Containers

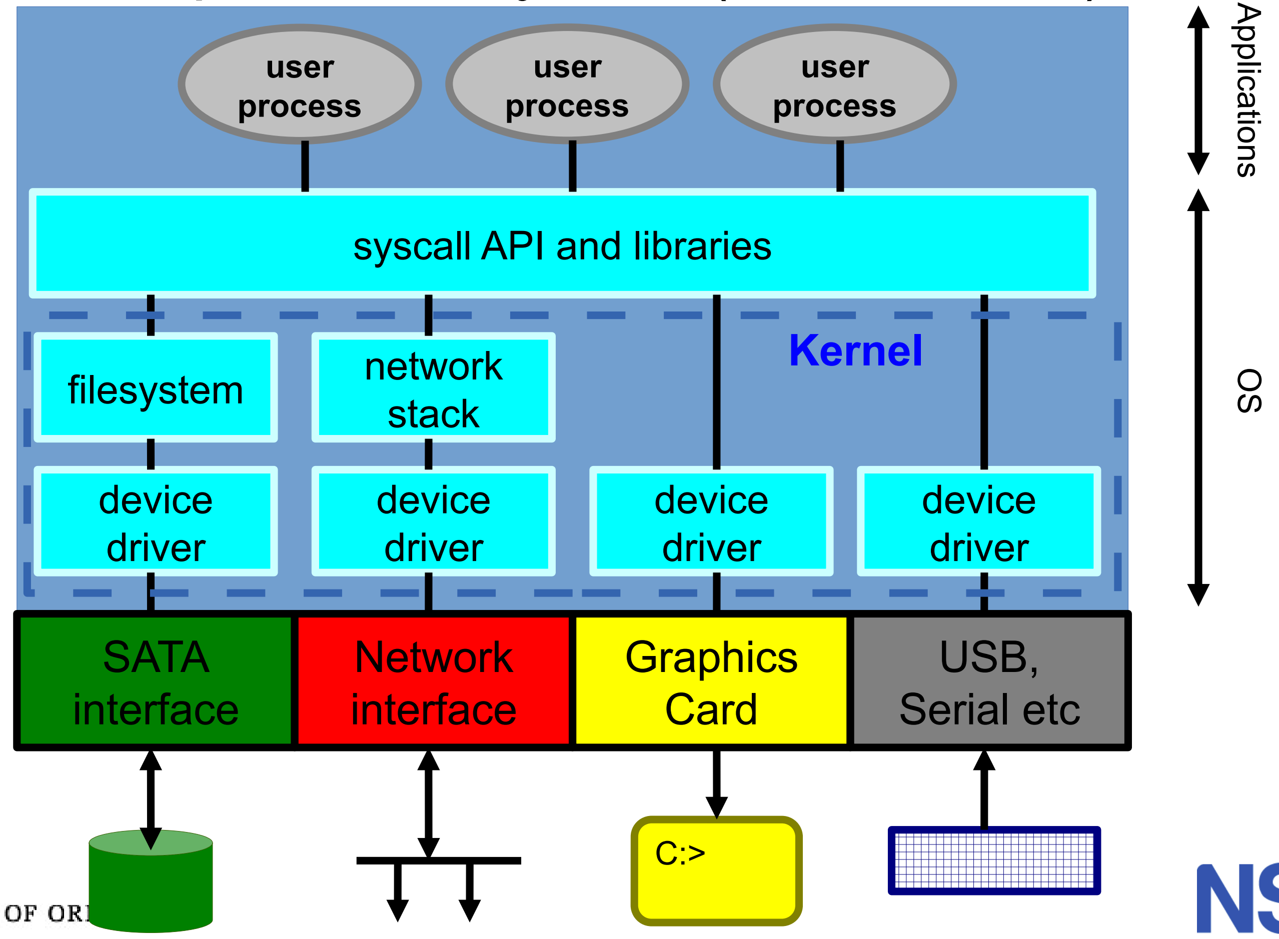
UNIVERSITY OF OREGON

NSRC Network Startup Resource Center

What are containers?

- A way of isolating processes from each other
  - Processes in one container cannot see processes in another container
- Each container has its own filesystem
  - which is just a *subdirectory* in the host filesystem
- Each container can have its own network stack and hostname
- Uses modern kernel isolation mechanisms
  - (cgroups, chroot, PID namespaces, network namespaces… the details aren't important)

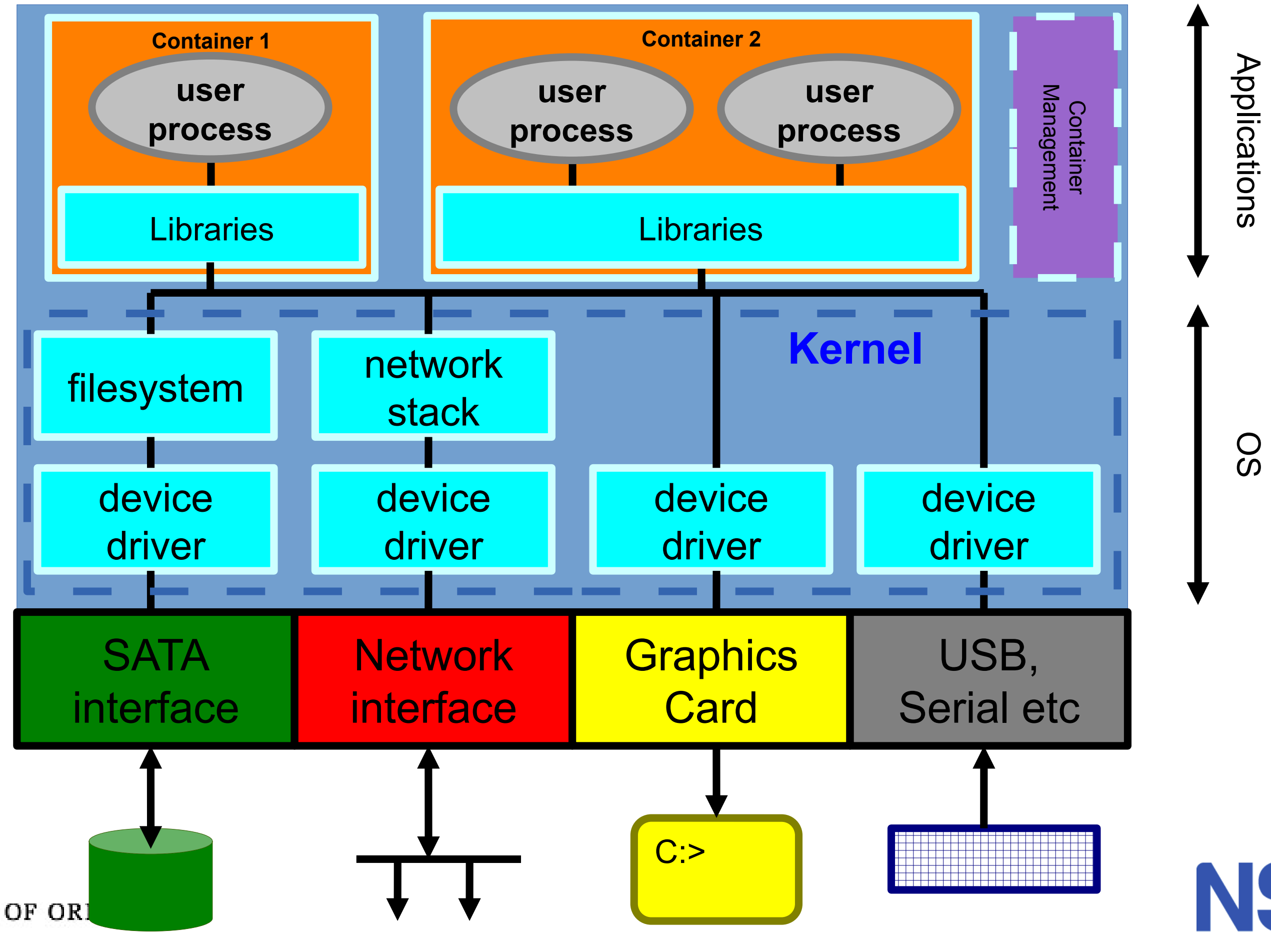
Recap: normal system ("bare metal")



UNIVERSITY OF ORE

NSRC
Network Startup Resource Center

Containers



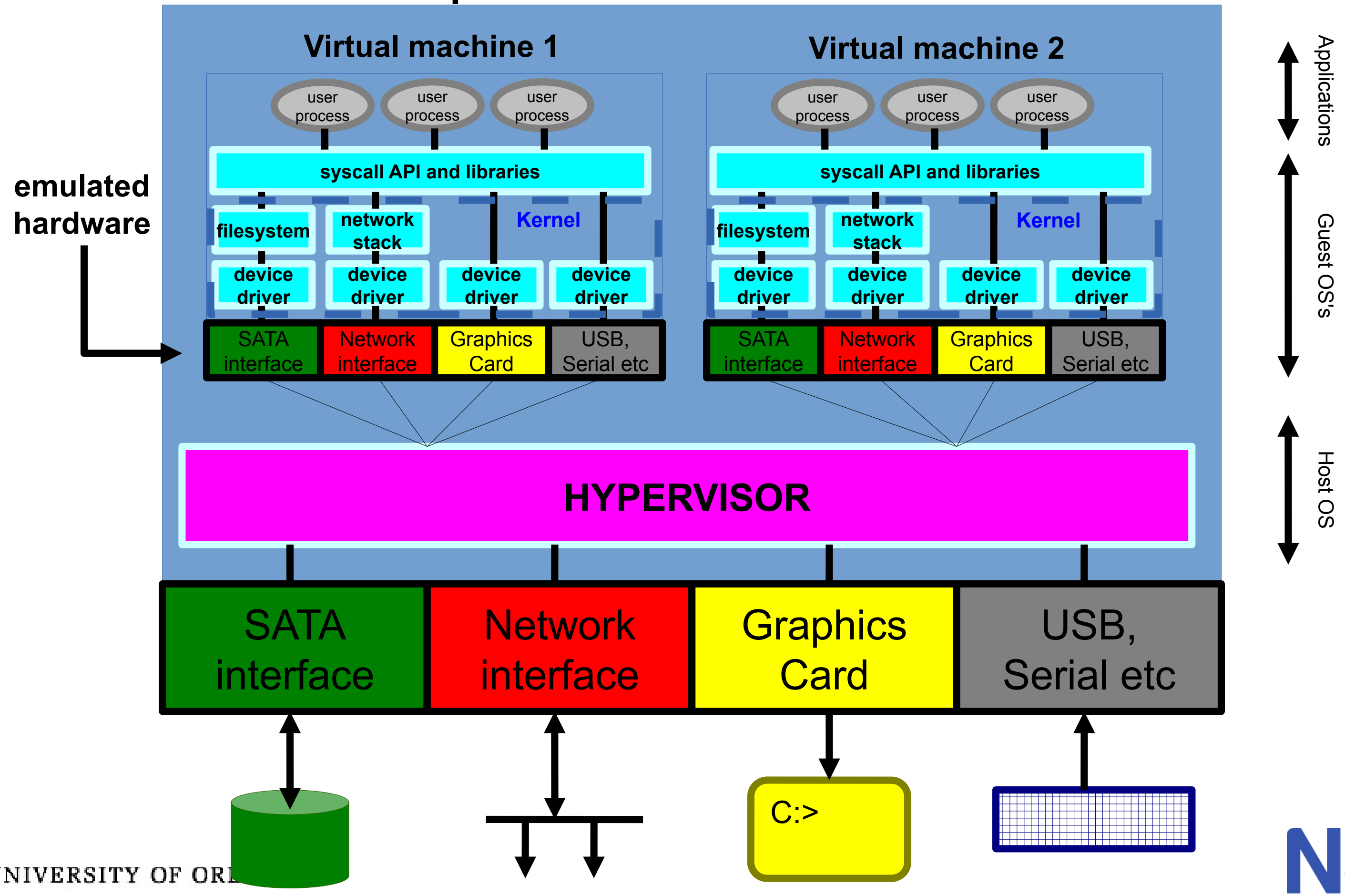
UNIVERSITY OF ORE

NSRC Network Startup Resource Center

Advantages of containers

- There is only one kernel and OS running
- As a result, containers are very lightweight
  - It is possible to run hundreds of containers on a single host
- Containers share the host's RAM
  - although you can apply per-container RAM and CPU limits too
- Much lower overhead than full virtualization
- Can be an alternative to VMs for some use cases

Compare: Virtual Machines



UNIVERSITY OF ORE

NSRC Network Startup Resource Center

Disadvantages of containers

- Container uses the host's kernel
  - You cannot have a Windows container running on a Linux host; you need full virtualization for that
  - However, container has its own complete filesystem
  - Can have a Fedora container running on an Ubuntu host, for example
- No live migration
  - Theoretically possible with "CRIU", but there are so many limitations it almost never works
  - Container stop/start is very quick anyway

Container security

- It ought to be hard to break out of a container
- But it's easier than breaking out of a VM
- So if you're running *untrusted* workloads (e.g. managed by external users) then safer to stick to VMs
- Another option is to run a number of containers inside a single VM

Container security options

- *Unprivileged* containers map user IDs to safe ones
  - e.g. uid 0 (root) inside container is uid 100000 outside the container
  - very good security; if a user breaks out, very little damage possible
  - container can't do things like mount block devices
- *Nesting* relaxes some constraints to allow containers in containers
  - not a major issue when used with unprivileged containers
- *Privileged* containers run same uids, e.g. root is root
  - dangerous: root in container can mess with the host.
- *Privileged* + *Nesting* = very dangerous indeed

Container management

- You need some software to manage containers
  - create, start, stop, etc
- These fall into two broad categories

Two types of container platform

- "System containers" (lxc, lxd / incus, OpenVZ, …)
  - Container looks like a VM, and is managed like a VM
  - Container filesystem has a complete OS image
  - Connect to it via SSH, install multiple software packages, etc
- "Application containers" (docker, podman, kubernetes)
  - An application is bundled with all its dependencies in one big image
  - Container starts from a copy of this image
  - Generally one container per application
  - You don't manage or upgrade the container; you throw it away and recreate it from an updated image

Proxmox CT

- Proxmox system containers
- Uses lxc behind the scenes
- Creates a block storage volume for each container
  - Advantage: can use any storage, including Linstor or Ceph
  - Disadvantage: fixed storage size
- Provides some downloadable, ready-to-run images
- Manage them alongside your VMs in the same UI
- These containers run *directly on your Proxmox host* and therefore shouldn't be used for untrusted workloads

incus (fork of lxd)

- Lightweight, can easily be installed inside a VM
  - for macOS see colima
- Powerful, primarily CLI/API based
  - amenable to scripting
  - remotely controllable, CLI uses the API
- Many different ready-to-run images provided
- Can use host ZFS or btrfs for efficient snapshots and replication
- Can also run VMs and make the tea

Proxmox CT lab

UNIVERSITY OF OREGON

NSRC
Network Startup Resource Center