دانشگاه بجنورد

CE177

Advanced Operating Systems

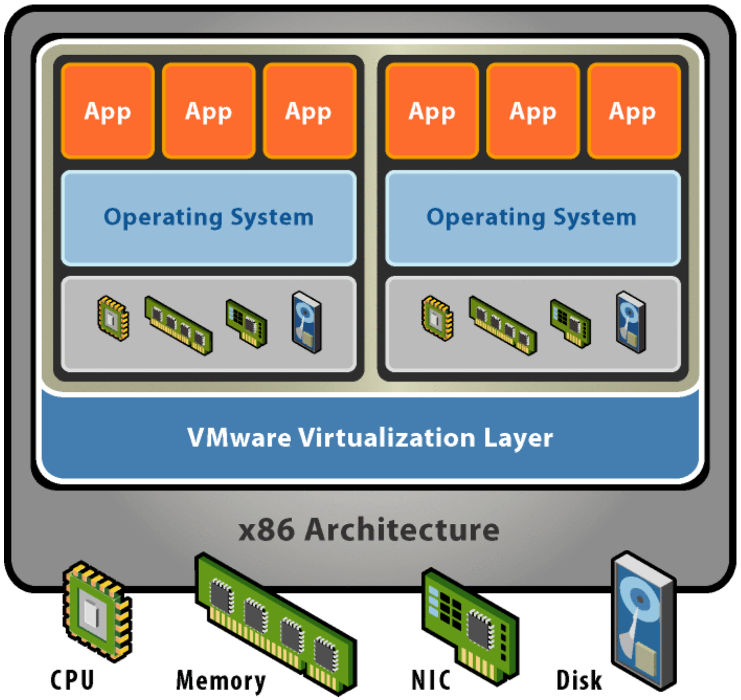
Hamid Fadishei, Assistant Professor
Computer Engineerign Departemnt,
University of Bojnord

Fall 2018

Virtualization: Intro and Concepts

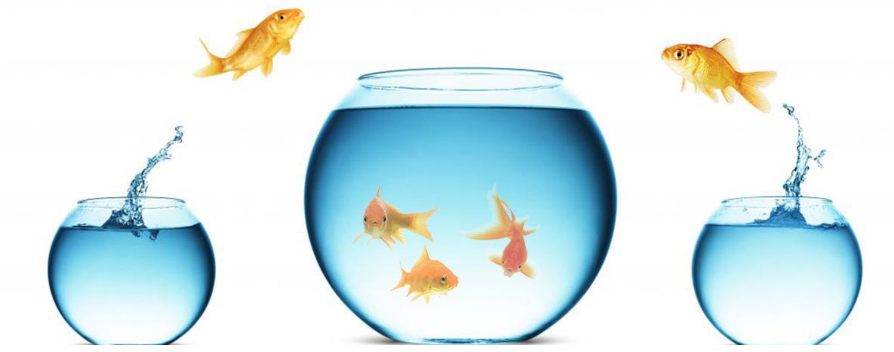
Hardware Virtualization

- Enables a single PC or server to simultaneously run multiple OSs
- Virtual vs Physical Machine (PM/VM)
- The PM hosts a number of VMs
- Each VM is has the characteristics of a particular OS (or in some cases a particular hardware)
- Guests (VMs) share the host resources
  - Managed by who? VMM!



Some Virtualization Terms

- Physical Machine vs Virtual Machine (PM/VM)
- Host OS vs Guest OS
- Isolation
- Consolidation
- Consolidation Ratio
- Virtual Machine Monitor (VMM) aka Hypervisor



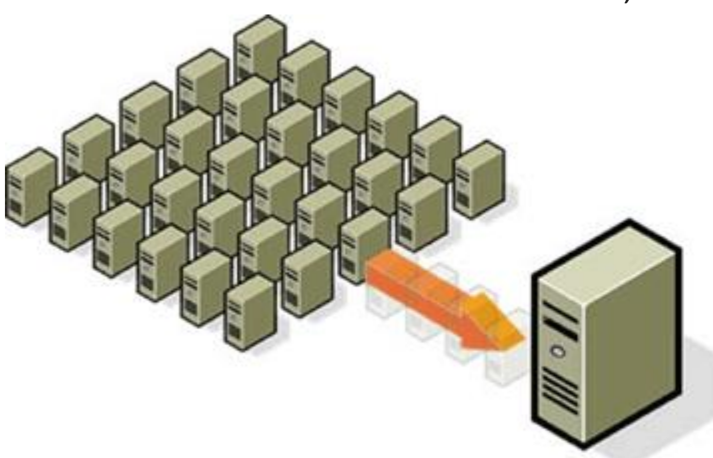
Some Virtualization History

- Not a new technology
  - The idea goes back to 1960's
  - A Survey of Virtual Machine Research (1974). By Robert P. Goldberg.
- Popular research idea in 1960s and 1970s
  - Entire conferences on virtual machine monitors
  - Allowed multiple users to share a batch oriented system
- Interest died out in the 1980s and 1990s
  - Hardware got more cheaper
  - Operating systems got more powerful (e.g. multi-user)
- Became popular again in 1990's
- Became hot and important since 2000

Why Virtualize?

- Servers
  - Consolidation → Utilization↑ Efficiency↑
  - Isolation → Security↑
  - Quickly create, snapshot, and migrate VMs-> Management Ease↑
  - Live Migration → Availability↑
  - On demand creation → Provisioning, Load Balancing
  - Versatitliy → Run different types and versions of software, Deliver software as VMs

- Desktop Environments
  - Isolation → Sandbox and Test Environments
  - Versatitliy → Run different types and versions of OS and software, Deliver software as VMs



Why Virtualize?

The answer in terms of the good properties of virtualization

- Isolation
  - Fault isolation
  - Performance isolation
  - Software isolation
- Encapsulation
  - Cleanly capture all VM state
  - Enables VM snapshots, clones
- Portability (Independent from physical hardware)
  - Enables migration of live, running VMs
  - Clone VMs easily, make copies
- Interposition (Transformations on instructions, memory, I/O)
  - Enables transparent resource overcommitment
  - Encryption
  - Compression
  - Replication

Adapted from: Yurvaj Agarwal, CMU Course 15-440/640 (Distributed Systems), Fall 2016

"Tell that intern that you can't migrate physical machines."

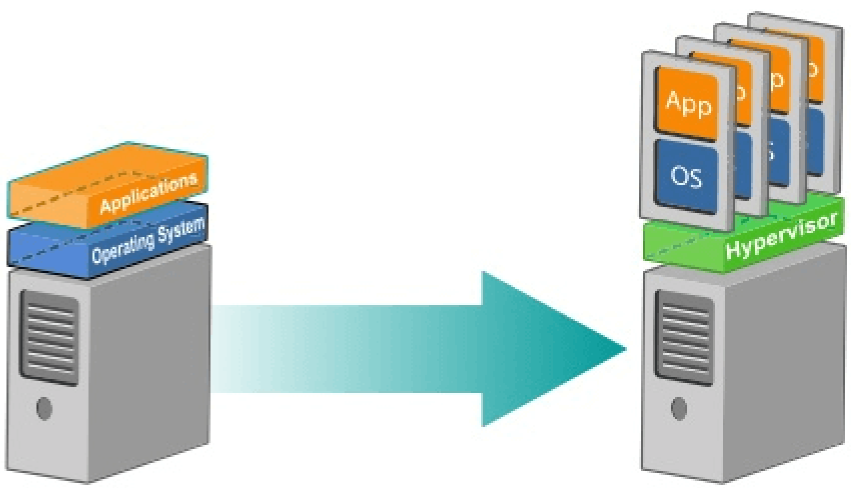
Virtualization Types

We have lots of VSs ahead

- Process virtualization vs system virtualization
- Hosted virtualization vs native virtualization
- Full virtualization vs paravirtualization
- Hardware-assisted vs software-assisted virtualization

Hypervisor

- Remember: OS abstracts hardware resources from user applications
- Simliarly, Hypervisor abstracts hardware resources from VMs
- A VM mimics characteristics of a physical machine
  - Has RAM, CPUs, Disk, etc
  - Once created, it can be powered on (like real machines)
- Hypervisor facilitates translation and I/O from VM to PM

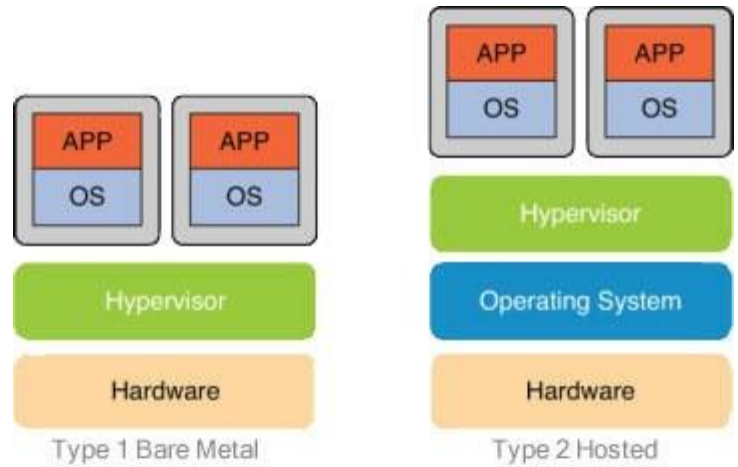


Hypervisor Functions

- Execution management of VMs: Scheduling, Memory management, Context switching etc
- Device emulation and access control
- Execution of privileged operations on hehalf of guests
- VM lifecycle management

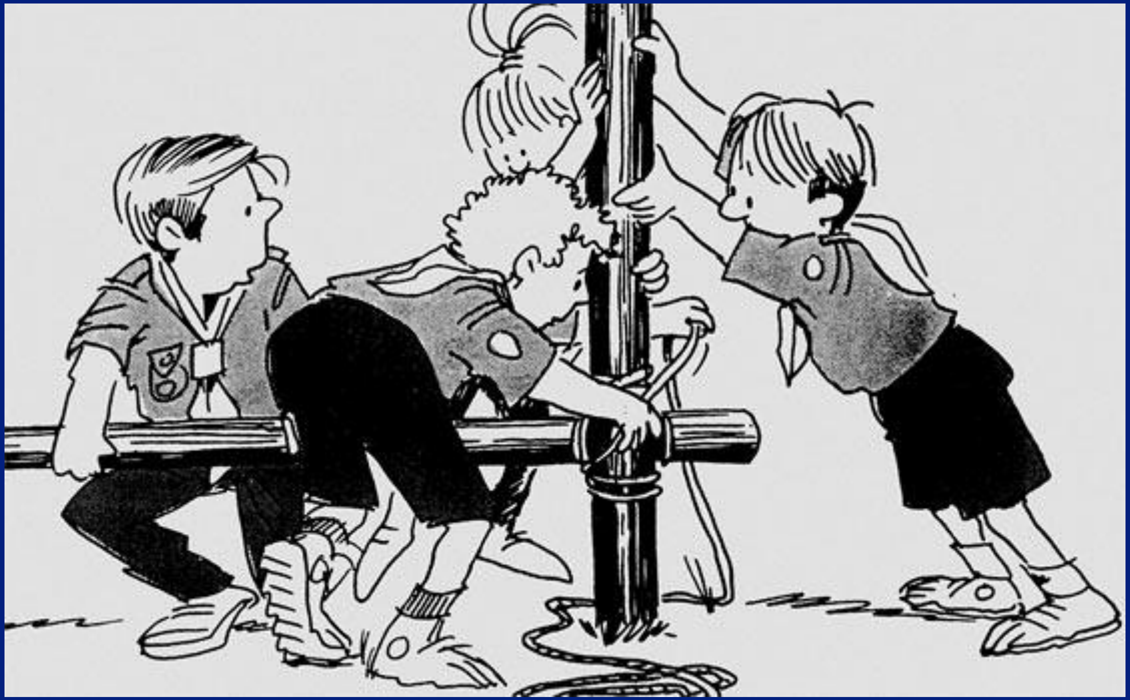
Hypervisor Types

- Type 1 aka Bare Metal aka Native
  - Loaded directly on PM like an OS
  - Examples: VMware ESXi, Microsoft Hyper-V
  - Pros
    - *Better performance (No competition with an OS)*
    - *More secure (Better isolation)*



- Type 2 aka Hosted
  - Loads on top of OS
  - Relies on OS functions
  - Examples: VMware Workstation and Oracle VM Virtual Box
  - Pros:
    - *The host can perform tasks other than virtualization*

Hands on
Oracle Virtualbox (Hosted VMM)

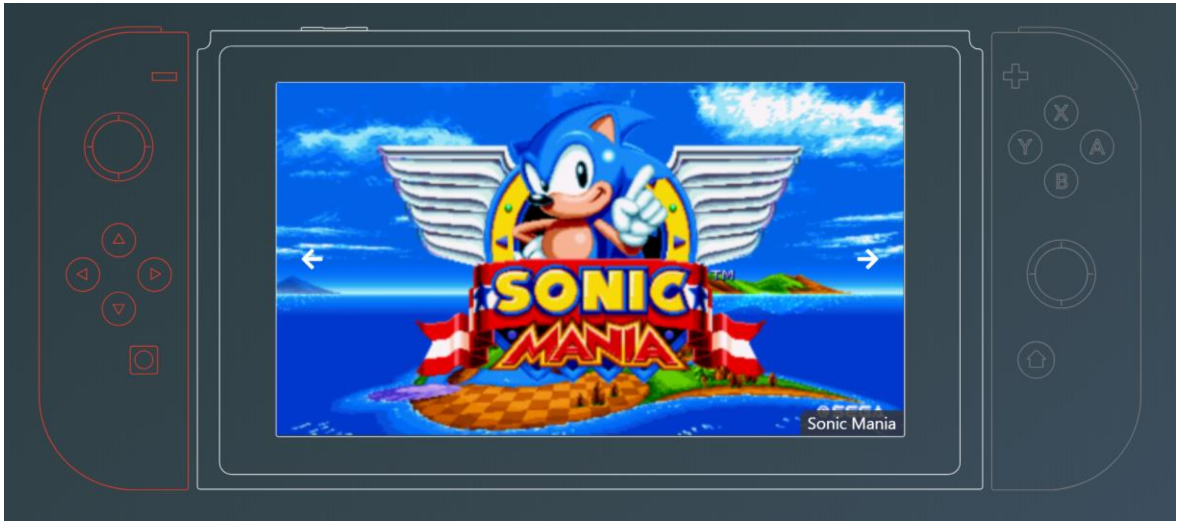


Virualization: hardware-assisted vs software assisted

- Hardware-assisted
  - The virtualization funcationality is implemented inside hardware (CPU)
  - Example: Intel VT-x
  - Pro: Performance (Lower overhead)
- Software-assisted
  - The virtualization funcationality is implemented inside softare (Hypervisor)
  - Example: QEMU
  - Pro: Lets virtualizing different architectures. For example, ARM VM on x86 PM

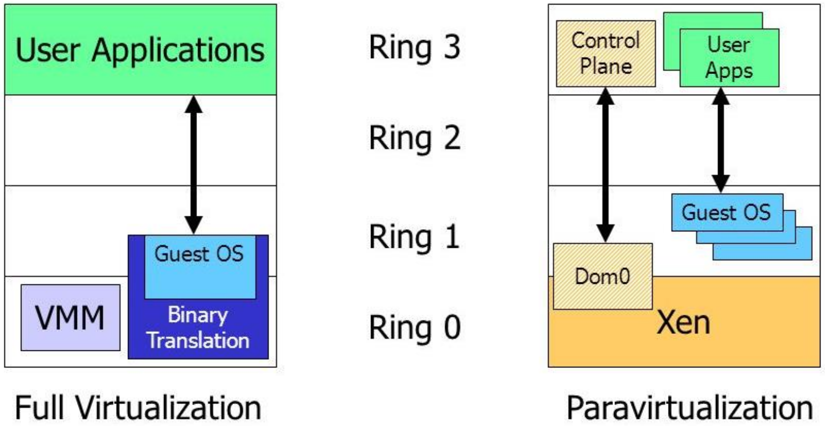
Emulation aka Hosted Interpretation aka Binary Translation

- VMM is a regular application on top of the OS
  - It reads guest binary instruction-by-instruction
  - Each instruciton is then translated to a number of host's native instructions
  - Significant overhead
  - It is a hosted approach. It is a software-assisted approach
  - Example: Yuzu (Nintendo emulator)



Full Virtualization vs Paravirtualization

- Full virtualization
  - No need to modify the guest OS
  - VM looks exactly like a physical machine
  - Needs binary translation at some points
- Paravirtualization
  - Need to modify some parts of the guest OS to be virtualizable
  - Better performance at the cost of less transparency

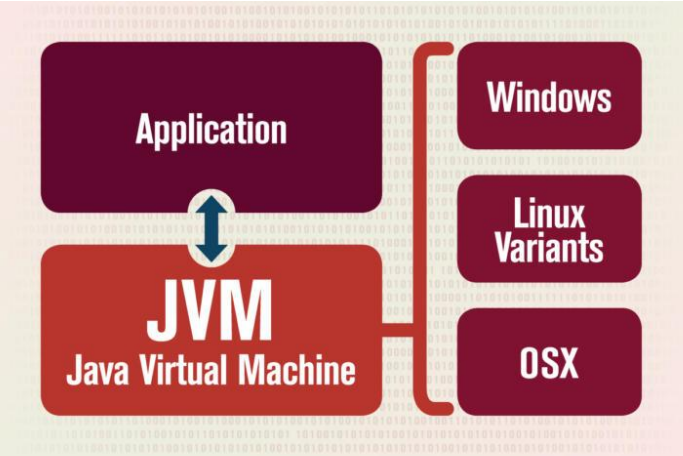


Process Virtualization vs System Virtualization

- Process virtualization
  - Language-level. Example: Java
  - Process-level aka Continers. A very hot topic today. Example: Docker, LXC
  - Cross-ISA emulation. You already know from previous slides
- System (Hardware) virtualization
  - You already know from previous slides

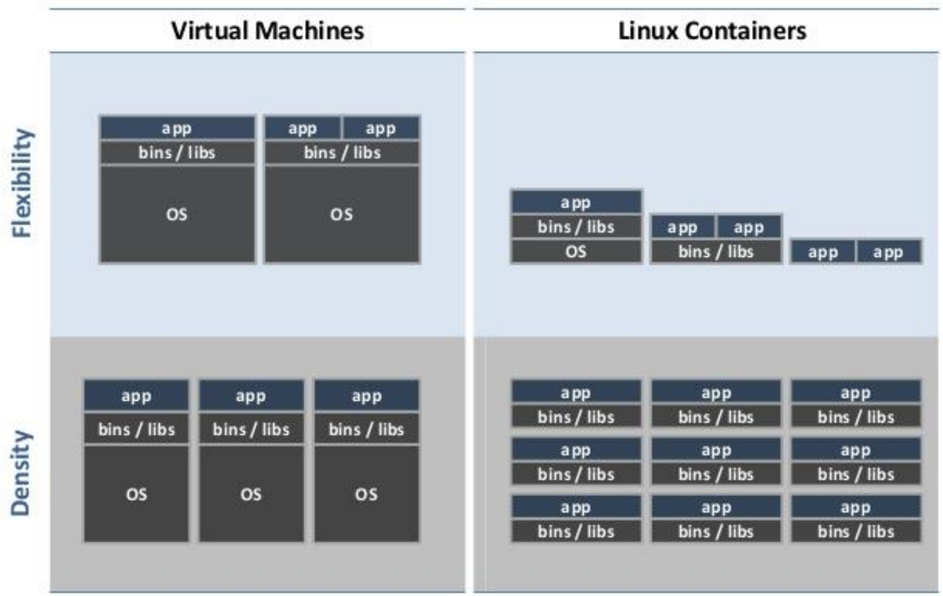
Process Virtualization: Language-Level

- Not really virtualization. But using the concept and providing the same features
- Programming language is designed to run within custom-built virtualized environment
- Virtualization is defined as providing APIs that define a set of features made available to a language and programs written in that language to provide an improved execution environment
  - JVM compiled to run on many systems
  - Programs written in Java run in the JVM no matter the underlying system



Process Virtualization: OS-Level

- Known as application containers
  - Only one kernel is running
  - Processes inside each container think they are alone!
  - Each container has its own filesystem, network addresses and ports
  - System resources like CPU slices and memory are shared among them
- Examples: Docker, LXD
- Probably the future of virtualization
- Con: Weak isolation
- Pro: Extremely condense (Thousands of VMs on a single PM now possible)

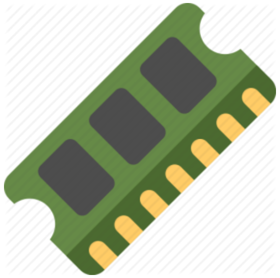


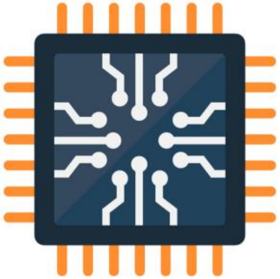
Hands On
LXC (Container)

Implementing Virtualization

Implementing System Virtualization

- Different subsystems, different concerns
  - Processor virtualization
  - Memory virtualization
  - I/O virtualization





Processor Virtualization

Processor Virtualization

- VMM essential properties (Popek-Goldberg, 1974)
  - **Equivalent execution:** Programs running in the virtualized environment run identically to running natively.
  - **Performance:** A statistically dominant subset of the instructions must be executed directly on the CPU.
  - **Safety and isolation:** A VMM most completely control access to system resources.

Formal Requirements for Virtualizable Third Generation Architectures

Gerald J. Popek
University of California, Los Angeles
and
Robert P. Goldberg
Honeywell Information Systems and
Harvard University

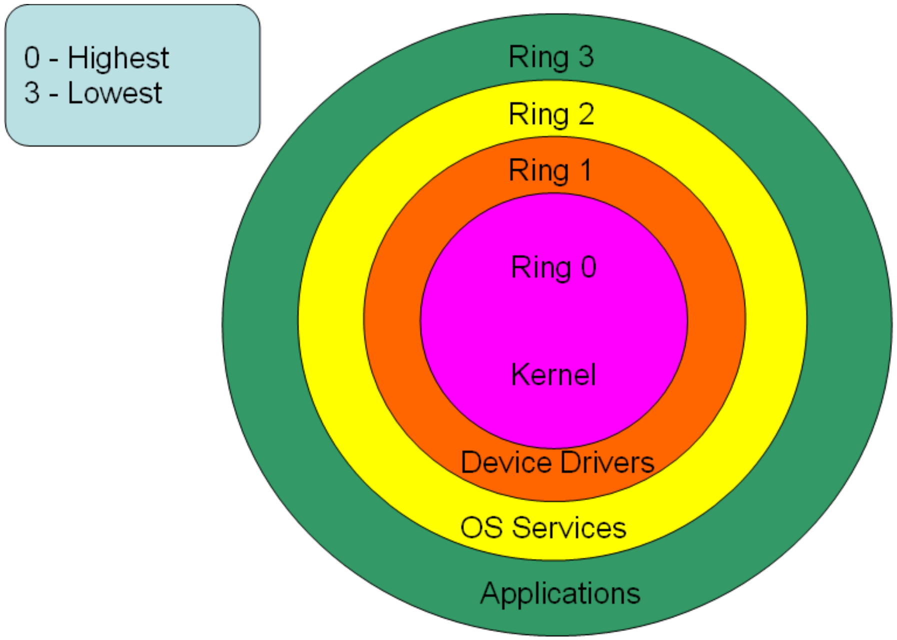
A virtual machine is taken to be an *efficient, isolated duplicate* of the real machine. We explain these notions through the idea of a *virtual machine monitor* (VMM). See Figure 1. As a piece of software a VMM has three essential characteristics. First, the VMM provides an environment for programs 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.

Processor Virtualization

- Virtualizable architecture
  - An architecture is classically/strictly virtualizable if all its sensitive instructions (those that violate safety and encapsulation) are a subset of the privileged instructions.

Processor Virtualization

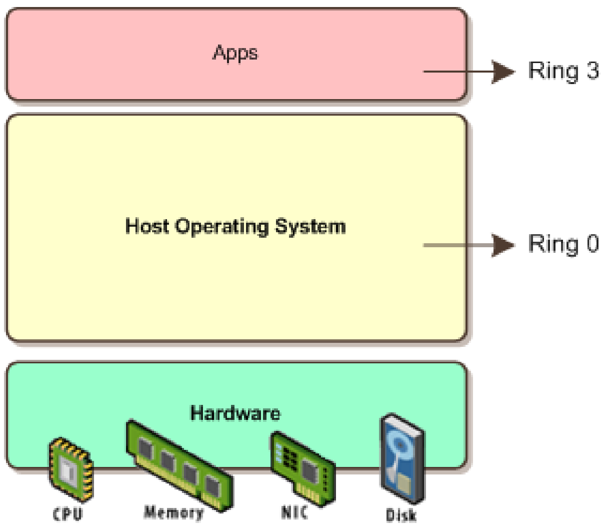
- x86 protection rings
  - Code in a more privileged ring can read and write memory in a lower privilege ring, but function calls between rings can only happen through hardware-enforced mechanisms (e.g., system calls, gates)
  - Only Ring 0 can execute privileged instructions; Rings 1, 2, and 3 will trap when executing privileged instructions



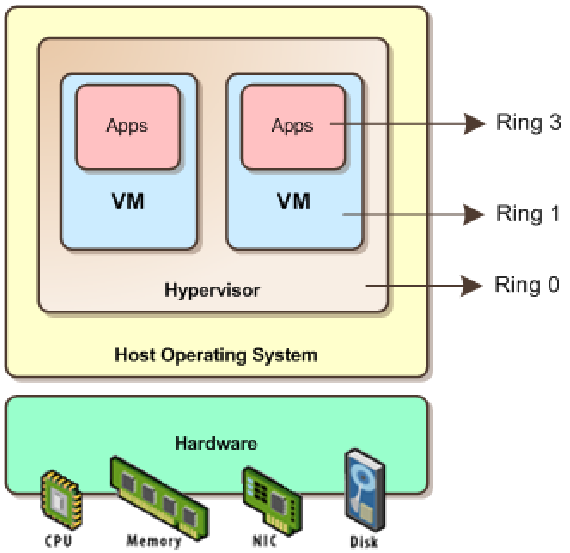
Processor Virtualization

- x86 protection rings

Without Virtualizaiton

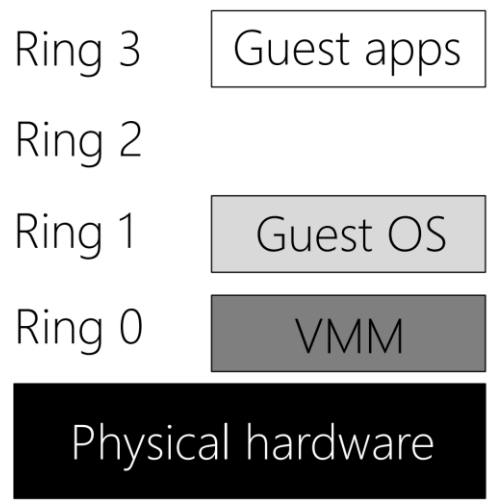


With Virtualizaiton



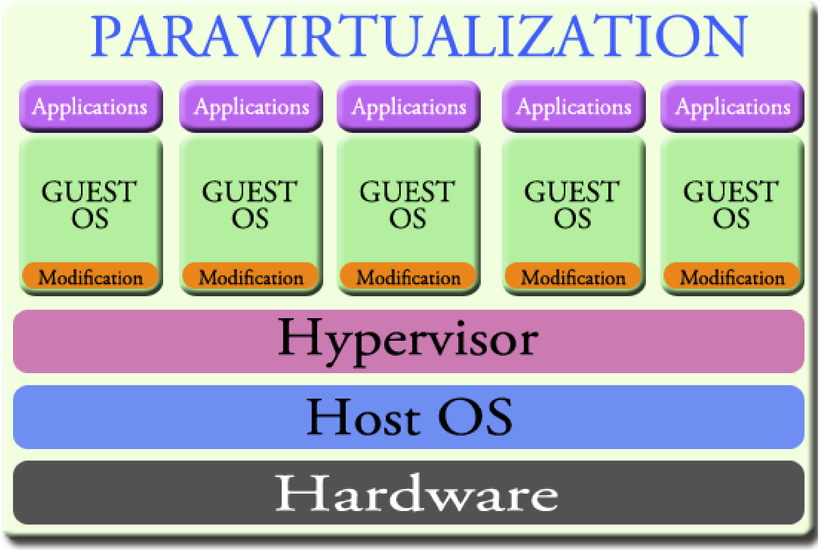
The Trap and Emulate Approach

- Guest apps can’t tamper with the guest OS due to ring protections
- Guest apps and guest OS can’t tamper with VMM due to ring protections
- When the guest OS executes a privileged instruction, it will trap into the VMM
- When a guest app generates a system call or exception, the app will trap into the VMM
- VMM’s trap handler uses a policy to decide what to do (e.g., emulate the instruction, kill the VM, etc.)



Handling Nonvirtualizabe Processors

- Binary translation (Full virtualizatin)
  - Sensitive instructions are replaced with “calls to hypervisor” or “unknown opcodes which cause trap to hypervisor”
    - *Hosted: all sensitive instructions are replaced*
    - *Bare metal: only nonvirtualizable instructions are replaced*
- Paravirtualization
  - Modifying the guest OS and porting it to the subset of virtualizable instructions

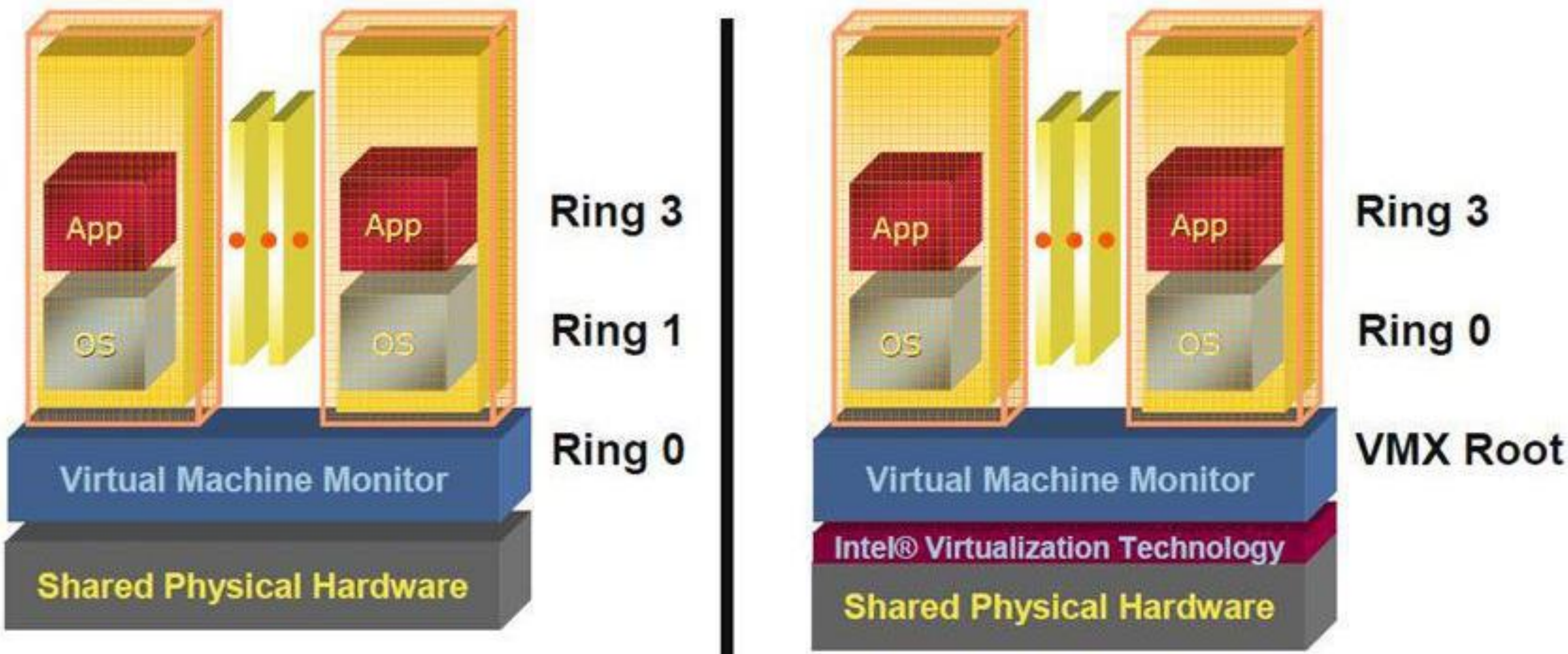


Hardware-accelerated virtualization

Intel VT-x

- Remember x86 protection rings?
  - Normal execution: OS->ring0, APP->ring3
  - Virtualized: Host->ring0, Guest->ring1, APP->ring3
  - Guest OS is unaware that it is running on ring 1 not ring0
  - Critical instructions are trapped and a transition from guest to VMM occures (called VMexit)
- New Inel processors introduce two new modes:
  - Root (or ring -1) where host OS runs
  - Non-root where protection rings apply
  - Guest OS goes back where it belongs: ring 0
  - Not all system calls cause VMexit (less overhead)
  - VMCS can be configured to make guest exit on certain conditions

Hardware-accelerated virtualization



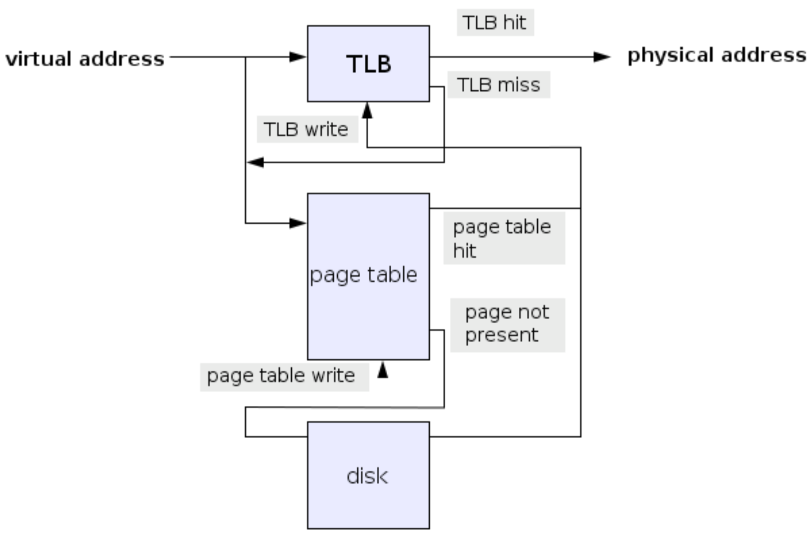
Pre VT-x | Post VT-x

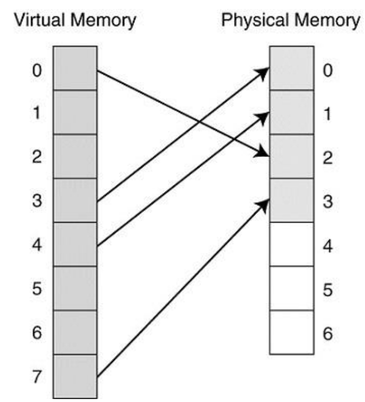
VMM ring de-privileging of guest OS	VMM executes in VMX root-mode
Guest OS aware its not at Ring 0	Guest OS de-privileging eliminated
	Guest OS runs directly on hardware

Memory Virtualization

Memory Virtualization

- Memory Virtualization ≠ Virtual Memory
- Virtual Memory (Basic OS Course)
  - Address Space
  - Page
  - Frame
  - Page Table
  - TLB
  - Page Fault
- Memory Virtualization
  - Memroy Management of virtual machines (virtual machines' virtual memory
  - Performed by VMM



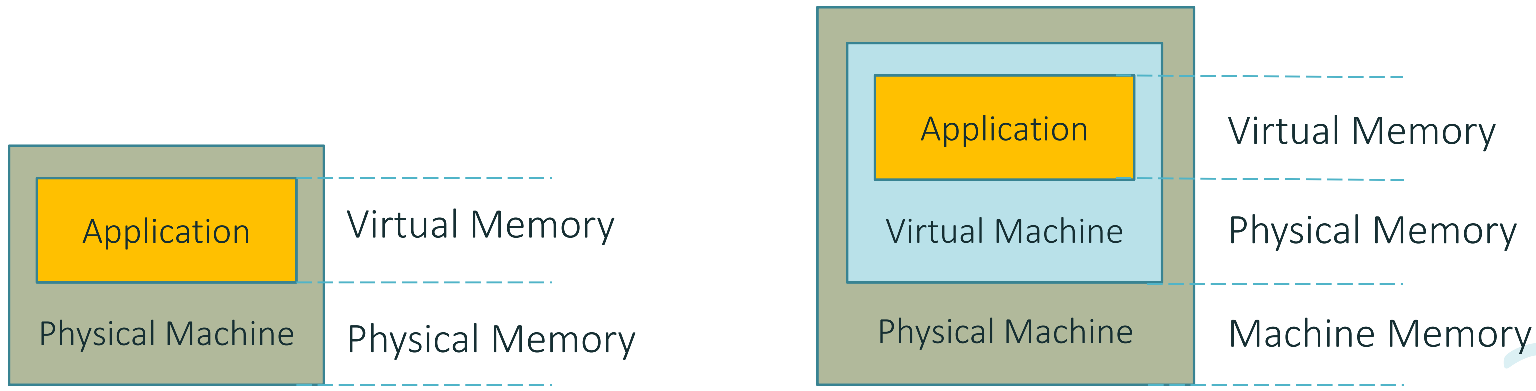


Page Table

Page	Frame #	Valid Bit
0	2	1
1	-	0
2	-	0
3	0	1
4	1	1
5	-	0
6	-	0
7	3	1

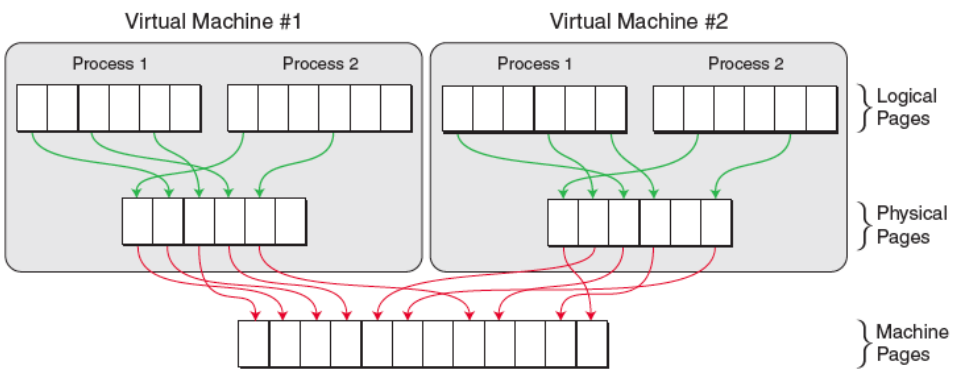
Memory Virtualization

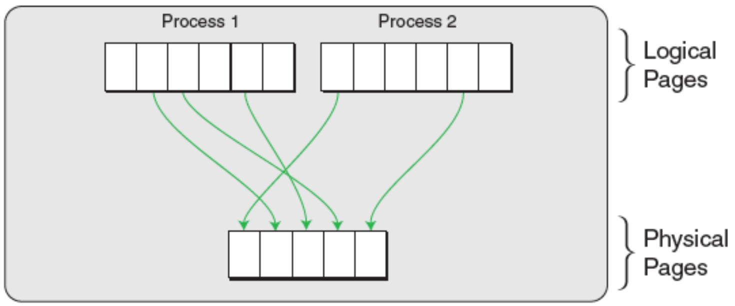
- Non-virtualized environments
  - Two levels of abstraction
- Virtualized environments
  - Three levels
    - *Machine memory: A server has 32GB of RAM*
    - *Physical memory: A VM on this server gets 4GB of RAM*
    - *Virtual memory: Apache web server on this VM consumes 2GB of VM RAM*



Memory Virtualization

- Memory virtualization
  - Mapping between logical and physical pages
    - *Performed by the guest OS (inside VMs)*
  - Mapping between physical pages and machine pages
    - *Needs to be performed by VMM*
    - *An extra layer of memory management*
    - *Means extra overhead*





Shadow Page Tables

- Guest OS maintains L=>P mapping
- VMM manages P=>M mapping
  - Page tables are loaded into the MMU on a context switch
- Shadow page tables
  - Map L=>M directly (one lookup instead of two)
- VMM needs to keep its L=>M tables consistent with changes made by OS to its L=>P tables
  - VMM maps OS page tables as read only
  - When OS writes to page tables, trap to VMM
  - VMM applies write to shadow table and OS table, returns
  - Also known as memory tracing

Hardware-assisted Memory Virtualization

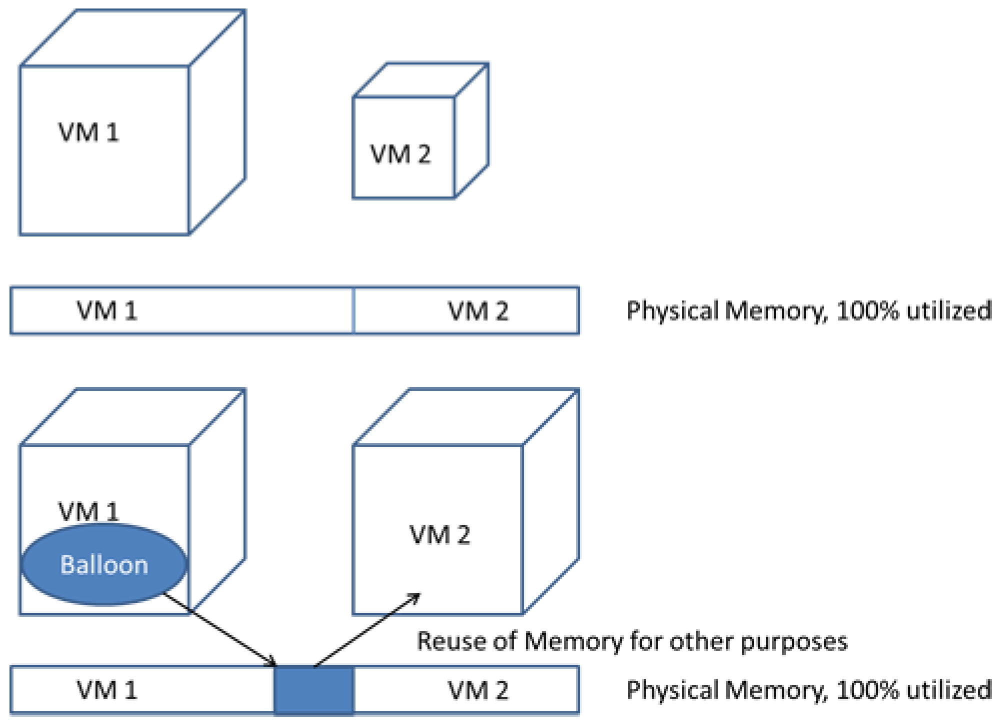
- Intel EPT (Extended Page Table)
  - Modern Intel processors implement memory virtualization (shadow page tables) in hardware
  - Helps with performance

Advanced Topics in Memory Virtualization

- Simple policy
  - Each VM is assigned with a fixed amount of physical memory when created
  - No dynamic adjustment
  - No swapping to disk
- More complex implementation
  - Increase or decrease the amount of VM's physical memory at runtime
  - Known as "ballooning"
- Even more complex
  - Detect and eliminate identical pages
  - Known as "deduplication"
- Both techniques allow "overcommitment"

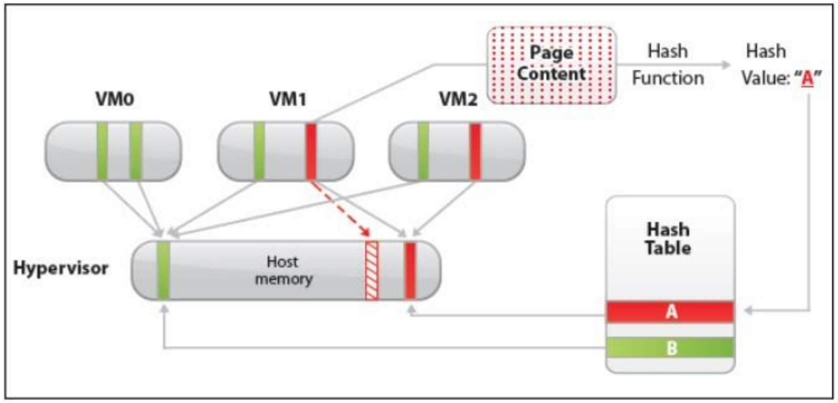
Memory Ballooning

- Normally, when the hypervisor allocates a memory range to a VM, this range is assumed completely used
- However, the VM may not use all the allocated space
- Ballooning lets hypervisor claim unused memory from some VMs and give them to others



Memory Deduplication

- Allows not repeating the pages with the same content
  - Effective in virtualized environments
  - Can impose security threats
    - *Why? Violates which of Popek-Goldberg properties?*
    - *Memory disclosure attack. Exploits access time variation between shared and normal pages*



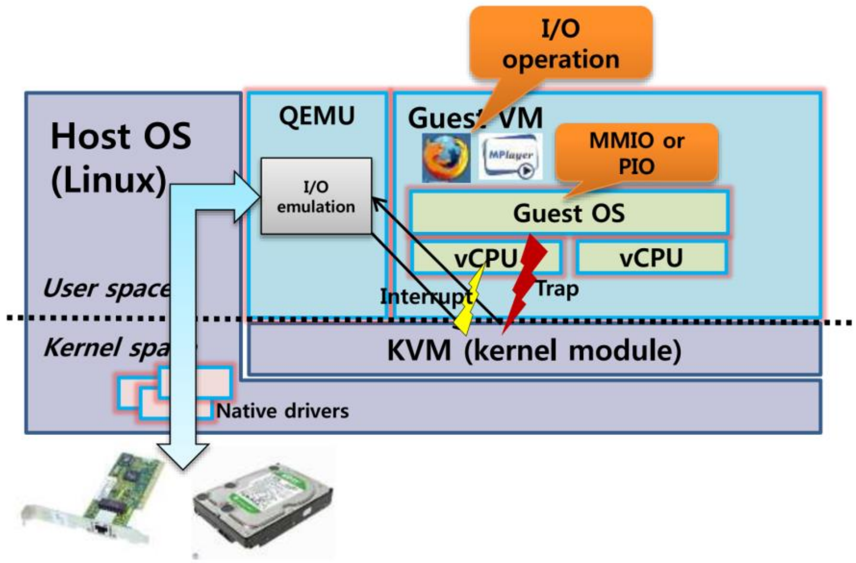
I/O Virtualization

I/O Virtualization

- A complex task
  - Various I/O devices
    - *Storage*
    - *Network*
    - *Other*
  - Different solutions
  - Many short paths for performance
- Two main methods for I/O virtualization
  - Rewrite device drivers inside VMM
    - *Con: High engineering cost*
    - *Pro: Low overhead*
  - Use existing drivers in host OS WITH virtual devices in guest OS
    - *Pro: Low engineering cost*
    - *Con: High overhead*
- I/O requests are all privileged and trapped
  - Architectures are I/O-virtualizable in general
  - Trap and emulate approach is possible

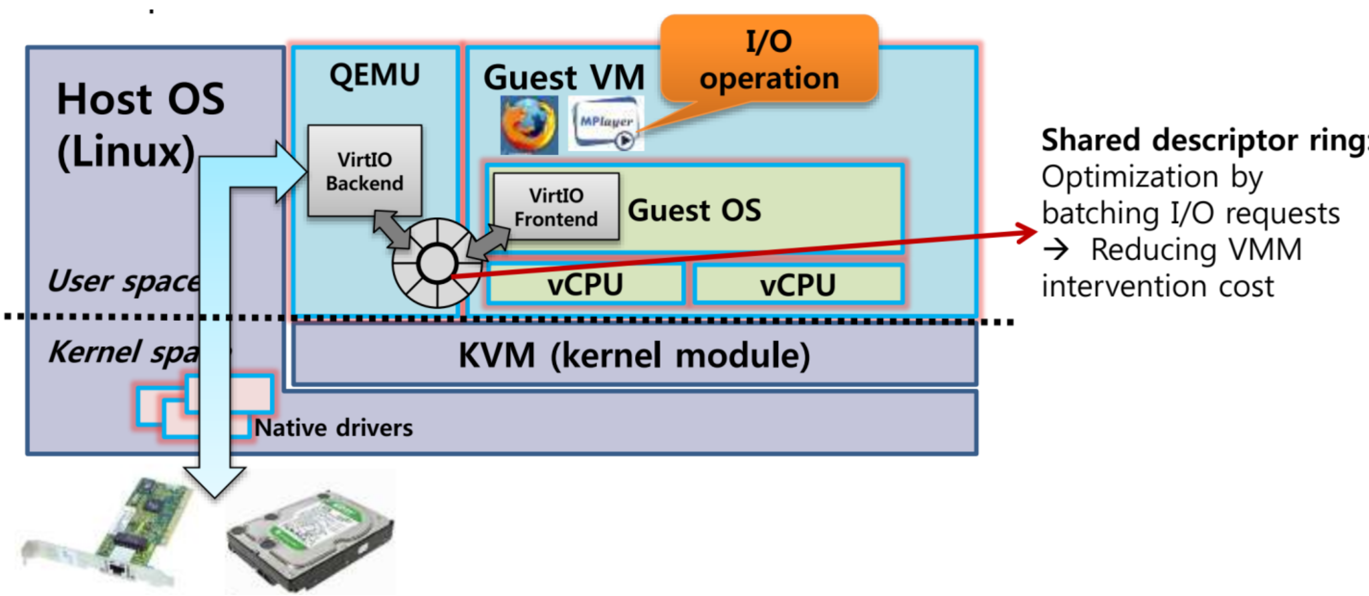
I/O Virtualization via Trap and Emulate

- Approach 1: Full virtualization
  - An emulation layer inside VMM
    - *Mimics a well-known device*
    - *Example: Intel Pro 1000 Network Adapter in Virtualbox*
  - An existing driver inside VM
- Approach 2: Paravirtualization
  - A backend driver inside VMM
  - A frontend driver inside VM (the reason to call this para)
  - Example: virtio for KVM



I/O Virtualization via Trap and Emulate

- Approach 1: Full virtualization
  - An emulation layer inside VMM
    - *Mimics a well-known device*
    - *Example: Intel Pro 1000 Network Adapter in Virtualbox*
  - An existing driver inside VM
- Approach 2: Paravirtualization
  - A backend driver inside VMM
  - A frontend driver inside VM (the reason to call this para)
  - Example: virtio for KVM



I/O Virtualization

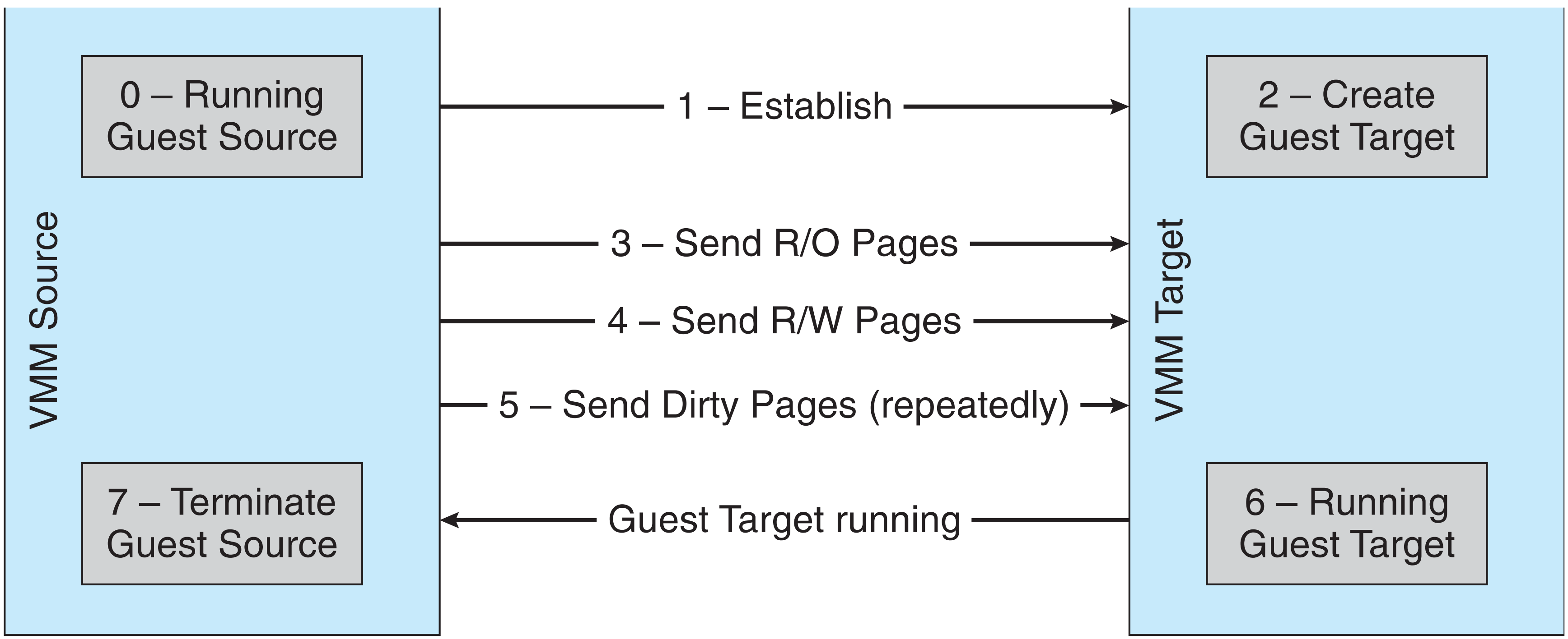
- Device-specific consideration
  - Network devices
    - *Bridge existing NIC to VM*
    - *NAT specific addresses and ports to VM*
    - *Assign a whole NIC to a VM*
  - Storage
    - *VM filesystem is stored normally on the host (Example: LXC)*
    - *VM disk is a single-file image somewhere on the host (Example: Virtualbox)*
    - *VM disk is a whole partition or even a whole disk on the host (Example: Virtualbox)*

Hardware-assisted I/O Virtualization

- IOMMU
  - Per-VM address space for DMA operations
  - Allows Direct assignment of I/O devices to VMs
  - Example: Assign a PCIe network adapter to a VM
  - Intel VT-d
- Multi-queue NIC
  - Network adapters with multiple queues
  - Assign each queue to a VM
  - Intel VT-c

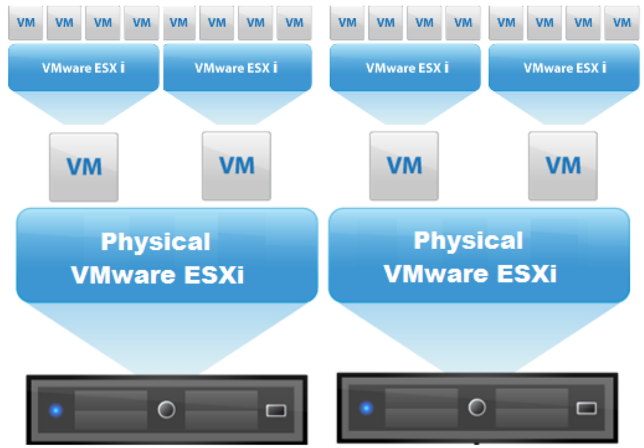
Advanced Topics

Live Migration

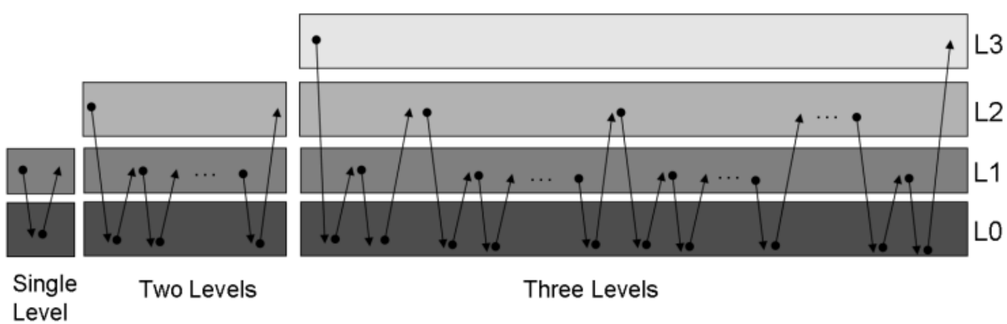


Nested Virtualization

- Having VMs inside VMs inside VMs…

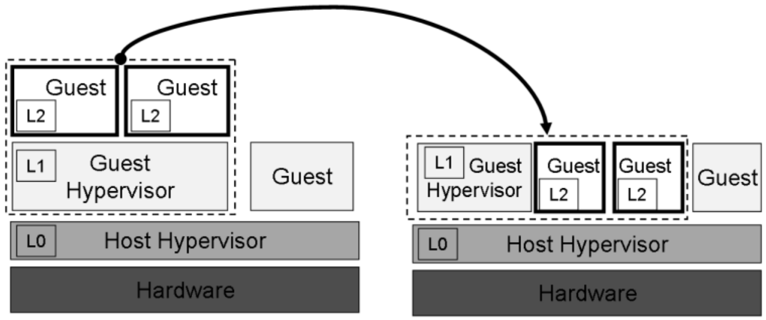


- VMexit and VMEntry can occur between any layer



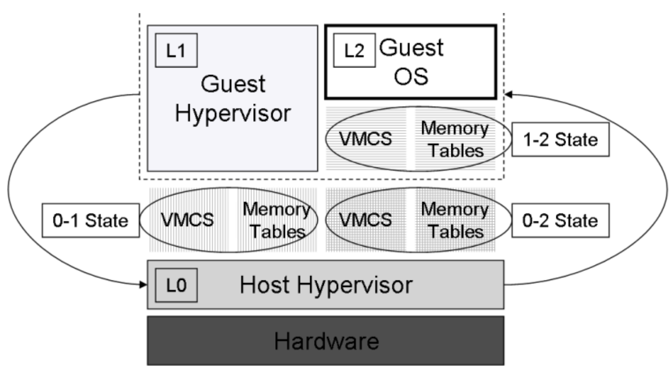
Nested Virtualization

- Implementation
  - Reading: Ben-Yehuda, [The turtles project](), 2010
  - Approach 1: multiplexing in a single-level environment



Multiple levels support   Multiplexing a single level

  - Approach 2: extending hardware support (VT-x)
    - *Remember Page Table Shadowing?*
    - *Remember VMCS?*
    - *What we need for hardware-assisted nested VMs: VMCS Shadowing*



Some Research Topics

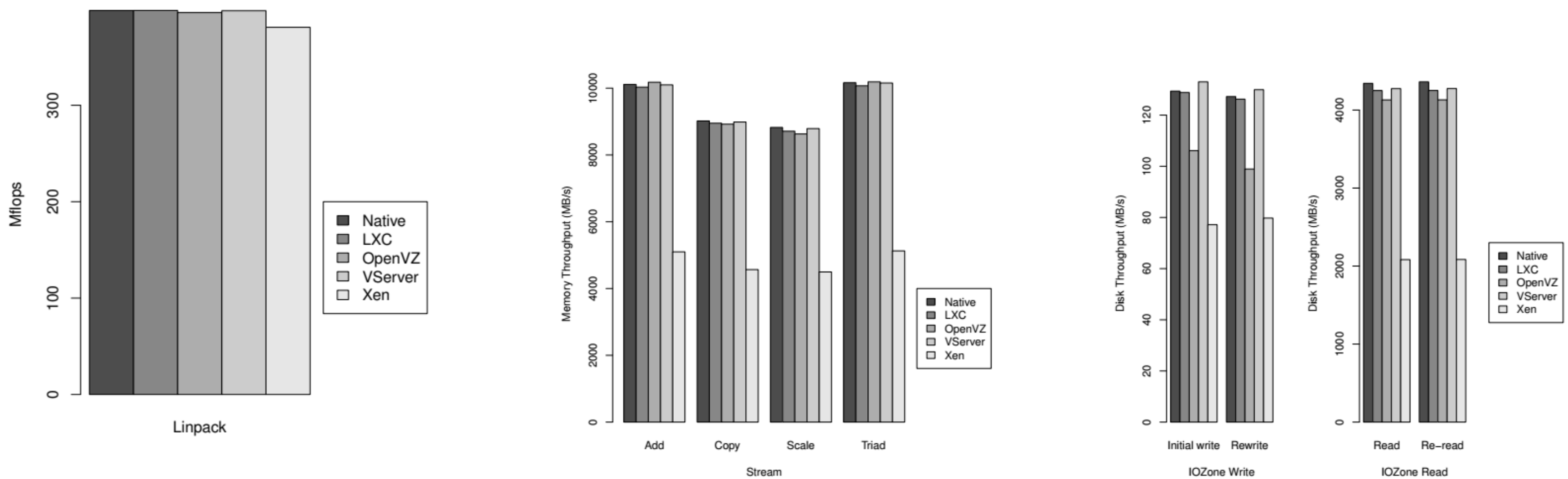
Some AOS research topics

- Performance overhead comparison between approaches
- Energy efficiency comparison between approaches
- Inter-VM communications
- Virtualization for IoT
- Analyze the security implications of new approaches

Tasks and Exercises

Task 1

- All students, due next session
- Find and post me a graph image comparing some aspect of some virtualization methods quantitatively
- You’ll explain your graph to us at next class
    - Example: Xavier, 2013, Performance Evaluation of Container-based Virtualization for High Performance Computing Environments



(a) Computing performance using Linpack for matrices of order 3000.

(b) Memory throughput using STREAM.

(c) Disk throughput using IOZone.

Figure 2. Performance evaluation for different micro-benchmarks

Task 2 (Term Project)

- Single student
  - Study the assigned paper
  - Present in class (due 2 weeks)
  - Find and study 5 related papers
  - Deliver a paper-formatted survey (due final exam)

- 1st student:
  - Morabito, Roberto, et al. "**Consolidate IoT edge computing with lightweight virtualization**." IEEE Network 32.1 (2018): 102-111.

- 2nd student:
  - Li, Zheng, et al. "**Performance overhead comparison between hypervisor and container based virtualization**." Advanced Information Networking and Applications (AINA), 2017 IEEE 31st International Conference on. IEEE, 2017.