CHAPTER 16 - VIRTUAL MACHINES

OBJECTIVES

- Explore 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.
- Containerization

OVERVIEW

OVERVIEW

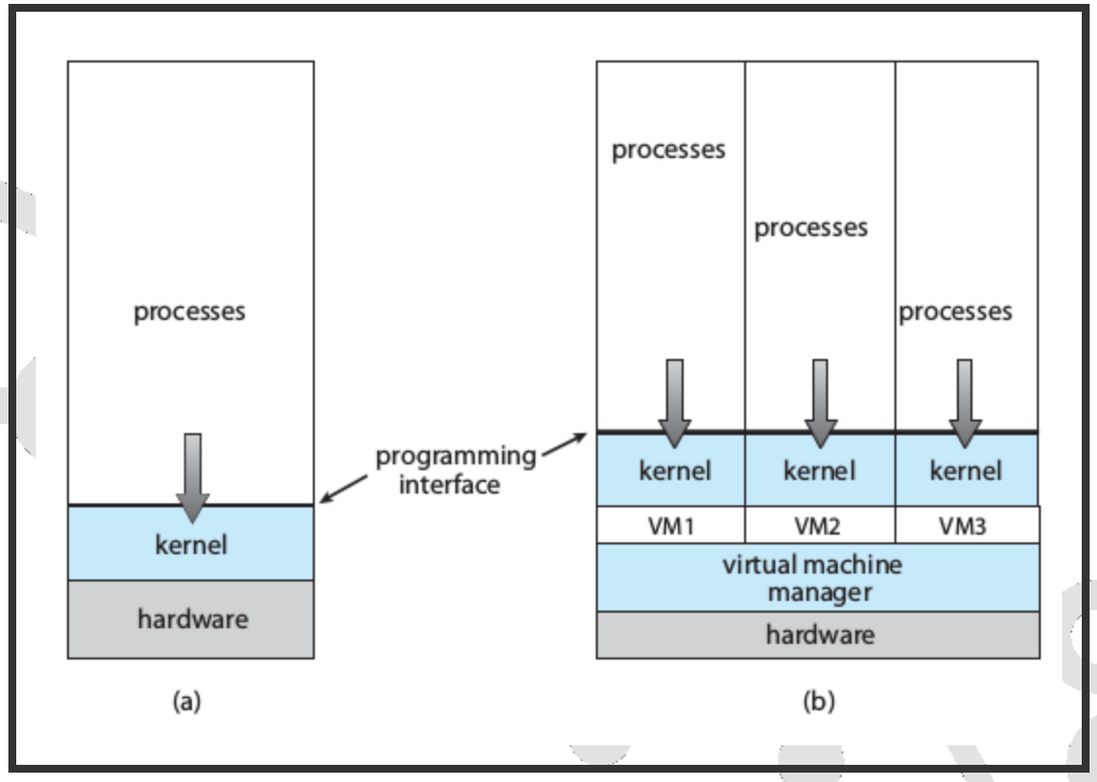
- Fundamental idea – abstract hardware of a single computer into several different execution environments

- Similar to layered approach
- But layer creates virtual system (**virtual machine**, or **VM**) on which operation systems or applications can run

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

SYSTEM MODELS



IMPLEMENTATION OF VMMS

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

IMPLEMENTATION OF VMMS

- **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

IMPLEMENTATION OF VMMS

- **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

- **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

OTHER VARIATIONS

- **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

HISTORY

HISTORY

- First appeared in IBM mainframes in 1972
- Allowed multiple users to share a batch-oriented system
- IBM VM 370 divided a mainframe into multiple virtual machines, each running its own OS

HISTORY

- 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

HISTORY

- 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
  - Even more important with containerization

BENEFITS AND FEATURES

BENEFITS AND FEATURES

- fundamentally related to the ability to share the same hardware yet run several different execution environments (i.e. different OS's) concurrently.

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

BENEFITS AND FEATURES

- 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

BENEFITS AND FEATURES

- **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

- 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

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

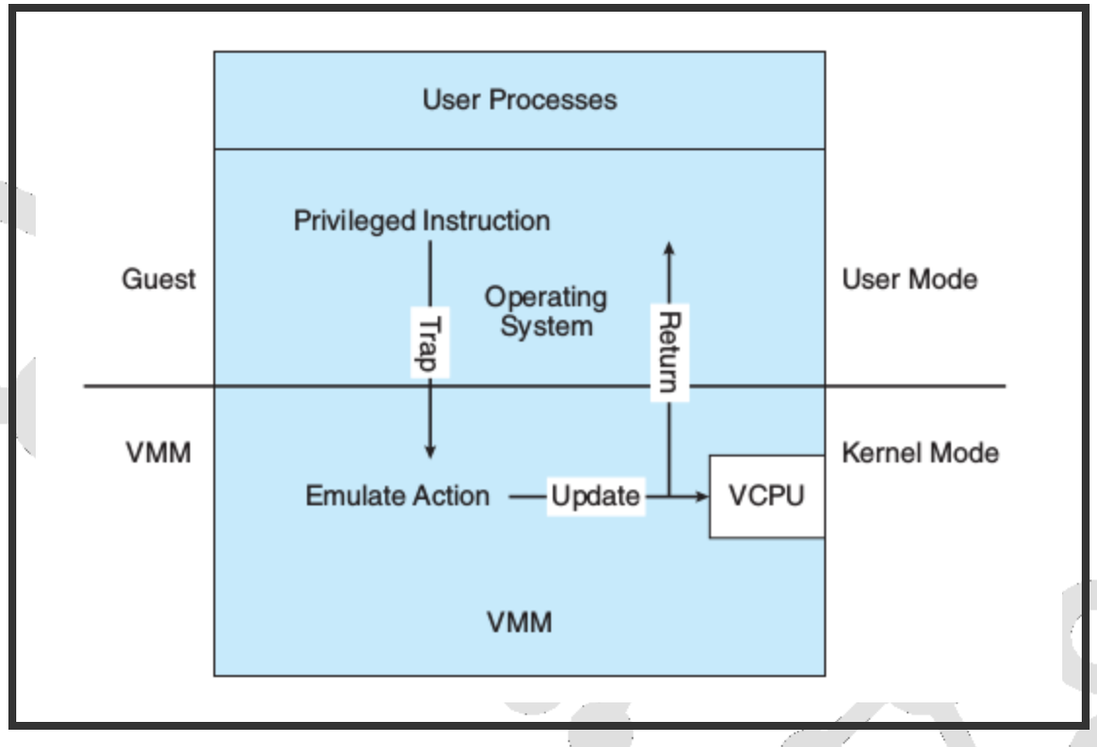
How does switch from virtual user mode to virtual kernel mode occur?

- 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

TRAP AND EMULATE

- 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



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

BINARY TRANSLATION

- 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 in user mode

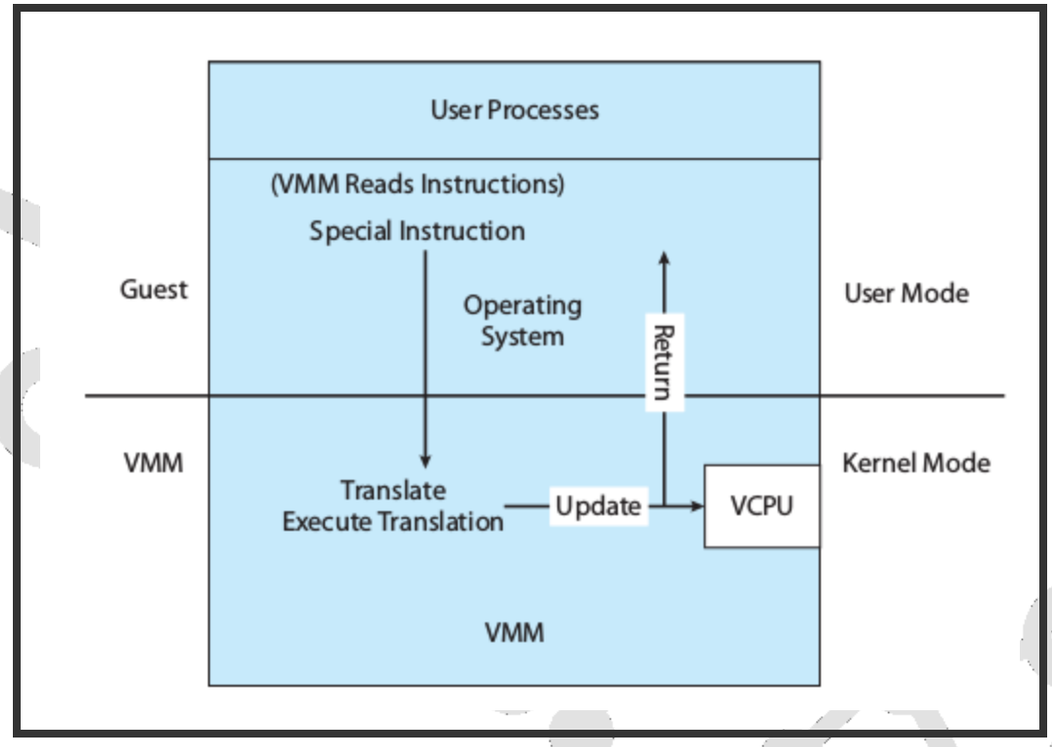
BINARY TRANSLATION

- 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

BINARY TRANSLATION

- If guest VCPU in user mode → run instructions natively
- If guest VCPU in kernel mode (believes is in kernel mode)
- 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 translated into new set of instructions that perform equivalent task
  - for example changing the flags in the VCPU

BINARY TRANSLATION VIRTUALIZATION IMPLEMENTATION



BINARY TRANSLATION

- 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

BINARY TRANSLATION

- Products like VMware use **caching**
- Translate once, and when guest executes code containing special instruction cached translation used instead of translating again
- Testing showed booting Windows XP as guest caused 950,000 translations, at 3 microseconds each, or 3 second (5%) slowdown over native

NESTED PAGE TABLES

Memory management another general challenge to VMM implementations

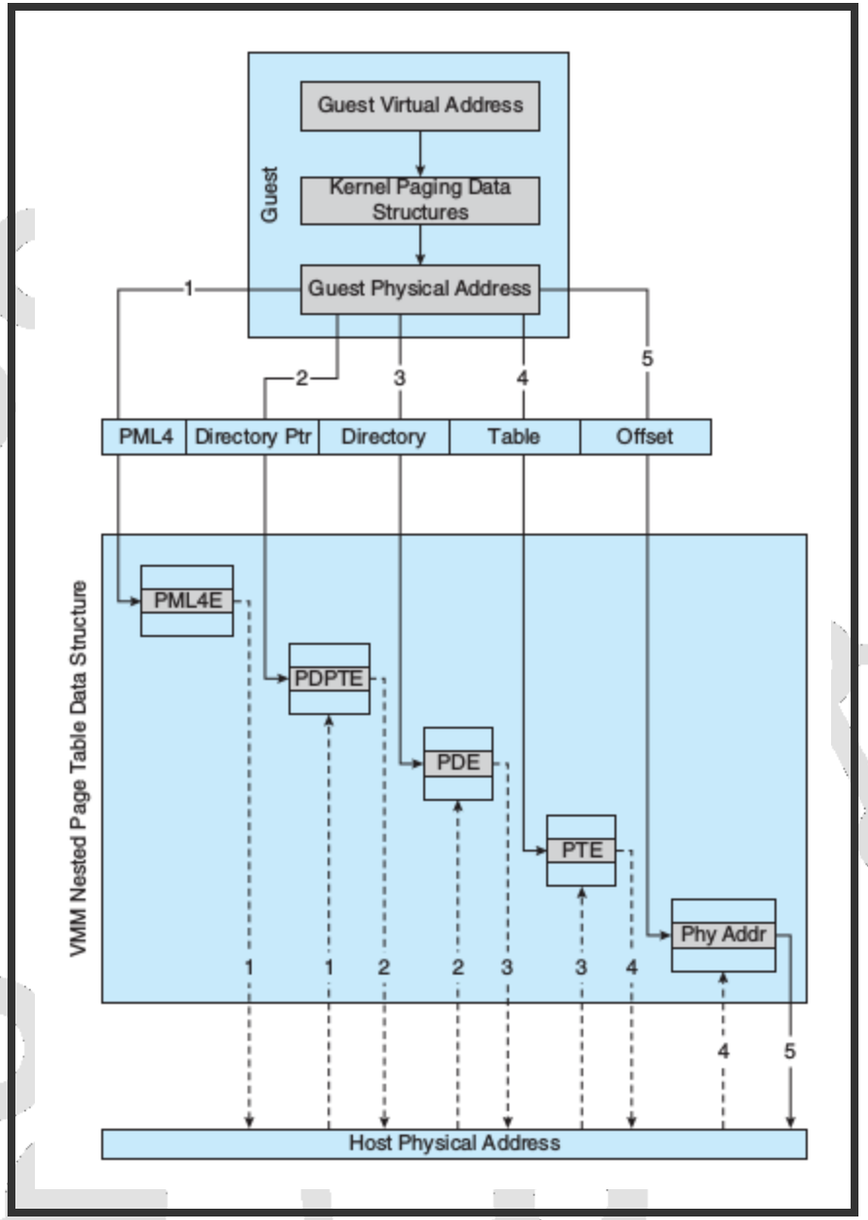
⚠ How can VMM keep page-table state for both guests believing they control the page tables and VMM that does control the tables?

Common method (for trap-and-emulate and binary translation) is **nested page tables** (**NPTs**)

NESTED PAGE TABLES

- Each guest maintains page tables to translate virtual to physical addresses
- VMM maintains per guest NPTs to represent guest's page-table 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

NESTED PAGE TABLES



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”

HARDWARE ASSISTANCE

- 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

TYPES OF VIRTUAL MACHINES AND THEIR IMPLEMENTATIONS

TYPES OF VIRTUAL MACHINES AND THEIR 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

VIRTUAL MACHINE LIFE CYCLE

1. Created by VMM
2. 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

VIRTUAL MACHINE LIFE CYCLE

- When no longer needed, VM can be deleted, freeing resouces
- 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

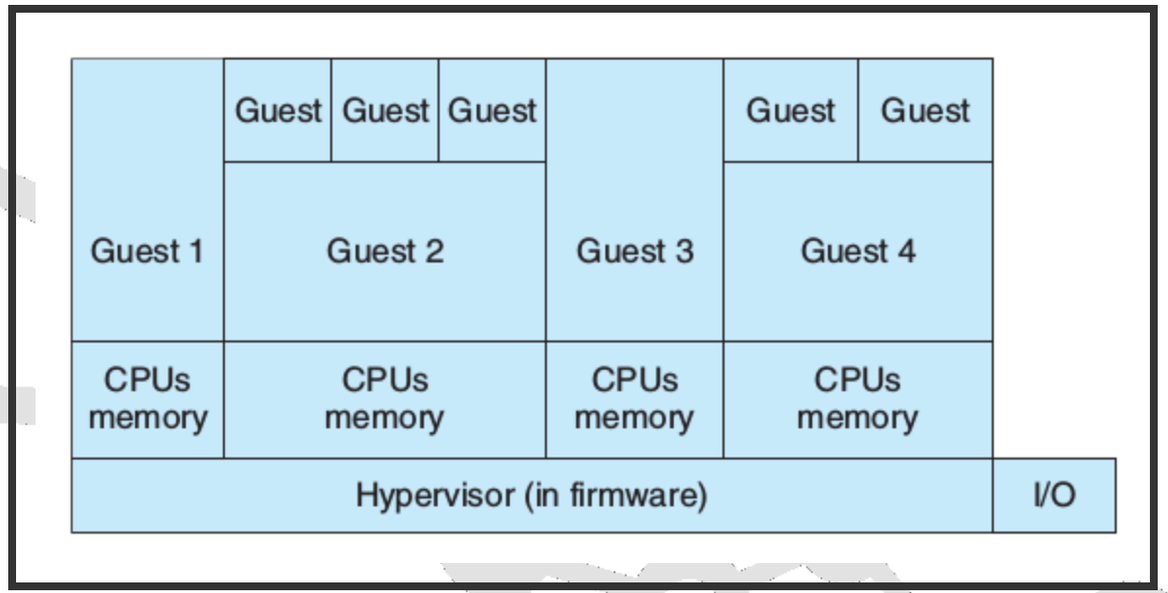
TYPE 0 HYPERVISOR

- Old idea, under many names by HW manufacturers
  - “partitions”, “domains”
  - A HW feature implemented by firmware
  - OS need to nothing special, VMM is in firmware
  - Smaller feature set than other types
  - Each guest has dedicated HW

TYPE 0 HYPERVISOR

- I/O a challenge as difficult to have enough devices, controllers to dedicate to each guest
- Sometimes VMM implements a **control partition** running daemons 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

TYPE 0 HYPERVISOR



TYPE 1 HYPERVISOR

- Special purpose OS 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 Types
  - Run in kernel mode
  - Guests generally don’t know they are running in a VM
  - Implement device drivers for host HW
  - Also provide other traditional OS services like CPU and memory management

TYPE 1 HYPERVISOR

- 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

TYPE 1 HYPERVISOR

- 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

TYPE 2 HYPERVISOR

- 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

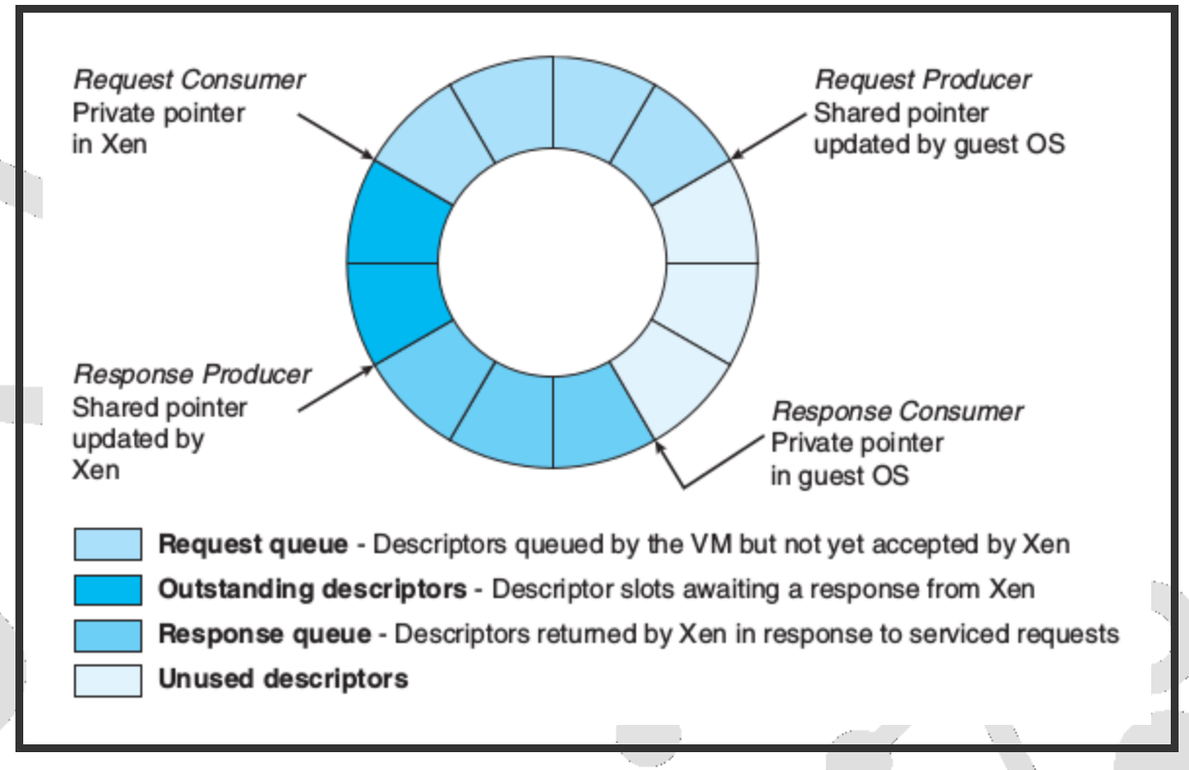
PARAVIRTUALIZATION

- 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

PARAVIRTUALIZATION

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

XEN I/O VIA SHARED CIRCULAR BUFFER



PARAVIRTUALIZATION

- Xen, leader in paravirtualized space, adds several techniques
  - 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

- Paravirtualization allowed virtualization of older x86 CPUs (and others) without binary translation
- Guest had to be modified to use run on paravirtualized VMM
- But on modern CPUs Xen no longer requires guest modification → no longer paravirtualization

PROGRAMMING ENVIRONMENT VIRTUALIZATION

- 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)
- Here, virtualization is defined as providing APIs that define set of features for a language and programs in that language to provide an improved execution environment

PROGRAMMING ENVIRONMENT VIRTUALIZATION

- 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

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

EMULATION

- 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 in gaming → old consoles emulated on new

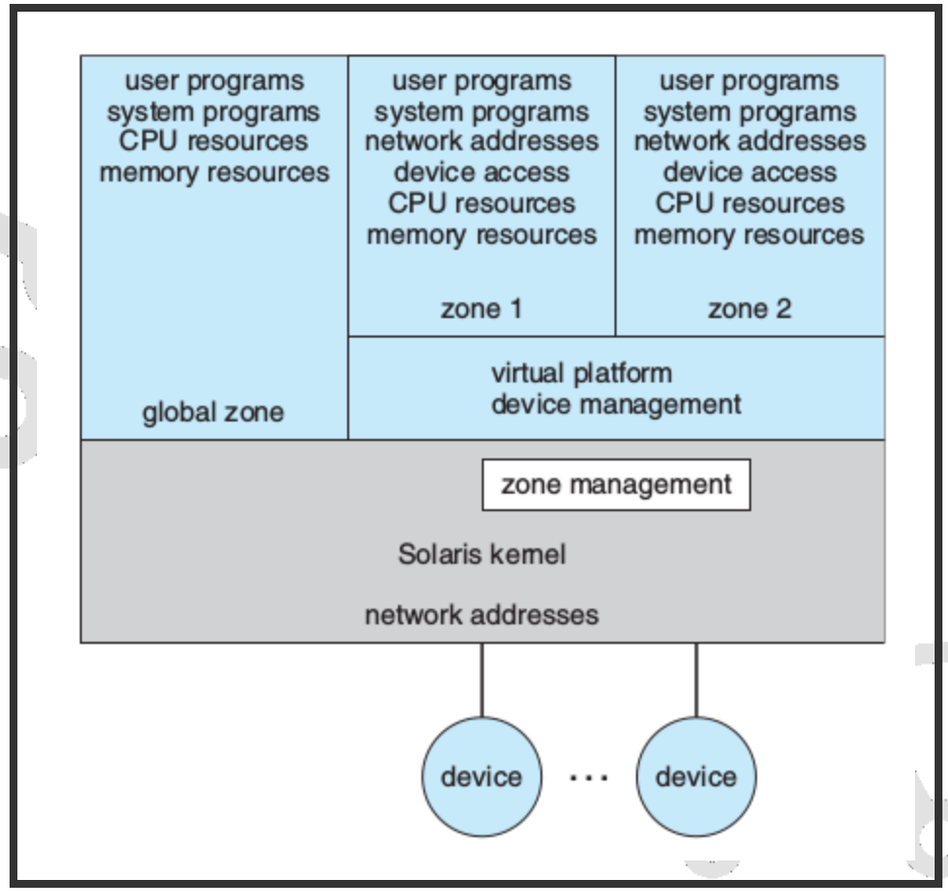
APPLICATION CONTAINMENT

- 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

APPLICATION CONTAINMENT

- 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 in zone with impression they are only processes
  - 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

SOLARIS 10 WITH TWO ZONES



VIRTUALIZATION AND OPERATING-SYSTEM COMPONENTS

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

OS COMPONENT – CPU SCHEDULING

- 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

OS COMPONENT – CPU SCHEDULING

- Usually not enough CPUs → **CPU overcommitment**
  - VMM can use standard scheduling algorithms to put threads on CPUs
  - Some add fairness aspect

OS COMPONENT – CPU SCHEDULING

- 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

MEMORY MANAGEMENT

1. 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
2. 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
3. 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

OS COMPONENT – I/O

- 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

OS COMPONENT – I/O

- 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

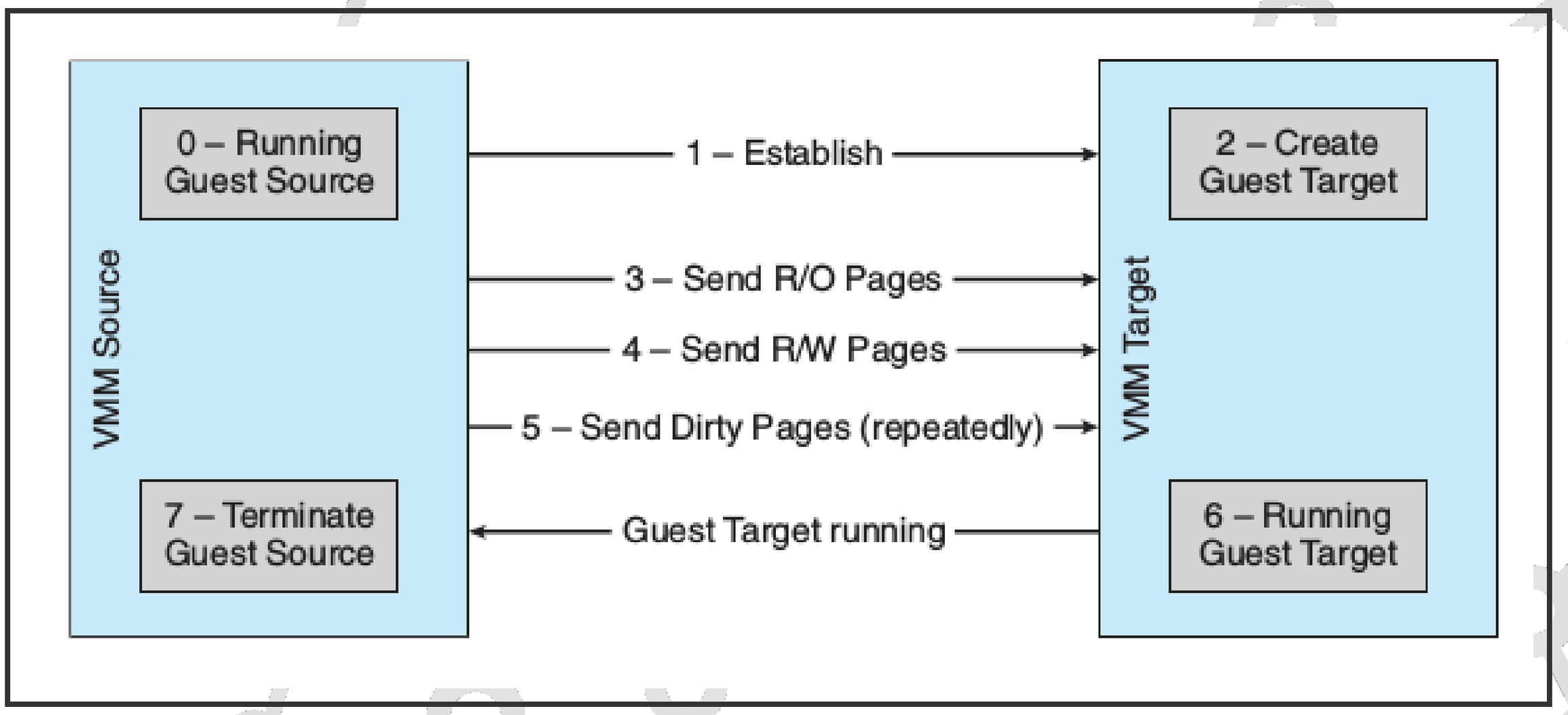
OS COMPONENT – STORAGE MANAGEMENT

- 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

OS COMPONENT – LIVE MIGRATION

- 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
- Very useful for resource management, maintenance downtime windows, etc

LIVE MIGRATION OF GUEST BETWEEN SERVERS



OS COMPONENT – LIVE MIGRATION

1. Source VMM establishes a connection with target VMM
2. Target creates a new guest by creating a new VCPU, etc
3. Source sends all read-only guest memory pages to target
4. Source sends all read-write pages to the target, marking them as clean
5. Source repeats step 4, as during that step some pages were probably modified by the guest and are now dirty
6. When cycle of steps 4+5 becomes very short, source VMM freezes guest, sends VCPU's final state, other state details, and final dirty pages, + tells target to start running guest

EXAMPLES

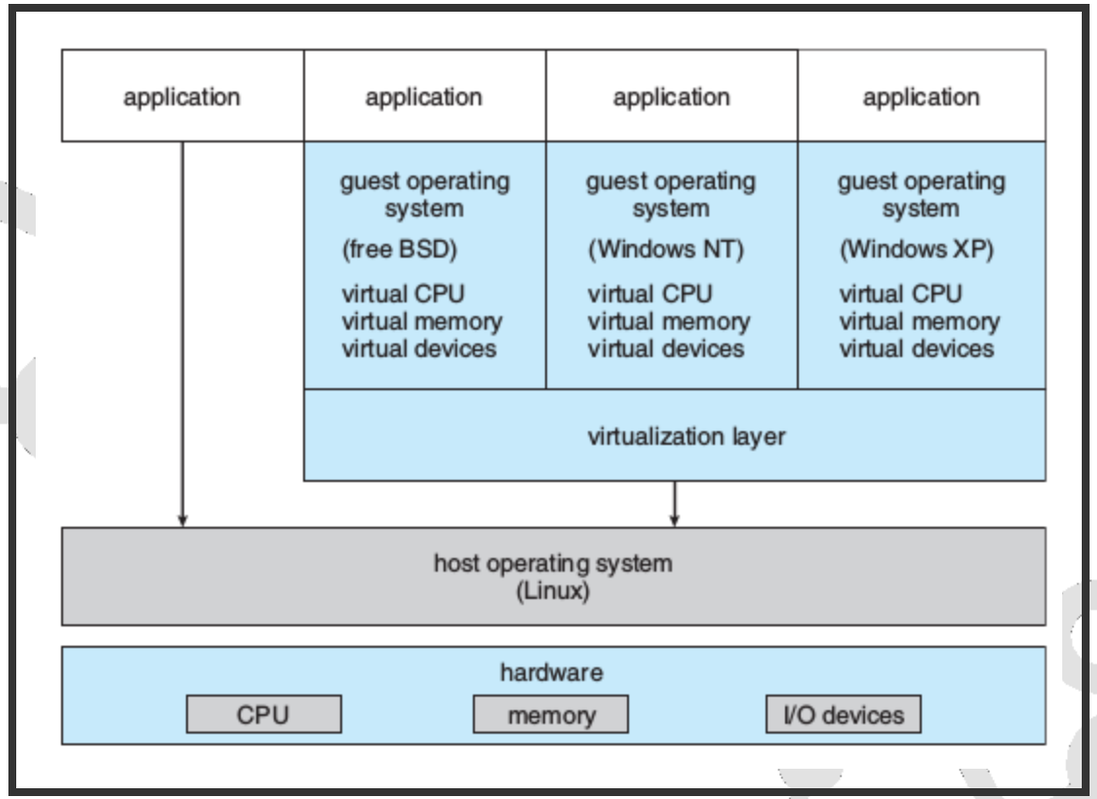
VMWARE

- VMware Workstation runs on x86, provides VMM for guests
- Runs as application on other native, installed host operating system → Type 2
- Lots of guests possible, including Windows, Linux, etc all runnable concurrently (as resources allow)

VMWARE

- Virtualization layer abstracts underlying HW, providing guest with is own virtual CPUs, memory, disk drives, network interfaces, etc
- Physical disks can be provided to guests, or virtual physical disks (just files within host file system)

VMWARE WORKSTATION ARCHITECTURE



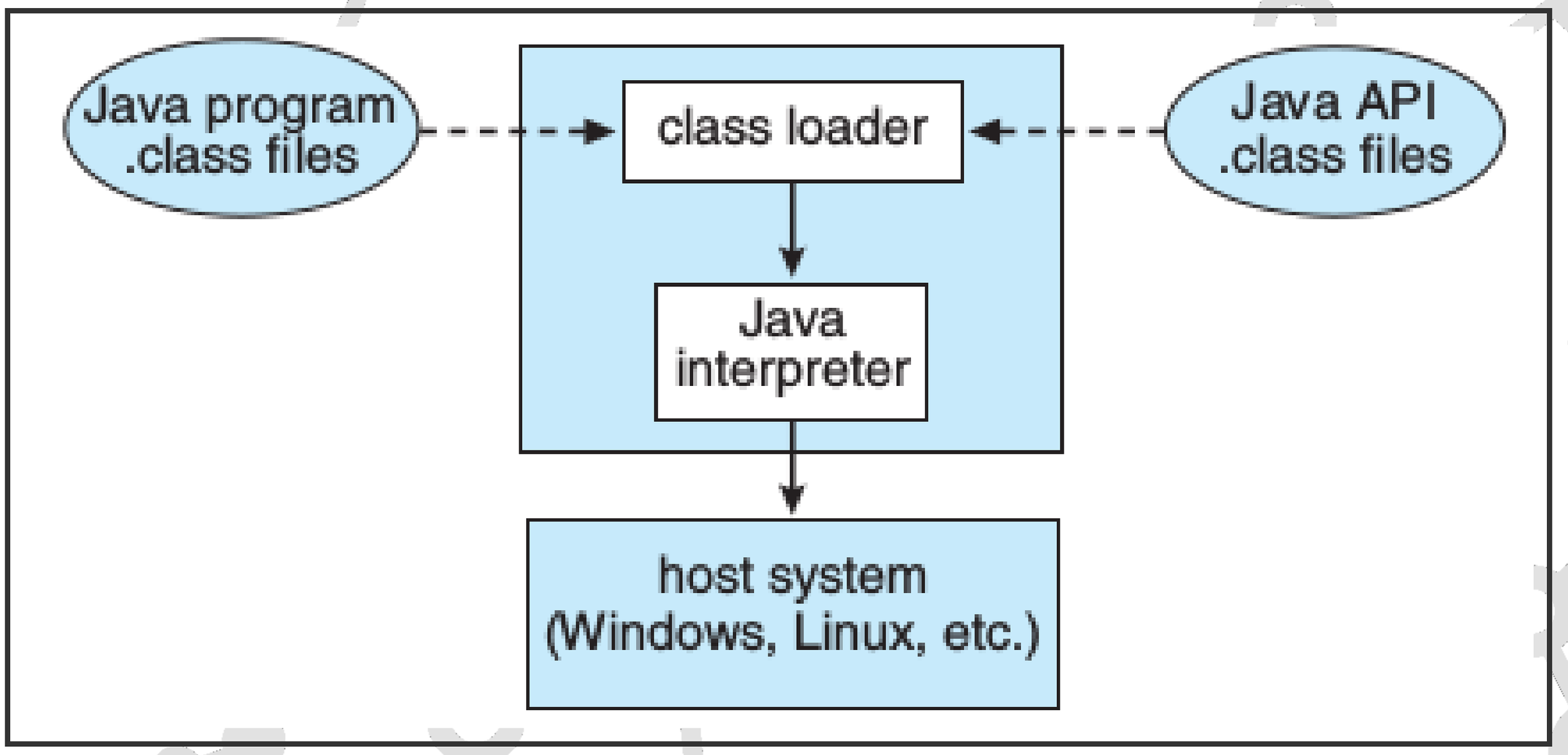
JAVA VIRTUAL MACHINE

- Example of programming-environment virtualization
- Very popular language / application environment invented by Sun Microsystems in 1995
- Write once, run anywhere
- Includes language specification (Java), API library, Java virtual machine (JVM)
- Java objects specified by class construct, Java program is one or more objects

JAVA VIRTUAL MACHINE

- Each Java object compiled into architecture-neutral bytecode output (.class) which JVM class loader loads
- JVM compiled per architecture, reads bytecode and executes
- 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

JAVA VIRTUAL MACHINE



CONTAINERS

DOCKER

Docker originally used LinuX Containers (LXC), but later switched to runC (formerly known as libcontainer)

- Runs in the same operating system as its host.
- Allows to share a lot of the host operating system resources.
- Uses a layered filesystem (AuFS)
- Manages networking.

DOCKER - AUFS

AuFS is a layered file system, so you can have a read only part and a write part which are merged together.

One could have the common parts of the operating system as read only (and shared amongst all of your containers) and then give each container its own mount for writing.

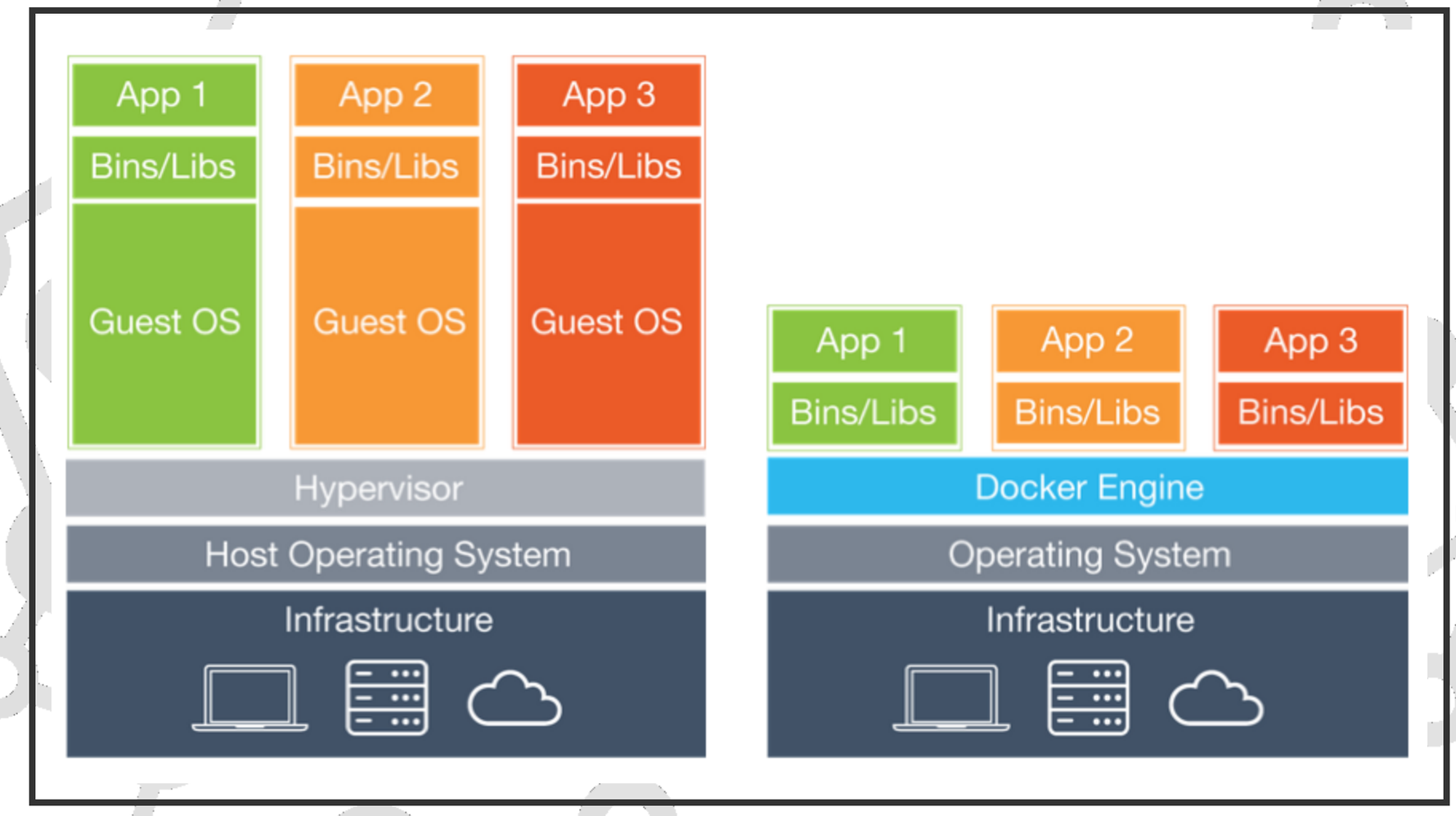
LAYERS

Docker gives you the ability to snapshot the OS into a shared image, and makes it easy to deploy on other Docker hosts.

All the same image.

You start with a base image, and then make your changes, and commit those changes using docker, and it creates an image. This image contains only the differences from the base.

DIFFERENCE


App 1
Bins/Libs
Guest OS
App 2
Bins/Libs
Guest OS
App 3
Bins/Libs
Guest OS
Hypervisor
Host Operating System
Infrastructure
App 1
Bins/Libs
App 2
Bins/Libs
App 3
Bins/Libs
Docker Engine
Operating System
Infrastructure

SIZE

Let’s say you have a 1 GB container image

If you wanted to use a full VM, you would need to have 1 GB times x number of VMs you want.

With Docker and AuFS you can share the bulk of the 1 GB between all the containers.

If you have 1000 containers you still might only have a little over 1 GB of space for the containers OS (assuming they are all running the same OS image).

DIFFERENCES

VM

A full virtualized system gets its own set of resources allocated to it, and does minimal sharing.

You get more isolation, but it is much heavier (requires more resources).

Container

With Docker you get less isolation, but the containers are lightweight (require fewer resources).

So you could easily run thousands of containers on a host, and it won't even blink.

SPEED

- Full virtualized system usually takes minutes to start
- Docker/LXC/runC containers take seconds, and often even less than a second.

PROS/CONS

If you want full isolation with guaranteed resources, a full VM is the way to go.

If you just want to isolate processes from each other and want to run a ton of them on a reasonably sized host, then Docker/LXC/runC seems to be the way to go.

QUESTIONS

BONUS

- Exam question number 12: **Virtual Machines**