

# Distributed Infrastructures beyond the Grid paradigm

How to choose where to run

sara.vallero@to.infn.it

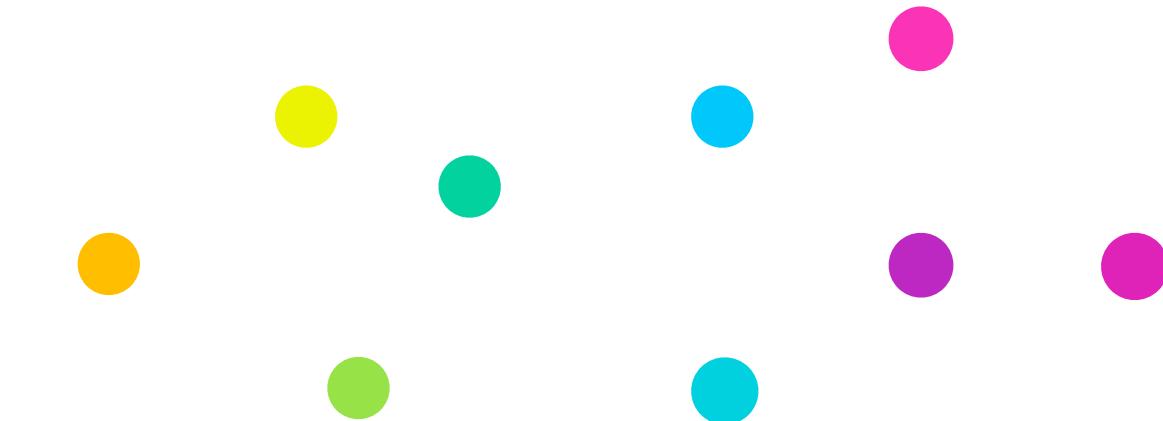
# Identify your problem's features

# High Throughput

- Efficient execution
- Long execution times
- Large amount of tasks
- Tasks are loosely coupled (sequential)

## Example:

- a relatively simple code to analyse a huge amount of experimental data
- each task processes a chunk of the data sample independently



# High Performance

- Large amount of computing power
- Short execution times
- Tasks are tightly coupled (parallel)
- Low latency interconnect

## Example:

- simulate the atomic motion of a protein in water
- each atom interacts with the others in the system
- communicate data back and forth each sub-task

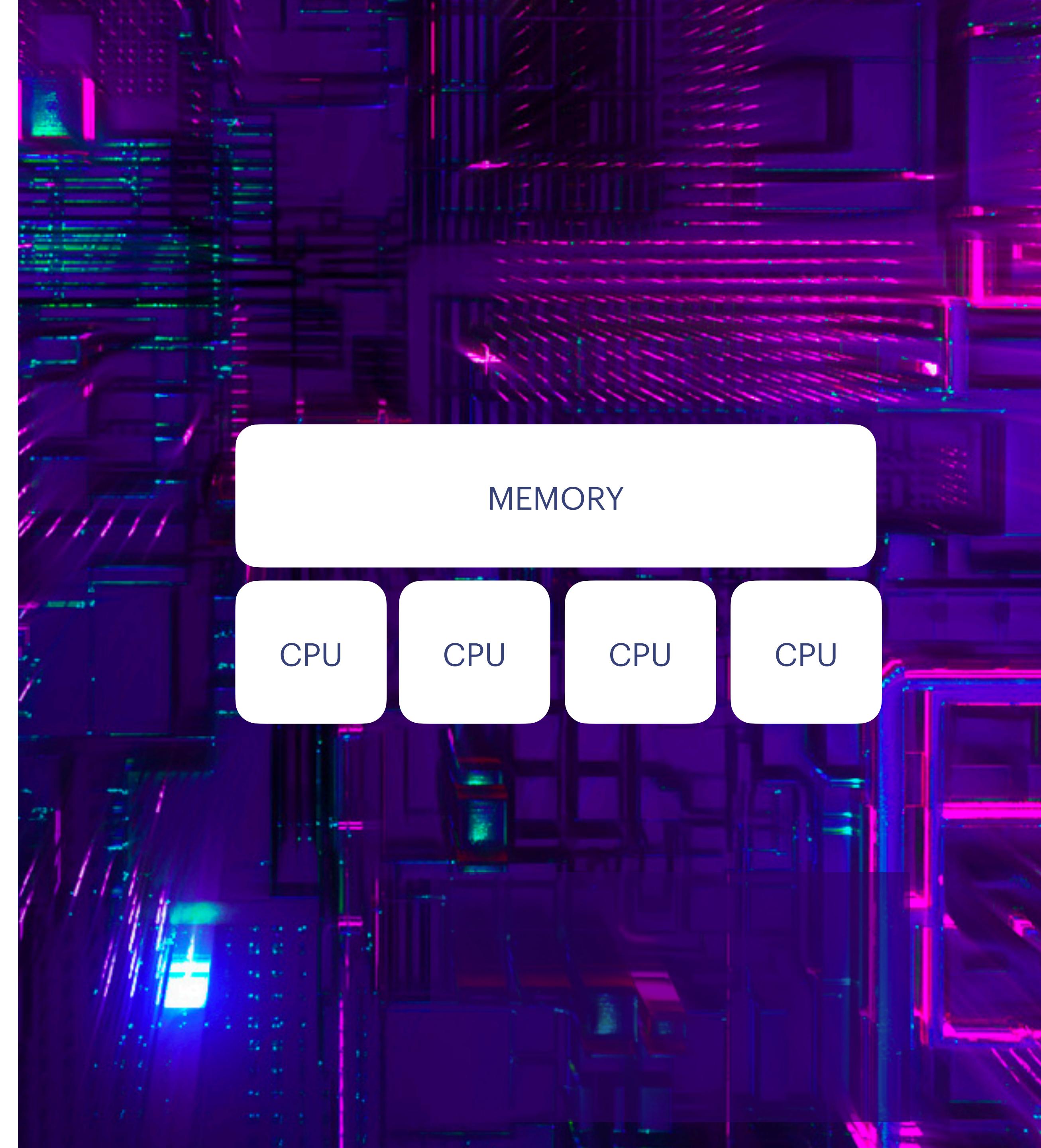


# Choose the infrastructure type

# Multicore Server

## Shared memory:

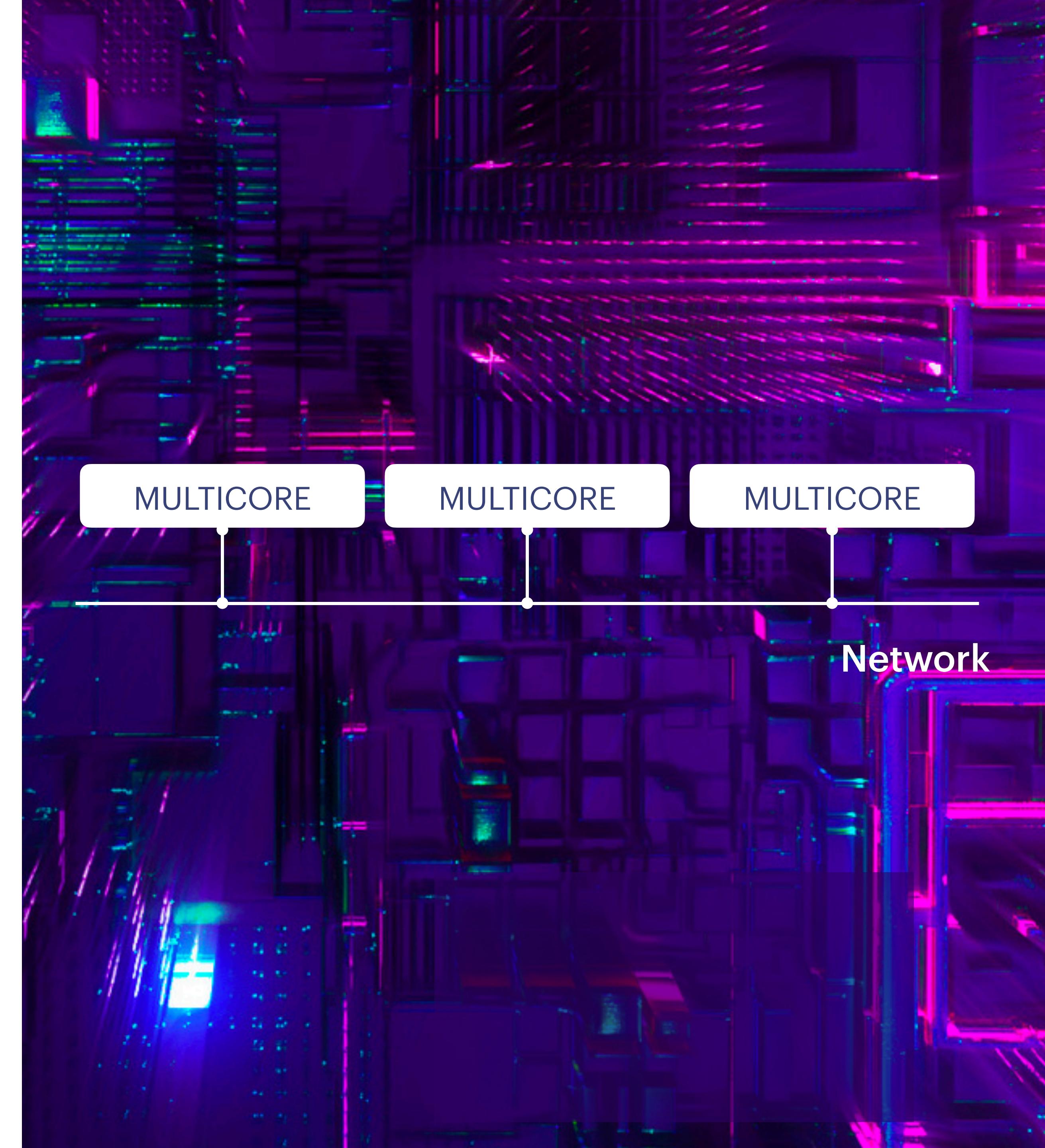
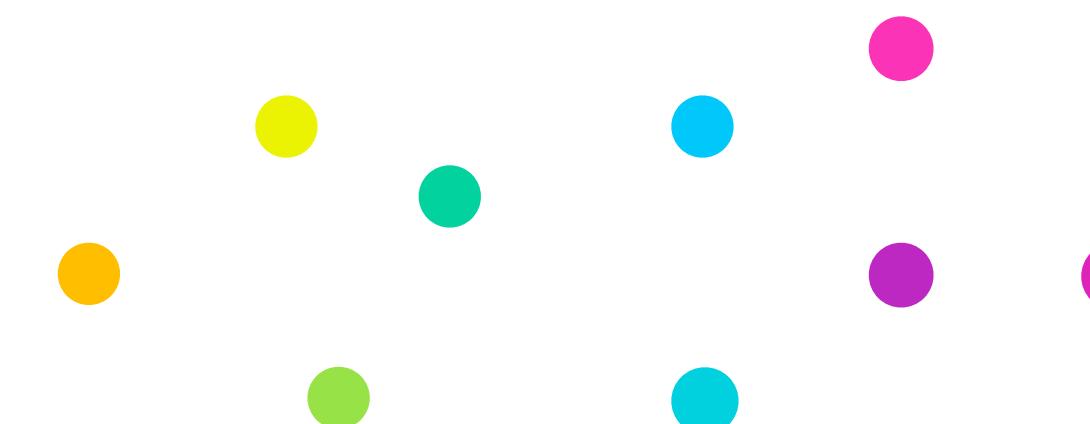
- Single address space
- All processes have access to the pool of shared memory



# Cluster

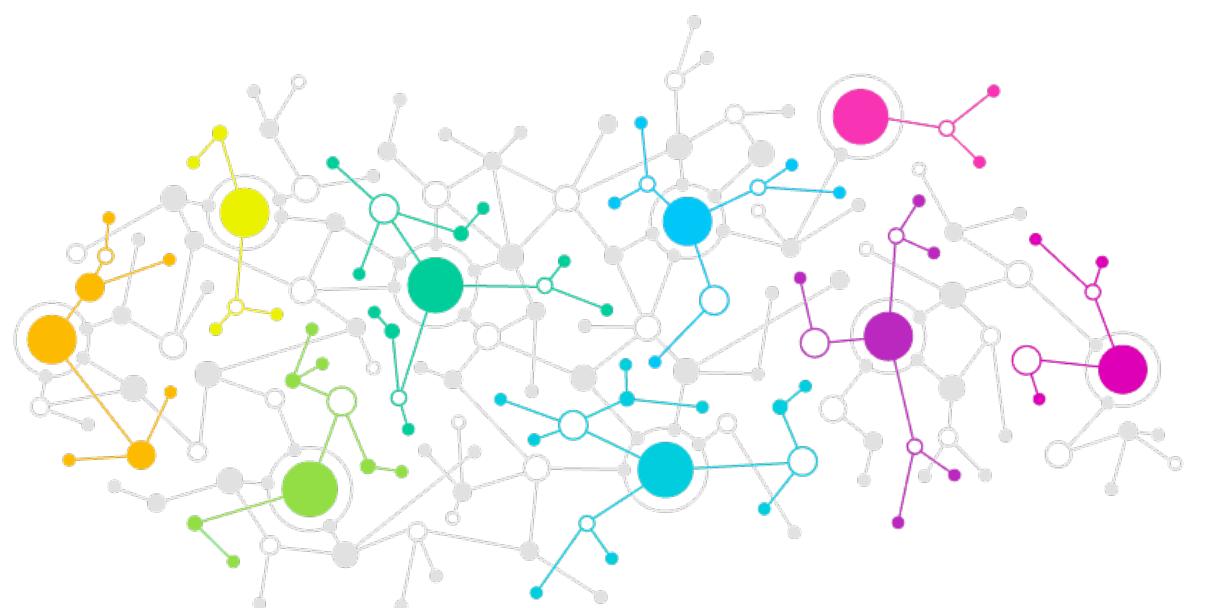
## Distributed memory:

- Each processor has its own local memory
- Message passing (mainly) is used to exchange data between processors



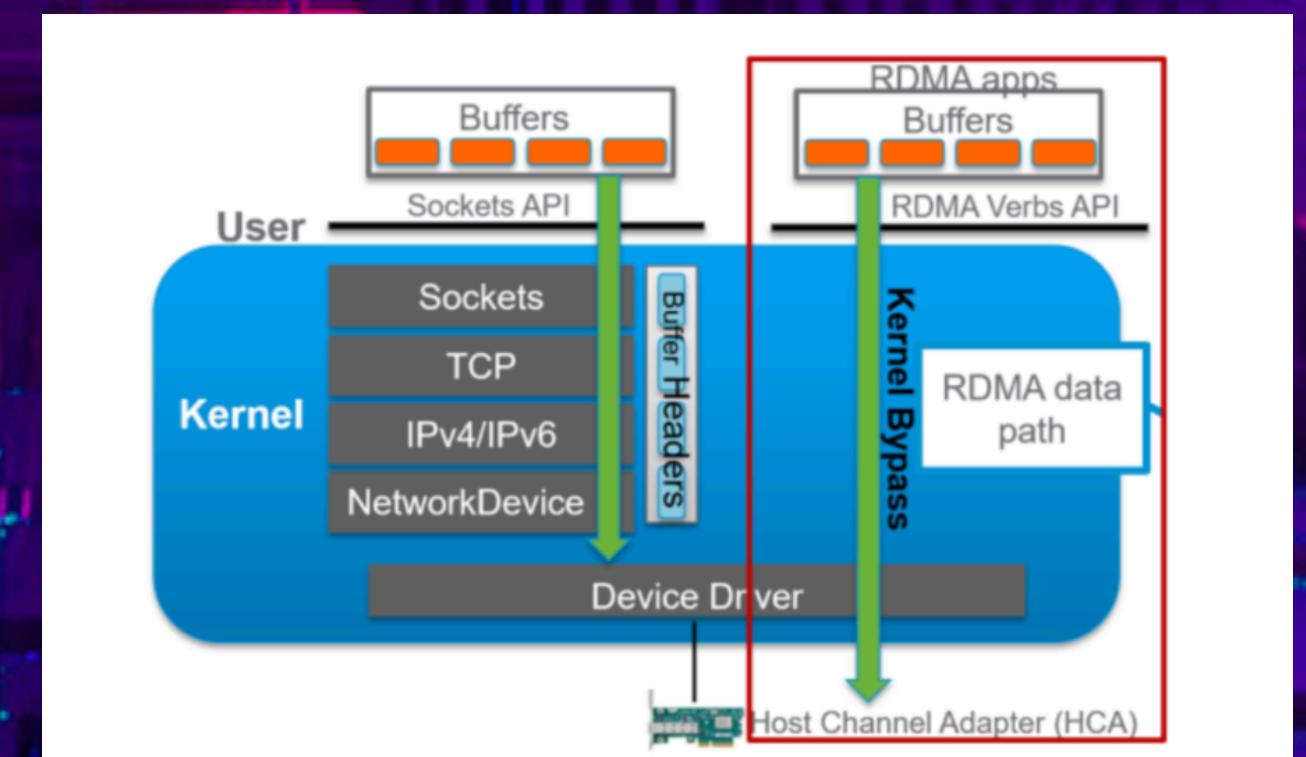
# HPC Cluster

- Performant CPUs (i.e. high clock rate)
- Low latency interconnect:
  - i.e. Mellanox's *InfiniBand* or Intel's *OmniPath*
  - offer an RDMA implementation

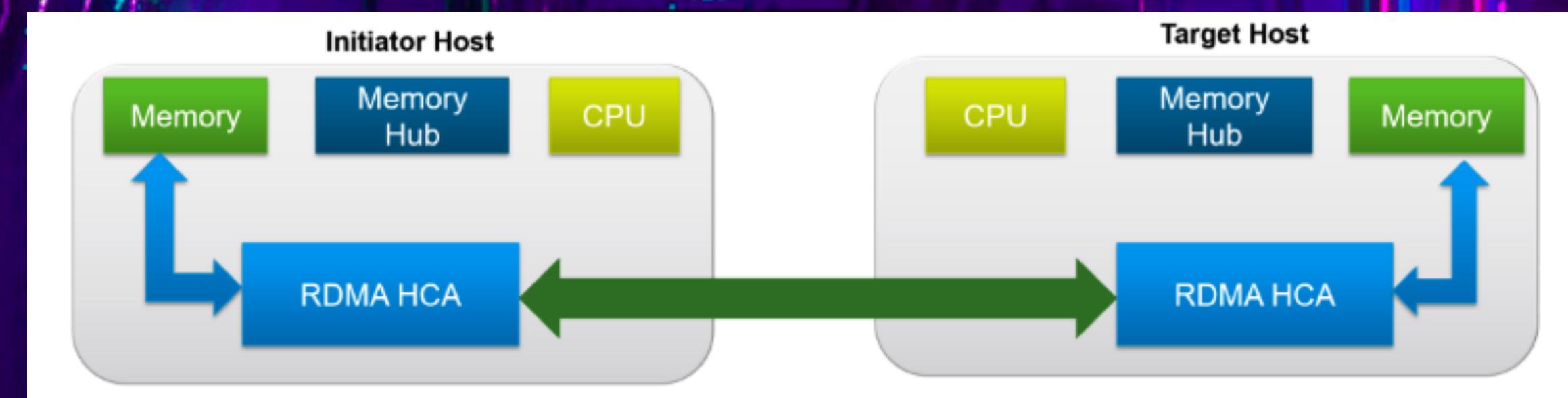


## Remote Direct Memory Access (RDMA)

- Transfer of memory between different computers
- Direct transfer minimising CPU/Kernel involvement
- Bypassing the Kernel allows high I/O bandwidth and low latency

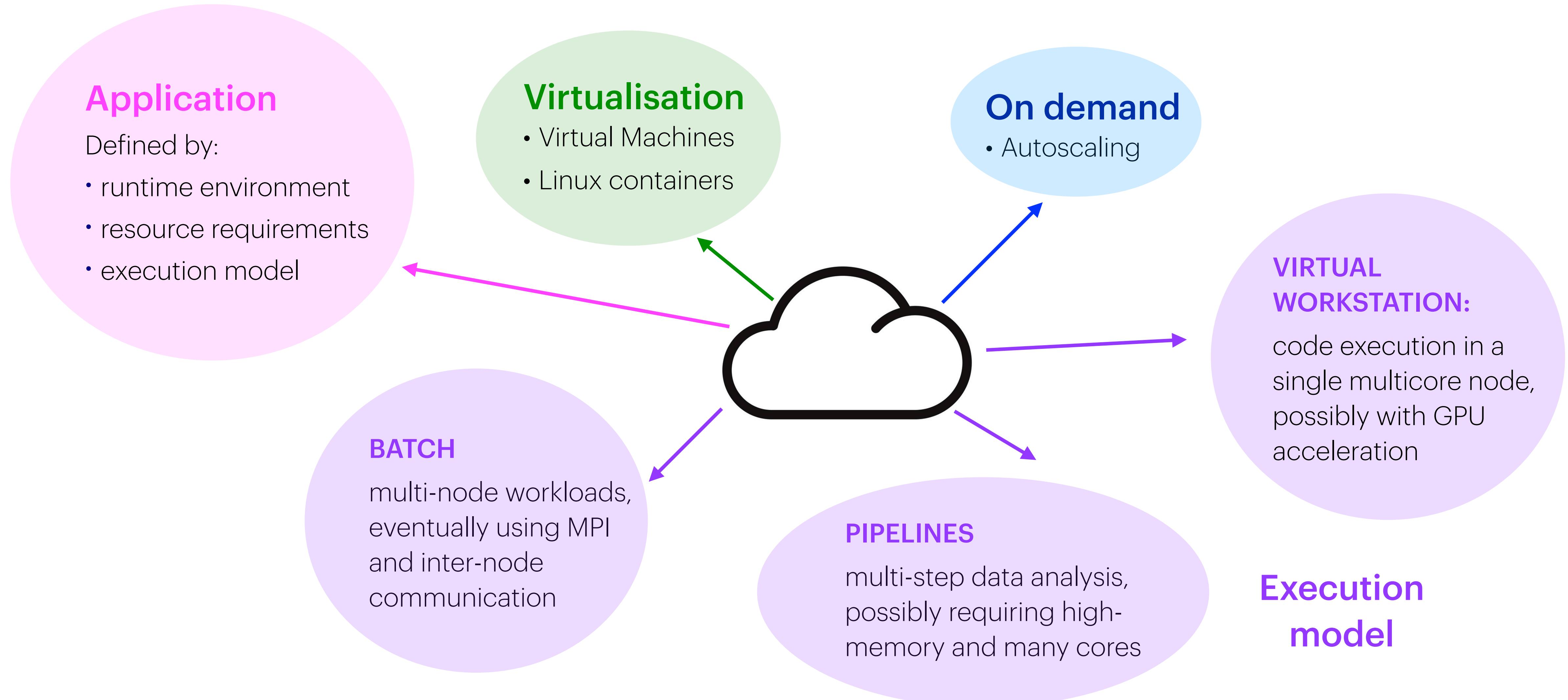


- Host Channel Adapter (HCA) needed on both source and destination



# What about The Cloud?

**Cloud computing:** a style of computing in which scalable and elastic IT-enabled capabilities are delivered **as a service** using **Internet** technologies.

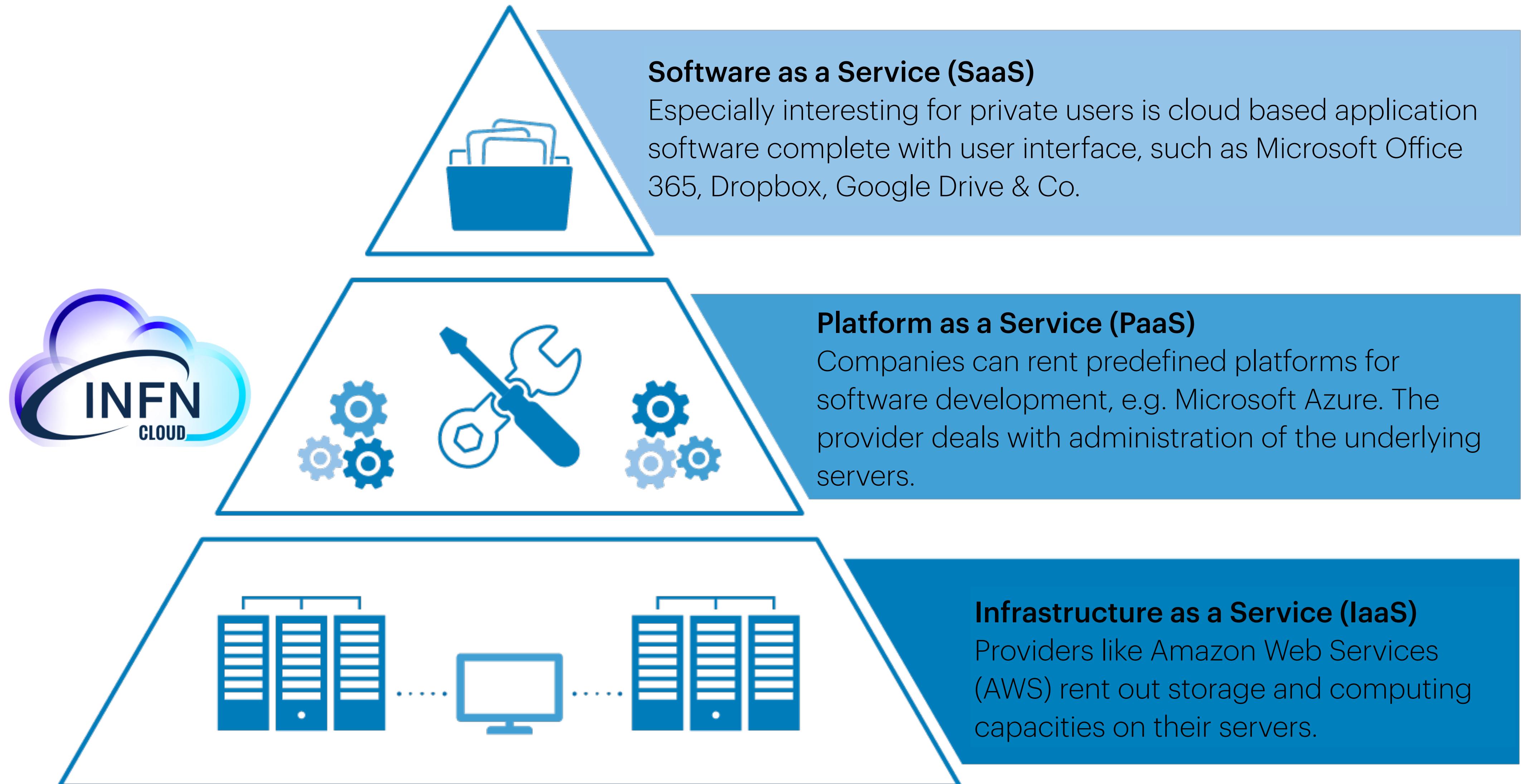




Google Cloud



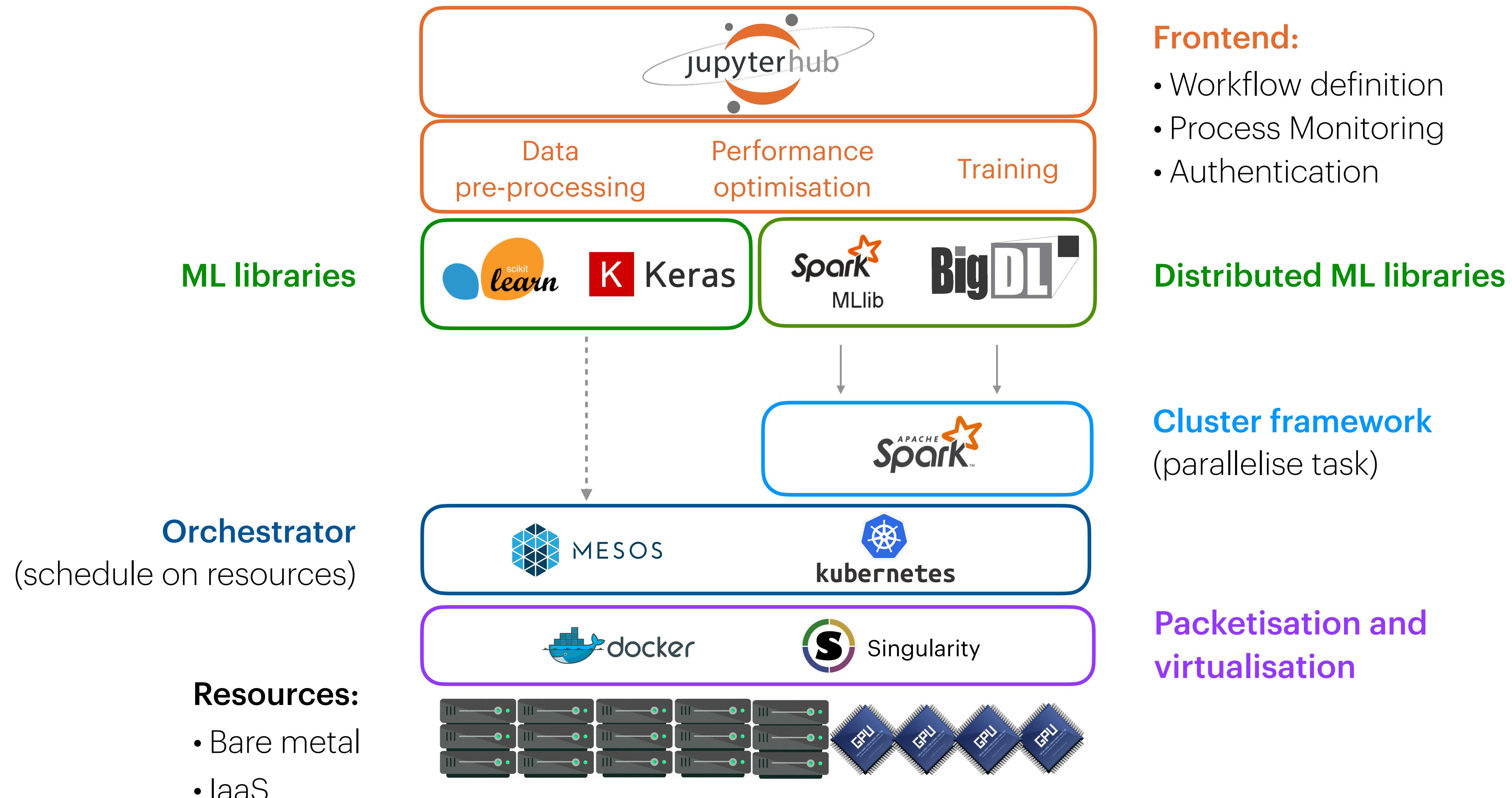
# The Cloud Pyramid



# Enabling Machine Learning workflows

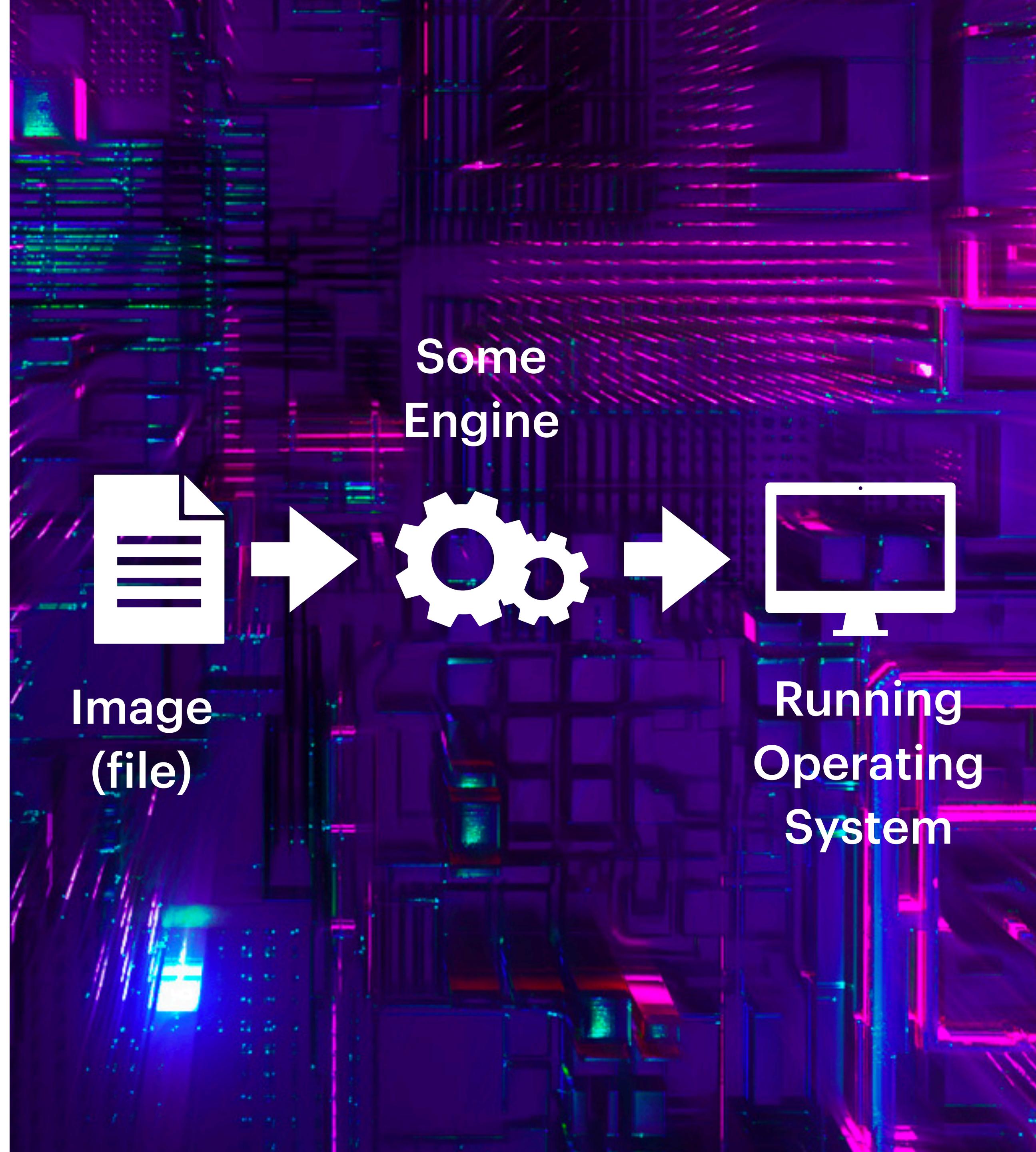
**ML as a Service**

# Layers

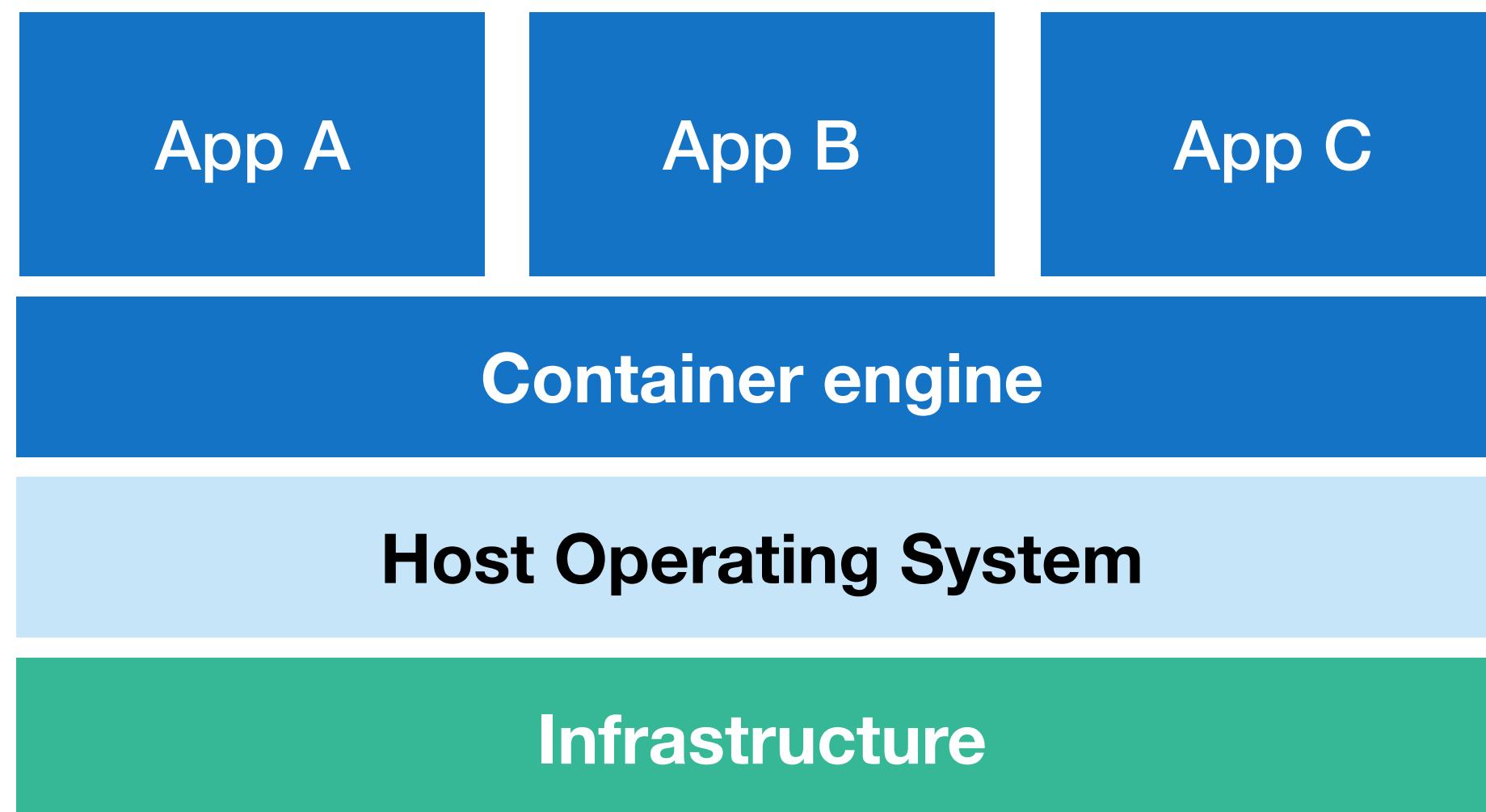


# Virtualisation

- **Operational definition:** a way to provide an **isolated environment** in which to run an application. The environment is represented as a **binary artifact** that can be moved between hosts.
- The foundation of **Continuous Delivery:** code changes are automatically built, tested, and packaged for release into production.

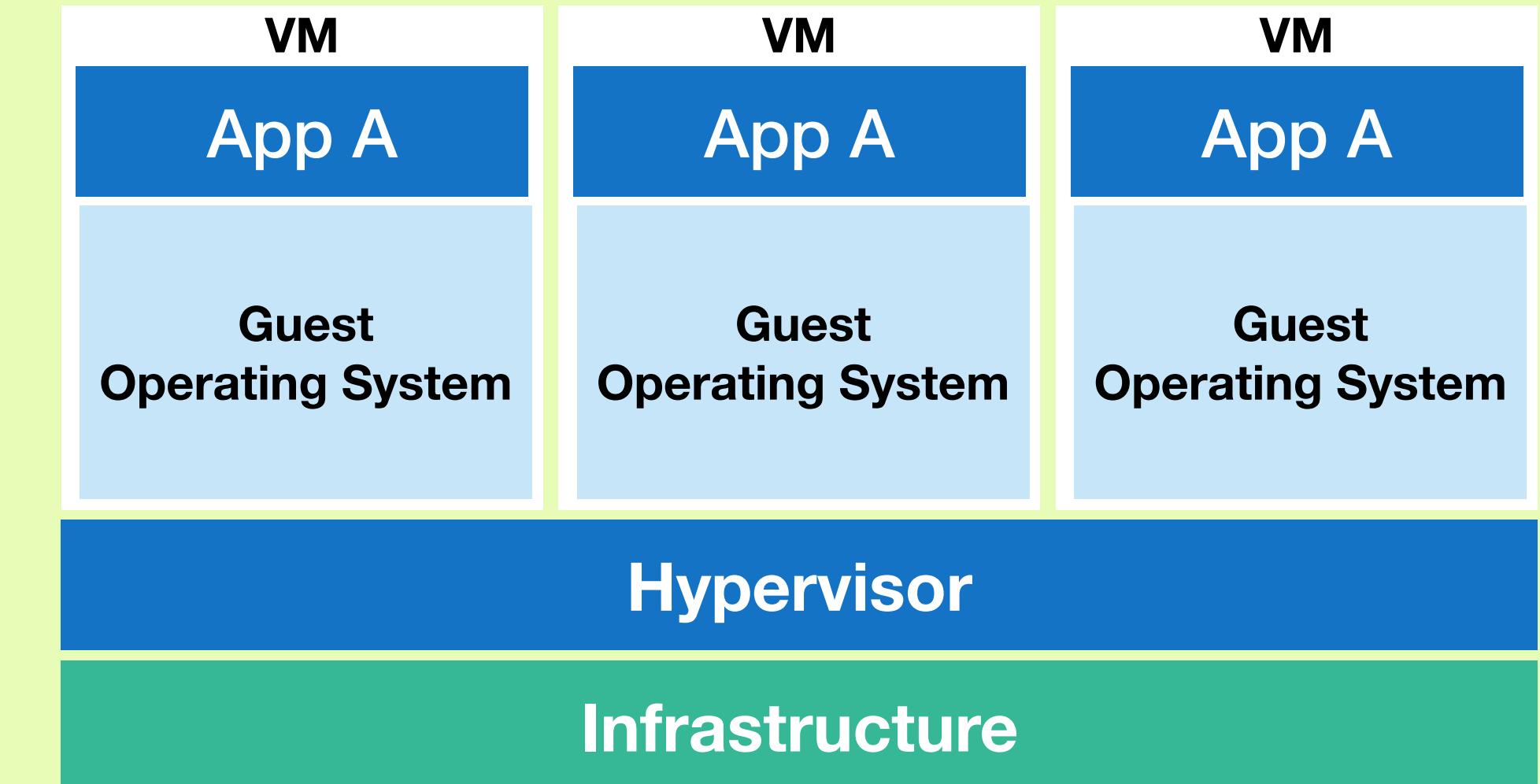


# Linux Containers



- **Lightweight** encapsulation: portable and efficient
- Virtualise the **operating system** (not hardware)
  - Minimal image containing just what is needed

# Virtual Machines

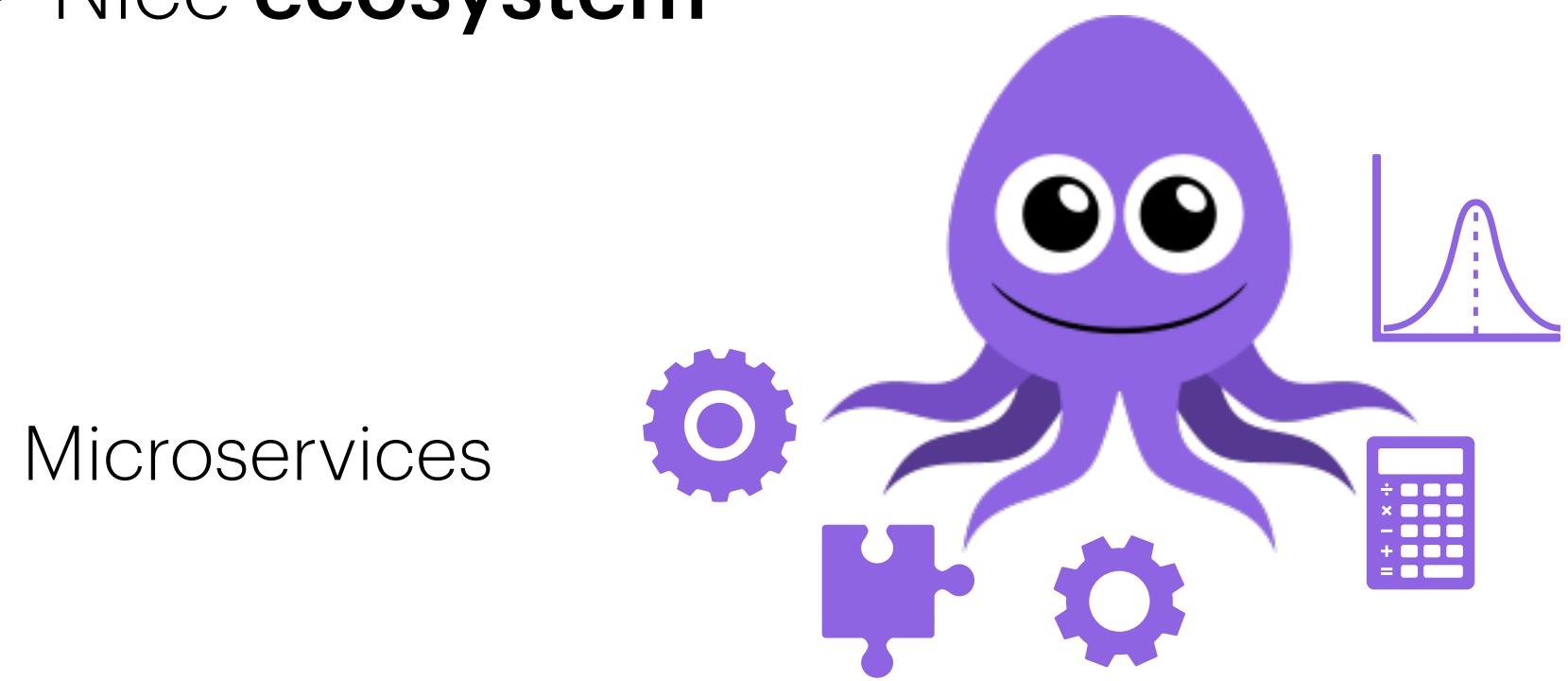


- **Hardware** virtualisation:
  - More **isolation**
  - But overhead
- Image containing the full operating system

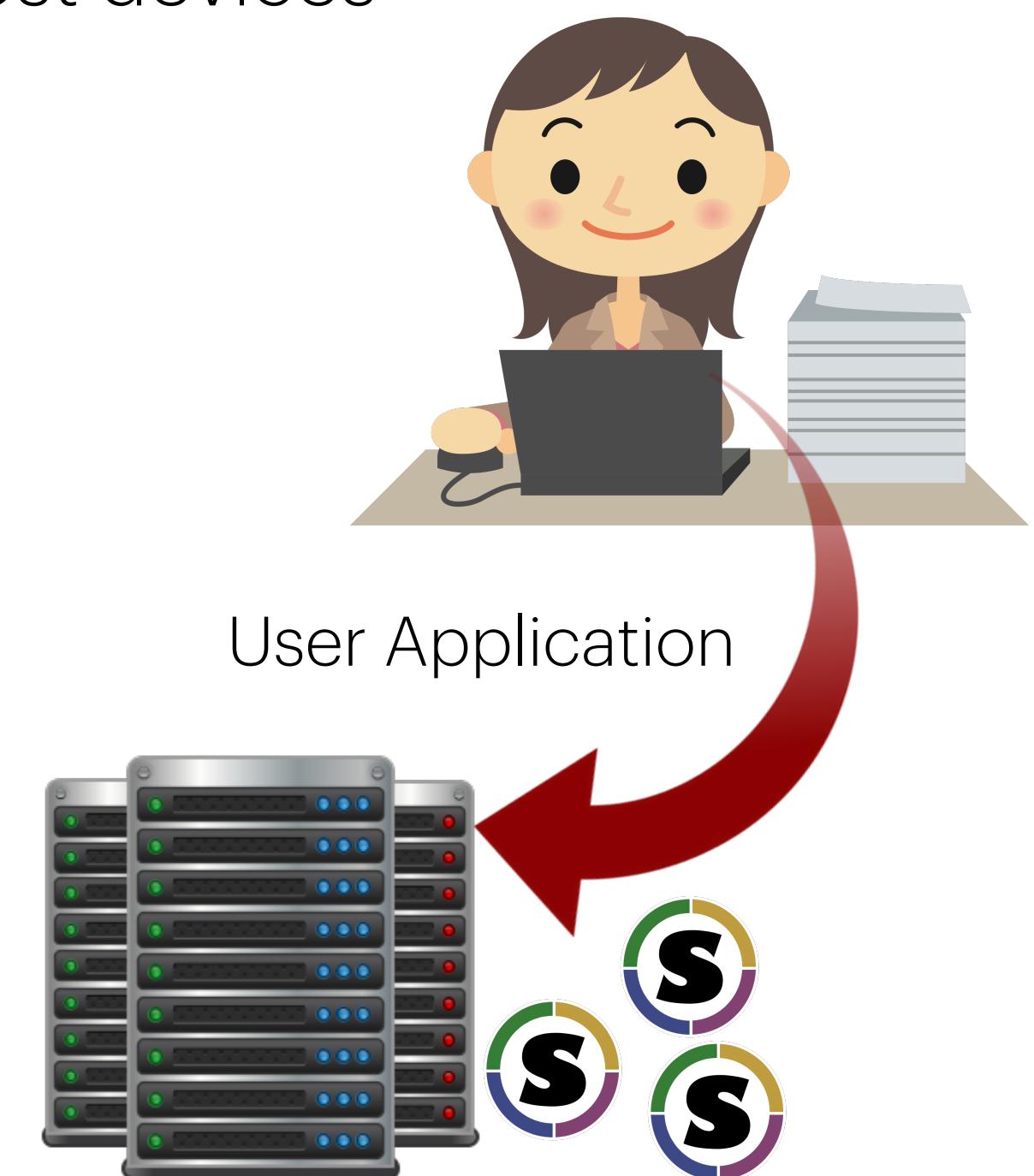
# Linux Containers: what about “the engine”?



- **Microservices**
- Fancy network set-ups
- Host devices access
- Host storage access
- Requires root privileges (almost)
- Nice **ecosystem**



- Runs in **user-space**
- No access to host devices
- No networking



# Orchestration



Container orchestration refers to the process of **organising** the work of individual (**loosely coupled**) containerized components and application layers, that need to work together to allow a given app to function as designed.



# kubernetes

- Builds upon 15 years of experience of running production workloads at Google
- Integrates ideas and practices from the community
- A widely used tool for **Continuous Deployment**: every validated change in the application is automatically released to users.

## Some nice features

- Service discovery and load balancing
- Storage orchestration
- Automatically places containers based on their resource requirements and other constraints (bin packing)
- Self-healing
- Automated rollouts and rollbacks
- Horizontal scaling

# Kubernetes concepts

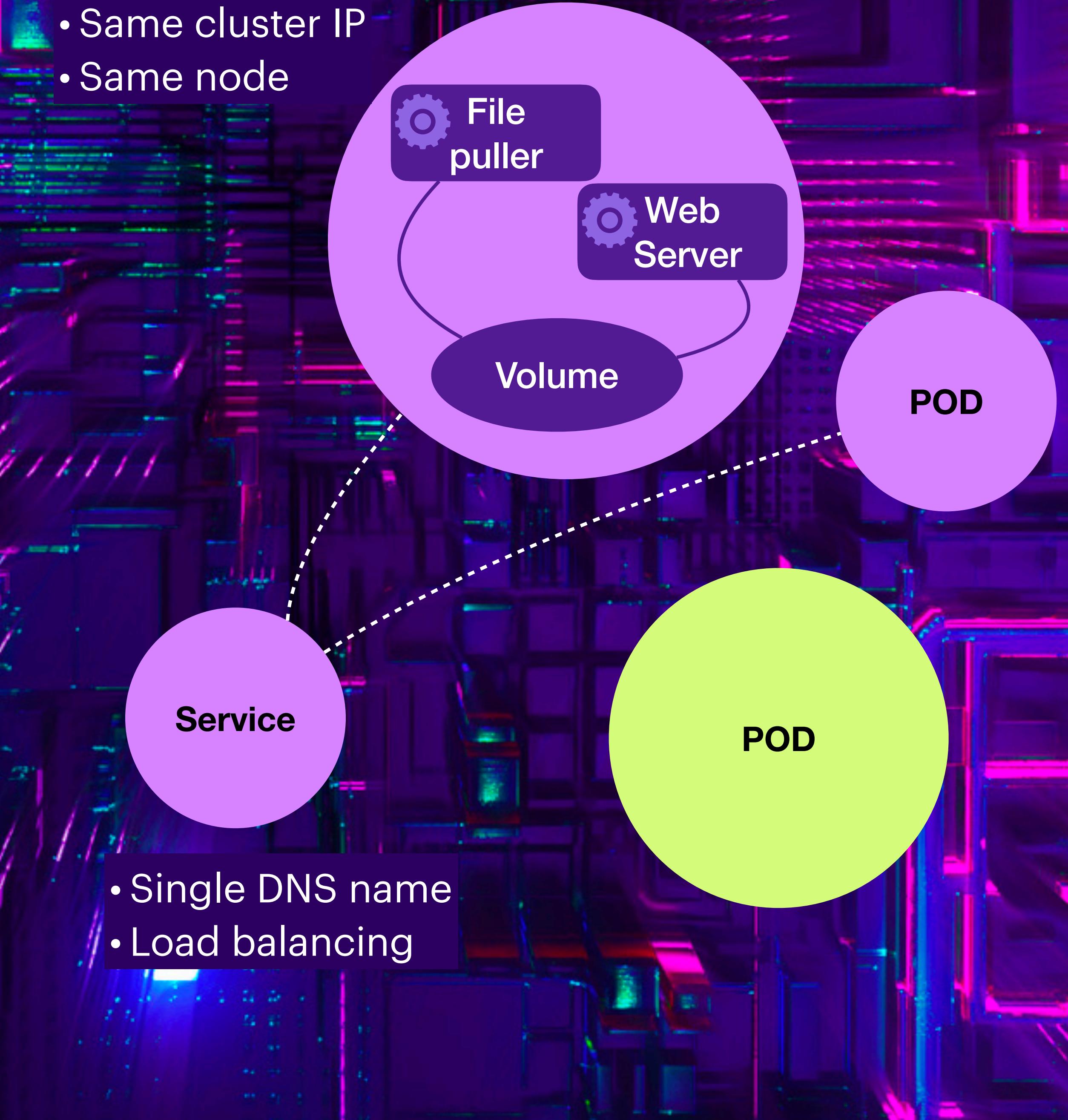
**OBJECTS** are abstractions that represent your system's state.

Basic objects include:

- Pod
- Service
- Volume
- Namespace

Plus higher-level abstractions that rely on Controllers to build upon the basic objects (i.e. Deployments).

- Same cluster IP
- Same node



# The declarative model



**IMPERATIVE:** define the exact steps to be executed to achieve the desired final state.

**DECLARATIVE:** describe your desired final state. Let the system decide how to achieve the goal.



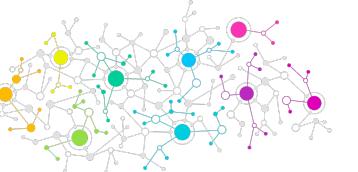
- write local configuration files (yaml)
- use the Kubernetes CLI to apply it
- create, update, and delete operations are automatically detected per-object

my\_service\_configuration.yaml

```
apiVersion: v1
kind: Service
metadata:
  name: my-service
spec:
  selector:
    app: MyApp
  ports:
    - protocol: TCP
      port: 80
      targetPort: 9376
```

POD

# To take home

- Understand your problem's features: is it embarrassingly parallel or not?
- Choose the right type of infrastructure to execute your task:
  - Multi-core  
  - HTC cluster 
  - Grid 
  - HPC cluster 
  - Any of those over a Cloud (maybe not HPC)

# Wrap-up

No BigData/MachineLearning expert is such if she cannot compute a real-size problem (scale-out/scale-up)

We introduced several concepts and tools which are at the basis of modern computing strategies, now you know how to place them in the right context.

We hope this will help you in your daily work.