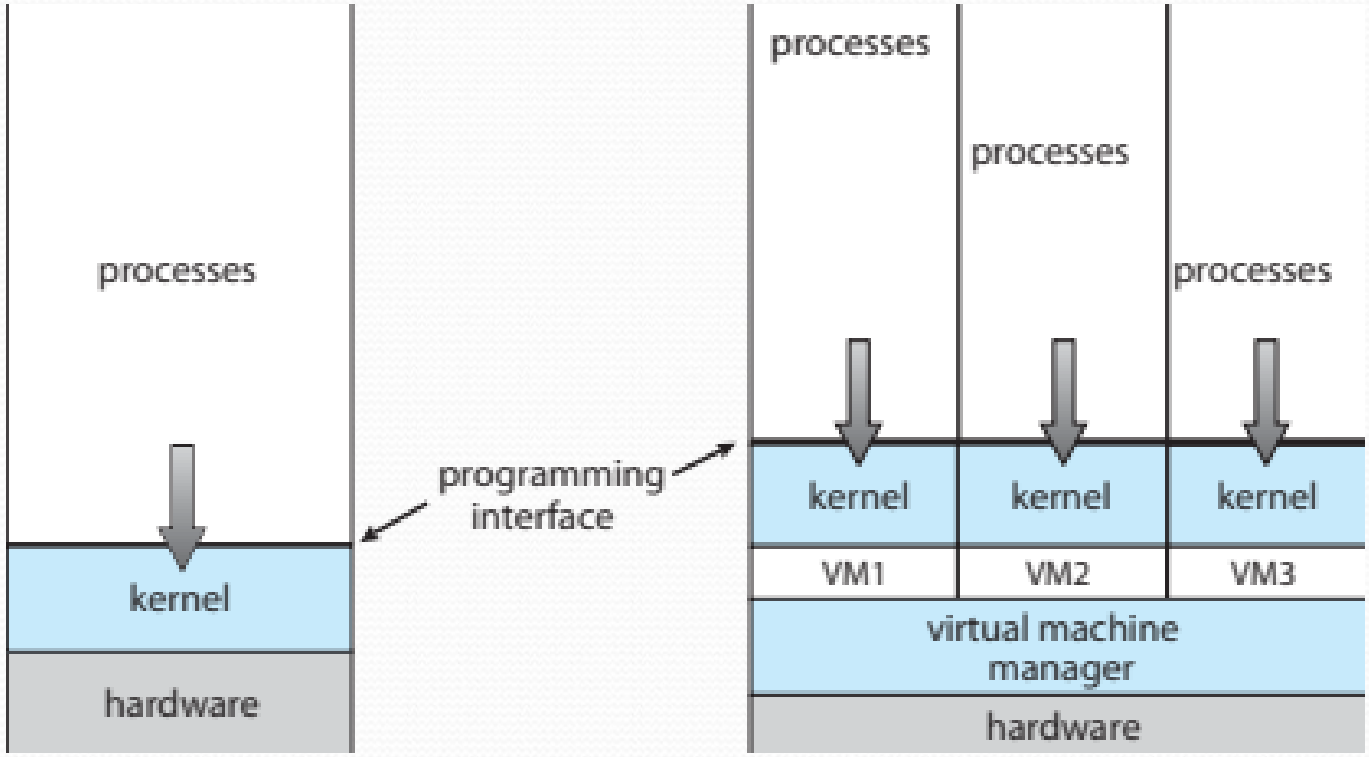
Module 6
Virtualization

Virtual Machine - 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
  - **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



Non-virtual machine

Virtual machine

Why are VM's important in today's Computing?

- Increasing importance of isolation and security in modern systems
- Failures in security and reliability of standard operating systems
- Sharing of a single computer among many unrelated users
- Dramatic increases in raw speed of processors, which makes the overhead of VMs more acceptable

Benefits and Features

- Host system is protected from the virtual machines
- Some VMMs include a **live migration** feature that moves a running guest from one physical server to another without interrupting its operation or active network connections.
- **Cloud computing** is made possible by virtualization in which resources such as CPU, memory, and I/O are provided as services to customers using Internet technologies.

Hypervisor

- Software that supports VMs is called a virtual machine monitor (VMM) or hypervisor
- Underlying hardware platform → Host
- Its resources shared among → Guest VMs
- Functionalities of Hypervisor
  - Managing Software
  - Managing Hardware

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
  - **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
    - Includeing VMware Workstation and Fusion, Parallels Desktop, and Oracle VirtualBox

Implementation of VMMs (cont.)

- 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

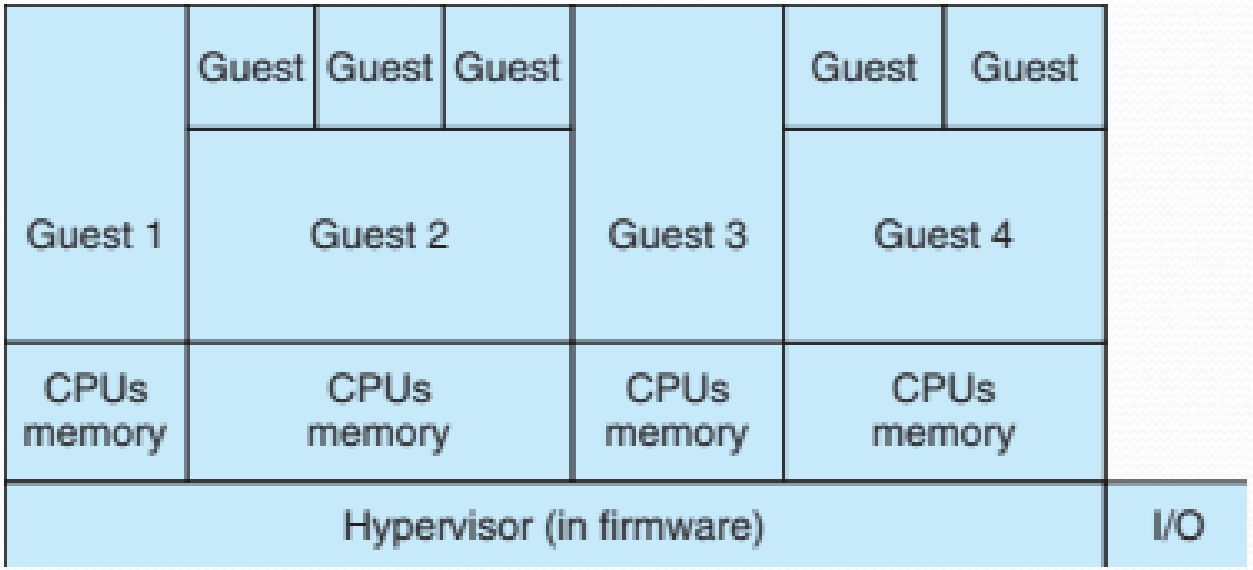
Types of Virtual Machines and Implementations

- 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 o hypervisor, resources usually dedicated
  - Other types dedicate or share resources, or a mix
  - 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

Types of VMs – 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
- 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



Types of VMs – 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
- 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
  - Implement device drivers for host HW because no other component can
  - Also provide other traditional OS services like CPU and memory management

Types of VMs – Type 1 Hypervisor (cont.)

- 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

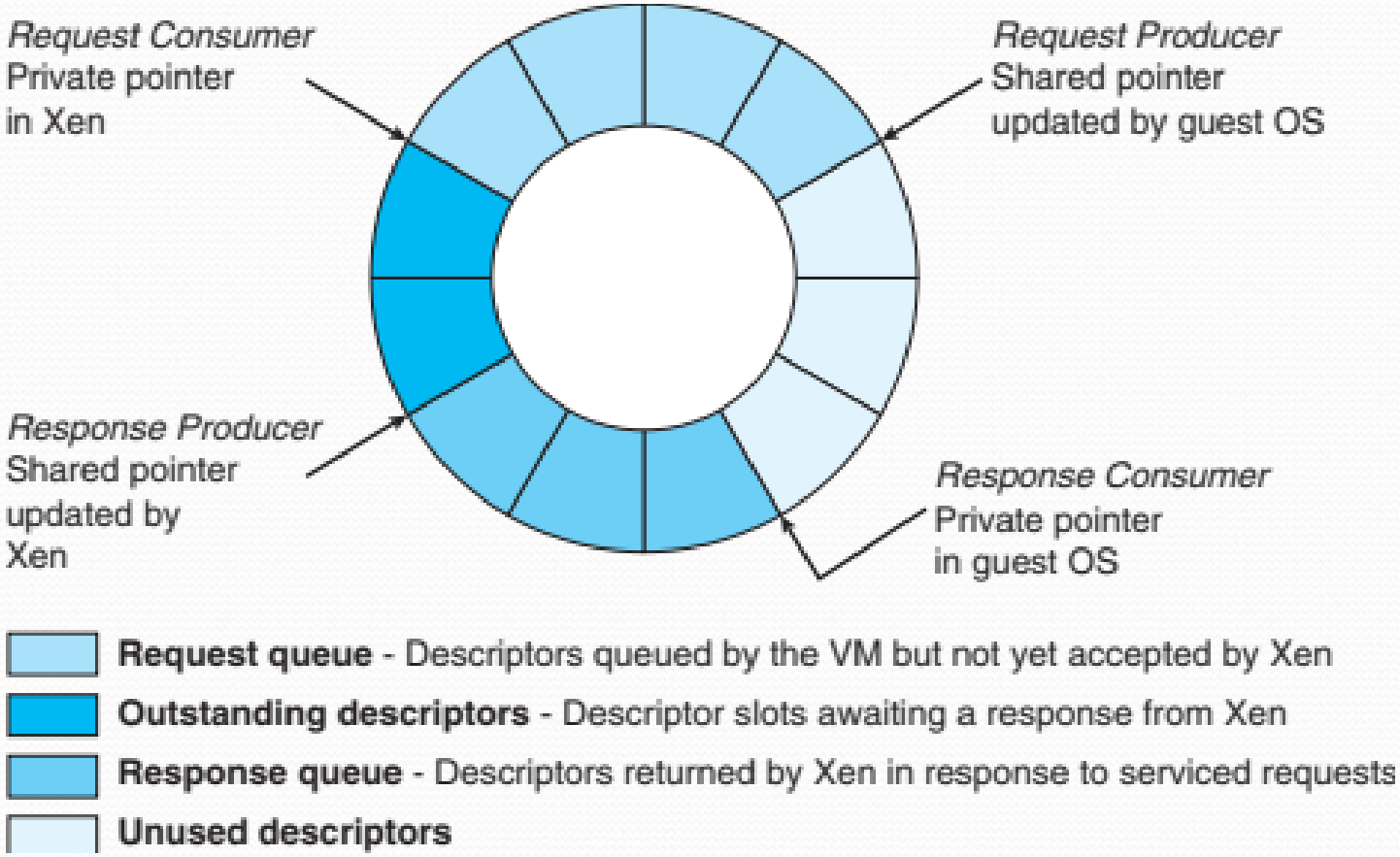
Types of VMs – Type 2 Hypervisor

- 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

Types of VMs – 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
- 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



Types of VMs – Paravirtualization (cont.)

- 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
- But on modern CPUs Xen no longer requires guest modification -> no longer paravirtualization

Types of VMs – 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**)
- 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**

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

Virtual Memory Ballooning

- Virtual memory ballooning is a computer memory reclamation technique used by a hypervisor to allow the physical host system to retrieve unused memory from certain guest VMs and share it with others.

- Memory ballooning allows the total amount of RAM required by guest VMs to exceed the amount of physical RAM available on the host.

- If a VM only uses a portion of the memory that it was allocated, the ballooning technique makes it available for the host to use.

Virtual Memory Ballooning

- Virtualization providers such as Vmware enable memory ballooning.
- The host uses balloon drivers running on the VMs to determine how much memory it can take back from an under-utilizing VM.
- Balloon drivers must be installed on any VM that participates in the memory ballooning technique.

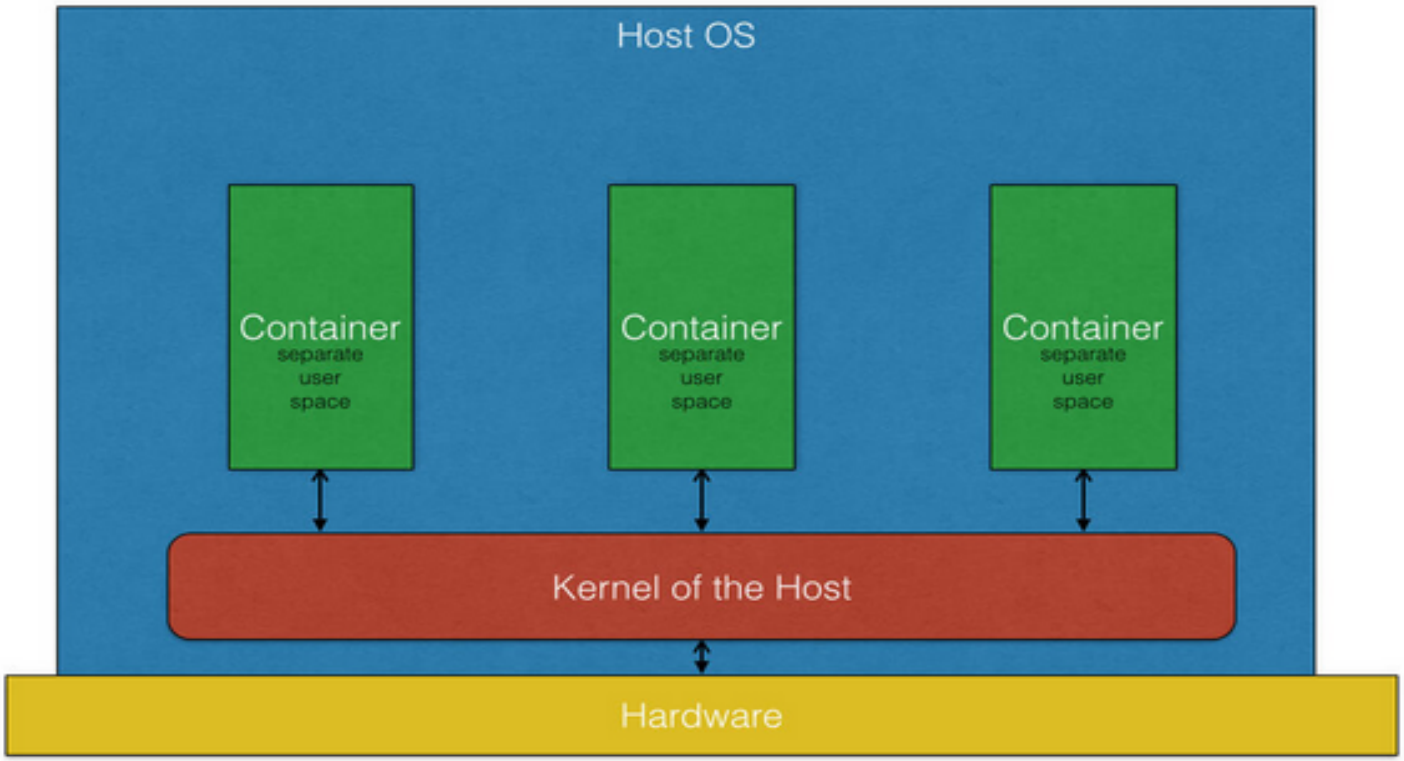
Virtual Memory Ballooning

- Balloon drivers get the target balloon size from the hypervisor and then inflate by allocating the proper number of guest physical pages within the VM. This process is known as inflating the balloon
- The process of releasing the available pages is known as deflating the balloon.
- VM memory ballooning can create performance problems.
- When a balloon driver inflates to the point where the VM no longer has enough memory to run its processes within itself, it starts using another VM memory technique known as memory swapping.

Containers Versus Virtual Machines

What are Containers?

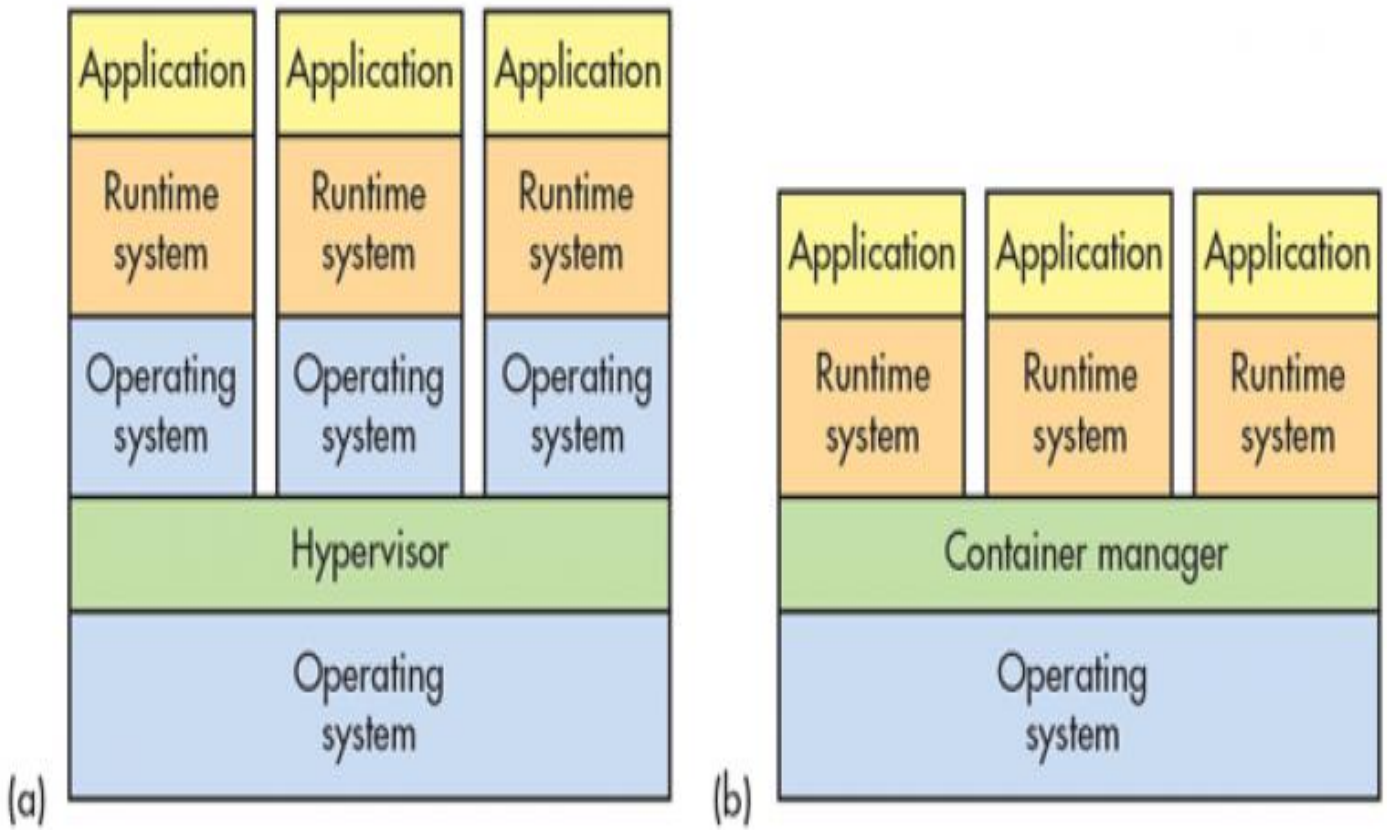
- Containers are the products of operating system virtualization



Operating System/Container Virtualization

Difference between VMs and Containers

- Containers → OS Level Virtualization
- Hypervisor Provides Hardware Level Virtualization
- Containers share the same kernel of the host system
- Type of containers that can be installed on the host should work with the kernel of the host
  - Cannot install a Windows container on a Linux host or vice-versa
- VMs are better with security compared to containers



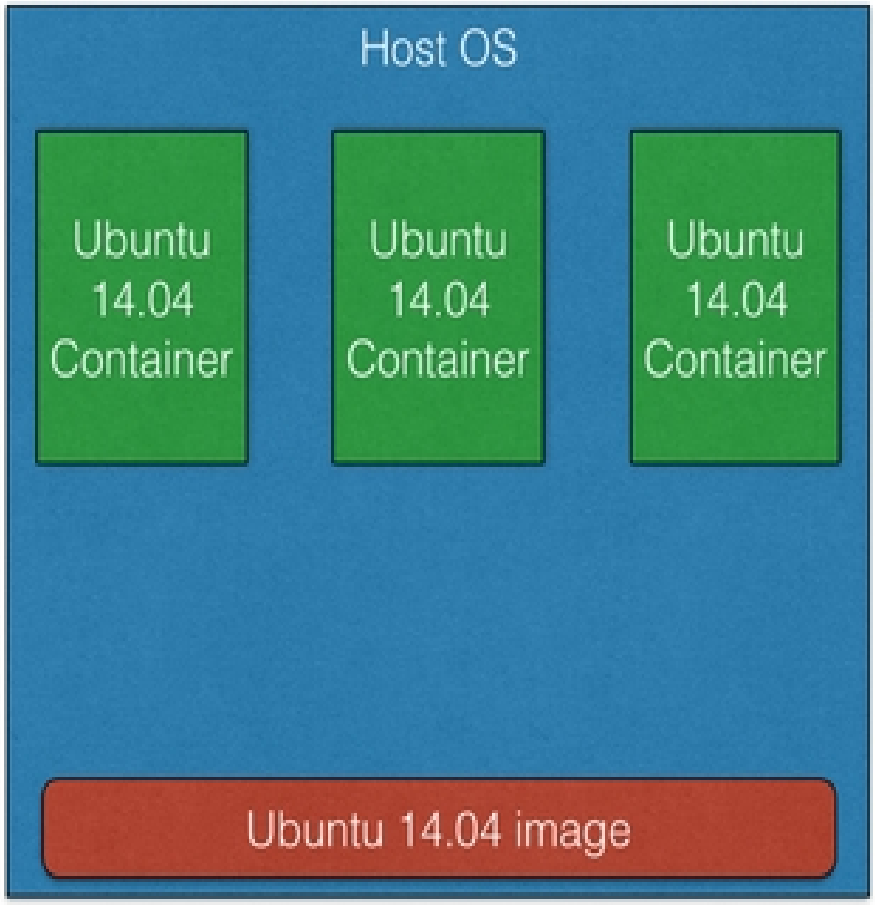
Virtual Machine

Container

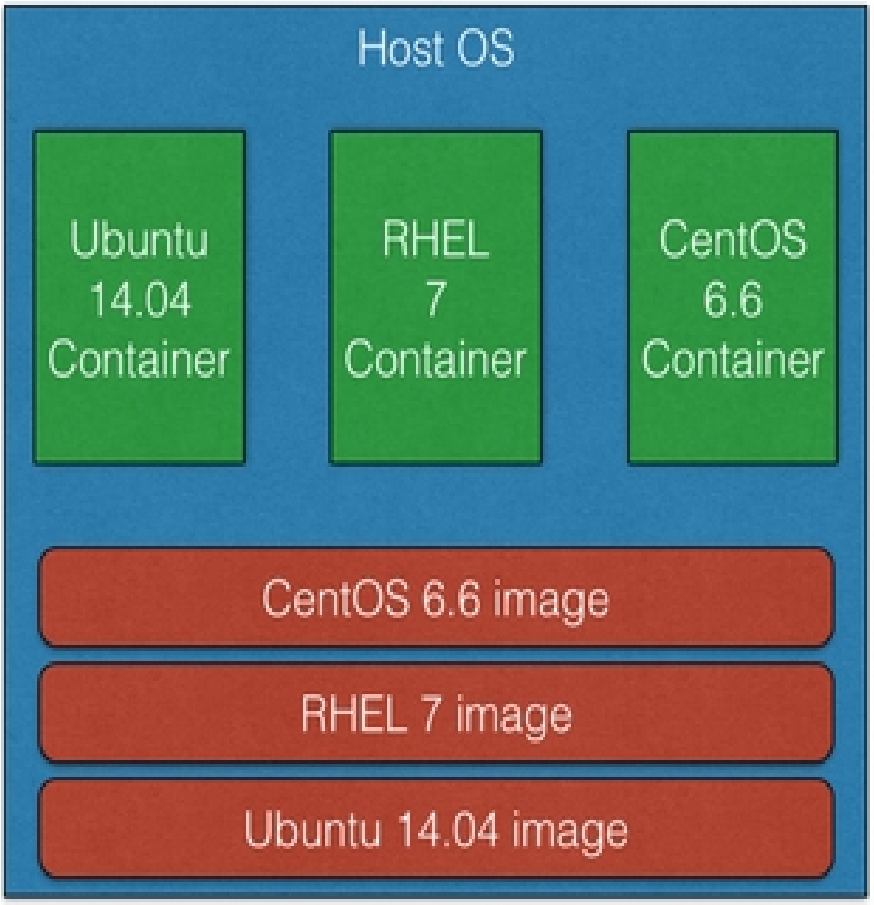
When to use Containers?

- OS Containers
  - share the kernel of the host
  - provide user space isolation
  - Used to run a fleet of identical or different flavours of OS
- Application containers
  - Designed to package and run a single service
  - Eg: Docker and Rocket

OS Container



Identical OS containers



Different flavoured OS containers

Live Migration

- **Live migration** refers to the process of moving a running virtual machine or application between different physical machines without disconnecting the client or application.
- Memory, storage, and network connectivity of the virtual machine are transferred from the original guest machine to the destination

Live Migration Timeline

Stage 0:

- Pre-Migration stage – A target host will be preselected where the resources required to receive migration will be guaranteed.

Stage 1:

- Reservation – A request is submitted to migrate a VM from Host-A to Host-B. If the request is not fulfilled, then VM will continue to run on Host-A.

Stage 2:

- Repetitive Pre-Copy: During the first iteration, all memory pages are transferred from Host-A to Host-B. Subsequent iterations copy are only those pages dirtied during the previous transfer.

Live Migration Timeline

Stage 3:

- Stop (suspend)-and-Copy: In this phase, VM will be suspended on Host-A and redirect its network traffic to Host-B. CPU state and any remaining inconsistent memory pages are then transferred like a final sync. This process will reach a consistent suspended copy of the VM at both Host-A and Host-B. Host-A will remain primary and it will be resumed in case of failure at this stage.

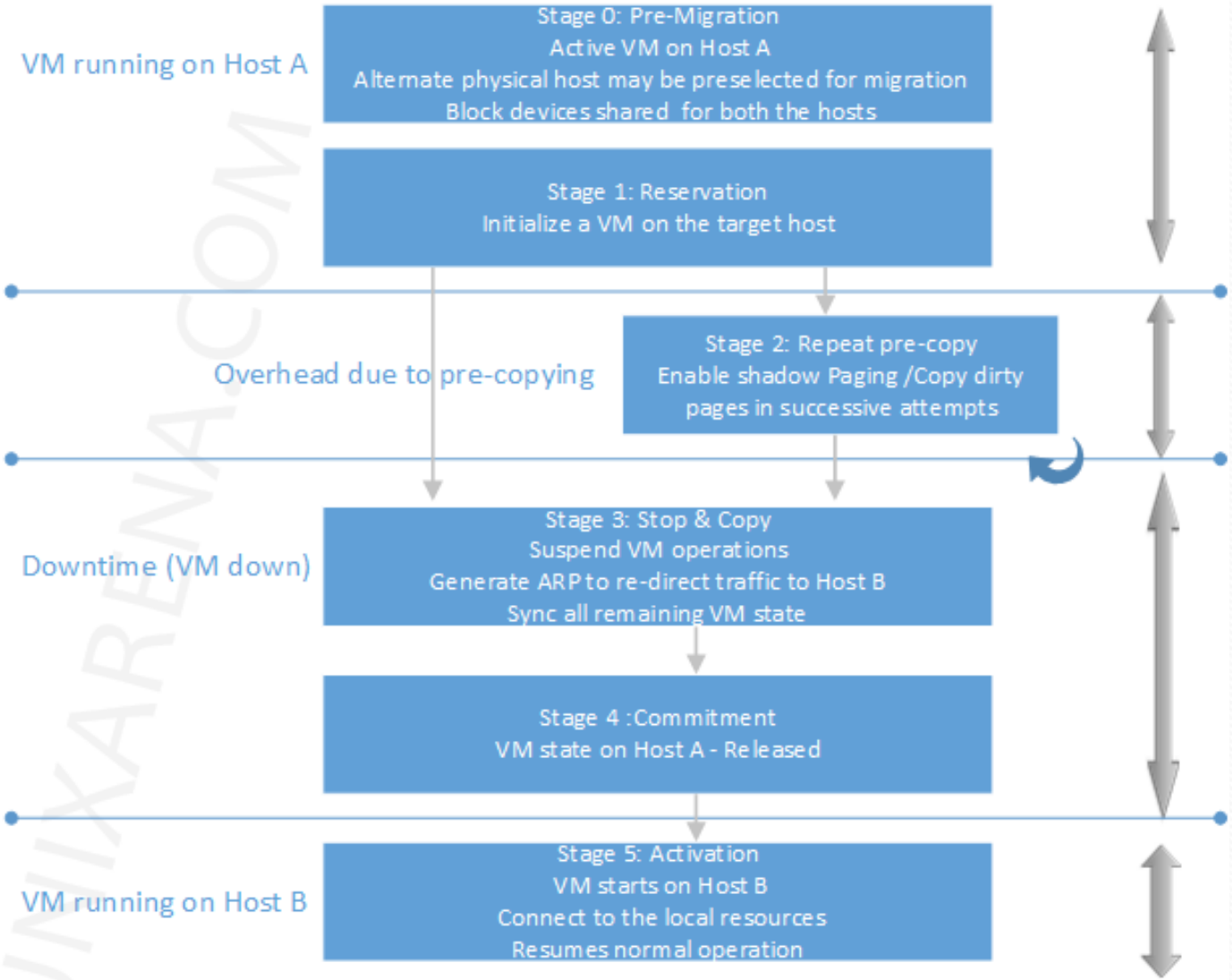
Stage 4:

- Commitment to the hosts: Host-B sends the signal to Host-A that it has successfully received a consistent VM OS image. Host-A acknowledges the signal and destroys the VM. Host-B becomes the primary host for migrated VM.

Stage 5:

- Activation of VM: The migrated VM on Host-B is now activated. Post-migration code connects to the local resources and resumes the operation.

VM Live - Migration Timeline



Live Migration

List of hypervisor products which supports the Guest OS Live Migration

- VMware vSphere – vMotion
- Xen – Live Migration
- Oracle VM for x86 – Live Migration
- RHEV – Live Migration.
- Oracle LDOM – Live Migration
- AIX – LPAR – Live Migraiton
- Microsoft Hyper-V

Any Queries?