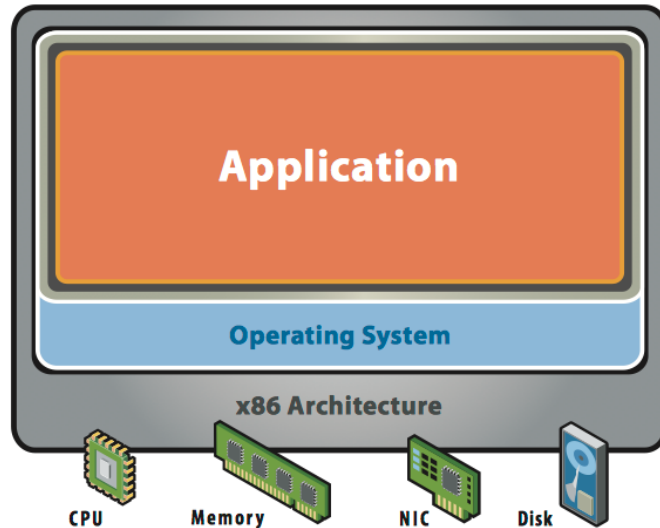


CS 5450

Virtualization

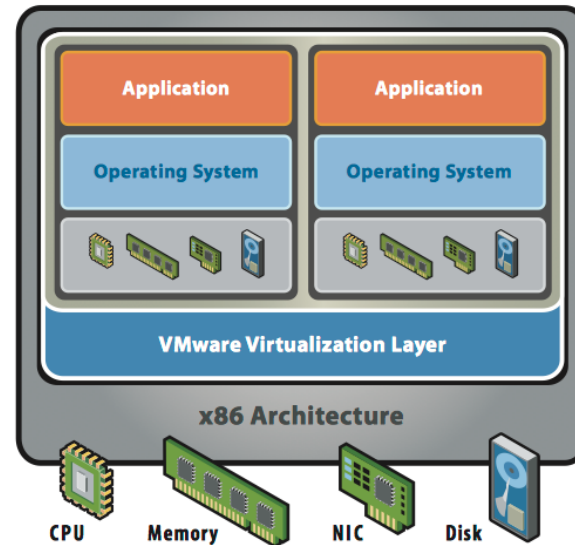
Vitaly Shmatikov

Virtualization



Before Virtualization:

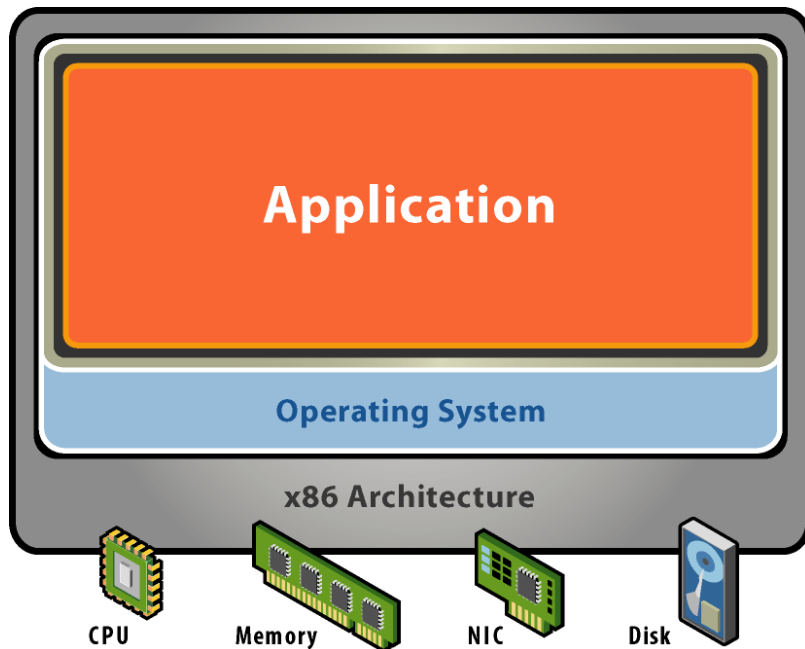
- Single OS image per machine
- Software and hardware tightly coupled
- Running multiple applications on same machine often creates conflict
- Underutilized resources
- Inflexible and costly infrastructure



After Virtualization:

- Hardware-independence of operating system and applications
- Virtual machines can be provisioned to any system
- Can manage OS and application as a single unit by encapsulating them into virtual machines

Physical Machine



◆ Physical hardware

- Processors, memory, chipset, I/O devices, etc.
- Resources often grossly underutilized

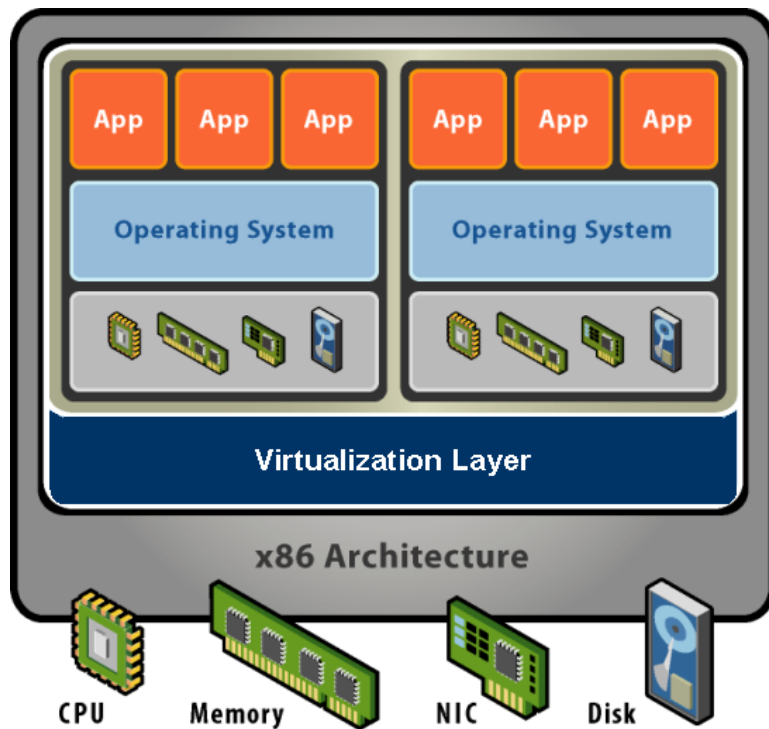
◆ Software

- Tightly coupled to physical hardware
- Single active OS instance
- OS controls hardware

OS Limitations

- ◆ OSes provide a way of virtualizing hardware resources among processes
- ◆ Helps isolate processes from one another, but does not provide a virtual machine to a user who may wish to run a different OS
- ◆ Having hardware resources managed by a single OS limits the flexibility of the system in terms of available software, security, and failure isolation

Virtual Machine



◆ Software abstraction

- Behaves like hardware
- Encapsulates all OS and application state

◆ Virtualization layer

- Extra level of indirection
- Decouples hardware, OS
- Enforces isolation
- Multiplexes physical hardware across VMs

Types of Virtualization

◆ Process virtualization

- **Language-level** Java, .NET, Smalltalk
- **OS-level** processes, Solaris Zones, BSD Jails, Virtuozzo
- **Cross-ISA emulation** Apple 68K-PPC-x86, Digital FX!32

◆ Device virtualization

- **Logical vs. physical** VLAN, VPN, NPIV, LUN, RAID

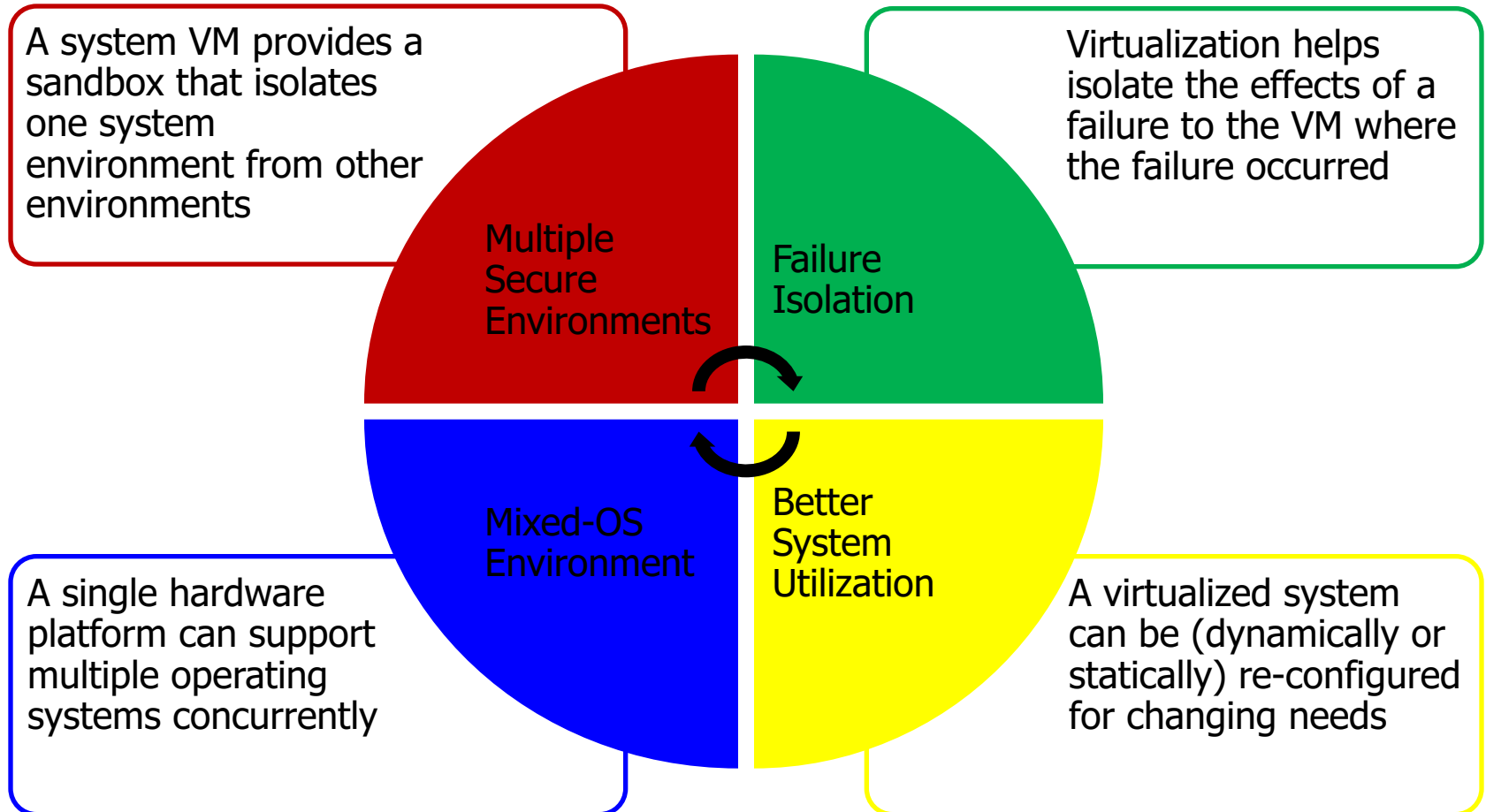
◆ System virtualization

- **"Hosted"** VMware Workstation, Microsoft VPC, Parallels
- **"Bare metal"** VMware ESX, Xen, Microsoft Hyper-V

Virtualization Properties

- ◆ Isolation of faults and performance
- ◆ Encapsulation of entire VM state
 - Enables snapshots and cloning of VMs
- ◆ Portability
 - Independent of physical hardware
 - Enables migration of live, running VMs
- ◆ Interposition
 - Transformations on instructions, memory, I/O
 - Enables transparent resource overcommitment, encryption, compression, replication ...

Benefits of Virtualization



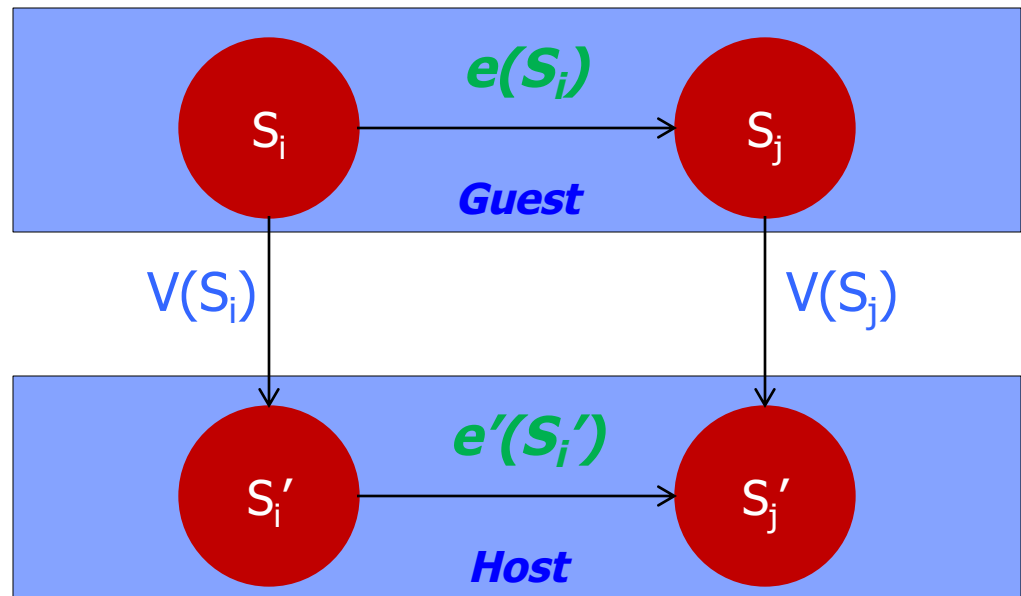
What Is Virtualization?

Informally, a virtualized system (or subsystem) is a mapping of its interface, and all resources visible through that interface, to the interface and resources of a real system

Formally, virtualization involves the construction of an isomorphism that maps a virtual **guest** system to a real **host** system (Popek and Goldberg 1974)

Function V maps the guest state to the host state

For a sequence of operations, e , that modifies a guest state, there is a corresponding e' in the host that performs an equivalent modification



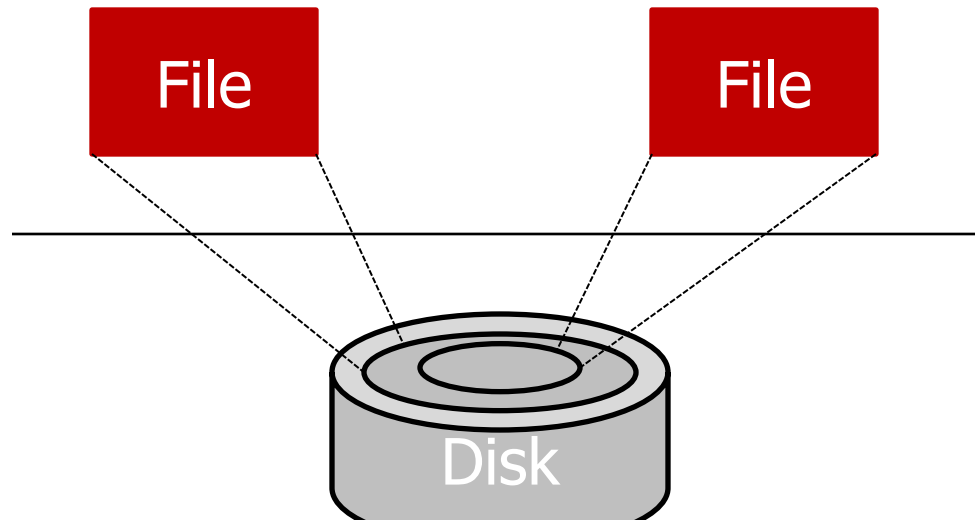
Virtual Machine Monitor

[Popek and Goldberg 1974]

A virtual machine is taken to be an *efficient, isolated duplicate* of the real machine. We explain these notions through the idea of a *virtual machine monitor* (VMM). See Figure 1. As a piece of software a VMM has three essential characteristics. First, the VMM provides an environment for programs which is essentially identical with the original machine; second, programs run in this environment show at worst only minor decreases in speed; and last, the VMM is in complete control of system resources.

Abstraction

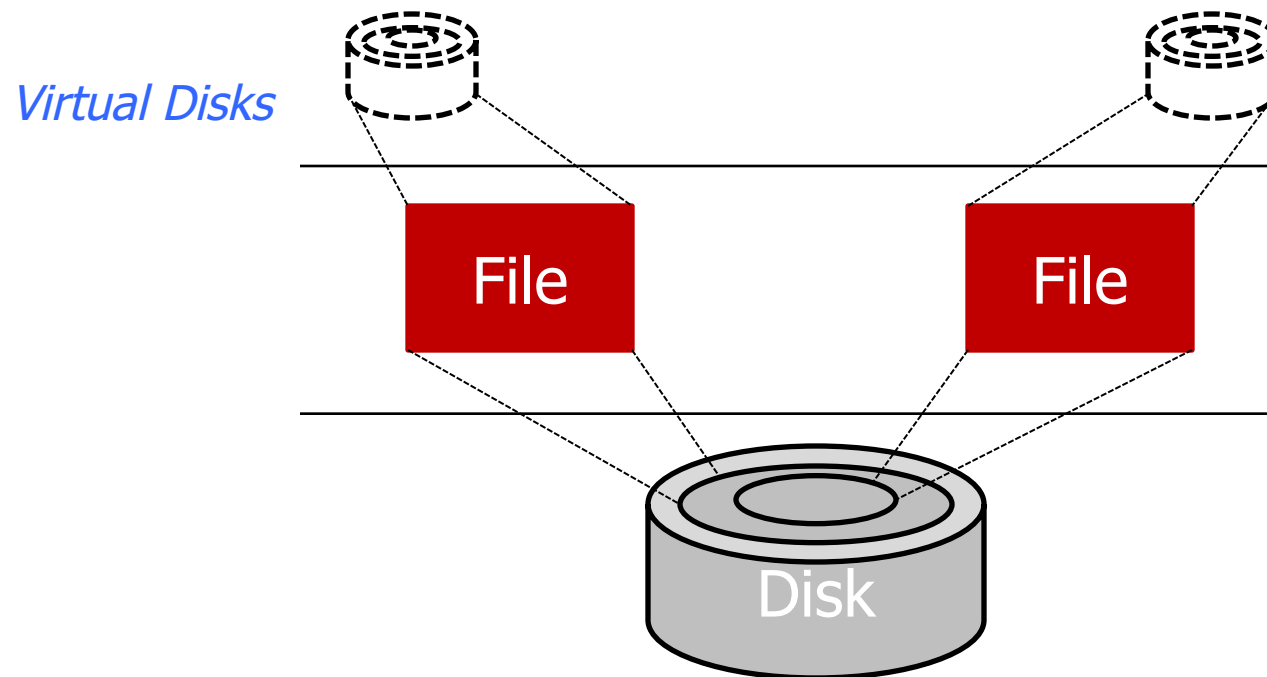
- ◆ The key to managing complexity in computer systems is their division into levels of **abstraction** separated by **well-defined interfaces**



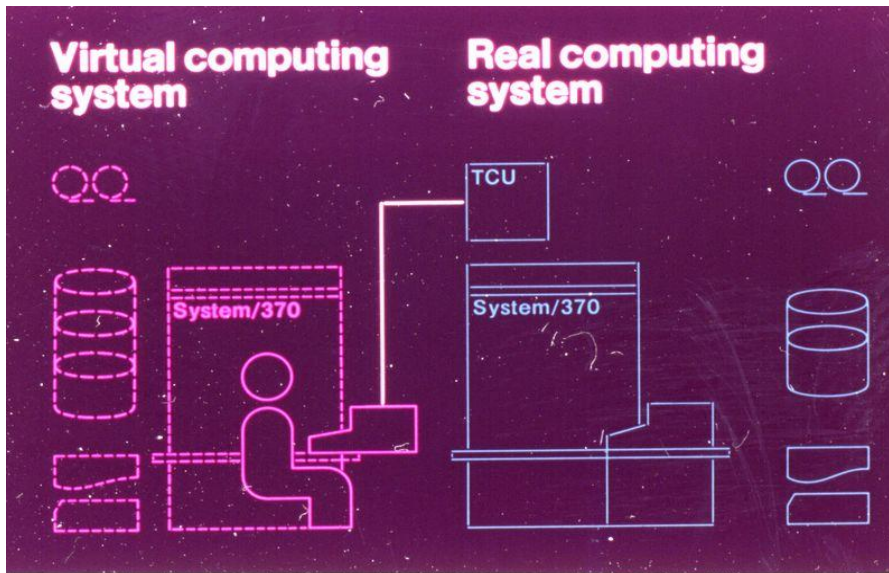
- Files are an abstraction of a disk
- A level of abstraction provides a simplified interface to underlying resources

Virtualization and Abstraction

- ◆ Virtualization uses abstraction but the level of detail is often the same as in underlying system



“Classic” Virtualization



From IBM VM/370 product announcement, *ca.* 1972

◆ Classical VMM

- IBM mainframes:
IBM S/360, IBM VM/370
- Co-designed proprietary hardware, OS, VMM
- “Trap and emulate” model

◆ Applications

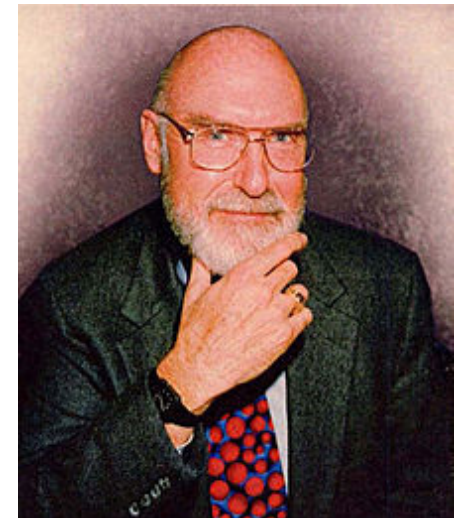
- Timeshare several single-user OS instances on expensive hardware
- Compatibility

IBM VM/370

◆ Robert Jay Creasy (1939-2005)

- Project leader of the first full virtualization hypervisor: IBM CP-40, a core component in the VM system

◆ First VM system: VM/370



Virtualization Renaissance

◆ Recent proliferation of VMs

- Considered exotic mainframe technology in 1990s
- Now pervasive in datacenters and clouds
- Huge commercial success

◆ Why?

- Introduction on commodity x86 hardware
- Ability to “do more with less” saves \$\$\$
- Innovative new capabilities
- Extremely versatile technology

Modern Virtualization Applications

◆ Server consolidation

- Convert underutilized servers to VMs
- Significant cost savings (equipment, space, power)
- Increasingly used for virtual desktops

◆ Simplified management

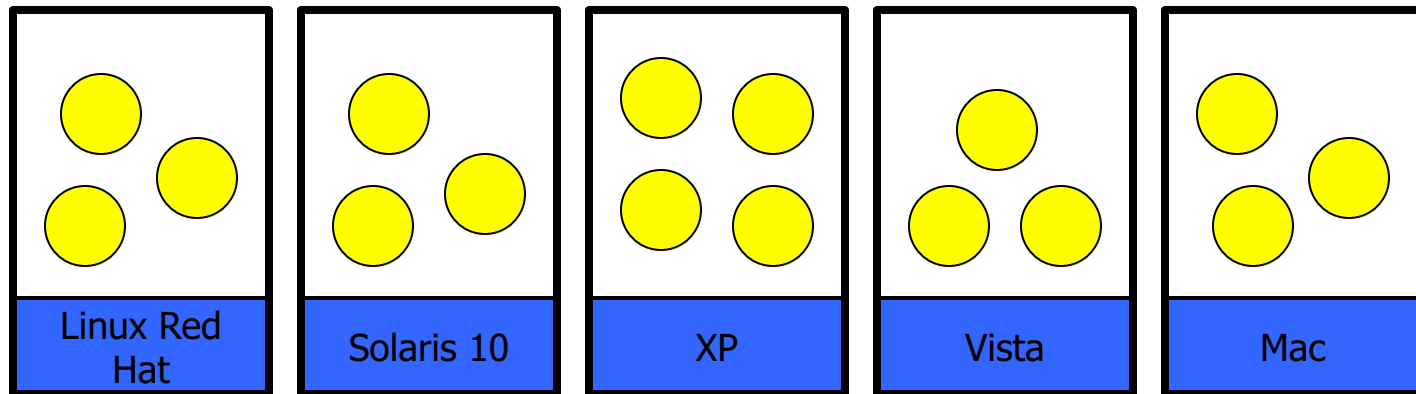
- Datacenter provisioning and monitoring
- Dynamic load balancing

◆ Improved availability

- Automatic restart, fault tolerance, disaster recovery

◆ Test and development

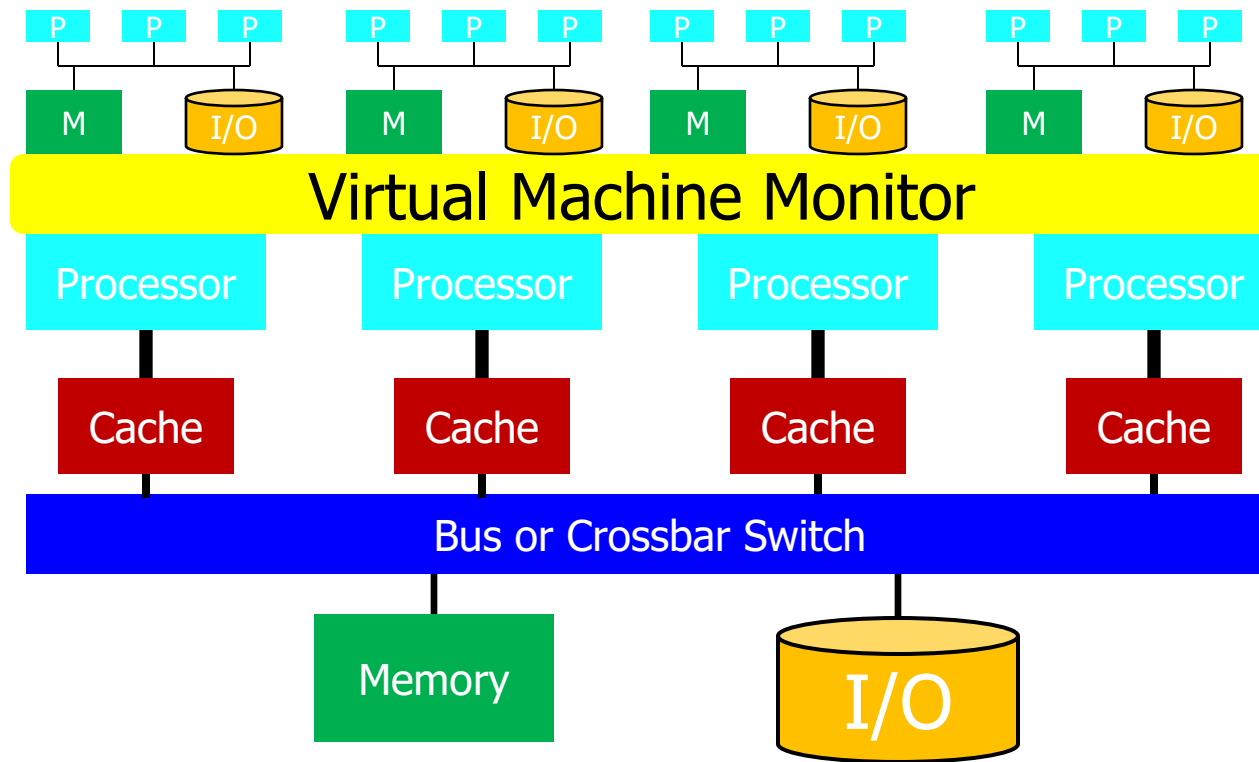
A Mixed OS Environment



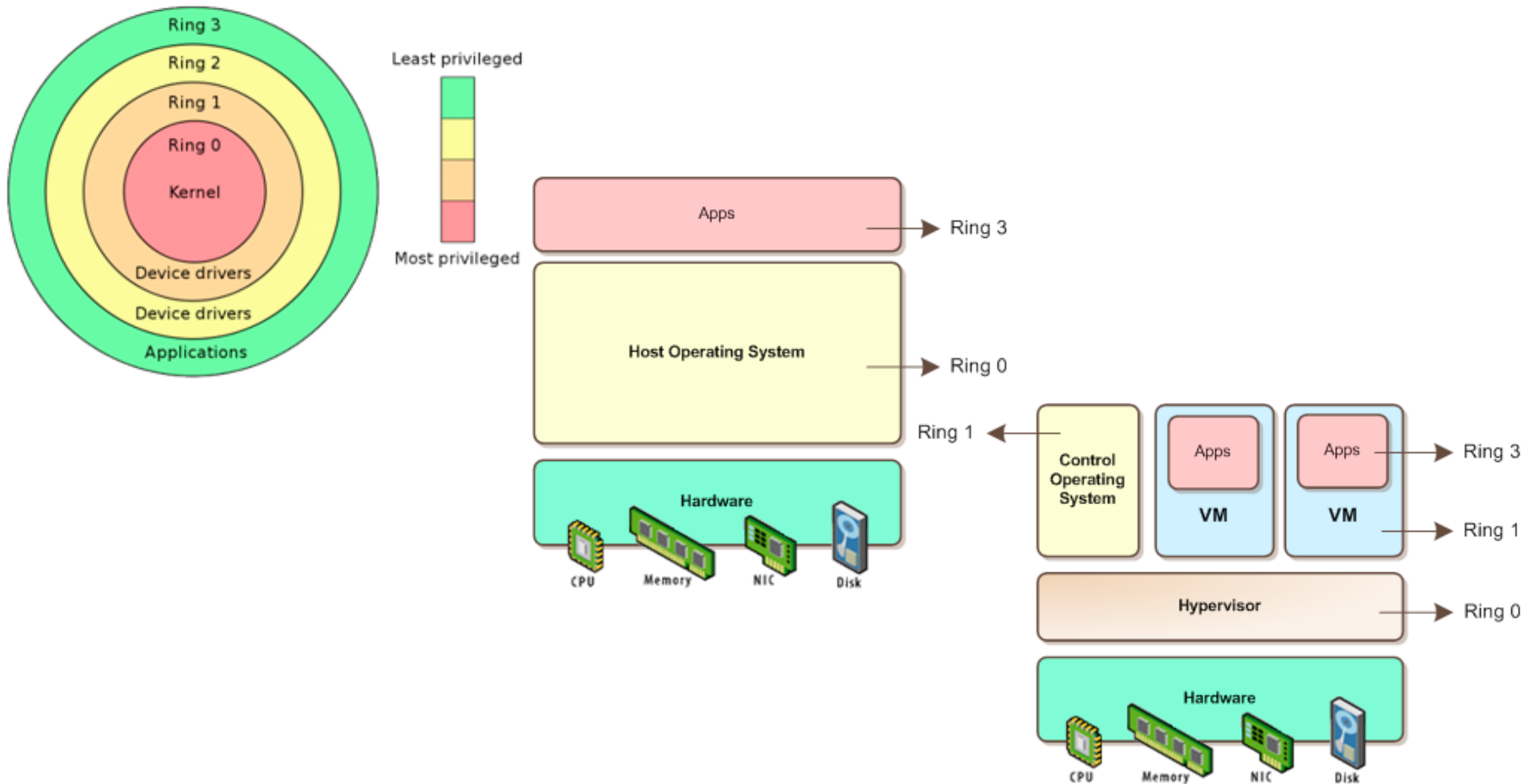
Virtual Machine Monitor

Hardware

Multiprocessor Virtualization



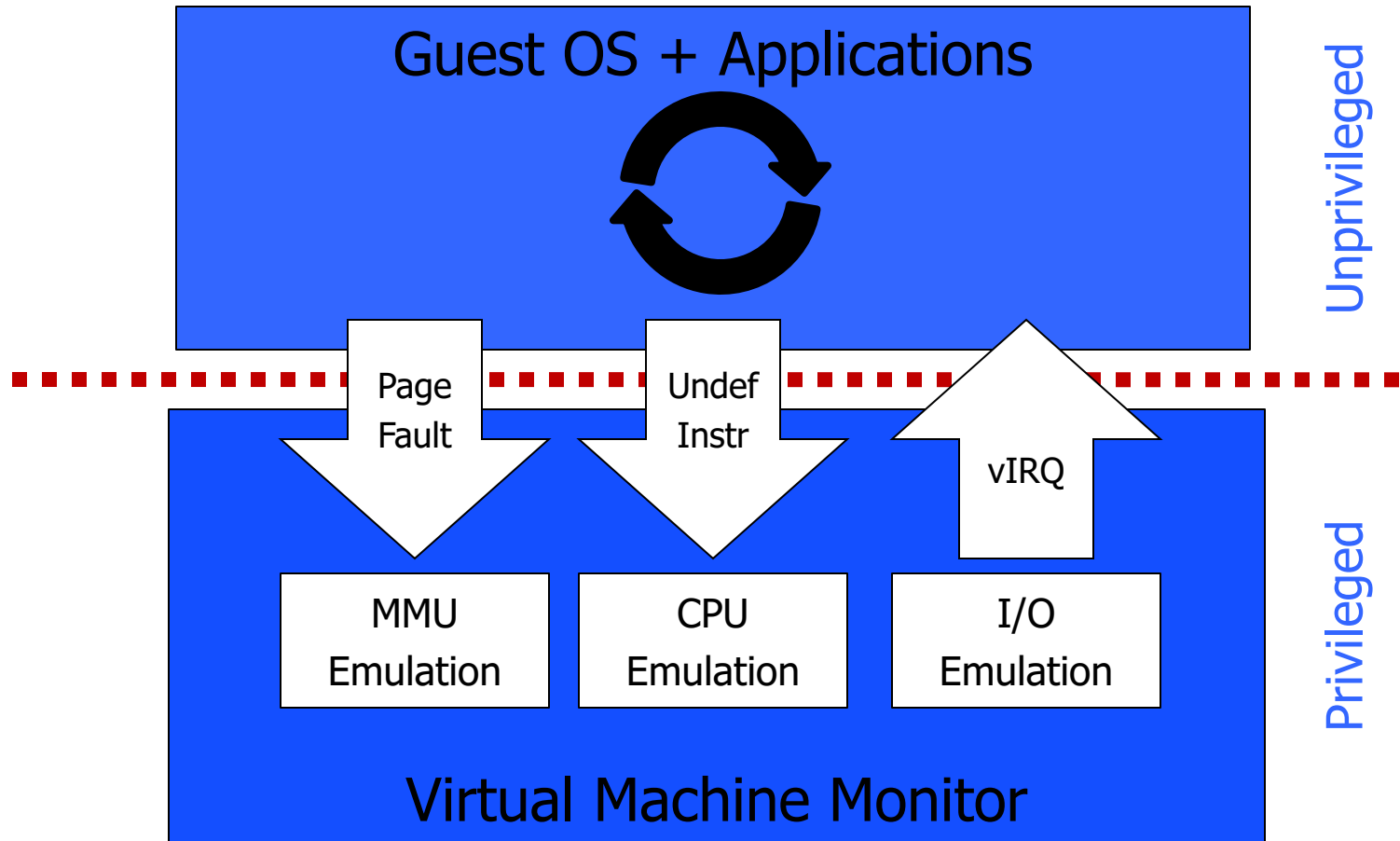
Virtualizing Processor



Trap and Emulate

- ◆ Run guest operating system deprivileged
- ◆ All privileged instructions trap into VMM
- ◆ VMM emulates instructions against virtual state
 - E.g., disable virtual interrupts, not physical interrupts
- ◆ Resume direct execution from next guest instruction

Trap and Emulate



Pretend to OS it's still running in privileged mode

Issues with Trap-and-Emulate

- ◆ Not all architectures support it
- ◆ Trap costs may be high
- ◆ VMM consumes a privilege level
 - Need to virtualize the protection levels

“Strictly Virtualizable”

A processor or mode of a processor is **strictly virtualizable** if, when executed in a lesser privileged mode:

- ◆ all instructions that access privileged state trap
- ◆ all instructions either trap or execute identically

x86 is Not Strictly Virtualizable

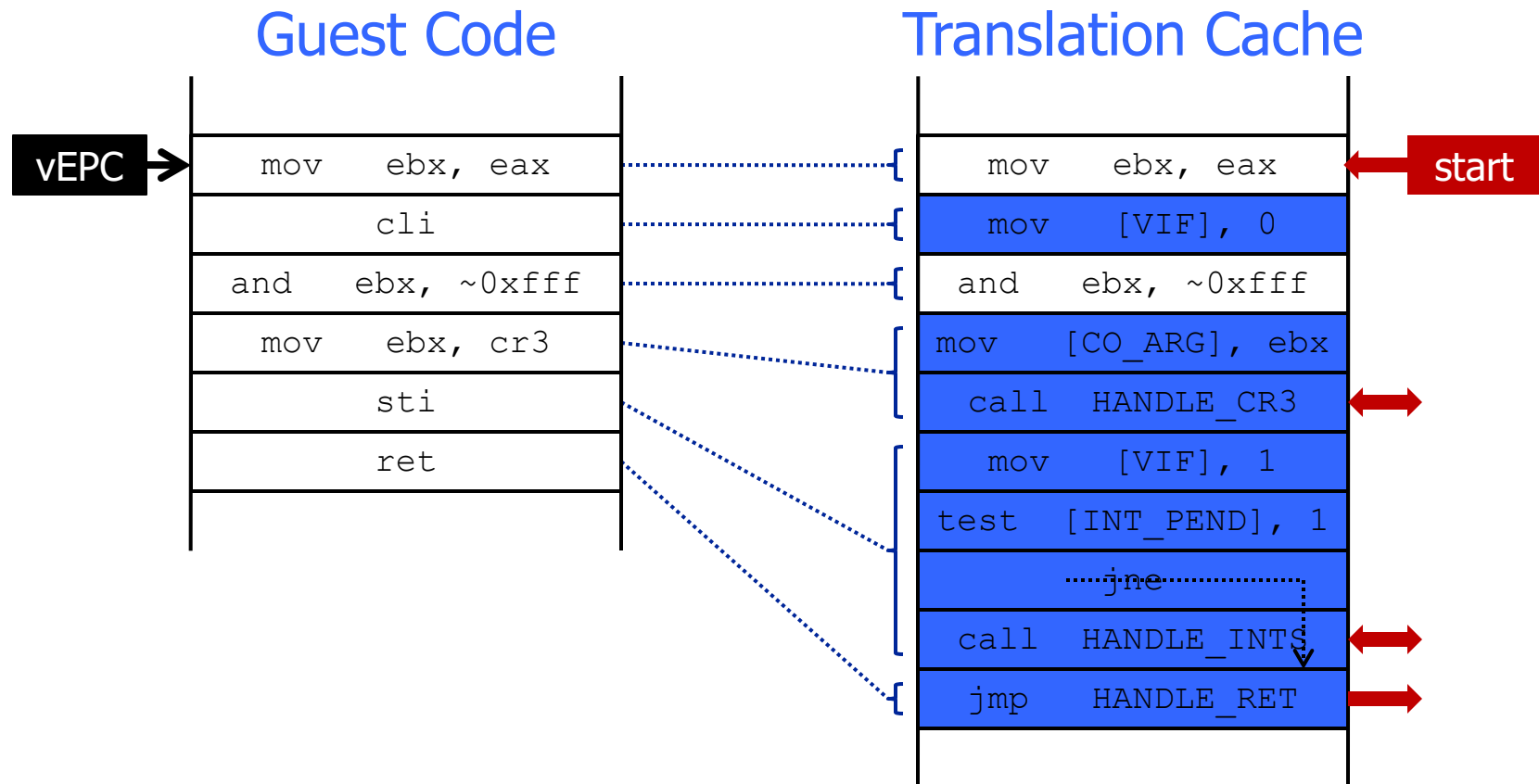
- ◆ On x86, **popf** instruction takes a word off the stack and puts it into the flags register
- ◆ At OS level, **popf** updates interrupt enable flag (IF)
- ◆ At user level, updates to IF silently dropped
 - To prevent user-level code from messing up the OS
- ◆ When VMM runs the OS at user level, all OS modifications to IF are dropped and VMM doesn't know whether OS wants interrupts enabled or not
- ◆ Several other reasons why trap-and-emulate wouldn't work for x86

Binary Translation

Goal: translate potentially dangerous and non-virtualizable instruction sequences one-by-one into safe sequences

- Privileged instructions, control flow, memory accesses
- ◆ If safe instruction, copy into translation cache
- ◆ If simple dangerous instruction, “inline” translate into short sequence of emulation code, copy into translation cache
 - Example: modification of the Interrupt Enable flag
- ◆ If another dangerous instructions, execute emulation code in the monitor (“call-out”)
 - Example: change to the page table base, branch ending basic block
- ◆ Monitor jumps to the start of the translated basic block with the virtual registers in the hardware registers.

Example of Binary Translation



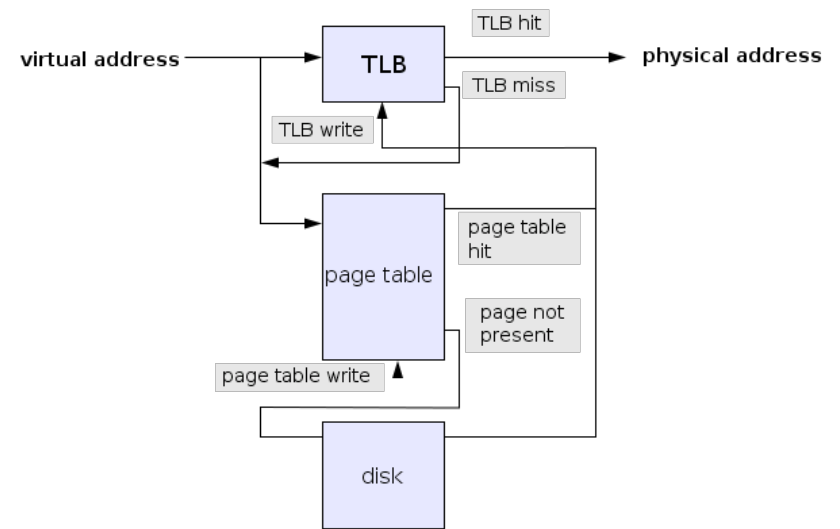
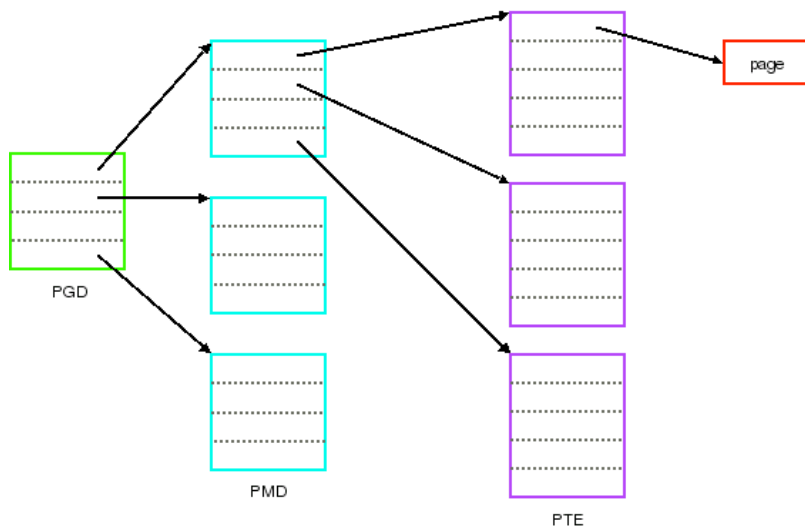
Issues with Binary Translation

- ◆ Translation cache management
- ◆ PC synchronization on interrupts
- ◆ Self-modifying code
 - Notified on writes to translated guest code
- ◆ Protecting VMM from guest

x86 Memory Management

- ◆ The processor operates with virtual addresses
- ◆ Physical memory operates with physical addresses
- ◆ Hardware translation lookaside buffer (TLB) maps virtual to physical page addresses
- ◆ TLB misses handled in hardware
 - Hardware walks the page tables and inserts virtual to physical mapping

x86 Memory Management

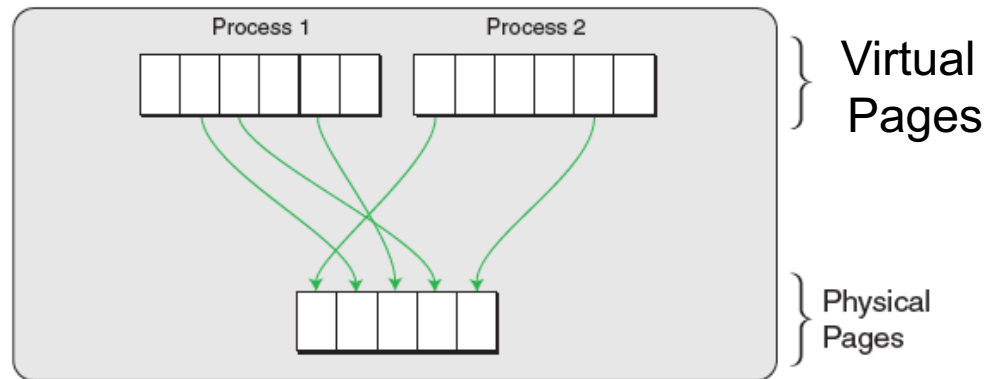


Virtualizing Memory

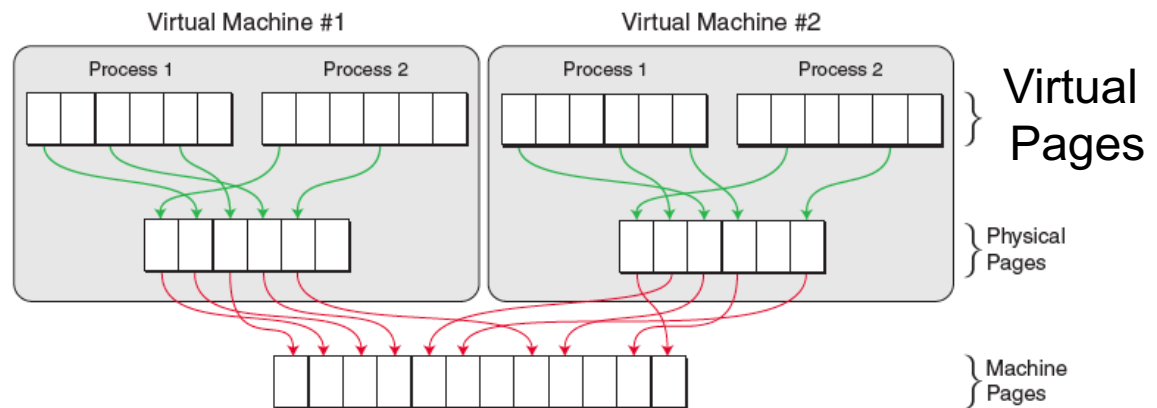
- ◆ OS assumes that it has full control over memory
 - Management: assumes it owns it all
 - Mapping: assumes it can map any virtual → physical
- ◆ However, VMM partitions memory among VMs
 - VMM needs to assign hardware pages to VMs
 - VMM needs to control mapping for isolation
 - Cannot allow OS to map any virtual → hardware page
- ◆ Hardware-managed TLBs make this difficult
 - On TLB misses, hardware walks page tables in memory
 - VMM needs to control access by OS to page tables

Virtualized Memory

Native



Virtualized



Shadow Page Tables

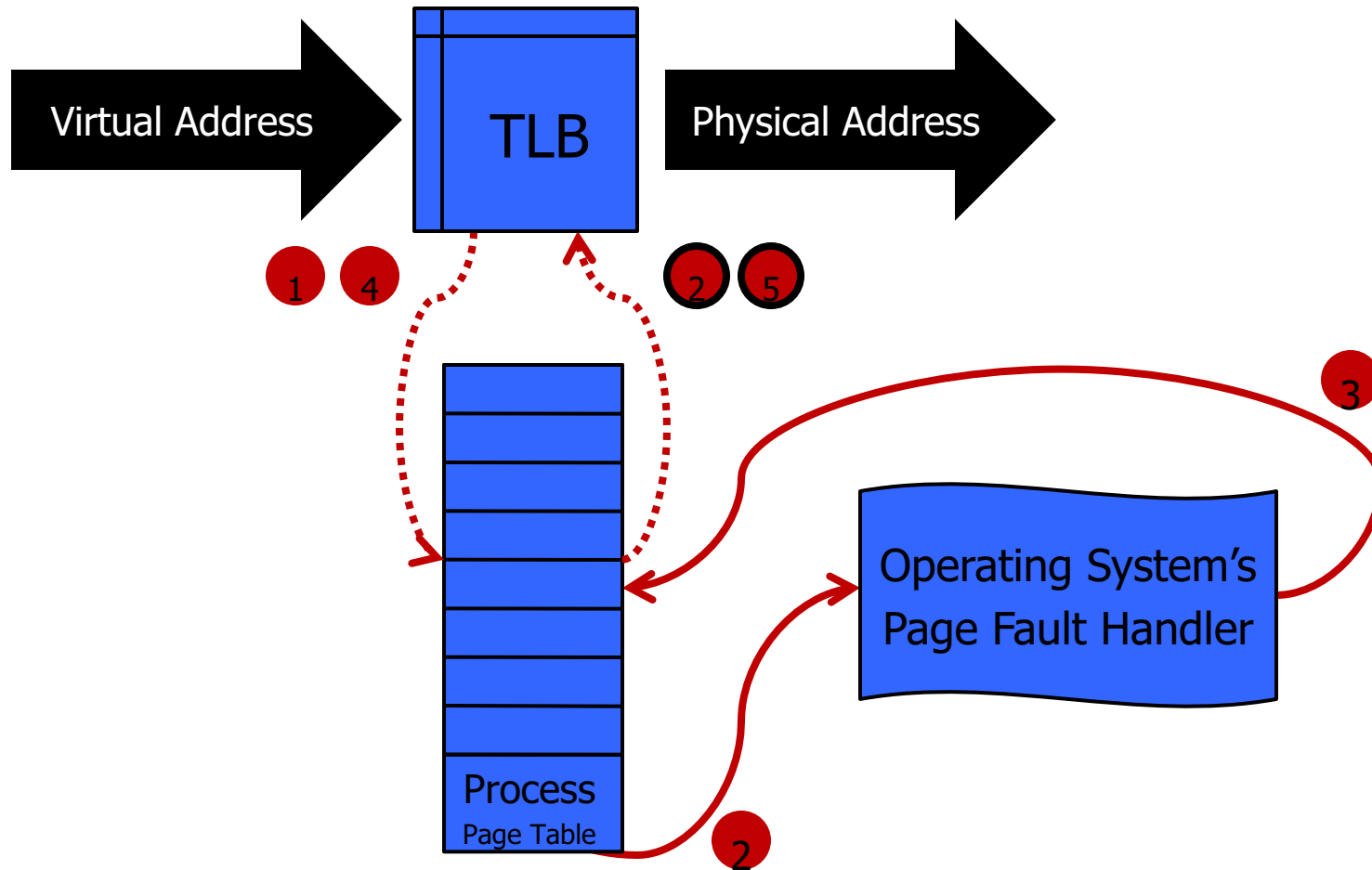
◆ Three abstractions of memory

- Machine: actual hardware memory (e.g. 2GB of DRAM)
- Physical: abstraction of hardware memory, OS managed
 - E.g. VMM allocates 512 MB to a VM, the OS thinks the computer has 512 MB of contiguous physical memory (underlying machine memory may be discontinuous)
- Virtual: virtual address space
 - Standard 2^{32} address space

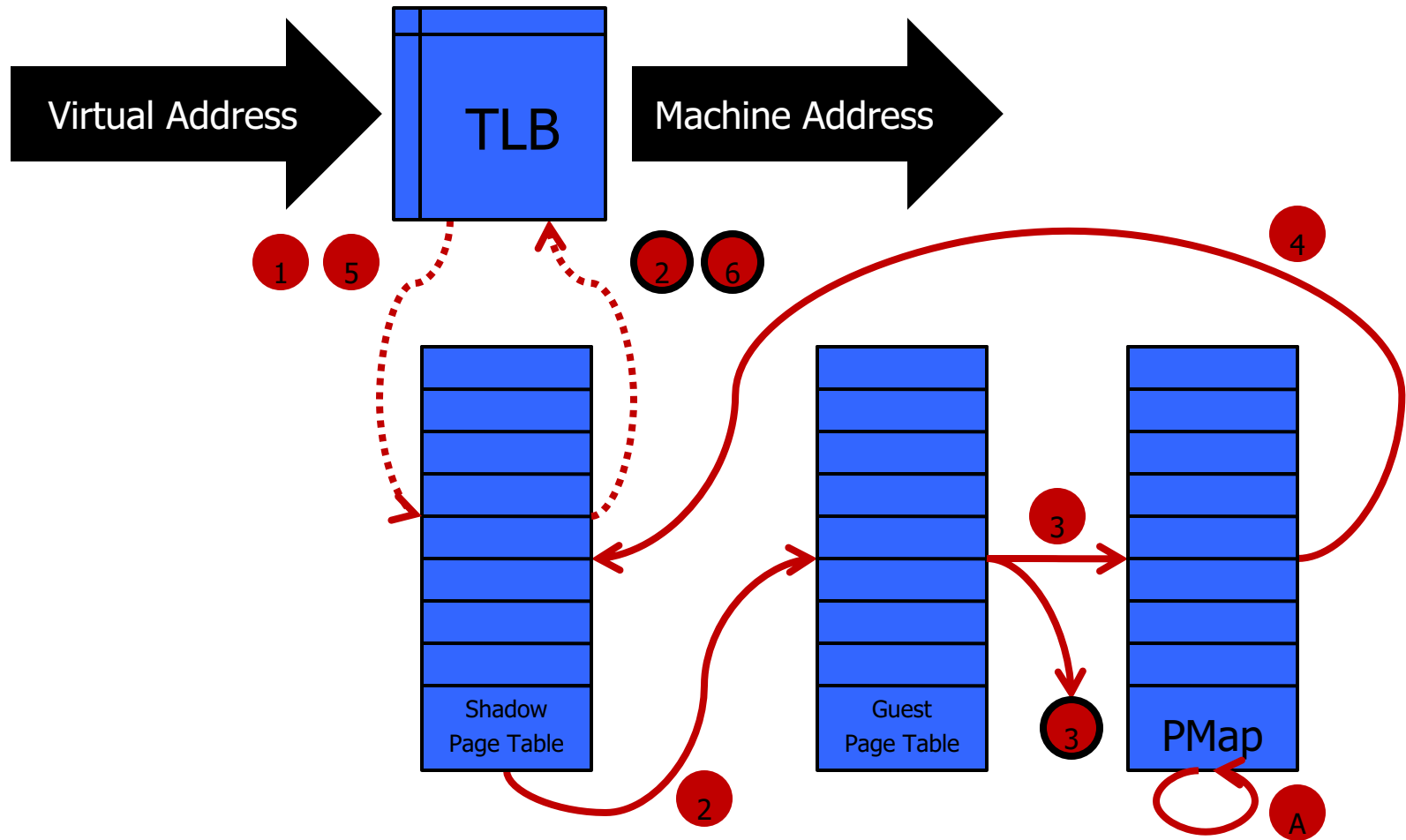
◆ In each VM, OS creates and manages page tables for its virtual address spaces without modification

- But these page tables are not used by the MMU

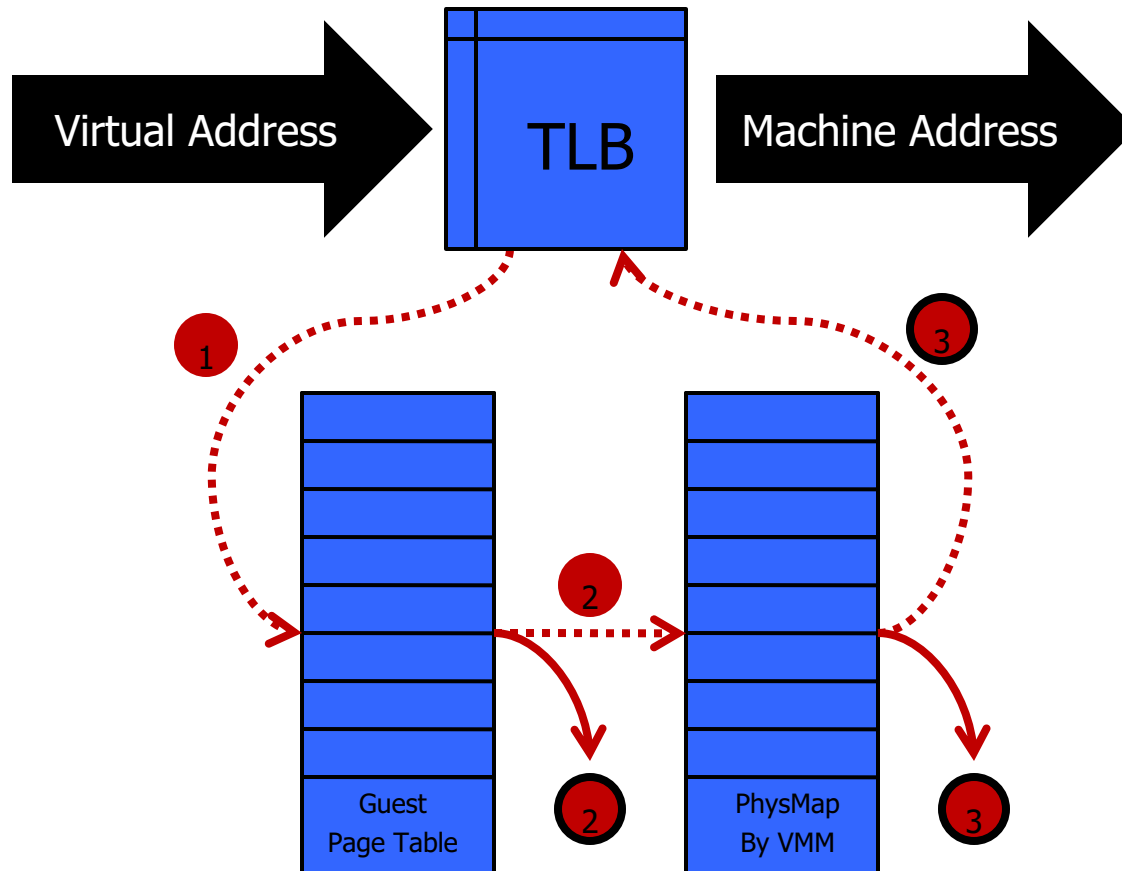
Traditional Address Translation



Using Shadow Page Tables

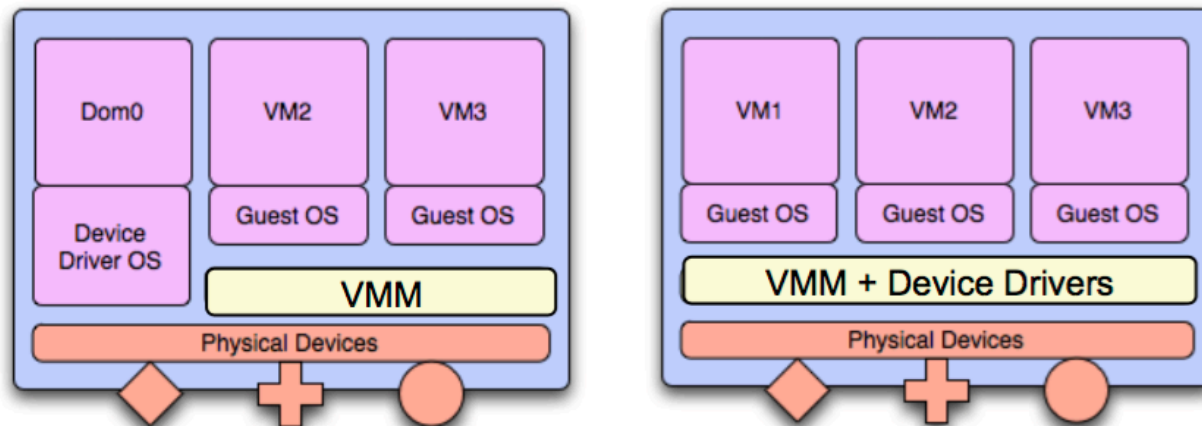


Using Nested Page Tables

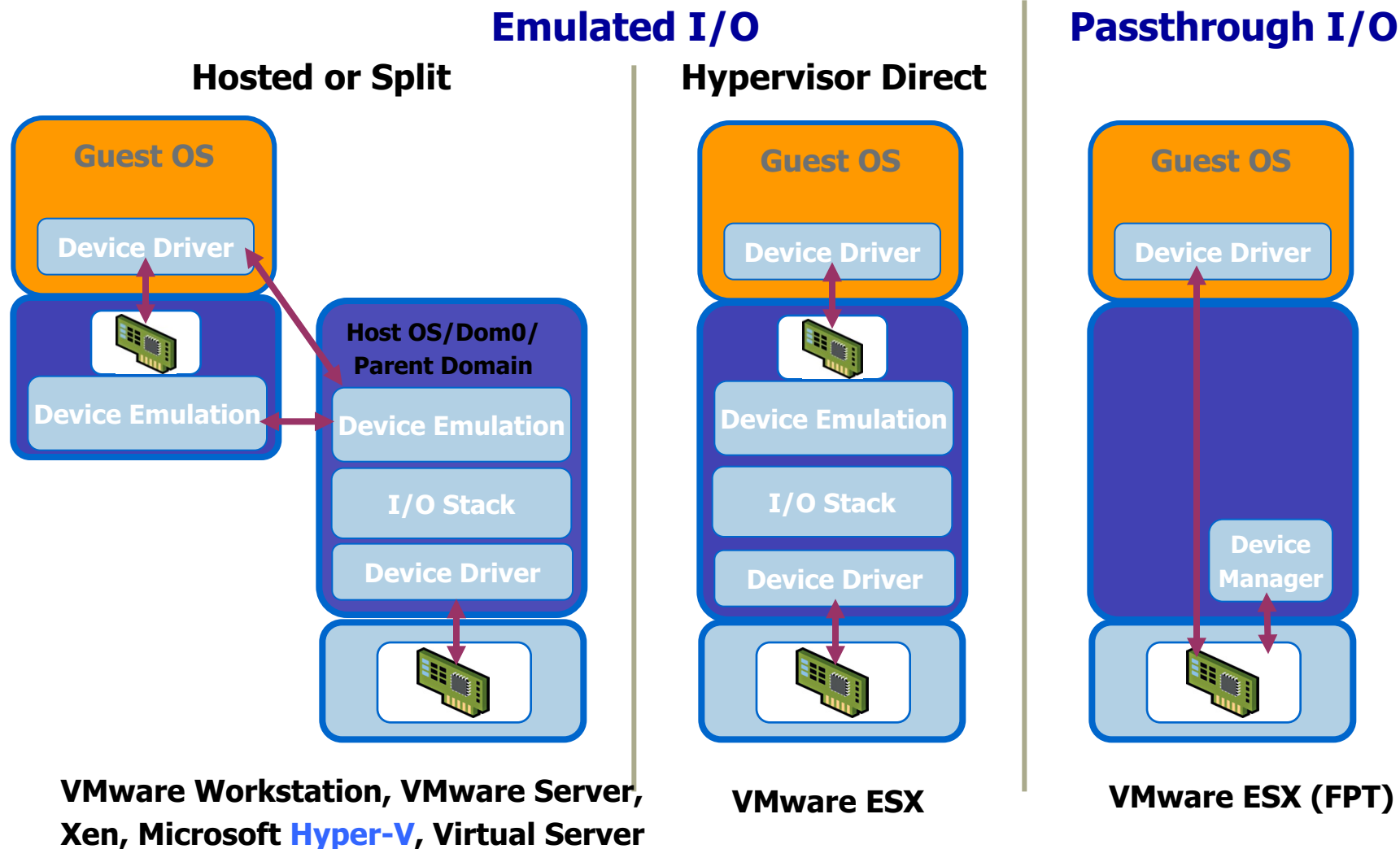


Virtualizing I/O

- ◆ Challenge: Lots of I/O devices... writing device drivers for all of them in the VMM layer is not a feasible option
- ◆ Insight: Device drivers already written for popular operating systems
- ◆ Solution: Present virtual I/O devices to guest VMs and channel I/O requests to a trusted host VM (popular OS)



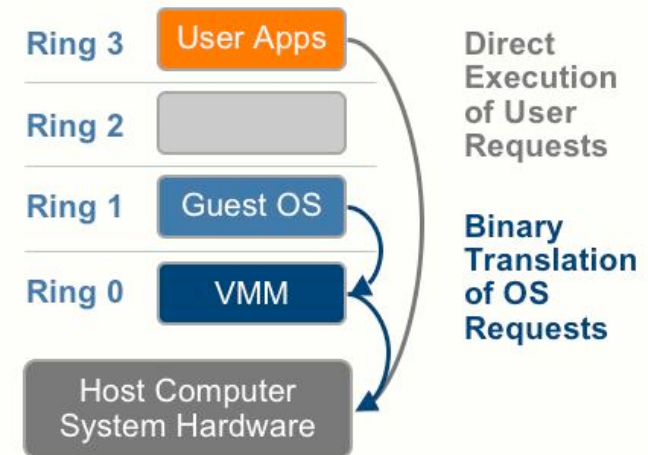
I/O Virtualization Implementations



Full Virtualization

Example: VMware ESX

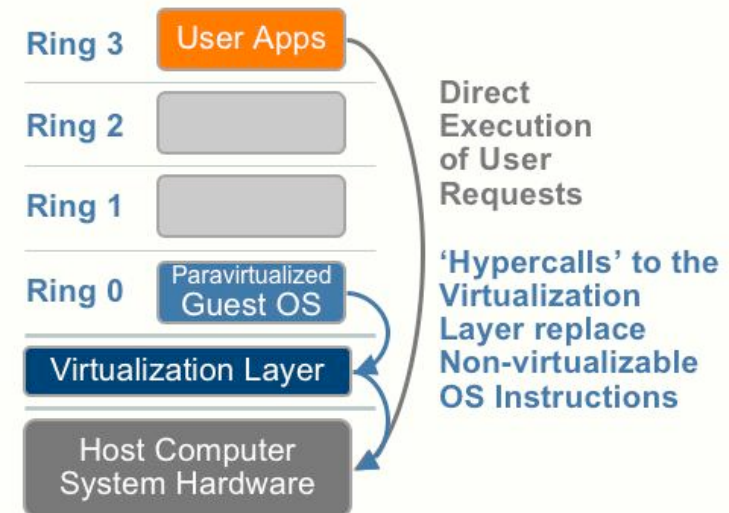
- ◆ Functionally identical to underlying physical hardware
- ◆ Functionality is exposed to the VMs
- ◆ Allows unmodified guest OS to execute on the VMs
 - Transparent to OS: VM looks like the physical machine
 - This might result in some performance degradation



Para-Virtualization

Example: Xen

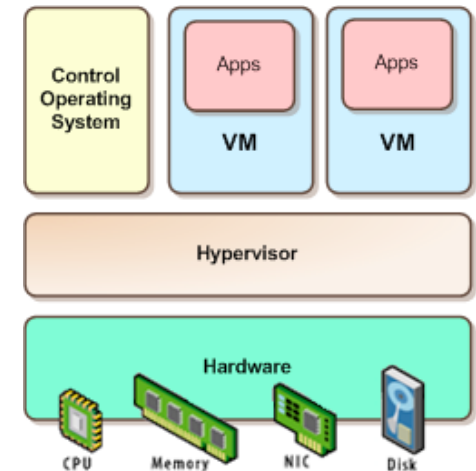
- ◆ Virtual hardware abstraction similar, but not identical to the real hardware
- ◆ Guest OS modified to cooperate with the VMM
 - Lower overhead leading to better performance



Type 1 vs. Type 2

◆ Native/Bare metal (Type 1)

- Higher performance
- ESX, Xen, HyperV



◆ Hosted (Type 2)

- Easier to install
- Leverage host's device drivers
- VMware Workstation, Parallels

