VIRTUALIZATION and STORAGE

Virtualization isolates the details of the hardware from the software that uses it. You can break it down into two broad categories:

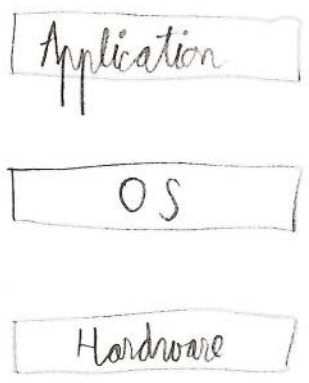
Process Virtualization
- run a process under the control layer of software, e.g the JVM running Java bytecode

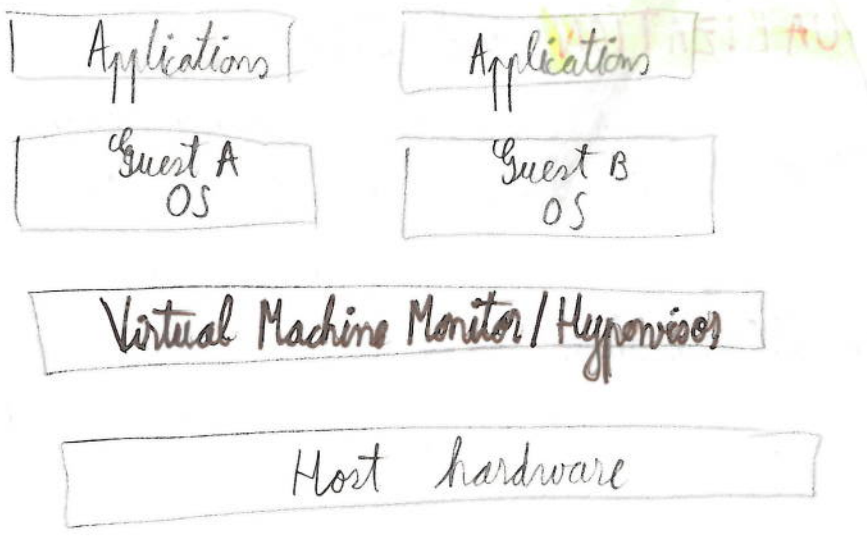
System Virtualization
- run an operating system under the control of a layer of software, e.g VMware

Goals of System Virtualization
- isolation of the guest O.S from the exact details of the hardware (e.g CPU type, memory configuration, peripheral devices)

What can virtualization do?
- translate between technologies (different instruction sets, system calls etc.)
- change the level of abstraction (providing garbage collection, debugging etc.)
- make the system resources look different (emulate CD drives, reduce RAM amount for a virtual machine)

Reverse Debugging
- when you hit a breakpoint, the debugger lets you step back through the code. This is often implemented by having the VM keep track of what each instruction did and reversing the operation each time you step back.





Unvirtualized Virtualized

VMM (Virtual Machine Monitor)

- a virtualised OS runs on top of a VMM
- handles physical resources access for the $\overbrace{\text{guest OS}}^{\text{unprivileged}}$ since it runs in a privileged mode such as: timers, CPU registers, CPU flags, device control registers (DMA), memory mapping (page table)
- when a guest OS tries to access resources its not allowed to, it will trigge a trap instruction on the VMM, which allows the VMM to check the bounds of acces for that instruction/OS and proceed accordingly
- some instructions behave differently according to what mode they are in, so the VMM must be able to hande that

tions on VMs

t **start**, the VMM saves current registers, loads VM's initial registers and jump o the VM's PC. At **stop**, the VMM will save its registers into its own memory space (stopped so that CPU can be shared)

Quiescent - in a state of inactivity or dormancy - best time to stop a VM

Freeze - save VM's state into a file. Because of this we can move a VM onto a different machine, snapshot its state, quickly start it

Live migration

= move a VM from one machine to another without **pausing** execution

Phases:

Warm-up phase

- VMM copies all memory pages from source to destination while the VM is still running

Stop-and-copy phase

- VMM stopped on source, dirty pages copied, then VM resumed on destinatio
  Downtime = time between stopping the VM on source and resuming it on destination

Load Balancing

- management software monitors "load" on all physical machines
- if loads are mismatched, migrate a VM from a loaded to a less-loaded machine

High Availability

- for critical applications, keep a standby VM available on a different hardware syst
- regularly copy active VM image to standby VM (but don't activate it)
- activate standby VM if active VM stops responding (VM crashes? VMM crashes? Hardware system fails?)

Rapid provisioning = deployment of an OS image, including libraries and applicatio
that has been previously configured and saved, perhaps in an
archive of virtual machines, onto a hardware system

How can it be implemented using System Virtualization?
By copying the file containing a "frozen" image of the required VM to the host system and resuming it under control of the Virtual Machine Monitor/ Hypervisor

Checkpointing and restoring
Every time the VMM/ Hypervisor pauses a Virtual Machine, enough state is saved to be able to resume the VM later. Checkpointing saves this state in a file, alongside an image of the "physical" memory of the VM and its configuration, for later usage. Restoring this checkpoint is a matter of moving the memory image in the correct place, reloading relevant hypervisor state from that stored earlier, and resuming execution