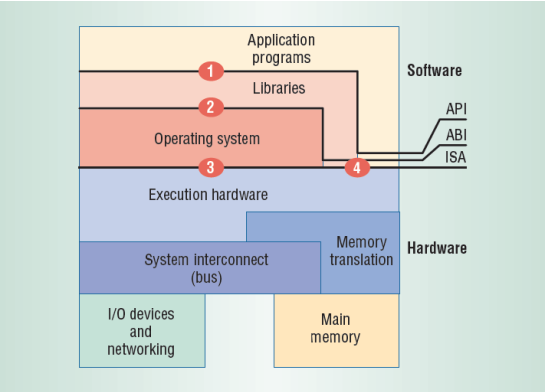

School of Computer Science & Engineering

**COMP3891/9283 Extended Operating Systems**

2026 T2 Week 01

# Process Abstractions I: Containers

@GernotHeiser

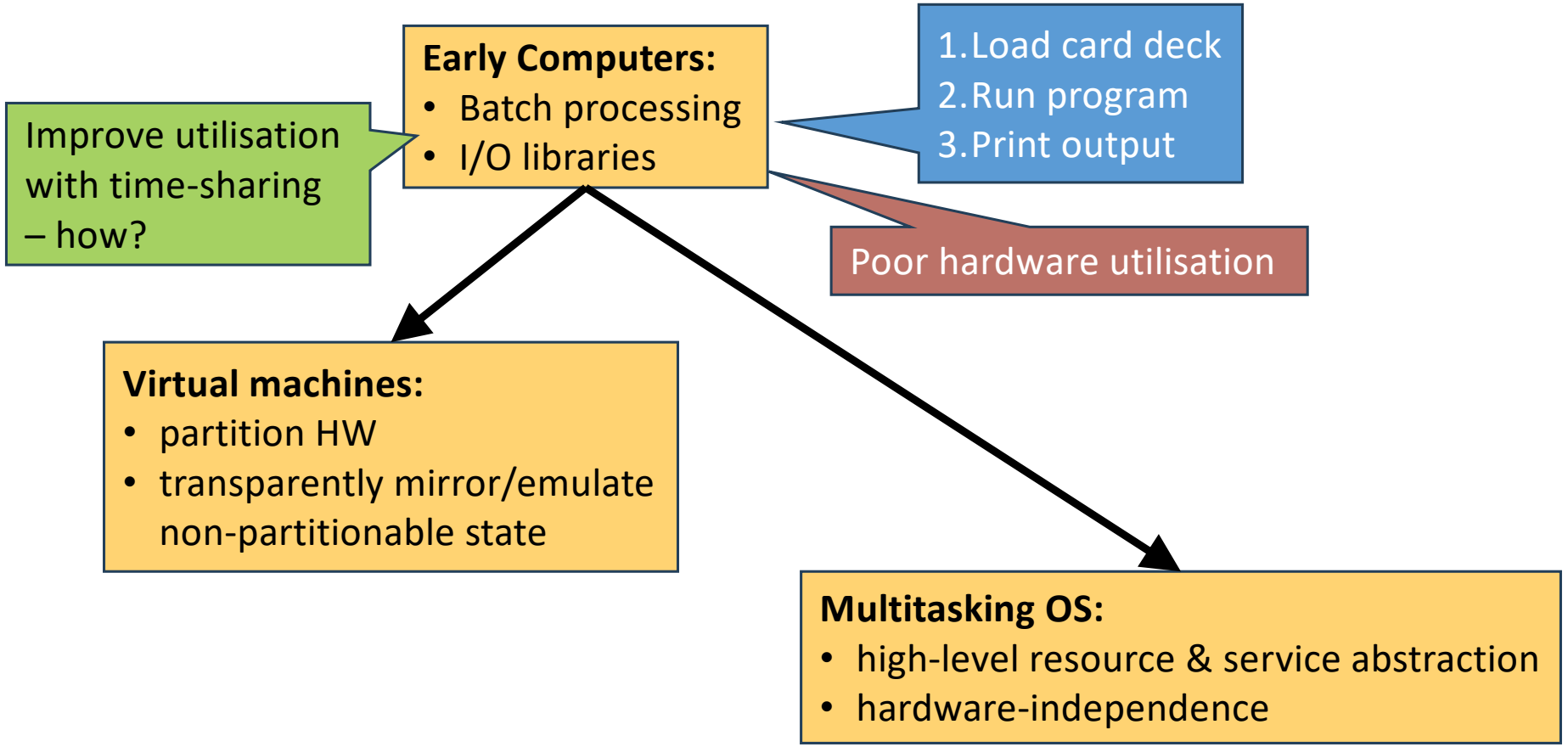


# Copyright Notice

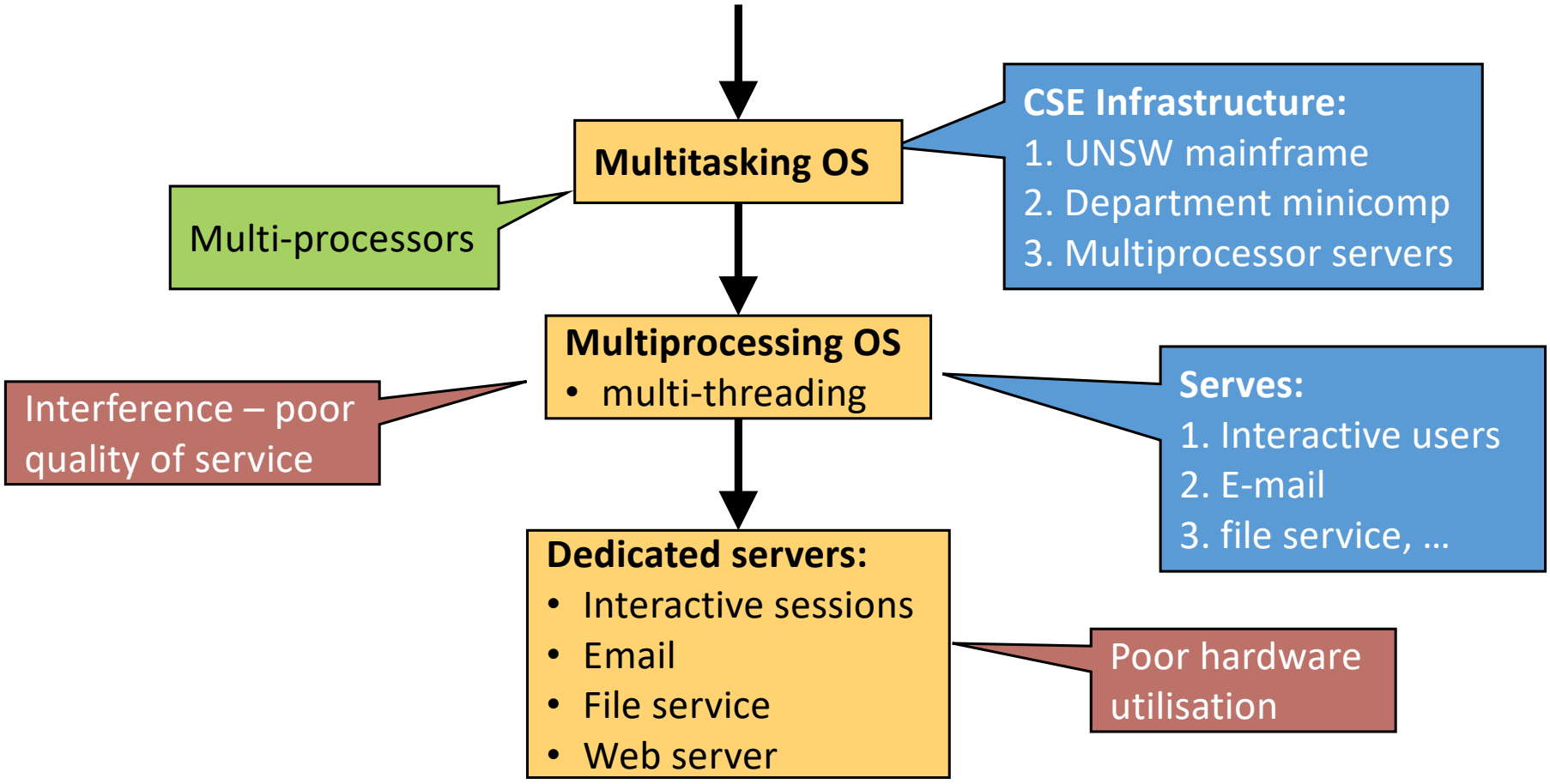
# Learning Outcomes

- An understanding of processes as a more general concept than just an executing program.
- A high-level understanding of containers as processes with strong resource isolation.

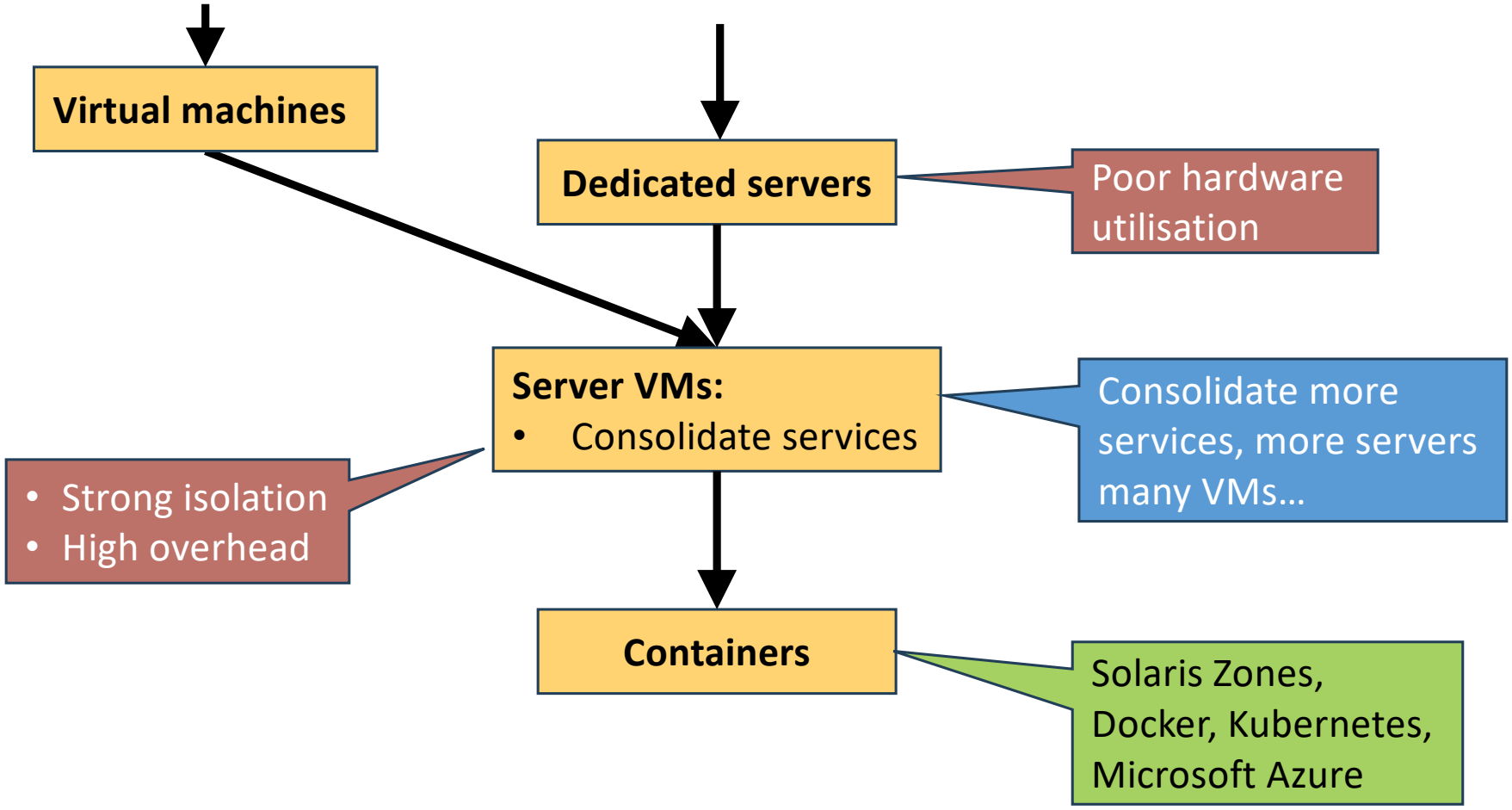
# A Brief (& Incomplete) History of Processes



# A Brief (& Incomplete) History of Processes



# A Brief (& Incomplete) History of Processes



# Practical Barriers to Server Consolidation

- Server-class applications written assuming a machine to itself
  - Clashing network ports
  - Clashing user IDs
  - Hard-coded log/config file locations
- Security & quality-of service requires isolation
  - File system
  - Memory
  - CPU (scheduling)

# Retrofit Isolation into OS

"Applications can be run within [containers] with no changes, and with no significant performance impact for either the performance of the application or the base operating system."

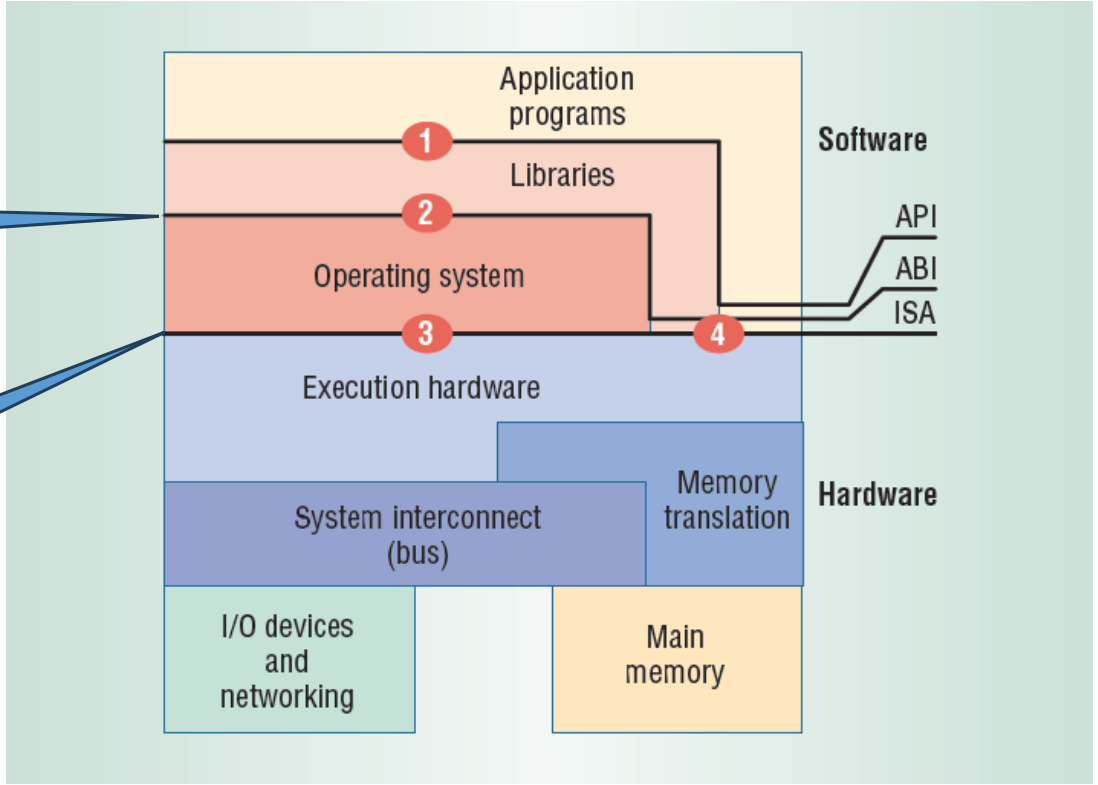
[Price&Tucker, LISA'04]

Virtualises OS API rather than processor ISA

# Interfaces

Virtualise ABI: Containers

Virtualise ISA: Virtual Machine



# Design Requirements

- Basic process isolation:
  - A process in one container cannot locate, examine, or signal a process in another container.
- Each container has at least one logical network interface:
  - Applications running in distinct containers cannot observe the network traffic of the other containers even though their respective streams of packets travel through the same physical interface.
- Each container is provided a disjoint portion of the file system hierarchy, to which it is confined.

# Linux Mechanisms: **chroot**

```
/  proc/
   bin/
   home/
   cont/  1/  proc/
   …          bin/
              home/
              …
          2/
          3/
          …
```

```
/  proc/
   bin/
   home/
   …
```

```
// create new process
// in new process:
chroot(/cont/1);
```

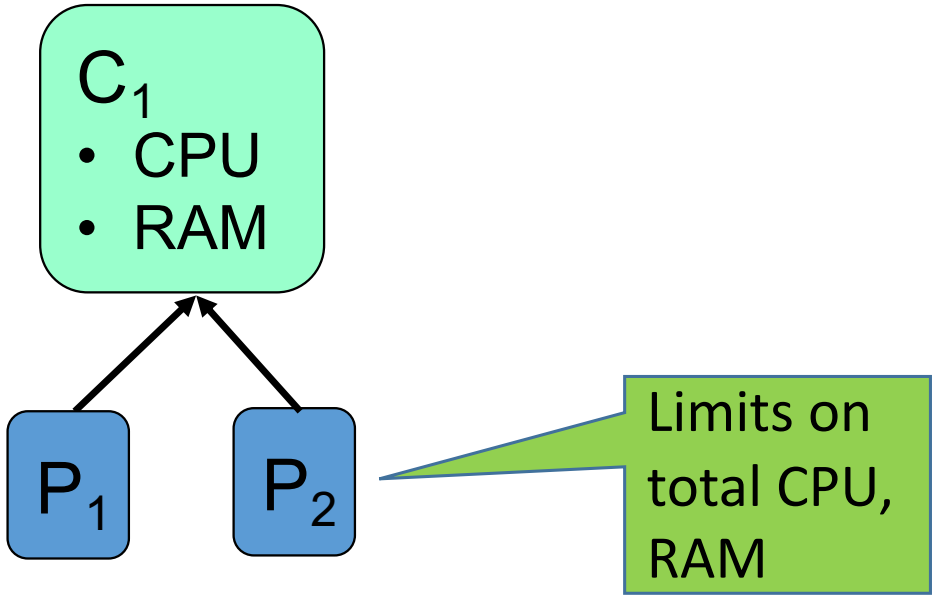
# Linux Mechanisms: **cgroups**

- Control groups: abstraction for tracking/limiting resources
  - Memory
  - CPU
  - Network bandwidth
  - I/O (storage) bandwidth

- Assign process to cgroup
- Child processes inherit parent's cgroup
- Control group shares set of resources

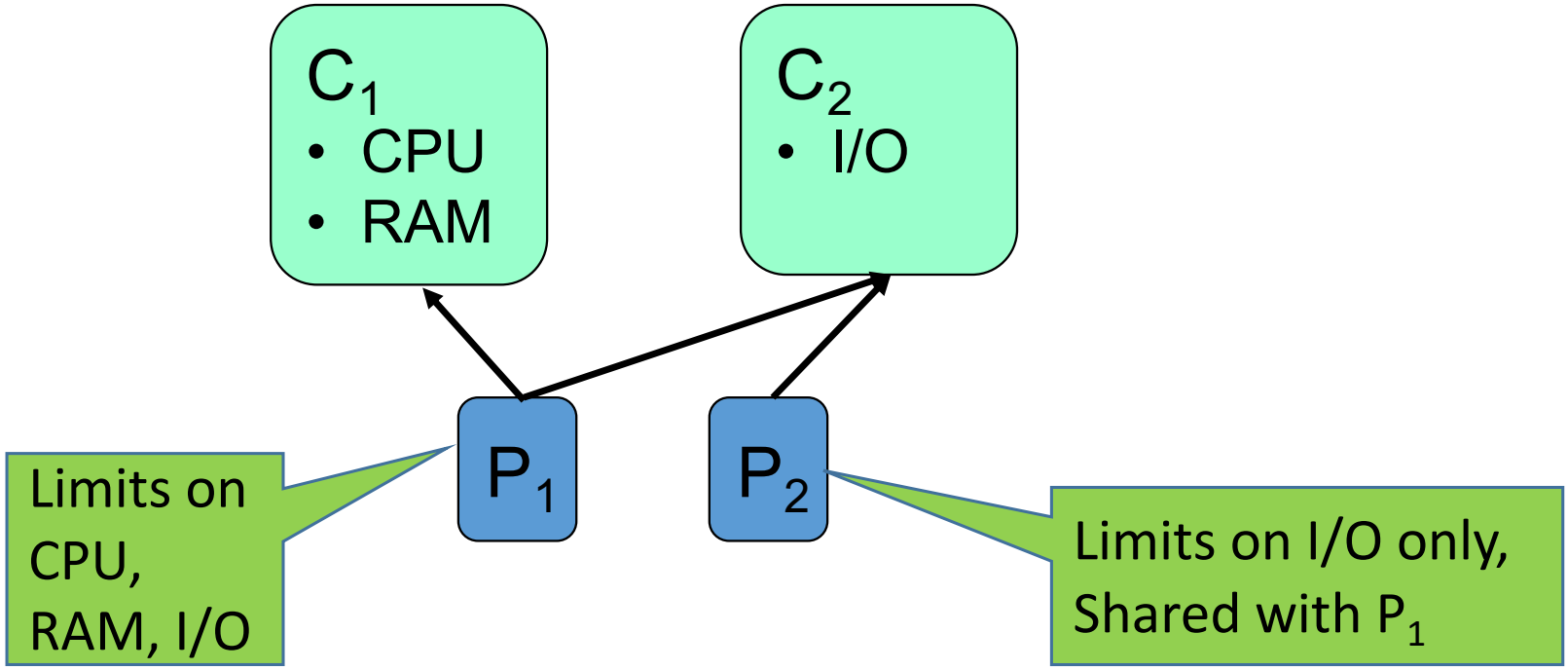
# Linux Mechanisms: **cgroups**

- Processes sharing cgroup share resources



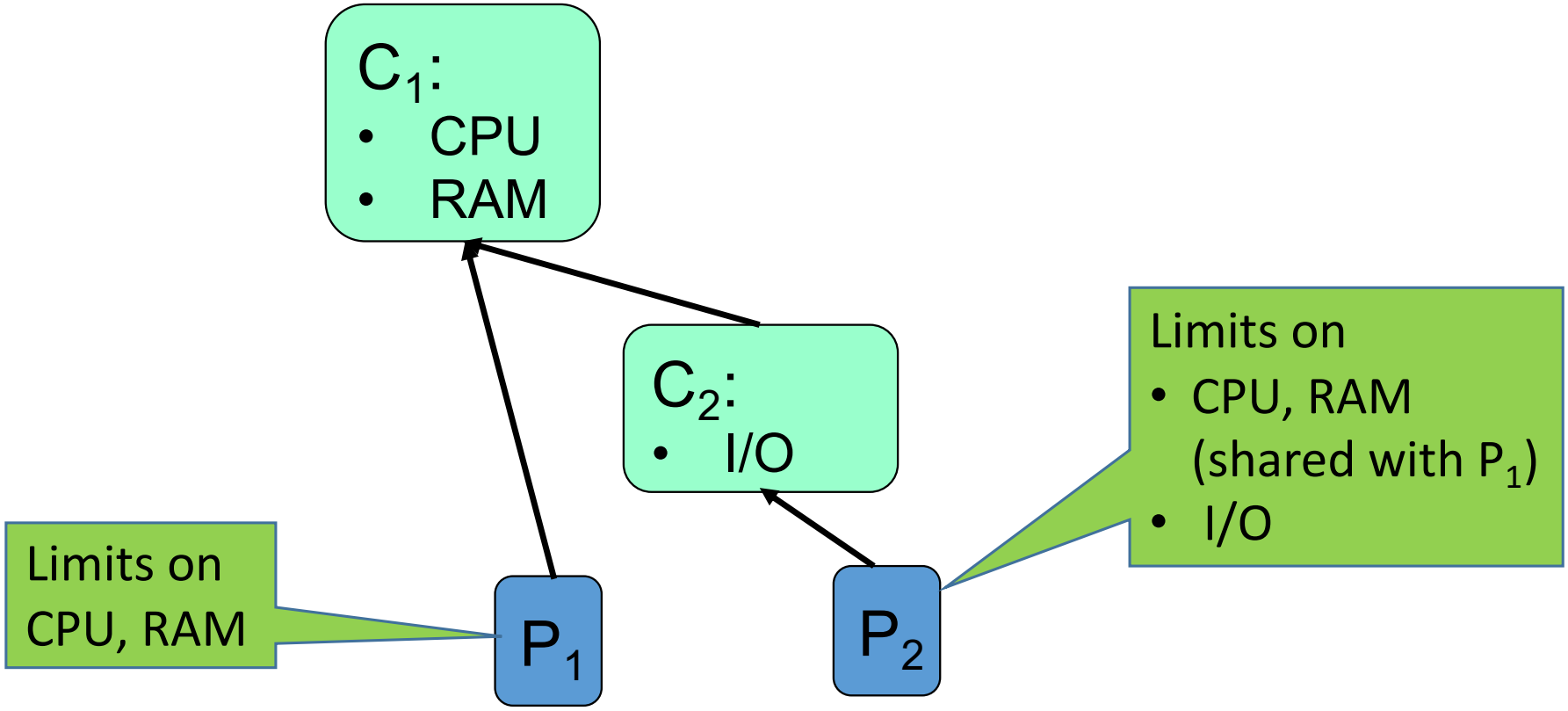
# Linux Mechanisms: **cgroups**

- Process can be associated with multiple cgroups



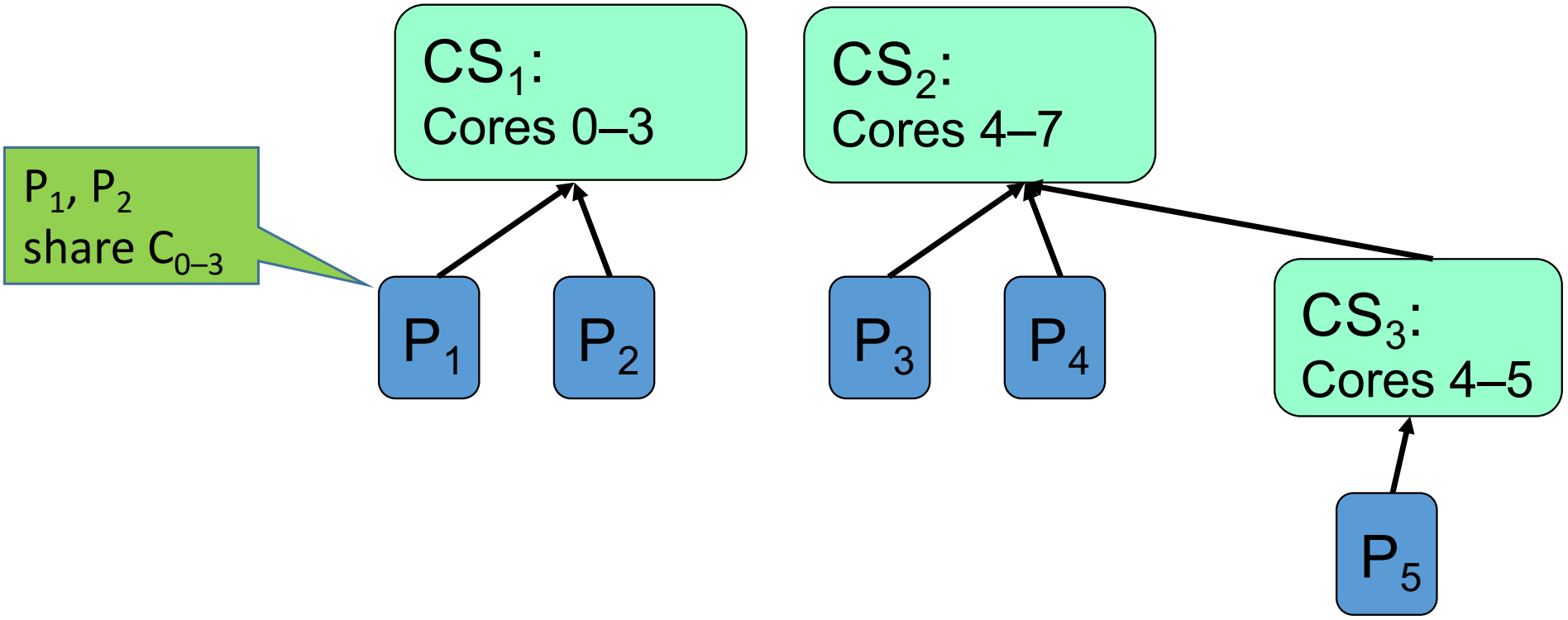
# Linux Mechanisms: **cgroups**

- Cgroups can be hierarchical

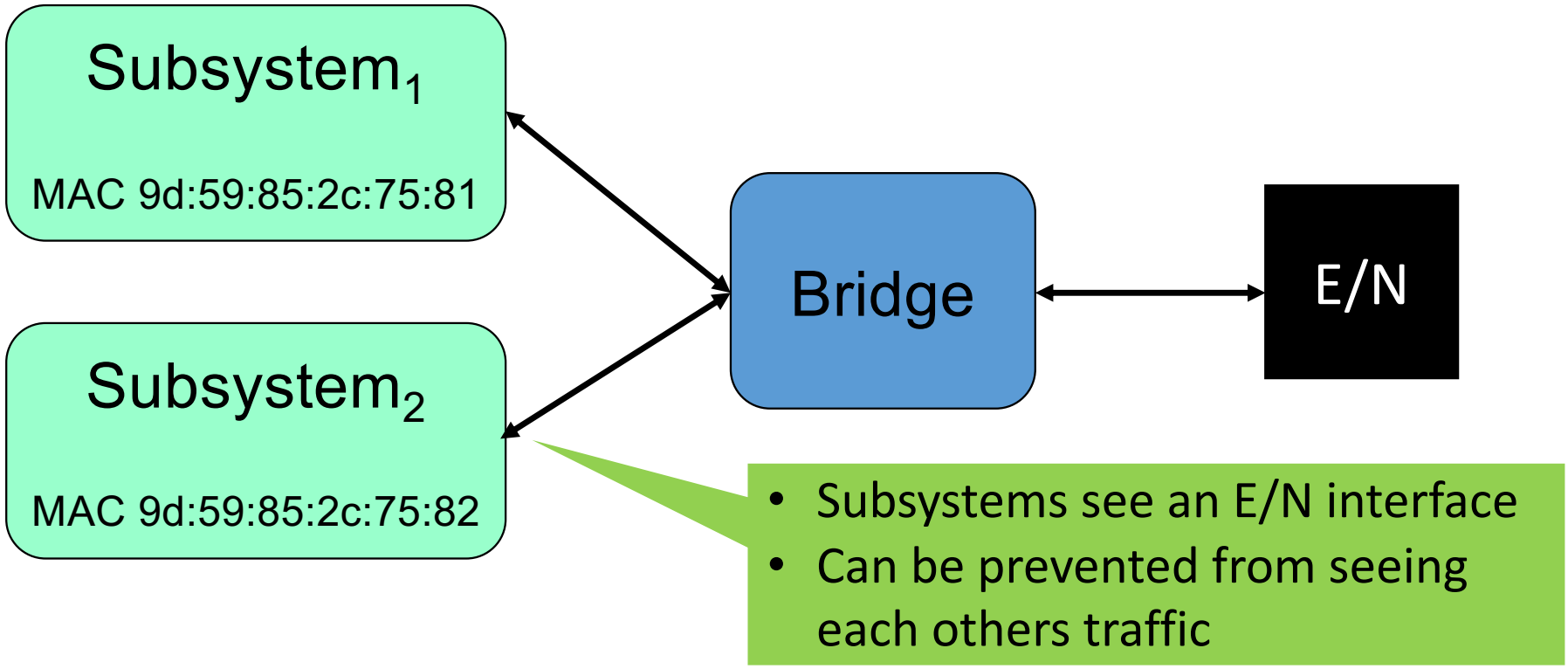


# Linux Mechanisms: **CPU Sets**

- CPU sets can be hierarchical



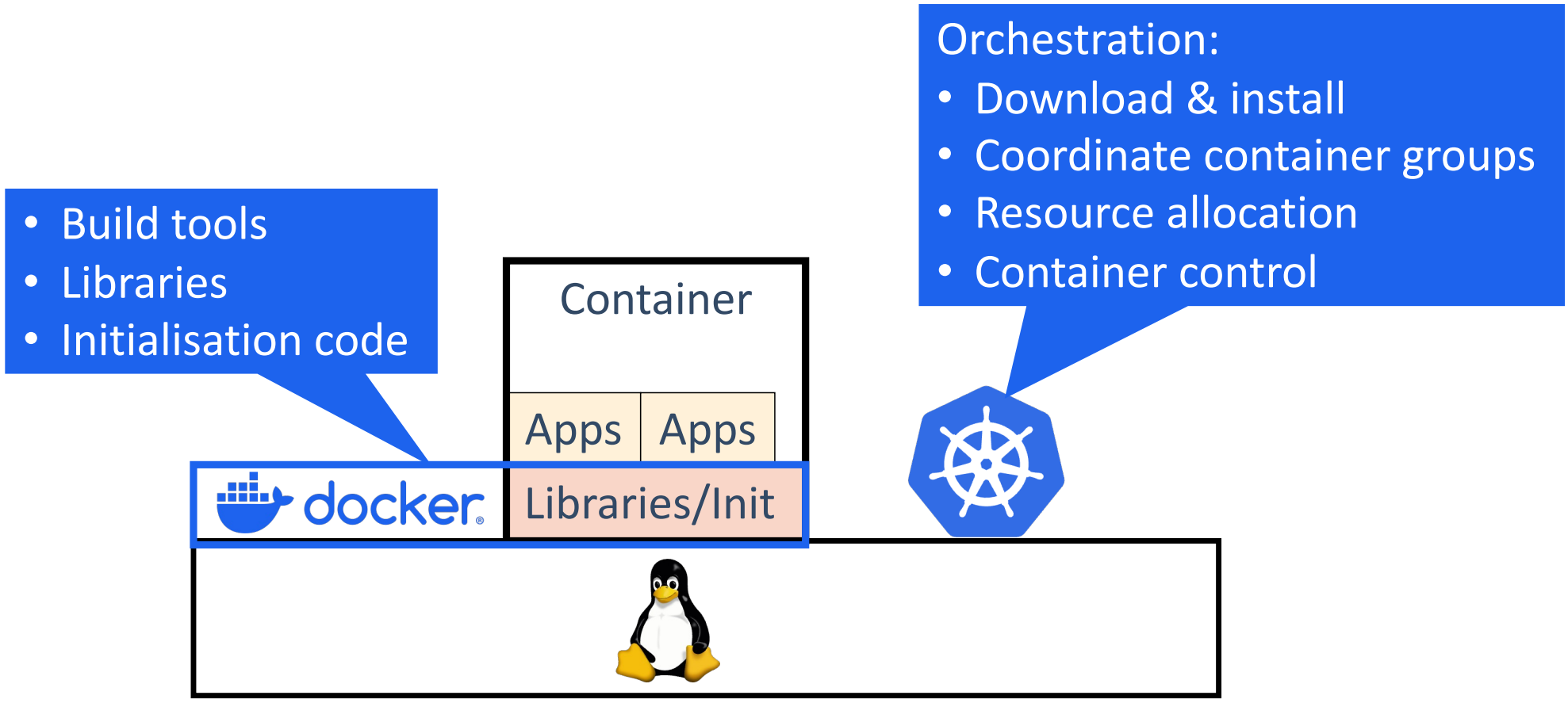
# Linux Mechanisms: **Ethernet Bridging**



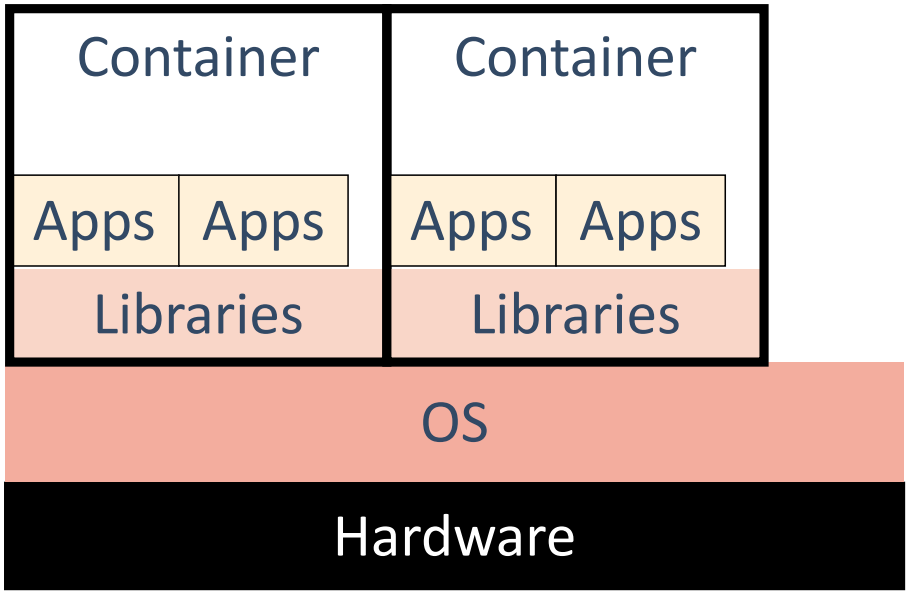
# Containers: Isolated Subsystems

- Separate name spaces
  - **`chroot`** for separating file systems, with controlled sharing
  - **`unshare`** for UIDs/PIDs
  - Ethernet bridges (among others) for separating networks
- cgroups for limiting CPU time, RAM, I/O bandwidth
- CPU sets for limiting access to cores
- “Capabilities” to restrict system calls (see later lecture)

# Containers on Linux: Docker & Kubernetes



# Containers: Limits of Isolation




Containers are (sets of) processes

- Separate data
- Shared OS & metadata

- OS compromises
- Global resources