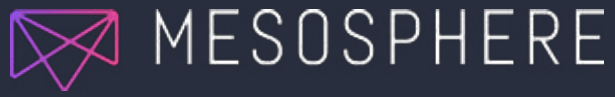
Microservices on DC/OS and Container Orchestration on Mesos

Gilbert Song(宋子豪)



Who am I

- Apache Mesos PMC, Committer
- Mesosphere Distributed Systems Engineer
- M.S. of Computer Engineering from University of California, Santa Barbara
- Focus on Mesos Containerization
- Passionate about Cloud Computing and Distributed Systems

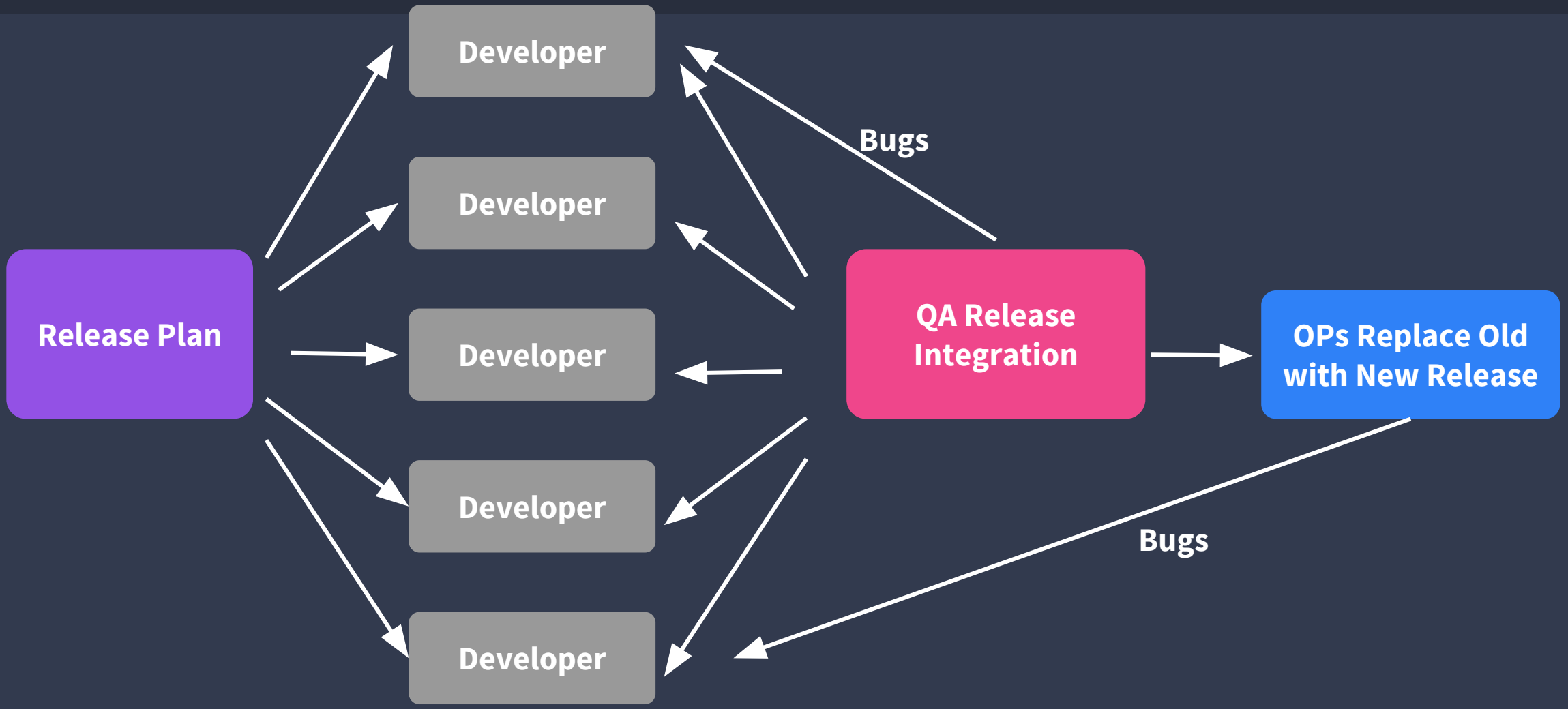
E-mail: gilbert@apache.org

WeChat: songzihao888358

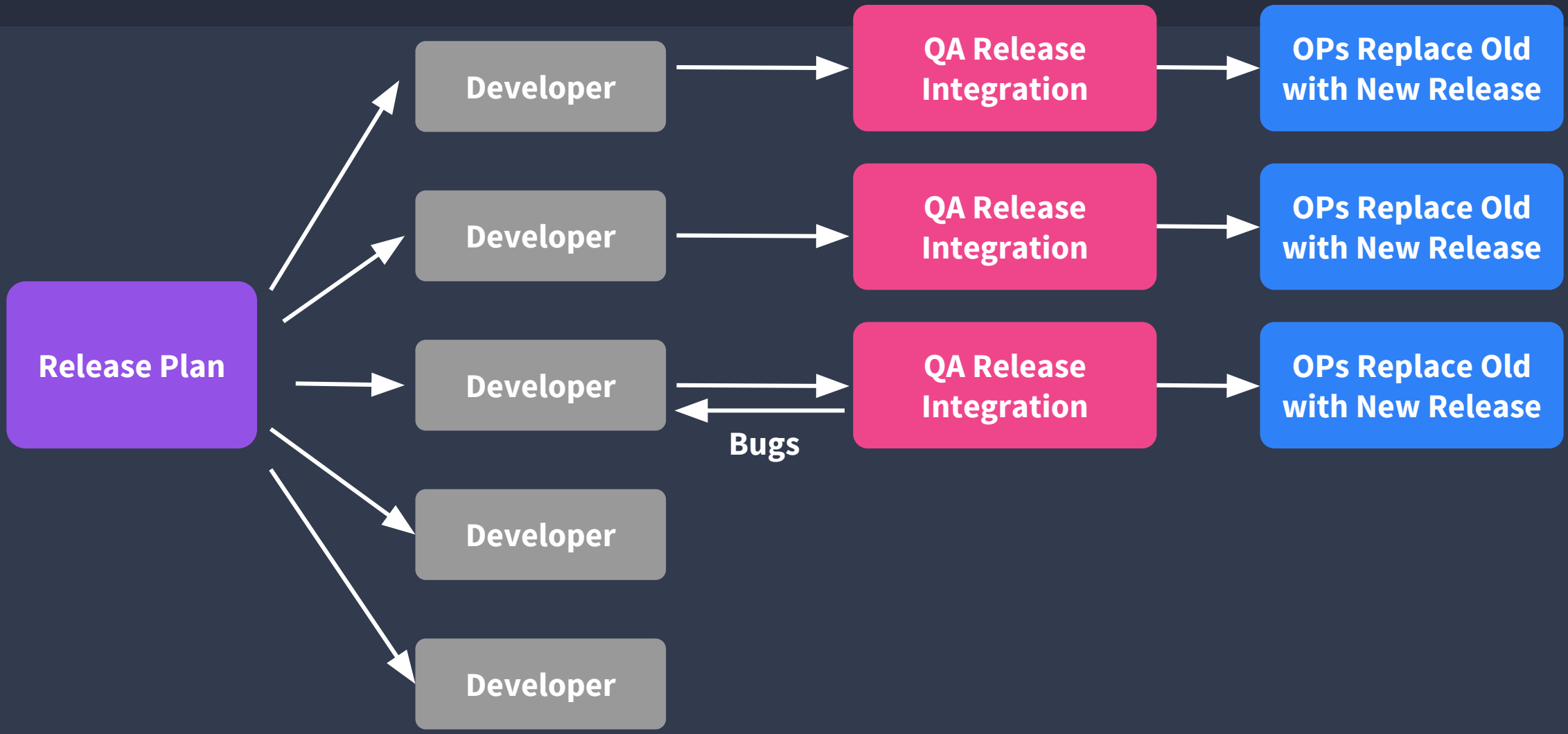
Outline

- From monolithic to microservices
- Microservices on Mesosphere DC/OS
- The architecture of DC/OS
- Apache Mesos overview and fundamentals
- Container standards/specifications supported by Mesos
- Container Orchestration on Apache Mesos
- Why should I pick Mesos
- Latest features

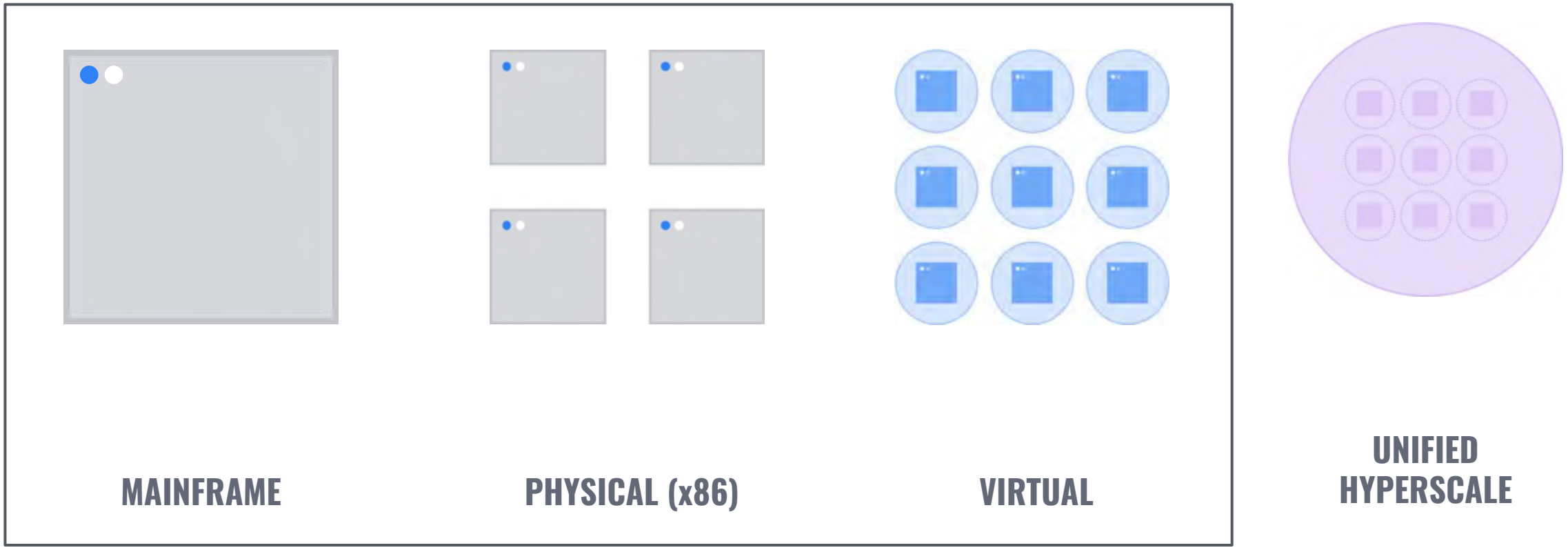
From monolithic to microservices



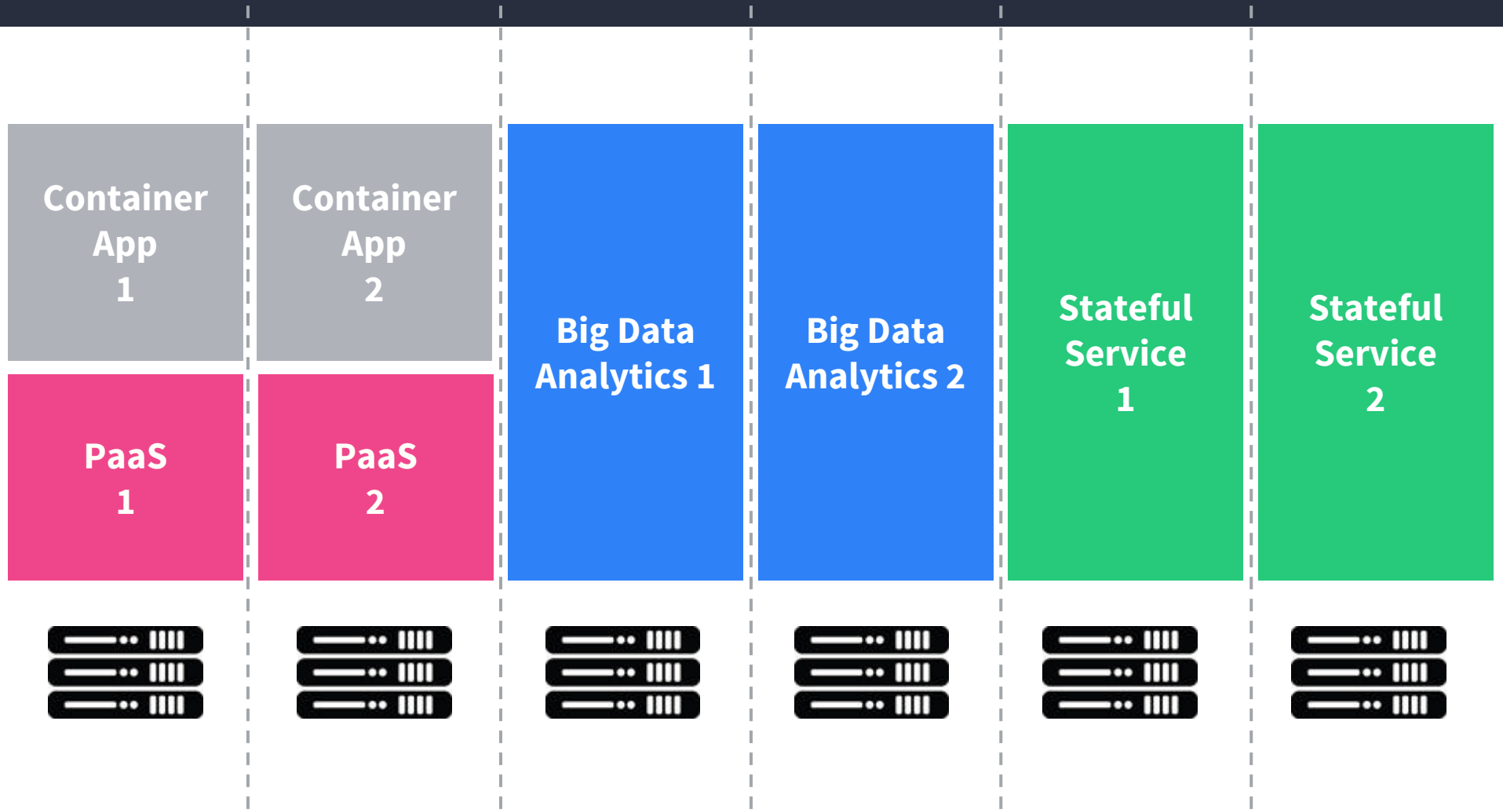
From monolithic to microservices



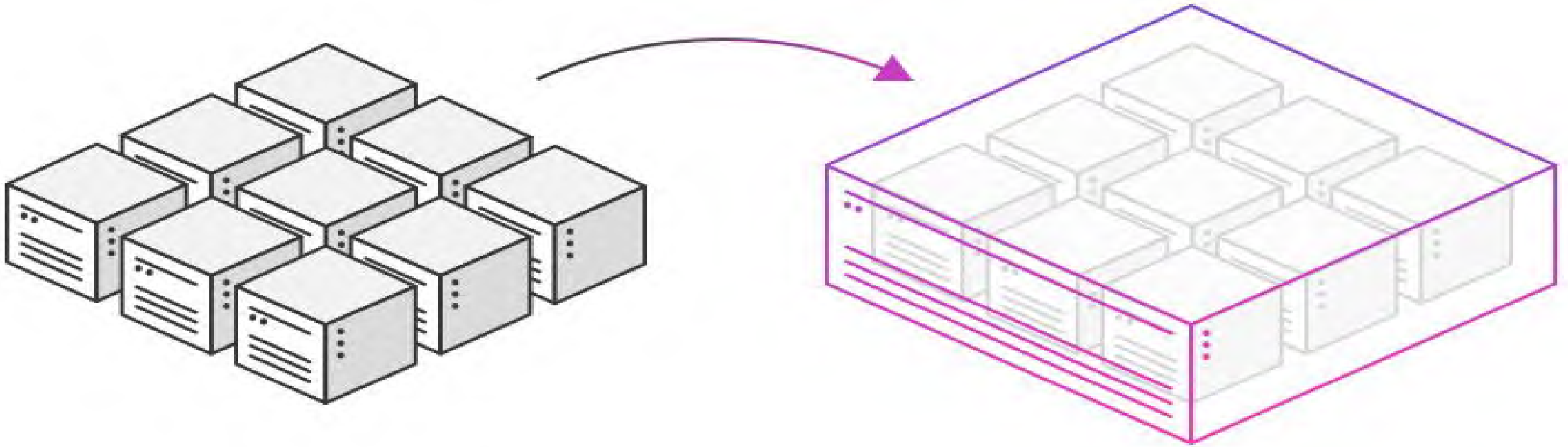
THE HISTORY OF INFRASTRUCTURE



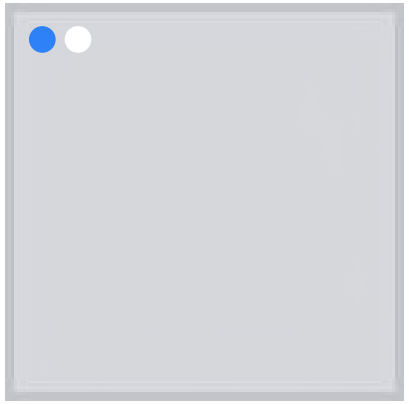
TRADITIONAL IT APPROACH



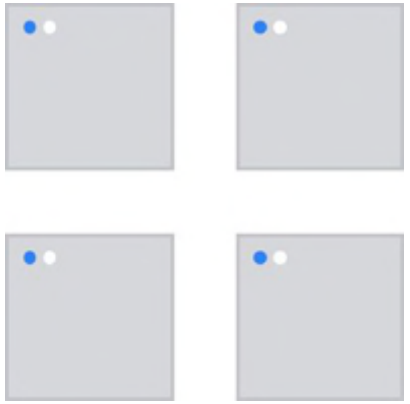
INFRASTRUCTURE EVOLUTION



THE NEXT WAVE OF COMPUTING

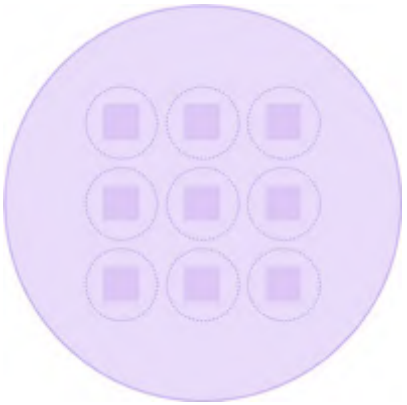


MAINFRAME



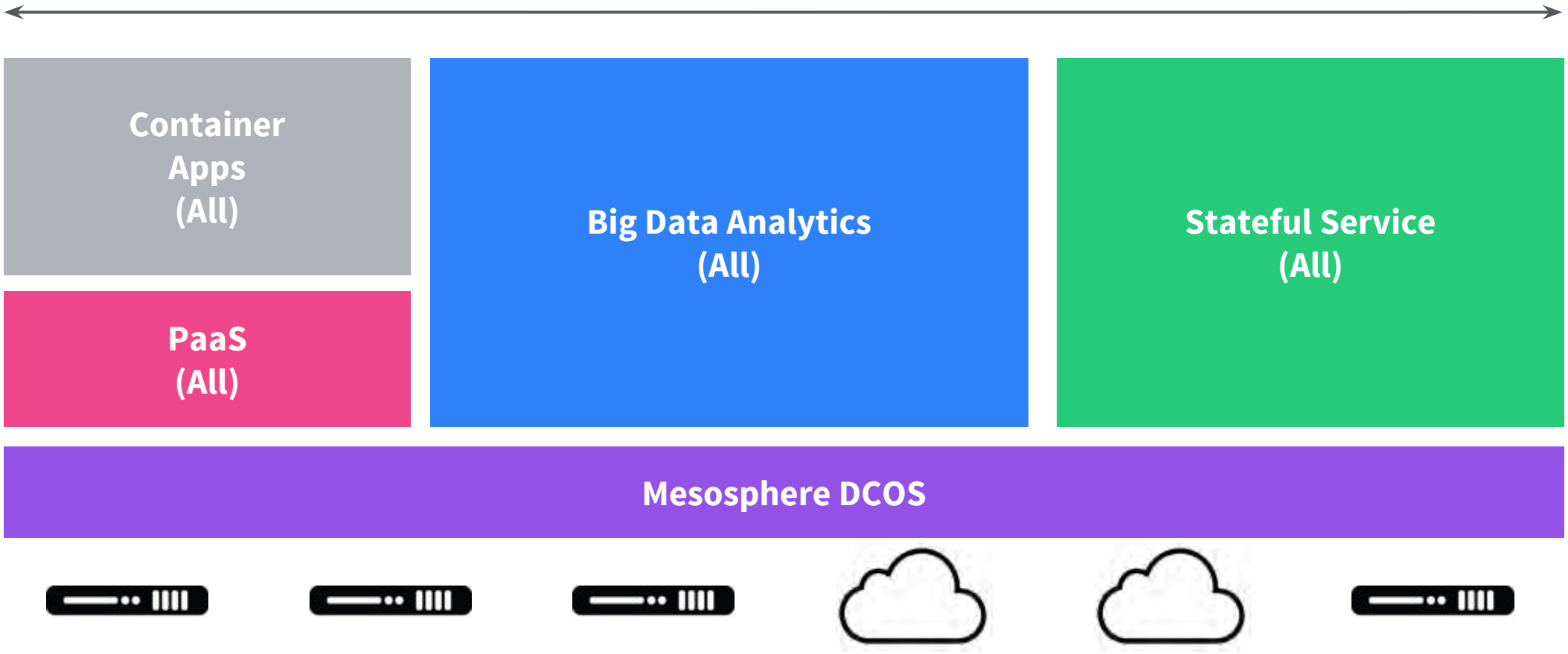
PHYSICAL (x86)

VIRTUAL

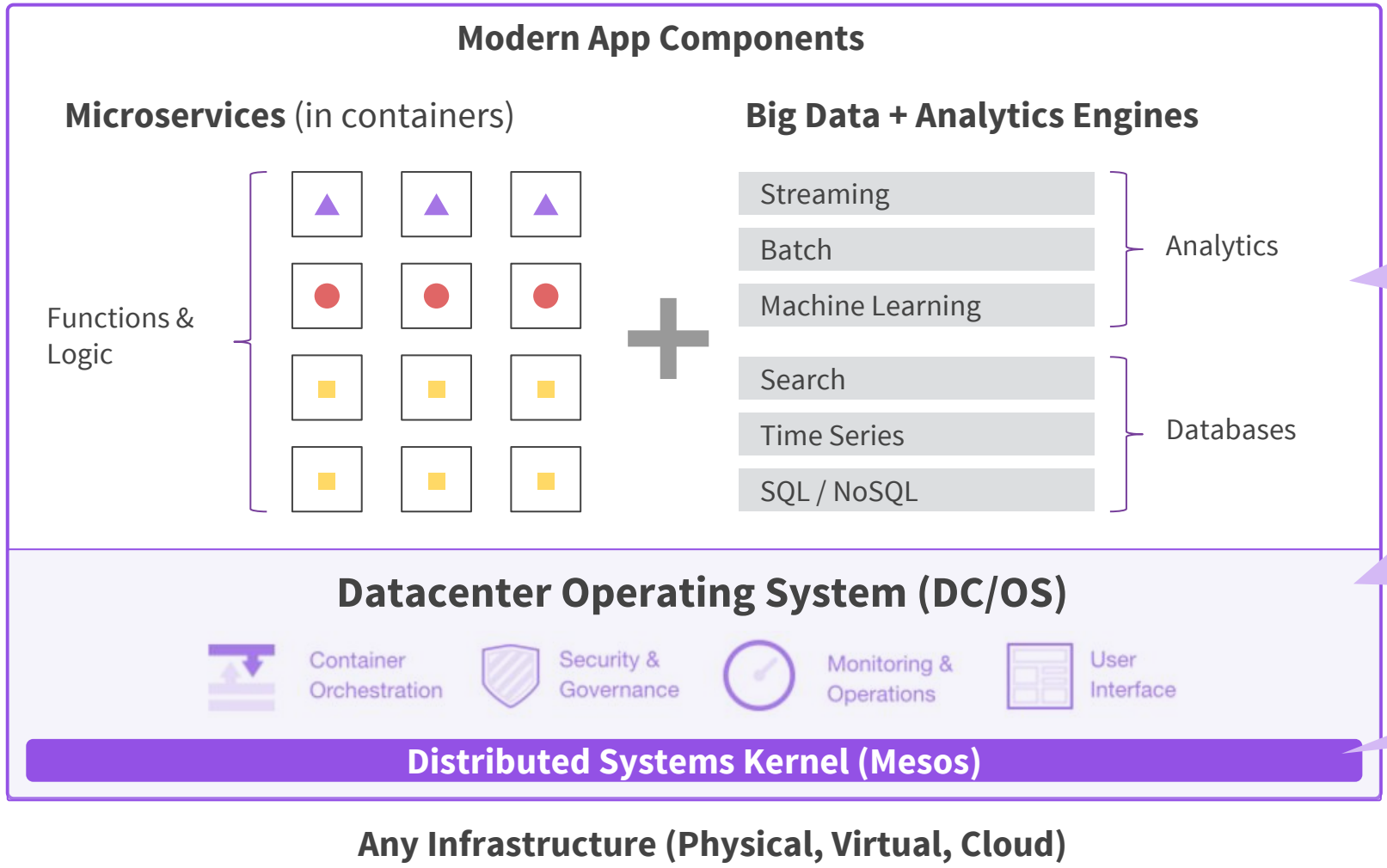


UNIFIED HYPERSCALE

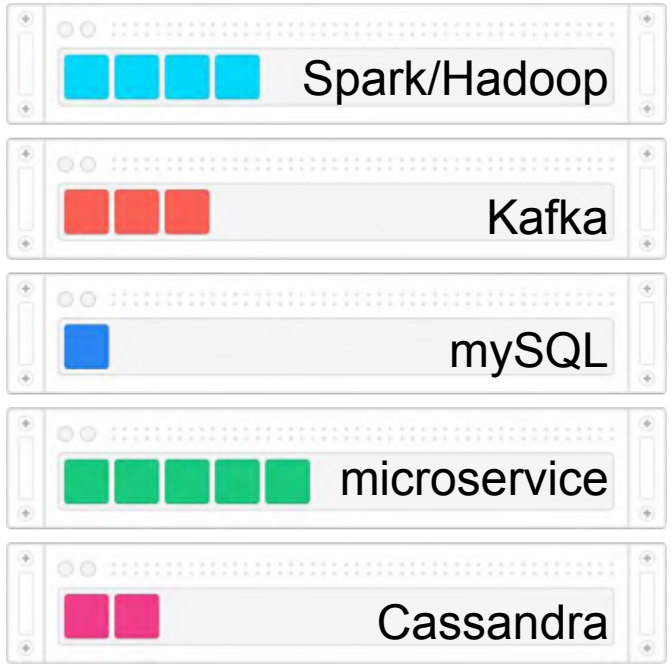
THE MESOSPHERE DC/OS APPROACH



DC/OS ENABLES MODERN DISTRIBUTED APPS



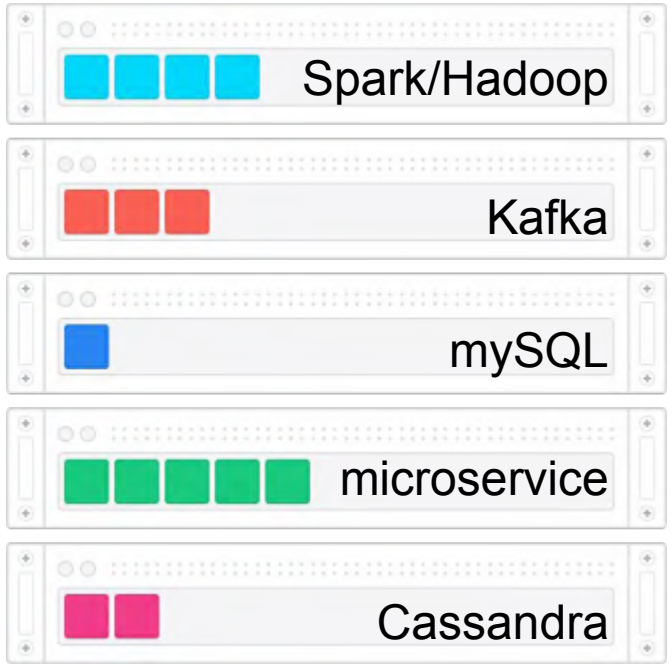
SILOED WORKLOADS



Industry Average
12-15% utilization

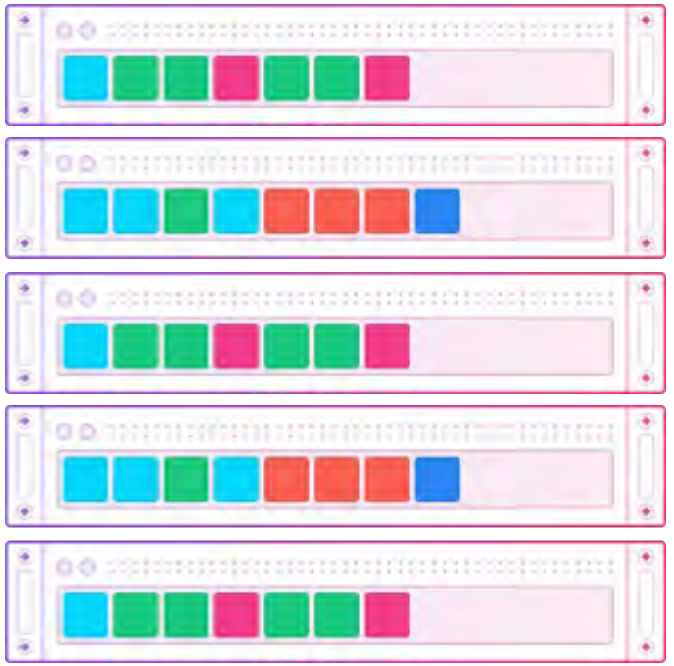
Typical Datacenter
siloed, over-provisioned servers,
low utilization

DC/OS MULTIPLEXING



Industry Average
12-15% utilization

Typical Datacenter
siloed, over-provisioned servers, low utilization



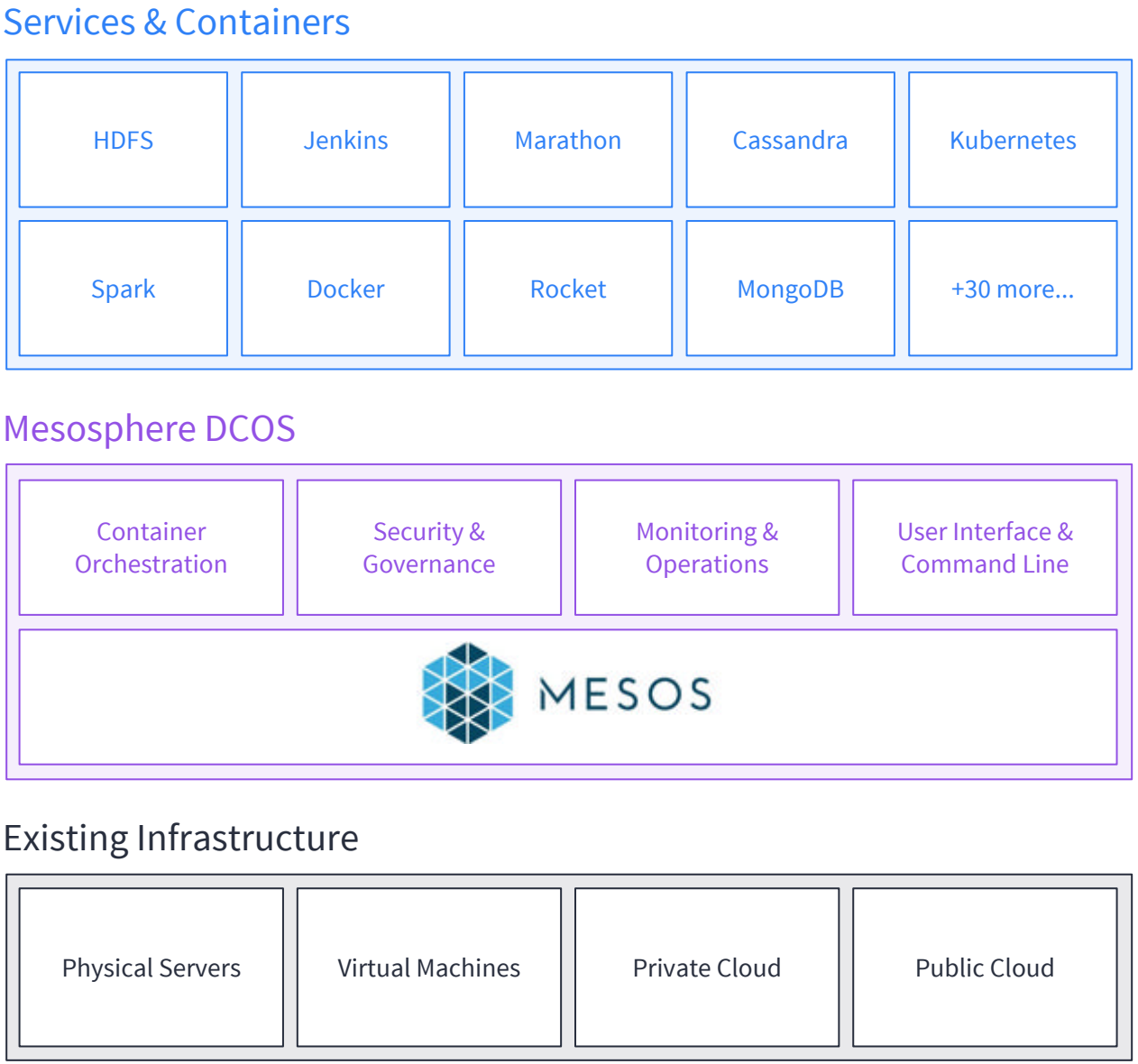
DC/OS Multiplexing
30-40% utilization, up to 96% at some customers

4X

DC/OS Datacenter
automated schedulers, workload multiplexing onto the same machines

Overview

MESOSPHERE DC/OS Architecture



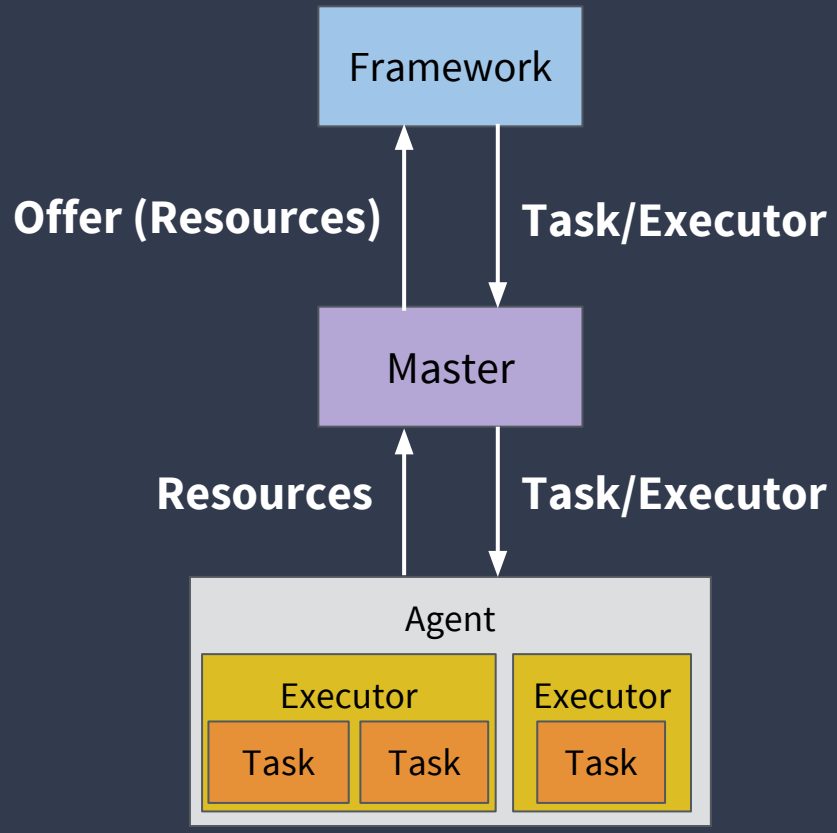
Mesos: A kernel for data center applications

- What does a traditional OS kernel provide?
  - Resource management — Host cpu, memory, etc.
  - Programming abstractions — POSIX API: processes, threads, etc.
  - Security and isolation — Virtual memory, user, etc.

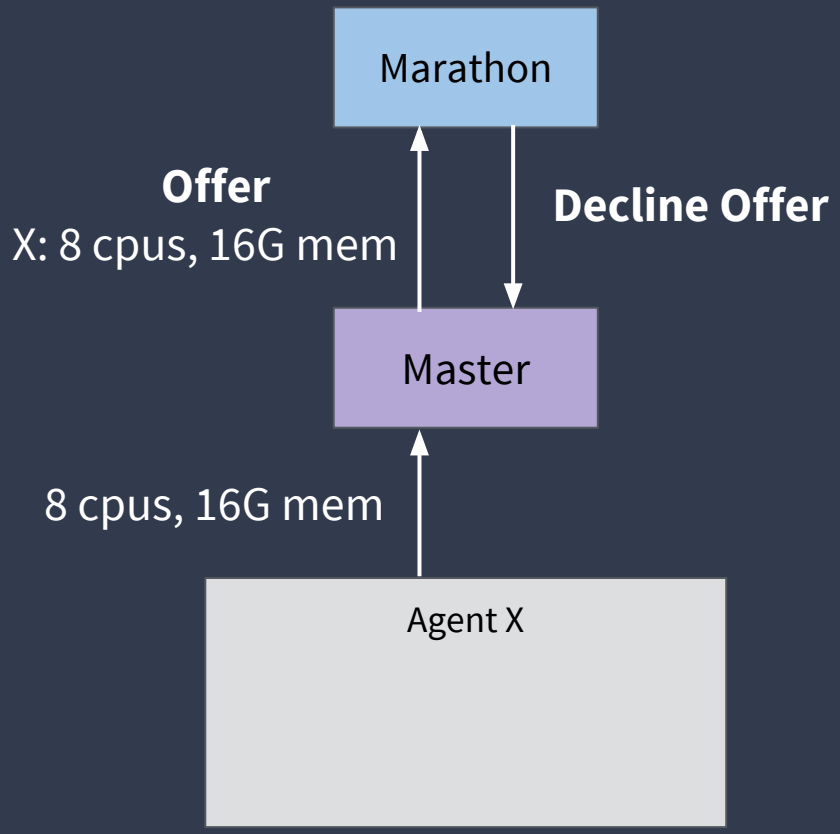
- Mesos: A kernel for data center applications
  - Resource management — Cluster cpu, memory, etc.
  - Programming abstractions — Mesos API: Task, Resource, etc.
  - Security and isolation — Containerization

Programming abstractions

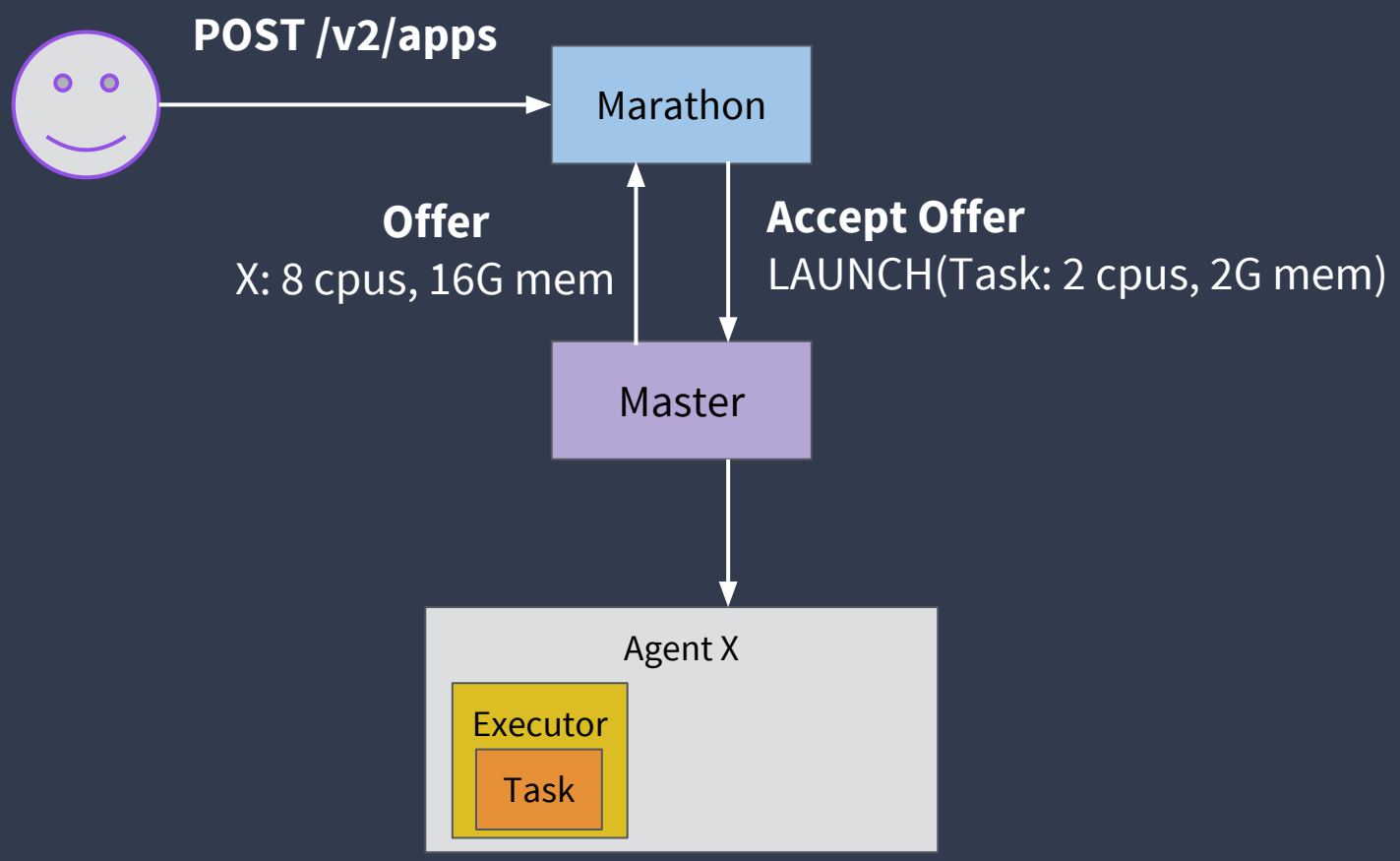
- Key concepts
  - Framework
  - Resource/Offer
  - Task
  - Executor



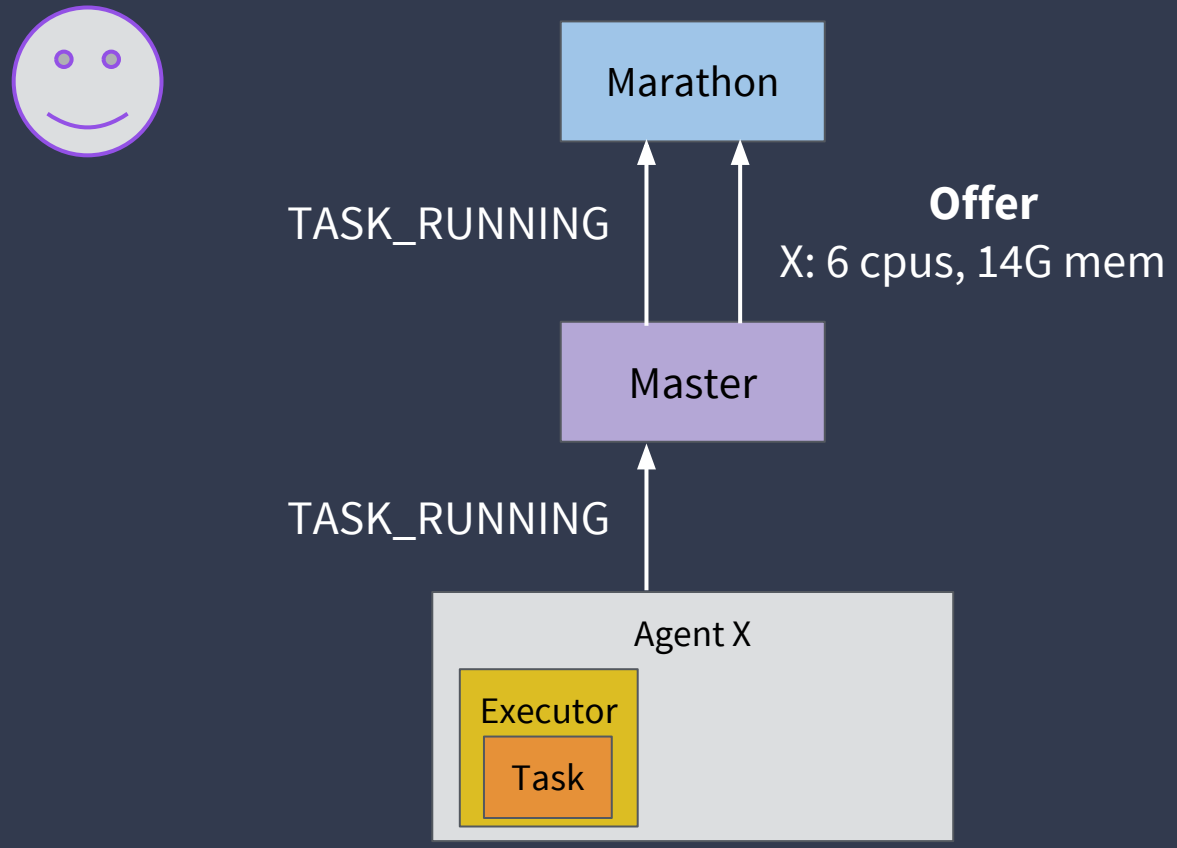
Case study: Marathon



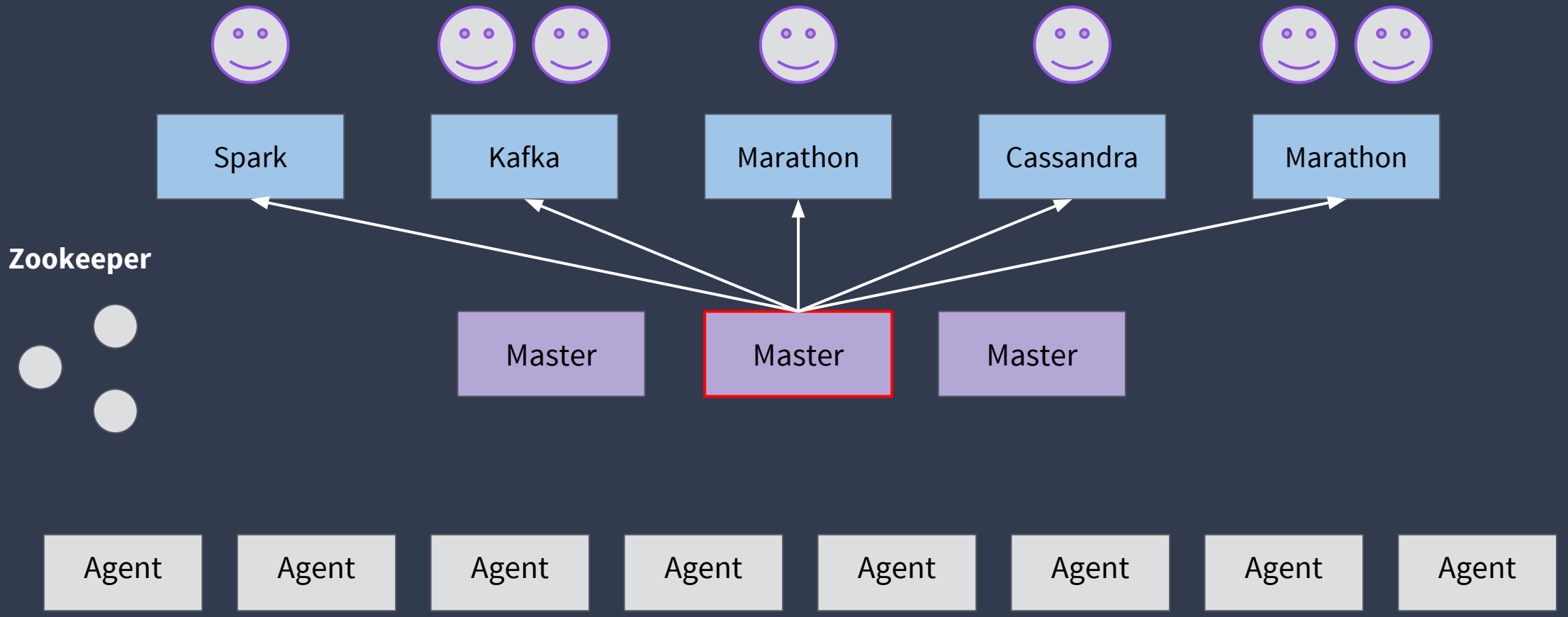
Create a Marathon app



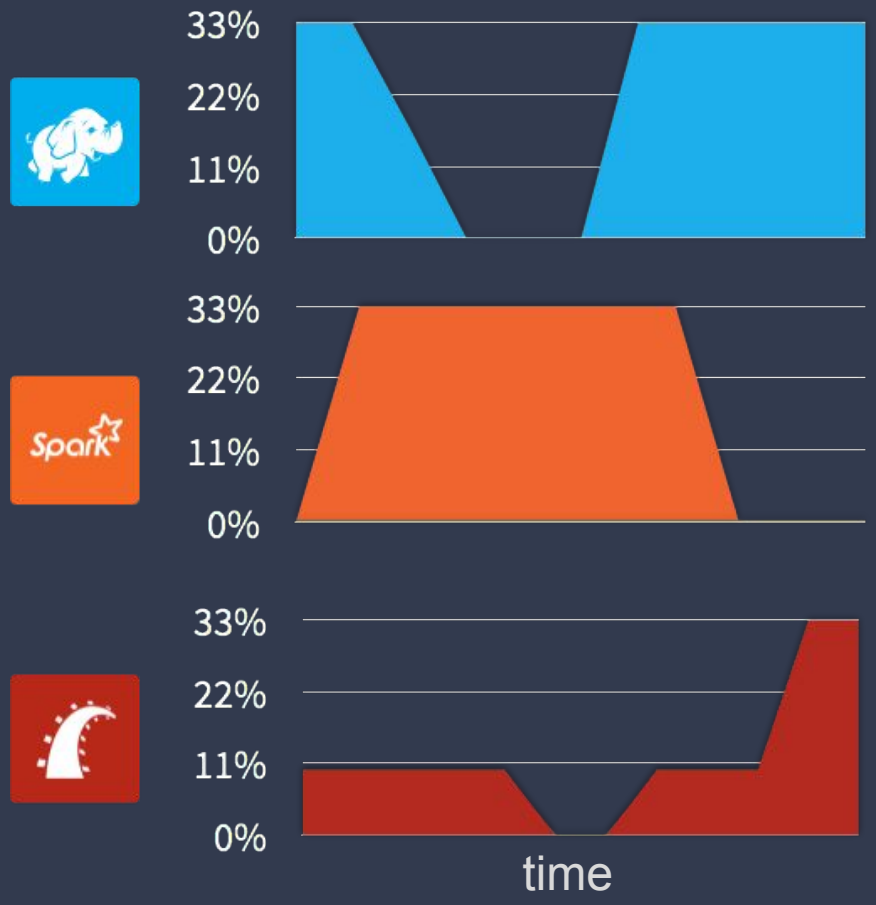
Create a Marathon app

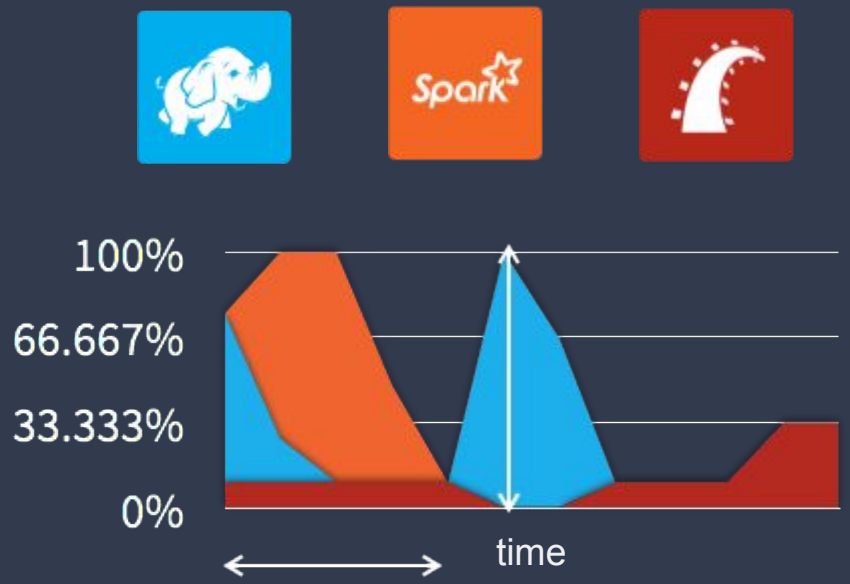


A typical Mesos cluster



Mesos helps improve cluster utilization





Why should I pick Mesos?

- Production ready
- Proven scalability
- Highly customizable and extensible

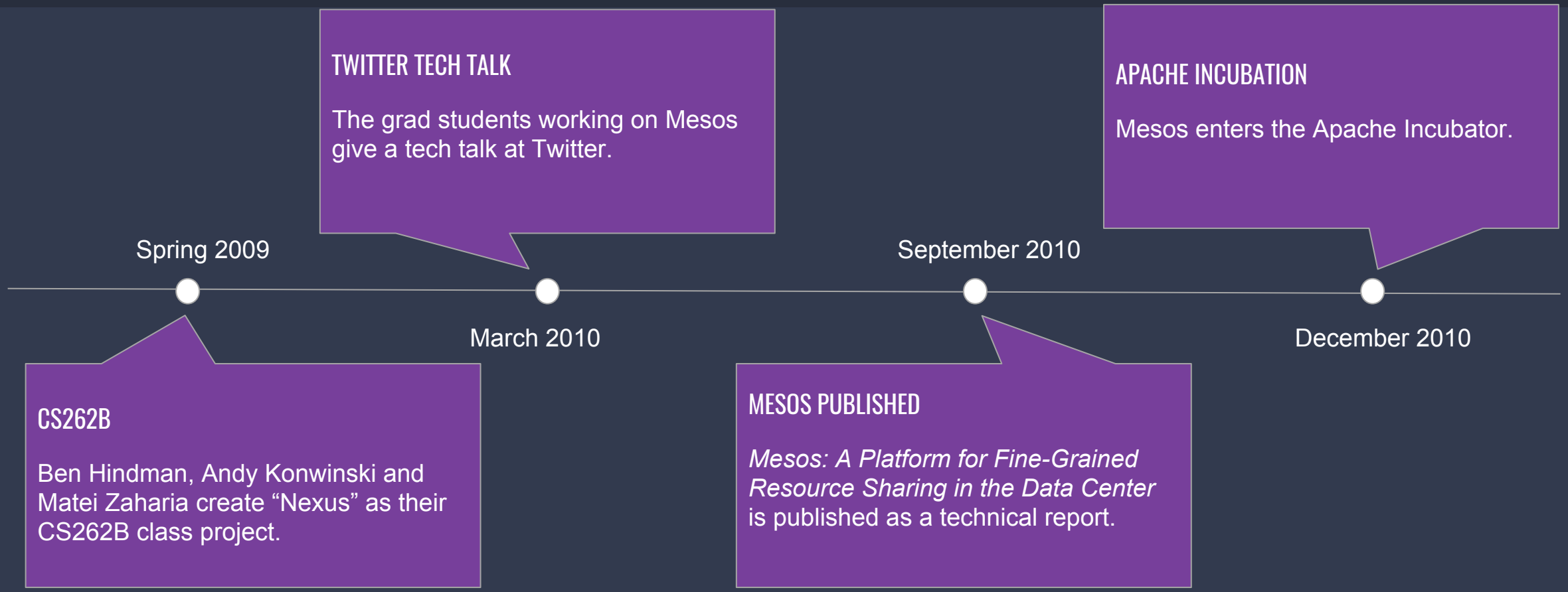
Production Ready

PRODUCTION CUSTOMERS AND MESOS USERS

Proven reliable for large scale, mission-critical deployments

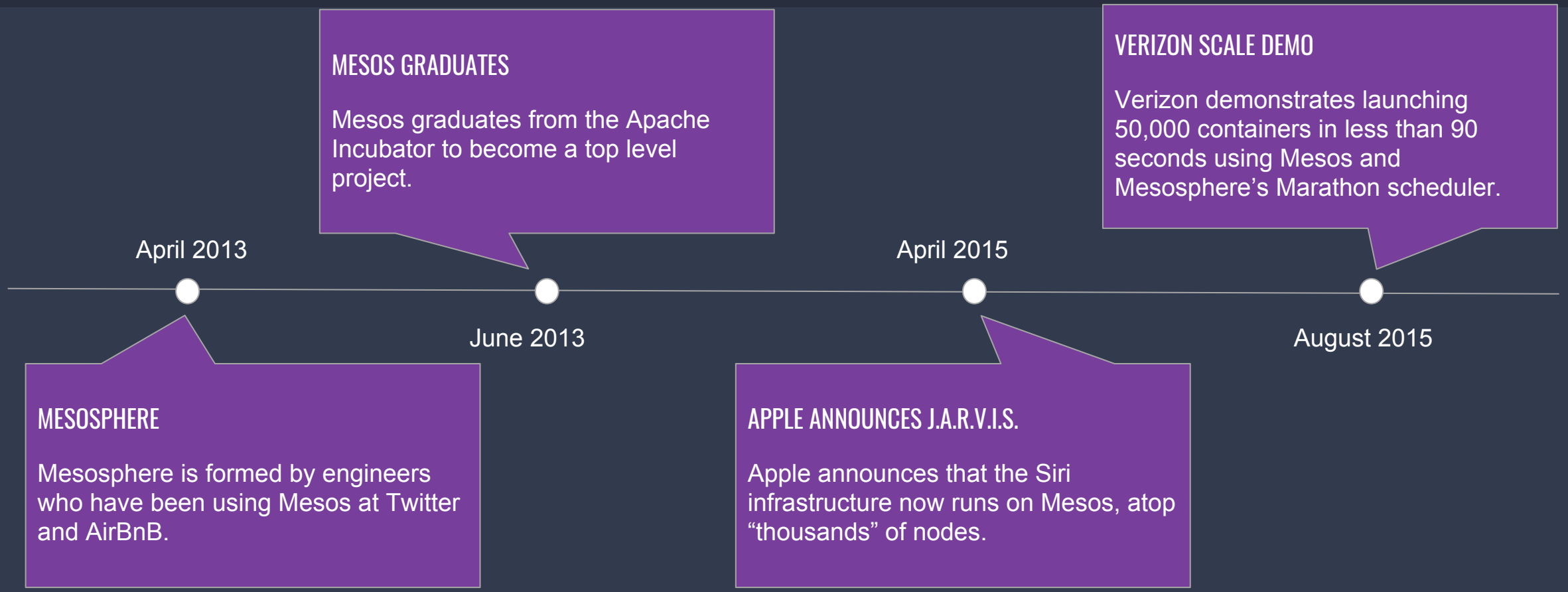

POWERED BY MESOS

The history of Mesos



Why Mesos?

The history of Mesos



Proven Scalability

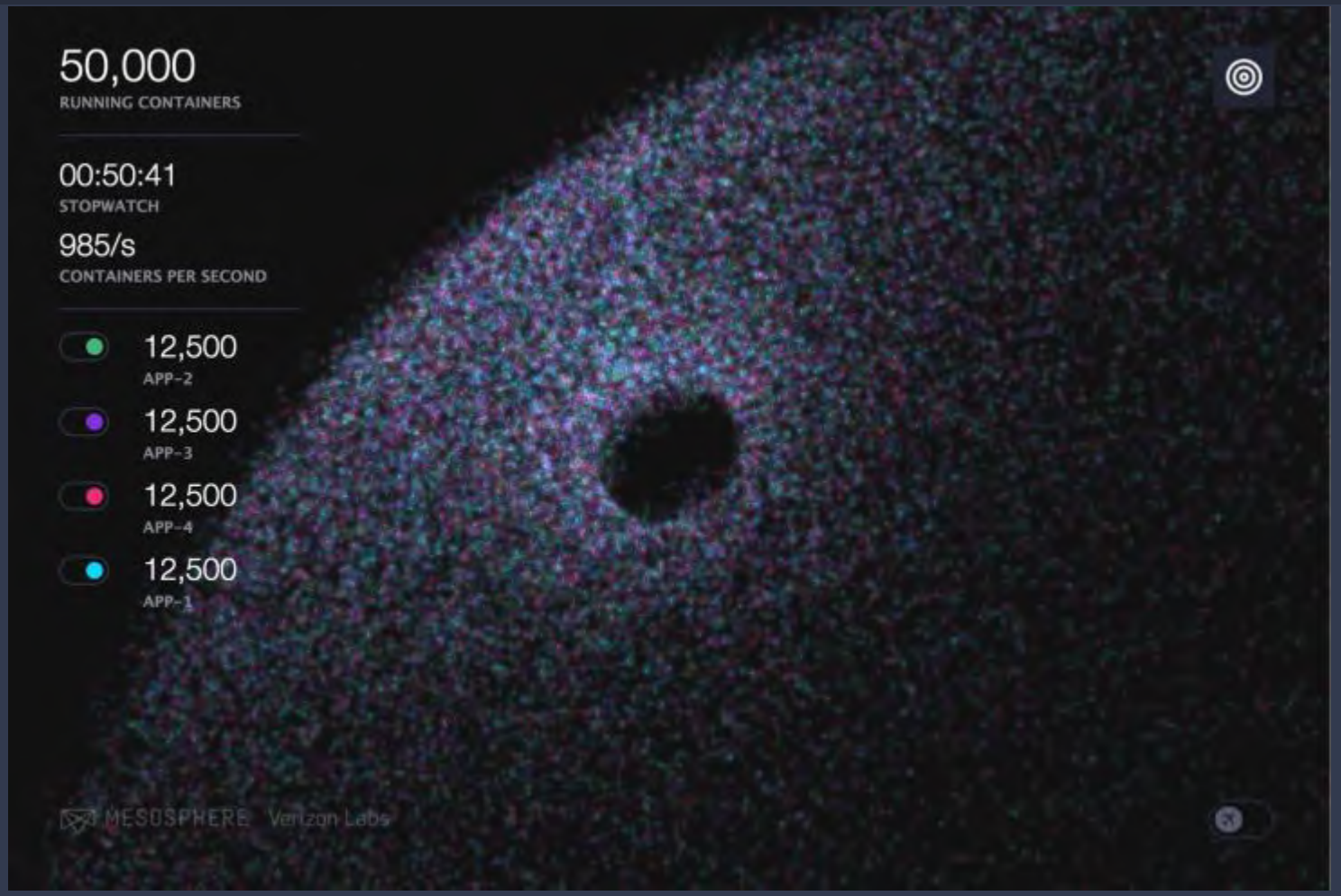
Twitter

- **Largest Mesos cluster**
  - > 30000 nodes
  - > 250K containers

Apple

- **Siri is powered by Mesos!**

Verizon



- **50K containers in 50 seconds**

Why Mesos is so scalable?

- **Stateless master**
  - Inspired from the GFS design
  - Agents hold truth about running tasks (distributed)
  - Master state can be reconstructed when agents register
- **Simple, only cares about**
  - Resource allocation and isolation
  - Task management
- **Implemented in C++**
  - Native performance
  - No GC issue

Why Mesos?

What does it mean to you?

- Known that Mesos will scale to Twitter/Apple level
  - Feature is easy to add, took time to make it scalable
- Quality assurance for free
  - Imagine a test environment having 30k+ nodes with real workload
- Take backwards compatibility seriously
  - We don’t want to break their production environment

Highly Customizable and Extensible

Why this is important?

- Every company's environment is different
  - Scheduling
  - Service discovery
  - Container image format
  - Networking
  - Storage
  - Special hardware/accelerators (e.g., GPU, FPGA)

- No one-fits-all solution typically

Pluggable schedulers

- For instance, you need separate schedulers for
  - Long running stateless services
  - Cron jobs
  - Stateful services (e.g., database, DFS)
  - Batch jobs (e.g., map-reduce)

Mesos frameworks == pluggable schedulers

- Monolithic scheduler?

  Monolithic schedulers do not make it easy to add new policies and specialized implementations, and may not scale up to the cluster sizes we are planning for.

  --- From Google Omega Paper (EuroSys'13)

Flexible service discovery

- Mesos is not opinionated about service discovery
  - DNS based
  - ZK/Etcd/Chubby based (e.g., twitter, google, with client libraries)
  - Your custom way, every company is different
  - Mesos provides an endpoint to stream SD information

- DNS based solution does not scale well

  Larger jobs create worse problems, and several jobs many be running at once. The variability in our DNS load had been a serious problem for Google before Chubby was introduced.

  --- From Google Chubby paper (OSDI'06)

Why Mesos?

Pluggable and extensible containerization

- Container image format
- Networking
- Storage
- Security
- Custom isolation
- Container lifecycle hooks

Container standards/specifications supported by Mesos

- Container Image
  - OCI (Open Container Initiative)
  - Docker
  - Appc
- Container Network
  - CNI
- Container Storage
  - Docker Volume (dvdi)
  - CSI (new Container Storage Interface)

Container network support

- Support Container Network Interface (CNI) from 1.0
  - A spec for container networking
  - Supported by most network vendors

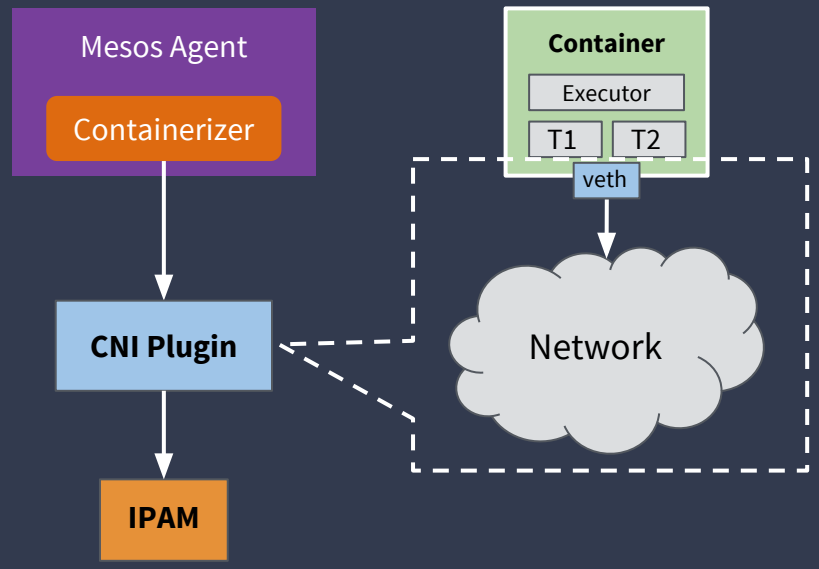
- Implemented as an isolator
  - `--isolation=network/cni,...`

Container Network Interface (CNI)

- Proposed by CoreOS :

  https://github.com/containernetworking/cni

- Simple contract between container runtime and CNI plugin defined in the form of a JSON schema
  - CLI interface
  - ADD: attach to network
  - DEL: detach from network



Why CNI?

- Simpler and less dependencies than Docker CNM
- Backed by Kubernetes community as well
- Rich plugins from network vendors
- Clear separation between container and network management
- IPAM has its own pluggable interface

CNI plugins

Existing CNI plugins

- ipvlan
- macvlan
- bridge
- flannel
- calico
- contiv
- contrail
- weave
- …

You can write your own plugin, and Mesos supports it!

Container storage support

- Support Docker volume plugins from 1.0
  - Define the interface between container runtime and storage provider
  - https://docs.docker.com/engine/extend/plugins_volume/
- A variety of Docker volume plugins
  - Ceph
  - Convoy
  - Flocker
  - Glusterfs
  - Rexray

CSI

CSI: Towards a more universal storage interface for containers

Benjamin Hindman

March 30, 2017

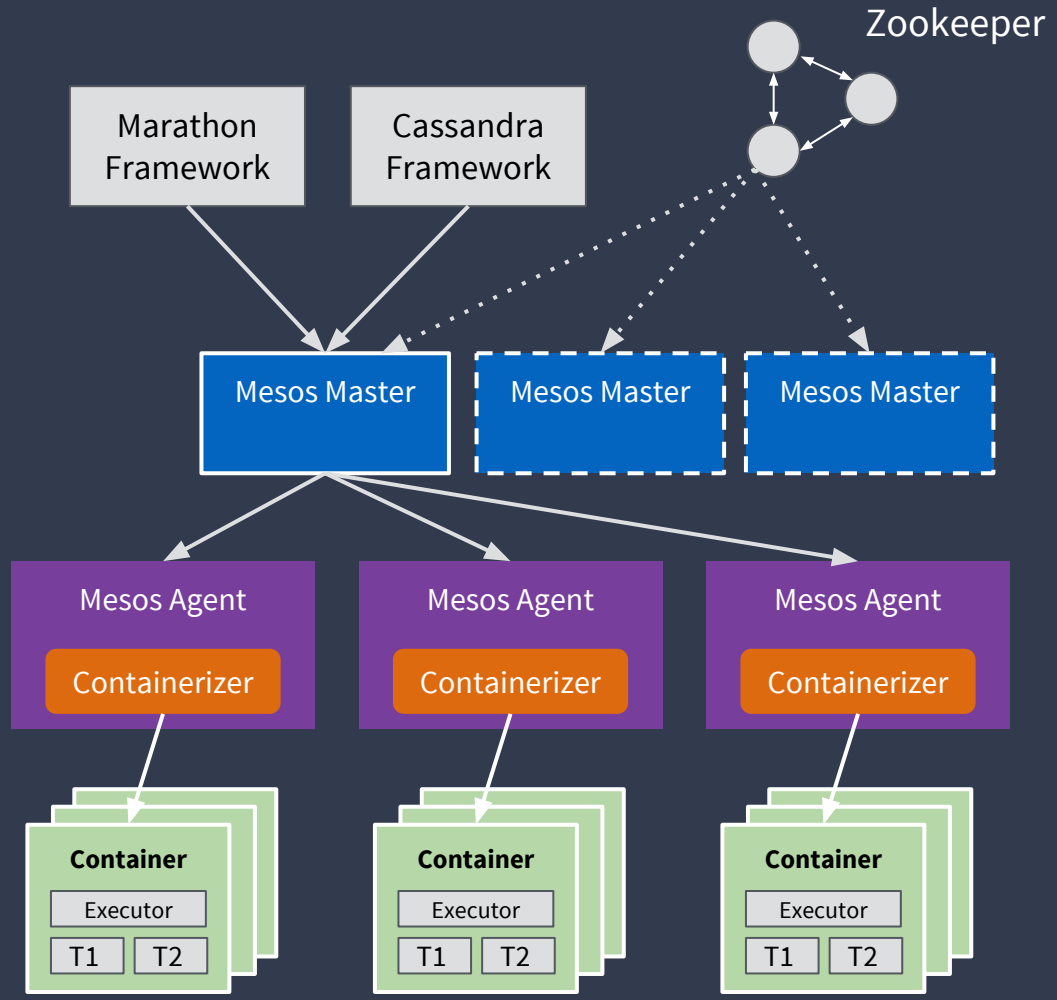
Standard interfaces can provide many benefits. For technology vendors, they facilitate collaboration, enable interoperability, and save time and resources from building one-off integrations. For customers, standard interfaces accelerate technology adoption, simplify the user experience, and enable choice. The success of the Container Networking Interface got us to think: can we do the same for container storage?

Why CSI

- Design issues with docker volume plugins
- Issues with other storage spec:
  - Kubernetes Flex Volume
  - Libstorage
- Need a new container storage spec

Containerization in Mesos

Containerizer



Containerizer

- Between agents and containers
- Launch/update/destroy containers
- Provide isolations between containers
- Report container stats and status

Currently supported containerizers

Docker containerizer

- Delegate to Docker daemon

Mesos containerizer

- Using standard OS features (e.g., cgroups, namespaces)
- Pluggable architecture allowing customization and extension

Very stable. Used in large scale production clusters

Currently supported containerizers

Docker containerizer

- Delegate to Docker daemon

Mesos containerizer

- Using standard OS features (e.g., cgroups, namespaces)
- Pluggable architecture allowing customization and extension
- Support Docker, Appc, OCI (soon) images natively w/o dependency

Very stable. Used in large scale production clusters

Currently supported containerizers

Docker containerizer

- Delegate to Docker daemon

Unified containerizer

- Using standard OS features (e.g., cgroups, namespaces)
- Pluggable architecture allowing customization and extension
- Support Docker, Appc, OCI (soon) images natively w/o dependency

Very stable. Used in large scale production clusters

Unified Containerizer

Container image support

Start from 0.28, you can run your Docker container on Mesos without a Docker daemon installed!

- One less dependency in your stack
- Agent restart handled gracefully, task not affected
- Compose well with all existing isolators
- Easier to add extensions

Pluggable container image format

- Mesos supports multiple container image format
  - Docker (without docker daemon)
  - Appc (without rkt)
  - OCI (ready soon)
  - CVMFS (experimental)
  - Host filesystem with tars/jars
  - Your own image format!

Used in large scale production clusters

Latest Features

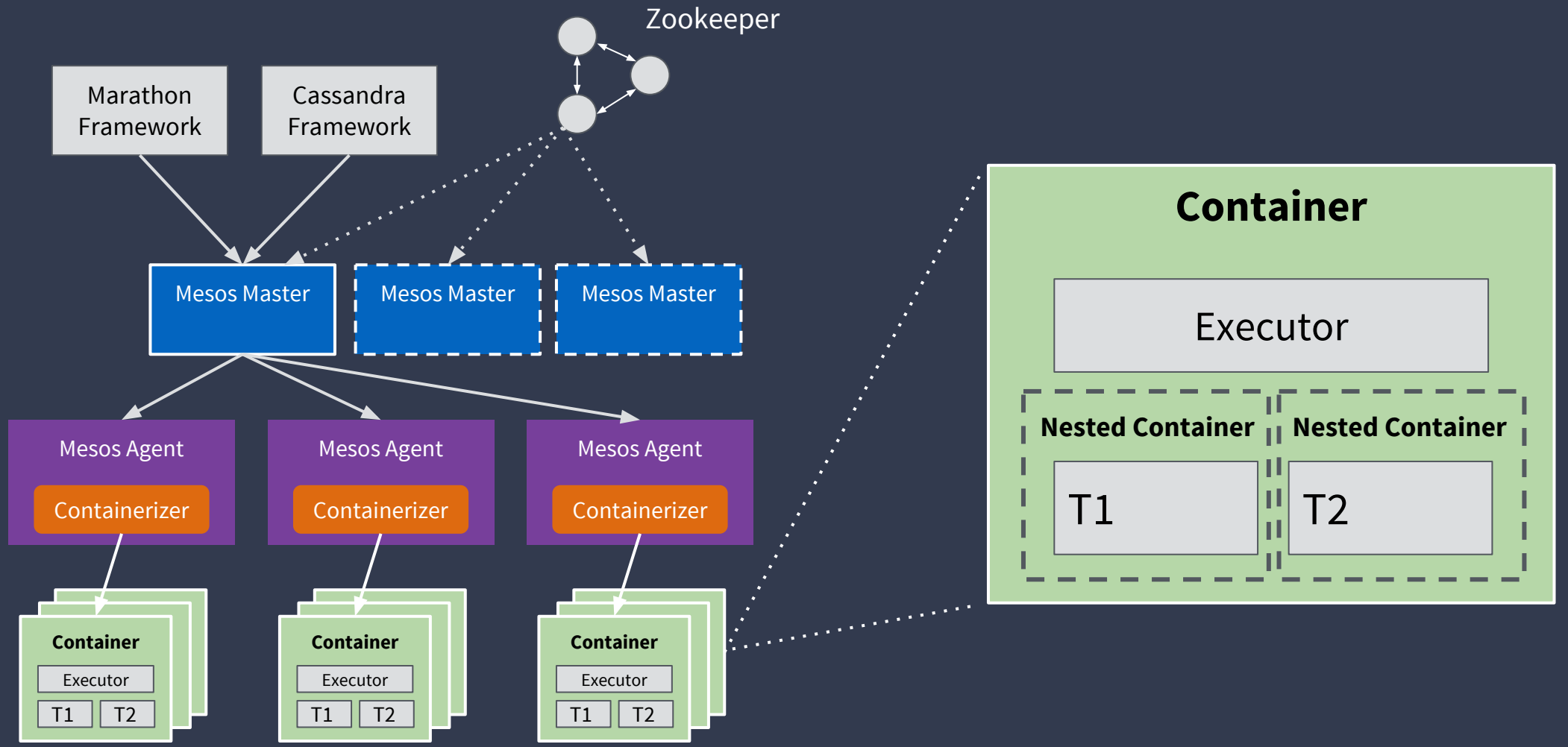
- Unified containerizer
- GPU support
- Nested container and task group (Pods)
- Debug container
- Multi role and hierarchy role

Nested container support

- New in Mesos 1.1
  - Building block for supporting Pod like feature

- Highlighted features
  - Support arbitrary levels of nesting
  - Re-use all existing isolators
  - Allow dynamically creation of nested containers

Nested container support



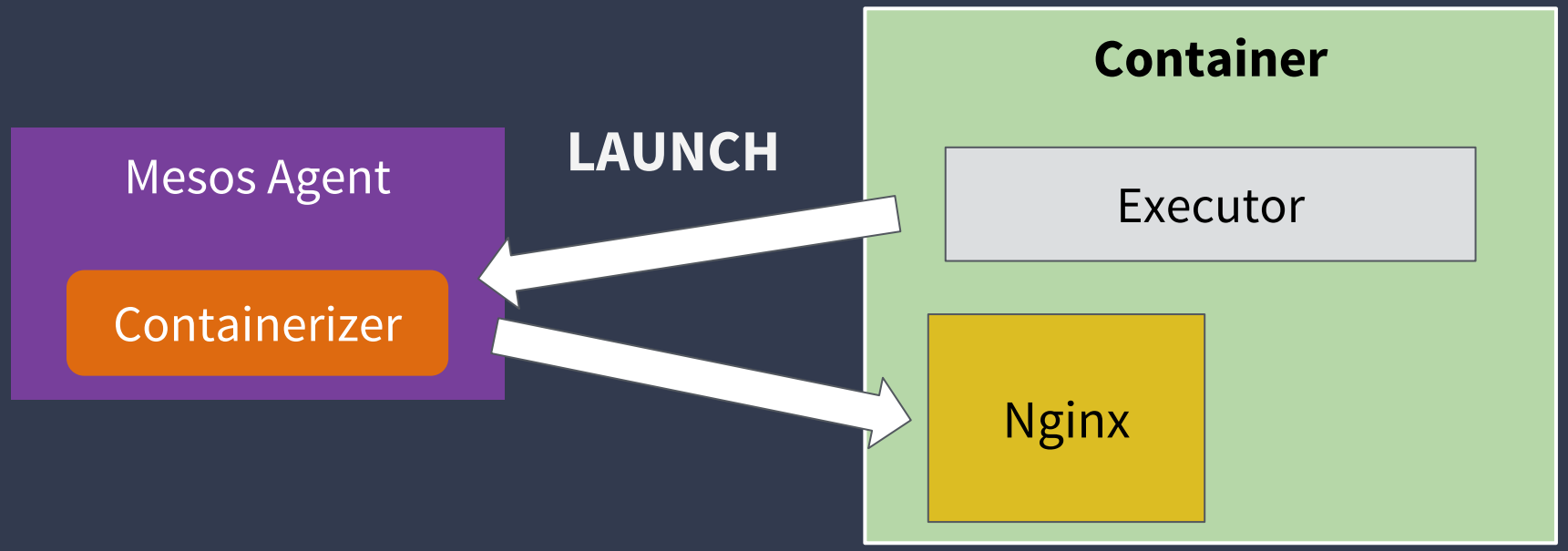
Nested container support

New Agent API for Nested Containers

```
message agent::Call {
  enum Type {
    // Calls for managing nested containers
    // under an executor's container.
    LAUNCH_NESTED_CONTAINER = 14;
    WAIT_NESTED_CONTAINER = 15;
    KILL_NESTED_CONTAINER = 16;
  }
}
```

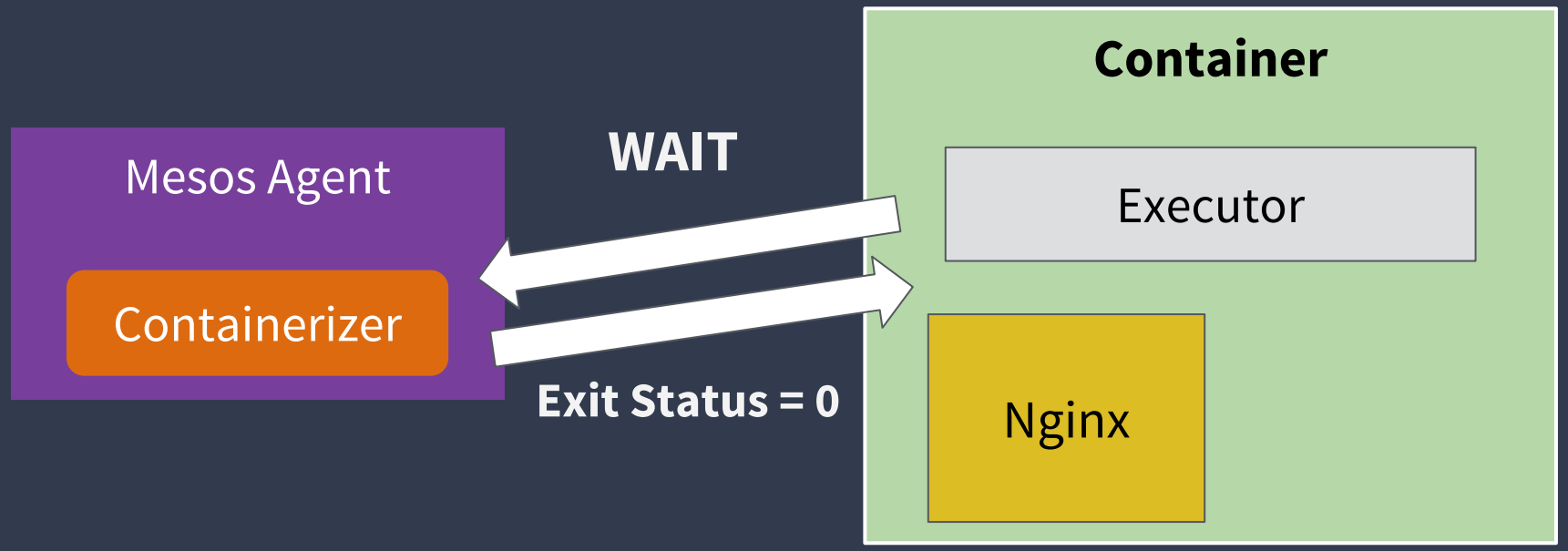
Nested container support

Launch nested container



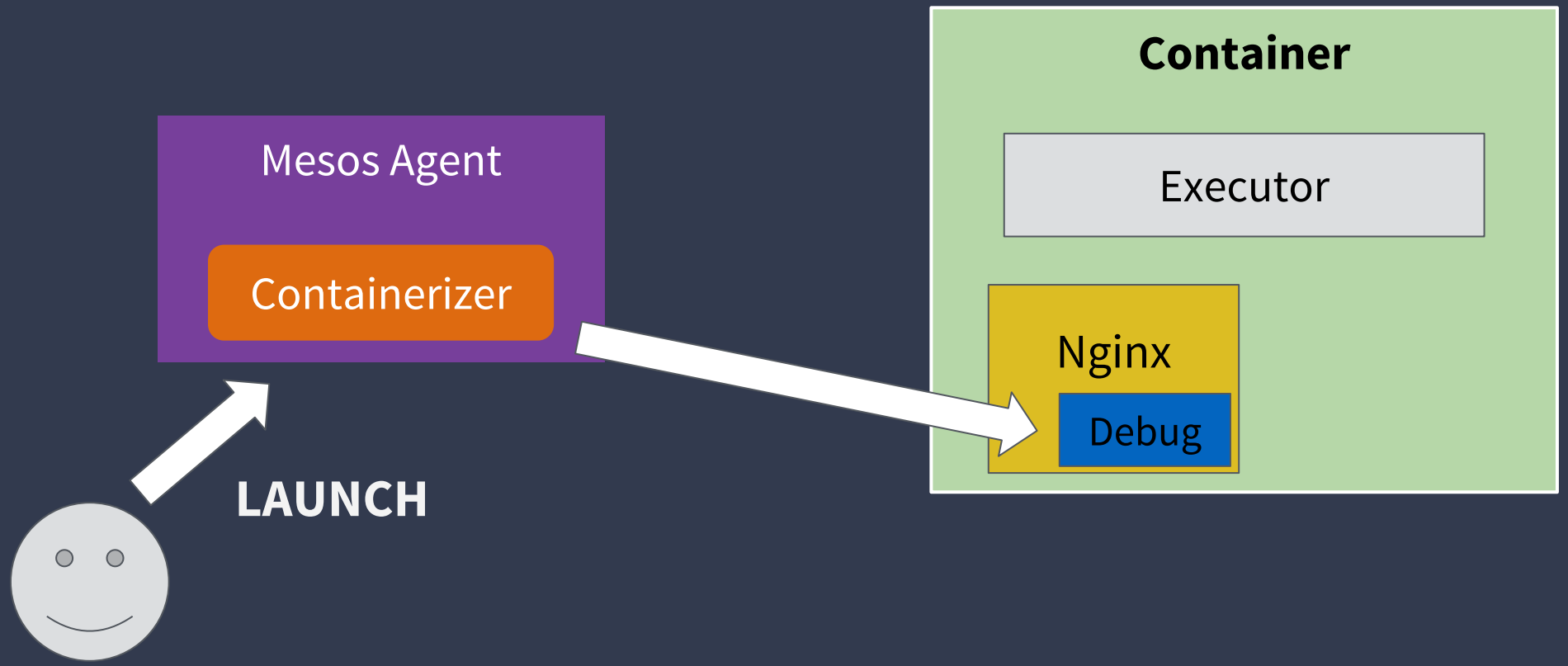
Nested container support

Watch nested container



Nested container support

Arbitrary levels of nesting



Demo