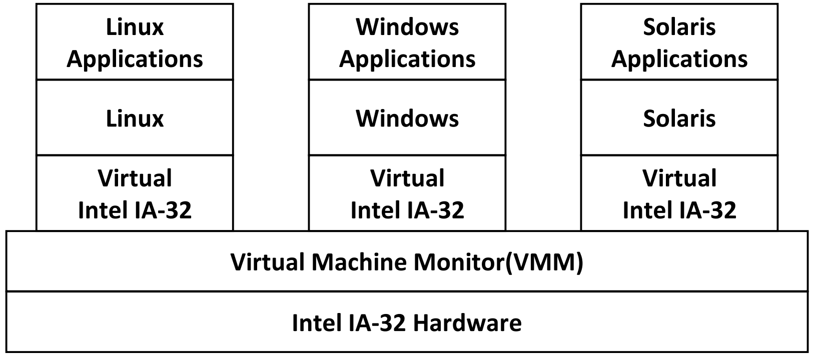
Lecture 12

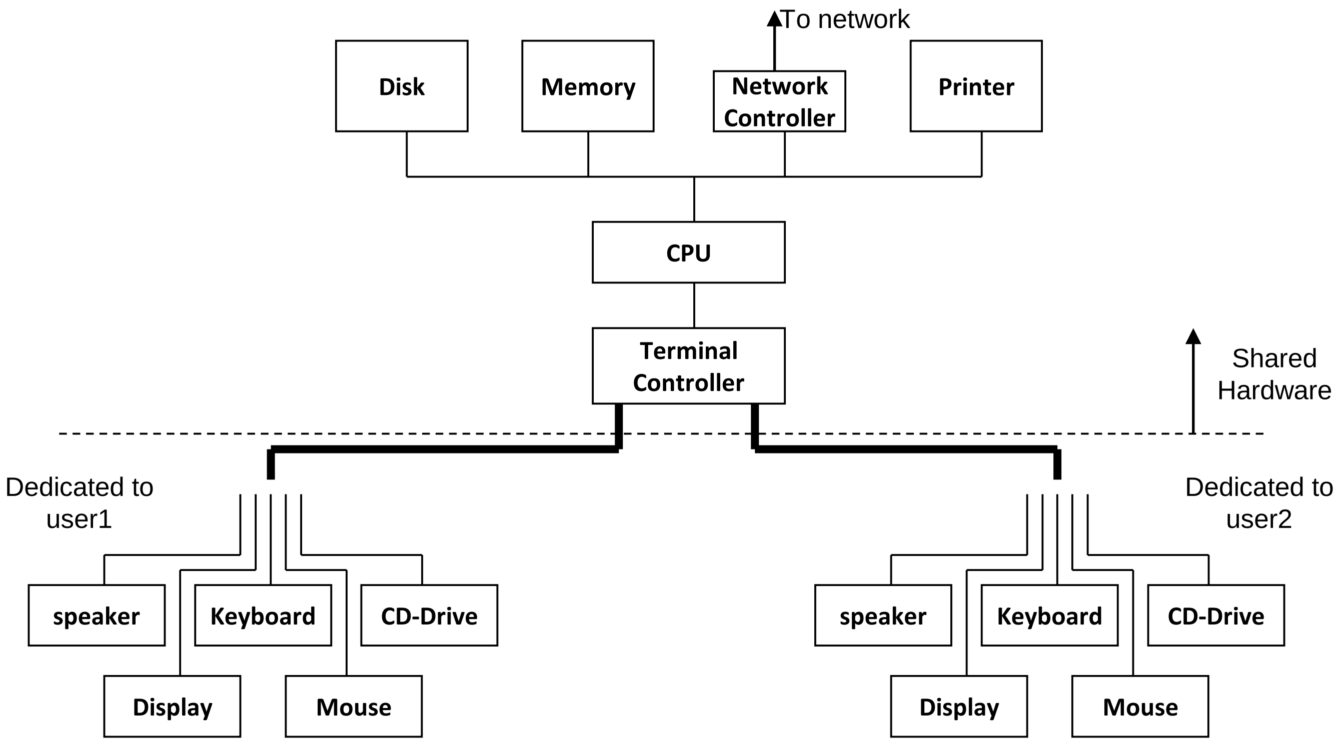
Virtual Machines

System Virtual Machines

- A system VM environment is capable of supporting multiple system images simultaneously, each running its own operating system and associated application programs
- Real resources of the *host* platform are shared among the *guest* system with the virtual machine monitor (VMM)
- Focus on VMs where ISA of the host and guest are the same

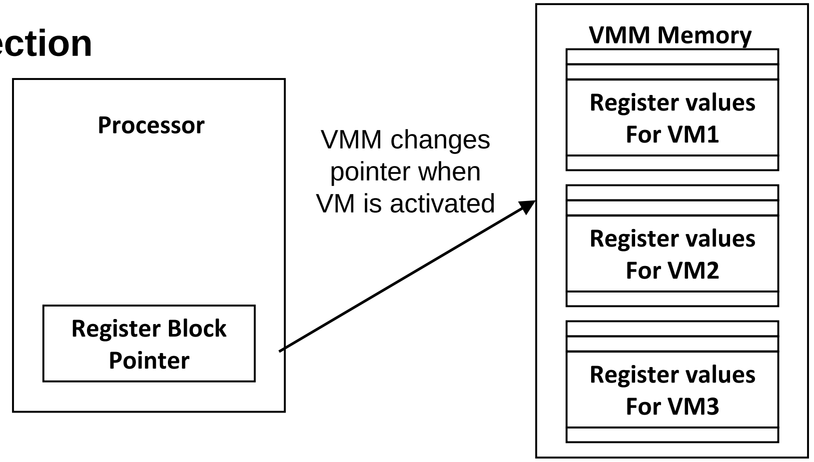


Outward Appearance



State Management

Indirection



Load register block pointer
To point to VM's registers
In VMM memory

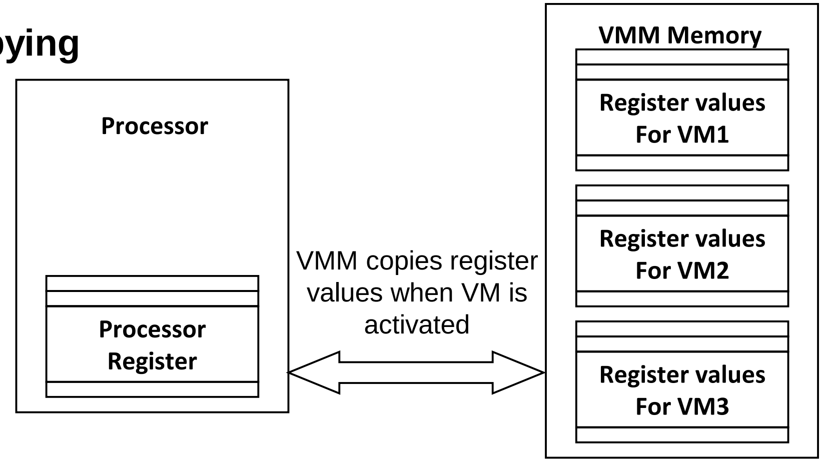
Load program counter to
Point to VM program and
Start execution

.
.

Load temp <- reg_pointer, index(A)
Store reg_pointer, index(B) <- temp

.
.

Copying



Copy register state from
VMM memory

Load program counter to
Point to VM program and
Start execution

.
.

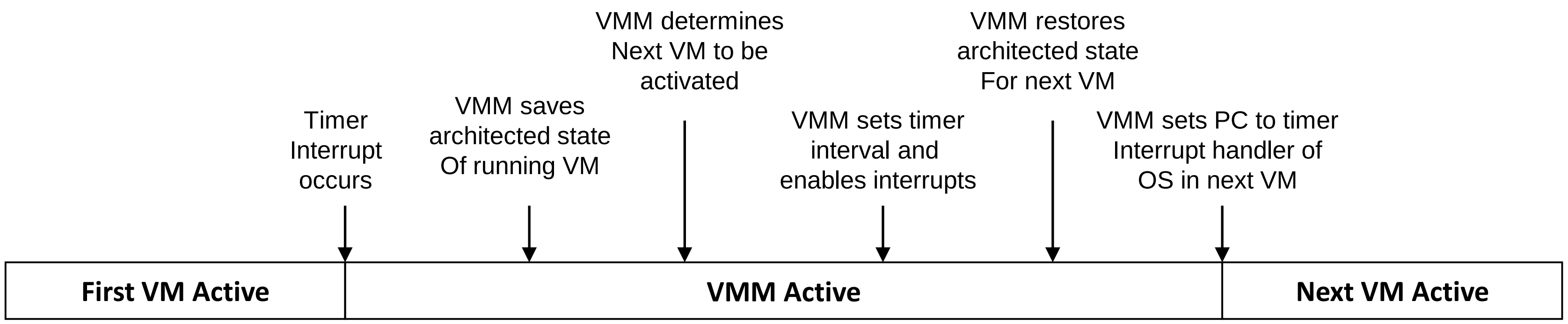
Mov reg A -> reg B

.
.

Copy register state from
Processor back to system memory

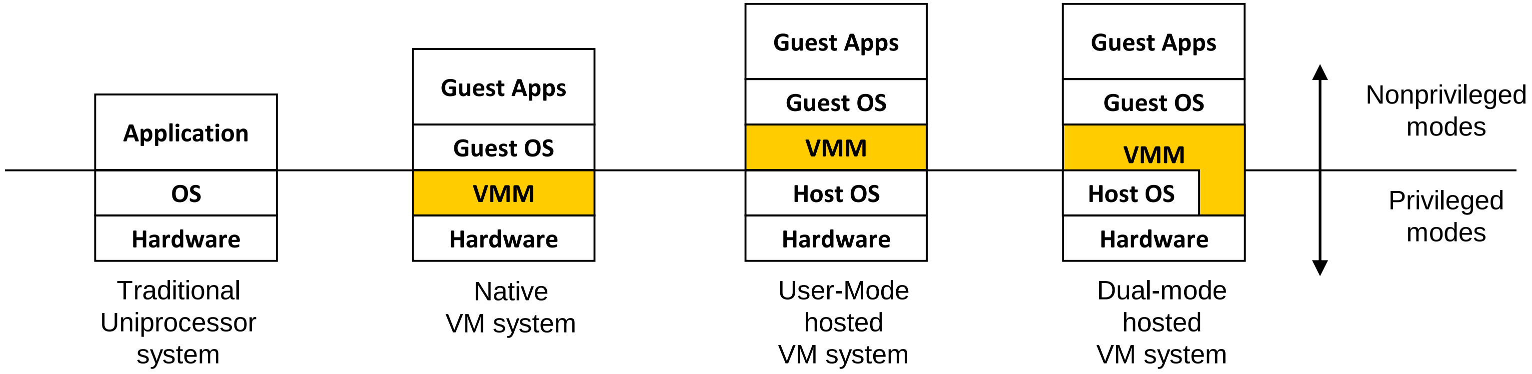
Resource Control

- Adopt time-sharing systems
- The VMM maintain overall control of all the hardware resources
- The interval timer interrupt
  - Guarantee that control is transferred back to VMM and the VMM handles the interrupt itself



Native and Hosted Virtual Machine

- Native VM system
  - A system where a VMM operates in a privilege mode higher than the mode of the guest virtual machines
  - The privilege level of the guest OS is emulated by the VMM
- Hosted VM system
  - The VMM is installed on a host platform that is already running an existing OS
  - The VMM utilizes the functions already available on the host OS to control and manage resources desired by each of the virtual machine



Resource Virtualization - Processor

- The key aspect of virtualizing a processor is the execution of the guest instructions, including both system-level and user-level instruction

- Processor virtualization method
  - Emulation
    - Interpretation, binary translation
    - Emulation is the only processor virtualization mechanism for different ISA between the guest and the host
  - Direct native execution
    - Only if the ISA of the host is identical to the ISA of the guest
    - The goal is to gain the same performance as it runs on the virtual machine

Resource Virtualization - Processor Conditions for ISA Virtualizability

- We restrict the discussion here to native system VMs
- In a native system VM, the VMM runs in system mode, and all other software runs in user mode
- The VMM keeps track of the intended mode of operation of a guest virtual machine but it will always use user mode in executing the instructions from the guest virtual machine

Resource Virtualization - Processor Conditions for ISA Virtualizability

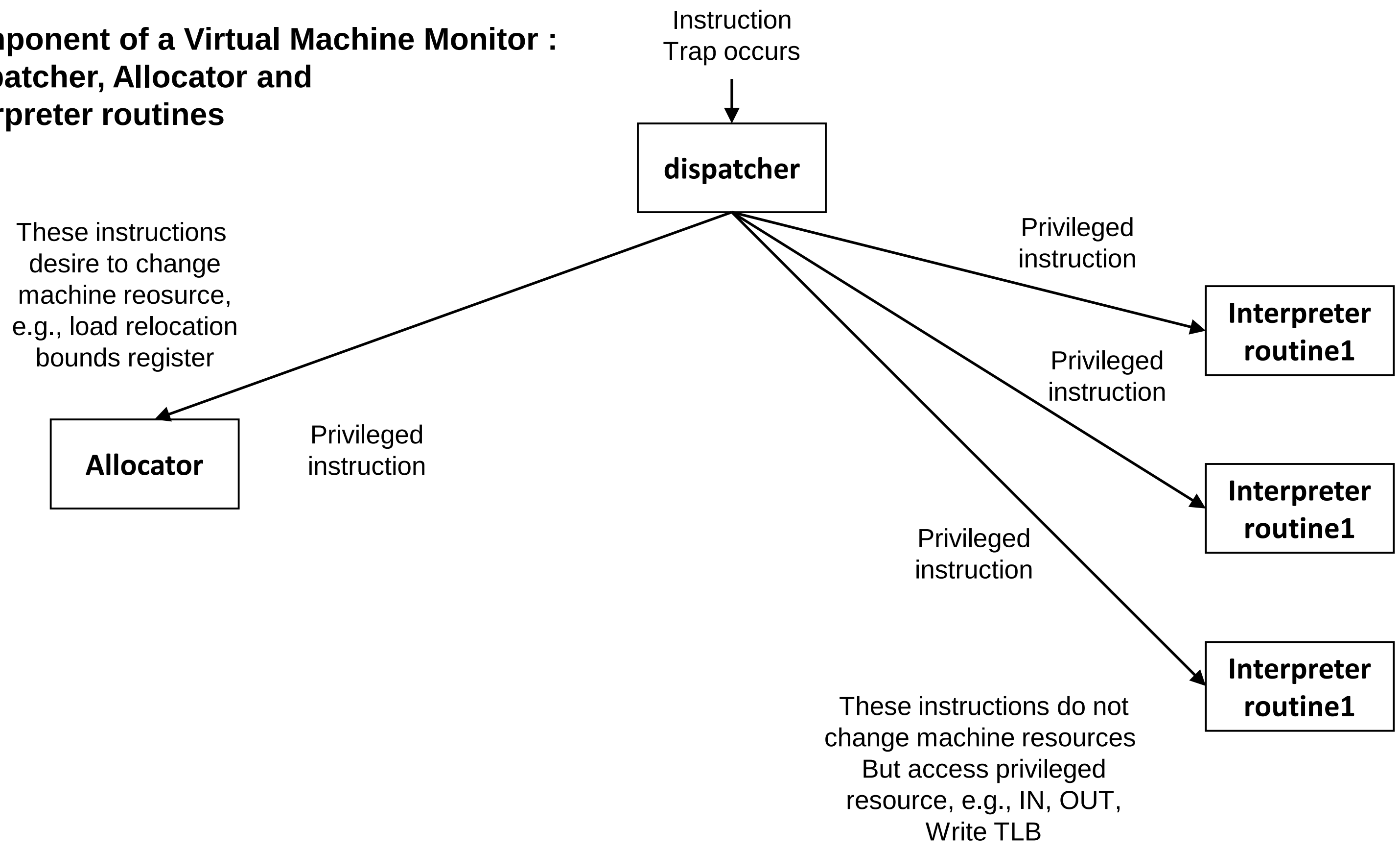
- A privileged instruction is defined as one that traps if the machine is in user mode and does not trap if the machine is in system mode
  - Load PSW (LPSW, IBM System/370)
    - Load the processor status word (PSW) from a location in memory if the processor is in system mode. If it is not in system mode, the machine traps
  - Set CPU Timer (SPT, IBM System/370)
    - Replaces the CPU interval timer with the contents of a location in memory if the CPU is in system mode and traps if it is not

Resource Virtualization - Processor Conditions for ISA Virtualizability

- To specify instructions that interact with hardware, two categories of special instructions are defined
  - Control-sensitive instruction
    - Attempt to change the configuration of resources in the system
    - For example Load PSW and Set CPU Timer
  - Behavior-sensitive instruction
    - Behavior or results produced depend on the configuration of resource such as : value in relocation bound register R or the mode of operation
    - For example Load Real Address (LRA) and Pop Stack into Flags Register (POPF)
  - Innocuous instruction
    - An instruction which is neither control sensitive or behaviour sensitive

Resource Virtualization - Processor Conditions for ISA Virtualizability

Component of a Virtual Machine Monitor : Dispatcher, Allocator and Interpreter routines

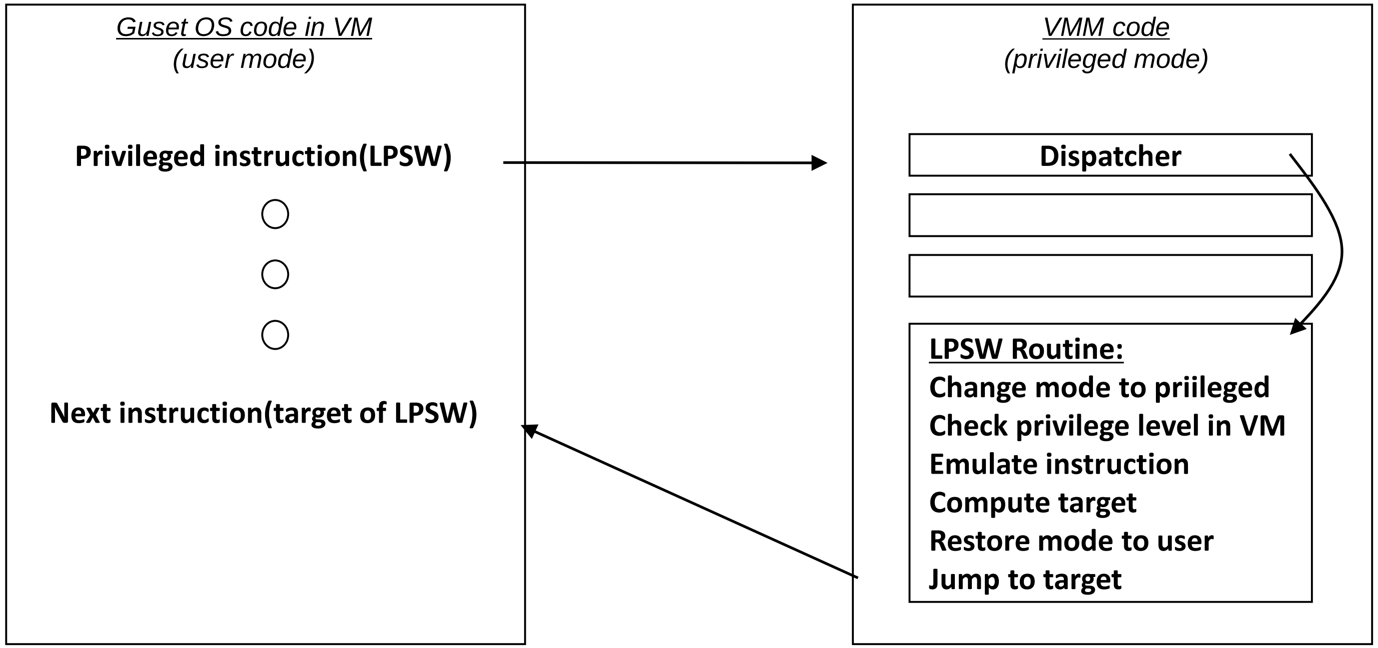


Resource Virtualization - Processor Conditions for ISA Virtualizability

- A potential virtual machine must satisfy three properties
  - Efficiency
  - Resource control
  - Equivalence
- Theorem 1 regarding (efficient) VMM construction
  - A virtual machine monitor may be constructed if the set of sensitive instruction is a subset of the set of privileged instructions
  - It means that an efficient virtual machine implementation can be constructed if instructions that could interfere with the functioning of the VMM always trap in the user mode

Resource Virtualization - Processor Conditions for ISA Virtualizability

- The VMM interprets a sensitive instruction according to the prevailing status of the virtual system resources and the state of the virtual machine

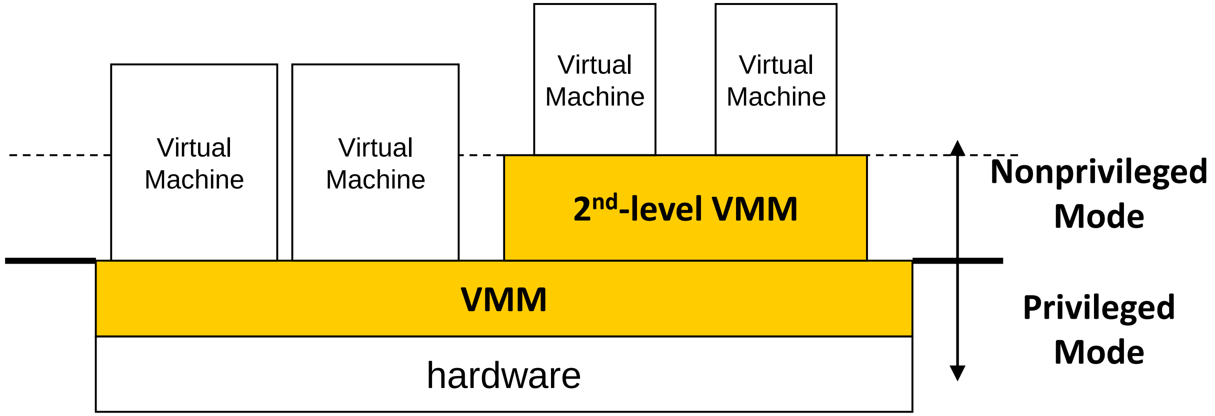


Resource Virtualization - Processor Conditions for ISA Virtualizability

- Interpreting the SPT instruction
  - The VMM examines the contents of the location to be loaded into the CPU timer
    - If(t < T) t is loaded, else T is loaded
      - t : the content of the location
      - T : the time remaining from the allocated time for the virtual machine itself
  - Meanwhile, it keeps the time difference(t - T) in an internal table so that this time can be restored when the guest VM is again activated

Resource Virtualization – Processor
Recursive Virtualization

- The concept of running the virtual machine system on a copy itself



- Two effects that usually restrict the ability to create an efficient recursively virtualizable system
  - Timing dependancies
  - Memory allocation

Resource Virtualization – Processor Recursive Virtualization

- Theorem 2
  - A conventional third-generation computer is recursively virtualizable if
    - it is virtualizable and
    - a VMM without any timing dependences can be constructed for it

Resource Virtualization – Processor
Handling Problem Instructions

- The POPF in IA32 instruction is sensitive but not privileged
  - It is a **critical instruction** (sensitive but not privileged)
  - It does not generate a trap in user mode
  - It violate the virtualizability condition of Theorem 1

- An additional set of steps must be taken in order to implement a system virtual machine (with possible loss of some efficiency)
  - It is possible for a VMM intercepts POPF and other critical instructions if all guest software were interpreted instruction by instruction
  - Techniques related to those described in Chapters 2 and 3 can be used to reduce the inefficiency due to the use of interpretation

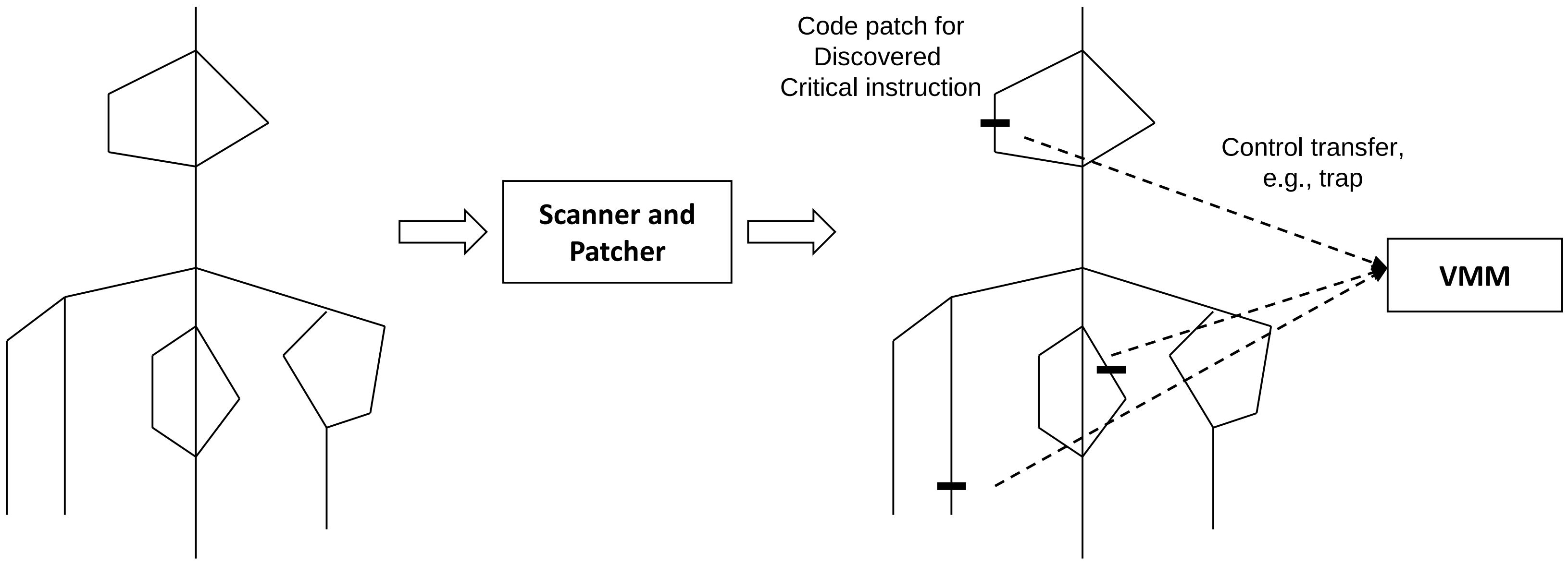
Resource Virtualization – Processor Handling Problem Instructions

- The VMM scans the guest code stream to discover all critical instructions and replace them with a trap or jump to VMM

- The process above is called *patching*

Resource Virtualization – Processor Handling Problem Instructions

- Patching

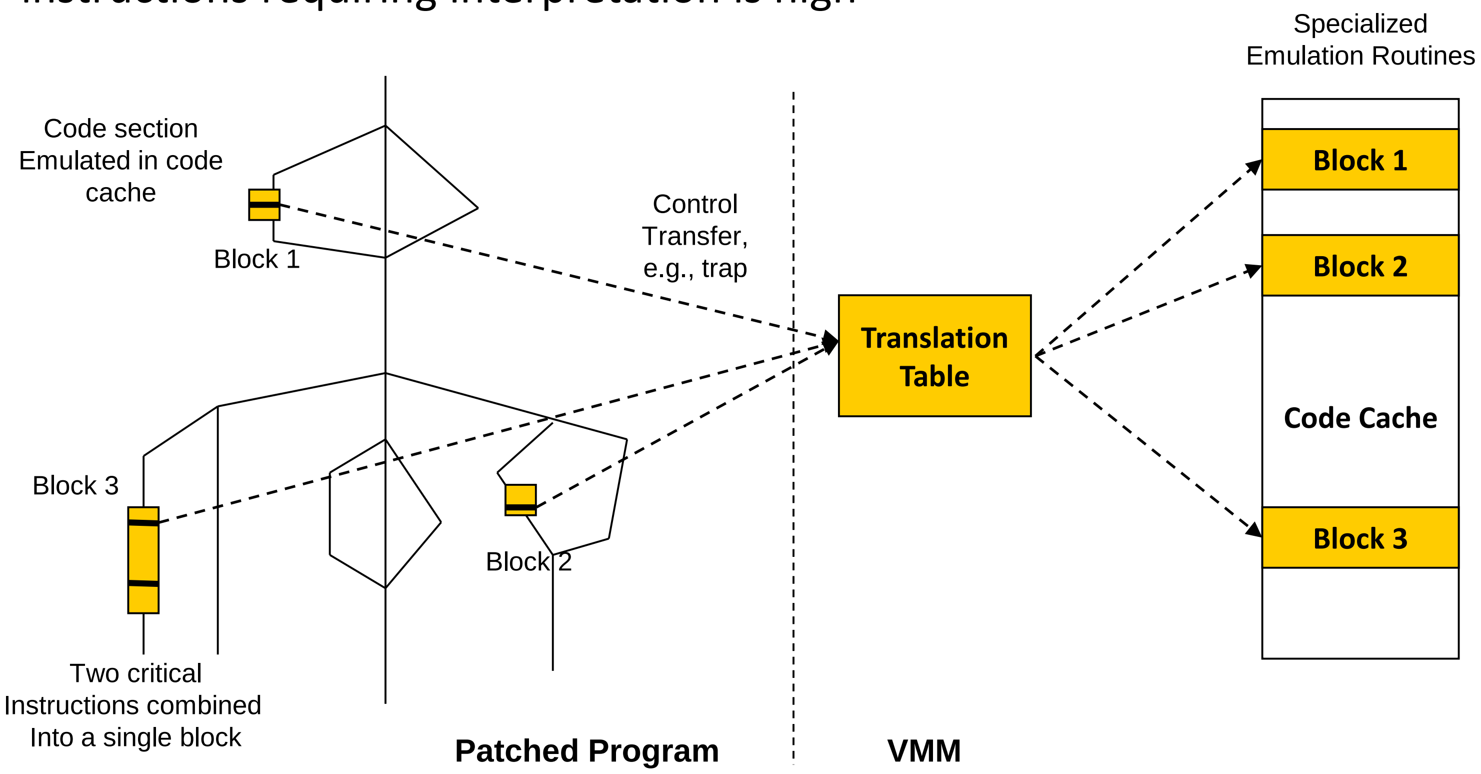


Resource Virtualization – Processor Patching of Critical Instructions

- One way to discover critical instructions
  - The VMM takes control at the head of each guest basic block and scan instructions in sequence until the end of the basic block is reached
    - If a critical instruction is found, it is replaced with a trap to the VMM
    - Another trap back to the VMM is placed at the end of the basic block
- To reduce overhead, the trap at the end of a scanned basic block can be replaced by the original branch or jump instruction

Resource Virtualization – Processor Caching Emulation Code

- Reduce the overhead of VMM interpretation when the frequency of sensitive instructions requiring interpretation is high



Resource Virtualization – Input/Output Virtualizing Device

- Dedicated Devices
  - Some I/O device is dedicated to a particular guest VM or at least are switched from one guest to another on a very long time scale
  - The device itself does not necessarily have to be virtualized
  - Requests to and from the device could theoretically bypass the VMM and go directly to the guest operating system

- Partitioned Device
  - A very large disk, for example, can be partitioned into several smaller virtual disk that are then made available to the virtual machine as dedicated devices

Resource Virtualization – Input/Output Virtualizing Device

- Shared Devices
  - Some device, such as a network adapter, can be shared among a number of guest VMs at a fine time granularity
  - Each guest may have its own virtual state related to usage of the device, e.g., a virtual network address.
    - This state information is maintained by the VMM for each guest VM

- Nonexistent Physical Device
  - Virtual devices “attached” to a virtual machine for which there is no corresponding physical device
  - For example, a network adapter that is used for communicating with other virtual machines on the same platform

Resource Virtualization – Input/Output
Virtualizing Device

- Spooled Device
  - Virtualization of spooled device can be performed by using a two-level spool table approach

Virtual Machine 1 Spool Table

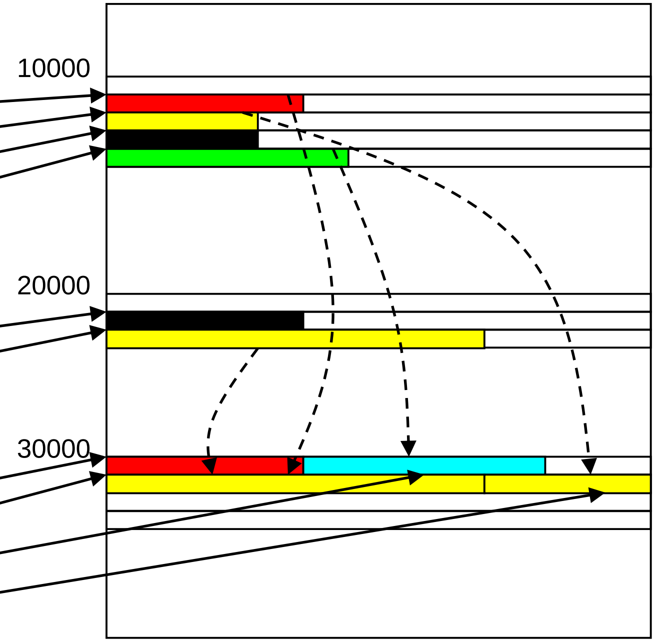
Program	Status	Location	Real loc	Size
A	Printed	1000	11000	400
B	Completed	2000	12000	200
C	Running	3000	13000	200
D	Completed	4000	14000	500

Virtual Machine 2 Spool Table

Program	Status	Location	Real loc	Size
P	Running	1000	21000	400
Q	Completed	2000	22000	800

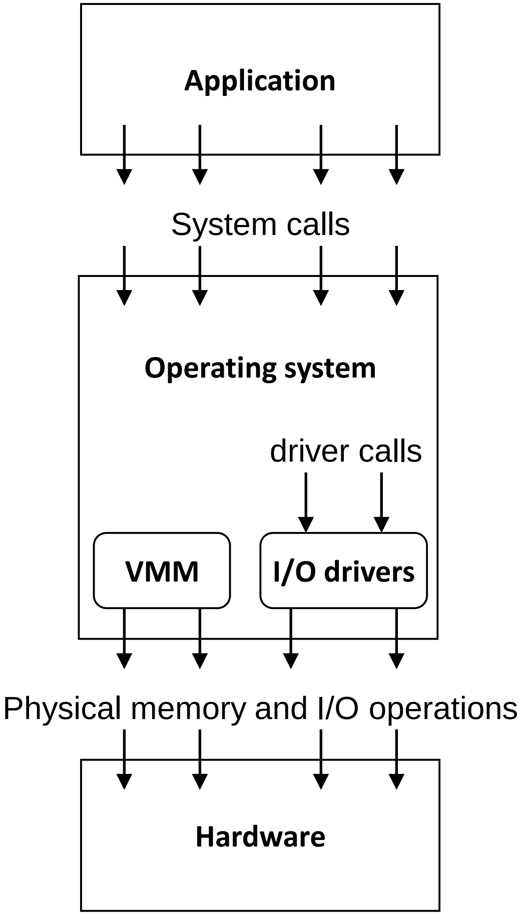
VMM Spool Table

VM	Status	Status	Real loc	Size
1	A	Printed	30000	400
2	Q	Printing	31000	800
1	B	Waiting	31800	200
1	D	Waiting	30400	500



Resource Virtualization – Input/Output
Virtualizing I/O Activity

- An application program makes device-independent I/O request
- The Operating system converts the device-independent request into calls to device driver routines
- A device driver takes care of device-specific aspects of performing an I/O transaction
- The VMM can intercept a guest’s I/O action and convert it from a virtual device action to a real device action at any of the three interface
    - The system call interface
    - The device driver interface
    - The operational-level interface



Resource Virtualization – Input/Output
Virtualizing I/O Activity

- Virtualizing at the I/O operation Level
  - The privileged nature of the I/O operations make them easy for the VMM to intercept because they trap in user mode

- Virtualizing at the Device Driver Level
  - If the VMM can intercept the call to the virtual device driver, it can convert the virtual device information to the corresponding physical device and redirect the call to a driver program for the physical device
  - It requires that the VMM developer have some knowledge of the guest operating system and its internal device driver interfaces

- Virtualizing at the System call Level
  - The virtualization process could be made more efficient by intercepting the initial I/O request at the OS interface, the ABI. Then the entire I/O action could be done by the VMM

Input / Output Virtualization and Hosted Virtual Machines

- An I/O request from a guest virtual machine is converted by the native-mode portion of the VMM into a user application request made to the host

- An advantage of a hosted virtual machine
    - It is not necessary to provide device drivers in the VMM
    - the actual device drivers do not have to be incorporated as part of the VMM

Input / Output Virtualization and Hosted Virtual Machines

- A component that form a dual mode hosted virtual machine system
  - VMM-n(native)
    - Intercepts traps due to privileged instructions or patched critical instructions encountered in a virtual machine
  - VMM-u(user)
    - Makes resource requests to the host OS
  - VMM-d(driver)
    - Provide a means for communication between the other two components

Memory Virtualization

Virtual Memory for System VMs

"Any **programming** problem can be solved
by **adding** a level of indirection."

Anonymous

“Any **programming** problem can be solved
by **adding** a level of indirection.”

Anonymous

“Any **performance** problem can be solved
by **removing** a level of indirection.”

M. Haertel

Implementation

- Translation Lookaside Buffer is used to cache locations of frequently-accessed pages
  - “Architected Page Table” ➔ Hardware maintains page table & instructions regarding it are part of the ISA. The TLB is invisible to ISA and OS.
  - “Architected TLB” ➔ Instructions to manipulate TLB are part of the ISA. OS implements the page table (hardware is unaware) More recent.

Implementation

- Architected Page Tables
  - Guest OS manages virtual-to-real mappings for its processes.
  - VMM manages virtual-to-physical mappings in a *Shadow Page Table*, and virtualizes the page table pointer. Instructions to modify this trap & the VMM updates guest's page table pointer.

Page Table Pointer
P1V1
P1V2
P2V1
P2V2
P3V2
…

Virtual Page	Physical Page
------	------
1,000	1,000
------	------
2,000	500
------	------
Program 1 on VM1	

Virtual Page	Physical Page
------	------
1,000	not mapped
------	------
4000	not mapped
------	------
Program 2 on VM1	

Hardware support for virtualization

Intel Virtualization Solution

Source: Y. Chung, National Tsinghua University, Taiwan

What is needed

- CPU Virtualization: Intel VT-x
- Memory Virtualization: Extended Page Tables (EPT)
- IO Virtualization: Intel VT-d

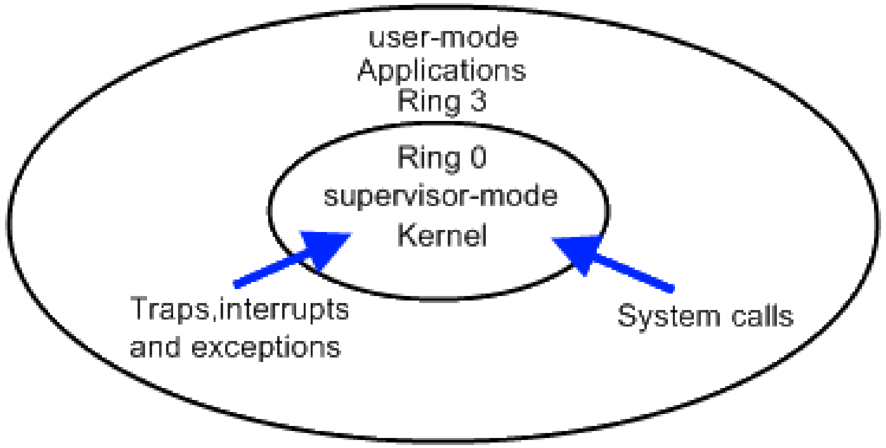
Trap and Emulate Model

- We want CPU virtualization to be efficient
  - We should make guest binaries run on CPU as fast as possible.
  - Theoretically speaking, if we can run all guest binaries natively, there will NO overhead at all.
  - But we cannot let guest OS handle everything, VMM should be able to control all hardware resources.

- Solution :
  - Ring Compression
    - Shift traditional OS from kernel mode(Ring 0) to user mode(Ring 1), and run VMM in kernel mode.
    - Then VMM will be able to intercept all trapping event.

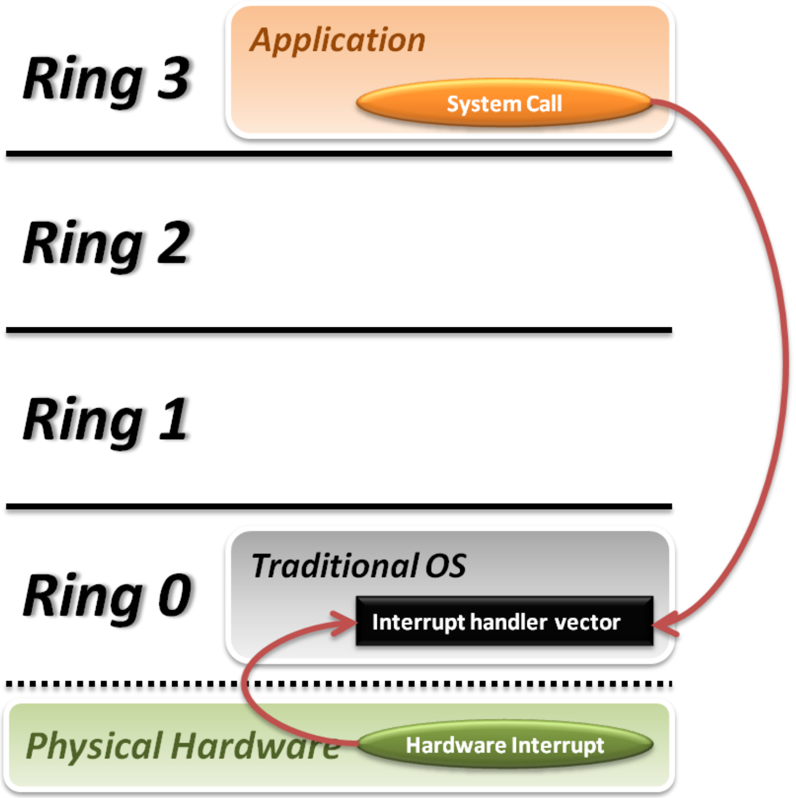
Trap and Emulate Model

- VMM virtualization paradigm *(trap and emulate)* :
  1. Let normal instructions of guest OS run directly on processor in user mode.
  2. When executing privileged instructions, hardware will make processor trap into the VMM.
  3. The VMM emulates the effect of the privileged instructions for the guest OS and return to guest.



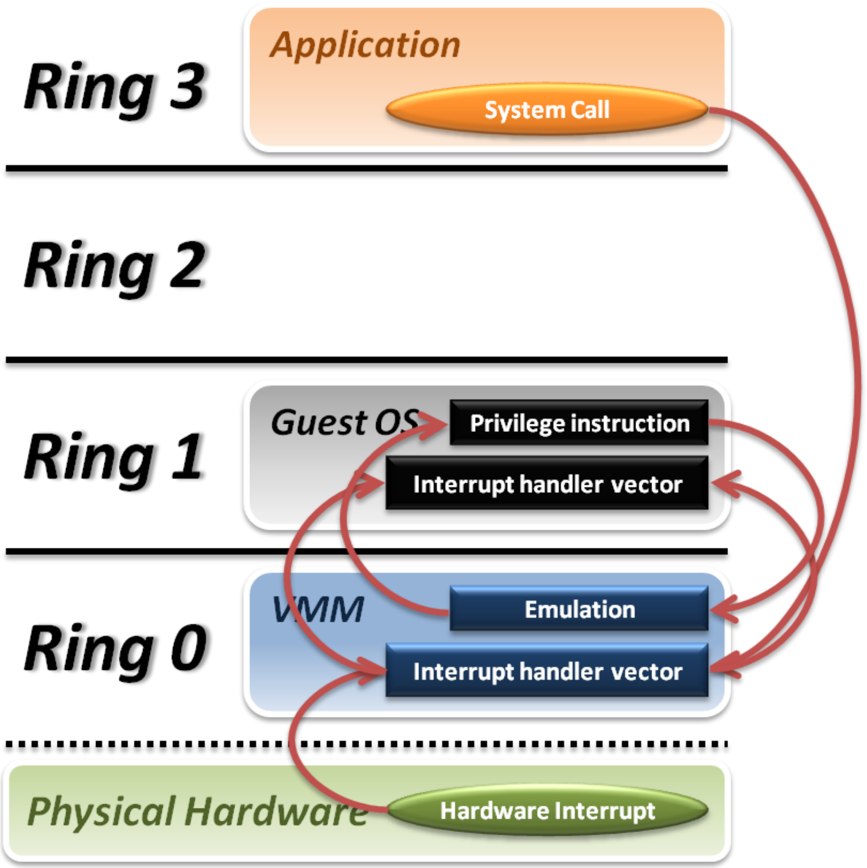
Trap and Emulate Model

- Traditional OS :
  - When application invoke a system call :
    - CPU will trap to interrupt handler vector in OS.
    - CPU will switch to kernel mode (Ring 0) and execute OS instructions.
  - When hardware event :
    - Hardware will interrupt CPU execution, and jump to interrupt handler in OS.

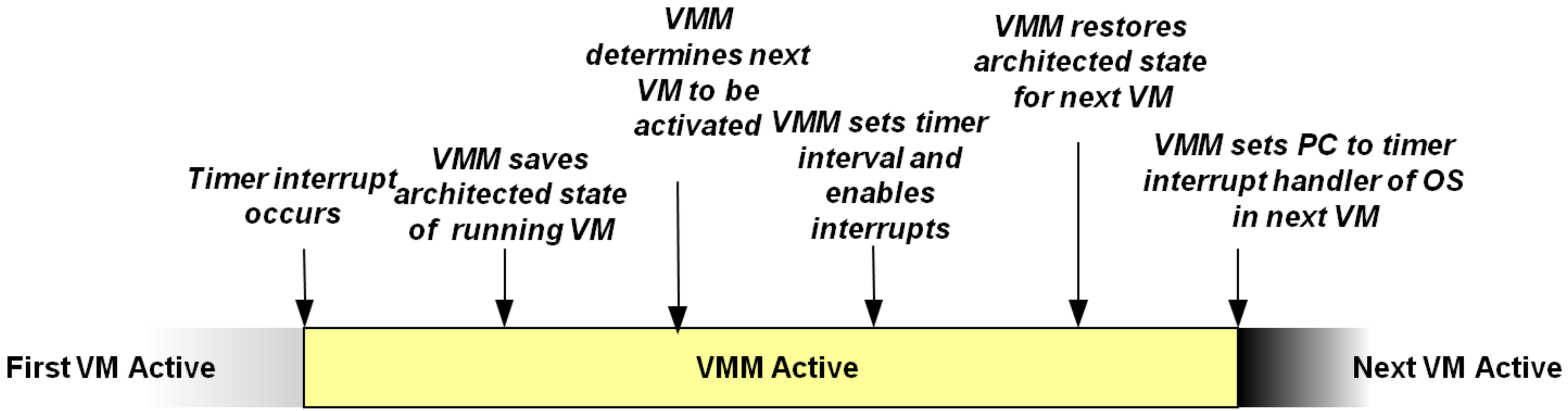


Trap and Emulate Model

- VMM and Guest OS :
  - System Call
    - CPU will trap to interrupt handler vector of VMM.
    - VMM jump back into guest OS.
  - Hardware Interrupt
    - Hardware make CPU trap to interrupt handler of VMM.
    - VMM jump to corresponding interrupt handler of guest OS.
  - Privilege Instruction
    - Running privilege instructions in guest OS will be trapped to VMM for instruction emulation.
    - After emulation, VMM jump back to guest OS.

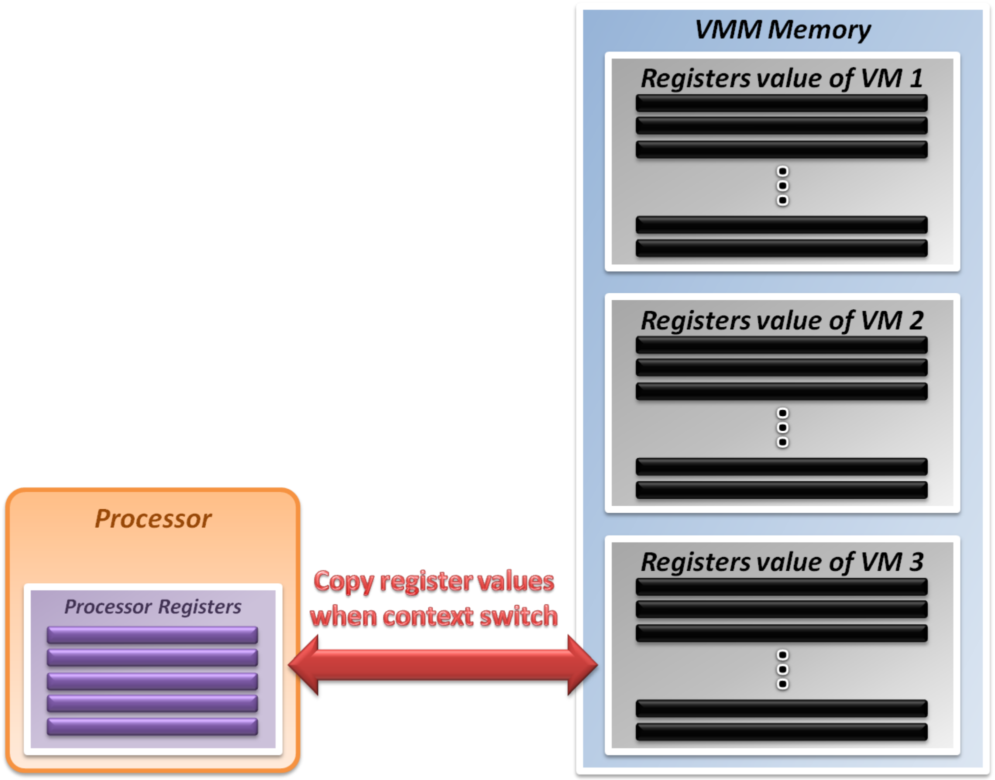


Context Switch



System State Management

- Virtualizing system state :
  - VMM will hold the system states of all virtual machines in memory.
  - When VMM context switch from one virtual machine to another
    - Write the register values back to memory
    - Copy the register values of next guest OS to CPU registers.

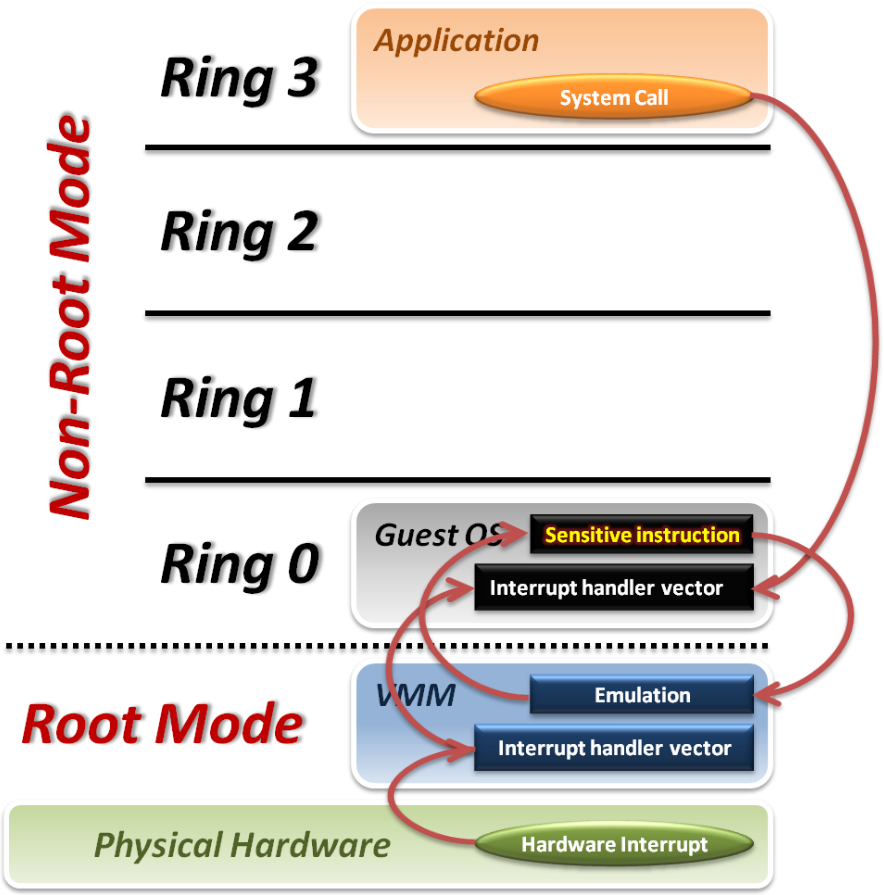


Intel VT-x

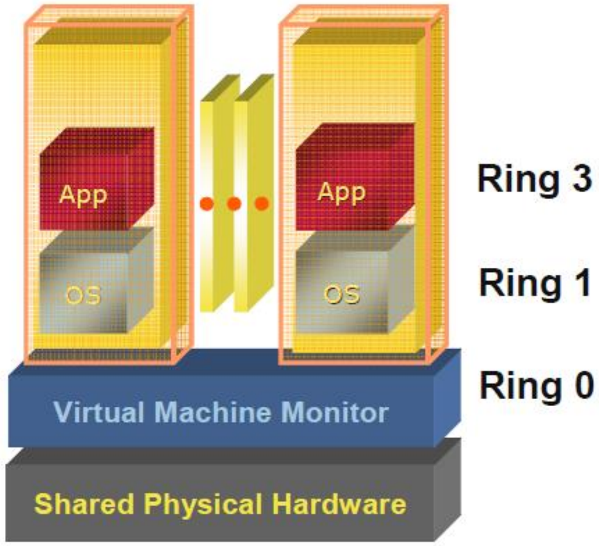
- In order to straighten those problems out, Intel introduces one more operation mode of x86 architecture.
  - VMX Root Operation (Root Mode)
    - All instruction behaviors in this mode are no different to traditional ones.
    - All legacy software can run in this mode correctly.
    - VMM should run in this mode and control all system resources.
  - VMX Non-Root Operation (Non-Root Mode)
    - All sensitive instruction behaviors in this mode are redefined.
    - The sensitive instructions will trap to Root Mode.
    - Guest OS should run in this mode and be fully virtualized through typical "*trap and emulation model*".

Intel VT-x

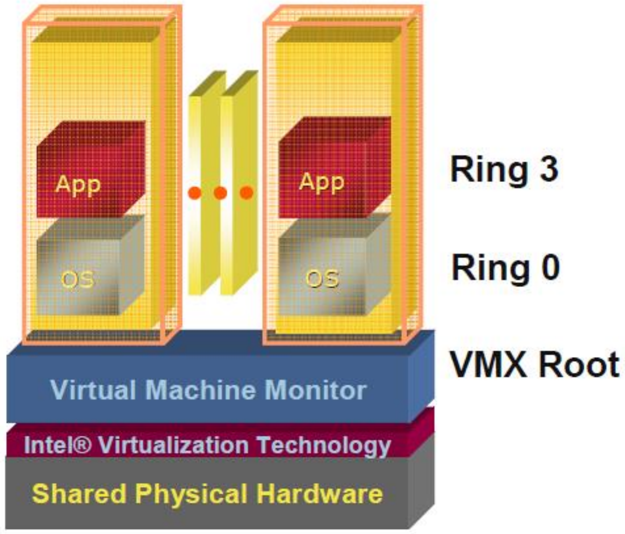
- VMM with VT-x :
  - System Call
    - CPU will directly trap to interrupt handler vector of guest OS.
  - Hardware Interrupt
    - Still, hardware events need to be handled by VMM first.
  - Sensitive Instruction
    - Instead of trap all privilege instructions, running guest OS in Non-root mode will trap sensitive instruction only.



Pre & Post Intel VT-x



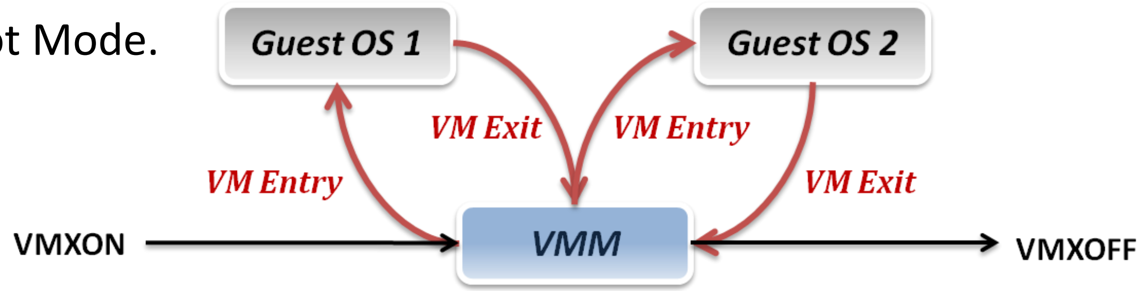
- VMM de-privileges the guest OS into Ring 1, and takes up Ring 0
- OS un-aware it is not running in traditional ring 0 privilege
- Requires compute intensive SW translation to mitigate



- VMM has its own privileged level where it executes
- No need to de-privilege the guest OS
- OSes run directly on the hardware

Context Switch

- VMM switch different virtual machines with Intel VT-x :
  - `VMXON/VMXOFF`
    - These two instructions are used to turn on/off CPU Root Mode.
  - VM Entry
    - This is usually caused by the execution of `VMLAUNCH/VMRESUME` instructions, which will switch CPU mode from Root Mode to Non-Root Mode.
  - VM Exit
    - This may be caused by many reasons, such as hardware interrupts or sensitive instruction executions.
    - Switch CPU mode from Non-Root Mode to Root Mode.



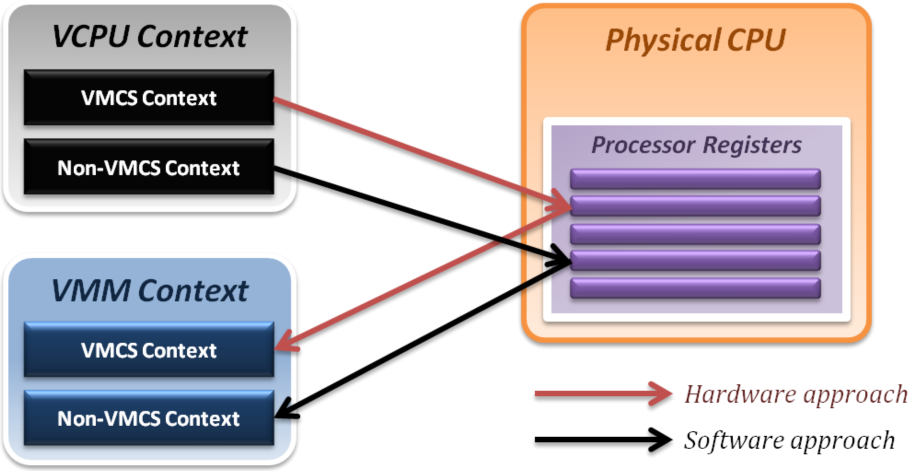
System State Management

- Intel introduces a more efficient hardware approach for register switching, **VMCS** (*Virtual Machine Control Structure*) :
  - State Area
    - Store host OS system state when VM-Entry.
    - Store guest OS system state when VM-Exit.
  - Control Area
    - Control instruction behaviors in Non-Root Mode.
    - Control VM-Entry and VM-Exit process.
  - Exit Information
    - Provide the VM-Exit reason and some hardware information.

- Whenever VM Entry or VM Exit occur, CPU will automatically read or write corresponding information into VMCS.

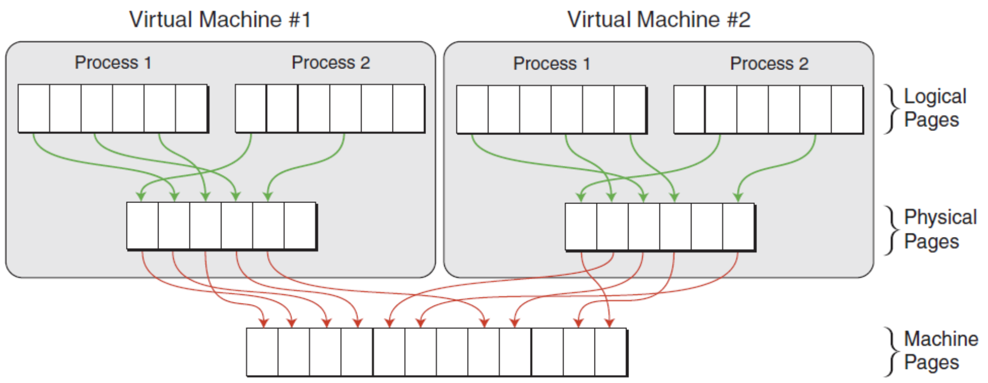
System State Management

- Binding virtual machine to virtual CPU
  - VCPU (Virtual CPU) contains two parts
    - VMCS maintains virtual system states, which is approached by hardware.
    - Non-VMCS maintains other non-essential system information, which is approach by software.
  - VMM needs to handle Non-VMCS part.

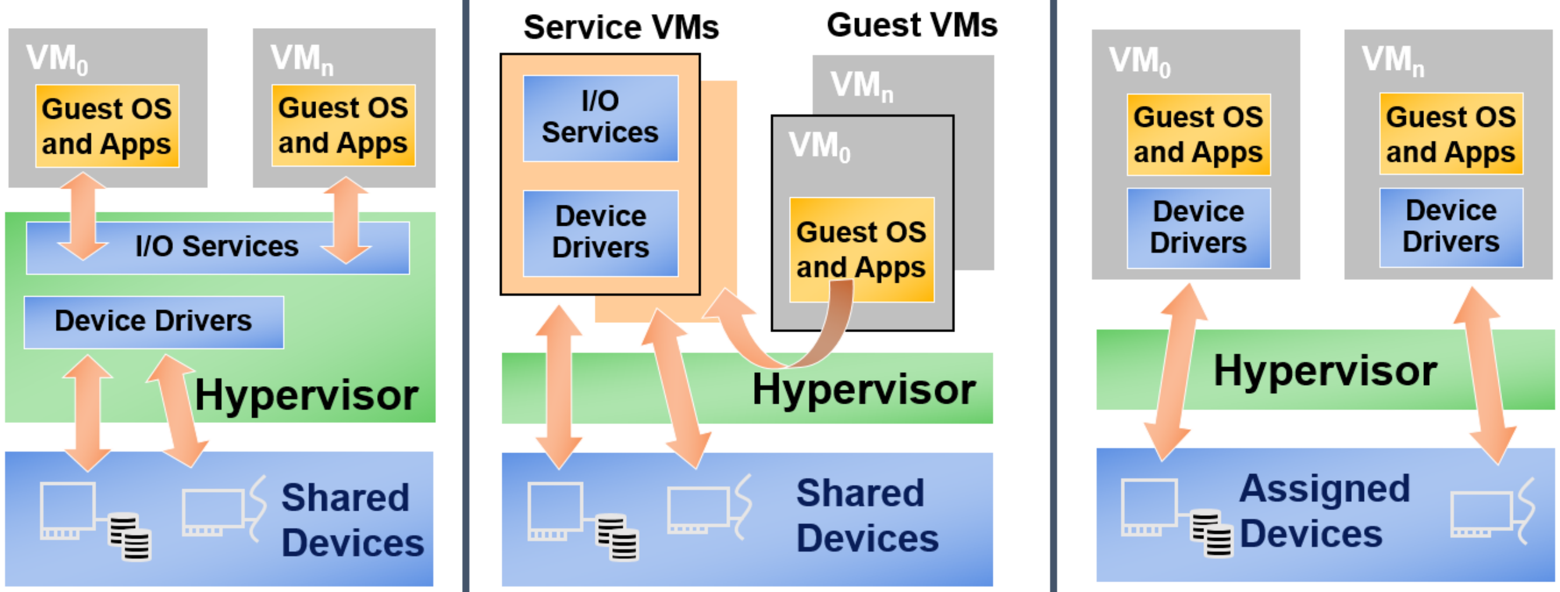


Memory Virtualization: Extended Page Table

- Concept of Extended Page Table (EPT) :
  - Instead of walking along with only one page table hierarchy, EPT technique implement one more page table hierarchy.
    - One page table is maintained by guest OS, which is used to generate guest physical address.
    - The other page table is maintained by VMM, which is used to map guest physical address to host physical address.
  - For each memory access operation, EPT MMU will directly get guest physical address from guest page table, and then get host physical address by the VMM mapping table automatically.



Options For I/O Virtualization



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

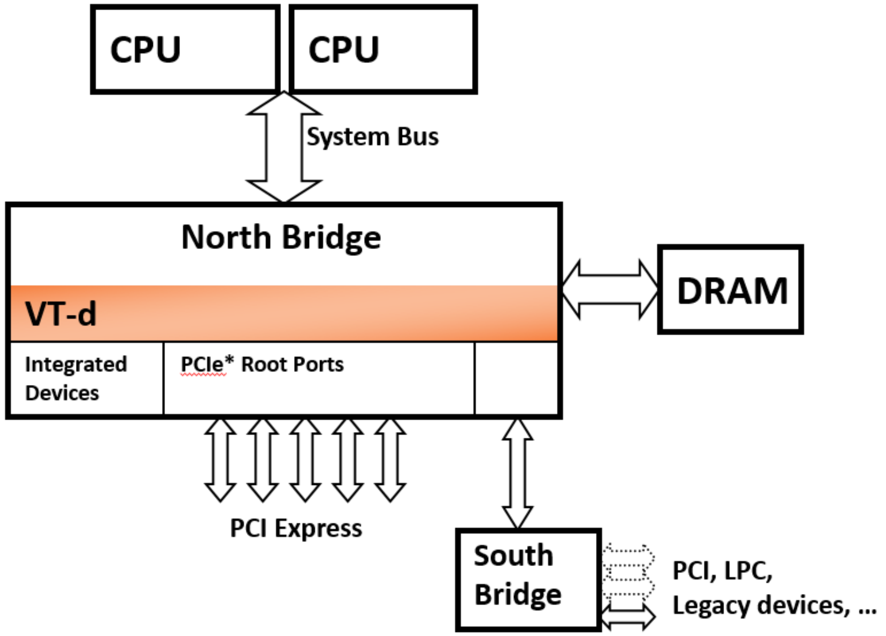
Pass-through Model

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

VT-d Goal: Support all Models

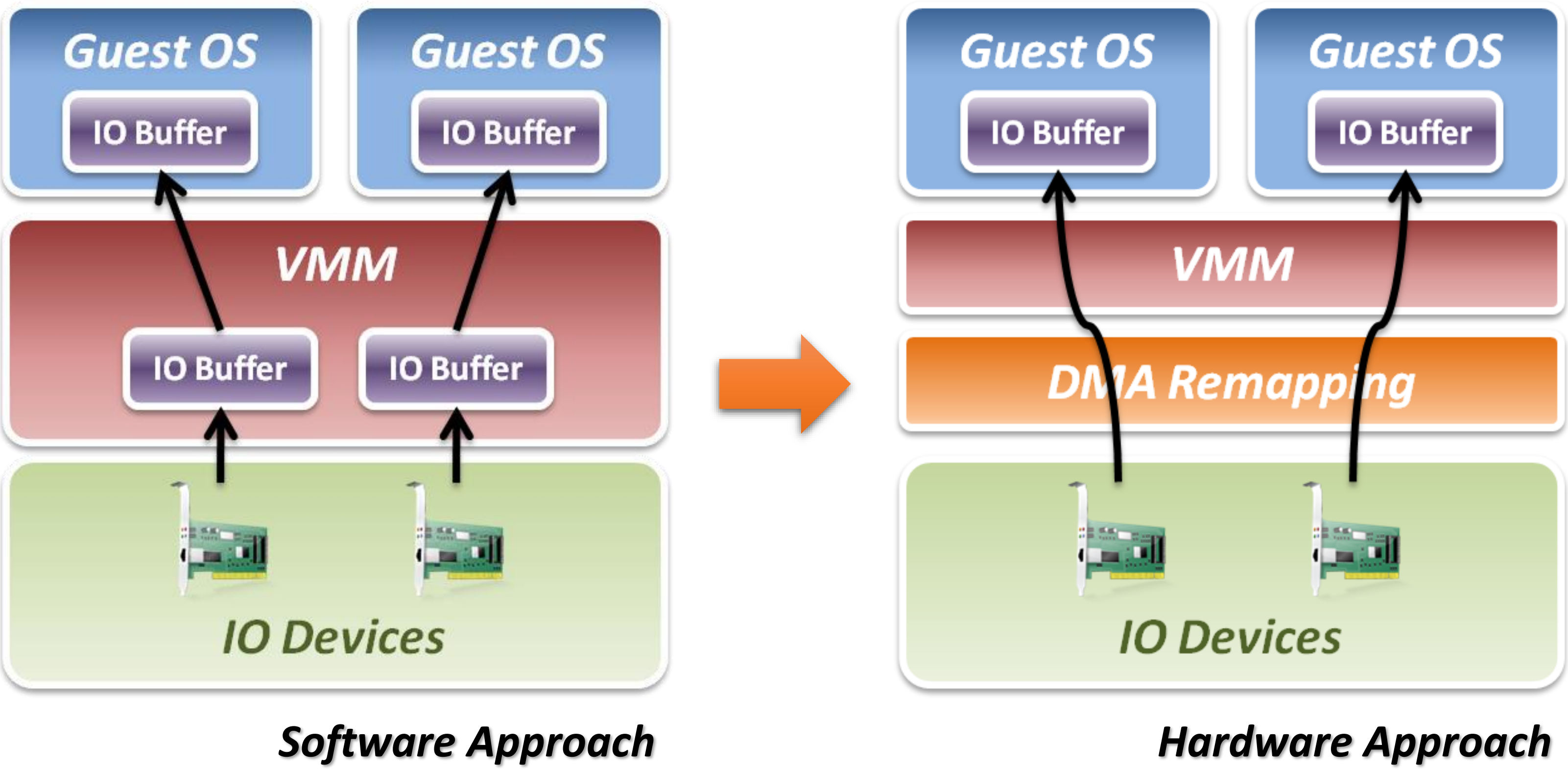
VT-d Overview

- VT-d is platform infrastructure for I/O virtualization
  - Defines architecture for DMA remapping
  - Implemented as part of platform core logic
  - Will be supported broadly in Intel server and client chipsets

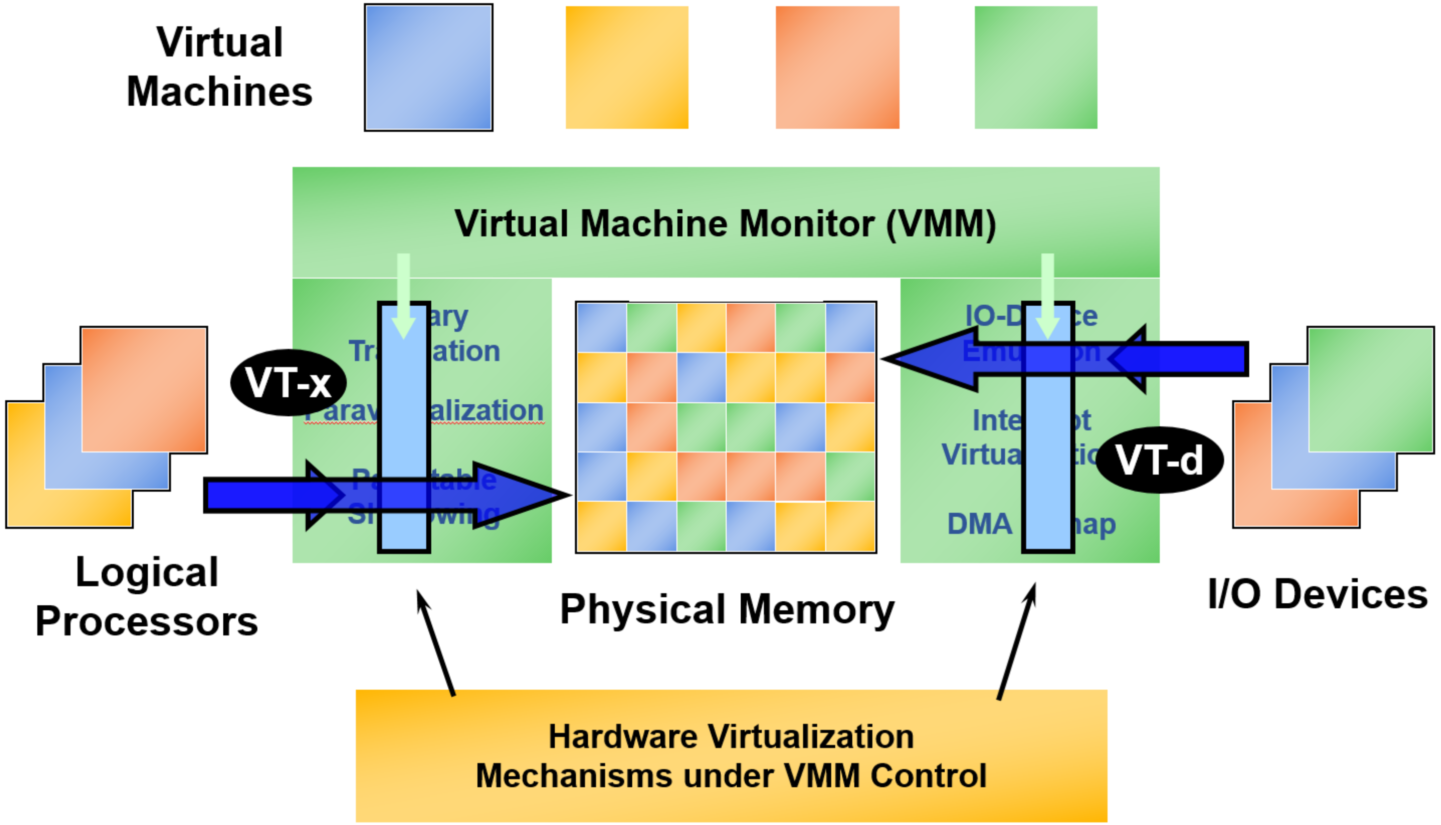


Intel VT-d

- Add DMA remapping hardware component.



VT-x & VT-d Working Together



Summary

- CPU Virtualization
  - Trap and Emulate Model
  - Virtualization technique, VMX Root/Non-Root Operation, VMM and Guest OS, VMCS … etc.
- Memory Virtualization: Extended Page Tables (EPT)
  - EPT implement one more page table hierarchy
  - MMU virtualize, EPT translation, Memory Operation,… etc.
- IO Virtualization: Intel VT-d
  - Implement DMA remapping in hardware
  - Hardware Page Walk, Translation Caching

VMware Virtual Platform

What Is VMware Virtual Platform?

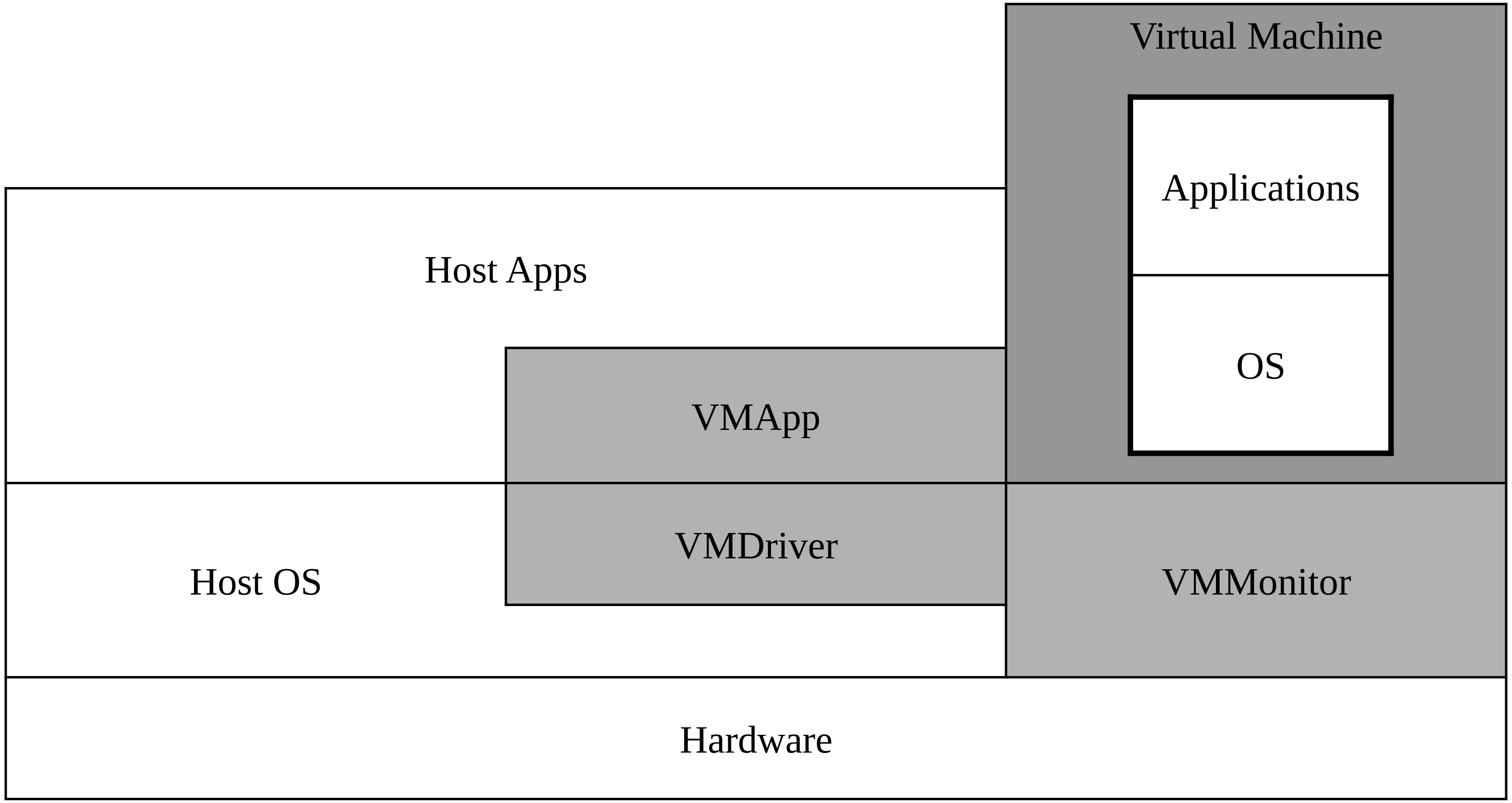
- A popular virtual machine infrastructure for IA-32-based machines
- Available both as a hosted VM system (VMware Server – formerly GSX Server) and native VM system (VMware ESX Server)

- Discussion will be limited to the hosted system only

VMware Components

- Three components:
  - VMMonitor – running in privileged mode, responsible for most of the virtualization tasks
  - VMApp – running as an application on the host OS, acting as a mediator requesting services from the host OS
  - VMDriver – installed as a host OS driver, facilitating the control transfer between the host OS and VMMonitor

VMware Components



Processor Virtualization

- IA-32 architecture is not efficiently virtualizable
  - There are 17 instructions that are critical (sensitive, but not privileged)
- The VM needs to be hybridized
  - Runs natively, but needs scanning and patching of critical instructions first

IA-32 Critical Instructions

- The 17 critical instructions fall into 2 broad categories:
  - Protection system references – instructions that reference the storage protection system, memory system, or address relocation system
    - E.g., **`mov ax, cs`** in user mode generates a no-op
  - Sensitive register instructions – instructions that attempt to read or change resource-related registers and memory locations
    - E.g., **`popf`**, which could potentially change IF flag

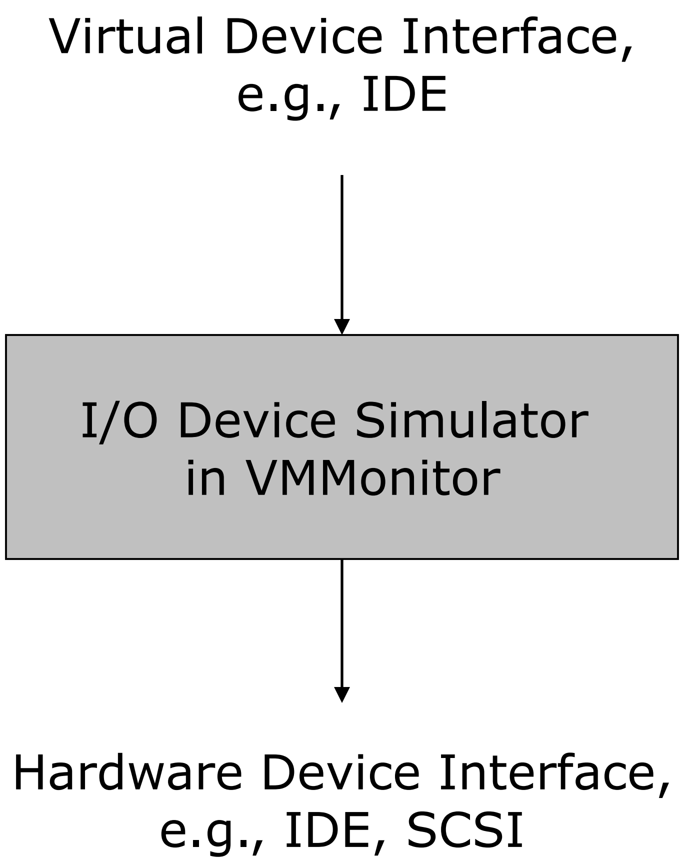
Input/Output Virtualization

- Difficulty of virtualizing I/O in IA-32-based platforms lead VMware developers to adopt dual-mode hosted VM style
- VMware utilizes multiple approaches to I/O virtualization:
  - Emulation in VMMonitor
  - Using the services of the host OS
  - New capability for devices through abstraction layer

Emulation in VMMonitor

- If a physical counterpart exists, emulation simply involves converting the parameters in some virtual device interface (VDI) into the actual hardware device interface (HDI)
- Done by intercepting **`in`** and **`out`** I/O instructions in VMMonitor and converting to an appropriate instruction sequence

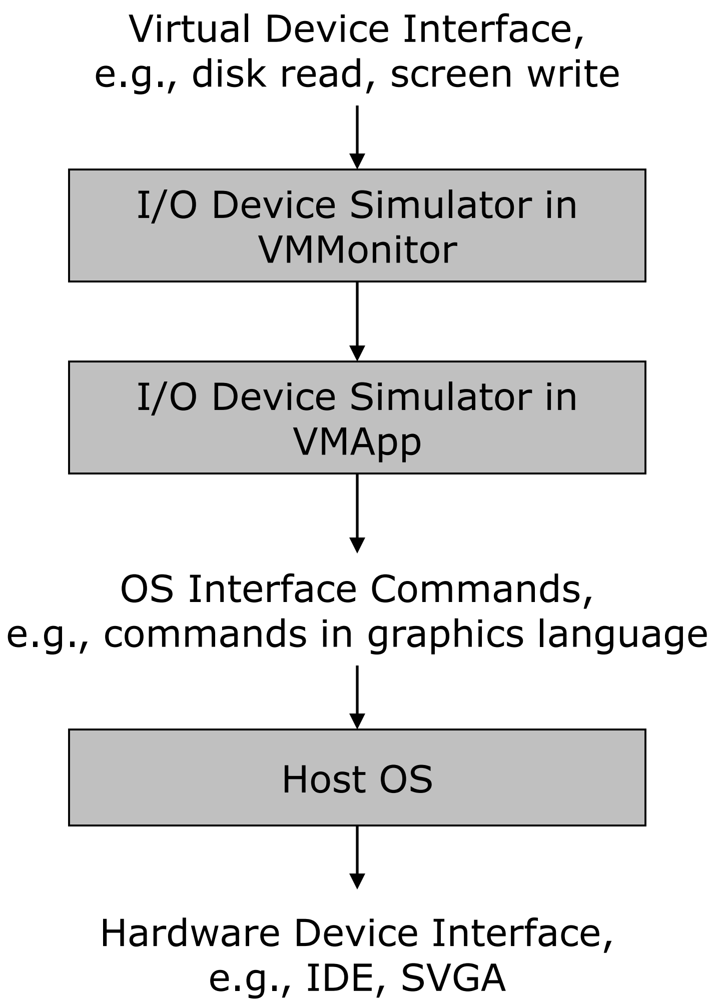
Mapping of Device Interface



Using the Services of the Host OS

- Some devices are not well-standardized and are not suitable for the former approach
- VMMonitor intercepts requests, but pass them on to the host OS through VMApp using the OS system calls

Mapping through OS



Added Benefits

- All devices supported by the host OS are automatically available to the VMMonitor
- VMMonitor can also take advantage of other services provided by the host OS, not just the I/O features
- Transition from an old OS to a new OS may benefit from this approach, using the old OS as the host OS

Adding New Capabilities

- It is also possible to enhance the functions of the original devices by an abstraction layer in VMMonitor
  - Undoable disk
    - The VM disk can be viewed either as a disk or a file on the host VM
    - Explicit commit command to save the session
  - A virtual Ethernet switch can be emulated within VMware to provide direct communication between VMs and/or host OS
  - Alternative UI
    - Windowed or full-screen mode

Memory Virtualization

- VMMonitor virtualizes the physical memory of a VM by using the host OS to allocate or release the physical memory
- Paging requests are not directly intercepted, but the disk read/writes are translated by VMMonitor and carried out on the host OS through VMApp
- The host could replace critical pages with pages from other host applications
- VMDriver pins some critical pages

END