Week 2 Lecture 1

0 mins

Overview of Virtualization

Dr. Rajiv Misra, Professor
Dept. of Computer Science & Engineering
Indian Institute of Technology Patna
rajivm@iitp.ac.in

Contents

Review: Computer Organisation

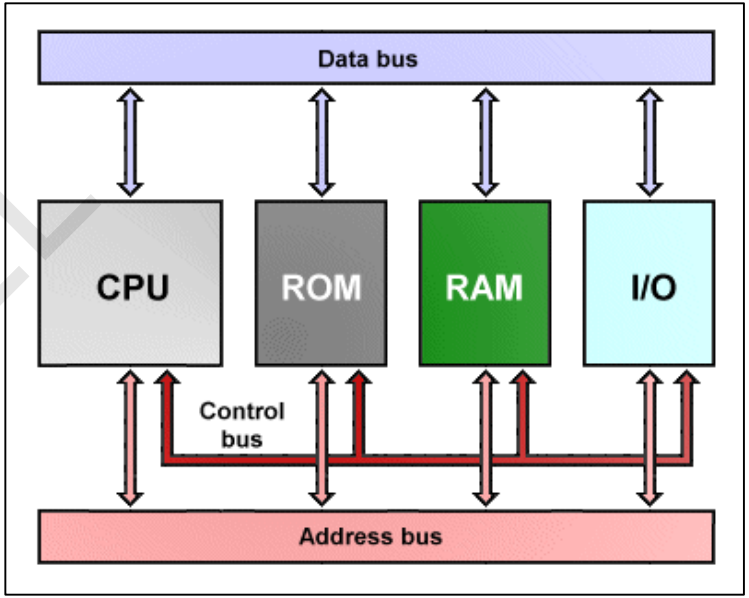
The major components in computer organization include:

Central Processing Unit (CPU): considered the *brain* of the computer, executes instructions stored in memory.

Memory: used to store data and instructions that the CPU can quickly access. RAM (Random Access Memory) for temporary storage and ROM (Read-Only Memory) for permanent storage of essential instructions.

IO Devices: Input, Output and various Peripheral devices that allow users to input data and commands into the computer and present the results of processing to the user.

Bus System: consists of buses (data, address, and control buses) that facilitate communication between different components of the computer, such as the CPU, memory, and peripherals.



Virtualization Overview

Virtualization: refers to the abstraction of physical components into a logical (virtual) objects.

- Components like processing elements (PEs), storage, memory, networking, OS can be virtualized
- An umbrella term for technologies & concepts to provide abstract environments containing virtual computing resources including virtual hardware, OS etc., to run applications.
- A *virtual machine* is an isolated duplicate of the real physical machine.

Virtualization was initially developed during the mainframe era. The IBM CP/CMS mainframes were the first systems to introduce the concept of hardware virtualization and hypervisors. These systems, able to run multiple operating systems at the same time, provided a backward-compatible environment that allowed customers to run previous versions of their applications.

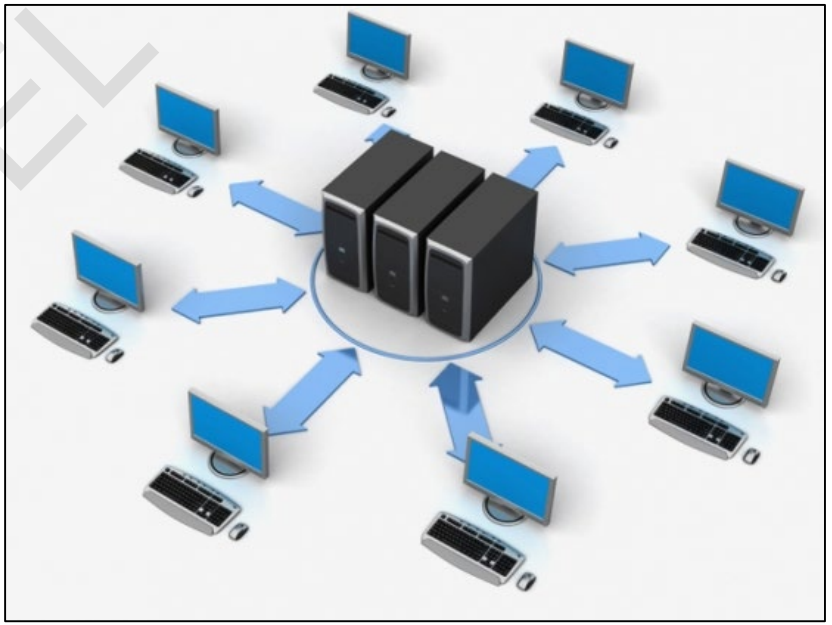
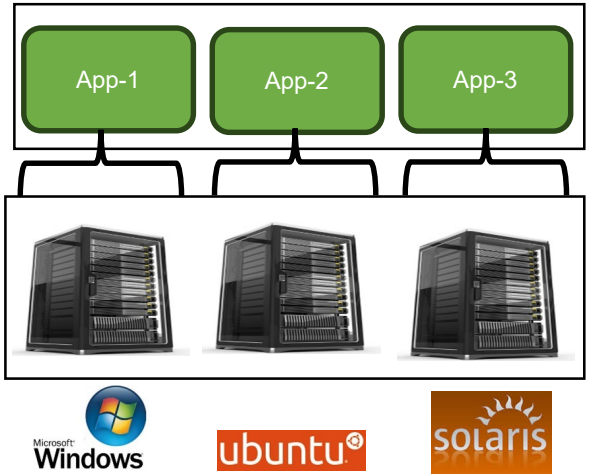


Figure Sources: https://ophtek.com

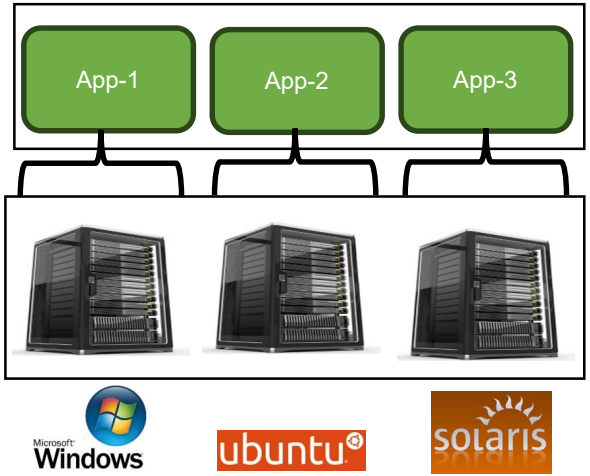
Problems with Non-Virtualized System



Some of the problems with non-virtualized (physical) systems are:

- **Resource Underutilization:** Non-virtualized systems often lead to inefficient use of resources. *Dedicated* servers may not fully utilize their processing power, memory, and storage capacity because they are dedicated to specific applications or tasks.
- **Isolation and Security:** Without virtualization, different applications or workloads run on separate physical servers. This lack of isolation can pose security risks, as a vulnerability in one application or workload may affect others on the same server.
- **Hardware Dependency:** Non-virtualized systems are tightly coupled with specific hardware configurations. This dependency can make it difficult to migrate applications between different hardware platforms without significant effort.

Problems with Non-Virtualized System



Limited Flexibility: The lack of virtualization limits the flexibility to allocate and reallocate resources dynamically. This is especially problematic in environments where workloads have varying resource requirements over time.

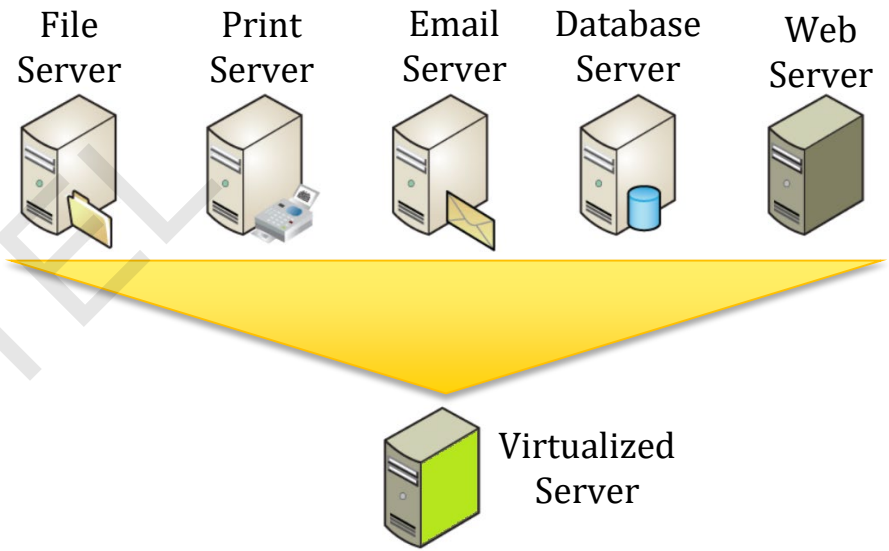
High Infrastructure Costs: Maintaining separate physical servers for each application or service can result in higher infrastructure costs. Each server requires its own set of hardware, which may be underutilized much of the time.

Difficult Disaster Recovery: Disaster recovery in non-virtualized environments can be complex and time-consuming. Creating backups and restoring services often involves lengthy manual processes and downtime.

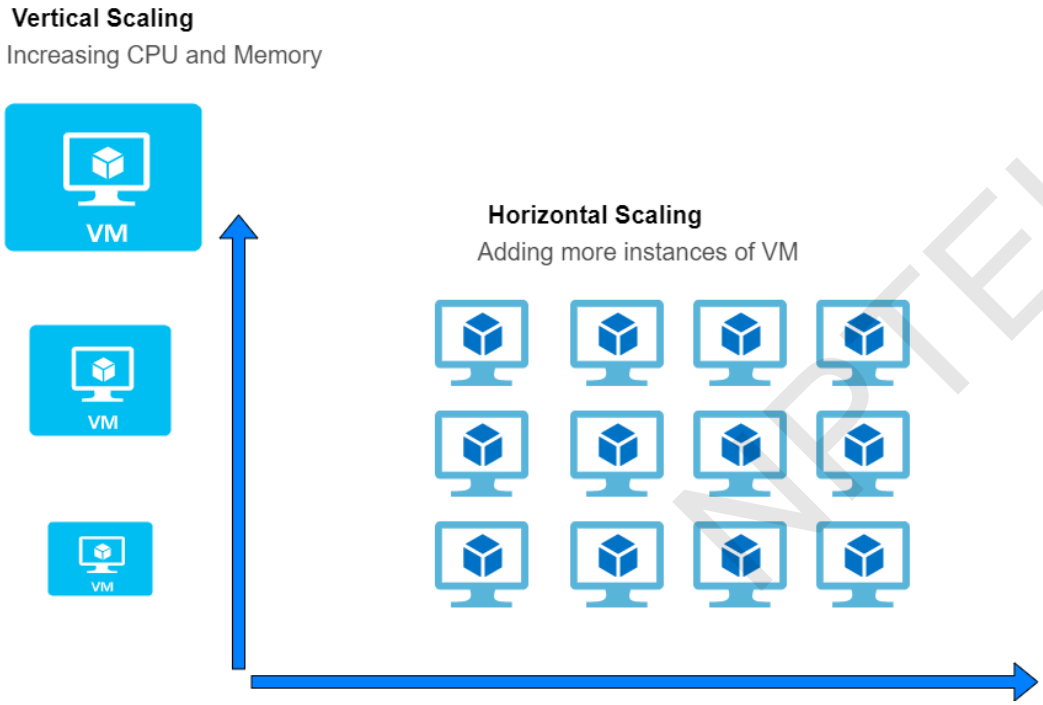
Limited Mobility: Applications in non-virtualized environments are often tied to specific physical servers. This lack of mobility makes it challenging to move applications between servers for load balancing or hardware maintenance.

Benefits of Virtualization

- **Consolidation:**
- Virtualization allows multiple virtual machines (VMs) to run on a single physical server, optimizing hardware utilization and reducing the number of physical servers needed.
- With server consolidation, organizations can achieve **cost savings** by requiring fewer physical servers and associated hardware components.
- Virtualization enables dynamic allocation of resources, allowing for better **utilization** of CPU, memory, and storage based on workload demands.
- Fewer physical servers result in lower power and cooling requirements, leading to **reduced energy costs**.



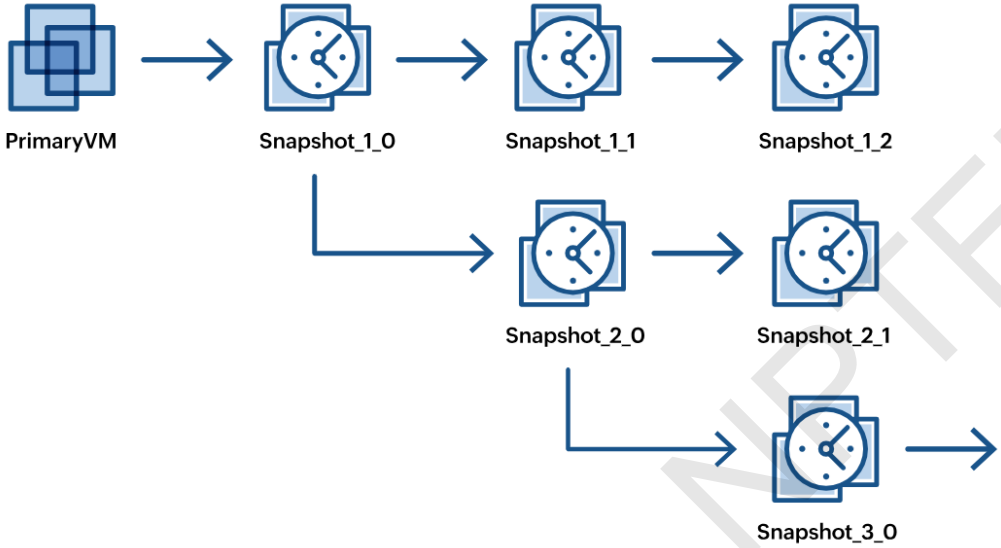
Benefits of Virtualization



- **Elasticity and Scalability:**
- Virtualization simplifies the process of scaling IT resources by adding or removing virtual machines as needed, providing flexibility to meet changing business requirements.
- Virtual machines can be **provisioned dynamically and rapidly,** allowing for faster deployment of new applications or services.

Figure Sources: https://www.site24x7.com/vmware-scaling.html

Benefits of Virtualization



- **Enhanced Reliability:**
- **Snapshot and Cloning**: Virtualization facilitates the creation of snapshots and clones of virtual machines, simplifying backup and recovery processes.
- **High Availability:** Virtualization technologies often include features like live migration and automatic failover, ensuring minimal downtime in the event of hardware failures.

Figure Sources: https://www.site24x7.com/vmware-snapshot-monitoring.html

Benefits of Virtualization

- **Isolation:**

- **Isolation of Workloads:** Virtual machines are isolated from each other, providing a level of security and preventing the impact of one workload on others.

- Virtualization enables the creation of isolated test environments that closely mimic production systems, facilitating efficient development and testing processes and reducing the risk of conflicts with production systems.

Figure Sources: https://www.site24x7.com/vmware-isolation.html

Virtualization

- To summarize, virtualization involves the abstraction and partitioning of physical resources into virtual resources, enabling greater flexibility, resource optimization, and efficient management of infrastructure.

- The **virtualization layer**, often facilitated by *hypervisors* or *virtualization platforms*, manages the allocation and operation of these virtual resources on the underlying physical hardware.

Virtualization in Conventional Infrastructure

Some examples of physical resources and their corresponding virtual resources in virtualization:

Physical Resource: Server (Compute) **Virtual Resource: Virtual Machine (VM)**

Physical servers are abstracted into virtual machines, each running its own operating system and applications. Hypervisors or virtual machine monitors (VMMs) manage the execution of these VMs on the physical server.

Physical Resource: Memory (RAM) **Virtual Resource: Virtual Memory**

Physical RAM is partitioned and assigned to virtual machines as virtual memory. Each VM believes it has its dedicated portion of memory, even though it's sharing the physical RAM with other VMs.

Physical Resource: Storage **Virtual Resource: Virtual Disk**

Physical storage devices (hard drives, SSDs) are abstracted into virtual disks. These virtual disks are then assigned to virtual machines, allowing them to store their operating system, applications, and data.

Physical Resource: Firewall **Virtual Resource: Virtual Firewall**

Physical firewalls can be virtualized into virtual firewalls, allowing for the segmentation and control of network traffic within a virtualized environment.

Virtualization in Conventional Infrastructure

Some examples of physical resources and their corresponding virtual resources in virtualization:

Physical Resource: Switch and Router **Virtual Resource: Virtual Switch and Virtual Router**

Physical network switches and routers can be virtualized into software-based counterparts, providing network connectivity and routing for virtual machines within a virtualized environment.

Physical Resource: CPU (Processor) **Virtual Resource: Virtual CPU (vCPU)**

Physical CPUs are divided into virtual CPUs, each assigned to a virtual machine. The hypervisor manages the allocation of CPU resources among VMs, allowing them to share the processing power of the physical CPU.

Physical Resource: Graphics Processing Unit (GPU) **Virtual Resource: Virtual GPU (vGPU)**

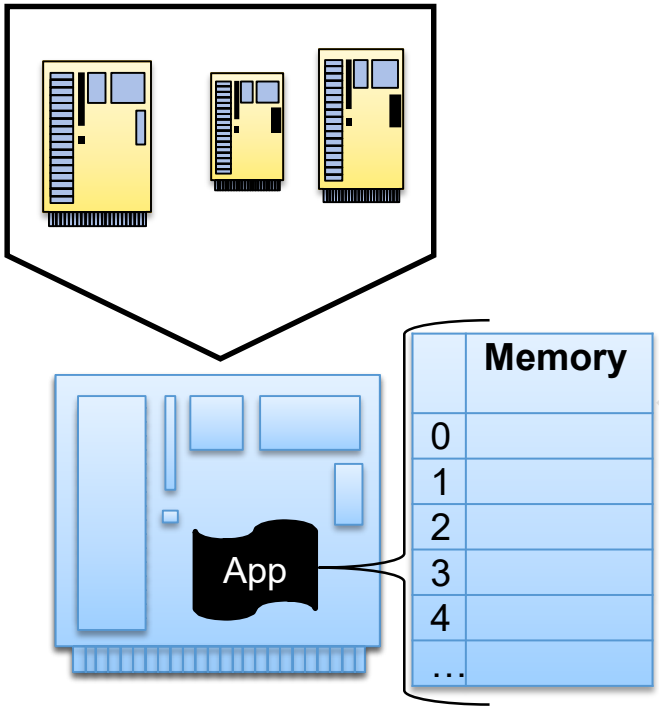
In environments where graphical processing is important (e.g., virtual desktops or GPU-intensive applications), physical GPUs can be virtualized into vGPUs, which are then allocated to virtual machines.

Physical Resource: Load Balancer **Virtual Resource: Virtual Load Balancer**

Physical load balancers can be virtualized into software-based virtual load balancers. These virtual load balancers distribute network or application traffic across multiple virtual servers or services.

Why is Virtualization needed at Edge and Cloud?

Mainly Consolidation: Replace several machines with (a bigger) one without changing any software.



- **Save the cost** (space, energy, personnel) of running several machines
- **Efficiency**: Use the (otherwise wasted) CPU power
- **Availability:** *Clone* servers at low cost
- **Migration:** *Migrate* a machine at low cost (for e.g., load-balancing)
- **Security:** *Isolate* an appliance—a server for a specific purpose (such as security)—without buying new hardware

Each virtualized application thinks that it has

- Large contiguous memory, starting from address 0
- CPU power to itself
- All I/O devices

Virtualization Reference Model

- In a *virtualized environment*, there are three major components:

Guest
The **guest** represents the system component that interacts with the virtualization layer rather than with the host directly.

Virtualization Layer
The **virtualization layer** is responsible for recreating the environment where the guest will operate.

Host
The **host** represents the original environment where the guest is supposed to be managed.

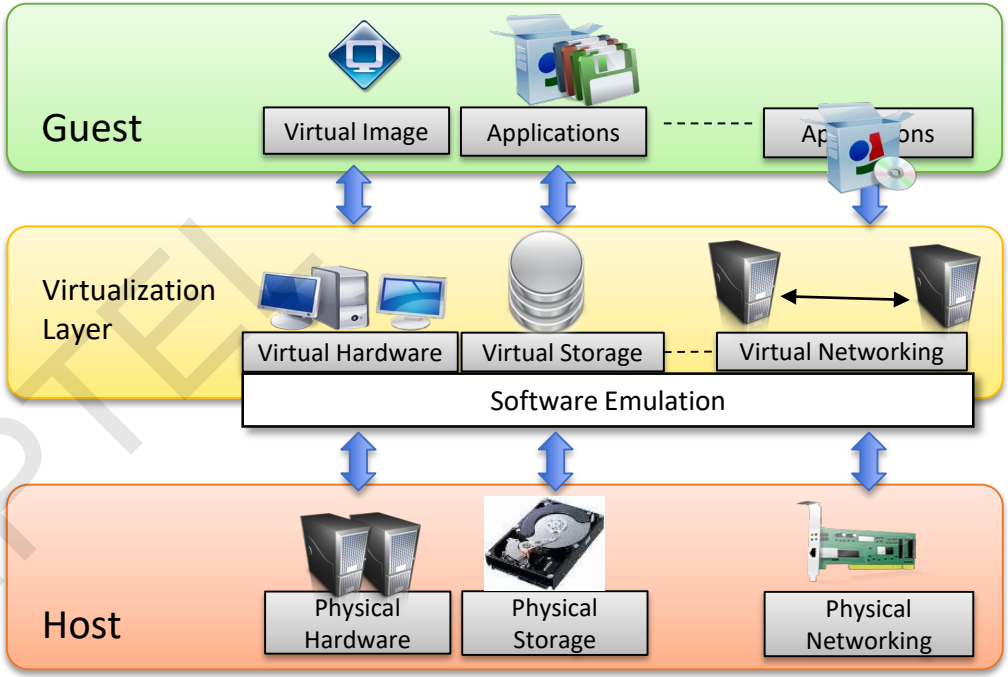
Virtualization Reference Model

- e.g., ***hardware virtualization***;

- **Guest**
- represented by a system image comprising an operating system and installed applications
- installed on top of virtual hardware, controlled and managed by the **virtualization layer** (virtual machine manager or VMM).

- **Host**

  is represented by the physical hardware, and in some cases the operating system where the VMM is running.



- **Virtual storage**:
  - guest (e.g., client applications) interact with the virtual storage management software deployed on top of the real storage system.
- **Virtual networking**:
  - guest interacts with a virtual network (VPN) managed by specific software (VPN client) using the physical network on the node.

Virtualization: a Technique for Multiple Applications/OS

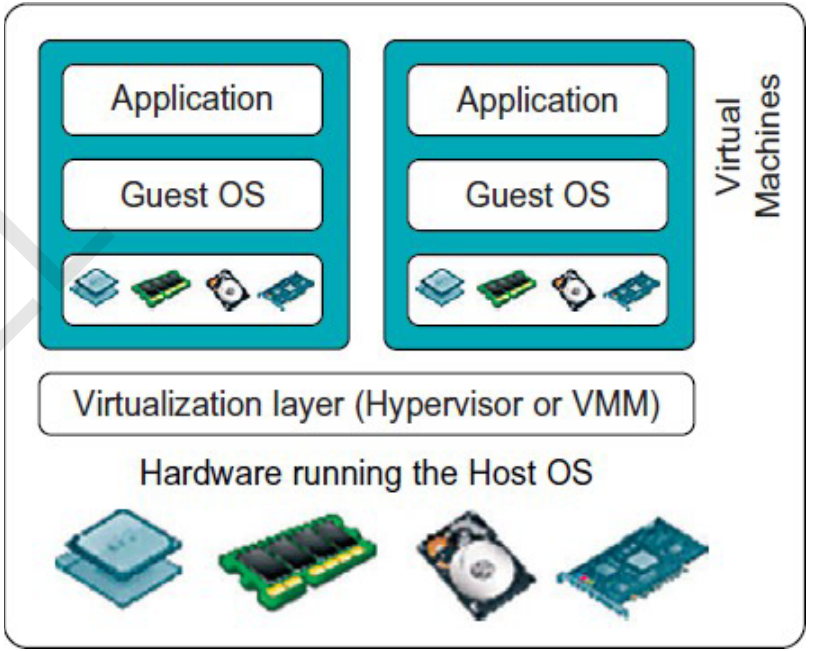
Virtualization allows *simultaneous* execution of multiple OSs (and their applications) on the same physical machine.

Virtual resources:
- each OS thinks that it *owns* hardware resources
- ***Virtual machine (VM)*** *= OS + applications + virtual resources*

Virtualization layer: management of physical hardware (performed by virtual machine monitor or hypervisor)

Virtual machine monitor (VMM):

1. Provides environment identical with the original machine
2. Programs show at worst only minor decrease in speed.
3. VMM is in complete control of system resources.
4. **Goals** – Fidelity, Performance, Safety & Isolation



Virtual Machine Lifecycle

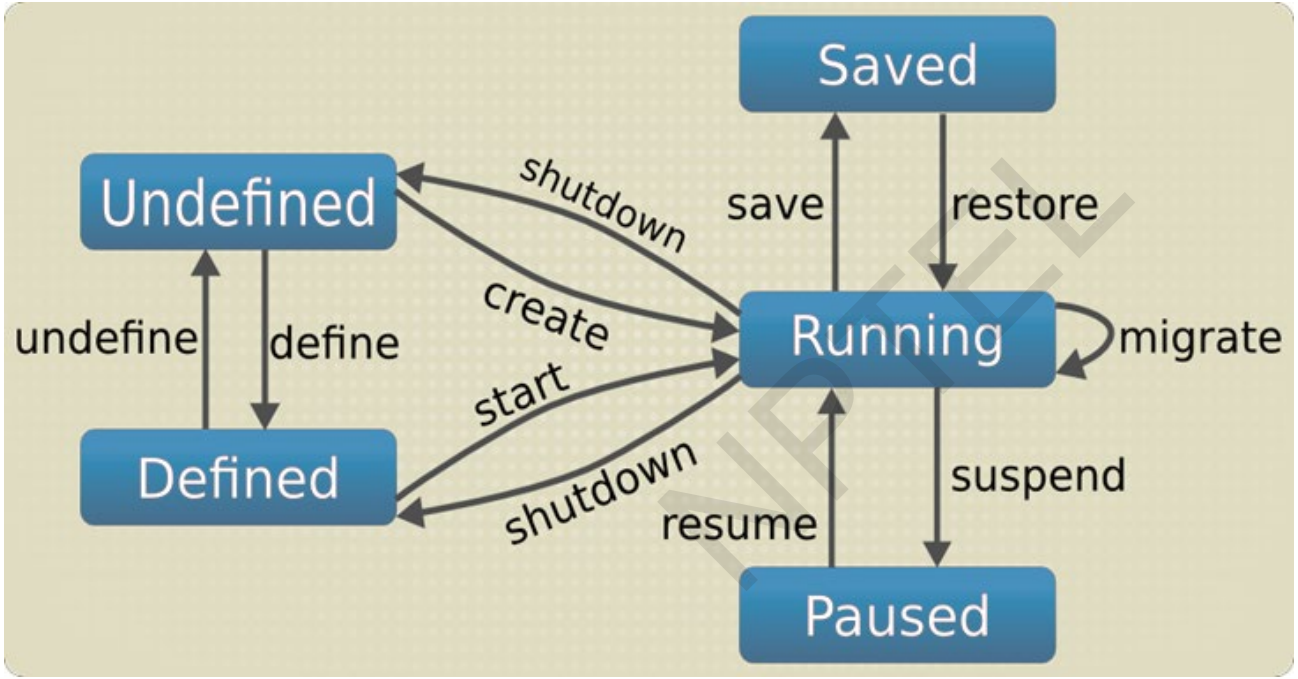


Figure Source: http://wiki.libvirt.org/page/VM_lifecycle# Virtual_Machine_Lifecycle

Virtual Machine Interaction

- The interaction with physical hardware is primarily facilitated through software, typically the operating system. The operating system serves as an intermediary between applications and the underlying hardware, managing resource allocation, scheduling processes, and providing a layer of abstraction for hardware-specific details.

- Two primary mechanisms through which software interacts with physical hardware are: **System Calls** and **Interrupts/Traps**.

- The operating system's kernel contains the core components responsible for handling system calls, interrupts, and traps.

Virtual Machine Interaction

System Calls

Applications make requests for services from the operating system through system calls. These services may include file operations, network communication, or hardware-related operations.
For e.g., An application might use a system call to open a file, request input from a keyboard
UNIX system calls e.g., - open, read, write, close, wait, exec, fork, exit, kill

Interrupts and Traps:

A **trap** is a computer-generated occurrence that results from an error or exception in the program(user process) that is running at the time.
For e.g., Division by zero, invalid memory access.

An **interrupt** is generated by the hardware
For e.g., When a hardware device, such as a network interface card, has data ready to be processed, it may trigger an interrupt, prompting the operating system to handle the incoming data.

Virtualized Infrastructure

Server Virtualization – The division of a physical system's resources (e.g. processors, memory, I/O, and storage), where each such set of resources operates independently with its own System Image instance and applications

Virtual Machines

- Proxies for physical resources and have the same external interfaces and functions
- Composed from physical resources

Virtualization Layer

- Creates virtual resources and "maps" them to physical resources
- Provides isolation between Virtual Resources
- Accomplished through a combination of software, firmware, and hardware mechanisms

Physical Hardware Resources

- Hardware components with architected interfaces / functions
- Examples: Memory, disk drives, networks, servers

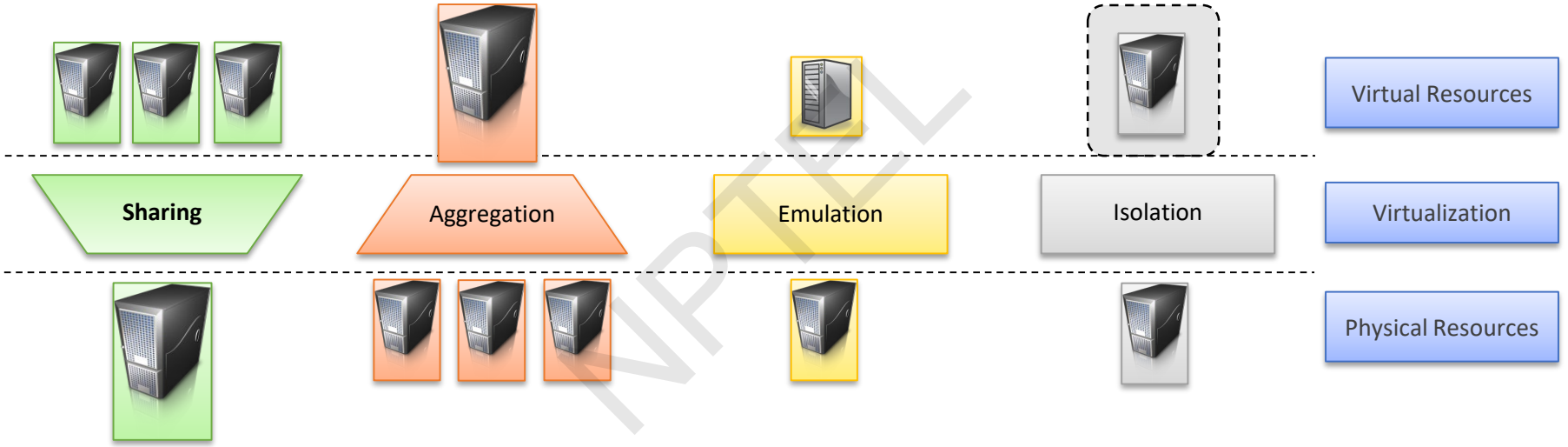
Characteristics of virtualized environments

Increased security

- For guest, a completely **transparent and controlled execution** environment is provided.
- An **emulated** environment in which the guest is executed.
- Guest operations performed on VM are translated and applied to the host.
- VMM controls and filters the guest activity, thus **preventing some harmful operations** from being performed, Host is hidden or protected from the guest.
- Sensitive information naturally hidden without installing complex security policies.
- Increased security is a requirement when dealing with **untrusted code**.
  For example, applets run in a sandboxed Java Virtual Machine(JVM), which provides them with limited access to the hosting operating system resources.
- Hardware virtualization solutions (VMware Desktop, Virtual Box, and Parallels) create a virtual computer with customized virtual hardware on top of which a new operating system can be installed. By default, the file system exposed by the virtual computer is **completely separated** from the one of the host machine.

Characteristics of virtualized environments

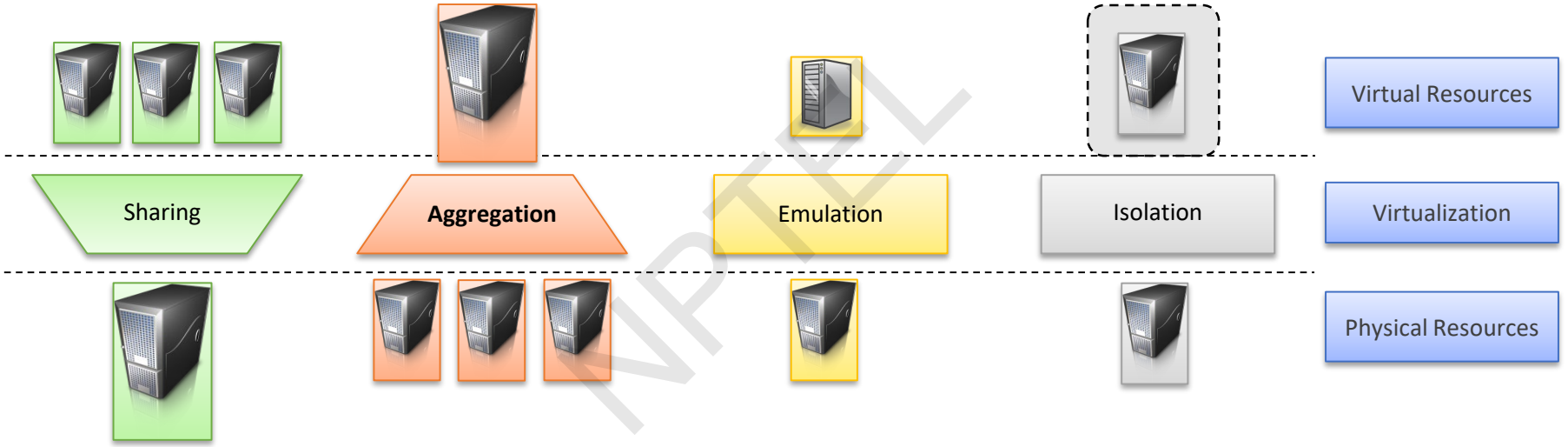
Managed execution



Sharing - in virtualized data centers to reduce the number of active servers and limit power consumption

Characteristics of virtualized environments

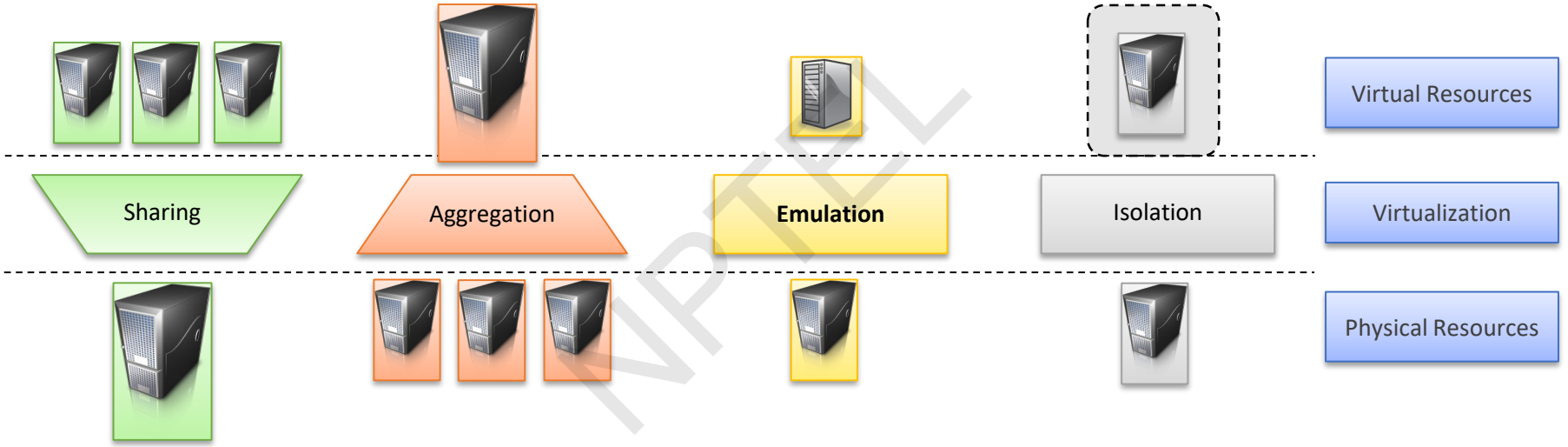
Managed execution



Aggregation - A group of separate hosts tied together and represented to guests as a single virtual host, implemented in middleware for distributed computing.

Characteristics of virtualized environments

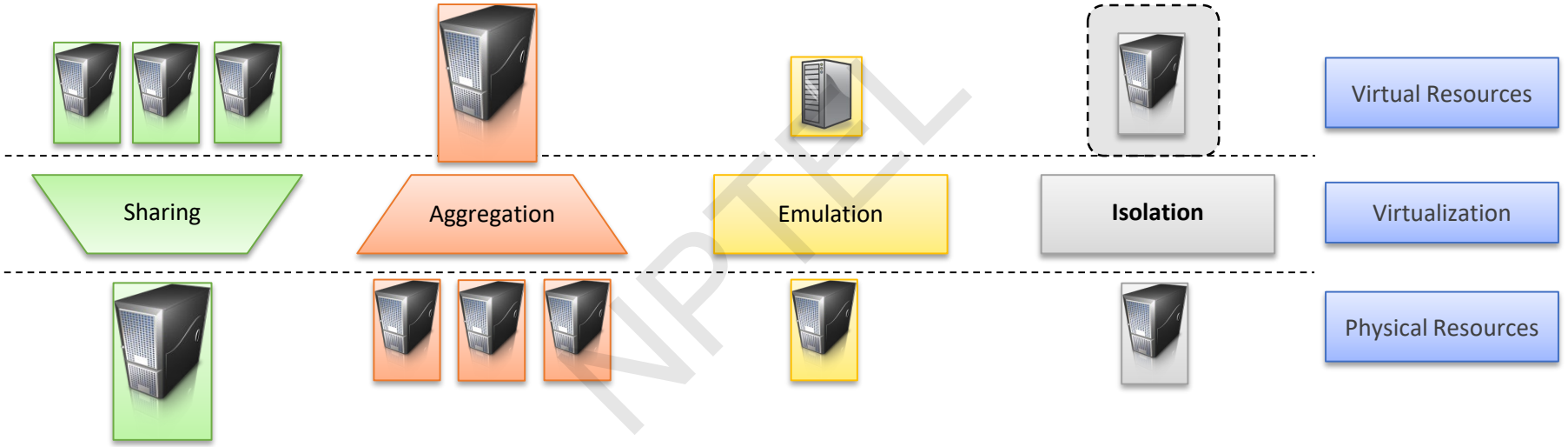
Managed execution



Emulation - Controlling and tuning the environment that is exposed to guests. E.g., arcade-game emulator to play arcade games on a normal personal

Characteristics of virtualized environments

Managed execution



Isolation - Allows multiple guests to run on the same host without interfering with each other, Also, Separation between the host and the guest.

Characteristics of virtualized environments

Performance-Tuning

- The guest's performance can be easily controlled by exposing appropriate amount or quality of resources provided by the host.
- Allows effective implementation of Quality of Service and fulfilling of Service Level Agreements for the guest

Migration

- With managed execution, the state of virtual machines/environments can be captured and persisted to storage. This allows for migration of virtual objects across server even across datacenters.

Characteristics of virtualized environments

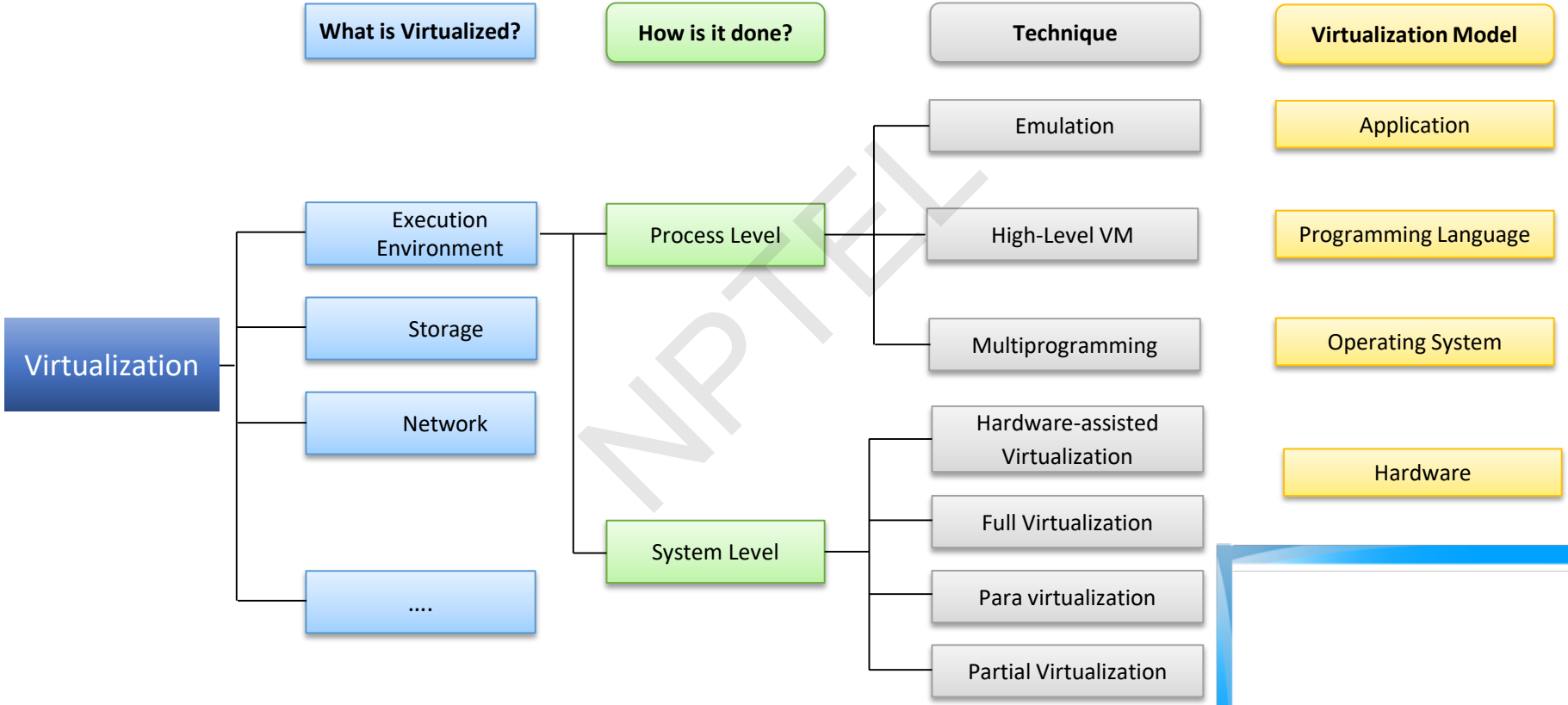
Portability

- **Hardware virtualization solution**
  guest packaged into a virtual image that can be safely moved and executed on top of different virtual machines.

- **Programming-level virtualization**
  Implemented by JVM or .NET runtime, binary code can be run without any recompilation.
  One version can run on different platforms with no changes.

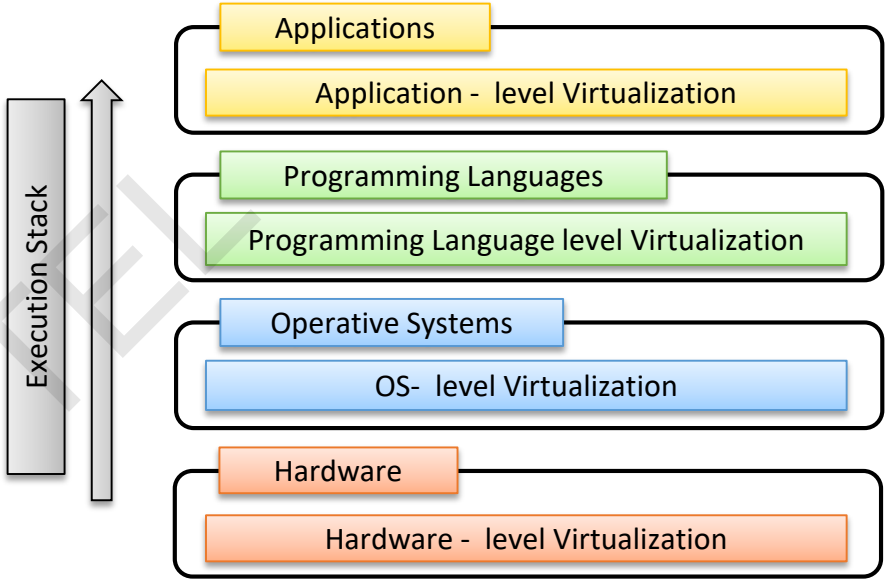
- Allows your own system always with you and ready to use if the required virtual machine manager is available (services you need available to you anywhere you go).

Virtualization Taxonomy



Execution Virtualization

It emulates an Execution Environment (EE), separated from host, implemented on top of H/W by OS, application (libraries) dynamically or statically linked to an application image.



Machine reference model

- Virtualizing an execution environment at different levels of **execution or computing stack** requires a reference model that defines interfaces between levels of abstractions.
- Virtualization techniques replace one of the layers.
- A clear separation between layers simplifies their implementation.
- It requires the emulation of interfaces and interaction with underlying layer.

Execution Virtualization

- **Instruction Set Architecture (ISA) :** processor, registers, memory and the interrupt management or H/W.
- **Application Binary Interface (ABI):** separates OS layer from application and libraries which are managed by the OS, System Calls defined, allows portability of applications and libraries across OS.
- **Application Programming Interfaces (API):** interfaces applications to libraries and/or the underlying OS.
- ISA has been divided into two security classes**:** **Privileged** Instructions and **Non-privileged** Instructions.

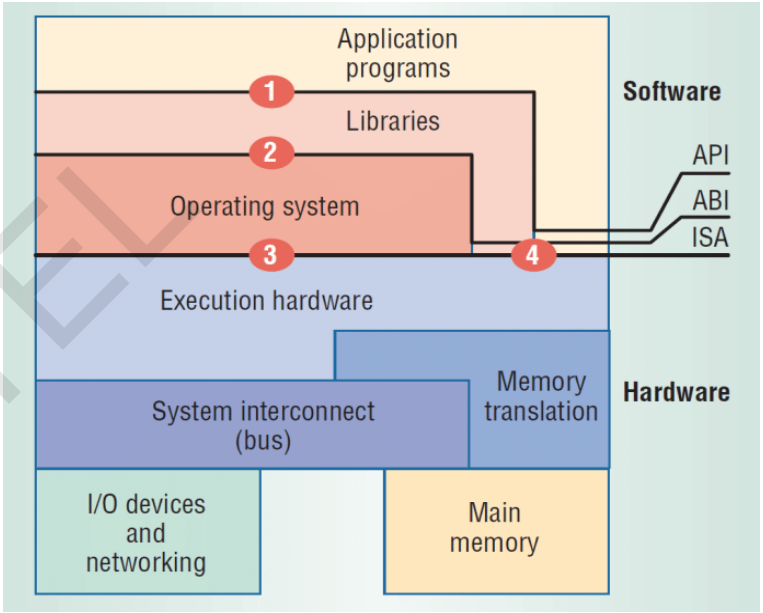


Figure Source: Homem, Irvin. (2016). Towards Automation in Digital Investigations: Seeking Efficiency in Digital Forensics in Mobile and Cloud Environments. 10.13140/RG.2.2.15114.26565.

Execution Virtualization

- **Non-privileged** instructions can be used without interfering with other tasks because they do not access shared resources (floating, fixed-point, and arithmetic instructions).

- **Privileged instructions** are executed under specific restrictions and mostly used for sensitive operations: behavior-sensitive: operate on the I/O) or control-sensitive: alter the state of the CPU.

- Some architectures feature more than one class of privileged instructions
- For instance, a hierarchy of privileges see (*Fig. Next Slide*), in the form of ring-based security: Ring 0, Ring 1, Ring 2, and Ring 3.

Security Rings

Security rings and **privilege modes** provide different levels of *access and control*. The most common implementation of security rings and privilege modes is found in the x86 architecture.

The idea behind **security rings** is to create a hierarchy of privilege levels, ensuring that critical system components operate with the highest privilege (Ring 0) while user applications operate at a lower privilege level (Ring 3). This helps **protect the system from accidental or malicious interference by user-level programs**.

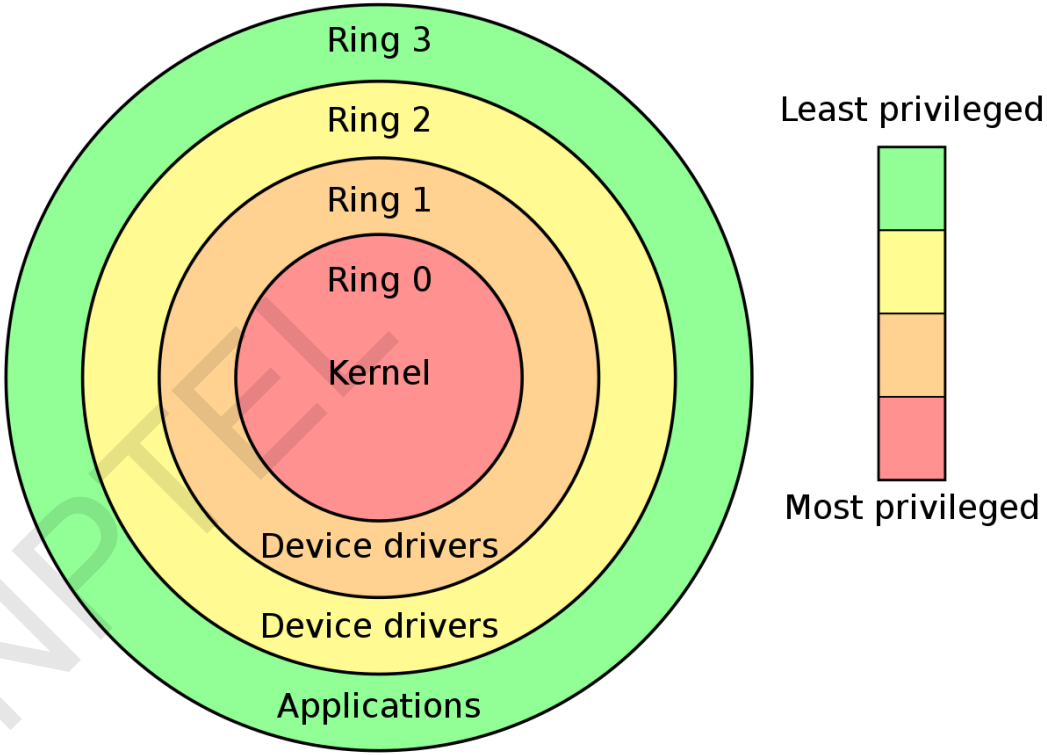


Figure Sources: https://en.wikipedia.org/wiki/Protection_ring

Security Rings

Security Rings: Security rings, also known as protection rings or privilege levels, define different layers of access and control in a computer system. Traditionally, these rings are numbered from 0 to 3, with 0 being the most privileged and 3 the least privileged. Each ring has a specific role and set of permissions:

Ring 0 (Kernel Mode): The most privileged ring. Has full access to the system's resources and can execute any instruction. Typically, the operating system kernel operates in Ring 0.

Ring 1 and Ring 2: Intermediate privilege levels. Not commonly used in mainstream operating systems. Some older systems experimented with these levels, but modern operating systems mostly use Ring 0 and Ring 3.

Ring 3 (User Mode): The least privileged ring. User applications run in Ring 3. Restricted in terms of direct hardware access and certain instructions.

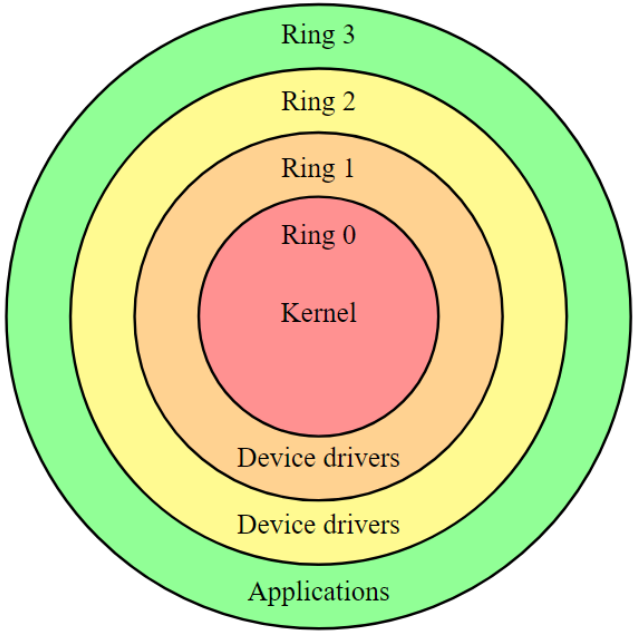


Figure Sources: https://en.wikipedia.org/wiki/Protection_ring

Privilege Modes

Privilege Modes: Privilege modes are related to the execution level of a processor, indicating the level of privilege a particular piece of code or instruction has. The x86 architecture, for example, defines four privilege levels:

Real Mode: The least privileged mode. Represents the mode of operation in early x86 processors. Limited memory addressing capabilities.
Protected Mode: Introduced with the 80286 processor. Provides memory protection, virtual memory, and multitasking support. Has four security rings (Ring 0 to Ring 3).
Virtual 8086 Mode: Allows running multiple virtual 8086 environments within protected mode. Used for running real-mode applications in a multitasking environment.
Long Mode (64-bit mode): Introduced with 64-bit processors (x86-64 or AMD64). Provides support for 64-bit addressing and additional features. Maintains compatibility with 32-bit protected mode.

These privilege modes allow the processor to switch between different levels of access and control, ensuring that certain operations can only be performed by code with the appropriate level of privilege. This helps enforce security boundaries and maintain the integrity of the system.

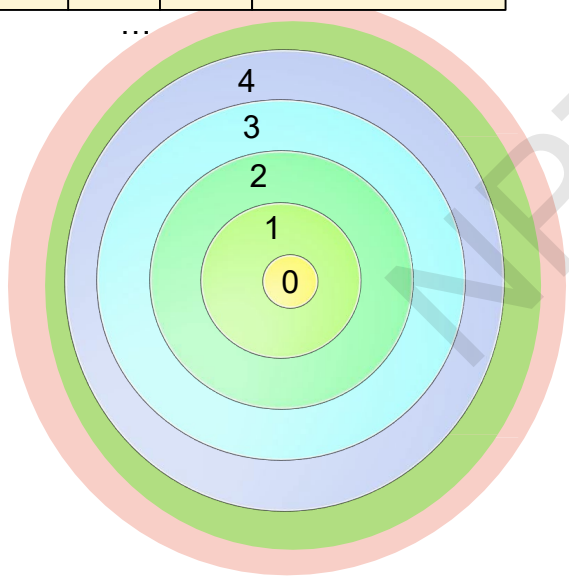
Privilege Modes

Segment Table revisited

…	…	…	…	…
Start address		size	**ring**	Access type
…		…		…

{read, write, execute}

…



Code permitted to access ring **i** may access ring **j** if and only if **j > i**

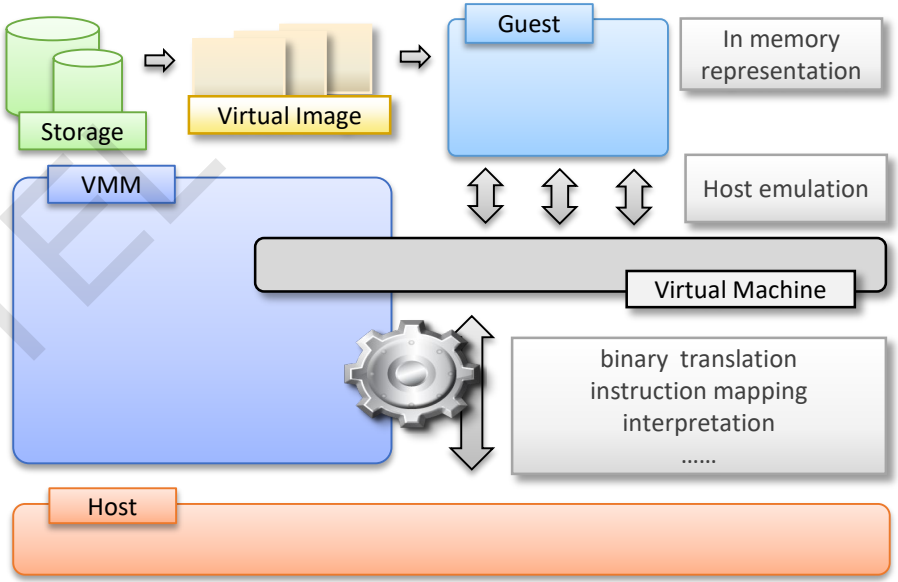
Execution virtualization

- Distinction between **user** & **supervisor** mode signifies role of the **hypervisor.**
- Hypervisors run in **supervisor** mode, and division between privileged and non-privileged instructions makes challenging design of virtual machine managers.
- ***Sensitive instructions execute in privileged mode, requires supervisor mode to avoid traps.***
- Without this assumption it is impossible to fully emulate and manage the status of the CPU for guest operating systems.
- This is not true for the original ISA, allows 17 sensitive instructions to be called in **user** mode - prevents multiple OSs managed by a single hypervisor to be isolated from each other.
- Since, they access the privileged state of the processor and change it.

- Recent implementations of ISA (**Intel VT and AMD Pacifica**) solved this problem by **redesigning such instructions as privileged ones**.
- ***In hypervisor-managed environment, all guest OS code will run in user mode in order to prevent it from directly accessing the status of the CPU.***
- If there are sensitive instructions that can be called in user mode, it is no longer possible to completely isolate the guest OS.

Hardware Virtualization

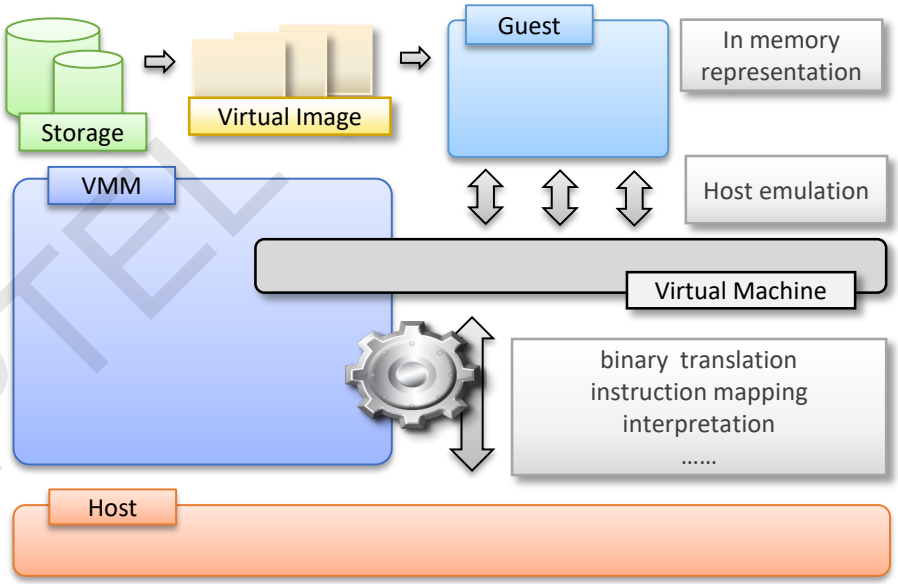
- Provides an abstract execution environment in terms of computer hardware on top of which a guest operating system can be run.
- The **guest** is represented by the operating system, the **host** by the physical computer hardware, the **virtual machine** by its emulation, and the **virtual machine manager** by the hypervisor (see Figure).
- The **hypervisor** is generally a **program** or a combination of **software** and **hardware** that allows the abstraction of the underlying physical hardware.
- Hardware-level virtualization is also called System level Virtualization.
- Since, it provides ISA to virtual machines, which is the representation of the hardware interface of a system.
- This is to differentiate it from process virtual machines, which expose ABI to virtual machines.



A hardware virtualization reference model

Hypervisors(or VMM):

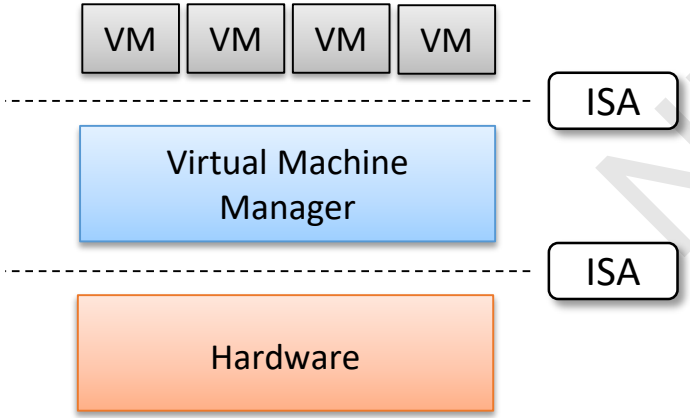
- Hypervisor runs above the physical hardware.
- It runs in supervisor mode.
- It recreates a h/w environment in which guest operating systems are installed.
- It is a piece of s/w that enables us to run one or more VMs on a physical server(host).
- Two major types of hypervisor
- Bare-metal or Native – ***Type -I***
- Hosted – ***Type-II***



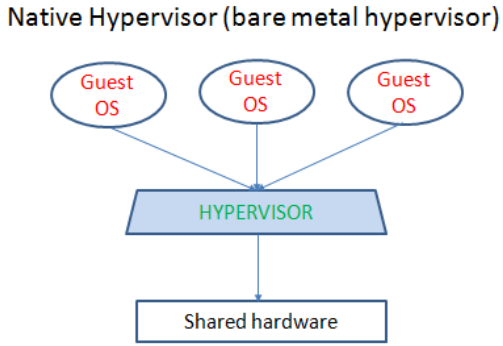
Hardware Virtualization

Hypervisors (Type-I):

- It runs directly on top of the hardware, takes place of OS.
- Directly interact with the ISA exposed by the underlying hardware and emulate this interface in order to allow the management of guest OS.
- Also known as *native virtual machine.*



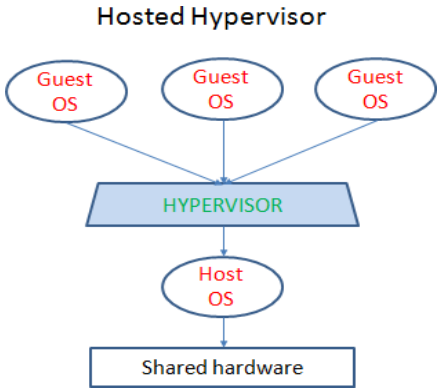
Type-I (Native)

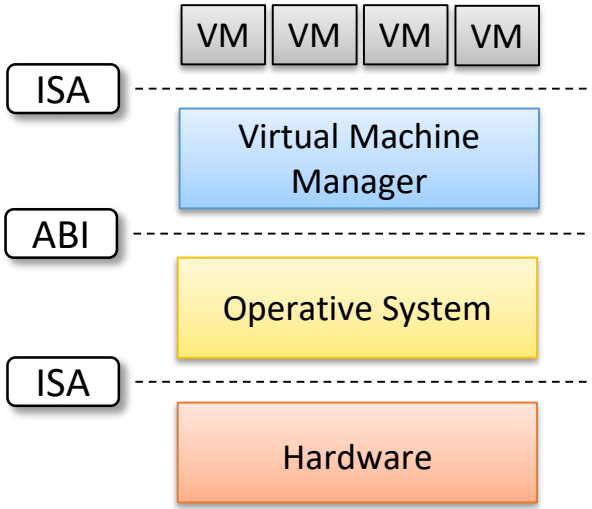


Hardware Virtualization

Hypervisors (Type-2):

- It requires the support of an operating system to provide virtualization services.
- Programs managed by the OS, interact with OS through the ABI.
- Emulate the ISA of virtual h/w for guest OS.
- Also called hosted virtual machine.
- *No modification to Operating System code for both Type-1 and Type-2 hypervisors*

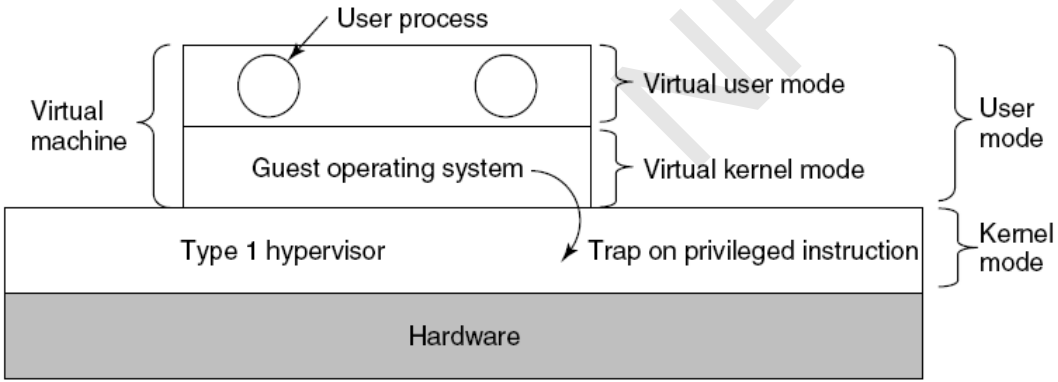




Type 1 Hypervisor

- Unmodified Operating System is running in user mode
  - But it thinks it is running in kernel mode (*virtual kernel mode)*
  - privileged instructions trap;
  - Hypervisor is the “real kernel”
    - Upon trap, executes privileged operations
    - Or emulates what the hardware would do

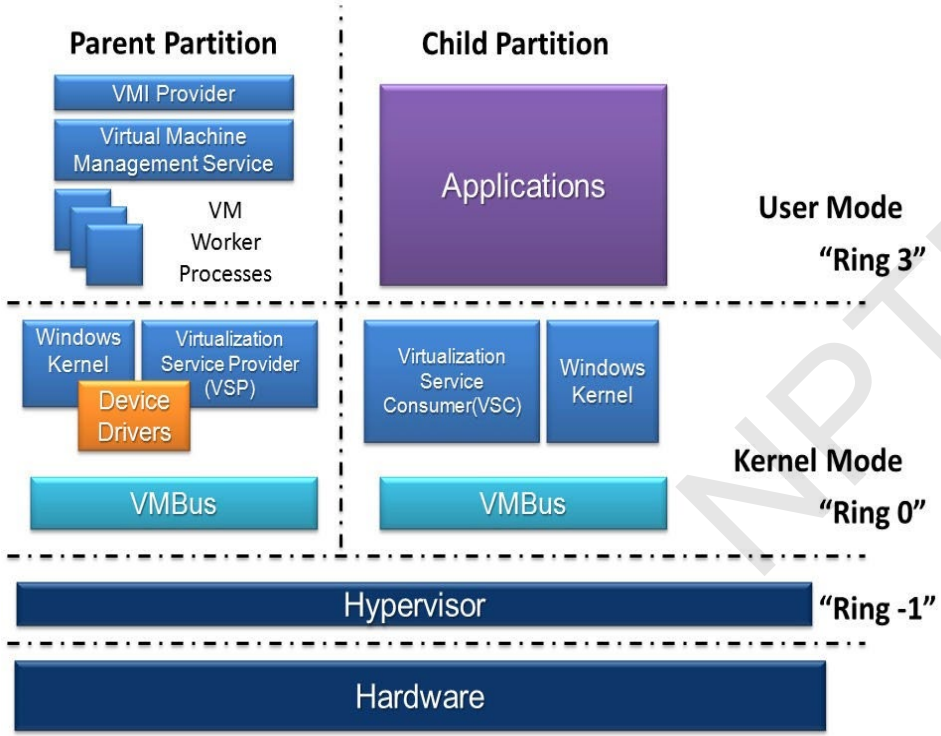
    Ex: VMware ESXi or Microsoft Hyper-V



Source: https://www.doc.ic.ac.uk/~etheresk/etheresk-211-vms.pdf

Type 1 Hypervisor Example

Microsoft Hyper-V



- **Microsoft Hyper-V** implements isolation of virtual machines in terms of a partition. A partition is a logical unit of isolation, supported by the hypervisor, in which each guest operating system executes.

- There must be at least one parent partition in a hypervisor instance, running a supported version of Windows.

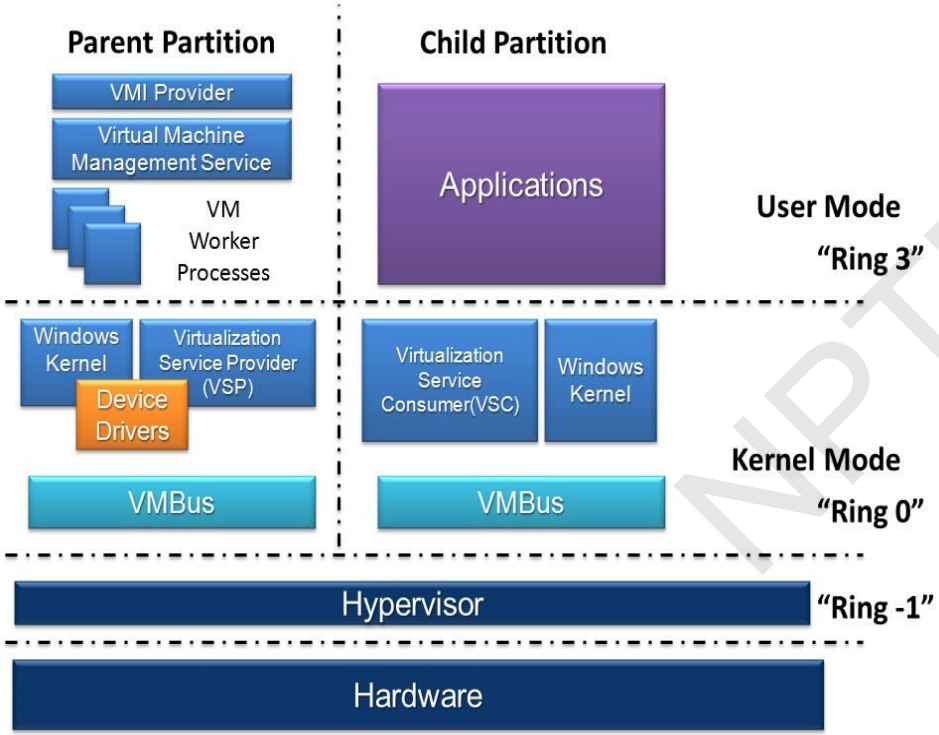
- The **parent partition** creates **child partitions** which host the guest OSs.

- child partitions are created using the *hypercall* API, which is the application programming interface exposed by Hyper-V

Figure source: https://en.wikipedia.org/wiki/Hyper-V

Type 1 Hypervisor Example

Microsoft Hyper-V



- A **child partition** does not have access to the physical processor, nor does it handle its real interrupts.

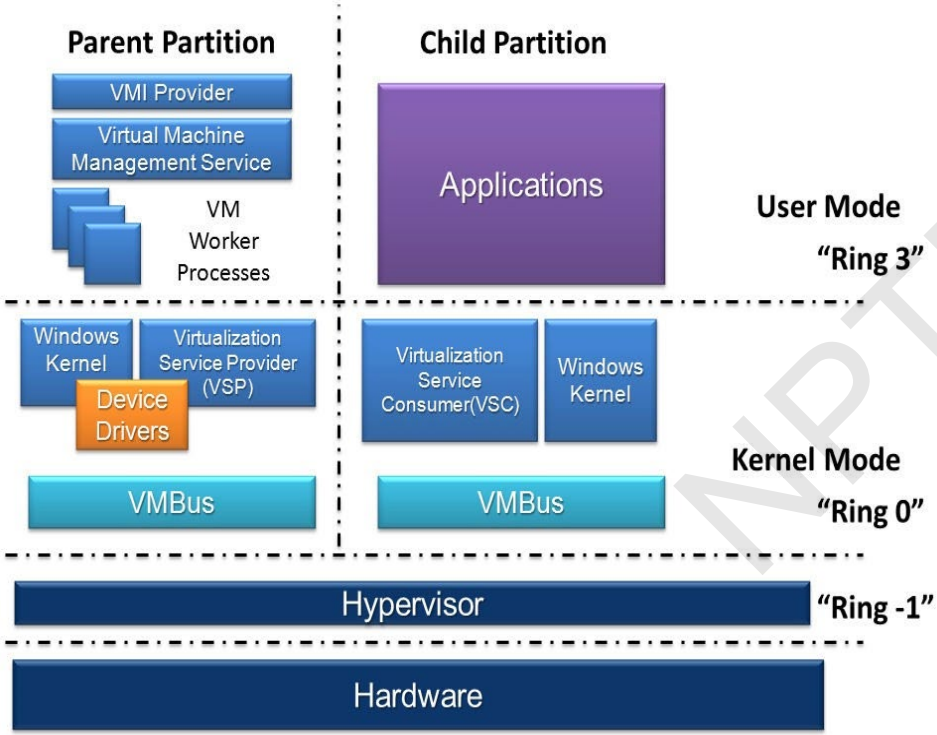
- Instead, it has a virtual view of the processor and runs in **Guest Virtual Address**.

- The hypervisor handles the interrupts to the processor (hardware) and redirects them to the respective partition using a logical **Synthetic Interrupt Controller** (SynIC).

Figure source: *https://en.wikipedia.org/wiki/Hyper-V*

Type 1 Hypervisor Example

Microsoft Hyper-V

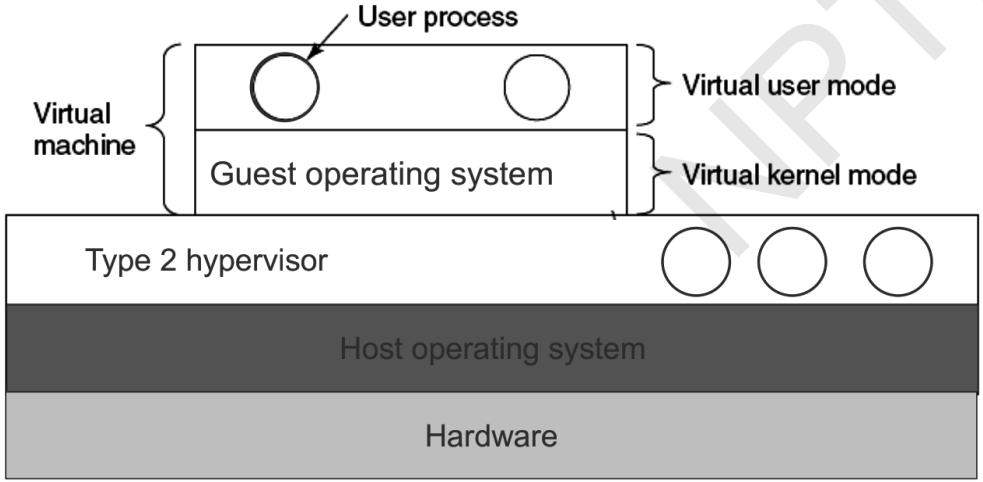


- **Child partitions** do not have direct access to hardware resources, but instead have a virtual view of the resources, in terms of **virtual devices**.
- The **VMBus** is a logical channel which enables inter-partition communication. Any request to the virtual devices is redirected via the **VMBus** to the devices in the parent partition, which will manage the requests.
- **Parent partitions** run a **Virtualization Service Provider (VSP)**, which connects to the **VMBus** and handles device access requests from **child partitions**.
- virtual devices on **child partitions** internally run a **Virtualization Service Client (VSC)**, which redirect the request to **VSPs** in the **parent partition** via the **VMBus**.

Figure source: *https://en.wikipedia.org/wiki/Hyper-V*

Type 2 Hypervisor

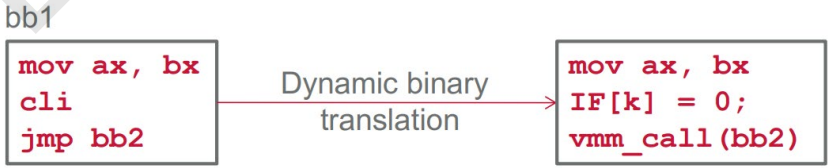
- Run hypervisor inside host OS
- Great for end-users: easy to install (like a regular app!)
- Can use I/O drivers from host OS
- Can use services (e.g., scheduling) from host OS
- Ex: VMWare Workstation , Oracle VirualBox



Source: https://www.doc.ic.ac.uk/~etheresk/etheresk-211-vms.pdf

Type 2 Hypervisor Example

- **VMWare Workstation**
  - Type 2 hypervisor Scan each **basic block** (bb) just before it executes
  - If it contains privileged instructions, replace by hypervisor calls - **Binary translation**
  - Replace last instruction with hypervisor call
  - Execute basic block natively
  - Cache modified(translated) basic block in VMWare cache - Execute; load next basic block etc.
- Type 2 hypervisors work without VT support
  (no automatic trapping of privileged instructions by hypervisors)



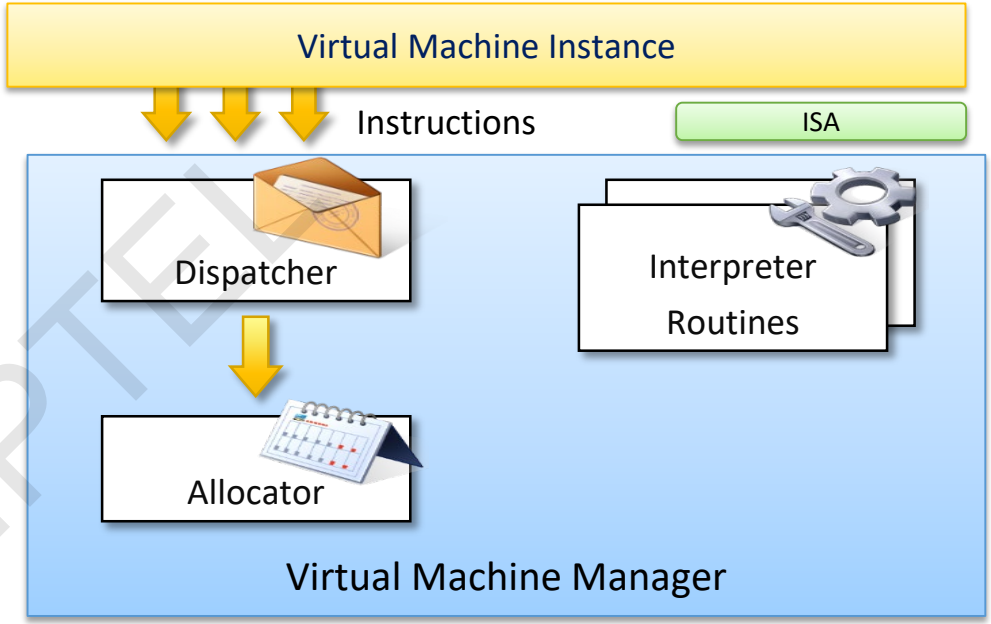
Binary translation
Introduced by the VMWare Workstation (Type 2 hypervisor)

- **Issue** - Frequent traps can incur significant overhead to clear TLBs, caches, branch predictions tables, etc.
- **Solution** - Eliminate some of the traps via dynamic **binary translation** i.e., Instead of emulating code (slow!), dynamically translate it so that it can run natively
- Main goal was to handle un-virtualizable architectures
- But can also be used by Type 1 hypervisors to improve performance

Hardware Virtualization

VMM: Main Modules coordinate to emulate the underlying hardware:-

- **Dispatcher:** Entry Point of VMM
- Reroutes the instructions issued by VM instance.
- **Allocator:** Decides system resources to be provided to the VM;
- a VM executes an instruction, results in changing machine resources associated with that VM.
- Invoked by dispatcher
- **Interpreter:** Consists of interpreter routines
- Executed whenever a VM executes a privileged instruction.
- Trap is triggered and the corresponding routine is executed.



Hardware Virtualization

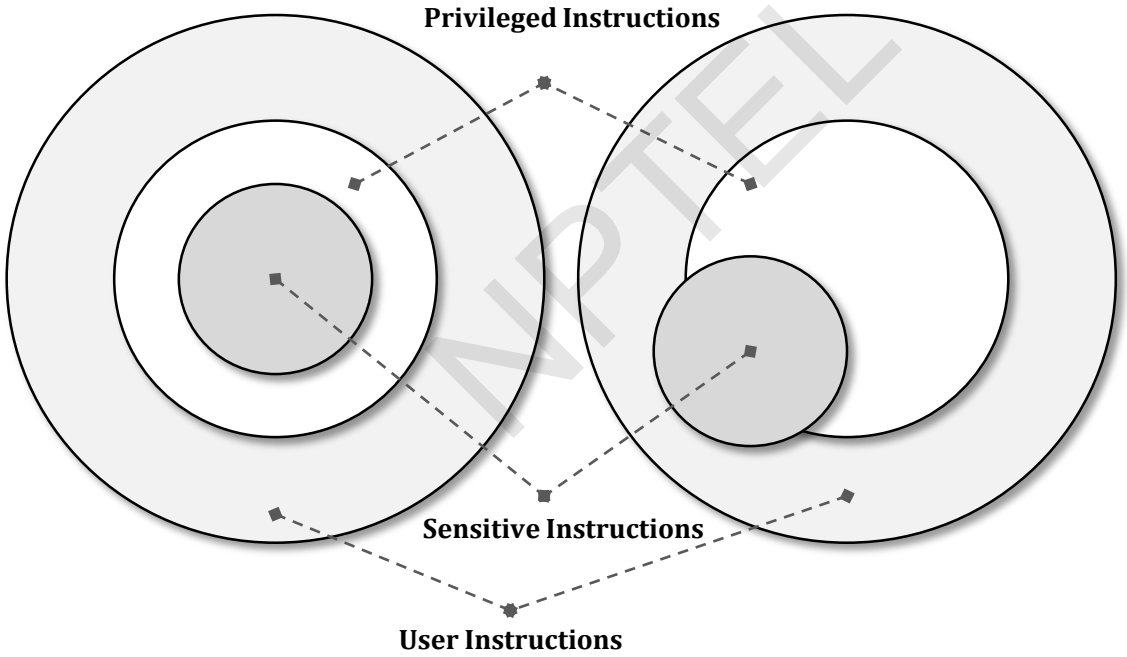
Criteria of VMM*:

- **Equivalence** – same behavior as when it is ***executed directly*** on the physical host when it was running under control of VMM.
- **Resource control** – VMM should be in ***complete control of virtualized resources***.
- **Efficiency** – a statistically dominant fraction of the machine instructions should be ***executed without intervention*** from the VMM.
- **Theorems*: C*lassification of the instruction set*** and proposed three theorems that define the properties that ***hardware instructions need to satisfy*** in order to efficiently support virtualization.
- Classification of IS:
- Privileged Instructions: trap if the processor is in user mode
- Control sensitive Instructions
- Behavior sensitive Instructions

** criteria's are established by Goldberg and Popek in 1974*

Hardware Virtualization

Theorem1 : For any conventional third-generation computer, a VMM may be constructed if the set of sensitive instructions for that computer is a subset of the set of privileged instructions.



Hardware Virtualization

Theorem 2:

- A conventional third-generation computers is recursively virtualizable if:
- It is virtualizable and
- A VMM without any timing dependencies can be constructed for it.

Theorem 3:

- A hybrid VMM may be constructed third-generation machine in which the set of user-sensitive instructions is a subset of the set of privileged instructions.
- *In HVM, more instructions are interpreted rather than being executed directly.*

Hardware Virtualization

Hardware Virtualization Techniques:

- CPU installed on the host is only one set, but each VM that runs on the host requires their own CPU.
- CPU needs to be virtualized, done by hypervisor.

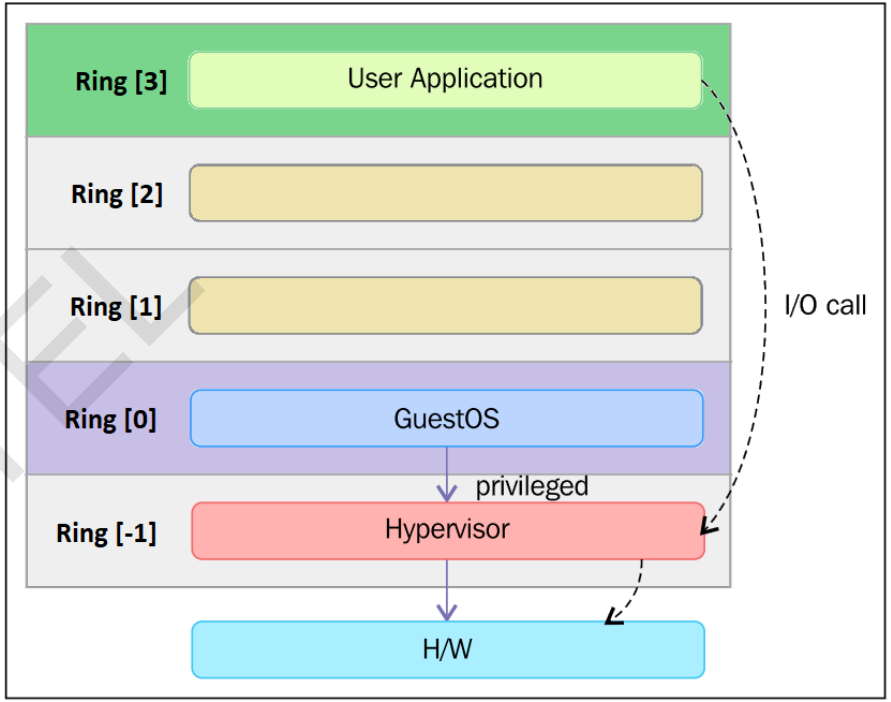
Types of HVT:

- **Full virtualization**
- **Hardware-assisted virtualization**
- **Para virtualization**
- **Partial virtualization**

Hardware Virtualization

Hardware-assisted virtualization:

- In this hardware provides architectural support for building a VMM able to run a guest OS in complete isolation.
- ***Intel VT*** and ***AMD V*** extensions.
- Early products were using ***binary translation to trap some sensitive instructions*** and provide an emulated version.
- Additional Ring -1
- ***No binary translation*** of privileged instructions.
- Commands are executed directly to h/w via the hypervisor.



Hardware Virtualization

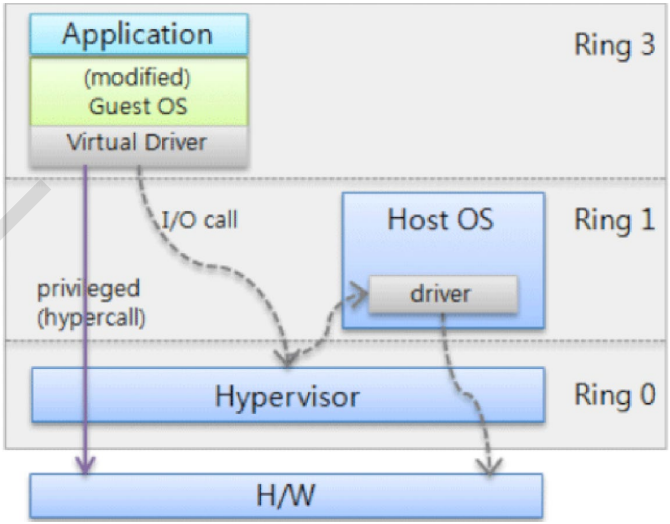
Para virtualization: Not-transparent virtualization

– Thin VMM

– Expose software interface to the virtual machine that is slightly modified from the host.

– Guest OS needs to be modified.

– Simply transfer the execution of instructions which were hard to virtualized, directly to the host.

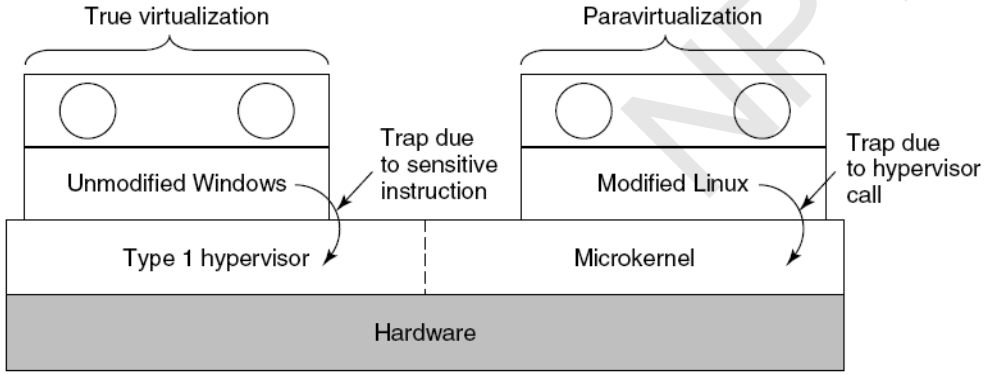


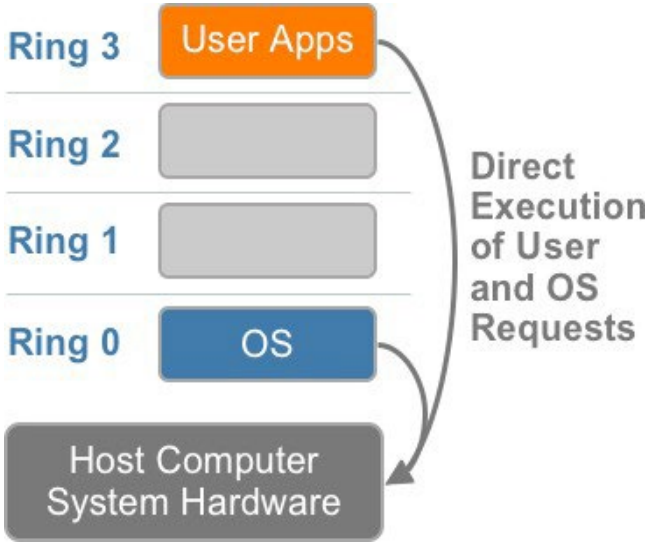
- Privileged instructions of guest ***OS is delivered to the hypervisor*** by using hyper calls
- Hyper calls handles these instructions and accesses the h/w and return the result.
- Guest has authority to ***directly control*** of resources.

Figure Source: https://www.unixarena.com/2017/12/para-virtualization-full-virtualization-hardware-assisted-virtualization.html

Hardware Virtualization

- Both type 1 and 2 hypervisors work on unmodified OS
- Para virtualization: modify OS kernel to replace all sensitive instructions with hyper calls
  - OS behaves like a user program making system calls
  - Hypervisor executes the privileged operation invoked by hyper call.





Hyper calls to the Virtualization Layer replace Non-virtualized OS instructions

Source: https://www.doc.ic.ac.uk/~etheresk/etheresk-211-vms.pdf

Hardware Virtualization

Partial virtualization:

- Partial emulation of the underlying hardware
- Not allow complete isolation to guest OS.
- Address space virtualization is a common feature of
- Contemporary operating systems.
- Address space virtualization used in time-sharing system.

Virtual CPU (VM) scheduling

- Based on fair-share balancing algorithm
  - Each VM gets equal time on the processor
- The VMs are allocated to physical processor core based on (Availability, Utilization, Priority)
- How privileged instructions requested by VMs depends on
  - Type of virtualization (Full virtualization, Para Virtualization, …)

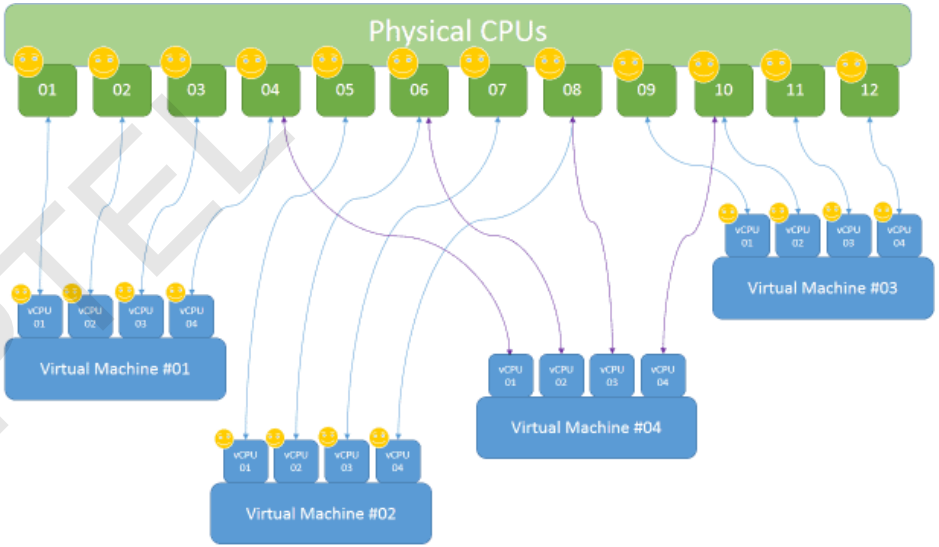


Figure Source: *https://www.flackbox.com/virtual-processor-scheduling-how-vmware-and-microsoft-hypervisors-work-at-the-cpu-level*

Operating System-level virtualization

- To create different and ***separated execution environments*** for applications that are managed concurrently.
- No VMM or hypervisor.
- Virtualization is in single OS.
- OS kernel allows for multiple isolated user space instances.
- OS kernel is responsible for sharing the system resources among instances and other management.
- A user space instance has a proper view of the file system (isolated), separate IP addresses, software configurations, and access to devices.
- OS supporting virtualization are general-purpose, time-shared operating systems with the capability to provide resource isolation.

Operating System-level virtualization

- Considered an evolution of the Chroot mechanism in Unix systems.
- Compared to H/W virtualization, no overhead because applications directly use OS system calls and there is no need for emulation.
- No need to modify applications, nor to modify any specific hardware.
- Good for server consolidation; multiple application servers share the same technology: operating system, application server framework etc.
- Different servers are aggregated into one physical server, each server is run in a different user space, completely isolated from the others. Ex: OpenVZ, IBM Logical Partition (LPAR), Solaris Zones and Containers.

Programming language-level virtualization

- To ease deployment of applications, managed execution, Security and portability across different platforms (OSs).
- A virtual machine executes the byte code of a program.
- VMs constitute a simplification of the underlying hardware instruction set and provide some high-level instructions that map some of the features of the languages compiled for them.
- At runtime, byte code can be either interpreted or compiled on the fly (JIT) against underlying hardware instruction set.
- Both Java and the Common Language Infrastructure (CLI), are stack-based virtual machines.
- Stack-based virtual machines possess the property of being easily interpreted and executed simply by lexical analysis and hence are easily portable over different architectures.

Application-level virtualization

- Applications run in runtime environments that do not natively support all the features required by such applications.
- Applications are not installed in the expected runtime environment but are run as though they were.
- Mostly concerned with partial file systems, libraries, and operating system component emulation.
- Such emulation is performed by a thin layer—a program or an operating system component—that is in charge of executing the application.
- *Emulation can also be used to execute program binaries compiled for different hardware architectures.*

Application-level virtualization

Emulation strategies implemented:

- **Interpretation**: source instruction is ***interpreted*** by an emulator for executing ***native ISA instructions***, ***minimal*** start up cost but ***huge overhead***.
- **Binary translation:** source instruction is ***converted to native*** instructions with equivalent functions.
- Block of instructions ***translated*** , ***cached*** and ***reused***.
- Large ***overhead cost*** , but over time it is subject to ***better performance***.
- In h/w virtualization , it allows the execution of a program ***compiled against a different h/w***.
- In Application-level emulation , complete ***h/w environment***.

Other types of virtualization

- **Desktop virtualization** : abstracts the desktop environment available on a personal computer to access it using a client/server approach.
- Same outcome of hardware virtualization but serves a different purpose, makes accessible a different system as though it were natively installed on the host, but this system is remotely stored on a different host and accessed through a network connection.
- Same desktop environment accessible from everywhere.
- A highly available data center ensures the accessibility and persistence of the data, that is required.

Virtual Desktop Infrastructure (VDI) and Remote Desktop Services session-based desktops are the key technologies that enable virtual desktops, whereby a desktop that runs in the data center can be delivered to the end-user's device using Remote Desktop Protocol (RDP). When combined with technologies that enable application and user state virtualization, organizations can achieve a high degree of desktop optimization and security and reduced TCO.

Other types of virtualization

- **Storage virtualization**: A system administration practices decoupling the physical organization of the hardware from its logical representation.
- No worried about the specific location of their data, it can be identified using a logical path.
- Harness a wide range of storage facilities and represent them under a single logical file system.
- Storage Area Network (SANs) use a network-accessible device through a large bandwidth connection to provide storage facilities.

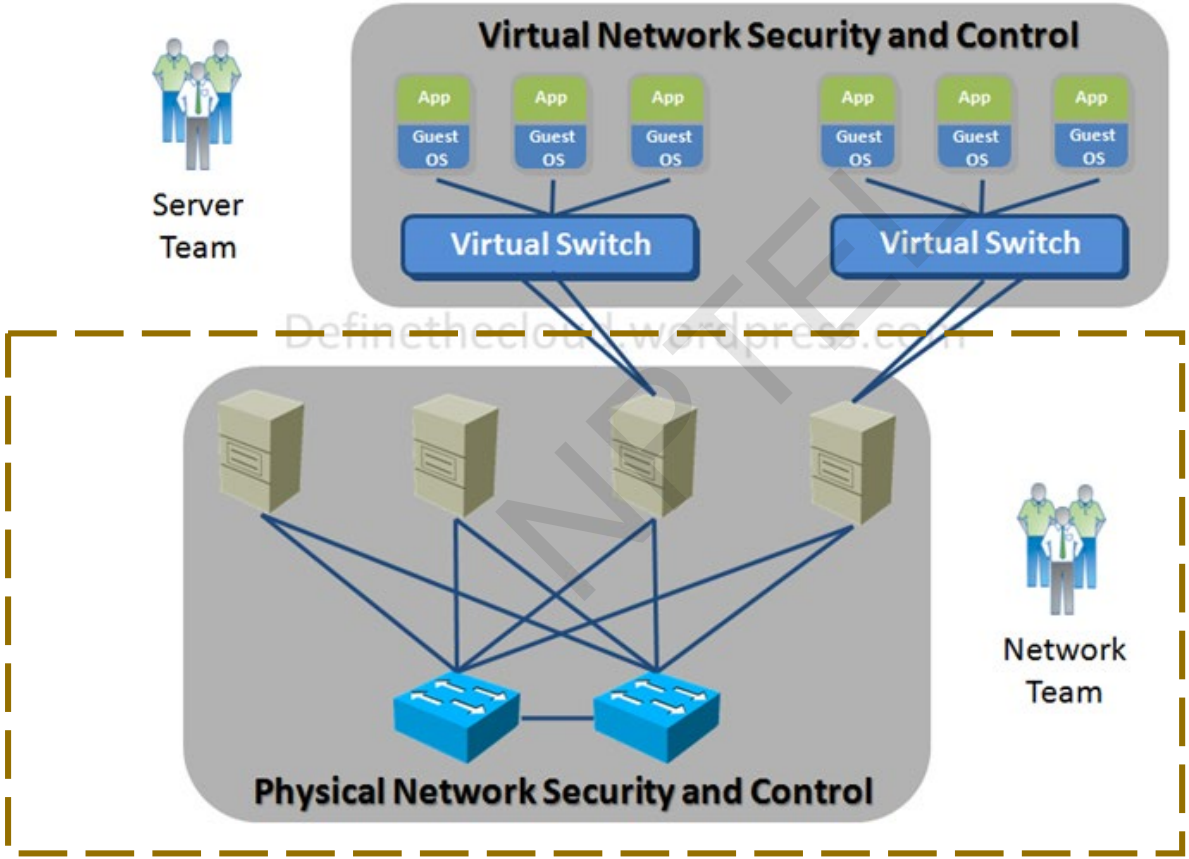
Other types of virtualization

- **Network Virtualization** combines hardware appliances and specific software for the creation and management of a virtual network.
- Aggregate different physical networks into a single logical network (external network virtualization).
- Provide network-like functionality to an operating system partition (internal network virtualization)
- The result of external network virtualization is generally a virtual LAN.
- A VLAN is an aggregation of hosts that communicate with each other as though they were located under the same broadcasting domain.
- Internal network virtualization is applied together with hardware and operating system-level virtualization, in which the guests obtain a virtual network interface to communicate with.

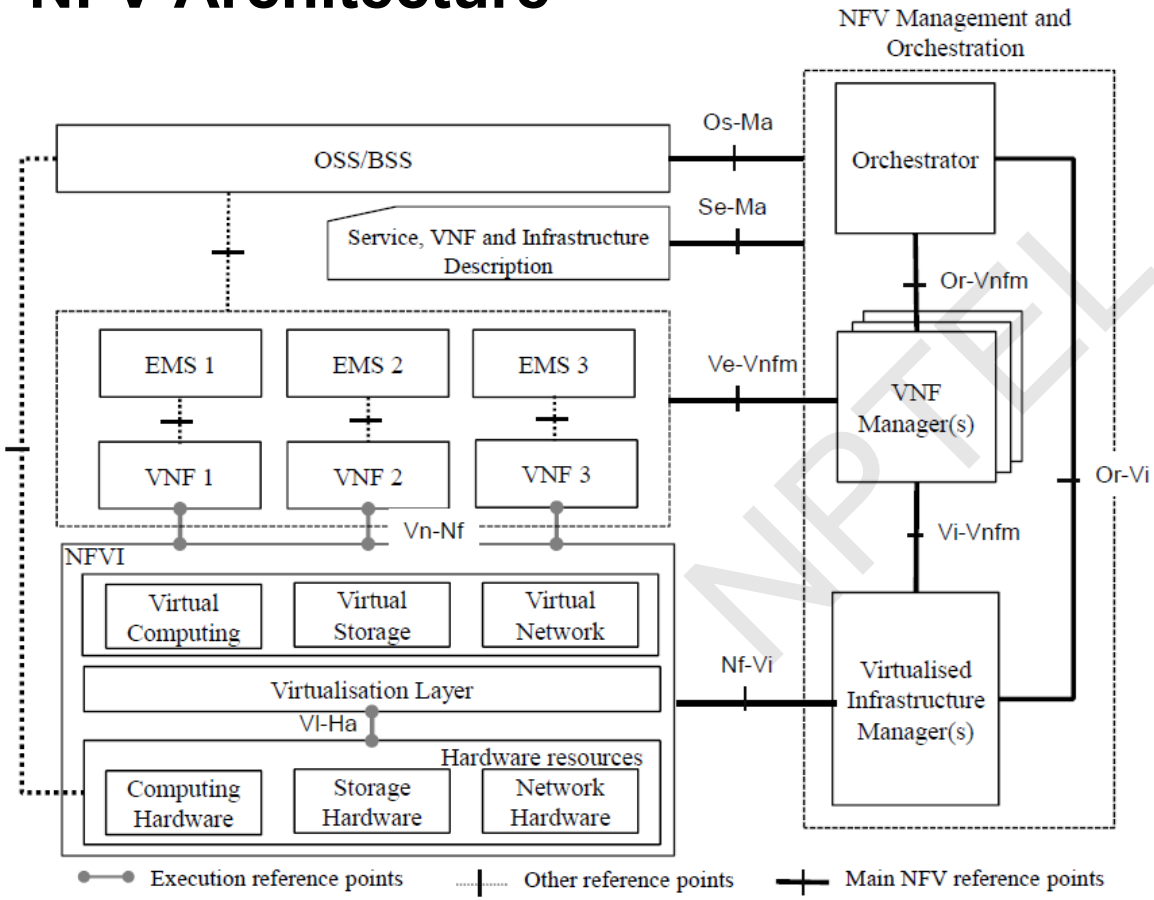
Network Function Virtualization

- Network Function Virtualization (NFV) is a technology that leverages virtualization to consolidate the heterogeneous network devices onto industry standard high-volume servers, switches and storage.
- Relationship to SDN (Software Defined Networking)
  - NFV is complementary to SDN as NFV can provide the infrastructure on which SDN can run.
  - NFV and SDN are mutually beneficial to each other but not dependent.
  - Network functions can be virtualized without SDN, similarly, SDN can run without NFV.
- NFV comprises of network functions implemented in software that run on virtualized resources in the cloud.
- NFV enables a separation the network functions which are implemented in software from the underlying hardware.

Data Centre Network



NFV Architecture



- Key elements of the NFV architecture is the Virtualized Network Function (VNF): VNF is a software implementation of a network function which is capable of running over the NFV Infrastructure (NFVI).
- NFV is a network architecture concept that uses the technologies of IT virtualization to virtualize entire classes of network node functions into building blocks that may connect, or chain together, to create communication services.

Figure Source: *https://www.etsi.org/*

NFV Architecture

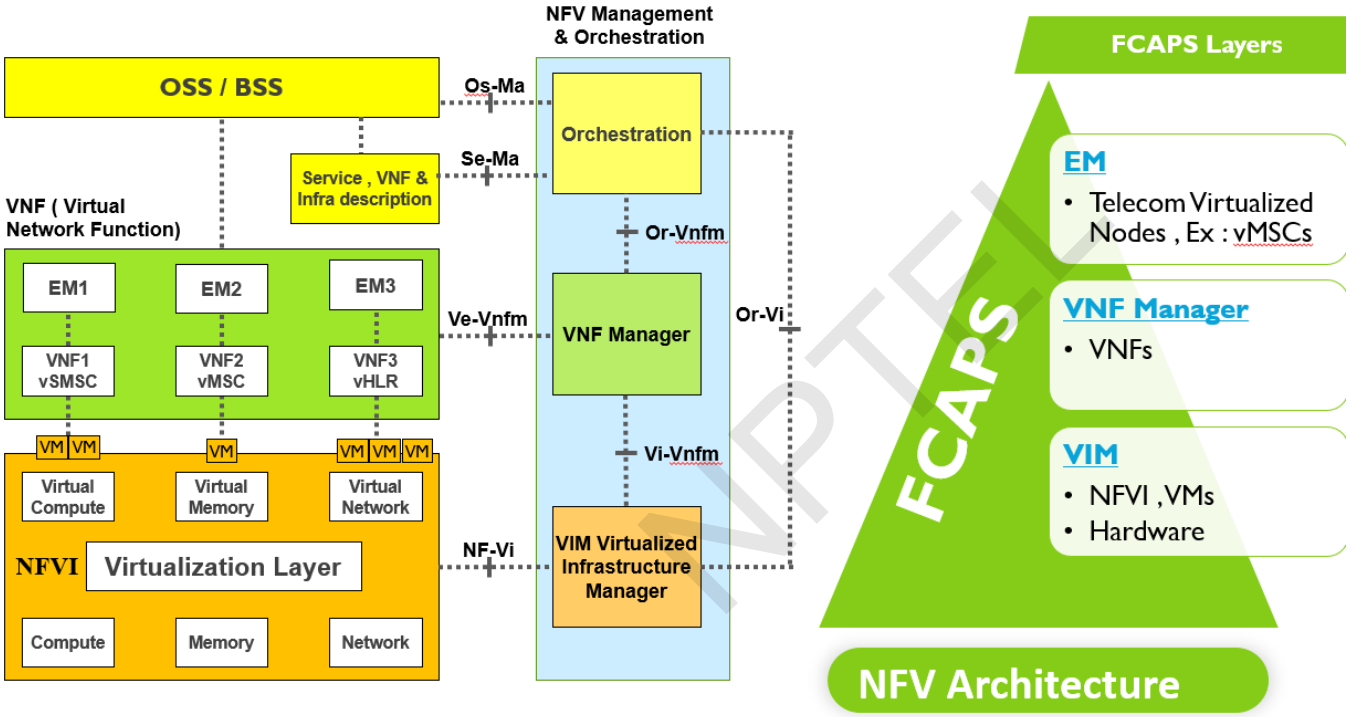
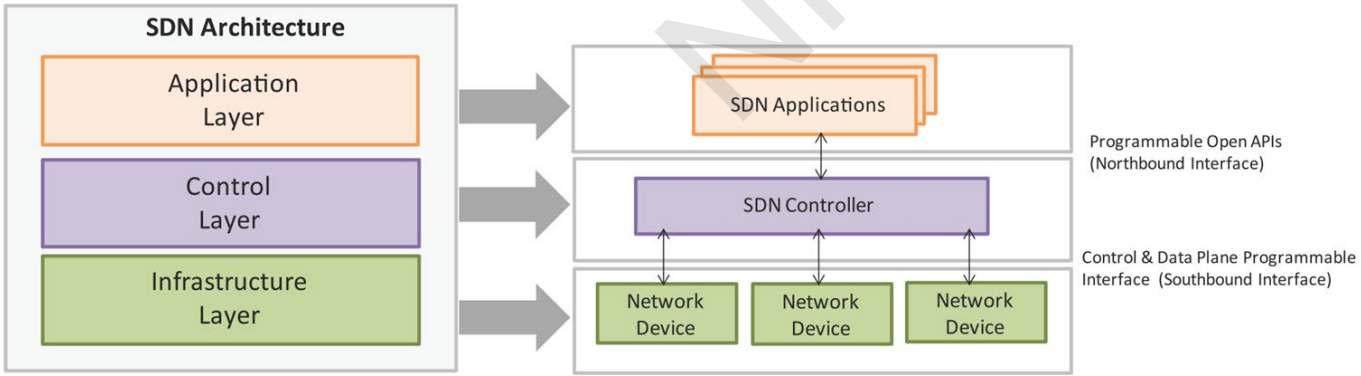


Figure Source: *https://www.telecomtutorial.info/post/introduction-to-nfv-network-function-virtualization*

Software Defined Networking

- Software-Defined Networking (SDN) is a networking architecture that separates the control plane from the data plane and centralizes the network controller.
- Conventional network architecture
  - The control plane and data plane are coupled. Control plane is the part of the network that carries the signaling and routing message traffic while the data plane is the part of the network that carries the payload data traffic.
- SDN Architecture
  - The control and data planes are decoupled, and the network controller is centralized.

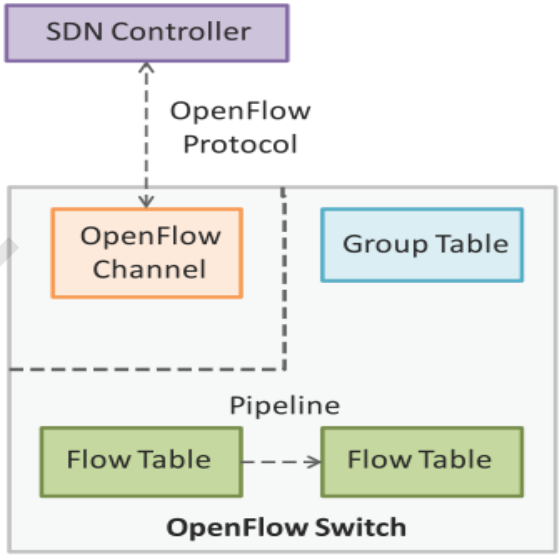


SDN - Key Elements

- Centralized Network Controller
  - With decoupled the control and data planes and centralized network controller, the network administrators can rapidly configure the network.
- Programmable Open APIs
  - SDN architecture supports programmable open APIs for interface between the SDN application and control layers (Northbound interface). These open APIs that allow implementing various network services such as routing, quality of service (QoS), access control, etc.
- Standard Communication Interface (Open Flow)
  - SDN architecture uses a standard communication interface between the control and infrastructure layers (Southbound interface). Open Flow, which is defined by the Open Networking Foundation (ONF) is the broadly accepted SDN protocol for the Southbound interface.

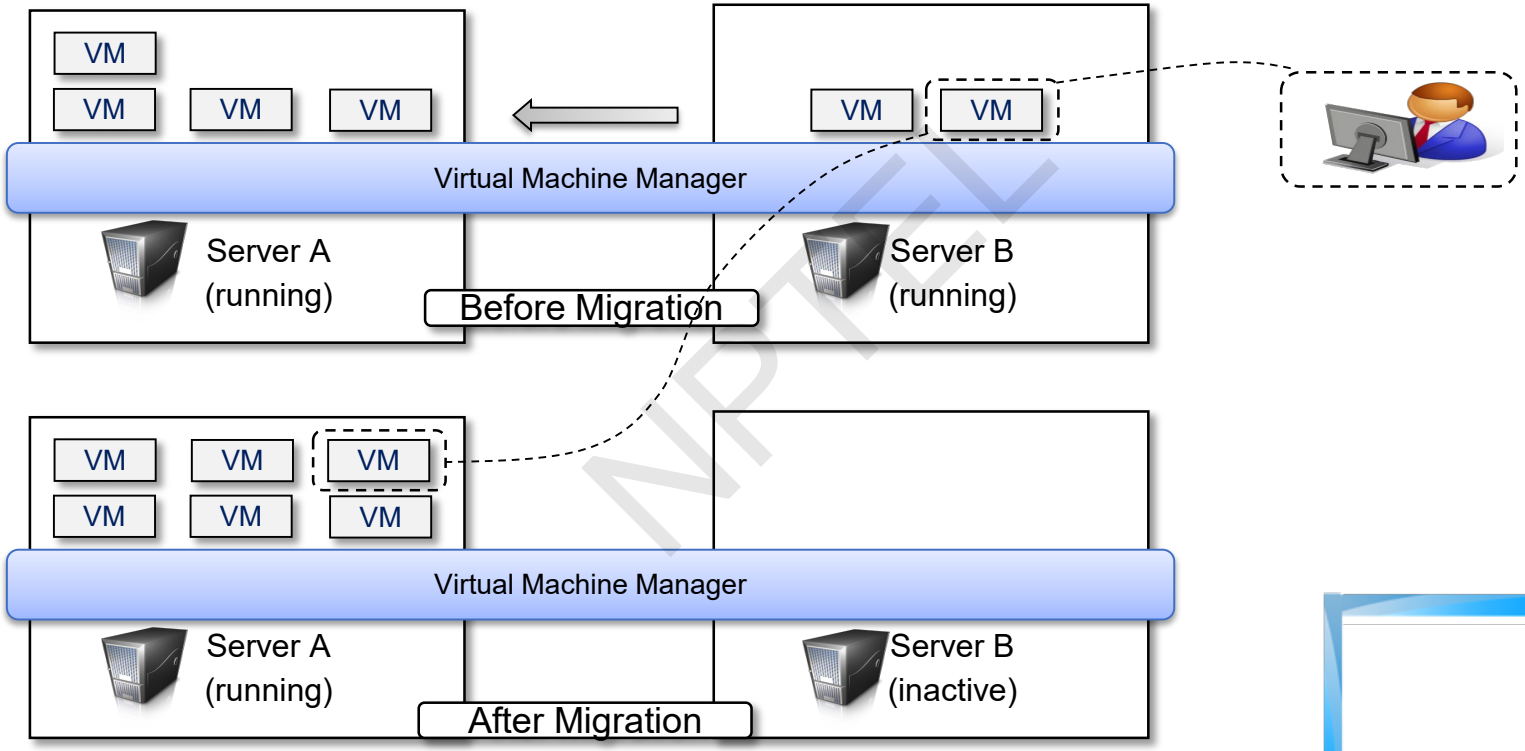
Open Flow

- Open Flow is the broadly accepted SDN protocol for the Southbound interface.
- With Open Flow, the forwarding plane of the network devices can be directly accessed and manipulated.
- Open Flow uses the concept of flows to identify network traffic based on pre-defined match rules.
- Flows can be programmed statically or dynamically by the SDN control software.
- Open Flow protocol is implemented on both sides of the interface between the controller and the network devices.



Open Flow switch comprising of one or more flow tables and a group table, which perform packet lookups and forwarding, and Open Flow channel to an external controller.

Virtualization and Cloud Computing (VM Migration)



Conclusion

In this lecture we covered the following,

- overview of virtualization and the befinits it can provide.
- virtualization at various levels – execution, hardware, programming, operating system, hardware etc.
- virtualization in edge and cloud.

Thank You!

References

- http://wiki.libvirt.org (libvert documentation)
- *Mastering Cloud Computing Foundation and Applications Programming* – Rajkumar Buyya (The University of Melbourne and Manjrasoft Pty Ltd, Australia), Christian Vecchiola (The University of Melbourne and IBM Research, Australia), S. Thamarai Selvi (Madras Institute of Technology, Anna University, Chennai, India). ISBN: 978-0-12-411454-8
- *Virtualization Essentials* - Matthew Portnoy. ISBN: 978-1-119-26772-0

Week 2 Lecture 2

0 mins

Docker Containers

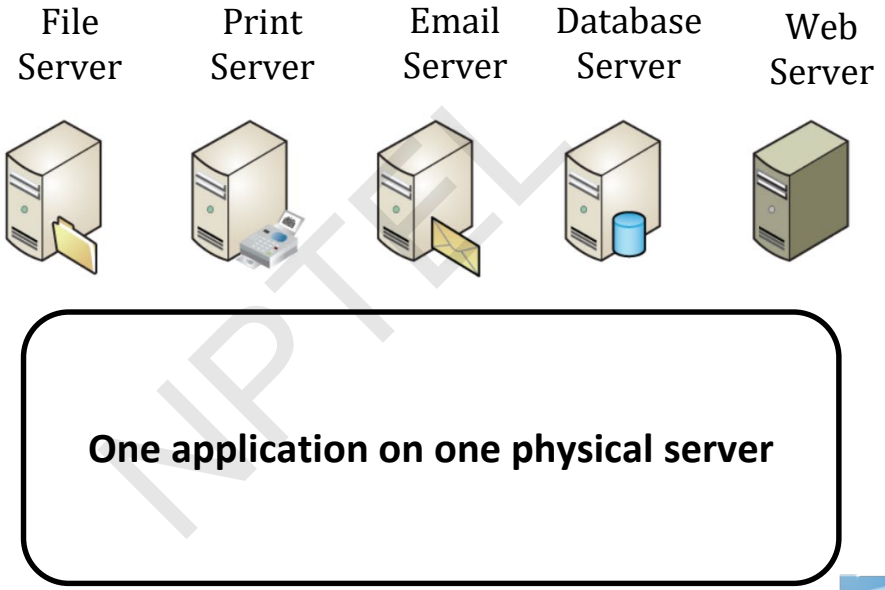
Dr. Rajiv Misra, Professor
Dept. of Computer Science & Engineering
Indian Institute of Technology Patna
rajivm@iitp.ac.in

Contents

- Modern Application Development Requirements
- Introduction to Containerization
- Containers v/s VMs – key comparisons
- Contains as a service (CaaS)
- Docker and its components

Historical limitations of application deployment

- Slow deployment
- High costs
- Underutilized resources
- Difficult to scale
- Difficult to migrate
- Vendor lock in

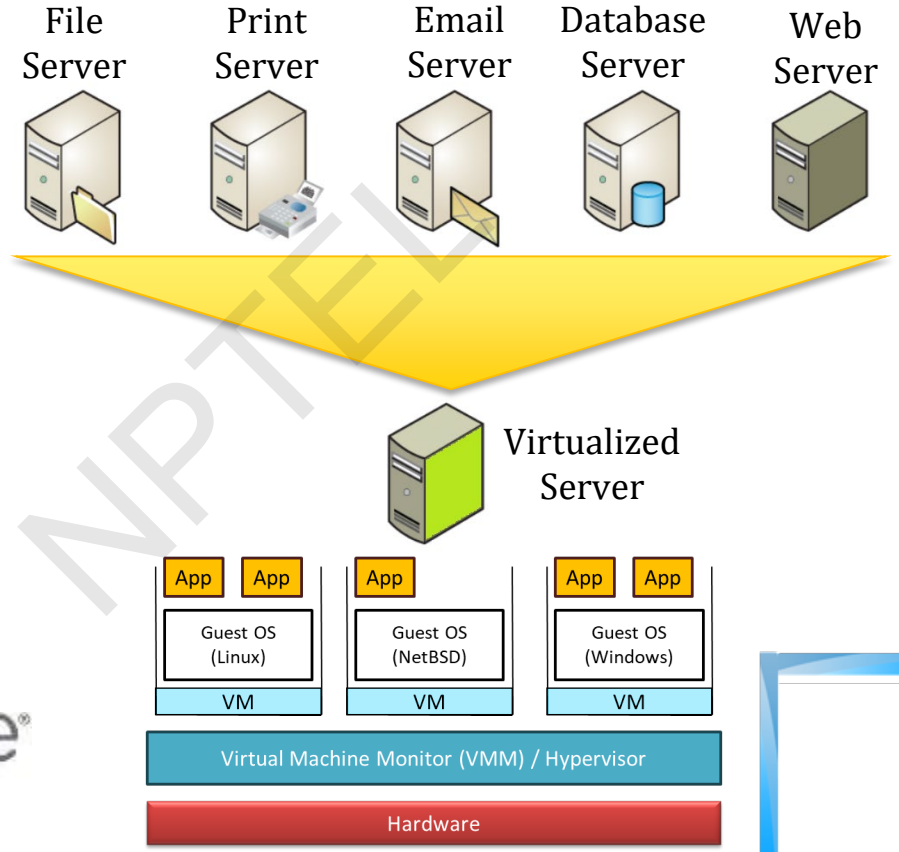


Historical limitations of application deployment

Hypervisor-based Virtualization

- One physical server can contain multiple applications
- Each application runs in a virtual machine (VM)
- Better resource pooling

- VMs in the cloud
  - Rapid elasticity
  - Pay as you go model

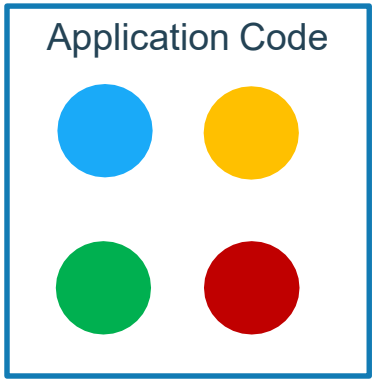


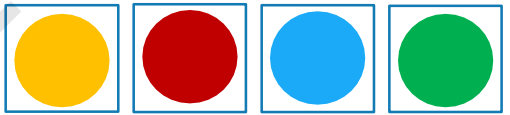
amazon web services™ vmware® Microsoft Azure

Limitations of VMs

- Each VM stills requires
  - CPU allocation
  - Storage
  - RAM
  - An entire guest operating system
- Running a full guest OS results in additional resources requirement
- Application portability not guaranteed

Application Modernization





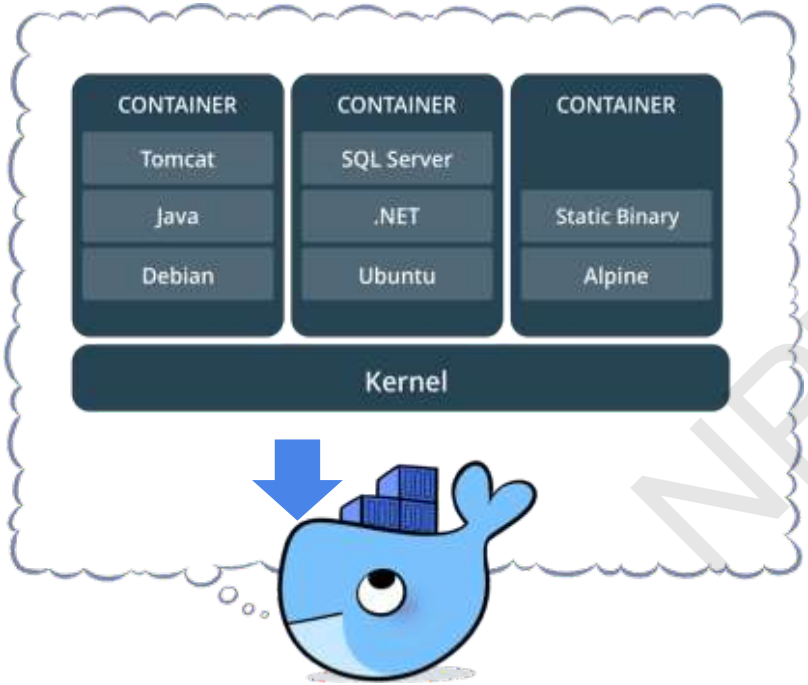
Micro services: Break application into separate operations

12-Factor Apps: Make the app independently scalable, stateless, highly available by design

Developer Issues:

- Minor code changes require full re-compile and re-test
- Application becomes single point of failure
- Application is difficult to scale

What is a container?



- Standardized packaging for software and dependencies
- Isolate apps from each other
- Share the same OS kernel
- Works with all major Linux and Windows Server

Figure Source: *https://blog.knoldus.com/what-is-docker/*

Comparing Containers and VMs

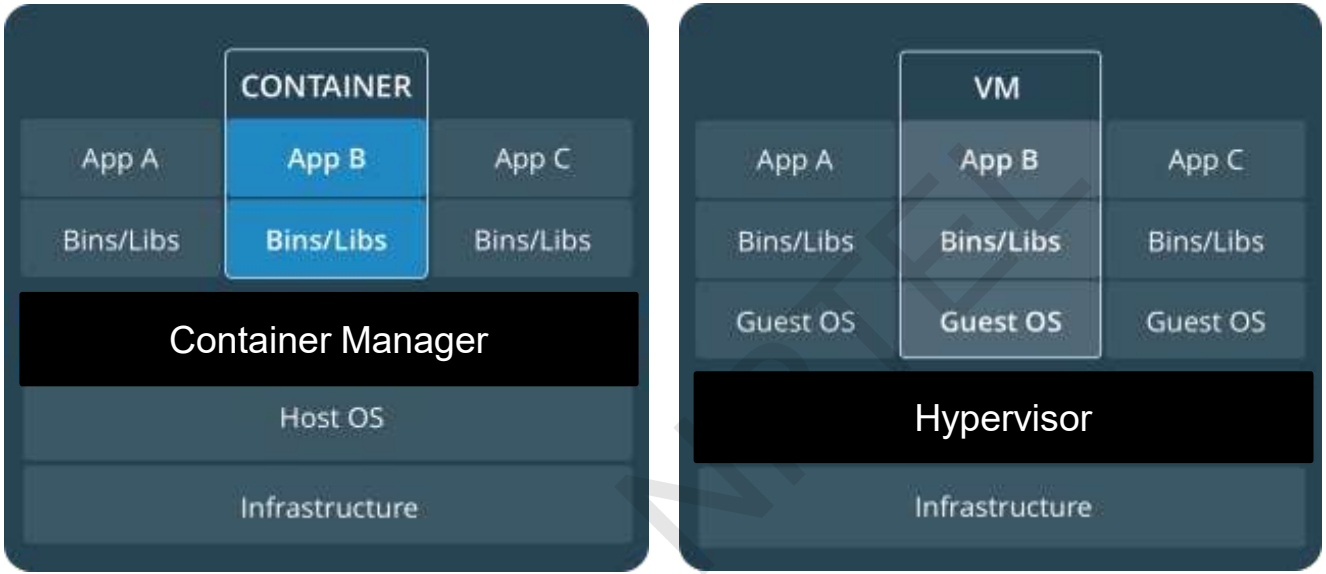


Figure Source: *https://blog.knoldus.com/what-is-docker/*

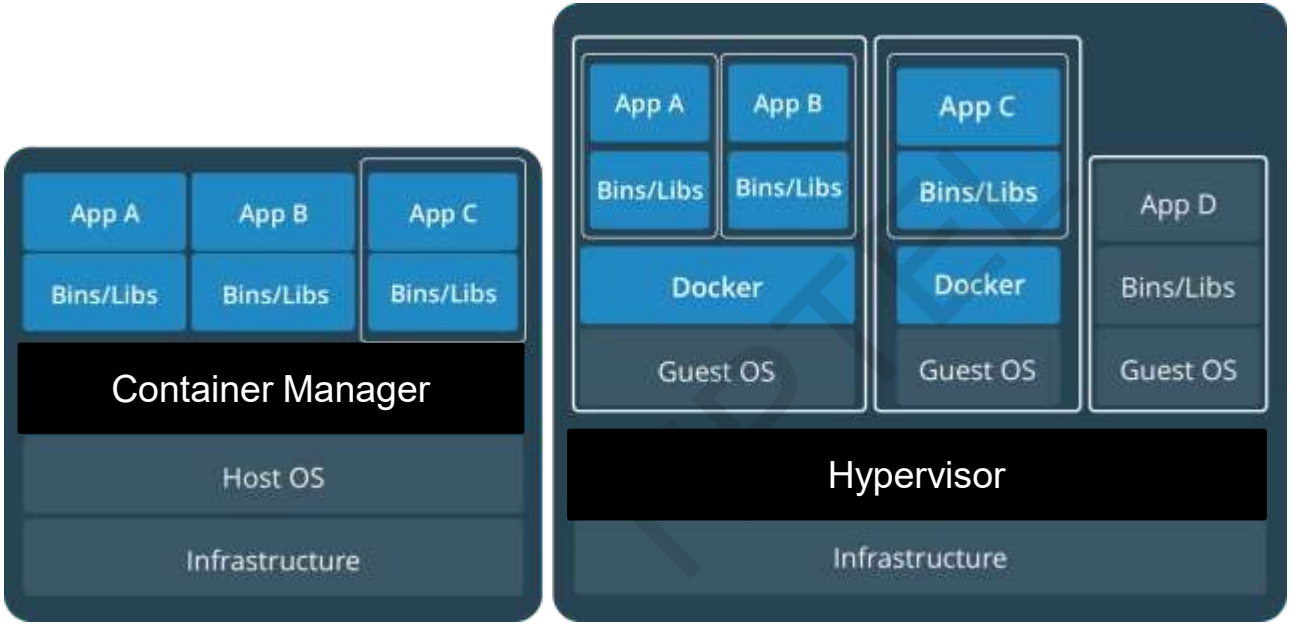
Container vs VM

	VMs	Containers
Abstraction Level	**Hardware Level:** VMs virtualize at the hardware level, creating a full virtualized environment, including an independent OS kernel for each VM.	**Operating System Level:** Containers virtualize at the operating system level, sharing the host OS kernel while isolating user space and processes.
Resource Efficiency	**Heavier:** VMs are heavier than containers, as they require a full operating system for each instance, leading to higher resource usage and longer startup times.	**Lightweight:** Containers are more lightweight than VMs, as they share the host OS kernel and require fewer resources. They can start quickly and have lower overhead.
Isolation	**Strong Isolation:** VMs provide stronger isolation since each VM has its own kernel and resources, making them more suitable for running different operating systems.	**Process Isolation:** Containers isolate applications at the process level, providing a level of separation between different containers. **Shared Kernel:** Containers share the host OS kernel, which makes them more efficient but may pose security challenges if not properly configured.

Container vs VM

	VMs	Containers
Portability	**Less Portable:** VMs are less portable than containers because they encapsulate the entire OS. Moving VMs between different hypervisors or cloud providers may require additional configuration.	**Consistency Across Environments:** Containers are highly portable and can run consistently across different environments as long as the host supports containerization.
Scaling	**Slower Scaling:** VMs generally have slower scaling capabilities due to their larger size and longer startup times.	**Rapid Scaling:** Containers can be quickly scaled up or down to meet changing demands, making them suitable for microservices architectures.
Boot Time	**Slower Startup:** VMs take longer to start as they need to boot an entire operating system.	**Fast Startup:** Containers start quickly since they don't need to boot an entire operating system.

Containers and VMs together



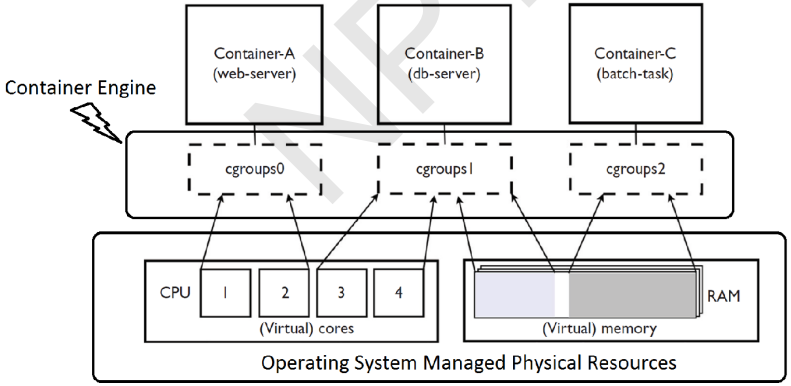
Containers and VMs together provide a tremendous amount of flexibility for IT to optimally deploy and manage apps.

Figure Source: https://blog.knoldus.com/what-is-docker/; https://www.docker.com/resources/what-container/#/virtual_machines

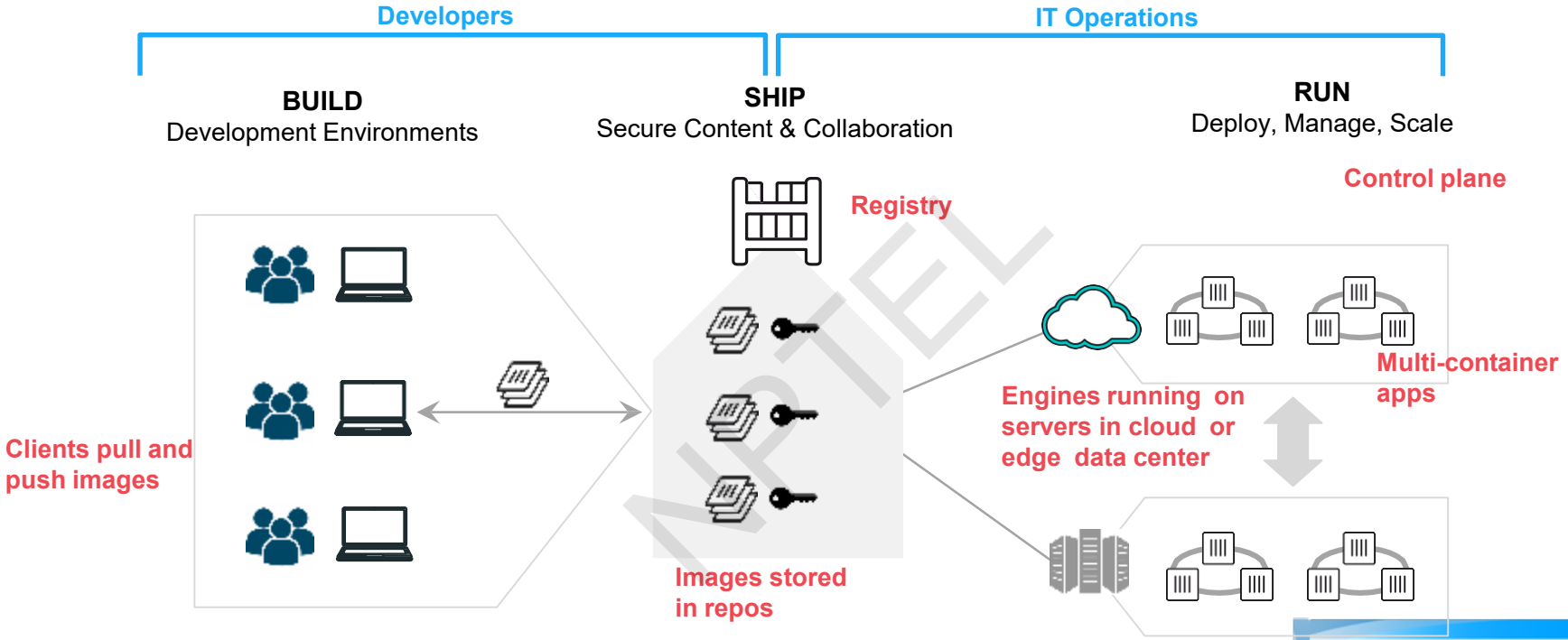
Resource sharing (CPU, Disk, I/o) across containers

Containers encapsulate applications and their dependencies, allowing them to run consistently across different environments. A container engine or a containerization software utilizes several mechanisms for resource sharing and management of containers, such as:

- **Kernel Namespace Isolation:** using separate kernel namespaces to provide process isolation.
- **Control Groups (cgroups):** Control groups allow the allocation of resource and setting up constrains on their usage i.e., to specify resource reservations and limits for containers.
- **CPU Resource Management:** controlling the amount of CPU resources allocated to containers.
- **Network Isolation:** Each container can have its own network namespace.
- **Volume Mounting:** containers can share directories with the host.
- **Dynamic Resource Adjustment:** dynamic adjustments to resource allocations.



Containers as a Service



What is Docker?

Docker is a platform for developing, shipping & running application using container-based virtualization technology.

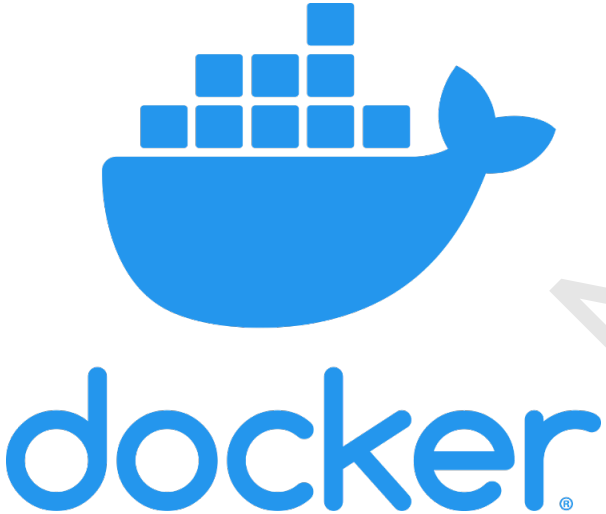


Figure Source: *https://www.docker.com/*

History of Containers

2004
Solaris Containers / Zones technology introduced

2008
Linux containers (LXC 1.0) introduced

2013
Solomon Hykes starts Docker as an internal project within dot Cloud

2013
Docker released to open source

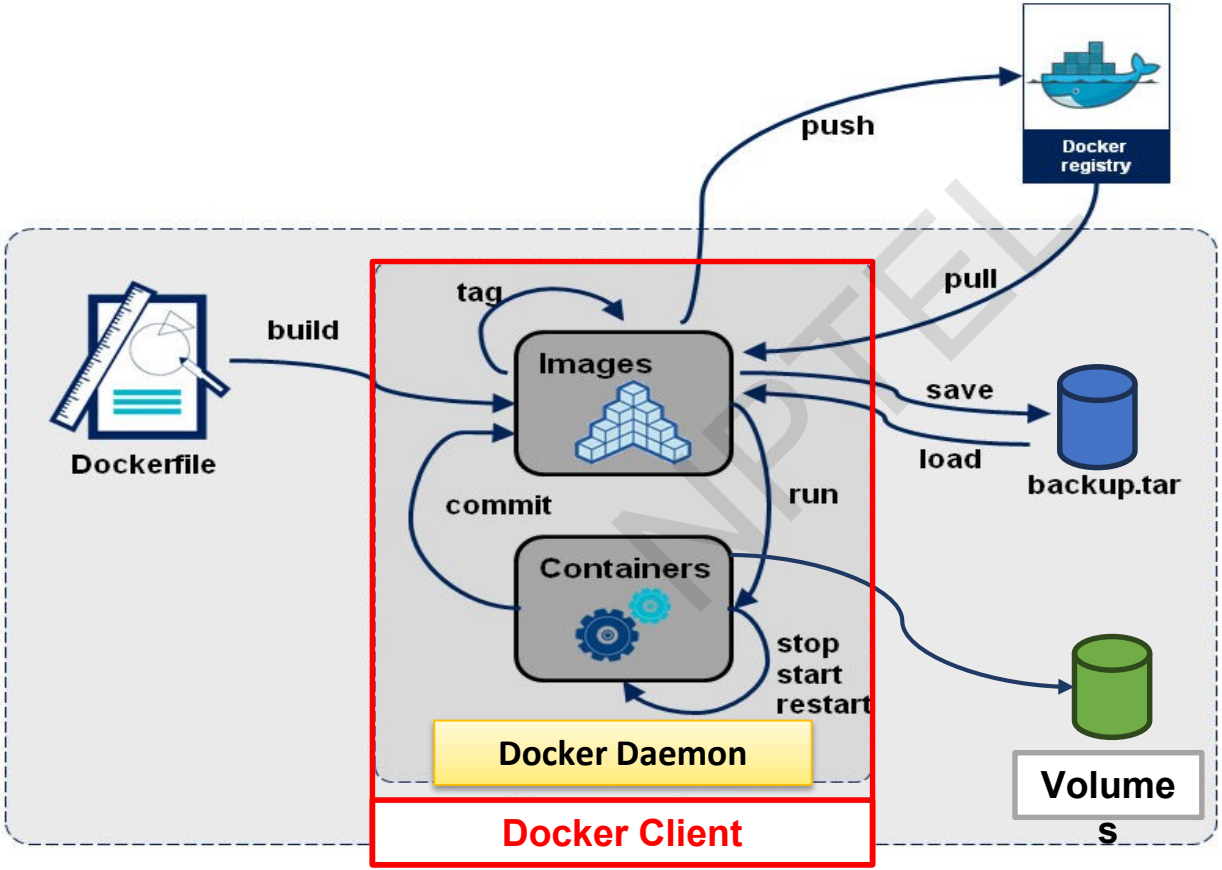
2016
Microsoft introduced Windows Containers,

Today
Open-source community includes 3,300+ contributors

Incredible adoption in just 4 years

Figure Source: https://www.code4projects.net/getting-started-with-docker/

Docker Components and Workflow



Docker Networking

- Containers can talk to each other without having to expose ports to host.
- Essential for micro service application architecture.
- Example:

  -Container with Tomcat running

  -Container with MySQL running

  -Application on tomcat needs to connect to MySQL

- Each container is assigned a virtual network interface
- External network connectivity is provided through a NAT

Benefits of using Docker

Isolation

Portability

Lightweight

Reproducibility

Versioning and Rollback

Modularity

Scalability

Security

Efficient Resource Utilization

Docker Users

Conclusion

- In this lecture
- we discussed containerization technology and its comparision to hypervisor based virtual machines
- we covered the advantages of using containers in application development cycle
- we discussed about Docker – a containerization software; and its major components
- we discussed the development workflow with docker which can provide operation efficiency for service providers and developers.

Thank You!

References

- *Container Networking, From Docker to Kubernetes - Michael Hausenblas, O'Reilly Media (2018)*
- *Ian Miell, Aiden Hobson Sayers - Docker in Practice-Manning Publications (2019)*
- *Kubernetes in Action, Manning Publications, Marko Lukša*