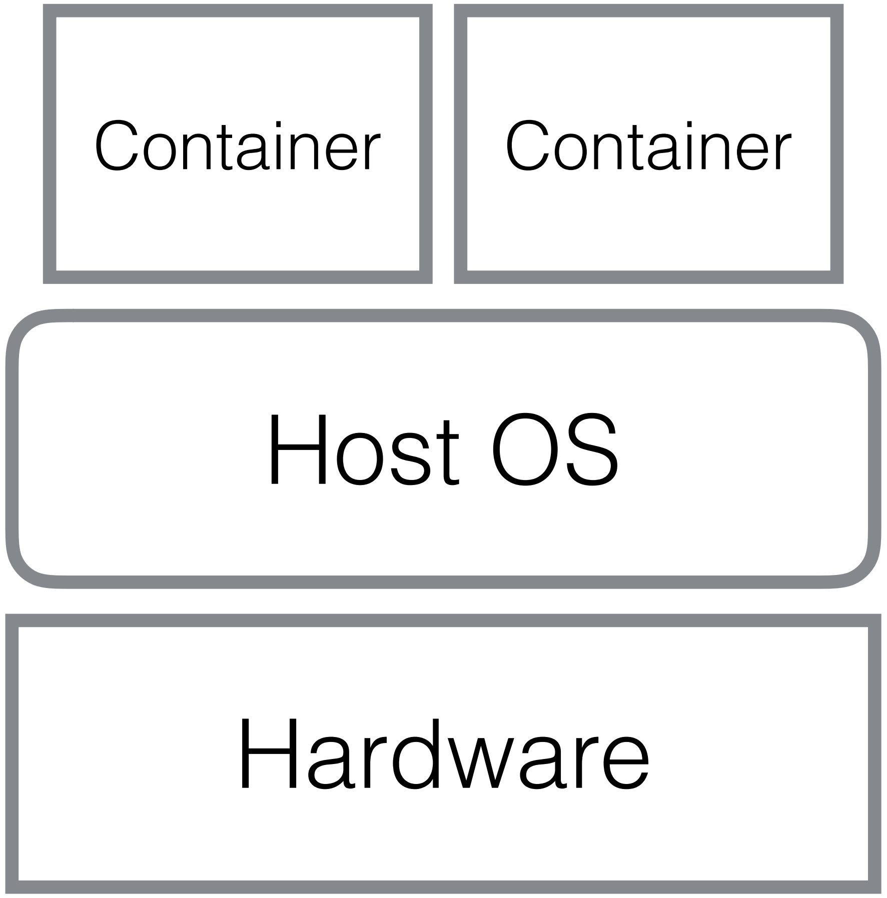
Cauldron: A Framework to Defend Against Cache-based Side-channel Attacks in Clouds

Mohammad Ahmad, Read Sprabery, Konstantin Evchenko, Abhilash Raj,

Dr. Rakesh Bobba, Dr. Sibin Mohan, Dr. Roy Campbell

Introduction to Containers

- Lightweight VM
  - Own process, network space
  - Can install own packages
- How are they different from a VM?
  - Containers share the host kernel
- Multiple implementations available
  - Docker, rkt, LXC



Building blocks of containers

- Linux Control Groups (cgroups)
    - Resource limiting & accounting
    - CPU, memory, block I/O, network
- Namespaces
    - Limit what a container can see
    - Process, network, mount, uts, ipc, user

Container Usage

- Platform as a Service Clouds (PaaS)
    - Openshift, DotCloud, Heroku
- Customers upload source code and executables
- Multi-tenant environment
- Containers often used for isolation

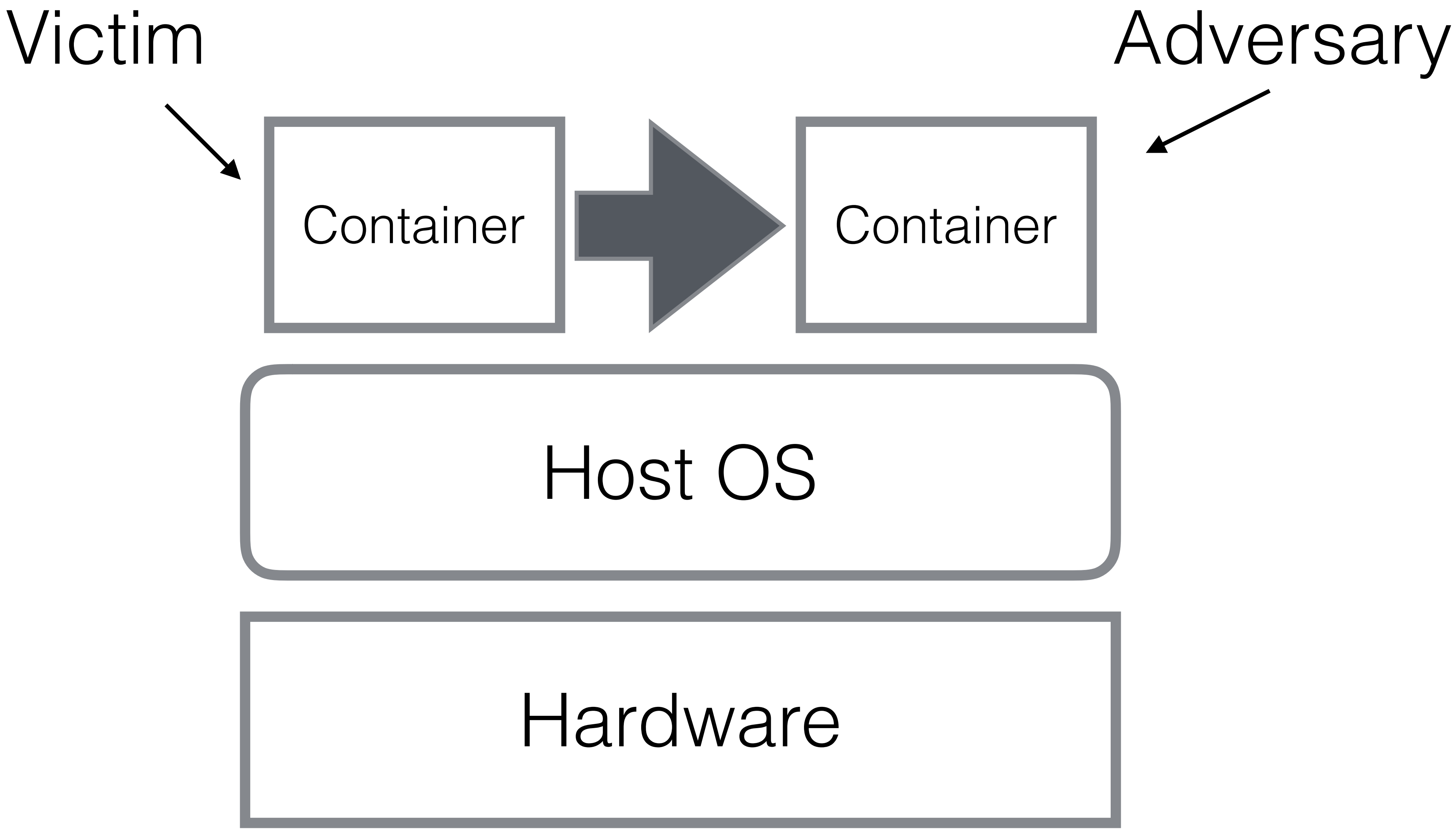
Problem Statement

- Cross container side-channel attacks on public clouds
- Cauldron aims to defend against such attacks

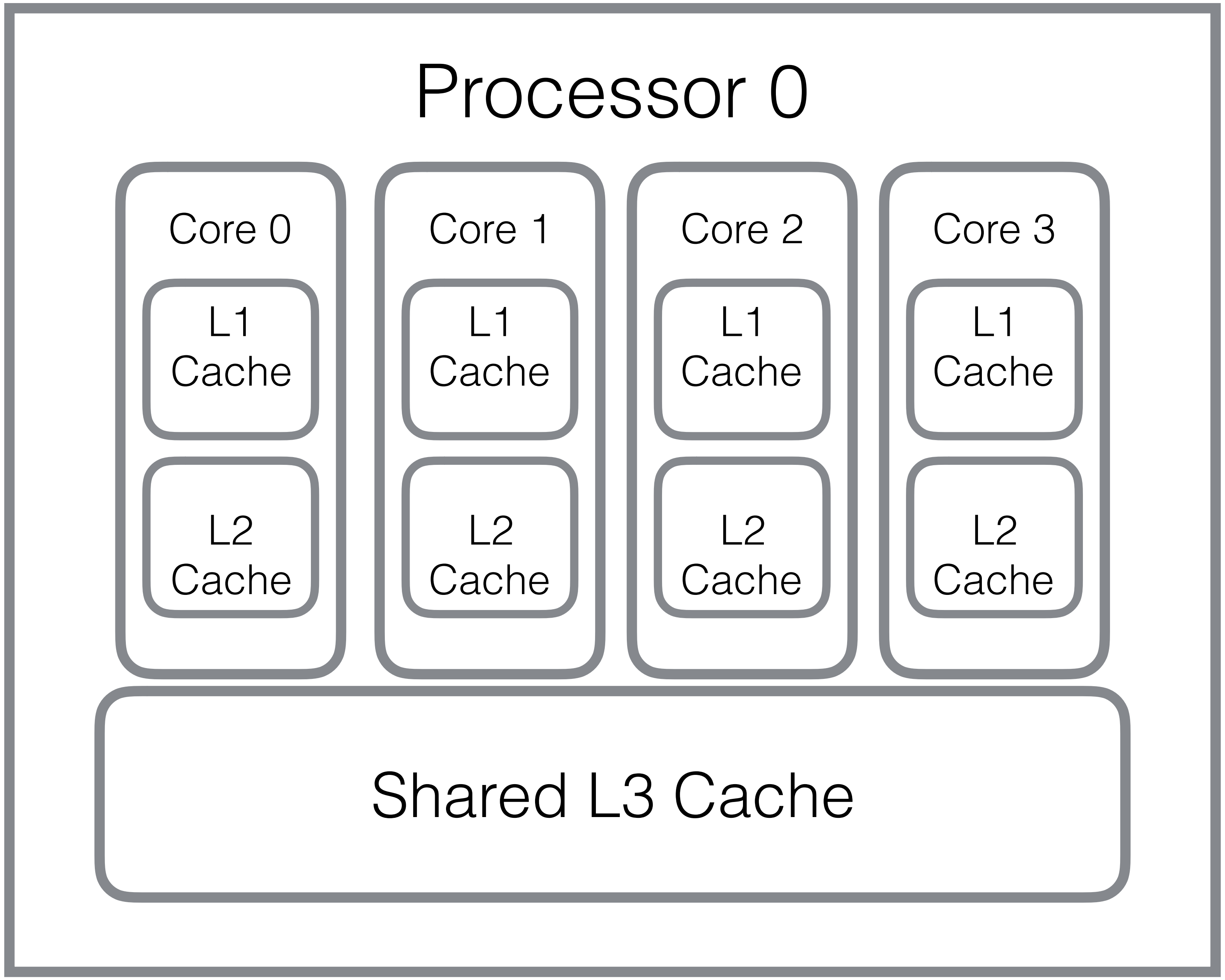
Motivation

- Defense against such attacks could prove to be a win-win for both
    - Cloud providers: Increase cloud adoption
    - Users: Reduced costs
- Private clouds with multiple security levels

Threat model



Cache Hierarchy



Processor 0

Core 0

L1 Cache

L2 Cache

Core 1

L1 Cache

L2 Cache

Core 2

L1 Cache

L2 Cache

Core 3

L1 Cache

L2 Cache

Shared L3 Cache

Flush+Reload attack

- Leverages shared libraries/binaries with the victim
- Step 1: Flush
  - Specific chunks containing instructions in the memory page shared with the victim are flushed
- Step 2: Wait…
- Step 3: Reload
  - Adversary times the reload of the same chunks

Prime+Probe Attack

- Follows similar steps as Flush+Reload
- Does not rely on shared libraries
- Added burden on attacker to identify `interesting` sets
- Can be launched from across cores or the same core

Goals for Cauldron

1. Protect against same-core and cross-core side-channel attacks

2. Not require any changes to user applications

3. Easy to deploy and adopt

4. Incur reasonable performance overheads

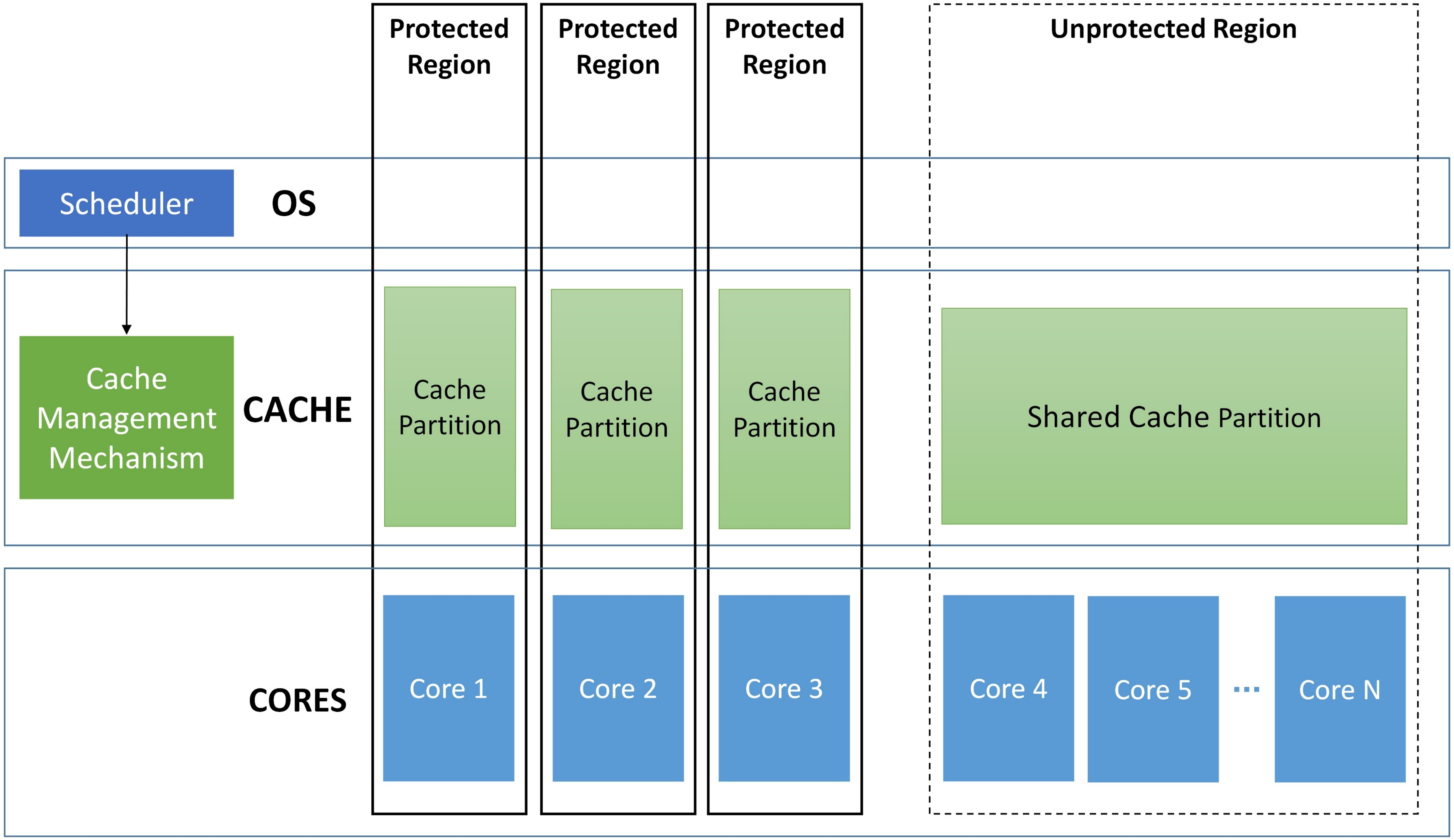
Intel Cache Allocation Technology (CAT)

- Partition the last level cache (LLC) between cores
- Protects against cross-core Prime+Probe attacks
- Limitations
    - Four partitions supported
    - Vulnerable to same-core side-channel attacks & Flush+Reload

Cache Flushing without Partitioning

- Flush the cache on each context switch
- High cache flushing overhead
- Limitation
    - Vulnerable to LLC based cross-core side-channel attacks

Cauldron Architecture



Cauldron

- Each protected region consists of
    - One core & partitioned LLC
- Cache flush between context switches between different clients in each protected region
- Only flush LLC partition allocated to the protected region

Cauldron: Gang Scheduling

- Hyperthreading disabled
- Gang schedule tasks belonging to the same client on the logical cores that map to the same physical core
- Increase the number of cores available in the protected regions

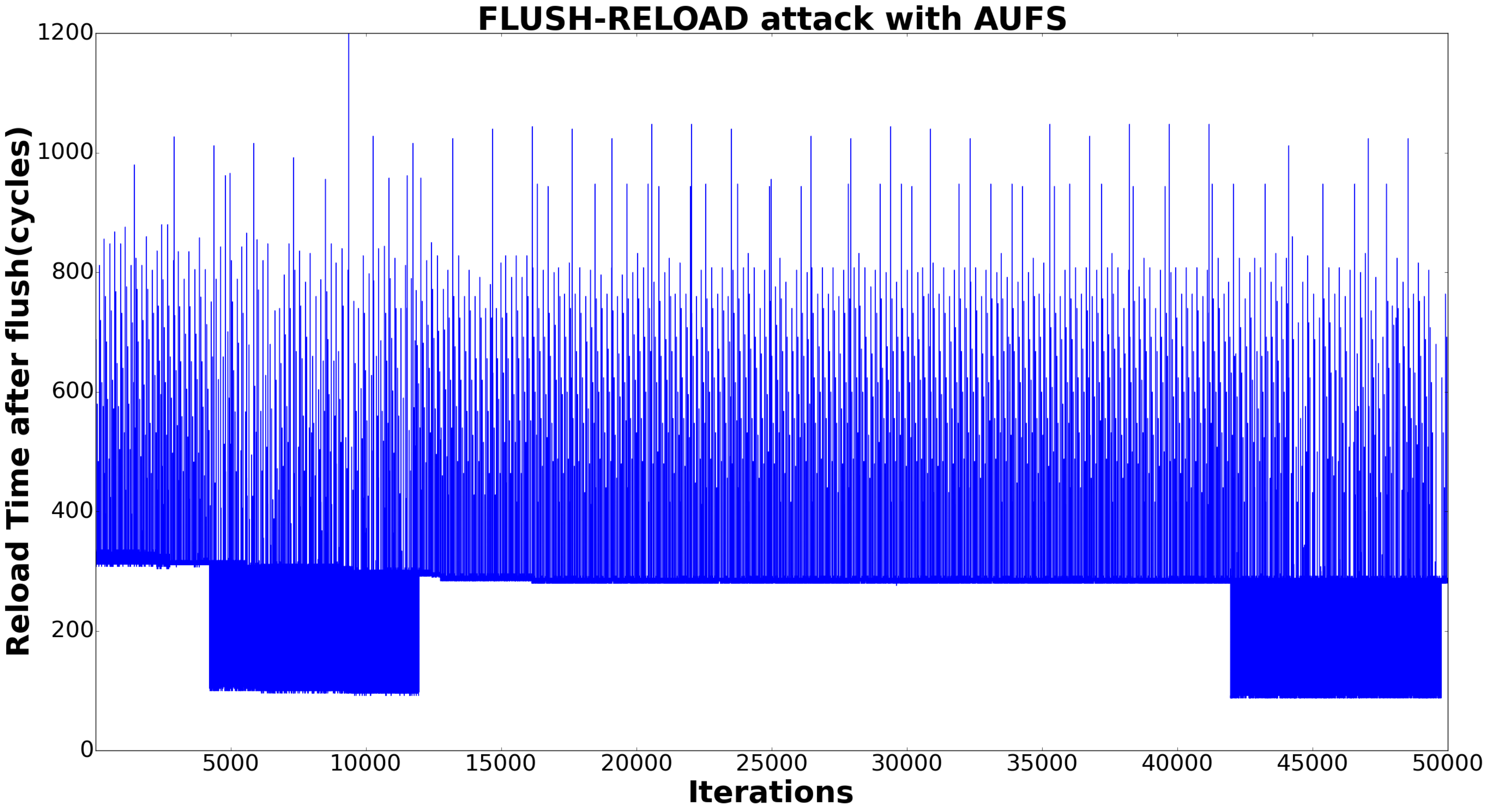
Cauldron: Implementation

- Userspace utility to configure cache partitions
- Client differentiation using cgroups
- Scheduler
    - Loadable kernel module
    - Return probes (kretprobes)
    - Plug into the Linux scheduler routine

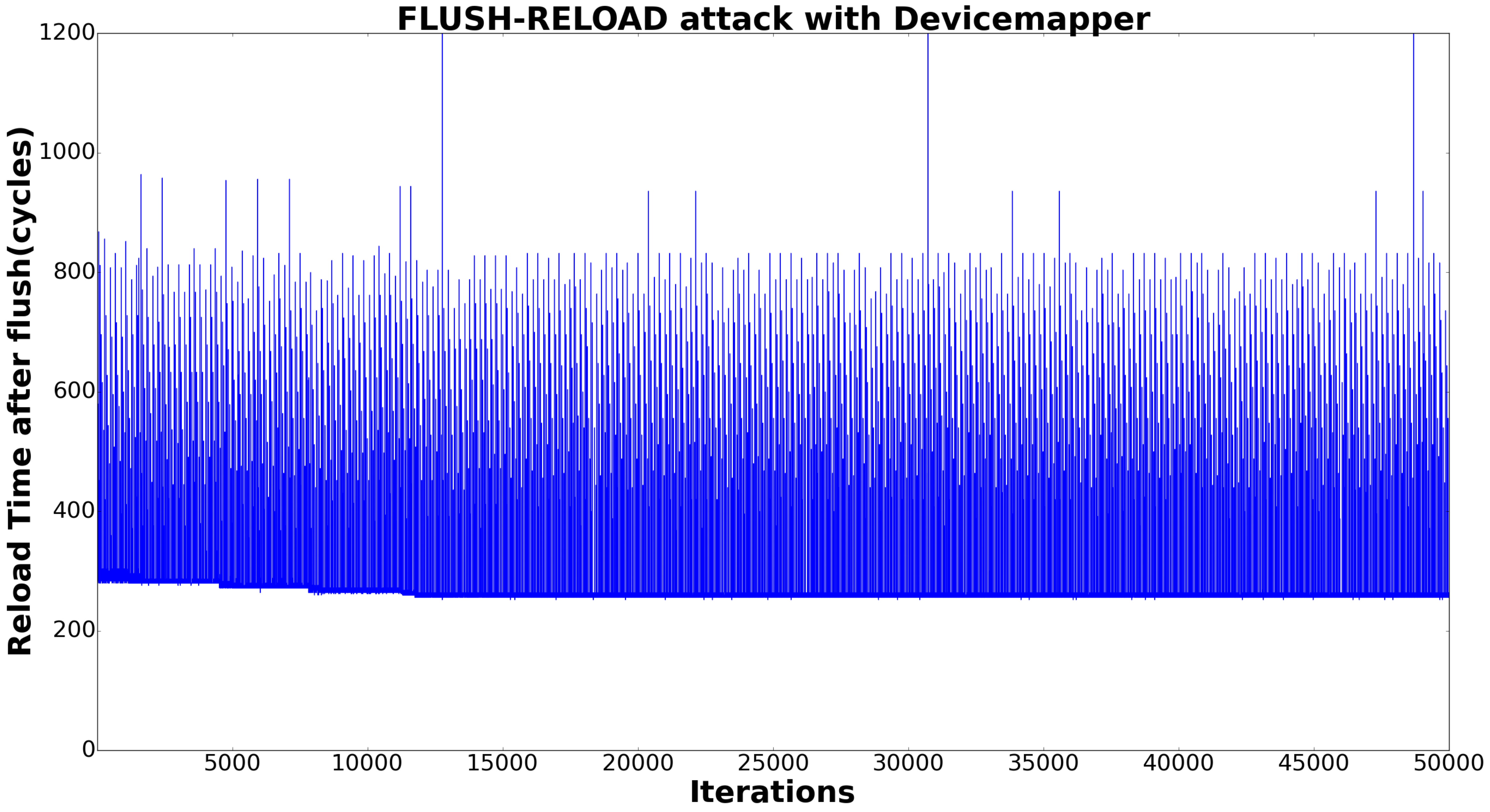
Security Evaluation

- Intel Xeon E5-2618 v3
- 8 physical cores
- Victim application: GnuPG 1.4.13

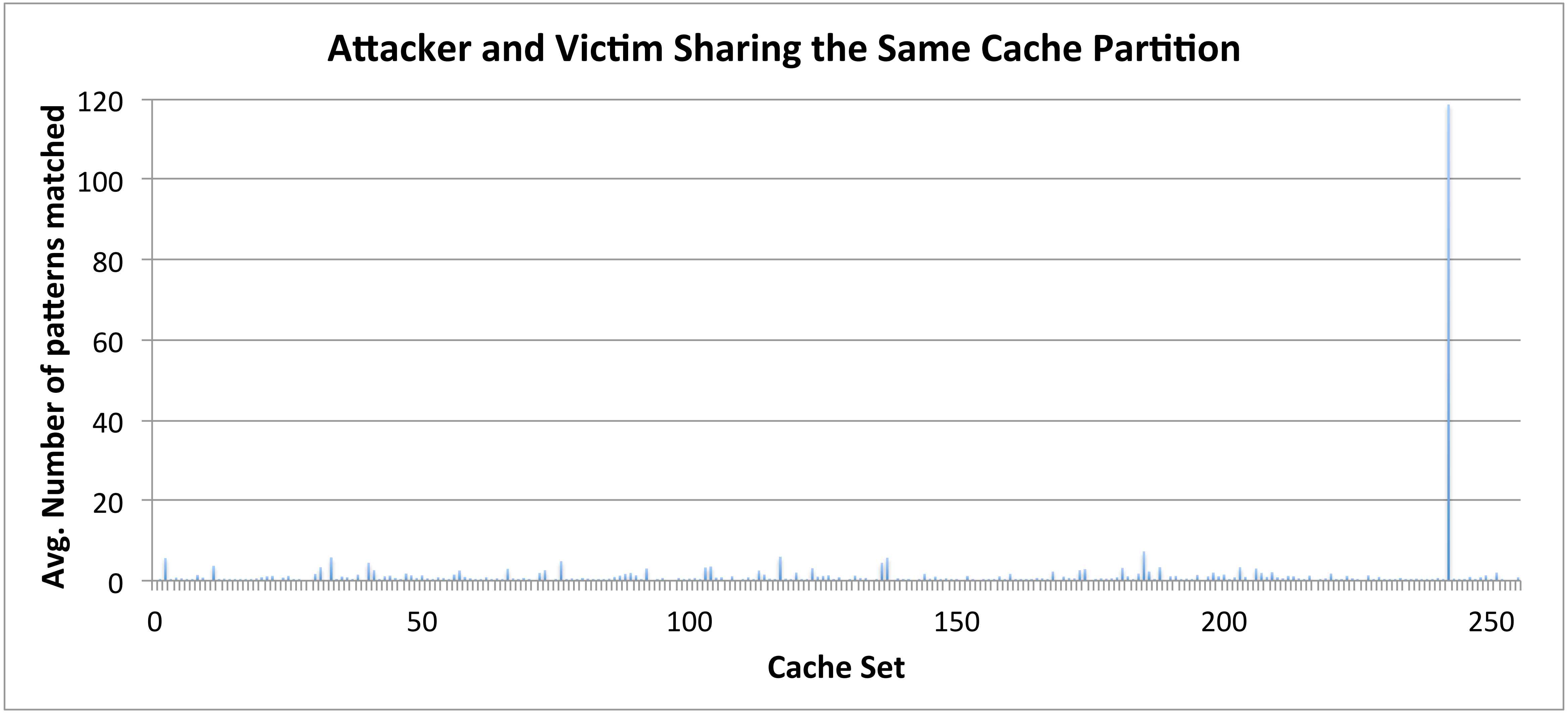
Flush+Reload



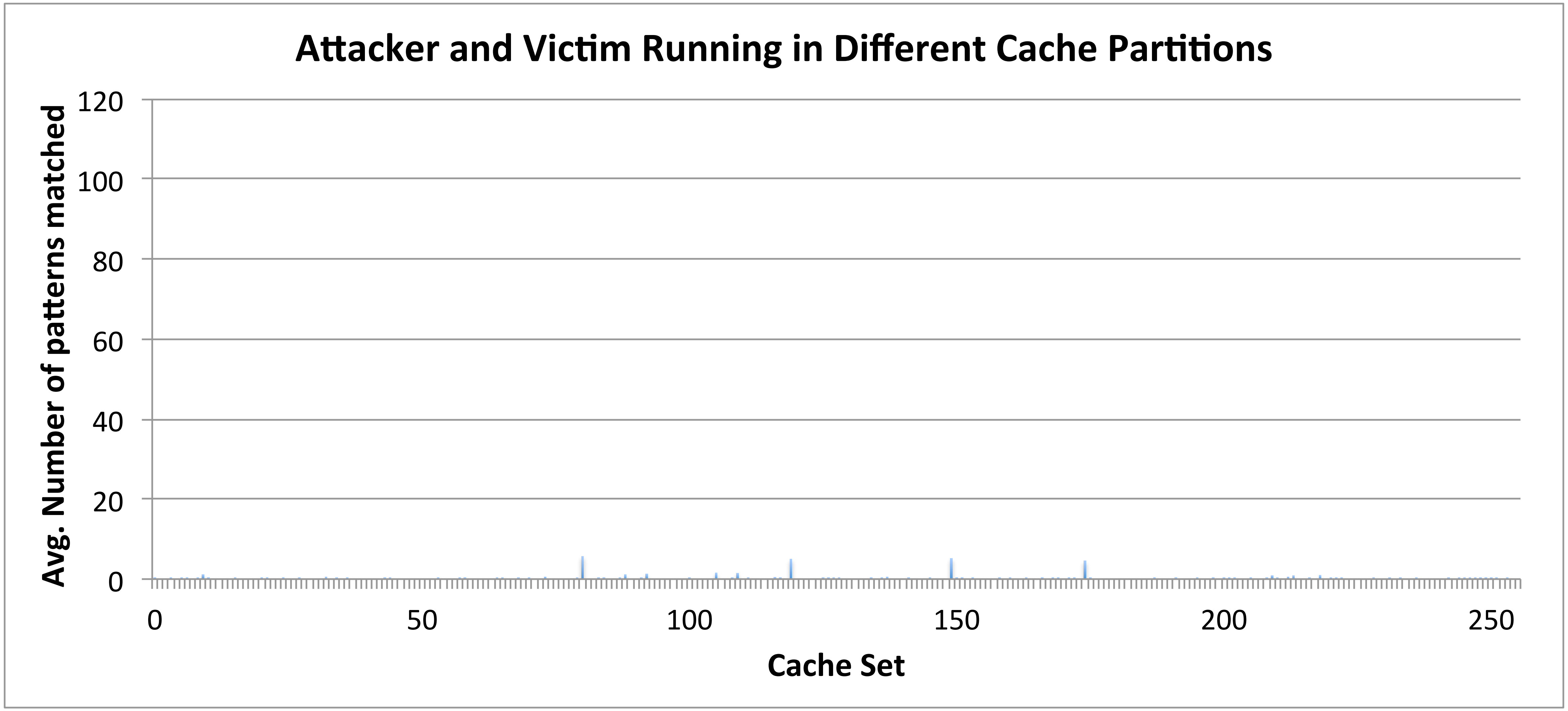
Flush+Reload cont'd



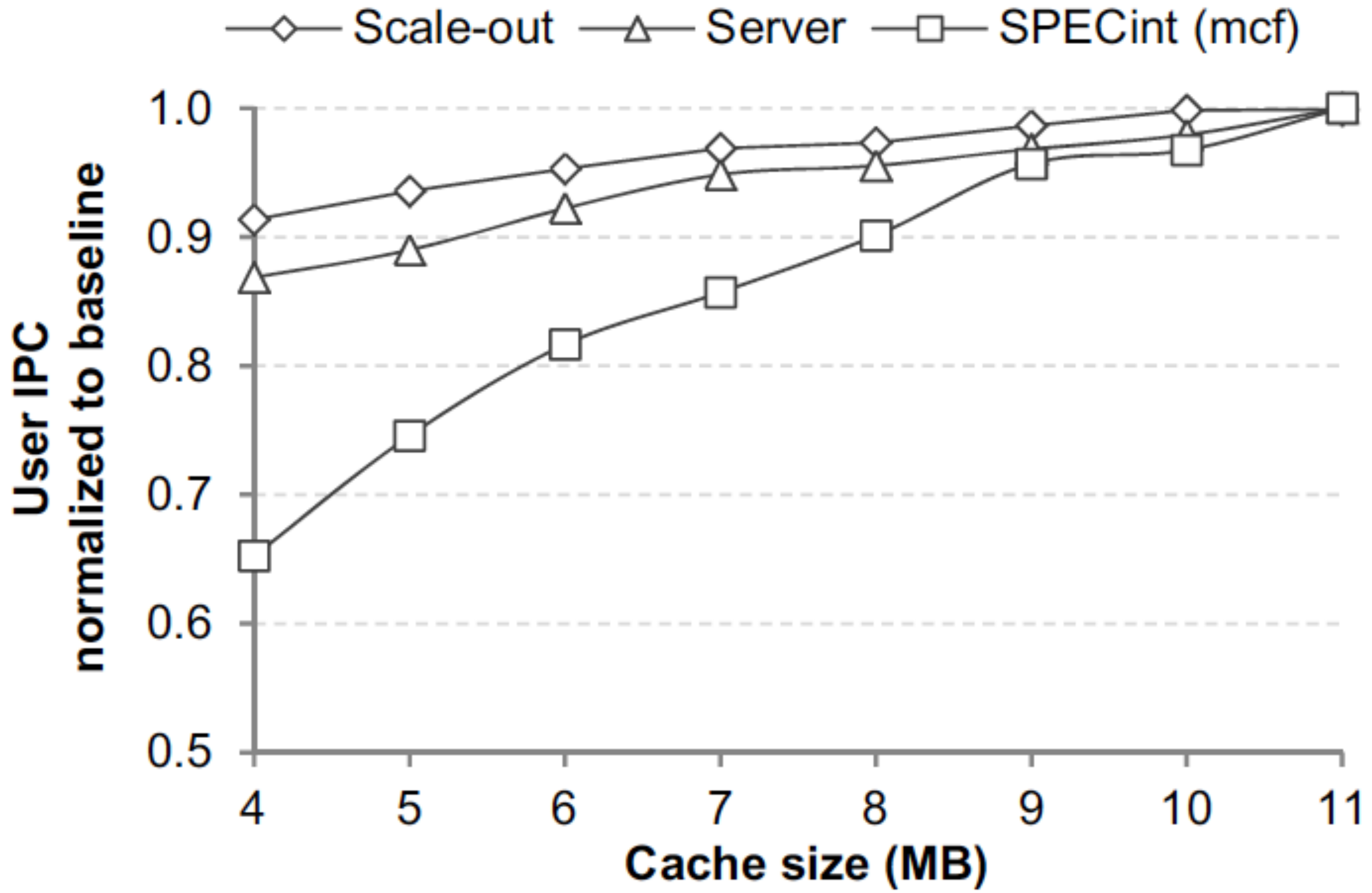
Prime+Probe


Attacker and Victim Sharing the Same Cache Partition
Avg. Number of patterns matched
120
100
80
60
40
20
0
0
50
100
150
200
250
Cache Set

Prime+Probe cont’d



Performance Evaluation



Ferdman, Michael, et al. "Clearing the clouds: a study of emerging scale-out workloads on modern hardware."

Research Challenges

- Scheduler optimizations
- Detection of malicious containers
- Selective sharing of libraries
- Container placement

Conclusion

Goals for Cauldron

1. Protect against same-core and cross-core side-channel attacks

2. Not require any changes to user applications

3. Easy to deploy and adopt

4. Incur reasonable performance overheads