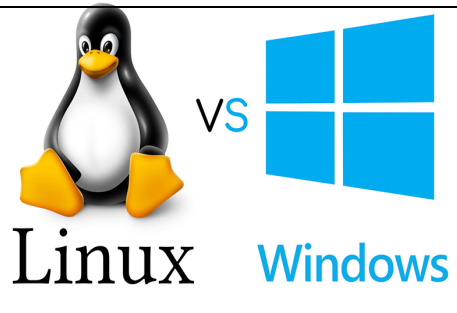
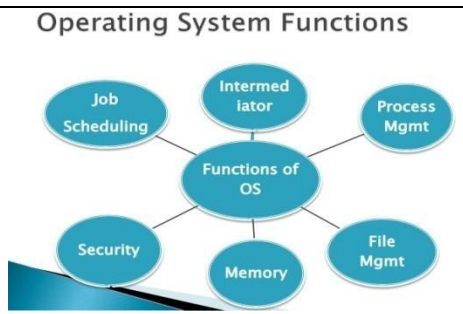
CIT210 OP. SYS. MANAG.

CSUN
COLLEGE OF ENGINEERING AND COMPUTER SCIENCE

Week 9 Day 1: OS Virtualization, Types of Virtual Machines Implementations, Hypervisors, Virtualization on Linux or Windows Platforms, Containers Docker Engines and Docker Containers, Docker Life Cycle





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**

Hypervisors

- A hypervisor, a.k.a. a virtual machine manager/monitor (VMM), or virtualization manager, is a program that allows multiple operating systems to share a single hardware host.
- Each guest operating system appears to have the host's processor, memory, and other resources all to itself.
- Hypervisor is actually controlling the host processor and resources, allocating what is needed to each operating system in turn and making sure that the guest operating systems (called virtual machines) cannot disrupt each other.

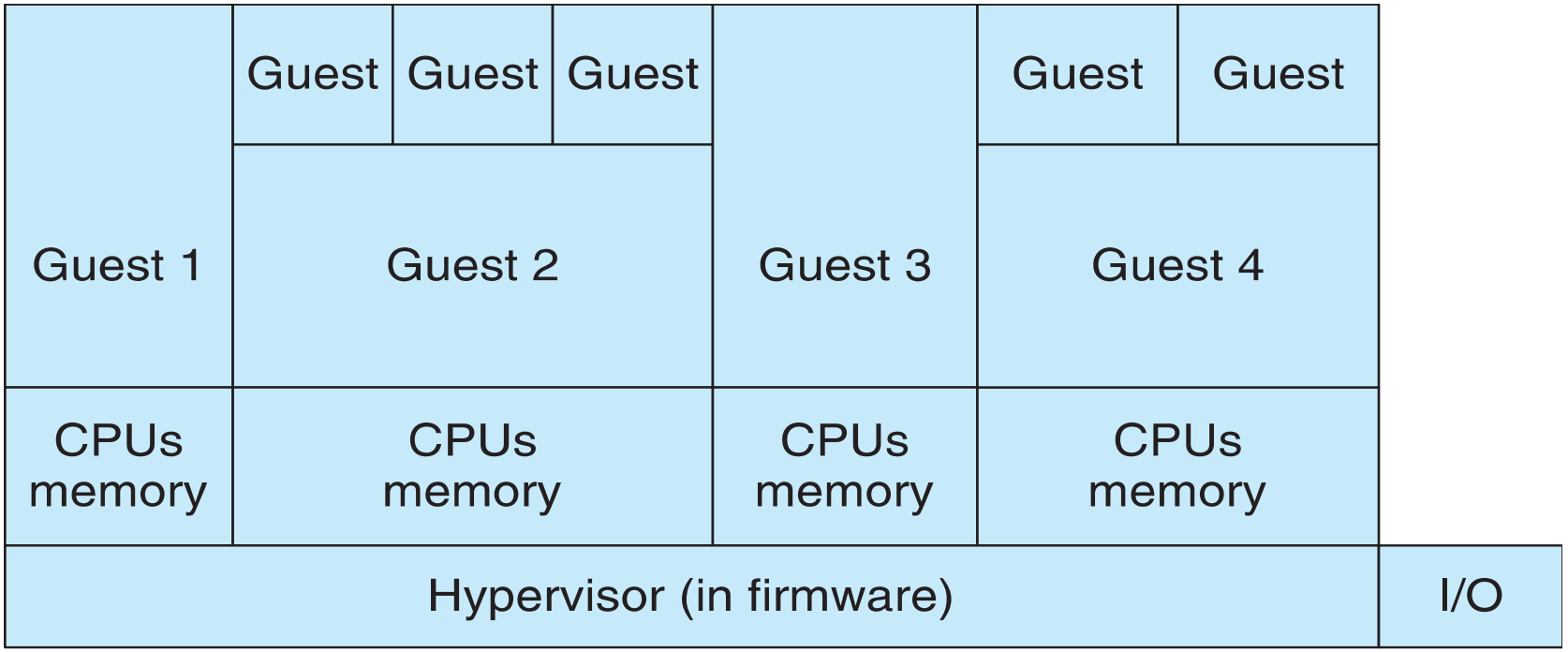
Types of VMs – Type 0 Hypervisor

- ***There is not such thing as a type 0 Hypervisor.***
- **Type 0 (older Technologies) is based on an architecture that allows for higher levels of performance, reliability, and security over Type-1 hypervisors.**
- **Type Zero hypervisor is built with the minimum software components required to fully virtualize guest OSs and control information flow between guest OSs.**
- **The Type 0 architecture removes the need for an embedded host OS to support virtualization, allowing the hypervisor to run in an "Un-Hosted" environment.**

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

Type 1 hypervisors:

- 1. **VMware ESX and ESXi**
  - **These hypervisors offer advanced features and scalability, but require licensing, so the costs are higher.**
  - **There are some lower-cost bundles that VMware offers and they can make hypervisor technology more affordable for small infrastructures.**
  - **VMware is the leader in the Type-1 hypervisors. Their vSphere/ESXi product is available in a free edition and 5 commercial editions.**

Type 1 hypervisors:

- 2. Microsoft Hyper-V
  - The Microsoft hypervisor, Hyper-V doesn't offer many of the advanced features that VMware's products provide. However, with XenServer and vSphere, Hyper-V is one of the top 3 Type-1 hypervisors.
  - It was first released with Windows Server, but now Hyper-V has been greatly enhanced with Windows Server 2012 Hyper-V. Hyper-V is available in both a free edition (with no GUI and no virtualization rights) and 4 commercial editions – Foundations (OEM only), Essentials, Standard, and Datacenter. Hyper-V

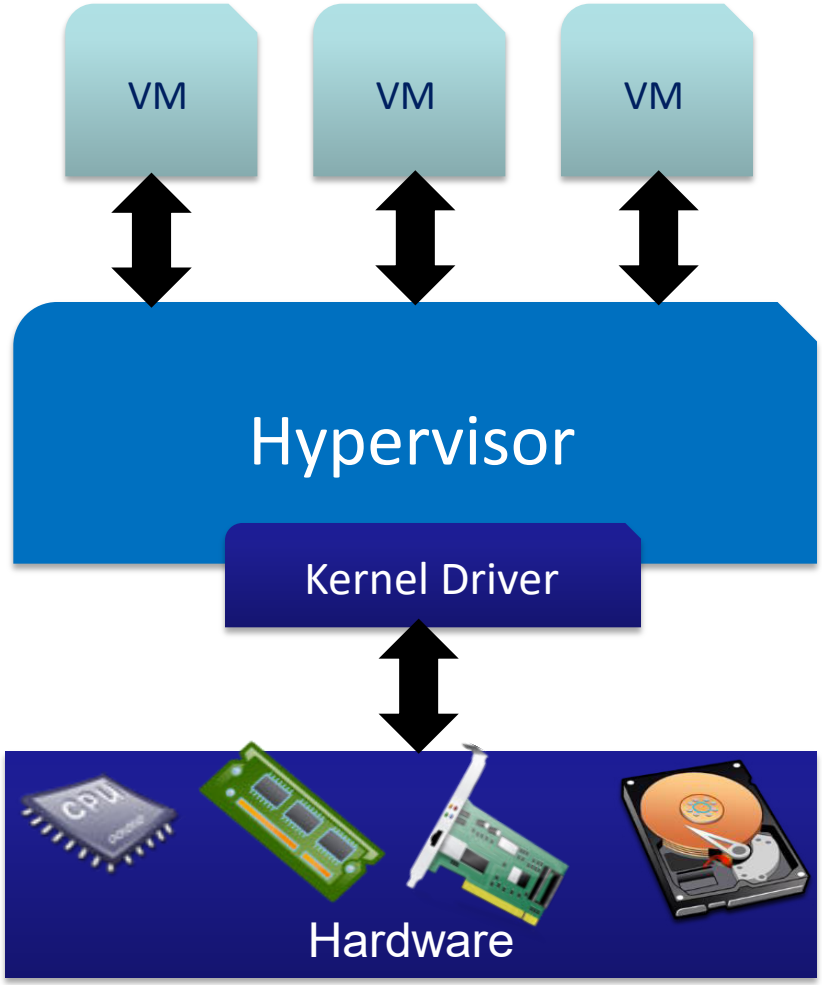
Type 1 hypervisors:

- 3. Citrix Xen Server
  - It began as an open source project.
    The core hypervisor technology is free, but like VMware's free ESXi, it has almost no advanced features. Xen is a type-1 bare-metal hypervisor. Just as Red Hat Enterprise Virtualization uses KVM, Citrix uses Xen in the commercial XenServer.
  - Today, the Xen open source projects and community are at Xen.org. Today, XenServer is a commercial type-1 hypervisor solution from Citrix, offered in 4 editions. Confusingly, Citrix has also branded their other proprietary solutions like XenApp and XenDesktop with the Xen name.

Type 1 hypervisors:

- 4. Oracle VM
  - The Oracle hypervisor is based on the open source Xen. However, if you need hypervisor support and product updates, it will cost you.
    Oracle VM lacks many of the advanced features found in other bare-metal virtualization hypervisors.

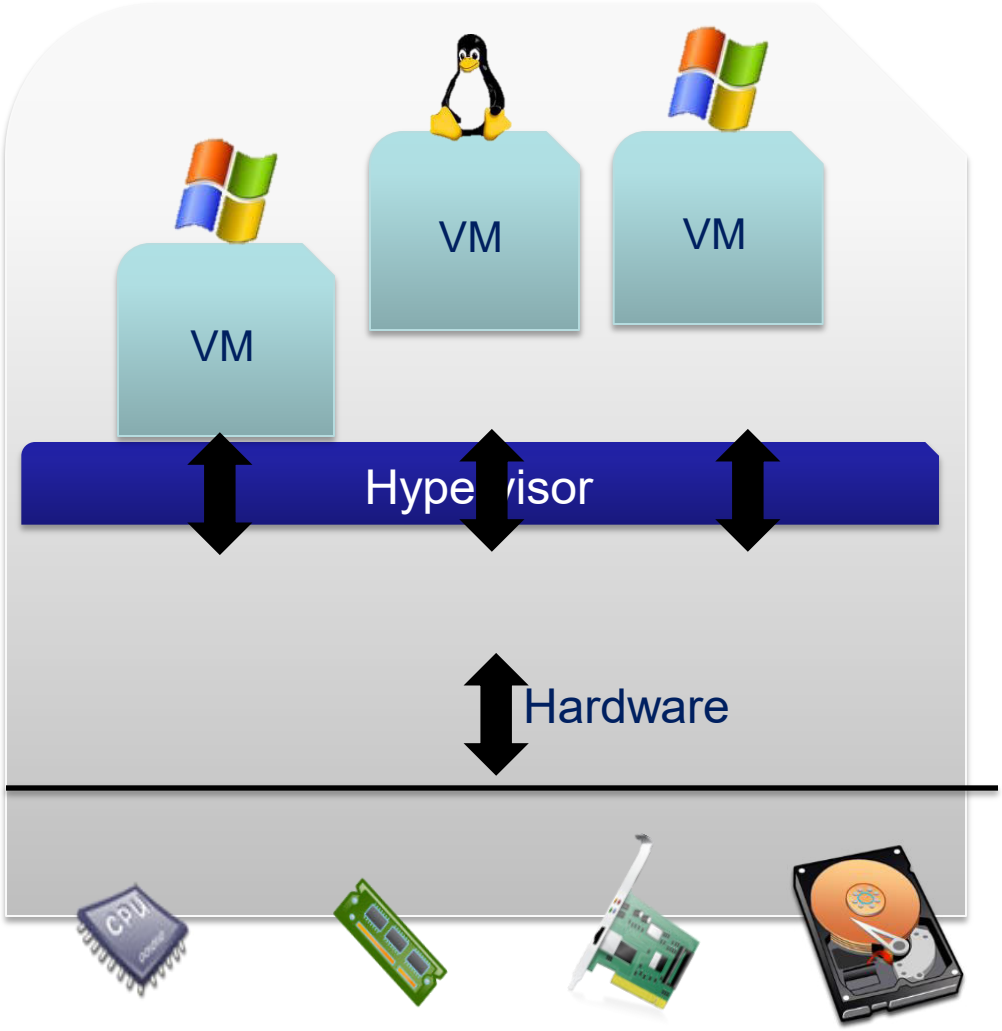
Hypervisor implementation approaches



Bare metal Approach

- Type I Hypervisor.
- Runs directly on the system hardware.
- May require hardware assisted virtualization technology support by the CPU.
- Limited set of hardware drivers provided by the hypervisor vendor.
- E.g.: Xen, VMWare ESXi

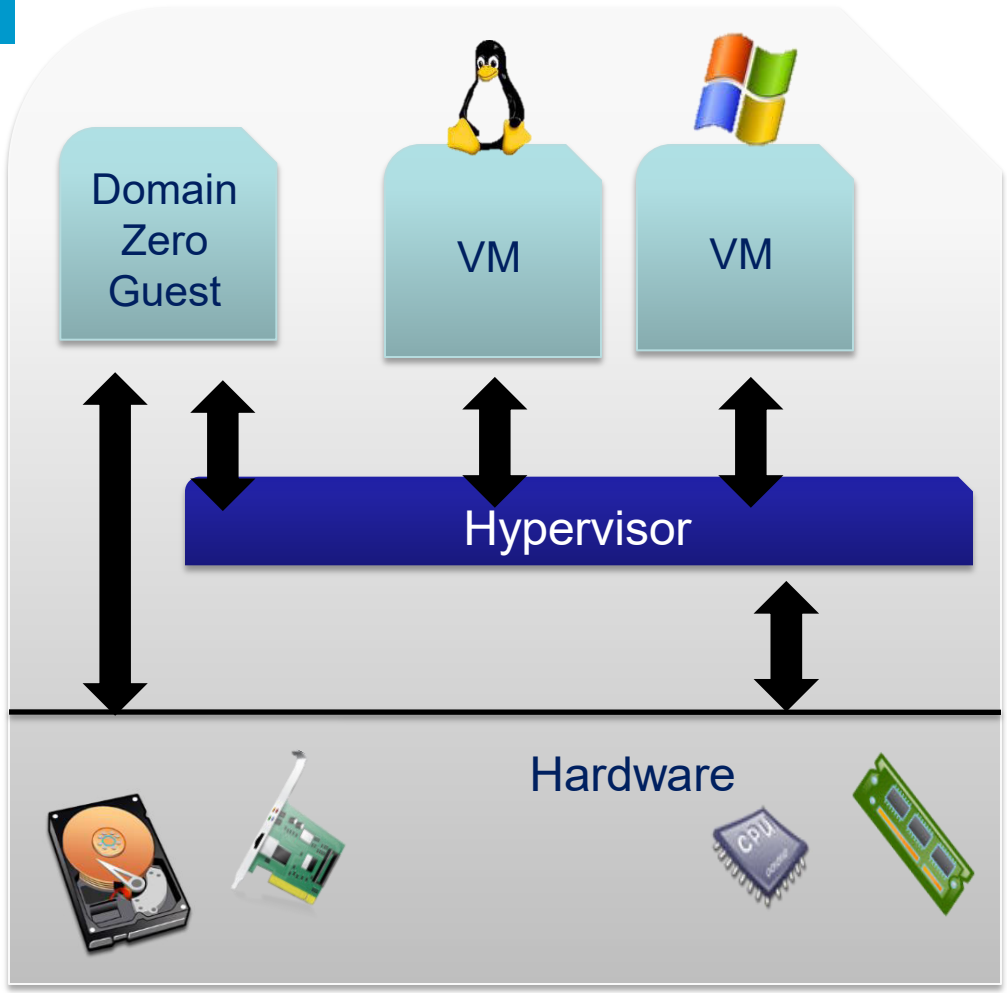
Vmware esxi



Architecture of VMWare ESXi

- Bare Metal Approach.
- Full virtualization.
- Proven technology.
- Used for secure and robust virtualization solutions for virtual data centers and cloud infrastructures.
- Takes advantage of support for hardware assisted virtualization for 64-bit OS on Intel processors.

Citrix xen server



Architecture of Xen

- Open source; bare metal.
- Offers both Hardware Assisted Virtualization (HVM) and Para-Virtualization (PV)
- Needs virtualization support in the CPU for HVM.
- Xen loads an initial OS which runs as a privileged guest called “domain 0”.
- The domain 0 OS, typically a Linux or UNIX variant, can talk directly to the system hardware (whereas the other guests cannot) and also talk directly to the hypervisor itself. It allocates and maps hardware resources for other guest domains.

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 – Type 2 Hypervisor

- 1. VMware Workstation/Fusion/Player
- VMware Player is a free virtualization hypervisor.
- It is intended to run only one virtual machine (VM) and does not allow creating VMs.
  VMware Workstation is a more robust hypervisor with some advanced features, such as record-and-replay and VM snapshot support.
- VMware Workstation has three major use cases:
- for running multiple different operating systems or versions of one OS on one desktop,
- for developers that need sandbox environments and snapshots, or for labs and demonstration purposes.

Types of VMs – Type 2 Hypervisor

- 2. VMware Server
- VMware Server is a free, hosted virtualization hypervisor that's very similar to the VMware Workstation. VMware has halted development on Server since 2009
- 3. Microsoft Virtual PC
- This is the latest Microsoft's version of this hypervisor technology, Windows Virtual PC and runs only on Windows 7 and supports only Windows operating systems running on it.

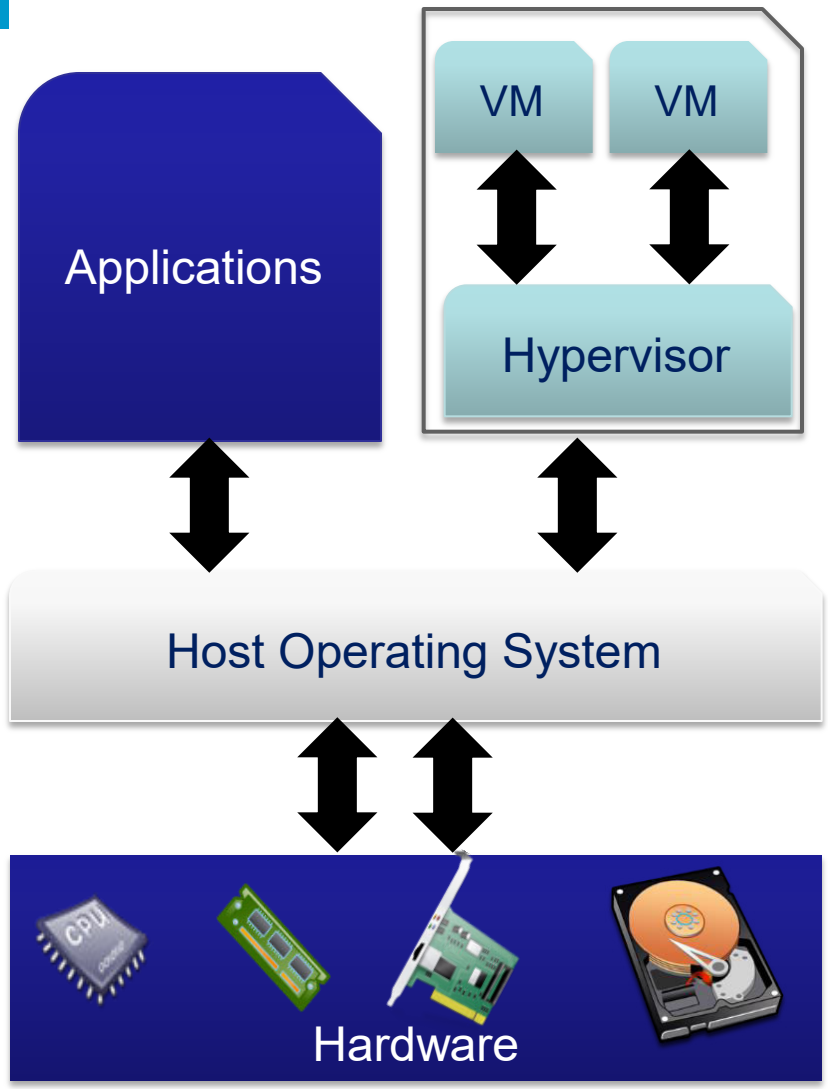
Types of VMs – Type 2 Hypervisor

- 4. Oracle VM VirtualBox
  - VirtualBox hypervisor technology provides reasonable performance and features if you want to virtualize on a budget. Despite being a free, hosted product with a very small footprint, VirtualBox shares many features with VMware vSphere and Microsoft Hyper-V.
- 5. Red Hat Enterprise Virtualization
  - Red Hat's Kernel-based Virtual Machine (KVM) has qualities of both a hosted and a bare-metal virtualization hypervisor. It can turn the Linux kernel itself into a hypervisor so the VMs have direct access to the physical hardware.

Types of VMs – Type 2 Hypervisor

- **KVM**
  - This is a virtualization infrastructure for the Linux kernel. It supports native virtualization on processors with hardware virtualization extensions.
  - The open-source KVM (or Kernel-Based Virtual Machine) is a Linux-based type-1 hypervisor that can be added to most Linux operating systems including Ubuntu, Debian, SUSE, and Red Hat Enterprise Linux, but also Solaris, and Windows.
  - We use KVM in VapourApps Private Cloud:
  - Virtualization engine – OpenStack on KVM
  - Predefined virtual servers based on Debian
  - Orchestration and management web dashboard, a customized Horizon dashboard.
  - The owner of the tenant or the IT administrator, can manage his virtual servers, users, groups and monitor the status of the used application from a single dashboard.

Hypervisor implementation approaches



Hosted Approach

- Type II Hypervisor.
- Runs virtual machines on top of a host OS (windows, Unix etc.)
- Relies on host OS for physical resource management.
- Host operating system provides drivers for communicating with the server hardware.
- E.g.: VirtualBox

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

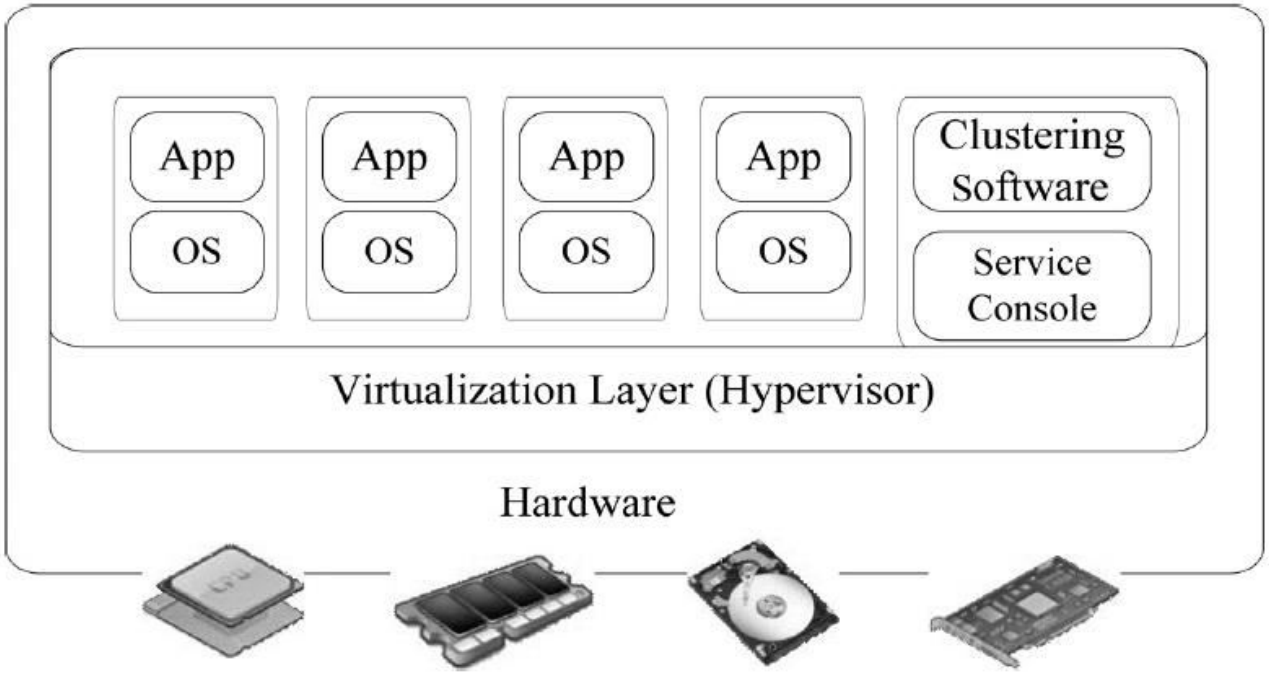
Hypervisors

- **Hypervisor supports a hardware-level virtualization**
  - **The hypervisor software sits directly between the physical hardware and the guest OS**
    - The hypervisor provides binary translation for the guest OSs and applications
  - **Depending on the functionality**
    - Can assume micro-kernel architecture like the Microsoft Hyper-V
    - Or monolithic hypervisor architecture like the VMware ESX for server virtualization
- **A micro-kernel hypervisor**
  - **Includes only the basic and unchanging functions**

Hypervisors (cont.)

- Such as physical memory management and processor scheduling

- **The device drivers and other changeable components are outside of the hypervisor**
  - The size of hypervisor code of a micro-kernel hypervisor is smaller



Hypervisors (cont.)

- A monolithic hypervisor implements all the functions
  - Including device drivers
- Hypervisor-created VMs are heavily weighted
  - Consist of the user application code which could be only KB
  - Plus a guest OS demanding GB of memory
    - The guest OS supervises the execution of the user applications on the VM
- The XEN is the most popular hypervisor
  - Used in almost all x86-based PCs, servers, or workstations

Hypervisors (cont.)

- **Kernel-based virtual machine (KVM) is a Linux kernel-based VMM**
  - **Mostly used in Linux hosts**
    - A part of the Linux version 2.6.20 kernel
  - **Memory management and scheduling activities are carried out by the existing Linux kernel**
    - The KVM does the rest, and thus is simpler than hypervisor, which controls the entire machine
  - **KVM is a hardware-assisted paravirtualization tool to improve performance and support OSs**
    - Like Windows, Linux, Solaris, and other UNIX
- **The Microsoft Hyper-V must be used for Windows server virtualization**

Hypervisors (cont.)

- **Involves OS integration at the lowest level**
- **Malware and rootkits could post potential threats to hypervisor security**
  - A rootkit is a collection of malicious computer software
  - Microsoft and academia have developed some anti-rootkit HookSafe software to protect hypervisors from malware and rootkit attacks

- **All VMware software is for hardware virtualization**
  - **Offers the largest selection of virtualization software products, toolkits, and systems**
    - Develops virtualization tools from desktops and servers to virtual infrastructure for large data centers

Hypervisors (cont.)

- Some software is supplied from a third party or via open sources

- **VMware started with a workstation version that can run with Windows and Linux hosts**
  - Really for full virtualization
- **Later, VMware launched the server ESX package for virtualization use in x.86 servers**
  - Requires no use of host OS to virtualize the resources
  - These hypervisors perform the para-virtualization
- **Now the cloud OS, vSphere, is entirely supported by VMware's own virtualization packages**
- **Some VMware VMM packages are not responsible for allocation resources for all user programs**
  - i.e., Players or VirtualBox

Hypervisors (cont.)

- **Only allocate restricted resources to selected special applications and controls these resources**

Hypervisor	Host CPU	Host OS	Guest OS	Architecture, Applications and User Community
XEN	×-86, ×-86–64, IA-64	NetBSD, Linux	Linux, Windows, BSD, Linux, Solaris	Native Hypervisor (Example 1.6) developed at Cambridge University
KVM	×-86, ×-86–64, IA-64, S390, PowerPC	Linux	Linux, Windows, FreeBSD, Solaris	Hosted Hypervisor based on para-virtualization at the user space
Hyper V	×-86 based	Server 2003	Windows servers	Windows based native hypervisor, marketed by Microsoft
VMWare Player, Workstation, VirtualBox	×-86, ×-8-6–64	Any host OS	Windows, Linux, Darwin Solaris, OS/2, Free BSD	Hosted hypervisor with a para-virtualization architecture shown in Figure 1.20(c)

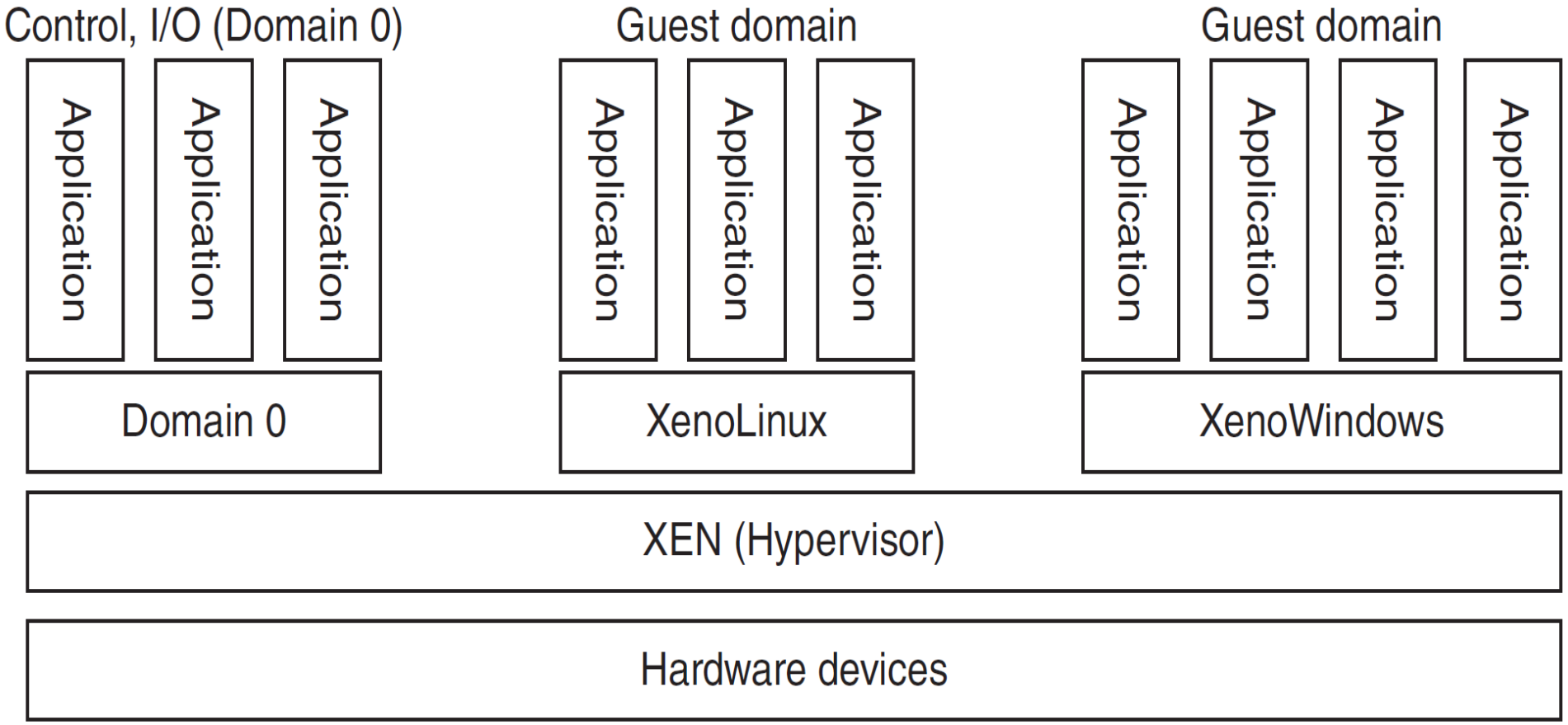
Hypervisors (cont.)

- **The Xen Hypervisor Architecture and Resources Control**
  - **The Xen is an open-source, micro-kernel hypervisor developed at Cambridge University**
  - **The Xen hypervisor implements all mechanisms**
    - Does not include any device drivers natively
    - The core components are the hypervisor, kernel, and applications
  - **The guest OS with the control ability is called Domain0**
    - The others are called DomainU
  - **Domain0 is a privileged guest OS of Xen**
    - Initially loaded when Xen boots without any file system drivers

Hypervisors (cont.)

  - Can access hardware directly and manage devices
  - One function of Domain0 is to allocate and map hardware resources to the guest domains, DomainUs
- **The Domain0 behaves like a hypervisor**
  - **Allows users to create, copy, save, read, modify, share, migrate, and rollback VMs as easily as manipulating a file**
- **The Xen is based on Linux**
  - **The security level is higher**
  - **Domain0 has the privilege of managing other VMs implemented on the same host**
  - **If the Domain0 is compromised, the hacker can control the entire system**
  - **Special security policy is applied to secure Domain0**

Hypervisors (cont.)



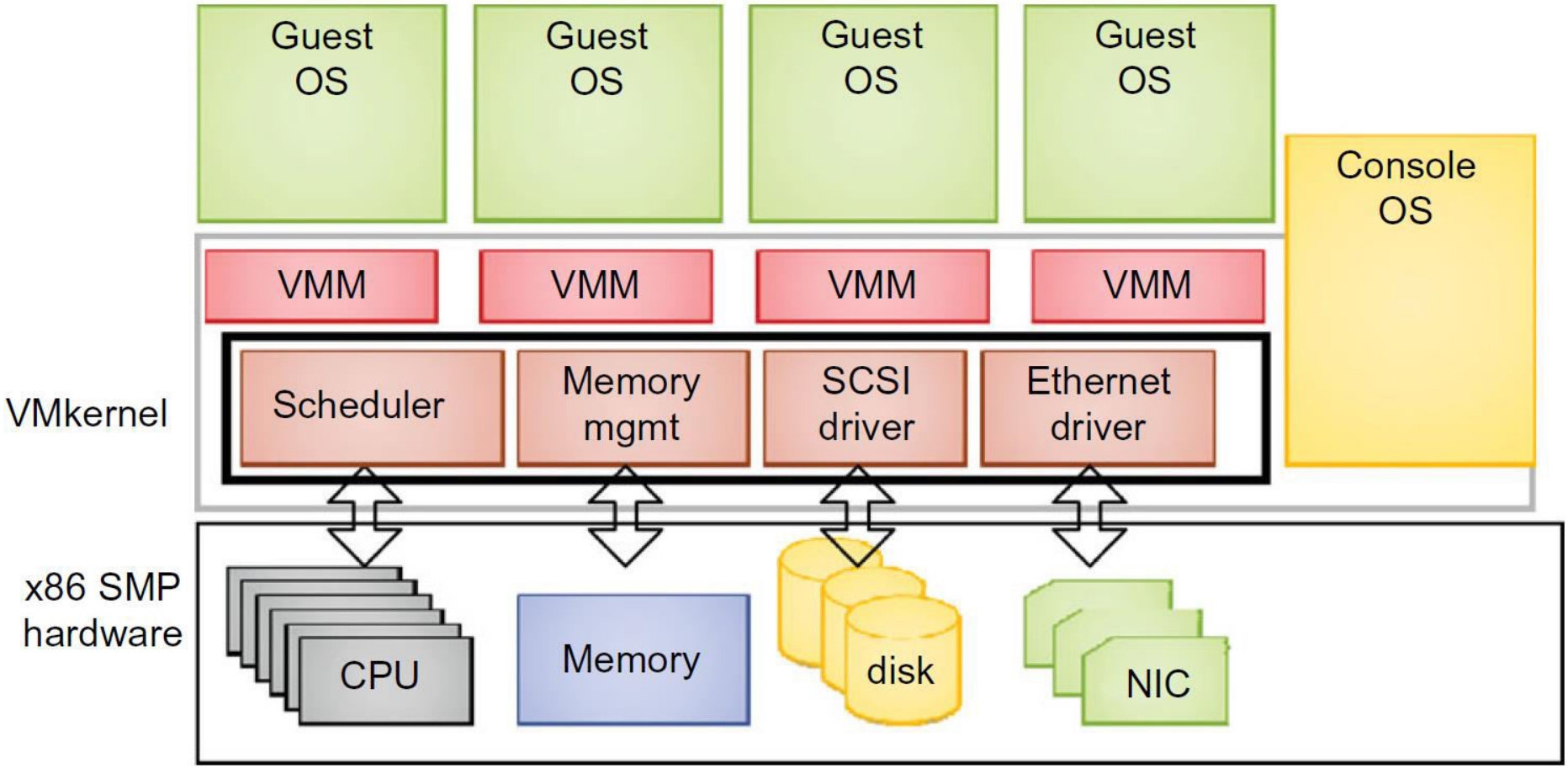
Hypervisors (cont.)

- **VMware ESX Server for Paravirtualization**
  - **The ESX is VMM or a hypervisor for bare-metal x-86 SMP servers**
    - SMP: Symmetrical MultiProcessing
    - Accesses hardware resources such as I/O directly
    - Has complete resource management control
  - **An ESX-enabled server consists of four components**
    - Virtualization layer, resource manager, hardware interface components, and service console
  - **To improve the performance, the ESX server employs the paravirtualization architecture**
    - The VM kernel interacts directly with the hardware without involving the host OS

Hypervisors (cont.)

- **Hardware interface components are device drivers and the VMware ESX Server File System**
- **The service console is responsible for booting the system**
  - Initiating the execution of the VMM layer and resource manager
  - Relinquishing control to those layers
  - Also providing some functions to facilitate the system administrators

Hypervisors (cont.)



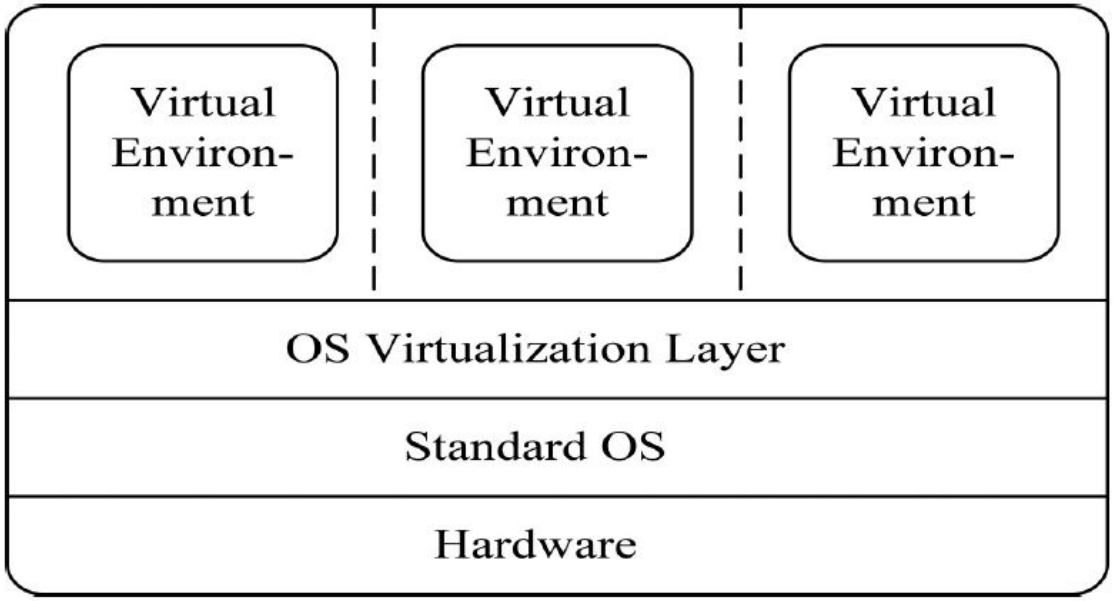
Why OS Level of Virtualization?

- **Each VM creates its own image from scratch**
  - Perhaps thousands of VMs need to be initialized simultaneously
- **The storage of the VM images is also an issue**
  - Considerable repeated content among the VM images

- **Full virtualization at the hardware level**
  - **The disadvantages of low performance and low density**
  - **Paravirtualization needs to modify guest OS**
  - **Hardware modification is needed to reduce the performance overhead of hardware-level virtualization**

Why OS Level of Virtualization? (cont.)

- Also routing tables, firewall rules, and other personal settings
- Can be customized for different people
  - But share the same OS kernel
- OS-level virtualization is also called single OS image virtualization



Benefits of OS Extensions

- **The benefits of OS-level virtualization versus hardware-level virtualization**
  - **VMs at the OS level have minimal startup/shutdown costs, low resource requirements, and high scalability**
    - Used to overcome the defect of slow initialization of VMs at the hardware level
  - **An OS-level VM and its host environment can synchronize state changes when necessary**
    - Used to overcome the defect of unawareness of the current application state
- **The two benefits can be achieved by two mechanisms of OS-level virtualization**

Virtualization on Linux or Windows Platforms

- **Most reported OS-level virtualization systems are Linux based**
  - **Virtualization support of Windows-based platforms is less frequent**
- **The Linux kernel offers an abstraction layer**
  - **To allow software processes to work with and operate on resources**
    - Without knowing the hardware details
  - **New hardware may need new Linux kernels to support it**
    - Different Linux platforms use patched kernels to provide special support for extended functionality

Virtualization on Linux or Windows Platforms (cont.)

- **Most Linux platforms are not tied to a special kernel**
  - **In such a case, a host can run several VMs simultaneously on the same hardware**
- **Examples of OS-level virtualization tools**
  - **Two OS tools: Linux VServer and OpenVZ**
    - Support Linux platforms to run other platform-based applications through virtualization
  - **The third one: Feather-weight VM (FVM)**
    - An attempt for virtualization on the Windows NT platform
- **Virtualization Support for Linux Platform**
  - **OpenVZ is an OS-level tool**

Virtualization on Linux or Windows Platforms (cont.)

- Inserts a virtualization layer inside the host OS
- Provides some OS images to create VMs quickly at the user space
- Support Linux create virtual environments for running VMs under different guest OS

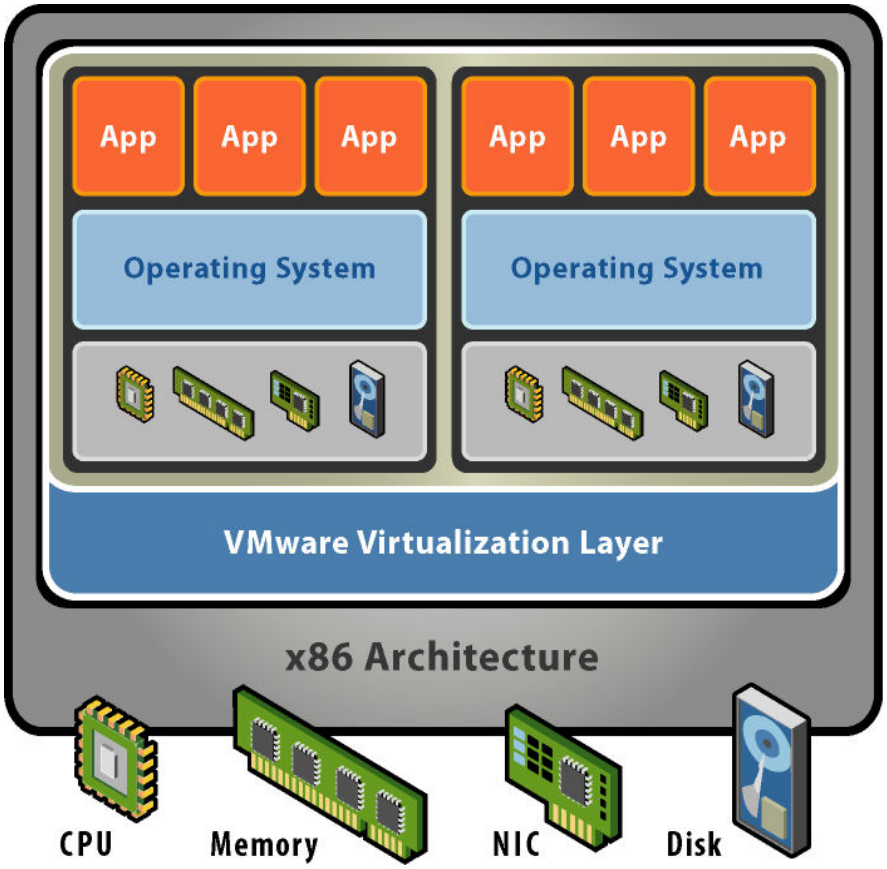
Virtualization Support and Source of Information	Brief Introduction on Functionality and Application Platforms
Linux vServer for Linux platforms (http://linux-vserver.org/)	Extending the Linux kernels to implement a security mechanism to help building VMs by setting resource limits and file attributes and changing the root environment for VM isolation, etc.
OpenVZ for Linux platforms (http://ftp.openvz.org/doc/OpenVZ-Users-Guide.pdf) (Example 3.3)	Creating virtual private servers (VPS), where the VPS has its own files, users, process tree, and virtual devices. These can be isolated from other VPSs with support of checkpointing and live migration.
FVM for virtualizing the Windows NT platforms	FVM (Feather-weight VM) uses system call interfaces to create VMs at NT kernel spaces. Multiple VMs are supported by virtualized namespaces and copy-on-write.

Vendors and Projects

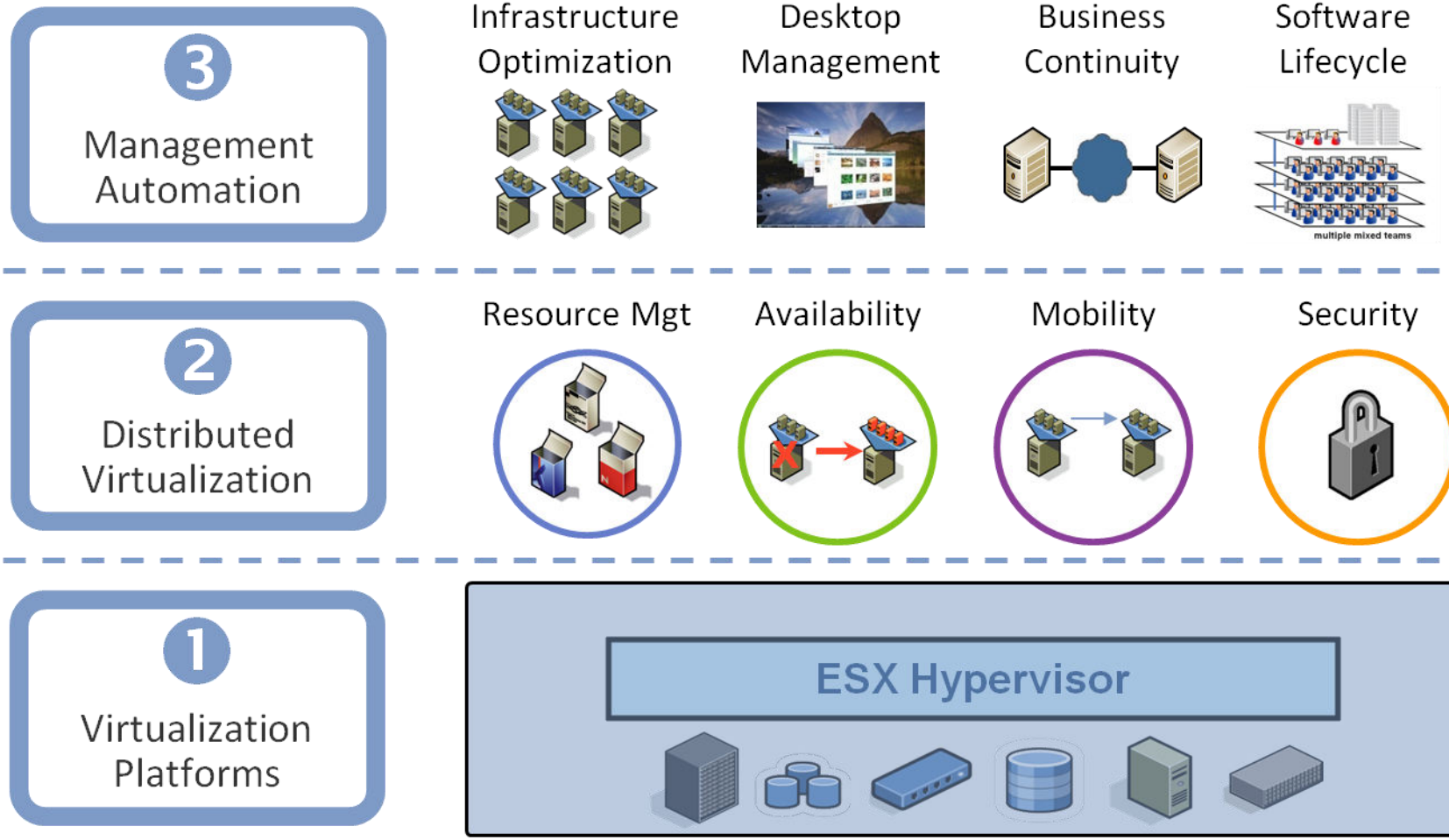
- **Virtual machine vendors :**
  - **VMware**
    - The company was founded in 1998 and is based in Palo Alto, California. The company is majority owned by EMC Corporation.
    - Implement both type-1 and type-2 VM.
  - **Xen**
    - First developed in University of Cambridge Computer Laboratory.
    - As of 2010 the Xen community develops and maintains Xen as free software, licensed under the GNU General Public License (GPLv2).
    - Implement para-virtualization.
- **Virtual machine project :**
  - **KVM (Kernel-based Virtual Machine)**
    - A Linux kernel virtualization infrastructure.
    - As of 2010, KVM supports native virtualization using Intel VT-x or AMD-V.

VMware

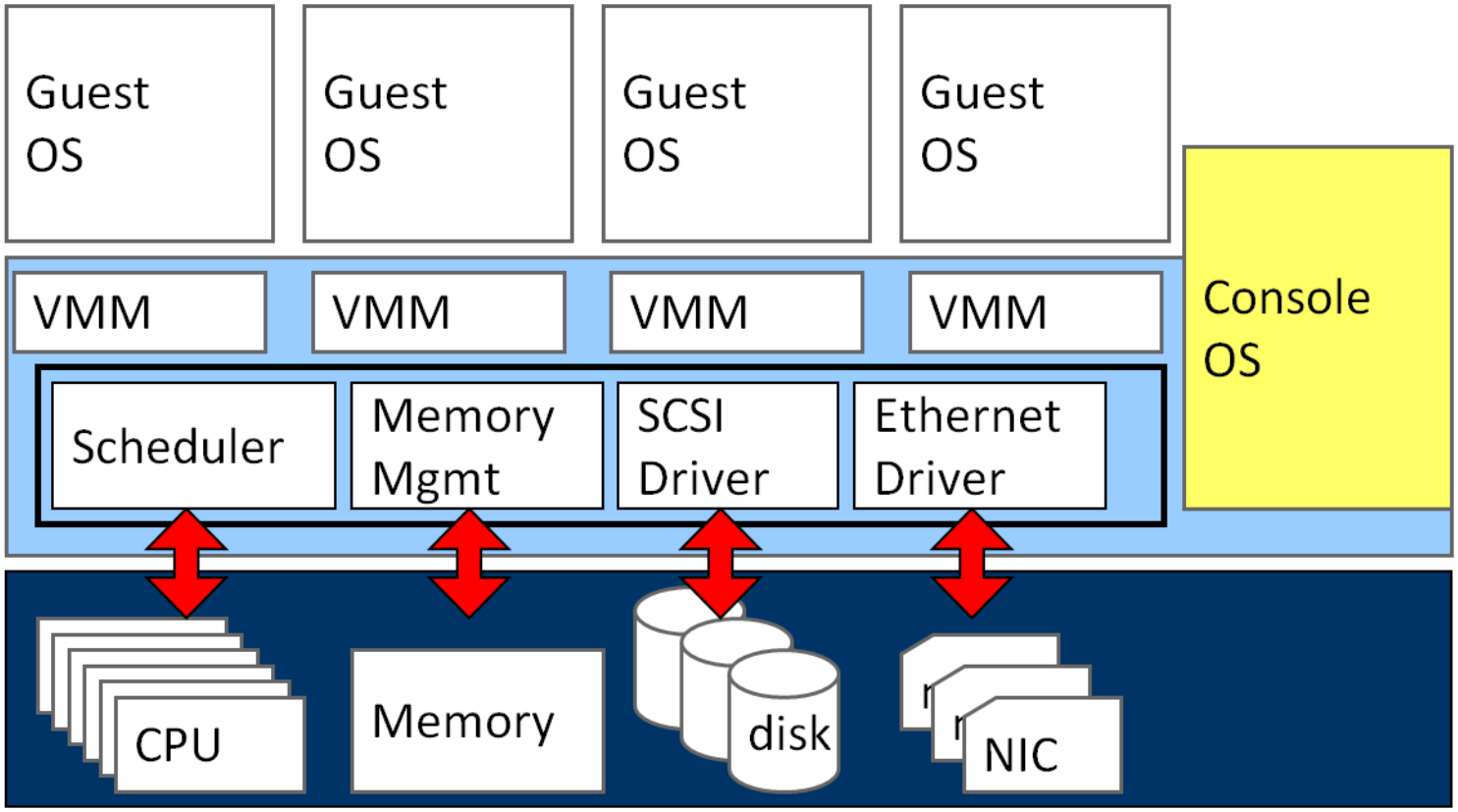
- **Basic properties :**
  - **Separate OS and hardware – break hardware dependencies**
  - **OS and Application as single unit by encapsulation**
  - **Strong fault and security isolation**
  - **Standard, HW independent environments can be provisioned anywhere**
  - **Flexibility to chose the right OS for the right application**



VMware Virtualization Stack



VMware ESX Server Architecture

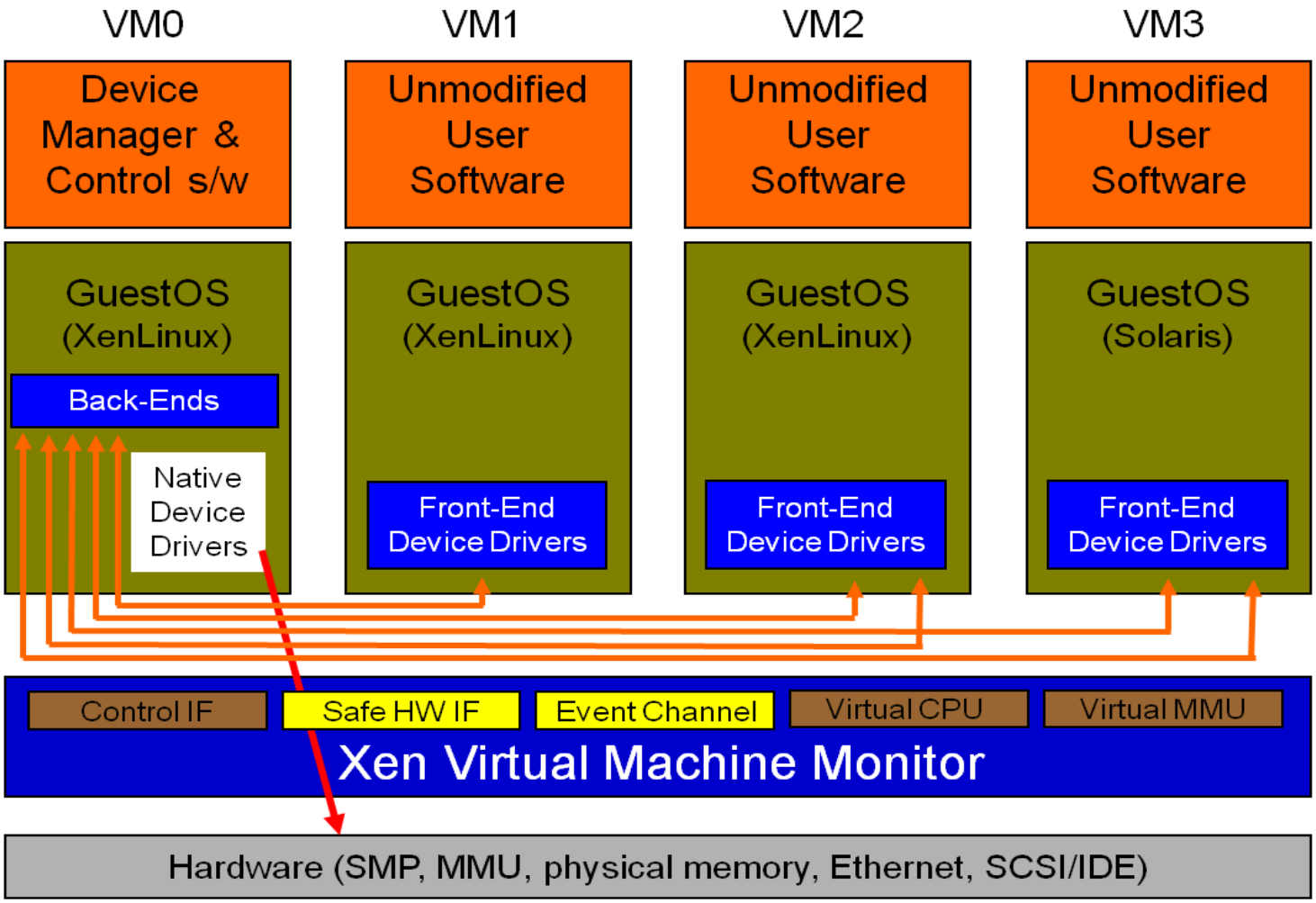


Xen

- **Basic properties :**
  - **Para-virtualization**
    - Achieve high performance even on its host architecture (x86) which has a reputation for non-cooperation with traditional virtualization techniques.
  - **Hardware assisted virtualization**
    - Both Intel and AMD have contributed modifications to Xen to support their respective Intel VT-x and AMD-V architecture extensions.
  - **Live migration**
    - The LAN iteratively copies the memory of the virtual machine to the destination without stopping its execution.

- **Implement system:**
  - **Novell's SUSE Linux Enterprise**
  - **Red Hat's RHEL**
  - **Sun Microsystems' Solaris**

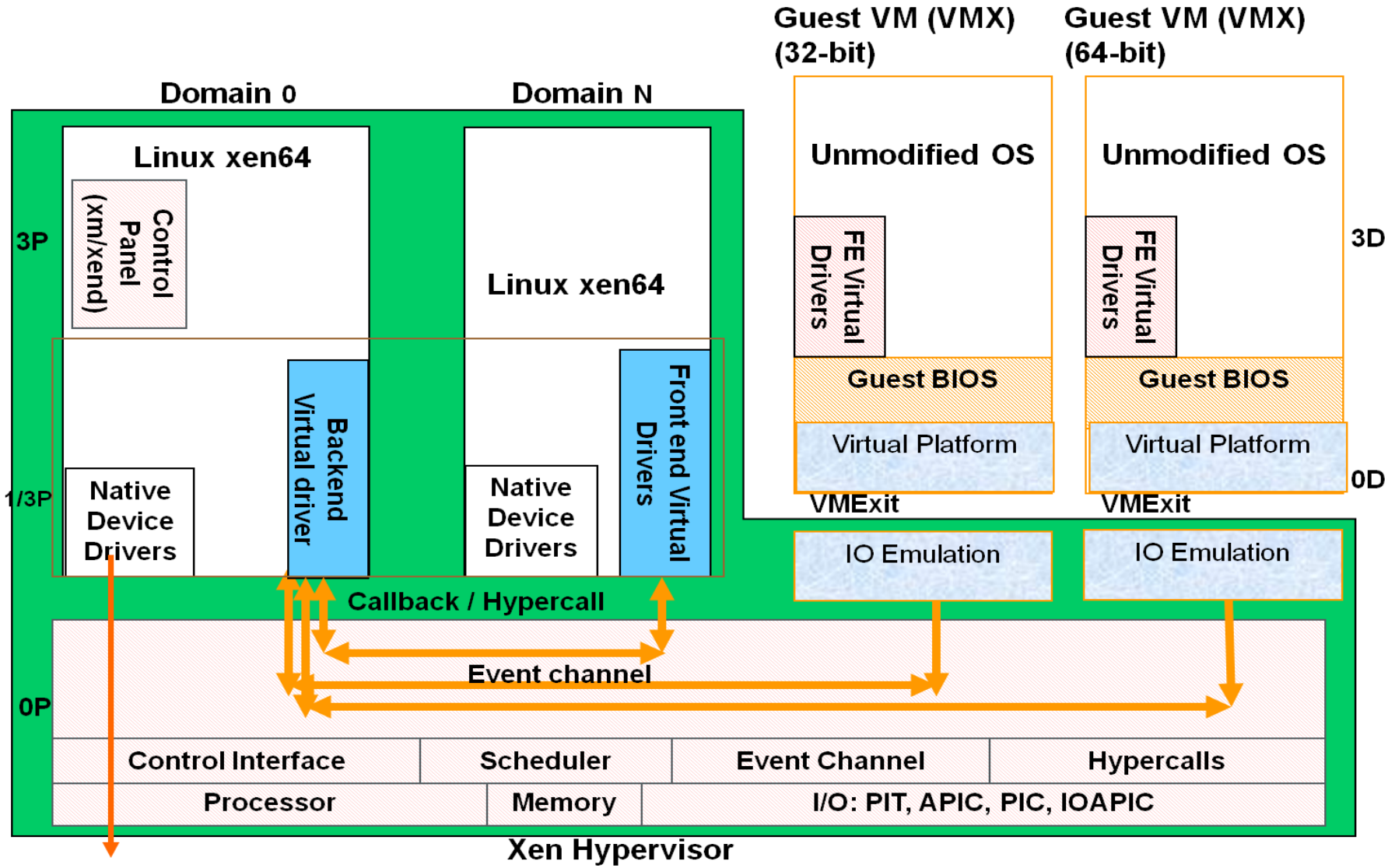
Original Xen Architecture



Hardware Assistance in Xen

- **Hardware assistance :**
  - **CPU provides certain privileged instructions**
  - **Extend page tables used to virtualize memory**
- **Xen features :**
  - **Enable Guest OS to be run without modification**
    - For example, legacy Linux and Windows
  - **Provide simple platform emulation**
    - BIOS, apic, iopaic, rtc, Net (pcnet32), IDE emulation
  - **Install para-virtualized drivers after booting for high-performance IO**
  - **Possibility for CPU and memory para-virtualization**
    - Non-invasive hypervisor hints from OS

New Xen Architecture

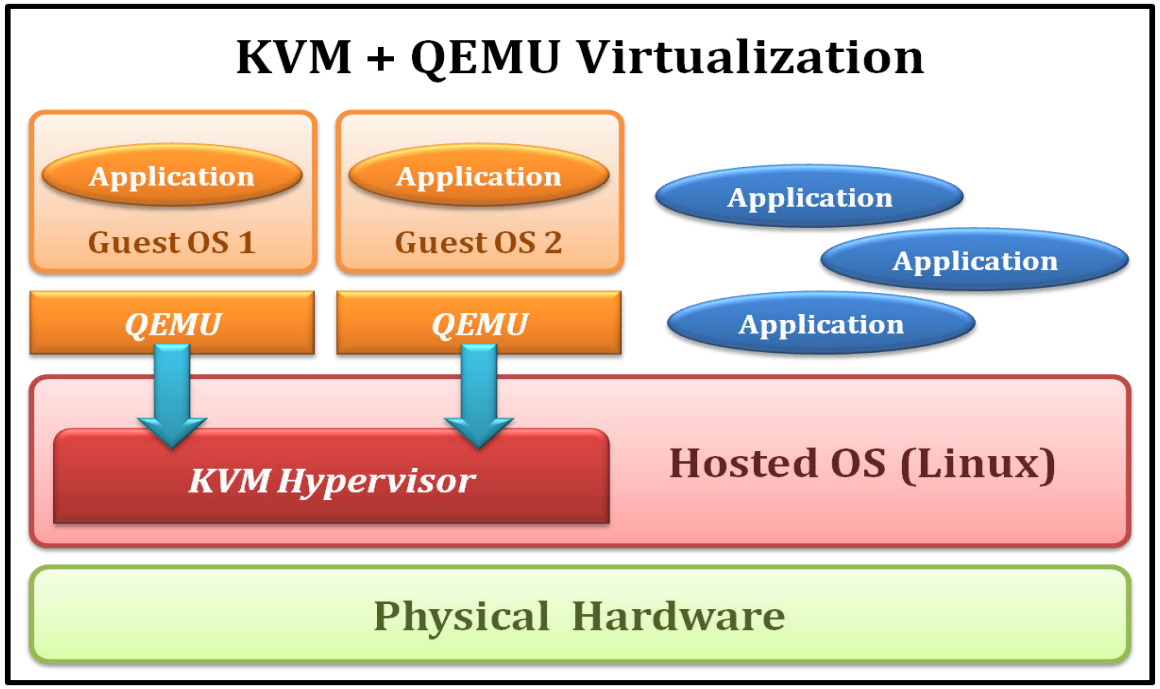


KVM

- **KVM (Kernel-based Virtual Machine)**
  - **Linux host OS**
    - The kernel component of KVM is included in mainline Linux, as of 2.6.20.
  - **Full-virtualization**
    - KVM is a full virtualization solution for Linux on x86 hardware containing virtualization extensions (Intel VT or AMD-V).
    - Using KVM, one can run multiple virtual machines running unmodified Linux or Windows images.
  - **IO device model in KVM :**
    - KVM requires a modified QEMU for IO virtualization framework.
    - Improve IO performance by **`virtio`** para-virtualization framework.

KVM Full Virtualization

- **It consists of a loadable kernel module**
  - `kvm.ko`
    - provides the core virtualization infrastructure
  - `kvm-intel.ko` / `kvm-amd.ko`
    - processor specific modules



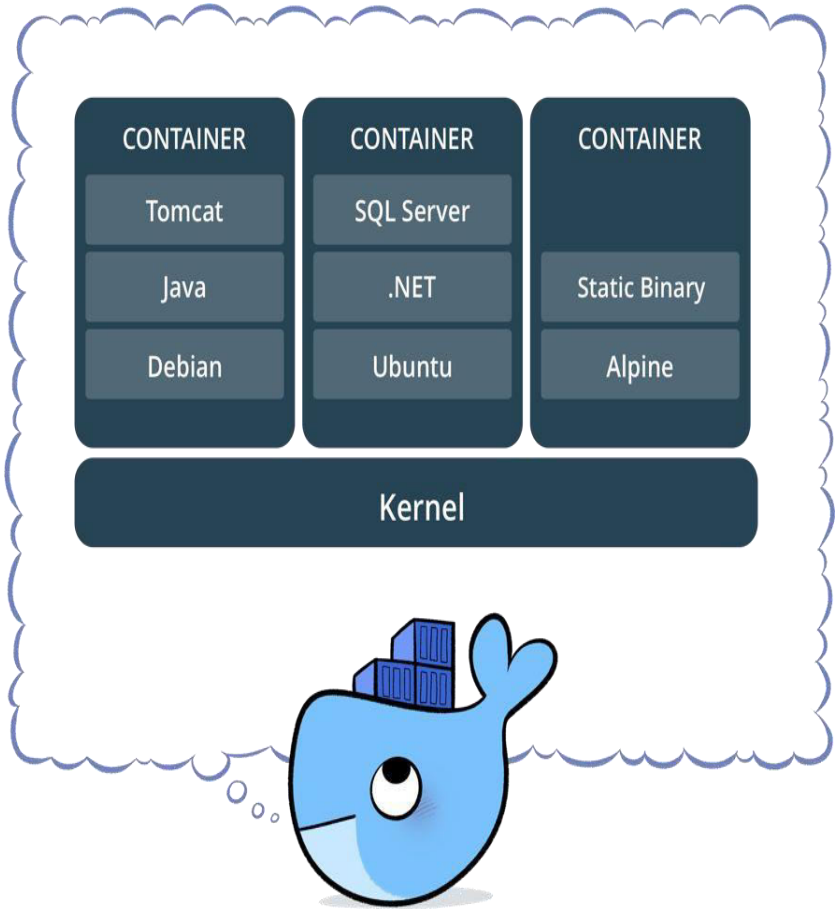
Containers – Introduction

- Containers virtualize the OS just like hypervisors virtualizes the hardware

- Containers enable any payload to be encapsulated as a lightweight, Portable self-sufficient container, that can be manipulated using standard operations and run consistently on any hardware platform.

- Wraps up a piece of software in a complete filesystem that contains everything it needs to run such as : code, runtime, system tools, libraries etc., they share the OS kernel and bins/libs where needed, otherwise each of them operate in a self contained environment.

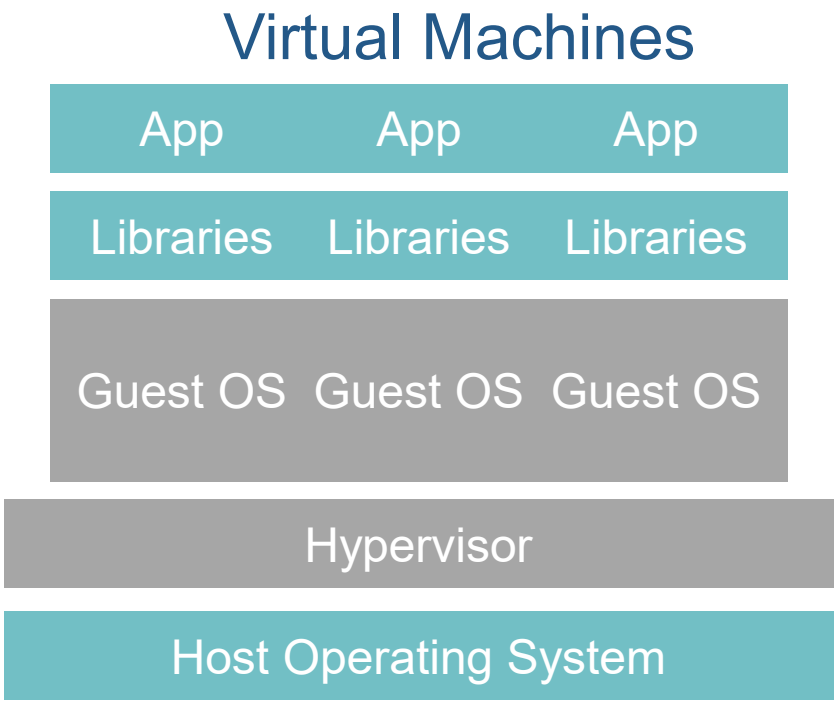
What is a container?

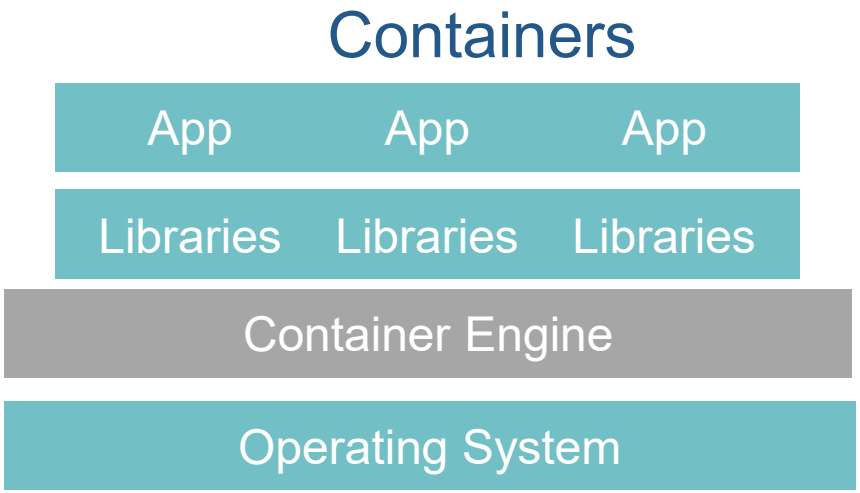


- Standardized packaging for software and dependencies
- Isolate apps from each other
- Share the same OS kernel
- Works for all major Linux distributions
- Containers native to Windows Server 2016

Containers

- **Lightweight alternative to virtual machines**
- **Smaller, less expensive, faster to start up, and self-contained**





What is Docker?

- ***Docker is an open-source project that automates the deployment of applications inside software containers, by providing an additional layer of abstraction and automation of operating system–level virtualization on Linux.***

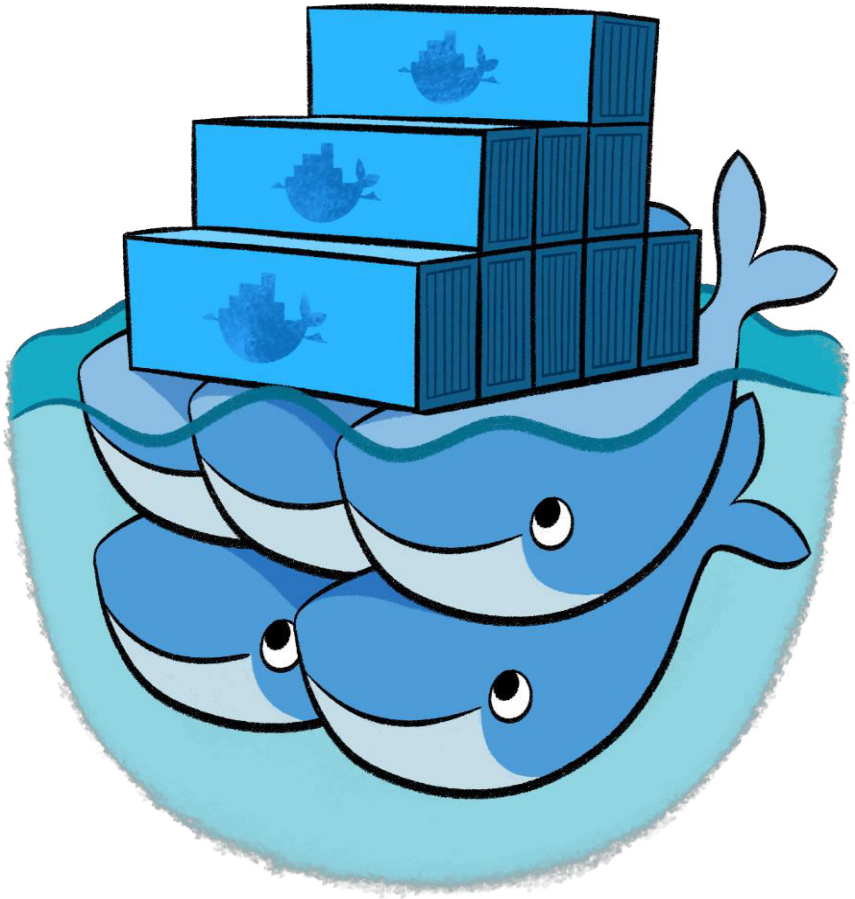
[Source: en.wikipedia.org]

Docker

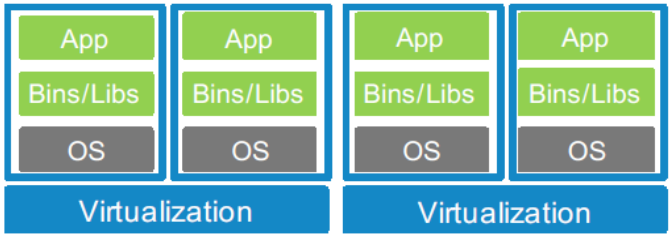
- **Leading open-source containerization platform**

  Docker containers wrap up a piece of software in a complete filesystem that contains everything it needs to run: code, runtime, system tools, system libraries – anything you can install on a server. This guarantees that it will always run the same, regardless of the environment it is running in

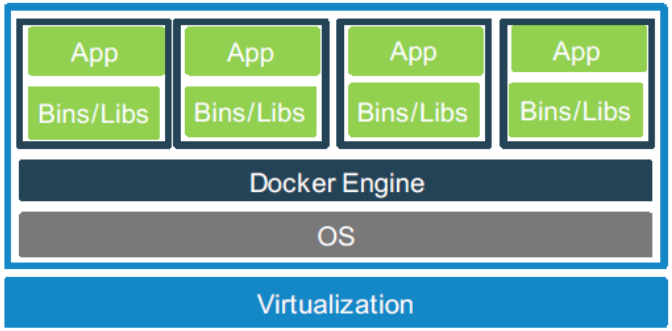
- **Supported natively in Azure**



Docker



Before: One service and OS per VM



After: One container per service.
Multiple containers running per VM

Containers and VMs together:

- **Optimize:** Added flexibility in using infrastructure resources
- **Consolidation:** Greater workload density across existing servers.
- **Reduce Costs:** Lower OS and VM licensing CapEx and reduce OpEx costs to support and maintain smaller infrastructure

Why Docker?

- **Containers are far from new;**
  - **Google has been using their own container technology for years.**
  - **Others Linux container technologies include**
    - Solaris Zones,
    - BSD jails, and
    - LXC, which have been around for many years.
- **Docker is an open-source project based on Linux containers. It uses Linux Kernel features.**

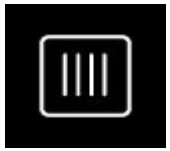
Why Docker?

- Ease of use. It allows anyone to package an application on their laptop, which in turn can run unmodified anywhere
  - The mantra is: “build once, run anywhere.”
- Speed. Docker containers are very lightweight and fast. Since containers are just sandboxed environments running on the kernel, they take up fewer resources. You can create and run a Docker container in seconds, compared to VMs which might take longer because they have to boot up a full virtual operating system every time.
- Docker Hub. Docker users also benefit from the increasingly rich ecosystem of Docker Hub, which you can think of as an “app store for Docker images.” Docker Hub has tens of thousands of public images created by the community that are readily available for use.
- Modularity and Scalability. Docker makes it easy to break out your application’s functionality into individual containers. With Docker, it’s become easier to link containers together to create your application, making it easy to scale or update components independently in the future.

Some Docker vocabulary

Docker Image

The basis of a Docker container. Represents a full application



Docker Container

The standard unit in which the application service resides and executes

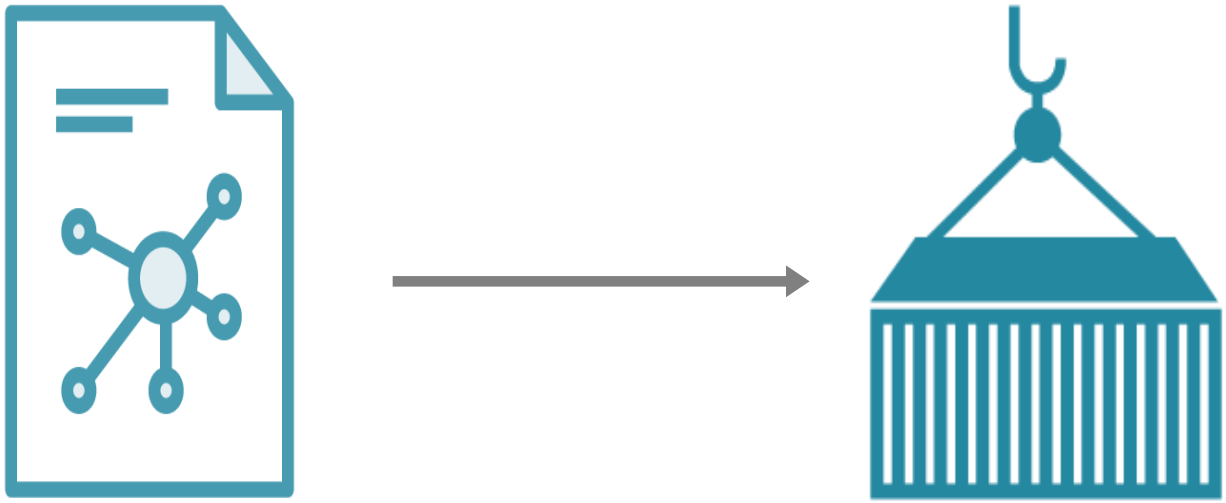
Docker Engine

Creates, ships and runs Docker containers deployable on a physical or virtual, host locally, in a datacenter or cloud service provider

Registry Service (Docker Hub(Public) or Docker Trusted Registry(Private))

Cloud or server based storage and distribution service for your images

The Role of Images and Containers



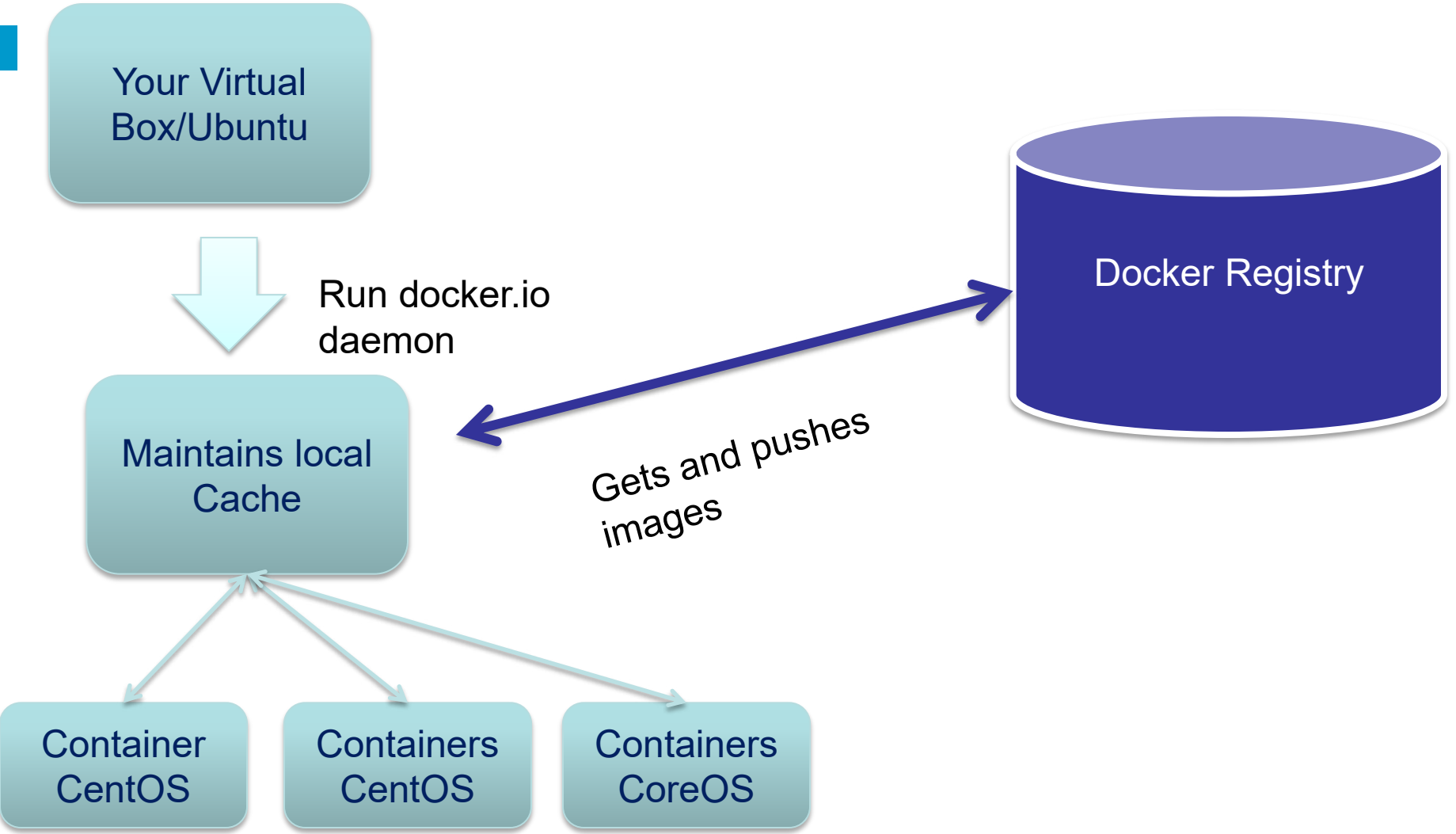
Docker Image

Example: Ubuntu with Node.js and Application Code

Docker Container

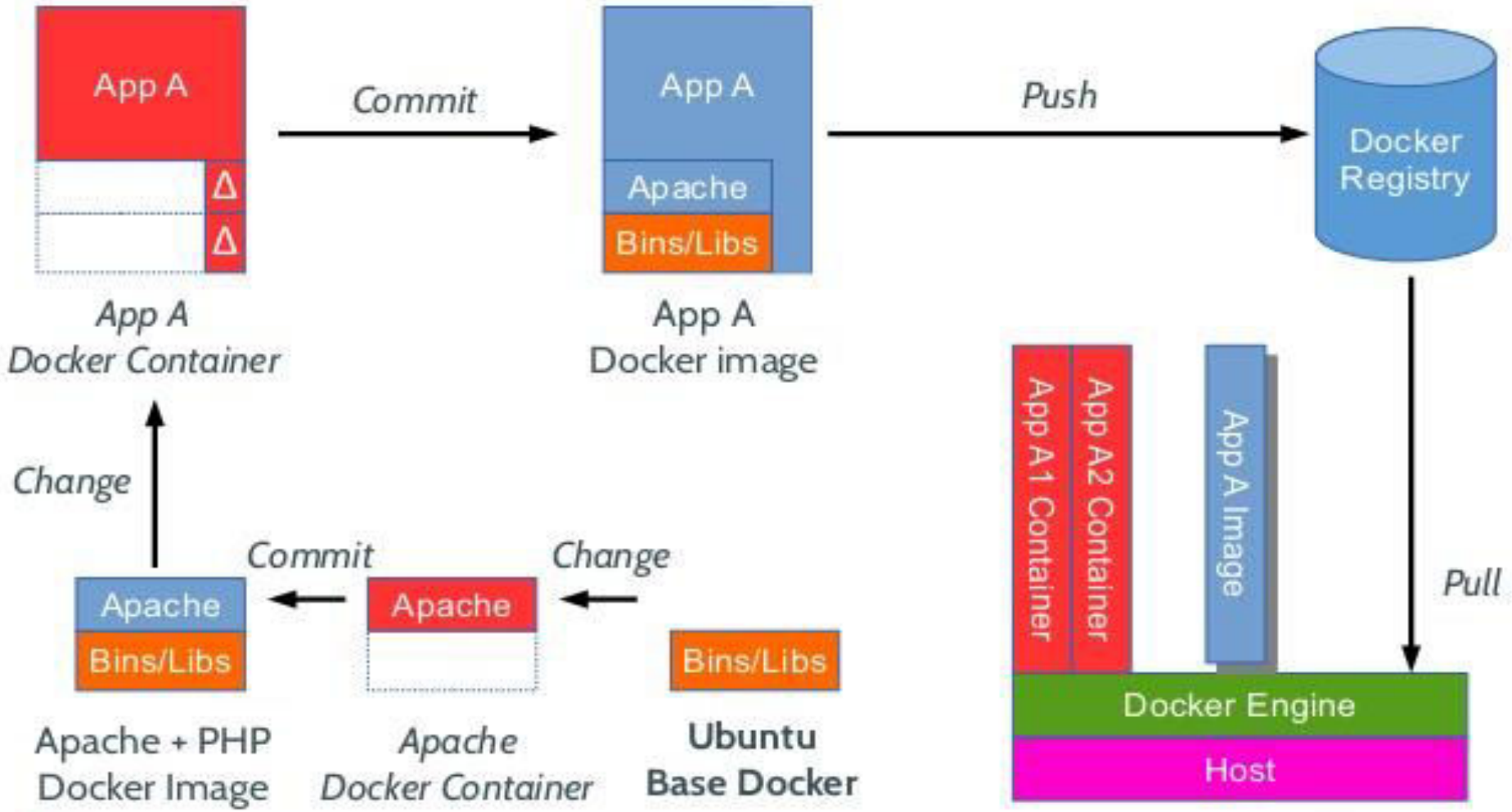
Created by using an image. Runs your application.

Understanding how the pieces work



You run copies of images from the local cache which become "containers" when they "run"

Docker life cycle



Docker Benefits

1. Local development environments can be set up that are exact replicas of a live environment/server.
2. It simplifies collaboration by allowing anyone to work on the same project with the same settings, irrespective of the local host environment.
3. Multiple development environments can be run from the same host each one having different configurations, operating systems, and software.
4. Projects can be tested on different servers.
5. It gives you instant application portability. Build, ship, and run any application as a portable container that can run almost anywhere.

Application Containers

- **Application containers can be considered as simplified VMs**
  - **The purpose is to reduce implementation costs**
    - Also to enhance the scalability and orchestration capability
- **Most data centers are built with low-cost x86 servers in a large scale**
  - **The growing interest of cloud builders and providers to switch to application containers for scalable user applications**
  - **The VMs are still useful in different types of applications**
    - May coexist for an extended period of time

Docker Engines and Docker Containers

- **Docker is an open-source project**
  - **Automates the development of user applications as software containers**
  - **The docker engine provides an additional layer of abstraction and automation of OS-level virtualization in Linux-based host platforms**
    - Written in Go language
    - Building a Docker container on top of facilities provided by the Linux kernel
  - **Docker differs from the traditional VMs**
    - Only consists of the application plus its needed binaries and library
    - Each application container takes *10* MB of memory

Docker Engines and Docker Containers (cont.)

- **Docker packages user application codes and binaries and libraries within a container**
- **Advantages in using containers**
  - Resources can be isolated
  - Services can be restricted
- **Docker engine implements a high-level API**
  - To provide lightweight containers that run software processes in isolation
  - Enables flexibility and portability regarding where the application can run, whether on the premises, public cloud, private cloud, bare metal, etc
- **The Docker virtualization concept**
- **Docker engine uses resources isolation features of the Linux kernel**

Docker Engines and Docker Containers (cont.)

- *cgroups* and kernel namespaces allow independent containers to run within separate Linux instances
- Namespace: Each process is associated with a specific view of the file system hierarchy

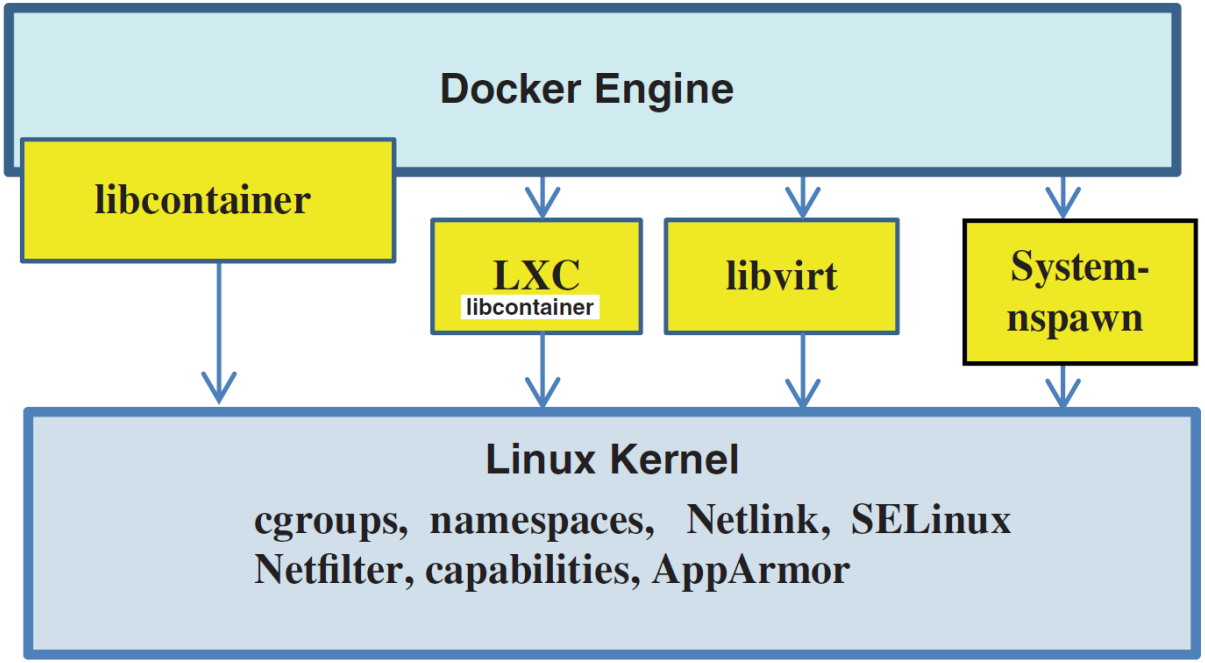
- **These isolated containers avoid the overhead of creating VMs**
  - Each can be provisioned to have a private view of the OS with its own process ID space, file system structure, and network interfaces

- **There is no guest OS needed to run the user applications**
  - **The containers apply the kernel functionalities**
    - The resource isolation is done using separate namespaces for different applications

Docker Engines and Docker Containers (cont.)

- Including CPU, memory and block I/O, and network

- **Docker applies the kernel's virtualization features directly using the *libcontainer* library**
- **Docker engine can also access kernel indirectly**
  - Through the use of interfaces: *LXC* (Linux containers), *libvirt*, or *systemd-nspawn*



Docker Engines and Docker Containers (cont.)

- **Multiple containers may share the same kernel**
  - **But each container can be constrained to use only a predefined amount of resources**
    - e.g., CPU, memory, and I/O
- **Using Docker to create and manage containers**
  - **Simplify creation of highly distributed systems**
  - **Enable multiple user applications, worker tasks, and other processes to run autonomously on a single PM or across multiple VM**
  - **This container approach scales well**

Docker Engines and Docker Containers (cont.)

- **Process to Create Application Containers in an Isolated Execution Environment**
  - **Container creation process using Docker engine**
    - The client submits the request at the user end using the software Docker
    - The daemon is a backend unit built with a Docker server and engine
    - Daemon accepts and processes client requests and manages all Docker containers
    - The server handles http requests, routing, and interfaces with the Docker engine
  - **The engine is the core of all Docker operations**
    - Handles multiple job requests concurrently
  - **The registry is used to store container images**

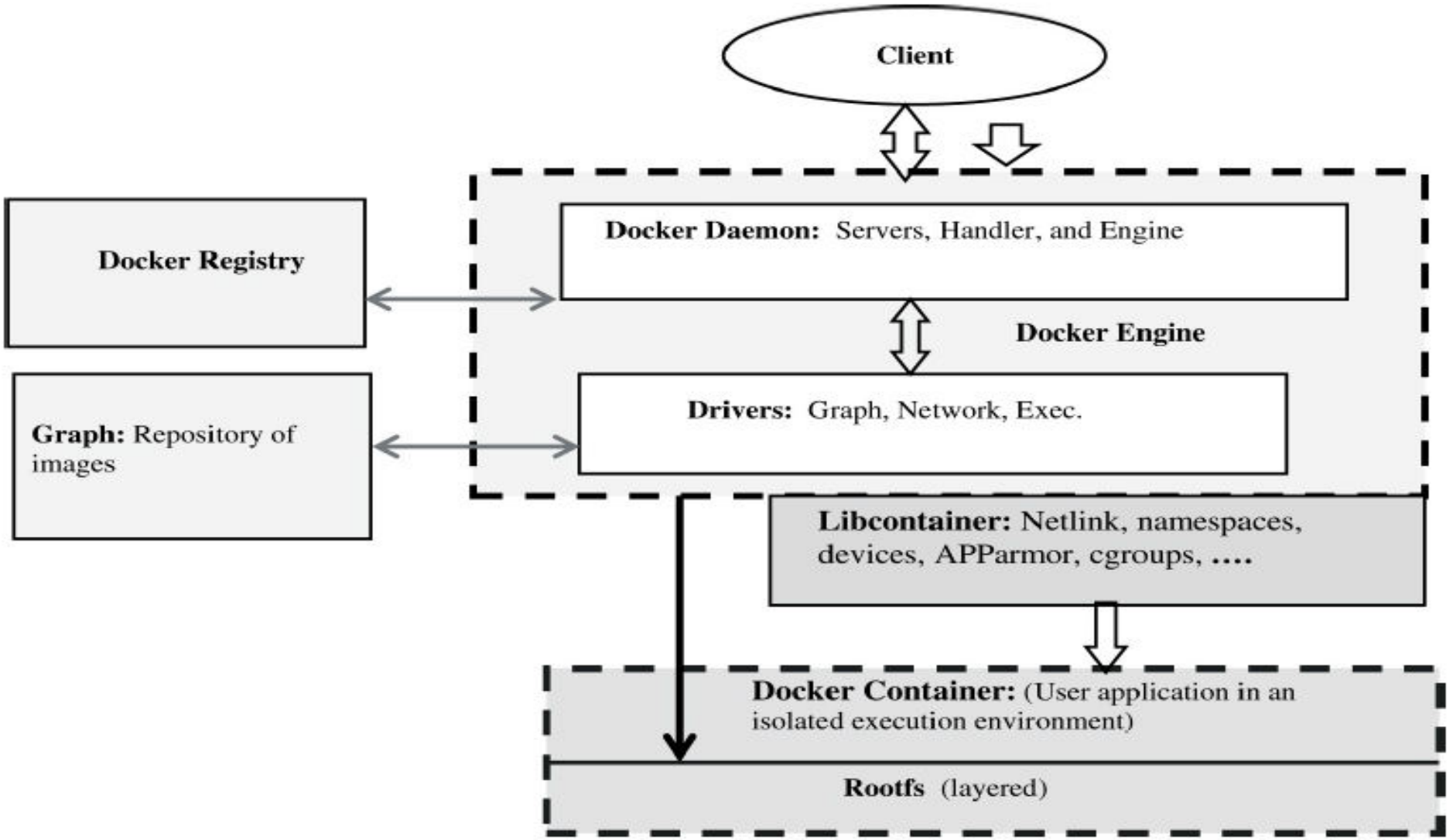
Docker Engines and Docker Containers (cont.)

- **The daemon interacts with three drivers in a Docker engine**
  - Drivers control the creation of the container execution environment
- **The *graphdriver* is a container image manager**
  - Communicates with the layered *rootfs* files associated with the containers created
  - *roofs*: root file system
- **The *networkdrive*r completes the Docker container deployment**
- ***The execudriver* is responsible for driving the container execution**
  - By working with the namespaces and *cgroups* in the *libcontainer*

Docker Engines and Docker Containers (cont.)

  - Written in Go language and used as a base to control all the containers created
- **The Docker uses daemon as a manager**
  - Uses *libcontaine*r as an executioner
- **The containers are similar in function to the VM under an isolated execution environment**
  - Created with low overhead, demands minimum memory, and is well protected with kernel isolation

Docker Engines and Docker Containers (cont.)



Docker Containers versus Virtual Machines

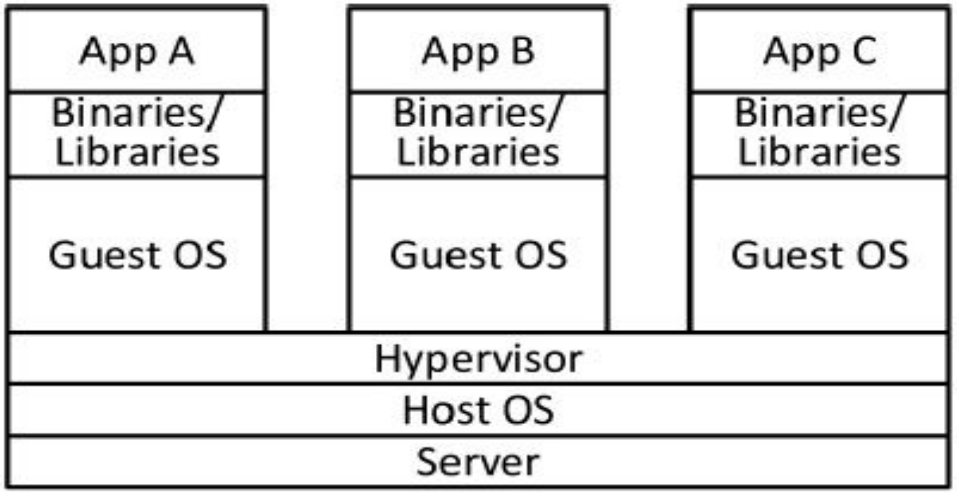
- A hypervisor-generated VM may demand tens of GB for hosting the guest OS
  - In addition to the application codes
- Docker containers are isolated to use their own binaries and libraries
  - Containers may also share binaries and libraries
  - The obvious advantage of the lightweight containers over the heavy-duty VMs
    - Do not have its own guest OS and binaries/libraries
    - May cost less to build and use containers than create and use VMs
  - Docker containers are gradually replacing traditional VMs in cloud applications

Docker Containers versus Virtual Machines (cont.)

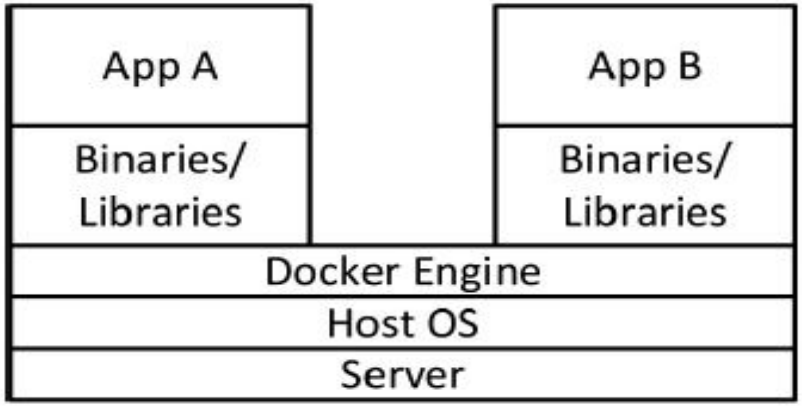
- Emphasize orchestrated massive parallelism, especially in scale-out workloads

- **The Docker engine generates lightweight virtualized containers on the Linux platform**
  - **The system is essentially a container generation and management engine**
    - The Docker source code is small to fit most computers
  - **For the clients, Docker assumes a client/server architecture**
  - **A container can be booted and ready for application in *500*ms**
    - A hypervisor may boot in *20*s according to the OS used

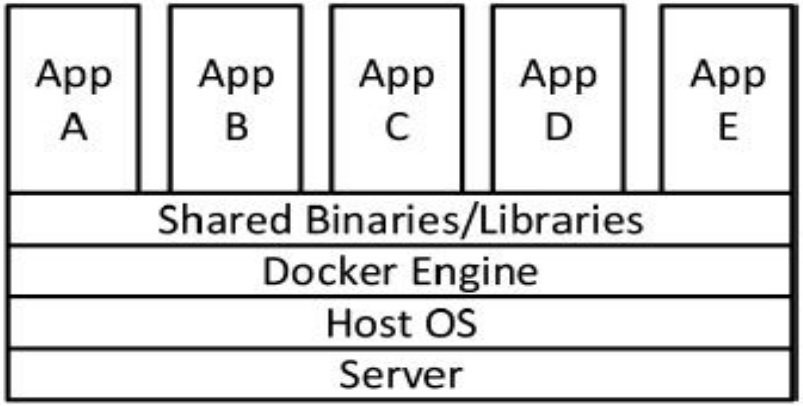
Docker Containers versus Virtual Machines (cont.)



(a) Three hypervisor-created virtual machines (VMs) on the same hardware hosts; each VM is heavily loaded with its own guest OS and specific binaries and libraries



(b) Each container is loaded with its own binaries/libraries which are not shared



(c) The Docker engine creates many lightweight app containers, which are isolated, but share OS

Docker Containers versus Virtual Machines (cont.)

- The lightweight containers are suitable for scalable use with multiple copies for cloud orchestration
  - i.e., Containers act in favor of running multiple copies of a single application, say MySQL
- The hypervisors are often more suitable for heavy-duty applications
  - With limited cloud orchestration demand
  - If flexibility of running multiple applications is wanted, use VMs
- Hypervisor-created VMs compared with Docker containers

Docker Containers versus Virtual Machines (cont.)

- **May cost less to build and use containers than create and use the VMs**
- **e.g., Amazon Web Service Elastic Container 2 (AWS EC2) has already offered the Elastic Container Service (ECS)**
  - Offers users containers to implement applications with significantly lowered memory demand and complexity

- **Using containers for building highly distributed systems**
  - **Can simplify the creation, security, and management issues significantly**