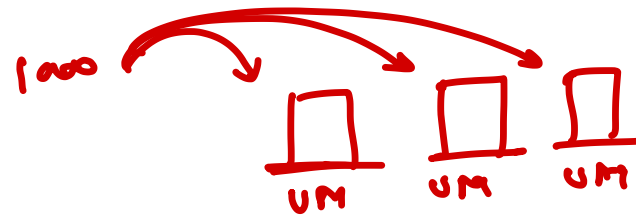
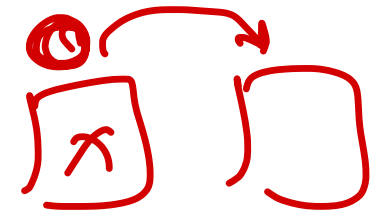


Container Orchestration

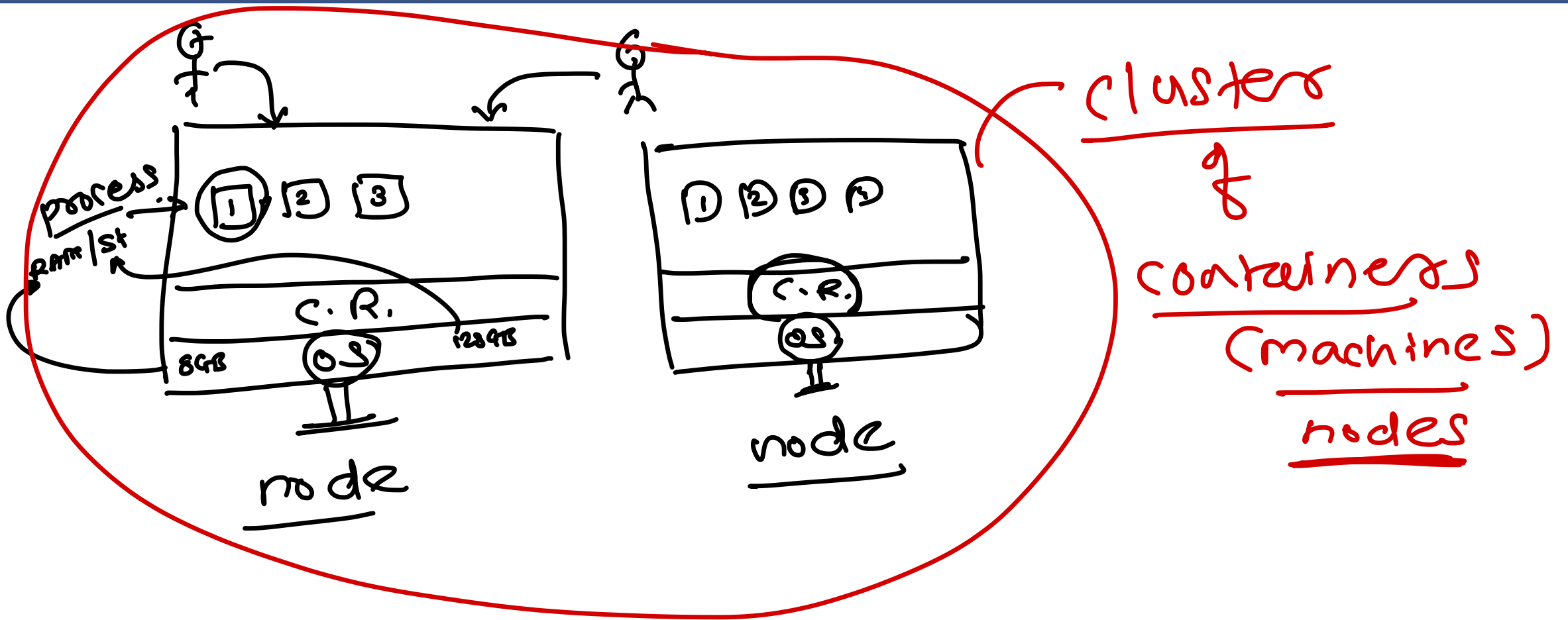


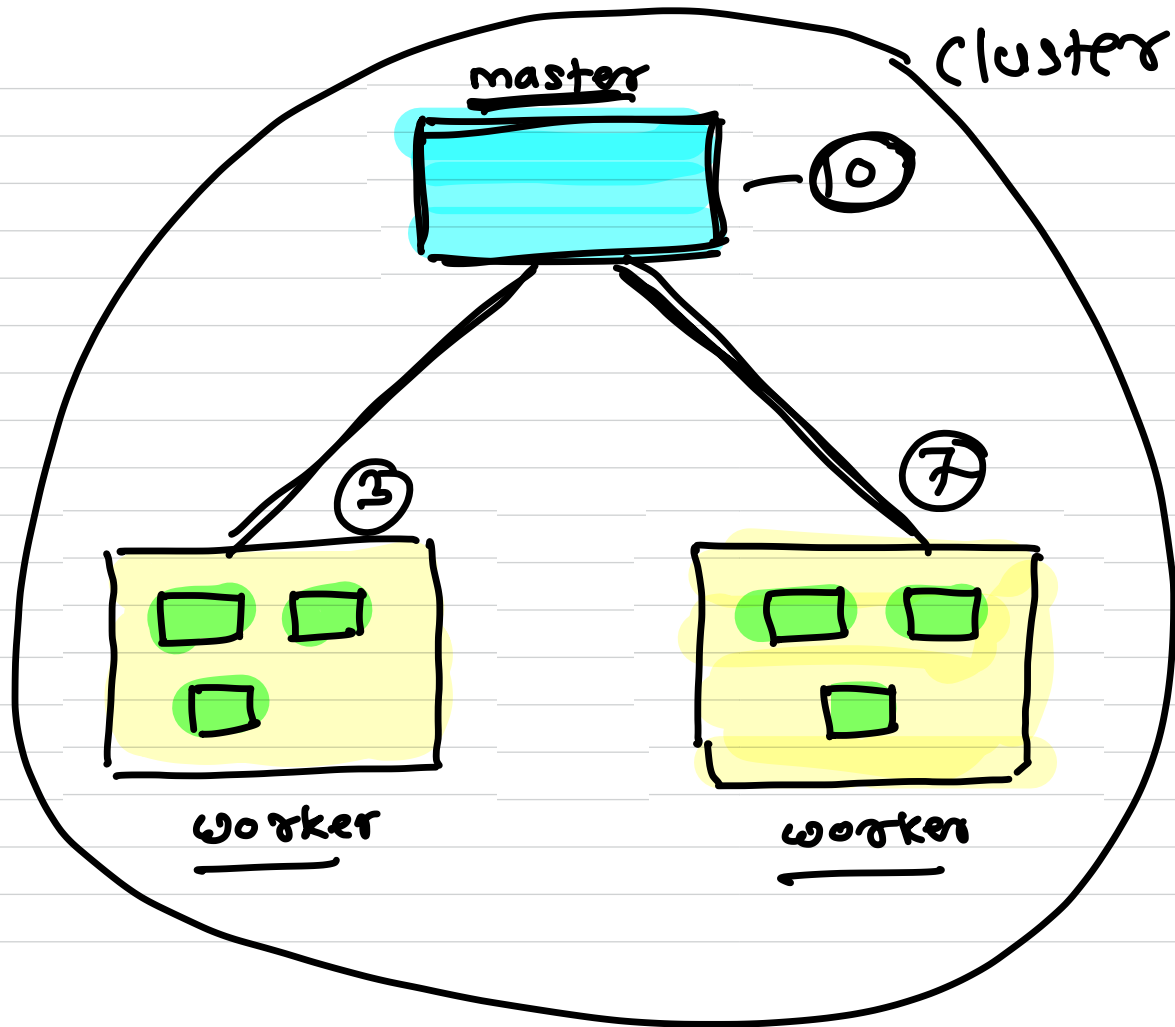
Overview

- Container orchestration is all about managing the lifecycles of containers, especially in large, dynamic environments
- Software teams use container orchestration to control and automate many tasks
 - Provisioning and deployment of containers
 - Redundancy and availability of containers
 - Scaling up or removing containers to spread application load evenly across host infrastructure
 - Movement of containers from one host to another if there is a shortage of resources in a host, or if a host dies
 - Allocation of resources between containers
 - External exposure of services running in a container with the outside world
 - Load balancing of service discovery between containers
 - Health monitoring of containers and hosts
 - Configuration of an application in relation to the containers running it



How does container orchestration work ?





Orchestration Tools

- Docker Swarm → 100,000
- Kubernetes → 1m
- Mesos
- Marathon



Docker Swarm



Overview

- Docker Swarm is a container orchestration engine
 - It takes multiple Docker Engines running on different hosts and lets you use them together
 - The usage is simple: declare your applications as stacks of services, and let Docker handle the rest
 - Services can be anything from application instances to databases
- service





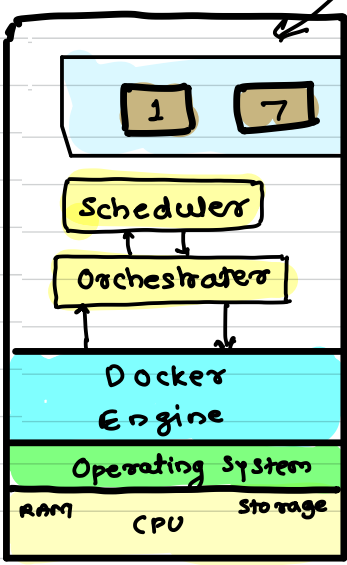
Docker Swarm [cluster]

> docker service create --replicas 10 myapp

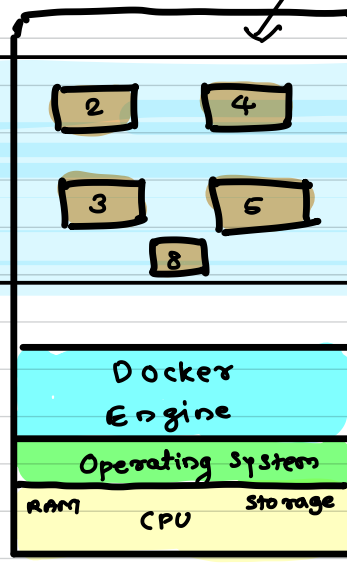
manager / worker

Worker 1

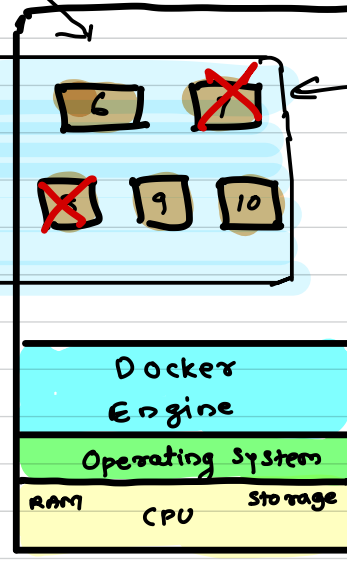
Worker 2



17.4.5.6



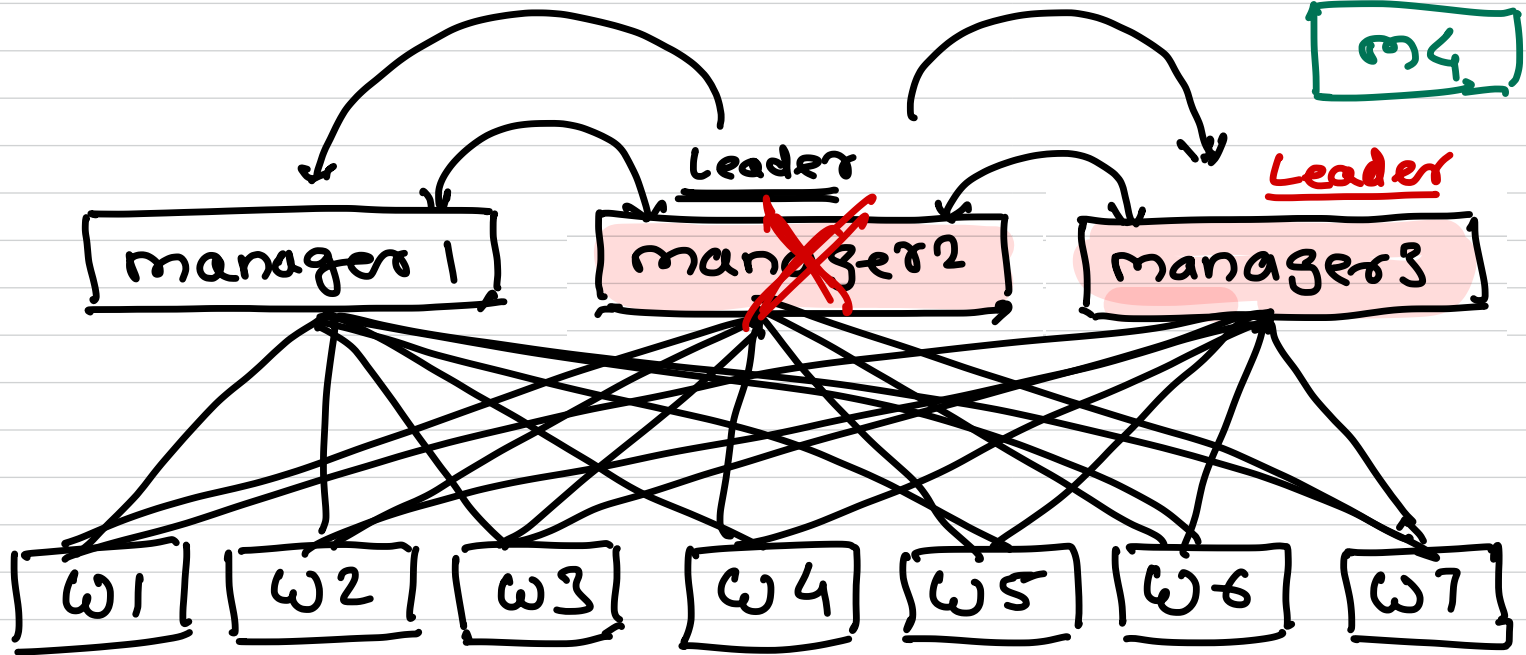
17.4.5.7



17.4.5.8

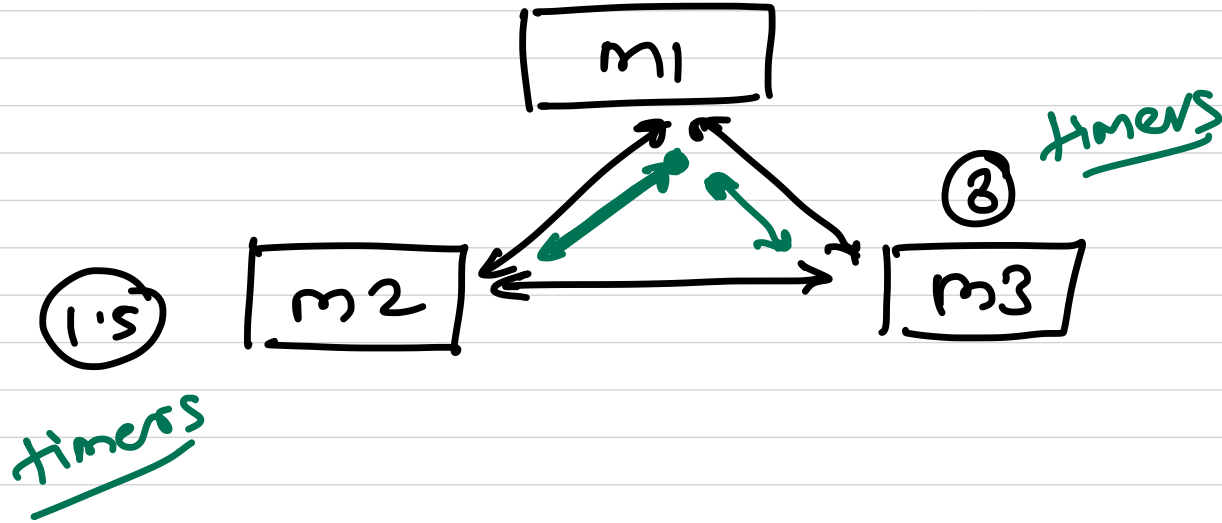
Service

* Raft



mesh

timers ① ✓ Leader



What is a swarm?

- A swarm consists of multiple Docker hosts which run in **swarm mode**
- A given Docker host can be a manager, a worker, or perform both roles
- When you create a service, you define its optimal state
- Docker works to maintain that desired state
 - For instance, if a worker node becomes unavailable, Docker schedules that node's tasks on other nodes
- A *task* is a running container which is part of a swarm service and managed by a swarm manager, as opposed to a standalone container
- When Docker is running in swarm mode, you can still run standalone containers on any of the Docker hosts participating in the swarm, as well as swarm services
- A key difference between standalone containers and swarm services is that only swarm managers can manage a swarm, while standalone containers can be started on any daemon



Features

- Cluster management integrated with Docker Engine
- Decentralized design
- Declarative service model
- Scaling
- Desired state reconciliation
- Multi-host networking
- Service discovery
- Load balancing
- Secure by default
- Rolling updates



Nodes

- A **node** is an instance of the Docker engine participating in the swarm
- You can run one or more nodes on a single physical computer or cloud server
- To deploy your application to a swarm, you submit a service definition to a **manager node**
- **Manager Node**
 - The manager node dispatches units of work called tasks to worker nodes
 - Manager nodes also perform the orchestration and cluster management functions required to maintain the desired state of the swarm
 - Manager nodes elect a single leader to conduct orchestration tasks
- **Worker nodes**
 - Worker nodes receive and execute tasks dispatched from manager nodes
 - An agent runs on each worker node and reports on the tasks assigned to it
 - The worker node notifies the manager node of the current state of its assigned tasks so that the manager can maintain the desired state of each worker



Services and tasks

- **Service**

- A service is the definition of the tasks to execute on the manager or worker nodes
- It is the central structure of the swarm system and the primary root of user interaction with the swarm
- When you create a service, you specify which container image to use and which commands to execute inside running containers

- **Task**

- A task carries a Docker container and the commands to run inside the container
- It is the atomic scheduling unit of swarm
- Manager nodes assign tasks to worker nodes according to the number of replicas set in the service scale
- Once a task is assigned to a node, it cannot move to another node
- It can only run on the assigned node or fail



Swarm Setup

- **Create swarm**

- > `docker swarm init --advertise-addr <MANAGER-IP>`

- **Get current status of swarm**

- > `docker info`

- **Get the list of nodes**

- > `docker node ls`



Swarm Setup

- **Get token (on manager node)**
 - > docker swarm join-token worker
- **Add node (on worker node)**
 - > docker swarm join --token <token>



Swarm Service

- **Deploy a service**

> docker service create --replicas <no> --name <name> -p <ports> <image> <command>

- **Get running services**

> docker service ls

- **Inspect service**

> docker service inspect <service>

- **Get the nodes running service**

> docker service ps <service>



Swarm Service

- **Scale service**

- > docker service scale <service>=<scale>

- **Update service**

- > docker service update --image <image> <service>

- **Delete service**

- > docker service rm <service>

