UNIT-2

VIRTUALIZATION

1. What is Virtualization and Types ?

Virtualization is a computer architecture technology by which multiple virtual machines (VMs) are multiplexed in the same hardware machine.

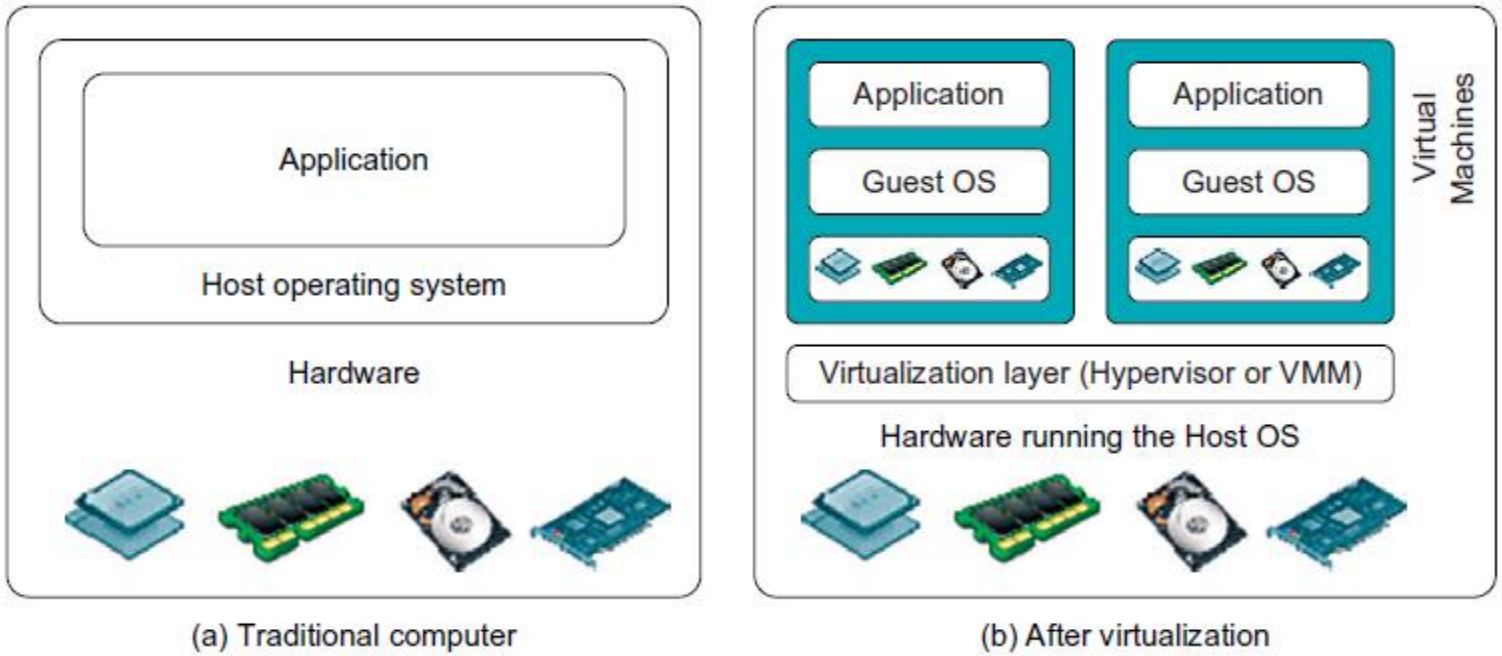
Virtualization is a technology that helps us to install different Operating Systems on a hardware. They are completely separated and independent from each other.

Virtualization hides the physical characteristics of computing resources from their users, their applications or end users. This includes making a single physical resource (such as a server, an operating system, an application or a storage device) appear to function as multiple virtual resources.

It can also include making multiple physical resources (such as storage devices or servers) appear as a single virtual resource...”

Virtualization is often −

- The creation of many virtual resources from one physical resource.
- The creation of one virtual resource from one or more physical resource.



(a) Traditional computer

(b) After virtualization

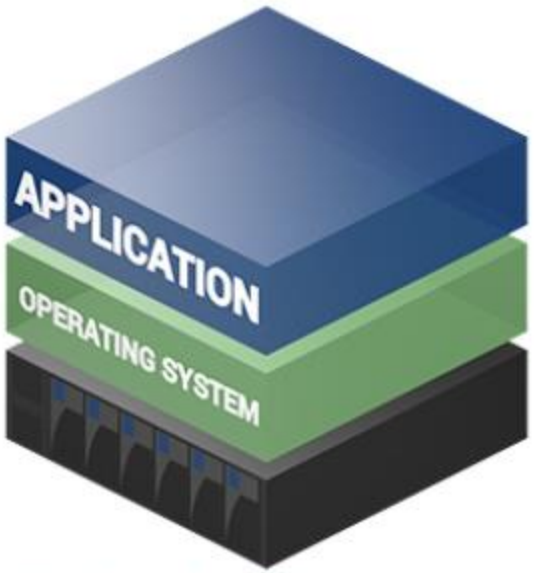
- The purpose of a VM is to enhance resource sharing by many users and improve computer performance in terms of resource utilization and application flexibility.
- Hardware resources (CPU, memory,I/O devices, etc.) or software resources (operating system and software libraries) can be virtualized in various functional layers.
- This virtualization technology has been revitalized as the demand for distributed and cloud computing

Types of Virtualization

- Server Virtualization
- Client & Desktop Virtualization
- Services and Applications Virtualization
- Network Virtualization
- Storage Virtualization

Server Virtualization

It is virtualizing your server infrastructure where you do not have to use any more physical servers for different purposes.

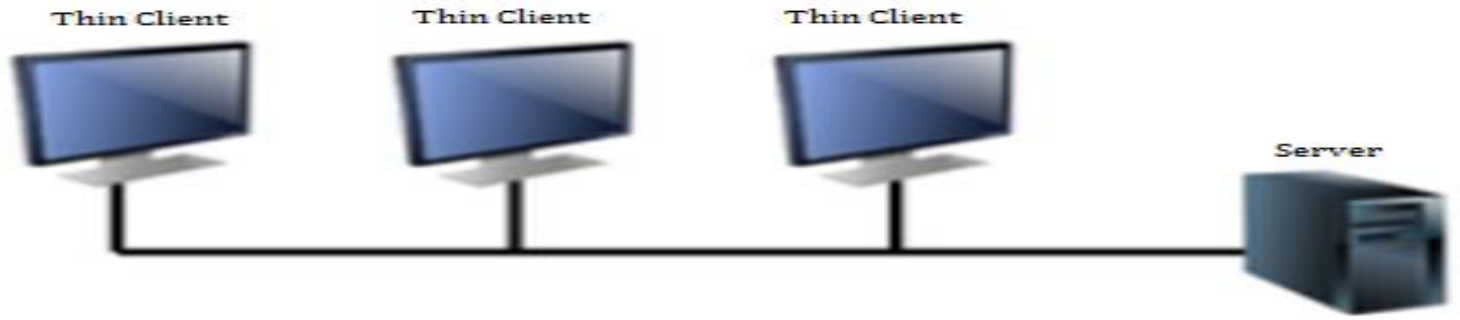


Clasic Server Installation

Virtualized server Intallation

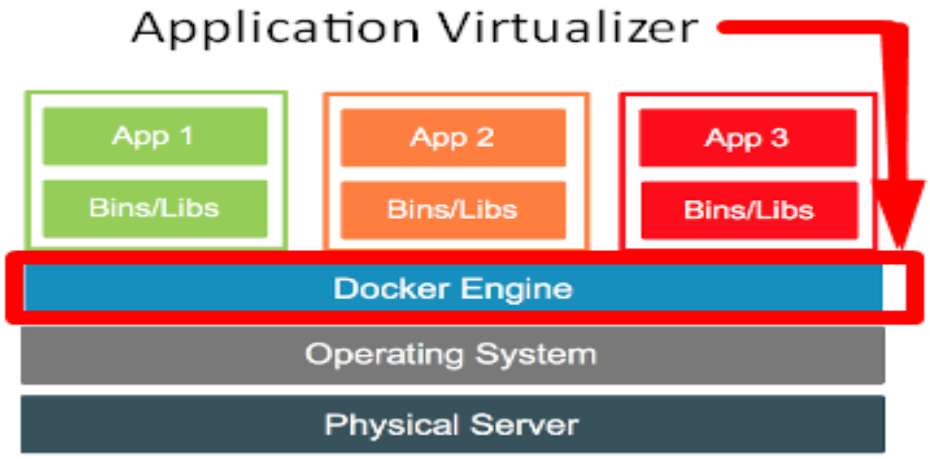
Client & Desktop Virtualization

This is similar to server virtualization, but this time is on the user's site where you virtualize their desktops. We change their desktops with thin clients and by utilizing the datacenter resources.



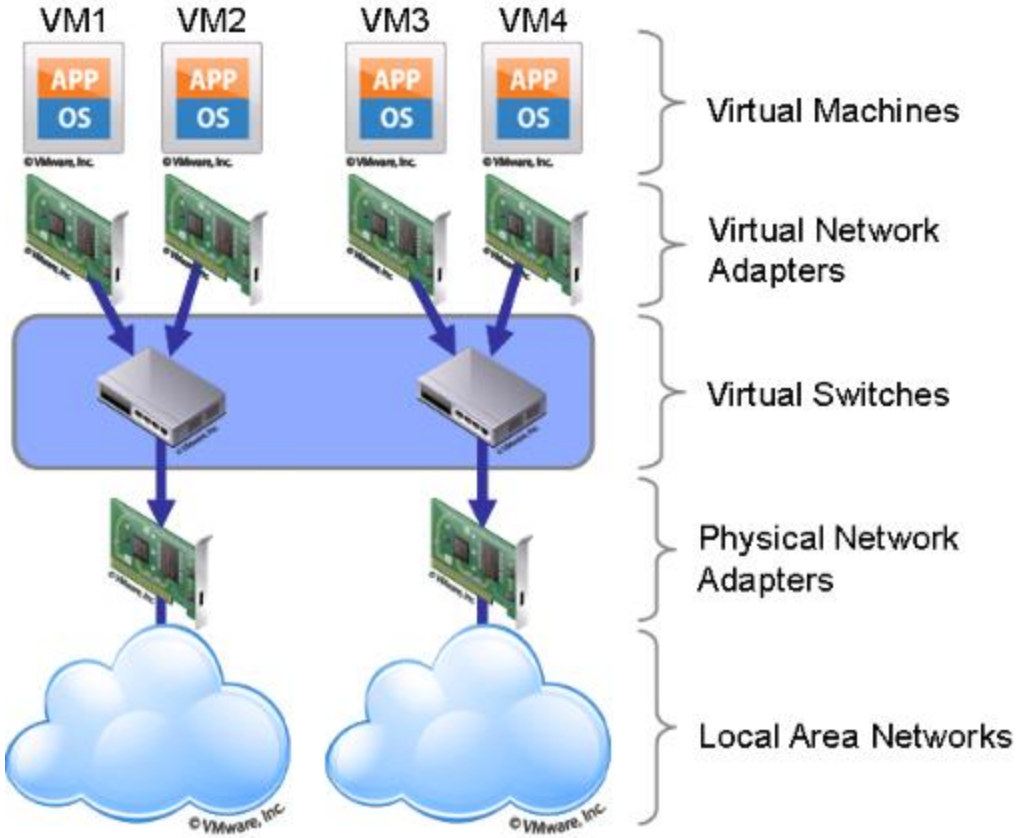
Services and Applications Virtualization

The virtualization technology isolates applications from the underlying operating system and from other applications, in order to increase compatibility and manageability. For example – Docker can be used for that purpose.



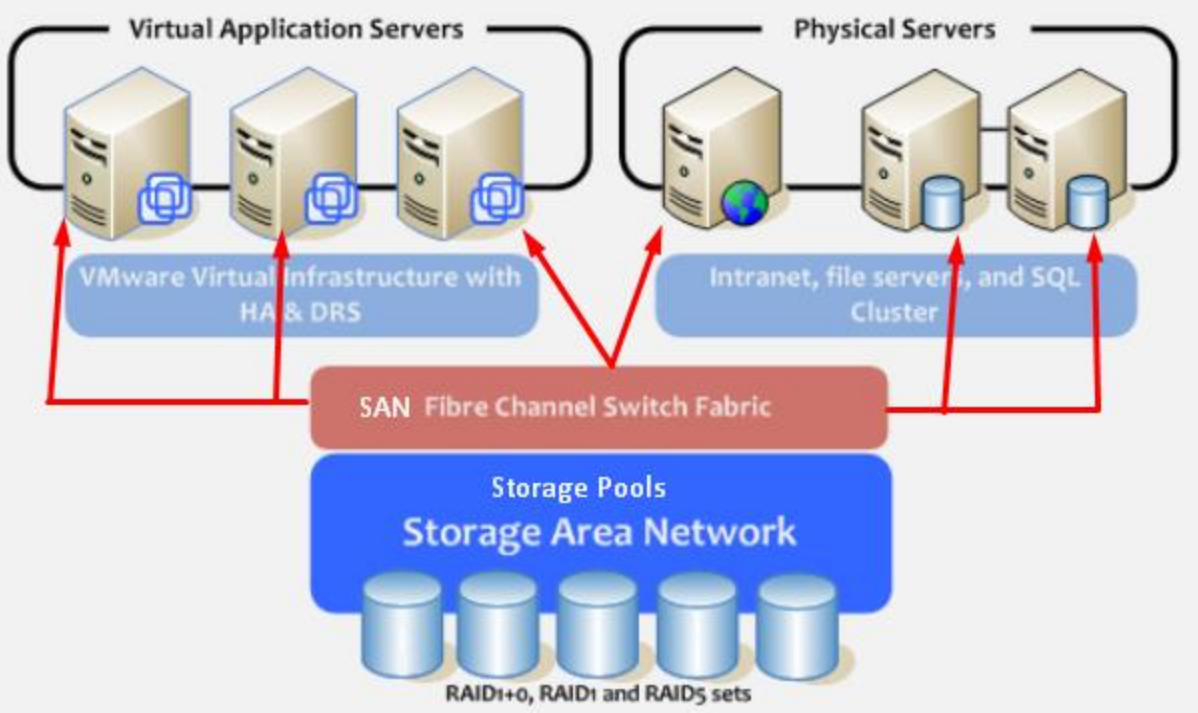
Network Virtualization

It is a part of virtualization infrastructure, which is used especially if you are going to visualize your servers. It helps you in creating multiple switching, Vlans, NAT-ing, etc. The following illustration shows the VMware schema −



Storage Virtualization

This is widely used in datacenters where you have a big storage and it helps you to create, delete, allocated storage to different hardware. This allocation is done through network connection. The leader on storage is SAN. A schematic illustration is given below −



2. What is Hypervisor and types of Hypervisors

A hypervisor is a thin software layer that intercepts operating system calls to the hardware. It is also called as the **Virtual Machine Monitor** (VMM). It creates a virtual platform on the host computer, on top of which multiple guest operating systems are executed and monitored.

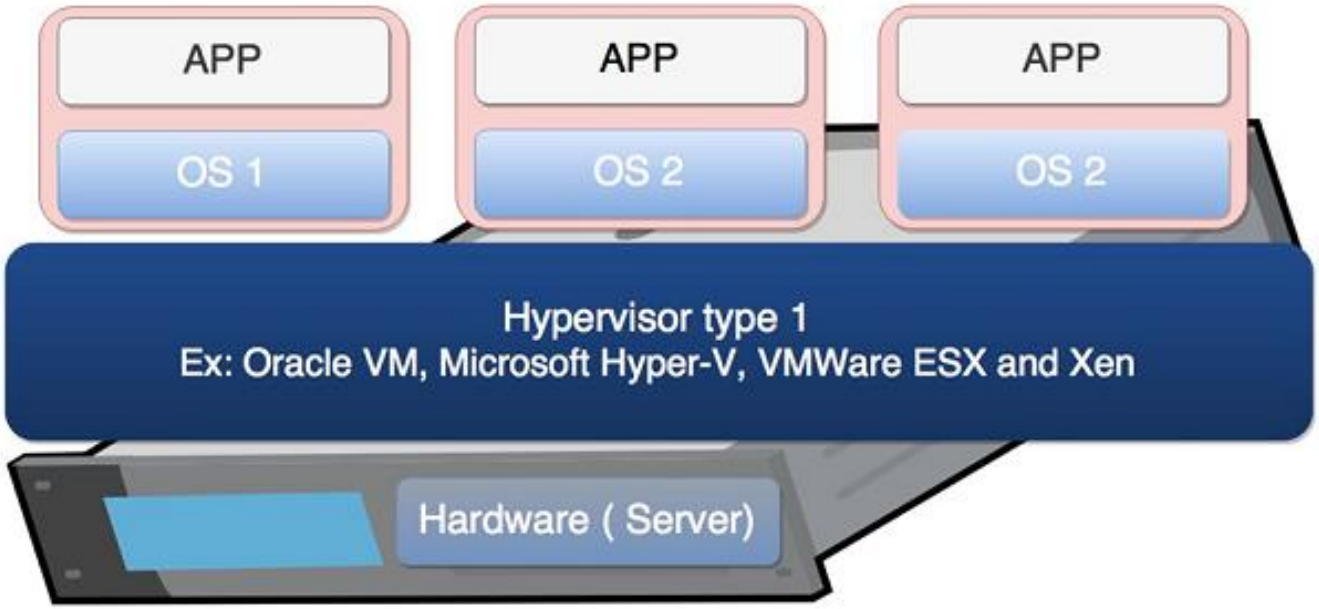
Hypervisors are two types –

- Native of Bare Metal Hypervisor and
- Hosted Hypervisor

Native or Bare Metal Hypervisor

Native hypervisors are software systems that run directly on the host's hardware to control the hardware and to monitor the **Guest Operating Systems**. The guest operating system runs on a separate level above the hypervisor. All of them have a Virtual Machine Manager.

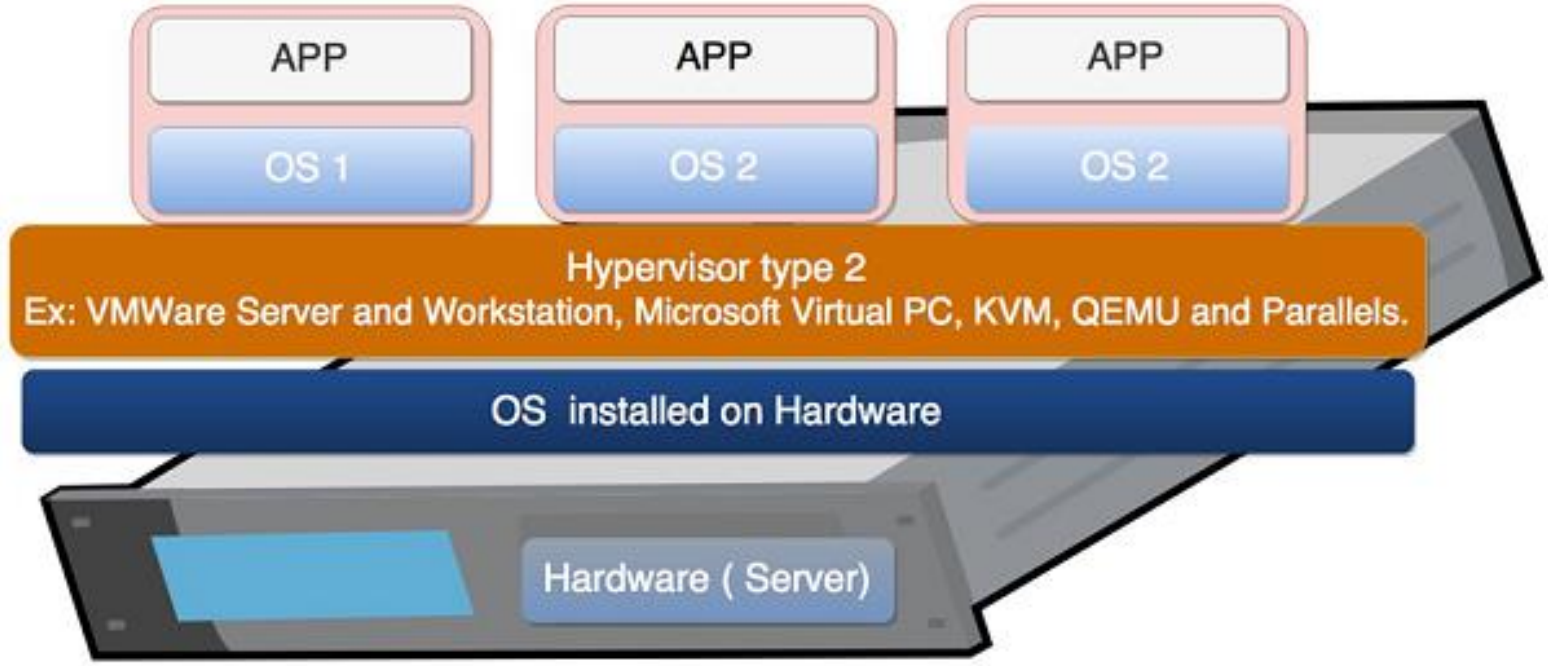
Examples of this virtual machine architecture are **Oracle VM, Microsoft Hyper-V, VMWare ESX** and **Xen**.



Hosted Hypervisor

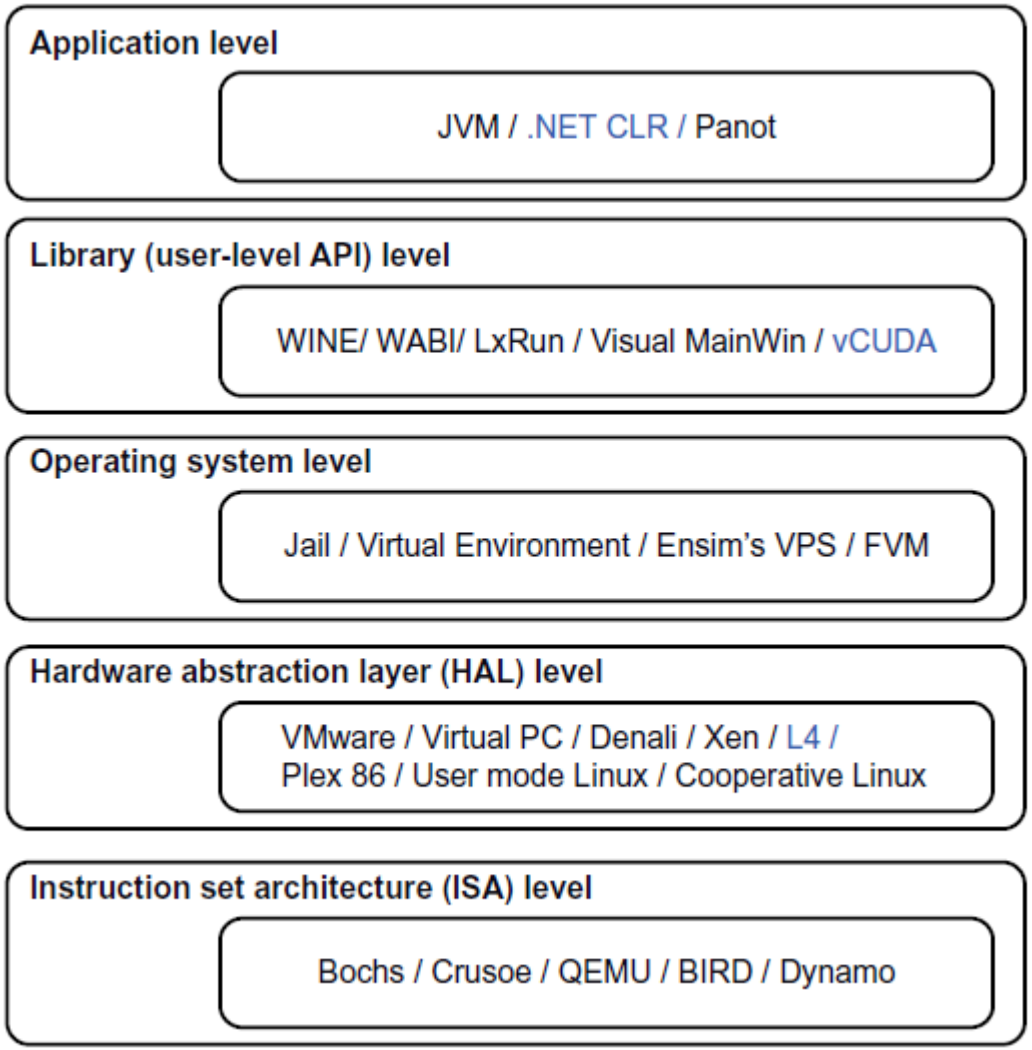
Hosted hypervisors are designed to run within a traditional operating system. In other words, a hosted hypervisor adds a distinct software layer on top of the host operating system. While, the guest operating system becomes a third software level above the hardware.

A well-known example of a hosted hypervisor is **Oracle VM VirtualBox**. Others include **VMWare Server and Workstation, Microsoft Virtual PC, KVM, QEMU** and **Parallels**.



3. Discuss about Levels of Virtualization Implementation

The main function of the software layer for virtualization is to virtualize the physical hardware of a host machine into virtual resources to be used by the VMs, exclusively. This can be implemented at various operational levels.



Instruction Set Architecture Level

At the ISA level, virtualization is performed by emulating a given ISA by the ISA of the host machine. For example, MIPS binary code can run on an x86-based host machine with the help of ISA emulation.

With this approach, it is possible to run a large amount of legacy binary code written for various processors on any given new hardware host machine. Instruction set emulation leads to virtual ISAs created on any hardware machine.

- The basic emulation method is through code interpretation. An interpreter program interprets the source instructions to target instructions one by one.
- One source instruction may require tens or hundreds of native target instructions to perform its function. Obviously, this process is relatively slow.
- For better performance, dynamic binary translation is desired. This approach translates basic blocks of dynamic source instructions to target instructions.

Instruction set emulation requires binary translation and optimization. A virtual instruction set architecture (V-ISA) thus requires adding a processor-specific software translation layer to the compiler.

Hardware Abstraction Level

Hardware-level virtualization is performed right on top of the bare hardware. On the one hand, this approach generates a virtual hardware environment for a VM.

On the other hand, the process manages the underlying hardware through virtualization. The idea is to virtualize a computer's resources, such as its processors, memory, and I/O devices.

The idea was implemented in the IBM VM/370 in the 1960s. More recently, the Xen hypervisor has been applied to virtualize x86-based machines to run Linux or other guest OS applications.

Operating System Level

This refers to an abstraction layer between traditional OS and user applications.

OS-level virtualization creates isolated containers on a single physical server and the OS instances to utilize the hardware and software in data centers.

The containers behave like real servers. OS-level virtualization is commonly used in creating virtual hosting environments to allocate hardware resources among a large number of mutually distrusting users.

It is also used, to a lesser extent, in consolidating server hardware by moving services on separate hosts into containers or VMs on one server

Library Support Level

Most applications use APIs exported by user-level libraries rather than using lengthy system calls by the OS.

Since most systems provide well-documented APIs, such an interface becomes another candidate for virtualization.

Virtualization with library interfaces is possible by controlling the communication link between applications and the rest of a system through API hooks.

The software tool WINE has implemented this approach to support Windows applications on top of UNIX hosts.

Another example is the vCUDA which allows applications executing within VMs to leverage GPU hardware acceleration.

User-Application Level

Virtualization at the application level virtualizes an application as a VM. On a traditional OS, an application often runs as a process. Therefore, application-level virtualization is also known as process-level virtualization.

The most popular approach is to deploy high level language (HLL) VMs.

- In this scenario, the virtualization layer sits as an application program on top of the operating system, and the layer exports an abstraction of a VM that can run programs written and compiled to a particular abstract machine definition.

- Any program written in the HLL and compiled for this VM will be able to run on it. The Microsoft .NET CLR and Java Virtual Machine (JVM) are two good examples of this class of VM.

- Other forms of application-level virtualization are known as application isolation, application sandboxing, or application streaming.

- The process involves wrapping the application in a layer that is isolated from the host OS and other applications.

Table 3.1 Relative Merits of Virtualization at Various Levels (More "X"'s Means Higher Merit, with a Maximum of 5 X's)

Level of Implementation	Higher Performance	Application Flexibility	Implementation Complexity	Application Isolation
ISA	X	XXXX	XXX	XXX
Hardware-level virtualization	XXXX	XXX	XXXX	XXX
OS-level virtualization	XXXX	XX	XXX	XX
Runtime library support	XXX	XX	XX	XX
User application level	XX	XX	XXXX	XXXX

4. List out VMM Design Requirements and Providers

Hardware-level virtualization inserts a layer between real hardware and traditional operating systems. This layer is commonly called the Virtual Machine Monitor (VMM) and it manages the hardware resources of a computing system.

Each time programs access the hardware the VMM captures the process. In this sense, the VMM acts as a traditional OS.

One hardware component, such as the CPU, can be virtualized as several virtual copies. Therefore, several traditional operating systems which are the same or different can sit on the same set of hardware simultaneously

There are three requirements for a VMM.

- First, a VMM should provide an environment for programs which is essentially identical to the original machine.
- Second, programs run in this environment should show, at worst, only minor decreases in speed.
- Third, a VMM should be in complete control of the system resources. Any program run under a VMM should exhibit a function identical to that which it runs on the original machine directly.

VMM includes the following aspects:

(1) The VMM is responsible for allocating hardware resources for programs;

(2) it is not possible for a program to access any resource not explicitly allocated to it; and

(3) it is possible under certain circumstances for a VMM to regain control of resources already allocated.

Table 3.2 Comparison of Four VMM and Hypervisor Software Packages

Provider and References	Host CPU	Host OS	Guest OS	Architecture
VMware Workstation [71]	x86, x86-64	Windows, Linux	Windows, Linux, Solaris, FreeBSD, Netware, OS/2, SCO, BeOS, Darwin	Full Virtualization
VMware ESX Server [71]	x86, x86-64	No host OS	The same as VMware Workstation	Para-Virtualization
Xen [7,13,42]	x86, x86-64, IA-64	NetBSD, Linux, Solaris	FreeBSD, NetBSD, Linux, Solaris, Windows XP and 2003 Server	Hypervisor
KVM [31]	x86, x86-64, IA-64, S390, PowerPC	Linux	Linux, Windows, FreeBSD, Solaris	Para-Virtualization

5. Discuss about Virtualization Support at the OS Level

Cloud computing has at least two challenges regarding Virtualization.

- The first is the ability to use a variable number of physical machines and VM instances depending on the needs of a problem. For example, a task may need only a single CPU during some phases of execution but may need hundreds of CPUs at other times.

- The second challenge concerns the slow operation of instantiating new VMs. Currently, new VMs originate either as fresh boots or as replicates of a template VM, unaware of the current application state

Why OS-Level Virtualization?

- It is slow to initialize a hardware-level VM because each VM creates its own image from scratch.

- In a cloud computing environment, perhaps thousands of VMs need to be initialized simultaneously.

- Besides slow operation, storing the VM images also becomes an issue.

- OS-level virtualization provides a feasible solution for these hardware-level virtualization issues.

Operating system virtualization inserts a virtualization layer inside an operating system to partition a machine's physical resources.

It enables multiple isolated VMs within a single operating system kernel. This kind of VM is often called a virtual execution environment (VE), Virtual Private System (VPS), or simply container.

- From the user's point of view, VEs look like real servers.
- This means a VE has its own set of processes, file system, user accounts, network interfaces with IP addresses, routing tables, firewall rules, and other personal settings.
- Although VEs can be customized for different people, they share the same operating system kernel.
- Therefore, OS-level virtualization is also called single-OS image virtualization.

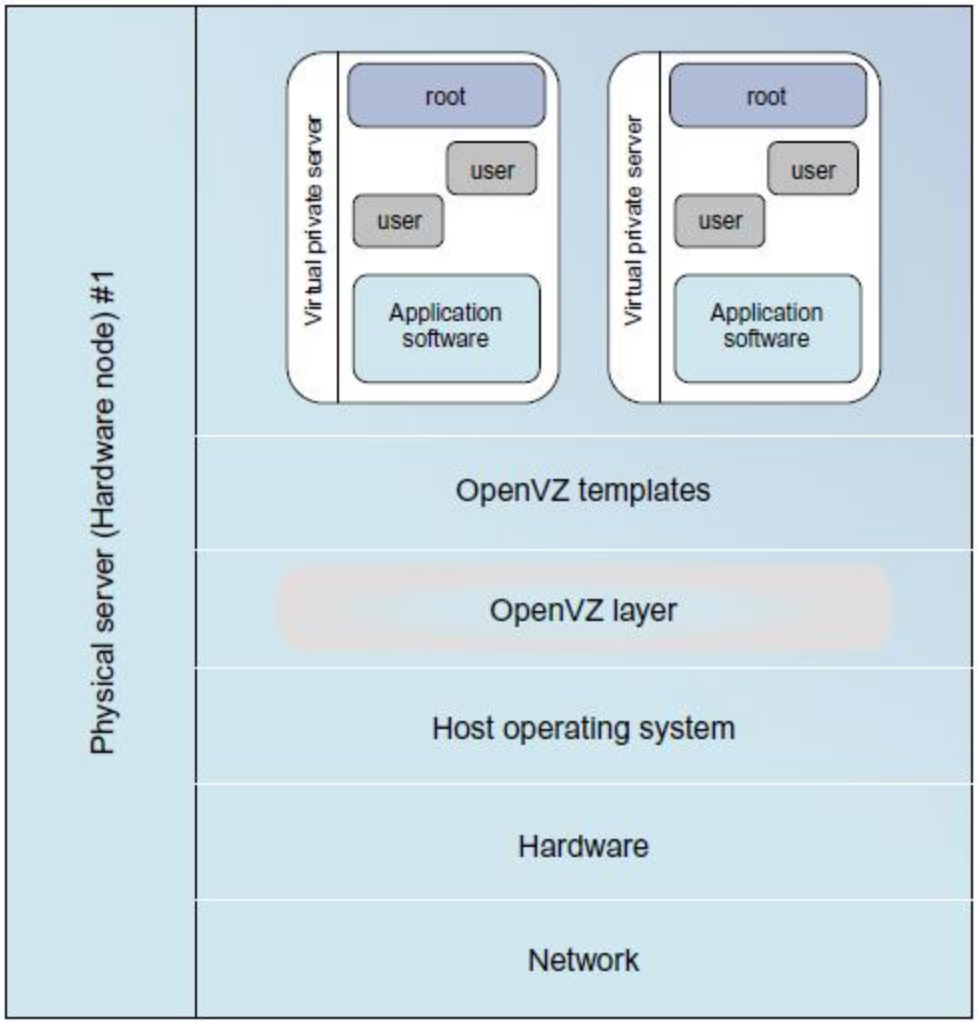


FIGURE 3.3

The OpenVZ virtualization layer inside the host OS, which provides some OS images to create VMs quickly.

(*Courtesy of OpenVZ User's Guide [65]*)

Advantages of OS Extensions

Compared to hardware-level virtualization, the benefits of OS extensions are twofold:

(1) VMs at the operating system level have minimal startup/shutdown costs, low resource requirements, and high scalability;
(2) for an OS-level VM, it is possible for a VM and its host environment to synchronize state changes when necessary.

These benefits can be achieved via two mechanisms of OS-level virtualization:

(1) All OS-level VMs on the same physical machine share a single operating system kernel; and

(2) the virtualization layer can be designed in a way that allows processes in VMs to access as many resources of the host machine as possible, but never to modify them.

The main disadvantage of OS extensions is that all the VMs at operating system level on a single container must have the same kind of guest operating system. That is, although different OS-level VMs may have different operating system distributions, they must pertain to the same operating system family.

Middleware Support for Virtualization

Library-level virtualization is also known as user-level Application Binary Interface (ABI) or API emulation.

This type of virtualization can create execution environments for running alien programs on a platform rather than creating a VM to run the entire operating system. API call interception and remapping are the key functions performed

Table 3.4 Middleware and Library Support for Virtualization

Middleware or Runtime Library and References or Web Link	Brief Introduction and Application Platforms
WABI (http://docs.sun.com/app/docs/doc/802-6306)	Middleware that converts Windows system calls running on x86 PCs to Solaris system calls running on SPARC workstations
Lxrun (Linux Run) (http://www.ugcs.caltech.edu/~steven/lxrun/)	A system call emulator that enables Linux applications written for x86 hosts to run on UNIX systems such as the SCO OpenServer
WINE (http://www.winehq.org/)	A library support system for virtualizing x86 processors to run Windows applications under Linux, FreeBSD, and Solaris
Visual MainWin (http://www.mainsoft.com/)	A compiler support system to develop Windows applications using Visual Studio to run on Solaris, Linux, and AIX hosts
vCUDA (Example 3.2) (IEEE *IPDPS* 2009 [57])	Virtualization support for using general-purpose GPUs to run data-intensive applications under a special guest OS

5. Discuss about various VIRTUALIZATION STRUCTURES/TOOLS AND MECHANISMS

There are three typical classes of VM **architecture**

- Before virtualization, the operating system manages the hardware.
- After virtualization, a virtualization layer is inserted between the hardware and the operating system.
- In such a case, the virtualization layer is responsible for converting portions of the real hardware into virtual hardware.
- Therefore, different operating systems such as Linux and Windows can run on the same physical machine.

Depending on the position of the virtualization layer, there are several classes of VM architectures, namely the

- Hypervisor architecture
- Para-virtualization
- Host-based virtualization

a)Hypervisor and Xen Architecture

- The hypervisor supports hardware-level virtualization (see Figure 3.1(b)) on bare metal devices like CPU, memory, disk and network interfaces.
- The hypervisor software sits directly between the physical hardware and its OS.
- This virtualization layer is referred to as either the VMM or the hypervisor.
- The hypervisor provides hypercalls for the guest OSes and applications.
- Depending on the functionality, a hypervisor can assume a micro-kernel architecture like the Microsoft Hyper-V. Or it can assume a monolithic hypervisor architecture like the VMware ESX for server virtualization.

A micro-kernel hypervisor includes only the basic and unchanging functions (such as physical memory management and processor scheduling). The device drivers and other changeable components are outside the hypervisor.

- **A monolithic hypervisor** implements all the aforementioned functions, including those of the device drivers. Therefore, the size of the hypervisor code of a micro-kernel hypervisor is smaller than that of a monolithic hypervisor.

- Essentially, a hypervisor must be able to convert physical devices into virtual resources dedicated for the deployed VM to use.

The Xen Architecture

- Xen is an open source hypervisor program developed by Cambridge University. Xen is a microkernel hypervisor, which separates the policy from the mechanism.

- The Xen hypervisor implements all the mechanisms, leaving the policy to be handled by Domain 0.

- Xen does not include any device drivers natively . It just provides a mechanism by which a guest OS can have direct access to the physical devices.

- As a result, the size of the Xen hypervisor is kept rather small.

- Xen provides a virtual environment located between the hardware and the OS.

- A number of vendors are in the process of developing commercial Xen hypervisors, among them are Citrix XenServer [62] and Oracle VM [42].

- The core components of a Xen system are the hypervisor, kernel, and applications. The organization of the three components is important.

- Like other virtualization systems, many guest Oses can run on top of the hypervisor.

- However, not all guest OSes are created equal, and one in particular controls the others. The guest OS, which has control ability, is called Domain 0, and the others are called Domain U.

- Domain 0 is a privileged guest OS of Xen.

- It is first loaded when Xen boots without any file system drivers being available. Domain 0 is designed to access hardware directly and manage devices.

- Therefore, one of the responsibilities of Domain 0 is to allocate and map hardware resources for the guest domains (the Domain U domains).

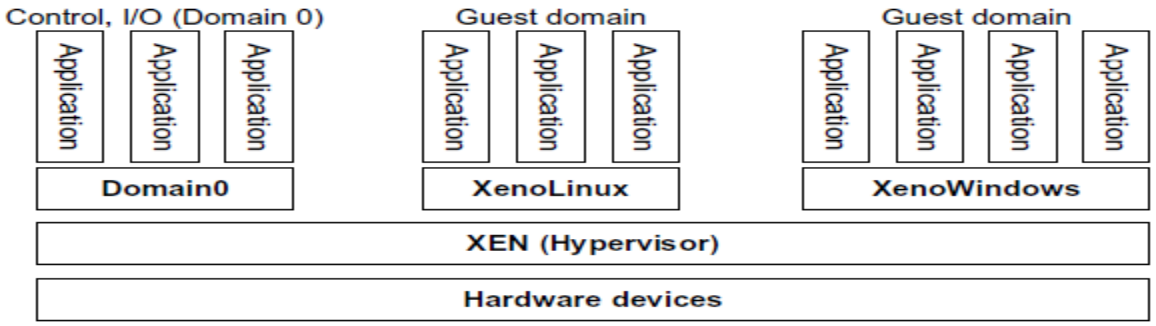


FIGURE 3.5

KMMIPS :: TIRUPATI

The Xen architecture's special domain 0 for control and I/O, and several guest domains for user applications.

- Its management VM is named Domain 0, which has the privilege to manage other VMs implemented on the same host.

- If Domain 0 is compromised, the hacker can control the entire system. So, in the VM system, security policies are needed to improve the security of Domain 0.

- Domain 0, behaving as a VMM, allows users to create, copy, save, read, modify, share, migrate, and roll back VMs as easily as manipulating a file, which flexibly provides tremendous benefits for users.

- Unfortunately, it also brings a series of security problems during the software life cycle and data lifetime.

b)Hardware virtualization can be classified into two categories:

- Full virtualization
- Host-based virtualization.

Full virtualization does not need to modify the host OS.

It relies on binary translation to trap and to virtualize the execution of certain sensitive, non-virtualizable instructions .

The guest OS's and their applications consist of noncritical and critical instructions.

In a host-based system, both a host OS and a guest OS are used. A virtualization software layer is built between the host OS and guest OS.

Full Virtualization

With full virtualization, noncritical instructions run on the hardware directly while critical instructions are discovered and replaced with traps into the VMM to be emulated by software.

Both the hypervisor and VMM approaches are considered full virtualization.

Why are only critical instructions trapped into the VMM?

- This is because binary translation can incur a large performance overhead.
- Noncritical instructions do not control hardware or threaten the security of the system, but
- critical instructions do.
- Therefore, running noncritical instructions on hardware not only can promote efficiency, but also can ensure system security.

Binary Translation of Guest OS Requests Using a VMM

This approach was implemented by VMware and many other software companies.

- VMware puts the VMM at Ring 0 and the guest OS at Ring 1.
- The VMM scans the instruction stream and identifies the privileged, control- and behavior-sensitive instructions.
- When these instructions are identified, they are trapped into the VMM, which emulates the behavior of these instructions.
- The method used in this emulation is called binary translation.
- Therefore, full virtualization combines binary translation and direct execution.
- The guest OS is completely decoupled from the underlying hardware.

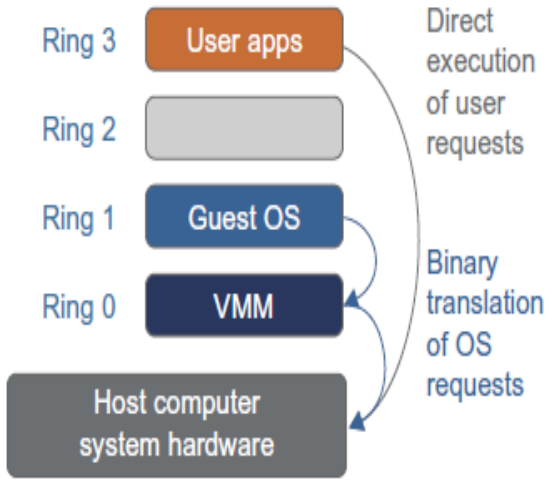


FIGURE 3.6

Indirect execution of complex instructions via binary translation of guest OS requests using the VMM plus direct execution of simple instructions on the same host.

- The performance of full virtualization may not be ideal, because it involves binary translation which is rather time-consuming.

- In particular, the full virtualization of I/O-intensive applications is a really a big challenge.

- Binary translation employs a code cache to store translated hot instructions
- to improve performance, but it increases the cost of memory usage

Host-Based Virtualization

- VM architecture is to install a virtualization layer on top of the host OS. This host OS is still responsible for managing the hardware.
- The guest OSes are installed and run on top of the virtualization layer.
- Dedicated applications may run on the VMs.

This host based architecture has some distinct advantages.

- First, the user can install this VM architecture without modifying the host OS.

The virtualizing software can rely on the host OS to provide device drivers and other low-level services. This will simplify the VM design and ease its deployment.

- Second, the host-based approach appeals to many host machine configurations.

Compared to the hypervisor/VMM architecture, the performance of the host-based architecture may also be low.

- When an application requests hardware access, it involves four layers of mapping which downgrades performance significantly.

- When the ISA of a guest OS is different from the ISA of the underlying hardware, binary translation must be adopted.

- Although the host-based architecture has flexibility, the performance is too low to be useful in practice.

c)Para-Virtualization with Compiler Support

- Para-virtualization needs to modify the guest operating systems.
- A Para-virtualized VM provides special APIs requiring substantial OS modifications in user applications.
- Performance degradation is a critical issue of a virtualized system. No one wants to use a VM if it is much slower than using a physical machine.
- The virtualization layer can be inserted at different positions in a machine software stack.
- However, Para-virtualization attempts to reduce the virtualization overhead, and thus improve performance by modifying only the guest OS kernel.

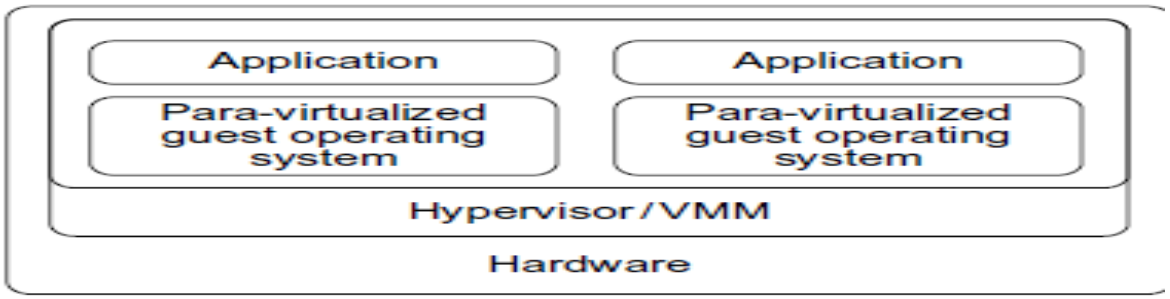


FIGURE 3.7

Para-virtualized VM architecture, which involves modifying the guest OS kernel to replace nonvirtualizable instructions with hypercalls for the hypervisor or the VMM to carry out the virtualization

Although para-virtualization reduces the overhead, it has incurred other problems.

- First, its compatibility and portability may be in doubt, because it must support the unmodified OS as well.
- Second, the cost of maintaining para-virtualized OSes is high, because they may require deep OS kernel modifications.
- Finally, the performance advantage of para-virtualization varies greatly due to workload variations.

- The main problem in full virtualization is its low performance in binary translation.
- To speed up binary translation is difficult. Therefore, many virtualization products employ the para-virtualization architecture.
- The popular Xen, KVM, and VMware ESX are good examples.

Para-Virtualization with Compiler Support

Unlike the full virtualization architecture which intercepts and emulates privileged and sensitive instructions at runtime, para-virtualization handles these instructions at compile time.

The guest OS kernel is modified to replace the privileged and sensitive instructions with hyper calls to the hypervisor or VMM.

Xen assumes such a para-virtualization architecture.

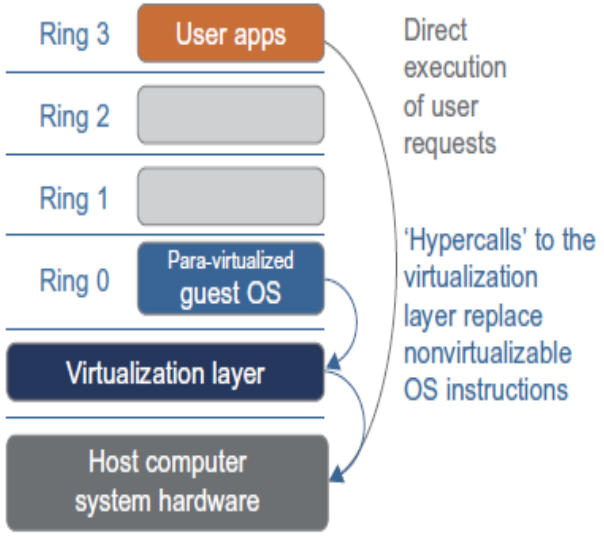


FIGURE 3.8

The use of a para-virtualized guest OS assisted by an intelligent compiler to replace nonvirtualizable OS instructions by hypercalls.

- This implies that the guest OS may not be able to execute some privileged and sensitive instructions.
- The privileged instructions are implemented by hypercalls to the hypervisor.
- After replacing the instructions with hypercalls, the modified guest OS emulates the behavior of the original guest OS

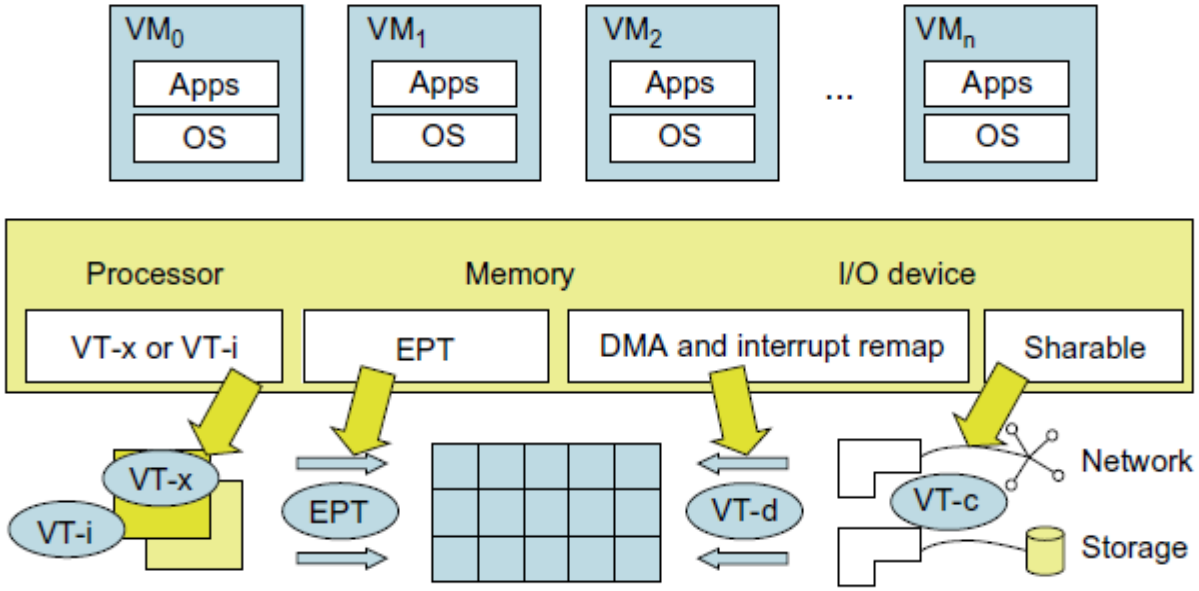
6.VIRTUALIZATION OF CPU, MEMORY, AND I/O DEVICES

- To support virtualization, processors such as the x86 employ a special running mode and instructions, known as hardware-assisted virtualization.
- In this way, the VMM and guest OS run in different modes and all sensitive instructions of the guest OS and its applications are trapped in the VMM.
- To save processor states, mode switching is completed by hardware. For the x86 architecture, Intel and AMD have proprietary technologies for hardware-assisted virtualization.

Hardware Support for Virtualization

- ✓ Modern operating systems and processors permit multiple processes to run simultaneously. If there is no protection mechanism in a processor, all instructions from different processes will access the hardware directly and cause a system crash.
- ✓ Therefore, all processors have at least two modes, user mode and supervisor mode, to ensure controlled access of critical hardware.
- ✓ Instructions running in supervisor mode are called privileged instructions.
- ✓ Other instructions are unprivileged instructions.

Hardware Support for Virtualization in the Intel x86 Processor



For processor virtualization, Intel offers the VT-x or VT-i technique. VT-x adds a privileged mode (VMX Root Mode) and some instructions to processors. This enhancement traps all sensitive instructions in the VMM automatically.

For memory virtualization, Intel offers the EPT, which translates the virtual address to the machine's physical addresses to improve performance.

For I/O virtualization, Intel implements VT-d and VT-c to support this.

Some of the hardware virtualization products are…..

- The Vmware Workstation is a VM software suite for x86 and x86-64 computers, allows users to set up multiple x86 and x86-64 virtual computers and to use one or more of these VMs simultaneously with the host operating system. The VMware Workstation assumes the host-based virtualization.

- Xen is a hypervisor for use in IA-32, x86-64, Itanium, and PowerPC 970 hosts. Actually, Xen modifies Linux as the lowest and most privileged layer, or a hypervisor. One or more guest OS can run on top of the hypervisor.

- KVM (Kernel-based Virtual Machine)is a Linux kernel virtualization infrastructure. KVM can support hardware-assisted virtualization and paravirtualization by using the Intel VT-x or AMD-v and VirtIO framework, respectively.

a)CPU Virtualization

A VM is a duplicate of an existing computer system in which a majority of the VM instructions are executed on the host processor in native mode. Thus, unprivileged instructions of VMs run directly on the host machine for higher efficiency. Other critical instructions should be handled carefully for correctness and stability.

The critical instructions are divided into three categories: privileged instructions, control sensitive instructions, and behavior-sensitive instructions.

- Privileged instructions execute in a privileged mode and will be trapped if executed outside this mode.
- Control-sensitive instructions attempt to change the configuration of resources used.
- Behavior-sensitive instructions have different behaviors depending on the configuration of resources, including the load and store operations over the virtual memory.

✓ A CPU architecture is **virtualizable** if it supports the ability to run the VM's privileged and unprivileged instructions in the CPU's user mode while the VMM runs in supervisor mode.

- When the privileged instructions including control- and behavior-sensitive instructions of a VM are executed, they are trapped in the VMM.

- In this case, the VMM acts as a unified mediator for hardware access from different VMs to guarantee the correctness and stability of the whole system.

- However, not all CPU architectures are virtualizable.

- RISC CPU architectures can be naturally virtualized because all control- and behavior-sensitive instructions are privileged instructions.

- On the contrary, x86 CPU architectures are not primarily designed to support virtualization. This is because about 10 sensitive instructions, such as SGDT and SMSW, are not privileged instructions.

- When these instructions execute in virtualization, they cannot be trapped in the VMM.

Hardware-Assisted CPU Virtualization

This technique attempts to simplify virtualization because full or para virtualization is complicated.

Intel and AMD add an additional mode called privilege mode level (some people call it Ring-1) to x86 processors.

- Therefore, operating systems can still run at Ring 0 and the hypervisor can run at Ring -1. All the privileged and sensitive instructions are trapped in the hypervisor automatically.

- This technique removes the difficulty of implementing binary translation of full virtualization. It also lets the operating system run in VMs without modification

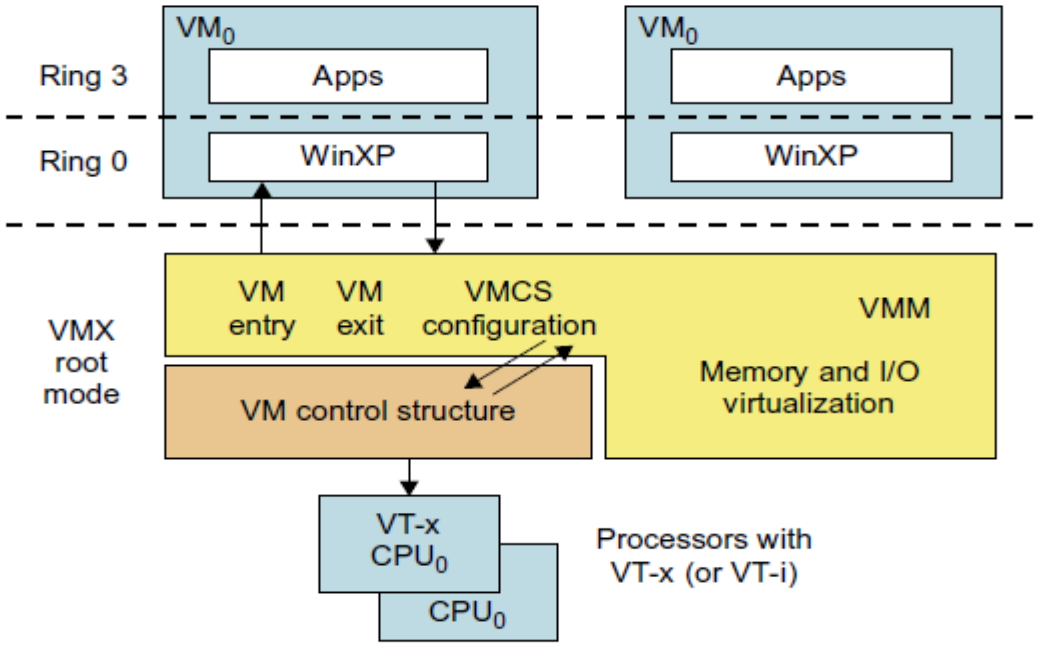


FIGURE 3.11

Intel hardware-assisted CPU virtualization.

b)Memory Virtualization

- Virtual memory virtualization is similar to the virtual memory support provided by modern operating systems.
- In a traditional execution environment, the operating system maintains mappings of virtual memory to machine memory using page tables, which is a one-stage mapping from virtual memory to machine memory.
- All modern x86 CPUs include a memory management unit (MMU) and a translation look aside buffer (TLB) to optimize virtual memory performance.
- However, in a virtual execution environment, virtual memory virtualization involves sharing the physical system memory in RAM and dynamically allocating it to the physical memory of the VMs.
- That means a two-stage mapping process should be maintained by the guest OS and the VMM, respectively:

  Virtual memory to physical memory and
  Physical memory to machine memory.

- Furthermore, MMU virtualization should be supported, which is transparent to the guest OS.
- The guest OS continues to control the mapping of virtual addresses to the physical memory addresses of VMs.
- But the guest OS cannot directly access the actual machine memory.
- The VMM is responsible for mapping the guest physical memory to the actual machine memory.

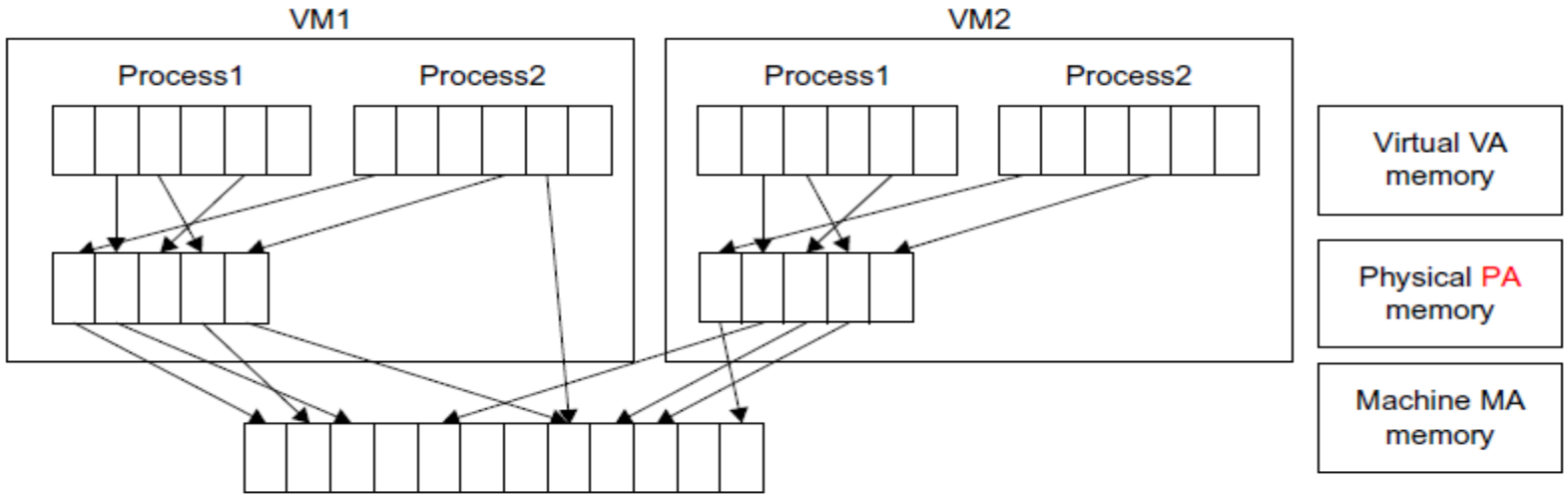


FIGURE 3.12

Two-level memory mapping procedure.

- Since each page table of the guest OSes has a separate page table in the VMM corresponding to it, the VMM page table is called the shadow page table.

- Nested page tables add another layer of indirection to virtual memory.

- The MMU already handles virtual-to-physical translations as defined by the OS.

- Then the physical memory addresses are translated to machine addresses using another set of page tables defined by the hypervisor. Since modern operating systems maintain a set of page tables for every process, the shadow page tables will get flooded.

- Consequently, the performance overhead and cost of memory will be very high.

- VMware uses shadow page tables to perform virtual-memory-to-machine-memory address translation.
- Processors use TLB hardware to map the virtual memory directly to the machine memory to avoid the two levels of translation on every access.

- When the guest OS changes the virtual memory to a physical memory mapping, the VMM updates the shadow page tables to enable a direct lookup.

I/O Virtualization

I/O virtualization involves managing the routing of I/O requests between virtual devices and the shared physical hardware.

At the time of this writing, there are three ways to implement I/O virtualization:

- full device emulation,
- para-virtualization,
- direct I/O.

Full device emulation is the first approach for I/O virtualization. Generally, this approach emulates well-known, real-world devices.

All the functions of a device or bus infrastructure, such as device enumeration, identification, interrupts, and DMA, are replicated in software. This software is located in the VMM and acts as a virtual device.

The I/O access requests of the guest OS are trapped in the VMM which interacts with the I/O devices.

A single hardware device can be shared by multiple VMs that run concurrently.

However, software emulation runs much slower than the hardware it emulates

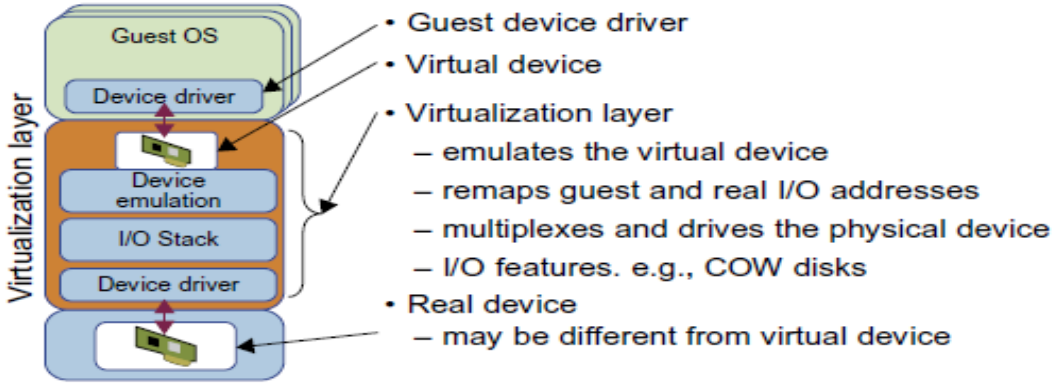


FIGURE 3.14

Device emulation for I/O virtualization implemented inside the middle layer that maps real I/O devices into the virtual devices for the guest device driver to use.

The para-virtualization method of I/O virtualization is typically used in Xen. It is also known as the split driver model consisting of a frontend driver and a backend driver.

The frontend driver is running in Domain U and the backend driver is running in Domain 0. They interact with each other via a block of shared memory.

The frontend driver manages the I/O requests of the guest OSes and the backend driver is responsible for managing the real I/O devices and multiplexing the I/O data of different VMs.

Although para-I/O-virtualization achieves better device performance than full device emulation, it comes with a higher CPU overhead.

Direct I/O virtualization lets the VM access devices directly. It can achieve close-to-native performance without high CPU costs.

However, current direct I/O virtualization implementations focus on networking for mainframes. There are a lot of challenges for commodity hardware devices.

For example, when a physical device is reclaimed (required by workload migration) for later reassignment, it may have been set to an arbitrary state (e.g., DMA to some arbitrary memory locations) that can function incorrectly or even crash the whole system.

Since software-based I/O virtualization requires a very high overhead of device emulation, hardware-assisted I/O virtualization is critical.

Intel VT-d supports the remapping of I/O DMA transfers and device-generated interrupts.

The architecture of VT-d provides the flexibility to support multiple usage models that may run unmodified, special-purpose, or “virtualization-aware” guest OSes.