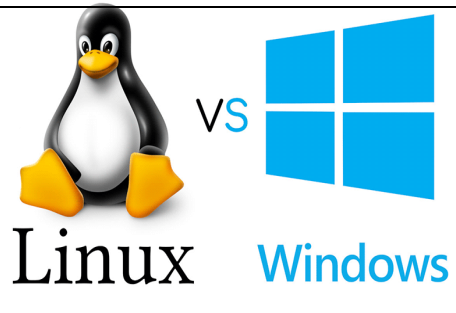
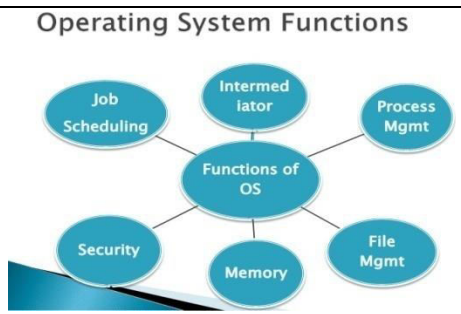
CIT210 OP. SYS. MANAG.

CSUN
COLLEGE OF ENGINEERING AND COMPUTER SCIENCE

Week8Day2: Basic Concepts of Machine Virtualization, Building Blocks, Virtual Machine Architecture, Emulation, Application Containment, Memory Management, CPU Scheduling, I/O Control





Objectives

- Basic Concept of Machine Virtualization
- Explore the history and benefits of virtual machines
- Discuss the various virtual machine technologies
- Describe the methods used to implement virtualization
- Show the most common hardware features that support virtualization and explain how they are used by operating-system modules
- Discuss current virtualization research areas

History

- First appeared in IBM mainframes in 1972
- Allowed multiple users to share a batch-oriented system
- Formal definition of virtualization helped move it beyond IBM
  1. A VMM provides an environment for programs that is essentially identical to the original machine
  2. Programs running within that environment show only minor performance decreases
  3. The VMM is in complete control of system resources
- In late 1990s Intel CPUs fast enough for researchers to try virtualizing on general purpose PCs
  - Xen and VMware created technologies, still used today
  - Virtualization has expanded to many OSes, CPUs, VMMs

Early days of VM

Technology was initially popularized in 1960's by IBM.

IBM 370 /VM

1990 VMware released its virtualization product: slow, based on x86 platform (no multi-core), expensive

Now, we have

- VMware suite;
- MS virtual PC, Hypervisor;
- QEMU (Quick Emulator): open source
- XEN
- OpenVZ
- KVM
- VirtualBox

Overview

- **Fundamental idea – abstract hardware of a single computer into several different execution environments**
  - **Similar to layered approach**
  - **But layer creates virtual system (virtual machine, or VM) on which operation systems or applications can run**
- **Several components**
  - **Host – underlying hardware system**
  - **Virtual machine manager (VMM) or hypervisor – creates and runs virtual machines by providing interface that is *identical* to the host**
    - (Except in the case of paravirtualization)
  - **Guest – process provided with virtual copy of the host**
    - Usually an operating system
- **Single physical machine can run multiple operating systems concurrently, each in its own virtual machine**

Basic Concept of Machine Virtualization

- **The conventional computer has a simple architecture**
  - **The operating system (OS) manages all hardware resources at the privileged system space**
    - All applications run at the user space under the control of the OS
- **After virtualization**
  - **Different user applications are managed by their own operating systems (guest OS)**
    - Can run on the same hardware
    - Independent of the host OS
  - **Often done by adding additional software**
    - Known as a virtualization layer

Virtualization

- **Virtualization -- the abstraction of computer resources.**

- **Virtualization hides the physical characteristics of computing resources from their users, be they applications, or end users.**

- **This includes making a single physical resource (such as a server, an operating system, an application, or storage device) appear to function as multiple virtual resources; it can also include making multiple physical resources (such as storage devices or servers) appear as a single virtual resource.**

Virtualization

Virtualization is creation of an alternative to actual version of something:

- **virtual memory (more memory than physically available),**
- **virtual time (buffering provides virtual/effective download time that less than the actual time),**
- **Virtual hardware, desktop, disk, appliances, scenes,**
- **Virtual worlds**

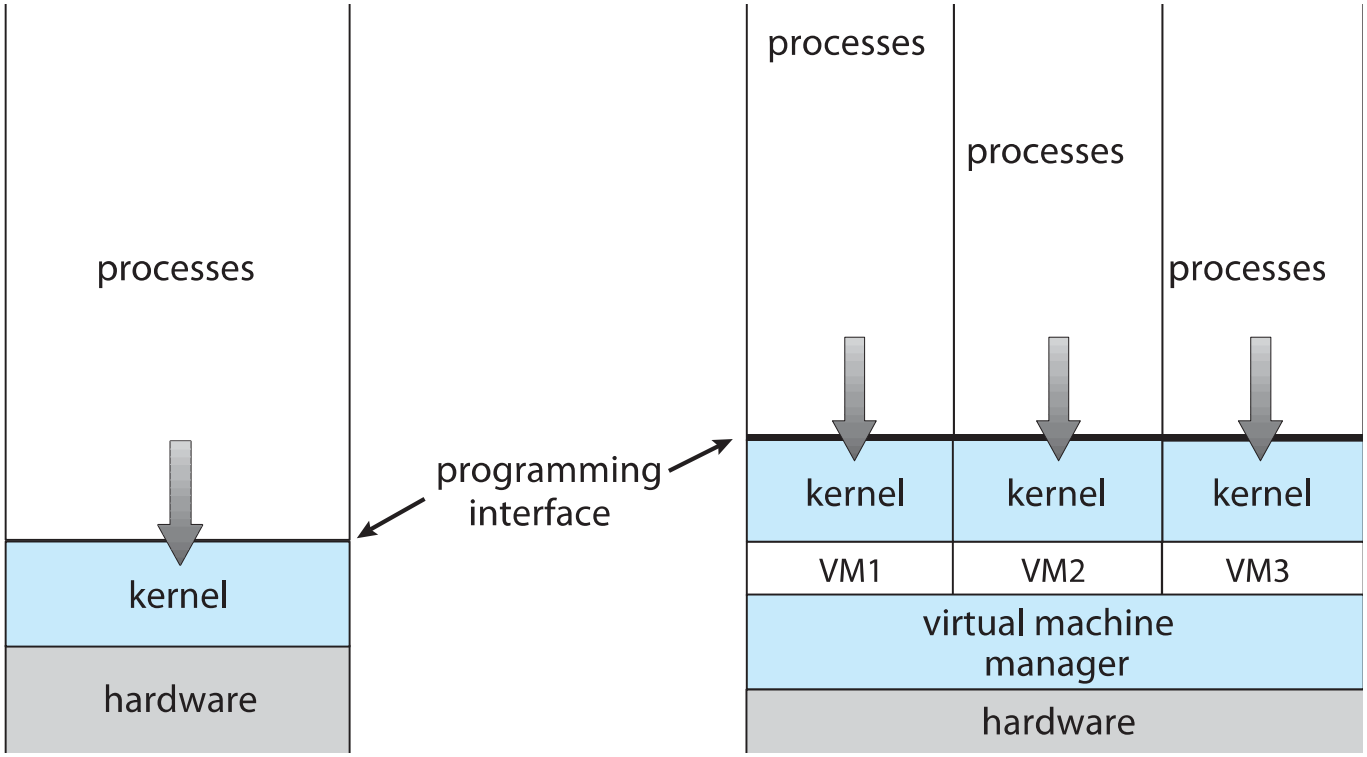
In our context it is realizing one or more complete computer systems as guests on the base machine/operating system.

This offers an excellent conduit for delivering the vastly underutilized power of the multi-core and other resources such as storage and devices.

Virtual Machines

- Virtualization technology enables a single PC or server to simultaneously run multiple operating systems or multiple sessions of a single OS
- A machine with virtualization software can host numerous applications, including those that run on different operating systems, on a single platform
- The host operating system can support a number of virtual machines, each of which has the characteristics of a particular OS
- The solution that enables virtualization is a *virtual machine monitor (VMM), or hypervisor*

System Models



Non-virtual machine

Virtual machine

Approaches to Virtualization

A Virtual Machine is a software construct that mimics the characteristics of a physical server

- it is configured with some number of processors, some amount of RAM, storage resources, and connectivity through the network ports
- once the VM is created it can be powered on like a physical server, loaded with an operating system and software solutions, and utilized in the manner of a physical server
- unlike a physical server, this virtual server only sees the resources it has been configured with, not all of the resources of the physical host itself
- the hypervisor facilitates the translation and I/O from the virtual machine to the physical server devices and back again to the correct virtual machine

Virtual Machine Files

Virtual machines are made up of files:

- configuration file describes the attributes of the virtual machine
- it contains the server definition, how many virtual processors (vCPUs) are allocated to this virtual machine, how much RAM is allocated, which I/O devices the VM has access to, how many network interface cards (NICs) are in the virtual server, and more
- it also describes the storage that the VM can access
- when a virtual machine is powered on, or instantiated, additional files are created for logging, for memory paging, and other functions
- since VMs are already files, copying them produces not only a backup of the data but also a copy of the entire server, including the operating system, applications, and the hardware configuration itself

Benefits and Features

- Host system protected from VMs, VMs protected from each other
  - I.e. A virus less likely to spread
  - Sharing is provided though via shared file system volume, network communication
- Freeze, suspend, running VM
  - Then can move or copy somewhere else and resume
  - Snapshot of a given state, able to restore back to that state
    - Some VMMs allow multiple snapshots per VM
  - Clone by creating copy and running both original and copy
- Great for OS research, better system development efficiency
- Run multiple, different OSes on a single machine
  - Consolidation, app dev, …

Benefits and Features (cont.)

- Templating – create an OS + application VM, provide it to customers, use it to create multiple instances of that combination
- Live migration – move a running VM from one host to another!
  - No interruption of user access
- All those features taken together -> cloud computing
  - Using APIs, programs tell cloud infrastructure (servers, networking, storage) to create new guests, VMs, virtual desktops

Building Blocks

- **Generally difficult to provide an *exact* duplicate of underlying machine**
  - **Especially if only dual-mode operation available on CPU**
  - **But getting easier over time as CPU features and support for VMM improves**
  - **Most VMMs implement virtual CPU (VCPU) to represent state of CPU per guest as guest believes it to be**
    - When guest context switched onto CPU by VMM, information from VCPU loaded and stored

***Dual-mode operation** forms the basis for I/O protection, memory protection and CPU protection. In **dual-mode operation**, there are two separate **modes**: monitor **mode** (also called 'system **mode**' and 'kernel **mode**') and user **mode**. ... In user **mode**, the CPU is restricted to unprivileged instructions and a specified area of memory.*

Building Block – Trap and Emulate

- An operation system is designed to have full control of the system.
- When an OS is running as a virtual machine in a hypervisor, some of its instructions may conflict with the host operation system.
  - So what does the hypervisor do?
- It emulates the effect of that specific instruction or action without carrying it out. In this way, the host OS is not effected by the guest's actions.
  - This is called trap and emulate.

An OS can be run in a virtual machine (for a variety of reasons), thus becoming a non-privileged program (so it can't run in kernel mode, with all the privileged instructions and registers available). Trap-and-emulate is a technique used by the virtual machine to emulate privileged instructions and registers and pretend to the OS that it's still in kernel mode

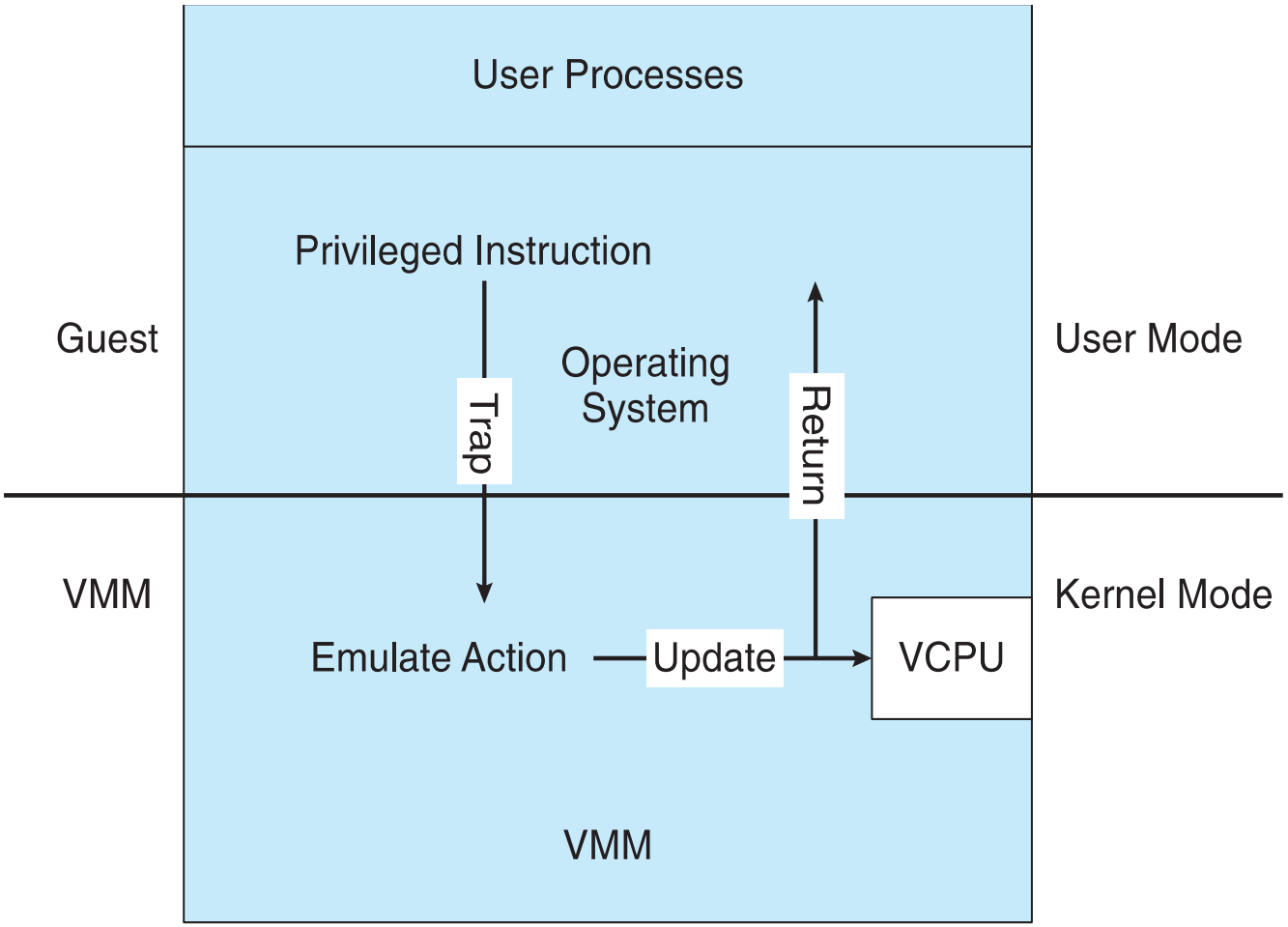
Building Block – Trap and Emulate

- Dual mode CPU means guest executes in user mode
  - Kernel runs in kernel mode
  - Not safe to let guest kernel run in kernel mode too
  - So VM needs two modes – virtual user mode and virtual kernel mode
    - Both of which run in real user mode
  - Actions in guest that usually cause switch to kernel mode must cause switch to virtual kernel mode

Trap-and-Emulate (cont.)

- How does switch from virtual user mode to virtual kernel mode occur?
  - Attempting a privileged instruction in user mode causes an error -> trap
  - VMM gains control, analyzes error, executes operation as attempted by guest
  - Returns control to guest in user mode
  - Known as trap-and-emulate
  - Most virtualization products use this at least in part
- User mode code in guest runs at same speed as if not a guest
- But kernel mode privilege mode code runs slower due to trap-and-emulate
  - Especially a problem when multiple guests running, each needing trap-and-emulate
- CPUs adding hardware support, mode CPU modes to improve virtualization performance

Trap-and-Emulate Virtualization Implementation



Building Block – Binary Translation

- **Binary translation is a system virtualization technique.**
  - **The sensitive instructions in the binary of Guest OS are replaced by either Hypervisor calls which safely handle such sensitive instructions or by some undefined opcodes which result in a CPU trap. Such a CPU trap is handled by the Hypervisor.**
- **On most modern CPUs, context sensitive instructions are Non-Virtualizable.**
  - **Binary translation is a technique to overcome this limitation.**
  - **For example, if the Guest had wanted to modify/read the CPUs Processor Status Word containing important flags/control bitfields, the Host program would scan the guest binary for such instructions and replace them with either a call to hypervisor or some dummy opcode.**
- ***Here the hypervisor includes a binary translator which replaces the sensitive instructions by equivalent non-sensitive instructions at run-time, and leaves non-sensitive instructions unchanged***

Para-Virtualization Binary Translation

- Para-Virtualization on the other hand is a technique where the source code of the guest OS is modified.
- All system resource access related code is modified with Hypervisor APIs.
  - In the case of para-virtualization, the source code has already been modified. Such an image directly calls hypervisor APIs.
  - In the case of Binary translation, the native OS must first scan the guest OS instruction stream and make modifications to the stream as needed. Thus between the two, Para-Virtualization incurs lower overhead

Building Block – Binary Translation

- Some CPUs don't have clean separation between privileged and nonprivileged instructions
  - Earlier Intel x86 CPUs are among them
    - Earliest Intel CPU designed for a calculator
  - Backward compatibility means difficult to improve
  - Consider Intel x86 `popf` instruction
    - Loads CPU flags register from contents of the stack
    - If CPU in privileged mode -> all flags replaced
    - If CPU in user mode -> on some flags replaced
      - No trap is generated

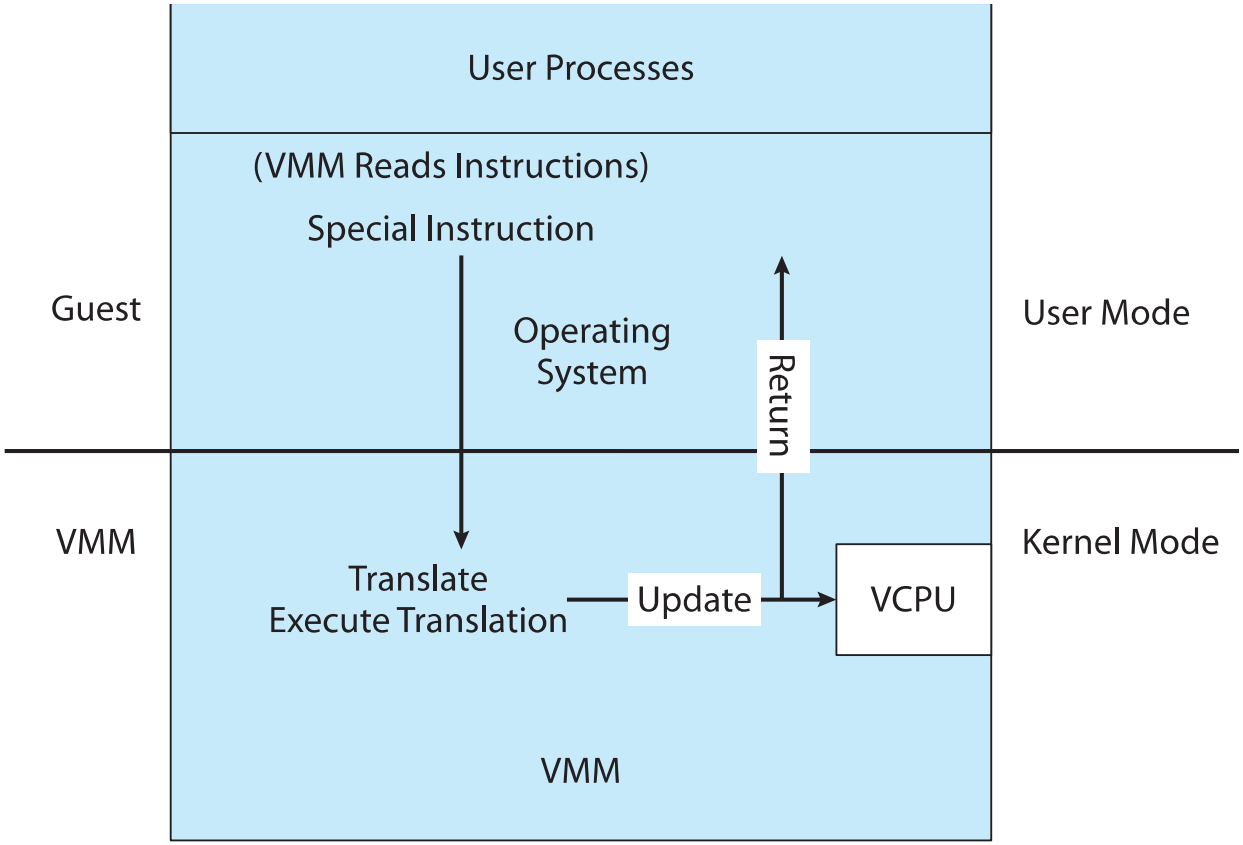
Binary Translation (cont.)

- **Other similar problem instructions we will call *special instructions***
  - Caused trap-and-emulate method considered impossible until 1998
- **Binary translation solves the problem**
  - Basics are simple, but implementation very complex
  1. If guest VCPU is in user mode, guest can run instructions natively
  2. If guest VCPU in kernel mode (guest believes it is in kernel mode)
     1. *VMM examines every instruction guest is about to execute by reading a few instructions ahead of program counter*
     2. *Non-special-instructions run natively*
     3. *Special instructions translated into new set of instructions that perform equivalent task (for example changing the flags in the VCPU)*

Binary Translation (cont.)

- Implemented by translation of code within VMM
- Code reads native instructions dynamically from guest, on demand, generates native binary code that executes in place of original code
- Performance of this method would be poor without optimizations
  - Products like VMware use caching
    - Translate once, and when guest executes code containing special instruction cached translation used instead of translating again

Binary Translation Virtualization Implementation



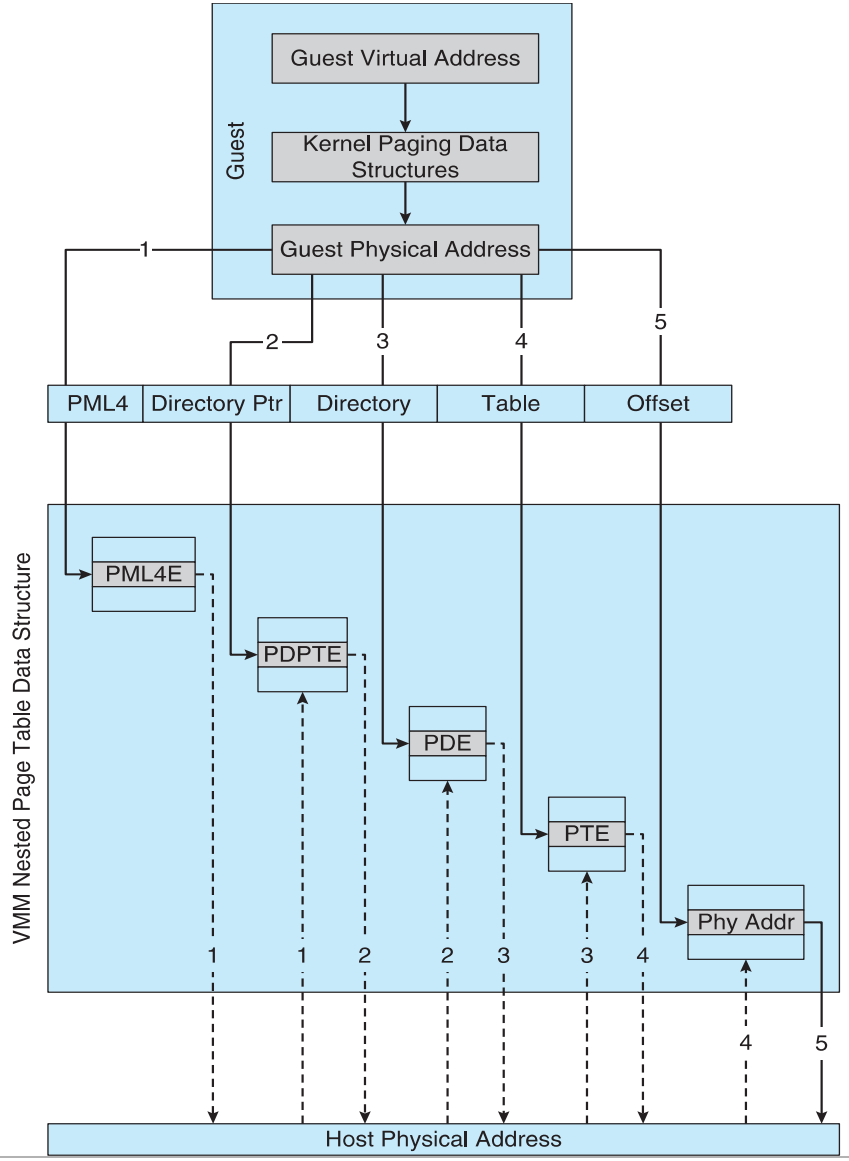
Nested Page Tables

- Memory management another general challenge to VMM implementations
- How can VMM keep page-table state for both guests believing they control the page tables and VMM that does control the tables?
- Common method (for trap-and-emulate and binary translation) is nested page tables (NPTs)
  - Each guest maintains page tables to translate virtual to physical addresses
  - VMM maintains per guest NPTs to represent guest's page-table state
    - Just as VCPU stores guest CPU state
  - When guest on CPU -> VMM makes that guest's NPTs the active system page tables
  - Guest tries to change page table -> VMM makes equivalent change to NPTs and its own page tables
  - Can cause many more TLB misses -> much slower performance
    - TLB -Translation Look-aside Buffer – TLB caches recently used Virtual to Physical mappings

Building Blocks – Hardware Assistance

- All virtualization needs some HW support
- More support -> more feature rich, stable, better performance of guests
- Intel added new VT-x instructions in 2005 and AMD the AMD-V instructions in 2006
  - CPUs with these instructions remove need for binary translation
  - Generally define more CPU modes – “guest” and “host”
  - VMM can enable host mode, define characteristics of each guest VM, switch to guest mode and guest(s) on CPU(s)
  - In guest mode, guest OS thinks it is running natively, sees devices (as defined by VMM for that guest)
    - Access to virtualized device, priv instructions cause trap to VMM
    - CPU maintains VCPU, context switches it as needed
- HW support for Nested Page Tables, DMA, interrupts as well over time

Nested Page Tables



Types of Virtual Machines and Implementations

- **Many variations as well as HW details**
  - **Assume VMMs take advantage of HW features**
    - HW features can simplify implementation, improve performance
- **Whatever the type, a VM has a lifecycle**
  - **Created by VMM**
  - **Resources assigned to it (number of cores, amount of memory, networking details, storage details)**
  - **In type 0 hypervisor, resources usually dedicated**
  - **Other types dedicate or share resources, or a mix**
  - **When no longer needed, VM can be deleted, freeing resources**
- **Steps simpler, faster than with a physical machine install**
  - **Can lead to virtual machine sprawl with lots of VMs, history and state difficult to track**

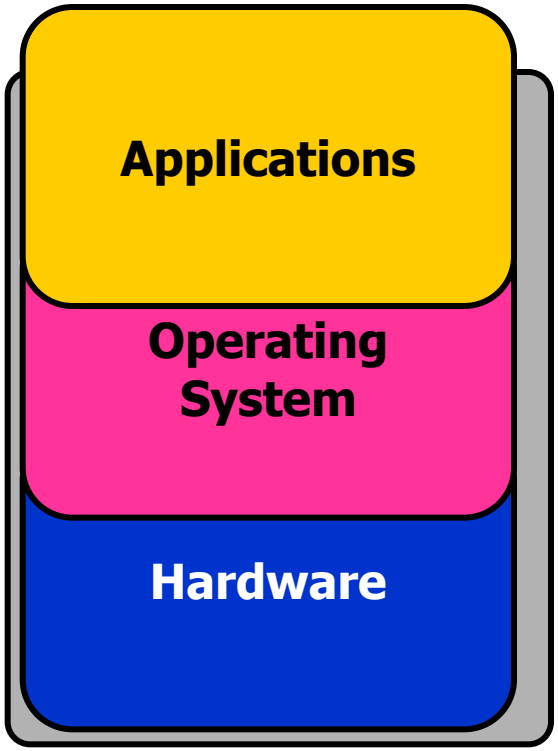
Types of Virtualization

- Virtual memory
- Desktop virtualization
- Platform virtualization
  - Full virtualization
  - Paravirtualization
  - Hardware-assisted virtualization
  - Partial virtualization
  - OS-level virtualization
  - Hosted environment (e.g. User-mode Linux)
- Storage virtualization
- Network virtualization
- Application virtualization Portable application
  - Cross-platform virtualization
  - Emulation or simulation
  - Hosted Virtual Desktop

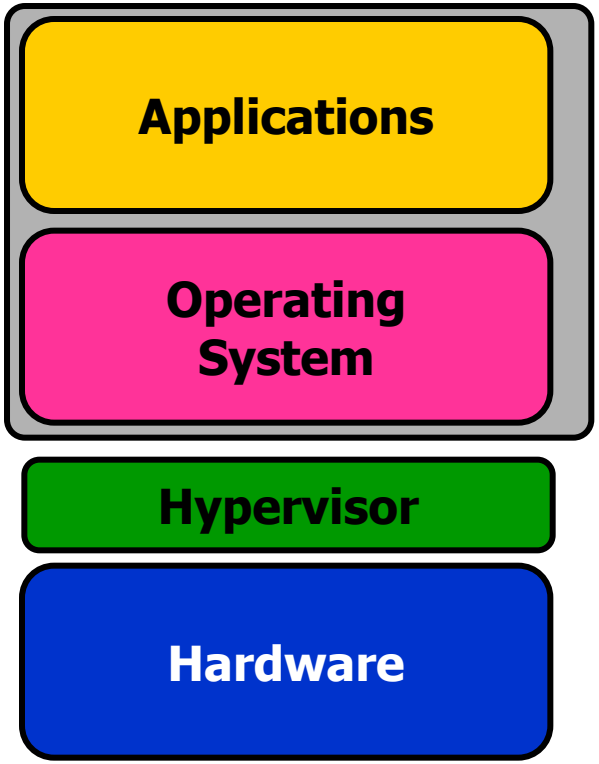
- In this talk, we mainly focus on Platform virtualization which is mostly related to cloud-computing
  - Full virtualization
  - Binary translation
  - Hardware-assisted virtualization
  - Paravirtualization
  - OS-level virtualization
  - Hosted environment (e.g. User-mode Linux)

  - Hardware level
  - Operating system level
  - Application level

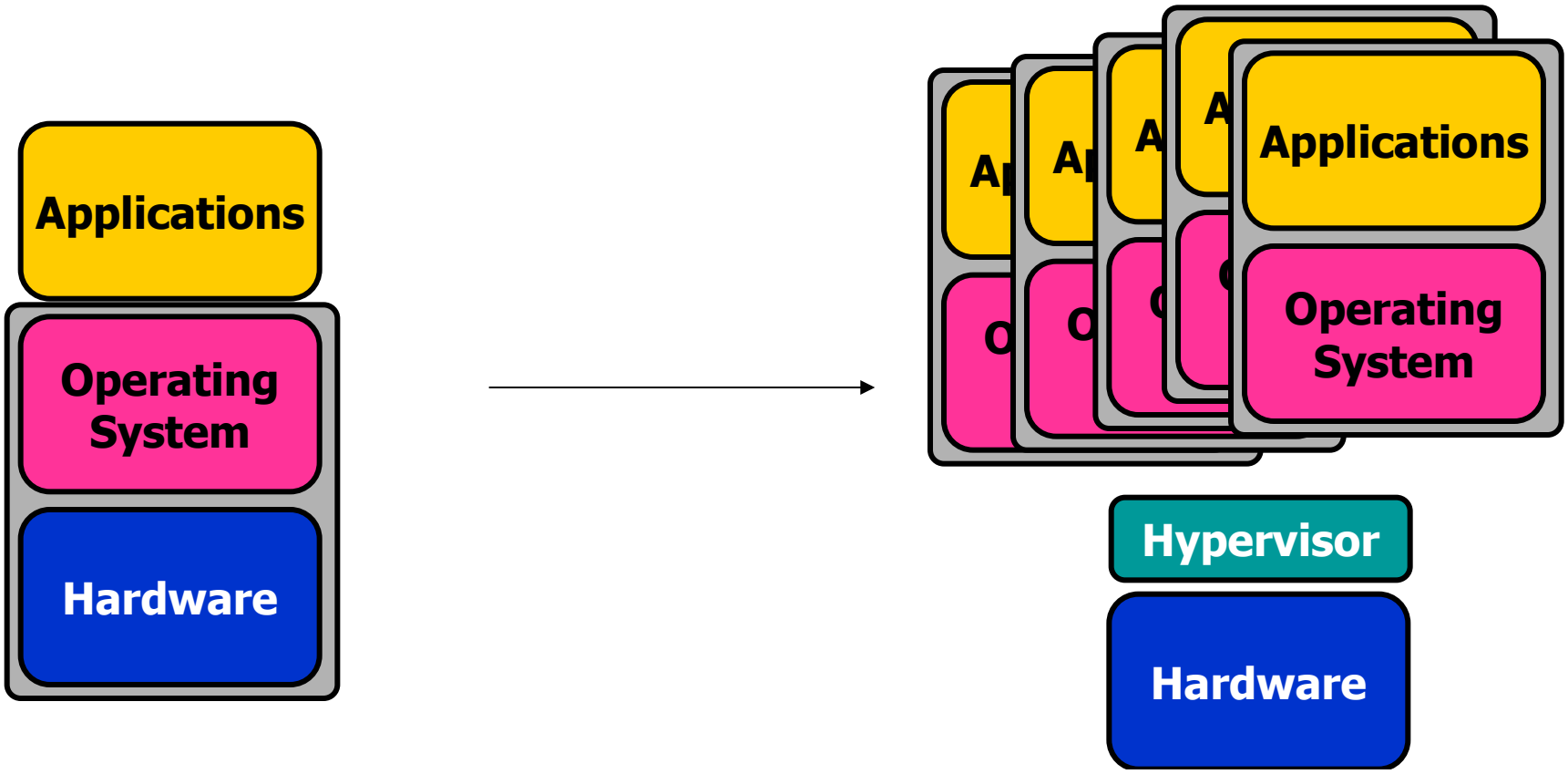
The Use of Computers



Virtualization



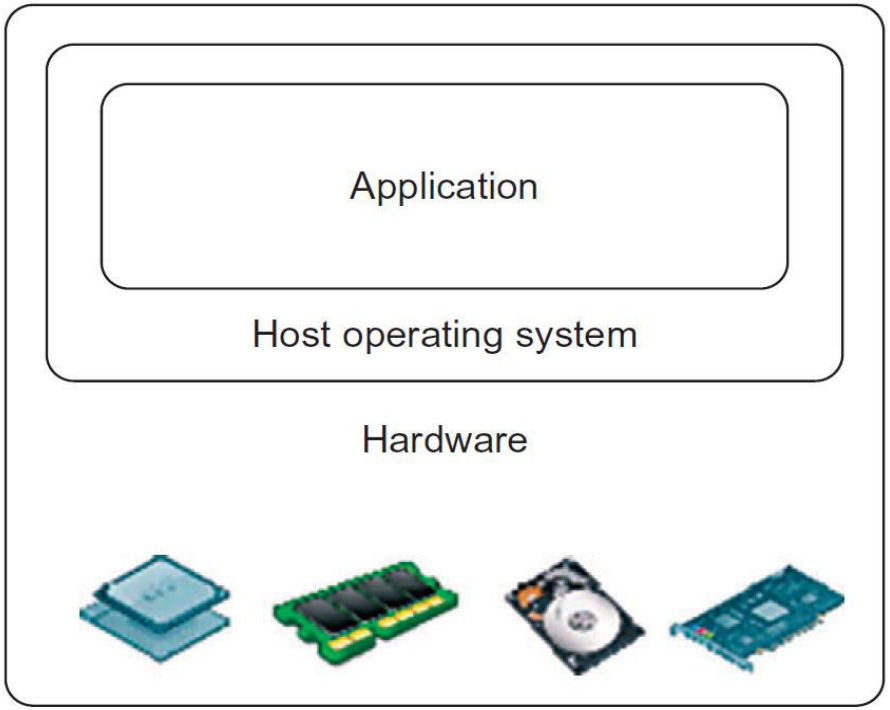
Virtualization -- a Server for Multiple Applications/OS



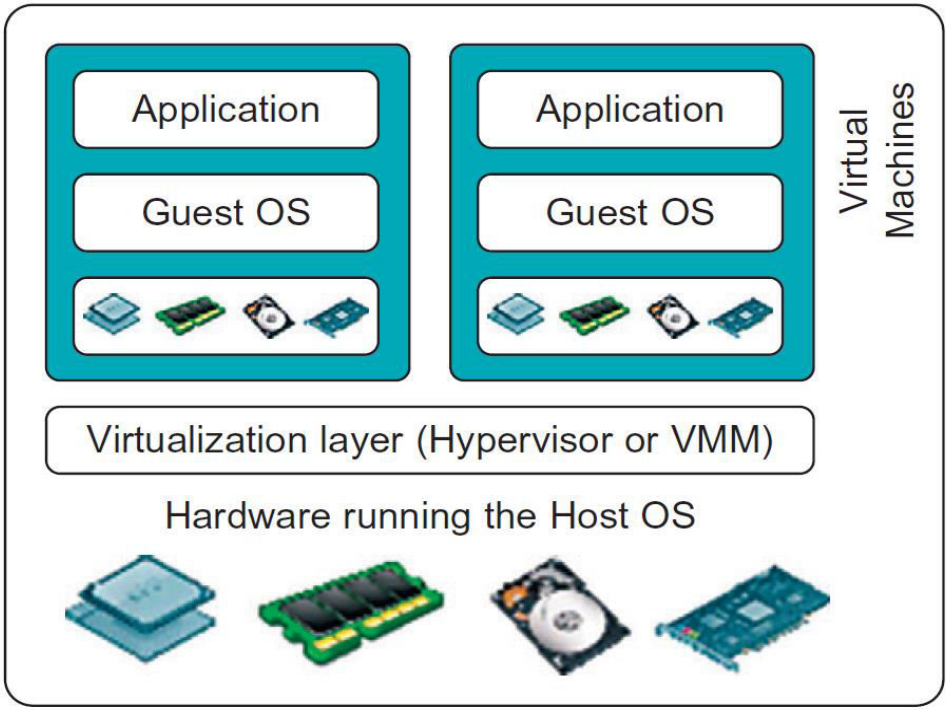
Hypervisor is a software program that manages multiple operating systems (or multiple instances of the same operating system) on a single computer system.
The hypervisor manages the system's processor, memory, and other resources to allocate what each operating system requires.
Hypervisors are designed for a particular processor architecture and may also be called **virtualization managers**.

Basic Concept of Machine Virtualization (cont.)

- **Applications run with their own guest OS over the virtualized CPU, memory, and I/O resources**



(a) Traditional computer



(b) After virtualization

Basic Concept of Machine Virtualization (cont.)

- **A VM is essentially built as a software package**
  - **Can be loaded into a host computer to execute certain user applications**
    - Once the jobs are done, the VM package can be removed from the host computer
  - **The host acts like a *hotel* to accommodate different *guests* at different timeframes**
    - VMs offer a high degree of resources shared within a computer
  - **Multiple VMs can co-exist in the same host**
    - As long as the host has enough memory to handle the guest VMs

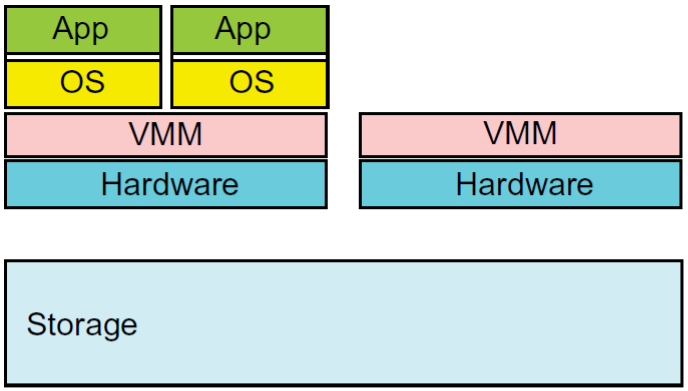
Basic Concept of Machine Virtualization (cont.)

- **Two guest VMs are being hosted by the computer**
  - **The two VMs can run with different guest OS**
  - **The virtual resources allocated to each VM**
    - Known as virtual processors, virtual memory, virtual disks, or virtual I/O devices
- **Virtualization technology primarily benefits the computer and IT industry**
  - **Allows for the sharing of expensive hardware resources**
    - By multiplexing VM on the same set of hardware hosts

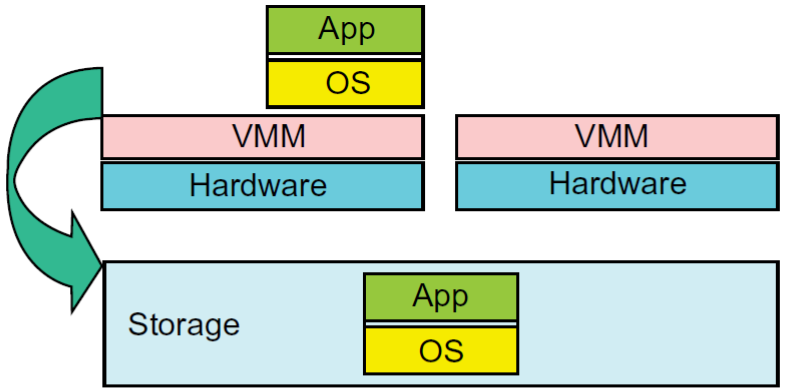
Virtualization Operations

- **The *virtual machine monitor (VMM)* provides the VM abstraction to guest OSs**
  - **With full virtualization, the VMM exports a VM abstraction identical to the physical machine (PM)**
    - A standard OS such as Windows 2016/2019 or Linux can run just as it would on the physical hardware
- **Four categories of basic VMM operations**
  - **Can be multiplexed between hardware machines**
  - **Can be suspended and stored in a stable storage**
  - **A suspended VM can be resumed or provisioned to a new hardware platform**
  - **Can be migrated from one hardware platform to another**

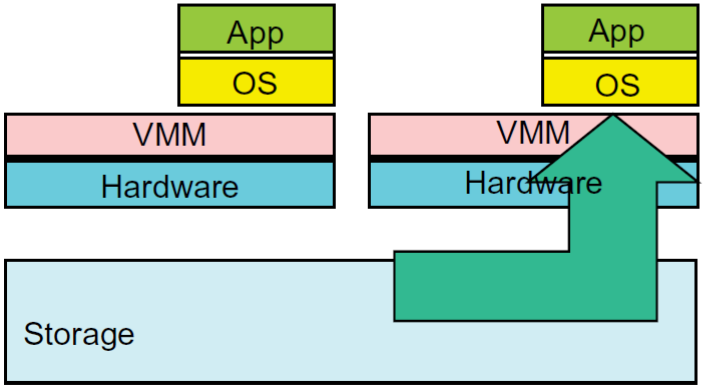
Virtualization Operations (cont.)



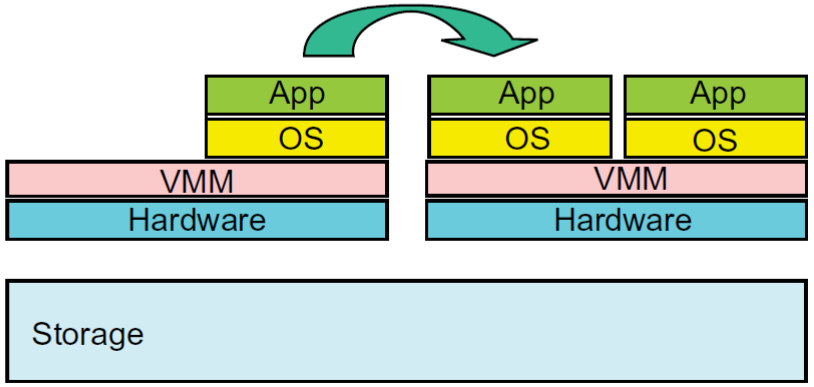
(a) Multiplexing



(b) Suspension (storage)



(c) Provision (resume)



(d) Life migration

Implementation Levels of Virtualization (cont.)

Level of Virtualization	Functional Description	Example Packages	Merits, App Flexibility/ Isolation, Implementation Complexity
Instruction Set Architecture	Emulation of a guest ISA by host	Dynamo, Bird, Bochs, Crusoe	Low performance, high app flexibility, median complexity and isolation
Hardware-Level Virtualization	Virtualization on top of bare-metal hardware	XEN, VMWare, Virtual PC	High performance and complexity, median app flexibility, and good app isolation
Operating System Level	Isolated containers of user app with isolated resources	Docker Engine, Jail, FVM	Highest performance, low app flexibility and best isolation, and average complexity
Run-Time Library Level	Creating VM via run-time library through API hooks	Wine, vCUDA, WABI, LxRun	Average performance, low app flexibility and isolation, and low complexity
User Application Level	Deploy HLL VMs at user app level	JVM, .NET CLR, Panot	Low performance and app flexibility, very high complexity and app isolation

Hardware Virtualization

- **The use of special software to create a VM on a host hardware machine**
  - **The VM acts like a real computer with a guest OS**
  - **The host machine is the actual machine where the VM is executed**
  - **The software that creates VM on the host hardware is called a hypervisor or VMM**
- **Three types of hardware virtualization**
  - **Full virtualization**
    - A complete simulation or translation of the host hardware to some sort of virtual CPU, virtual memory, or virtual disks
    - The VM uses its own unmodified OS

Hardware Virtualization (cont.)

- **Partial virtualization**
  - Selected resources are virtualized and some are not
  - Some guest programs must be modified to run in such an environment
- **Host-based virtualization**
  - The hardware environment of the VM is not virtualized
  - The guest applications are executed in an isolated domain, sometimes called software containers
  - A VMM is installed at the user space to guide the execution of user programs
  - The guest OS must be modified

Hypervisors for Creating Native Virtual Machines

- **Traditional computers are PMs**
  - **Each physical host runs with its own OS**
- **A VM is a software-defined abstract machine created by virtualization**
  - **A physical computer running an OS X can execute application programs only specially tailored to the X platform**
    - Other programs written for a different OS Y may not be executable on the X-platform
  - **The guest OS of VMs can differ from the host OS**
    - e.g., The X-platform is an Apple OS and the Y-platform could be a Windows-based computer
    - VMs offer a solution to bypass the software portability barrier

Virtual Machine Architecture Types

- **The host machine is equipped with the physical hardware**
  - **e.g., A desktop with x-86 architecture runs its installed Windows OS**
  - **VM can be provisioned to any hardware system**
    - The VM approach offers hardware independence of the OS and applications
    - Built with virtual resources managed by a guest OS to run a specific application
    - Between the VMs and the host platform needs to deploy a middleware layer known as VMM
- **On a native VM**
  - **The VM consists of the user application controlled by a guest OS**

Virtual Machine Architecture Types (cont.)

- **Created by the hypervisor installed at the privileged system space**
  - Multiple VMs can be ported to one physical computer
  - e.g., The guest OS could be a Linux system and the hypervisor the XEN system
  - The VM approach extends the software portability beyond the platform boundaries
- **This hypervisor approach is also called bare-metal VM**
  - This hypervisor sits right on top of the bare metal
  - A hypervisor handles the bare hardware directly

- **On a hosted VM**
  - **The VMM runs with a nonprivileged mode**

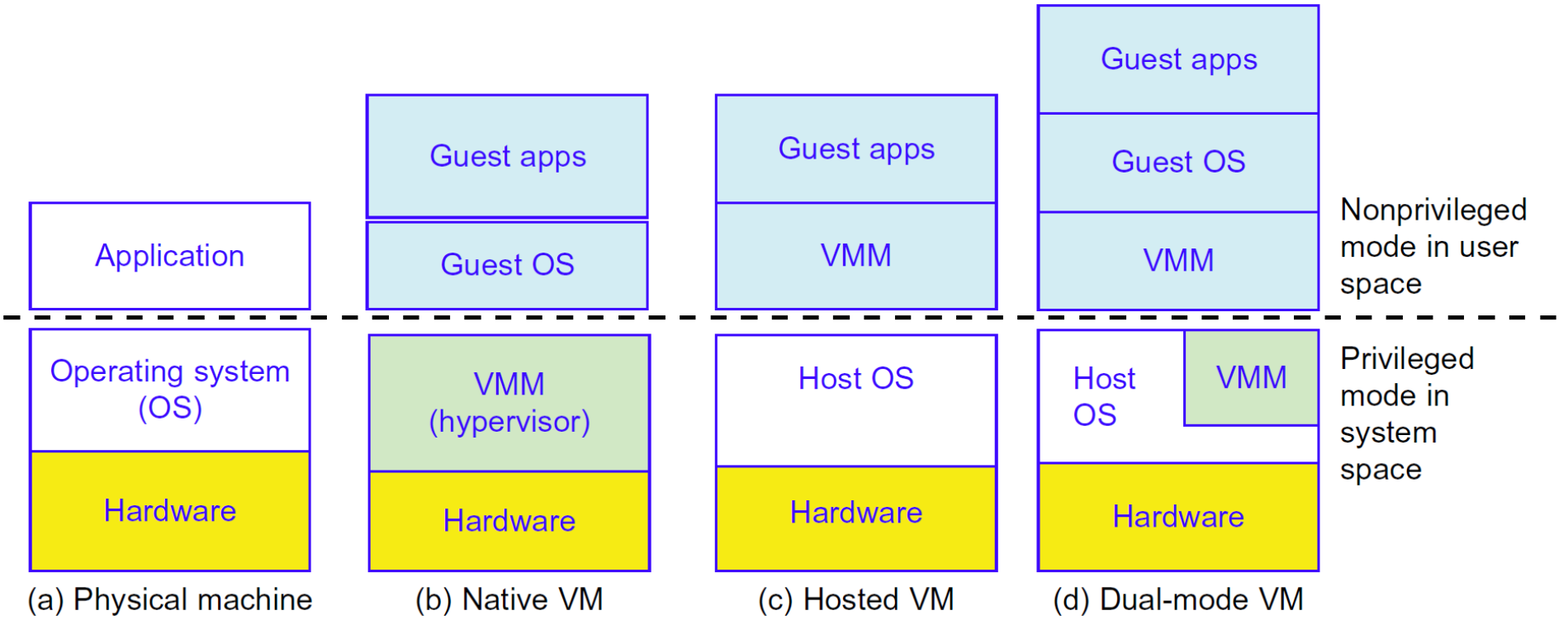
Virtual Machine Architecture Types (cont.)

- **The host OS does not need to be modified**
- **Created by a VMM or a hosted hypervisor implemented on top of the host OS**
  - The VMM is a middleware between the host OS and the user application
  - Replaces the guest OS used in a native VM
  - The VMM monitors the execution of the user application directly
  - Thus abstracts the guest OS from the host OS
- **VMware Workstation, VM player, and VirtualBox are examples of hosted VMs**

- **The VM can be also implemented with a dual mode**

Virtual Machine Architecture Types (cont.)

- Part of VMM runs at the user level
- Another portion runs at the privileged level
- In this mixed mode, the host OS may have to be modified to some extent



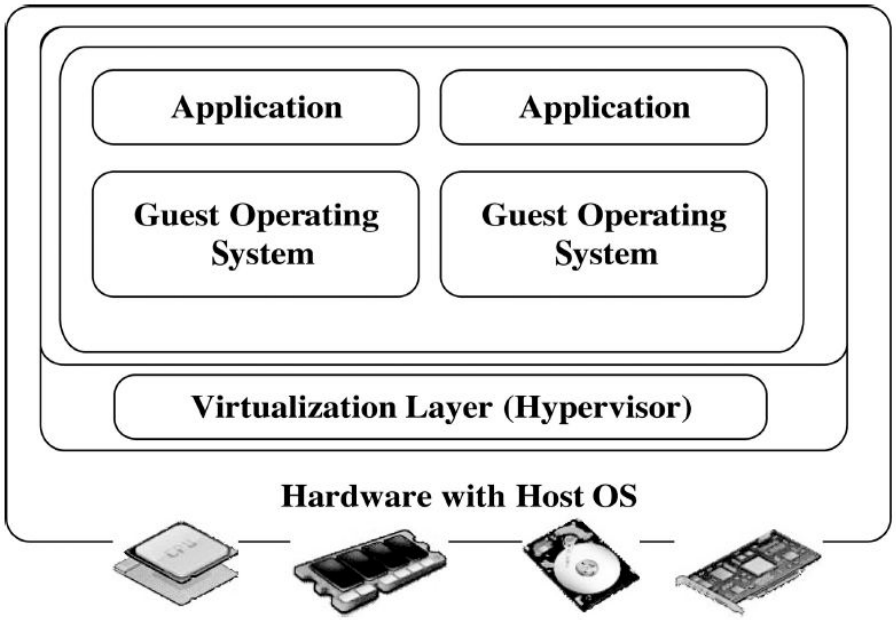
Virtual Machine Architecture Types (cont.)

- **The VMM layer virtualizes the physical hardware resources**
  - **e.g., CPU, memory, network and disk controllers, and human interface devices**
    - Any VM has its own set of virtual hardware resources
  - **This resource manager allocates CPU, memory disk, and network bandwidth**
    - Maps them to the virtual hardware resource set of each VM created
- **The user application running on its dedicated OS can be bundled together**
  - **As a virtual appliance ported on any hardware platform**

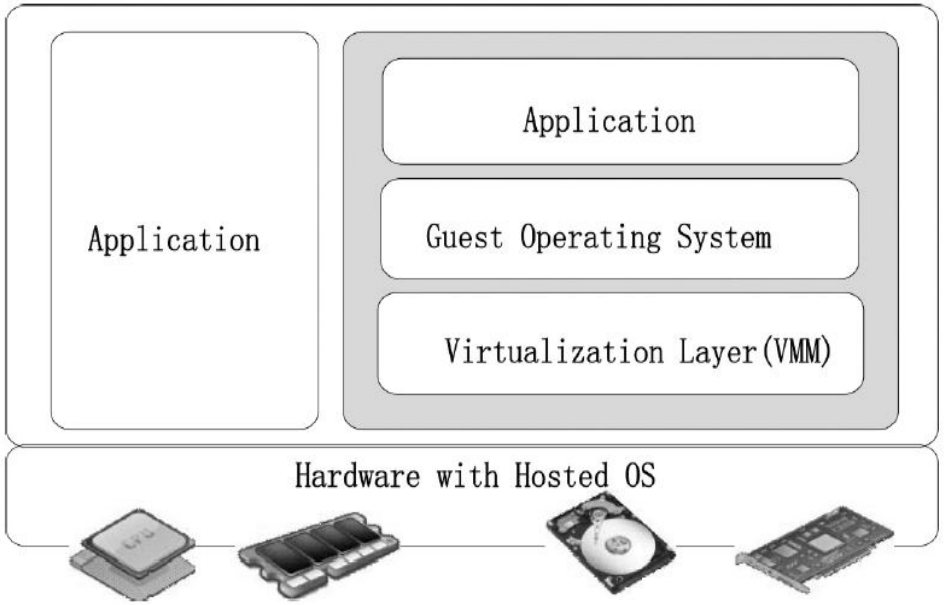
Hardware virtualization

- **Can be classified into two categories**
  - **Full virtualization and host-based virtualization**
    - Depending on implementation technologies
  - **Full virtualization does not need to modify the guest OS**
    - The guest OS and their applications consist of noncritical and critical instructions
    - Relies on binary translation to trap and virtualize the execution of certain sensitive, nonvirtualizable instructions
  - **In a host-based system, both a host OS and a guest OS are used**
    - A virtualization software layer is built between the host OS and guest OS

Hardware virtualization (cont.)



(a) Full virtualization

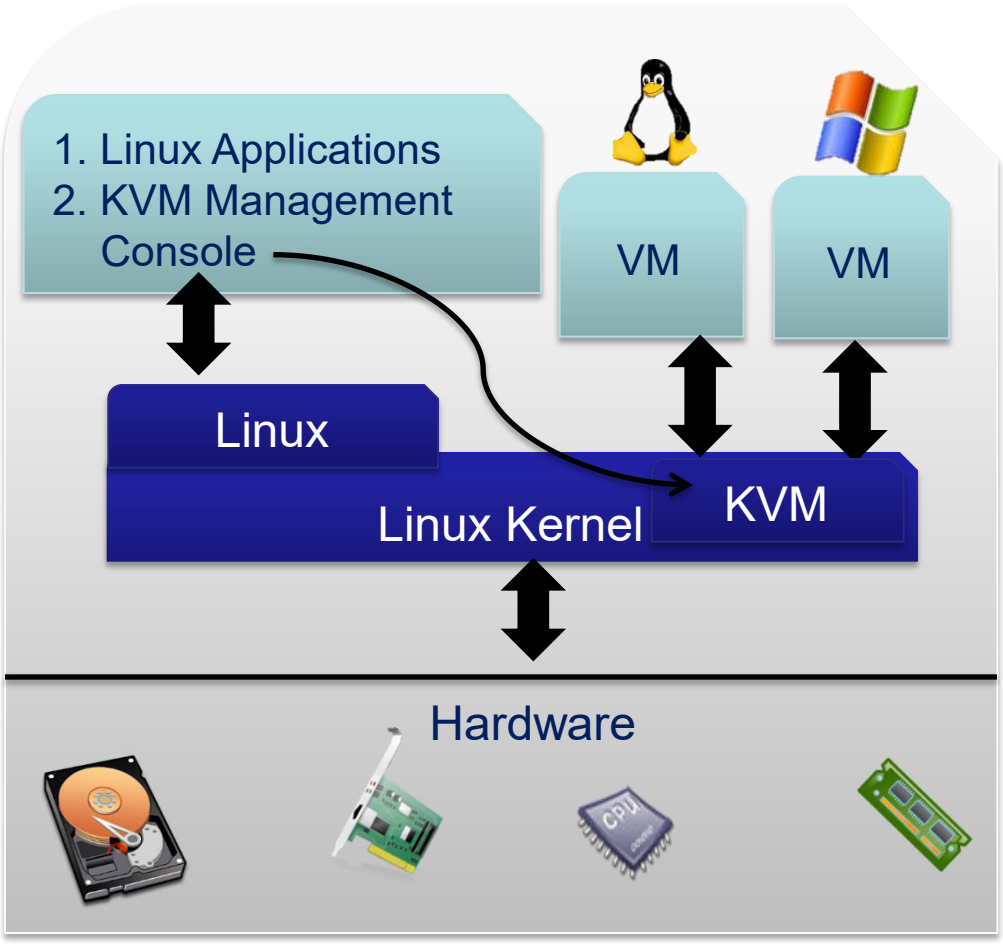


(b) host-based virtualization

Full Virtualization

- **With full virtualization**
  - **The hypervisor/VMM approach is considered full virtualization**
  - **Critical instructions are discovered and replaced with traps into the VMM**
    - To be emulated by software
    - Only critical instructions are trapped in the VMM
  - **Noncritical instructions run on hardware directly**
    - Binary translation incurs large performance overhead
  - **Noncritical instructions do not control hardware and threaten the security of the system**
    - Running noncritical instructions on hardware can promote efficiency and also ensure system security

Ubuntu kvm



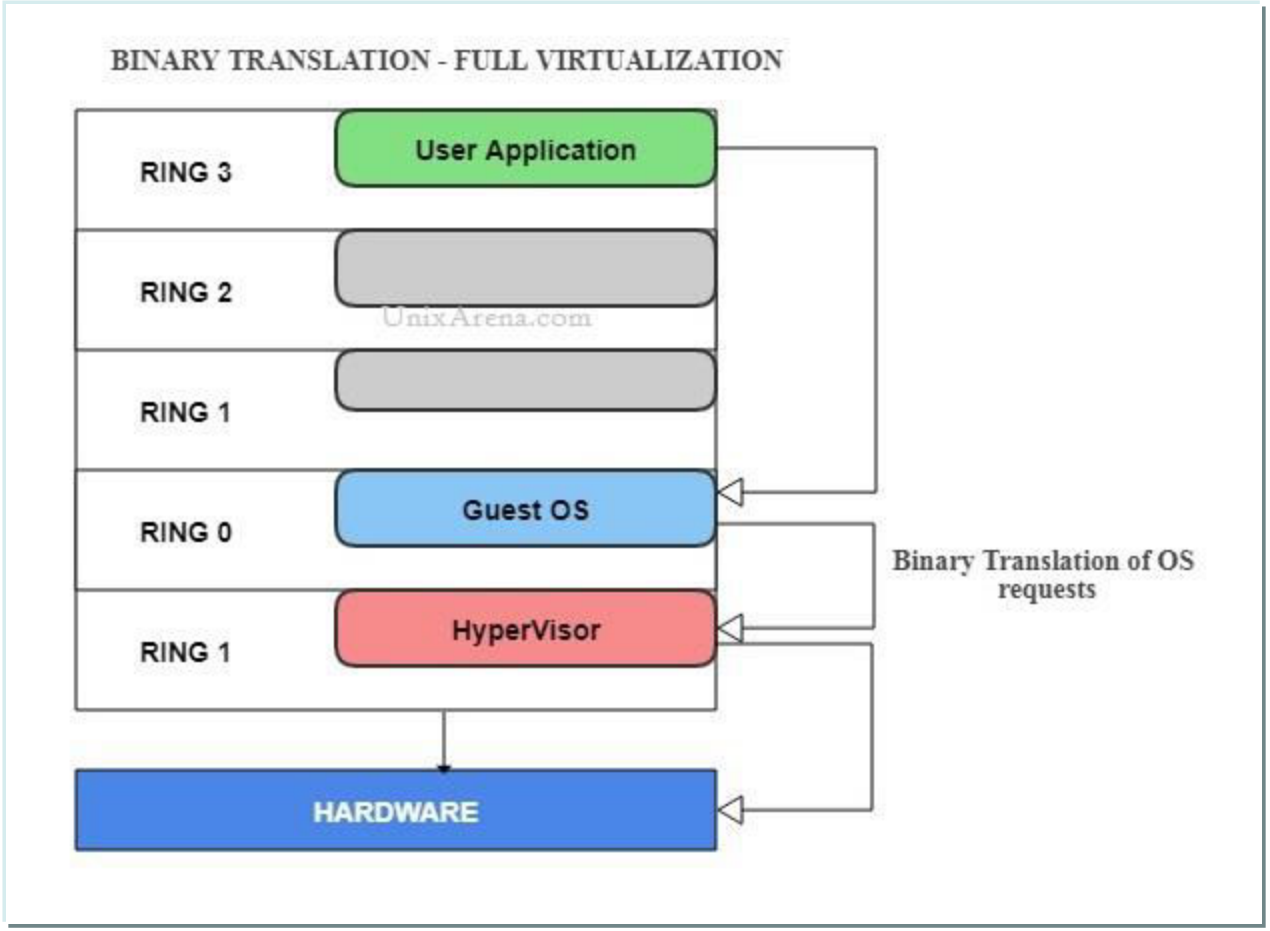
Architecture of KVM

- Kernel based virtual machine (Kernel Based VM)
- Open source.
- Kernel-level extension to Linux.
- Full virtualization.
- Supports full virtualization and hence does not need hardware assisted virtualization support in the CPU.

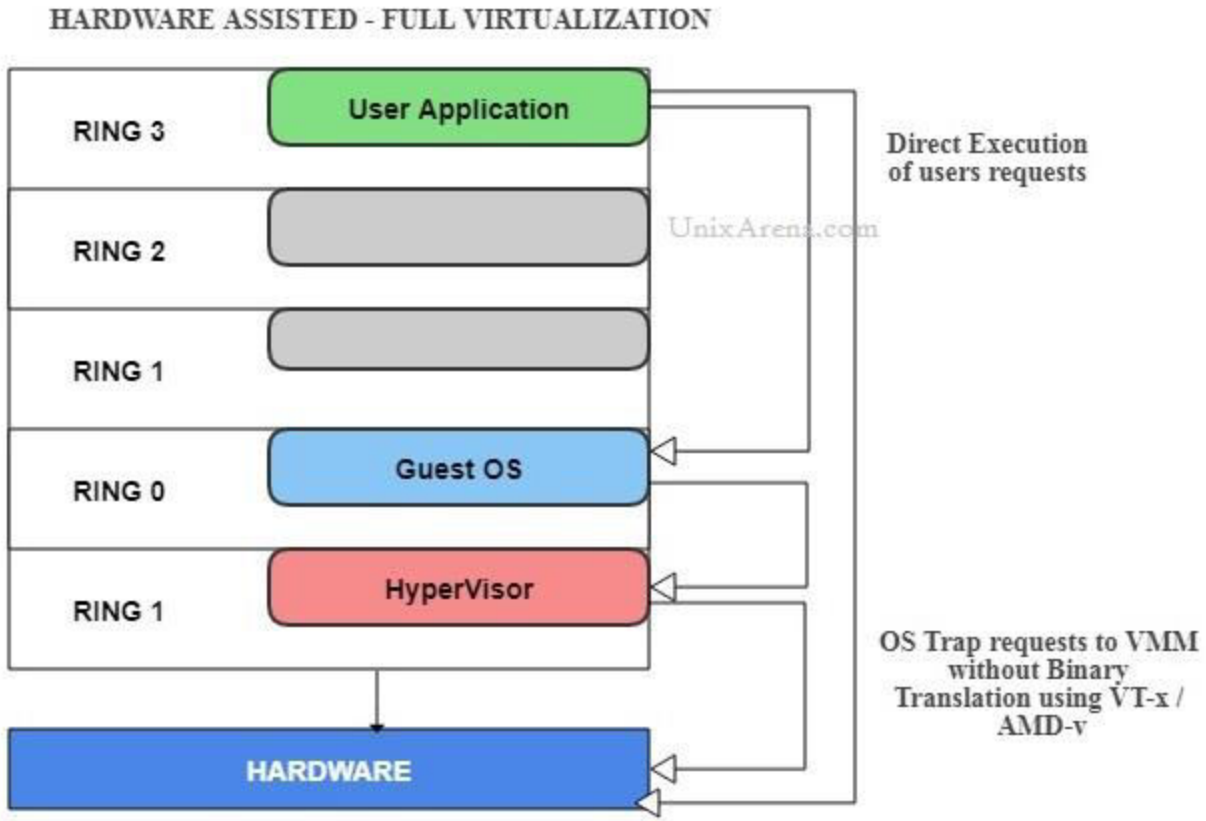
Binary translation

- Kernel code of non-virtualizable instructions are translated to replace with new sequences of instructions that have the intended effect on the virtual hardware. Each virtual machine monitor provides each Virtual Machine with all the services of the physical system, including a virtual BIOS, virtual devices and virtualized memory management.
- This combination of binary translation and direct execution provides Full Virtualization as the guest OS is fully abstracted (completely decoupled) from the underlying hardware by the virtualization layer. The guest OS is not aware it is being virtualized and requires no modification.
- The hypervisor translates all operating system instructions on the fly and caches the results for future use, while user level instructions run unmodified at native speed.
- Examples
  - VMware
  - Microsoft Virtual Server

Binary translation



Hardware-assisted virtualization – Hypervisor



Binary Translation of Guest OS Requests Using a VMM

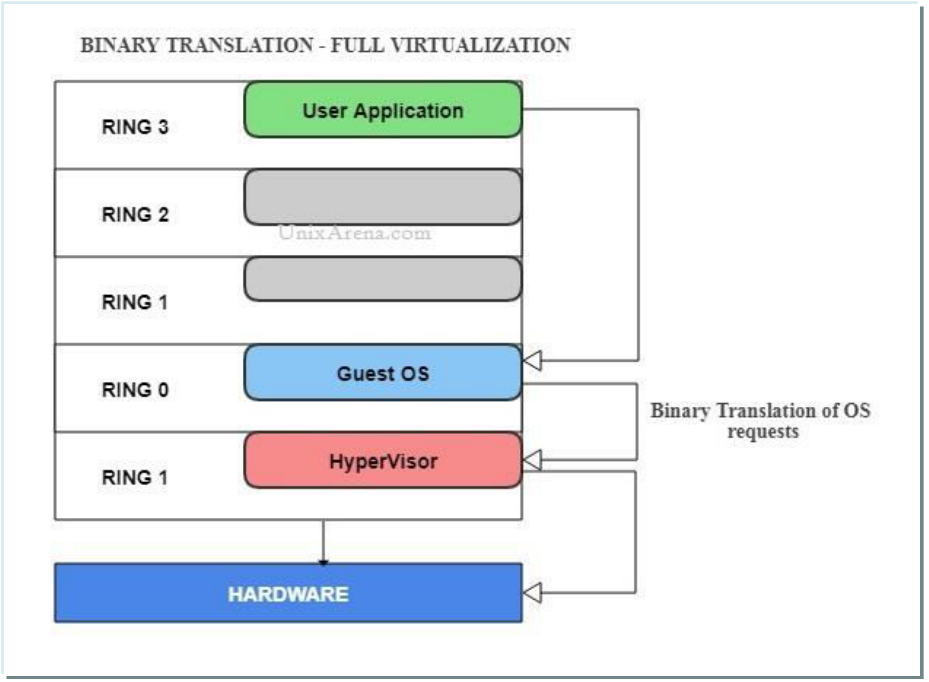
- **Implemented by VMware and many other software companies**
- **The traditional x86 processor offers four instruction execution rings**
  - **Ring *0, 1, 2* and *3***
    - The lower the ring number, the higher the privilege of instruction being executed
    - The OS is responsible for managing the hardware and the privileged instructions to execute at ring *0*
    - User level applications run at ring *3*
  - **VMware puts the VMM at Ring *0***
  - **The guest OS is put at Ring *1***
  - **The VMM scans the instruction stream**

Binary Translation of Guest OS Requests Using a VMM (cont.)

  - Identifies the privileged, control, and behavior-sensitive instructions
- **When these instructions are identified**
  - They are trapped in the VMM
  - The VMM emulates the behavior of these instructions
  - The method used in emulation is called binary translation
- **Full virtualization combines binary translation and direct execution**
- **The guest OS is completely decoupled from the underlying hardware**
  - Consequently, the guest OS is not modified
- **The performance of full virtualization may not be ideal**

Binary Translation of Guest OS Requests Using a VMM (cont.)

- **Involves binary translation**
  - Rather time consuming and thus low performance
  - Speeding up binary translation is difficult to achieve
- **The full virtualization of I/O-intensive applications is a big challenge**
- **Binary translation employs code cache**
  - To store translated hot instructions to improve the performance but increases the cost of memory usage
- **The performance of full virtualization on x86 architecture**
  - Typically *80-97%* **that of the host machine**



Host-Based Virtualization

- **An alternative VM architecture**
  - **Install a virtualization layer on top of the host OS**
    - The user can install this VM architecture without modifying the host OS
  - **The host OS is still responsible for managing the hardware**
    - The guest OSs are installed and run on top of the virtualization layer
  - **Dedicated applications may run on the VMs**
    - Certainly some other applications can also run with the host OS directly
  - **The virtualizing software can rely on the host OS**
    - To provide device drivers and low-level services
    - This will ease the VM design and its deployment

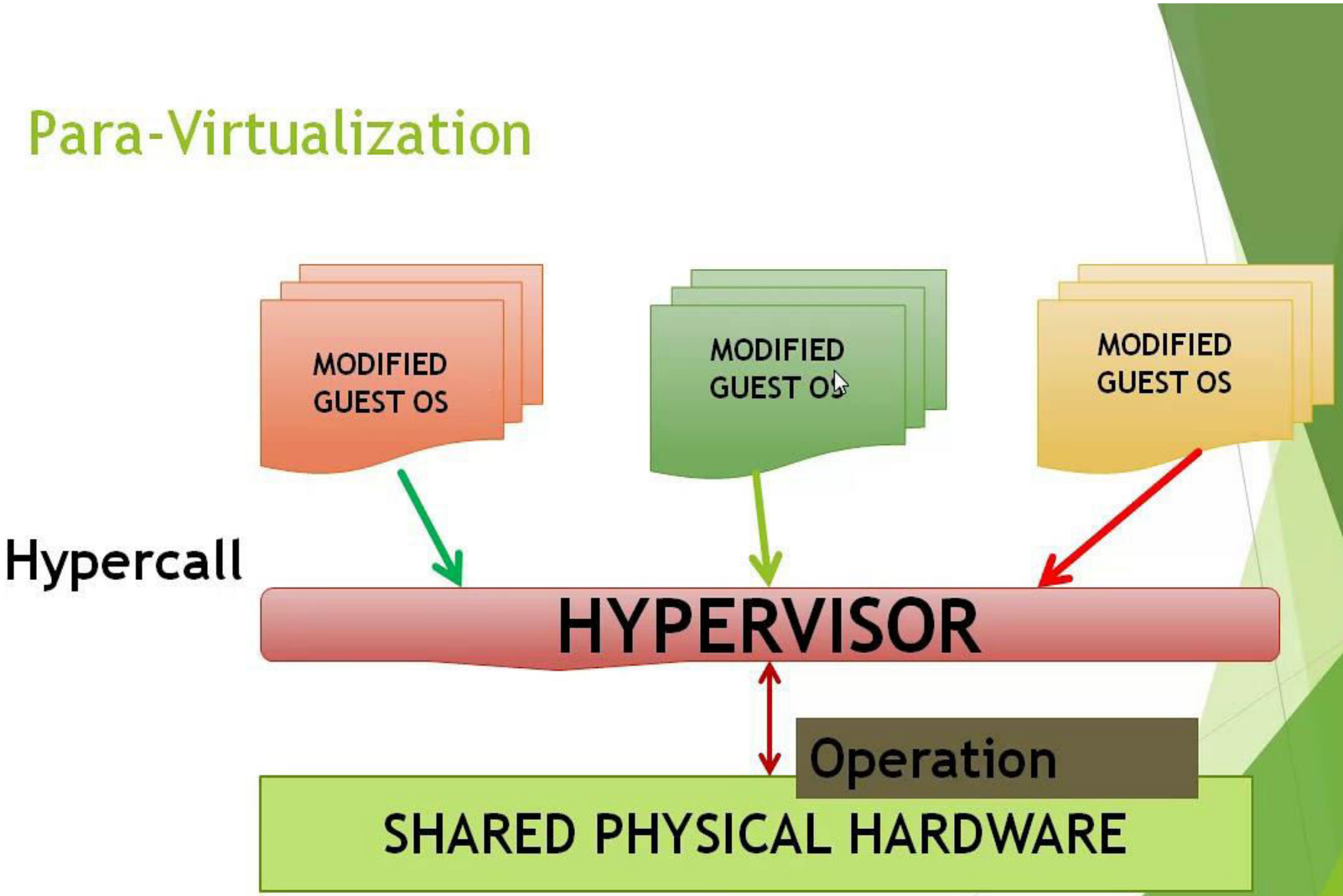
Host-Based Virtualization (cont.)

- **The performance of host-based architecture may be low**
  - **Compared to the hypervisor/VMM architecture**
  - **When an application requests hardware access**
    - Involves four layers of mapping
    - Downgrade the performance significantly
  - **When the ISA of a guest OS is different from the ISA of the underlying hardware**
    - Binary translation must be adopted
  - **The host-based architecture has flexibility**
    - But the performance is too low to be useful in practice

Paravirtualization

- Modify Guest OS so that all calls to sensitive instructions are changed to hypervisor calls.
- Much easier (and more efficient) to modify source code than to emulate hardware instructions (as in binary translation).
- In effect, turns the hypervisor into a microkernel.

Paravirtualization



OS assisted (Paravirtualization)

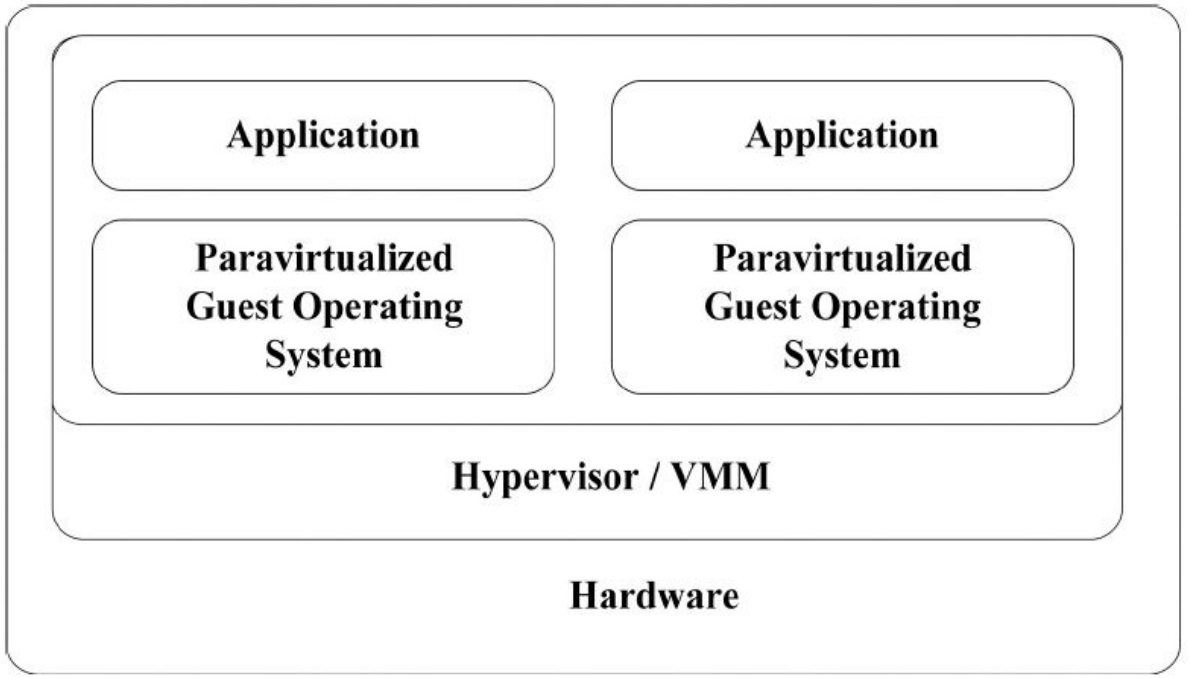
- Paravirtualization – via an modified OS kernel as guest OS
  - It is very difficult to build the more sophisticated binary translation support necessary for full virtualization.
  - Paravirtualization involves modifying the OS kernel to replace non-virtualizable instructions with hypercalls that communicate directly with the virtualization layer hypervisor.
  - The hypervisor also provides hypercall interfaces for other critical kernel operations such as memory management, interrupt handling and time keeping.
  - Paravirtualization is different from full virtualization, where the unmodified OS does not know it is virtualized and sensitive OS calls are trapped using binary translation.
  - Paravirtualization cannot support unmodified OS
- Example:
  - Xen -- modified Linux kernel and a version of Windows 10

Paravirtualization with Guest OS Modification

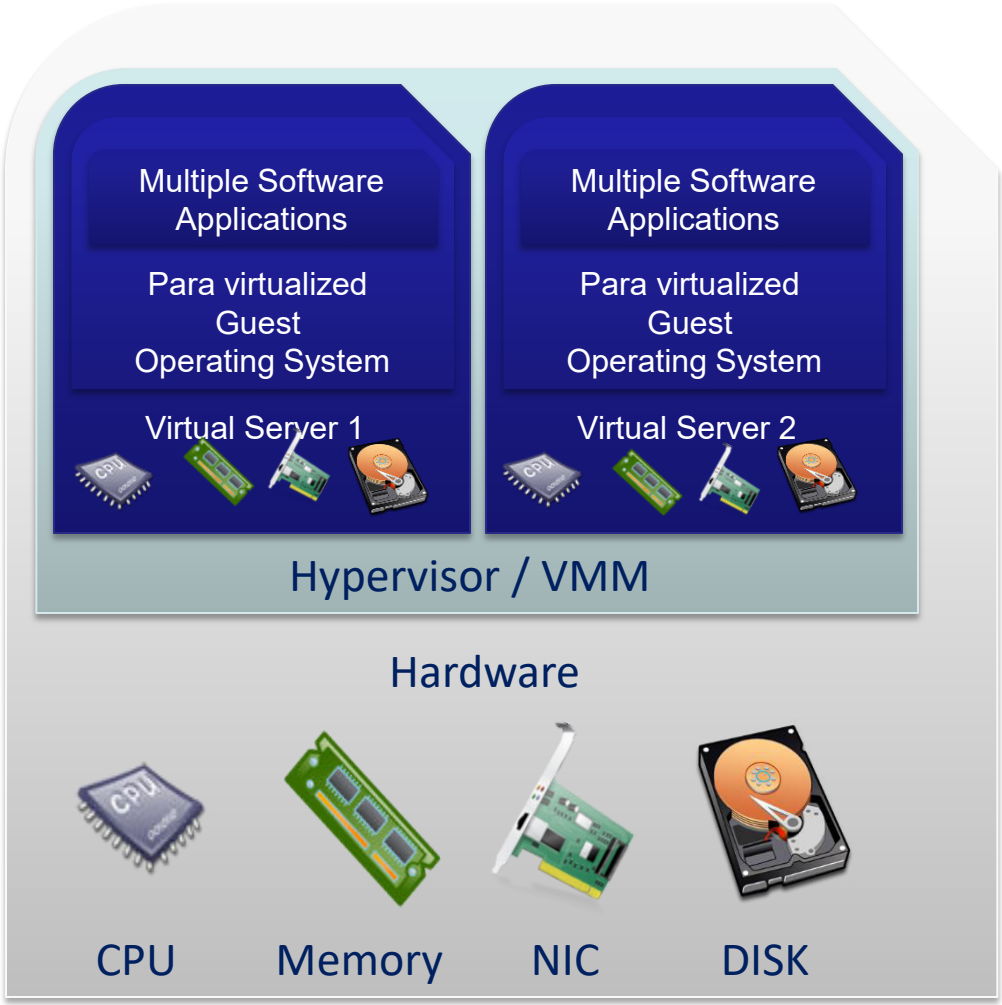
- **The performance degradation is a critical issue of a virtualized system**
  - **No one prefers to use a VM if it is much slower than using a PM**
  - **Paravirtualization attempts to reduce the virtualization overhead**
    - Improve the performance by modifying only the guest OS kernel
    - Such a VM provides special APIs requiring substantial guest OS modifications
  - **The guest operating systems are para-virtualized**
    - Assisted by an intelligent compiler to replace the nonvirtualizable OS instructions by hypercalls
    - Hypercalls communicate directly with the hypervisor

Paravirtualization with Guest OS Modification (cont.)

- **When the guest OS kernel is modified for virtualization**
  - Can no longer run on the hardware directly
- **The concept of paravirtualized VM architecture**



Para virtualization



- Involves explicitly modifying guest operating system (e.g. SUSE Linux Enterprise Server 11) so that it is aware of being virtualized to allow near native performance.
- Improves performance.
- Lower overhead.
- E.g.: Xen supports both Hardware Assisted Virtualization (HVM) and Para-Virtualization (PV).

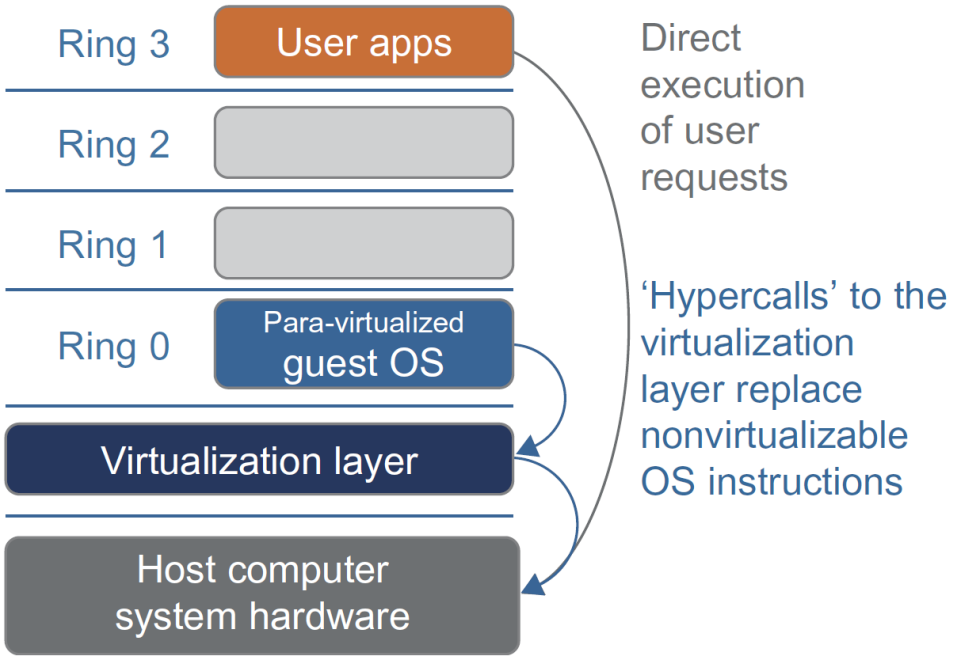
Paravirtualization Architecture

- **Full virtualization intercepts and emulates privileged instructions at runtime**
  - **Paravirtualization handles these at compile time**
    - The guest OS kernel is modified to replace critical instructions with hypercalls to the hypervisor or VMM
- **When the x86 processor is virtualized**
  - **A virtualization layer is inserted between the hardware and the OS**
    - The virtualization layer should be installed at ring *0*
  - **The Xen assumes such an architecture**
    - The guest OS running in a guest domain may run at ring *1* instead of at ring *0*
    - The guest OS may not be able to execute some privileged and sensitive instructions

Paravirtualization Architecture (cont.)

- These instructions are implemented by ***hypercalls***

- **After replacing the instructions with hypercalls**
  - The modified guest OS emulates the behavior of the original guest OS
- **On an UNIX system, a system call involves an interrupt or service routine**
  - Hypercalls apply a dedicated service routine in Xen



Paravirtualization Architecture (cont.)

- **Para-virtualization reduces the overhead**
  - **But its compatibility and portability may be in doubt**
    - Must support the unmodified OS as well
  - **Cost of maintaining paravirtualized OSs is high**
    - May require deep OS kernel modifications
  - **The performance advantage of paravirtualization varies greatly due to workload variations**
  - **Paravirtualization is relatively easy and more practical, compared to full virtualization**
  - **Many virtualization products employ the paravirtualization architecture**
    - e.g., The popular Xen, KVM, and VMware ESX

Types of VMs – Emulation

- Another (older) way for running one operating system on a different operating system
  - Virtualization requires underlying CPU to be same as guest was compiled for
  - Emulation allows guest to run on different CPU
- Necessary to translate all guest instructions from guest CPU to native CPU
  - Emulation, not virtualization
- Useful when host system has one architecture, guest compiled for other architecture
  - Company replacing outdated servers with new servers containing different CPU architecture, but still want to run old applications
- Performance challenge – order of magnitude slower than native code
  - New machines faster than older machines so can reduce slowdown
- Very popular – especially in gaming where old consoles emulated on new

Types of VMs – Application Containment

- Some goals of virtualization are segregation of apps, performance and resource management, easy start, stop, move, and management of them
- Can do those things without full-fledged virtualization
  - If applications compiled for the host operating system, don't need full virtualization to meet these goals
- Oracle containers / zones for example create virtual layer between OS and apps
  - Only one kernel running – host OS
  - OS and devices are virtualized, providing resources within zone with impression that they are only processes on system
  - Each zone has its own applications; networking stack, addresses, and ports; user accounts, etc
  - CPU and memory resources divided between zones
    - Zone can have its own scheduler to use those resources

Virtualization and Operating-System Components

- **Now look at operating system aspects of virtualization**
  - **CPU scheduling, memory management, I/O, storage, and unique VM migration feature**
    - How do VMMs schedule CPU use when guests believe they have dedicated CPUs?
    - How can memory management work when many guests require large amounts of memory?

OS Component – CPU Scheduling

- **Even single-CPU systems act like multiprocessor ones when virtualized**
  - **One or more virtual CPUs per guest**
- **Generally VMM has one or more physical CPUs and number of threads to run on them**
  - **Guests configured with certain number of VCPUs**
    - Can be adjusted throughout life of VM
  - **When enough CPUs for all guests -> VMM can allocate dedicated CPUs, each guest much like native operating system managing its CPUs**
  - **Usually not enough CPUs -> CPU over commitment**
    - VMM can use standard scheduling algorithms to put threads on CPUs
    - Some add fairness aspect

OS Component – CPU Scheduling (cont.)

- **Cycle stealing by VMM and oversubscription of CPUs means guests don't get CPU cycles they expect**
  - **Consider timesharing scheduler in a guest trying to schedule 100ms time slices -> each may take 100ms, 1 second, or longer**
    - Poor response times for users of guest
    - Time-of-day clocks incorrect
  - **Some VMMs provide application to run in each guest to fix time-of-day and provide other integration features**

OS Component – Memory Management

- **Also suffers from oversubscription -> requires extra management efficiency from VMM**
- **For example, VMware ESX guests have a configured amount of physical memory, then ESX uses 3 methods of memory management**
  - **Double-paging, in which the guest page table indicates a page is in a physical frame but the VMM moves some of those pages to backing store**
  - **Install a pseudo-device driver in each guest (it looks like a device driver to the guest kernel but really just adds kernel-mode code to the guest)**
    - **Balloon** memory manager communicates with VMM and is told to allocate or deallocate memory to decrease or increase physical memory use of guest, causing guest OS to free or have more memory available
  - **Deduplication by VMM determining if same page loaded more than once, memory mapping the same page into multiple guests**

OS Component – I/O

- **Easier for VMMs to integrate with guests because I/O has lots of variation**
  - **Already somewhat segregated / flexible via device drivers**
  - **VMM can provide new devices and device drivers**
- **But overall I/O is complicated for VMMs**
  - **Many short paths for I/O in standard OSes for improved performance**
  - **Less hypervisor needs to do for I/O for guests, the better**
  - **Possibilities include direct device access, DMA pass-through, direct interrupt delivery**
    - Again, HW support needed for these
- **Networking also complex as VMM and guests all need network access**
  - **VMM can bridge guest to network (allowing direct access)**
  - **And / or provide network address translation (NAT)**
    - NAT address local to machine on which guest is running, VMM provides address translation to guest to hide its address

OS Component – Storage Management

- Both boot disk and general data access need be provided by VMM
- Need to support potentially dozens of guests per VMM (so standard disk partitioning not sufficient)
- Type 1 – storage guest root disks and config information within file system provided by VMM as a disk image
- Type 2 – store as files in file system provided by host OS
- Duplicate file -> create new guest
- Move file to another system -> move guest
- Physical-to-virtual (P-to-V) convert native disk blocks into VMM format
- Virtual-to-physical (V-to-P) convert from virtual format to native or disk format
- VMM also needs to provide access to network attached storage (just networking) and other disk images, disk partitions, disks, etc