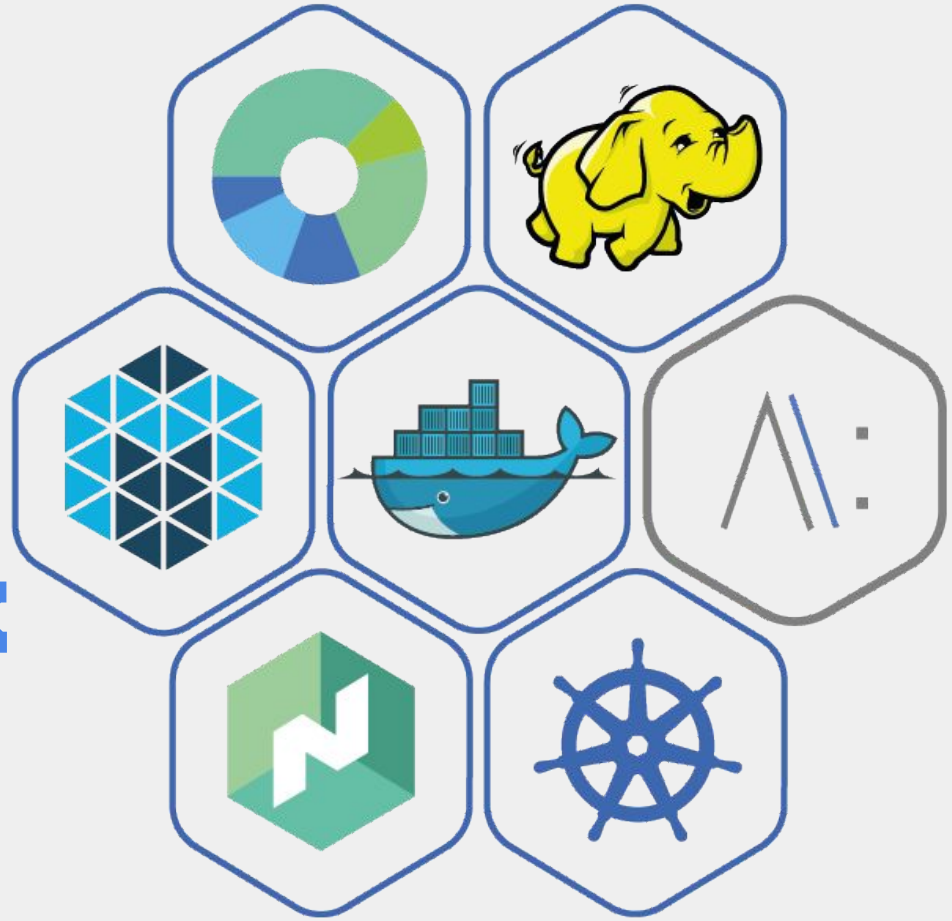
Il ne faut pas vendre la peau de YARN parce qu'un Mesos vaut mieux que deux Kubernetes



Hi! My name is

I am a software engineer, specialized in Big Data with a growing interest in AI/ML/DL.
In my everyday work, I am involved in cloud computing and data management in general, including the use of open data, issues related to the capture, storage, retrieval, sharing, analysis and visualization of big data, and finally make sense of data with the semantic Web. I define myself as a technical person and I love getting my hands in the code.
More gibberish with a little bit of gobbledygook, some rigmarole, gabble, hocus-pocus and so on...

BIG DATA, CLOUD

Pascal GILLET
Cloud Architect
A: STACK LABS
@pagillet

Expert en *nuages*

“A cluster is a set of nodes with at least one master node and several worker nodes.

The cluster manager is mostly used to dispatch work for the cluster (or cloud) to perform.”

Wikipedia

“An operating system architecture for cluster-level resource management.”

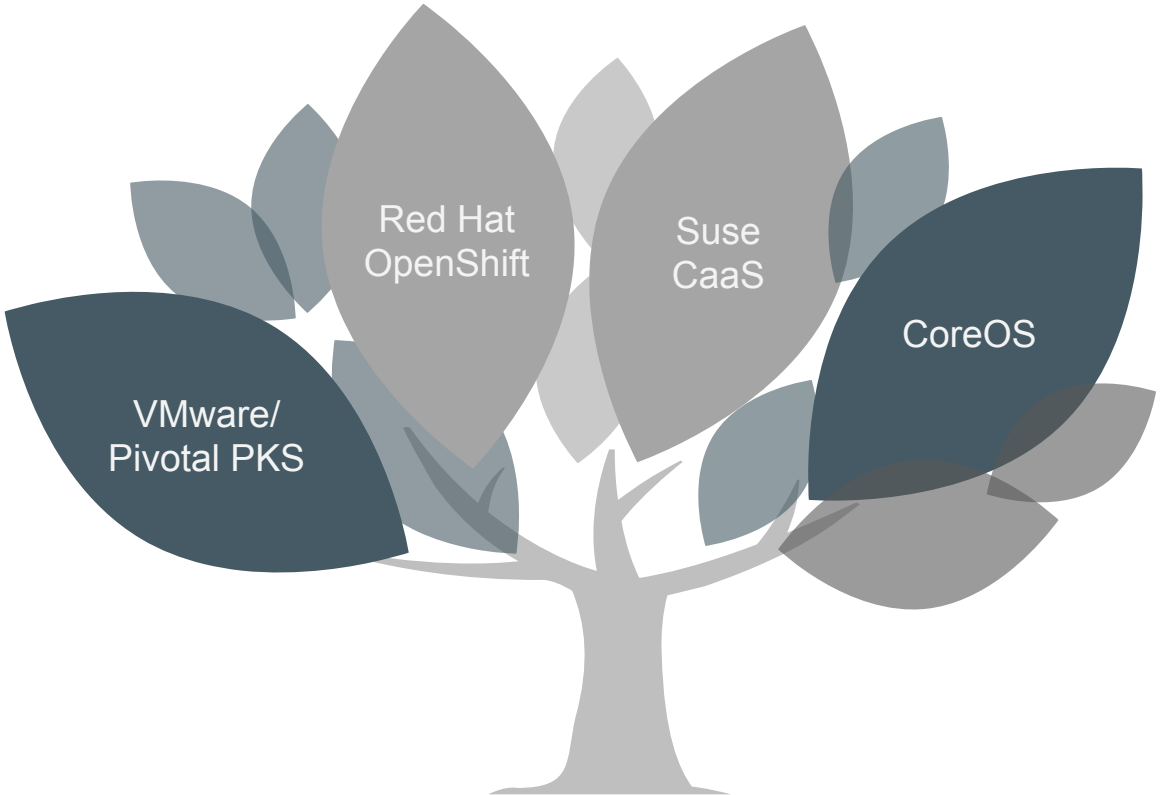
The Dalaï-Lama

Cluster managers

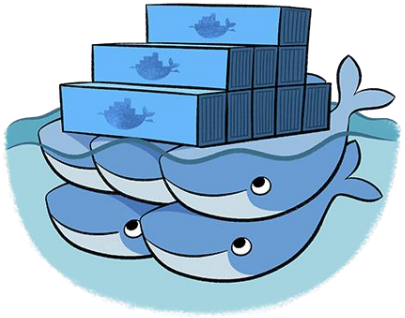
vs

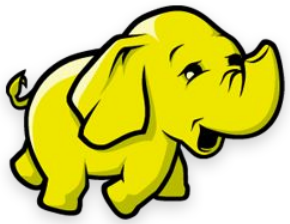
Container orchestration systems

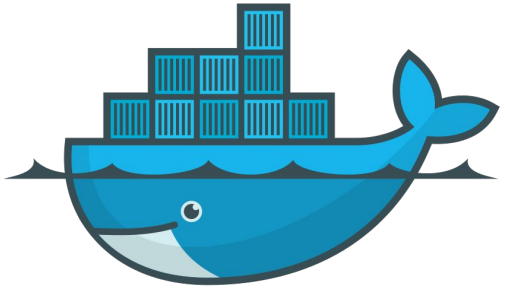
Kubernetes is everywhere...

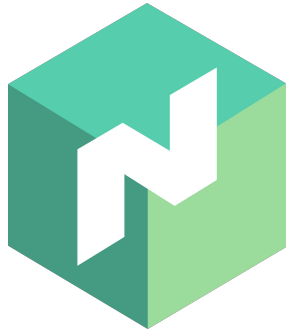


… But there are many others









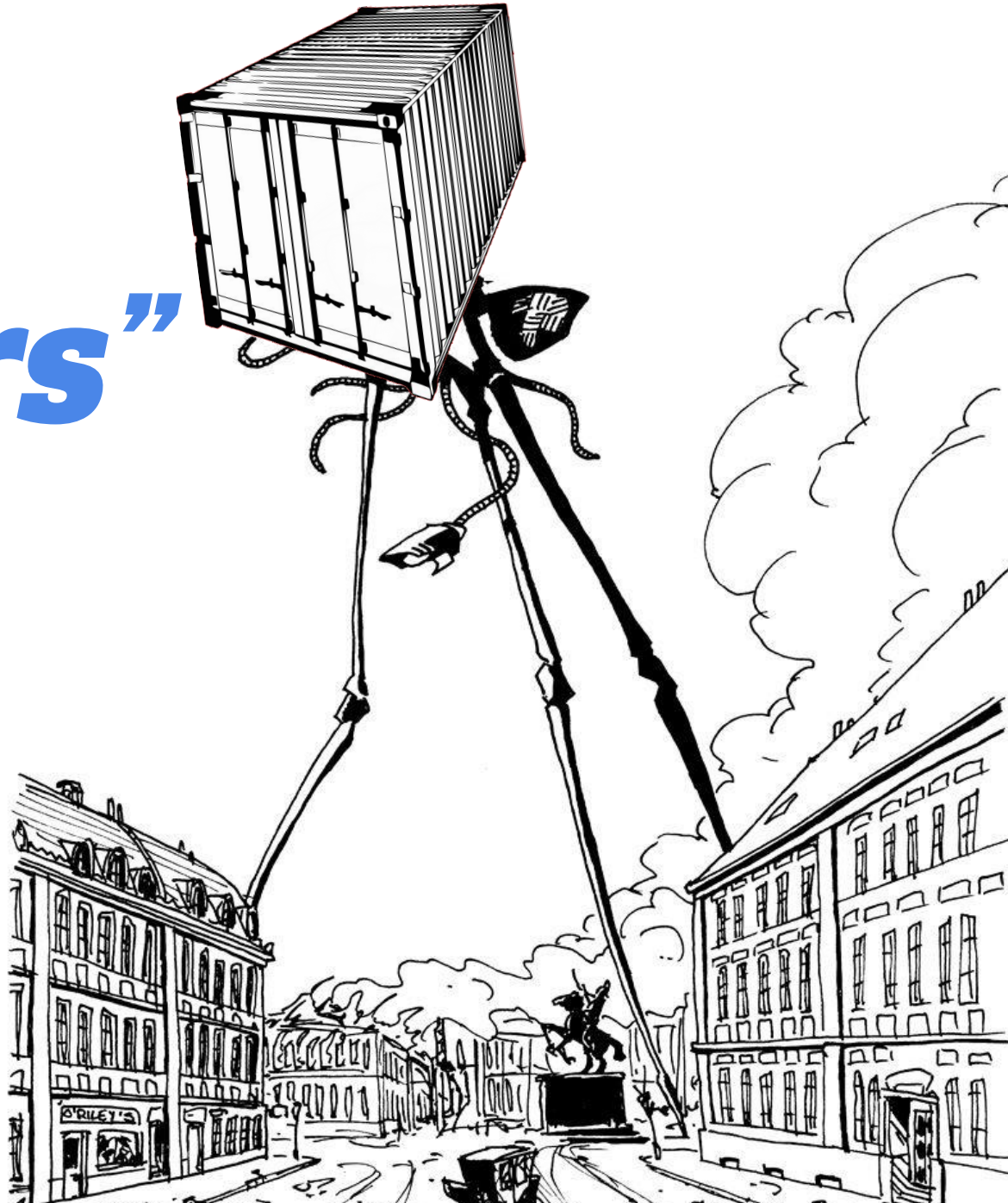
“The War of the Containers”

“One Container Orchestration System To Rule Them All”

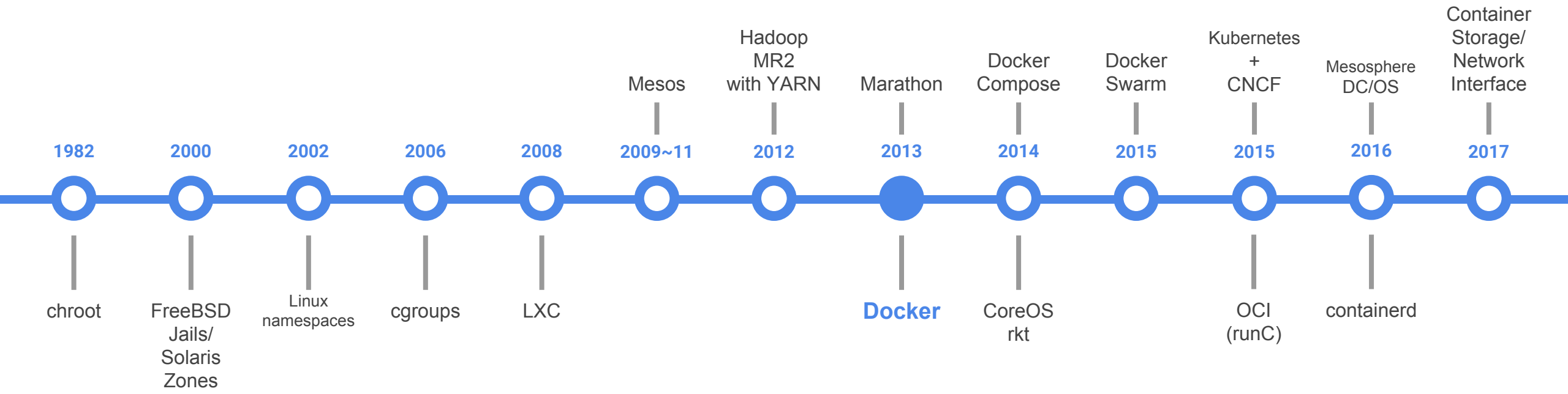
“Docker, Inc cheated on Swarm with K8S!”

“Which one is the best?”

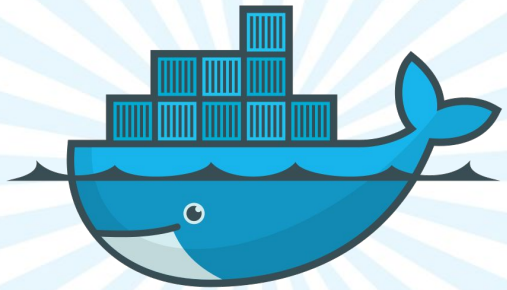
“There can be only one”



A brief history of Containers: from chroot to Kubernetes



The Docker revolution



Agnostic & self-sufficient single package

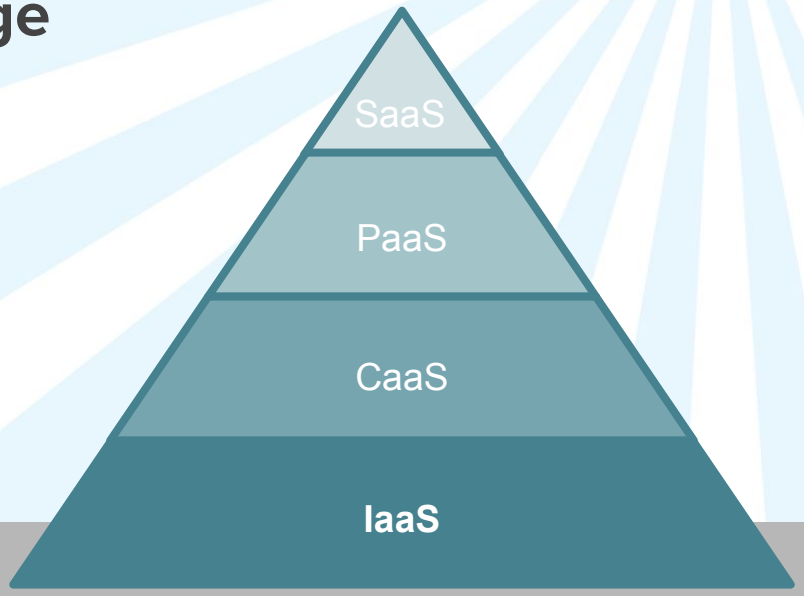
Deterministic app packaging

Complete information

SCM-like semantics

Portability

Image immutability & predictability



But needs additional tooling for container management on multiple hosts!

Workload heterogeneity

Batch jobs

One-off or time-scheduled (cron)

Short-lived

Analytics pipelines

MapReduce/Spark jobs

Machine learning

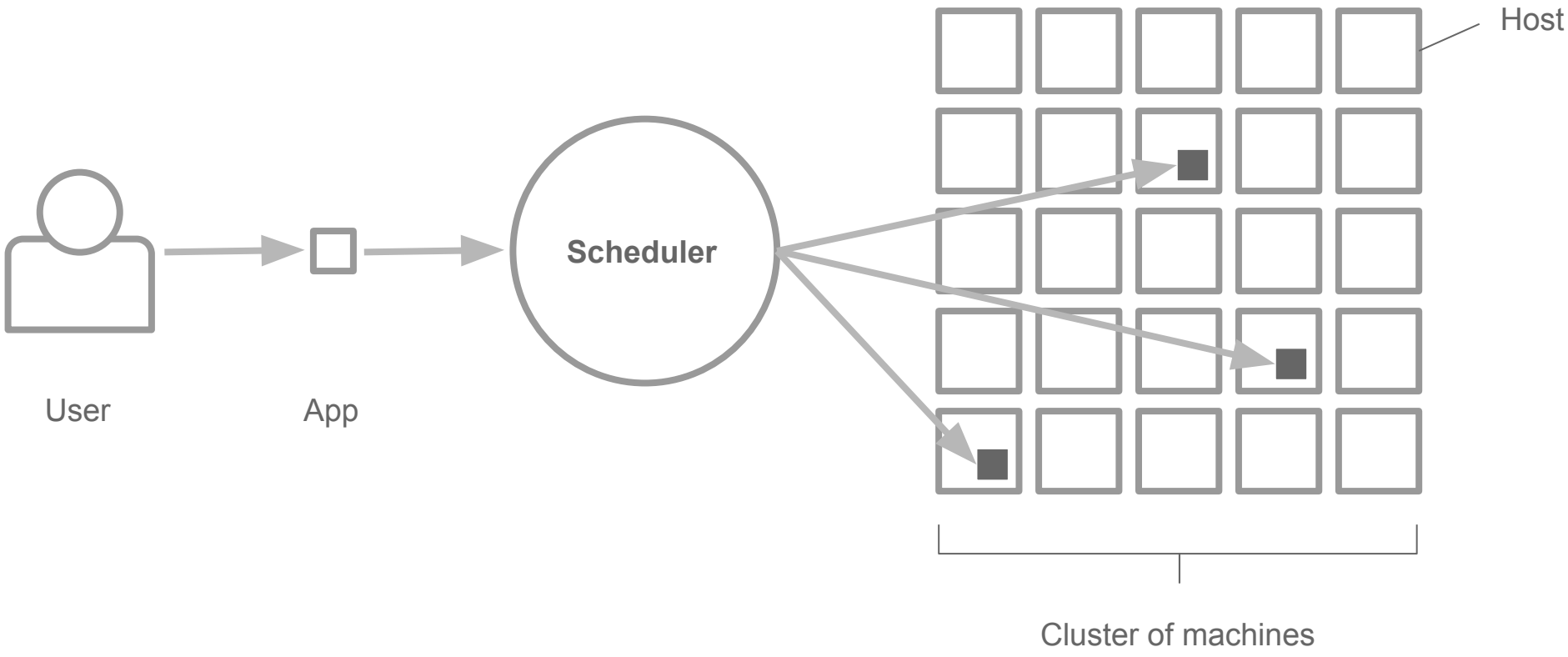
Average duration

Long-running services

Web services

Data services

Scheduling



Scheduling goals & requirements

1. Using the cluster resources efficiently
2. Working with user-supplied placement constraints
3. (Data locality)
4. Scheduling applications rapidly
5. Having a degree of *fairness* and/or business importance
6. Robustness & Availability

Design issues

Interference (concurrency):

- pessimistic approach: Ensure that a particular resource is only made available to one scheduler at a time
- optimistic approach (conflict detection)

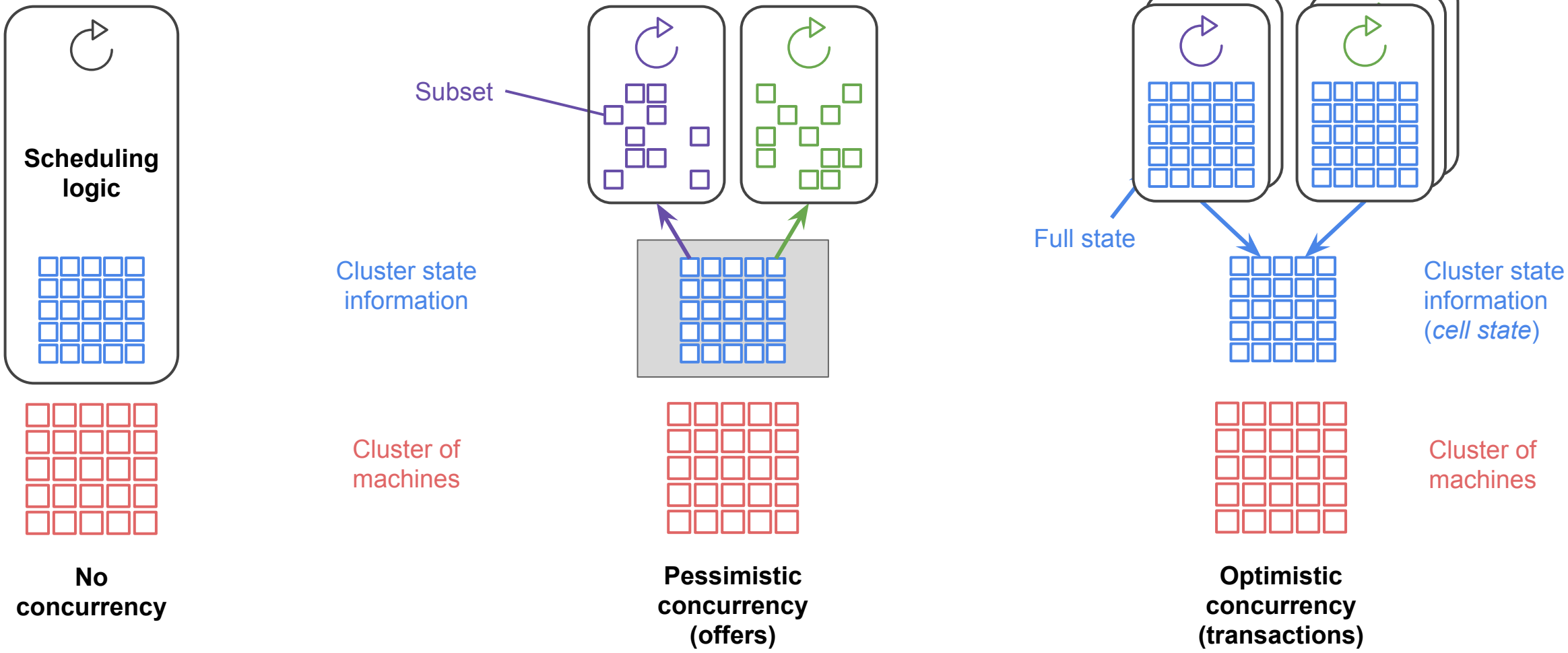
Allocation granularity:

- Atomic all-or-nothing gang scheduling / hoarding
- Incremental placement (MapReduce)

Cluster-wide behaviors:

- priority preemption
- strict fairness
- ...

Scheduling architectures



Comparison of scheduling approaches

Approach	Resource choice	Interference	Alloc. granularity	Cluster-wide policies
Monolithic	all available	none (serialized)	**global policy**	**strict priority (preemption)**
Statically partitioned	fixed subset	none (partitioned)	per-partition policy	scheduler-dependent
Two-level (Mesos)	dynamic subset	**pessimistic**	**hoarding**	**strict fairness**
Shared-state (Omega)	all available	**optimistic**	**per-scheduler policy**	**free-for-all, priority preemption**

Resource allocation

Mesos: *Dominant Resource Fairness* (DRF)

YARN: Capacity Scheduler / Fair Scheduler

Kubernetes:
Container resource requests / limits (cpu, memory)
Pod resources: sum of its containers' resource requests / limits
"The scheduler ensures that, for each resource type, the sum of the resource requests of the scheduled Containers is less than the capacity of the node"

Swarm strategies for ranking nodes:

- *Spread:* Node with the least number of containers
- Binpack: Node which is most packed

Nomad: bin packing which "optimize the resource utilization and density of applications"

Features 

Declarative configuration: blue print (*desired state*)

Rules & Constraints (affinity)

Provisioning (OSB)

Rolling updates

Scaling

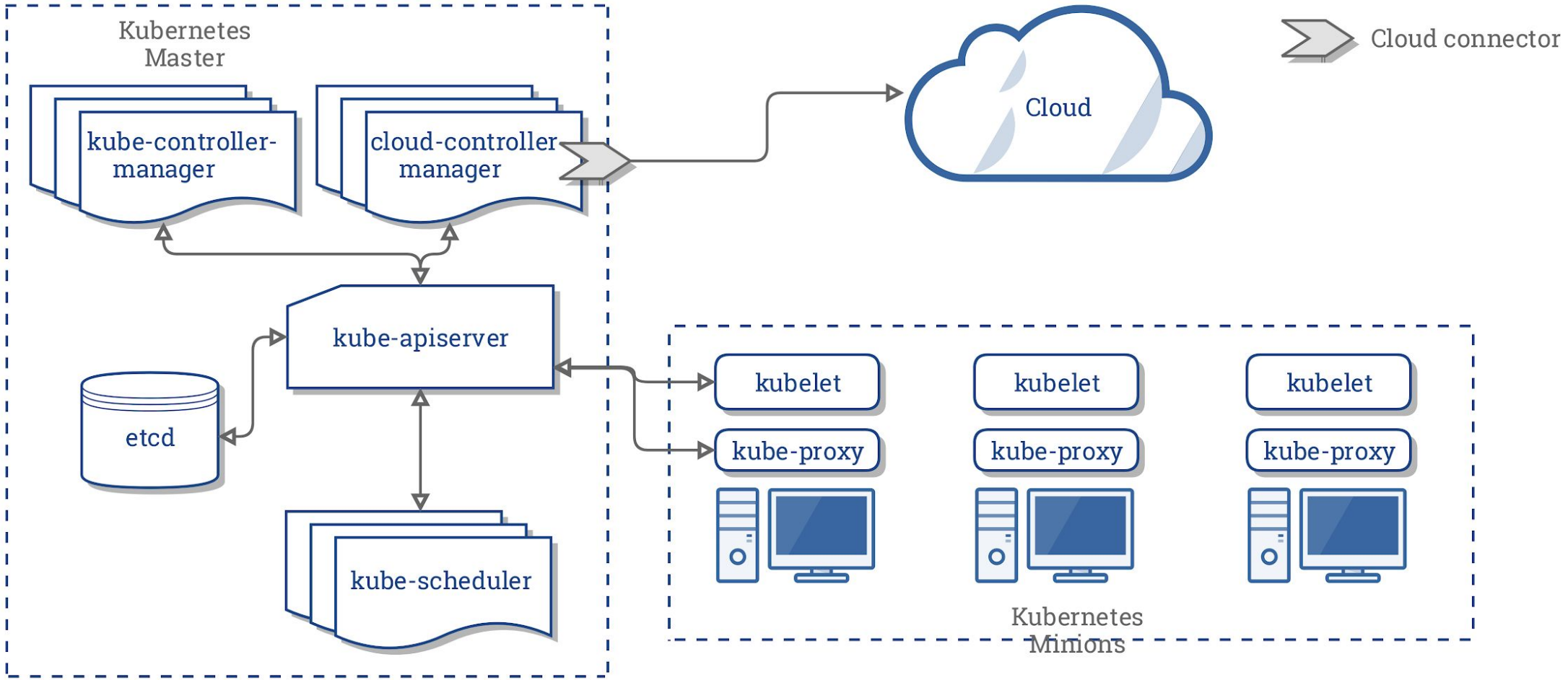
Service Discovery

Load balancing

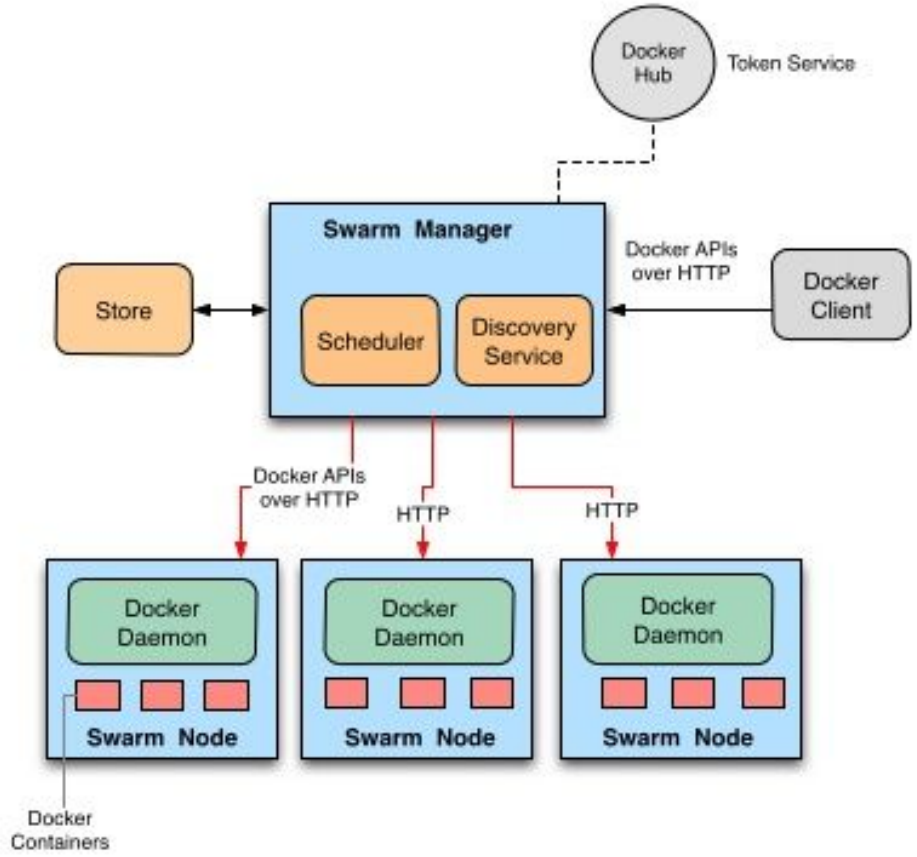
Health check & Monitoring (*desired state reconciliation*)

Persistent storage (stateful applications)

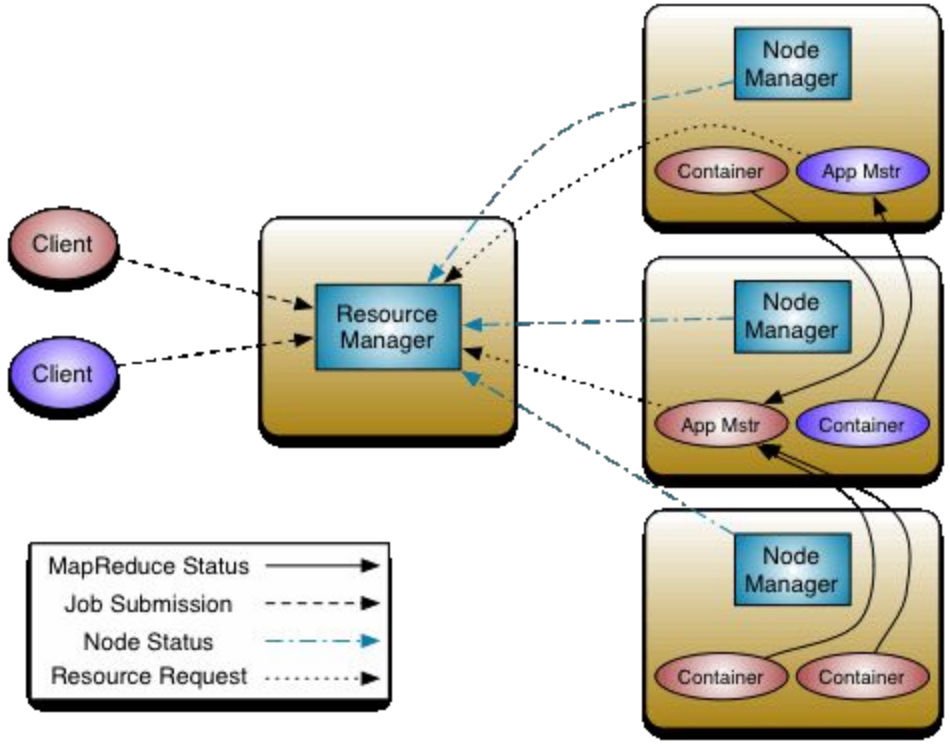
Kubernetes (Monolithic)



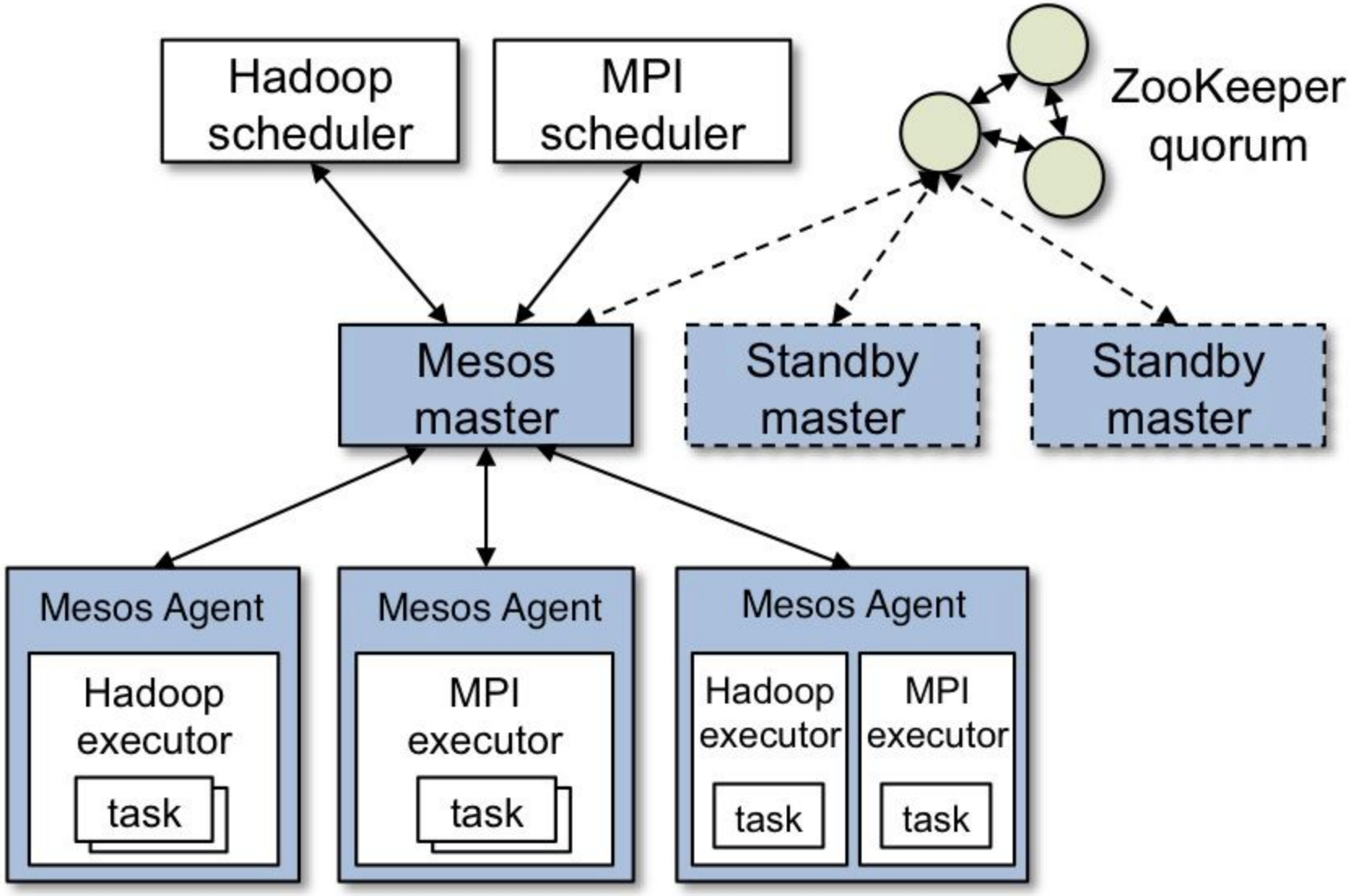
Swarm (Monolithic)



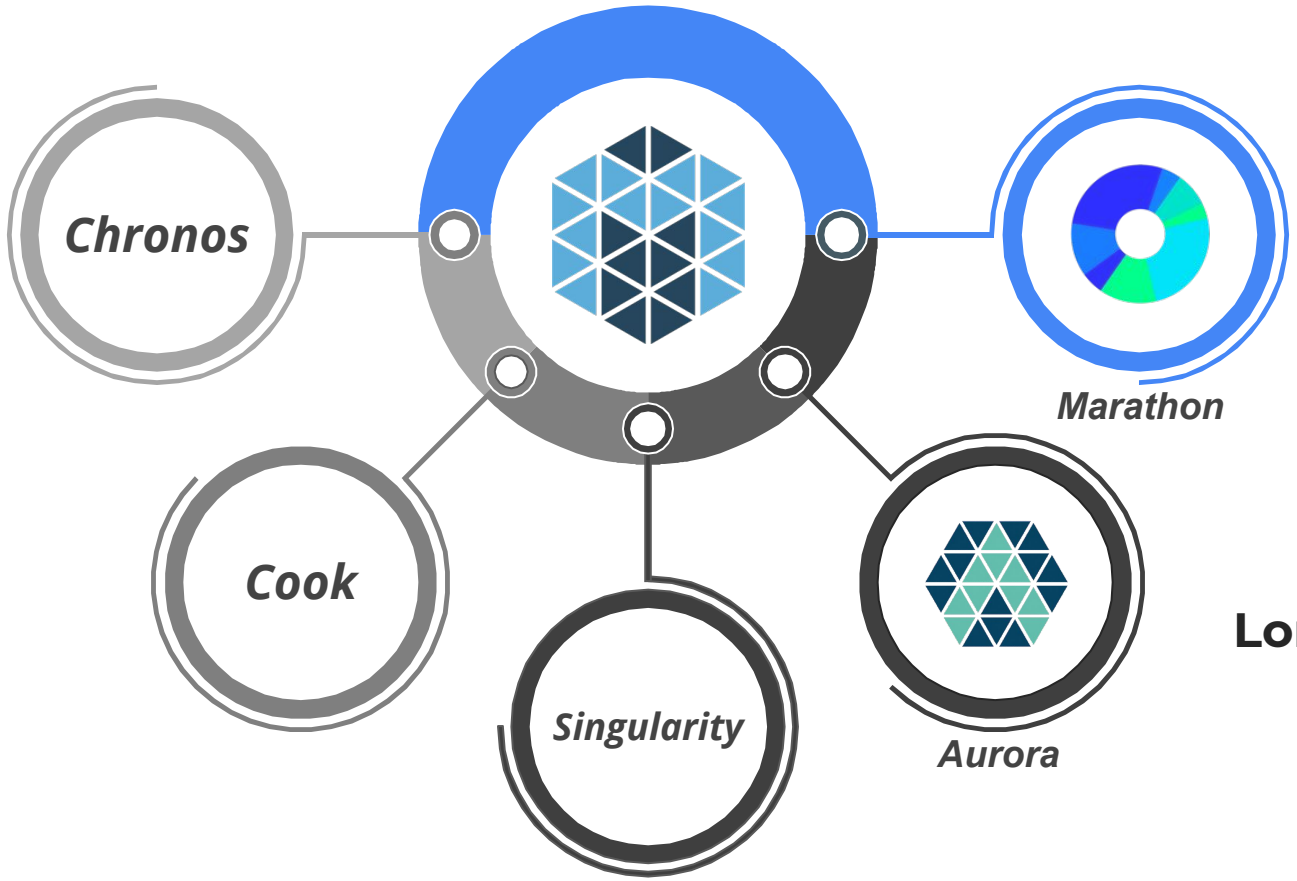
YARN (Monolithic)



Mesos (two-level)



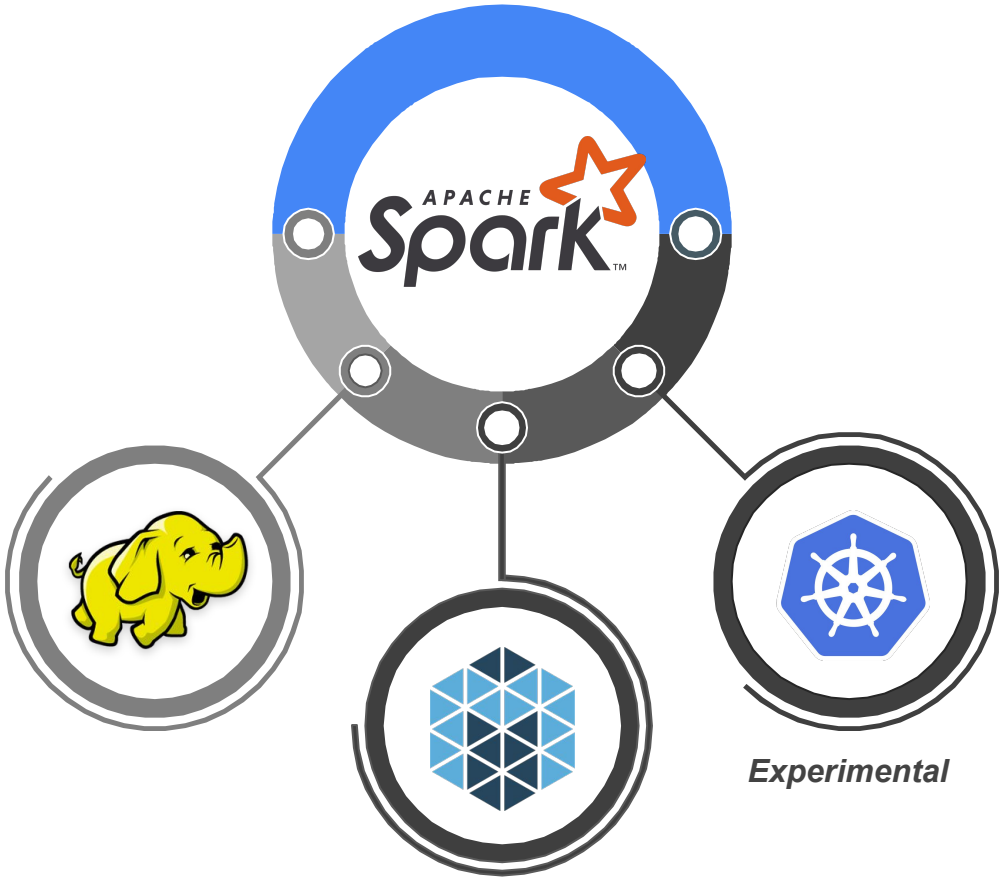
Mesos scheduling frameworks



***Batch* jobs**

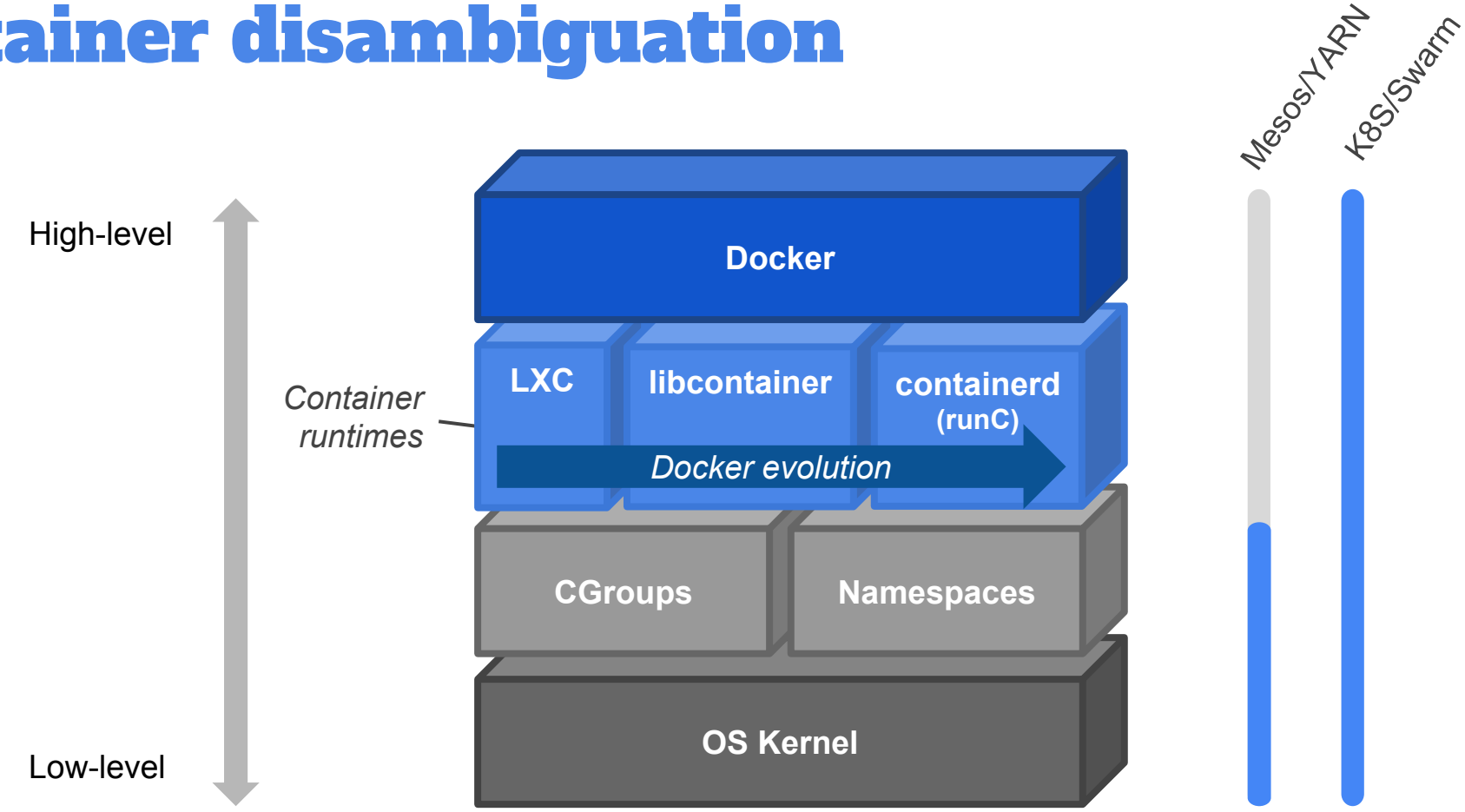
Long-running services

Aparté on Spark



Comparison in terms of Component Architecture

Container disambiguation



"Without CGroups, it becomes hard to limit container CPU usage" - YARN

Paradigm shift in application deployment

Managed containerization

The cluster manager is responsible for containerizing the application

Standalone applications: Shell scripts, Java

Agent's working directory for dependencies & native libraries:

- Mesos sandbox
- YARN LocalResources

Self-containerized applications

The cluster manager supports the application's container format and run it directly

Docker

Stateful services

Persistent volumes
enable stateful services

Motivation & current limitations

Volume management is tightly coupled to the COs: adding support to new storage systems requires adding code into the core COs codebase

No well-defined interface allowing third-party storage vendors to plug into COs

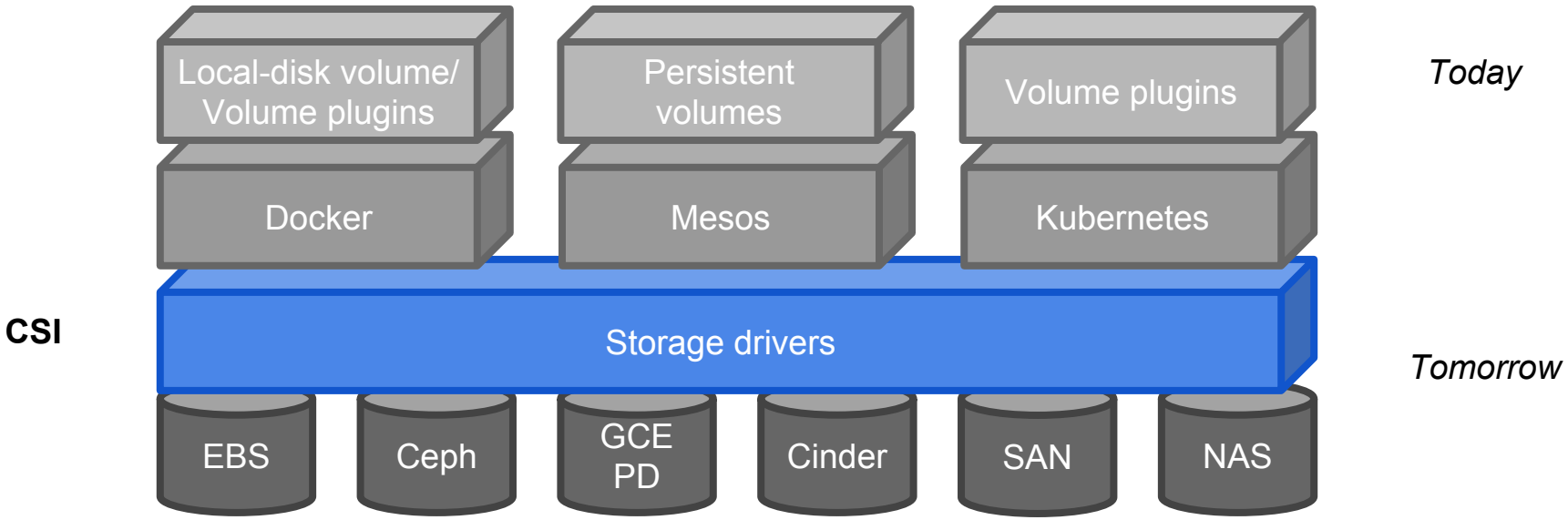
Container Storage Interface (CSI)

Specification that defines a common set of APIs for all interactions between storage vendors and container orchestration platforms

Close collaboration among Kubernetes, CloudFoundry, Docker and Mesos communities

Primary goal: establish a standardized mechanism for COs to expose arbitrary storage systems to their (containerized) workloads, using a consistent API that decouples the release cycle of COs from that of the storage systems, making the integration itself more sustainable and maintainable

Container Storage Interface

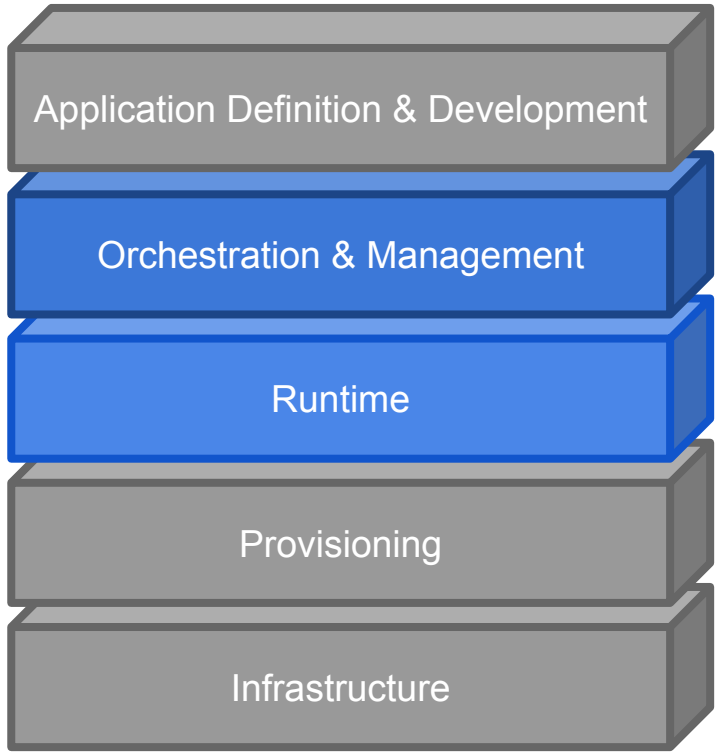


Today

CSI

Tomorrow

Cloud Native Reference Stack



Observability

- View / Filter / Replay
- Monitoring / Trace / Stream / Log
- Business Intelligence

Orchestration and scheduling

Name resolution and service discovery (e.g., DNS)

Service Management

- Routing / Proxy / Load Balancer
- Policy / Placement / Traffic Management

Resource Management

- Image Management
- Container Management
- Compute Resources

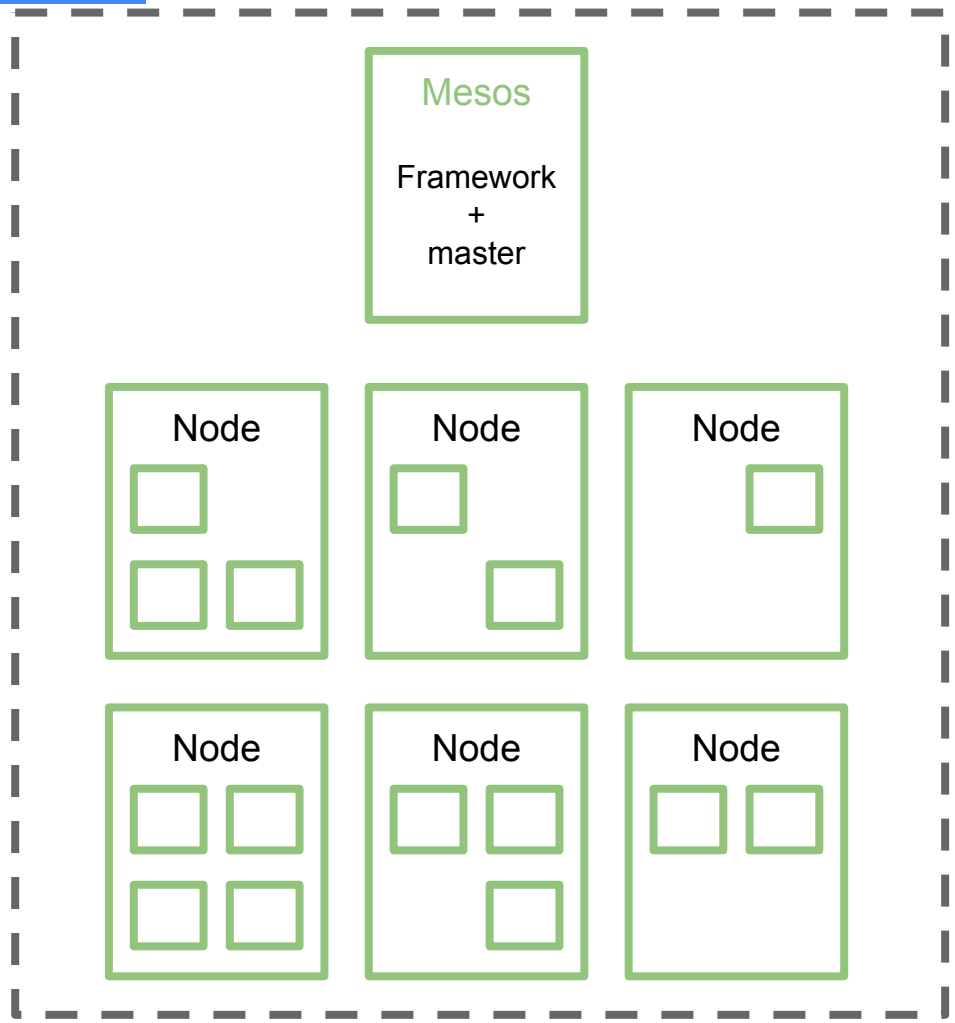
Cloud Native – Network

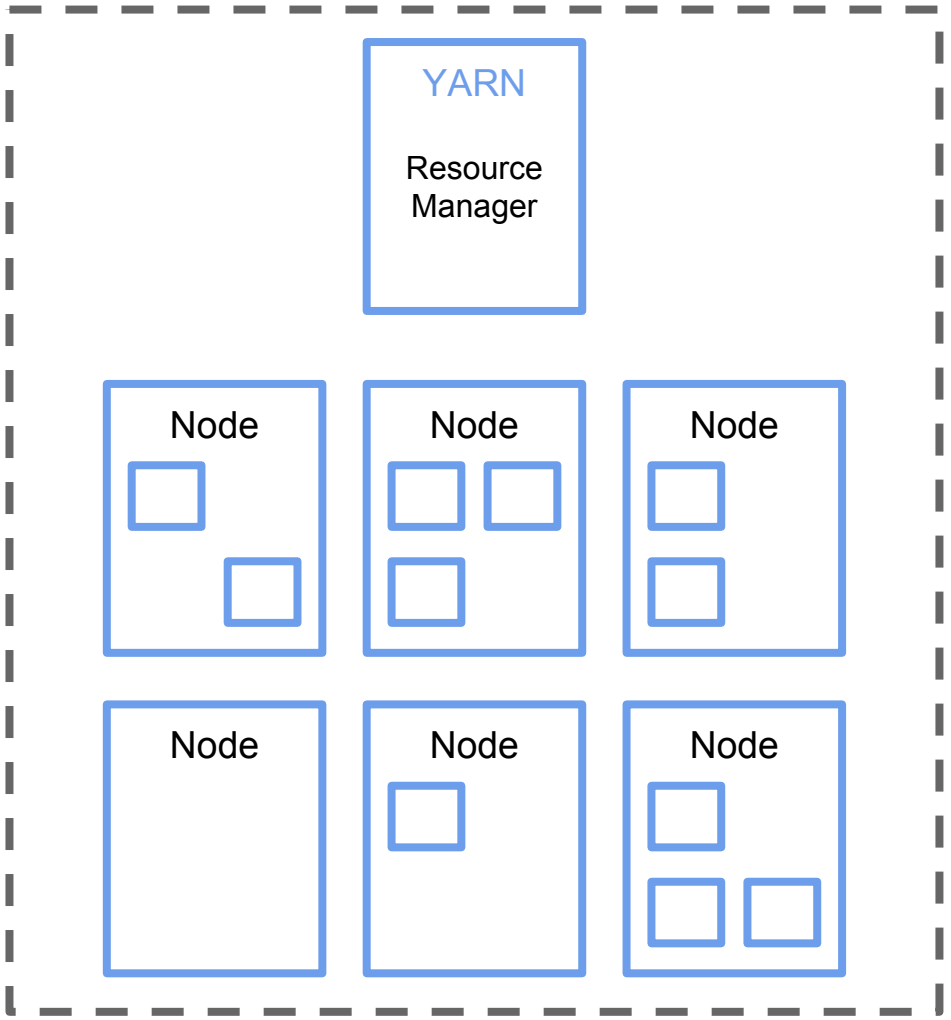
- Network Segmentation and Policy
- SDN & APIs (e.g., CNI, libnetwork)

Cloud Native- Storage

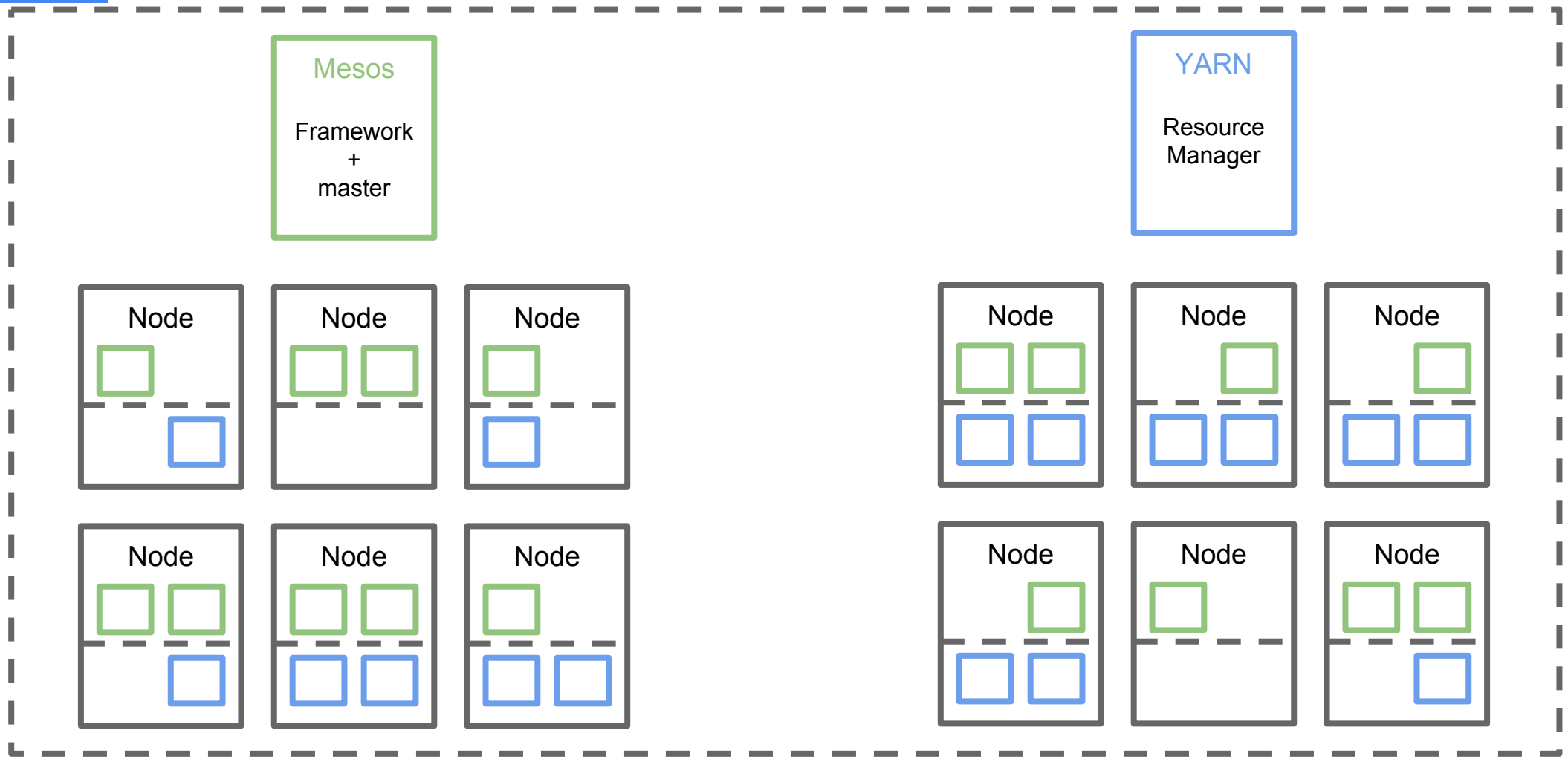
- Volume Drivers/Plugins
- Local Storage Management
- Remote Storage Access

The static partition issue: siloed clusters

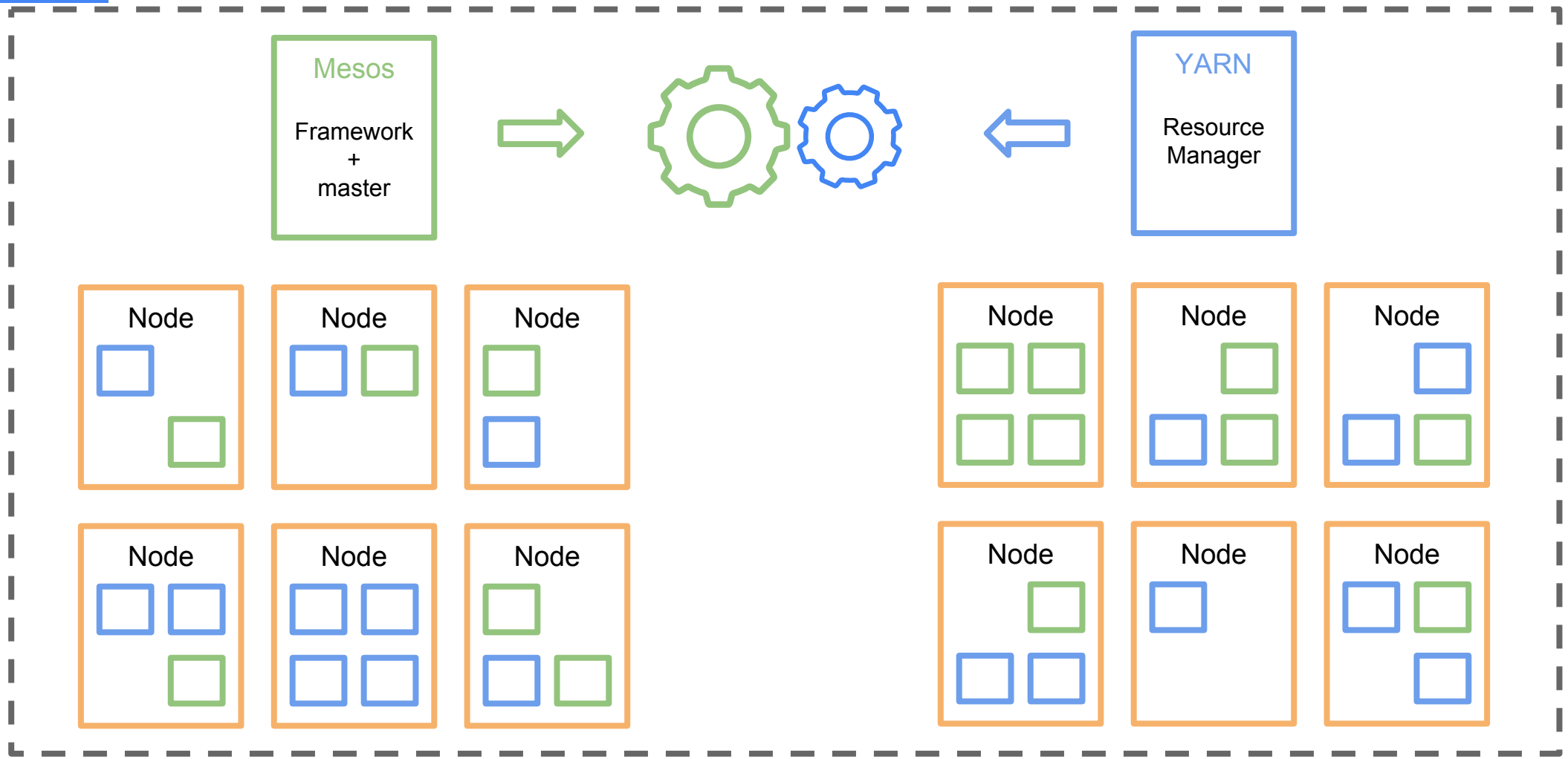




The static partition issue: node resource partitioning



Sharing a single pool of resources



Integrations

Original k8s on Mesos framework has been retired
New Mesosphere MKE (Mesosphere Kubernetes Engine):

- Not open source
- Only runs on Mesosphere DC/OS

More like a k8s installer through Mesos

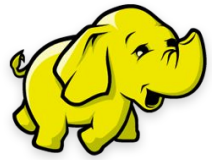
Kubernetes

Mesos

Running Mesos on Hadoop MRv1

Apache Myriad

YARN



Hadoop

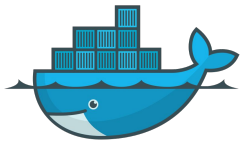
Mesos as backend for Swarm

Seamless integration of Kubernetes into the Docker platform, offering users the choice to use Kubernetes and/or Docker Swarm for orchestration

Swarm

Mesos supports Docker and Mesos containers.
Marathon framework is used for container orchestration.
As of version 1.4, Marathon supports the creation and management of *pods*.

Launching YARN applications using Docker containers
Experimental

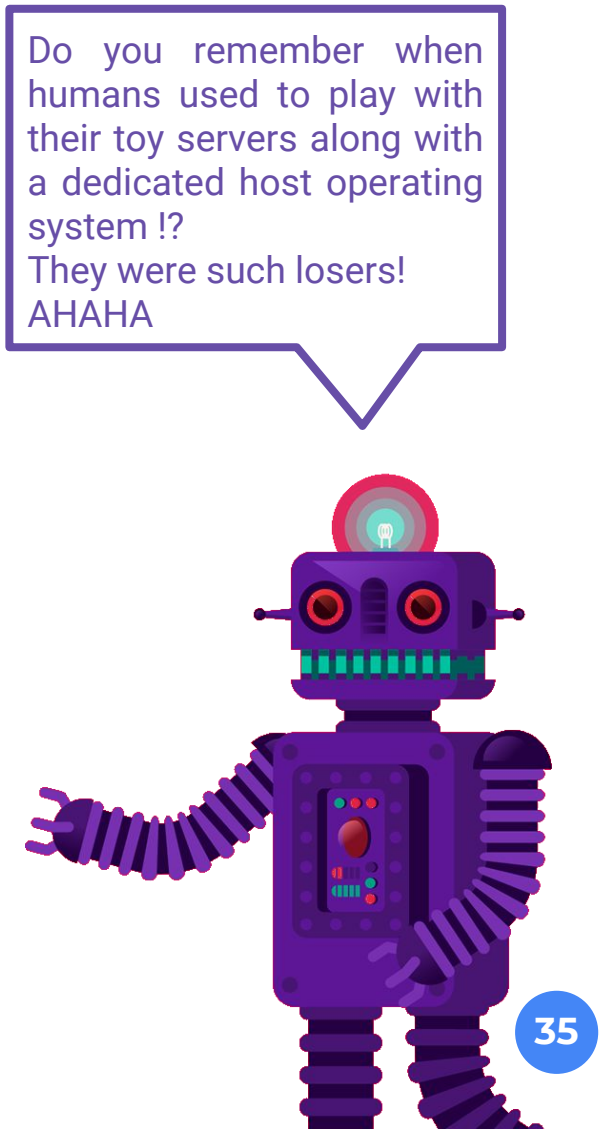


Docker

The server disaggregation

Operating system with minimal functionality required for deploying applications inside containers

Distributed operating system based on Apache Mesos



Conclusion

No unique solution to solve every problems with cluster computing

Developers & DevOps like to change tools and want to replace them easily

Vendors need to create multiple integrations to be supported across the container ecosystem

Propose an architecture that gives control to the developers

CNCF, OCI & OSB standards

Thanks

Would You Like To Know More?

Malte Schwarzkopf, Andy Konwinski, Michael Abd-El-Malek, John Wilkes, *Omega: flexible, scalable schedulers for large compute clusters* (2013)

Abhishek Verma, Luis Pedrosa, Madhukar Korupolu, David Oppenheimer, Eric Tune, John Wilkes, *Large-scale cluster management at Google with Borg* (2015)

A.Ghodsi, M. Zaharia, B. Hindman, A. Konwinski, S. Shenker, I. Stoica, *Dominant resource fairness: fair allocation of multiple resource types* (2011)

Mesos Architecture providing an overview of Mesos concepts:
http://mesos.apache.org/documentation/latest/architecture/