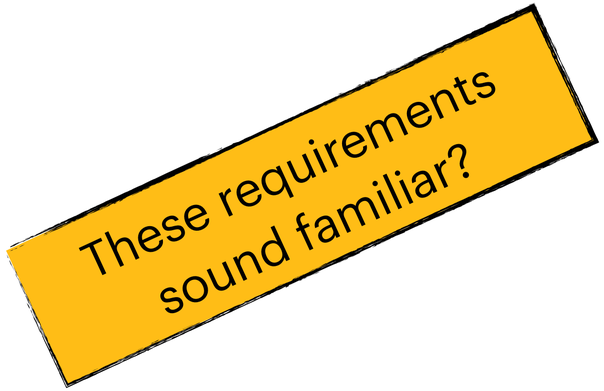
Virtualization of Computing Systems

Why Do We Need Virtualization?

To Multiplex Resources

- We want to **split the available compute resources** into small pieces and share them across different users
- One piece of resource must be isolated from the other pieces
- We must be able to create and delete a resource pieces quite efficiently
- Adjust the dimension of a piece quite easily
- Manage the pieces very efficiently



Virtualization to Enable Multiplexing

We have seen numerous examples

- CPU and memory virtualization are fundamental elements of modern Ones
- Multi-threading uses CPU multiplexing - each thread gets its own CPU for a limited time
- Multi-processing extends to CPU + memory multiplexing => each process gets its own CPU and memory for a limited time
- How can we extend multiplexing beyond CPU and memory?

Concept	CPU	Memory	File System	Kernel	Hardware
Multi-threading	Virtualized	Shared	Shared	Shared	Shared
Multi-processing	Virtualized	Virtualized	Shared	Shared	Shared
Containers	Virtualized	Virtualized	Virtualized	Shared	Shared
Virtual Machines	Virtualized	Virtualized	Non-Shared**	Non-Shared	Shared
Cluster	Non-shared	Non-shared	Non-shared	Non-shared	Non-shared

Virtual Machines

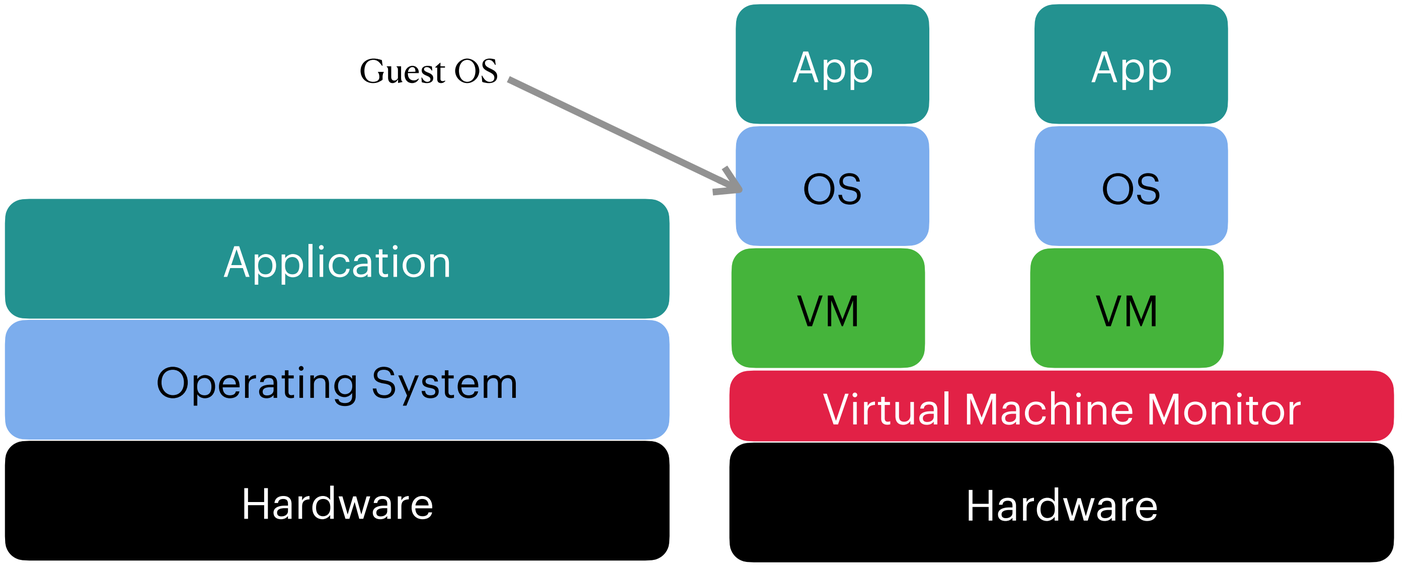
Partition a Physical Machine into Smaller Pieces

- Take a physical machine and split it into software machines that are identical copies of the hardware machine but smaller capacities
- Software machines would run anything that would run in the physical machine
- One software machine is isolated from another software machine although both run on the same hardware machine
- One software machine can fail without affecting another software machine
- How do we do virtualization without any slowdown?
- We want an application running inside a virtual machine to run as fast as it would if it ran in the physical machine

How is Virtualization Done?

Virtualization Compared Against Normal Operating System

Overhead (resource consumption) due to VM creation and deletion operations is a major concern with tenants having short tasks



- How many VMs can we make in a physical machine?
- How fast can we make VMs?
- What is the overhead of making the VMs?
- Tenants (cloud customers) use VMs to launch their tasks
- Small tasks mean VMs created and deleted more often

Also referred to as Hypervisors

Approaches to Virtual Machines

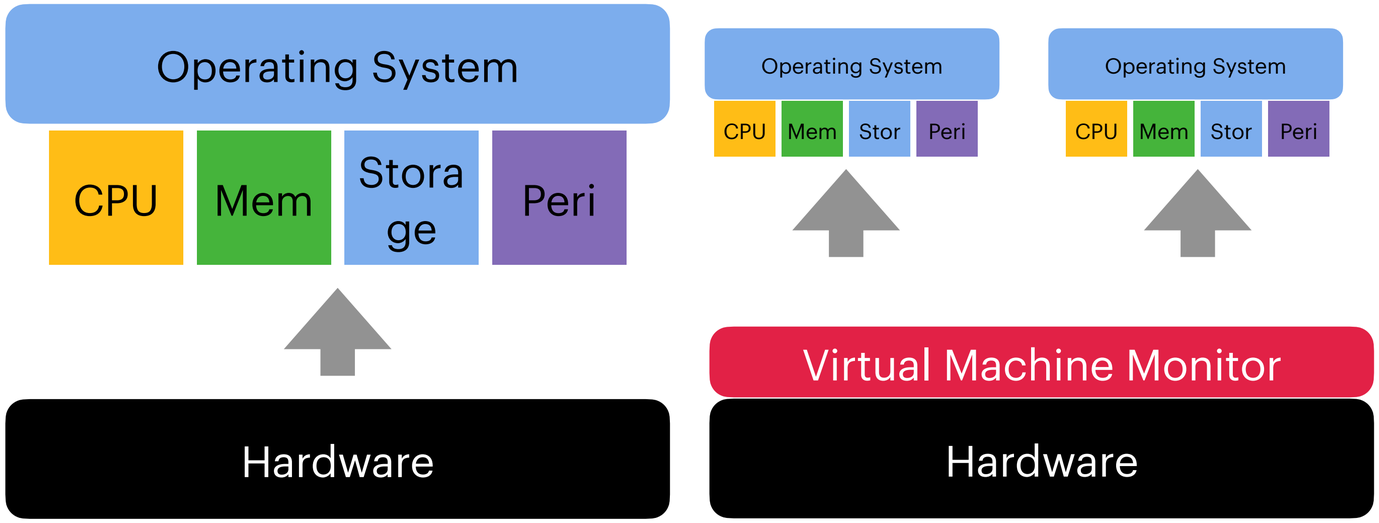
How to Provide Virtualization?

- **Type 0 hypervisors** - Implemented in the firmware. Found in mainframe or large servers. IBM and Oracle make these type of hypervisors
- **Type 1 hypervisors** - Operating system like software built to provide virtualization, include VMware ESX, Citrix XenServer. Type 1 hypervisors run directly on the hardware. A cloud installation would use this type of hypervisors
- **Type 2 hypervisors** - Run as applications on standard OSes (these are the host OS) by provide virtualization to support guest OS instances
- **Paravirtulization** - The guest OS is modified to work in cooperation with the VMM to maximize performance
- **Programming-environment virtualization** - Instead of hardware, virtualization is used to create an optimized virtual system - Oracle Java, Web Assembly
- **Emulators** - Allow applications written for one hardware to run on a very different hardware. QEMU is one of widely used computing system emulator.
- **Application containment** - Provides virtualization like features by segregating applications

In Depth on VMs

Virtualization Involves What Type of Resources?

Virtualizing hardware involves virtualizing many different types of resources



- CPU: overhead of virtualization must be very low in terms of time
- Memory: overhead of virtualization in time is very low but memory overhead can grow with number of VMs and memory size
- Storage and peripheral virtualization can add latency

Isolation

One of the Core Benefits of Virtualization

- **CPU Performance**
- **Memory**
- **File**
- **Peripherals** - can be software devices connected via networks

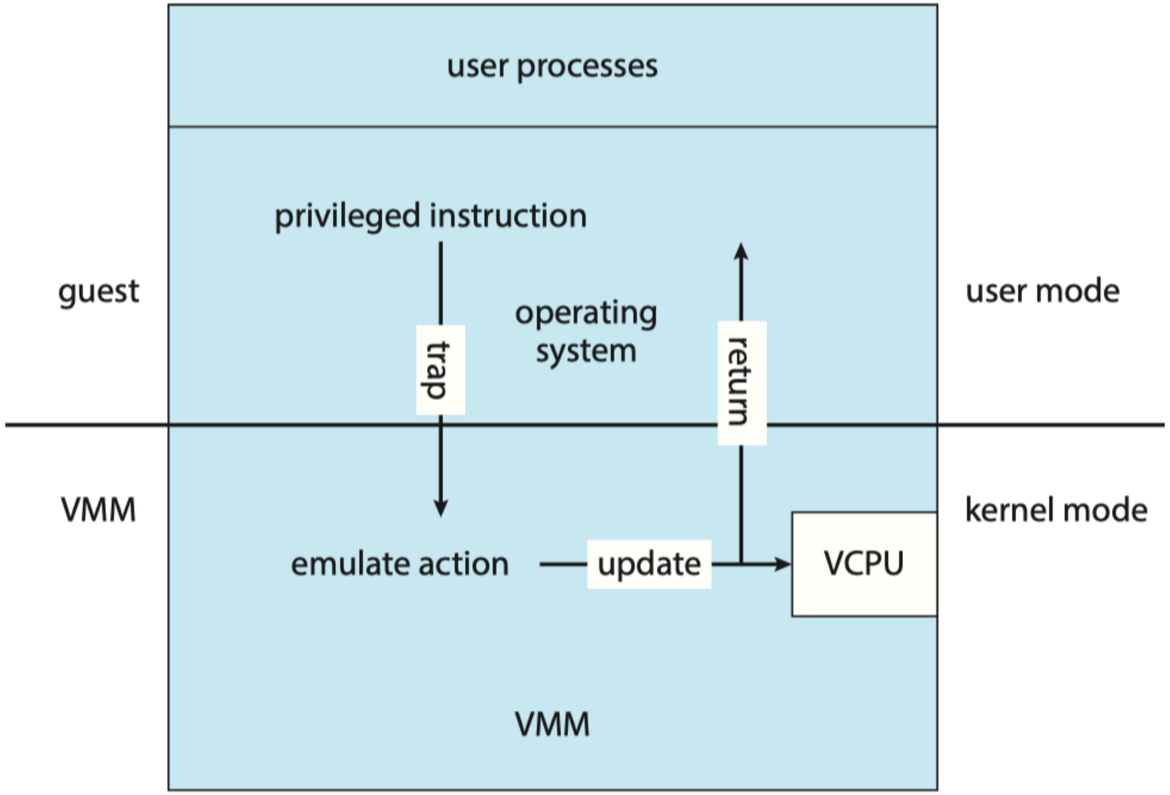
Give each VM an isolated virtual copy of the fundamental resource

- Dynamic CPU and memory usage can be challenging
- How to limit usage in a fair manner?
- CPU isolation without wasting CPU cycles - you don't want to set aside CPU for VMs that are not active
- Memory isolation without wasting too much memory

CPU Virtualization

How to create virtual CPUs?

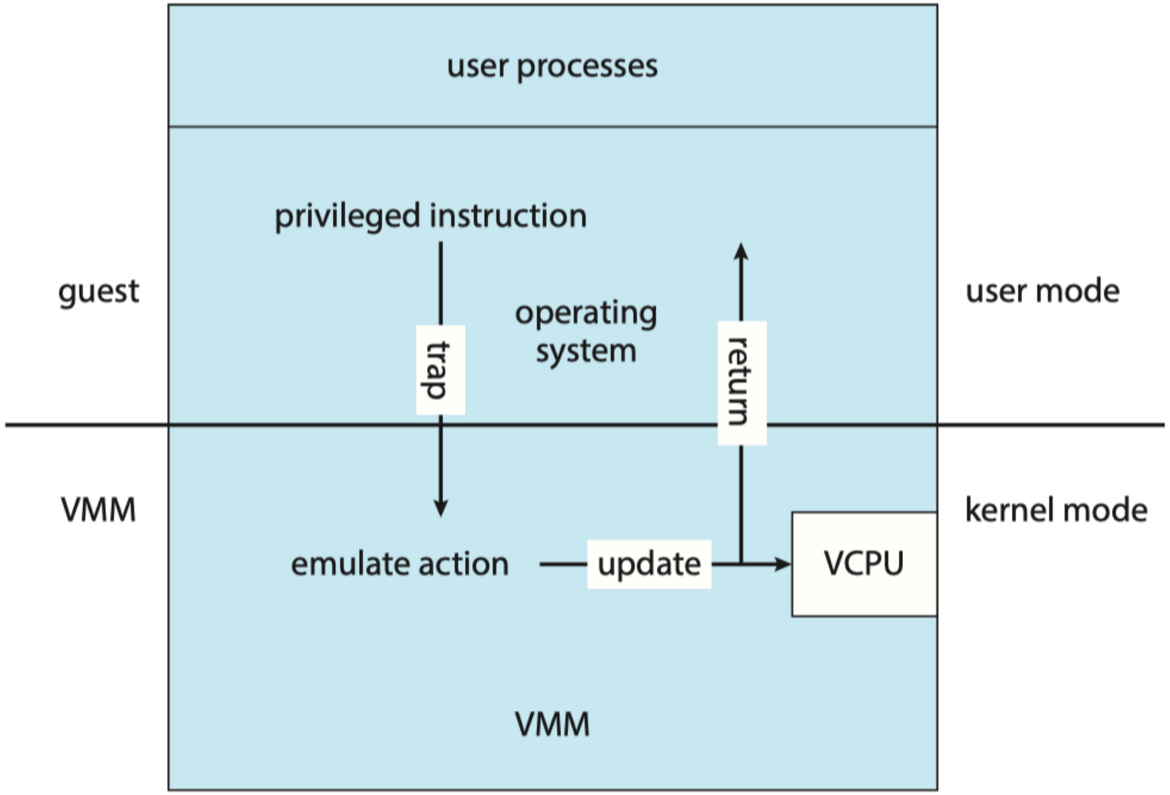
- Some CPUs are virtualizable and others are not virtualizable
- Intel x86 CPUs were not virtualizable until Pentium 4 was introduced in 2005
- IBM used virtualizable CPUs in their mainframes since earl 1970s
- If the CPU is virtualizable, we can use the Trap-and-Emulate approach for CPU virtualization



How Trap-and-Emulate Work?

How to create virtual CPUs?

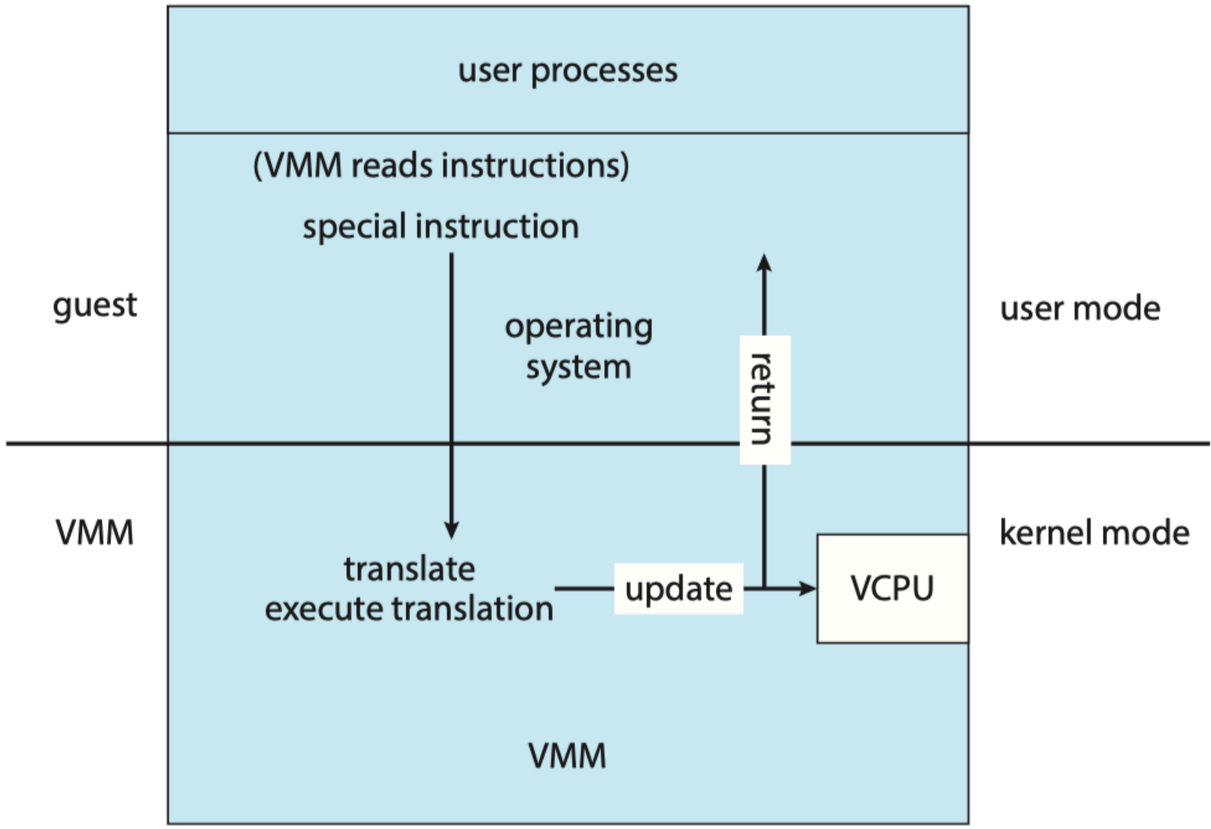
- Guest is an OS that is running in the real user mode
- Guest has Guest User Mode and Guest Kernel Mode
- Virtual Machine Monitor (hypervisor) is running in the real kernel mode
- Guest can run privileged and non-privileged instructions and privileged instructions cause a trap from user->kernel mode
- The VMM running in the kernel mode does the emulation of privileged execution and returns the call.
- So, system call execution is emulated by the VMM



When would Trap-and-Emulate Not Work?

Privileged Instructions vs Non-Privileged Instructions

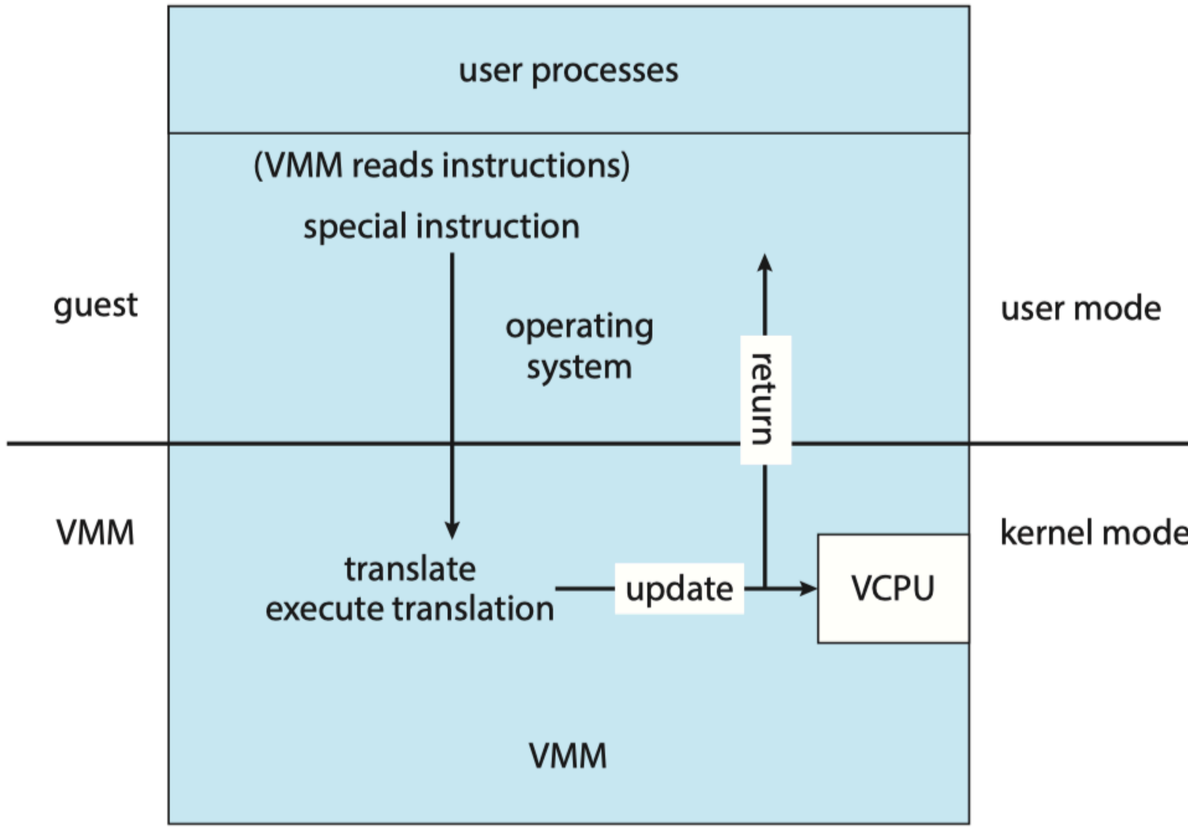
- Intel x86 CPUs had instructions that behaved differently based on whether the CPU is in privileged mode or not
- Such instructions are going to cause problems for Trap-and-Emulate because these instructions would not cause the trap
- Since the Guest is in user mode such instructions would lead to wrong execution - different from native execution
- We call such instructions sensitive instructions.
- If all sensitive instructions are privileged, the CPU is fully virtualizable - that is, Trap-and-Emulate would work
- For CPUs not satisfying the above, Trap-and-Emulate would not work
- We need Binary Translation



Binary Translation

How to Virtualize x86 CPUs before Pentium 4?

- Guest is in the Guest User Mode, it can run the instructions natively on the physical CPU
- Guest is in the Guest Kernel Mode, it needs the VMM to examine every instruction the Guest would execute. The VMM need to at least translate the sensitive instructions to equivalent code that would manipulate the VCPU
- Binary translation can be slow - to speed up we need to cache the translated instructions



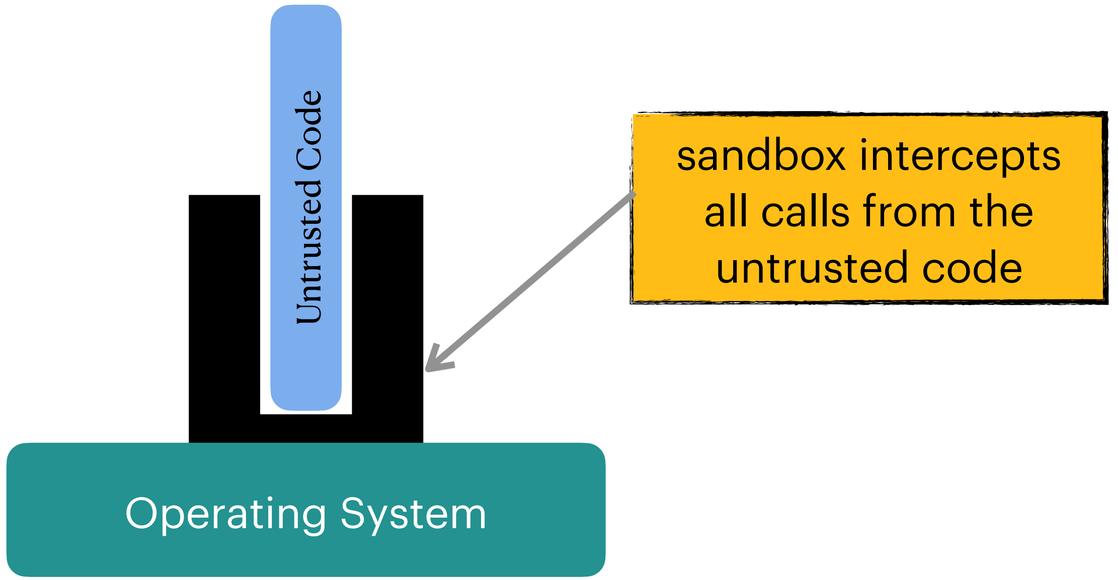
Containers

The Fake Virtual Machines

- We need something that is really fast, does not consume as much resources
- Allows us to run many applications on a single server with the applications not interfering with each other
- We want same level of isolation from containers as VMs
- Need to provide the applications with all authorized resources
- An application should not be adversely affected by another application that is overusing its resource allotment

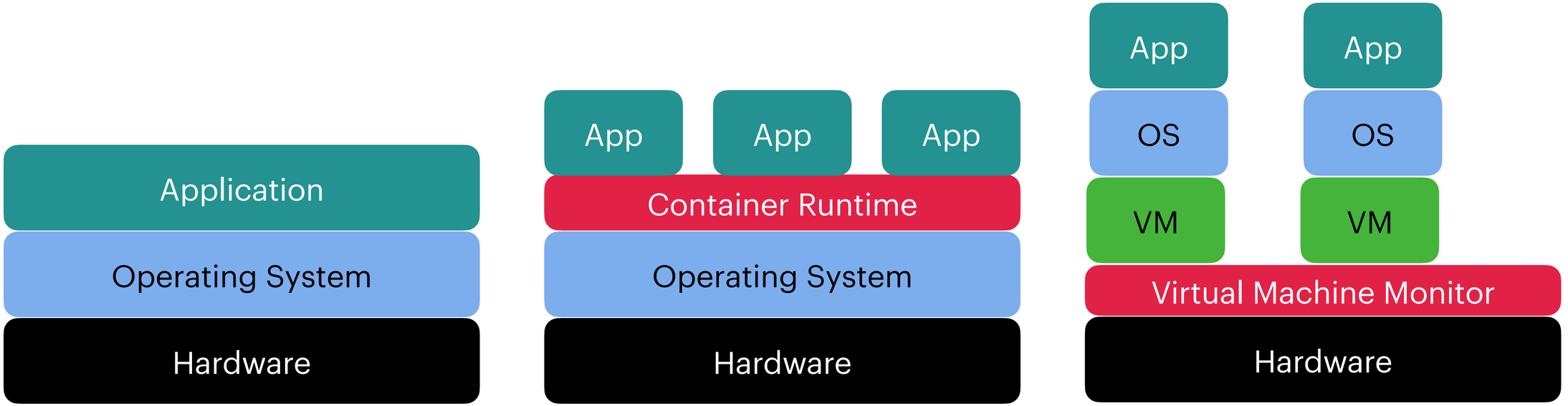
Containers: How it All Began?

- Containers started with process jails created for running untrusted code
- Process jails monitor what the code is doing and rejects certain access to files, to keep running on the CPU, etc
- Process jails have been enhanced and built into web browsers to all users to run remote scripts - most websites rely on scripts - they need to be executed safely
- In the context of browsers process jails are known as sandboxes



Containers Versus VMs

Illustration Using App Deployment

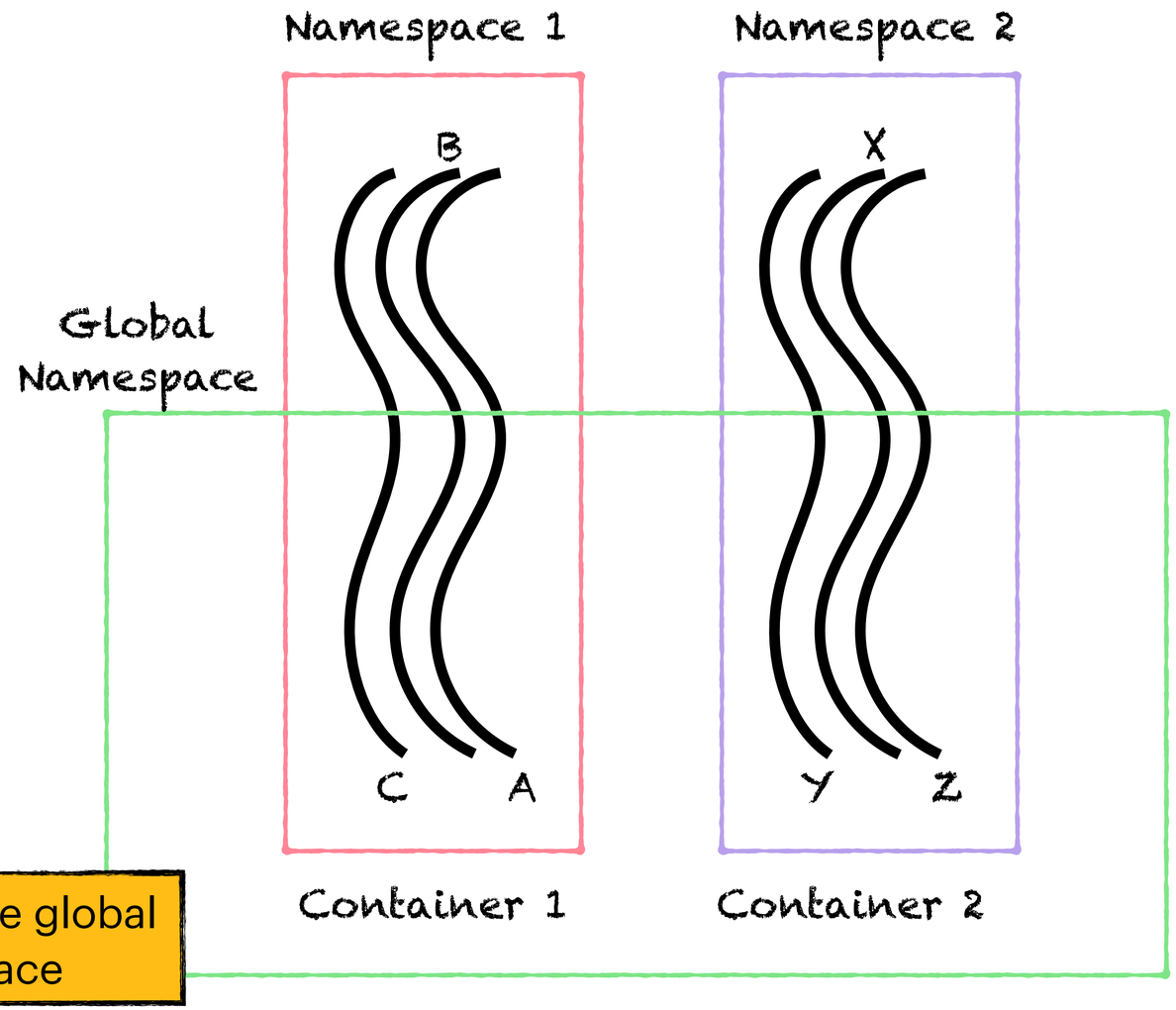


- Containers are reusing facilities that come with the OS - additional overhead very small
- No duplication like VMs
- File systems are also created on top of the native file system provided by the OS

Container Architecture

What are the Major Elements of a Container?

- A containers is a group of processes with its own file system
- All containers running on a host share the same kernel instance
- Three major ideas are used to build a container: namespaces, control groups, and union file systems



Namespaces: Isolation of Access

Limit what an application can access

- Namespaces is an extension of the address space isolation used by memory virtualization
- Memory virtualization offers isolation of memory portions allocated to one process from others co-residing in the computer
- Namespaces is a way to isolate access to any arbitrary resource
- To access a resource (file handle, process, etc) we use handles
- For example: to send a signal to a process, you use **kill processid**.
- What if we are able to virtualize the process ID space? This way we cannot have a process send a signal to another from another namespace

Control Groups: Isolation of Performance

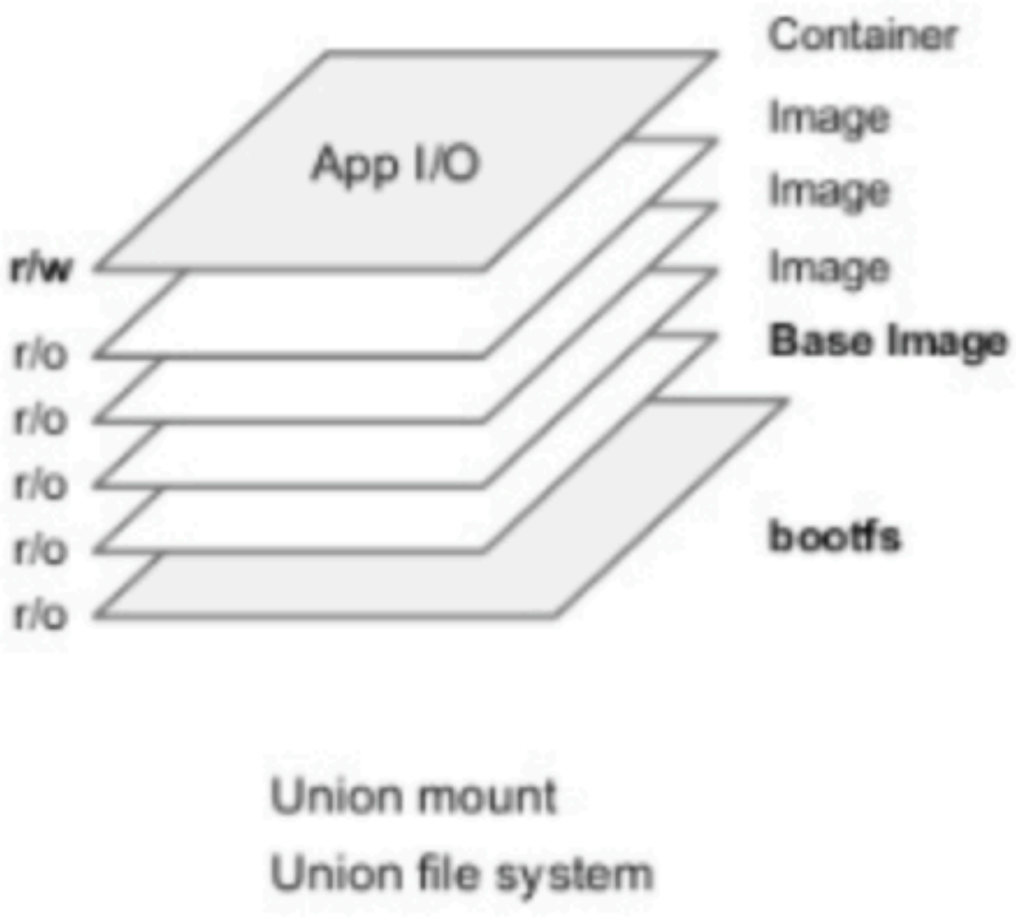
Limit the amount of resource consumed by an application

- A machine has a size - limited CPU, memory, and storage capacity
- A container emulating a machine needs to enforce these limits
- Containers is splitting the server on which it runs into smaller machines and enforces the limits
- Control groups are used for this enforcement
- Control groups use an extended proportional resource sharing scheme - like lottery scheduling
-

Union File System

Create a flexible but efficient file system

- Multiple layered file system
- Bottom layers are read only
- Top layer is writeable



The Downside of Containers

Why We Should Not Use Containers All The Time?

- Containers are less secure compared to VMs
- One application compromising the OS instance is sufficient
- Containers fail when the operating system instance fails - there is only one OS instance with containers
- If we have many different customers - containers is not a good way of achieving isolation
- Containers are good for isolation within the different applications of a single customer
- We need VMs for general cloud multi-tenant situations

Wrap Up

Summary of Main Points Learned So Far

- We studied the main concepts of cloud computing
- We saw that a flexible resource partitioning scheme is important for cloud computing.
- Virtual machines provide such a flexible resource partitioning scheme although with high overhead
- Containers is another option that is quite lightweight
- We examined containers and virtual machines in detail
- Containers are very useful subject some limitations and we studied them