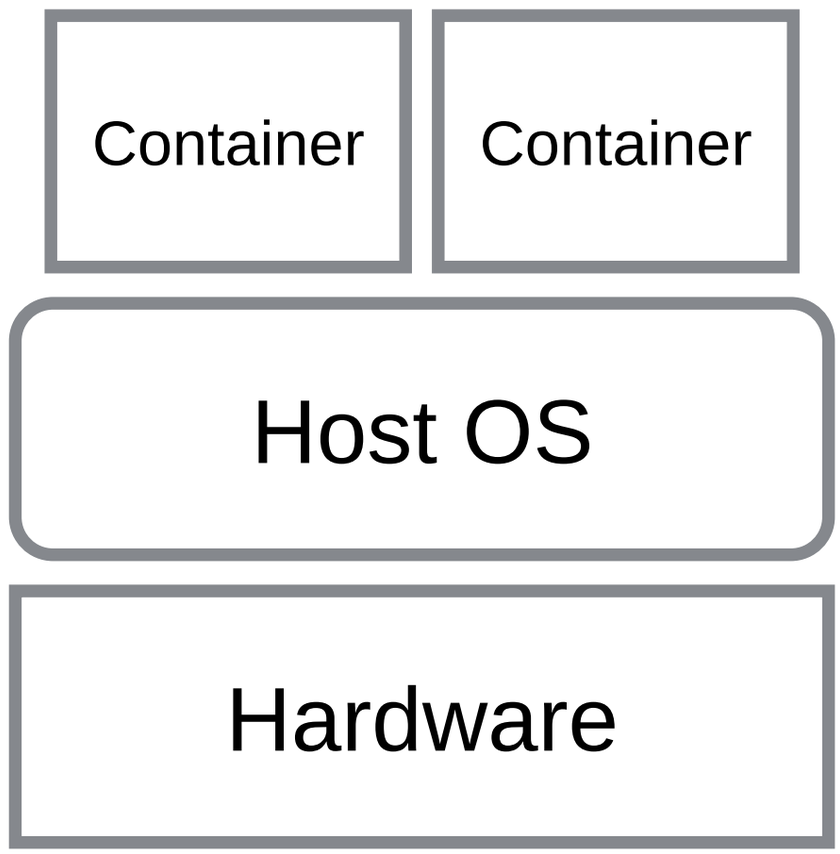
Moving towards a Secure Container Framework

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

Introduction to Containers

- Lightweight VM
  - Own process space
  - Network space
  - Can install own packages
- How are they different from a VM?
  - OS based virtualization
  - Uses the host Kernel
  - Can not boot a different Kernel



Building blocks of containers

- Cgroups
  - Resource accounting & limiting
    - CPU, memory, Block I/O
- Namespaces
  - Limit what a container can see
  - Process, filesystem, network

What’s all the hype about?

- Drivers of adoption
- Startup on the order of milliseconds
- Packaging dependencies
  - Portability

Container Usage

- Containers are gaining popularity in Platform as a Service Clouds (PaaS)
  - Openshift
  - DotCloud
  - Heroku
- Multiple implementations available
  - docker, rkt, LXC

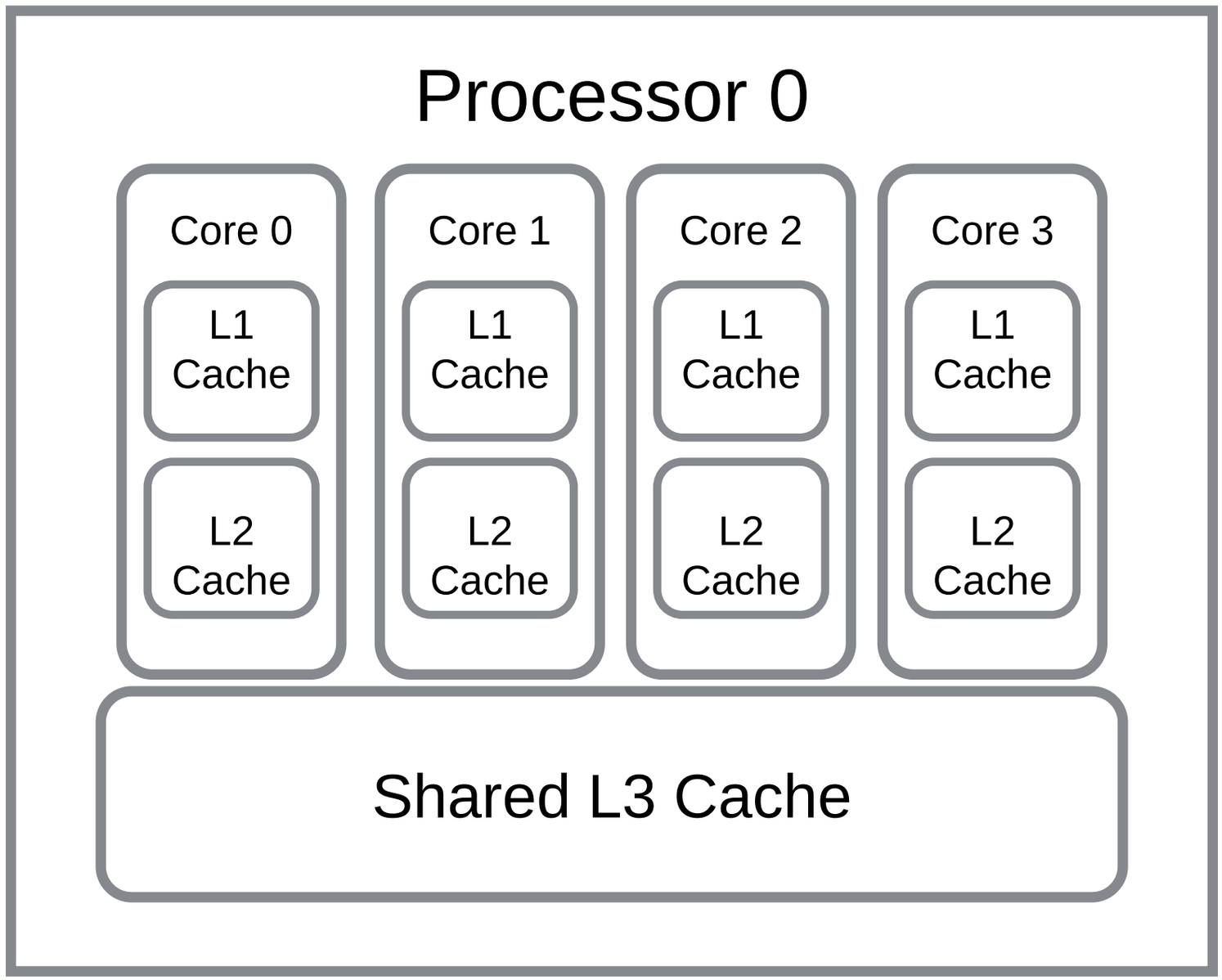
But wait

- We have a problem …
  - Cross container side-channel attacks shown on public clouds!

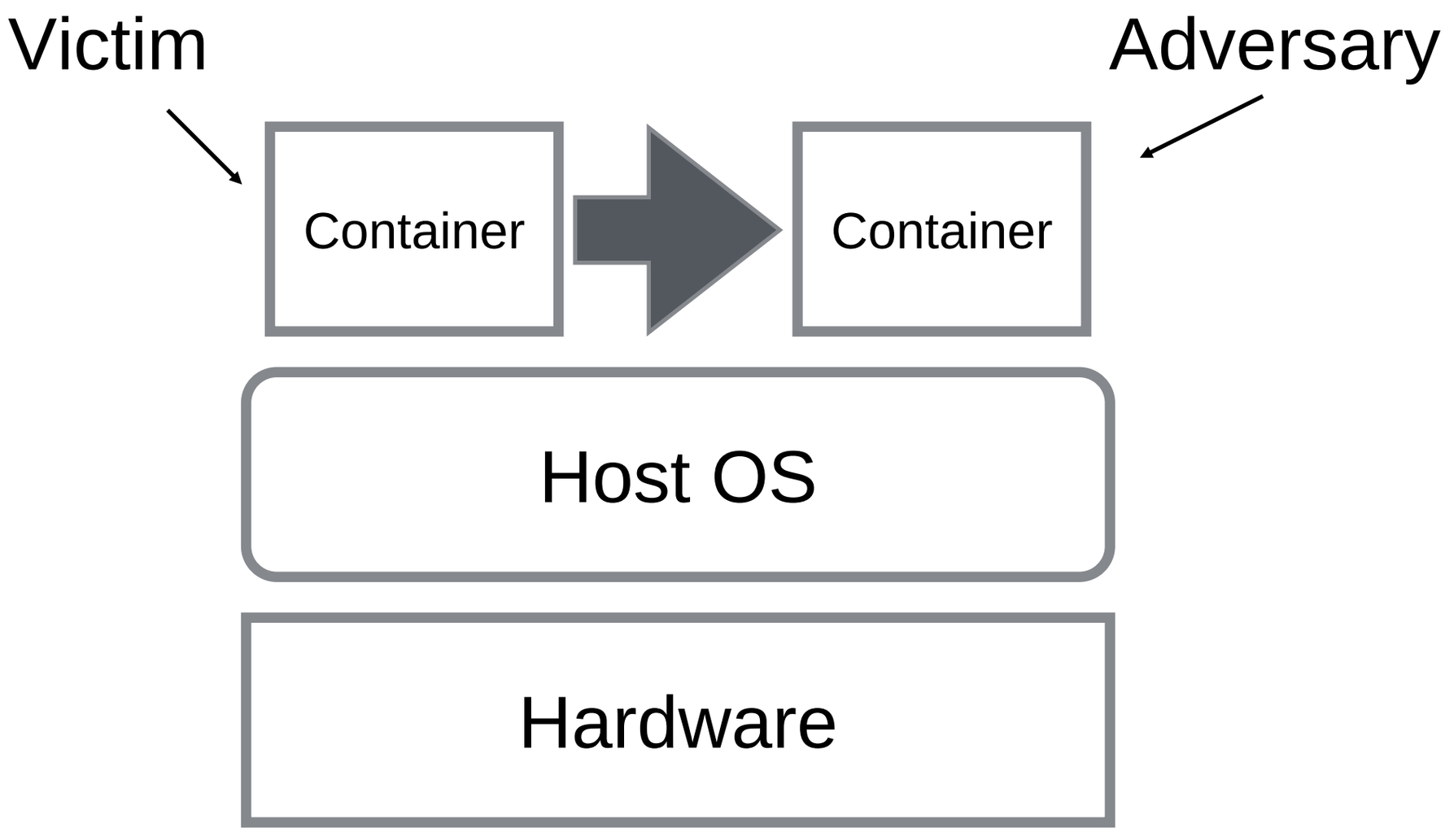
Platform as a Service (Paas)

- Customers upload source code and executables
- Cloud provider facilitates data storage, monitoring and logging
- Multi-tenant environment
  - Containers used for isolation

Cache Hierarchy



Threat model



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
  - Using the `clflush` instruction
- Step 2: Flush-Reload interval
- Step 3: Reload
  - Adversary times the reload of the same chunks

Why are we interested in cache based side-channels?

- Fine-grained cross-tenant attacks shown in public clouds

Motivation

- Such attacks inhibit users from moving to public clouds
- Defense against such attacks could prove to be a win-win for both
  - Cloud providers: More customers
  - Cloud users: Reduced costs
- Private clouds with multiple security levels

How can we defend against such attacks?

- Disallow resource sharing
  - Duplicate binaries
    - Increase in memory footprint
    - Decrease number of tenants (Profit!)
  - Selective sharing
- What about coarse-grained attacks?

Dedicated Instances

- No multi-tenancy
- Expensive!

Our approach

Cache flushing

- Lets flush the cache on each context switch
- Problem
  - Overhead of cache flushing
  - Vulnerable to LLC based cross-core side-channel attacks

Security aware scheduling

- Gang schedule trusted tenants
- Flush caches between context switches

Security aware scheduling

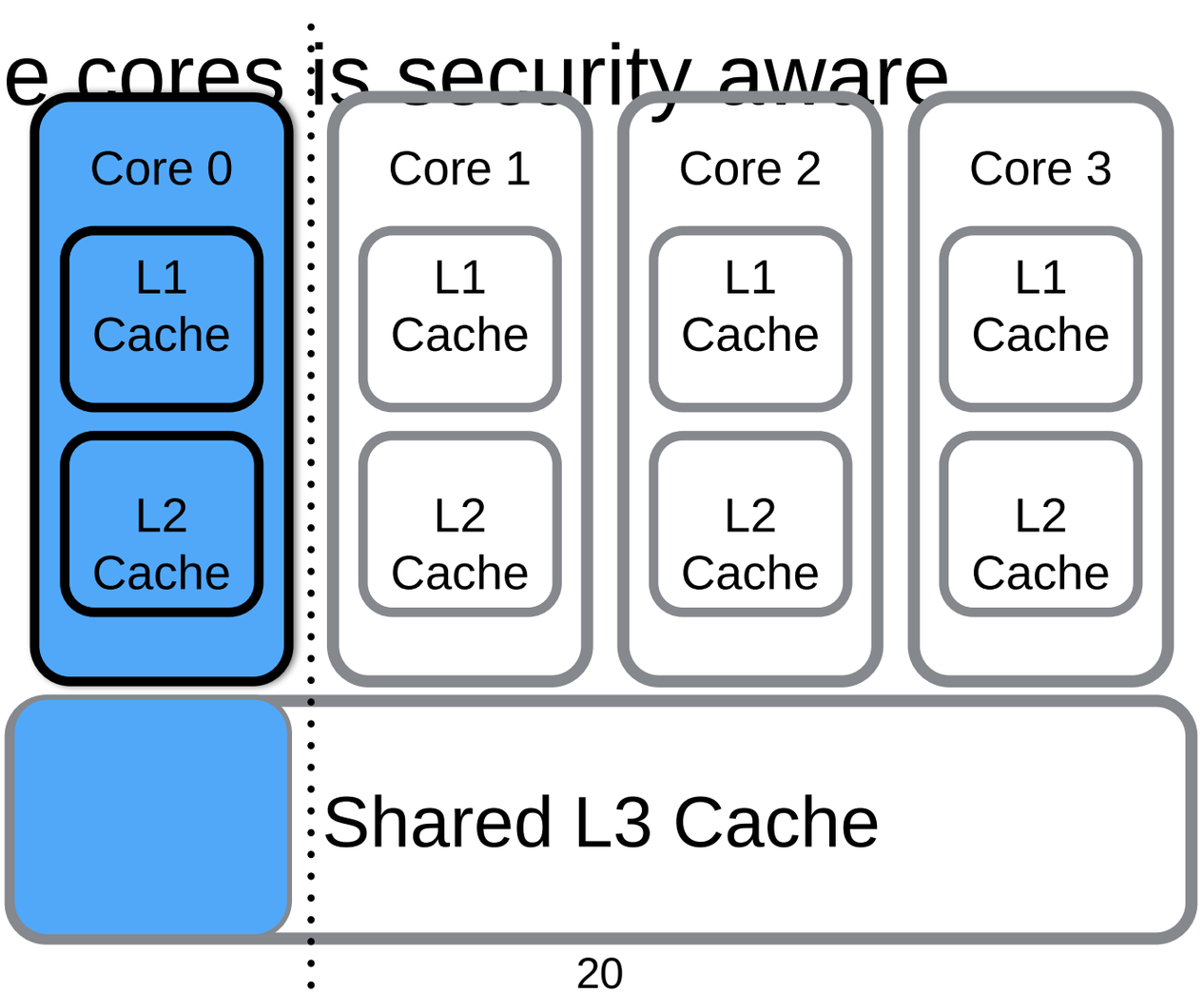
- Problem
  - Gang scheduling can lead to reduced utilization
  - Overhead of cache flushing

Cache partitioning

- Partition the last level cache (LLC) between tenants
- Hardware support
  - Intel Cache Allocation Technology (CAT)
  - Allows us to dynamically partition the LLC
  - Enables increased isolation

Scheduling + Cache Partitioning

- SecureCore
- One of the cores is security aware



Scheduling + Cache Partitioning

- SecureCore
- One of the cores is security aware
- Isolated LLC
- Cache flush between context switches on this core
- Only flush LLC partition allocated to this core

Current Implementation

- Built a loadable kernel module
  - Return probes (kretprobes)
  - Plug into the Linux scheduler routine
- Adapted Google PerfKitBenchmarker

Additional optimizations

- Scheduler optimizations to minimize the number of cache flushes

- Increased Minimum Runtime for processes

Research Challenges

- Scheduler optimizations
- Detection of malicious containers
- Container placement
- Alternate approaches for isolation

Questions

- Scheduler optimizations
- Detection of malicious containers
- Container placement
- Alternate approaches for isolation