Queen Mary
University of London

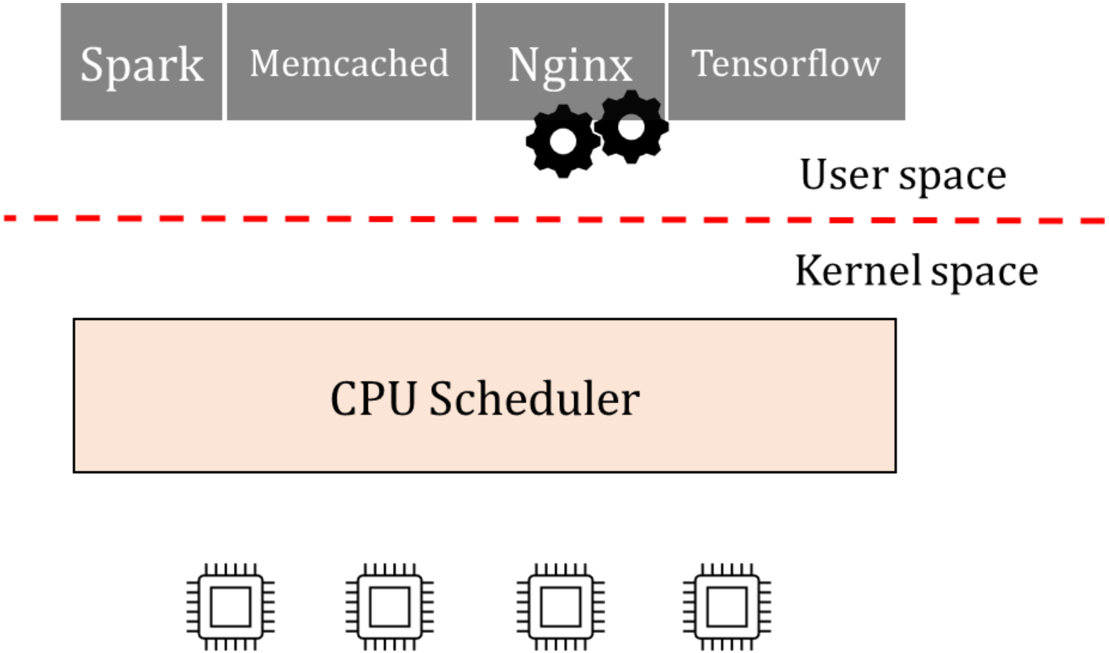
CPU Schedulers are Interesting

Alireza Sanaee (QMUL)

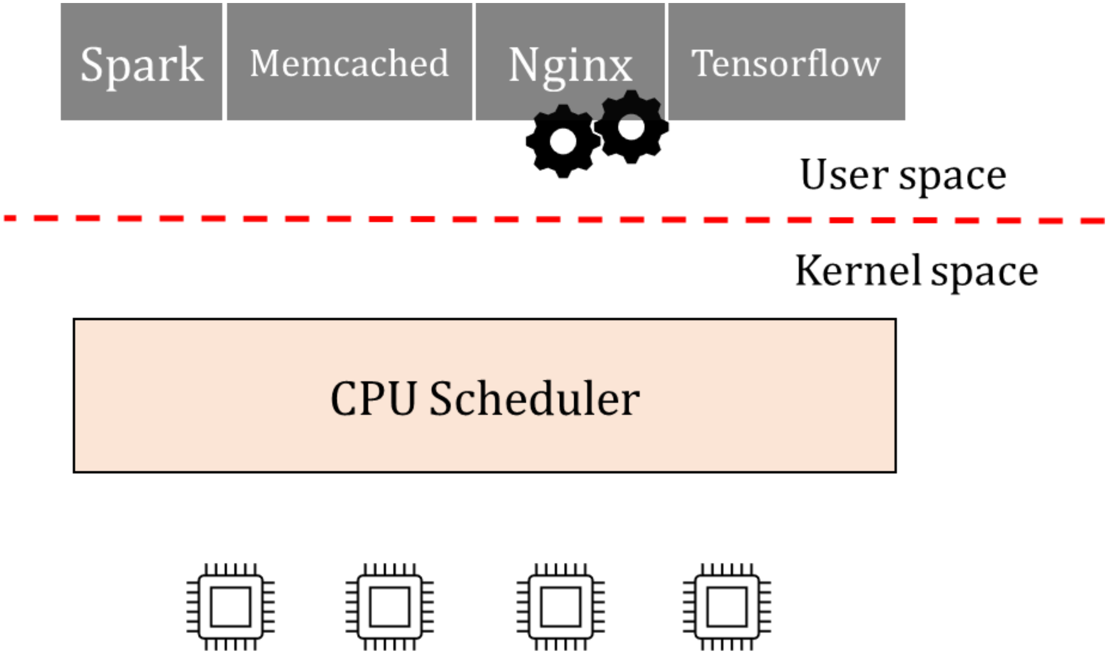
Work in collaboration with
Jack Humphries (Stanford),
Sebastiano Miano (QMUL),
Christos Kozyrakis (Stanford),
Gianni Antichi (QMUL)

What a CPU scheduler is:

What a CPU scheduler is:



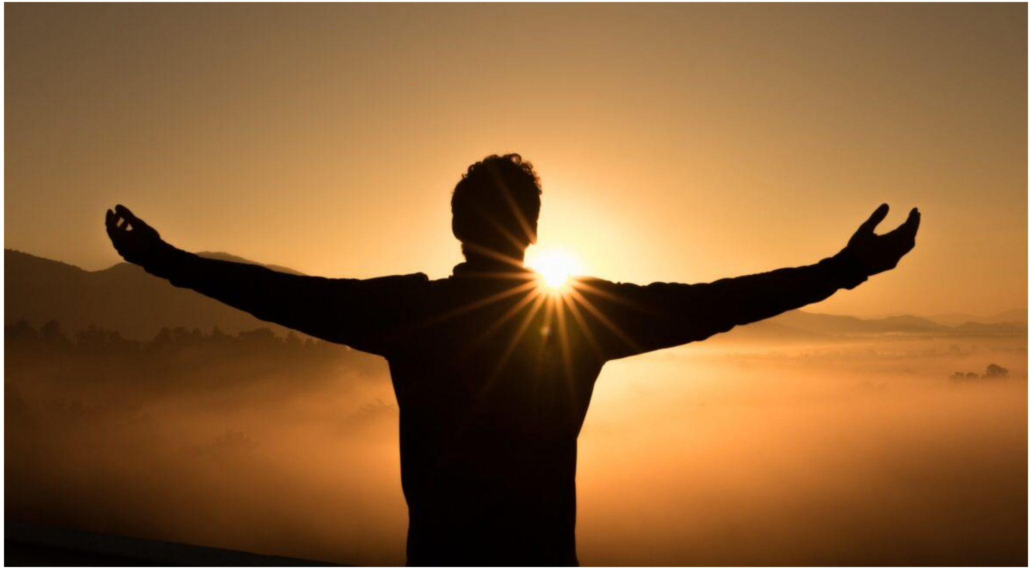
What a CPU scheduler is:



CPU scheduler decides who gets the CPU next!

Why is CPU scheduling important now?

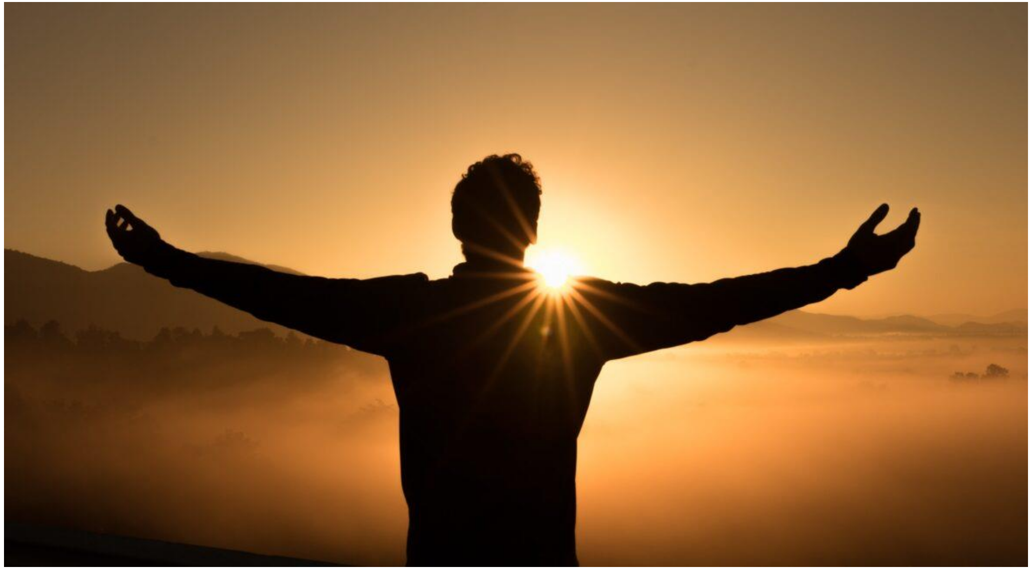
Why is CPU scheduling important now?



[ghOSt, SOSP’21]

Server machines typically have 256 or 512 logical cores at data centers!

Why is CPU scheduling important now?

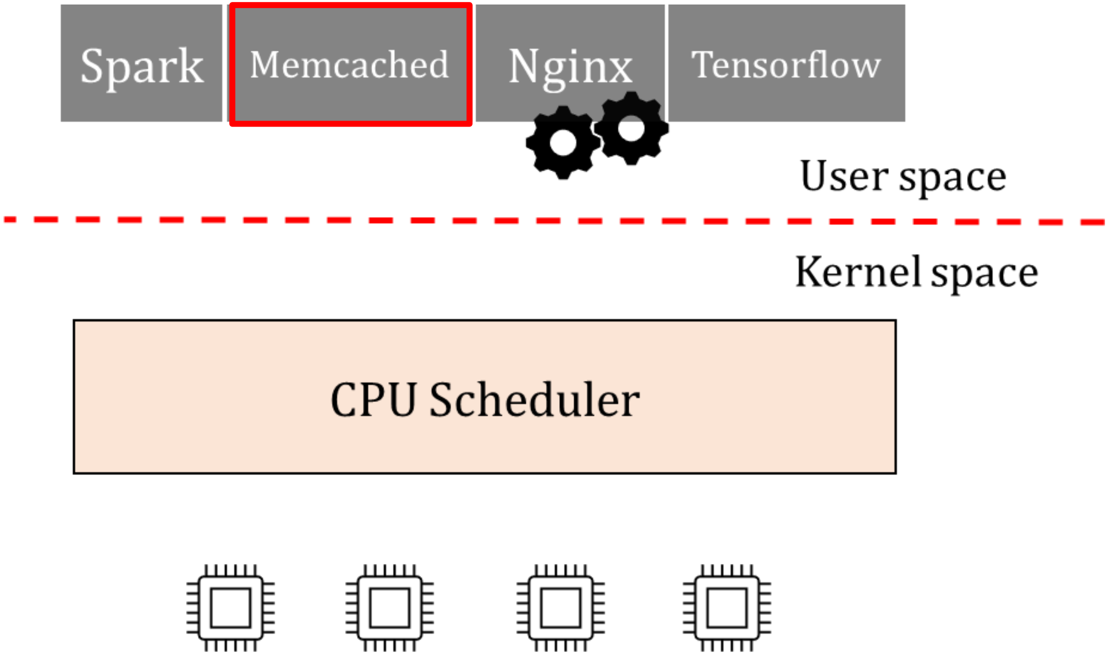


[ghOSt, SOSP'21]

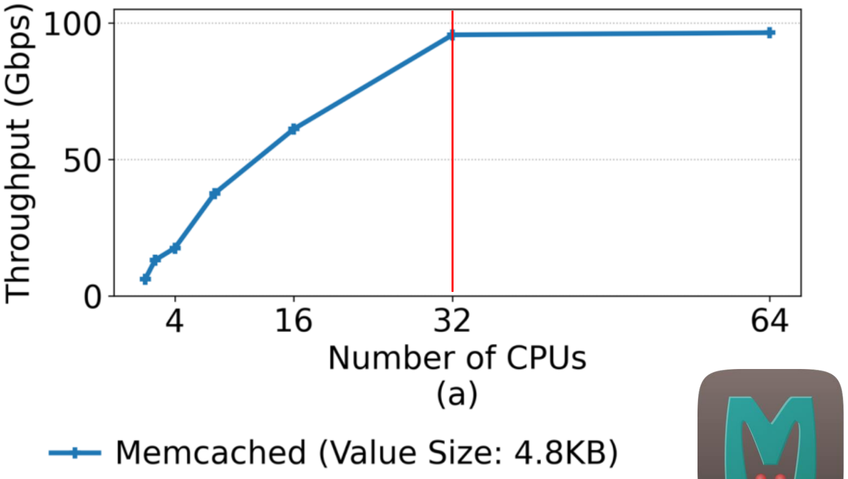
Server machines typically have 256 or 512 logical cores at data centers!

I have lots of cores! Who cares?

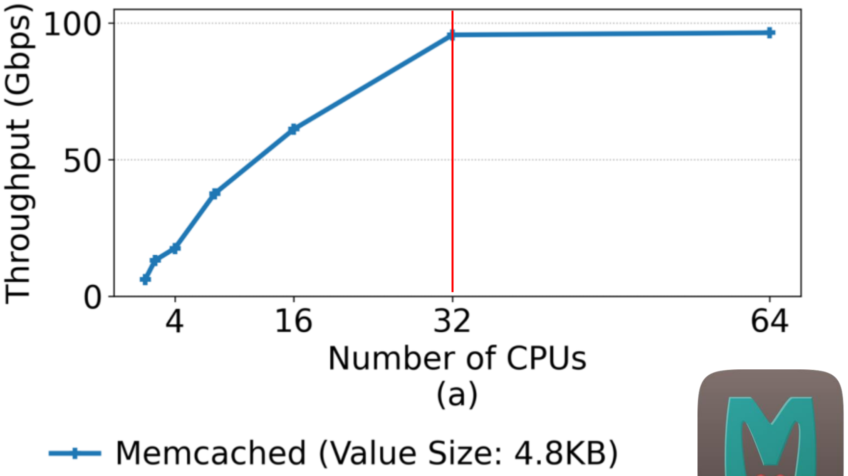
What does it take for an application to achieve 100 Gbps?



Workload demand is high!



Workload demand is high!

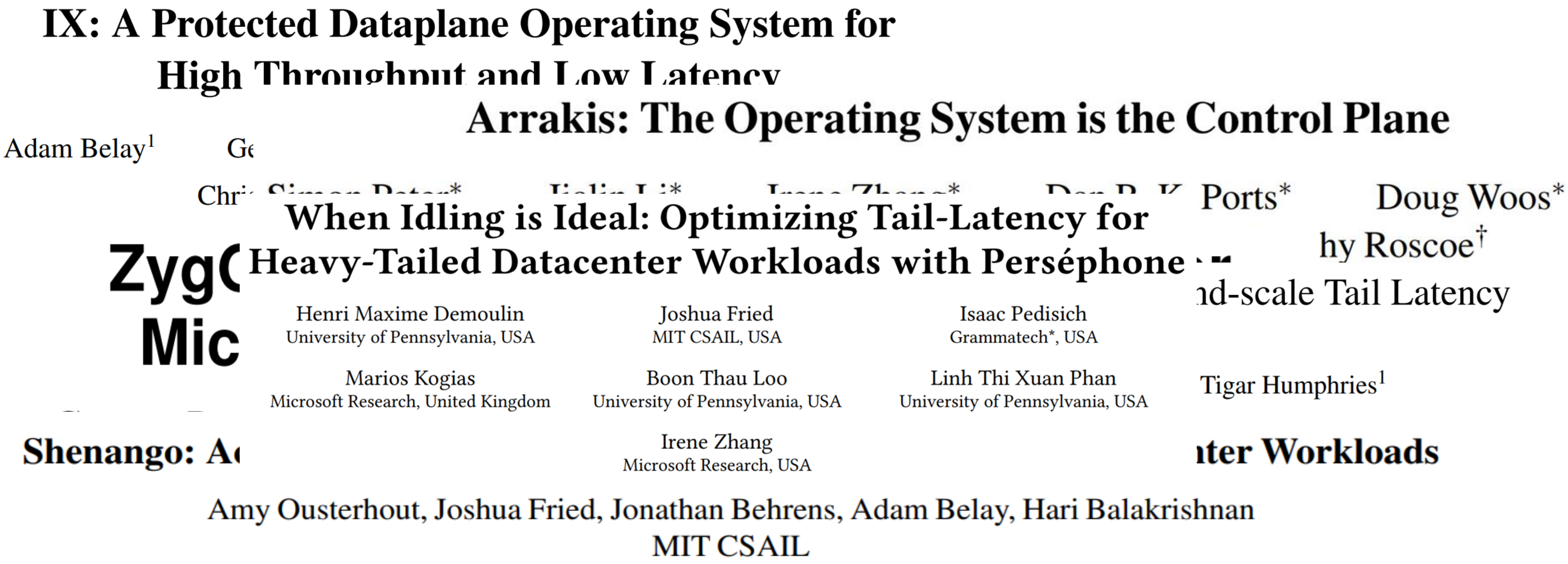


Memcached needs 32 cores to achieve 100 Gbps with large values

There is a poor guy, **CPU scheduler**!

Needs to do a lot of things, so FAST

We do have fast CPU scheduling mechanisms, don’t we?



IX: A Protected Dataplane Operating System for High Throughput and Low Latency

Adam Belay[1] Ge

Chri

Arrakis: The Operating System is the Control Plane

Simon Peter* Jialin Li* Irene Zhang* Dan R. K. Ports* Doug Woos*

hy Roscoe[†]

Zyg

Mic

nd-scale Tail Latency

Tigar Humphries[1]

When Idling is Ideal: Optimizing Tail-Latency for Heavy-Tailed Datacenter Workloads with Perséphone

Henri Maxime Demoulin
University of Pennsylvania, USA

Joshua Fried
MIT CSAIL, USA

Isaac Pedisich
Grammatech*, USA

Marios Kogias
Microsoft Research, United Kingdom

Boon Thau Loo
University of Pennsylvania, USA

Linh Thi Xuan Phan
University of Pennsylvania, USA

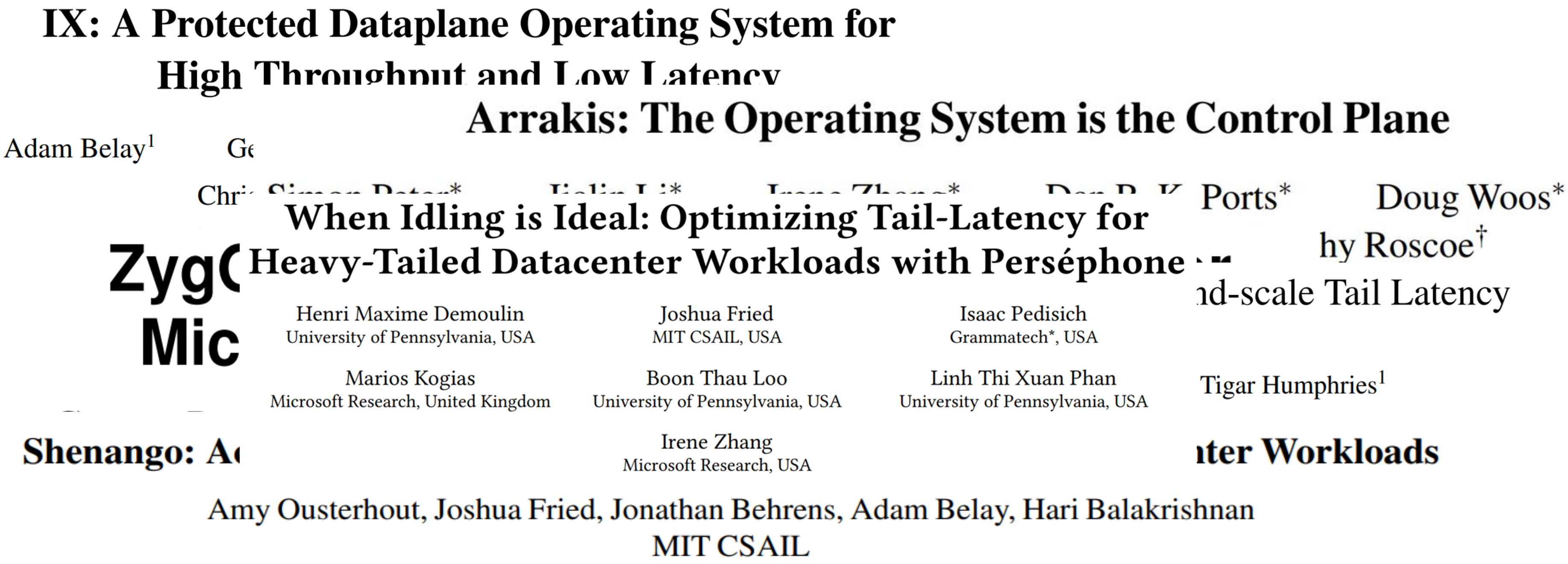
Irene Zhang
Microsoft Research, USA

Shenango: A

ter Workloads

Amy Ousterhout, Joshua Fried, Jonathan Behrens, Adam Belay, Hari Balakrishnan
MIT CSAIL

We do have fast CPU scheduling mechanisms, don't we?



IX: A Protected Dataplane Operating System for High Throughput and Low Latency

Adam Belay[1] Ge

Arrakis: The Operating System is the Control Plane

Chri Simon Peter* Jialin Li* Irene Zhang* Dan R. K. Ports* Doug Woos*

hy Roscoe†

When Idling is Ideal: Optimizing Tail-Latency for Heavy-Tailed Datacenter Workloads with Perséphone

Henri Maxime Demoulin
University of Pennsylvania, USA

Joshua Fried
MIT CSAIL, USA

Isaac Pedisich
Grammatech*, USA

Marios Kogias
Microsoft Research, United Kingdom

Boon Thau Loo
University of Pennsylvania, USA

Linh Thi Xuan Phan
University of Pennsylvania, USA

Irene Zhang
Microsoft Research, USA

Zyg(Mic

nd-scale Tail Latency

Tigar Humphries[1]

Shenango: A ter Workloads

Amy Ousterhout, Joshua Fried, Jonathan Behrens, Adam Belay, Hari Balakrishnan
MIT CSAIL

These systems are highly specialized for a particular application.

It is not only about one or two types of applications!

APACHE Spark TM
NGINX
druid
pinot
TIMESCALE

It is not only about one or two types of applications!

A single application with different workloads may need multiple policies

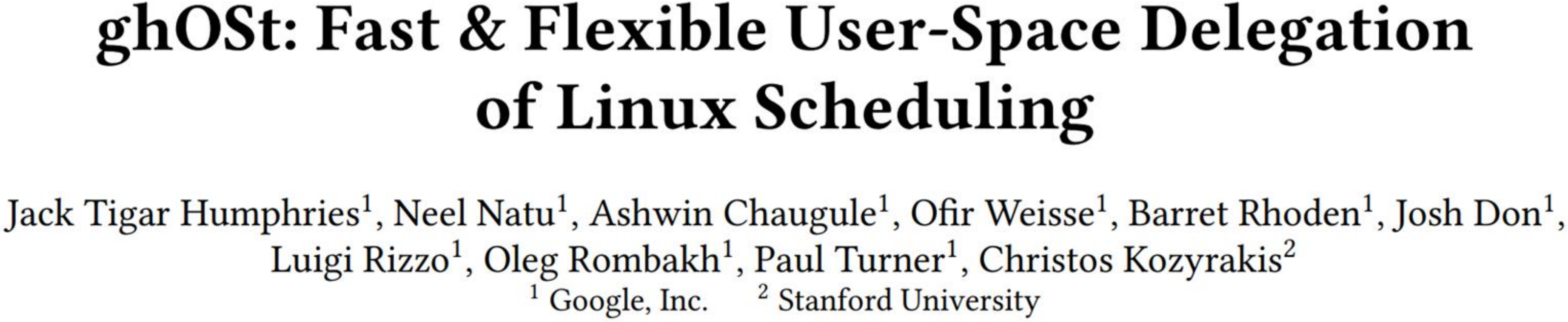
It is not only about one or two types of applications!

A single application with different workloads may need multiple policies

Co-located applications just exacerbates the situation

We need flexibility in enforcing policies for different applications and workloads

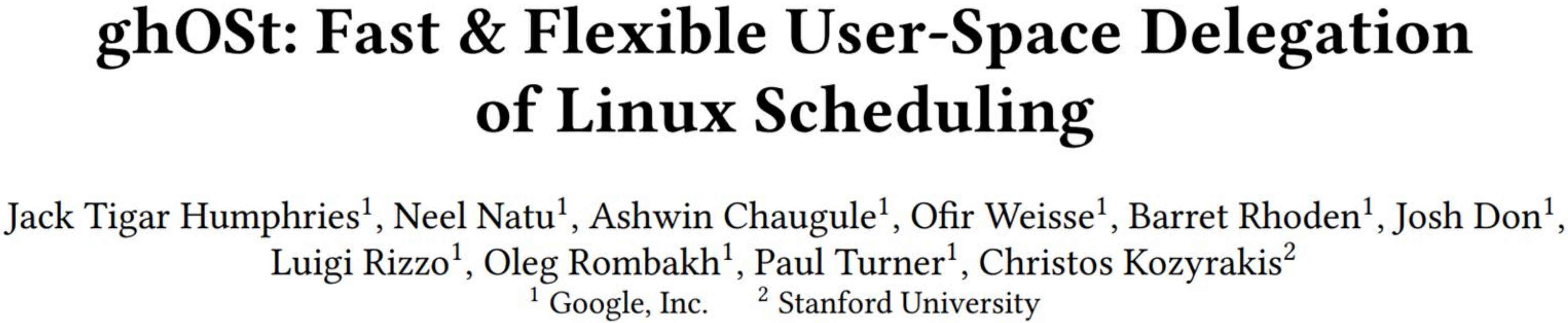
We need flexibility in enforcing policies for different applications and workloads



ghOSt: Fast & Flexible User-Space Delegation of Linux Scheduling

Jack Tigar Humphries[1], Neel Natu[1], Ashwin Chaugule[1], Ofir Weisse[1], Barret Rhoden[1], Josh Don[1], Luigi Rizzo[1], Oleg Rombakh[1], Paul Turner[1], Christos Kozyrakis[2]
[1] Google, Inc. [2] Stanford University

We need flexibility in enforcing policies for different applications and workloads

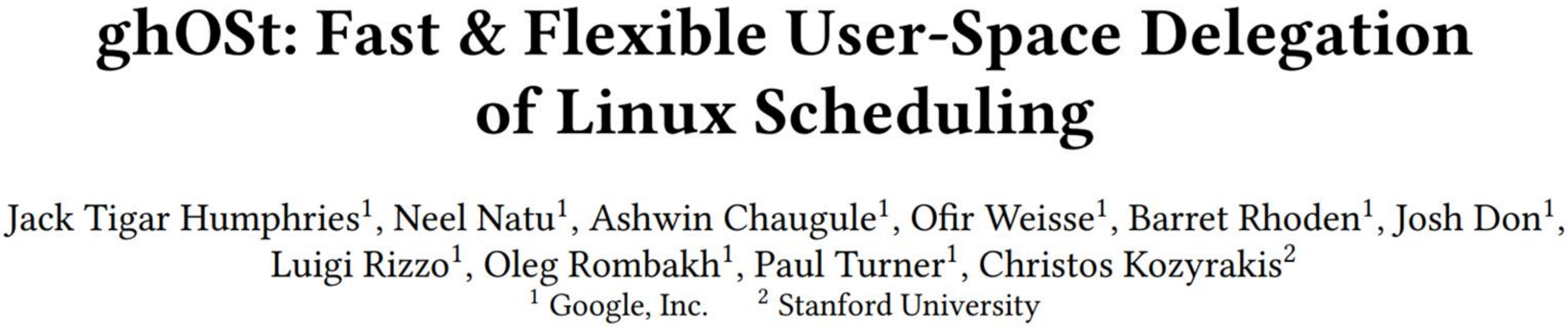


ghOSt: Fast & Flexible User-Space Delegation of Linux Scheduling

Jack Tigar Humphries[1], Neel Natu[1], Ashwin Chaugule[1], Ofir Weisse[1], Barret Rhoden[1], Josh Don[1], Luigi Rizzo[1], Oleg Rombakh[1], Paul Turner[1], Christos Kozyrakis[2]
[1] Google, Inc. [2] Stanford University

***Flexible** policy enforcement*

We need flexibility in enforcing policies for different applications and workloads



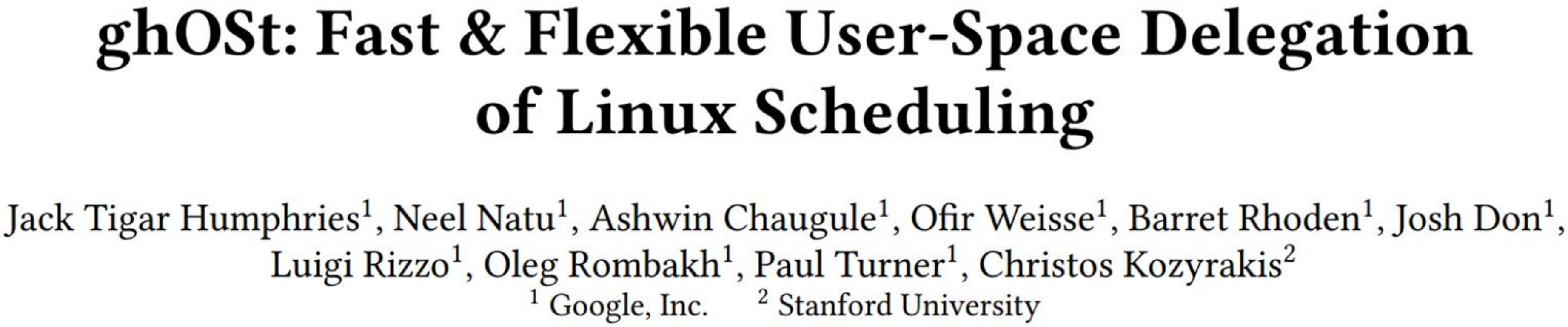
ghOSt: Fast & Flexible User-Space Delegation of Linux Scheduling

Jack Tigar Humphries[1], Neel Natu[1], Ashwin Chaugule[1], Ofir Weisse[1], Barret Rhoden[1], Josh Don[1],
Luigi Rizzo[1], Oleg Rombakh[1], Paul Turner[1], Christos Kozyrakis[2]
[1] Google, Inc. [2] Stanford University

Flexible *policy enforcement*

Ease of policy development in the ***userspace***

We need flexibility in enforcing policies for different applications and workloads



ghOSt: Fast & Flexible User-Space Delegation of Linux Scheduling

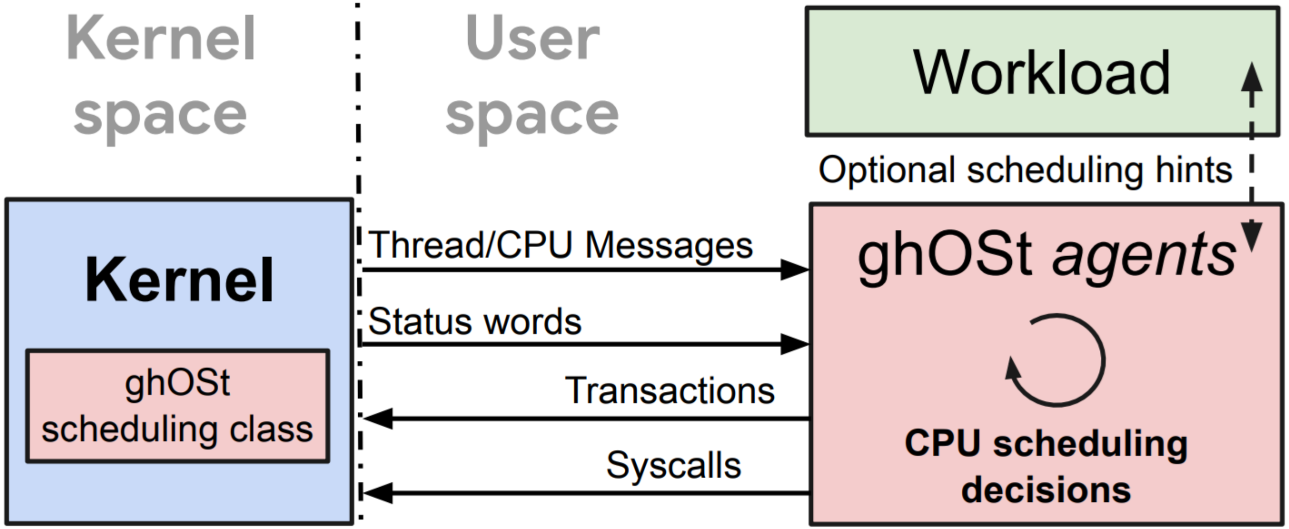
Jack Tigar Humphries[1], Neel Natu[1], Ashwin Chaugule[1], Ofir Weisse[1], Barret Rhoden[1], Josh Don[1], Luigi Rizzo[1], Oleg Rombakh[1], Paul Turner[1], Christos Kozyrakis[2]
[1] Google, Inc. [2] Stanford University

***Flexible** policy enforcement*

Ease of policy development in the ***userspace***

***No change** to the legacy applications*

How does ghOSt work?



Oh wait! but these do not come for FREE!

Consequences of user space scheduling

High Latency

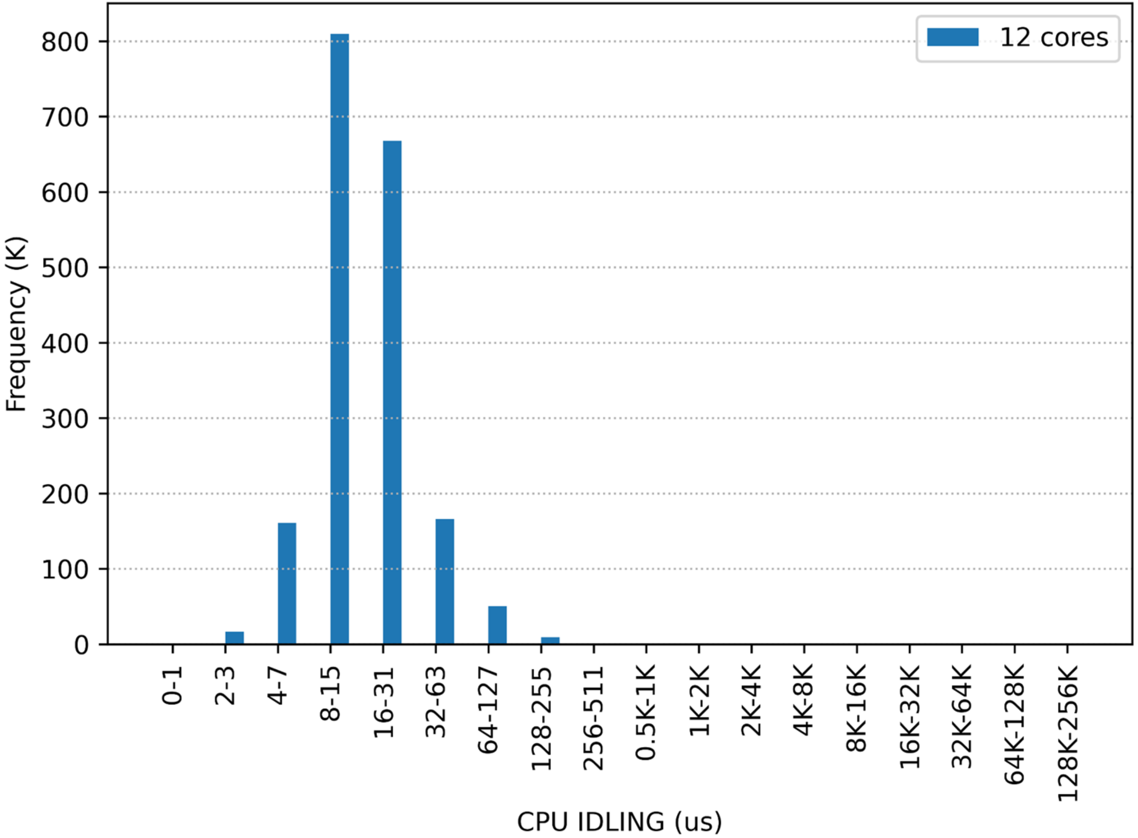
Wasted Compute

Consequences of user space scheduling

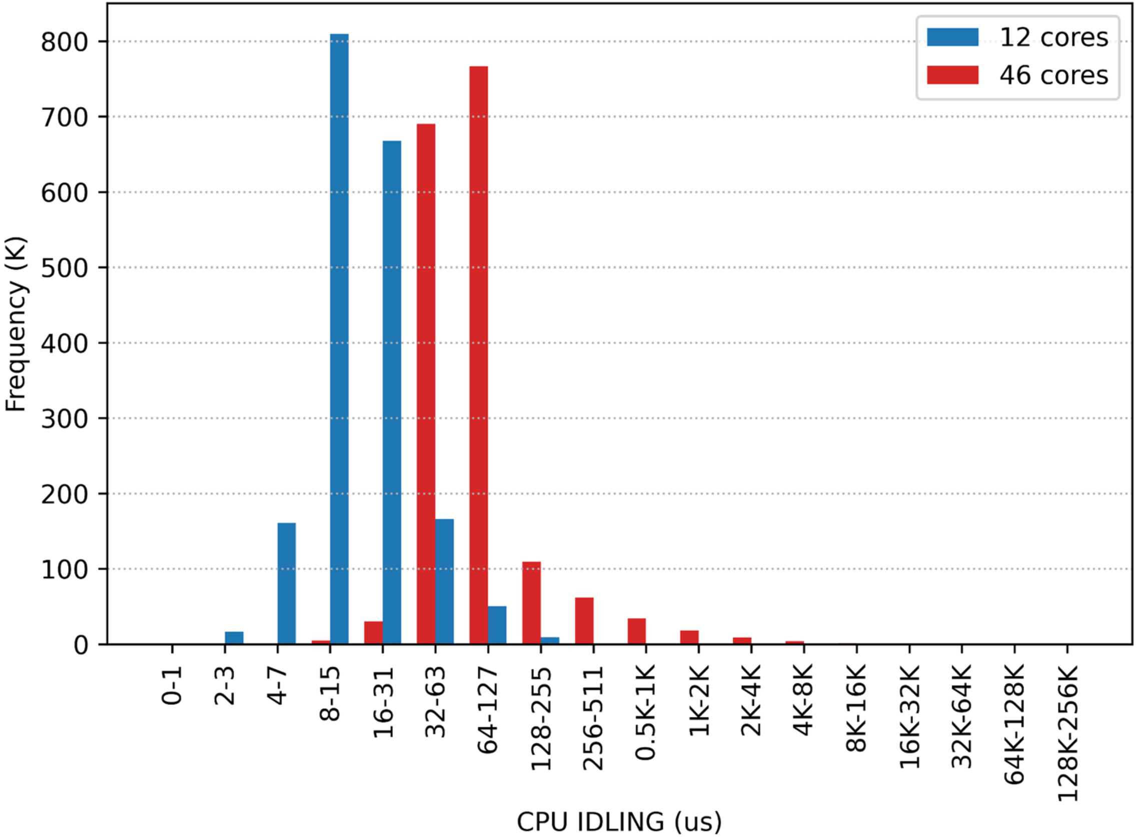
High Latency

Wasted Compute

IDLING


12 cores
Frequency (K)
800
700
600
500
400
300
200
100
0
0-1
2-3
4-7
8-15
16-31
32-63
64-127
128-255
256-511
0.5K-1K
1K-2K
2K-4K
4K-8K
8K-16K
16K-32K
32K-64K
64K-128K
128K-256K
CPU IDLING (us)

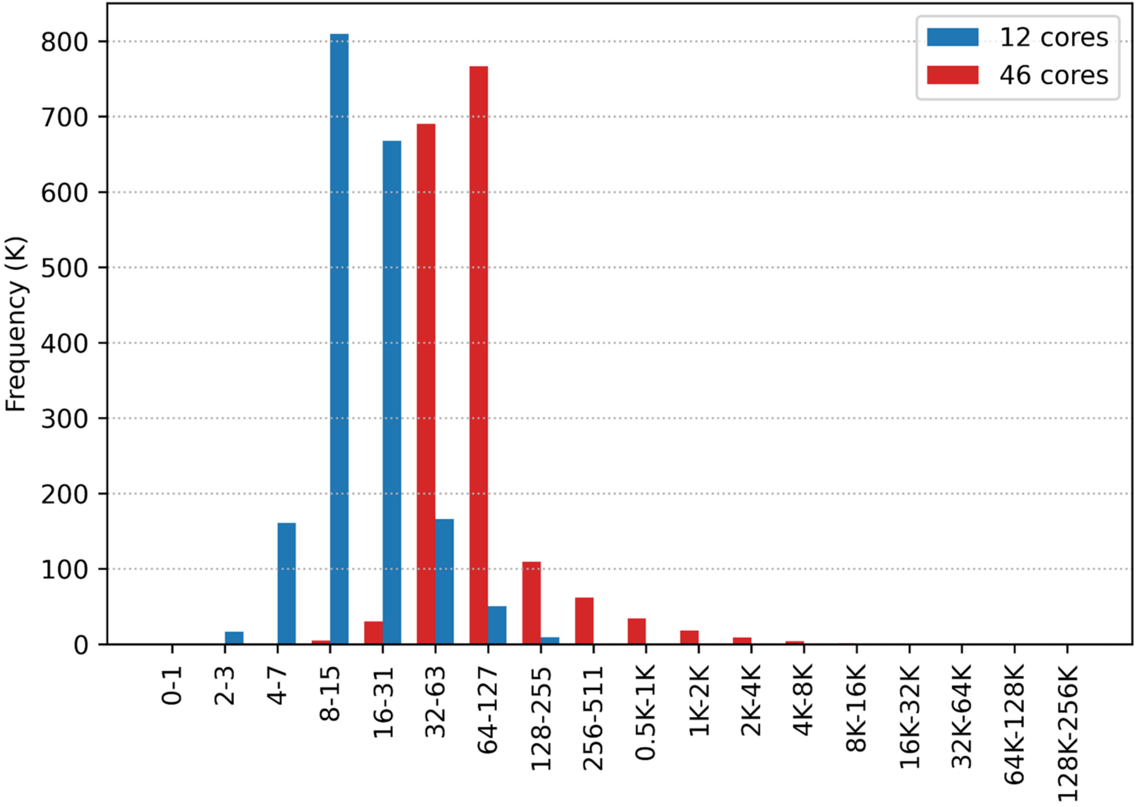
IDLING



12 cores
46 cores
Frequency (K)
800
700
600
500
400
300
200
100
0
0-1
2-3
4-7
8-15
16-31
32-63
64-127
128-255
256-511
0.5K-1K
1K-2K
2K-4K
4K-8K
8K-16K
16K-32K
32K-64K
64K-128K
128K-256K
CPU IDLING (us)

IDLING

Lower IDLING ⟷ Higher IDLING

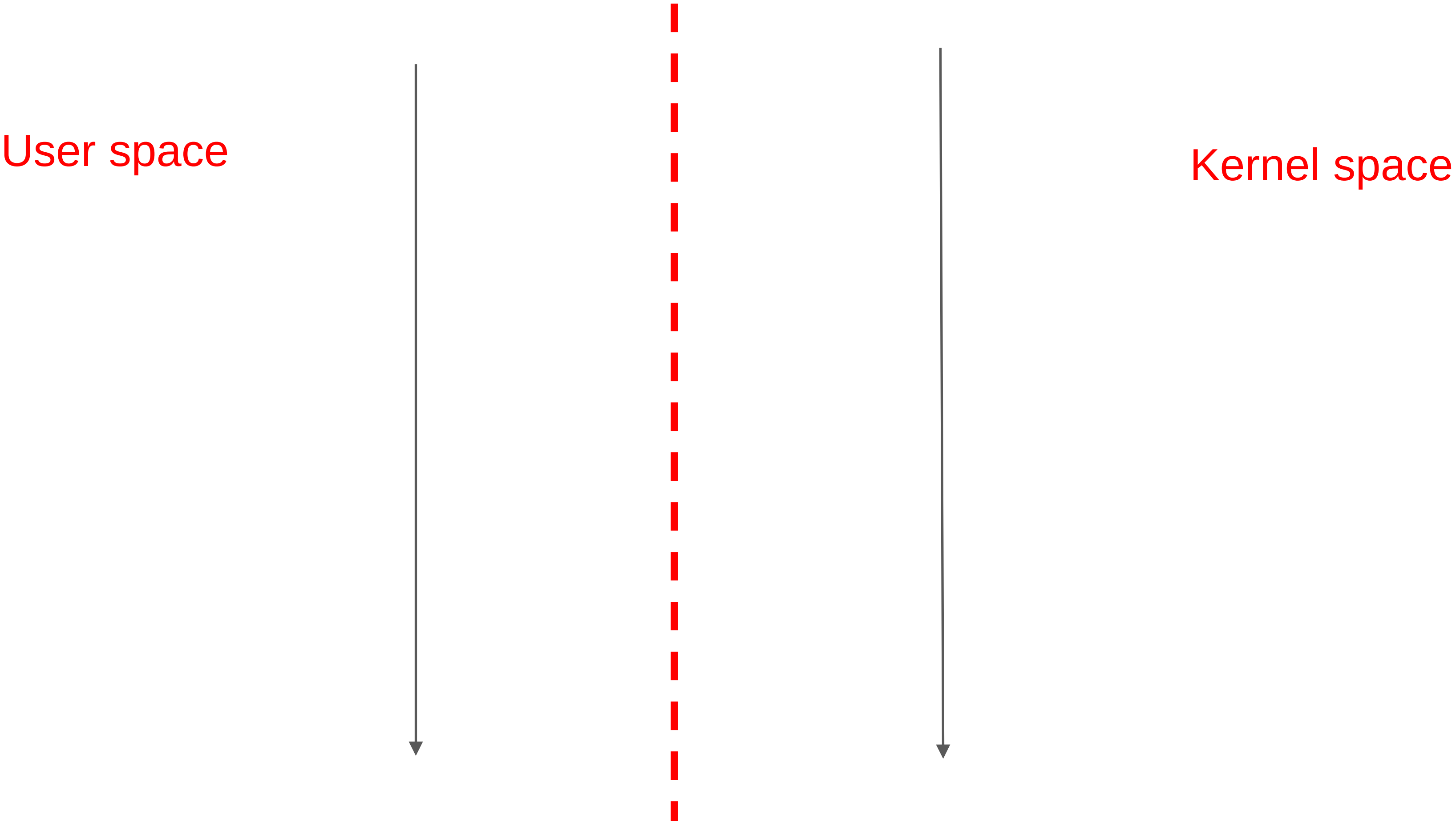


Why do we have IDLING?

User space

Kernel space

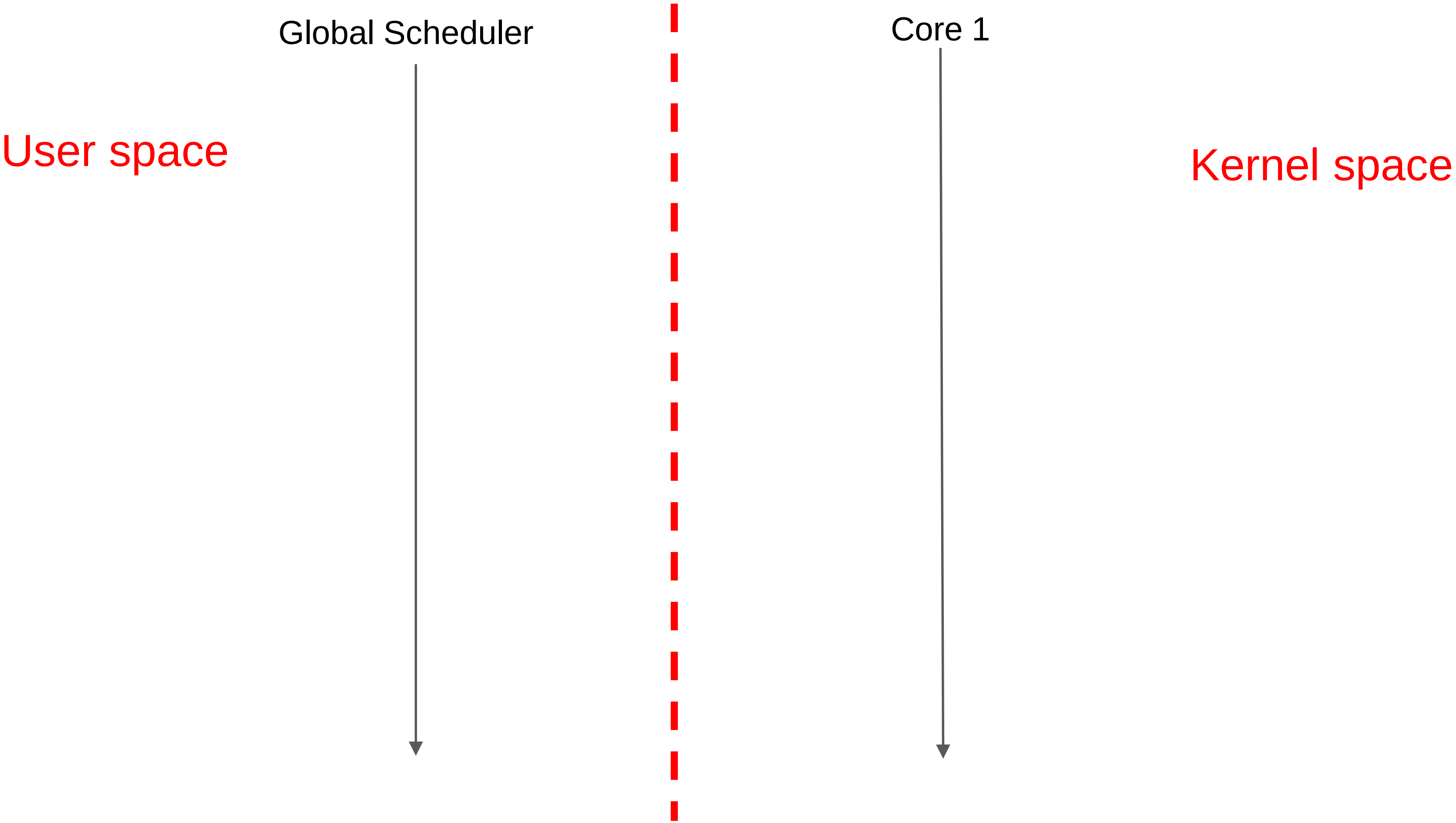
Why do we have IDLING?



User space

Kernel space

Why do we have IDLING?



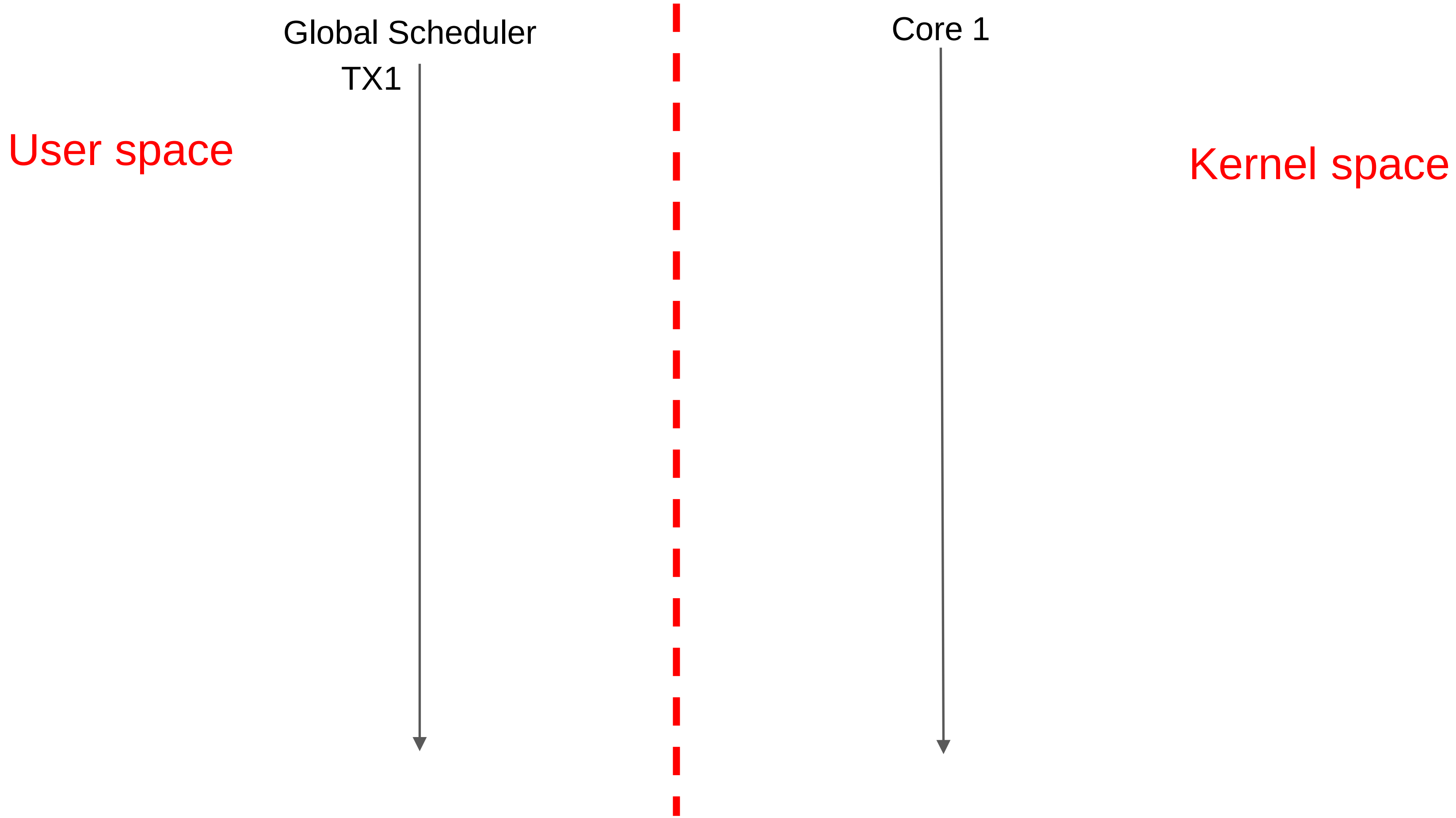
Global Scheduler

Core 1

User space

Kernel space

Why do we have IDLING?



Global Scheduler

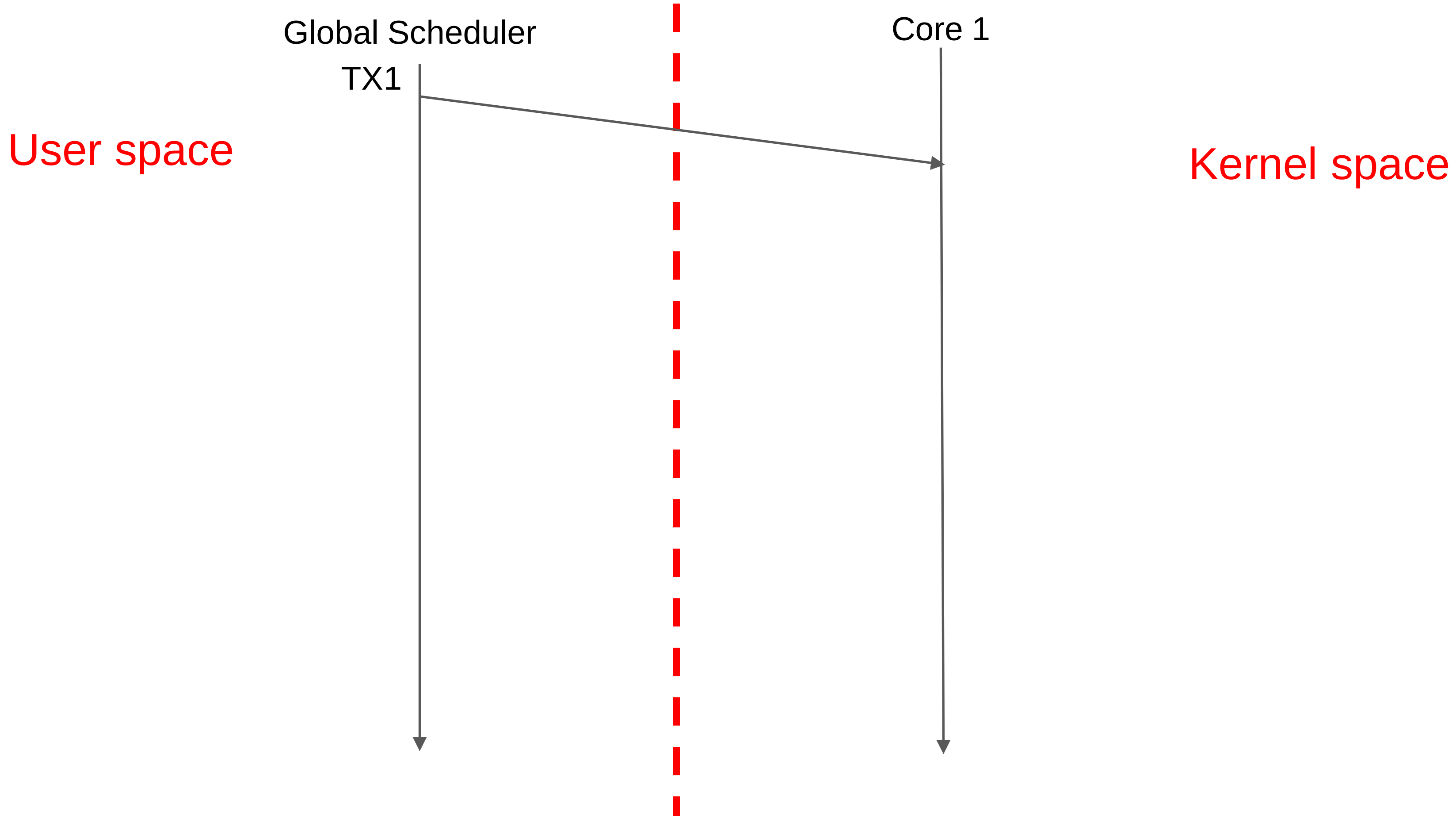
TX1

Core 1

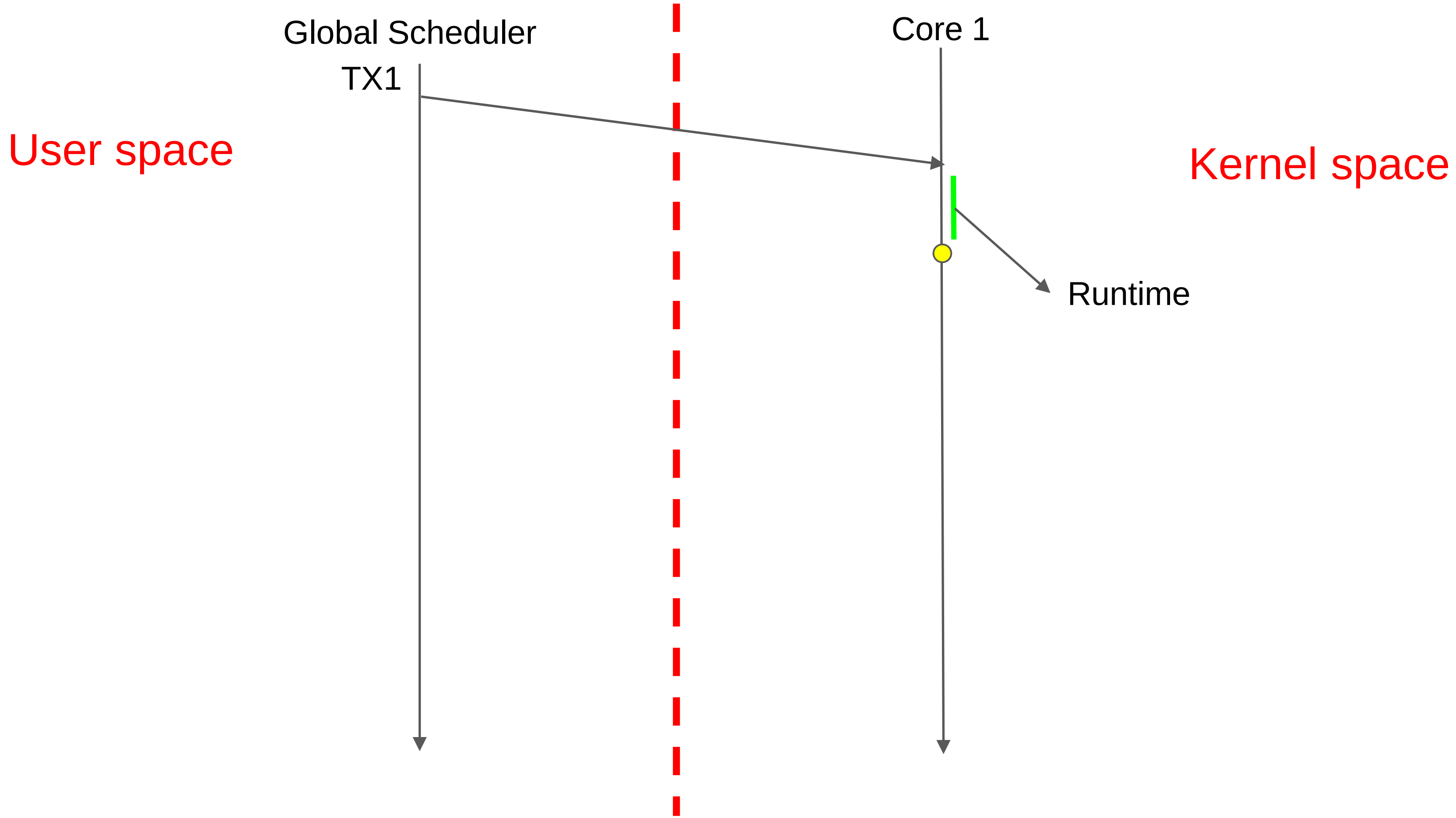
User space

Kernel space

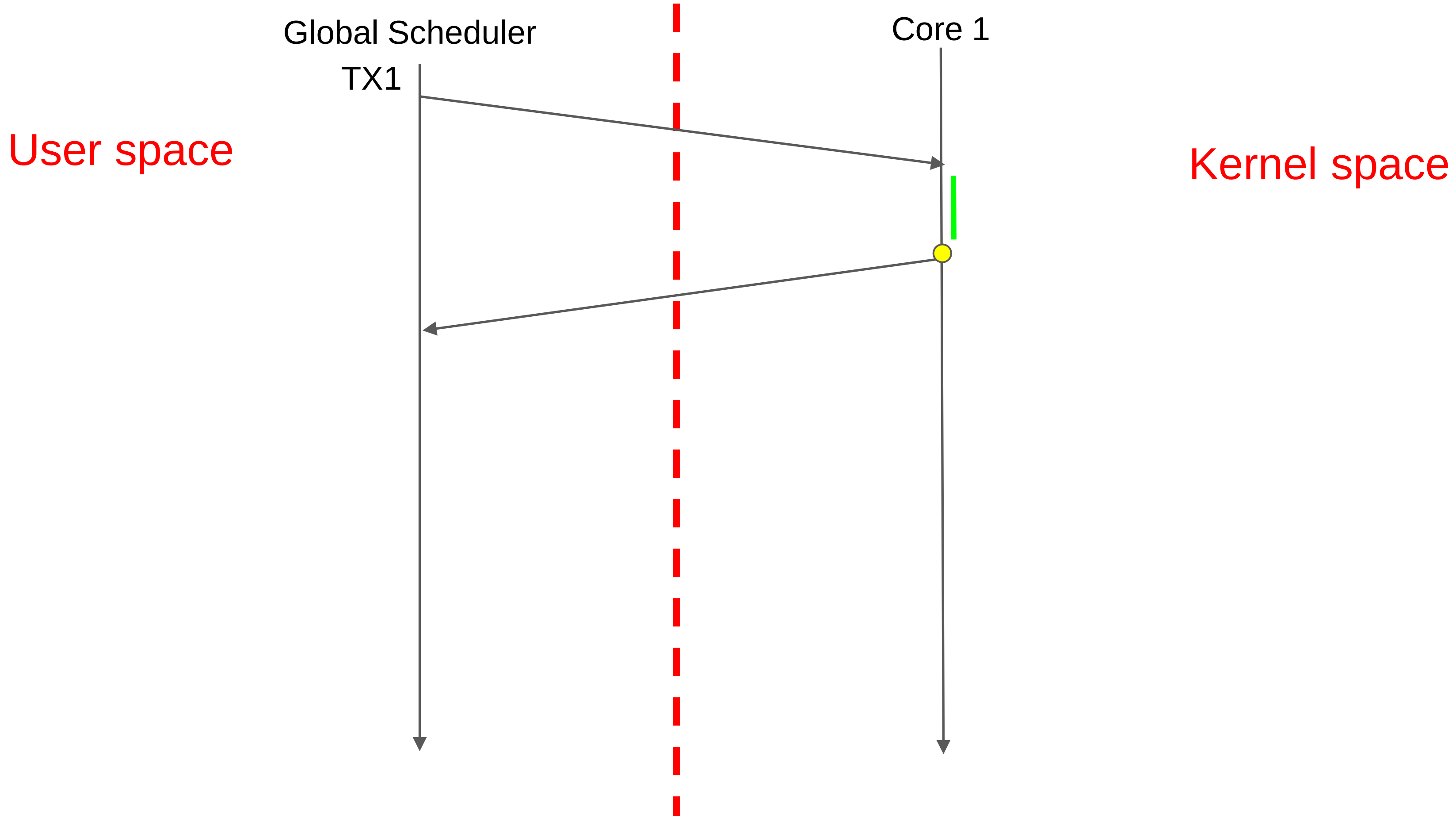
Why do we have IDLING?



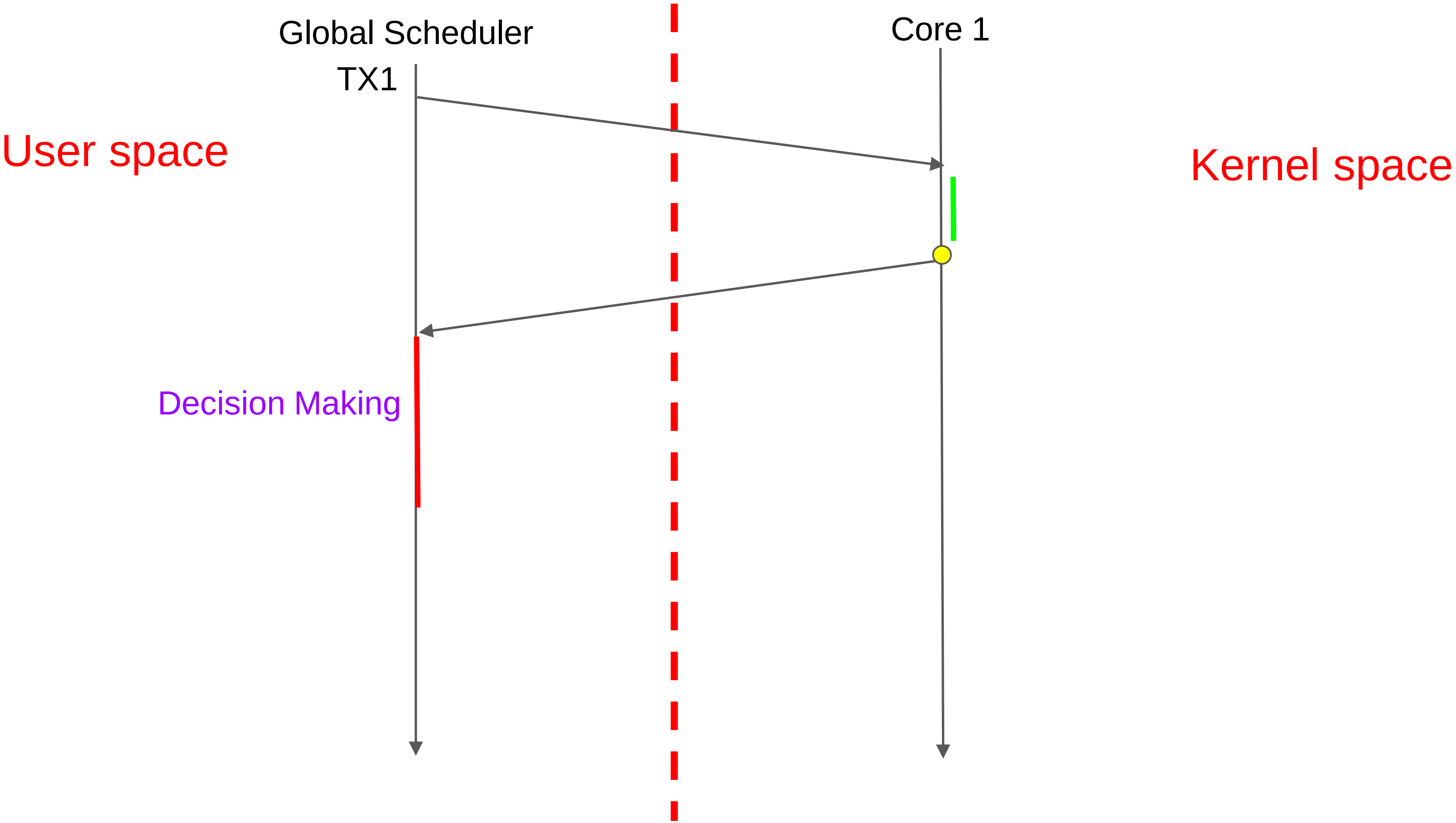
Why do we have IDLING?



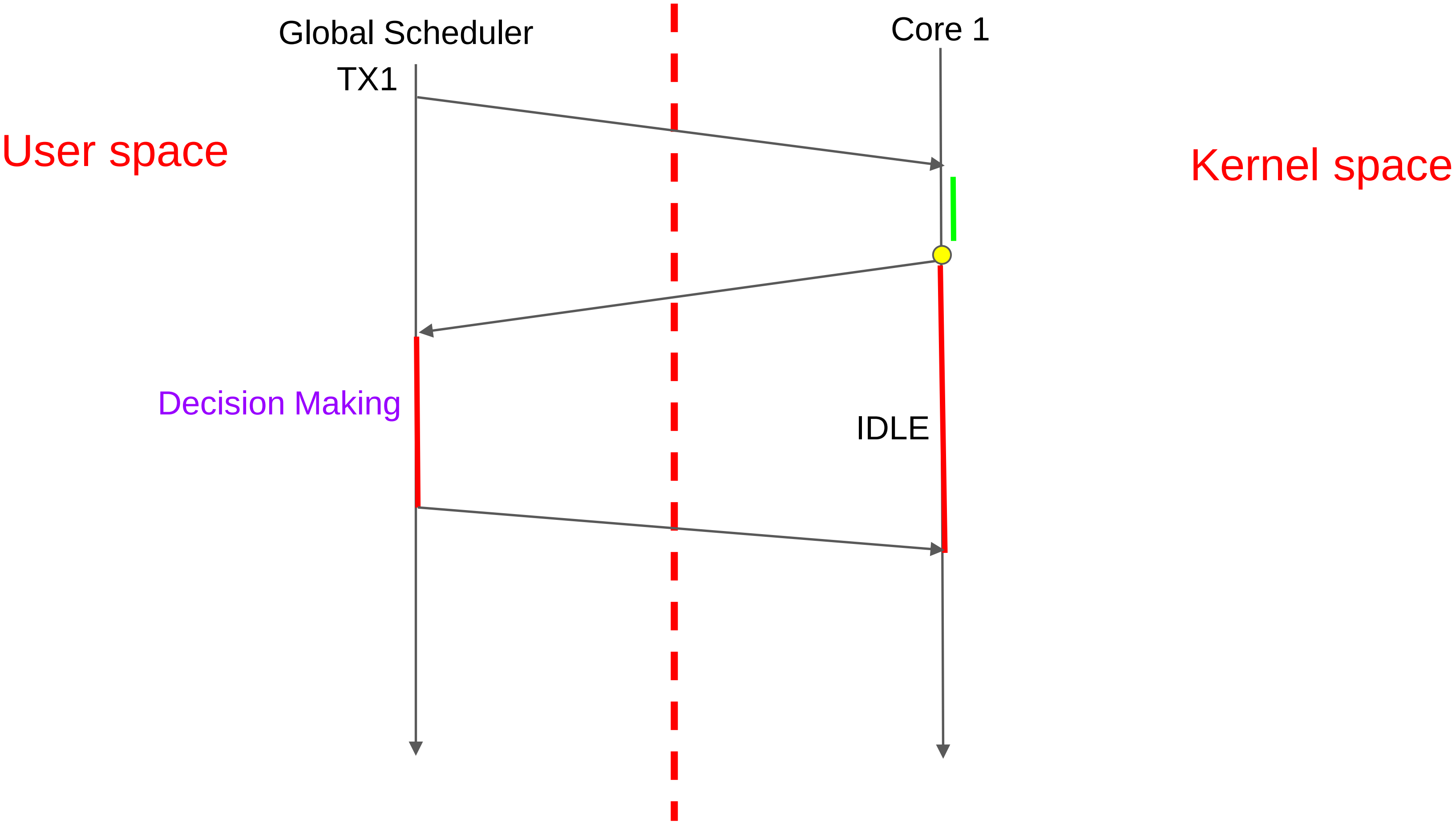
Why do we have IDLING?



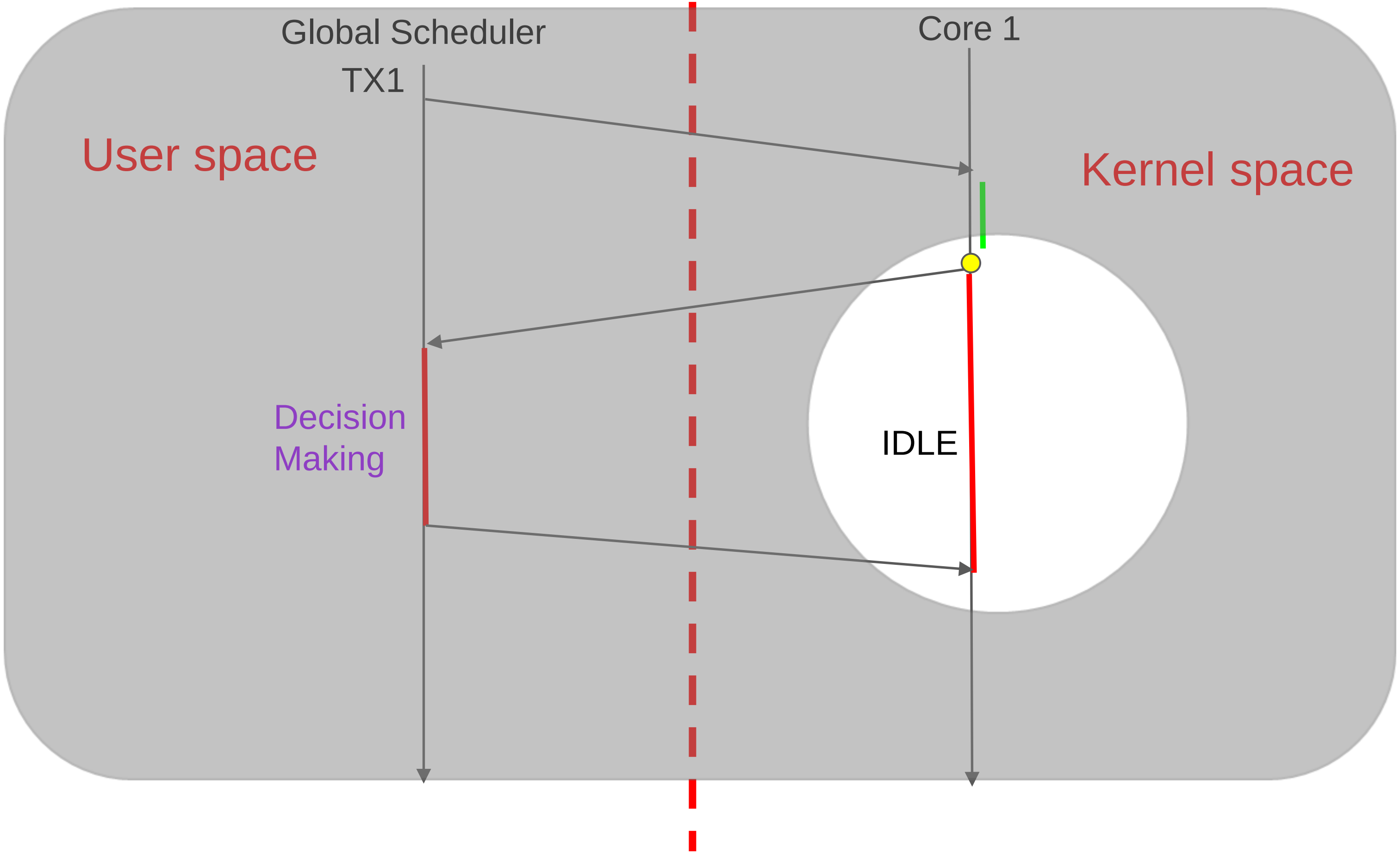
Why do we have IDLING?



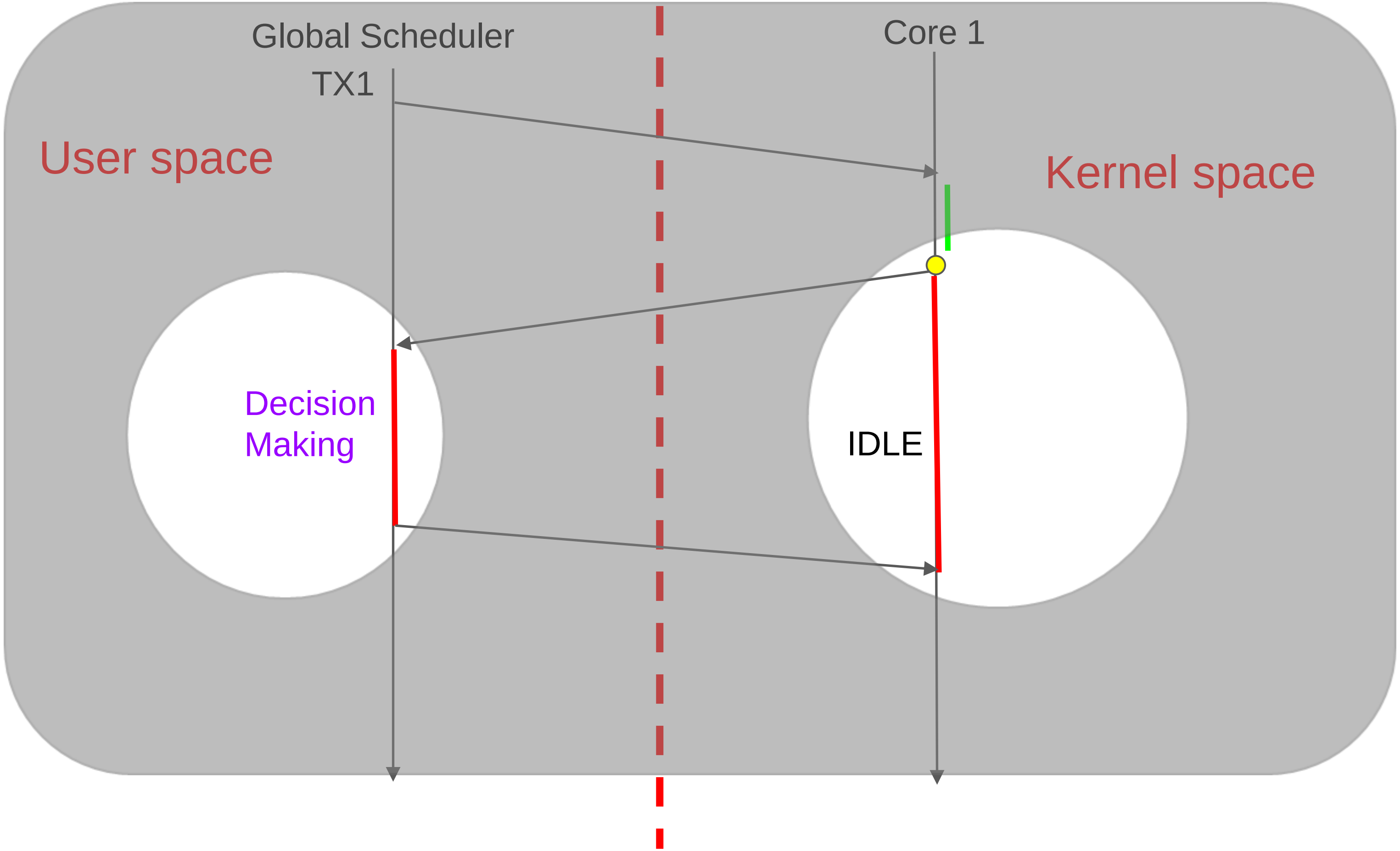
Why do we have IDLING?



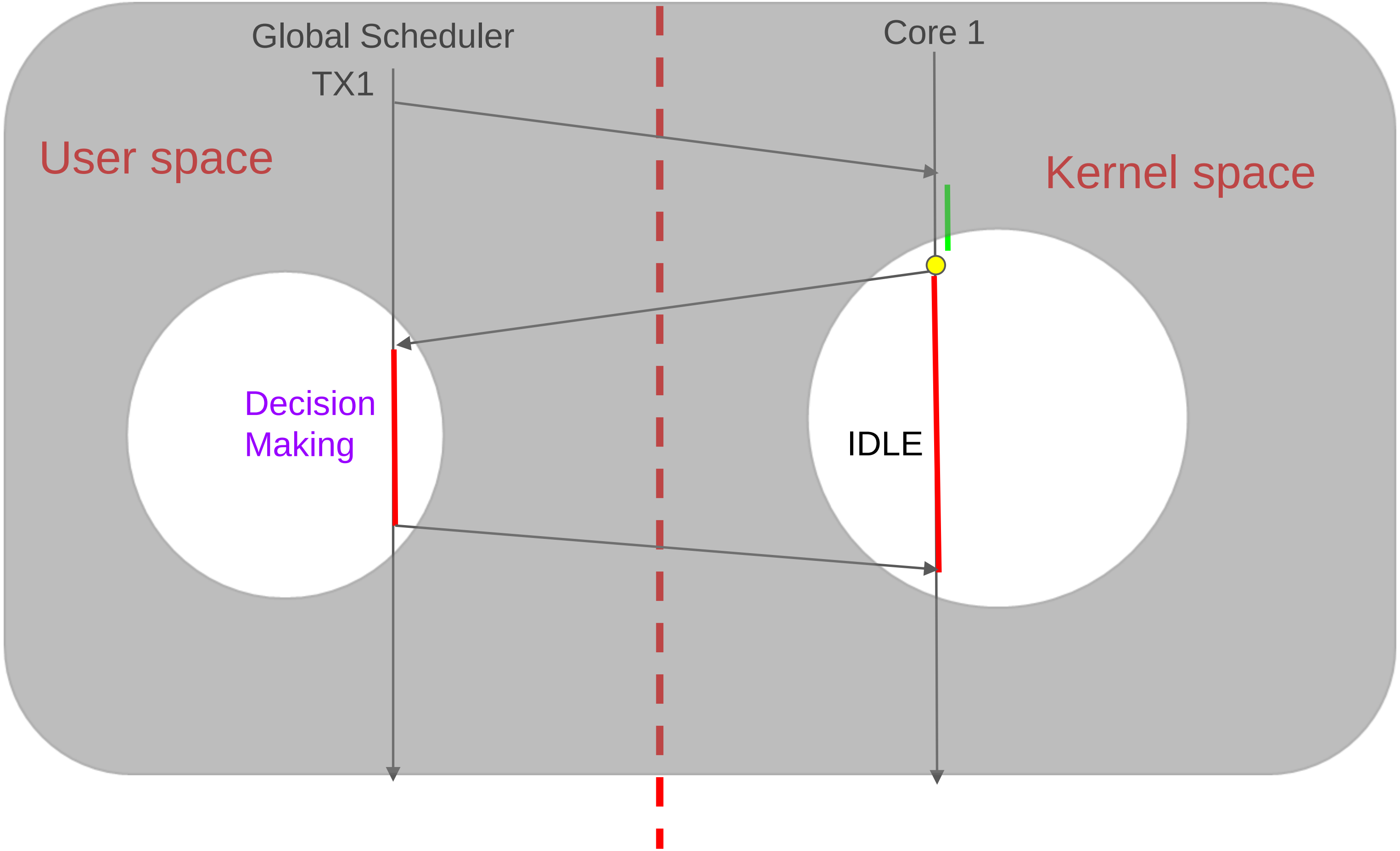
Can we do better?



Run an application in while making a decision



Oh wait! But we are in the kernel, not in userspace!



BPF allows for:

Pushing arbitrary code without kernel recompile!

Verifying code snippets!

With BPF, we can:

*Make **quicker decisions** in the kernel*

With BPF, we can:

*Make **quicker decisions** in the kernel*

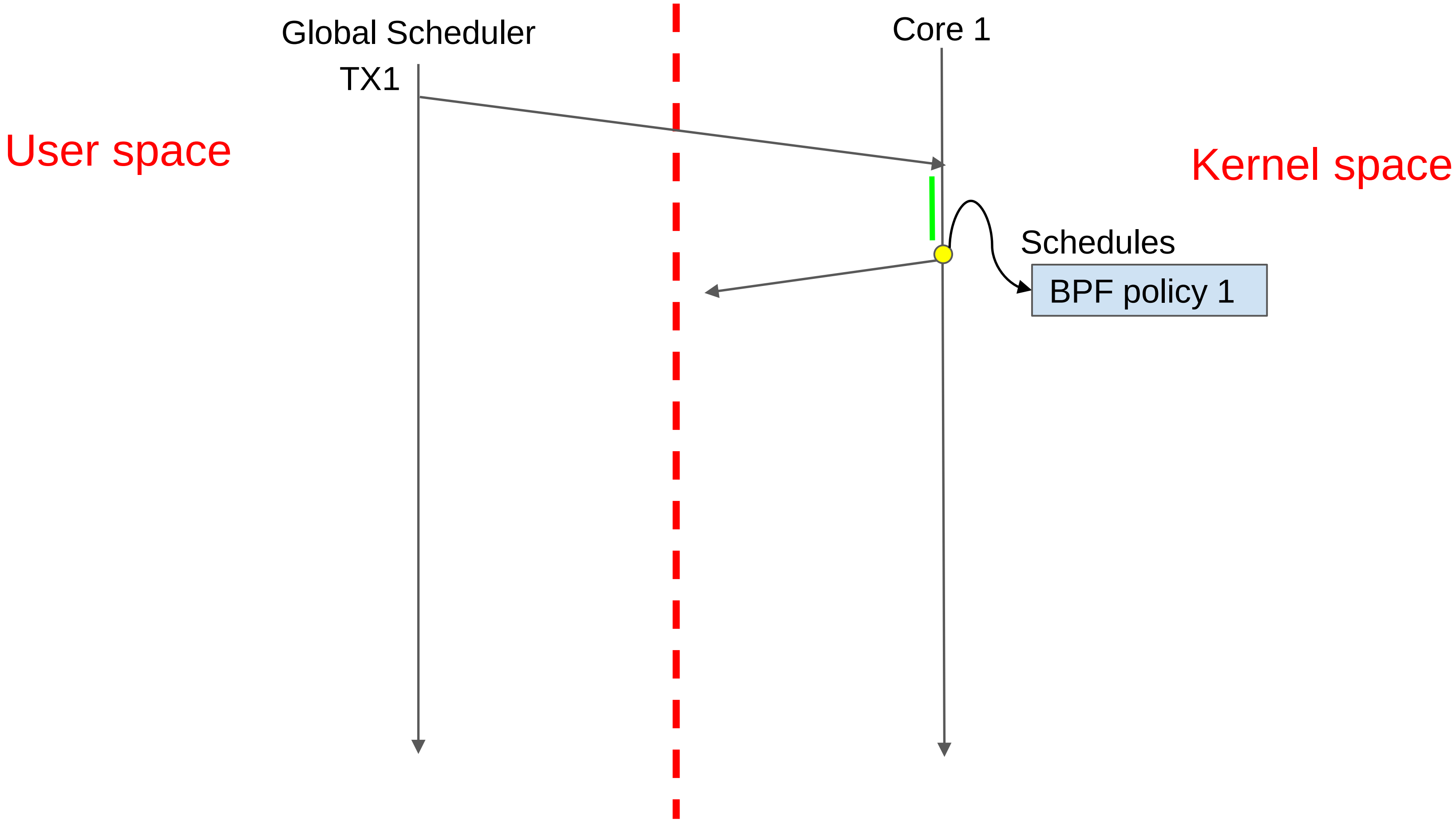
*Have better **insights** in the kernel*

With BPF we can:

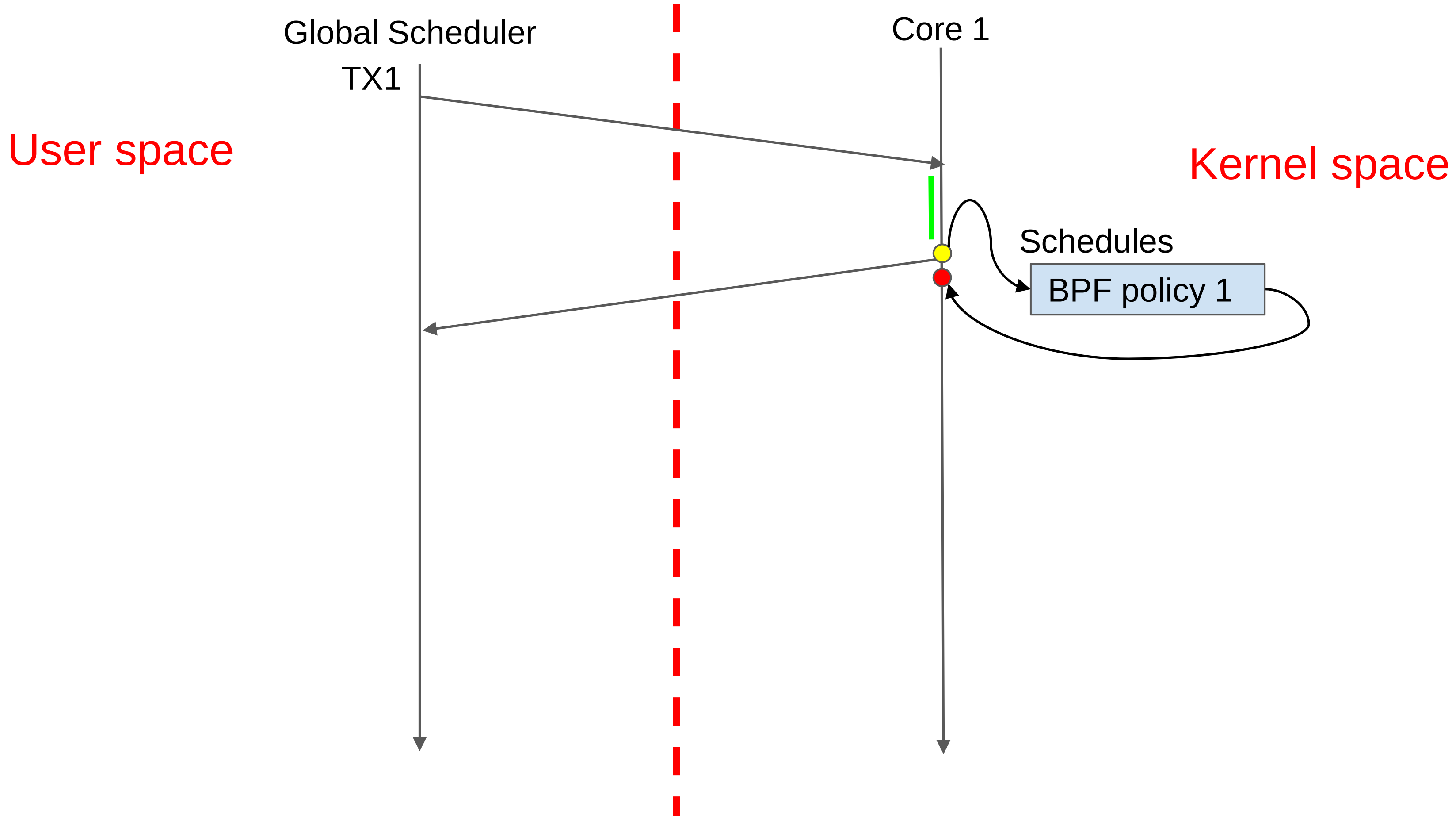
*Make **quicker decisions** in the kernel*

*Have better **insights** in the kernel*

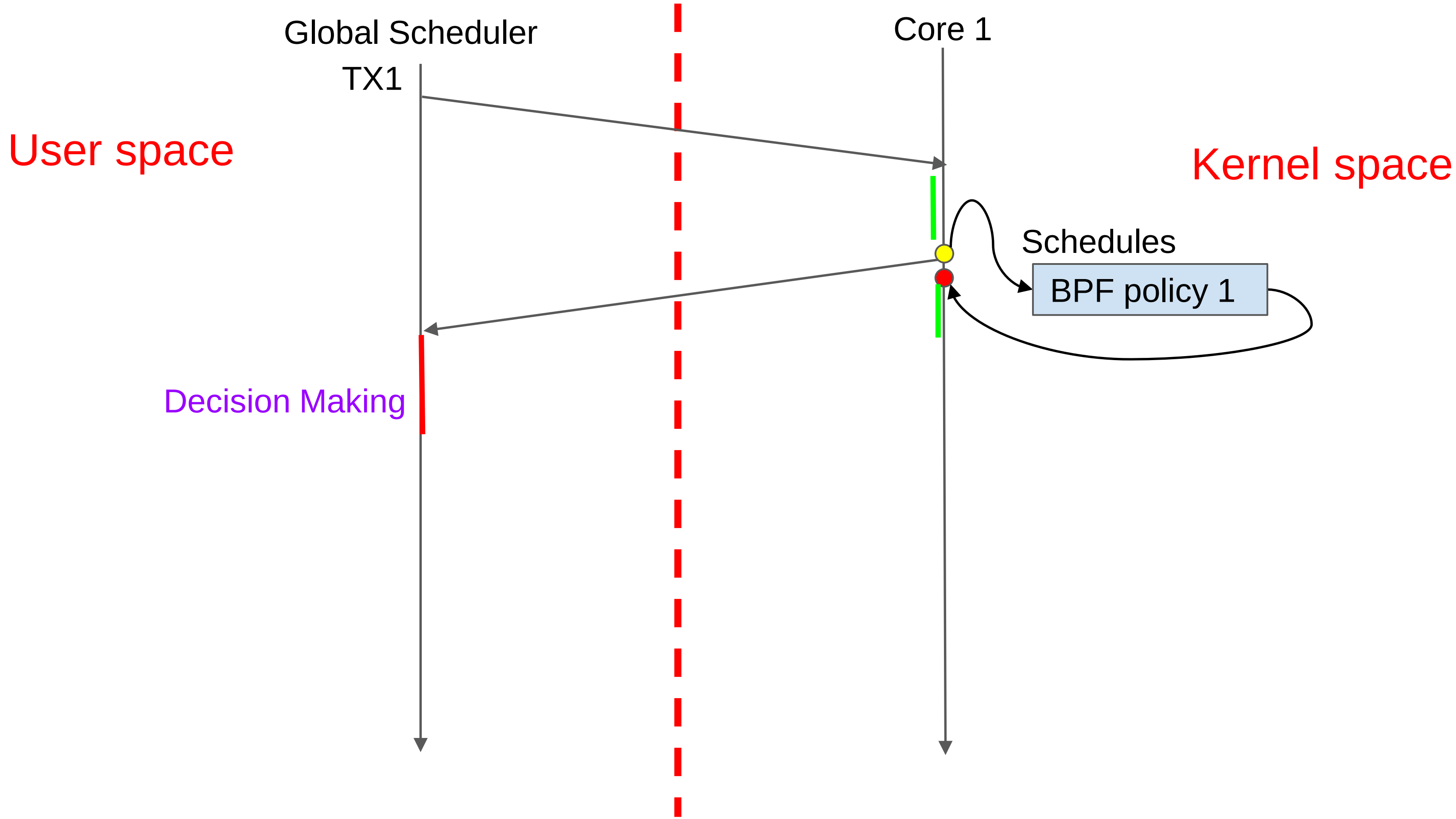
With BPF, we can do better!



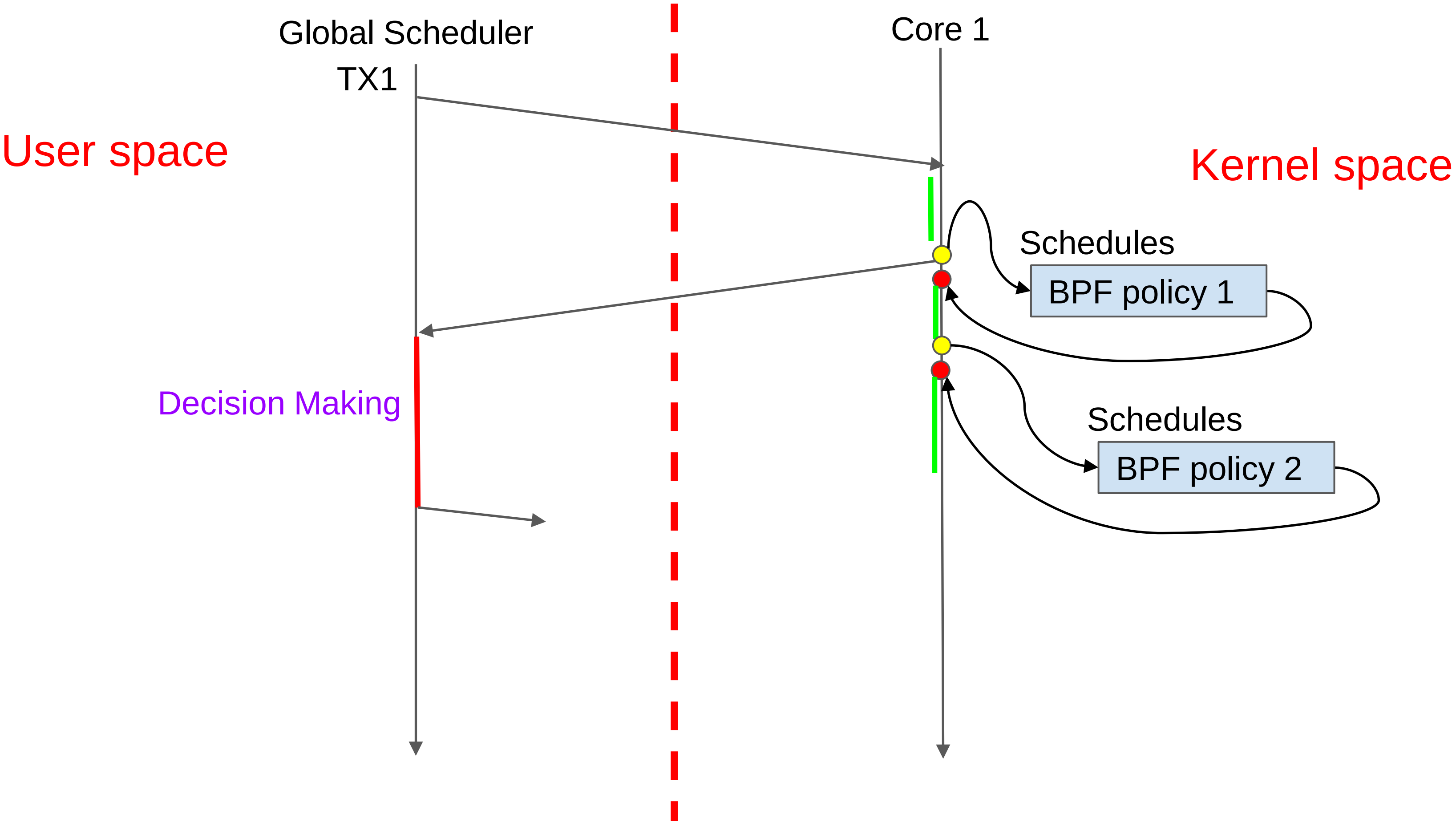
With BPF, we can do better!



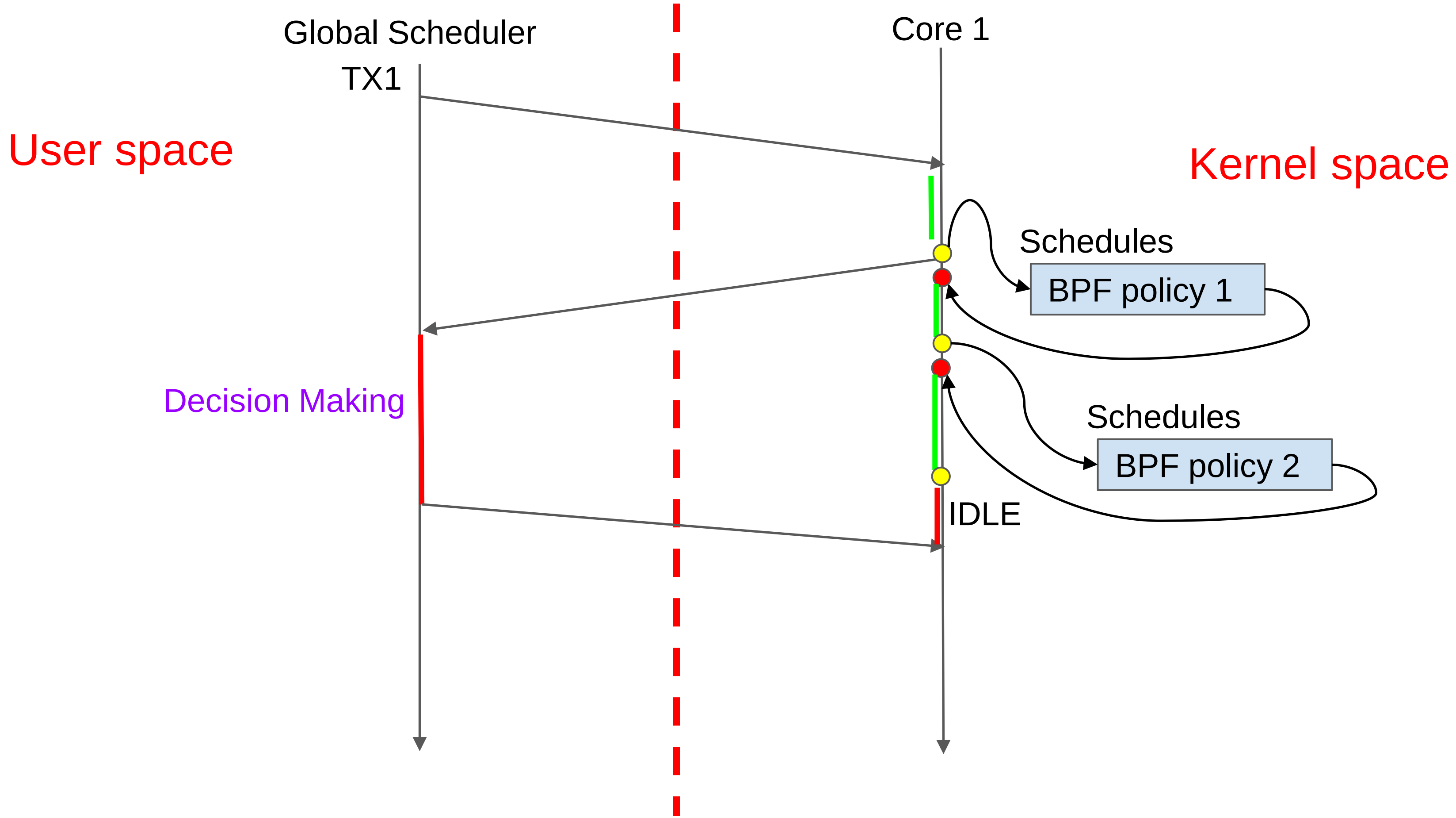
With BPF, we can do better!



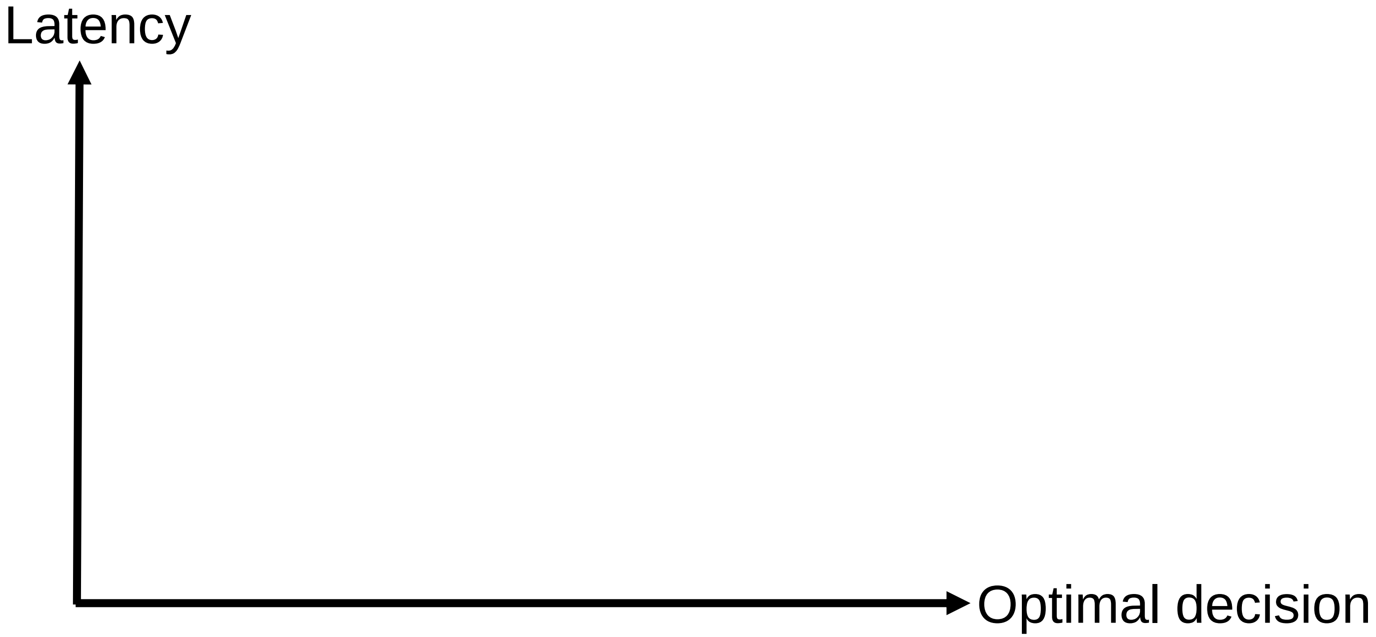
With BPF, we can do better!



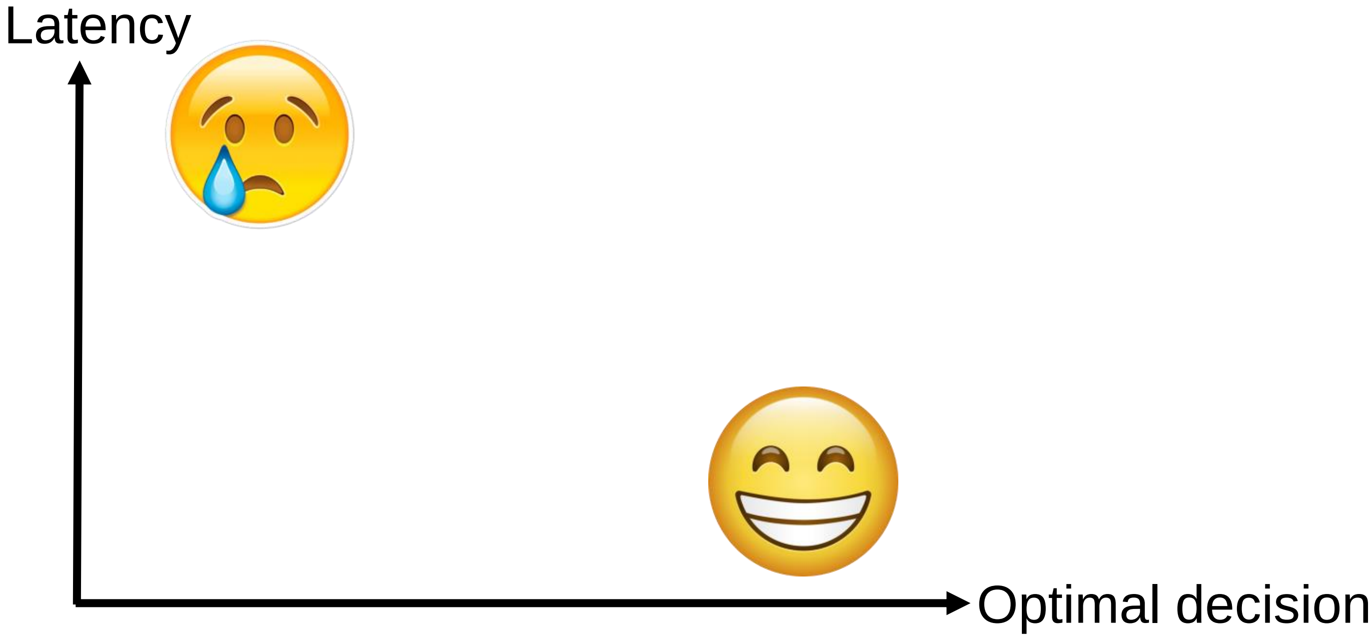
With BPF, we can do better!



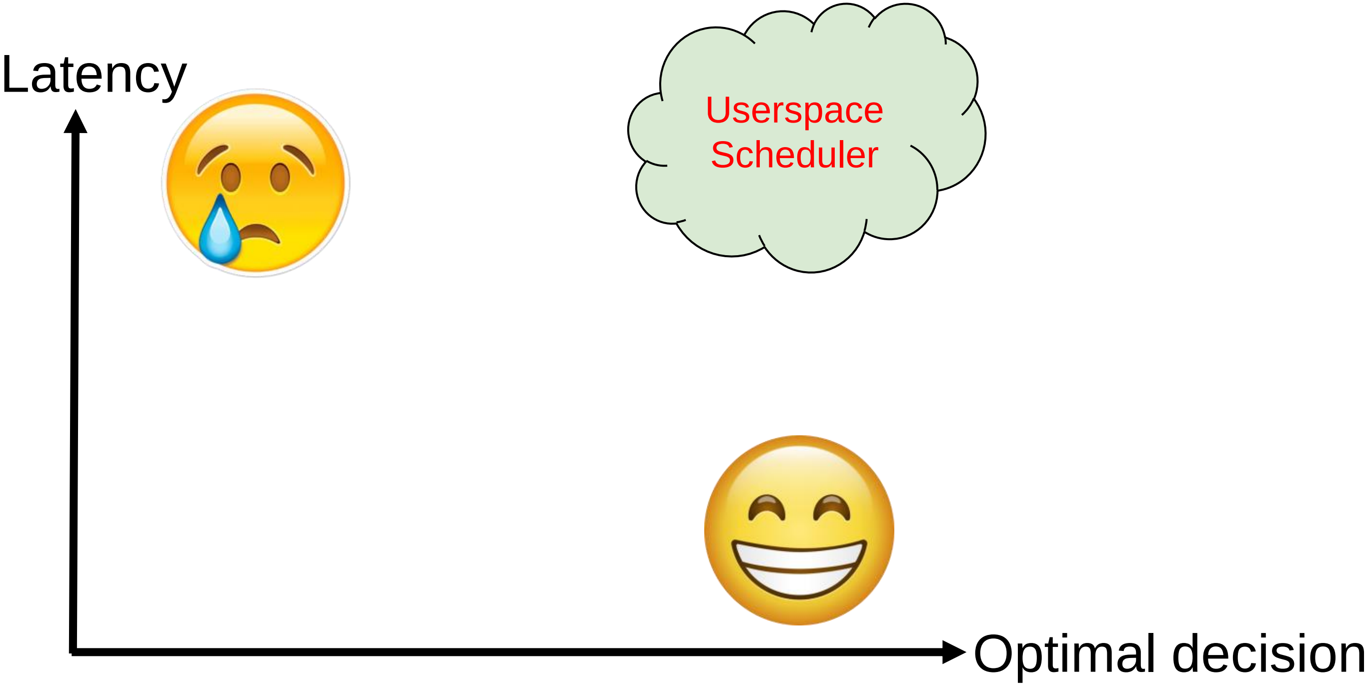
What is the right trade-off between centralized CPU scheduler and BPF?



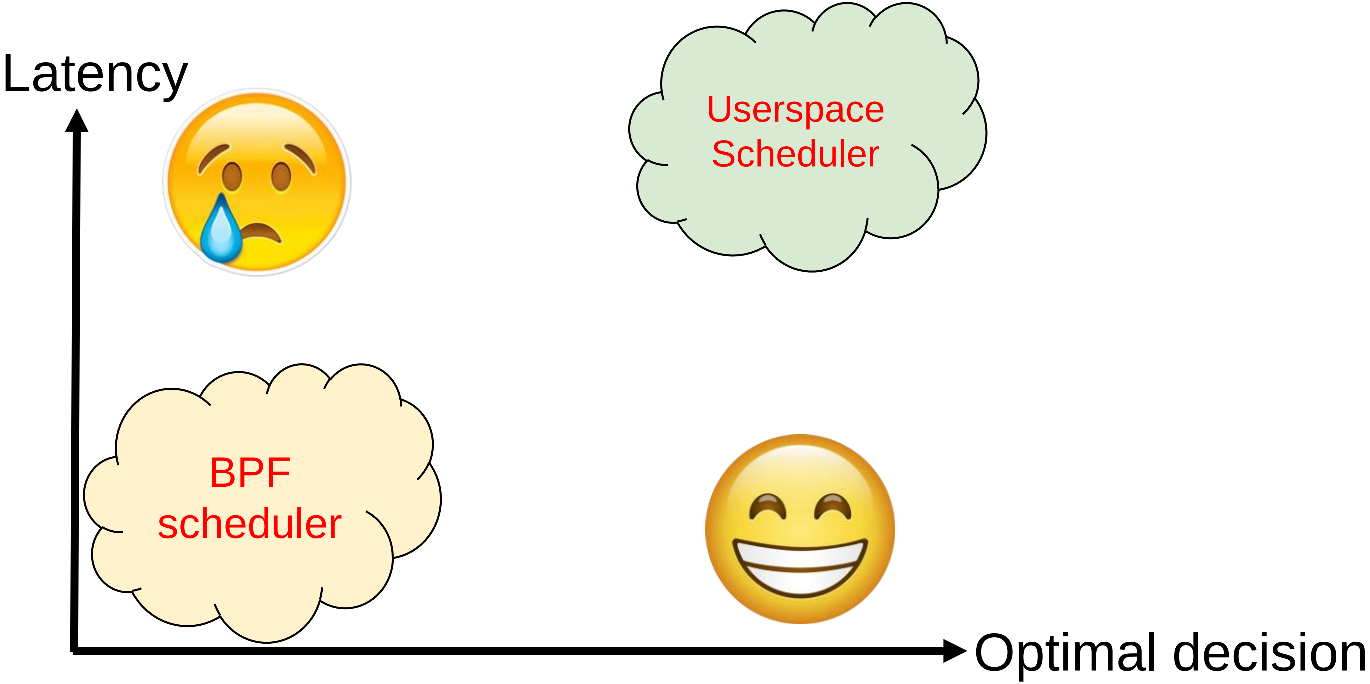
What is the right trade-off between centralized CPU scheduler and BPF?



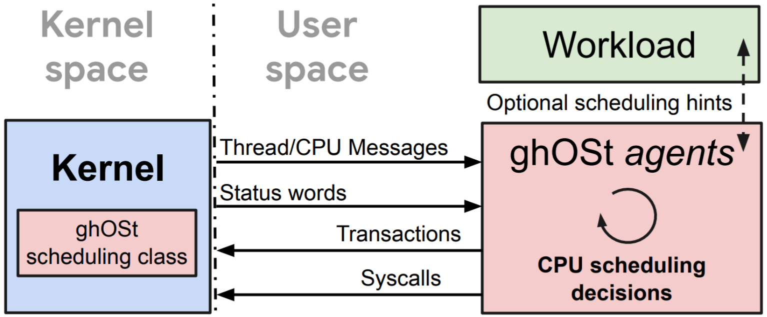
What is the right trade-off between centralized CPU scheduler and BPF?



What is the right trade-off between centralized CPU scheduler and BPF?

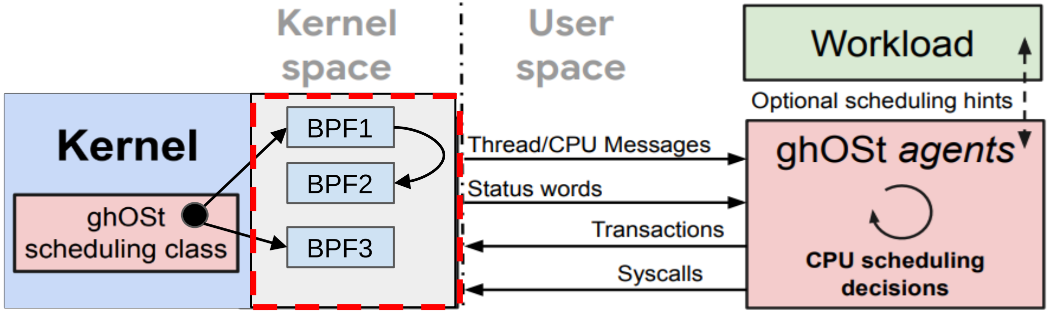


Let’s extend ghost!

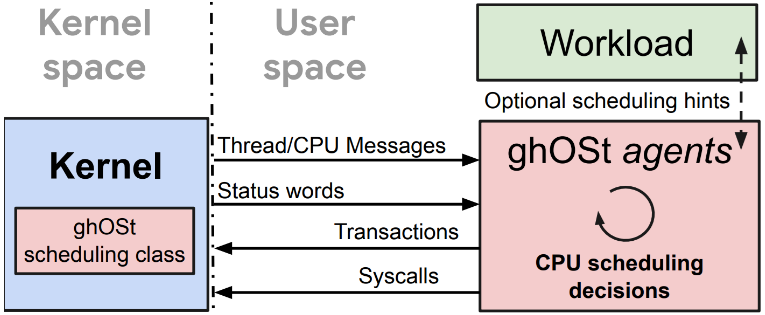


Legacy ghOSt

Let’s extend ghost!



ghOSt BPF



Legacy ghOSt

Let’s implement a policy in BPF

How to implement CFS in BPF?

Existing BPF data structures

How to implement CFS in BPF?

BPF only support hash-map && array

Existing BPF data structures

How to implement CFS in BPF?

BPF only support hash-map && array

But we need red-black tree!

Conclusion

How much expressivity eBPF would have for scheduling policies?

Conclusion

How much expressivity eBPF would have for scheduling policies?

Do we need to extend the eBPF ecosystem to be more suitable for policy implementation?

Conclusion

How much expressivity eBPF would have for scheduling policies?

Do we need to extend the eBPF ecosystem to be more suitable for policy implementation?

How to address the trade-off between faster reaction at BPF and optimal decision at the userspace?