COMPUTING PLATFORMS

Virtualization and Containers

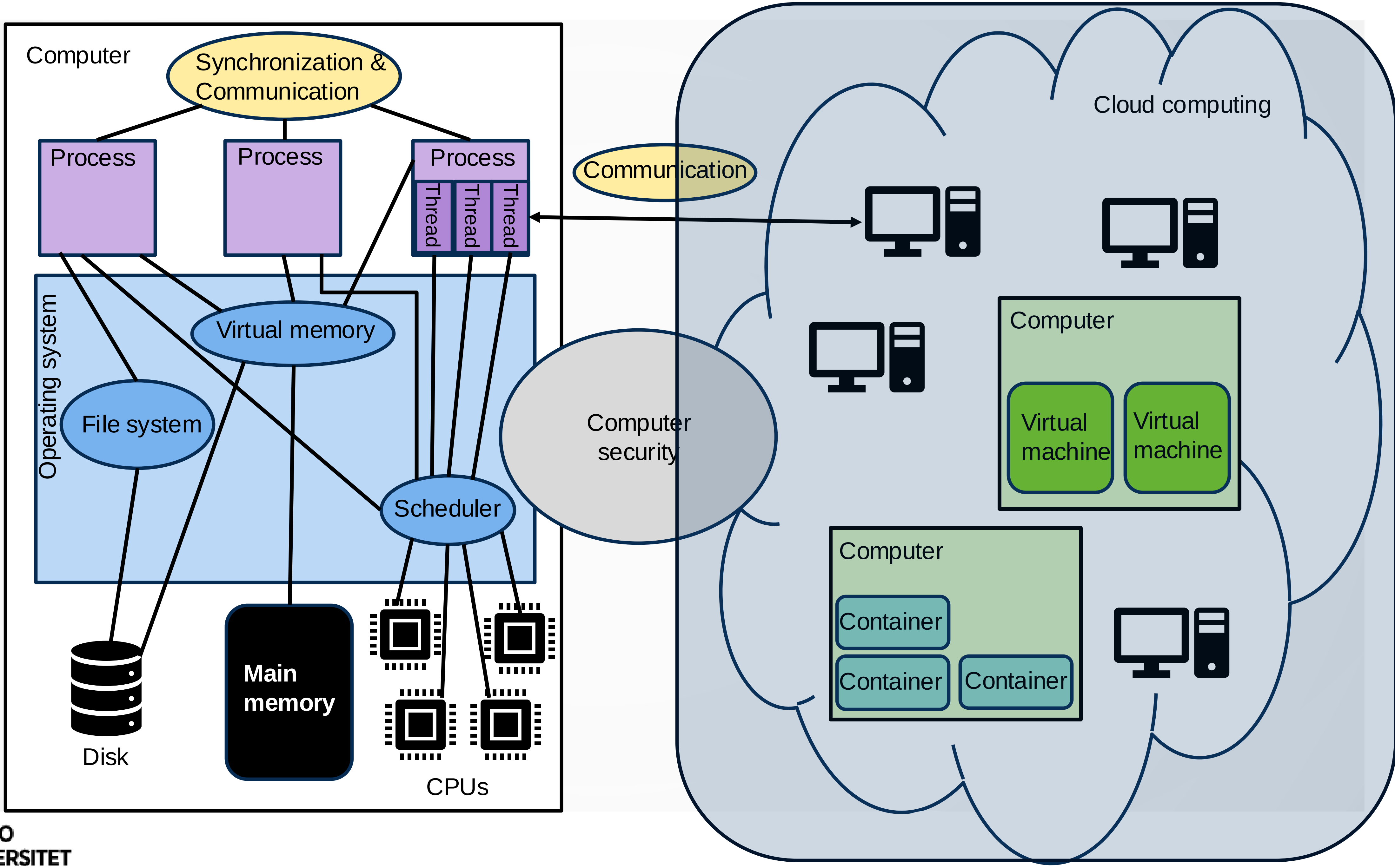
OVERVIEW OF LECTURE

- Introduction to Virtualization
  - Definition and historical context
  - Types of virtualization (hardware, software, network, storage)
- Virtual Machines
  - Architecture of VMs
  - Role in cloud computing
- Introduction to Containers
  - Comparison with VMs
- Container Orchestration
  - Introduction to Kubernetes
- Other issues with virtualization

CONTINUOUS FEEDBACK

- Comment on Norppa about importance of paging
  - Still relevant to know about it even though it is not in programmer’s direct control

CONTENTS OF THE COURSE



WHAT IS VIRTUALIZATION?

- Definition: **Abstracting physical hardware into virtual resources**
- Creates multiple simulated environments from one physical hardware system
- Enables running multiple operating systems on a single physical machine
- Isolates applications and operating systems from the hardware
- Increases resource utilization and efficiency
- Simplifies management and deployment of resources
- Supports cloud computing by facilitating scalability and flexibility
- We cover mostly general principles and use cases
  - No concrete technical details on CPU level implementation of virtualization

HISTORICAL CONTEXT OF VIRTUALIZATION

- Originated in the 1960s with mainframe computers
- IBM's development of time-sharing systems in the 1970s
- Expansion in the 1990s with virtual private networks (VPNs)
- **Early 2000s**: VMware popularizes virtualization for x86 architecture
  - Development of hypervisor technology
  - Rapid growth with cloud computing advancements
- **Modern virtualization**: integral to data centers, cloud services, and enterprise IT

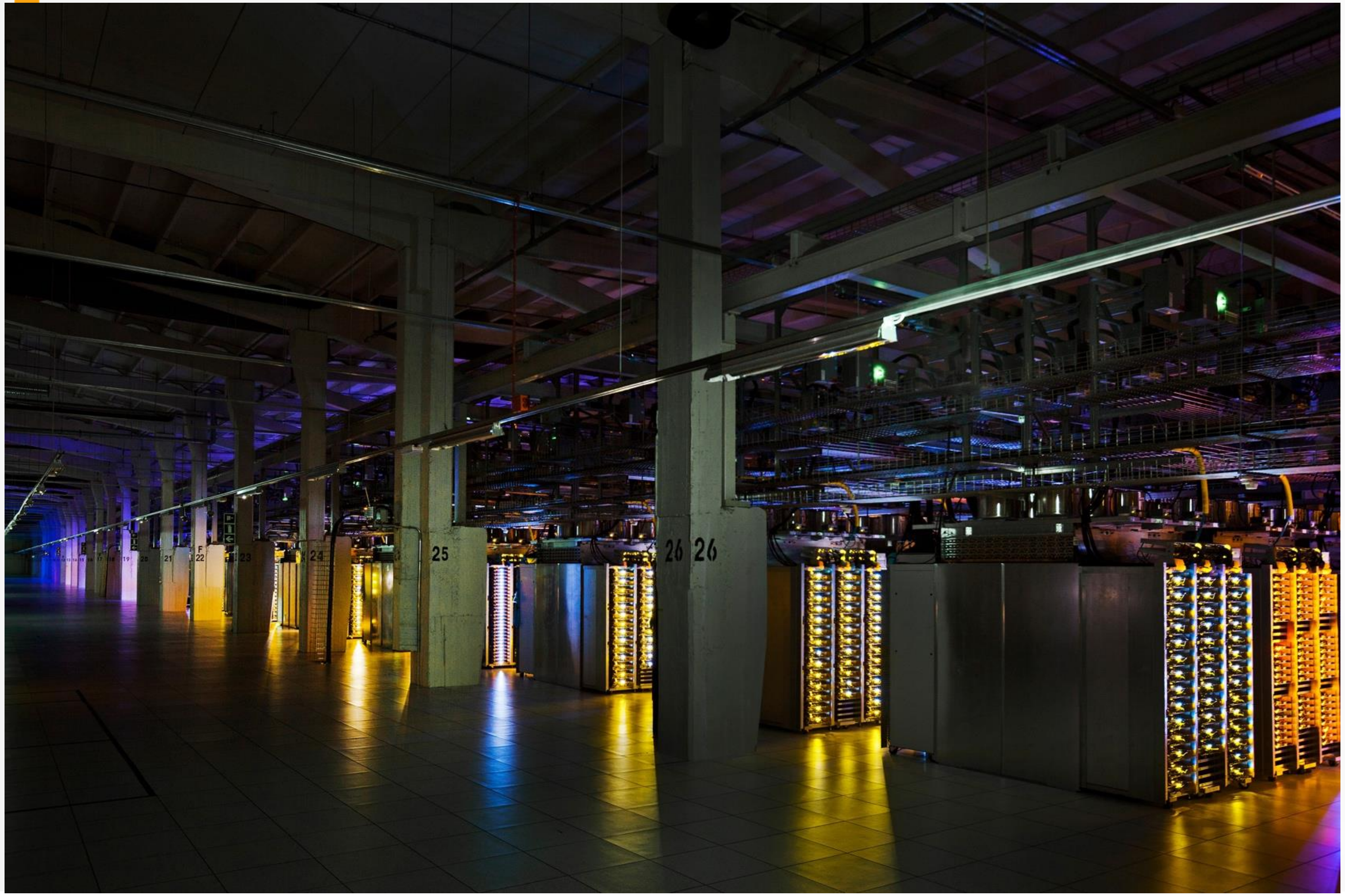
WHY VIRTUALIZATION?

- **Resource optimization**: Maximizes hardware use
- **Cost reduction**: Less physical hardware needed
- **Flexibility and scalability**: Easier resource allocation and adjustment
- **Improved disaster recovery**: Simplifies backup and recovery processes
- **Enhanced security**: Isolation of different systems and applications
- **Facilitates testing and development**: Allows for safe testing environments
- **Supports legacy applications**: Runs old applications on new hardware
- **Environmentally friendly**: Reduces physical servers, saving energy
- Main factors: Cost reduction, resource optimization, flexibility

WHAT DOES THE CLOUD LOOK LIKE?

Google's datacenter
in Hamina
Source: www.google.com

WHAT DOES THE CLOUD LOOK LIKE?



Google’s datacenter
in Hamina
Source: www.google.com

TYPES OF VIRTUALIZATION - OVERVIEW

- Categories of virtualization based on resources
  - Hardware Virtualization: Simulating entire computers
  - Software Virtualization: Emulating software environments
  - Network Virtualization: Creating virtual networks
  - Storage Virtualization: Pooling physical storage from multiple devices
  - Data Virtualization: Abstracting data management
  - Desktop Virtualization: Separating personal computing desktop environment
  - Application Virtualization: Running applications in isolated environments
- Some are more common than others

HARDWARE VIRTUALIZATION

- Creates a virtual machine (VM) that acts like a real computer
- Uses a hypervisor to allocate physical resources
- Enables multiple OS on a single physical machine
- Improves resource utilization and efficiency
- Ideal for server consolidation
- Supports cloud computing infrastructure
- Key for creating scalable IT environments
- Can be full or partial virtualization

SOFTWARE VIRTUALIZATION

- Focuses on emulating software or an OS
- Allows incompatible software to run on a host OS
- Useful for legacy application support
- Can be implemented as application or platform virtualization
- Enhances security by isolating software
- Simplifies software testing and deployment
- Reduces compatibility issues
- Facilitates easier management of software updates

NETWORK VIRTUALIZATION

- Combines hardware and software network resources
- Creates a programmable network
- Allows for multiple virtual networks on a single physical network
- Enhances security and efficiency
- Supports complex network topologies
- Facilitates efficient network management and troubleshooting
- Key for cloud services and data centers
- Provides scalability and flexibility
- Overlay networks

STORAGE VIRTUALIZATION

- Abstracts physical storage across multiple network storage devices
- Simplifies storage management
- Improves data accessibility and reliability
- Facilitates backup, archiving, and recovery processes
- Enhances performance and resource utilization
- Supports dynamic storage allocation
- Reduces storage costs
- Crucial for disaster recovery strategies

VIRTUAL MACHINES

WHAT ARE VIRTUAL MACHINES (VM)?

- **Definition**: Software emulation of physical computers
- **Components**: Hypervisor, virtual hardware, guest operating systems
- Provides isolation between multiple VMs on a single host
- Each VM operates independently with its own OS
- Used for server consolidation, testing, and legacy application support
- Enables better resource allocation and energy efficiency
- Facilitates migration and backup of entire systems
- Supports various operating systems on one physical server

VIRTUAL MACHINE ARCHITECTURE

- **Hypervisor types**: Type 1 (bare-metal) and Type 2 (hosted)
- Hypervisor manages physical resources
- Emulated virtual hardware: CPU, memory, storage, network
- Virtualization of CPUs and memory
- Storage virtualization: Virtual Hard Drives
- Network interface Virtualization
- Integration with physical host resources
- Security considerations in VM architecture?

TYPE 1 HYPERVISORS

- Bare-metal hypervisors
- Run directly on the host's hardware
- Main properties:
  - Direct hardware access: Runs directly on the host's physical hardware
  - Higher performance: Offers high efficiency and performance
  - Isolation: Provides strong isolation between virtual machines
  - Security: Generally more secure due to less layers
  - Resource management: Efficiently manages and allocates hardware resources
  - Scalability: Highly scalable, suitable for enterprise environments
  - Examples: VMware ESXi, Microsoft Hyper-V for Servers, Citrix XenServer

TYPE 2 HYPERVISORS

- Hosted hypervisors
- Run on a conventional operating system
- Main properties:
  - Host operating system dependency: Runs on top of a host operating system
  - Ease of use: Generally easier to install and manage
  - Reduced performance: Lower performance compared to Type 1
  - Flexibility: Offers more flexibility for desktop or testing environments
  - Resource Overhead: Additional overhead due to the host OS layer
  - Examples: Oracle VirtualBox, VMware Workstation, Parallels Desktop

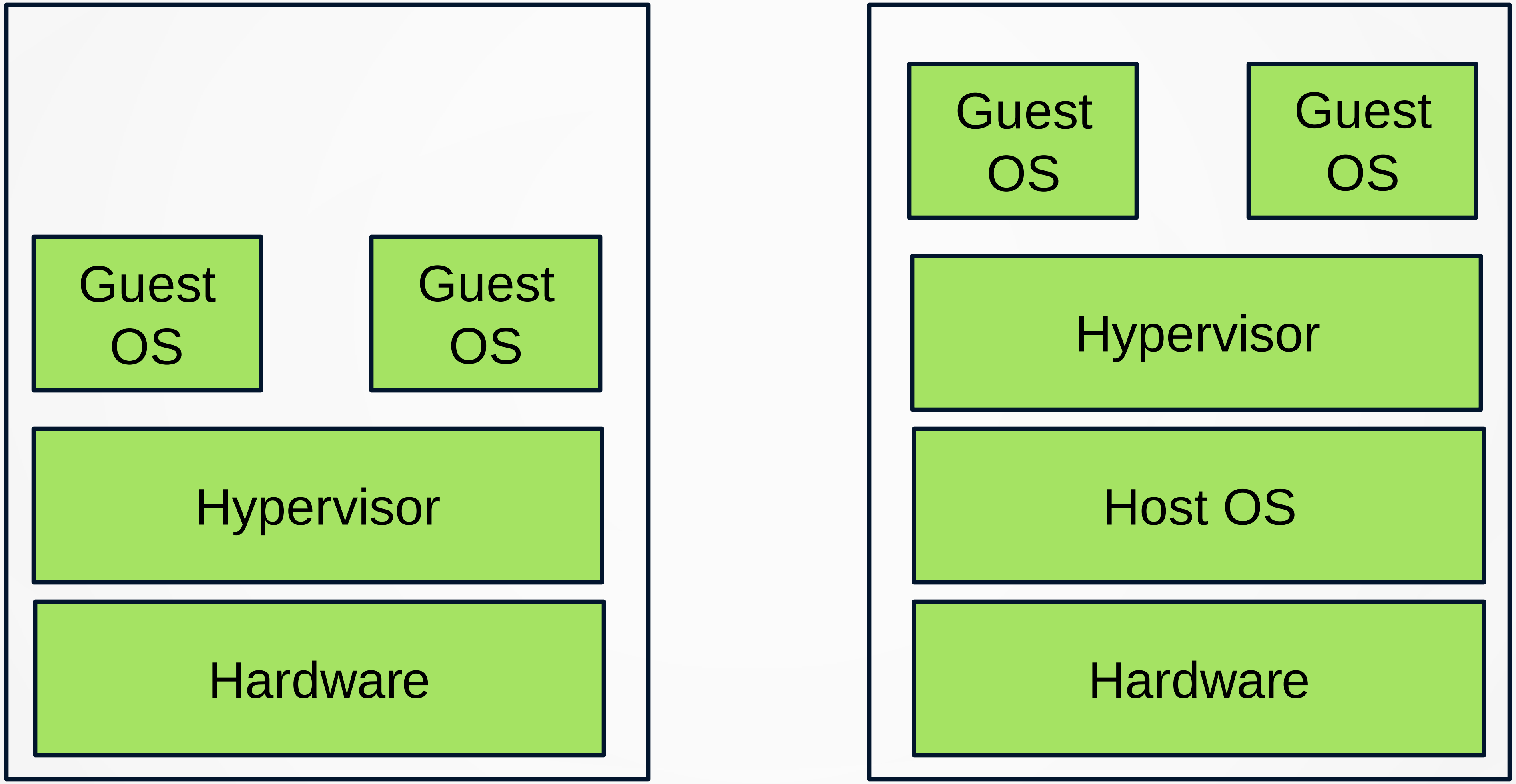
PARAVIRTUALIZATION

- **Definition of paravirtualization**: A virtualization technique in which the guest operating systems are modified to work in a virtualized environment, improving performance by reducing the overhead typically associated with full virtualization.
- More efficient, cooperative approach between the host and guest systems.
- Near-native performance
  - Allows guests to directly call the hypervisor for resource management
- Example system: Xen Hypervisor
  - An open-source hypervisor using paravirtualization
- Key difference to Type 1 and 2: **Guest OS needs modifications**

COMPARISON BETWEEN TYPE 1 AND 2

- **Operating layer**: Type 1 runs directly on hardware, Type 2 runs on top of a host OS
- **Performance**: Type 1 offers better performance and efficiency due to direct access to physical hardware
- **Use cases**: Type 1 is more suited for enterprise environments, whereas Type 2 is better for individual users and smaller scale deployments
- **Security**: Type 1 is generally more secure, as it has a smaller attack surface
- **Installation and management**: Type 2 is easier to install and manage
- **Resource utilization**: Type 1 has more efficient resource utilization, Type 2 can have additional overhead from the host OS.
- Both have advantages and disadvantages: Which is better depends on use case

COMPARISON AS A PICTURE



Type 1 Type 2

ROLE OF VMS IN CLOUD COMPUTING

- Foundation for Infrastructure as a Service (IaaS)
- Enables rapid provisioning of resources
- Facilitates scalability and elasticity in cloud environments
- Supports multi-tenancy and resource sharing
- Critical for disaster recovery and business continuity in the cloud
- Enhances security through isolation in the cloud
- Allows for workload mobility across cloud environments
- Reduces physical infrastructure costs and complexity
- VMs were key technology in cloud computing expansion
    - More about cloud computing in next lecture

CONTAINERS

INTRODUCTION TO CONTAINERS

- **Definition**: Lightweight, executable packages of software
- Containers include application and all its dependencies
- Provides consistent environment across development, testing, and production
- **Rapid deployment**: Containers can be created, replicated, and destroyed quickly
- **Resource efficiency**: Shares the OS kernel, uses less resources than VMs
- **Isolation**: Each container operates independently
- Popular containerization platforms: Docker, Kubernetes
- Use cases: Microservices architecture, DevOps practices

WHAT IS DOCKER?

- **Docker**: Leading software platform for containerization
- Provides tools to package, ship, and run applications in containers
- Docker Hub: Repository for Docker images
- Dockerfiles: Scripts to automate building Docker images
- Supports cross-platform compatibility
- Integrates with various CI/CD tools
  - CI/CD = Continuous Integration, Continuous Delivery
- Ecosystem includes Docker Compose, Docker Swarm, Docker Desktop
- Widely used in industry for development and deployment

CONTAINER ECOSYSTEM

- **Key components**: Runtime, Orchestration, Registry, Networking, Storage
- **Container runtimes**: Docker, containerd, rkt
- **Orchestration tools**: Kubernetes, Docker Swarm, Amazon ECS
- **Image registries**: Docker Hub, Quay.io, AWS ECR
- **Networking in containers**: Overlay networks, bridge networks
- **Persistent storage solutions**: Volumes, bind mounts

EXAMPLE: DOCKER COMPOSE

- A tool for defining and running multi-container Docker applications
- Uses a YAML file to configure application services, networks, and volumes
- Allows users to launch, execute, and manage multiple containers as a single service
- Ideal for local development and testing, ensuring consistency across environments
- Easily links together multiple containers and manages their dependencies
- Offers a straightforward CLI for managing the lifecycle of your application
- Supports extending and overriding configurations for different environments
- Wide support by the Docker community, with an ecosystem of shared configurations

EXAMPLE: DOCKER SWARM

- A native clustering and orchestration tool for Docker containers
- Turns a pool of Docker hosts into a single, virtual Docker host
- Ensures high availability and redundancy through multiple manager nodes and worker nodes
- Automatically distributes container workloads for optimal resource utilization
- Allows for easy scaling of applications up or down as needed
- Users can declare the desired state of a service, and Docker Swarm ensures the system's state matches the user's intentions
- Built into the Docker platform, providing a seamless user experience
- Offers secure node communication and provides ways to encrypt container data traffic

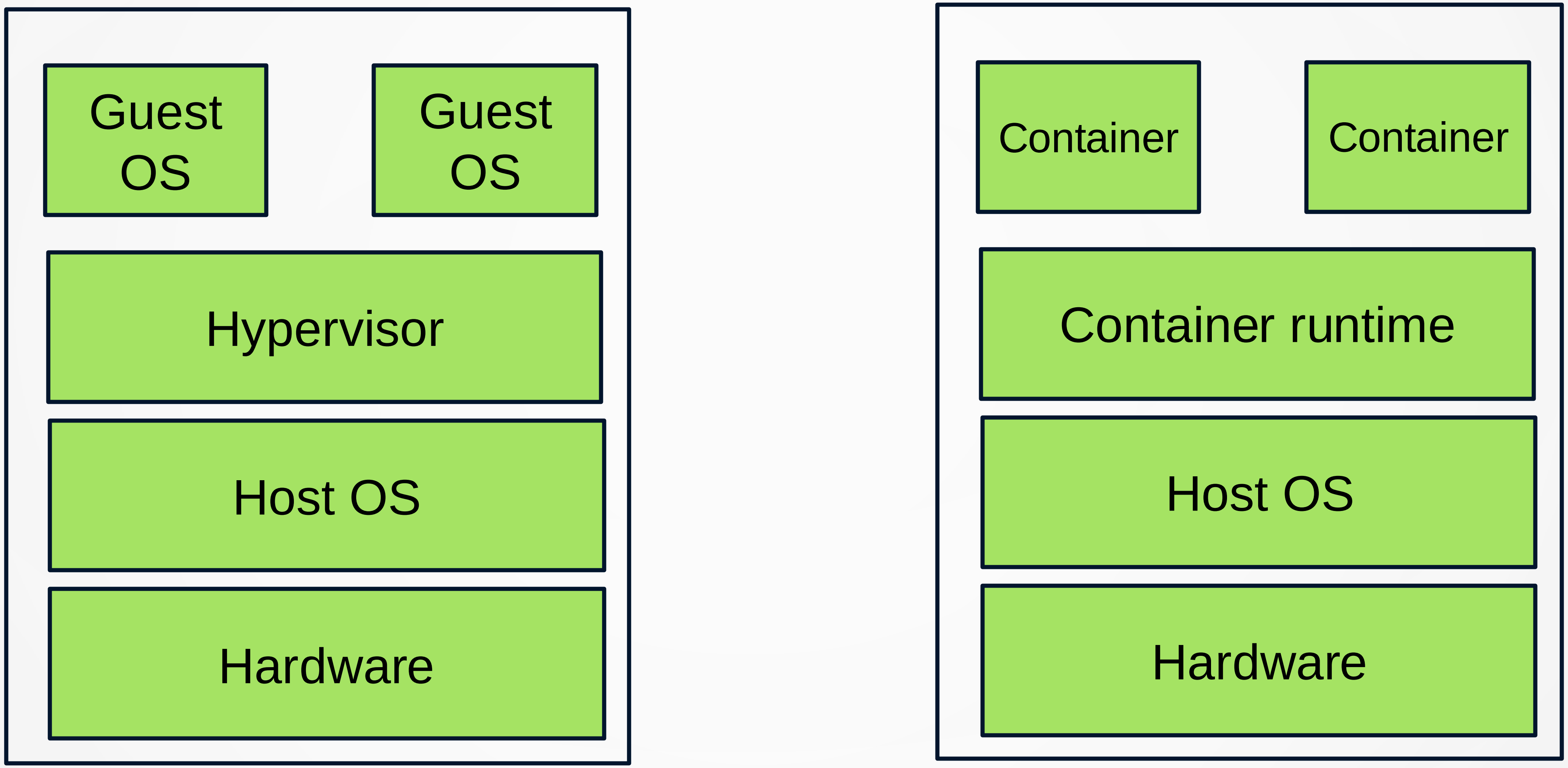
EXAMPLE: DOCKER DESKTOP

- Application for desktop: Enables building and sharing containerized applications and microservices
- Comes bundled with Docker Engine, Docker CLI client, Docker Compose, Docker Content Trust, Kubernetes, and Credential Helper
- Available for both Windows and Mac: Consistent experience across operating systems
- Includes a GUI for managing containers, viewing logs, and configuring settings
- Offers a single-node Kubernetes cluster, making it easy to test deployments locally
- Simplifies volume management for persisting data
- Includes easy-to-configure networking options like bridge networks and port forwarding for container communication
- Integrates seamlessly with development workflows, supporting CI/CD pipelines for containerized applications

VMS VS CONTAINERS

- VMs virtualize the hardware, Containers virtualize the OS
- VMs have full OS images for each instance, Containers share the host OS kernel
- Start-up time: VMs take minutes, Containers take seconds
- Resource utilization: VMs require more resources, Containers are lightweight
- Isolation level: VMs provide strong isolation, Containers offer process-level isolation
- Scalability: Containers are more scalable due to their lightweight nature
- Use cases: VMs for full environment isolation, Containers for microservices

CONTAINERS VS. VMS AS PICTURE



Type 2

Containers

CONTAINERS VS. VMS: KEY POINTS

- Container systems conceptually same as Type 2 virtualization
  - Host OS, ”resource management layer”, ”virtualized entities”
- Practical difference:
  - Containers on one machine all share the host OS
  - VMs can run different Guest OSes
- Containers have many advantages, but sometimes full VMs are better
  - Nowadays containers are much more popular and becoming more used

VMS: ADVANTAGES

- VMs offer better isolation and security
  - Orchestration frameworks like Kubernetes mitigate this somewhat
- VMs are more flexible
  - Possible to run Windows VM on Linux
  - Containers on one machine limited to one host OS
- Running legacy software that needs a specific OS
  - Can set this up with containers, but takes more work generally

BENEFITS OF CONTAINERS: SUMMARY

- **Portability**: Run across different environments consistently
- **Faster deployment**: Quick to start and easy to scale
- **Efficient resource utilization**: Lower overhead than VMs
- **Environment consistency**: Avoids "works on my machine" issues
- **Simplifies CI/CD pipeline**: Easy integration and testing
- Facilitates microservices architecture
- Enhances DevOps practices: Seamless development and deployment
- **Cost-effective**: Reduces need for additional hardware
- VMs have their use cases, but containers are becoming more common

CONTAINER ORHCESTRATION

OVERVIEW OF CONTAINER ORCHESTRATION

- Automating deployment, scaling, and management of containerized applications
- Essential for managing large-scale, distributed container deployments
- Includes service discovery, load balancing, and resource allocation
- Increased efficiency, better resource utilization, and fault tolerance
- **Common orchestration tools**: Kubernetes, Docker Swarm, Apache Mesos
- **Orchestration vs. containerization**: Expands upon basic containerization
- **Use cases**: Ideal for microservices architecture and cloud-native applications
- **Evolution**: From manual management to automated orchestration

INTRODUCTION TO KUBERNETES

- Developed by Google
  - Now maintained by the Cloud Native Computing Foundation (CNCF)
- Open-source platform for automating deployment, scaling, and operations of application containers
- Leading tool in container orchestration
- Modular and scalable
- Kubernetes clusters: Collections of nodes that run containerized applications
- Community and ecosystem: Rich set of tools and widespread community support

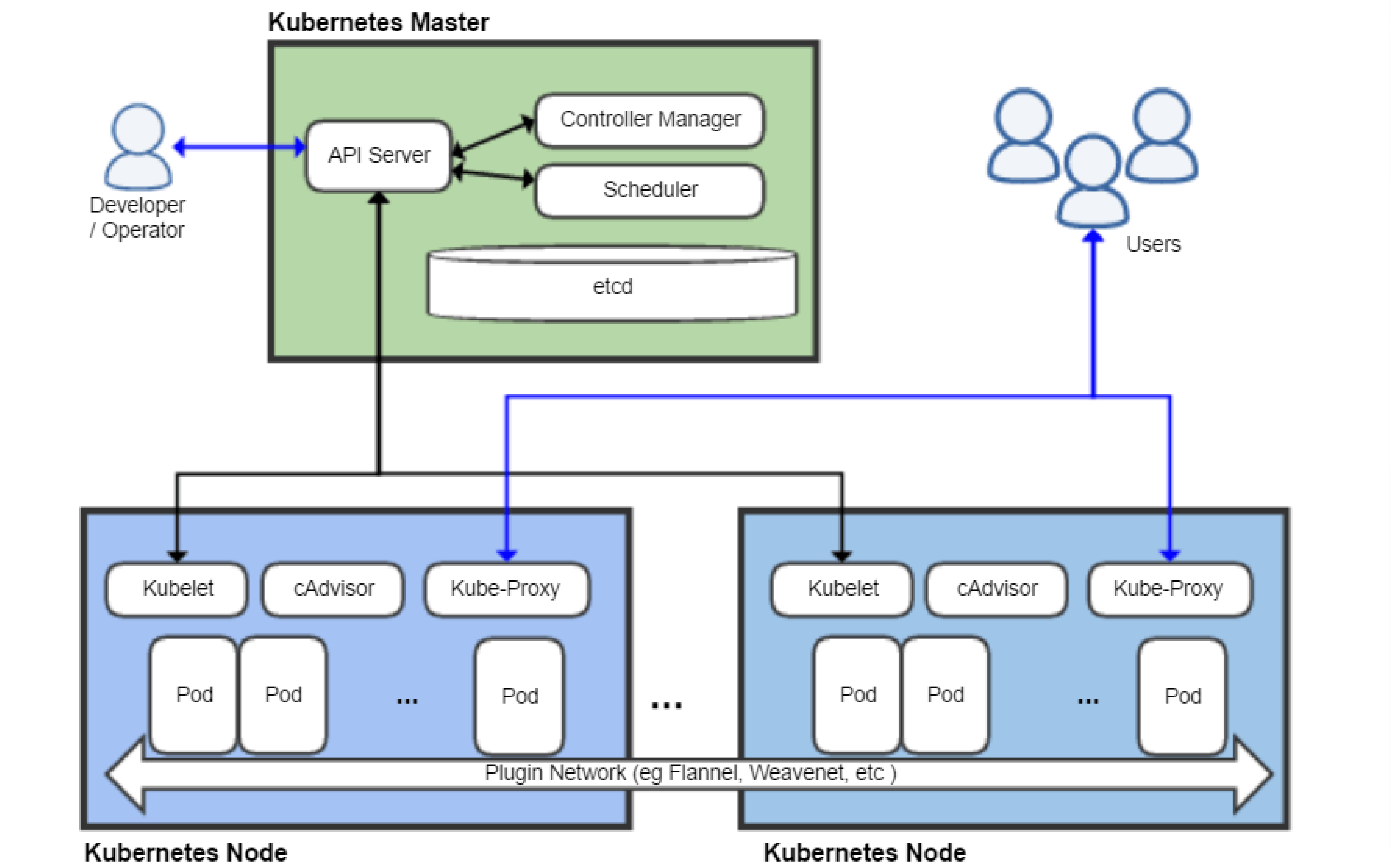
KEY FEATURES OF KUBERNETES

- **Automated rollouts and rollbacks**: Manages changes to applications and configurations
- **Self-healing**: Automatically restarts failed containers, replaces, and reschedules
- **Load balancing and service discovery**: Distributes network traffic and organizes containers
- **Horizontal scaling**: Simple command or UI adjustments to scale applications
- **Storage orchestration**: Automatically mounts storage system of choice
- **Secret and configuration management**: Manages sensitive information and application configurations
- **Batch execution**: Manages batch and CI workloads, replacing containers that fail

KUBERNETES ARCHITECTURE AND COMPONENTS

- Master node: Central control panel for managing cluster operations
- Worker nodes: Host the actual applications running in containers
- Pods: Smallest deployable units that can be created and managed
  - Pod may contain one or more containers
- Services: Abstraction layer for pod networking and load balancing
- Replica sets: Ensures specified number of pod replicas are running
- Deployments: Abstraction for pods and replica sets, enabling declarative updates
- Kubernetes dashboard: Web-based UI for managing Kubernetes clusters
- Namespaces: Segregates cluster resources between multiple users

KUBERNETES ARCHITECTURE



OTHER ISSUES

PERFORMANCE CONSIDERATIONS

- Understanding resource allocation: CPU, memory, and storage impacts
- Balancing workload performance in virtualized environments
- Network performance: bandwidth and latency considerations
- Impact of overcommitting resources in VMs and containers
- Performance monitoring and optimization tools
- The role of storage I/O in virtualization performance
- Impact of virtualization on application performance
- Above are things to keep in mind when developing containerized applications

FUTURE TRENDS IN VIRTUALIZATION

- Growth of containerization and microservices architecture
- Increasing adoption of serverless computing models
  - More details about serverless computing in next lecture
- Evolution of Kubernetes and container orchestration platforms
- Edge computing and its integration with virtualization technologies
- AI and ML in managing and optimizing virtual environments
- Hybrid cloud solutions and cross-platform virtualization
- Security advancements in virtualized infrastructures
- Sustainability in virtualization: energy-efficient data centers

SUMMARY

- Introduction to Virtualization
- Virtual Machines
- Introduction to Containers
- Container Orchestration

- Overview of concepts and their properties

- Next lecture: Cloud computing and putting these into use