Inside Out: A Paradigm Shift In Live VM Introspection

Dufy Teguia[1,2], Louis Duval[1], Teo Pisenti[3], Kahina Lazri[2], Daniel Hagimont[3], Thomas Pasquier[4], Renaud Lachaize[1], Alain Tchana[1]

[1] Univ. Grenoble Alpes, CNRS, Inria, Grenoble INP, LIG
[2] Orange Research
[3] IRIT, Université de Toulouse, CNRS, Toulouse INP, UT3 Toulouse
[4] University of British Columbia

August 3, 2026









THE UNIVERSITY
OF BRITISH COLUMBIA



Virtual Machine Introspection (VMI)

- VMI reconstructs a VM's internal state by observing it from the outside
- The observed VM is called **target**
- Three use cases: *post-mortem* (forensics), *security operations* (live), *program observation* (live)

Scope

We focus on **live** VMI, valuable to both cloud tenants and providers.

Tenant side scenarios

- Real-time threat detection & prevention
- Memory dumps for evidence gathering
- Live kernel debugging / tracing

Provider side scenarios

- Security-as-a-Service (e.g. ransomware detection)
- Resource optimization (memory overcommit)

orange™

Virtual Machine Introspection (VMI)

- VMI reconstructs a VM's internal state by observing it from the outside
- The observed VM is called **target**
- Three use cases: *post-mortem* (forensics), *security operations* (live), *program observation* (live)

Scope

We focus on **live** VMI, valuable to both cloud tenants and providers.

Tenant side scenarios

- Real-time threat detection & prevention
- Memory dumps for evidence gathering
- Live kernel debugging / tracing

Provider side scenarios

- Security-as-a-Service (e.g. ransomware detection)
- Resource optimization (memory overcommit)

orange™

Virtual Machine Introspection (VMI)

- VMI reconstructs a VM's internal state by observing it from the outside
- The observed VM is called **target**
- Three use cases: *post-mortem* (forensics), *security operations* (live), *program observation* (live)

Scope

We focus on **live** VMI, valuable to both cloud tenants and providers.

Tenant side scenarios

- Real-time threat detection & prevention
- Memory dumps for evidence gathering
- Live kernel debugging / tracing

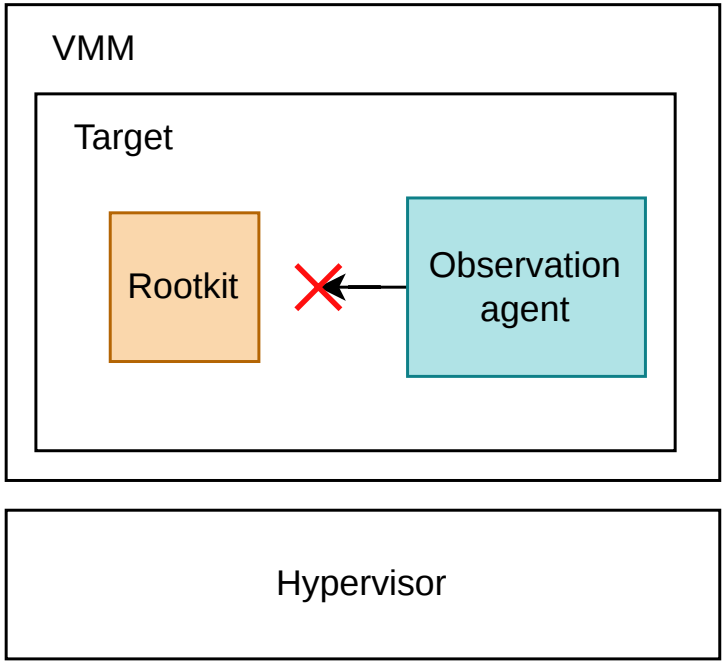
Provider side scenarios

- Security-as-a-Service (e.g. ransomware detection)
- Resource optimization (memory overcommit)

orange™

Motivation: Malware Hiding in the Guest

- Advanced malware hides inside the guest OS by manipulating kernel structures
- Once the kernel is compromised, in-guest agents can no longer be trusted
- Other use cases also require an external observation of a VM



Take-away

We need an observation agent that is **isolated** from the target.

Challenges of Live VMI

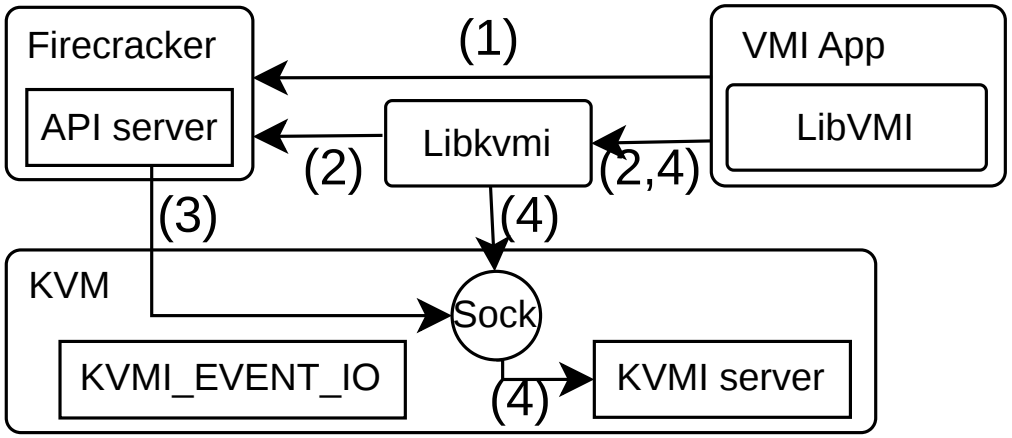
1. **Semantic gap**: reconstruct high-level state from raw binary data
2. **Memory-access consistency**: read target memory concurrently and safely
3. **Performance**: minimize service disruption on the target

Take-away

GOODKIT is designed to address these challenges without giving up isolation.

orange™

Existing Solutions : LibVMI



Overview

LibVMI is a userspace library that interacts with KVM through a socket to gather introspection data.

Existing Solutions

In-guest agents [Chiueh+, CC GRID 12'], [Ibrahim+, NSS 11']

- Deployed *inside* the target
- Share the execution context of the monitored programs

⇒ Not stealthy; untrusted once the kernel is compromised

⇒ A faulty observer may disrupt the target

LibVMI-based [Dolan-Gavit+, S&P'11], [Schwarz+, USENIX Sec'24]

- Introspection logic lives entirely *outside*
- Talks to the hypervisor via standard libKVMI

⇒ Not isolated from the cloud provider

⇒ Rely on global VM pauses

orange™

Existing Solutions

In-guest agents [Chiueh+, CC GRID 12'], [Ibrahim+, NSS 11']

- Deployed *inside* the target
- Share the execution context of the monitored programs

⇒ Not stealthy; untrusted once the kernel is compromised

⇒ A faulty observer may disrupt the target

LibVMI-based [Dolan-Gavit+, S&P'11], [Schwarz+, USENIX Sec'24]

- Introspection logic lives entirely *outside*
- Talks to the hypervisor via standard libKVMI

⇒ Not isolated from the cloud provider

⇒ Rely on global VM pauses

orange

Limitations of Existing Solutions

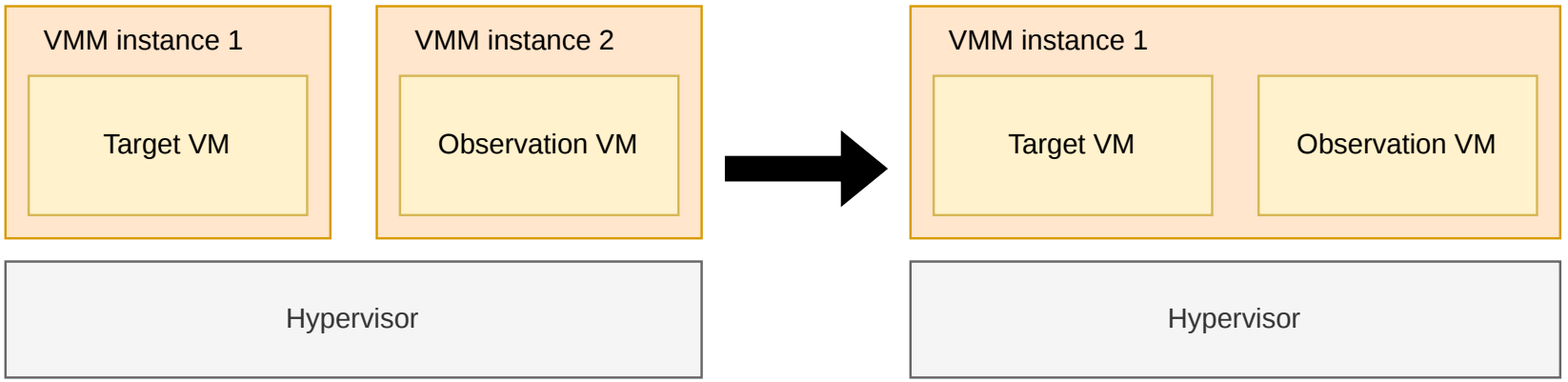
Approach	Target Isolation	Performance	Main drawback
In-guest agents	None	Native	Fate sharing
LibVMI-based	Strong	Low	Heavy VM pausing
GOODKIT (ours)	Medium	High	Enlarges TCB (VMM only)

Take-away

GOODKIT trades a slightly larger TCB (**limited to the VMM**) for both isolation *and* performance.

orange™

GOODKIT: Overview



Key idea

Observers are distinct **side-VMs**, launched alongside the target *in the same VMM instance*.

VMM colocation gives native access to target memory and controlled resource access, with no hypervisor modification.

GOODKIT: Overview

A framework for building VMI observation agents, called **observers**.
Design goals:

- **Isolation**: from the target *and* the provider
- **Efficiency**: low observer resource use
- **Consistency**: reliable access to target memory
- **Speed**: near native, low overhead
- **Flexibility**: many use cases

orange™

Three Design Challenges

GOODKIT must solve three challenges:

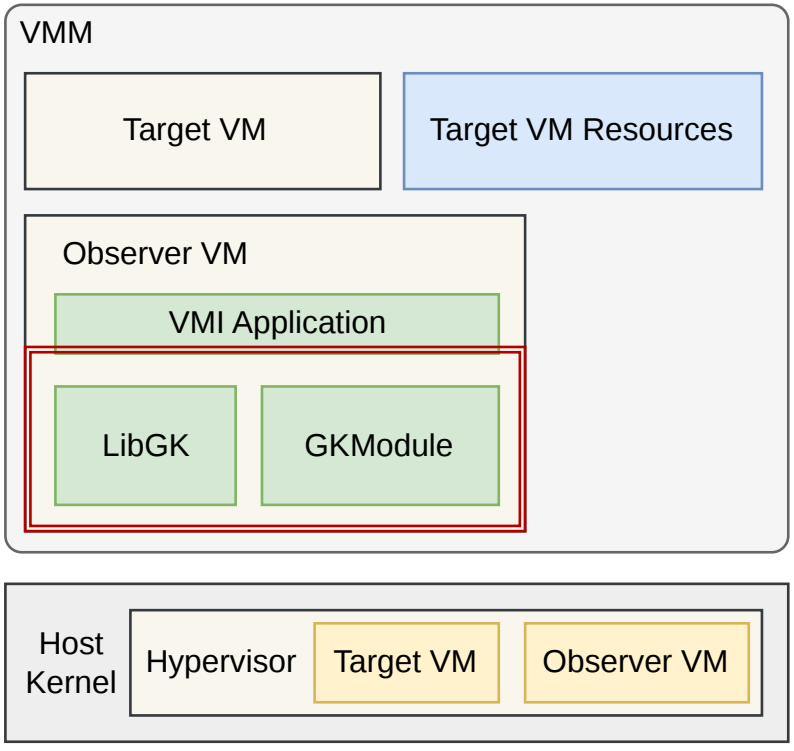
1. **Observer–VMM communication**: configuration, resource access, logging
2. **Consistent memory access**: ensure consistent and reliable memory access
3. **Multi-observer synchronization**: coordinate concurrent observers

orange™

GOODKIT: Architecture

Components:

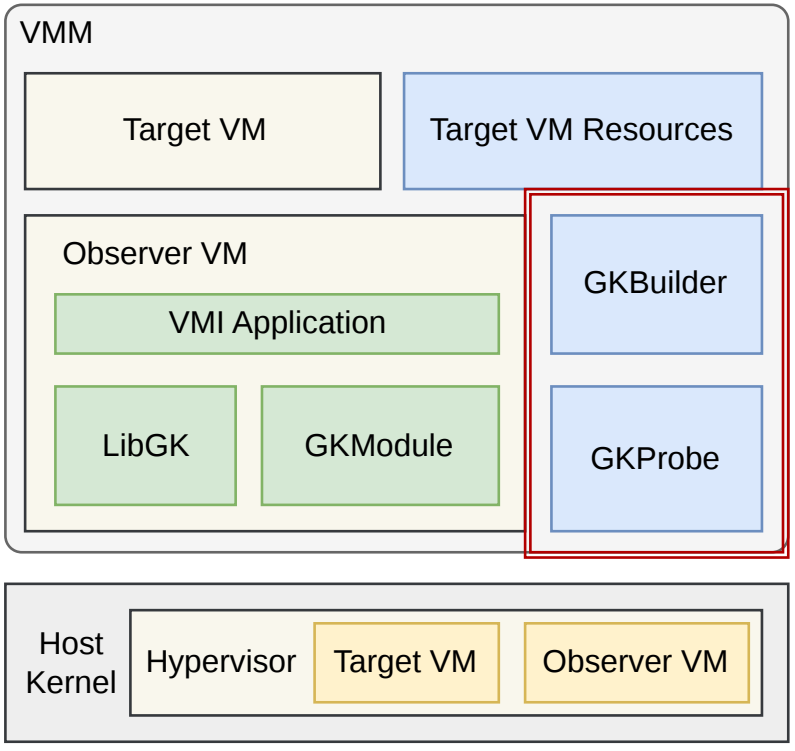
- Frontend in the observer guest: `LibGK`, `GKModule`
- Backend in the VMM: `GKBuilder`, `GKProbe`
- Observer $\leftrightarrow$ VMM channel over **virtIO**
- Dedicated in-observer and in-VMM observability mechanisms



GOODKIT: Architecture

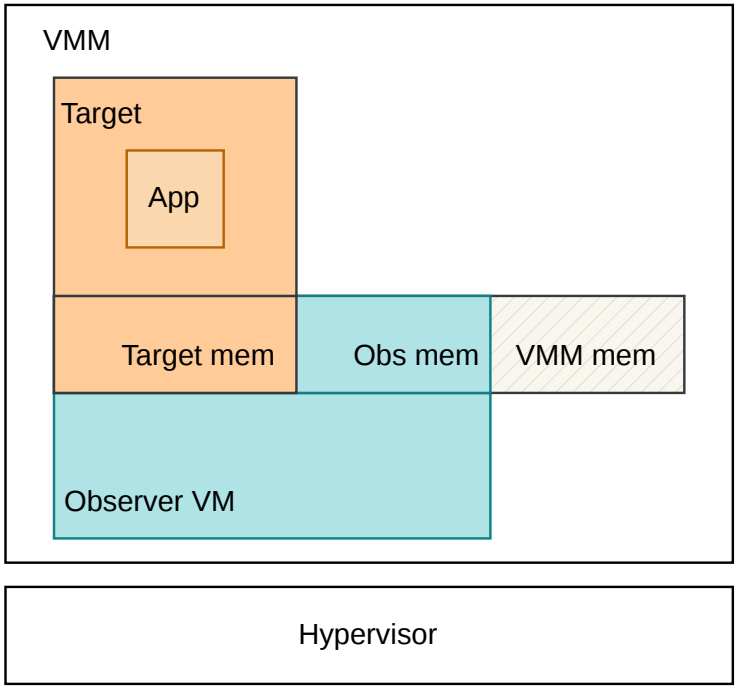
Components:

- Frontend in the observer guest: `LibGK`, `GKModule`
- Backend in the VMM: `GKBuilder`, `GKProbe`
- Observer $\leftrightarrow$ VMM channel over **virtIO**
- Dedicated in-observer and in-VMM observability mechanisms



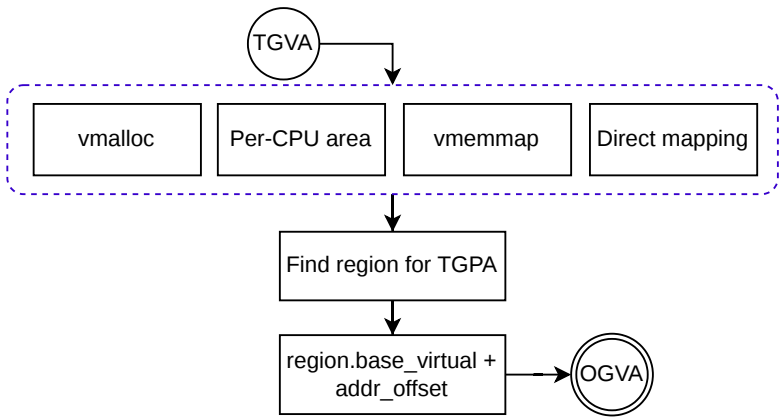
Consistent Memory Access — Mapping

- **Live memory mapping**: the target's memory is mapped straight into the observer's address space



Consistent Memory Access — Translation

- The observer gets a new **logical view**: the target's physical layout as a set of regions (`base, size`)
- Accessing data needs one extra translation: TGVA → TGPA → OGVA



TGVA/TGPA = Target Guest Virtual/Physical Address
OGVA = Observer Guest Virtual Address

Consistent Memory Access — Consistency

The consistency problem

Reading a live VM's memory can return inconsistent or erroneous results.

Prior approaches

- Pause the VM on every memory access
- Generate memory dumps (live or paused)
- Optimistic retry until consistent

Our approach — act as an internal entity of the target

- **Read-only / rarely-modified** data: direct unrestricted read
- **Often or heavily modified** data: acquire target locks (currently spinlocks, RW-locks, mutexes)

E.g. walking the modules list requires acquiring its mutex.

orange™

Consistent Memory Access — Consistency

The consistency problem

Reading a live VM's memory can return inconsistent or erroneous results.

Prior approaches

- Pause the VM on every memory access
- Generate memory dumps (live or paused)
- Optimistic retry until consistent

Our approach — act as an internal entity of the target

- **Read-only / rarely-modified** data: direct unrestricted read
- **Often or heavily modified** data: acquire target locks (currently spinlocks, RW-locks, mutexes)

E.g. walking the modules list requires acquiring its mutex.

orange™

Consistent Memory Access — Consistency

The consistency problem

Reading a live VM's memory can return inconsistent or erroneous results.

Prior approaches

- Pause the VM on every memory access
- Generate memory dumps (live or paused)
- Optimistic retry until consistent

Our approach — act as an internal entity of the target

- **Read-only / rarely-modified** data: direct unrestricted read
- **Often or heavily modified** data: acquire target locks (currently spinlocks, RW-locks, mutexes)

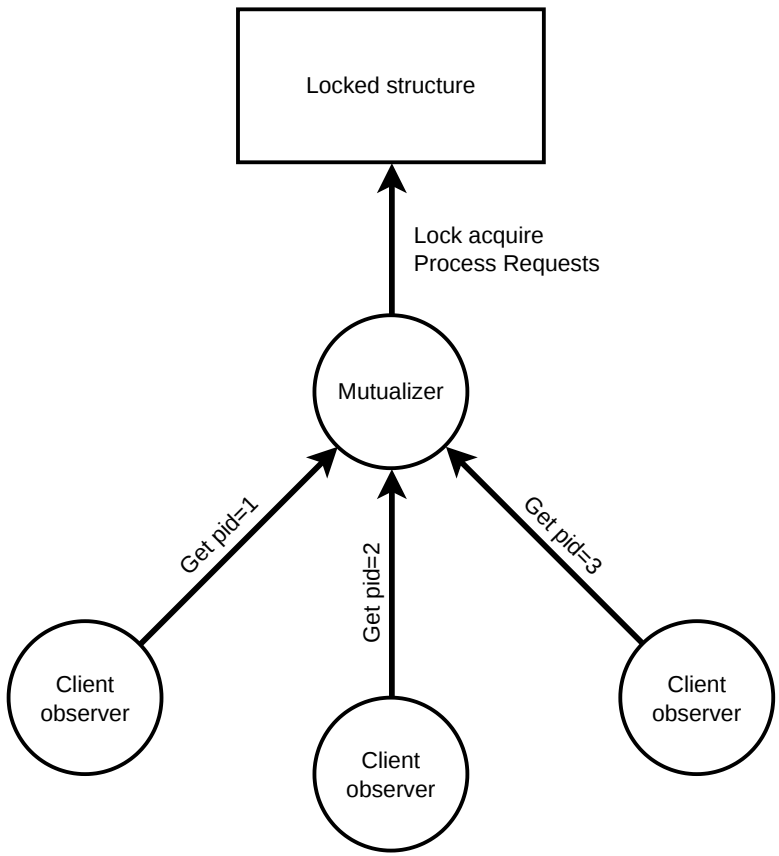
E.g. walking the modules list requires acquiring its mutex.

orange

Mutualized Introspection

- Multiple observers can run for the *same* target
- Concurrent accesses to the same structures contend on the target's locks
- A **mutualizer** serializes access: it takes requests from client observers over **shared memory** and returns the results

More details in the paper.



Evaluation Setup

Testbed

- 16-core Intel Xeon Silver 4215 @ 2.50 GHz, 256 GB RAM
- Host: Ubuntu 20.04, Linux 5.4.24+
- Guest: Alpine, Linux 5.10.198
- VMM: **Firecracker v1.5**

Baseline & workloads

- Baseline: **LibVMI** (KVMI v7)
- Micro-benchmarks (isolate features)
- Macro-benchmark: **14 Phoronix apps**

orange

Results: Implemented Use Cases

We have implemented several use cases to validate the prototype:

- Rootkit detection: Sutekh, Spy, Diamorphine, Adore-ng
- Liveness monitoring via `task_struct` state inspection
- Ransomware detection via block I/O patterns
- Live CFS runqueue monitoring

orange™

Results: Introspection Turnaround Time

ID	Name	ID	Name
P1	Process List	P6	/proc File Operations
P2	System Calls	P7	TTY Check
P3	Credentials List	P8	VM Area Check
P4	Keylogger Check	P9	Open File List
P5	Netfilter Check	Idle	Lock/Unlock or Pause/Resume

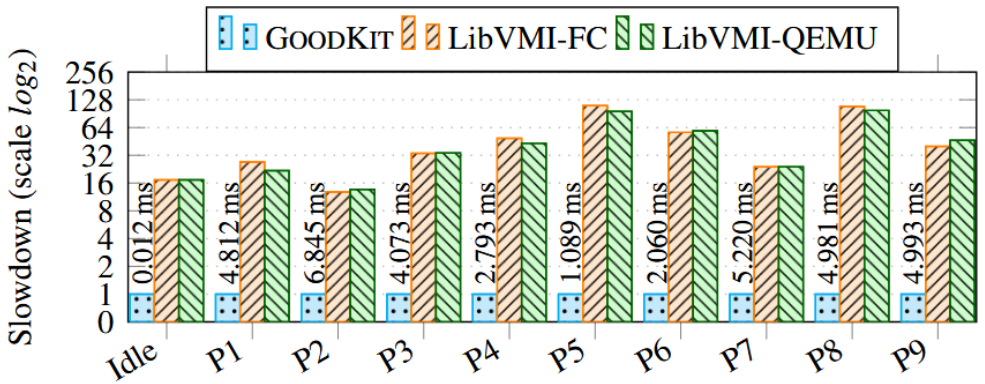


Figure: 1 GOODKIT observer vs. 1 LibVMI observer (Firecracker & QEMU) — **Log$_2$***scale*

Result

LibVMI on idle use case $\approx$ 17$\times$ slowdown;
LibVMI on netfilter check: 110$\times$.

Results: Application Overhead (Phoronix)

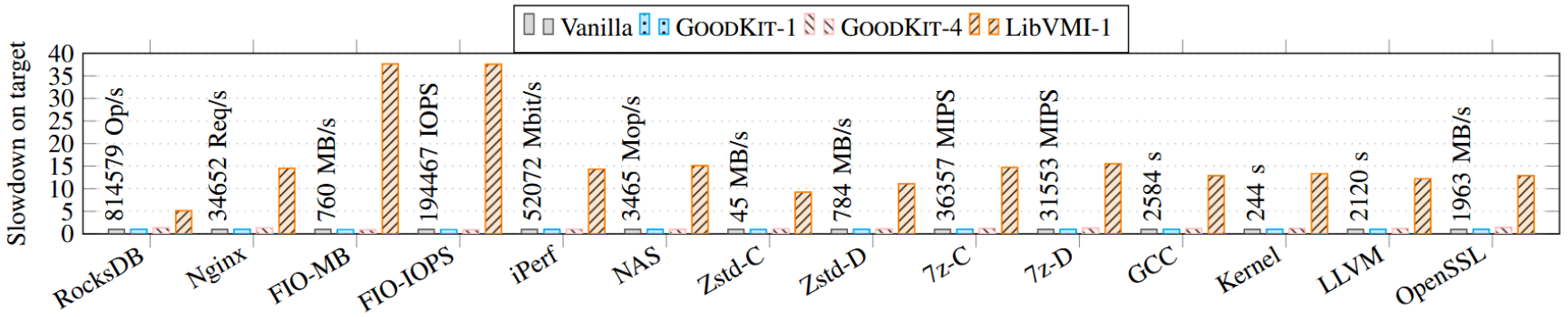


Figure: Overhead of GOODKIT (1 and 4 observers) vs. LibVMI, 14 apps

Result

Single GOODKIT observer is around **1.06**×;
4-observers GOODKIT is **1.16**×;
LibVMI goes from **5**× to **37**×.

More evaluations in our paper.

orange

Conclusion

Live VMI is critical for various use cases for both cloud providers and tenants.

- **Paradigm shift**: In Live VMI, the hypervisor is no longer in the critical path, colocated VMs are used instead.
- **Designed for today's cloud infrastructure** : near native, Isolation, flexibility, scalability.

Contact Us

- Dufy Teguia (dufy.teguia@orange.com)
- Louis Duval (louis.duval1@univ-grenoble-alpes.fr)
- Teo Pisenti (teo.pisenti@etu.toulouse- inp.fr)
- Kahina Lazri (kahina.lazri@orange.com)
- Thomas Pasquier (tfjmp@cs.ubc.ca)
- Daniel Hagimont (daniel.hagimont@irit.fr)
- Renaud Lachaize (renaud.lachaize@univ-grenoble-alpes.fr)
- Alain Tchana (alain.tchana@grenoble-inp.fr)

Thank you!

