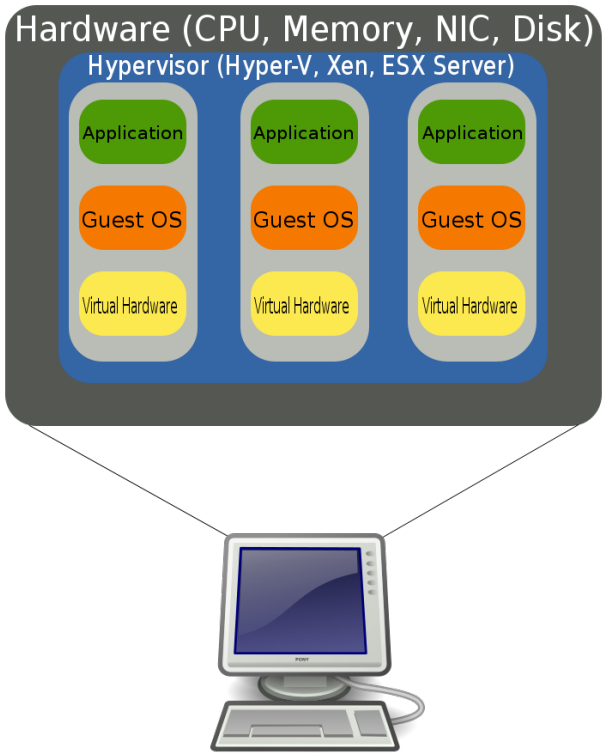
Virtualization

Credit: https://gcallah.github.io/OperatingSystems/index.html

Types of Virtualization

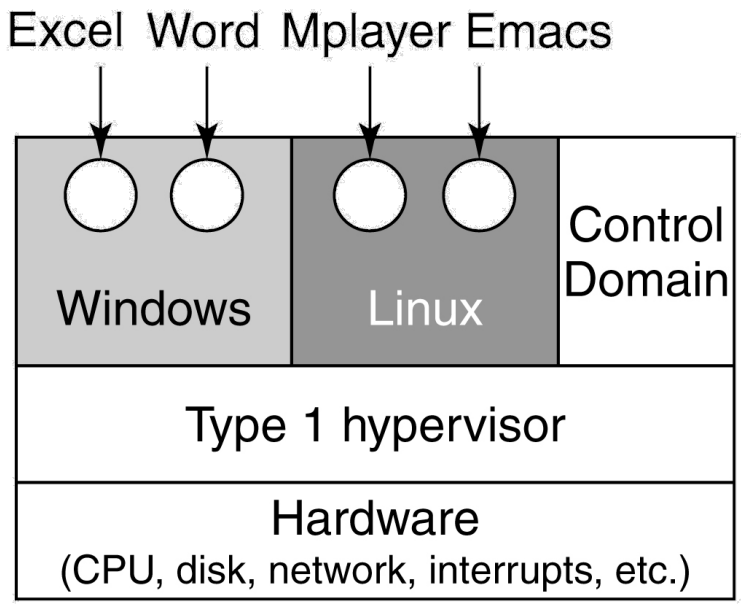
- Virtualize a computer/hardware
  - System Virtual Machine (e.g., VMWare, Xen, KVM)
- Virtualize an OS
  - Containers (e.g., LXC, Docker)
- Virtualize a process:
  - WINE, user-mode QEMU
- Virtualize a language environment
  - JVM, JavaScript Engine, etc.

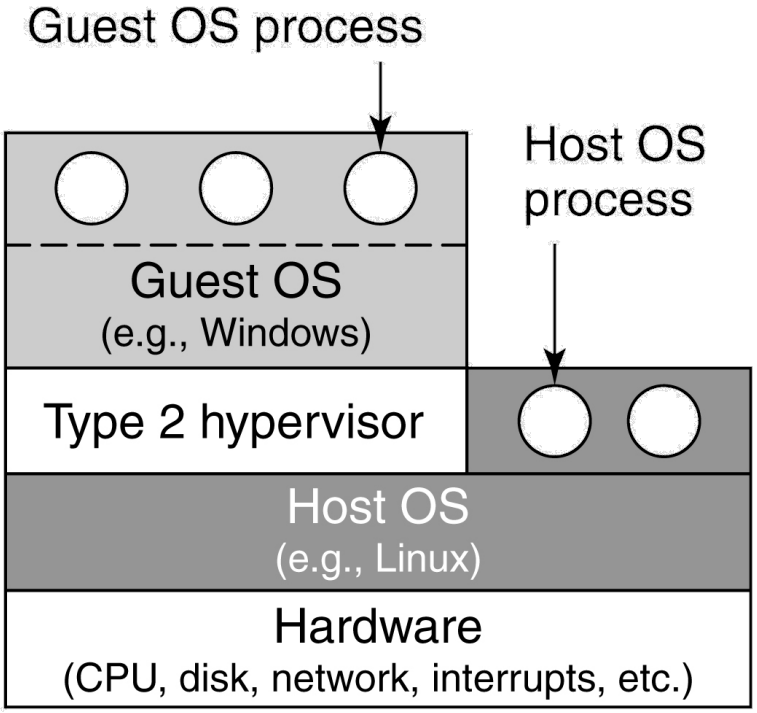
Requirements for Virtualization



- Requirements:
  - Safety
  - Fidelity/Transparency
  - Efficiency
- Possible Approaches
  - Interpreter: safe and faithful, but inefficient
  - Trap-and-emulate: efficient and faithful, safe?
- Popek and Goldberg theorems
  - Sensitive instructions must be a subset of privileged instructions

Type 1 and Type 2 Hypervisors





Virtualizaton method	Type 1 hypervisor	Type 2 hypervisor
Virtualization without HW support	ESX Server 1.0	VMware Workstation 1
Paravirtualization	Xen 1.0	
Virtualization with HW support	vSphere, Xen, Hyper-V	VMware Fusion, KVM, Parallels
Process virtualization		Wine

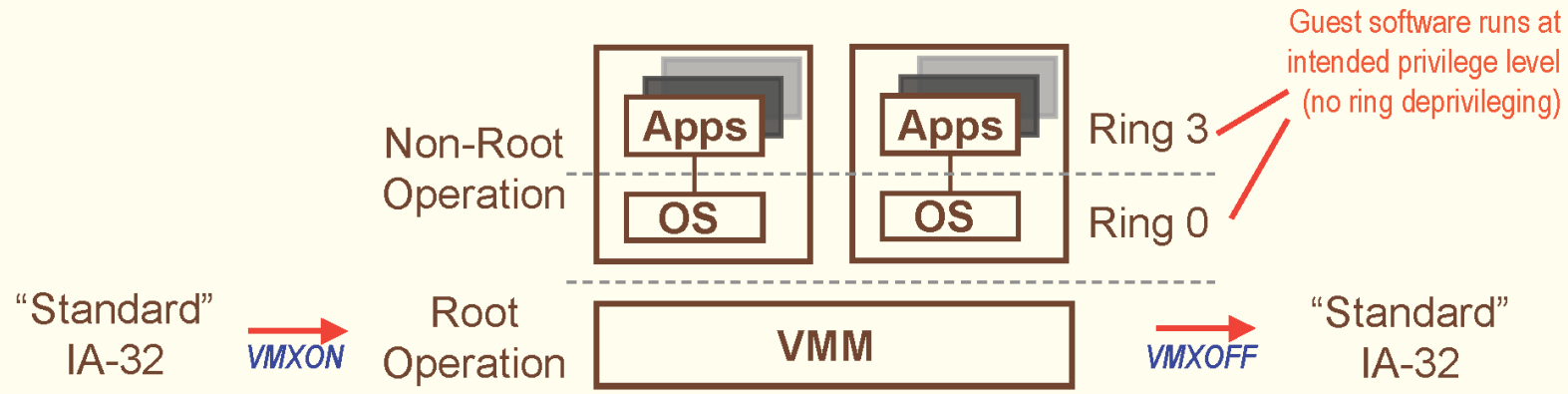
CPU Virtualization

- Paravirtualization:
  - Patch the sensitive instructions in the guest kernel to make hypercalls
  - E.g., Xen 1.0
- Dynamic Binary Translation:
  - At runtime, translate the sensitive instruction in the guest kernel
- Architectural Support:
  - Change the architecture, such that sensitive instructions are protected
  - E.g., Intel VT

Case Study: IA-32 Virtualization w/ VT-x

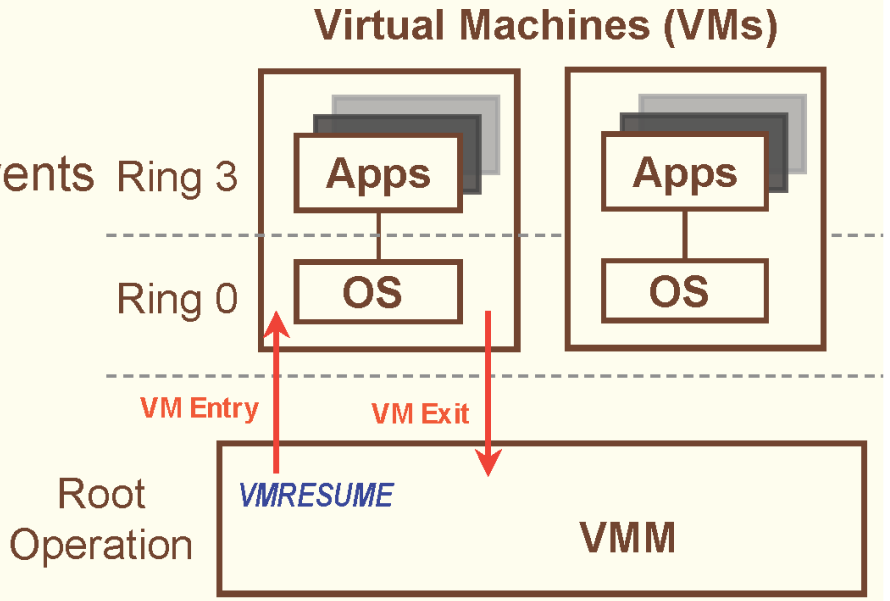
- **VT-x is a new operating mode for IA-32 processors**
  - Part of Intel® Virtualization Technology (VT)
  - Will be launched in Intel desktop CPUs in second half of 2005
- **Operating mode enabled with VMXON / VMXOFF**
- **VT-x provides two new forms of operation:**
  - Root Operation: Fully privileged, intended for VMM
  - Non-root Operation: Not fully privileged, intended for guest OS

Case Study: IA-32 Virtualization w/ VT-x



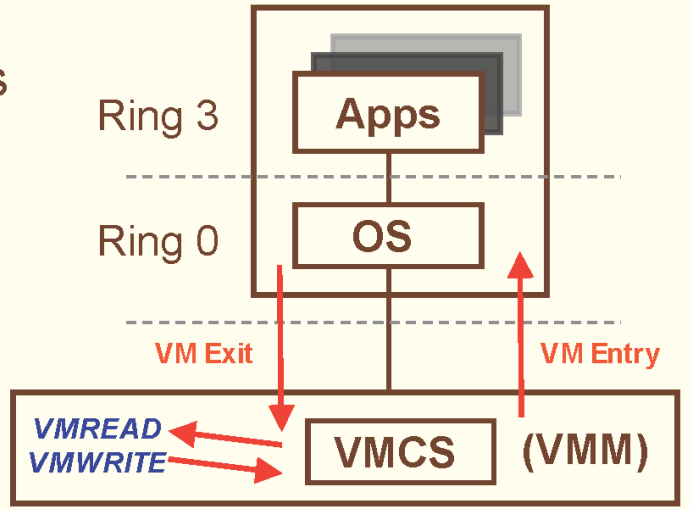
VT-x Transitions: VM Entry and VM Exit

- **VM Entry**
  - VMM-to-guest transition
  - Initiated by new instructions: *VMLAUNCH* or *VMRESUME*
  - Enters non-root operation, loading guest state
  - Establishes key guest state in a single, atomic operation
- **VM Exit**
  - Guest-to-VMM transition
  - Caused by virtualization events
  - Enters root operation
  - Saves guest state
  - Load VMM state



VT-x Config Flexibility with the VMCS

- **VM Control Structure (VMCS) specifies CPU behavior**
  - Holds guest state loaded / stored on VM entry / exit
  - Accessed through a ***VMREAD*** / ***VMWRITE*** interface
- **Configuration of VMCS controls guest OS behavior**
  - VMM programs VMCS to cause VM exits on desired events
- **VM exits possible on:**
  - Privileged State: CRn, DRn, MSRs
  - Sensitive Ops: CPUID, HLT, etc.
  - Paging events: #PF, INVLPG
  - Interrupts and Exceptions
- **Other optimizations:**
  - Bitmaps, shadow registers, etc.



The VM Control Structure (VMCS)

VM-execution controls	Determines what operations cause VM exits	CR0, CR3, CR4, Exceptions, IO Ports, Interrupts, Pin Events, etc.
Guest-state area	Saved on VM exits Reloaded on VM entry	EIP, ESP, EFLAGS, IDTR, Segment Regs, Exit info, etc.
Host-state are a	Loaded on VM exits	CR3, EIP set to monitor entry point, EFLAGS hardcoded, etc.
VM-exit controls	Determines which state to save, load, how to transition	Example: MSR save -load list
VM-entry controls	Determines which state to load, how to transition	Incl uding injecting events (interrupts, exceptions) on entry

- **Each virtual CPU has a separate VMCS**
  - For MP guest OS: separate VMCS for each “virtual CPU”
- **One VMCS per logical CPU is active at any given time**
  - ***VMPTRLD*** instruction used to switch from one VMCS to another

Example VM-exit Causes

- **Sensitive Instructions**
  - ***CPUID*** – Reports processor capabilities
  - ***RDMSR***, ***WRMSR*** – Read and write "Model-Specific Registers"
  - ***INVLPG*** – Invalidate TLB Entry
  - ***RDPMC, RDTSC*** – Read Perf Mon or Time-Stamp Counters
  - ***HLT***, ***MWAIT***, ***PAUSE*** – Indicate Guest OS Inactivity
  - ***VMCALL*** – New Instruction for Explicit Call to VMM
- **Accesses to Sensitive State**
  - ***MOV DRx*** – Accesses to Debug Registers
  - ***MOV CRx*** – Accesses to Control Registers
  - ***Task Switch*** – Accesses to CR3
- **Exceptions and Asynchronous Events**
  - Page Faults, Debug Exceptions, Interrupts, NMIs, etc.

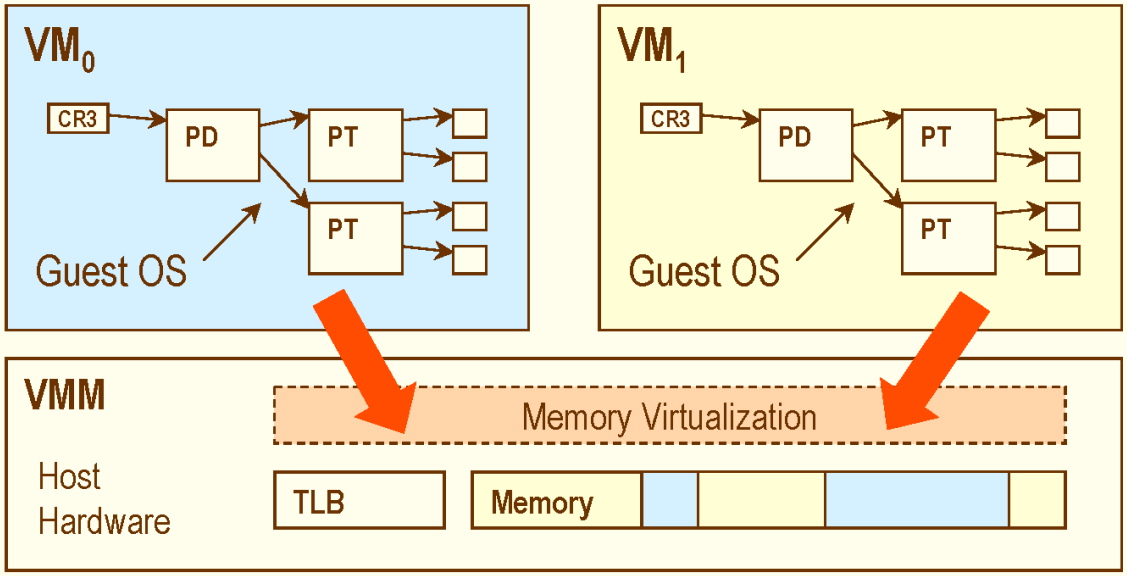
Some Example VM-Exit Optimizations

- **VT-x provides various optimizations to minimize frequency of VM exits:**
- **Shadow Registers and Masks**
  - Reads from CR0 and CR4 are satisfied from shadow registers established by the VMM
  - VM exits can be conditional based on the specific bits modified on a CR write (via a mask)
- **Execution-Control Bitmaps**
  - VM exits can be selectively controlled via bitmaps (e.g., for exceptions, IO-port accesses)

System Virtualization Case Studies

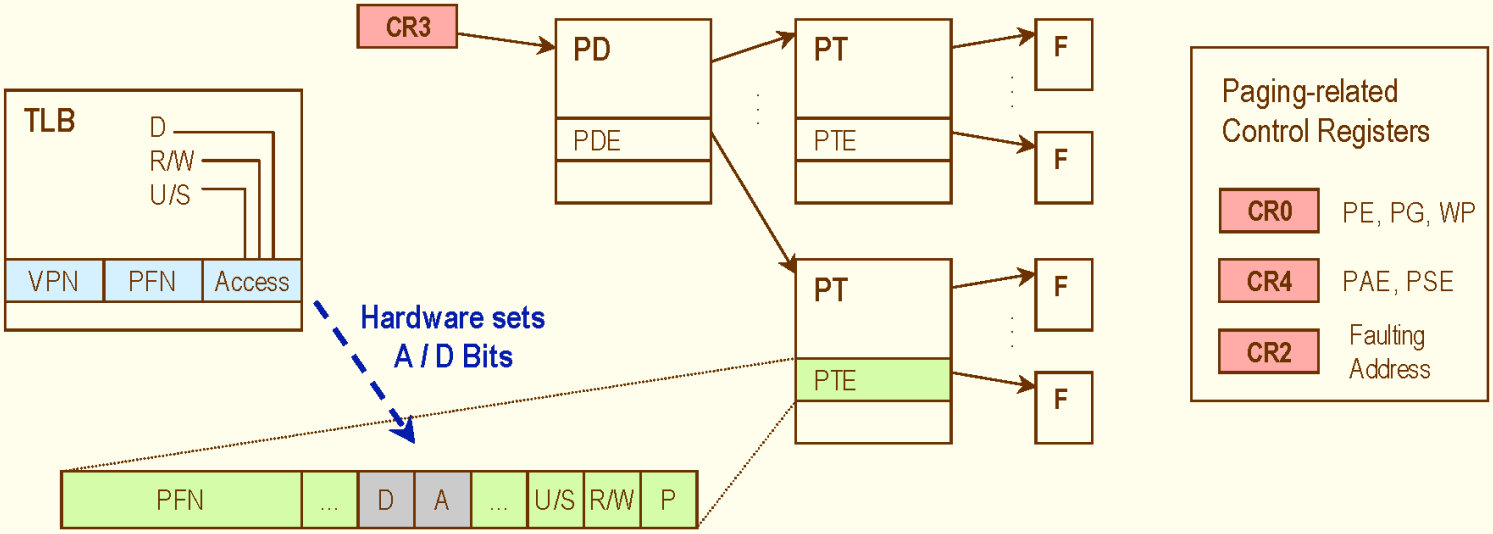
Memory Virtualization

Mem Virtualization: General Principles



- **Guest OS expects to control address translation**
  - Allocates memory, page tables, manages TLB consistency, etc.
- **But, VMM must have ultimate control over phys mem**
  - Must map guest-physical address space to host-physical space

A Case Study: IA-32 Address Translation

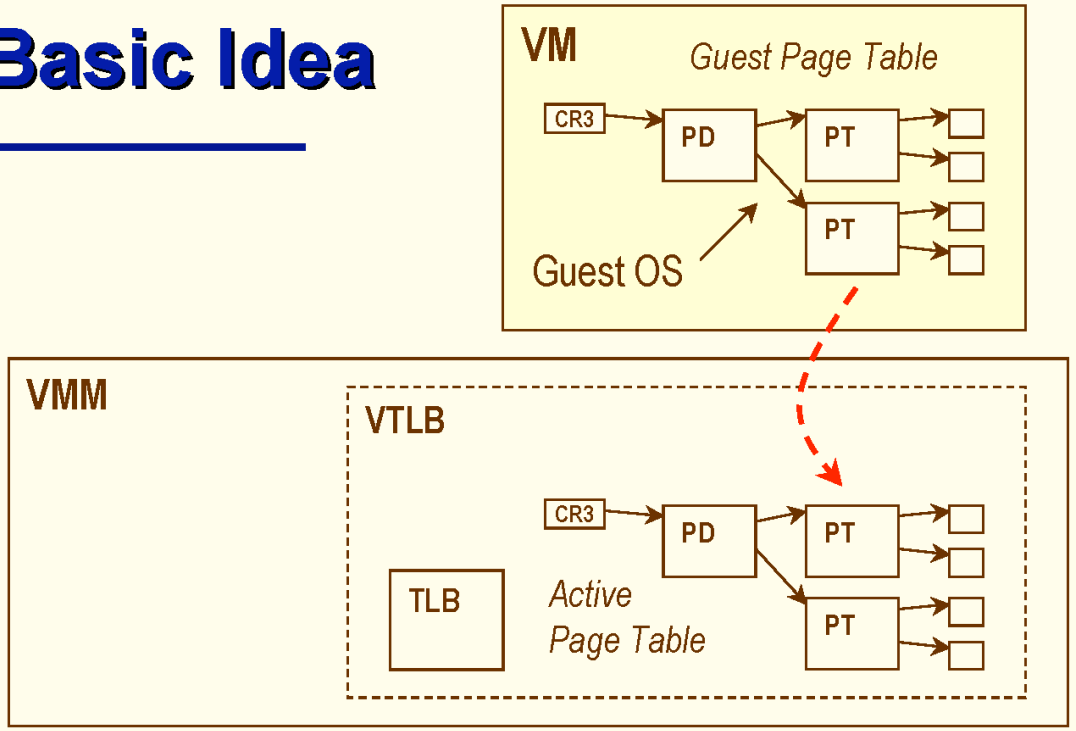


- **IA-32 defines a hierarchical page-table structure**
  - Defines linear-to-physical address translation
  - After page-table walk, page-table Entries (PTEs) are cached in a hardware TLB
- **IA-32 address translation configured via control registers (CR3, etc.)**
- **Invalidation of PTEs signaled by OS via *INVLPG* instruction**

Virtualizing Page Tables: Some Options

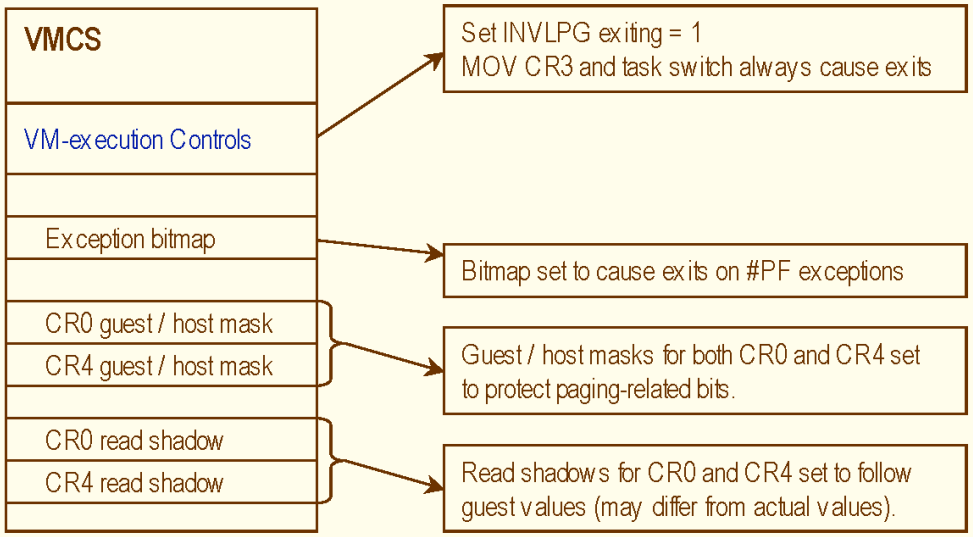
- **Option 1: Protect access to guest-OS page tables (PTs)**
  - Use paging protections or binary translation to detect changes
  - Upon *write* access, substitute remapped phys address in PTE
  - Also need VM exit on page-table *reads* (to report original PTE value to guest OS)
- **Option 2: Make a shadow copy of page tables**
  - Guest OS freely changes its page tables
  - VM exit occurs whenever CR3 changes
  - VMM copies contents of *guest* page tables to *active* page tables
  - Copy operation is analogous to a TLB refill, hence: "Virtual TLB"
- **Details of option 2 follow**
  - As illustration of the use of VT-x…

Virtual TLB: Basic Idea



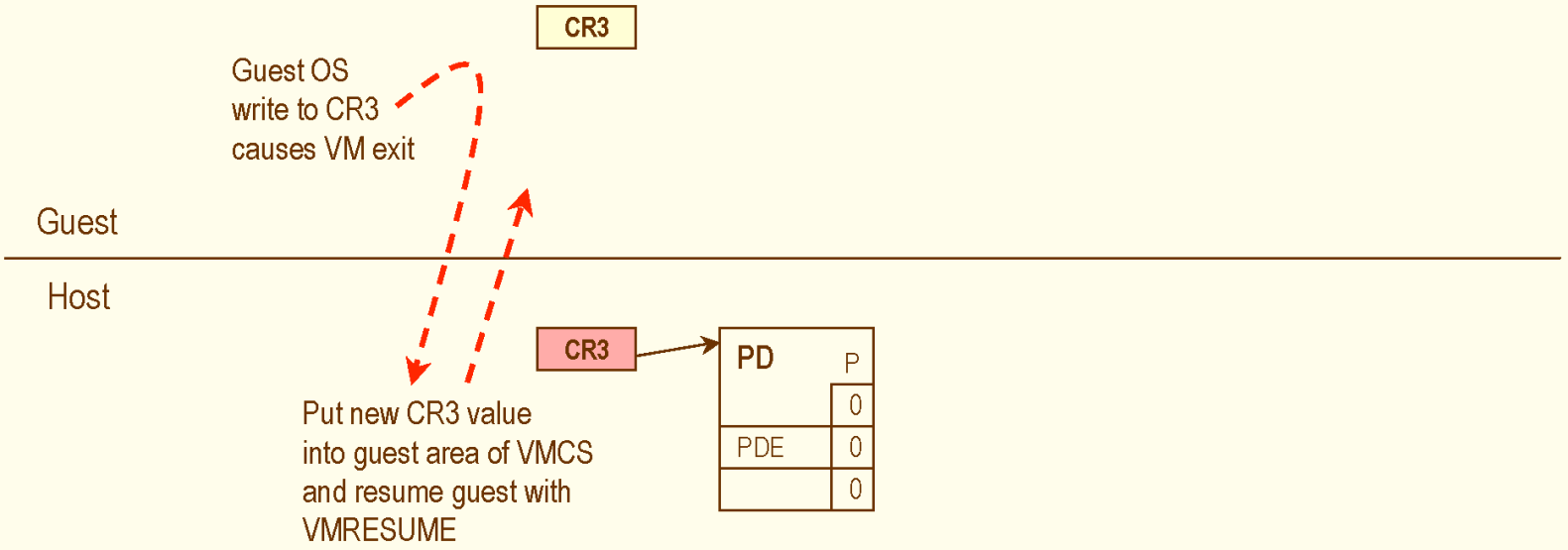
- **VTLB = Processor TLB + Active Page Table**
  - VMM initializes an empty VTLB and starts guest execution
  - When guest accesses memory, #PF occurs, and is sent to VMM
  - VMM copies needed translation (VTLB refill) and resumes guest

Virtual TLB: VT-x Setup



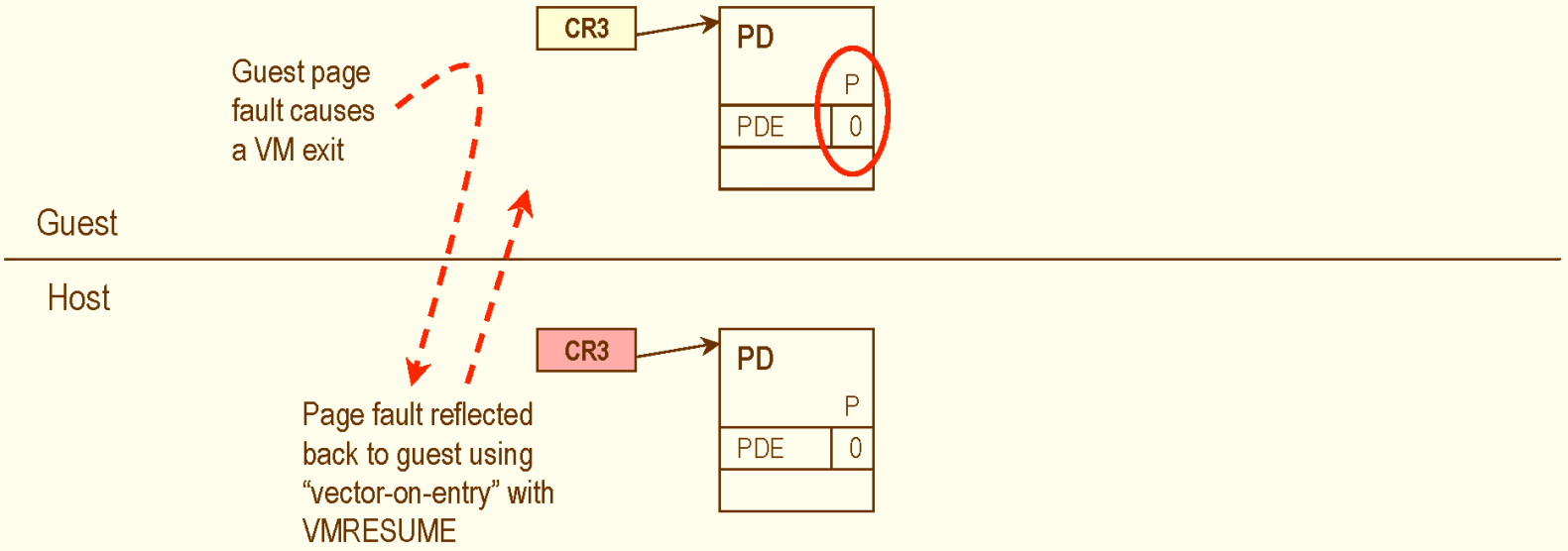
- **VTLB algorithm programs VMX to cause VM exits on:**
  - Any writes to CR3 and relevant writes to CR0 and CR4
  - Any page-fault (#PF) exceptions
  - Any executions of INVLPG

Virtual TLB: Actions on CR3 Write



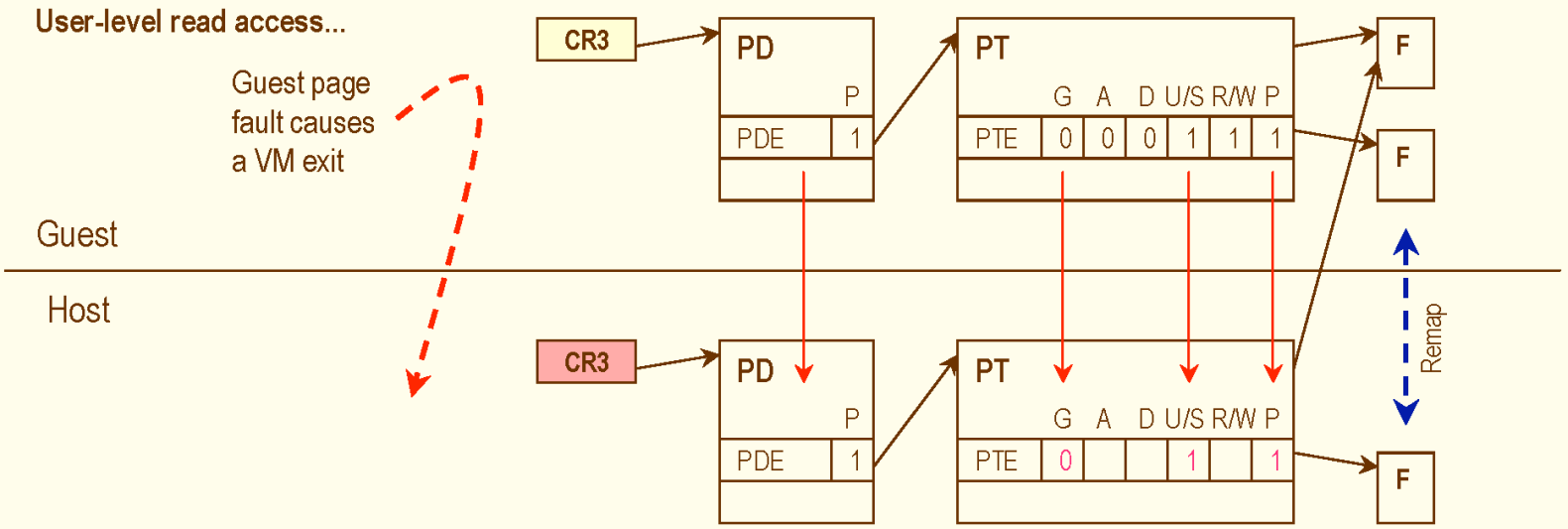
- **CR3 write implies a TLB flush and page-table change**
  - VMM notes new CR3 value (used later to walk guest PT)
  - VMM allocates a new PD page, with all invalid entries
  - VMM sets actual CPU CR3 register to point to the new PD page

Virtual TLB: Actions on a Page Fault



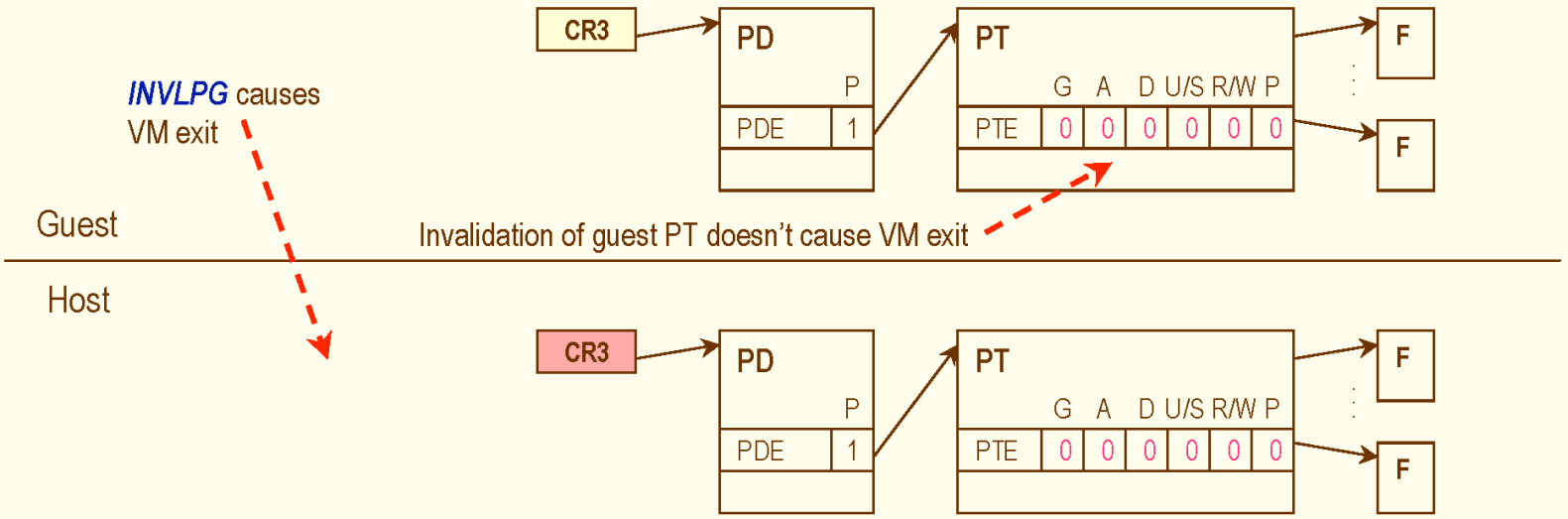
- **VMM examines guest PT using faulting addr**
  - If relevant PTE or PDE is invalid (P=0), then the #PF must be reflected to the guest OS.
  - VMM configures VMCS for a "#PF vector-on-entry"
  - Then resumes guest execution with a VMRESUME

Virtual TLB: Actions on a Page Fault (2)



- **If guest page table indicates sufficient access, then…**
  - VMM allocates PT and copies guest PTE to the active PT
  - PFN of active PTE remapped to new value as per VMM policy
  - Other active PTE bits set as in guest PTE (e.g., P, G, U/S)

Virtual TLB: Actions on INVLPG



- **Guest OS permitted to freely modify its page tables**
  - Implies guest PTs and active PTs can become inconsistent
  - This is okay! (same as inconsistencies between PTs and TLB)
  - If guest reduces access, signals via INVLPG, causing a VM exit
  - VMM invalidates corresponding PTE in the active PT

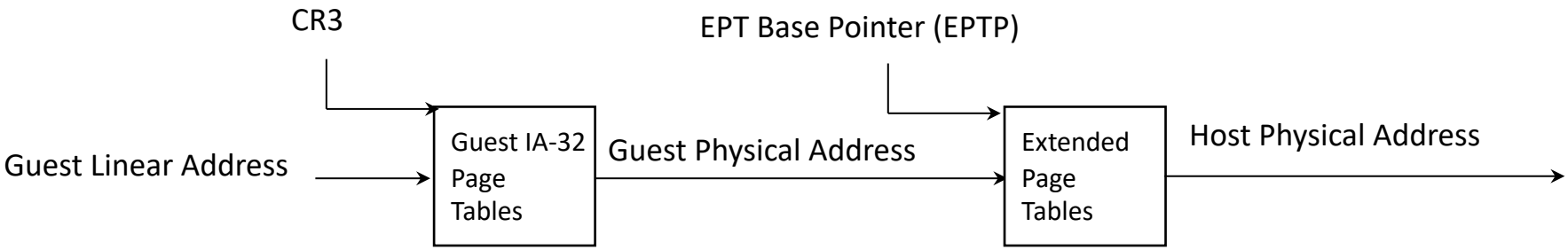
Extended Page Tables (EPT)

- A VMM must protect host physical memory
  - Multiple guest operating systems share the same host physical memory
  - VMM typically implements protections through "page-table shadowing" in software

- Page-table shadowing accounts for a large portion of virtualization overheads
  - VM exits due to: #PF, INVLPG, MOV CR3

Goal of EPT is to reduce these overheads

What Is EPT?

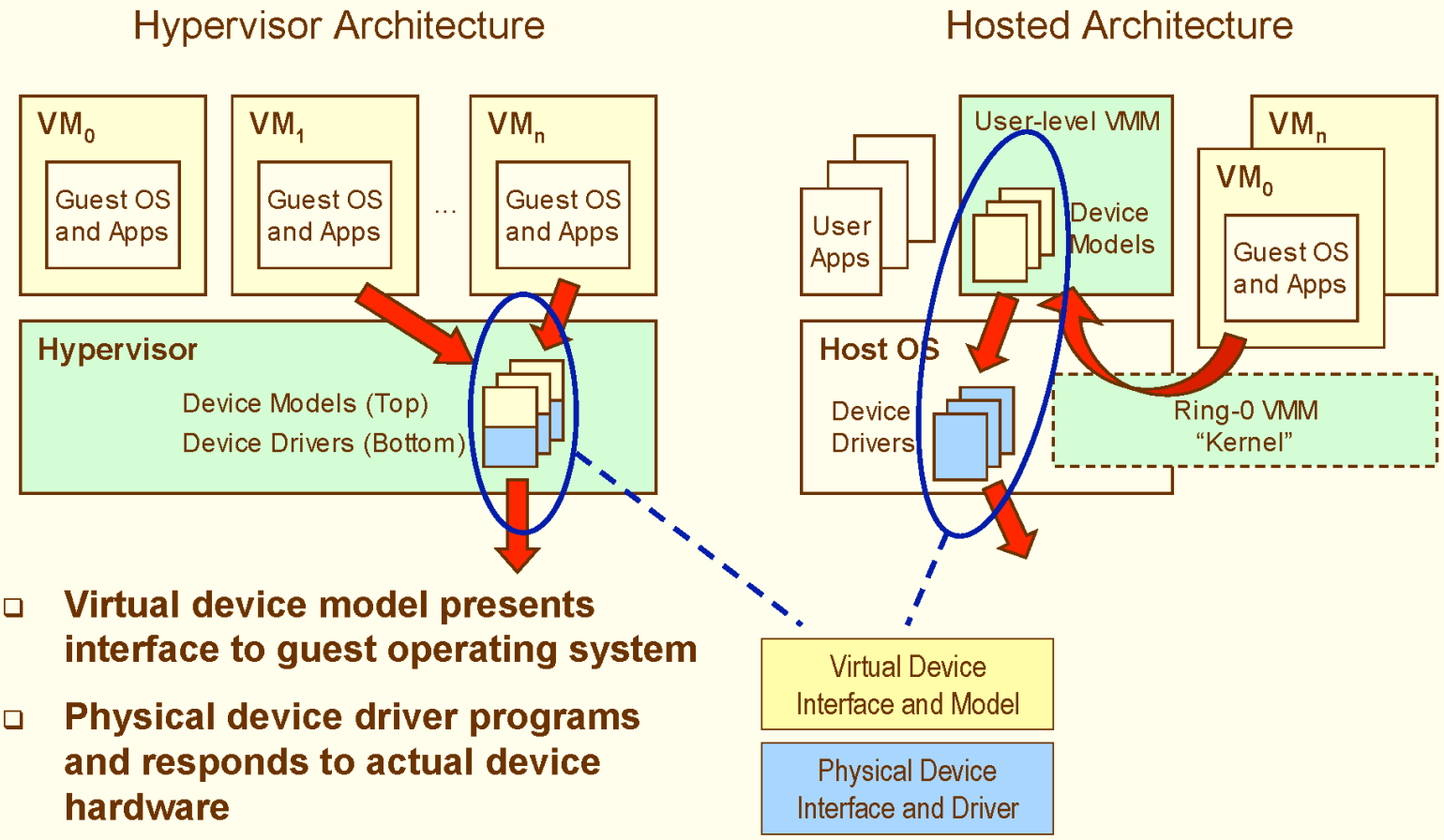


- Extended Page Table
- A new page-table structure, under the control of the VMM
  - Defines mapping between guest- and host-physical addresses
  - EPT base pointer (new VMCS field) points to the EPT page tables
  - EPT (optionally) activated on VM entry, deactivated on VM exit
- Guest has full control over its own IA-32 page tables
  - No VM exits due to guest page faults, INVLPG, or CR3 changes

System Virtualization Case Studies

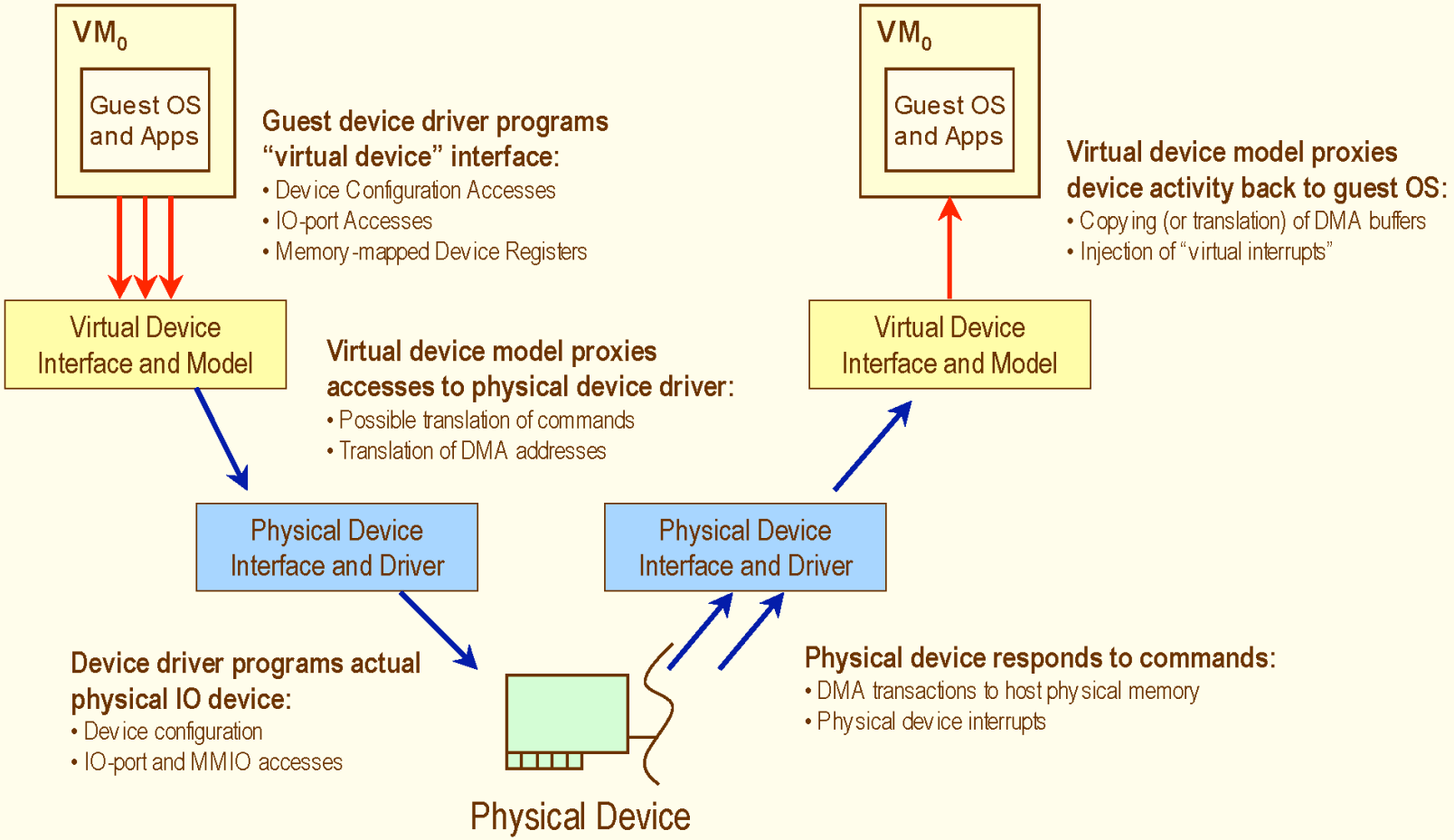
IO-Device Virtualization

IO Virtualization: General Principles

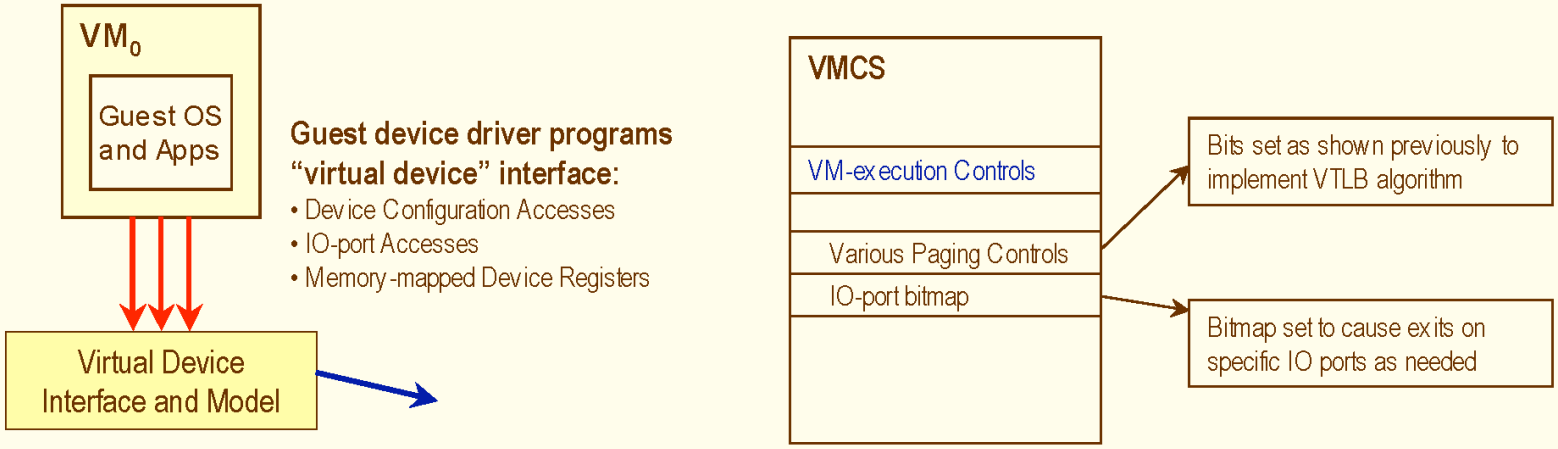


- **Virtual device model presents interface to guest operating system**
- **Physical device driver programs and responds to actual device hardware**

Virtual and Physical Device Interfaces

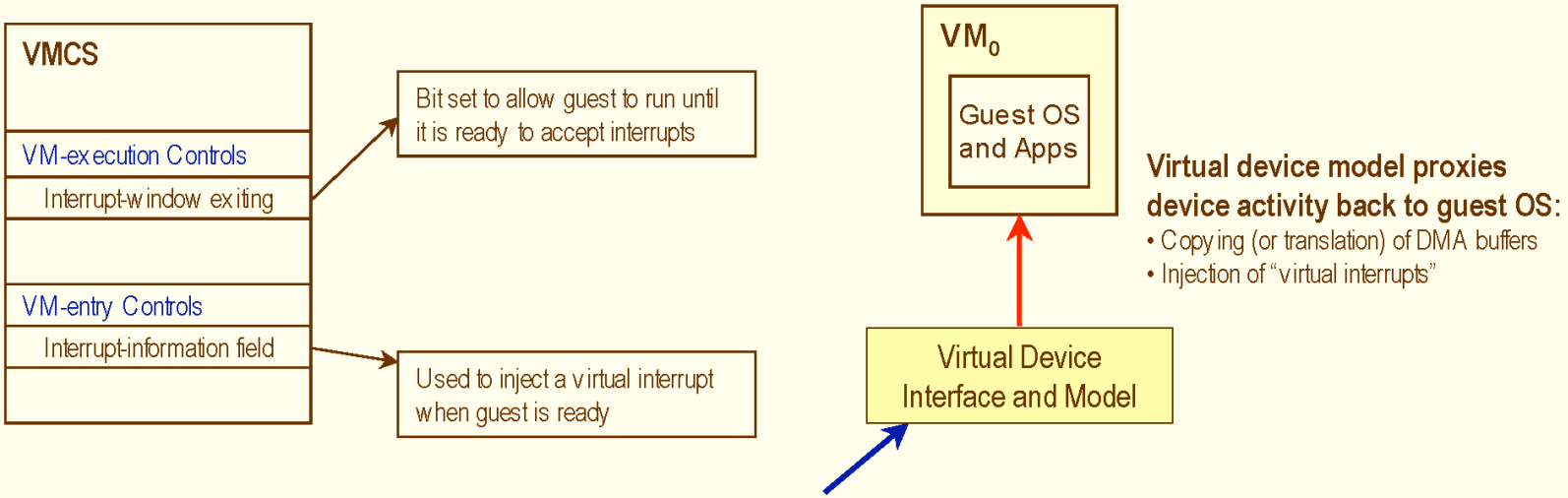


Case Study: IO Virtualization with VT-x



- **VT-x provides and IO-port bitmap execution control**
  - Enables VMM to intercept any IO-port accesses for bus configuration or IO-device control
- **VT-x provides paging controls to intercept MMIO**
  - VTLB-like algorithm can enforce VM exits on physical pages with memory-mapped IO (MMIO) registers

IO Virtualization with VT-x (cont.)



- **VT-x Interrupt-window exiting**
  - Guest OS may not be interruptible (e.g., critical section)
  - Interrupt-window exiting allows guest OS to run until it has enabled interrupts (via EFLAGS.IF)
- **VT-x Event Injection on VM entry**
  - Enables VMM to vector interrupt through guest IDT on VM entry

IO Paravirtualization