From Dev to Prod

Robert Clements

MSDS Program

University of San Francisco

What to Expect

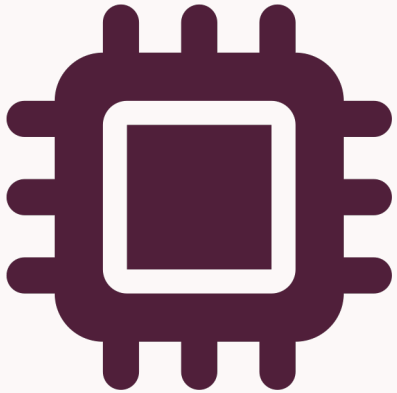
- Goal: to learn about moving from the dev environment to the prod environment using containers and orchestration.

- How: we will quickly learn about containerization, orchestration, and infrastructure-as-code at a high level and practice some tutorials. We will mention CI/CD, but save the bulk of this topic for a future lecture.

Compute

Compute Layer

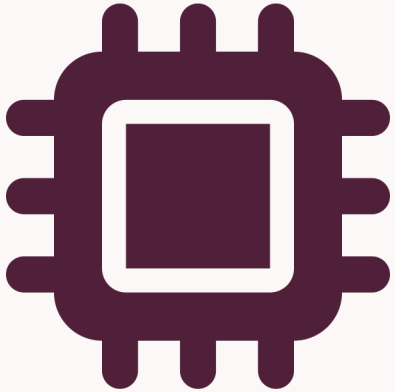
ML applications need two things: **data** and **computing resources**



Compute Layer

ML applications need two things: **data** and **computing resources**

Where do we execute our code?

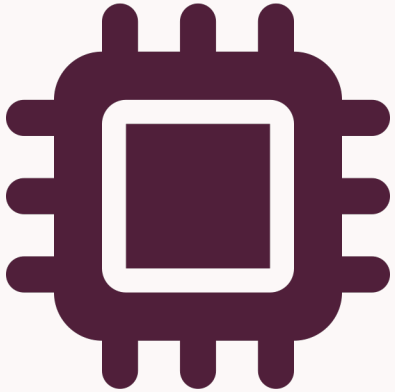


Compute Layer

ML applications need two things: **data** and **computing resources**

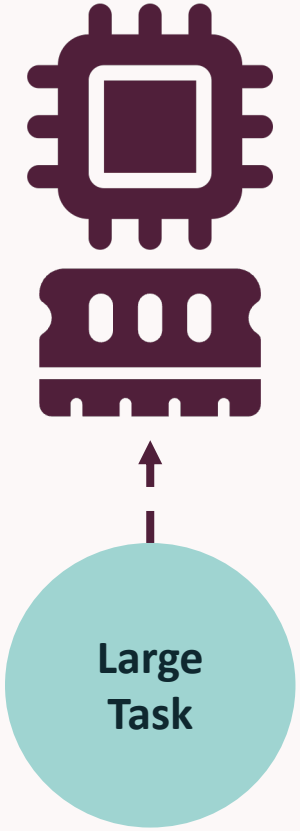
How do we make our code scalable?

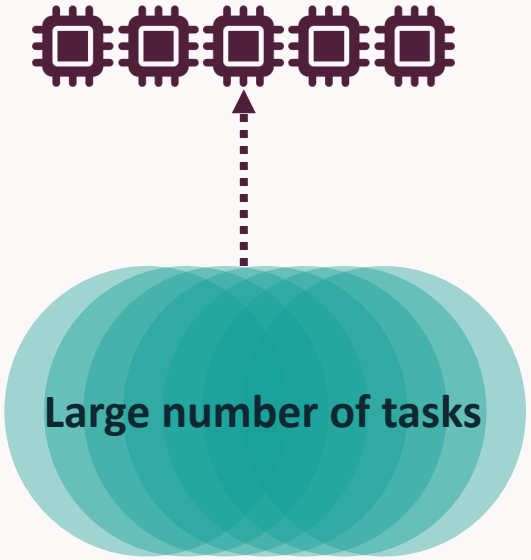
Do more computations
(CPUs/GPUs)



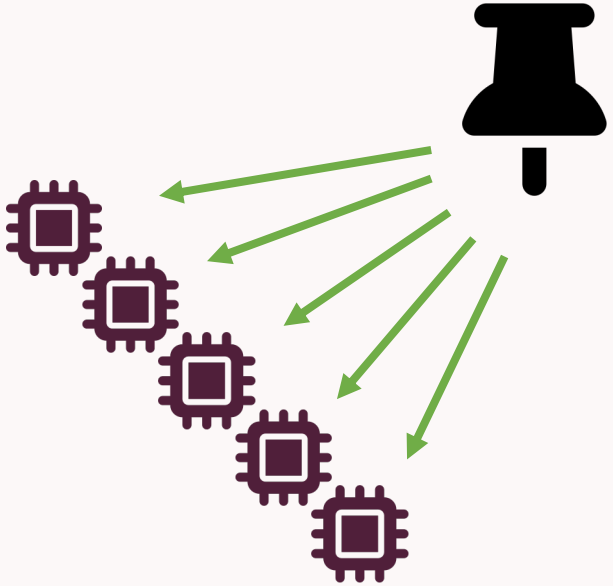
Handle more data
(memory)

Scaling Vertically and Horizontally

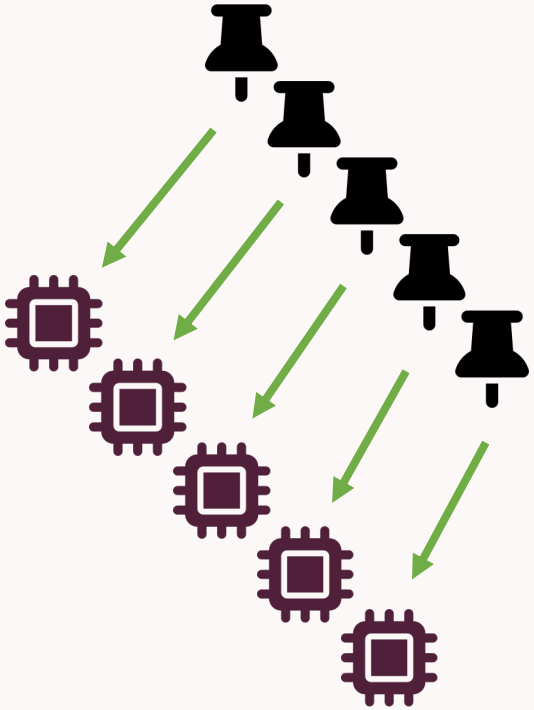




Scalable Systems

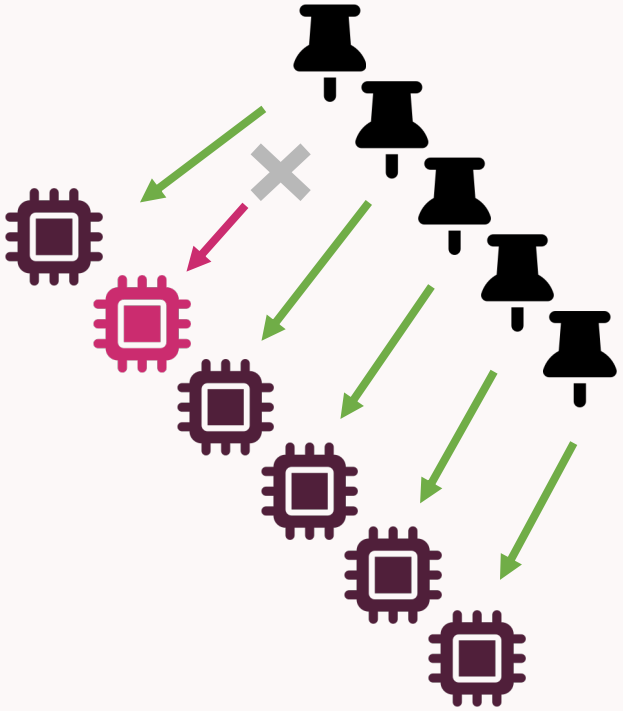


Match a task (packing) to resources



Match a large number of tasks to resources

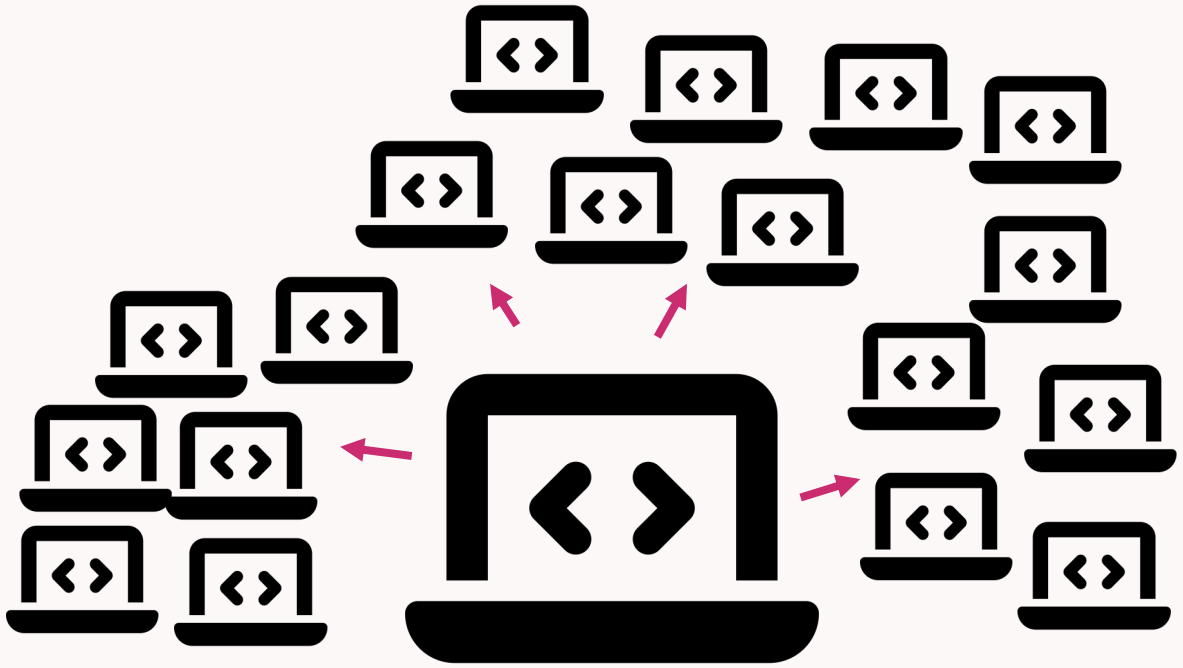
Keep the system running no matter what problems occur.



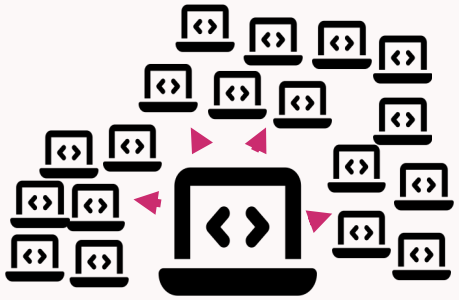
Containerization

Containers

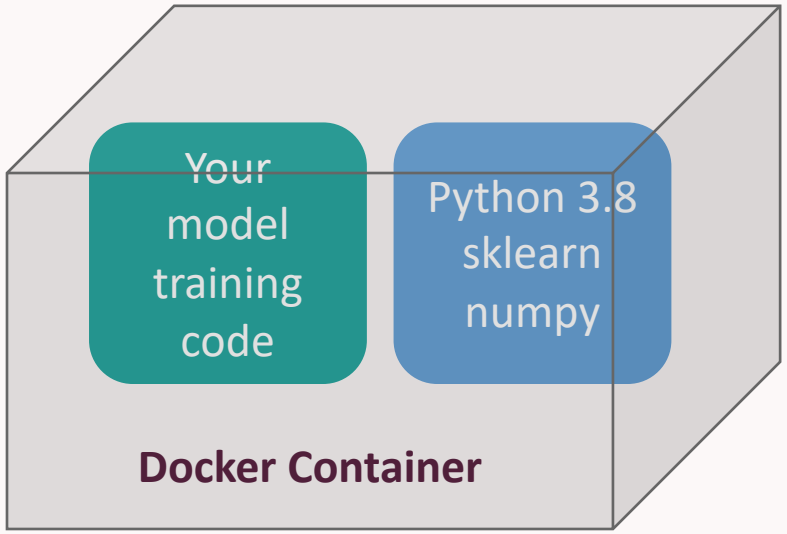
- Create environment once, replicate as many times as needed



Containers

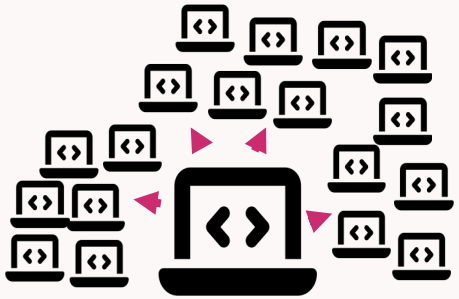


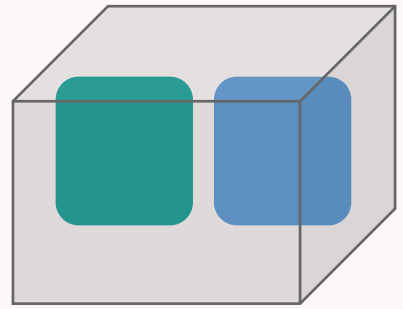
- Create environment once, replicate as many times as needed
- A form of batch processing, running a snippet of code, without affecting the rest of the system



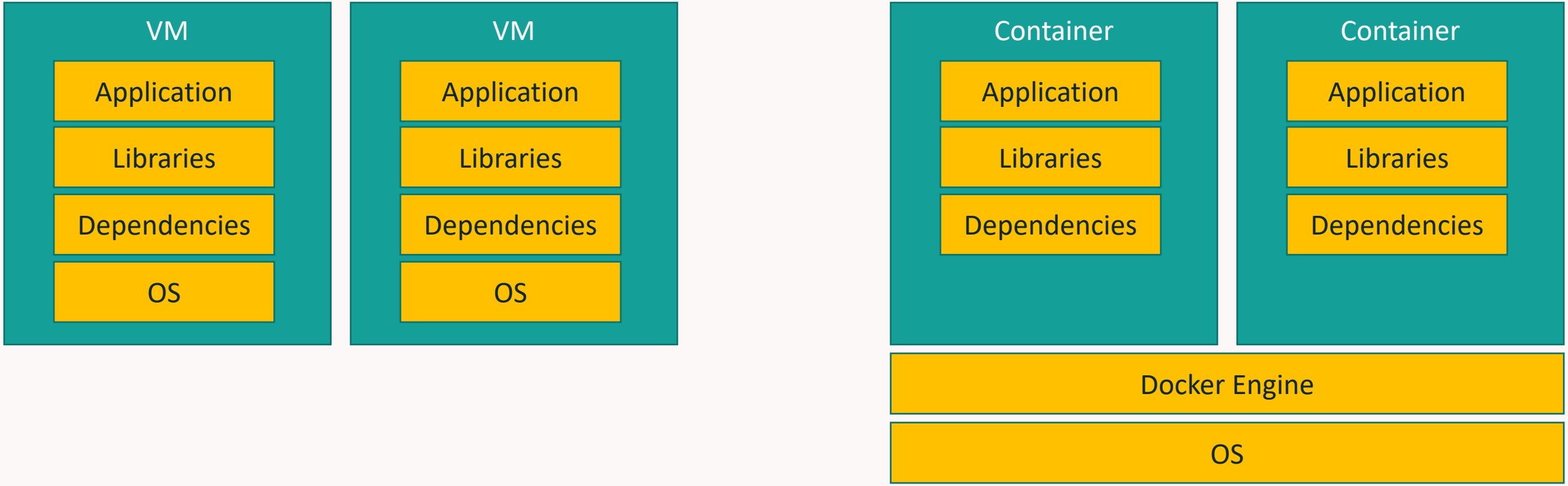
Containers

- Create environment once, replicate as many times as needed
- A form of batch processing, running a snippet of code, without affecting the rest of the system
- Not inherently scalable without a proper orchestration engine
  - If you run the code in a container on your laptop, it is essentially the same as running it on a supercomputer



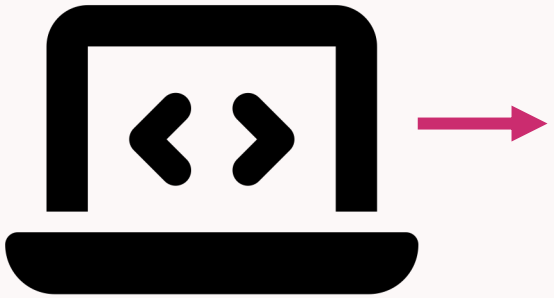


Containers vs VMs



Containers

- Dockerfile is the ***recipe*** to replicate your environment



Dockerfile

1. Install packages/libraries
2. Load objects/files
3. Set variables

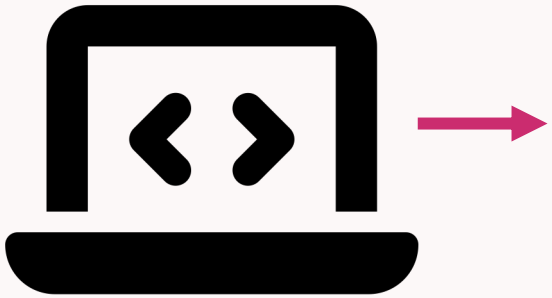
Always starts with FROM
Each line is a command

What ingredients does your application need in order to run?

1. OS
2. Libraries and dependencies
3. Python and python libraries
4. Source code

Containers

- Dockerfile is the ***recipe*** to replicate your environment



Dockerfile

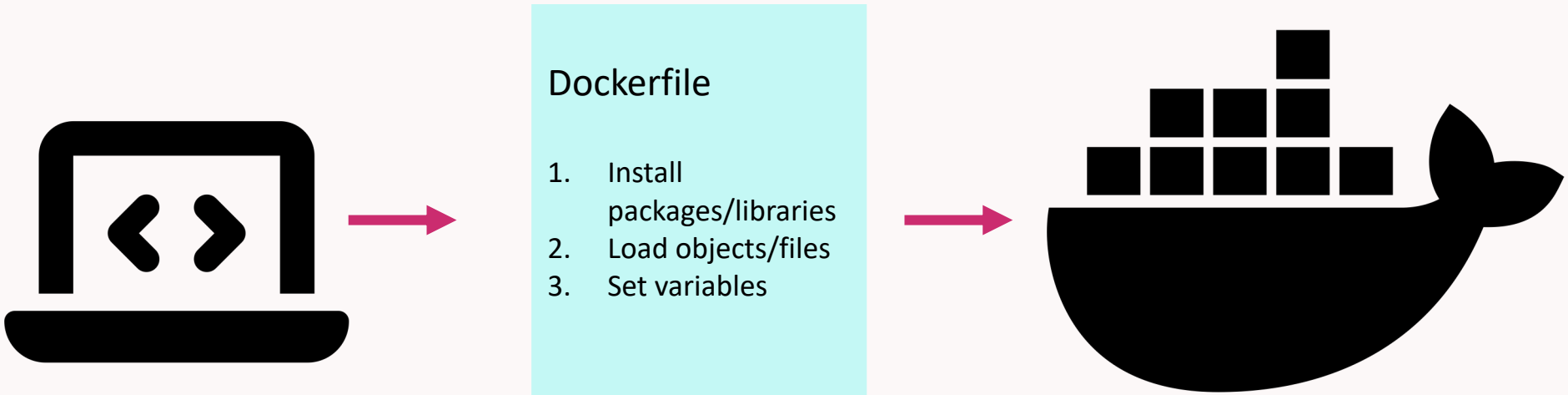
1. Install packages/libraries
2. Load objects/files
3. Set variables

Always starts with FROM
Each line is a command

```
FROM python:3.12-slim

WORKDIR /

COPY requirements.txt requirements.txt
COPY server.sh server.sh

RUN pip install --upgrade pip && pip install -r requirements.txt

EXPOSE 8080

RUN chmod +x server.sh

ENTRYPOINT ["./server.sh"]
```

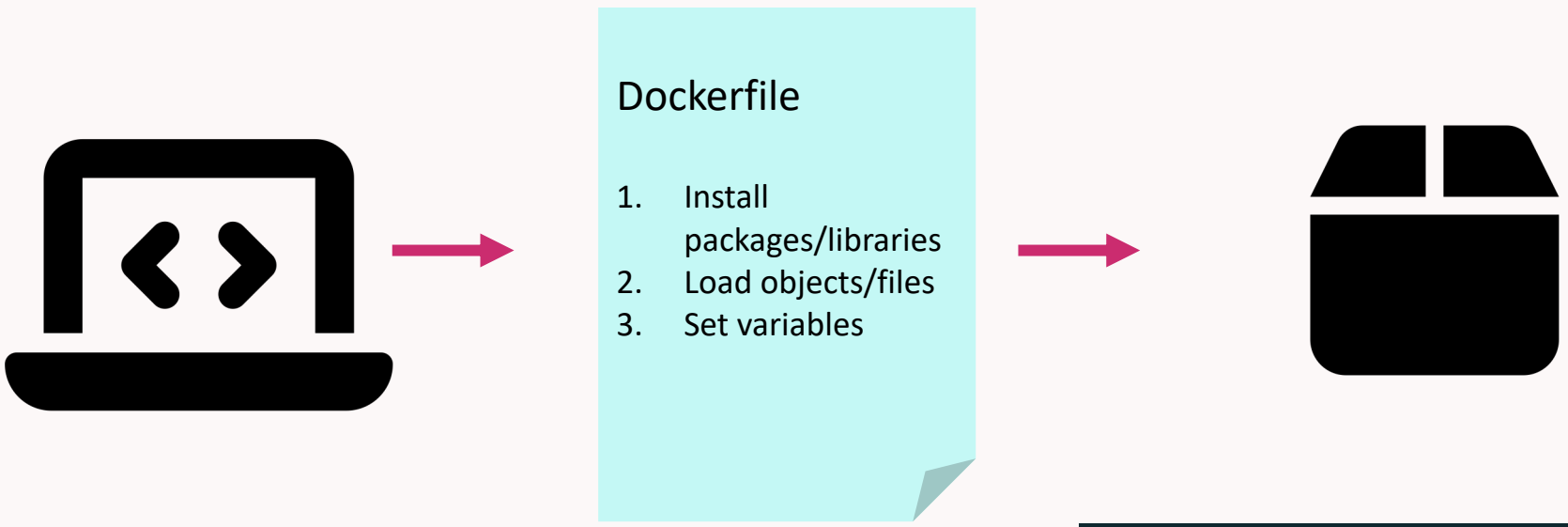
Containers

- Dockerfiles create ***images***, can be stored in a docker ***registry***



Containers

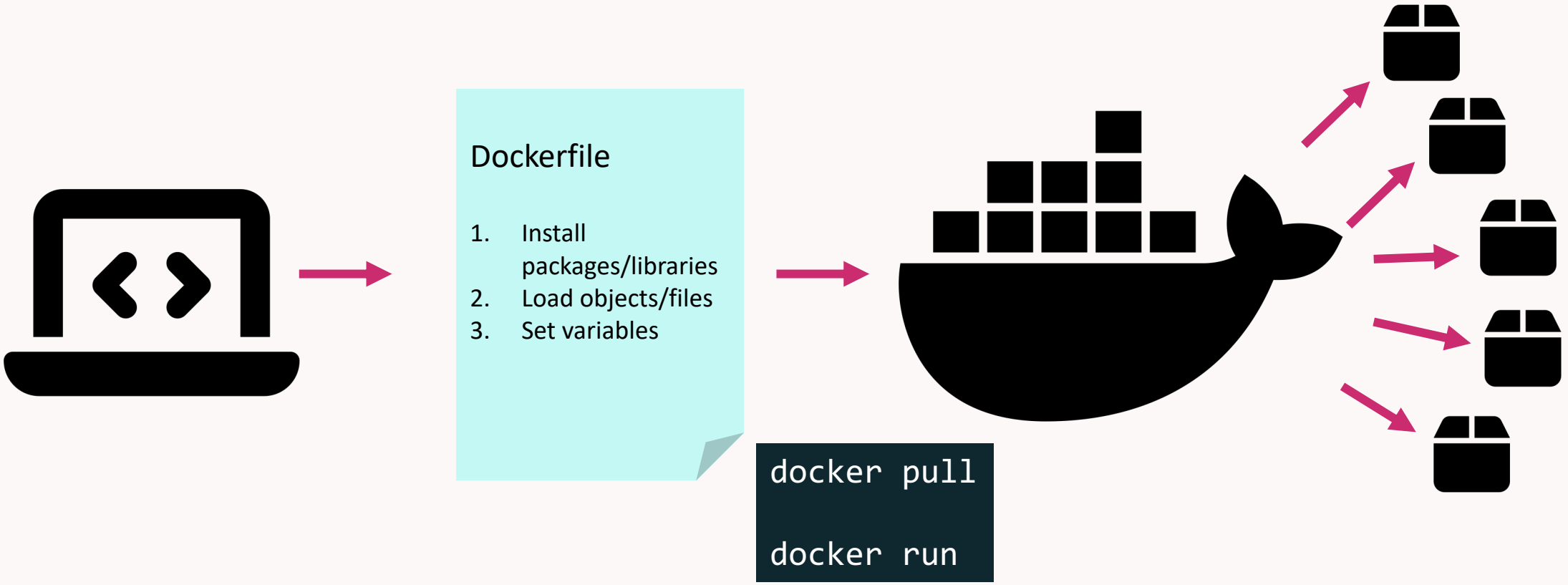
- ***Images*** are like packages, and are built in layers




```
docker history <image_name>
```

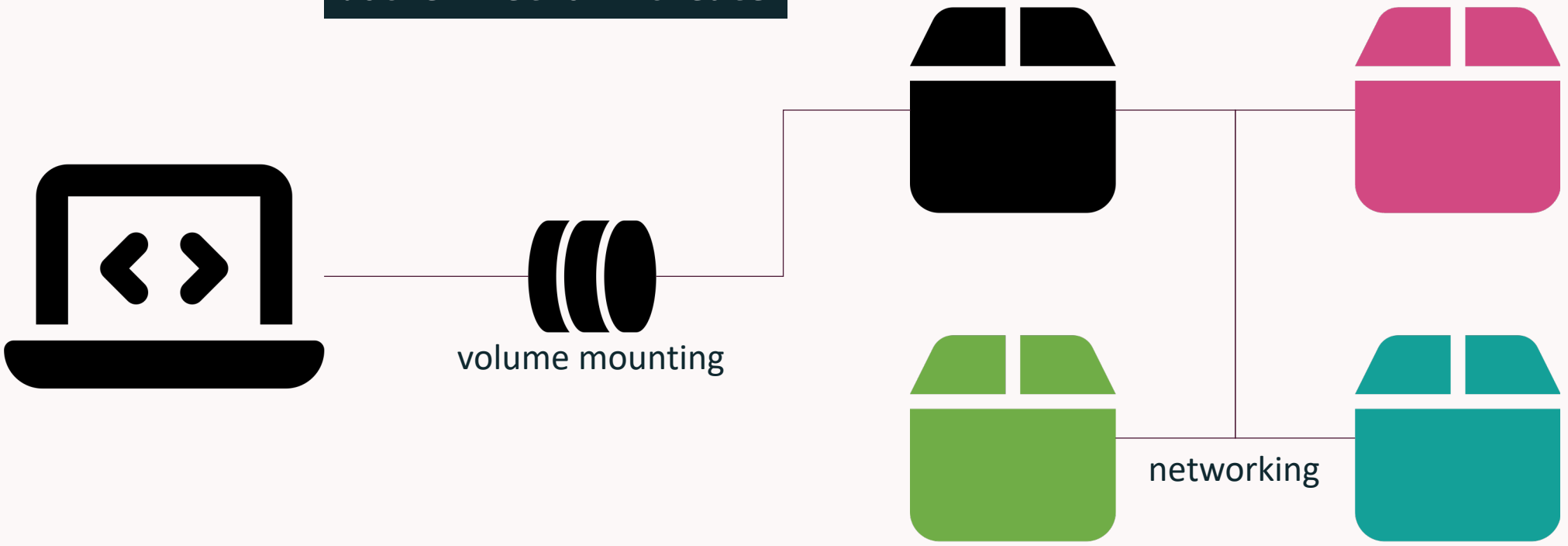
Containers

- An instance of the docker image is called a ***container***



Containers

```
docker volume create

docker network create
```



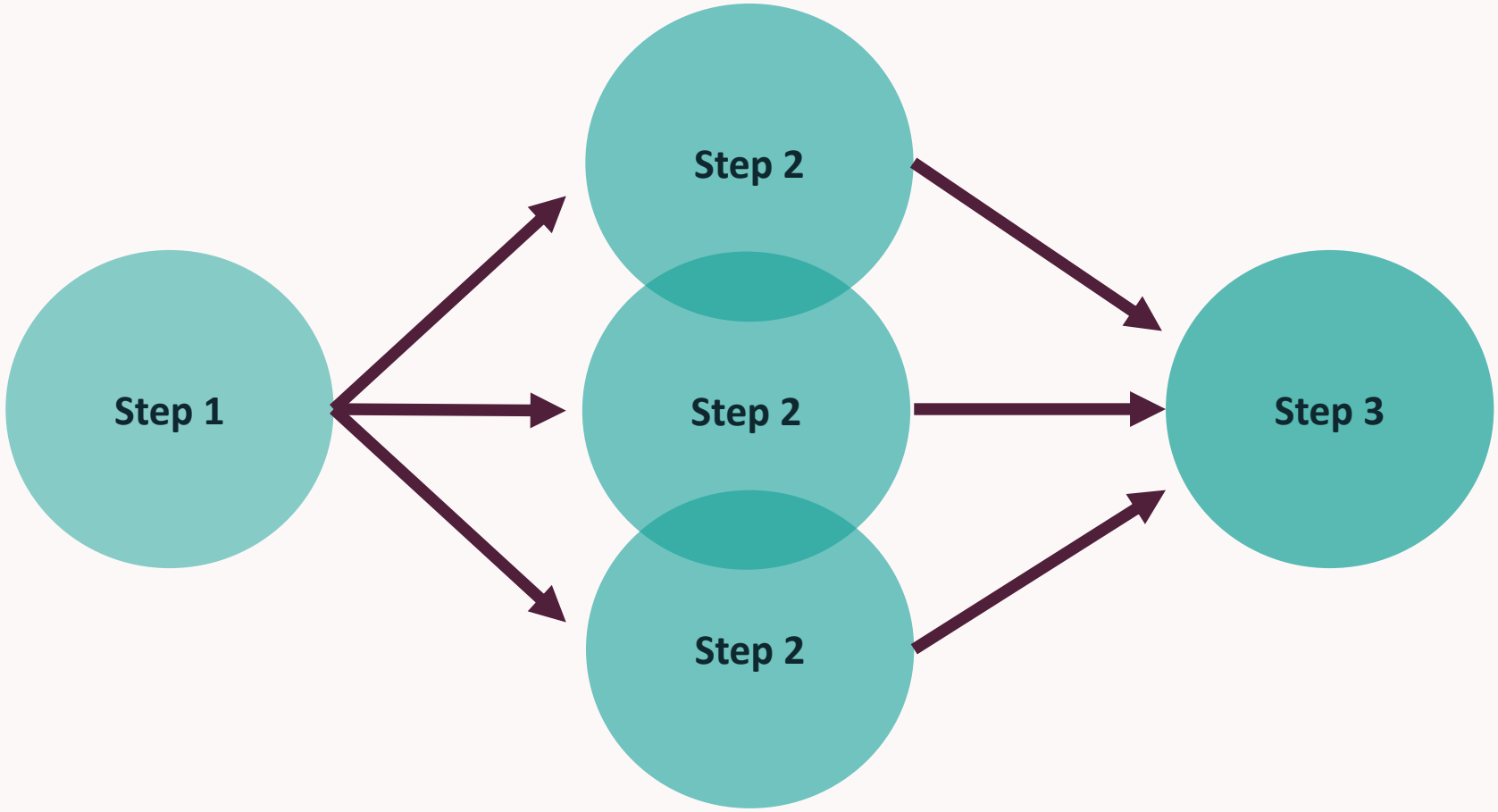
Containerization Demo

Schedulers

Job Schedulers

Each task (step) in the workflow is processed in order, starting with step 1, which is triggered at a specific time according to the scheduler

Job Schedulers

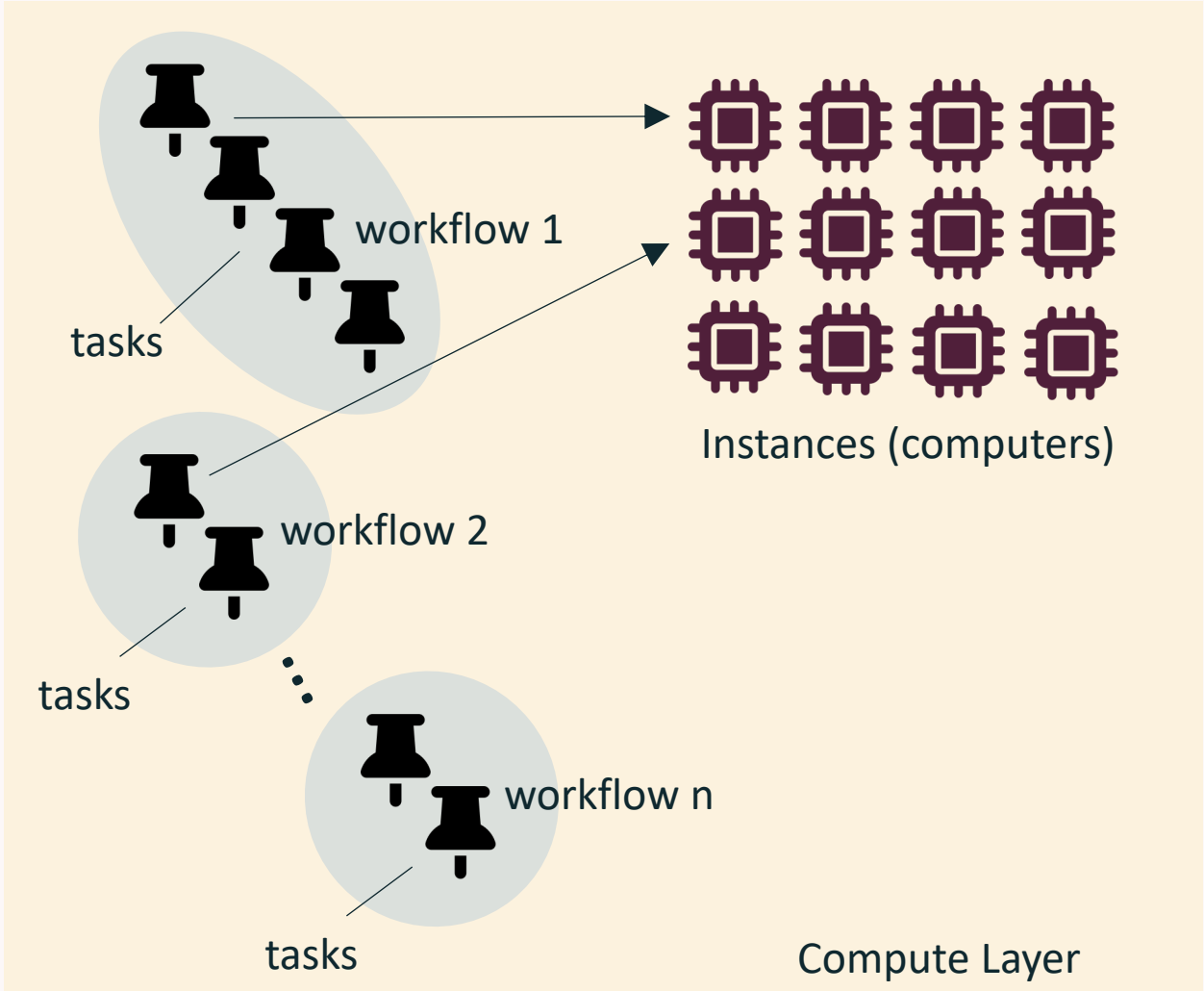


Each instance of Step 2 can run in parallel

Orchestration

Compute, Containers, and ML Workflows

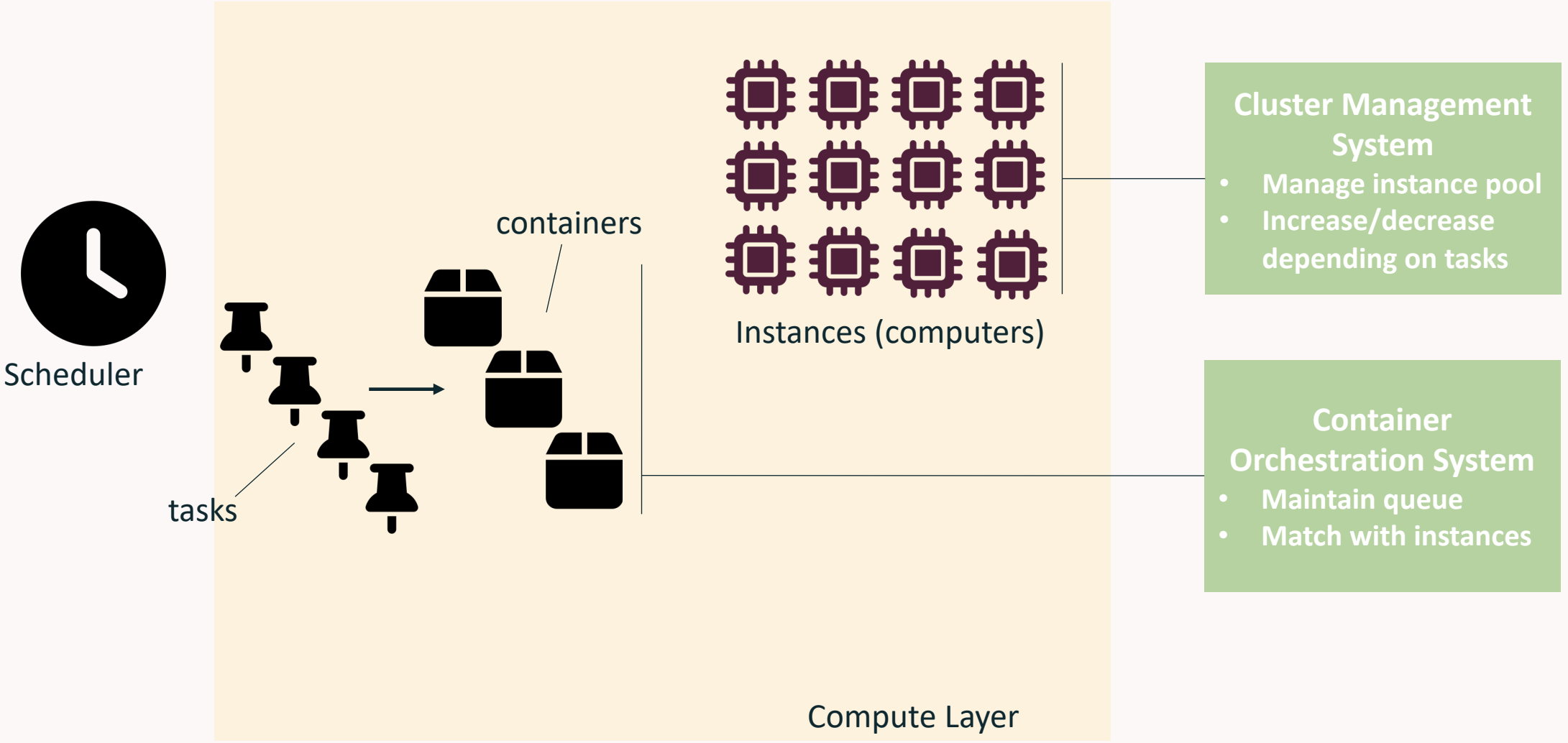
Compute Layer Finds a Place to Execute Tasks



Scheduler

1. Scheduler kicks off a workflow
2. Tasks are submitted
3. The tasks are put in a queue
4. An instance is assigned to a task, from a pool of available instances
5. If no instance is available, new instances can be created
6. Task is executed in a container

Compute Layer Finds a Place to Execute Tasks



Cluster Management System

- Manage instance pool
- Increase/decrease depending on tasks

Container Orchestration System

- Maintain queue
- Match with instances

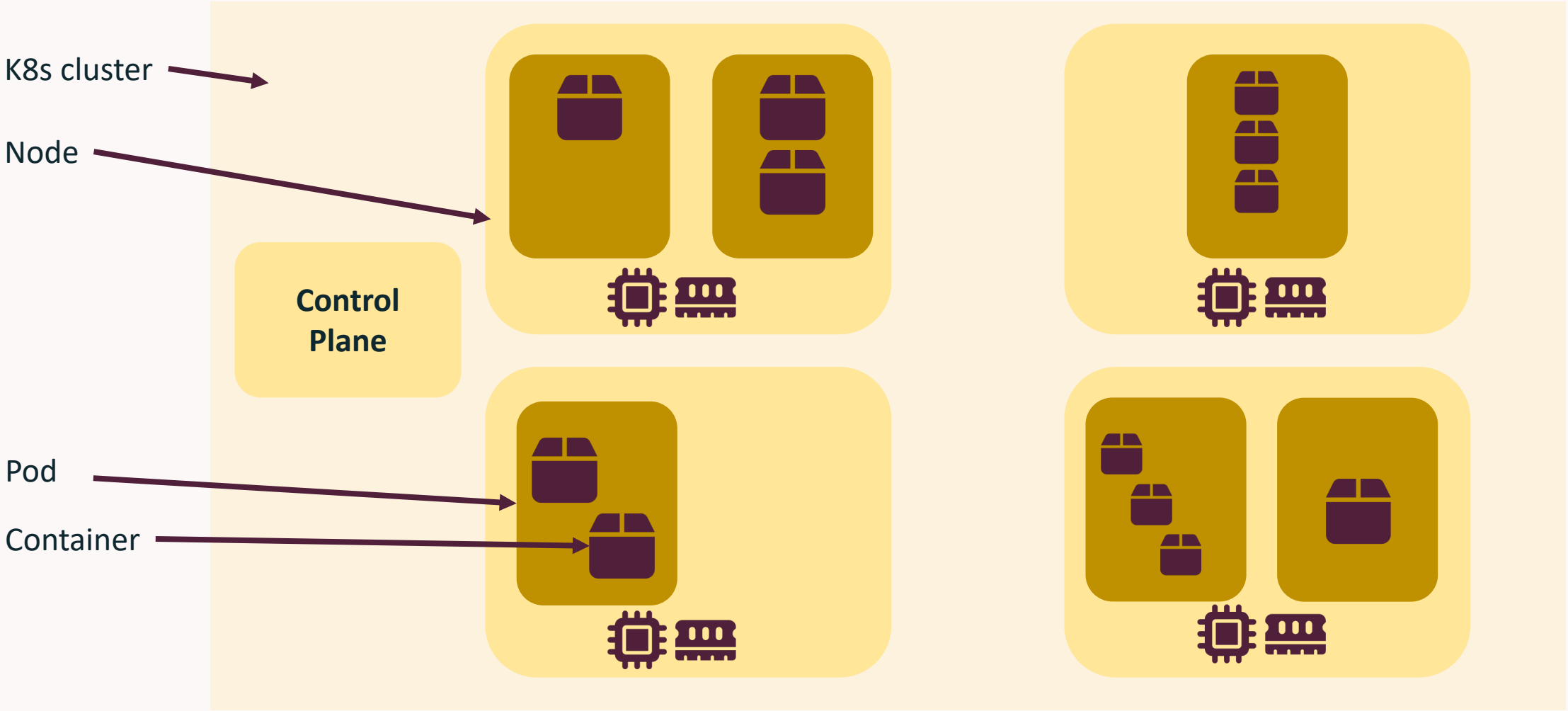
Considerations

- Workload support – big data processing (spark), deep learning training (GPUs)
- Latency –
  - model development = low latency;
  - batch processing != low latency
- Workload management – decline, add to queue, terminate
- Cost-efficiency – billing granularity (by hour, minute, second, etc.)
- Operational complexity – ease of use and maintenance

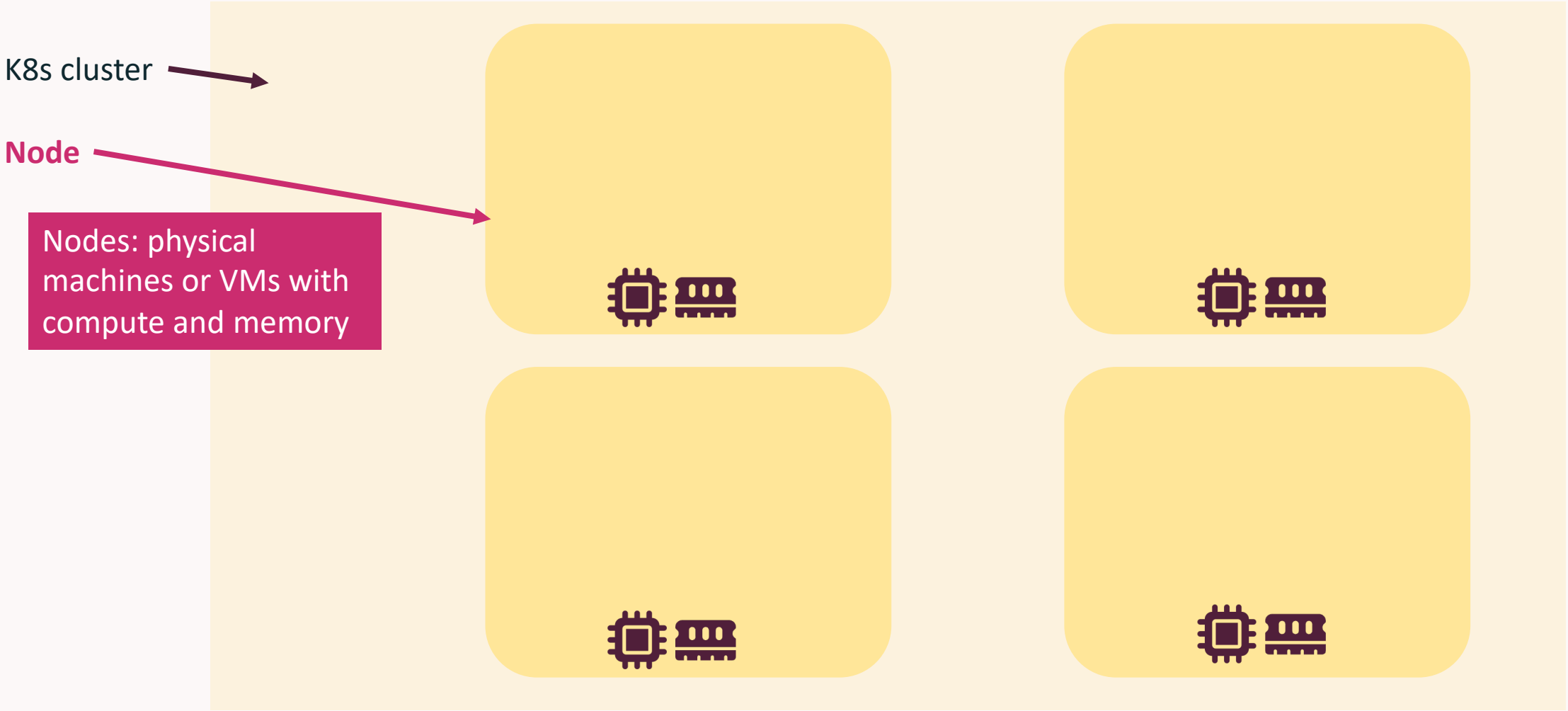
Options

- Kubernetes (K8s) on AWS, GCP, Azure, or on-prem
- AWS Batch/Lambda/Fargate, GCP Cloud Run (serverless) and Azure equivalents
- Spark
- Sagemaker and alternative fully managed services
- Laptop/workstation

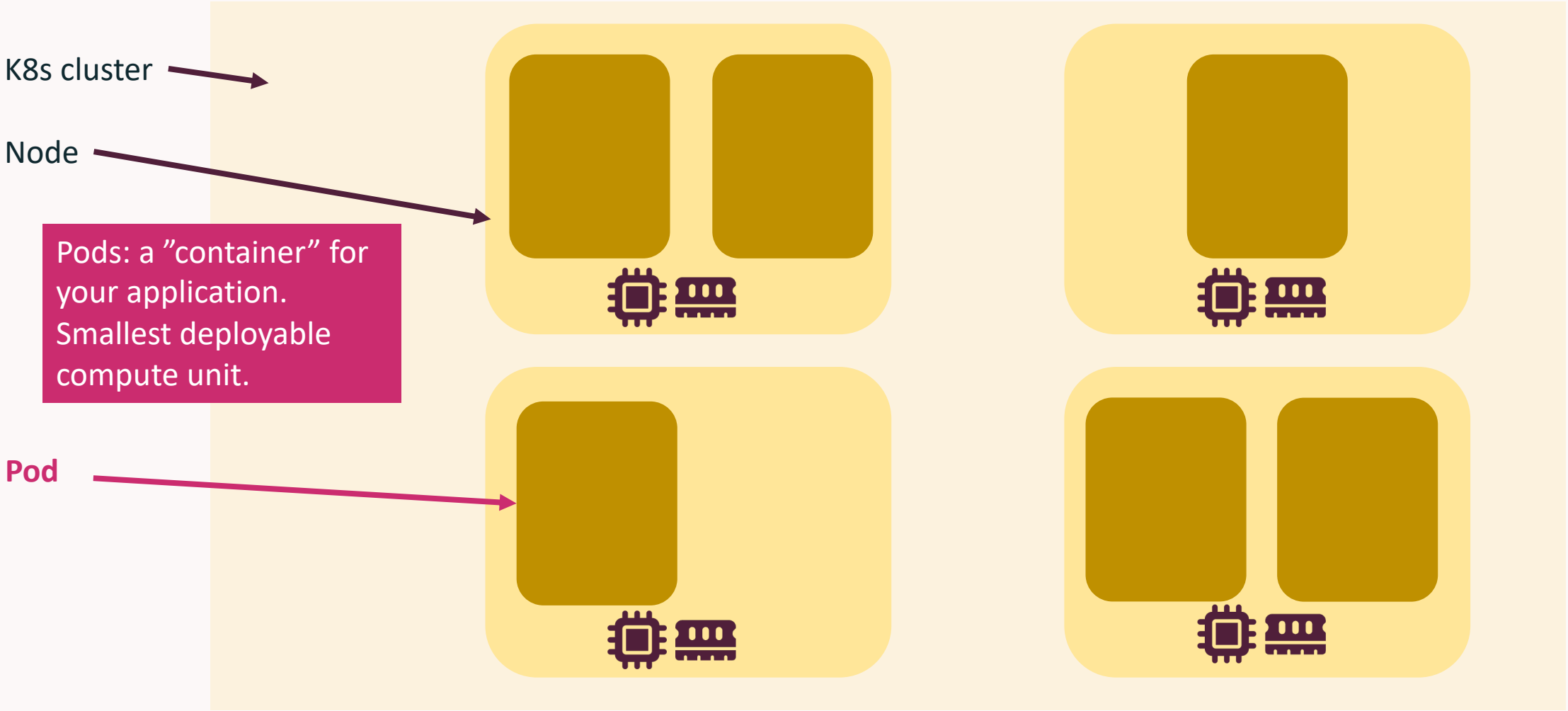
Kubernetes



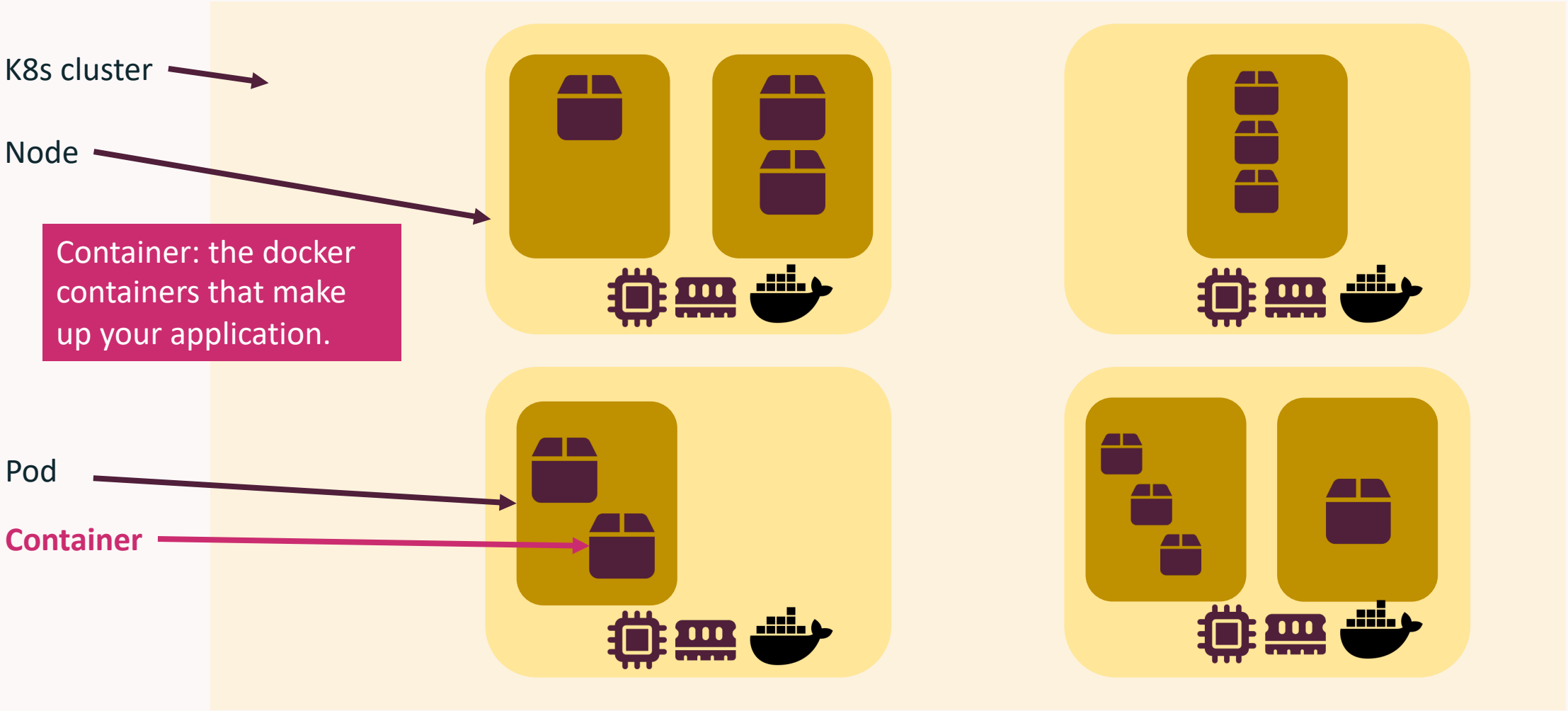
Kubernetes


K8s cluster
Node
Nodes: physical machines or VMs with compute and memory

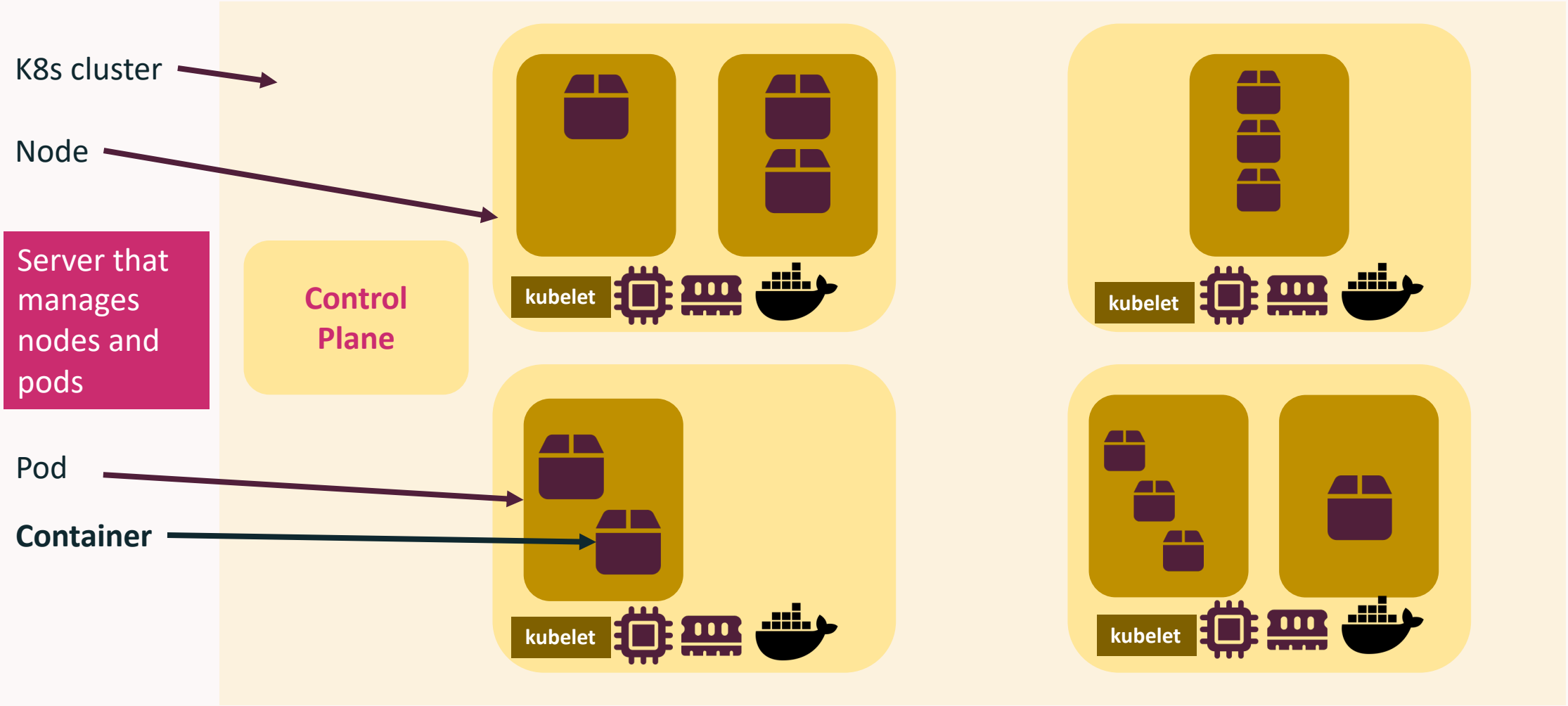
Kubernetes


K8s cluster
Node
Pods: a ”container” for your application. Smallest deployable compute unit.
Pod

Kubernetes


K8s cluster
Node
Container: the docker containers that make up your application.
Pod
Container

Kubernetes


K8s cluster
Node
Server that manages nodes and pods
Control Plane
Pod
Container
kubelet
kubelet
kubelet
kubelet

Pods

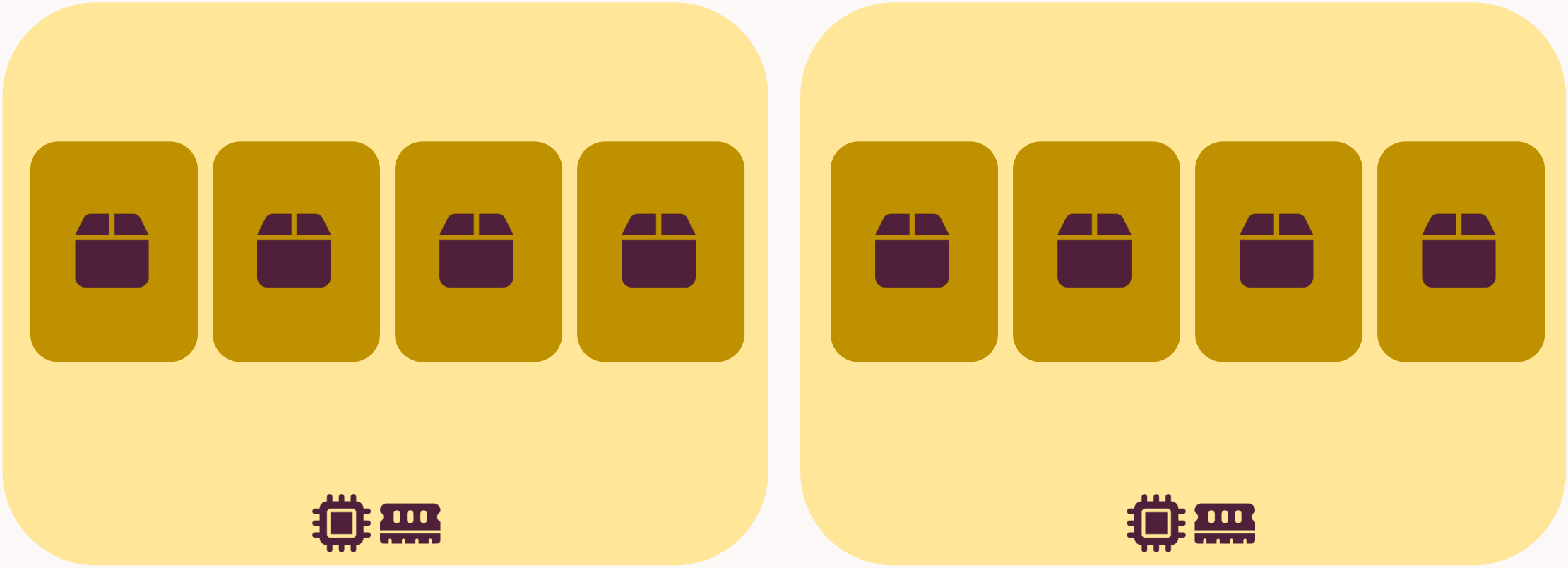
```
kubectl describe pods
kubectl describe deployment
```

- Kubernetes Pods:
  - Single container or group of containers sharing resources, network, port space
  - Hosts entire application, or part of one
  - K8s decides which node to use, spins up new instance if a node fails, and spins up multiple if asked
  - Pod defined by a manifest (yaml)

```yaml
apiVersion: v1
kind: Pod
metadata:
     name: nginx
spec:
     containers:
     - name: nginx
        image: nginx:1.14.2
        ports:
        - containerPort: 80
```

Pods

Replication = Horizontal scaling



Deployments

```
kubectl describe pods
kubectl describe deployment
```

- Deployments:
  - Define how the pod will be “deployed” (created), e.g. how many instances, whether to upgrade the image, etc.
  - If pod goes down, the deployment file instructs the Control Plane how to bring up a new pod
  - Deployment defined by a yaml file

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deployment
  labels:
    app: nginx
spec:
  replicas: 3
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image: nginx:1.14.2
        ports:
        - containerPort: 80
```

Orchestration Demo