CS-580K/480K Advanced Topics in Cloud Computing

4. Containerization

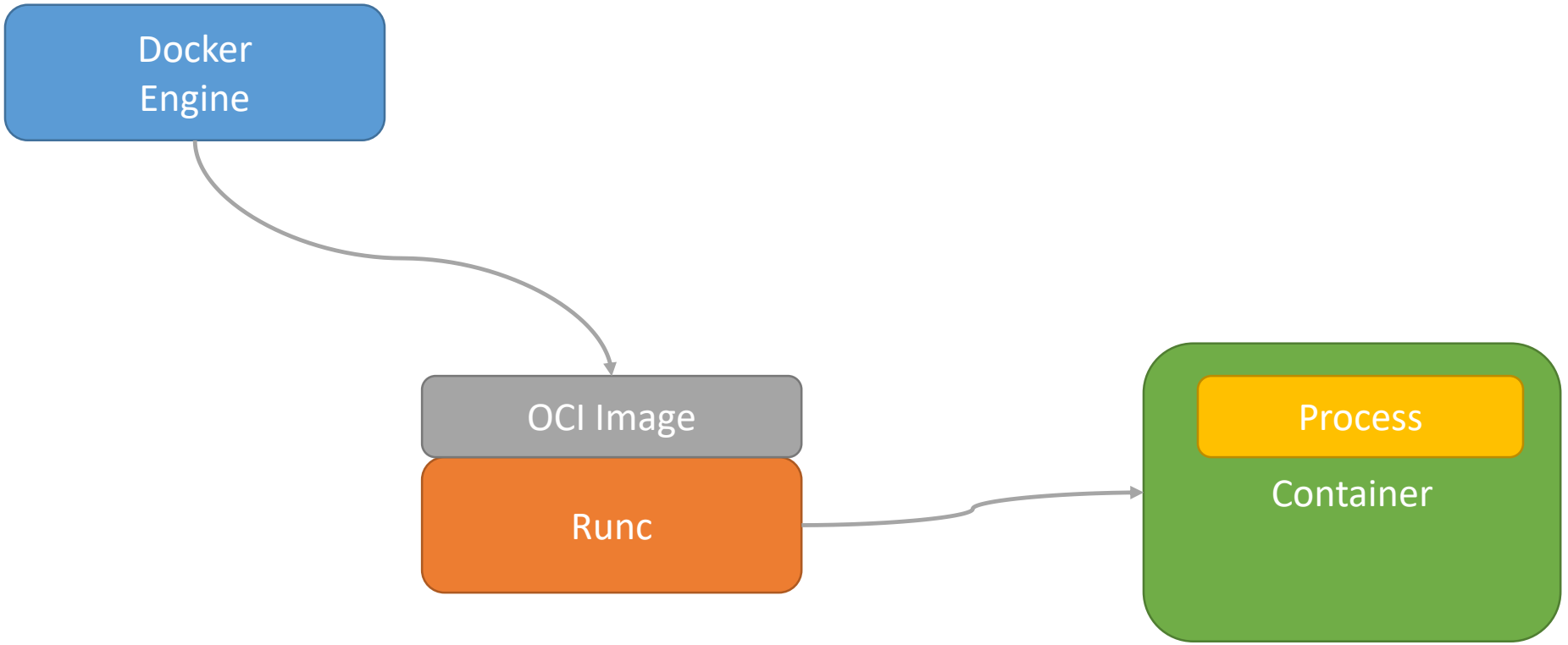
State-of-the-art Container Solutions

What is needed to run a container?

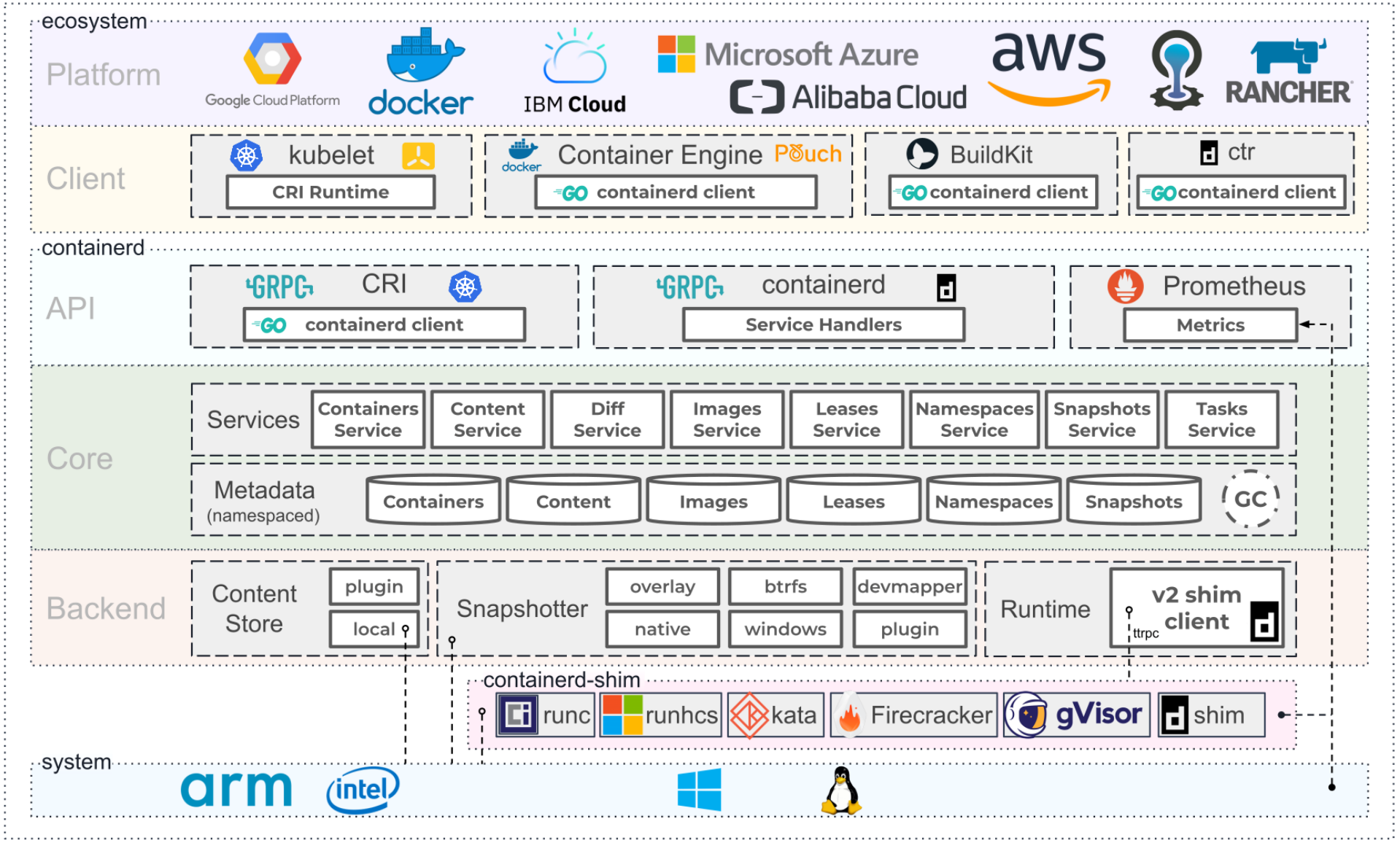
- Container image:
  - a minimal OS (file organizations) + the application (binaries) and its dependencies
- Container engine:
  - takes container images, spawns a container based on it
- Containers have standards:
  - The Open Container Initiative (OCI) is a lightweight, open governance structure (project), formed under the auspices of the Linux Foundation, for the express purpose of creating open industry standards around container formats and runtime.
- OCI Image Specification
  - Covers how a container image is structured
  - All container image building tools are producing OCI images (e.g., Dockers)
  - With the OCI images we can use the same container image with different container engines

What is needed to run a container?

- A simplified flow



Container Ecosystem



State-of-the-art Container Solutions

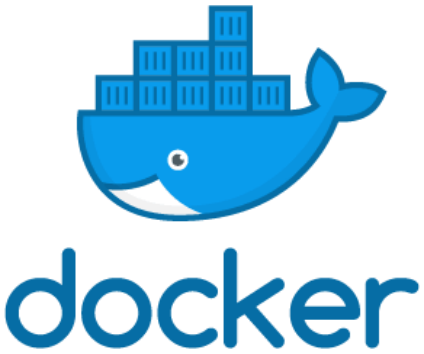
- **Docker**: **https://www.docker.com/**
- **gVisor** (google): https://github.com/google/gvisor
- **Kata** container: https://katacontainers.io/supporters/

- Docker and gvisor container are built upon Namespaces and Cgroups (Linux kernel features), while Kata runs containers within VMs

Docker

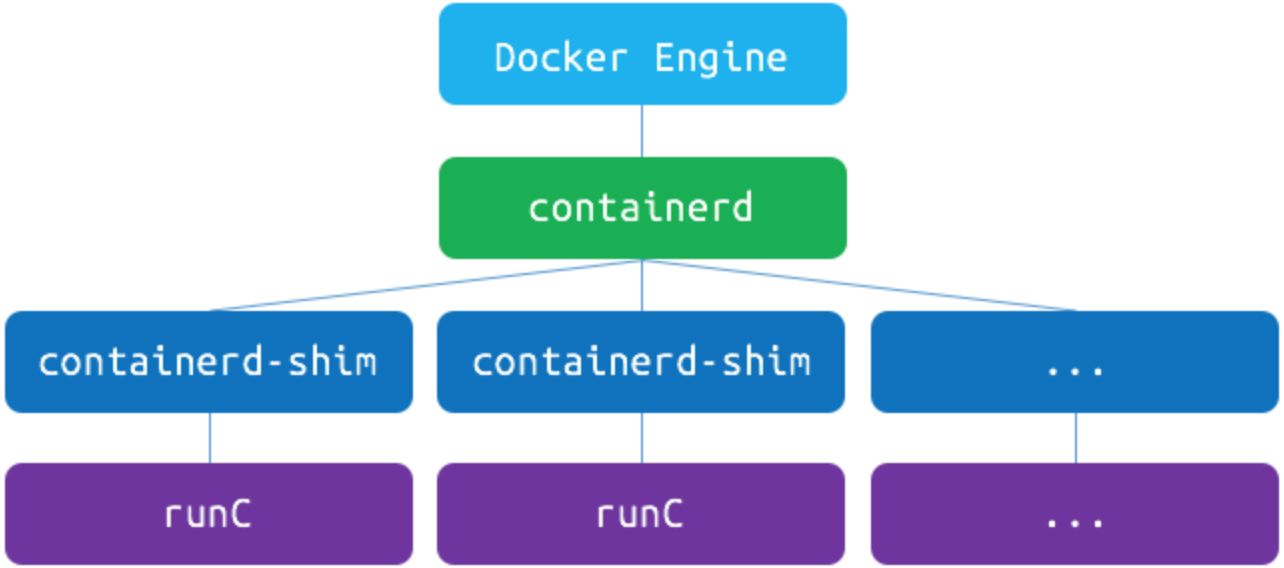
- Broadly speaking, Docker is a platform for developing, shipping and running applications using container technology
- It focuses on Docker containers and growing the Docker ecosystem:
  - Founded in 2009.
  - Formerly dotCloud Inc.
  - Released Docker in 2013.
- It consists of a bunch of products/tools
  - Docker Engine
  - Docker Hub
  - Docker Trusted Registry
  - Docker Machine
  - Docker Compose
  - Docker for Windows/Mac
  - Docker Datacenter

  http://wiki.zenoss.org/download/core/drich_slides/DockerSlides.pdf



Architecture of Docker

- Docker Engine receives requests from upstream clients
- **containerd** manages the complete container lifecycle
- runC is a lightweight tool that does one thing, it creates a container: https://github.com/opencontainers/runc



runC

- https://github.com/opencontainers/runc
- It takes two inputs to start a container: a JSON configuration file and a (OCI) root filesystem bundle.

Container Images

A container is launched by running an image.

An **image** is a root file systemthat includes everything needed to run an application(s)

-- the application code, a runtime, libraries, environment variables, and configuration files.

-- consisting of folders and files just like a file system

When we launch a container, a **container** is a runtime instance of an image—

-- You can see a list of your running containers with the command,

Layout of a Docker Container Image

- A container originates from a base image layer, including a base file system (and applications).
- When you launch a container, another layer is created on top of the base image layer

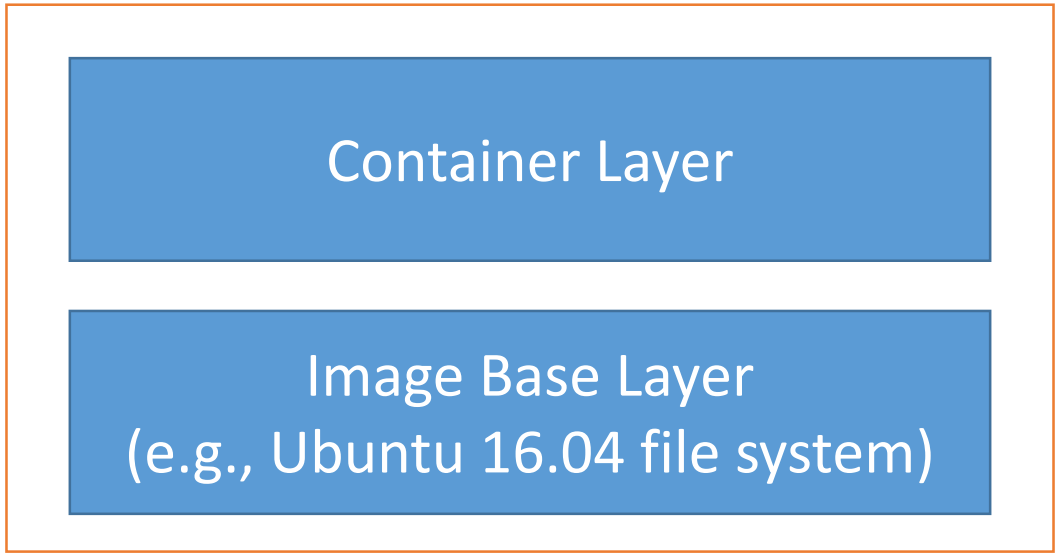
Container Layer

Image Base Layer
(e.g., Ubuntu 16.04 file system)

Read/Write Permissions

- The image base is read only
- The container layer can both read and write

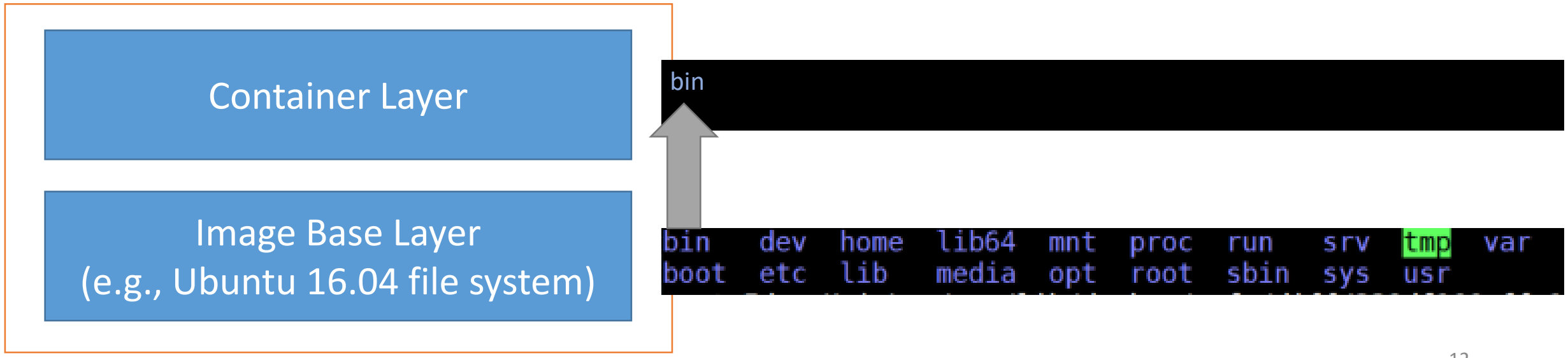
A merged view through overlay file system such as AUFS



https://docs.docker.com/storage/storagedriver/aufs-driver/#how-the-aufs-storage-driver-works

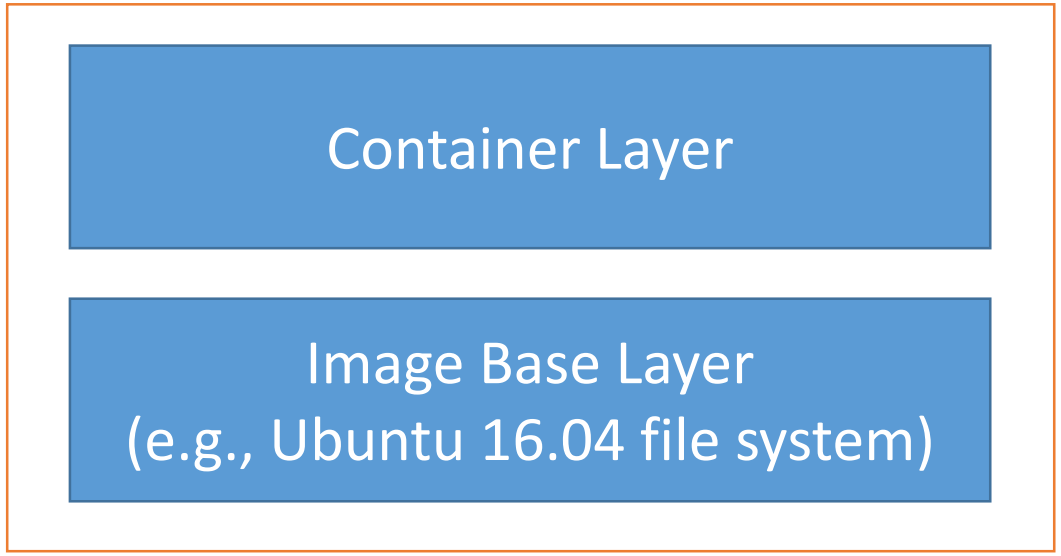
Write Polices

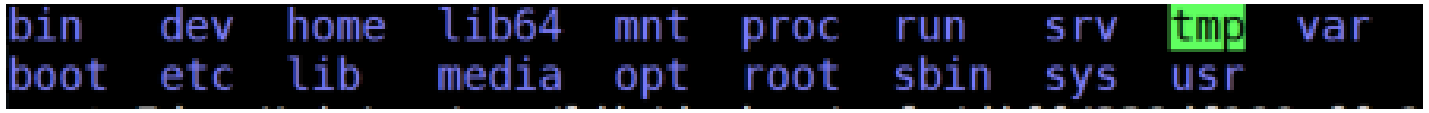
- Writing files:
  - Writing to a file for the first time (the file exists in the image layer)
    - --- copy_up: copy files from the base layer to the container layer, and write changes to it.



Read Policies

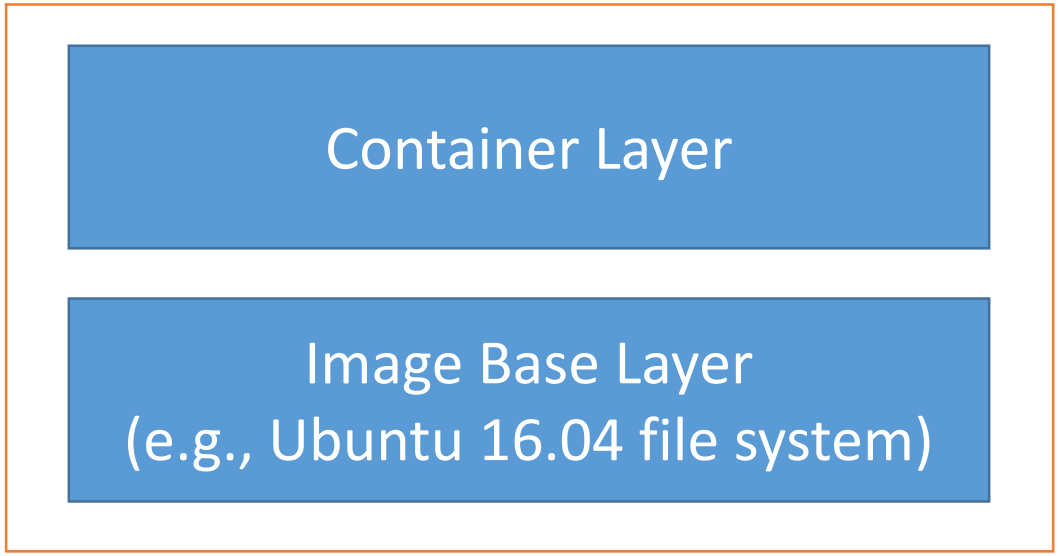
- Reading files:
  - Files only exist in image layer, it is read from image layer
  - Files only exist in container layer, it is read from container layer
  - Files exist in both layer, it is read from container. Files in the container layer obscure files with the same name in the image layer.





How container reads and writes work with

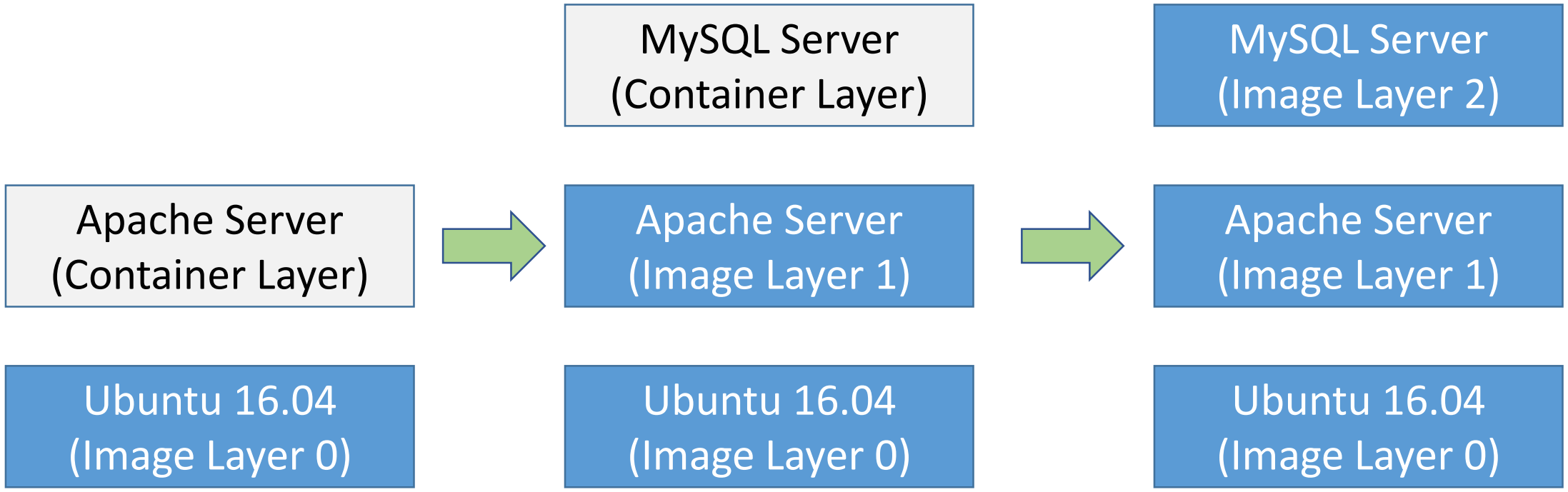
- Writing files:
  - Writing to a file for the first time --- copy_up: copy files from the base layer to the container layer, and write changes to it.
  - Deleting a file – a whiteout file is created in the container layer marking that the file with the same name in the image layer is invalid



Pros/Cons of Overlay file systems

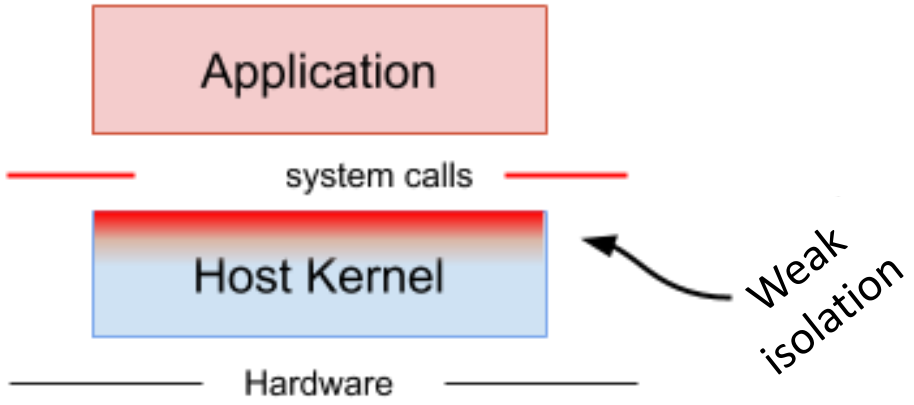
- Cons
  - Overhead
- Pros
  - Many container instances share the same base images
    - Saving space
  - Container image can be stackable
    - Easy to build new images

Stackable Container Images



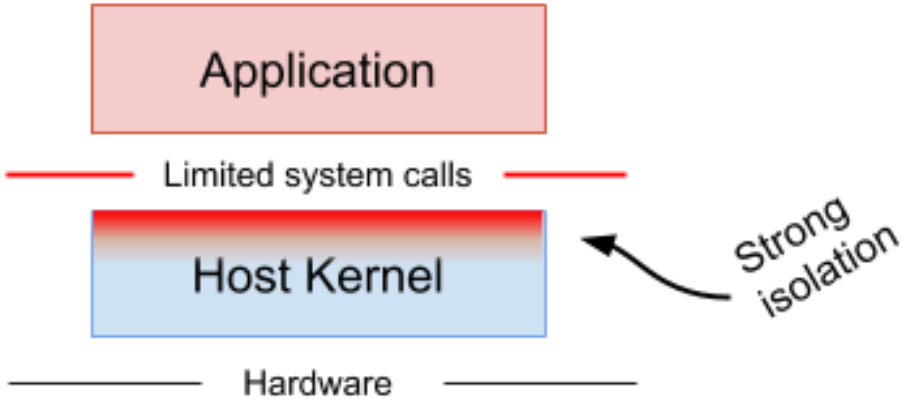
Security Concerns of Containers

- If any one of the container breaks out, it can allow unauthorized access across containers, hosts or data centers etc., thus affecting all the containers hosted on the Host OS.
- Attack surface: system calls
- How to mitigate?



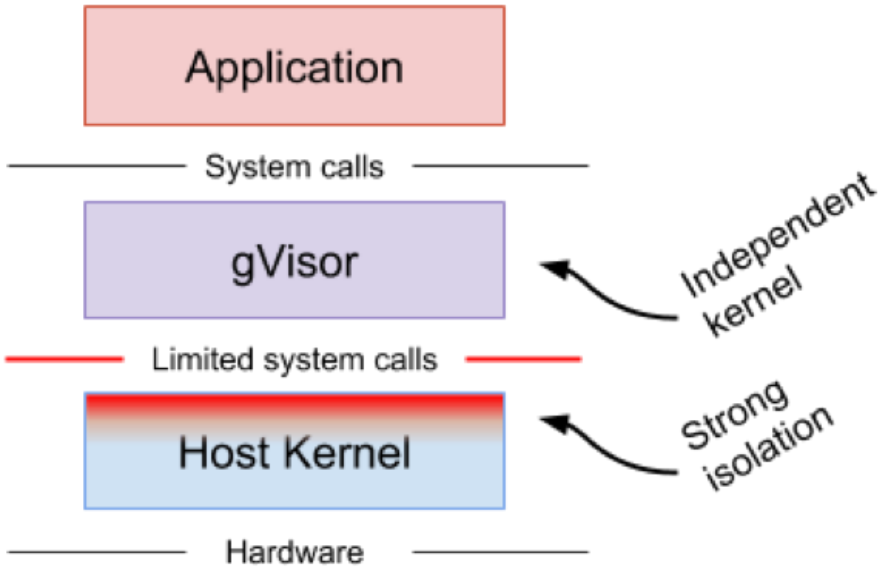
Security Concerns of Containers

- **Rule-based execution** allows the specification of a fine-grained security policy for an application or container. (e.g., Linux's **seccomp**)
- In practice, not easy but unknow applications.



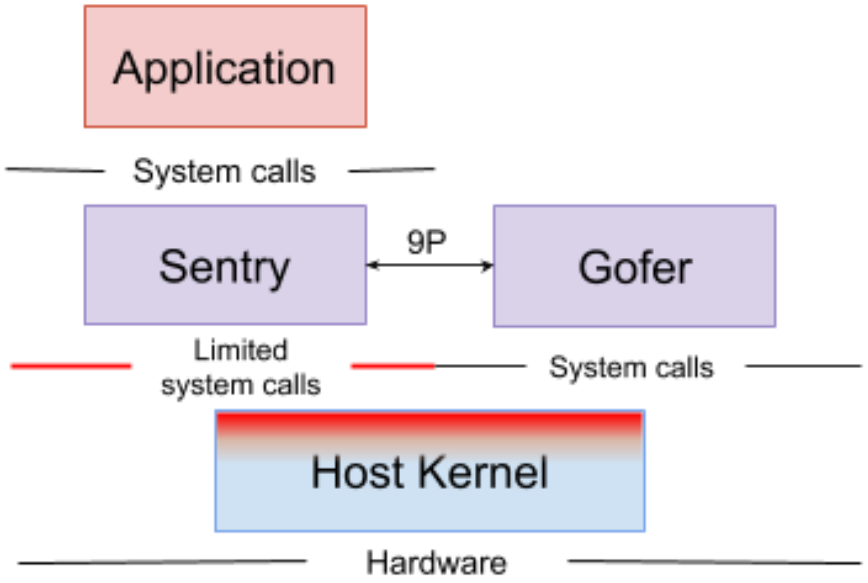
gVisor -- Google's Secure Containerization

- gVisor intercepts application system calls and acts as a guest kernel
- It implements a substantial portion of the Linux system surface
- The isolation boundary between the application and the host kernel is maintained
- Drawbacks?
  - Reduced application compatibility
  - High per-system call overhead



gVisor – Architecture

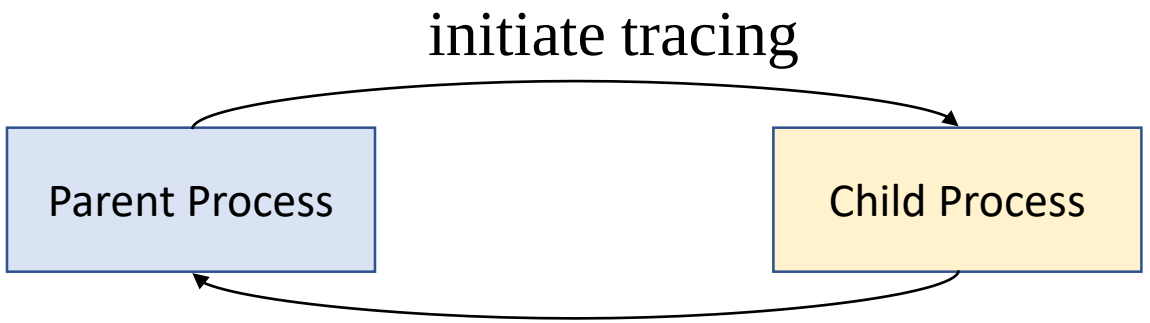
- **Sentry:** the largest component of gVisor
  - Can be thought of as a userspace OS kernel, implementing all the kernel functionality needed by the untrusted application
  - System calls are redirected to Sentry
  - Sentry will make some host system calls to support its operation, but it will not allow the application to directly control the system calls it makes.



gVisor – Sentry

- gVisor requires a **way** to implement interception of syscalls
- **The ptrace() system call** provides a mechanism by which a parent process may observe and control the execution of another process.

long ptrace(enum __ptrace_request *request*, pid_t *pid*, void * *addr*, void * *data*);



- PTRACE_SYSEMU request
  - causing the traced process to stop on entry to the next syscall

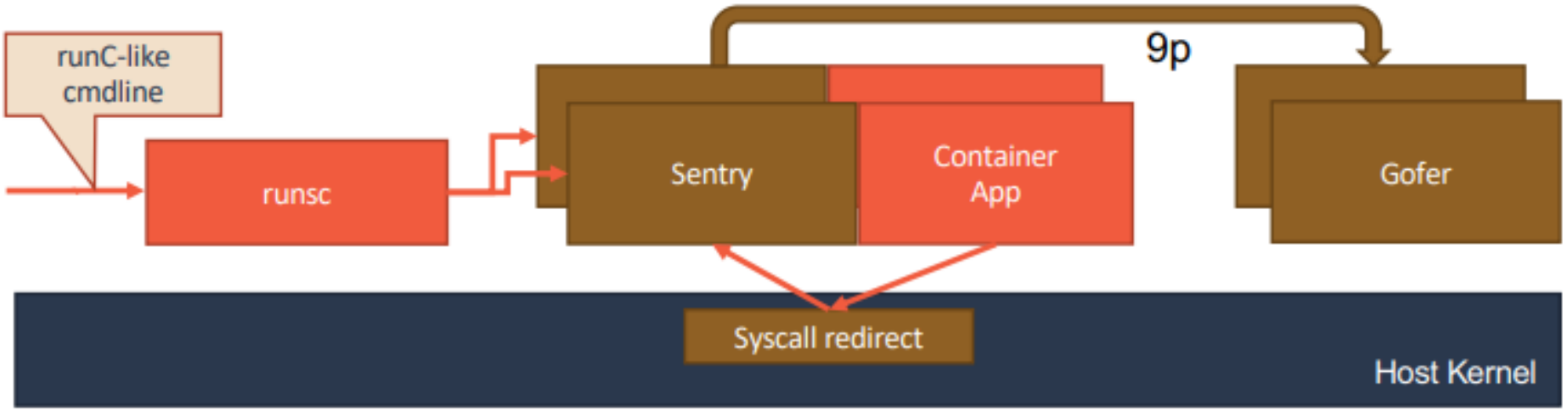
gVisor – Sentry

- Studying this could be a possible “large project” for this course
- Interested?

```
for (;;) {
    ptrace(PTRACE_SYSEMU, pid, 0, 0);
    waitpid(pid, 0, 0);

    struct user_regs_struct regs;
    ptrace(PTRACE_GETREGS, pid, 0, &regs);

    switch (regs.orig_rax) {
        case OS_read:
            /* ... */

        case OS_write:
            /* ... */

        case OS_open:
            /* ... */

        case OS_exit:
            /* ... */

        /* ... and so on ... */
    }
}
```

gVisor – runsc

- Runsc – the entrypoint to running a sandboxed container
  - Implements an OCI runtime specification including:
    - A **config.json** file contains container configurations
    - A **root filesystem**



Performance Comparisons

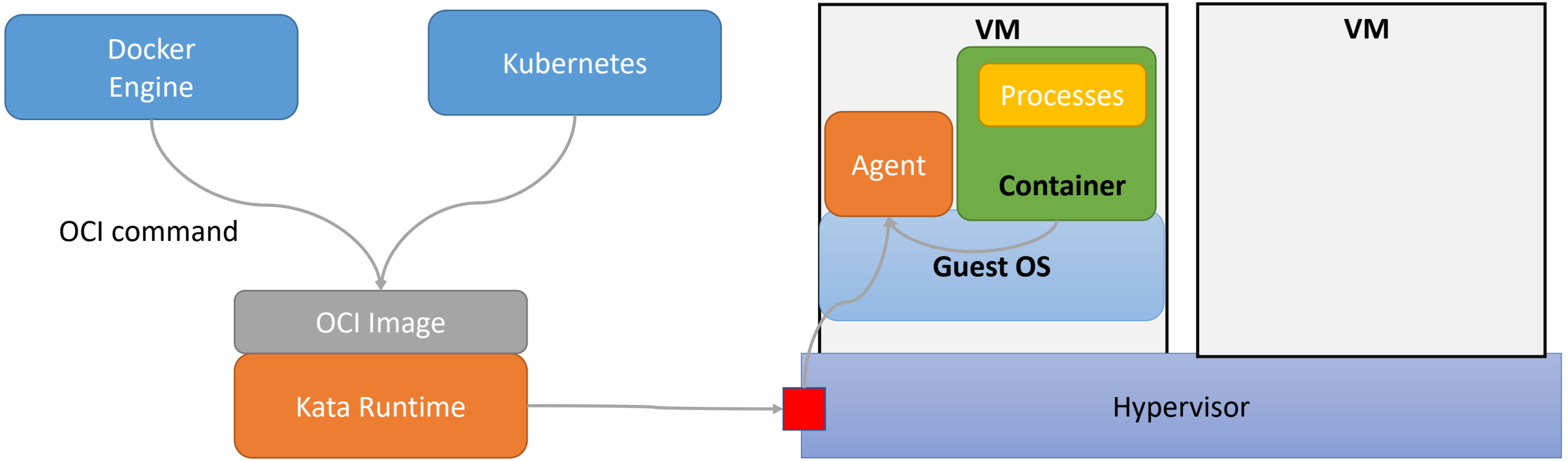
- Start docker container or gVisor container:
  - docker run -it --runtime=runc severalnines/sysbench /bin/bash
  - docker run -it --runtime=runsc severalnines/sysbench /bin/bash
- Within the container, run the tests:
  - File IOs
    - sysbench --test=fileio --file-total-size=1G prepare
    - sysbench --test=fileio --file-total-size=1G --file-test-mode=rndrw --max-time=60 --max-requests=0 run
  - CPU
    - sysbench --test=cpu --cpu-max-prime=20000 run
  - Memory
    - sysbench --test=memory --memory-block-size=1M --memory-total-size=10G run

Kata Container

- Basic Ideas:
  - It’s possible to run containers inside of virtual machines
  - Pretty common deployment method (think about the cloud)
  - Introduces another layer of protection: the hypervisor
  - But notice that Hypervisors can have security bugs as well
- Kata Container
  - Born in 2017 from the merge of Intel’s Clear Containers and Hyper’s runV
  - “Wraps” containers into dedicated virtual machines
  - OCI runtime implementation: can be plugged into the container engine (e.g., Docker)
  - Can consume the same container images already existing:

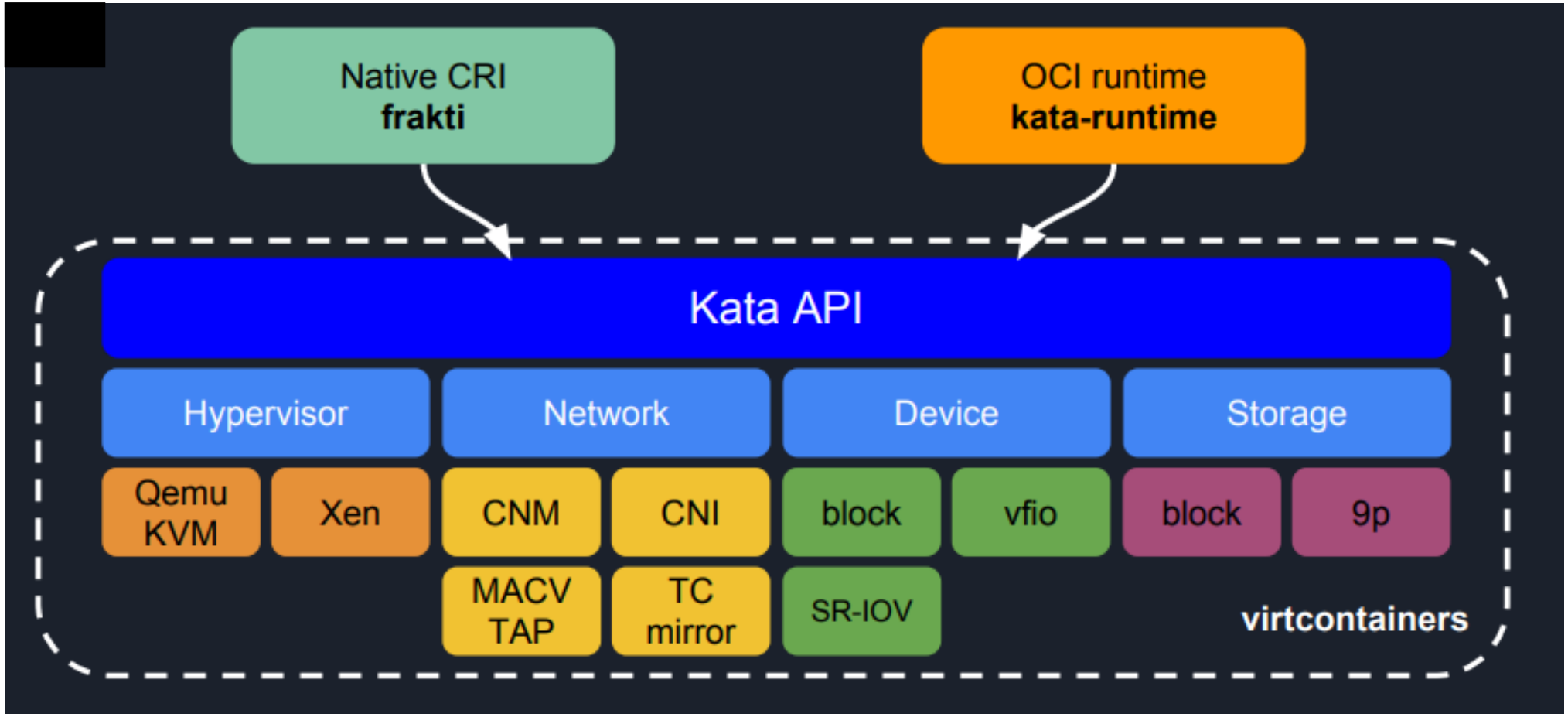
Kata Container

- Architecture



Kata Container

- More than OCI



Sources

1. How to Implement Secure Containers Using Google's gVisor https://thenewstack.io/how-to-implement-secure-containers-using-googles-gvisor/

2. Intercepting and Emulating Linux System Calls with Ptrace: https://nullprogram.com/blog/2018/06/23/

3. The True Cost of Containing: A gVisor Case Study: https://www.usenix.org/system/files/hotcloud19-paper-young.pdf

4. kata containers & gVisor: https://object-storage-ca-ymq-1.vexxhost.net/swift/v1/6e4619c416ff4bd19e1c087f27a43eea/www-assets-prod/presentation-media/kata-containers-and-gvisor-a-quantitave-comparison.pdf

5. Kata Containers The way to run virtualized containers: https://object-storage-ca-ymq-1.vexxhost.net/swift/v1/6e4619c416ff4bd19e1c087f27a43eea/www-assets-prod/presentation-media/Kata-Containers-The-way-to-run-virtualized-containers.pdf

6. Bringing container security to the next level using Kata Containers: https://www.suse.com/media/presentation/TUT1201_Bringing_Container_Security_to_the_Next_Level_Using_Kata_Containers.pdf