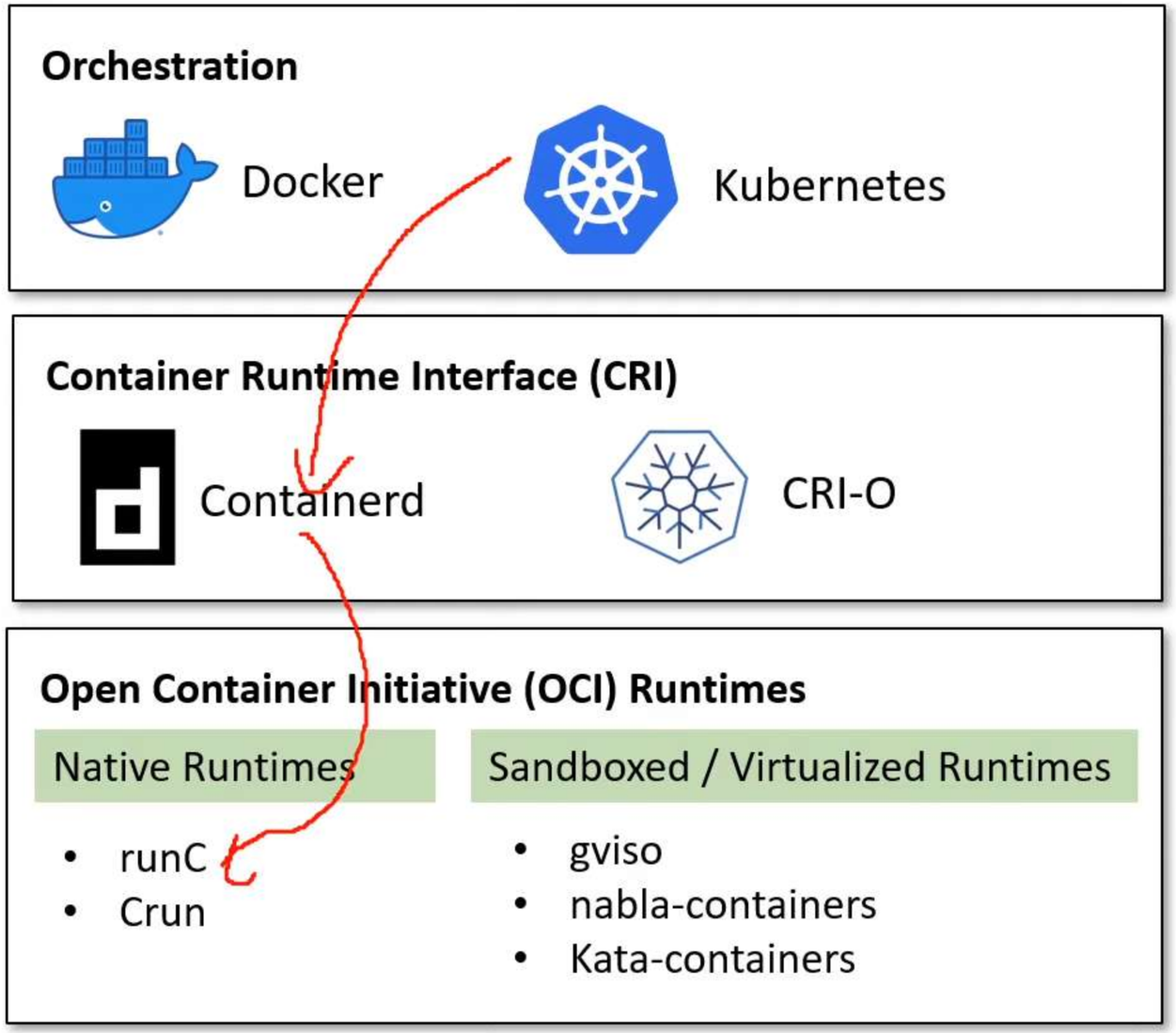
Container Runtime Interfaces



Orchestration systems will use a container runtime interface.

Container Runtime Interface allows you to run a variety of different container runtimes.
Eg. Push and pull images, supervise containers

Runtimes create and run containers.
The major difference between native and virtual is ***isolation***.

Virtualized can provide security benefits through isolation.

ContainerD

ContainerD is an industry-standard container runtime with an emphasis on simplicity, robustness and portability.

Docker extracted their container runtime in the project ContainerD and then donated it to CNCF

This includes Docker's functionality for executing containers, handling low-level storage and managing image transfers.

containerd makes it easier for projects like Kubernetes to access the low-level "Docker" elements they need. Instead of actually using Docker

Images you build with Docker aren't really "Docker images".

- Docker now uses the containerd runtime
  - your images are built in the standardized Open Container Initiative (OCI) format.

CRI-O

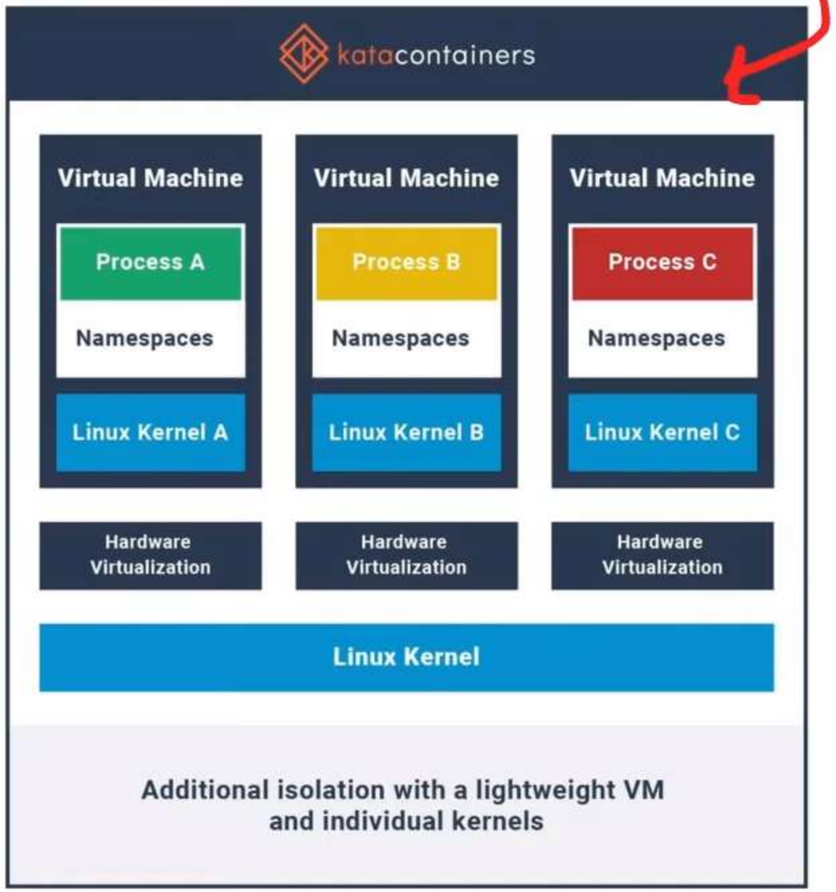
CRI-O is an implementation of the Kubernetes CRI (Container Runtime Interface) to enable using OCI (Open Container Initiative) compatible runtimes.

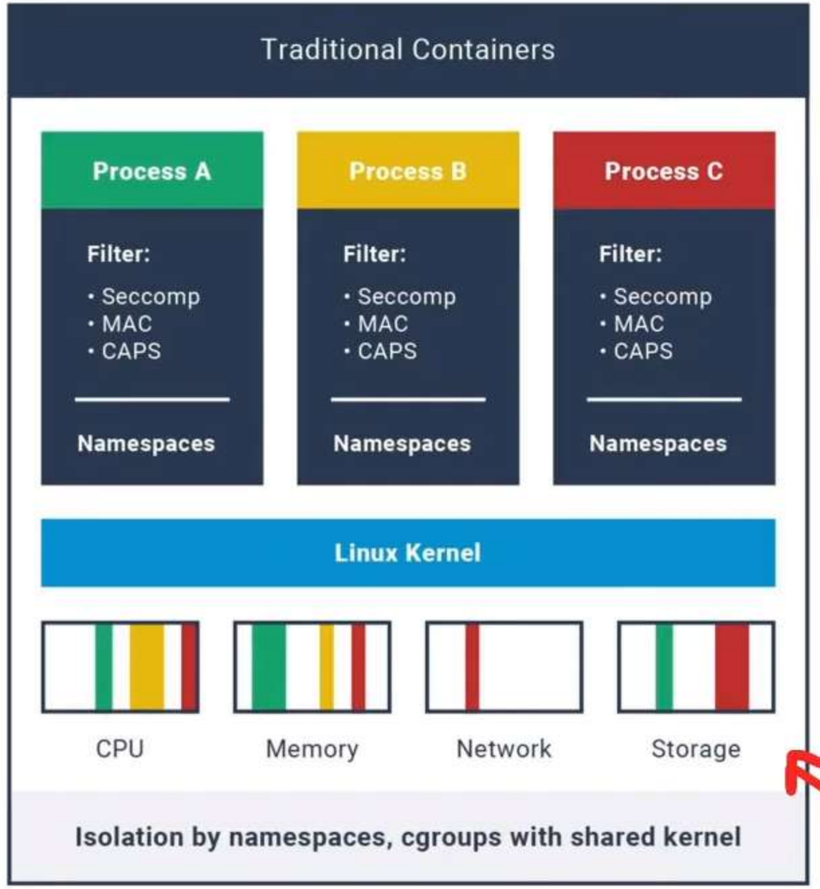
It is a **lightweight alternative to using Docker (containerd)** as the runtime for Kubernetes.

It allows Kubernetes to use any OCI-compliant runtime as the container runtime for running pods. Today it supports **runc** and **Kata Containers** as the container runtimes but any OCI-conformant runtime can be plugged in principle.

Container Runtimes

Virtualized Runtimes use **lightweight Vms** for isolation





Native Runtimes with use **cgroups** for isolation

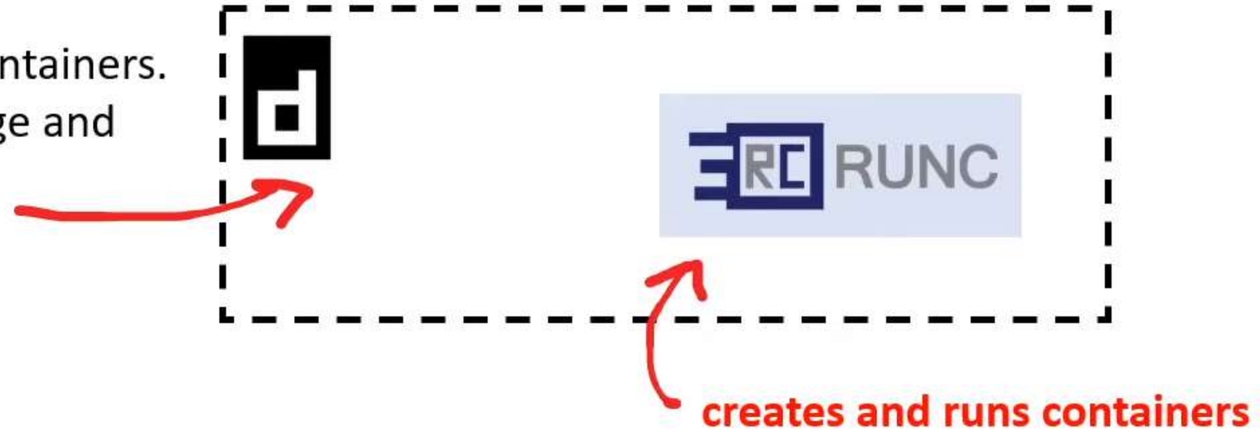
Container Runtimes

Runc is a low-level container runtime that **creates and runs containers**.

Runc would be used alongside a ContainerD or CRI-O

daemon process that manages and runs containers.

- pushes and pulls images manages storage and networking
- supervises the running of containers



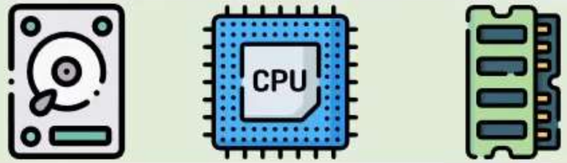
CGroups

What is a process?
A process is an instance of a running program on Linux

What is a CGroup?
Control groups (cgroups) allows you to group processes to apply different kinds of limitations:

- **Resource limiting —** groups can be set to not exceed a configured memory limit, which also includes the file system cache
- **Prioritization —** some groups may get a larger share of CPU utilization or disk I/O throughput
- **Accounting —** measures a group's resource usage, which may be used, for example, for billing purposes
- **Control —** freezing groups of processes, their checkpointing and restarting

Think of CGroups as a way to limit programs on linux
From overusing CPU, Memory or Storage.



The primary design goal for cgroups was to provide a unified interface to manage processes or whole operating-system-level virtualization, including **Linux Containers (LXC)**

Linux Containers

LXC (LinuX Containers) is a OS-level virtualization technology that allows creation and running of multiple isolated Linux virtual environments (VE) on a single control host.

VEs vs VMs

VE there is no preloaded emulation manager software as in a VM

In a VE, the application (or OS) is spawned in a container and runs with no added overhead, except for a usually minuscule VE initialization process

There is no hardware emulation, which means that aside from the small memory software penalty

LXC will boast bare metal performance characteristics because it only packages the needed application

Ves cannot be easily managed via neat GUI management consoles and they don’t offer some other neat features of VM’s such as IaaS setups and live migration.