Accelerating Nested Virtualization with HyperTurtle

Ori Ben Zur*, Jakob Krebs*, Shai Bergman†, Mark Silberstein*

*

†

ori.b@campus.technion.ac.il

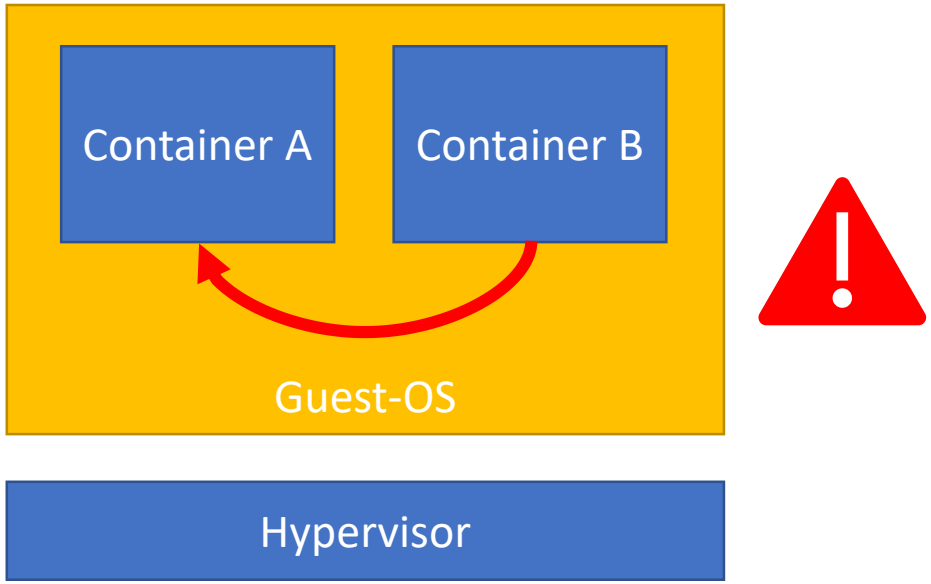
Why use nested virtualization?

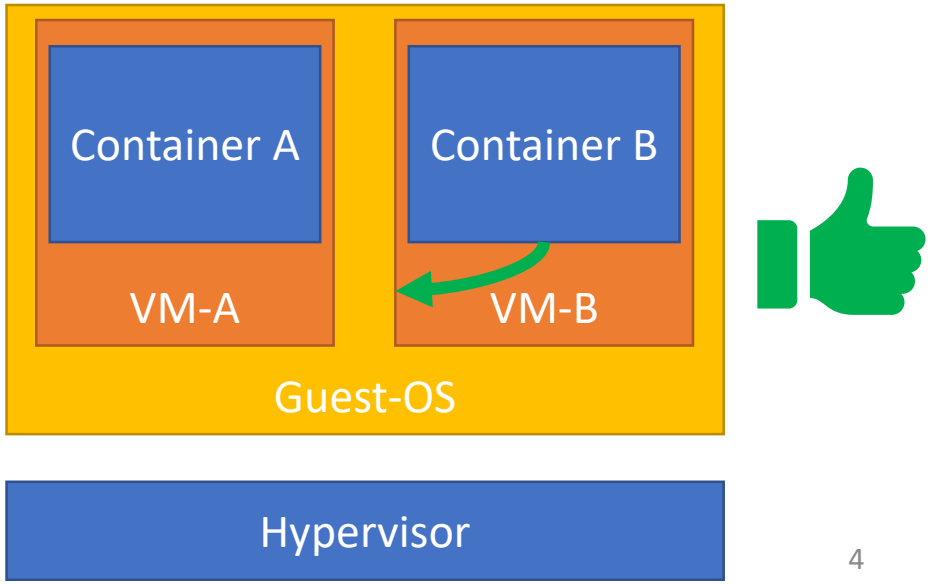
Let’s talk about containers

- Sandboxed environment for applications
- Enables multitenancy
- Often deployed on virtualized infrastructure

The Weak Isolation of Containers

- Containers share a kernel – has a large attack surface
- Solution: strengthen isolation using virtualization-based containers[1]
- Problem: performance penalty of nested virtualization





[1] Kata Containers. The Speed of Containers, The speed of containers, the security of VMs. katacontainers.io

The Nested Virtualization Tax

- Compared to non-nested virtualization:
  - EPT fault: 5.1X slower
  - Container startup: 2.1X slower
  - Profiling sampler: 10.7X slower

Our Solution: HyperTurtle

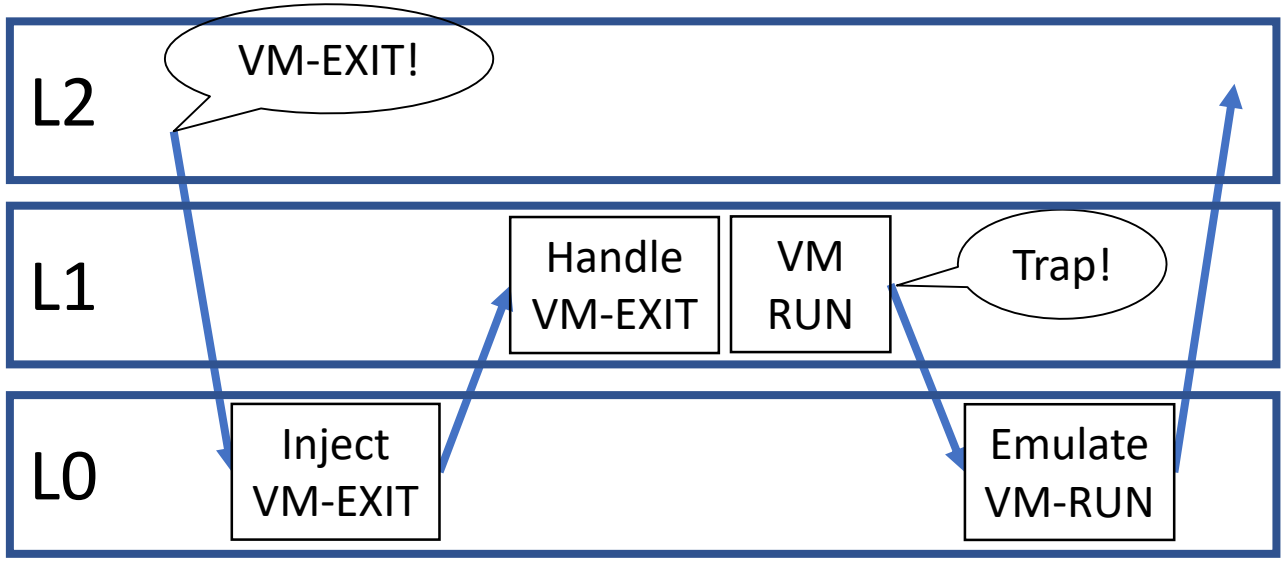
- What: General approach for accelerating nested VMs
- How: Safely *offload code* from the guest-hypervisor (L1) to the host (L0)
  - Rewriting code as eBPF running *in L0*
- Non-Intrusive: Minimal code changes:
  - <300 LoCs in L0 kernel
  - <700 LoCs in L1 kernel
- Results: Improves performance in 3 subsystems
  - **Virtual Memory Management:** -27% container startup time
  - **Networking:** +72% Memcached throughput
  - **Profiling:** Near non-nested profiling performance

Agenda

- Motivation
- **The nested virtualization performance problem**
- HyperTurtle – generic approach for accelerating nested VMs
- Showcase: Accelerating virtual memory management
- Evaluation

Why is Nested Virtualization Slow?

- L1 CPU is virtualized (non-root)
- L1's virtualization operations are *trapped & emulated* by L0
- All L2's vm-exits *pass through* L0
- ⇒Large amount of *world switches*



Overheads of Nested Virtualization

- **World switches** cause significantly **higher overheads** in nested VMs
  - Slowdown **depends** on the exit reason (up to 10.7X slower)
- Cause: Increased **handling time** and **amount** of vm-exits
- Goal: reduce world switch overhead
- Problem: overheads hide everywhere, **no silver bullet**


See paper for detailed analysis!

Previous Work

- Lluís Vilanova et al. **Using SMT to Accelerate Nested Virtualization.** ISCA'19
  - Run each virtualization level on a different HW thread
  - Only reduces context switch time
- Jin Tack Lam and Jason Nieh. **Optimizing Nested Virtualization Performance Using Direct Virtual Hardware.** ASPLOS 2020
  - Attach L0-supplied virtual devices to L2, bypassing L1
  - Limited scope
  - L1 cannot enforce policies on I/O traffic
- Hang Huang et al. **PVM: Efficient Shadow Paging for Deploying Secure Containers in Cloud-native Environment.** SOSP'23
  - Use specialized hypervisor + SW virtualization for L2 guest
  - Limited scope
  - Increased overheads in syscall/memory intensive applications

Agenda

- Motivation
- The nested virtualization performance problem
- **HyperTurtle – generic approach for accelerating nested VMs**
- Showcase: Accelerating virtual memory management
- Evaluation

Design Goals of HyperTurtle

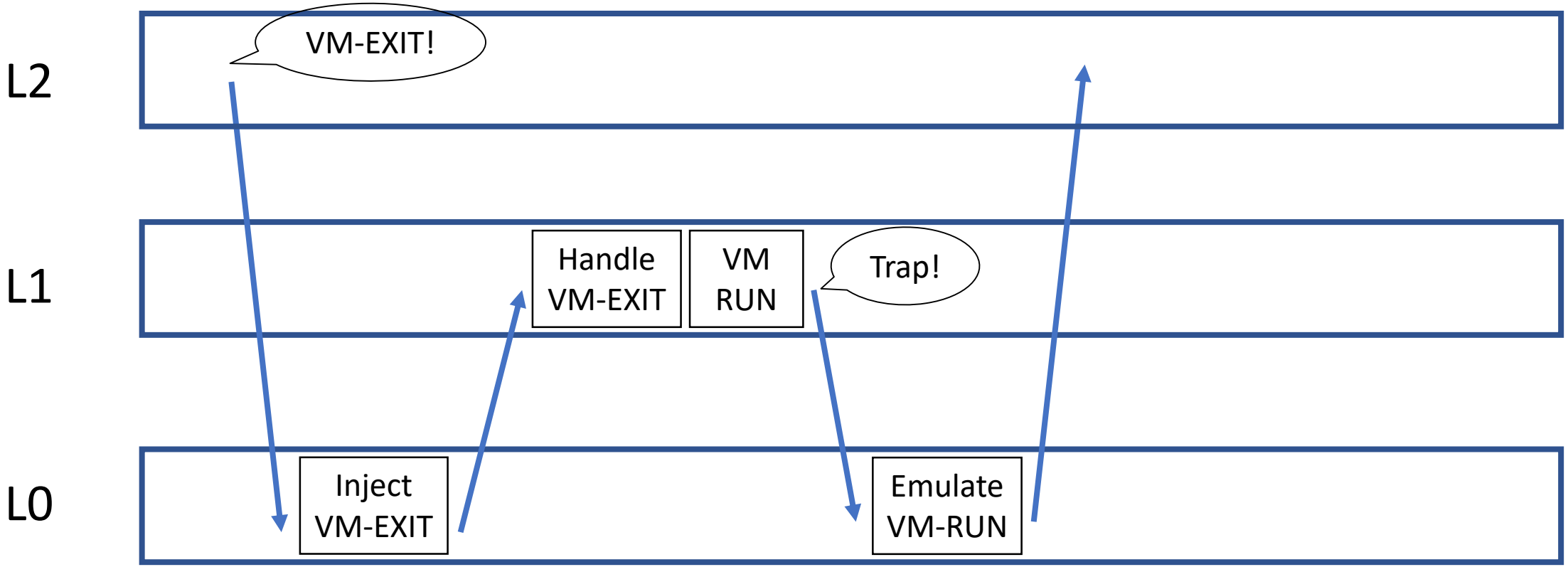
- General infrastructure
  - A single approach that alleviates overheads
- Non-intrusive
  - No structural kernel code changes, faster upstreaming
- Keep L1 in control
  - Maintain compatibly with container infrastructure
- Performance
  - Obviously

The HyperTurtle Approach

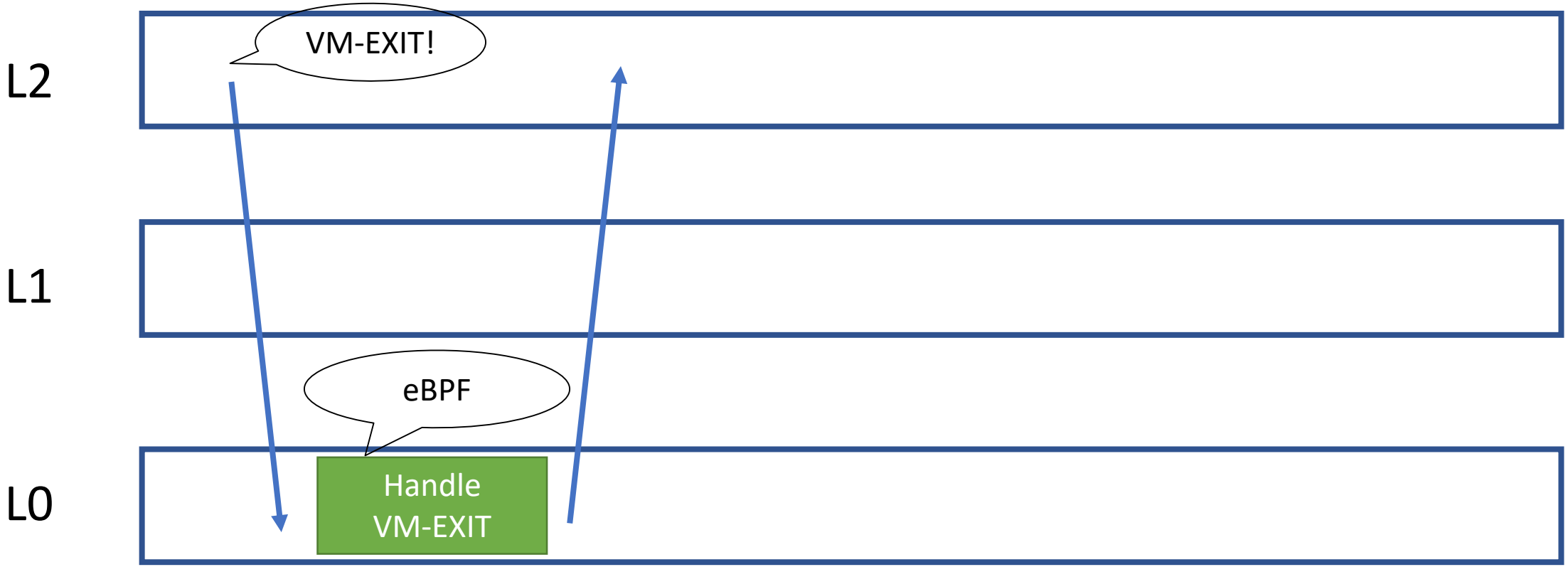
- HyperTurtle: Handle L2 exits by *offloading code* from L1 to L0
  - All while maintaining isolation between L1s
- Provide a *development recipe* to accelerate the handling of a vm-exit
- Result: Accelerated 3 subsystems: **virtual memory management**, **networking**, and **profiling**
  - Each subsystem uses a different eBPF program

[2] Nadav Amit and Michael Wei. The Design and Implementation of Hyperupcalls. In ATC'18

This way, HyperTurtle turns this…



Into this!



Tying it Together: Development Recipe

Share information: between L1 and the eBPF program, using *eBPF maps* and *new helper functions*

Hook into L0: define a hook in L0 for the eBPF program

Develop eBPF program: write the desired functionality as an eBPF program

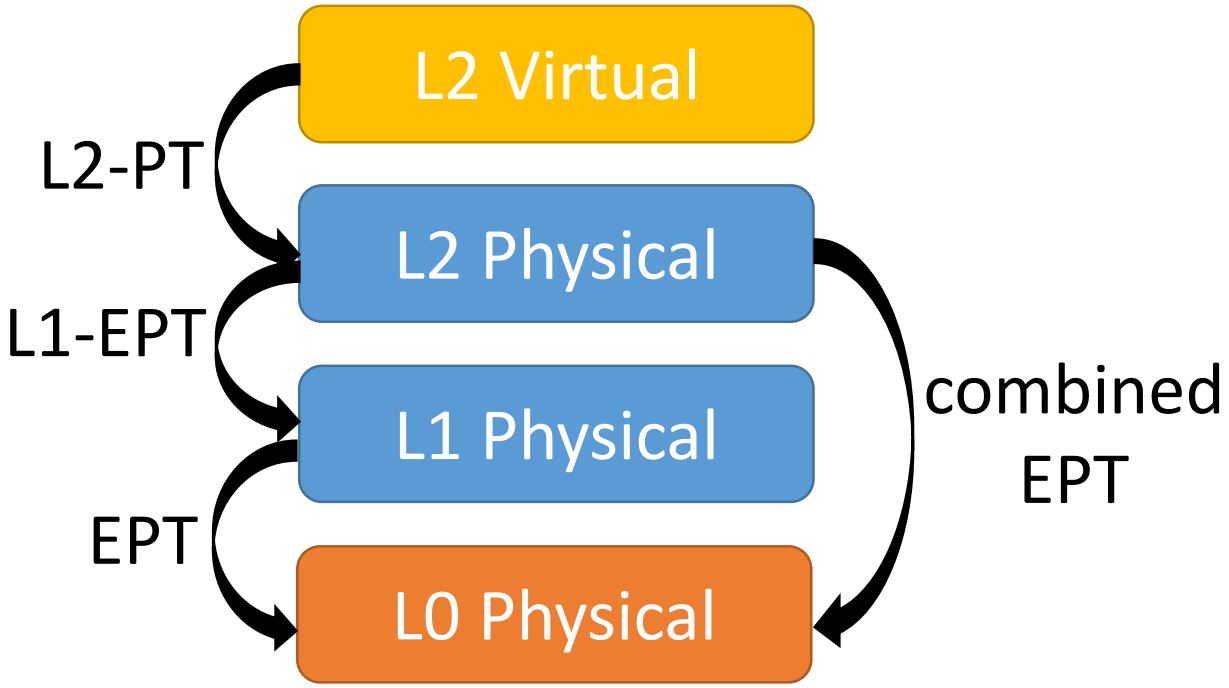
- Helper function and hook must keep L1s isolated

See paper for more details!

Agenda

- Motivation
- The nested virtualization performance problem
- HyperTurtle – generic approach for accelerating nested VMs
- **Showcase: Accelerating virtual memory management**
- Evaluation

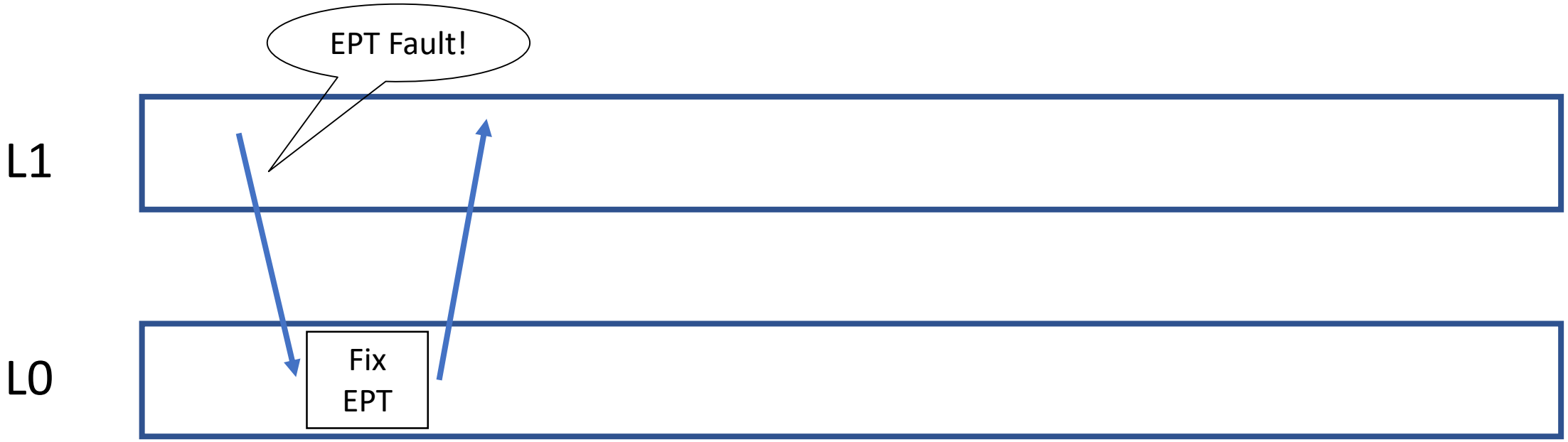
Background: Nested Paging in Nested Virtualization



EPT Faults are ~5X slower

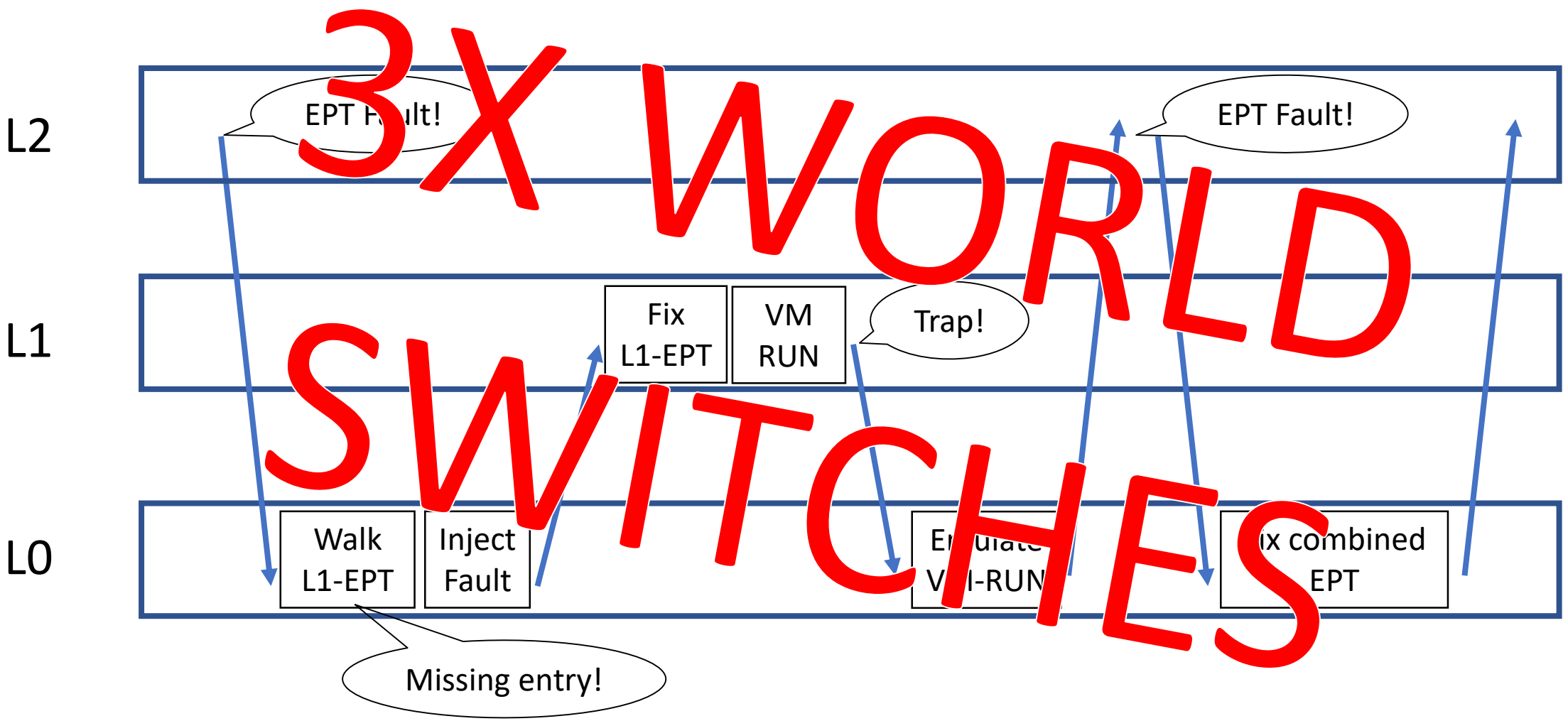
Account for 46% of container startup time

EPT Fault in a Non-Nested Guest - ~7us

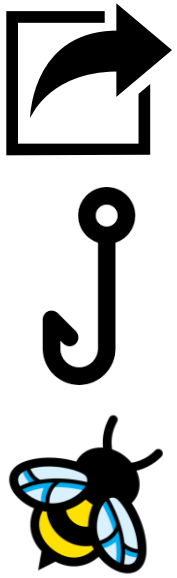


EPT Fault in a Nested Guest - ~35us

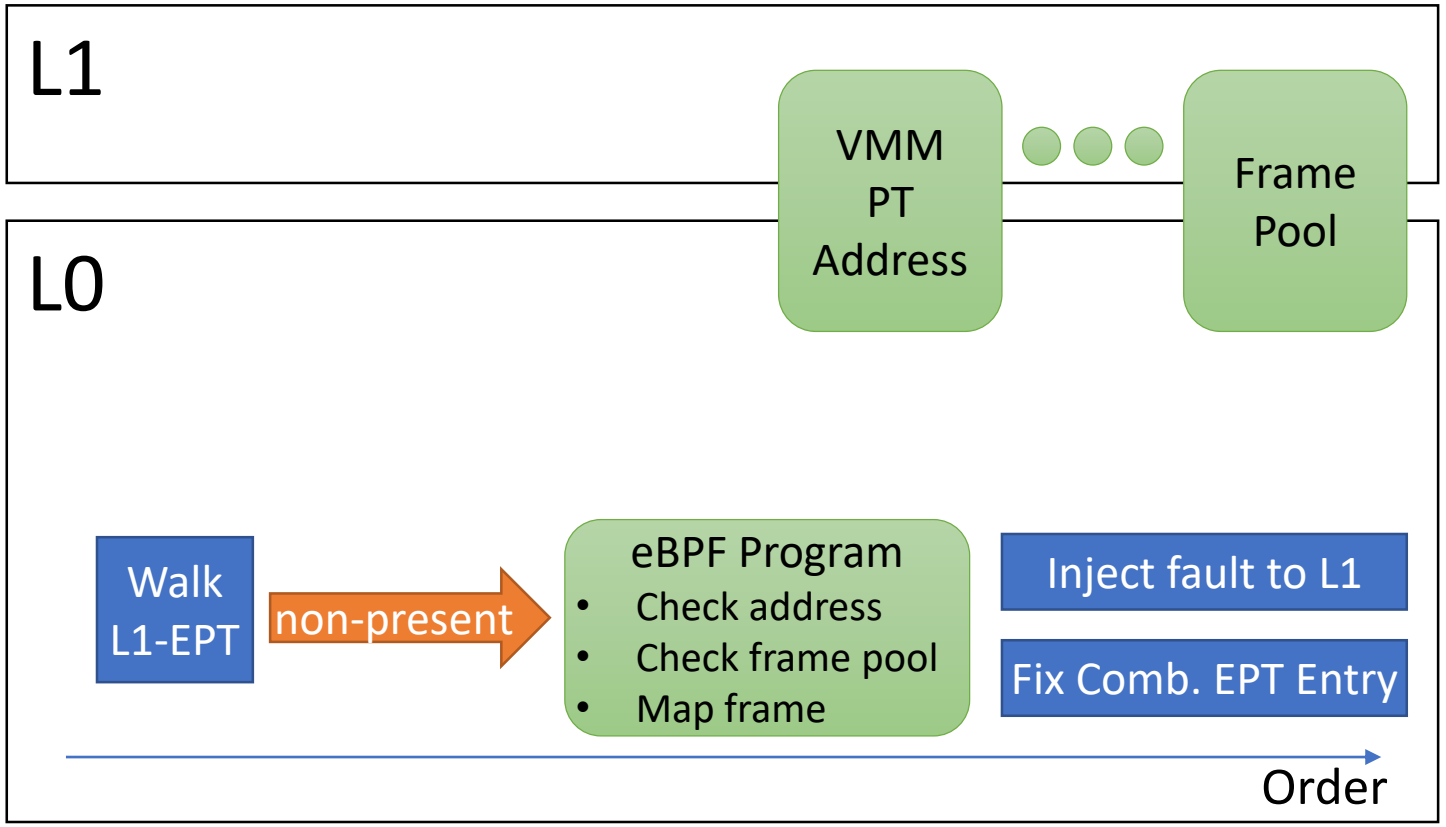
L1-EPT Entry	Comb. EPT Entry
Present	Present



HyperTurtle for EPT Fault – Design Overview



Result: handle EPT fault in **~7us**



More details in the paper!

Agenda

- Motivation
- The nested virtualization performance problem
- HyperTurtle – generic approach for accelerating nested VMs
- Showcase: Accelerating virtual memory management
- **Evaluation**

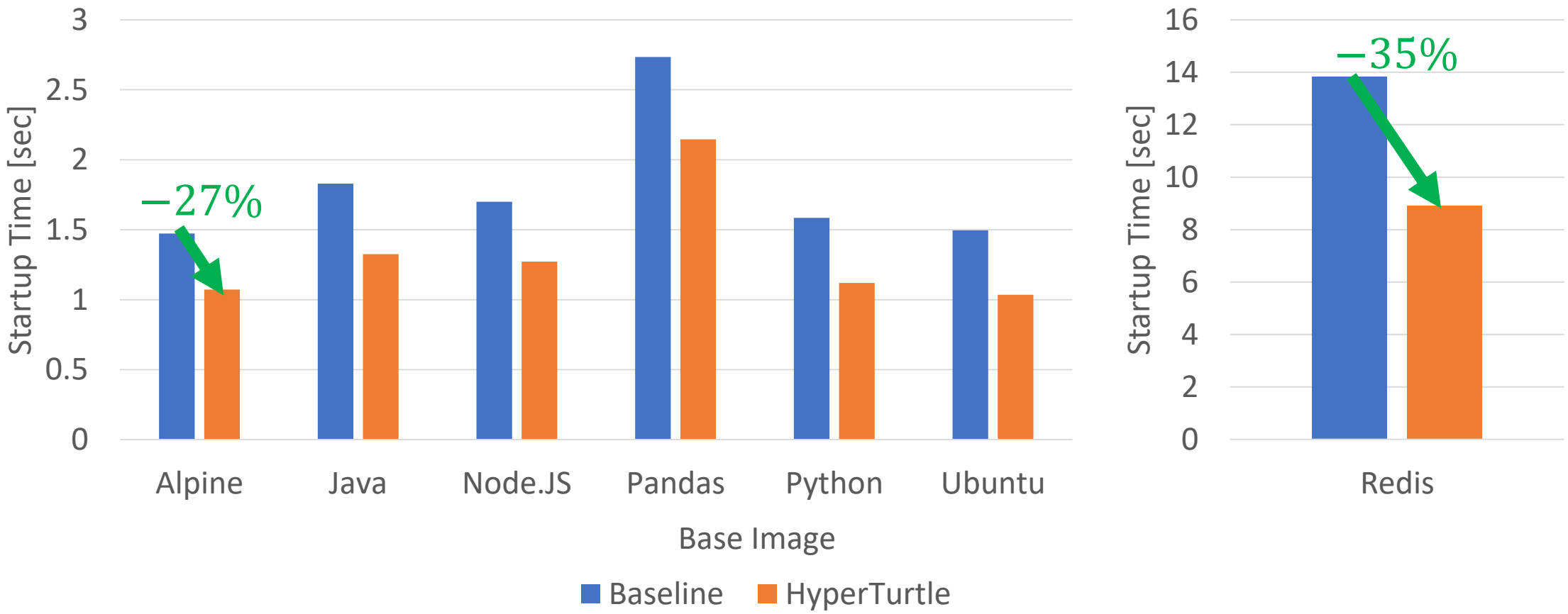
Evaluation

- Microbenchmarks
  - EPT fault latency
  - **Profiler sampling latency**
  - UDP ping latency
  - TCP throughput
- Kata Container Startup Time
  - **Various base images**
  - DeathStarBench applications[3]
  - **Redis from populated snapshot**
- Networking with different eBPF programs
  - Nginx throughput
  - Redis throughput
  - **Memcached latency-throughput**
  - **Memcached latency-throughput w/ profiling**


More details in the paper!

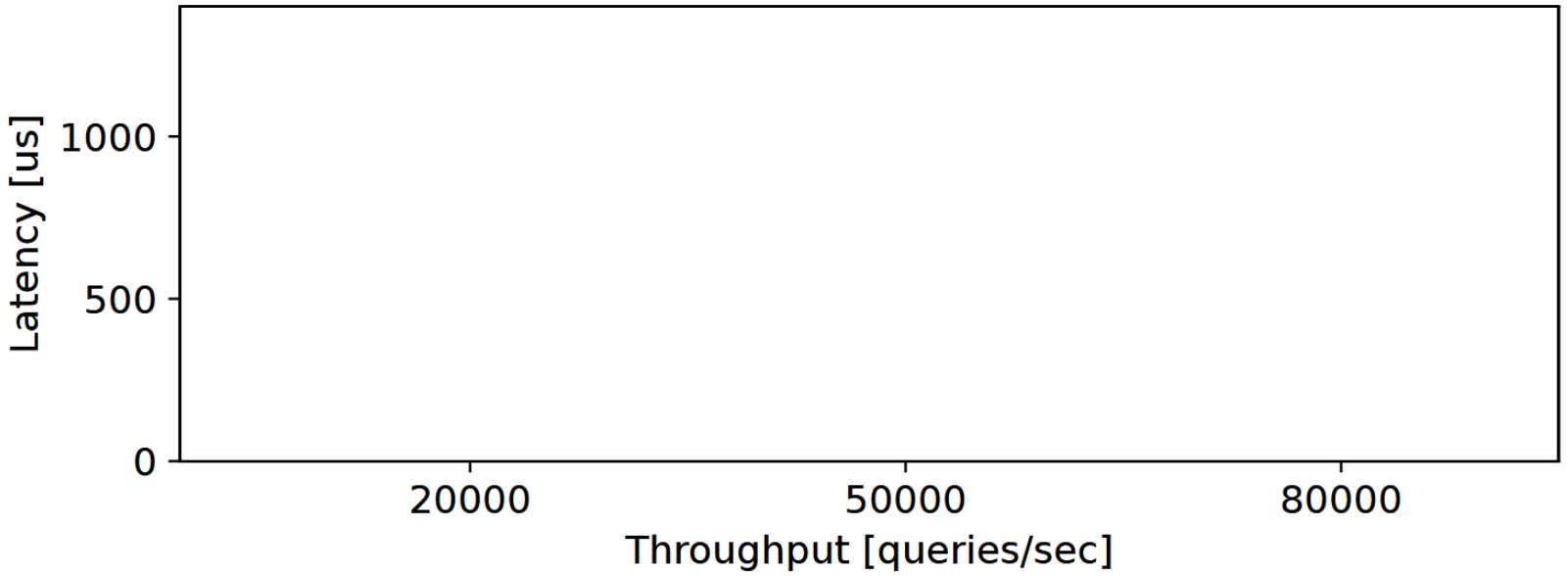
[3] Y. Gan et al., An Open-Source Benchmark Suite for Microservices and Their Hardware-Software Implications for Cloud and Edge Systems, ASPLOS 2019.

Kata Container Startup Time



Memcached Performance

- Reminder: L1 is in control
- Example: Implement L1 firewall policy as eBPF program

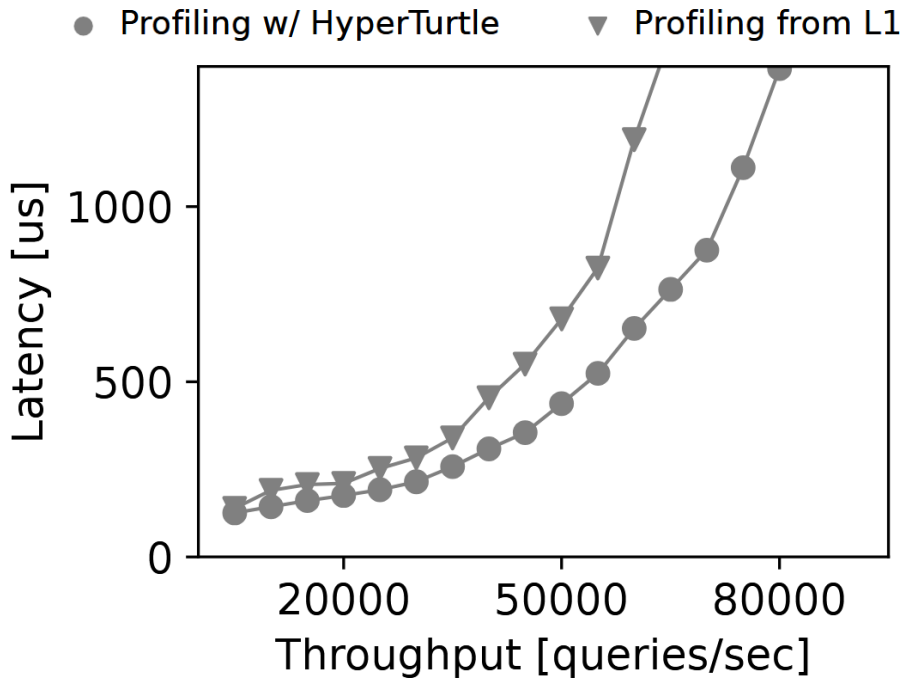


[4] Jin Tack Lam and Jason Nieh. Optimizing Nested Virtualization Performance Using Direct Virtual Hardware. ASPLOS 2020

Profiling

“Sample your application” latency

Memcached Performance



Conclusion

- Nested virtualization suffers from expensive vm-exits
- HyperTurtle: accelerate nested virtualization by *offloading code* from L1 to L0
- Improves performance in 3 subsystems:
  - **Virtual Memory Management:** -27% container startup time
  - **Networking:** +72% Memcached throughput
  - **Profiling:** Near non-nested profiling performance

Thanks for listening!

Questions?

ori.b@campus.technion.ac.il

github.com/acsl-technion/hyperturtle