v1.11

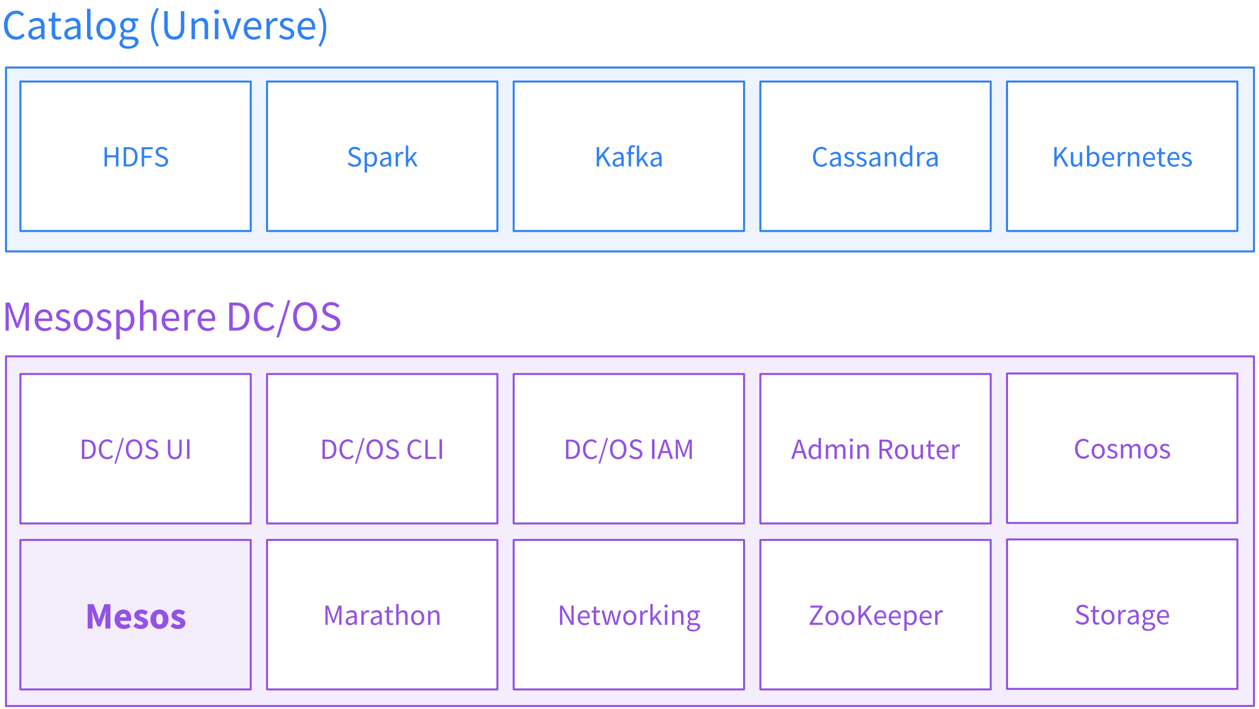
APACHE MESOS

MESOSPHERE

AGENDA

1. Overview
2. Mesos Architecture
3. Offer Cycle
4. Allocators
5. Containerizers and Isolators
6. Resources, Roles, Attributes, Quotas

DC/OS COMPONENTS



Catalog (Universe)

HDFS | Spark | Kafka | Cassandra | Kubernetes

Mesosphere DC/OS

DC/OS UI | DC/OS CLI | DC/OS IAM | Admin Router | Cosmos

Mesos | Marathon | Networking | ZooKeeper | Storage

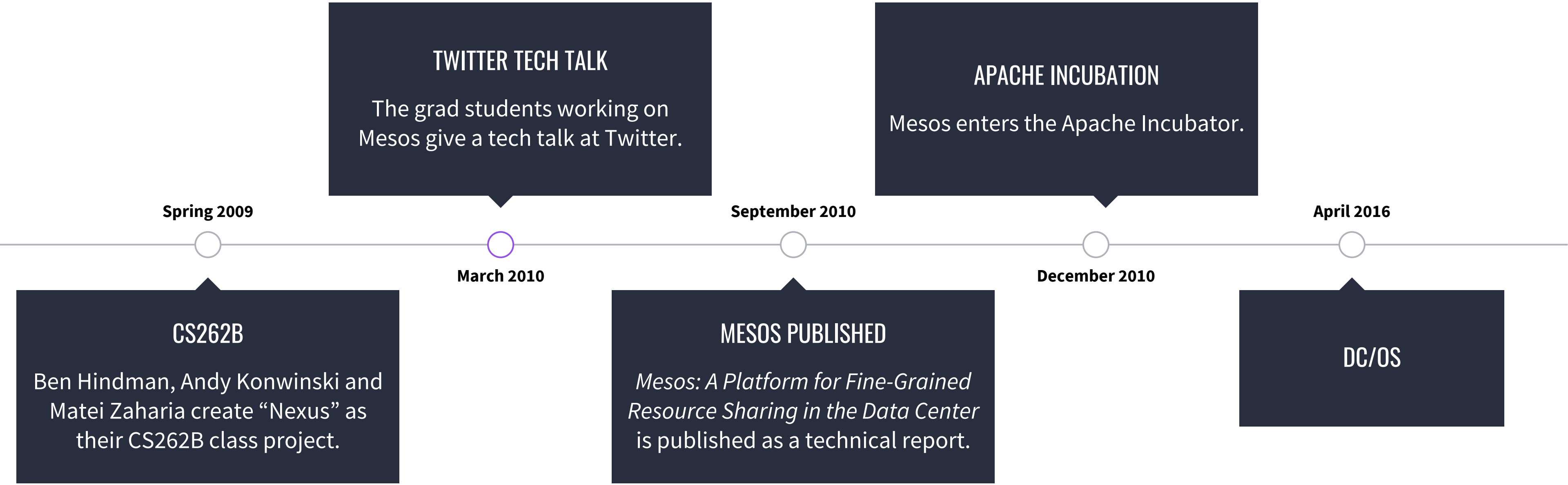
Mesos

OVERVIEW

OVERVIEW

- Mesos is an open-source resource cluster manager
- Developed at the UC Berkeley AMPLab
- Battle-tested and hardened at Twitter
- Scalable to tens of thousands of nodes

HISTORY



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.

April 2016

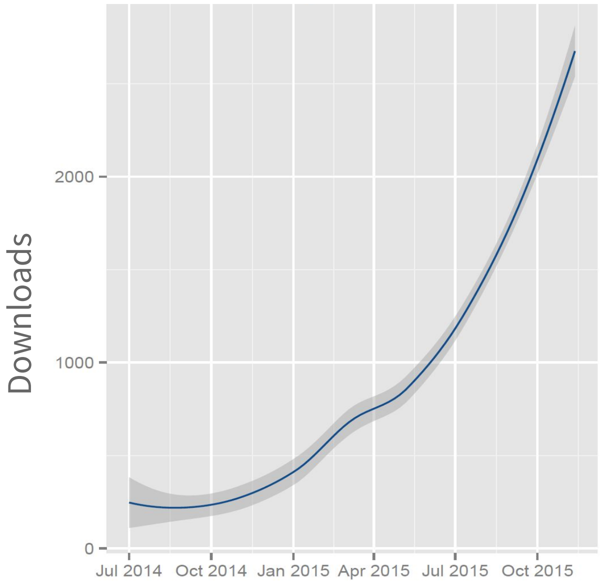
DC/OS

OVERVIEW

Designed to be flexible

- **Aggregates all resources** in the datacenter for modern apps
- **Intentionally simple** to enable massive scalability
- **Handles different types of tasks** - long running, batch, and real-time
- **Two-level scheduler architecture** enables multiple scheduling logics
(a key challenge at Google)
- **Extensible** to work with new technologies

Gaining massive adoption



Mesos daily downloads, July 2014 - November 2015

airbnb

NETFLIX

PayPal

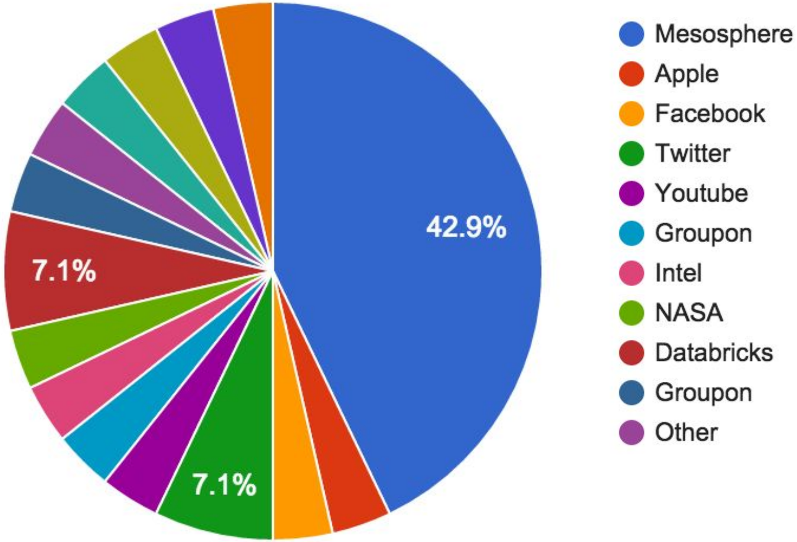
yelp

TWO SIGMA

MESOS CONTRIBUTORS

Project Management Committee (PMC): 43

Committers, by Company/Affiliation



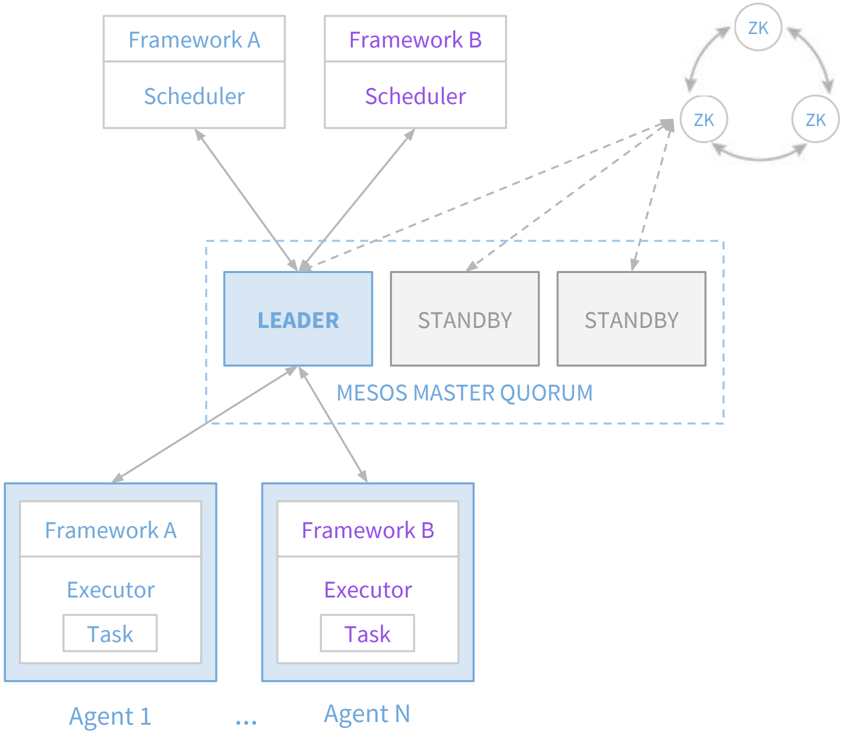
Mesosphere has 83% of the commits to the Apache Mesos Project

Mesos

ARCHITECTURE

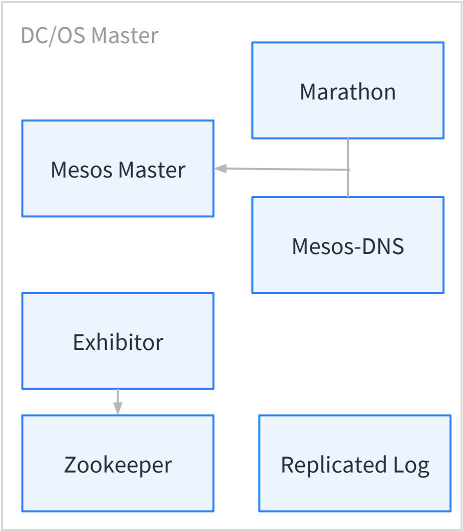
MESOS ARCHITECTURE

- Masters
- Agents
  - Resources
    - Roles
    - Attributes
- Frameworks
  - Schedulers
  - Executors
    - Tasks



MESOS MASTER

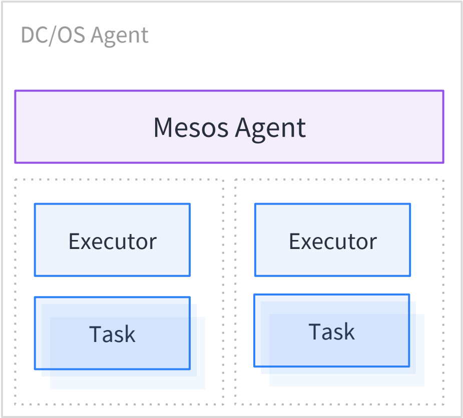
- Cluster resource manager which manages task lifecycle
- Collects resources reported by Mesos agents and makes resource offers to Mesos schedulers
- Hosts the Mesos user interface
- Persists state (e.g., quota) via replicated log
- Uses zookeeper to elect a leader and provide failover in the event of a failure.
    - Deploy Mesos masters in odd numbers (starting at three) to ensure quorum and avoid split brain



MESOS AGENT

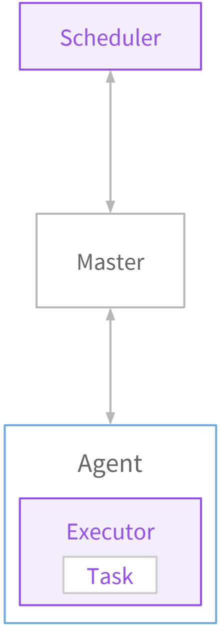
- A process running on agent nodes that manages the executors, tasks, and resources of that node:

- Mesos agents have two primary functions:
  - Manage and offer some or all of the local resources on the Mesos agent node
  - Launch and manage the executors using containers to run a task
- Pre Mesos 1.0, agents were named slaves



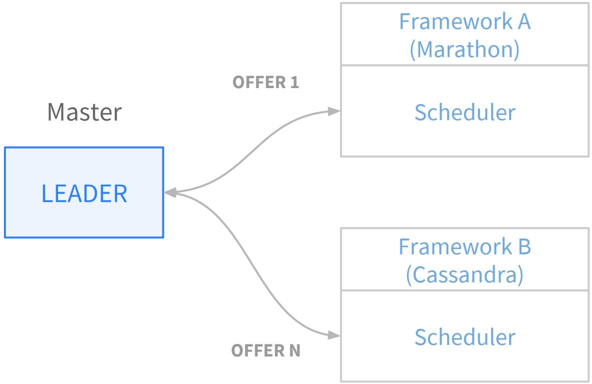
FRAMEWORKS

- Frameworks are distributed applications that run on Mesos. They consist of two components:
  - Scheduler
  - Executor



FRAMEWORK SCHEDULER

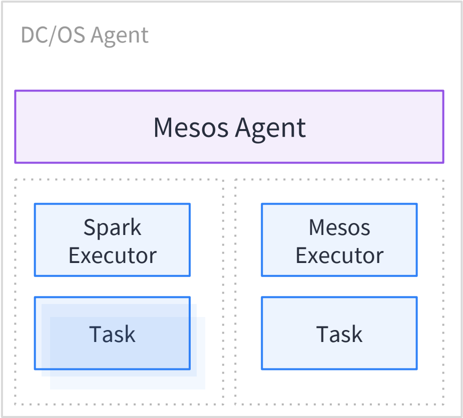
- A framework scheduler decides which Mesos resource offers to accept or reject to complete the work of that specific framework
- The scheduler makes these decisions by:
  - Examining the offer’s
    - Resources
    - Attributes
  - Matching the scheduler’s resource needs and placement constraints to the offer



FRAMEWORK EXECUTOR

Executors perform work on behalf of the framework on the agent nodes.

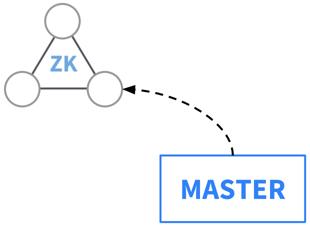
- An executor runs within a container
- An executor can run multiple tasks



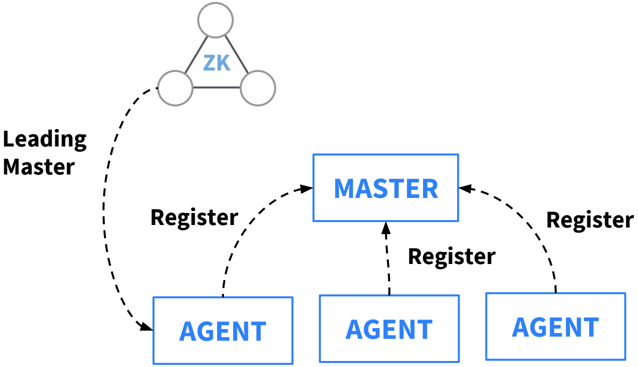
Mesos

OFFER CYCLE

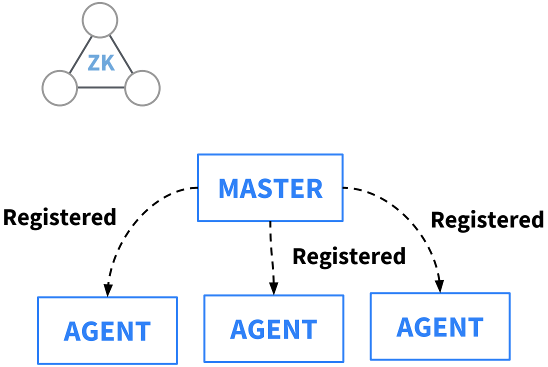
MASTER STARTUP



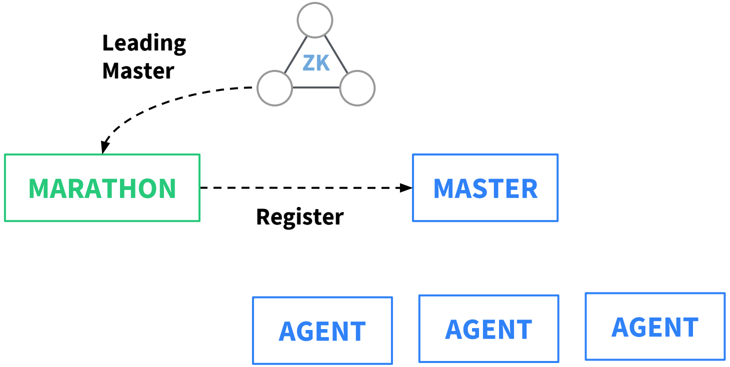
AGENT STARTUP



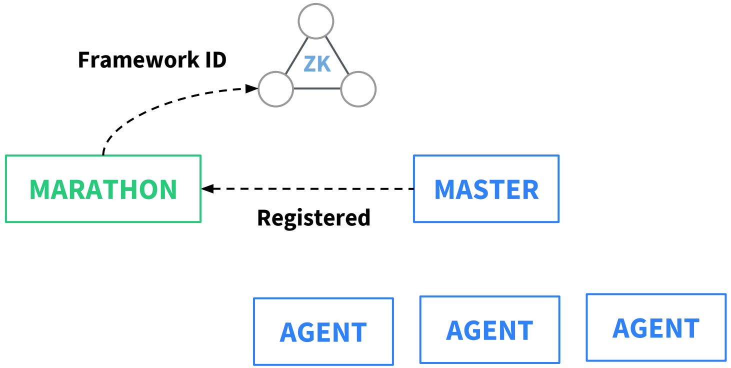
AGENT STARTUP



FRAMEWORK STARTUP



FRAMEWORK STARTUP



Framework ID

ZK

MARATHON

MASTER

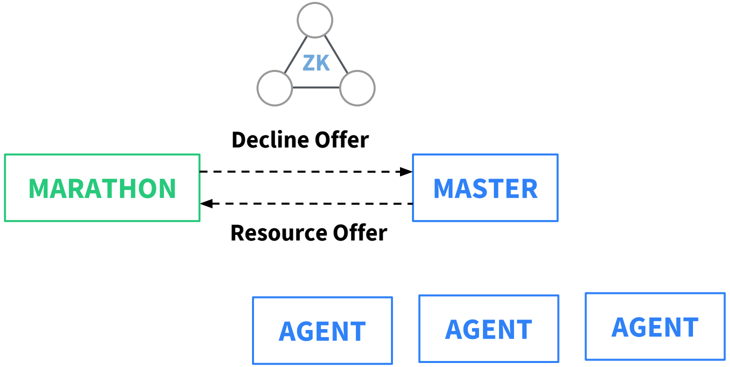
Registered

AGENT

AGENT

AGENT

OFFER CYCLE



ZK

Decline Offer

MARATHON

MASTER

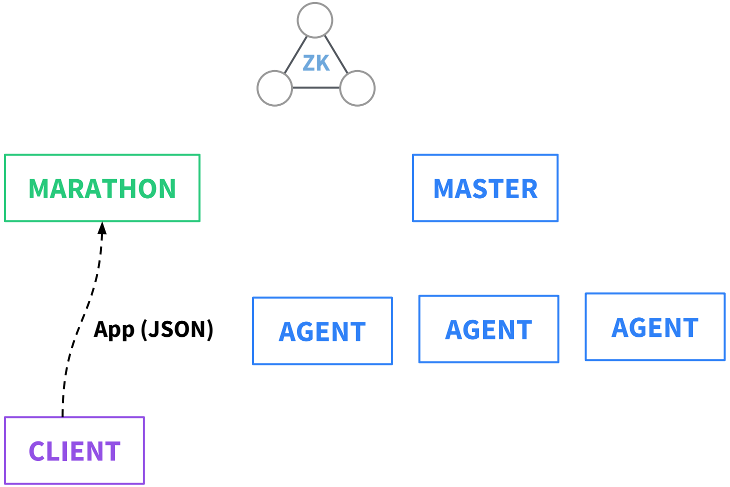
Resource Offer

AGENT

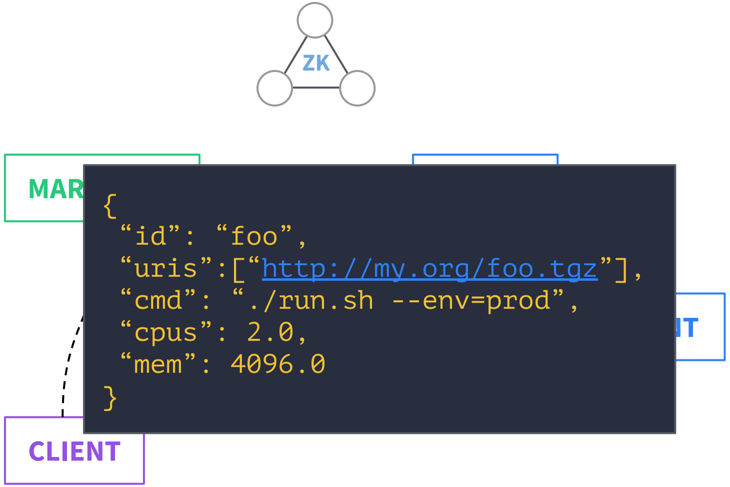
AGENT

AGENT

OFFER CYCLE

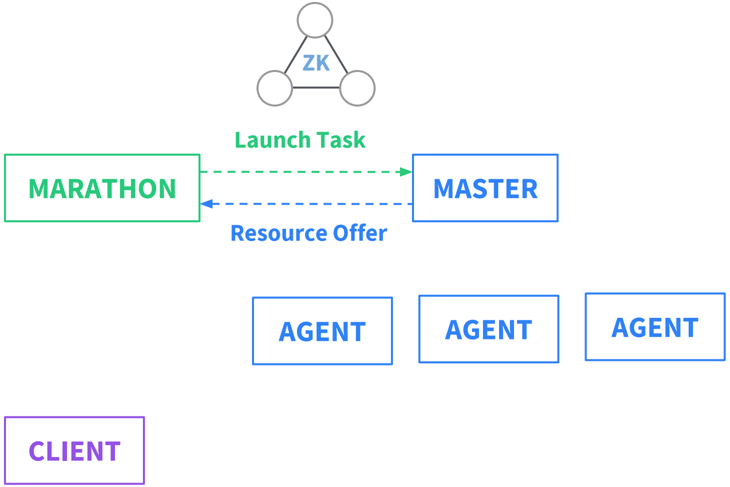


OFFER CYCLE

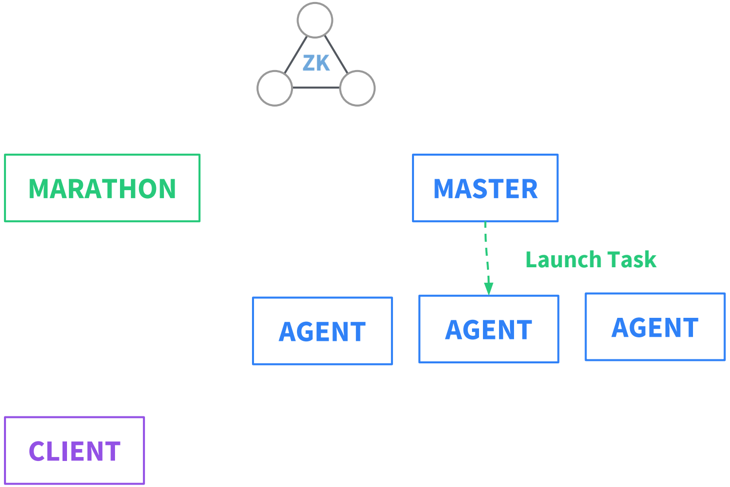


```
{
 “id”: “foo”,
 “uris”:[“http://my.org/foo.tgz”],
 “cmd”: “./run.sh --env=prod”,
 “cpus”: 2.0,
 “mem”: 4096.0
}
```

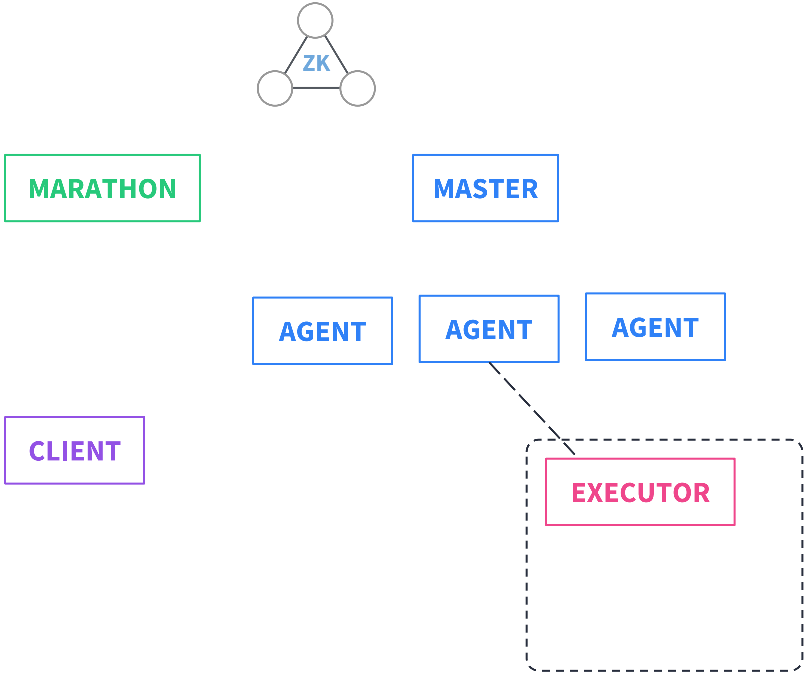
OFFER CYCLE


ZK
Launch Task
MARATHON
MASTER
Resource Offer
AGENT
AGENT
AGENT
CLIENT

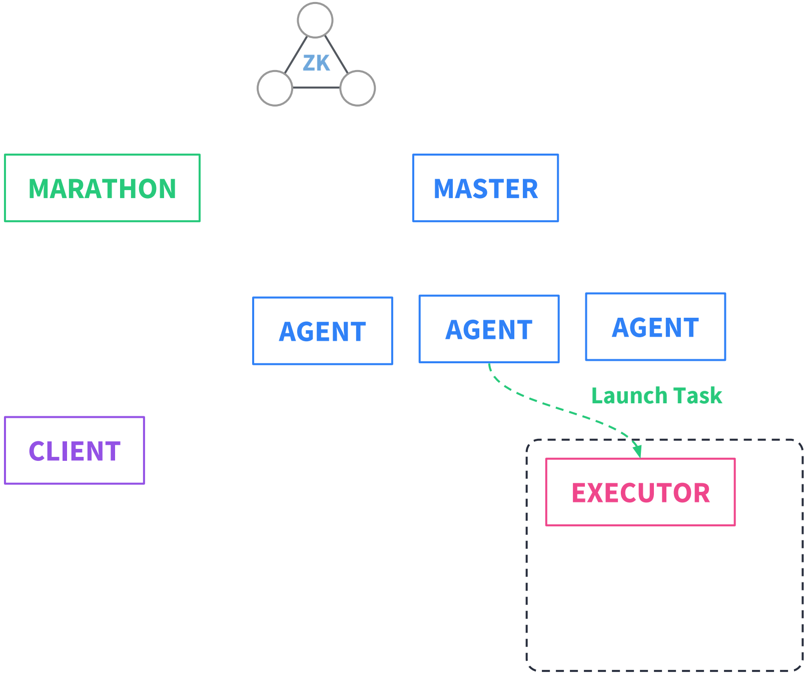
OFFER CYCLE



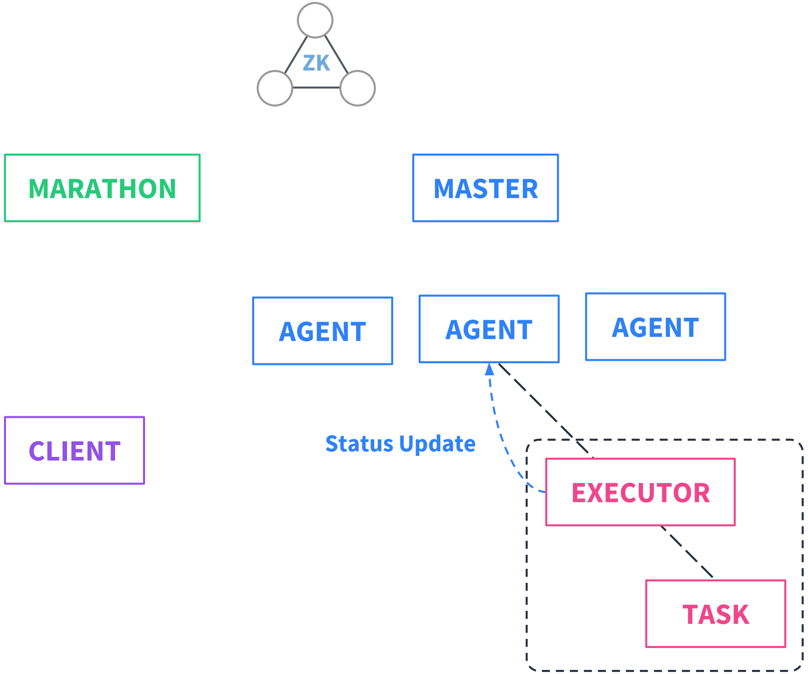
OFFER CYCLE



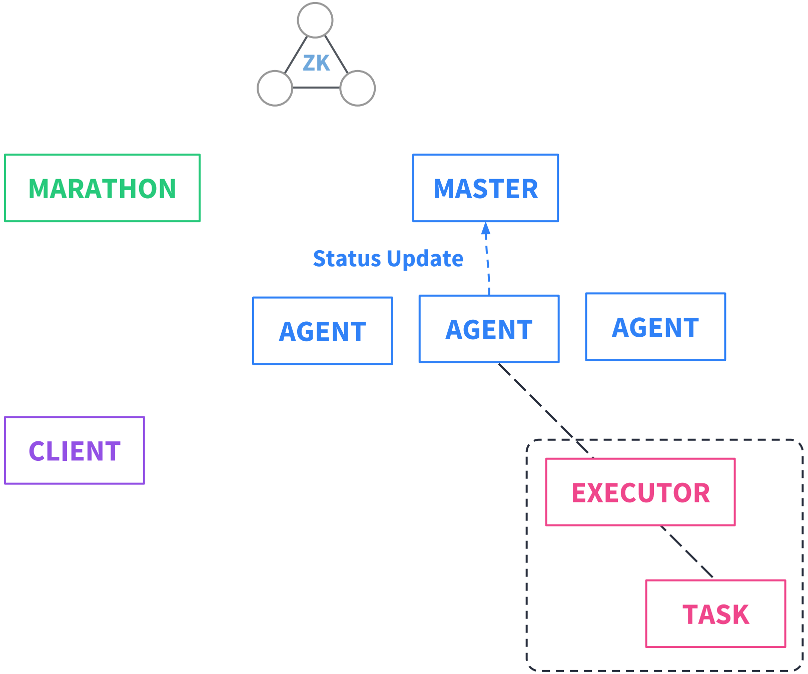
OFFER CYCLE


ZK
MARATHON
MASTER
AGENT
AGENT
AGENT
Launch Task
CLIENT
EXECUTOR

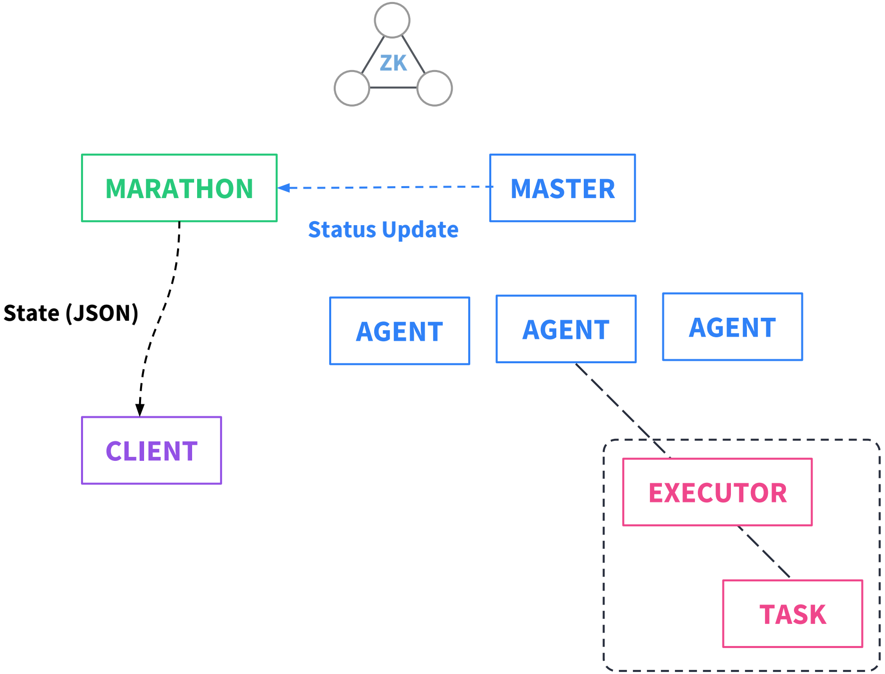
OFFER CYCLE


ZK
MARATHON
MASTER
AGENT
AGENT
AGENT
CLIENT
Status Update
EXECUTOR
TASK

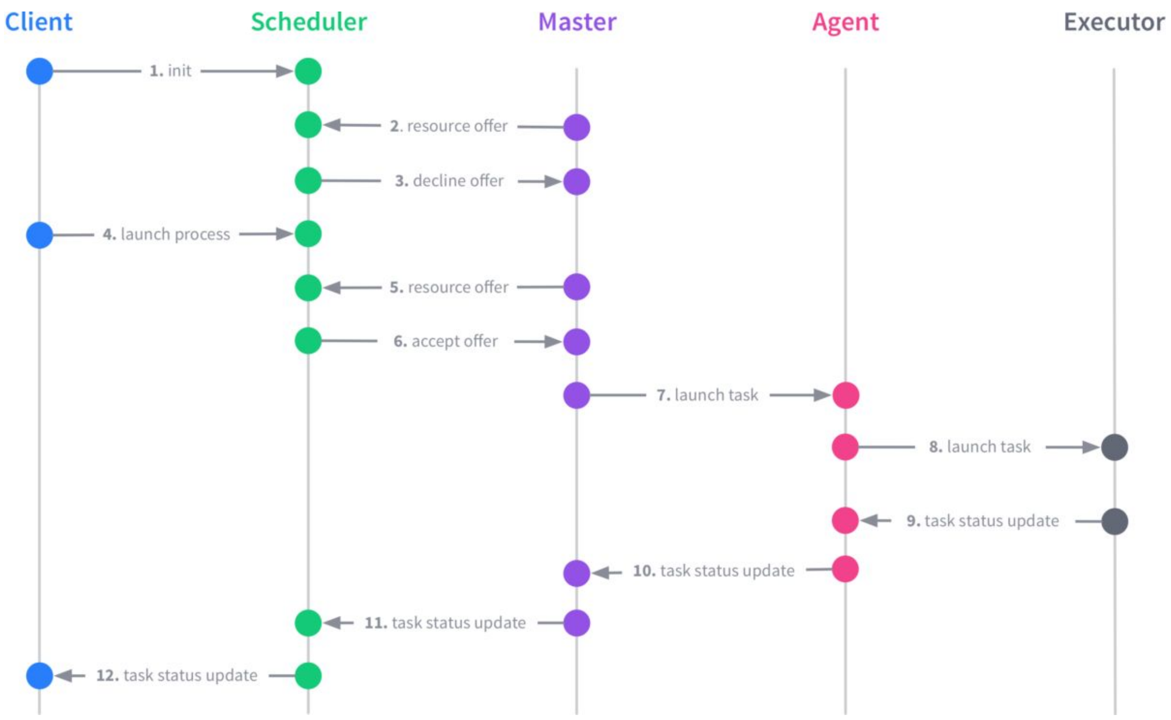
OFFER CYCLE


ZK
MARATHON
MASTER
Status Update
AGENT
AGENT
AGENT
CLIENT
EXECUTOR
TASK

OFFER CYCLE


ZK
MARATHON
MASTER
Status Update
State (JSON)
AGENT
AGENT
AGENT
CLIENT
EXECUTOR
TASK

SUMMARY

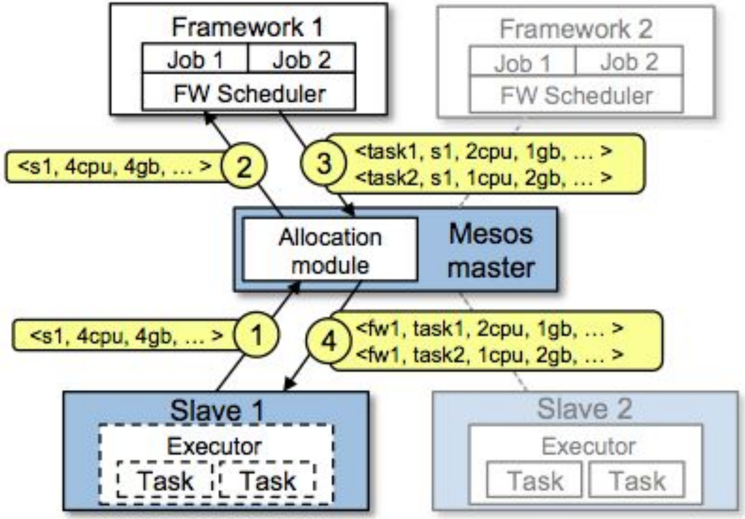

Client
Scheduler
Master
Agent
Executor
1. init
2. resource offer
3. decline offer
4. launch process
5. resource offer
6. accept offer
7. launch task
8. launch task
9. task status update
10. task status update
11. task status update
12. task status update

Advanced

ALLOCATORS

RESOURCE ALLOCATORS

- **Allocator module**: pluggable component that can call a custom allocator
- Mesos master uses allocator module to determine which frameworks to make resource offers to
- The default is the hierarchical based Dominant Resource Fairness (DRF) algorithm



DOMINANT RESOURCE FAIRNESS

- Dominant Resource Fairness is the default Resource Allocation policy in Mesos
- DRF is a modified fair share “Max-Min” algorithm designed to work with multiple types of resources in a heterogeneous environment.
- Ensures each framework receives a fair share of resources most needed by that framework.
- https://www.cs.berkeley.edu/~alig/papers/drf.pdf

Mesos

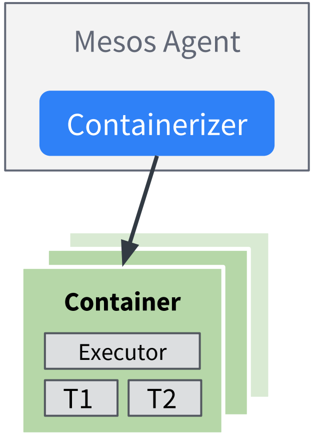
CONTAINERIZERS & ISOLATORS

OVERVIEW

Containerizers are Mesos components responsible for launching containers

Containerization provides:

- Isolation of a task from other running tasks.
- Containing tasks to run in limited resource runtime environment
- Controlling task’s individual resources (e.g, CPU, memory) programmatically
- Running software in a pre-packaged file system image, allowing it to run in different environments



SUPPORTED CONTAINERIZERS

Universal Container Runtime (UCR) containerizer

- Using standard OS features (e.g., cgroups, namespaces)
- Pluggable architecture allowing customization and extension
- Supports Docker file format, pods, GPU, dcos-metrics

Docker containerizer

- Delegate to Docker daemon
- Does not support pods, GPU

UNIVERSAL CONTAINER RUNTIME

The Universal Container Runtime (UCR) provides lightweight containerization and
resource isolation of executors using Linux-specific functionality
such as control groups (cgroups) and namespaces

UCR is composable so operators can selectively enable different isolators

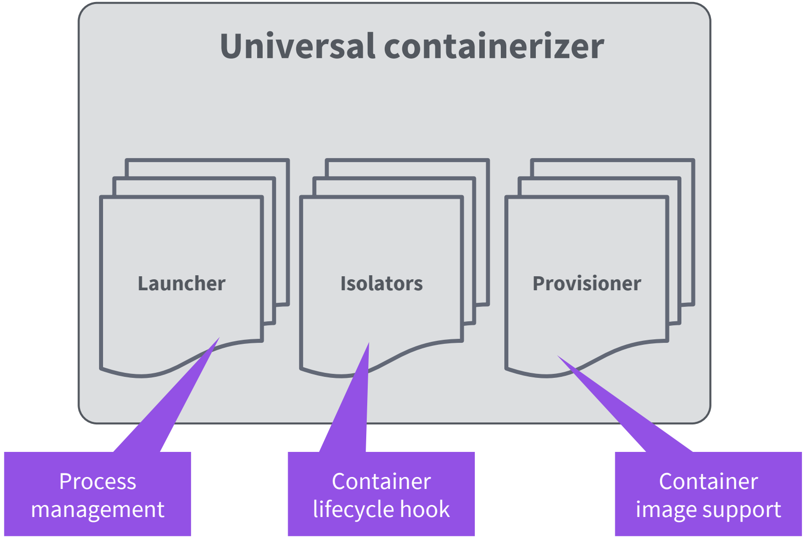
Isolators create an environment for containers where resources like CPU, network, storage and memory can be isolated from other containers.

UNIVERSAL CONTAINER RUNTIME

Enhancements to the Mesos Containerizer to allow support launching specific container formats (Docker, AppC, OCI (future), etc)

- Reduces need to maintain and update multiple containerizers
- Support multiple container formats with a single containerizer

Image provisioner component added to the Mesos containerizer - responsible for pulling, caching, and preparing container root filesystems



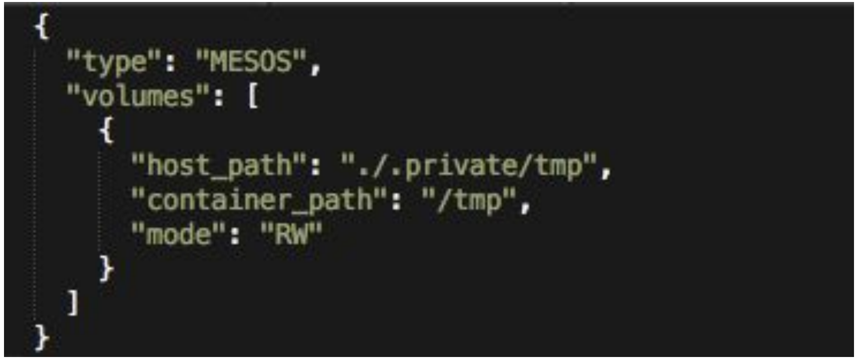
BUILT-IN ISOLATORS

- 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**

EXAMPLE: SHARED FILE SYSTEM ISOLATOR

- Enables modifications to each container's view of the shared filesystem
- Selectively make parts of the shared filesystem private to each container.
- Leveraged by the framework or by using the --default_container_info agent flag

```
$ mesos-slave --master=<master>:5050
--default_container_info=file://container.json
```



```
{
  "type": "MESOS",
  "volumes": [
    {
      "host_path": "./.private/tmp",
      "container_path": "/tmp",
      "mode": "RW"
    }
  ]
}
```

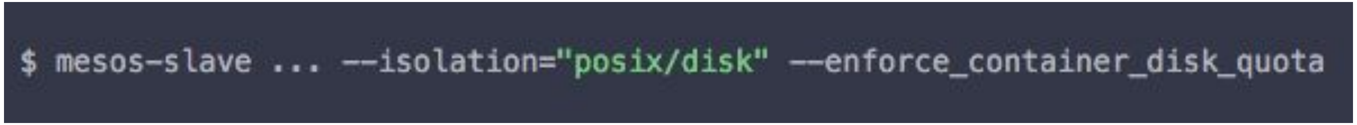
EXAMPLE: PID NAMESPACE ISOLATOR

Used to isolate each container in a separate PID namespace for:

- **Visibility:** Processes running in the container (executor and descendants) are unable to see or signal processes outside the namespace
- **Clean Termination:** Termination of the leading process in a PID namespace will result in the kernel terminating all other processes in the namespace

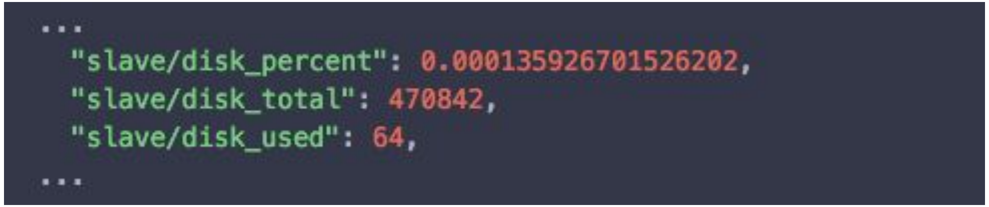
EXAMPLE: POSIX DISK ISOLATOR

- Disk isolation for Linux and OS X
- Reports disk usage for task sandbox and optionally enforces disk quotas



```
$ mesos-slave ... --isolation="posix/disk" --enforce_container_disk_quota
```

- Reports disk usage for each sandbox by periodically running the du command every 15 secs by default
  - Disk usage can be retrieved from the resource statistics endpoint: /monitor/statistics.json



```
...
  "slave/disk_percent": 0.000135926701526202,
  "slave/disk_total": 470842,
  "slave/disk_used": 64,
...
```

DOCKER CONTAINERIZER

Translates task/executor launch and destroy calls to Docker CLI commands

When launching a task, the Docker containerizer:

1) Fetches all files specified in CommandInfo and puts them into the sandbox
2) Pulls the Docker image from the remote container registry
3) Runs the Docker image with the Docker executor, maps the sandbox directory into the Docker container and sets the directory mapping to the MESOS_SANDBOX environment variable
4) The executor streams container logs to stdout & stderr files in the sandbox
5) On container exit or containerizer destroy, stops and removes the Docker container

CONTAINER COMPARISON

Container feature comparison

Feature	Mesos (UCR)	Docker
CPU isolation	✓	✓
Memory limits	✓	✓
PID namespace	✓	✓
Filesystem isolation	✓	✓
POSIX disk quotas	✓	✗
SDN integration (Calico)	✓	✓ (with Mesos)
Recovery	✓	✗
Runtime upgrades	✓	✓?
Container resizing	✓	✗

CONTAINER ENV VARIABLES

```
MESOS_AGENT_ENDPOINT=10.0.3.145:5051
HOST=10.0.3.145
PORT0=23498
MESOS_TASK_ID=test.a1521dd1-4e42-11e6-ada1-76f15f3b9194
MESOS_EXECUTOR_ID=test.a1521dd1-4e42-11e6-ada1-76f15f3b9194
MESOS_NATIVE_JAVA_LIBRARY=/opt/mesosphere/packages/mesos--64cc034646a9249221949adc4a64ccc54d41e823/lib/libmesos-0.28.1.so
PORTS=23498
MESOS_SLAVE_PID=slave(1)@10.0.3.145:5051
MESOS_FRAMEWORK_ID=044e2fb4-732b-4e61-ae05-8e144c5d06fe-0000
MARATHON_APP_LABELS=
MARATHON_APP_ID=/test
MESOS_SLAVE_ID=044e2fb4-732b-4e61-ae05-8e144c5d06fe-S4
MARATHON_APP_RESOURCE_CPUS=1.0
MESOS_SANDBOX=/var/lib/mesos/slave/slaves/044e2fb4-732b-4e61-ae05-8e144c5d06fe-S4/frameworks/044e2fb4-732b-4e61-ae05-8e144c5d06fe-0000/executors/test.a1521dd1-4e42-11e6-ada1-76f15f3b9194/runs/880c8111-90de-4c5d-9316-c018d66e2ed8
...
```

Mesos

RESOURCES, ROLES, ATTRIBUTES, QUOTAS

RESOURCES

Resources are the fundamental abstraction in Mesos. Agent nodes advertise available resources when connecting to a Master.

Resources can be grouped by roles to partition cluster resources. Example use cases:

- agents that are public-facing versus internal
- delineate between different environments (prod vs dev)

By default, resources are allocated with the * role

RESOURCES

Resources represent what an agent has to offer (a resource offer)

The following resources have predefined behavior:

- cpus, mem (MiB), disk (MiB), ports, gpus

An agent without `cpus` and `mem` resources will never have its resources advertised to any frameworks.

RESOURCES (STATIC)

```
--resources='cpus:24;mem:24576;disk:409600;ports:[21000-24000];ssd:{a,b,c}'
```

- Three types of resources are specified: three scalars, a range, and a set.
  - A scalar called `cpus`, with the value `24`
  - A scalar called `mem`, with the value `24576`
  - A scalar called `disk`, with the value `409600`
  - A range called `ports`, with values `21000` through `24000` (inclusive)
  - A set called `ssd`, with the values `a`, `b` and `c`

RESOURCES (DYNAMIC)

Dynamic reservation enables operators and authorized frameworks to reserve and unreserve resources after slave-startup

`Offer::Operation::Reserve` and `Offer::Operation::Unreserve` messages are available for frameworks to send back via the acceptOffers API as a response to a resource offer

`/reserve` and `/unreserve` HTTP endpoints allow operators to manage dynamic reservations through the master.

ROLES

Roles are used to reserve resources for frameworks

- Roles are defined on the agent node with specific resources (cpu, mem)
- Frameworks register with specific roles (*, slave_public)
- Default role is *, considered “unreserved”
- Weights can be used to control the relative share of cluster resources offered to different roles

Use cases:

- Dividing cluster resources across different organizations, environments
- Classifying between different agent types (private, public)

ATTRIBUTES

Attributes are key-value pairs (whose value is optional) that Mesos passes along when it sends offers to frameworks

Attributes can be used by frameworks to make scheduling decisions

Attributes are configured on the agent node via command line flags or environment variables

```
attributes : attribute ( ";" attribute )*
```

```
attribute : text ":" ( scalar | range | text )
```

AGENT ATTRIBUTES

```
--attributes='rack:abc;zone:west;os:centos8;level:10;keys:[1000-1500]'
```

We specified three types of attributes: three texts, a scalar, and a range

- rack with text value abc
- zone with text value west
- os with text value centos5
- level with scalar value 10
- keys with range value 1000 through 1500 (inclusive)

QUOTAS

A quota specifies a minimum amount of resources that the role is guaranteed to receive (unless the total resources in the cluster are less than the configured quota resources)

Quota resources are not tied to a particular agent, unlike resource reservations

Quotas can only be configured by operators, via the HTTP endpoint described below; dynamic reservations can be made by frameworks, provided the framework’s principal is authorized to make reservations.

```
{
   "role": "role1",
   "guarantee": [
     {
       "name": "cpus",
       "type": "SCALAR",
       "scalar": { "value": 12 }
     },
     {
       "name": "mem",
       "type": "SCALAR",
       "scalar": { "value": 6144 }
     }
   ]
 }
```

SUMMARY

In this module we looked at the Mesos architecture and components

- Masters, Agents, Frameworks
- Framework schedulers, executors
- Mesos Offer Cycle
- Containerizers
- Resources, Roles, Attributes, Quotas

QUESTIONS

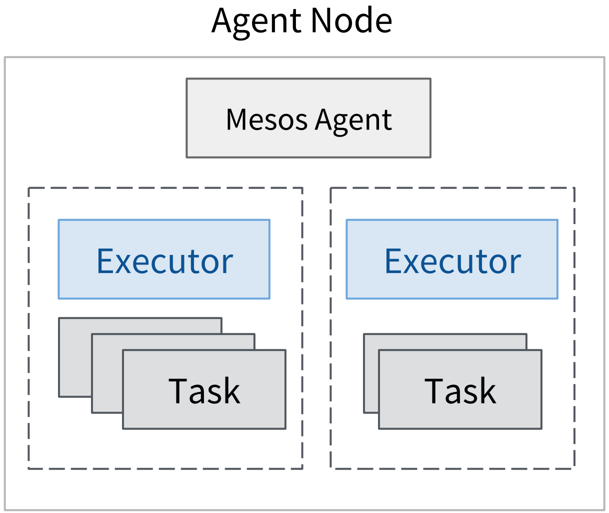
MESOSPHERE

BACKUP SLIDES

MESOS AGENT

- A process running on agent nodes that manages the executors, tasks, and resources of that node:

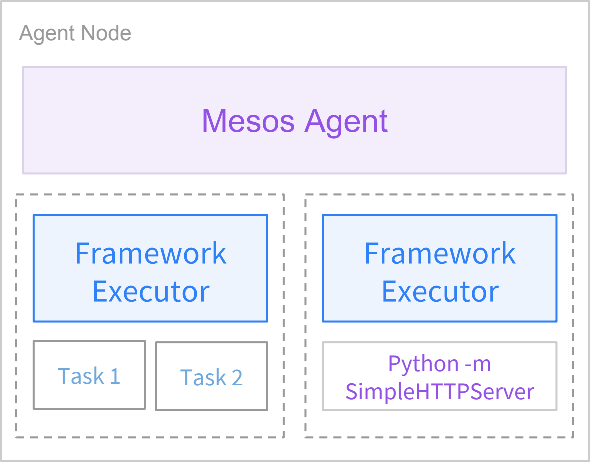
- Mesos agents have two primary functions:
  - Manage and offer some or all of the local resources on the Mesos agent node
  - Launch and manage the executors using containers to run a task
- In Mesos 1.0, slaves will be renamed to agents



FRAMEWORK EXECUTOR

Executors do work on behalf of the framework on the agent nodes.

- An executor runs within a container
- An executor can run multiple tasks



DOCKER CONTAINERIZER

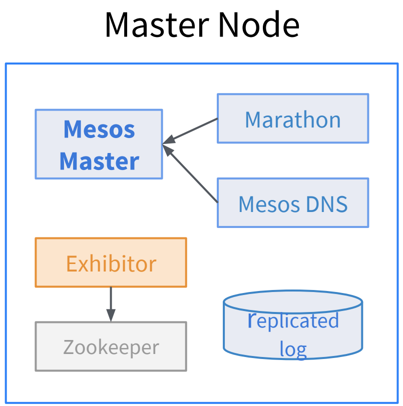
- ○ Launches all containers with the "mesos-" prefix plus the agent id (ie: mesos-slave1-abcdefghji)
- ○ Assumes all containers with the "mesos-" prefix are managed by the agent and that it is free to stop or kill the containers
- ○ When launching the Docker image as an executor, it skips launching a command executor and reaps on the Docker container executor PID

Notes:

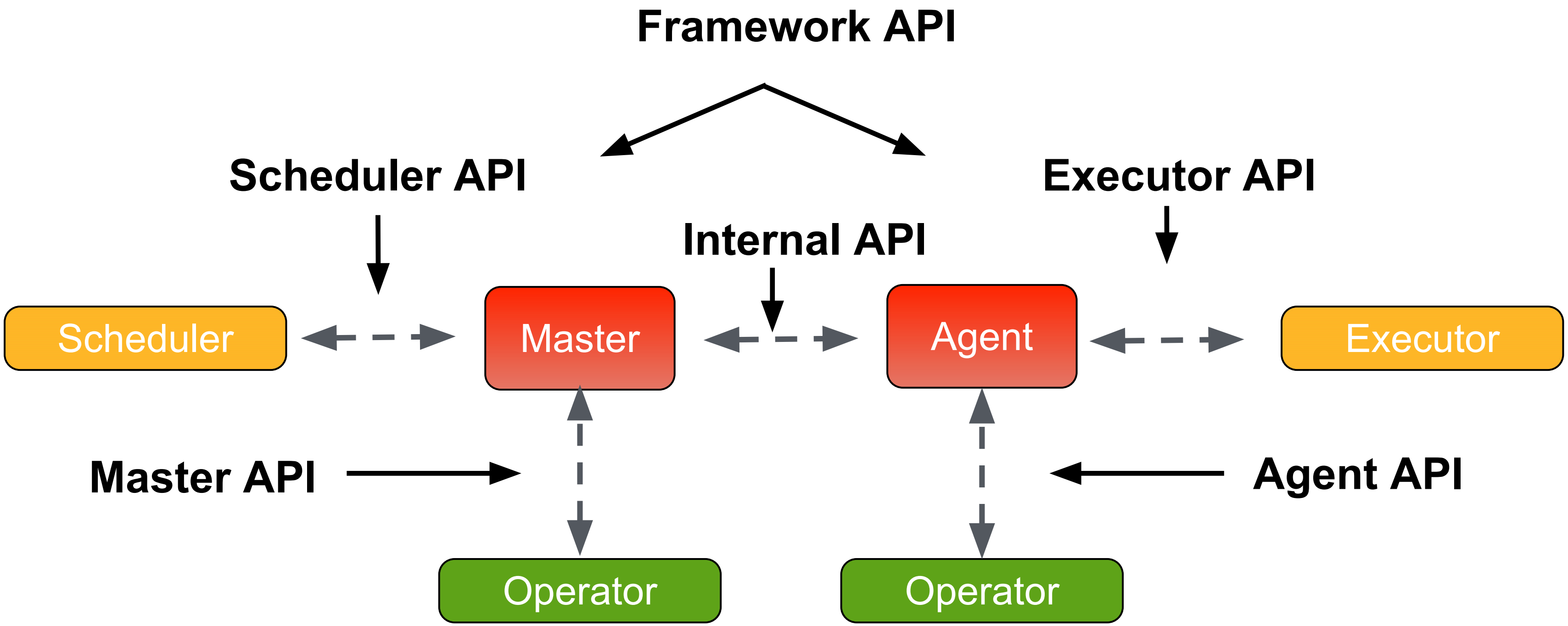
- ○ To easily support running a Docker image as an executor, Mesos defaults to using host networking
- ○ The containerizer also supports an optional forceful pull of the docker image. If force pull is disabled, the image will only be updated if it's not already available on the host

MESOS MASTER

- Coordinates cluster resource management and manages task lifecycle
- Collects resources reported by Mesos agents and makes resource offers to Mesos schedulers
- Hosts the Mesos user interface
- Uses Zookeeper to elect a leader and provide failover in the event of a failure.
    - Deploy Mesos masters in odd numbers (starting at three) to ensure quorum and avoid split brain



Mesos APIs



Framework API
Scheduler API
Executor API
Internal API
Scheduler
Master
Agent
Executor
Master API
Agent API
Operator
Operator

DC/OS APIs

- DC/OS
  - https://github.com/dcos/dcos-cli
  - https://dcos.io/docs/1.7/administration/id-and-access-mgt/auth-api/#passing-your-http-api-token-to-dc-os-endpoints
  - Roadmap
- Marathon
  - http://mesosphere.github.io/marathon/docs/rest-api.html
- Mesos
  - http://mesos.apache.org/documentation/latest/endpoints/

STORAGE OPTIONS

- Default Sandbox

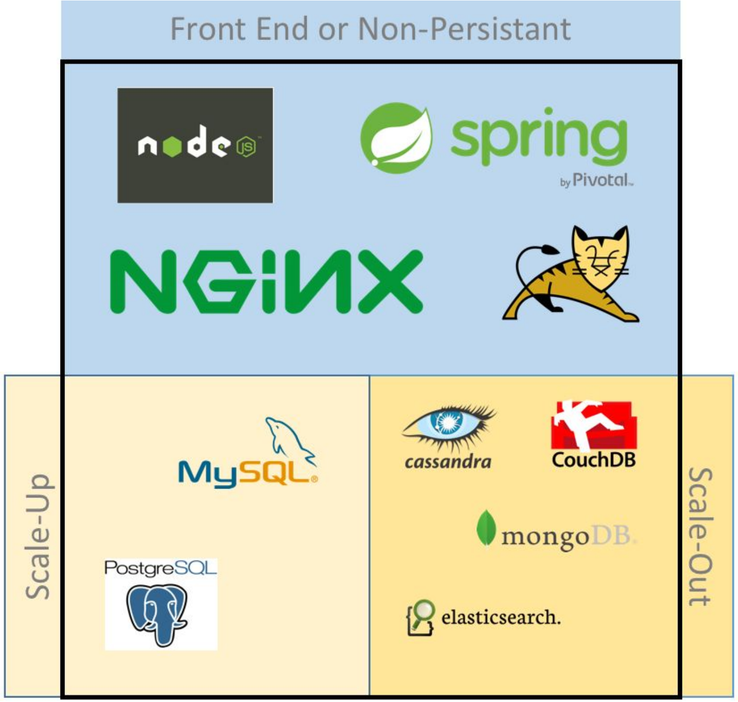
  ⇒ Simple to use, Task failures

- Persistent Volumes

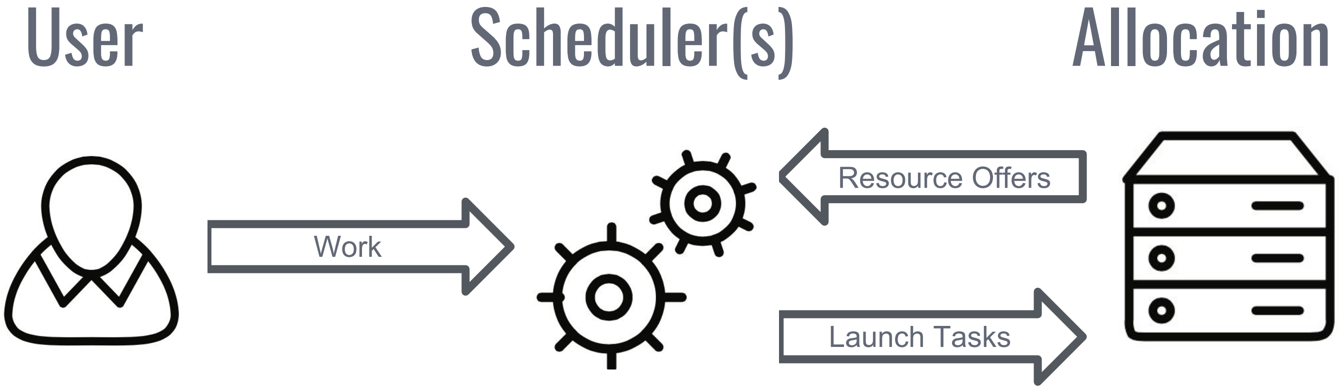
  ⇒ Task failures, (permanent) Node failures

- Distributed File System/External Storage

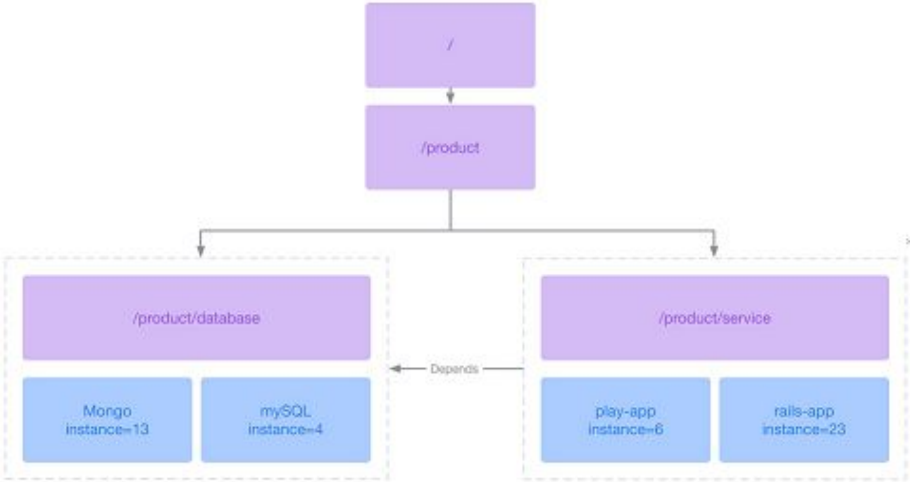
  ⇒ Node failures, non-local writes



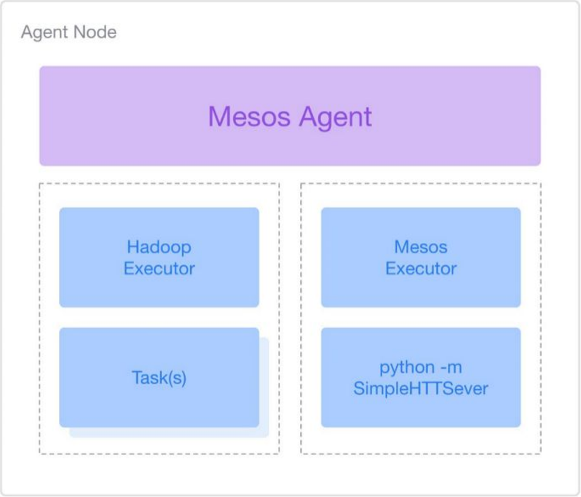
2-Level Scheduling

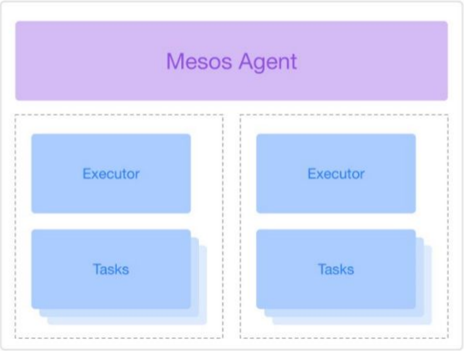


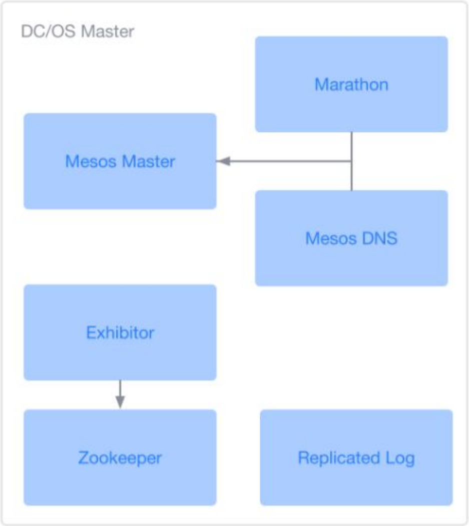
SCRATCH


/
/product
/product/database
/product/service
Depends
Mongo
instance=13
mySQL
instance=4
play-app
instance=6
rails-app
instance=23

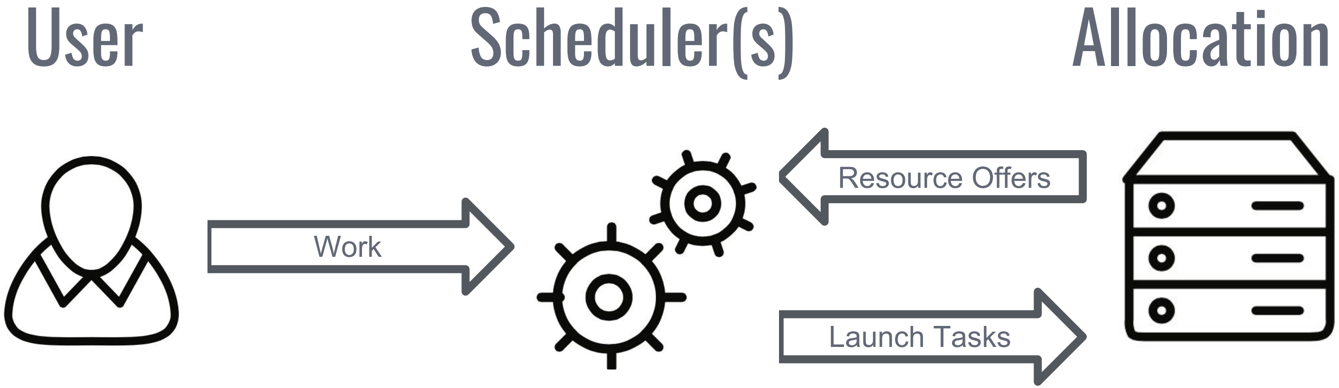
SCRATCH


Agent Node
Mesos Agent
Hadoop Executor
Task(s)
Mesos Executor
python -m SimpleHTTSever


Mesos Agent
Executor
Tasks
Executor
Tasks


DC/OS Master
Marathon
Mesos Master
Mesos DNS
Exhibitor
Zookeeper
Replicated Log

2-Level Scheduling



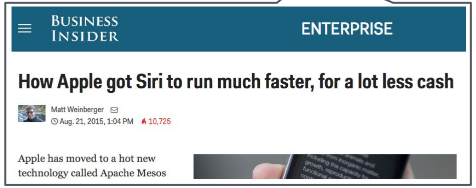
EVOLVED FROM EARLY INNOVATORS

Borg/Omega	Tupperware/Bistro	Apache Mesos
~2001	*~2007*	*2010+*
Proprietary	Proprietary	Open Source (Apache license)
Google	facebook.	airbnb, TWO SIGMA, PayPal, NETFLIX, yelp

CADE METZ BUSINESS 03.05.13 6:30 AM

WIRED

RETURN OF THE BORG: HOW TWITTER REBUILT GOOGLE'S SECRET WEAPON



BUSINESS INSIDER ENTERPRISE

How Apple got Siri to run much faster, for a lot less cash

Matt Weinberger

Aug. 21, 2015, 1:04 PM 10,725

Apple has moved to a hot new technology called Apache Mesos

- Built at UC Berkeley AMPLab by **Ben Hindman** (Mesosphere Co-founder)
- Built in collaboration with Google to overcome some Borg Challenges
- Production proven at scale +80K hosts @ Twitter