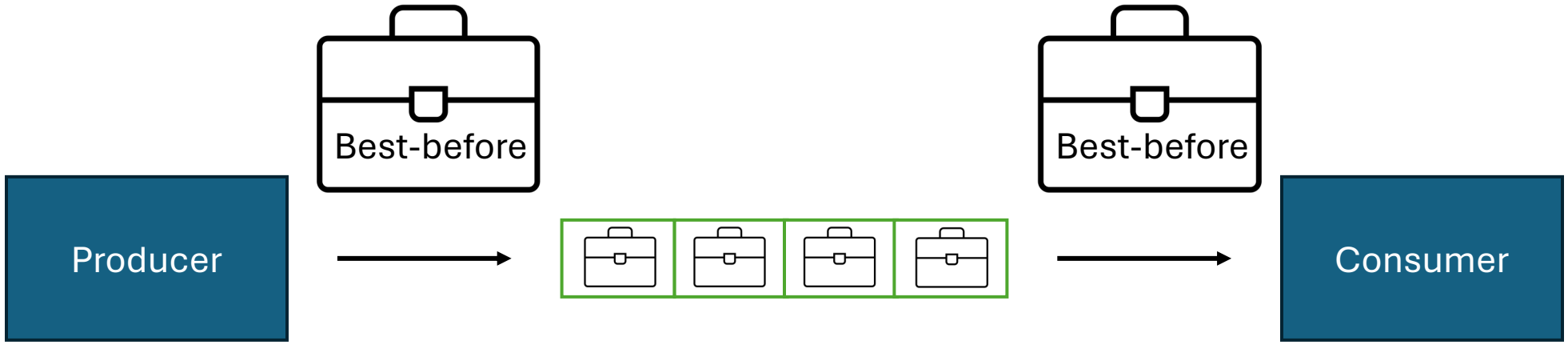
Concurrency Testing using Custom Linux Schedulers

Johannes Bechberger and Jake Hillion

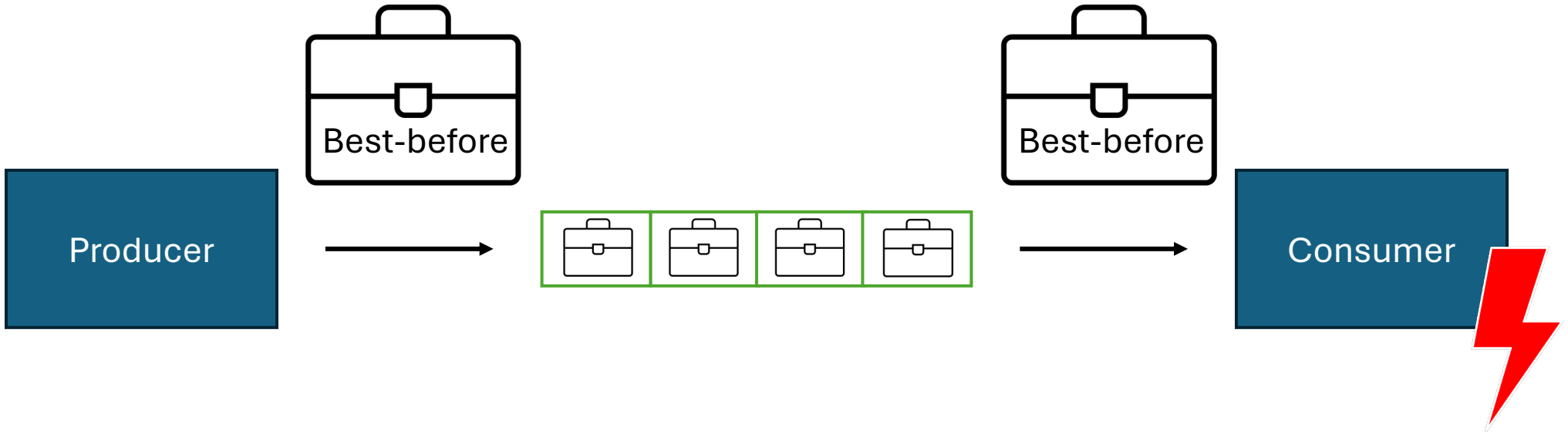
timetableworld.com

Heisenbugs

Consider the following

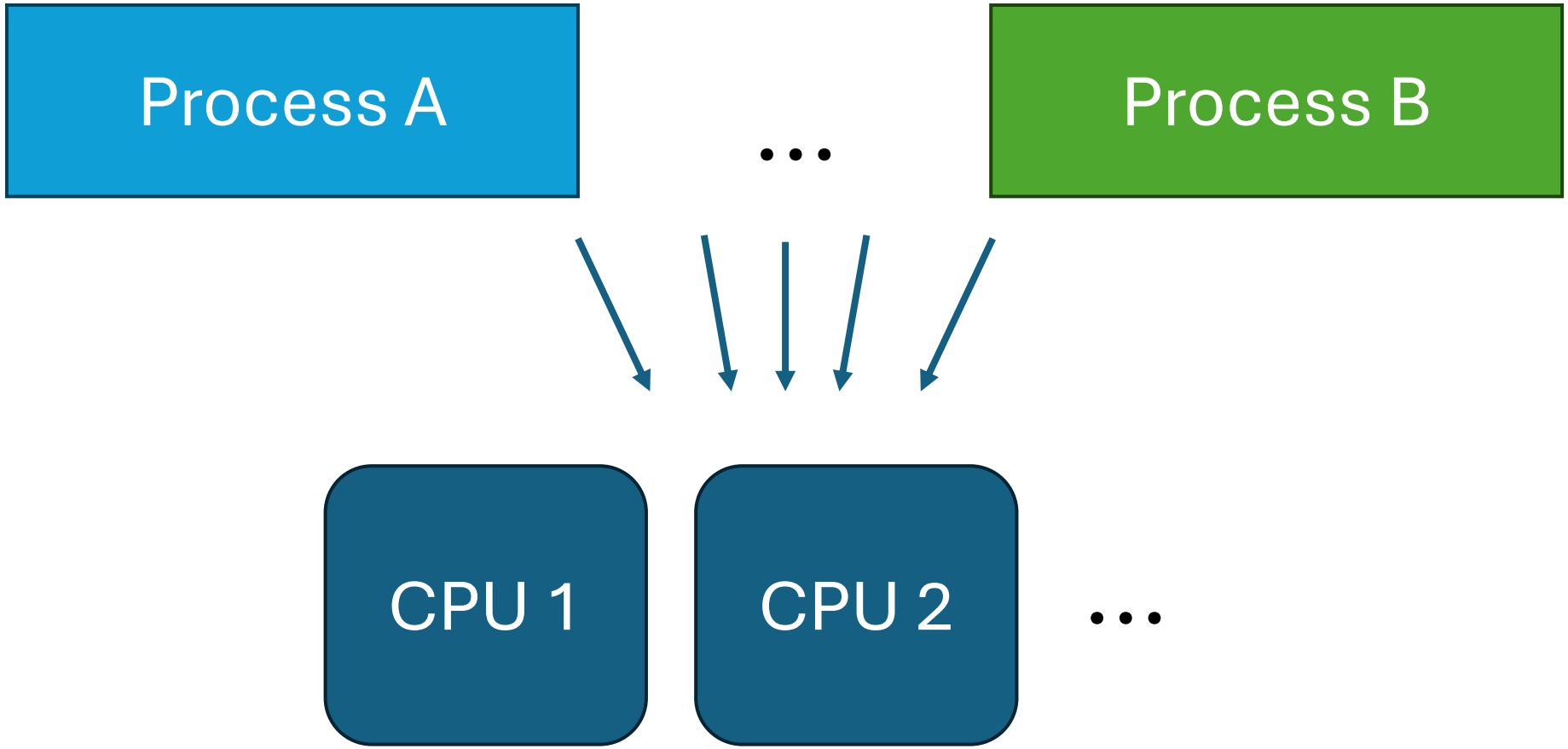


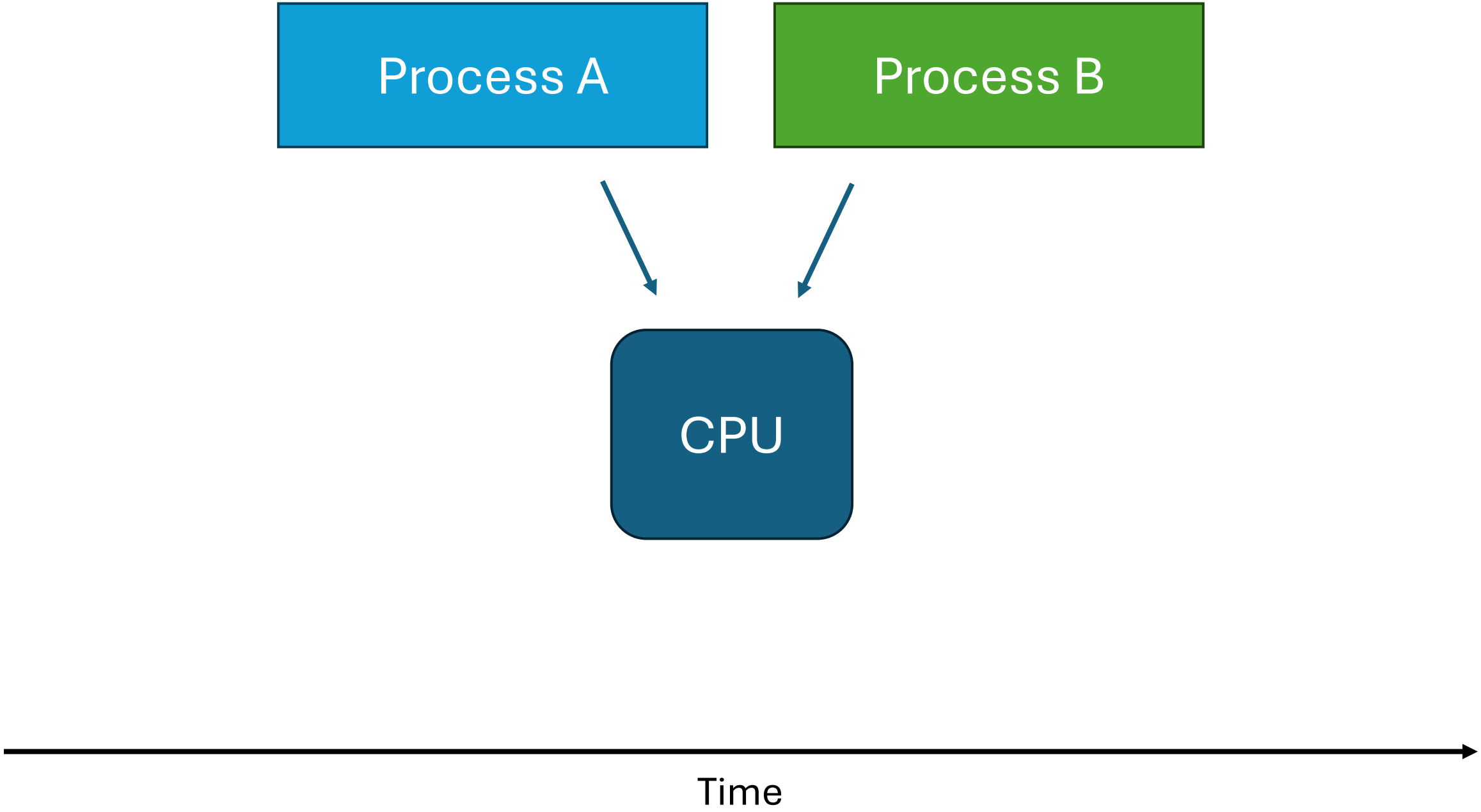
This never crashes except when it does

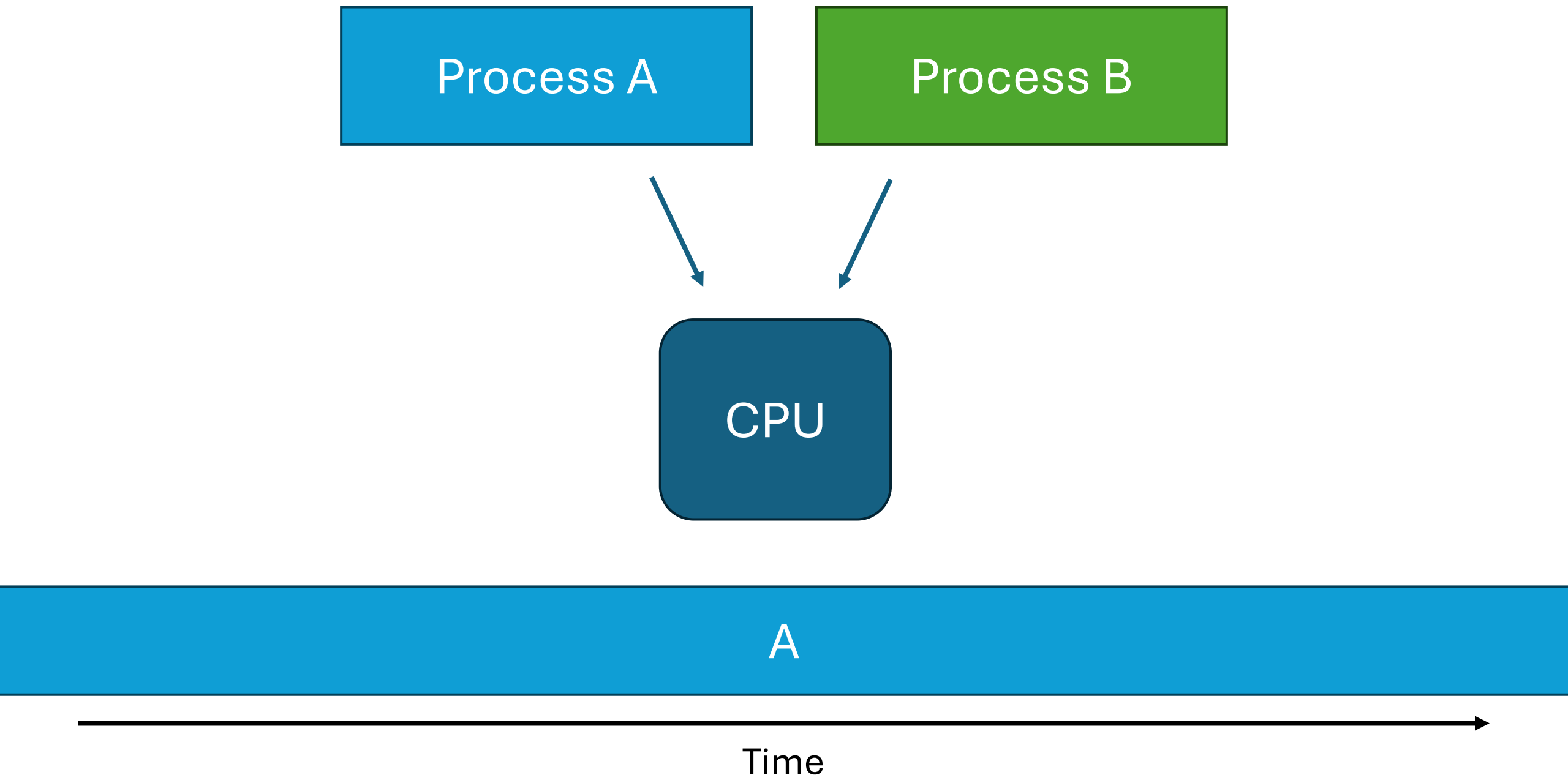


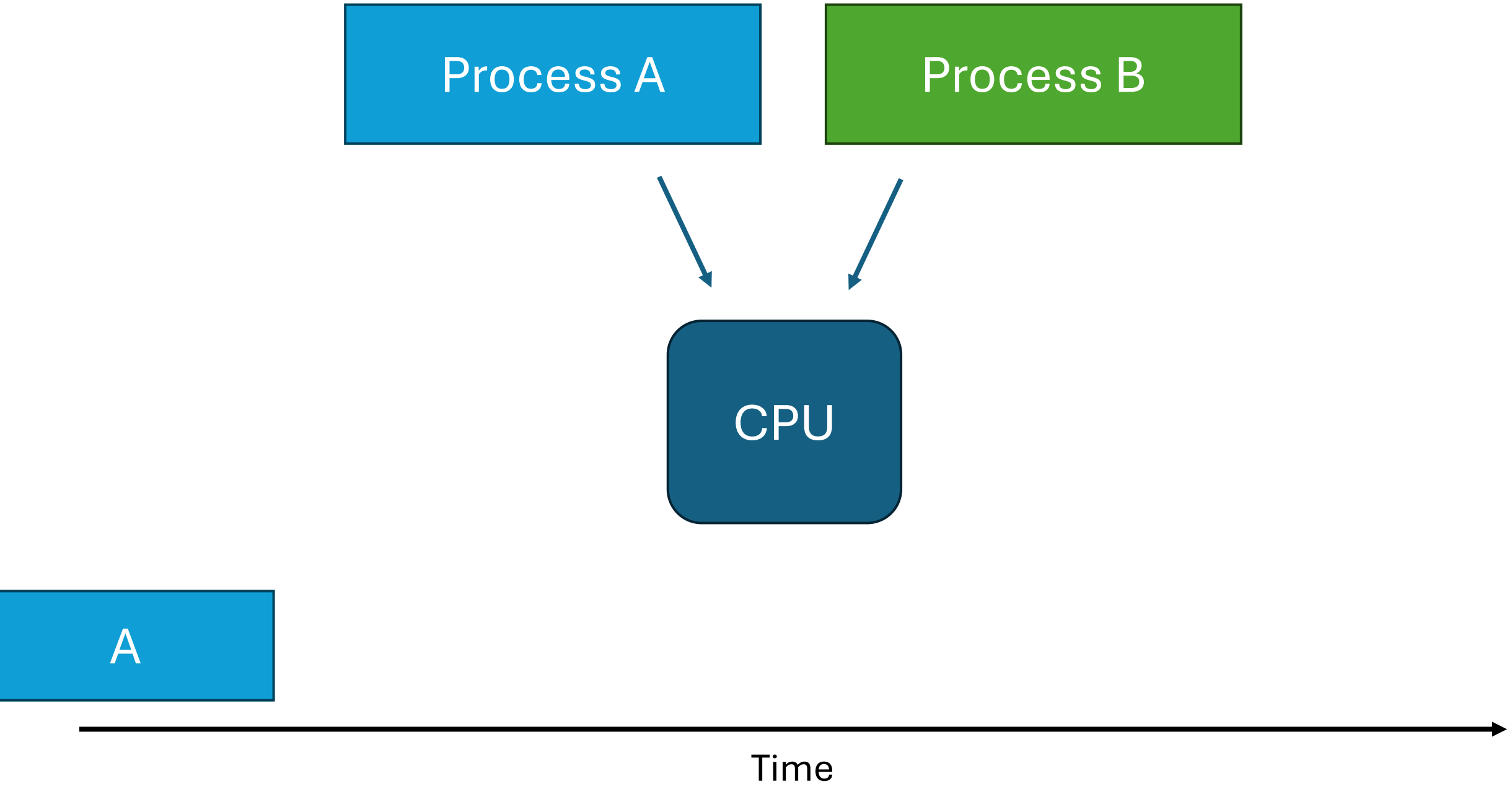
This is related to scheduling

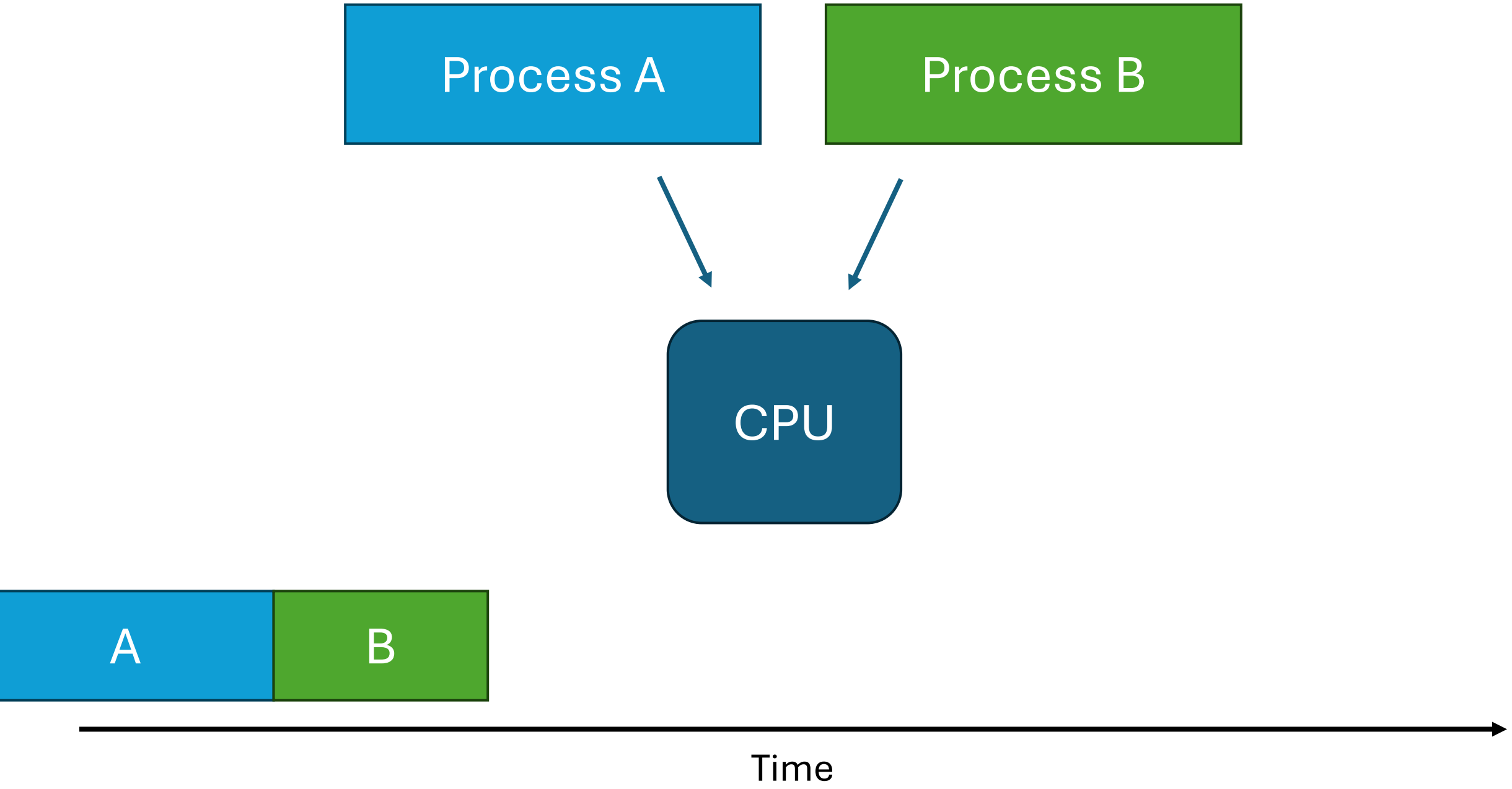
What is scheduling?

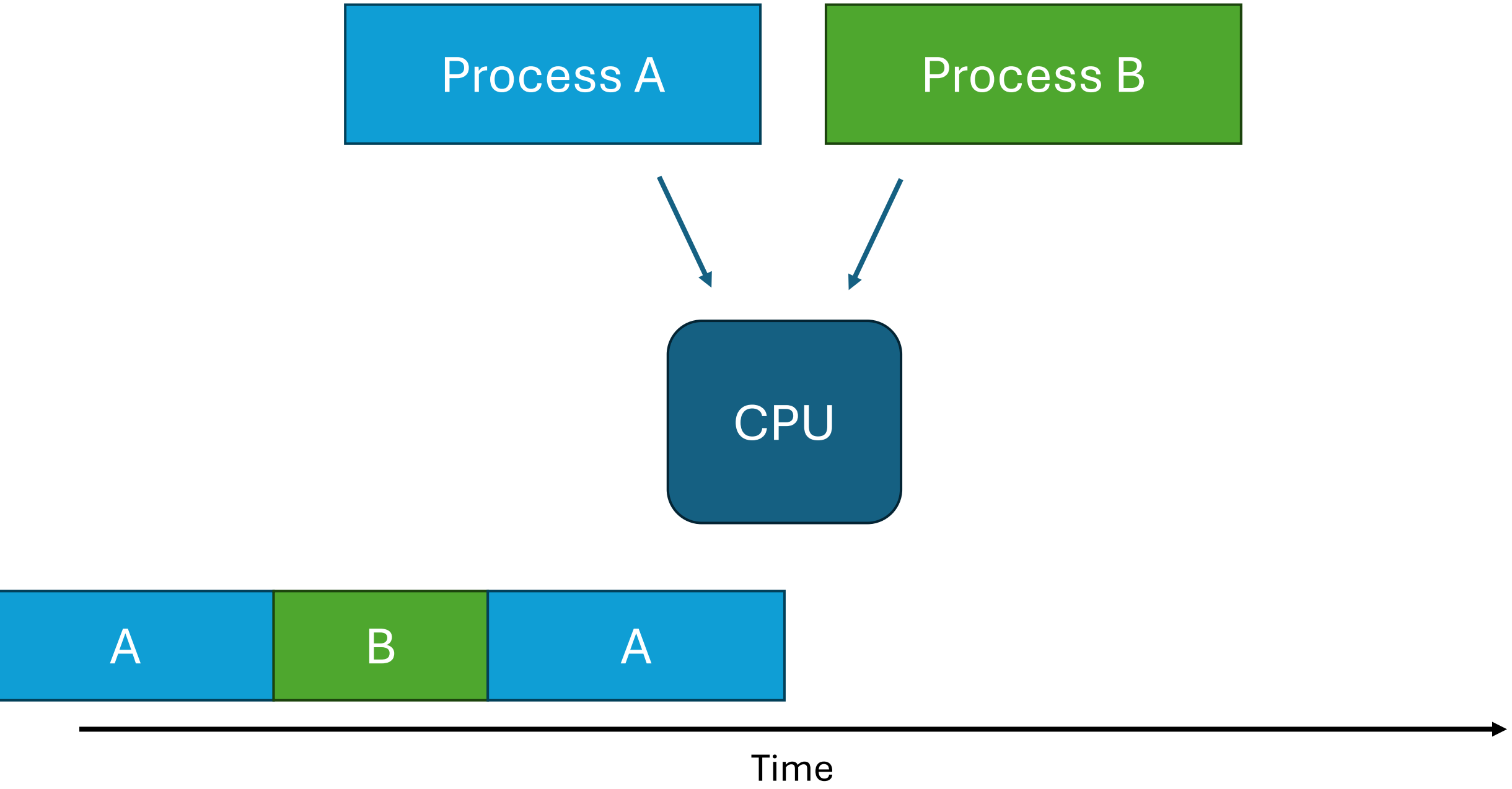

Process A
...
Process B
CPU 1
CPU 2
...

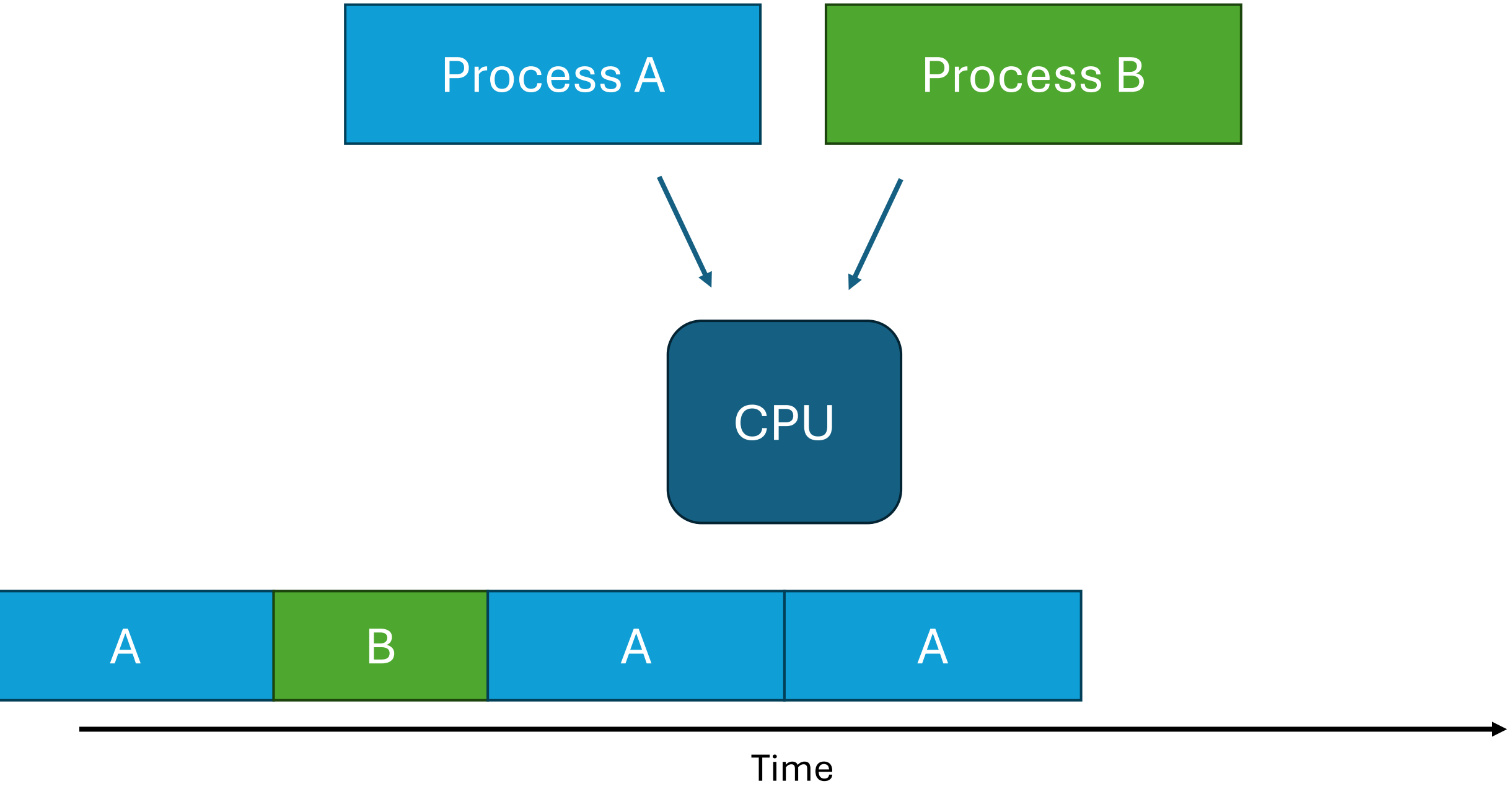

Process A
Process B
CPU
Time

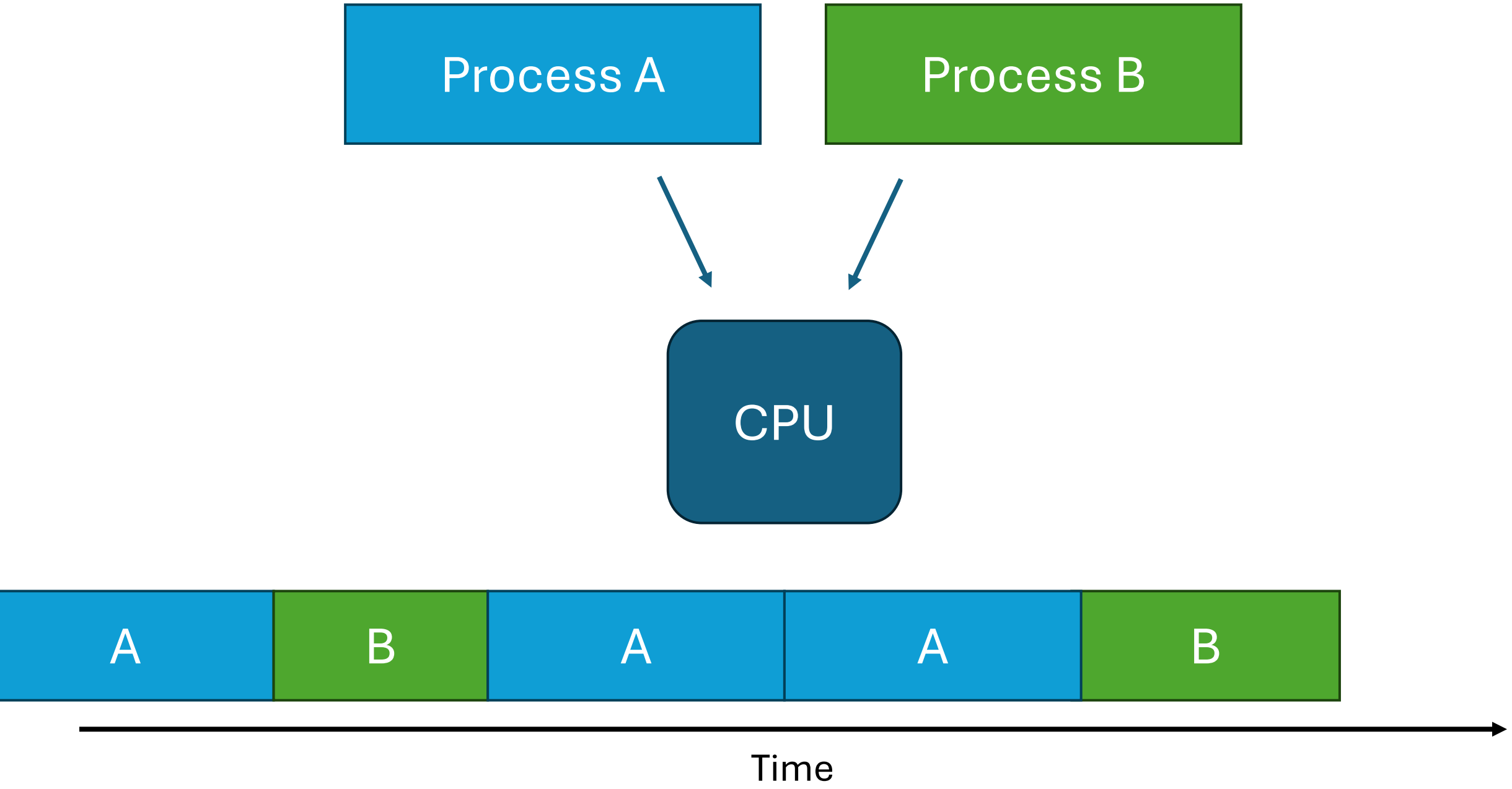

Process A
Process B
CPU
A
Time

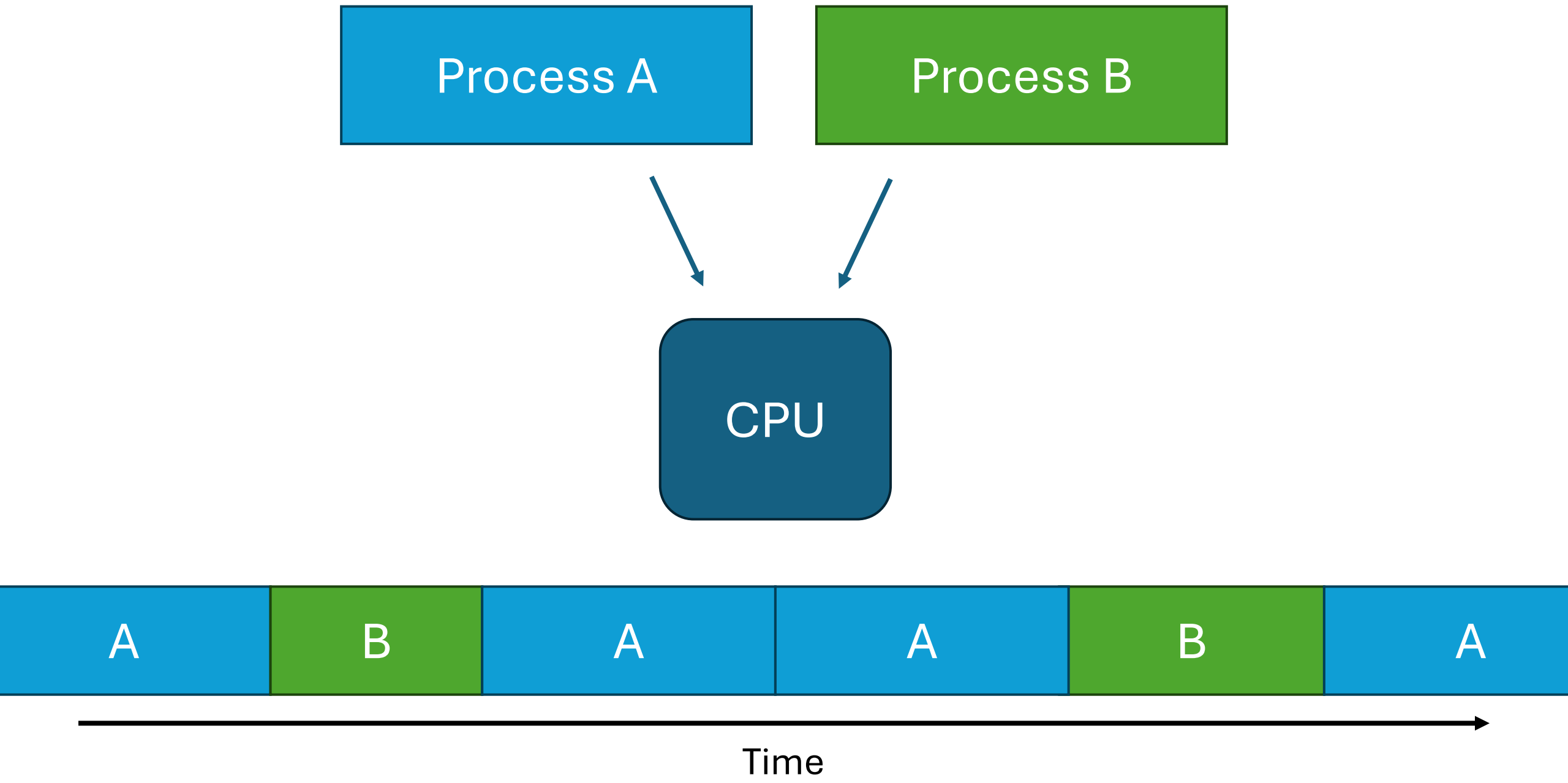

Process A
Process B
CPU
A
Time


Process A
Process B
CPU
A
B
Time


Process A
Process B
CPU
A
B
A
Time


Process A
Process B
CPU
A
B
A
A
Time


Process A
Process B
CPU
A
B
A
A
B
Time


Process A
Process B
CPU
A
B
A
A
B
A
Time

Default Workspace
Cpu 0
Cpu 1
Cpu 2
Cpu 3
Cpu 4
Cpu 5
Cpu 6
Cpu 7
Cpu 8
Cpu 9
Cpu 10
Cpu 11
Cpu 12
Cpu 13
Cpu 14
Cpu 15
6us 960ns

Standard schedulers are too good

We need an erratic scheduler!!!

Let's create our own

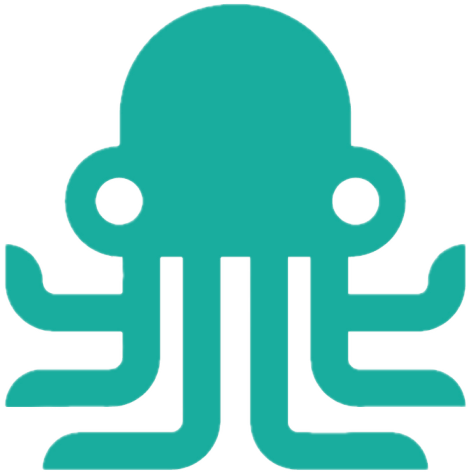
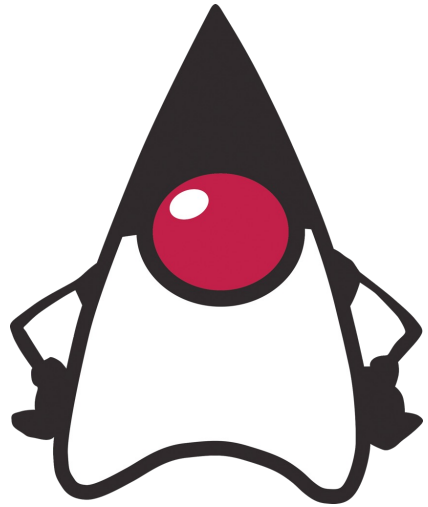
Let's create our own

without Kernel Development

Let's create our own

in Java (or Rust)

Let's create our own

+  + 

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

Brendan Gregg

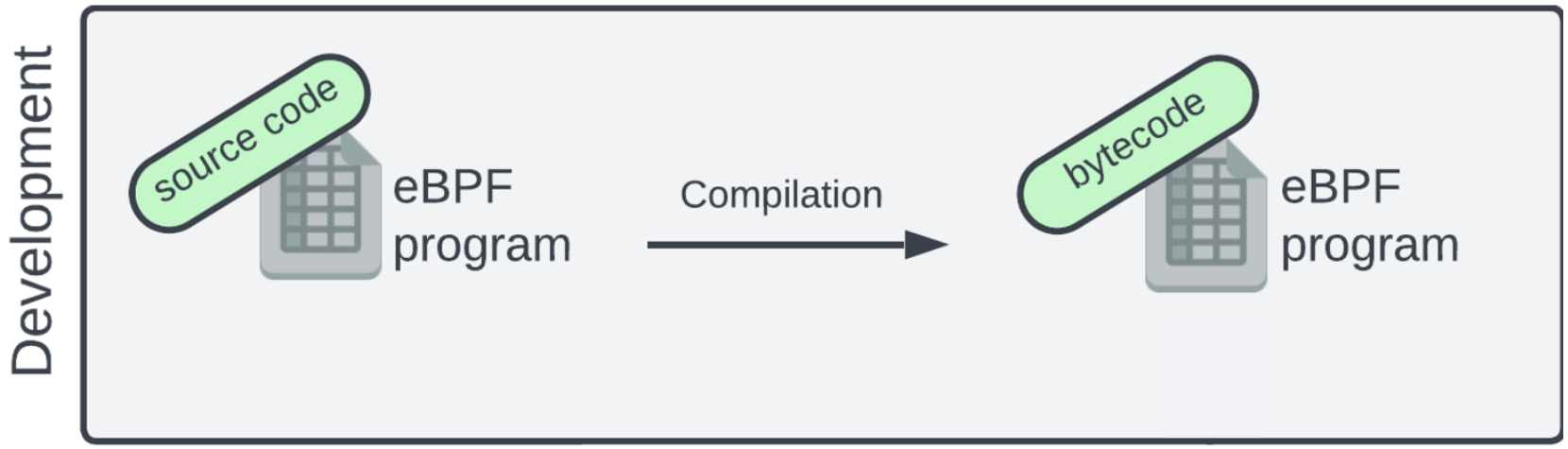
https://www.youtube.com/watch?v=tDacjrSCeq4

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

Brendan Gregg

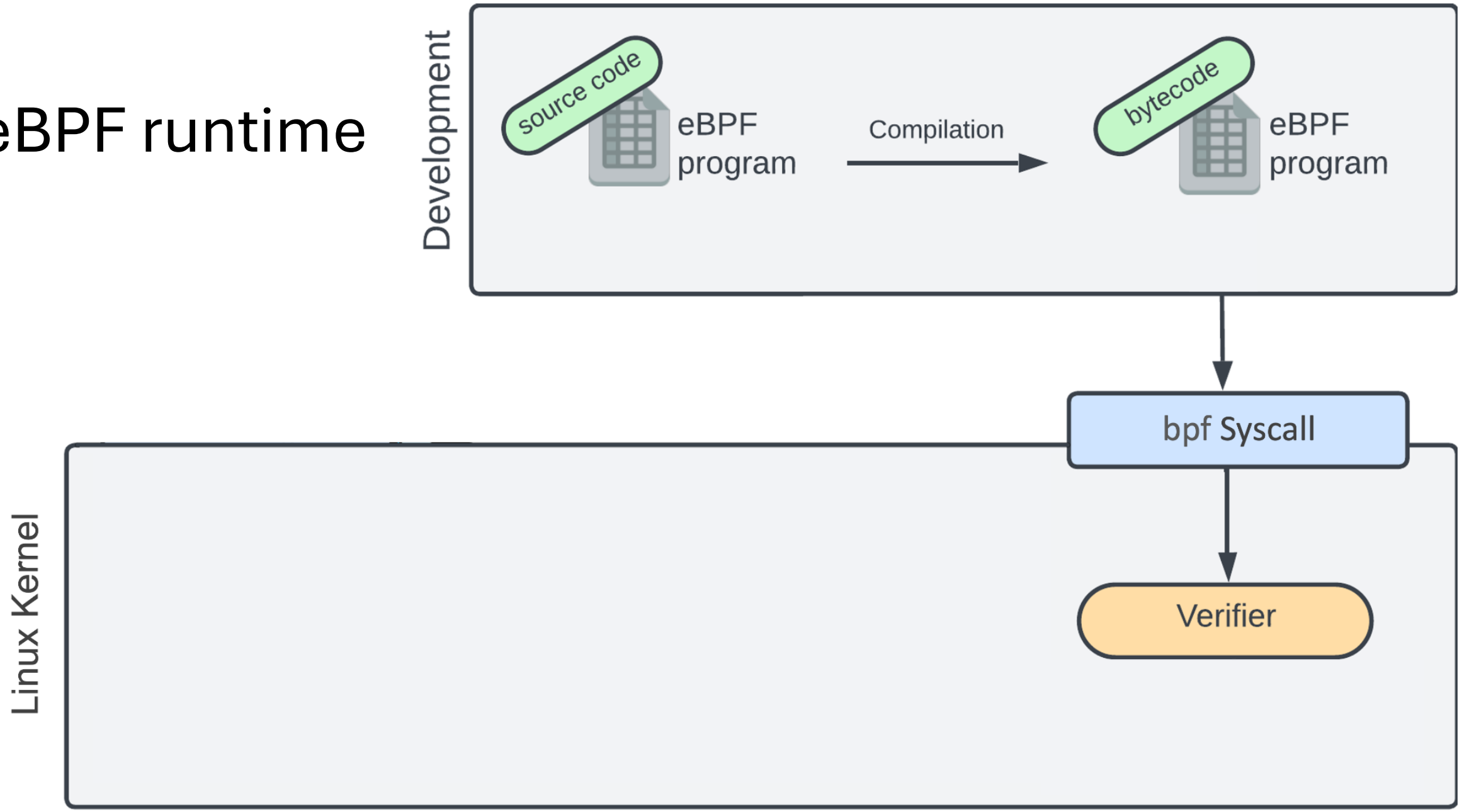
https://www.facesofopensource.com/brendan-gregg/

eBPF runtime

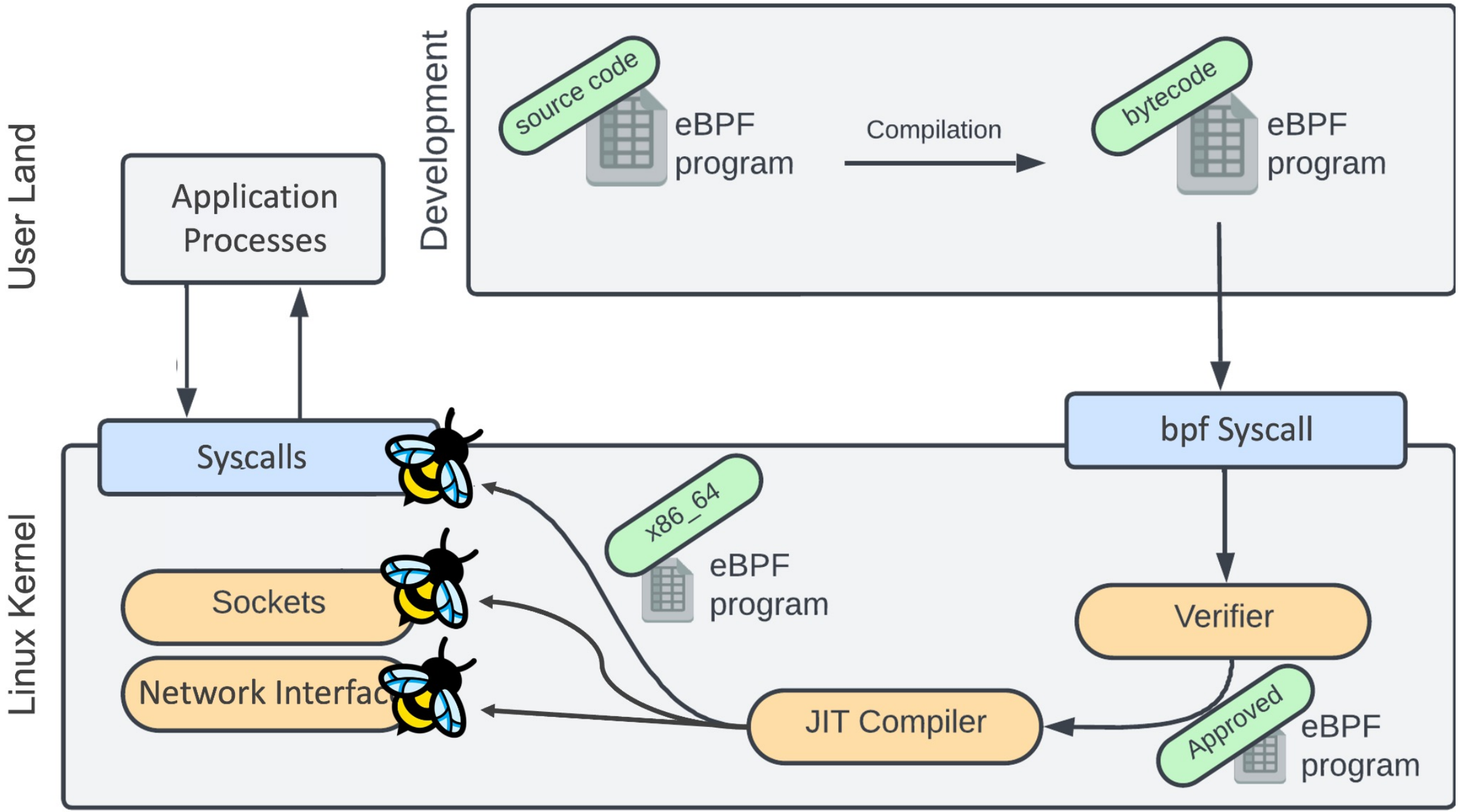


Courtesy of Mohammed Aboullaite

eBPF runtime

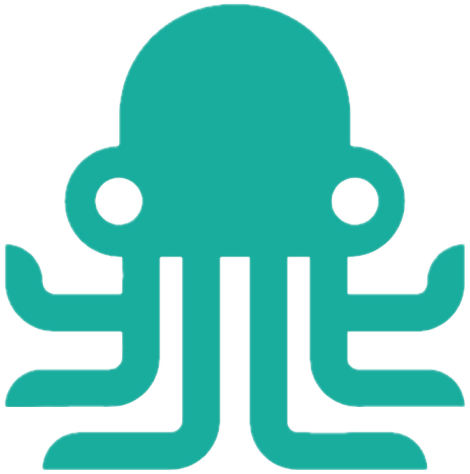


Courtesy of Mohammed Aboullaite



Courtesy of Mohammed Aboullaite

Here comes sched-ext

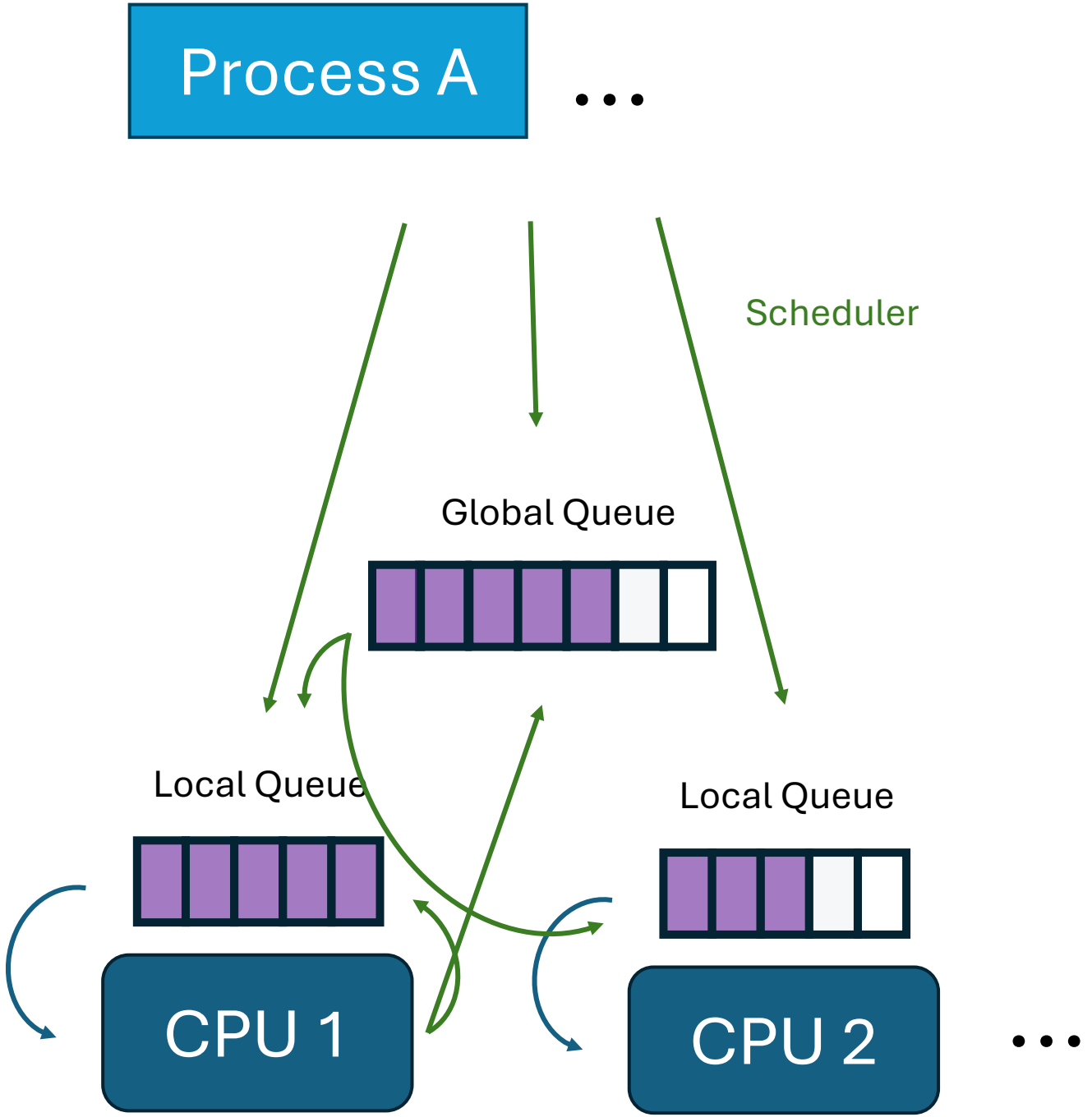


“

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

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

Scheduler dance



```
@BPF(license = "GPL")
public abstract class SampleScheduler
  extends BPFProgram implements Scheduler, Runnable {

  static final long SHARED_DSQ_ID = 0;

  @Override
  public int init() {
    return scx_bpf_create_dsq(SHARED_DSQ_ID, -1);
  }

}
```

```
@BPF(license = "GPL")
public abstract class SampleScheduler {

  @Override
  public void enqueue(Ptr<task_struct> p,
    long enq_flags) {
    scx_bpf_dispatch(p, SHARED_DSQ_ID,
      5_000_000, enq_flags);
  }


}
```

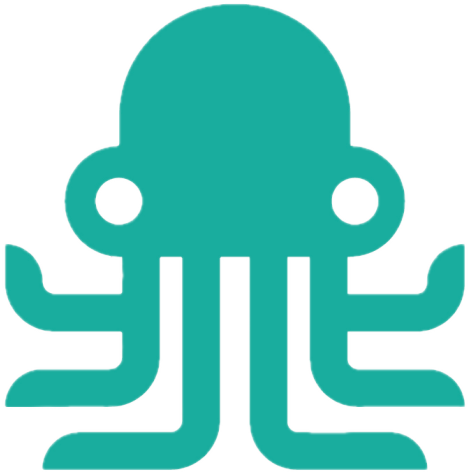
```
@BPF(license = "GPL")
public abstract class SampleScheduler {

  @Override
  public void dispatch(int cpu,
    Ptr<task_struct> prev) {
    scx_bpf_consume(SHARED_DSQ_ID);
  }



}
```

Produce erratic scheduling orders



Demo

https://github.com/parttimenerd/concurrency-fuzz-scheduler

```
var producer = new Thread(() -> {
    while (true) {
        produce((int) (Math.random() * 100));
        sleep(20);
    }
});
producer.setName("Producer");
producer.start();
var consumer = new Thread(() -> {
    while (true) {
        consume();
        sleep(10);
    }
});
consumer.setName("Consumer");
consumer.start();
```

Demo

Highly Experimental

https://github.com/parttimenerd/concurrency-fuzz-scheduler

Fin.

Johannes Bechberger
mostlynerdless.de
OpenJDK Developer, SAP

concurrency-fuzz-scheduler

Jake Hillion
hillion.co.uk
scx Developer, Meta

timetableworld.com