Imperial College London

LSDS

Large-Scale Data & Systems Group

Security as a Performance Principle

A tale of hardware/software co-design

Lluís Vilanova

<vilanova@imperial.ac.uk>

Imperial College London

LSDS

Large-Scale Data & Systems Group

Isolation

~~Security~~ as a Performance Principle

A tale of hardware/software co-design

Lluís Vilanova

<vilanova@imperial.ac.uk>

Imperial College London

LSDS

Large-Scale Data & Systems Group

Isolation

~~Security~~ as a Performance Principle

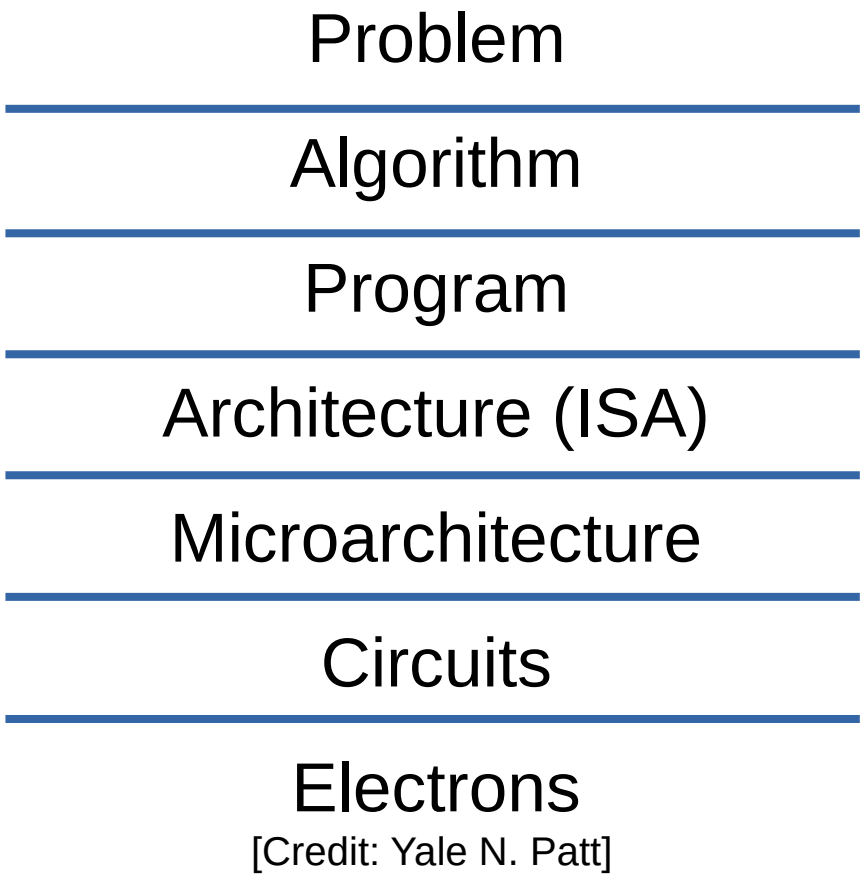
A tale of hardware/software co-design

(or: the Turing tax of systems)

Lluís Vilanova

<vilanova@imperial.ac.uk>

The Bad: Performance vs Security



[Credit: Yale N. Patt]

The Bad: Performance vs Security

Problem

Algorithm

Program

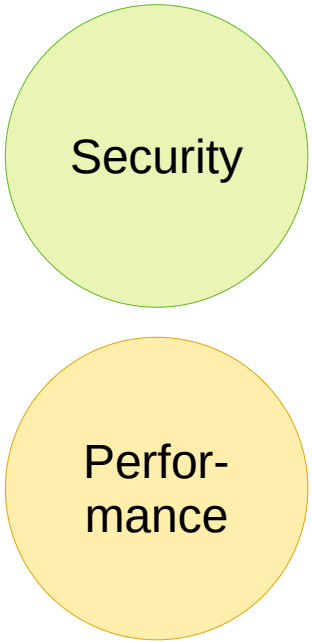
Architecture (ISA)

Microarchitecture

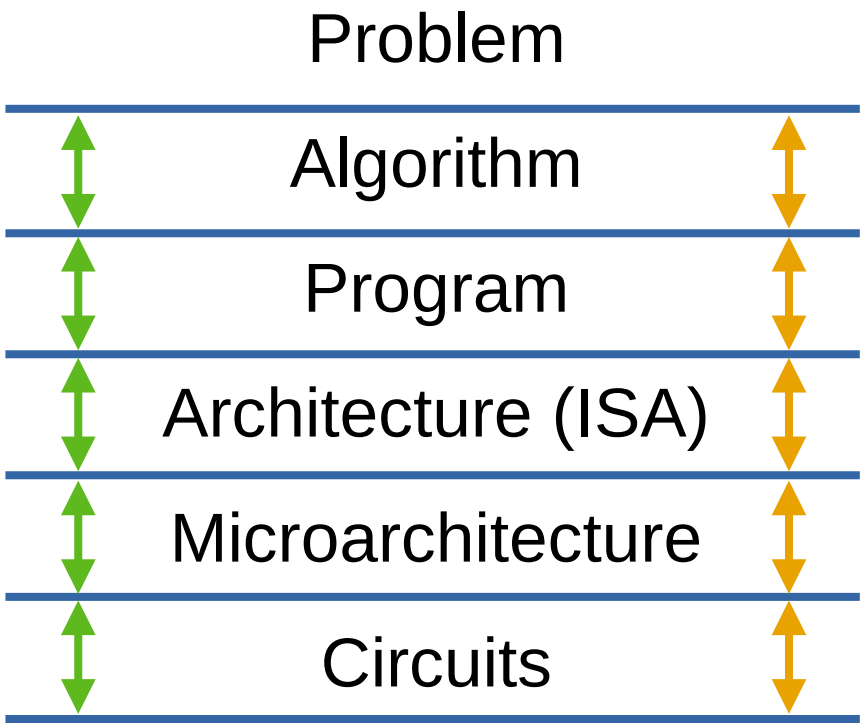
Circuits

Electrons

[Credit: Yale N. Patt]

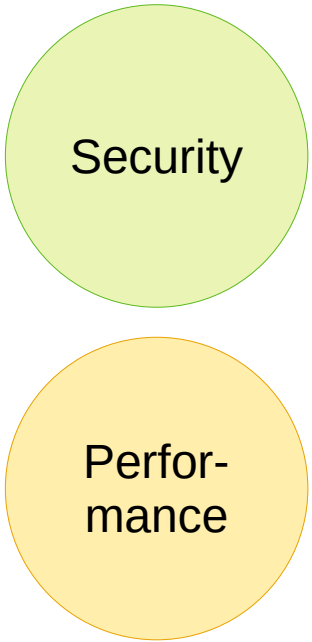


The Bad: Performance vs Security

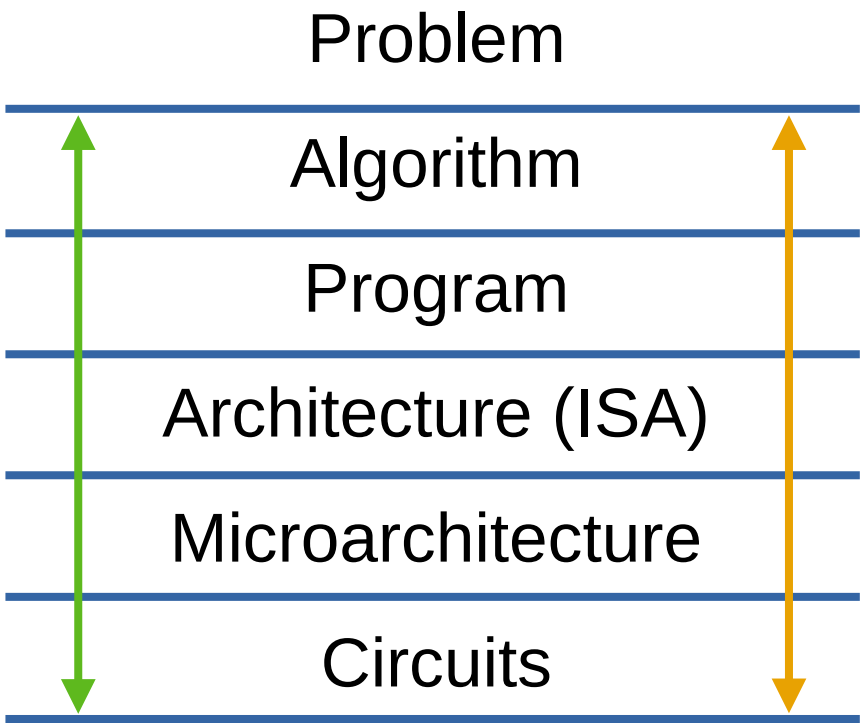


[Credit: Yale N. Patt]

Hierarchies help complexity, but harm cross-cutting concerns

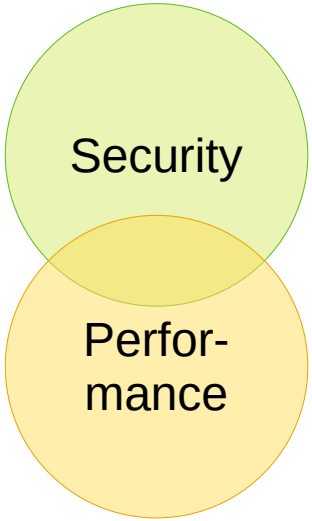


The Good: Performance and Security



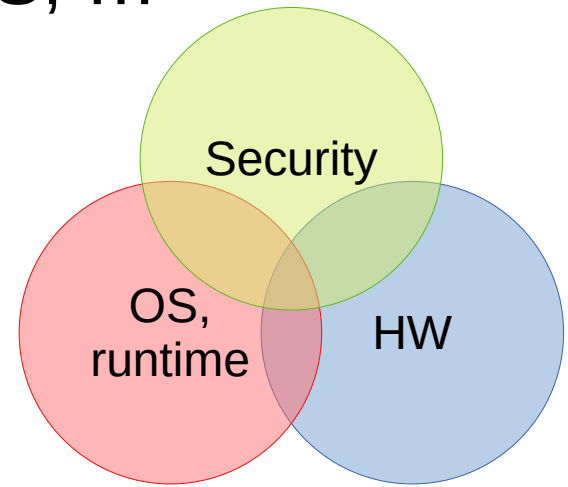
[Credit: Yale N. Patt]

Co-design for security and perf. as end-to-end principles



The Opportunity

- Vertical integration, reloaded
  - New ISAs, accelerators, IaaS, SaaS, …
  - Vendors w/ end-to-end solutions for serving, ML, automotive, ...



Google Cloud ML

aws

AWS Nitro

HUAWEI

RISC-V

Mellanox TECHNOLOGIES

NVIDIA

It’s All About **Communication**

- Happens across logic units *[performance]*
  - Caches, functions, libraries, programs, services, ...
- Happens across isolation units *[security]*
  - Processes
  - Containers
  - VMs
  - Data center

It’s All About **Communication**

- Happens across logic units *[performance]*
  - Caches, functions, libraries, programs, services, ...
- Happens across isolation units *[security]*
  - Processes → *dIPC [EuroSys’17], CODOMs [ISCA’14]*
  - Containers → *CAP-VMs [OSDI’22]*
  - VMs → *SVT [ISCA’19]*
  - Data center → *FractOS [EuroSys’22]*

Hardware/software co-design as a key enabler

It's All About **Communication**

- Happens across logic units *[performance]*
  - Caches, functions, libraries, programs, services, ...
- Happens across isolation units *[security]*
  - Processes → *dIPC [EuroSys'17], CODOMs [ISCA'14]*
  - Containers → *CAP-VMs [OSDI'22]*
  - VMs → *SVT [ISCA'19]*
  - Data center → *FractOS [EuroSys'22]*

Hardware/software co-design as a key enabler

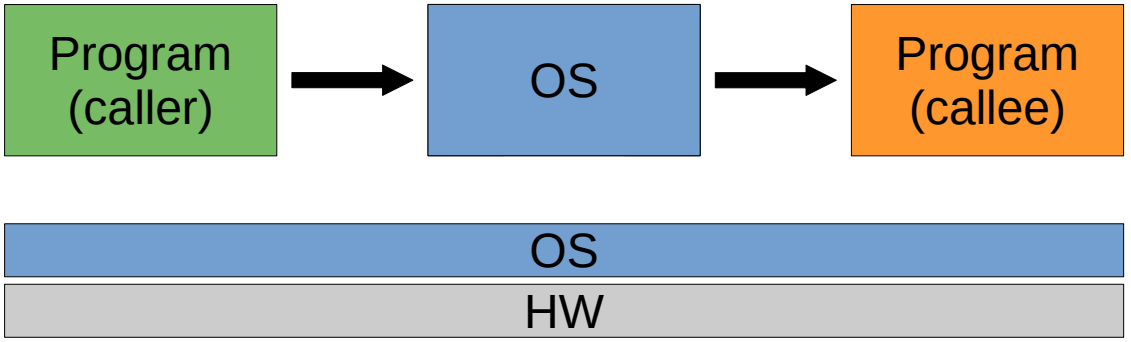
Communication in Processes

[dIPC, EuroSys'17] Direct Inter-Process Communication (dIPC): Repurposing the CODOMs Architecture to Accelerate IPC

[CODOMs, ISCA'14] CODOMs: Protecting Software with Code-centric Memory Domains

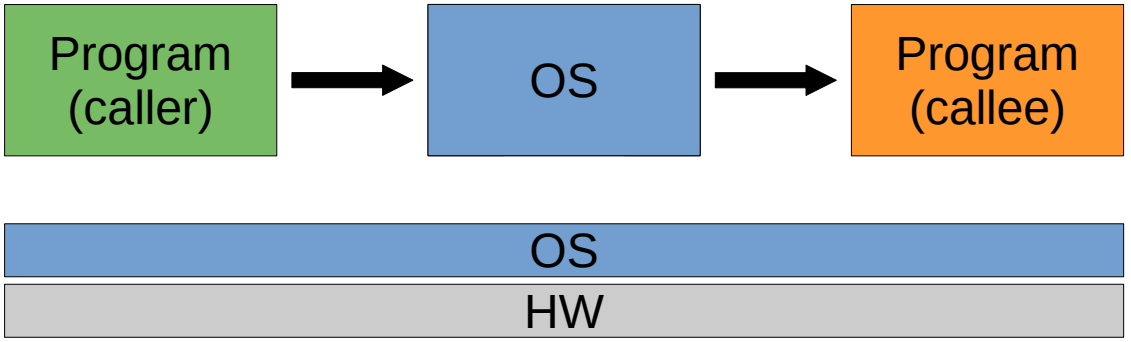
Program Isolation Trade-offs

- Isolation is everywhere
  - Service → Memcached router
  - NGINX → FastCGI
  - Kernel → Secure modules
  - App → µkernel services



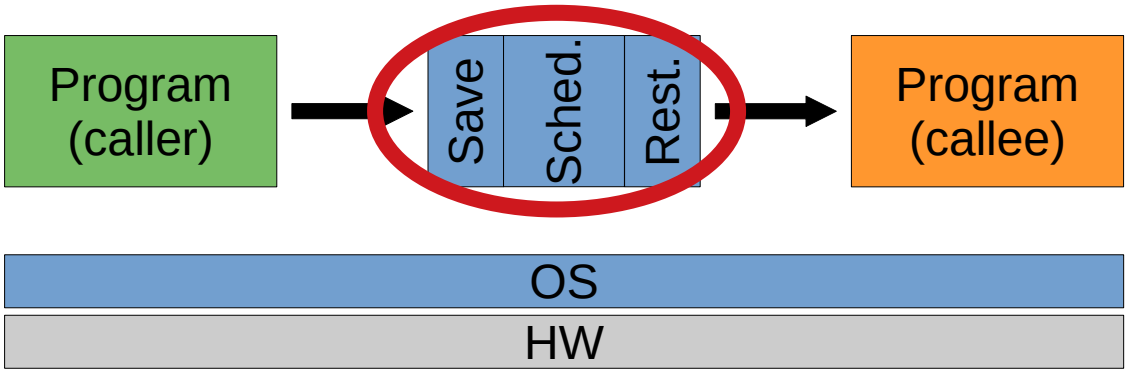
Program Isolation Trade-offs

- Isolation is everywhere
  - Service → Memcached router
  - NGINX → FastCGI
  - Kernel → Secure modules
  - App → µkernel services



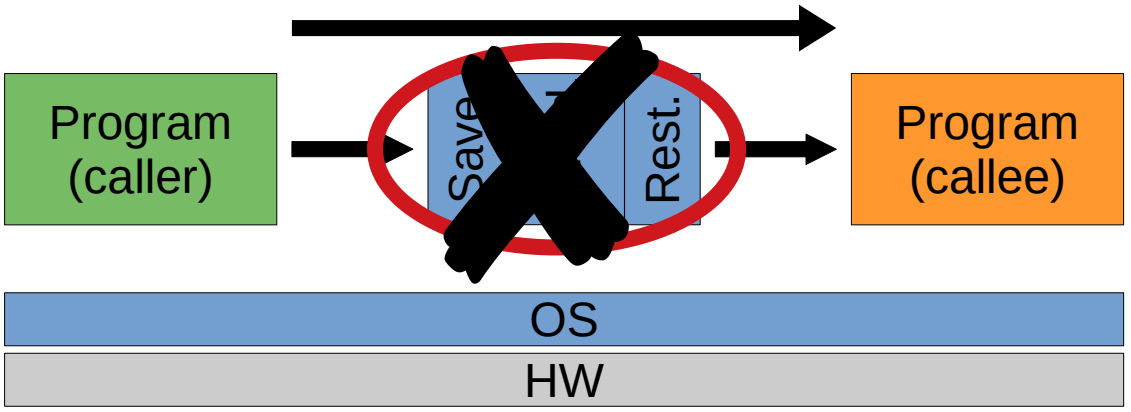
Program Isolation Trade-offs

- Isolation is everywhere
  - Service → Memcached router
  - NGINX → FastCGI
  - Kernel → Secure modules
  - App → µkernel services

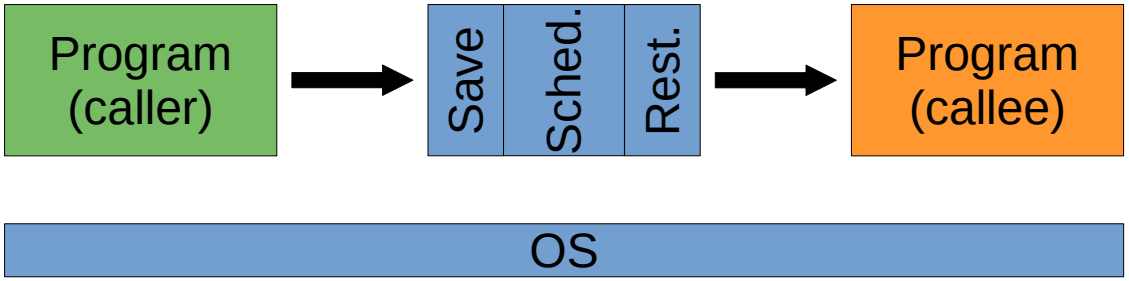


Program Isolation Trade-offs

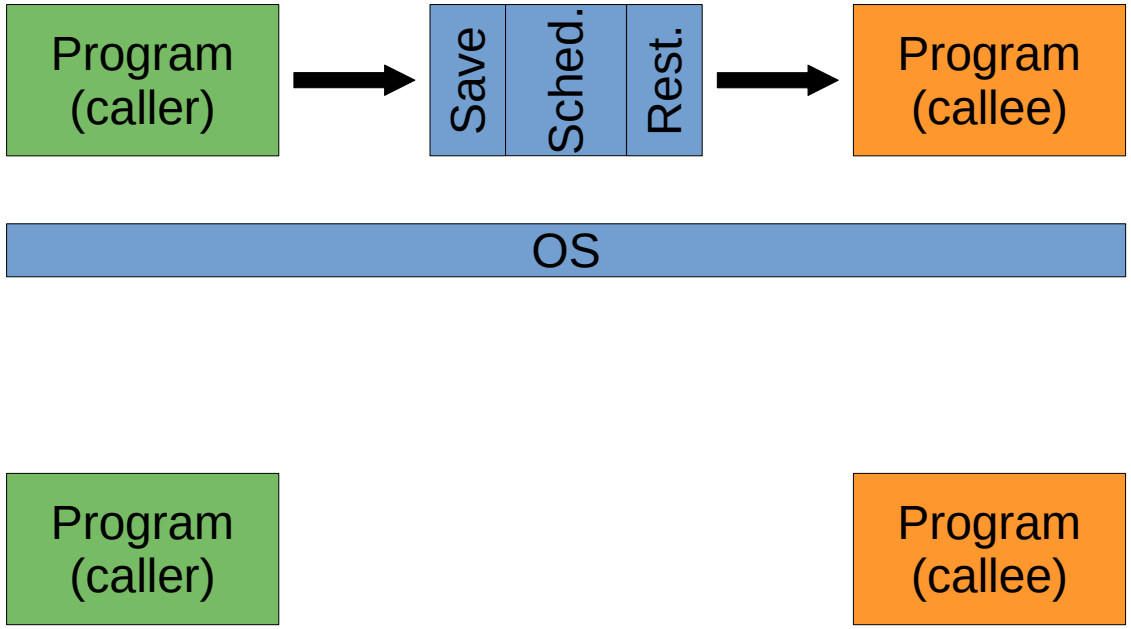
- Isolation is everywhere
  - Service → Memcached router
  - NGINX → FastCGI
  - Kernel → Secure modules
  - App → µkernel services
- **Problem**: OS always mediates (context switch + data copies)
- **Goal**: bypass OS, stay secure



System Overview *[dIPC]*

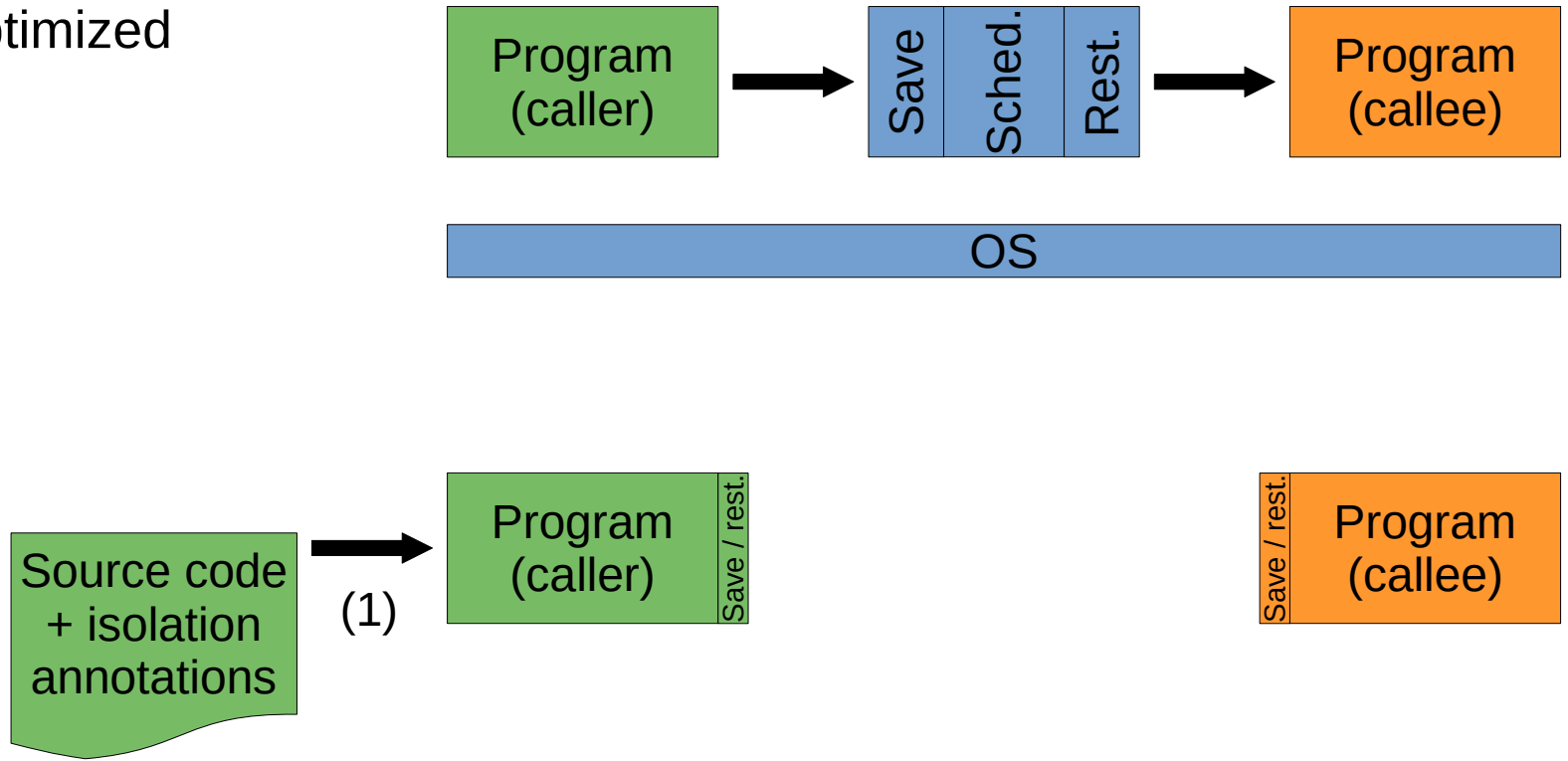


System Overview *[dIPC]*



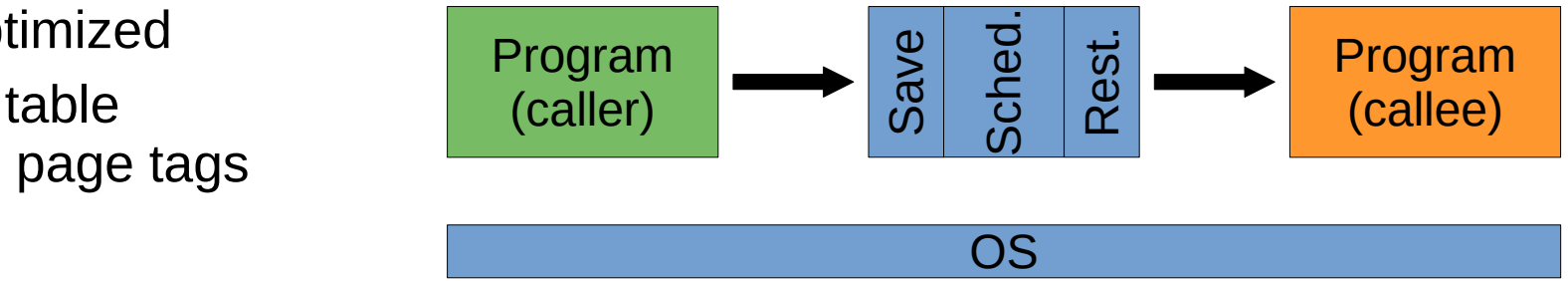
System Overview *[dIPC]*

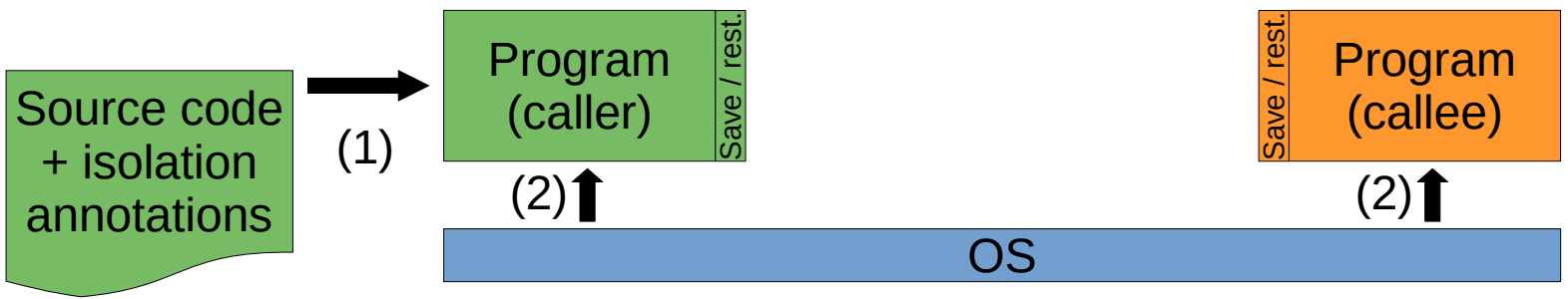
1) User-directed isolation
   - Compiler-optimized



System Overview *[dIPC]*

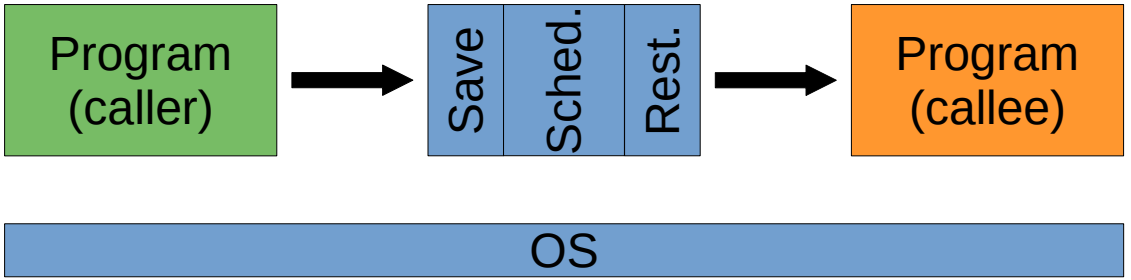
1) User-directed isolation
   - Compiler-optimized
2) Shared page table
   - Per-process page tags

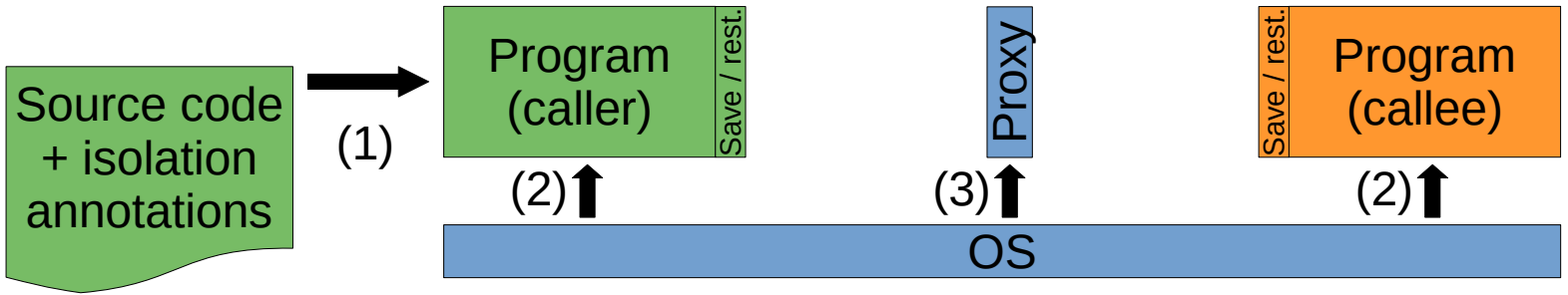




System Overview *[dIPC]*

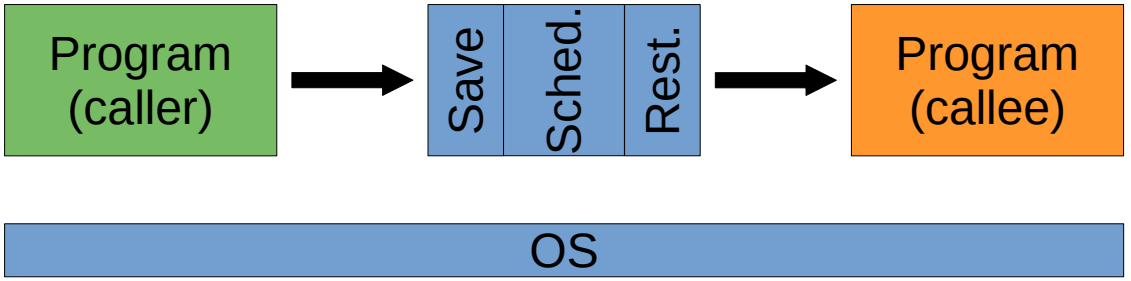
1) User-directed isolation
   - Compiler-optimized
2) Shared page table
   - Per-process page tags
3) Tiny proxy to track process
   - Runtime-optimized (policies)



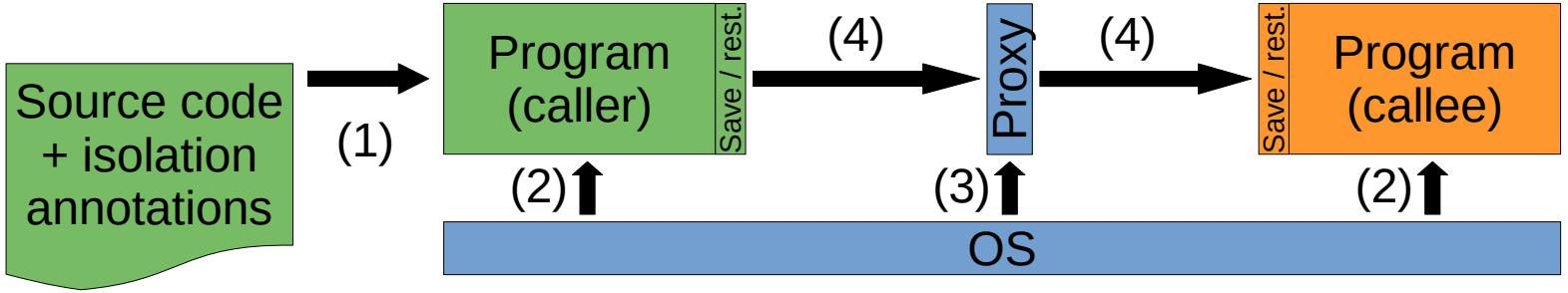


System Overview *[dIPC]*

1) User-directed isolation
   - Compiler-optimized
2) Shared page table
   - Per-process page tags
3) Tiny proxy to track process
   - Runtime-optimized (policies)



4) Direct function call across processes
   - HW memory capabilities to “pass-by-reference”



Memory Capabilities

- Unforgeable "*fat pointers*" with permissions, protec

Base address	Size	{Read, Write, Exec, ...}

  - Identify a memory region (integer, array, etc)
  - Can be passed via registers and memory
  - *If I have access to memory, I can pass it along*

Memory Capabilities

- Unforgeable "*fat pointers*" with permissions, protected by HW

Base address	Size	{Read, Write, Exec, ...}

  - Identify a memory region (integer, array, etc)
  - Can be passed via registers and memory
  - *If I have access to memory, I can pass it along*
- Revocation (i.e., invalidation) is traditionally problematic
  - Traditionally: forbid OR garbage-collect OR indirection table
  - Tie capabilities to scopes (stack frames) → zero-cost revocation!

Results

- ***Kernel module isolation***
  - Full isolation: 0,1% - 0,15% overhead
  - Light-weight policy: 0,03% - 0,05%
  - Scope-based revocation covers >98% of cases

Results

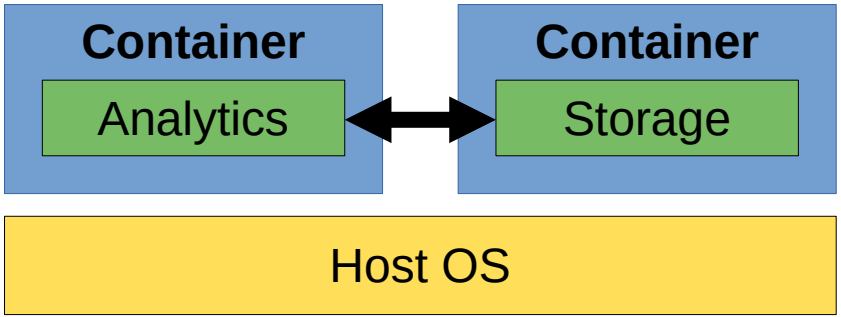
- ***Kernel module isolation***
  - Full isolation: 0,1% - 0,15% overhead
  - Light-weight policy: 0,03% - 0,05%
  - Scope-based revocation covers >98% of cases
- ***Web server + app + DB***
  - Libraries in a single process: 100% efficiency
  - Vanilla Linux: 19% – 40%
  - OS bypass with dIPC: 97% – 98%

Communication in Containers

[CAP-VMs, OSDI'22] CAP-VMs: Capability-Based Isolation and Sharing in the Cloud

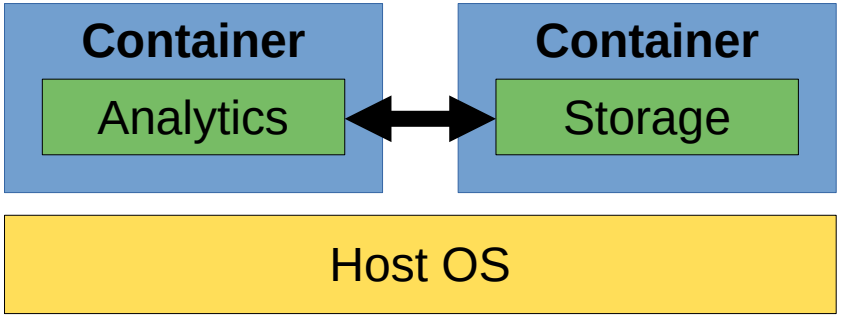
Modern Containers

- Lots of **communication** between cloud containers



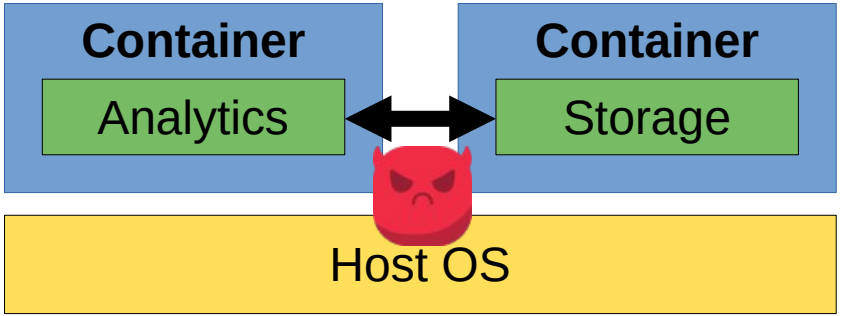
Modern Containers

- Lots of **communication** between cloud containers
- **Light-weight** OS virtualization
  - Efficient communication compared to hardware VMs

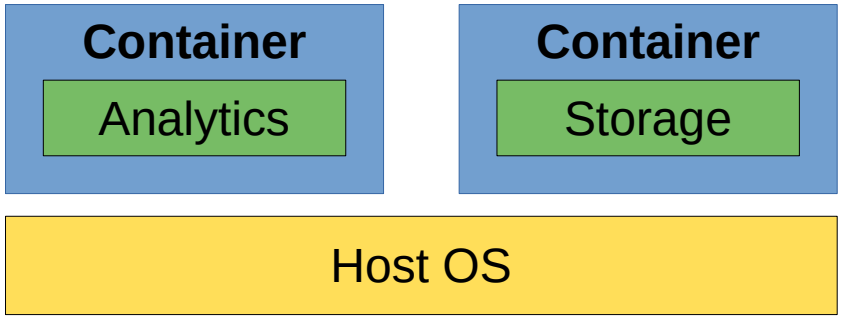


Modern Containers

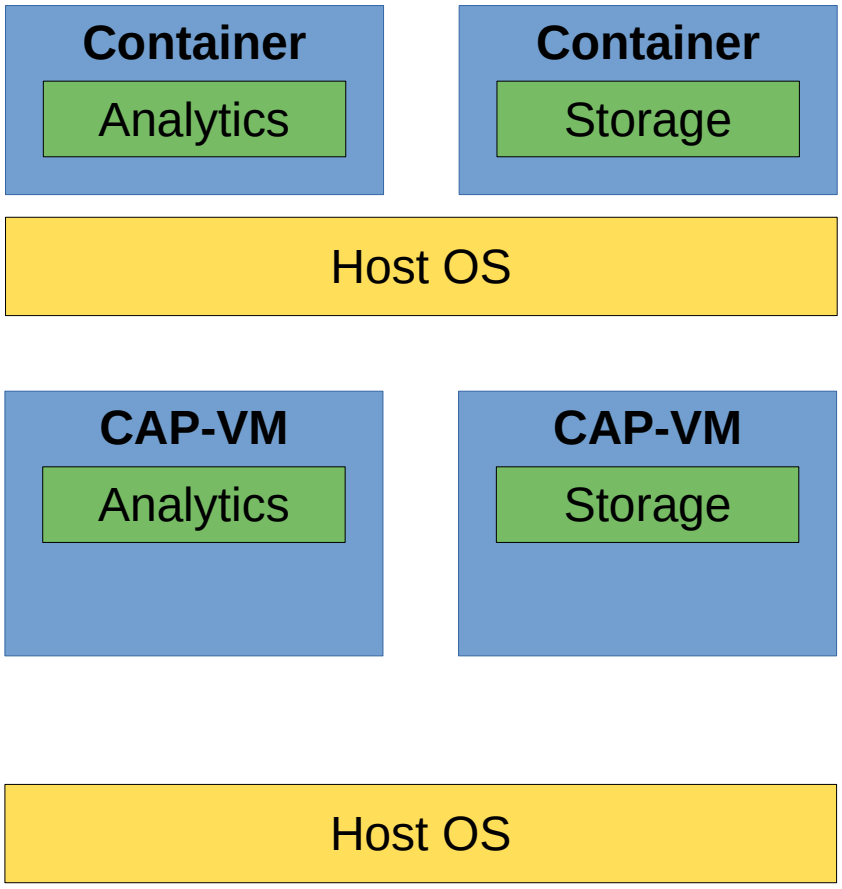
- Lots of **communication** between cloud containers
- **Light-weight** OS virtualization
  - Efficient communication compared to hardware VMs
- **Shared** host OS
  - Very large codebase, hard to secure across tenants



System Overview *[CAP-VMs]*



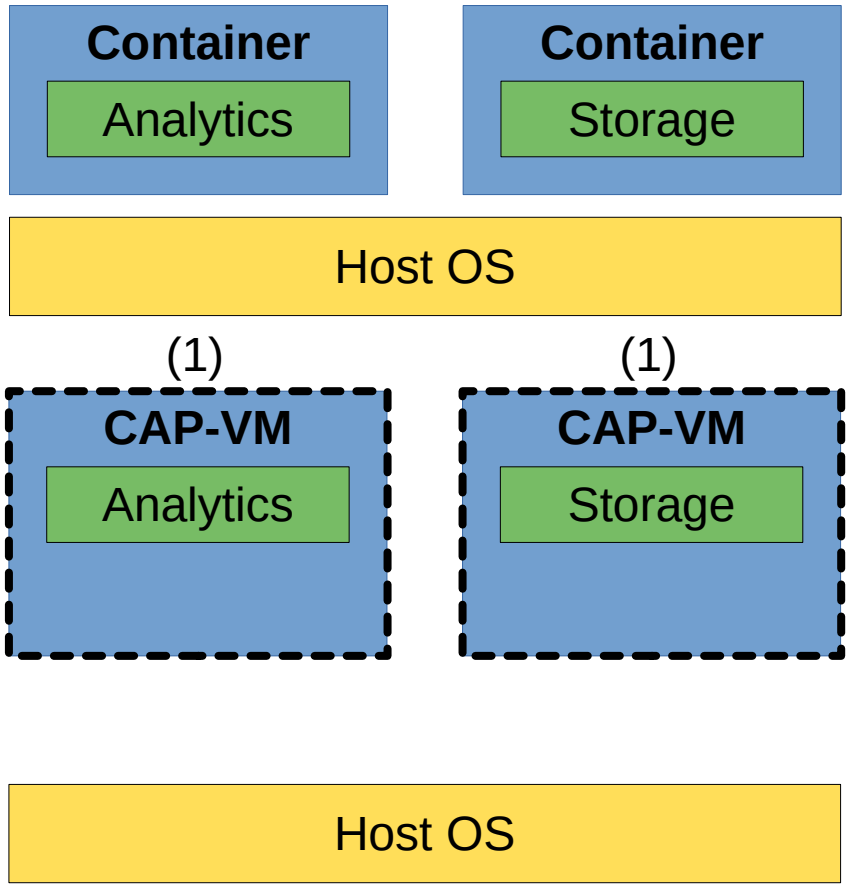
System Overview *[CAP-VMs]*



System Overview *[CAP-VMs]*

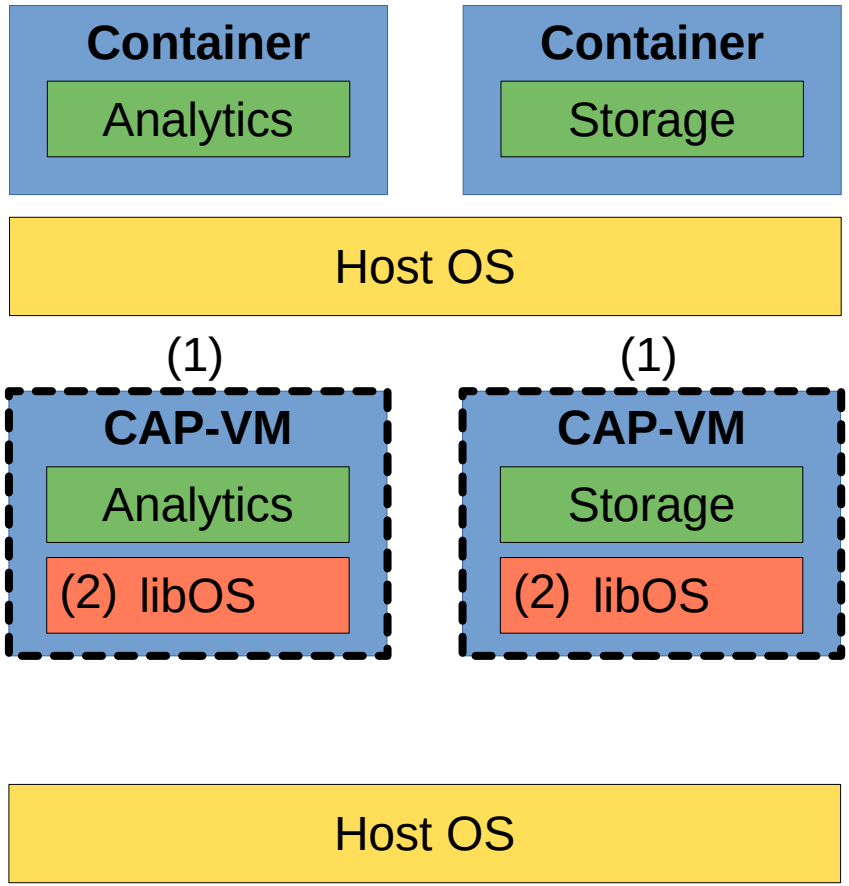
1) Per-container **strong isolation**

- Shared page table, delimited by CHERI default capabilities (similar to i386 default segments)
- Transparent to applications



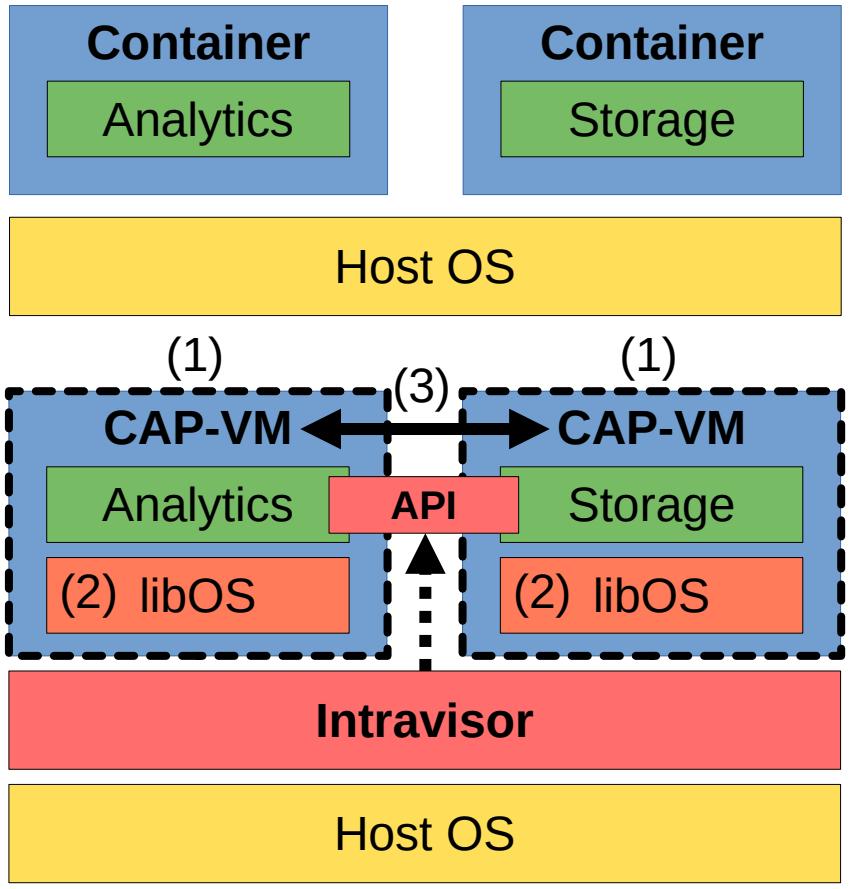
System Overview *[CAP-VMs]*

1) Per-container **strong isolation**
   - Shared page table, delimited by CHERI default capabilities (similar to i386 default segments)
   - Transparent to applications
2) Per-container "**library OS**"
   - LKL (vanilla Linux as a library)
   - Per-container OS, trivially isolated



System Overview *[CAP-VMs]*

1) Per-container **strong isolation**
   - Shared page table, delimited by CHERI default capabilities (similar to i386 default segments)
   - Transparent to applications
2) Per-container "**library OS**"
   - LKL (vanilla Linux as a library)
   - Per-container OS, trivially isolated
3) Message passing **API**
   - Asynchronous buffer, file, and call APIs
   - Controlled by tiny trusted intravisor
   - Similar to dIPC, but no visible capabilities



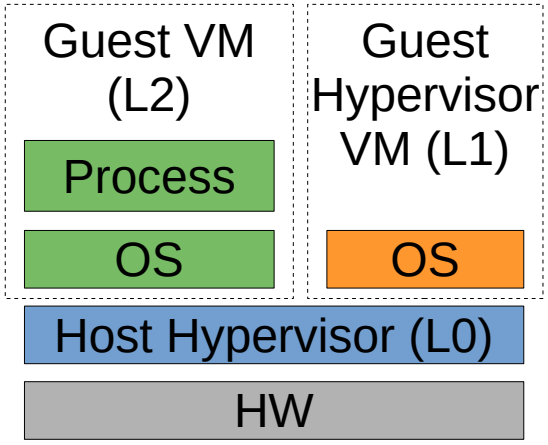
Results

- Key-value-store
  - NGINX + Redis
  - Docker + TCP/IP
    vs
    CAP-VMs
  - 1.5x throughput
    at 95th percentile latency
- Compartmentalization
  - CPython +
    libPyCryptoDome.so
  - Better security within container
  - Negligible performance impact

Communication in VMs

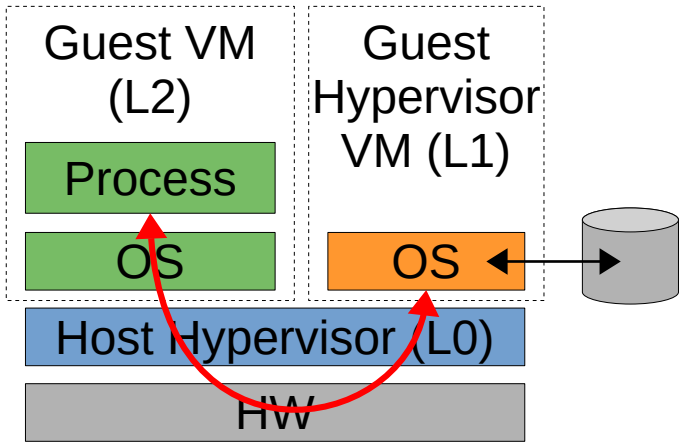
[SVT, ISCA'19] Using SMT to accelerate nested virtualization

From Processes to (Nested) VMs



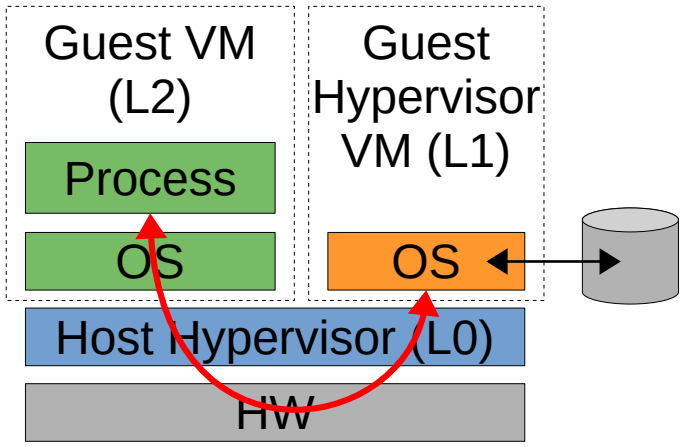
- VMs are today's tenant isolation units
- *Nested virtualization* is the next frontier
  - HV+VMs on virtualized data centers

From Processes to (Nested) VMs

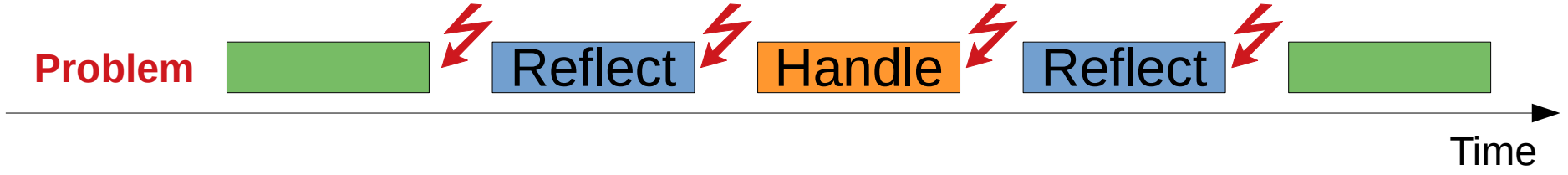


- VMs are today's tenant isolation units
- *Nested virtualization* is the next frontier
  - HV+VMs on virtualized data centers
- **Problem: device access is mediated**
  - **Save/restore large register context**

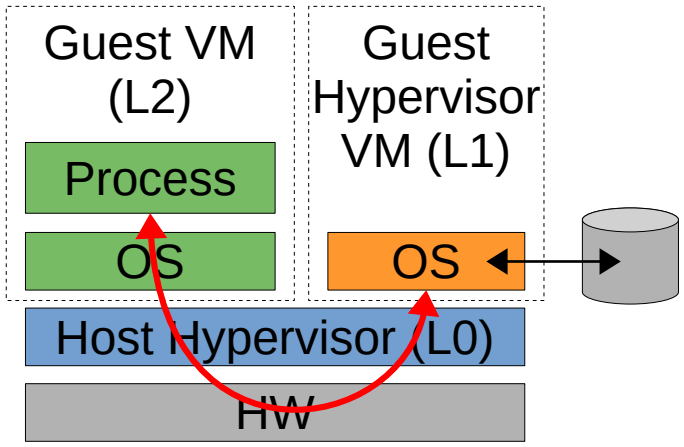
From Processes to (Nested) VMs



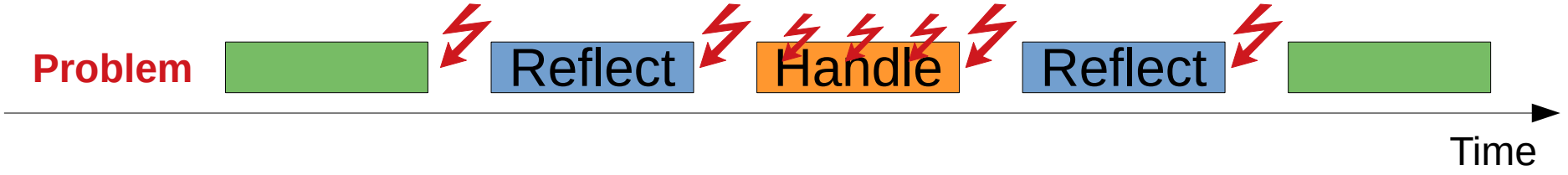
- VMs are today's tenant isolation units
- *Nested virtualization* is the next frontier
  - HV+VMs on virtualized data centers
- **Problem: device access is mediated**
  - **Save/restore large register context**
  - **≥ 2x context switches**



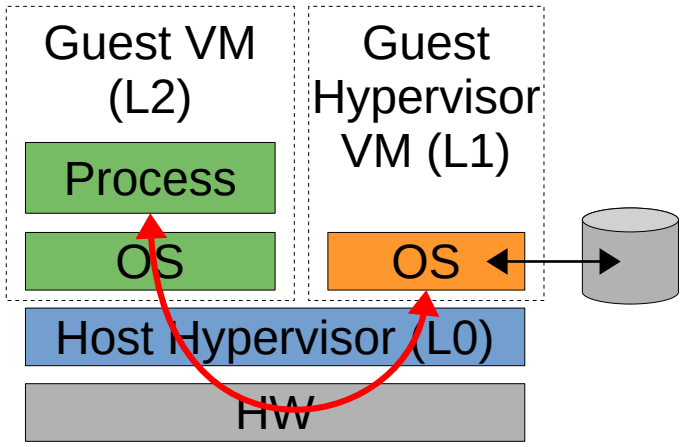
From Processes to (Nested) VMs



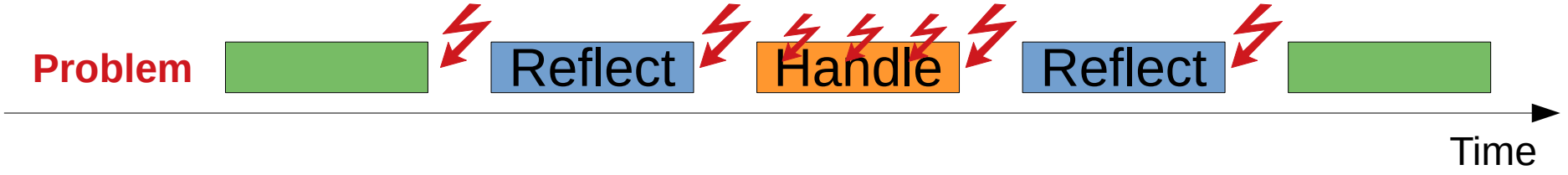
- VMs are today's tenant isolation units
- *Nested virtualization* is the next frontier
  - HV+VMs on virtualized data centers
- **Problem: device access is mediated**
  - **Save/restore large register context**
  - **≥ 2x context switches**



From Processes to (Nested) VMs

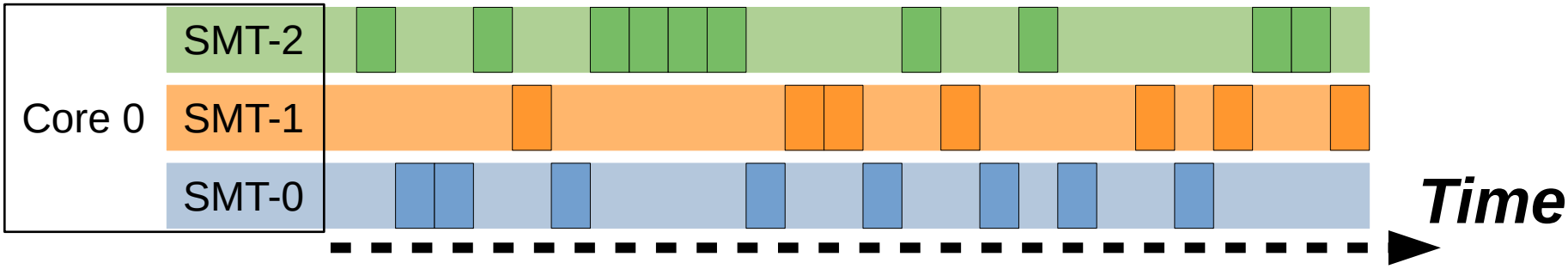


- VMs are today's tenant isolation units
- *Nested virtualization* is the next frontier
  - HV+VMs on virtualized data centers
- **Problem: device access is mediated**
  - **Save/restore large register context**
  - **≥ 2x context switches**
- **Goal: multiple contexts in HW**



Keeping Multiple Contexts in HW

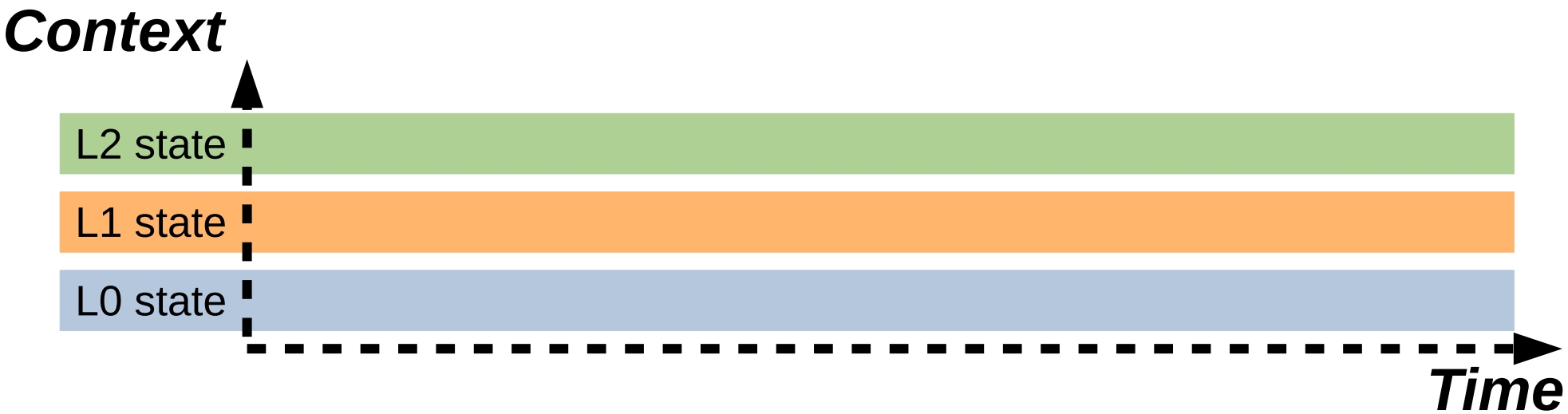
- ***Observation***: Multi-threading has multiple contexts in HW and per-cycle context switches



System Overview *[SVT]*

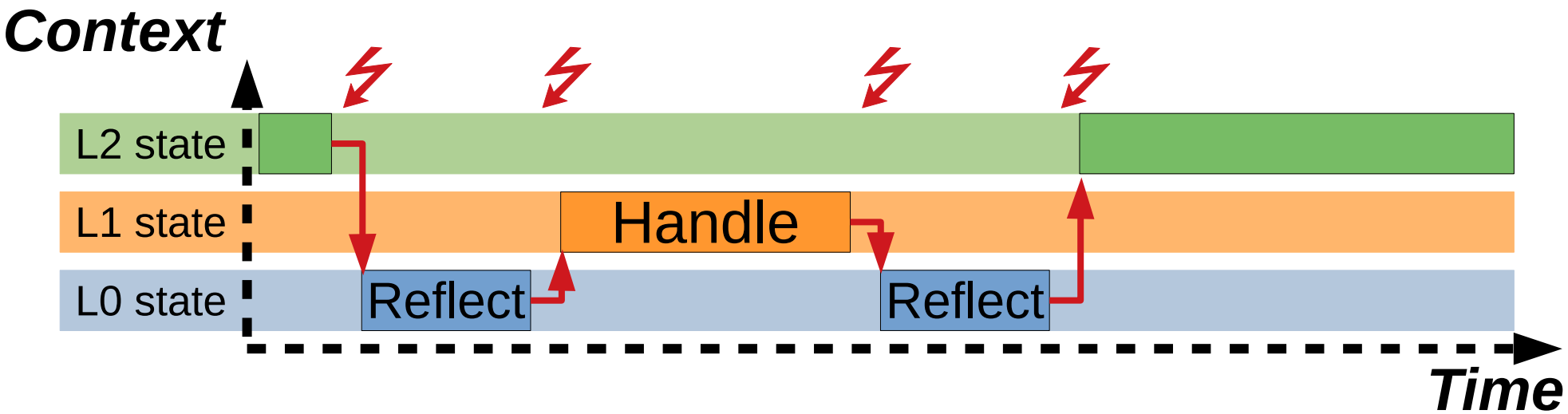
(1) Load L0 / L1 / L2 on ***separate HW contexts***

Only one context executing at a time!



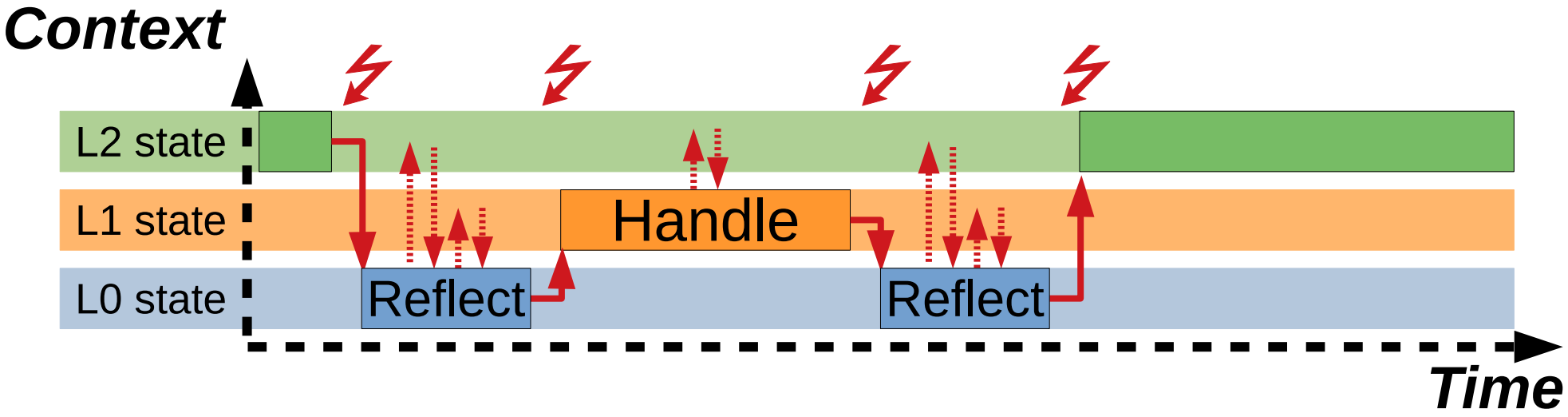
System Overview

(2) Switch instruction fetch on **trap/resume**



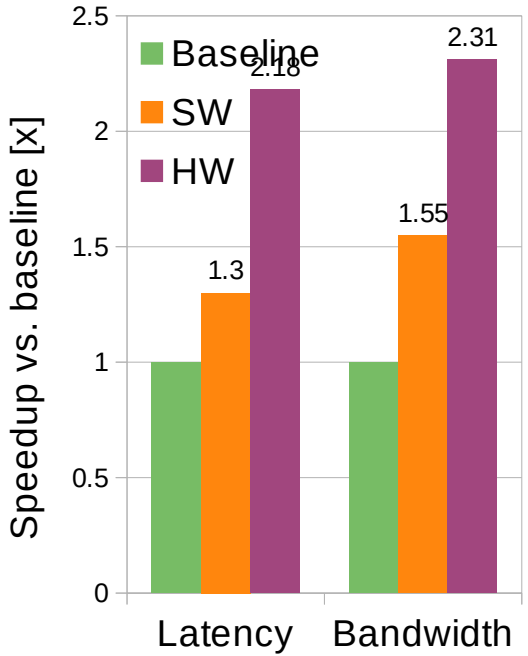
System Overview

(3) Extend ISA to ***access context*** of subordinate VMs (shared physical register file)



Results

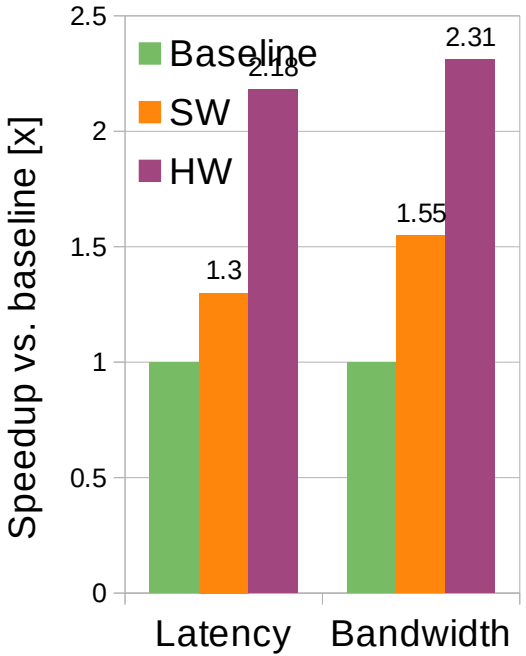
- Disk I/O (rand read)



- SW prototype: 1.55x
- HW model: >2x

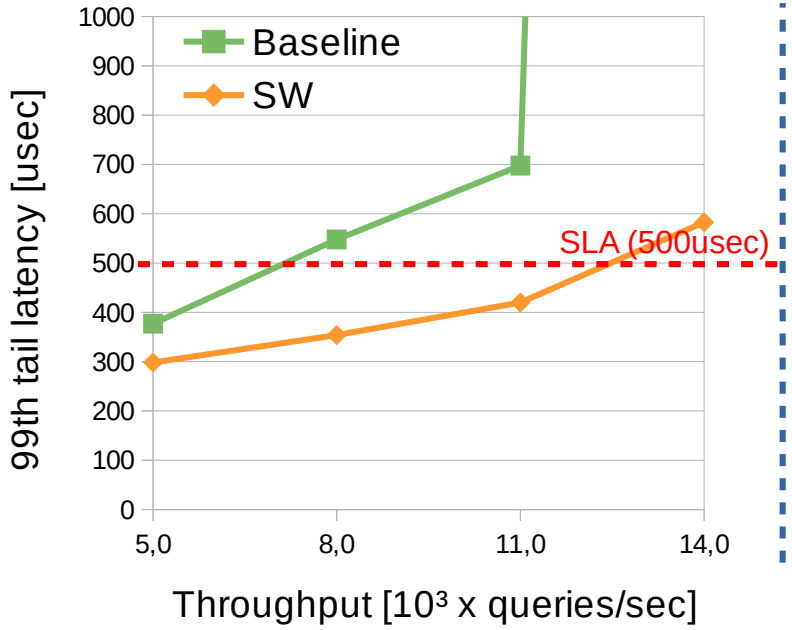
Results

- Disk I/O (rand read)



  - SW prototype: 1.55x
  - HW model: >2x

- Memcached latency



  - 2.20x throughput within SLA

Conclusions

- Isolation is a must, but brings communication overheads everywhere
  - Libraries, processes, containers, VMs, data center nodes, ...
- ~~Breaking~~ Co-designing the layers
  - Rethink separation of concerns across HW and SW layers
  - Increase performance **and** isolation
- Exciting opportunities
  - Lots of vertical integration in HPC
  - Bypass “one-size-fits-all” solutions for heterogeneous HW *[FractOS @ EuroSys’22]*
  - End-to-end solutions happening in more spaces: cloud, automotive, …

Lluís Vilanova
<vilanova@imperial.ac.uk>