Towards enhanced performance and energy savings for new Cloud services

DJOB MVONDO

UNIVERSITY OF RENNES 1-IRISA-INRIA, FRANCE

21-09-2022

CONTEXT : FUNCTION AS A SERVICE CLOUD MODEL

Serverless cloud model is gaining a lot of traction.

~ 22 Billion $ estimated by 2025[1]

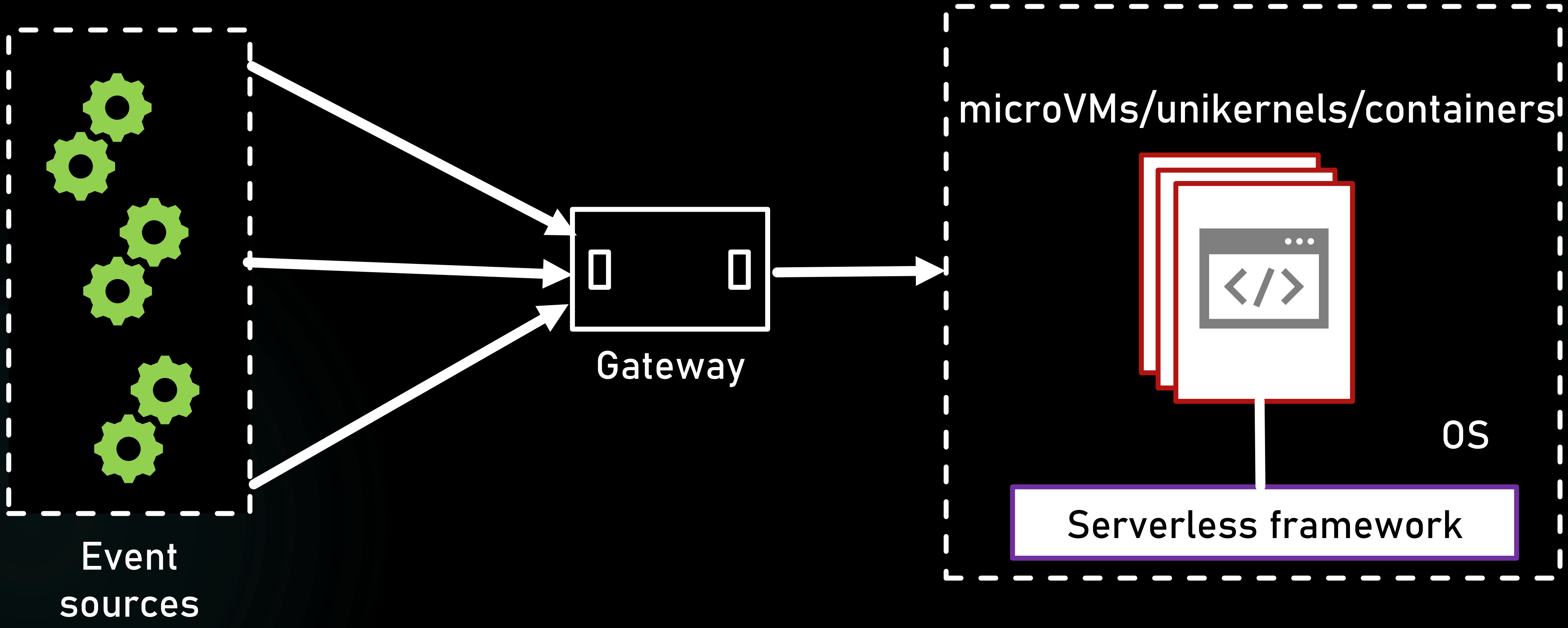
Amazon Lambda Google Functions Azure Functions

...

[1] https://www.alliedmarketresearch.com/serverless-architecture-market

CONTEXT: FUNCTION AS A SERVICE CLOUD MODEL

Developers send the code and configures the events/trigger



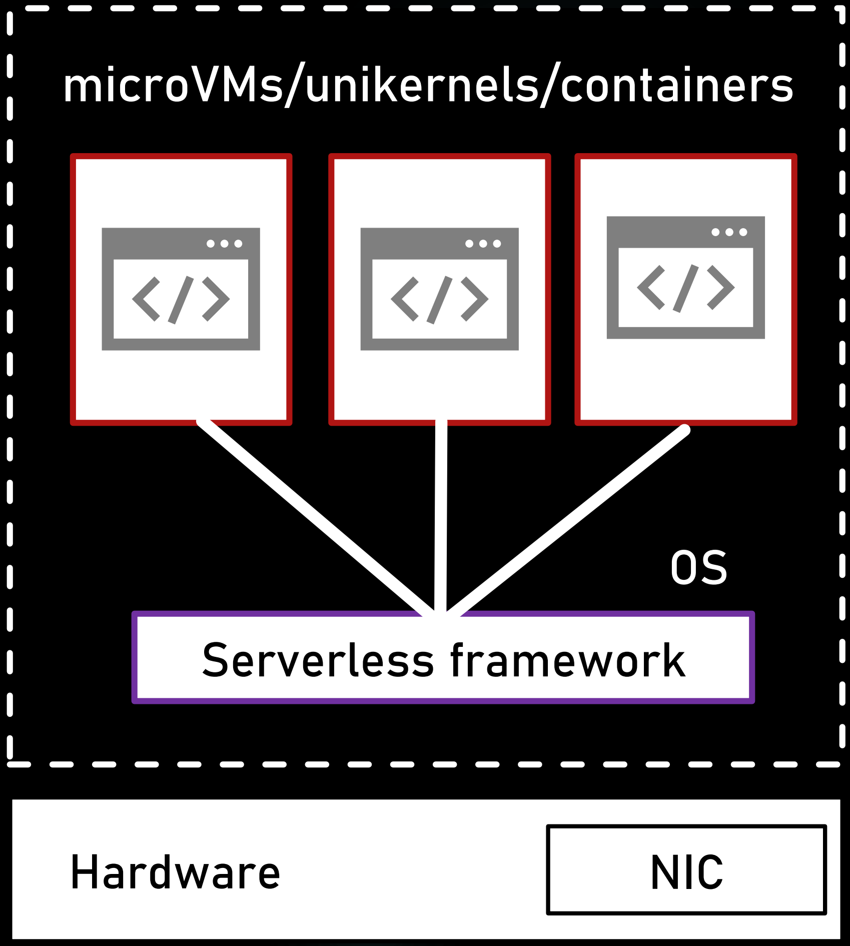
+ Billed based on execution time and memory used.

+ Focus on your code and leave the rest to the provider

CONTEXT : FUNCTION AS A SERVICE CLOUD MODEL

The cloud scheduler will direct the request to a server to launch the isolation units.

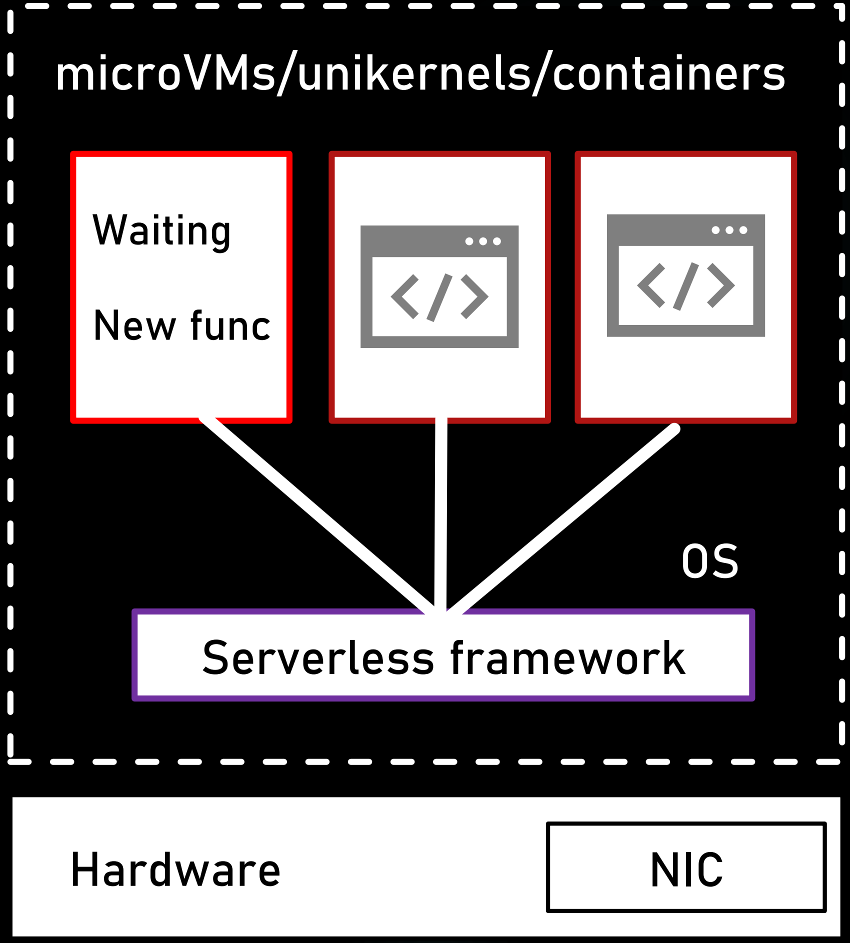
On each server, the OS scheduler must ensures fair sharing of CPU time for every isolation unit



PROBLEM: IDLE ISOLATION UNITS

However, some isolation units may be idle

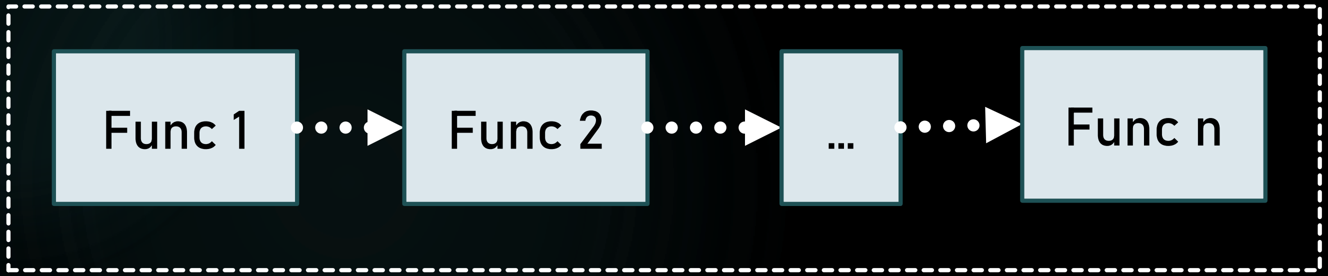
- Keep alive policy to reduce functions' start-up time.



PROBLEM : IDLE ISOLATION UNITS

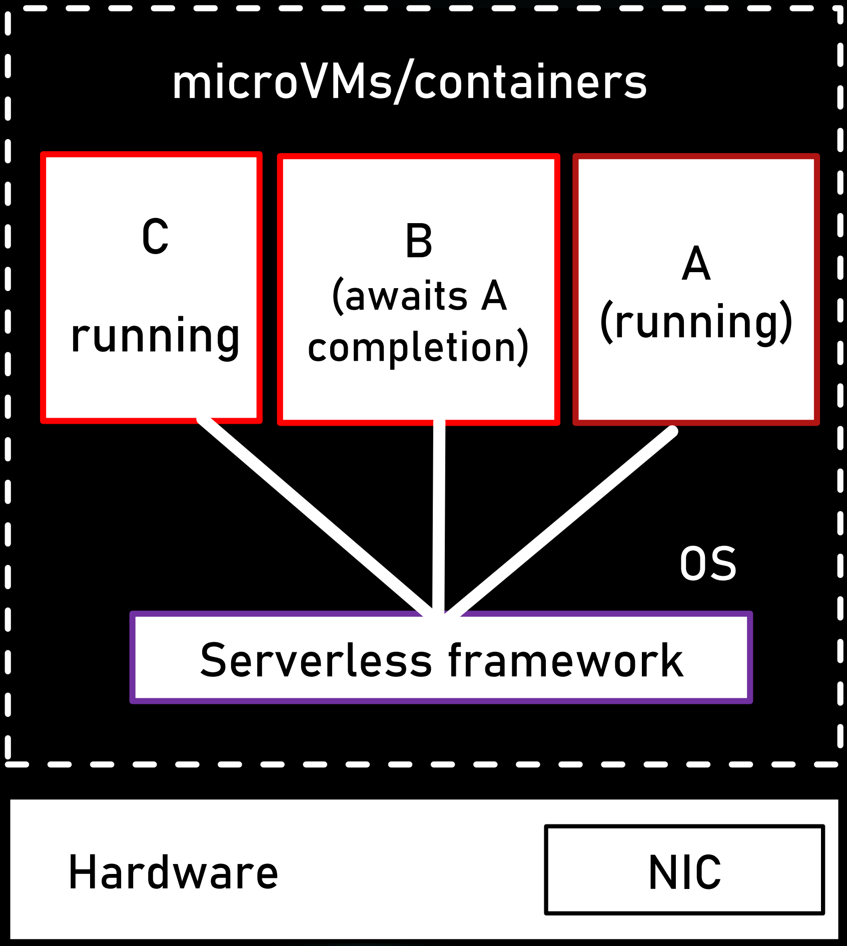
However, some isolation units may be idle

- Functions awaiting inputs from other functions



Sequence of functions

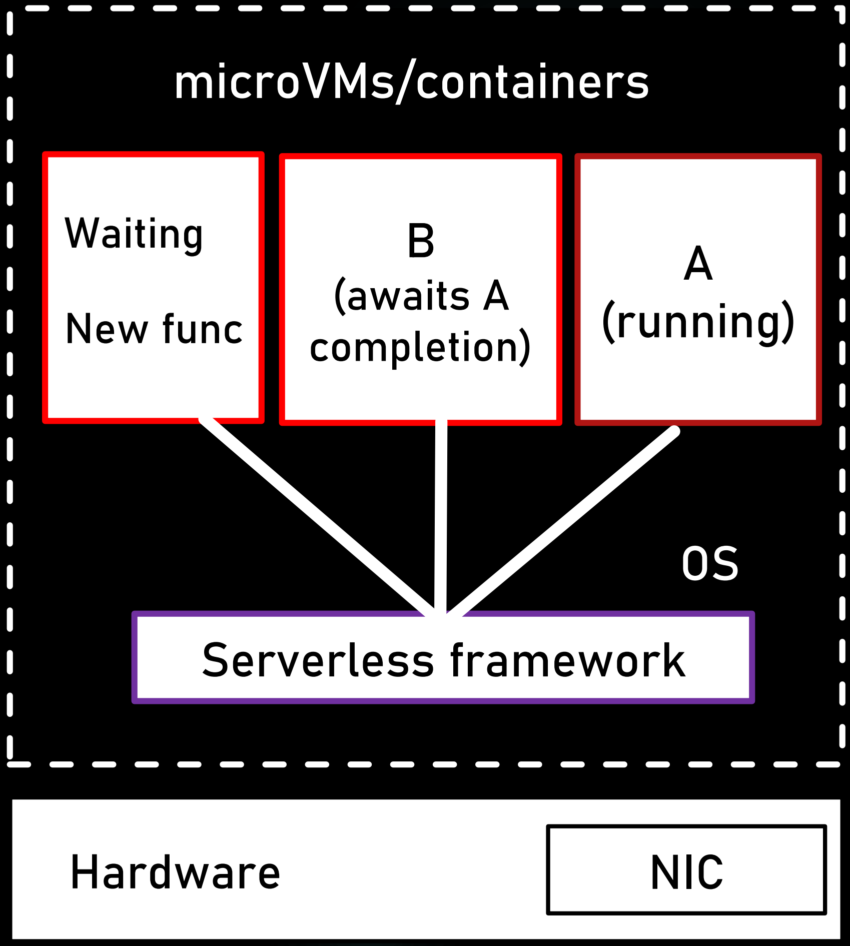
All isolation units are triggered at the same time to reduce cold latencies



PROBLEM: IDLE ISOLATION UNITS

The isolation units idleness raises two main issues

1. Wasted CPU time
2. Incorrect accounting misleading CPU frequency variations



PROBLEM 1: WASTED CPU TIME ON IDLE ISOLATION UNITS

For a sequence of 3 functions, Func{1,2,3}.

Func{2,3} isolation units are initialized but await func1 completion

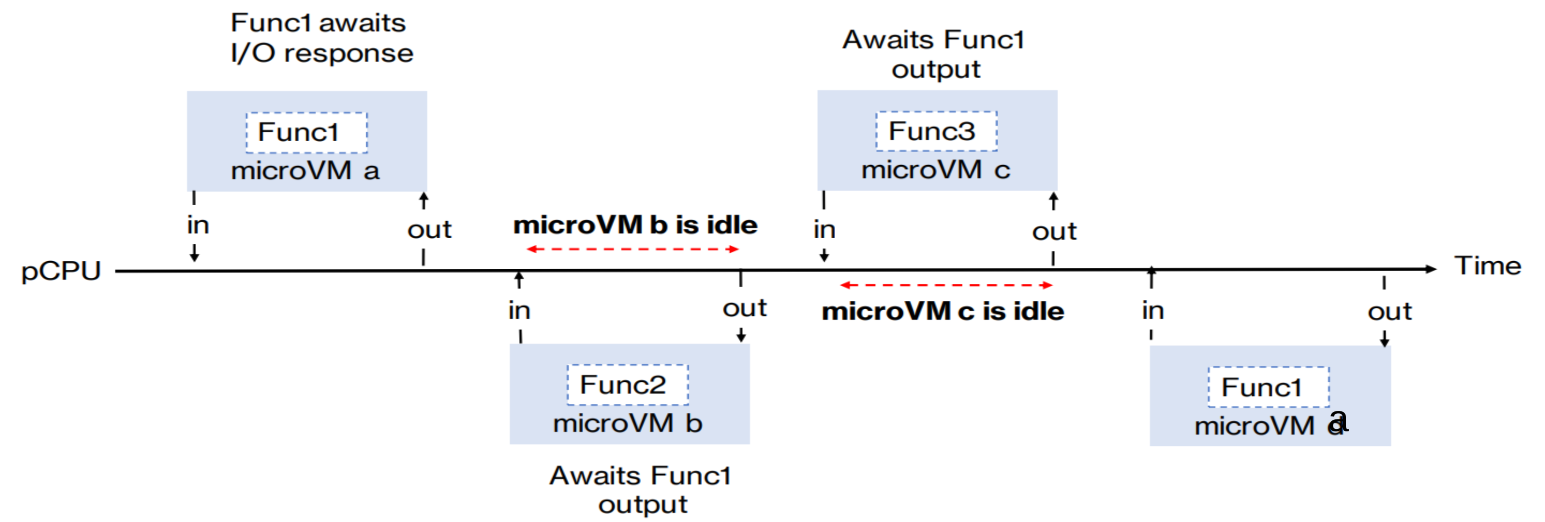


Figure 1. *Illustration of micro-VMs idle times. Micro-VMs b and c running Func2 and Func3 respectively, are scheduled even though they await Func1 output which has not finished running. This results in wasted CPU time.*

PROBLEM 1: WASTED CPU TIME ON IDLE ISOLATION UNITS

We analyzed the wasted CPU time on idle isolation units.

In-lab setup and ec2 a1.metal with Firecracker[2]

Triggering up to 50 pipelines image processing functions

Inputs and outputs images stored in AWS S3

We compute isolation units idle CPU usage

[2] Alexandru Agache et al. Firecracker: Lightweight Virtualization for Serverless Applications NSDI'20

PROBLEM: WASTED CPU TIME ON IDLE ISOLATION UNITS

Avg Pipeline execution time

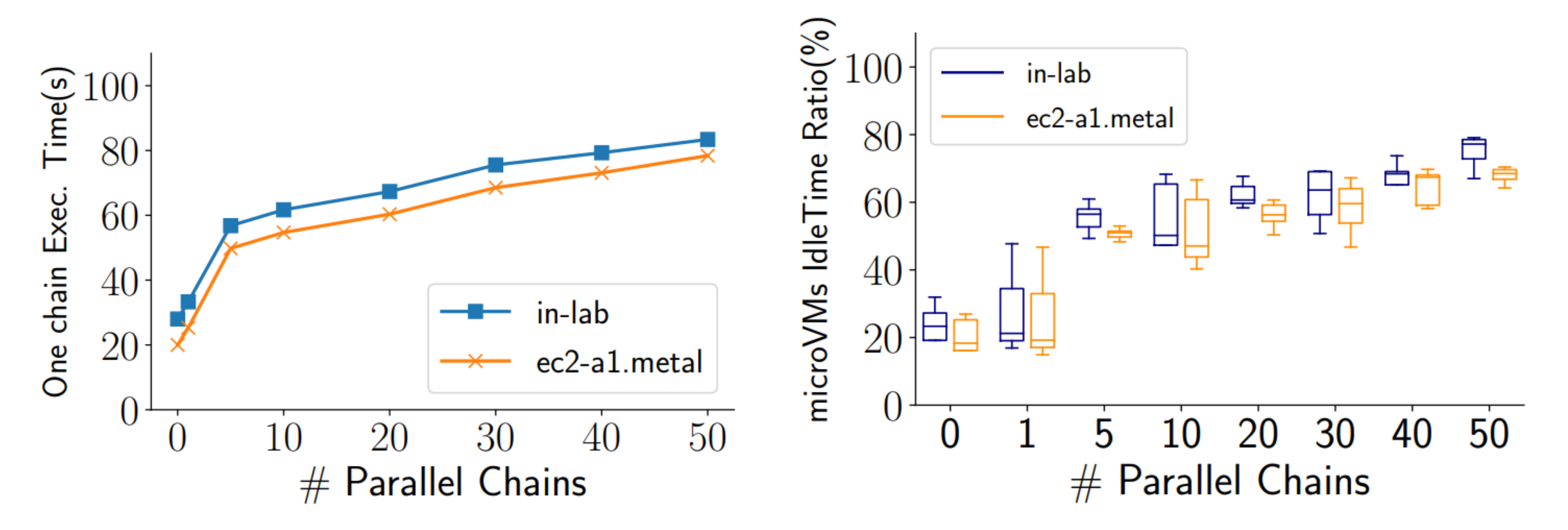
28.3s to 83.41s --- inlab

20s to 78.52s --- AWS a1.metal

Idle time ratio

20.18% to 75.31% --- inlab

16.25% to 69% --- AWS a1.metal

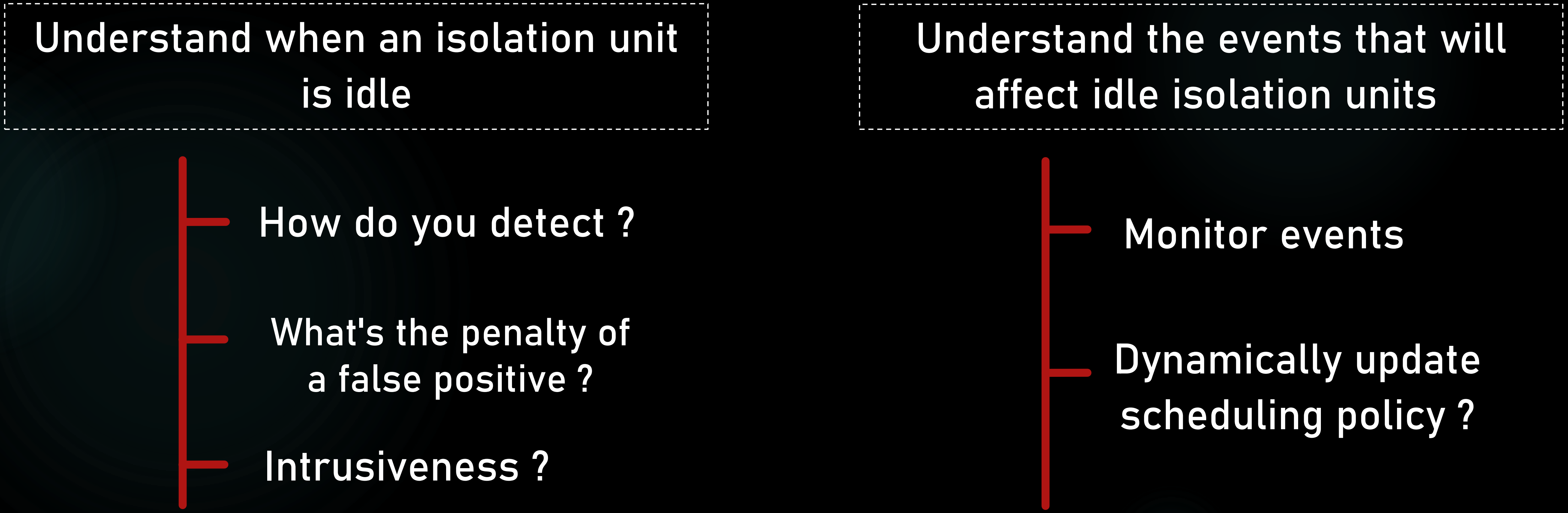


CPU time is wasted, smarter use could improve overall execution time

Possible ideas: Scheduling semantic gap

Well known problem in the context of virtualization.

The host scheduler should :



Understand when an isolation unit is idle

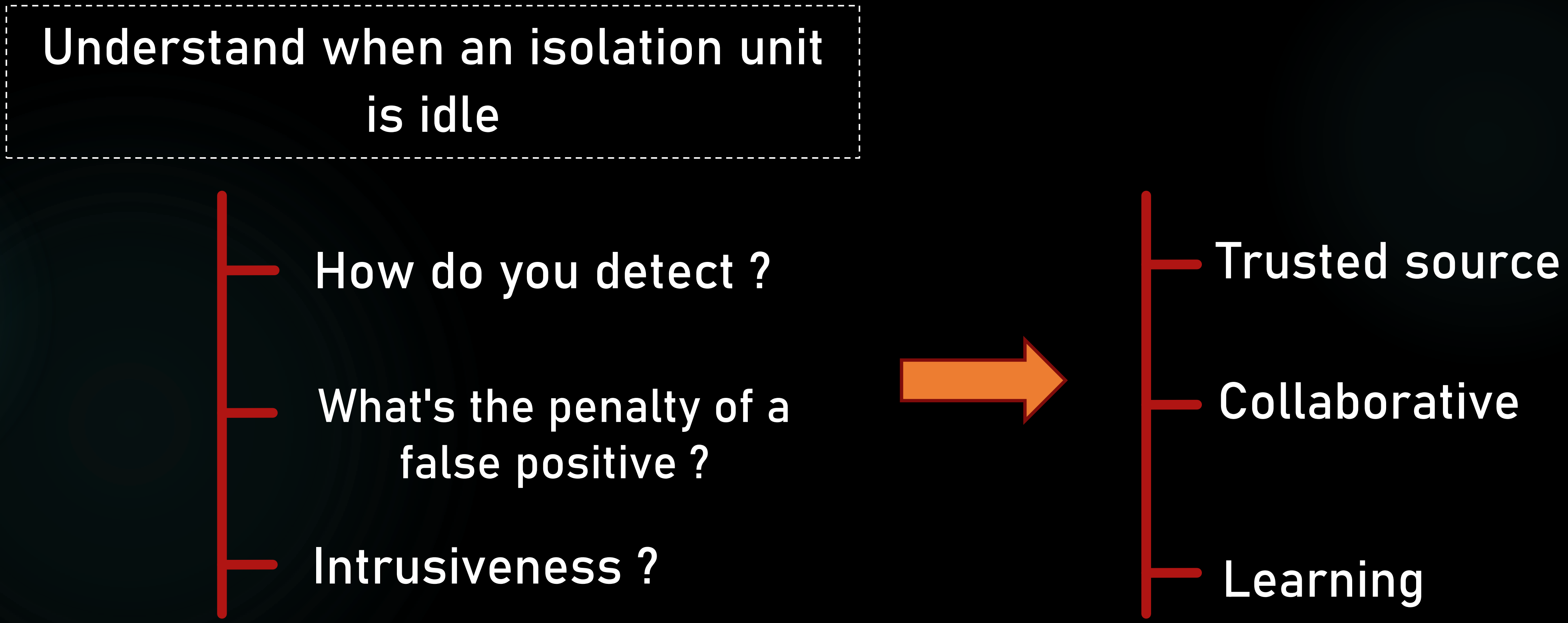
- How do you detect ?
- What's the penalty of a false positive ?
- Intrusiveness ?

Understand the events that will affect idle isolation units

- Monitor events
- Dynamically update scheduling policy ?

Possible ideas : Scheduling semantic gap

Approaches worth exploring



Understand when an isolation unit is idle

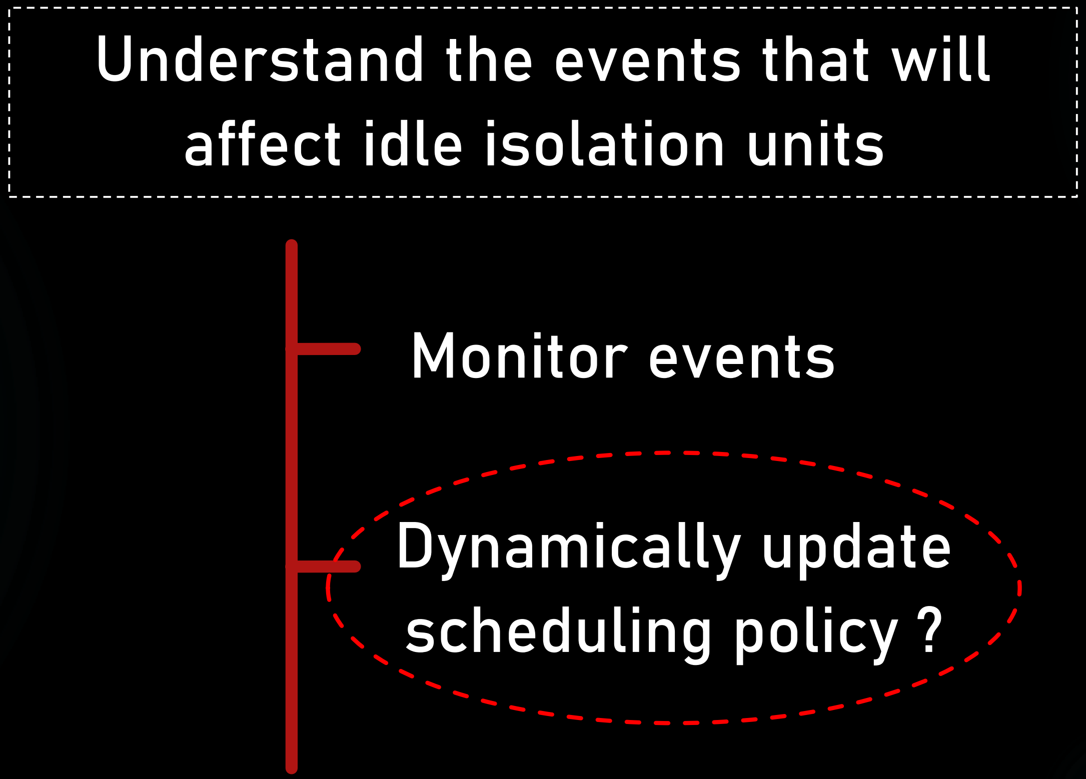
- How do you detect ?
- What's the penalty of a false positive ?
- Intrusiveness ?

- Trusted source
- Collaborative
- Learning

Possible ideas: Scheduling semantic gap

Well known problem in the context of virtualization.

The host scheduler should :



Understand the events that will affect idle isolation units

- Monitor events
- Dynamically update scheduling policy ?

Problem : Scheduling bad performance

Dynamic scheduler behavior…

Not an easy task

- **Fixed policies (extensible to some extend but remain rigid)**

Patching, scheduler class with a light interface

- **Lack of visibility on users' thread behavior**

Justinien Bouron et al. Thee Battle of the Schedulers: FreeBSD ULE vs. Linux CFS. ATC 2018

Redha Gouicem et al. Fewer Cores, More Hertz: Leveraging High-Frequency Cores in the OS Scheduler for Improved Application Performance. ATC 2020

Weiwei Jia et al. vSMT-IO: Improving I/O Performance and Efficiency on SMT Processors in Virtualized Clouds. ATC 2020

Bao Bui et al. When eXtended Para-Virtualization (XPV) meets NUMA. Eurosys 2019

Jean Pierre Lozi et al. . The Linux scheduler: a decade of wasted cores. Eurosys 2016

Our idea: Towards user-programmable schedulers

Users **express** the scheduler behavior at **runtime**

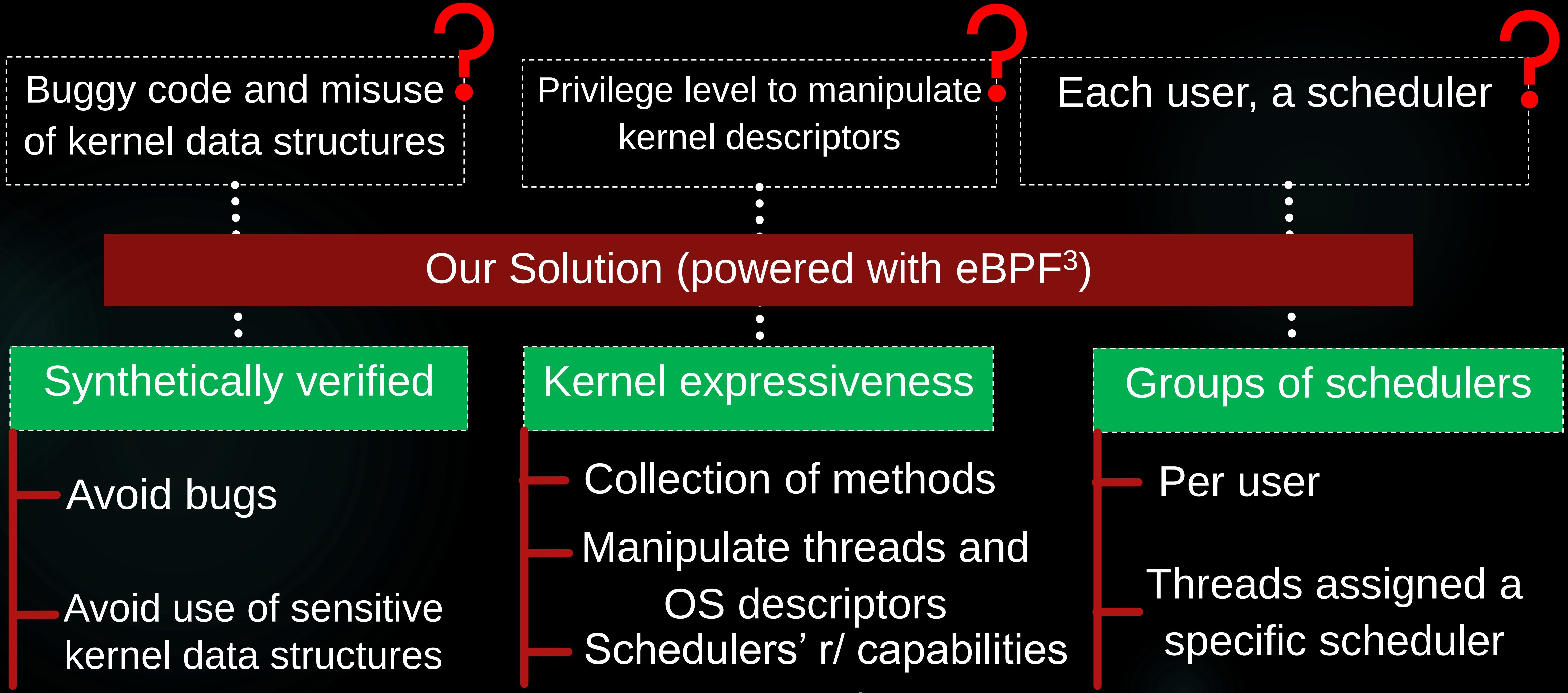
Buggy code and misuse of kernel data structures

Privilege level to manipulate kernel descriptors

Each user, a scheduler

Our idea: Towards user-programmable schedulers

Users **express** the scheduler behavior at **runtime**



Buggy code and misuse of kernel data structures ?

Privilege level to manipulate kernel descriptors ?

Each user, a scheduler ?

Our Solution (powered with eBPF[3])

Synthetically verified

- Avoid bugs
- Avoid use of sensitive kernel data structures

Kernel expressiveness

- Collection of methods
- Manipulate threads and OS descriptors
- Schedulers’ r/ capabilities

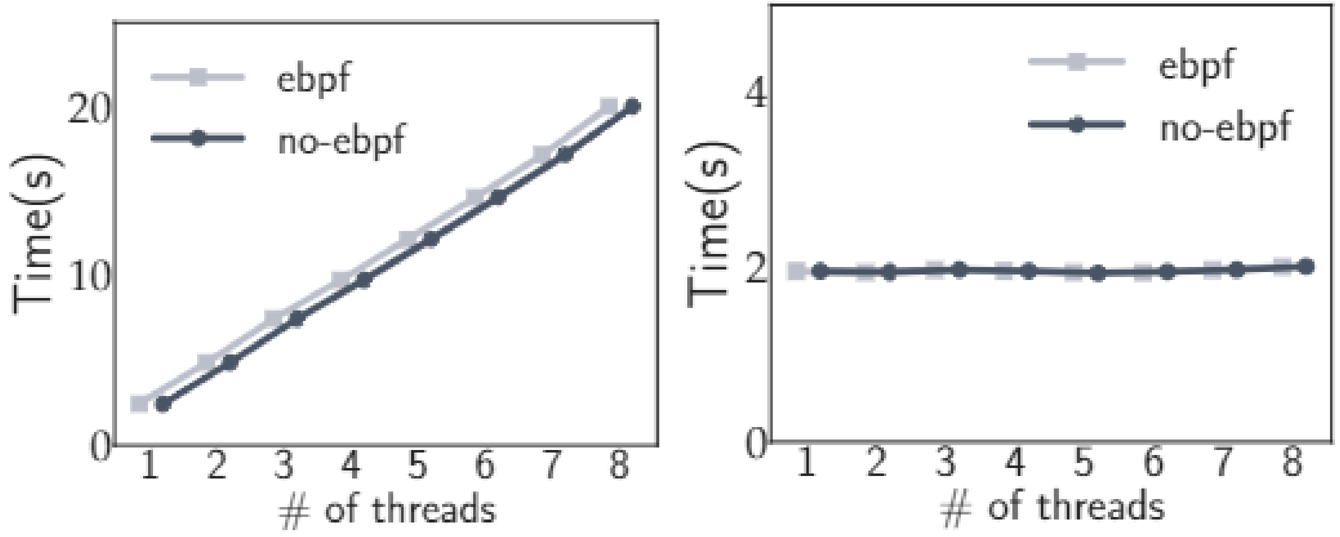
Groups of schedulers

- Per user
- Threads assigned a specific scheduler

[3] https://ebpf.io

Our idea: Towards user-programmable schedulers

Some initial results --- simple FIFO scheduler with our mechanism



(a) *Execution time of a process.*

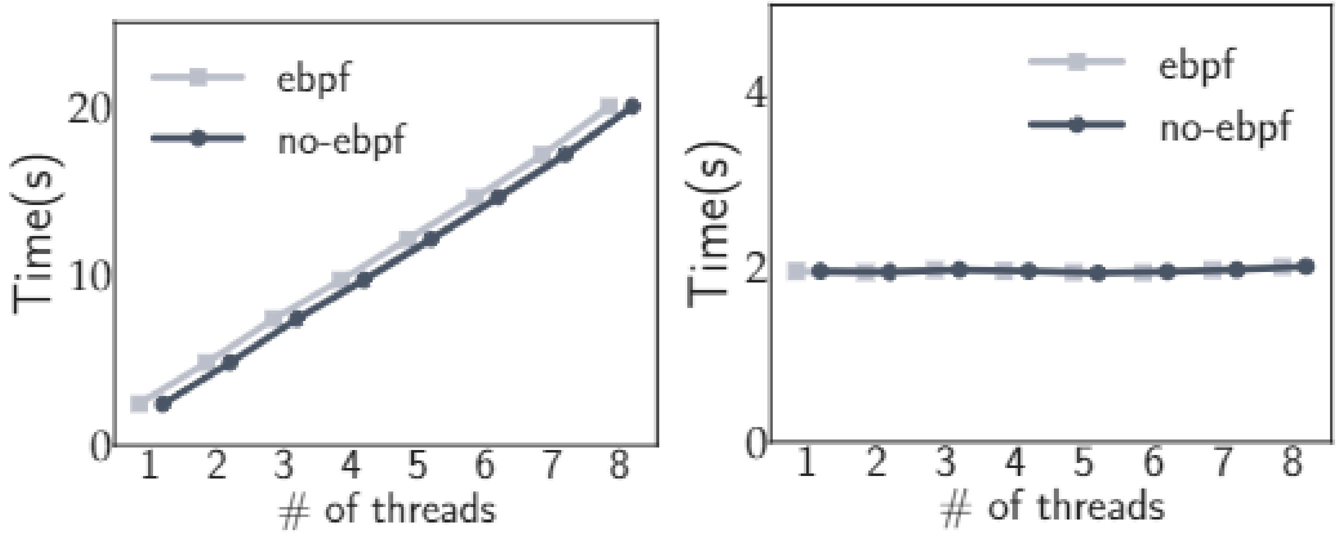
(b) *avg CPU time per process.*

Figure 2. *eBPF-custom FIFO against Linux standard FIFO. We compute the (a) the execution time (wall-clock time) of one program, and (b) the average CPU time of each program as we increase the number of running programs.*

Each thread computes prime numbers between 1 and 100,000

Our idea: Towards user-programmable schedulers

Some initial results --- simple FIFO scheduler with our mechanism



(a) *Execution time of a process.*

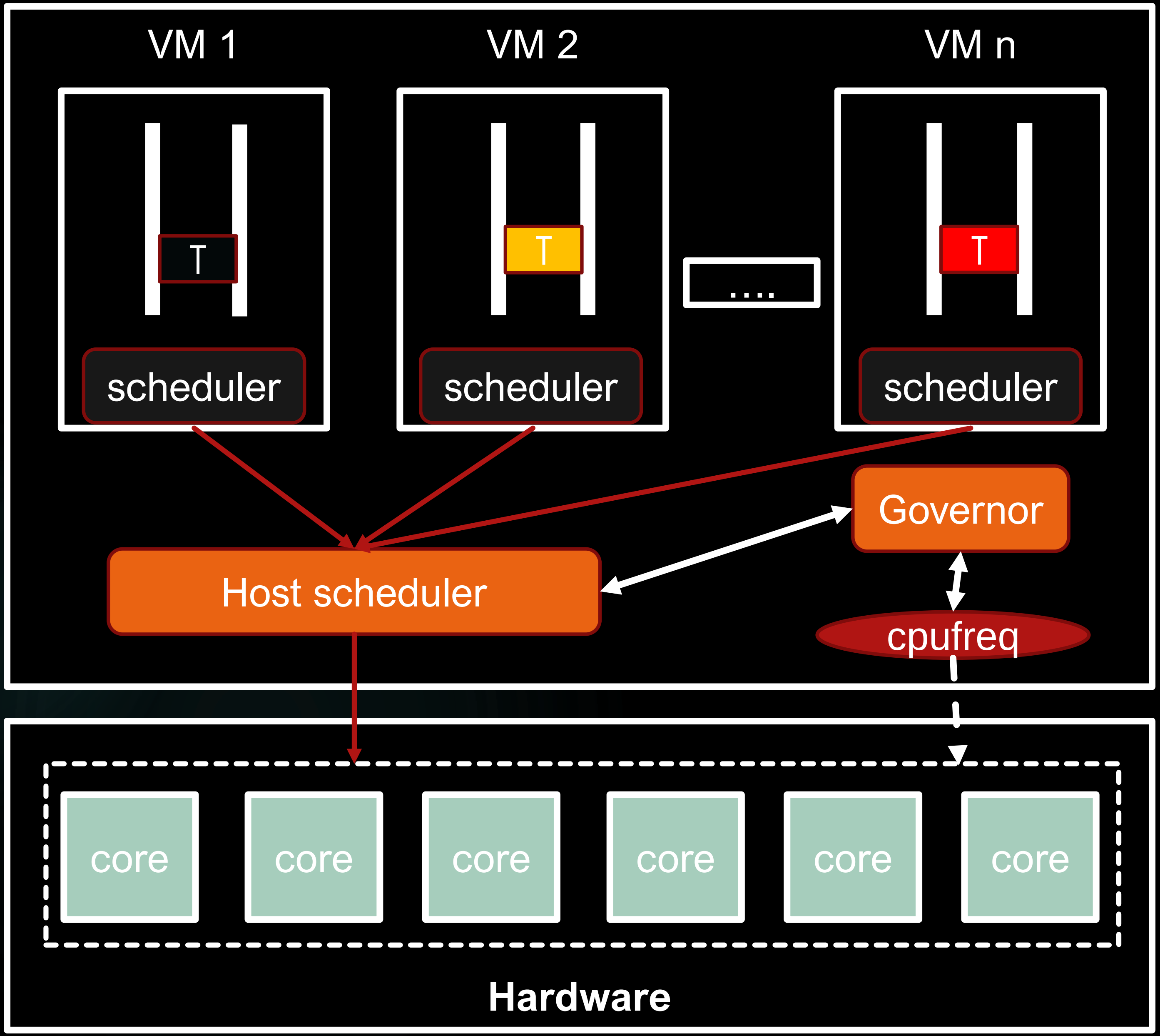
(b) *avg CPU time per process.*

Figure 2. *eBPF-custom FIFO against Linux standard FIFO. We compute the (a) the execution time (wall-clock time) of one program, and (b) the average CPU time of each program as we increase the number of running programs.*

Interesting results but some work remains

- **JIT compiler performs poorly for loops**
- **Security issues regarding kernel data usage --- what to allow?**

PROBLEM 2: Incorrect accounting for IDLE ISOLATION UNITS



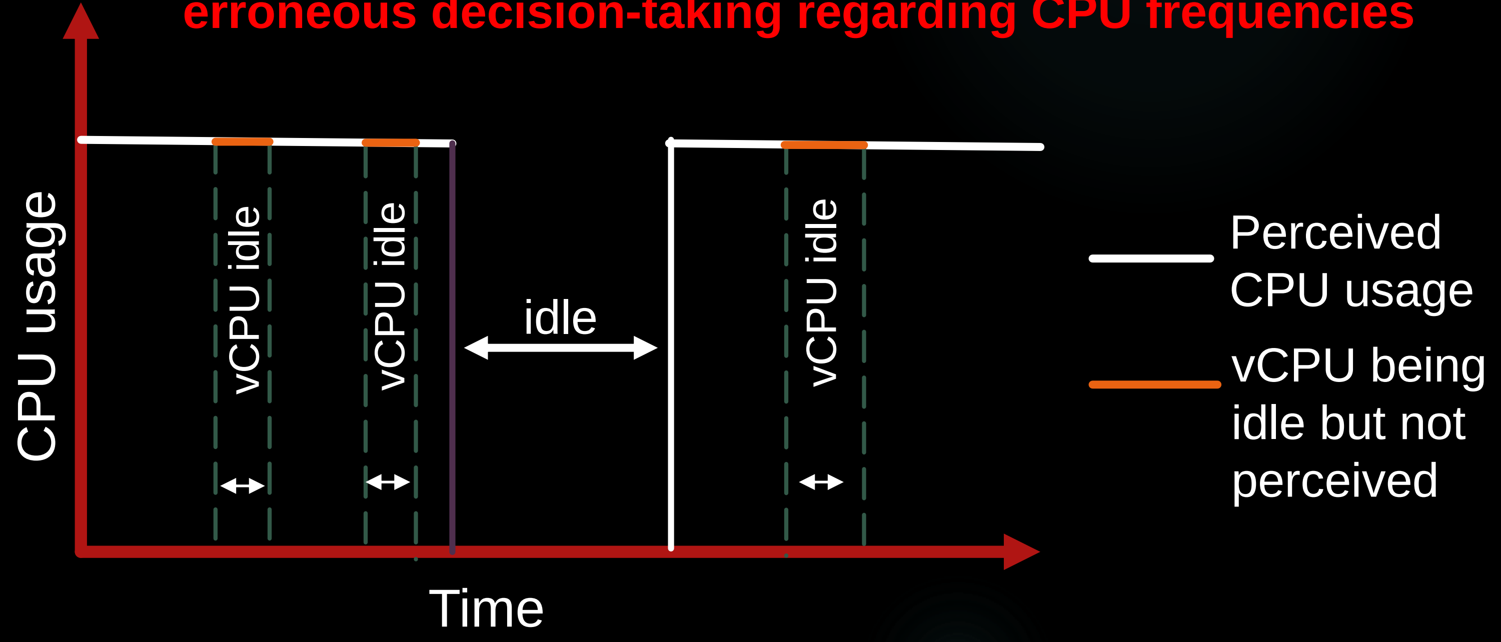
- ❑ Governors decide at which frequency the CPU should run favoring either performance or energy savings.

- ❑ Rely on metrics maintained by the scheduler to have an idea of each core CPU usage

PROBLEM 2: Incorrect accounting for IDLE ISOLATION UNITS

- The host scheduler view of the vCPU is limited. Apart from explicit idle instructions (e.g., sleep), the vCPU is viewed as running.
- However, a running vCPU can be idle
  - Idle loop
  - Waiting for software interrupts
  - …

This leads to **erroneous CPU usage computation** and **erroneous decision-taking regarding CPU frequencies**



Example: On-demand algorithm for Xen

```
Curr_time = NOW()
Time_since_epoch=curr_time-prev_time
```

Get time and elapsed time since last call

```
For each cpu (j):
        curr_idle_ns=get_cpu_idle_time(j)
        idle_since_epoch=curr_idle_ns-j.prev_idle_ns
        j.prev_idle_ns=curr_idle
```

Compute idle time since last call

```
        if(time_since_epoch<idle_since_epoch) continue;

        curr_freq = get_current_freq()
        load = 100* (time_since_epoch – idle_since_epoch)/(time_since_epoch)
        load_freq = load * curr_freq
```

Compute current load based on the idleness

```
        if(load_freq > max_load_freq) max_load_freq = load_freq
```

Get the maximum percentage load across each cpu

```
if( load_freq > upper_threshold) push_to_max_frequency()
if( load_freq < lower_threshold) :
        next_freq = load_freq/(threshold-10);
        push_to_next_freq(next_freq);
```

Increase or decrease frequency --- when decreasing, tries to get the frequency that will not instantaneously trigger up policy

Governors overview across hypervisors

Xen

- Tracks for each vCPU the idle time
- Assumes the vCPU was running beside the idle time
- Basic threshold computation to decide the next frequency

Linux/KVM

- For each run queue, tracks runnable and running time with a polynomial approximation based on the scheduling entity load (PELT)
- By that, they can leave out the idleness and focus on the running part
- Basic threshold computation to decide the next frequency

Our idea - WIP

Collaboration

- Add a tracker in each isolation unit that observes the main process and can identify idle periods
- Updates a flag via shared memory to tell the host scheduler that the corresponding isolation unit is idle and should be considered in the accounting.

Our idea - WIP

Collaboration mechanism

- Add a tracker in each isolation unit that observes the main process and can identify idle periods
- Updates a flag via shared memory to tell the host scheduler that the corresponding isolation unit is idle and should be considered in the accounting.

Issues

- Identifying idle periods can require to peek into the VM
- Sharing introduces challenges regarding security --- Cloud platforms may not like
- What is the right behavior regarding shared memory? Event-based or periodic reading …

Exploring Hardware-based tracking --- extend HWP

Thanks

Questions and ideas ? ☺