

# **Virtual Machines: Architectures, Implementations and Applications**

## **HOTCHIPS 17 Tutorial 1, Part 2**

**J. E. Smith  
University of Wisconsin-Madison**

**Rich Uhlig  
Intel Corporation**

*August 14, 2005*

# **System Virtual Machines**

**Rich Uhlig  
Intel Corporation**

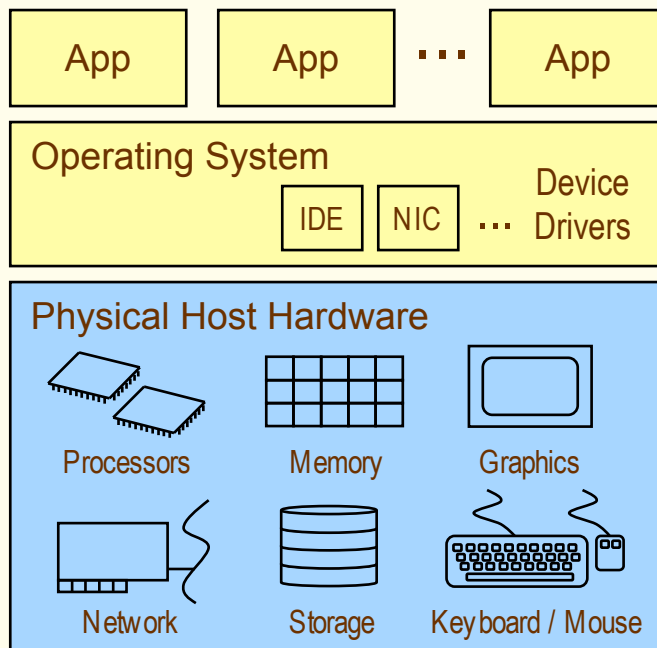
# System Virtual Machines: Outline

---

- ❑ Applications and Usage Models
- ❑ Virtualization Methods and VMM Software Architecture
- ❑ Hardware Resource Virtualization
  - General principles of CPU virtualization (with IA-32 / Intel VT\* case study)
  - General principles of memory virtualization (page-table shadowing case study)
  - General principles of IO virtualization
- ❑ Wrap-up

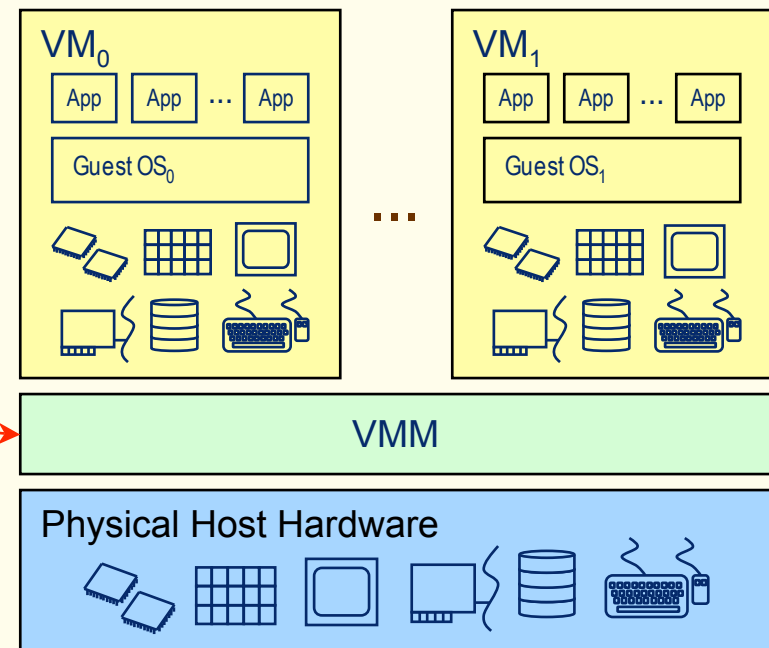
\* Intel® Virtualization Technology (VT)

# System Virtual Machines (VMs)



Without VMs: Single OS owns all hardware resources

A new layer of software...



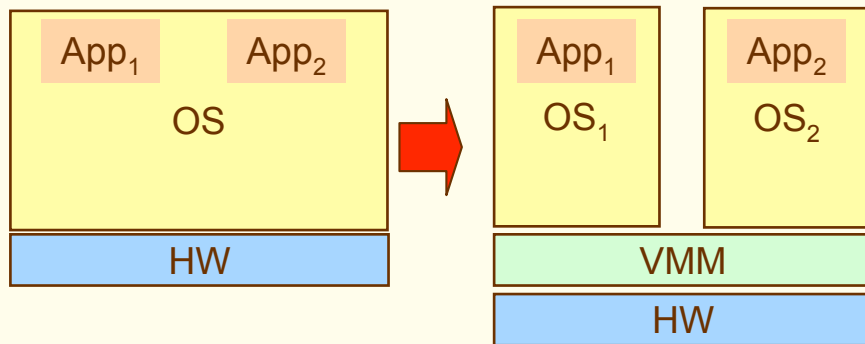
With VMs: Multiple OSES share hardware resources

- **A Virtual Machine Monitor (VMM) honors existing hardware interfaces to create virtual copies of a complete hardware system**

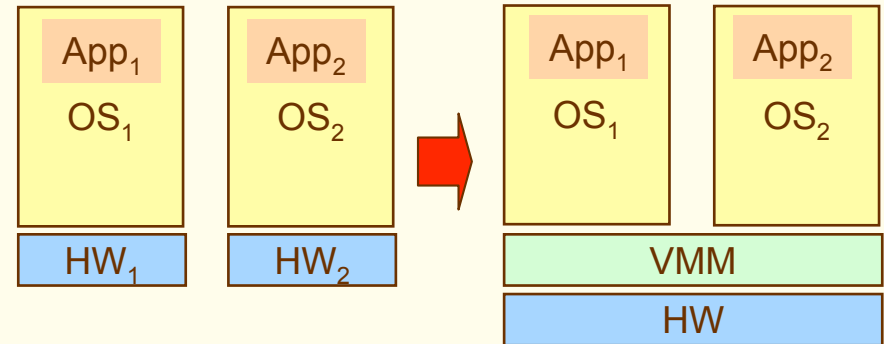
# **System VMs: Applications and Usage Models**

# Basic System VM Capabilities

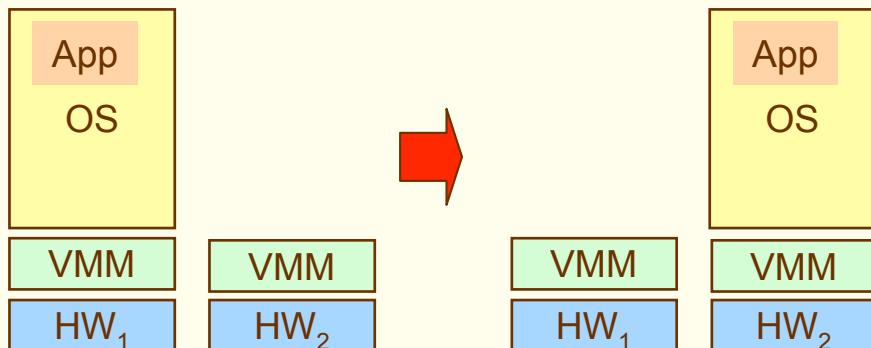
## Workload Isolation



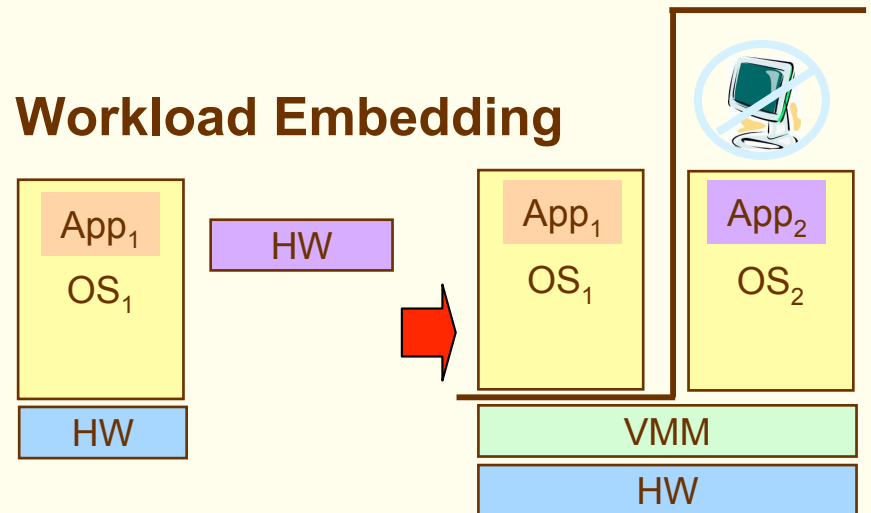
## Workload Aggregation



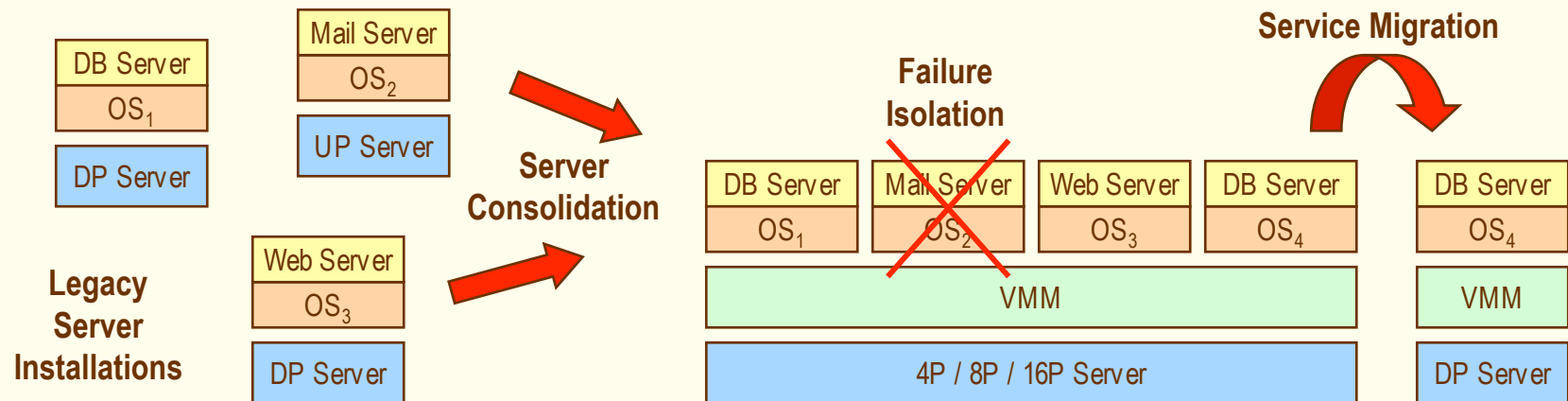
## Workload Migration



## Workload Embedding



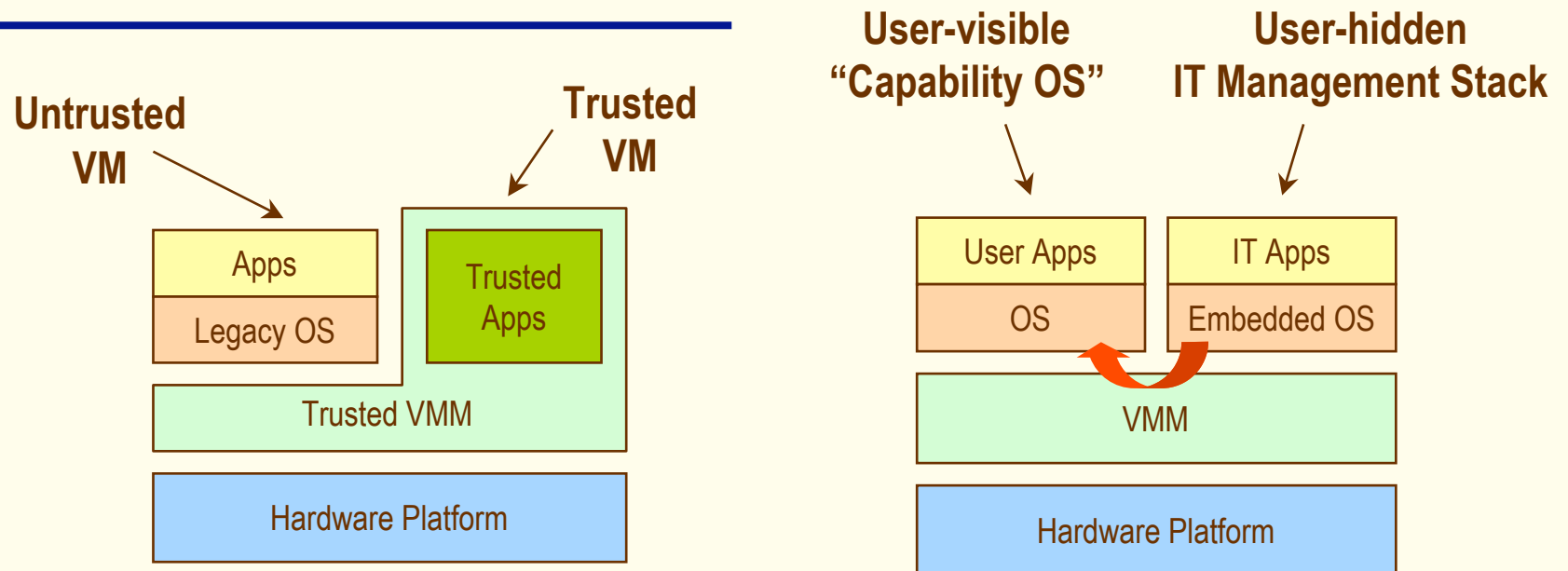
# Traditional Server Applications



## ❑ Manageability, Reliability, Availability

- Server consolidation (Legacy OSes, “One App per OS”)
- Staged deployment of OS upgrades, security patches, etc.
- Software failures confined to VM in which they occur
- Service migration in “Virtual Data Center”

# Emerging Client Applications



## ❑ Security / Trusted Computing

- VMs encapsulate untrusted legacy software
- Create new environment for trusted code

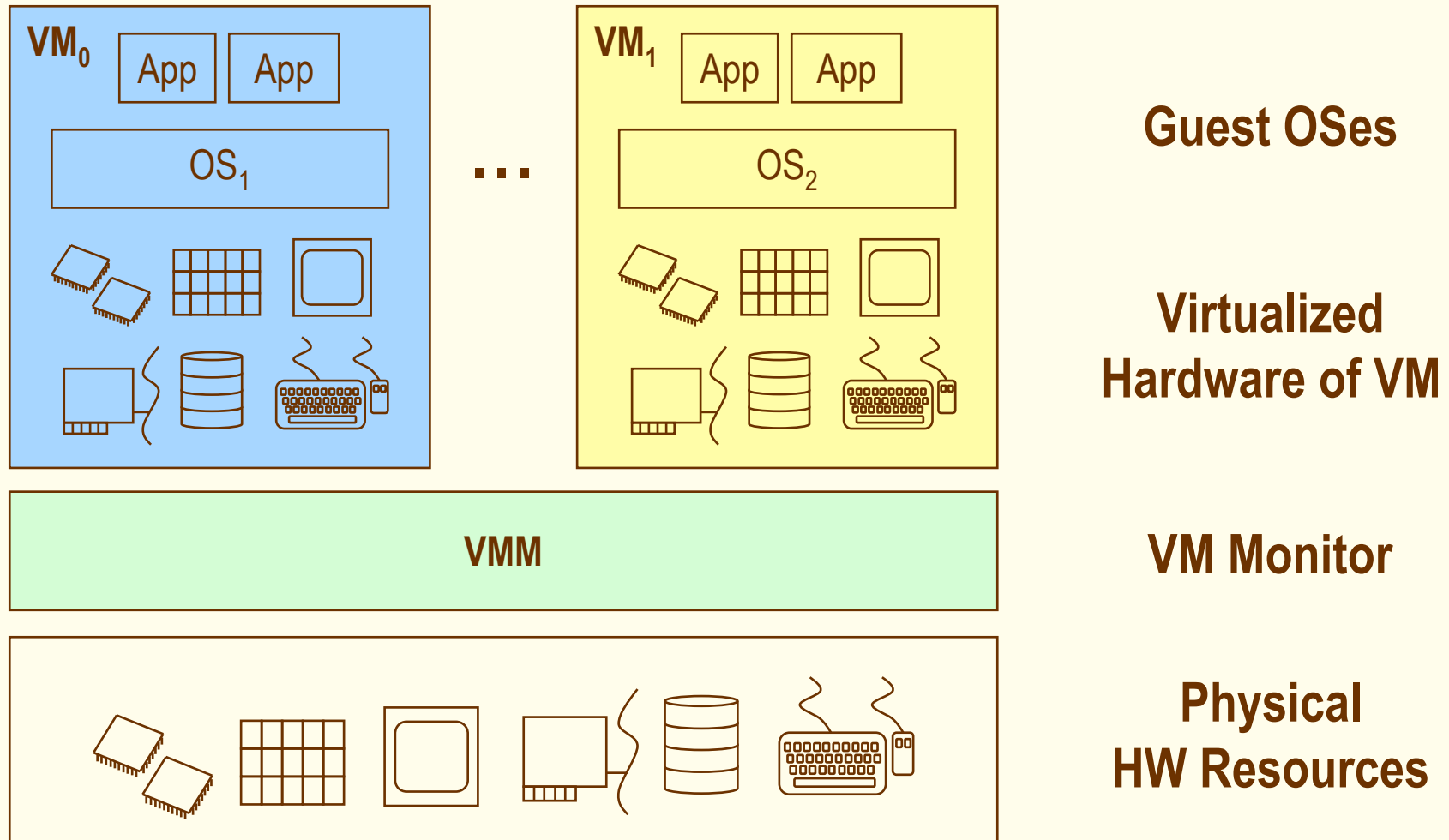
## ❑ Client Partitioning

- Extending server manageability features to the client (e.g., "Embedded IT" client)



# **Virtualization Methods and VMM Software Architecture**

# Anatomy of a Virtualized System



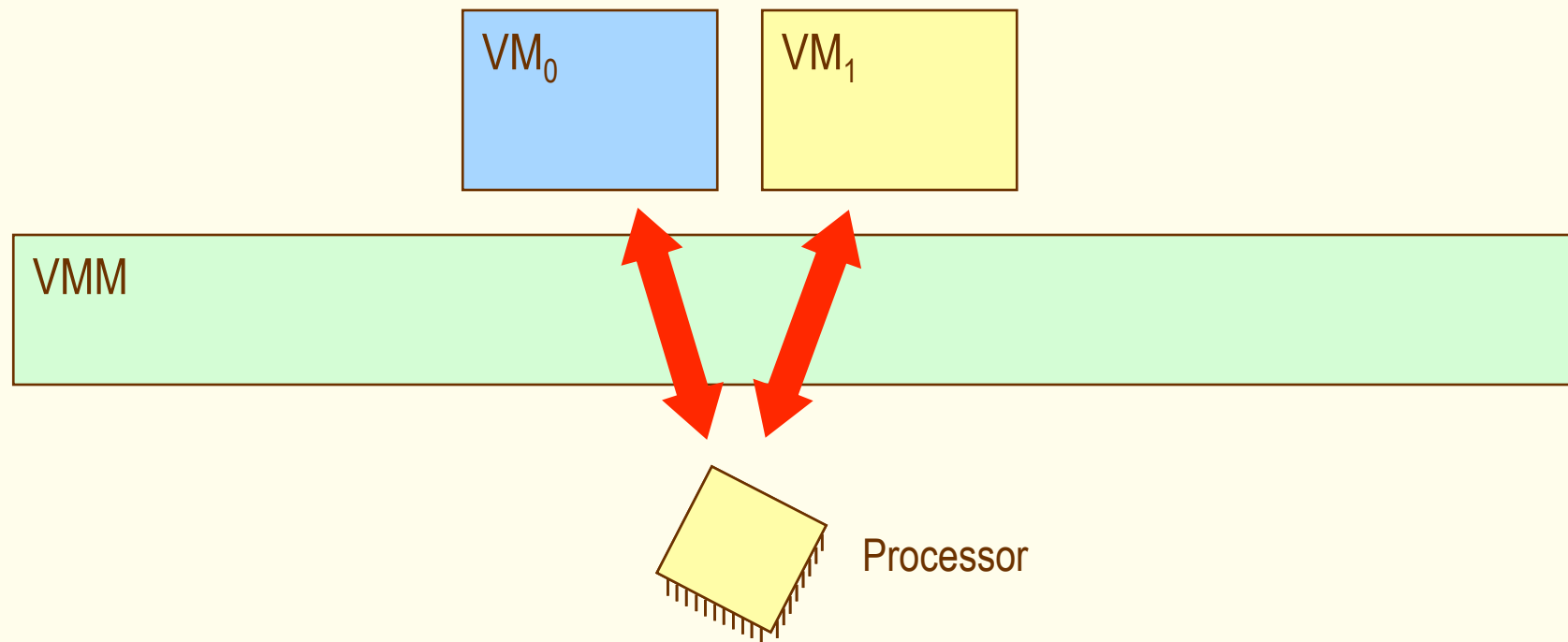
# Base VMM Requirements

---

- ❑ **A VMM must be able to:**
  - Protect itself from guest software
  - Isolate guest software stacks (OS + Apps) from one another
  - Present a (virtual) platform interface to guest software
- ❑ **To achieve this, VMM must control access to:**
  - CPUs, Memory and I/O Devices
- ❑ **Ways that a VMM can share resources between VMs**
  - Time multiplexing
  - Resource partitioning
  - Mediating hardware interfaces

# (1) Time Multiplexing

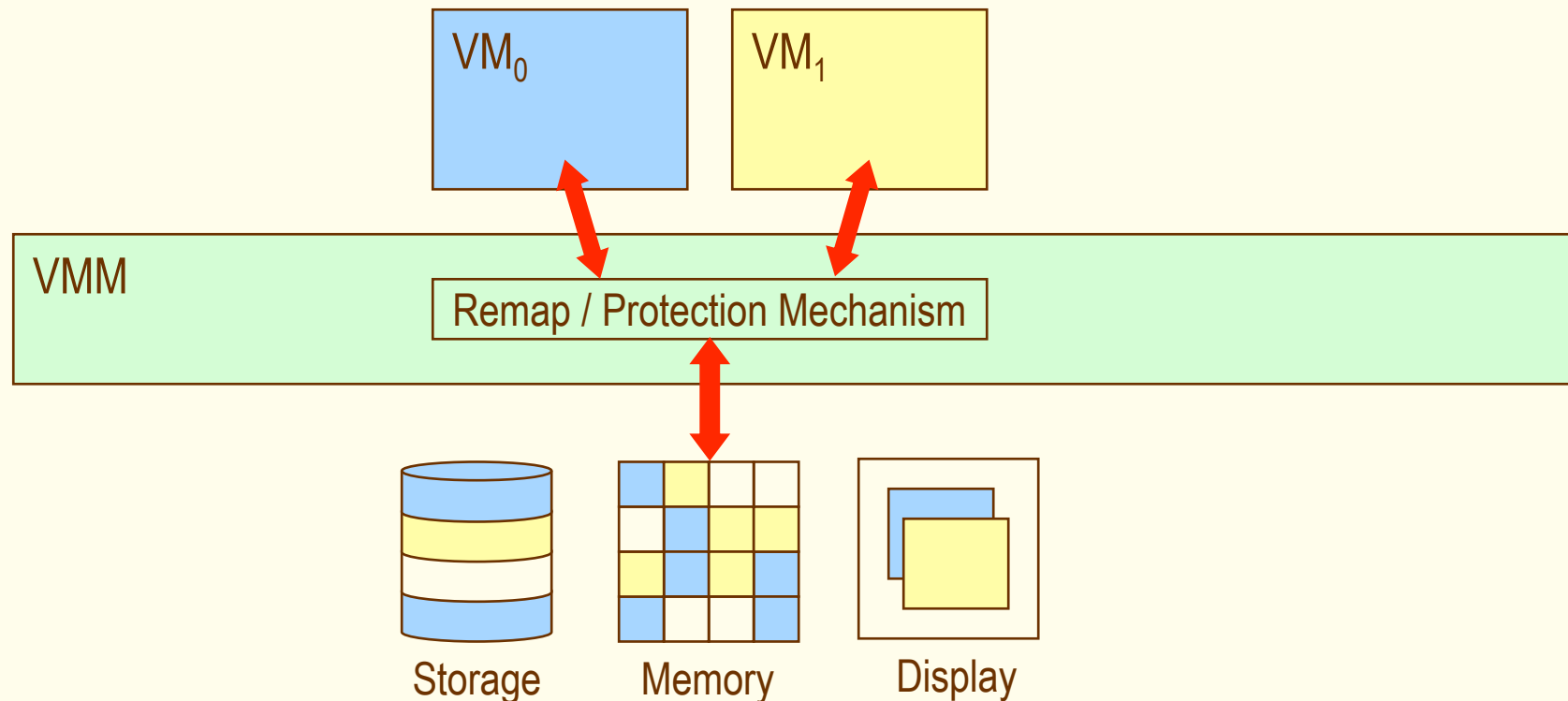
---



- ❑ **VM is allowed direct access to resource for a period of time before being context switched to another VM (e.g., CPU resource)**
- ❑ **Devil is in the details (will examine via a case study in later foils)**

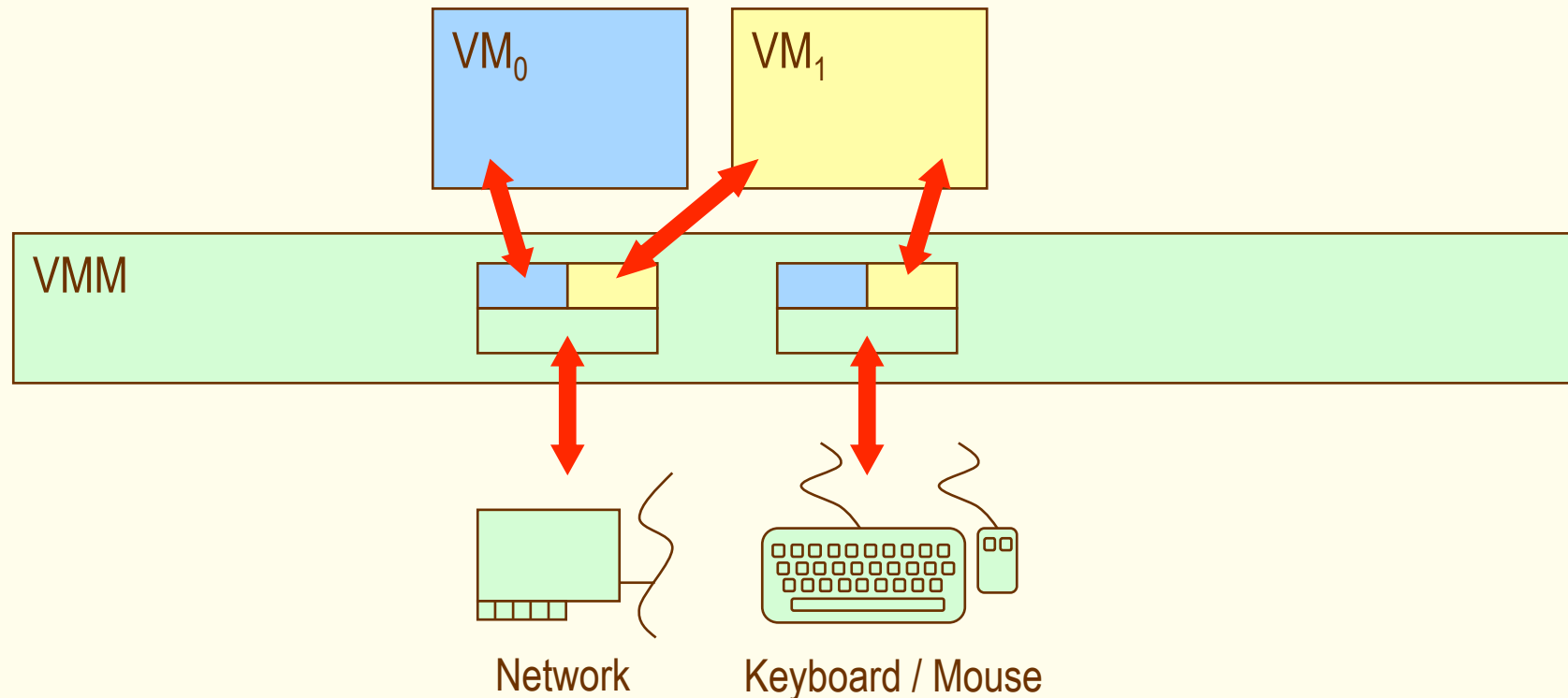
## (2) Resource Partitioning

---



- **VMM allocates “ownership” of phys resources to VMs**
  - Typically involves some remapping and protection mechanism
  - Examples: **physical memory**, **disk partitions**, **graphical display**

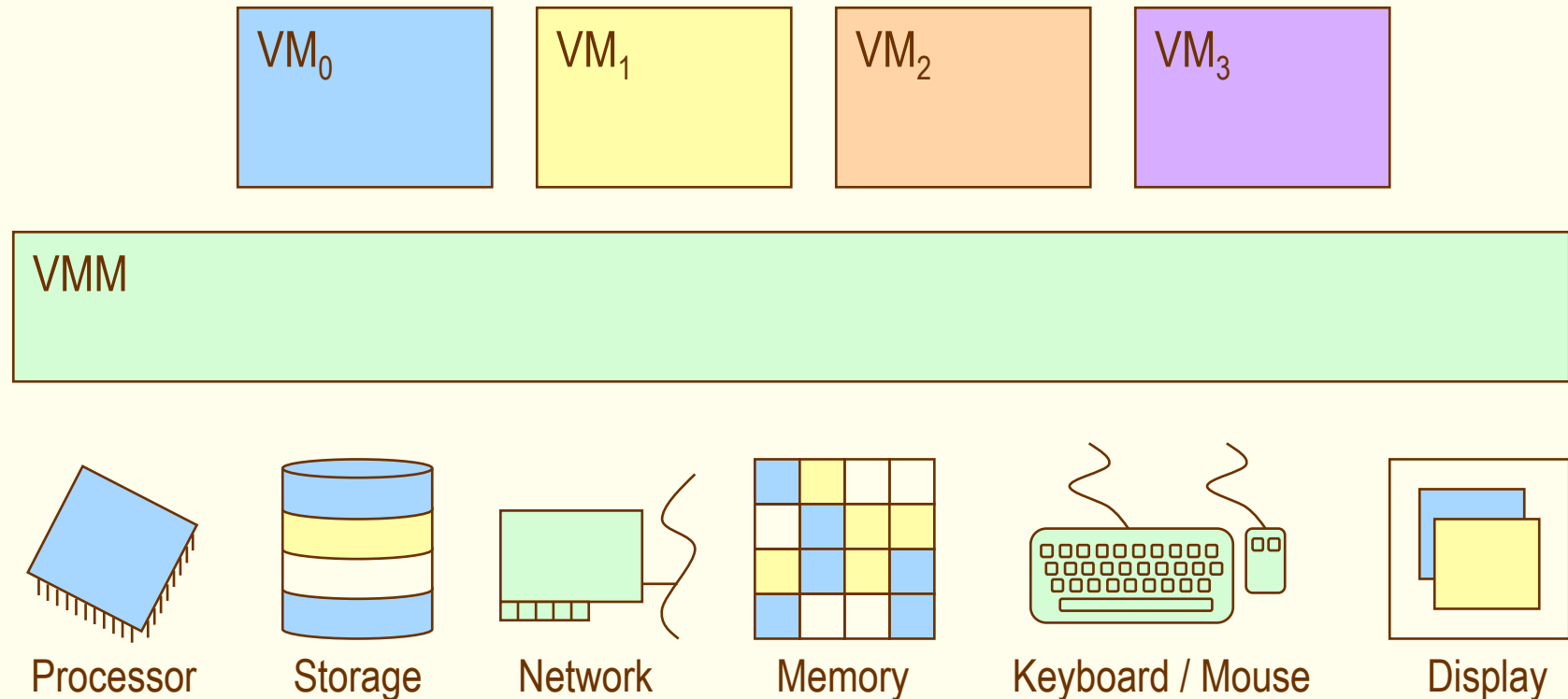
## (3) Mediating Hardware Interfaces



- **VMM retains direct ownership of physical resource**
  - VMM hosts device driver as well as a virtualized device interface
  - Virtual interface can be same as or different than physical device

# Putting it all Together...

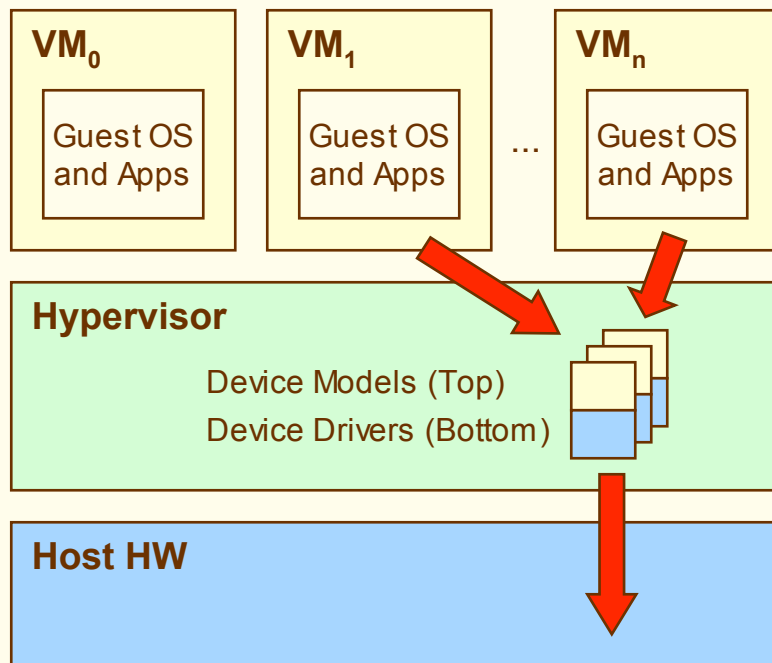
---



- **VMM applies all 3 sharing methods, as needed, to create illusion of platform ownership to each guest OS**

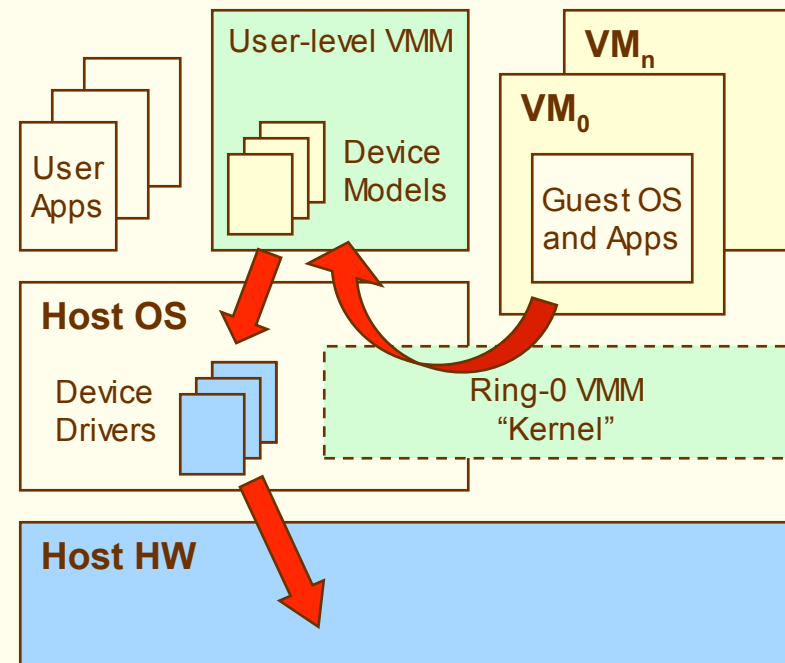
# Some VMM Architecture Options

Hypervisor Architecture



- ❑ Hypervisor architecture provides its own device drivers and services

Hosted Architecture



- ❑ Hosted architecture leverages device drivers and services of a "host OS"

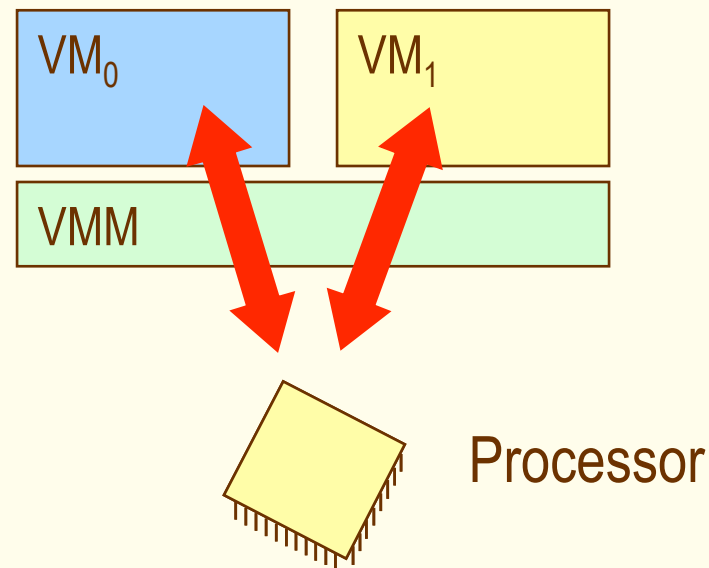


# **System Virtualization Case Studies**

## **Processor Virtualization**

# CPU Virtualization: General Principles

---



- ❑ **To virtualize a CPU, a VMM must retain control over:**
  - Accesses to privileged state (control regs, debug regs, etc.)
  - Exceptions (page faults, machine-check exceptions, etc.)
  - Interrupts and interrupt masking
  - Address translation (via page tables)
  - CPU access to I/O (via I/O ports or MMIO)

# CPU Control via “Ring Deprivileging”

---

## □ Ring Deprivileging Defined:

- Guest OS kernel runs in a less privileged ring than usual (i.e., above ring 0)
- VMM runs in the most privileged ring 0

## □ Goal of ring deprivileging is to prevent guest OS from:

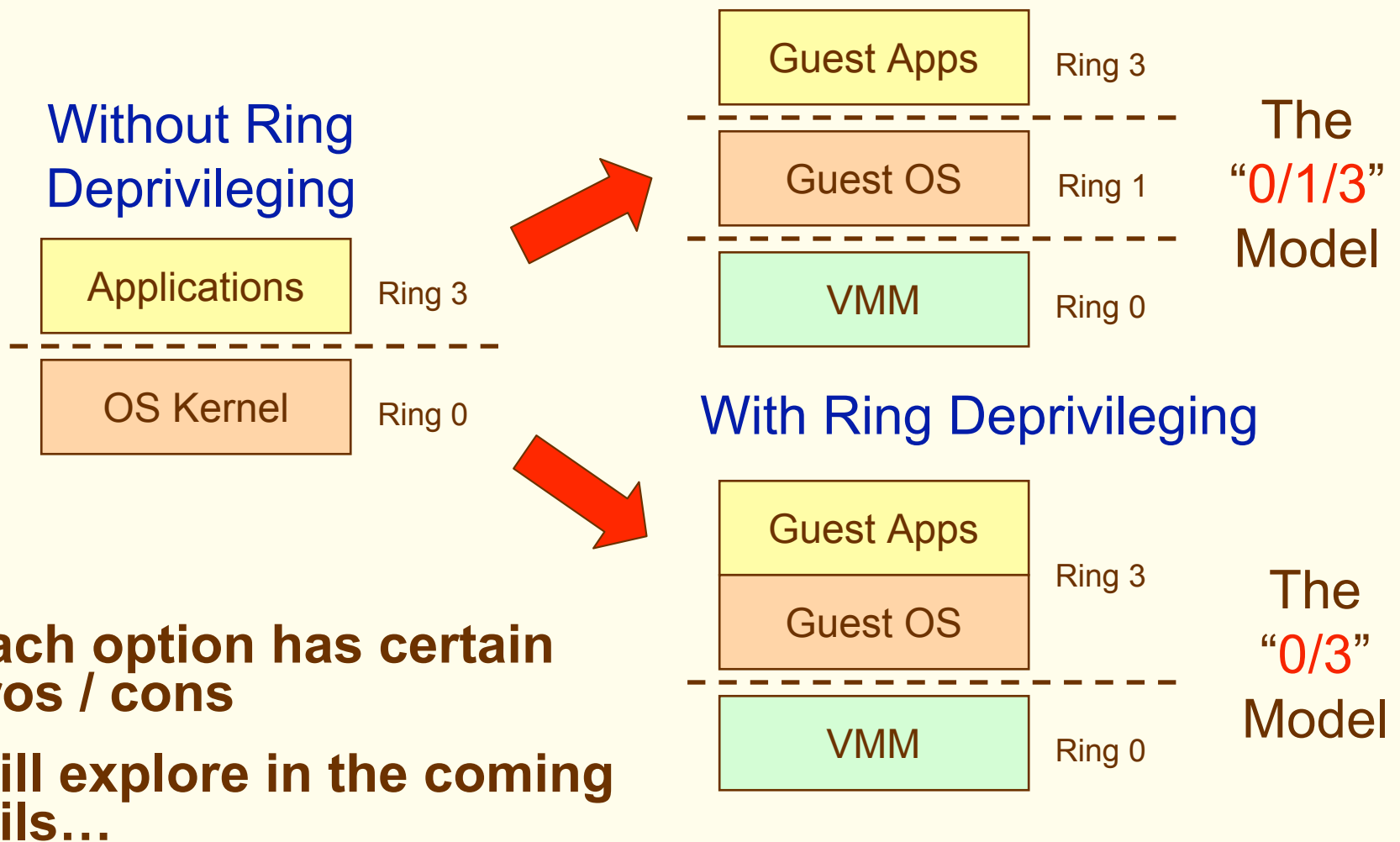
- Accessing privileged instructions / state
- Modifying VMM code and data

# Case Study: IA-32 CPU Virtualization

---

- ❑ **IA-32 Provides 4 Privilege Levels (Rings)**
  
- ❑ **Segment-based Protections**
  - Distinguish between all 4 rings
  
- ❑ **Page-based Protections**
  - Separate only User and Supervisor modes
  - User mode: Code running in ring 3
  - Supervisor mode: Code running in rings 0, 1, or 2

# Ring Deprivileging: Some Options

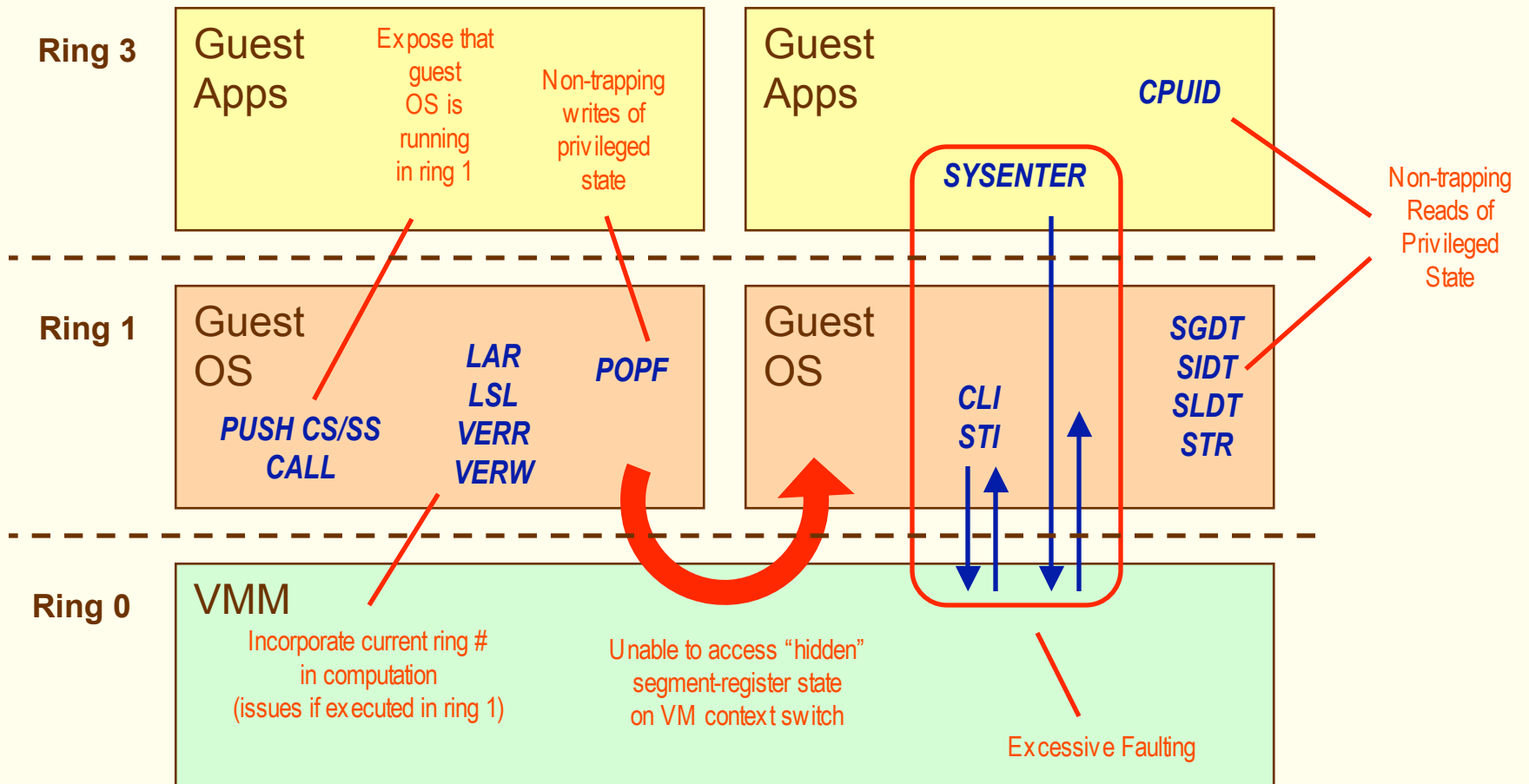


# Ring Compression

---

- ❑ **For the case of the 0/3 Model:**
  - Guest OS and Apps run in the same ring (3)
  - Lose ring protections between guest OS / Apps
  - Two rings are “compressed” into one
  
- ❑ **For the case of the 0/1/3 Model:**
  - No ring compression, but...
  - Can't use paging to protect VMM from guest OS
  - VMM forced to use segment-based protections
  
- ❑ **The following foils assume 0/1/3 Model...**

# IA-32 Virtualization Holes




# Addressing IA-32 “Virtualization Holes”

---

## ❑ Method 1: Paravirtualization Techniques

- Modify guest OS to work around virtualization holes
- Requires ability to modify guest-OS source code

Software-only  
Methods



## ❑ Method 2: Binary Translation or Patching

- Modify guest OS binaries “on-the-fly”
- Source code not required, but introduces new complexities
- E.g., self-modifying code, translation caching, etc.
- Some excessive trapping remains (e.g., SYSENTER case)

## ❑ Method 3: Change Processor ISA

- Re-architect instruction set to close virtualization holes by design
- Example: New VT-x features for IA-32 processors...

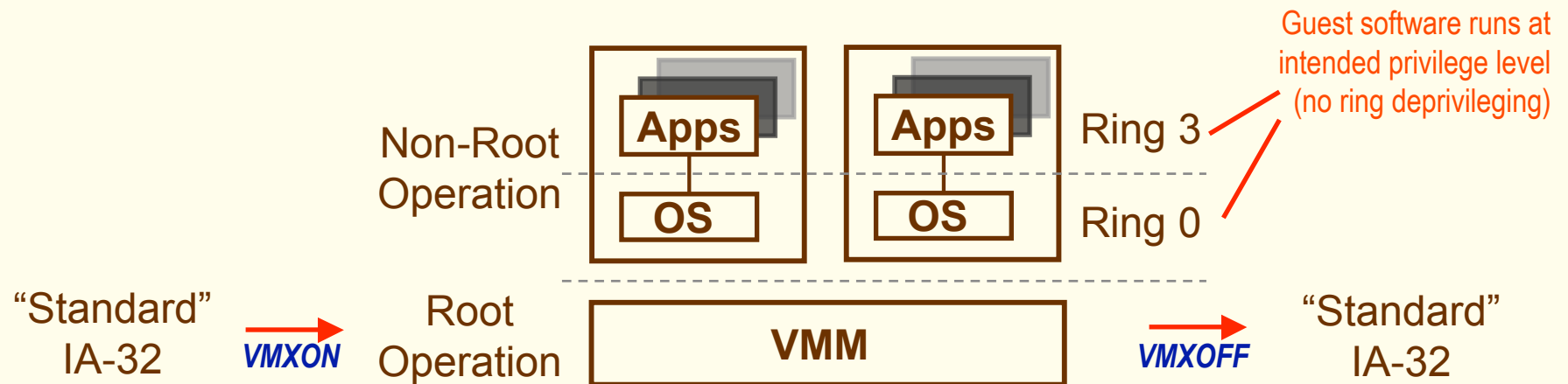


# Case Study: IA-32 Virtualization w/ VT-x

---

- ❑ **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



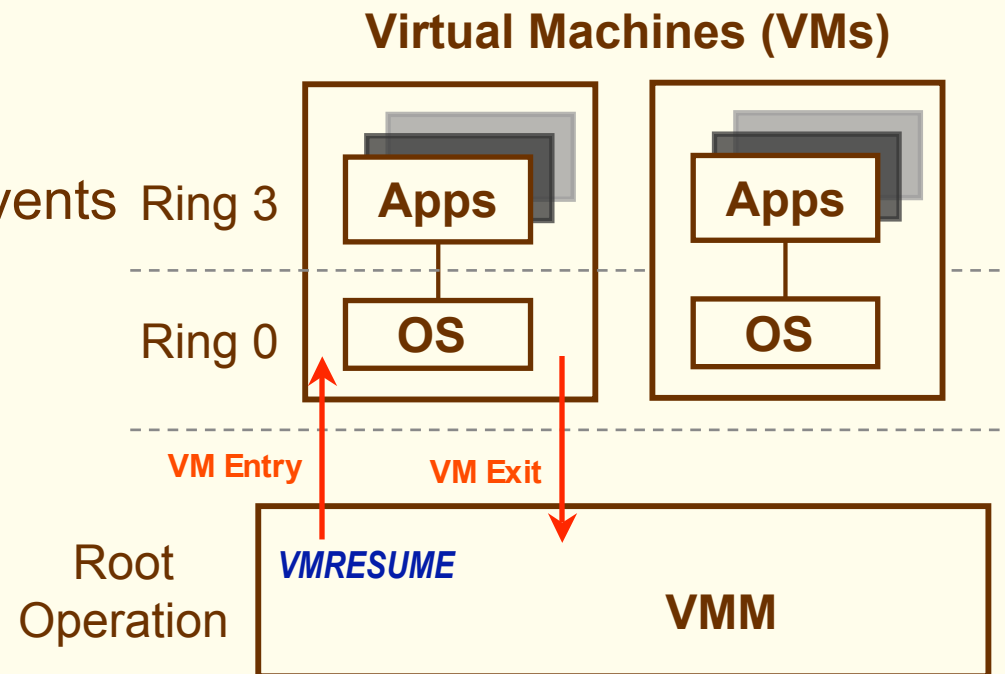
# VT-x Transitions: VM Entry and VM Exit

## □ 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

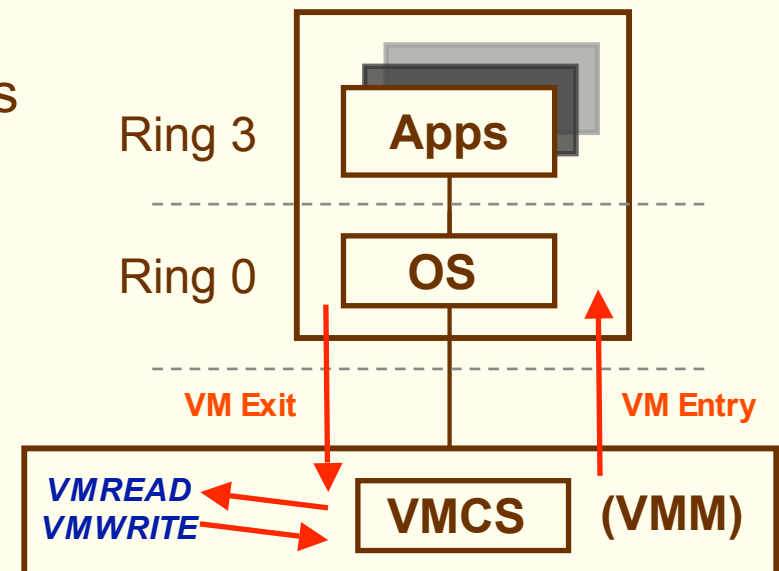
## □ VM Exit

- 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, MSRn
  - Sensitive Ops: CPUID, HLT, etc.
  - Paging events: #PF, INVLPG
  - Interrupts and Exceptions
- ❑ **Other optimizations:**
  - Bitmaps, shadow registers, etc.



# The VM Control Structure (VMCS)

---

<b>VM-execution controls</b>	Determines what operations cause VM exits	CR0, CR3, CR4, Exceptions, IO Ports, Interrupts, Pin Events, etc.
<b>Guest -state area</b>	Saved on VM exits Reloaded on VM entry	EIP, ESP, EFLAGS, IDTR, Segment Regs, Exit info, etc.
<b>Host -state area</b>	Loaded on VM exits	CR3, EIP set to monitor entry point, EFLAGS hardcoded, etc.
<b>VM-exit controls</b>	Determines which state to save, load, how to transition	Example: MSR save -load list
<b>VM-entry controls</b>	Determines which state to load, how to transition	Including 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)

# Some Example VM-Exit Optimizations (2)

---

- **Time-Stamp Counter (TSC) Offsets**
  - VMM can supply an offset that is applied to reads of the TSC during guest execution
  - Eliminates VM exits on executions of **RDTSC** and reduces distortions of “virtual time”
- **External-interrupt Exiting**
  - External interrupts cause VM exits
  - Interrupts never masked; no need for VM exits on **CLI**, **STI**, etc.
- **Optimized Interrupt Delivery**
  - VMM can pend a “virtual interrupt” to a guest OS
  - VM exit occurs only when guest-OS interrupt window is open
  - Eliminates exits on most executions of **CLI**, **STI**, **IRET**, etc.

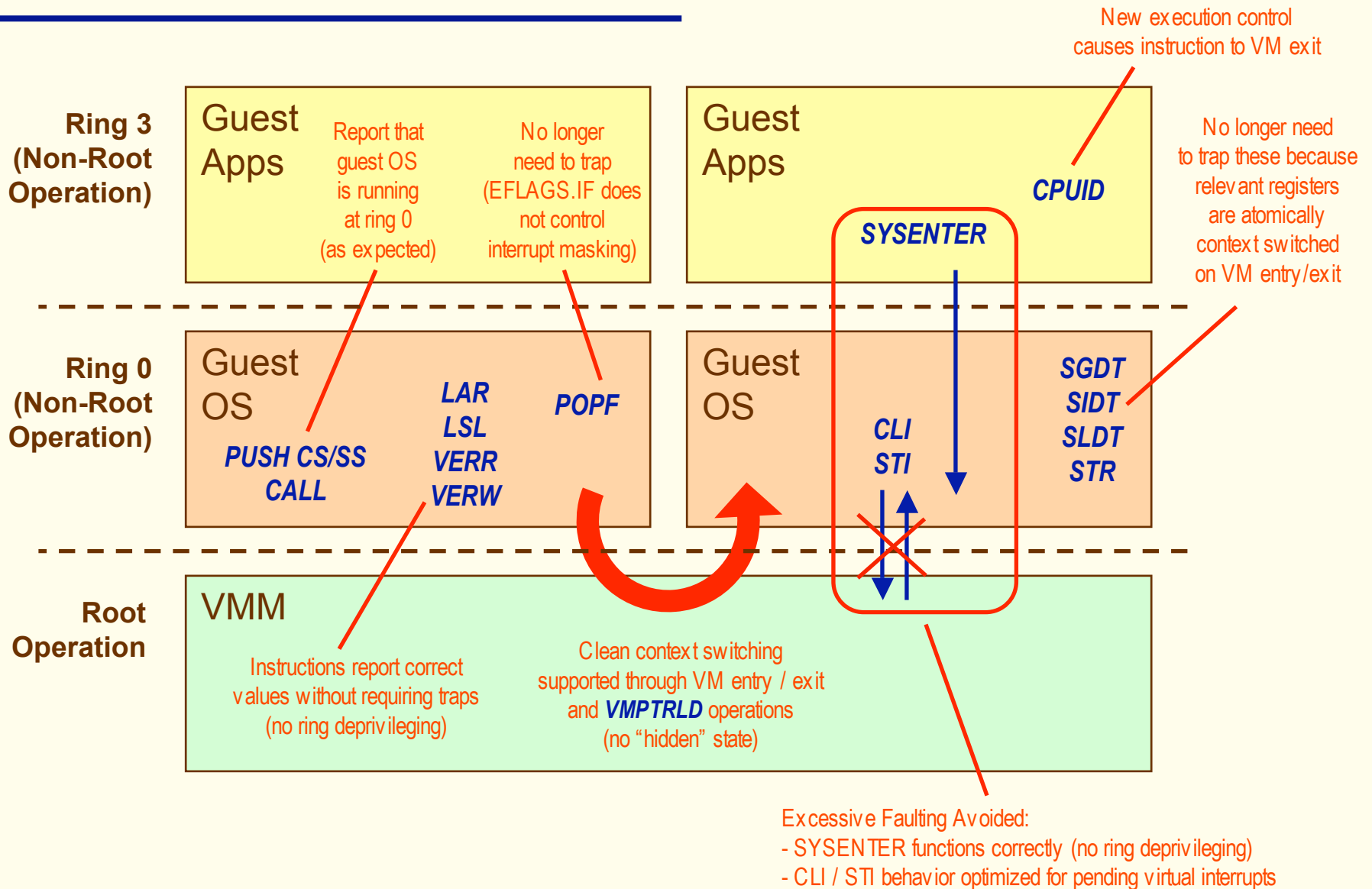


# VM Entry: Event Injection

---

- ❑ **Allows VMM to inject events on VM entry:**
  - External interrupts
  - NMI
  - Exceptions (e.g., page fault)
- ❑ **Injection occurs after all guest state is loaded**
- ❑ **Performs all the normal IDT checks, etc.**
- ❑ **Removes burden from VMM of emulating IDT, fault checking, etc.**

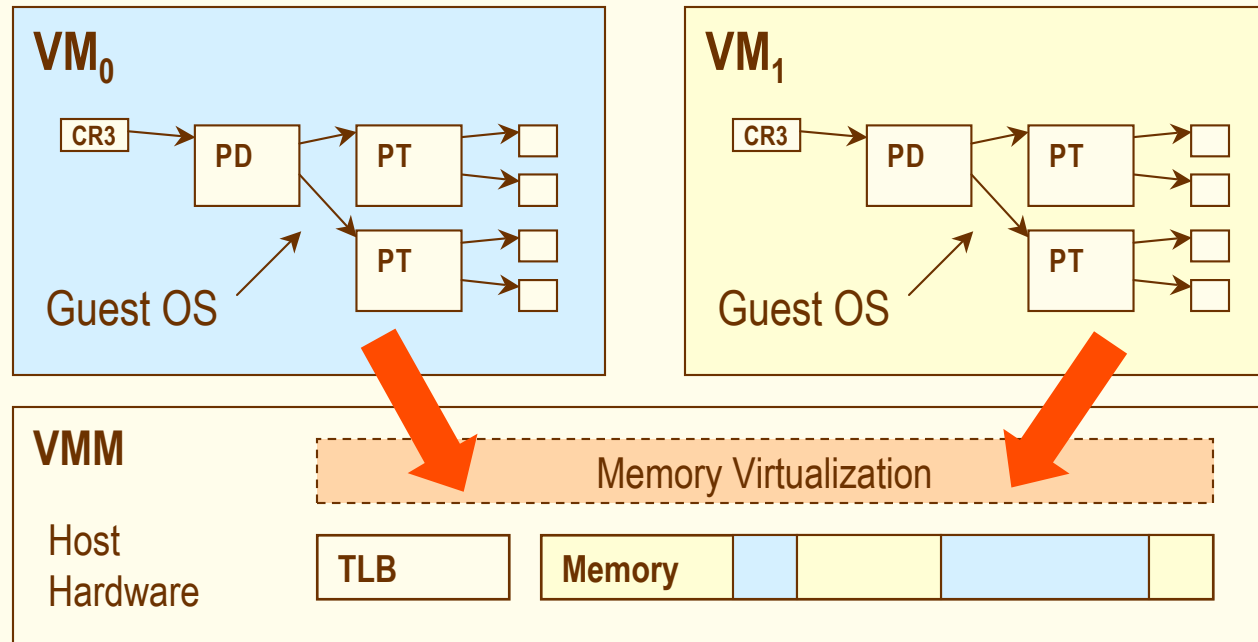
# How VT-x Closes Virtualization Holes



# **System Virtualization Case Studies**

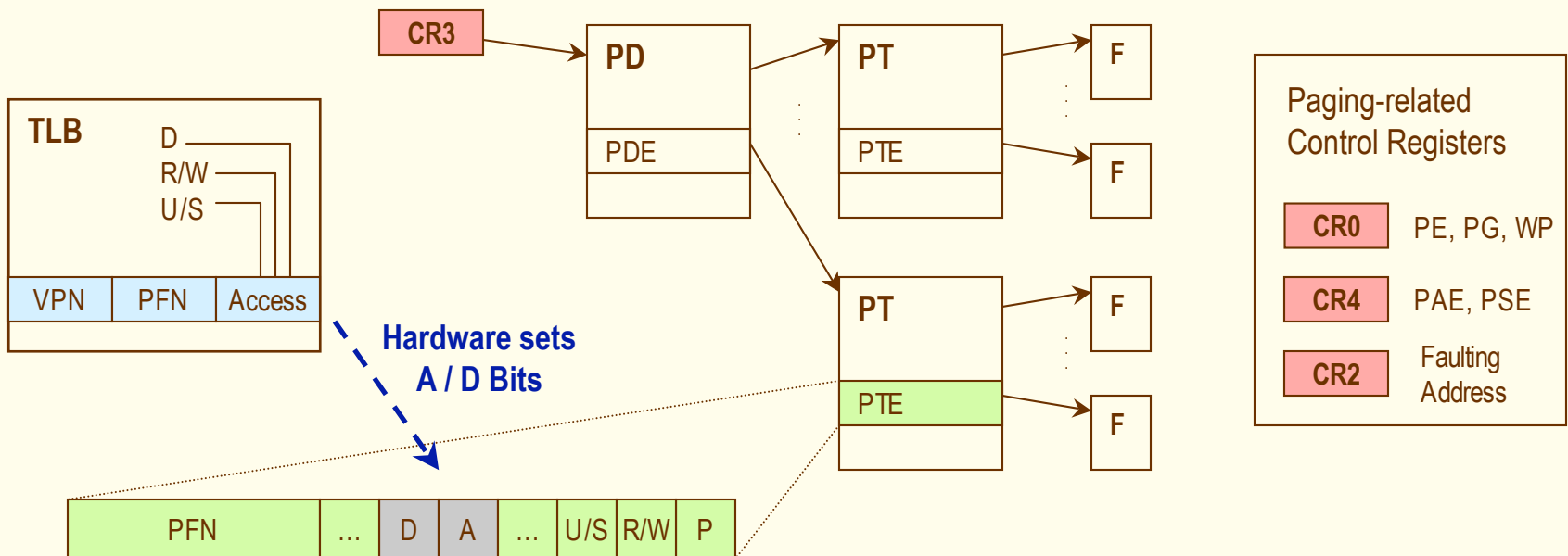
## **Memory Virtualization**

# 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



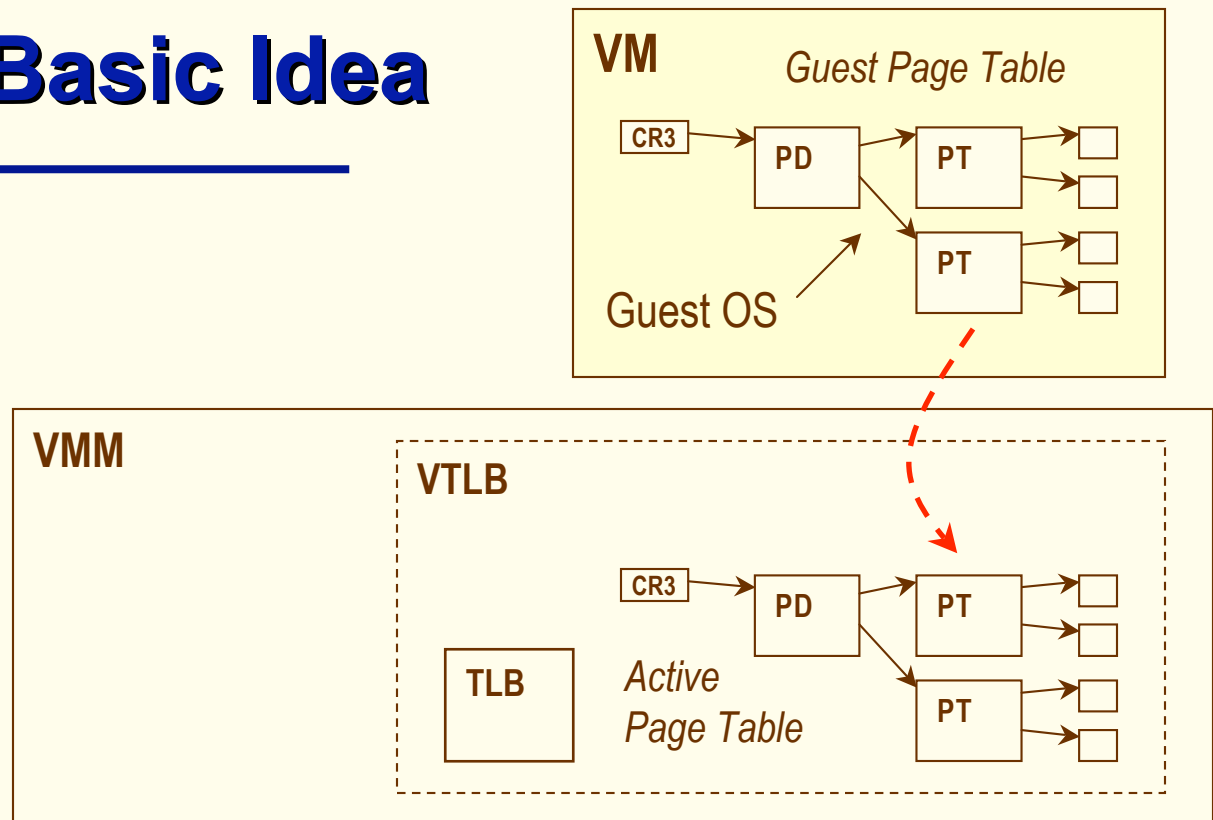
- ❑ **IA-32 defines a hierarchical page-table structure**
  - Defines linear-to-physical address translation
  - After page-table walk, page-table Entries (PTEs) are cached in a hardware TLB
- ❑ **IA-32 address translation configured via control registers (CR3, etc.)**
- ❑ **Invalidation of PTEs signaled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

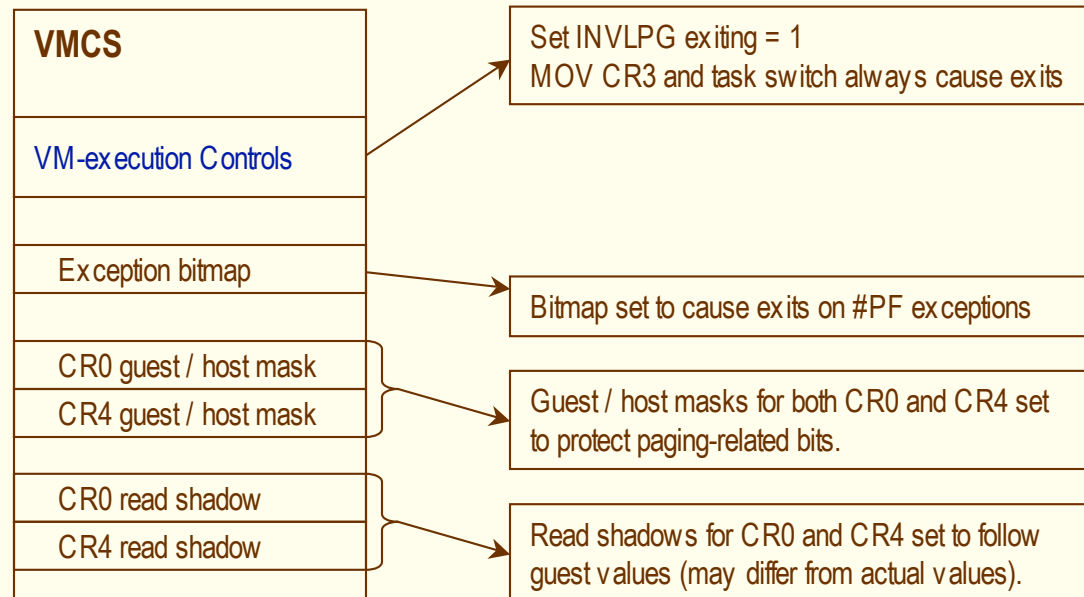
# Virtual TLB: Basic Idea



## □ 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

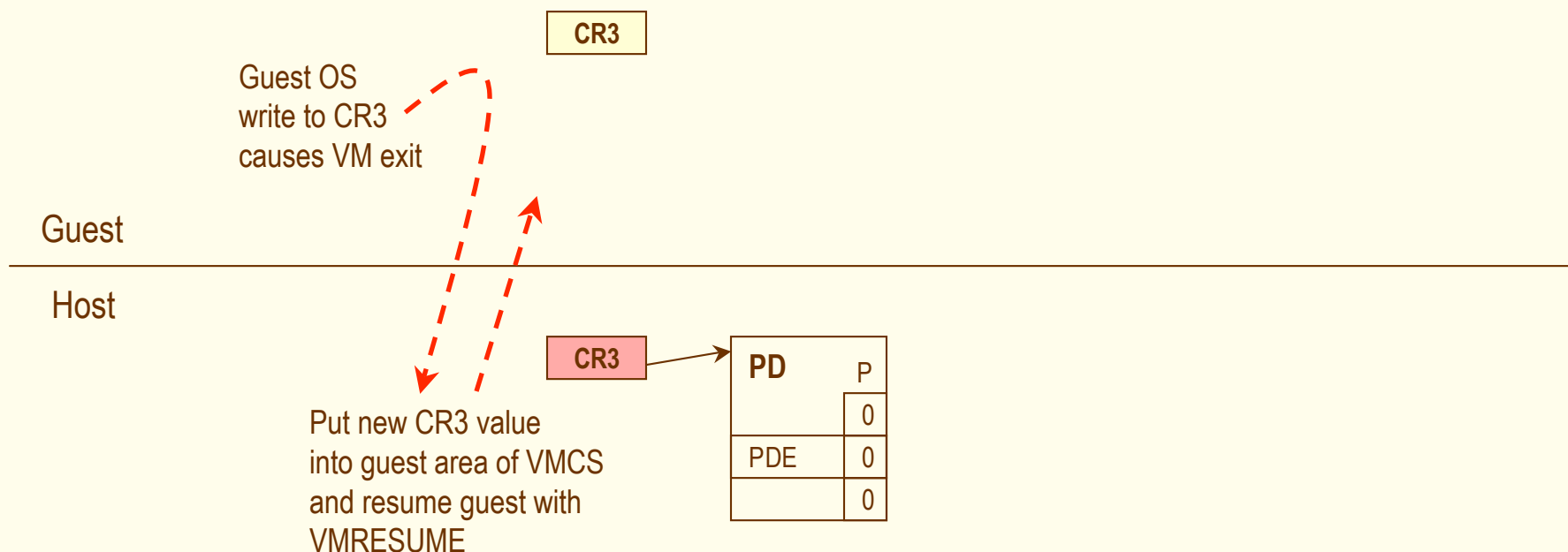
# Virtual TLB: VT-x Setup



- ❑ **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



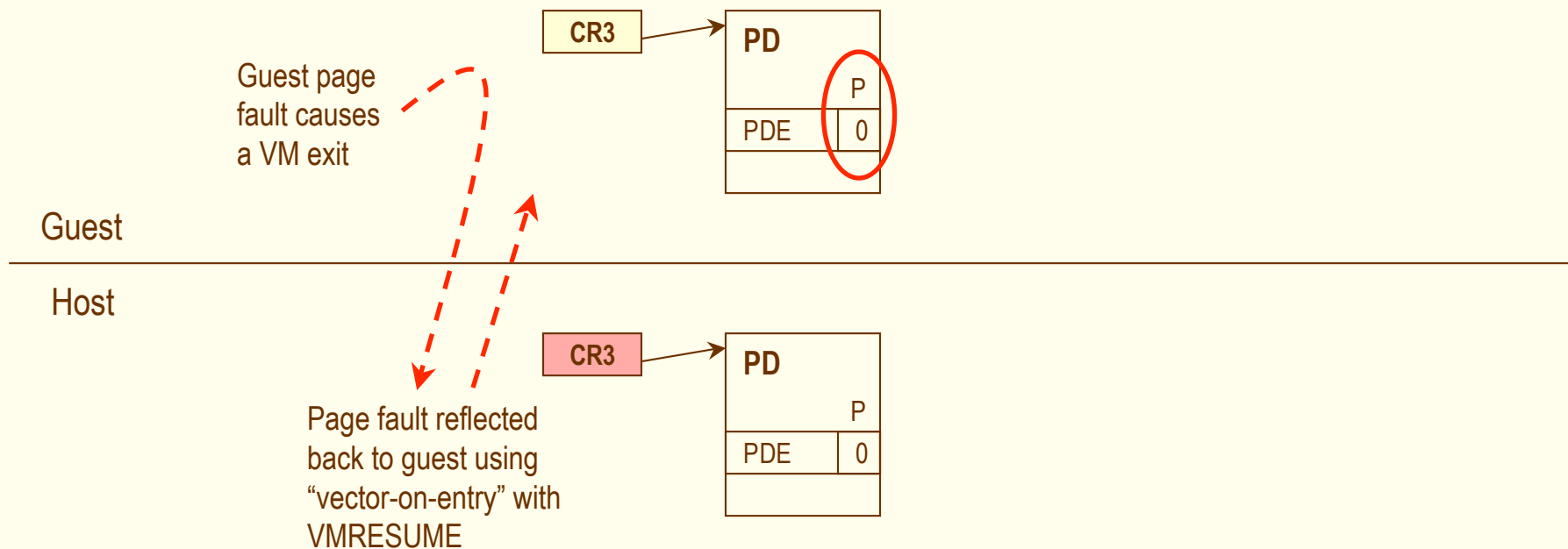
# Virtual TLB: Actions on CR3 Write



## ❑ 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

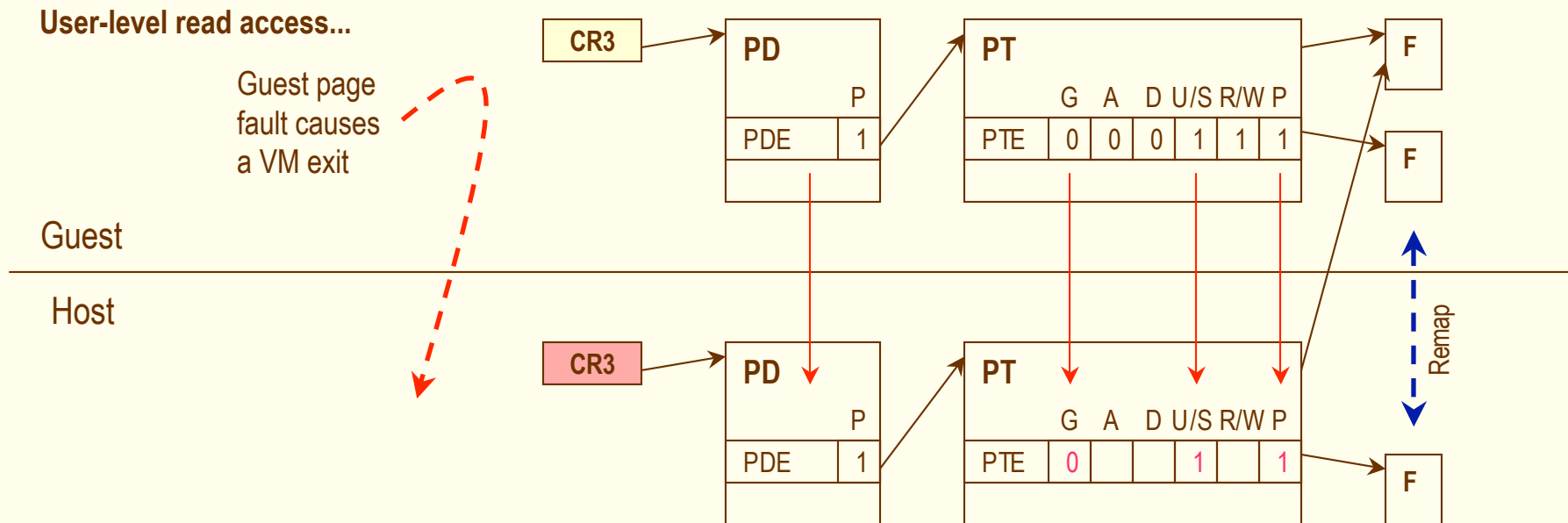
# Virtual TLB: Actions on a Page Fault



## □ VMM examines guest PT using faulting addr

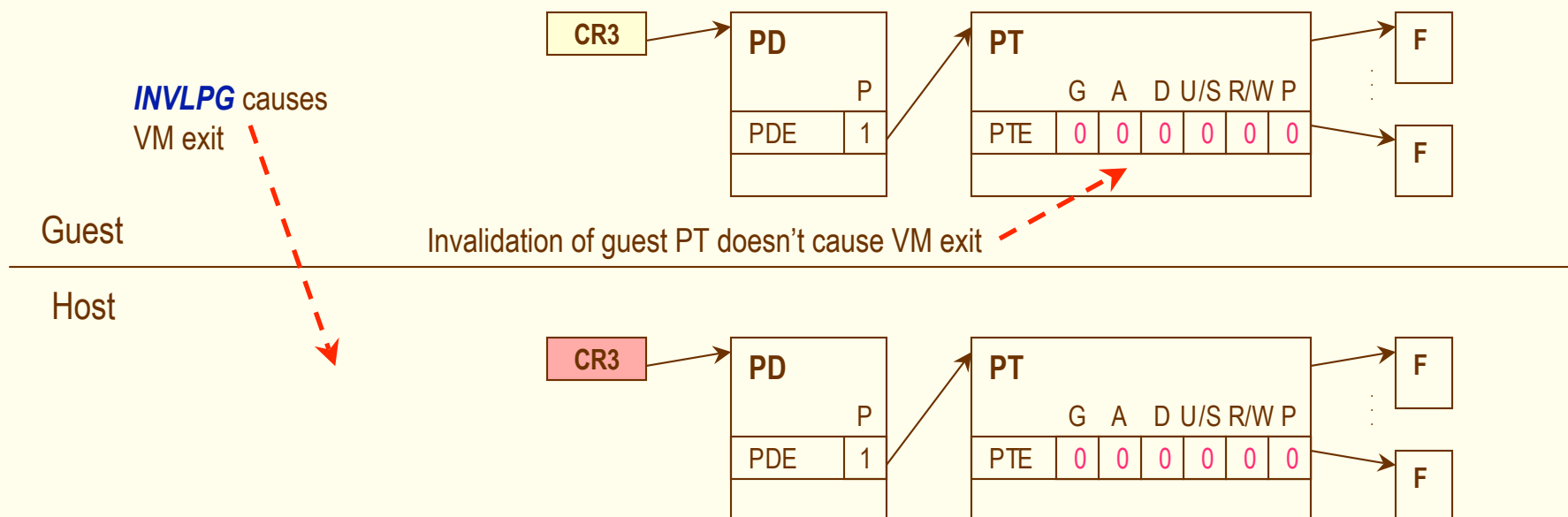
- If relevant PTE or PDE is invalid (P=0), then the #PF must be reflected to the guest OS.
- VMM configures VMCS for a "#PF vector-on-entry"
- Then resumes guest execution with a VMRESUME

# 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

# Virtual TLB: A few other details

---

## ❑ **MP considerations (TLB shutdown)**

- Each logical processor has its own VTLB (just as it has a TLB)
- TLB shutdown in software resolves down to cases shown previously (e.g., INVLPG)

## ❑ **Other Details**

- Accessed and Dirty Bits require special treatment (emulated through R/W and P page protections)
- Real-mode supported through an “identity” page table

## ❑ **Other Optimizations**

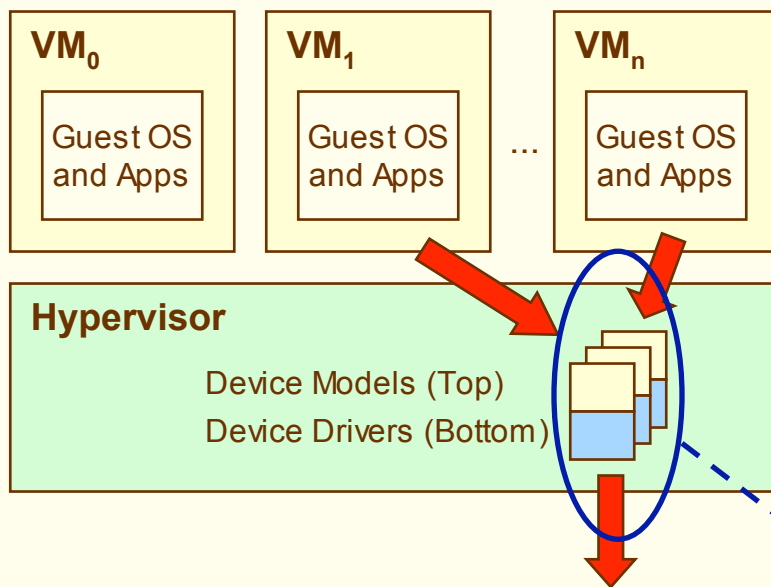
- Other VTLB refill policies possible (eager vs. lazy refill) with different trade-offs
- Possible to maintain multiple shadow page tables to reduce VTLB flush cost

# **System Virtualization Case Studies**

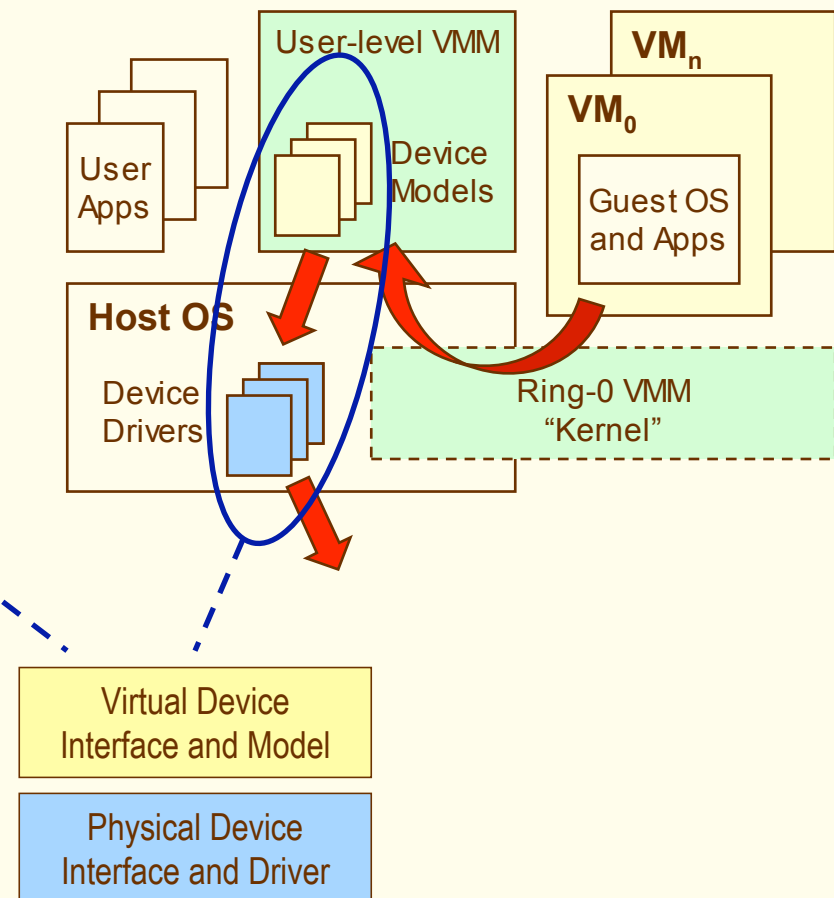
## **IO-Device Virtualization**

# IO Virtualization: General Principles

Hypervisor Architecture

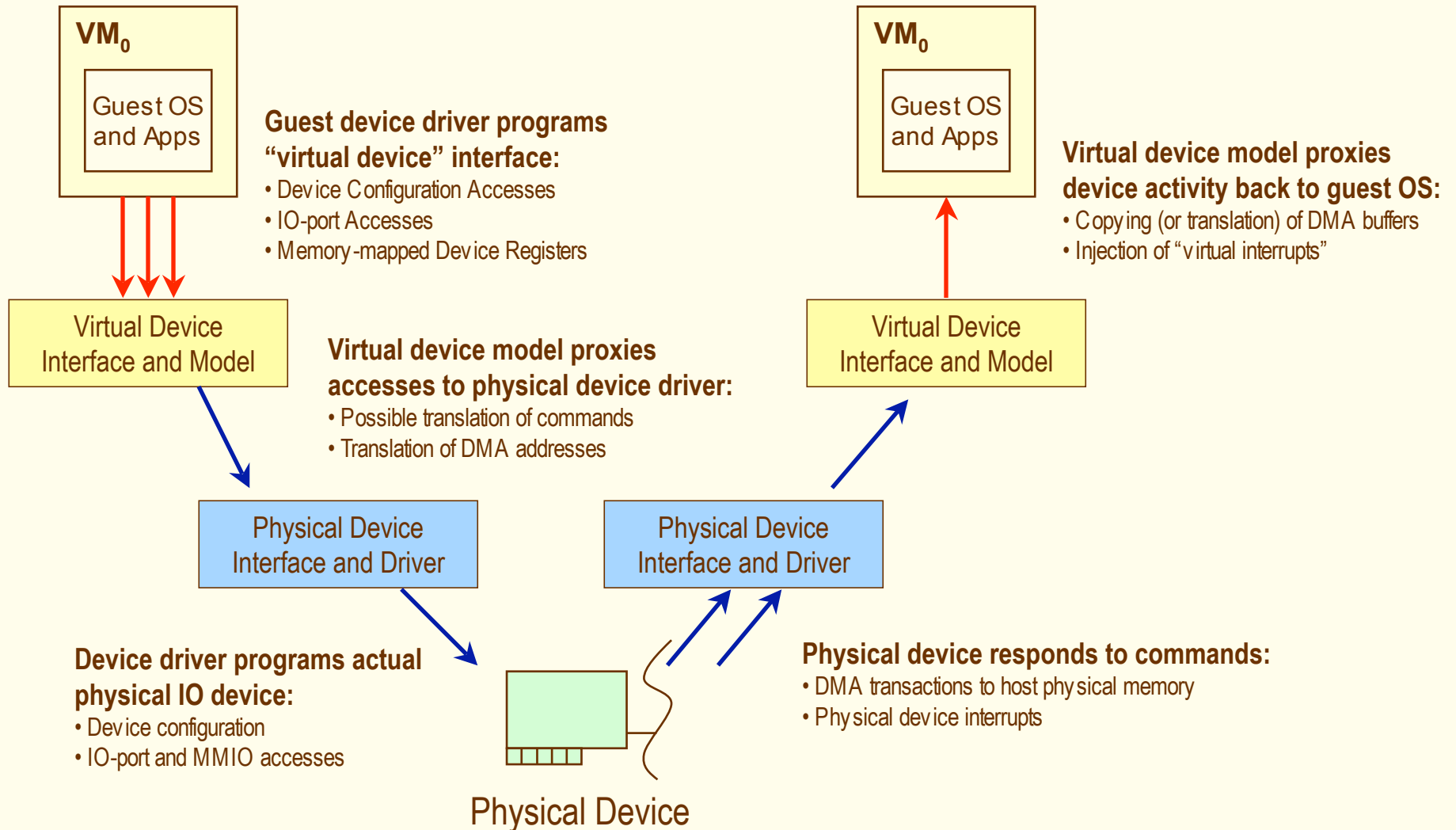


Hosted Architecture



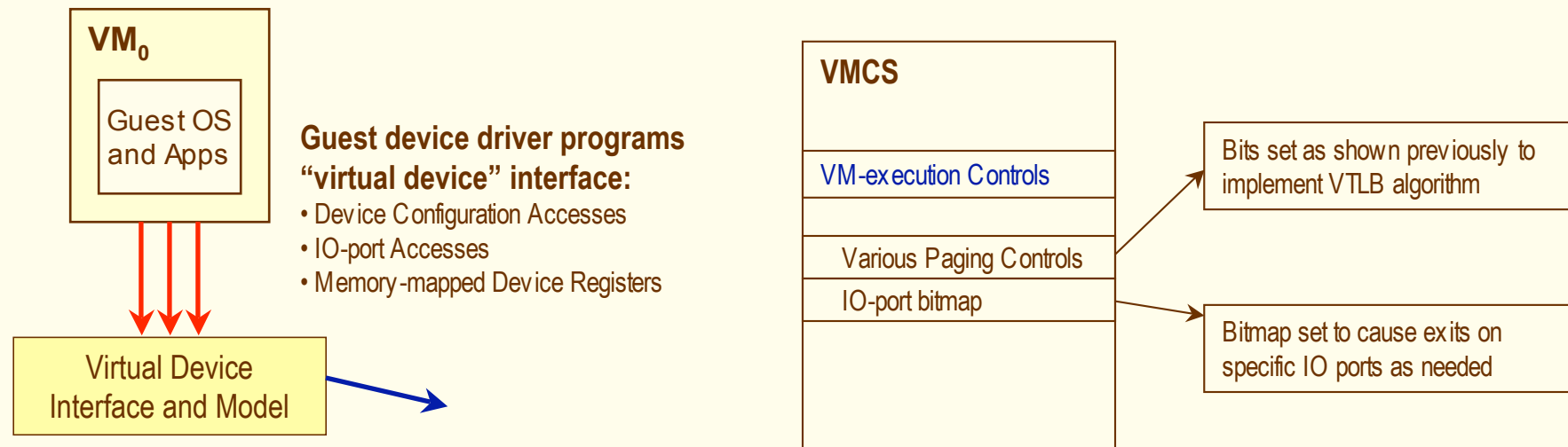
- ❑ **Virtual device model presents interface to guest operating system**
- ❑ **Physical device driver programs and responds to actual device hardware**

# Virtual and Physical Device Interfaces



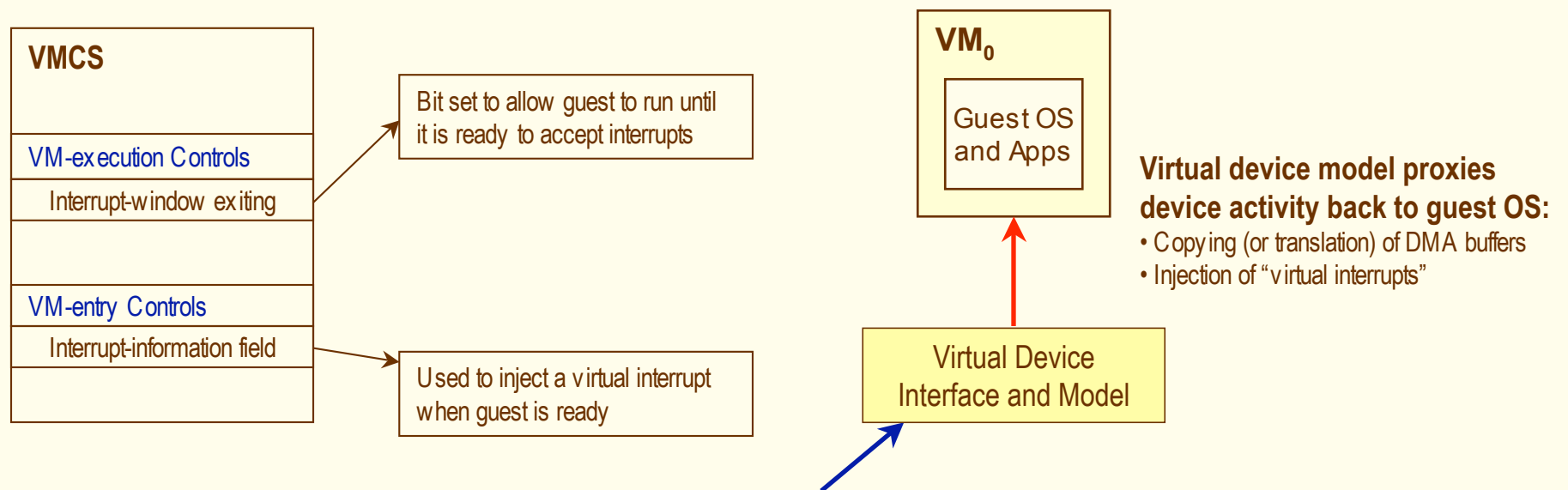


# Case Study: IO Virtualization with VT-x



- ❑ **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

# Summary and Wrap-up

---

- ❑ **For more information on Intel® Virtualization Technology (VT):**
  - <http://www.intel.com/technology/computing/vptech/>
  
- ❑ **Questions?**