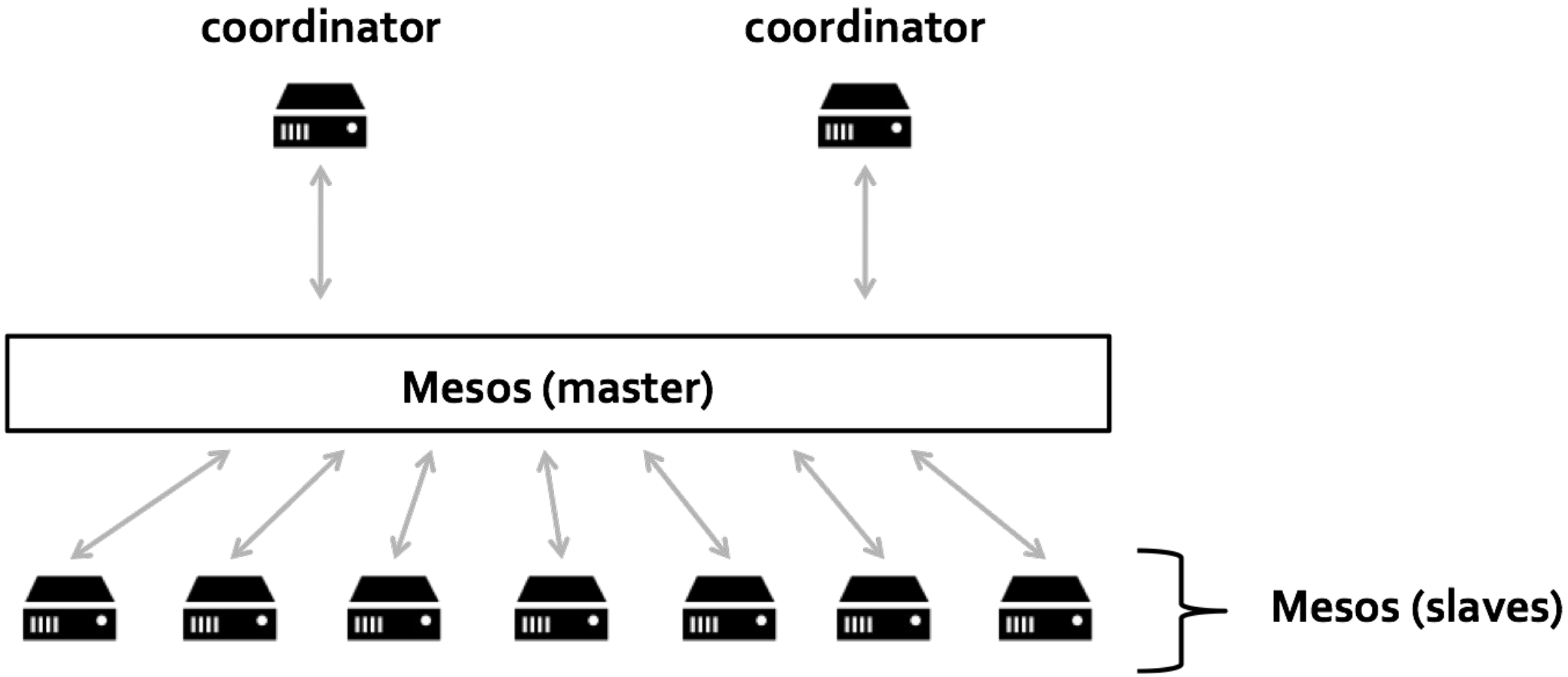
Sunil Shah

RUNNING & DEPLOYING SERVICES WITH MESOS

MESOSPHERE

Apache Mesos

LAYER OF ABSTRACTION



INTRODUCTION

Apache Mesos is a **cluster resource manager**.

It handles:

- **Aggregating resources** and **offering them to schedulers**
- **Launching tasks** (i.e. processes) on those resources
- **Communicating the state of those tasks** back to schedulers
- Tasks can be:
  - Long running services
  - Ephemeral / batch jobs

PRODUCTION MESOS USERS

MESOS: ORIGINS

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.

TECHNOLOGY

Mesos: A Platform for Fine-Grained Resource Sharing in the Data Center

Benjamin Hindman, Andy Konwinski, Matei Zaharia,
Ali Ghodsi, Anthony D. Joseph, Randy Katz, Scott Shenker, Ion Stoica
University of California, Berkeley

Sharing resources between batch processing frameworks

- Hadoop
- MPI
- Spark

VISION

The Datacenter Needs an Operating System

Matei Zaharia, Benjamin Hindman, Andy Konwinski, Ali Ghodsi,
Anthony D. Joseph, Randy Katz, Scott Shenker, Ion Stoica
University of California, Berkeley

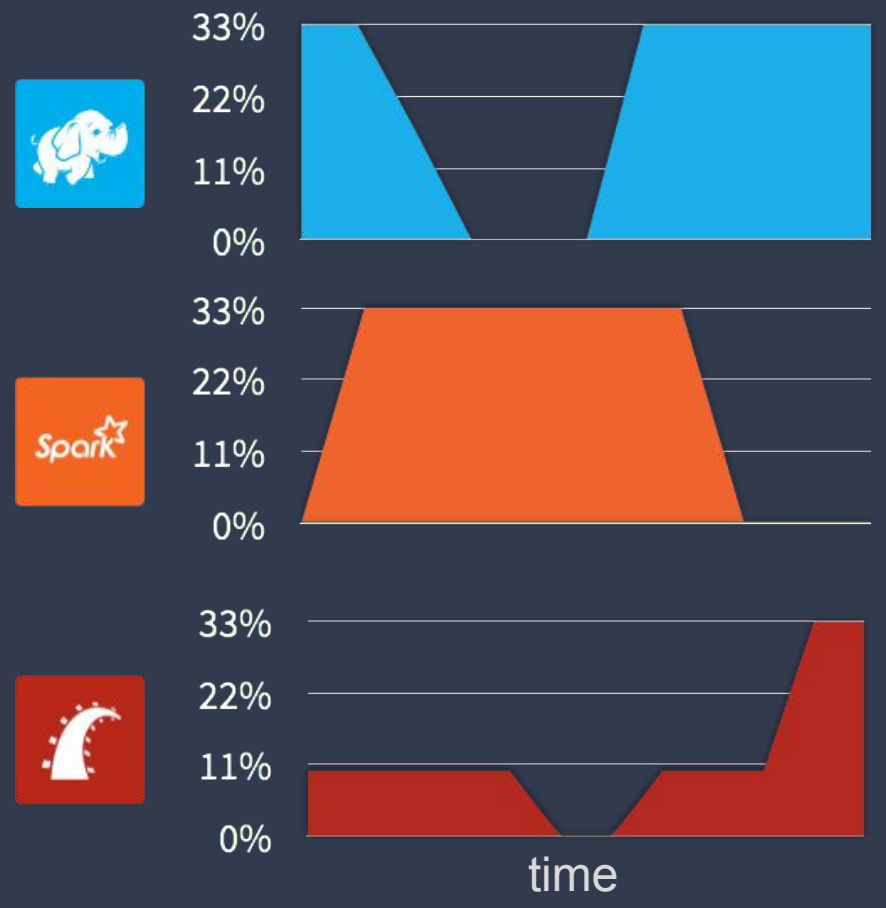
What does an operating system provide?

- Resource management
- Programming abstractions
- Security
- Monitoring, debugging, logging

KEEP IT STATIC

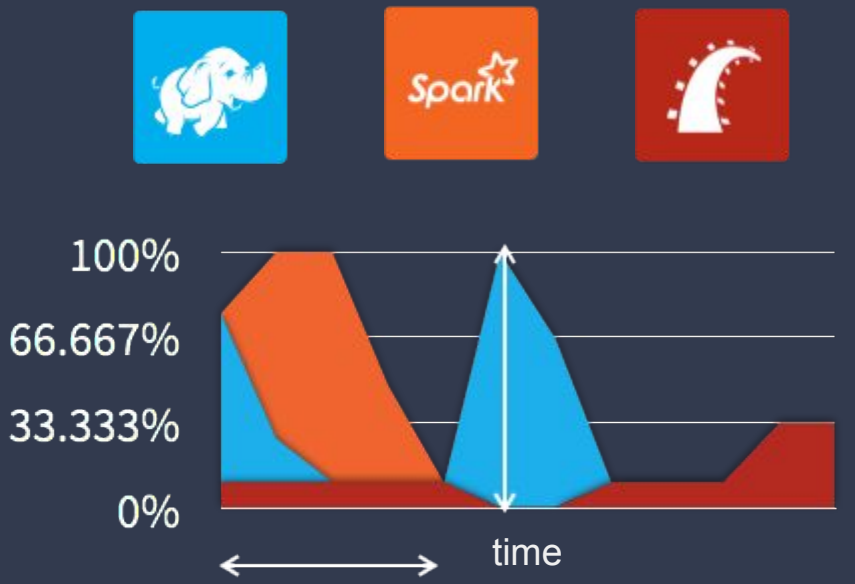
A naive approach to handling varied app requirements: **static partitioning**.

This can cope with heterogeneity, but is very expensive.



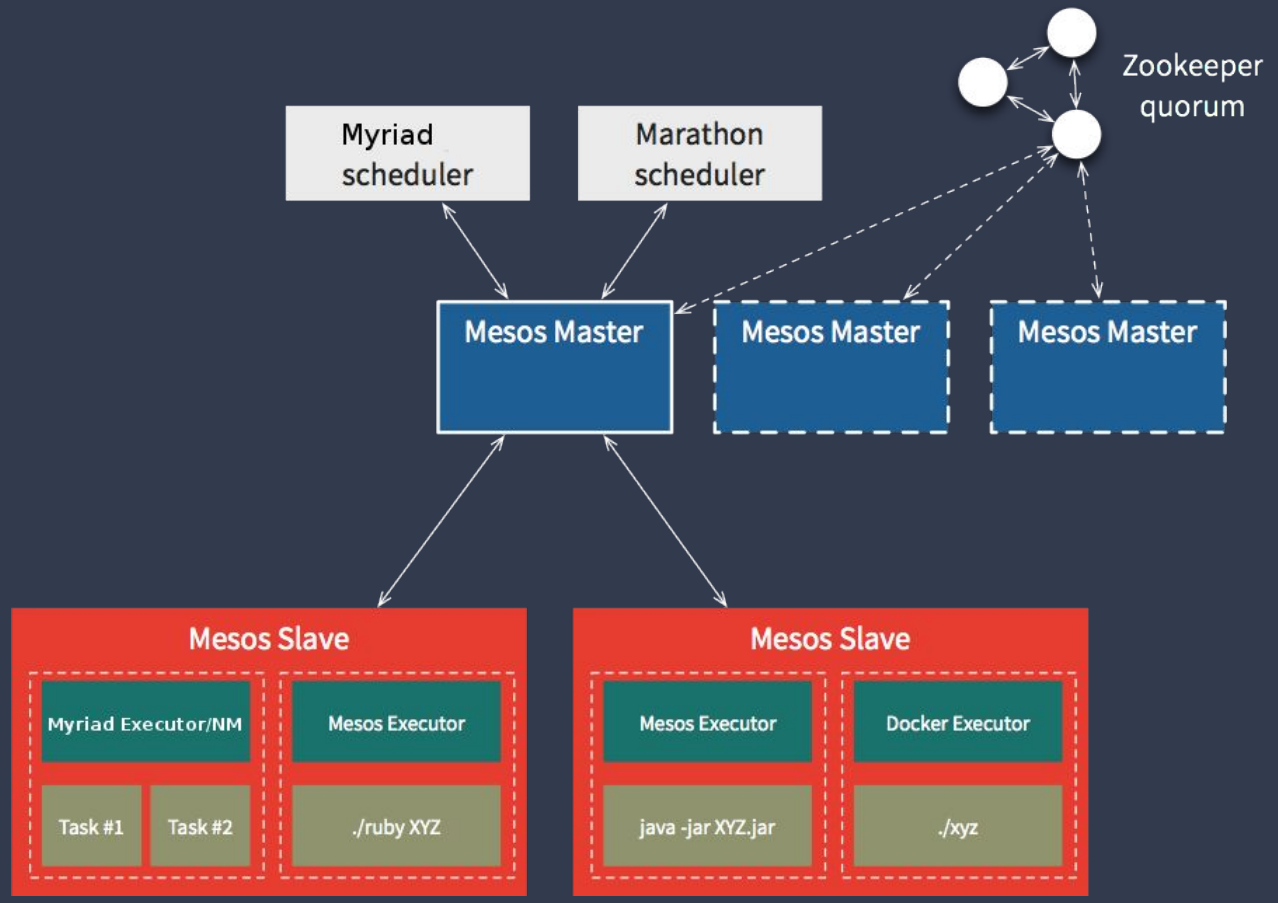
SHARED RESOURCES

Multiple frameworks can use the same cluster resources, with their share adjusting dynamically.



MESOS FUNDAMENTALS

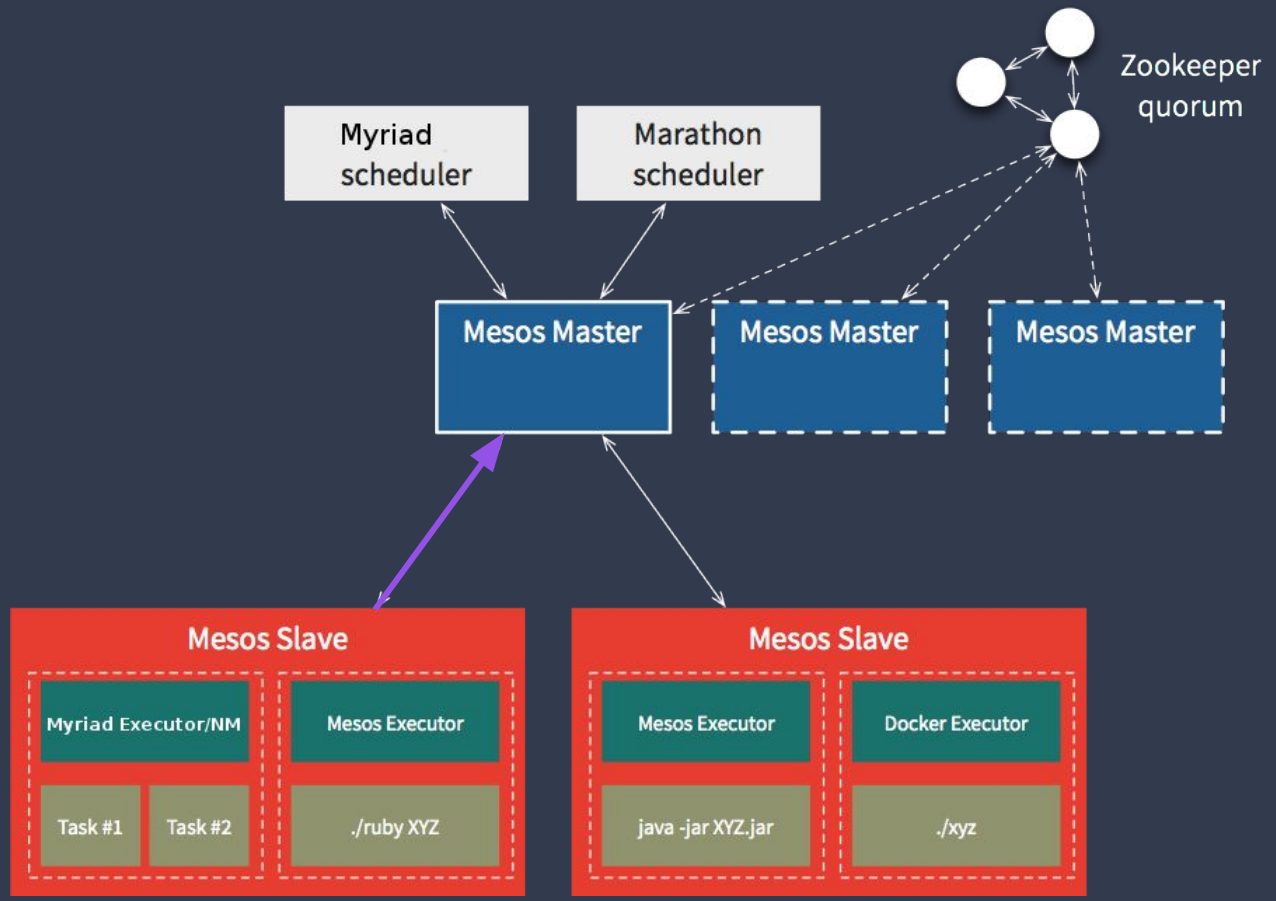
ARCHITECTURE



MESOS FUNDAMENTALS

ARCHITECTURE

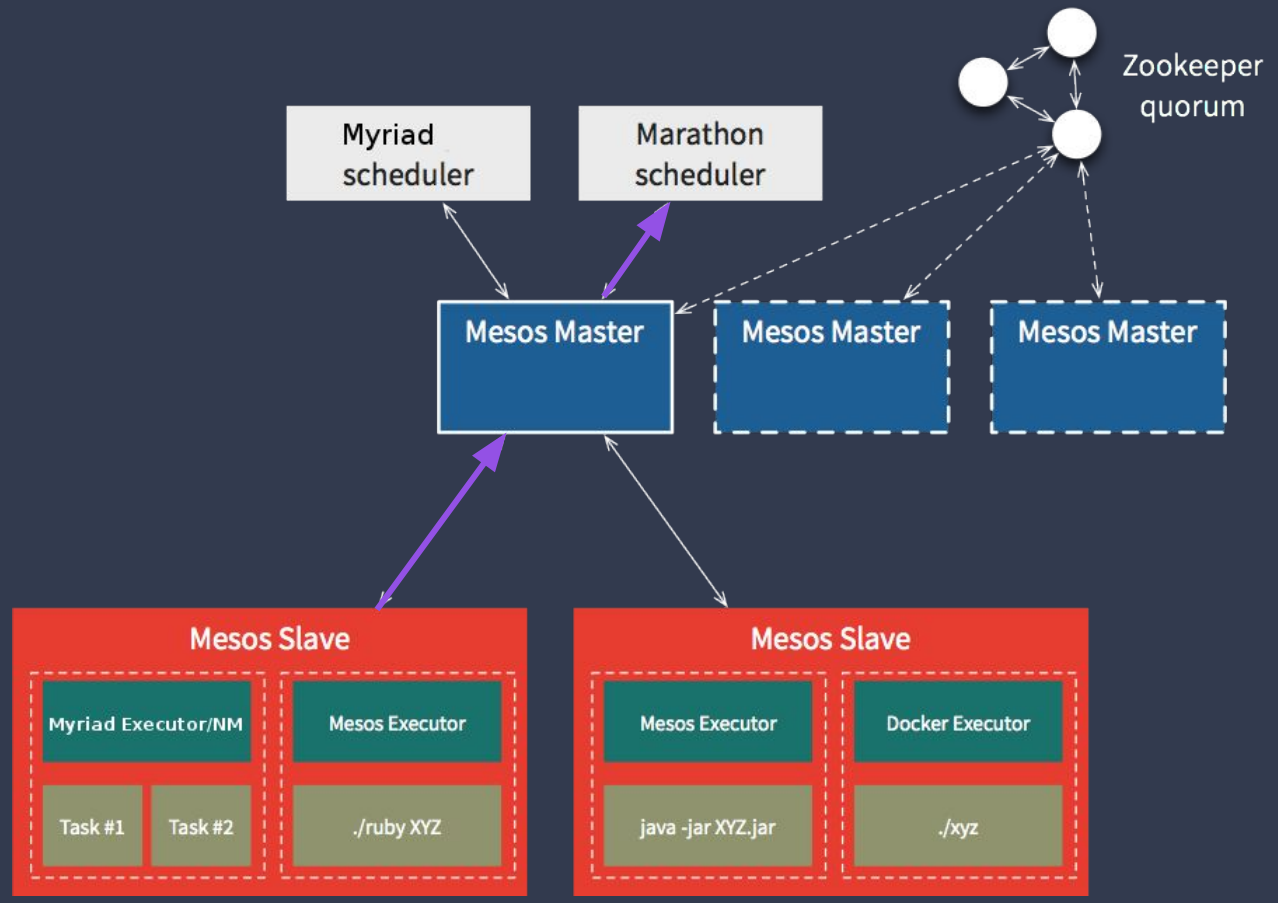
- Agents advertise resources to Master
- Master offers resources to Framework
- Framework rejects/uses resources
- Agents report task status to Master



MESOS FUNDAMENTALS

ARCHITECTURE

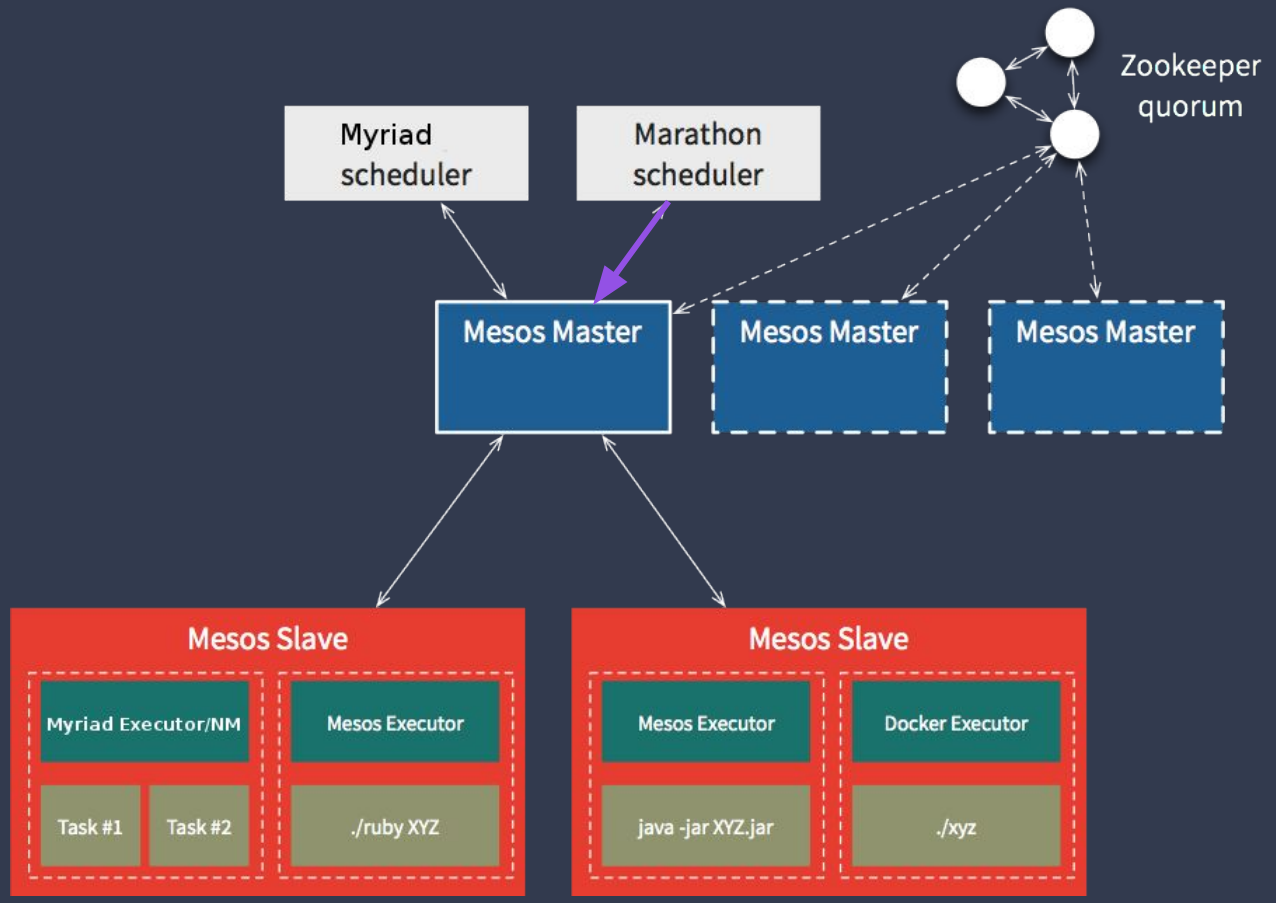
- Agents advertise resources to Master
- Master offers resources to Framework
- Framework rejects/uses resources
- Agents report task status to Master



MESOS FUNDAMENTALS

ARCHITECTURE

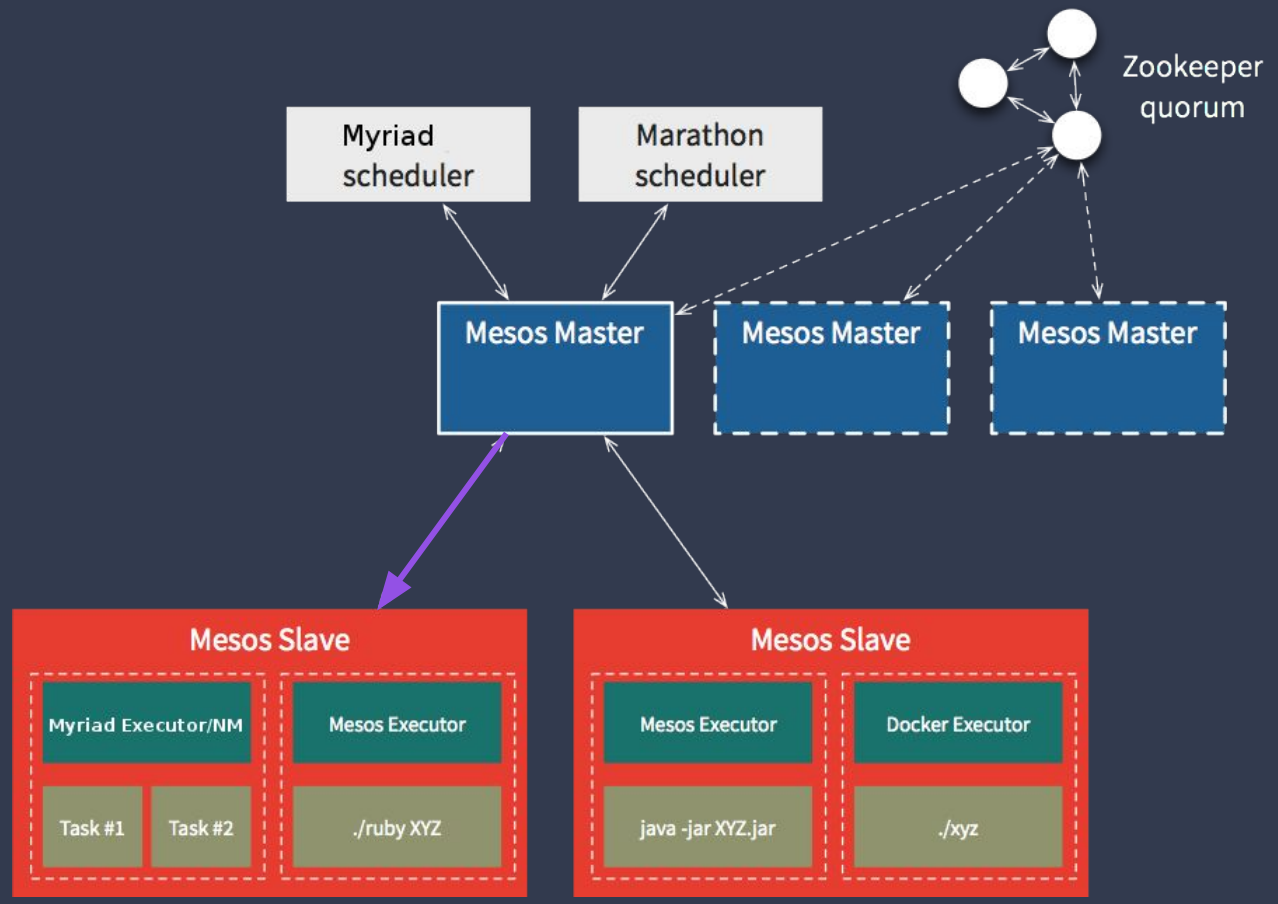
- Agents advertise resources to Master
- Master offers resources to Framework
- Framework rejects/uses resources
- Agents report task status to Master



MESOS FUNDAMENTALS

ARCHITECTURE

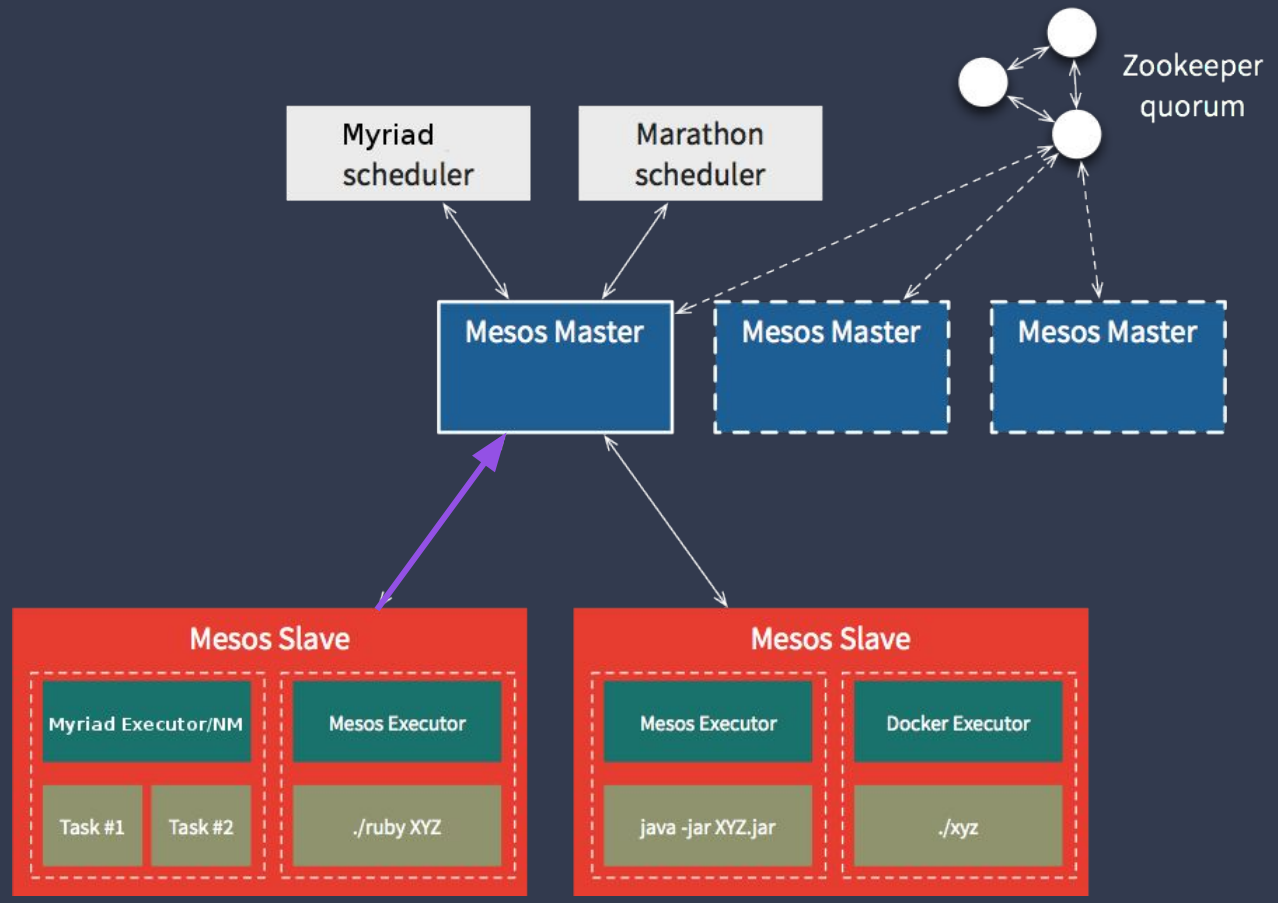
- Agents advertise resources to Master
- Master offers resources to Framework
- Framework rejects/uses resources
- Agents report task status to Master



MESOS FUNDAMENTALS

ARCHITECTURE

- Agents advertise resources to Master
- Master offers resources to Framework
- Framework rejects/uses resources
- Agents report task status to Master



TWITTER & 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.

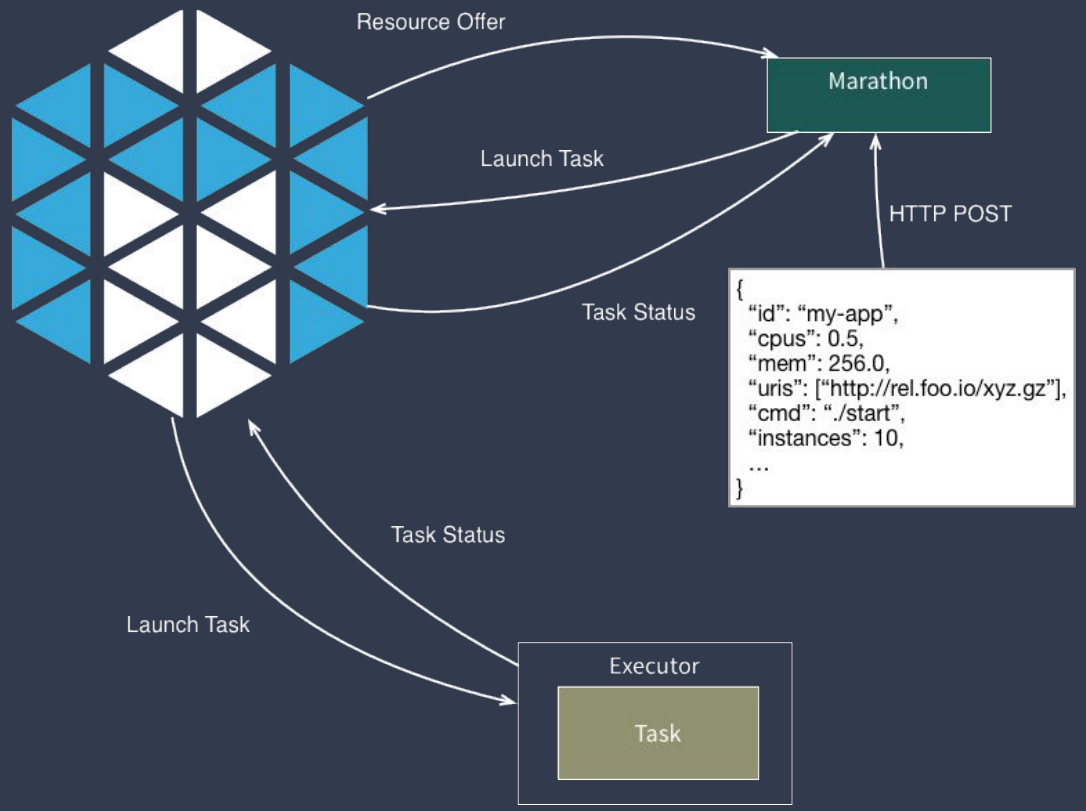
December 2010

APACHE INCUBATION

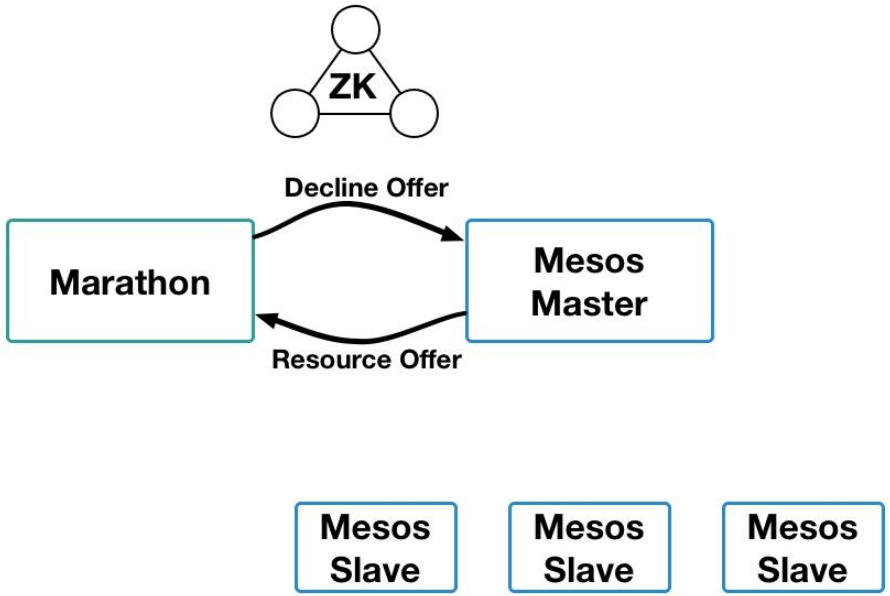
Mesos enters the Apache incubator.

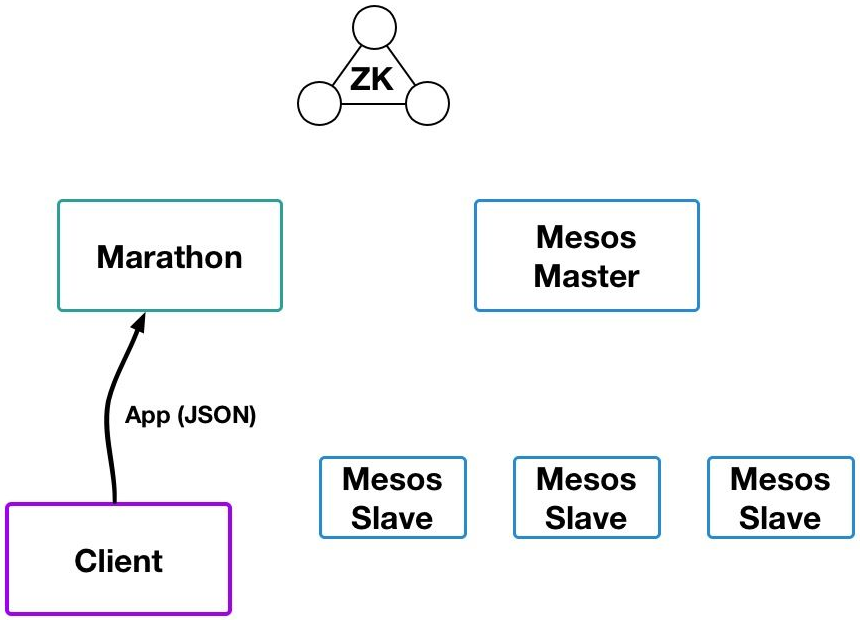
MESOS REALLY HELPS

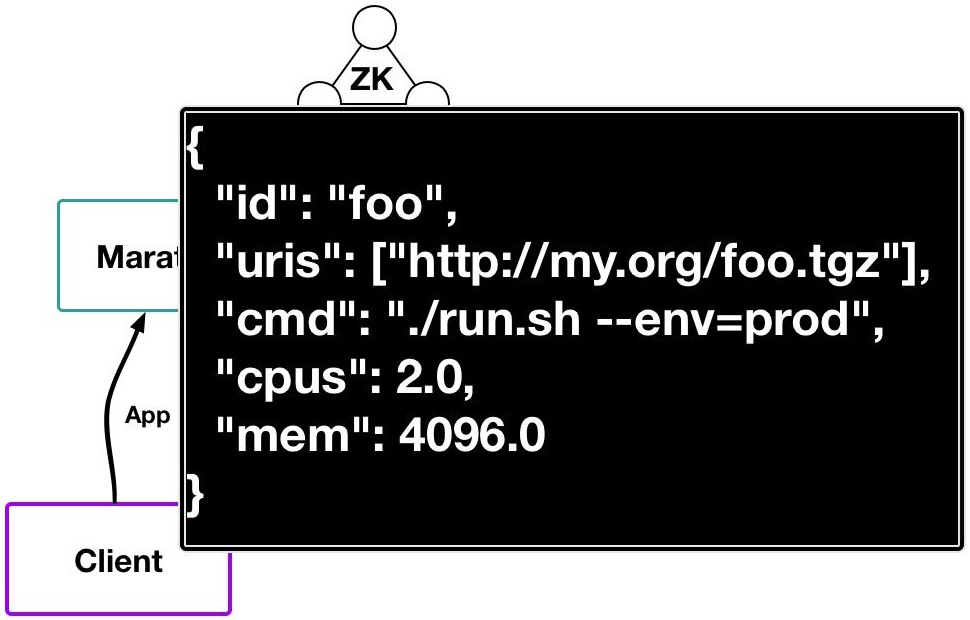
- Former Google engineers at Twitter thought Mesos could provide the same functionality as Borg.
- Mesos actually works pretty well for long running services.

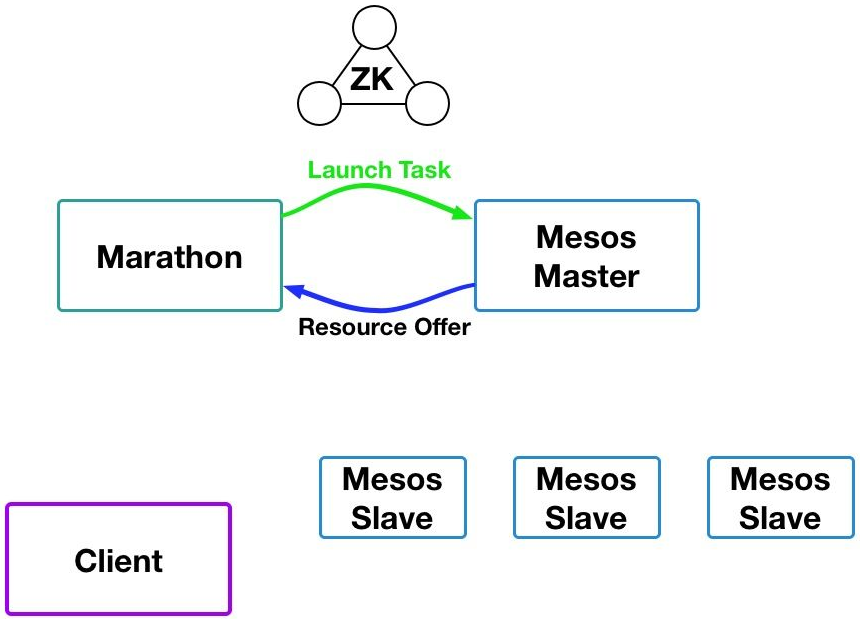


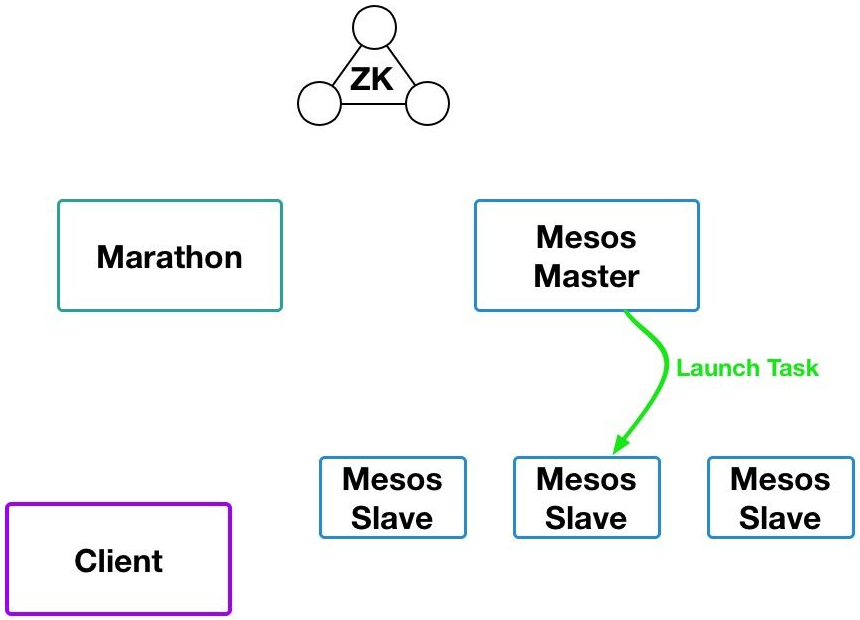
MESOS IN ACTION

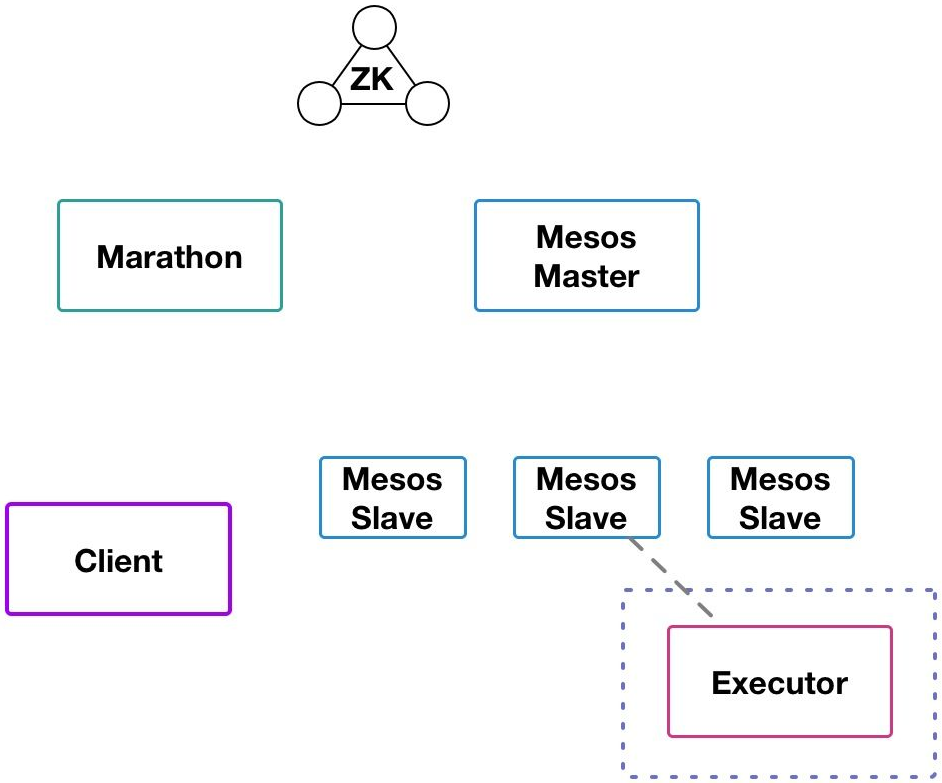


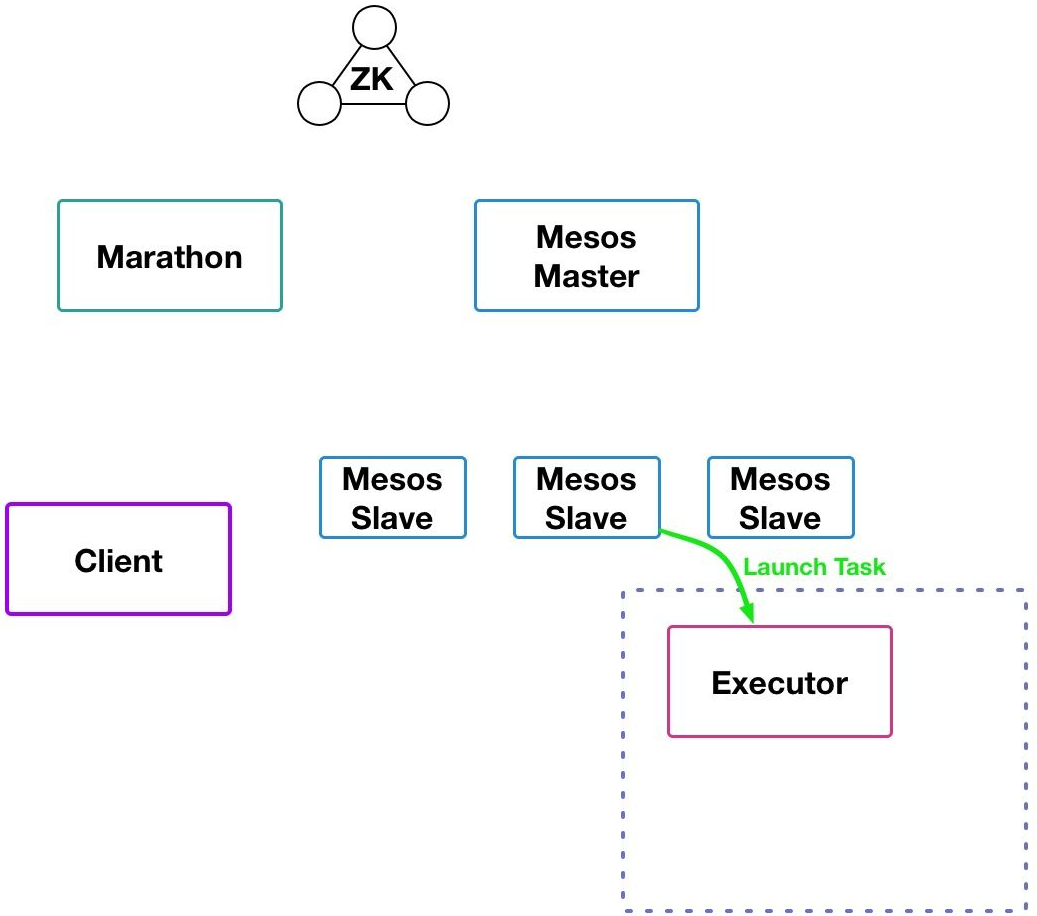


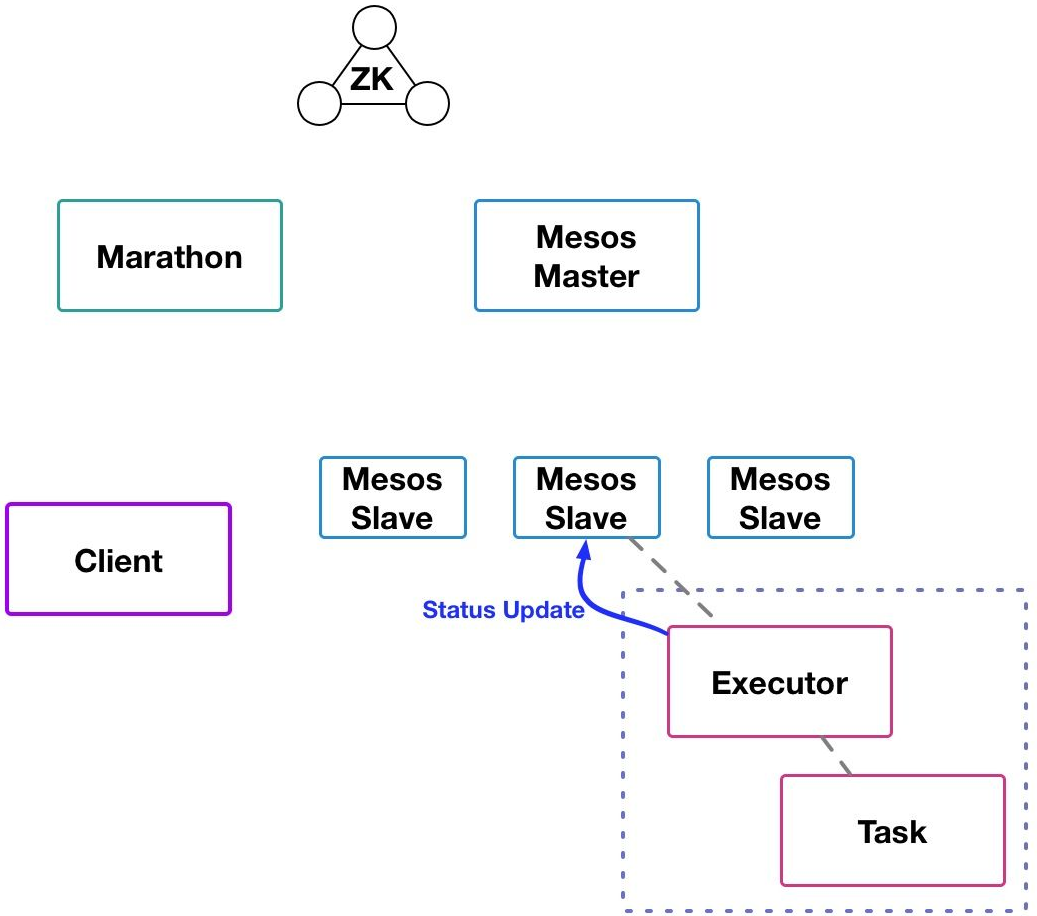


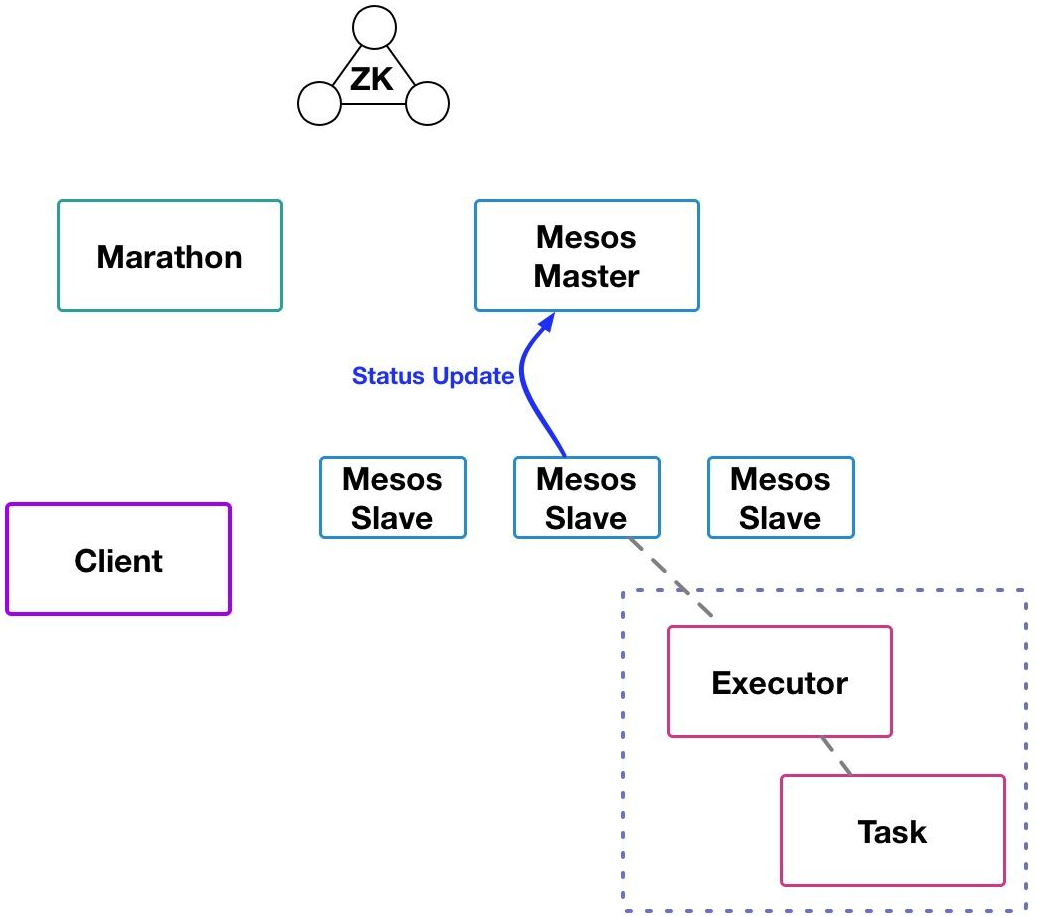


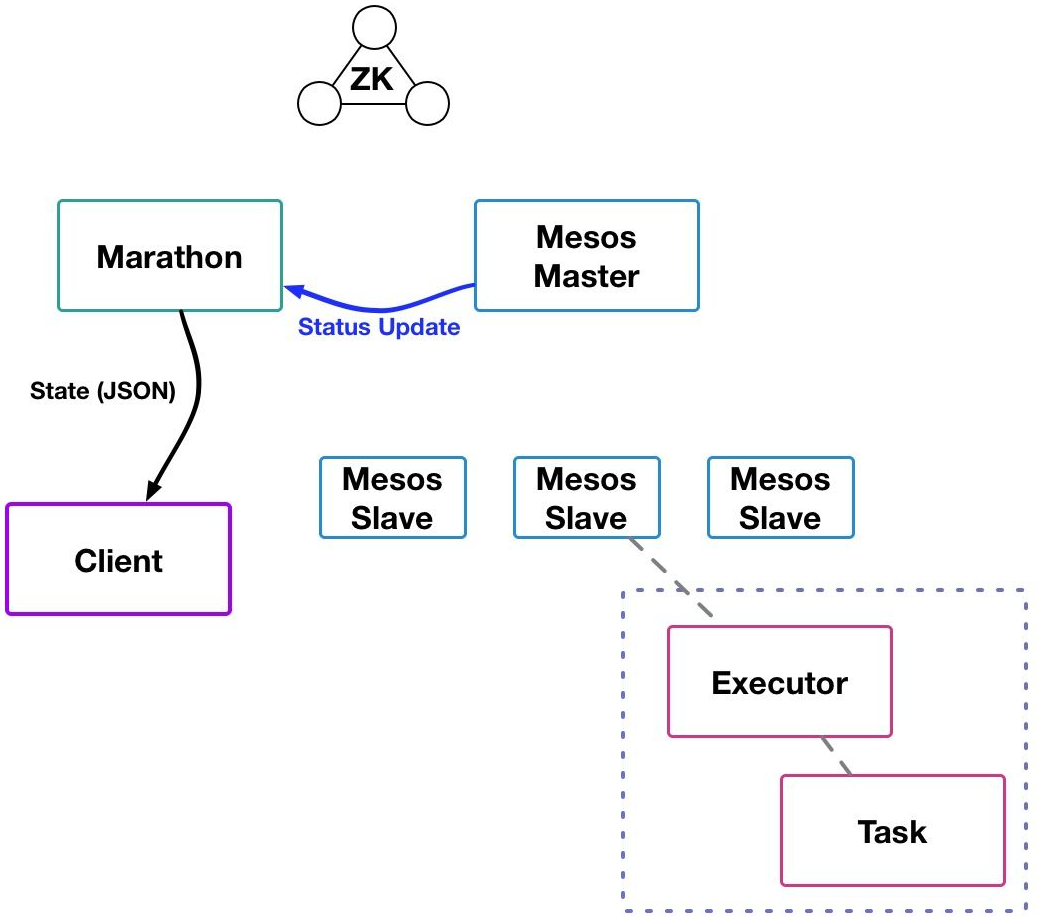


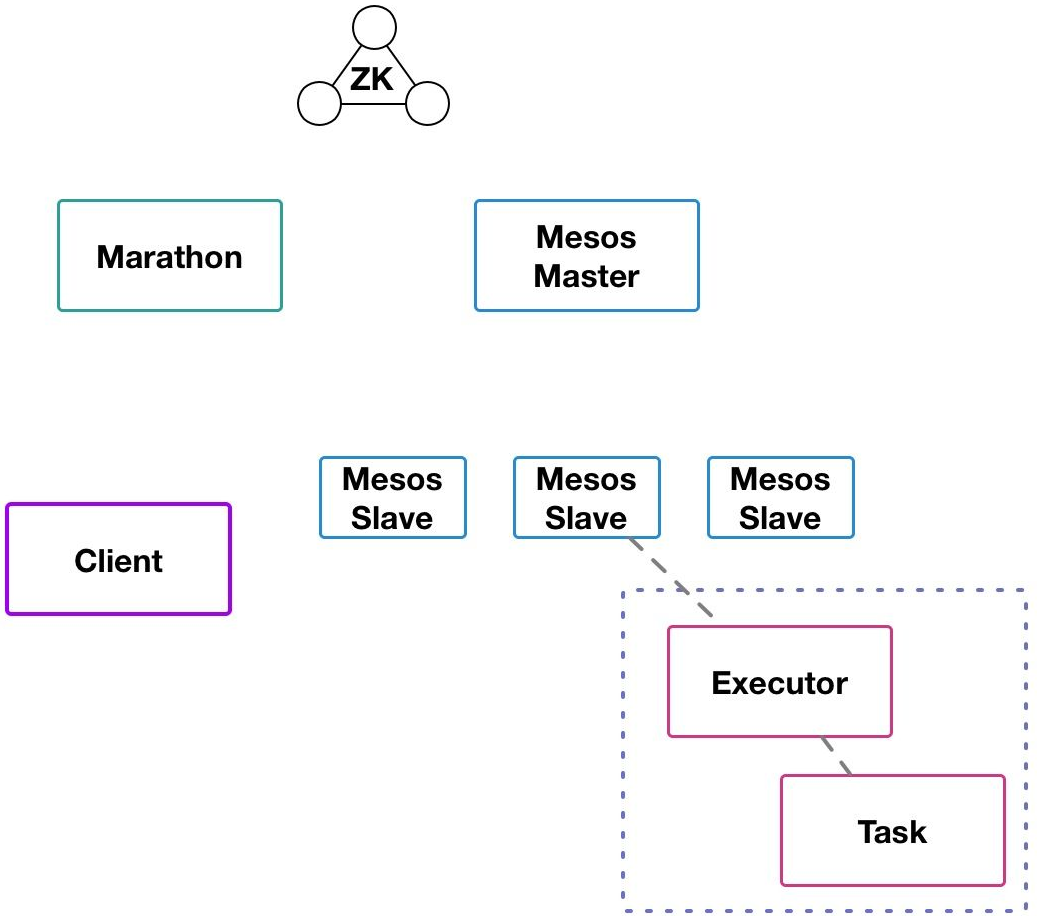






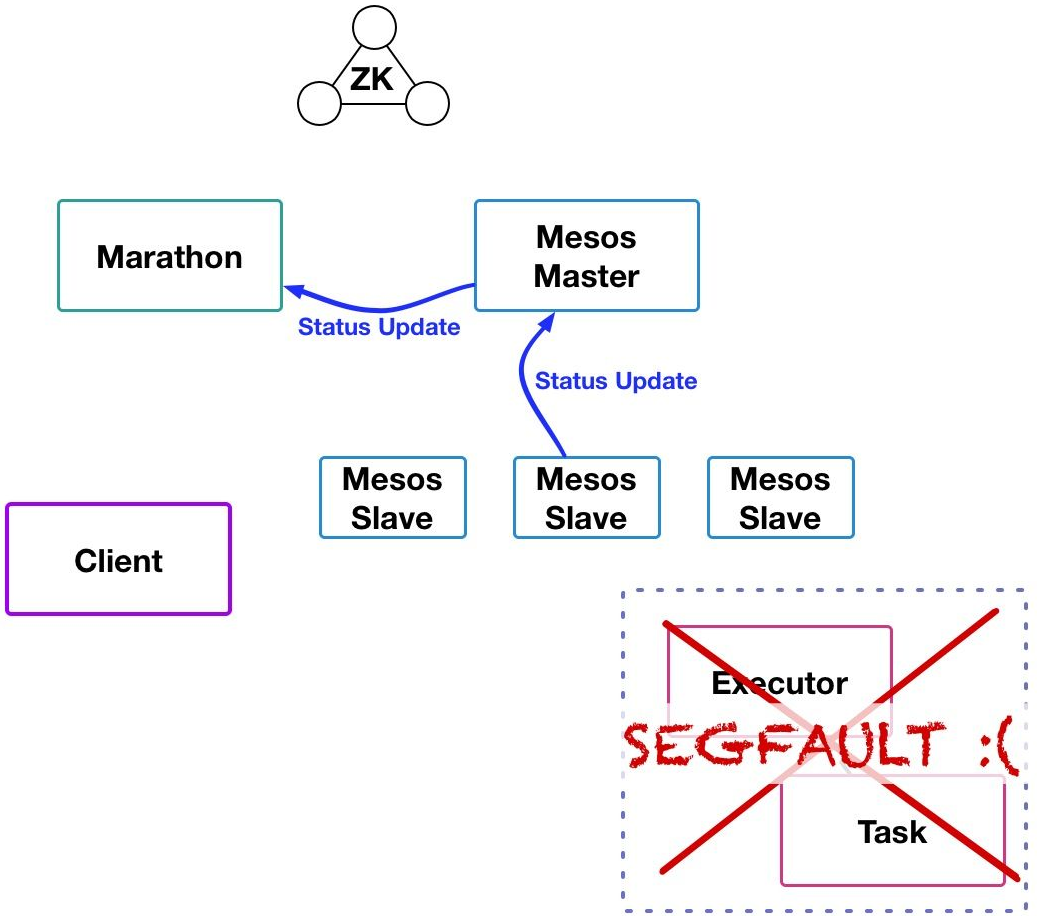


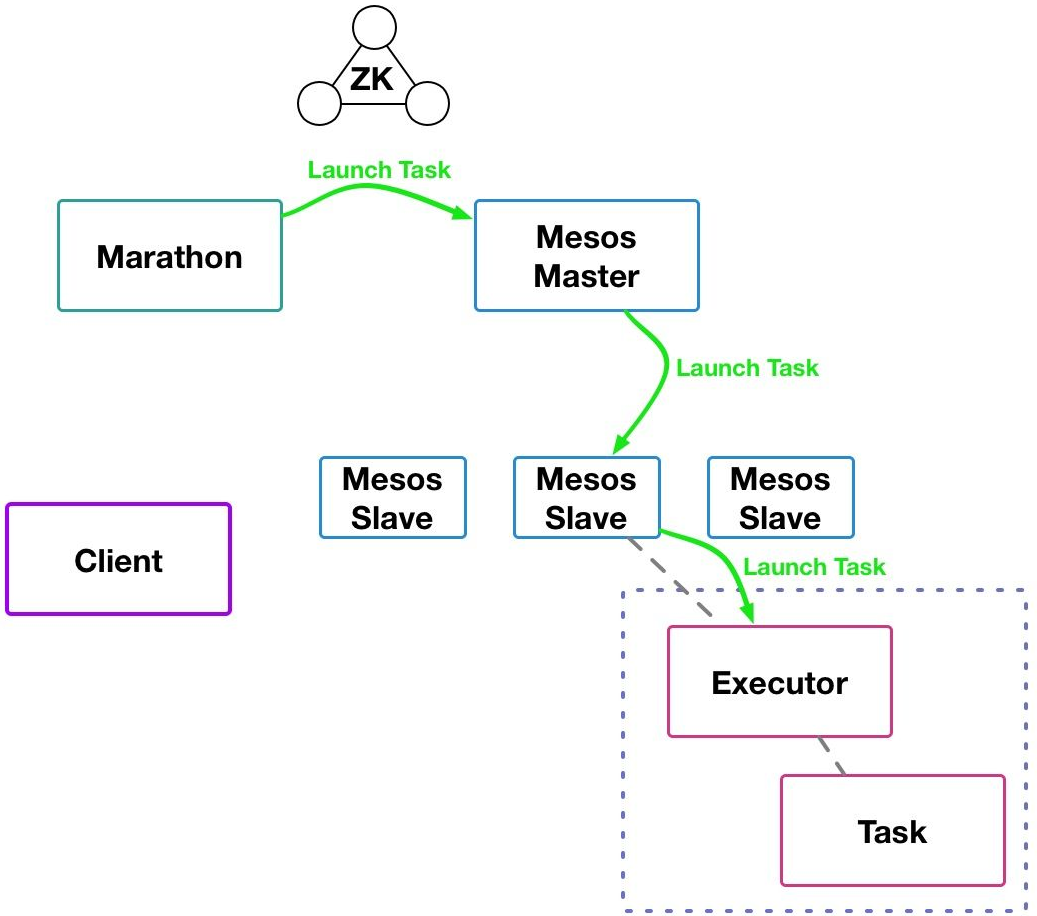


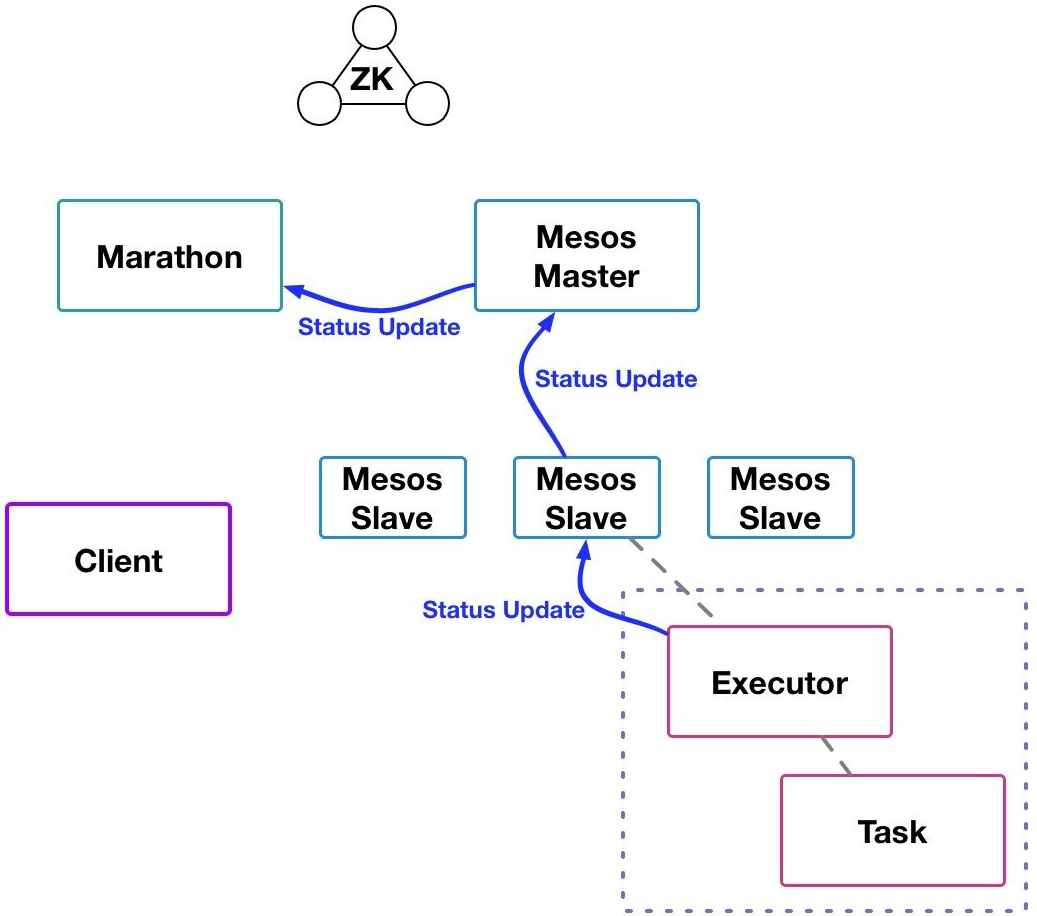


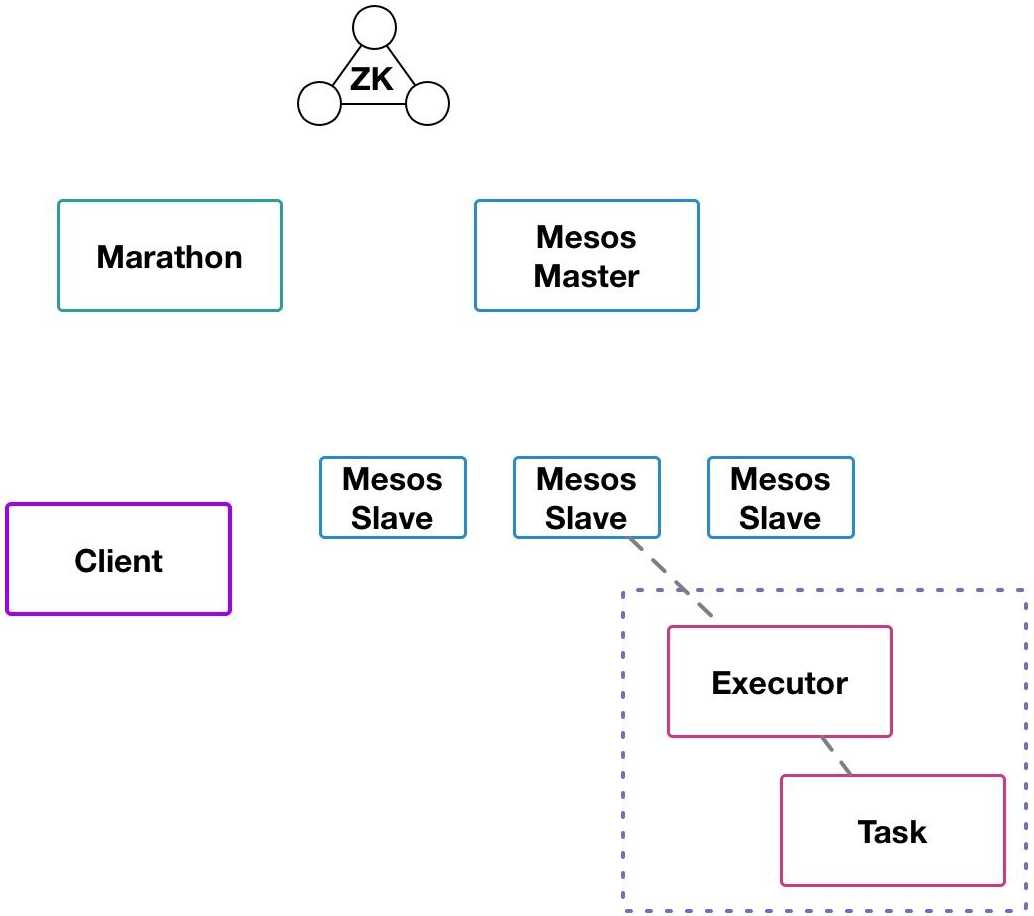
Mesos in Action

TASK FAILURE



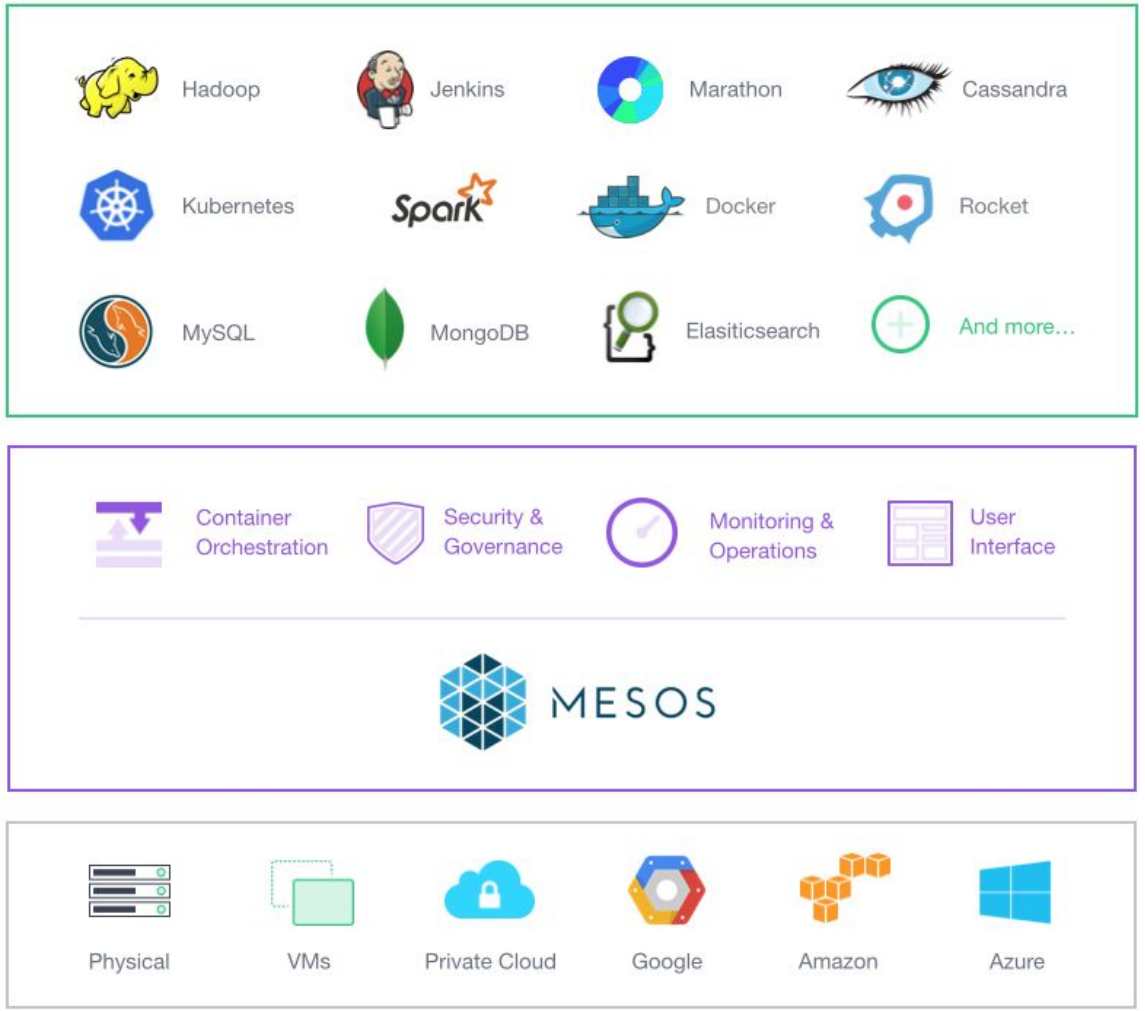






MESOS AS THE DATACENTER KERNEL

MESOS AS THE DATACENTER KERNEL



Services & Containers

Mesosphere DCOS

Existing Infrastructure

DOMINANT RESOURCE FAIRNESS

Algorithm 1 DRF pseudo-code

$R = \langle r_1, \cdots, r_m \rangle$ ▷ total resource capacities
$C = \langle c_1, \cdots, c_m \rangle$ ▷ consumed resources, initially 0
$s_i \ \ (i = 1..n)$ ▷ user i's dominant shares, initially 0
$U_i = \langle u_{i,1}, \cdots, u_{i,m} \rangle \ \ (i = 1..n)$ ▷ resources given to user i, initially 0

pick user i with lowest dominant share s_i
$D_i \leftarrow$ demand of user i's next task
if $C + D_i \leq R$ **then**
    $C = C + D_i$ ▷ update consumed vector
    $U_i = U_i + D_i$ ▷ update i's allocation vector
    $s_i = \max_{j=1}^{m} \{u_{i,j}/r_j\}$
else
    return ▷ the cluster is full
end if

CONTAINERS & CLUSTERS

CONTAINERS EVERYWHERE

Many Mesos tasks run in **containers**:

- Mesos containerizer
- Docker
- Universal containerizer (in progress)

CONTAINERS EVERYWHERE

Many Mesos tasks run in **containers**:

- Mesos containerizer
- Docker
- Universal containerizer (in progress)

Containers use standard linux features to create an isolated execution environment:

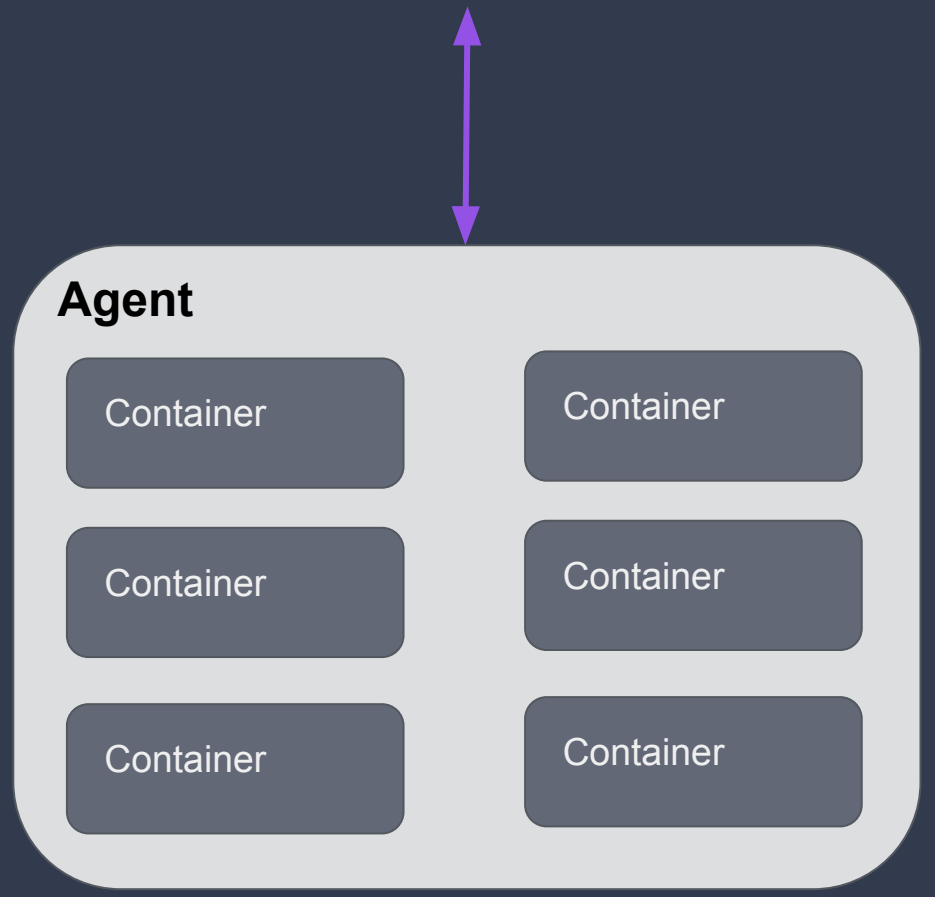
- kernel namespaces
  - process isolation
- control groups (cgroups)
  - resource isolation
- chroot
  - filesystem isolation
- seccomp
  - restricted kernel access

CONTAINER NETWORKING

Containers isolate tasks on the agent, but what about their communication?

The status quo in a Mesos cluster: one IP per agent.

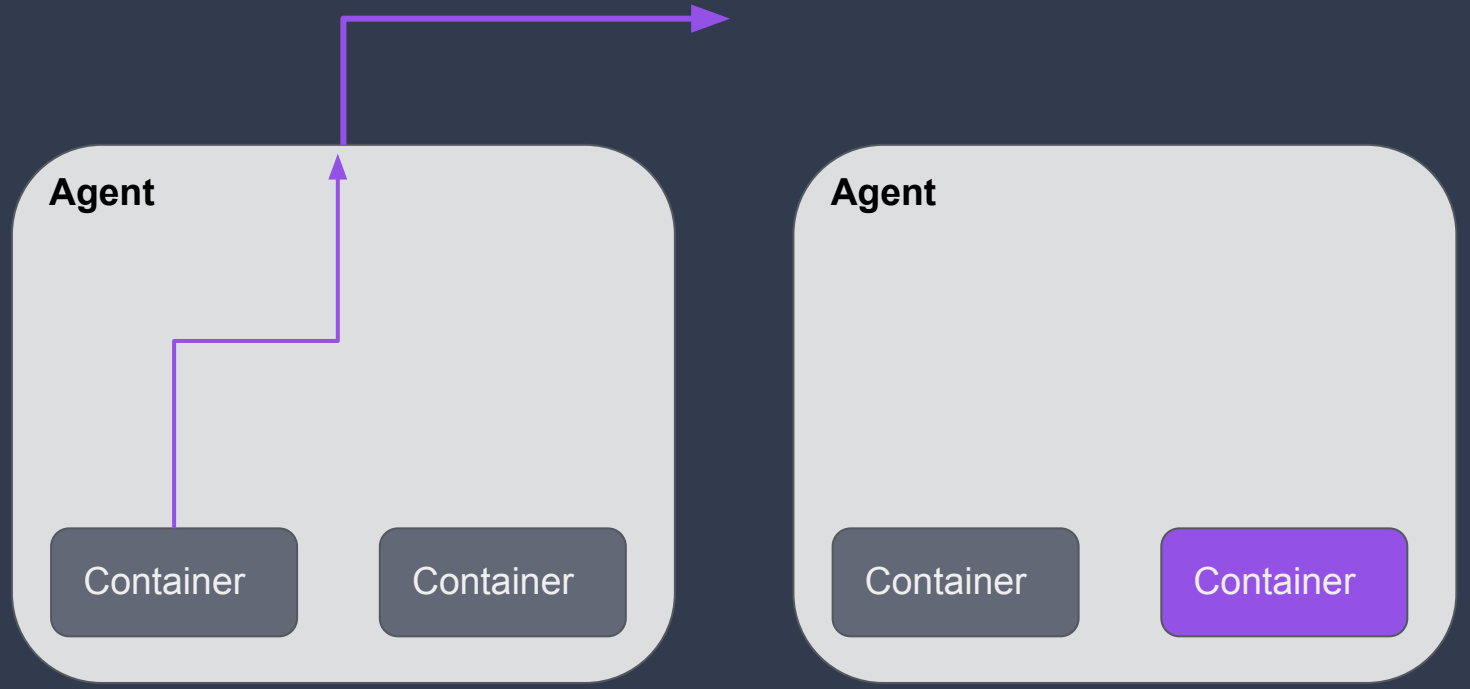
Many containers per agent: they must share a single IP.



CONTAINER NETWORKING

This causes headaches:

- Port conflicts
- Security compromises
- Performance
- Service discovery



Service Discovery

WHERE ARE MY SERVICES?

Service Discovery

TWO APPROACHES

1) Static ports
2) Dynamic ports

Service Discovery

STATIC PORTS

- Each instance service is given a unique hostname and runs on the same well known, port
- In order to co-locate multiple instances of service on same physical host, each container must have its own IP
- Typically use in conjunction with DNS A records

Service Discovery

STATIC PORTS

This approach is necessary for legacy applications but limits you to one instance per machine.

Service Discovery

DYNAMIC PORTS

Routing to services on unique ports requires running a secondary process:

- Using DNS server with SRV records (which resolve both IP address and port)
- Use proxy/iptables to remap well known ports to dynamically allocated ports

Or using a directory service where services register themselves (e.g. ZooKeeper)

Service Discovery

DYNAMIC PORTS

This requires your applications to be able to run on any port! In practice, not easy.

NETWORK ISOLATION

Segregating containers' network traffic can solve these problems in an elegant, maintainable way.

Implemented as Mesos modules:

- Project Calico
- Port-mapping isolation
- …

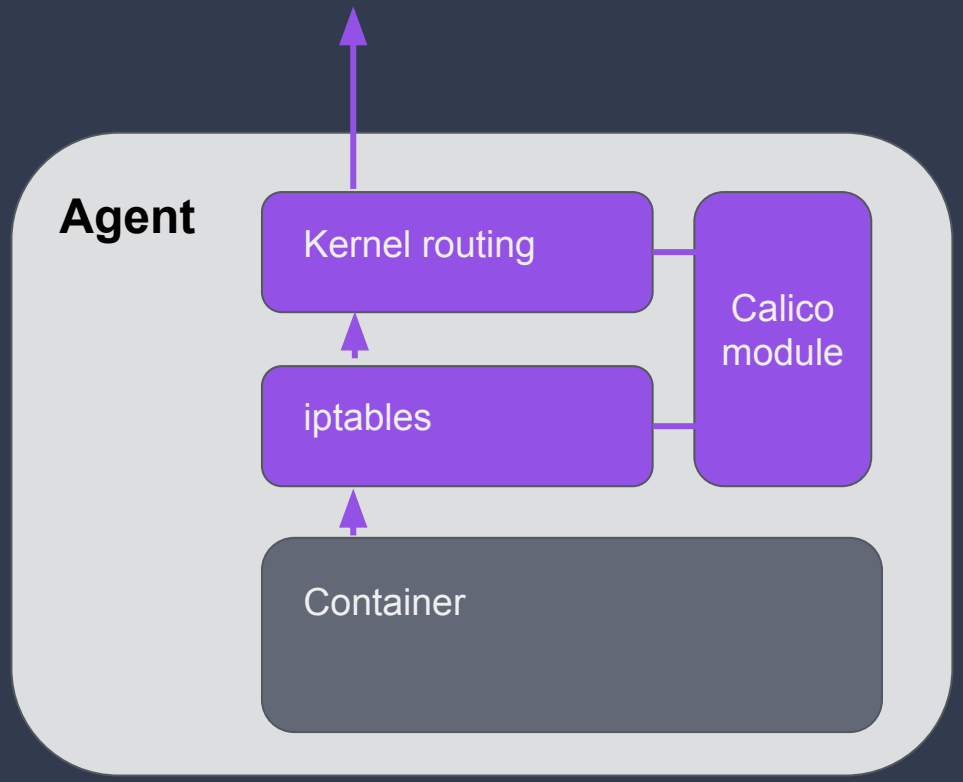
CALICO NETWORK ISOLATION

Calico Network Virtualizer & IP Address Manager:

- Pure Layer-3 solution
- Uses linux features to route container traffic
- Provides security policies
- Advertises routes to local containers via BGP
- Can assign IP-per-container

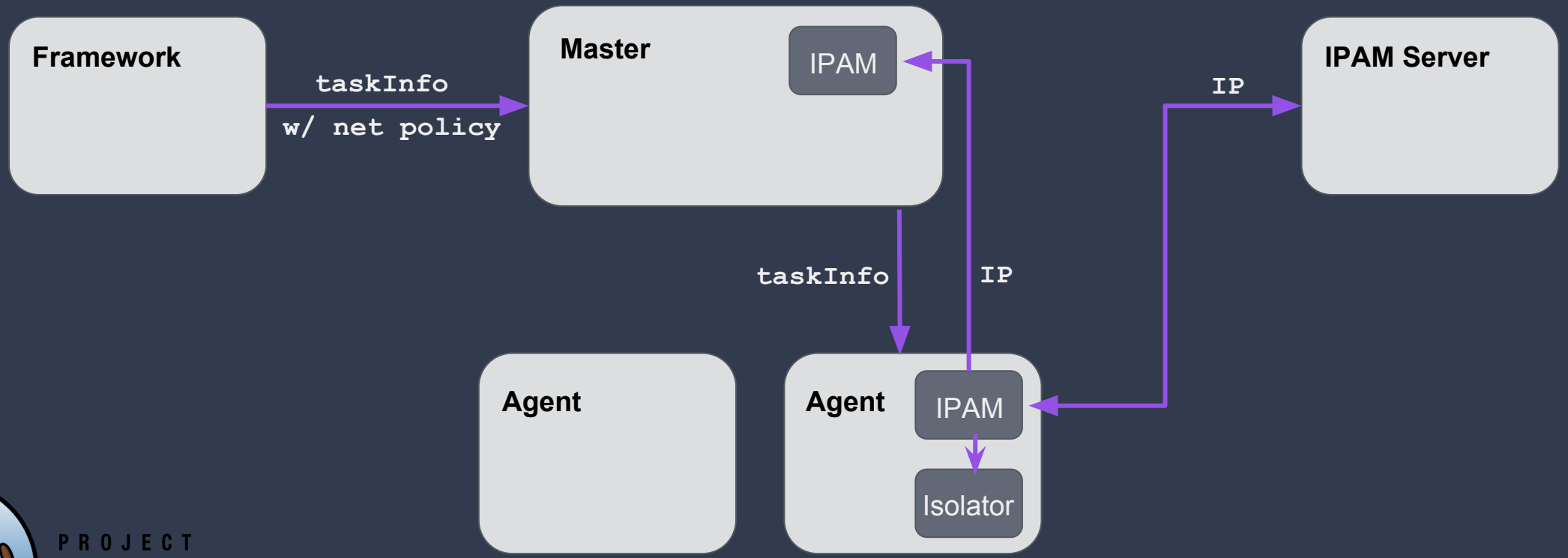
PROJECT CALICO

CALICO NETWORK ISOLATION



PROJECT CALICO

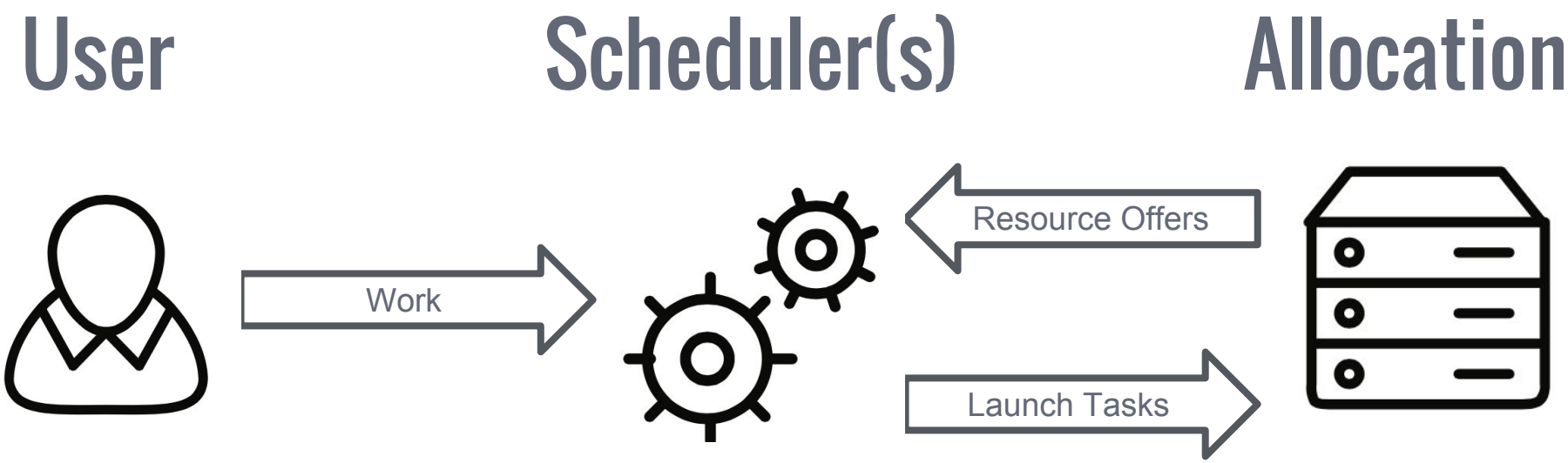
CALICO NETWORK ISOLATION



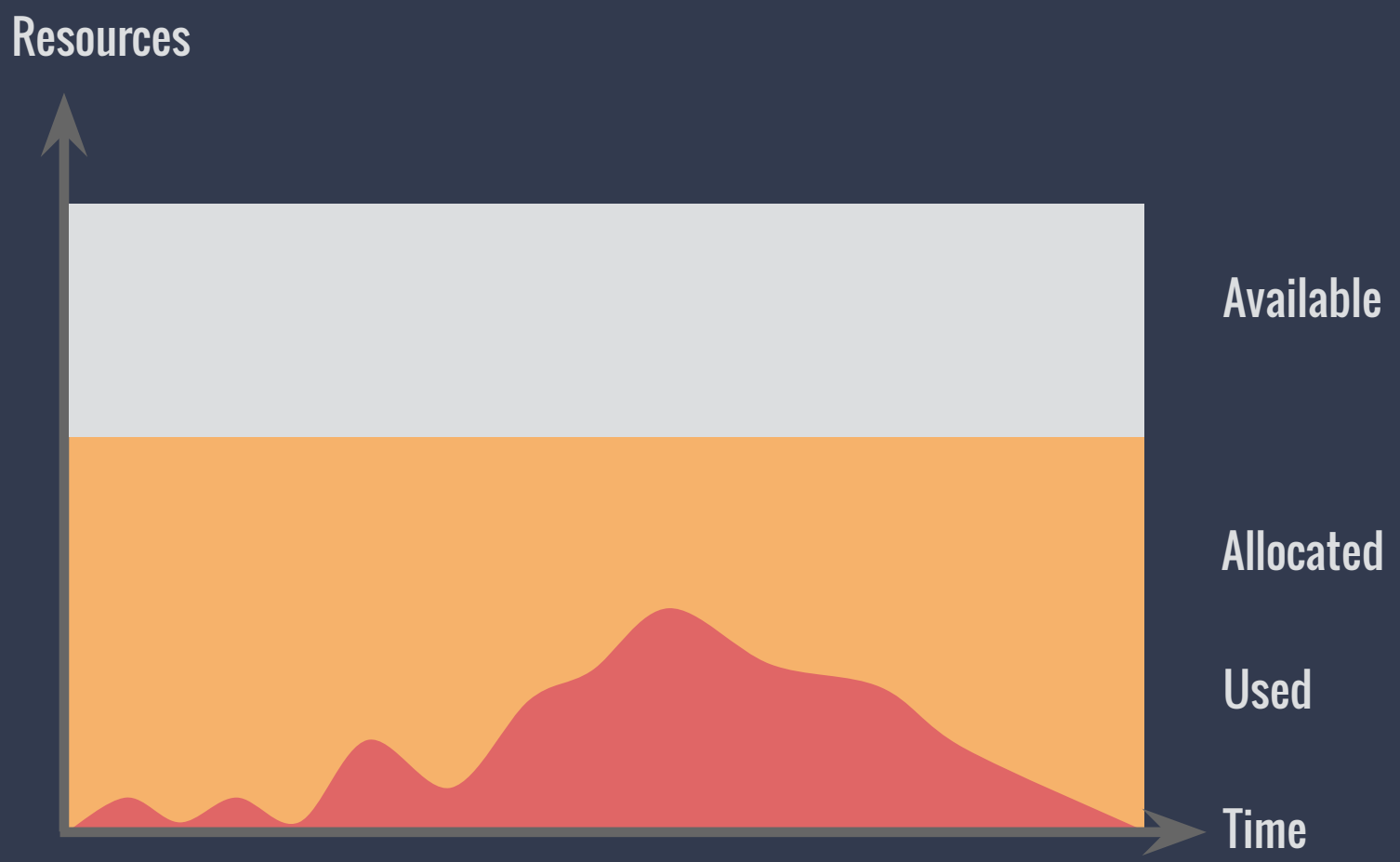
PROJECT CALICO

ESTIMATING RESOURCES IS HARD

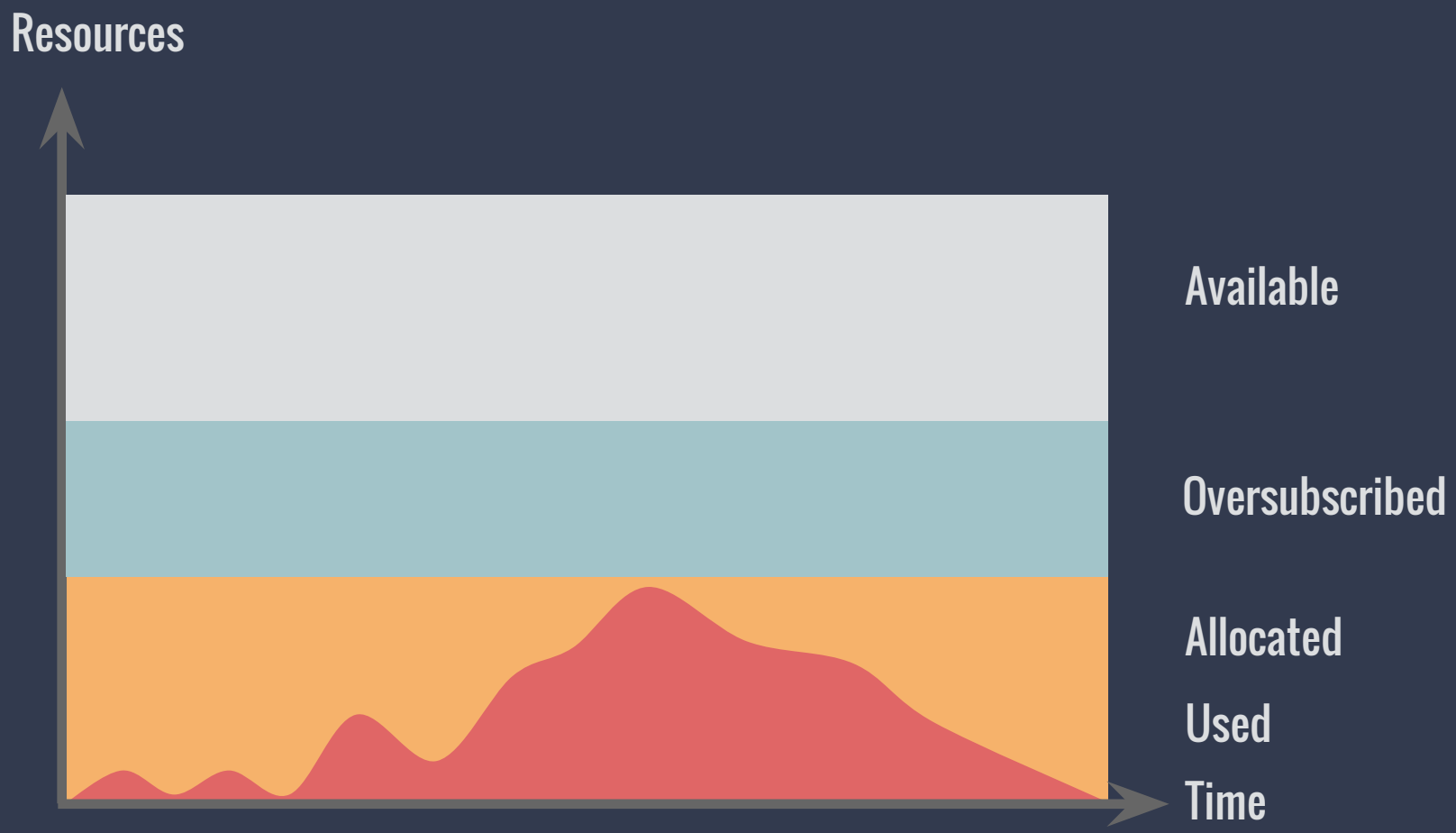
MESOS ENABLES MULTIPLE SCHEDULER ALGORITHMS



USAGE SLACK HURTS UTILISATION

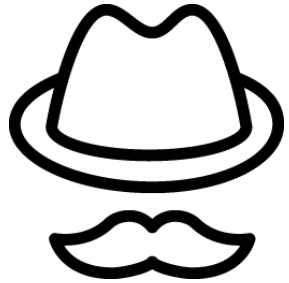


OVERSUBSCRIPTION ENABLES TASKS TO RUN ON SLACK



TWO COMPONENTS ENABLE OVERSUBSCRIPTION

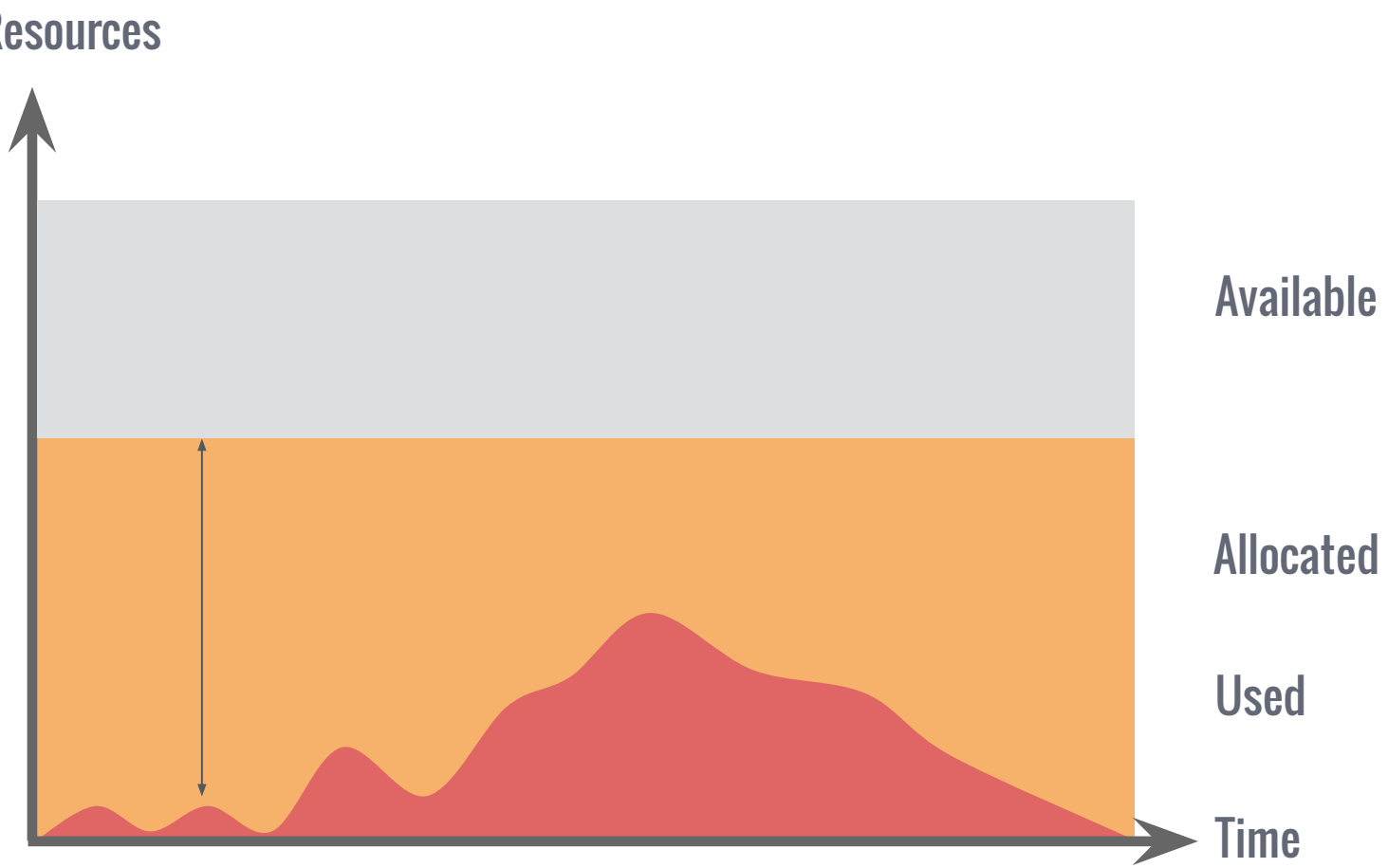
Resource Estimator



Quality of Service Controller

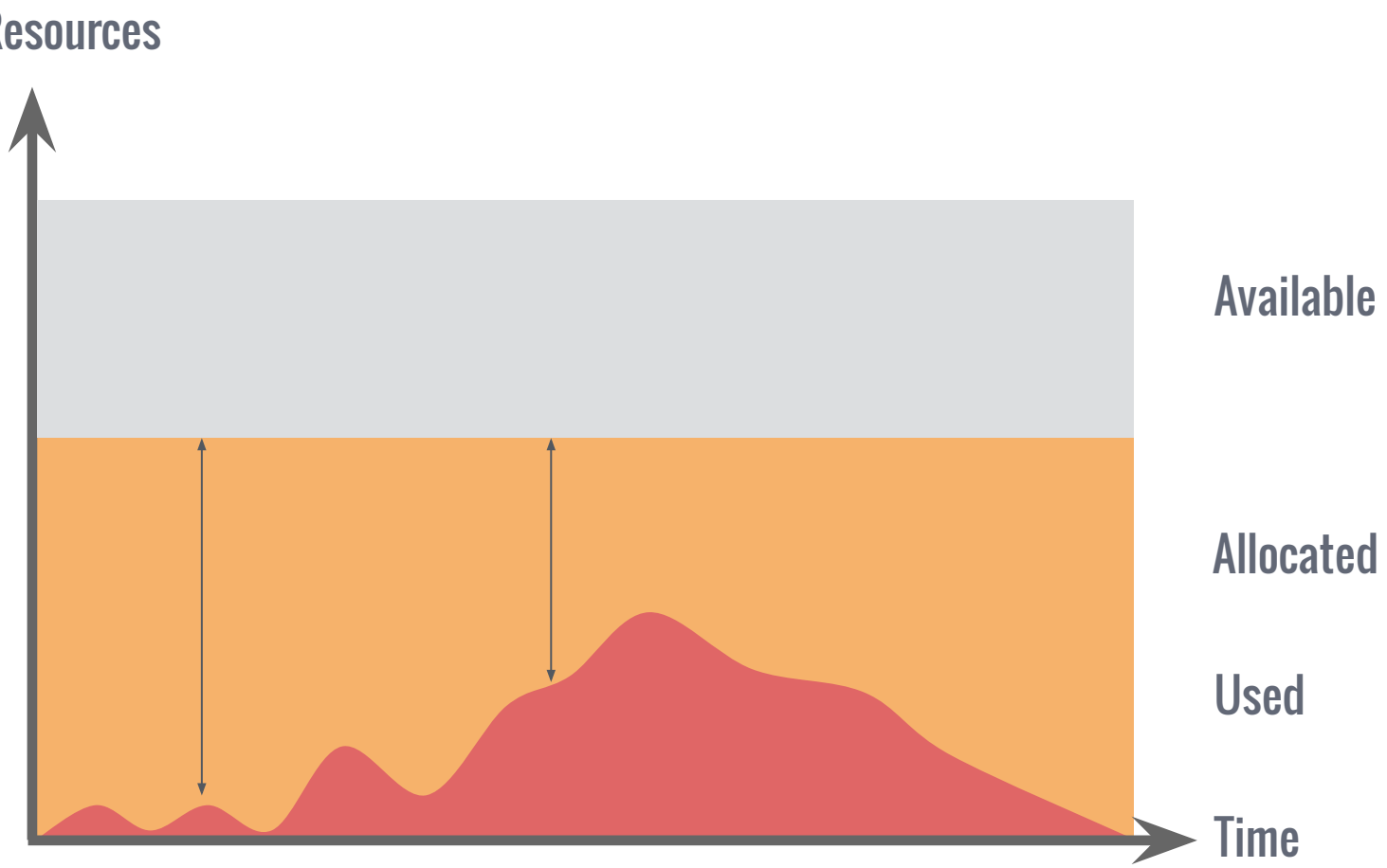
ESTIMATING OVERSUBSCRIBABLE RESOURCES

How many resources should be oversubscribed?



WHAT DO WE DO ABOUT MISPREDICTIONS?

Now, what happens when things change?



THE QoS CONTROLLER

- Can shut down best effort containers
- In the future, it will be able to correct by
  - Freezing
  - Throttling
  - Resizing
  - Cooperating with the framework

MANY RESOURCES CANNOT BE ISOLATED

- Logical units on the chip
- Last level caches
- Memory bandwidth
- I/O
- Chip power supply

OVERSUBSCRIPTION WITH INTEL: SERENITY

https://goo.gl/jWtu7V

ALSO IN THE WORKS

- Quotas ensure minimum set of resources for frameworks
- Optimistic offers enables resource parallelism
- Cooperative preemption through Inverse offers
- Persistence primitives (for storage)