

Operating Systems

Prepared by Dr. Alaa Mohamed

Module 7



Virtualization – (Virtual Machines)

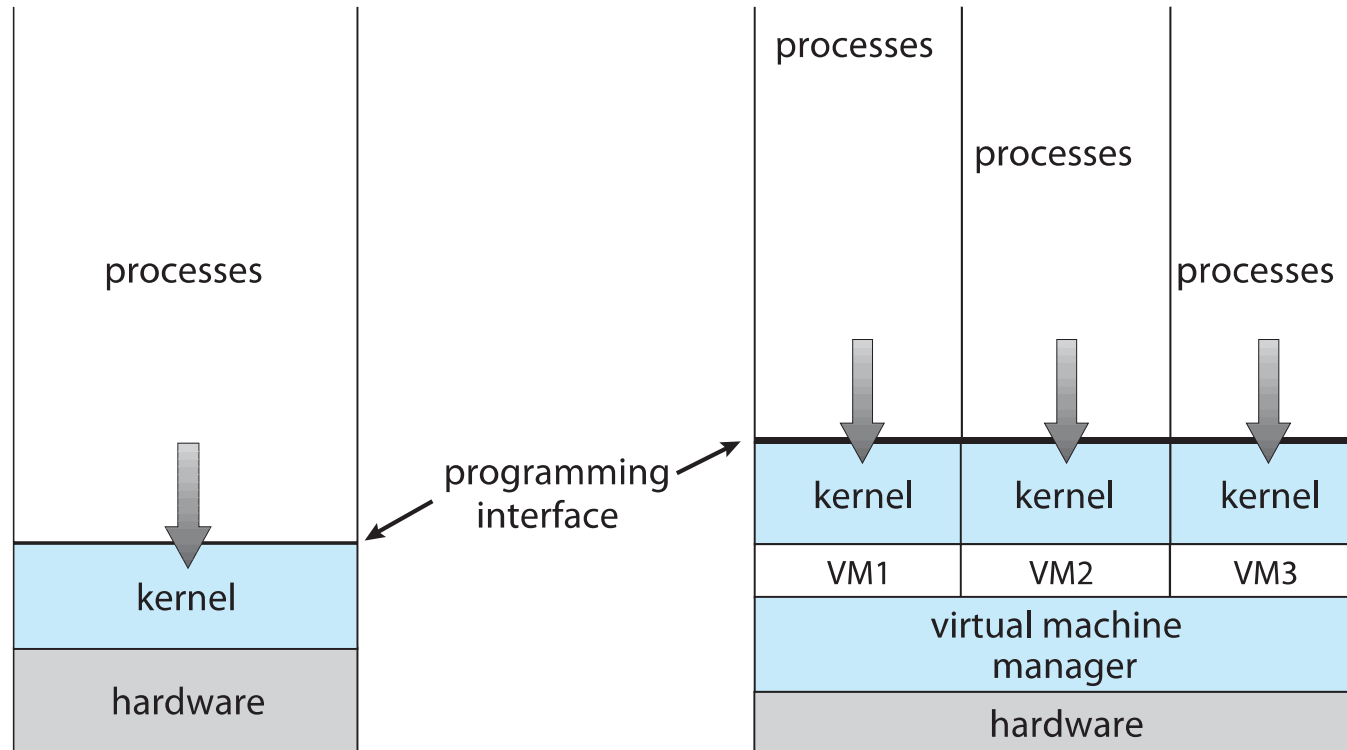


جامعة الجميع الذكية
EVERYONE'S SMART UNIVERSITY

- Overview
- Benefits and Features
- Building Blocks of Virtual Machines
- Types of Virtual Machines and Their Implementations
- Virtualization and Operating-System Components

- Fundamental idea behind a virtual machine is to abstract hardware of a single computer (The Cpu. Memory, Disk drives, Network interface) into several different execution environments.
- Thereby creating The illusion that each separate execution environment is running its own private computer.
- 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



Non-virtual machine

Virtual machine

- Vary greatly, with options including:
 - **Type 0 hypervisors** - Hardware-based solutions that provide support for virtual machine creation and management via firmware
 - IBM LPARs and Oracle LDOMs are examples
 - **Type 1 hypervisors** - Operating-system-like software built to provide virtualization
 - Including VMware ESX, Joyent SmartOS, and Citrix XenServer

- **Type 1 hypervisors** – Also includes general-purpose operating systems that provide standard functions as well as VMM functions
 - Including Microsoft Windows Server with HyperV and RedHat Linux with KVM
- **Type 2 hypervisors** - Applications that run on standard operating systems but provide VMM features to guest operating systems
 - Including VMware Workstation and Fusion, Parallels Desktop, and Oracle VirtualBox

- Other variations include:
 - **Paravirtualization** - Technique in which the guest operating system is modified to work in cooperation with the VMM to optimize performance
 - **Programming-environment virtualization** - VMMs do not virtualize real hardware but instead create an optimized virtual system
 - Used by Oracle Java and Microsoft.Net

- **Emulators** – Allow applications written for one hardware environment to run on a very different hardware environment, such as a different type of CPU
- **Application containment** - Not virtualization at all but rather provides virtualization-like features by segregating applications from the operating system, making them more secure, manageable
 - Including Oracle Solaris Zones, BSD Jails, and IBM AIX WPARs
- Much variation due to breadth, depth and importance of virtualization in modern computing

- 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, ...

- **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

- 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
- Several techniques, as described in next slides

- Methods of Virtualization:
 1. Emulation
 2. Trap and Emulate.
 3. Binary Translation

- Emulation is the process where the virtualizing software mimics that portion of hardware, which is provided to the guest operating system in the virtual machine.
- The presented emulated hardware is independent of the underlying physical hardware.
- The guest operating system interacts only with the emulated hardware.

- In an emulated environment, both the **application** and **guest operating system** in virtual machines run in the **user mode** of base operating system (the behavior of the hardware is produced by a software program)
- Emulation process involves only those hardware components so that user or virtual machines does not understand the underlying environment. **Only CPU & memory are sufficient for basic level of emulation.**

- Typically, emulation is implemented using interpretation.
- The emulator component takes **each and every instruction** of user mode and translates to equivalent instruction suitable according to the underlying hardware.
- Guest OS remains completely unaware of the virtualization.
- Every instruction issued by a VM is trapped in the VMM and interpreted for execution in the hardware.
- The disadvantage of performance, emulation using interpretation is hardly used in virtualization.

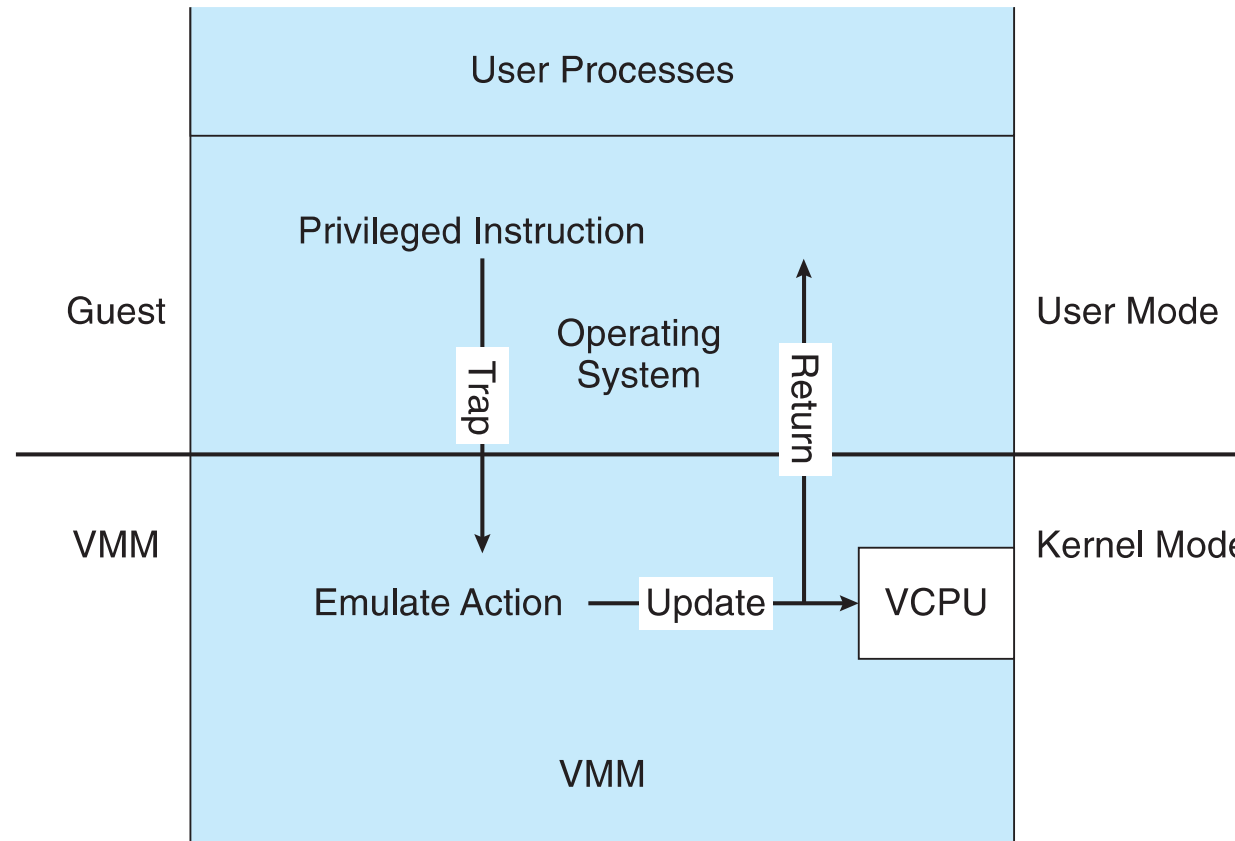
- Trap and emulate is a technique that takes the **basic of the emulation** but improves performance by using interpretation **selectively**.
- Both the user applications and **guest operating system** of virtual machines run in the user mode.
- Not safe to let **guest operating system** kernel run in 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

- How does switch from virtual user mode to virtual kernel mode occur?
 - Attempting a privileged instruction in user mode causes an error -> trap
 - VMM (Hypervisor) 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
- (Solution) CPUs adding hardware support, mode CPU modes to improve virtualization performance

Module 7

Trap and Emulate-Virtualization Implementation



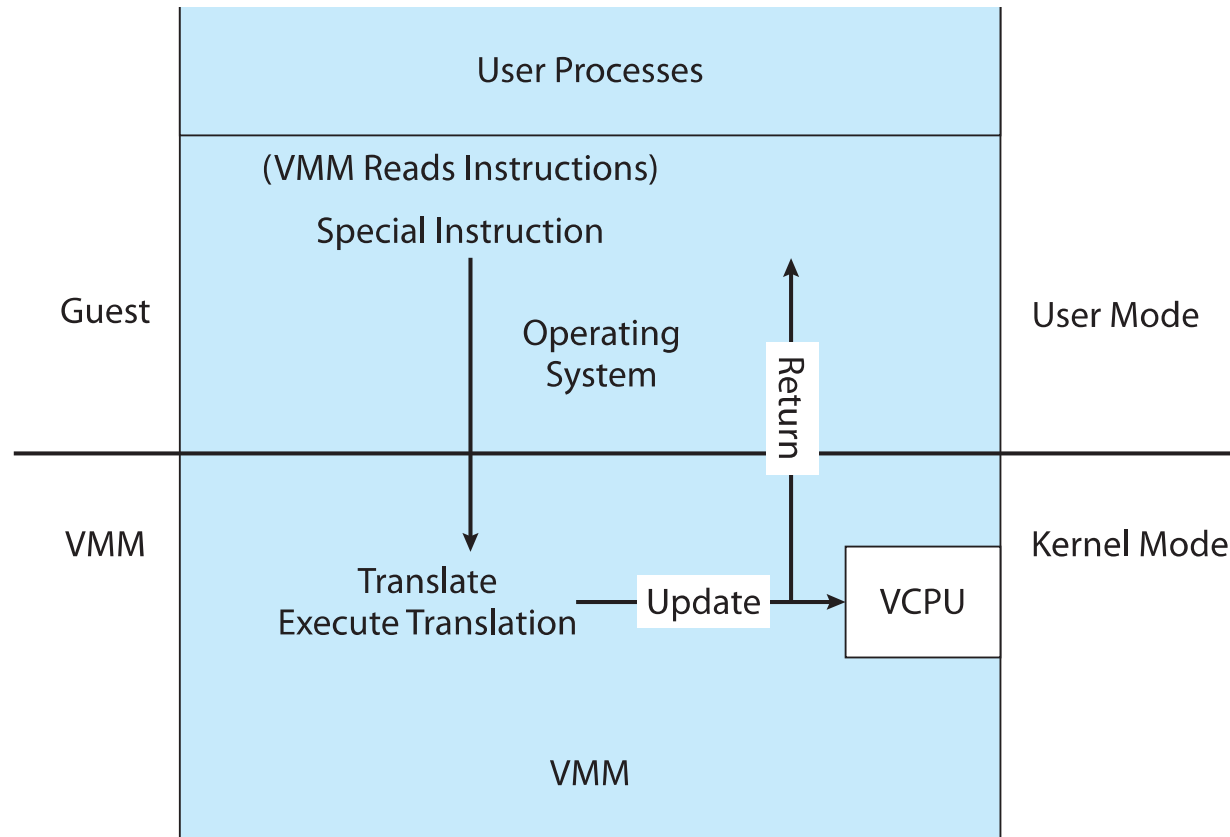
- Some CPUs don't have clean separation between privileged and nonprivileged instructions
 - Earlier Intel x86 CPUs are among them
 - Earliest Intel CPU designed for a calculator
 - Backward compatibility means difficult to improve
 - Consider Intel x86 **popf** instruction
 - Loads CPU flags register 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

- Other similar problem instructions we will call *special instructions*
 - trap-and-emulate will not solve this issue, So.....
- Binary translation solves the problem
 - Binary translation its Basics are simple, but implementation very complex
 - VMM examines every instruction guest is about to execute by reading a few instructions ahead of program counter
 - Non-special-instructions run natively
 - *Special instructions are* translated (**and does not wait for a trap!**). into new set of instructions that perform equivalent task.

- Translation of code is done within VMM
- 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

Module 7

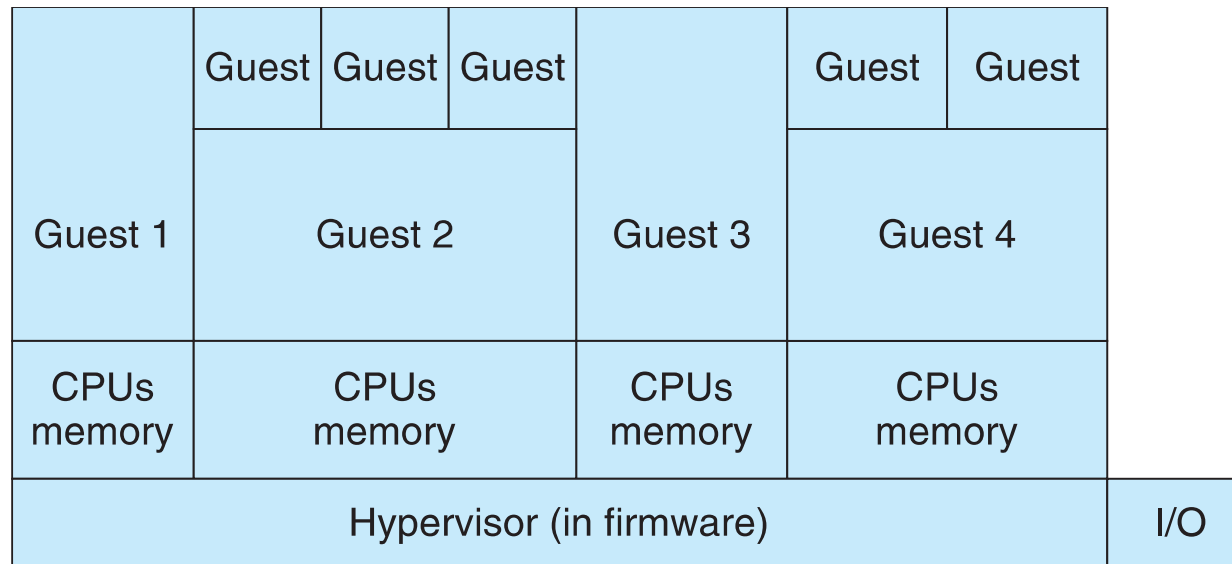
Binary Translation



- 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

- 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

- Old idea, under many names by HW manufacturers
 - “partitions”, “domains”
 - A HW features are implemented by firmware
 - OS need to do nothing special, VMM is in firmware
 - Smaller feature set than other types
 - Each guest has dedicated HW
- I/O is a challenge as difficult to have enough devices, controllers to dedicate to each guest
- Sometimes VMM implements a **control partition** running daemons (background service) that other guests communicate with for shared I/O
- Can provide virtualization-within-virtualization (guest itself can be a VMM with guests)
 - Other types have difficulty doing this



- Commonly found in company datacenters
 - In a sense becoming “datacenter operating systems”
 - Datacenter managers control and manage OSES in new, sophisticated ways by controlling the Type 1 hypervisor
 - Consolidation of multiple OSES and apps onto less HW
 - Move guests between systems to balance performance
 - Snapshots and cloning

- Special purpose operating systems that run natively on HW
 - Rather than providing system call interface, create run and manage guest OSes
 - Can run on Type 0 hypervisors but not on other Type 1s
 - Run in kernel mode
 - Guests generally don't know they are running in a VM
 - Also provide other traditional OS services like CPU and memory management

- Another variation is a general purpose OS that also provides VMM functionality
 - RedHat Enterprise Linux with KVM, Windows with Hyper-V, Oracle Solaris
 - Perform normal duties as well as VMM duties
 - Typically less feature rich than dedicated Type 1 hypervisors
- In many ways, treat guests OSes as just another process
 - Albeit with special handling when guest tries to execute special instructions

- Less interesting from an OS perspective
 - Very little OS involvement in virtualization
 - VMM is simply another process, run and managed by host
 - Even the host doesn't know they are a VMM running guests
 - Tend to have poorer overall performance because can't take advantage of some HW features
- But also a benefit because require no changes to host OS
 - Student could have Type 2 hypervisor on native host, run multiple guests, all on standard host OS such as Windows, Linux, MacOS

- Does not fit the definition of virtualization – VMM not presenting an exact duplication of underlying hardware
 - But still useful!
 - VMM provides services that guest must be modified to use
 - Leads to increased performance
 - Less needed as hardware support for VMs grows

- Xen, leader in paravirtualized space, adds several techniques
 - For example, clean and simple device abstractions
 - Good communication between guest and VMM about device I/O
 - Each device has circular buffer shared by guest and VMM via shared memory



- Xen, leader in paravirtualized space, adds several techniques (Cont.)
 - Memory management does not include nested page tables
 - Each guest has own read-only tables
 - Guest uses **hypercall** (call to hypervisor) when page-table changes needed
- Paravirtualization allowed virtualization of older x86 CPUs (and others) without binary translation
- Guest had to be modified to use run on paravirtualized VMM

- Also not-really-virtualization but using same techniques, providing similar features
- Programming language is designed to run within custom-built virtualized environment
 - For example Oracle Java has many features that depend on running in **Java Virtual Machine (JVM)**

- In this case 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 (including some smart phones even)
- Programs written in Java run in the JVM no matter the underlying system
- Similar to **interpreted languages**

- 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

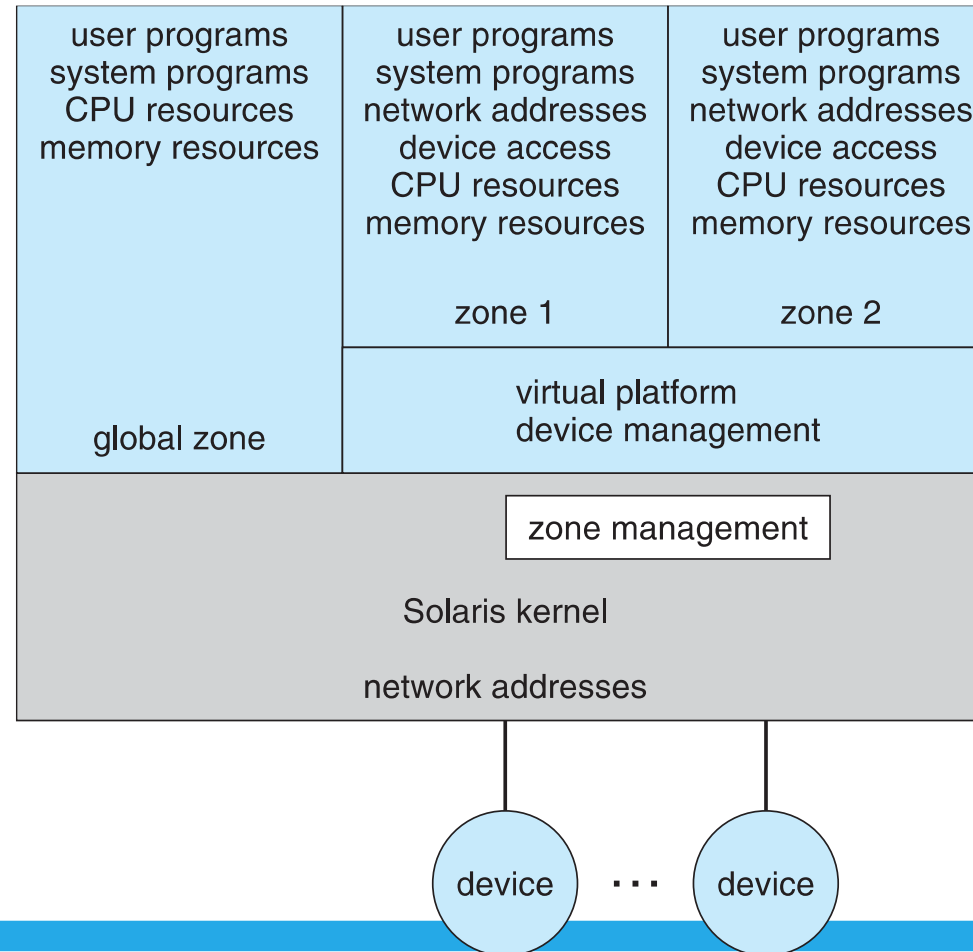
- Useful when host system has one architecture, guest compiled for other architecture
 - Company replacing outdated servers with new servers containing different CPU architecture, but still want to run old applications
- Performance challenge – order of magnitude slower than native code
 - New machines faster than older machines so can reduce slowdown
- Very popular – especially in gaming where old consoles emulated on new

Types of VMs – Application Containment

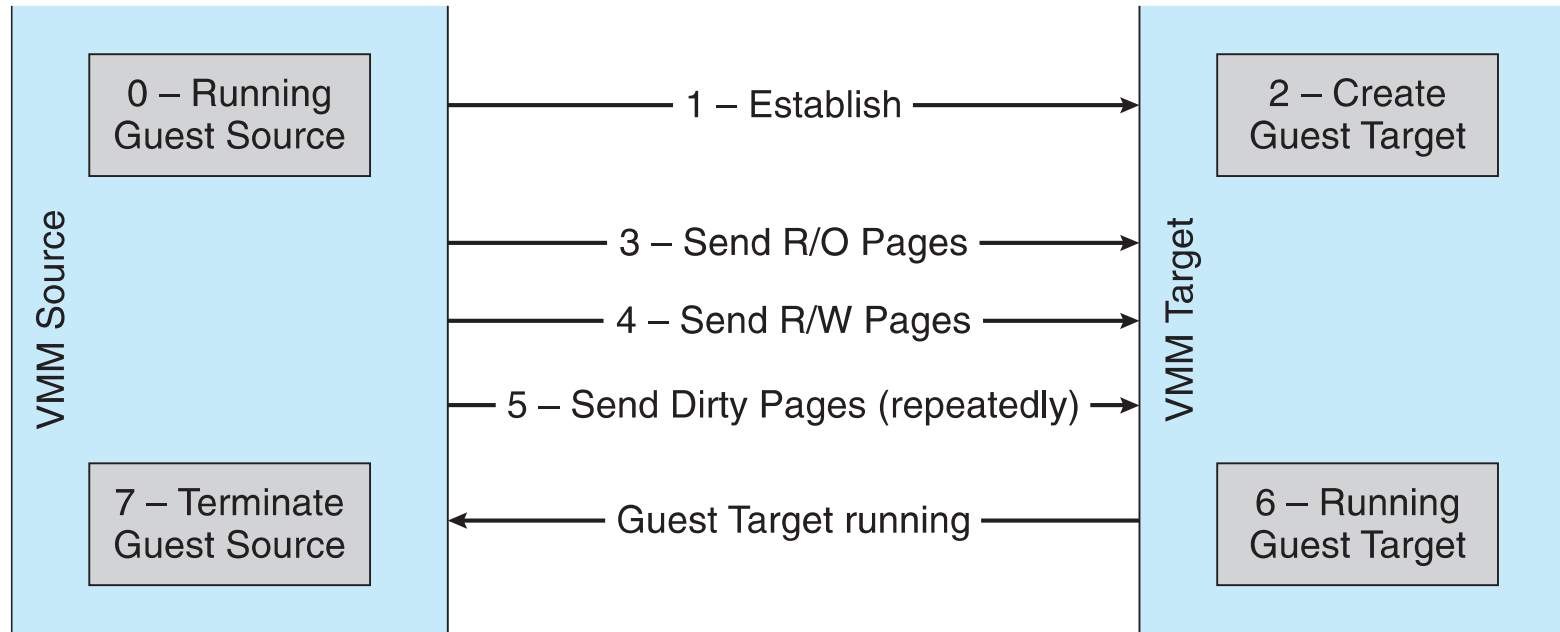
- Some goals of virtualization are segregation of apps, performance and resource management, easy start, stop, move, and management of them
- Can do those things without full-fledged virtualization
 - If applications compiled for the host operating system, don't need full virtualization to meet these 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

Module 7

Solaris 10 with Two Zones



- Taking advantage of VMM features leads to new functionality not found on general operating systems such as live migration
- Running guest can be moved between systems, without interrupting user access to the guest or its apps



End of Module 7

