Advanced Topics in Operating Systems

Dr. Oren Laadan

Lecture 9

Course #3471

Term 1, 2022

Review

- Virtualization: what and how
- Virtualization taxonomy
- Virtualization violations
- Virtualization methods
- Virtualization case studies: VMWare, Xen

VMware vs Xen

	VMWare	Xen

- VMM type?
- Memory mapping?
- World switch?
- Device I/O?

VMware vs Xen

	VMWare	Xen
• VMM type?	Type I / II	Type I
• Memory mapping?	alternate	resident
• World switch?	yes	no
• Device I/O?	single thread	multi-thread

Virtualization

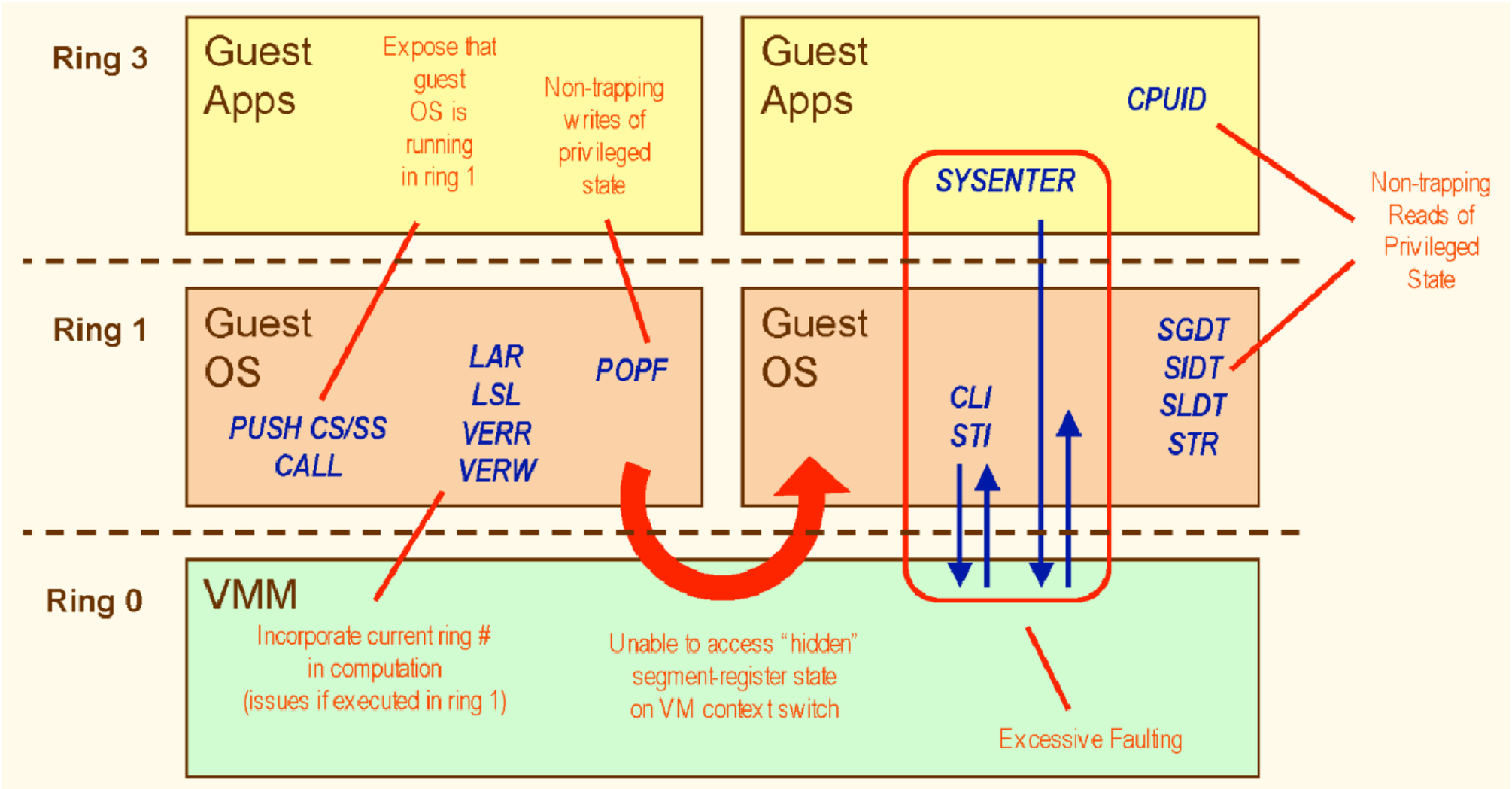
- Motivation, applications, ane usage models
- VMM theory and architecture
- Virtualization types and techniques
- Hardware virtualization
- Software virtualization

IA-32 Virtualization Support

- Challenges
  - Ring aliasing, ring compression
  - Address space compression
  - Non-faulting access to privileged state
  - Unnecessary impact of guest transitions
  - Interrupt virtualization
  - Access to hidden state

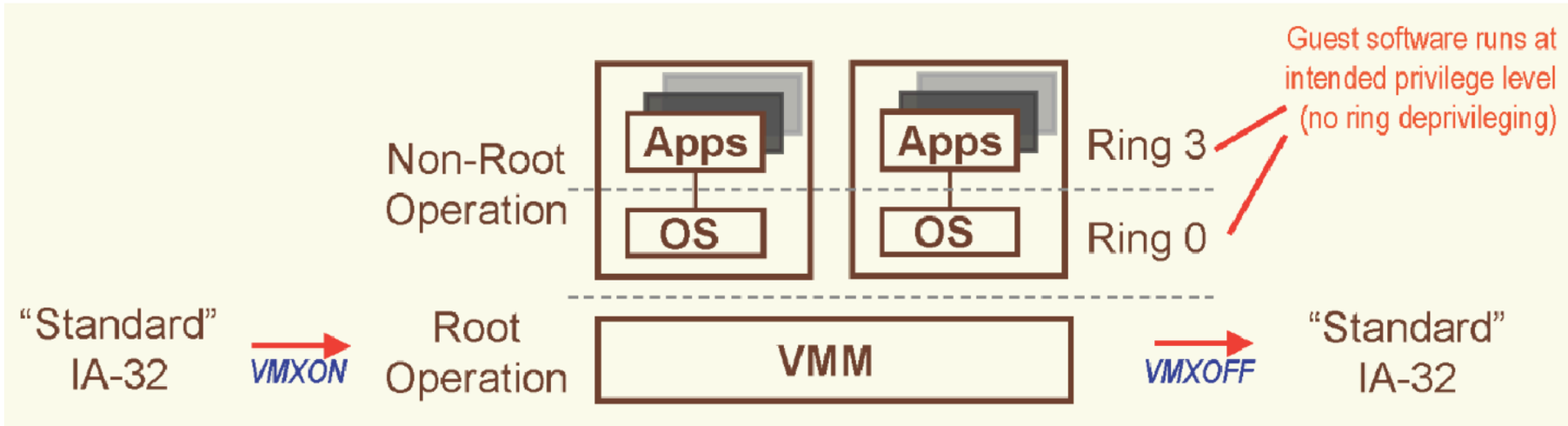
IA-32 CPU Virtualization

- Virtualization holes



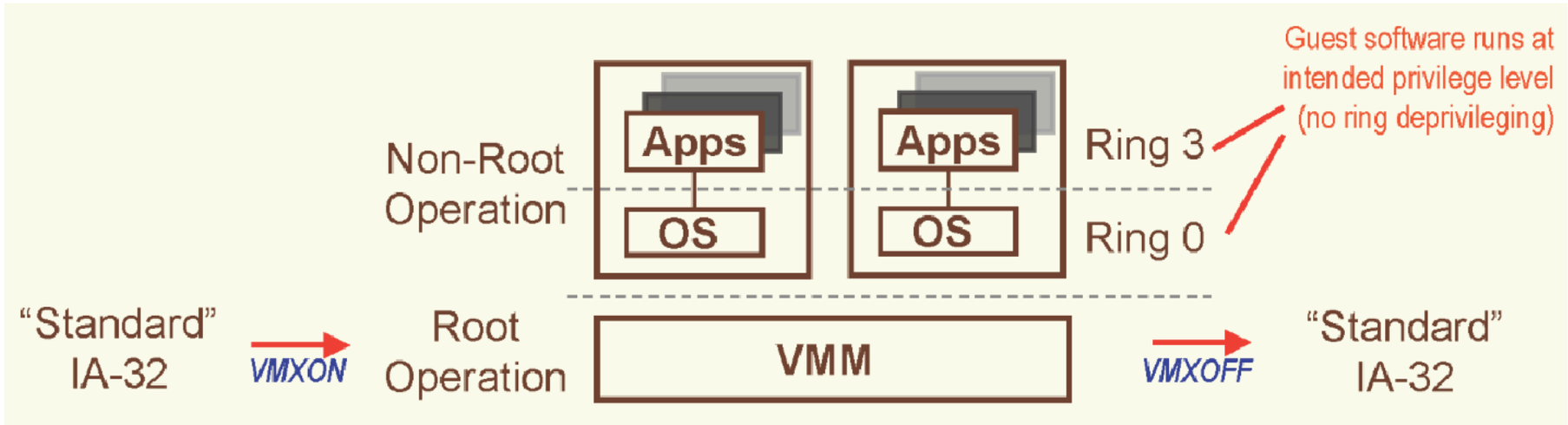
IA-32 Virtualization: VT-x

- New mode of execution:
  - Root operation: fully privileged, intended for VMM
  - Non-root operation: intended for guest OS
  - Enabled/disable with new instruction VMXON/VMXOFF



IA-32 Virtualization: VT-x

- New mode of execution:
  - Root operation: fully privileged, intended for VMM
  - Non-root operation: intended for guest OS
  - Enabled/disable with new instruction VMXON/VMXOFF
  - VM Entry: VMM to guest transition (using VMLAUNCH, VMRESUME)
  - VM Exit: guest to VMM transition (caused by virtualization events)



VT-x: VMCS

- VM Control Structure: hold guest state
  - Accessed using VMREAD, VMWRITE
  - Switched using VMPTRLD
- Controls behaviour of the guest (e.g. select VM-Exit events)
  - Privileged state, sensitive ops, paging events, interrupts/exceptions

VT-x: VMCS

- VM Control Structure: hold guest state
  - Accessed using VMREAD, VMWRITE
  - Switched using VMPTRLD
- Controls behaviour of the guest (e.g. select VM-Exit events)
  - Privileged state, sensitive ops, paging events, interrupts/exceptions
  - One VMCS per virtual CPU, one VMCS per active CPU

VM execution controls	Determines operations that cause VM exit	CR0, CR3, CR4, Exceptions, I/O ports, Interrupts, etc
Guest state area	Saved on VM exits, reloaded on VM entries	EIP, ESP, EFLAGS, IDTR, Segments regs, Exit info, etc
Host state area	Loaded in VM exits	CR3, EIP to to monitor entry
VM exit controls	Determines which state to save, load, how to transition	E.g. MSR Save/Load list
VM entry controls	Determines which state to save, load, how to transition	Including injection of events on VM Entrentry

VT-x: Optimizations

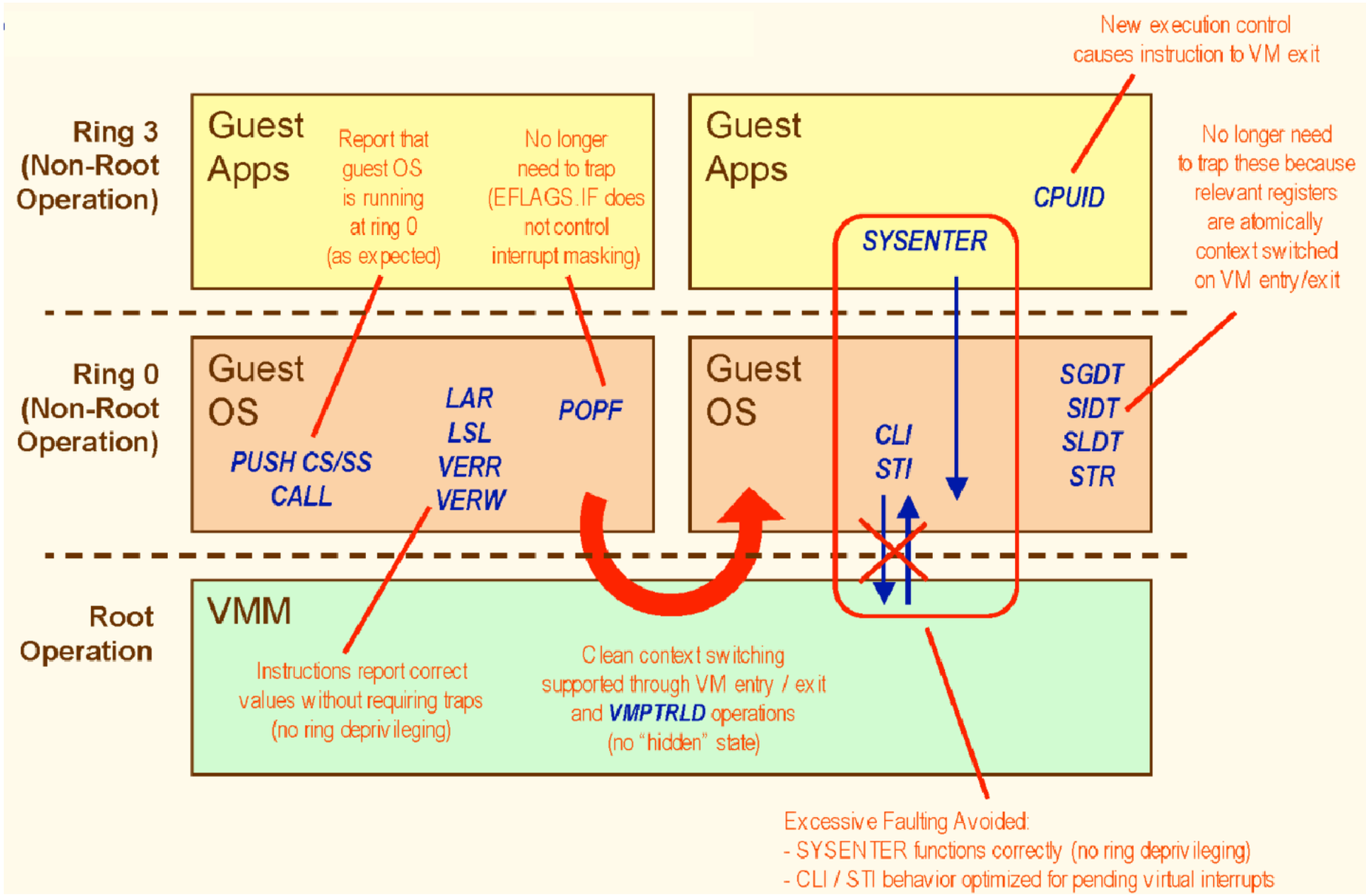
VT-x: Optimizations

- Shadow registers and masks
  - Read from CR0, CR4 are satisfied from shadow registers
  - VM exit can be conditional based on a bitmap
- Execution-control bitmaps
  - VM exits selectively controlled via bitmaps (exceptions, IO port, etc)
- External interrupt exiting
  - Interrupts (in guest) never really masked
- Interrupt delivery
  - Send pending virtual-interrupt to guest
  - Delivered only when guest OS interrupt is enabled
- Timestamp counters offsets

VT-x: Optimizations

- Event injection
  - Occurs after guest state is fully loaded
  - Remove burden from VMM to emulate everything

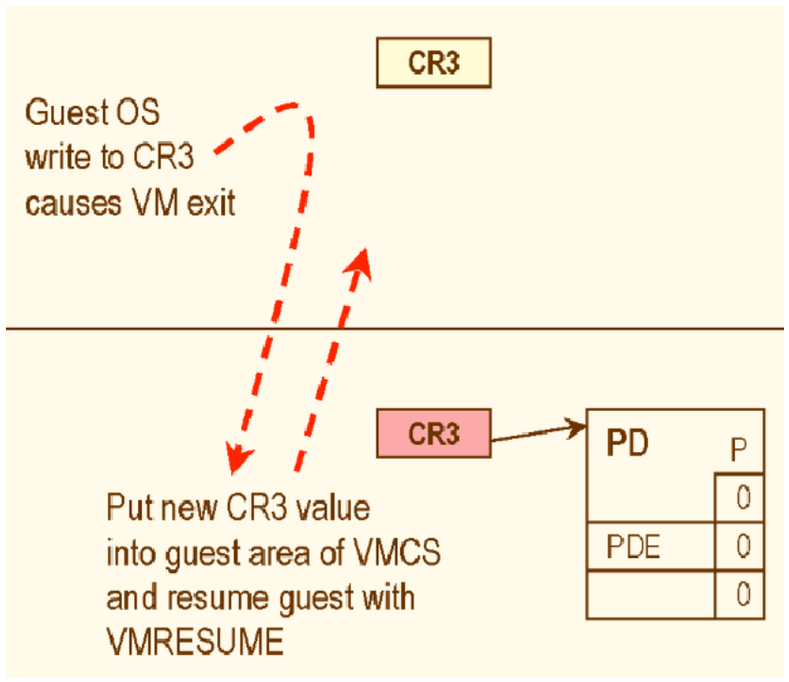
VT-x: CPU Virtualization



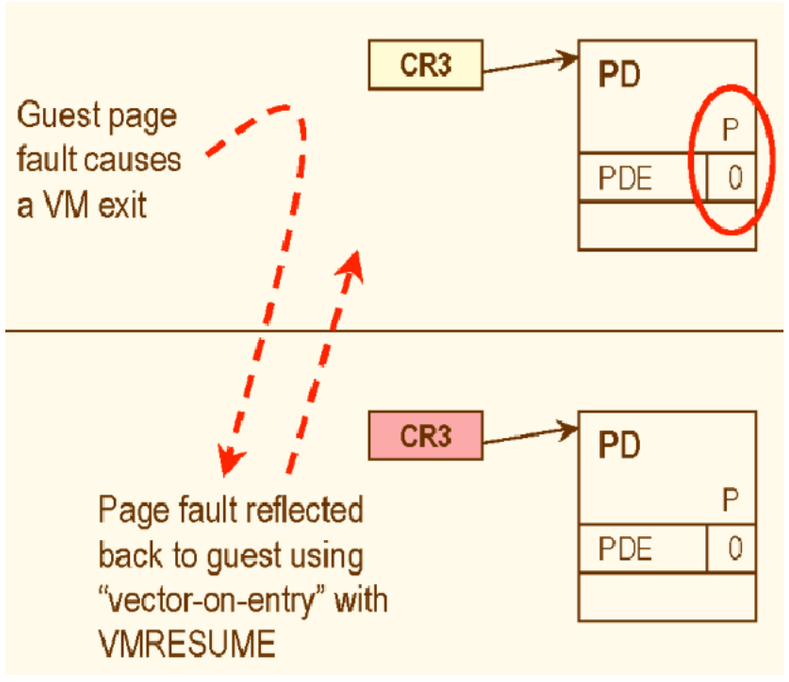
VT-x: Shadow Page Tables

- Shadow page tables
  - CR3 write implies a page-table change and TLB flush
  - Page fault requires VMM to examine guest page-tables

Action on CR3 write



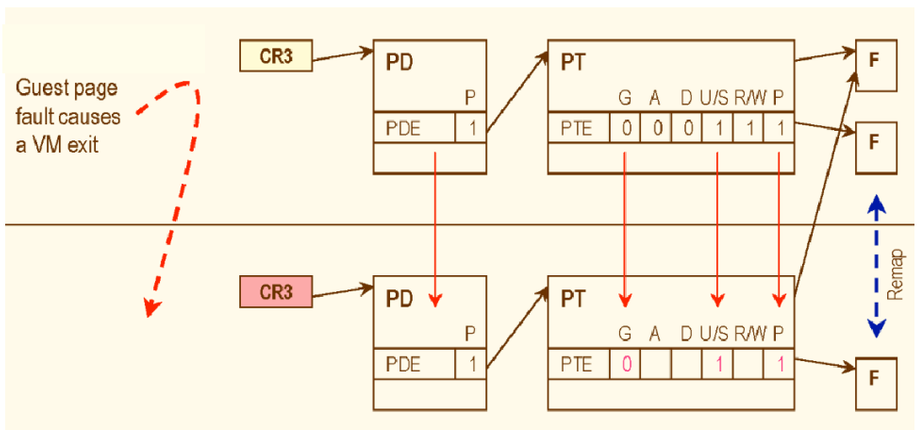
Action on page fault



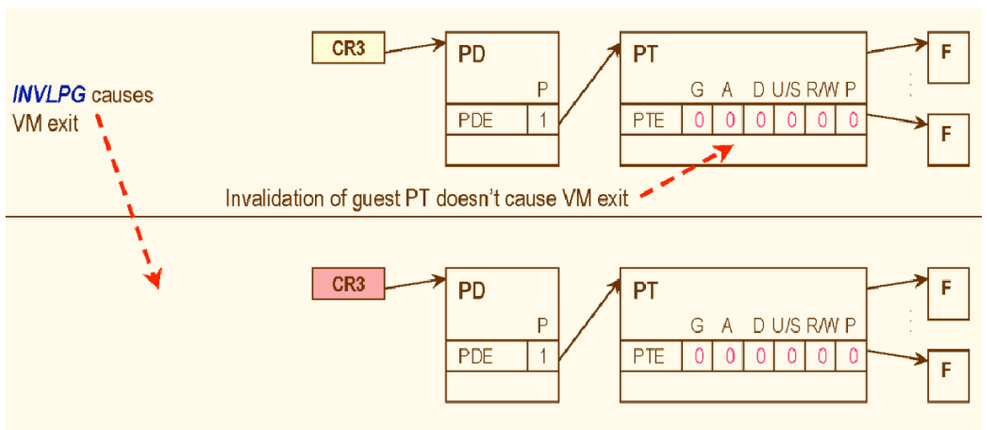
VT-x: Shadow Page Tables

- Shadow page tables (cont)
  - Page fault leads VMM to allocate new page table (entries)
  - INVLPG implies TLB flush
  - Guest permitted to freely modify its page tables
  - Real and shadow pages tables may differ

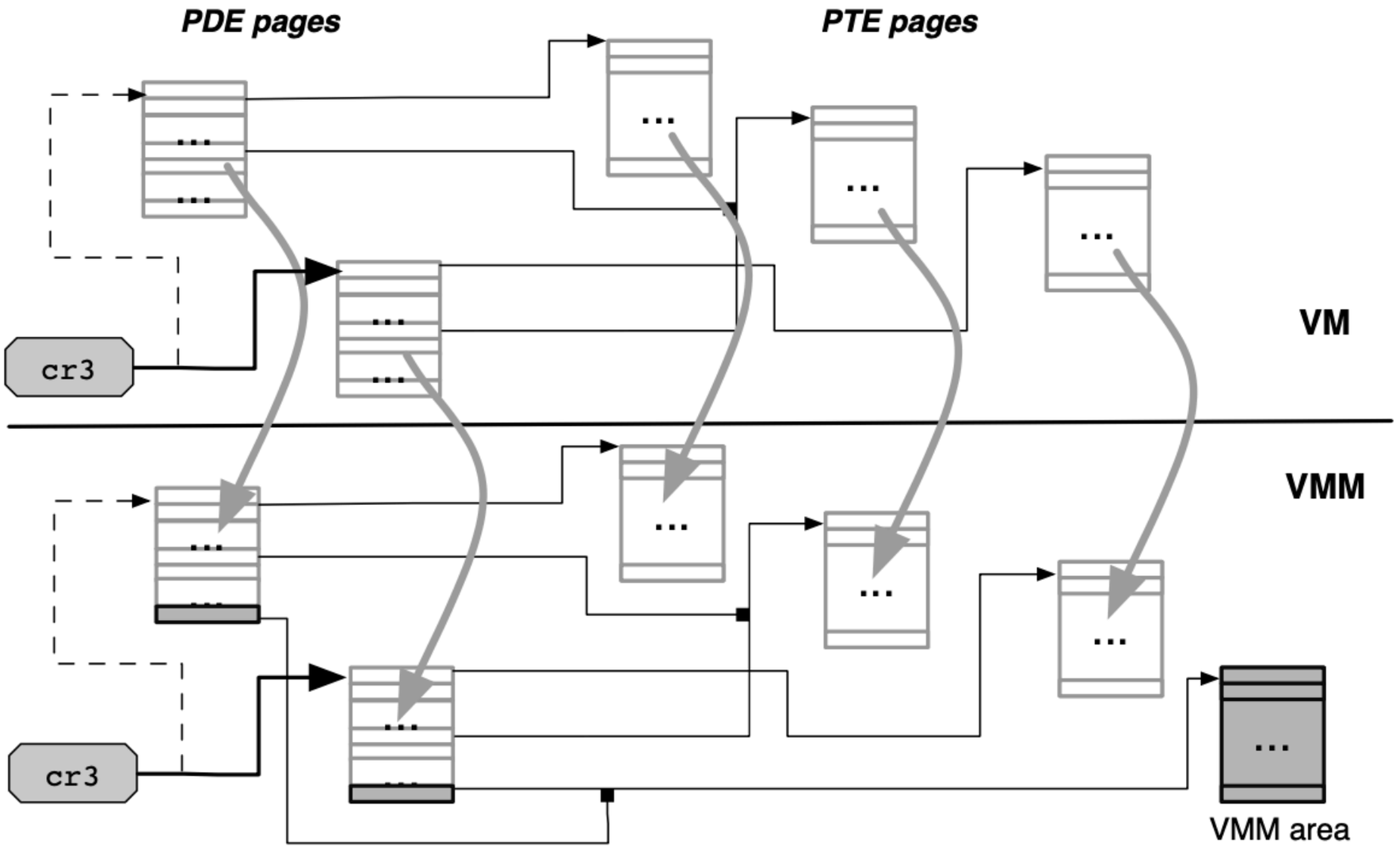
Action on page fault



Action on INVLPG

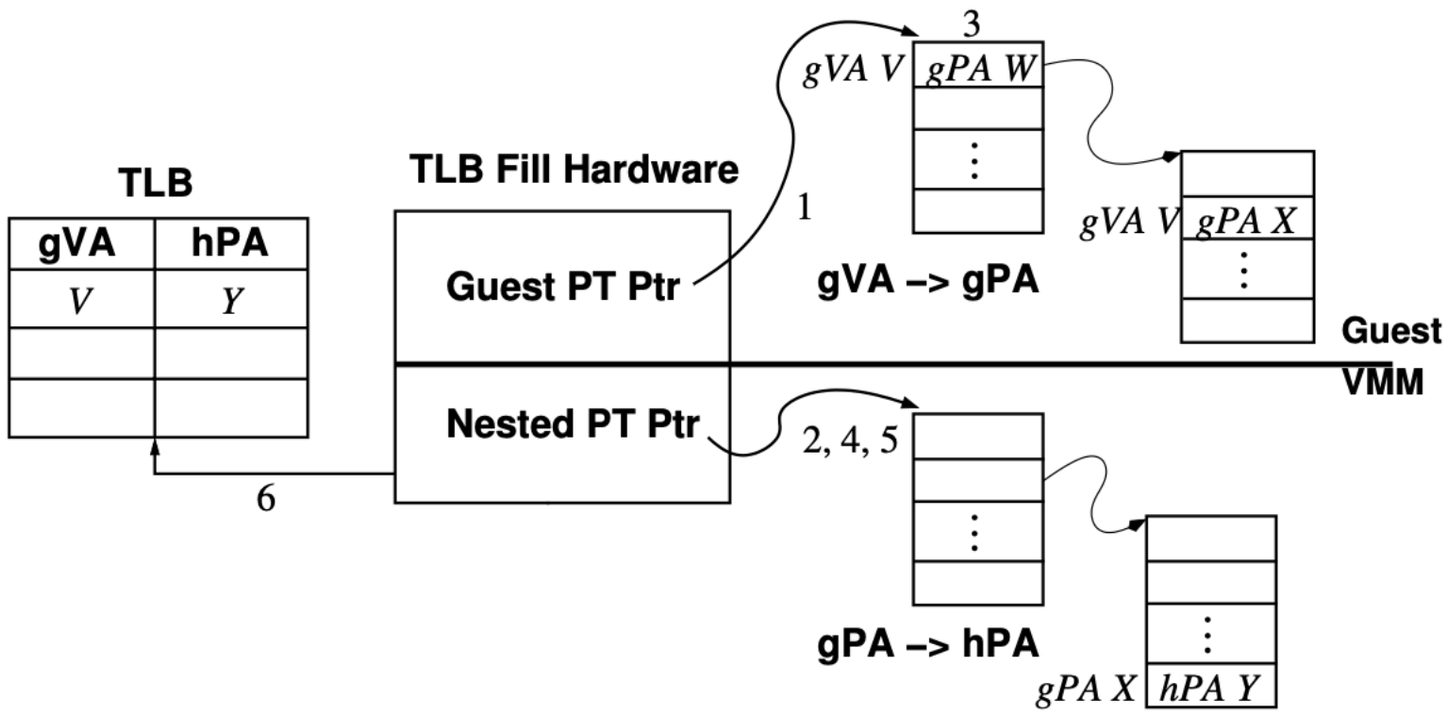


Shadow Page Tables



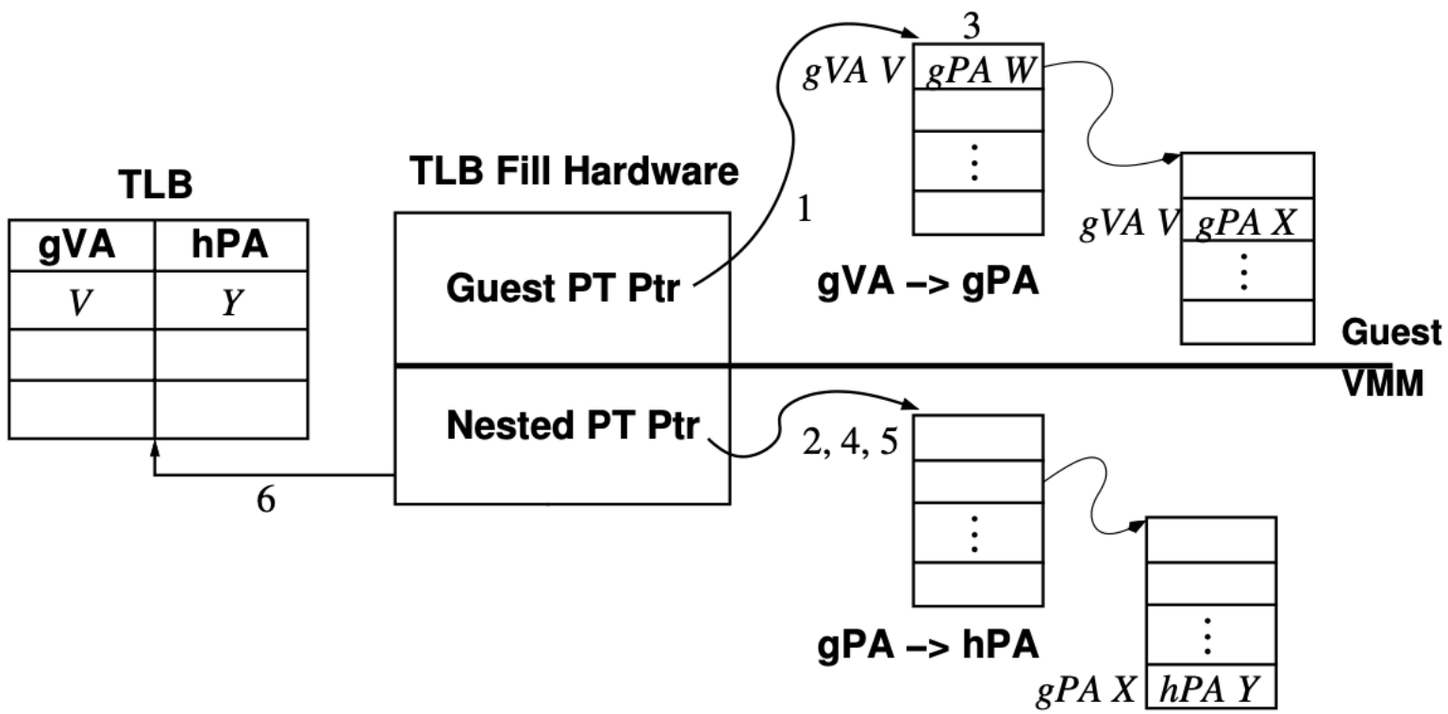
VT-x: EPT, NPT

- Extended/Nested Page Tables
  - One (base) page table maintained by VMM
  - One (nested) page table maintained by the guest



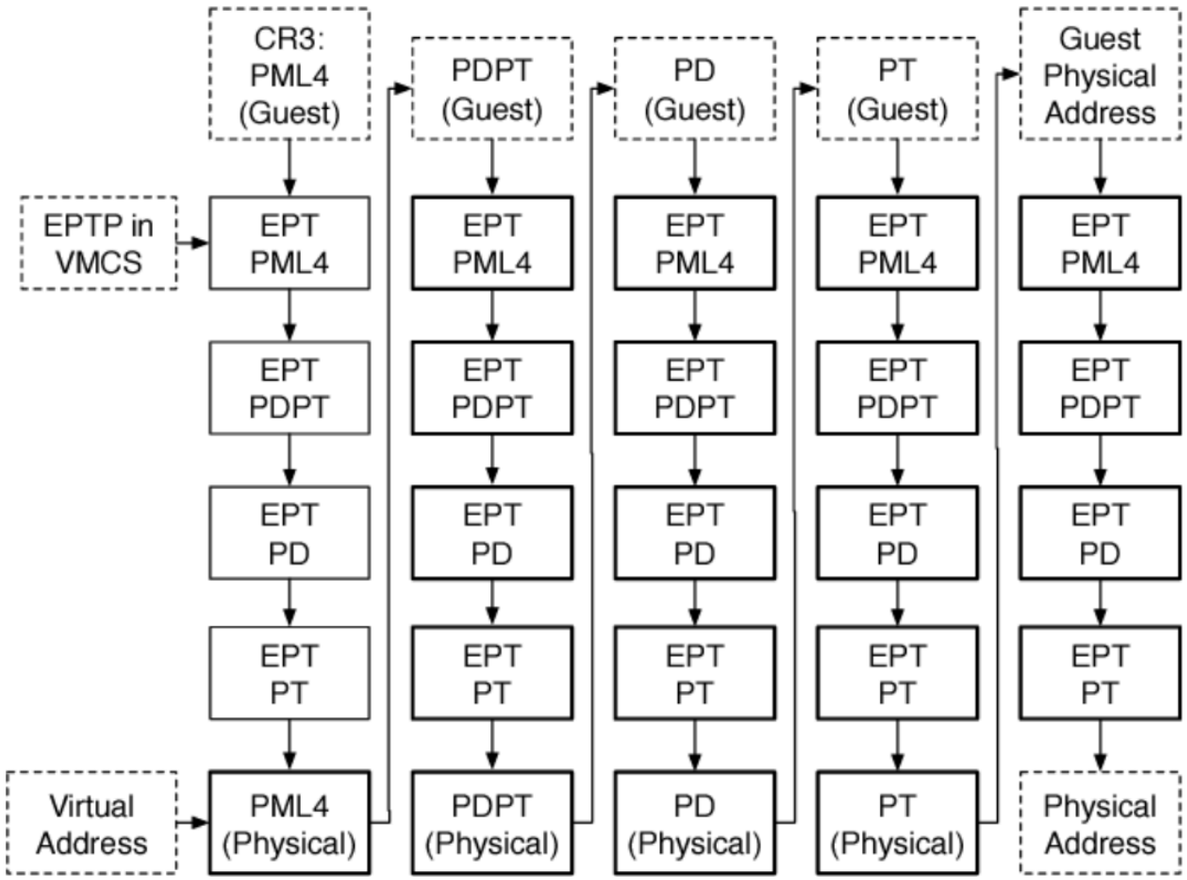
VT-x: EPT, NPT

- Extended/Nested Page Tables
  - One (base) page table maintained by VMM
  - One (nested) page table maintained by the guest
  - Guest has full control over its page tables
  - No VM exits due to (guest) page faults, INVLPG, or CR3 changes



VT-x: EPT, NPT

- EPT treats GPA like a HVA
- How many memory accesses upon TLB miss?

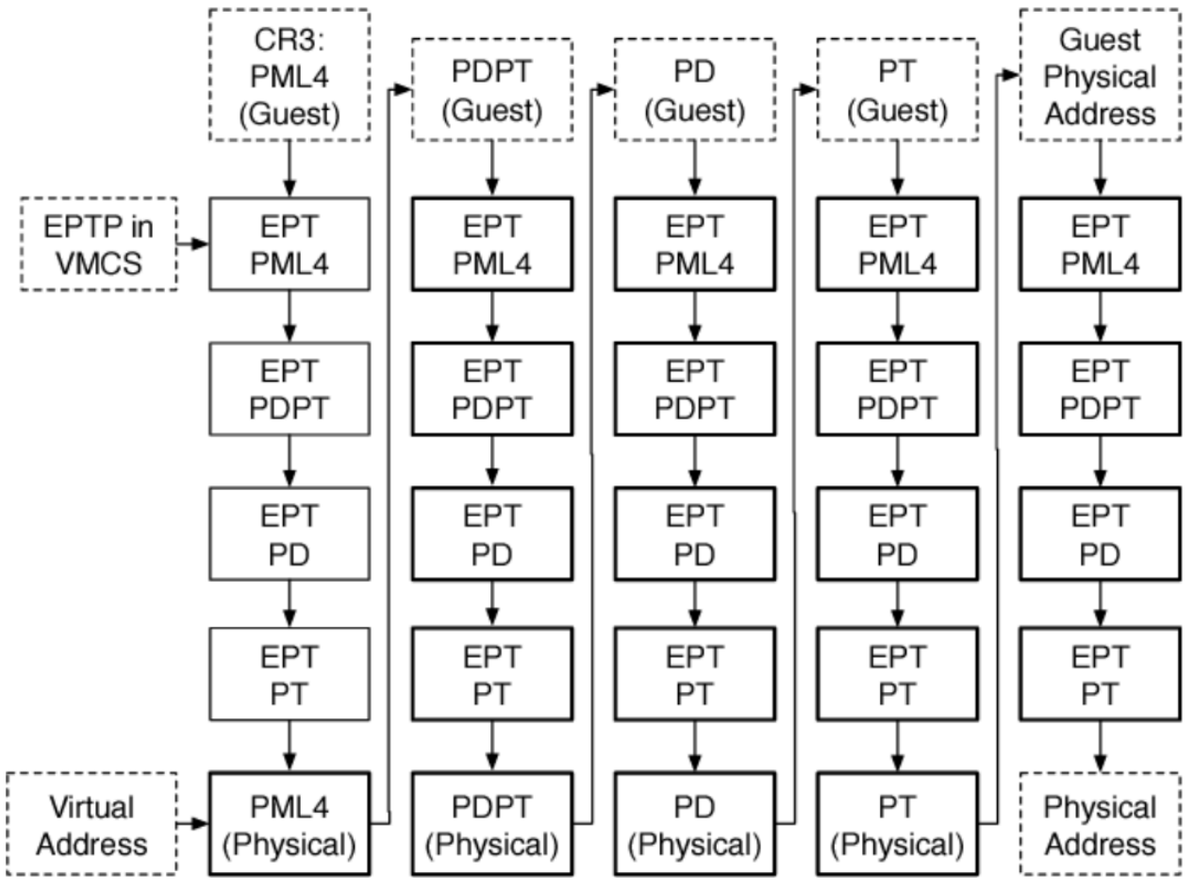


VT-x: EPT, NPT

- EPT treats GPA like a HVA
- How many memory accesses upon TLB miss?

N - depth of host page tables
M - depth of guest page tables

References = (N x M) + N + M



VT-x: EPT/NPT vs Shadow

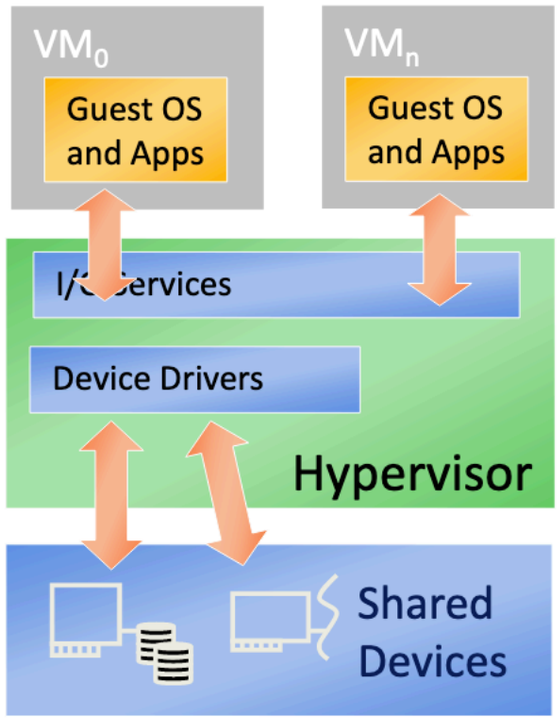
- Extended Page Tables
  - Walk any requested address
  - Reduced volume of VM exits
  - Two-layer reference (HW walk)
  - Many particular registers
  - HW support to notify VMM
- Shadow page tables
  - Only walk on SPT entry miss
  - Frequent intercepted accesses
  - One reference (HW walk)
  - Complicated reverse map
  - Permission emulation

VT-x: IO Virtualization

- IO-port bitmap execution control
- Paging controls to intercept MMIO
- Event injection by the VMM

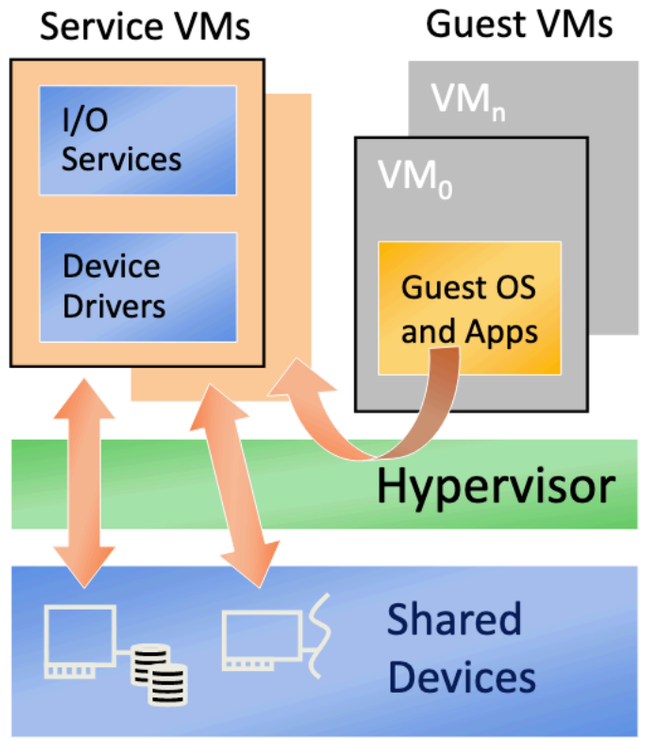
VT-d: IO Models

Monolithic Model



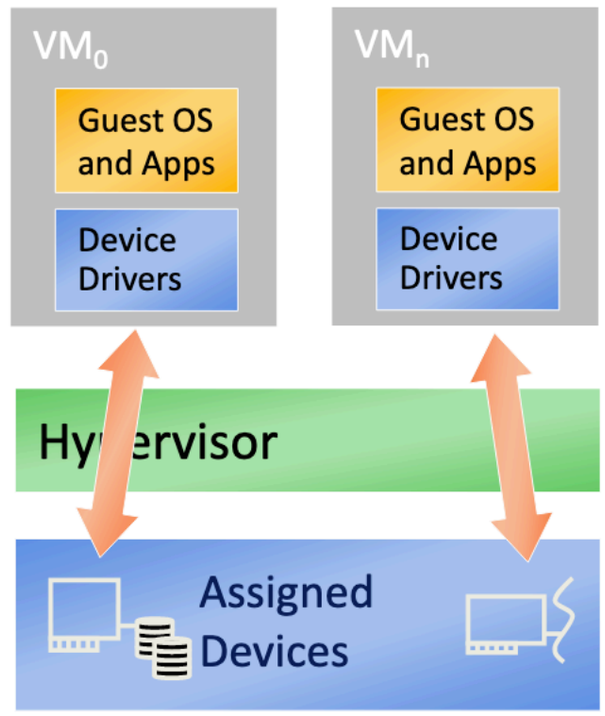
- Pro: Higher Performance
- Pro: I/O Device Sharing
- Pro: VM Migration
- Con: Larger Hypervisor

Service VM Model



- Pro: High Security
- Pro: I/O Device Sharing
- Pro: VM Migration
- Con: Lower Performance

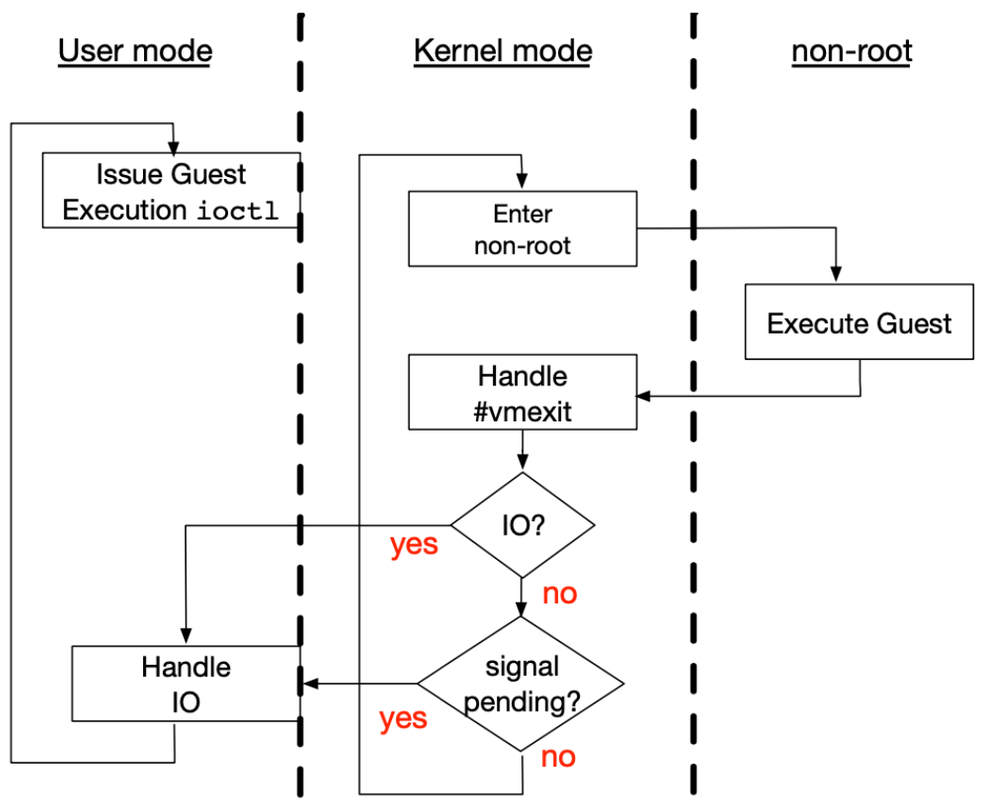
Pass-through Model



- Pro: Highest Performance
- Pro: Smaller Hypervisor
- Pro: Device assisted sharing
- Con: Migration Challenges

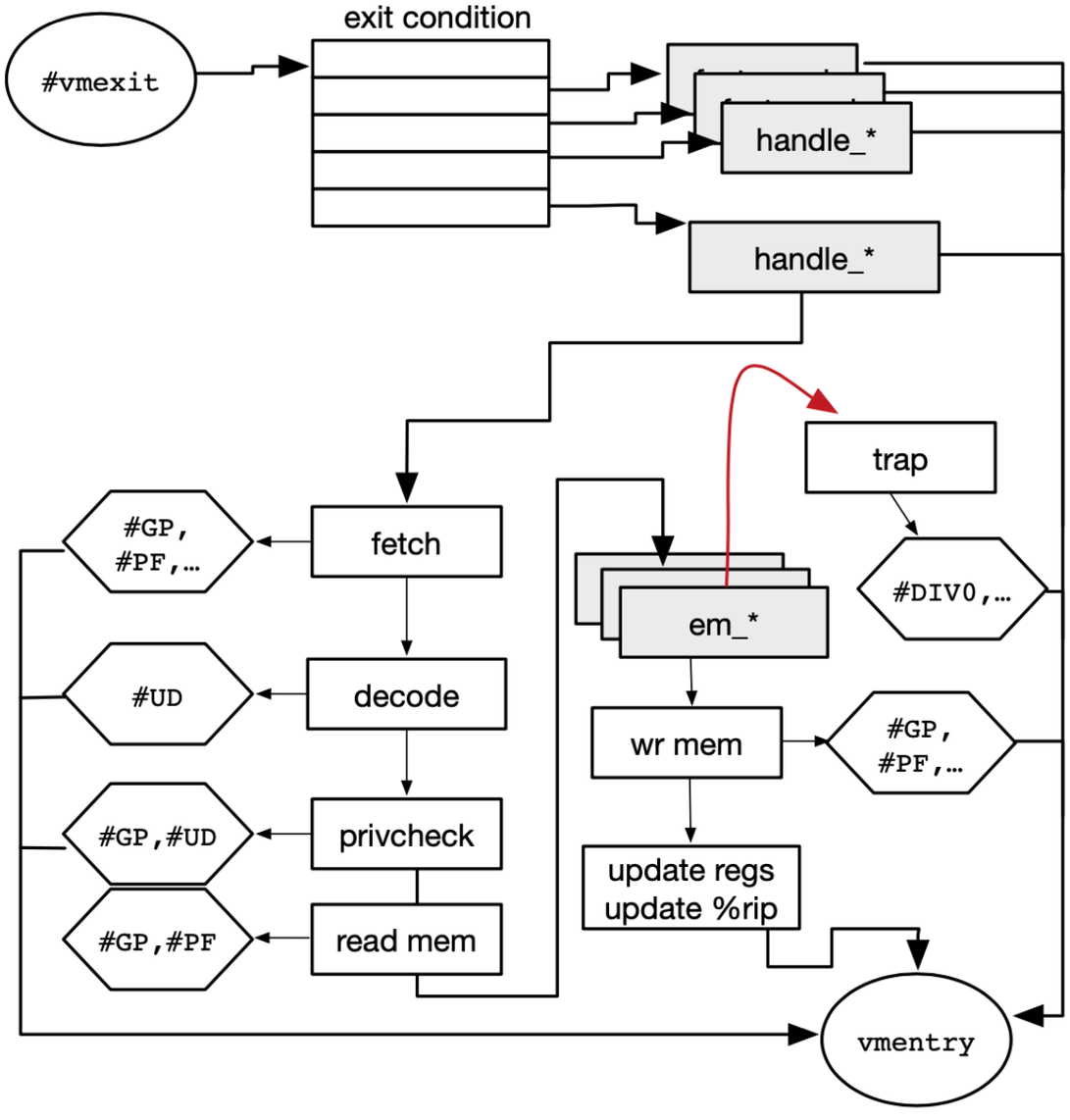
KVM: Kernel Virtual Machine

- Kernel Virtual Machine
  - Type II hypervisor, written as Linux kernel module
  - Separates CPU/MMU from device emulation (rely on QEMU)



KVM: Trap and Emulate

- General purpose flowchart of KVM



KVM: VM Exit Dispatch

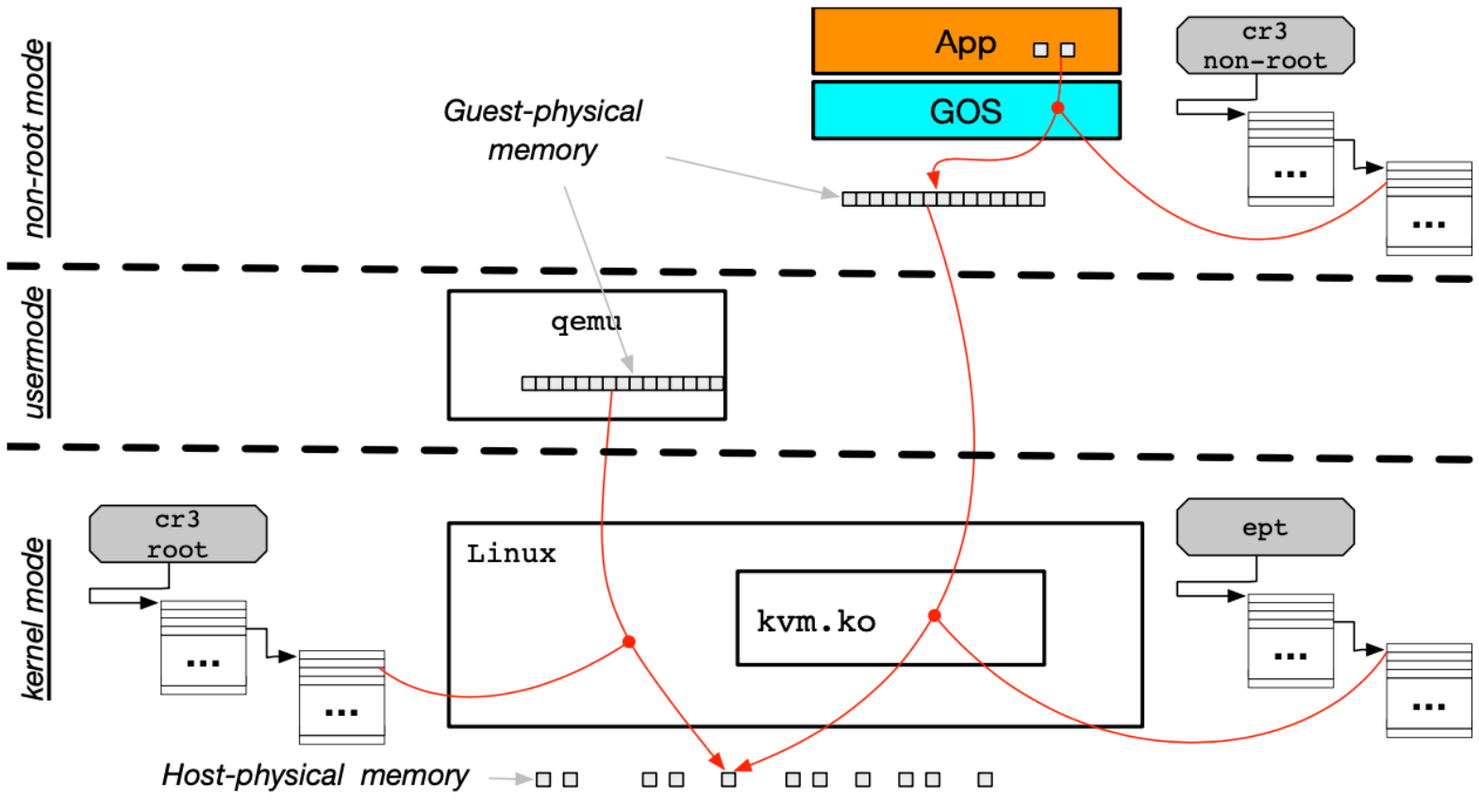
- Depending on the situation, KVM may either:
  - Emulate the instruction and increment the program counter
  - Forward the fault or interrupt to the guest (on the guest stack)
  - Adjust the guest environment and re-try the execution. (e.g.?)
  - Do nothing to the virtual machine state. (e.g. ?)

KVM: VM Exit Dispatch

- Emulate the instruction:
  - Fetch the instruction from guest virtual memory (how?)
  - *Decode* the instruction, extracting its operator and operands
  - *Verify* the instruction can execute (given current VMCS)
  - Read any memory read-operands from guest virtual machine (how?)
  - *Emulate* the decoded instruction (could be any ISA)
  - Write any memory write-operands to guest virtual machine (how?)
  - Update guest registers and the instruction pointer as needed

KVM: Memory Virtualization

- Support both nested page tables and shadow page tables
- 3 layers of memory tables: QEMU, VMM, guest



KVM Internals

Looking at the code

VMM Assortment

- How does VMM deal with resource overcommit?
- How does VMM migrate guests between hosts?
- Can VMM record and later replay guest execution?
- Can VMM introspect into guest execution?
- How to build a secure desktop with VMM?

VMM: Overcommit

- How does VMM deal with resource overcommit?

VMM: Overcommit

- How does VMM deal with resource overcommit?
  - Transparent page sharing (like Linux KSM)
    - Reclaim memory by consolidating pages with identical content
  - Ballooning of guest memory
    - Reclaim memory by increasing memory pressure inside guest
  - Hypervisor swapping
    - Reclaim memory by the host swapping out guest memory
  - Memory compression
    - Reclaim memory by compressing pages swapped out (to RAM)

VMM: Migration

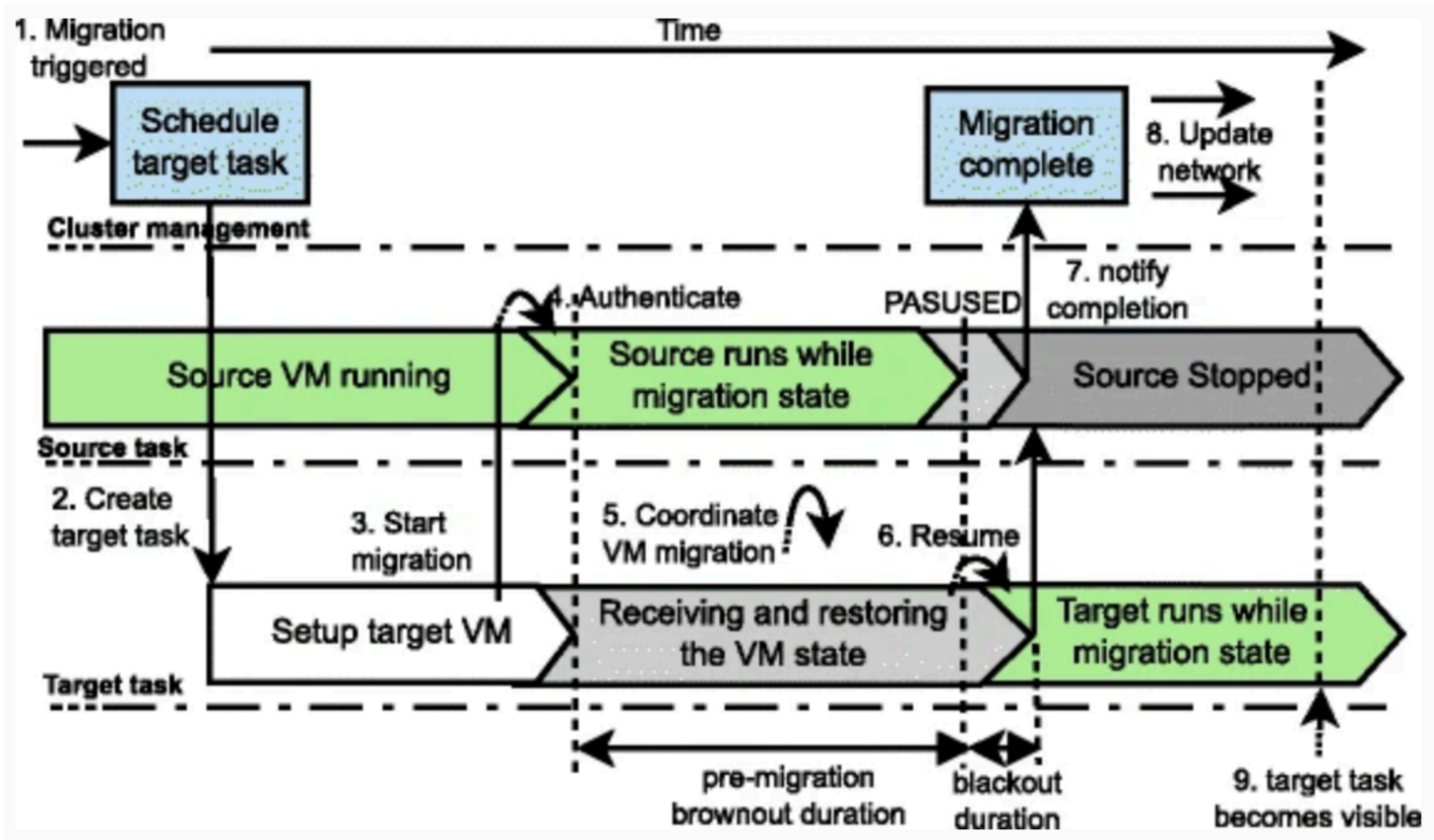
- How does VMM migrate guests between hosts?

VMM: Migration

- How does VMM migrate guests between hosts?
  - What needs to be migrated? How?
    - State? Storage? Devices? Network?
  - Non-live vs live migration
    - Guest downtime during migration vs host overhead?
  - Pre-copy vs post-copy vs hybrid
    - Pre-copy: warm-up then stop-and-copy
    - Post-copy: stop-and-setup then post-copy

VMM: Migration

- How does VMM migrate guests between hosts?



VMM: Migration

- How does VMM migrate guests between hosts?
  - What needs to be migrated? How?
    - State? Storage? Devices? Network?
  - Non-live vs live migration
    - Guest downtime during migration vs host overhead?
  - Pre-copy vs post-copy vs hybrid
    - Pre-copy: warm-up then stop-and-copy
    - Post-copy: stop-and-setup then post-copy

Non live	Simple; But large downtime
Pre copy live	Short downtime, safe; But overhead due to duplicate copy
Post copy live	Minimal downtime, copy once; But fetch latency, no error recovery