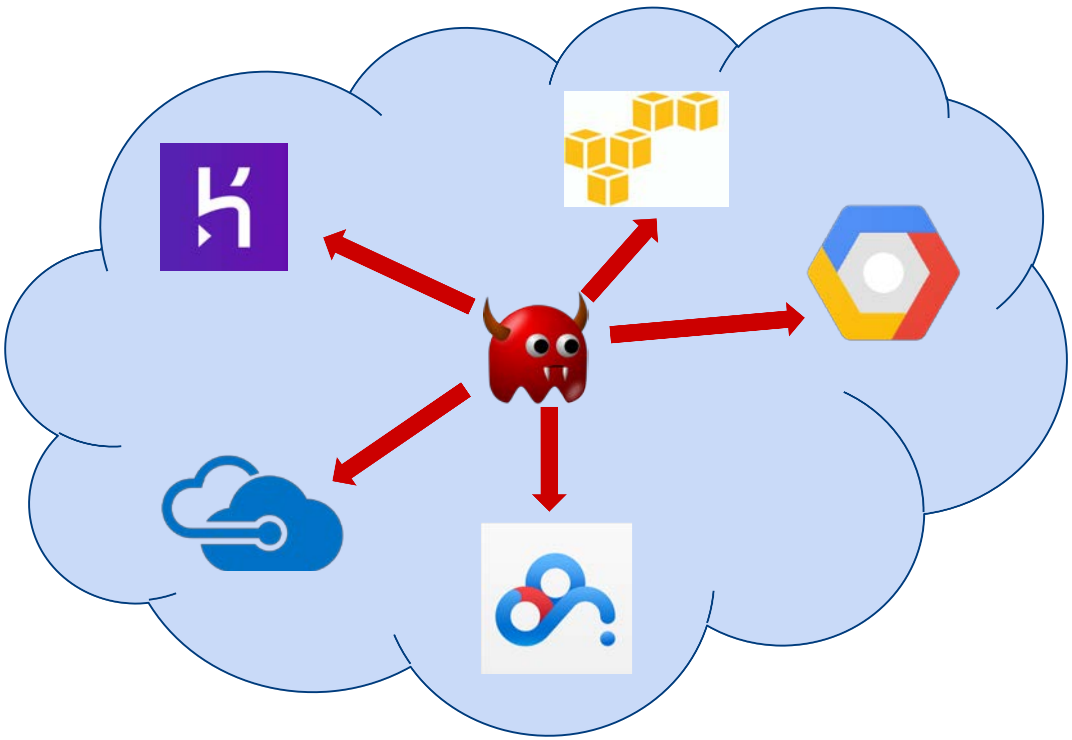
Label-based Defenses Against Side Channel Attacks in PaaS Cloud Infrastructure

Read Sprabery, Konstantin Evchenko, Abhilash Raj*, Shivana Wanjara*, Sibin Mohan, Rakesh Bobba*, Roy H. Campbell

University of Illinois at Urbana-Champaign

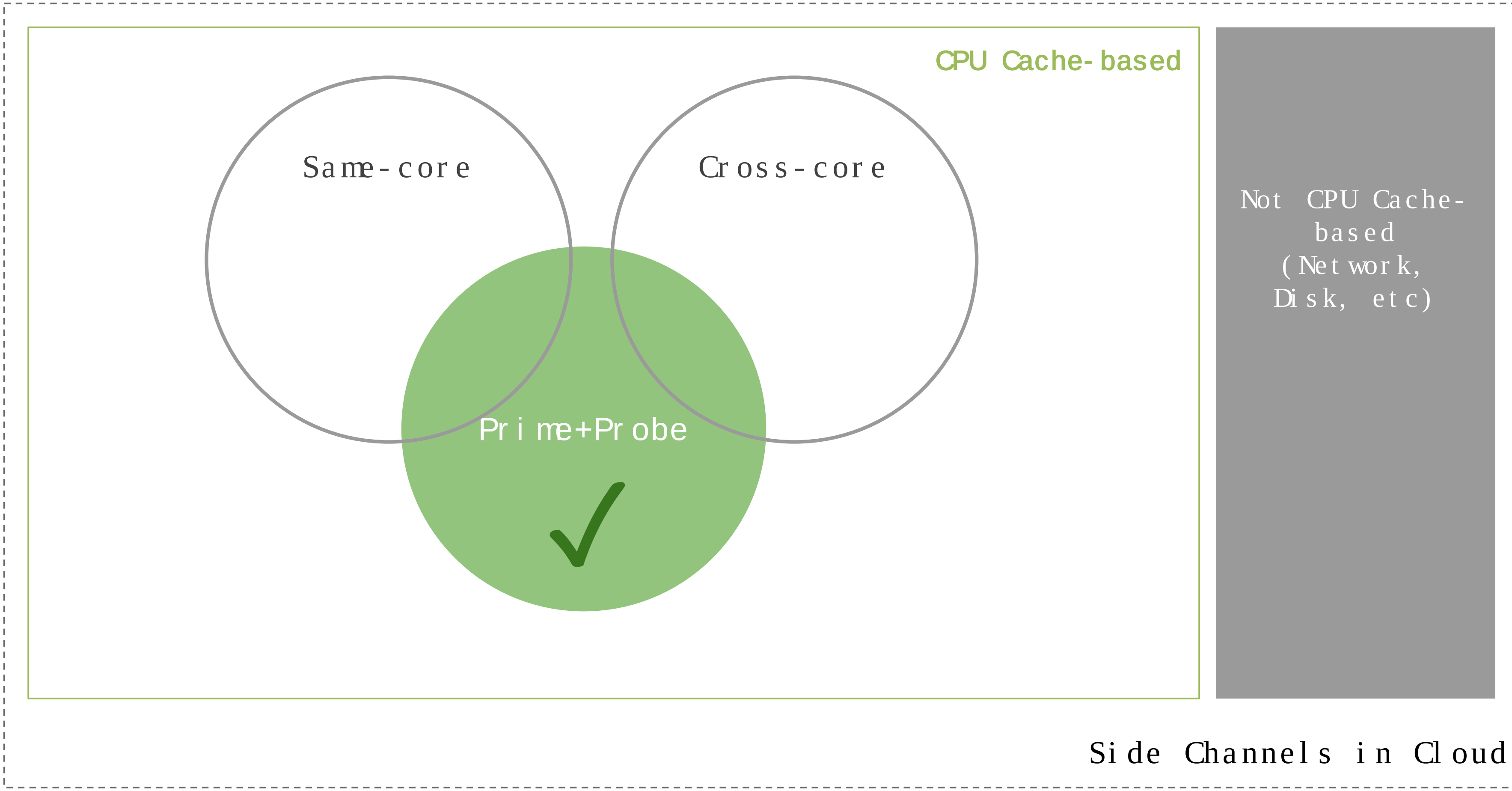
*Oregon State University

Why do we care?

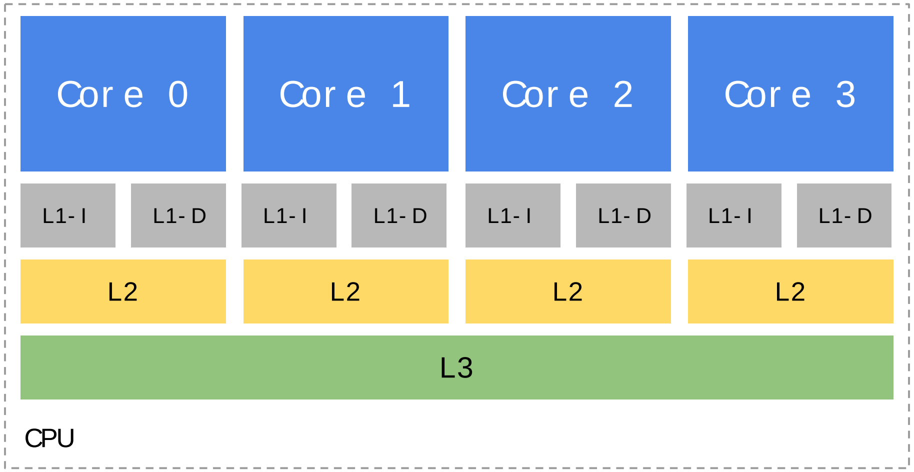


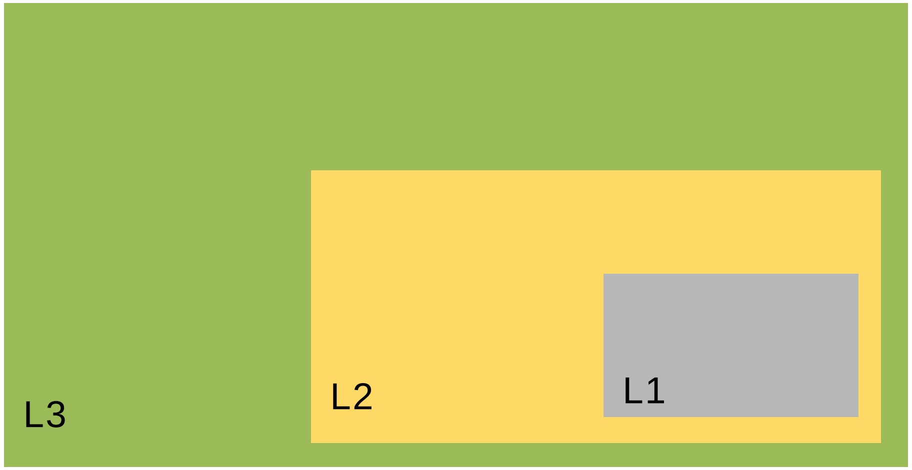
- ✓ First attempts to extract sensitive information go back in **2005**
- ✓ This work has been extended in many ways
- ✓ In **2012**, cache side channel helped to extract a secret key across VMs
- ✓ In **2014**, the attack was successfully demonstrated in a **public cloud**

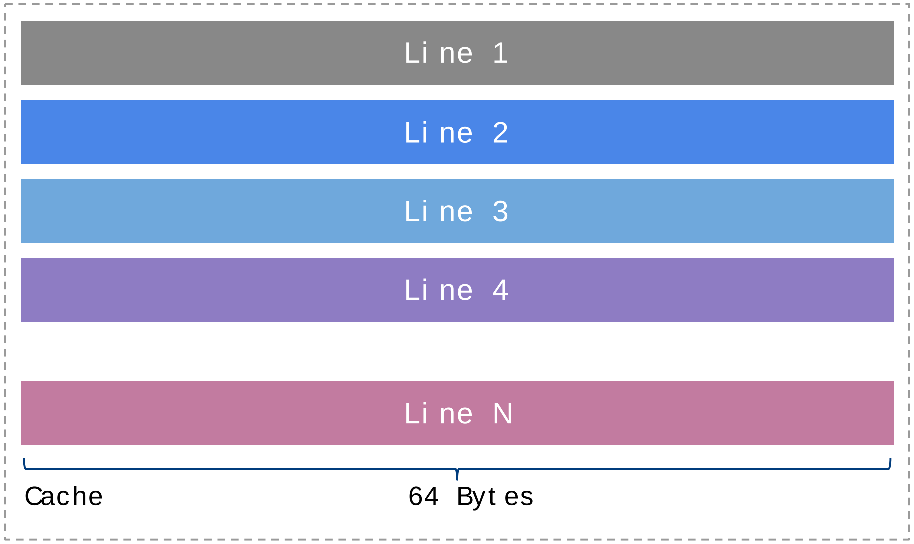
Focus

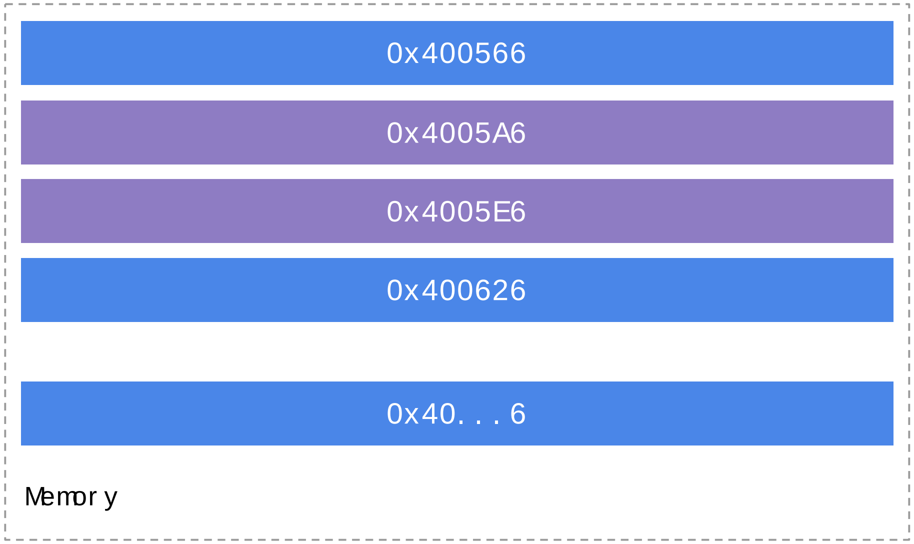

CPU Cache-based
Same-core
Cross-core
Prime+Probe
Not CPU Cache-based
(Network, Disk, etc)
Side Channels in Cloud

Background: Modern Cache Architecture

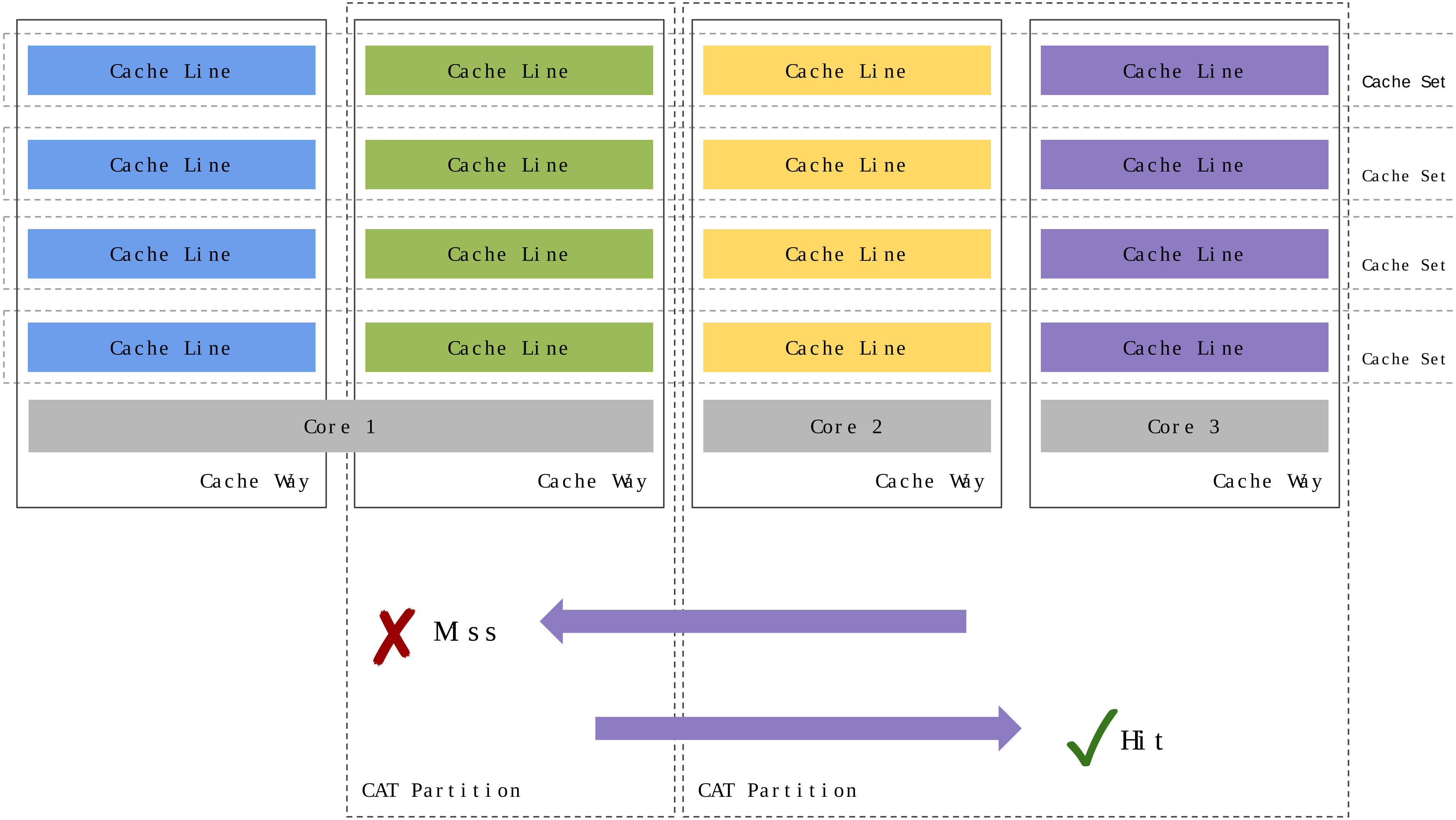




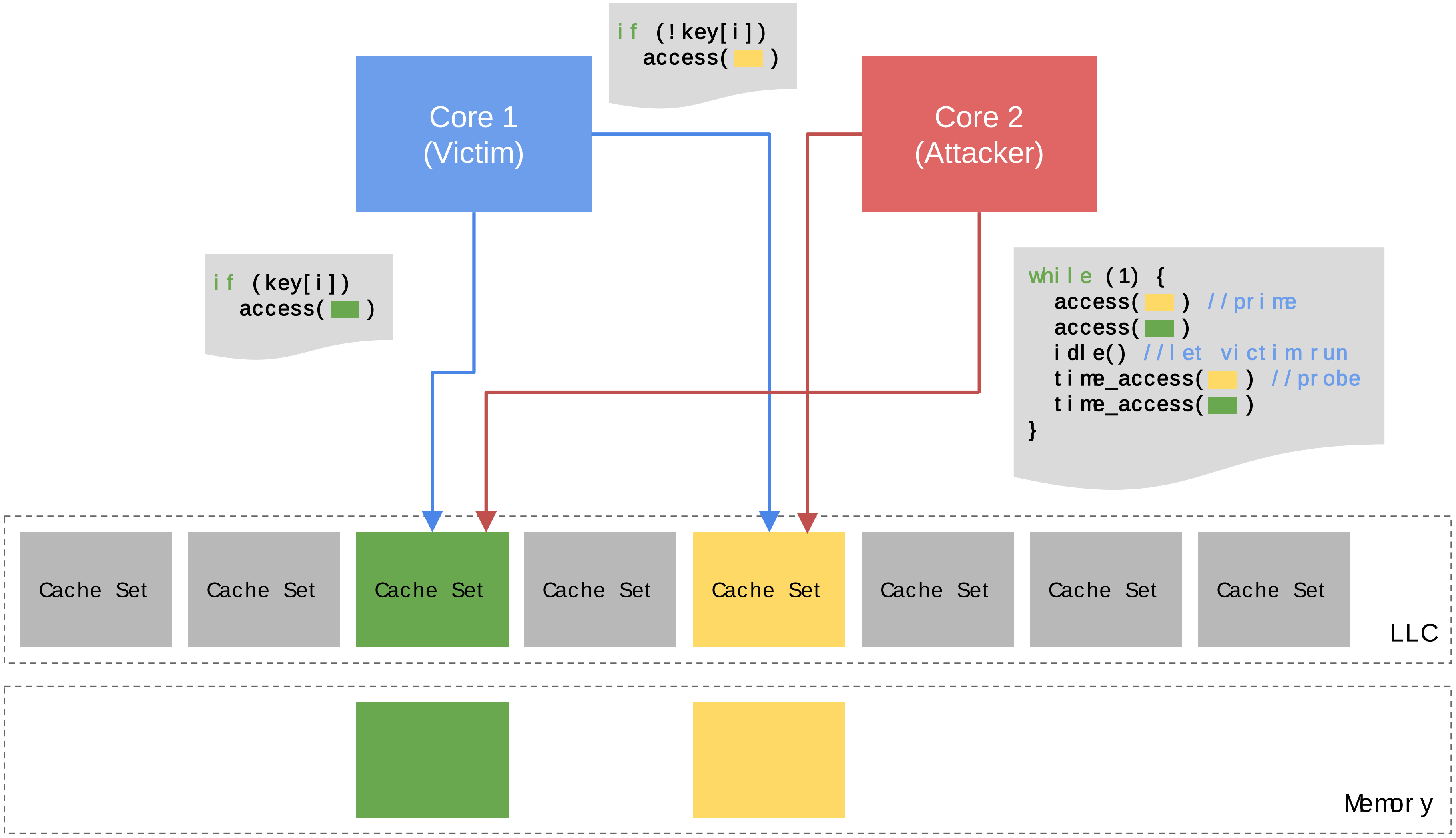




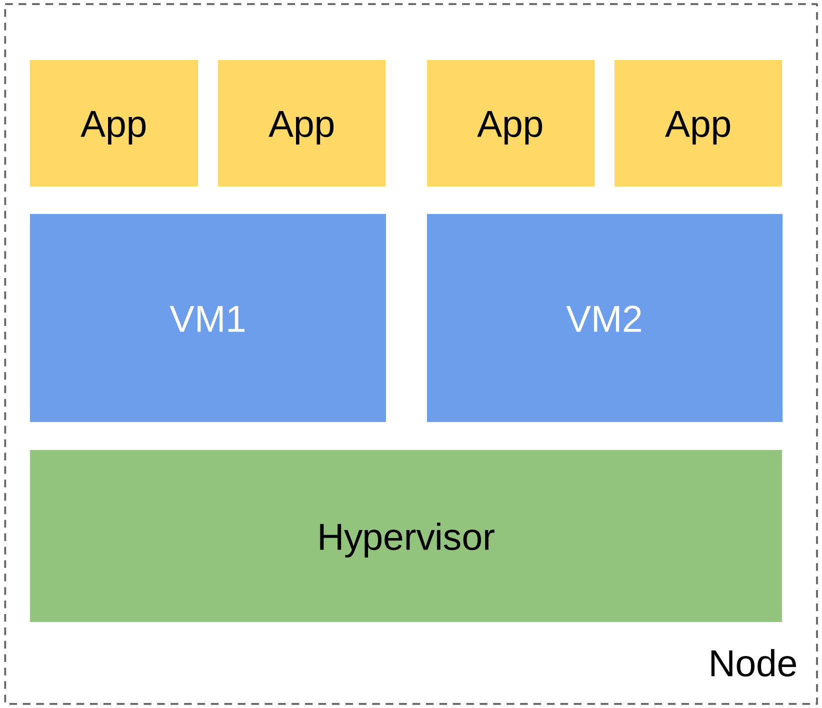
Background: Cache Allocation Technology

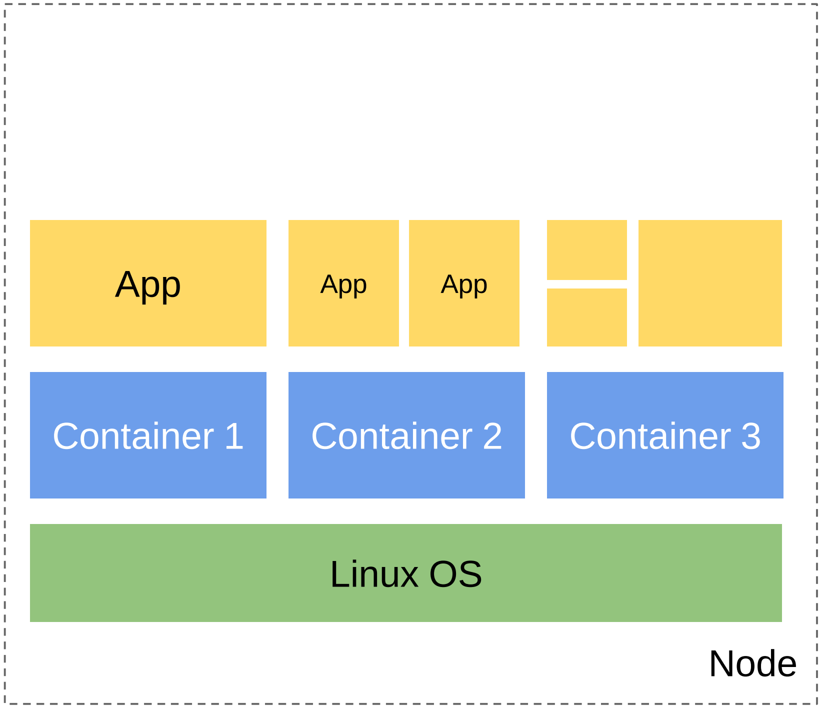


Background: Attack Example



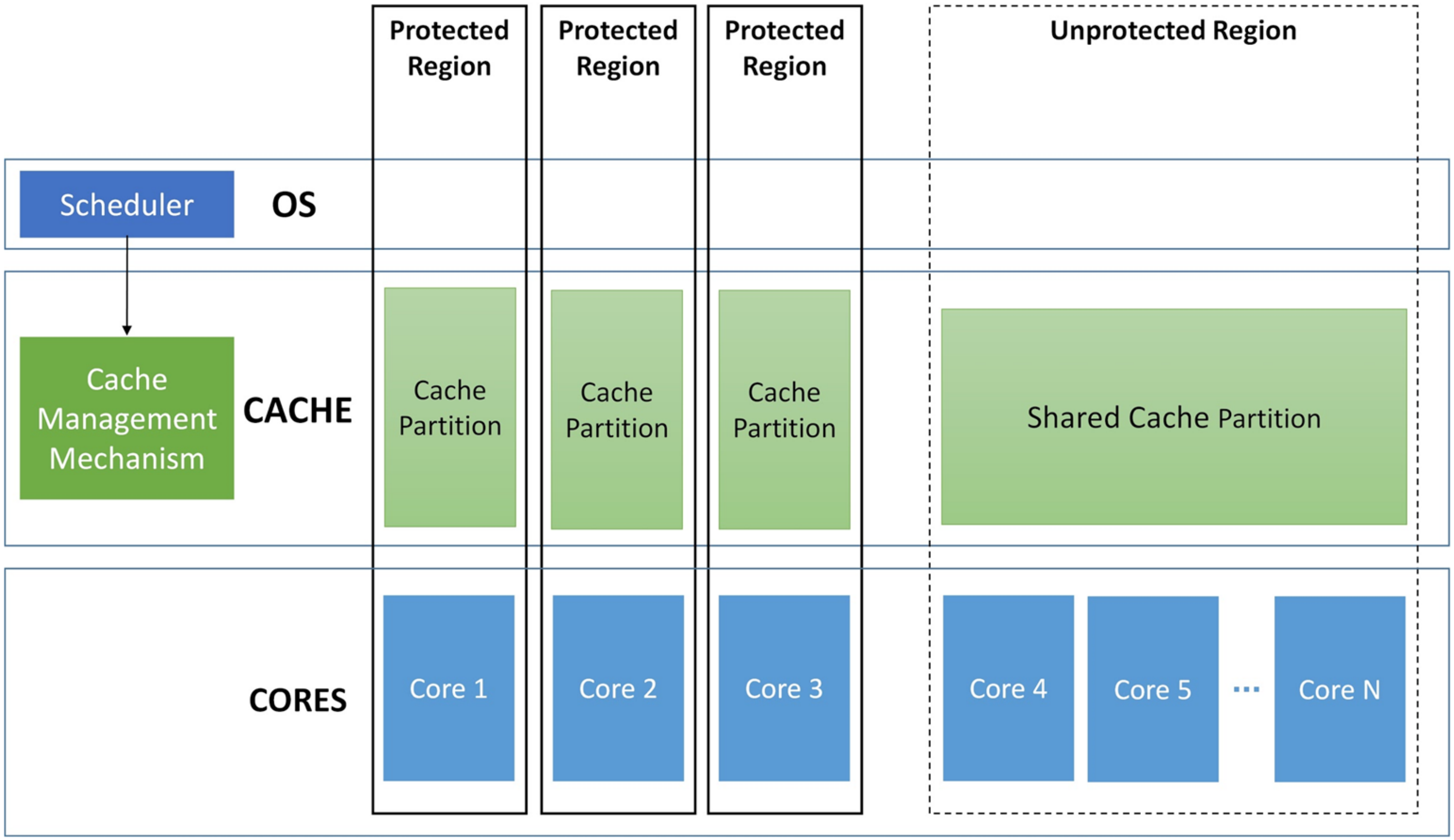
Background: Linux Containers





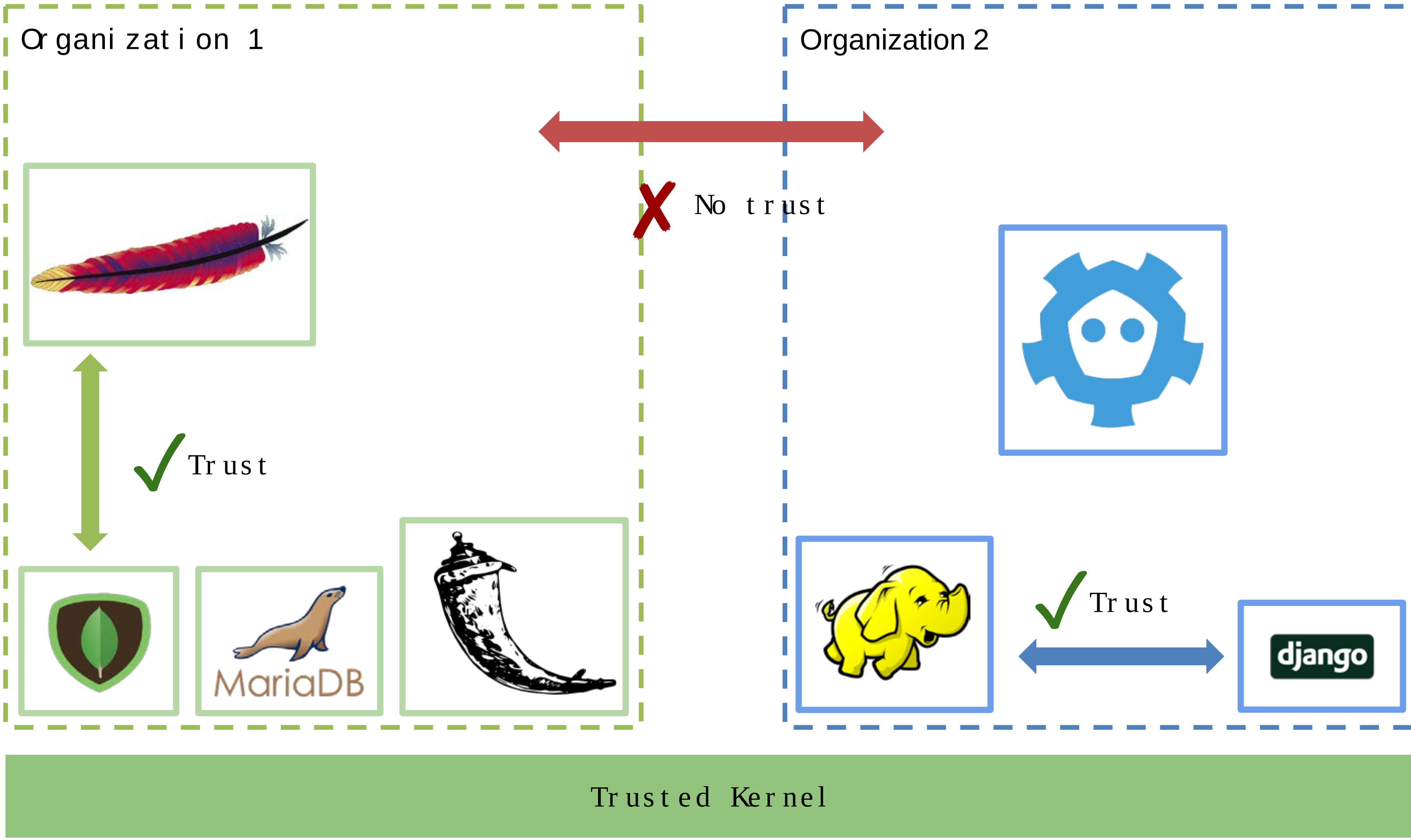
- ✓ Just a process within the kernel
- ✓ Isolated with cgroups and namespaces
- ✓ Scheduled by default Linux scheduler

Initial System Design

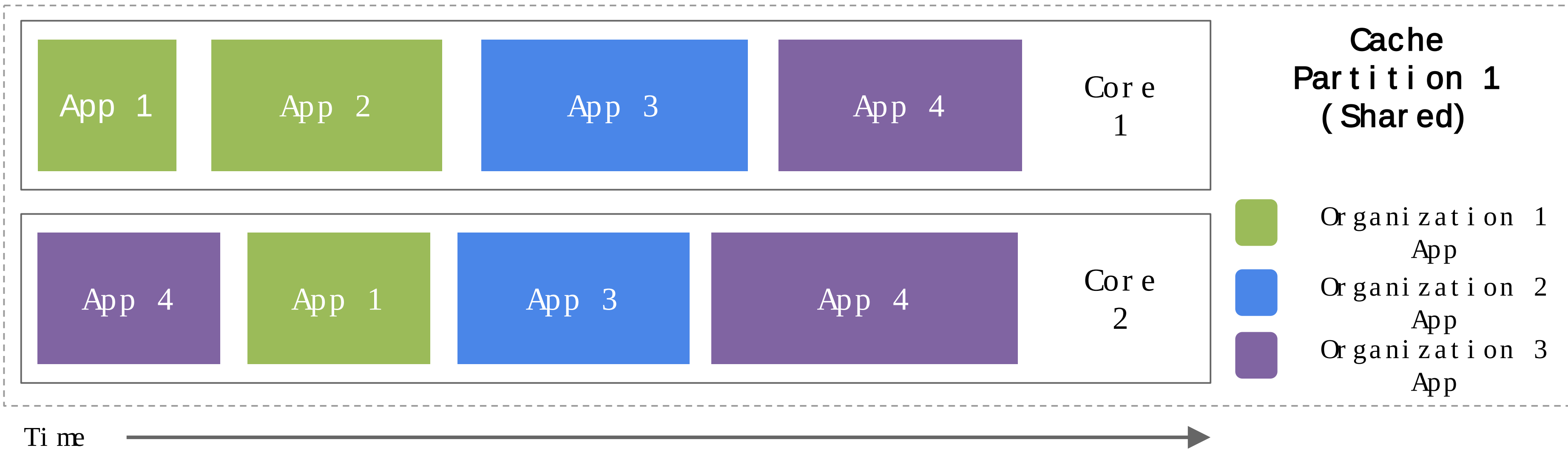


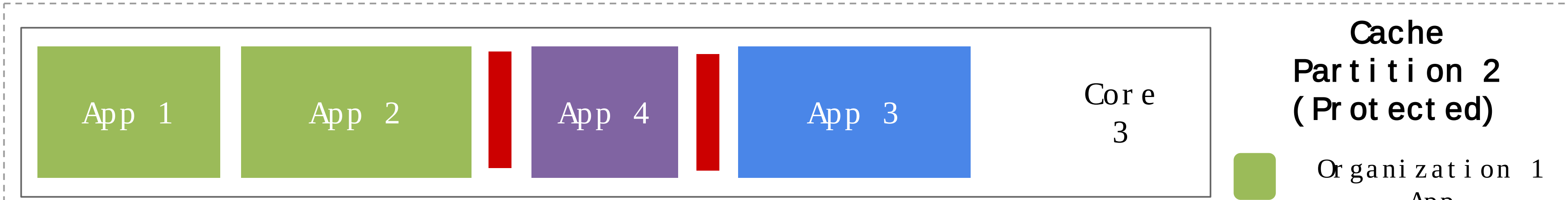
Secure partition per core is expensive, stay tuned

Introducing labels



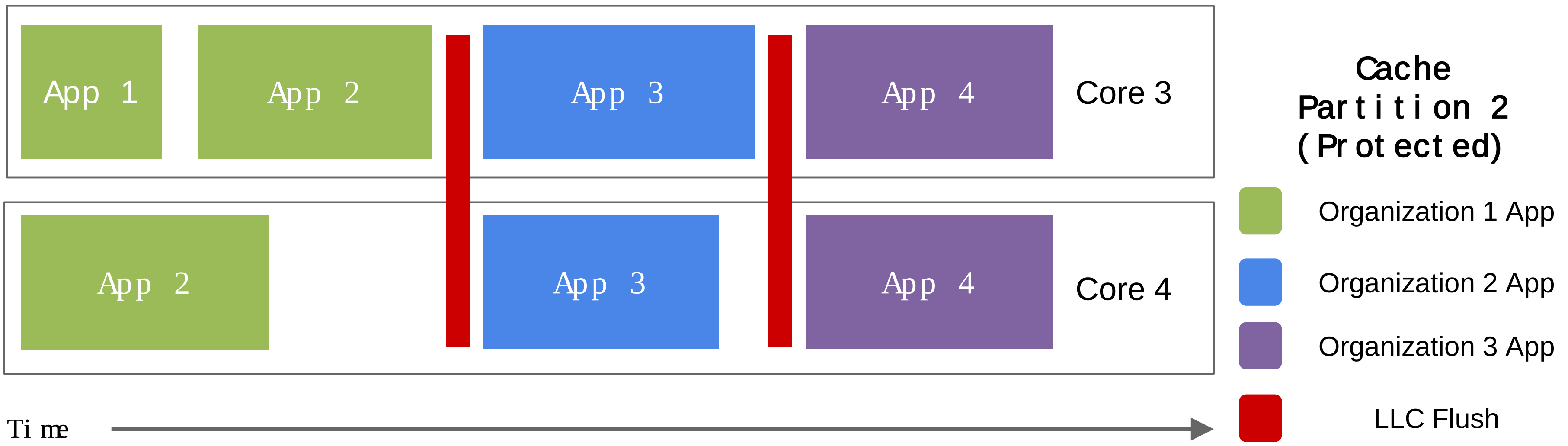
Mitigation: Naive Approach





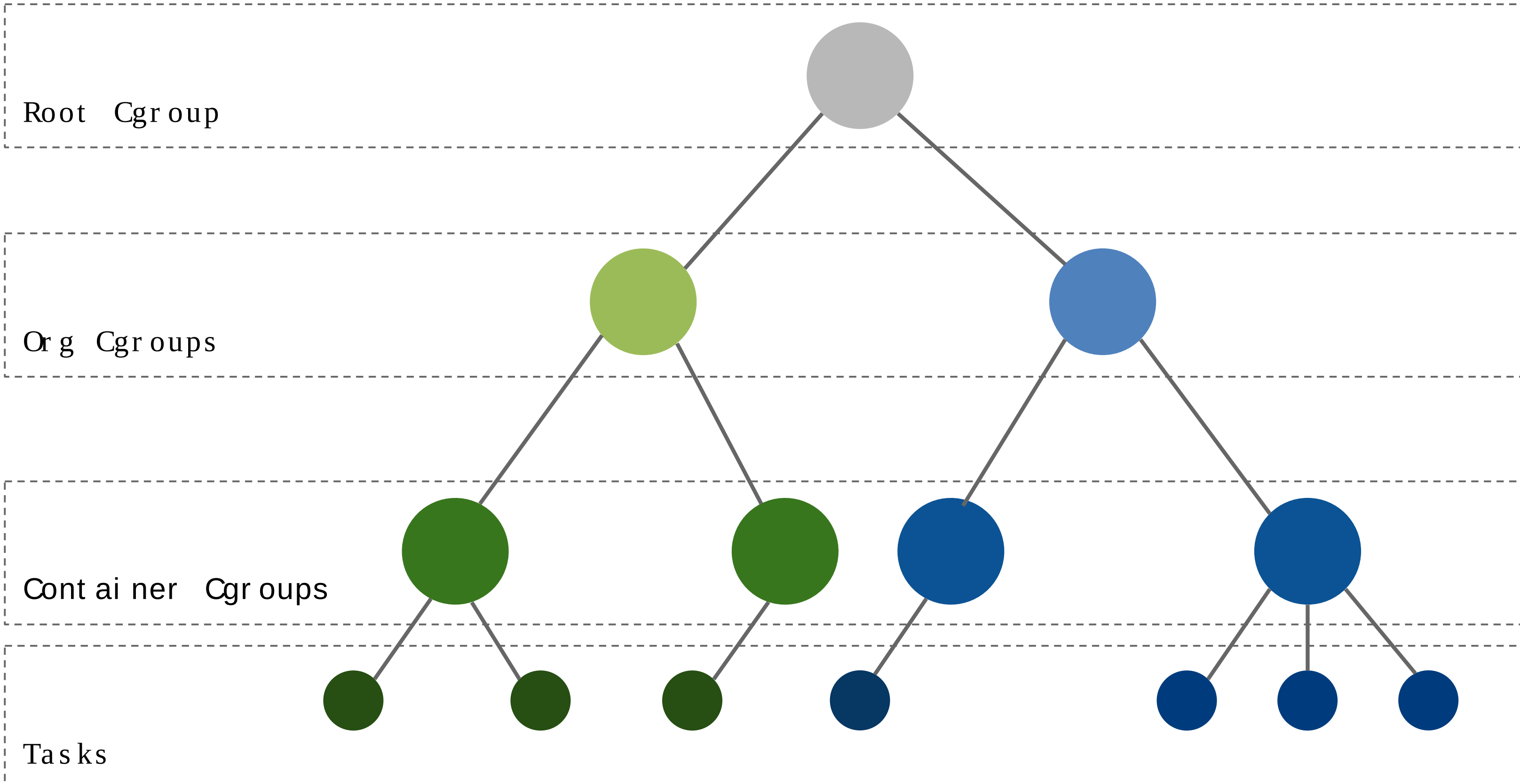
- ➔ Flushing the cache eliminates information leak
- ➔ By using CAT we assign smaller partition to security-sensitive apps
- ➔ Flushing smaller partition reduces overhead

Mitigation: Improved Approach

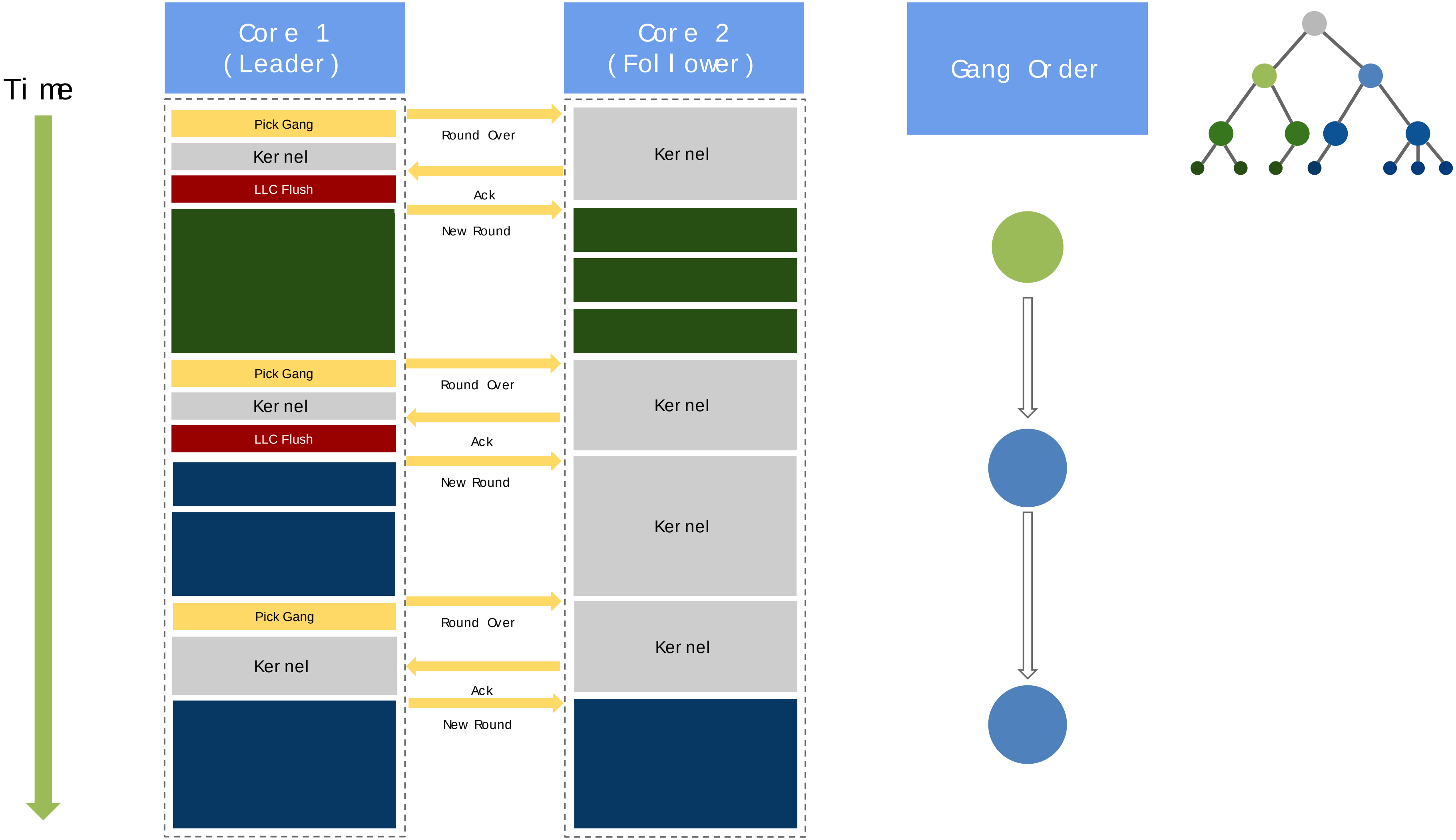


- ➔ Gang-schedule apps from the same organization
- ➔ Reduces the number of flushes
- ➔ Potentially increases idling (workload-dependent)

Implementation: Cgroup Hierarchy



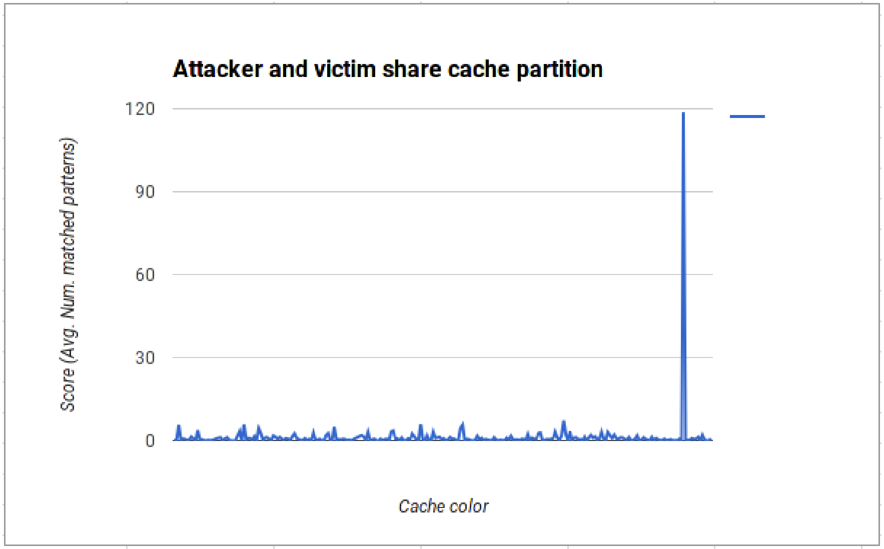
Follow-the-leader Algorithm

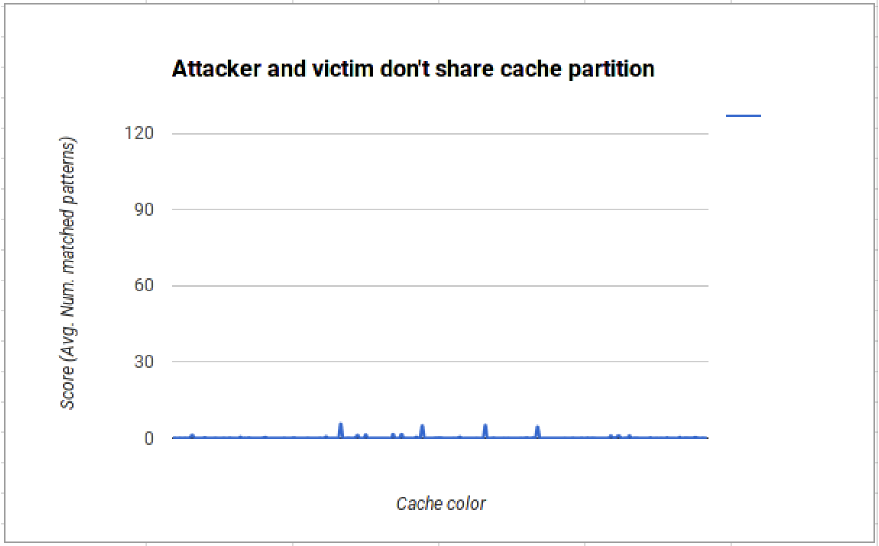


Challenges

- Reducing the idle time
- Minimizing flushing overhead
- Improving synchronization overhead
- Reducing amount of gang switches
- Improving fairness
- Scalability to the number of cores in secure partitions

Initial results





Complementary work

Container Live Migration recently introduced by:

- Virtuozzo
- runC
- Jelastic

Possible to combine the approach with Nomad

Future Work

- Improve the cost of synchronization
- Move to lazy-per-core gang changing
  - ✓ Using advanced features of CAT
  - ✓ Dynamically change cache partitions
  - ✓ No leader is needed
  - ✓ Significantly reduces synchronization
- Extend Docker framework for Flush+Reload mitigation
- Extensive performance evaluation

Discussion

Pros

- Transparent to apps
- Non-secure apps are not affected (almost)
- Easy to deploy
- Secure by design (not probabilistic defense)

Cons

- Requires the notion of organization
- Requires separating apps (secure/non-secure)
- Requires CAT
- Potential overheads for secure apps