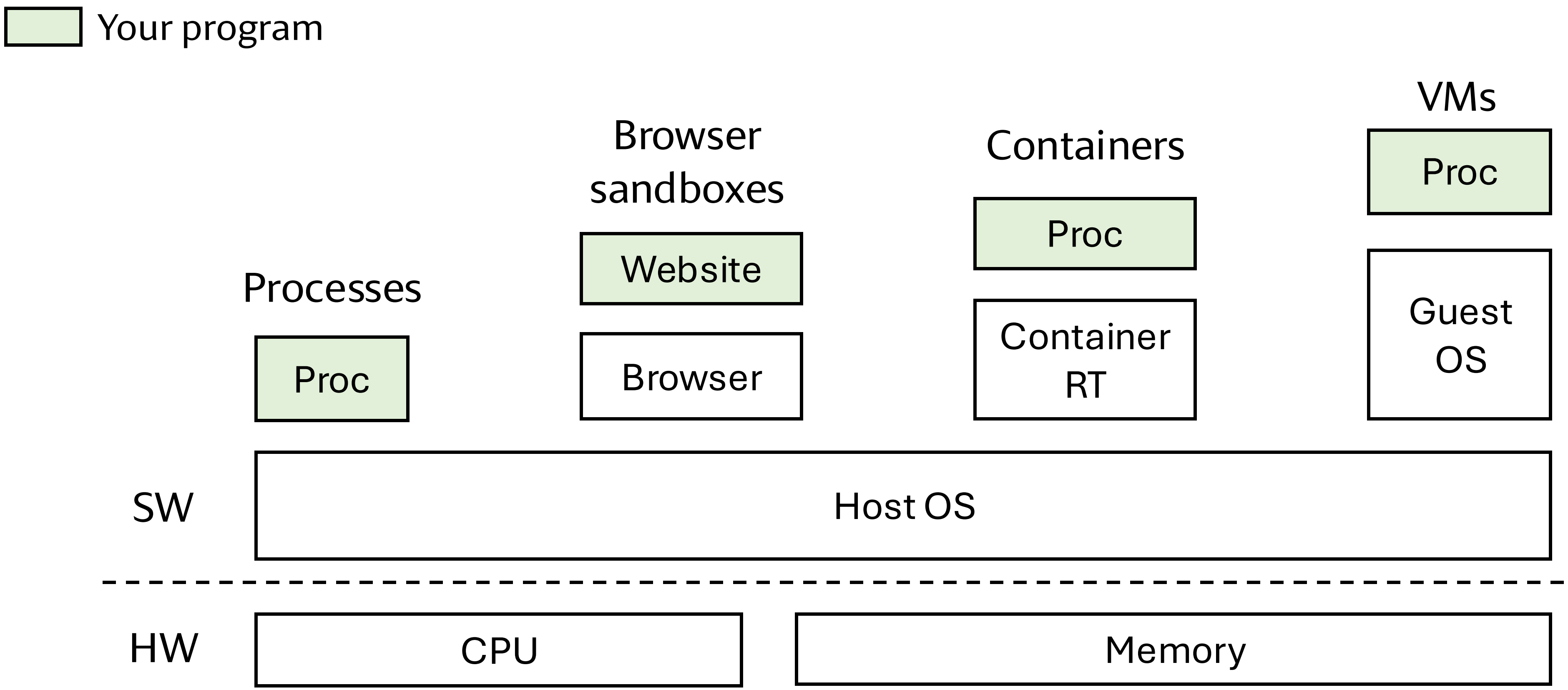
ECE 382N-Sec (FA25):

L9: OS and VM Isolation

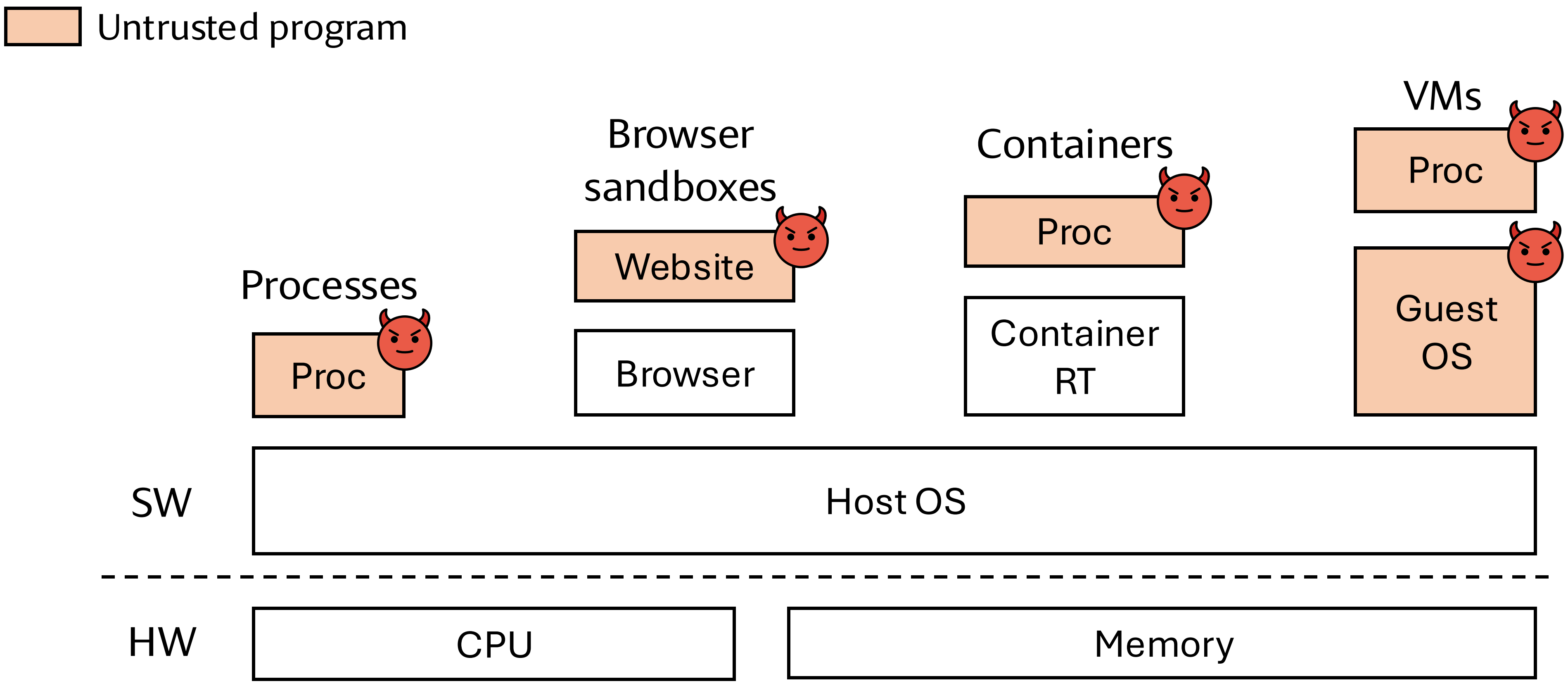
Neil Zhao

neil.zhao@utexas.edu

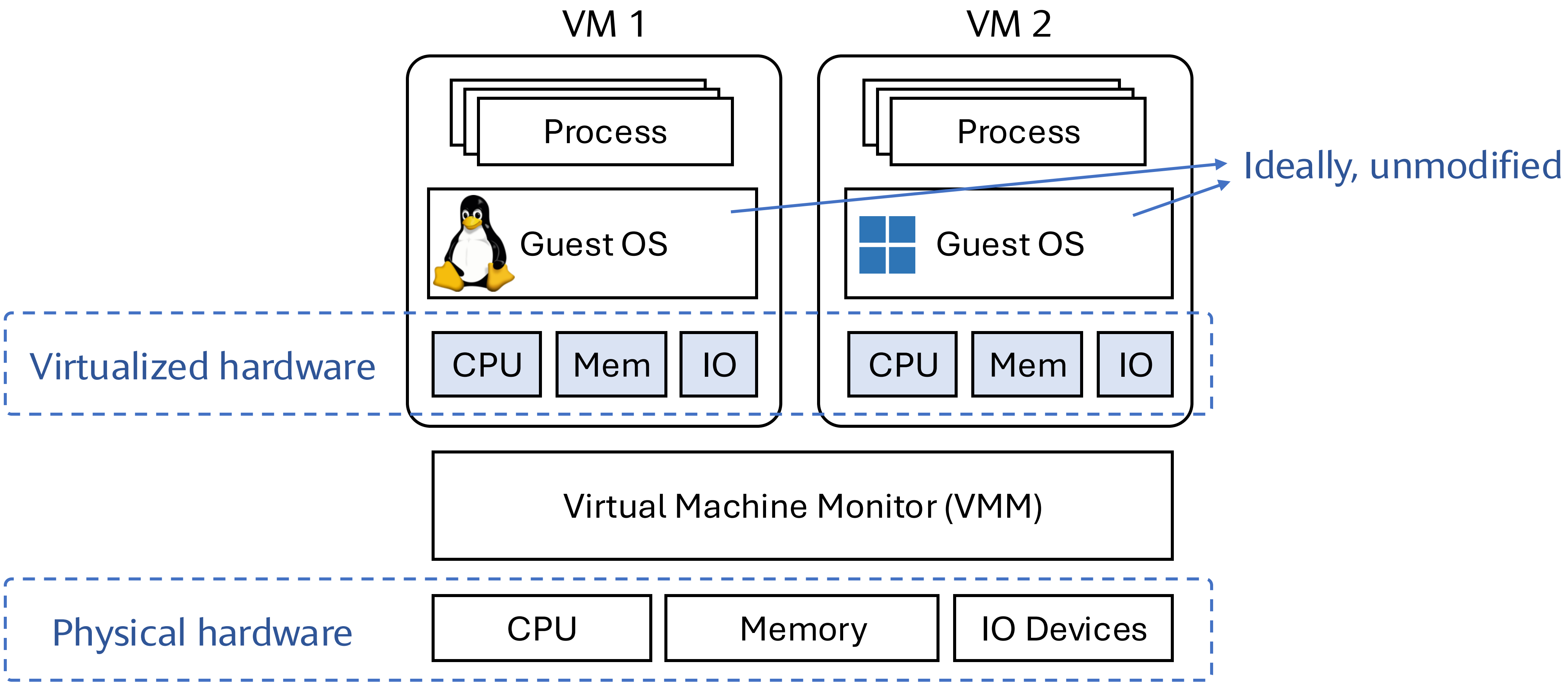
Different Isolation Techniques



Different Isolation Techniques



Virtual Machine

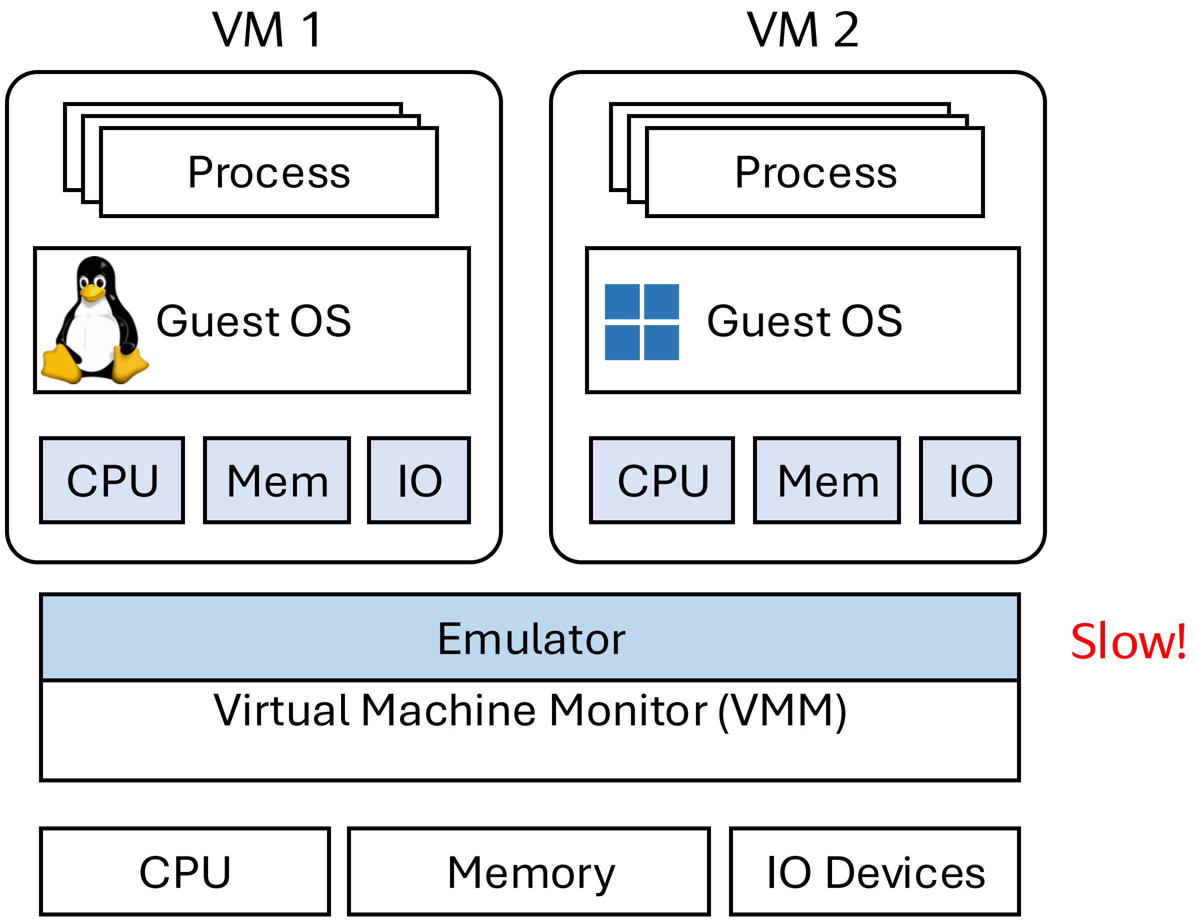


Virtual Machine Principles

Popek and Goldberg's virtualization principles in 1974:

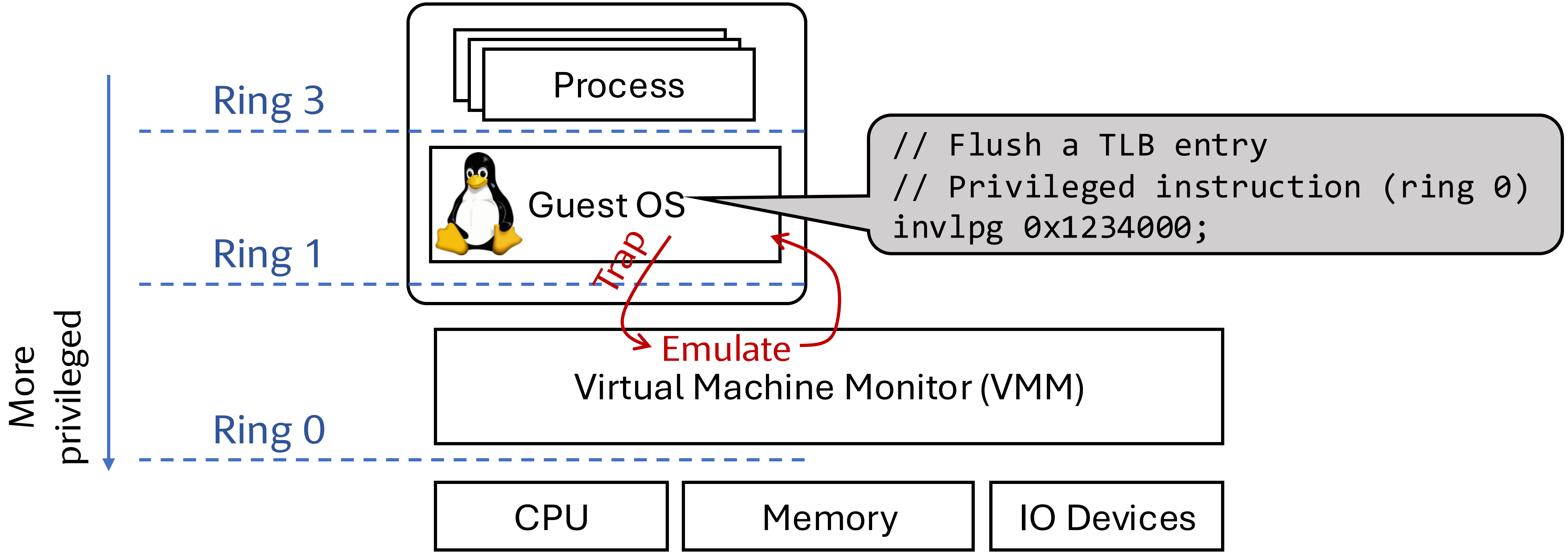
- **Fidelity:** Software on the VMM executes identically to its execution on hardware, barring timing effects
- **Performance:** An overwhelming majority of guest instructions are executed by the hardware without the intervention of the VMM
- **Safety:** The VMM manages all hardware resources

Virtualizing CPU - Emulation



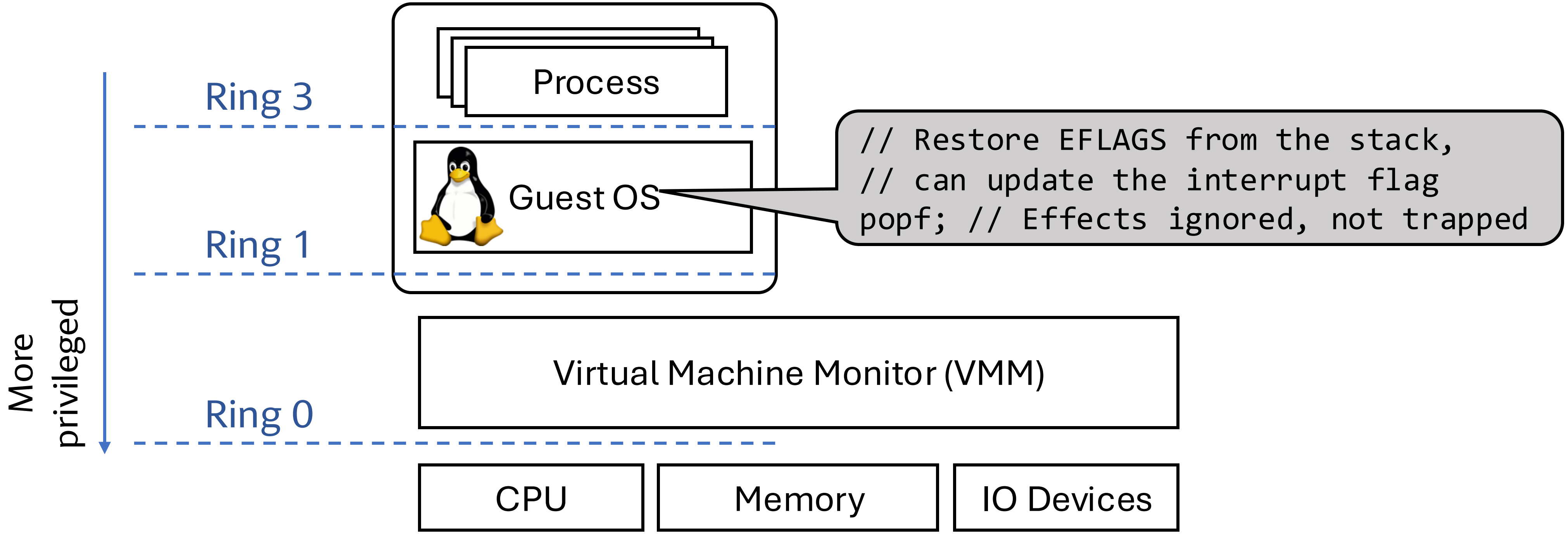
Idea: Execute VM Instructions Natively on Physical CPUs

Assuming the VM uses the same architecture as the host



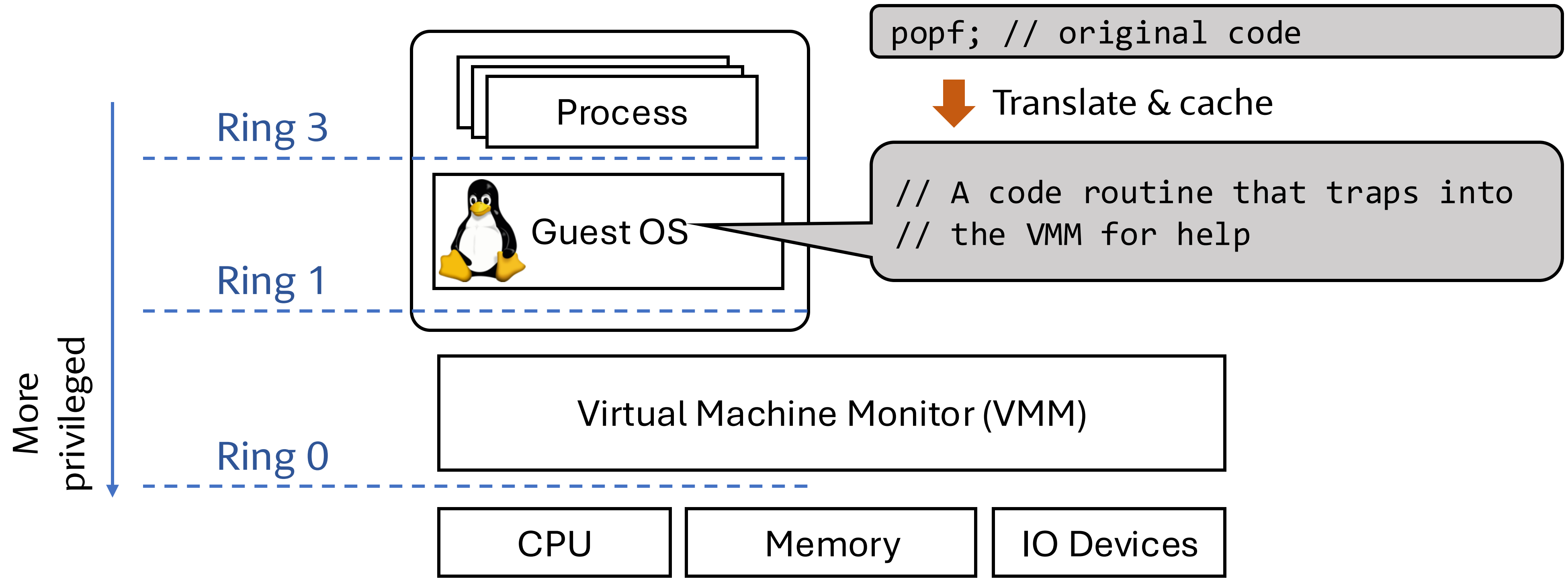
Trap-and-Emulate

Catch: The guest can execute *sensitive but not privileged* instructions without being trapped



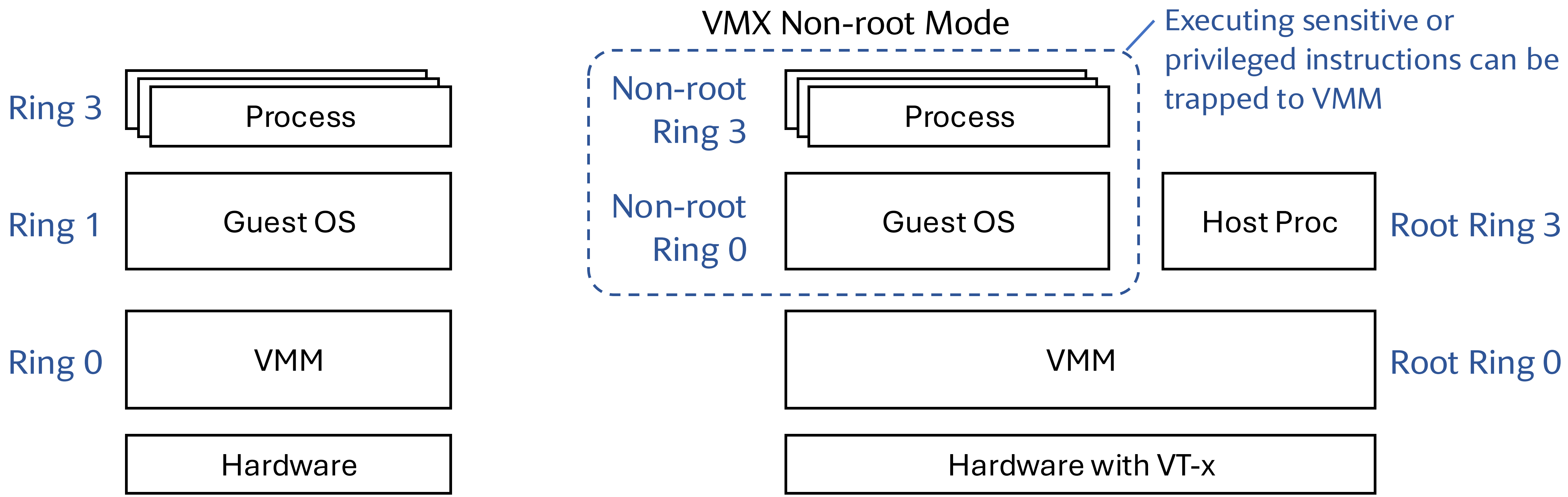
Solution: Dynamic Binary Translation

Idea: Replace non-virtualizable instructions with code sequences for trapping execution into the VMM (first implemented by VMWare)



Hardware Support for Virtualization

Virtual Machine Extensions (e.g., Intel VT-x)

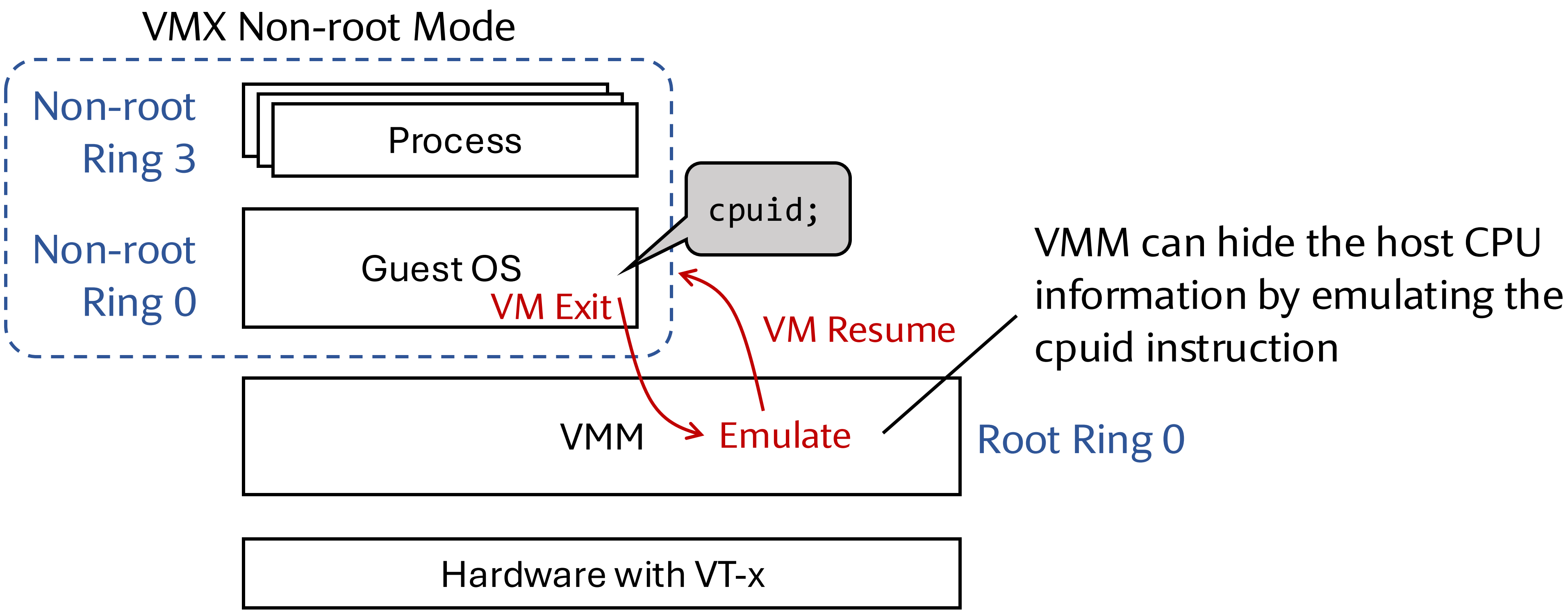


Orthogonal to x86 protection rings

$\Rightarrow$ {Root, Non-root} $\times$ {Ring 0, Ring 1, Ring 2, Ring 3}

Hardware Support for Virtualization

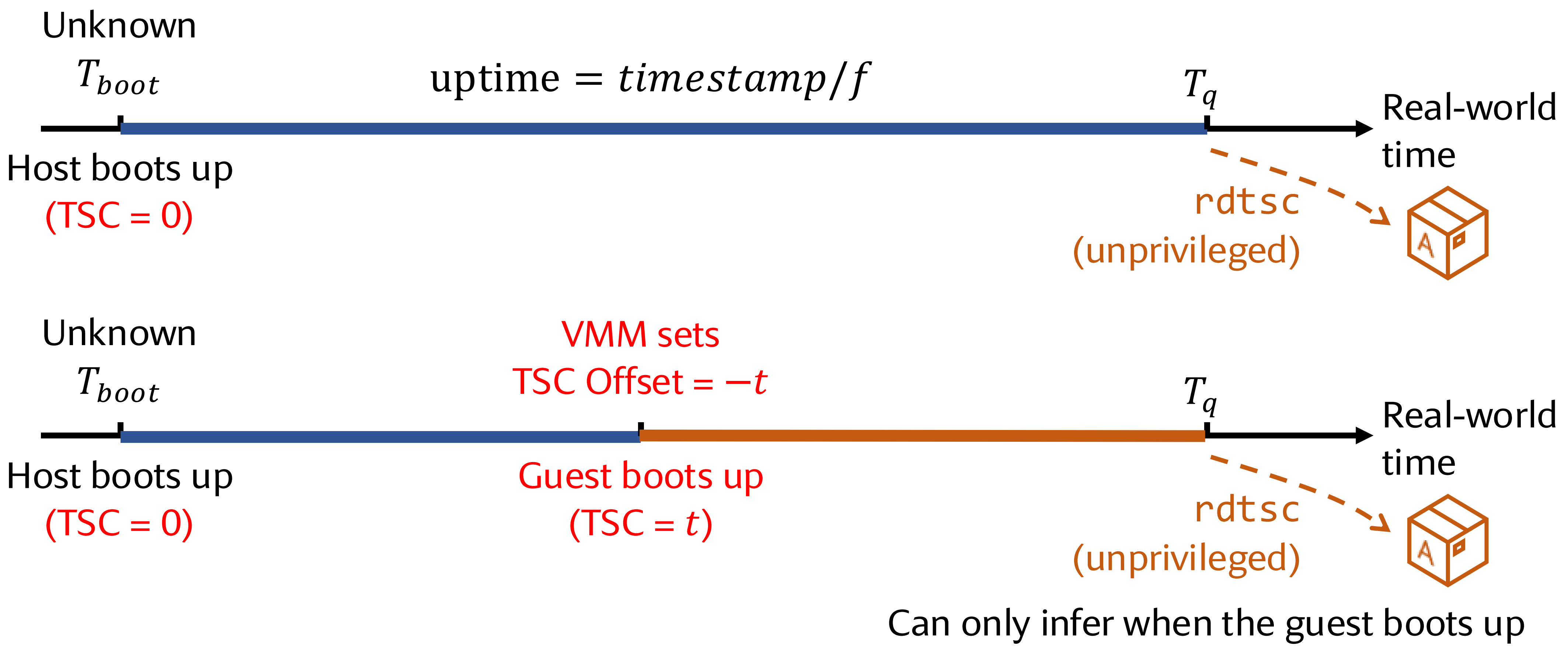
Virtual Machine Extensions (e.g., Intel VT-x)



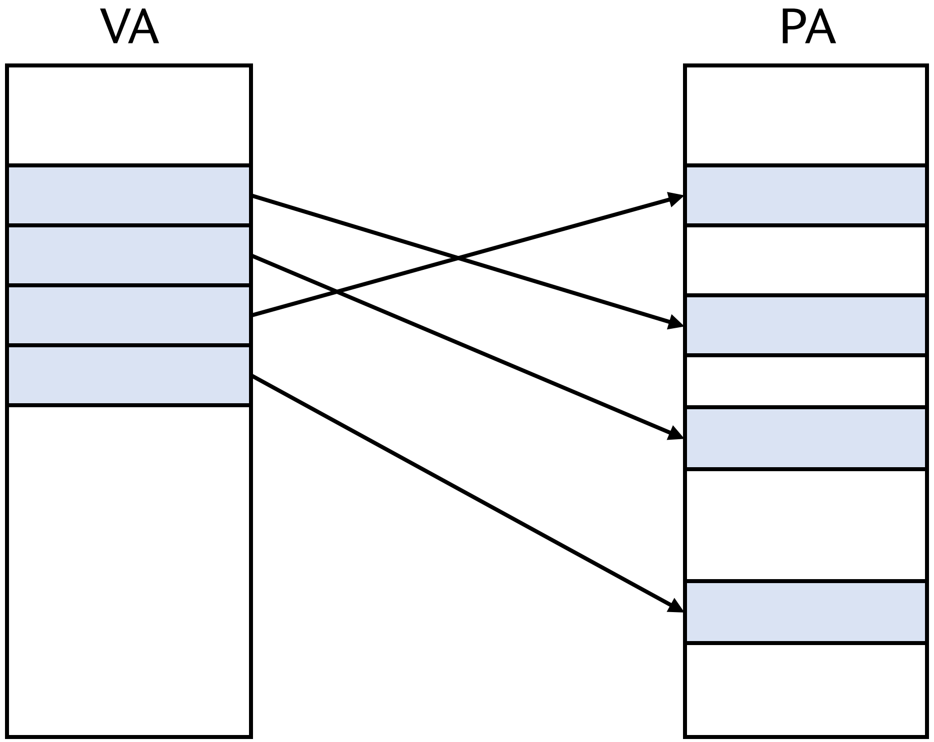
VMM can control which instruction to intercept (only a subset of x64 instructions are supported)

Hardware Support for Virtualization

TSC Offsetting: Guest TSC = Host TSC + VM-Specific Offset

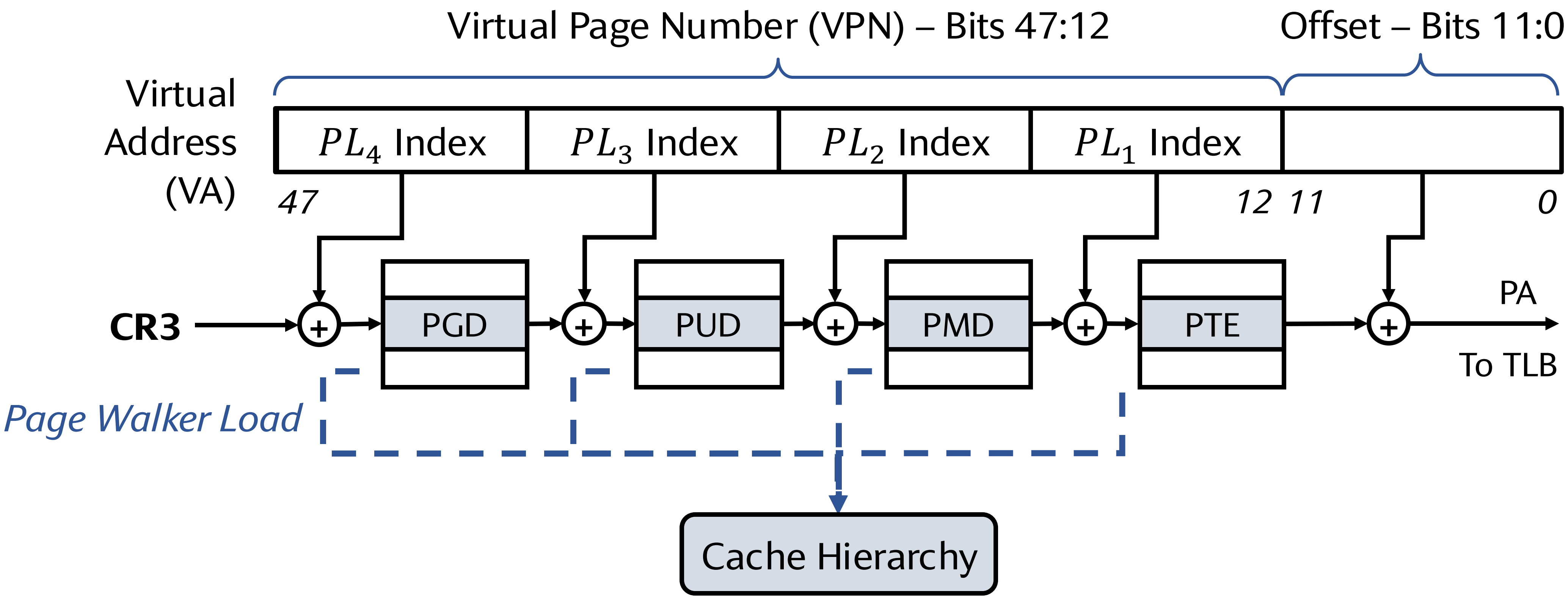


Virtualizing Guest Memory



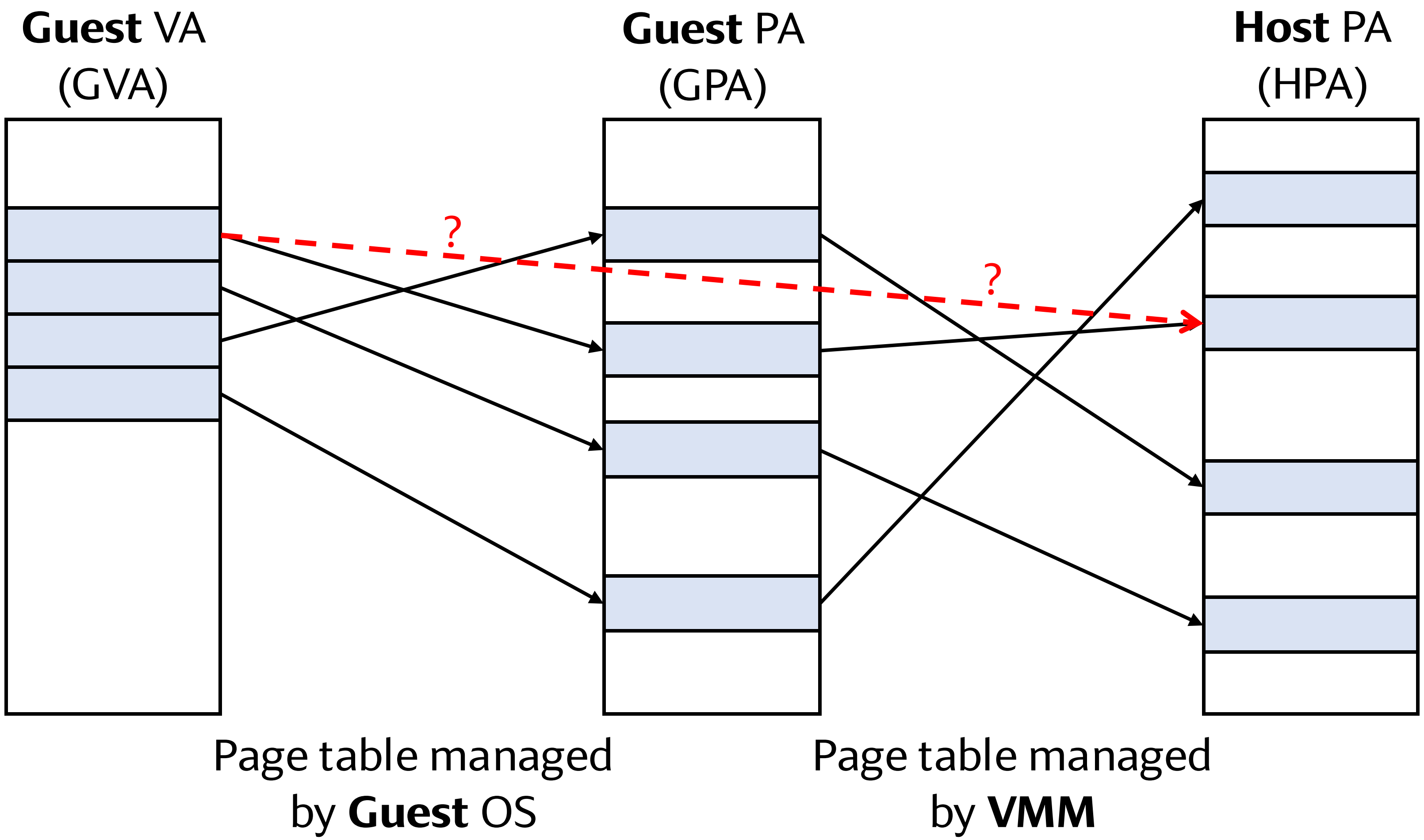
Page table
managed by OS

Page Walk

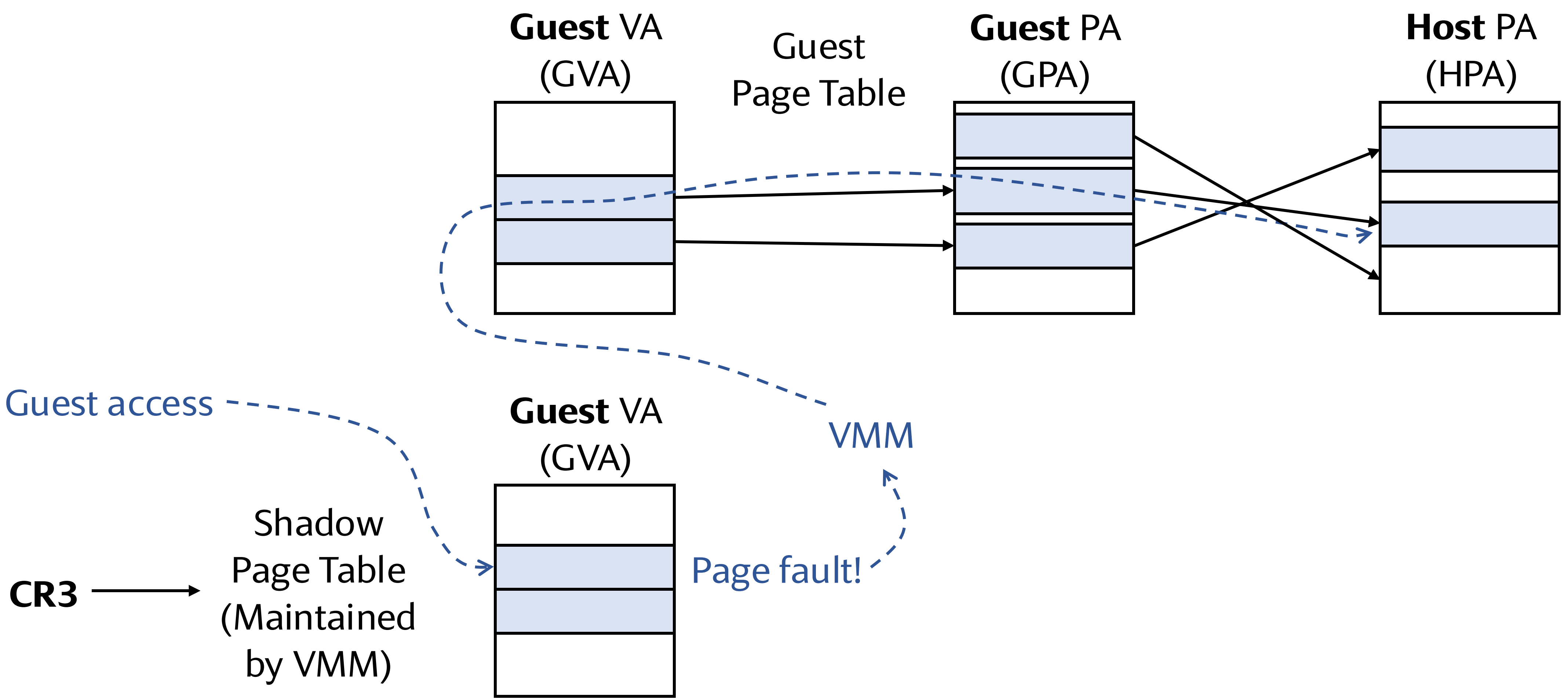


Virtual Page Number (VPN) – Bits 47:12
Offset – Bits 11:0
Virtual Address (VA)
PL_4 Index
PL_3 Index
PL_2 Index
PL_1 Index
47
12
11
0
CR3
PGD
PUD
PMD
PTE
PA
To TLB
Page Walker Load
Cache Hierarchy

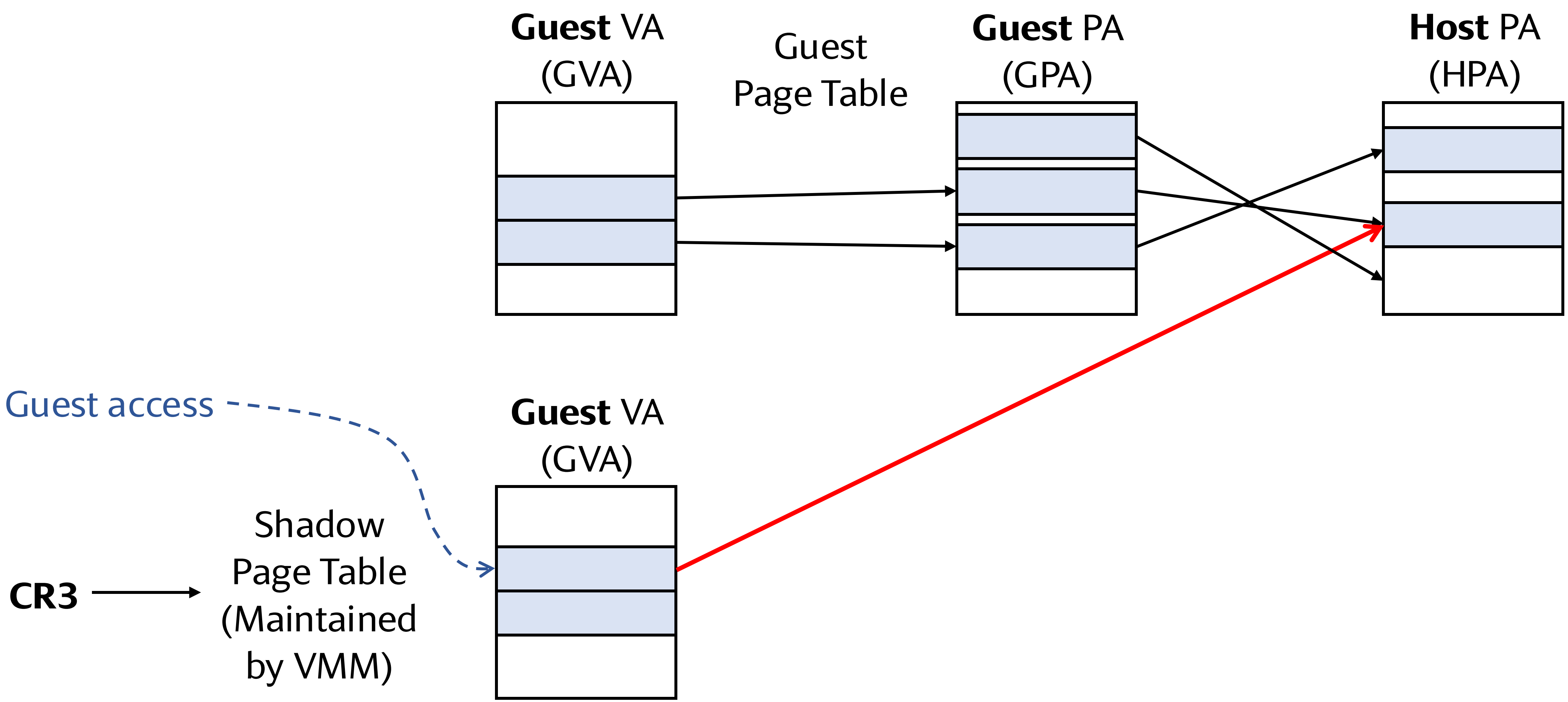
Virtualizing Guest Memory



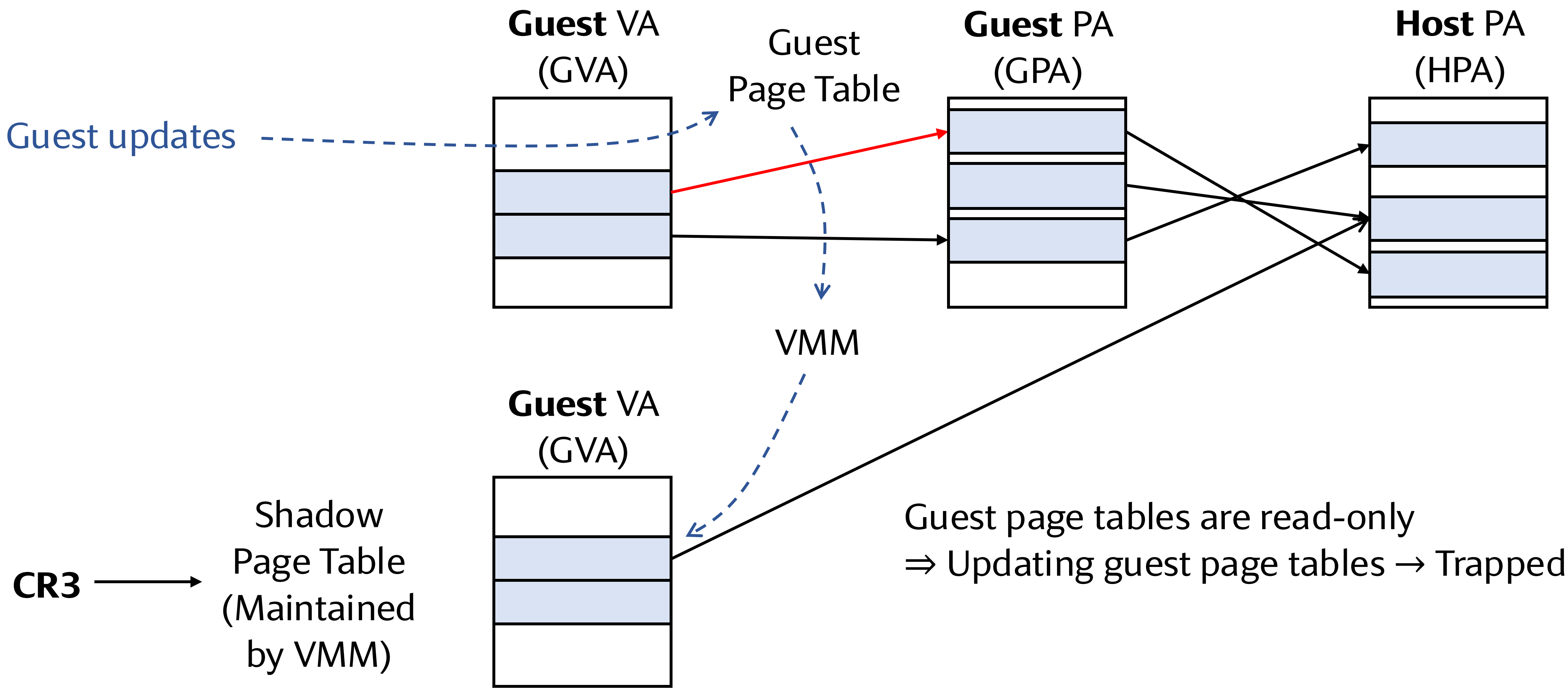
Shadow Page Table



Shadow Page Table

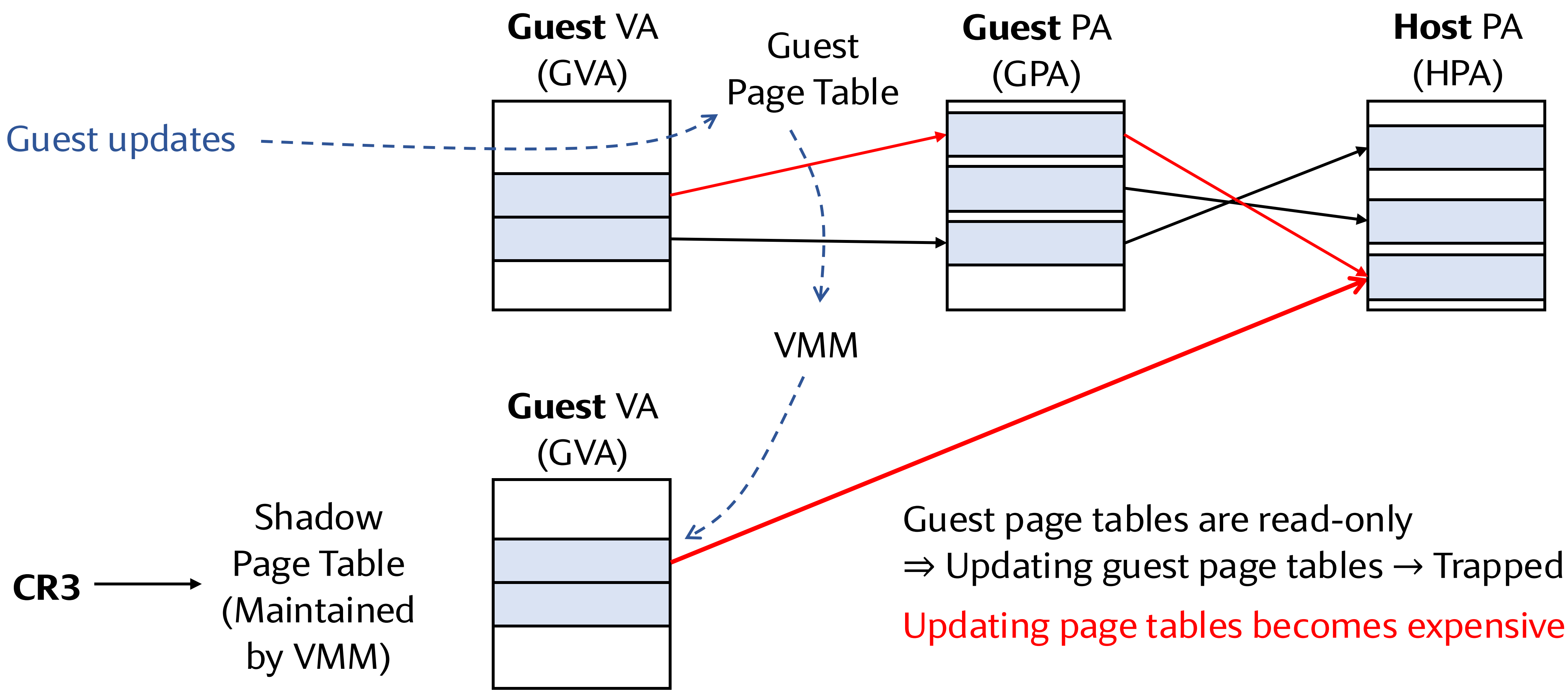


Shadow Page Table



Guest page tables are read-only
⇒ Updating guest page tables → Trapped

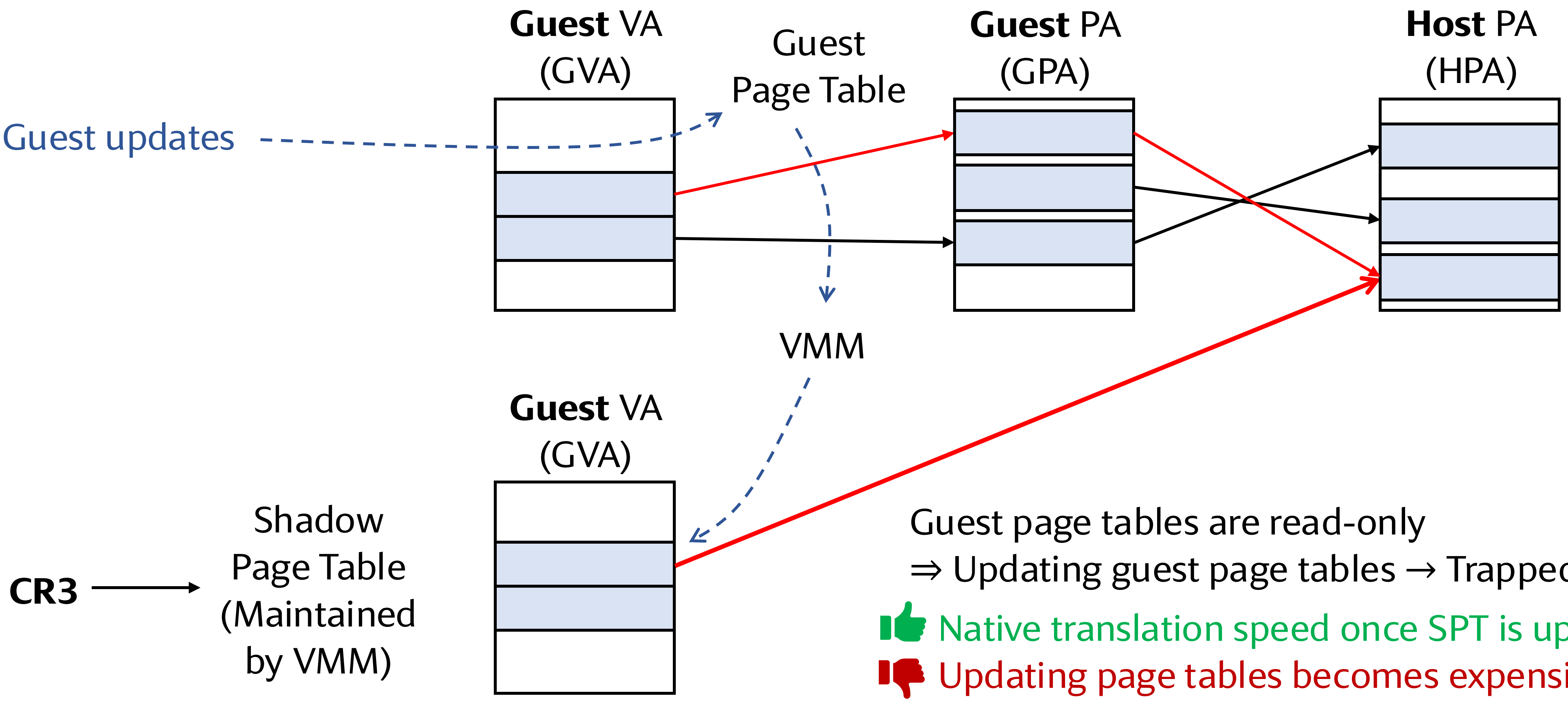
Shadow Page Table



Guest page tables are read-only
⇒ Updating guest page tables → Trapped

Updating page tables becomes expensive!

Shadow Page Table



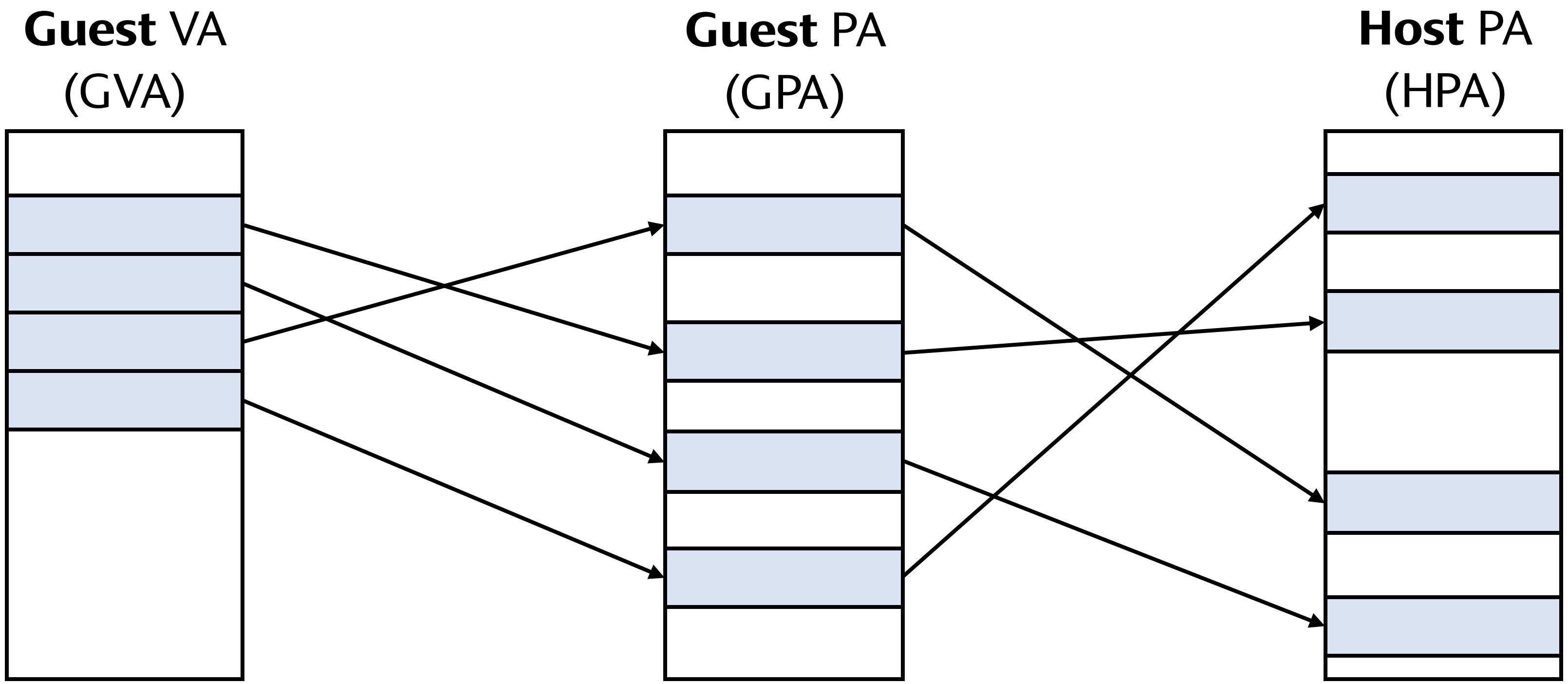
Guest page tables are read-only
⇒ Updating guest page tables → Trapped

👍 Native translation speed once SPT is up

👎 Updating page tables becomes expensive

Hardware Support for Virtualizing Guest Memory

Extended Page Table (Intel) or Nested Page Table (AMD)

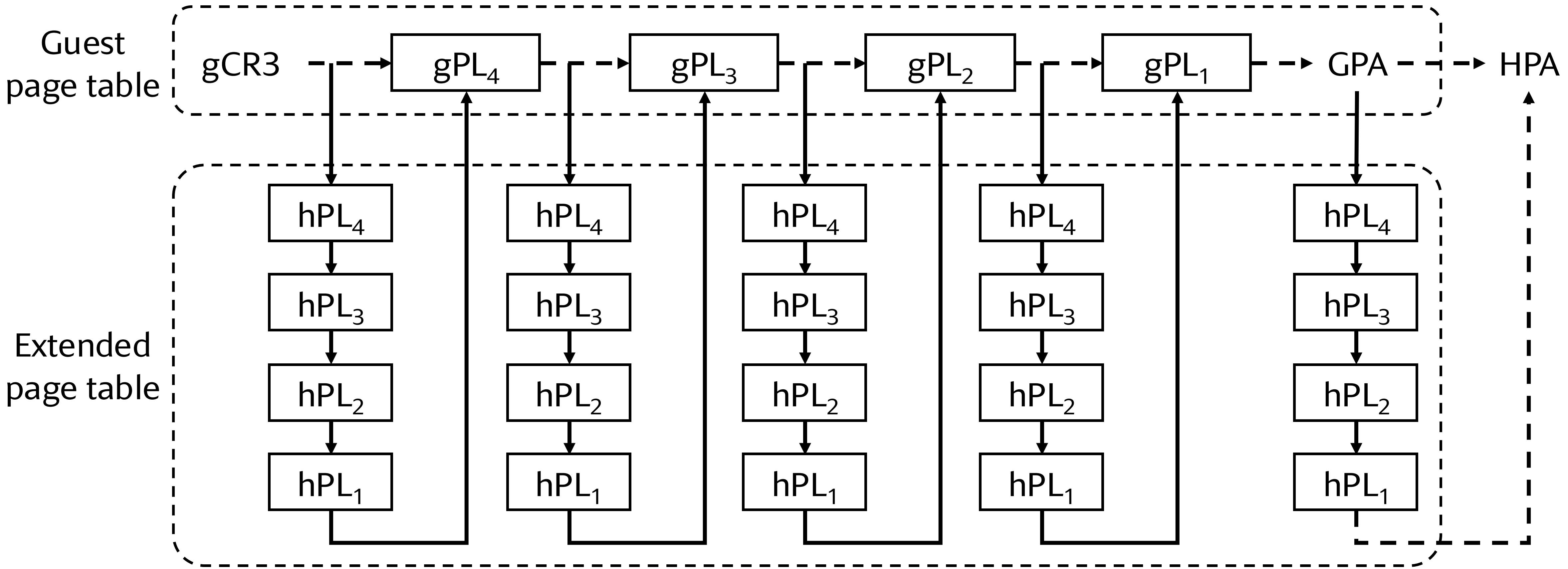


Page table managed by **Guest** OS

Page table managed by **VMM**

Both exposed to the hardware page walker

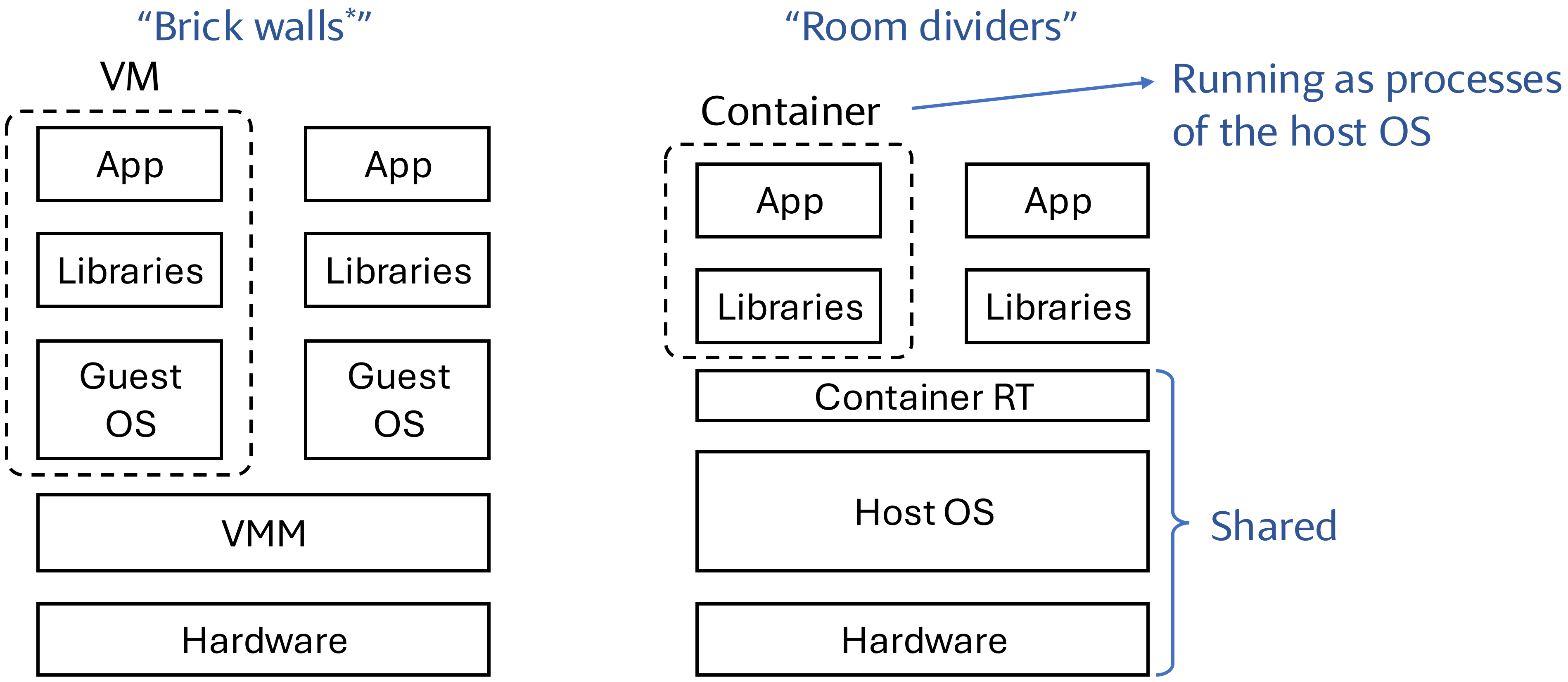
Hardware Support for Virtualizing Guest Memory



24 serialized memory accesses in the worst case (instead of 4 in the native execution)

Containers

OS-level virtualization: Small resource footprint, poor isolation



*Analogy from Docker's Jérôme Petazzoni

Requirements of Containers

- **Visibility restrictions:** Containers should **not** have unrestricted view of or access to host resources. Each one has
  - Its own root directory
  - Virtualized process IDs
  - Virtualized network interfaces
  - …
- **Resource restrictions:** Containers should **not** exhaust host resources. E.g., it **cannot**
  - Launch forkbomb
  - Exhaust host memory
  - Monopolize host CPU
  - …
- **Interface restrictions:** Container should **not** have access to all the system calls

Visibility Violation Example - /proc Filesystem

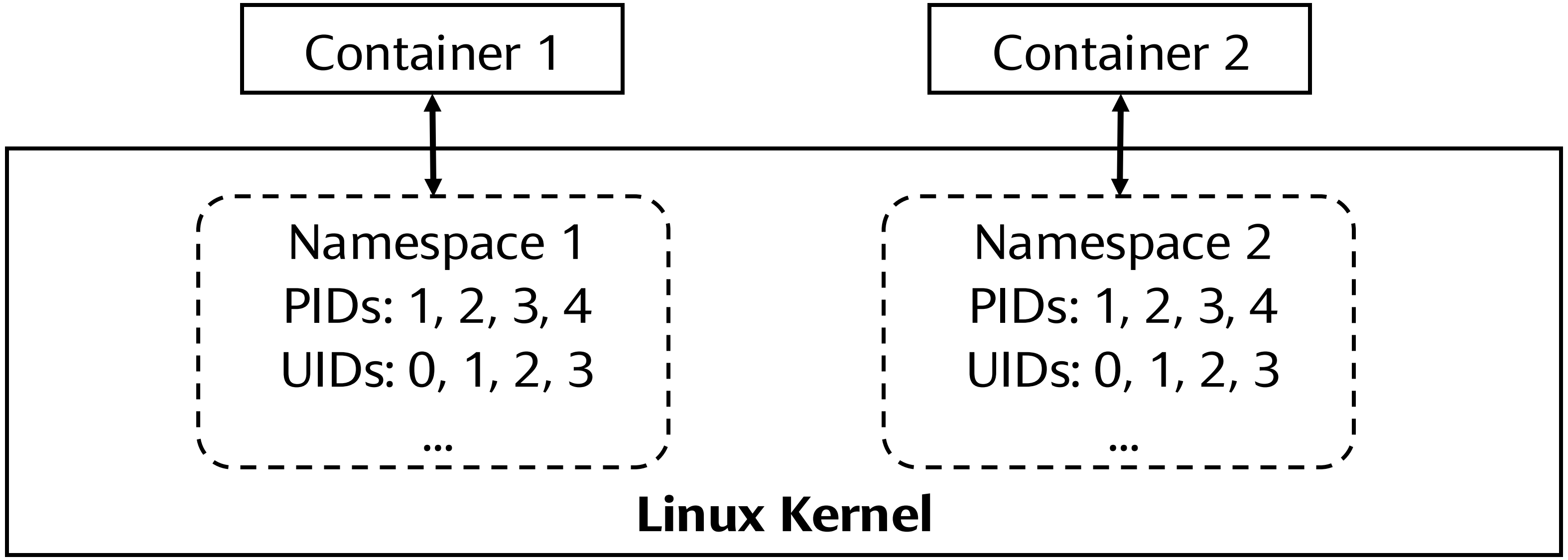
- A special filesystem presenting information about processes and the system
- E.g., /proc/interrupts ⇒ Detect host co-location

```
           CPU0       CPU1       CPU2       CPU3
NMI:        868       1180       1017        855   Non-maskable interrupts
LOC:   47381509   57330040   56830366   69039703   Local timer interrupts
SPU:          0          0          0          0   Spurious interrupts
PMI:        868       1180       1017        855   Performance monitoring interrupts
IWI:         11          4          1         20   IRQ work interrupts
RTR:          0          0          0          0   APIC ICR read retries
RES:     339907     346897     348481     251065   Rescheduling interrupts
CAL:    6736377    7967333    7302809    4965812   Function call interrupts
TLB:    3080825    3186638    2895960    3240937   TLB shootdowns
```

Restricting Visibility - Namespace

Namespaces partition provides each container its own isolated view of the OS:

- PIDs
- Mount
- Network
- User



Restricting Resources – Control Groups (cgroups)

Limit, account, and isolate resource usage of a **group** of processes

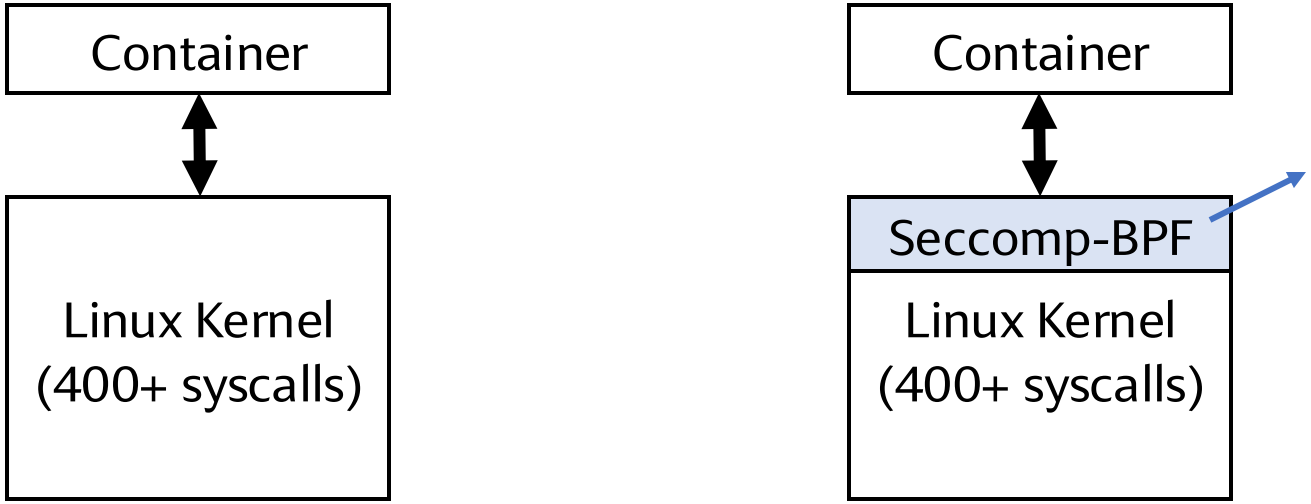
- CPU time
- Memory
- I/O
- PIDs
- …

Example: Restricting container's CPU "bandwidth" to 0.5 core

- For every 100ms, the container receives 50ms of CPU time of a single core
- The CPU time is accumulated across threads in the container
  - Run 1 thread for 50ms on 1 CPU core, or
  - Run 2 threads for 25ms each on 2 CPU cores

Restricting Interface – Seccomp-BPF

Seccomp or “SECure COMPuting” + Berkeley Packet Filter (BPF)



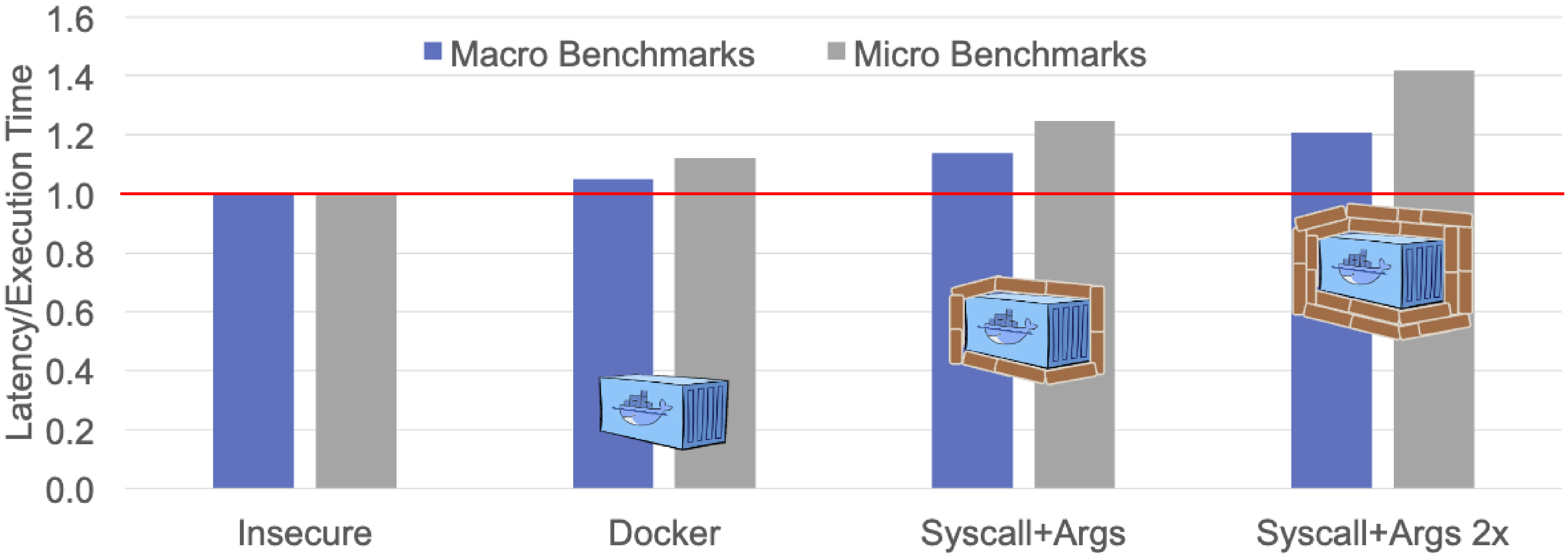
Large attack surface through syscalls (2k+ CVEs since 1999)

Reduced attack surface

Allow or deny syscalls based on syscall ID and their arguments

Inherited by child processes and can only become stricter

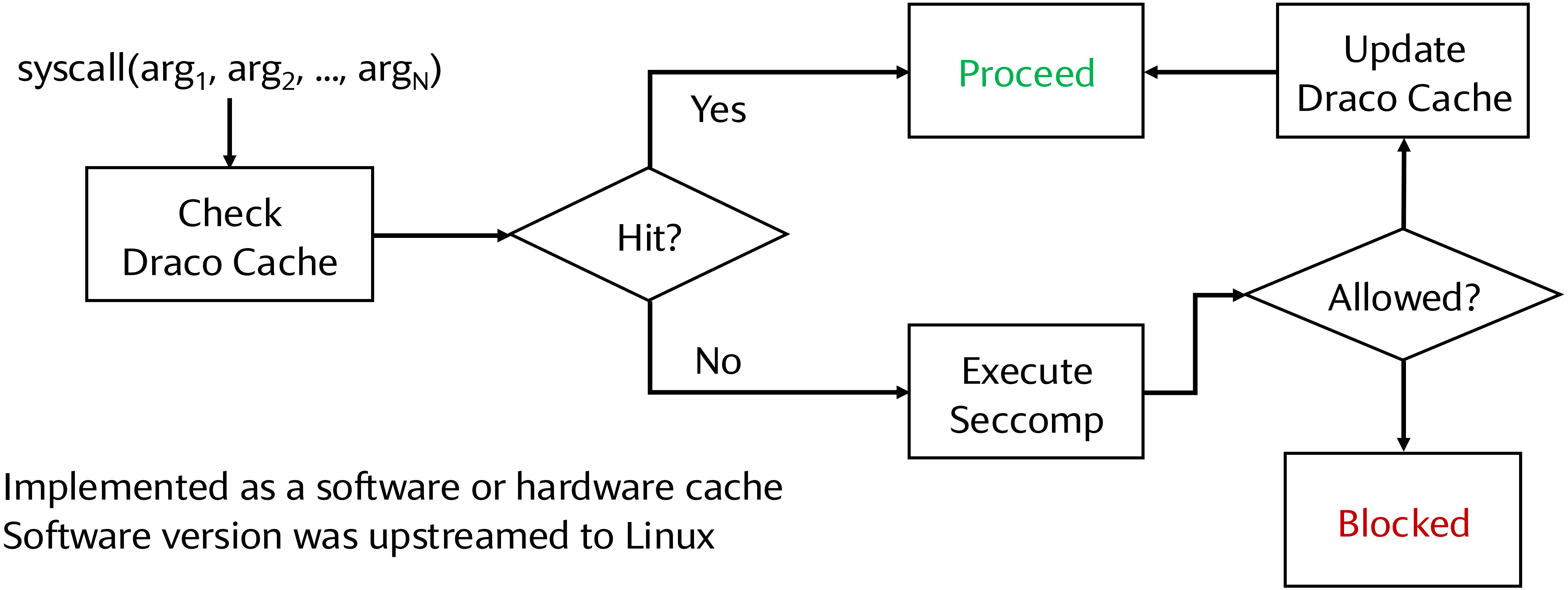
Performance Overhead of Seccomp-BPF



*Skarlatos et al, "Draco: Architectural and Operating System Support for System Call Security," MICRO '20

Draco*: Caching Seccomp Results

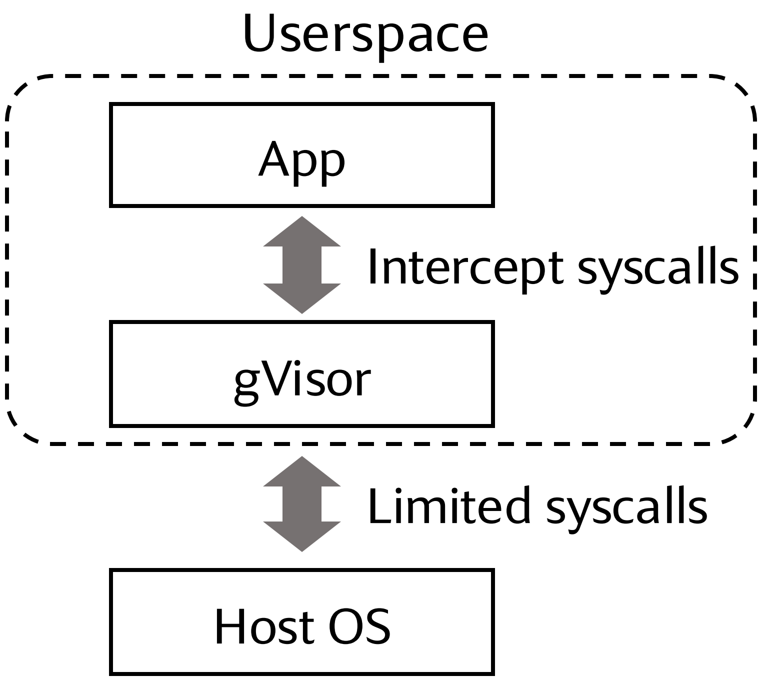
Seccomp is stateless



Implemented as a software or hardware cache
Software version was upstreamed to Linux

*Skarlatos et al, "Draco: Architectural and Operating System Support for System Call Security," MICRO '20

gVisor



gVisor Intercepts and emulates syscalls in user space (similar to a userspace kernel)
Developed in Golang for memory safety

gVisor Hides Sensitive Host Information

```
Attacker: CPU Model? (i.e., lscpu)
gVisor: unknown

Attacker: Boot log? (i.e., dmesg)
gVisor:
  Starting gVisor...
  Granting licence to kill(2)...
  Recruiting cron-ies...
  Creating process schedule...
  Checking naughty and nice process list...
  Gathering forks...
  Rewriting operating system in Javascript...
  ...
```

Lightweight, small resource footprint

Compatibility issue, syscall performance issue