Virtualization Overview

NSRC

UNIVERSITY OF OREGON

NSRC
Network Startup Resource Center

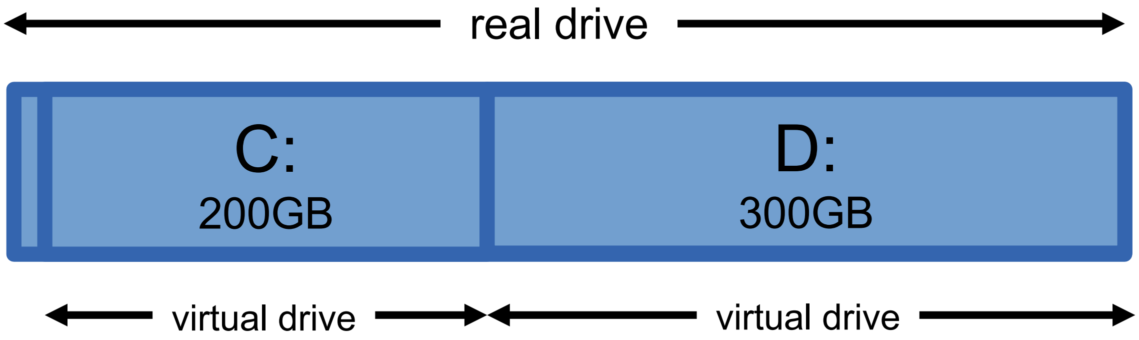
Terminology

- Virtualization: dividing available resources into smaller independent units
- Emulation: using software to simulate hardware which you do not have
- The two often come hand-in-hand
  - e.g. we can *virtualize* a PC by using it to *emulate* a collection of less-powerful PCs

Benefits (versus dedicated hardware)

- Consolidation
  - Most systems are under-utilized (especially the CPU)
  - Reduce space and power requirements
  - Run different OSes on the same machine at once
- Flexibility
  - Create, grow/shrink and delete instances as required
- Additional capabilities
  - Snapshots, migration, off-site replicas, …

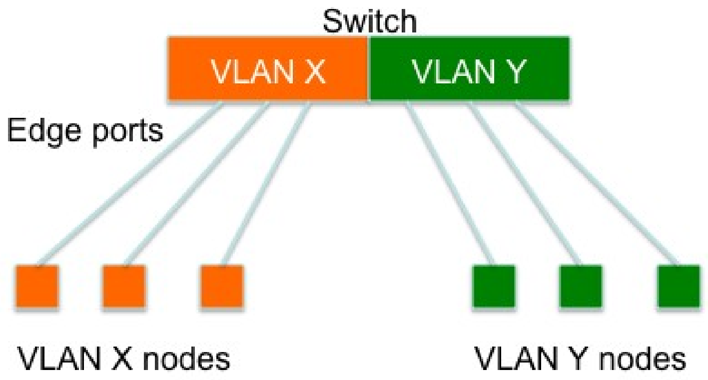
Virtualization: a familiar example



- **Who has not seen this before?! (Raise hands please)**
- **Like having two (or more) hard drives**
  - **you get to choose the sizes**
- ***Why*** **is this useful?**

Another example

- Virtualize a switch: VLANs
  - like dividing a switch into separate switches
- Benefits:
  - isolation: separate broadcast domains
  - can create and assign VLANs purely through software configuration
  - can combine VLANs onto a single cable (tagging/trunking)

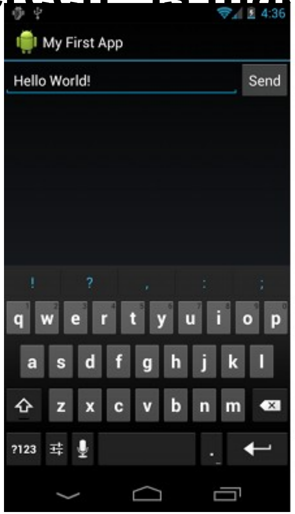


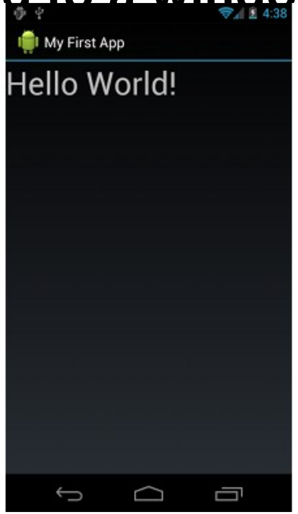
Emulation

- In software, you can simulate the behavior of a device which doesn't exist
  - Example: emulation of a CD-ROM drive using an ISO file
- a request to read block N of the (virtual) CD-ROM drive instead reads block N of the ISO file
  - similar to partition mapping
  - You can simulate any hardware - including the CPU or an entire system!

Entire system emulation - example

- **Android SDK**
  - **Emulates an Android smartphone with ARM CPU**
  - **The "screen" is mapped to a window on your PC**





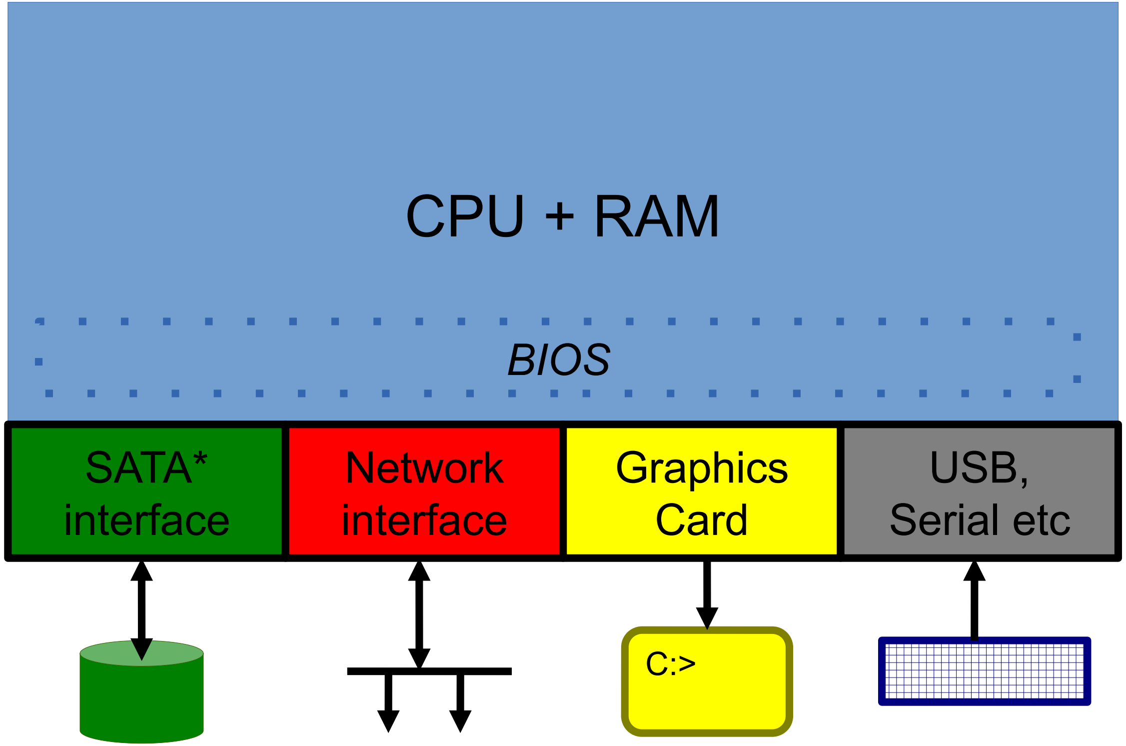
UNIVERSITY OF OREGON

NSRC
Network Startup Resource Center

System emulation

- There is no physical phone hardware
- The ARM CPU code is interpreted in software
- When the application executes an instruction which tries to write to the "screen", this is intercepted
  - It instead updates a buffer in memory, which then gets drawn in a window
- The software running inside the emulator is unaware that this is happening

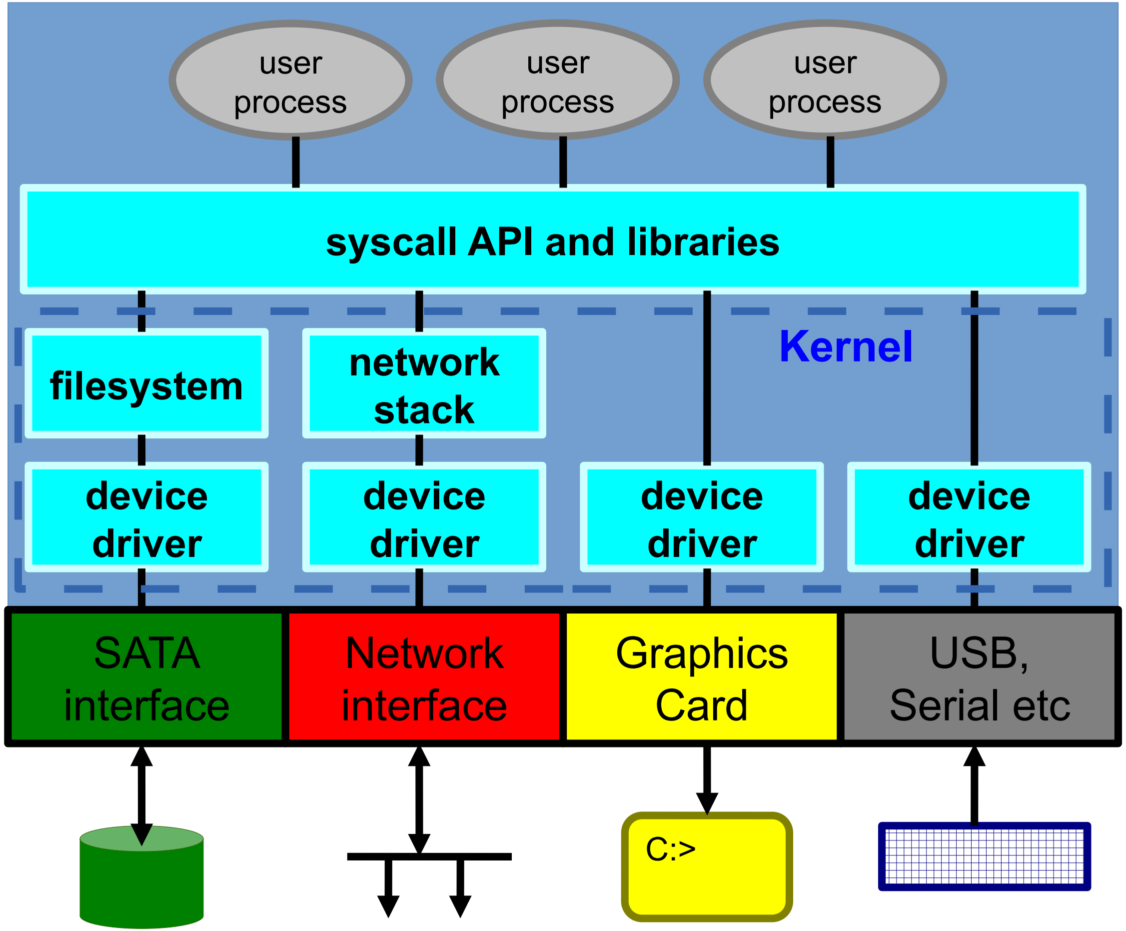
What's in a PC?



*i.e., hard drive

Boot sequence

- A small program (the BIOS) runs when machine is switched on
- It uses the hardware to load an operating system
  - boot from hard drive, USB/CD-ROM, network...
  - *name comes from "lifting yourself off the ground by your own bootstraps"*
- Modern operating systems then ignore the BIOS from that point onwards
- The next slide shows a machine after it has booted up (simplified)



UNIVERSITY OF OREGON

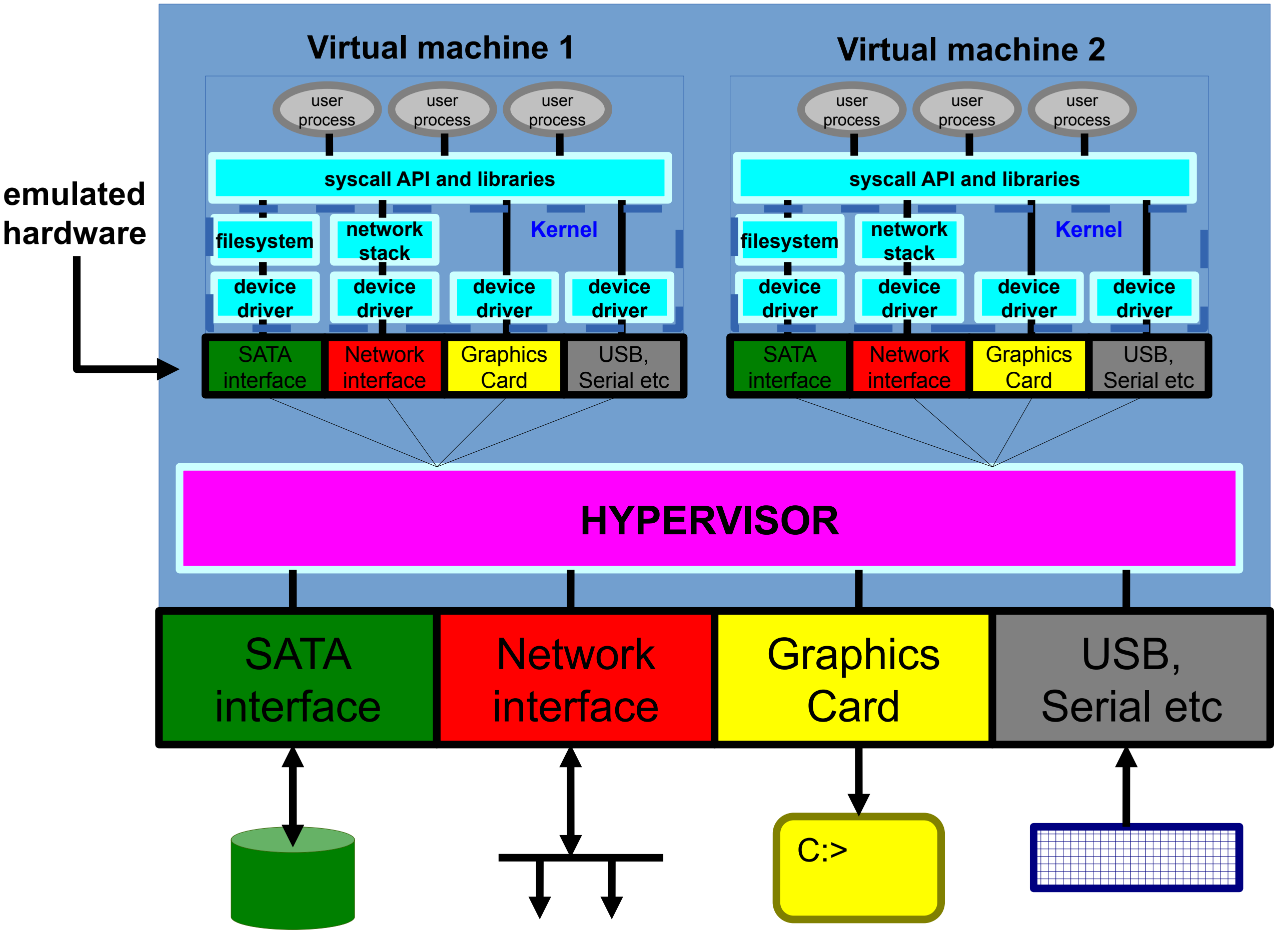
NSRC
Network Startup Resource Center

Points to note

- The device drivers in the OS interact with the hardware
- User processes are forbidden by the OS from interacting directly with the hardware
  - the OS configures protection mechanisms to enforce this

What we need to emulate a PC

- We must emulate all the components of the PC
  - CPU and BIOS
  - hard disk interface, network card
  - graphics card, keyboard, mouse
  - clock, memory management unit etc
- We want multiple instances to co-exist and not be able to interfere with each other
  - access to memory must also be controlled
- The software to do this is called a <u>**hypervisor**</u>



UNIVERSITY OF OREGON

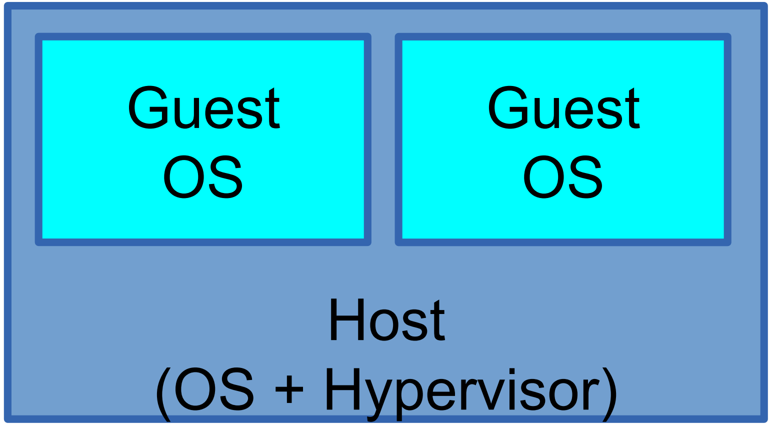
NSRC
Network Startup Resource Center

Virtual Machines

- Each emulated PC is a "virtual machine"
- Hypervisor allocates some real system RAM to each VM, and shares the CPU time
- Hypervisor emulates other hardware, e.g. disk and network
  - (or in some cases, you might pass through the whole device)
- Within each VM you can boot an operating system
- Full hardware virtualization means different VMs can be running different OSes

Virtualization terminology

- The host is the machine running the emulation
- The guest is the emulated (virtual) machine
- One host could be running many guests



The Hypervisor

- Note that the Hypervisor itself needs an operating system *
  - It needs device drivers, a filesystem, a network stack for remote management, etc
- So there is a host OS for the hypervisor, plus guest OSes
- The hypervisor needs a management interface for you to create, configure, start/stop and otherwise manage the guests

* Even so-called "bare-metal" or "Type 1" Hypervisors include a cut-down operating system

CPU emulation

- Emulating a CPU in software is very expensive
  - One guest CPU instruction takes many host CPU instructions to emulate
- If the host and the guest have the same CPU architecture, then guest code can run directly on the CPU at full speed
- However, the hypervisor still has to intercept any attempt by the guest to access hardware directly
- Modern CPUs provide hardware support to make this efficient
  - Intel: "VT-x", "VT-d"
  - AMD: "AMD-V", "AMD-Vi"

Emulated disk hardware

- A hard drive is a "block device"
  - OS makes requests like "read block number 42", "write block number 99"
- Real hard drives have a fixed size!
  - This is what the guest OS will expect to see
- The hypervisor must redirect these accesses to something else
- Options include:
  - a disk image file on the host (simple)
  - a partition or logical volume on the host (faster)
  - a remote file or remote block device (via network)

Disk image files

- VM disk which is a regular file inside the host's filesystem
- A disk image file is easy to backup and transfer from host to host
- It's a bit less efficient than direct-to-disk access
  - it has to go through more layers in the host filesystem
- There are different types of disk image file
- Suppose we want the guest to have a 10GiB virtual hard drive. What options are there?

Option 1: 10GiB raw file

- A "raw" file is a just a plain data file
  - a 10GiB virtual disk is exactly a 10GiB file on the host
  - N^{th} block of the virtual hard drive corresponds to the N^{th} block in the image file
- If this is allocated up-front, you use 10GiB of (probably) contiguous space on the host
  - Fast in operation, avoids fragmentation on the host
  - Wasteful of space
  - Slow to create
  - Slow to copy

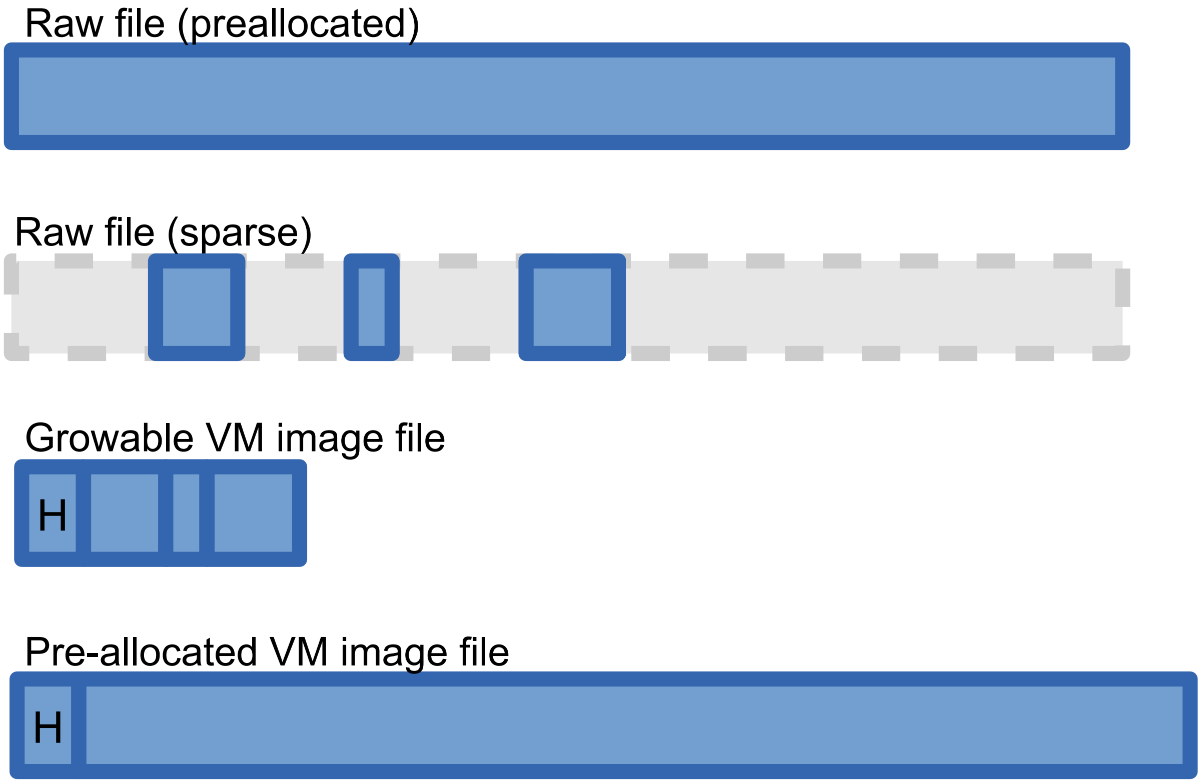
Option 2: 10GiB sparse raw file

- Some OSes support "sparse" files, files with "holes"
  - still looks like a plain 10GiB file
  - but it doesn't allocate space until each block is written to; this is known as "thin provisioning"
  - the size of the file ("`ls -l`") is larger than the disk space used by the file ("`ls -s`" or "`du`")
- can lead to fragmentation
- can lead to failures if filesystem becomes full
  - "overcommitting": creating more VM images than actual space available
- if you are not careful, may expand to the full 10GiB when copied

Option 3: custom VM disk image format

- Various formats, e.g. QCOW2 (qemu/kvm), VDI (virtualbox), VMDK (VMware)
- Has a header which maps blocks to file offsets
- Efficient space utilization
  - supports thin provisioning without needing OS support for sparse files
  - can be copied without losing its "sparseness"
  - still leads to fragmentation, unless you pre-allocate all the space
- Other features, e.g. snapshots within the same file
  - only the differences between the snapshots are stored

Comparison of disk image types



Emulated network hardware

- Each guest NIC gets a fake MAC address
- Different ways to interconnect with host NIC
  - "NAT": outbound packets translated to share the host's IP address
  - "Bridging": packets sent/received untranslated over the host's NIC, and each VM gets its own IP address on the external network
  - More complex setups, e.g. overlay networks

Performance optimizations

- Emulating disk hardware and network hardware is also expensive
  - has to emulate bits in registers, interrupts etc
- "Paravirtualization" is where the guest OS communicates explicitly with the hypervisor, and is more efficient
  - A popular implementation is called "VirtIO"
  - The guest OS has to have suitable drivers and be aware it is running inside a virtualization environment
  - Most Linux kernels do; Windows VirtIO drivers are available
- Allows other features like host file sharing and "balloon memory"

Choosing a virtualization platform

Popular hypervisors

- For Linux hosts: KVM most common; also Xen, VirtualBox
- For Windows hosts: Hyper-V, VirtualBox
- Others: FreeBSD Bhyve…
- Commercial offerings
  - ESXi used to be free, but not any more
- We will focus on KVM
  - It's very popular, very actively developed, widely supported

About KVM

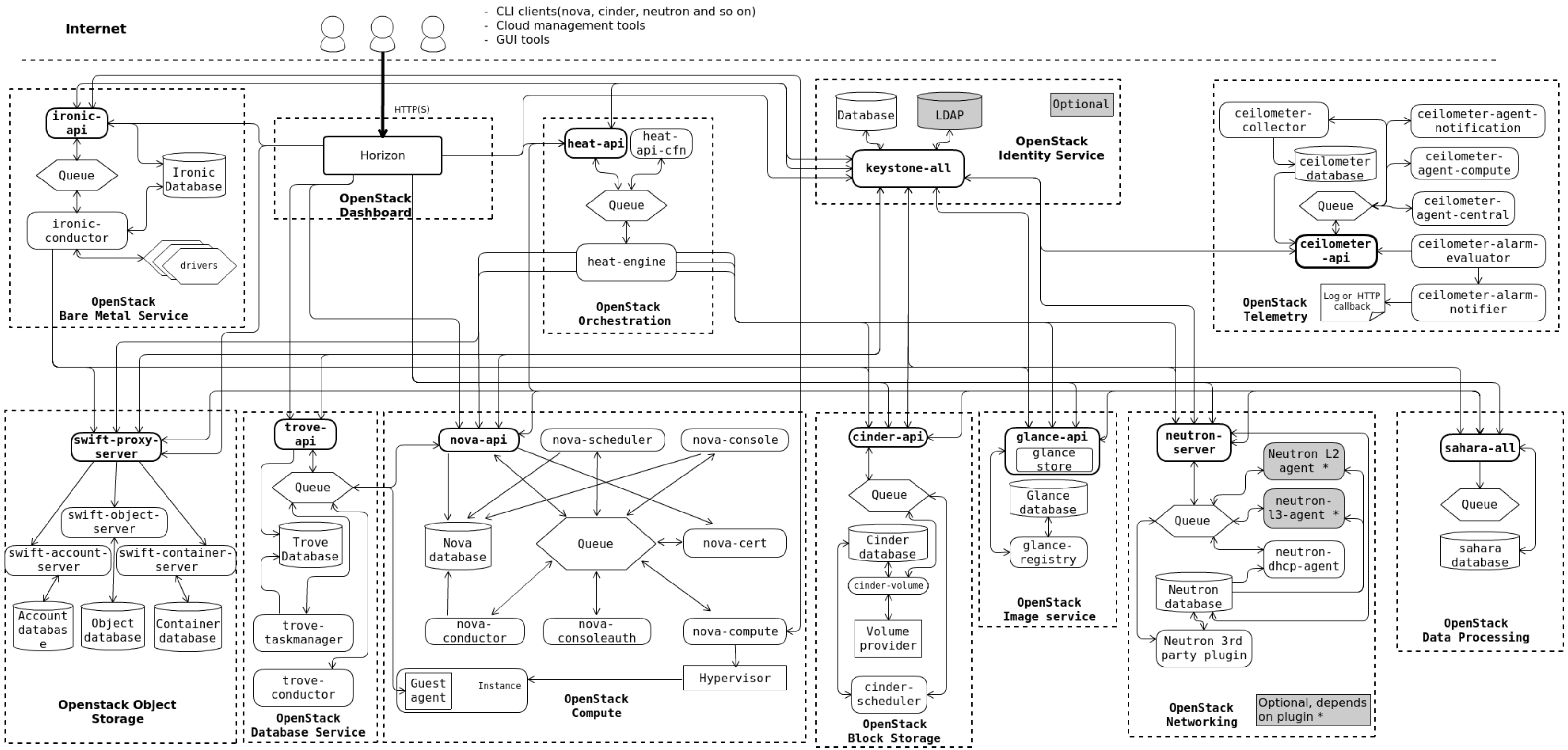
- KVM = Kernel Virtual Machine
  - Built-in to the Linux kernel
  - The host must be Linux (but not necessarily the guests, of course)
- KVM *requires* VT-x or AMD-V to run
- Separate software to emulate PC devices (normally QEMU)
- Each VM is just a userland process
- Can even run it directly from the command line!

```
kvm -cdrom /path/to/image.iso
```

Management framework

- You need software to start and stop those KVM processes
  - need to pass large numbers of command line arguments to configure CPUs, RAM, disks, networks, and other devices
- There are lots of options
  - libvirt runs on a single machine; virt-manager is X11 GUI for libvirt
  - proxmox VE: VMs and containers
  - lxd / incus: system containers and VMs (Canonical "MicroCloud")
  - oVirt (upstream of commercial RHV)
  - commercial software built on top of KVM (e.g. Nutanix…)

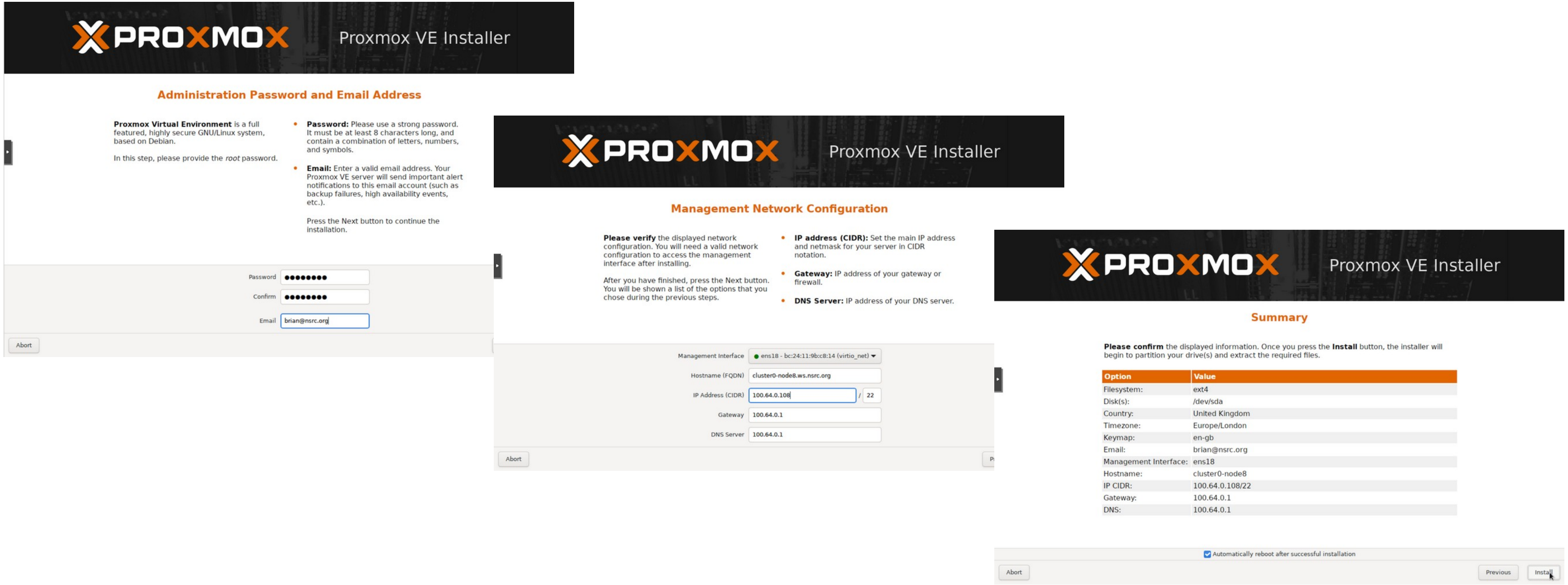
What about Openstack?



We've chosen Proxmox VE

- "Mostly free"
  - Free installation from ISO or Debian packages
  - Paid-for access to their "enterprise repository" for updates, and support
  - Free repository of "less well tested" updates (CentOS stream-like model)
- Start small, grow to clusters of 15-20 servers
  - Above that, go for multiple independent clusters
- Very good web UI, useful API and command line
  - Manage the whole cluster from any host
- Many storage options
  - including LVM, ZFS, Ceph, Linstor (as a plugin)

Proxmox VE ISO installation



The End

… and Proxmox Lab