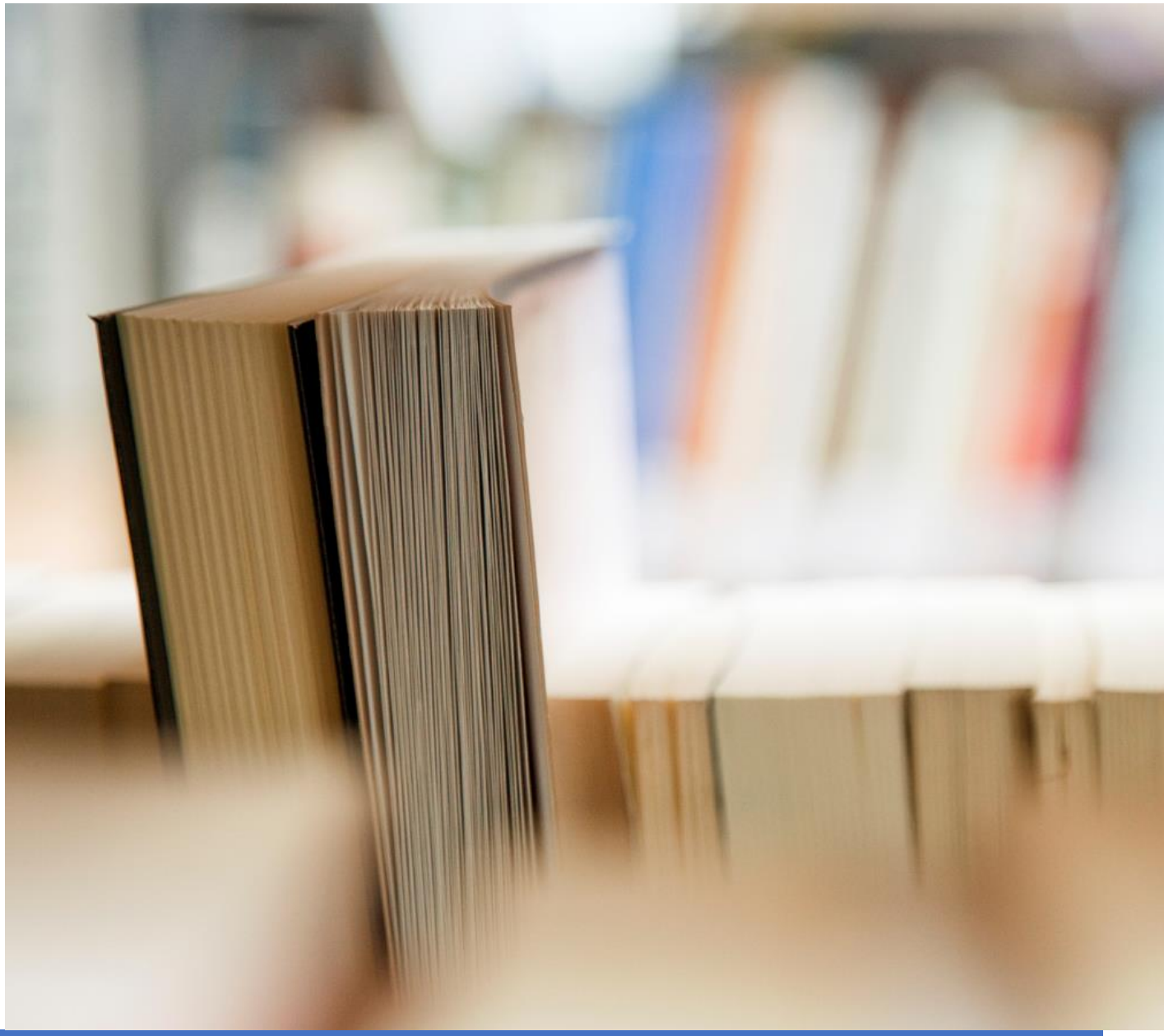


Introduction to Virtualization & Cloud Computing

INTRODUCTION

Raj Ishu | B.Tech. CSE – CCVT | Batch - 2

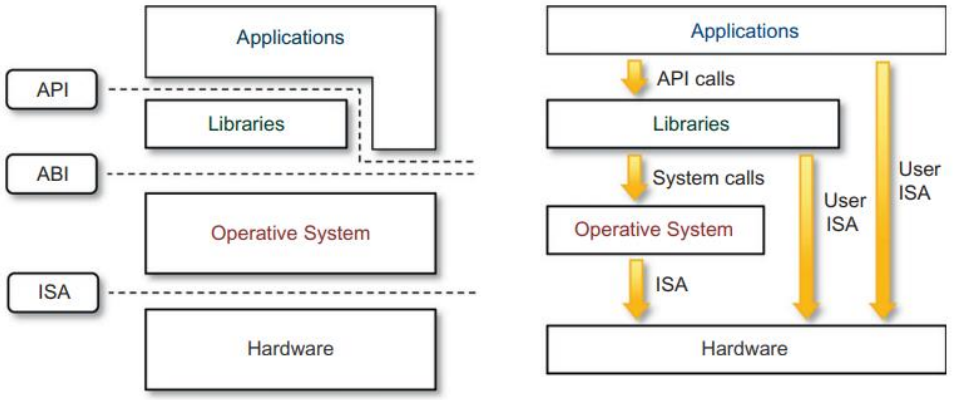
Virtualization Techniques

1. **Introduction:**
   - Virtualization covers a wide range of emulation techniques that are applied to different areas of computing.
   - Virtualization is mainly used to emulate execution environments, storage, and networks.

2. **Execution virtualization:**
   - Execution virtualization includes all techniques that aim to emulate an execution environment that is separate from the one hosting the virtualization layer.
     - An execution environment that is separate from the one hosting the virtualization layer means an environment which will not interfere with real execution environment. It will be isolated from real execution environment.
   - **Execution virtualization** techniques can be divided into two major categories by considering the type of host they require.
     a. **Process-level** techniques are implemented on top of an existing operating system, which has full control of the hardware.
     b. **System-level** techniques are implemented directly on hardware and do not require support from an existing operating system.

3. **Machine Reference Model:**
   - Virtualizing an execution environment at different levels of the computing stack requires a reference model that defines the interfaces between the levels of abstractions, and this level of abstraction hides the details of implementations.
   - Virtualization techniques actually replace one of the layers and intercept the calls that are directed toward it. Therefore, a clear separation between layers simplifies their implementation, which only requires the emulation of the interfaces and a proper interaction with the underlying layer.
   -
   - **Interface:** It is an "existing entity" layer between the functionality and consumer of that functionality. An interface by itself doesn't do anything. It just invokes the functionality lying behind.

   - Modern computing systems can be expressed in terms of the reference model described below:



- **Instruction Set Architecture (ISA):**
  - The ISA acts as an interface between the hardware and the software, specifying both what the processor is capable of doing as well as how it gets done.
  - ISA defines the instruction set for the processor, registers, memory, and interrupt management.
    - **Instruction Set:**
      - An instruction set is a group of commands for a CPU in **machine language**.
      - **Examples of instruction set:**
        a. **ADD** - Add two numbers together.
        b. **COMPARE** - Compare numbers.
        c. **IN** - Input information from a device, e.g., keyboard.
        d. **JUMP** - Jump to designated RAM address.
        e. **JUMP IF** - Conditional statement that jumps to a designated RAM address.
        f. **LOAD** - Load information from RAM to the CPU.
        g. **OUT** - Output information to device, e.g., monitor.
        h. **STORE** - Store information to RAM.
  - In computer science, an **instruction set architecture (ISA)**, also called **computer architecture**, is an abstract model of a computer.
  - An Instruction Set Architecture (ISA) is part of the abstract model of a computer that defines how the CPU is controlled by the software.
  - The ISA provides the only way through which a user is able to interact with the hardware.
  - ISA is important to the operating system (OS) developer (System ISA) and developers of applications that directly manage the underlying hardware (User ISA) as it can be viewed as a programmer's manual because it's the portion of the machine that's visible to the assembly language programmer, the compiler writer, and the application programmer.
  - Some Popular Instruction Set Architectures are:
    a. Reduced Instruction Set Computer (RISC)
    b. Complex Instruction Set Computer (CISC)
    c. Minimal instruction set computers (MISC)
    d. Very long instruction word (VLIW)
    e. Explicitly parallel instruction computing (EPIC)
    f. One instruction set computer (OISC)
    g. Zero instruction set computer (ZISC)

- **Application Binary Interface (ABI):**
  - The application binary interface (ABI) separates the operating system layer from the applications and libraries, which are managed by the OS.

    - ABI covers details such as low-level data types, alignment, calling convention, the system call numbers and how an application should make system calls to OS.
    - An application binary interface (ABI) is an interface between two binary program modules to utilize external and/or underlying software or hardware.

      Here, binary interface means as it as an interface between two applications that deal with binary code or binary language.

    - ABI defines how an application should make system calls to the operating system.

- **Application Programming Interface:**
    - API interfaces applications to libraries and/or the underlying operating system.
    - An API defines the interfaces by which one piece of software communicates with another at the source level.
    - APIs simplify software development and innovation by enabling applications to exchange data and functionality easily and securely.

- For any operation to be performed in **the application-level** API, ABI and ISA are responsible for making it happen.
- The high-level abstraction is converted into machine-level instructions to perform the actual operations supported by the processor.
- The machine-level resources, such as processor registers and main memory capacities, are used to perform the operation at the hardware level of the central processing unit (CPU).
- This layered approach simplifies the development and implementation of computing systems.
  In fact, such a model not only requires limited knowledge of the entire computing stack, but it also provides ways to implement a minimal security model for managing and accessing shared resources.

- The instruction set exposed by the hardware has been divided into different security classes that define who can operate with them.
    1. **<u>Non privileged Instructions</u>:**
        - Nonprivileged instructions are those instructions that can be used without interfering with other tasks because they do not access shared resources.
        - *The Instructions that can run only in **User Mode** are called Non-Privileged Instructions.*
        -

    2. **<u>Privileged Instructions</u>:**
        - Privileged instructions are those that are executed under specific restrictions and are mostly used for sensitive operations, which expose (behaviour-sensitive) or modify (control-sensitive) the privileged state.

Behaviour-sensitive instructions are those that operate on the I/O, whereas **control-sensitive instructions** alter the state of the CPU registers.

- *The Instructions that can run only in **Kernel Mode** are called Privileged Instructions.*
- A trap or intercept or in other words we can say system call is required or generated to switch from user mode to kernel mode.
- Interchanging the Mode from user to kernel or kernel to the user is referred to as **mode switching.**
- Non-privileged Instruction that does not generate any interrupt is used **to change the mode from Privileged to Non-Privileged**.

- **Protection Rings:**
  - Computer operating systems provide different levels of access to resources.
  - In Computer Science, the ordered protection domains are referred to as **Protection Rings**.
  - A hierarchy of privileges in the form of ring-based security, is referred as **Protection Rings.**
  - Rings are arranged in a hierarchy from **most privileged** (most trusted, usually **numbered zero**) to **least privileged** (least trusted, usually with **the highest ring number**).

  - **Use of Protection Rings:**

    Use of Protection Rings provides logical space for the levels of permissions and execution.
    Two important uses of Protection Rings are:
    a. Improving Fault Tolerance
    b. Provide Computer Security

  - **Levels of Protection Ring:**

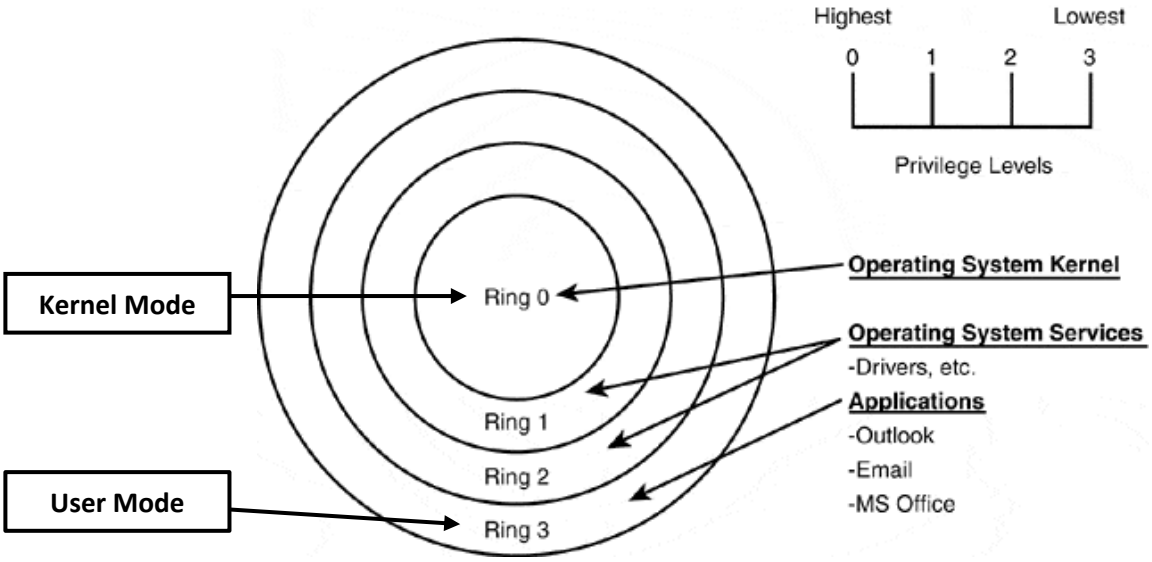
    There are basically 4 levels ranging from 0 which is the most privileged to 3 which is least privileged.

    a. Ring 0 or **Layer 0:**
    - Layer 0 is the most trusted level.
    - The operating system kernel resides at this level.
    - Any process running at layer 0 is said to be operating in privileged mode.
    - Ring 0 interacts most directly with the physical hardware such as the CPU and memory.

    b. Ring 1 or **Layer 1:**
    - Layer 1 contains nonprivileged portions of the operating system.

    c. Ring 2 or **Layer 2:**
    - Layer 2 is where I/O drivers, low level operations, and utilities reside.

d. **Ring 3** or **Layer 3:**
   - Layer 3 is where applications and processes operate.
   - Items such as FTP, DNS, Telnet, and HTTP all operate at this level.
   - This is the level at which individuals usually interact with the operating system.
   - Applications operating here are said to be working in user mode.



NOTE: Most Operating Systems use only two level, level 0 as the kernel or executive and use level 3 for application programs.

- **Execution Modes:**

  There are basically two modes:

  a. **Kernel Mode:**
     - Also known as **Supervisor Mode** or **Master Mode**.
     - It is an execution mode in which all the instructions (privileged and nonprivileged) can be executed without any restriction.
     - In Kernel mode, the executing code has complete and unrestricted access to the underlying hardware.
     - Kernel mode is generally used by the operating system (or the hypervisor) to perform sensitive operations on hardware level resources.
     - Kernel mode can execute any CPU instruction and reference any memory address.

b. User Mode:

- In user mode, there are restrictions to control the machine-level resources.
- In User mode, the executing code has no ability to directly access hardware or reference memory.
- If code running in user mode invokes the privileged instructions, hardware interrupts occur and trap the potentially harmful execution of the instruction.

NOTE:

1. **CPU cycle:** The time required for the execution of one simple processor operation such as an addition.
2. **Clock Rate:**
   - As we know, **frequency** is the number of occurrences of a repeating event per unit of time.
   - The **clock rate** is the frequency at which a CPU is running.
   - It is also known as **Clock Speed**.
   - It is measured in Hertz (Hz) or Cycles per second.
   - One cycle per second is known as 1 hertz.
   - A CPU with a clock speed of 2 gigahertz (GHz) can carry out two thousand million (**or two billion**) cycles per second.
   - The higher the clock speed a CPU has, the faster it can process instructions.

3. **Cycles per Instruction (CPI):**
   - A measure of how a given piece of code or the entire application is performing is to look at the average number of cycles that are needed to retire an instruction.
   - Cycles per instruction (CPI) is actually a ratio of two values. The numerator is the number of CPU cycles uses divided by the number of instructions executed.

4. **Instruction cycle:**
   - The time required by the CPU to execute one single instruction.
   - The instruction cycle is the basic operation of the CPU which consist of three steps.
   - The CPU repetitively performs fetch, decode, execute cycles to execute one program instruction.

5. **Context Switching:**
   - Switching of CPU from one process to another
6.

4. Hardware – Level Virtualization:

- Hardware-level virtualization is a virtualization technique that provides an abstract execution environment in terms of computer hardware on top of which a guest operating system can be run.

- A fundamental element of hardware virtualization is the **hypervisor**, or **virtual machine manager (VMM)**.
  It recreates a hardware environment in which guest operating systems are installed.

Hypervisors:

- The hypervisor manages shared the physical resources of the hardware between the guest operating systems and host operating system.
- Also known as Virtual Machine Monitor (VMM) or Virtual Machine Manager.
- Hypervisors are Resource Managers.

- **Types of Hypervisors:**

1. **Type – 1** (or **Bare Metal**)**:**
   - It acts like a lightweight operating system and runs directly on top of the hardware.
     Therefore, they take the place of the operating systems and interact directly with the ISA interface exposed by the underlying hardware, and they emulate this interface in order to allow the management of guest operating systems.

   - This type of hypervisor is also called a **native virtual machine** since it runs natively on hardware.

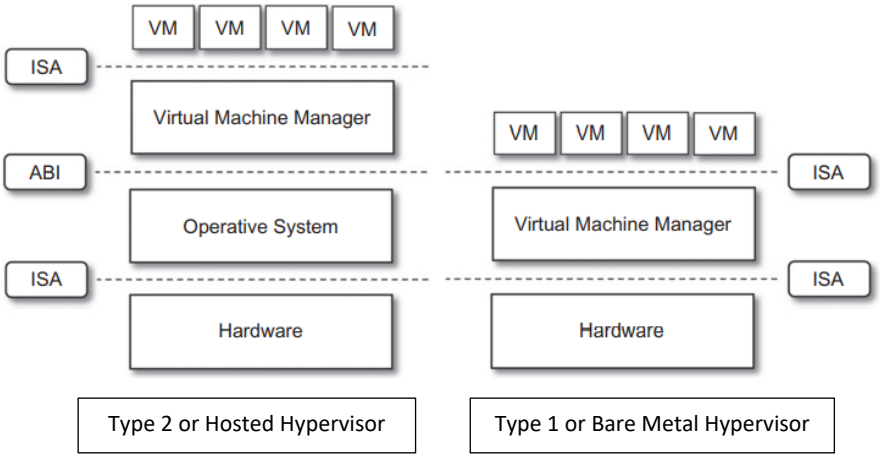
   - Examples:
     - VMware ESXi
     - Citrix XenServer
     - Microsoft Hyper-V hypervisor.
   - These hypervisors are most commonly used or deployed i.e., most enterprise companies choose bare-metal hypervisors for data center computing needs.
     - **Reason:**
       - **Bare Metal Hypervisors** or Virtualization Software is installed directly on the hardware where the operating system is normally installed. Because bare-metal hypervisors are isolated from the attack-prone operating system, they are extremely secure.
       - **Bare Metal Hypervisors**, generally perform better and more efficiently than hosted hypervisors because of **low latency**.
       - **Lower Latency** than Hosted Hypervisors because virtualization software is installed directly on the hardware so there is direct communication between the hardware and the hypervisor.

2. **Type – 2** (or **Hosted**)**:**
   - It runs as a software layer on an operating system, like other computer programs.

- The Virtualization Software is installed directly on the Operating System layer like other computer programs.

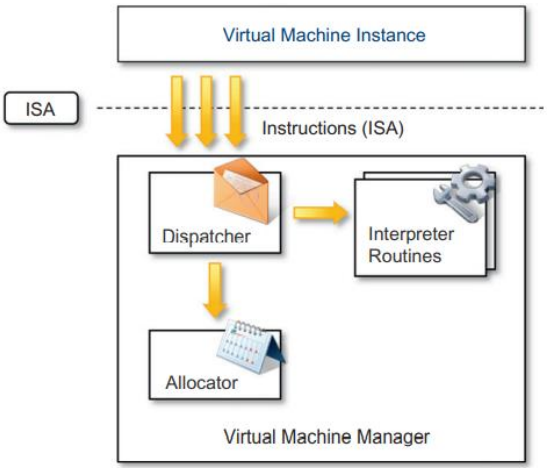
- These types of hypervisors require the support of an operating system to provide virtualization services.
  This means that they are programs managed by the operating system, which **interact with it through the ABI** and emulate the ISA of virtual hardware for guest operating systems.

- Hosted hypervisors run within the OS, additional (and different) operating systems can be installed on top of the hypervisor.
- Hypervisor asks the operating system to make hardware calls.
- **Latency is very high** because communication between the hardware and the hypervisor must pass through the extra layer of the OS.
- Hosted hypervisors are sometimes known as **Client Hypervisors** because they are most often used with end users and software testing, where higher latency is less of a concern.
- Examples:
  - VMware Player
  - Virtual Box
  - Parallels Desktop



- **<u>Hypervisor Reference Model</u>:**

  - Conceptually, a virtual machine manager is internally organized as:



- There are 3 main modules coordinates in order to emulate the underlying hardware:
  a. **DISPATCHER**:
     - The dispatcher behaves like the entry point of the monitor and reroutes the instructions of the virtual machine instance to one of the other two modules.

  b. **ALLOCATOR:**
     - The allocator is responsible for **deciding the system resources to be provided to the virtual machine instance**.
     - It means whenever virtual machine tries to execute an instruction that results in changing the machine resources associated with the virtual machine, the allocator is invoked by the dispatcher.

  c. **INTERPRETER:**
     - The interpreter module consists of interpreter routines.
     - These are executed, whenever virtual machine executes a privileged instruction, a trap is triggered and the corresponding routine is executed.

- **Goldberg and Popek Requirements and Theorem:**

  According to Popek and Goldberg, three properties have to be satisfied by a virtual machine manager to efficiently support virtualization –

  a. Equivalence:
     - A guest running under the control of a virtual machine manager should exhibit the same behaviour as when it is executed directly on the physical host.

  b. Resource Control or **Safety:**
     - The VMM must be in complete control of the virtualized resources.

  c. Efficiency:
     - A statistically dominant fraction of the machine instructions should be executed without intervention from the virtual machine manager.
     - All safe guest instructions are executed by the hardware directly, no VMM intervention required.

Theorems:

- **Theorem 1:** For any conventional third-generation computer, an effective VMM may be constructed if the set of sensitive instructions for that computer is a subset of the set of privileged instructions.

  Explanation: The theorem states that to build a VMM it is sufficient that all instructions that could affect the correct functioning of the VMM (sensitive instructions) always generate a trap and pass control to the VMM. This guarantees the resource control property.

Non-privileged instructions must instead be executed natively (i.e., efficiently). The holding of the equivalence property also follows.

This theorem also provides a simple technique for implementing a VMM, called ***trap-and-emulate virtualization***, more recently called ***classic virtualization***: because all sensitive instructions behave nicely, all the VMM has to do is trap and emulate every one time of them.

- **Theorem 2:** A conventional third-generation computer is recursively virtualizable if:
    a. it is virtualizable and
    b. a VMM without any timing dependencies can be constructed for it.

  Explanation: Recursive virtualization is the ability to run a virtual machine manager on top of another virtual machine manager. This allows nesting hypervisors as long as the capacity of the underlying resources can accommodate that. Virtualizable hardware is a prerequisite to recursive virtualization.

- **Theorem 3:** A hybrid VMM may be constructed for any third-generation machine in which the set of user sensitive instructions are a subset of the set of privileged instructions.

  Explanation: The hybrid VMM consists of a software-based VMM (QEMU) and hardware-based VMM (KVM), and it dynamically switches between them.
  The software-based approach is preferred for implementing security mechanisms, whereas the hardware-based approach is preferred from the viewpoint of performance.
  Using the hybrid VMM, security- and reliability-critical software can be executed on the software-based VMM, and performance-critical software can be executed on the hardware-based VMM.
    - HVM are less effective than VMM
    - In the case of an HVM, more instructions are interpreted rather than being executed directly. All instructions in virtual supervisor (Kernel) mode are interpreted.
    - Whenever there is an attempt to execute a behaviour-sensitive or control-sensitive instruction, HVM controls the execution directly or gains the control via a trap.

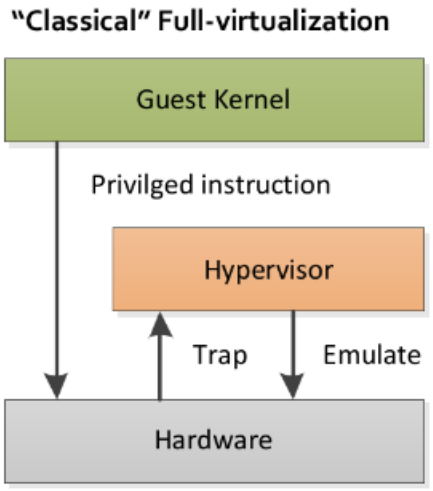
5. **Hardware – assisted Virtualization:**
    - This term refers to a scenario in which the hardware provides architectural support for building a virtual machine manager able to run a guest operating system in complete isolation.
    - This technique was originally introduced in the IBM System/370.
    - Hardware-assisted virtualization is also known as **accelerated virtualization.**
    - Hardware-assisted virtualization are meant to reduce the performance penalties experienced by emulating x86 hardware with hypervisors.

6. **Full Virtualization:**
   - When we emulate all the hardware of our host machine i.e., complete virtualization, then we call it **Full Virtualization**.

     OR

     In Full Virtualization, virtual machine managers are required to provide a complete emulation of the entire underlying hardware.

   - Full Virtualization is less secure than Paravirtualization.
   - In full virtualization, guest OS is completely isolated by the virtual machine from the virtualization layer and hardware.
   - Guest operating systems are unaware of each other.
   - In Full Virtualization, the guest OS doesn't know that it has been virtualized.
   - Hypervisor directly interact with the hardware such as CPU, disks.
   - Full virtualization is usually bit slower, because of complete emulation.
   - Full Virtualization is more portable and compatible.
   - In Full virtualization, virtual machine permits the execution of the instructions with running of **unmodified OS** in an entire isolated way.
   - VirtualBox, VMware Workstation (for 32-bit guests only), and Microsoft Virtual PC are examples of hypervisors which implements Full Virtualization.



7. **Paravirtualization:**
   - Some hardware is only virtualized to solve a real time problem or any particular problem is called **paravirtualization**.

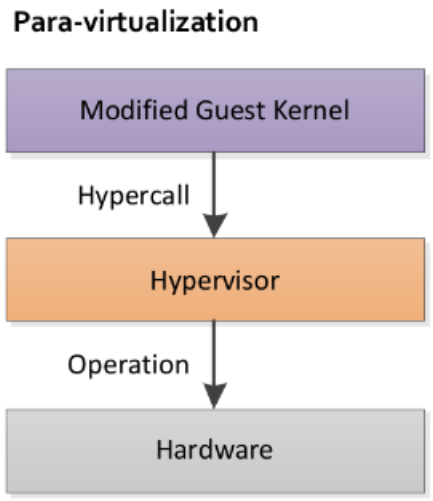
     OR

     Real-time, as per demand of problem, machine is partially virtualized, this is called **paravirtualization**.

   - In paravirtualization, guest OS is not completely isolated but it is partially isolated by the virtual machine from the virtualization layer and hardware.
   - Guest operating systems are aware of each other.
   - In Paravirtualization, virtual guest is aware that it has been virtualized.
   - Paravirtualization is more secure than the Full Virtualization.
   - Paravirtualization is faster in operation as compared to full virtualization.
   - Paravirtualization is less portable and compatible.
   - Here, **modified Guest OS** are used.

In Paravirtualization, a guest operating system (guest OS) is **modified** prior to installation inside a virtual machine (VM) in order to allow all guest OS within the system to share resources and successfully collaborate, rather than attempt to emulate an entire hardware environment.

- VMware and Xen are examples of hypervisors which implements Paravirtualization.



8. **<u>Partial Virtualization</u>:**
   - In Partial Virtualization, all underlying hardware would be virtualized but not in full capacity.