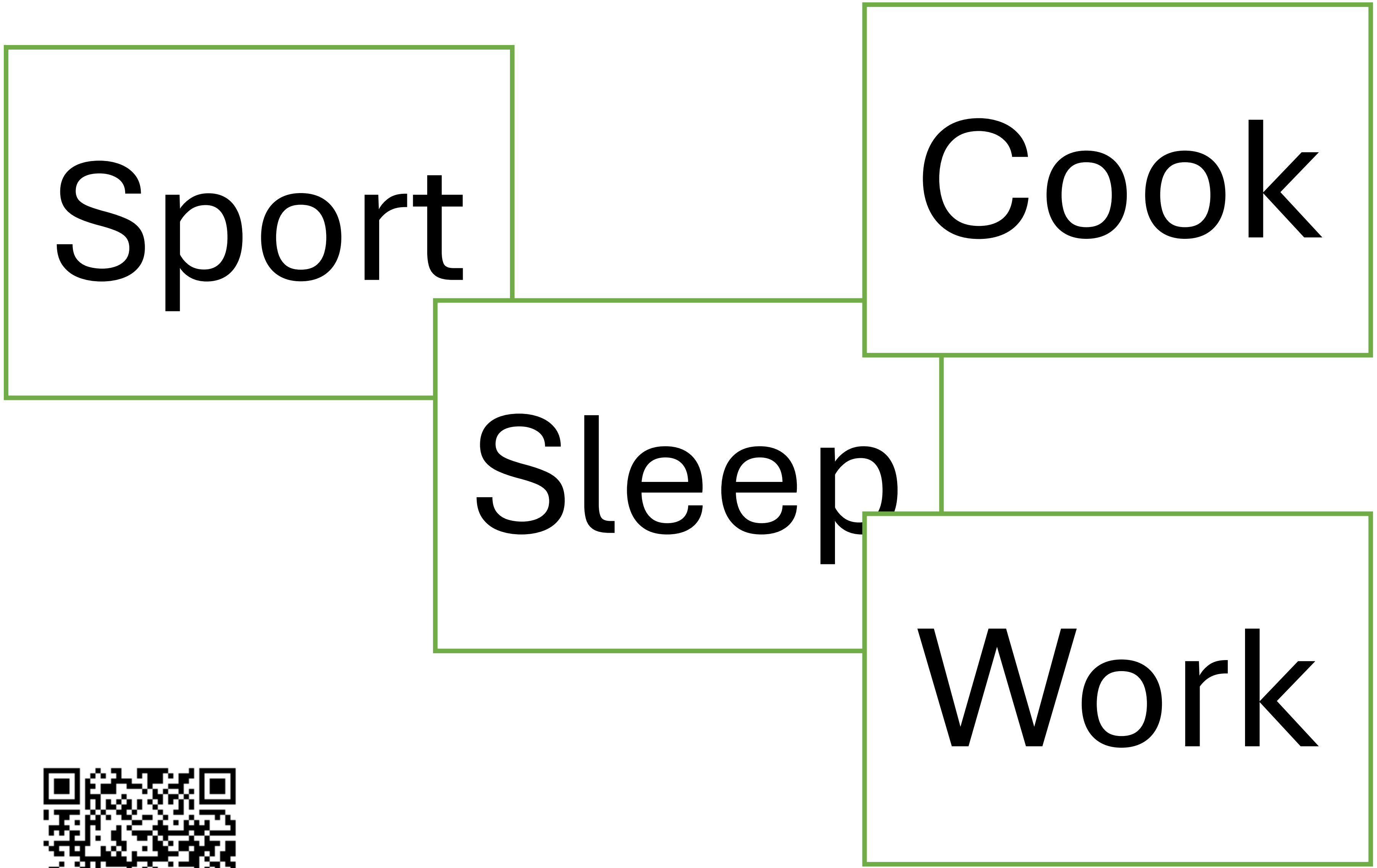
Sound of Scheduling

Writing Linux Schedulers in Java

Johannes Bechberger
David Kiefer

timetableworld.com



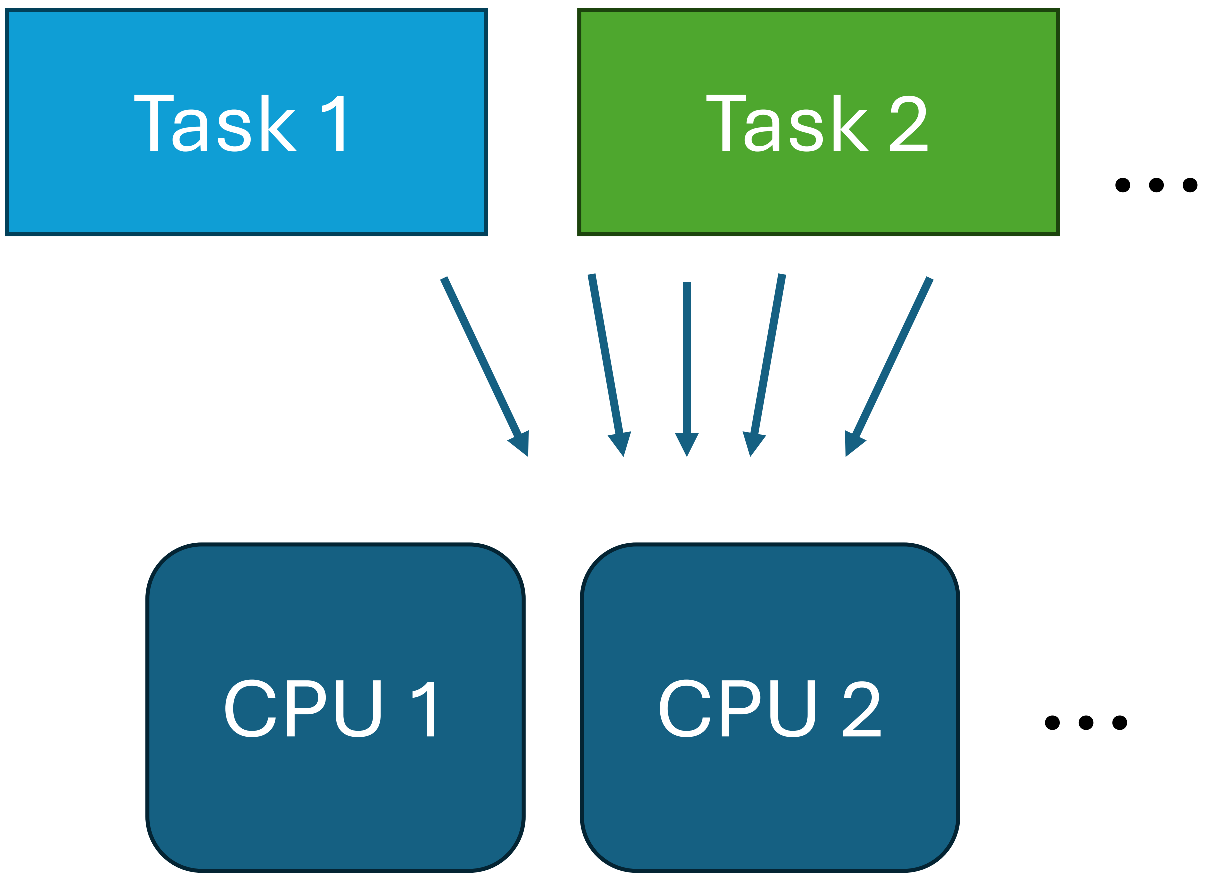
Sport

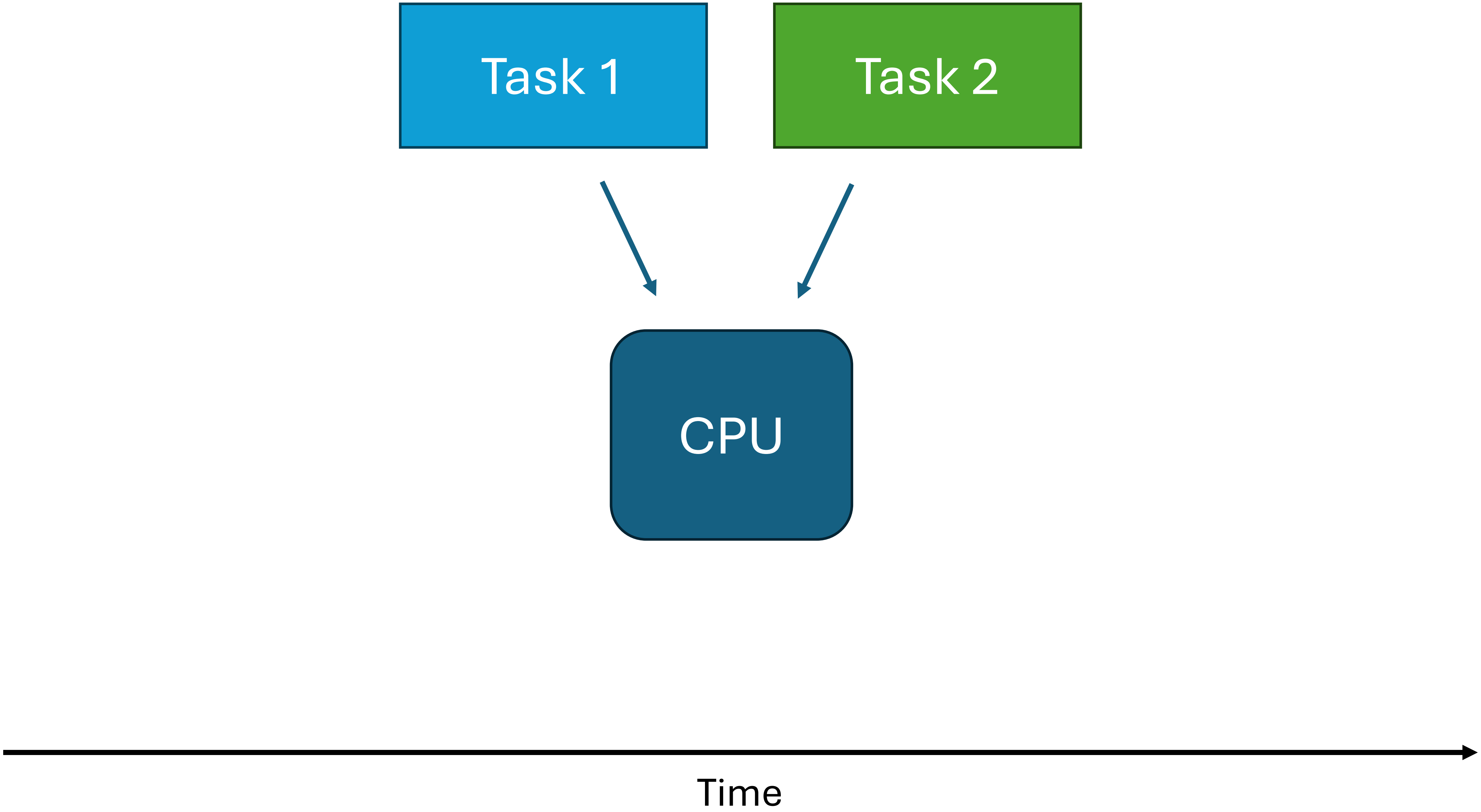
Sleep

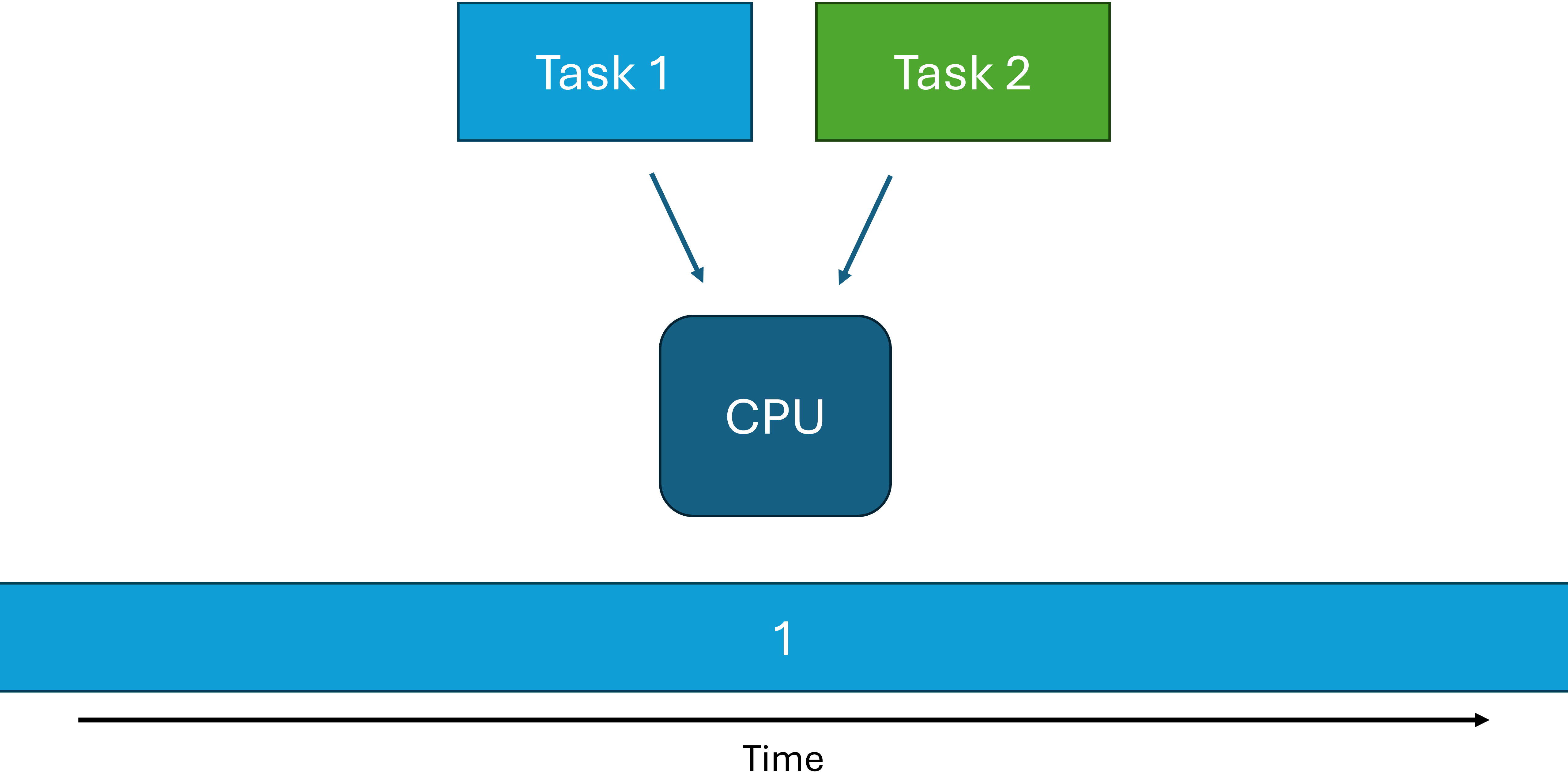
Cook

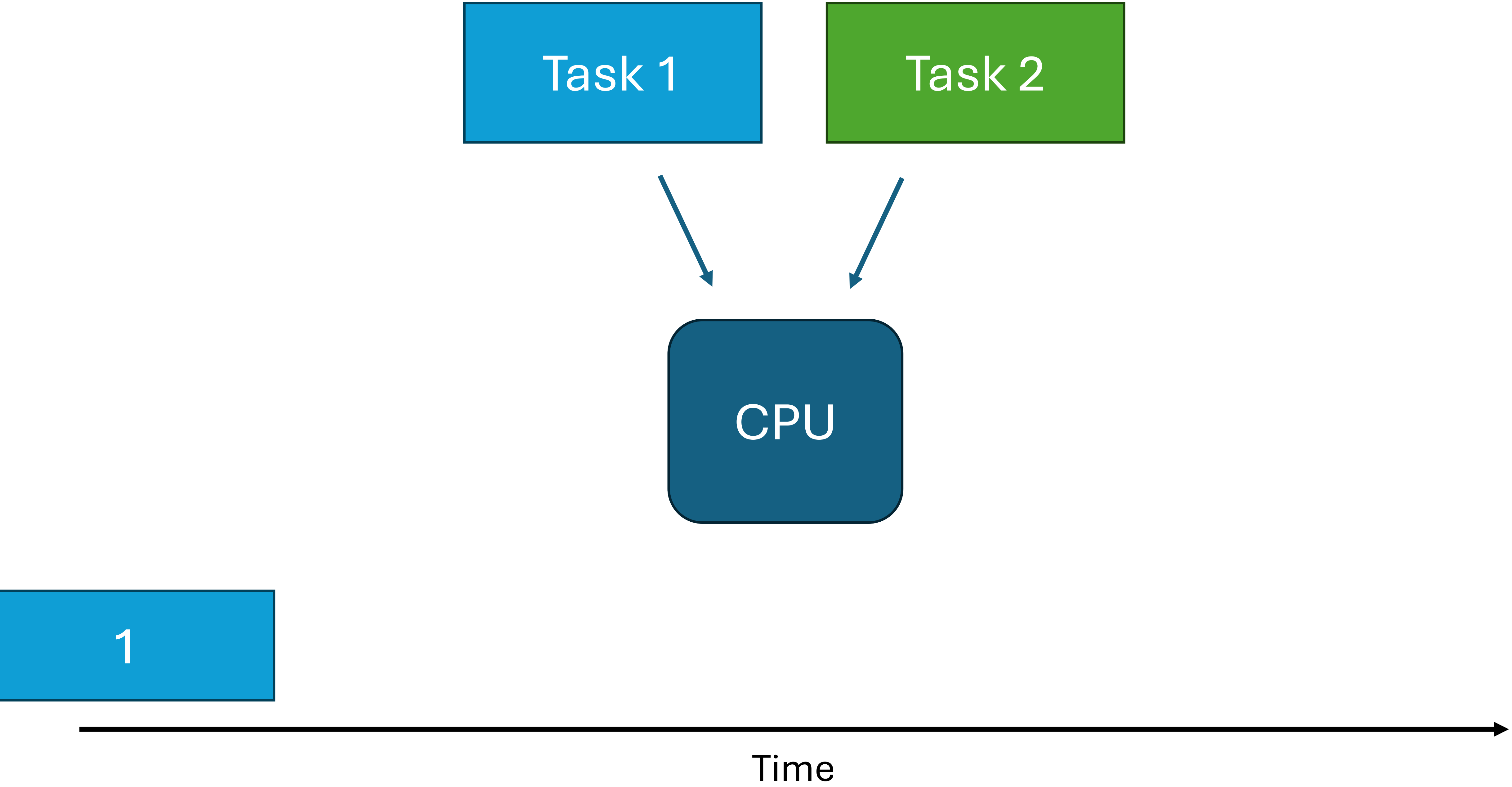
Work

https://mostlynerdless.de/blog/2013/12/08/real-life-practice-in-lottery-scheduling/


Task 1
Task 2
…
CPU 1
CPU 2
…


Task 1
Task 2
CPU
Time


Task 1
Task 2
CPU
1
Time



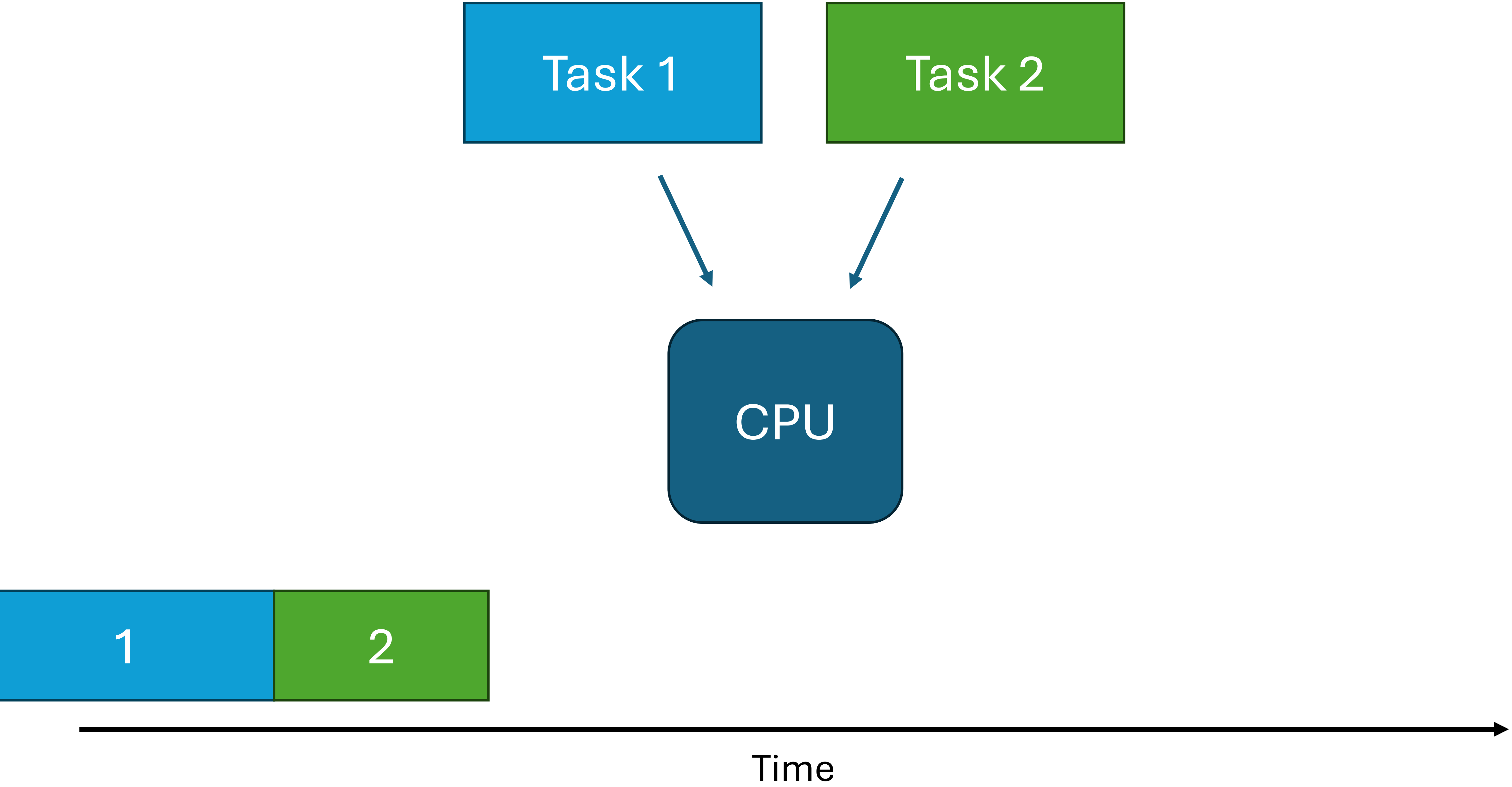
Task 1

Task 2

CPU

1

Time



Task 1

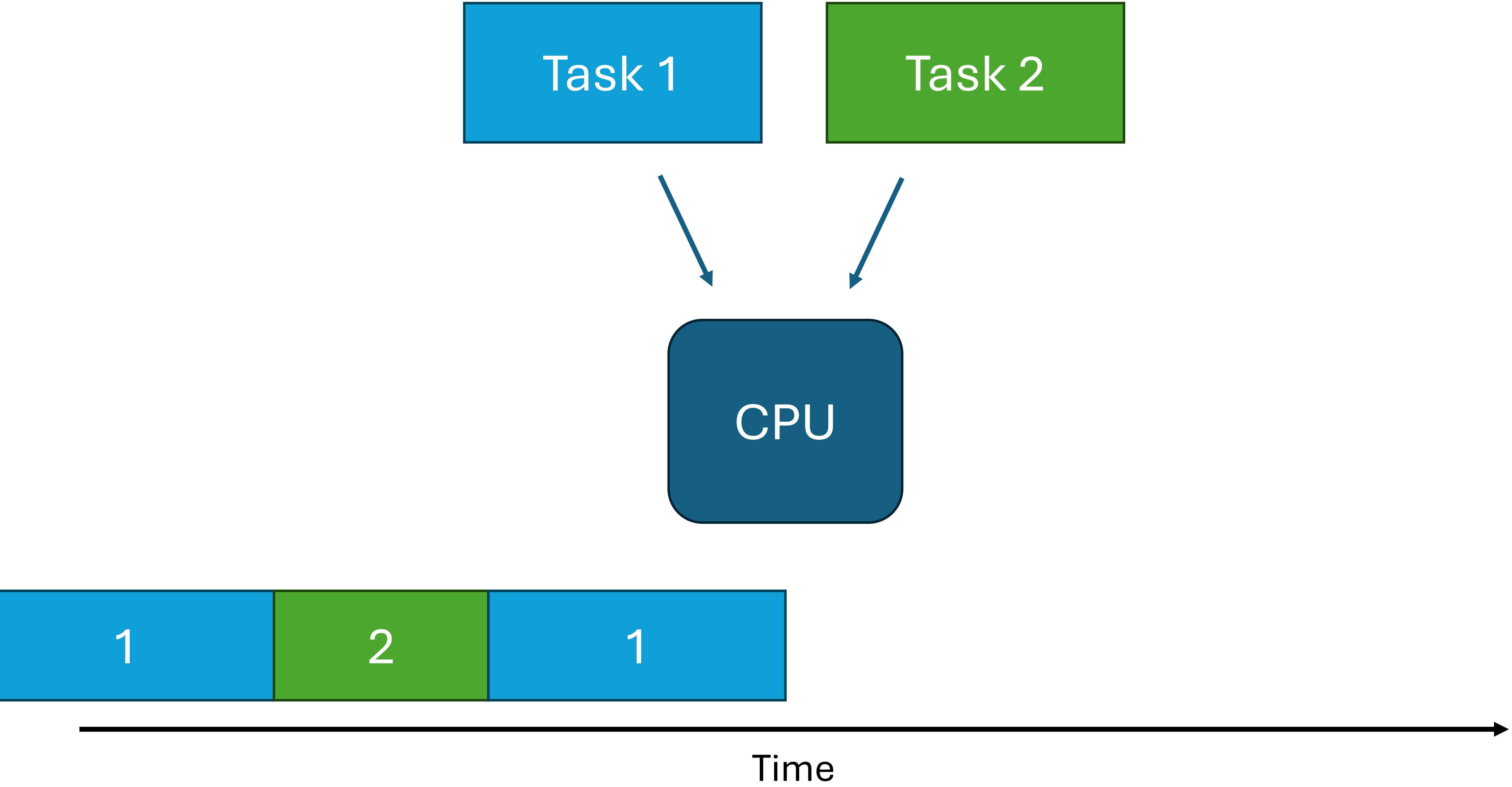
Task 2

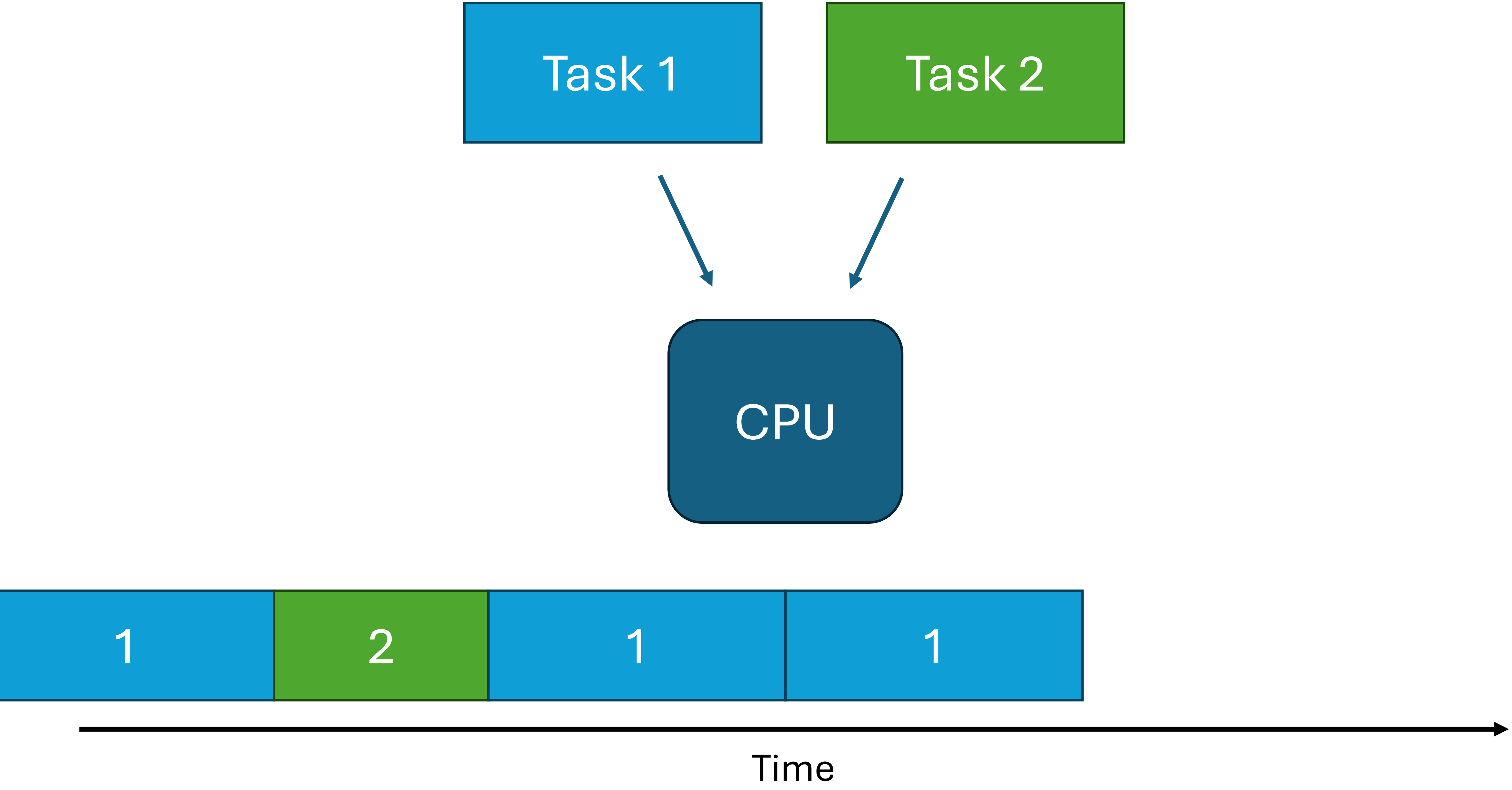
CPU

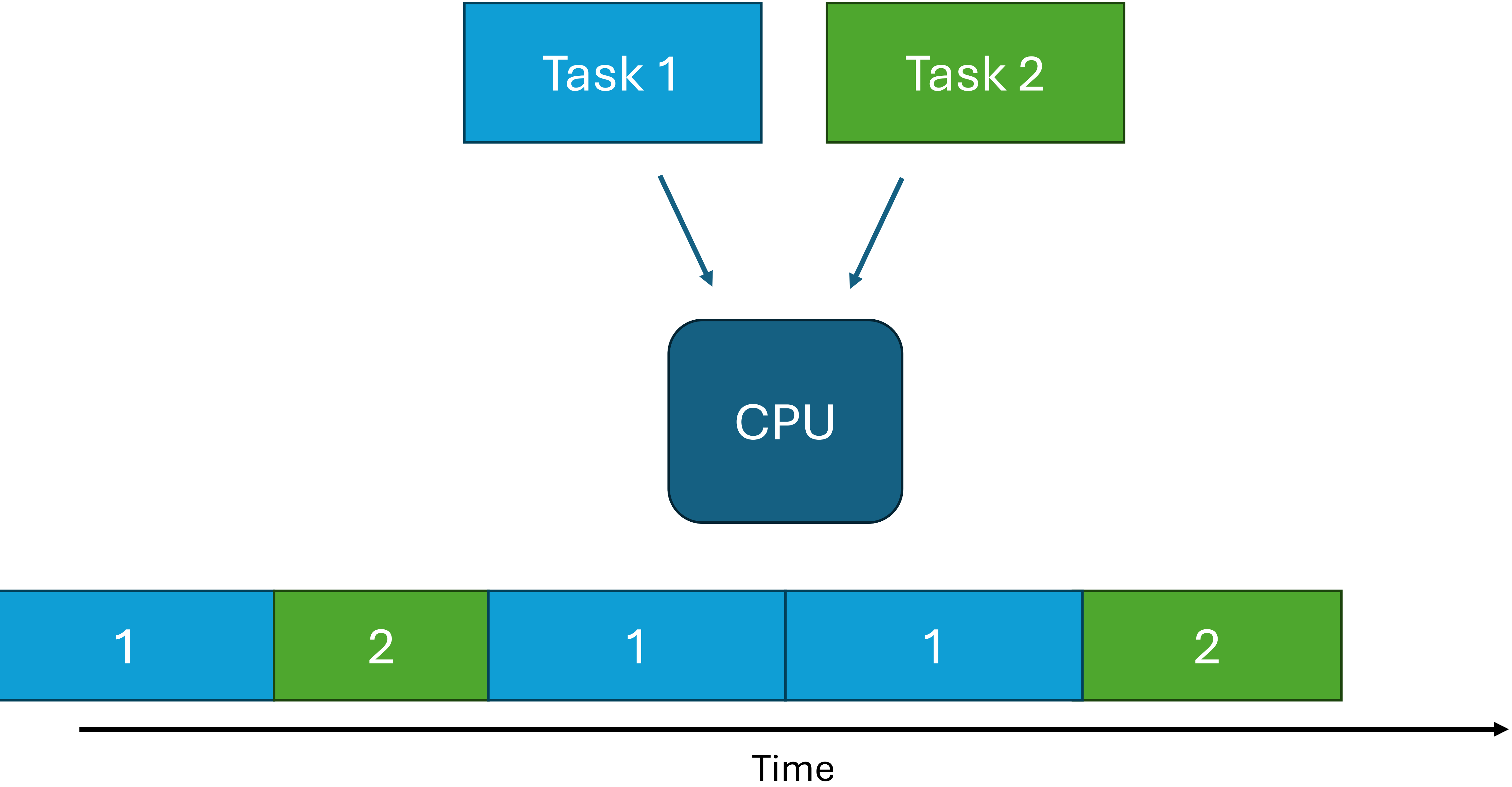
1

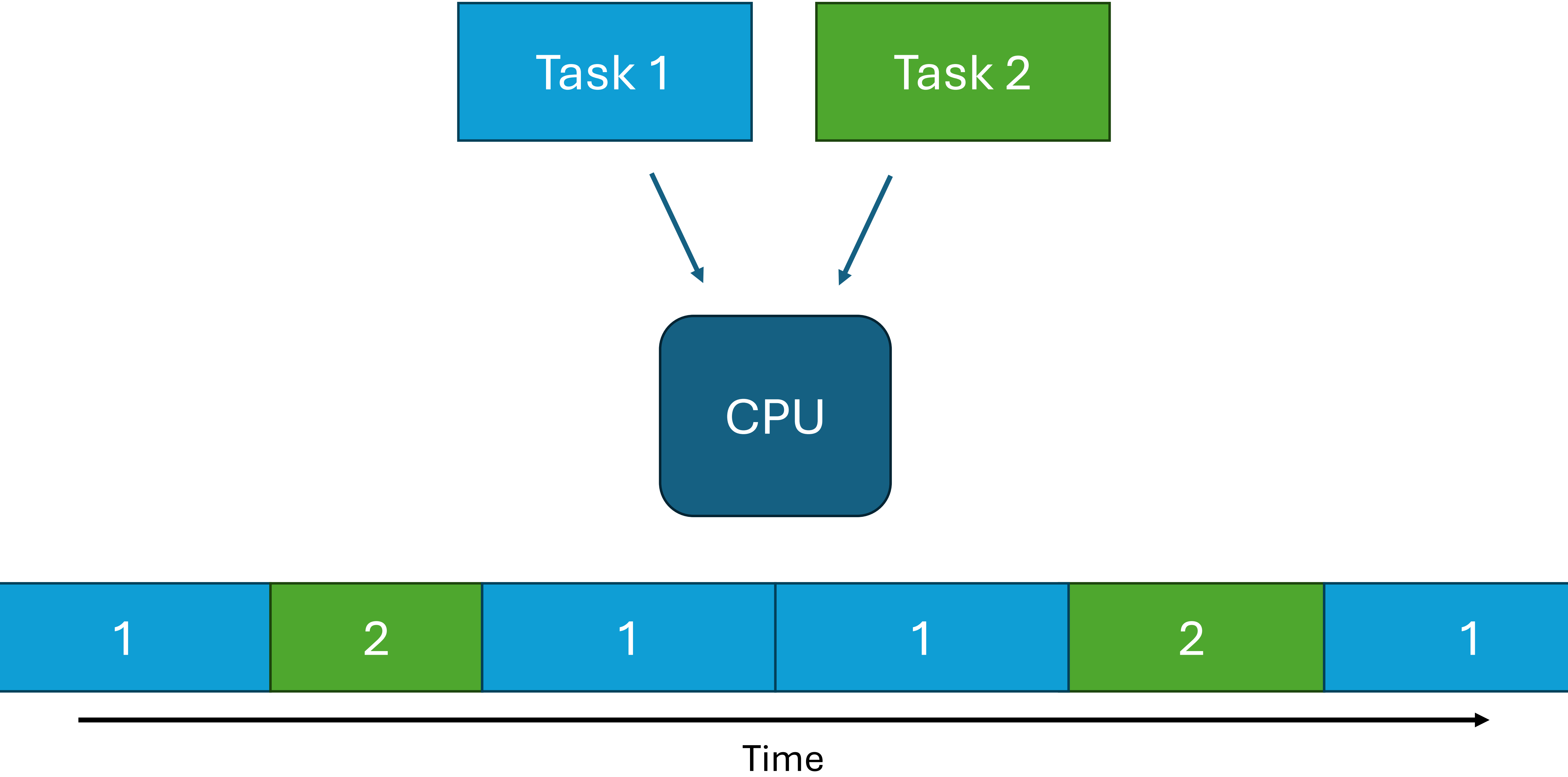
2

Time


Task 1
Task 2
CPU
1
2
1
Time


Task 1
Task 2
CPU
1
2
1
1
Time


Task 1
Task 2
CPU
1
2
1
1
2
Time



Task 1

Task 2

CPU

1 2 1 1 2 1

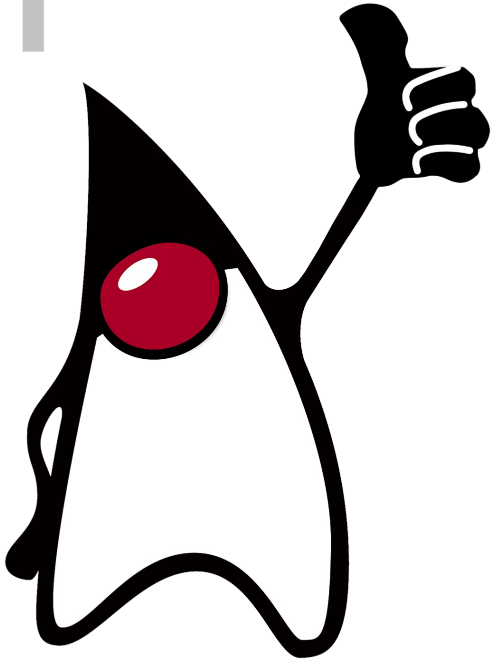
Time

Hear this sound?

It's my scheduler

It's my scheduler

Written in Java



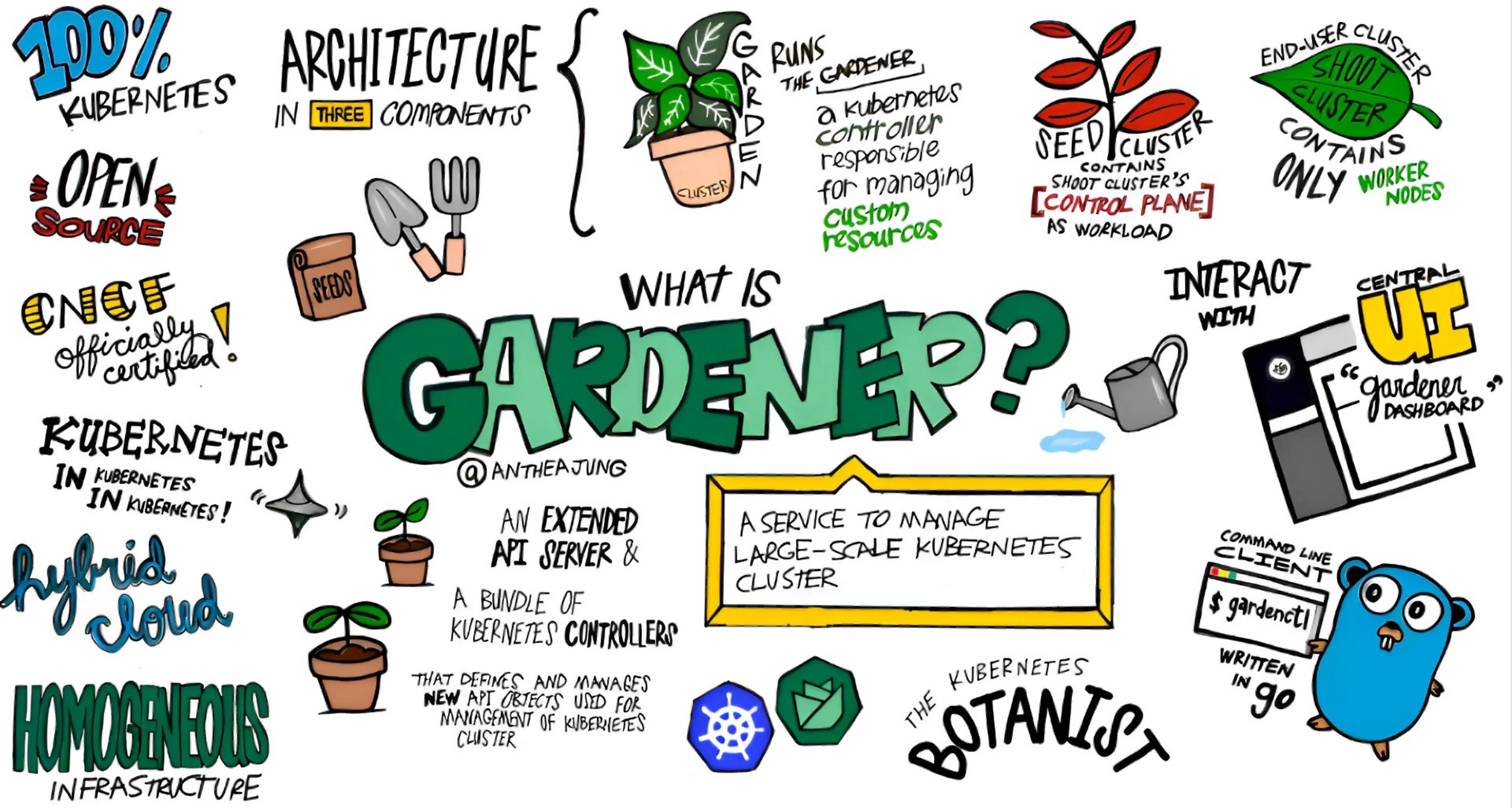
Why?

> “The only way of discovering the limits of the possible is to venture a little way past them into the impossible.

Clarke’s second law

https://www.flickr.com/photos/itupictures/16636142906

Jap Machine



WHAT IS GARDENER?

@ANTHEAJUNG

A SERVICE TO MANAGE LARGE-SCALE KUBERNETES CLUSTER

100% KUBERNETES

OPEN SOURCE

CNCF officially certified!

KUBERNETES IN KUBERNETES IN KUBERNETES!

hybrid cloud

HOMOGENEOUS INFRASTRUCTURE

ARCHITECTURE IN THREE COMPONENTS

GARDEN CLUSTER RUNS THE GARDENER

a kubernetes controller responsible for managing custom resources

SEED CLUSTER CONTAINS SHOOT CLUSTER'S [CONTROL PLANE] AS WORKLOAD

END-USER CLUSTER SHOOT CLUSTER CONTAINS ONLY WORKER NODES

AN EXTENDED API SERVER &

A BUNDLE OF KUBERNETES CONTROLLERS

THAT DEFINES AND MANAGES NEW API OBJECTS USED FOR MANAGEMENT OF KUBERNETES CLUSTER

INTERACT WITH CENTRAL UI "gardener DASHBOARD"

COMMAND LINE CLIENT

```
$ gardenctl
```

WRITTEN IN go

THE KUBERNETES BOTANIST

How to modify the kernel?

Traditional ways

1. Change the Kernel
2. Kernel module

Traditional ways

1. Change the Kernel
2. Kernel module

Interfaces are complicated

Traditional ways

1. Change the Kernel
2. Kernel module

Not possible with schedulers

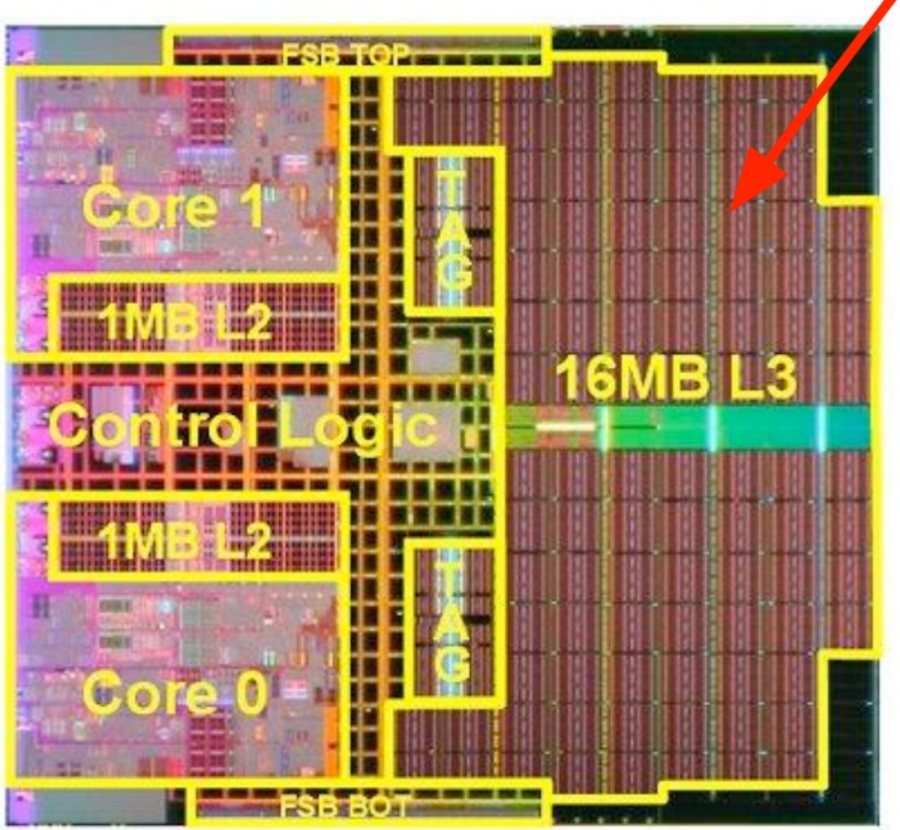
Problem: Only a few are implemented on your system

CFS was built in a simpler time

- Much smaller CPUs
- Topologies much more homogeneous
- Cores spaced further apart, migration cost typically high
- Power consumption and die area wasn't as important
- The fundamental assumptions behind heuristics may be easier to justify

Just two cores

Just one L3 cache

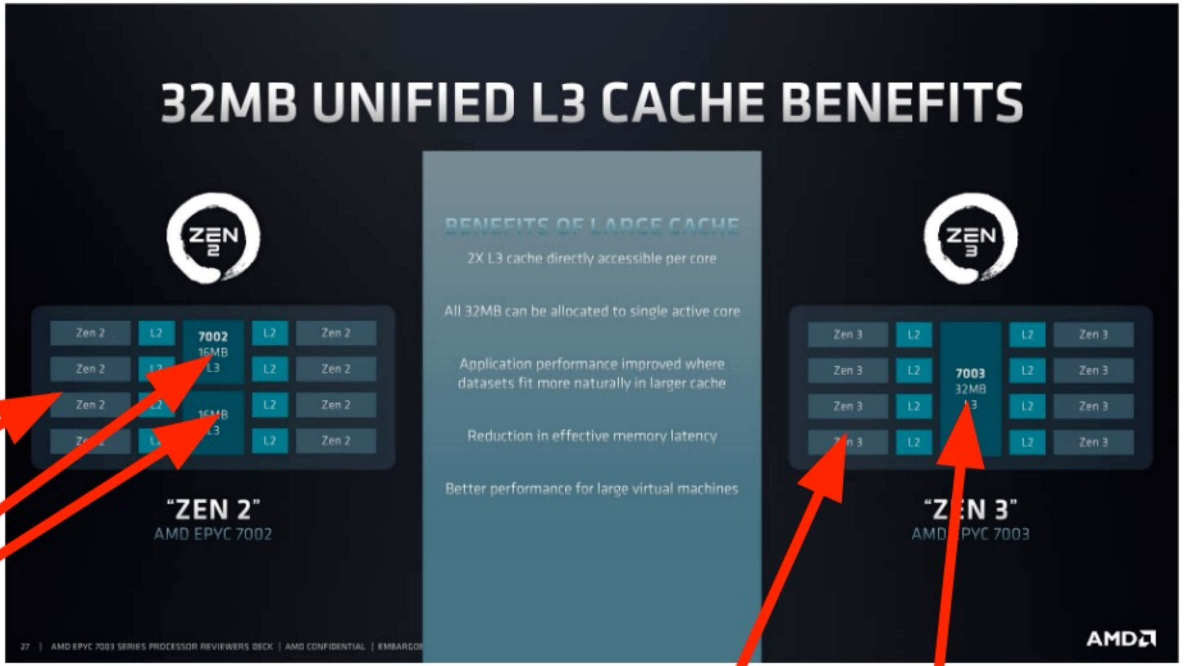


Intel Xeon MP 71xx die

David Vernet, https://archives.kernel-recipes.org/wp-content/uploads/2025/01/Sched_Ext.pdf

Architectures *much* more complicated now

- Heterogeneity is becoming the norm
- Non-uniform memory accesses between sockets
- Non-uniform memory accesses between CCDs
- Non-uniform memory accesses between CCXs
- Non-uniform memory accesses between CCXs in the same CCD



32MB UNIFIED L3 CACHE BENEFITS

BENEFITS OF LARGE CACHE

2X L3 cache directly accessible per core

All 32MB can be allocated to single active core

Application performance improved where datasets fit more naturally in larger cache

Reduction in effective memory latency

Better performance for large virtual machines

AMD Zen 2 Rome

4 cores per "CCX"
8 cores per "CCD"

2 L3 caches per CCD!

8 cores per "CCX"
8 cores per "CCD"

1 L3 cache per CCD!

David Vernet, https://archives.kernel-recipes.org/wp-content/uploads/2025/01/Sched_Ext.pdf

Let's create our own

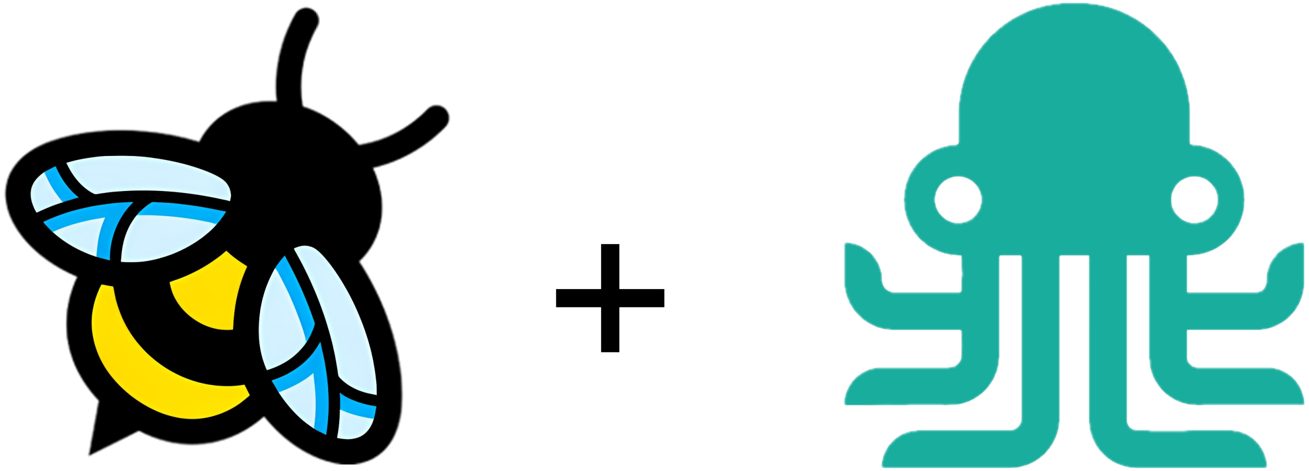
Let's create our own

Has someone done this before in this room?

Let's create our own

How hard can it be?

Let's create our own



Who was there in the Firewall Talk?

Skip ahead

eBPF

> “eBPF is a crazy technology, it’s like putting JavaScript into the Linux kernel

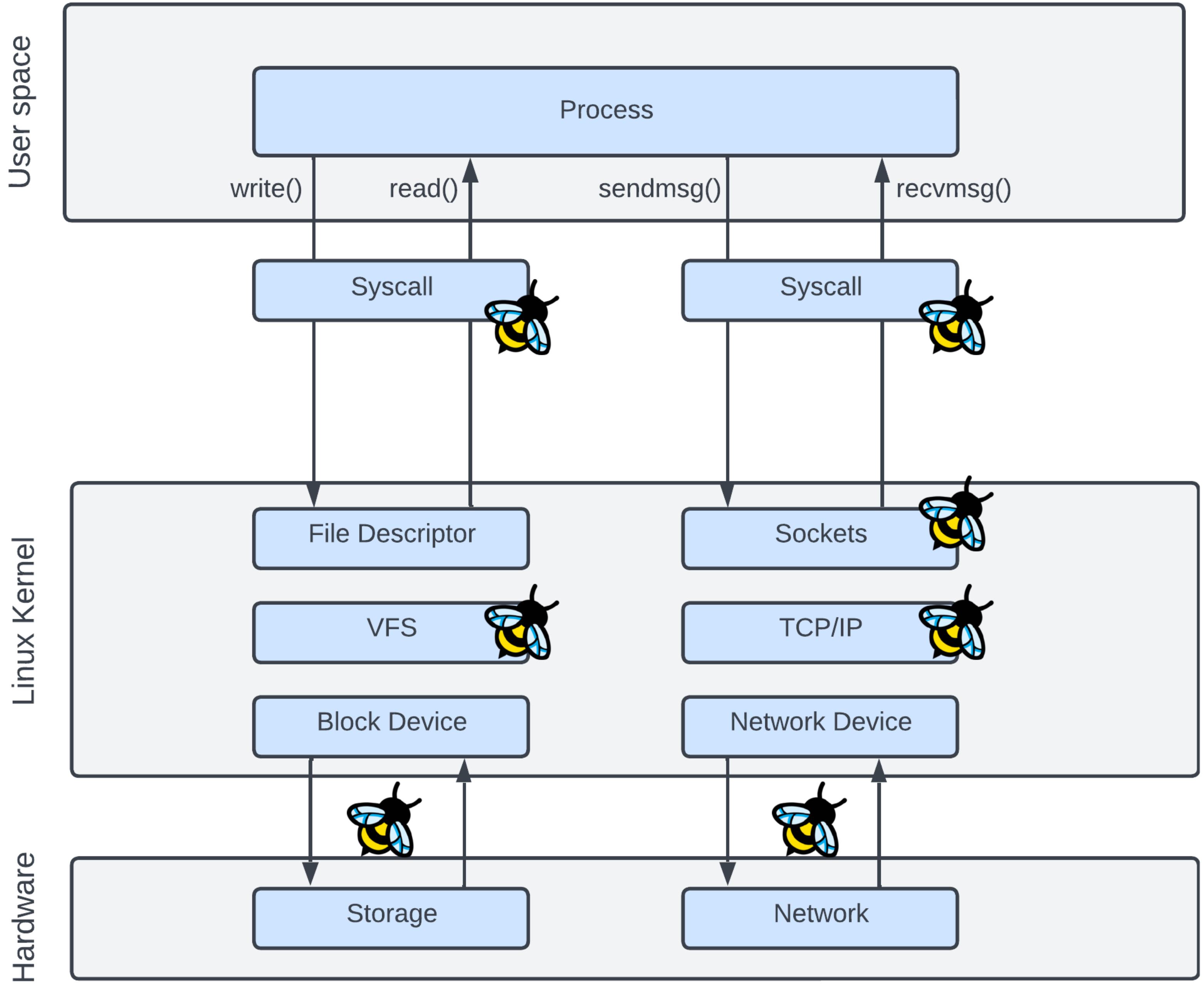
Brendan Gregg

https://www.youtube.com/watch?v=tDacjrSCeq4

> “eBPF is a crazy technology, it’s like putting JavaScript into the Linux kernel

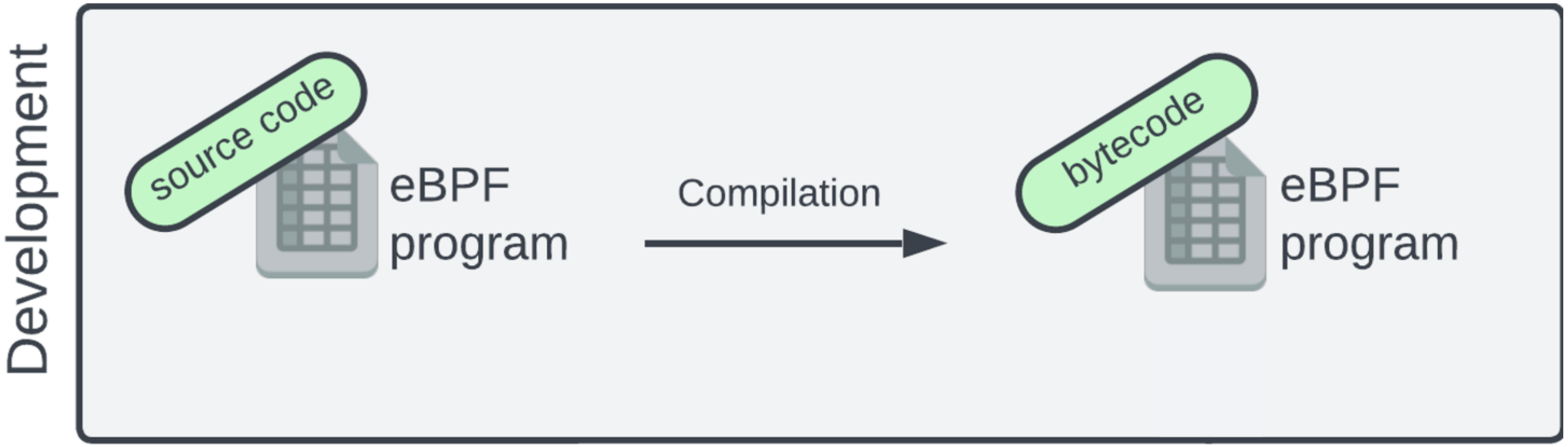
Brendan Gregg

https://www.facesofopensource.com/brendan-gregg/



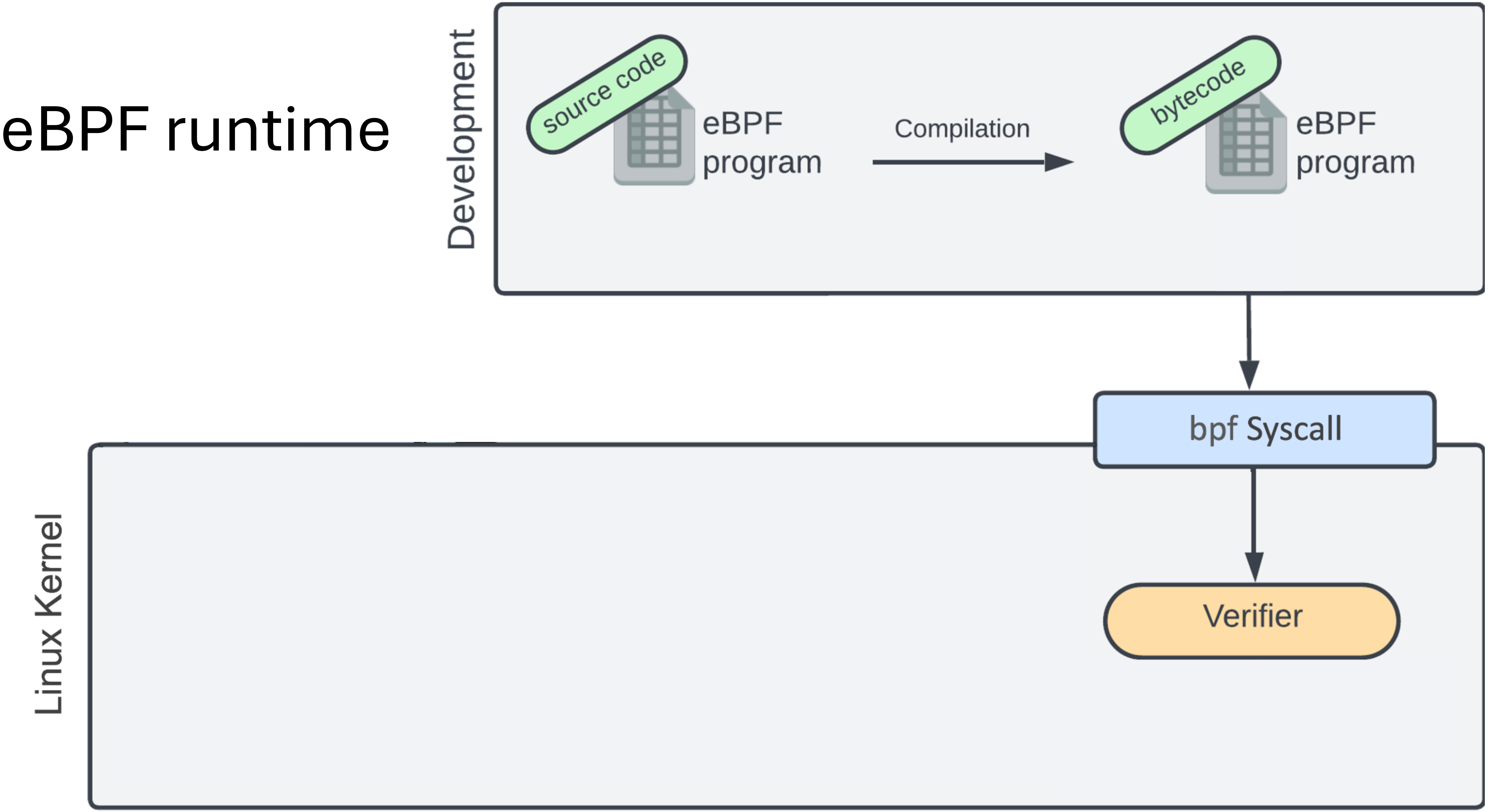
https://ebpf.io/what-is-ebpf/

eBPF runtime

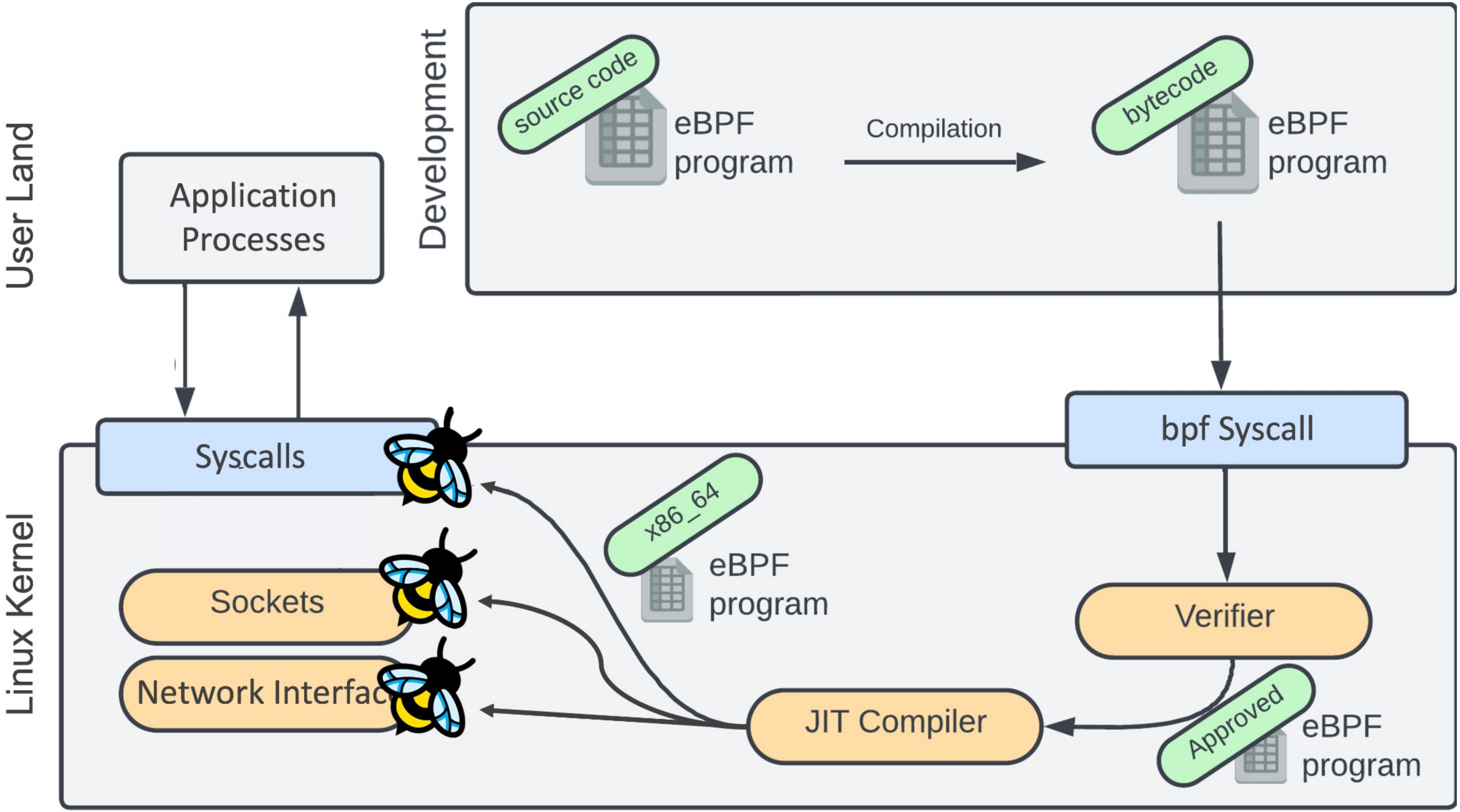


Courtesy of Mohammed Aboullaite

eBPF runtime



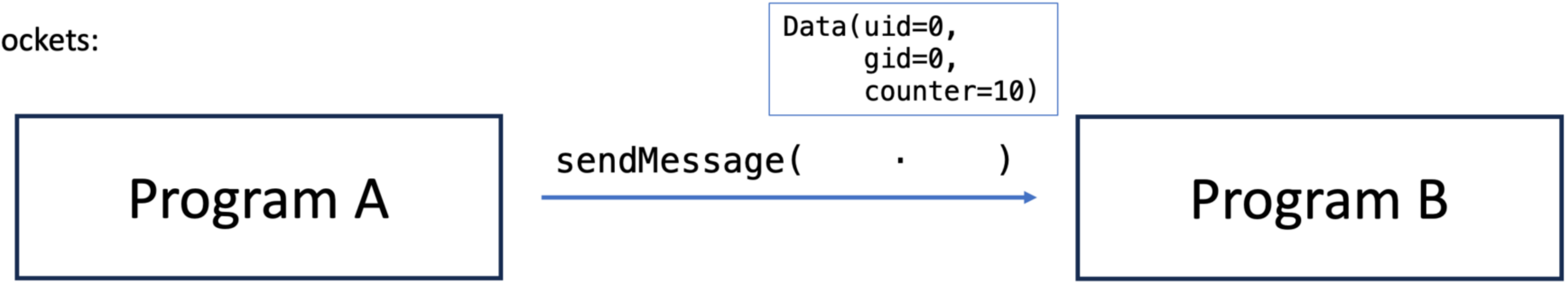
Courtesy of Mohammed Aboullaite



Courtesy of Mohammed Aboullaite

How to share data?

via sockets:



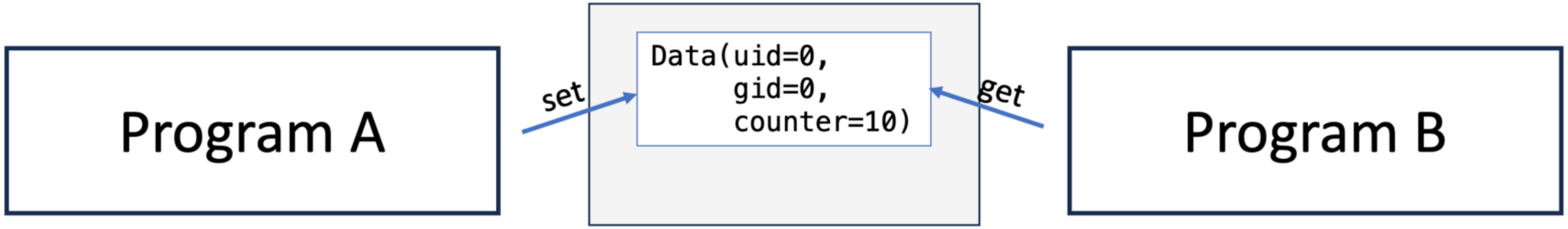
Program A

sendMessage(·)

```
Data(uid=0,
     gid=0,
     counter=10)
```

Program B

via shared memory:



Program A

set

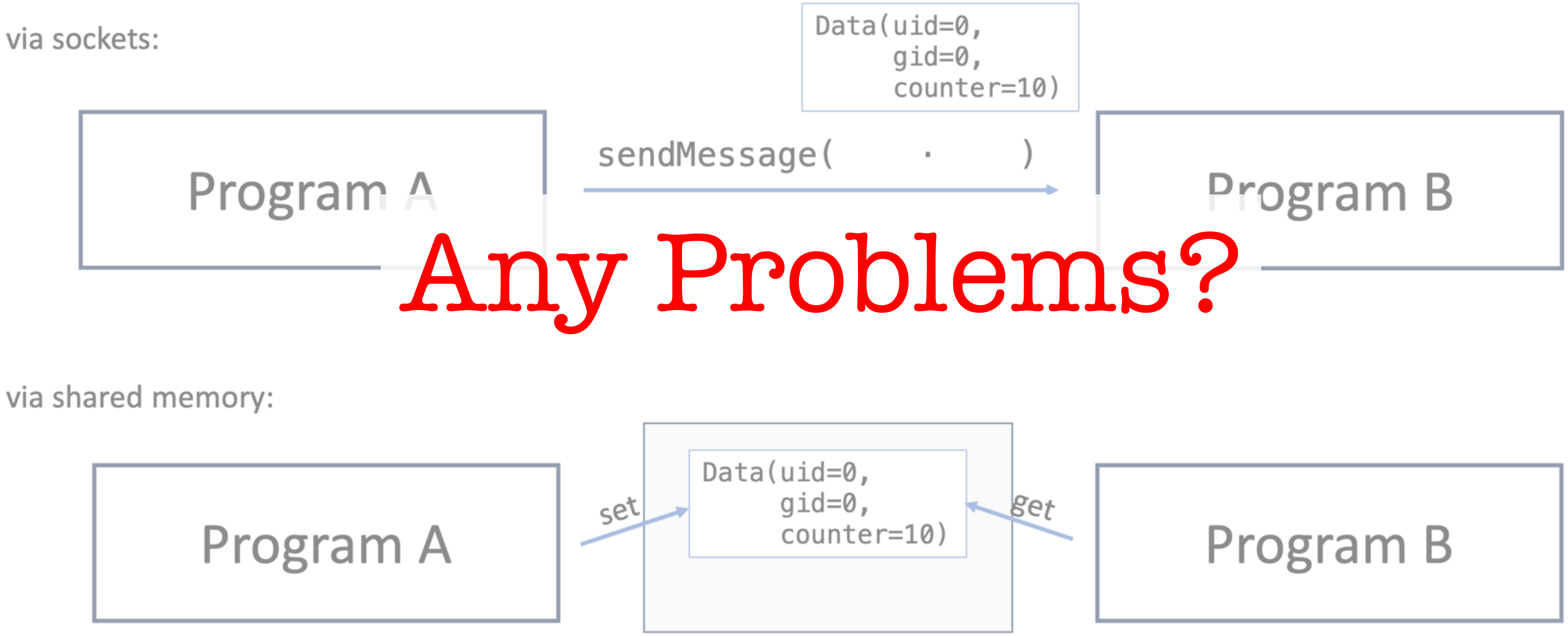
```
Data(uid=0,
     gid=0,
     counter=10)
```

get

Program B

https://mostlynerdless.de/blog/2024/01/12/hello-ebpf-recording-data-in-basic-ebpf-maps-2/

How to share data?

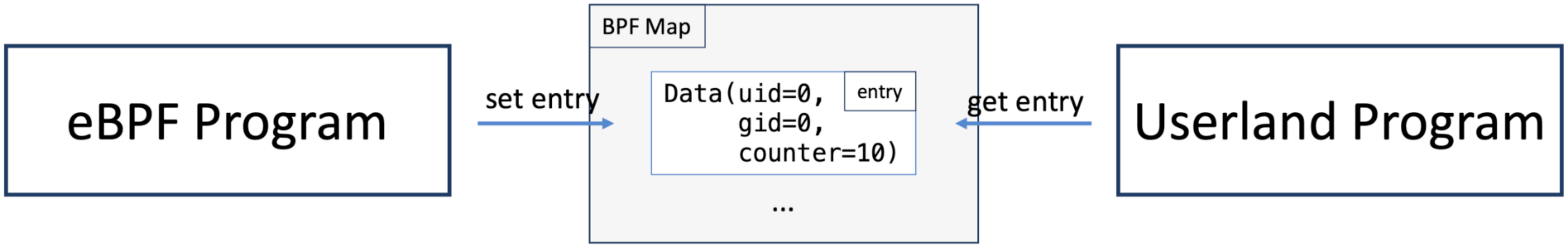


Any Problems?

https://mostlynerdless.de/blog/2024/01/12/hello-ebpf-recording-data-in-basic-ebpf-maps-2/

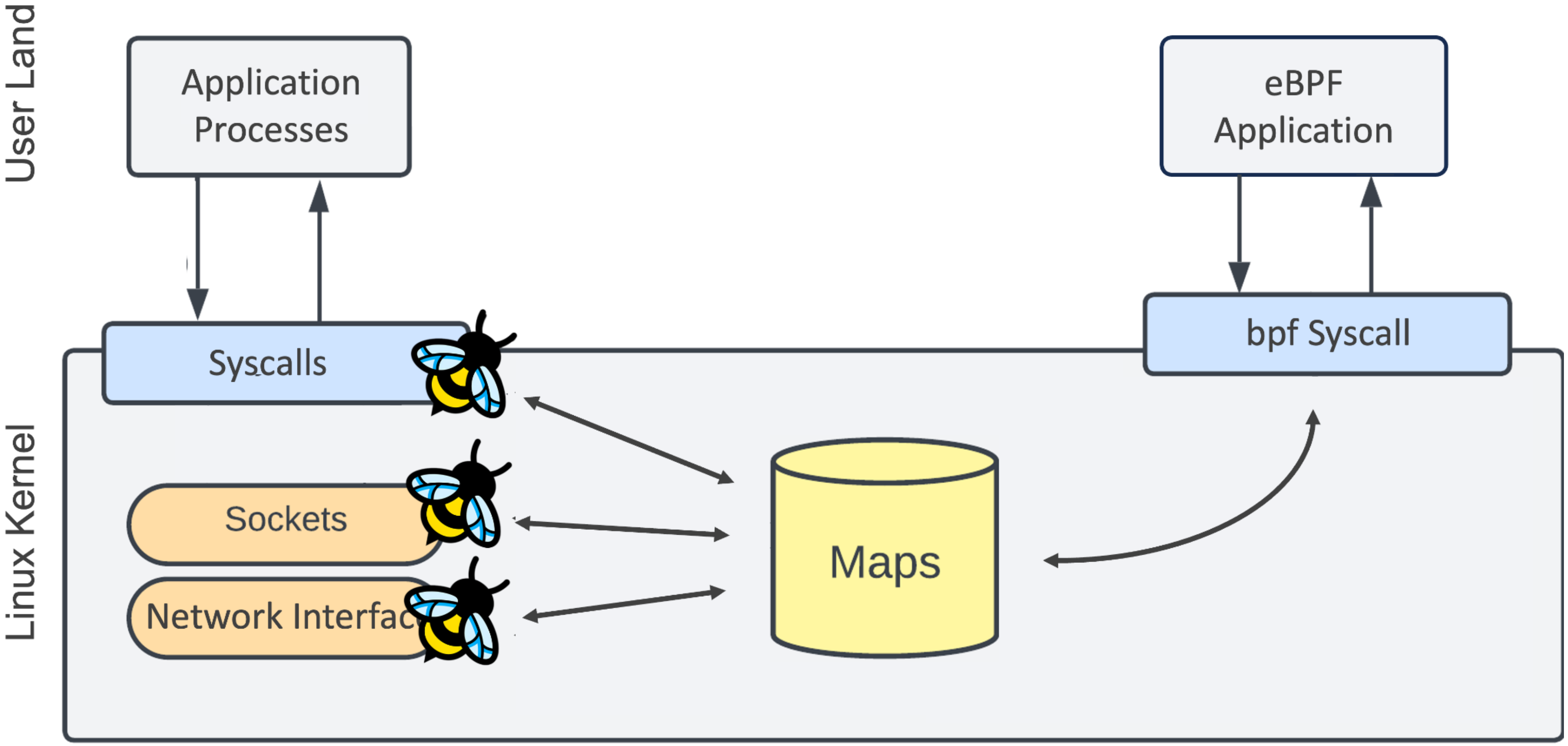
How to share data?

via eBPF maps:

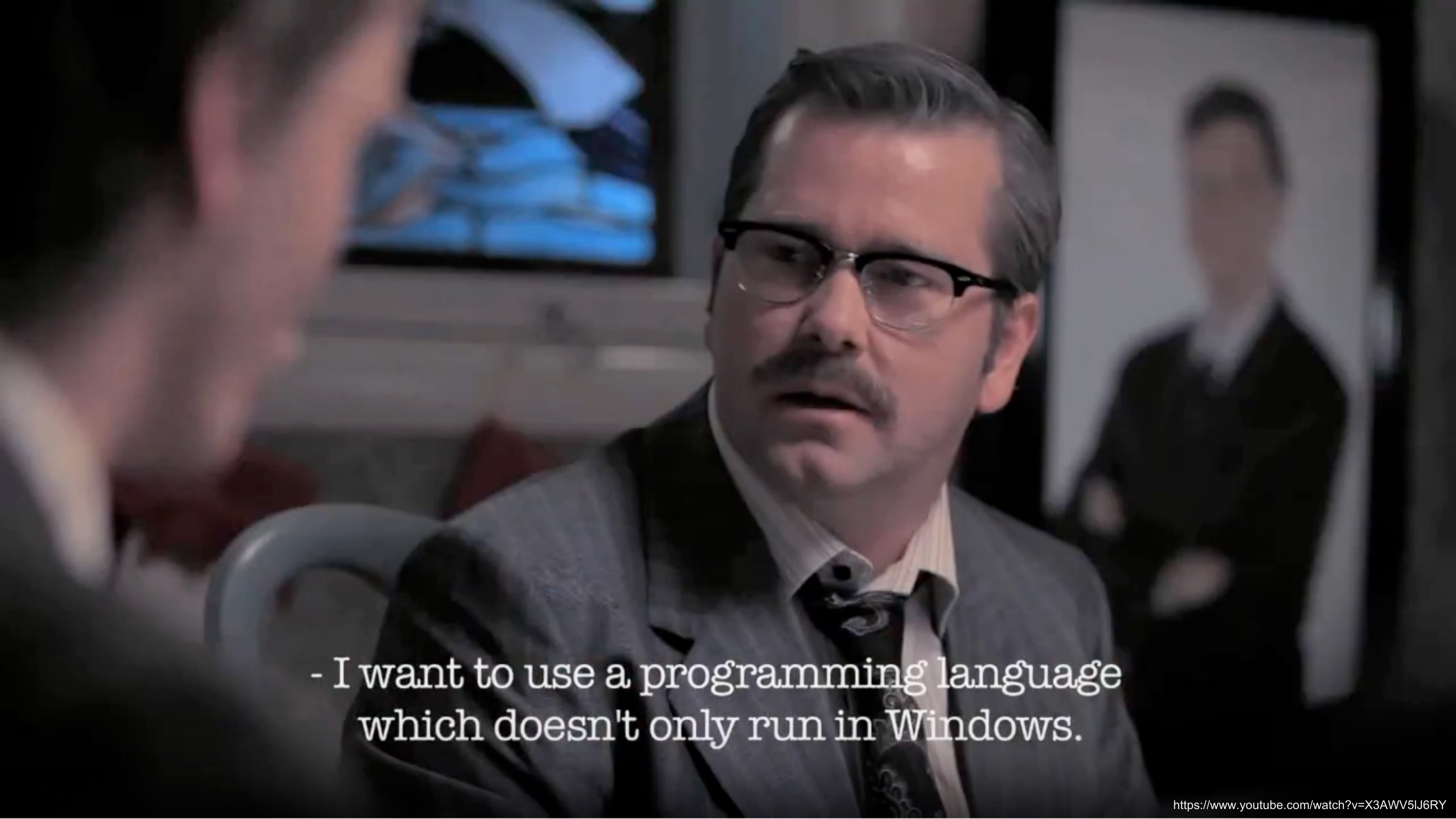


https://mostlynerdless.de/blog/2024/01/12/hello-ebpf-recording-data-in-basic-ebpf-maps-2/

eBPF Maps

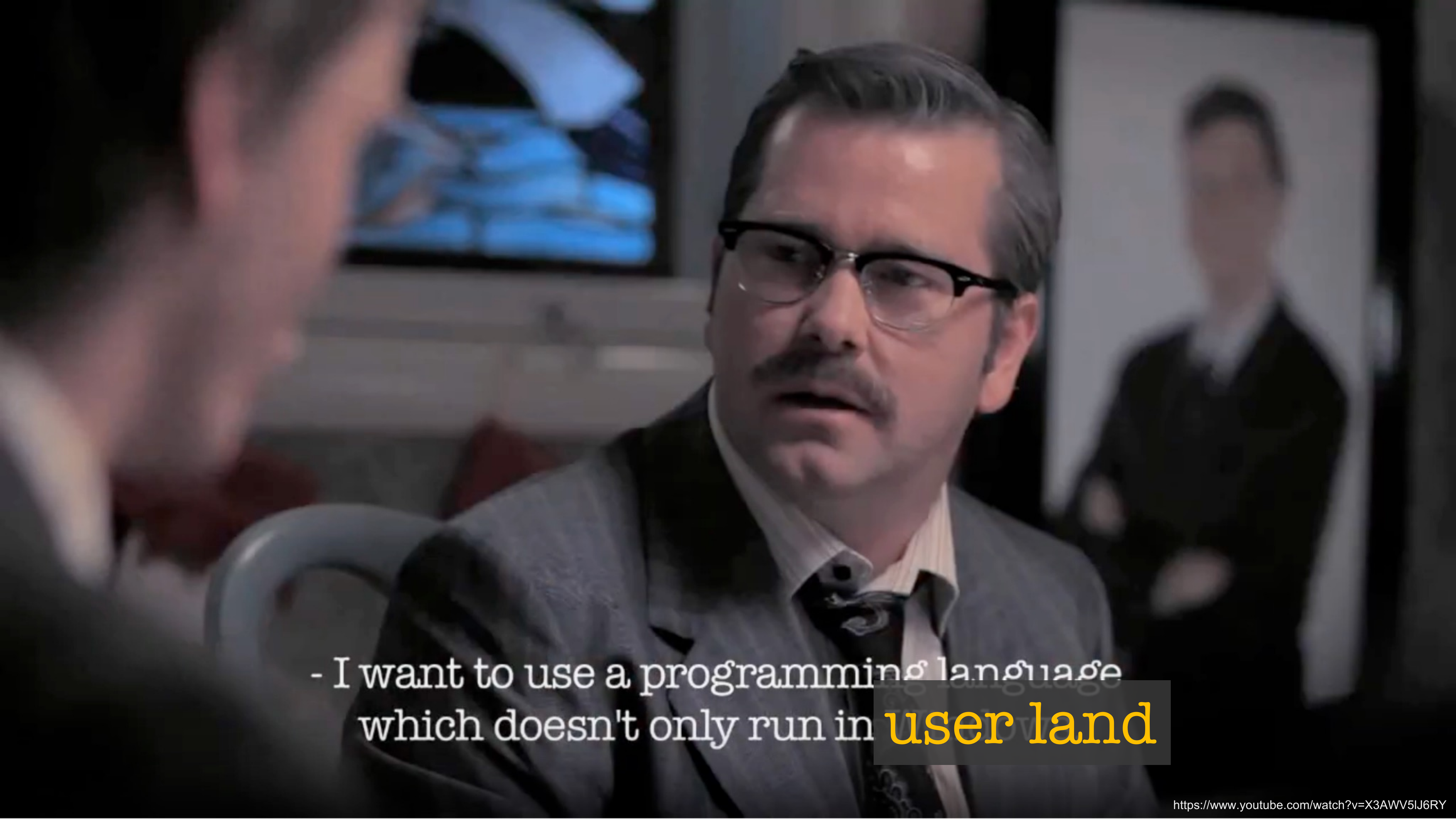


Courtesy of Mohammed Aboullaite



- I want to use a programming language
which doesn't only run in Windows.

https://www.youtube.com/watch?v=X3AWV5lJ6RY

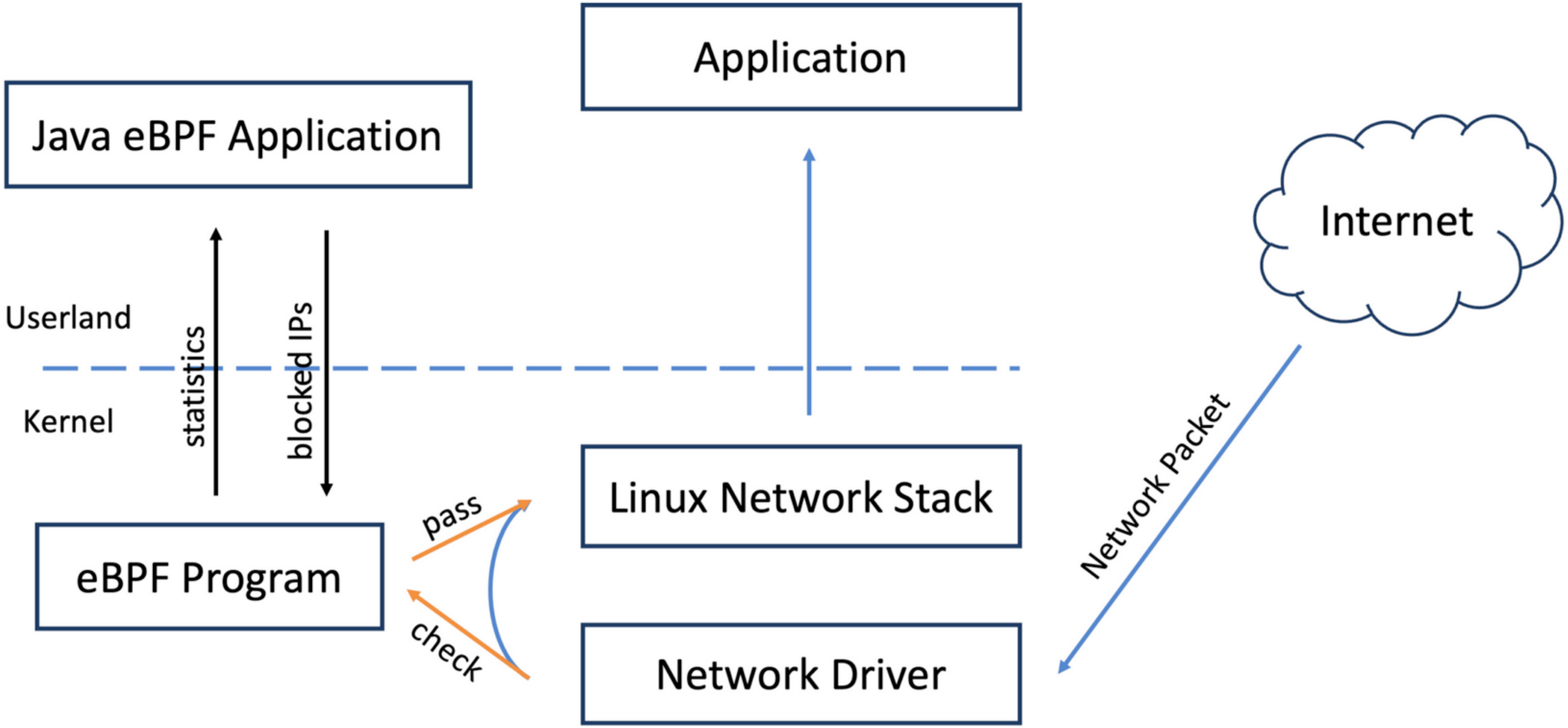

- I want to use a programming language
which doesn't only run in

user land

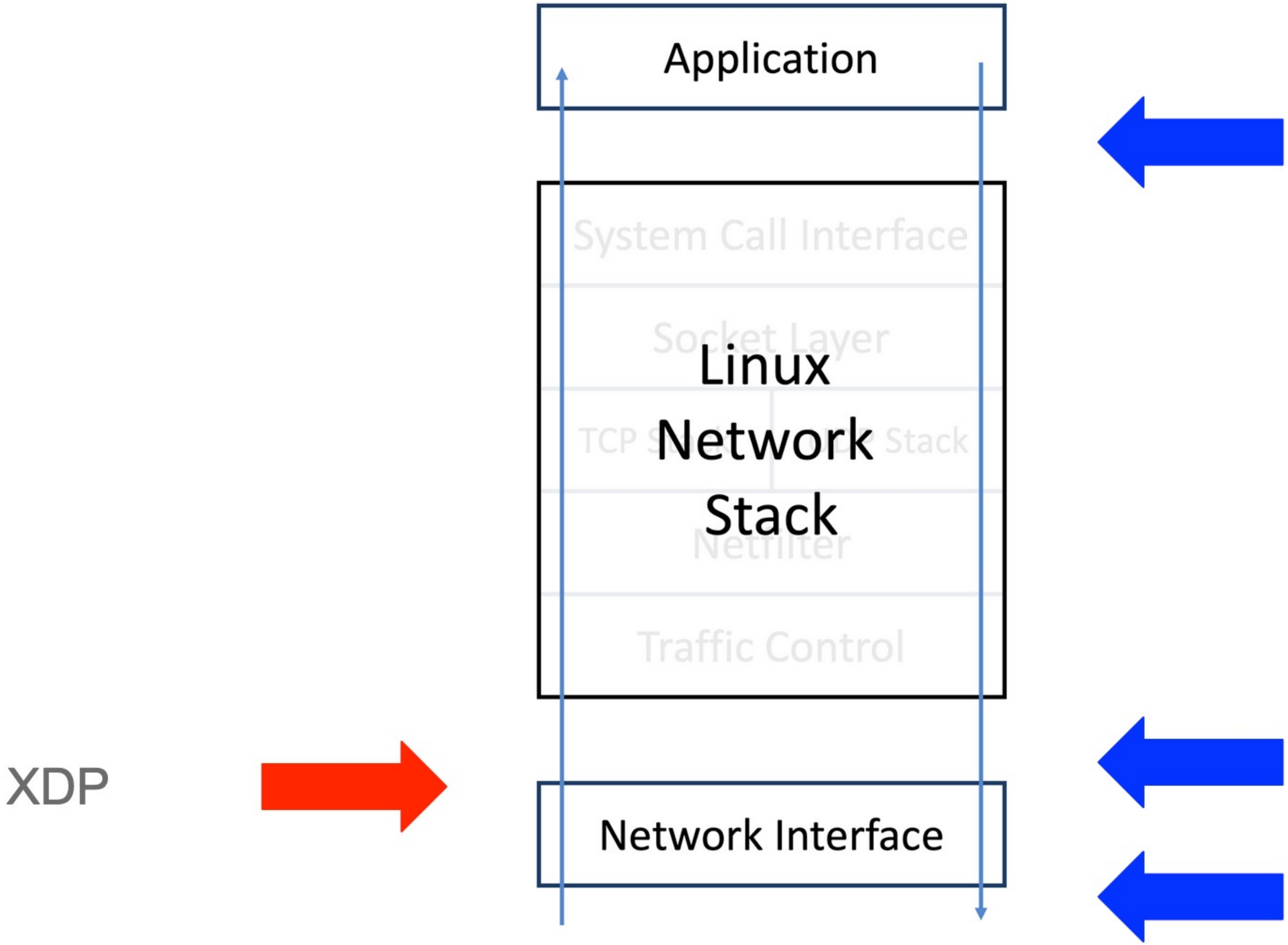
https://www.youtube.com/watch?v=X3AWV5lJ6RY

Demo

XDP



https://mostlynerdless.de/blog/2024/04/22/hello-ebpf-xdp-based-packet-filter-9



Application

System Call Interface

Socket Layer

Linux Network Stack

TCP Stack

UDP Stack

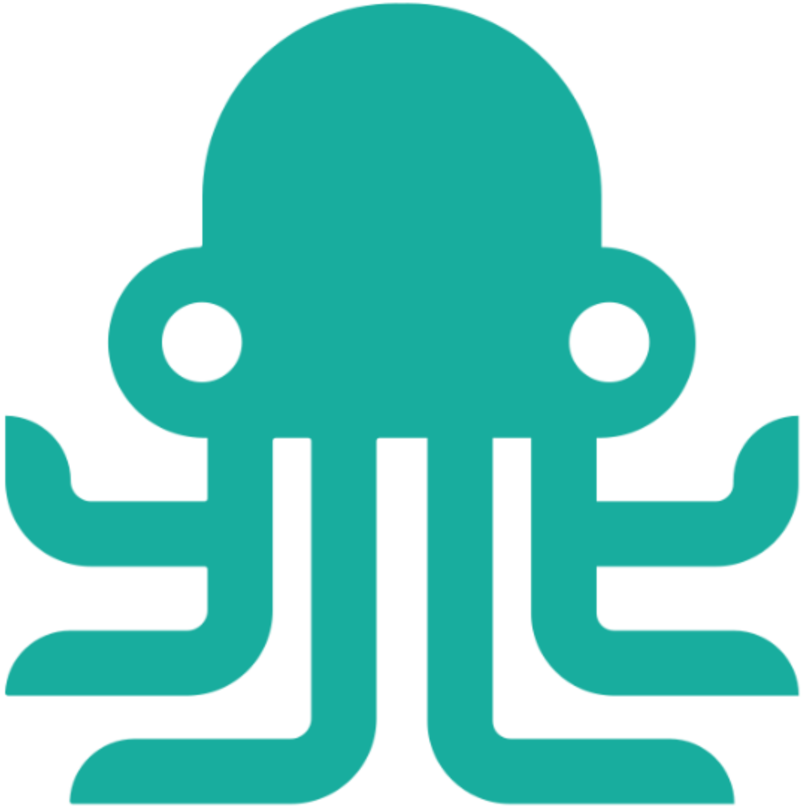
Netfilter

Traffic Control

XDP

Network Interface

Back to scheduling



sched_ext

Sched Ext

The extensible sched_class

David Vernet
Kernel engineer

Meta

“

1. Ease of experimentation and exploration
2. Customization
3. Rapid scheduler deployments

https://lwn.net/Articles/978911/

Typical Scheduler Goals

Fairness

Typical Scheduler Goals

Resource Utilization

Typical Scheduler Goals

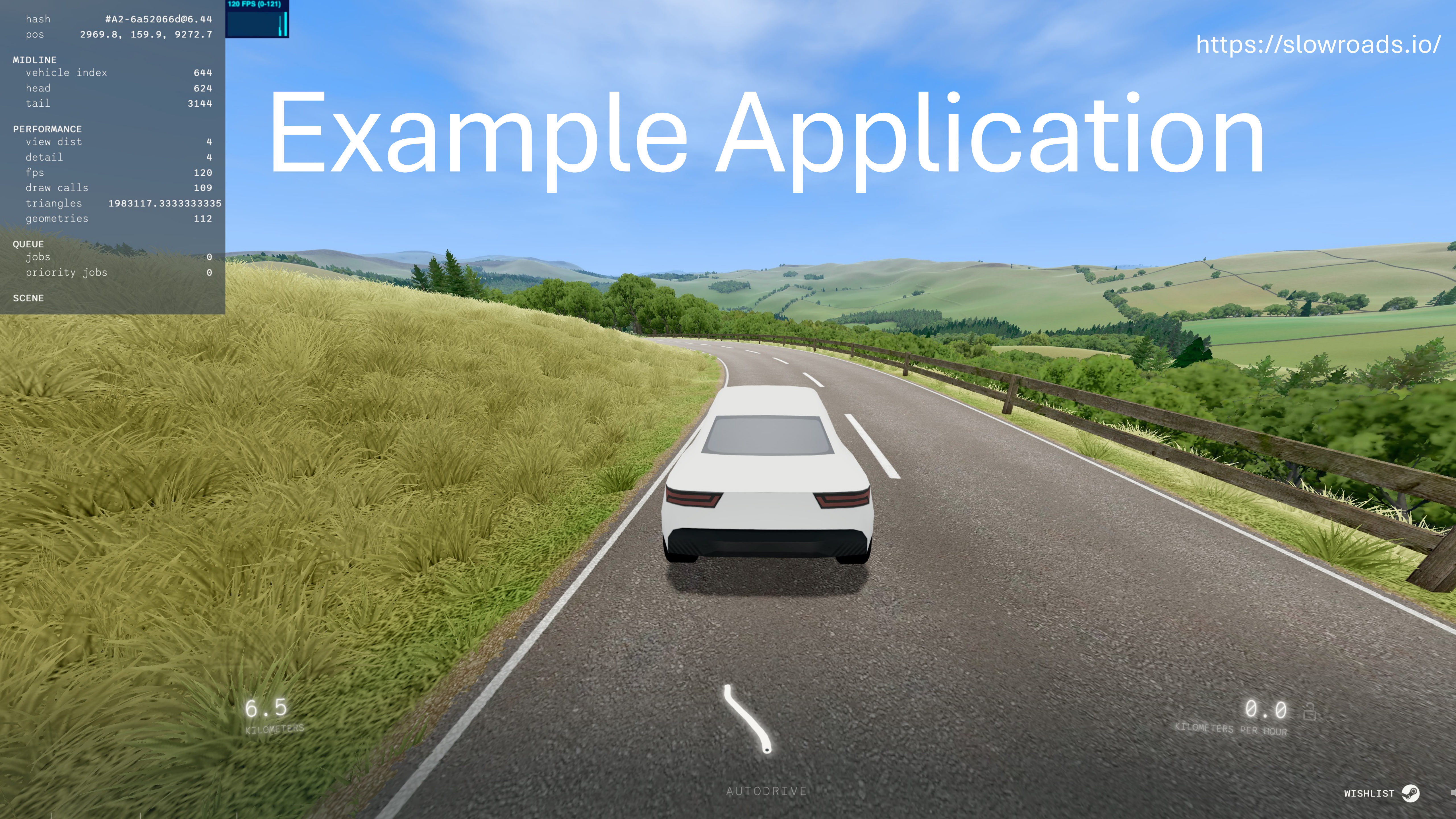
Overhead

Typical Scheduler Goals

Responsiveness

Example Application

https://slowroads.io/



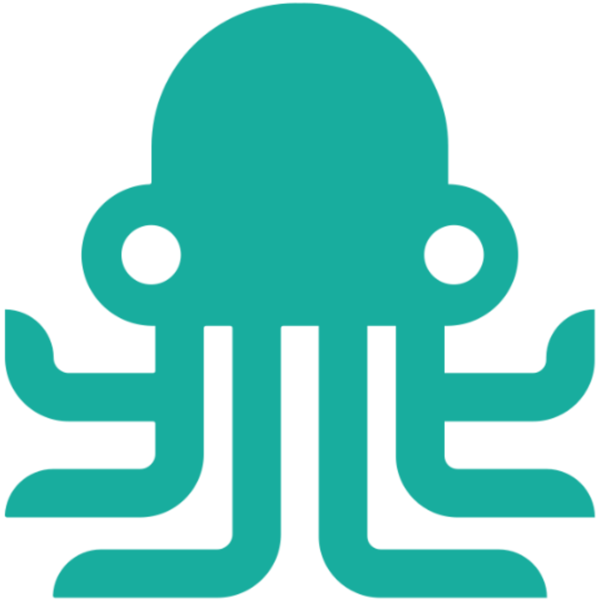
```
hash          #A2-6a52066d@6.44
pos         2969.8, 159.9, 9272.7

MIDLINE
  vehicle index                 644
  head                          624
  tail                         3144

PERFORMANCE
  view dist                       4
  detail                          4
  fps                           120
  draw calls                    109
  triangles       1983117.3333333335
  geometries                    112

QUEUE
  jobs                            0
  priority jobs                   0

SCENE
```

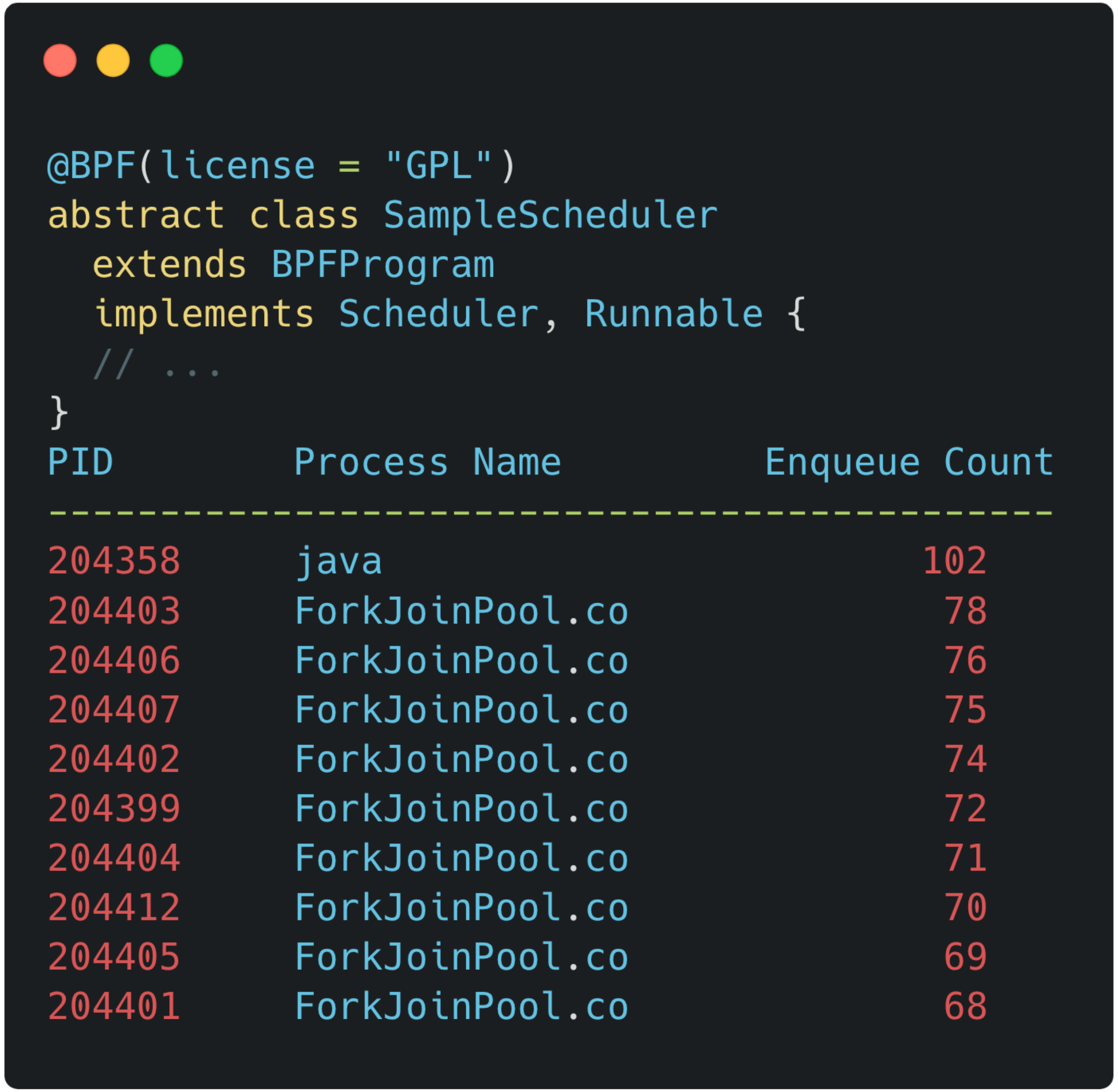


sched_ext

eBPF

hello eBPF

Let's create a scheduler



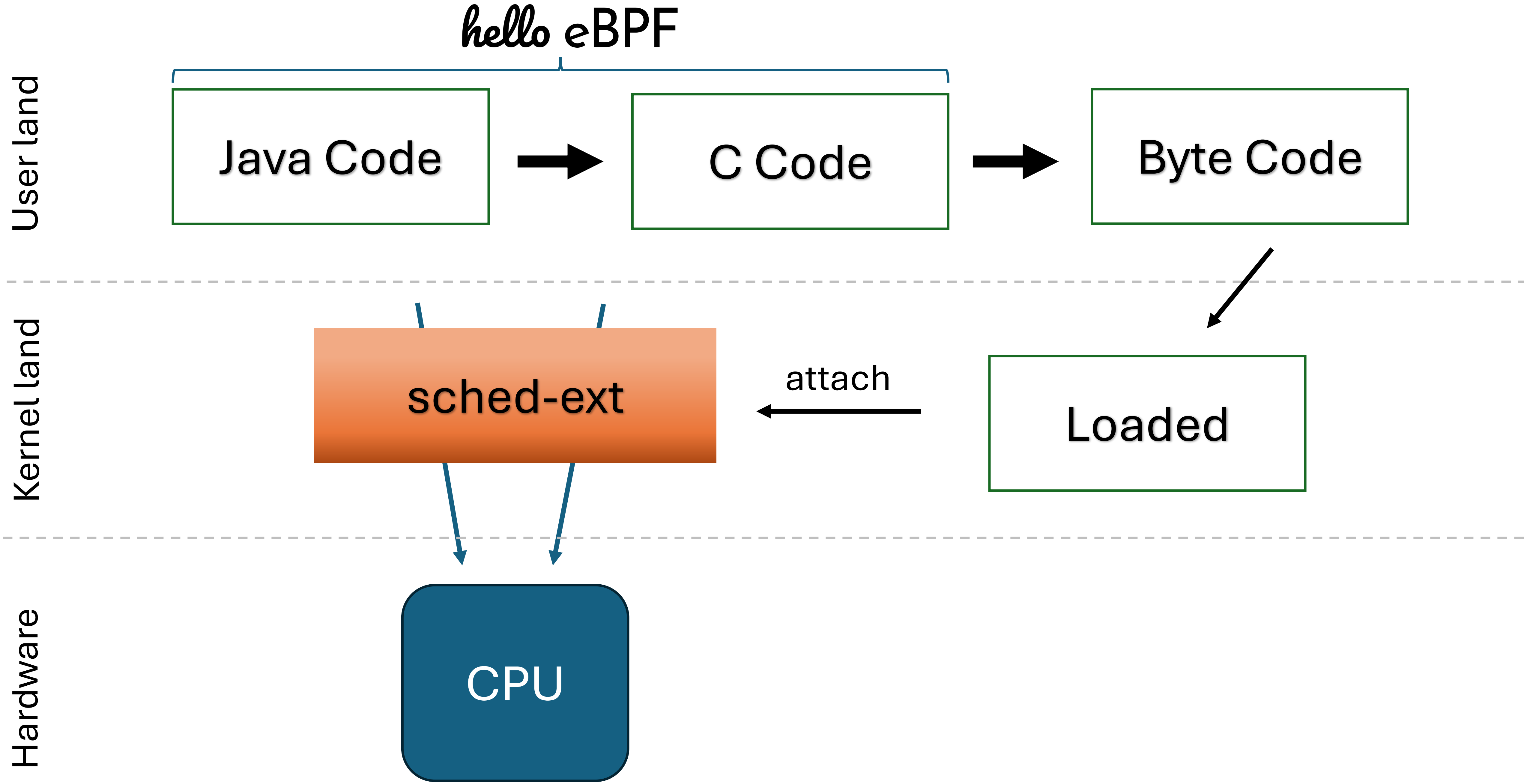
```
@BPF(license = "GPL")
abstract class SampleScheduler
  extends BPFProgram
  implements Scheduler, Runnable {
  // ...
}
PID        Process Name         Enqueue Count
---------------------------------------------
204358     java                        102
204403     ForkJoinPool.co              78
204406     ForkJoinPool.co              76
204407     ForkJoinPool.co              75
204402     ForkJoinPool.co              74
204399     ForkJoinPool.co              72
204404     ForkJoinPool.co              71
204412     ForkJoinPool.co              70
204405     ForkJoinPool.co              69
204401     ForkJoinPool.co              68
```

What is the performance?

Good*

* *For a typical Java benchmark*

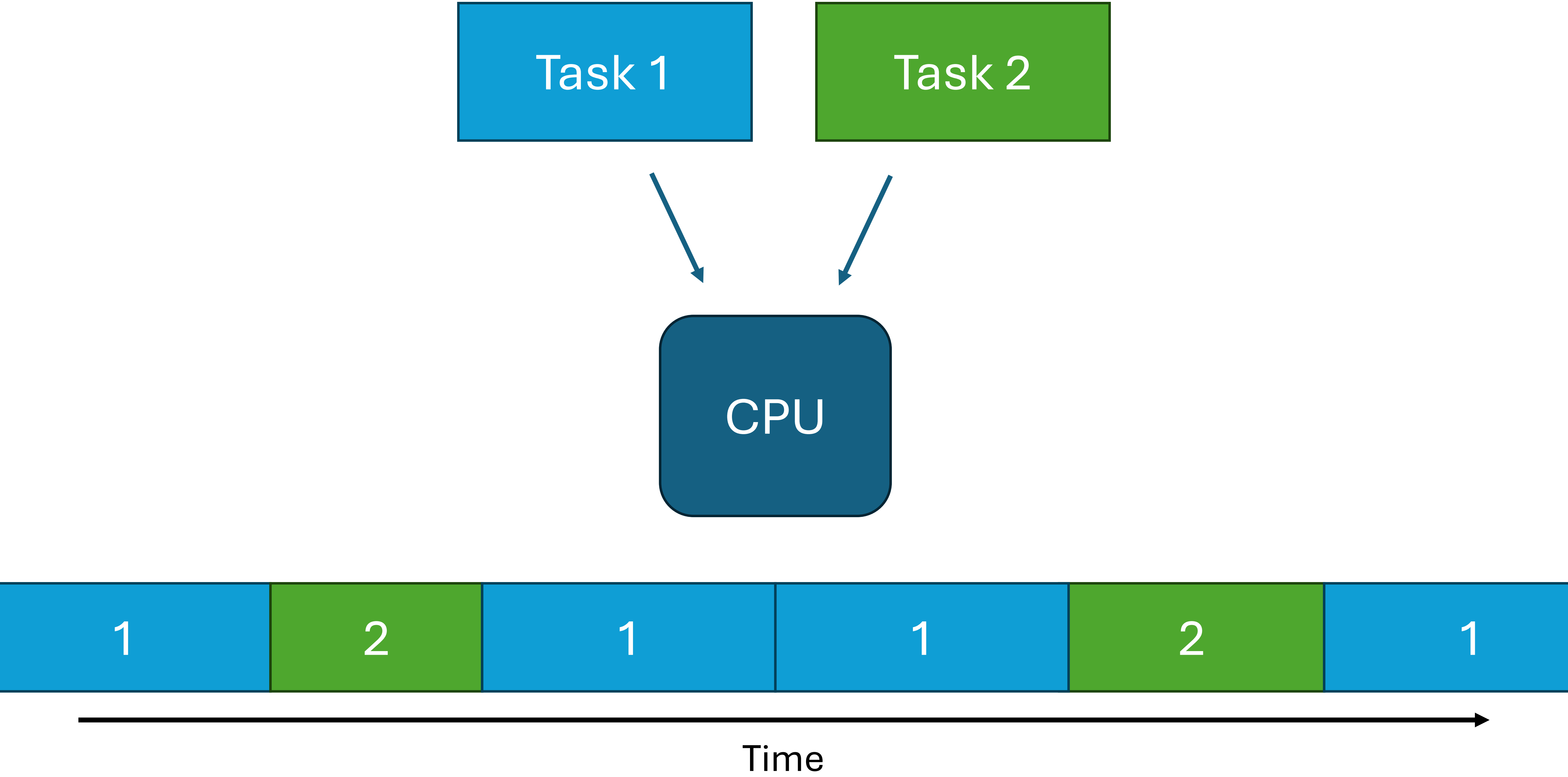
How does it work?

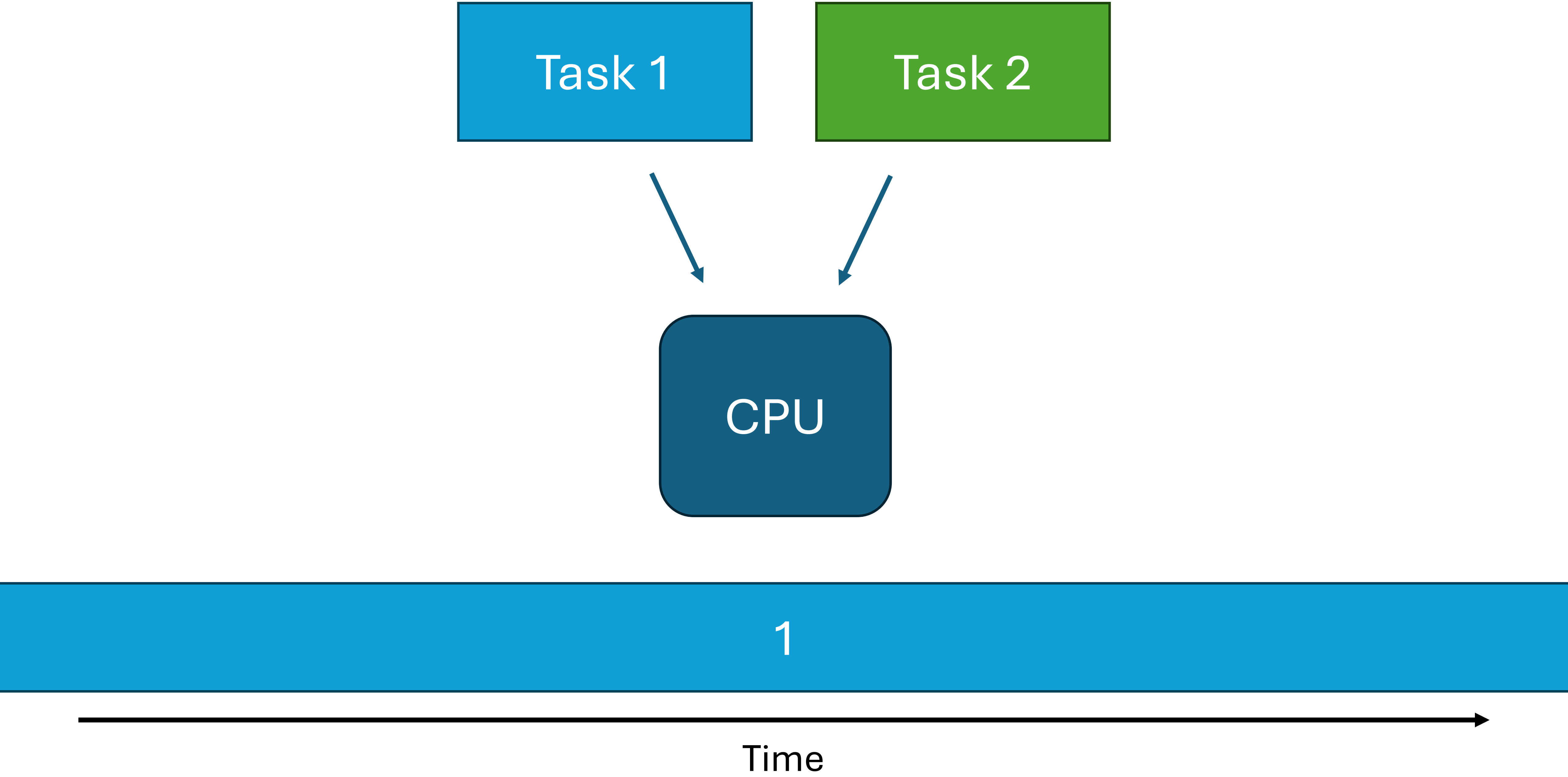

hello eBPF
User land
Java Code
C Code
Byte Code
Kernel land
sched-ext
attach
Loaded
Hardware
CPU

Let’s see some schedulers

First-Come, First-Served Scheduler

Run as long as you want,
we won't stop you


Task 1
Task 2
CPU
1
2
1
1
2
1
Time



Task 1

Task 2

CPU

1

Time

FCFSScheduler

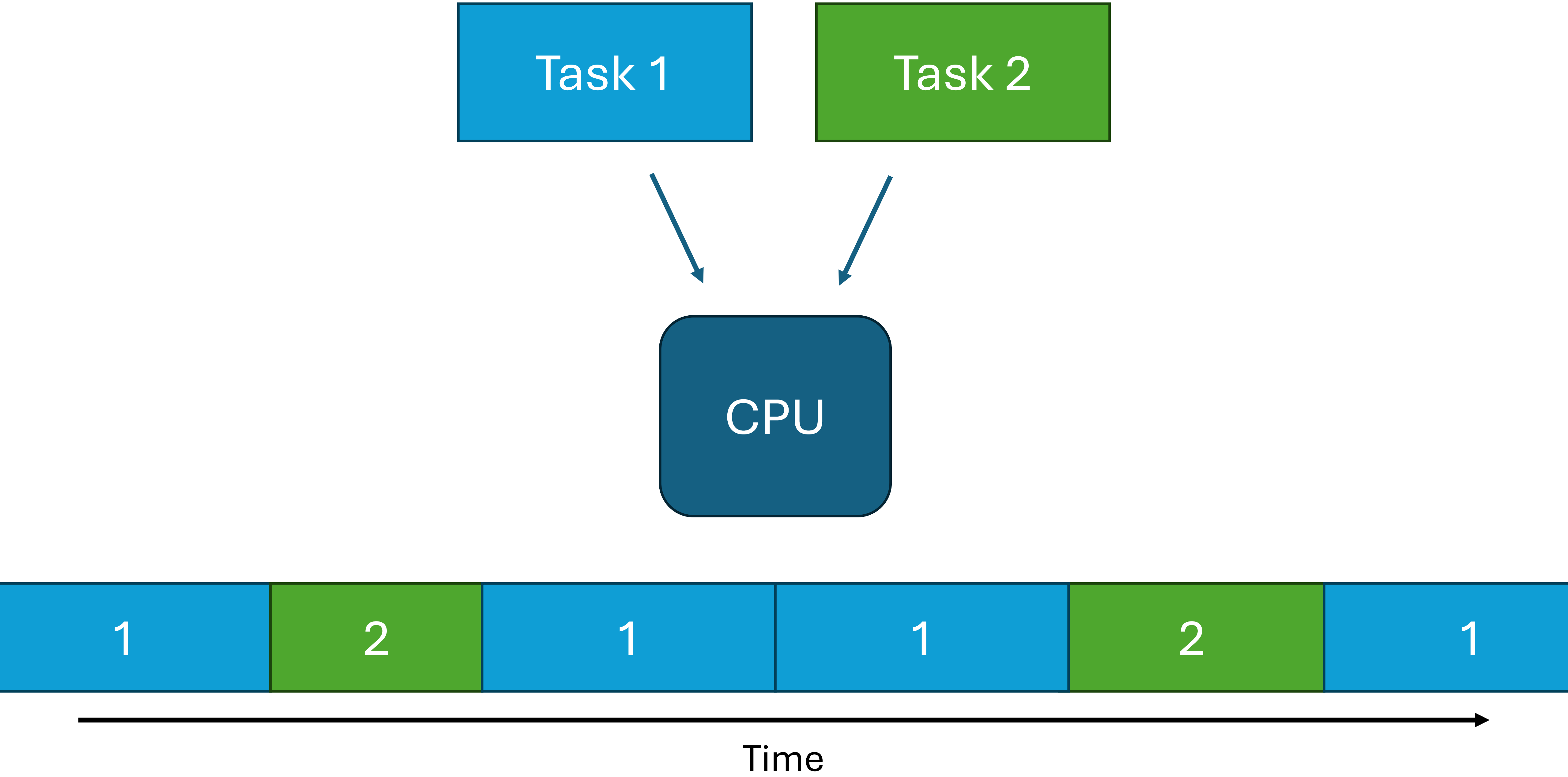
Making errors is normal

```
/**
 * @timeout_ms: The maximum amount of time, in milliseconds, that a
 * runnable task should be able to wait before being scheduled. The
 * maximum timeout may not exceed the default timeout of 30 seconds.
 *
 * Defaults to the maximum allowed timeout value of 30 seconds.
 */
u32 timeout_ms;
```

https://github.com/torvalds/linux/blob/master/kernel/sched/ext.c

First-Come, First-Out Scheduler

The early bird eats the time slice



Task 1

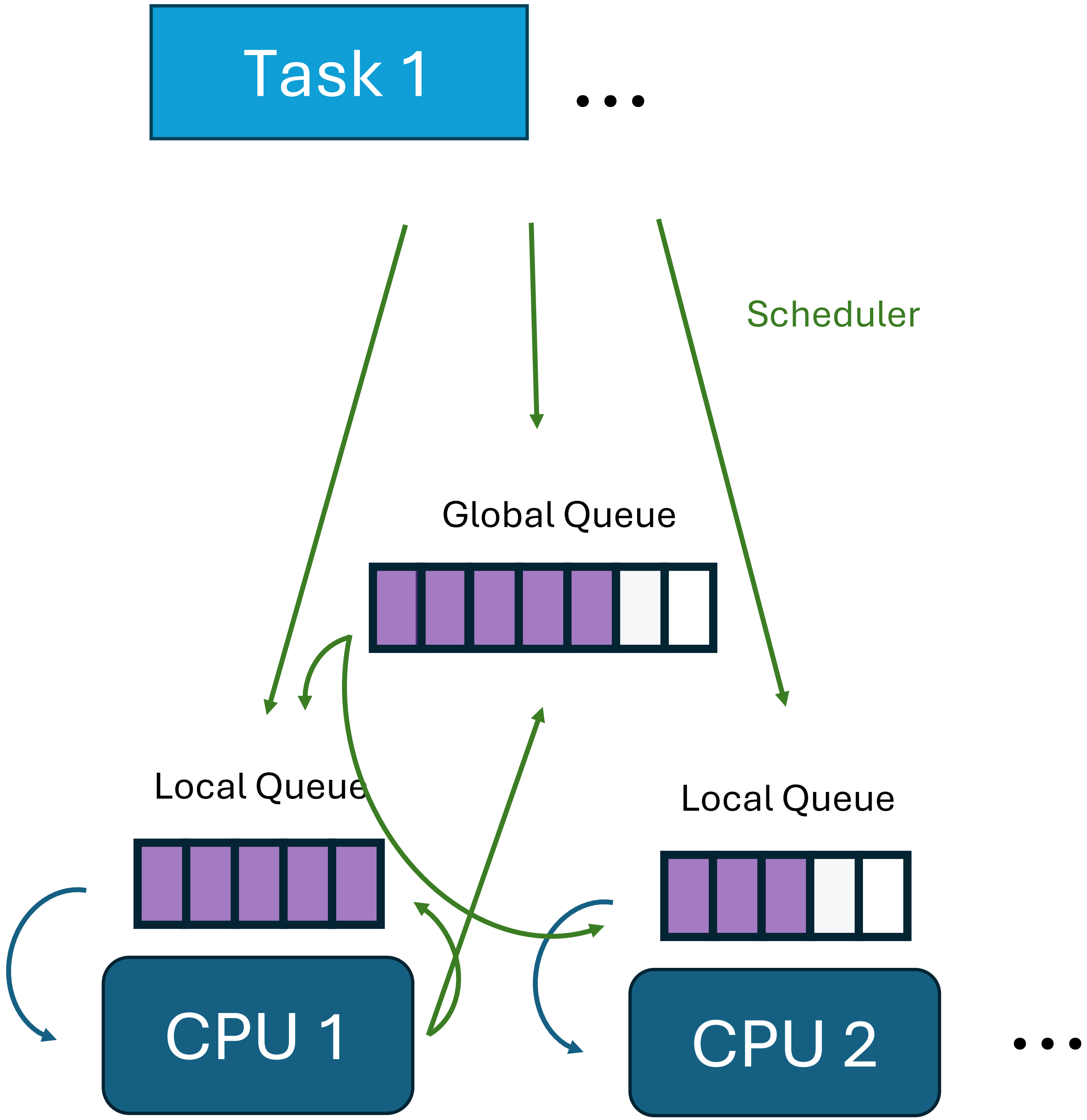
Task 2

CPU

1 2 1 1 2 1

Time

Scheduler dance

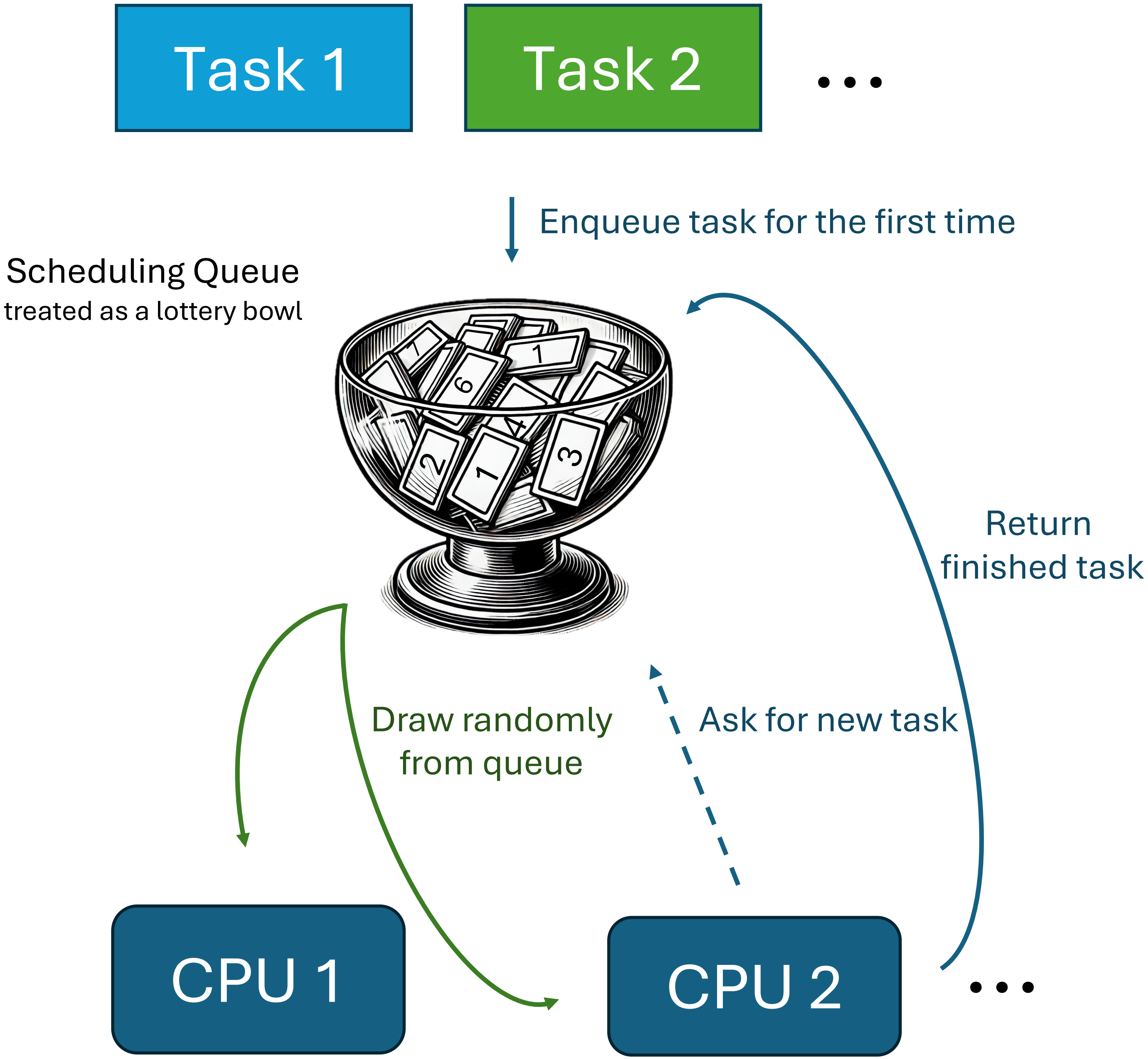


MinimalScheduler

Lottery Scheduler

Are you the lucky task who gets the time slice?

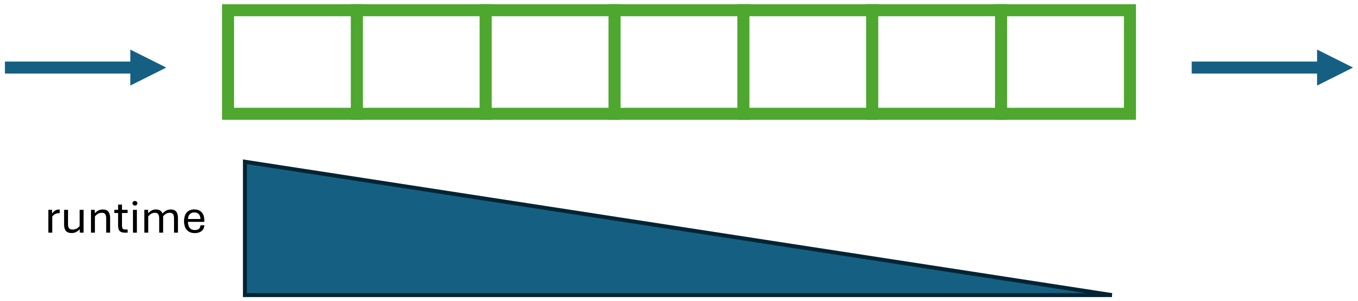
Lottery Scheduler



LotteryScheduler

VRuntime-based Scheduler

- Tracks virtual runtime (vruntime) of tasks (time on CPU)
- Task with shortest vruntime runs first
- Use a simple priority queue



VRuntime-based Scheduler

You already run quite a long time, lets choose another task

Proportional weight-based CPU allocation: fairness

- Each task T_i has a weight w_i
- The runtime assigned to each task T_i is proportional to its weight w_i divided by the sum of all the runnable tasks' weight

$$runtime(T_i) = \int_{t_0}^{t_1} \frac{w_i}{\sum_{j=0}^{N} w_j} dt \simeq \frac{w_i}{\sum_{j=0}^{N} w_j} \cdot (t_1 - t_0)$$

Andrea Righi, https://fosdem.org/2025/schedule/event/fosdem-2025-4618-level-up-your-linux-gaming-how-schedext-can-save-your-fps/

How fairness is implemented: vruntime

- Virtual runtime (vruntime)
  - Charge each task a runtime proportional to w_{base} and inversely proportional to its weight w_{i}
- Tasks are scheduled in order of increasing vruntime

$$V_{T_i}(t_1) = \frac{w_{base}}{w_i} \cdot (t_1 - t_0)$$

Andrea Righi, https://fosdem.org/2025/schedule/event/fosdem-2025-4618-level-up-your-linux-gaming-how-schedext-can-save-your-fps/

VTimeScheduler

What else can we do?

Implement good schedulers

Implement Schedulers

Typically not in Java

Implement

Schedulers

Typically not in Java

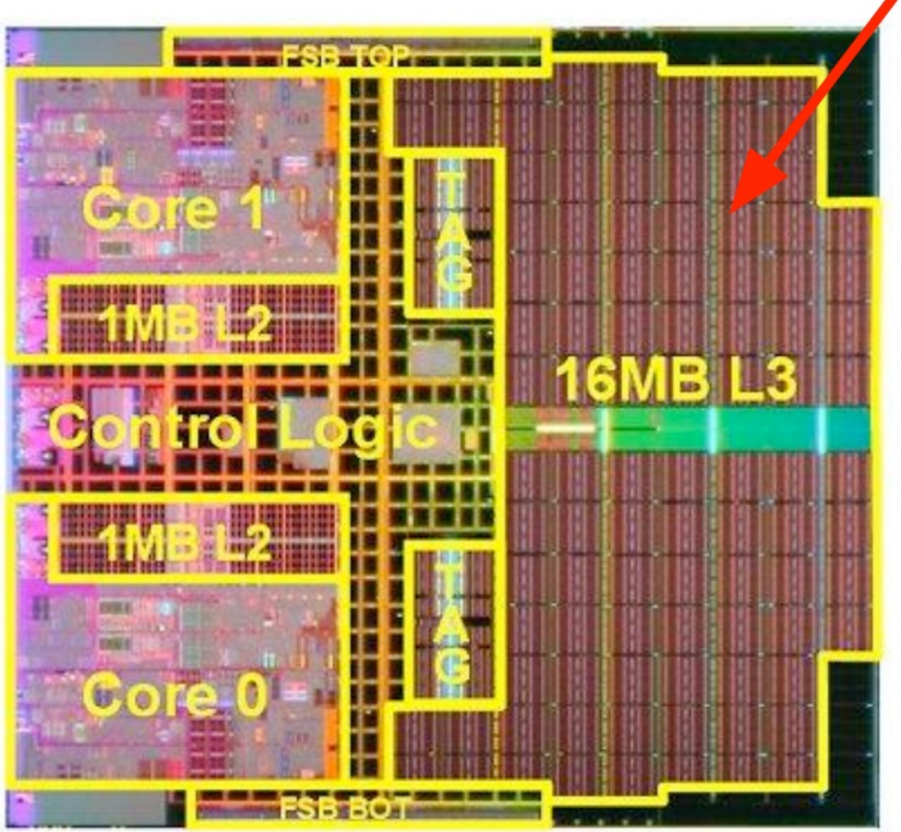
Source: Marcus Biel

CFS was built in a simpler time

- Much smaller CPUs
- Topologies much more homogeneous
- Cores spaced further apart, migration cost typically high
- Power consumption and die area wasn't as important
- The fundamental assumptions behind heuristics may be easier to justify

Just two cores

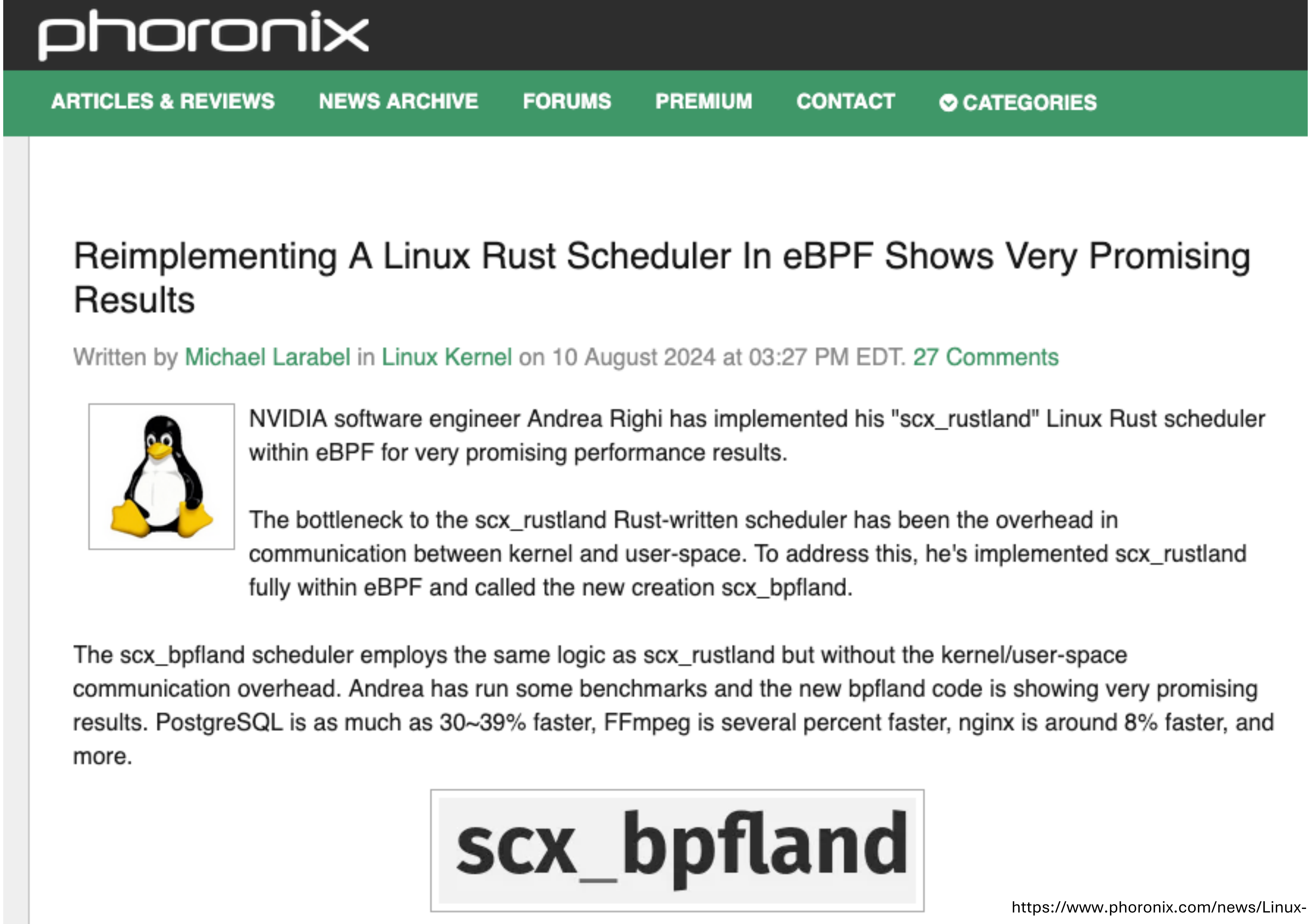
Just one L3 cache



Intel Xeon MP 71xx die

David Vernet, https://archives.kernel-recipes.org/wp-content/uploads/2025/01/Sched_Ext.pdf

https://github.com/sched-ext/scx



Reimplementing A Linux Rust Scheduler In eBPF Shows Very Promising Results

Written by Michael Larabel in Linux Kernel on 10 August 2024 at 03:27 PM EDT. 27 Comments

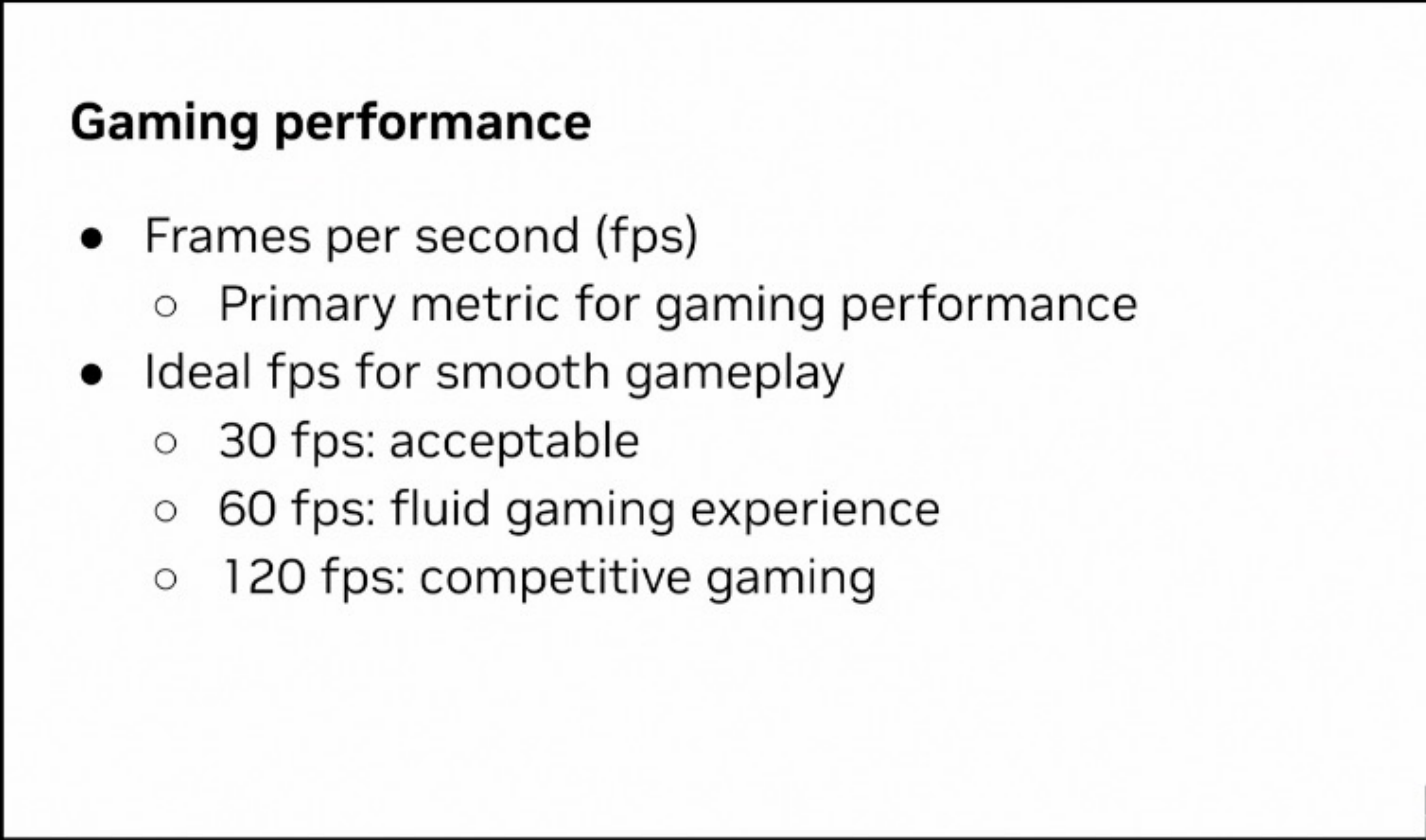
NVIDIA software engineer Andrea Righi has implemented his "scx_rustland" Linux Rust scheduler within eBPF for very promising performance results.

The bottleneck to the scx_rustland Rust-written scheduler has been the overhead in communication between kernel and user-space. To address this, he's implemented scx_rustland fully within eBPF and called the new creation scx_bpfland.

The scx_bpfland scheduler employs the same logic as scx_rustland but without the kernel/user-space communication overhead. Andrea has run some benchmarks and the new bpfland code is showing very promising results. PostgreSQL is as much as 30~39% faster, FFmpeg is several percent faster, nginx is around 8% faster, and more.

scx_bpfland

https://www.phoronix.com/news/Linux-Rust-Sched-To-eBPF



Gaming performance

- Frames per second (fps)
  - Primary metric for gaming performance
- Ideal fps for smooth gameplay
  - 30 fps: acceptable
  - 60 fps: fluid gaming experience
  - 120 fps: competitive gaming

Experiments

An erratic scheduler

https://lwn.net/SubscriberLink/1007689/922423e440f5e68a/

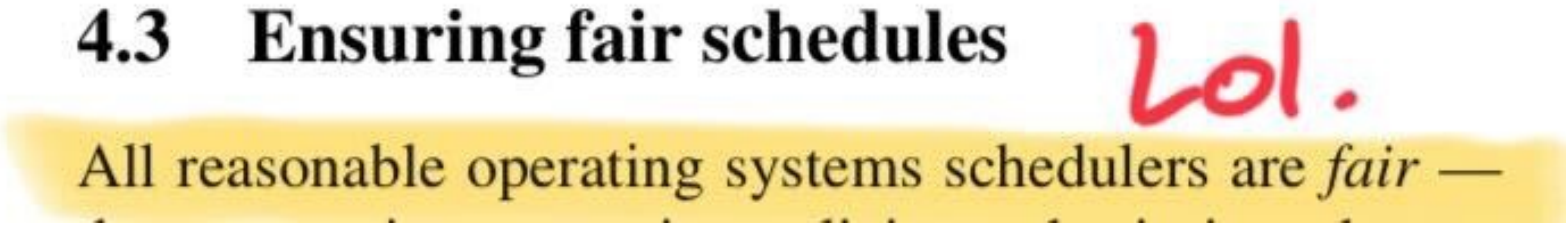
An erratic scheduler

Written in Java

https://lwn.net/SubscriberLink/1007689/922423e440f5e68a/

Experiments

Having fun
with sched-ext



4.3 Ensuring fair schedules

Lol.

All reasonable operating systems schedulers are *fair* —

Paper: https://www.microsoft.com/en-us/research/wp-content/uploads/2016/02/tr-2007-149.pdf

One that produces sound

https://github.com/parttimenerd/loudness-scheduler

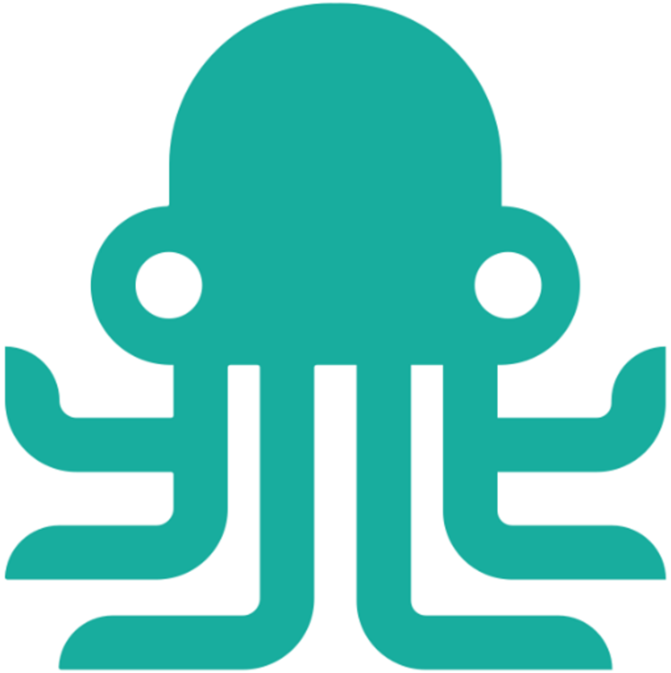
One that reacts to sound

https://github.com/parttimenerd/sound-of-scheduling

TaskClicker

https://github.com/Mr-Pine/taskclicker

Winner of the



Scheduler Contest

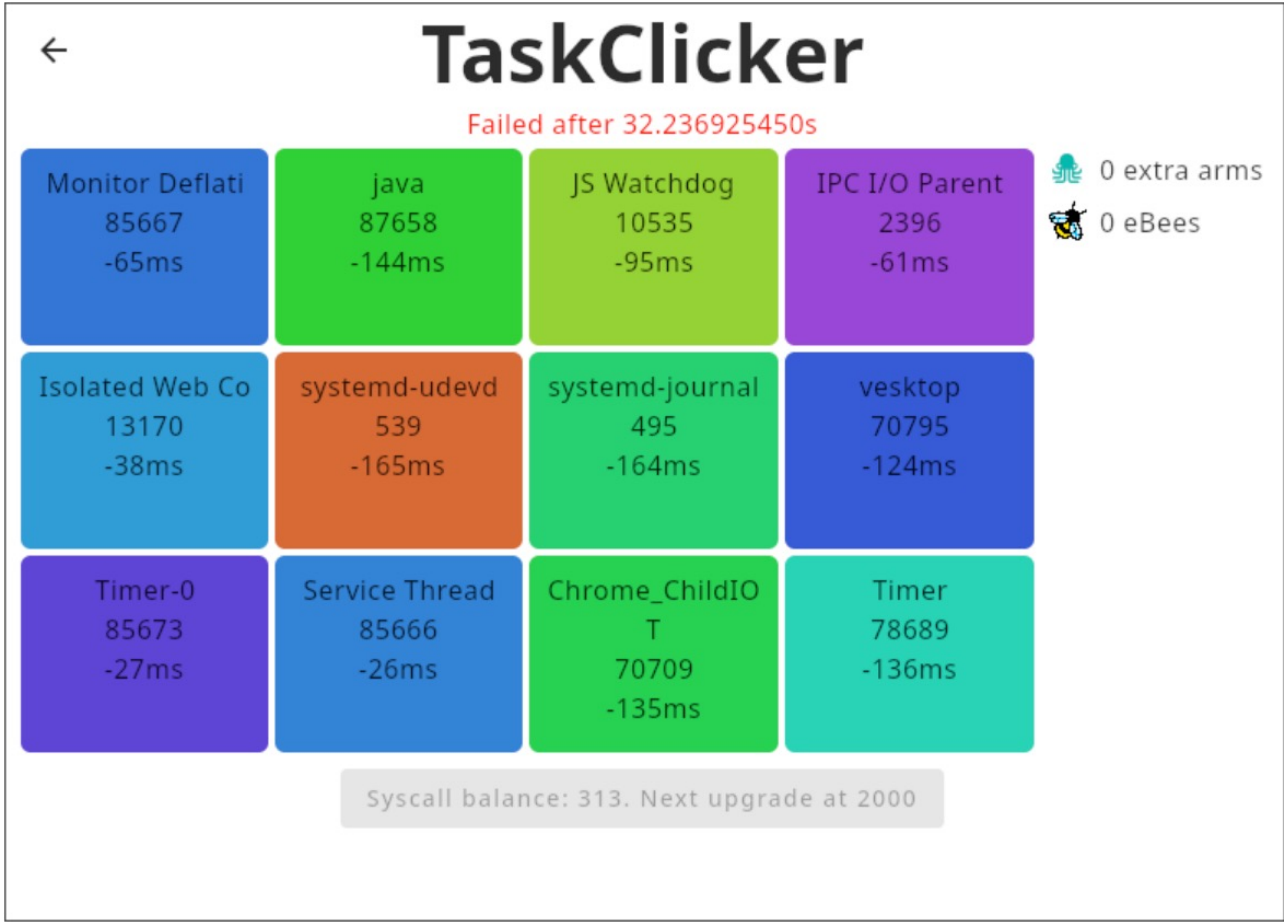
Submit your best scheduling ideas and implementations

KIT'25

Interactive,
First Come , First Served
Scheduler

Interactive, First Clicked, First Served Scheduler

The First Idle Game Scheduler



←

TaskClicker

Failed after 32.236925450s

Monitor Deflati 85667 -65ms	java 87658 -144ms	JS Watchdog 10535 -95ms	IPC I/O Parent 2396 -61ms
Isolated Web Co 13170 -38ms	systemd-udevd 539 -165ms	systemd-journal 495 -164ms	vesktop 70795 -124ms
Timer-0 85673 -27ms	Service Thread 85666 -26ms	Chrome_ChildIO T 70709 -135ms	Timer 78689 -136ms

0 extra arms

0 eBees

Syscall balance: 313. Next upgrade at 2000

Fin.

github.com/parttimenerd/hello-ebpf

Johannes Bechberger
mostlynerdless.de
OpenJDK Developer, SAP

David Kiefer
mr-pine.de
Student, KIT

timetableworld.com