**Unit - IV: Virtualization**

**Virtualization** is the "creation of a virtual (rather than actual) version of something, such as a server, a desktop, a storage device, an operating system or network resources".

In other words, Virtualization is a technique, which allows to share a single physical instance of a resource or an application among multiple customers and organizations. It does by assigning a logical name to a physical storage and providing a pointer to that physical resource when demanded.

## **What is the concept behind the Virtualization?**

Creation of a virtual machine over existing operating system and hardware is known as Hardware Virtualization. A Virtual machine provides an environment that is logically separated from the underlying hardware.

The machine on which the virtual machine is going to create is known as **Host Machine** and that virtual machine is referred as a **Guest Machine**

**BENEFITS OF VIRTUALIZATION**

1. More flexible and efficient allocation of resources.

2. Enhance development productivity.

3. It lowers the cost of IT infrastructure.

4. Remote access and rapid scalability.

5. High availability and disaster recovery.

6. Pay peruse of the IT infrastructure on demand.

7. Enables running multiple operating systems.

**Implementation Levels of virtualization**

The Five Levels of Implementing Virtualization

1. Instruction Set Architecture Level (ISA)
2. Hardware Abstraction Level (HAL)
3. Operating System Level
4. Library Level
5. Application Level

There are five levels of virtualizations available that are most commonly used in the industry. These are as follows:

### Instruction Set Architecture Level (ISA)

In ISA, virtualization works through an ISA emulation. This is helpful to run heaps of legacy code which was originally written for different hardware configurations.

These codes can be run on the virtual machine through an ISA.

A binary code that might need additional layers to run can now run on an x86 machine or with some tweaking, even on x64 machines. ISA helps make this a hardware-agnostic virtual machine.

The basic emulation, though, requires an interpreter. This interpreter interprets the source code and converts it to a hardware readable format for processing.

### Hardware Abstraction Level (HAL)

As the name suggests, this level helps perform virtualization at the hardware level. It uses a bare hypervisor for its functioning.

This level helps form the virtual machine and manages the hardware through virtualization.

It enables virtualization of each hardware component such as I/O devices, processors, memory, etc.

This way multiple users can use the same hardware with numerous instances of virtualization at the same time.

IBM had first implemented this on the IBM VM/370 back in 1960. It is more usable for cloud-based infrastructure.

Thus, it is no surprise that currently, Xen hypervisors are using HAL to run Linux and other OS on x86 based machines.

### Operating System Level

At the operating system level, the virtualization model creates an abstract layer between the applications and the OS.

It is like an isolated container on the physical server and operating system that utilizes hardware and software. Each of these containers functions like servers.

When the number of users is high, and no one is willing to share hardware, this level of virtualization comes in handy.

Here, every user gets their own virtual environment with dedicated virtual hardware resources. This way, no conflicts arise.

### Library Level

OS system calls are lengthy and cumbersome. Which is why applications opt for APIs from user-level libraries.

Most of the APIs provided by systems are rather well documented. Hence, library level virtualization is preferred in such scenarios.

Library interfacing virtualization is made possible by API hooks. These API hooks control the communication link from the system to the applications.

Some tools available today, such as vCUDA and WINE, have successfully demonstrated this technique.

### Application Level

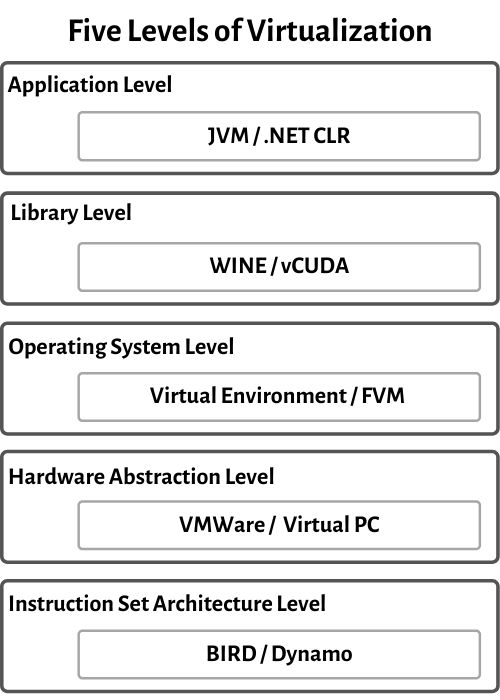
Application-level virtualization comes handy when you wish to virtualize only an application. It does not virtualize an entire platform or environment.

On an operating system, applications work as one process. Hence it is also known as process-level virtualization.

It is generally useful when running virtual machines with high-level languages. Here, the application sits on top of the virtualization layer, which is above the application program.

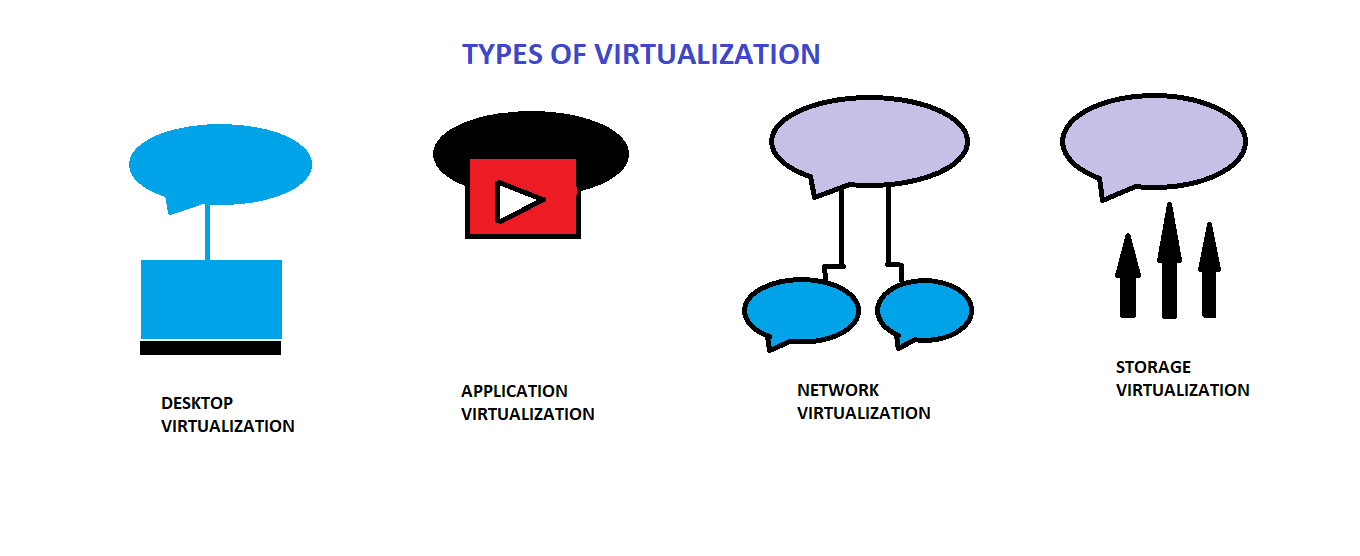
The application program is, in turn, residing in the operating system.

Programs written in high-level languages and compiled for an application-level virtual machine can run fluently here.



**Types of Virtualization:**

1. .Application Virtualization.
2. .Network Virtualization.
3. .Desktop Virtualization.
4. .Storage Virtualization.
5. .Server Virtualization.
6. .Data virtualization.



**1.** **Application Virtualization:**   
Application virtualization helps a user to have remote access of an application from a server. The server stores all personal information and other characteristics of the application but can still run on a local workstation through the internet. Example of this would be a user who needs to run two different versions of the same software. Technologies that use application virtualization are hosted applications and packaged applications.

**2.** **Network Virtualization:**   
The ability to run multiple virtual networks with each has a separate control and data plan. It co-exists together on top of one physical network. It can be managed by individual parties that potentially confidential to each other.   
Network virtualization provides a facility to create and provision virtual networks—logical switches, routers, firewalls, load balancer, Virtual Private Network (VPN), and workload security within days or even in weeks.

**3.** **Desktop Virtualization:**   
Desktop virtualization allows the users’ OS to be remotely stored on a server in the data centre. It allows the user to access their desktop virtually, from any location by a different machine. Users who want specific operating systems other than Windows Server will need to have a virtual desktop. Main benefits of desktop virtualization are user mobility, portability, easy management of software installation, updates, and patches.

**4.** **Storage Virtualization:**   
Storage virtualization is an array of servers that are managed by a virtual storage system. The servers aren’t aware of exactly where their data is stored, and instead function more like worker bees in a hive. It makes managing storage from multiple sources to be managed and utilized as a single repository. storage virtualization software maintains smooth operations, consistent performance and a continuous suite of advanced functions despite changes, break down and differences in the underlying equipment.

**5.** **Server Virtualization:**  
This is a kind of virtualization in which masking of server resources takes place. Here, the central-server(physical server) is divided into multiple different virtual servers by changing the identity number, processors. So, each system can operate its own operating systems in isolate manner. Where each sub-server knows the identity of the central server. It causes an increase in the performance and reduces the operating cost by the deployment of main server resources into a sub-server resource. It’s beneficial in virtual migration, reduce energy consumption, reduce infrastructural cost, etc.

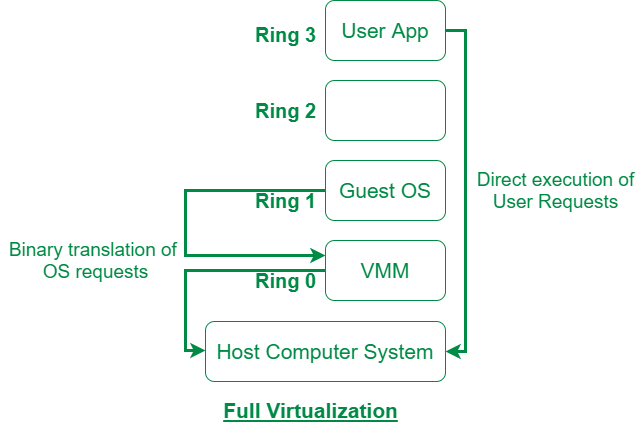
**6.** **Data virtualization:**  
This is the kind of virtualization in which the data is collected from various sources and managed that at a single place without knowing more about the technical information like how data is collected, stored & formatted then arranged that data logically so that its virtual view can be accessed by its interested people and stakeholders, and users through the various cloud services remotely. Many big giant companies are providing their services like Oracle, IBM, At scale, Cdata, etc.

 It can be used to performing various kind of tasks such as:

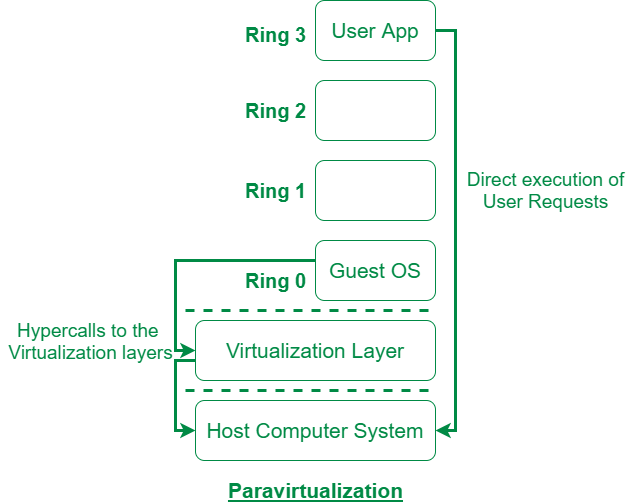
* **Data-integration**
* **Business-integration**
* **Service-oriented architecture data-services**
* **Searching organizational data**

**Full Virtualization and Paravirtualization**

1. **Full Virtualization:**
   1. Full Virtualization was introduced by IBM in the year 1966.
   2. It is the first software solution for server virtualization and uses binary translation and direct approach techniques.
   3. In full virtualization, guest OS is completely isolated by the virtual machine from the virtualization layer and hardware.
   4. Microsoft and Parallels systems are examples of full virtualization



1. **Paravirtualization:**
   1. Paravirtualization is the category of CPU virtualization which uses hypercalls for operations to handle instructions at compile time.
   2. In paravirtualization, guest OS is not completely isolated but it is partially isolated by the virtual machine from the virtualization layer and hardware.
   3. VMware and Xen are some examples of paravirtualization.



The difference between Full Virtualization and Paravirtualization are as follows:

| **S. No.** | **Full Virtualization** | **Paravirtualization** |
| --- | --- | --- |
| 1. | In Full virtualization, virtual machines permit the execution of the instructions with the running of unmodified OS in an entirely isolated way. | In paravirtualization, a virtual machine does not implement full isolation of OS but rather provides a different API which is utilized when OS is subjected to alteration. |
| 2. | Full Virtualization is less secure. | While the Paravirtualization is more secure than the Full Virtualization. |
| 3. | Full Virtualization uses binary translation and a direct approach as a technique for operations. | While Paravirtualization uses hypercalls at compile time for operations. |
| 4. | Full Virtualization is slow than paravirtualization in operation. | Paravirtualization is faster in operation as compared to full virtualization. |
| 5. | Full Virtualization is more portable and compatible. | Paravirtualization is less portable and compatible. |
| 6. | Examples of full virtualization are Microsoft and Parallels systems. | Examples of paravirtualization are Microsoft Hyper-V, Citrix Xen, etc. |
| 7. | It supports all guest operating systems without modification. | The guest operating system has to be modified and only a few operating systems support it. |
| 8. | The guest operating system will issue hardware calls. | Using the drivers, the guest operating system will directly communicate with the hypervisor. |
| 9. | It is less streamlined compared to para-virtualization. | It is more streamlined. |
| 10. | It provides the best isolation. | It provides less isolation compared to full virtualization. |

**Virtualization of CPU, memory and IO 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.

**1. Hardware Support for Virtualization**

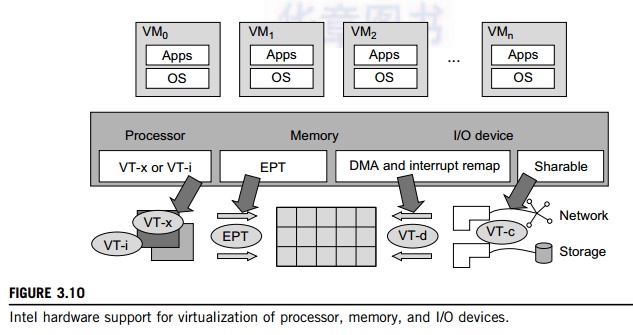
Modern operating systems and processors permit multiple processes to run simultaneously. If there is no protection mechanism in a processor, all instructions from different processes will access the hardware directly and cause a system crash. Therefore, all processors have at least two modes, user mode and supervisor mode, to ensure controlled access of critical hardware. Instructions running in supervisor mode are called privileged instructions. Other instructions are unprivileged instructions. In a virtualized environment, it is more difficult to make OSes and applications run correctly because there are more layers in the machine stack. Example 3.4 discusses Intel’s hardware support approach.

At the time of this writing, many hardware virtualization products were available. The VMware Workstation is a VM software suite for x86 and x86-64 computers. This software suite 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. The VirtIO framework includes a paravirtual Ethernet card, a disk I/O controller, a balloon device for adjusting guest memory usage, and a VGA graphics interface using VMware drivers.

Example 3.4 Hardware Support for Virtualization in the Intel x86 Processor

Since software-based virtualization techniques are complicated and incur performance overhead, Intel provides a hardware-assist technique to make virtualization easy and improve performance. Figure 3.10 provides an overview of Intel’s full virtualization techniques. 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.



**2. 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 exe-cuted, 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 instruc-tions execute in virtualization, they cannot be trapped in the VMM.

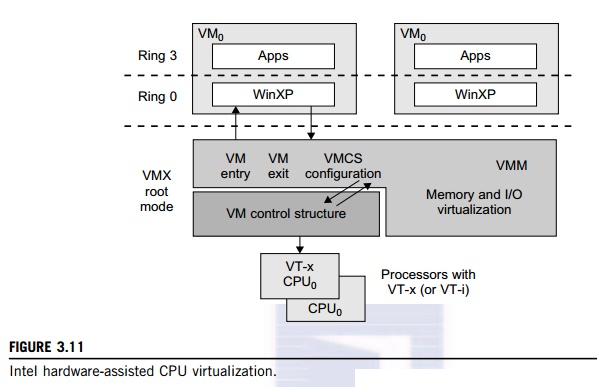
On a native UNIX-like system, a system call triggers the 80h interrupt and passes control to the OS kernel. The interrupt handler in the kernel is then invoked to process the system call. On a para-virtualization system such as Xen, a system call in the guest OS first triggers the 80h interrupt nor-mally. Almost at the same time, the 82h interrupt in the hypervisor is triggered. Incidentally, control is passed on to the hypervisor as well. When the hypervisor completes its task for the guest OS system call, it passes control back to the guest OS kernel. Certainly, the guest OS kernel may also invoke the hypercall while it’s running. Although paravirtualization of a CPU lets unmodified applications run in the VM, it causes a small performance penalty.

**2.1 Hardware-Assisted CPU Virtualization**

This technique attempts to simplify virtualization because full or paravirtualization 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.

Example 3.5 Intel Hardware-Assisted CPU Virtualization

Although x86 processors are not virtualizable primarily, great effort is taken to virtualize them. They are used widely in comparing RISC processors that the bulk of x86-based legacy systems cannot discard easily. Virtuali-zation of x86 processors is detailed in the following sections. Intel’s VT-x technology is an example of hardware-assisted virtualization, as shown in Figure 3.11. Intel calls the privilege level of x86 processors the VMX Root Mode. In order to control the start and stop of a VM and allocate a memory page to maintain the



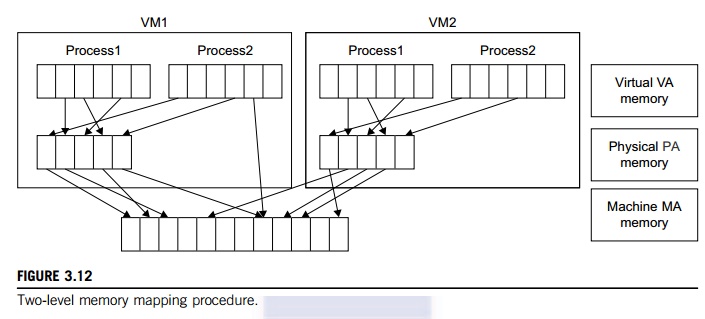
CPU state for VMs, a set of additional instructions is added. At the time of this writing, Xen, VMware, and the Microsoft Virtual PC all implement their hypervisors by using the VT-x technology.

Generally, hardware-assisted virtualization should have high efficiency. However, since the transition from the hypervisor to the guest OS incurs high overhead switches between processor modes, it sometimes cannot outperform binary translation. Hence, virtualization systems such as VMware now use a hybrid approach, in which a few tasks are offloaded to the hardware but the rest is still done in software. In addition, para-virtualization and hardware-assisted virtualization can be combined to improve the performance further.

**3. Memory Virtualization**

Virtual memory virtualization is similar to the virtual memory support provided by modern operat-ing 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 lookaside 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 shows the 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 perfor-mance overhead and cost of memory will be very high.

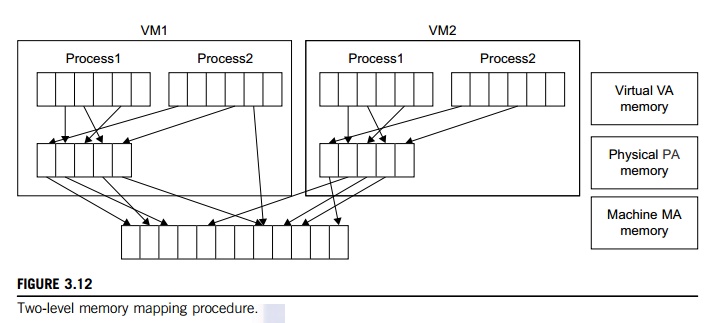
VMware uses shadow page tables to perform virtual-memory-to-machine-memory address translation. Processors use TLB hardware to map the virtual memory directly to the machine memory to avoid the two levels of translation on every access. When the guest OS changes the virtual memory to a physical memory mapping, the VMM updates the shadow page tables to enable a direct lookup. The AMD Barcelona processor has featured hardware-assisted memory virtualization since 2007. It provides hardware assistance to the two-stage address translation in a virtual execution environment by using a technology called nested paging.

Example 3.6 Extended Page Table by Intel for Memory Virtualization

Since the efficiency of the software shadow page table technique was too low, Intel developed a hardware-based EPT technique to improve it, as illustrated in Figure 3.13. In addition, Intel offers a Virtual Processor ID (VPID) to improve use of the TLB. Therefore, the performance of memory virtualization is greatly improved. In Figure 3.13, the page tables of the guest OS and EPT are all four-level.

When a virtual address needs to be translated, the CPU will first look for the L4 page table pointed to by Guest CR3. Since the address in Guest CR3 is a physical address in the guest OS, the CPU needs to convert the Guest CR3 GPA to the host physical address (HPA) using EPT. In this procedure, the CPU will check the EPT TLB to see if the translation is there. If there is no required translation in the EPT TLB, the CPU will look for it in the EPT. If the CPU cannot find the translation in the EPT, an EPT violation exception will be raised.

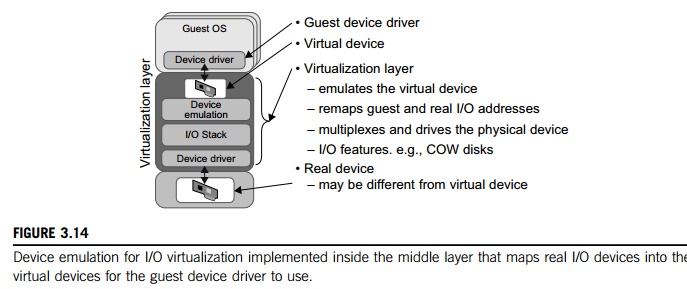
When the GPA of the L4 page table is obtained, the CPU will calculate the GPA of the L3 page table by using the GVA and the content of the L4 page table. If the entry corresponding to the GVA in the L4



page table is a page fault, the CPU will generate a page fault interrupt and will let the guest OS kernel handle the interrupt. When the PGA of the L3 page table is obtained, the CPU will look for the EPT to get the HPA of the L3 page table, as described earlier. To get the HPA corresponding to a GVA, the CPU needs to look for the EPT five times, and each time, the memory needs to be accessed four times. There-fore, there are 20 memory accesses in the worst case, which is still very slow. To overcome this short-coming, Intel increased the size of the EPT TLB to decrease the number of memory accesses.

**4. 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, and 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. The full device emulation approach is shown in Figure 3.14.

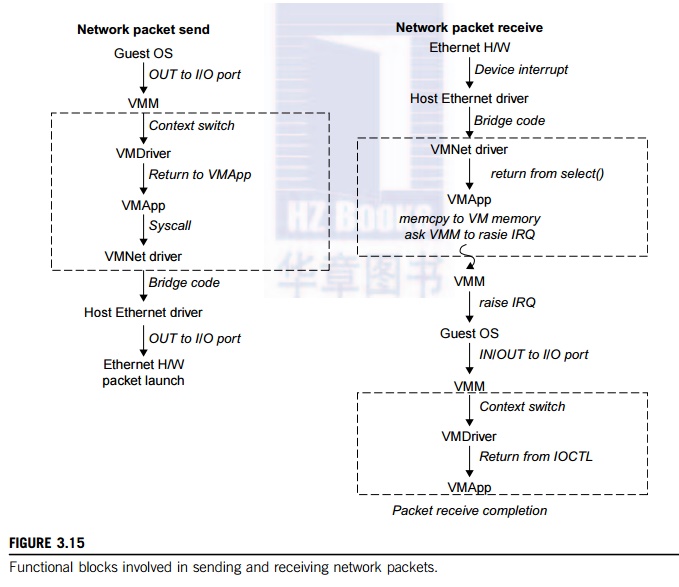
A single hardware device can be shared by multiple VMs that run concurrently. However, software emulation runs much slower than the hardware it emulates [10,15]. The para-virtualization method of I/O virtualization is typically used in Xen. It is also known as the split driver model consisting of a frontend driver and a backend driver. The frontend driver is running in Domain U and the backend dri-ver 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 reassign-ment, 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.

Another way to help I/O virtualization is via self-virtualized I/O (SV-IO) [47]. The key idea of SV-IO is to harness the rich resources of a multicore processor. All tasks associated with virtualizing an I/O device are encapsulated in SV-IO. It provides virtual devices and an associated access API to VMs and a management API to the VMM. SV-IO defines one virtual interface (VIF) for every kind of virtua-lized I/O device, such as virtual network interfaces, virtual block devices (disk), virtual camera devices, and others. The guest OS interacts with the VIFs via VIF device drivers. Each VIF consists of two mes-sage queues. One is for outgoing messages to the devices and the other is for incoming messages from the devices. In addition, each VIF has a unique ID for identifying it in SV-IO.

**Example : VMware Workstation for I/O Virtualization**

The VMware Workstation runs as an application. It leverages the I/O device support in guest OSes, host OSes, and VMM to implement I/O virtualization. The application portion (VMApp) uses a driver loaded into the host operating system (VMDriver) to establish the privileged VMM, which runs directly on the hardware. A given physical processor is executed in either the host world or the VMM world, with the VMDriver facilitating the transfer of control between the two worlds. The VMware Workstation employs full device emulation to implement I/O virtualization. Figure 3.15 shows the functional blocks used in sending and receiving packets via the emulated virtual NIC.



The virtual NIC models an AMD Lance Am79C970A controller. The device driver for a Lance controller in the guest OS initiates packet transmissions by reading and writing a sequence of virtual I/O ports; each read or write switches back to the VMApp to emulate the Lance port accesses. When the last OUT instruc-tion of the sequence is encountered, the Lance emulator calls a normal write() to the VMNet driver. The VMNet driver then passes the packet onto the network via a host NIC and then the VMApp switches back to the VMM. The switch raises a virtual interrupt to notify the guest device driver that the packet was sent. Packet receives occur in reverse.

**5. Virtualization in Multi-Core Processors**

Virtualizing a multi-core processor is relatively more complicated than virtualizing a uni-core processor. Though multicore processors are claimed to have higher performance by integrating multiple processor cores in a single chip, muti-core virtualiuzation has raised some new challenges to computer architects, compiler constructors, system designers, and application programmers. There are mainly two difficulties: Application programs must be parallelized to use all cores fully, and software must explicitly assign tasks to the cores, which is a very complex problem.

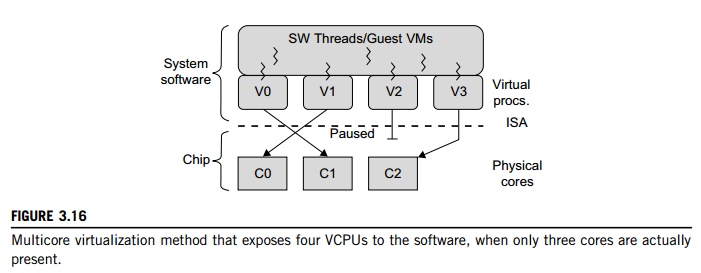
Concerning the first challenge, new programming models, languages, and libraries are needed to make parallel programming easier. The second challenge has spawned research involving scheduling algorithms and resource management policies. Yet these efforts cannot balance well among performance, complexity, and other issues. What is worse, as technology scales, a new challenge called dynamic heterogeneity is emerging to mix the fat CPU core and thin GPU cores on the same chip, which further complicates the multi-core or many-core resource management. The dynamic heterogeneity of hardware infrastructure mainly comes from less reliable transistors and increased complexity in using the transistors [33,66].

**5.1 Physical versus Virtual Processor Cores**

Wells, et al. [74] proposed a multicore virtualization method to allow hardware designers to get an abstraction of the low-level details of the processor cores. This technique alleviates the burden and inefficiency of managing hardware resources by software. It is located under the ISA and remains unmodified by the operating system or VMM (hypervisor). Figure 3.16 illustrates the technique of a software-visible VCPU moving from one core to another and temporarily suspending execution of a VCPU when there are no appropriate cores on which it can run.

**5.2 Virtual Hierarchy**

The emerging many-core chip multiprocessors (CMPs) provides a new computing landscape. Instead of supporting time-sharing jobs on one or a few cores, we can use the abundant cores in a space-sharing, where single-threaded or multithreaded jobs are simultaneously assigned to separate groups of cores for long time intervals. This idea was originally suggested by Marty and Hill [39]. To optimize for space-shared workloads, they propose using virtual hierarchies to overlay a coherence and caching hierarchy onto a physical processor. Unlike a fixed physical hierarchy, a virtual hierarchy can adapt to fit how the work is space shared for improved performance and performance isolation.



Today’s many-core CMPs use a physical hierarchy of two or more cache levels that statically determine the cache allocation and mapping. A virtual hierarchy is a cache hierarchy that can adapt to fit the workload or mix of workloads [39]. The hierarchy’s first level locates data blocks close to the cores needing them for faster access, establishes a shared-cache domain, and establishes a point of coherence for faster communication. When a miss leaves a tile, it first attempts to locate the block (or sharers) within the first level. The first level can also pro-vide isolation between independent workloads. A miss at the L1 cache can invoke the L2 access.

The idea is illustrated in Figure 3.17(a). Space sharing is applied to assign three workloads to three clusters of virtual cores: namely VM0 and VM3 for database workload, VM1 and VM2 for web server workload, and VM4–VM7 for middleware workload. The basic assumption is that each workload runs in its own VM. However, space sharing applies equally within a single operating system. Statically distributing the directory among tiles can do much better, provided operating sys-tems or hypervisors carefully map virtual pages to physical frames. Marty and Hill suggested a two-level virtual coherence and caching hierarchy that harmonizes with the assignment of tiles to the virtual clusters of VMs.

Figure 3.17(b) illustrates a logical view of such a virtual cluster hierarchy in two levels. Each VM operates in a isolated fashion at the first level. This will minimize both miss access time and performance interference with other workloads or VMs. Moreover, the shared resources of cache capacity, inter-connect links, and miss handling are mostly isolated between VMs. The second level maintains a globally shared memory. This facilitates dynamically repartitioning resources without costly cache flushes. Furthermore, maintaining globally shared memory minimizes changes to existing system software and allows virtualization features such as content-based page sharing. A virtual hierarchy adapts to space-shared workloads like multiprogramming and server consolidation. Figure 3.17 shows a case study focused on consolidated server workloads in a tiled architecture. This many-core mapping scheme can also optimize for space-shared multiprogrammed workloads in a single-OS environment.

