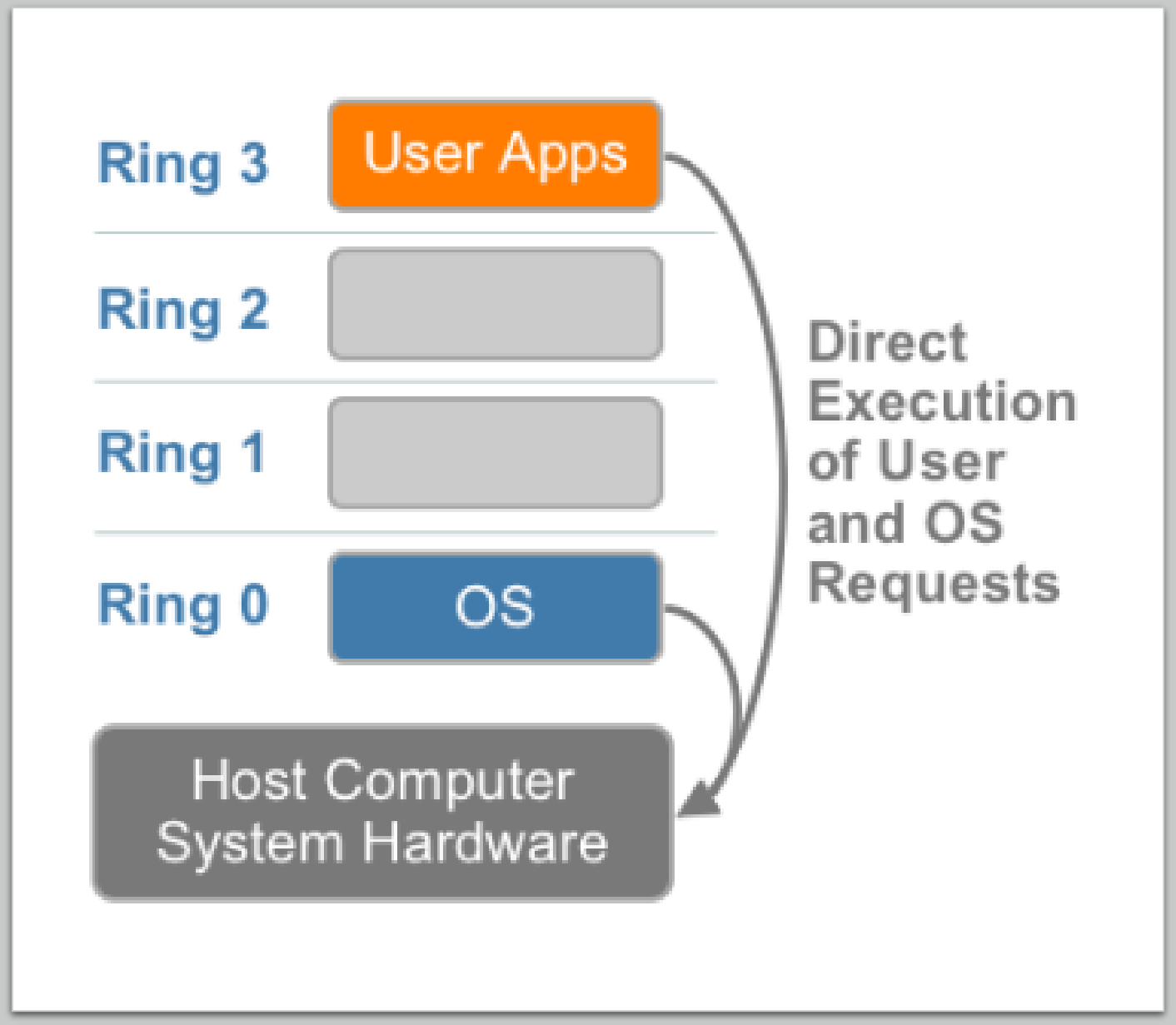
Cloud Computing Concepts

CS3132

Dr. Anand Kumar Mishra

NIIT University

x86 privilege level architecture without virtualization



Privileged instructions

- Privileged instructions - that are executed under specific restrictions and are mostly used for sensitive operations, which:
  - expose (behavior-sensitive)
    - operate on the I/O
  - modify (control-sensitive) the privileged state
    - alter the state of the CPU registers

Type 1 Hypervisor

- Theory: Type 1 virtualization is feasible if sensitive instruction is subset of privileged instructions or all sensitive instructions always cause a trap
- Reasoning:
  - On booting a Type 1 hypervisor, it runs in kernel mode
  - A Windows VM run on the hypervisor should not be trusted as much as the hypervisor and is therefore run in user mode
  - Windows assumes it is the kernel and can run sensitive instructions, but these sensitive instructions won't run because it will be running in user mode
  - The **solution** is that the hypervisor intervenes and runs each sensitive instruction attempted by the Windows VM
    - How will the hypervisor be alerted when Windows attempts so?
    - If the sensitive instruction causes a trap, the hypervisor intervenes and executes it for the VM

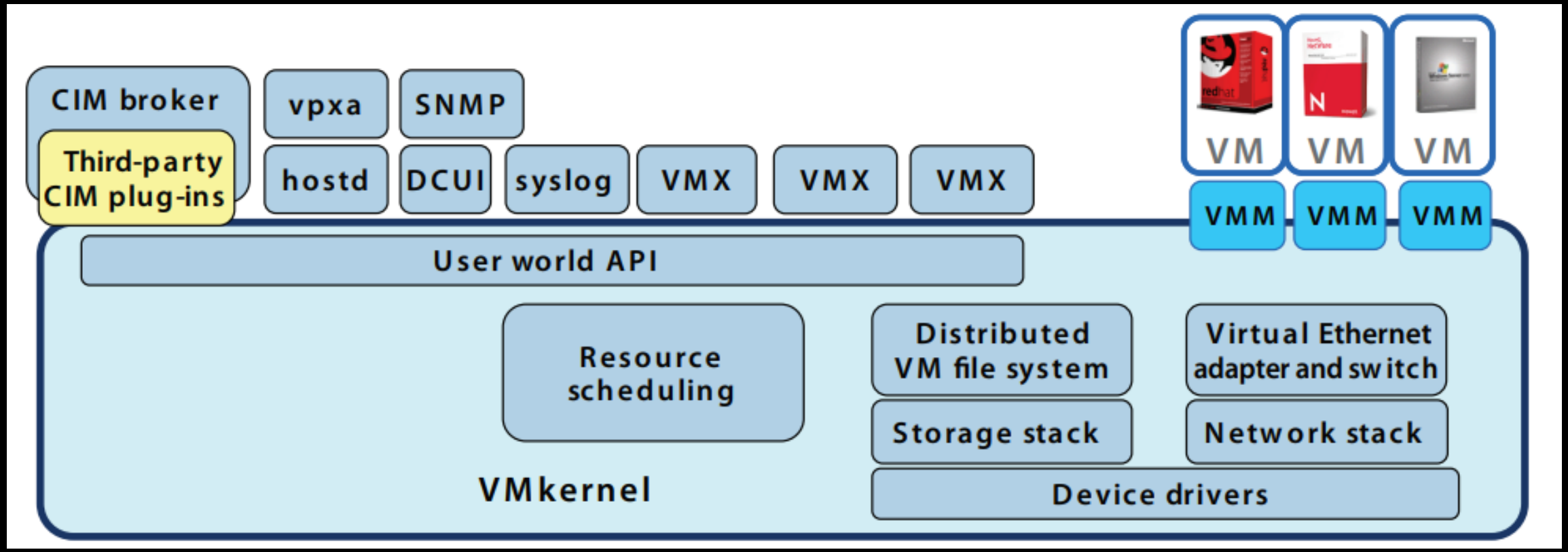
Type 1 Hypervisor

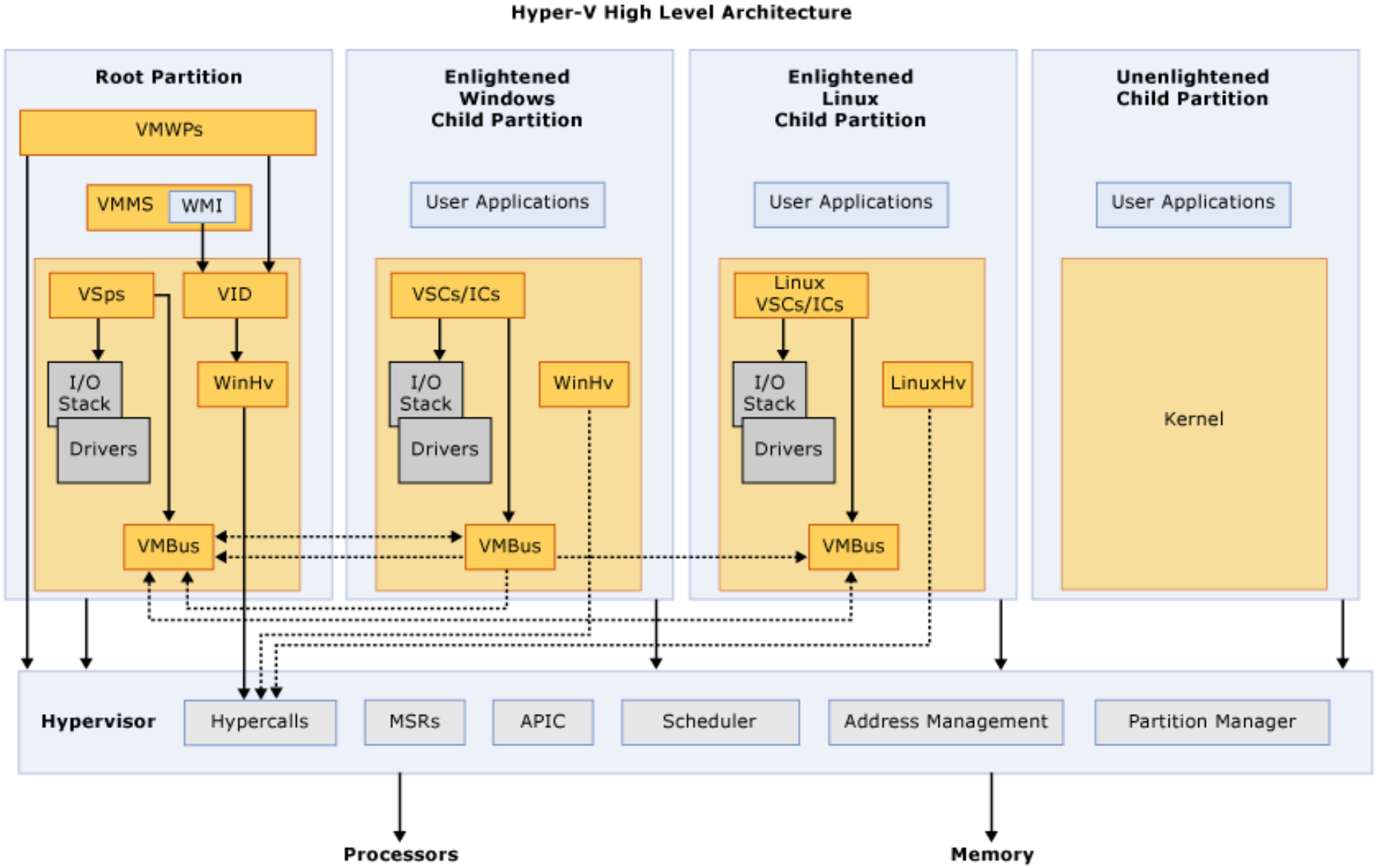
- NOTE:
  - Early Intel processors did not have Type 1 support
  - Recent Intel/AMD CPUs have hardware support, named Intel VT and AMD SVM
  - The idea is to create containers where a VM and guest can run and that hypervisor uses hardware bitmap to specify which instruction should trap, so that sensitive instruction in guest traps to hypervisor
  - This bitmap property can also be turned off

Type 1 Hypervisor - Examples

1. VMWare ESXI running, a specialized OS kernel that can run any arbritary VMs on it
2. Windows Hyper-V creates partitions, runs Windows in the parent partition (one copy of windows is mandatory), and the child partitions can run Linux or any other OS. Less flexible than ESXI.
3. Linux KVM or kernel virtual machine: Implemented as a device driver that gives barebone support for Type 1. Along with some other components (like QEMU) gives the functionality for Type 1 hypervisor.

The streamlined architecture of VMware ESXi eliminates the need for a service console




Hyper-V High Level Architecture
Root Partition
VMWPs
VMMS
WMI
VSps
VID
I/O Stack
Drivers
WinHv
VMBus
Enlightened Windows Child Partition
User Applications
VSCs/ICs
I/O Stack
Drivers
WinHv
VMBus
Enlightened Linux Child Partition
User Applications
Linux VSCs/ICs
I/O Stack
Drivers
LinuxHv
VMBus
Unenlightened Child Partition
User Applications
Kernel
Hypervisor
Hypercalls
MSRs
APIC
Scheduler
Address Management
Partition Manager
Processors
Memory

At its core, Hyper-V is a Type 1 hypervisor.

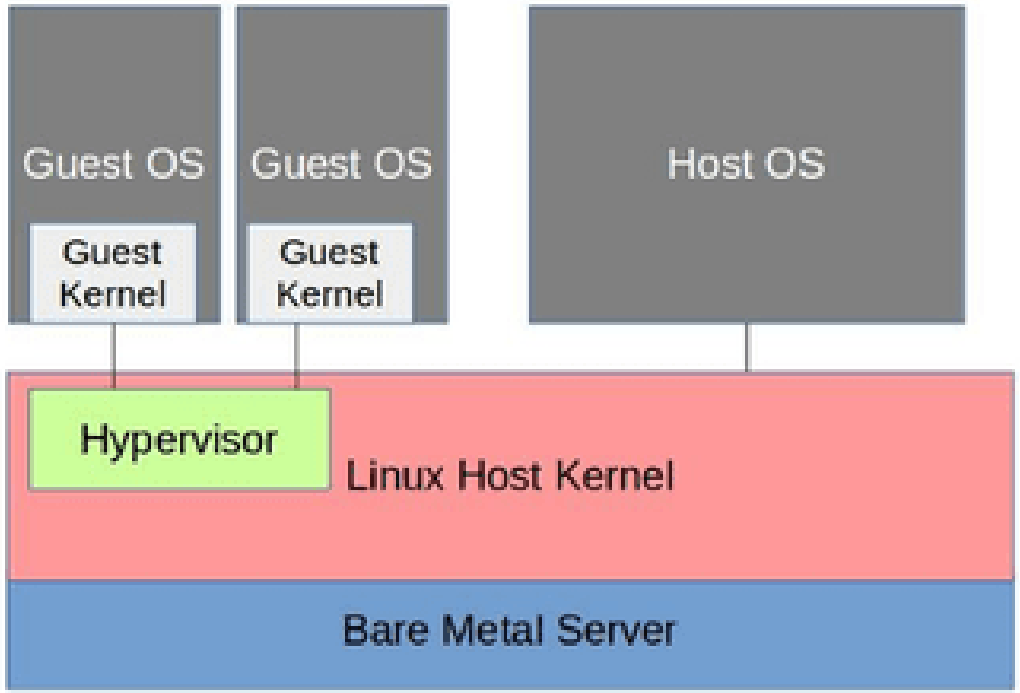
It virtualizes memory and processors and manages how the root partition manages virtual machines.

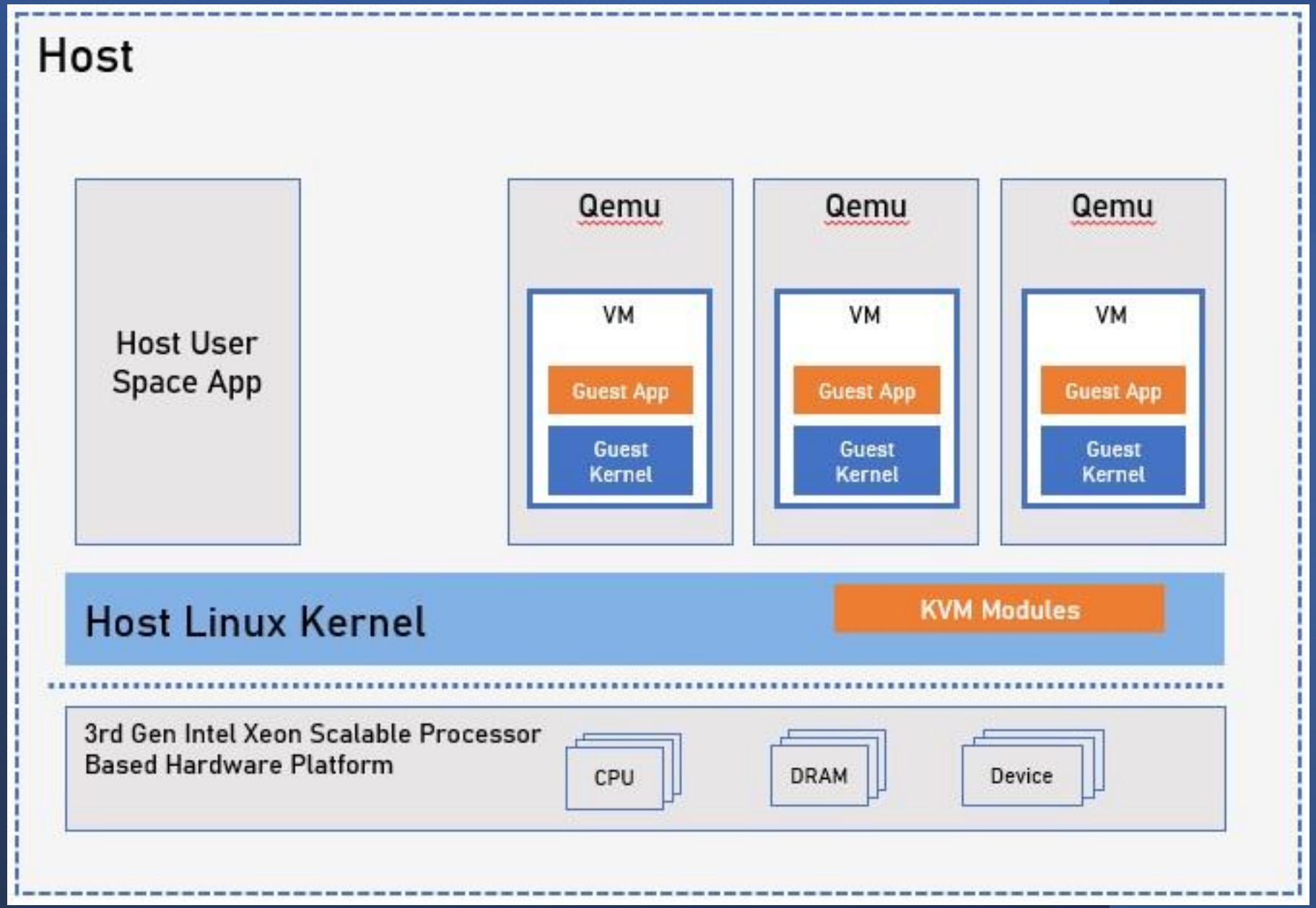
Its root partition directly accesses physical I/O devices.

The hypervisor provides a virtualization stack that includes virtualized I/O devices, management APIs, and a virtual machine memory manager.

KVM Hypervisor

Architecture




Host
Host User Space App
Qemu
VM
Guest App
Guest Kernel
Qemu
VM
Guest App
Guest Kernel
Qemu
VM
Guest App
Guest Kernel
Host Linux Kernel
KVM Modules
3rd Gen Intel Xeon Scalable Processor Based Hardware Platform
CPU
DRAM
Device

Type 2 Hypervisor

- A Type 2 hypervisor runs as an application on the host OS and therefore does not have kernel level privileges
- Type 2 hypervisor performs dynamic code translation
  - It scans instructions executed by the guest OS**, replacing the sensitive instructions with function calls**
  - These **function calls in turn make system calls** to the OS,
    - thereby involving the host OS
  - This process is called **binary translation**
- This leads to slowdown as every piece of sensitive code has to be translated
- Therefore, VT support is not needed, no support is needed from the hardware to ensure that all sensitive instructions are privileged

Type 2 Hypervisor - Example

- VMware Fusion delivers the best way to run Windows, Linux and more on Apple Macs without rebooting
- Fusion 13 supports Intel and Apple Silicon Macs running macOS 12 and newer, and includes features for developers, IT admins and everyday users
- VMWare Fusion, upon loading program, scans code for basic blocks and replaces sensitive instructions by VMWare procedure using binary translation
- Only the guess OS's instructions need to be scanned, not the applications

VMWare

- VMware ESXi - bare-metal hypervisor that installs directly onto your physical server
- VMware vSphere - Virtualize servers to manage your IT infrastructure; allowing you to consolidate your applications
- VMware Fusion - to run Windows, Linux, containers, Kubernetes and more in virtual machines (VMs) without rebooting
  - With Fusion Player and Fusion Pro, run nearly any OS as VMs on Mac
- VMware Workstation Pro - Run Windows, Linux and BSD virtual machines on a Windows or Linux desktop with VMware Workstation Pro, the industry standard desktop hypervisor
- VMware Workstation Player - Easily run multiple operating systems as virtual machines on your Windows or Linux PC with VMware Workstation Player

Virtual Machine Manager

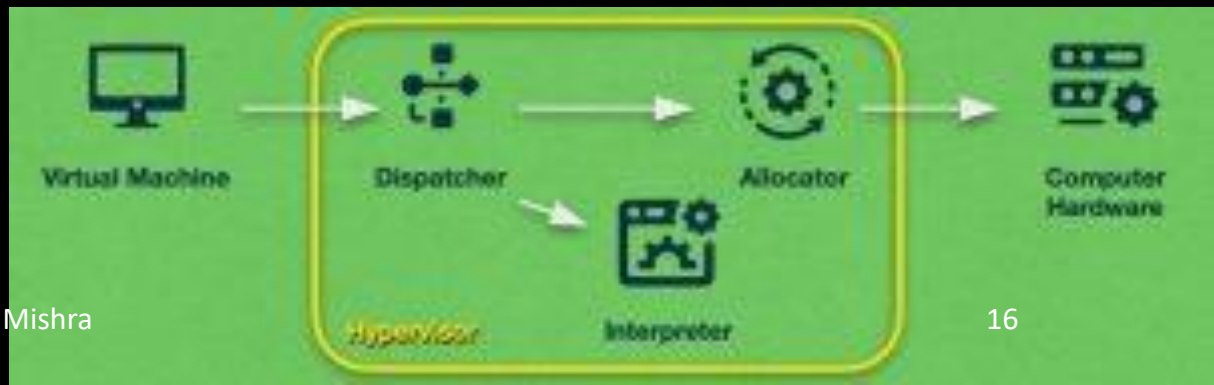
- Three main modules:
  - dispatcher,
  - allocator, and
  - interpreter,
    - coordinate their activity in order to emulate the underlying hardware

VMM - dispatcher

- The dispatcher constitutes the entry point of the monitor and reroutes the instructions issued by the virtual machine instance to one of the two other modules

OR

- The dispatcher is the component that receives the instructions sent from the virtual machine. It does not carry out these instructions, but instead, it just forwards them to one of the other two modules

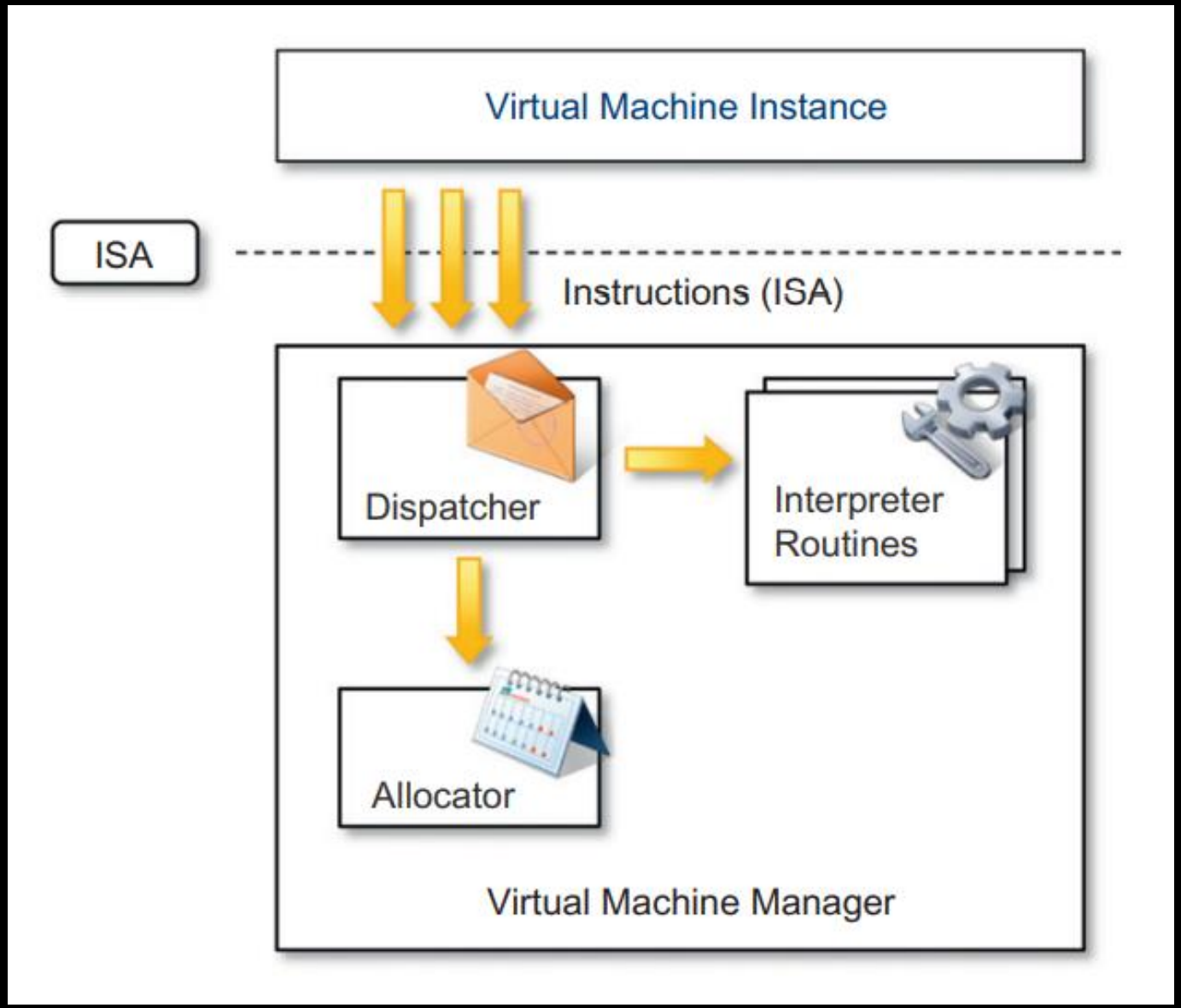


VMM - allocator

- The allocator is responsible for deciding the system resources to be provided to the VM:
  - whenever a virtual machine tries to execute an instruction that results in changing the machine resources associated with that VM, the allocator is invoked by the dispatcher
- The allocator responds to the dispatcher's commands to determine the resources needed and allocates them
- The interpreter module has stored routines that are executed based on the allocator's commands.

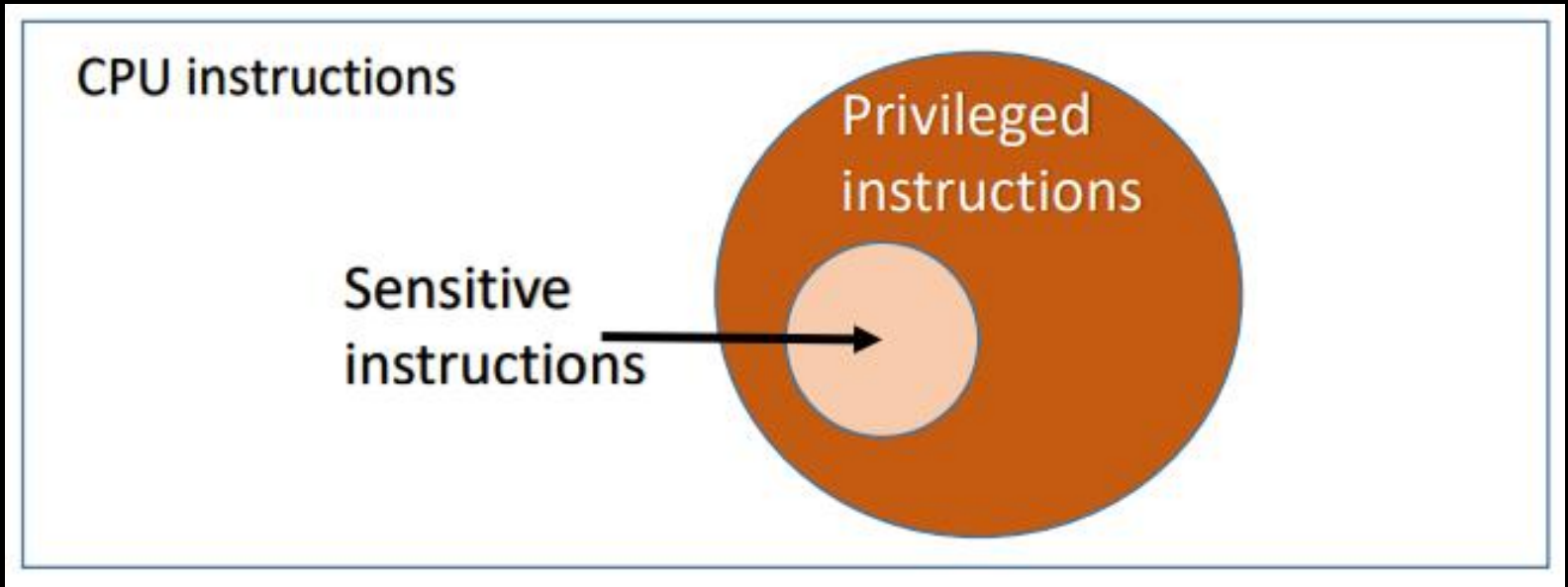
VMM - interpreter

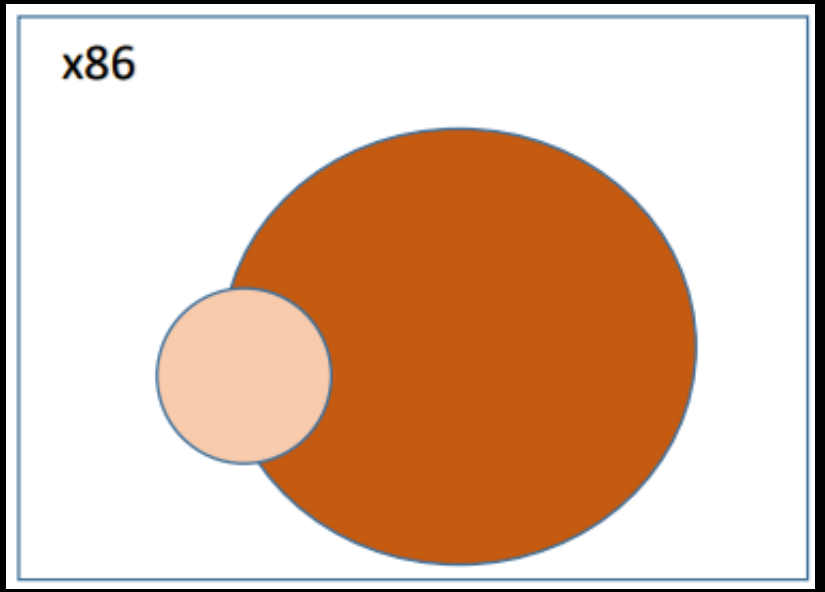
- **Allocator: allocates system resources**
- **Interpreter: instructions that are executed**
- The interpreter module consists of interpreter routines
- These are executed whenever a virtual machine executes a privileged instruction:
  - a trap is triggered and the corresponding routine is executed


Virtual Machine Instance
ISA
Instructions (ISA)
Dispatcher
Interpreter Routines
Allocator
Virtual Machine Manager

Formal Requirements for Virtualizable Third Generation Architectures

- Research Paper by –
  - Gerald J. Popek University of California, Los Angeles
  - Robert P. Goldberg Honeywell Information Systems and Harvard University
- THEOREM 1.
  - For any conventional third generation computer, a virtual machine monitor may be constructed if **the set of sensitive instructions for that computer is a subset of the set of privileged instructions**.
- THEOREM 2.
  - A conventional third generation computer is" recursively virtualizable if it is: (a) virtualizable, and (b) a VMM without any timing dependencies can be constructed for it.
- THEOREM 3.
  - A hybrid virtual machine monitor may be constructed for any conventional third generation machine in which the set of user sensitive instructions are a subset of the set of privileged instructions.


CPU instructions
Privileged instructions
Sensitive instructions


x86

Complicated Situation in x86 Virtualization

- Some sensitive instructions can’t effectively be virtualized as they have different semantics when they are not executed in **Ring 0**
- Challenge: Difficulty in trapping and translating these sensitive and privileged instruction requests **at runtime**
  - This made x86 architecture virtualization look impossible

Major challenges to building a VMM for the x86 architecture

1. The x86 architecture was not virtualizable
2. The x86 architecture was of daunting (Discouraging) complexity
   - The x86 architecture was a big CISC (Complex Instruction Set) architecture
     - CISC Characteristics
       - Complex instruction, hence complex instruction decoding
       - Complex Addressing Modes

Major challenges to building a VMM for the x86 architecture

3. x86 machines had diverse peripherals
   - vendor-specific device drivers, virtualizing all these peripherals was intractable
4. Need for a simple user experience
   - It was required to add VMM to existing systems
   - Focus to consider software delivery options
   - A user experience that encouraged simple user adoption

Three core attributes of a virtual machine to x86-based target platform (adopted by VMware)

- Compatibility:
  - The notion of an essentially identical environment meant that any x86 operating system, and all of its applications, would be able to run without modifications as a virtual machine
  - A VMM needed to provide sufficient compatibility at the hardware level such that users could run whichever operating system, down to the update and patch version, they wished to install within a particular virtual machine, without restrictions

Three core attributes of a virtual machine to x86-based target platform (adopted by VMware)

- Performance
  - Minor decreases in speed - meant sufficiently low VMM overheads that users could use a virtual machine as their primary work environment
  - Aim: To run relevant workloads at near native speeds, and
    - in the worst case: To run them on then-current processor with the same performance as if they were running natively on the immediately prior generation of processors

Three core attributes of a virtual machine to x86-based target platform (adopted by VMware)

- Isolation
  - A VMM had to guarantee the isolation of the virtual machine without making any assumptions about the software running inside
  - A VMM needed to be in complete control of resources
  - Software running inside virtual machines had to be prevented from any access that would allow it to modify or subvert its VMM
  - A VMM had to ensure the privacy of all data not belonging to the virtual machine
  - A VMM had to assume that the guest operating system could be infected with unknown, malicious code (a much bigger concern today than during the mainframe era)

- VM introduces in the 60's IBM/370 models
- 1979 : chroot (version 7 Unix)
- 1982 : chroot (BSD)
  - chroot environment allows to create a separate file system
- 1999 : VMware (virtualization for x86)
- 2000 : FreeBSD Jail (*BSD)
- 2003 : Xen (Linux)
- 2005 : Solaris Zones
- 2005/2006 : Intel-VT et AMDV (hardware-assisted virtualization)
- 2006 : OpenVZ (Linux)
- 2007 : KVM (Linux)
- 2008 : LxC (Linux)
- 2008 : HyperV (Microsoft)