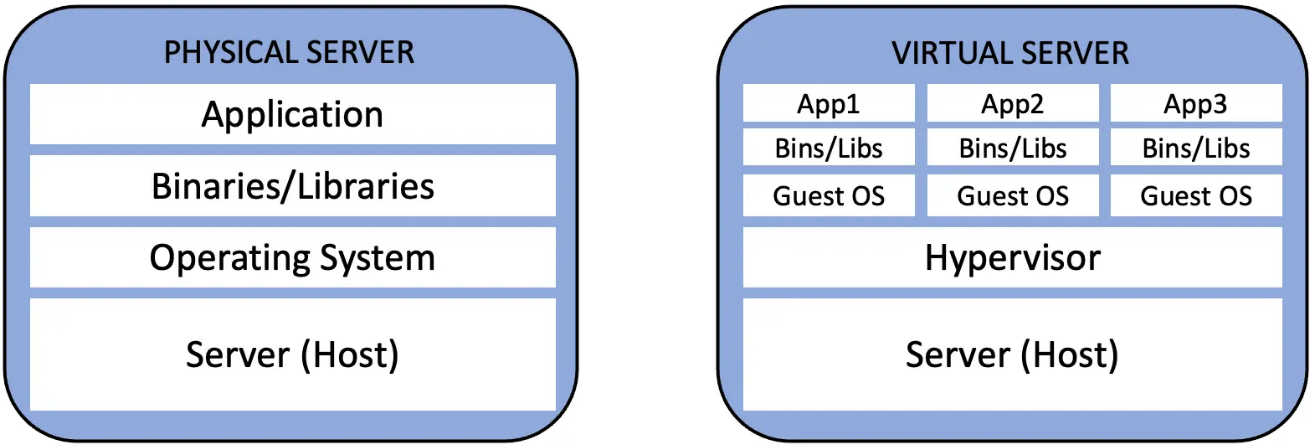
Virtualization

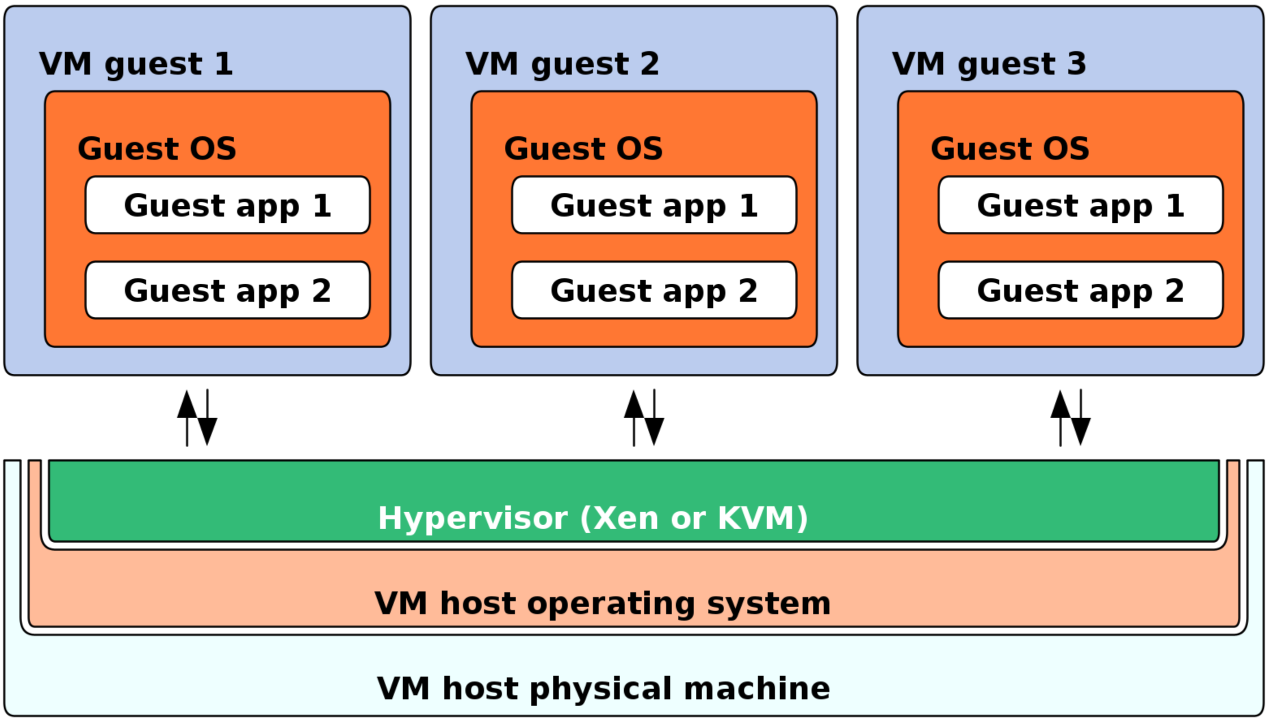
Introduction to Virtualization

what is a Virtual Machine (VM)?

- A **virtual machine** (VM) is software that runs programs or applications without being tied to a physical machine. In a VM instance, one or more guest machines can run on a host computer.
- Each VM has its own operating system, and functions separately from other VMs, even if they are located on the same physical host.
- Multiple VMs can share resources from a physical host, including CPU cycles, network bandwidth and memory.
- Virtualization converts physical servers into logical folders or files. These folders or files can be divided into two parts: those that store VM configuration information, and those that store user data.
- Without virtualization, running multiple primary application programs in the same operating system of a physical server may cause runtime conflicts and performance bottlenecks.
- Running only one application on a dedicated server could solve these problems but will easily cause low resource utilization.
- With virtualization, multiple VMs can run on a single physical server, and each VM can run an independent OS. This improves resource utilization.



Physical VS Virtual



Virtualization Schema

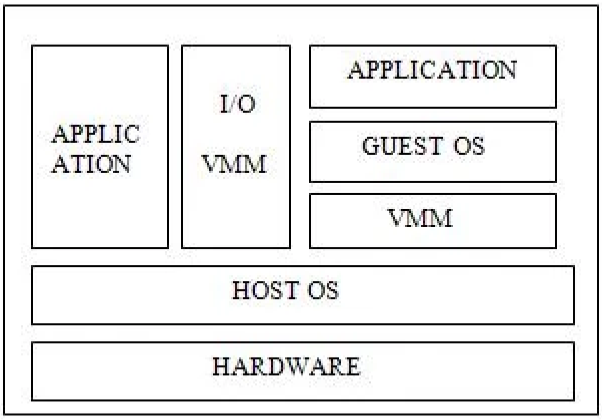


Figure 3: Type 3 VMM

Virtualization Schema

Important Concepts in Compute Virtualization

- **Guest OS**:
  - Operating system running in a virtual machine (VM)
- **Guest Machine**:
  - Virtual machine created through virtualization
- **Hypervisor**:
  - Virtualization software layer, or Virtual Machine Monitor (VMM)
- **Host OS**:
  - Operating system running in a physical machine
- **Host Machine**:
  - Physical machine

Types of Compute Virtualization

- A **Type 1 hypervisor** (bare-metal hypervisor):
  - This type of hypervisor has direct access to hardware resources and does not need to access the host OS. The hypervisor can be seen as a customized host OS, which merely functions as VMM and does not run other applications.
  - In Type 1 virtualization, the hypervisor is dedicated to converting host resources into virtual resources for the guest OS to use. The guest OS runs as a process on the host. Therefore, such hypervisors are called bare-metal hypervisors.
  - Type 1 hypervisors have the following advantages and disadvantages:
    1. *Advantages*: VMs can run different types of guest OSs and applications independent of the host OS.
    2. *Disadvantages*: The kernel of the virtualization layer is hard to develop.
  - The virtualization products that use Type 1 hypervisors include **VMWare ESX Server**, **Citrix XenServer**, and **FusionCompute**.
- A **Type 2 hypervisor** is also called a hosted hypervisor:
  - Physical resources are managed by the host OS (for example, ~~Windows~~ or Linux).
  - VMM provides virtualization services and functions as a common application in the underlying OS (for example, ~~Windows~~ or Linux).
  - VMs can be created using VMM to share underlying server resources. VMM obtains resources by calling the host OS services to virtualize the CPUs, memory, and I/O devices.
  - After a VM is created, VMM usually schedules the VM as a process of the host OS.
  - Unlike a Type 1 hypervisor, a Type 2 hypervisor is only a program in the host OS. All hardware resources are managed by the host OS.
  - Type 2 hypervisors have the following advantages and disadvantages:
    1. *Advantages*: They are easy to implement.
    2. *Disadvantages*: Since a Type 2 hypervisor shares CPU, RAM, storage, and network bandwidth from the underlying physical infrastructure with a host OS, the amount of resources a Type 2 hypervisor has access to is limited compared to that of a Type 1. The performance overheads are high.
  - The virtualization products that use Type 2 hypervisors include **VMware Workstation**, **Virtual Box**, and **Qemu**.

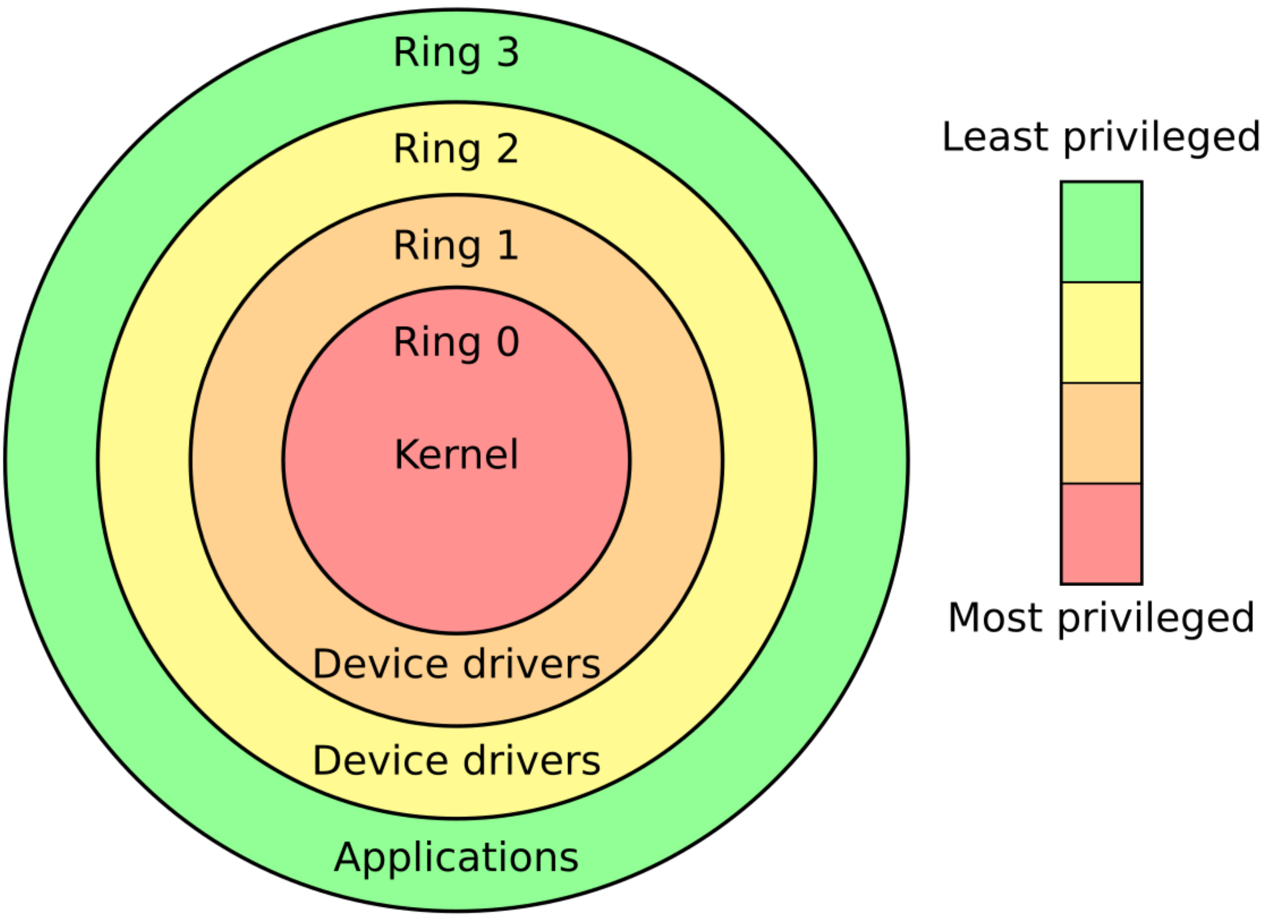
Characteristics of Virtualization

- **Partitioning**: indicates the VMM capability of allocating server resources to multiple VMs. Each VM can run an independent OS, so that multiple applications can coexist on one server.
- **Isolation**: Multiple VMs created in a partition are logically isolated from each other.
- **Encapsulation**: Each VM is saved as a group of hardware-independent files, including the hardware configuration, BIOS configuration, memory status, disk status, and CPU status. You can copy, save, and move a VM by copying only a few files.
- **Hardware independence**: The migration can be successful as long as the same VMM running on the target host as that on the source host, regardless of the underlying hardware specifications and

configuration

CPUs hierarchical protection domains

- There is **hierarchical protection domains** of CPUs, often called protection rings.
  - There are four rings: **Ring 0**, **Ring 1**, **Ring 2**, **and Ring 3**, which is a hierarchy of control from the most to least privilege.
  - **Ring 0** has direct access to the hardware. Generally, only the OS and driver have this privilege.
  - **Ring3** has the least privilege. All programs have the privilege of Ring 3.
  - To **protect** the computer, some dangerous instructions can only be executed by the OS, preventing malicious software from randomly calling hardware resources.
- The OS on a common host sends two types of instructions: **privileged instructions** and **common instructions**.
  - **Privileged instructions** are instructions used to manipulate and manage key system resources. These instructions can be executed by programs of the highest privilege level, that is, Ring 0.
  - **Common instructions** can be executed by programs of the common privilege level, that is, Ring 3.



CPU Virtualization

- Thereis two types of CPU Virtualization, **Full virtualization** and **Paravirtualization**.
- **Full virtualization**:

- All OS requests sent by **VMs** are forwarded to **VMM**, and **VMM** performs binary translation on the requests. When **VMM** detects privileged or sensitive instructions, the requests are trapped into **VMM** for emulation.
- Then, the requests are scheduled to the **CPU privilege level** for execution. When **VMM** detects program instructions, the instructions are executed at the **CPU non-privilege level**.
- This technique is called **full virtualization** because all request instructions sent by **VMs** need to be filtered.
- **Full virtualization** was first proposed and implemented by **VMware**.
- Full virtualization **disadvantages**:
  - Modifying the guest OS binary code during running causes **large performance loss** and increases the VMM development **complexity**.

- **Paravirtualization**:
  - If the guest OS can be modified to be able to **aware** that it is virtualized, the VM OS uses the **Hypercall** to replace sensitive instructions in the virtualization with the hypervisor layer to implement virtualization.
  - **Non-sensitive instructions** such as privileged and program instructions are directly executed at the **CPU non-privilege level**.
  - **Xen** developed the paravirtualization technique, which compensates for the disadvantages of full virtualization.
  - Paravirtualization **advantages**:
    - **Multiple** types of guest OSs can run at the same time. Paravirtualization delivers **performance** similar to that of the original non-virtualized system.
  - Paravirtualization **disadvantages**:
    - The host OS can be modified only for **open-source** systems, such as **Linux**.
    - ~~Non-open-source~~ systems, such as ~~Windows~~, do not support **paravirtualization**. In addition, the modified guest OS has **poor portability**.

Memory Virtualization

- **Memory virtualization** is a process of centrally managing the physical memory of a physical machine and aggregating the physical memory into a virtualized memory pool available to VMs.
- Memory virtualization creates a **new layer** of address spaces, that is, the address spaces of VMs. The **VMs** are made to believe that they run in a real **physical** address space when in fact their access requests are relayed by **VMM**.
- Memory virtualization involves the translation of **three types** of memory addresses:
  - VM memory address (VA).
  - physical memory address (PA).
  - machine memory address (MA).

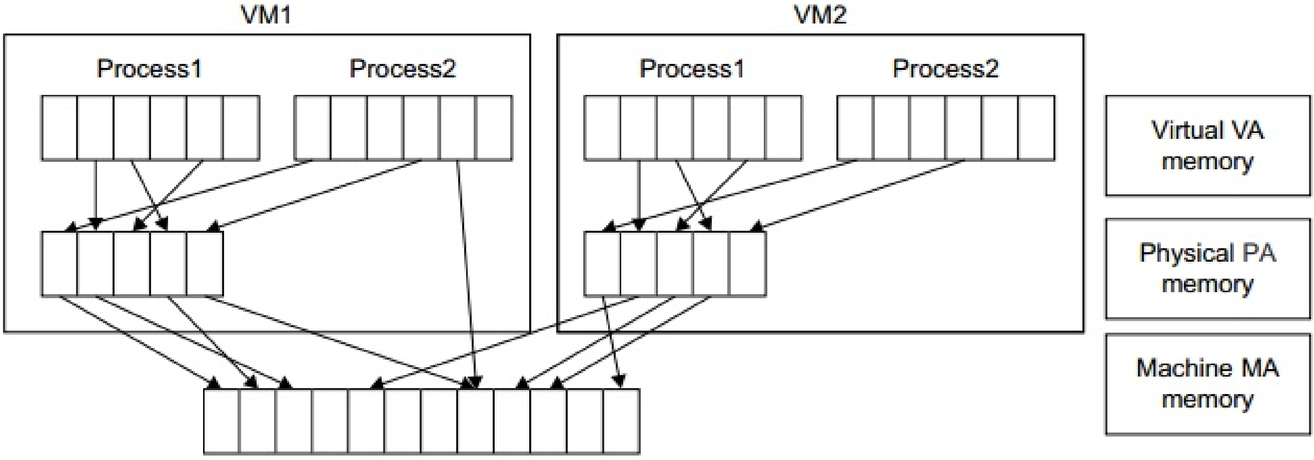
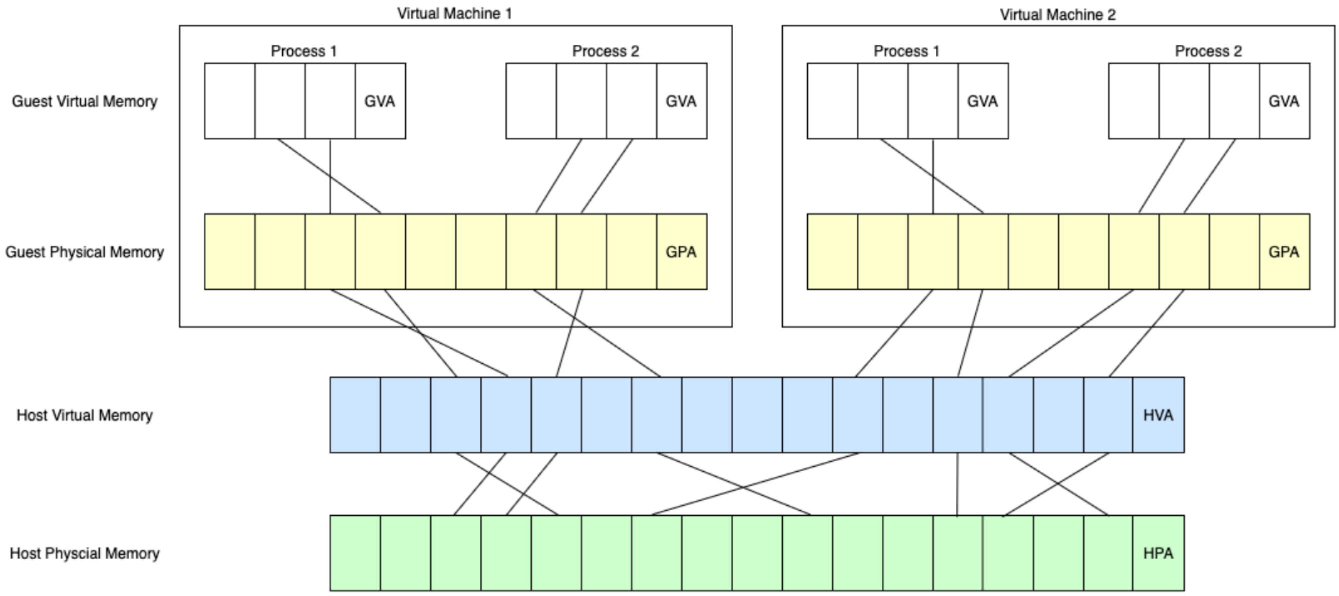


FIGURE 3.12

Two-level memory mapping procedure.



- The following direct address translation path must be supported so that multiple **VMs** can run a physical host:
    - VA (virtual memory) → PA (physical memory) → MA (machine memory).
- The **VM** OS controls the mapping from the virtual address to the physical address of the customer memory (VA → PA).
- However, the **VM** OS cannot directly access the machine memory. Therefore, the **hypervisor** needs to map the physical memory to the machine memory (PA → MA).

I/O Virtualization

With compute virtualization, a large number of VMs can be created on a single host, and these VMs all need to access the I/O devices of this host. However, I/O devices are limited.

- **I/O** device sharing among multiple VMs requires **VMM**. VMM **intercepts access requests** from VMs to I/O devices, **simulates** I/O devices using software, and **responds** to I/O requests. This way, multiple VMs can access I/O resources concurrently.

- **Emulation [Full Virtualization]**:
  - When a VM initiates an I/O request, VMM **intercepts** the request sent by the VM, and then **sends the real access request to the physical device for processing**.
  - No matter which type of OS is used by the VM, the OS does not need to be modified for I/O virtualization.
  - Real-time monitoring and emulation are implemented by software programs on the CPU, which causes **severe performance loss to the server**.
  - Full Virtualization **advantages**:
    - Best option for correctness and abstraction
  - Full Virtualization **disadvantages**:
    - High performance cost
- **Paravirtualization**:
  - Paravirtualization requires each VM to run a **frontend driver**. When VMs need to access an I/O device, the VMs send I/O requests to the **privileged VM** (act as hypervisor) through the **frontend driver**, and the **backend driver** of the privileged VM collects the I/O request sent by each VM.
  - Access to hardware drivers is transferred from the **I/O frontend** to the **I/O backend**. This mode is usually only used for **hard disks** and **Network Interface Cards (NICs)** and delivers **high performance**.
  - This reduces the performance loss of VMM and therefore delivers better I/O performance... However, the VM OS needs to be **modified** (usually Linux). Specifically, the I/O request processing method of the OS needs to be changed so that all the I/O requests can be sent to the privileged VM for processing.
  - Paravirtualization **advantages**:
    - Optimize driver and virtual device interaction
    - Guest is “aware” of virtualization
  - Paravirtualization **disadvantages**:
    - Guest needs to be modified
- **IO-through (Hardware-assisted virtualization)**:
  - Hardware-assisted virtualization directly installs the I/O device driver in the VM OS without any change to the OS.
  - the time required for a VM to access the I/O hardware is **the same** as that for a traditional PC to access the I/O hardware.
  - hardware-assisted virtualization **outperforms** other methods in terms of I/O performance. However, hardware-assisted virtualization requires special hardware support.
  - IO-through **advantages**:
    - Best option for performance
  - IO-through **disadvantages**:
    - Each device is limited to use by one VM
    - Strong coupling with hardware

Mainstream Compute Virtualization Technologies

Introduction to KVM

KVM Architecture

- **Kernel-based Virtual Machine** (KVM) is a **Type-II full virtualization** solution.
- It is a Linux kernel module. A physical machine with a Linux kernel module installed can function as a hypervisor, which does not affect the other applications running on the Linux OS.
- Each VM is one or more processes. You can run the kill command to kill the processes.
- The KVM **kernel module** is the core of a KVM VM. This module initializes the **CPU hardware**, enables the **virtualization** mode, runs the guest machine in the VM mode, and supports the running of the virtual client.
- However, a VM requires other **I/O devices** such as Network Interface Cards (NICs) and hard disks besides CPUs and memory. **QEMU** is required to implement other virtualization functions.
- **QEMU** is interacting with hardware. This means that all interactions with the hardware need to pass through QEMU.
- Therefore, the simulation performance delivered by QEMU is **low**. QMEU is able to **simulate** CPUs and memory. In KVM, only QEMU is used to simulate I/O devices.
- Therefore, the KVM kernel module and QEMU form a complete **virtualization technology**.

KVM I/O Process - Default

- In steps 2, 3, and 7 (Default) , **KVM** does not make any modification on the I/O operation except for capturing the request and sending the notification.
- The **Virtio technology** was developed to simply this procedure.

KVM I/O Process - Virtio

- The I/O operation request is not captured by the I/O capture program. Instead, the request is stored in the ring buffer between the frontend and backend drivers. At the same time, the KVM module notifies the backend driver.
- **Virtio** advantages:
  - Saves the hardware resources required for QEMU emulation.
  - Reduces the number of I/O request paths and improves the performance of virtualization devices.
- **Virtio** disadvantages:
  - some old or uncommon devices cannot use Virtio but can only use QEMU.

Qemu

Install Qemu:

- Debian/Ubuntu:

```
sudo apt install qemu-system
```

- Arch:

```
pacman -S qemu
```

- bulid it from the source:

```
git clone https://gitlab.com/qemu-project/qemu.git
cd qemu
./configure
make
```

Disk Image:

- To make a Disk Image, we will use a `qemu-img` Command:

```
qemu-img [standard options] command [command options]
```

- The `[standard options]` are:
  - ```
    -h, --help
    ```
  - Display this help and exit

  - ```
    -V, --version
    ```
  - Display version information and exit

- The `command` are:
  - **Create**
    - ```
      #Syntax and Options
      create [-q] [-f FMT] FILENAME [SIZE]
      ```
      ```

- Create the new disk image FILENAME of size SIZE and format FMT. Depending on the file format, you can add one or more OPTIONS that enable additional features of this format.

- **command options**:

command option	Meaning
`-q`	Runs the command in quiet mode (suppresses output).
`-f FMT`	Specifies the format of the disk image.
`FILENAME`	The name of the disk image file to create.
`SIZE`	The size of the image. Specify size in bytes or use suffixes like `K`, `M`, `G`, `T` (e.g., 10G for 10 gigabytes).

formats	Meaning
`raw`	Plain binary image.
`qcow2`	QEMU Copy-On-Write version 2 (most commonly used for its features like compression and snapshots).
`vmdk`	~~VMware~~ disk format.
`vdi`	VirtualBox disk format.
`vpc`	VirtualPC disk format.s
`qed`	QEMU Enhanced Disk format.

- **exmples**:

```
qemu-img create -f raw Pop_OS_img.img 200M
ls -l
# output ->
# ...
# Pop_OS_img.img
# ...
```

- **info**

  - ```
    info [-f FMT] [--output=OFMT] FILENAME
    ```
  - Give information about the disk image FILENAME
  - **command options**:

command option	Meaning
    ```

command option	Meaning
`-f FMT`	Specifies the format of the disk image.
`--output=json`	Displays the information in JSON format.
`FILENAME`	The name of the disk image file to analyze.

- **exmples**:

```
qemu-img info Pop_OS_img.img
ls -l
# output ->
# image: Pop_OS_img.img
# file format: raw
# virtual size: 200 MiB (209715200 bytes)
# disk size: 4 KiB
```

- **convert**

  - ```
    convert [-q] [-f FMT] [-O OUTPUT_FMT] FILENAME [FILENAME2 [...]] OUTPUT_FILENAME
    ```
  - Convert the disk image FILENAME to disk image OUTPUT_FILENAME using format OUTPUT_FMT.
  - **command options**:

command option	Meaning
`-f FMT`	Specifies the format of the source image file.
`-O OUTPUT_FMT`	Specifies the format of the target image file.
`FILENAME [FILENAME2 [...]]`	The input image file to be converted.
`OUTPUT_FILENAME`	The name of the output image file.

  - **exmples**:

```
qemu-img convert -f raw -O qcow2 Pop_OS_img.raw Pop_OS_img.qcow2
ls -l
# output ->
# ...
# Pop_OS_img.qcow2
# Pop_OS_img.raw
# ...
qemu-img info Pop_OS_img.qcow2
# image: Pop_OS_img.qcow2
# file format: qcow2
```
```

```
# virtual size: 200 MiB (209715200 bytes)
# disk size: 196 KiB
# cluster_size: 65536
# Format specific information:
#   compat: 1.1
#   compression type: zlib
#   lazy refcounts: false
#   refcount bits: 16
#   corrupt: false
#   extended l2: false
```

- **resize**
  - ```
    resize [-f FMT] [--shrink] FILENAME [+ | -]SIZE
    ```
  - Change the disk image as if it had been created with SIZE.
  - **command options**:

command option	Meaning
`-f FMT`	Specifies the format of the image file being resized.
`--shrink`	Allows shrinking the image to a smaller size.
`FILENAME`	The name of the disk image to resize.
`SIZE`	The new size for the disk image. This can be specified in bytes or with suffixes like `K`, `M`, `G`, or `T`.
`[+ | -]`	Relative Size: Adjust the size by adding or subtracting a value.

  - **exmples**:

```
qemu-img resize -f raw Pop_OS_img.img 300M
qemu-img info Pop_OS_img.img
# image: Pop_OS_img.img
# file format: raw
# virtual size: 300 MiB (314572800 bytes)
# disk size: 4 KiB
qemu-img resize -f raw --shrink Pop_OS_img.img 200M
qemu-img info Pop_OS_img.img
# image: Pop_OS_img.img
# file format: raw
# virtual size: 200 MiB (209715200 bytes)
# disk size: 4 KiB
qemu-img resize -f raw Pop_OS_img.img +200M
qemu-img info Pop_OS_img.img
# image: Pop_OS_img.img
# file format: raw
```
```

```
   # virtual size: 400 MiB (419430400 bytes)
   # disk size: 4 KiB
   qemu-img resize -f raw Pop_OS_img.img -200M
   # qemu-img: Use the --shrink option to perform a shrink
operation.
   # qemu-img: warning: Shrinking an image will delete all data
beyond the shrunken image's end. Before performing such an
operation, make sure there is no important data there.
   qemu-img resize -f raw --shrink Pop_OS_img.img -200M
   # image: Pop_OS_img.img
   # file format: raw
   # virtual size: 200 MiB (209715200 bytes)
   # disk size: 4 KiB
```

Qemu Systems:

- **Qemu** has multiple of systems from different Architectures:
- This is the most interesting thing about **Qemu**.
- to make a Emulator with **Qemu**:

```
qemu-system-x86_64 [machine opts] \
                [cpu opts] \
                [smp opts] \
                [drive opts]\
                [device opts] \
                [cdrom FILENAME] \
                [net opts]\
                [display opts]\
                [vga opts]\
                [boot opts]\
                [snapshot]\
                [monitor opts]\
                [usb opts]\
                [audiodev opts]\
                [rtc opts]
```

- **Options**:
  - **machine**:

Machine Type	Meaning
pc	Emulates the standard PC (i440FX chipset). Suitable for most 64-bit x86 systems. Provides legacy PCI and IDE support.

Machine Type	Meaning
`q35`	Emulates a modern PC with a Q35 chipset. Uses PCI Express (PCIe) instead of legacy PCI. Recommended for newer operating systems and hardware configurations.
`isapc`	Emulates an older ISA-only PC. Useful for running older software or operating systems.
`virt`	Emulates a paravirtualized machine for x86_64. Used for lightweight VMs or virtualization-optimized environments.
`xenfv`	Emulates a Xen-compatible fully virtualized machine. Suitable for running under Xen hypervisor environments.

- (**Extra**) additional Options for `machine opts`:

Option	Meaning
accel=	Specifies the accelerator backend: `kvm`: Use KVM acceleration. `tcg`: Use Tiny Code Generator (no hardware acceleration).
kernel_irqchip=<on|off>	Enables or disables in-kernel IRQ chip emulation (useful for performance tuning with KVM).
mem-merge=<on|off>	Enables or disables memory merging.
usb=<on|off>	Enables or disables USB support.
dump-guest-core=<on|off>	Controls whether to allow dumping guest core on VM crash.
smm=<on|off>	Enables or disables System Management Mode (SMM).

- **exmples**:

```
qemu-system-x86_64 -machine type=pc
qemu-system-x86_64 -machine type=q35
qemu-system-x86_64 -machine type=pc,accel=kvm
qemu-system-x86_64 -machine type=pc,usb=off,accel=kvm
```

- **cpu**:

CPU Type	Meaning
`host`	Uses the host's physical CPU features for the guest. Provides maximum performance with KVM acceleration. Ideal for VMs running on the same hardware.

CPU Type	Meaning
`qemu64`	Default CPU model for QEMU. Emulates a basic 64-bit x86 CPU with generic features. Good for general compatibility but may lack advanced features.
`qemu32`	Emulates a basic 32-bit x86 CPU. Use for legacy 32-bit guest OSes.
`kvm64`	Optimized for KVM acceleration. Provides better performance for KVM guests compared to qemu64.
Specific CPU Models	Emulates specific Intel or AMD CPUs, enabling features compatible with the named CPU.
Intel	SandyBridge, Haswell, IvyBridge, Skylake-Client, Broadwell
AMD	Opteron_G1, Opteron_G5, EPYC, EPYC-Rome

- **exmples**:

```
qemu-system-x86_64 -cpu host
qemu-system-x86_64 -cpu qemu64
qemu-system-x86_64 -cpu Haswell # Intel Core Processor (Haswell)
qemu-system-x86_64 -cpu Opteron_G1 # AMD Opteron 240 (Gen 1 Class Opteron)
```

- **smp**:
  - Sets the number of Processors.

```
-smp cpus=<n>[,sockets=<n>,cores=<n>,threads=<n>,maxcpus=<n>]
```

SMP Parameters	Meaning
`cpus`	Total number of CPUs or cores available to the guest.
`sockets`	Number of physical CPU sockets (default: 1)
`cores`	Number of cores per CPU socket (default: matches cpus).
`threads`	Number of threads per core (default: 1).
`maxcpus`	Maximum number of CPUs that can be hot-plugged.

  - The total number of virtual CPUs is calculated as:

```
total_cpus = sockets * cores * threads
```

- **exmples**:

```
qemu-system-x86_64 -smp cpus=4
qemu-system-x86_64 -smp smp cpus=4,sockets=2
qemu-system-x86_64 -smp cpus=8,sockets=1,cores=4,threads=2
```

- **m**:
  - Specifies the amount of memory for the virtual machine.
  - **exmples**:

```
qemu-system-x86_64 -m 4G # 4 GB of RAM
```

- **drive**:

```
-drive file=<filename>,format=<fmt>,if=<type>,media=
<media>,cache=<mode>
```

Drive Parameters	Meaning
`file`	Specifies the image file for the storage device.
`format`	Specifies the format of the image file (e.g., raw, qcow2, vdi, etc.).

`if`	**Specifies the interface type for the storage device.**
`if=ide`	Emulates an IDE device.
`if=scsi`	Emulates a SCSI device.
`if=virtio`	High-performance paravirtualized storage.
`if=sd`	Emulates an SD card.
`if=floppy`	Emulates a floppy drive.

`media`	**Specifies the type of media.**
`media=disk`	Treats the file as a hard disk image.
`media=cdrom`	Treats the file as a CD-ROM image.

`cache`	**Specifies the caching behavior for the device.**
`cache=none`	Direct I/O without caching.
`cache=writeback`	(default) Writes are cached for performance but not immediately written to disk.
`cache=writethrough`	Writes are immediately flushed to disk.

`cache`	**Specifies the caching behavior for the device.**
`cache=unsafe`	Optimized for performance but risks data loss.

Extra Drive Parameters	Meaning
`readonly<on|off>`	Specifies that the device should be mounted as read-only.

- **exmples**:

```
  qemu-system-x86_64 -drive file=Pop_OS_img.img,format=raw # Basic Hard Disk
  qemu-system-x86_64 -drive file=Pop_OS_img.qcow2,format=qcow2,if=virtio # High-Performance Virtio Disk
  qemu-system-x86_64 -drive file=cdrom.iso,media=cdrom,readonly=on # Attach a read-only CD-ROM image
  qemu-system-x86_64 -drive file=disk.img,format=qcow2,cache=writethrough # Attach a disk with write-through cache for data safety
```

- **cdrom**:
- Specifies an ISO file to emulate as a virtual CD-ROM drive.
- **exmples**:

```
qemu-system-x86_64 -cdrom pop-os_22.04_amd64_intel_42.iso
```

- **net**:
  - Configures networking for the guest machine.

```
-net <type>[,options]
```

NET Types	Meaning
`user`	(Default) Simple NAT-based networking for internet access.
`none`	No network device is attached.
`tap`	Bridge guest networking to a TAP device on the host.
`bridge`	Direct connection to the host's bridge.

- **exmples**:

```
  qemu-system-x86_64 -net user
```

- **display**:
  - Specifies how the VM's display is rendered.

Display Types	Meaning
`gtk`	GTK+ graphical window (default).
`none`	No display output (headless mode).
`vnc`	Opens a VNC server for remote control.
`sdl`	Uses SDL for the display window.

  - **exmples**:

```
qemu-system-x86_64 -display gtk
qemu-system-x86_64 -display none
```

- **vga**:
  - Configures the emulated graphics card for the VM.

VGA Types	Meaning
`std`	Standard VGA, basic graphics (default).
`virtio`	High-performance, paravirtualized graphics (requires drivers).
`qxl`	Optimized for SPICE displays.
`none`	No graphics device.

  - **exmples**:

```
qemu-system-x86_64 -vga virtio
qemu-system-x86_64 -vga none
```

- **boot**:
  - Specifies the boot order of devices.

Boot Order	Meaning
`a`	Floppy is tried first during the boot process.
`c`	Hard disk is tried first during the boot process.
`d`	CD-ROM is tried first during the boot process.
`n`	Network is tried first during the boot process.
`menu=on`	Boot Menu is tried first during the boot process.

  - **exmples**:

```
qemu-system-x86_64 -boot order=d
qemu-system-x86_64 -boot order=c
qemu-system-x86_64 -boot menu=on
```

- **snapshot**:
  - Runs the VM in snapshot mode, discarding changes on shutdown.
  - **exmples**:

```
qemu-system-x86_64 -snapshot -drive
Pop_OS_img.img,format=qcow2
```

- **monitor**:
  - Enables the QEMU monitor interface for managing the VM at runtime. It's and interactive command-line interface for VM control.

Monitor Modes	Meaning
`stdio`	Access via terminal.
`tcp`	Open a TCP socket for remote access.

  - **exmples**:

```
qemu-system-x86_64 -monitor stdio
```

- **usb**:
  - Enables USB device emulation for the VM.
  - **exmples**:

```
qemu-system-x86_64 -usb
```

- **audiodev**:
  - Configures audio output for the VM.

```
-audiodev <backend>,id=<id>[,options]
```

AudioDev Backends	Meaning
`alsa`	ALSA for Linux.
`pa`	PulseAudio.

AudioDev Backends	Meaning
`none`	Disable audio.

- **exmples**:

```
qemu-system-x86_64 -audiodev pa,id=sound0
```

- **rtc**:
  - Configures the real-time clock for the VM.

RTC Types	Meaning
`base=utc`	Base time set to UTC.
`clock=host`	Use host clock for synchronization.
`driftfix=slew`	Adjusts for clock drift.

- **exmples**:

```
qemu-system-x86_64 -rtc base=utc,clock=host
```

Let's Make Our VM:

- Basic Virtual Machine

```
qemu-system-x86_64 -smp cpus=4,sockets=1,cores=4 \
                    -m 8G \
                    -cdrom pop-os_22.04_amd64_intel_42.iso \
                    -boot menu=on \
                    -drive file=Pop_OS_img.img,format=raw \
                    -usb -rtc base=utc,clock=host
```

- Optimizations Solutions:

```
qemu-system-x86_64 -machine type=pc-q35-2.10 \
                    -smp cpus=4,sockets=1,cores=4 \
                    -m 8G \
                    -enable-kvm \
                    -object memory-backend-
ram,id=mem1,size=8G,prealloc=on \
                    -cdrom pop-os_22.04_amd64_intel_42.iso \
                    -boot menu=on \
                    -drive
file=Pop_OS_img.img,format=qcow2,if=virtio \
```

```
                         -vga virtio -device virtio-gpu-pci \
                         -usb -rtc base=utc,clock=host
```