Infrastructure Security Using Linux

Computer science / Cybersecurity

Virtualization and Containers

24 → 25

Virtualization

Server Virtualization

- Runs multiple OS instances on a single physical server
- Virtualization software allocates CPU, memory, and I/O across “guest” OSes
- To users, a virtual server behaves like a physical one
- **Why It Matters**
    - Boosts resource efficiency
    - Enables isolation and scalability
    - Foundation for **cloud computing** and **on-demand provisioning**
- **Evolution**
    - VMware pioneered x86 virtualization
    - Led to cloud infrastructure as we know it
    - **OS-level virtualization** (e.g., containers) now drives lightweight abstraction

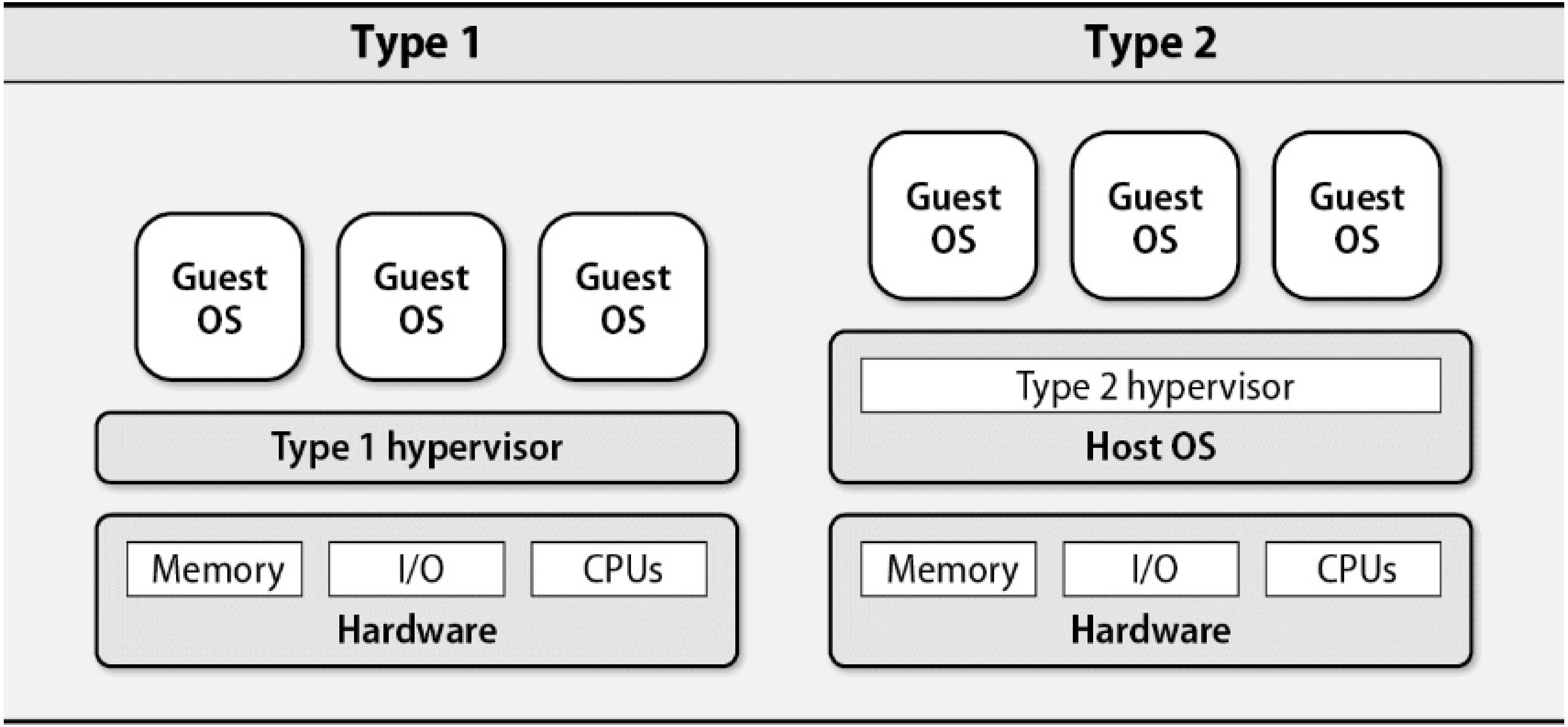
Hypervisor

- Software layer between virtual machines and hardware
- Shares resources among isolated guest OSes
- Examples: VMware ESXi, XenServer, ProxMox, KVM (Linux)
- **Full Virtualization**
  - Emulates all hardware components
  - Slower due to instruction translation
  - Often uses QEMU for emulation

Paravirtualization

- **Paravirtualization**
  - Guest OS communicates directly with hypervisor
  - Faster, but requires guest OS modifications
- **Hardware-Assisted Virtualization**
  - Uses CPU support (Intel VT, AMD-V) for faster virtualization
  - Virtualizes CPU and memory under hypervisor control
- **Paravirtualized Drivers**
  - Improve performance of hardware-assisted virtualization
  - Used for disk, network, and display I/O
  - Avoid major OS changes while bypassing full emulation

type 1(0) vs type 2 hypervisors



Live Migration & VM Images

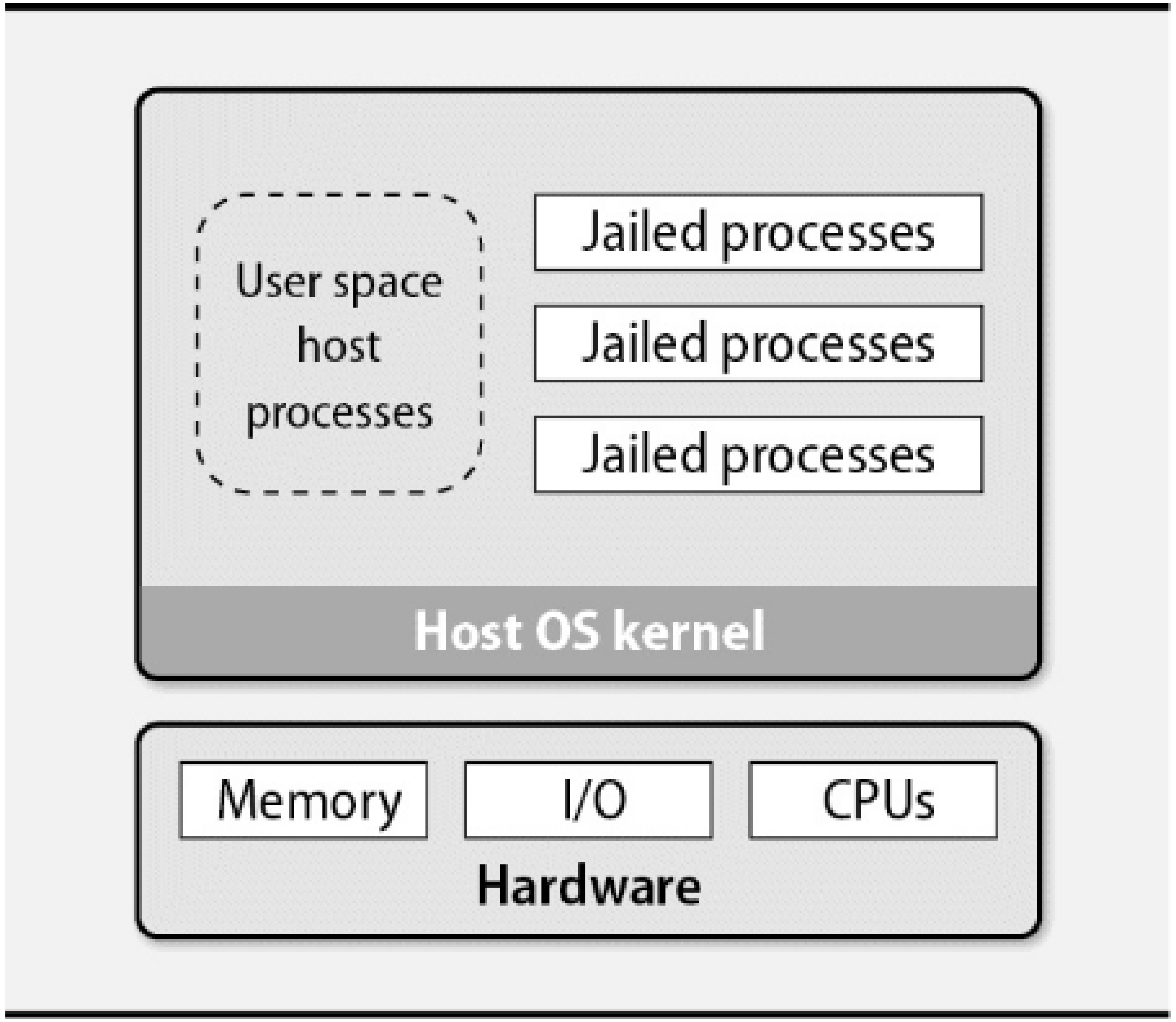
- Moves a running VM between hypervisors **without downtime**
- Enables **load balancing**, **maintenance**, and **disaster recovery**
- Achieved by syncing VM memory state between source and target
- **VMware ESXi** implements this as **vMotion**

- **Virtual Machine Images**
  - Preconfigured OS templates used to create virtual servers
  - Loaded by hypervisors at VM creation
  - Image format depends on the hypervisor used (e.g., VMDK, QCOW2, VDI)

Containerization

- OS-level virtualization, also known as containerization, is a different approach that does not use a hypervisor. Instead, it relies on kernel features that isolate processes from the rest of the system.

Exhibit B: Containerization

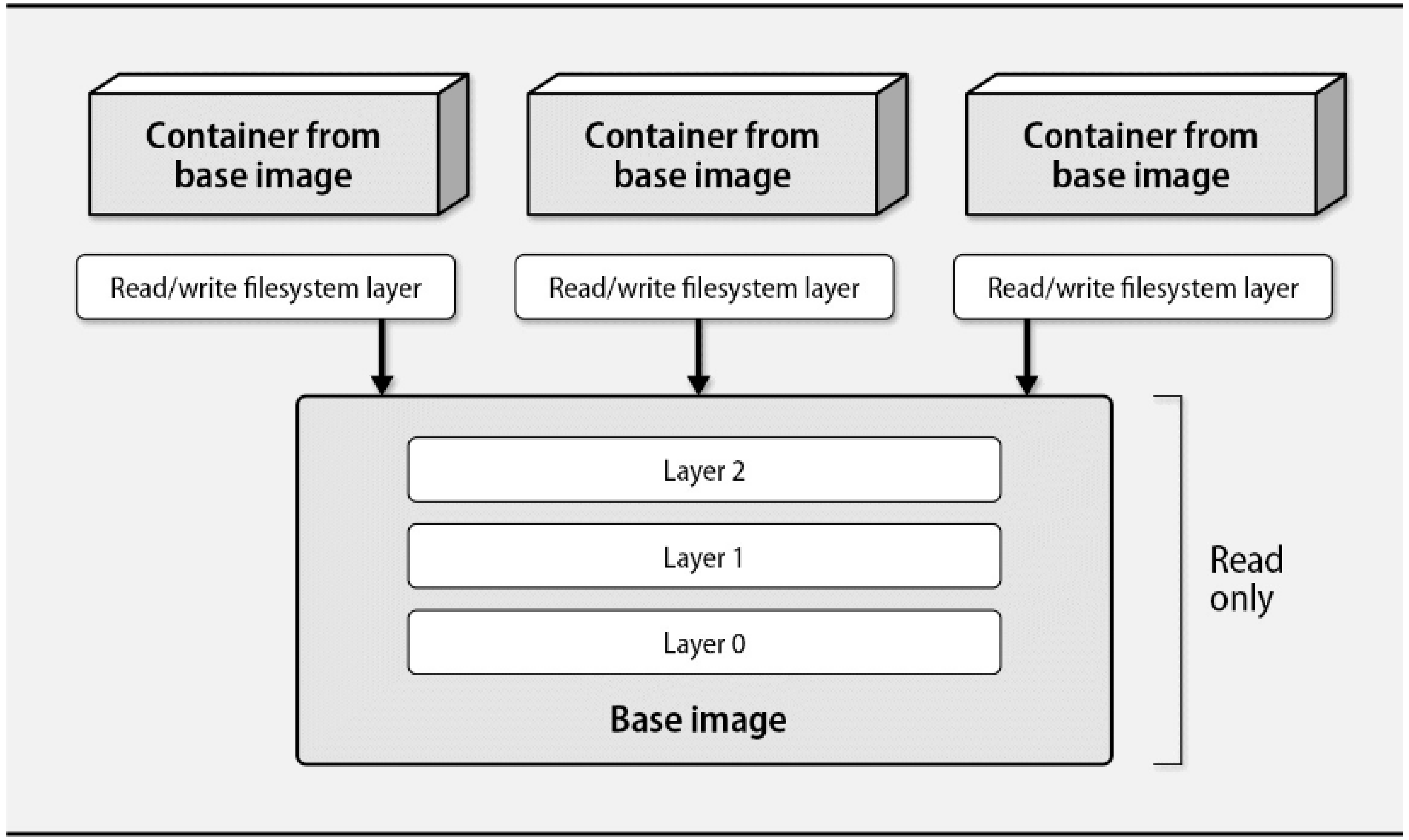


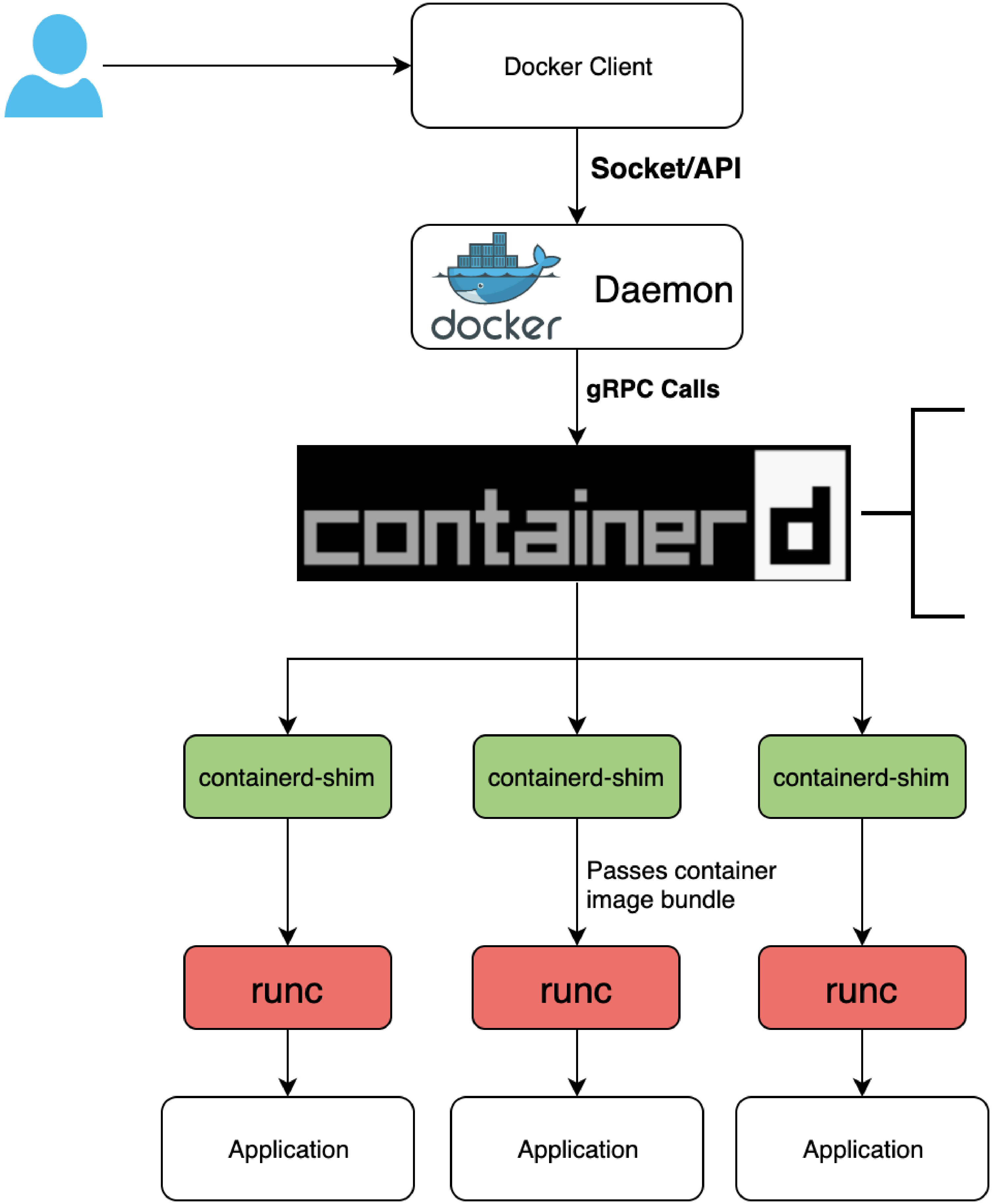
Containers – Background & Core Concepts

- Containers combine kernel features, filesystem isolation, and networking to create lightweight, portable environments.
- A **container** is a group of isolated processes with access to a private root filesystem and namespace. Containers share the host OS kernel but are restricted from accessing system resources outside their scope.
- **Key Points**
  - Managed by a **container engine** (e.g., Docker, containerd)
  - No OS-level emulation—low overhead
  - Apps run unmodified and are unaware of being containerized

Docker – The Open-Source Container Engine

- **Open Standards**
  - Docker joined the **Open Container Initiative (OCI)** to promote interoperability and avoid vendor lock-in
  - Founded the **Moby project** to modularize container engine components


Container from base image
Container from base image
Container from base image
Read/write filesystem layer
Read/write filesystem layer
Read/write filesystem layer
Layer 2
Layer 1
Layer 0
Base image
Read only



Docker Client

Socket/API

Daemon

gRPC Calls

containerd

containerd-shim

containerd-shim

containerd-shim

Passes container image bundle

runc

runc

runc

Application

Application

Application

- High level container runtime
- Responsible for container lifecycle management
- Responsible for container image management
- unpacks the image into an OCI runtime bundle and shells out to `runc` to run it

- OCI container runtime spec reference implementation
- Container runtime code
- Create containers

Basic commands

- **docker run**: Create a new container from an image and start it
- **docker ps**: List running containers
- **docker build**: Create a new image from a Dockerfile
- **docker images**: List images
- **docker exec**: Run a command in a running container
- **docker stop**: Stop a running container
- **docker rm**: Remove a container
- **docker rmi**: Remove an image

- **Example**: run a nginx container

```
docker run -d -p 80:80 --name my-nginx --hostname my-nginx nginx
```

Container Security

- Container isolation depends on kernel features that prevent access to files, processes, and resources outside the container. These features are mature and stable, with origins going back to 2008.
- **Key Risk: Misconfiguration**
  Security breaches are usually caused by insecure setups, not kernel flaws.
- **Critical Practice: Protect the Docker Daemon**
  - **dockerd** runs with elevated privileges
  - Anyone with access to the daemon can escalate to root on the host
  - Restrict access via socket permissions, firewall rules, and proper user roles