**UNIT – II**

**Virtual Machines and Virtualization of Clusters and Data Centers:** Implementation Levels of Virtualization, Virtualization Structures/ Tools and mechanisms, Virtualization of CPU, Memory and I/O Devices, Virtual Clusters and Resource Management, Virtualization for Data Centre Automation, Technology Examples like Xen: Paravirtualization, VMware: Full Virtualization, Microsoft Hyper-V.

**-**------------------------------------------------------------------------------------------------------------------------------

**Implementation Levels of Virtualization:**

Virtualization in cloud computing is applied at multiple levels depending on which resource is abstracted. These levels enable efficient utilization of hardware, isolation of workloads, and scalability of cloud environments.

**1. Instruction Set Architecture (ISA) Level**

* Virtualization occurs at the **machine instruction level**.
* Provides compatibility between different hardware and software platforms.
* Example: **Binary Translation** allows execution of one ISA on a different ISA (e.g., running x86 code on ARM).
* Useful for cross-platform execution but **slower** due to heavy translation overhead.

**2. Hardware Level**

* Virtualization happens directly on **physical hardware** using a **Virtual Machine Monitor (VMM)/Hypervisor**.
* Creates multiple isolated **Virtual Machines (VMs)** on the same hardware.
* Examples: VMware ESXi, Microsoft Hyper-V.
* Provides **high performance and security isolation**, but hardware support (Intel VT-x, AMD-V) is required.

**3. Operating System (OS) Level**

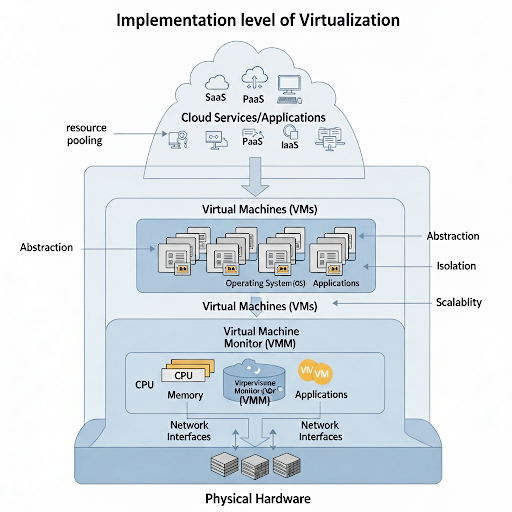
* Virtualization occurs at the **OS kernel level**.
* Instead of full VMs, it creates **multiple isolated containers** (lightweight).
* Examples: Docker, OpenVZ, LXC.
* Efficient resource utilization, but all containers must use the **same OS kernel**.

**4. Library Level**

* Virtualization at the **API/library interface** between applications and the OS.
* Applications link against **virtualized libraries** instead of real ones.
* Example: Wine (to run Windows apps on Linux), JVM (Java Virtual Machine).
* Allows **cross-platform compatibility**, but limited in scope compared to full virtualization.

**5. Application Level**

* Virtualization applied at the **application runtime environment**.
* Each application runs in its own isolated environment (sandboxing).
* Example: Java Virtual Machine (JVM), .NET CLR.
* Provides **portability and isolation** but **does not virtualize hardware**.



**Virtualization Structures, Tools, and Mechanisms in Cloud Computing**

Virtualization uses different **structures and tools** to abstract physical resources and provide multiple virtual environments.

**1. Virtual Machine Monitor (VMM) / Hypervisor**

* **Function:** Manages virtual machines (VMs) and allocates resources.
* **Types:**
  + **Type-1 (Bare Metal):** Runs directly on hardware. (e.g., VMware ESXi, Xen, Hyper-V)
  + **Type-2 (Hosted):** Runs on top of an OS. (e.g., VirtualBox, VMware Workstation)
* **Key Role:** Provides **isolation, scheduling, and resource control**.

**2. Binary Translation**

* Translates **guest OS instructions** into host machine instructions.
* Ensures compatibility across different instruction sets (ISA).
* Example: VMware Workstation uses **binary translation** for privileged instructions.

**3. Hardware-Assisted Virtualization**

* Uses CPU features for virtualization (Intel VT-x, AMD-V).
* Improves **efficiency** by allowing direct execution of VM instructions.
* Example: Modern hypervisors like KVM, Hyper-V.

**4. Paravirtualization**

* Guest OS is **modified** to interact with hypervisor via **hypercalls**.
* Provides **better performance** than full virtualization.
* Example: Xen Paravirtualization.

**5. Operating System-Level Virtualization**

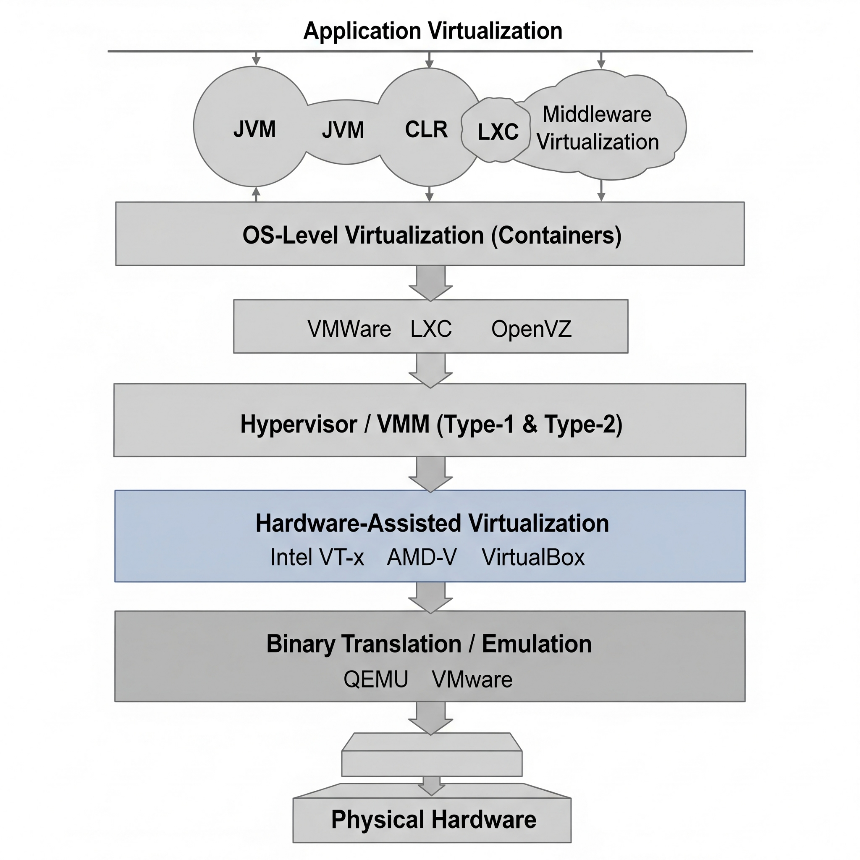
* Multiple **isolated user-spaces** (containers) share the same OS kernel.
* Lightweight, fast startup.
* Examples: Docker, LXC, OpenVZ.

**6. Emulation**

* Complete hardware architecture emulated in software.
* Allows running **different OS architectures** (e.g., ARM apps on x86).
* Example: QEMU.

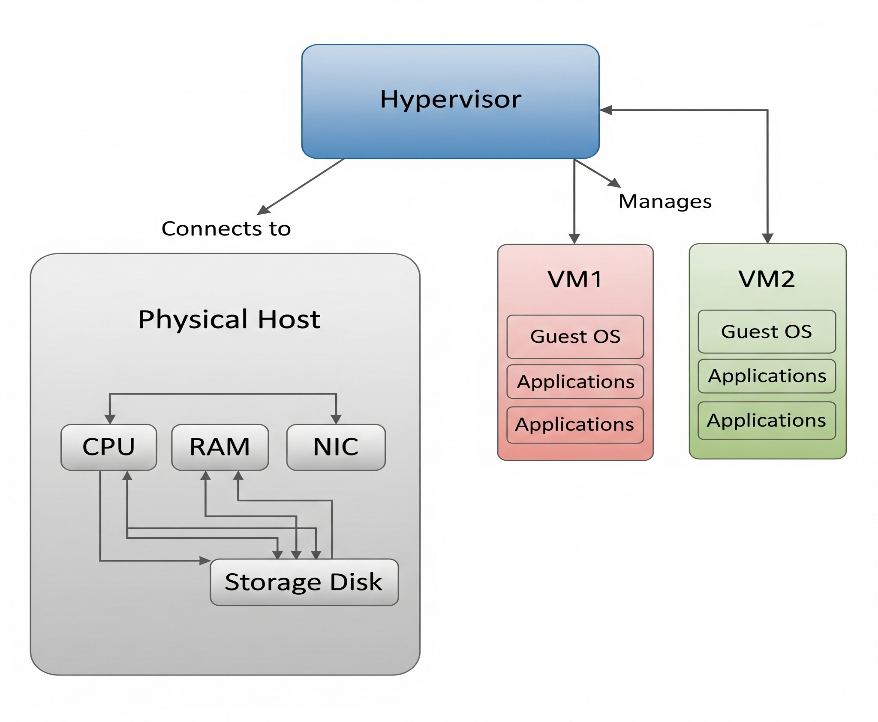
**7. Middleware Virtualization**

* Abstracts APIs and provides **platform-independent execution**.
* Example: Java Virtual Machine (JVM), .NET CLR.



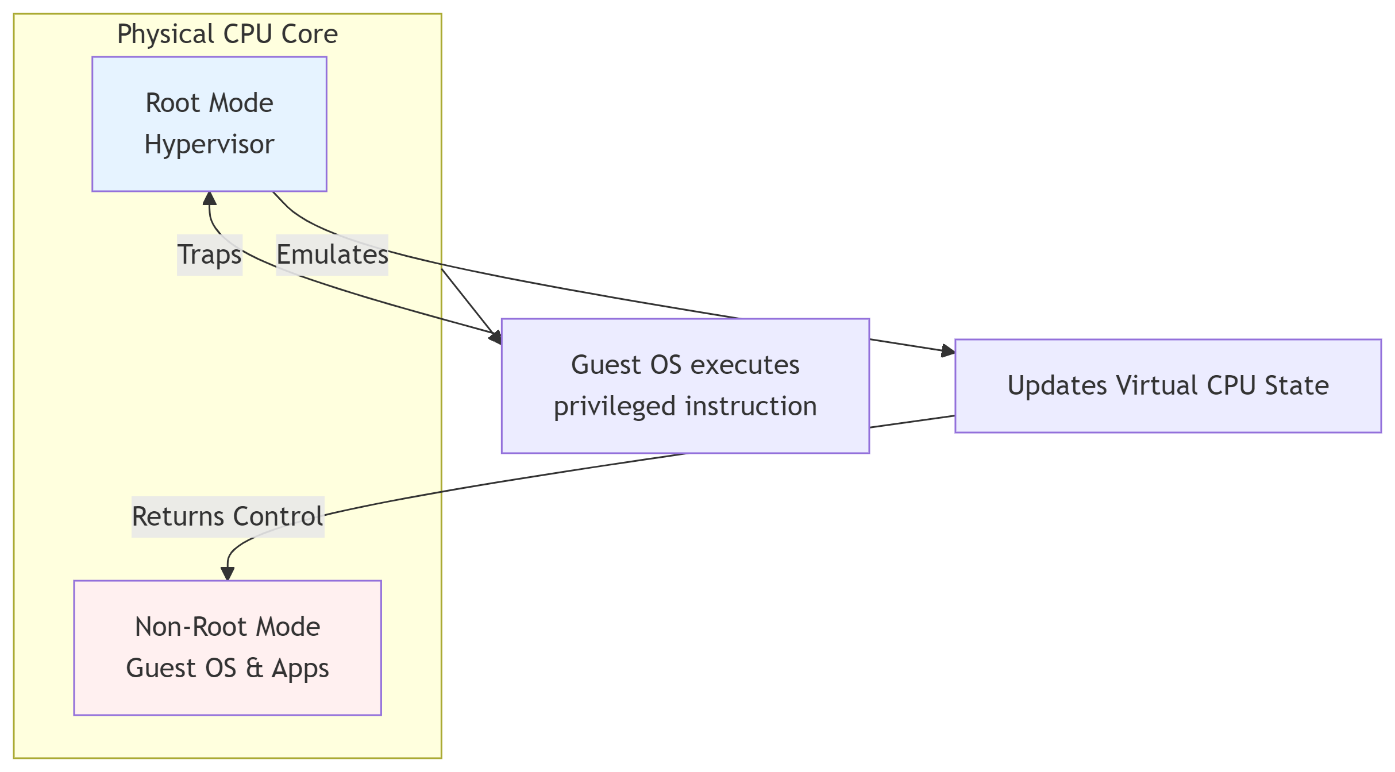
**Virtualization of CPU, Memory, and I/O Devices in Cloud Computing:**

At the heart of Infrastructure-as-a-Service (IaaS) cloud computing lies the ability to virtualize the core physical components of a server—CPU, Memory, and I/O devices—and present them as multiple, independent virtual machines (VMs). The **Hypervisor** (or Virtual Machine Monitor - VMM) is the software layer that manages this abstraction and allocation.

The core challenge is that operating systems are designed to have full, exclusive control over the hardware (they run in **privileged mode**, kernel mode). The hypervisor must allow multiple guest OSes to run simultaneously, tricking each into thinking it has exclusive control, while the hypervisor itself retains ultimate control.

CPU virtualization involves multiplexing and time-sharing the physical CPU cores among multiple virtual machines.

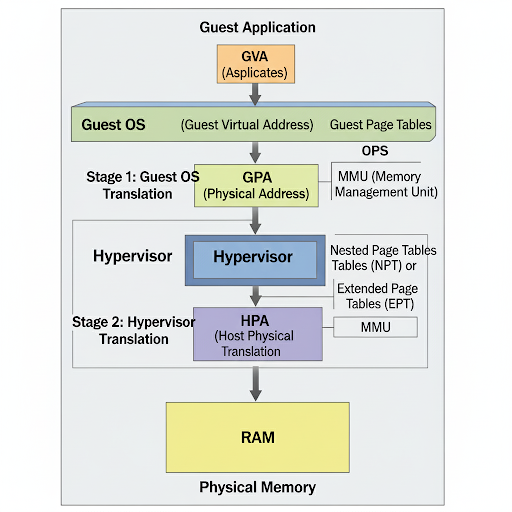
* **The Challenge:** A guest OS inside a VM tries to execute a **privileged instruction** (e.g., HLT to halt the CPU, or change memory mapping). If executed directly, this would give the guest OS control over the entire physical machine, breaking isolation.
* **The Solution:**
  1. **Trap-and-Emulate (Classic Method):**
     + The guest OS runs in **user mode** (de-privileged), while the hypervisor runs in **privileged mode**.
     + When the guest OS executes a privileged instruction, it **traps** (causes a fault) to the hypervisor.
     + The hypervisor **emulates** the effect of that instruction in a safe way (e.g., updating the virtual machine's state structures) and returns control to the guest OS.
     + This emulation is transparent to the guest OS.
  2. **Binary Translation (Software Assist):**
     + Used before hardware support was widespread. The hypervisor dynamically scans and translates sensitive guest kernel code into safe instructions on the fly.
  3. **Hardware-Assisted Virtualization (Modern Standard):**
     + Intel (VT-x) and AMD (AMD-V) introduced new CPU extensions.
     + **Intel VT-x:** Introduces two new modes:
       - **Root Mode:** The privileged mode where the hypervisor runs.
       - **Non-Root Mode:** The de-privileged mode where guest OSes run. This mode has its own set of privileged instructions that **trap** to the hypervisor in Root Mode without causing a general fault.
     + This hardware support drastically reduces the performance overhead of virtualization.



**Memory Virtualization**

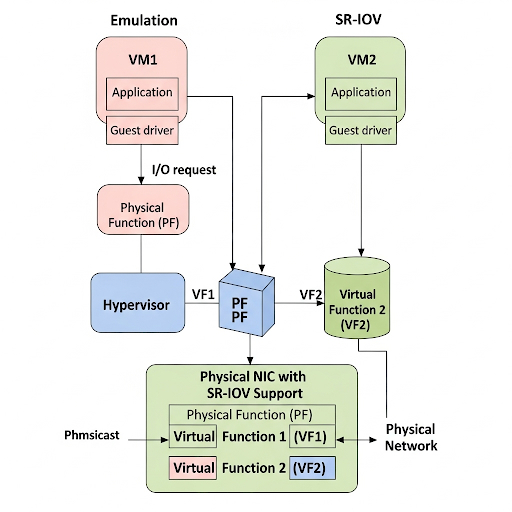
Memory virtualization involves mapping the **physical memory (RAM)** of the host to the **physical memory** as seen by each guest VM. Each VM believes it has a contiguous address space starting from address 0.

* **The Challenge:** The guest OS expects to manage its own **Virtual-to-Physical** address mapping using its page tables. However, the "physical" addresses it sees are actually **pseudo-physical addresses**. The hypervisor must translate these into **true machine addresses**.
* **The Solution: Nested Page Tables / Extended Page Tables (Hardware Assist)**
  1. The guest OS maintains its own page tables, translating **guest virtual addresses (GVA)** to **guest physical addresses (GPA)**.
  2. The hypervisor maintains a **shadow page table** (older software method) or, more efficiently, uses hardware support to maintain a **nested page table (NPT - AMD)** or **extended page table (EPT - Intel)**.
  3. This second set of page tables, managed by the Memory Management Unit (MMU) in the CPU, automatically translates the **GPA** into the **host physical address (HPA)**.
  4. The CPU's MMU hardware handles this two-level translation on the fly, minimizing performance penalty.



**I/O Device Virtualization**

I/O virtualization involves managing access to shared physical hardware like network cards (NICs) and storage disks (HBAs) among VMs.

* **The Challenge:** Physical I/O devices are limited and cannot be directly accessed by multiple VMs. Emulating them in software is slow.
* **Common Solutions:**
  1. **Full Emulation (e.g., QEMU):**
     + The hypervisor emulates a well-known, simple hardware device (e.g., an Intel E1000 NIC).
     + The guest OS uses its standard driver for this emulated device.
     + All read/write commands are trapped to the hypervisor, which translates them to commands for the real physical device.
     + **Pros:** Highly compatible. **Cons:** High CPU overhead due to many traps and context switches.
  2. **Paravirtualization (PV):**
     + The guest OS is modified and uses a special **paravirtualized driver**.
     + This driver knows it's running in a VM and communicates directly with the hypervisor through efficient hypercalls (a lightweight call into the hypervisor), instead of emulating hardware.
     + **Pros:** Much higher performance than emulation. **Cons:** Requires modified guest OS drivers.
  3. **Direct I/O Passthrough (e.g., Intel VT-d, AMD-Vi):**
     + The hypervisor grants a VM exclusive access to a physical PCI/PCIe device (e.g., a high-performance NIC).
     + The VM's driver talks directly to the hardware with minimal hypervisor involvement.
     + **Pros:** Near-native performance. **Cons:** Defeats multi-tenancy; one device is dedicated to one VM.

**Virtual Clusters and Resource Management:**

Virtual Clusters in cloud computing are groups of virtual machines (VMs) that are configured to work together as a single, unified computing resource. They abstract the underlying physical infrastructure, providing users with a flexible and scalable environment to deploy and manage applications.

**Virtual Clusters**

* **Definition:** A group of **virtual machines (VMs)** running on physical servers, connected via a virtual network to function as a single cluster.
* **Purpose:** To provide **scalability, flexibility, and isolation** for applications.
* **Key Features:**
  + VMs can run different **operating systems**.
  + Easy to **provision and scale** compared to physical clusters.
  + Enable **fault tolerance** and **load balancing**.
* **Examples:** Hadoop clusters on VMs, Kubernetes clusters.

**Components of a Virtual Cluster**

* **Virtual Machines (VMs):** Provide compute instances.
* **Virtual Network:** Ensures communication between VMs.
* **Hypervisor (VMM):** Creates and manages VMs.
* **Physical Infrastructure:** CPU, memory, storage, and network hardware.

**Resource Management in Virtual Clusters**

Resource management ensures **efficient allocation and utilization** of cloud resources.

**a) CPU Management**

* Allocates **virtual CPUs (vCPUs)** to VMs.
* Uses **scheduling** (time-sharing, fair allocation).
* Example: VMware CPU scheduler.

**b) Memory Management**

* Provides each VM with **virtual memory**.
* Supports **memory overcommitment, ballooning, and swapping**.
* Ensures isolation and efficient usage.

**c) Storage Management**

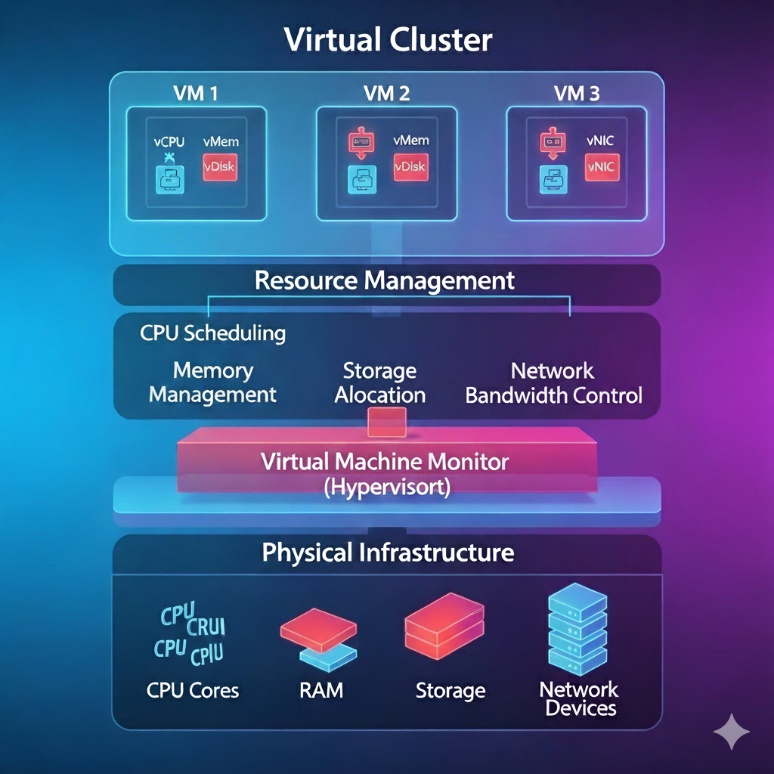
* Abstracts physical disks into **virtual storage (vDisks)**.
* Supports **block storage, object storage, distributed file systems**.
* Enables replication and fault tolerance.

**d) I/O & Network Management**

* Virtual NICs (vNICs) allow network connectivity.
* **Bandwidth allocation & traffic shaping** for QoS.
* Supports **SR-IOV** and paravirtualized drivers for performance.

**e) Scheduling & Load Balancing**

* Distributes workloads across multiple VMs.
* Prevents **resource hotspots**.
* Improves system performance and fault tolerance.



**Virtualization for Data Centre Automation:**

**Introduction**

* Data centres host **large numbers of servers, storage, and networking devices**.
* Virtualization is the **key enabler** for automation in data centres.
* It abstracts physical resources into **virtual pools**, allowing **automation tools** to dynamically allocate them.

**Role of Virtualization in Data Centre Automation:**

1. **Server Virtualization**
   * Multiple VMs run on a single physical server.
   * Enables **server consolidation** and better utilization.
2. **Storage Virtualization**
   * Abstracts multiple physical storage devices into a **single logical pool**.
   * Automates **allocation, replication, and backup**.
3. **Network Virtualization**
   * Creates **virtual networks, firewalls, and load balancers**.
   * Automates **traffic routing, bandwidth management, and QoS**.
4. **Application Virtualization**
   * Applications run in isolated containers/VMs.
   * Automates **deployment, scaling, and updates**.

**Benefits of Virtualization in Data Centre Automation:**

* **Resource Pooling** → Unified management of compute, storage, and network.
* **Dynamic Resource Allocation** → Automatically adjusts resources as per workload.
* **Load Balancing** → Distributes workloads across VMs for efficiency.
* **Energy Efficiency** → Reduces power usage by consolidating servers.
* **High Availability & Fault Tolerance** → VMs can migrate (vMotion, Live Migration).
* **Simplified Management** → Automated monitoring and provisioning via orchestration tools (OpenStack, vCloud, Kubernetes).

**Mechanisms Used:**

* **Hypervisors** (VMware ESXi, KVM, Xen).
* **Cloud Orchestration Tools** (OpenStack, Kubernetes).
* **Automation Platforms** (VMware vRealize, Ansible, Terraform).

