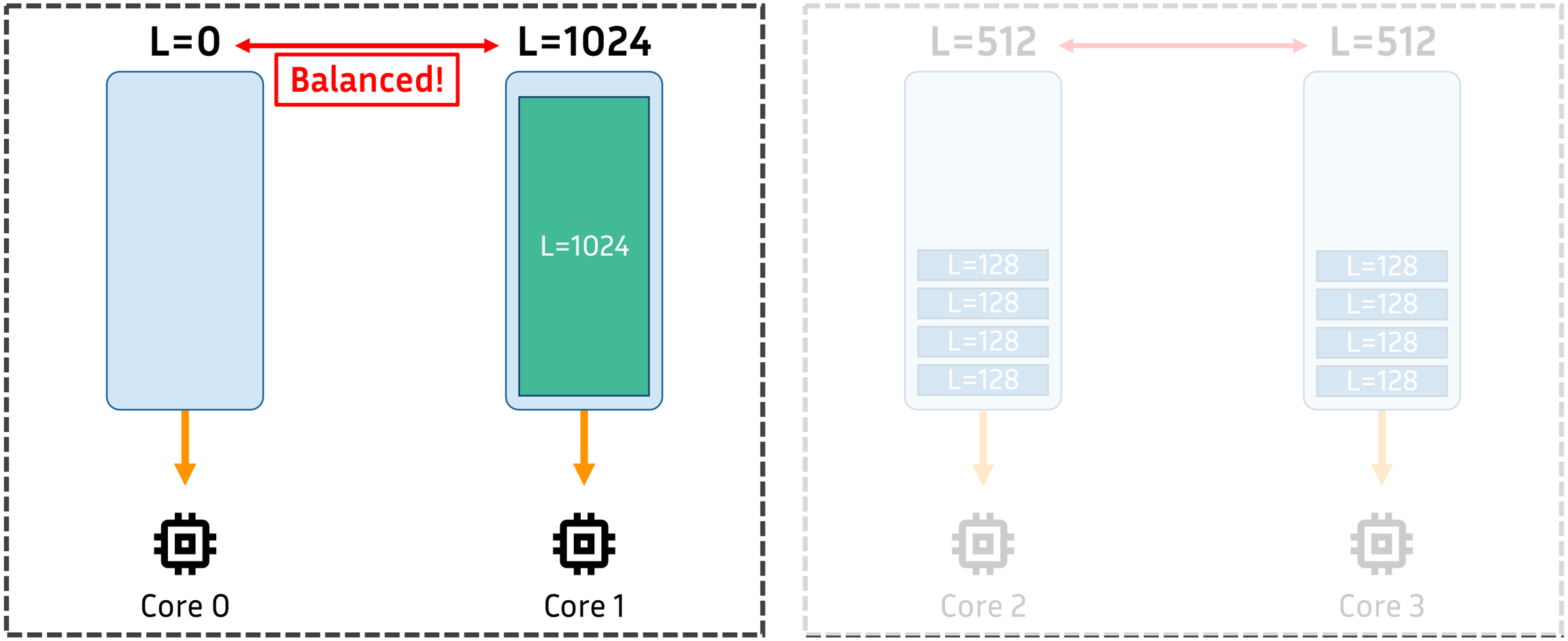




1. Task Scheduling: "Wasted Cores" [EuroSys '16]

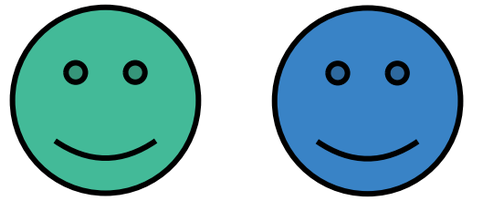
Bug #1: "Group Imbalance"

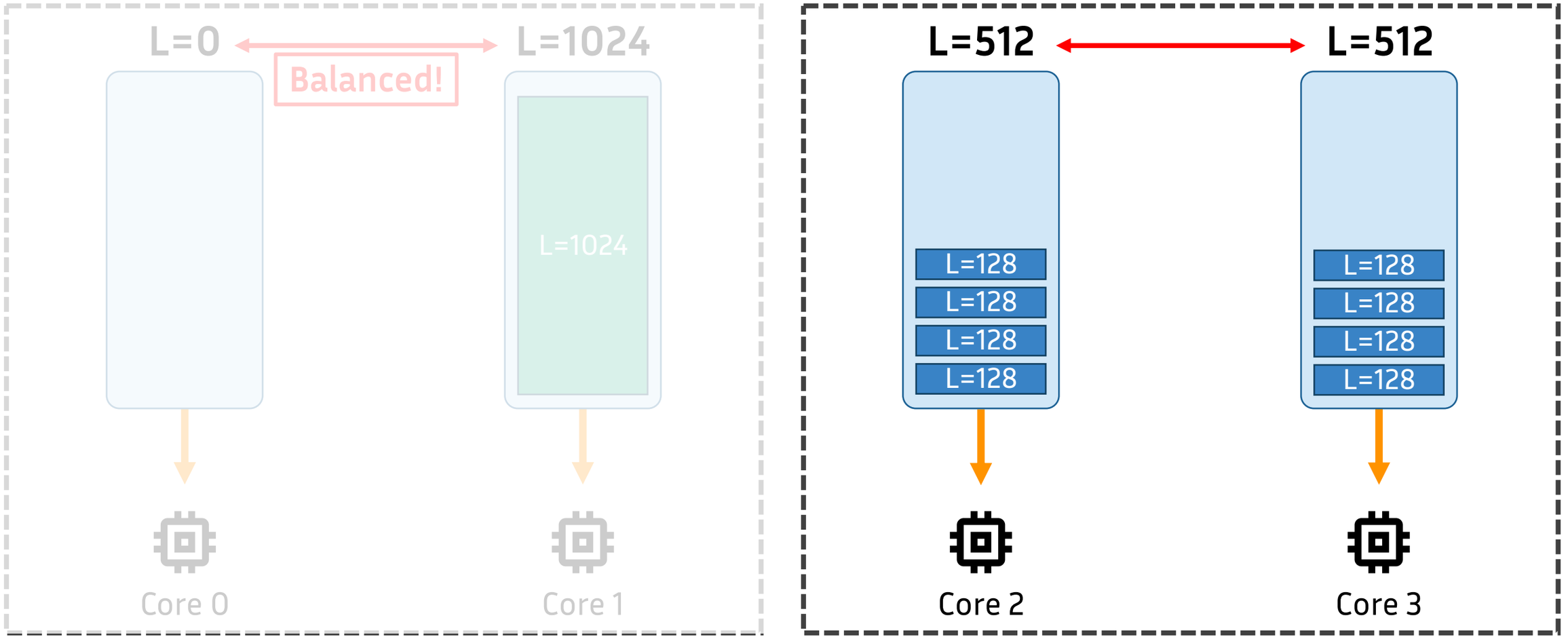




1. Task Scheduling: "Wasted Cores" [EuroSys '16]

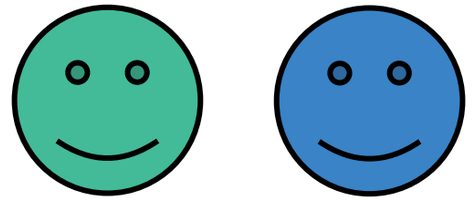
Bug #1: "Group Imbalance"

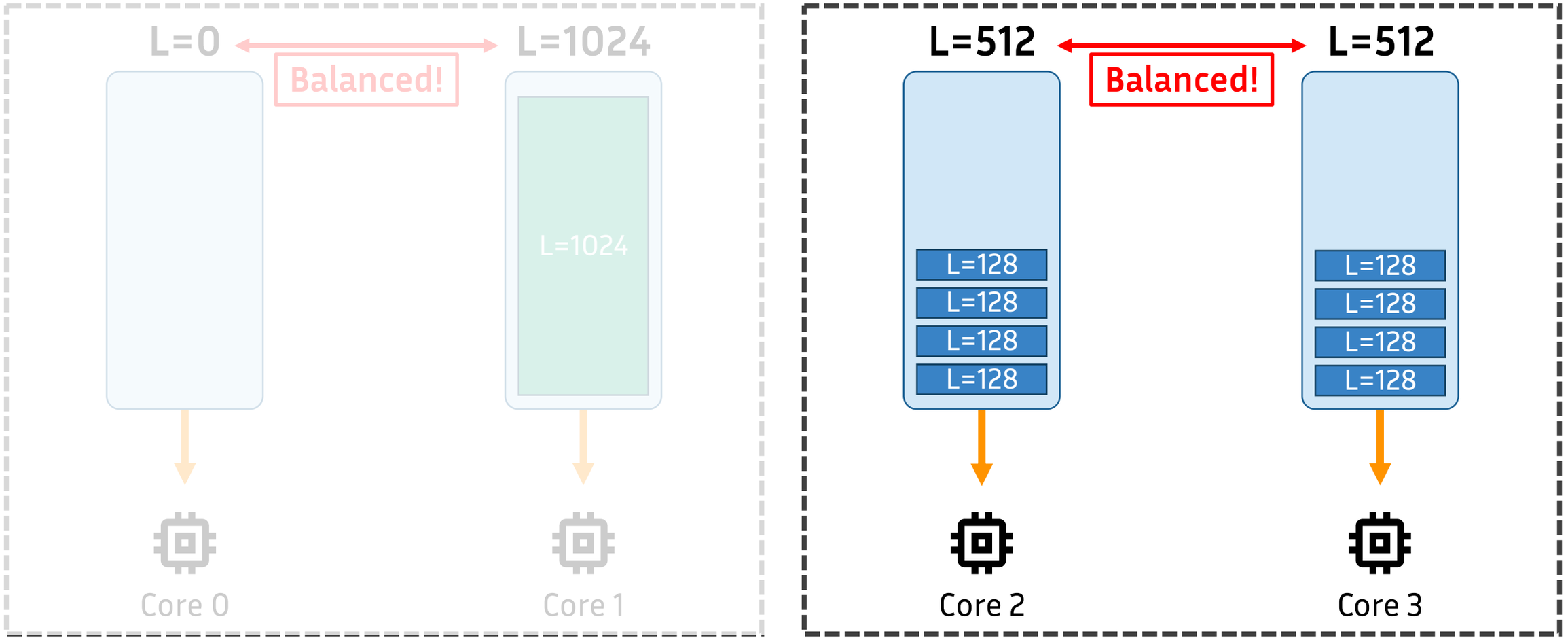




1. Task Scheduling: "Wasted Cores" [EuroSys '16]

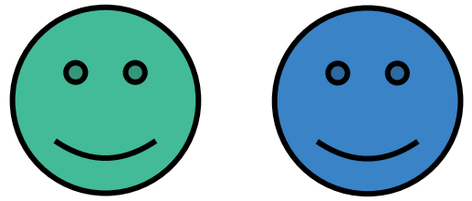
Bug #1: "Group Imbalance"

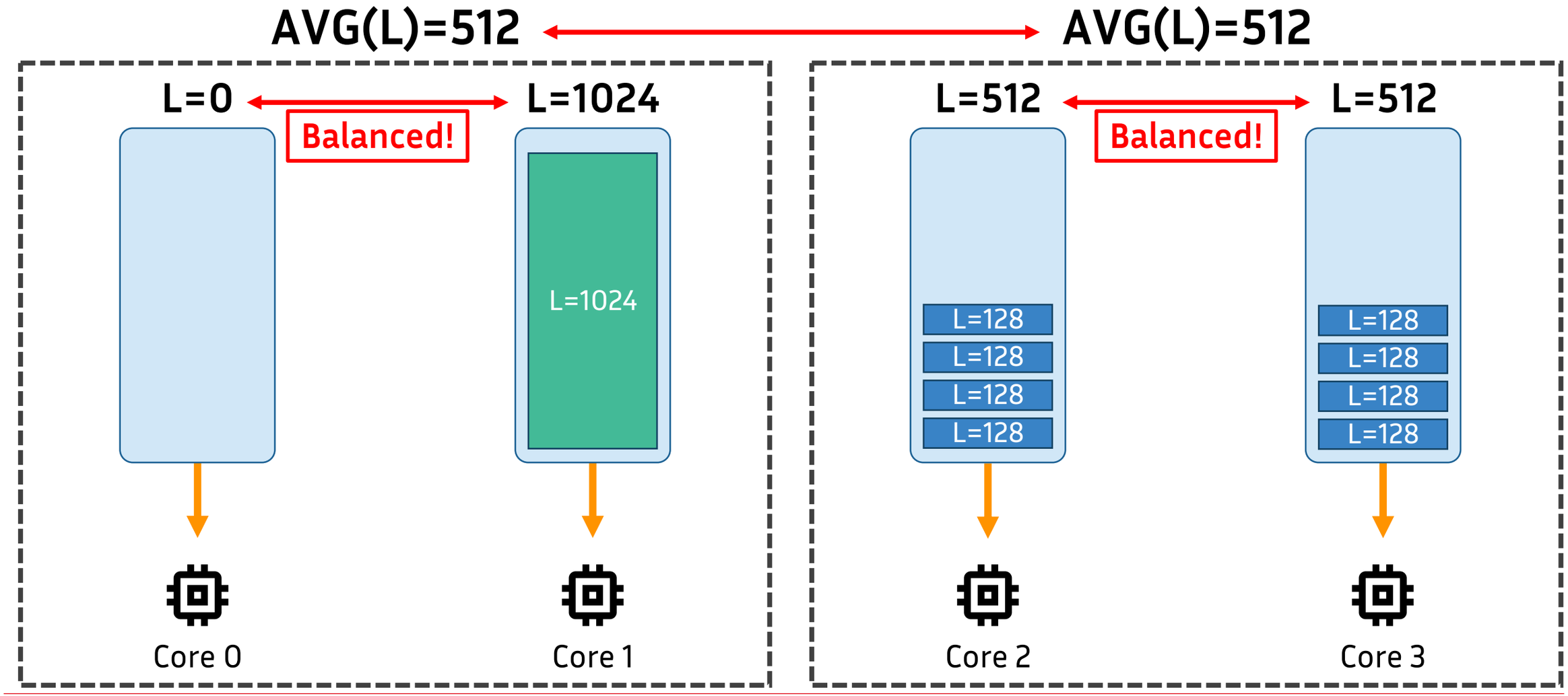




1. Task Scheduling: "Wasted Cores" [EuroSys '16]

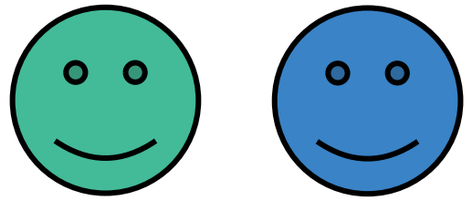
Bug #1: "Group Imbalance"

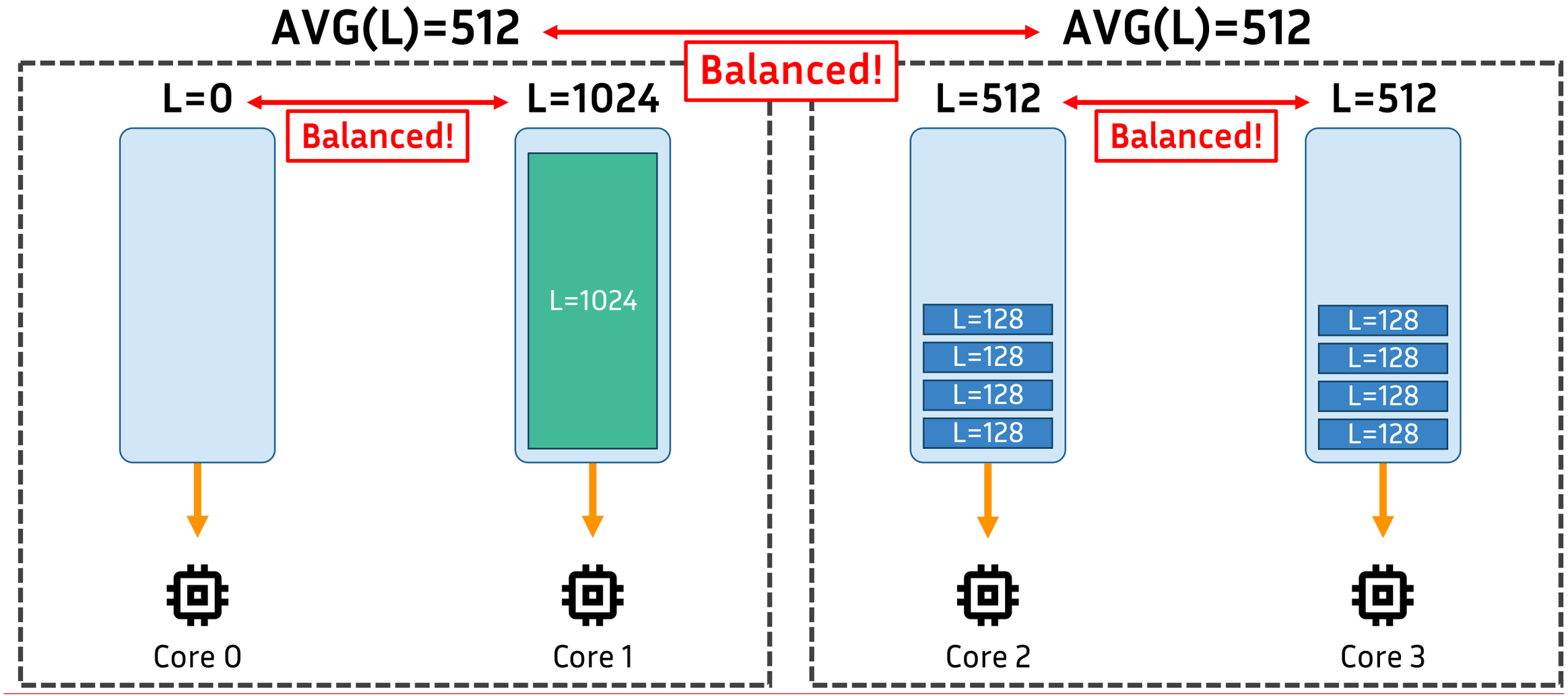




1. Task Scheduling: "Wasted Cores" [EuroSys '16]

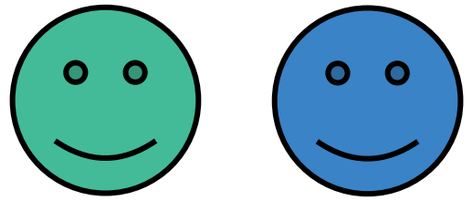
Bug #1: "Group Imbalance"

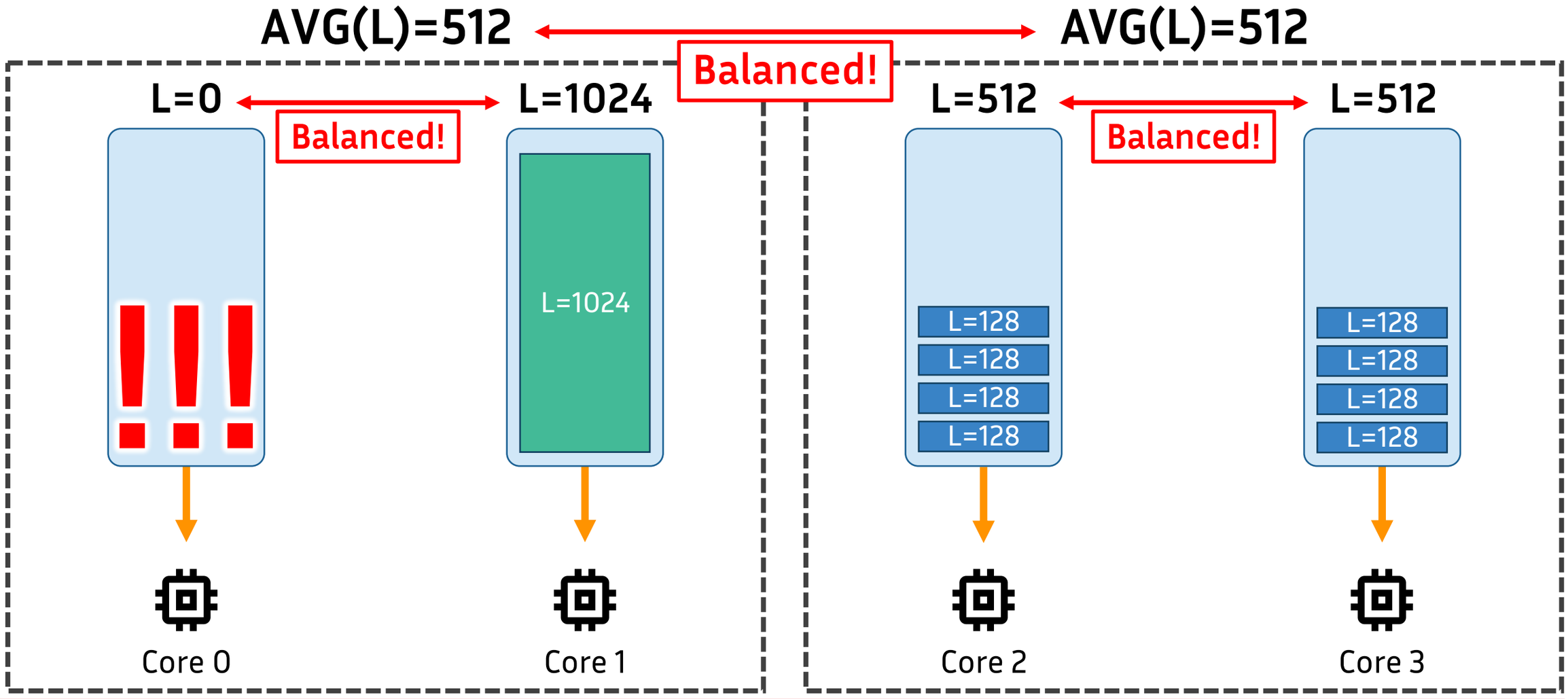




1. Task Scheduling: "Wasted Cores" [EuroSys '16]

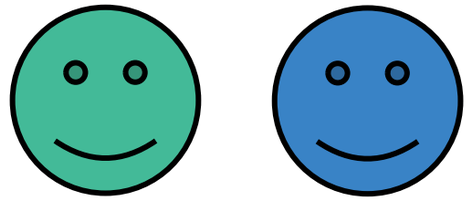
Bug #1: "Group Imbalance"

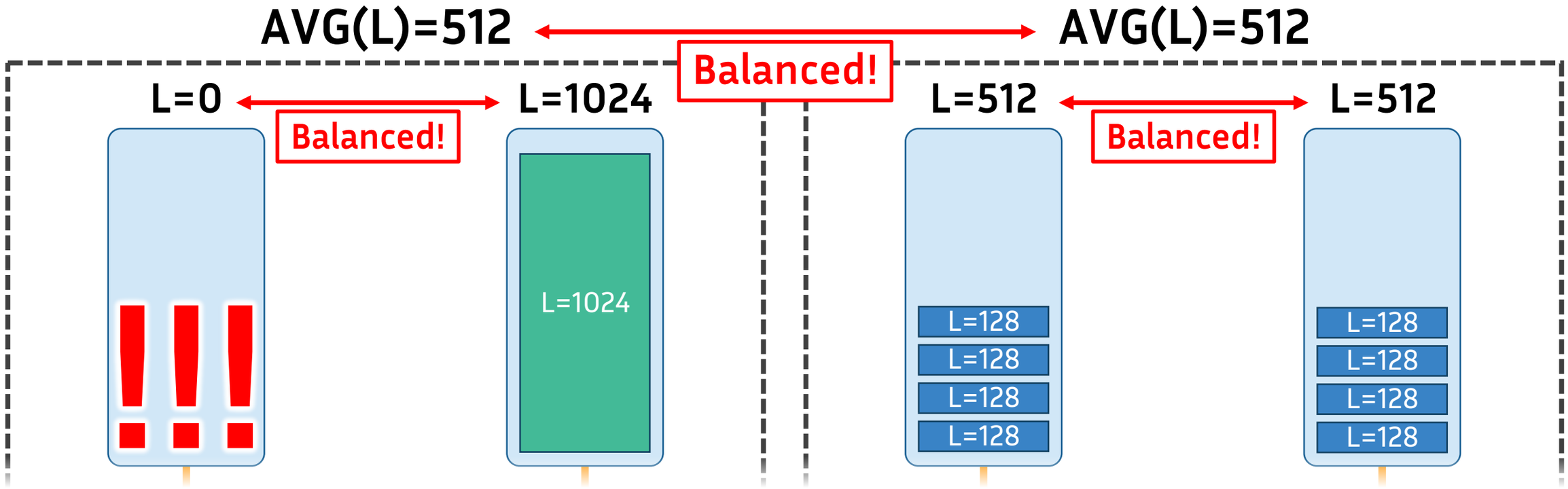




1. Task Scheduling: "Wasted Cores" [EuroSys '16]

Bug #1: "Group Imbalance"

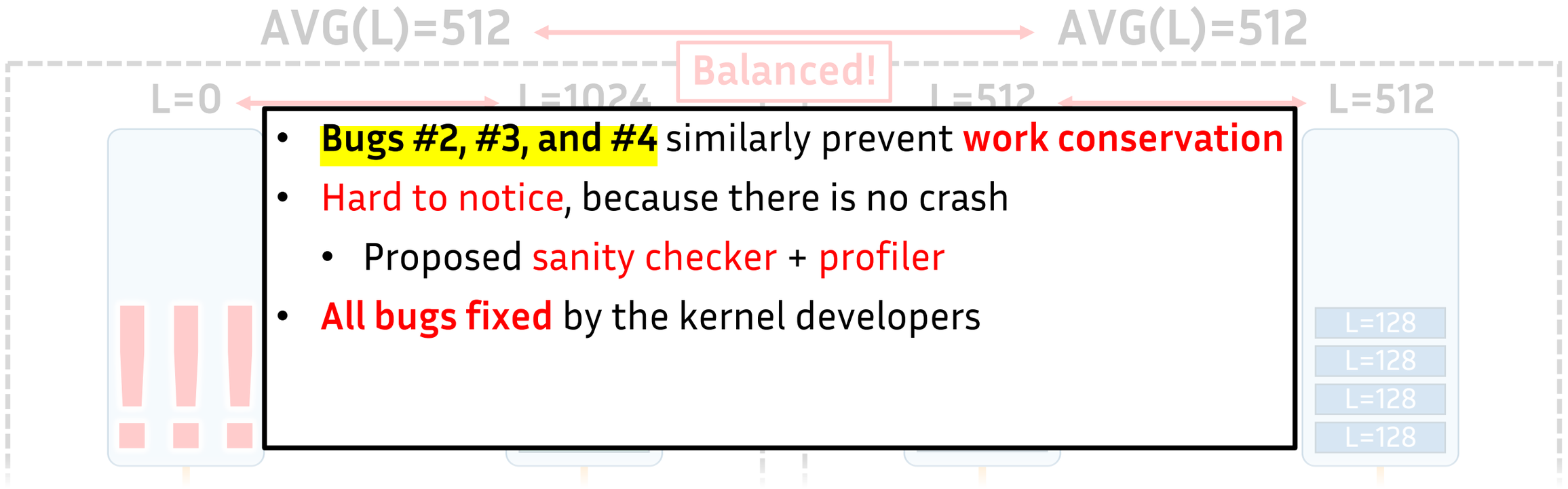




Fundamental work conservation violation in the load balancing algorithm…

1. Task Scheduling: "Wasted Cores" [EuroSys '16]

Bug #1: "Group Imbalance"

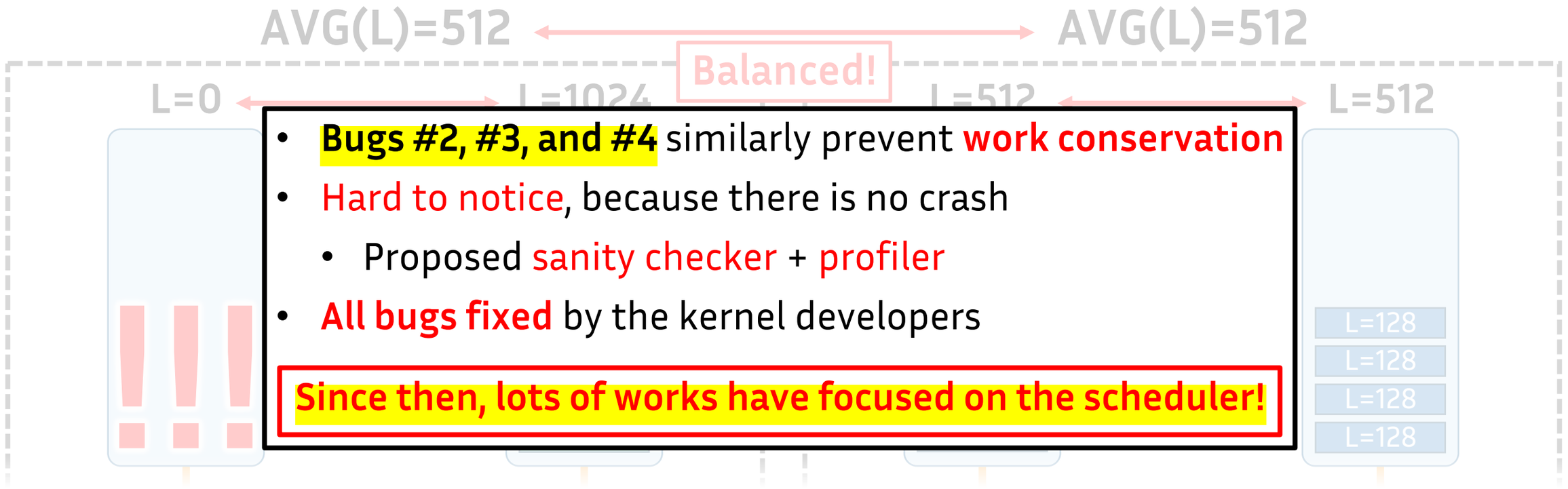


- **Bugs #2, #3, and #4** similarly prevent **work conservation**
- Hard to notice, because there is no crash
  - Proposed sanity checker + profiler
- **All bugs fixed** by the kernel developers

Fundamental work conservation violation in the load balancing algorithm…

1. Task Scheduling: "Wasted Cores" [EuroSys '16]

Bug #1: "Group Imbalance"



- **Bugs #2, #3, and #4** similarly prevent **work conservation**
- Hard to notice, because there is no crash
  - Proposed sanity checker + profiler
- **All bugs fixed** by the kernel developers

Since then, lots of works have focused on the scheduler!

Fundamental work conservation violation in the load balancing algorithm...

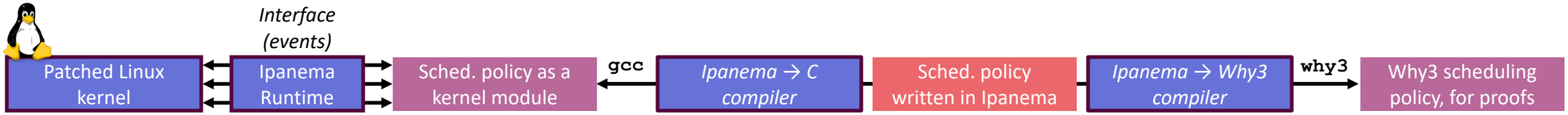
1. Task Scheduling: Ipanema [HotOS '17, EuroSys '20]

- **Ipanema:** a DSL to write scheduling policies with proven **work conservation**
  - Simple (expressiveness ↔ expertise), safe (bounded loops)

1. Task Scheduling: Ipanema [HotOS '17, EuroSys '20]

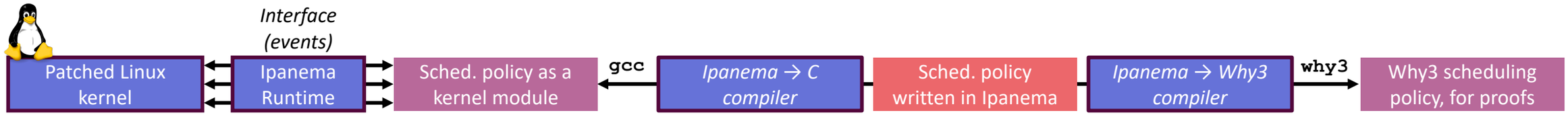
- **Ipanema:** a DSL to write scheduling policies with proven **work conservation**
  - Simple (expressiveness ↔ expertise), safe (bounded loops)
  - Produces C (kernel module)

1. Task Scheduling: Ipanema [HotOS '17, EuroSys '20]



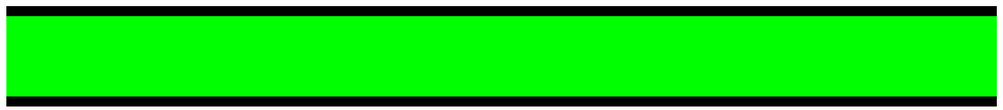
- **Ipanema:** a DSL to write scheduling policies with proven **work conservation**
  - Simple (expressiveness ↔ expertise), safe (bounded loops)
  - Produces C (kernel module) and Why3 code to prove work conservation

1. Task Scheduling: Ipanema [HotOS '17, EuroSys '20]



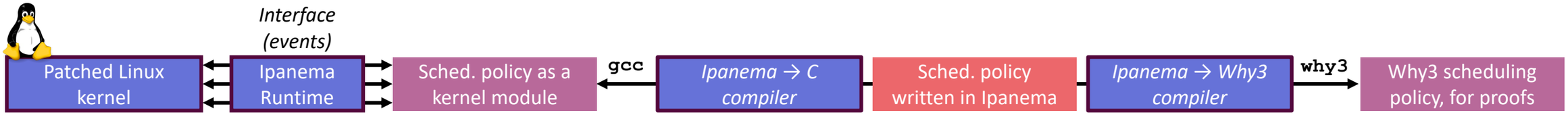
- **Ipanema:** a DSL to write scheduling policies with proven **work conservation**
  - Simple (expressiveness ↔ expertise), safe (bounded loops)
  - Produces C (kernel module) and Why3 code to prove work conservation

- **Problem:** full work conservation is too strong a property



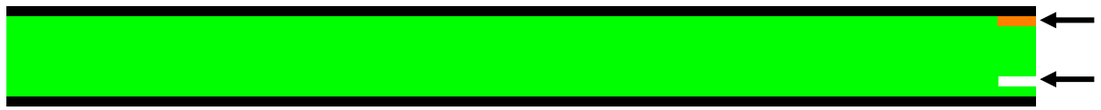
tasks in run queue: 0 1 2

1. Task Scheduling: Ipanema [HotOS '17, EuroSys '20]



- **Ipanema:** a DSL to write scheduling policies with proven **work conservation**
  - Simple (expressiveness ↔ expertise), safe (bounded loops)
  - Produces C (kernel module) and Why3 code to prove work conservation

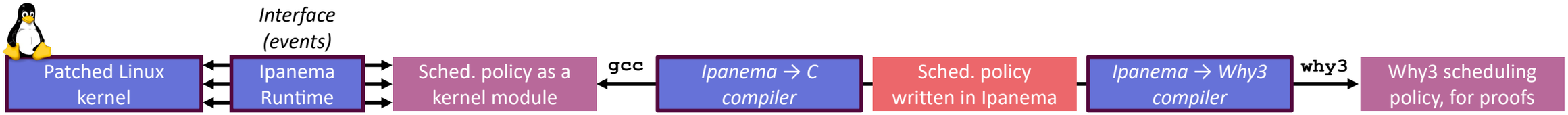
- **Problem:** full work conservation is too strong a property



If tasks wake up and
block at the exact same time...

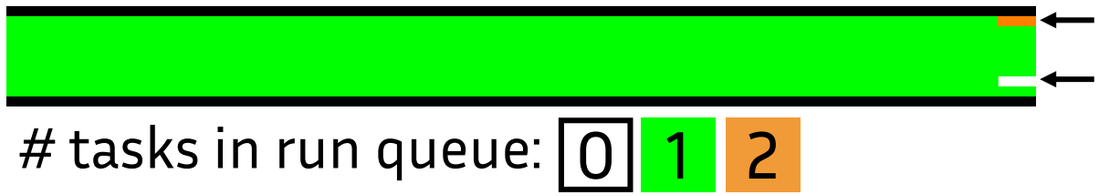
tasks in run queue: 0 1 2

1. Task Scheduling: Ipanema [HotOS '17, EuroSys '20]



- **Ipanema:** a DSL to write scheduling policies with proven **work conservation**
  - Simple (expressiveness ↔ expertise), safe (bounded loops)
  - Produces C (kernel module) and Why3 code to prove work conservation

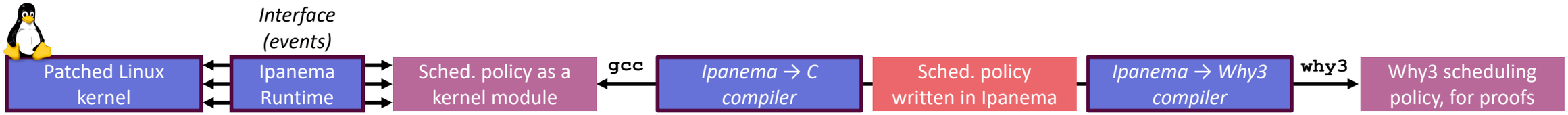
- **Problem:** full work conservation is too strong a property



If tasks wake up and block at the exact same time...

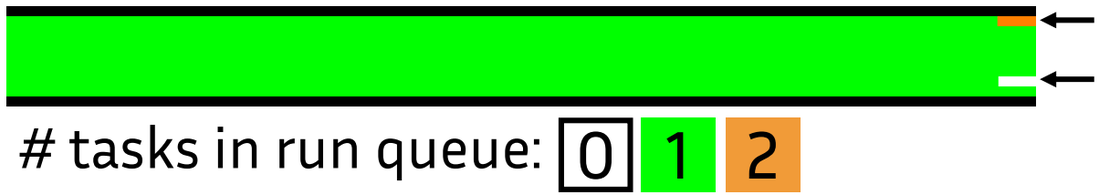
...work conservation not maintained before the scheduler can even react!

1. Task Scheduling: Ipanema [HotOS '17, EuroSys '20]



- **Ipanema:** a DSL to write scheduling policies with proven **work conservation**
  - Simple (expressiveness ↔ expertise), safe (bounded loops)
  - Produces C (kernel module) and Why3 code to prove work conservation

- **Problem:** full work conservation is too strong a property

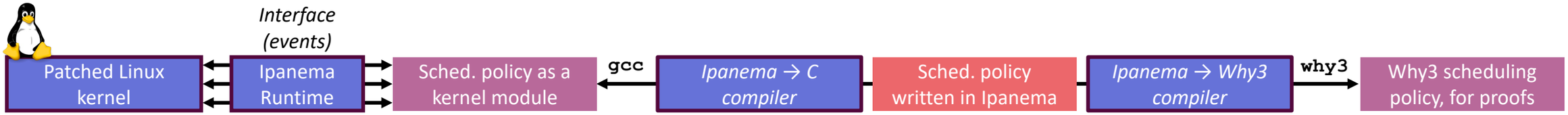


If tasks wake up and block at the exact same time...

...work conservation not maintained before the scheduler can even react!

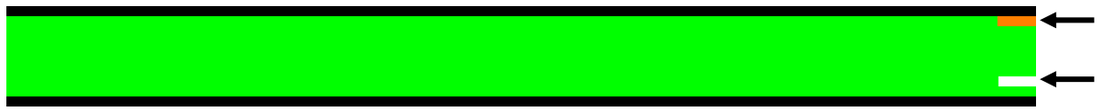
- **Solution:** Concurrent Work Conservation (CWC)

1. Task Scheduling: Ipanema [HotOS '17, EuroSys '20]



- **Ipanema:** a DSL to write scheduling policies with proven **work conservation**
  - Simple (expressiveness ↔ expertise), safe (bounded loops)
  - Produces C (kernel module) and Why3 code to prove work conservation

- **Problem:** full work conservation is too strong a property



tasks in run queue: 0 1 2

If tasks wake up and block at the exact same time...

...work conservation not maintained before the scheduler can even react!

- **Solution:** Concurrent Work Conservation (CWC)

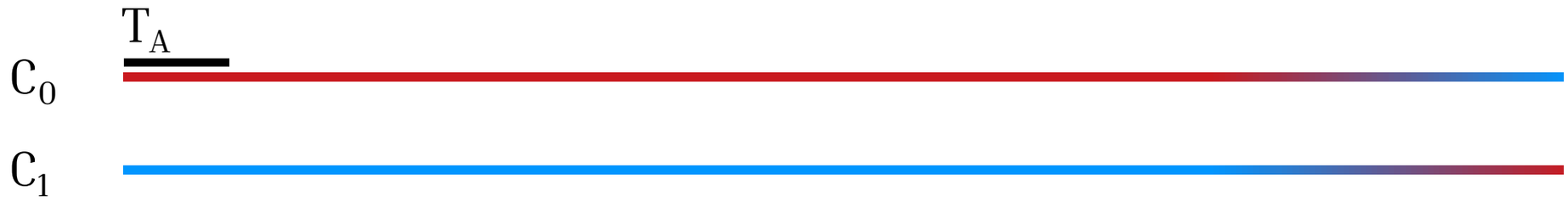
"Work-conserving decisions disregarding concurrent events"

1. Task Scheduling: Frequency Inversions [PLOS '19, EuroSys '20]

- Another problem we discovered: **frequency inversions**

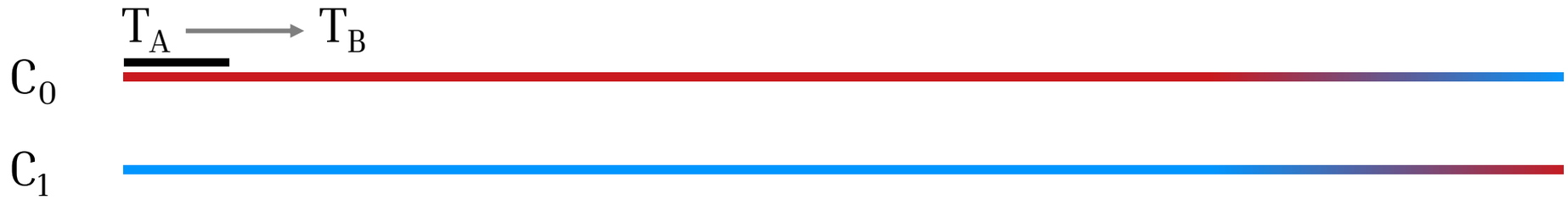
1. Task Scheduling: Frequency Inversions [PLOS '19, EuroSys '20]

- Another problem we discovered: **frequency inversions**
- A common scenario (forks, producer/consumer):



1. Task Scheduling: Frequency Inversions [PLOS '19, EuroSys '20]

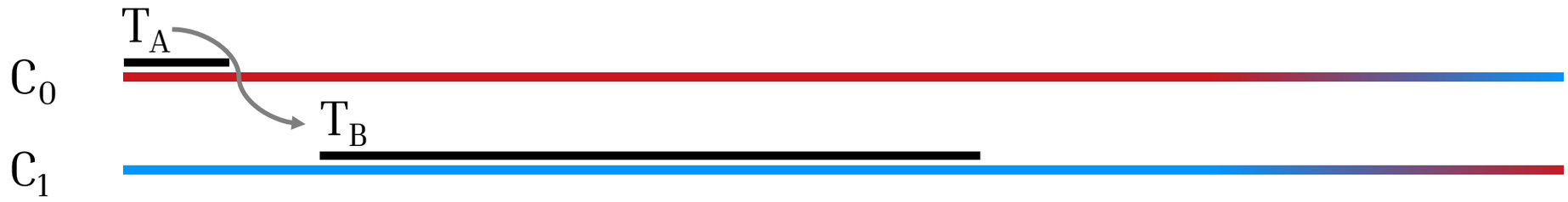
- Another problem we discovered: **frequency inversions**
- A common scenario (forks, producer/consumer):



T_A on C_0 wakes/creates T_B

1. Task Scheduling: Frequency Inversions [PLOS '19, EuroSys '20]

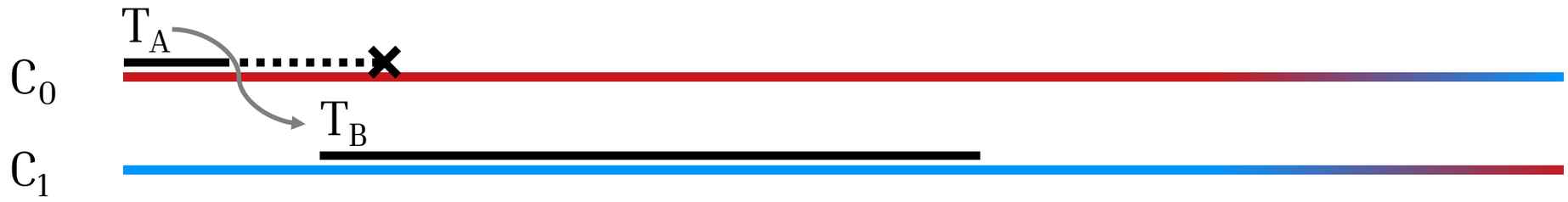
- Another problem we discovered: **frequency inversions**
- A common scenario (forks, producer/consumer):



T_A on C_0 wakes/creates T_B ⟶ T_B placed on C_1

1. Task Scheduling: Frequency Inversions [PLOS '19, EuroSys '20]

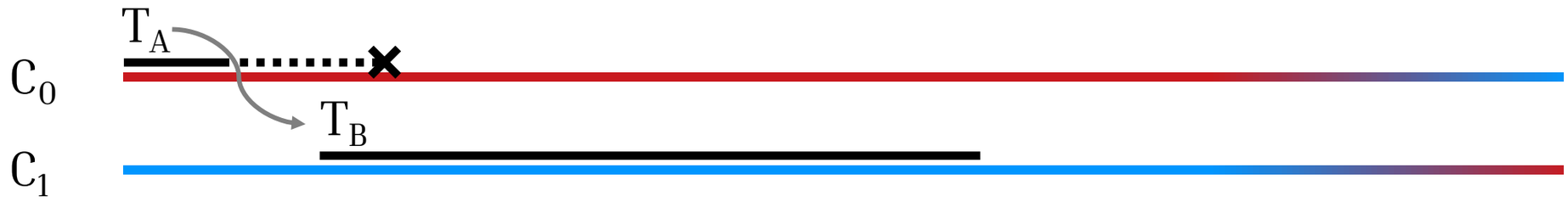
- Another problem we discovered: **frequency inversions**
- A common scenario (forks, producer/consumer):



T_A on C_0 wakes/creates T_B ⟶ T_B placed on C_1 ⟶ T_A blocks/terminates

1. Task Scheduling: Frequency Inversions [PLOS '19, EuroSys '20]

- Another problem we discovered: **frequency inversions**
- A common scenario (forks, producer/consumer):

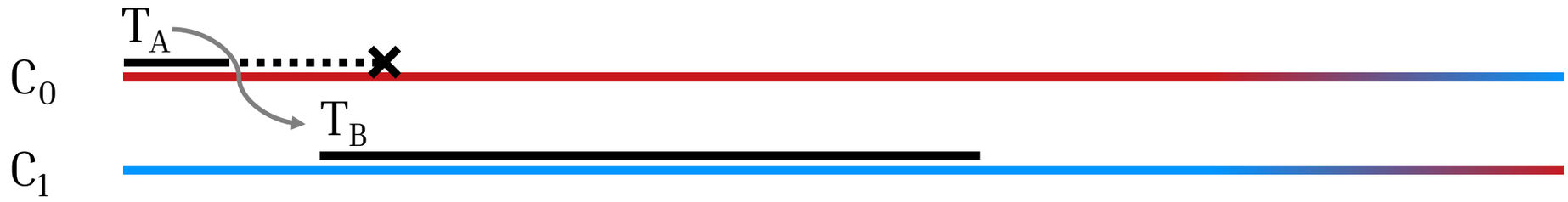


T_A on C_0 wakes/creates T_B ⟶ T_B placed on C_1 ⟶ T_A blocks/terminates

- C_0 runs at a high frequency (« hot »), even though it's not executing a task
- C_1 runs at a low frequency (« cold »), even though it's executing a task

1. Task Scheduling: Frequency Inversions [PLOS '19, EuroSys '20]

- Another problem we discovered: **frequency inversions**
- A common scenario (forks, producer/consumer):

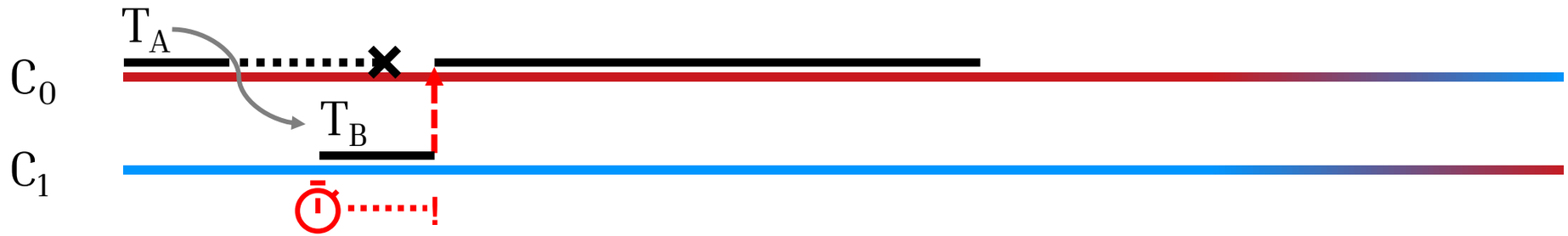


T_A on C_0 wakes/creates T_B ⟶ T_B placed on C_1 ⟶ T_A blocks/terminates

- C_0 runs at a high frequency (« hot »), even though it's not executing a task
- C_1 runs at a low frequency (« cold »), even though it's executing a task
- **Problem:** impossible to know a priori whether T_A will block/terminate

1. Task Scheduling: Frequency Inversions [PLOS '19, EuroSys '20]

- Another problem we discovered: **frequency inversions**
- A common scenario (forks, producer/consumer):

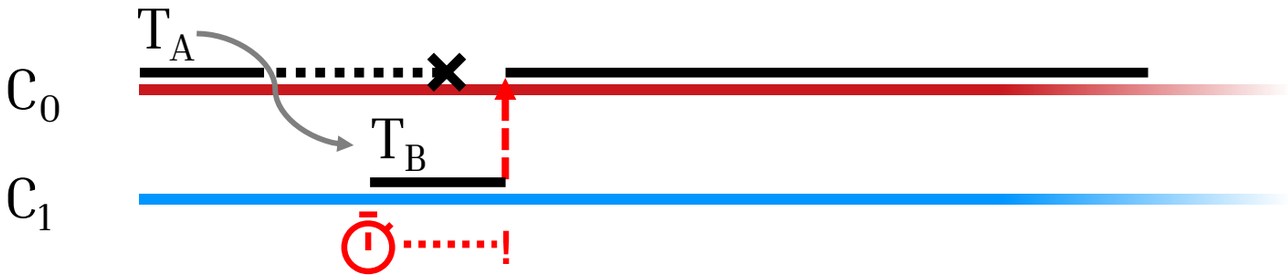


T_A on C_0 wakes/creates T_B ⟶ T_B placed on C_1 ⟶ T_A blocks/terminates

- C_0 runs at a high frequency (« hot »), even though it's not executing a task
- C_1 runs at a low frequency (« cold »), even though it's executing a task
- **Problem:** impossible to know a priori whether T_A will block/terminate
  - **Simple solution:** arm a short timer, if T_A blocked/terminated, move T_B to C_0

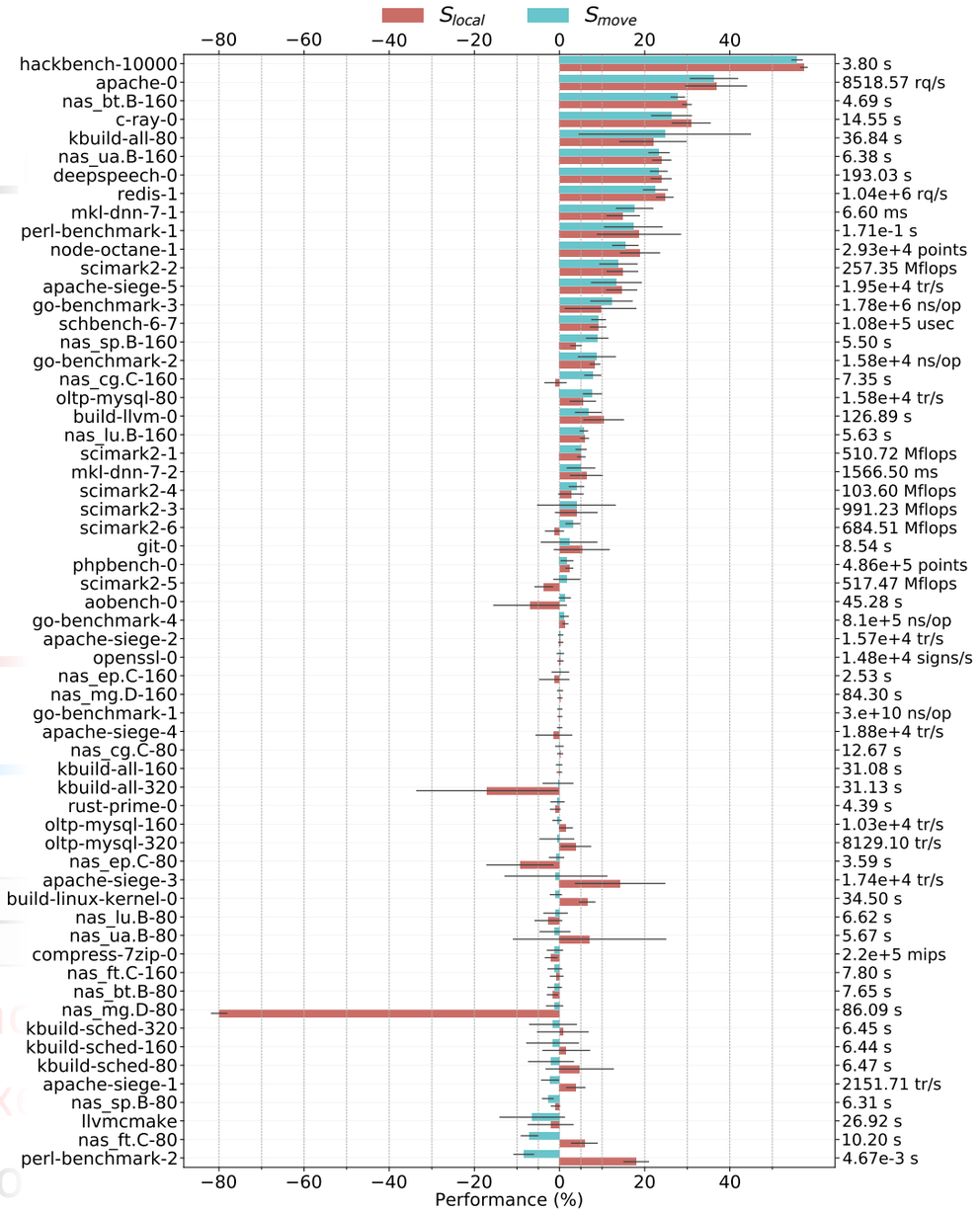
1. Task Scheduling: Frequency Inversions [PL

- Another problem we discovered: **frequency inversions**
- A common scenario (forks, producer/consumer):

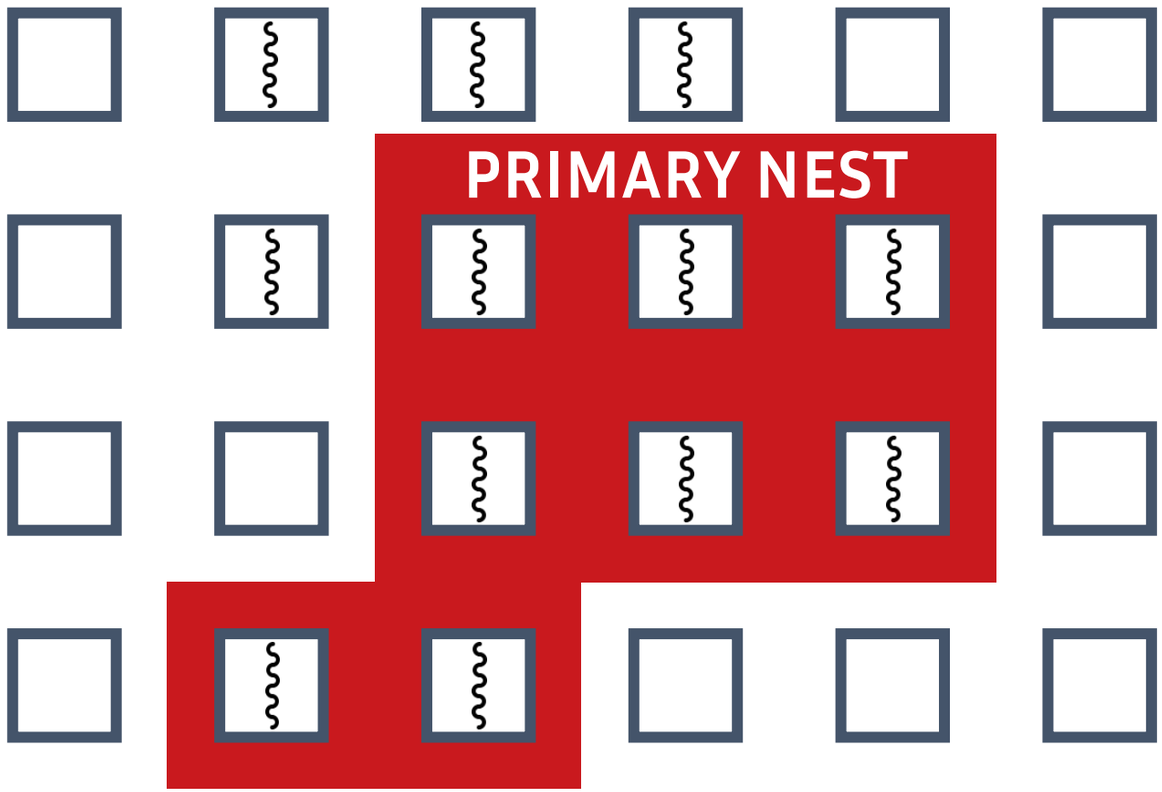


T_A on C_0 wakes/creates T_B ⟶ T_B placed on C_1

- C_0 runs at a high frequency (« hot »), even though it's n
- C_1 runs at a low frequency (« cold »), even though it's ex
- **Problem:** impossible to know a priori whether T_A will blo
  - **Simple solution:** arm a short timer, if T_A blocked/terminated, move T_B to C_0

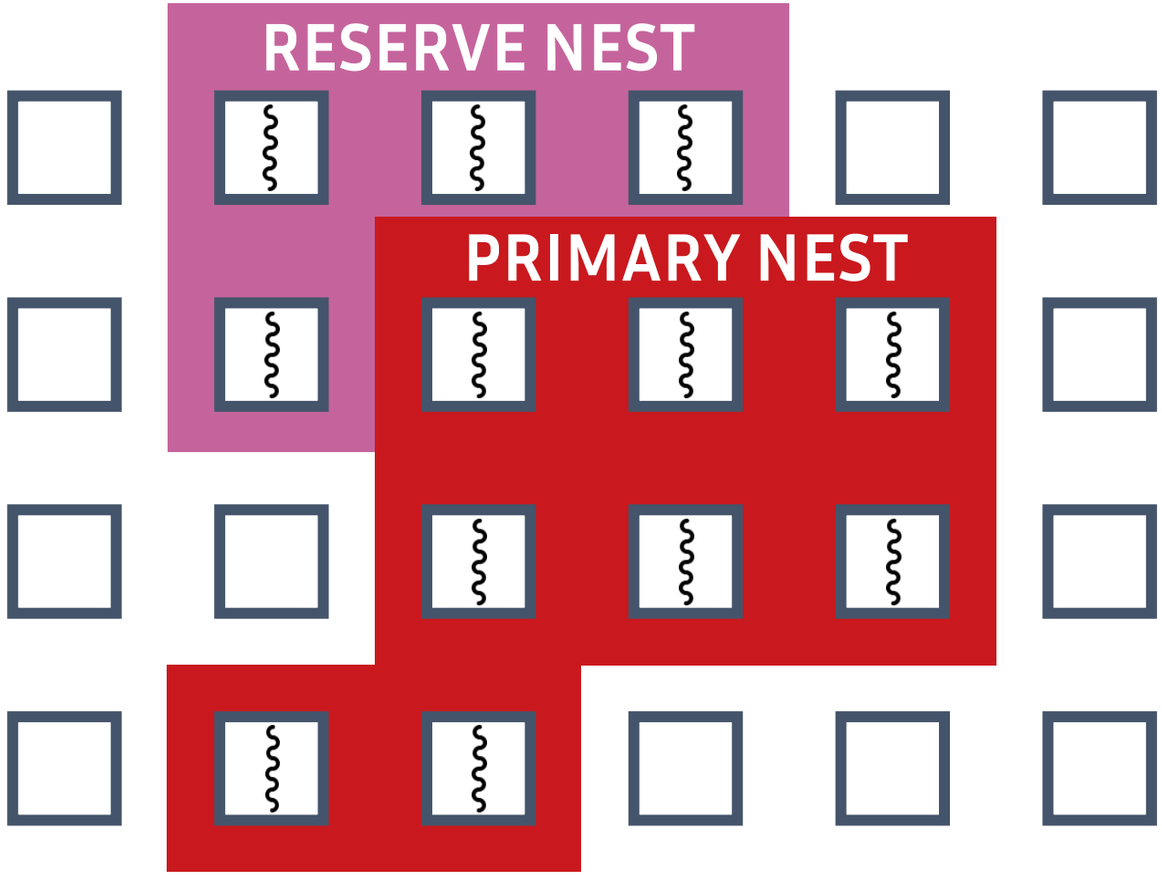


1. Task Scheduling: Nest [EuroSys '22]



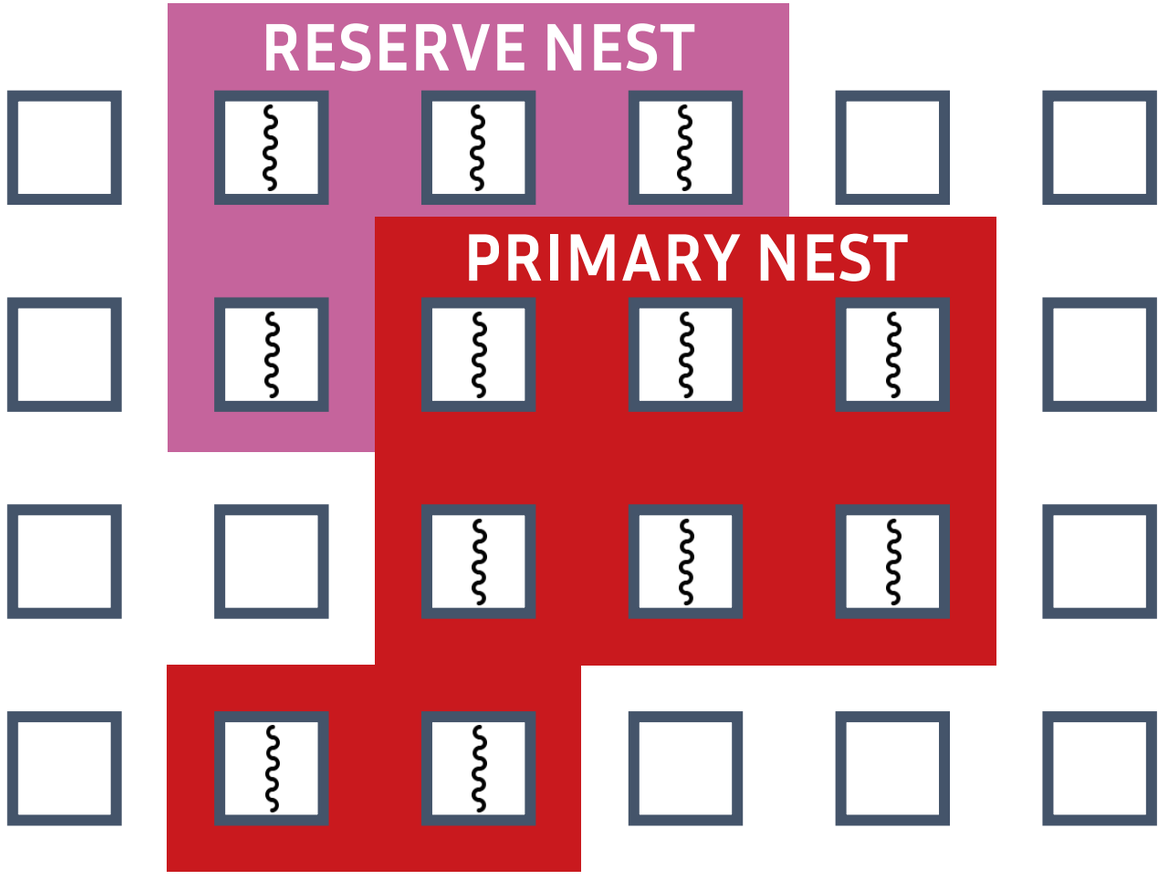
- **Idea: keep threads inside "warm" nests!**
- **Cores in the primary nest:**
  - Currently/recently used, and
  - Expected to be useful in the near future

1. Task Scheduling: Nest [EuroSys '22]



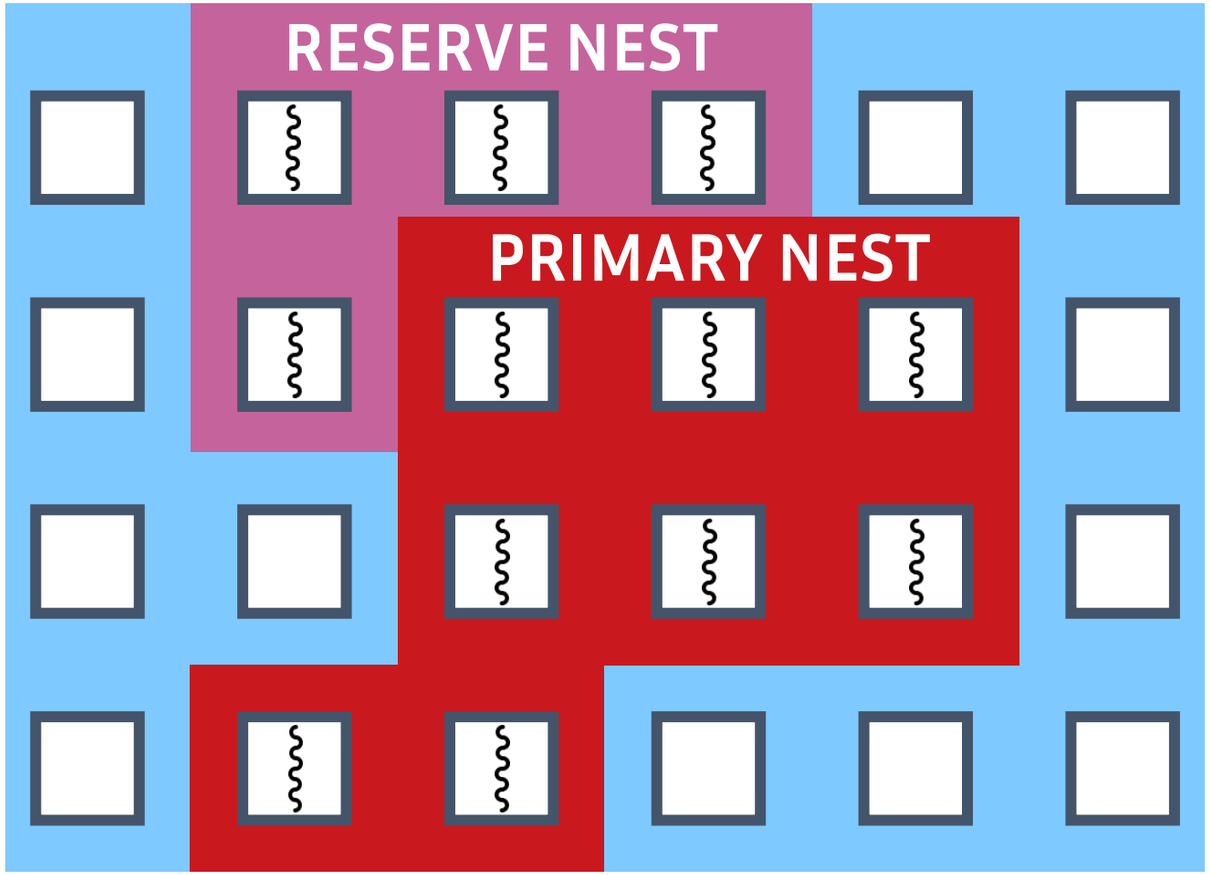
- **Idea: keep threads inside "warm" nests!**
- **Cores in the primary nest:**
  - Currently/recently used, and
  - Expected to be useful in the near future
- **Cores in the reserve nest:**
  - Previously in the primary nest, but not used in a while, or
  - Selected recently by Linux, but not yet deemed necessary in the primary nest

1. Task Scheduling: Nest [EuroSys '22]



- **Idea: keep threads inside "warm" nests!**
- **Cores in the primary nest:**
  - Currently/recently used, and
  - Expected to be useful in the near future
- **Cores in the reserve nest:**
  - Previously in the primary nest, but not used in a while, or
  - Selected recently by Linux, but not yet deemed necessary in the primary nest
- **The primary nest and the reserve nest grow and shrink automatically**

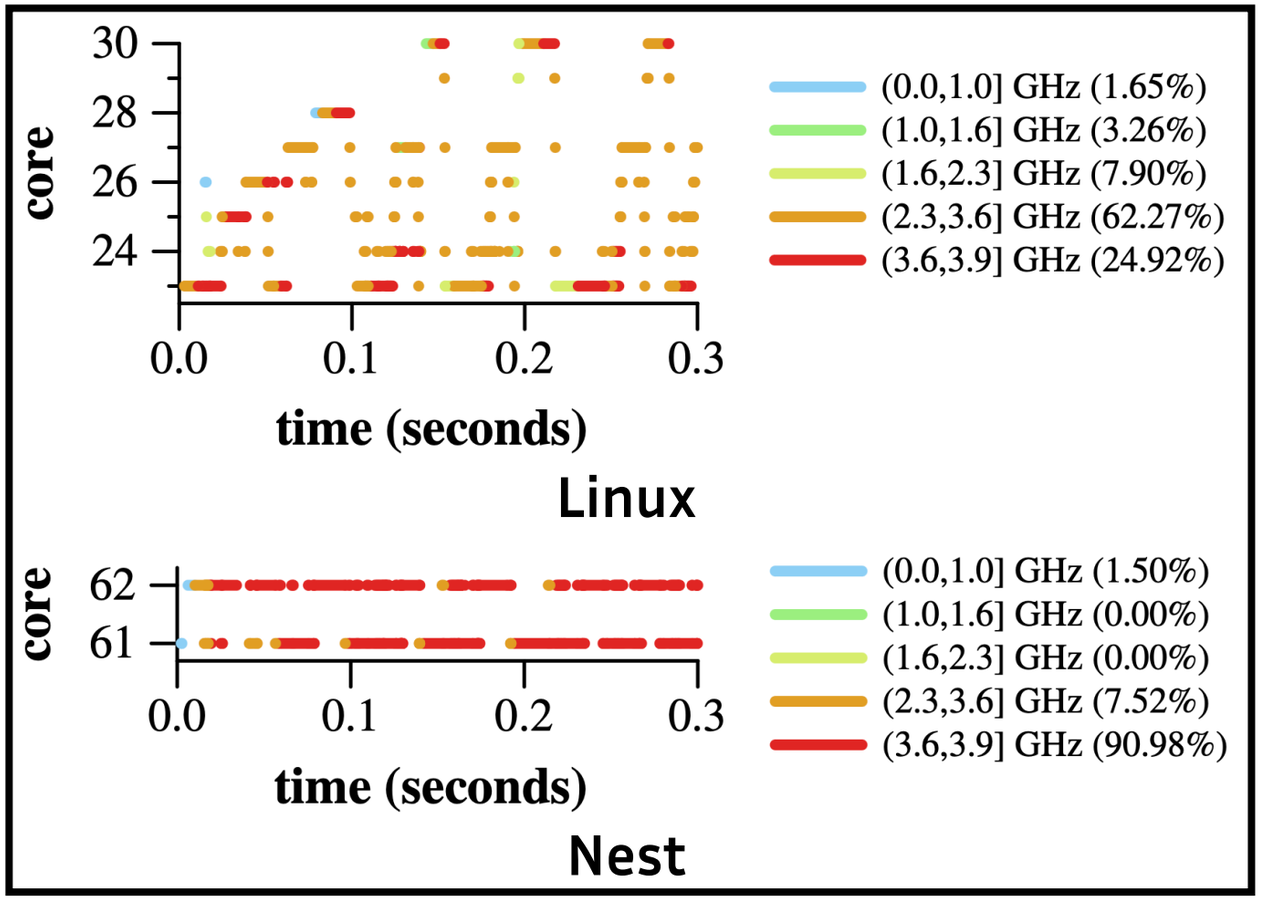
1. Task Scheduling: Nest [EuroSys '22]



Other cores idle, allowing the nests to reach turbo frequencies...

- **Idea: keep threads inside "warm" nests!**
- **Cores in the primary nest:**
  - Currently/recently used, and
  - Expected to be useful in the near future
- **Cores in the reserve nest:**
  - Previously in the primary nest, but not used in a while, or
  - Selected recently by Linux, but not yet deemed necessary in the primary nest
- **The primary nest and the reserve nest grow and shrink automatically**

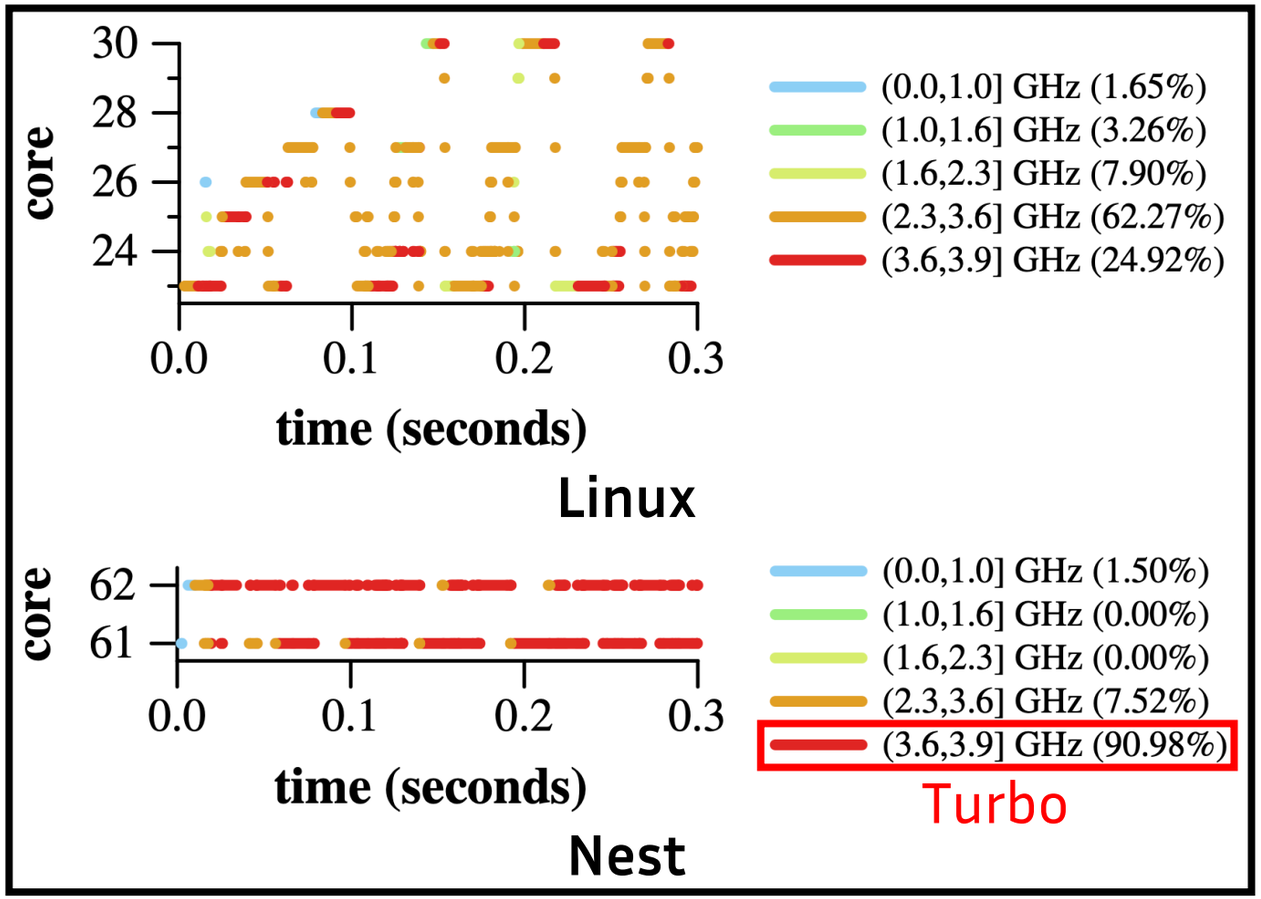
1. Task Scheduling: Nest [EuroSys '22]



Other cores idle, allowing the nests to reach turbo frequencies...

- **Idea: keep threads inside "warm" nests!**
- **Cores in the primary nest:**
  - Currently/recently used, and
  - Expected to be useful in the near future
- **Cores in the reserve nest:**
  - Previously in the primary nest, but not used in a while, or
  - Selected recently by Linux, but not yet deemed necessary in the primary nest
- **The primary nest and the reserve nest grow and shrink automatically**

1. Task Scheduling: Nest [EuroSys '22]



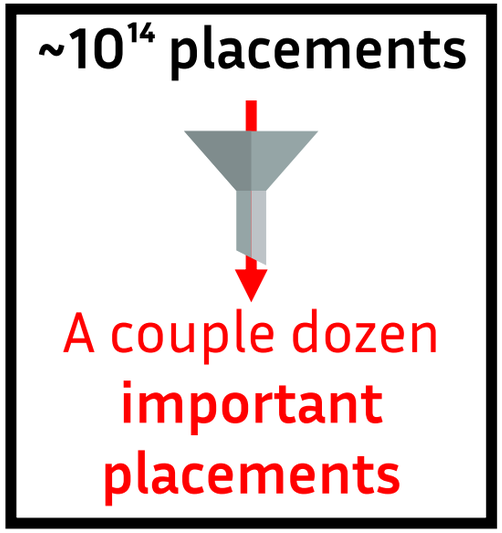
Other cores idle, allowing the nests to reach turbo frequencies...

- **Idea: keep threads inside "warm" nests!**
- **Cores in the primary nest:**
  - Currently/recently used, and
  - Expected to be useful in the near future
- **Cores in the reserve nest:**
  - Previously in the primary nest, but not used in a while, or
  - Selected recently by Linux, but not yet deemed necessary in the primary nest
- **The primary nest and the reserve nest grow and shrink automatically**

1. Task Scheduling: Other and Future Work

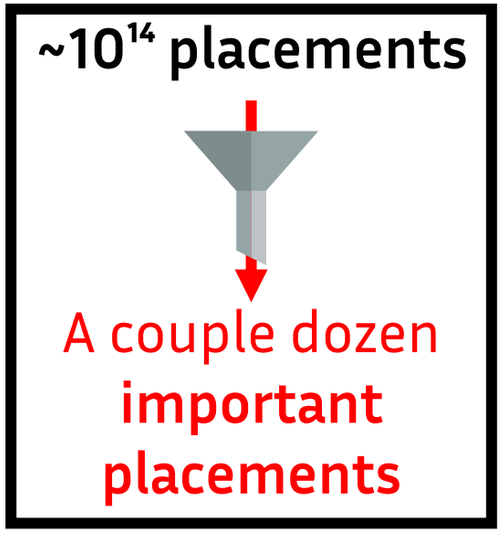
- **Other work:** NUMA placements [USENIX ATC '18]
  - **Unlike other presented works:** static, userspace container placement

1. Task Scheduling: Other and Future Work



- **Other work:** NUMA placements [USENIX ATC '18]
  - **Unlike other presented works:** static, userspace container placement
  - "Scheduling concerns" prune the search space (L2/SMT, L3/NUMA, interconnects)

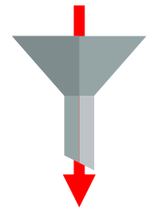
1. Task Scheduling: Other and Future Work



- **Other work:** NUMA placements [USENIX ATC '18]
  - **Unlike other presented works:** static, userspace container placement
  - "Scheduling concerns" prune the search space (L2/SMT, L3/NUMA, interconnects)
  - Random Forest model predicts performance from 2 short probe runs

1. Task Scheduling: Other and Future Work

~10^{14} placements



A couple dozen **important placements**

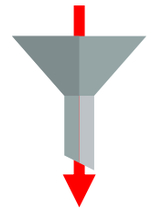
- **Other work:** NUMA placements [USENIX ATC '18]
  - **Unlike other presented works:** static, userspace container placement
  - "Scheduling concerns" prune the search space (L2/SMT, L3/NUMA, interconnects)
  - Random Forest model predicts performance from 2 short probe runs
- **Future work:**

Scheduler fuzzing

Detect work conservation bugs

1. Task Scheduling: Other and Future Work

~10^{14} placements



A couple dozen **important placements**

- **Other work:** NUMA placements [USENIX ATC '18]
  - **Unlike other presented works:** static, userspace container placement
  - "Scheduling concerns" prune the search space (L2/SMT, L3/NUMA, interconnects)
  - Random Forest model predicts performance from 2 short probe runs
- **Future work:**

Scheduler fuzzing

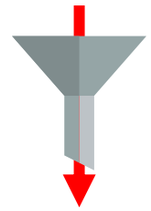
Detect work conservation bugs

Scheduler benchmarks

Synthetic workloads using e.g., LLMs

1. Task Scheduling: Other and Future Work

~10^{14} placements



A couple dozen **important placements**

- **Other work:** NUMA placements [USENIX ATC '18]
  - **Unlike other presented works:** static, userspace container placement
  - "Scheduling concerns" prune the search space (L2/SMT, L3/NUMA, interconnects)
  - Random Forest model predicts performance from 2 short probe runs
- **Future work:**

Scheduler fuzzing

Detect work conservation bugs

Scheduler benchmarks

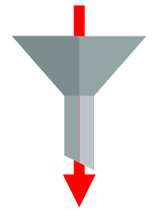
Synthetic workloads using e.g., LLMs

Virtualized scheduling

Could the host schedule guest tasks?

1. Task Scheduling: Other and Future Work

~10^{14} placements



A couple dozen **important placements**

- **Other work:** NUMA placements [USENIX ATC '18]
  - **Unlike other presented works:** static, userspace container placement
  - "Scheduling concerns" prune the search space (L2/SMT, L3/NUMA, interconnects)
  - Random Forest model predicts performance from 2 short probe runs
- **Future work:**

Scheduler fuzzing

Detect work conservation bugs

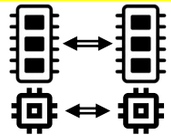
Scheduler benchmarks

Synthetic workloads using e.g., LLMs

Virtualized scheduling

Could the host schedule guest tasks?

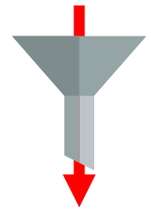
Better AutoNUMA



E.g., make it more reactive w/ Intel MPK

1. Task Scheduling: Other and Future Work

~10^{14} placements



A couple dozen **important placements**

- **Other work:** NUMA placements [USENIX ATC '18]
  - **Unlike other presented works:** static, userspace container placement
  - "Scheduling concerns" prune the search space (L2/SMT, L3/NUMA, interconnects)
  - Random Forest model predicts performance from 2 short probe runs
- **Future work:**

Scheduler fuzzing

Detect work conservation bugs

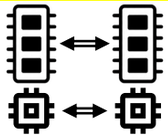
Scheduler benchmarks

Synthetic workloads using e.g., LLMs

Virtualized scheduling

Could the host schedule guest tasks?

Better AutoNUMA



E.g., make it more reactive w/ Intel MPK

Scheduler verification

E.g., generate loop invariants (w/ Keisuke)

2. Task Synchronization: FlexGuard [SOSP '25] *(Victor Laforet)*

- **Blocking locks:** block if lock already acquired

vs.

- **Spinlocks:** busy-wait instead

2. Task Synchronization: FlexGuard [SOSP '25] (*Victor Laforet*)

- **Blocking locks:** block if lock already acquired
  - E.g., POSIX (`pthread_mutex_lock()`)

vs.

- **Spinlocks:** busy-wait instead

2. Task Synchronization: FlexGuard [SOSP '25] (*Victor Laforet*)

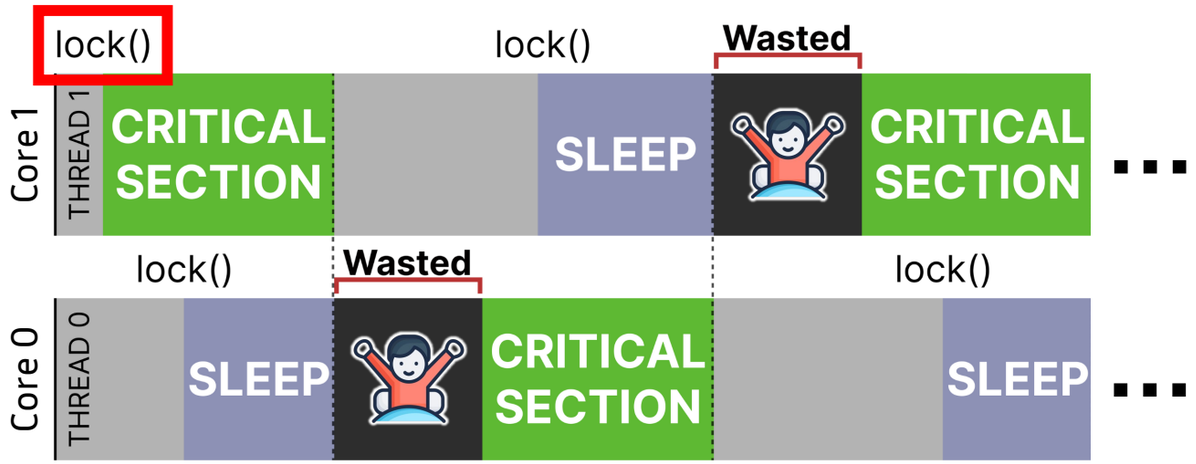
- **Blocking locks:** block if lock already acquired
  - E.g., POSIX (`pthread_mutex_lock()`)
  - **Problem:** costly wakeups ⇒ **slow**!

vs.

- **Spinlocks:** busy-wait instead

2. Task Synchronization: FlexGuard [SOSP '25] *(Victor Laforet)*

- **Blocking locks:** block if lock already acquired
  - E.g., POSIX (`pthread_mutex_lock()`)
  - **Problem:** costly wakeups ⇒ **slow**!

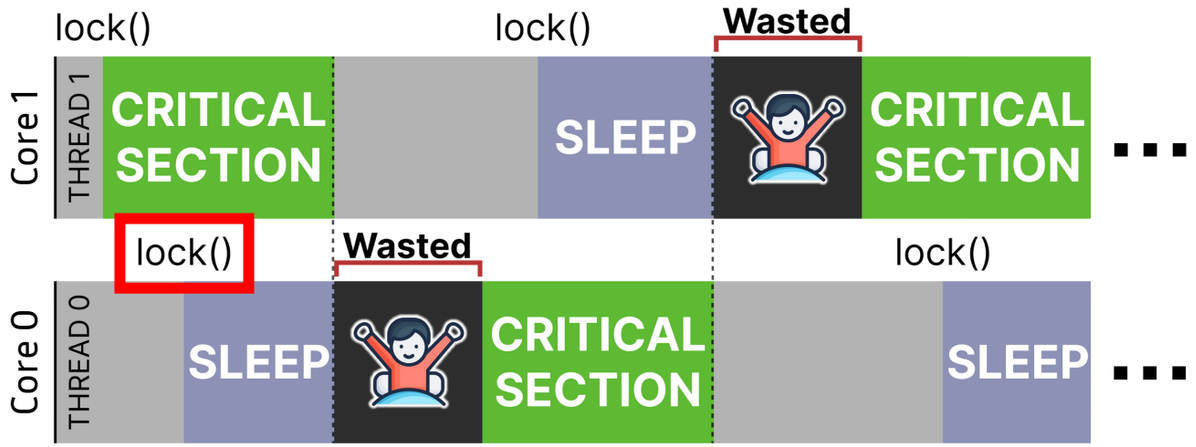


VS.

- **Spinlocks:** busy-wait instead

2. Task Synchronization: FlexGuard [SOSP '25] *(Victor Laforet)*

- **Blocking locks:** block if lock already acquired
  - E.g., POSIX (`pthread_mutex_lock()`)
  - **Problem:** costly wakeups ⇒ **slow**!

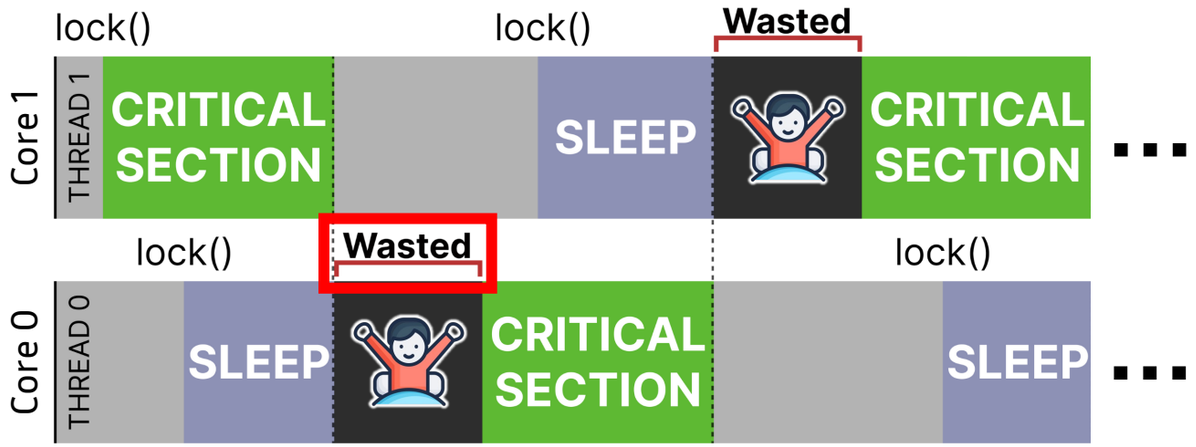


VS.

- **Spinlocks:** busy-wait instead

2. Task Synchronization: FlexGuard [SOSP '25] *(Victor Laforet)*

- **Blocking locks:** block if lock already acquired
  - E.g., POSIX (`pthread_mutex_lock()`)
  - **Problem:** costly wakeups ⇒ **slow**!

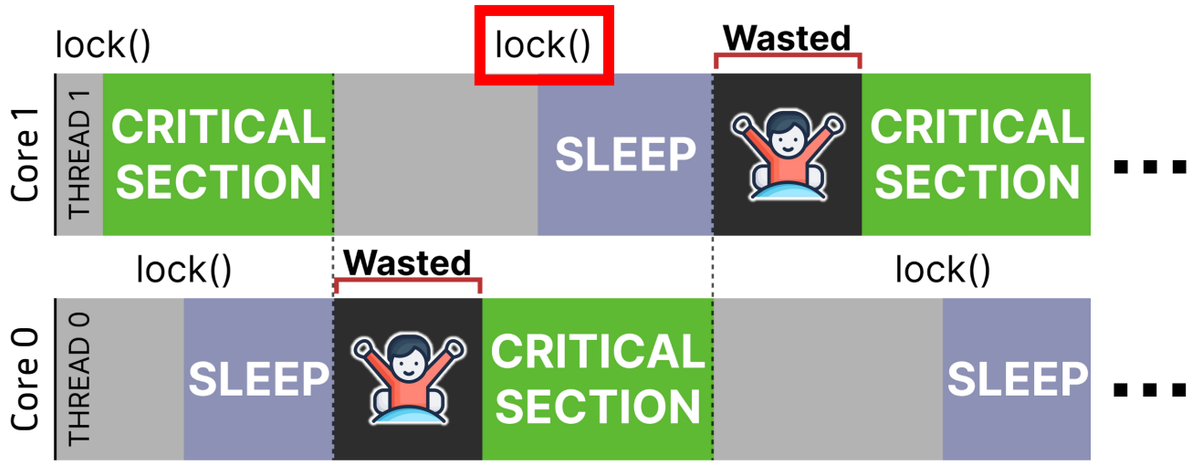


VS.

- **Spinlocks:** busy-wait instead

2. Task Synchronization: FlexGuard [SOSP '25] *(Victor Laforet)*

- **Blocking locks:** block if lock already acquired
  - E.g., POSIX (`pthread_mutex_lock()`)
  - **Problem:** costly wakeups ⇒ **slow**!

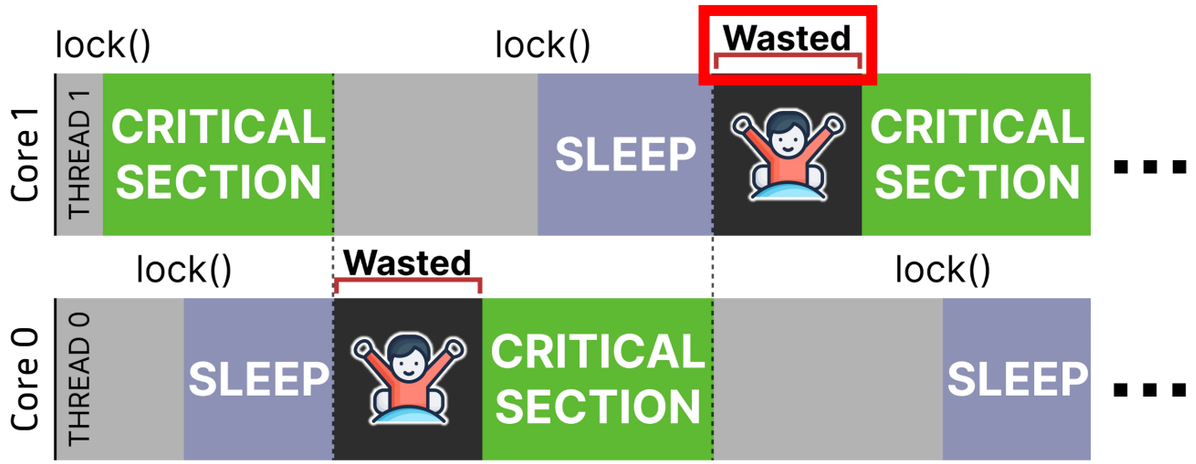


VS.

- **Spinlocks:** busy-wait instead

2. Task Synchronization: FlexGuard [SOSP '25] *(Victor Laforet)*

- **Blocking locks:** block if lock already acquired
  - E.g., POSIX (`pthread_mutex_lock()`)
  - **Problem:** costly wakeups ⇒ **slow**!

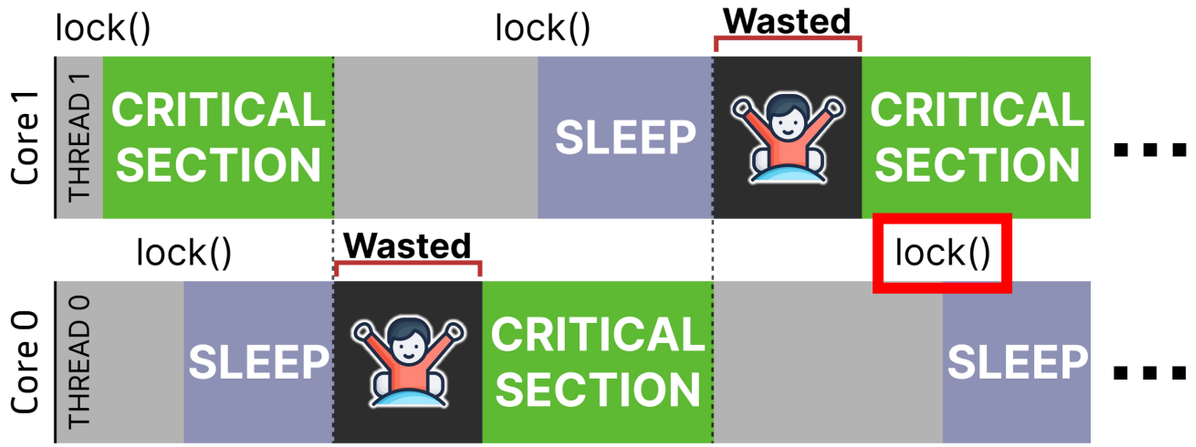


VS.

- **Spinlocks:** busy-wait instead

2. Task Synchronization: FlexGuard [SOSP '25] *(Victor Laforet)*

- **Blocking locks:** block if lock already acquired
  - E.g., POSIX (`pthread_mutex_lock()`)
  - **Problem:** costly wakeups ⇒ **slow**!

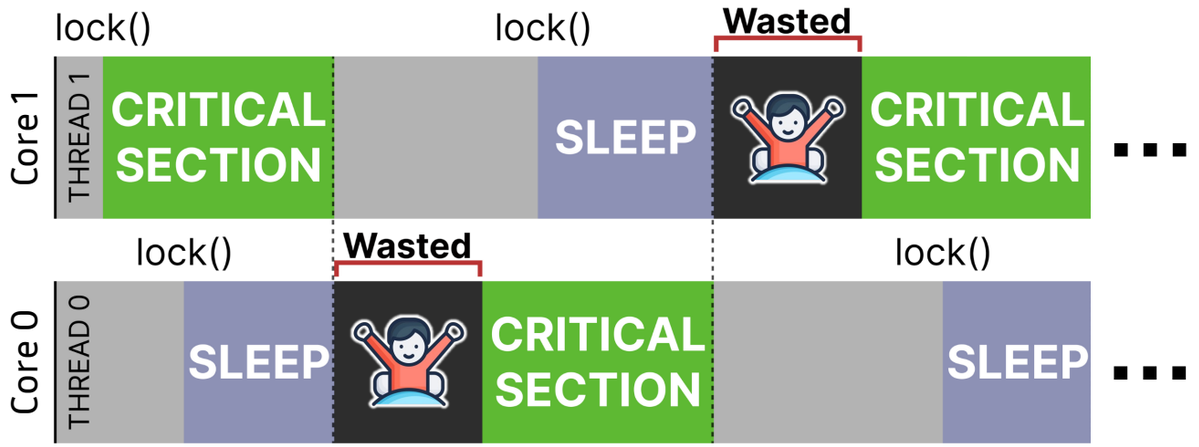


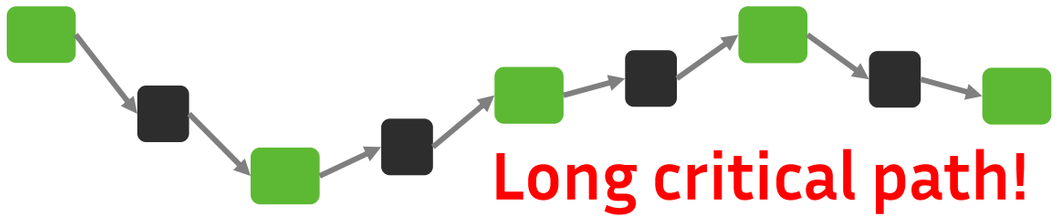
VS.

- **Spinlocks:** busy-wait instead

2. Task Synchronization: FlexGuard [SOSP '25] *(Victor Laforet)*

- **Blocking locks:** block if lock already acquired
  - E.g., POSIX (`pthread_mutex_lock()`)
  - **Problem:** costly wakeups ⇒ **slow**!





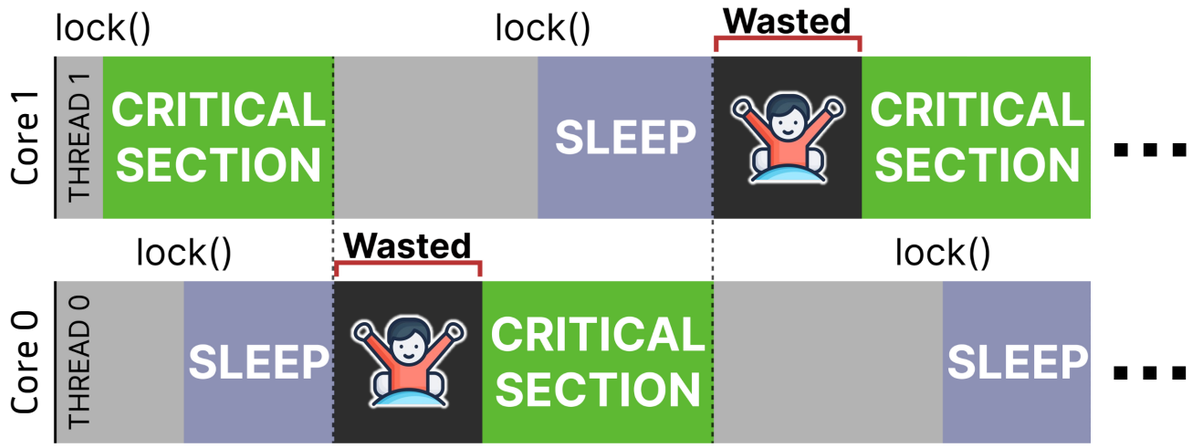
Long critical path!

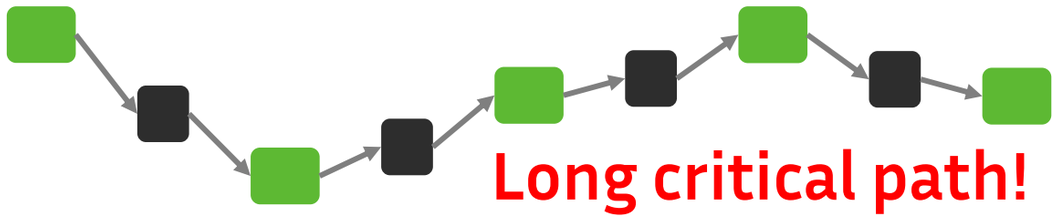
VS.

- **Spinlocks:** busy-wait instead

2. Task Synchronization: FlexGuard [SOSP '25] *(Victor Laforet)*

- **Blocking locks:** block if lock already acquired
  - E.g., POSIX (`pthread_mutex_lock()`)
  - **Problem:** costly wakeups ⇒ **slow**!





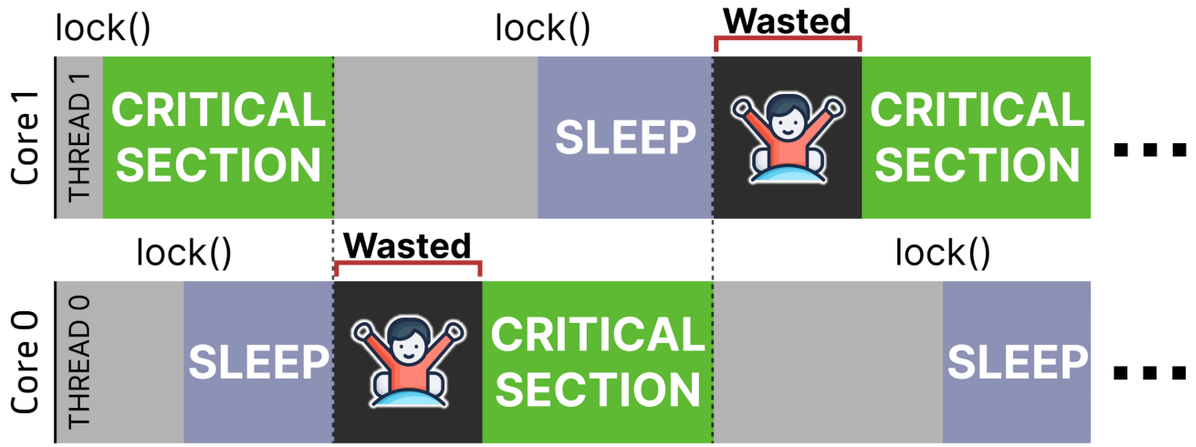
Long critical path!

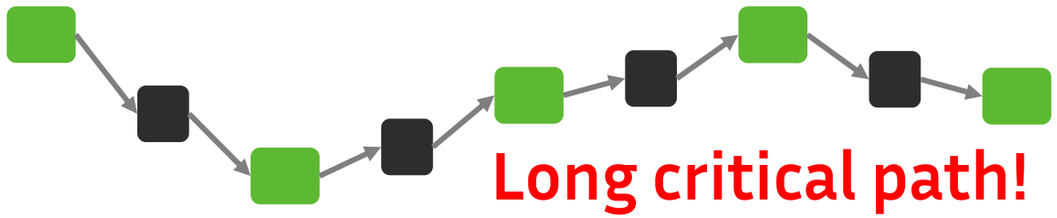
VS.

- **Spinlocks:** busy-wait instead
  - Many variants exist (with queues, tickets…)

2. Task Synchronization: FlexGuard [SOSP '25] *(Victor Laforet)*

- **Blocking locks:** block if lock already acquired
  - E.g., POSIX (`pthread_mutex_lock()`)
  - **Problem:** costly wakeups ⇒ **slow**!





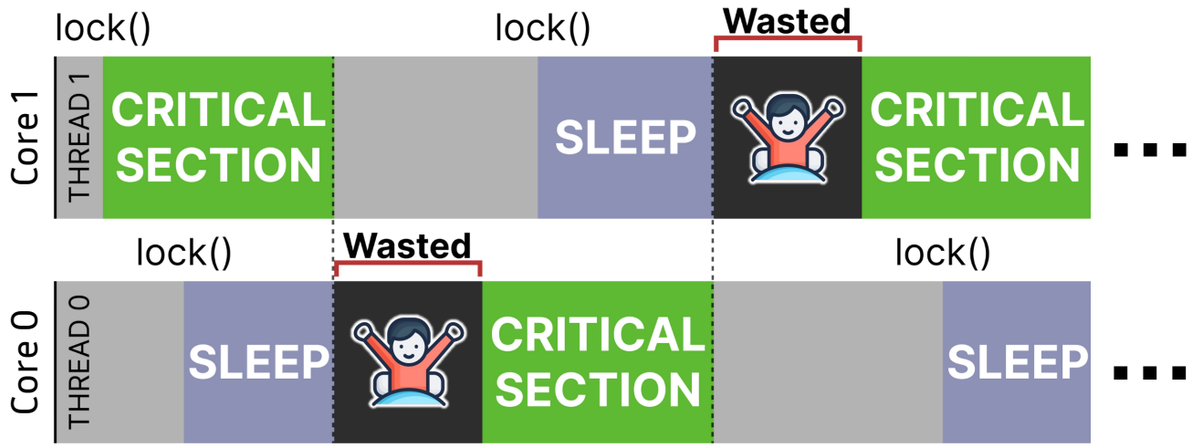
Long critical path!

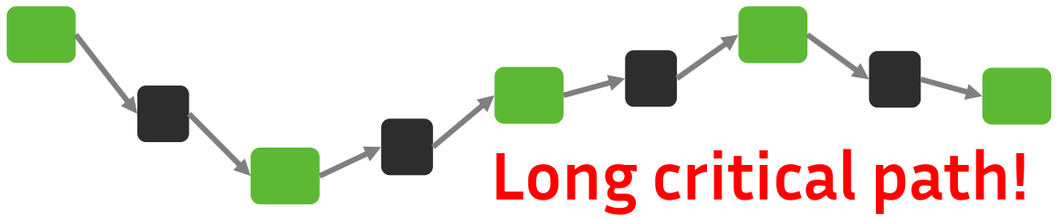
VS.

- **Spinlocks:** busy-wait instead
  - Many variants exist (with queues, tickets…)
  - **Fast** lock handovers: one cache miss!

2. Task Synchronization: FlexGuard [SOSP '25] *(Victor Laforet)*

- **Blocking locks:** block if lock already acquired
  - E.g., POSIX (`pthread_mutex_lock()`)
  - **Problem:** costly wakeups ⇒ **slow**!

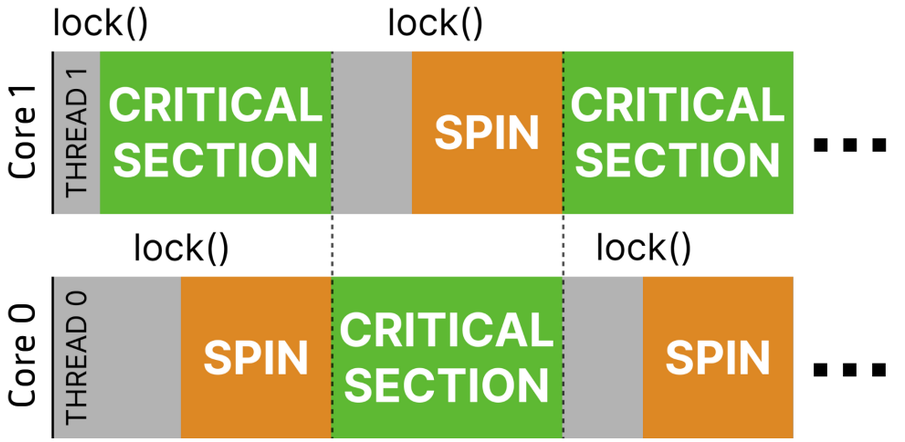




Long critical path!

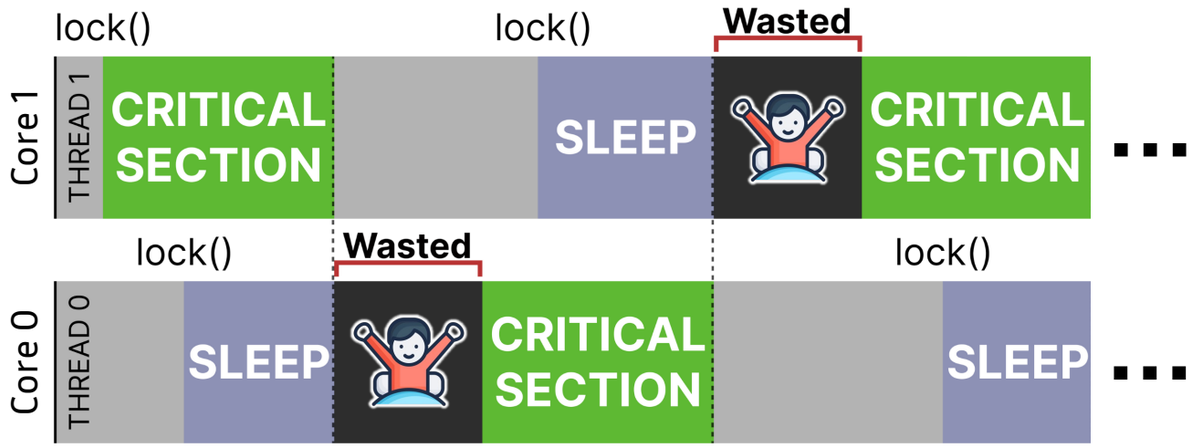
VS.

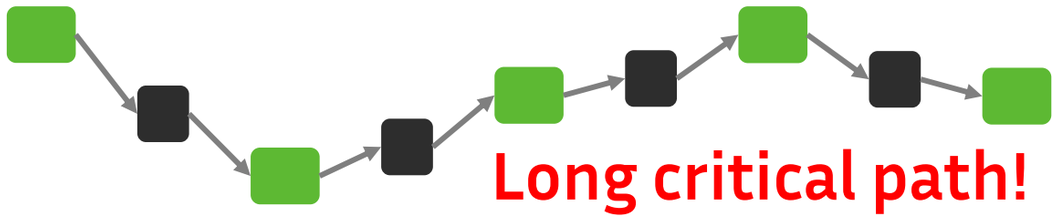
- **Spinlocks:** busy-wait instead
  - Many variants exist (with queues, tickets…)
  - **Fast** lock handovers: one cache miss!



2. Task Synchronization: FlexGuard [SOSP '25] *(Victor Laforet)*

- **Blocking locks:** block if lock already acquired
  - E.g., POSIX (`pthread_mutex_lock()`)
  - **Problem:** costly wakeups ⇒ **slow**!

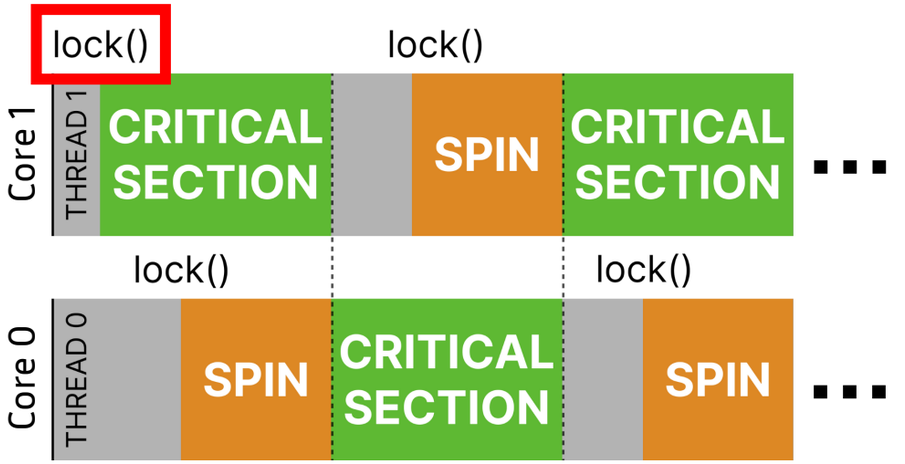




Long critical path!

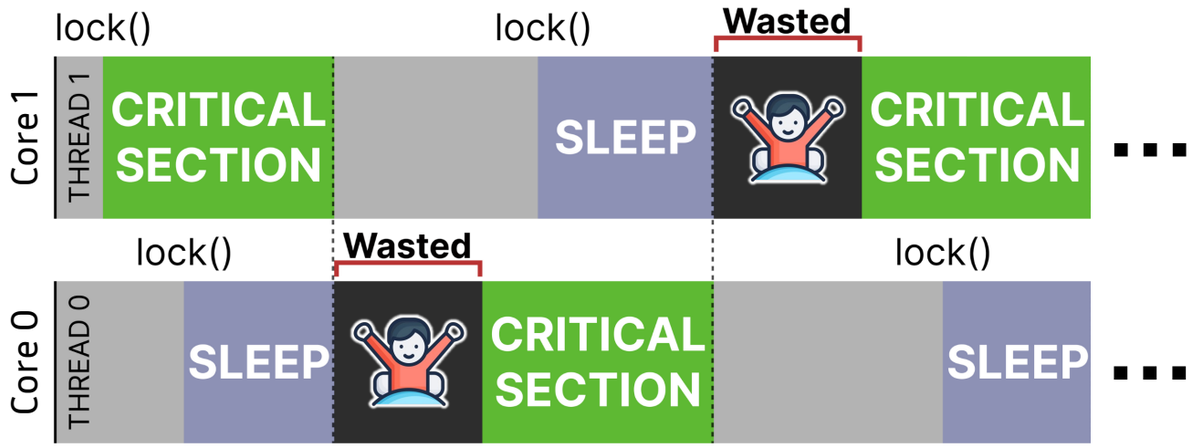
VS.

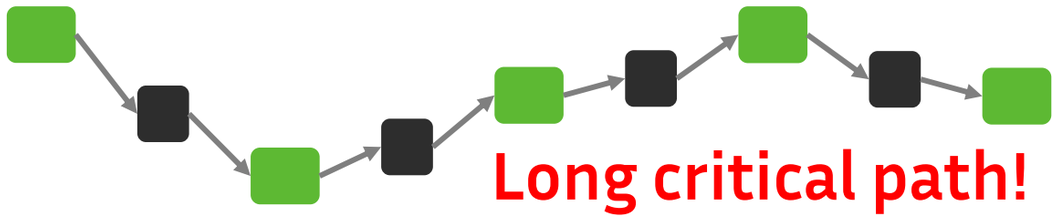
- **Spinlocks:** busy-wait instead
  - Many variants exist (with queues, tickets…)
  - **Fast** lock handovers: one cache miss!



2. Task Synchronization: FlexGuard [SOSP '25] *(Victor Laforet)*

- **Blocking locks:** block if lock already acquired
  - E.g., POSIX (`pthread_mutex_lock()`)
  - **Problem:** costly wakeups ⇒ **slow**!

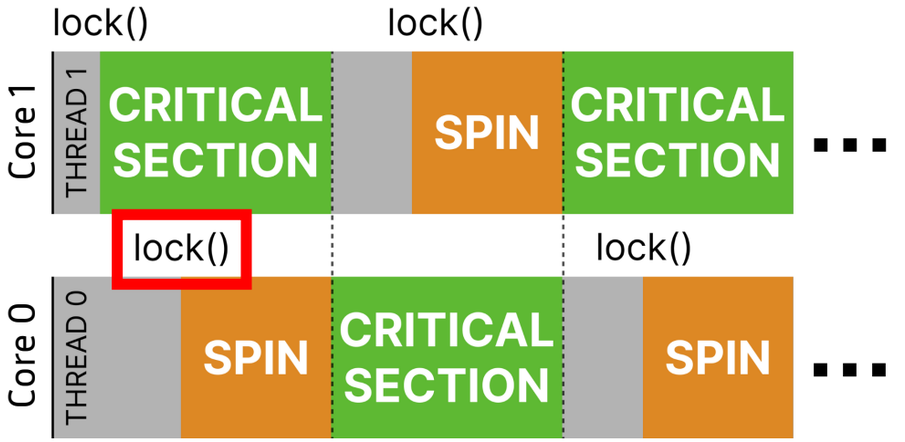




Long critical path!

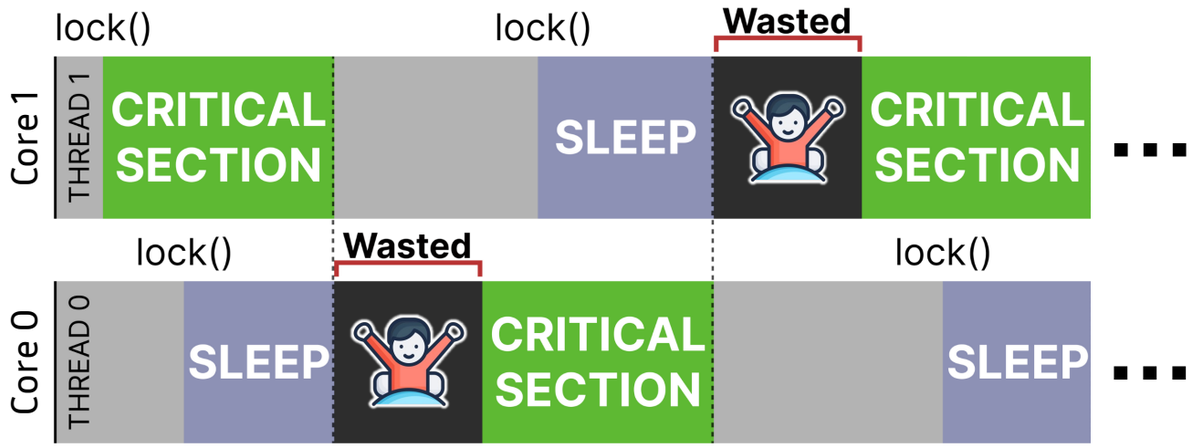
VS.

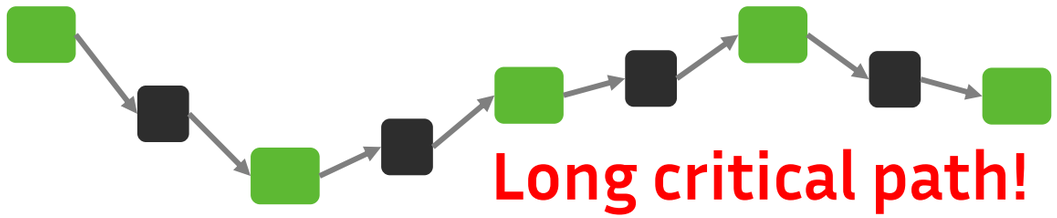
- **Spinlocks:** busy-wait instead
  - Many variants exist (with queues, tickets…)
  - **Fast** lock handovers: one cache miss!



2. Task Synchronization: FlexGuard [SOSP '25] (*Victor Laforet*)

- **Blocking locks:** block if lock already acquired
  - E.g., POSIX (`pthread_mutex_lock()`)
  - **Problem:** costly wakeups ⇒ **slow**!

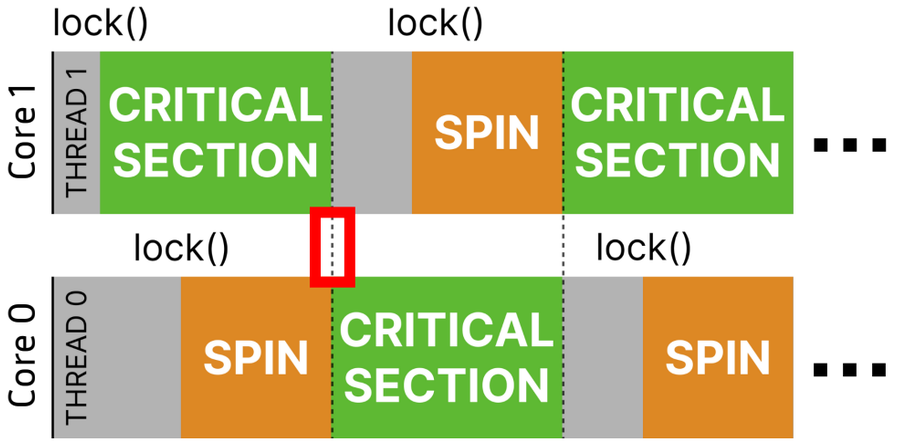




Long critical path!

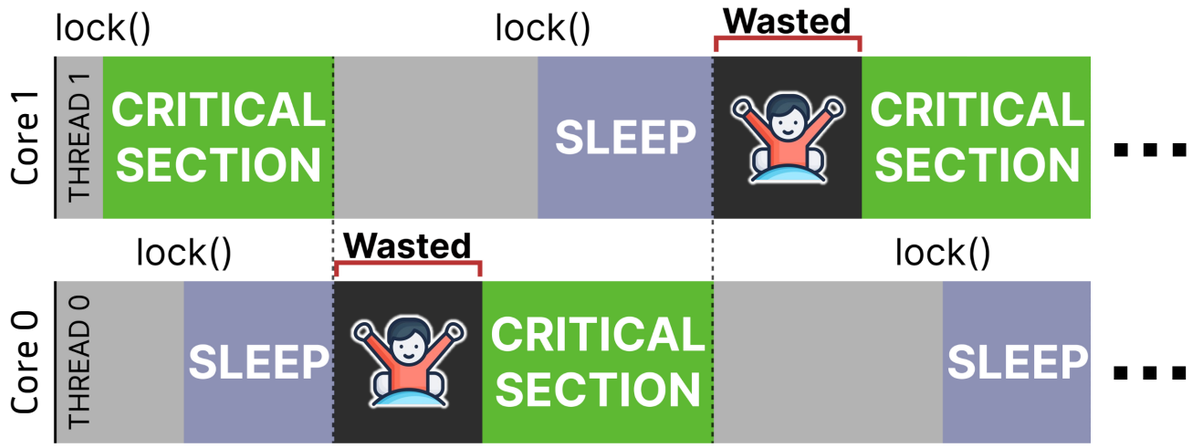
VS.

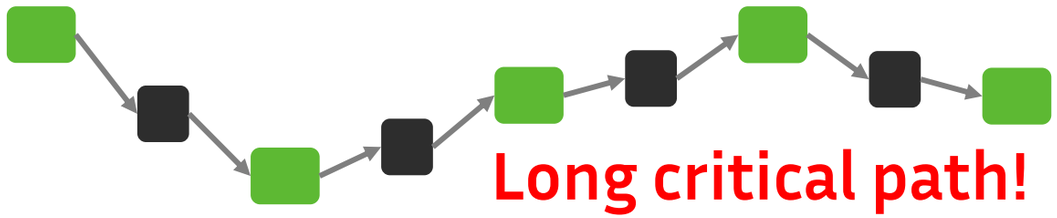
- **Spinlocks:** busy-wait instead
  - Many variants exist (with queues, tickets…)
  - **Fast** lock handovers: one cache miss!



2. Task Synchronization: FlexGuard [SOSP '25] *(Victor Laforet)*

- **Blocking locks:** block if lock already acquired
  - E.g., POSIX (`pthread_mutex_lock()`)
  - **Problem:** costly wakeups ⇒ **slow**!

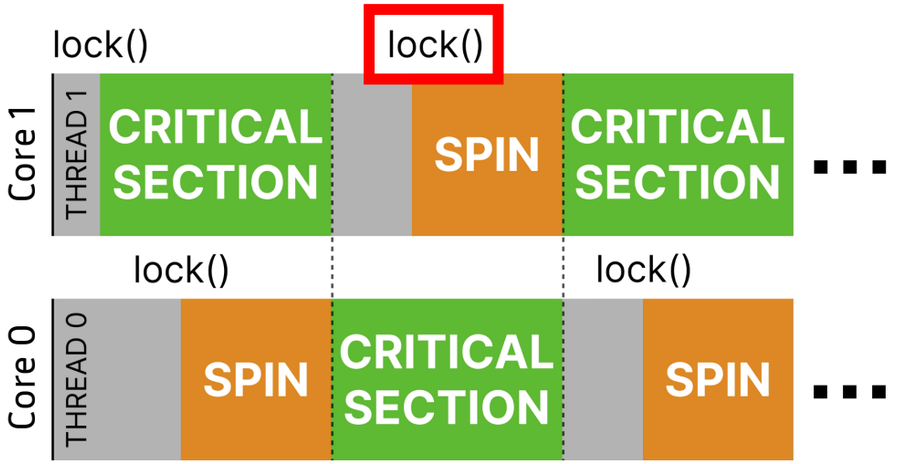




Long critical path!

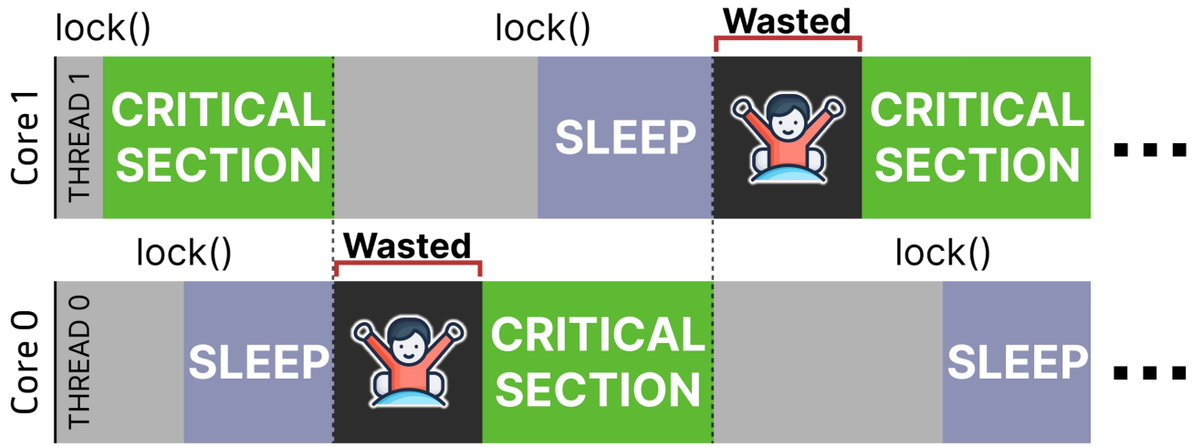
VS.

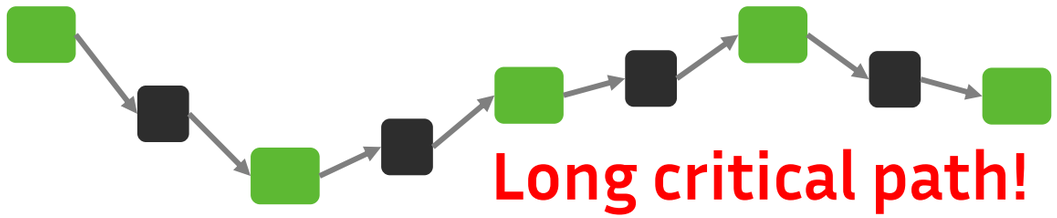
- **Spinlocks:** busy-wait instead
  - Many variants exist (with queues, tickets…)
  - **Fast** lock handovers: one cache miss!



2. Task Synchronization: FlexGuard [SOSP '25] *(Victor Laforet)*

- **Blocking locks:** block if lock already acquired
  - E.g., POSIX (`pthread_mutex_lock()`)
  - **Problem:** costly wakeups ⇒ **slow**!

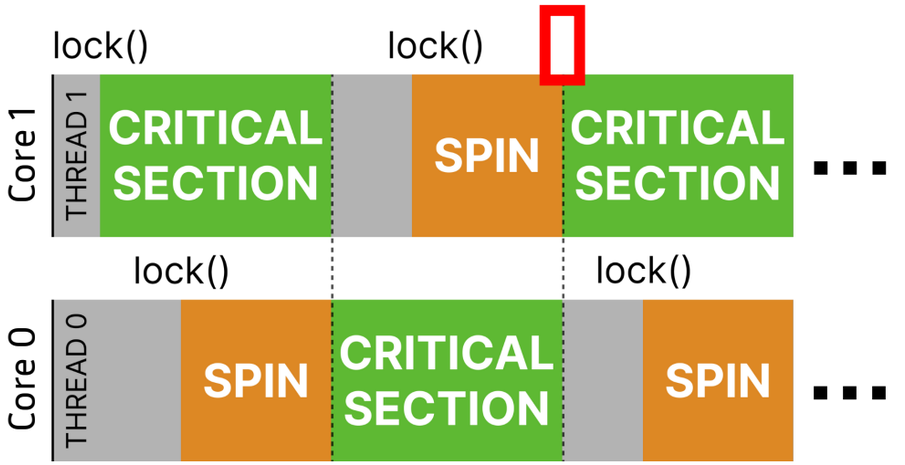




Long critical path!

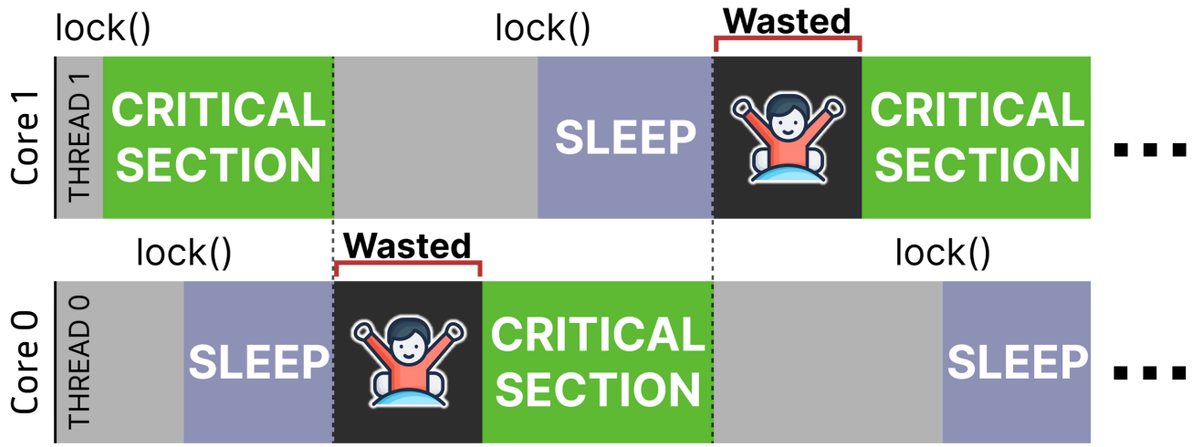
VS.

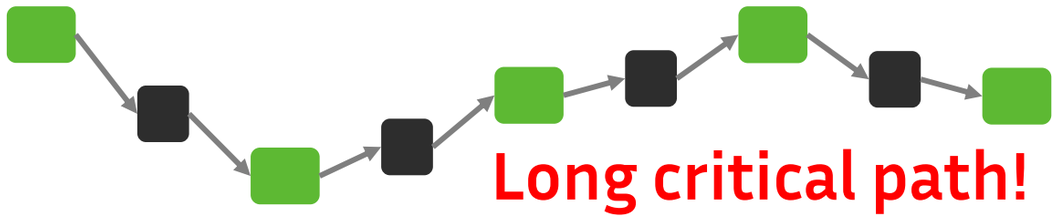
- **Spinlocks:** busy-wait instead
  - Many variants exist (with queues, tickets…)
  - **Fast** lock handovers: one cache miss!



2. Task Synchronization: FlexGuard [SOSP '25] *(Victor Laforet)*

- **Blocking locks:** block if lock already acquired
  - E.g., POSIX (`pthread_mutex_lock()`)
  - **Problem:** costly wakeups ⇒ **slow**!

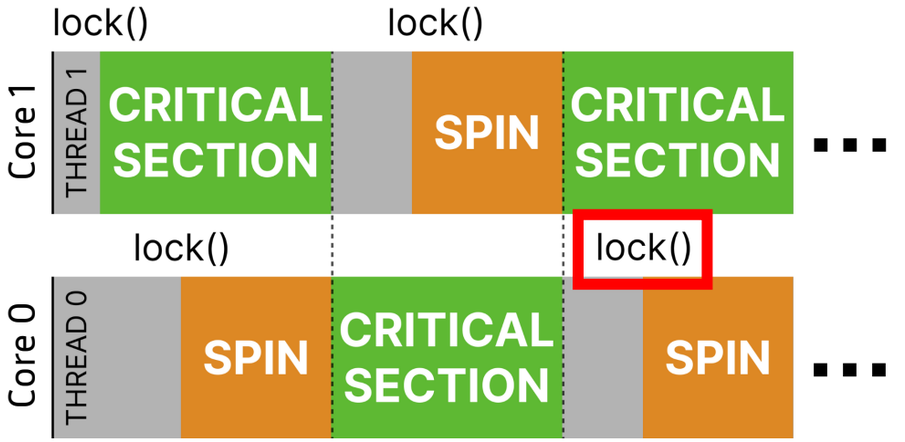




Long critical path!

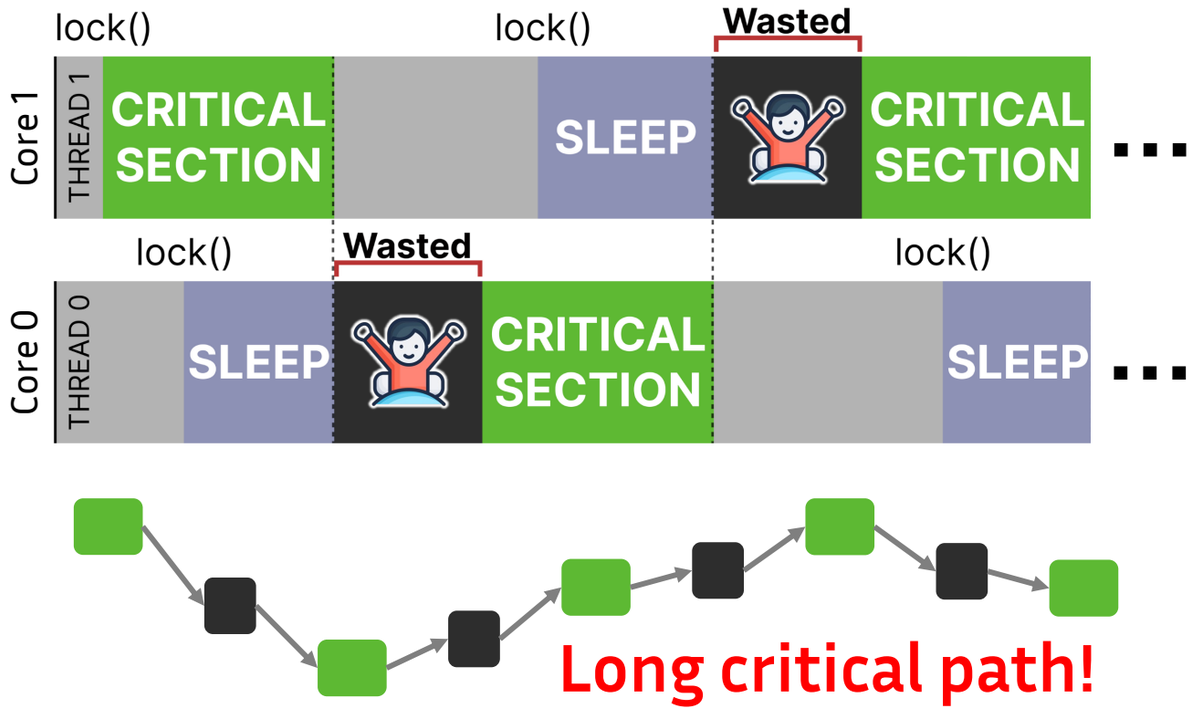
VS.

- **Spinlocks:** busy-wait instead
  - Many variants exist (with queues, tickets…)
  - **Fast** lock handovers: one cache miss!



2. Task Synchronization: FlexGuard [SOSP '25] *(Victor Laforet)*

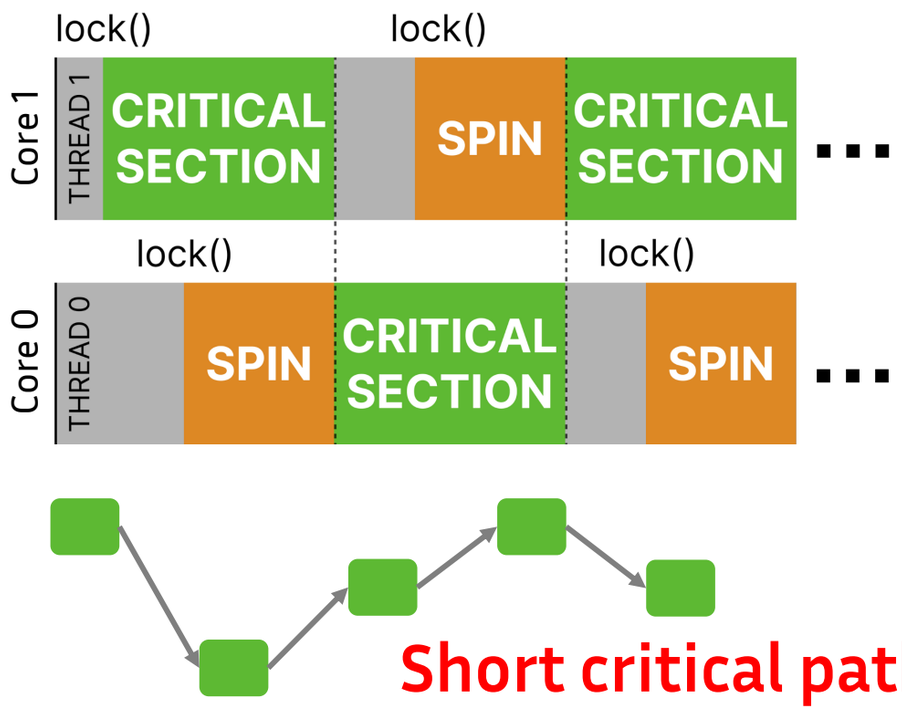
- **Blocking locks:** block if lock already acquired
  - E.g., POSIX (`pthread_mutex_lock()`)
  - **Problem:** costly wakeups ⇒ **slow**!



Long critical path!

VS.

- **Spinlocks:** busy-wait instead
  - Many variants exist (with queues, tickets…)
  - **Fast** lock handovers: one cache miss!



Short critical path!

2. Task Synchronization: FlexGuard [SOSP '25] (*Victor Laforet*)

- If spinlocks are so fast, why do standard libraries (e.g., POSIX) use blocking locks?

2. Task Synchronization: FlexGuard [SOSP '25] (*Victor Laforet*)

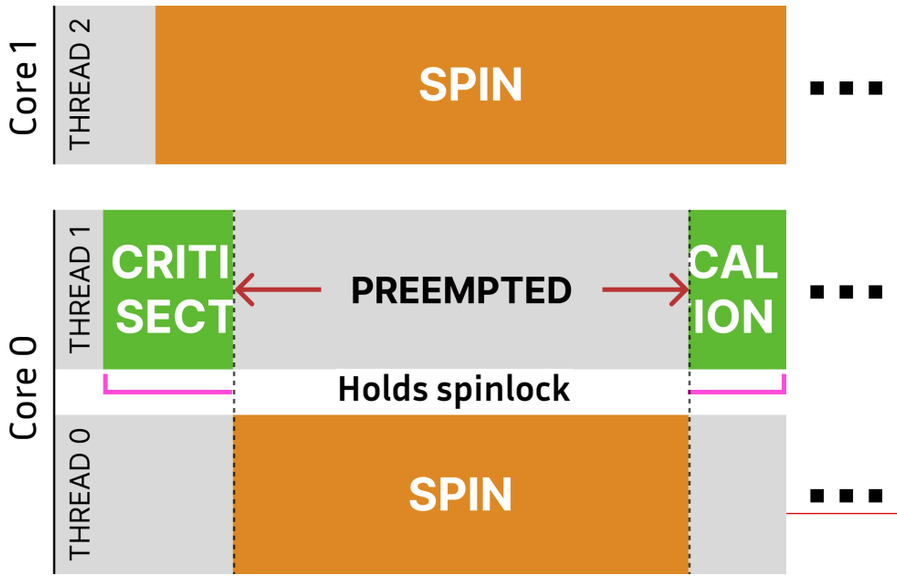
- **If spinlocks are so fast, why do standard libraries (e.g., POSIX) use blocking locks?**
  - **Answer:** stability!
    - Spinlocks perform great in non-oversubscription (# threads ≤ # available cores)
    - But their performance collapses in oversubscription (# threads > # available cores)

2. Task Synchronization: FlexGuard [SOSP '25] (*Victor Laforet*)

- **If spinlocks are so fast, why do standard libraries (e.g., POSIX) use blocking locks?**
  - **Answer:** stability!
    - Spinlocks perform great in non-oversubscription (# threads ≤ # available cores)
    - But their performance collapses in oversubscription (# threads > # available cores)
  - **Reason:** spinners preempt the critical sections, stopping progress on the critical path!

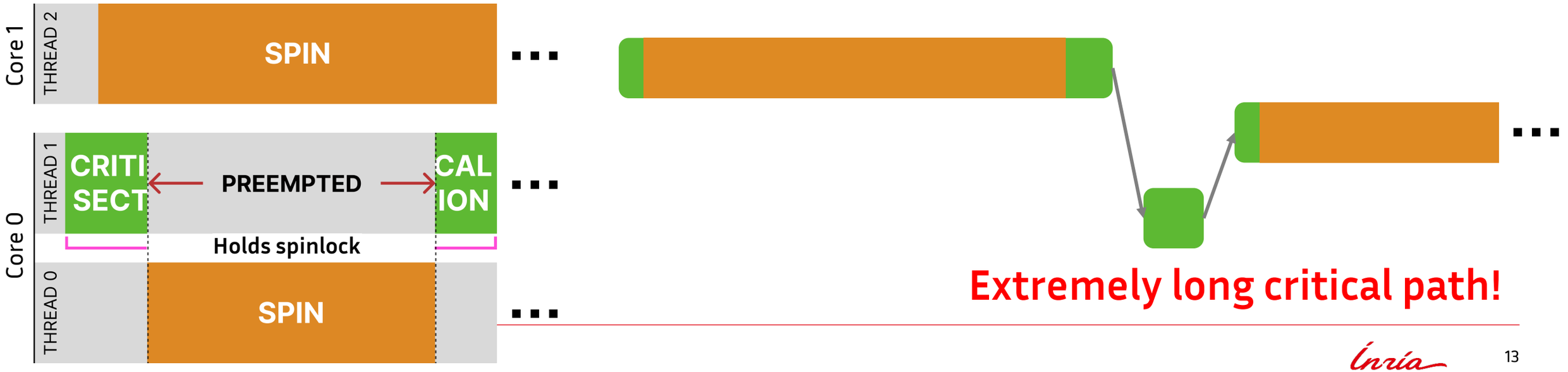
2. Task Synchronization: FlexGuard [SOSP '25] (*Victor Laforet*)

- **If spinlocks are so fast, why do standard libraries (e.g., POSIX) use blocking locks?**
  - **Answer:** stability!
    - Spinlocks perform great in non-oversubscription (# threads ≤ # available cores)
    - But their performance collapses in oversubscription (# threads > # available cores)
  - **Reason:** spinners preempt the critical sections, stopping progress on the critical path!



2. Task Synchronization: FlexGuard [SOSP '25] (*Victor Laforet*)

- **If spinlocks are so fast, why do standard libraries (e.g., POSIX) use blocking locks?**
  - **Answer:** stability!
    - Spinlocks perform great in non-oversubscription (# threads ≤ # available cores)
    - But their performance collapses in oversubscription (# threads > # available cores)
  - **Reason:** spinners preempt the critical sections, stopping progress on the critical path!

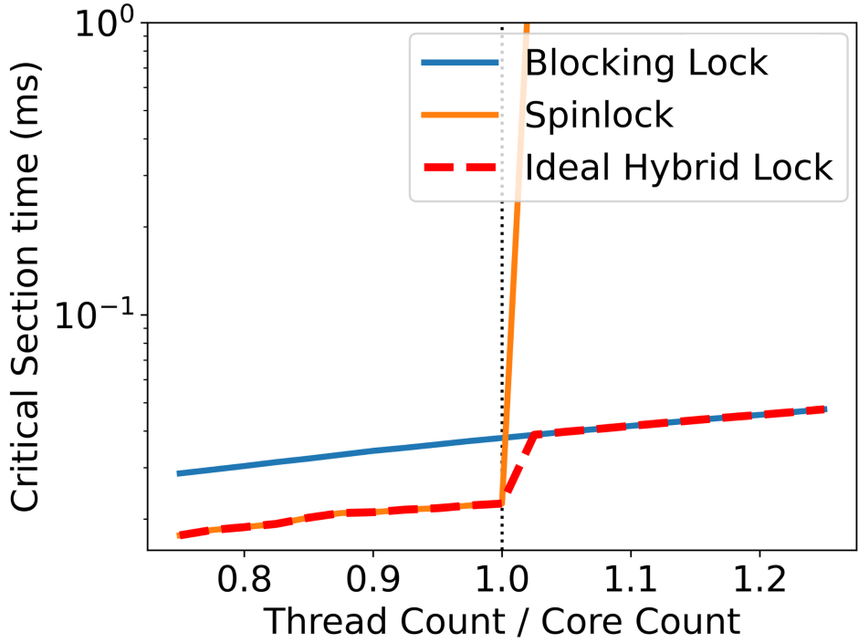


2. Task Synchronization: FlexGuard [SOSP '25] (*Victor Laforet*)

- **Goal:** get the best of both worlds!
  - In **non-oversubscription**, **spinlock performance**
  - In **oversubscription**, **blocking lock performance**

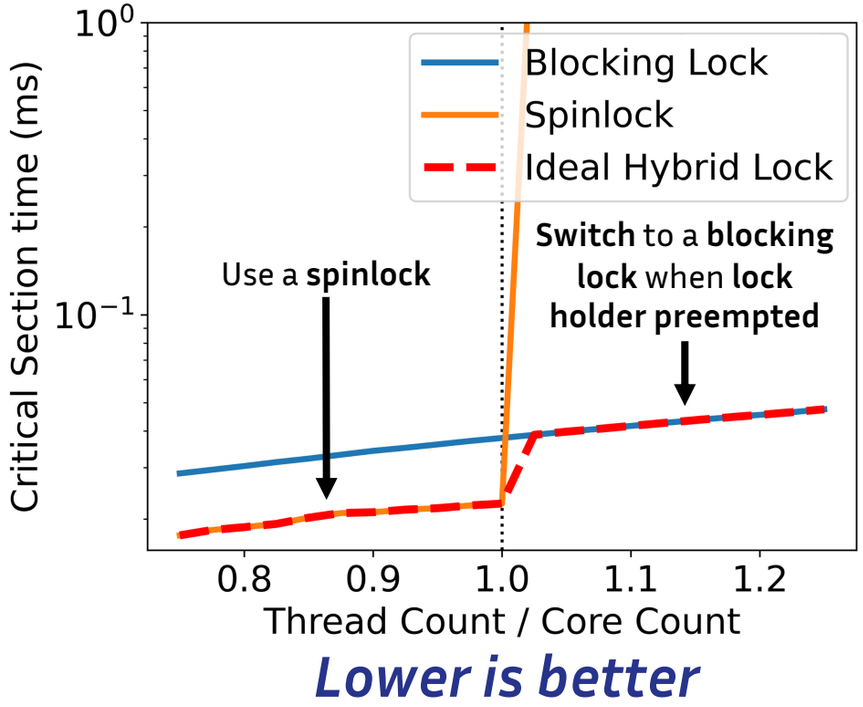
2. Task Synchronization: FlexGuard [SOSP '25]

- **Goal:** get the best of both worlds!
  - In **non-oversubscription**, **spinlock performance**
  - In **oversubscription**, **blocking lock performance**



Lower is better

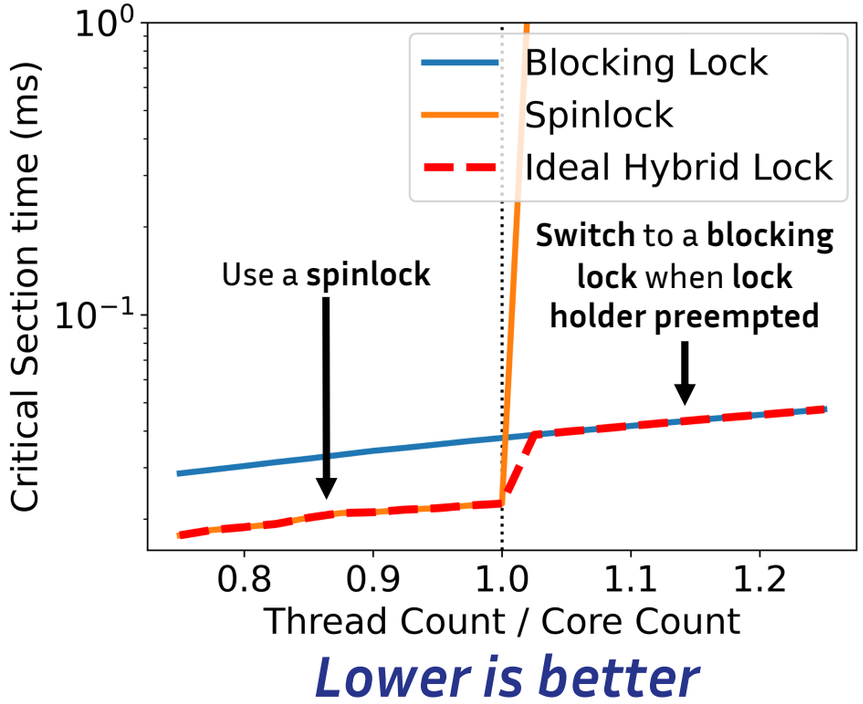
2. Task Synchronization: FlexGuard [SOSP '25]



Lower is better

- **Goal:** get the best of both worlds!
  - In **non-oversubscription**, **spinlock performance**
  - In **oversubscription**, **blocking lock performance**
- **Idea:** use a spinlock, when critical section preempted, block the waiters!

2. Task Synchronization: FlexGuard [SOSP '25]

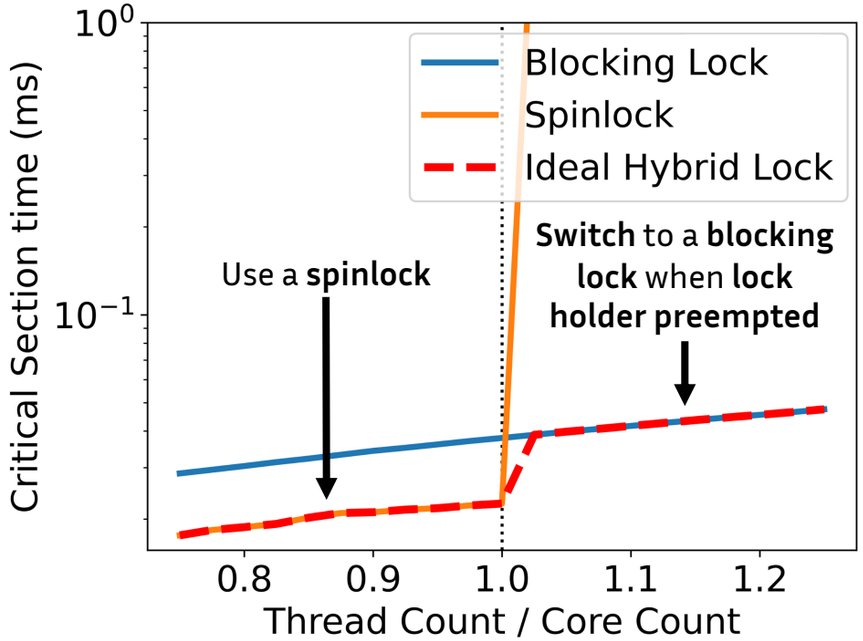


Lower is better

- **Goal:** get the best of both worlds!
  - In **non-oversubscription**, **spinlock performance**
  - In **oversubscription**, **blocking lock performance**

- **Idea:** use a spinlock, when critical section preempted, block the waiters!
  - *Can we do this?* 🤔

2. Task Synchronization: FlexGuard [SOSP '25]



Lower is better

- **Goal:** get the best of both worlds!
  - In **non-oversubscription**, **spinlock performance**
  - In **oversubscription**, **blocking lock performance**

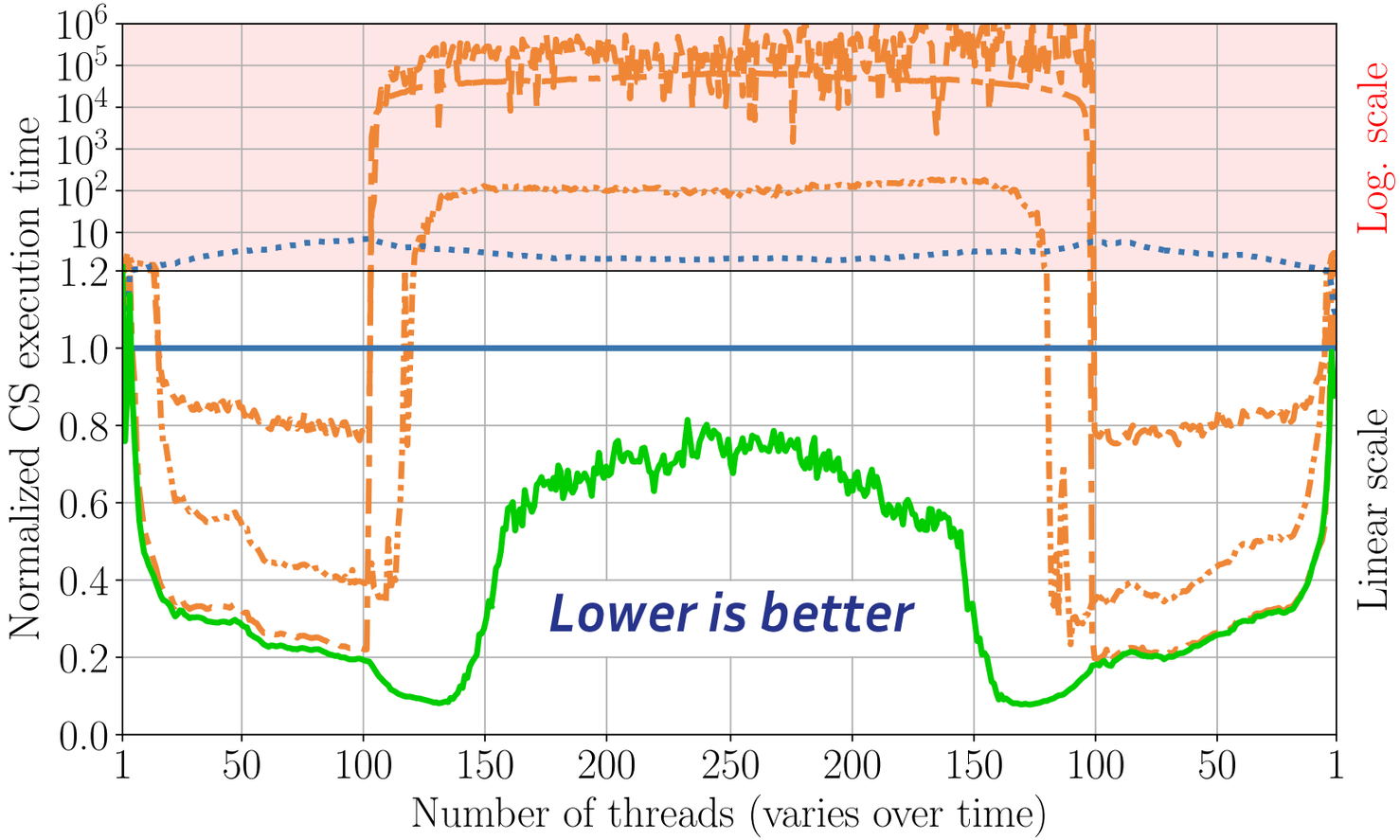
- **Idea:** use a spinlock, when critical section preempted, block the waiters!
  - *Can we do this?* 🤔

- **Insight:** nowadays, with eBPF we can!
  - We can instrument context switches to see all preemptions
  - We can view the full state of the thread: preemption address + register contents

⇒ We can 100% tell whether we are in a critical section!

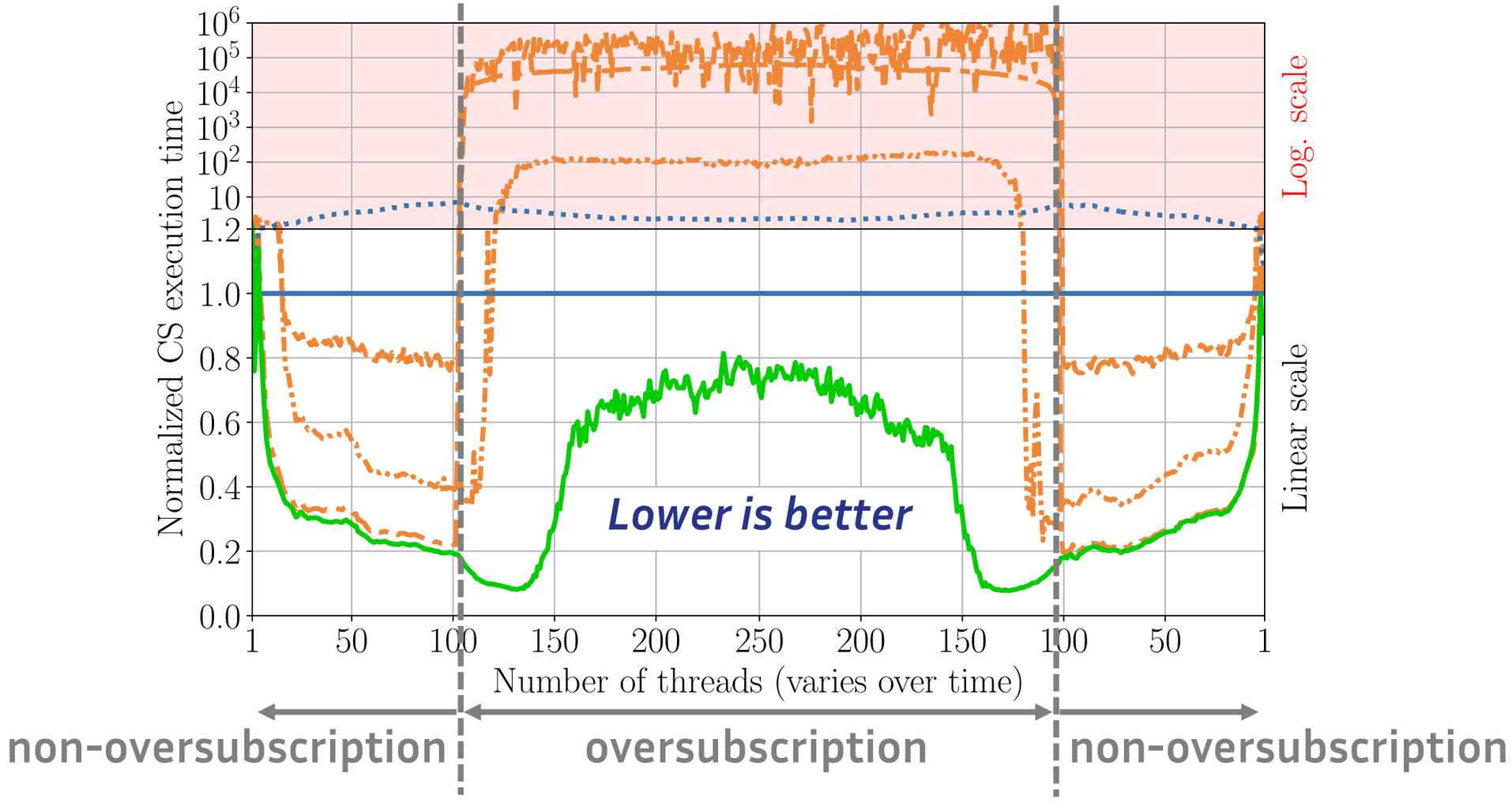
2. Task Synchronization: FlexGuard [SOSP '25] (*Victor Laforet*)

Orange: spinlocks **Blue:** blocking locks **Green:** our solution, **FlexGuard**



2. Task Synchronization: FlexGuard [SOSP '25] (*Victor Laforet*)

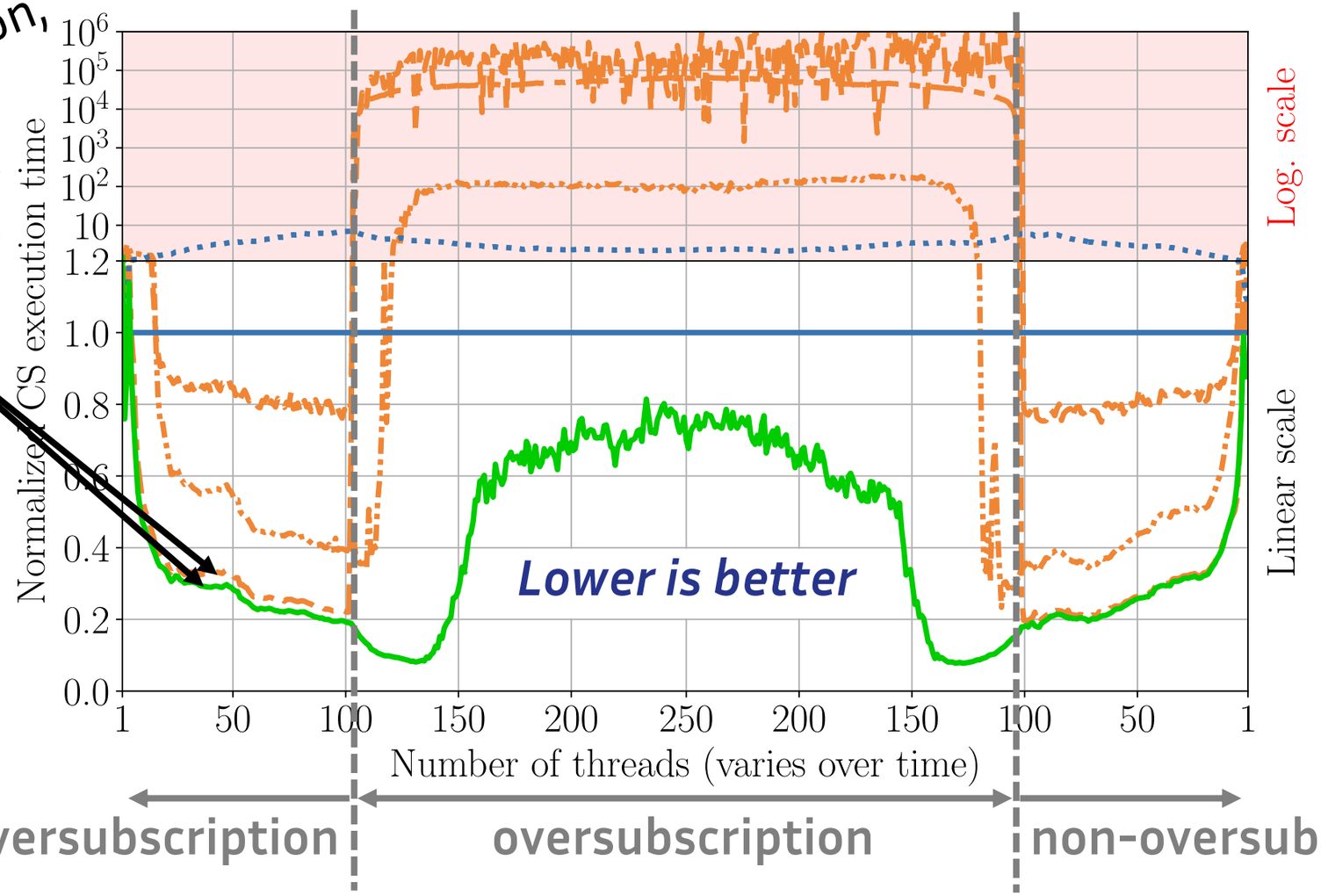
Orange: spinlocks **Blue:** blocking locks **Green:** our solution, **FlexGuard**



2. Task Synchronization: FlexGuard [SOSP '25] (*Victor Laforet*)

Orange: spinlocks **Blue:** blocking locks **Green:** our solution, **FlexGuard**

In non-oversubcription, FlexGuard matches the performance of the best spinlocks

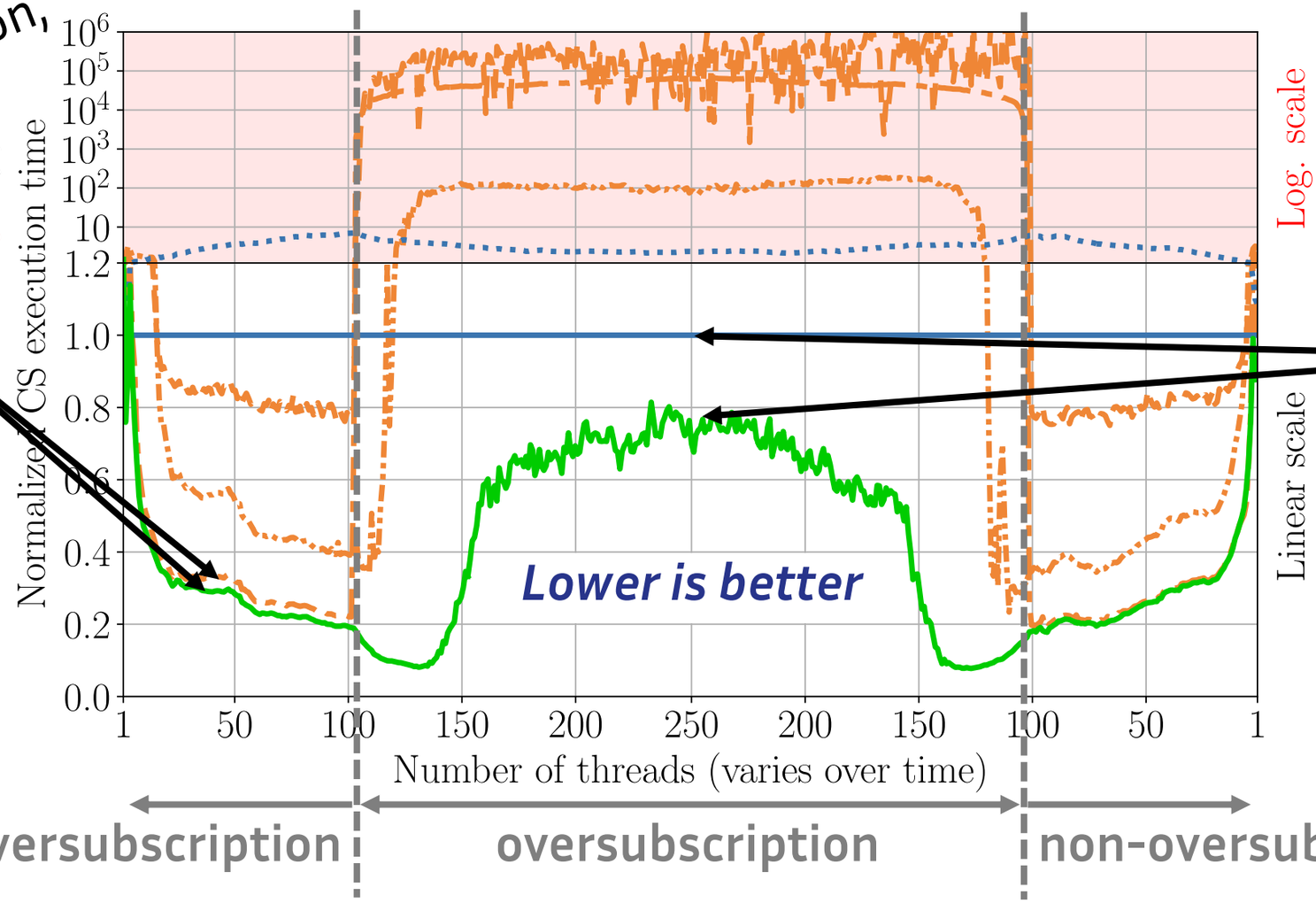


2. Task Synchronization: FlexGuard [SOSP '25] (*Victor Laforet*)

Orange: spinlocks **Blue:** blocking locks **Green:** our solution, **FlexGuard**

In non-oversubcription, **FlexGuard** matches the performance of the best spinlocks

In oversubscription **FlexGuard** not only matches, but outperforms the best blocking locks

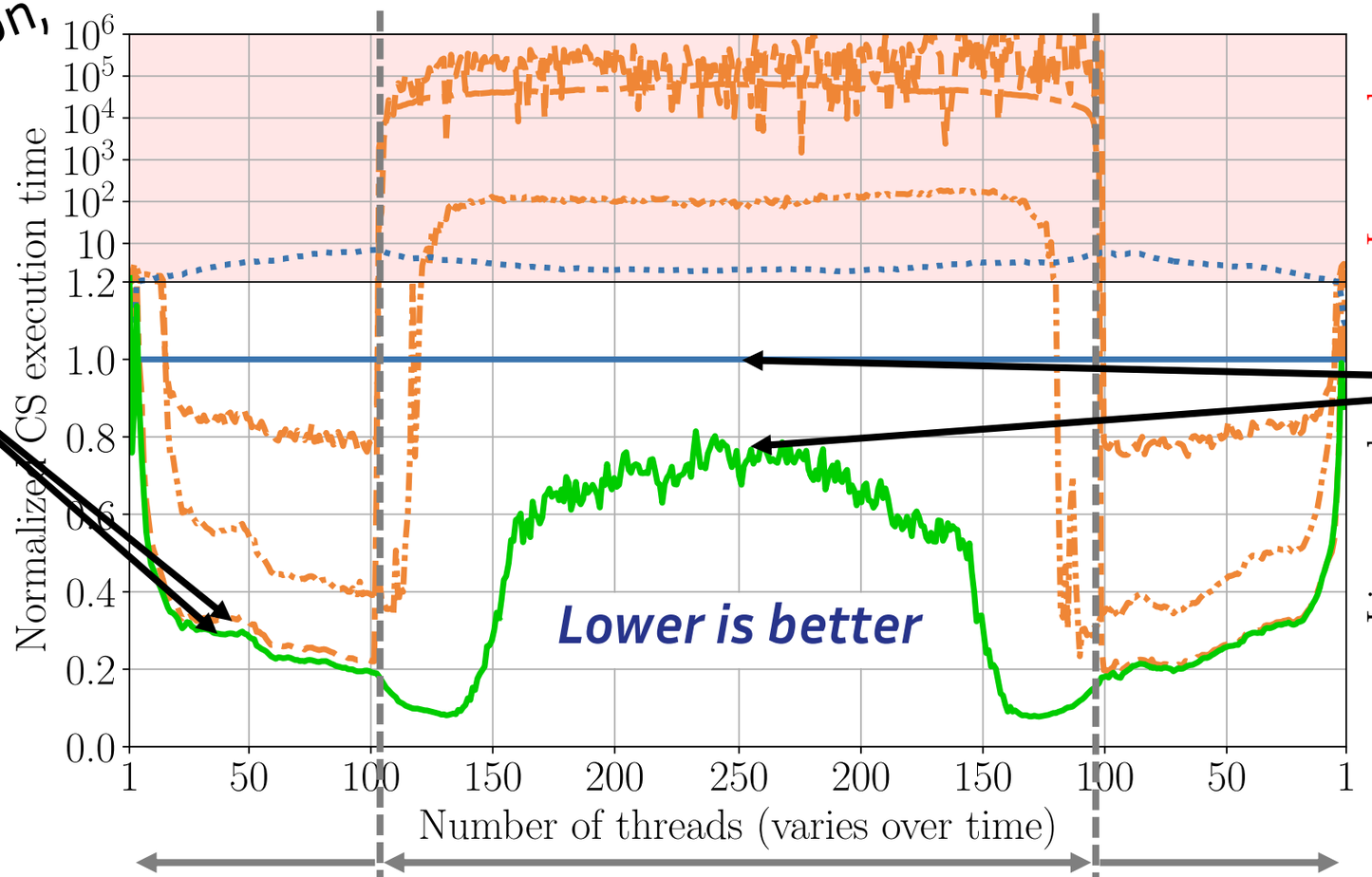


2. Task Synchronization: FlexGuard [SOSP '25] (*Victor Laforet*)

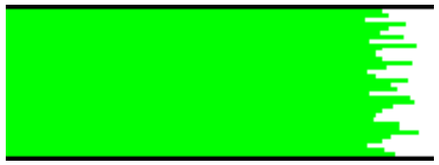
Orange: spinlocks **Blue:** blocking locks **Green:** our solution, **FlexGuard**

In non-oversubcription, **FlexGuard** matches the performance of the best spinlocks

In oversubscription **FlexGuard** not only matches, but outperforms the best blocking locks

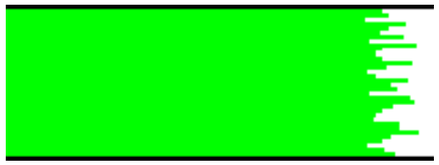


2. Task Synchronization: Tapestry [PLOS '25] (*Tomáš Faltín*)



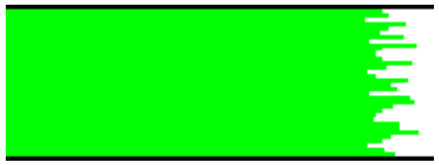
- **Profiling issue:** inter-thread dependencies invisible, especially with spin synchronization

2. Task Synchronization: Tapestry [PLOS '25] (*Tomáš Faltín*)



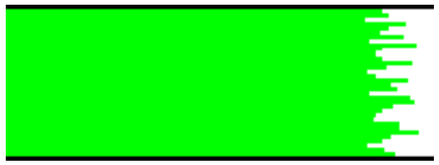
- **Profiling issue:** inter-thread dependencies invisible, especially with spin synchronization
- **Insight:** both block- and spin-dependencies usually depend on a `*addr ⋈ val` condition

2. Task Synchronization: Tapestry [PLOS '25] *(Tomáš Faltín)*



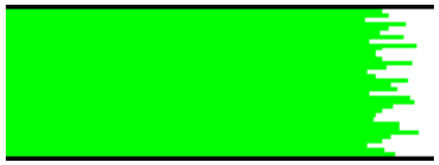
- **Profiling issue:** inter-thread dependencies invisible, especially with spin synchronization
- **Insight:** both block- and spin-dependencies usually depend on a **`*addr ⋈ val`** condition
  - E.g., a T_A only progresses when `*addr == 0`; T_B does `*addr = 0`

2. Task Synchronization: Tapestry [PLOS '25] *(Tomáš Faltín)*



- **Profiling issue:** inter-thread dependencies invisible, especially with spin synchronization
- **Insight:** both block- and spin-dependencies usually depend on a **`*addr ⋈ val`** condition
  - E.g., a T_A only progresses when `*addr == 0`; T_B does `*addr = 0`
  - *How to capture that?* 🤔

2. Task Synchronization: Tapestry [PLOS '25] *(Tomáš Faltín)*



- **Profiling issue:** inter-thread dependencies invisible, especially with spin synchronization
- **Insight:** both block- and spin-dependencies usually depend on a **`*addr ⋈ val`** condition
  - E.g., a T_A only progresses when `*addr == 0`; T_B does `*addr = 0`
  - *How to capture that?* 🤔

T_A busy-waits

```
while (compare_and_swap(&lock, 0, 1))
    ;
```

reads

Shared memory

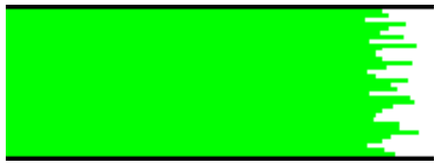
0x…a0	0x00000001
data address	**lock** value

T_B resolves the dependency

0x…38	cmp	%eax, %edx
0x…3c	mov	$0, (**lock**)
0x…40	ret	

code address

2. Task Synchronization: Tapestry [PLOS '25] *(Tomáš Faltín)*



- **Profiling issue:** inter-thread dependencies invisible, especially with spin synchronization
- **Insight:** both block- and spin-dependencies usually depend on a **`*addr ⋈ val`** condition
  - E.g., a T_A only progresses when `*addr == 0`; T_B does `*addr = 0`
  - *How to capture that?* 🤔

T_A busy-waits

```
while (compare_and_swap(&lock, 0, 1))
    ;
```

reads

Shared memory

0x…a0	0x00000001
data address	**lock** value

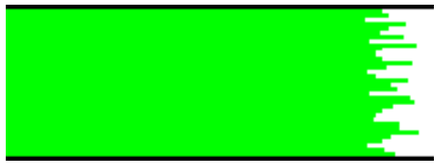
Hardware watchpoint `@0x…a0`

T_B resolves the dependency

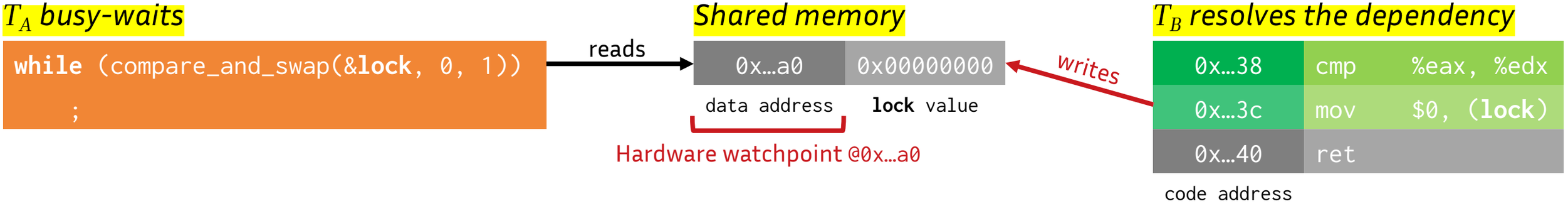
code address		
0x…38	cmp	%eax, %edx
0x…3c	mov	$0, (**lock**)
0x…40	ret	

1. Hardware watchpoint on the data address, **only 4 on x86**

2. Task Synchronization: Tapestry [PLOS '25] *(Tomáš Faltín)*

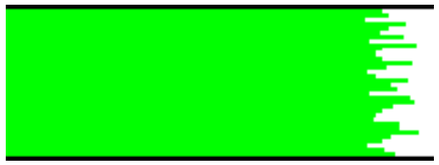


- **Profiling issue:** inter-thread dependencies invisible, especially with spin synchronization
- **Insight:** both block- and spin-dependencies usually depend on a **`*addr ⋈ val`** condition
  - E.g., a T_A only progresses when `*addr == 0`; T_B does `*addr = 0`
  - *How to capture that?* 🤔

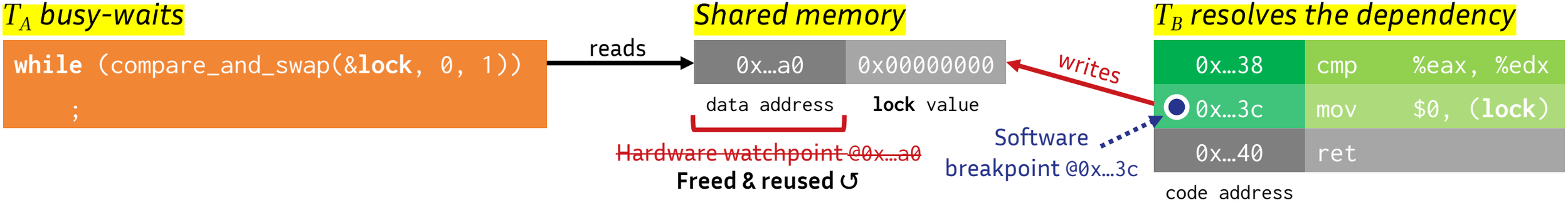


1. Hardware watchpoint on the data address, **only 4 on x86**
2. The store fires it, revealing the code address

2. Task Synchronization: Tapestry [PLOS '25] *(Tomáš Faltín)*

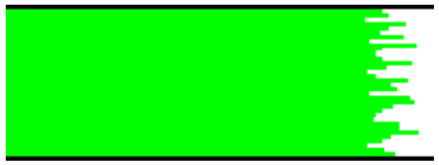


- **Profiling issue:** inter-thread dependencies invisible, especially with spin synchronization
- **Insight:** both block- and spin-dependencies usually depend on a **`*addr ⋈ val`** condition
  - E.g., a T_A only progresses when `*addr == 0`; T_B does `*addr = 0`
  - *How to capture that?* 🤔

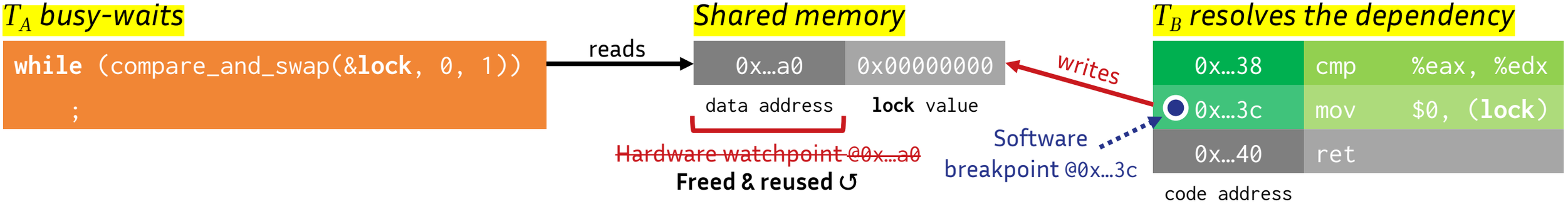


1. Hardware watchpoint on the data address, **only 4 on x86**
2. The store fires it, revealing the code address
3. Add a software breakpoint there; free the watchpoint

2. Task Synchronization: Tapestry [PLOS '25] (*Tomáš Faltín*)



- **Profiling issue:** inter-thread dependencies invisible, especially with spin synchronization
- **Insight:** both block- and spin-dependencies usually depend on a **`*addr ⋈ val`** condition
  - E.g., a T_A only progresses when `*addr == 0`; T_B does `*addr = 0`
  - *How to capture that?* 🤔



1. Hardware watchpoint on the data address, **only 4 on x86**
2. The store fires it, revealing the code address
3. Add a software breakpoint there; free the watchpoint

With a dependency graph, we can run a longest path algorithm to highlight the **critical path!**

2. Task Synchronization: Other and Future Work

- **Other work:** Remote Core Locking [USENIX ATC '12, TOCS '15]
  - **Idea:** dedicate cores to the execution of critical sections

2. Task Synchronization: Other and Future Work

- **Other work:** Remote Core Locking [USENIX ATC '12, TOCS '15]
  - **Idea:** dedicate cores to the execution of critical sections
  - No synchronization = faster lock handovers; higher locality = faster critical sections

2. Task Synchronization: Other and Future Work

- **Other work:** Remote Core Locking [USENIX ATC '12, TOCS '15]
  - **Idea:** dedicate cores to the execution of critical sections
  - No synchronization = faster lock handovers; higher locality = faster critical sections
  - Delegation = fastest lock design (confirmed by ffwd [SOSP '17], TCLocks [OSDI '23])

2. Task Synchronization: Other and Future Work

- **Other work:** Remote Core Locking [USENIX ATC '12, TOCS '15]
  - **Idea:** dedicate cores to the execution of critical sections
  - No synchronization = faster lock handovers; higher locality = faster critical sections
  - Delegation = fastest lock design (confirmed by ffwd [SOSP '17], TCLocks [OSDI '23])
- **Future work:**

FlexGuard × delegation

Gracefully handle oversubscription + contention

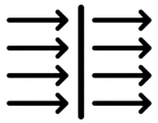
2. Task Synchronization: Other and Future Work

- **Other work:** Remote Core Locking [USENIX ATC '12, TOCS '15]
  - **Idea:** dedicate cores to the execution of critical sections
  - No synchronization = faster lock handovers; higher locality = faster critical sections
  - Delegation = fastest lock design (confirmed by ffwd [SOSP '17], TCLocks [OSDI '23])
- **Future work:**

FlexGuard × delegation

Gracefully handle oversubscription + contention

FlexGuard × other synch.



Abortable locks, barriers (w/ Himadri)

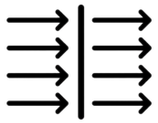
2. Task Synchronization: Other and Future Work

- **Other work:** Remote Core Locking [USENIX ATC '12, TOCS '15]
  - **Idea:** dedicate cores to the execution of critical sections
  - No synchronization = faster lock handovers; higher locality = faster critical sections
  - Delegation = fastest lock design (confirmed by ffwd [SOSP '17], TCLocks [OSDI '23])
- **Future work:**

FlexGuard × delegation

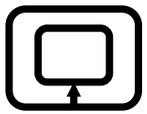
Gracefully handle oversubscription + contention

FlexGuard × other synch.



Abortable locks, barriers (w/ Himadri)

CS preemption detection in VMs



For guest kernel & user locks (w/ Maxime)

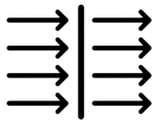
2. Task Synchronization: Other and Future Work

- **Other work:** Remote Core Locking [USENIX ATC '12, TOCS '15]
  - **Idea:** dedicate cores to the execution of critical sections
  - No synchronization = faster lock handovers; higher locality = faster critical sections
  - Delegation = fastest lock design (confirmed by ffwd [SOSP '17], TCLocks [OSDI '23])
- **Future work:**

FlexGuard × delegation

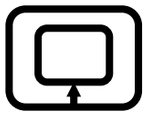
Gracefully handle oversubscription + contention

FlexGuard × other synch.



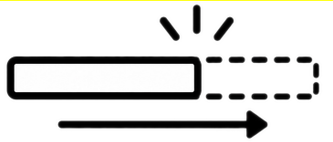
Abortable locks, barriers (w/ Himadri)

CS preemption detection in VMs



For guest kernel & user locks (w/ Maxime)

Algorithms for timeslice ext.



How to best use it?

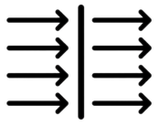
2. Task Synchronization: Other and Future Work

- **Other work:** Remote Core Locking [USENIX ATC '12, TOCS '15]
  - **Idea:** dedicate cores to the execution of critical sections
  - No synchronization = faster lock handovers; higher locality = faster critical sections
  - Delegation = fastest lock design (confirmed by ffwd [SOSP '17], TCLocks [OSDI '23])
- **Future work:**

FlexGuard × delegation

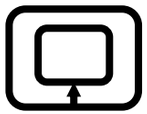
Gracefully handle oversubscription + contention

FlexGuard × other synch.



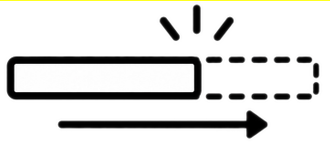
Abortable locks, barriers (w/ Himadri)

CS preemption detection in VMs



For guest kernel & user locks (w/ Maxime)

Algorithms for timeslice ext.



How to best use it?

Towards a new pthread lib?

Fast but also practical, stable algorithms

3. Graph Analytics: Smart Arrays [EuroSys '18]

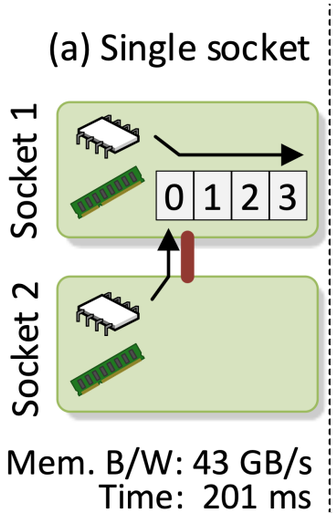
- Graph analytics stream over huge arrays (CSR, properties): **memory-bandwidth-bound**

3. Graph Analytics: Smart Arrays [EuroSys '18]

- Graph analytics stream over huge arrays (CSR, properties): **memory-bandwidth-bound**
- **What matters:** where pages sit (local vs. remote) and bytes crossing the interconnect

3. Graph Analytics: Smart Arrays [EuroSys '18]

- Graph analytics stream over huge arrays (CSR, properties): **memory-bandwidth-bound**
- **What matters:** where pages sit (local vs. remote) and bytes crossing the interconnect



Single-socket

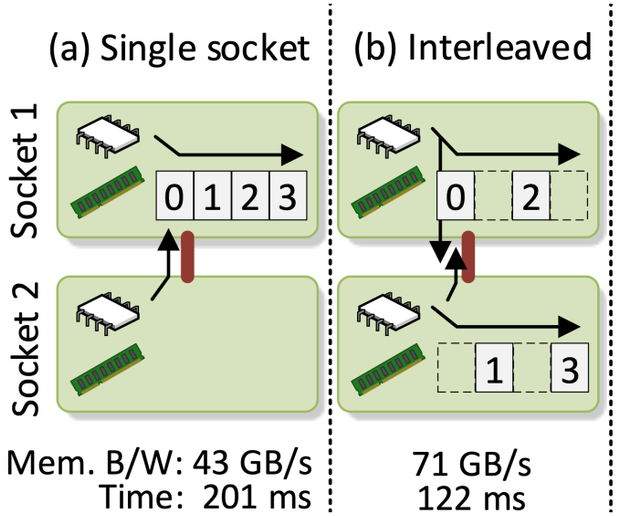
→ **Bottleneck:** socket's memory bandwidth

→ Fast for half the threads

→ *Can perform fine with e.g., good load balancing*

3. Graph Analytics: Smart Arrays [EuroSys '18]

- Graph analytics stream over huge arrays (CSR, properties): **memory-bandwidth-bound**
- **What matters:** where pages sit (local vs. remote) and bytes crossing the interconnect



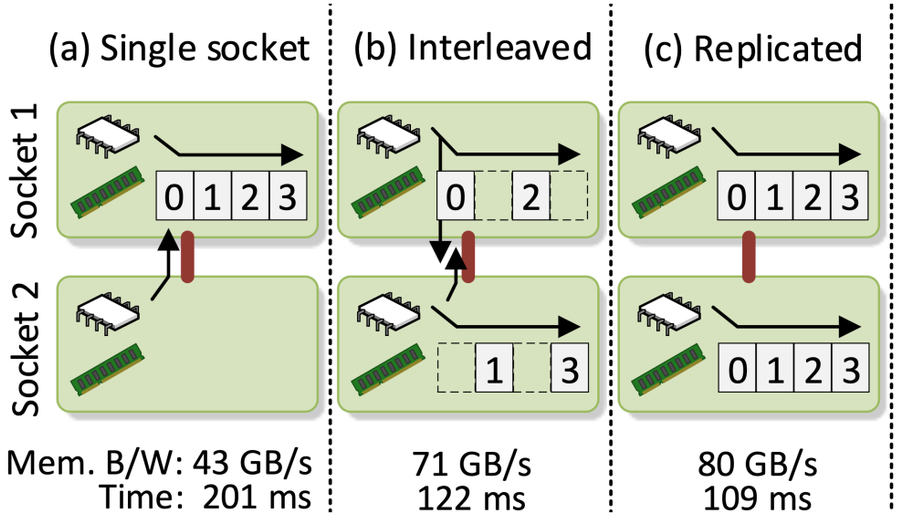
Interleaved

→ More bandwidth (2 sockets)

→ **Bottleneck:** interconnect

3. Graph Analytics: Smart Arrays [EuroSys '18]

- Graph analytics stream over huge arrays (CSR, properties): **memory-bandwidth-bound**
- **What matters:** where pages sit (local vs. remote) and bytes crossing the interconnect



Replicated

→ Only works for read-mostly workloads

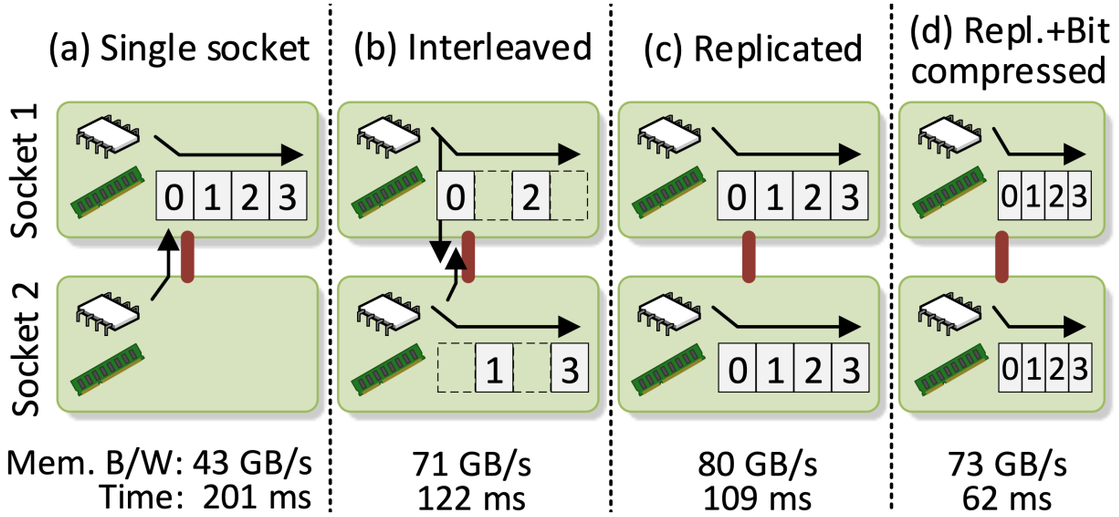
→ Removes the interconnect bottleneck

→ Needs enough memory

→ *Replica-building worth it?*
(Multiple/random accesses?)

3. Graph Analytics: Smart Arrays [EuroSys '18]

- Graph analytics stream over huge arrays (CSR, properties): **memory-bandwidth-bound**
- **What matters:** where pages sit (local vs. remote) and bytes crossing the interconnect



Replicated + Bit compressed

→ Use memory bandwidth more productively

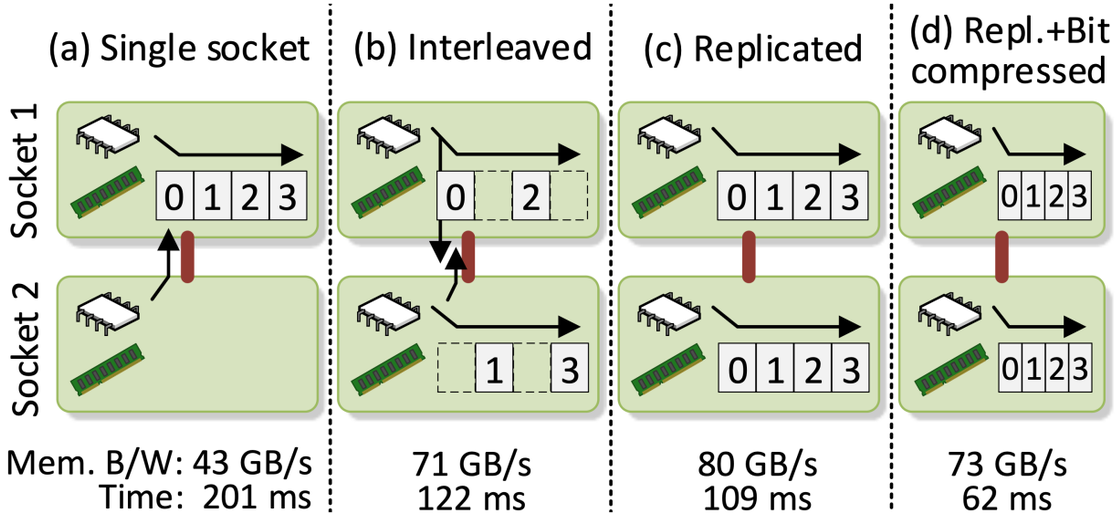
→ Shrinks memory footprint

→ CPU can be a bottleneck

→ *Would be costly with random accesses*

3. Graph Analytics: Smart Arrays [EuroSys '18]

- Graph analytics stream over huge arrays (CSR, properties): **memory-bandwidth-bound**
- **What matters:** where pages sit (local vs. remote) and bytes crossing the interconnect



Replicated + Bit compressed

→ Use memory bandwidth more productively

→ Shrinks memory footprint

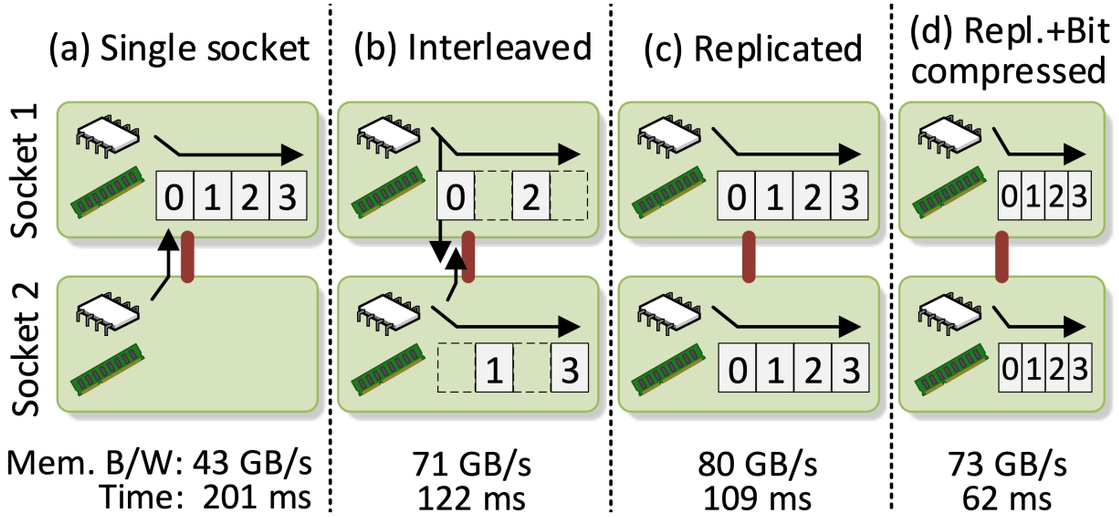
→ CPU can be a bottleneck

→ *Would be costly with random accesses*

- **Problem:** managed languages (e.g., Java) hide the layout

3. Graph Analytics: Smart Arrays [EuroSys '18]

- Graph analytics stream over huge arrays (CSR, properties): **memory-bandwidth-bound**
- **What matters:** where pages sit (local vs. remote) and bytes crossing the interconnect



Replicated + Bit compressed

→ Use memory bandwidth more productively

→ Shrinks memory footprint

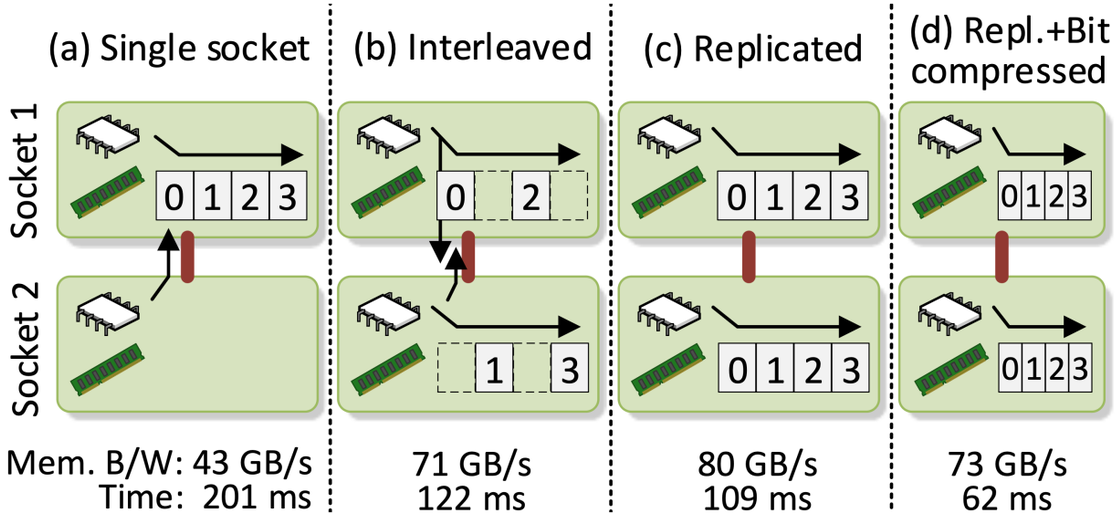
→ CPU can be a bottleneck

→ *Would be costly with random accesses*

- **Problem:** managed languages (e.g., Java) hide the layout
- **Smart arrays:** smart memory features (placement+compression), language-independent
  - Compiled once in C++, usable from e.g., C++ and Java thanks to **GraalVM**

3. Graph Analytics: Smart Arrays [EuroSys '18]

- Graph analytics stream over huge arrays (CSR, properties): **memory-bandwidth-bound**
- **What matters:** where pages sit (local vs. remote) and bytes crossing the interconnect



Replicated + Bit compressed

→ Use memory bandwidth more productively

→ Shrinks memory footprint

→ CPU can be a bottleneck

→ *Would be costly with random accesses*

- **Problem:** managed languages (e.g., Java) hide the layout
- **Smart arrays:** smart memory features (placement+compression), language-independent
  - Compiled once in C++, usable from e.g., C++ and Java thanks to **GraalVM**

+Adaptivity!

3. Graph Analytics: aDFS [USENIX ATC '21]

- **Goal:** process PGQL queries like,

```
SELECT a, c WHERE (a:author) -[:wrote]-> (b:paper) -[:cite]->
                  (c:paper) <-[:wrote]- (d:author)
```

"Pairs of authors who wrote papers that cite each other"

3. Graph Analytics: aDFS [USENIX ATC '21]

- **Goal:** process PGQL queries like,

```
SELECT a, c WHERE (a:author) -[:wrote]-> (b:paper) -[:cite]->
                  (c:paper) <-[:wrote]- (d:author)
```

  "Pairs of authors who wrote papers that cite each other"

- **BFS vs. DFS tradeoff:** `(a) -> (b) -> (c) <- (d)`

3. Graph Analytics: aDFS [USENIX ATC '21]

- **Goal:** process PGQL queries like,

```
SELECT a, c WHERE (a:author) -[:wrote]-> (b:paper) -[:cite]->
                  (c:paper) <-[:wrote]- (d:author)
```

"Pairs of authors who wrote papers that cite each other"

- **BFS vs. DFS tradeoff:** `(a) -> (b) -> (c) <- (d)`

1. Find all matching **a**s

BFS

3. Graph Analytics: aDFS [USENIX ATC '21]

- **Goal:** process PGQL queries like,

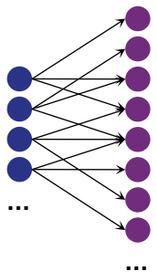
```
SELECT a, c WHERE (a:author) -[:wrote]-> (b:paper) -[:cite]->
                  (c:paper) <-[:wrote]- (d:author)
```

  "Pairs of authors who wrote papers that cite each other"

- **BFS vs. DFS tradeoff:** `(a) -> (b) -> (c) <- (d)`

1. Find all matching `a`s
2. Materialize all matching `a->b`s

BFS



3. Graph Analytics: aDFS [USENIX ATC '21]

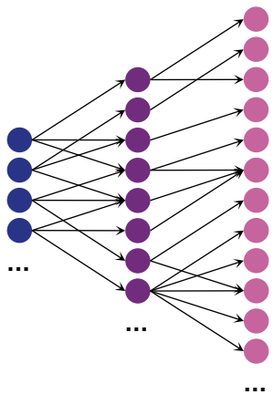
- **Goal:** process PGQL queries like,

```
SELECT a, c WHERE (a:author) -[:wrote]-> (b:paper) -[:cite]->
                  (c:paper) <-[:wrote]- (d:author)
```

  "Pairs of authors who wrote papers that cite each other"

- **BFS vs. DFS tradeoff:** `(a) -> (b) -> (c) <- (d)`

BFS



1. Find all matching **a**s
2. Materialize all matching **a**->**b**s
3. Materialize all matching **a**->**b**->**c**s

3. Graph Analytics: aDFS [USENIX ATC '21]

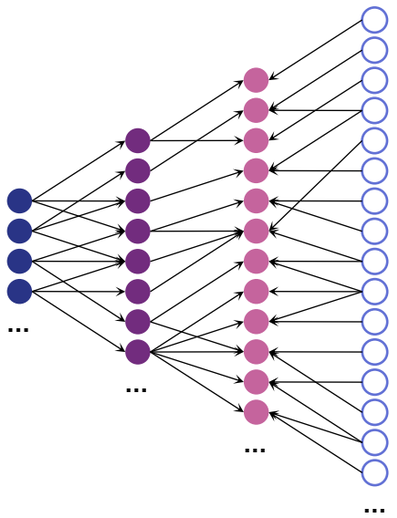
- **Goal:** process PGQL queries like,

```
SELECT a, c WHERE (a:author) -[:wrote]-> (b:paper) -[:cite]->
                  (c:paper) <-[:wrote]- (d:author)
```

"Pairs of authors who wrote papers that cite each other"

- **BFS vs. DFS tradeoff:** `(a) -> (b) -> (c) <- (d)`

BFS



1. Find all matching `a`s
2. Materialize all matching `a->b`s
3. Materialize all matching `a->b->c`s
4. Output all matching `a->b->c<-d`s

3. Graph Analytics: aDFS [USENIX ATC '21]

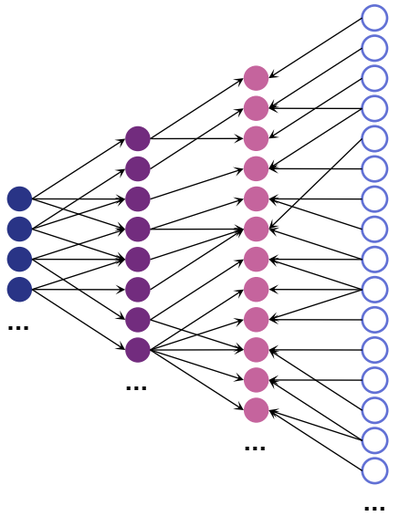
- **Goal:** process PGQL queries like,

```
SELECT a, c WHERE (a:author) -[:wrote]-> (b:paper) -[:cite]->
                  (c:paper) <-[:wrote]- (d:author)
```

"Pairs of authors who wrote papers that cite each other"

- **BFS vs. DFS tradeoff:** `(a) -> (b) -> (c) <- (d)`

BFS



1. Find all matching `a`s
2. Materialize all matching `a->b`s
3. Materialize all matching `a->b->c`s
4. Output all matching `a->b->c<-d`s

🙂 **Easy to parallelize:** split intermediate work among threads

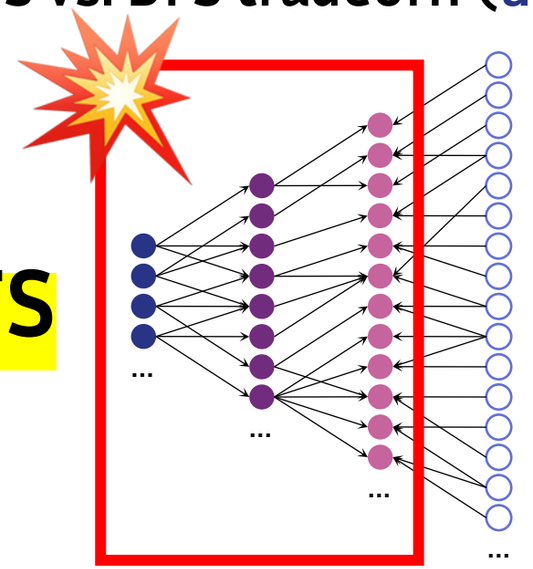
3. Graph Analytics: aDFS [USENIX ATC '21]

- **Goal:** process PGQL queries like,

```
SELECT a, c WHERE (a:author) -[:wrote]-> (b:paper) -[:cite]->
                  (c:paper) <-[:wrote]- (d:author)
```

"Pairs of authors who wrote papers that cite each other"

- **BFS vs. DFS tradeoff:** `(a) -> (b) -> (c) <- (d)`



1. Find all matching **a**s
2. Materialize all matching **a**->**b**s
3. Materialize all matching **a**->**b**->**c**s
4. Output all matching **a**->**b**->**c**<-**d**s

🙂 **Easy to parallelize:** split intermediate work among threads

☹ **Intermediate result explosion!**

3. Graph Analytics: aDFS [USENIX ATC '21]

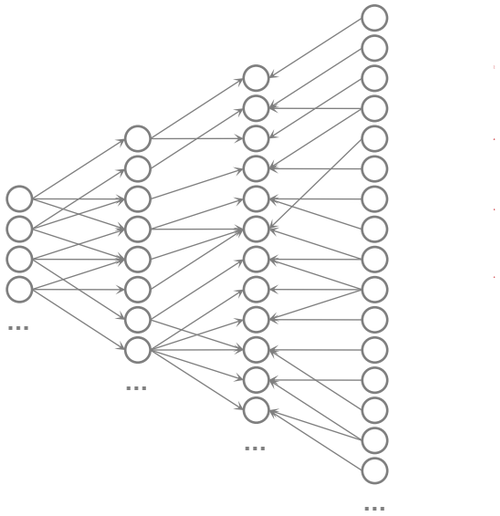
- **Goal:** process PGQL queries like,

```
SELECT a, c WHERE (a:author) -[:wrote]-> (b:paper) -[:cite]->
                  (c:paper) <-[:wrote]- (d:author)
```

  "Pairs of authors who wrote papers that cite each other"

- **BFS vs. DFS tradeoff:** `(a) -> (b) -> (c) <- (d)`

DFS



3. Graph Analytics: aDFS [USENIX ATC '21]

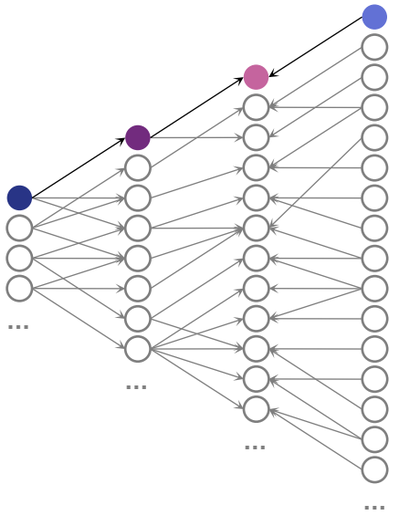
- **Goal:** process PGQL queries like,

```
SELECT a, c WHERE (a:author) -[:wrote]-> (b:paper) -[:cite]->
                  (c:paper) <-[:wrote]- (d:author)
```

"Pairs of authors who wrote papers that cite each other"

- **BFS vs. DFS tradeoff:** `(a) -> (b) -> (c) <- (d)`

DFS



1. Output the 1st matching `a->b->c<-d`

3. Graph Analytics: aDFS [USENIX ATC '21]

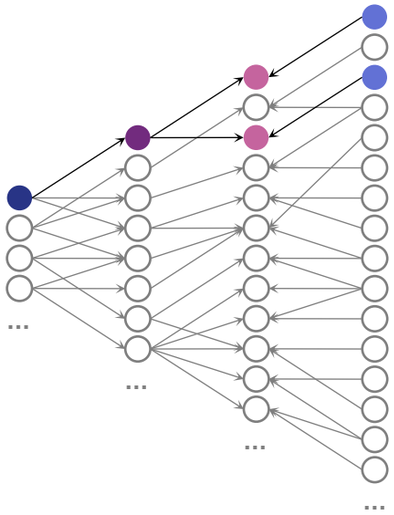
- **Goal:** process PGQL queries like,

```
SELECT a, c WHERE (a:author) -[:wrote]-> (b:paper) -[:cite]->
                  (c:paper) <-[:wrote]- (d:author)
```

"Pairs of authors who wrote papers that cite each other"

- **BFS vs. DFS tradeoff:** `(a) -> (b) -> (c) <- (d)`

DFS



1. Output the 1st matching a->b->c<-d
2. Output the 2st matching a->b->c<-d

3. Graph Analytics: aDFS [USENIX ATC '21]

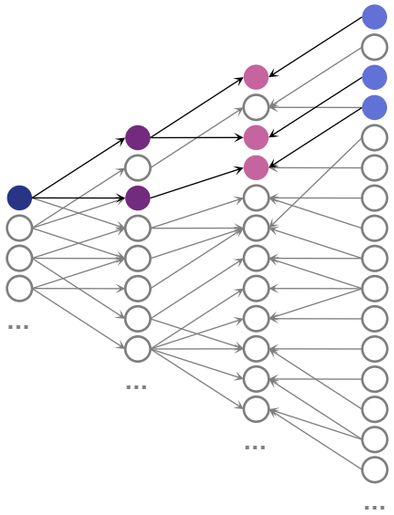
- **Goal:** process PGQL queries like,

```
SELECT a, c WHERE (a:author) -[:wrote]-> (b:paper) -[:cite]->
                  (c:paper) <-[:wrote]- (d:author)
```

"Pairs of authors who wrote papers that cite each other"

- **BFS vs. DFS tradeoff:** `(a) -> (b) -> (c) <- (d)`

DFS



1. Output the 1st matching a->b->c<-d
2. Output the 2st matching a->b->c<-d
3. Output the 3rd matching a->b->c<-d

3. Graph Analytics: aDFS [USENIX ATC '21]

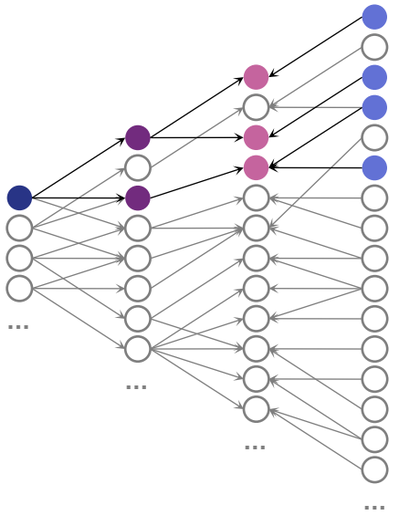
- **Goal:** process PGQL queries like,

```
SELECT a, c WHERE (a:author) -[:wrote]-> (b:paper) -[:cite]->
                  (c:paper) <-[:wrote]- (d:author)
```

"Pairs of authors who wrote papers that cite each other"

- **BFS vs. DFS tradeoff:** `(a) -> (b) -> (c) <- (d)`

DFS



1. Output the 1st matching a->b->c<-d
2. Output the 2st matching a->b->c<-d
3. Output the 3rd matching a->b->c<-d
4. …

3. Graph Analytics: aDFS [USENIX ATC '21]

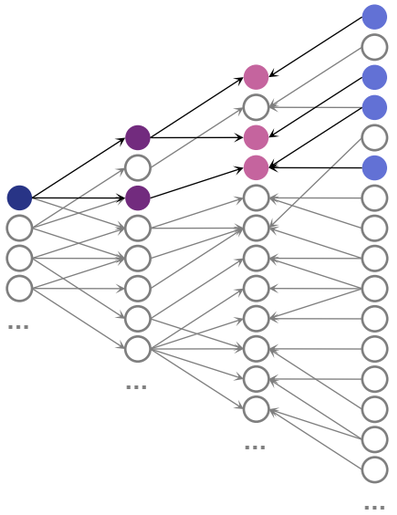
- **Goal:** process PGQL queries like,

```
SELECT a, c WHERE (a:author) -[:wrote]-> (b:paper) -[:cite]->
                  (c:paper) <-[:wrote]- (d:author)
```

"Pairs of authors who wrote papers that cite each other"

- **BFS vs. DFS tradeoff:** `(a) -> (b) -> (c) <- (d)`

DFS



1. Output the 1st matching a->b->c<-d
2. Output the 2st matching a->b->c<-d
3. Output the 3rd matching a->b->c<-d
4. …

🙂 **No intermediate results!**

3. Graph Analytics: aDFS [USENIX ATC '21]

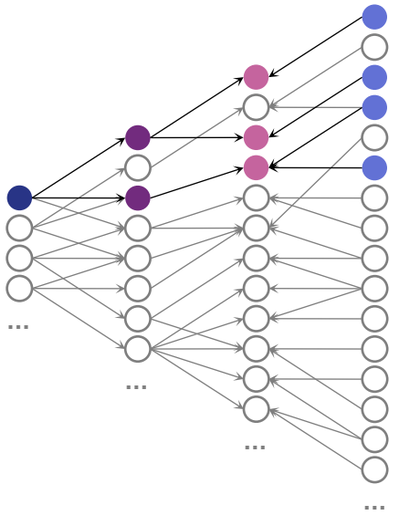
- **Goal:** process PGQL queries like,

```
SELECT a, c WHERE (a:author) -[:wrote]-> (b:paper) -[:cite]->
                  (c:paper) <-[:wrote]- (d:author)
```

"Pairs of authors who wrote papers that cite each other"

- **BFS vs. DFS tradeoff:** `(a) -> (b) -> (c) <- (d)`

DFS



1. Output the 1st matching a->b->c<-d
2. Output the 2st matching a->b->c<-d
3. Output the 3rd matching a->b->c<-d
4. …

🙂 **No intermediate results!**
☹ **Hard to parallelize**

3. Graph Analytics: aDFS [USENIX ATC '21]

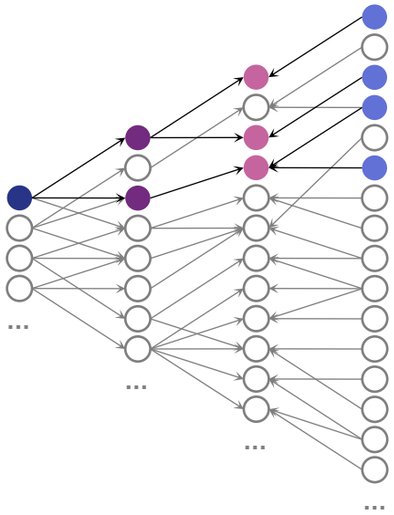
- **Goal:** process PGQL queries like,

```
SELECT a, c WHERE (a:author) -[:wrote]-> (b:paper) -[:cite]->
                  (c:paper) <-[:wrote]- (d:author)
```

"Pairs of authors who wrote papers that cite each other"

- **BFS vs. DFS tradeoff:** `(a) -> (b) -> (c) <- (d)`

DFS

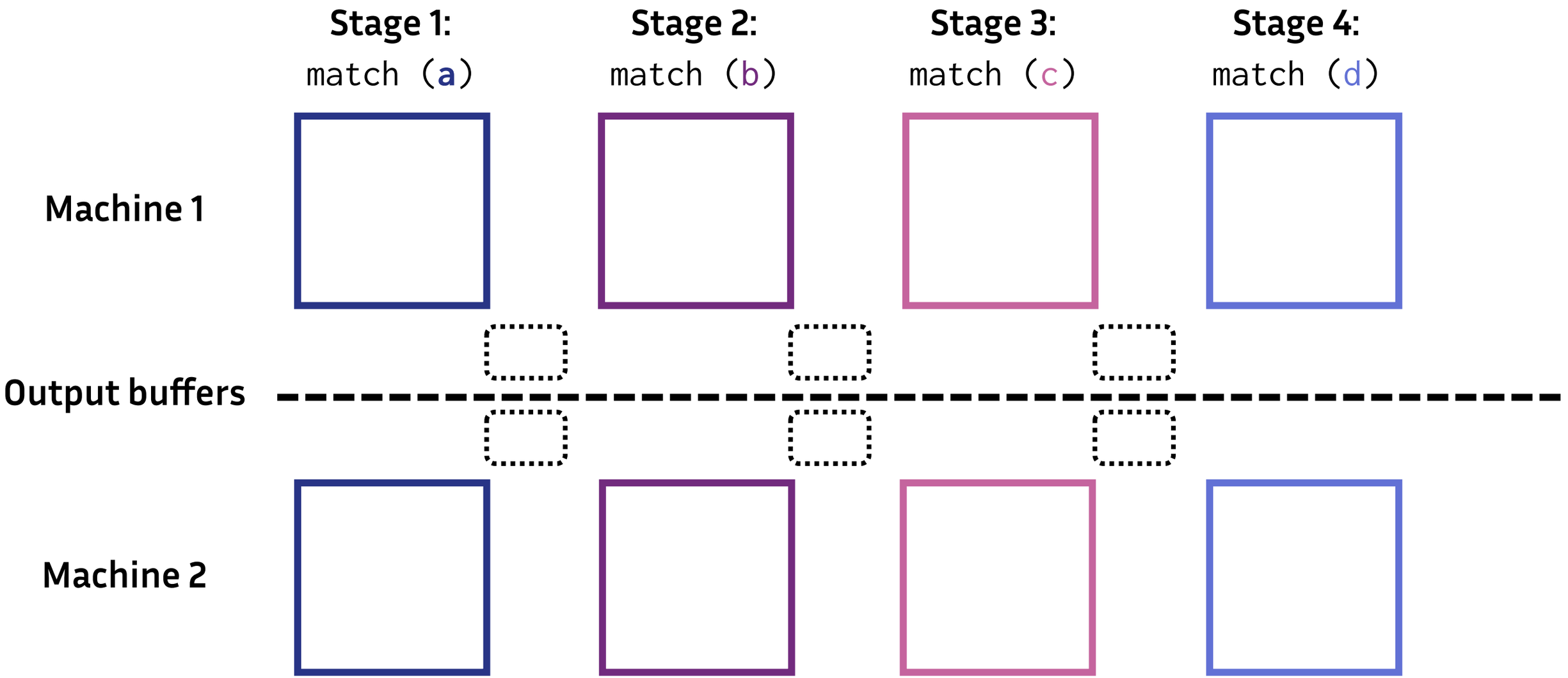


1. Output the 1st matching a->b->c<-d
2. Output the 2st matching a->b->c<-d
3. Output the 3rd matching a->b->c<-d
4. …

🙂 No intermediate results!
☹ Hard to parallelize

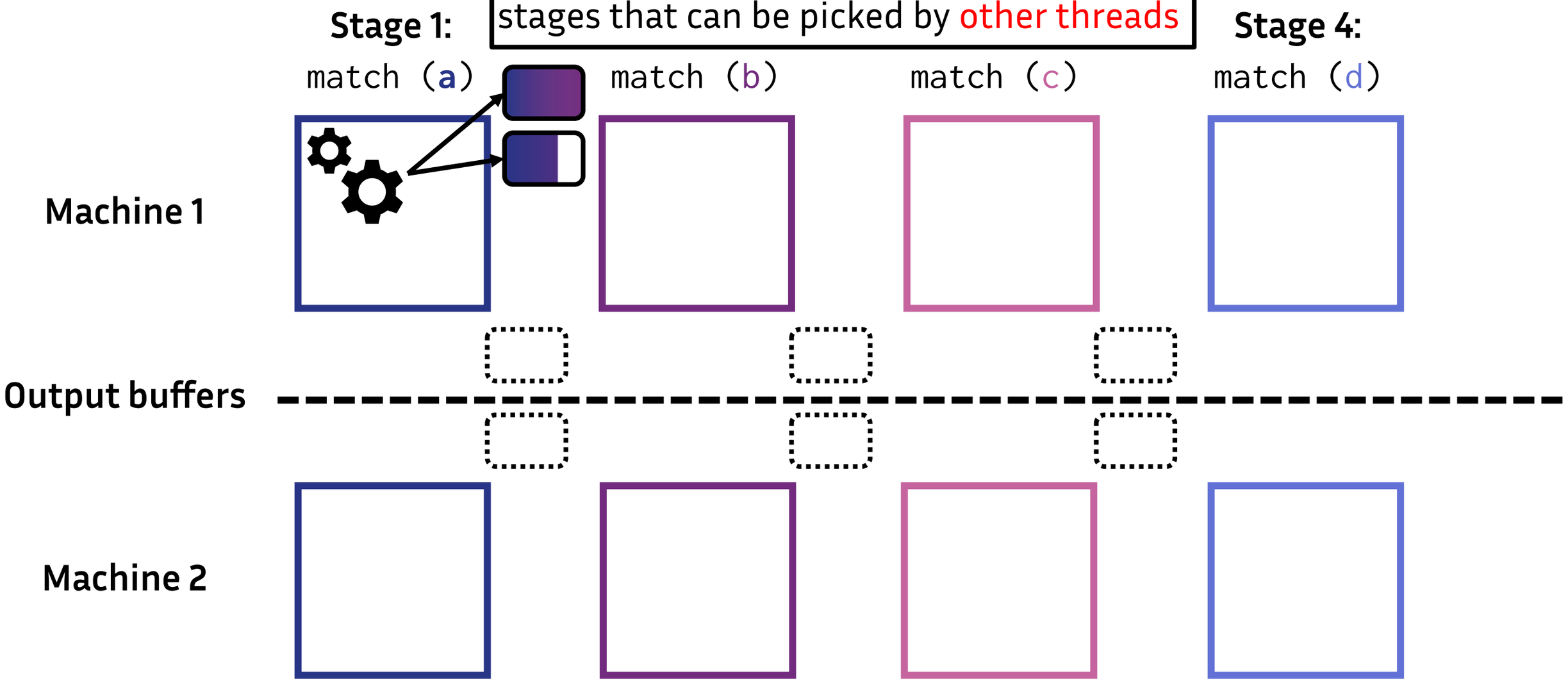
Insight of aDFS: just enough BFS to keep all cores busy!

3. Graph Analytics: aDFS [USENIX ATC '21]

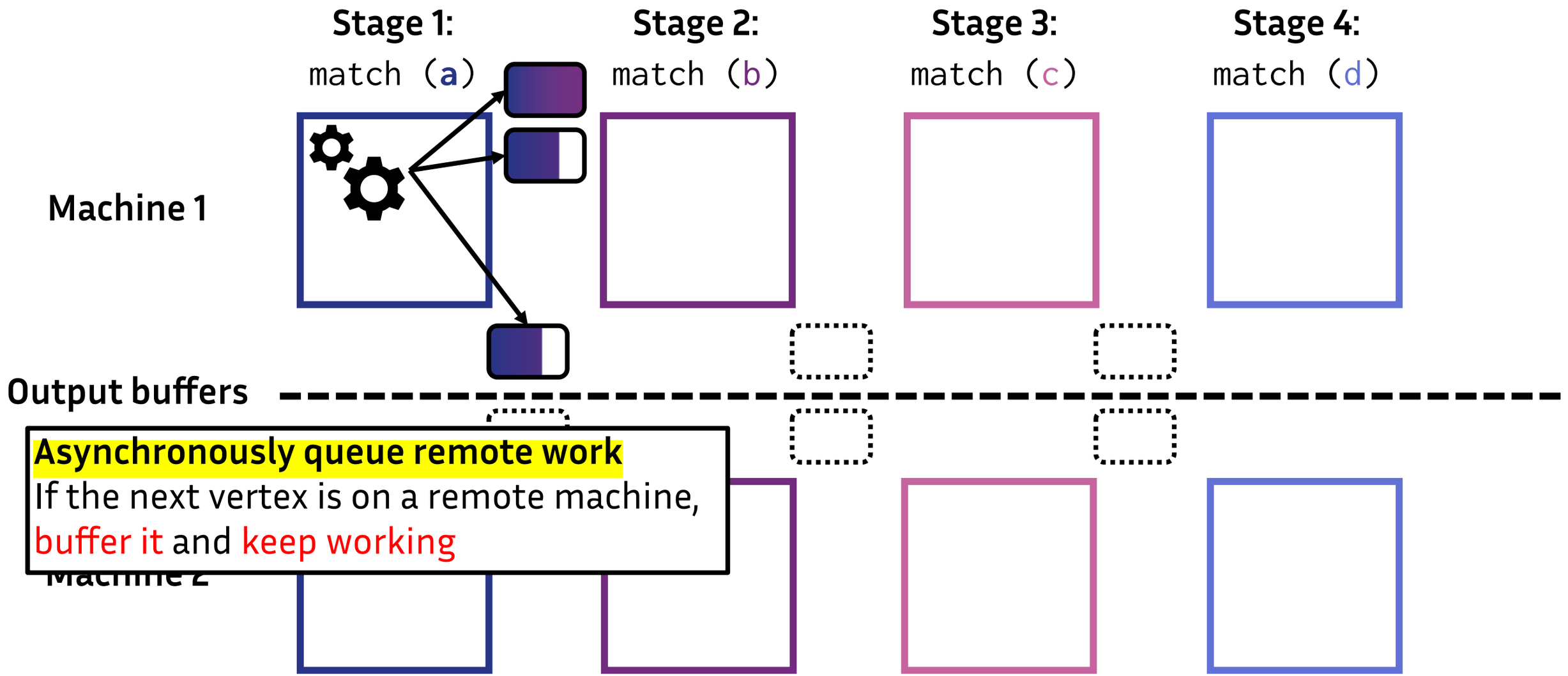


3. Graph Analytics:

Start with BFS
Not enough work to keep threads busy?
Buffer intermediate results for the next stages that can be picked by other threads



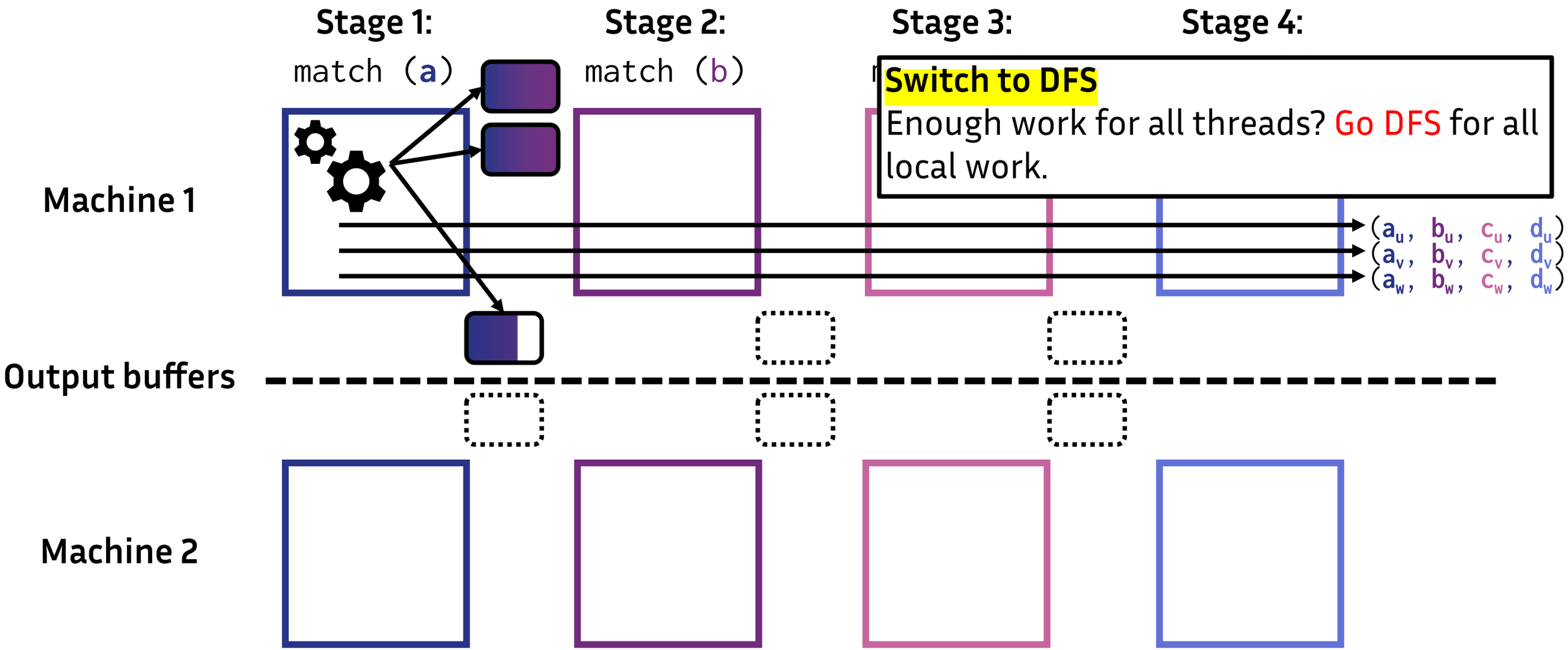
3. Graph Analytics: aDFS [USENIX ATC '21]



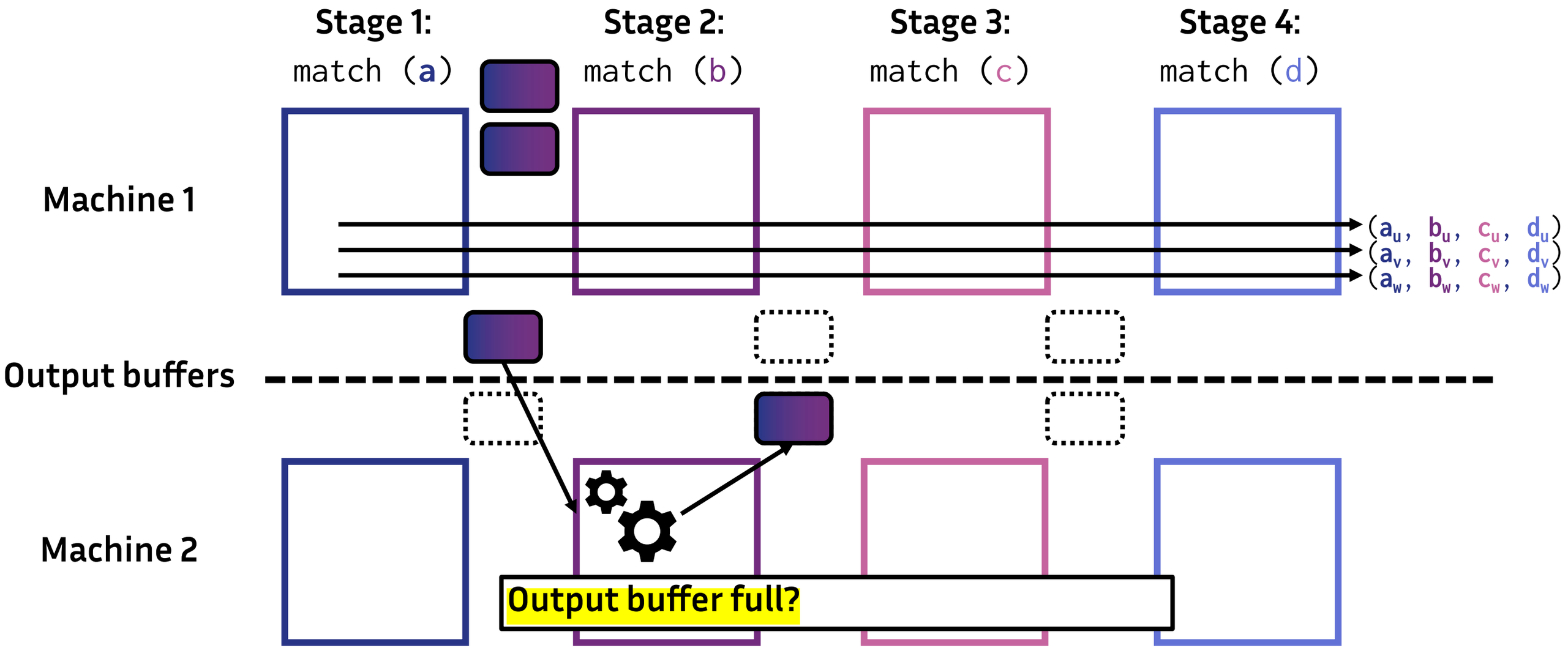
Asynchronously queue remote work

If the next vertex is on a remote machine, buffer it and keep working

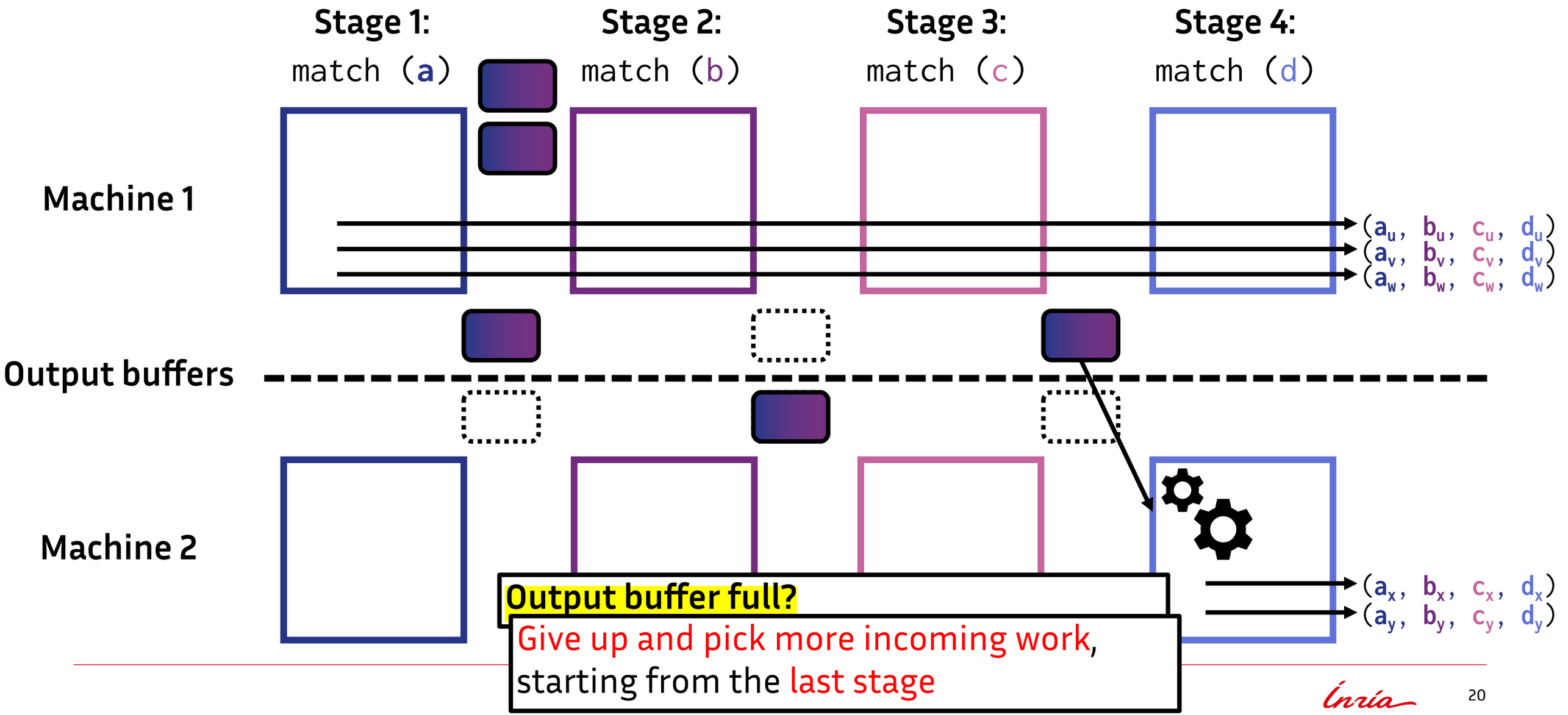
3. Graph Analytics: aDFS [USENIX ATC '21]



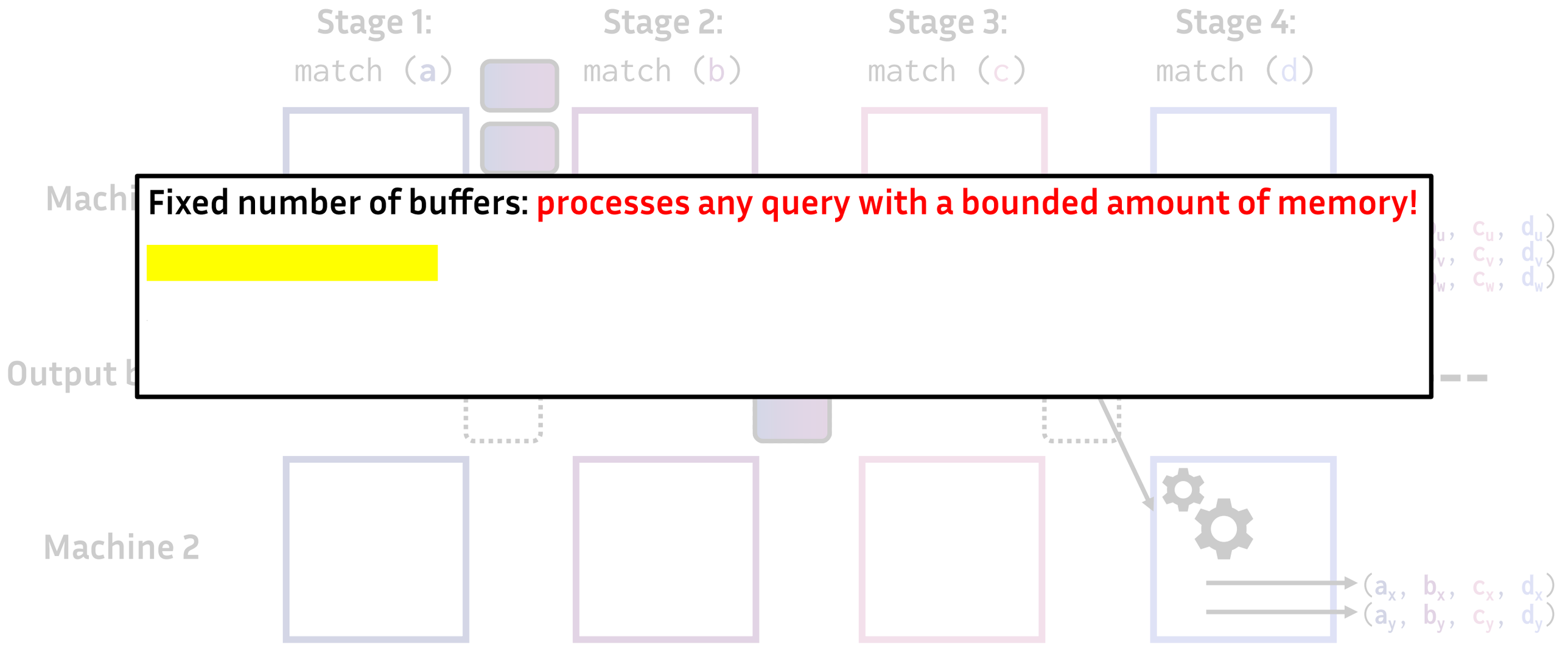
3. Graph Analytics: aDFS [USENIX ATC '21]



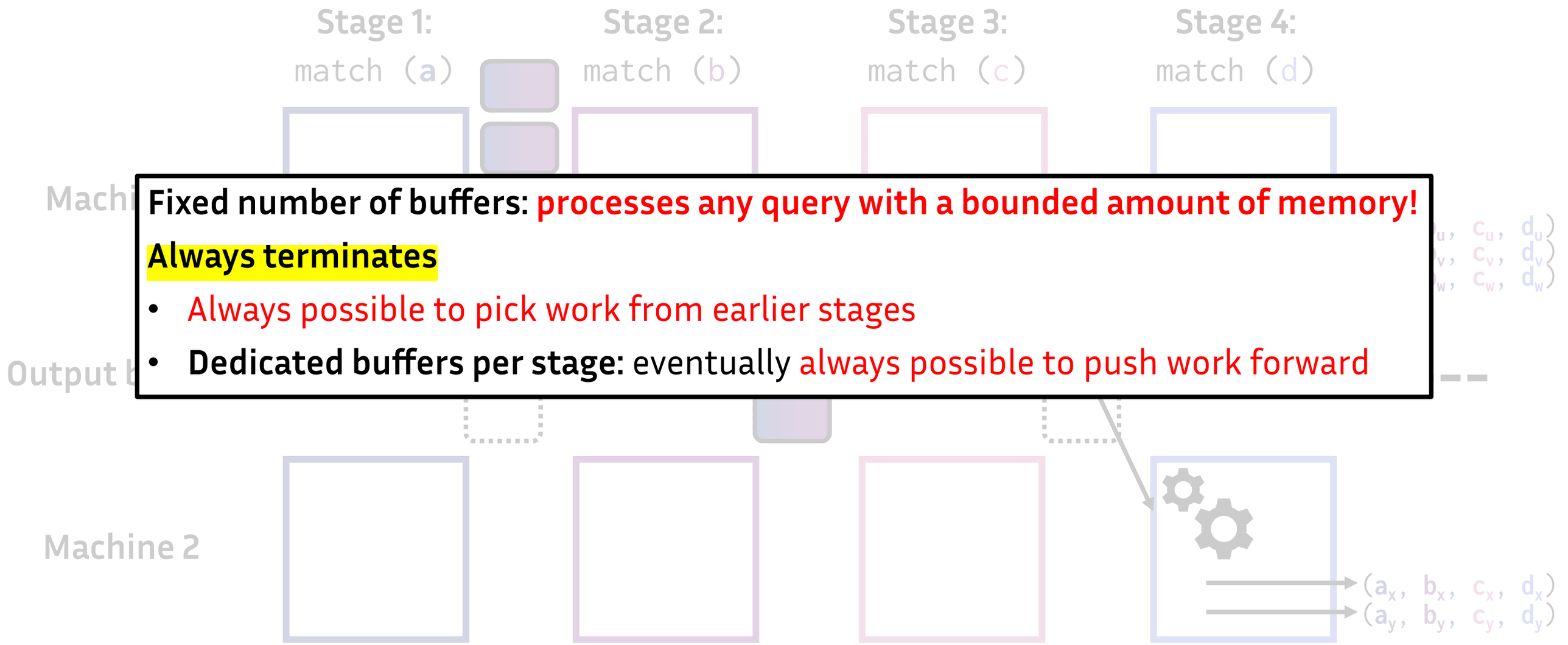
3. Graph Analytics: aDFS [USENIX ATC '21]



3. Graph Analytics: aDFS [USENIX ATC '21]



3. Graph Analytics: aDFS [USENIX ATC '21]



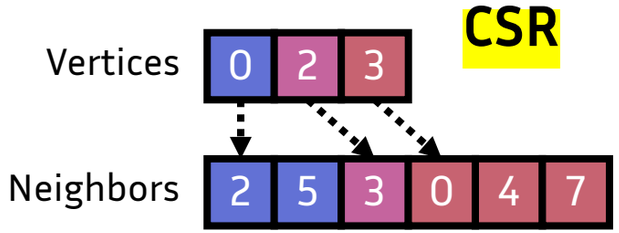
Fixed number of buffers: processes any query with a bounded amount of memory!

Always terminates

- Always possible to pick work from earlier stages
- **Dedicated buffers per stage:** eventually always possible to push work forward

3. Graph Analytics: Other Works

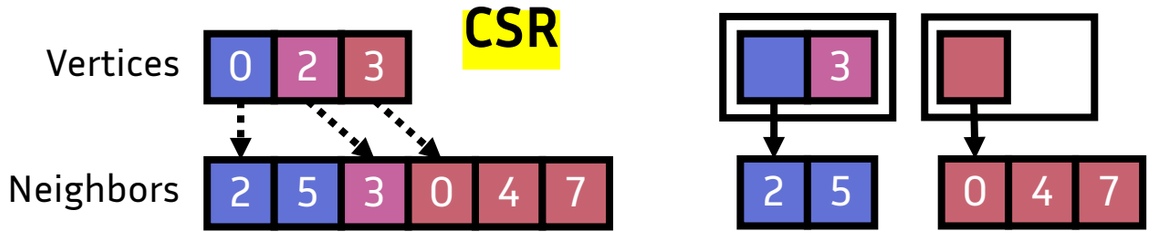
- **CSR++ [OPODIS '20]**



- CSR packs vertices and edges in two arrays for fast reads

3. Graph Analytics: Other Works

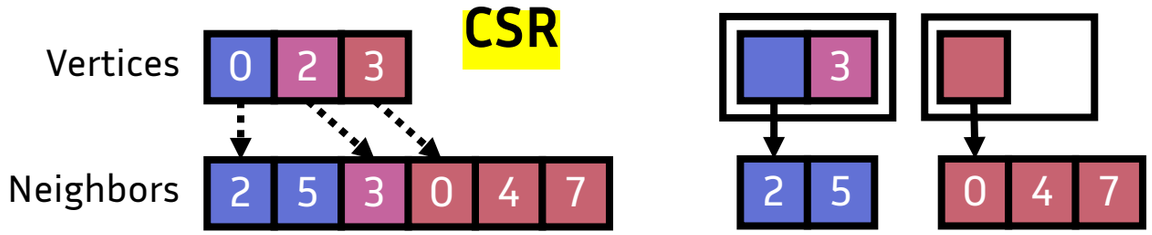
- **CSR++ [OPODIS '20]**



- CSR packs vertices and edges in two arrays for fast reads, CSR++ segments vertices to support **in-place updates**

3. Graph Analytics: Other Works

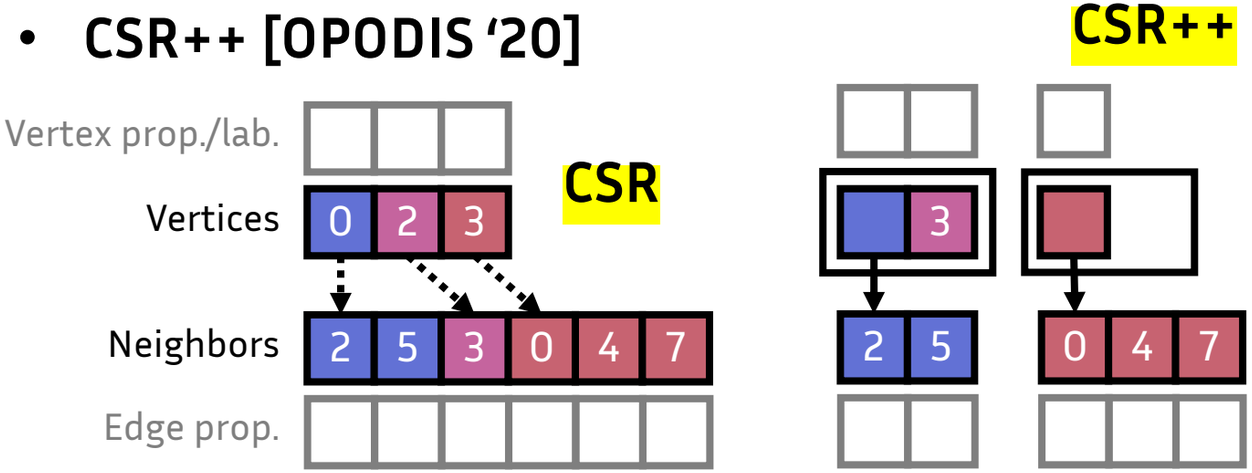
- **CSR++ [OPODIS '20]**



- CSR packs vertices and edges in two arrays for fast reads, CSR++ segments vertices to support **in-place updates**
- CSR++ keeps sorted neighbors for fast scans/deletions, inlines single neighbors

3. Graph Analytics: Other Works

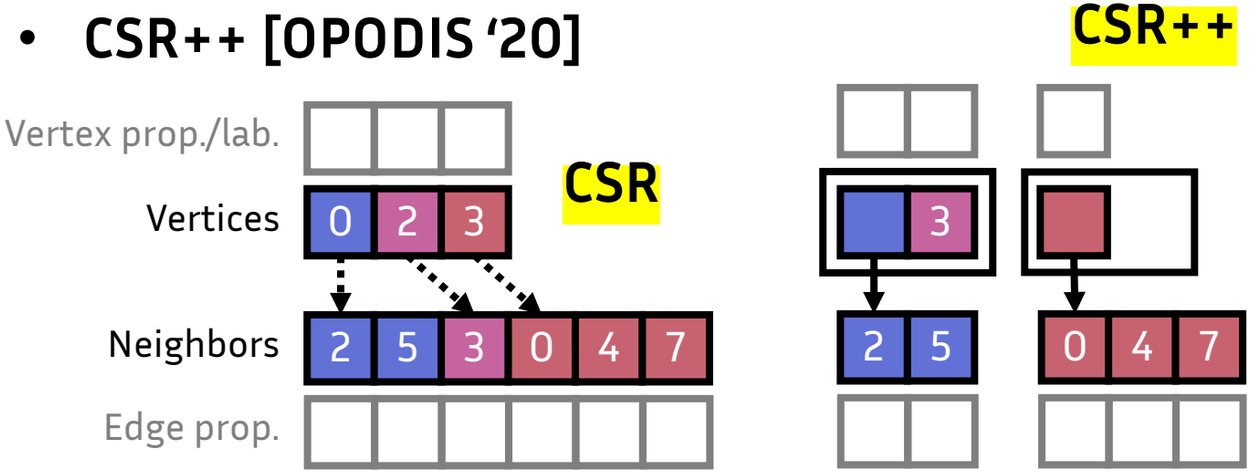
- **CSR++ [OPODIS '20]**



- CSR packs vertices and edges in two arrays for fast reads, CSR++ segments vertices to support **in-place updates**
- CSR++ keeps sorted neighbors for fast scans/deletions, inlines single neighbors
  - + supports parallel property/label arrays

3. Graph Analytics: Other Works

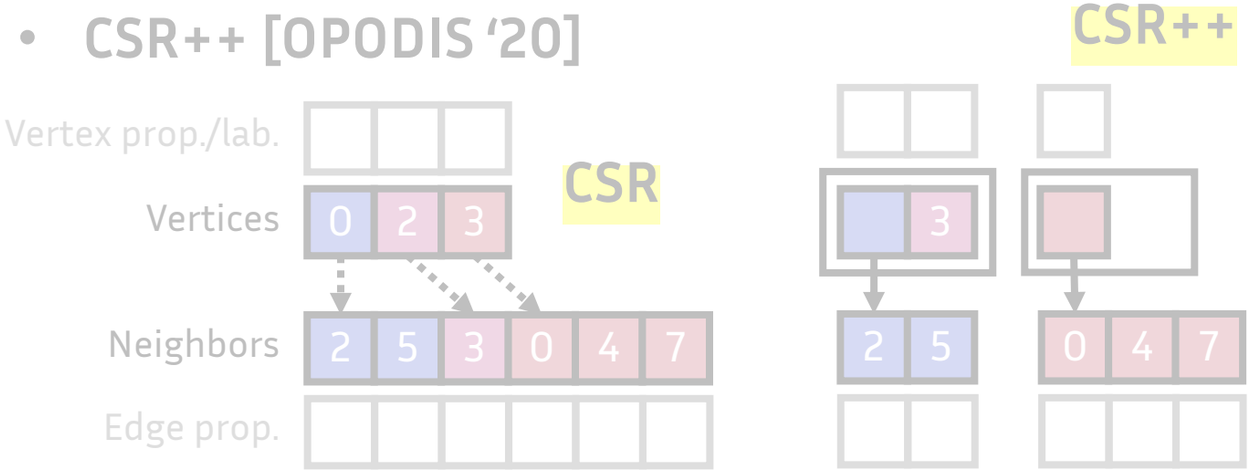
- **CSR++ [OPODIS '20]**



- CSR packs vertices and edges in two arrays for fast reads, CSR++ segments vertices to support **in-place updates**
- CSR++ keeps sorted neighbors for fast scans/deletions, inlines single neighbors
  - + supports parallel property/label arrays
- For batched updates (vs. **multiversioning**)

3. Graph Analytics: Other Works

- CSR++ [OPODIS '20]



- CSR packs vertices and edges in two arrays for fast reads, CSR++ segments vertices to support **in-place updates**
- CSR++ keeps sorted neighbors for fast scans/deletions, inlines single neighbors
  - + supports parallel property/label arrays
- For batched updates (vs. **multiversioning**)

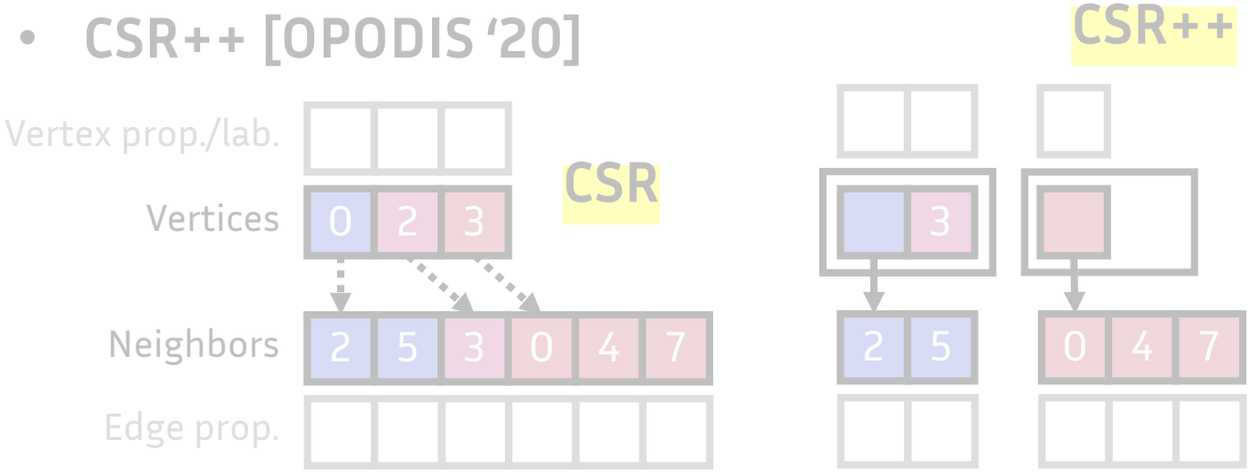
- **RLC Index [ICDE '23]**

`Q(A14, A19, (withdrawal·deposit)+)` ✅

- Is t reachable from s by a path whose edge labels match a regular expression `(L)+`?

3. Graph Analytics: Other Works

- CSR++ [OPODIS '20]



- CSR packs vertices and edges in two arrays for fast reads, CSR++ segments vertices to support **in-place updates**
- CSR++ keeps sorted neighbors for fast scans/deletions, inlines single neighbors
  - + supports parallel property/label arrays
- For batched updates (vs. **multiversioning**)

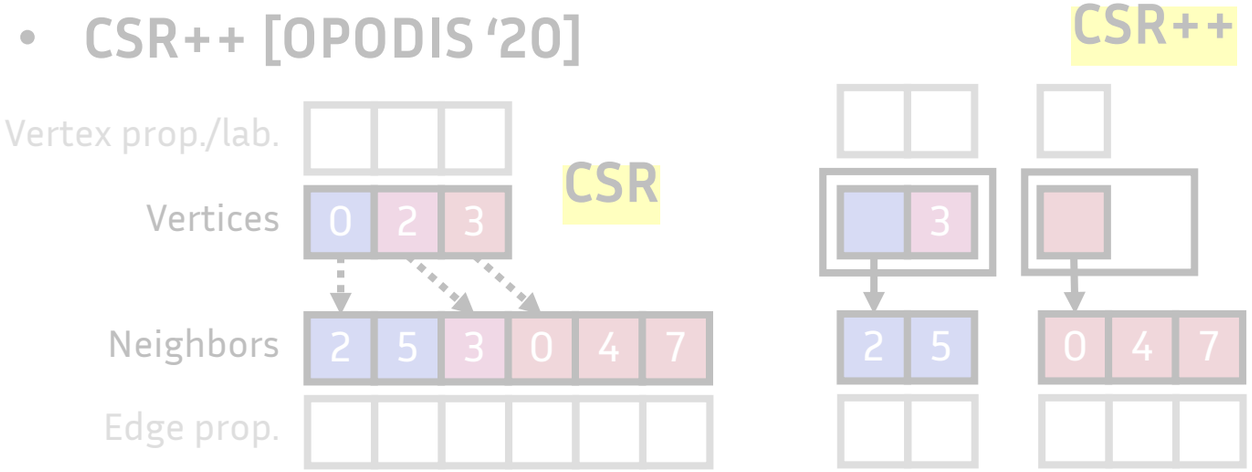
- **RLC Index [ICDE '23]**

`Q(A14, A19, (withdrawal·deposit)+)` ✅

- Is t reachable from s by a path whose edge labels match a regular expression $(L)+$?
- The transitive closure (all reachable pairs + patterns) explodes in size

3. Graph Analytics: Other Works

- CSR++ [OPODIS '20]



- CSR packs vertices and edges in two arrays for fast reads, CSR++ segments vertices to support **in-place updates**
- CSR++ keeps sorted neighbors for fast scans/deletions, inlines single neighbors
  - + supports parallel property/label arrays
- For batched updates (vs. **multiversioning**)

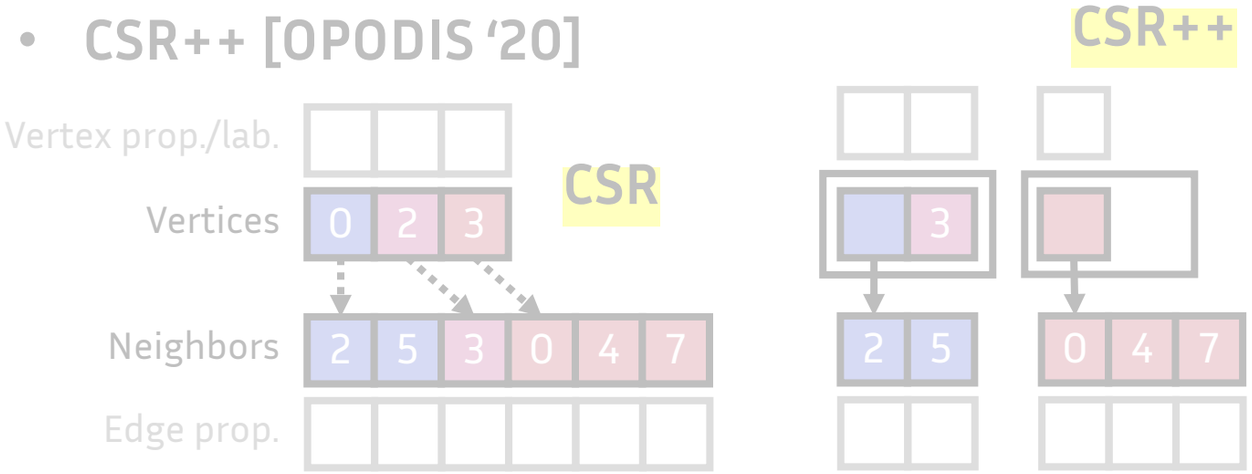
- **RLC Index [ICDE '23]**

Q(A14, A19, (withdrawal·deposit)+) ✅

- Is t reachable from s by a path whose edge labels match a regular expression (L)+?
- The transitive closure (all reachable pairs + patterns) explodes in size
- **RLC index:** hub nodes each vertex reaches (+patterns), queries intersect two sorted lists

3. Graph Analytics: Other Works

- CSR++ [OPODIS '20]



- CSR packs vertices and edges in two arrays for fast reads, CSR++ segments vertices to support **in-place updates**
- CSR++ keeps sorted neighbors for fast scans/deletions, inlines single neighbors
  - + supports parallel property/label arrays
- For batched updates (vs. **multiversioning**)

- **RLC Index [ICDE '23]**

Q(A14, A19, (withdrawal·deposit)+) ✅

- Is t reachable from s by a path whose edge labels match a regular expression (L)+?
- The transitive closure (all reachable pairs + patterns) explodes in size
- **RLC index:** hub nodes each vertex reaches (+patterns), queries intersect two sorted lists
- Hubs are high-degree nodes; includes pruning rules; tractable due to max L length

Optimization Mechanisms

		Ensuring work conservation	Preferring high-frequency cores	Optimizing inter-thread synchronization	Avoiding critical path preemptions	Leveraging the spinning vs. blocking tradeoff	Optimizing parallelization patterns	Ensuring high memory locality	Sequential performance improvements
Chapter 1 Task scheduling	"Wasted cores"	●							
	Ipanema	●							
	NUMA placement							●	
	Frequency inversions		●						
	Nest		●			●		●	
Chapter 2 Task sync.	RCL			●	●		●	●	
	FlexGuard			●	●	●			
	Tapestry			●	●	●		●	
Chapter 3 Graph analytics	Smart arrays							●	●
	CSR++			●			●	●	
	aDFS						●		
	RLC Index								●

Optimization Mechanisms

		Ensuring work conservation	Preferring high-frequency cores	Optimizing inter-thread synchronization	Avoiding critical path preemptions	Leveraging the spinning vs. blocking tradeoff	Optimizing parallelization patterns	Ensuring high memory locality	Sequential performance improvements
Chapter 1 Task scheduling	"Wasted cores"	●							
	Ipanema	●							
	NUMA placement							●	
	Frequency inversions		●						
	Nest		●			●		●	
Chapter 2 Task sync.	RCL			●	●		●	●	
	FlexGuard			●	●	●			
	Tapestry			●	●	●		●	
Chapter 3 Graph analytics	Smart arrays							●	●
	CSR++			●			●	●	
	aDFS						●		
	RLC Index								●

Optimization Mechanisms

		Ensuring work conservation	Preferring high-frequency cores	Optimizing inter-thread synchronization	Avoiding critical path preemptions	Leveraging the spinning vs. blocking tradeoff	Optimizing parallelization patterns	Ensuring high memory locality	Sequential performance improvements
Chapter 1 Task scheduling	"Wasted cores"	●							
	Ipanema	●							
	NUMA placement							●	
	Frequency inversions		●						
	Nest		●			●		●	
Chapter 2 Task sync.	RCL			●	●		●	●	
	FlexGuard			●	●	●			
	Tapestry			●	●	●		●	
Chapter 3 Graph analytics	Smart arrays							●	●
	CSR++			●			●	●	
	aDFS						●		
	RLC Index								●

Optimization Mechanisms

		Ensuring work conservation	Preferring high-frequency cores	Optimizing inter-thread synchronization	Avoiding critical path preemptions	Leveraging the spinning vs. blocking tradeoff	Optimizing parallelization patterns	Ensuring high memory locality	Sequential performance improvements
Chapter 1 Task scheduling	"Wasted cores"	●							
	Ipanema	●							
	NUMA placement							●	
	Frequency inversions		●						
	Nest		●				●		●	
Chapter 2 Task sync.	RCL			●	●		●	●	
	FlexGuard			●	●	●			
	Tapestry			●	●	●		●	
Chapter 3 Graph analytics	Smart arrays							●	●
	CSR++			●			●	●	
	aDFS						●		
	RLC Index								●

Optimization Mechanisms

		Ensuring work conservation	Preferring high-frequency cores	Optimizing inter-thread synchronization	Avoiding critical path preemptions	Leveraging the spinning vs. blocking tradeoff	Optimizing parallelization patterns	Ensuring high memory locality	Sequential performance improvements
Chapter 1 Task scheduling	"Wasted cores"	●							
	Ipanema	●							
	NUMA placement							●	
	Frequency inversions		●						
	Nest		●			●		●	
Chapter 2 Task sync.	RCL			●	●		●	●	
	FlexGuard			●	●	●			
	Tapestry			●	●	●		●	
Chapter 3 Graph analytics	Smart arrays							●	●
	CSR++			●			●	●	
	aDFS						●		
	RLC Index								●

Optimization Mechanisms

		Ensuring work conservation	Preferring high-frequency cores	Optimizing inter-thread synchronization	Avoiding critical path preemptions	Leveraging the spinning vs. blocking tradeoff	Optimizing parallelization patterns	Ensuring high memory locality	Sequential performance improvements
Chapter 1 Task scheduling	"Wasted cores"	●							
	Ipanema	●							
	NUMA placement							●	
	Frequency inversions		●						
	Nest		●			●		●	
Chapter 2 Task sync.	RCL			●	●		●	●	
	FlexGuard			●	●	●			
	Tapestry			●	●	●		●	
Chapter 3 Graph analytics	Smart arrays							●	●
	CSR++			●			●	●	
	aDFS						●		
	RLC Index								●

Optimization Mechanisms

		Ensuring work conservation	Preferring high-frequency cores	Optimizing inter-thread synchronization	Avoiding critical path preemptions	Leveraging the spinning vs. blocking tradeoff	Optimizing parallelization patterns	Ensuring high memory locality	Sequential performance improvements
Chapter 1 Task scheduling	"Wasted cores"	●							
	Ipanema	●							
	NUMA placement							●	
	Frequency inversions		●						
	Nest		●			●		●	
Chapter 2 Task sync.	RCL			●	●		●	●	
	FlexGuard			●	●	●			
	Tapestry			●	●	●		●	
Chapter 3 Graph analytics	Smart arrays							●	●
	CSR++			●			●	●	
	aDFS						●		
	RLC Index								●

Optimization Mechanisms

		Ensuring work conservation	Preferring high-frequency cores	Optimizing inter-thread synchronization	Avoiding critical path preemptions	Leveraging the spinning vs. blocking tradeoff	Optimizing parallelization patterns	Ensuring high memory locality	Sequential performance improvements
Chapter 1 Task scheduling	"Wasted cores"	●							
	Ipanema	●							
	NUMA placement							●	
	Frequency inversions		●						
	Nest		●			●		●	
Chapter 2 Task sync.	RCL			●	●		●	●	
	FlexGuard			●	●	●			
	Tapestry			●	●	●		●	
Chapter 3 Graph analytics	Smart arrays							●	●
	CSR++			●			●	●	
	aDFS						●		
	RLC Index								●

Optimization Mechanisms

		Ensuring work conservation	Preferring high-frequency cores	Optimizing inter-thread synchronization	Avoiding critical path preemptions	Leveraging the spinning vs. blocking tradeoff	Optimizing parallelization patterns	Ensuring high memory locality	Sequential performance improvements
Chapter 1 Task scheduling	"Wasted cores"	●							
	Ipanema	●							
	NUMA placement							●	
	Frequency inversions		●						
	Nest		●			●		●	
Chapter 2 Task sync.	RCL			●	●		●	●	
	FlexGuard			●	●	●			
	Tapestry			●	●	●		●	
Chapter 3 Graph analytics	Smart arrays							●	●
	CSR++			●			●	●	
	aDFS						●		
	RLC Index								●

Thank you!

12 years of hunting inefficiencies across the software stack and finding **principled solutions**

Thank you!

12 years of hunting inefficiencies across the software stack and finding **principled solutions**

- **Community work:** includes 18+ program committees
  - SOSP, ASPLOS (x3), EuroSys (x5), USENIX ATC (x3), DSN (x2)...

Thank you!

12 years of hunting inefficiencies across the software stack and finding **principled solutions**

- **Community work:** includes 18+ program committees
  - SOSP, ASPLOS (x3), EuroSys (x5), USENIX ATC (x3), DSN (x2)…
- **Teaching** (including 2 years as Maître de Conférences), **engineering** (5 years at Oracle)

Thank you!

12 years of hunting inefficiencies across the software stack and finding **principled solutions**

- **Community work:** includes 18+ program committees
  - SOSP, ASPLOS (x3), EuroSys (x5), USENIX ATC (x3), DSN (x2)…
- **Teaching** (including 2 years as Maître de Conférences), **engineering** (5 years at Oracle)
- And of course, I wouldn't be where I am without the **students I co-supervize(d)**:

PhD Students

- Damien Thénot
- Papa Assane Fall
- Himadri Chhaya-Shailesh
- Keisuke Nishimura
- Victor Laforet
- Maxime Collette
- Tara Aggoun

Postdocs

- Tomáš Faltín
- William Wu