Tomasz Tuzel

Who Watches the Watcher?

Detecting Hypervisor Introspection from Unprivileged Guests

Overview

"The Cloud"

- Numerous organizations are moving en masse to **the cloud**
  - It's easier to **manage**
  - It's easier to **scale**
- This results in a loss of control of the **physical hardware** and **privileges on the system**
  - Users are tied to guest virtual machines

Overview

Hypervisors (Virtual Machine Monitors)

- The root-of-trust is the hypervisor (virtual machine monitor)
  - It can introspect on guests with **few restrictions**
  - It can introspect on guests with **little evidence of apparent action**

Overview

Hypervisors (Virtual Machine Monitors)

- This trade-off puts organizations and individuals in a difficult situation
  - They have **sensitive data and processes**
  - Compromises are **expensive and dangerous**

Overview

Related Work

- **Compromise of** or **introspection by** a hypervisor has been a known issue
- Previous work on hypervisor detection has focused on **hardware or software artifacts**
  - This is more oriented towards **detection** of past events rather than active **introspection**

Motivation

- Hypervisor's activities are **not entirely invisible** to its guests
- **System performance is impacted** as the hypervisor seeks to provides an environment which is functionally equivalent to native hardware
- Increases in an instruction's **execution time and/or cache artifacting** can provide evidence of a hypervisor's intervention

Implementation

- A **test framework** emulates inappropriate introspection
- A **monitoring module** on the guest employs a set of sensors to detect malicious behavior

Test Framework

- The *Xen Project* was used as the hypervisor
- Modifications were made to support hypercalls which would toggle introspection capabilities
- Modifications were made to support instructions that *Xen Project* does not support VM-exiting for

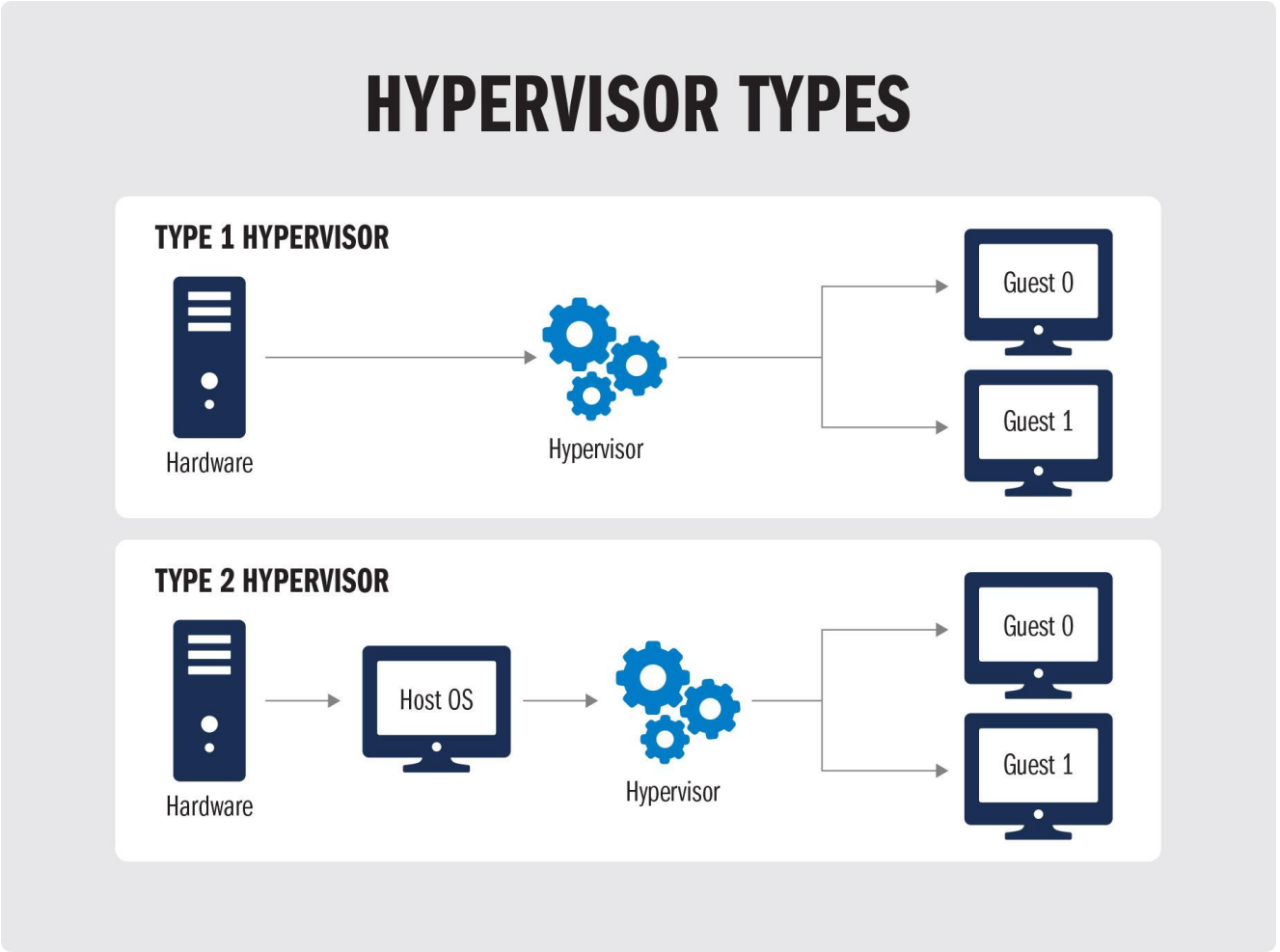
Monitoring Module and Sensors

- **Instruction intercession sensors** detect intercession of instruction execution using intercession timing
- **Active memory intercession sensors** detect the hypervisor actively interceding in memory access operations
- **Passive memory monitoring sensors** detect when a hypervisor accesses memory externally to a guest
- **Non-temporal access sensors** are not completed, but could be used to detect hypervisor introspection via non-caching page mappings or non-temporal instructions

Background

Hypervisor

- Privileged software that handles **execution** and **isolation** of guest virtual machines



Background

Virtual Machine Exits (VM-Exits)

- A hypervisor must be able to intercede in guest execution as necessary
- Guest execution is **paused** and execution is **handed to the hypervisor**
- Guest state information is stored in the Virtual Machine Control Structure (VMCS)
  - Resides in memory
  - Exits require that **state information is saved off during transition**
  - As such, an exit incurs significant overhead

Background

Timers & Timing Methods

- Time Stamp Counter (TSC)
- High Precision Event Timer (HPET)
- Thread racing

Background

Memory

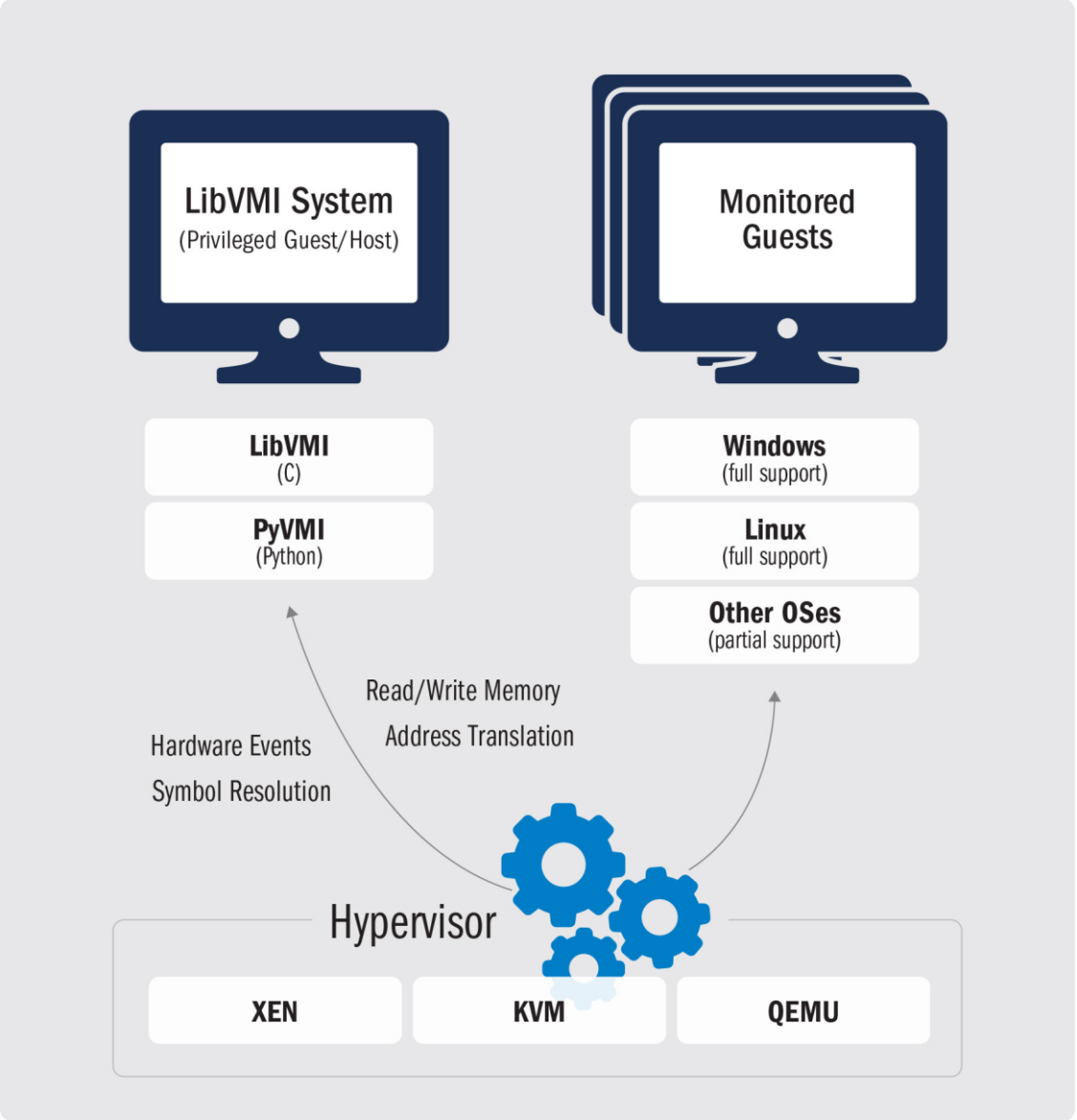
- Modern operating systems use hardware accelerated paging

- When it comes to virtual machine introspection, much of that hardware acceleration is unavailable

Background

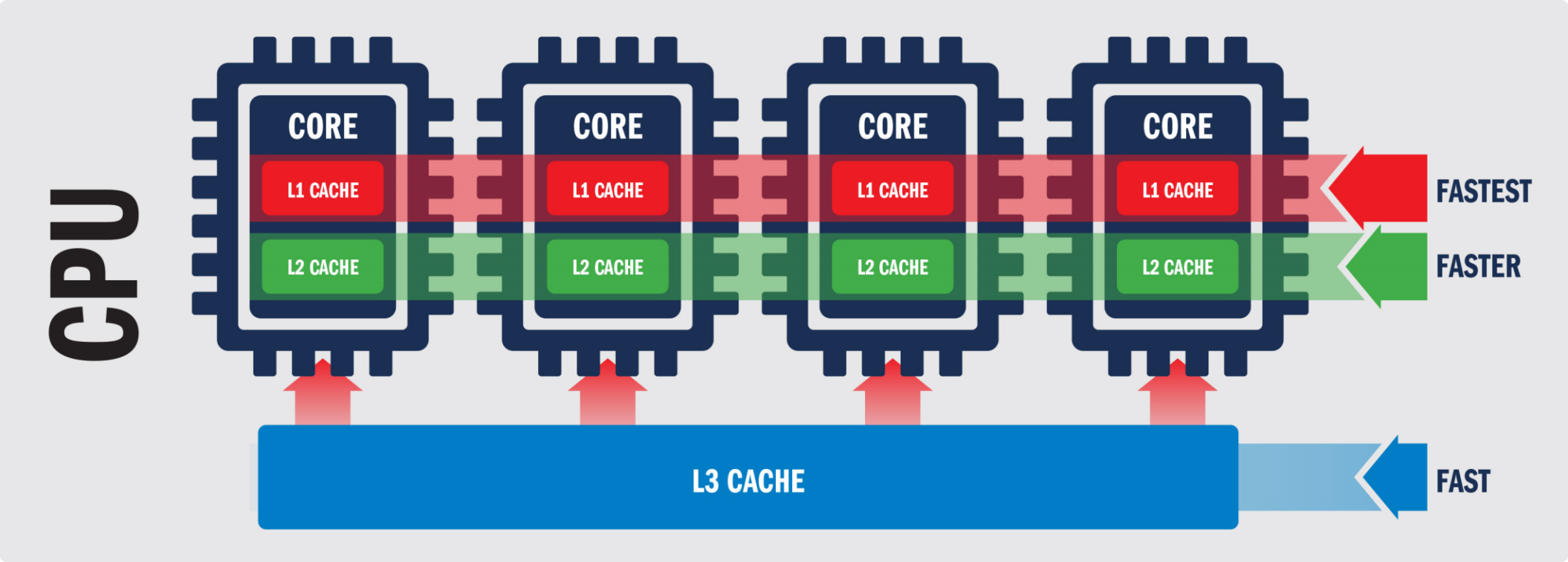
LibVMI

- C and Python library that enables introspection on virtual machines



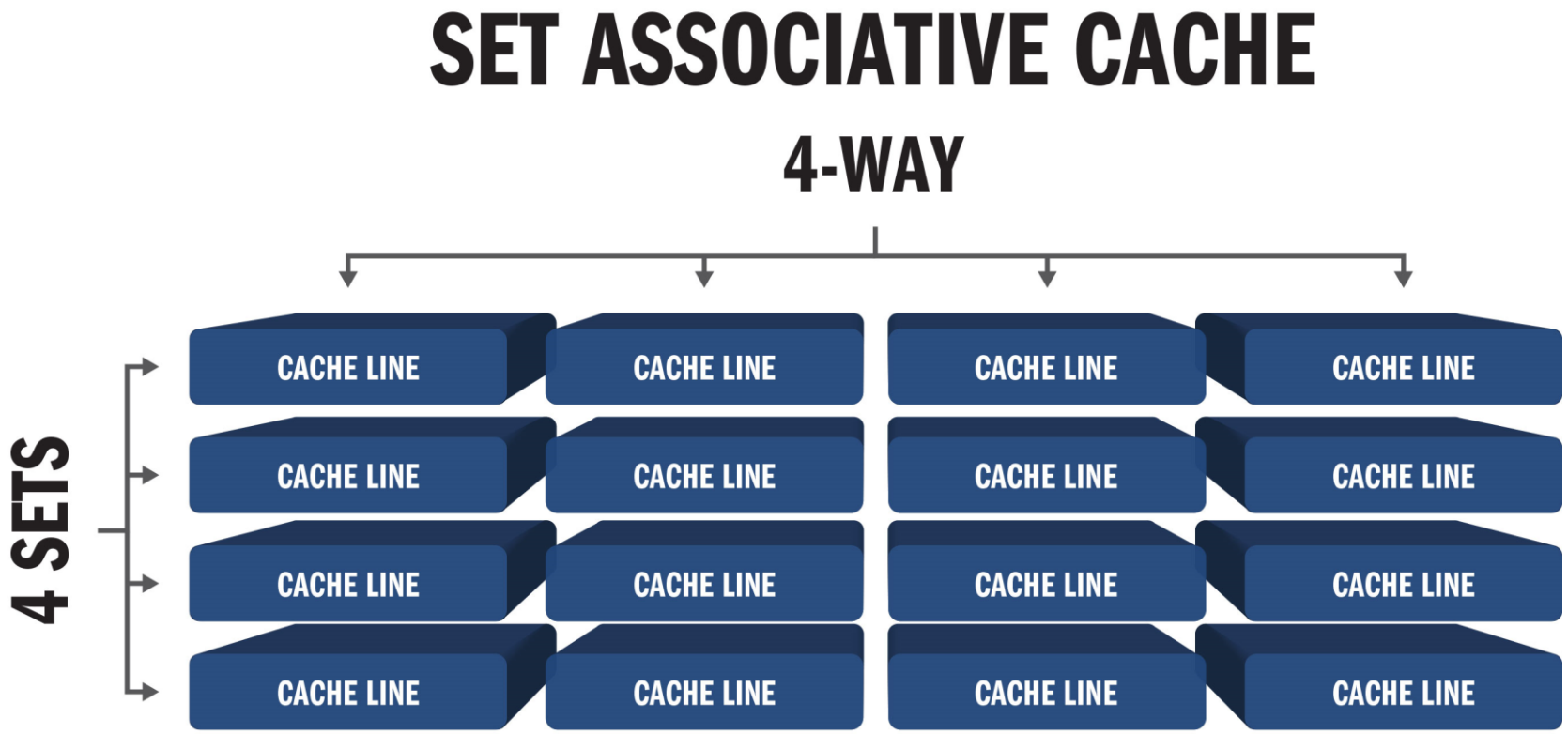
Background

Cache



Background

Cache



Background

Cache Side-Channel Attacks

- **Flush+Reload**: Attacker flushes shared virtual address, waits, times reload time of cache line
- **Prime+Probe**: Attacker primes cache, waits, probes and times access to the cache line
- **Evict+Time**: Attacker waits for cache sets to be loaded, evicts memory lines, waits, evicts again and times eviction
- **Flush+Flush**: Attacker continuously flushes shared memory line, timing flushes
- **Prime+Abort**: Uses Intel TSX to open a transaction, access memory, and wait for an abort. A hardware callback reveals that access has taken place.

Instruction Intercession Sensors

- A hypervisor can trap on certain instructions to:
  - **Modify** the guest's execution behavior
  - Determine when a guest **performs** various operations
  - **Extraction** of information
- Timing can be used to determine that this is happening, and how much work the hypervisor performed while trapping

Instruction Intercession Sensors

- Some instructions trap unconditionally
  - CPUID
- Some instructions trap conditionally
  - MOV to CR3

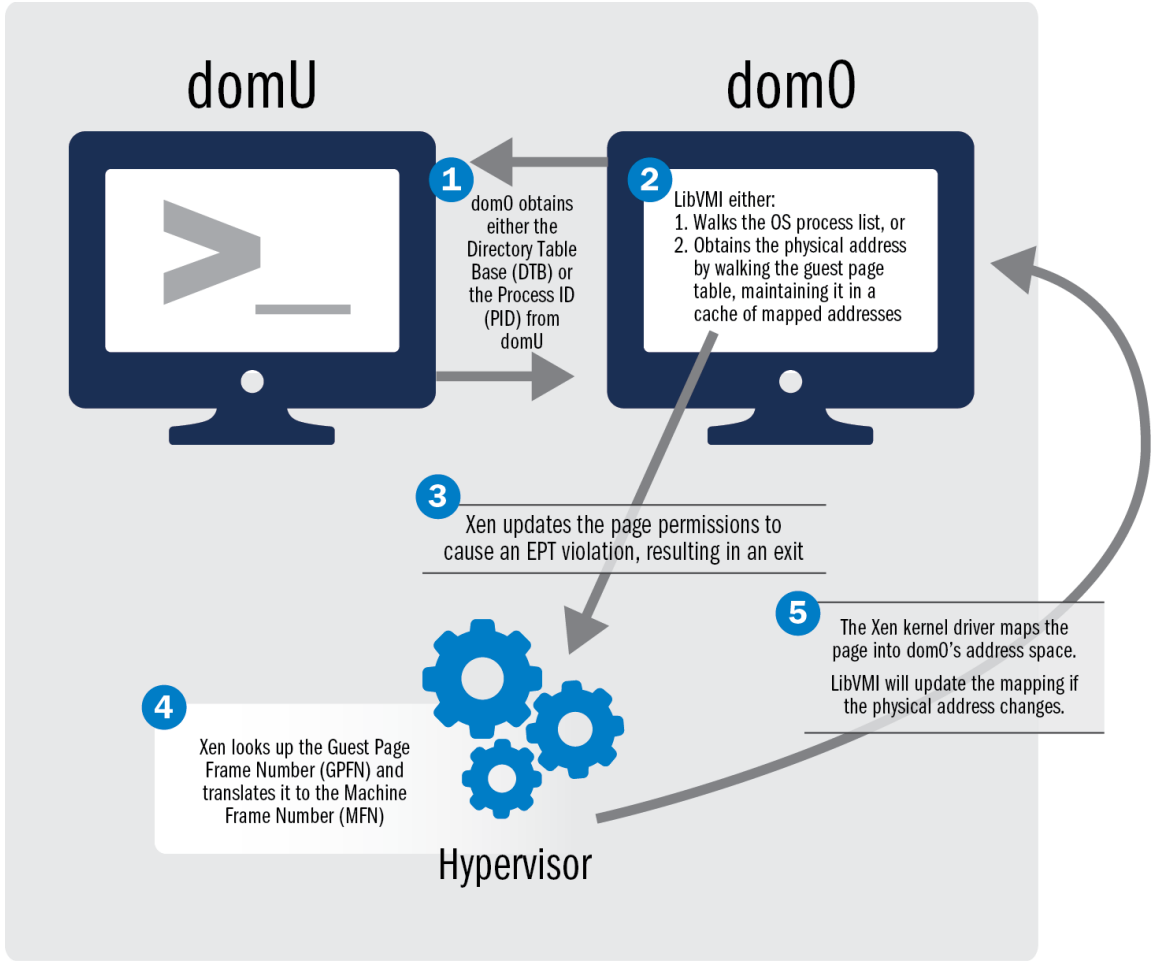
Instruction Intercession Sensors

- Wall timing via the HPET
  - Manipulation of the HPET would result in **noticeable interruptions** in operations to the observer
- Thread racing
  - Parallel threads run, one of which executes NOPs, with the number of executions counted upon completion

Active Memory Intercession Sensors

- Memory accesses can trap
  - EPT violations
- Virtual to physical memory mappings can be marked to **cause a VM-exit** (using LibVMI)
- VM-exit incurs a **sizable overhead**, and resultant **large timing increase**

Active Memory Monitoring Sensors



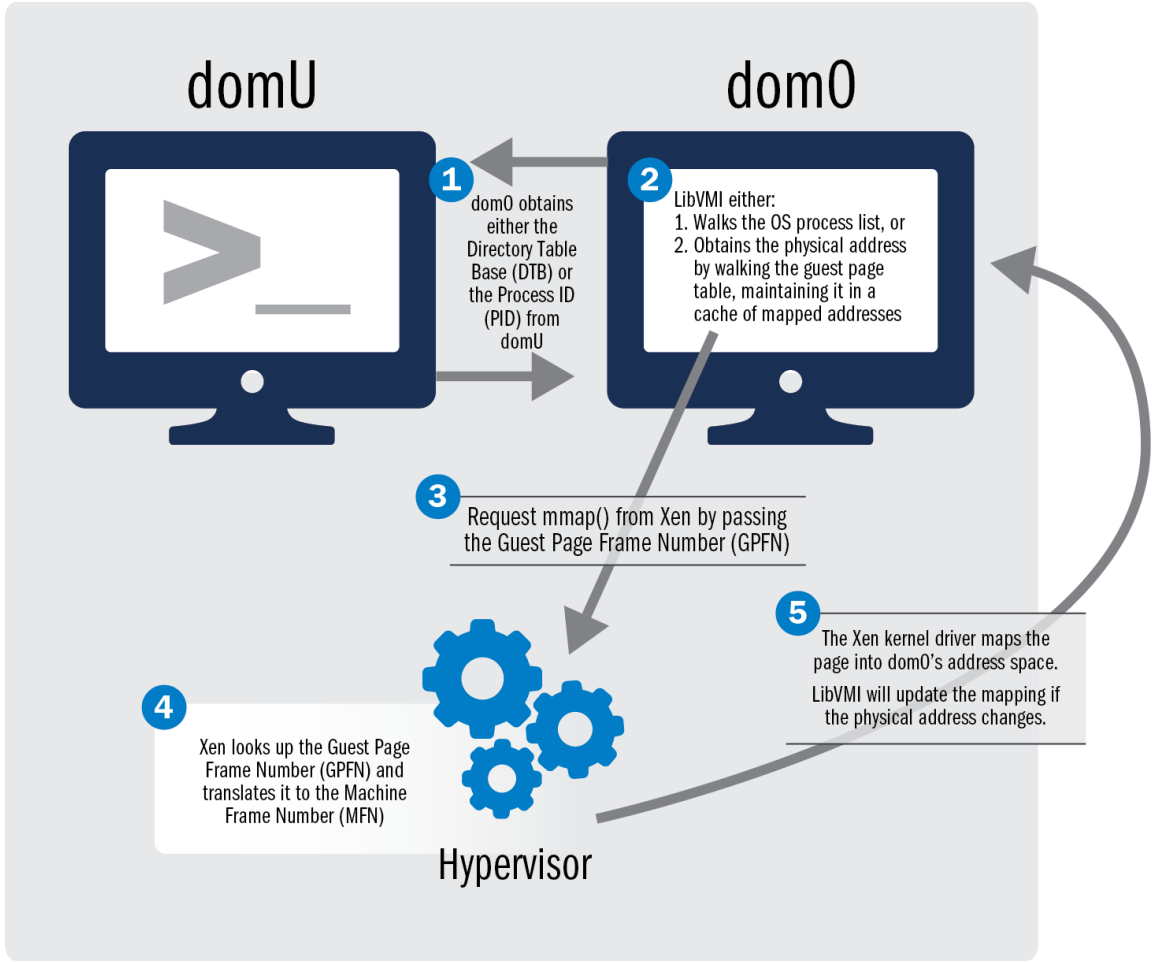
Passive Memory Monitoring Sensors

- A more stealthy hypervisor can choose to **map the guest's physical pages** into other contexts
  - The hypervisor can map memory into the privileged domain
- There are numerous translations occurring under the hood
  - Guest virtual to guest physical
  - Guest physical to host physical
  - Etc.

Passive Memory Monitoring Sensors

- Time required to access a memory line can be used
- Using Flush+Reload:
  - The memory line of interest is flushed from the cache
  - A period of time passes to allow for potential access to the memory region to occur
  - The memory line is reloaded, and the access is timed
- An **decrease in timing** is indicative of introspection

Passive Memory Intercession Sensors



Non-Temporal Access Experimentation

- Non-temporal, streaming, and vector instructions have cache-coherence side-effects, despite bypassing the cache
  - A non-temporal instruction reading/writing a populated page **triggers a cache flush**
- Intel's Page Attribute Table (PAT) allows specifying per-page caching behavior
  - Passive mappings can become **non-cache-interacting**

Analysis & Classification of Results

- A variety of machines are used which had different processor generations
- For instruction intercession, a baseline is derived from non-exiting instructions
  - Adequately high values can be flagged as potentially exiting, or of interest
- For memory intercession, baselines are observable from adjacent non-introspected memory

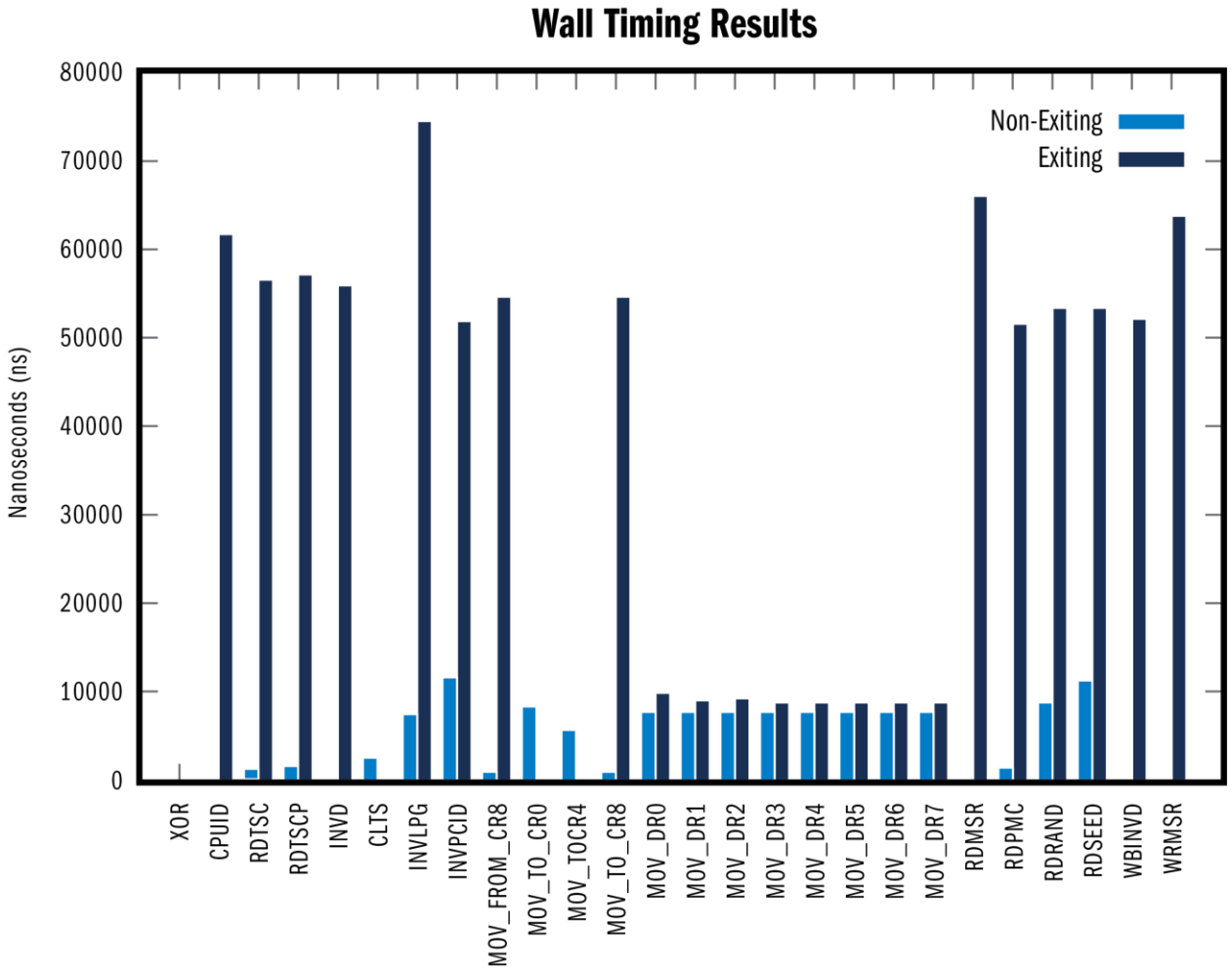
Results

Instruction Intercession

- Trapping by the hypervisor immediately returns, thus delivering the minimal possible impact on timing

Results

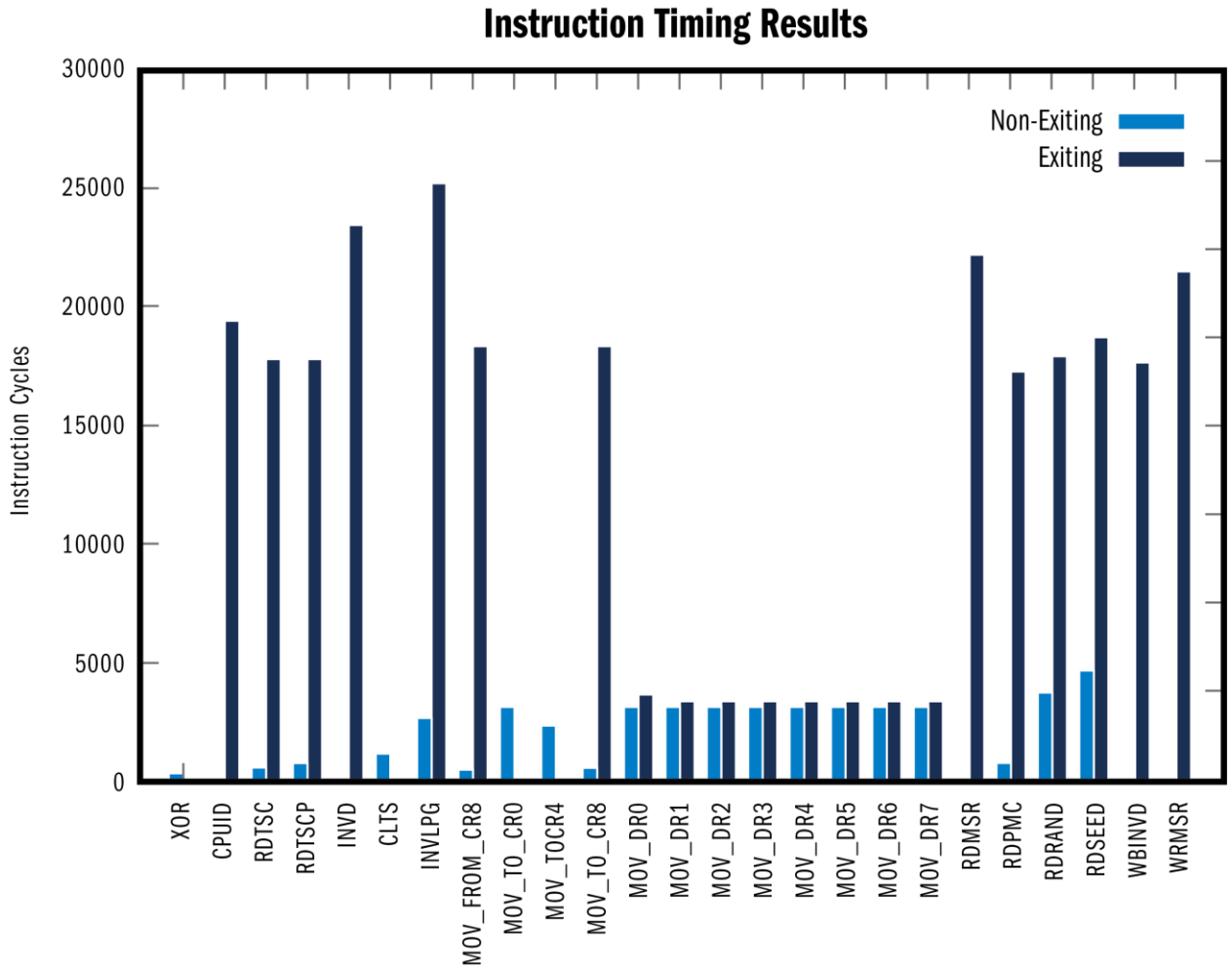
Instruction Intercession:
Wall Timing



Results

Instruction Intercession:

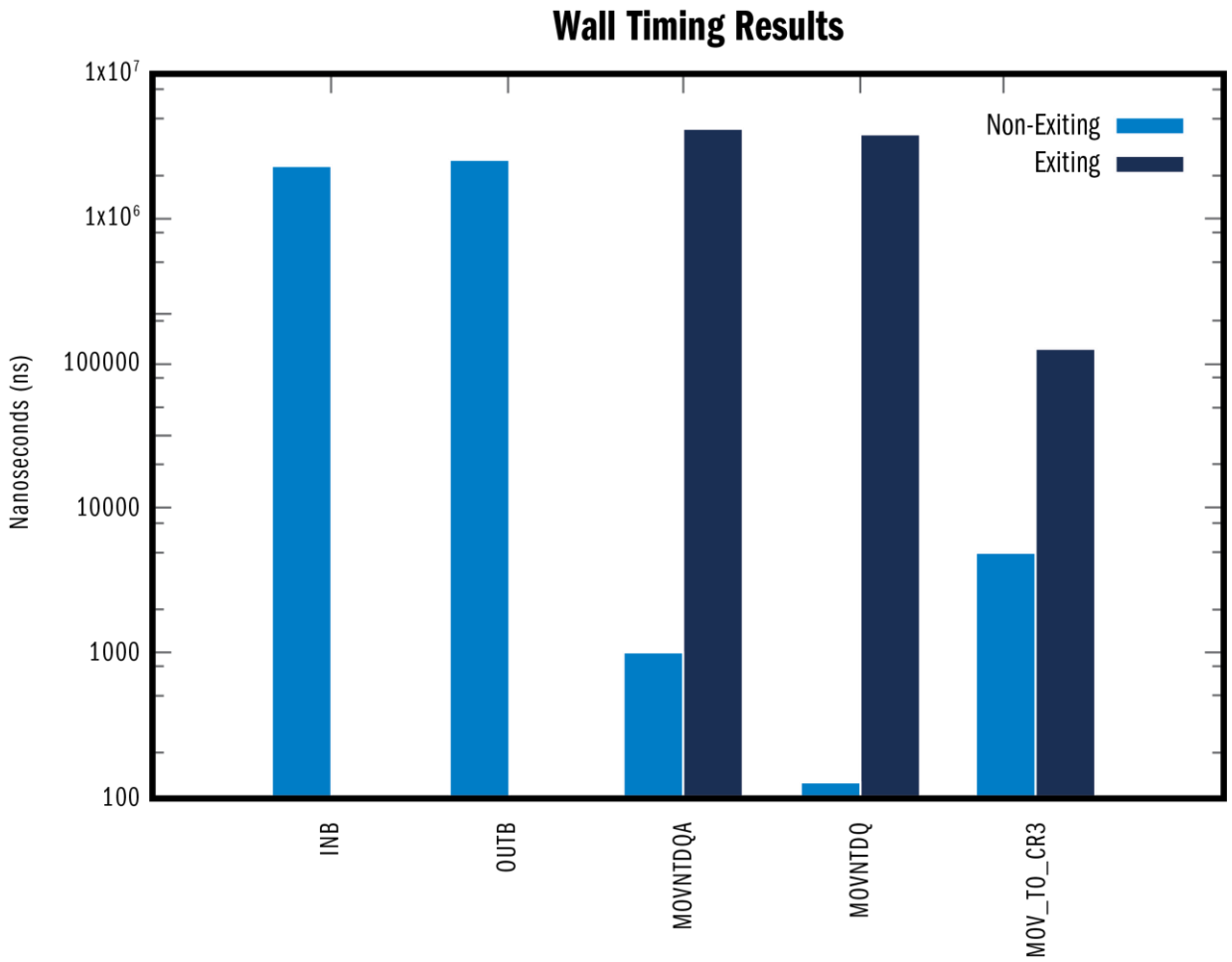
Instruction Timing



Results

Instruction Intercession:

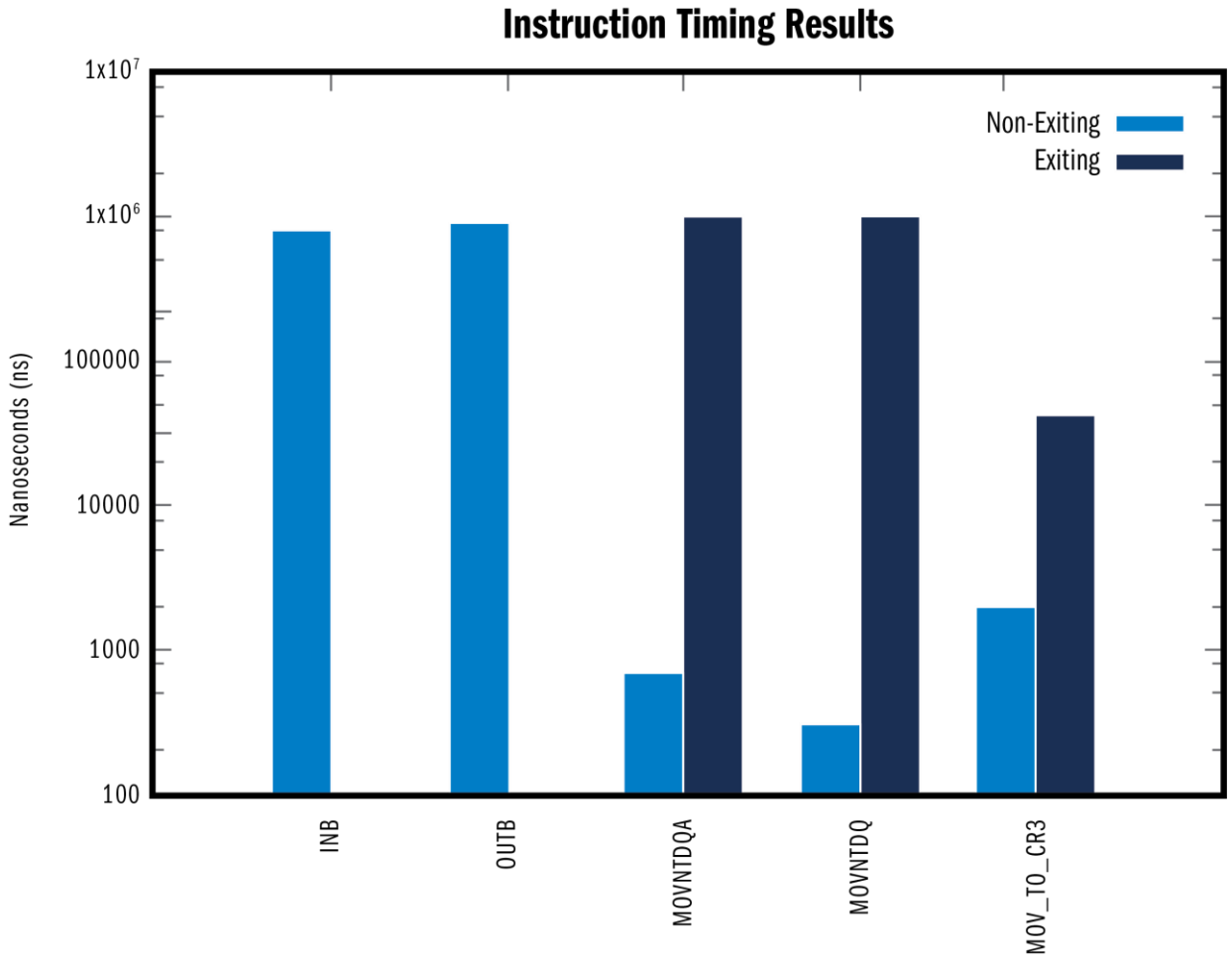
Wall Timing



Results

Instruction Intercession:

Instruction Timing



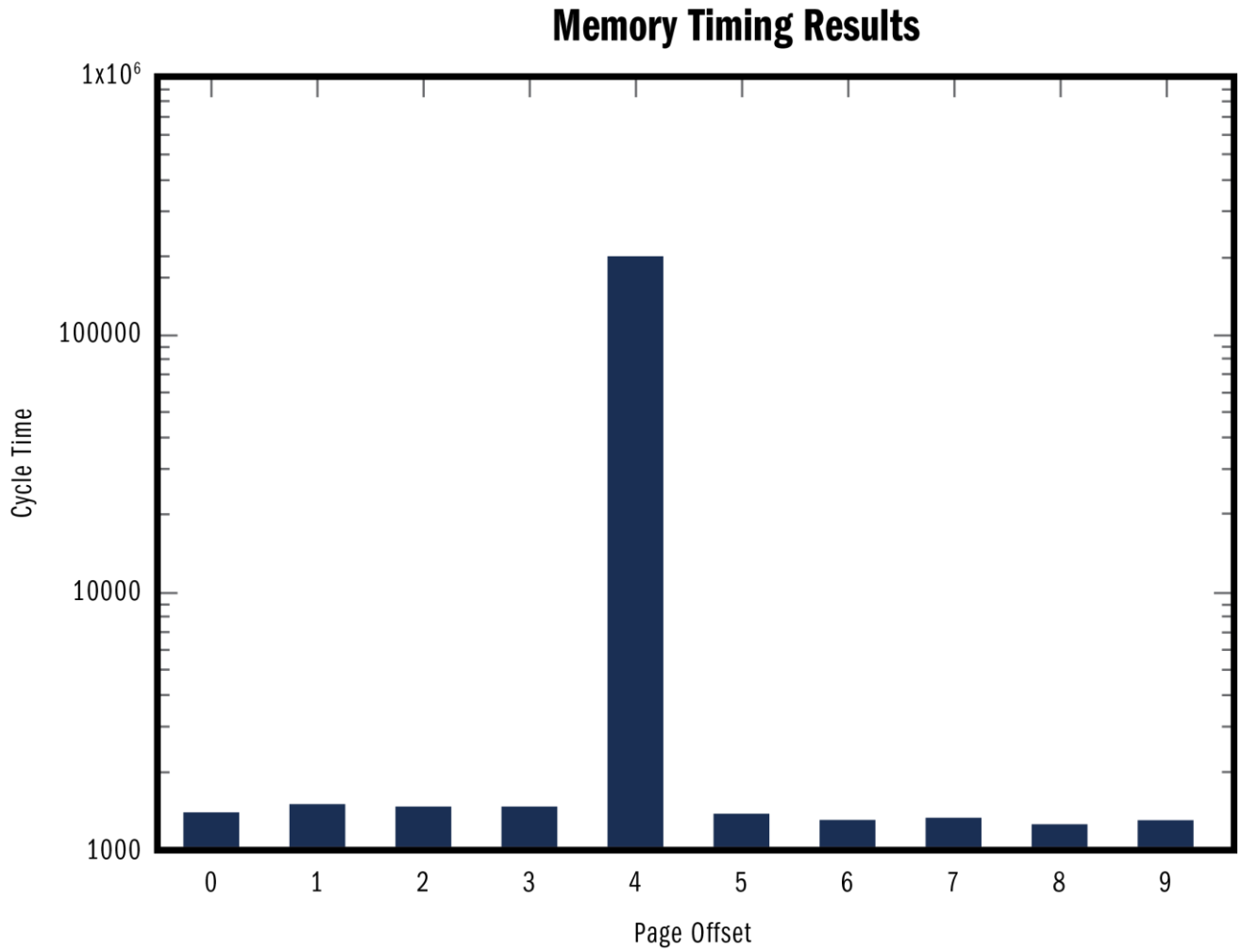
Results

Instruction Intercession

- Takeaway: Benign hypervisors trap as few instructions as possible

Results

Active Memory Intercession



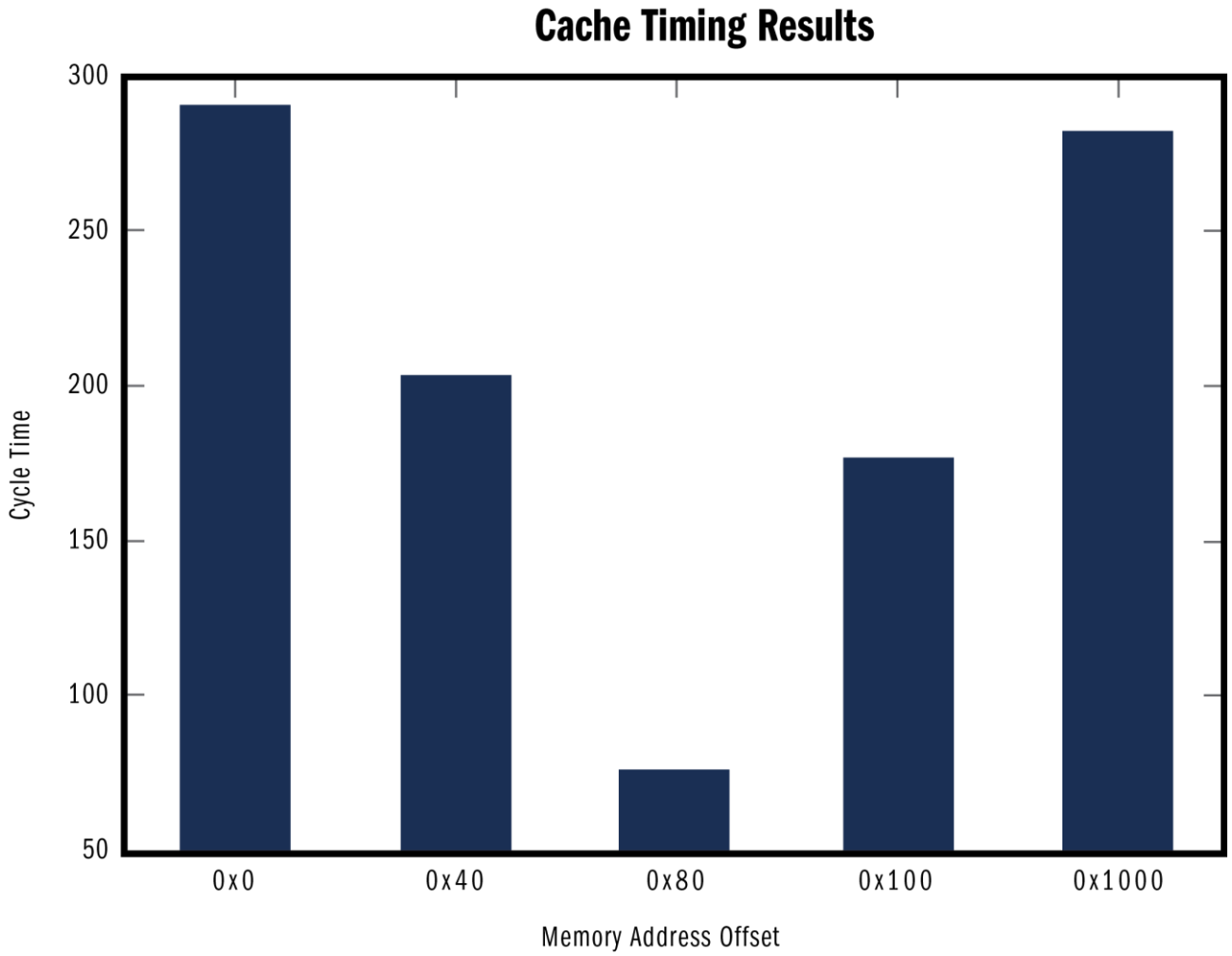
Results

Active Memory Intercession

- Takeaway: Benign hypervisors avoid trapping memory accesses whenever possible

Results

Passive Memory Intercession



Results

Passive Memory Intercession

- Takeaway: Shared-memory based passive memory accesses affect CPU caches

Future Work

Next Steps

- This is a **first look** into hypervisor detection technology
- A **continuous detection environment** could be implemented
- Full **binary classification** could better categorize introspection
- A **response strategy** when introspection is discovered

Guest Modes

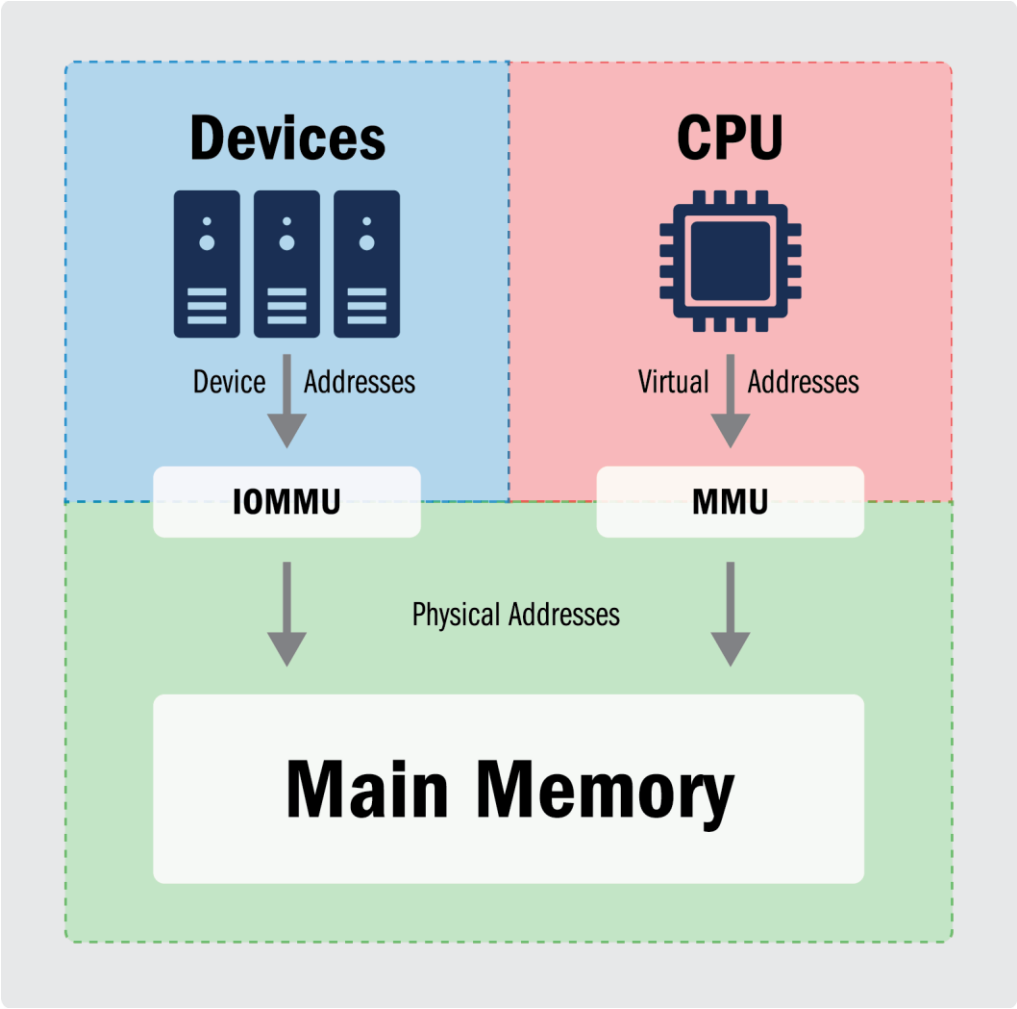
PV vs HVM vs PVH

- PV requires no hardware support (no VT-x)
  - Address space is **shared** for everything
- HVM requires VT-x
  - Uses **QEMU emulation** for devices
- PVH is the “new and improved” guest mode
  - Uses **VT-d** for devices
  - Is avoiding the use of QEMU

Other Technologies

VT-d

- Devices interface with main memory
  - Can a **rogue device** cause problems?



Other Technologies

New Virtualization Extensions

- Limit detection via timing since they **reduce overhead**
  - *Virtualization Exceptions (#VE)*
  - *VMFUNC*
- However, the guest must be aware of the exception(s)

Other Technologies

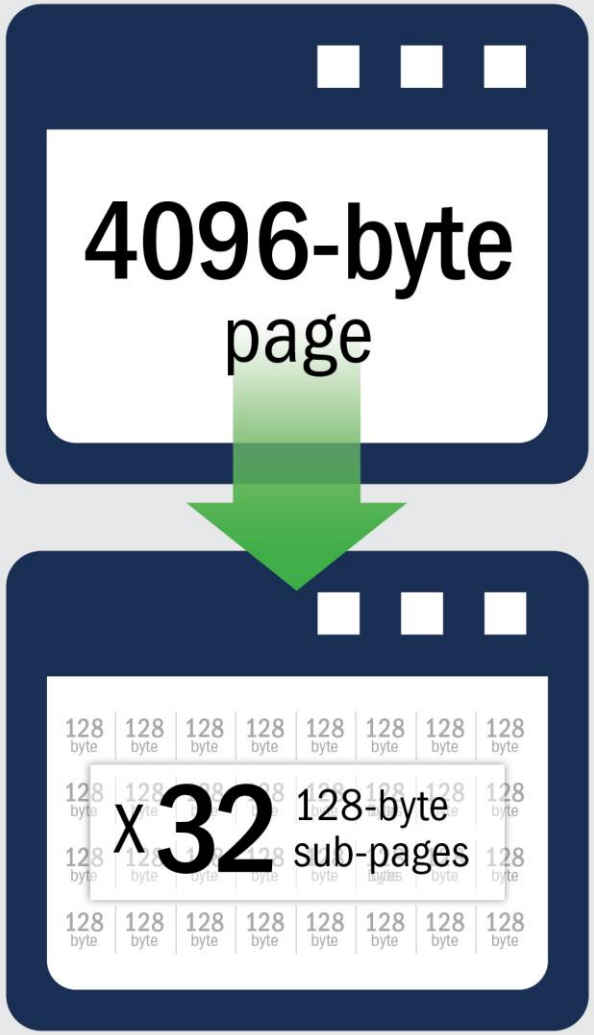
Different Cache Attacks

- Prime+Abort, along with naïve TSX-based techniques, permits the use of **hardware callbacks** rather than timers
  - Mark memory
  - Wait for access of targeted memory to occur
  - Abort status indicates access

Other Technologies

Sub-Page Permissions

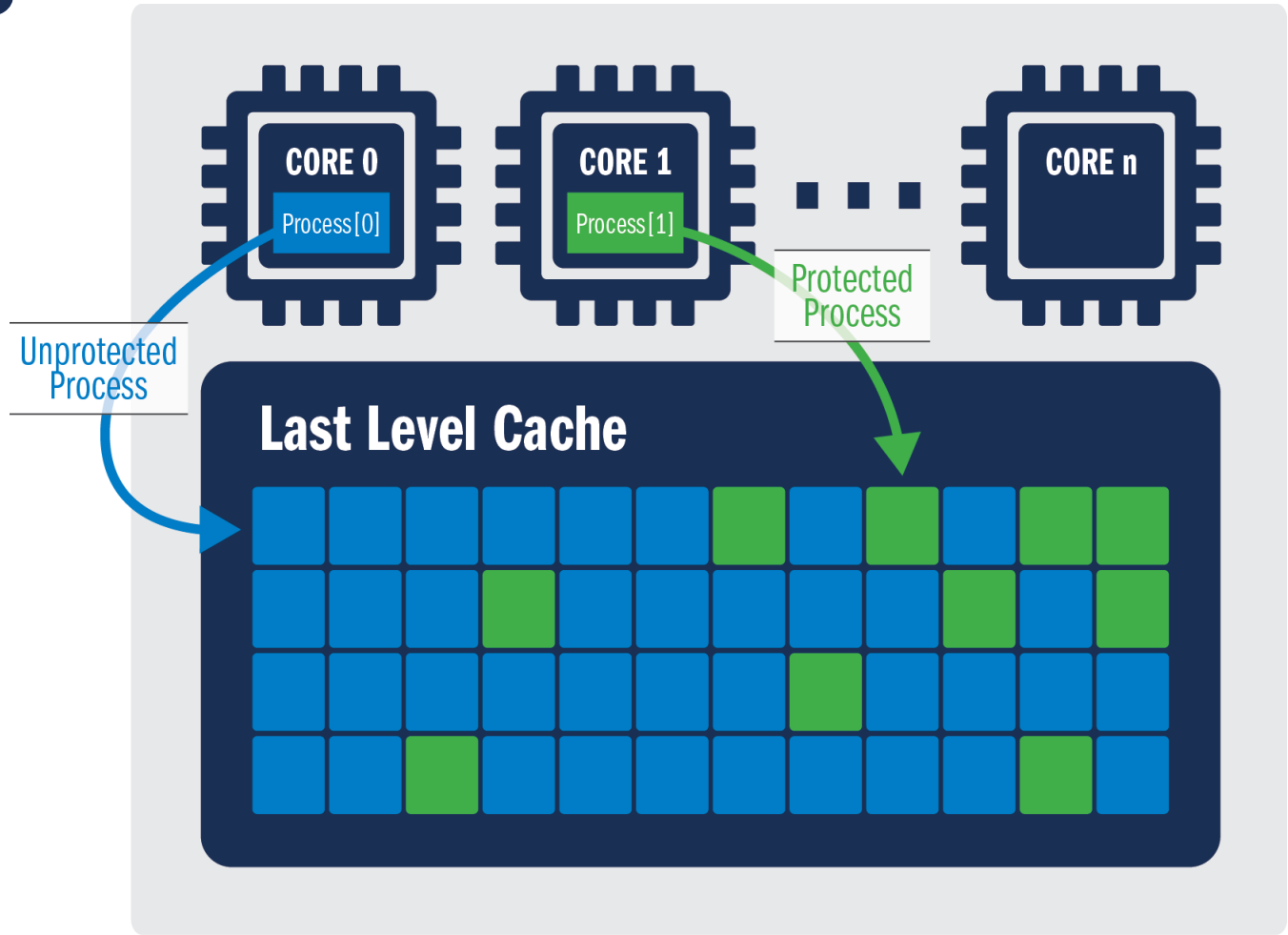
- Permit memory protections at a **much lower granularity**



Other Technologies

Intel Cache Allocation Technology (CAT)

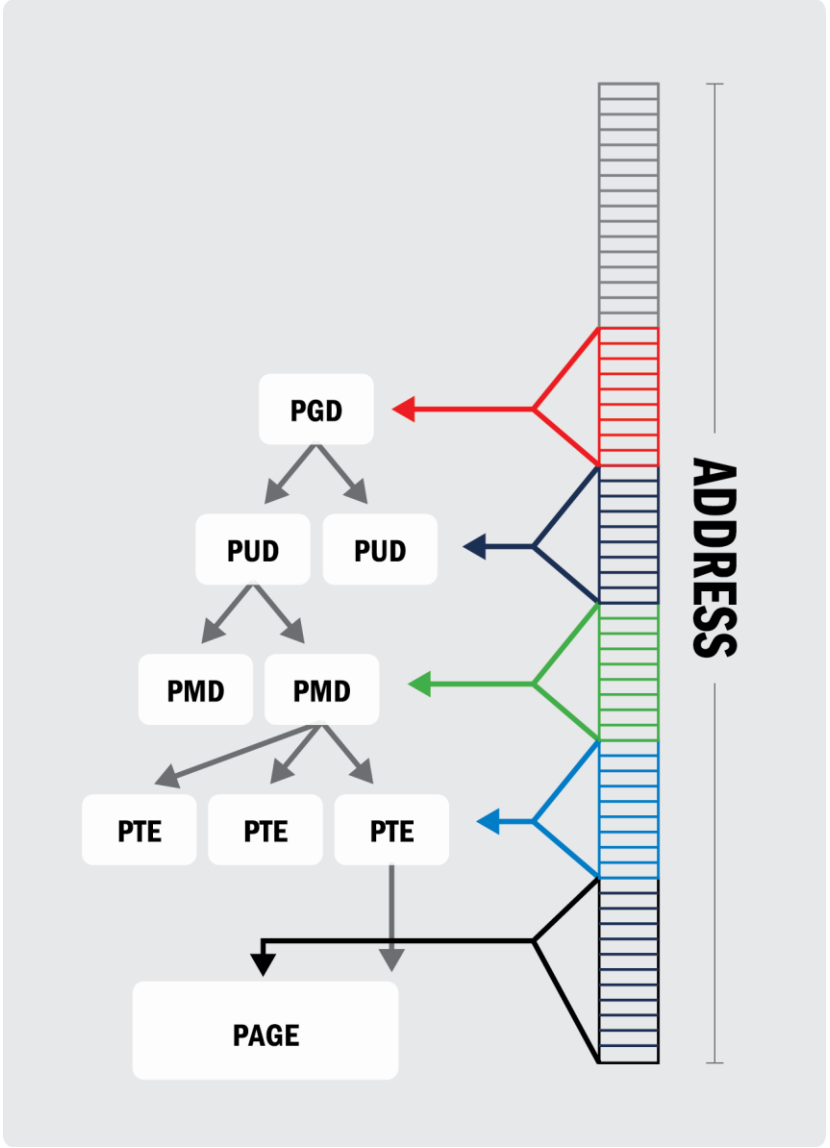
- Allows **isolation of cache space** to an individual process



Other Technologies

Five-Level Paging

- New *P4D* level in between *PGD* and *PUD*
- Increases 256 TiB of available virtual address space to 128 PiB
- Increases 64 TiB of available physical address space to 4 PiB
- The increase in address space makes building cache eviction sets easier



Other Technologies

Intel Software Guard Extensions (SGX)

- Provides proper isolation… Except when Foreshadow ends up in the mix

- ECDSA Attestation Key **permits offline support** of SGX

Demo

Conclusion

- Detection of instruction intercession is possible
- Active and passive memory monitoring is possible
  - Isolating a memory region to a specific process may be necessary
  - Active and passive monitoring used in conjunction could obfuscate results of timing techniques
- This work establishes the **limitations of hypervisor introspection detection**

Questions?

- We've open-sourced this work!
  - Toolkit: https://github.com/ainfosec/ecr_toolkit
  - Modified hypervisor: https://github.com/ainfosec/ecr_hypervisor
- We have a white paper (see the links in the Github README above)

This research is based upon work supported in part by the Office of the Director of National Intelligence (ODNI), Intelligence Advanced Research Projects Activity (IARPA). The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies, either expressed or implied, of ODNI, IARPA, or the U.S. Government. The U.S. Government is authorized to reproduce and distribute reprints for governmental purposes notwithstanding any copyright annotation therein.