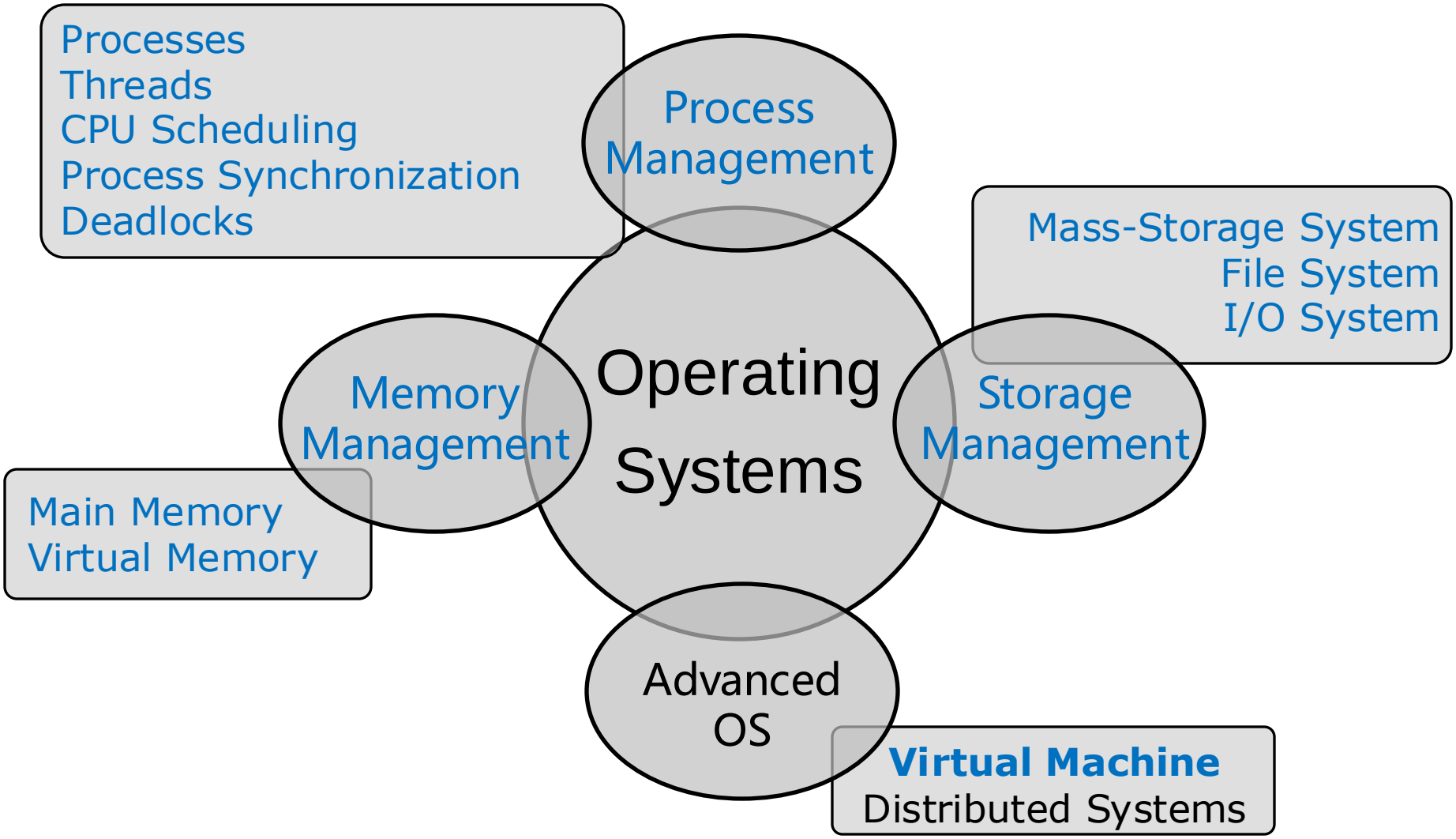
Virtual Machines

Shengzhong Liu

Department of Computer Science and Engineering

Shanghai Jiao Tong University

Operating System Topics

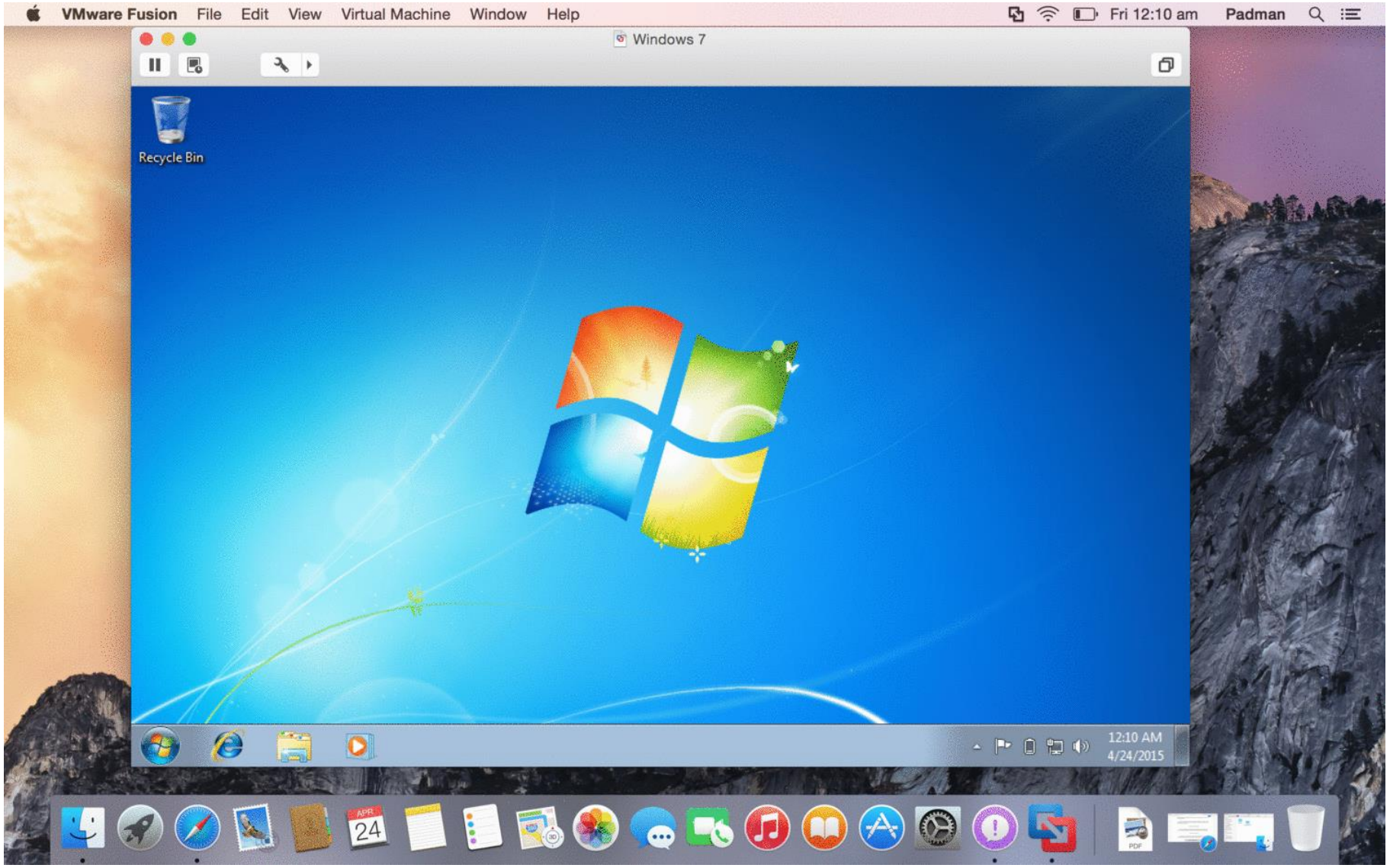


Outline

- Overview
- VM Benefits and Features
- VM Building Blocks
- VM Types and Implementations
- Virtualization and Operating-System Components

Overview

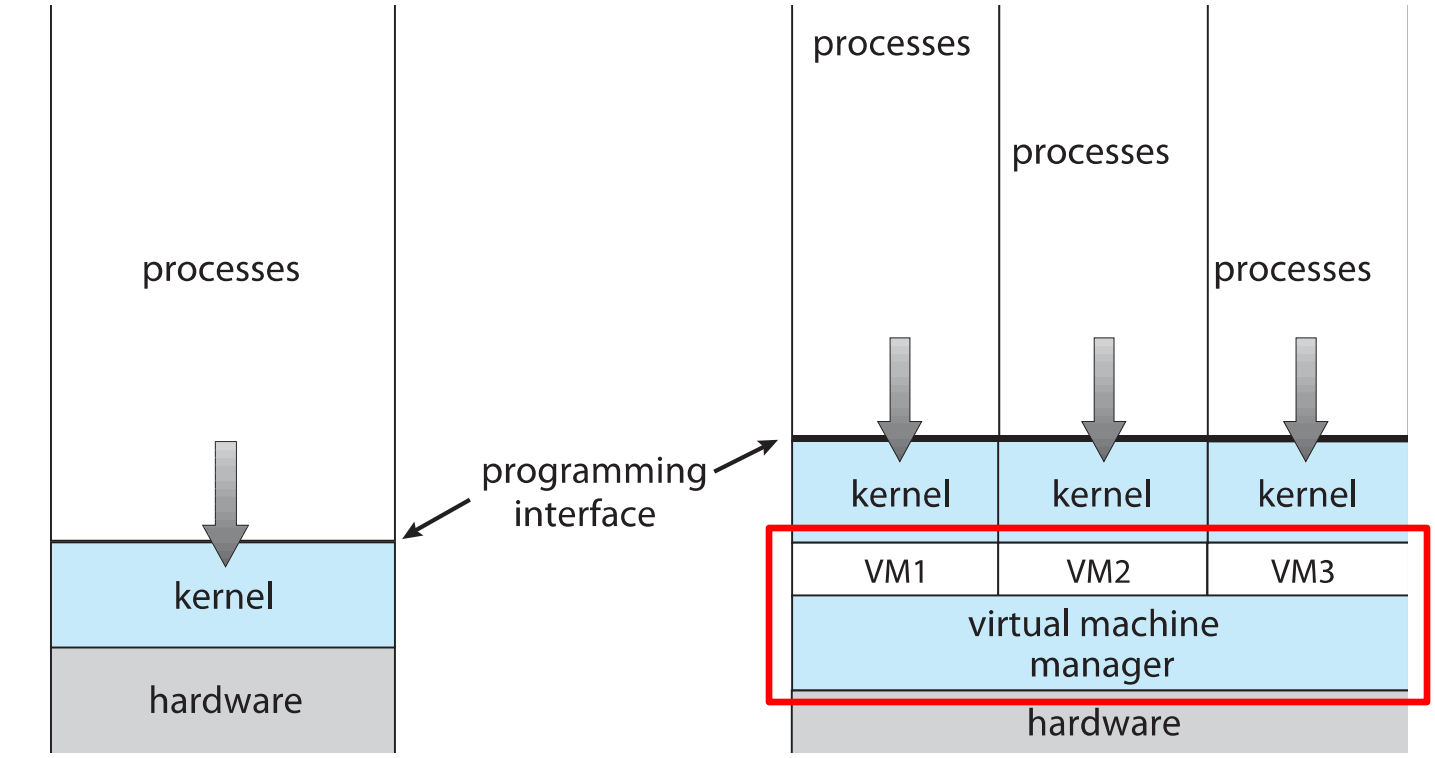
Overview


VMware Fusion File Edit View Virtual Machine Window Help
Fri 12:10 am Padman
Windows 7
Recycle Bin
12:10 AM
4/24/2015

Overview

- Fundamental idea – **abstract hardware of a single computer into several different execution environments**
  - Similar to the layered approach
  - Creates a virtual system (**virtual machine**, or **VM**) on which operating systems or applications can run
- Several components
  - **Host** – underlying hardware system where VM runs
  - **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 OSes concurrently, each in its own virtual machine

System Models



Non-virtual machine

Virtual machine

History

- First appeared in IBM commercial mainframes in 1972
  - **Allowed multiple users to share a batch-oriented system**
- The virtualization requirements called for:
  1. **Fidelity**: A VMM provides an environment for programs that is essentially identical to the original machine
  2. **Performance**: Programs running within that environment show only minor performance decreases
  3. **Safety**: The VMM is in complete control of system resources
- In late 1990s, Intel x86 CPUs fast enough for researchers to try virtualizing on general purpose PCs
  - **Xen** and **VMware** created technologies, which are still used today
  - Virtualization has expanded to many OSes, CPUs, VMMs

Benefits and Features of Virtualization

Benefits and Features

- Host system protected from VMs, and VMs protected from each other
  - i.e., a virus less likely to spread
  - Sharing is provided via shared file system volume, network communication
- Provide the ability to freeze (or suspend) a running VM
  - The VMs can move or copy somewhere else and resume
  - With snapshot of a given state, be able to restore back to that state
    - Some VMMs allow multiple snapshots per VM
  - **Clone**: creating copy and running both original and copy
- Great for OS research with better **system development efficiency**
  - Changing a normal OS is a difficult task
- Allow running multiple and 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

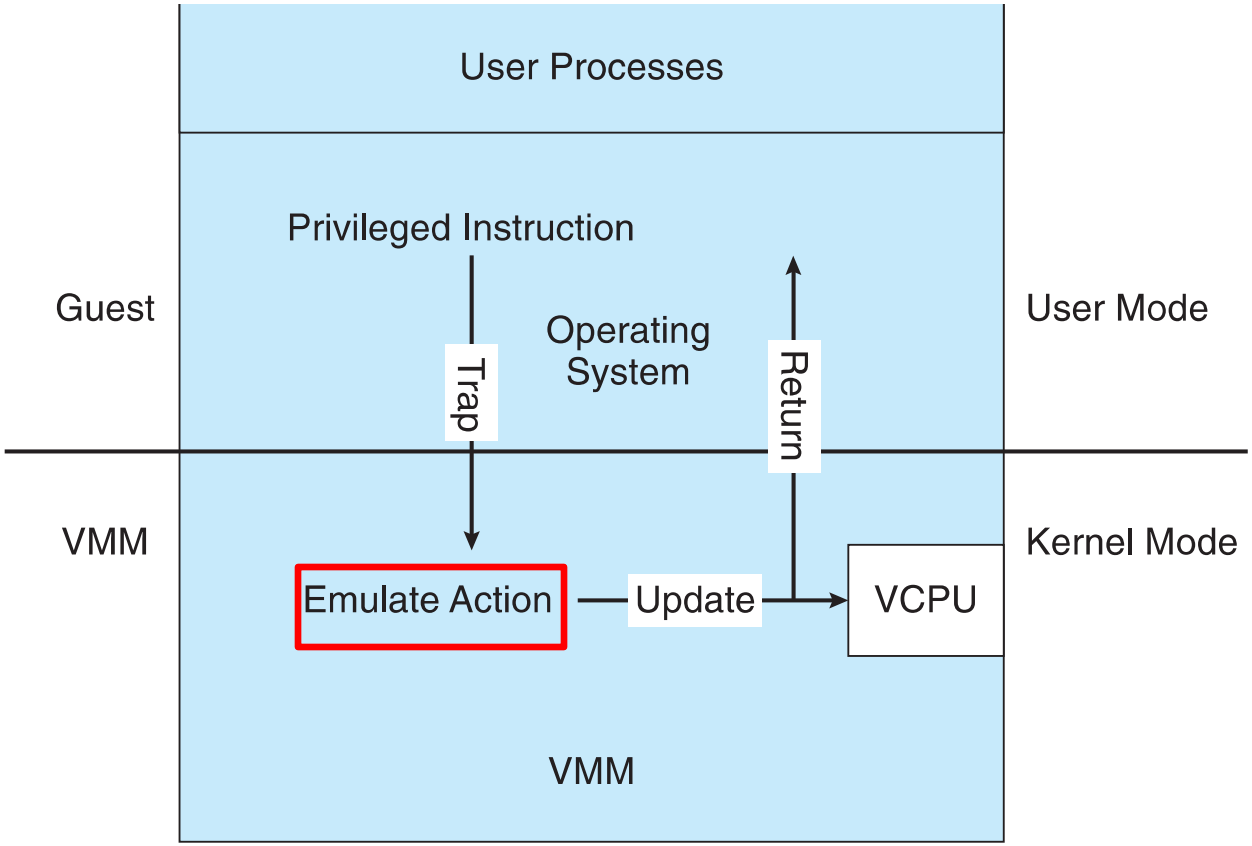
Building Blocks for CPU Management

- Generally difficult to provide an **exact** duplicate of underlying machine
  - 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
  - **Trap-and-emulate**
  - **Binary translation**

Building Block – (1) Trap and Emulate

- **Dual mode CPU**: guest executes in user mode, and kernel runs in kernel mode
  - Not safe to let guest kernel also run in kernel mode
- VM needs two modes – **virtual user mode** and **virtual kernel mode**
  - Both virtual modes run in the actual user mode
  - Guest OS actions that usually cause a switch to kernel mode must cause the switch to virtual kernel mode
- **Trap and emulate**:
  - When the kernel in the guest attempts to execute a privileged instruction, that is an error and causes a trap to the VMM in the real machine
  - The VMM gains control and emulates the action attempted by the guest kernel on the part of the guest
  - It then returns control to the virtual machine

Trap-and-Emulate Implementation



Building Block – (1) Trap-and-Emulate (Cont.)

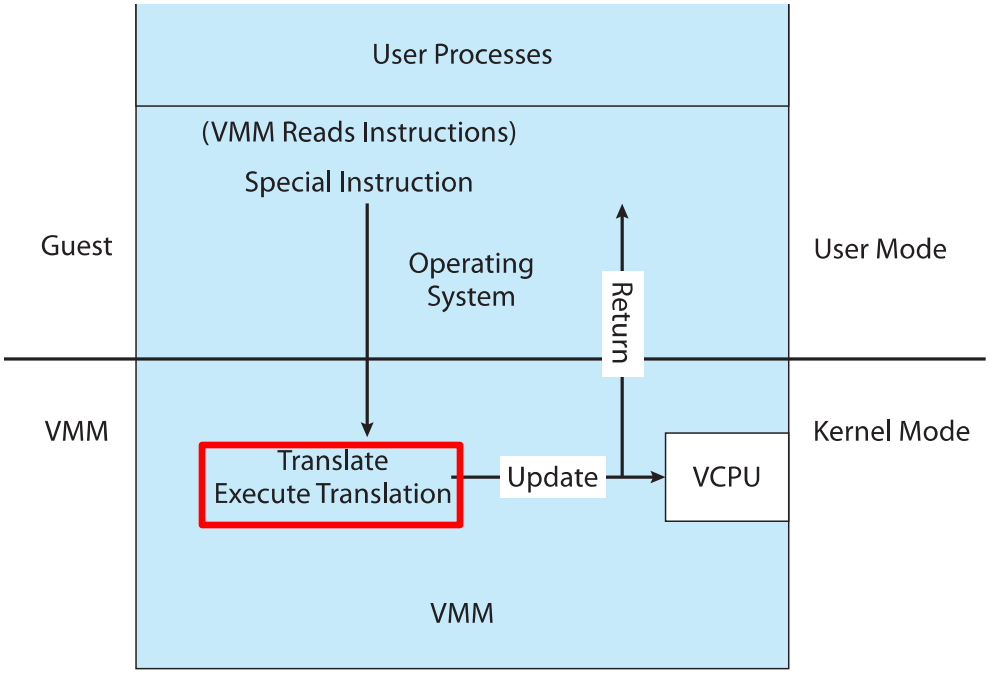
- User mode code runs natively on the hardware, providing the same performance for guests as native applications
- Kernel mode privilege code runs slower due to trap-and-emulate
  - Especially a problem when multiple guests running, each needing trap-and-emulate
- **Improved solution**: CPUs adding hardware support, and adding more CPU modes to improve virtualization performance

Building Block – (2) Binary Translation

- Some CPUs don't have clean separation between **privileged** and **nonprivileged** instructions
  - Earlier Intel x86 CPUs are among them
  - 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 -> only some flags replaced
    - No trap is generated for this instruction
- Other similar problem instructions are called **special instructions**
  - Caused trap-and-emulate method impossible until 1998

Building Block – (2) Binary Translation (Cont.)

- **Binary translation (二进制翻译)** solves the problem
  1. If guest VCPU is in **user mode**, the guest can run instructions natively
  2. If guest VCPU is in **kernel mode** (guest believes it is in kernel mode)
     a) VMM examines every instruction to execute by reading a few instructions in advance
     b) If instructions are **non-special**, then run natively
     c) If instructions are **special**, then they are translated into a new set of instructions that perform equivalent task
        a) e.g., changing the flags in the VCPU

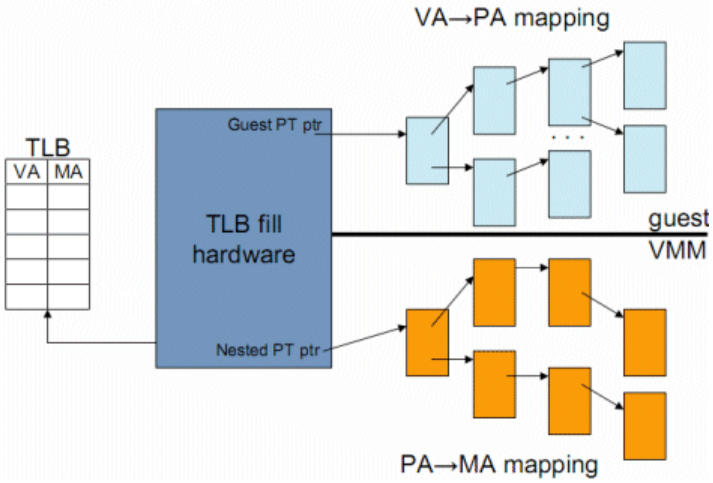


Building Block – (2) Binary Translation (Cont.)

- The basics are simple, but implementation is very **complex**
  - Implemented by **translation of code within VMM**
  - Code reads native instructions from the guest on demand, generates native binary code to execute in place of original code
- The performance of this method would be **poor without optimizations**
  - Products like VMware use caching
    - Translate once, and when the guest executes code containing special instruction, cached translation is used instead of translating again
    - Testing showed booting Windows XP as a guest caused 950,000 translations, at 3 microseconds each
      - 3 seconds (about 5 %) slowdown over native

Building Block – (3) Nested Page Tables

- Memory management is another general challenge to VMM implementations
  - How can both guests believe they control the page tables, which are actually controlled by VMM?
- Common method is **nested page tables (NPTs, 嵌套页表)**
  - Each guest maintains **page tables** to translate virtual to physical addresses
  - VMM maintains **per-guest NPTs** to represent guest's page-table state
  - Actions:
    - When a guest on CPU -> VMM makes guest's NPTs the active page tables
    - Guest changes page table -> VMM makes equivalent changes to NPTs and its own page tables
- Issue:
  - Can cause many more TLB misses -> much slower performance



Building Blocks – (4) Hardware Assistance

- All virtualization needs some hardware support
  - More support -> more feature rich, stable, better performance of guests
- Intel added new **VT-x instructions** in 2005 and AMD added **AMD-V instructions** in 2006
  - CPUs with these instructions **remove the need for binary translation**
- Generally, define more CPU modes – "**guest**" and "**host**"
  - In guest mode, guest OS thinks it is running natively
    - Access to virtualized resources cause trap to VMM
    - CPU maintains VCPU, context switches it as needed
- Hardware support for Nested Page Tables, DMA, interrupts as well over time

Virtual Machine Types

VM Types

- Many VM implementation variations as well as hardware details
  - VMMs take advantage of underlying hardware features
    - Hardware features can simplify implementation, improve performance
- Whatever the type, a VM has a lifecycle
  1. VM is created by VMM (hypervisor)
  2. Resources assigned to it
     - #cores, amount of memory, networking details, storage details
  3. In Type 0 hypervisor, resources are usually dedicated
     - Other types dedicate or share resources, or a mix
  4. When no longer needed, VM can be deleted, freeing resources
- Steps simpler, faster than with a physical machine install

VMM Type Overview

- 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** - OS-like software built to provide virtualization and general-purpose OSes providing standard functions and VMM functions
    - Softwares: VMware ESX, Joyent SmartOS, and Citrix XenServer
    - OS: **Microsoft Windows Server with HyperV** and **RedHat Linux with KVM**
  - **Type 2 hypervisors** - Applications that run on standard operating systems to provide VMM features to guest operating systems
    - VMware Workstation and Fusion
    - Parallels Desktop
    - Oracle VirtualBox

VMM Type Overview

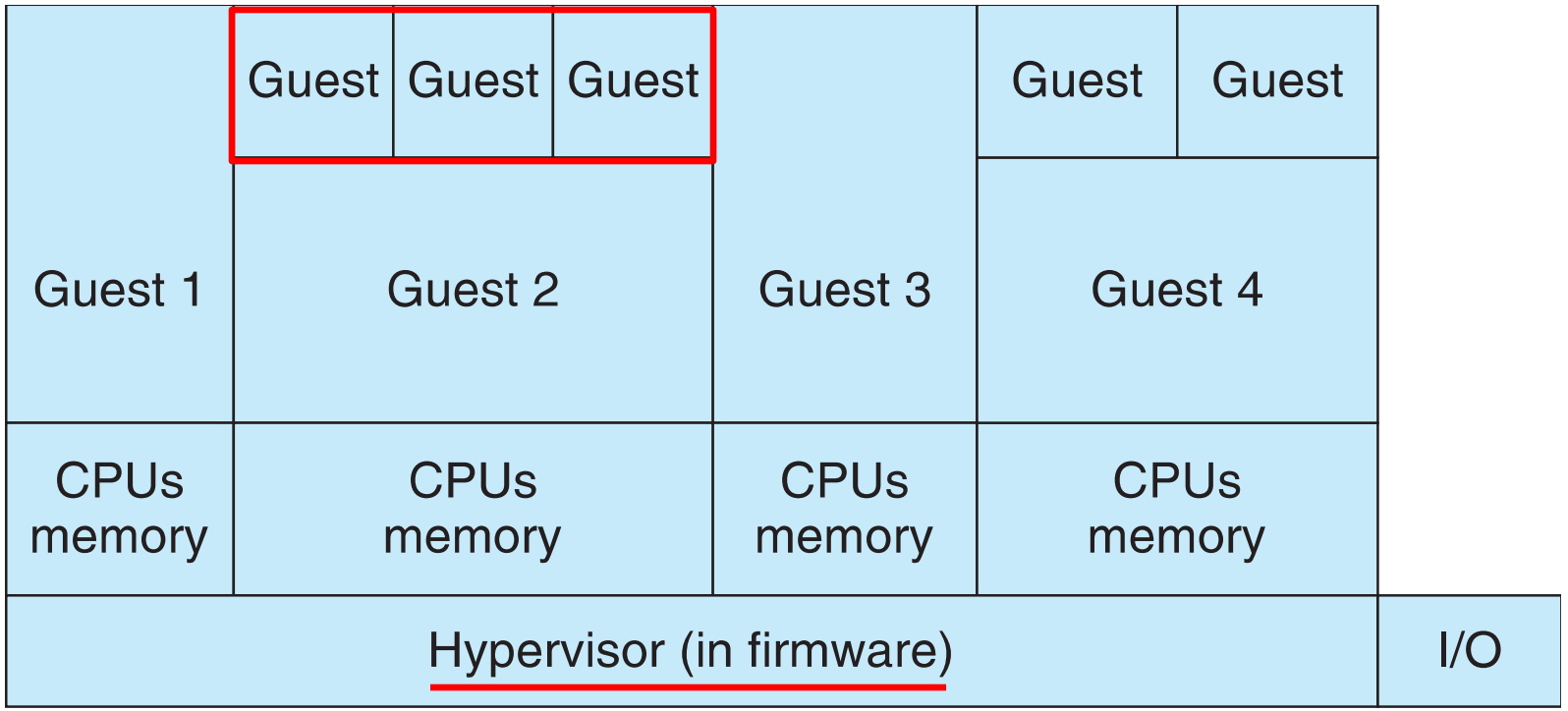
- Other variations include:
  - **Paravirtualization (半虚拟化)** – The guest OS is modified to work in cooperation with the VMM to optimize performance
  - **Programming-environment virtualization** - VMMs do not virtualize real hardware but instead create an optimized virtual system
    - Used by Oracle Java and Microsoft.Net
  - **Emulators （仿真器）** – Allow applications written for one hardware environment to run on a different hardware environment
    - e.g., a different type of CPU
  - **Application containment (容器)** - Not virtualization but provides virtualization-like features by segregating applications from the OS, making them more secure and manageable
    - Including Oracle Solaris Zones, BSD Jails, and IBM AIX WPARs

VMM Type: (1) Type 0 Hypervisor

- Type 0 hypervisors is **a hardware feature implemented by firmware**
  - OS needs to do nothing special because VMM is in the firmware
  - Smaller feature set than other types
  - Each guest has dedicated hardware
- I/O is a challenge as difficult to have enough devices and controllers to dedicate to each guest
  - Sometimes VMM implements a **control partition** running daemons
  - Other guests communicate with control partition for shared I/O
- Can provide **virtualization-within-virtualization**
  - Guest itself can be a VMM with guests
  - Other types have difficulty doing this

VMM Type: (1) Type 0 Hypervisor

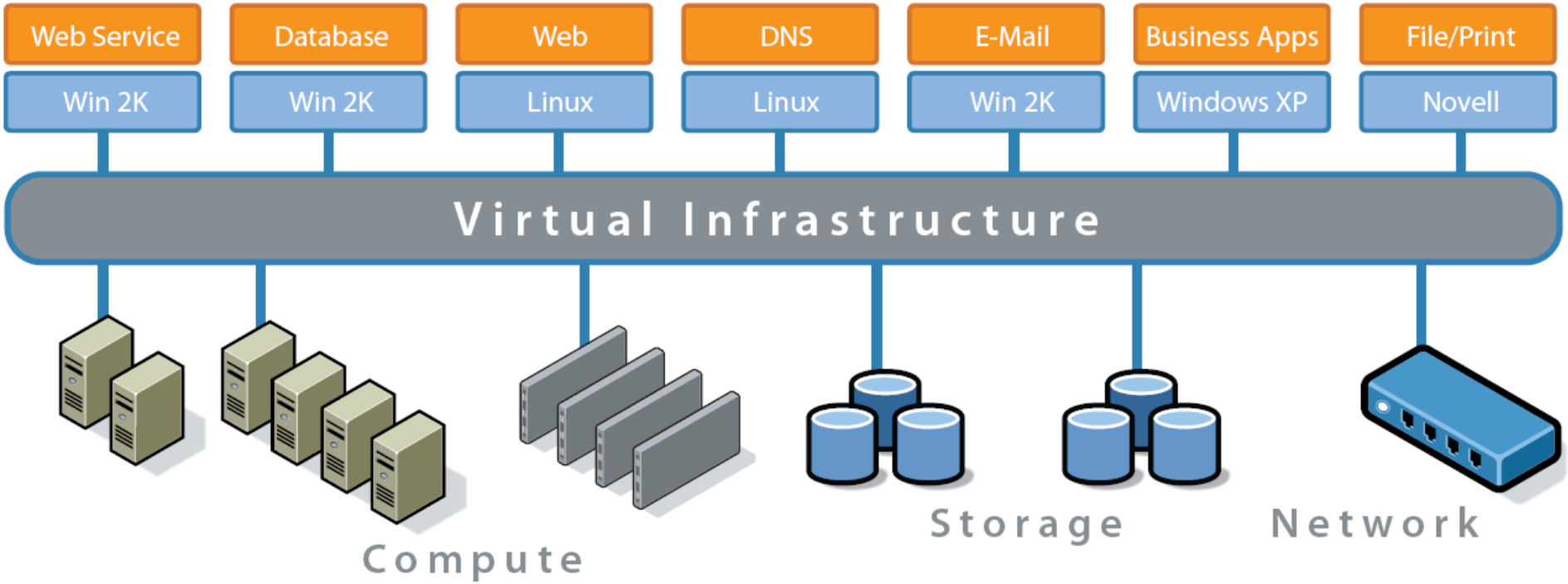
Virtualization within virtualization



VMM Type: (2) Type 1 Hypervisor

- Commonly found in company datacenters
  - In a sense becoming "**datacenter operating systems**"
    - Datacenter managers control and manage OSes in new and sophisticated ways by controlling the Type 1 hypervisor
    - Consolidation of multiple OSes and apps onto less hardware
    - Move guests between systems to balance performance
- Type 1 hypervisors are **special-purpose OS running in the kernel mode**
  - Rather than providing a system call, they create, run, and manage guest OS
    - They can run on Type 0 hypervisors but not on other Type 1 hypervisors
  - Effect:
    - Guests generally don't know they are running in a VM
  - Provide other traditional OS services like
    - CPU scheduling and
    - Memory management

Virtual Infrastructure for Data Center



VMM Type: (2) Type 1 Hypervisor (Cont.)

- Another variation of Type 1 hypervisor is a **general-purpose OS that also provides VMM functionality**
  - e.g., RedHat Enterprise Linux with KVM, Windows with Hyper-V, Oracle Solaris
  - Perform normal duties as well as VMM duties
  - Typically less features than dedicated Type 1 hypervisors
- In many ways, **treat guest OS as just another process**
  - Special handling when a guest tries to execute special instructions

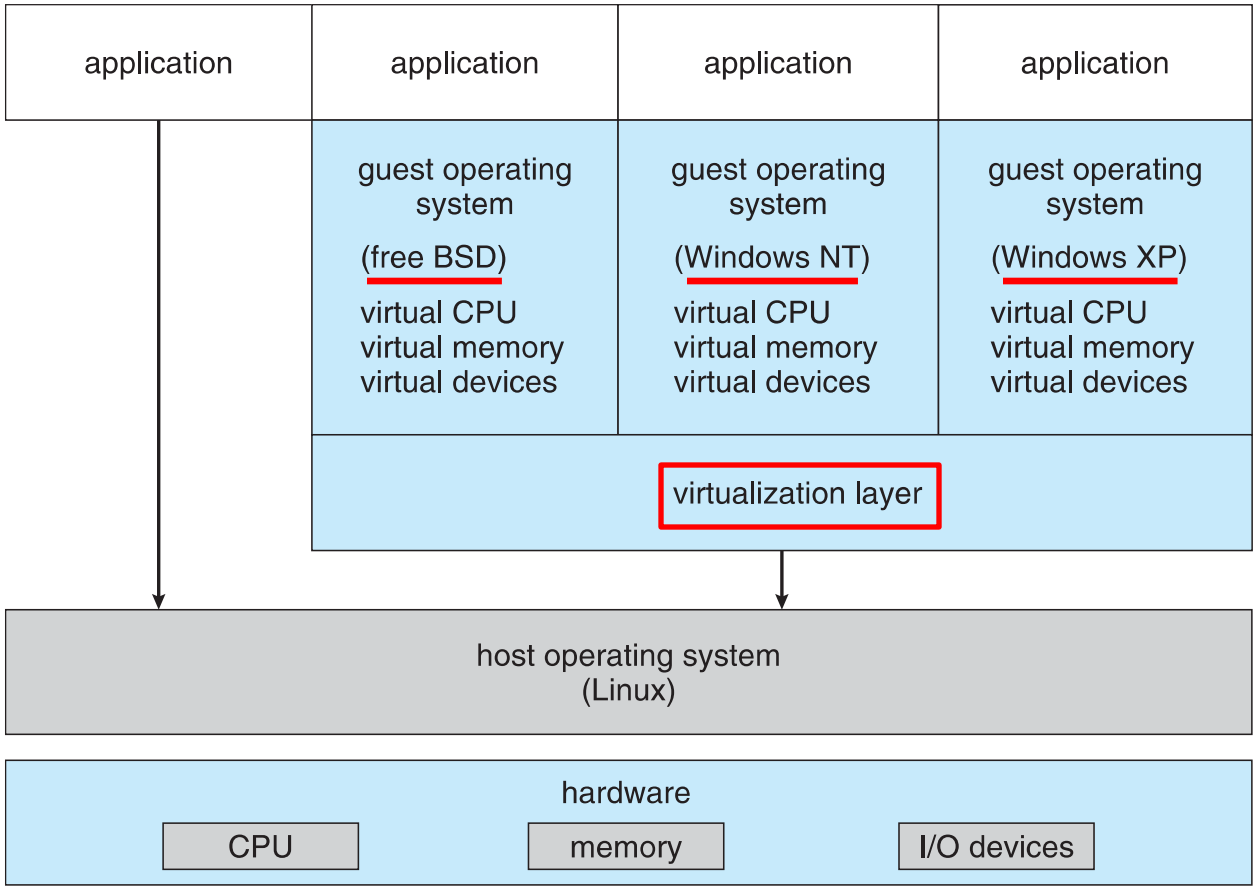
VMM Type: (3) Type 2 Hypervisor

- Type 2 hypervisors are applications that run on standard OS to provide VMM features
- Very little OS involvement in virtualization
  - VMM is simply **another process run and managed by the host**
  - Even the host doesn't know they are VMM running guests
- Comparison
  - Cons:
    - Tend to have **poorer overall performance** because they can't take advantage of some hardware features
  - Pros:
    - Benefit in **ease of use** without changes to host OS
    - You could have Type 2 hypervisor on native host, run multiple guests, all on standard host OS such as Windows, Linux, MacOS

Examples - VMware

- **VMware Workstation** runs on x86, provides VMM for guests
- Lots of guests possible, including Windows, Linux, etc.
  - If resources allow, all runnable concurrently
- The virtualization layer abstracts underlying hardware
  - Providing guest with its own virtual CPUs, memory, disk drives, network interfaces, etc.
  - Physical disks can be provided to guests, or as virtual physical disks
    - Files within host file system

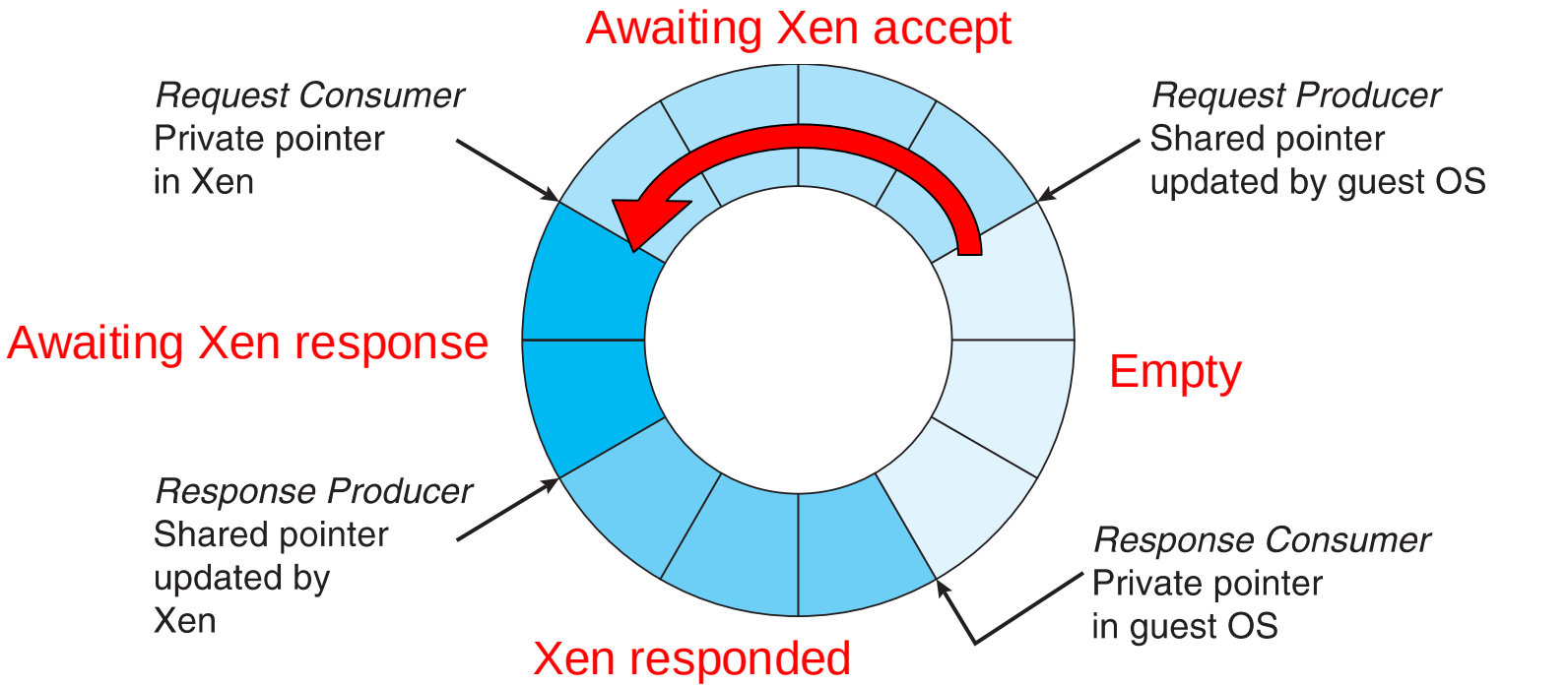
VMware Workstation Architecture



VMM Type: (4) Paravirtualization（半虚拟化）

- Does not fit the definition of virtualization – VMM not presenting an exact duplication of underlying hardware
  - VMM provides services that **guest must be modified to use**
  - Leads to increased performance
  - Less needed as hardware support for VMs grows
- **Xen**, the leader in paravirtualized space, adds several techniques
  - Clean and simple device abstractions
    - Efficient I/O
    - Good communication between guest and VMM on device I/O
    - Each device has circular buffer shared by guest and VMM via shared memory
- Pros and Cons:
  - Allowed virtualization of older x86 CPUs without binary translation
  - **Guest had to be modified** to use run on the paravirtualized VMM
    - Modern CPUs Xen no longer requires guest modification

Xen I/O via Shared Circular Buffer



Request queue - Descriptors queued by the VM but not yet accepted by Xen

Outstanding descriptors - Descriptor slots awaiting a response from Xen

Response queue - Descriptors returned by Xen in response to serviced requests

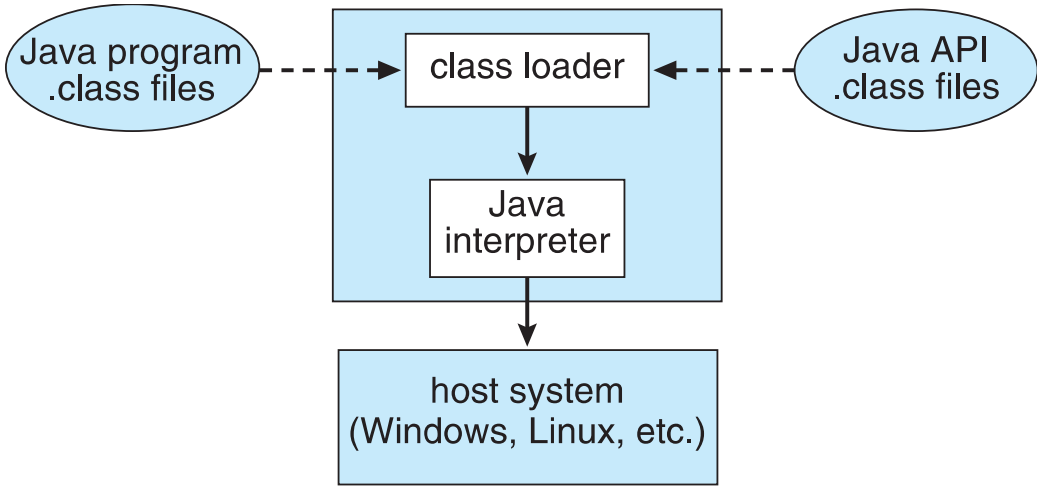
Unused descriptors

VMM Type: (5) Programming Environment Virtualization

- Also not-really-virtualization but uses same techniques with similar features
- Programming language is designed to run within virtualized environment
  - e.g., Java features depending on running in **Java Virtual Machine (JVM)**
- Virtualization **provides APIs with features available to a language**
  - Provide an improved execution environment
- JVM is compiled to run on many systems (including some smartphones)
  - Programs written in Java run in the JVM no matter the underlying system
  - Similar to **interpreted languages** like Python

Example – Java Virtual Machine

- Very popular application environment invented by Sun Microsystems in 1995
  - Write once, run anywhere
  - Includes language specification (Java), API library, Java virtual machine (JVM)
- **JVM compiled per architecture**, reads bytecode and executes
- Optimizations:
  - Includes **garbage collection** to reclaim memory no longer in use
  - Made faster by **just-in-time (JIT)** compiler that turns bytecodes into native code and caches them



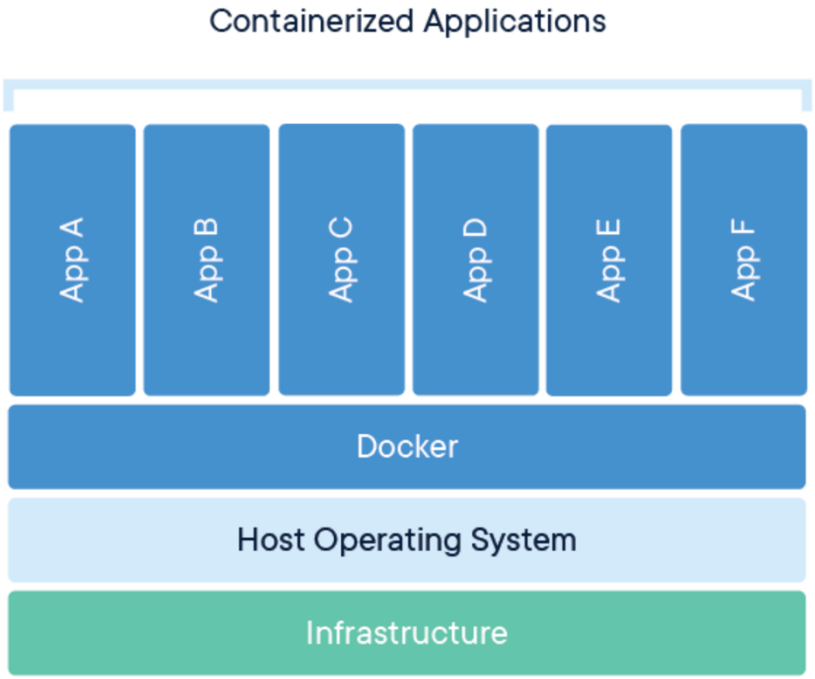
VMM Type: (6) Emulation（仿真）

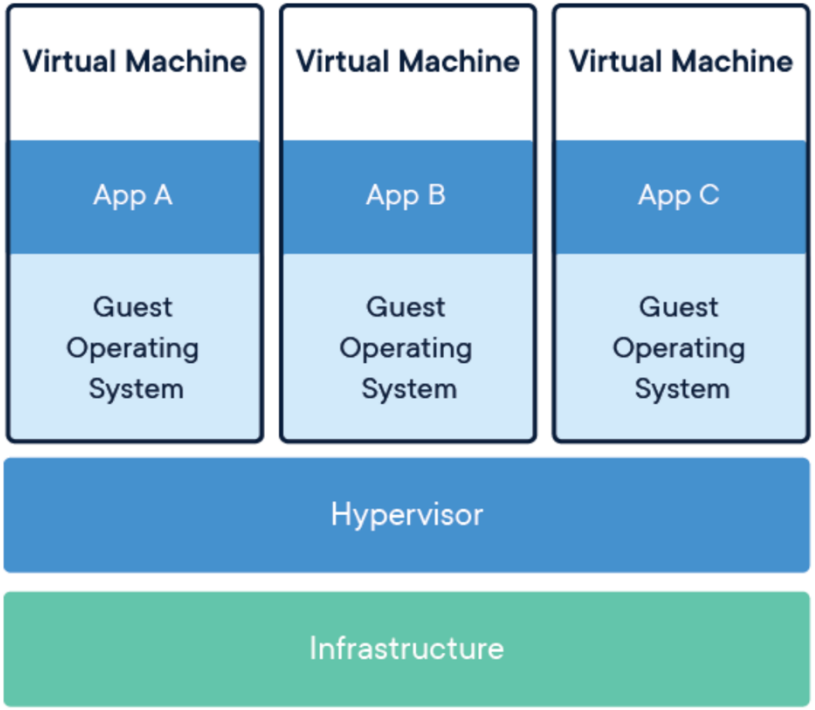
- Another way for running one operating system on a different OS
  - **Virtualization** requires underlying CPU to be same as guest was compiled for
  - **Emulation** allows guest to run on different CPU
- Need to **translate guest instructions from guest CPU to native CPU**
  - Emulation, not virtualization
- Useful when host system has one architecture, guest compiled for other architecture
  - Example:
    - Company replacing outdated servers with new servers of different CPUs
    - Still want to run old applications
  - Very popular – especially in gaming where old consoles emulated on new
- Performance challenge – **order of magnitude slower** than native code

VMM Type: (7) Application Containment (容器)

- Some goals of virtualization are
  - Segregation (隔离) 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 OS, don't need full virtualization to meet these goals
- Oracle **containers / zones** create virtual layer between OS and apps
  - Only one kernel running – host OS
  - OS and devices are virtualized, providing resources 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

Container vs. Virtual Machine

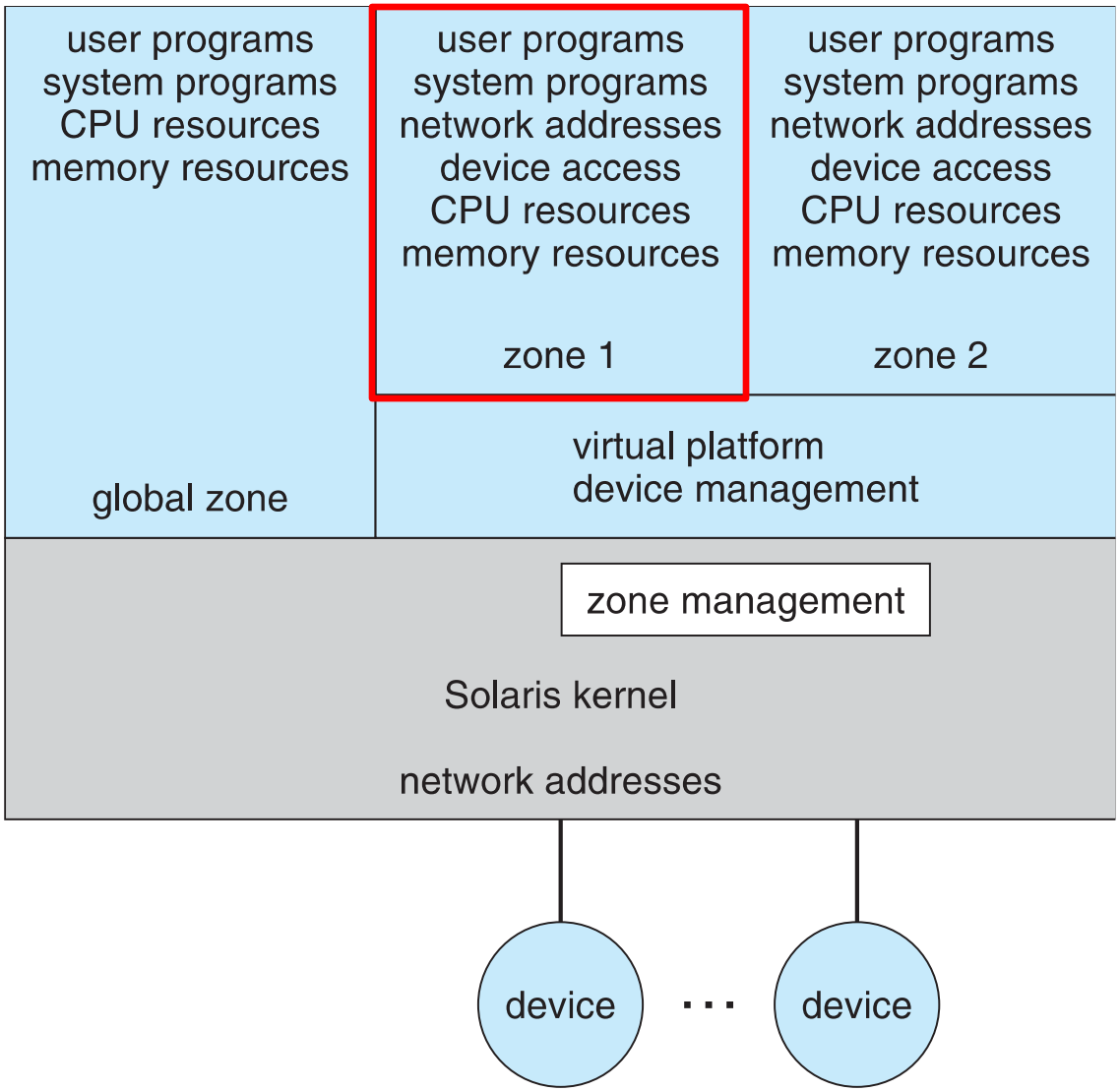




Containers are an abstraction at the app layer that packages code and dependencies together.

Virtual machines (VMs) are an abstraction of physical hardware turning one server into many servers.

Solaris 10 with Two Zones



Virtualization and OS Components

Virtualization and OS Components

- Now look at operating system aspects of virtualization
  - CPU scheduling, memory management, I/O, storage, and unique VM migration feature

- **Key questions**:
  - 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 **physical CPUs** and **multiple threads** to run on them
  - Guests configured with a certain number of VCPUs
    - Can be adjusted throughout the life of the VM
  - When enough CPUs for all guests -> VMM allocated dedicated CPUs
    - Each guest like a native OS managing its CPUs
  - When not enough CPUs -> CPU **overcommitment**
    - VMM can use standard scheduling algorithms to put threads on CPUs
    - Some add a fairness aspect to scheduling

OS Component – Memory Management

- Memory also suffers from **oversubscription** -> requires extra management efficiency from VMM
  - The total memory allocated to guests exceeds the amount that physically exists in the system

- E.g., VMware ESX guests have configured physical memory, then VMM uses three methods to **reclaim memory from guest**:
  1. **Double-paging:** the guest page table indicates a page is in a physical frame, but the VMM moves some of those pages to backing store
  2. **Install a pseudo-device driver** in each guest
     - It looks like a device driver to the guest kernel but actually adds kernel-mode code to the guest
     - Balloon memory manager communicates with VMM and is told to decrease or increase physical memory use of guest, causing guest OS to free or have more memory available
  3. **De-duplication by VMM** determining if same page loaded more than once, memory mapping the same page into multiple guests

OS Component – I/O Management

- VMM can provide virtualized devices to guest OS
  - Already somewhat flexible via device drivers
- Overall I/O is complicated for VMMs
  - Many short paths for I/O in standard OS for improved performance
  - The less hypervisor needs to do for I/O for guests, the faster the I/O is
  - **Examples**: direct device access, DMA pass-through, direct interrupt delivery
- Complex networking: VMM and guests all need network access
  - VMM can
    - **Bridge** guest to network (allowing direct access)
    - 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

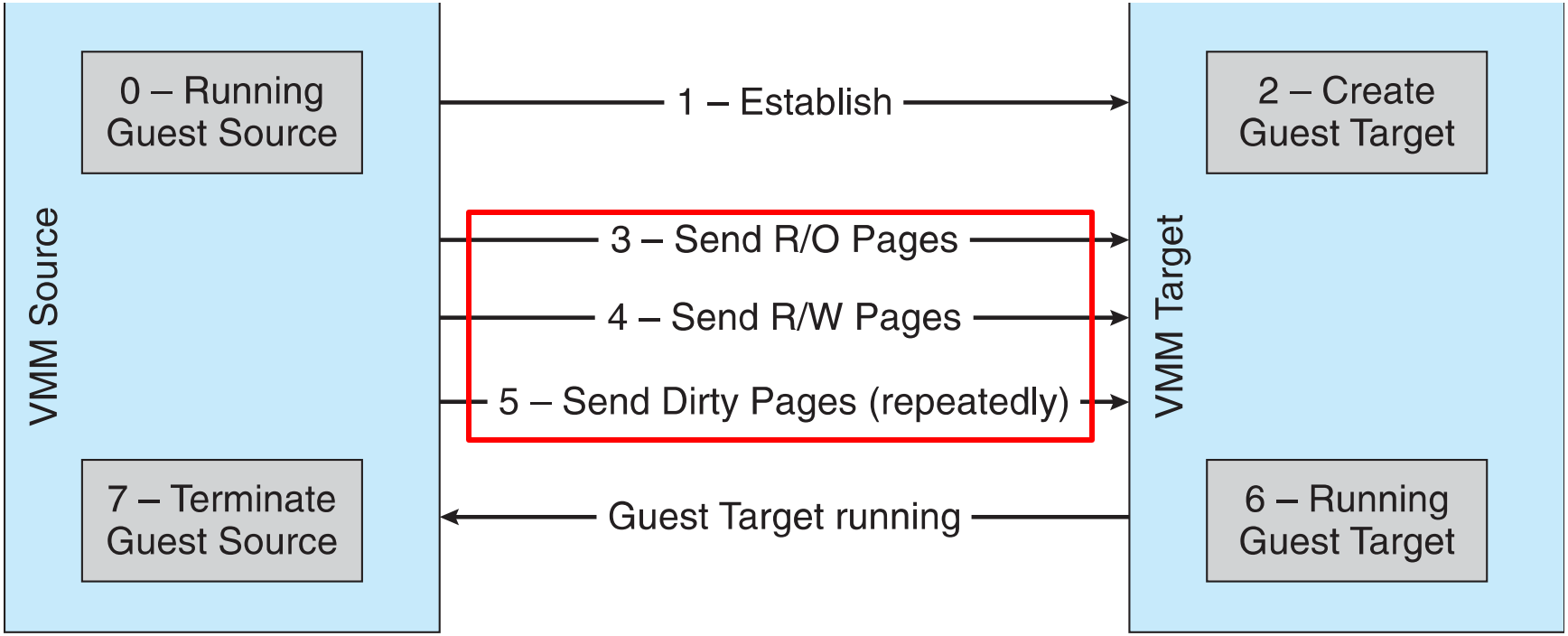
- Both **boot disk** and **general data access** need to be provided by VMM
  - Need to support potentially dozens of guests per VMM
  - Standard disk partitioning is not sufficient
- How to store **guest root disks** and **configuration information**:
  - Type 1 – store as files within the file system **provided by VMM**
  - Type 2 – store as files within the file system **provided by the host OS**
- Actions:
  - Duplicate file -> create new guest
  - Move the file to another system -> move guest
- Conversion:
  - **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

OS Component – Live Migration

- Running guest can be moved between systems, without interrupting user access to the guest or its apps

- **Live migration steps**:
  1. The **source VMM** establishes a connection with the **target VMM**
  2. The target creates a new guest by creating a new VCPU, etc.
  3. The source sends all **read-only guest memory pages** to the target
  4. The source sends all **read-write pages** to the target and mark them as clean
  5. The source **repeats** step 4
     - During the step, some pages were modified by the guest and are now dirty
  6. When the cycle of steps 4 and 5 becomes short, source VMM freezes the guest, sends VCPU's final state, other state details, and final dirty pages, and tells target to start running the guest
     - Once the target acks that the guest running, the source terminates guest

Live Migration of Guest Between Servers



Summary

- Virtualization is a method for providing a guest with a duplicate of a system's underlying hardware
- Multiple guests can run on a given system, each believing that it is the native operating system and is in full control
- The VMM or hypervisor, creates and runs the virtual machine
  - **Type 0 hypervisors** are implemented in the hardware with modifications to the OS
  - **Type 1 hypervisors** provide the environment and features needed to create, run, and manage guest virtual machines
  - **Type 2 hypervisors** are simply applications that run on other OS, which do not know that virtualization is taking place
- VMMs take advantage of available hardware support when optimizing CPU scheduling, memory management, and I/O modules to provide guests with optimal resource use and isolation
- Current research is extending the uses of virtualization

Homework

- Reading
  - Chapter 18

- HW4 due on **April 9, 23:59**!