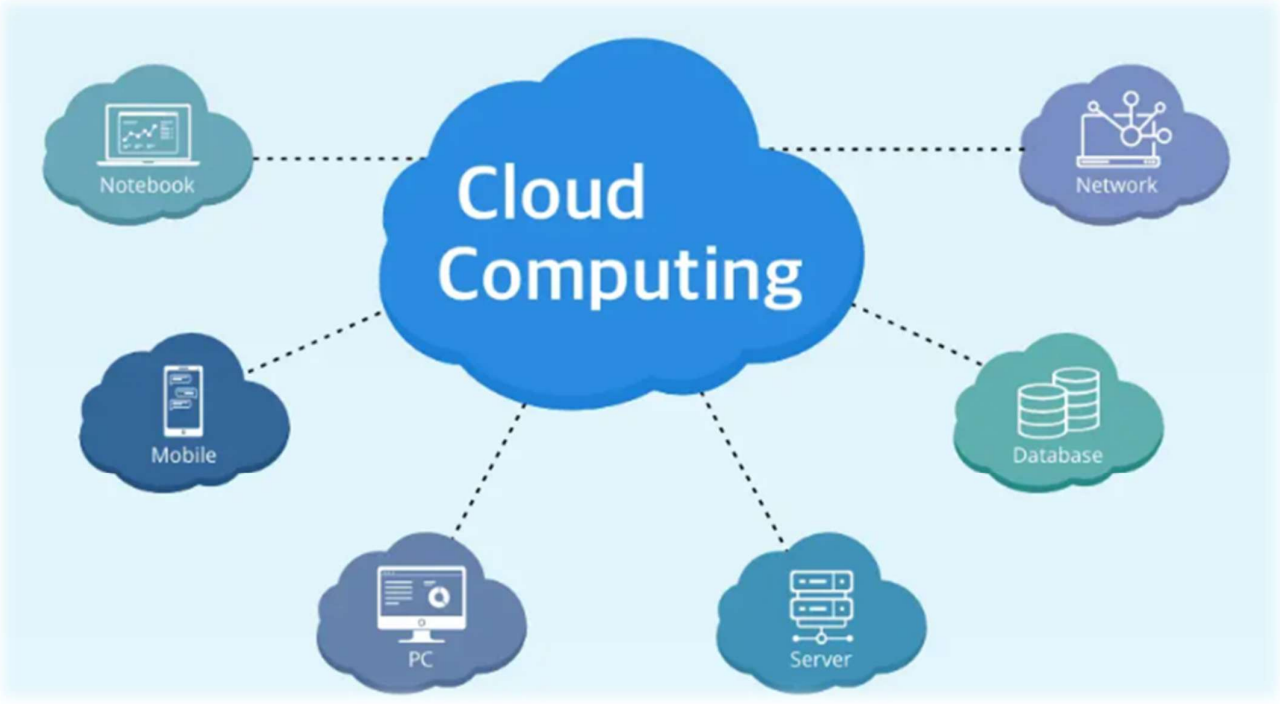
COURSE OBJECTIVE_03

VIRTUAL MACHINES AND VIRTUALIZATION:

- Implementation Levels of Virtualization
- VMM Design Requirements and Providers
- Virtualization Support at the OS Level
- Middleware Support for Virtualization
- Virtualization Structures/Tools and Mechanisms
- Virtualization of CPU
- Memory and I/O Devices
- Virtual Clusters and Resource Management



VIRTUAL MACHINES AND VIRTUALIZATION:

🧠 1. What is Virtualization?

Virtualization is a technology that allows you to create multiple virtual versions of computer resources — like hardware platforms, storage devices, or operating systems — on a single physical machine.

🔹 In simple words:

Virtualization lets one physical computer act like many computers.

💡 Example:

Imagine you have one powerful computer (host).
Using virtualization software, you can create:

- Windows 11 VM
- Linux VM
- macOS VM

All running simultaneously on the same hardware — each behaving like a separate machine.

⚙️ Types of Virtualization:

Type	Description	Example
Hardware Virtualization	Creates virtual machines using a hypervisor.	VMware, VirtualBox
OS Virtualization	Runs multiple isolated systems (containers) on one OS.	Docker, LXC
Storage Virtualization	Combines multiple storage devices into a single logical storage unit.	Storage Area Networks (SANs)
Network Virtualization	Combines physical network resources into one logical network.	SDN (Software Defined Networking)
Desktop Virtualization	Lets users access desktop environments remotely.	VDI (Virtual Desktop Infrastructure)

💻 2. What is a Virtual Machine (VM)?

A Virtual Machine (VM) is a software-based emulation of a physical computer.
Each VM has:

- Its own Operating System (Guest OS)
- Its own Applications
- Virtual hardware: CPU, RAM, Disk, and Network interfaces

⚙️ Hypervisor (Virtual Machine Monitor - VMM):

The hypervisor is the key software that creates and manages virtual machines.

Types of Hypervisors:

Type	Description	Example
Type 1 (Bare Metal)	Runs directly on hardware	VMware ESXi, Microsoft Hyper-V, Xen
Type 2 (Hosted)	Runs on top of a host OS	Oracle VirtualBox, VMware Workstation

---

📊 3. Advantages of Virtualization

Benefit	Description
Resource Utilization	Uses hardware more efficiently by sharing among VMs.
Cost Saving	Reduces need for multiple physical servers.
Isolation	Each VM runs independently — errors don't affect others.
Flexibility	Easy to create, delete, or move VMs.
Testing and Development	Ideal for running different OSes and testing environments.
Disaster Recovery	Easier to back up and restore VMs.

⚠️ 4. Disadvantages of Virtualization

Limitation	Explanation
Performance Overhead	VMs are slower than physical machines due to virtualization layers.
Hardware Dependency	Requires powerful CPU and memory resources.
Complex Management	Managing many VMs can be complicated.
Security Risks	If the hypervisor is compromised, all VMs are at risk.

🧩 5. Real-World Examples:

- VMware vSphere → Data centre virtualization
- Microsoft Azure / AWS EC2 → Cloud virtual machines
- Oracle VirtualBox → Personal use for OS testing
- Docker → Lightweight OS-level virtualization (containers)

📜 Summary Table

Concept	Description
Virtualization	Technique of creating virtual versions of computing resources
Virtual Machine	Software-based emulation of a physical computer
Hypervisor	Software layer that manages and runs VMs
Type 1 Hypervisor	Runs directly on hardware
Type 2 Hypervisor	Runs on a host operating system

Virtualization in Cloud Computing and Types

Virtualization is a way to use one computer as if it were many. Before virtualization, most computers were only doing one job at a time, and a lot of their power was wasted. Virtualization lets you run several virtual computers on one real computer, so you can use its full power and do more tasks at once. In cloud computing, this idea is taken further. Cloud providers use virtualization to split one big server into many smaller virtual ones, so businesses can use just what they need, no extra hardware, no extra cost.

Let us understand virtualization by taking a real-world example:
Suppose there is a company that requires servers for four different purposes:

- Store customer data securely
- Host an online shopping website
- Process employee payroll systems
- Run Social media campaign software for marketing

All these tasks require different things:

- The customer data server requires a lot of space and a Windows operating system.
- The online shopping website requires a high-traffic server and needs a Linux operating system.
- The payroll system requires greater internal memory (RAM) and must use a certain version of the operating system.

In order to fulfill these requirements, the company initially configures four individual physical servers, each for a different purpose. This implies that the company needs to purchase four servers, keep them running, and upgrade them individually, which is very expensive.

Now, by utilizing virtualization, the company can run these four applications on a few physical servers through multiple virtual machines (VMs). Each VM will behave as an independent server, possessing its own operating system and resources. Through this means, the company can cut down on expenses, conserve resources, and manage everything from a single location with ease.

Working of Virtualization

Virtualizations use special software known as hypervisor, to create many virtual computers (cloud instances) on one physical computer. The Virtual Machines behave like actual computers but use the same physical machine.

Virtual Machines (Cloud Instances)

- After installing virtualization software, you can create one or more virtual machines on your computer.
- Virtual machines (VMs) behave like regular applications on your system.
- The real physical computer is called the **Host**, while the virtual machines are called **Guests**.

- A single host can run multiple guest virtual machines.
- Each guest can have its own operating system, which may be the same or different from the host OS.
- Every virtual machine function like a standalone computer, with its own settings, programs, and configuration.
- VMs access system resources such as **CPU, RAM, and storage**, but they work as if they are using their own hardware.

Hypervisors

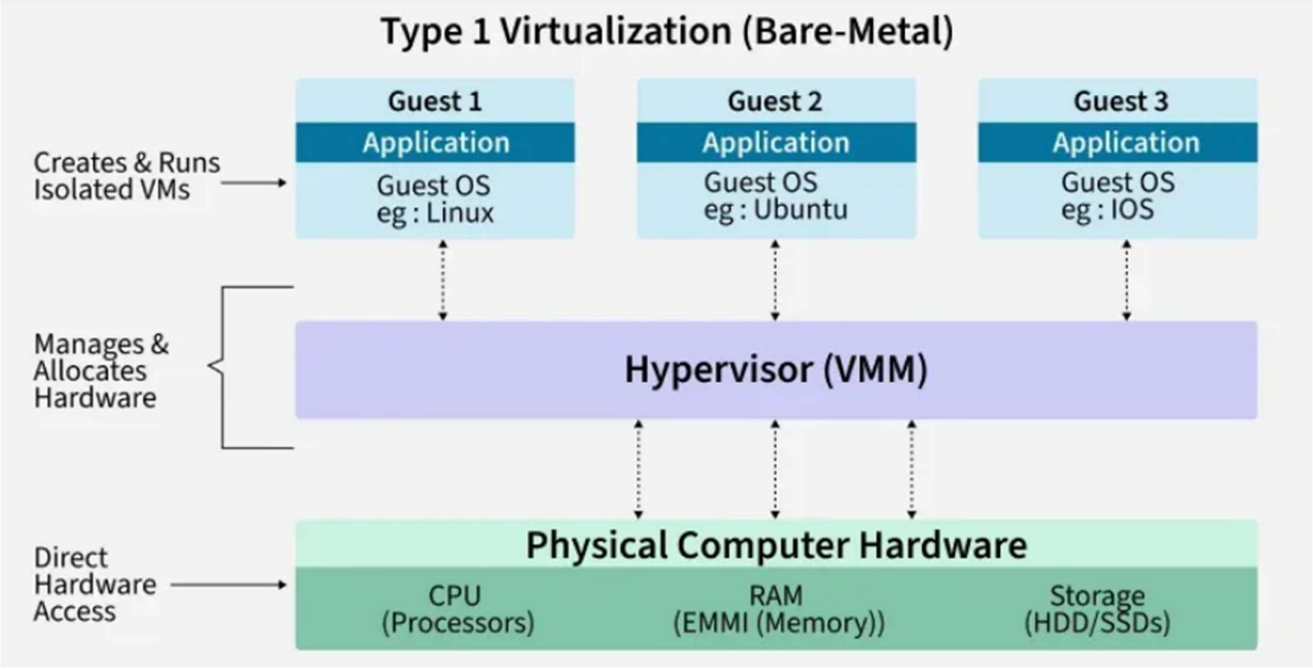
A hypervisor is the software that gets virtualization to work. It serves as an intermediary between the physical computer and the virtual machines. The hypervisor controls the virtual machines' use of the physical resources (such as the CPU and memory) of the host computer.

For instance, if one virtual machine wants additional computing capability, it requests it from the hypervisor. The hypervisor ensures the request is forwarded to the physical hardware, and it's accomplished.

There exist two categories of hypervisors:

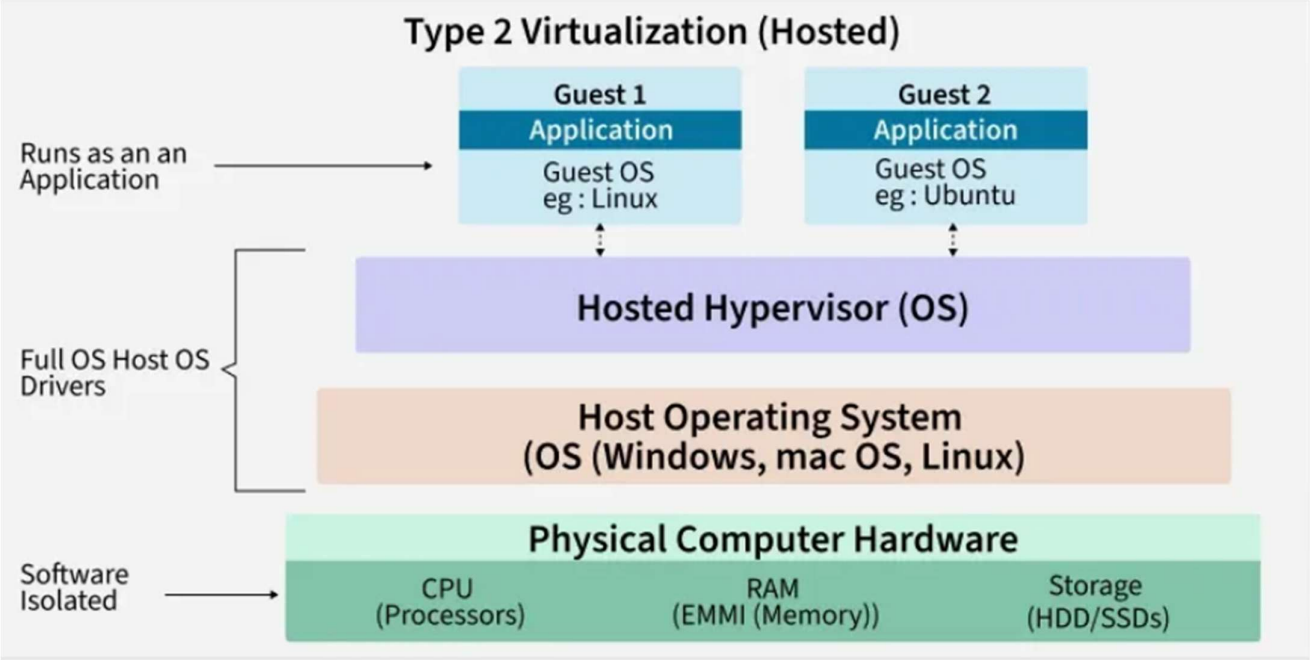
Type 1 Hypervisor (Bare-Metal Hypervisor):

- The hypervisor is installed directly onto the computer hardware, without an operating system sitting in between.
- It is highly efficient as it has a direct access to the resources of the computer.



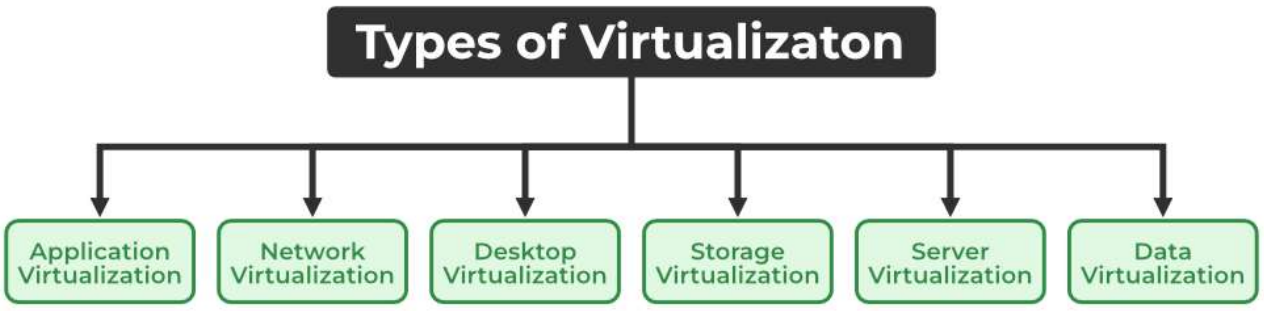
Type 2 Hypervisor:

- It is run over an installed operating system (such as Windows or macOS).
- It's employed when you need to execute more than one operating system on one machine.



Types of Virtualizations

- ✓ Application Virtualization
- ✓ Network Virtualization
- ✓ Desktop Virtualization
- ✓ Storage Virtualization
- ✓ Server Virtualization
- ✓ Data virtualization

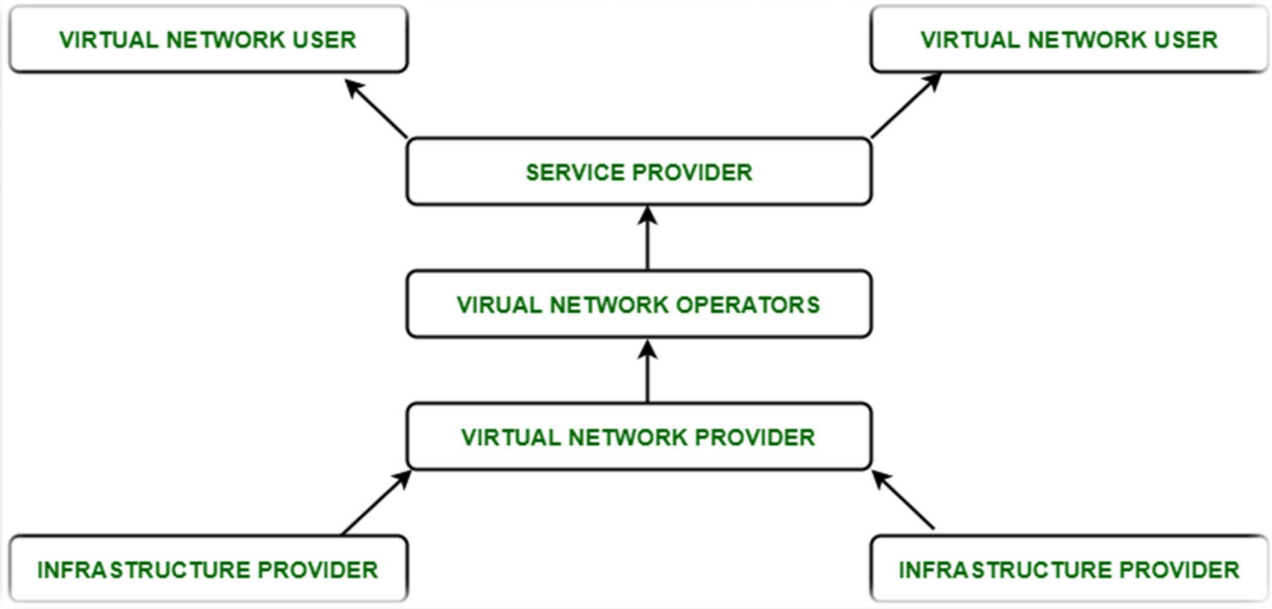


1. Application Virtualization: Application virtualization enables remote access by which users can directly interact with deployed applications without installing them on their local machine. Your personal data and the applications settings are stored on the server, but you can still run it locally via the internet. It's useful if you need to work with multiple versions of the same software. Common examples include hosted or packaged apps.

Example:
Microsoft Azure lets people use their applications without putting them on their own computers. Once this application is setup in the cloud then employees can use it from any device, like a laptop or tablet. It feels like the application is on their computer, but it's really running on Azure's servers. This makes things easier, faster, and safer for the company.

2. Network Virtualization: This allows multiple virtual networks to run on the same physical network, each operating independently. You can quickly set up virtual switches, routers, firewalls, and VPNs, making network management more flexible and efficient.

Example:
Google Cloud is an example of Network Virtualization. Companies create their own networks using software instead of physical devices with the help of Google Cloud. They can set up things like IP addresses, firewalls, and private connections all in the cloud. This makes it easy to manage, change, and grow their network without buying any hardware. It saves time, money, and gives more flexibility.

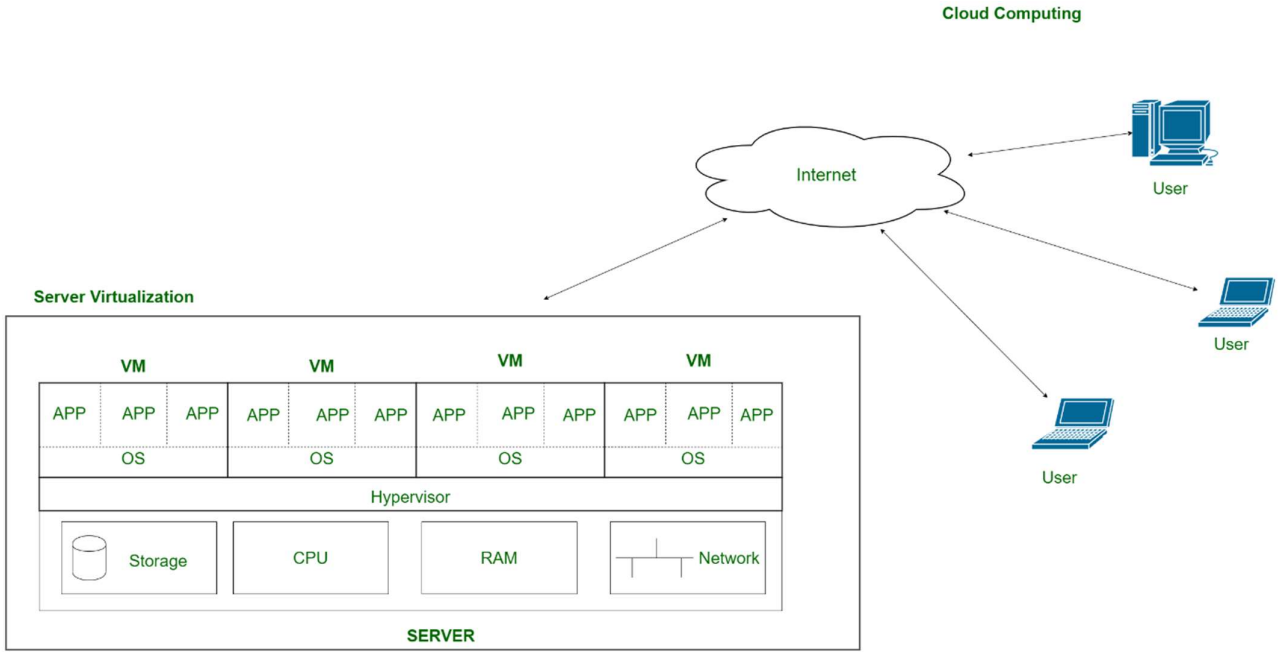


3. Desktop Virtualization: Desktop virtualization is a process in which you can create different virtual desktops that users can use from any device like laptop, tablet. It's great for users who need flexibility, as it simplifies software updates and provides portability.

Example:
GeeksforGeeks is a Edtech company which uses services like Amazon WorkSpaces or Google Cloud (GCP) Virtual Desktops to give its team members access to the same coding setup with all the tools they required for the easy access of this teamwork. Now their team members can easily log in from any device like a laptop, tablet, or even a phone and use a virtual desktop that will run perfectly in the cloud. This makes it easy for GeeksforGeeks company to manage, update, and keep everything secure without requirement of physical computers for everyone.

4. Storage Virtualization: This combines storage from different servers into a single system, making it easier to manage. It ensures smooth performance and efficient operations even when the underlying hardware changes or fails.

Example:
Amazon S3 is an example of storage virtualization because in S3 we can easily store any amount of data from anywhere. Suppose a MNC have lots of files and data of company to store. By Amazon S3 company can store all their files and data in one place and access these from anywhere without any kind of issue in secure way.



6. Data Virtualization: This brings data from different sources together in one place without needing to know where or how it's stored. It creates a unified view of the data, which can be accessed remotely via cloud services.

Example:
Companies like Oracle and IBM offer solutions for this.

What is Virtualization?

Virtualization is technology that you can use to create virtual representations of servers, storage, networks, and other physical machines. Virtual software mimics the functions of physical hardware to run multiple virtual machines simultaneously on a single physical machine. Businesses use virtualization to use their hardware resources efficiently and get greater returns from their investment. It also powers cloud computing services that help organizations manage infrastructure more efficiently.

Why is virtualization important?

By using virtualization, you can interact with any hardware resource with greater flexibility. Physical servers consume electricity, take up storage space, and need maintenance. You are often limited by physical proximity and network design if you want to access them. Virtualization removes all these limitations by abstracting physical hardware functionality into software. You can manage, maintain, and use your hardware infrastructure like an application on the web.

Virtualization example

Consider a company that needs servers for three functions:

1. Store business email securely
2. Run a customer-facing application
3. Run internal business applications

Each of these functions has different configuration requirements:

- The email application requires more storage capacity and a Windows operating system.
- The customer-facing application requires a Linux operating system and high processing power to handle large volumes of website traffic.
- The internal business application requires iOS and more internal memory (RAM).

To meet these requirements, the company sets up three different dedicated physical servers for each application. The company must make a high initial investment and perform ongoing maintenance and upgrades for one machine at a time. The company also cannot optimize its computing capacity. It pays 100% of the servers' maintenance costs but uses only a fraction of their storage and processing capacities.

Efficient hardware use

With virtualization, the company creates three digital servers, or virtual machines, on a single physical server. It specifies the operating system requirements for the virtual machines and can use them like the physical servers. However, the company now has less hardware and fewer related expenses.

Infrastructure as a service

The company can go one step further and use a cloud instance or virtual machine from a cloud computing provider such as AWS. AWS manages all the underlying hardware, and the company can request server resources with varying configurations. All the applications run on these virtual

servers without the users noticing any difference. Server management also becomes easier for the company's IT team.

What is virtualization?

To properly understand Kernel-based Virtual Machine (KVM), you first need to understand some basic concepts in *virtualization*. Virtualization is a process that allows a computer to share its hardware resources with multiple digitally separated environments. Each virtualized environment runs within its allocated resources, such as memory, processing power, and storage. With virtualization, organizations can switch between different operating systems on the same server without rebooting.

Virtual machines and hypervisors are two important concepts in virtualization.

Virtual machine

A *virtual machine* is a software-defined computer that runs on a physical computer with a separate operating system and computing resources. The physical computer is called the *host machine* and virtual machines are *guest machines*. Multiple virtual machines can run on a single physical machine. Virtual machines are abstracted from the computer hardware by a hypervisor.

Hypervisor

The *hypervisor* is a software component that manages multiple virtual machines in a computer. It ensures that each virtual machine gets the allocated resources and does not interfere with the operation of other virtual machines.

There are two types of hypervisors.

Type 1 hypervisor

A type 1 hypervisor, or bare-metal hypervisor, is a hypervisor program installed directly on the computer's hardware instead of the operating system. Therefore, type 1 hypervisors have better performance and are commonly used by enterprise applications. KVM uses the type 1 hypervisor to host multiple virtual machines on the Linux operating system.

Type 2 hypervisor

Also known as a hosted hypervisor, the type 2 hypervisor is installed on an operating system. Type 2 hypervisors are suitable for end-user computing.

What are the benefits of virtualization?

Virtualization provides several benefits to any organization:

Efficient resource use

Virtualization improves hardware resources used in your data centre. For example, instead of running one server on one computer system, you can create a virtual server pool on the same computer system by using and returning servers to the pool as required. Having fewer underlying physical servers frees up space in your data centre and saves money on electricity, generators, and cooling appliances.

Automated IT management

Now that physical computers are virtual, you can manage them by using software tools. Administrators create deployment and configuration programs to define virtual machine templates. You can duplicate your infrastructure repeatedly and consistently and avoid error-prone manual configurations.

Faster disaster recovery

When events such as natural disasters or cyberattacks negatively affect business operations, regaining access to IT infrastructure and replacing or fixing a physical server can take hours or even days. By contrast, the process takes minutes with virtualized environments. This prompt response significantly improves resiliency and facilitates business continuity so that operations can continue as scheduled.

How does virtualization work?

Virtualization uses specialized software, called a hypervisor, to create several cloud instances or virtual machines on one physical computer.

Cloud instances or virtual machines

After you install virtualization software on your computer, you can create one or more virtual machines. You can access the virtual machines in the same way that you access other applications on your computer. Your computer is called the host, and the virtual machine is called the guest. Several guests can run on the host. Each guest has its own operating system, which can be the same or different from the host operating system.

From the user's perspective, the virtual machine operates like a typical server. It has settings, configurations, and installed applications. Computing resources, such as central processing units (CPUs), Random Access Memory (RAM), and storage appear the same as on a physical server. You can also configure and update the guest operating systems and their applications as necessary without affecting the host operating system.

Hypervisors

The hypervisor is the virtualization software that you install on your physical machine. It is a software layer that acts as an intermediary between the virtual machines and the underlying hardware or host operating system. The hypervisor coordinates access to the physical environment so that several virtual machines have access to their own share of physical resources.

For example, if the virtual machine requires computing resources, such as computer processing power, the request first goes to the hypervisor. The hypervisor then passes the request to the underlying hardware, which performs the task.

The following are the two main types of hypervisors.

Type 1 hypervisors

A type 1 hypervisor—also called a bare-metal hypervisor—runs directly on the computer hardware. It has some operating system capabilities and is highly efficient because it interacts directly with the physical resources.

Type 2 hypervisors

A type 2 hypervisor runs as an application on computer hardware with an existing operating system. Use this type of hypervisor when running multiple operating systems on a single machine.

What are the different types of virtualizations?

You can use virtualization technology to get the functions of many different types of physical infrastructure and all the benefits of a virtualized environment. You can go beyond virtual machines to create a collection of virtual resources in your virtual environment.

Server virtualization

Server virtualization is a process that partitions a physical server into multiple virtual servers. It is an efficient and cost-effective way to use server resources and deploy IT services in an organization. Without server virtualization, physical servers use only a small amount of their processing capacities, which leave devices idle.

Storage virtualization

Storage virtualization combines the functions of physical storage devices such as network attached storage (NAS) and storage area network (SAN). You can pool the storage hardware in your data centre, even if it is from different vendors or of different types. Storage virtualization uses all your physical data storage and creates a large unit of virtual storage that you can assign and control by using management software. IT administrators can streamline storage activities, such as archiving, backup, and recovery, because they can combine multiple network storage devices virtually into a single storage device.

Network virtualization

Any computer network has hardware elements such as switches, routers, and firewalls. An organization with offices in multiple geographic locations can have several different network technologies working together to create its enterprise network. Network virtualization is a process that combines all of these network resources to centralize administrative tasks. Administrators can adjust and control these elements virtually without touching the physical components, which greatly simplifies network management.

The following are two approaches to network virtualization.

Software-defined networking

Software-defined networking (SDN) controls traffic routing by taking over routing management from data routing in the physical environment. For example, you can program your system to prioritize your video call traffic over application traffic to ensure consistent call quality in all online meetings.

Network function virtualization

Network function virtualization technology combines the functions of network appliances, such as firewalls, load balancers, and traffic analyzers that work together, to improve network performance.

Data virtualization

Modern organizations collect data from several sources and store it in different formats. They might also store data in different places, such as in a cloud infrastructure and an on-premises data center. Data virtualization creates a software layer between this data and the applications that need it. Data virtualization tools process an application's data request and return results in a suitable format. Thus, organizations use data virtualization solutions to increase flexibility for data integration and support cross-functional data analysis.

Application virtualization

Application virtualization pulls out the functions of applications to run on operating systems other than the operating systems for which they were designed. For example, users can run a Microsoft Windows application on a Linux machine without changing the machine configuration.

To achieve application virtualization, follow these practices:

- Application streaming – Users stream the application from a remote server, so it runs only on the end user's device when needed.
- Server-based application virtualization – Users can access the remote application from their browser or client interface without installing it.
- Local application virtualization – The application code is shipped with its own environment to run on all operating systems without changes.

Desktop virtualization

Most organizations have nontechnical staff that use desktop operating systems to run common business applications. For instance, you might have the following staff:

- A customer service team that requires a desktop computer with Windows 10 and customer-relationship management software
- A marketing team that requires Windows Vista for sales applications

You can use desktop virtualization to run these different desktop operating systems on virtual machines, which your teams can access remotely. This type of virtualization makes desktop management efficient and secure, saving money on desktop hardware. The following are types of desktop virtualization.

Virtual desktop infrastructure

Virtual desktop infrastructure runs virtual desktops on a remote server. Your users can access them by using client devices.

Local desktop virtualization

In local desktop virtualization, you run the hypervisor on a local computer and create a virtual computer with a different operating system. You can switch between your local and virtual environment in the same way you can switch between applications.

How is virtualization different from cloud computing?

Cloud computing is the on-demand delivery of computing resources over the internet with pay-as-you-go pricing. Instead of buying, owning, and maintaining a physical data centre, you can access technology services, such as computing power, storage, and databases, as you need them from a cloud provider.

Virtualization technology makes cloud computing possible. Cloud providers set up and maintain their own data centres. They create different virtual environments that use the underlying hardware resources. You can then program your system to access these cloud resources by using APIs. Your infrastructure needs can be met as a fully managed service.

How is server virtualization different from containerization?

Containerization is a way to deploy application code to run on any physical or virtual environment without changes. Developers bundle application code with related libraries, configuration files, and other dependencies that the code needs to run. This single package of the software, called a container, can run independently on any platform. Containerization is a type of application virtualization.

You can think of server virtualization as building a road to connect two places. You have to recreate an entire virtual environment and then run your application on it. By comparison, containerization is like building a helicopter that can fly to either of those places. Your application is inside a container and can run on all types of physical or virtual environments.

IMPLEMENTATION LEVELS OF VIRTUALIZATION

🧩 1. Instruction Set Architecture (ISA) Level Virtualization

- **Definition:**
  Virtualization is done at the lowest level — the **processor instruction set** level.
- **How it works:**
  The **Virtual Machine Monitor (VMM)** or **hypervisor** translates guest instructions into host instructions.
- **Example:**
  - Emulators like **Bochs**, **QEMU**, or **VirtualBox** in full emulation mode.
- **Advantage:**
  Can run any OS (even for different CPU architectures).
- **Disadvantage:**
  Very **slow** due to instruction translation overhead.

⚙️ 2. Hardware Level Virtualization

- **Definition:**
  Virtualization is implemented **directly on the physical hardware** using a **hypervisor**.
- **How it works:**
  The hypervisor creates **multiple virtual machines (VMs)** that share the same physical hardware.
- **Examples:**
  - **VMware ESXi**
  - **Microsoft Hyper-V**
  - **KVM (Kernel-based Virtual Machine)**
  - **Xen**
- **Advantage:**
  High performance, efficient resource utilization.
- **Disadvantage:**
  Requires hardware support (Intel VT-x, AMD-V).

💻 3. Operating System Level Virtualization

- **Definition:**
  Virtualization happens **at the OS kernel level**, allowing multiple **isolated user-space instances** (containers).
- **How it works:**
  All containers share the same OS kernel but behave as independent systems.
- **Examples:**
  - **Docker**, **LXC (Linux Containers)**, **OpenVZ**
- **Advantage:**
  Lightweight and fast startup; uses fewer resources.
- **Disadvantage:**
  All containers must use the same OS kernel.

🧠 4. Library (API) Level Virtualization

- **Definition:**
  Virtualization occurs at the **application programming interface (API)** level.
- **How it works:**
  A special library translates system calls of the guest OS into the host OS calls.
- **Example:**
  - **Wine** (to run Windows applications on Linux)
  - **POSIX compatibility layers**
- **Advantage:**
  Enables cross-platform execution.
- **Disadvantage:**
  Limited compatibility and functionality.

5. Application Level Virtualization

- **Definition:**
  Virtualization is implemented at the **application layer**.
- **How it works:**
  The application runs inside a virtual environment that emulates required resources (without installing on the host OS).
- **Examples:**
  - **VMware ThinApp**
  - **Microsoft App-V**
  - **Sandboxie**
- **Advantage:**
  Easy deployment, isolation of applications.
- **Disadvantage:**
  Limited to application behavior; doesn't virtualize full systems.

Summary Table

Level	Virtualized Component	Example	Performance	Isolation
ISA Level	Instruction Set	QEMU, Bochs	Low	High
Hardware Level	Physical Hardware	VMware ESXi, Hyper-V	High	High
OS Level	Operating System Kernel	Docker, LXC	Very High	Medium
Library Level	System Libraries	Wine	Medium	Low
Application Level	Application Environment	App-V, ThinApp	Medium	Medium

Implementation Levels of Virtualization

Virtualization is a computer architecture technology by which multiple virtual machines (VMs) are multiplexed in the same hardware machine. The idea of VMs can be dated back to the 1960s [53]. 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 increased sharply in recent years.

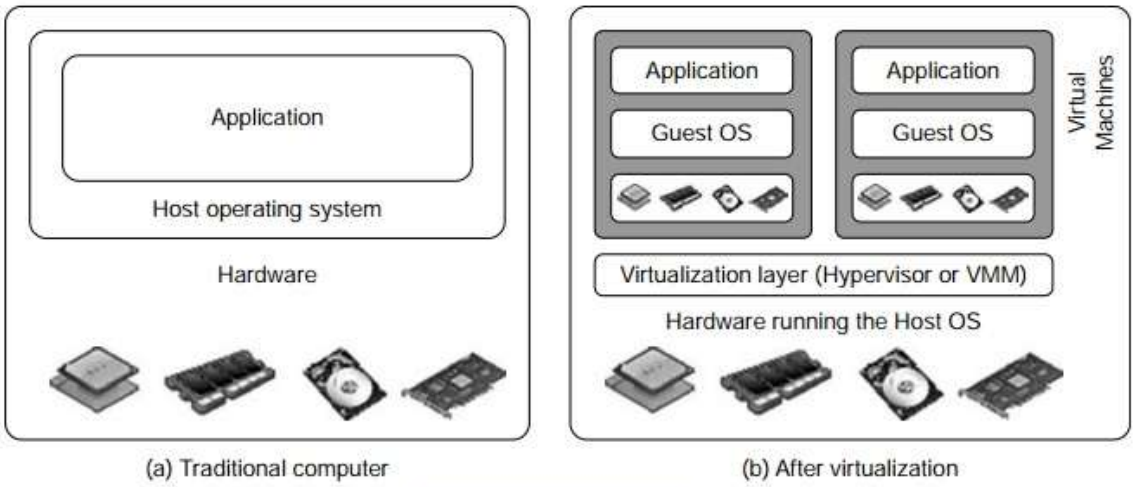
The idea is to separate the hardware from the software to yield better system efficiency. For example, computer users gained access to much enlarged memory space when the concept of virtual memory was introduced. Similarly, virtualization techniques can be applied to enhance the use of compute engines, networks, and storage. In this chapter we will discuss VMs and their applications for building distributed systems. According to a 2009 Gartner Report, virtualization was the top strategic technology poised to change the computer industry. With sufficient storage, any

computer platform can be installed in another host computer, even if they use processors with different instruction sets and run with distinct operating systems on the same hardware.

1. Levels of Virtualization Implementation

A traditional computer runs with a host operating system specially tailored for its hardware architecture, as shown in Figure 3.1(a). After virtualization, different user applications managed by their own operating systems (guest OS) can run on the same hardware, independent of the host OS. This is often done by adding additional software, called a virtualization layer as shown in Figure 3.1(b). This virtualization layer is known as hypervisor or virtual machine monitor (VMM) [54]. The VMs are shown in the upper boxes, where applications run with their own guest OS over the virtualized CPU, memory, and I/O resources.

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, as we will discuss shortly. The virtualization software creates the abstraction of VMs by interposing a virtualization layer at various levels of a computer system. Common virtualization layers include the instruction set architecture (ISA) level, hardware level, operating system level, library support level, and application level (see Figure 3.2).



1.1

FIGURE 3.1

The architecture of a computer system before and after virtualization, where VMM stands for virtual machine monitor.

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 writ-ten 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. The basic blocks can also be extended to program traces or super blocks to increase translation efficiency. 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.

1.2 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 intention is to upgrade the hardware utilization rate by multiple users concurrently. 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. We will discuss hardware virtualization approaches in more detail in Section 3.3.

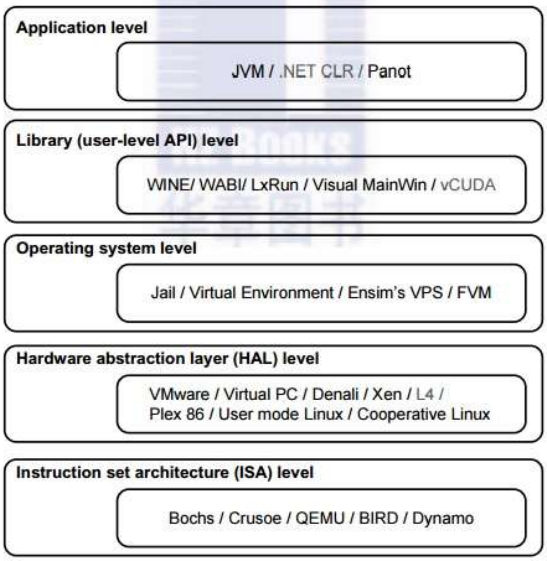


FIGURE 3.2
Virtualization ranging from hardware to applications in five abstraction levels.

1.3 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 many 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. OS-level virtualization is depicted in Section 3.1.3.

1.4 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. This approach is detailed in Section 3.1.4.

1.5 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. The result is an application that is much easier to distribute and remove from user workstations. An example is the LANDesk application virtualization platform which deploys software applications as self-contained, executable files in an isolated environment without requiring installation, system modifications, or elevated security privileges.

1.6 Relative Merits of Different Approaches

Table 3.1 compares the relative merits of implementing virtualization at various levels. The column headings correspond to four technical merits. “Higher Performance” and “Application Flexibility” are self-explanatory. “Implementation Complexity” implies the cost to implement that particular virtualization level. “Application Isolation” refers to the effort required to isolate resources committed to different VMs. Each row corresponds to a particular level of virtualization.

The number of X’s in the table cells reflects the advantage points of each implementation level. Five X’s implies the best case and one X implies the worst case. Overall, hardware and OS support will yield the highest performance. However, the hardware and application levels are also the most expensive to implement. User isolation is the most difficult to achieve. ISA implementation offers the best application flexibility.

2. VMM Design Requirements and Providers

As mentioned earlier, 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 compo-nent, such as the CPU, can be virtualized as several virtual copies. Therefore, several traditional oper-ating systems which are the same or different can sit on the same set of hardware simultaneously.

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	XXXXX	XXX	XXX
Hardware-level virtualization	XXXXX	XXX	XXXXX	XXXX
OS-level virtualization	XXXXX	XX	XXX	XX
Runtime library support	XXX	XX	XX	XX
User application level	XX	XX	XXXXX	XXXXX

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. Two possible exceptions in terms of differences are permitted with this requirement: differences caused by the availability of system resources and differences caused by timing dependencies. The former arises when more than one VM is running on the same machine.

The hardware resource requirements, such as memory, of each VM are reduced, but the sum of them is greater than that of the real machine installed. The latter qualification is required because of the intervening level of software and the effect of any other VMs concurrently existing on the same hardware. Obviously, these two differences pertain to performance, while the function a VMM provides stays the same as that of a real machine. However, the identical environment requirement excludes the behavior of the usual time-sharing operating system from being classed as a VMM.

A VMM should demonstrate efficiency in using the VMs. Compared with a physical machine, no one prefers a VMM if its efficiency is too low. Traditional emulators and complete software interpreters (simulators) emulate each instruction by means of functions or macros. Such a method provides the most flexible solutions for VMMs. However, emulators or simulators are too slow to be used as real machines. To guarantee the efficiency of a VMM, a statistically dominant subset of the virtual processor's instructions needs to be executed directly by the real processor, with no software intervention by the VMM. Table 3.2 compares four hypervisors and VMMs that are in use today.

Complete control of these resources by a 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. Not all processors satisfy these requirements for a VMM. A VMM is tightly related to the architectures of processors. It is difficult to

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

implement a VMM for some types of processors, such as the x86. Specific limitations include the inability to trap on some privileged instructions. If a processor is not designed to support virtualization primarily, it is necessary to modify the hardware to satisfy the three requirements for a VMM. This is known as hardware-assisted virtualization.

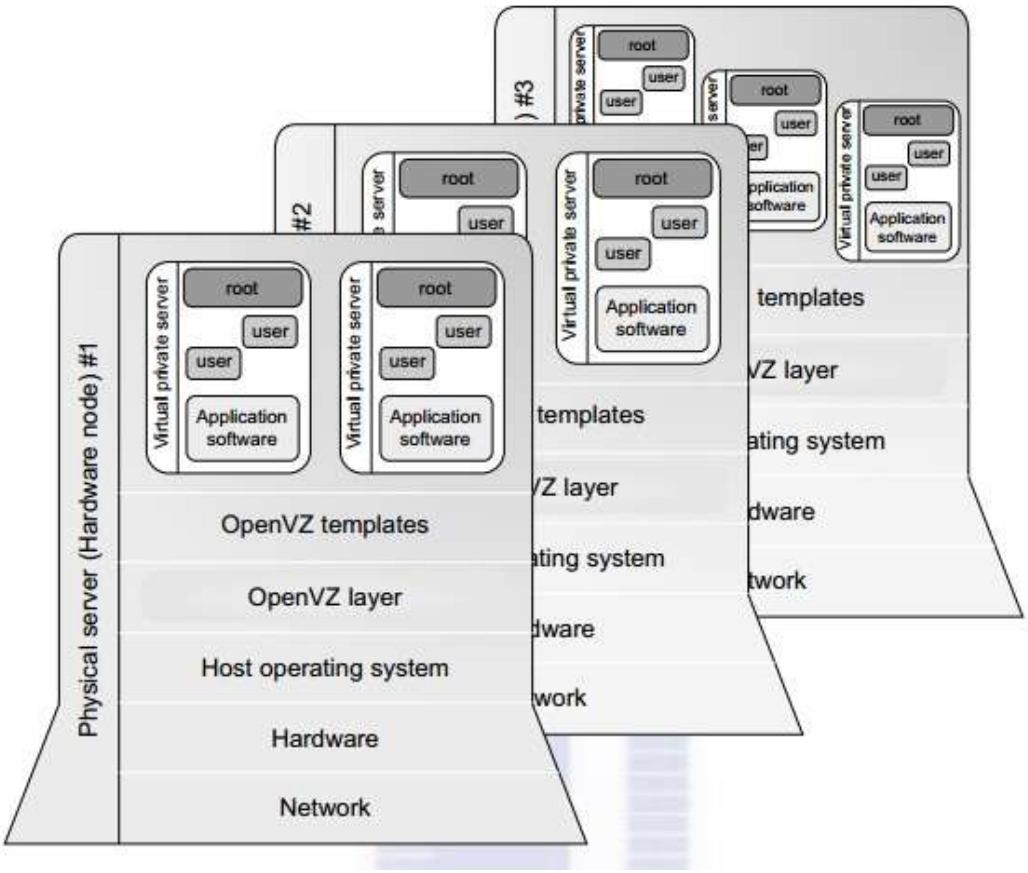
3. Virtualization Support at the OS Level

With the help of VM technology, a new computing mode known as cloud computing is emerging. Cloud computing is transforming the computing landscape by shifting the hardware and staffing costs of managing a computational centre to third parties, just like banks. However, cloud computing has at least two challenges. 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 dur-ing 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. Therefore, to better support cloud computing, a large amount of research and development should be done.

3.1 Why OS-Level Virtualization?

As mentioned earlier, 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 initi-alized simultaneously. Besides slow operation, storing the VM images also becomes an issue. As a matter of fact, there is considerable repeated content among VM images. Moreover, full virtualiza-tion at the hardware level also has the disadvantages of slow performance and low density, and the need for para-virtualization to modify the guest OS. To reduce the performance overhead of hardware-level virtualization, even hardware modification is needed. 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 ser-vers. 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 illustrates operating system virtualization from the point of view of a machine stack.

3.2 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; and (2) for an OS-level VM, it is possible for a VM and its host environment to synchro-nize 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. In cloud computing, the first and second benefits can be used to overcome the defects of slow

initialization of VMs at the hardware level, and being unaware of the current application state, respectively.

3.3 Disadvantages of OS Extensions

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. For example, a Windows distribution such as Windows XP cannot run on a Linux-based container. However, users of cloud computing have various preferences. Some prefer Windows and others prefer Linux or other operating systems. Therefore, there is a challenge for OS-level virtualization in such cases.

Figure 3.3 illustrates the concept of OS-level virtualization. The virtualization layer is inserted inside the OS to partition the hardware resources for multiple VMs to run their applications in multiple virtual environments. To implement OS-level virtualization, isolated execution environ-ments (VMs) should be created based on a single OS kernel. Furthermore, the access requests from a VM need to be redirected to the VM's local resource partition on the physical machine. For example, the chroot command in a UNIX system can create several virtual root directories within a host OS. These virtual root directories are the root directories of all VMs created.

There are two ways to implement virtual root directories: duplicating common resources to each VM partition; or sharing most resources with the host environment and only creating private resource copies on the VM on demand. The first way incurs significant resource costs and overhead on a physical machine. This issue neutralizes the benefits of OS-level virtualization, compared with hardware-assisted virtualization. Therefore, OS-level virtualization is often a second choice.

3.4 Virtualization on Linux or Windows Platforms

By far, most reported OS-level virtualization systems are Linux-based. Virtualization support on the Windows-based platform is still in the research stage. The Linux kernel offers an abstraction layer to allow software processes to work with and operate on resources without knowing the hardware details. New hardware may need a new Linux kernel to support. Therefore, different Linux plat-forms use patched kernels to provide special support for extended functionality.

However, most Linux platforms are not tied to a special kernel. In such a case, a host can run several VMs simultaneously on the same hardware. Table 3.3 summarizes several examples of OS-level virtualization tools that have been developed in recent years. Two OS tools (Linux vServer and OpenVZ) support Linux platforms to run other platform-based applications through virtualiza-tion. These two OS-level tools are illustrated in Example 3.1. The third tool, FVM, is an attempt specifically developed for virtualization on the Windows NT platform.

Example 3.1 Virtualization Support for the Linux Platform

OpenVZ is an OS-level tool designed to support Linux platforms to create virtual environments for running VMs under different guest OSes. OpenVZ is an open source container-based virtualization solution built on Linux. To support virtualization and isolation of various subsystems, limited resource management, and checkpointing, OpenVZ modifies the Linux kernel. The overall picture of the OpenVZ system is illustrated in Figure 3.3. Several VPSes can run simultaneously on a physical machine. These VPSes look like normal

Table 3.3 Virtualization Support for Linux and Windows NT Platforms

Table 3.3 Virtualization Support for Linux and Windows NT Platforms

Virtualization Support and Source of Information	Brief Introduction on Functionality and Application Platforms
Linux vServer for Linux platforms (http://linux-vserver.org/)	Extends Linux kernels to implement a security mechanism to help build VMs by setting resource limits and file attributes and changing the root environment for VM isolation
OpenVZ for Linux platforms [65]; http://ftp.openvz.org/doc/OpenVZ-Users-Guide.pdf)	Supports virtualization by creating *virtual private servers (VPSes)*; the VPS has its own files, users, process tree, and virtual devices, which can be isolated from other VPSes, and checkpointing and live migration are supported
FVM (Feather-Weight Virtual Machines) for virtualizing the Windows NT platforms [78])	Uses system call interfaces to create VMs at the NY kernel space; multiple VMs are supported by virtualized namespace and copy-on-write

Linux servers. Each VPS has its own files, users and groups, process tree, virtual network, virtual devices, and IPC through semaphores and messages.

The resource management subsystem of OpenVZ consists of three components: two-level disk alloca-tion, a two-level CPU scheduler, and a resource controller. The amount of disk space a VM can use is set by the OpenVZ server administrator. This is the first level of disk allocation. Each VM acts as a standard Linux system. Hence, the VM administrator is responsible for allocating disk space for each user and group. This is the second-level disk quota. The first-level CPU scheduler of OpenVZ decides which VM to give the time slice to, taking into account the virtual CPU priority and limit settings.The second-level CPU scheduler is the same as that of Linux. OpenVZ has a set of about 20 parameters which are carefully chosen to cover all aspects of VM operation. Therefore, the resources that a VM can use are well controlled. OpenVZ also supports checkpointing and live migration. The complete state of a VM can quickly be saved to a disk file. This file can then be transferred to another physical machine and the VM can be restored there. It only takes a few seconds to complete the whole process. However, there is still a delay in processing because the established network connections are also migrated.

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

4. 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. This section provides an overview of several library-level virtualization systems: namely the Windows Application Binary Interface (WABI), lxrun, WINE, Visual MainWin, and vCUDA, which are summarized in Table 3.4.

The WABI offers middleware to convert Windows system calls to Solaris system calls. Lxrun is really a system call emulator that enables Linux applications written for x86 hosts to run on UNIX systems. Similarly, Wine offers library support for virtualizing x86 processors to run Windows appli-cations on UNIX hosts. Visual MainWin offers a compiler support system to develop Windows appli-cations using Visual Studio to run on some UNIX hosts. The vCUDA is explained in Example 3.2 with a graphical illustration in Figure 3.4.

Example 3.2 The vCUDA for Virtualization of General-Purpose GPUs

CUDA is a programming model and library for general-purpose GPUs. It leverages the high performance of GPUs to run compute-intensive applications on host operating systems. However, it is difficult to run CUDA applications on hardware-level VMs directly. vCUDA virtualizes the CUDA library and can be installed on guest OSes. When CUDA applications run on a guest OS and issue a call to the CUDA API, vCUDA intercepts the call and redirects it to the CUDA API running on the host OS. Figure 3.4 shows the basic concept of the vCUDA architecture .The vCUDA employs a client-server model to implement CUDA virtualization. It consists of three user space components:

the vCUDA library, a virtual GPU in the guest OS (which acts as a client), and the vCUDA stub in the host OS (which acts as a server). The vCUDA library resides in the guest OS as a substitute for the standard CUDA library. It is responsible for intercepting and redirecting API calls from the client to the stub. Besides these tasks, vCUDA also creates vGPUs and manages them.

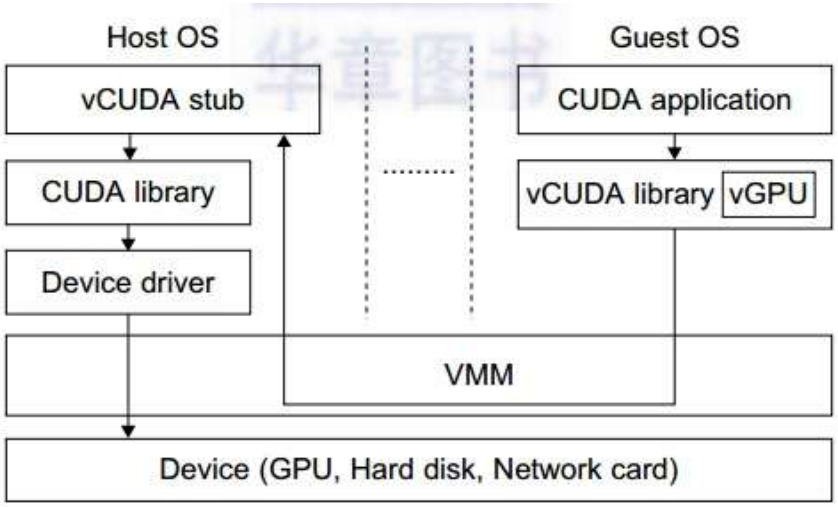


FIGURE 3.4
Basic concept of the vCUDA architecture.

The functionality of a vGPU is threefold: It abstracts the GPU structure and gives applications a uniform view of the underlying hardware; when a CUDA application in the guest OS allocates a device's memory the vGPU can return a local virtual address to the application and notify the remote stub to allocate the real device memory, and the vGPU is responsible for storing the CUDA API flow. The vCUDA stub receives and interprets remote requests and creates a corresponding execution context for the API calls from the guest OS, then returns the results to the guest OS. The vCUDA stub also manages actual physical resource allocation.

VMM DESIGN REQUIREMENTS AND PROVIDERS

🧠 VMM (Virtual Machine Monitor) – Overview

- The Virtual Machine Monitor (VMM) — also called a Hypervisor — is the core software that enables virtualization.
- It creates and manages Virtual Machines (VMs) and controls how they use the underlying hardware resources (CPU, memory, storage, and I/O).

⚙️ VMM Design Requirements

A well-designed VMM must satisfy several key requirements to ensure security, efficiency, and resource management.

Let's go through them one by one 👇

🧩 1. Fidelity (Equivalence)

- The VMM should faithfully reproduce the behavior of the actual hardware.

- A program running inside a virtual machine should behave exactly as it would on a real physical machine.
- ⚡ Goal: Make the VM environment indistinguishable from real hardware.

Example:
If you install Windows 11 in a VM, it should behave the same way as on a real PC.

🛡️ 2. Performance (Efficiency)

- Most instructions should run directly on the physical CPU without interpretation.
- The VMM should minimize overhead and allow efficient execution.
- Hardware features like Intel VT-x or AMD-V help achieve this.

Example:
KVM and VMware use hardware-assisted virtualization to reduce performance loss.

🧱 3. Safety (Resource Control / Isolation)

- The VMM must ensure complete isolation between virtual machines.
- A fault or malicious process in one VM must not affect other VMs or the host.
- The VMM must have complete control of hardware resources (CPU, memory, I/O).

Example:
If one VM crashes, others should keep running safely.

🔐 4. Security

- The VMM should prevent unauthorized access to the host or other VMs.
- Proper access control and sandboxing must be enforced.
- Protects data confidentiality and integrity across virtual machines.

🧩 5. Portability and Scalability

- The VMM should work across different hardware architectures and support scaling to multiple processors or nodes.
- It should also support adding/removing VMs dynamically.

⚙️ 6. Device and I/O Management

- The VMM should efficiently virtualize input/output devices (disks, network cards, USB).
- Provide virtual drivers or paravirtualized interfaces for faster communication.

Example:
VMware Tools or VirtIO drivers improve I/O speed inside virtual machines.

🧰 7. Resource Scheduling and Allocation

- Fair distribution of CPU, memory, and I/O bandwidth among VMs.
- Dynamic resource management to optimize performance.

Types of VMM Providers (Hypervisor Providers)

There are two major types of VMM implementations — based on where the VMM sits in the system.

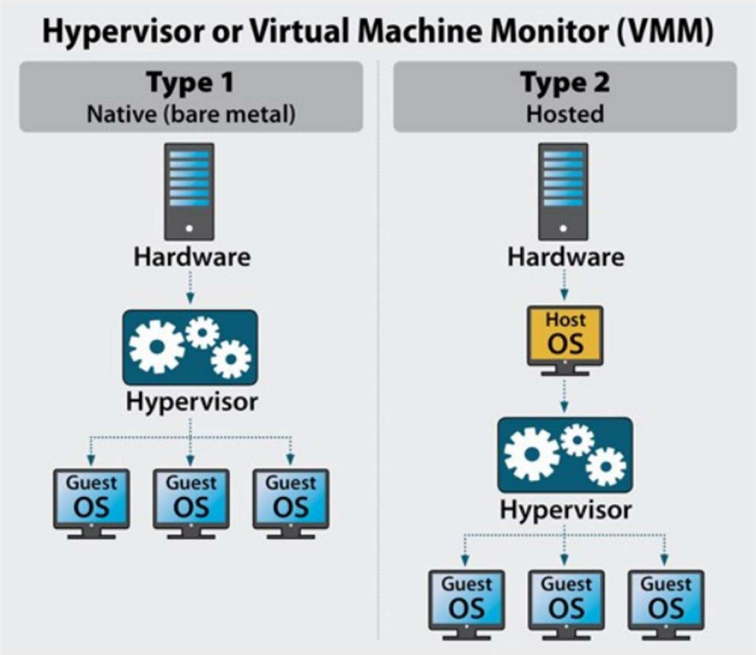
1. Type 1 – Bare Metal (Native) VMM

- Runs directly on the physical hardware.
- The VMM controls the hardware and manages guest operating systems directly.
- More secure and efficient.

Examples:

- VMware ESXi
- Microsoft Hyper-V
- KVM (Linux Kernel-based VMM)
- Xen

Used in: Datacentres, cloud servers, enterprise virtualization.



2. Type 2 – Hosted VMM

- Runs on top of a host operating system (like Windows or Linux).
- Easier to install but slightly less efficient.
- The host OS provides hardware access to the hypervisor.

Examples:

- Oracle VirtualBox
- VMware Workstation / Fusion
- Parallels Desktop (Mac)

Used in: Personal computers, testing, and development environments.

Summary Table

Aspect	Type 1 VMM	Type 2 VMM
Runs on	Hardware directly	On host OS
Performance	High	Moderate
Security	More secure	Less secure
Example	VMware ESXi, Hyper-V, KVM	VirtualBox, VMware Workstation
Use Case	Servers, data centres	Personal or testing systems

In Short:

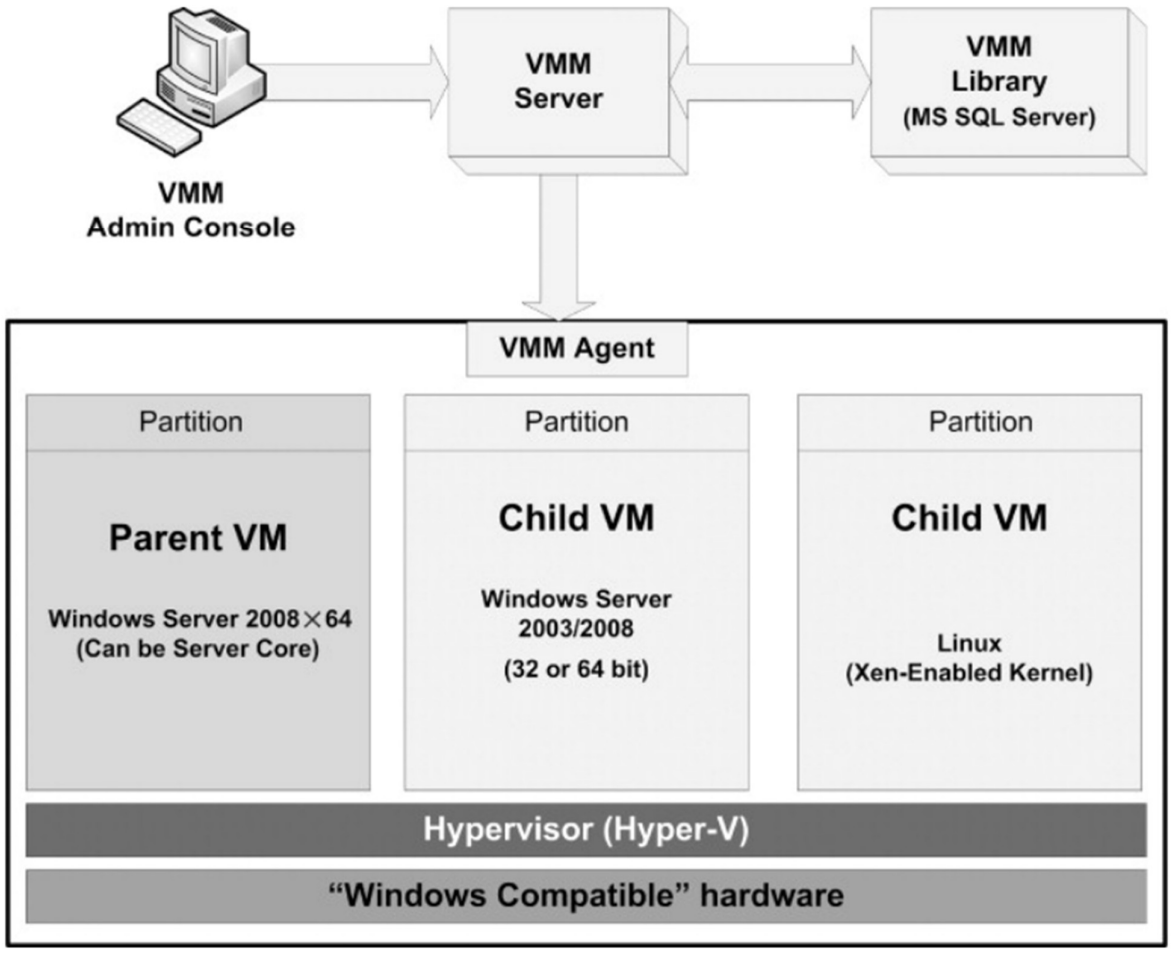
A VMM (Virtual Machine Monitor) must ensure fidelity, performance, isolation, and security, while managing resources and devices efficiently.
It can be implemented as a Type 1 (bare - metal) or Type 2 (hosted) hypervisor

Virtual Machine Monitor (VMM) design requires a balance between providing an identical environment to the physical hardware, ensuring minimal performance loss, and maintaining full control over system resources. Providers range from industry giants in cloud computing and data centres to specialized open-source solutions.

VMM design requirements

The core requirements for VMM design were first established in seminal virtualization research and continue to guide the development of modern hypervisors.

Core design principles



- **Fidelity:** The VMM must provide a virtual environment that is "essentially identical" to the original physical machine. Any program running in a VM should function as it would on bare metal, with timing differences being the only acceptable variation.
- **Performance:** A high-performance VMM must allow programs to run at near-native speed. To achieve this, the VMM executes the vast majority of instructions directly on the physical processor, with only privileged instructions requiring software intervention.
- **Isolation and control:** The VMM must have complete control over system resources and isolate virtual machines from one another. This includes:
  - **Resource allocation:** The VMM is responsible for allocating hardware resources to each VM.
  - **Access control:** VMs cannot access resources that have not been explicitly allocated to them.
  - **Secure re-allocation:** The VMM can reclaim allocated resources under certain circumstances.

Specialized design considerations

- **Hardware support:** Many modern processors include virtualization extensions (e.g., Intel VT-x and AMD-V). VMMs designed for these processors use hardware-assisted virtualization to efficiently trap and handle privileged instructions.
- **Embedded systems:** VMMs for embedded systems must consider performance constraints due to the typically low-power hardware. Designers must prioritize specific performance metrics to meet the requirements of real-world embedded applications.
- **Security services:** Some VMM designs include additional privilege levels to secure the virtualization environment. For instance, a Trusted Platform Module (TPM) can be virtualized to provide each VM with its own secure virtual TPM.
- **Cloud-specific features:** VMMs designed for large-scale cloud providers focus on efficiency, multi-tenancy, and advanced networking features like network virtualization (NV).

VMM providers

The landscape of VMM providers is diverse, with solutions for enterprise data centers, cloud computing, and desktop use.

Major cloud and data center providers

- **Amazon Web Services (AWS):** Uses its custom Nitro System, which combines dedicated hardware with a lightweight hypervisor, for its high-performance virtual machines (EC2). It previously relied on the open-source Xen hypervisor.
- **Microsoft:** Offers Hyper-V as a built-in, bare-metal hypervisor for Windows Server and uses a hardened version of it for Azure. It also provides System Center Virtual Machine Manager (VMM) for centralized management of virtual environments.
- **Google Cloud Platform (GCP):** Employs the open-source Kernel-based Virtual Machine (KVM), which is integrated into the Linux kernel, to run its virtual machine instances.
- **VMware:** A long-standing market leader, its products include the enterprise bare-metal hypervisor ESXi and the management platform vSphere.

Enterprise and open-source providers

- **Red Hat:** Offers Red Hat Virtualization, an enterprise virtualization platform based on the open-source KVM hypervisor.
- **Nutanix:** Features a hyper-converged infrastructure solution with its own built-in hypervisor, Acropolis Hypervisor (AHV).
- **Oracle:** Provides Oracle VM, a virtualization solution for data centers, along with the open-source desktop virtualization software VirtualBox.
- **Citrix:** Specializes in desktop and application virtualization, offering Citrix Hypervisor (based on Xen).
- **Proxmox VE:** An open-source virtualization management platform that supports both KVM-based virtual machines and Linux containers (LXC).
- **Xen:** An open-source bare-metal hypervisor popular with many cloud and hosting providers.

Choosing a VMM solution

The right VMM depends on the specific use case:

- For enterprise data centers, VMware vSphere and Microsoft Hyper-V are dominant choices, offering robust feature sets and integration with management suites.
- For cloud deployments, the hypervisors are often tied to the specific provider (AWS Nitro, Azure Hyper-V, GCP KVM), with performance, cost, and ecosystem integration being key factors.
- For development and desktop use, hosted VMMs like Oracle VirtualBox and VMware Workstation provide a user-friendly way to run multiple guest operating systems.
- For those prioritizing open source and flexibility, solutions based on KVM (like Red Hat Virtualization and Proximo VE) or the Xen hypervisor are excellent options.

VIRTUALIZATION SUPPORT AT THE OS LEVEL

🧩 Virtualization Support at the OS Level

💡 **Definition:**

OS-level virtualization is a technique where the operating system kernel allows multiple isolated user-space instances — called containers — to run on a single host machine.Each container behaves like a separate virtual machine, but all of them share the same OS kernel.

⚙️ **How It Works**

1. The host operating system runs a single kernel.
2. The kernel uses namespaces and control groups (cgroups) to:
    - Isolate processes and file systems.
    - Control resource usage (CPU, memory, disk, network).
3. Each container gets:
    - Its own process tree
    - Network interface
    - File system view
    - User IDs and permissions

In simple terms:
OS-level virtualization = Multiple lightweight "mini OS environments" running on one kernel.

Key Features

Feature	Description
Single Kernel	All containers share the same OS kernel (e.g., Linux kernel).
Isolation	Each container runs in a private environment, isolated from others.
Efficiency	Very fast and lightweight compared to full VMs.
Portability	Containers can easily be moved between systems.
Resource Control	The kernel limits resource usage using **cgroups**.

Technologies That Support OS-Level Virtualization

Technology	Description / Example
Docker	Most popular containerization platform; builds and runs containers.
LXC (Linux Containers)	Early container technology built directly into Linux.
OpenVZ	Container-based virtualization for Linux servers.
Kubernetes	Orchestrates and manages multiple containers across systems.
Solaris Zones / Containers	OS-level virtualization on Solaris OS.
FreeBSD Jails	OS-level isolation in FreeBSD Unix.

Advantages of OS-Level Virtualization

1. Lightweight:
No separate OS for each VM — containers share one kernel.

2. Fast Startup:
Containers start in seconds compared to VMs that take minutes.

3. Efficient Resource Usage:
Less CPU and memory overhead.

4. Easy Deployment & Scaling:
You can run many containers on a single host.

5. Consistency Across Environments:
Same container can run on any system with the same OS kernel.

Limitations

1. Same OS Requirement:
All containers must use the same OS kernel (e.g., only Linux-based containers on Linux host).

2. Less Isolation than VMs:
If the kernel is compromised, all containers may be affected.

❌ 3. Limited Flexibility:

Can't run a different OS inside a container (e.g., can't run Windows on Linux kernel).

🔍 Comparison: OS-Level Virtualization vs Hardware Virtualization

Feature	OS-Level Virtualization	Hardware Virtualization
Kernel	Shared single kernel	Separate OS kernel for each VM
Performance	High (low overhead)	Lower (more resource heavy)
Isolation	Moderate	Strong
Boot Time	Seconds	Minutes
Example	Docker, LXC	VMware, Hyper-V, KVM

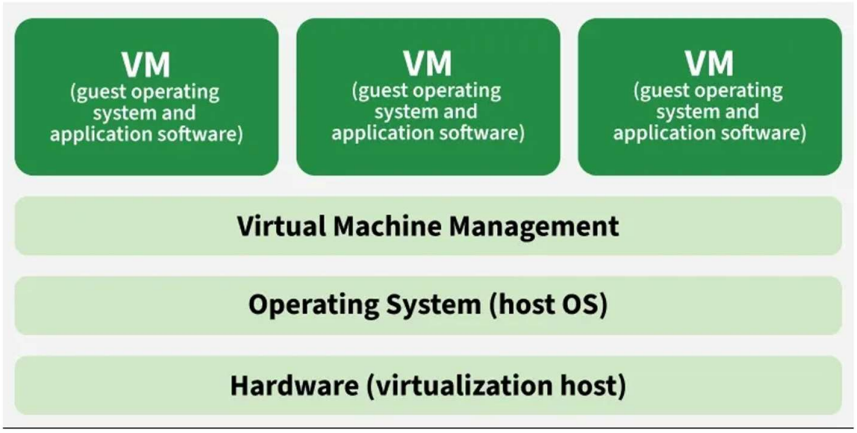
💬 Example in Real Use

- **Docker Container Example:**
  - You can run a web server, database, and application each in separate containers on one Linux host.
  - All share the same kernel but are isolated — you can start, stop, or update one container without affecting others.

✅ Summary

Aspect	Details
Concept	Multiple isolated containers running on one OS kernel
Kernel	Shared among all containers
Isolation Method	Namespaces + cgroups
Performance	Very efficient and lightweight
Examples	Docker, LXC, OpenVZ
Limitation	All containers must use the same OS kernel

Operating System-based Virtualization is also known as Containerization. It allows multiple isolated user-space instances called containers to run on a single operating system (OS) kernel.

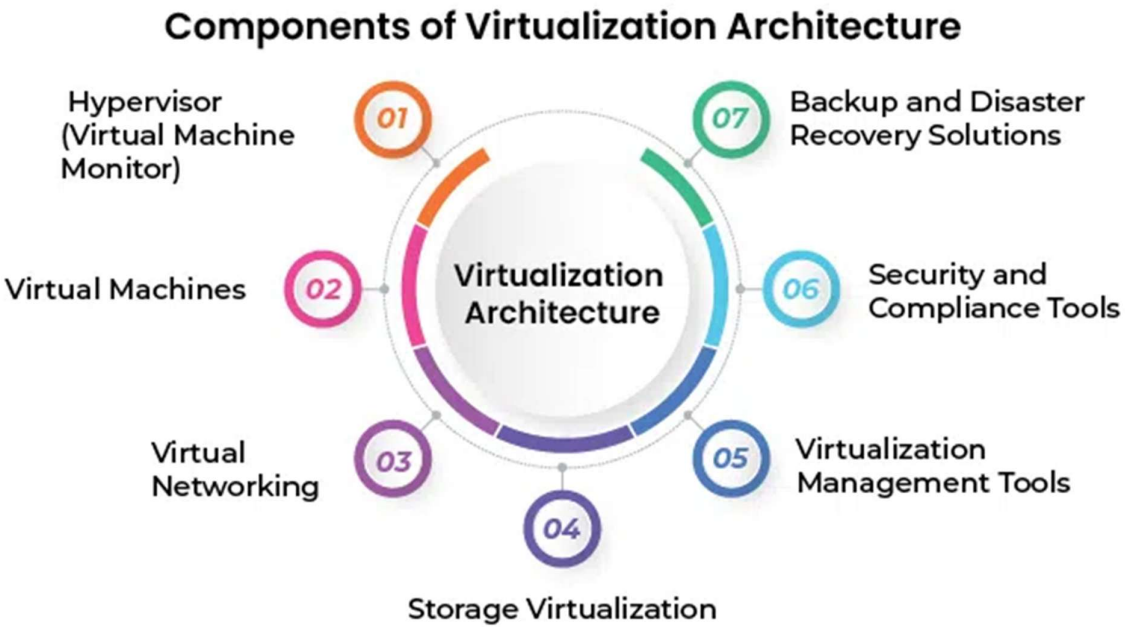


OS Based Virtualization

Note: Unlike traditional virtualization, where each VM requires its own OS, OS-based virtualization allows the sharing of the same OS while providing separate environments for running applications.

Traditional Virtualization Architecture (Using VMs)

- Each Virtual Machine (VM) operates as an isolated environment, having its own OS.
- This results in higher resource consumption (CPU, memory, storage).
- The Virtual Machine Management layer is responsible



.**Virtualization Architecture**

Managing and overseeing virtual machines, Proper resource allocation (CPU, memory, storage) & Maintaining isolation between VMs.

- The Host Operating System (OS) runs above the physical hardware and provides the environment for the hypervisor.
- The Hypervisor: Manages VMs & Allocates resources from the physical hardware to the VMs.
- The Hardware (Virtualization Host) is the physical machine that provides necessary CPU, memory, storage and I/O to run the hypervisor and VMs.
- Multiple VMs run simultaneously on the same physical hardware.

How OS-Based Virtualization Works

OS-Based Virtualization works as follows:

- The host OS kernel is shared among all containers, unlike full virtualization (e.g., VMs) where each VM has its own kernel.
- The kernel enforces isolation between containers using namespaces (for process, network, filesystem isolation) and cgroups (control groups) for resource allocation (CPU, memory, disk I/O, network).
- cgroups limit and prioritize resource usage (CPU, memory, disk, network) per container.
- The kernel ensures that a container cannot exceed its allocated resources (unless explicitly allowed).

- Namespaces prevent processes in one container from seeing or interfering with processes in another.
- Programs inside a container cannot access resources outside unless explicitly granted (e.g., mounted volumes, network ports).
- The overhead comes from kernel-level isolation mechanisms (namespaces, cgroups), but it's minimal compared to full virtualization.

Operating System Based Services

Some major operating system based services are mentioned below:

- **Backup and Recovery:** Host operating systems can be utilized to back up and restore virtual machines. Backup software tools can be used to ensure data safety and system recovery.
- **Security Management:** Host operating systems help manage the security of virtual machines. This includes configuring firewalls, installing antivirus software and applying other essential security settings.
- **Integration with Directory Services:** Host operating systems can be integrated with directory services like Active Directory, enabling centralized management of users and groups.

Operating System Based Operations

Various major operations of Operating System Based Virtualization are described below:

1. Hardware capabilities can be employed such as the network connection and CPU.
2. Connected peripherals with which Host OS can interact such as a webcam, printer, keyboard or scanners.
3. Host OS can be used to read or write data in files, folders and network shares.

Features of OS- Based Virtualization

- **Resource isolation:** Operating system based virtualization provides a high level of resource isolation which allows each container to have its own set of resources, including CPU, memory and I/O bandwidth.
- **Lightweight:** Containers are lighter compared to traditional virtual machines as they share the same host operating system. This results in faster startup and lower resource usage.
- **Portability:** Containers are highly portable. They can be easily moved from one environment to another without the need to modify the underlying application.
- **Scalability:** Containers can be easily scaled up or down based on the application requirements. This makes it easier for applications to be highly responsive to changes in demand.
- **Security:** Containers provide a high level of security by isolating the containerized application from the host operating system and other containers running on the same system.
- **Reduced Overhead:** Containers incur less overhead than traditional virtual machines as they do not need to emulate a full hardware environment.
- **Easy Management:** Containers are easy to manage as they can be started, stopped and monitored using simple commands.

Pros of OS-Based Virtualization

- **Resource Efficiency:** Operating system based virtualization allows for greater resource efficiency as containers do not need to emulate a complete hardware environment, which reduces resource overhead.

- **High Scalability:** Containers can be quickly and easily scaled up or down depending on the demand, which makes it easy to respond to changes in the workload**.**
- **Easy Management:** Containers are easy to manage as they can be managed through simple commands, which makes it easy to deploy and maintain large numbers of containers.
- **Reduced Costs:** Operating system based virtualization can significantly reduce costs, as it requires fewer resources and infrastructure than traditional virtual machines.
- **Faster Deployment:** Containers can be deployed quickly, reducing the time required to launch new applications or update existing ones**.**
- **Portability:** Containers are highly portable, making it easy to move them from one environment to another without requiring changes to the underlying application.

Cons of OS-Based Virtualization

- **Security:** Operating system based virtualization may pose security risks as containers share the same host operating system, which means that a security breach in one container could potentially affect all other containers running on the same system.
- **Limited Isolation:** Containers may not provide complete isolation between applications, which can lead to performance degradation or resource contention.
- Complexity: Operating system based virtualization can be complex to set up and manage, requiring specialized skills and knowledge.
- **Dependency Issues:** Containers may have dependency issues with other containers or the host operating system, which can lead to compatibility issues and hinder deployment.
- **Limited Hardware Access:** Containers may have limited access to hardware resources which can limit their ability to perform certain tasks or applications that require direct hardware access.

MIDDLEWARE SUPPORT FOR VIRTUALIZATION

What is middleware?

Middleware is software that different applications use to communicate with each other. It provides functionality to connect applications intelligently and efficiently so that you can innovate faster. Middleware acts as a bridge between diverse technologies, tools, and databases so that you can integrate them seamlessly into a single system. The single system then provides a unified service to its users. For example, a Windows frontend application sends and receives data from a Linux backend server, but the application users are unaware of the difference.

Why is middleware important?

Middleware started as a bridge between new applications and legacy systems before it gained popularity in the 1980s. Developers initially used it to integrate new programs with earlier systems without rewriting the earlier code. Middleware has become an important communication and data management tool in distributed systems.

Developers use middleware to support application development and simplify design processes. This leaves them free to focus on business logic and features instead of connectivity between different software components. Without middleware, developers would have to build a data exchange module

for each software component that connects to the application. This is challenging because modern applications consist of multiple microservices or small software components that talk to each other.

What are the use cases of middleware?

The following are the more common use cases of middleware:

Game development

Game developers use middleware as a game engine. For a game to work, the software must communicate with various image, audio, and video servers along with communication systems. The game engine facilitates this communication and makes game development more efficient.

Electronics

Electronics engineers use middleware to integrate various types of sensors with their controllers. The middleware layer allows sensors to communicate with the controller through a common messaging framework.

Software development

Software developers use middleware to integrate different software components into other applications. Middleware offers a standard Application Programming Interface (API) to manage the required input and output of data from the component. The internal linking with the component is hidden from the user. Developers use the APIs to request the services that they need from the software components.

Data transmission

Software applications use middleware to send and receive data streams reliably. Data streams are a high-speed transmission of continuous data. They are important for reliable video and audio streaming.

Distributed applications

Distributed applications are software programs that run on different computers on a network. They usually consist of frontend and backend applications. Frontend applications are software you use on a computer or mobile device, such as a social media app. By contrast, backend applications are software programs that handle data processing, business logic, and resource management tasks. Middleware communicates between the frontend and backend applications, so the distributed application works smoothly.

What is middleware architecture?

Middleware software architecture consists of several components that communicate to create a data pipeline. The data moves from one connecting application to the other through the middleware. The middleware processes the data for compatibility. The following are common components of middleware software:

Management console

The management console provides software developers with an overview of the middleware system's activities, software rules, and configurations.

Client interface

The client interface is the outer part of the middleware software that communicates with the applications. Developers use functions provided by the client interface to interact with other applications, databases, or other microservices.

Middleware internal interface

The middleware internal interface acts as the software glue that binds the various components together. The middleware components use the internal interface to function cohesively with their own protocol.

Platform interface

The middleware interface ensures that the middleware program is compatible with various platforms. It contains software components that work with different types of operating systems.

Contract manager

The contract manager defines the rules for data exchange in the middleware system. It also ensures that applications abide by the rules when sending data with the middleware. It sends an alert, or an exception, to the application when it breaches specific rules. For example, the contract manager will return an exception if the application sends a number when a word is expected.

Session manager

The session manager sets up a secure communication channel between applications and the middleware. It ensures that communication flows seamlessly and stores data activity records for reporting.

Database manager

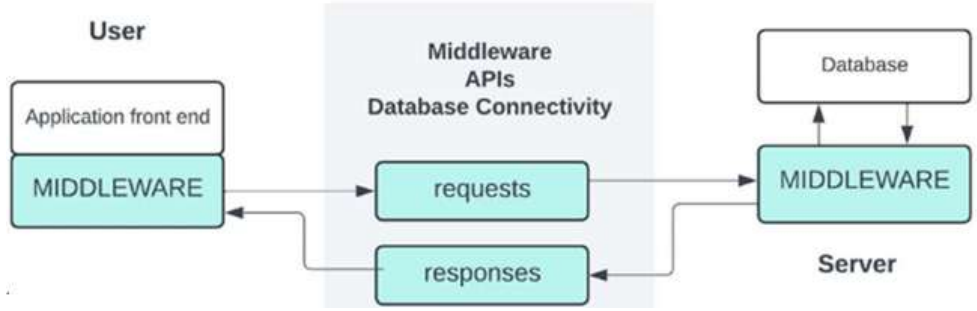
Some types of middleware also include a database manager. The database manager is responsible for integrating with different database types, as required.

Runtime monitor

The runtime monitor provides continuous monitoring of data movements in the middleware. It detects and reports unusual activities to developers.

How does middleware work?

Middleware abstracts the underlying communication process between components. This means that the frontend application communicates only to the middleware and does not have to learn the language of other backend software components.



Messaging framework

A messaging framework facilitates the exchange of data between frontend and backend applications. Common frameworks include the following:

- JavaScript Object Notation (JSON)
- Representational State Transfer (REST API)
- Extensible Markup Language (XML)

- Web services
- Simple Object Access Protocol (SOAP)

The messaging frameworks provide a common communication interface for applications in different operating platforms and languages. Applications write and read data in a standardized format provided by the messaging framework.

Example of middleware

For example, a web server is middleware that connects websites to the backend database. When you submit a form on a website, your computer sends the request in XML or JSON to the web server. Then, the web server runs the business logic based on the request, retrieves information from databases, or communicates to other microservices using different protocols.

Other middleware functions

Besides being an intermediary between software applications, middleware programs also do the following:

- Provide a secure communication channel between distributed applications so that websites send sensitive information safely to backend applications.
- Manage traffic flow and avoid overwhelming a particular application or file server.
- Automate and customize responses to the request. For example, the middleware sorts and filters the results before sending them to the frontend application.

What is platform middleware?

Platform middleware supports application development by providing a system of managed tools and resources. Developers use platform middleware to share or transfer resources between applications. following are some examples of platform middleware resources:

Runtime environments

A runtime environment is like a small operating system that allows a software program to run. For example, Java applications must run in the Java Runtime Environment. Developers can use AWS Lambda to set up a runtime environment for any programming language.

Web servers

A web server is a computer program that receives, processes, and responds to requests from websites. Web developers use Amazon Lightsail to host and manage web servers for simple applications.

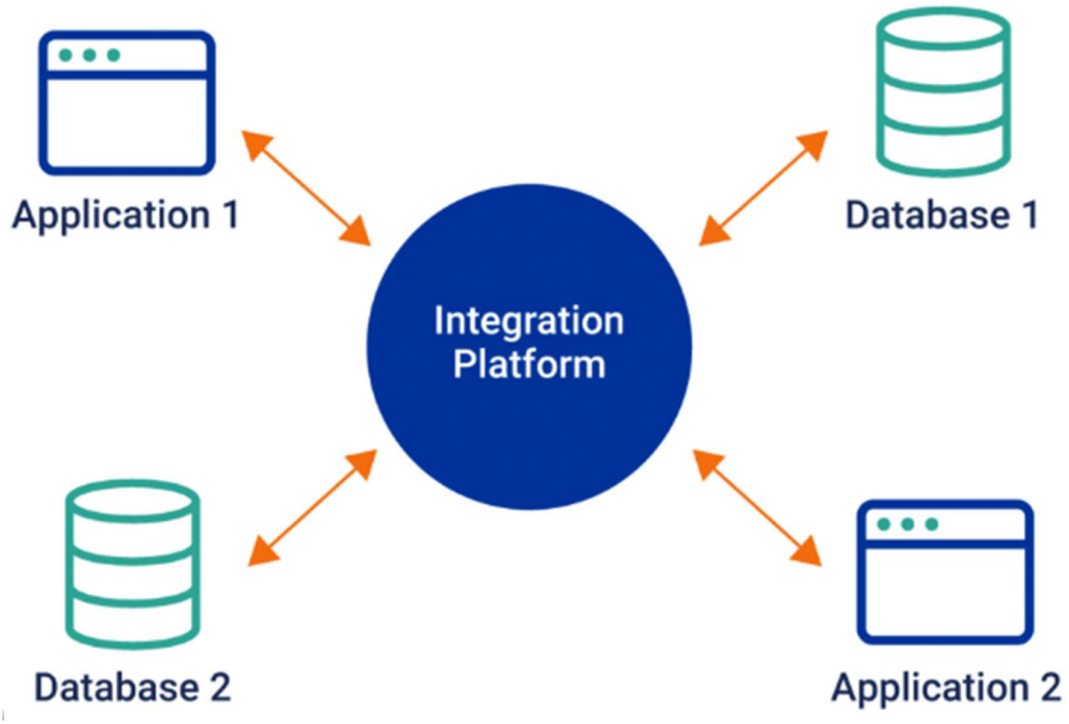
Content management systems

The content management system is software that creates, modifies, stores, and publishes digital information. For example, WordPress is an open-source content management system for building websites.

Containers

A container is a ready-to-deploy bundle of the application codes and necessary resources. Developers use Amazon Elastic Container Service (Amazon ECS) to deploy, manage, and scale containerized applications.

What is middleware in cloud computing?



Cloud computing involves building and deploying cloud-native applications across different infrastructures. Developers use middleware to access cloud resources without being overwhelmed by the complexity of managing the infrastructures. Developers deploy cloud applications in containers on a scalable cloud-based hosting such as Amazon Elastic Compute Cloud (Amazon EC2).

🧩 1. What is Middleware in Virtualization?

💡 Definition:

Middleware is software that provides services, communication, and management between:

- The operating systems / virtual machines and
- The applications or users that run on top of them.

In a virtualized system, middleware helps manage multiple virtual machines, coordinate resources, and enable distributed applications to run efficiently.

⚙️ 2. Role of Middleware in Virtualization

Middleware acts as a virtualization enabler by providing:

- **Abstraction** → hides hardware/OS complexities.
- **Integration** → allows different virtual systems to communicate.
- **Resource Management** → controls how virtual resources are used.
- **Automation** → manages creation, migration, and monitoring of VMs.

🧱 3. Middleware Functions in a Virtualized Environment

Function	Description
a. Resource Abstraction	Hides the complexity of hardware, networks, and storage.
b. Resource Allocation	Dynamically allocates CPU, memory, and storage among VMs.
c. Communication Support	Provides APIs, message queues, or brokers for inter-VM communication.
d. Load Balancing	Distributes workload across multiple virtual machines or servers.
e. Fault Tolerance	Detects VM failures and restarts them automatically.
f. Security & Access Control	Manages authentication, authorization, and isolation between VMs.
g. Monitoring & Management	Tracks resource usage, health, and performance of VMs.

🧰 4. Types of Middleware Supporting Virtualization

Middleware in virtualization can be classified into different categories depending on its purpose:

🖥️ A. System Middleware

- Manages the infrastructure layer of virtualization.
- Provides interfaces to the hypervisor and manages hardware-level virtualization.

Examples:

- VMware vCenter
- Microsoft System Center Virtual Machine Manager (SCVMM)
- OpenStack (for managing cloud-based VMs)

Responsibilities:

- VM provisioning, migration (live migration)
- Resource scheduling
- Backup and recovery

🌐 B. Communication Middleware

- Enables communication between distributed applications or VMs.

Examples:

- **Message-Oriented Middleware (MOM)** like:
  - RabbitMQ
  - Kafka
  - ActiveMQ
- CORBA, RMI, Web Services (SOAP/REST APIs)

Responsibilities:

- Data exchange across virtualized systems
- Event-driven or message-based communication
- Service interoperability

C. Database Middleware

- Provides virtualized access to databases regardless of where they are hosted (physical or virtual servers).

Examples:

- ODBC, JDBC, Oracle Middleware
- Hibernate (object-relational mapping layer)

Responsibilities:

- Data consistency across multiple virtual machines
- Abstracts database details from applications

D. Cloud / Virtualization Management Middleware

- Manages virtualization across data centers and cloud platforms.
- Offers APIs for provisioning, scaling, and monitoring VMs.

Examples:

- VMware vCloud Director
- OpenStack Nova
- CloudStack
- Kubernetes (for container orchestration)

Responsibilities:

- VM lifecycle management (create, start, stop, delete)
- Orchestration and automation
- Resource pooling and usage tracking

E. Application Middleware

- Supports applications running inside virtual environments.

Examples:

- Java EE Application Server (GlassFish, JBoss)
- .NET Framework
- Node.js, Spring Boot

Responsibilities:

- Runs virtualized enterprise applications
- Provides APIs, transaction management, and web services

5. Examples of Middleware in Virtualization

Middleware Type	Example Tools / Technologies	Purpose
System Management	VMware vCenter, OpenStack	VM management and orchestration
Communication	RabbitMQ, Kafka	Message exchange between VMs
Database	ODBC, JDBC	Database access abstraction
Application	JBoss, WebSphere	App deployment and services
Cloud Orchestration	Kubernetes, CloudStack	Container and VM orchestration

⚖️ 6. Benefits of Middleware Support in Virtualization

✅ **1. Simplified Management:**
Centralized control over virtualized resources.

✅ **2. Interoperability:**
Different systems and applications can work together.

✅ **3. Scalability:**
Easily add or remove virtual machines or containers.

✅ **4. Resource Optimization:**
Efficient utilization of CPU, memory, and storage.

✅ **5. High Availability & Reliability:**
Automatic failover, load balancing, and recovery.

✅ **6. Platform Independence:**
Applications can run on any virtualized or cloud platform.

⚠️ 7. Challenges

❌ **Complex Setup:** Requires configuration of multiple layers.
❌ **Security Risks:** Misconfigured middleware can expose virtual systems.
❌ **Performance Overhead:** Some middleware adds latency.
❌ **Integration Issues:** Compatibility between middleware versions.

🗄️ 8. Summary Table

Aspect	Description
Purpose	Bridge between virtual machines and applications
Core Functions	Resource management, communication, monitoring, automation
Examples	VMware vCenter, OpenStack, Kubernetes, RabbitMQ
Benefits	Simplified control, scalability, interoperability
Used In	Cloud computing, virtualization platforms, enterprise data centers

✅ **In Short:**
Middleware in virtualization provides the software layer that manages, connects, and optimizes virtualized environments — ensuring applications run smoothly over shared resources.

VIRTUALIZATION STRUCTURES/TOOLS AND MECHANISMS

In general, there are three typical classes of VM architecture. Showed 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,

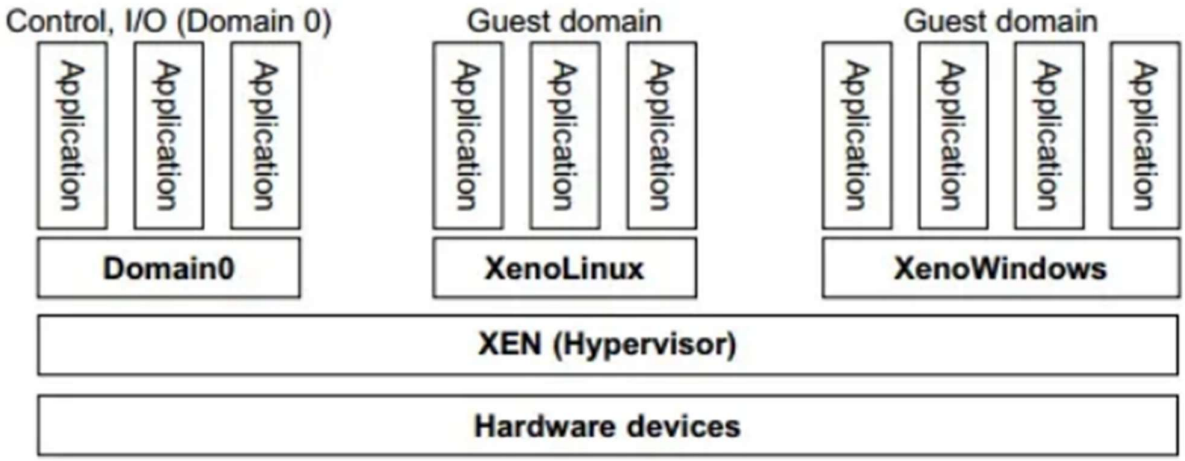
there are several classes of VM architectures, namely the hypervisor architecture, para-virtualization, and host-based virtualization. The hypervisor is also known as the VMM (Virtual Machine Monitor).
They both perform the same virtualization operations.

1. 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 hypercells for the guest OSes and applications. Depending on the functional-ity, 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 functions, including those of the device drivers. Therefore, the size of the hypervisor code of a micro-kernel hyper-visor 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.

1.1 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 3.5. 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. Several vendors are in the process of developing commercial Xen hypervisors, among them are Citrix XenServer 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 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).For example, Xen is based on Linux and its security level is C2. 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.

Traditionally, a machine's lifetime can be envisioned as a straight line where the current state of the machine is a point that progresses monotonically as the software executes. During this time, configuration changes are made, software is installed, and patches are applied. In such an environment, the VM state is akin to a tree: At any point, execution can go into N different branches where multiple instances of a VM can exist at any point in this tree at any given time. VMs are allowed to roll back to previous states in their execution (e.g., to fix configuration errors) or rerun from the same point many times (e.g., as a means of distributing dynamic content or circulating a "live" system image).

2. 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, nonvirtualizable 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 virtuali-zation software layer is built between the host OS and guest OS. These two classes of VM architec-ture are introduced next.

2.1 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.

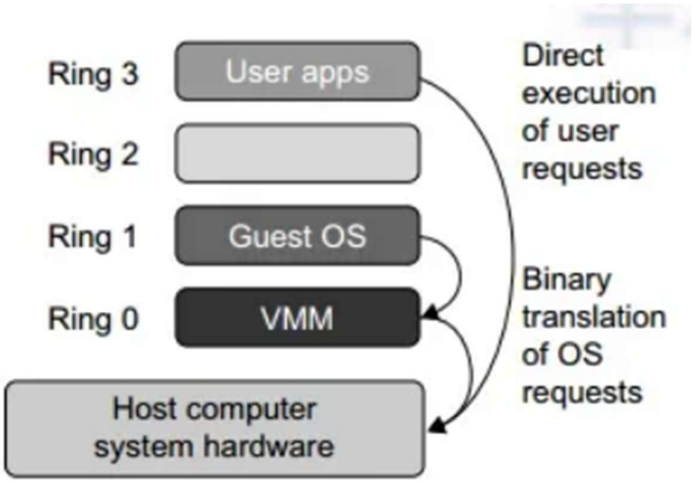
2.2 Binary Translation of Guest OS Requests Using a VMM

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 behaviour-sensitive instructions. When these instructions are identified, they are trapped into the VMM, which emulates the behaviour 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. Consequently, the guest OS is unaware that it is being virtualized.

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. At the time of this writing, the performance of full virtualization on the x86 architecture is typically 80 percent to 97 percent that of the host machine.

2.3 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 enumerated next. 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.



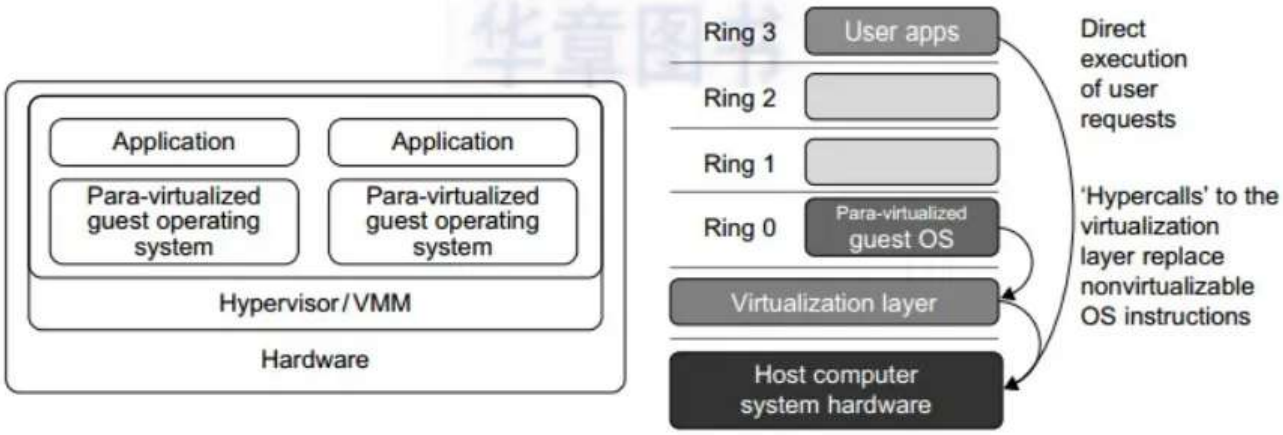
Second, the host-based approach appeals to many host machine configurations. Compared to the hypervisor/VMM architecture, the performance of the host-based architecture may also be low. When an application requests hardware access, it involves four layers of mapping which downgrades performance significantly. 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.

3. 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 soft-ware stack. However, para-virtualization attempts to reduce the virtualization overhead, and thus improve performance by modifying only the guest OS kernel. Illustrates the concept of a paravirtualized VM architecture. The guest operating systems are para virtualized. They are assisted by an intelligent compiler to replace the nonvirtualizable OS instructions by hyper calls as illustrated. 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. The best example of para-virtualization is the KVM to be described below.

3.1 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 definition, the virtualization layer should also be installed at Ring 0. Different instructions at Ring 0 may cause some problems.We show that para-virtualization replaces nonvirtualizable 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. Compared with full virtualization, para-virtualization is relatively easy and more practical. 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.

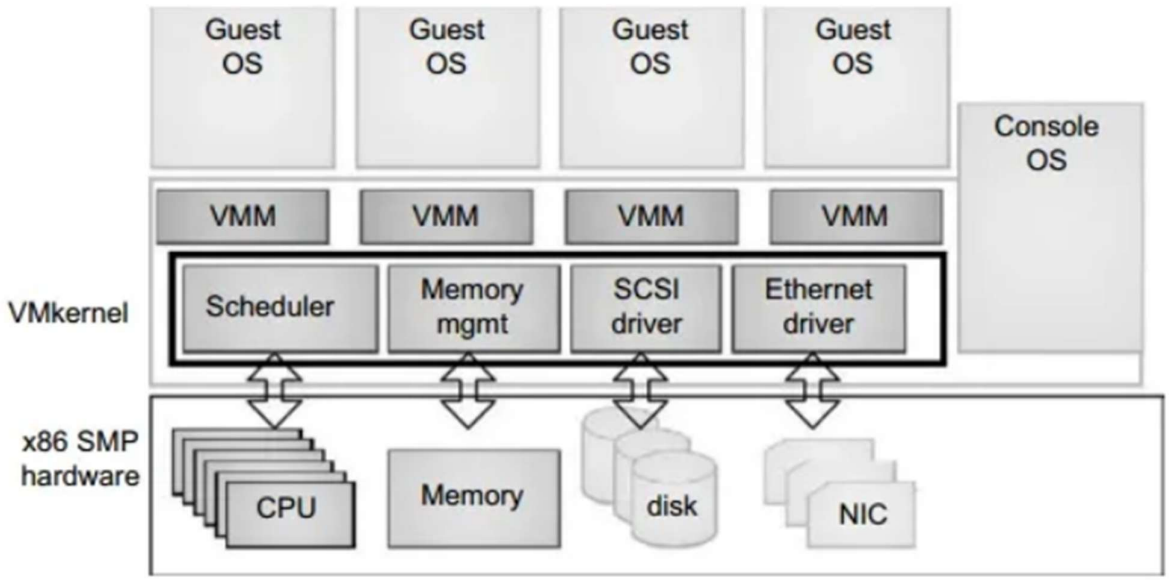
3.2 KVM (Kernel-Based VM)

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.

3.3 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.

The guest OS running in a guest domain may run at Ring 1 instead of at Ring 0. 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 behaviour of the original guest OS. On an UNIX system, a system call involves an interrupt or service routine. The hyper calls apply a dedicated service routine in Xen.



VMware ESX Server for Para-Virtualization

VMware pioneered the software market for virtualization. The company has developed virtualization tools for desktop systems and servers as well as virtual infrastructure for large data centres. ESX is a VMM or a hypervisor for bare metal x86 symmetric multiprocessing (SMP) servers. It accesses hardware resources such as I/O directly and has complete resource management control. An ESX-

enabled server consists of four components: a virtualization layer, a resource manager, hardware interface components, and a service console. To improve performance, the ESX server employs a para-virtualization architecture in which the VM kernel interacts directly with the hardware without involving the host OS.

The VMM layer virtualizes the physical hardware resources such as CPU, memory, network and disk controllers, and human interface devices. Every VM has its own set of virtual hardware resources. The resource manager allocates CPU, memory disk, and network bandwidth and maps them to the virtual hardware resource set of each VM created. Hardware interface components are the device drivers and the

VMware ESX Server File System. The service console is responsible for booting the system, initiating the execution of the VMM and resource manager, and relinquishing control to those layers. It also facilitates the process for system administrators.

🧩 1. Virtualization Structure

💡 Definition:

A virtualization structure defines how the virtualization environment is organized — from hardware to virtual machines.

It shows the relationship between:

- Physical hardware
- Virtual Machine Monitor (VMM) / Hypervisor
- Guest Operating Systems
- Applications

🔍 Explanation of Layers

Layer	Description
Hardware Layer	The actual physical resources (CPU, RAM, disk, etc.)
Host OS	Runs the hypervisor in Type 2 virtualization
VMM / Hypervisor	Controls and manages multiple virtual machines
Guest OS	OS installed inside each virtual machine
Applications	Software running inside virtual machines

🧱 Types of Virtualization Structures

1. **Type 1 – Bare-Metal Architecture**
   - VMM runs **directly on hardware** (no host OS).
   - Used in servers and data centres.
     Examples: VMware ESXi, Microsoft Hyper-V, KVM
2. **Type 2 – Hosted Architecture**
   - VMM runs **on top of a host OS**.
   - Used in personal computers and testing.
     Examples: VirtualBox, VMware Workstation

🧰 2. Virtualization Tools

Virtualization tools are the **software platforms** used to implement and manage virtualization.

⚙️ A. Hardware Virtualization Tools (Full/Paravirtualization)

Tool	Type	Description
VMware ESXi	Type 1	Enterprise-grade hypervisor for servers
KVM (Kernel-based Virtual Machine)	Type 1	Linux-based virtualization integrated into kernel
Microsoft Hyper-V	Type 1	Used in Windows Server environments
Xen Hypervisor	Type 1 (Paravirtualization)	Open-source hypervisor supporting Linux and Windows
Oracle VirtualBox	Type 2	Free desktop virtualization software

💡 B. OS-Level Virtualization Tools (Containers)

Tool	Description
Docker	Creates isolated containers for applications
LXC (Linux Containers)	Lightweight system-level virtualization
OpenVZ	OS-level virtualization for Linux servers
Kubernetes	Manages and orchestrates multiple containers

💾 C. Storage and Network Virtualization Tools

Tool	Type	Purpose
VMware vSAN	Storage	Virtual storage management
Open v Switch	Network	Virtual network switching
Cisco ACI / NSX	Network	Software-defined networking (SDN)
Ceph	Storage	Distributed virtual storage system

⚙️ 3. Virtualization Mechanisms

These are the core techniques used by the VMM (hypervisor) to create and manage virtual machines.

🧠 A. Trap and Emulate

- Certain privileged CPU instructions (like I/O operations) are trapped by the VMM.
- The VMM emulates their behaviour safely.
  - ✅ **Used In:** Full Virtualization
  - 📘 **Example:** VMware, VirtualBox

B. Binary Translation

- The VMM modifies (translates) sensitive instructions of the guest OS into safe instructions before execution.
  - ✅ **Used In:** Early VMware systems
  - ⚠️ **Disadvantage:** Slower performance due to translation overhead.

C. Hardware-Assisted Virtualization

- Modern CPUs (Intel VT-x, AMD-V) provide **special hardware instructions** to support virtualization directly.
- Reduces overhead and improves performance.
  - ✅ **Used In:** VMware ESXi, Hyper-V, KVM

D. Paravirtualization

- The guest OS is modified to interact directly with the VMM using special APIs instead of trapping instructions.
  - ✅ **Used In:** Xen Hypervisor, KVM (with Virto drivers)
  - ⚡ **Advantage:** Faster than full virtualization.

E. Memory Virtualization

- Each VM believes it has its own physical memory.
- The VMM maps guest virtual memory to real physical memory.
  Techniques: Shadow Page Tables, Extended Page Tables (EPT).

F. I/O Virtualization

- Virtual devices (like disks, NICs) are provided to VMs by the VMM.
- Achieved through device emulation or paravirtualized drivers (e.g., Virto).

G. OS-Level Virtualization

- Containers share the same kernel but run isolated applications.
  - ✅ **Used In:** Docker, LXC
  - ⚡ **Advantage:** Lightweight and efficient.

4. Summary Table

Aspect	Details	Examples
Structure	Layered organization from hardware to VM	Type 1, Type 2
Tools	Software for implementing virtualization	VMware, KVM, Docker
Mechanisms	Techniques used by VMM to virtualize	Trap & Emulate, Paravirtualization, Hardware Assist

5. CSE-Level Summary (for Exam)

Virtualization Structures, Tools, and Mechanisms describe how virtualization is organized, implemented, and managed.

- **Structure:** Defines the architecture (Type 1 & Type 2).
- **Tools:** Provide the software to create and manage VMs (VMware, KVM, Docker).
- **Mechanisms:** Explain how virtualization is achieved internally (Trap & Emulate, Paravirtualization, Hardware Assist).

In Short:

"Virtualization Structure" defines how it's built,
"Tools" define what is used, and
"Mechanisms" define how it works.

VIRTUALIZATION OF CPU

What is CPU Virtualization in Cloud Computing?

A single CPU can run numerous operating systems (OS) via CPU virtualization in cloud computing. This is possible by creating virtual machines (VMs) that share the physical resources of the CPU. Each Virtual Machine can't see or interact with each other's data or processes.

CPU virtualization is very important in cloud computing. It enables cloud providers to offer services like –

- Virtual private servers (VPSs)
- Cloud storage (EBS)
- Cloud computing platforms (AWS, Azure and Google Cloud)

Consider an example to understand CPU virtualization. Imagine we have a physical server with a single CPU. We want to run two different operating systems on this server, Windows & Linux. So it can easily be done by creating two Virtual Machines (VMs), one for Windows and one for Linux.

The virtualization software will create a virtual CPU for each VM. The virtualization software will create a virtual CPU for each VM. The virtual CPUs will execute on the physical CPU but separately. This means the Windows Virtual Machine cannot view or communicate with the Linux VM, and vice versa.

The virtualization software will also allocate memory and other resources to each VM. This guarantees each VM has enough resources to execute. CPU virtualization is made difficult but necessary for cloud computing.

How CPU Virtualization Works? in step by step process

Step 1: Creating Virtual Machines (VMs)

- Let's take an example you have a powerful computer with a CPU, memory, and other resources.

- To start CPU virtualization, you use special software called a hypervisor. This is like the conductor of a virtual orchestra.
- The hypervisor creates virtual machines (VMs) – these are like separate, isolated worlds within your computer.
- The "virtual" resources of each VM include CPU, memory, and storage. It's like having multiple minicomputers inside your main computer.

Step 2: Allocating Resources

- The hypervisor carefully divides the real CPU's processing power among the VMs. It's like giving each VM its own slice of the CPU pie.
- It also makes sure that each Virtual memory (VM) gets its share of memory, storage, and other resources.

Step 3: Isolation and Independence

Each VM operates in its own isolated environment. It can't see or interfere with what's happening in other VMs.

Step 4: Running Operating Systems and Apps

- Within each Virtual Machine, you can install & run different operating systems (like Windows, Linux) and applications.
- The VM thinks it's a real computer, even though it's sharing the actual computer's resources with other VMs.
-

Step 5: Managing Workloads

- The hypervisor acts as a smart manager, deciding when each VM gets to use the real CPU.
- It ensures that no VM takes up all the CPU time, making sure everyone gets their turn to work.

Step 6: Efficient Use of Resources

- Even though there's only one physical CPU, each VM believes it has its own dedicated CPU.
- The hypervisor cleverly switches between VMs so that all the tasks appear to be happening simultaneously.

Advantages of CPU Virtualization in Cloud Computing

1) **Efficient Resource Utilization**

CPU virtualization lets one powerful machine handle multiple tasks simultaneously. This maximizes the use of h/w resources and reduces wastage.

2) **Cost Savings**

By running multiple virtual machines on a single physical server, cloud providers save on hardware costs, energy consumption, and maintenance.

3) **Scalability**
CPU virtualization allows easy scaling, adding or removing virtual machines according to demand. This flexibility helps businesses adapt to changing needs as per requirements.
4) **Isolation and Security**
Each Virtual Machine (VM) is isolated from others, providing a layer of security. If one VM has a problem, it's less likely to affect others.
5) **Compatibility and Testing**
Different operating systems (OS) & applications can run on the same physical hardware (h/w), making it easier to test new software without affecting existing setups.

Disadvantages of CPU Virtualization in Cloud Computing:
1) **Overhead**
The virtualization layer adds some overhead, which means a small portion of CPU power is used to manage virtualization itself.
2) **Performance Variability**
Depending on the number of virtual machines and their demands, performance can vary. If one VM needs a lot of resources, others might experience slower performance.
3) **Complexity**
Handling multiple virtual machines and how they work together needs expertise. Creating and looking after virtualization systems can be complicated.
4) **Compatibility Challenges**
Some older software or hardware might not work well within virtualized environments. Compatibility issues can arise.
5) **Resource Sharing**
While CPU virtualization optimizes resource usage, if one VM suddenly requires a lot of resources, it might impact the performance of others.

Conclusion
In the world of cloud computing, CPU virtualization shines as a top-notch technology. CPU Virtualization in cloud computing like a cool trend that brings together great performance and work efficiency, all while saving money. Imagine using one computer for many tasks – that's what CPU virtualization does

🧠 Virtualization of CPU

(Explained in simple, CSE-standard format with examples and mechanisms)

💡 1. Definition

CPU Virtualization is the process of dividing and managing the physical CPU (processor) of a computer so that multiple virtual machines (VMs) can share and use it as if each VM had its own processor.
In simple words:
CPU virtualization allows many operating systems to run simultaneously on a single physical CPU, by allocating CPU time and resources among them.

⚙️ 2. Purpose of CPU Virtualization

Goal	Description
Resource Sharing	Share CPU among multiple virtual machines.
Isolation	Each VM acts as if it has a dedicated CPU.
Efficiency	Fully utilize CPU without idle time.
Security	Prevent one VM from affecting another.
Flexibility	Run multiple OSes or applications at the same time.

🔍 Explanation:

- Each VM is assigned one or more virtual CPUs (vCPUs).
- The VMM (Hypervisor) maps these vCPUs to the real physical CPU cores.
- The scheduler in the hypervisor decides when and how long each vCPU runs.

🧩 4. Mechanisms of CPU Virtualization

CPU virtualization is achieved through several mechanisms that help manage and control CPU instructions safely.

🧠 A. Trap-and-Emulate

- Certain privileged CPU instructions (like accessing hardware) are trapped by the VMM.
- The VMM emulates them safely before returning results to the VM.
  ✅ **Used in:** Full Virtualization (e.g., VMware Workstation).

⚙️ B. Binary Translation

- The VMM translates sensitive CPU instructions of the guest OS into safe instructions before execution.
  ✅ Used in: Older VMware products.
  ⚠️ **Disadvantage:** Adds performance overhead.

💡 C. Hardware-Assisted Virtualization

- Modern CPUs include special features to make virtualization faster and easier.
  - **Intel VT-x**
  - **AMD-V**
- These features allow **direct execution** of most guest instructions on the physical CPU, reducing traps and translation.
  ✅ **Used in:** KVM, Hyper-V, VMware ESXi

🧱 D. Paravirtualization

- The guest OS is **modified** to directly call the hypervisor for privileged operations instead of trapping.

✅ **Used in:** Xen, KVM (with Virto).

⚡ **Advantage:** Faster than full virtualization.

🧮 E. CPU Scheduling

- The VMM schedules the execution of virtual CPUs (vCPUs) on physical CPU cores.
- Common scheduling policies:
  - Round Robin
  - Fair Share
  - Priority-based
  - Credit-based Scheduler (used in Xen)

🔧 5. CPU Virtualization Techniques

Technique	Description	Example Tools / Technologies
Full Virtualization	Guest OS runs unmodified; VMM traps and emulates privileged instructions.	VMware Workstation, VirtualBox
Paravirtualization	Guest OS is aware of the hypervisor; communicates directly via APIs.	Xen
Hardware-Assisted Virtualization	Uses CPU features like Intel VT-x and AMD-V for efficient virtualization.	KVM, Hyper-V, VMware ESXi

🧰 6. CPU Virtualization Tools

Tool	Type	Description
VMware ESXi	Type 1	Uses hardware-assisted CPU virtualization.
KVM (Linux)	Type 1	Integrated into Linux kernel with Intel VT-x / AMD-V.
Xen Hypervisor	Type 1	Supports both para and full virtualization.
Microsoft Hyper-V	Type 1	Windows-based hypervisor with VT-x support.
VirtualBox	Type 2	CPU virtualization for desktop OSes.

🧮 7. Example of CPU Virtualization (Simplified)

Suppose you have a quad-core processor (4 physical cores).
The hypervisor can create 4 virtual machines, each assigned 1 virtual CPU (vCPU).

- **VM1:** Web Server
- **VM2:** Database Server
- **VM3:** File Server
- **VM4:** Application Server

➡️ The VMM schedules each vCPU to share the 4 physical cores efficiently, switching between them as needed.

⚖️ 8. Advantages of CPU Virtualization

✅ **1. Better Hardware Utilization:**
All CPU cores are efficiently used.

✅ **2. Isolation:**
Each VM operates independently and safely.

✅ **3. Flexibility:**
Run multiple OS and applications on one machine.

✅ **4. Cost Efficiency:**
Reduces need for multiple physical systems.

✅ **5. Load Balancing:**
VMM can dynamically allocate CPU time to VMs as per demand.

⚠️ 9. Limitations

❌ **Performance Overhead:** Context switching between VMs.
❌ **Complex Scheduling:** Hard to ensure fairness for all VMs.
❌ **Dependency on Hardware Support:** Older CPUs may not support VT-x or AMD-V.

📜 10. Summary Table

Aspect	Description
Concept	Sharing one physical CPU among multiple virtual machines
Controlled By	Virtual Machine Monitor (VMM) / Hypervisor
Mechanisms	Trap-and-Emulate, Paravirtualization, Hardware-Assisted Virtualization
Technologies	Intel VT-x, AMD-V
Examples	VMware ESXi, KVM, Xen, Hyper-V
Benefit	Efficient CPU usage and system isolation

✅ **In Short:**
CPU Virtualization allows multiple operating systems to share a single CPU simultaneously by creating virtual CPUs (vCPUs) managed by the hypervisor.
It uses techniques like Trap-and-Emulate, Binary Translation, Paravirtualization, and Hardware-Assisted Virtualization to achieve efficient and secure execution.

CPU virtualization is the technology that allows a single physical central processing unit (CPU) to be divided into multiple virtual CPUs (vCPUs). It enables a single physical machine to run multiple virtual machines (VMs) simultaneously, each with its own operating system and applications, completely isolated from one another. This process is managed by a software layer called a hypervisor.

How CPU virtualization works

A hypervisor manages the CPU resources and oversees the execution of instructions from each VM. Most unprivileged (non-critical) instructions from a VM can be run directly on the physical hardware for speed. However, critical, or "privileged," instructions that attempt to access or change hardware settings must be handled by the hypervisor to ensure isolation and stability.

There are three main techniques for CPU virtualization:

1. Hardware-assisted virtualization

- **Method**: This is the most common approach today. Modern CPUs from Intel (VT-x) and AMD (AMD-V) include special instruction sets that create new processor modes to assist virtualization.
- **How it works**: The hypervisor runs in a new, more privileged mode (often called the "root mode"), while the guest operating systems run in a less privileged mode. Any sensitive or privileged instructions from the guest OS are automatically trapped by the CPU and handed to the hypervisor, avoiding the need for software-based translation.
- **Benefit**: This method offers high efficiency and performance with minimal overhead.

2. Full virtualization (Software-based)

- **Method**: This technique provides a complete, software-based simulation of the underlying hardware. It was primarily used before hardware-assisted virtualization became widespread.
- **How it works**: Since guest OSes are not modified, a hypervisor must intercept and translate sensitive instructions that would normally run in privileged mode. This process, called binary translation, replaces sensitive instructions with new sequences that execute safely under the hypervisor's control.
- **Drawbacks**: Binary translation is a complex process that introduces significant performance overhead, which is why it is now considered obsolete.

3. Paravirtualization

- **Method**: In this approach, the guest operating system is modified to be "hypervisor-aware," allowing it to directly communicate with the hypervisor.
- **How it works**: Instead of simulating the hardware, the hypervisor provides a special application programming interface (API). The modified guest OS replaces its critical instructions with "hypercalls" to this API, which reduces the need for the hypervisor to intercept and translate them.
- **Benefit**: It is more efficient than full software virtualization because there is no emulation.
- **Drawback**: It requires modifications to the guest operating system, making it incompatible with unmodified operating systems like Windows.

Key components and processes

- **Hypervisor (or Virtual Machine Monitor):** This software layer is the core of virtualization. It creates and manages VMs, allocates system resources, and isolates VMs from each other. There are two types of hypervisors:

  - **Type 1 (Bare-Metal):** Runs directly on the host's hardware. This is common in data centers and cloud computing because it offers high performance and efficiency. Examples include VMware ESXi and Microsoft Hyper-V.
  - **Type 2 (Hosted):** Runs as a software application on top of a standard operating system. This is typically used for desktop virtualization. Examples include Oracle VirtualBox and VMware Workstation.
- **Virtual CPUs (vCPUs):** These are the virtual instances of the physical CPU that the hypervisor allocates to each VM.
- **CPU Scheduling:** The hypervisor determines when each VM gets access to the physical CPU, ensuring fair and efficient resource distribution.

Benefits of CPU virtualization

- **Efficient resource utilization:** Enables a single physical server to run multiple workloads, maximizing hardware usage and reducing the number of physical servers required.
- **Cost savings:** Reduces hardware, energy, and maintenance costs due to server consolidation.
- **Isolation and security:** Each VM is isolated from the others, preventing applications in one VM from affecting the performance or security of others.
- **Scalability:** Allows organizations to quickly add or remove virtual machines in response to changing workload demands.
- **Flexibility:** Enables the running of different operating systems and applications on the same hardware.

🧠 VIRTUALIZATION OF MEMORY AND I/O DEVICES

This topic explains how memory (RAM) and input/output devices (I/O) are virtualized so that multiple virtual machines (VMs) can share them safely and efficiently.

🧩 1. Memory Virtualization

💡 Definition:

Memory virtualization is the process of abstracting and managing physical memory (RAM) so that each virtual machine (VM) believes it has its own private memory space, even though all VMs are sharing the same physical RAM.

In simple words:

It makes every VM feel like it has separate, full access to system memory, even though the memory is shared.

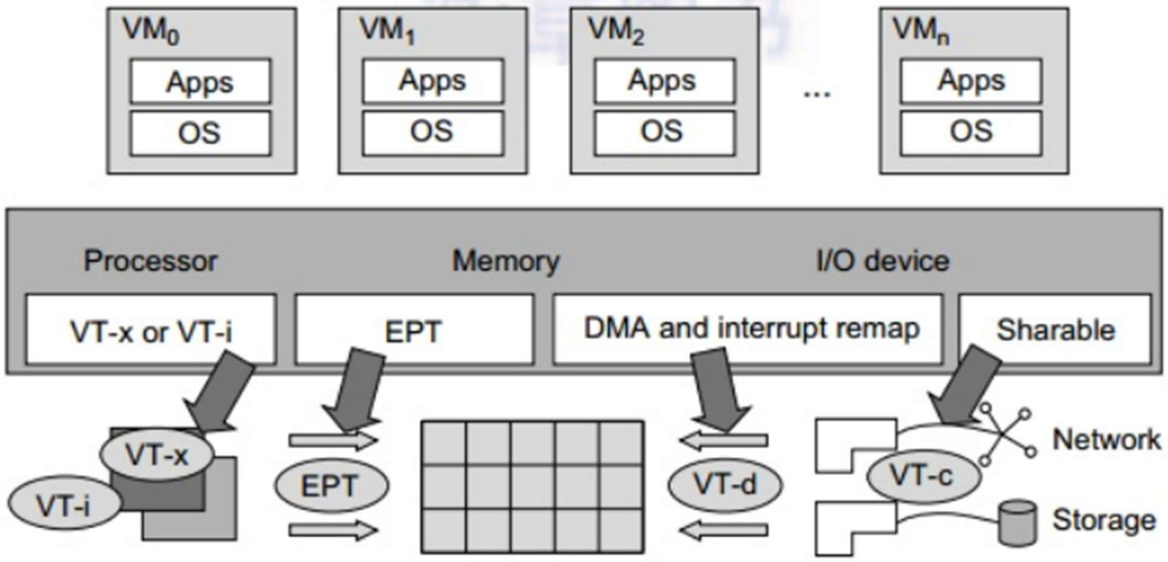
⚙️ Purpose of Memory Virtualization

Goal	Description
Isolation	Each VM's memory is protected from others.
Efficiency	Maximizes use of physical RAM.
Flexibility	VMs can have variable memory allocations.

Goal	Description
Security	Prevents one VM from reading another's data.

🔍 How It Works

1. Each **VM** has its **own virtual address space**.
2. The **VMM (Hypervisor)** translates:
   - **Guest Virtual Address (GVA) → Guest Physical Address (GPA)**
   - **Guest Physical Address (GPA) → Host Physical Address (HPA)**
3. These translations allow isolation between VMs and efficient memory usage.



🧠 Techniques of Memory Virtualization

Technique	Description
Shadow Page Tables (SPT)	The VMM maintains a mapping between guest and host memory pages. Slower but flexible.
Nested / Extended Page Tables (EPT/NPT)	Hardware-supported page tables in modern CPUs (Intel VT-x, AMD-V). Faster mapping.
Ballooning	Dynamically reallocates memory between VMs based on demand.
Memory Overcommitment	VMM gives more virtual memory to VMs than the actual physical RAM.
Copy-on-Write (CoW)	Multiple VMs share identical pages until one writes to them, saving memory.

💾 Example:

Suppose your system has **8 GB RAM**, and you create:

- VM1 = 4 GB

- VM2 = 4 GB
- VM3 = 2 GB

The hypervisor manages allocation dynamically, ensuring total physical RAM (8 GB) is efficiently shared using techniques like ballooning or overcommitment.

✅ Advantages of Memory Virtualization

- Efficient memory utilization
- Isolation between VMs
- Supports dynamic resizing
- Enables running more VMs than physical memory

⚠️ Disadvantages

- Overhead in memory translation
- Performance drop without hardware support
- Memory contention under heavy load

💻 2. I/O Device Virtualization

💡 Definition:

I/O Virtualization allows multiple VMs to share input/output devices (like disk drives, network cards, USB devices, etc.) as if each VM owns the device exclusively.
In simple words:
Each VM gets virtual devices (vNIC, vDisk, etc.) created by the hypervisor, which are connected to real physical devices underneath.

⚙️ Purpose of I/O Virtualization

Goal	Description
Sharing	Allow multiple VMs to use same physical devices.
Isolation	Prevent conflicts or interference.
Efficiency	Improve I/O performance through optimization.
Flexibility	Easily attach/detach devices from VMs.

🧠 Mechanisms of I/O Virtualization

Mechanism	Description
Device Emulation	Hypervisor emulates common hardware devices for VMs (e.g., virtual network card).
Direct I/O (Passthrough)	VM directly accesses physical device (needs IOMMU hardware). High performance.

Mechanism	Description
Paravirtualized Drivers	Special drivers (e.g., VirtIO) allow VMs to communicate directly with the VMM for faster I/O.
Split Driver Model	Driver divided into two parts: front-end (in guest VM) and back-end (in host), used in Xen.

Examples of Virtual I/O Devices

Type	Virtual Device Name	Physical Device Example
Network	vNIC (Virtual Network Interface Card)	Ethernet Adapter
Disk	vDisk (Virtual Disk)	SATA / SSD Drive
USB	vUSB	Physical USB Ports
Display	vGPU	Graphics Card

Advantages of I/O Virtualization

- Simplifies hardware sharing
- Reduces hardware cost
- Improves scalability
- Enables VM migration (vMotion)

Disadvantages

- May cause latency due to emulation
- Requires hardware support (VT-d / IOMMU)
- Complex configuration in some environments

3. Comparison Table

Aspect	Memory Virtualization	I/O Virtualization
Purpose	Share and manage system memory among VMs	Share and manage devices among VMs
Main Component	Page Tables, Address Translation	Device Drivers, Virtual I/O Interface
Techniques	Shadow Paging, EPT, Ballooning	Emulation, Paravirtualization, Passthrough
Hardware Support	Intel EPT / AMD NPT	Intel VT-d / AMD IOMMU
Example Tools	VMware, KVM	Xen, VMware vSphere, KVM
Benefit	Efficient RAM utilization	Shared device access

---

4. Summary

Memory Virtualization allows each virtual machine to have its own isolated memory space through address translation techniques like shadow paging and EPT.
I/O Virtualization enables multiple virtual machines to share and access hardware devices such as disks, network cards, and USB devices via emulation, passthrough, or paravirtualized drivers.

In Short :

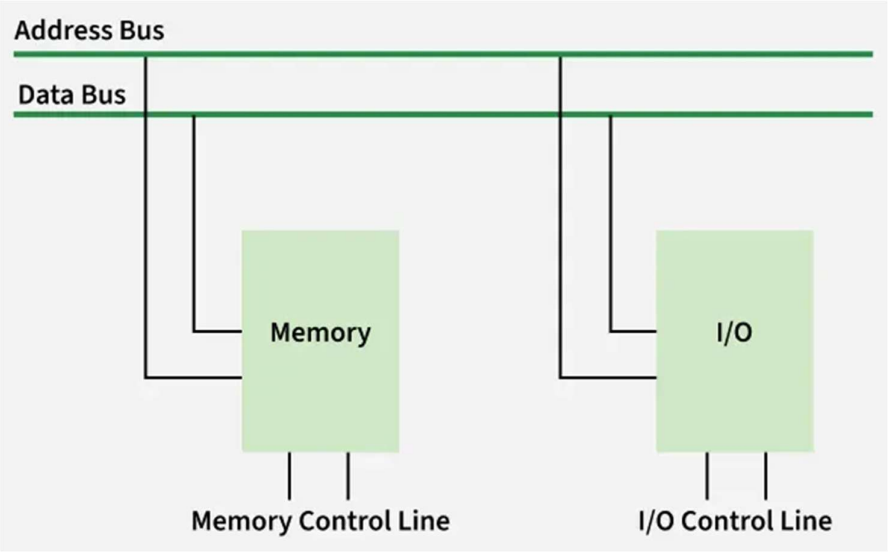
Topic	Definition	Example
Memory Virtualization	Allows multiple VMs to share physical memory safely by mapping guest memory to host memory.	Shadow Paging, EPT
I/O Virtualization	Allows multiple VMs to share physical I/O devices like disks or NICs through virtual interfaces.	

Memory Mapped I/O and Isolated I/O

CPU needs to communicate with the various memory and input-output devices (I/O). Data between the processor and these devices flow with the help of the system bus. There are three ways in which system bus can be allotted to them:

- Separate set of address, control and data bus to I/O and memory.
- Have common bus (data and address) for I/O and memory but separate control lines.
- Have common bus (data, address, and control) for I/O and memory.

In first case it is simple because both have different set of address space and instruction but it requires more buses.



Isolated I/O

In Isolated I/O, the CPU uses the same buses (wires) to talk to both memory and I/O devices, but it has separate control signals to tell whether it's dealing with memory or an I/O device.

- I/O devices have special addresses called ports.
- When the CPU wants to communicate with an I/O device:
- It puts the port address on the address bus.
- It uses special control lines like I/O Read or I/O Write.
- Then data is sent or received using the data bus.

As memory and I/O have separate address spaces, it's called Isolated I/O. Also, the CPU uses different instructions for memory and I/O (like IN and OUT for I/O).

Applications of Isolated I/O

- **Embedded Systems**: Isolated I/O is widely used in embedded systems where strict separation between the CPU and peripherals is essential. This includes applications like industrial control systems, robotics, and automotive electronics. Isolation helps ensure that faults or malfunctions in peripheral devices do not compromise the stability of the entire system.
- **Microcontrollers**: Microcontrollers often use isolated I/O to interface with various peripherals such as sensors, actuators, and displays. Each device is assigned a unique I/O port, allowing the microcontroller to communicate with multiple peripherals independently and efficiently.
- **Real-Time Systems**: In real-time systems, where precise timing and predictable behavior are critical, isolated I/O is preferred. It allows for strict control over timing and synchronization of external events, helping the system maintain reliable and deterministic performance.

Advantages of Isolated I/O

- **Large I/O Address Space**: Isolated I/O allows for a larger I/O address space because I/O devices have their own separate address space, independent of the system memory.
- **Greater Flexibility**: It offers greater flexibility, as I/O devices can be added or removed without affecting the memory address space.
- **Improved Reliability**: Since I/O devices do not share the same address space as memory, failures in I/O devices are less likely to affect the memory or other devices, improving system reliability.

Disadvantages of Isolated I/O

- **Slower I/O Operations**: I/O operations may be slower because isolated I/O requires special instructions, which add extra processing steps.
- **More Complex Programming**: Programming becomes more complex due to the need for dedicated I/O instructions, such as IN and OUT, which are separate from standard memory instructions.

Memory Mapped I/O

In a memory mapped I/O system, there are no special input or output instructions. Instead, the CPU uses the same instructions it uses for memory (like LOAD and STORE) to access I/O devices.

- Each I/O device is assigned a specific address in the regular memory address space.
- Devices are connected through interface registers, which act like memory locations.
- When the CPU wants to read from or write to an I/O device, it accesses the corresponding address, just like it would access a memory word.
- These interface registers respond to normal read/write operations as if they were memory cells.

This design allows I/O and memory to be treated uniformly, simplifying programming and hardware design.

Applications of Memory-Mapped I/O

- **Graphics Processing**: Memory-mapped I/O is widely used in graphics cards to provide fast access to frame buffers and control registers. Graphics data is mapped directly to memory, allowing the CPU to interact with the graphics hardware as if it were accessing normal memory. This enables efficient rendering and display operations.
- **Network Communication**: Network Interface Cards (NICs) often use memory mapped I/O to manage data transfer between the system memory and the network. The NIC's control and status registers are mapped to specific memory addresses, allowing the CPU to efficiently control and monitor network operations.
- **Direct Memory Access (DMA)**: DMA controllers use memory-mapped I/O to enable high-speed data transfers between I/O devices and system memory without involving the CPU. By mapping DMA control registers to memory, devices can transfer data directly, improving system performance and reducing CPU load.

Advantages of Memory-Mapped I/O

- **Faster I/O Operations**: Memory-mapped I/O allows the CPU to access I/O devices using the same mechanism and speed as regular memory access. This results in faster I/O operations compared to isolated I/O.
- **Simplified Programming**: Since the same instructions are used for both memory and I/O operations, programming becomes easier. Developers do not need to learn or use special I/O instructions, reducing complexity.
- **Efficient Use of Address Space**: Memory-mapped I/O enables I/O devices to share the same address space as memory. This can make the system more efficient, especially in systems with a unified memory model.

Disadvantages of Memory-Mapped I/O

- **Limited I/O Address Space**: Because memory and I/O devices share the same address space, the number of available addresses for I/O devices is limited. This can be a problem in systems with many peripherals.
- **Potential Performance Issues**: If an I/O device responds slowly, it may delay the CPU when accessing that memory-mapped region. This can affect overall system performance, especially in time-sensitive tasks.

Differences between memory mapped I/O and isolated I/O

Let us see the difference between the memory mapped I/O and isolated I/O in the below table:

Aspect	Isolated I/O	Memory-Mapped I/O
Address Space	Memory and I/O have separate address spaces	Memory and I/O share the same address space
Memory Usage	All addresses can be used for memory	Some address space is used for I/O, reducing memory space
Instruction Set	Separate instructions for I/O and memory read/write operations	Same instructions are used for both I/O and memory
I/O Addressing	I/O addresses are called ports	Regular memory addresses are used for both memory and I/O
Efficiency	More efficient due to separate control lines and buses	Slightly less efficient due to shared resources
Hardware Size	Larger hardware due to additional buses and logic	Smaller hardware as fewer buses are needed
Design Complexity	More complex requires separate logic for I/O and memory	Simpler design