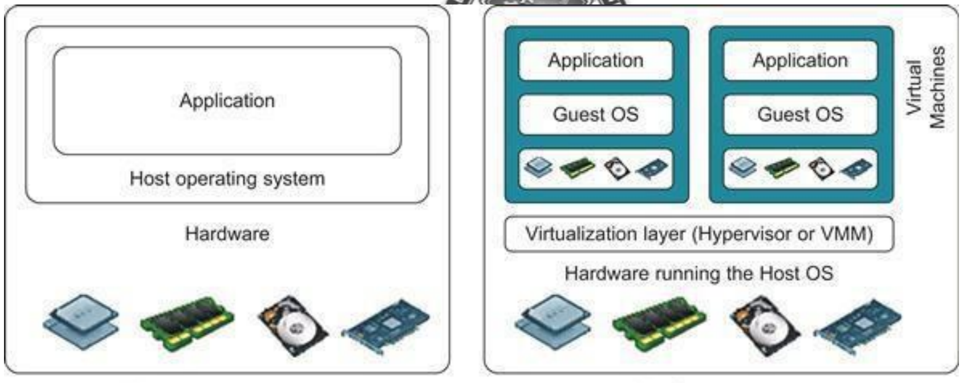
Virtualization Structures / Tools and Mechanisms

In general, there are three typical classes of VM architecture. Figure below shows the architectures of a machine before and after virtualization.

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, simultaneously.

Depending on the position of the virtualization layer, there are several classes of VM architectures, namely the hypervisor architecture, para-virtualization, and host-based virtualization. The hypervisor is also known as the VMM (Virtual Machine Monitor). They both perform the same virtualization operations.



(a) Traditional computer (b) After virtualization

Hypervisor and Xen Architecture

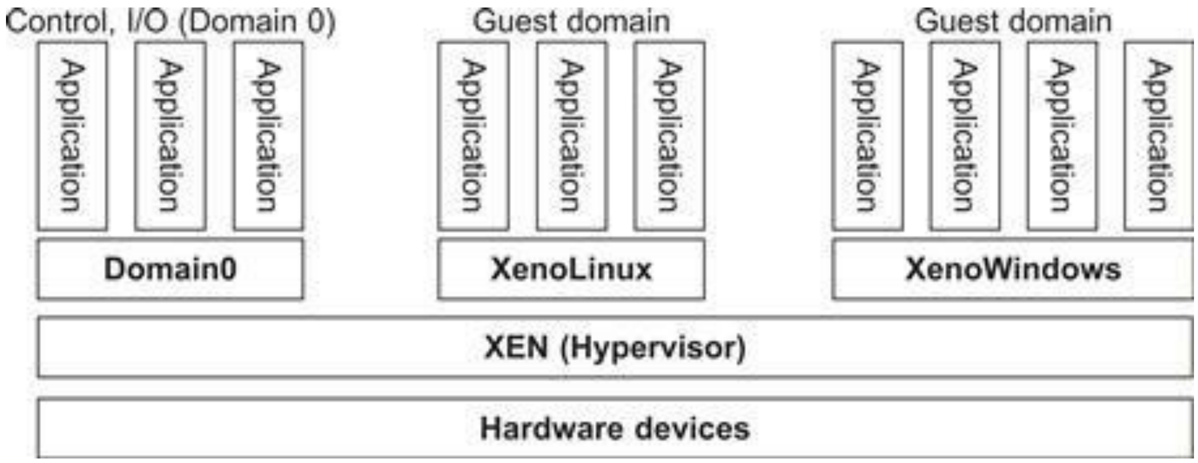
The hypervisor supports hardware-level virtualization (see Figure (b)) on bare metal devices like CPU, memory, disk and network interfaces. The hypervisor software's it's directly between the physical hardware and its OS. This virtualization layer is referred to as either the VMM or the hypervisor.

The hypervisor provides hyper calls for the guest OSes and applications. Depending on the functionality, a hypervisor can assume microkernel architecture like the Microsoft Hyper-V. Or it can assume 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 micro-kernel hypervisor, which separates the policy from the mechanism. The Xen hypervisor implements all the mechanisms, leaving the policy to be handled by Domain 0, as shown in Figure above.

Xen does not include any device drivers natively. It just provides a mechanism by which guests 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 environmentlocatedbetweenthehardwareandtheOS.Anumberofvendorsareintheprocess of developing commercial Xen hypervisors, among them are Citrix Xen Server and Oracle VM. 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 Domain0, and the others are called DomainU. Domain0 is a privileged guest OS of Xen. It is first loaded when Xen boots without any file system drivers being available. Domain0 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).

Binary Translation with Full Virtualization

Depending on implementation technologies, hardware virtualization can be classified into two categories: full virtualization and 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 OSes 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**

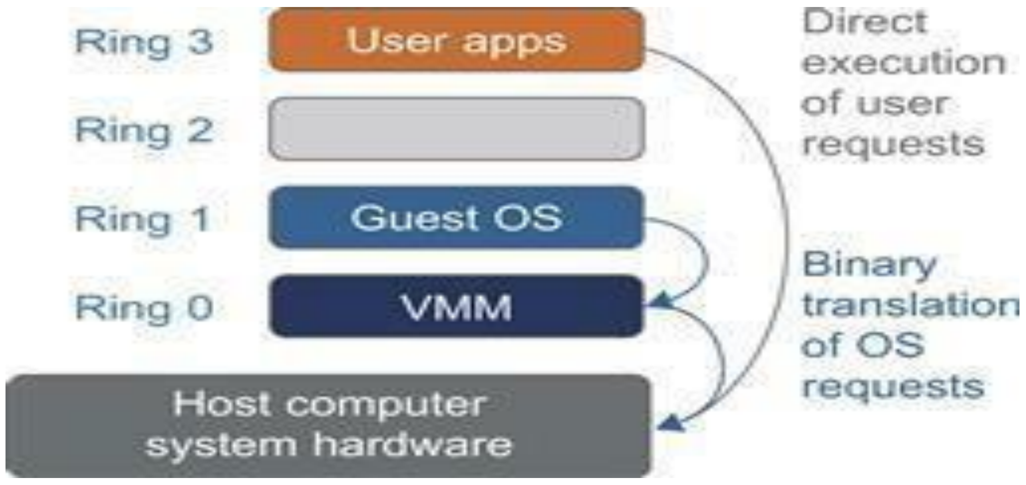
With full virtualization, noncritical instructions run on the hardware directly while critical instructions are discovered and replaced with traps into the VMM to be emulated by software.
Both the hypervisor and VMM approaches are considered full virtualization.

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. As shown in Figure below, 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 unaware that it is being virtualized.


Ring 3
User apps
Ring 2
Ring 1
Guest OS
Ring 0
VMM
Host computer system hardware
Direct execution of user requests
Binary translation of OS requests

In direct 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**

An alternative 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.
Certainly, some other applications can also run with the host OS directly. This host-based architecture has some distinct **advantages**, as,

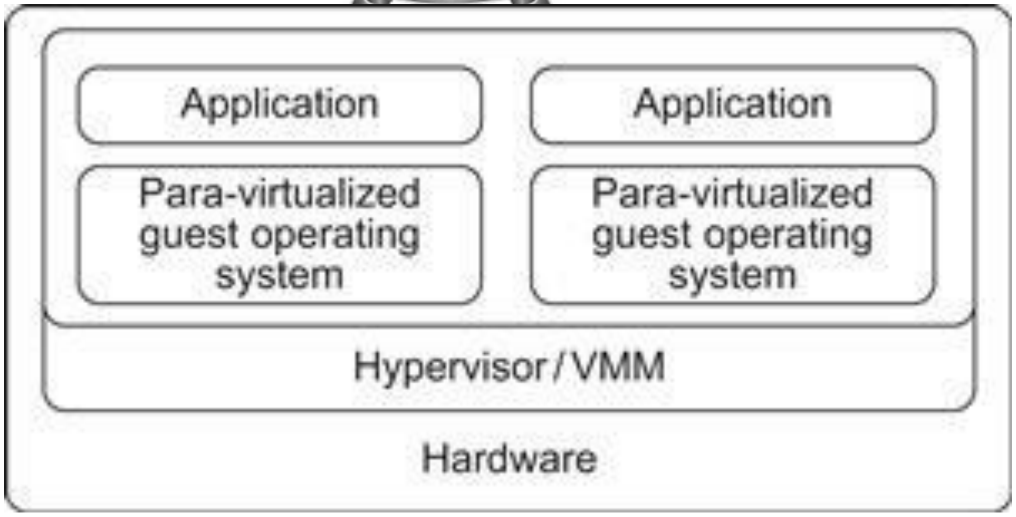
First, the user can install this VM architecture without modifying the host OS. The virtualizing software can rely on the host OS to provide device drivers and other low-level services. This will simplify the VM design and ease its deployment.

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.

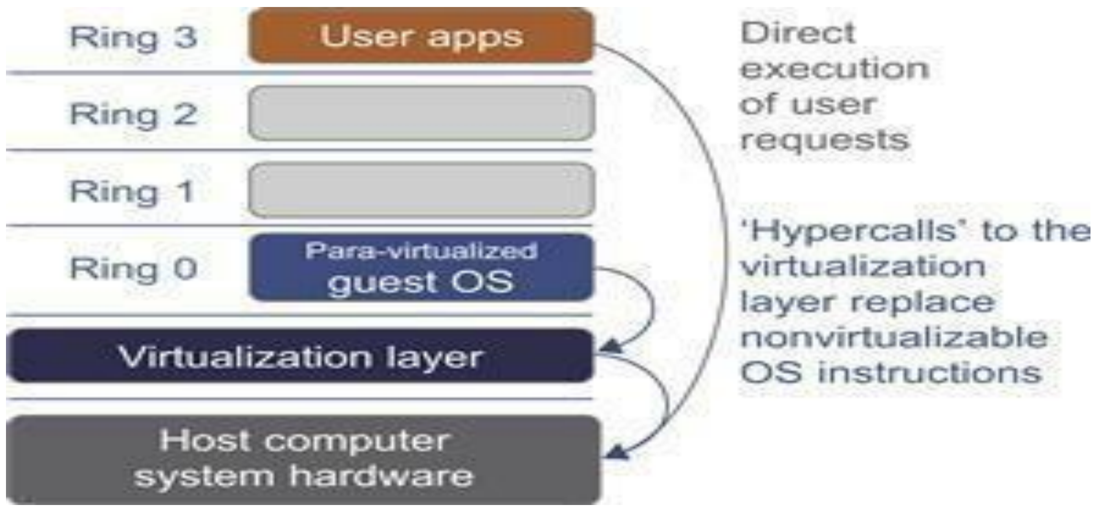
Para-Virtualization with Compiler Support

Para-virtualization needs to modify the guest operating systems. A para-virtualized VM providesspecialAPIsrequiringsubstantialOSmodificationsinuserapplications.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 attempt store duce the virtualization overhead, and thus improves performance by modifying only the guest OS kernel. The guest operating systems are para virtualized.

The traditional x86 processor offers four instruction execution rings: Rings 0,1, 2, and 3. The lower the ring number, the higher the privilege of instruction being executed. The OS is responsible for managing the hardware and the privileged instructions to execute at Ring 0, while user-level applications run at Ring 3.


Application
Application
Para-virtualized guest operating system
Para-virtualized guest operating system
Hypervisor / VMM
Hardware

Para-virtualized VM architecture


Ring 3
User apps
Ring 2
Ring 1
Ring 0
Para-virtualized guest OS
Virtualization layer
Host computer system hardware
Direct execution of user requests
'Hypercalls' to the virtualization layer replace nonvirtualizable OS instructions

The use of a para-virtualized guest OS assisted by an intelligent compiler to replace non virtualizable OS instructions by hyper calls.

Para-Virtualization Architecture:

When the x86 processor is virtualized, a virtualization layer is inserted between the hardware and the OS. According to the x86 ring definitions, the virtualization layer should also be installed at Ring 0. The para-virtualization replaces non virtualizable instructions with hyper calls that communicate directly with the hypervisor or VMM. However, when the guest OS kernel is modified for virtualization, it can no longer run on the hardware directly.
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.

KVM(Kernel-Based VM):

This is a Linux para-virtualization system—a part of the Linux version 2.6.20 kernel. Memory management and scheduling activities are carried out by the existing Linux kernel. The KVM does the rest, which makes it simpler than the hypervisor that controls the entire machine.

KVM is a hardware-assisted para-virtualization tool, which improves performance and supports unmodified guest OSes such as Windows, Linux, Solaris, and other UNIX variants. Unlike the full virtualization architecture which intercepts and emulates privileged and sensitive instructions at run time, 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. The guest OS running in a guest domain may run at Ring1 instead of at Ring0. This implies that the guest OS may not be able to execute some privileged and sensitive instructions. The privileged instructions are implemented by hyper calls to the hypervisor. After replacing the instructions with hyper calls, the modified guest OS emulates the behavior of the original guest OS.