

Introduction to Virtualization and Virtual Machines

Kartik Gopalan

From

“Virtual Machines” ,Smith and Nair, Chapter 1

Also, Chapter 7 Andrew Tanenbaum’s book

“Virtual machines have finally arrived. Dismissed for a number of years as merely academic curiosities, they are now seen as cost-effective techniques for organizing computer systems resources to provide extraordinary system flexibility and support for certain unique applications”. —Robert. P. Goldberg, IEEE Computer, 1974

The Rise and Fall of Virtual Machines

- 1970s:
 - IBM mainframes run VMs and hardware supports not just single-level, but also nested virtualization.
 - Hundreds of papers on virtualization.
 - Goldberg and Popek's seminal paper on formal requirements of virtualization support.
- 1980s and 1990s:
 - Personal computers (temporarily) obviate the need for VMs.
 - x86, MIPS, Sparc processors have no hardware support for virtualization
 - UNIX, Windows, Linux don't include virtualization support.

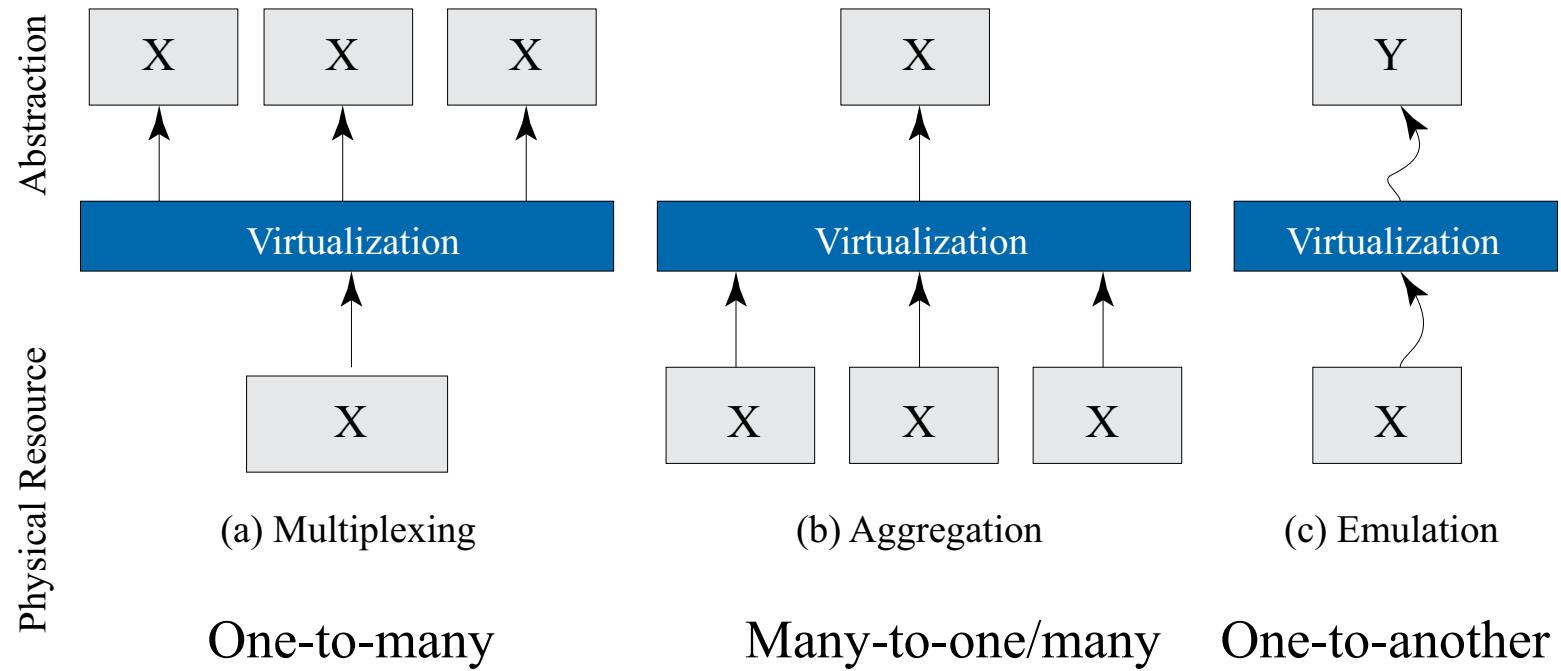
VMs — Back from Dead

- 1997: DISCO paper revisits VMs for running commodity OS on SMP machines
- 1999: VMWare releases VMWare workstation for x86 CPUs.
- Early 2000s:
 - VMWare ESX server, VirtualPC, Xen, Denali, Cells.
 - Still no hardware support for virtualization
- Late 2000s
 - Major chip vendors add hardware support
 - AMD-V, Intel VT-x, VT-d etc.
 - Storage virtualization becomes the norm
 - Turtles paper: Nested Virtualization revisited
- 2010s:
 - Multi-billion \$ cloud industry.
 - Network & storage virtualization are the norm
 - All cloud platforms are virtualized.
 - Confidential computing hardware
 - SGX, Trustzone, SGX, TDX
- 2020s:
 - AI, IoT, and Security are shaping the next generation virtualization technologies for the cloud

Virtualization

- Virtualization is a key technology that enables cloud computing.
- Provides a way to abstract and share physical resources.
 - Makes a real system appear to be a set of virtual systems.
- Any system resource can be virtualized
 - Computation (CPUs, GPUs, xPUs)
 - Storage (memory, disk)
 - Network (local, wide area)
 - Any other I/O (keyboard, mouse, display, sound?)

Types of Virtualization



Types of Virtualization - examples

- Multiplexing: One-to-many virtualization
 - E.g. one physical machine may appear as multiple virtual machines
 - one physical disk may look like multiple virtual disk
 - one physical network may look like multiple virtual networks
- Aggregation: Many-to-one/many virtualization
 - Many physical machines/disks/networks may appear to look like one virtual machine/disk/network etc
- Emulation: One-to-another virtualization
 - Run x86 VM on a PowerPC processor
 - Run Windows OS on a Linux OS, Android emulators
 - Run Java VM on any hardware

Virtualization = Abstraction + Multiplexing

Abstraction

- Abstraction hides unnecessary lower-layer details from upper layers, so that the upper-layers can focus on the essential aspects of their functionality.
- Examples:
 - A Process abstracts one unit of (computation + memory + I/O)
 - File systems abstract the storage hardware
 - VM abstracts an entire machine with (Hardware + OS + Processes)
 - Programming languages abstract how data is stored and manipulated.
 - User interfaces abstract how a program is implemented.

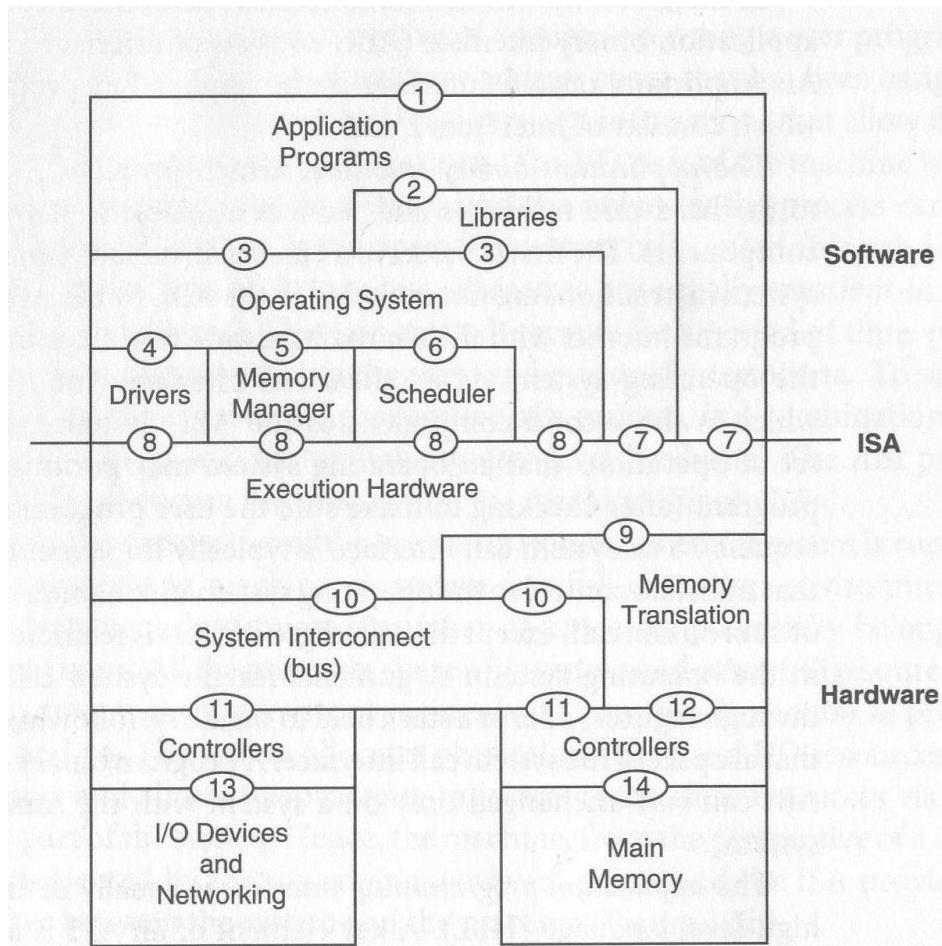
Multiplexing: Trading time for space and money

- Sharing one resource among many users
- E.g.
 - Teller at a bank : Space over waiting time
 - Long Distance Trunks : Bandwidth over queuing delay.
 - One CPU shared by multiple processes: Space and money for waiting time
- Temporal vs. Spatial multiplexing
 - Temporal = Share time (e.g. CPU or network link)
 - Spatial = Share space (e.g. memory)
- A server (scheduler) controls access to the resource
 - Boarding the plane
 - CPU scheduler
 - Network link scheduling
 - Memory manager
 - TA Office hours
- Statistical Multiplexing
 - Overcommitting (overbooking) a resource given some probability that not all allocations are fully utilized
 - Doctor's appointment schedule
 - Airplane seats
 - Virtual memory

Virtual Machines

- Logical/Emulated representations of full computing system environment
 - CPU + memory + I/O
 - Implemented by adding layers of software to the real machine to support the desired VM architecture.
- Uses:
 - Multiple OSes on one machine, including legacy OSes
 - Isolation
 - Enhanced security
 - Live migration of servers
 - Virtual environment for testing and development
 - Platform emulation
 - On-the-fly optimization
 - Realizing ISAs not found in physical machines

Interfaces of a computer system



User ISA : 7

System ISA : 8

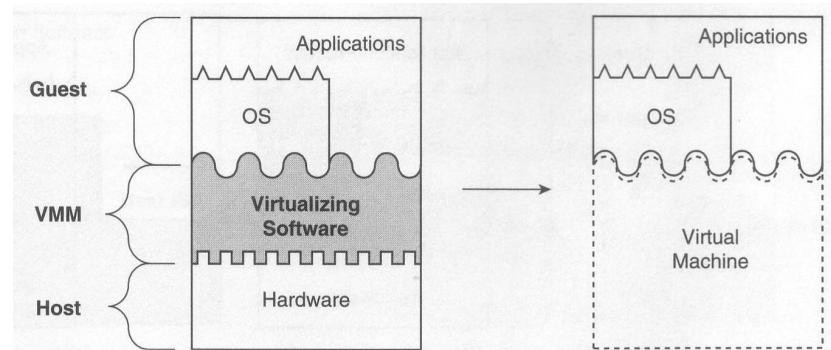
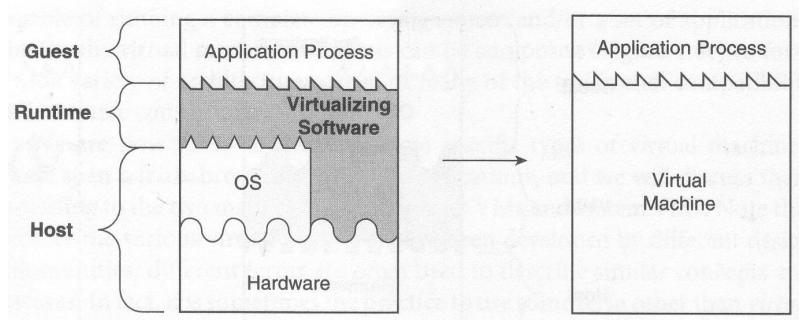
Syscalls : 3

ABI : 3, 7

API : 2,7

Two Types of VMs

- Process VM
 - Virtualizes the ABI
 - Virtualization software = Runtime
 - Runs in non-privileged mode (user space)
 - Performs binary translation.
 - Terminates when guest process terminates.
- System VM
 - Virtualizes the ISA
 - Virtualization software = Hypervisor
 - Runs in privileged mode
 - Traps and emulates privileged instructions



Process Virtual Machines

❑ Process in a multiprogramming OS

- Standard OS syscall interface + instruction set
- Multiple processes, each with its own address space and virtual machine view.

❑ Emulators

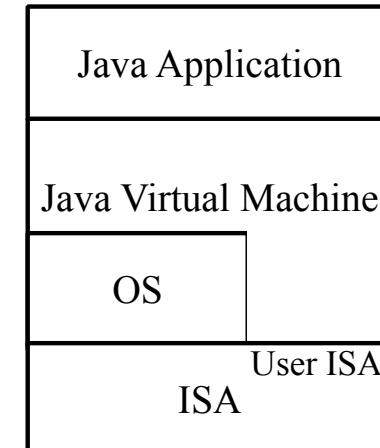
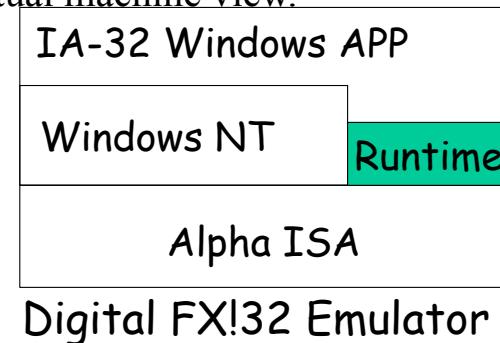
- Support one ISA on hardware designed for another ISA
- Interpreter:
 - Fetches, decodes and emulates individual instructions. Slow.
- Dynamic Binary Translator:
 - Blocks of source instructions converted to target instructions.
 - Translated blocks cached to exploit locality.

❑ Same ISA Binary Optimizers

- Optimize code on the fly
- Same as emulators except source and target ISAs are the same.

❑ High-Level Language VMs

- Virtual ISA (bytecode) designed for platform independence
- Platform-dependent VM executes virtual ISA
- E.g. Sun's JVM and Microsoft's CLI (part of .NET)
- Both are stack-based VMs that run on register-based m/c.

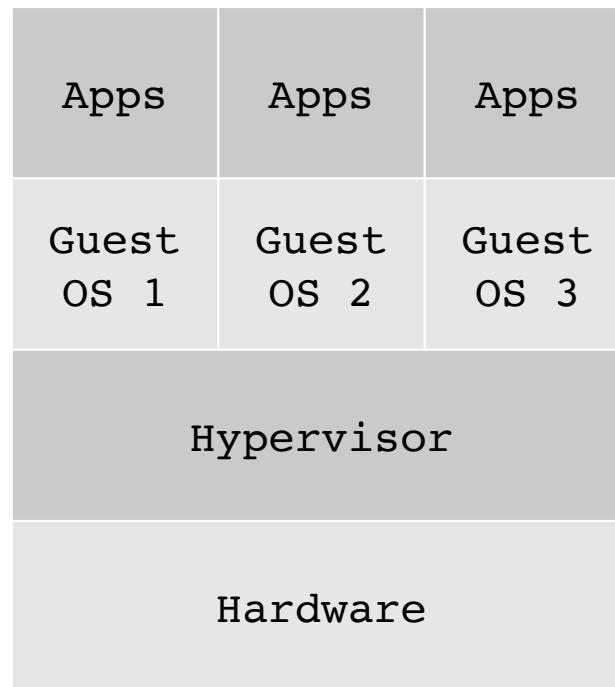


System Virtual Machines

(focus of this lecture)

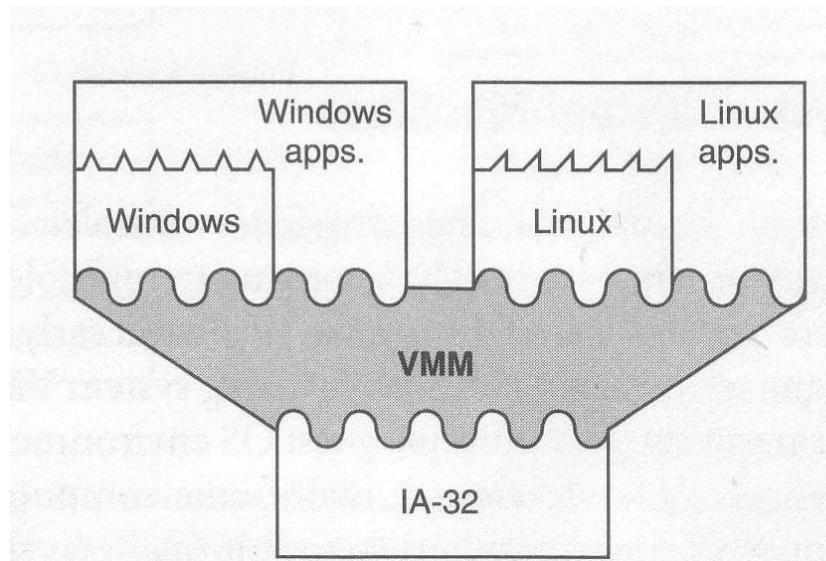
Hypervisor

- ❑ Also called Virtual Machine Monitor (VMM)
- ❑ A hypervisor is an operating system for operating systems
 - Provides a virtual execution environment for an entire OS and its applications
 - Controls access to hardware resources
 - When guest OS executes a privileged instruction, Hypervisor intercepts the instruction, checks for correctness and emulates the instruction.



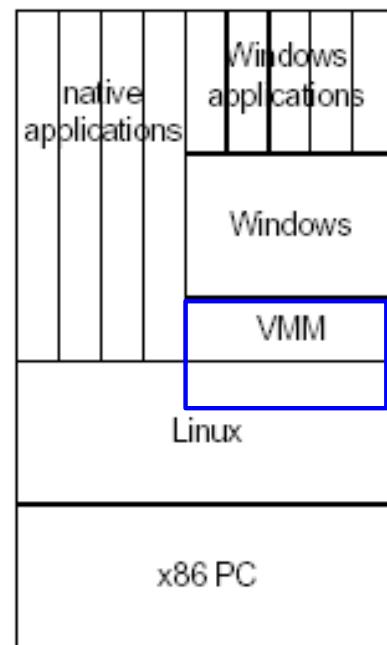
Type 1 Hypervisors (Classical System VMs)

- ❑ Hypervisor executes natively on the host ISA
- ❑ Hypervisor directly controls hardware and provides all device drivers
- ❑ Hypervisor emulates sensitive instructions executed by the Guest OS
- ❑ E.g. KVM and VMWare ESX Server



Type 2 Hypervisors (Hosted VMs)

- A host OS controls the hardware
- The Hypervisor runs partly in process space and partly in the host kernel
- Hypervisor Relies on host OS to provide drivers
- E.g. VMWare Desktop Client

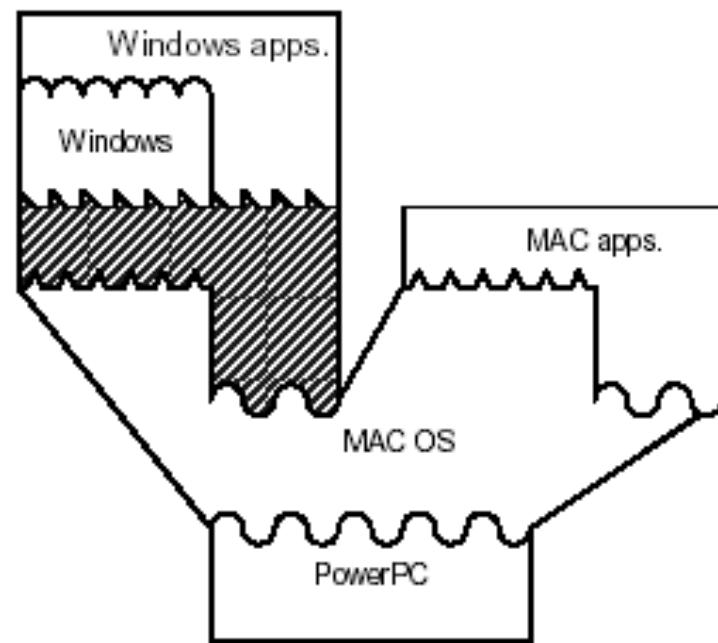


Para-virtualized VMs

- ❑ Modify guest OS for better performance
- ❑ Traditional Hypervisors provide full-virtualization
 - They expose to VMs virtual hardware that is functionally identical to the underlying physical hardware.
 - Advantage : allows unmodified guest OS to execute
 - Disadvantage: Sensitive instructions must be trapped and emulated by Hypervisor.
 - E.g. KVM and VMWare ESX provide full virtualization
- ❑ Para-virtualized VM
 - Sees a virtual hardware abstraction that is similar, but not identical to the real hardware.
 - Guest OS is modified to replace sensitive instructions with “hypcalls” to the Hypervisor.
 - Advantage: Results in lower performance overhead
 - Disadvantage: Needs modification to the guest OS.
 - E.g. Xen provides both para-virtual as well as full-virtualization
- ❑ Often traditional Hypervisors are partially para-virtualized
 - ❑ Device drivers in guest OS may be para-virtualized whereas CPU and Memory may be fully virtualized.

Whole System VMs: Emulation

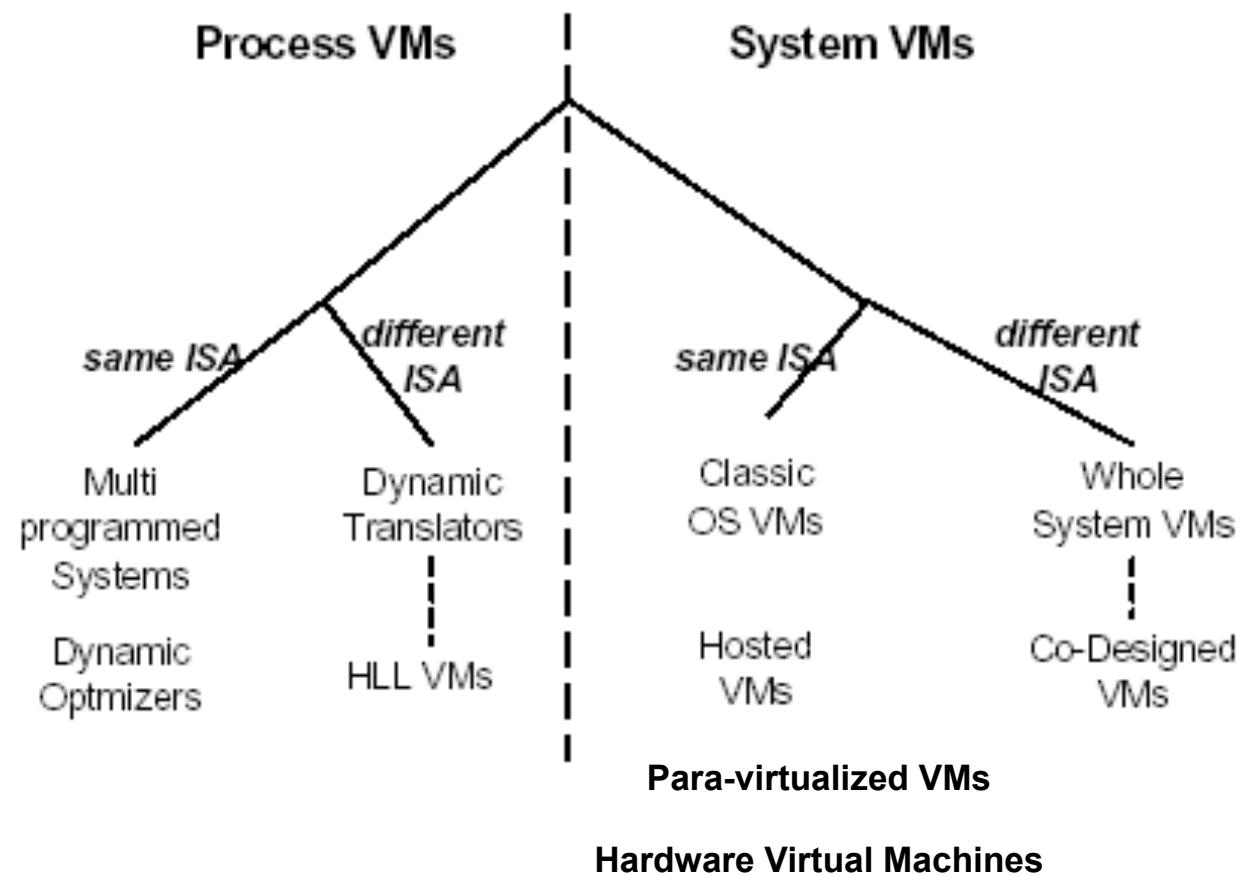
- ❑ Host and Guest ISA are different
- ❑ So emulation is required
- ❑ Hosted VM + emulation
- ❑ E.g. Virtual PC (Windows on MAC)



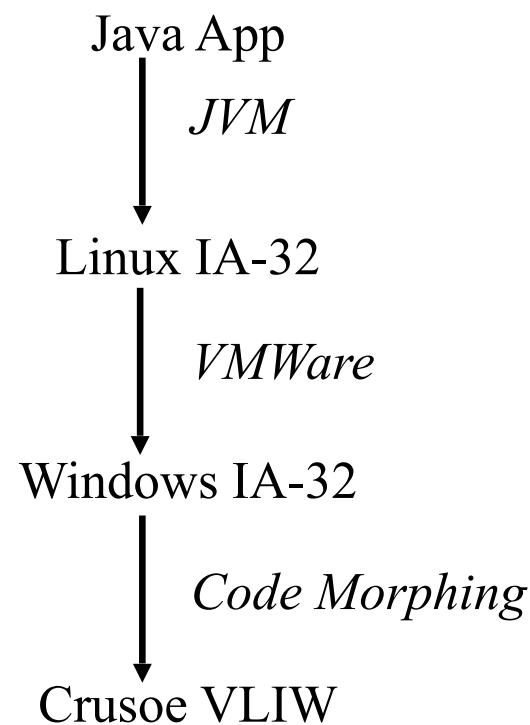
Co-designed VMs

- ❑ The hypervisor is designed closely with (and possibly built into) a specific type of hardware ISA (or native ISA).
- ❑ Goal: Performance improvement of existing ISA (or guest ISA) during runtime.
- ❑ Hypervisor performs Emulation from Guest ISA to Native ISA.
- ❑ E.g. Transmeta Crusoe
 - Native ISA based on VLIW
 - Guest ISA = x86
 - Goal power savings

Taxonomy



Versatility



Virtualizing individual resources in System VMs

Three resource types that are virtualized

- Computation in various types of processors
 - CPU, GPU, TPU, etc
- Memory
 - Primarily volatile random-access memory (DRAM)
 - But also, more recently, NVRAM
- I/O
 - Persistent storage, Network, Display, Keyboard, etc
- These three resources must be virtualized together, holistically, to provide the illusion of a complete “virtual machine”

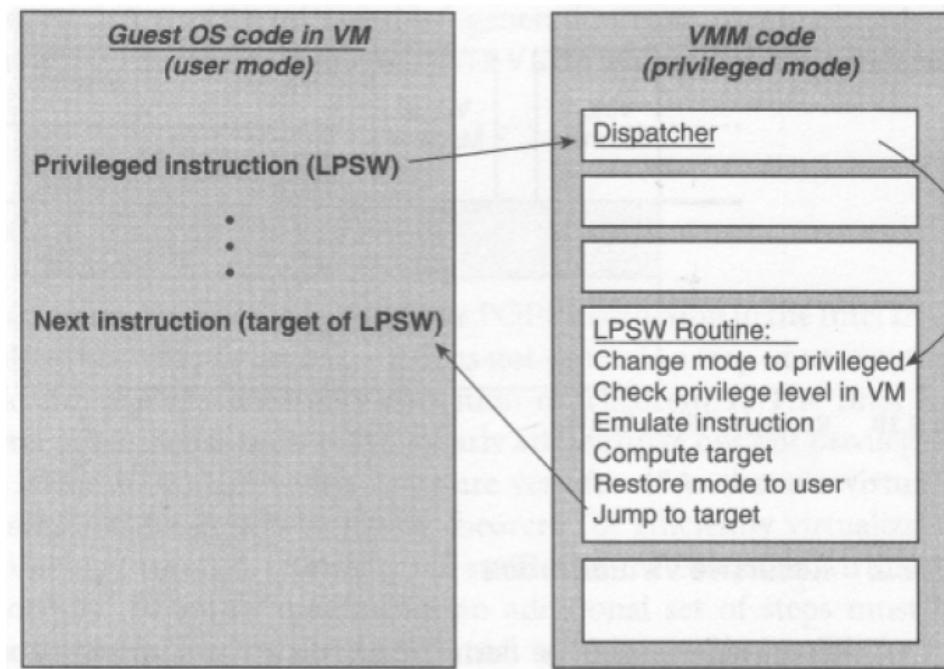
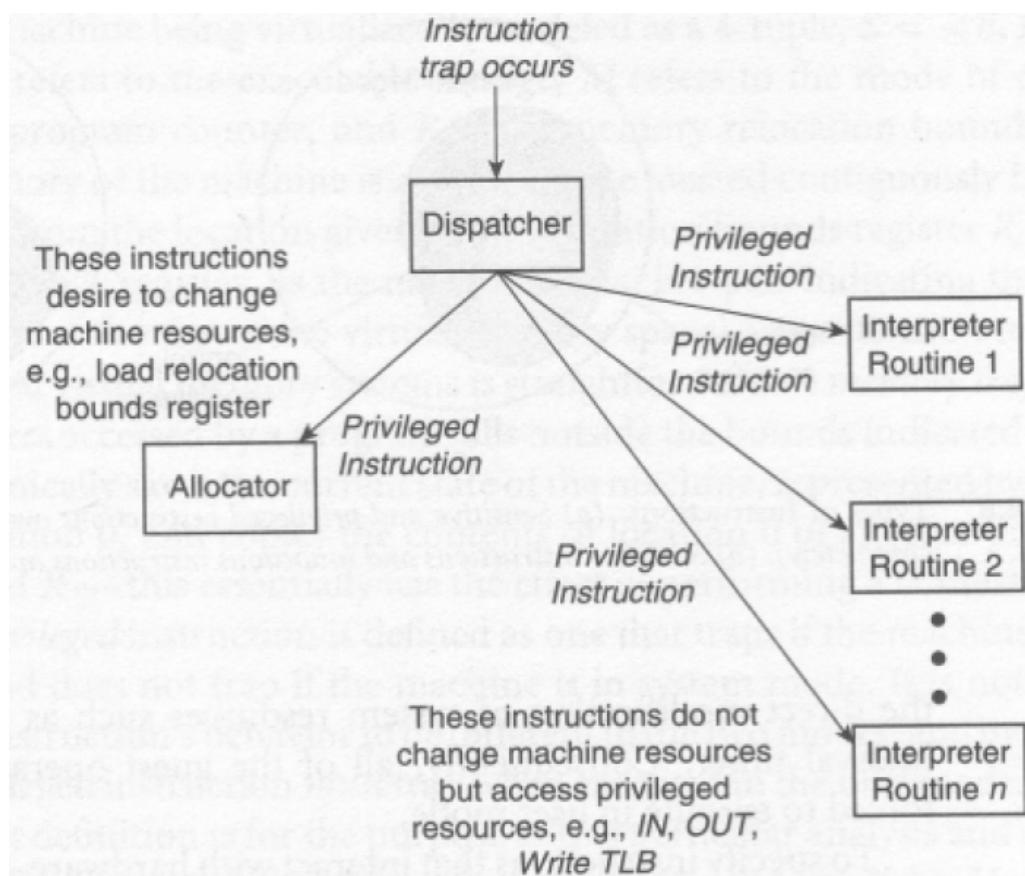
CPU Virtualization for VMs

- Each VM sees a set of “virtual CPUs”
- Hypervisors must emulate privileged instructions issued by guest OS.
- Modern ISAs provide special interfaces for Hypervisors to run VMs
 - Intel provides the VTx interface
 - AMD provides the AMD-v interface
- These special ISA interfaces allow the Hypervisors to efficiently emulate privileged instructions executed by the guest OS.
- When guest OS executes a privileged instruction
 - Hardware traps the instruction to the hypervisor
 - Hypervisor checks whether instruction must be emulated.
 - If so, Hypervisor reproduces the effect of privileged operation.

CPU Privilege Levels in Intel VTx

- Traditional non-virtualized privilege levels in x86: 0,1,2,3
 - 3 → user-level processes
 - 2 → traditionally unused
 - 1 → traditionally unused
 - 0 → operating system
- Virtualized privilege levels in x86+VTx - two orthogonal privilege levels
 - Non-root mode - For Virtual machine
 - Has four levels 0,1,2,3 as above
 - Root mode - for hypervisor
 - Has four levels 0,1,2,3 as above

Execution of Privileged Instruction by Guest



Resource Control

- Issue: How to retain control of resources in the Hypervisor?
- Timer interval control performed by Hypervisor
- Also, guest OS is not allowed to read the timer value
 - Guest OS sees a virtual interval timer
- Hypervisor also gains control whenever guest OS executes privileged instructions.

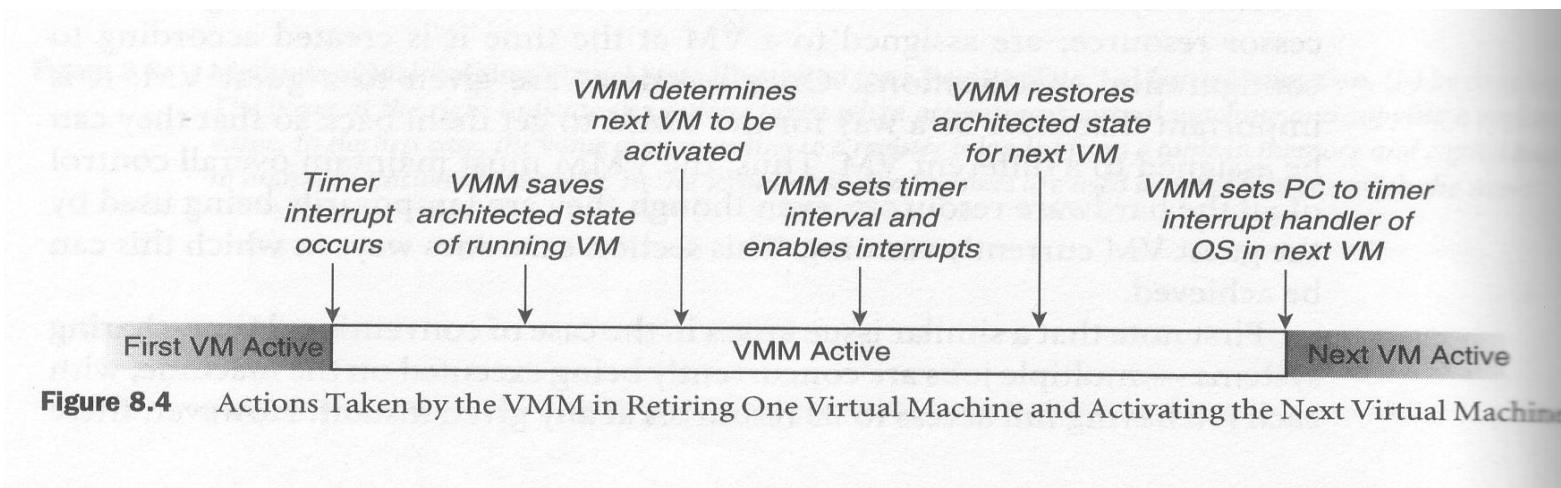
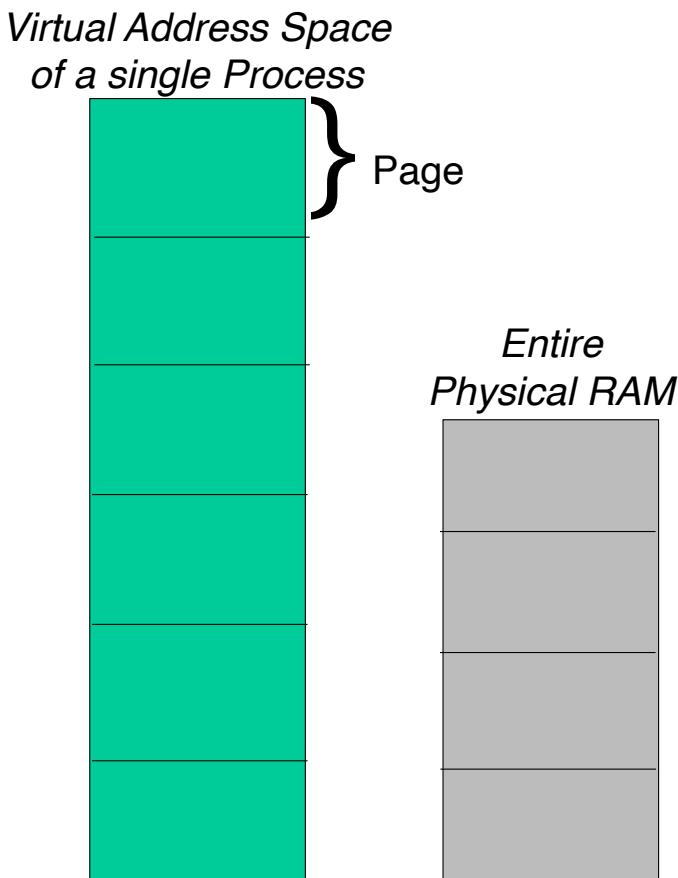


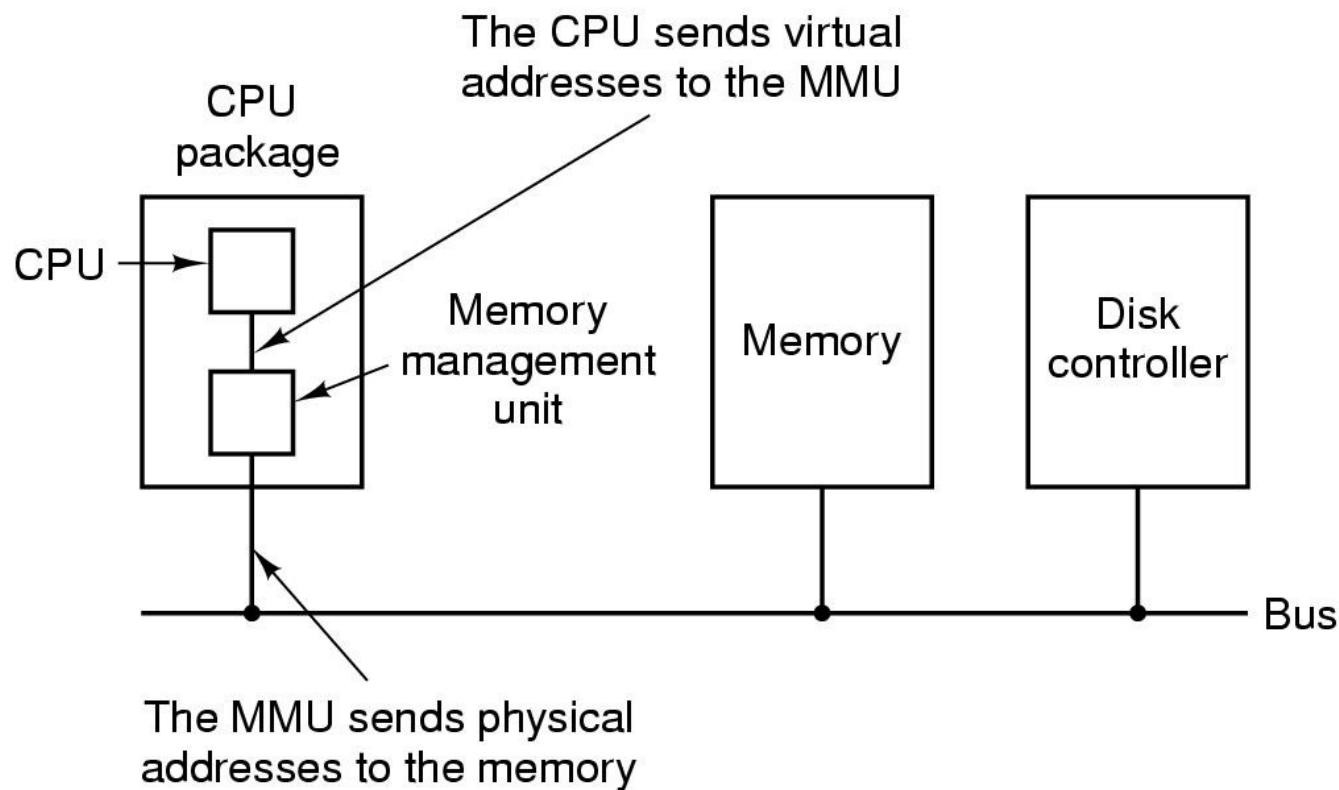
Figure 8.4 Actions Taken by the VMM in Retiring One Virtual Machine and Activating the Next Virtual Machine.

Virtual Memory Recap

- **Virtual memory**: means that each process gets an illusion that it has its own memory space whose size is independent of the size of physical RAM in the system.
- How? Break up the memory space of a process into equal-sized PAGES.
 - Typically, a page = 4KiB
- Memory is allocated to processes at the granularity of pages
 - E.g. 4KiB, 8KiB, 12KiB etc.
- OS then decides which pages stay in memory and which get paged (moved) out to disk.



Memory Management Unit

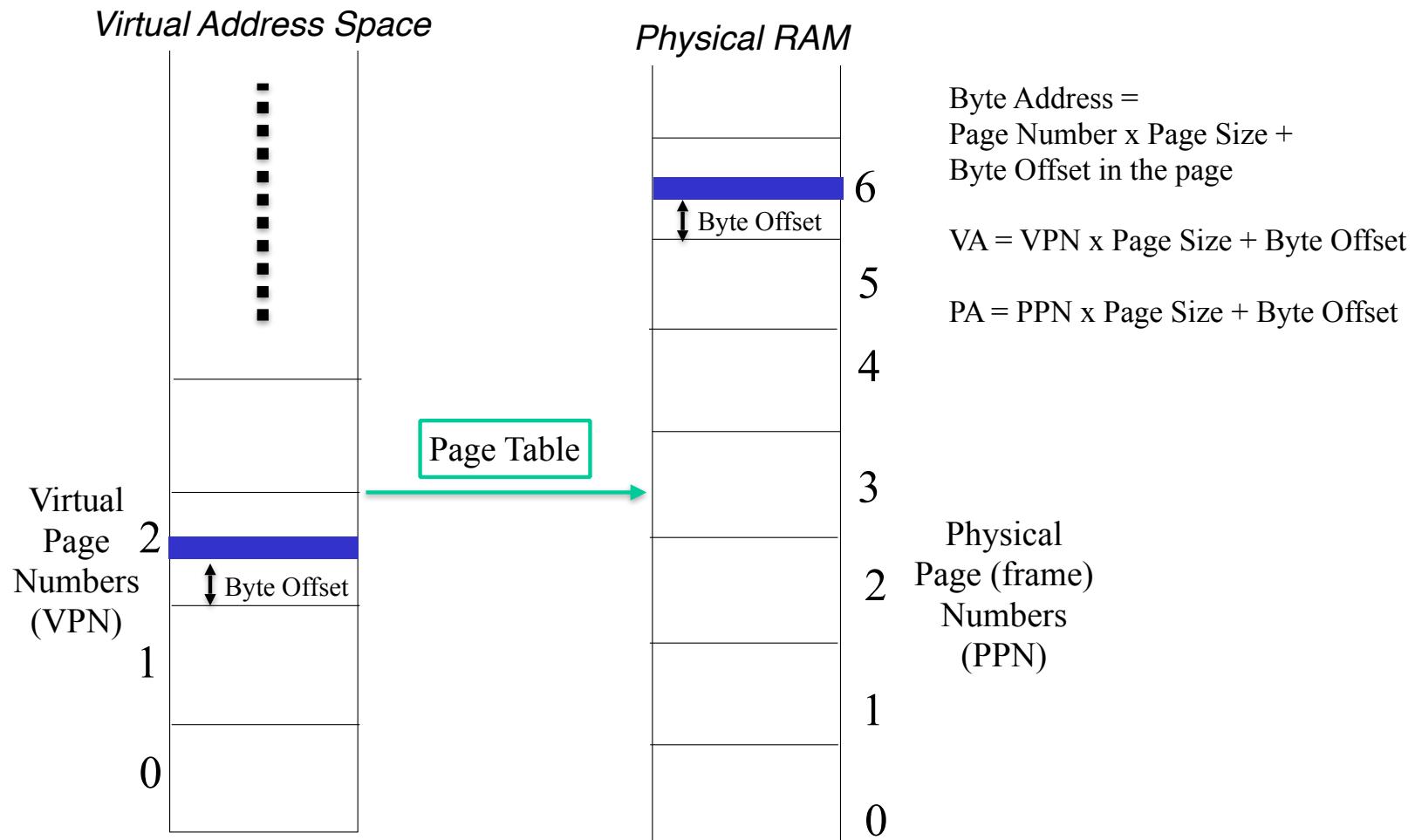


- MMU is a hardware module that accompanies the CPU
- It translates the Virtual Address used by executing instructions to Physical Addresses in the main memory.

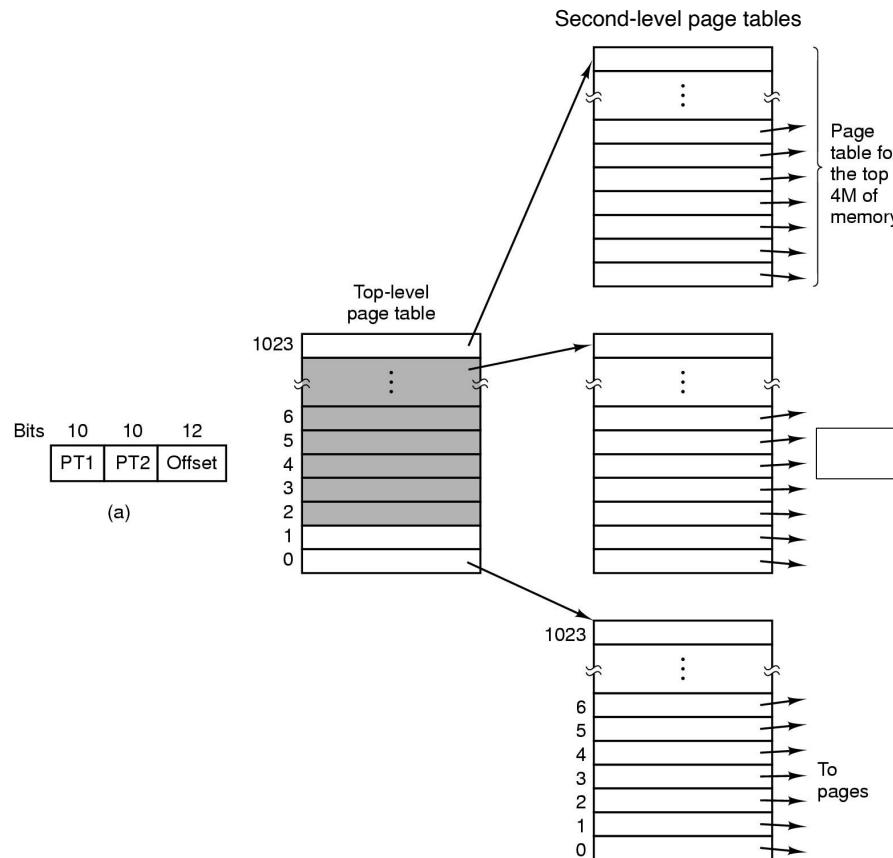
Size of address space (in bytes) as a function of address size (in bits)

| Number of bits in address | Maximum address space size (bytes) |
|---------------------------|---------------------------------------|
| 0 | $2^0 = 1$ byte |
| 1 | $2^1 = 2$ bytes |
| 2 | $2^2 = 4$ bytes |
| 10 | $2^{10} = 1024 = 1\text{KiB}$ |
| 12 | $2^{12} = 4\text{KiB}$ |
| 16 | $2^{16} = 64 \text{ KiB}$ |
| 32 | $2^{32} = 4\text{GiB}$ (Gibibytes) |
| 64 | $2^{64} = 16 \text{ EiB}$ (Exbibytes) |

Translating Virtual address (VA) to Physical address (PA) - Recap



Virtual Address Translation For Large Address Space



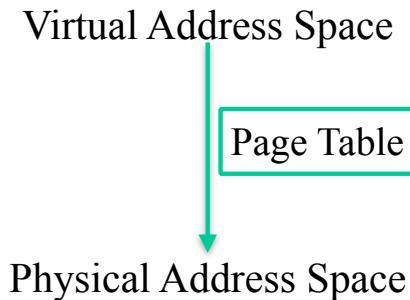
- 32 bit address with 2 page table fields
- Two-level page tables
- PT too Big for MMU
 - Keep it in main memory
- But how does MMU know where to find PT?
 - Registers (CR2 on Intel)

Memory Virtualization for VMs

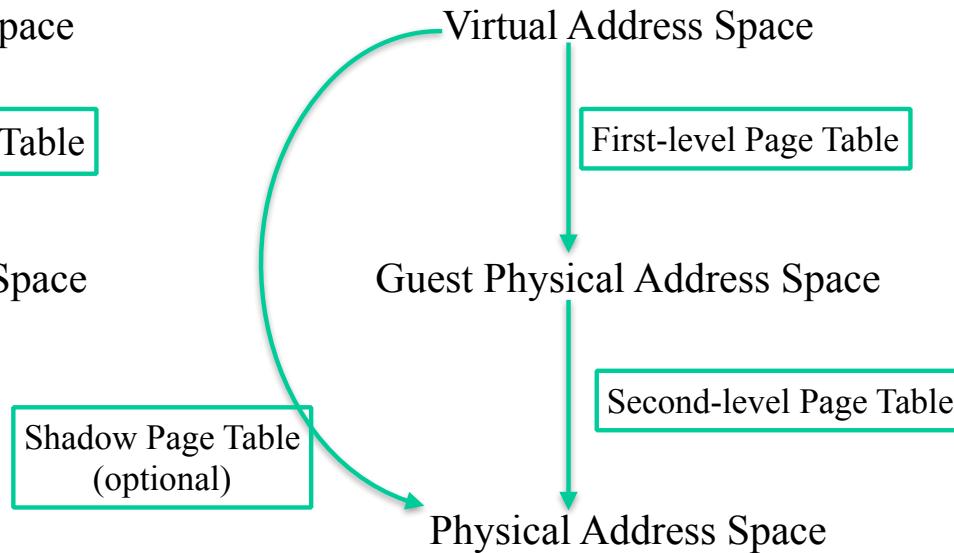
- Not the same as virtual memory for processes, but very similar
 - Just another layer of indirection
- Address Translation process
 - Level 1: Guest page table
 - Virtual address → Guest-physical address
 - Level 2: Extended Page table
 - Guest-physical address → host-physical address
- Sum of memory allocated to all guest VMs could be greater than physical memory of the system

Memory Virtualization for VMs

Traditional virtual memory



Virtual memory for VMs



- Guest OS in each VM sees a “guest”-physical address (GPA) space instead of the physical addresses
- Often hardware supports two-level page tables
 - EPT in Intel VT-x and NPT in AMD-v
- When hardware doesn’t, then Hypervisor emulates two-level page tables using “shadow page tables”.

I/O Virtualization for VMs

- Hypervisor provides a virtual version of each physical device
- I/O activity directed at the virtual device is trapped by Hypervisor and converted to equivalent request for the physical device.
- Device emulation
 - Hypervisor traps and emulates each I/O instruction from Guest in Hypervisor.
 - Very slow.
 - Difficult to emulate the effect of combinations of I/O instructions
- Para-virtual devices
 - Special device drivers inserted in guest OS to talk to Hypervisor.
 - Most common.
- Direct device access
 - Allow the VM to directly access physical device.
 - Fastest option but not scalable.
 - Requires IOMMU and VT-d support from hardware.