Secure Cloud Computing: The Monitoring Perspective

Peng Liu

Penn State University

Cloud Computing is

Less about Computer Design

CPU, OS, VMM, PL, …

Parallel computing

More about Use of Computing (UoC)

Multi-tenant,
Resource consolidation,
Computing as a service,
IT management, …

Cloud Computing = UoC innovations + …

Security issues due to UoC innovations

Isolation and inference channels

Trust-minimizing computing

Accountability

Isolation and inference channels

Physical isolation disables resource consolidation

Logical isolation = physical sharing

Logical isolation leads to **inference channels**

- Explicit data flows
- Implicit information flows
- Covert channels

Trust-minimizing computing

Tenants' apps do not need to trust OS.

Tenants' VMs no need to trust provider's VMMs/hardware.

Tenants' data no need to trust apps.

Accountability

Make data accountable

Make information flows accountable

Make code and control flows accountable

Make SLAs accountable

Security monitoring is essential

Without monitoring, accountability cannot be achieved.

Monitoring plays a critical role in inference control.

What to monitor?

- Data flows
- Control flows
- Information flows
- Data invariants
- Cross the isolation boundaries

State of the art

Coarse-grained monitoring is mature and widely deployed.

Fine-grained monitoring is not very practical.

- Dynamic taint analysis is still offline (3x-100x)
- Inlined monitoring is expensive
- …

Why so hard to make fine-grained monitoring practical?

The Collapse of Moore's Law is a fundamental reason.

> CPU core’s speed “simply cannot maintain its rapid exponential rise using standard silicon technology.”

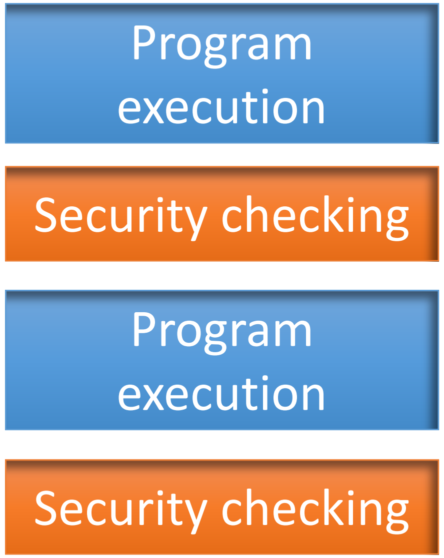
So unless we rewrite an app to do parallel computing, the app’s response time will not decrease in future.

So inlined monitoring will still be a “pain” in future.

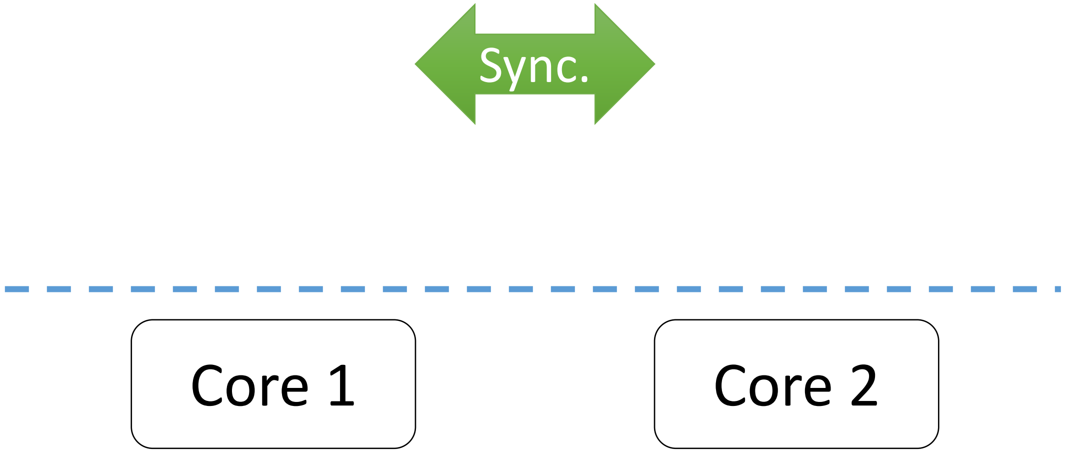
Non-Blocking Concurrent Security Monitoring

-- Let monitor code run on other cores during idle time

Motivation



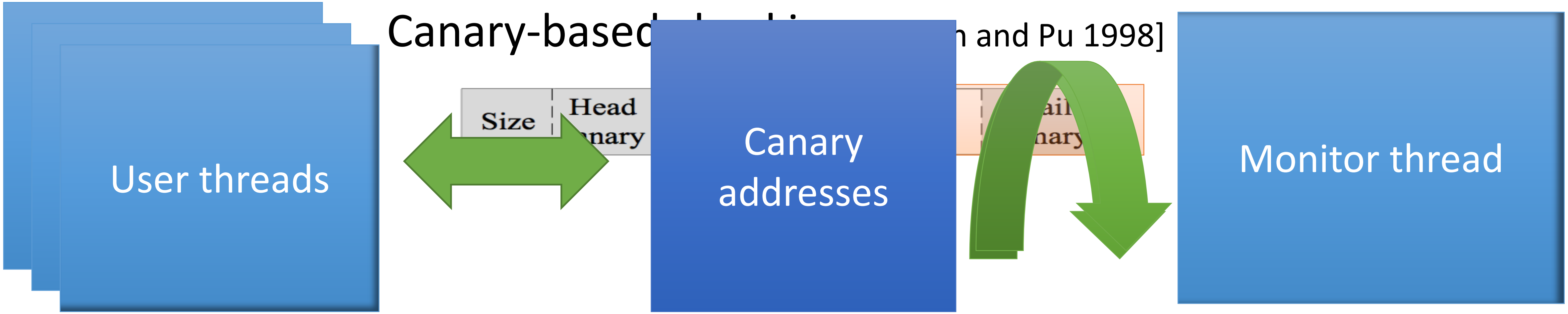
Inlined checking



Concurrent checking

Problem 1: App Heap Buffer Overflow Monitoring

Straightforward (but inefficient) attempts



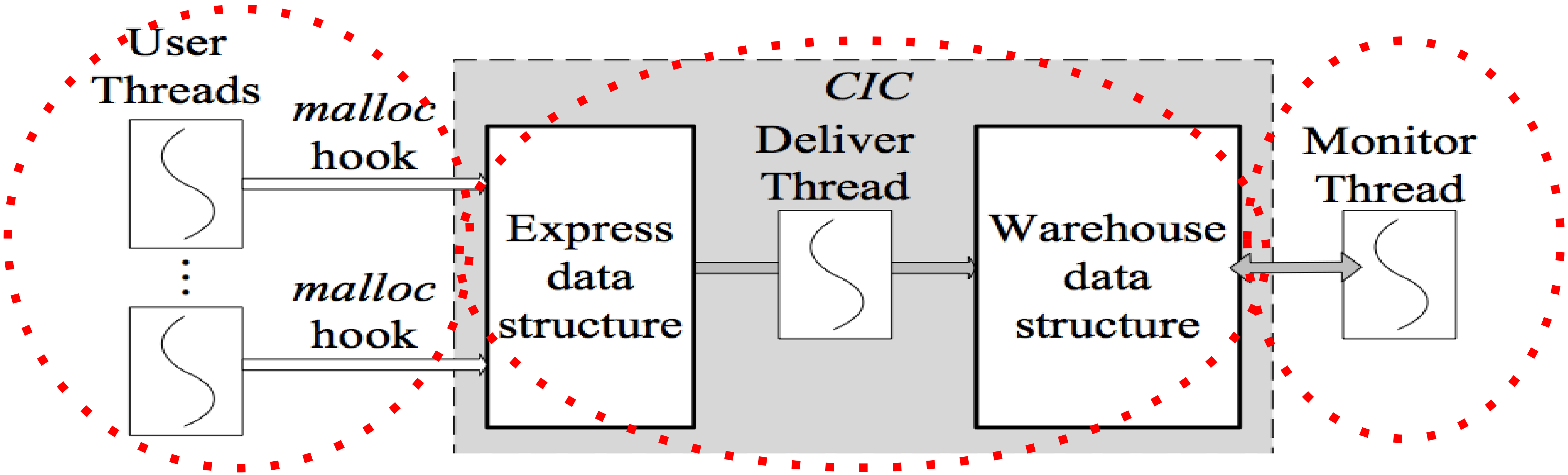
- Attempt1: *Lock-based red-black tree*

Monitoring blocks program execution

- Attempt2: *Lock-free hash table* [Shalev & Shavit 2006].

Complex operations and Contention

Cruiser Architecture



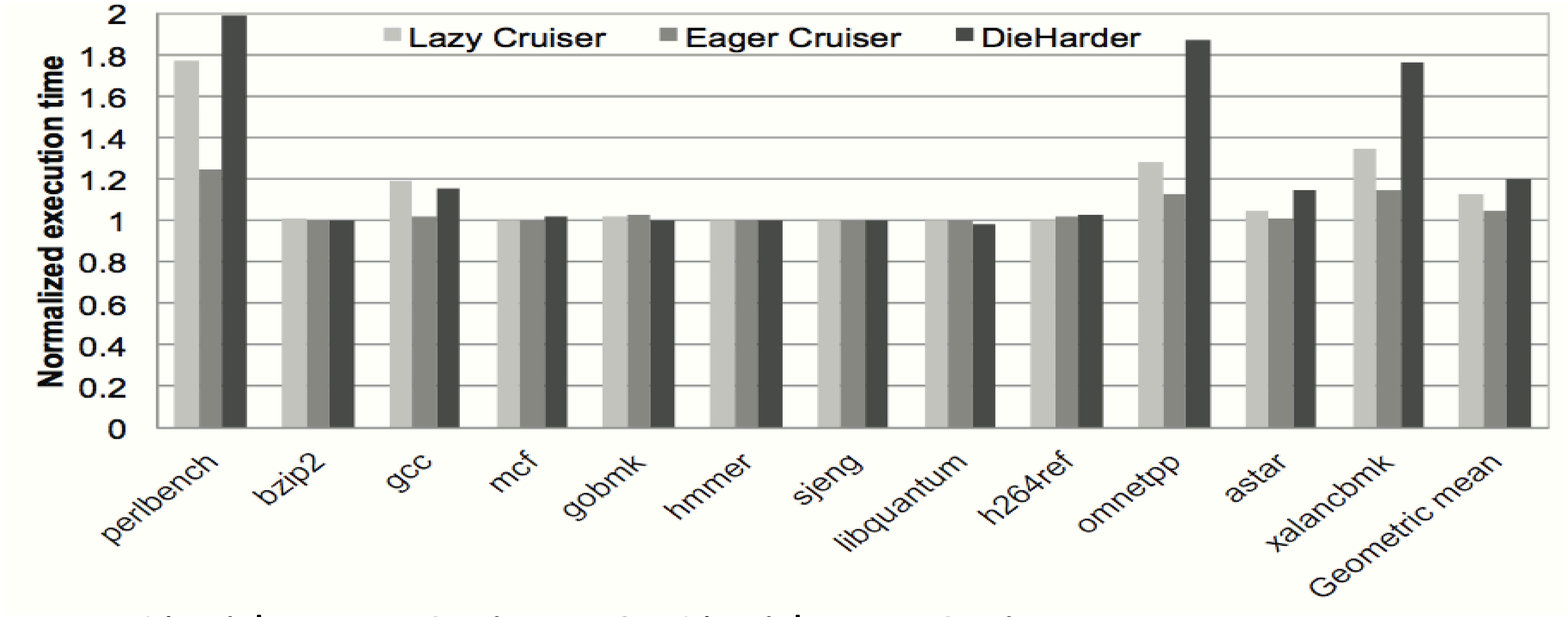
Custom lock-free data structures and non-blocking algorithms to collect canary addresses.

Technical hurdle

Theorems on impossibility of lock-free non-blocking synchronization.

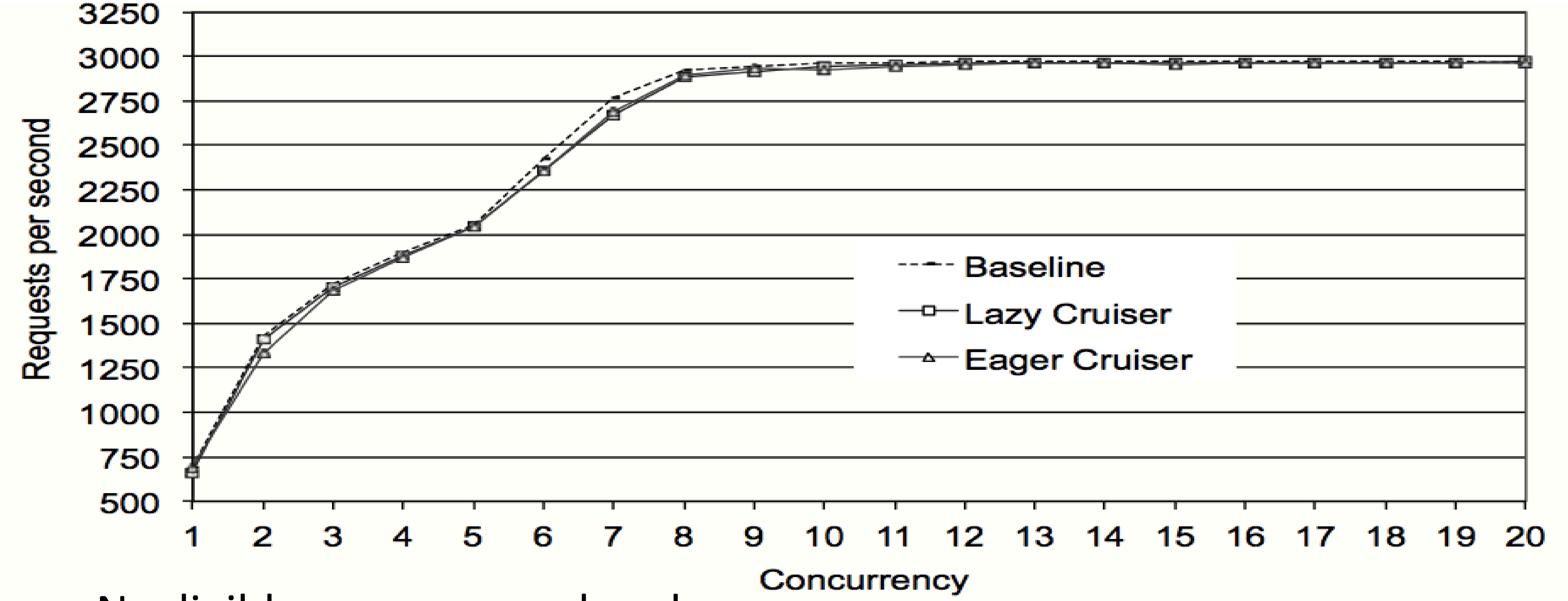
Please refer to our PLDI’11 paper.

Performance – SPEC CPU2006



- 5% with Eager Cruiser, 12.5% with Lazy Cruiser
- 5, 000 whole-heap checks per second

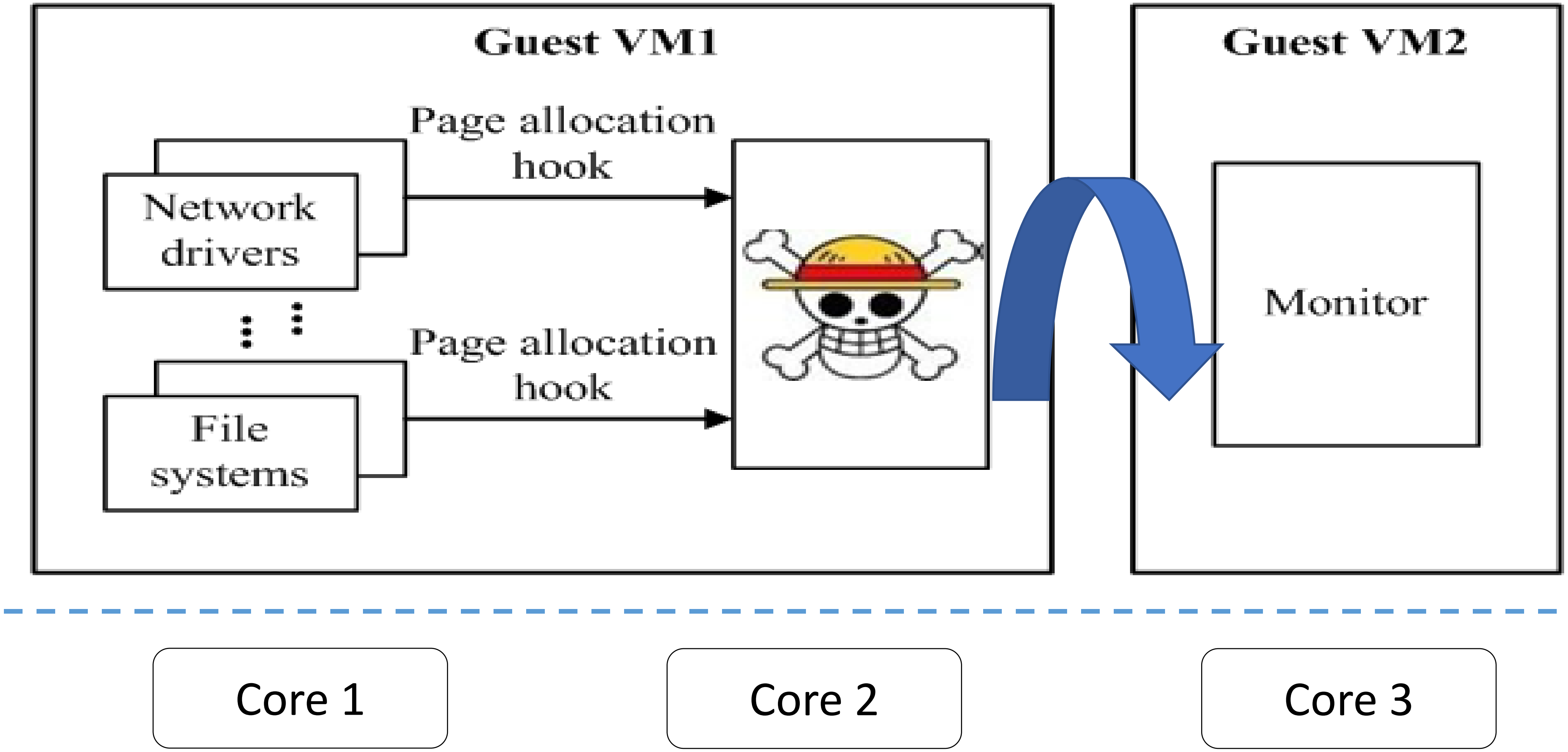
Scalability – Apache 2.2.8



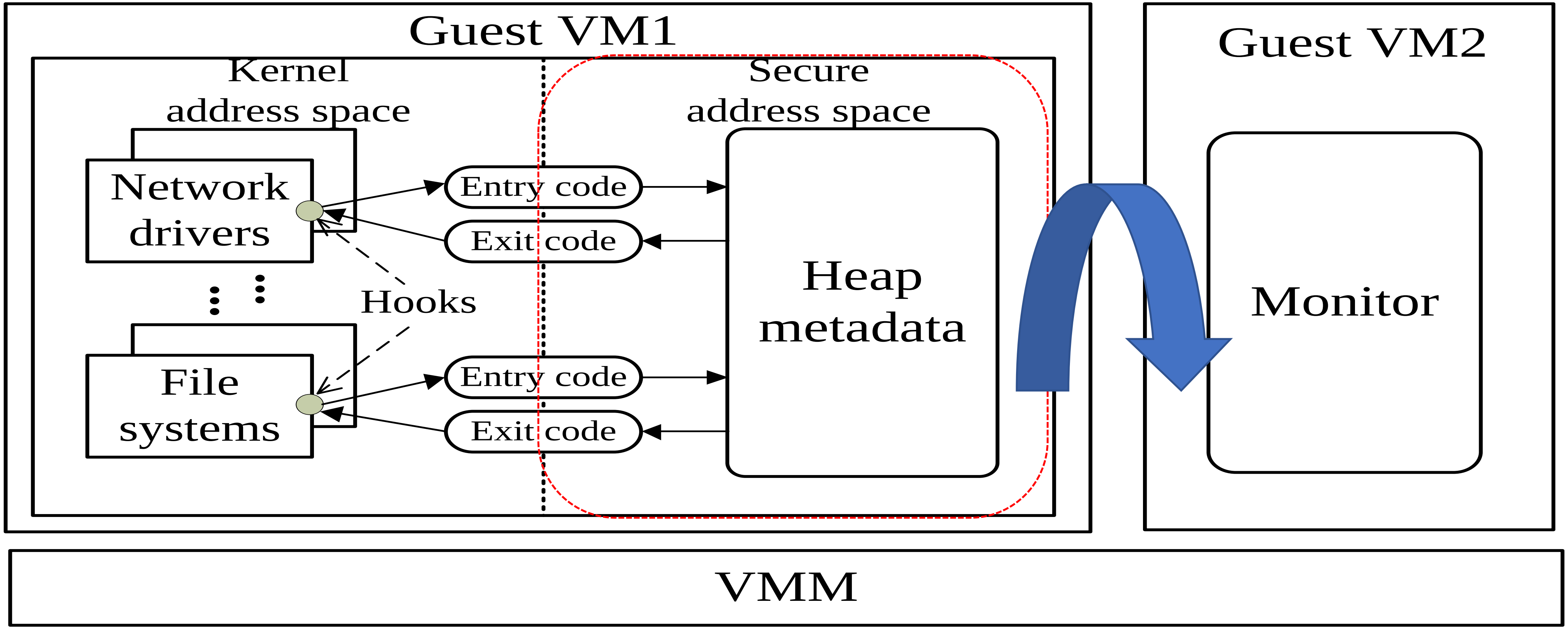
- Negligible average overhead
- Cruising cycle < 80 us (12, 500 times/second)

Problem 2: Kernel Heap Buffer Overflow Monitoring

Out-of-the-VM Architecture



Hybrid VM monitoring Architecture



- The cruising cycle = 7ms

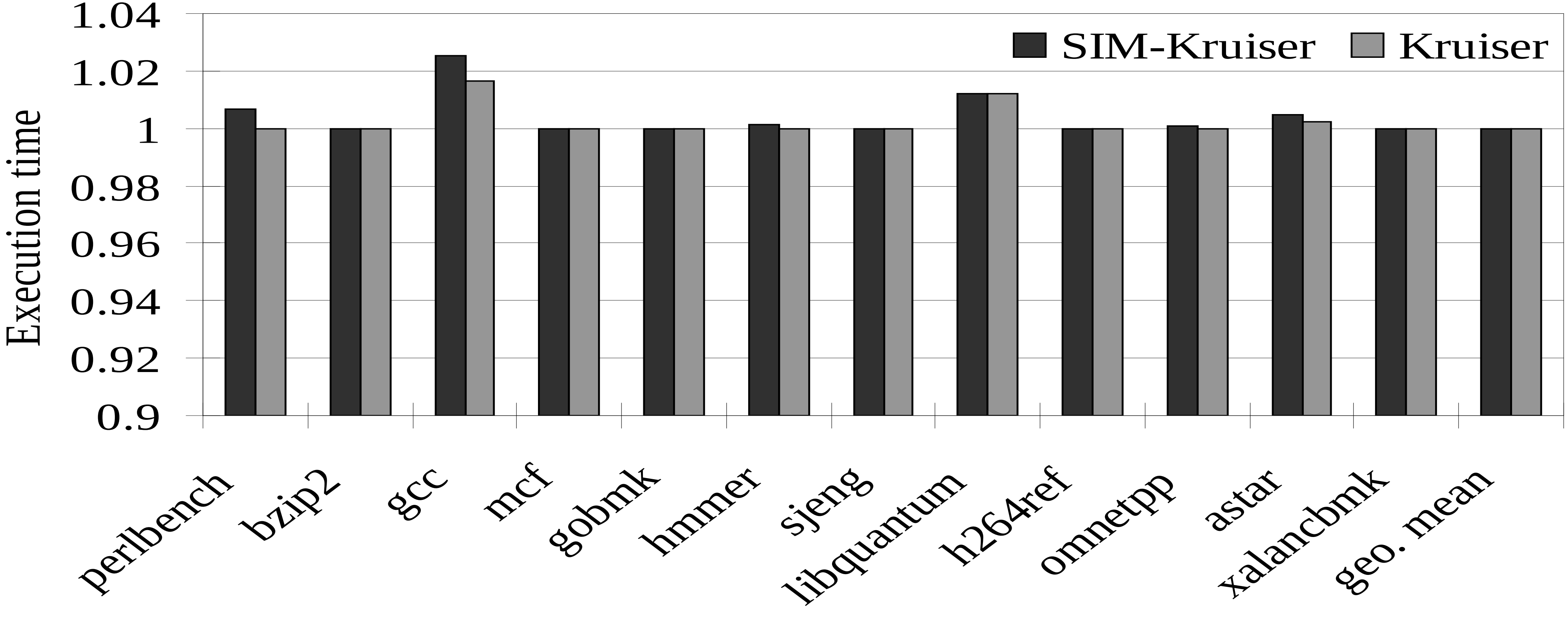
Technical hurdles

1. Race conditions

2. Self-protection

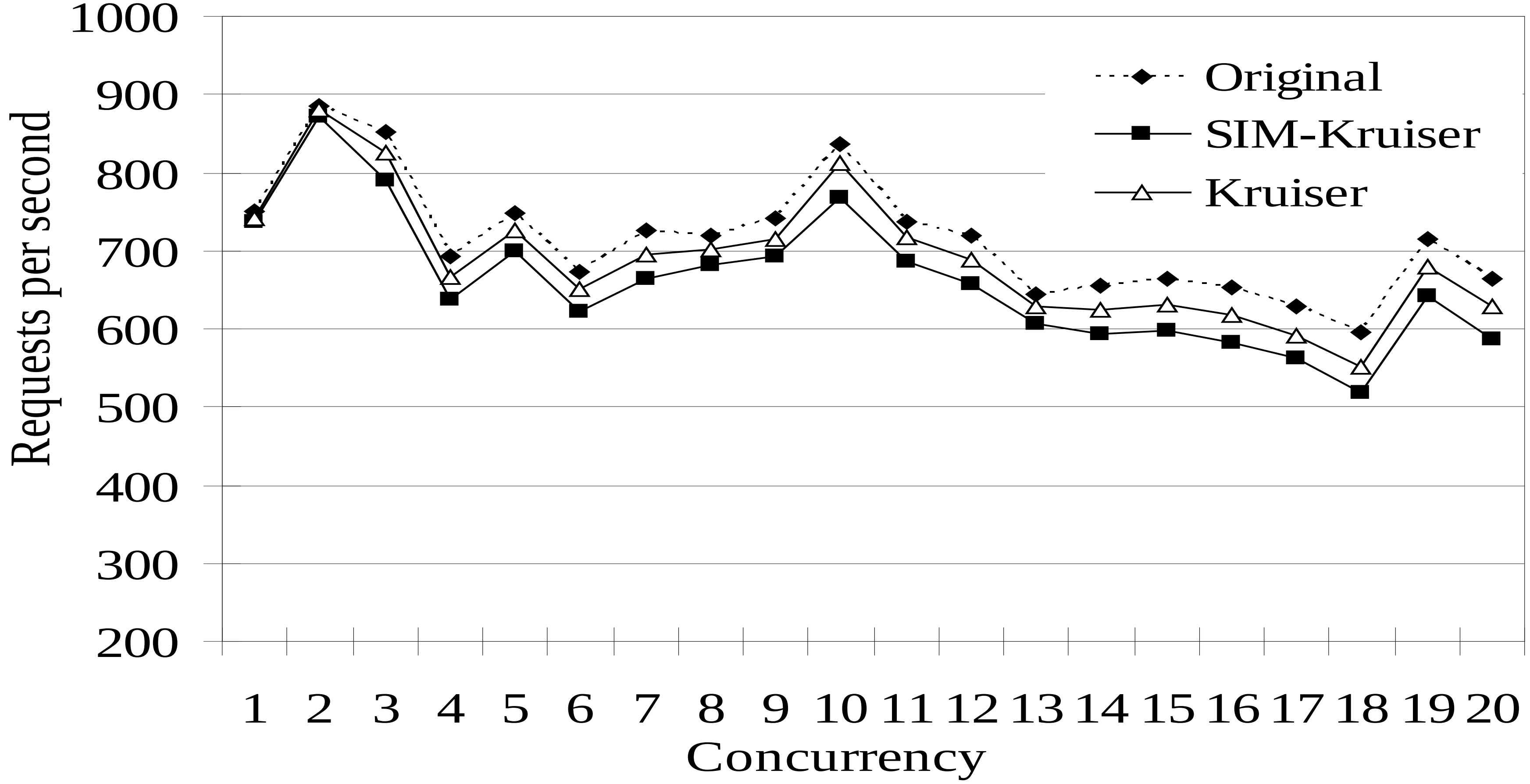
Please refer to our NDSS'12 paper.

Performance Overhead – SPEC CPU06



- Less than 3%
- Normalized to the execution time of original Linux

Scalability - Apache



- Throughput for varying numbers of concurrent requests.

Final remark

Exciting innovations on concurrent monitoring are yet to come.

- Data flows
- Control flows
- Information flows
- Data invariants
- Cross the isolation boundaries

Thank you!

Acknowledgment:

The works mentioned in this talk are supported by NSF, AFOSR MURI, and ARO MURI.