[544] Cloud Deployment

Tyler Caraza-Harter

Outline

Docker and Beyond

- Containers
- Images

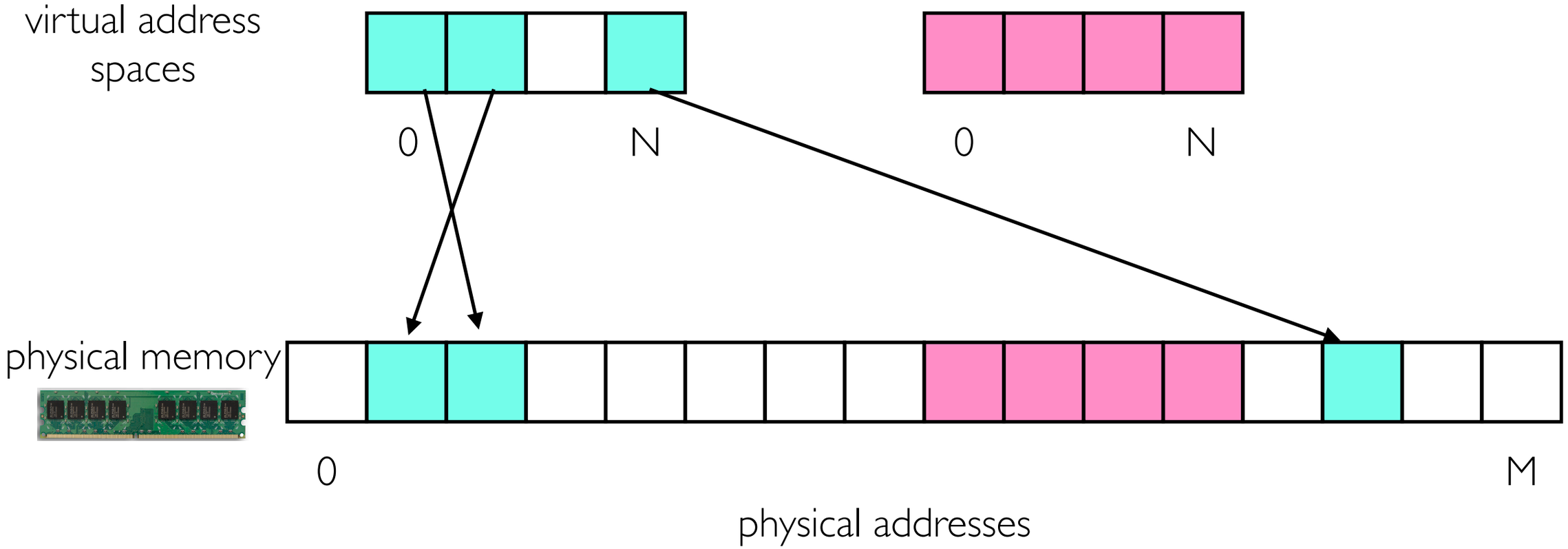
Cloud Services:

- Artifact Registry
- COS (Container-Optimized Operating System)
- GKE (Google Kubernetes Engine)

Review: Processes and Address Spaces

Address spaces

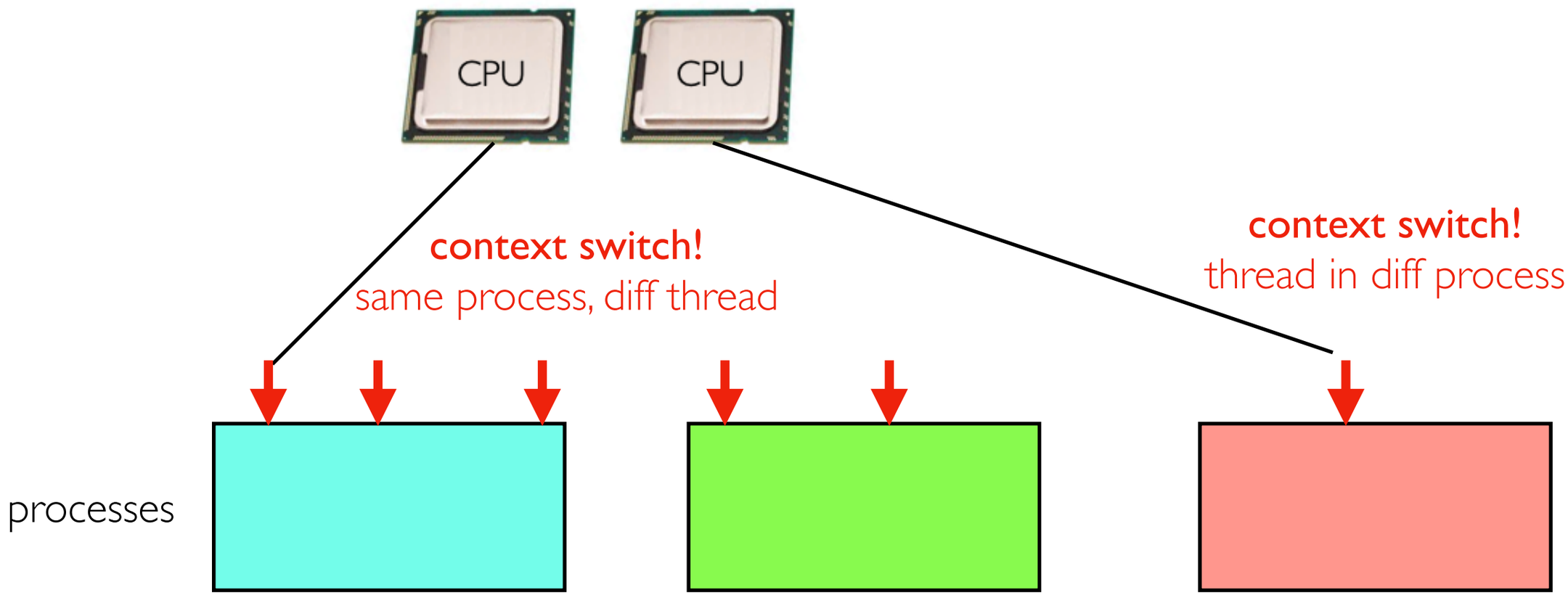
- A process is a running program
- Each process has it's own virtual address space
- The same virtual address generally refers to different memory in different processes
- Regular processes cannot directly access physical memory or other addr spaces
- Address spaces can have holes (N is usually MUCH bigger than M)
- Physical memory for a process need not be contiguous



Review: Context Switch

Schedulers

- CPU scheduler is an important sub system in an operating system
- schedulers decide when to context switch between threads
- context swich: change which thread a CPU is running



Isolation

We don't want different applications running on the same hardware to **interefere** with each other -- we want them to be isolated. Concerns:

- malicious programs, buggy programs, fairness.

Ways to interfere

- **directly**: seeing/modifying data of another process
- **indirectly**: inflicting bad performance on another process

Some Operating System isolation features with a long history:

- **virtual memory**: can't see another process's data (namespace isolation)
- **schedulers**: can't hog the whole CPU (performance isolation)

problem: CPU and memory aren't the only resources

goal: both namespace AND performance isolation for EVERY kind of resource

"Newer" Linux Features: cgroups and Namespaces

cgroup types (performance isolation)

- **cpu, memory**, cpuacct, cpuset, freezer, net_cls, blkio, perf_event, net_prio, hugetlb, pids, rdma

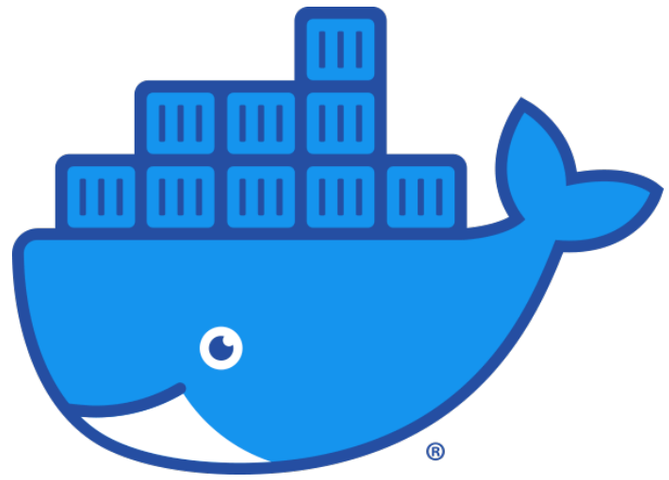
namespaces types (performance isolation)

- **network, mount**, time, user, cgroup, IPC, PID, UTS

Both cgroups and namespaces apply to sets of processes. Configuring all this by hand is VERY complicated.

One reason Docker is popular: "docker run ..." starts a process using all these features, each with reasonable configurations.

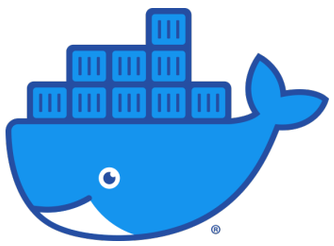
"Container" definition: set of processes using a combination of cgroup/namespace/other features.



Kubernetes (k8s)

8 letters

cgroups and namespaces are very flexible: Docker's approach is just ONE way to use them to build containers.



namespaces: mount, network, etc
cgroups: cpu, memory, etc

process 1

process 2

Docker "container"

"mount" is for file system

namespaces: mount, network, etc
cgroups: cpu, memory, etc

process 3

process 4

process 5

Docker "container"

namespaces: network, etc

namespaces: mount, etc
cgroups: cpu, etc

process 1

process 2

K8s "container"

K8s "pod"

namespaces: network, etc

namespaces: mount, etc
cgroups: cpu, etc

process 1

process 2

K8s "container"

namespaces: mount, etc
cgroups: cpu, etc

process 3

process 5

process 4

K8s "container"

K8s "pod"

Kubernetes (k8s)

8 letters

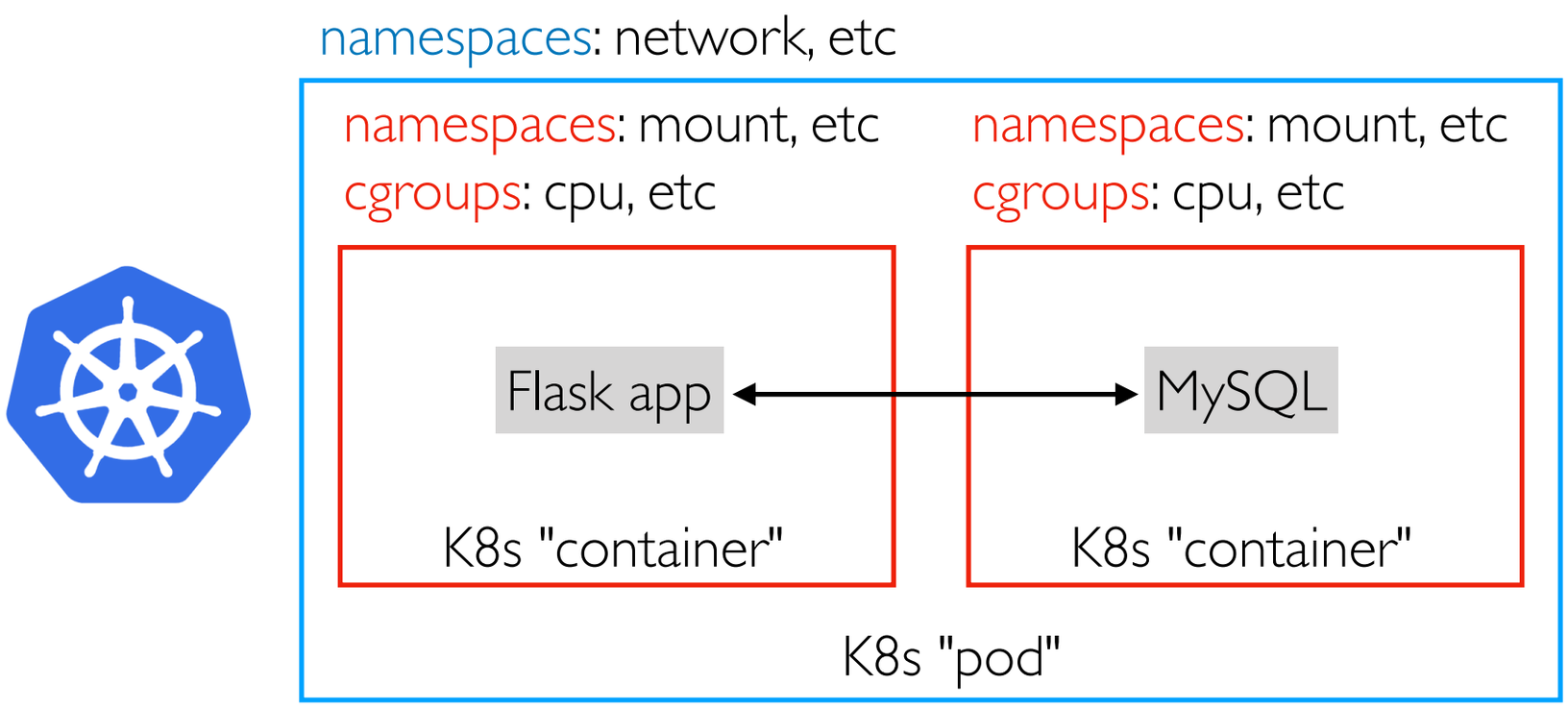
Motivation: we often want to deploy multiple applications that "work together"

shared between containers in same pod

- same VM, IP, port visibility

not shared

- CPU/memory resources (etc)
- files (great! each can have their own Linux distro, packages versions, etc)



Container Orchestration

The container orchestrator landscape, by Jordan Webb (https://lwn.net/Articles/905164/)

Kubernetes currently is the most popular *container orchestrator*. A container orchestrator can launch many collaborating containers in a cluster (of VMs or physical machines).

Other orchestrators:

- **Docker compose:** only launches containers on one node (so not necessarily an "orchestrator" depending on definition)
- **Docker swarm:** built from compose to support multiple nodes
- **Nomad:** simpler alternative to Kubernetes

Outline

Docker and Beyond

- Containers
- Images

Cloud Services:

- Artifact Registry
- COS (Container-Optimized Operating System)
- GKE (Google Kubernetes Engine)

Docker Images

The other reason Docker is very popular is that Dockerfiles can be used to directly build images. This is a BIG improvement over README files.

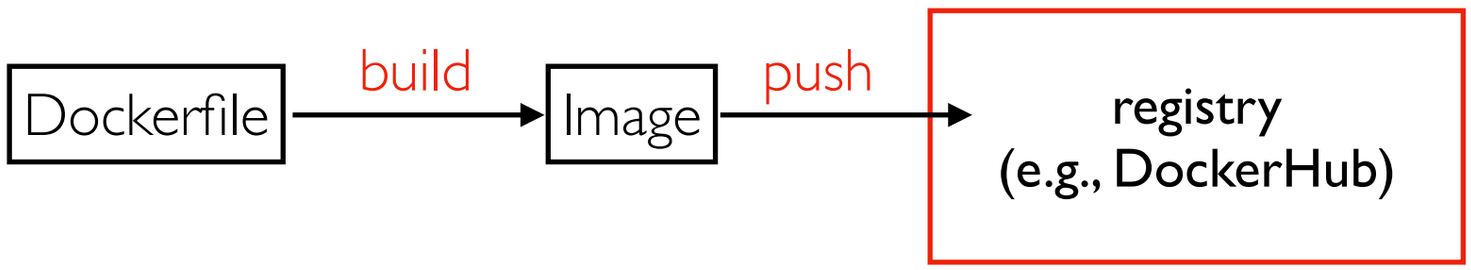
README.md

```
# Setup

This software runs on Ubuntu.  First, install Java 8 and Python 3.
Then ...
```

Dockerfile

```
FROM ubuntu:22.04
RUN apt-get update; apt-get install -y wget curl openjdk-8-jdk python3-
pip net-tools lsof nano unzip
RUN pip3 install jupyterlab==3.4.5 MarkupSafe==2.0.1 cassandra-driver
pyspark==3.2.2 pandas matplotlib kafka-python grpcio-tools
...
```



Open Container Initiative

https://opencontainers.org/

Goal: define standardized formats for images/containers
(alternative to original Docker formats)

image from OCI site

Outline

Docker and Beyond

- Containers
- Images

Cloud Services:

- Artifact Registry
- COS (Container-Optimized Operating System)
- GKE (Google Kubernetes Engine)

Common Cloud Services

Image building (from Dockerfiles)

Imagine registries (public/private alternative to DockerHub)

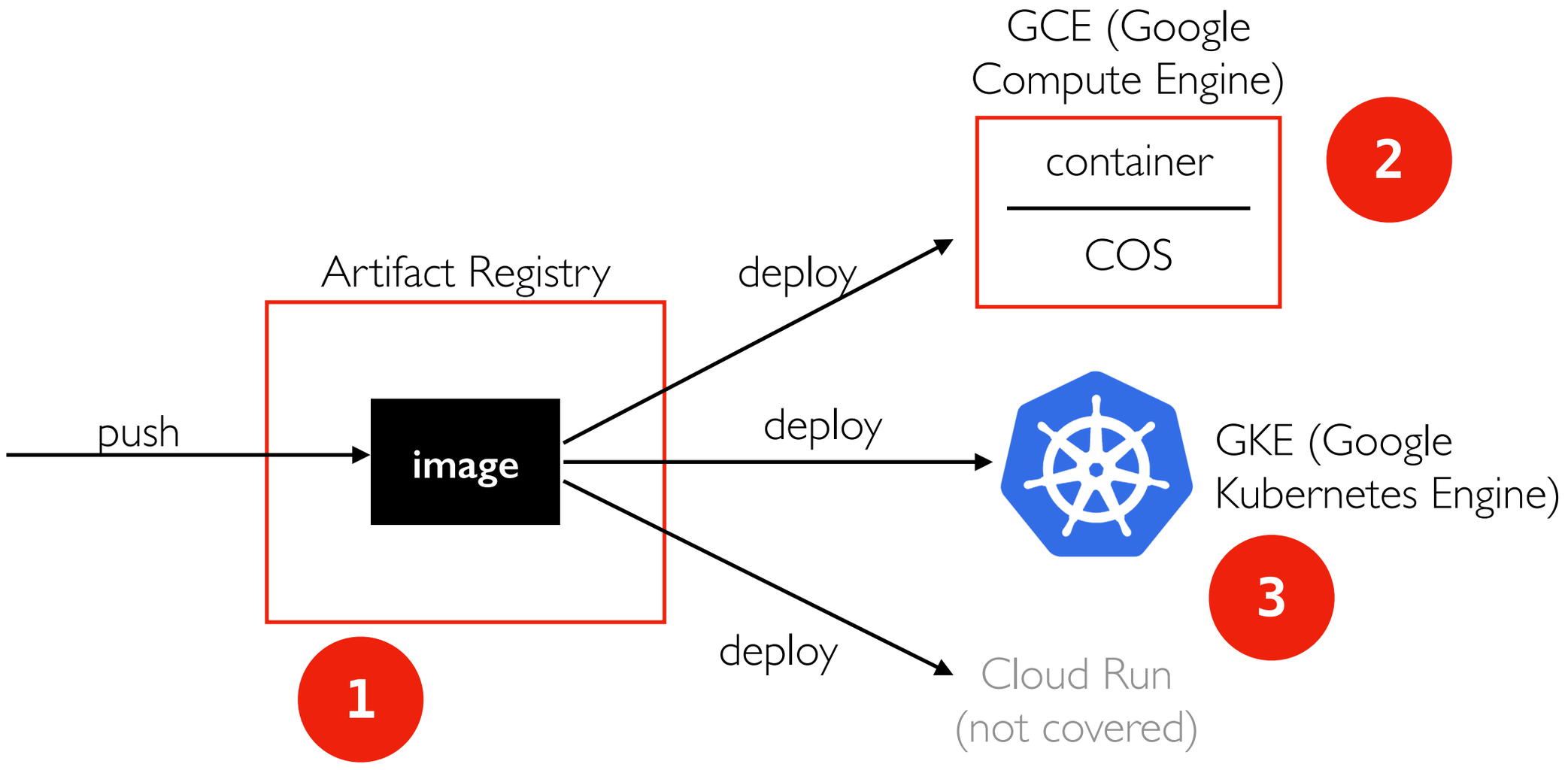
Deployment options (from OCI and/or Docker images)

- managed Kubernetes deployment
- containers on operating systems customized for containers
- various serverless platforms (that run code in a container when an event occurs)
- even directly to a VM!

Observation 1: using Dockerfiles to specify requirements is useful beyond containers (why not use it to specify what you want pre-installed on a VM?).

Observation 2: container images are compatible between Docker and Kubernetes even though they use cgroups/namespaces differently to isolate running containers.

Examples: Three Google Cloud Services



Outline

Docker and Beyond

- Containers
- Images

Cloud Services:

- Artifact Registry
- COS (Container-Optimized Operating System)
- GKE (Google Kubernetes Engine)

Using Artifact Registry

Artifact Registry come in different varieties for different types of resources

- Docker images, apt packages, Python packages, etc

"docker login" can be used to authenticate to a registry

- Docker Hub lets you type a password
- For GCP, we'll use an "access token"

Getting an access token corresponding to your whole user is overly broad -- create a service account (basically a non-human user) with limited roles, and use the access token from that. Security principle of "least priviledge".

To get the access token, we'll prove we can act as the service account with a "key file", similar to a private SSH key.

Docs: https://cloud.google.com/artifact-registry/docs/docker/authentication

Demos...

Outline

Docker and Beyond

- Containers
- Images

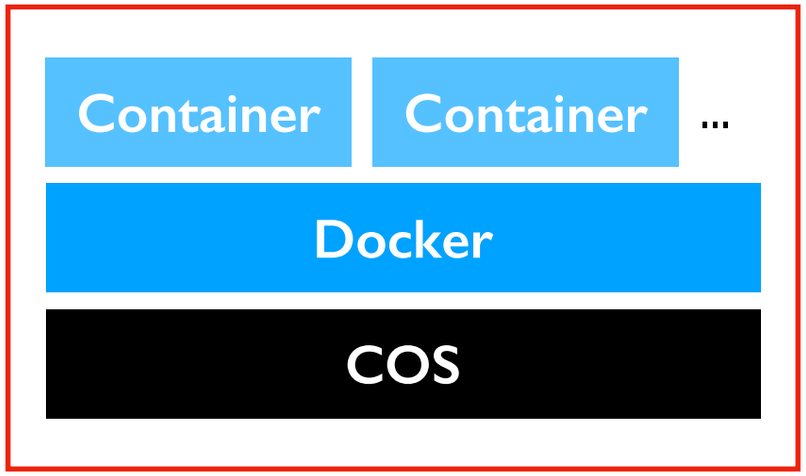
Cloud Services:

- Artifact Registry
- COS (Container-Optimized Operating System)
- GKE (Google Kubernetes Engine)

Container Optimized Operating System (COS)

Container-Optimized OS

- based on Chrome OS
- Docker pre-installed
- Containers launch with VM launch, no extra steps
- locked down for security (if you just need to run containers, you don't need much)
  - ➡ read-only root file system
  - ➡ cannot install packages



VM

Demos...

Outline

Docker and Beyond

- Containers
- Images

Cloud Services:

- Artifact Registry
- COS (Container-Optimized Operating System)
- GKE (Google Kubernetes Engine)

Google Kubernetes Engine (GKE)

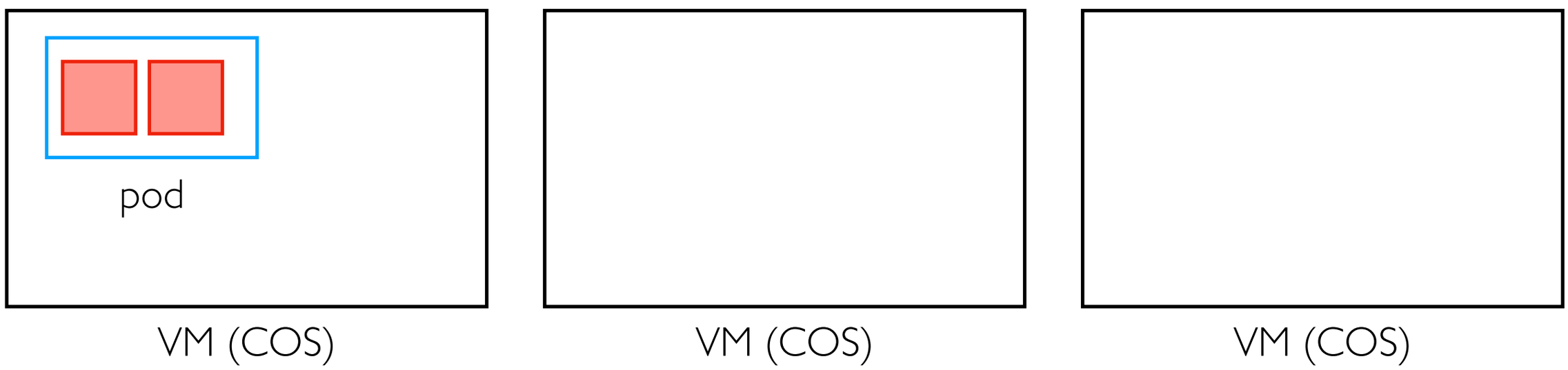
Google also open-sourced Kubernetes

Billing options (both have cluster management fee of ~$70/month, but free tier covers one cluster)

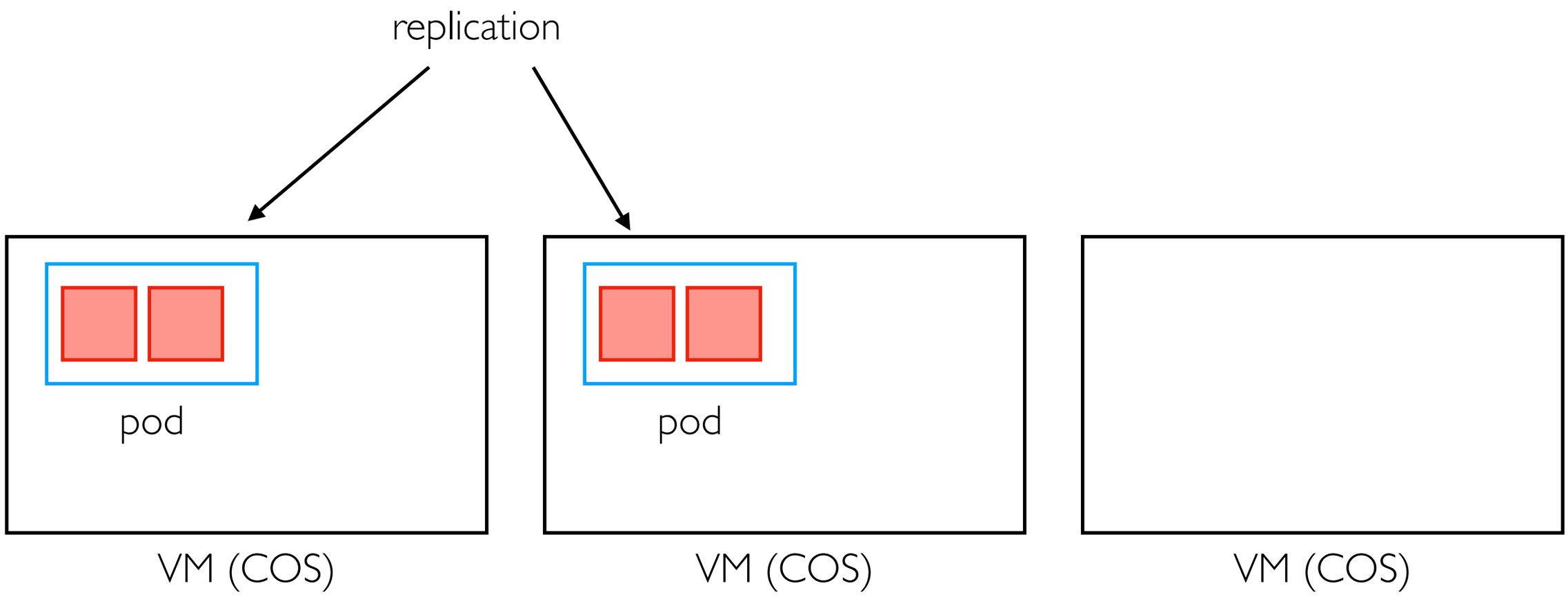
- pay for the VMs that run the containers (standard)
- pay for containers (autopilot)

Cluster workers:

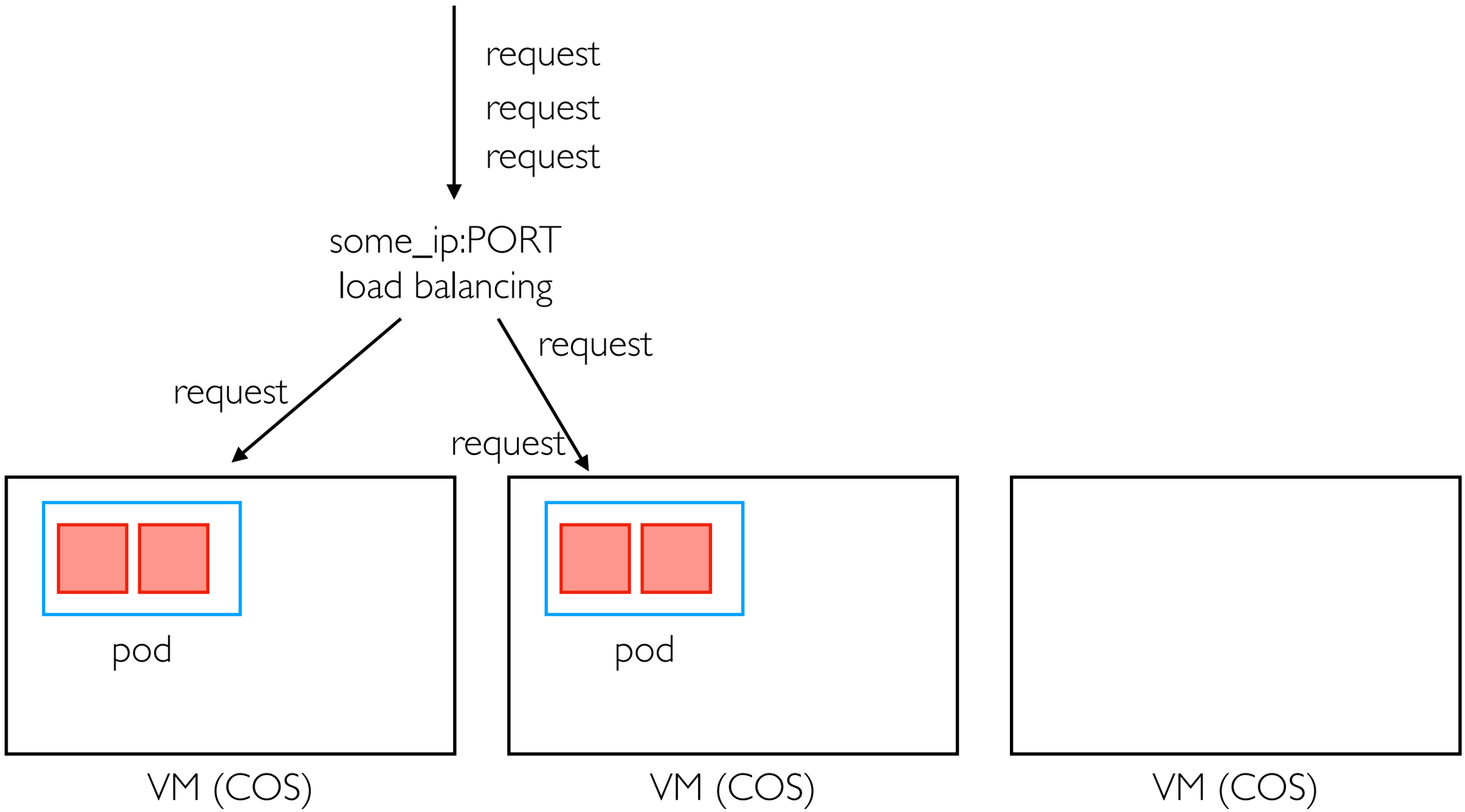
- many VMs running COS and managed by Kubernetes



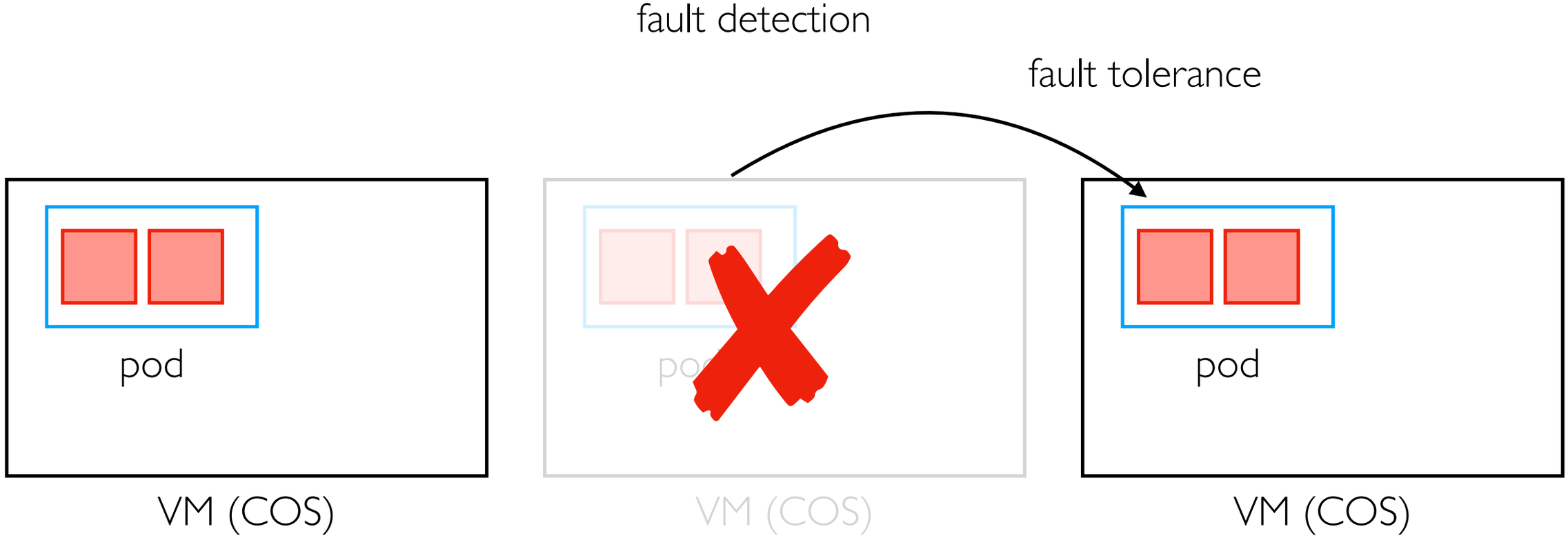
Kubernetes: Features



Kubernetes: Features



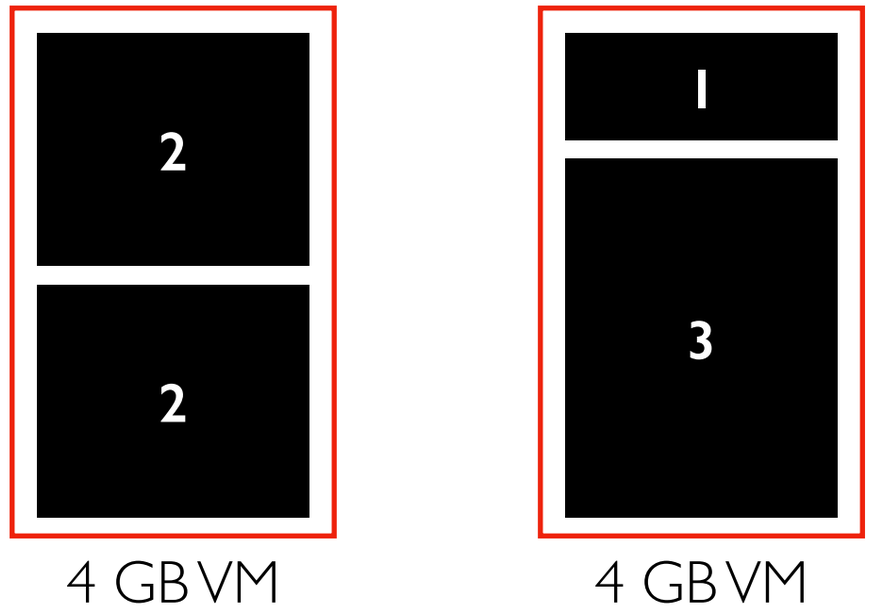
Kubernetes: Features



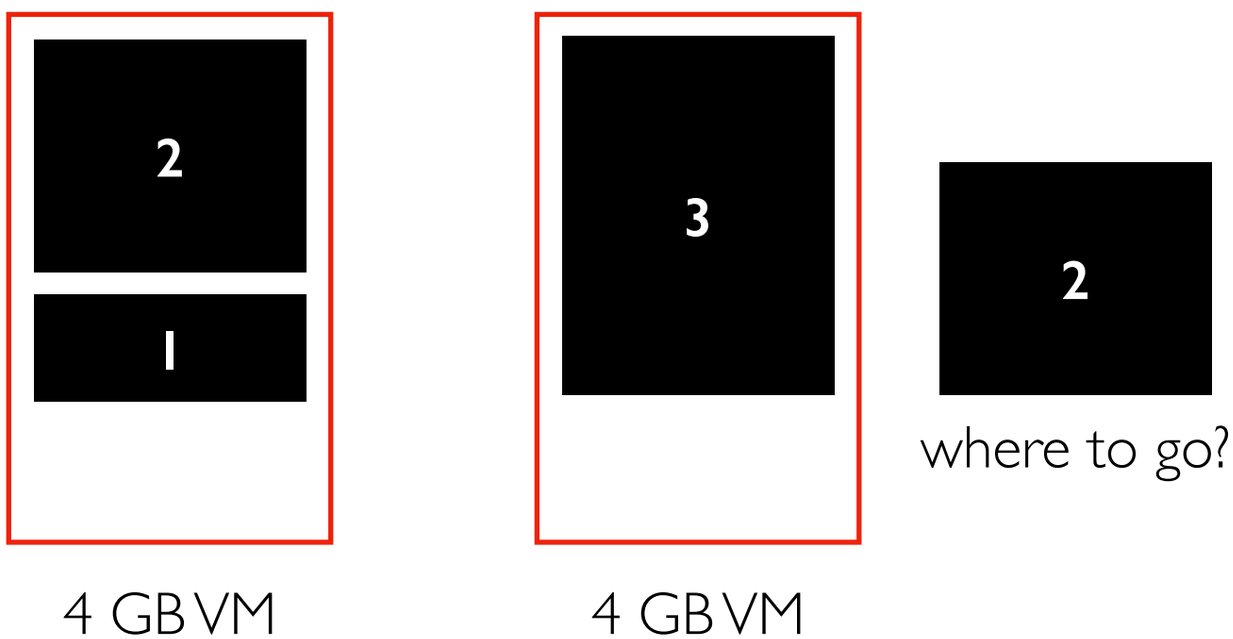
Kubernetes Bin Packing

Kubernetes will automatically try to make good "bin packing" decisions about where to place containers

good bin packing



bad bin packing



Demos