CLOUD COMPUTING

UNIT-2

Virtualization

feedback/corrections: vibha@pesu.pes.edu

VIBHA MASTI

VIRTUALIZATION

- Abstraction of physical resources into logical view
- Compute, memory, storage, networking

Compute Virtualization

- server virtualization
- Virtual Machines - break dependency b/w OS and h/w
- VM = OS + application
- Virtual Machine Monitor (VMM) / Hypervisor - layer of s/w

(i) Type 1 / Bare Metal

- VMM b/w hardware and os
- VMM directly manages hardware
- VMM acts as traditional OS
- 3 requirements
  - * identical env to programs as original machine
  - * at worst, minor reduction in performance
  - * VMM complete control of hardware
- Eg: xen, VMware ESX server, IBM CP/CMS

(ii) Type 2 / Hosted

- VMM on top of OS
- VMM : software level representation of hardware
- VMM can also be part of OS
- Eg: Oracle VirtualBox, VMWare Fusion, KVM for Linux

(iii) Type 3 / Hybrid

- VMM directly on hardware but leverages features of existing OS by running as a guest
- Eg: MS Virtual Server, MS Virtual PC

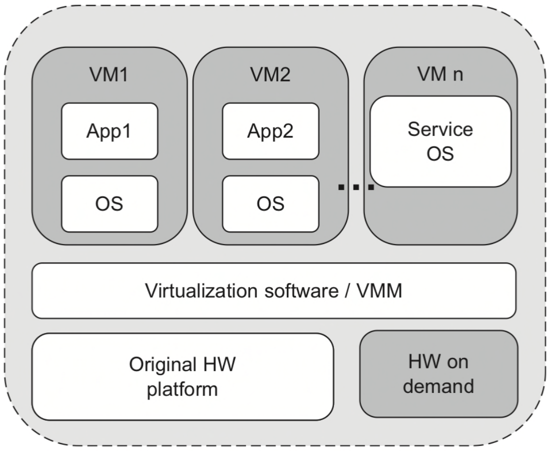


FIGURE 9.2

Hybrid hypervisor.

Physical Machine

- OS: dual mode
  - * Ring 0: kernel mode, all physical resources
  - * Ring 3: user mode, safe instructions – guest OSes in user mode, interrupt for privileged instructions and VMM takes control

or ring 1

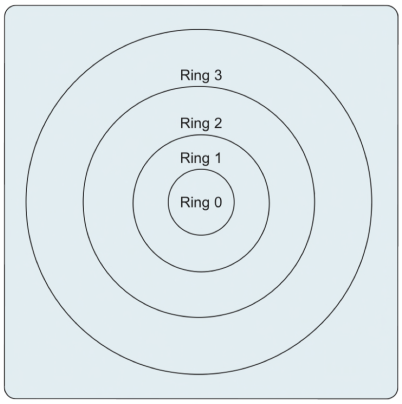
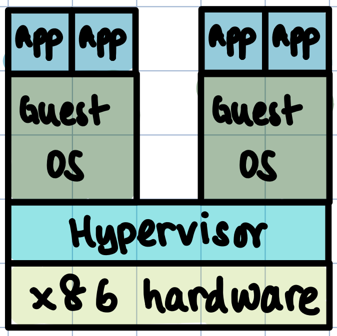


FIGURE 9.3

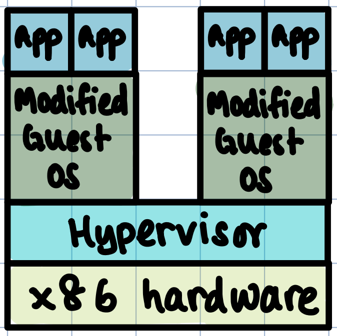
X86 protection rings.

PARAVIRTUALIZATION

- OS modified for guest
- Similar but not identical software interface to VMs
- Unlike full/transparent virtualization



FULL VIRTUALIZATION



PARA VIRTUALIZATION

- Full: intercepts & emulates privileged instructions at runtime
- Para: guest OS kernel modified & privileged instructions replaced with hypercalls – at compile time
- Attempts to improve performance speed
- Eg: xen, KVM, ESX
- Read tb/slides for more

Execution

(1) Direct Execution

- Run most instructions directly on the hardware
- How to ensure isolation?
- Close to native performance

(2) Trap and Emulate

- Trap to hypervisor when VM tries to execute a sensitive/privileged instruction

- Innocuous instructions directly on hardware
- Attempt to execute system instr in user mode → trap/gpf (general protection fault)
- Trap to VMM (running in Ring 0)
- VMM jumps to guest OS trap handler
- Issues
  - * performance overhead
  - * not all architectures support
  - * sensitive instructions (pushf, popf)
  - * guest OS can realize it is running at lower privilege level (CS, code segment)
  - * memory protection

Strictly Virtualizable

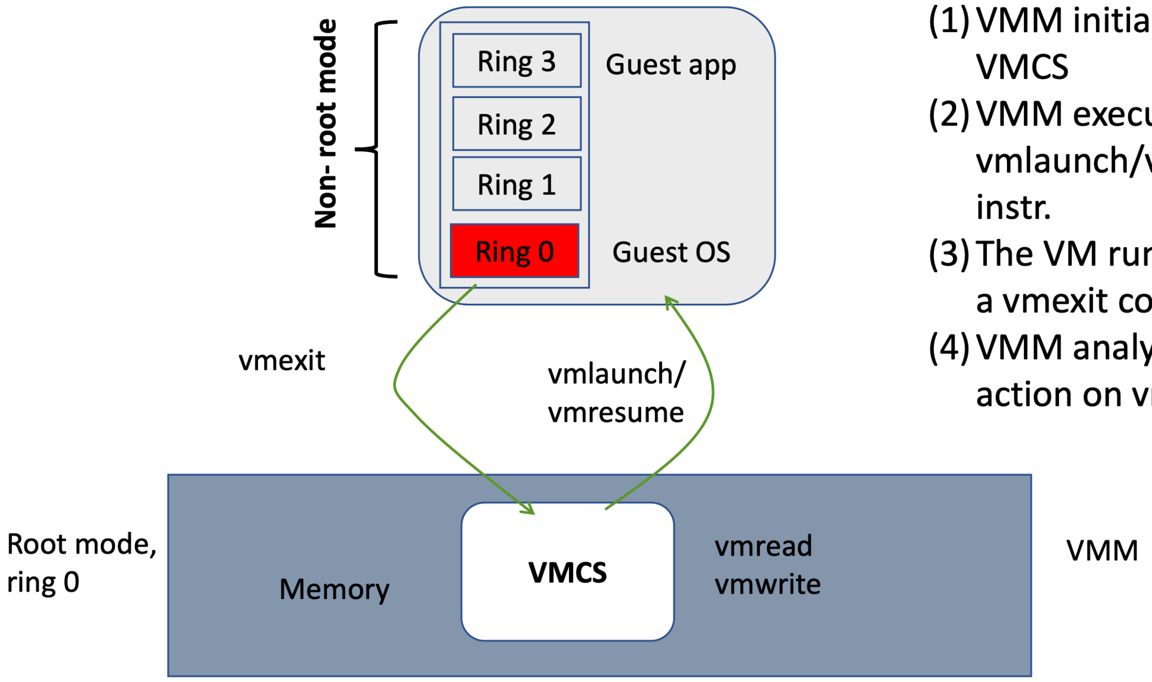
- If executed in a lower privileged mode
  - * all instr that access privileged state trap
  - * all instr execute identically or trap

(3) Binary Translation

- Does not req h/w virtualization features
- Hypervisor examines guest OS code for unsafe/sensitive but privileged instructions
- Translates to safe/privileged equivalents
- Supports full virtualization
- Translation
  - (i) Identical (safe/priv)
  - (ii) Inline translation (simple dangerous)
  - (iii) Call-outs (other dangerous)

(4) Hardware-assisted virtualization

- 2 modes: root & non-root
- Each has rings 0-3
- VMs in non-root, hypervisor in root
- Sensitive instruction in non-root
  - * executed by non-root proc
  - * trap to hypervisor



(1) VMM initializes/modifies VMCS

(2) VMM executes vmlaunch/vmresume instr.

(3) The VM runs until it hits a vmexit condition

(4) VMM analyzes and takes action on vmexit

QEMU

- Quick emulator
- Dynamic BT (type 2)

Address Translation

(i) Unvirtualized System

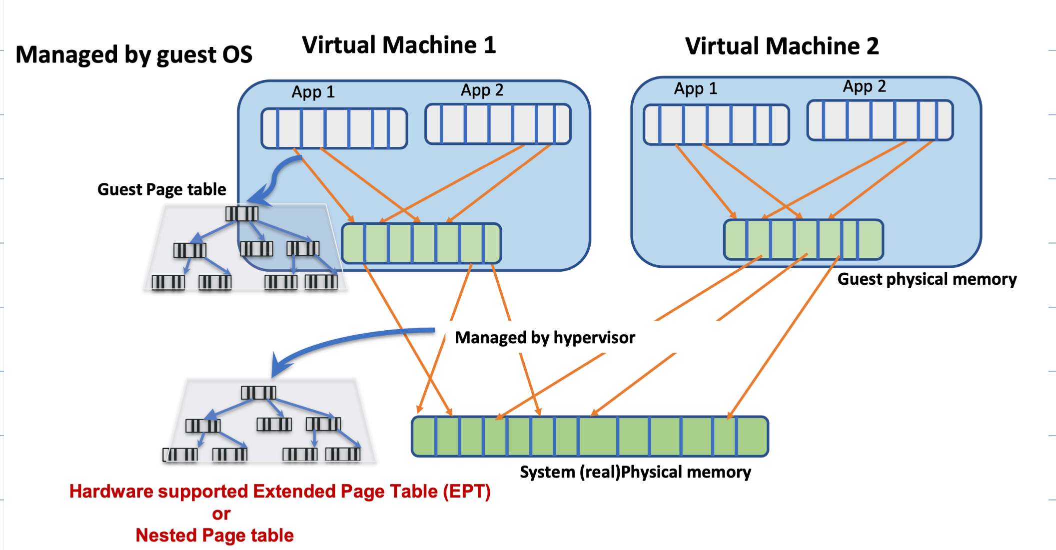
- Each process has virtual address space
- Its own page table

(ii) Virtualized system

- VMMs manage physical mem

(a) Nested Page Table

- 2 levels of translation
- Newer; no need for shadow ; better



(b) Shadow Page Table

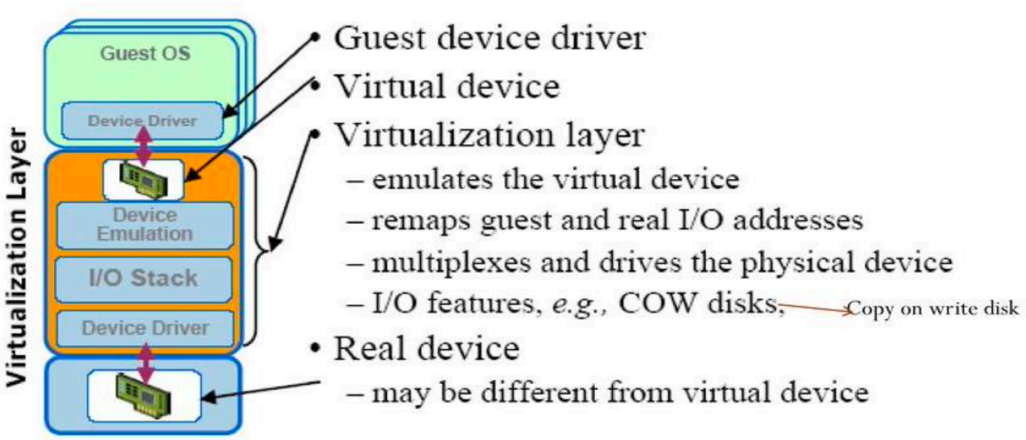
- S/W only technique
- Each guest OS PT has separate PT in VMM mapping guest VA to host PA directly
- Guest PT modifications (guest VA to guest PA) synced to shadow page tables
- Intercept guest PT modifications

I/O VIRTUALIZATION

- 3 ways
  1. Full device emulation
  2. Para virtualization
  3. Direct I/O

1. Full device emulation

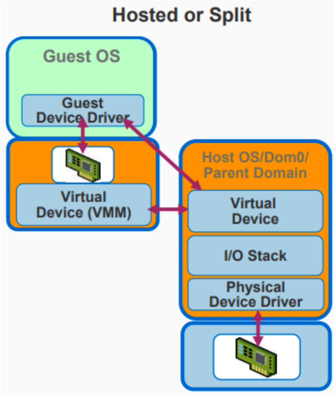
- Software in VMM
- Acts as virtual device
- I/O access trapped in VMM



- Guest device driver
- Virtual device
- Virtualization layer
  – emulates the virtual device
  – remaps guest and real I/O addresses
  – multiplexes and drives the physical device
  – I/O features, *e.g.*, COW disks; Copy on write disk
- Real device
  – may be different from virtual device

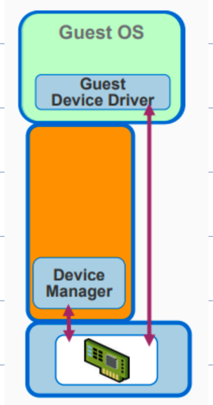
2. Paravirtualization / Split Driver / Hosted

- Frontend driver & backend driver
- Backend driver multiplexes I/O data of diff VMs
- Better performance
- Higher CPU overhead



3. Direct / Passthrough

- VM directly accesses
- Drivers on VM directly write to device registers
- One VM per device
- Scalability limits



GOLDBERG-POPEK PRINCIPLES

- 1974
- Requirements for architecture to efficiently support virtualization
  - (i) Equivalence
  - (ii) Resource control (VMM total control of resources)
  - (iii) Efficiency (majority of instruction without VMM intervention / trap)
- All instructions one of three types
  - (i) Privileged (cause trap)
  - (ii) Sensitive (access low-level machine states)
    - * Behavior sensitive (behavior depends on mode)
    - * Control sensitive (modify sys registers)
  - (iii) Safe

Theorem 1

- VMM maybe constructed if sensitive instructions subset of privileged

Theorem 2

- Computer virtualizable if
  - * Virtualizable and
  - * VMM can be constructed for it (no timing dependencies)

Theorem 3

- Hybrid VMM can be constructed if sensitive instructions subset of privileged

Note:

- Old (pre-2005) x86 not PG virtualizable
- Read slides for eg

VM Migration

1. Cold migration (powered off)
2. Offline / non-live (paused)
3. Live / hot (powered on, no disruption to service)

- Reasons - slides

1. COLD MIGRATION

- VM execution suspended before migration
- Resumed after
- Memory pages migrated only once
- Short, predictable

3. LIVE MIGRATION

- Degrades performance of running apps
- Two techniques
  - * Pre-copy
  - * Post-copy

(a) Pre-copy technique

1. Select destination host
2. Reservation of resources
3. Iterative pre-copy rounds
   ↳ execution state in memory
   ↳ entire memory data transferred
   ↳ migration controller keeps copying
   ↳ stop when threshold reached
4. Stop and transfer VM state
   ↳ suspend
   ↳ copy remainder of memrry
   ↳ non-memory (CPU, network states) sent
   ↳ downtime
5. Commitment
6. VM activation at destination
   ↳ network connection redirected

- Advantages
  * low downtime

- Disadvantages
  - * repeated copying of dirty pages → increase migration time

- Eg: KVM, Xen, VMWare hypervisor

(b) Post-copy technique

- Processor state transferred before memory
- Instant resume
- Memory contents transferred almost at once after destination begins running
- For pages not in TLB, page fault generated
- Pages fetched from source machine
- Techniques to reduce no. of page faults
  - * **Demand paging:** page faults → retransmission from source
    - ↳ slow, simple
  - * **Active push:** keeps pushing pages
    - ↳ if page fault, demand paging
  - * **Memory prepaging:** predicts memory pages most likely to be accessed
    - ↳ reduce page faults

- Disadvantages
  - * every page fault suspends dest VM until required page received

Issues with Live Migration

1. Memory migration

↳ Internet-Suspend-Resume : tree of small subfiles that were modified since migration started

2. File system migration

- Two approaches
  - * virtual disk contents transferred
  - * global file system across possible VM hosts (prevents file transfer)

3. Network Migration

- VM assigned virtual IP address known to other entities
- Migrating maintains IP address

CONTAINERS

- OS-level virtualization (LXC)
- Run multiple isolated Linux systems on host with single OS
- Own process & network space
- Shares host OS's kernel

Docker

- Create, test, ship, deploy apps using containers
- Client-server architecture
- Docker client talks to Docker daemon (REST API)
- Daemon can be local or remote
- Docker Compose - another client
- Docker daemon can communicate with other dockerds
- Docker client - docker
- Docker host runs daemon and hosts/connects to a Docker Registry

- Docker objects - images, containers, networks, volumes

(i) Images
  * read-only template
  * instructions for creating container
  * eg: image for a flask app = original image of Python with necessary dependencies
  * Dockerfile defines steps for creating image
    ↳ each instruction creates layer in image
    ↳ layer: set of files and file metadata
  * Docker registry stores Docker images

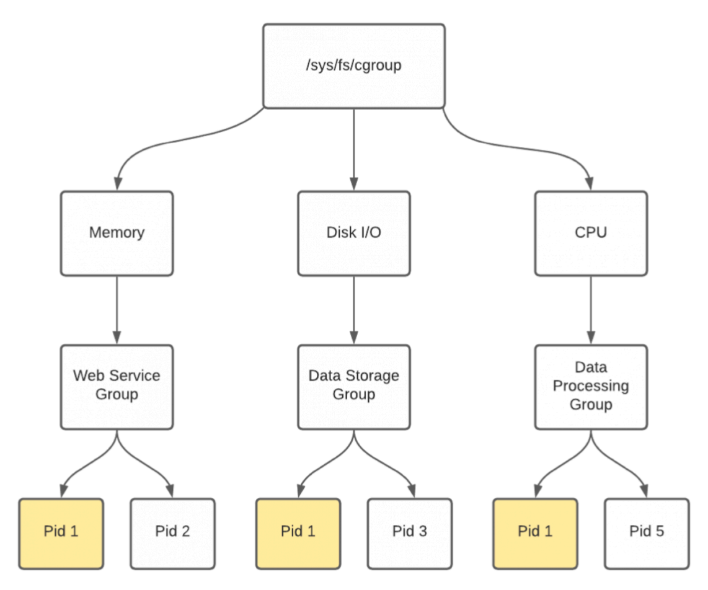
(ii) Containers
  * ready apps created from image
  * Docker creates set of namespaces

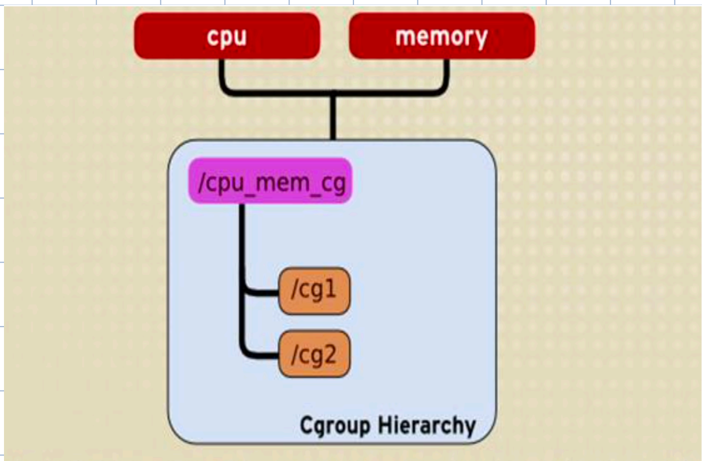
Namespaces

- All resources that a process sees - namespace
- Container access: namespace with subset of files of physical machine
- Docker namespace features
  * PID - create NS: create process with that NS
  * UTS - isolate hostnames
  * MNT
  * IPC
  * NET - each process within NS has access to net devices etc
  * USR
  * chroot - file sys root - each process has its own FS
  * CAP drop
  * cgroups - account resource usage for procs
    ↳ access (devices)

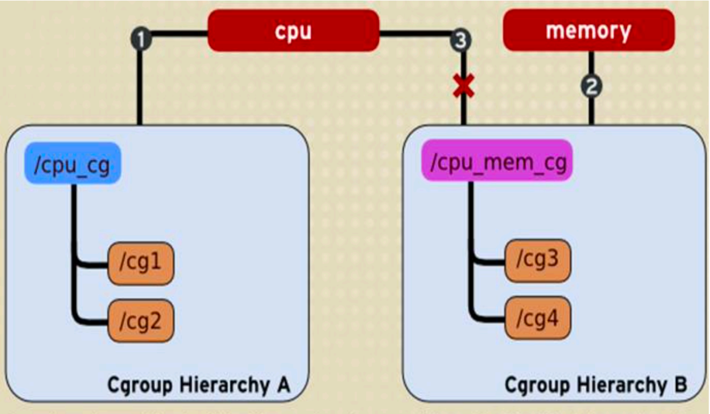
↳ resource limits
↳ prioritization
↳ accounting
↳ control
↳ injection

↳ tasks assigned to cgroups
↳ hierarchy for resources





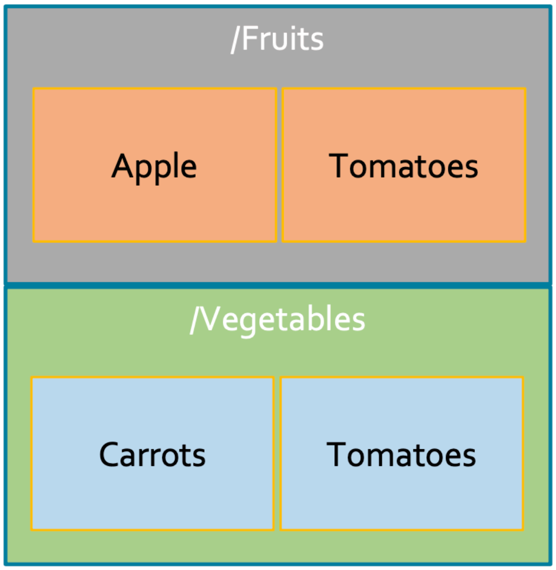
A single hierarchy can have one or more subsystems attached to it.



A subsystem attached to hierarchy A cannot be attached to hierarchy B if hierarchy B has a different subsystem already attached to it.

Union Filesystem

- Illusion of merging contents of several dirs into one
- Layering of FS's



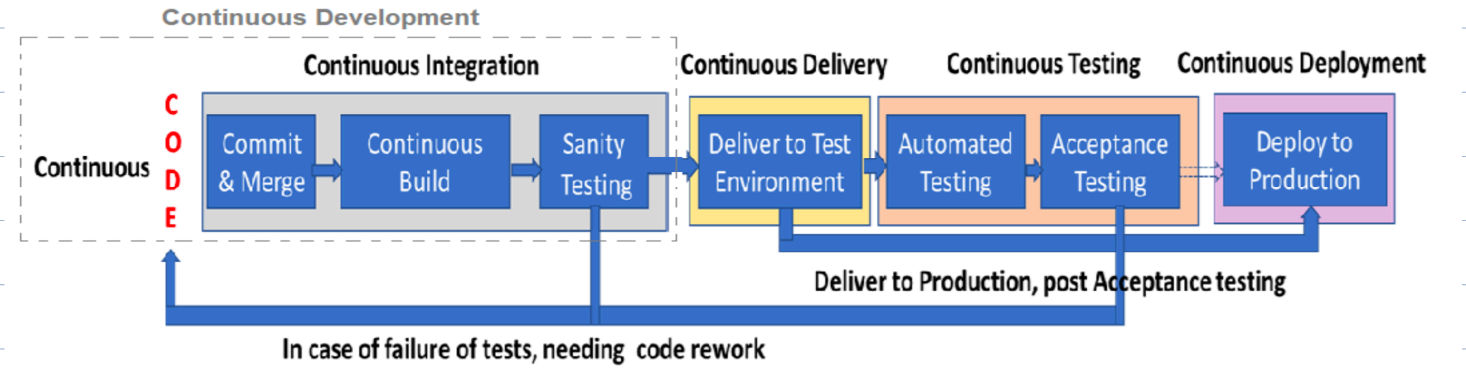
```
mount –t unionfs –o dirs=/Fruits:/Vegetables none /mnt/healthy
```

/mnt/healthy has 3 files – *Apple, Tomato, Carrots*

Tomato comes from */Fruits* (1st in *dirs* option)

DevOps

- Shorten SDLC
- Principles of continuous delivery
  - * Every build potential release
  - * Eliminate bottlenecks
  - * Automate wherever possible
  - * Trustworthy automated tests
- Small changes



Container Orchestration

- Automates scaling, management, deployment
- Immutable infrastructure

Kubernetes

- Objects

(i) Basic building objects

* Pod: group of containers
* Service: logical set of pods, policy, endpoint
* Volume: persistent data
* Namespace: segment of cluster

(ii) Controllers

* ReplicaSet (RS): guarantee availability of specified no. of identical pods
* Deployment: updates for pods in RS
* StatefulSet
* DaemonSet

* Job: creates pods, runs task, deletes pods

K8s Architecture

- Cluster contains
  * Master node
  * Worker nodes