Cloud Computing Technologies

DAT515 - Fall 2024

Virtualization and Containers

Prof. Hein Meling

University
of Stavanger

Cloud Infrastructure

Cloud Infrastructure

- Collection of components required to provide cloud computing
- Hardware Resources
  - Processing Units (CPU, TPU, GPU)
  - Storage Devices (RAM, SSD, HDD)
  - Network Devices (NIC, Switches, Routers)

Cloud Infrastructure

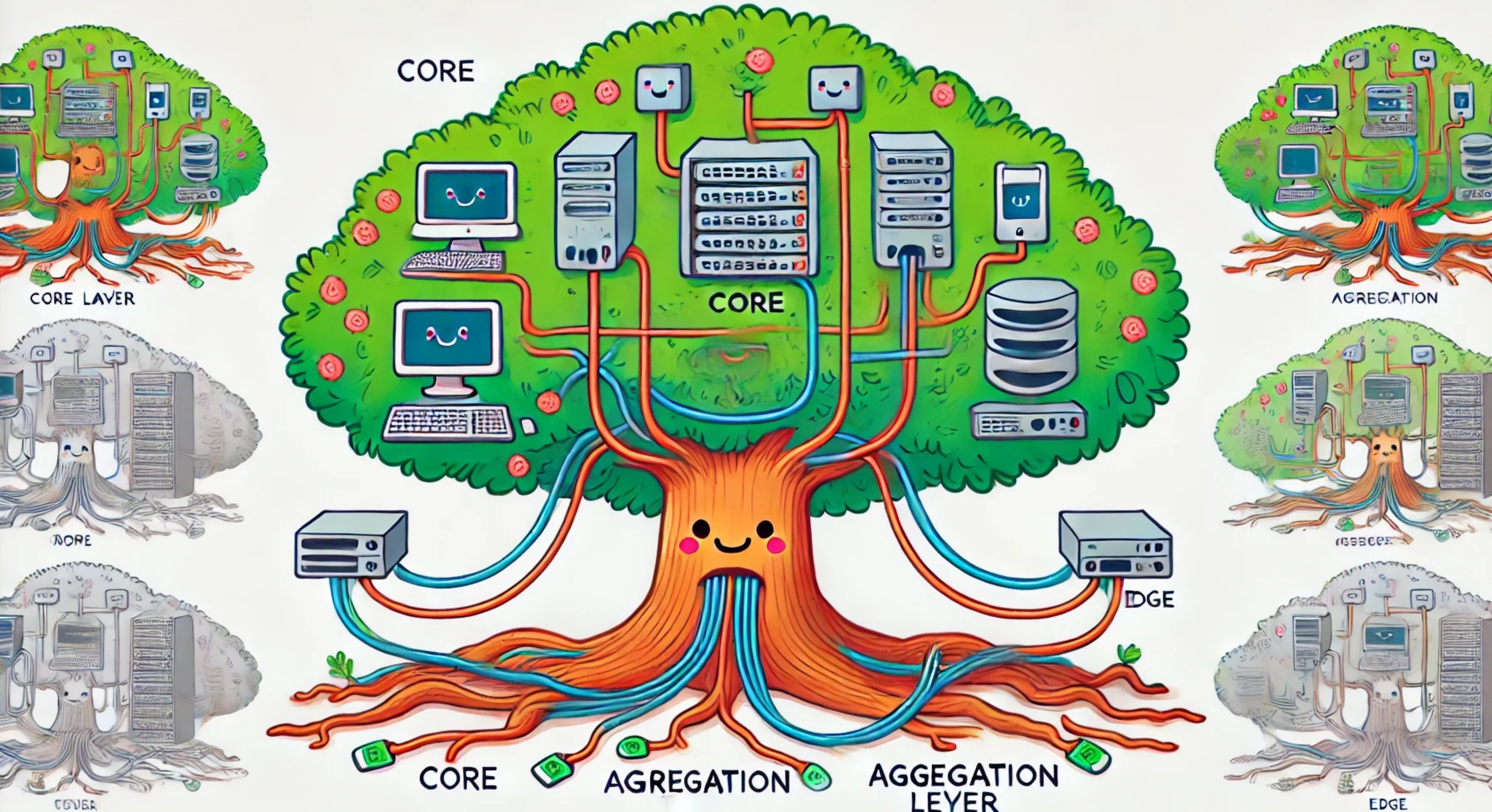
Networking Devices

- Connects processing units
  - Within server rack
  - Within data center
  - Across data centers
- Network topologies for efficient distribution of data
  - Spine-leaf topology
  - Fat-tree topology

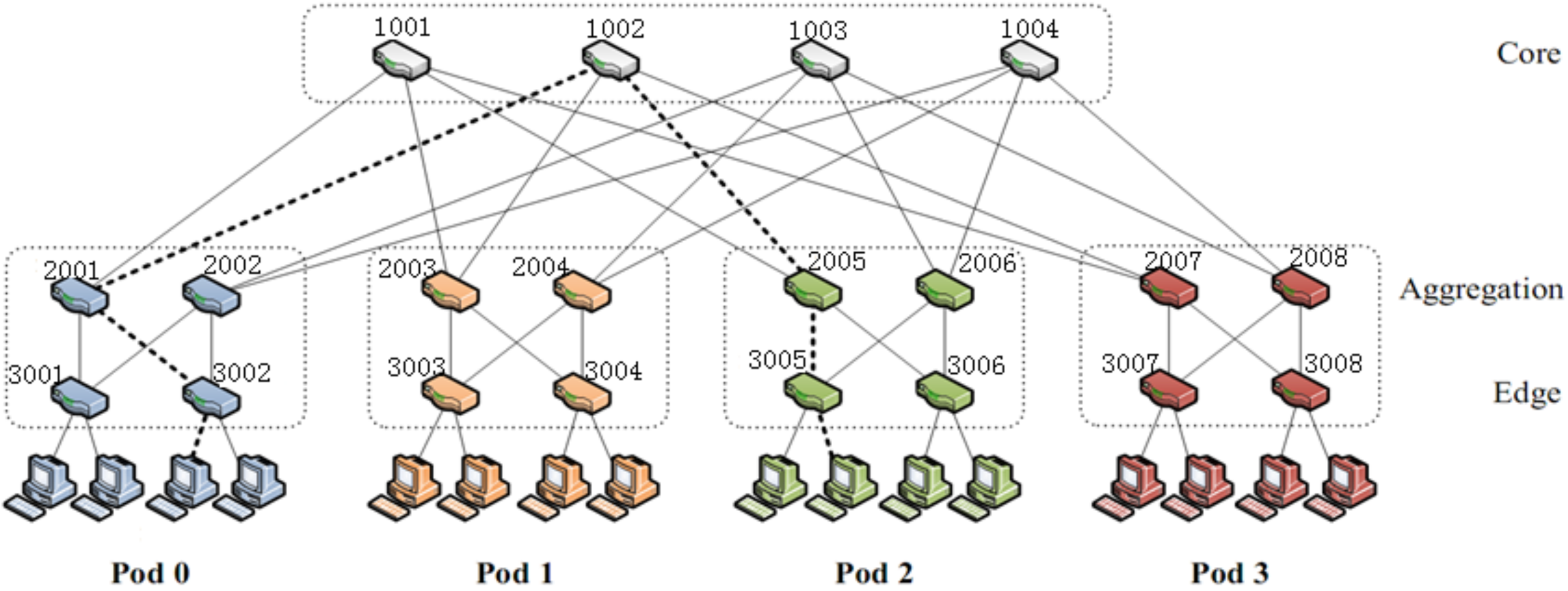
Low latency

High bandwidth

How can we connect tens of thousands of machines in a data center together?


CORE
CORE
CORE LAYER
AGREGATION
EDGE
CORE
AGREGATION
AGGEGATION
LEYER

Small Fat Tree Example



Cloud Infrastructure

Storage Devices Exposed to Cloud Users in Different Ways

- Storage Types
  - Block, Object, and File Storage
- Replication
  - Within single data center
  - Across data centers
  - Across availability zones
- Elastic Storage
  - Adapt storage resources to fluctuating demands
  - Scalability

Virtualization

Recap from Operating Systems

- Virtualizing the CPU
  - The Process Abstraction
- Virtualizing Memory
  - The Address Space Abstraction

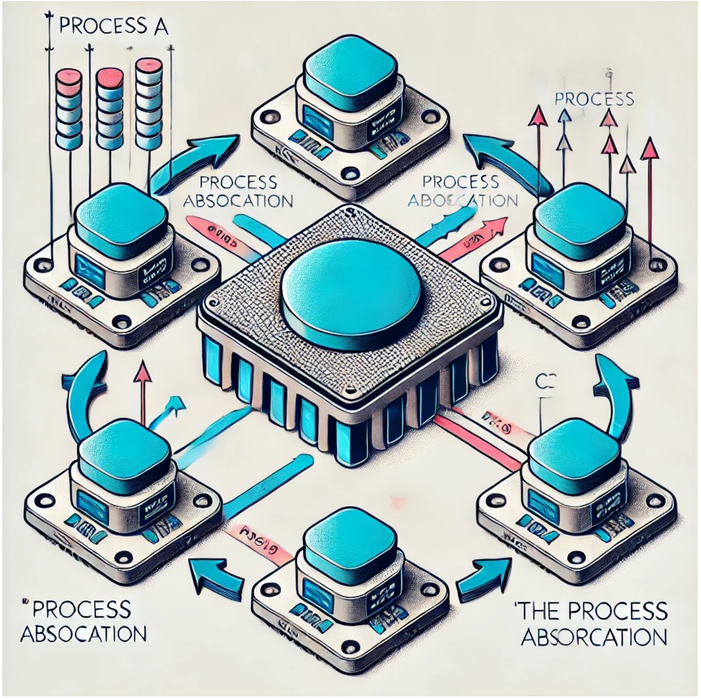


Image is generated and does not represent any useful technical information.

Why Virtualize?

Resource Sharing

Improve utilization of (expensive) compute resources

But Resource Sharing Needs Protection Mechanisms

- **Isolation** between “entities” using a virtual “unit”
  - E.g., two processes running on the same CPU should not interfere with each other

Simplification

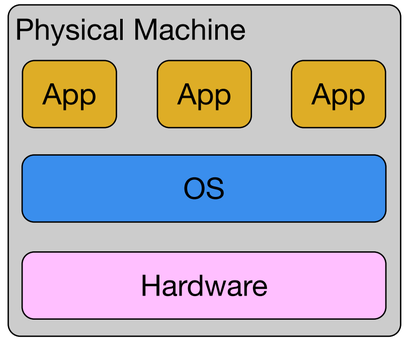
Simplification

- **Encapsulation** makes it easy to manage, move, and replicate a virtualized unit
  - Enables scalability and flexibility

The Virtual Machine

The Physical Machine

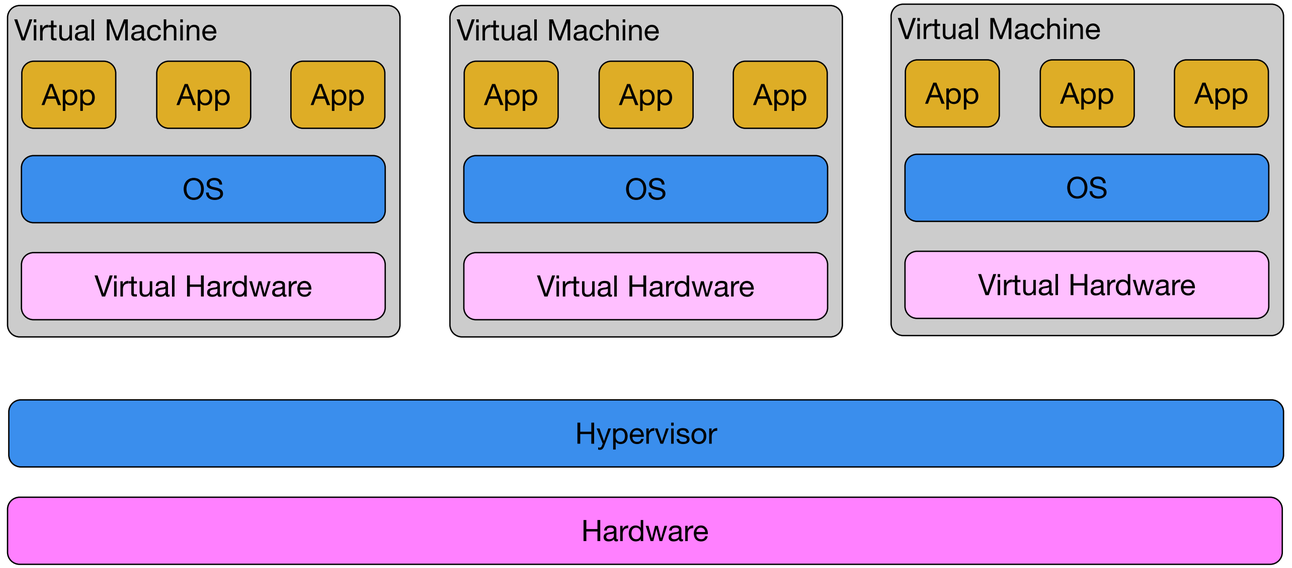
- Physical Machine
  - Processors
  - Memory
  - Storage
  - Network



The Virtual Machine

The Unit of Abstraction

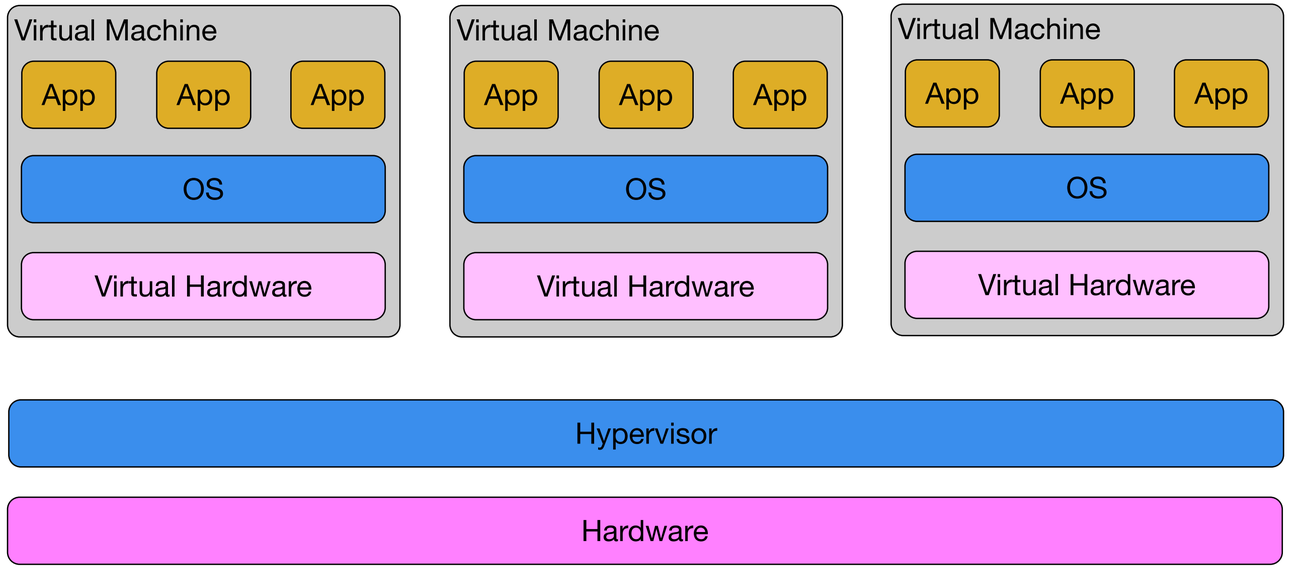
- Virtual Machine — Virtualizing all parts of computer
  - Processors
  - Memory
  - Storage
  - Network



Hypervisor ~ the supervisor above multiple operating systems

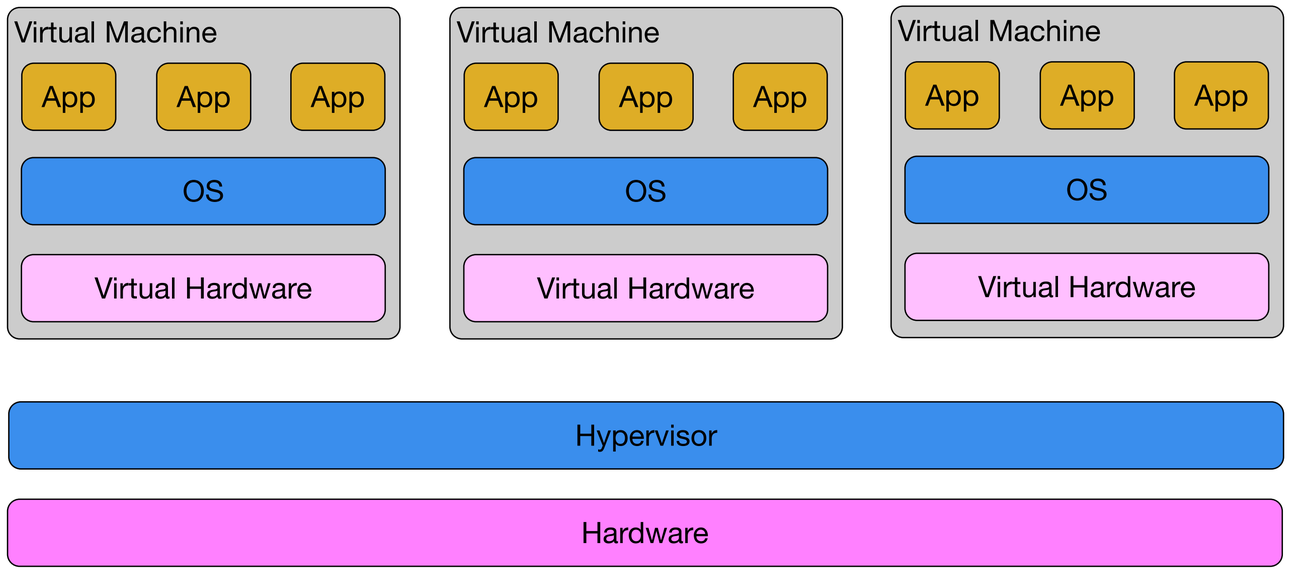
VMs Cannot Interact with Hardware Directly

- Software layer for managing resources and virtual machines
  - Creation, Monitoring, Deletion,
  - Execution of VMs
- Virtual Machine Monitor
  =
  Hypervisor



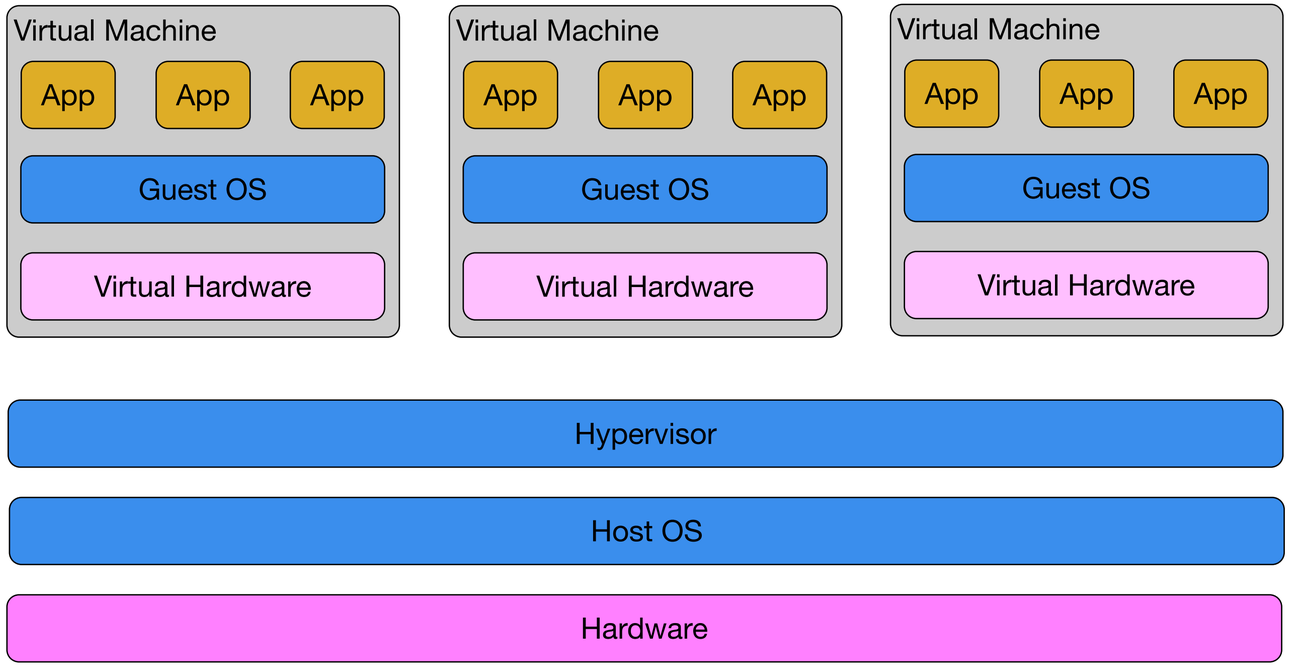
Type 1 Hypervisor (Bare Metal)

- Runs directly on the host's hardware
- Without an underlying OS
- Examples
  - KVM and XEN
  - VMware ESXi and Microsoft Hyper-V



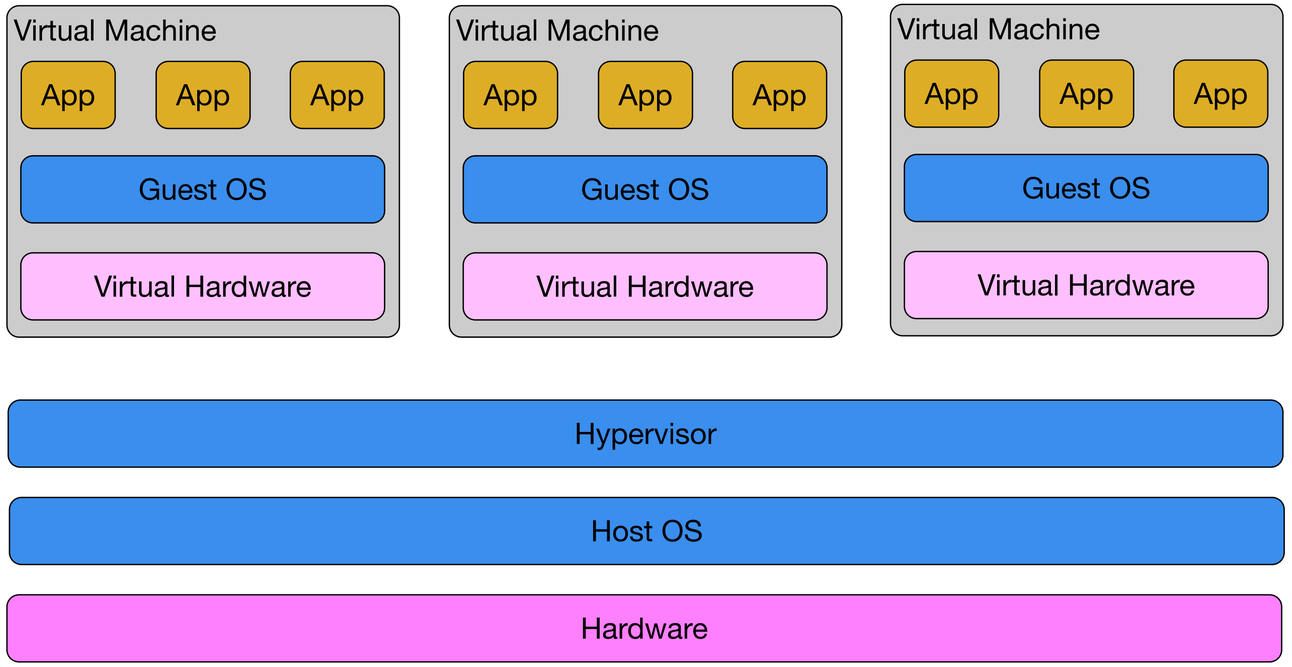
Type 2 Hypervisor (Hosted)

- Runs on top of a host OS
- Relies on the host OS for resource management
- Examples
  - VMware Workstation
  - Oracle VirtualBox



Type 2 Hypervisor (Hosted)

- Desktop app running on host OS
  - Create VMs manually
  - Install guest OS
  - Configure resources via the desktop app
- Typical use case:
  - Run other OS on laptop



Bare Metal vs Virtual Machines

	Bare Metal	Virtual Machines
Runs	Directly on hardware	In virtualized environment
Performance	High (direct access to hardware)	Slightly lower (due to virtualization overhead)
Resource Utilization	Dedicated* (full use of hardware resources)	Shared resources
Isolation	Strong* (physical separation of workloads)	Good* (but relies on hypervisor for isolation)
Flexibility	Low (tied to specific hardware)	High (can run multiple OSes on one host)
Deployment	Slow (physical setup and configuration)	Fast (VMs can be created quickly)
Scalability	Limited to physical hardware capacity	High (easy to provision additional VMs)
Management	Manual	Software-based (hypervisor)
Use Cases	High-performance applications (long-running batch)	General-purpose workloads (with high variability)

Joke time

Why did the virtual machine break up with the hypervisor?

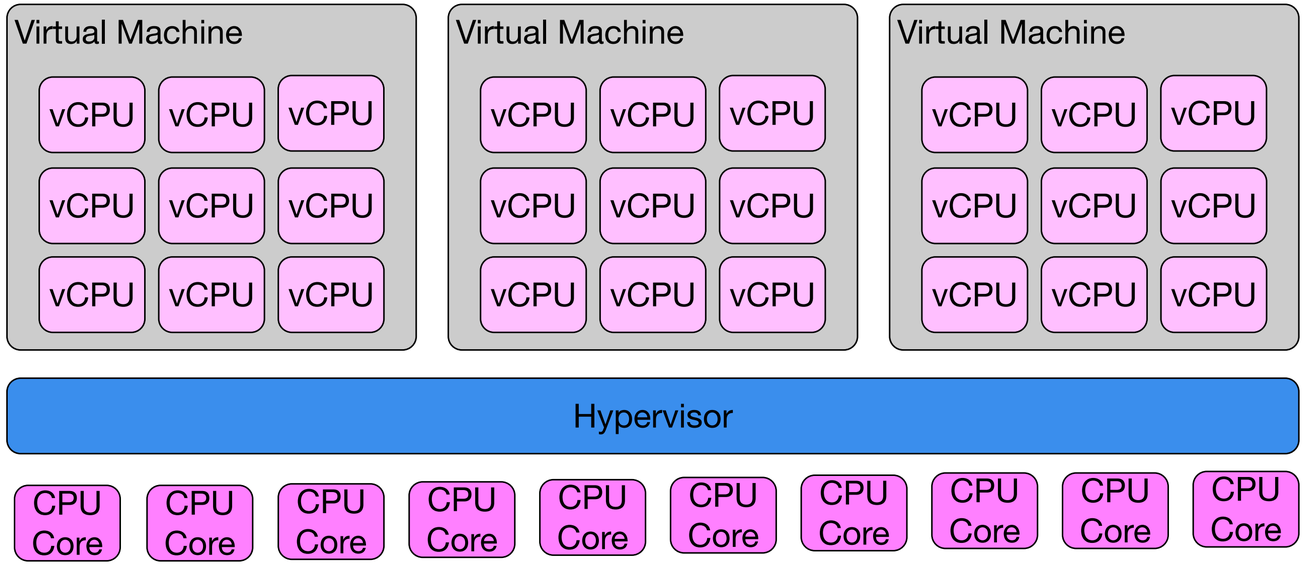
It felt like it was just another layer of abstraction!

Virtualized Processor Core

Virtualized Processor Core

Virtual CPU (vCPU) = a portion of the physical CPU resources

- Hypervisor:
  - Divide the physical CPU cores into smaller units called vCPUs
  - Assigns these to virtual machines
  - Schedules vCPUs to run on the physical cores as needed
- vCPU ≠ physical CPU core
- A vCPU shares time on a physical core with other vCPUs from different VMs



Virtualized Processor Core

Virtual CPU (vCPU) = a portion of the physical CPU resources

- The hypervisor manages vCPU allocation to optimize performance
- **Performance:** # of vCPUs assigned to a VM impacts its performance
  - Actual performance also depends on the overall load on the host machine
- **Scalability:** vCPUs allow for flexible and scalable resource allocation
  - Administrators can allocate more/fewer vCPUs to a VM depending on its workload

Virtualizing GPUs

Direct Pass-Through

GPU is dedicated to a specific VM or container

- Has direct, exclusive access to the GPU hardware
- **Pros:**
  - Maximum performance, as the workload, has full access to the GPU
- **Cons:**
  - Not resource-efficient for smaller workloads since the GPU cannot be shared

NVIDIA Multi-Instance GPU (MIG)

GPU partitioned into multiple smaller instances (hardware level)

- Each instance acts as an independent GPU
  - With dedicated memory and compute cores
- Can be assigned to a different container or VM
- **Pro:**
  - Efficient resource utilization (share a single GPU)
  - Strong isolation between instances prevents instances from affecting each other
- **Cons:**
  - Partitioned instances have limited resources compared to the full GPU

Virtual GPU (vGPU)

GPU partitioned into multiple virtual GPUs (software level)

- Each instance acts as an independent GPU
  - With dedicated memory and compute cores
- Can be assigned to a different container or VM
- **Pro:**
  - Efficient resource utilization (share a single GPU)
- **Cons:**
  - Partitioned instances have limited resources compared to the full GPU
  - Performance isolation is not as good as hardware-based partitioning

Job Scheduling and Resource Allocation

Slurm and Kubernetes

- GPU resources are managed by job schedulers
  - Allocate GPUs to different workloads based on demand and priority
  - Can involve time-sharing or exclusive access, depending on the workload requirements
- **Pro:**
  - Efficient resource management and workload prioritization
- **Cons:**
  - Complexity in scheduling and potential delays due to resource contention

Security

Security of Virtual Machines

Advantages

- **Isolation:** Compromised VM does not impact other VMs
- **Sandboxing:** For testing potentially unsafe code or applications without risking other parts of the system
- **Snapshots:** Compromised VM can be rolled back using snapshots
- **Access Control:** Hypervisors limit who can interact with VMs and underlying resources
- **Reduced Attack Surface:** Configure VM with minimal amount of services

Security of Virtual Machines

Disadvantages

- **Hypervisor Vulnerabilities:** Compromised hypervisor exposes all running VMs to compromise
- **Shared Resources:** VMs share hardware; vulnerabilities in resource management (like CPU, memory, or I/O) can potentially be exploited to affect other VMs
  - Rowhammer (DRAM)
  - Side-channel Attacks (CPU): Spectre and Meltdown
- **Complexity:** Complexity of virtualized environments can increase the attack surface, making it harder to secure and manage.

Containers

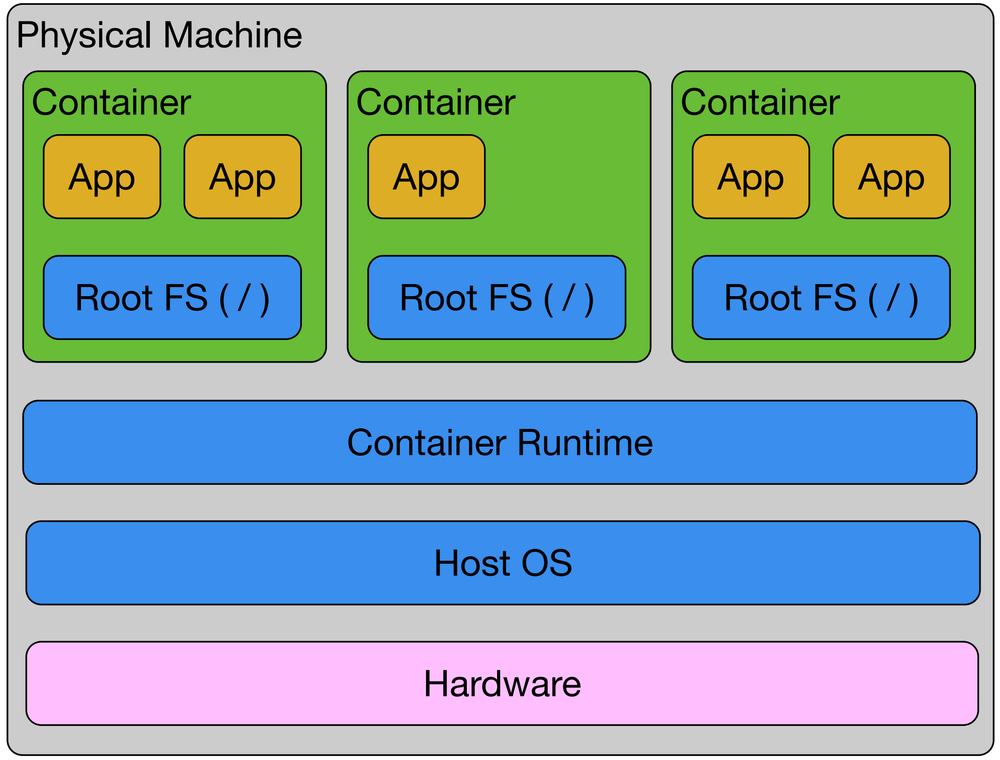

docker

What is a Container?

Lightweight, standalone, and executable software package

- Application binaries, Configuration files, Runtime libraries, Dependencies
- Pre-configured runtime environment
- Executable with a single command
  - Actually, they are just OS processes

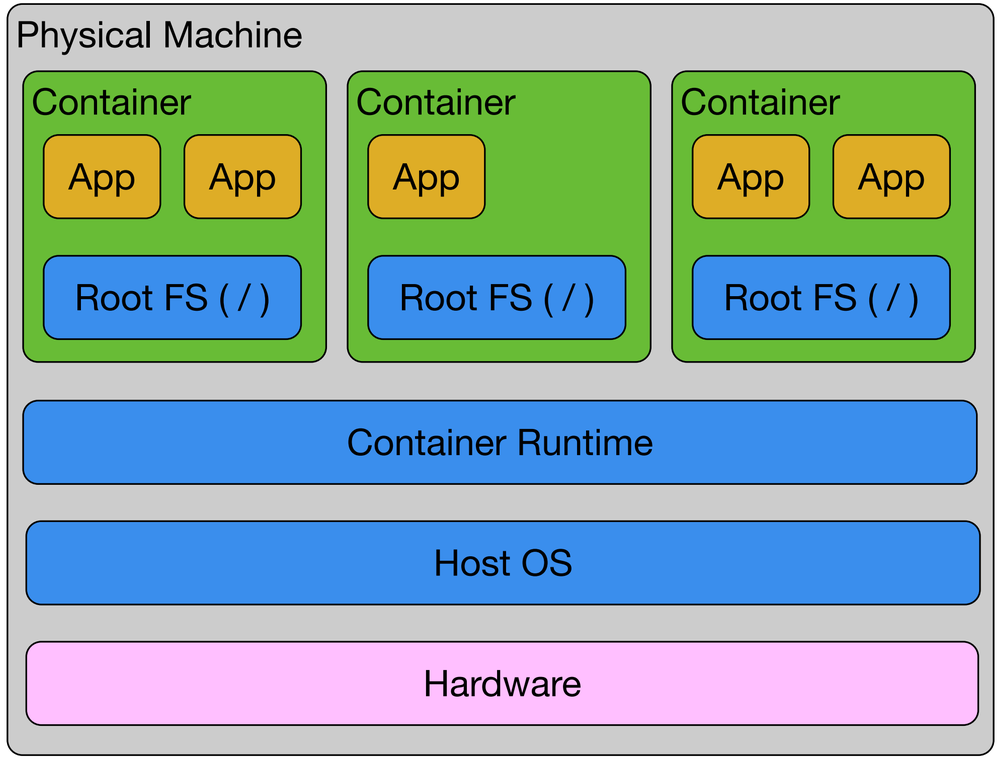
Write once, run anywhere!



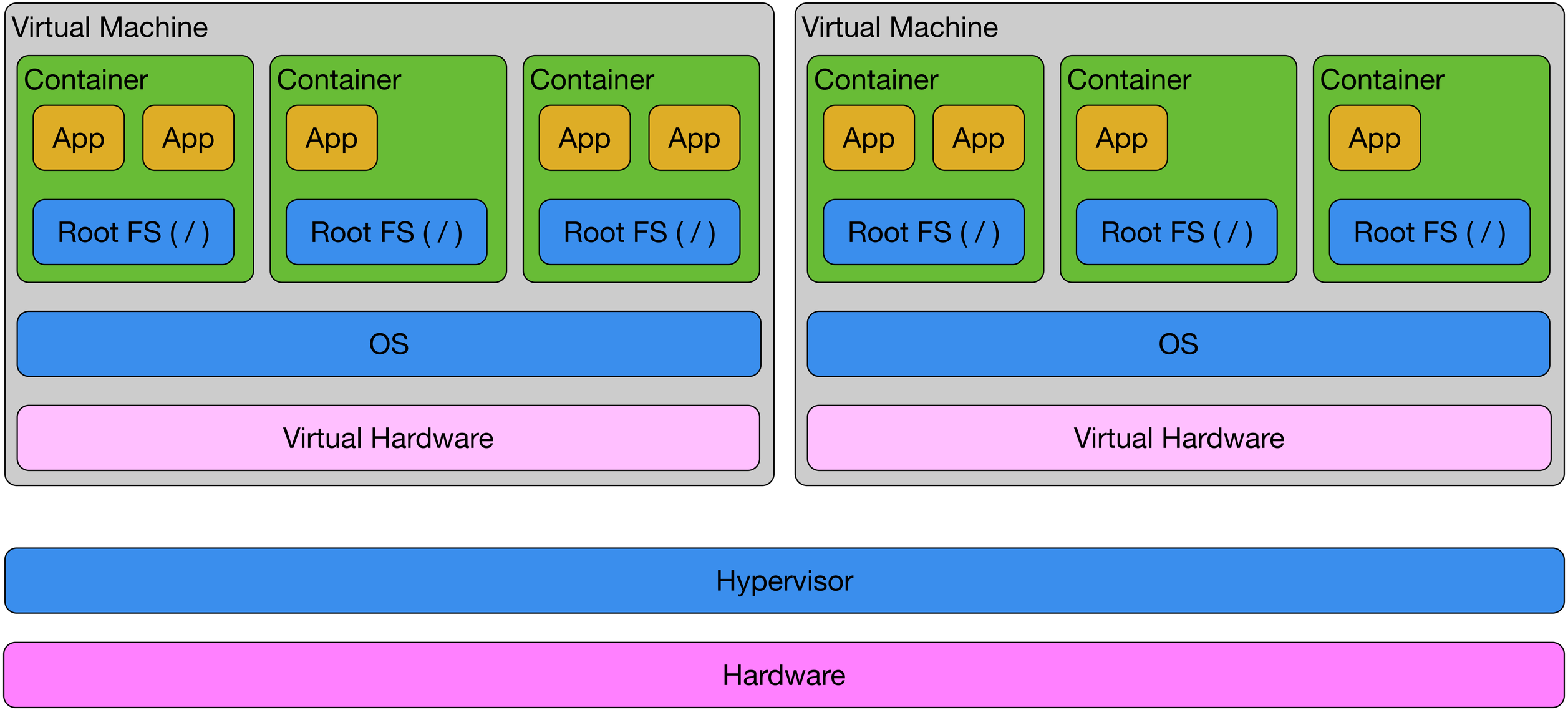
What is a Container?

Lightweight, standalone, and executable software package

- Isolate apps from each other
- Standardized packaging format
  - Open Container Initiative (OCI)
- Containers do not include the OS
  - Rely on host OS or container runtime



Containers Often Run on Virtual Machines



Containers

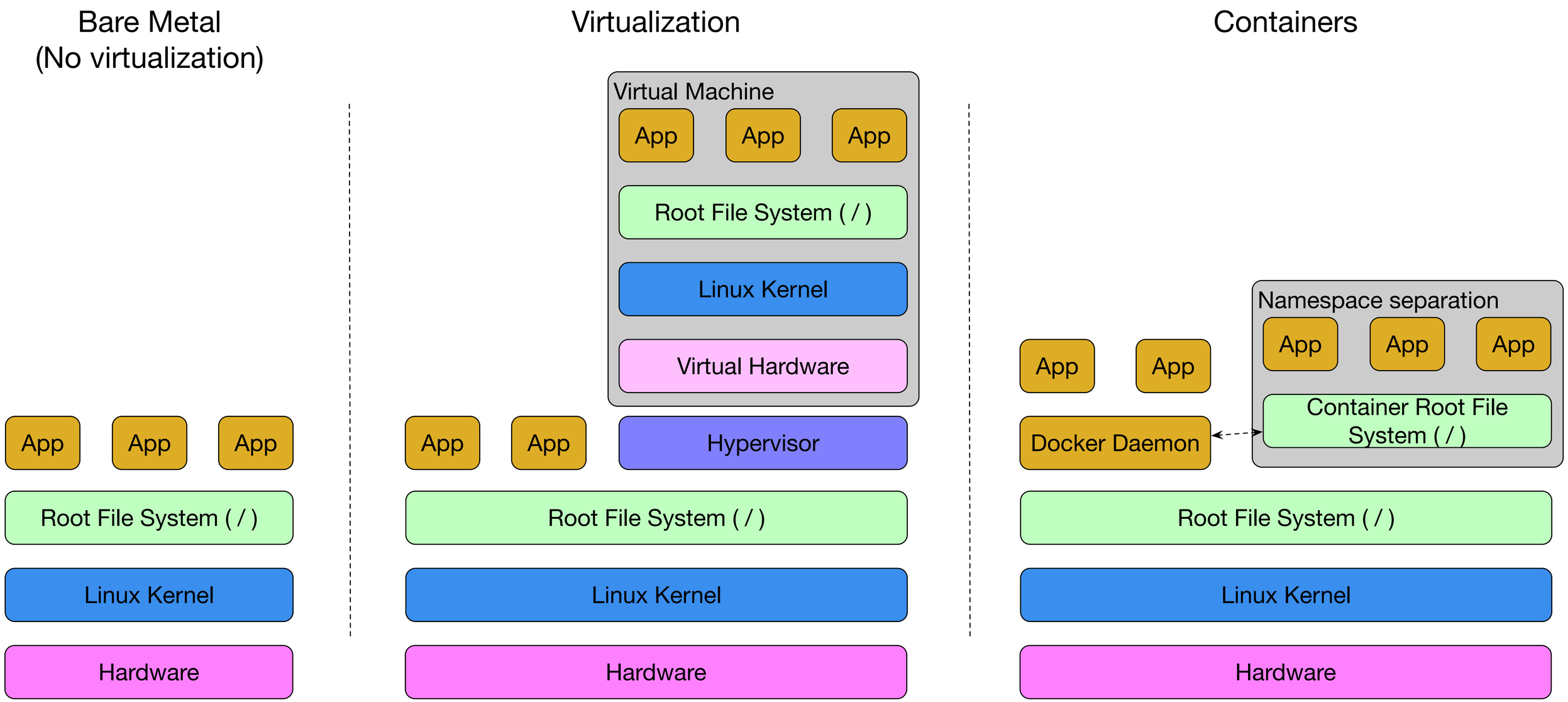
Advantages

- **Portability:** Run consistently in different environments (development, production)
- **Resource Efficiency:** Share host system's kernel (reduces overhead)
- **Isolation:** Process and filesystem isolation (avoid conflicts between applications)
  - **Security:** Isolating dependencies
- **Rapid Deployment:** Can be quickly started, stopped, and scaled
- **Migration:** Easy to move container between VMs, servers, and cloud machines

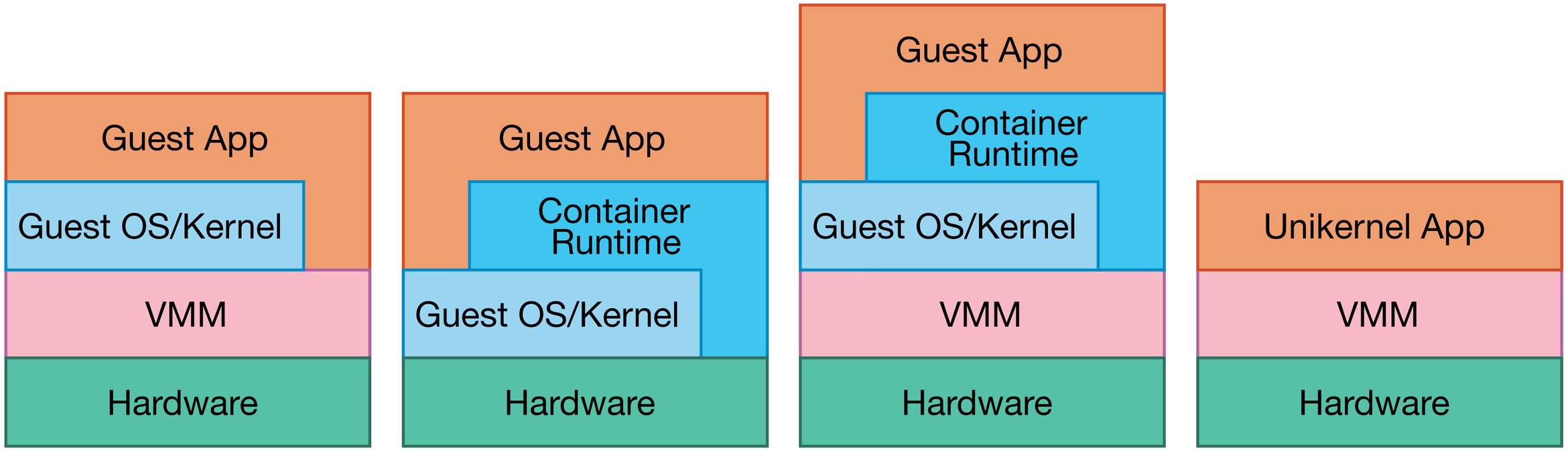
Disadvantages

- **Security Concerns:** Compromising the host's kernel can compromise container
- **Networking:** More complex than traditional environments
- **Limited Isolation:** Virtual machines offer stronger isolation than container/ process isolation

Containers vs Virtual Machines



Unikernels



(a) Traditional Virtual Machines

(b) Containers

(c) Containers in VMs

(d) Unikernels

Specialized OS/kernel

- Lightweight, single-purpose execution environment for applications
- Without any unnecessary components
- Minimize resource usage and attack surfaces
- Unikraft.cloud:
  - Use existing container images without the overhead of actually running containers on Linux

Containers vs Virtual Machines

	Containers	Virtual Machines
Boot Time	Seconds	Minutes
Runs on	Host OS Kernel	Hypervisor
Memory Efficiency	High (shares OS kernel)	Low (requires full OS install)
Isolation	Process-level (namespaces)	Strong (full OS isolation)
Deployment	Fast (lightweight)	Slower (heavyweight)
Performance	Near-native (minimal overhead)	Decent, but has some overhead
Scalability	Highly scalable	Less scalable (resource-intensive instances)
Resource Utilization	More efficient	Less efficient
Portability	Highly portable	Less portable
Security	Depends on configuration	Strong due to isolation

History of Virtual Machines and Containers

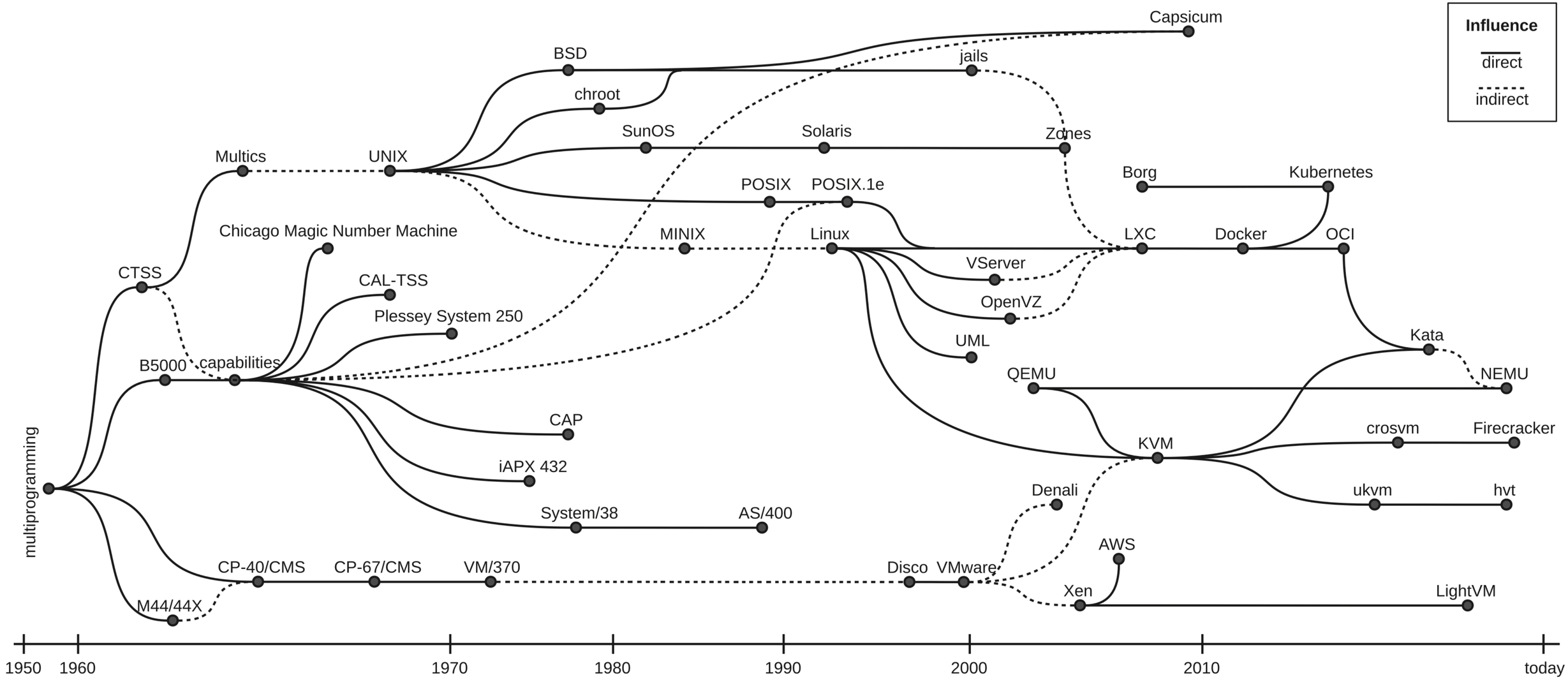


Fig. 1. The evolution of virtual machines and containers.

Source: https://dl.acm.org/doi/10.1145/3365199

OCI Containers

Open Container Initiative (OCI)

- **Image Format Specification:** Defines *standard format* for container images; allows to build, share, and run images on different container runtimes
- **Runtime Specification:** Specifies how a *compliant runtime* should run a OCI-compliant container image
- **Interoperability:** Adhering to OCI standards ensures container technologies from different vendors work together, offering flexibility in choice of tooling

Other Container Technologies

Linux Containers (LXC)

Linux has built-in support for containers

- Allows multiple Linux environments (containers) to run on a single host using a shared kernel
- Provide process isolation via **namespaces** and **control groups**
  - **cgroups:** controls the resource usage of a collection of processes
  - **namespaces:** create distinct namespaces for different system resources
    - Process ID, network, mount,… namespaces

Multi-Tenancy

Single-Tenant Architecture

- Each customer (or tenant) has their own dedicated instance of
  - Application
  - Hardware
- Provides full isolation of resources
  - Improved security
  - Better performance
  - Custom hardware (TPUs, Intel SGX, etc.)
  - Customers with stringent security requirements

Multi-Tenant Architecture

- Multiple customer (or tenant) share the same instance of
  - Application
  - Hardware
- Provides logic separation of resources
  - Lower cost
  - Better resource utilization via shared resources (CPU, memory, disk)
  - Shared maintenance
  - Common in SaaS applications (same software platform, different database)

Questions?

References

- Virtualization — https://www.ibm.com/topics/virtualization
- Virtual Machines — https://www.ibm.com/topics/virtual-machines
- Hypervisors — https://www.ibm.com/topics/hypervisors
- Containers — https://www.ibm.com/topics/containers
- Unikernels — https://unikraft.cloud/