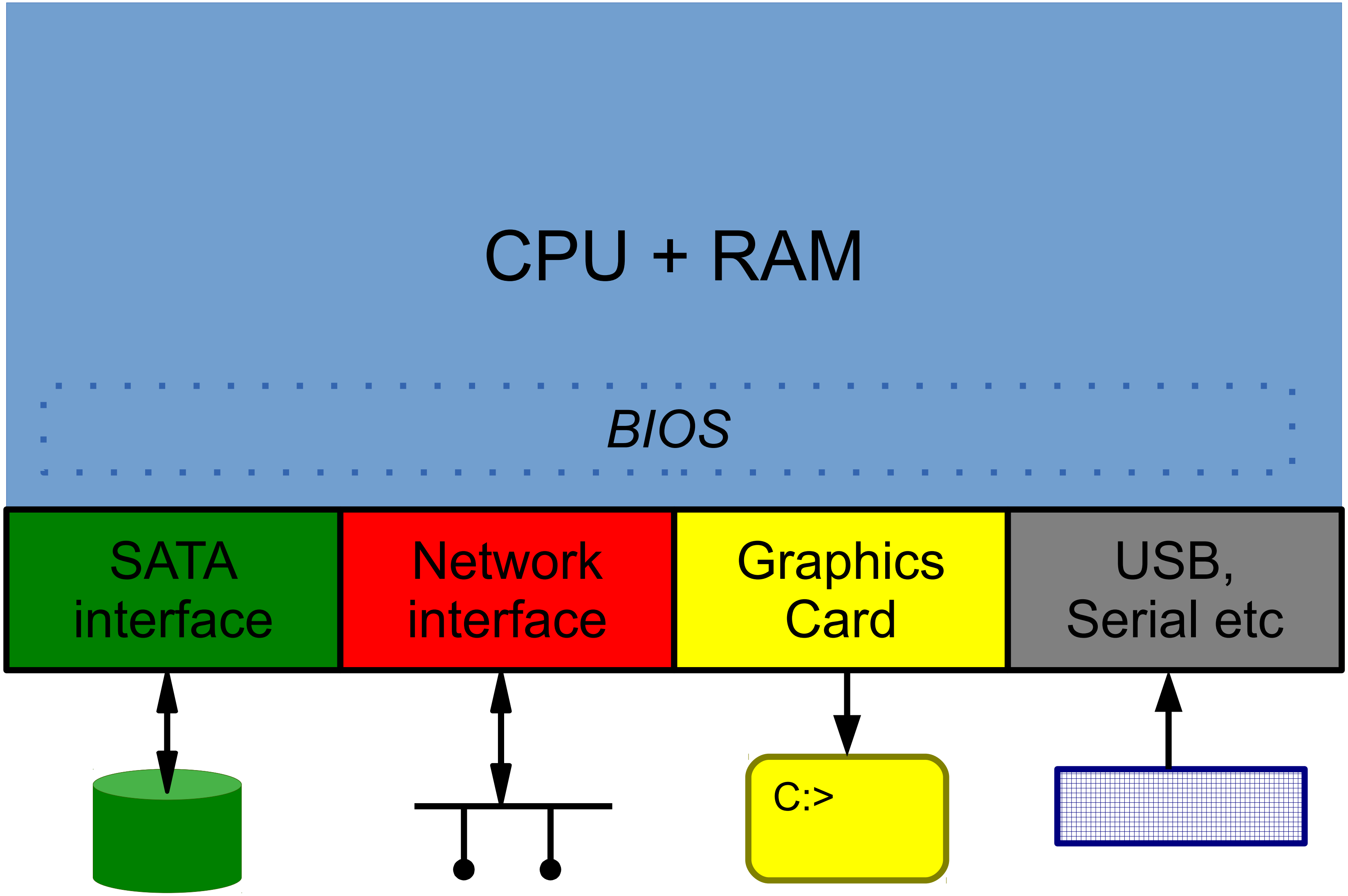
KVM and libvirt

NSRC

Virtualisation Recap

NSRC

What's in a PC?



CPU + RAM

BIOS

SATA interface

Network interface

Graphics Card

USB, Serial etc

C:>

Terminology

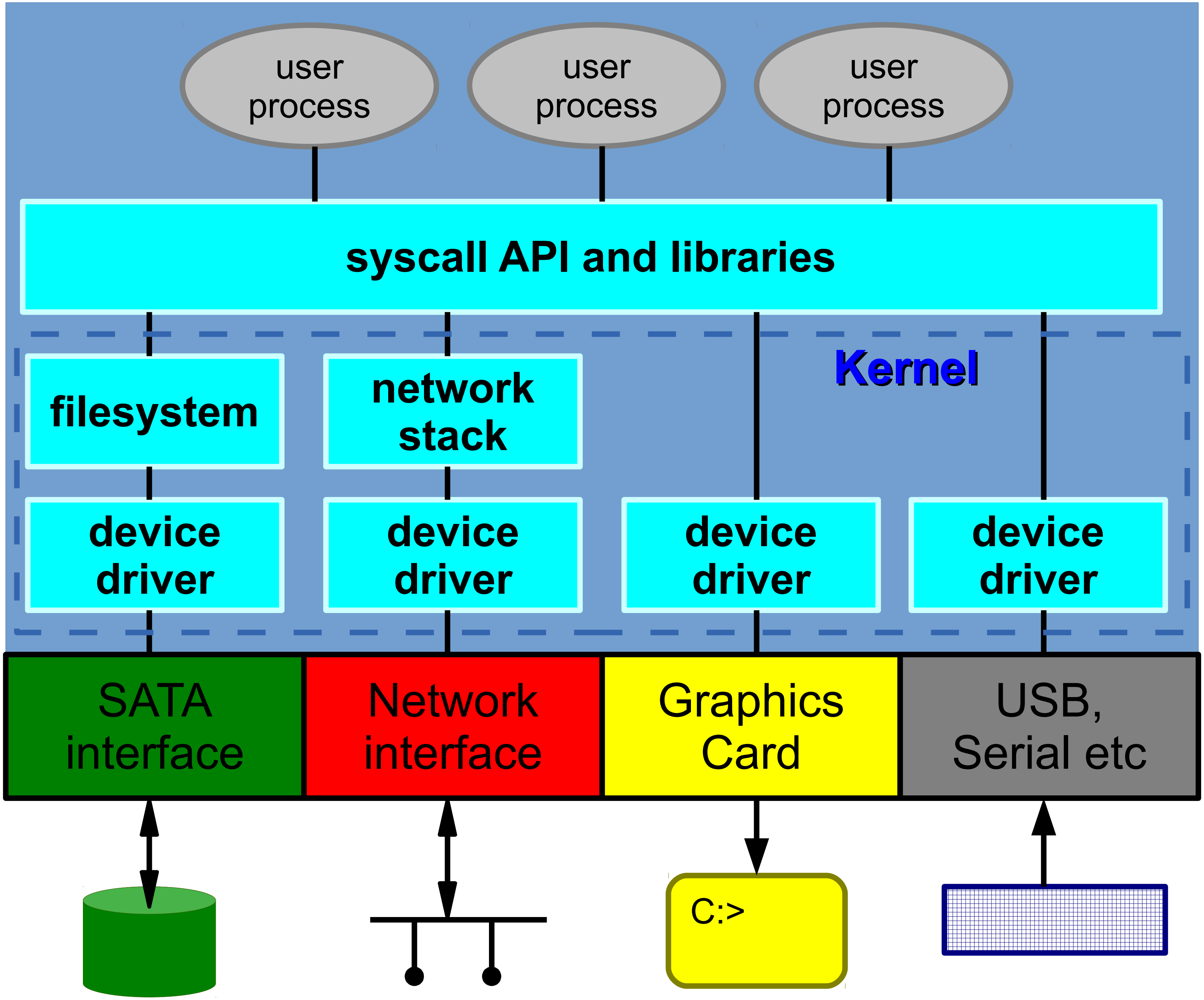
- Virtualization: dividing available resources into smaller independent units
- Emulation: using software to simulate hardware which you do not have
- The two often come hand-in-hand
  - e.g. we can *virtualize* a PC by using it to *emulate* a collection of less-powerful PCs

Benefits

- Consolidation
  - Most systems are under-utilized, especially the CPU is idle for much of the time
  - Do more work with less hardware
  - Reduced space and power requirements
- Management
  - Less hardware inventory to manage
  - Concentrate your resilience efforts
  - Increased isolation between services
  - Abstract away (hide) differences in hardware

Benefits

- Flexibility
  - Grow systems on demand (e.g. allocate more CPU or RAM where it is needed)
  - Create new services quickly without having to install new hardware every time
  - Dynamically create and destroy instances for testing and development
- New capabilities
  - Snapshot/restore, cloning, migration, …
  - Run different OSes on the same machine at once

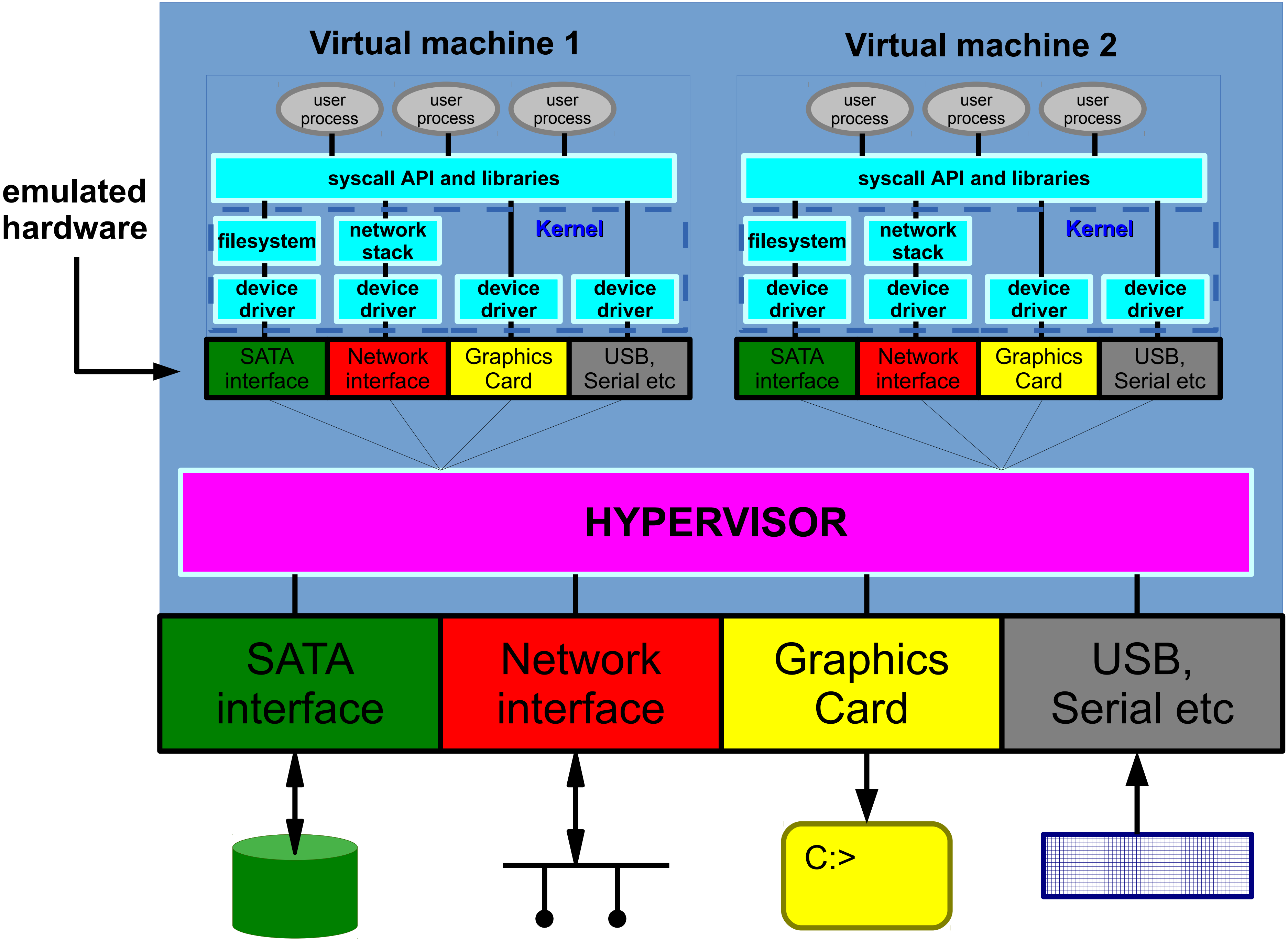

user process
user process
user process
syscall API and libraries
Kernel
filesystem
network stack
device driver
device driver
device driver
device driver
SATA interface
Network interface
Graphics Card
USB, Serial etc
C:>

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
  - 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 *hypervisor*

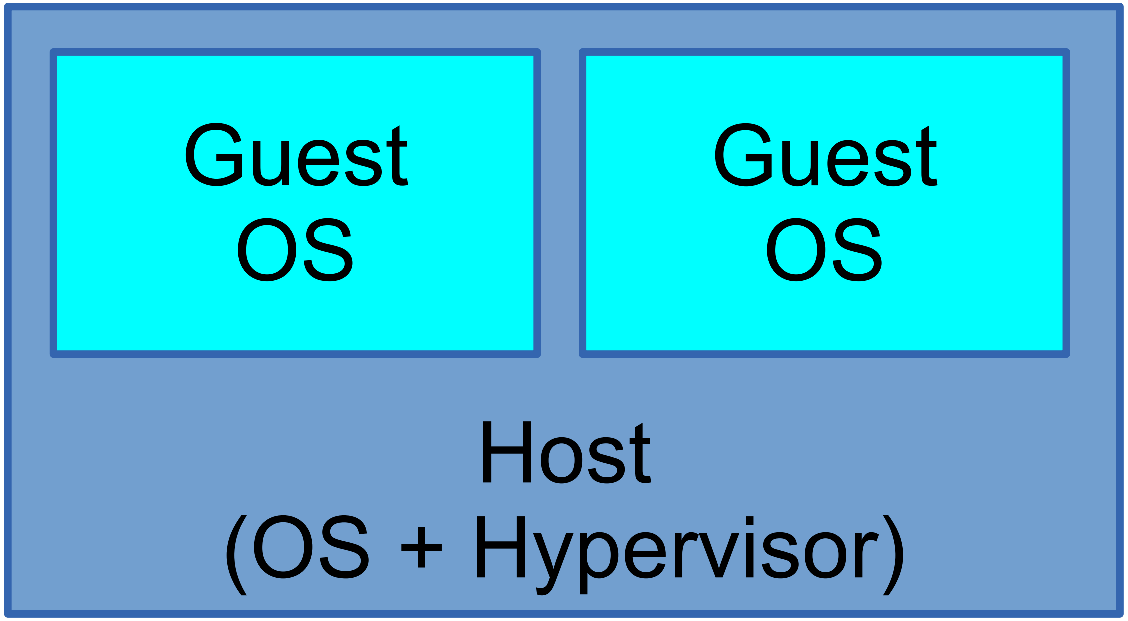

Virtual machine 1
Virtual machine 2
user process
user process
user process
user process
user process
user process
syscall API and libraries
syscall API and libraries
emulated hardware
filesystem
network stack
Kernel
device driver
device driver
device driver
device driver
filesystem
network stack
Kernel
device driver
device driver
device driver
device driver
SATA interface
Network interface
Graphics Card
USB, Serial etc
SATA interface
Network interface
Graphics Card
USB, Serial etc
HYPERVISOR
SATA interface
Network interface
Graphics Card
USB, Serial etc
C:>

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 interfaces
- 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 is a component of 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

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

Summary

- Virtualization can make better use of your hardware by emulating more machines than you really have
- The emulated environment is provided by a hypervisor
- The hypervisor (host) lets you start up virtual machines (guests) each with its own operating system and emulated devices
- Guest hardware emulated using resources on the host

KVM and libvirt

NSRC

Server virtualization

- Scenario: running VMs remotely on a server in a data centre
- We are more interested in:
  - Reliability
  - Performance / low overhead
  - Ability to grow to large clusters (without being tied into huge license fees!)
  - Remote management, scripted management
  - Features like machine migration

Choosing a hypervisor

- There are many hypervisor options out there
- Market has forced them all to be "free" - at least to begin with
- Commercial products: you pay later (heavily!) when you need to run clusters of machines

Our choice: KVM

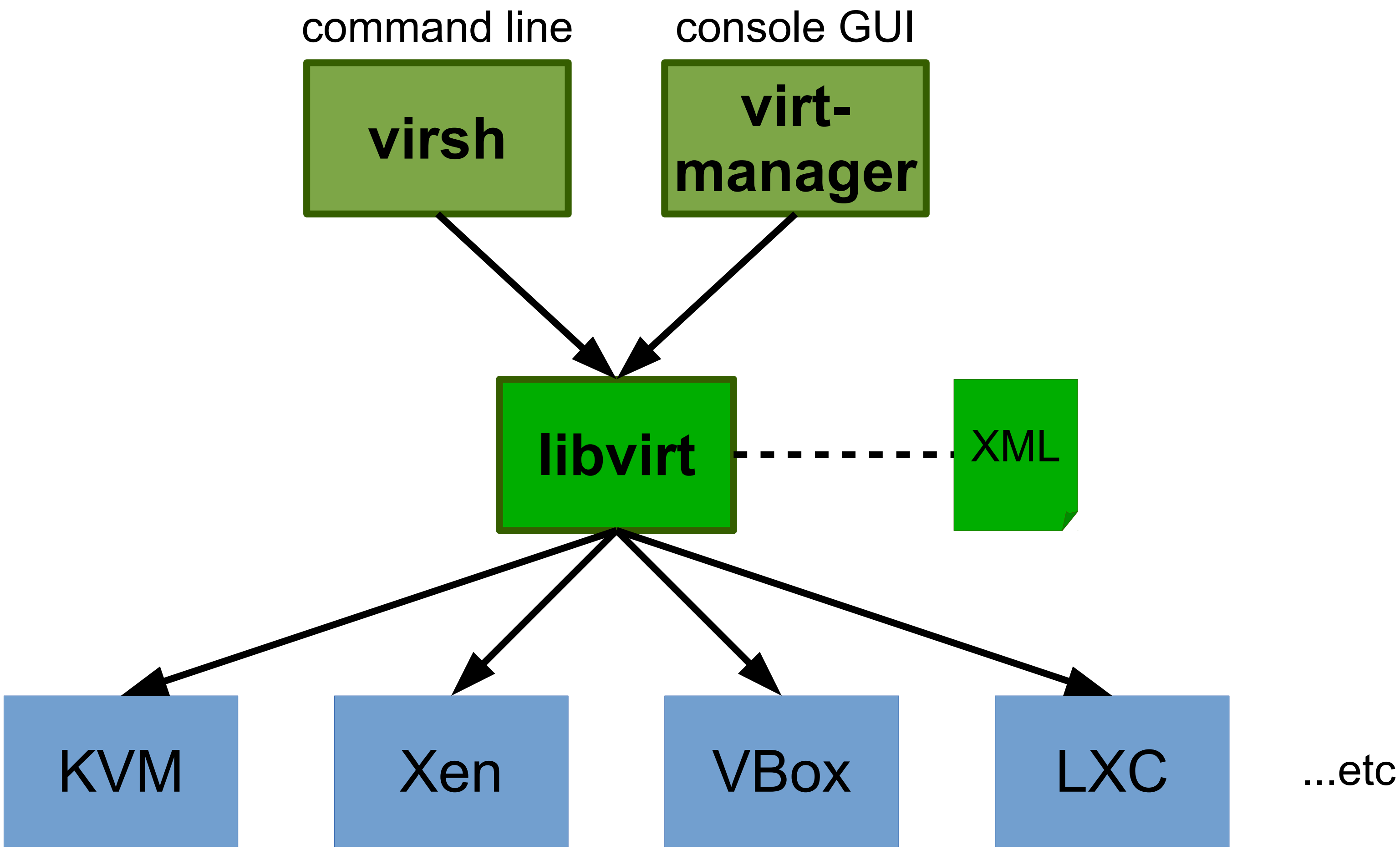
- KVM = Kernel Virtual Machine
- A hypervisor built into the Linux Kernel, based on QEMU
- It's where it's all happening!
  - Many, many projects using KVM
  - KVM gets all the development attention
- It *requires* VT-x or AMD-V to run
- The host must be Linux
  - but not necessarily the guests, of course

KVM is very simple

- Each VM is just a userland process
- Can run it directly from the command line
  - `kvm -cdrom /path/to/image.iso`
    - starts a VM, ISO image attached
- Painful to track all the command line options for RAM, disk drives, network interfaces, etc etc
- So you need something to remember all your VMs and how to start them

libvirt

- Red Hat's framework for managing hypervisors



libvirt

- API to create, modify, and control VMs
  - Terminology: VM is called "guest domain"
- Each VM has an XML file with all settings
  - Easy to read, backup and duplicate
  - Relatively easy to modify
- Two front-ends
  - virsh: command-line
  - virt-manager: X11 GUI
- Various other projects interface with libvirt API

libvirt limitations

- No simple web interface included
- virt-manager *can* talk to remote hypervisors, but virt-manager itself only runs under Linux
  - so you may end up running a VNC desktop into the Linux box, just to run virt-manager there
- XML format is unique to libvirt
  - different to OVF, VMX etc
  - too hard to write from scratch!
- libvirt's storage management is difficult

virsh commands (1)

- `virsh list [--all]`
  - list running (or all) VMs
- `virsh start` *`VM`*
  - start the VM named *VM*
- `virsh shutdown` *`VM`*
  - shutdown VM (properly)
- `virsh destroy` *`VM`*
  - kill a VM (power off)
- `virsh console` *`VM`*
  - connect to the serial console of a VM
- `virsh define` *`FILE`*
  - create VM definition from this XML file
- `virsh undefine` *`VM`*
  - erase the machine definition (danger!)

Easily scriptable - e.g. easy to write a shell loop to start or stop a bunch of VMs

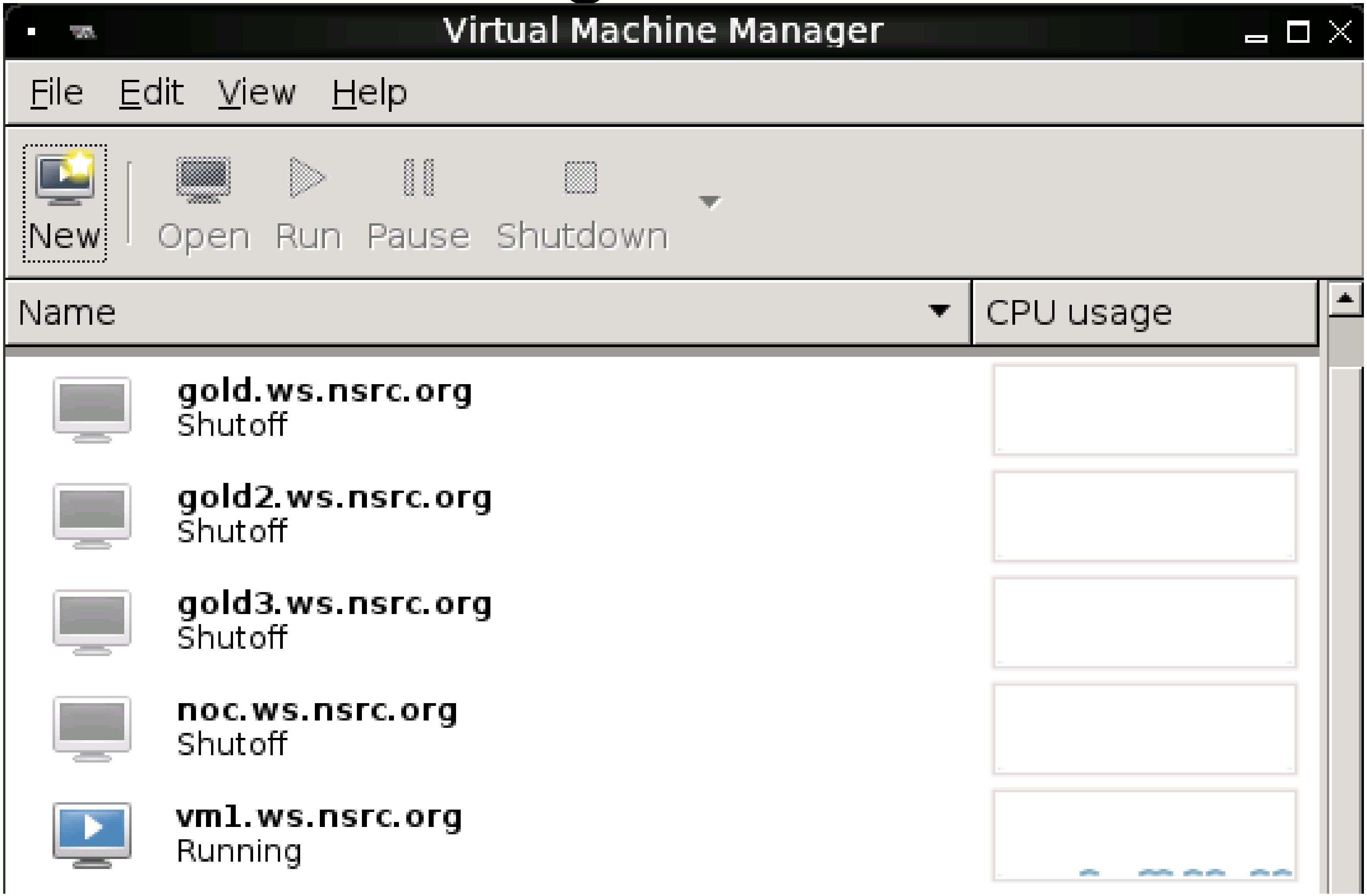
virsh commands (2)

- `virsh dumpxml VM`
  - show the XML

- `virsh edit VM`
  - open XML in editor

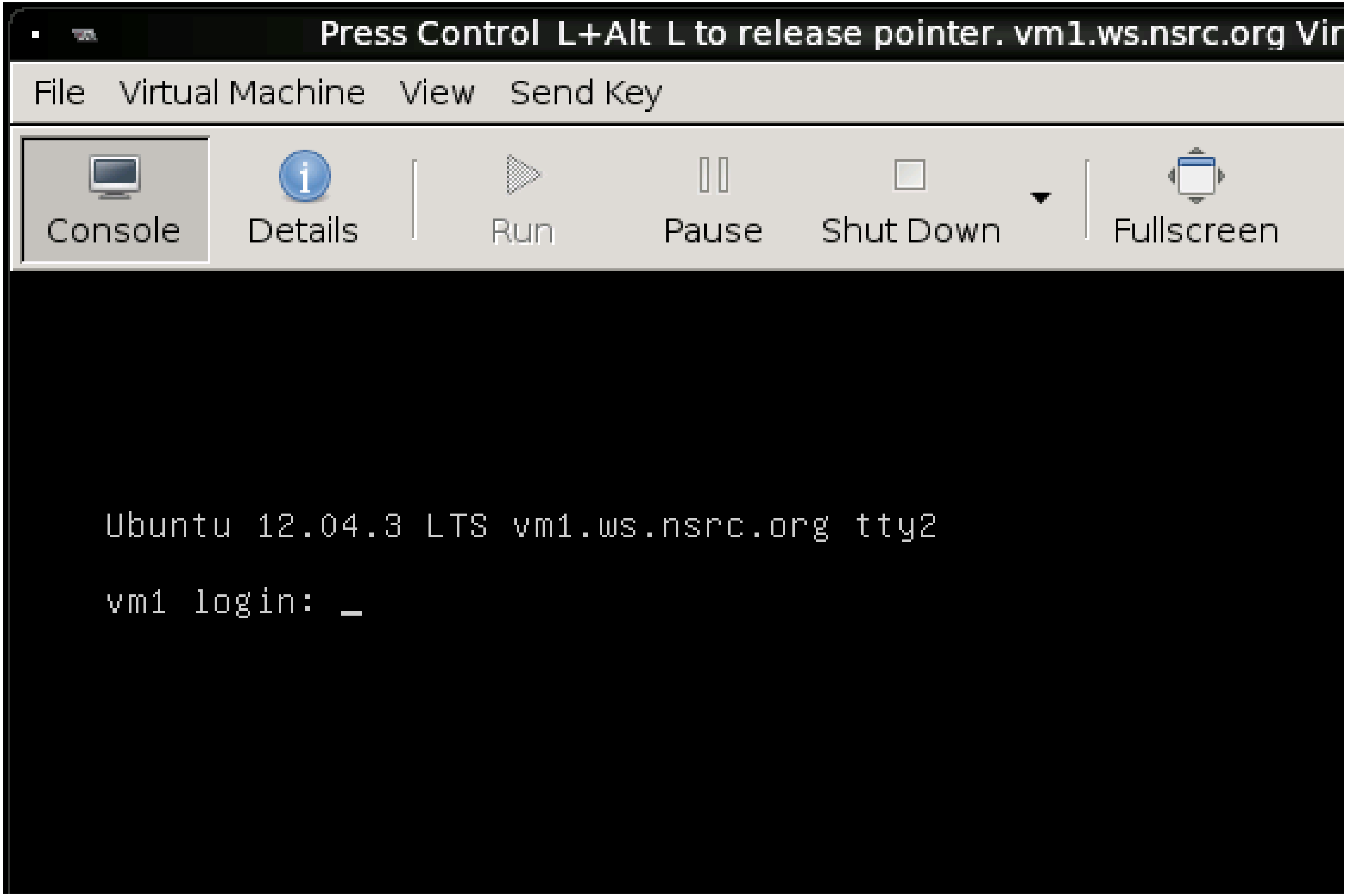
```
<domain type=’kvm’>
  <name>noc.ws.nsrc.org</name>
  <uuid>4641a945-abab-1c0b-0fb0-2db681c28130</uuid>
  <memory>1048576</memory>
  <currentMemory>1048576</currentMemory>
  <vcpu>1</vcpu>
  <os>
    <type arch=’x86_64’ machine=’pc-1.0’>hvm</type>
    <boot dev=’hd’/>
  </os>
...
```

virt-manager - main view



Name	CPU usage
gold.ws.nsrc.org Shutoff	
gold2.ws.nsrc.org Shutoff	
gold3.ws.nsrc.org Shutoff	
noc.ws.nsrc.org Shutoff	
vm1.ws.nsrc.org Running	

virt-manager - console view

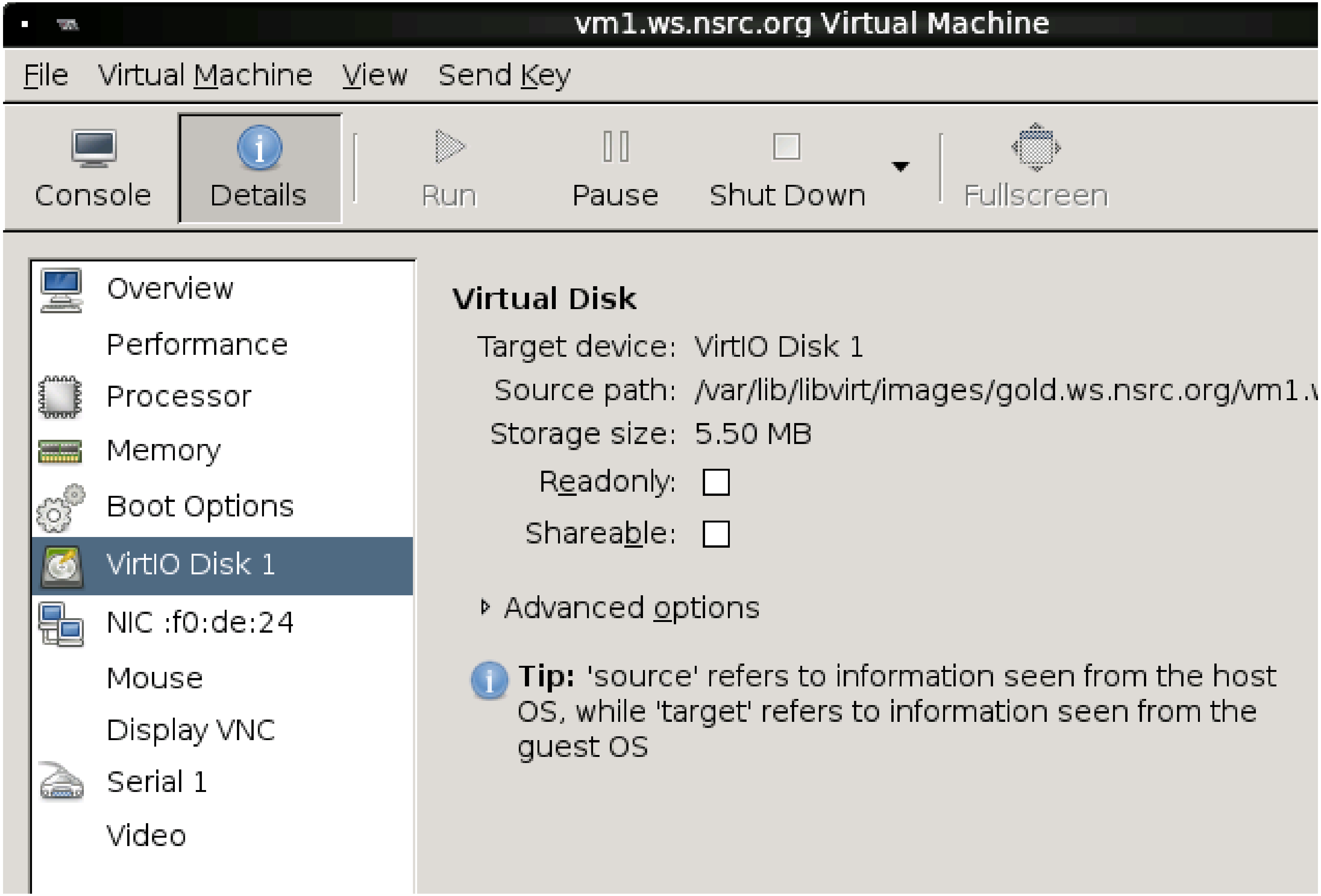


```
Ubuntu 12.04.3 LTS vm1.ws.nsrc.org tty2

vm1 login: _
```

NOTE: Press Left-CTRL and Left-ALT together to release the keyboard and mouse

virt-manager - VM details/settings



Virtual Disk

Target device: VirtIO Disk 1

Source path: /var/lib/libvirt/images/gold.ws.nsrc.org/vm1.v

Storage size: 5.50 MB

Readonly: □

Shareable: □

Advanced options

Tip: 'source' refers to information seen from the host OS, while 'target' refers to information seen from the guest OS

Summary

- KVM is a free, open-source hypervisor for Linux
- All major Linux distros support KVM
- libvirt is a simple admin interface
  - starts and stops the hypervisor
  - stores hypervisor settings in XML file
  - virsh: command line
  - virt-manager: GUI comparable to VirtualBox (albeit not as polished)

VMBuilder

- VMBuilder is a Python based software package for creating virtual machine images of Linux.
- Maintained by Ubuntu
- Supports building Xen, VirtualBox, VMware, KVM and Amazon EC2 images.
- Can be configured with default options for new images in /etc/vmbuilder.cfg