MESOS
A State-Of-The-Art Container Orchestrator

MESOSPHERE

About me

Jie Yu (@jie_yu)

- Tech Lead at Mesosphere
- Mesos PMC member and committer
- Formerly worked at Twitter
- PhD from University of Michigan
- Worked on Mesos since 2012

http://people.apache.org/~jieyu/

Outline

- Mesos overview and fundamentals
- Why should I pick Mesos?
- Containerization in Mesos

Mesos: A kernel for data center applications

- What does a traditional OS kernel provide?

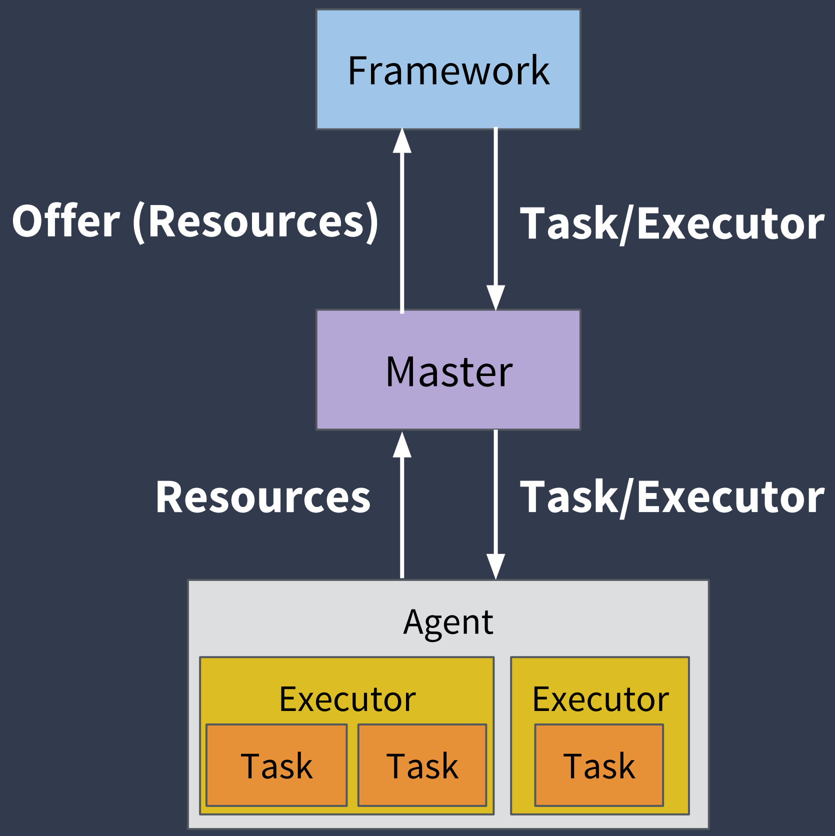
○ Resource management	Host cpu, memory, etc.
○ Programming abstractions	POSIX API: processes, threads, etc.
○ Security and isolation	Virtual memory, user, etc.

- Mesos: A kernel for data center applications

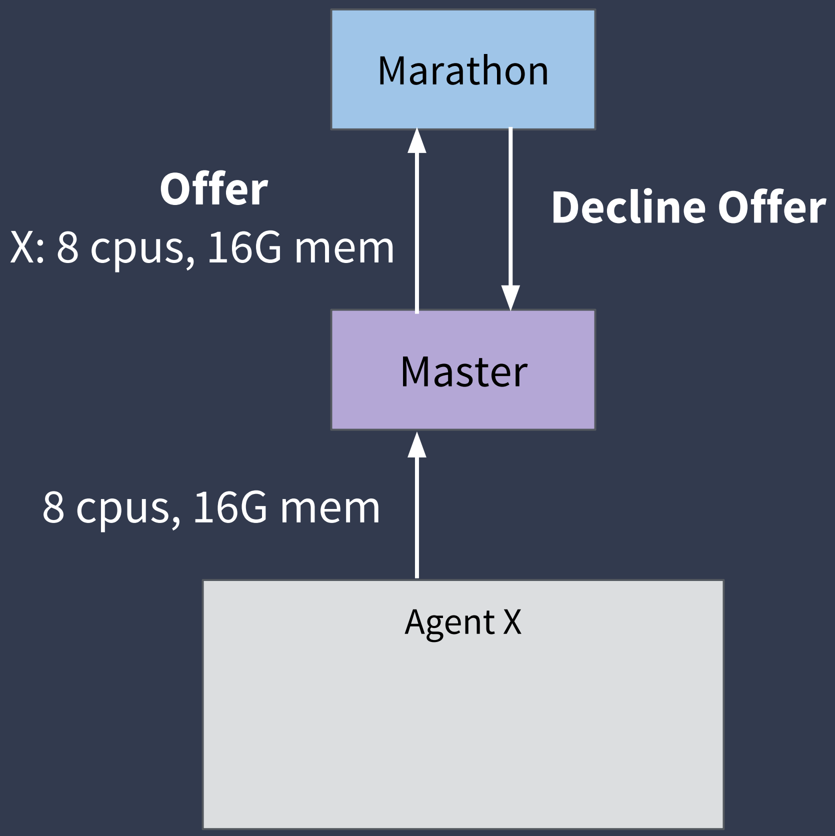
○ Resource management	Cluster cpu, memory, etc.
○ Programming abstractions	Mesos API: Task, Resource, etc.
○ Security and isolation	Containerization

Programming abstractions

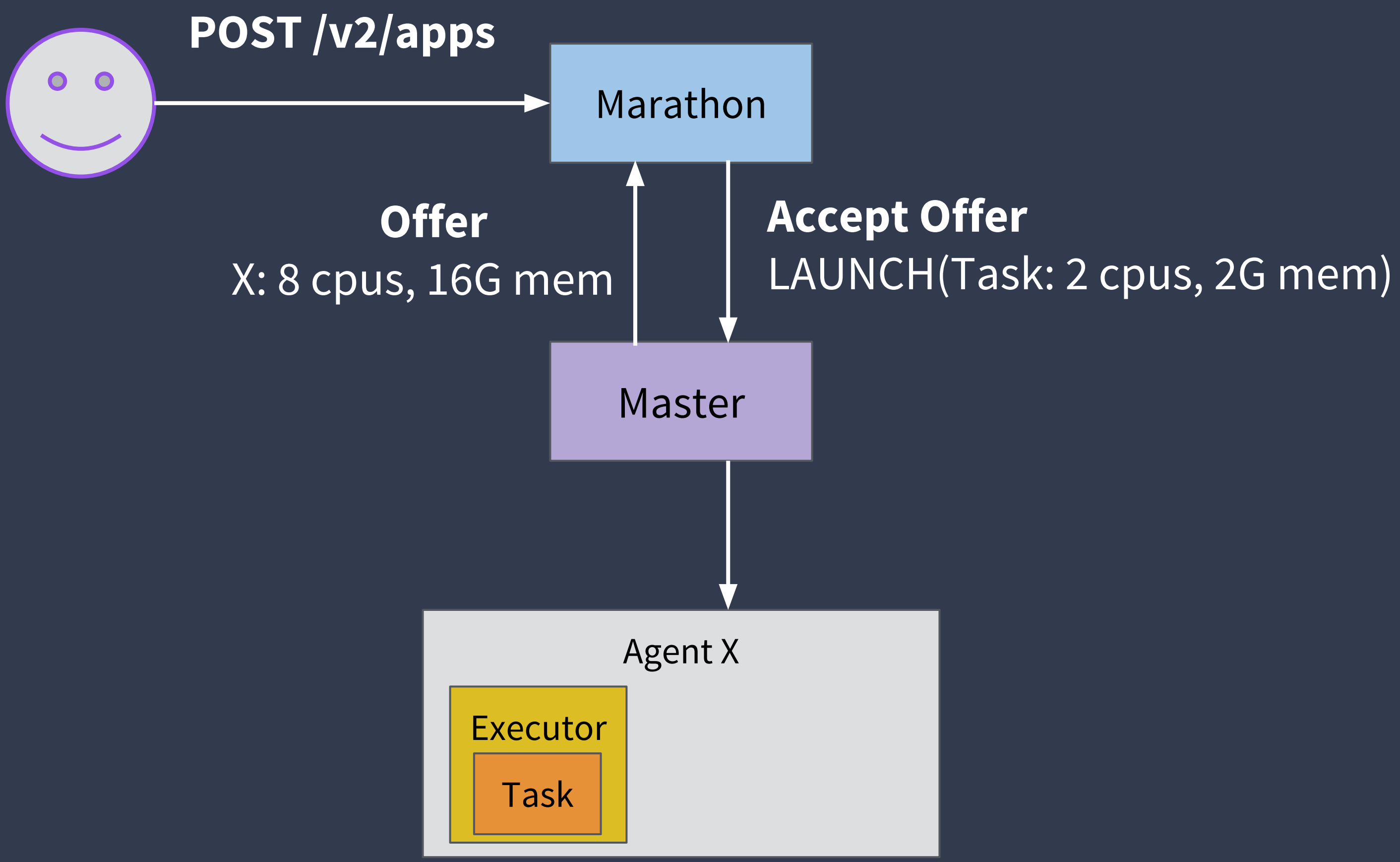
- Key concepts
  - Framework
  - Resource/Offer
  - Task
  - Executor



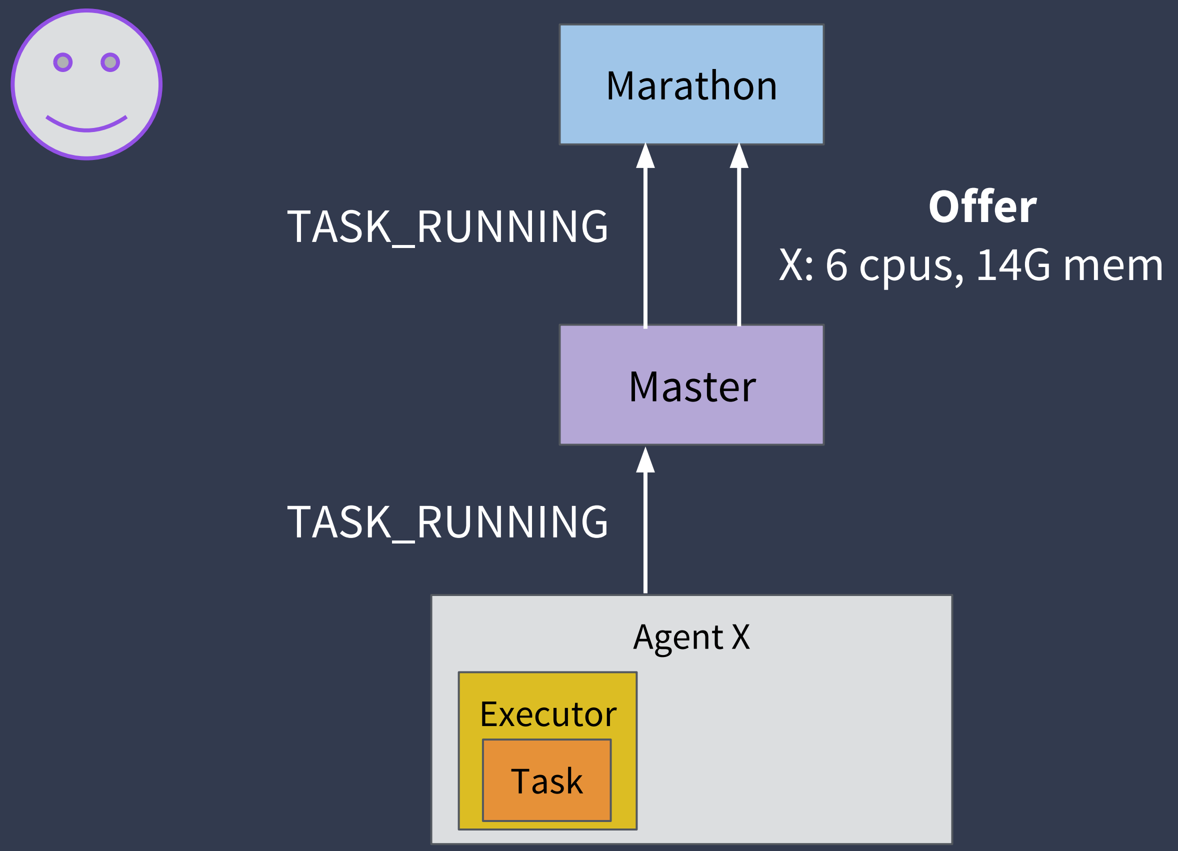
Case study: Marathon



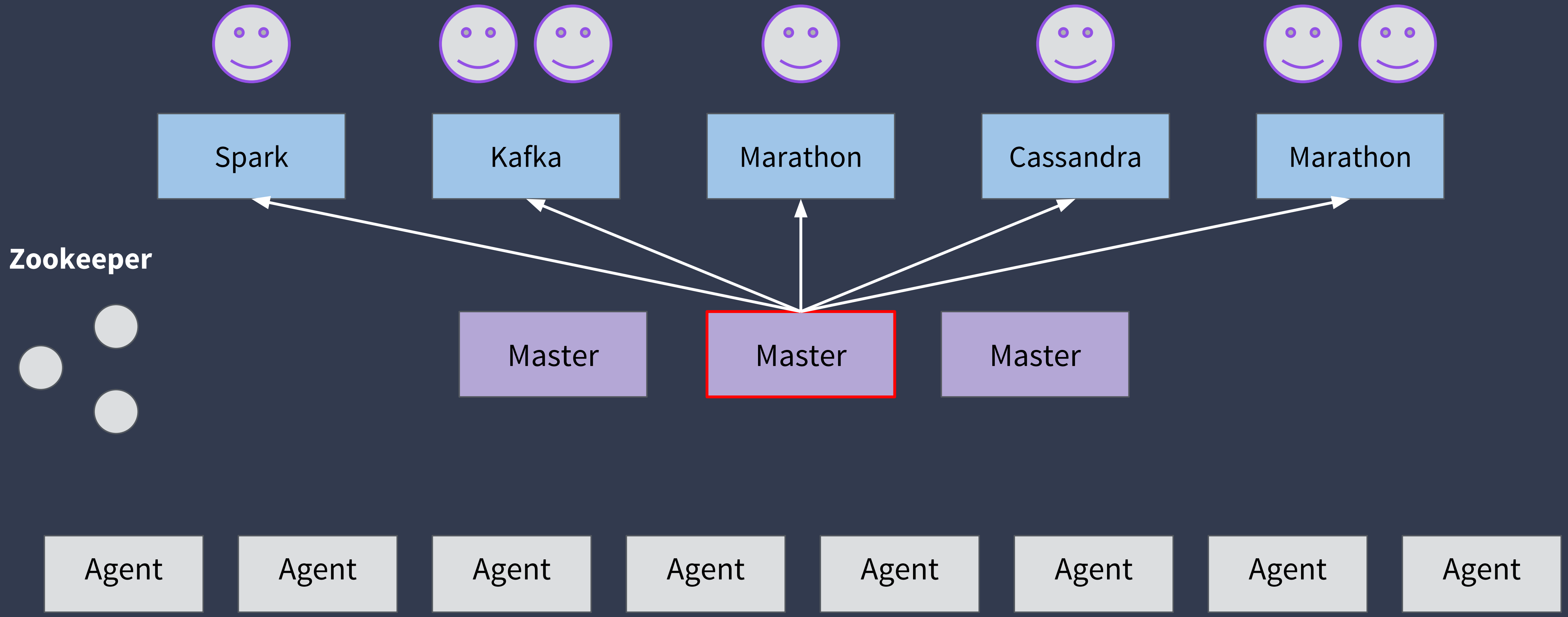
Create a Marathon app



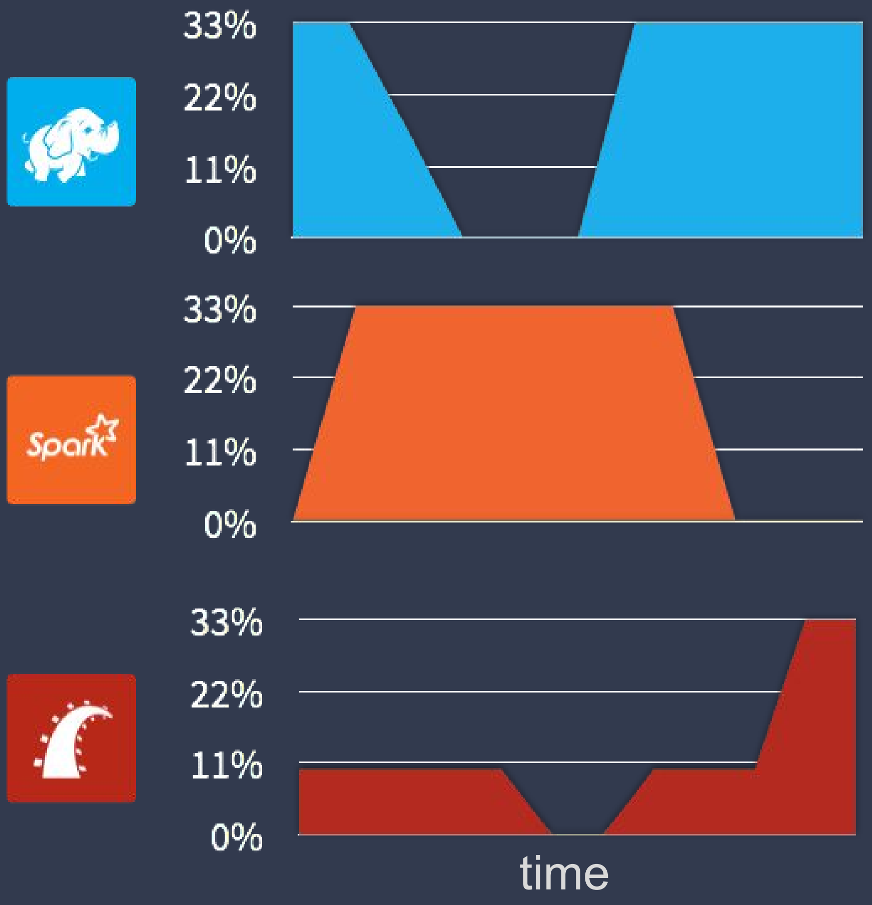
Create a Marathon app

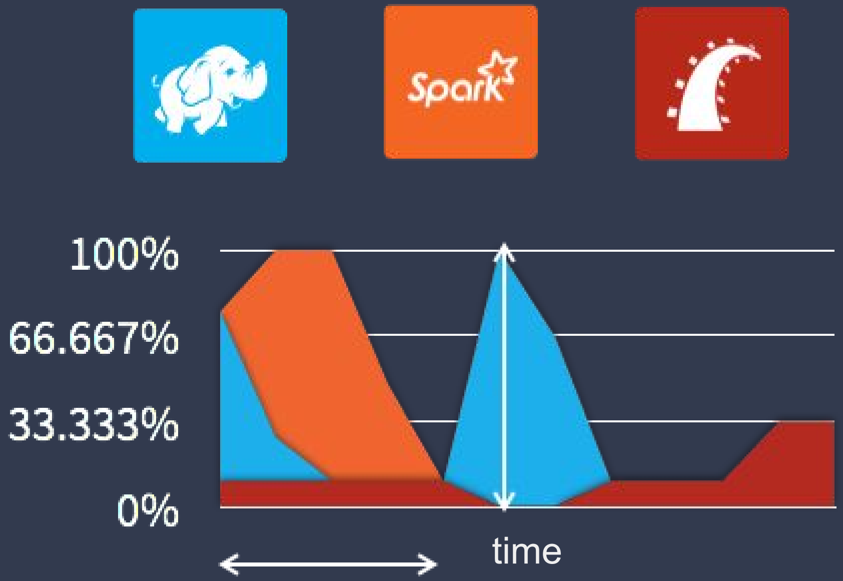


A typical Mesos cluster

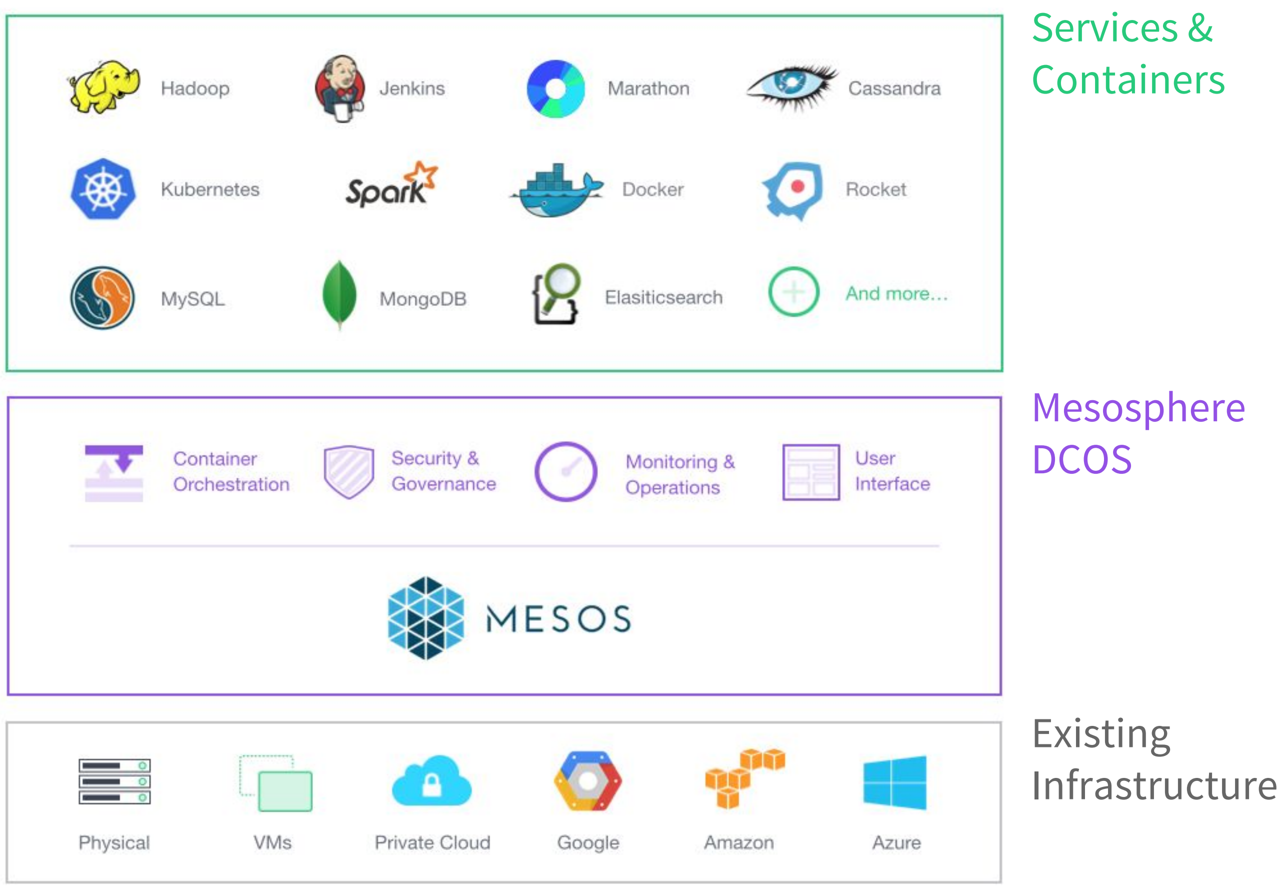


Mesos helps improve cluster utilization





DS/OS vs. Mesos



- Kernel alone is not enough
- DC/OS: the easiest way to run Mesos
  - CLI/UI
  - Package management
  - Service discovery
  - Load balancing
  - Day2 ops
  - Security
  - Framework SDK
- Yes, it is open source!

Why should I pick Mesos?

- Production ready
- Proven scalability
- Highly customizable and extensible

Production Ready

Why Mesos?

The birth of Mesos

Spring 2009

CS262B

Ben Hindman, Andy Konwinski and Matei Zaharia create “Nexus” as their CS262B class project.

March 2010

TWITTER TECH TALK

The grad students working on Mesos give a tech talk at Twitter.

September 2010

MESOS PUBLISHED

Mesos: A Platform for Fine-Grained Resource Sharing in the Data Center is published as a technical report.

December 2010

APACHE INCUBATION

Mesos enters the Apache Incubator.

Why Mesos?

Widely adopted

April 2013

MESOSPHERE

Mesosphere is formed by engineers who have been using Mesos at Twitter and AirBnB.

June 2013

MESOS GRADUATES

Mesos graduates from the Apache Incubator to become a top level project.

April 2015

APPLE ANNOUNCES J.A.R.V.I.S.

Apple announces that the Siri infrastructure now runs on Mesos, atop “thousands” of nodes.

August 2015

VERIZON SCALE DEMO

Verizon demonstrates launching 50,000 containers in less than 90 seconds using Mesos and Mesosphere’s Marathon scheduler.

Why Mesos?

Production Mesos users

Bloomberg

verizon

NETFLIX

ebay

PayPal

2σ TWO SIGMA

AUTODESK

airbnb

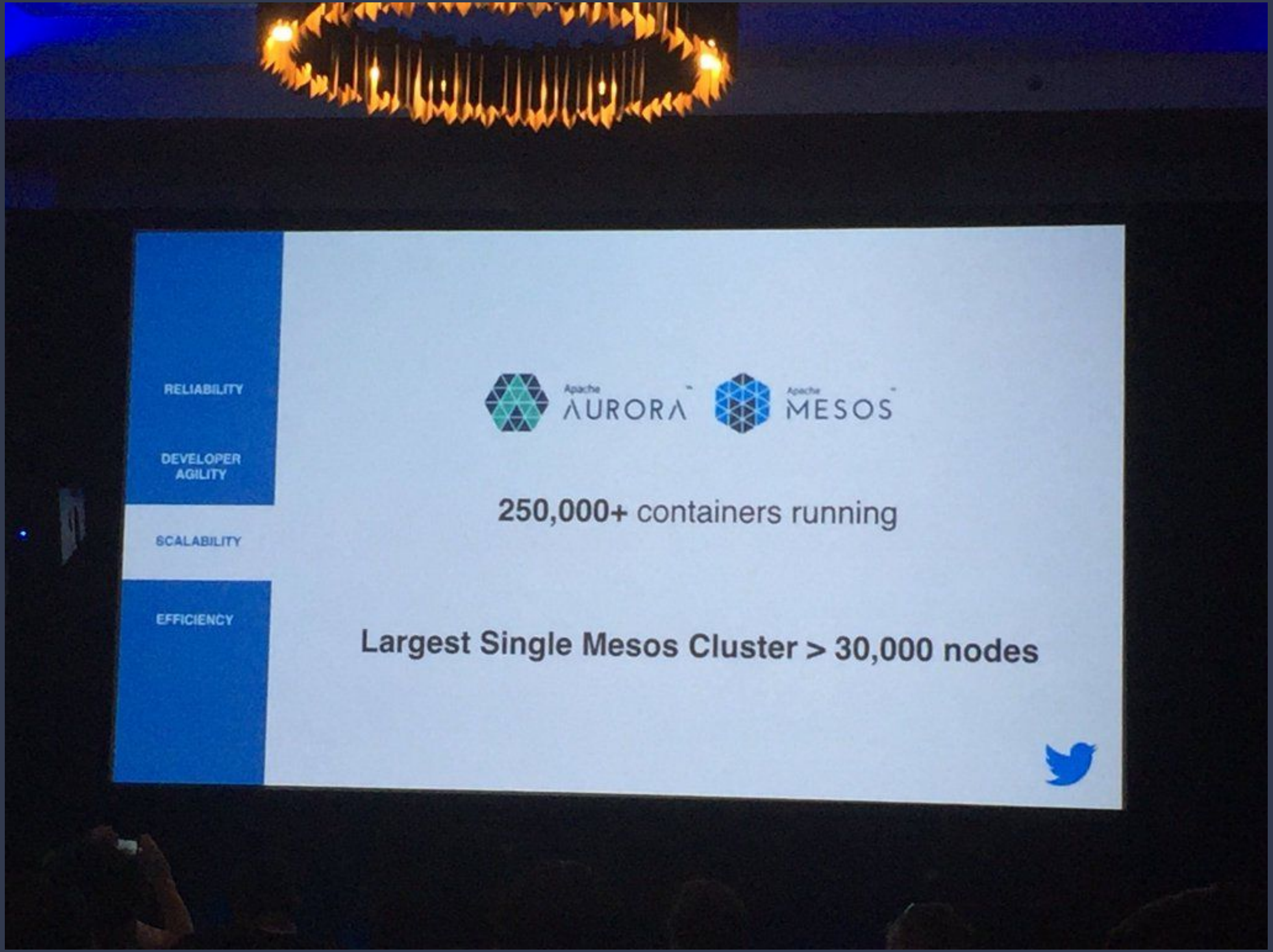
HubSpot

Government Agencies

yelp

Proven Scalability

Twitter



RELIABILITY

DEVELOPER AGILITY

SCALABILITY

EFFICIENCY

Apache AURORA Apache MESOS

250,000+ containers running

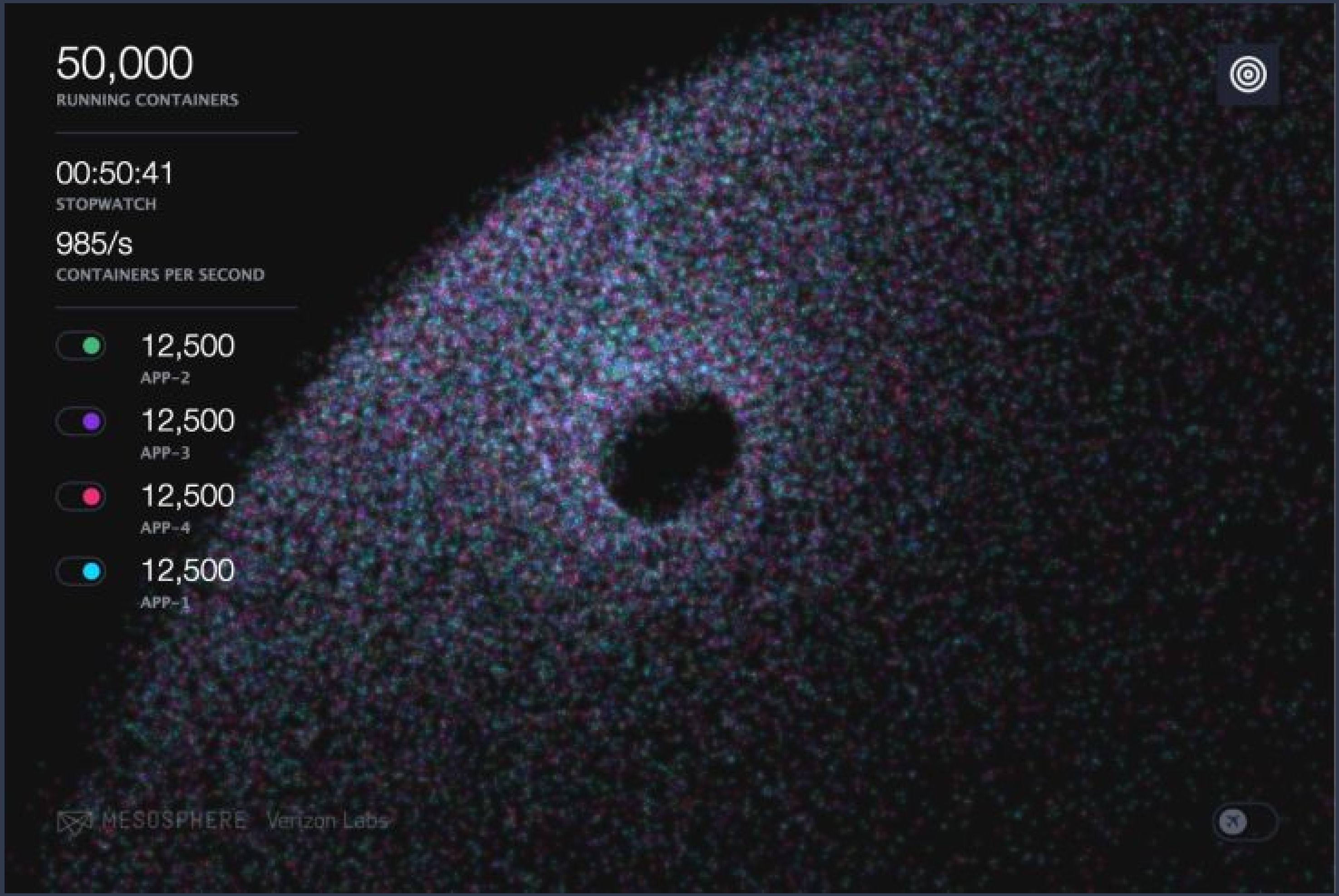
Largest Single Mesos Cluster > 30,000 nodes

- **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

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

Why Mesos?

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)

Pluggable and extensible containerization

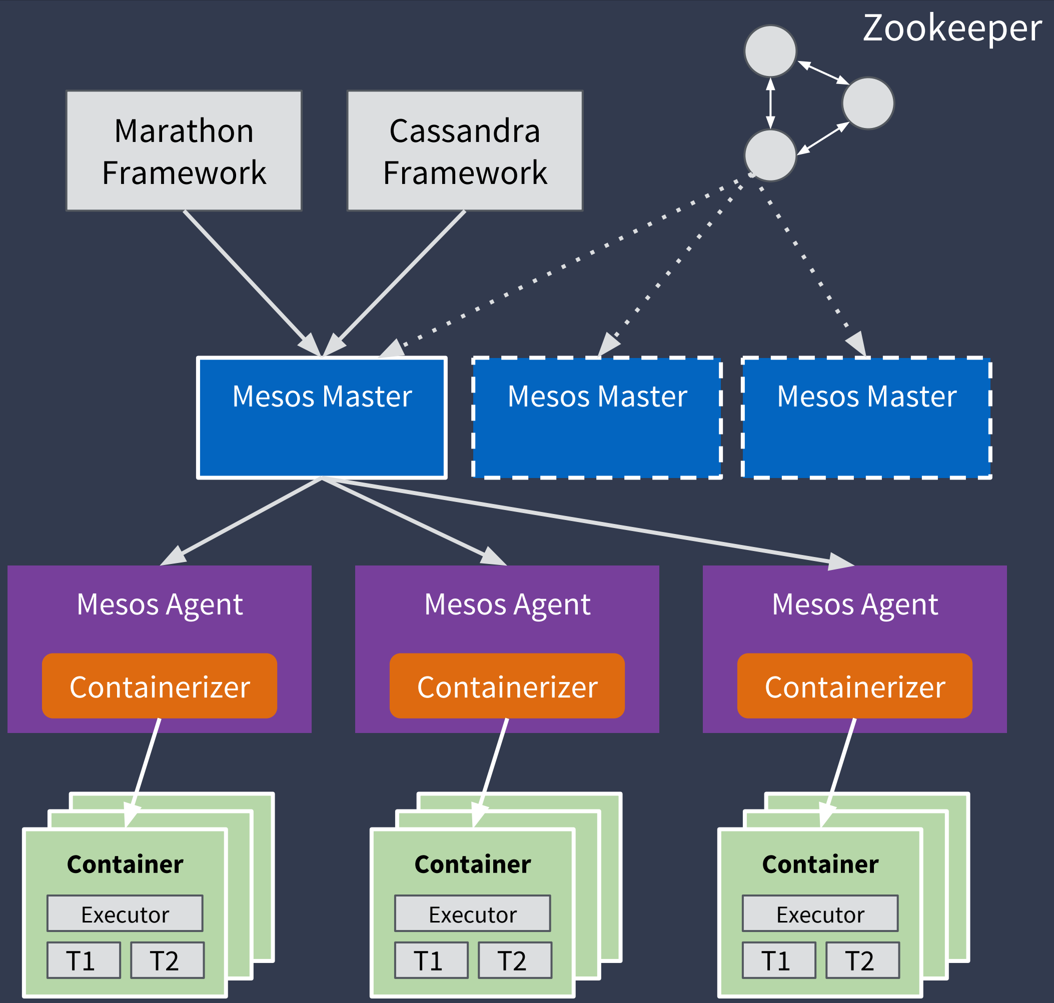
- Container image format
- Networking
- Storage
- Custom isolation
- Container lifecycle hooks

Outline

- Mesos overview and fundamentals
- Why should I pick Mesos?
- Containerization in Mesos
    - Pluggable architecture
    - Container image
    - Container network
    - Container storage
    - Customization and extensions
    - Nesting container support

Containerization in Mesos

What is 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

Containerization in Mesos

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

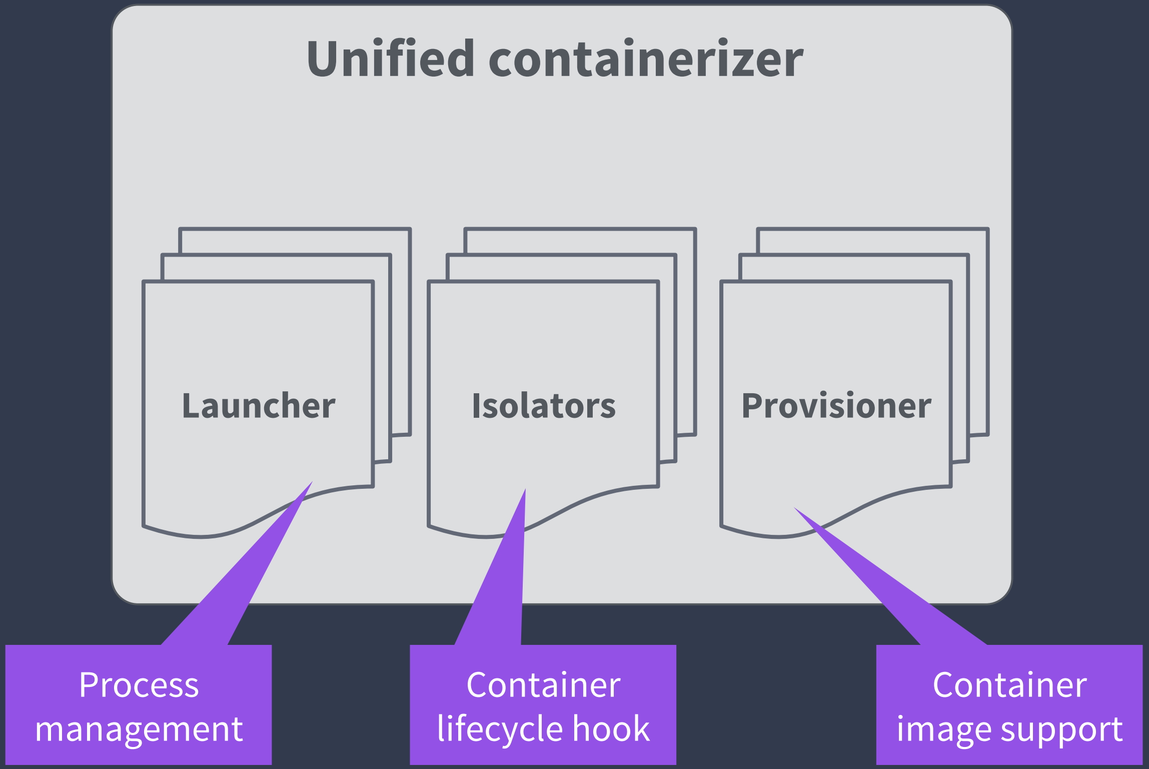
- Using standard OS features (e.g., cgroups, namespaces)
- Pluggable architecture allowing customization and extension
- Support Docker, Appc, OCI (soon) images natively w/o dependency

Very stable. Used in large scale production clusters

Unified Containerizer

- Pluggable architecture
- Container image
- Container network
- Container storage
- Customization and extensions
- Nesting container support

Pluggable architecture



Launcher

Responsible for process management

- Spawn containers
- Kill and wait containers

Supported launchers:

- Posix launcher
- Linux launcher
- Windows launcher

Unified Containerizer

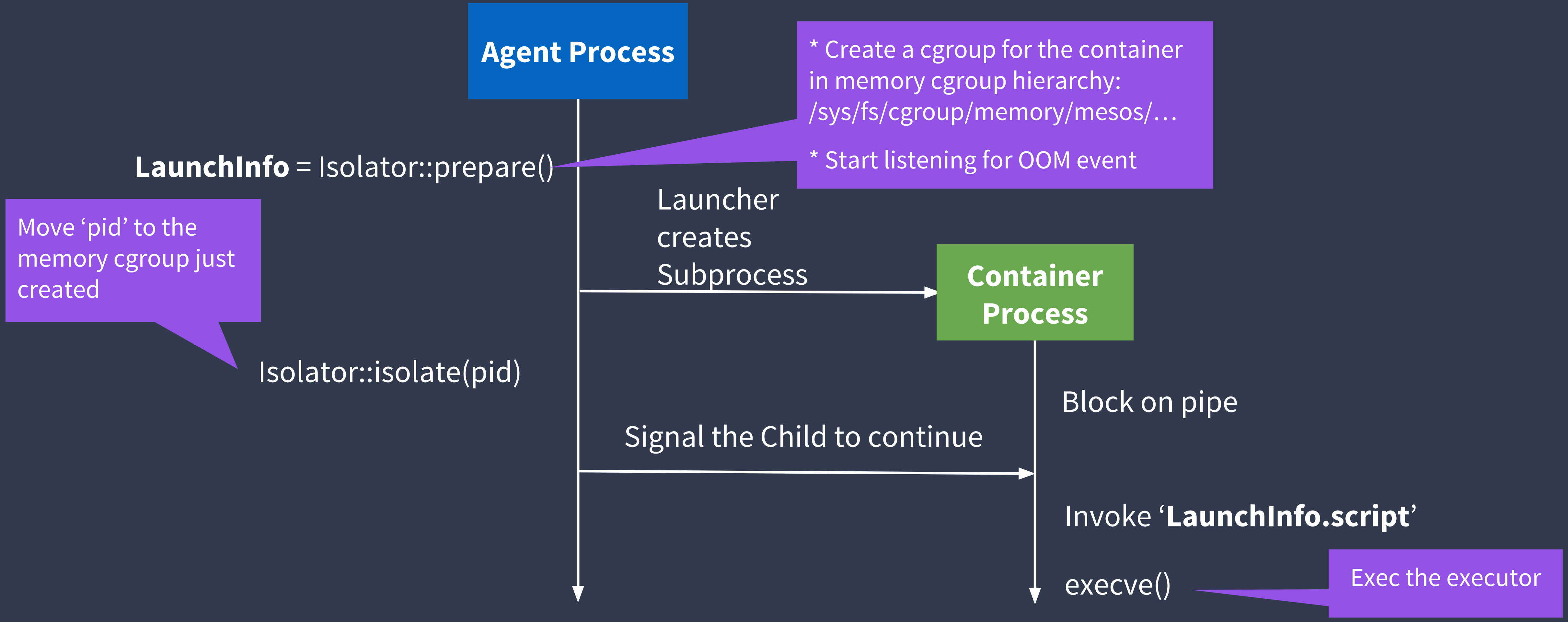
Isolator

Interface for extensions during the **life cycle** of a container

- Pre-launch - prepare()
- Post-launch (both in parent and child context) - isolate()
- Termination - cleanup()
- Resources update - update()
- Resources limitation reached - watch()
- Agent restart and recovery - recover()
- Stats and status pulling - usage()

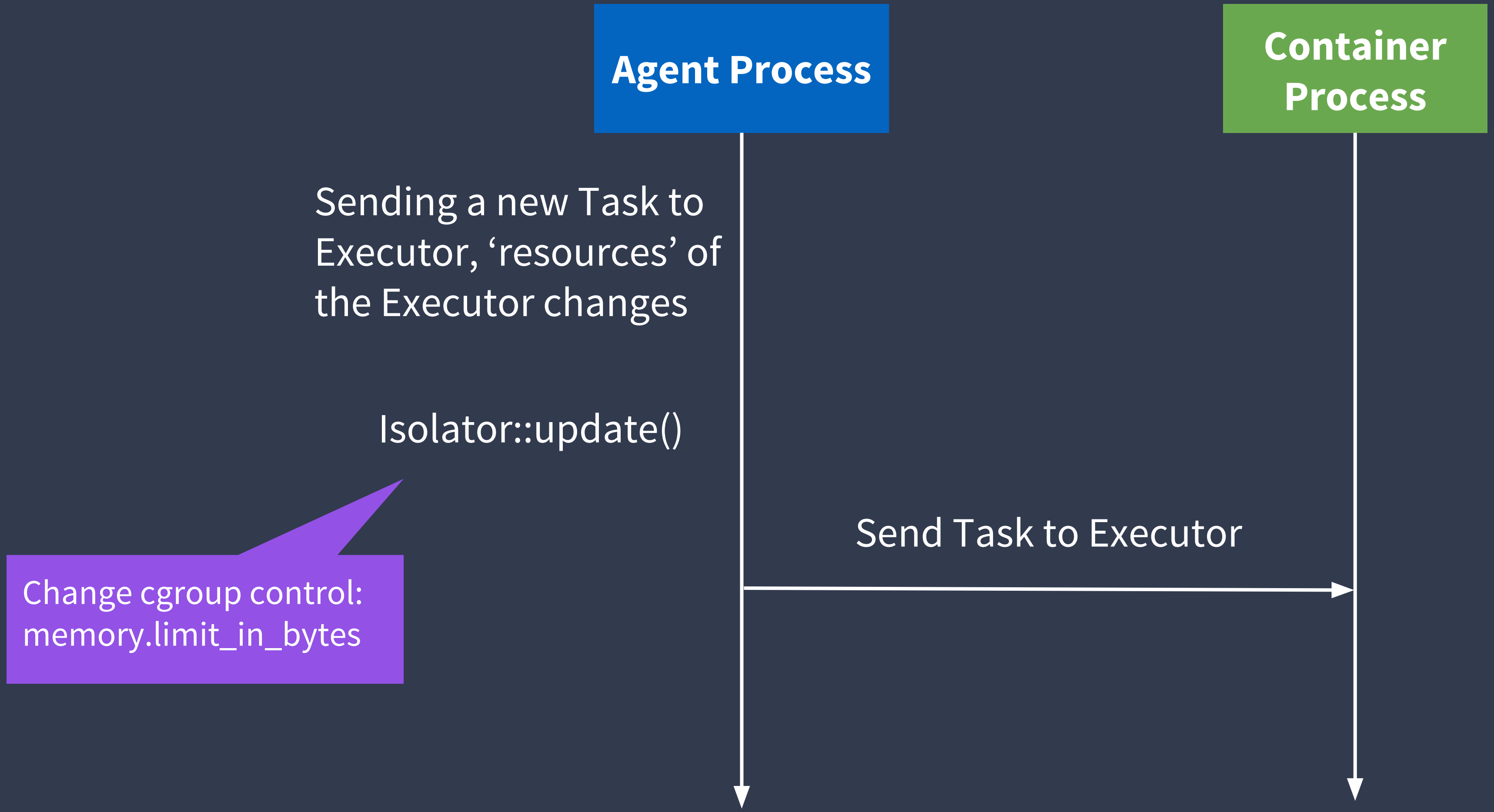
Sufficient for most of the extensions!

Isolator example: cgroups memory isolator

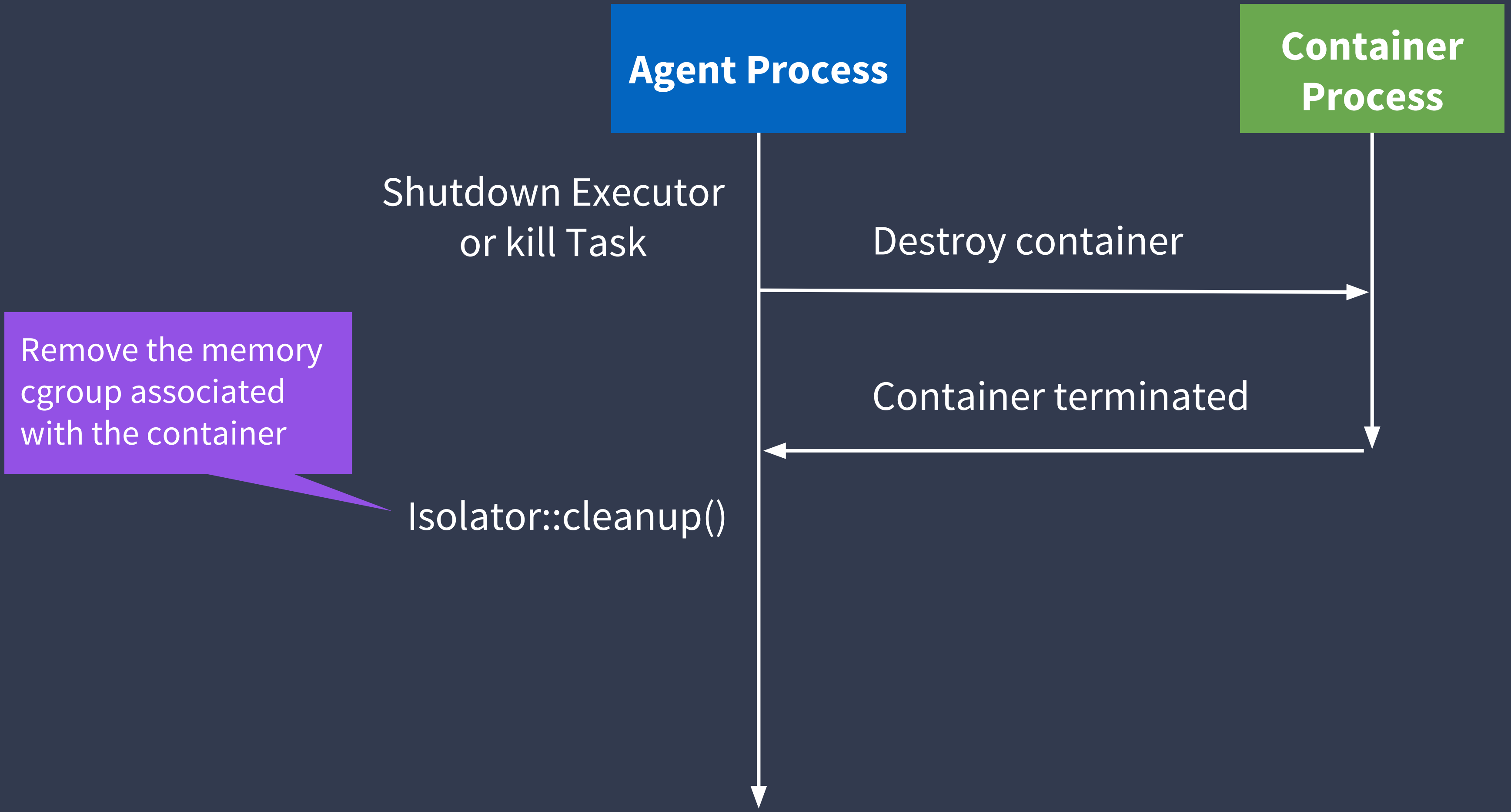

Agent Process
* Create a cgroup for the container in memory cgroup hierarchy: /sys/fs/cgroup/memory/mesos/…
* Start listening for OOM event
LaunchInfo = Isolator::prepare()
Launcher creates Subprocess
Move 'pid' to the memory cgroup just created
Container Process
Isolator::isolate(pid)
Block on pipe
Signal the Child to continue
Invoke '**LaunchInfo.script**'
execve()
Exec the executor

Unified Containerizer

Isolator example: cgroups memory isolator


Agent Process
Container Process
Sending a new Task to Executor, ‘resources’ of the Executor changes
Isolator::update()
Change cgroup control: memory.limit_in_bytes
Send Task to Executor

Isolator example: cgroups memory isolator



Unified Containerizer

Built-in isolators

Cgroups isolators:	`cgroups/cpu, cgroups/mem, ...`
Disk isolators:	`disk/du, disk/xfs`
Filesystem isolators:	`filesystem/posix, filesystem/linux`
Volume isolators:	`docker/volume`
Network isolators:	`network/cni, network/port_mapping`
GPU isolators:	`gpu/nvidia`

...... and more! Need your contribution!

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

Provisioner

- Manage container images
  - Store: fetch and cache image layers
  - Backend: assemble rootfs from image layers
    - E.g., copy, overlayfs, bind, aufs
- Store can be extended
  - Currently supported: Docker, Appc
  - Plan to support: OCI (ongoing), CVMFS
  - Custom fetching (e.g., p2p)

Unified Containerizer

Demo

Container network support

- Support <u>C</u>ontainer <u>N</u>etwork <u>I</u>nterface (CNI) from 1.0
  - A spec for container networking
  - Supported by most network vendors

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

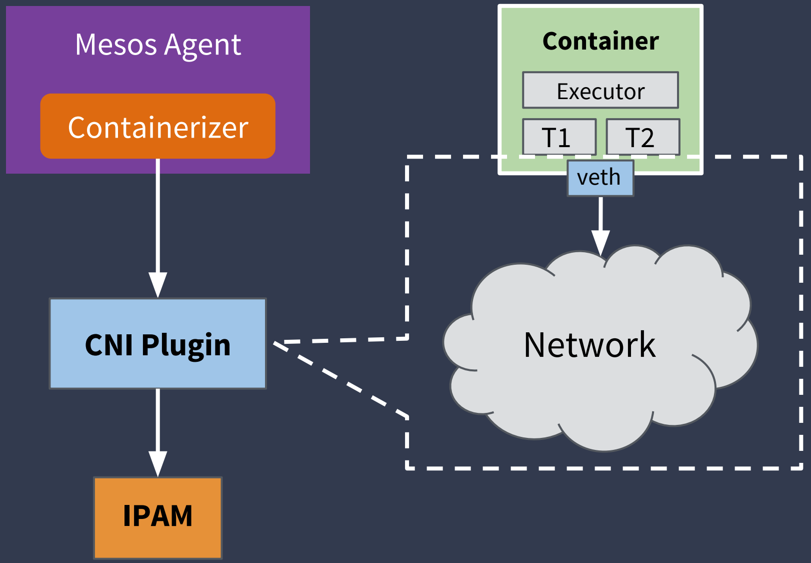
Unified Containerizer

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

Extensions

Launcher

- Custom container processes management

Isolator

- Extension to the life cycle of a container

Provisioner

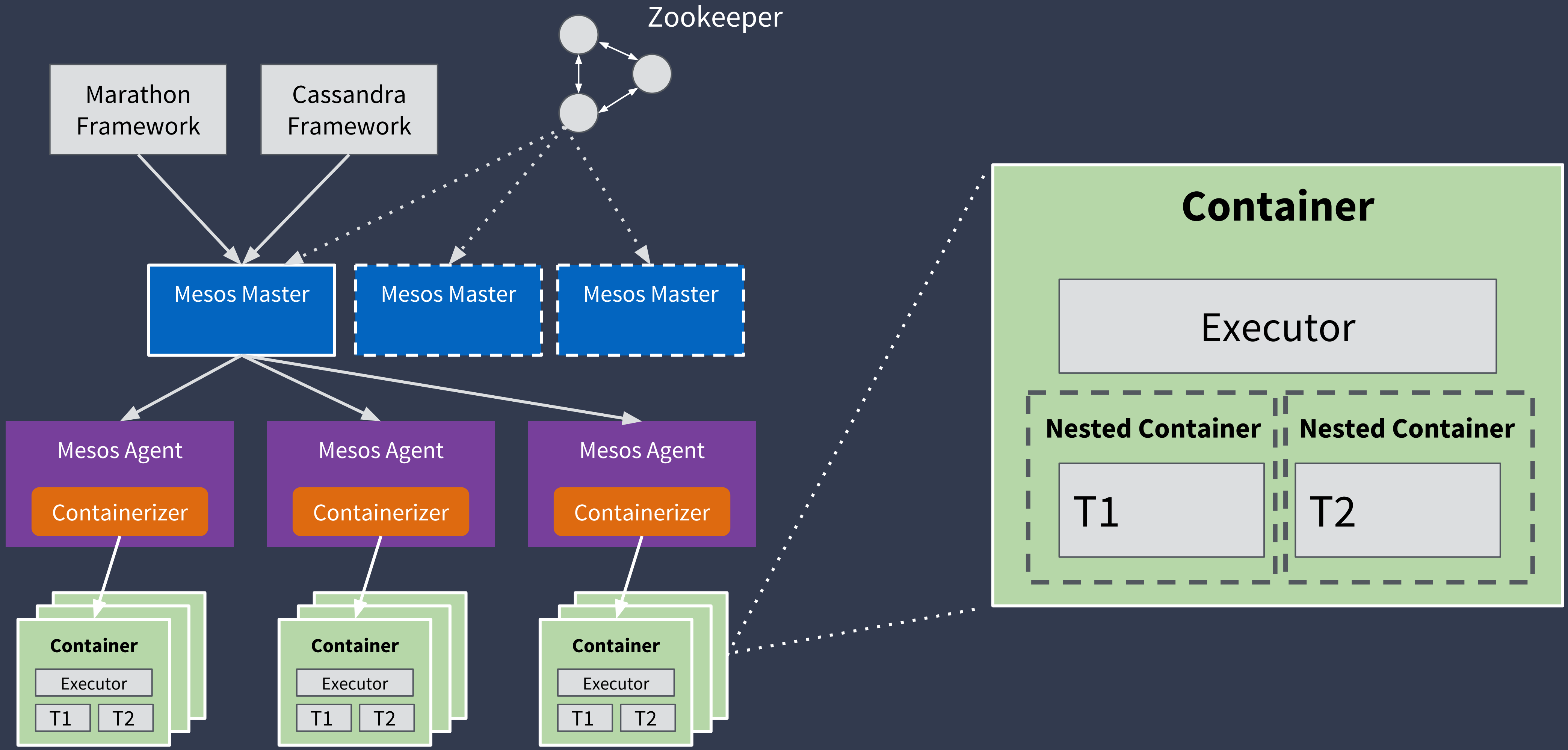
- New type of images
- Custom fetching and caching

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



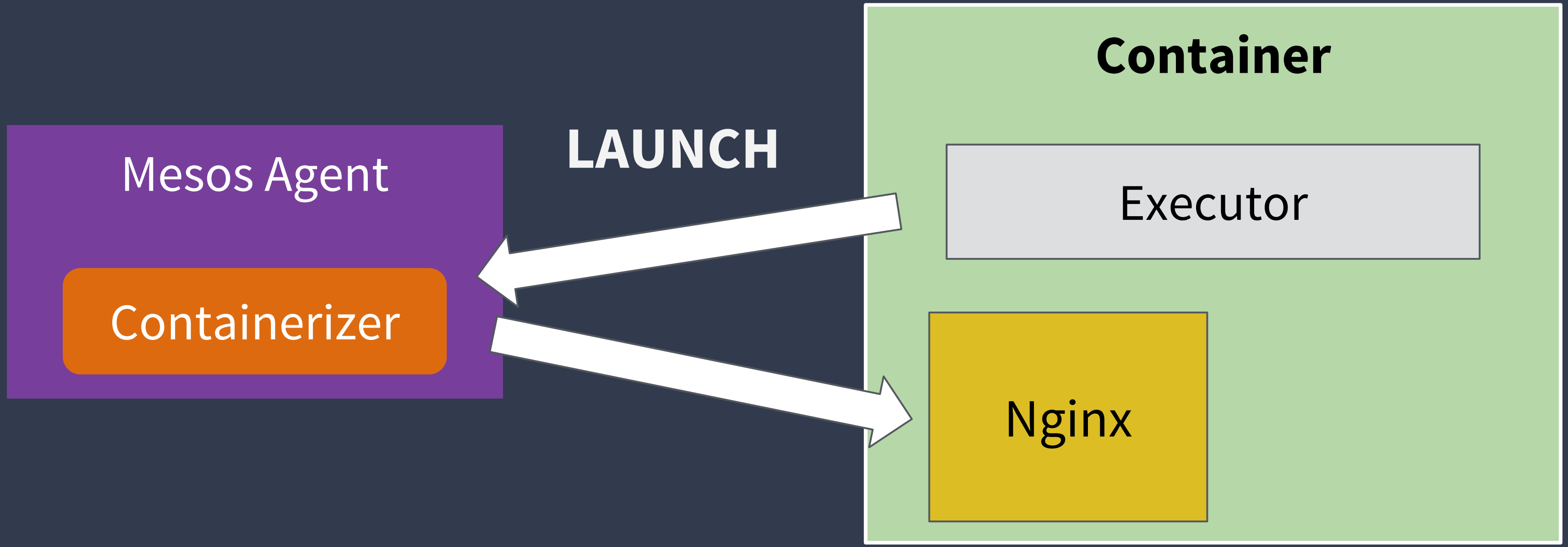
Nested container support

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;
  }
}
```

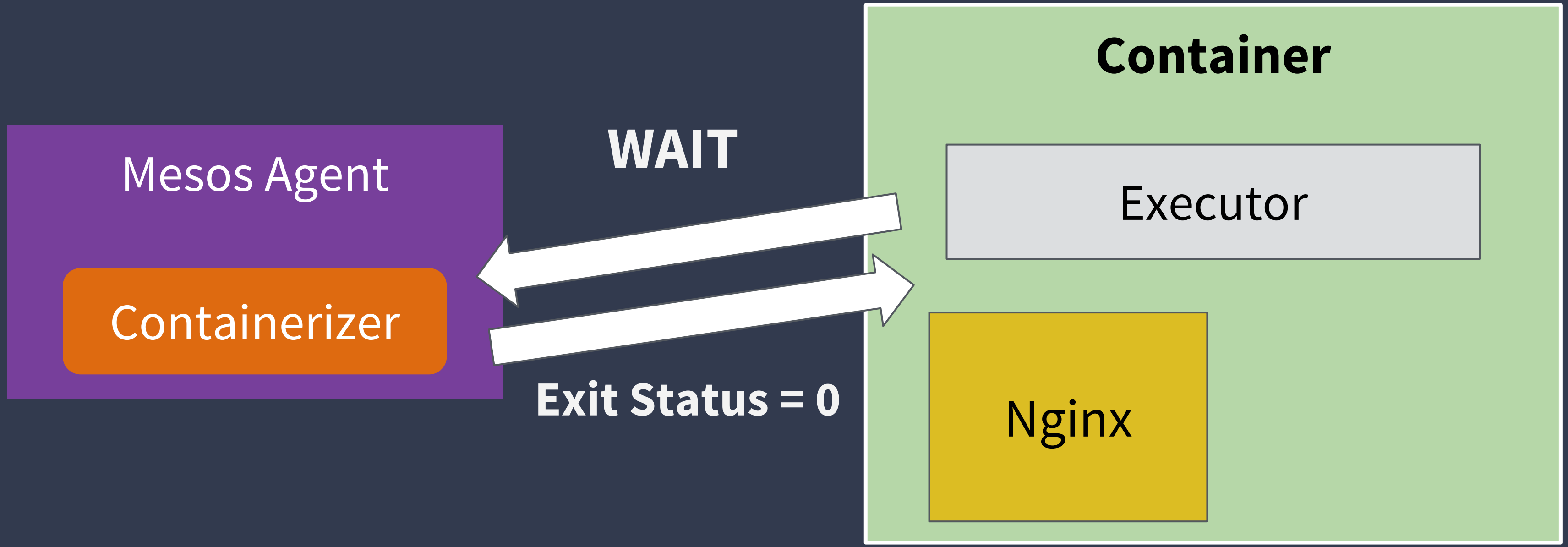
Nested container support

Launch nested container



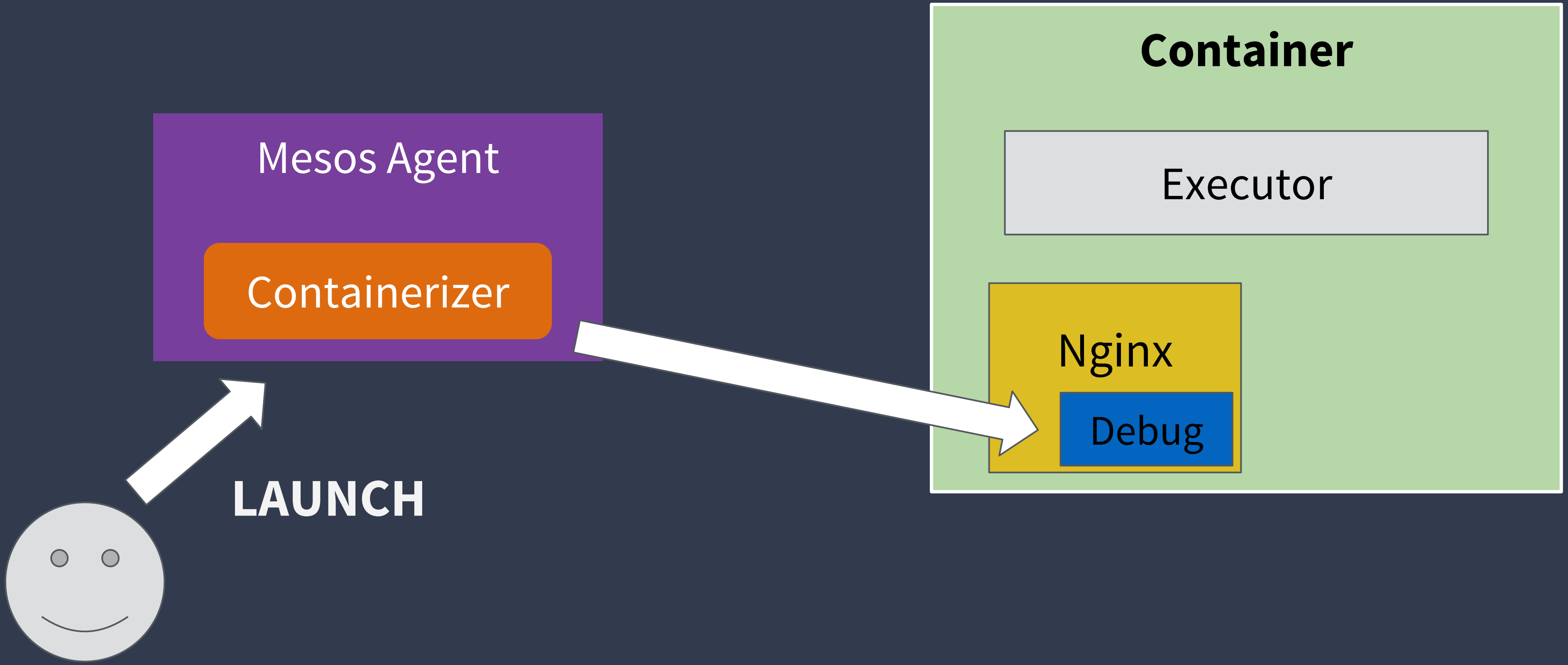
Nested container support

Watch nested container



Nested container support

Arbitrary levels of nesting



Unified Containerizer

Demo

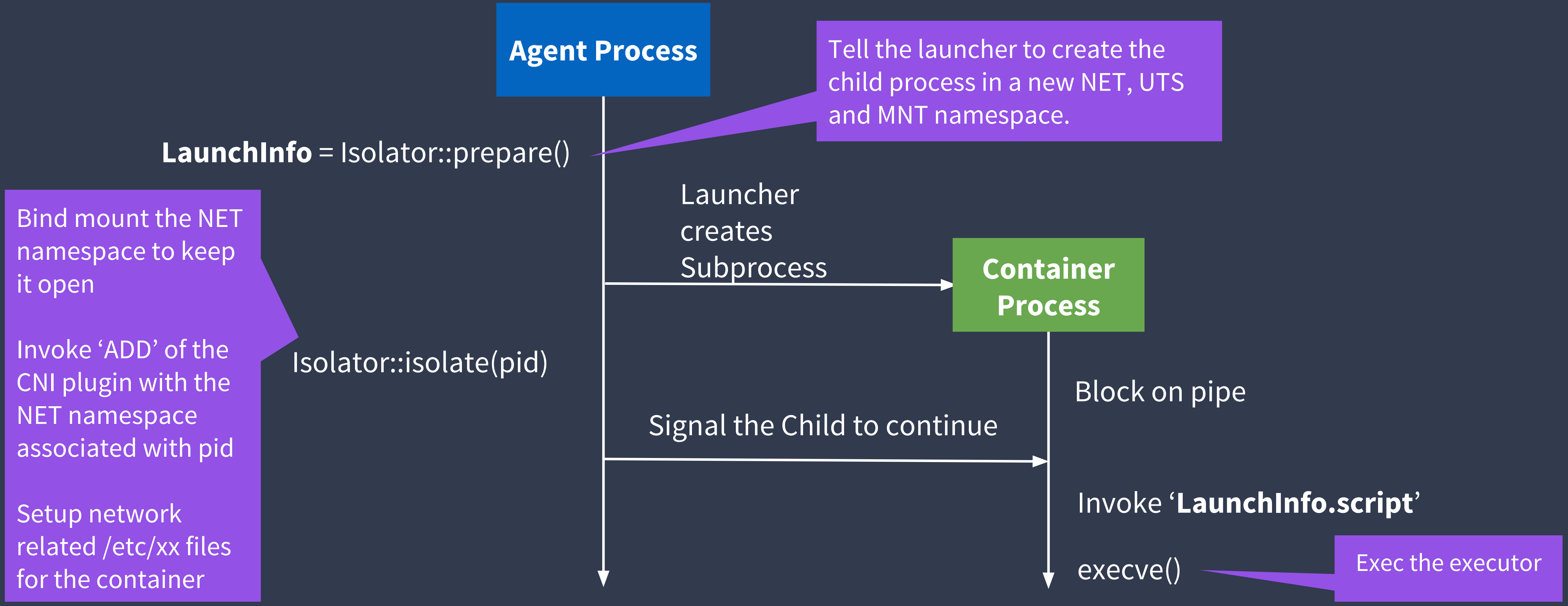
Summary

- Mesos: state of the art container orchestrator
  - Production ready
  - Proven scalability
  - Highly customizable and extensible
- Containerization in Mesos
  - Pluggable architecture
  - Native support for Docker/Appc images (w/o Docker daemon or rkt)
  - Container network: CNI
  - Container storage: DVD
  - Nested container support

Questions?

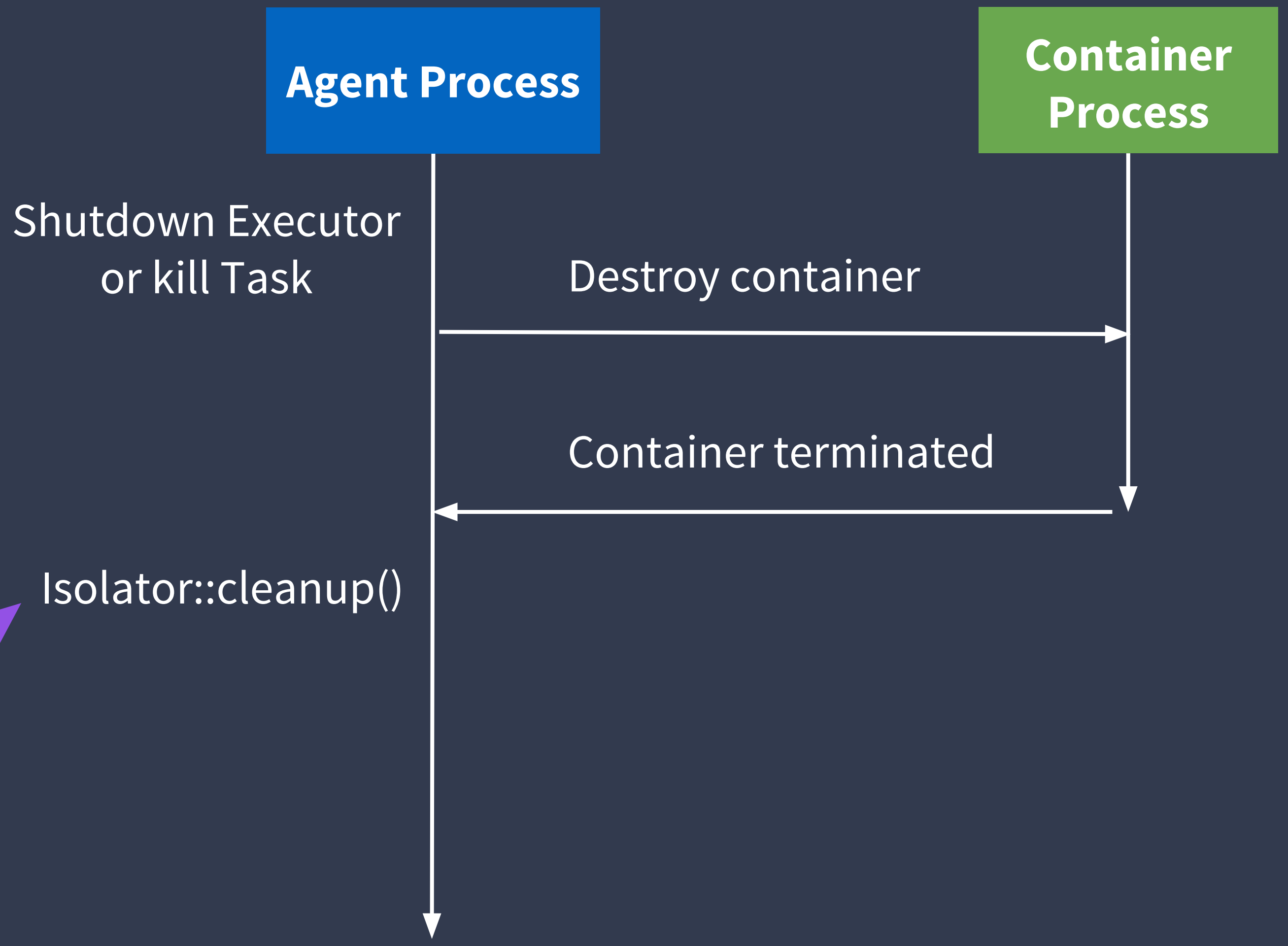
Unified Containerizer

CNI support using an isolator


Agent Process
Tell the launcher to create the child process in a new NET, UTS and MNT namespace.
LaunchInfo = Isolator::prepare()
Launcher creates Subprocess
Container Process
Bind mount the NET namespace to keep it open
Invoke 'ADD' of the CNI plugin with the NET namespace associated with pid
Setup network related /etc/xx files for the container
Isolator::isolate(pid)
Block on pipe
Signal the Child to continue
Invoke '**LaunchInfo.script**'
execve()
Exec the executor

Unified Containerizer

CNI support using an isolator



Invoke ‘DEL’ of the CNI plugin with the NET namespace handle

Umount and remove the NET namespace handle

Mesos, as one of the most powerful container orchestrators, greatly simplifies the deploy, provision and execution of containerized workloads. It automates the distribution of preprovisioned container images, injection of configuration, scheduling onto machines, life-cycle-management, and monitoring of applications, microservices, and jobs in the cloud.

In this talk, Jie Yu will first give you an overview about Mesos and its powerful API which allows users to easily deploy their stateless and stateful services. Then, Jie will talk about how containers are managed in Mesos. In particular, Jie will provide a deep dive into the unified containerizer which is first introduced in Mesos 1.0.

Jie will show some of the new container networking and storage features that are built recently, and how they benefit from the pluggable and extensible architecture of the unified containerizer. Finally, Jie will discuss the future of container support in Mesos.