VIRTUALIZATION

A virtualization layer is added between the hardware and operating system. This virtualization layer allows multiple operating system instances to run concurrently within virtual machines on a single computer, dynamically partitioning and sharing the available physical resources such as CPU, storage, memory and I/O devices.

Virtualization approaches use either a hosted (type 2) or a hypervisor (type 1) architecture.

A hosted architecture installs and runs the virtualization layer as an application on top of an operating system and supports the broadest range of hardware configurations.

In contrast, a hypervisor (bare-metal) architecture installs the virtualization layer directly on top of bare metal machine. Since it has direct access to the hardware resources rather than going through an operating system, a hypervisor is more efficient than a hosted architecture and delivers greater scalability, robustness and performance.

The functionality of the hypervisor varies greatly based on architecture and implementation. Each **Virtual Machine Manager (VMM)** running on the hypervisor implements the virtual machine hardware abstraction and is responsible for running a guest OS. Each VMM has to partition and share the CPU, memory and I/O devices to successfully virtualize the system.

1. CPU Virtualization

The Challenges of x86 Hardware Virtualization

Operating systems are designed to run directly on the bare-metal hardware, so they naturally assume they fully ‘own’ the computer hardware.

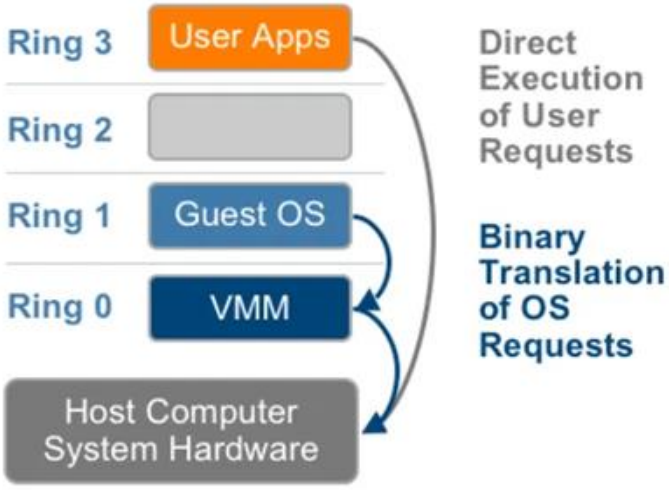
The architecture offers four levels of privilege known as Protection Ring 0, 1, 2 and 3 to operating systems and applications to manage access to the computer hardware. While user level applications typically run in Ring 3, the operating system needs to have direct access to the memory and hardware and must execute its privileged instructions in Ring 0.

Virtualizing the architecture requires placing a virtualization layer under the operating system (which expects to be in the most privileged Ring 0) to create and manage the virtual machines that deliver shared resources.
Further complicating the situation, some sensitive instructions can't effectively be virtualized as they have different semantics when they are not executed in Ring 0. The difficulty in trapping and translating these sensitive and privileged instruction requests at runtime was the challenge.

The challenge got resolved by developing binary translation techniques that allow the VMM to run in Ring 0 for isolation and performance, while moving the operating system to a user level ring with greater privilege than applications in Ring 3 but less privilege than the virtual machine monitor in Ring 0Three alternative techniques now exist for handling sensitive and privileged instructions to virtualize the CPU on the architecture:

1. Full virtualization using binary translation
2. OS assisted virtualization or paravirtualization
3. Hardware assisted virtualization (first generation)

Technique 1 — Full Virtualization using Binary Translation

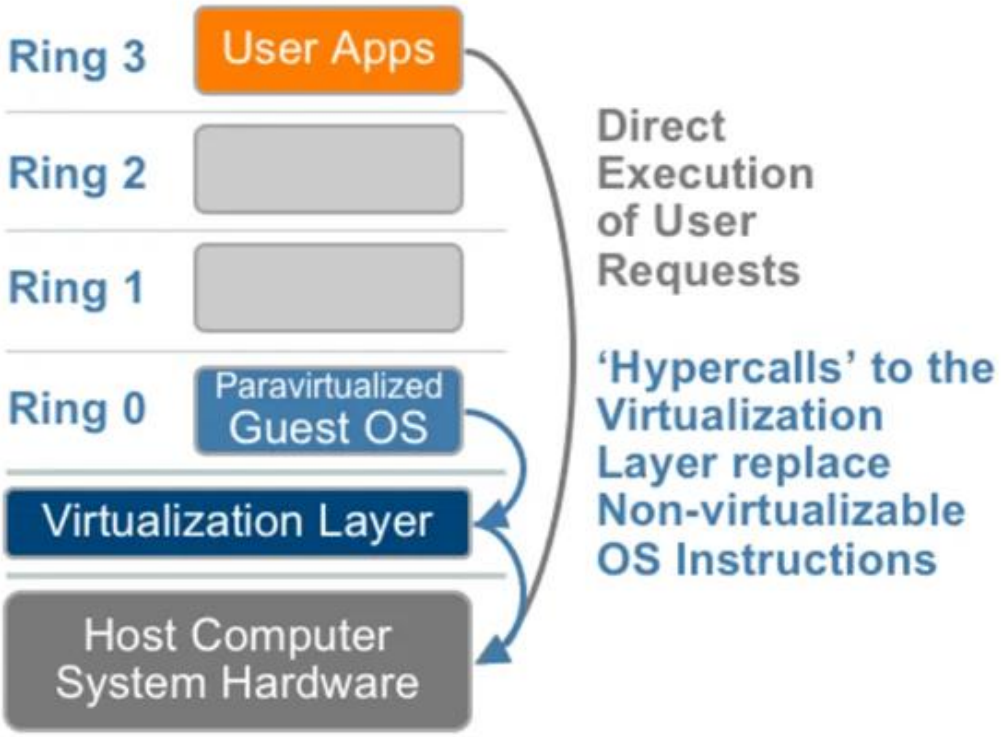


We can virtualize any operating system using a combination of binary translation and direct execution techniques. This approach translates kernel code to replace non virtualizable instructions with new sequences of instructions that have the intended effect on the virtual hardware. Meanwhile, user level code is directly executed on the processor for high performance virtualization.

Each virtual machine monitor (VMM) provides each Virtual Machine with all the services of the physical system, including a virtual BIOS, virtual devices and virtualized memory management. This combination of binary translation and direct execution provides Full Virtualization as the guest OS is fully abstracted (completely decoupled) from the underlying hardware by the virtualization layer.

The guest OS is not aware it is being virtualized and requires no modification. Full virtualization is the only option that requires no hardware assist or operating system assist to virtualize sensitive and privileged instructions. The hypervisor translates all operating system instructions on the fly and caches the results for future use, while user level instructions run unmodified at native speed.Full virtualization offers the best isolation and security for virtual machines, and simplifies migration and portability as the same guest OS instance can run virtualized or on native hardware.

Technique 2 — OS Assisted Virtualization or Paravirtualization



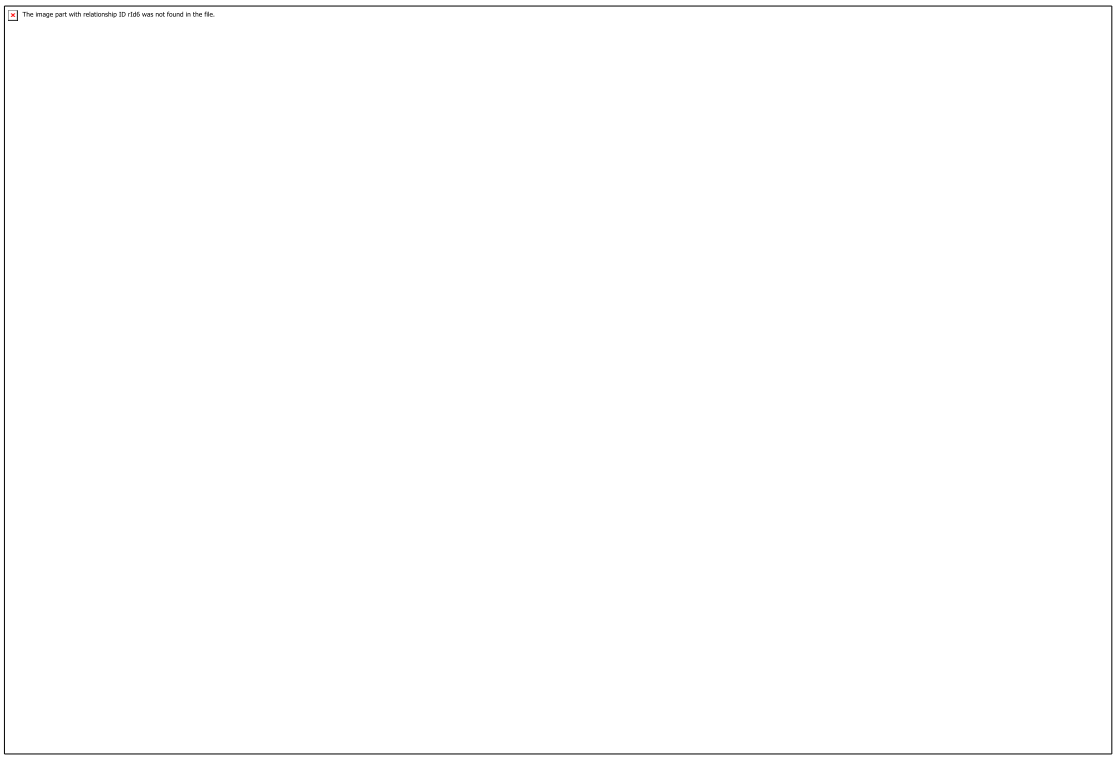
Paravirtualization refers to communication between the guest OS and the hypervisor to improve performance and efficiency. Paravirtualization involves modifying the OS kernel to replace nonvirtualizable instructions with hypercalls that communicate directly with the virtualization layer hypervisor.

The hypervisor also provides hypercall interfaces for other critical kernel operations such as memory management, interrupt handling and time keeping. Paravirtualization is different from full virtualization, where the unmodified OS does not know it is virtualized and sensitive OS calls are trapped using binary translation.

The performance advantage of paravirtualization over full virtualization can vary greatly depending on the workload. As paravirtualization cannot support unmodified operating systems, its compatibility and portability is poor. Paravirtualization can also introduce significant support and maintainability issues in production environments as it requires deep OS kernel modifications.

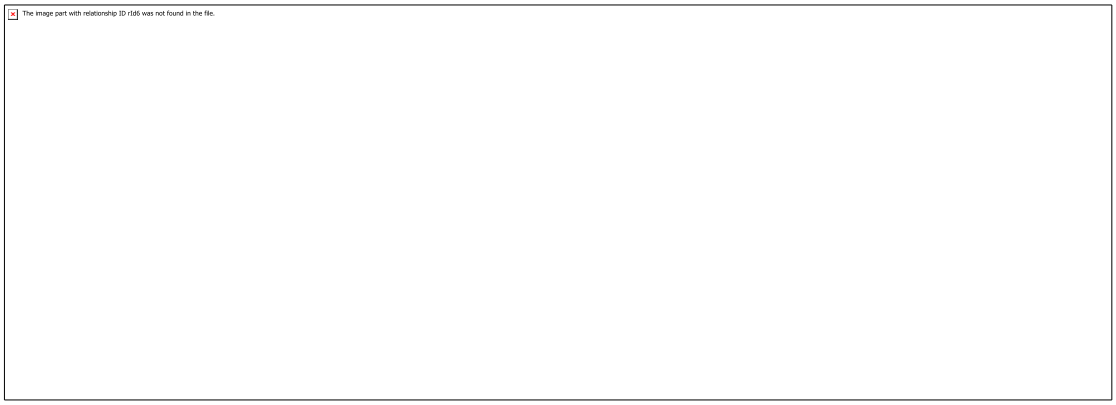
The open source Xen project is an example of paravirtualization that virtualizes the processor and memory using a modified Linux kernel and virtualizes the I/O using custom guest OS device drivers. While it is very difficult to build the more sophisticated binary translation support necessary for full virtualization, modifying the guest OS to enable paravirtualization is relatively easy. There are minimal, non-intrusive changes installed into the guest OS that do not require OS kernel modification.

Technique 3 — Hardware Assisted Virtualization



Hardware vendors are rapidly embracing virtualization and developing new features to simplify virtualization techniques. First generation enhancements include Intel Virtualization Technology (VT-x) and AMD's AMD-V which both target privileged instructions with a new CPU execution mode feature that allows the VMM to run in a new root mode below ring 0. Privileged and sensitive calls are set to automatically trap to the hypervisor, removing the need for either binary translation or paravirtualization. The guest state is stored in Virtual Machine Control Structures (VT-x) or Virtual Machine Control Blocks (AMD-V).

2. Memory Virtualization



Beyond CPU virtualization, the next critical component is memory virtualization. This involves sharing the physical system memory and dynamically allocating it to virtual machines. Virtual machine memory virtualization is very similar to the virtual memory support provided by modern operating systems. Applications see a contiguous address space that is not necessarily tied to the underlying physical memory in the system. The operating system keeps mappings of virtual page
numbers to physical page numbers stored in page tables. All modern CPUs include a memory management unit (MMU) and a translation lookaside buffer (TLB) to optimize virtual memory performance.

To run multiple virtual machines on a single system, another level of memory virtualization is required. In other words, one has to virtualize the MMU to support the guest OS. The guest OS continues to control the mapping of virtual addresses to the guest memory physical addresses, but the guest OS cannot have direct access to the actual machine memory. The VMM is responsible for mapping guest physical memory to the actual machine memory, and it uses shadow page tables to accelerate the mappings.

The VMM uses 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 physical memory mapping, the VMM updates the shadow page tables to enable a direct lookup. MMU virtualization creates some overhead for all virtualization approaches, but this is the area where second generation hardware assisted virtualization will offer efficiency gains

3. Device and I/O Virtualization

The final component required beyond CPU and memory virtualization is device and I/O virtualization. This involves managing routing I/O requests between virtual devices and the shared physical hardware.

Software based I/O virtualization and management, in contrast to a direct pass-through to the hardware, enables a rich set of features and simplified management. With networking for example, virtual NICs and switches create virtual networks between virtual machines without the network traffic consuming bandwidth on the physical network, NIC teaming allows multiple physical NICS to appear as one and failover transparently for virtual machines, and virtual machines can be seamlessly relocated to different systems while keeping their existing MAC addresses. The key to effective I/O virtualization is to preserve these virtualization benefits while keeping the added CPU utilization to a minimum