Process Migration in the Orchestration World

Isabel Jimenez

Distributed Systems Engineer
DC/OS Security Team & Apache Mesos Contributor

isabel@mesosphere.io
@ijimene

Kapil Arya

Distributed Systems Engineer
Apache Mesos Committer & DMTCP Developer

kapil@mesosphere.io
@karya0

MESOSPHERE

Overview

- Motivation
- Process Migration
- Apache Mesos
- Process/Container Migration for Mesos
- Demo

Motivation

Stateless vs. Stateful Applications

- Stateless applications:
  - No local state
  - Start from a (relatively) vanilla state
  - Perform transaction(s)
  - Kill when no longer needed

- Stateful application:
  - Some local state
  - Start from vanilla state and compute “work” state
  - Non-graceful shutdown results in loss of compute time

Scheduling Stateless vs. Stateful Applications

- Stateless applications:
  - Scale up: “on-demand” deployment by launching clones as needed
  - Scale down: kill unused instances without loss of computation time
  - Making room for high-priority task without significant penalty

- Stateful application:
  - Scale up: longer initialization times for new instances
  - Scale down: wait for instances to reach a “safe” state to preserve compute cycles.
  - Making room for high-priority tasks results in significant compute-time penalty

Similarly for moving applications from one node/cluster to another!

Scheduling Stateless vs. Stateful Applications

Modern container orchestration tools are optimized for stateless applications!

How to Better Schedule Stateful Applications?

Make them stateless!

- How?
  - Rewrite ‘em!

- Alternatively
  - Use process/container checkpointing and migration!

Process Migration

Terminology

- Process Migration
  - Move a running process from one node to another
- Container Migration
  - Move a running container from one node to another
- Virtual machine migration (e.g., vMotion)
  - Move a running virtual machine from one node to another

How to Migrate a Process/Container/VM?

1. Pause the running process/container/VM
2. Take a snapshot of the current state a.k.a. ***checkpointing***
3. Move the snapshot to the target node
4. Restart from the snapshot on the target node

Do this transparently to the outside world!

- Ensure minimal downtime
  - Reduce time required for stages (2) and (3)
  - Ideally on the order of milliseconds!

What is Checkpointing?

Checkpoint-Restart is the ability to save a set of running processes to a checkpoint-image on disk, and to later restart it from disk.

- A quick demo!

Checkpointing Use Cases

- Fault tolerance
- Scheduling and process migration
- Debugging (an executable bug report)
- Faster startup times (checkpoint after initialization)
- Save/restore workspace (for interactive sessions)
- Speculative execution (what-if scenarios)
- Managing long tails (single thread continues to run after other threads have exited)

Stateful Applications with Checkpointing

Stateful Application + Checkpointing ≈ Stateless Application

- Scale up: start from pre-initialized snapshot
- Scale down: checkpoint and kill
- Migrate: checkpoint, kill, and restart

How to Checkpoint/Restart a Process?

Checkpoint-restart involves saving and restoring:

- all of user-space memory
- state of all threads
- kernel state
- network state
- …

All this while ensure the state doesn't change while taking a checkpoint!

- Quiesce the process(es) before saving the state!

Different types of Checkpointing

- Application-level
  - Embed checkpointing code inside the application itself
  - Optimal
  - Burden on the application developer

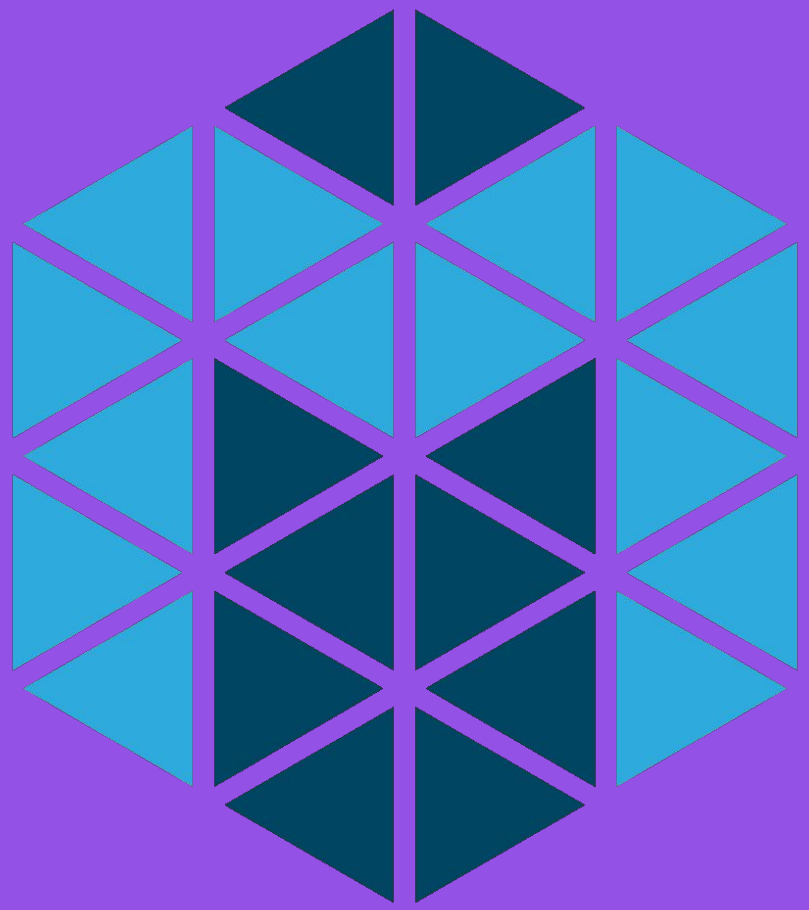
- Virtual machine level
  - Complete state
  - Higher cost

- System-level
  - No modification to application source/binary
  - Can be done at the kernel-level or in the user-space

Modern Checkpointing Systems

- CRIU (Checkpoint Restart In Userspace)
  - Single-node checkpointing
  - Recent kernels (3.9+)
  - Container-level
  - http://criu.org/

- DMTCP (Distributed MultiThreaded CheckPointing)
  - User-space libraries with LD_PRELOAD
  - Distributed processes across multiple nodes
  - http://dmtcp.sourceforge.net



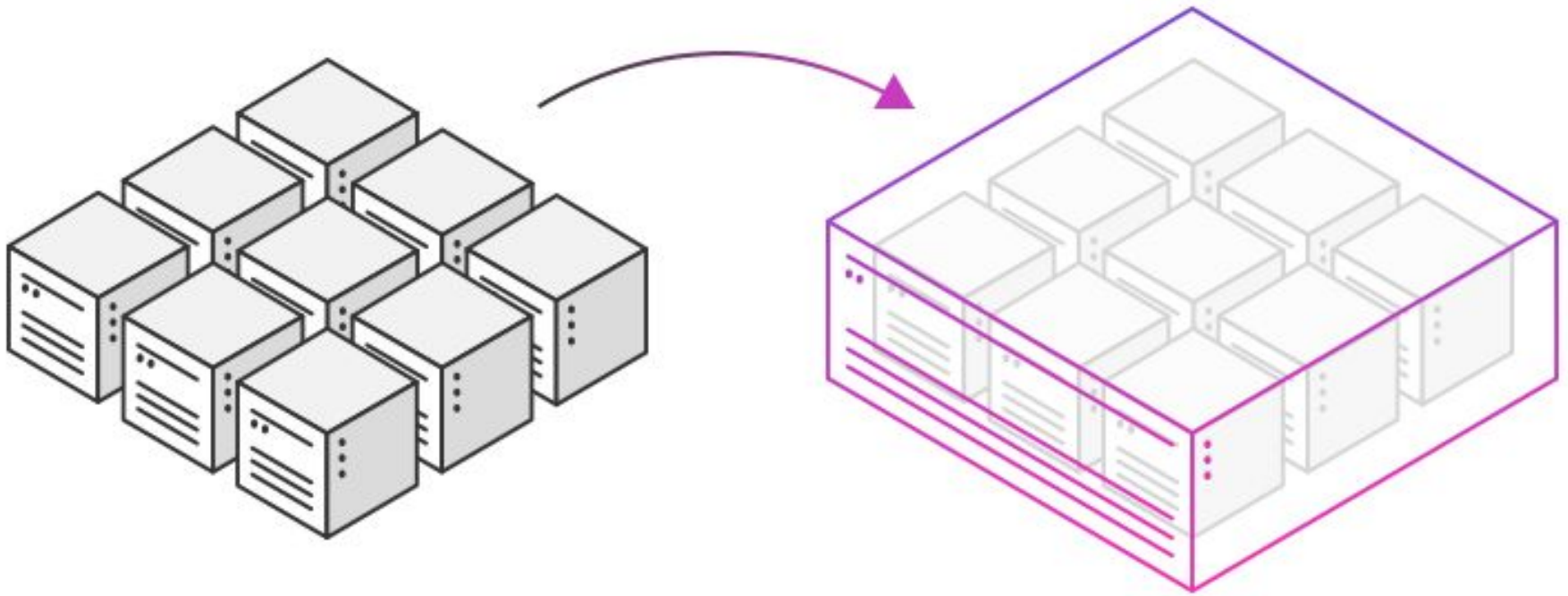
Apache Mesos: The datacenter kernel

Why?

Why can’t we run applications on our datacenters just like we run applications on our mobile phones?

We’re all building distributed systems.

The datacenter abstraction



The datacenter computer needs an operating system

Operating system

“a collection of software that manages the computer hardware resources and provides common services for computer programs”

- Wikipedia

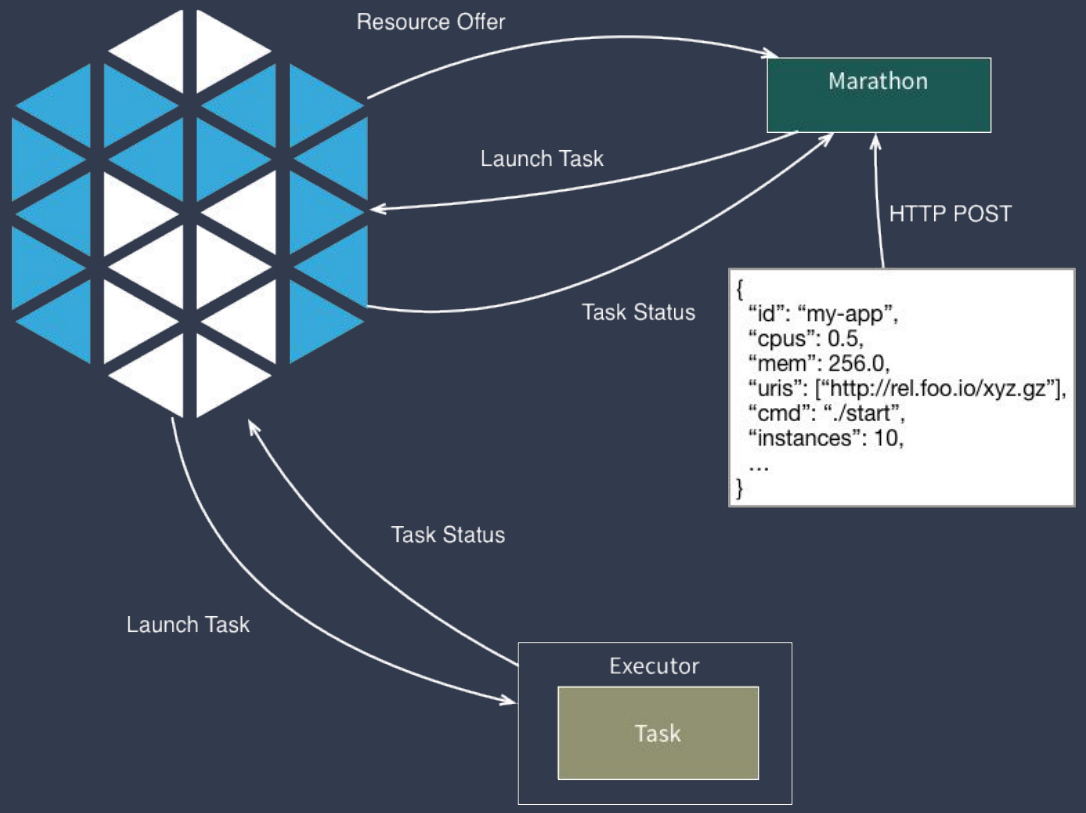
Offer based model

Resource offers

Mesos can’t run applications on its own

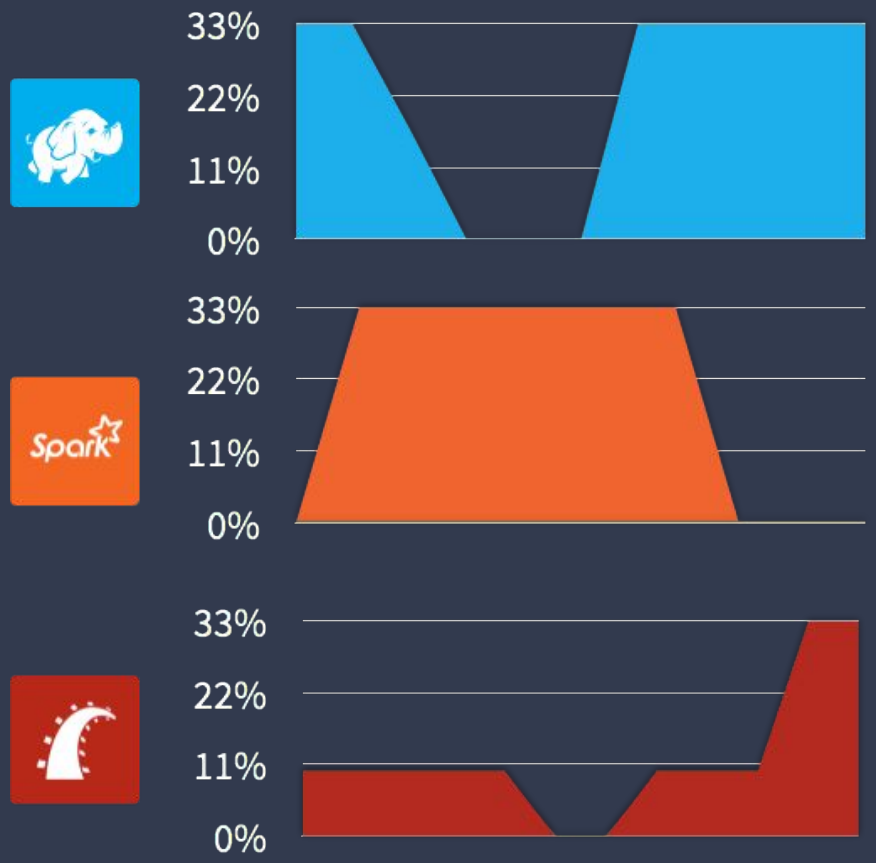
A Mesos framework is a distributed system that has a scheduler.

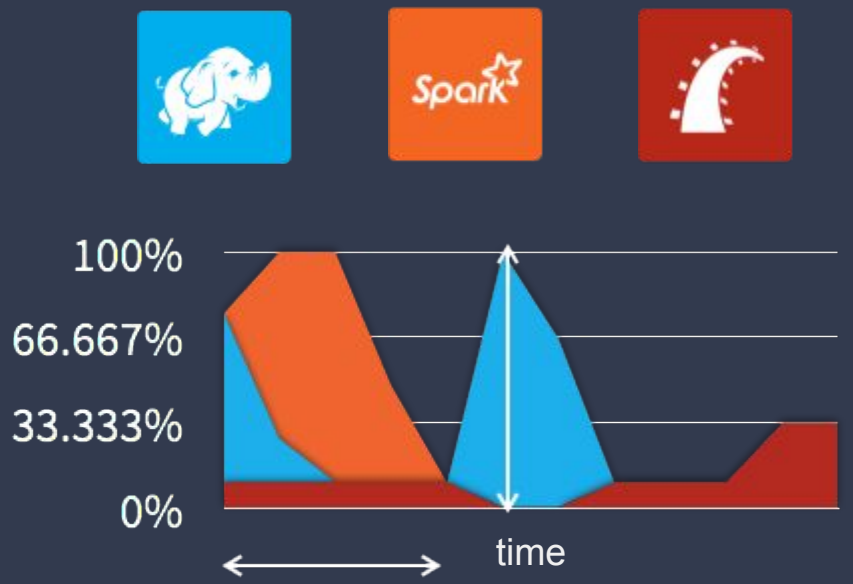
Schedulers like Marathon keeps your application running. A bit like a distributed “init.d”.



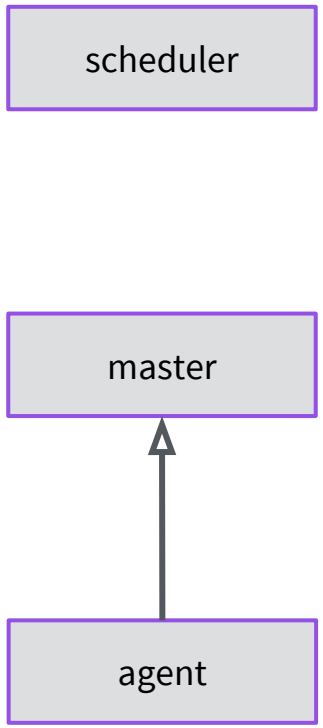
Apache Mesos

High utilization



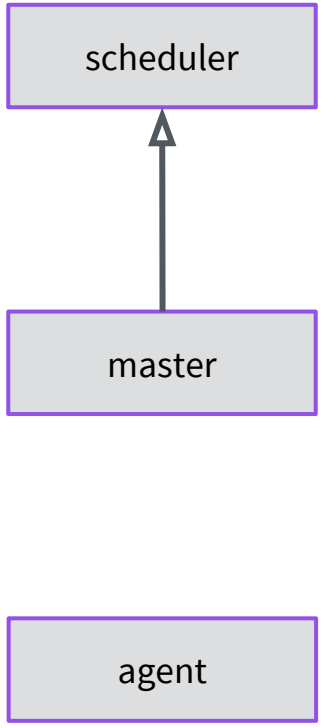


Mesos mechanics



RESOURCES
(cpu, mem, disk, etc)

Mesos mechanics



OFFER
(cpu, mem, disk, etc)

Mesos mechanics

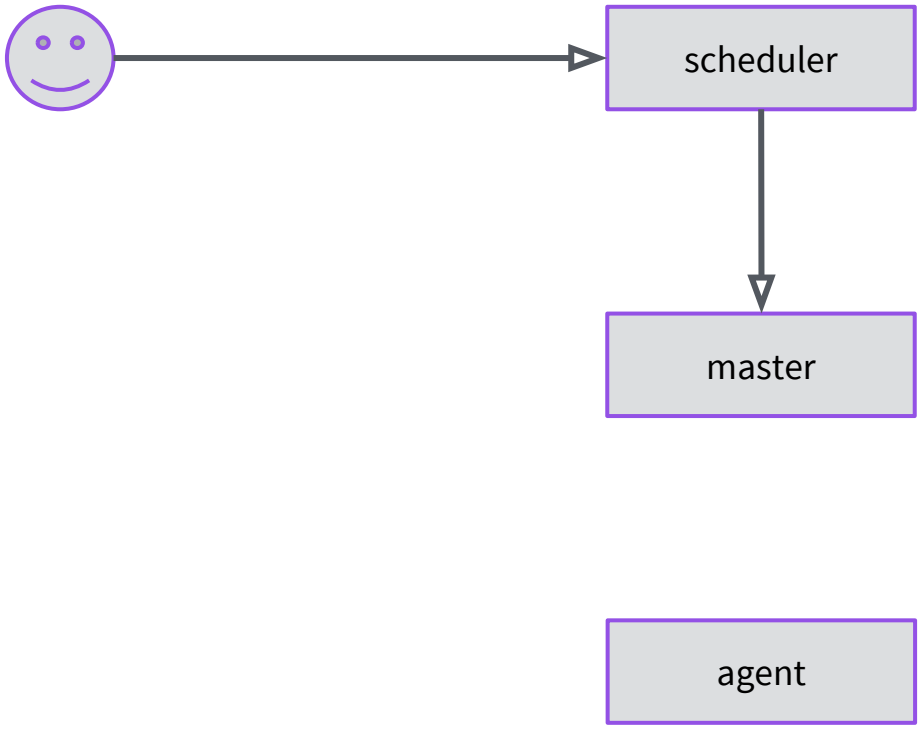
```
{
    "container": {
        "docker": {
            "image": "busybox",
        },
        "type": "DOCKER"
    },
    "cpus": 0.1,
    "id": "demo",
    "instances": 1,
    "mem": 128
}
```

scheduler

master

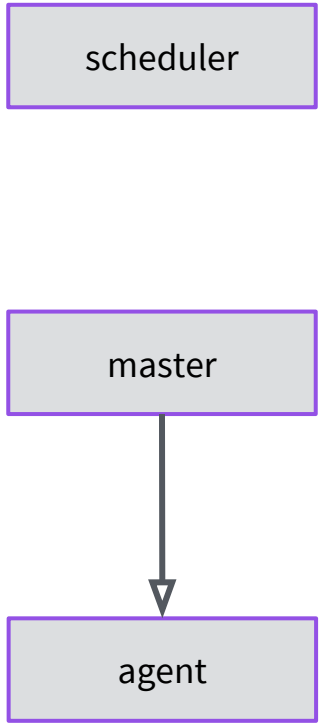
agent

Mesos mechanics



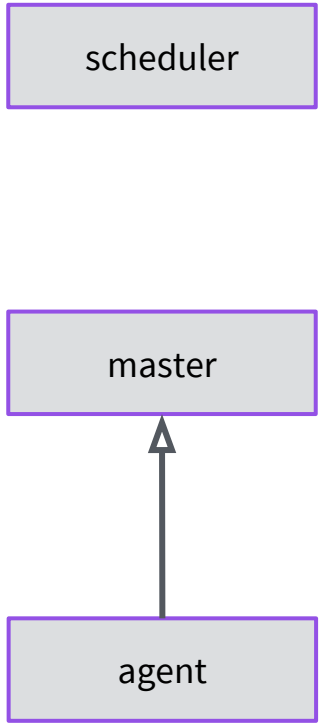
ACCEPT OFFER
(cpu, mem, disk, etc)

Mesos mechanics



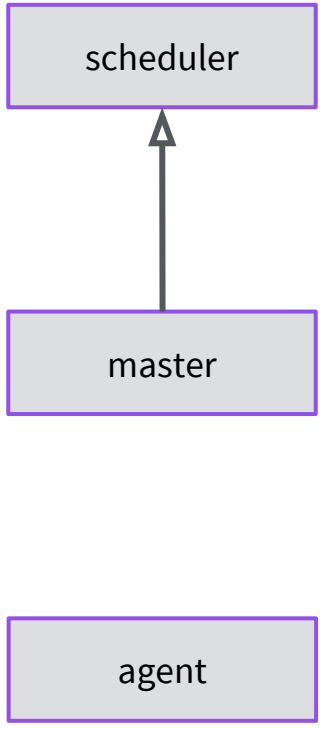
LAUNCH TASK

Mesos mechanics



UPDATE STATE
(STAGING, RUNNING, etc)

Mesos mechanics

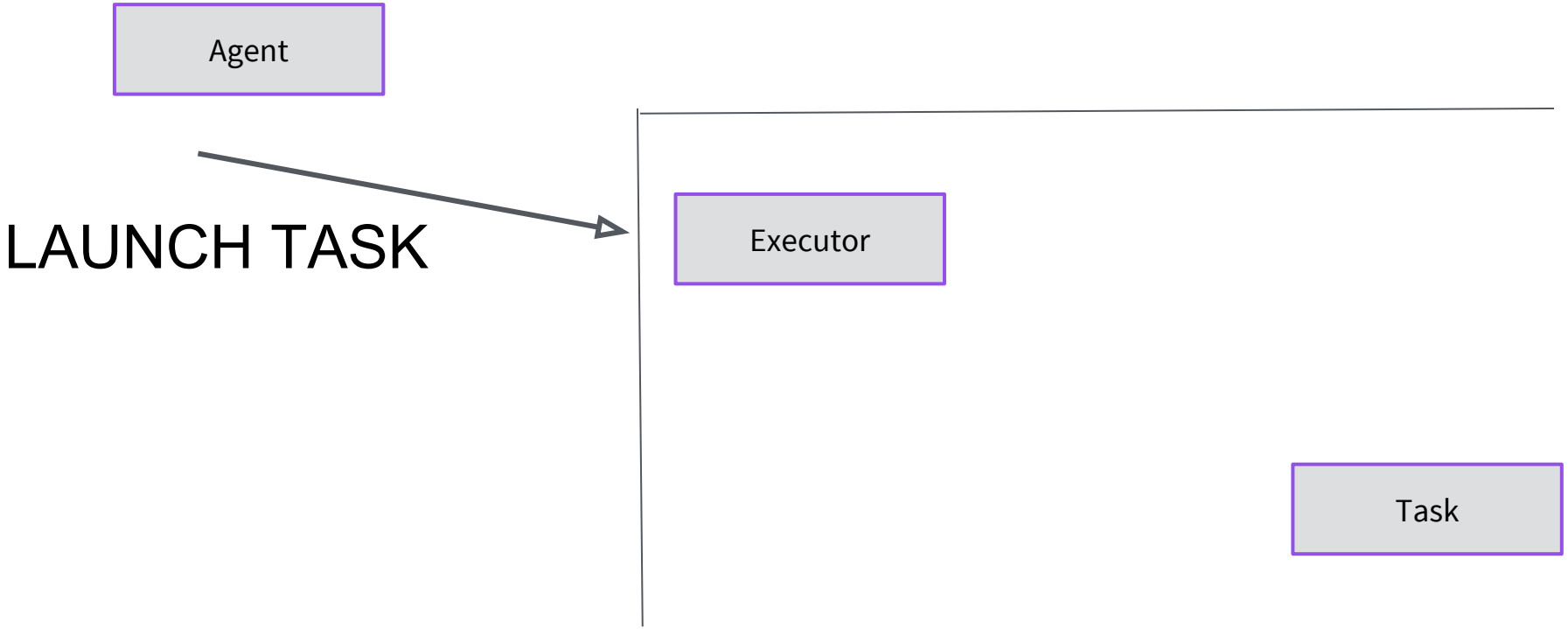


UPDATE STATE
(STAGING, FAILED, etc)

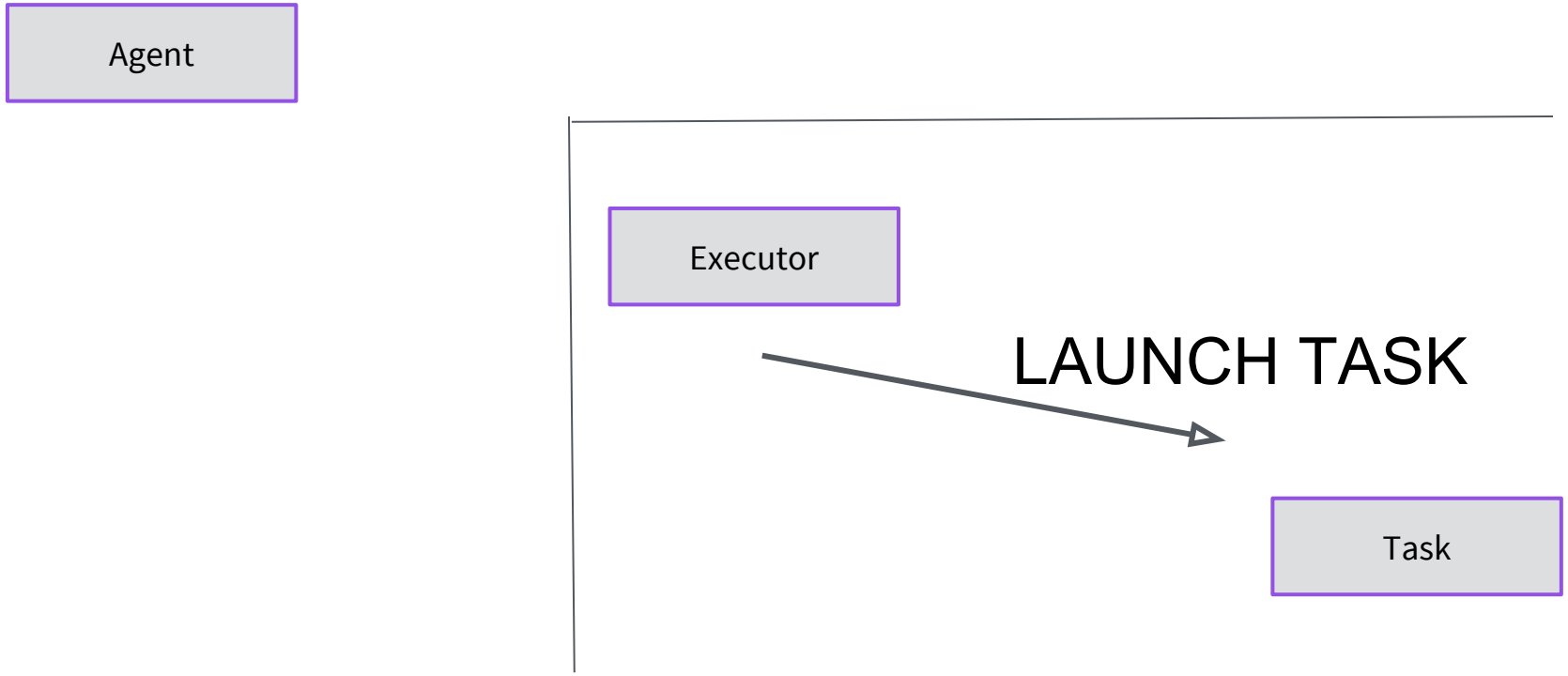
Mesos mechanics: Custom executor

```
{
    "container": {
        "docker": {
            "image": "busybox",
        },
        "type": "DOCKER"
    },
    "cpus": 0.1,
    "id": "demo",
    "executor": demo-executor,
    "mem": 128
}
```

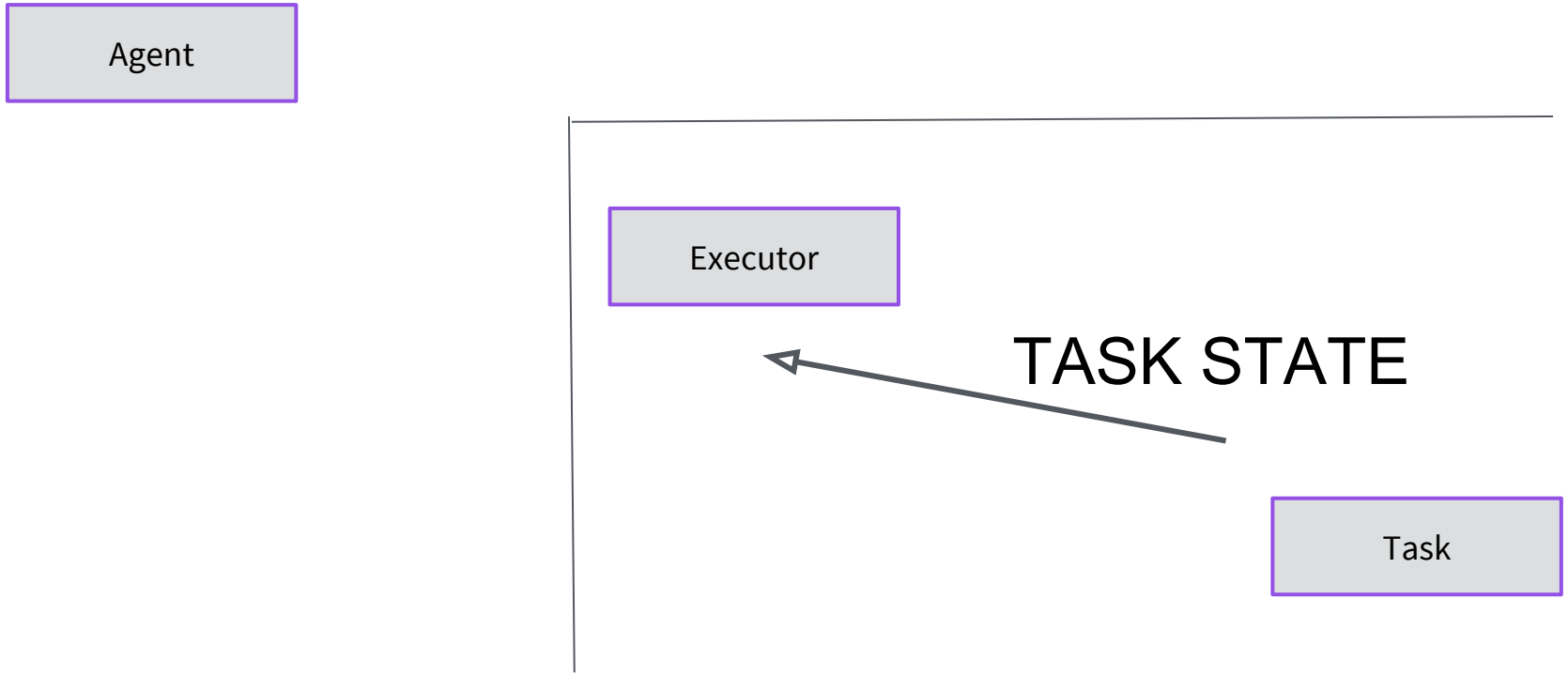

Mesos mechanics



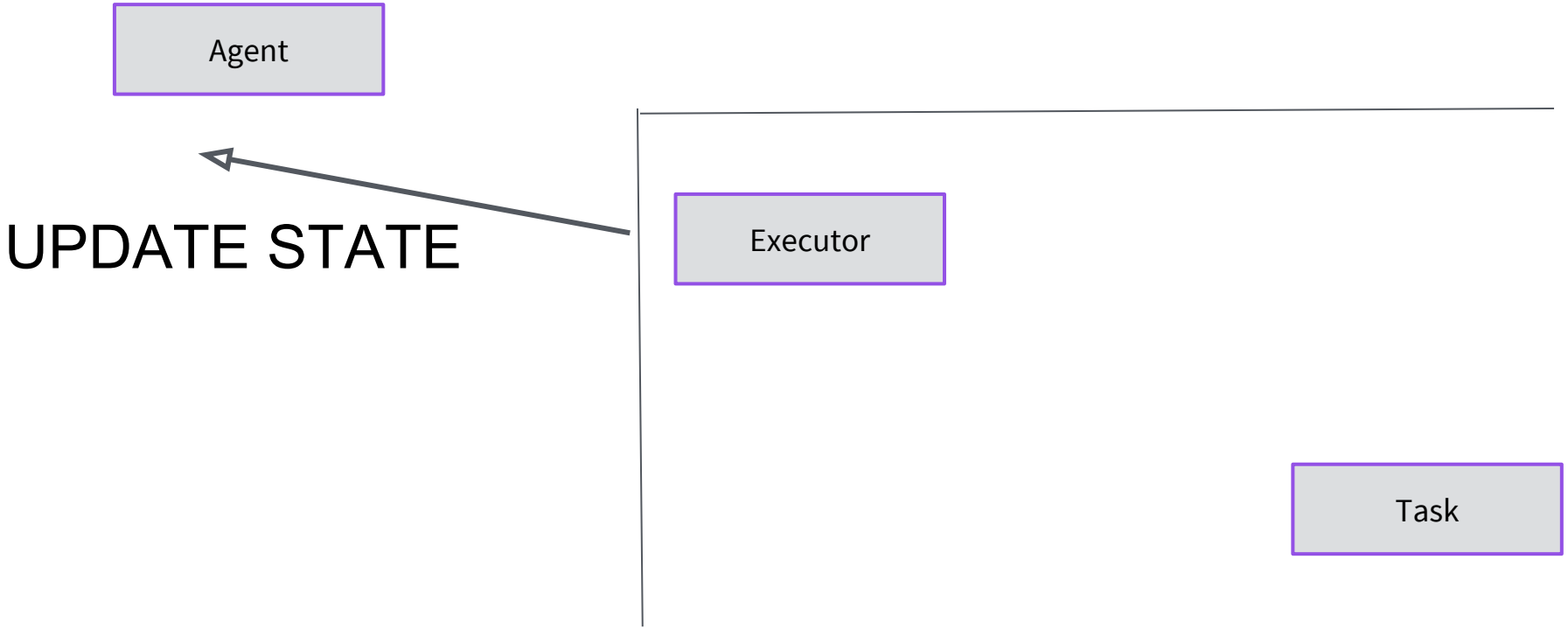
Mesos mechanics



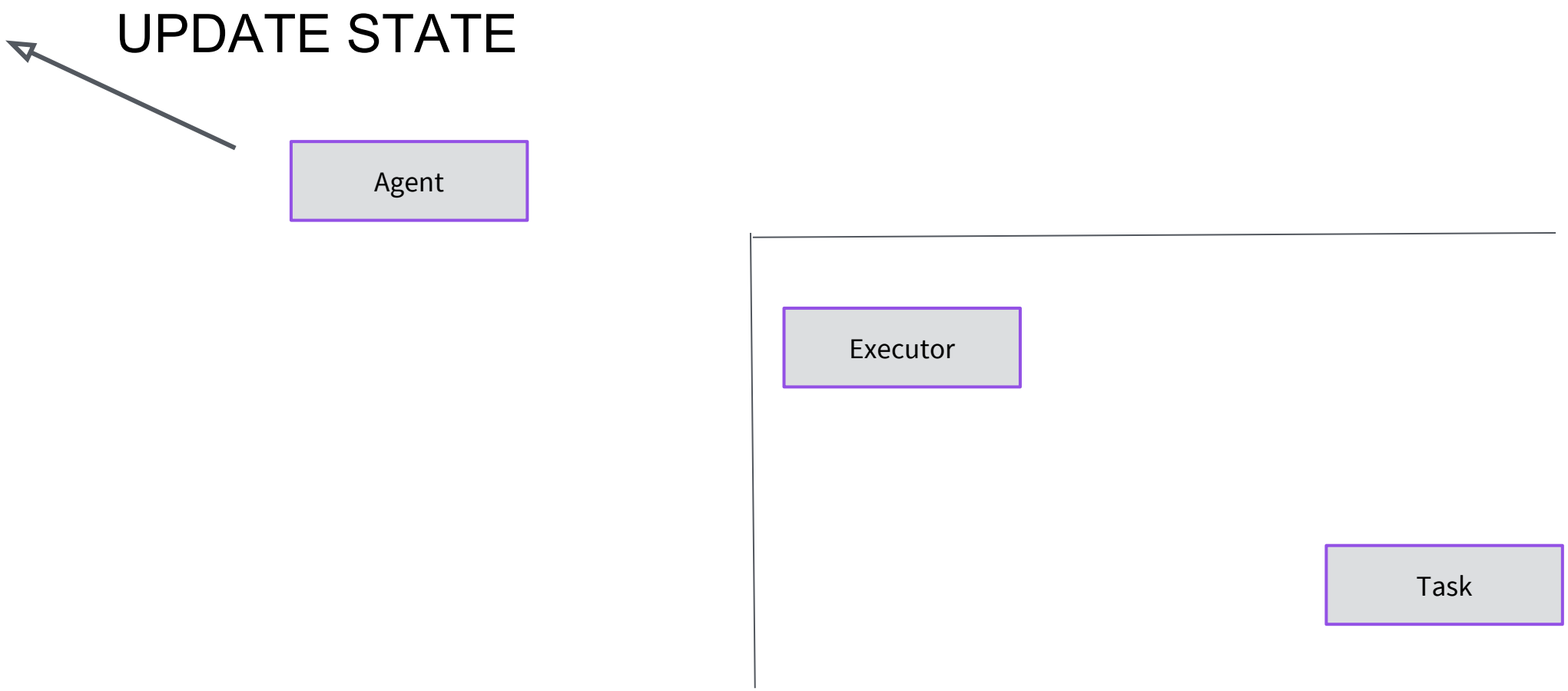
Mesos mechanics



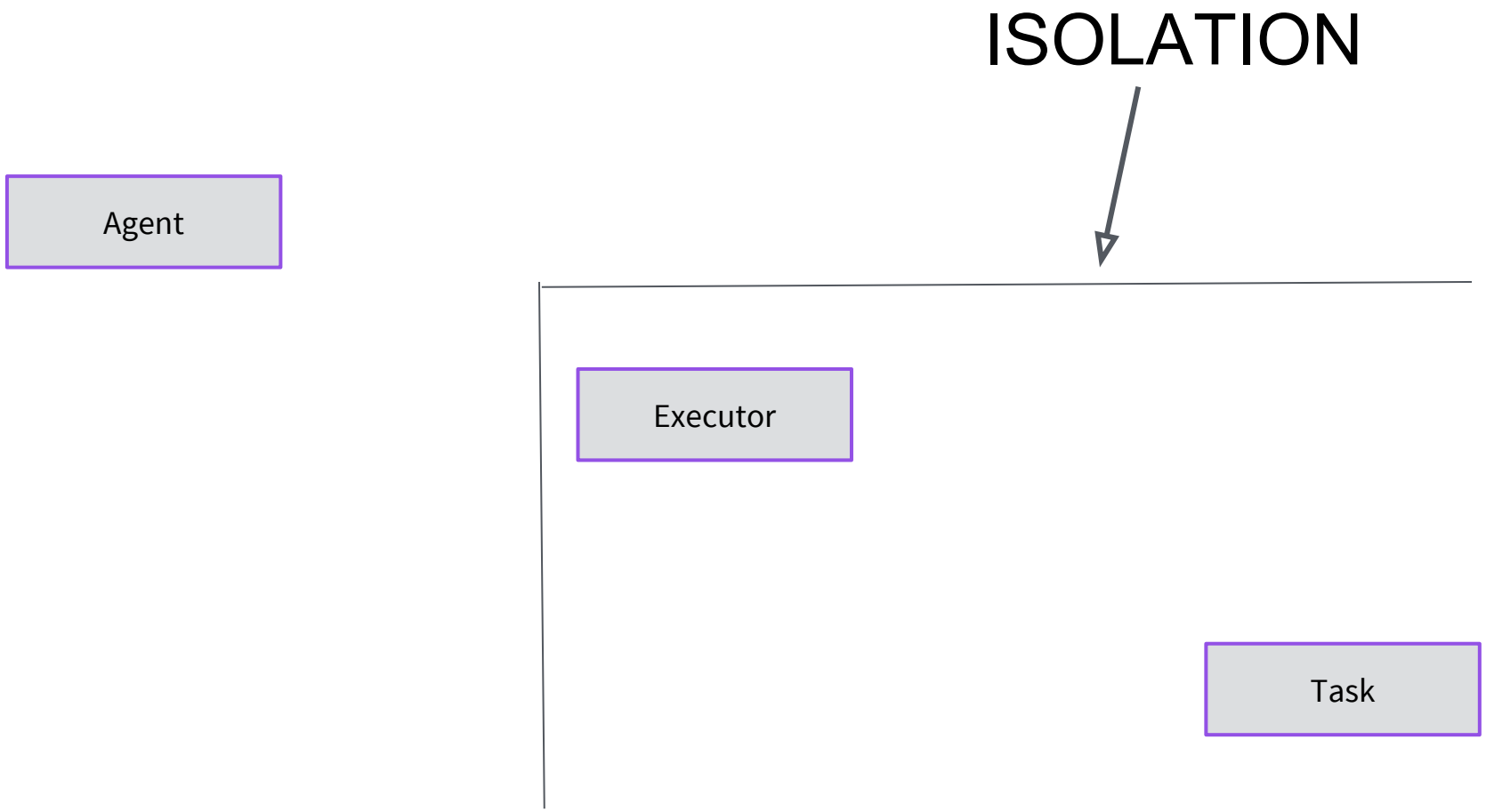
Mesos mechanics



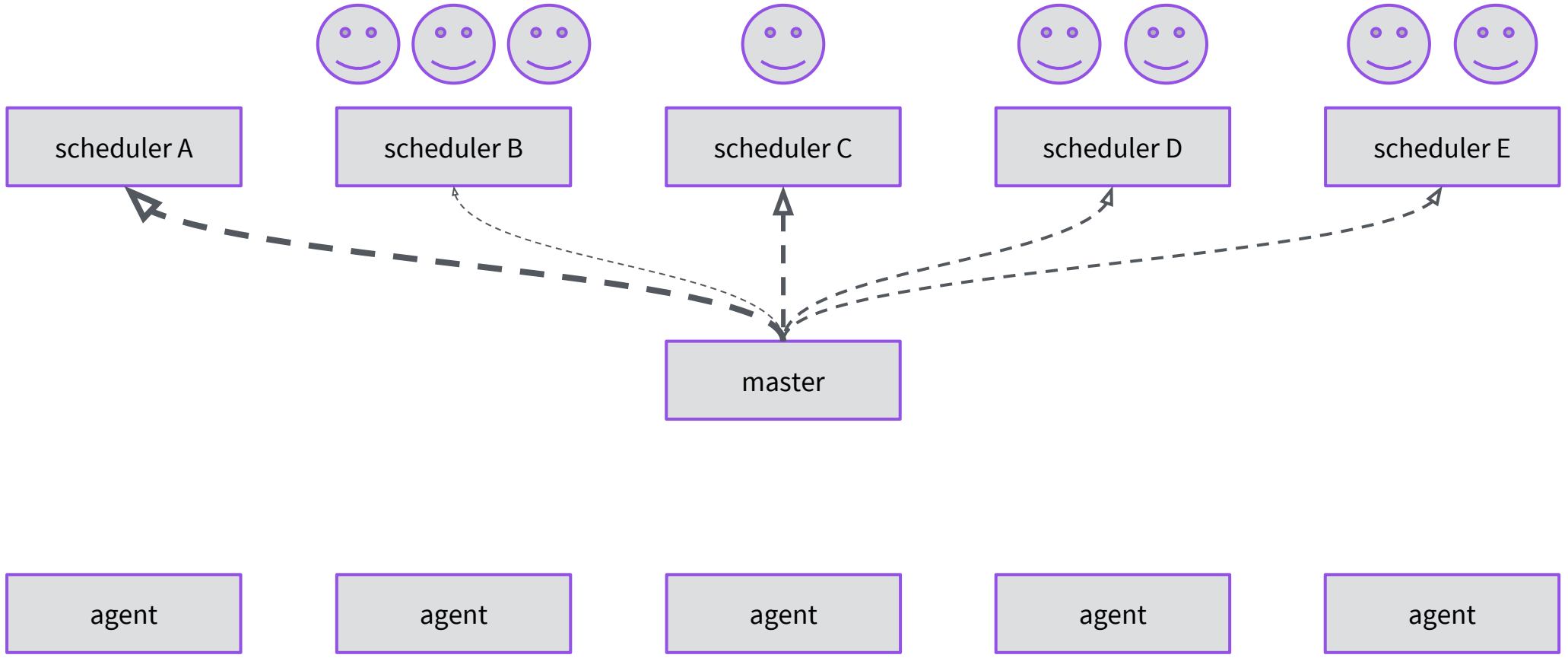
Mesos mechanics



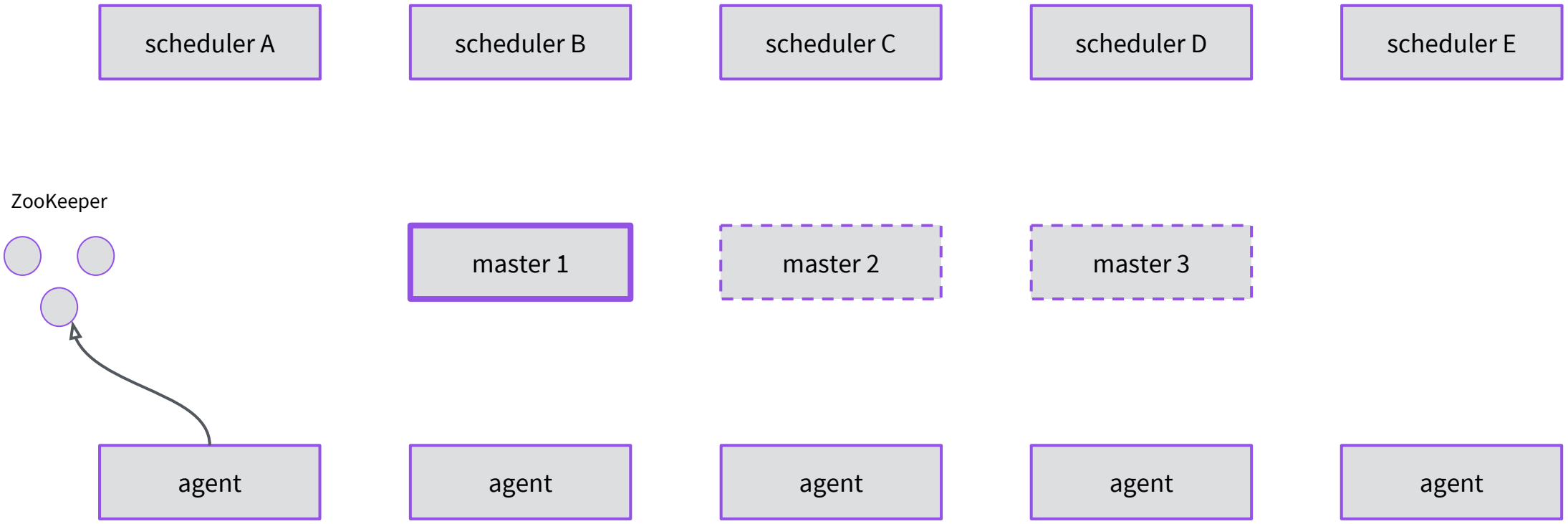
Mesos mechanics



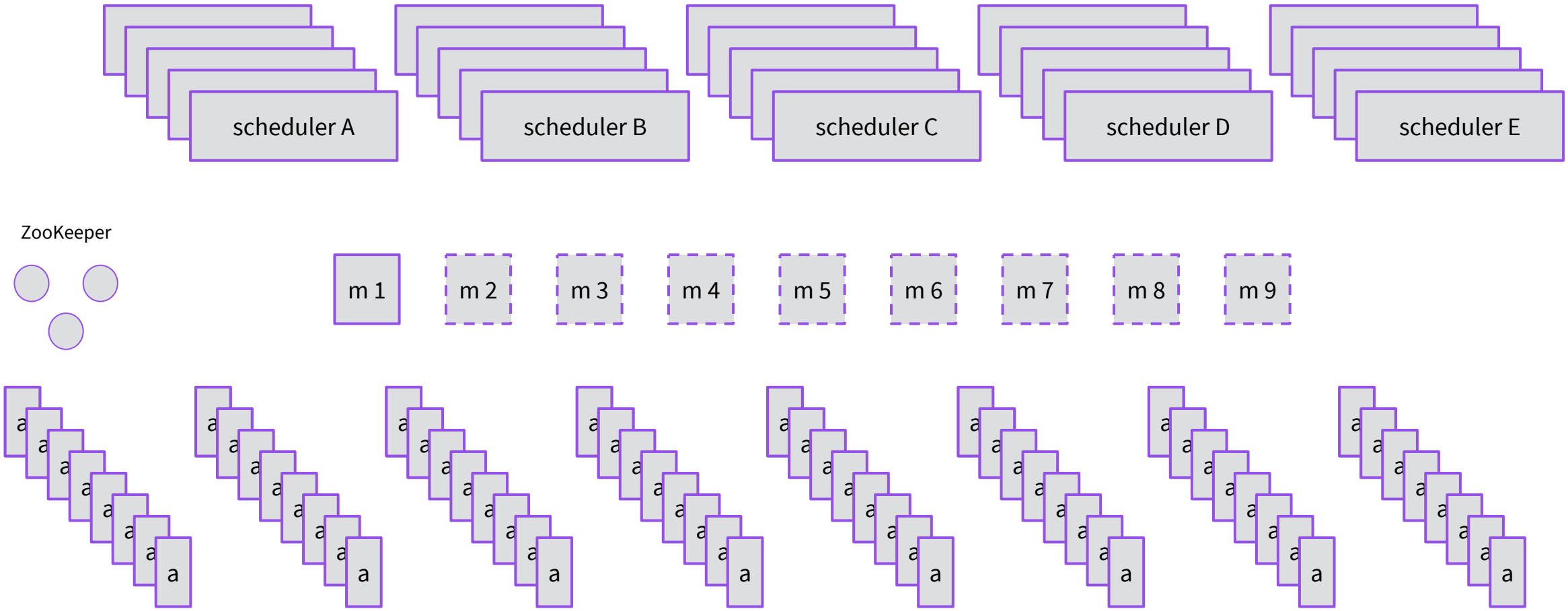
Mesos mechanics are fair



Mesos mechanics are HA



APACHE MESOS: Putting it all together



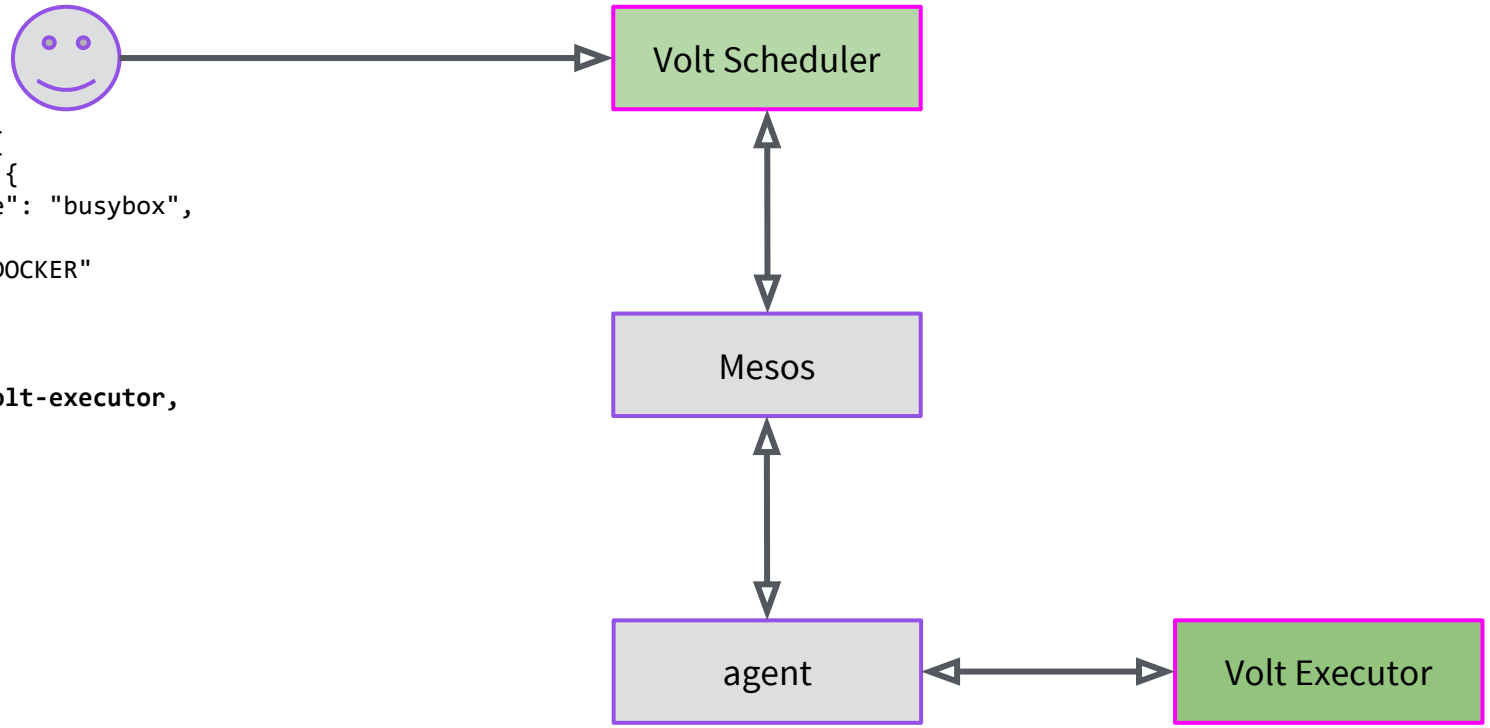
Mesos Container Migration

RUNC

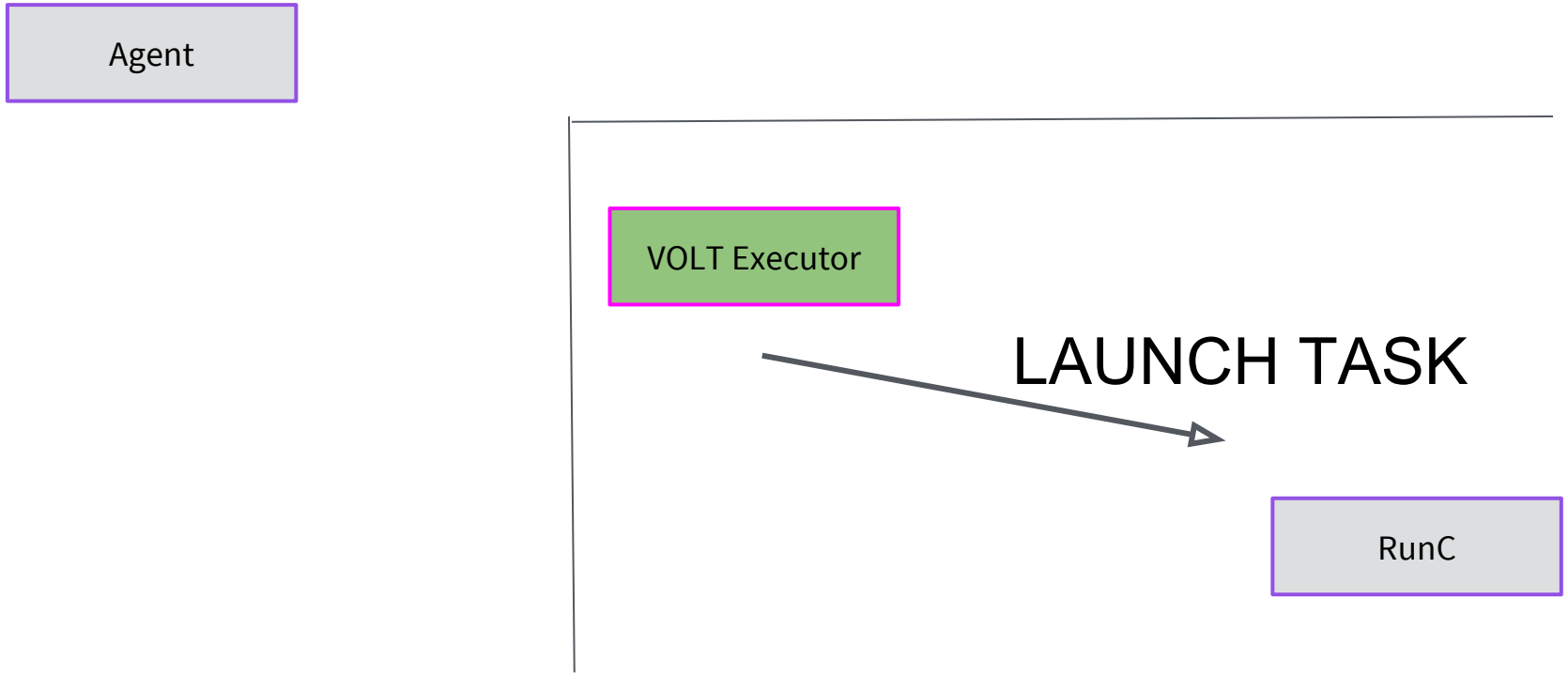
- *OCI specification*
- *Well integrated with CRIU*
- *Lightweight universal runtime container*
- *Compatible with Docker*

Mesos mechanics: Custom executor

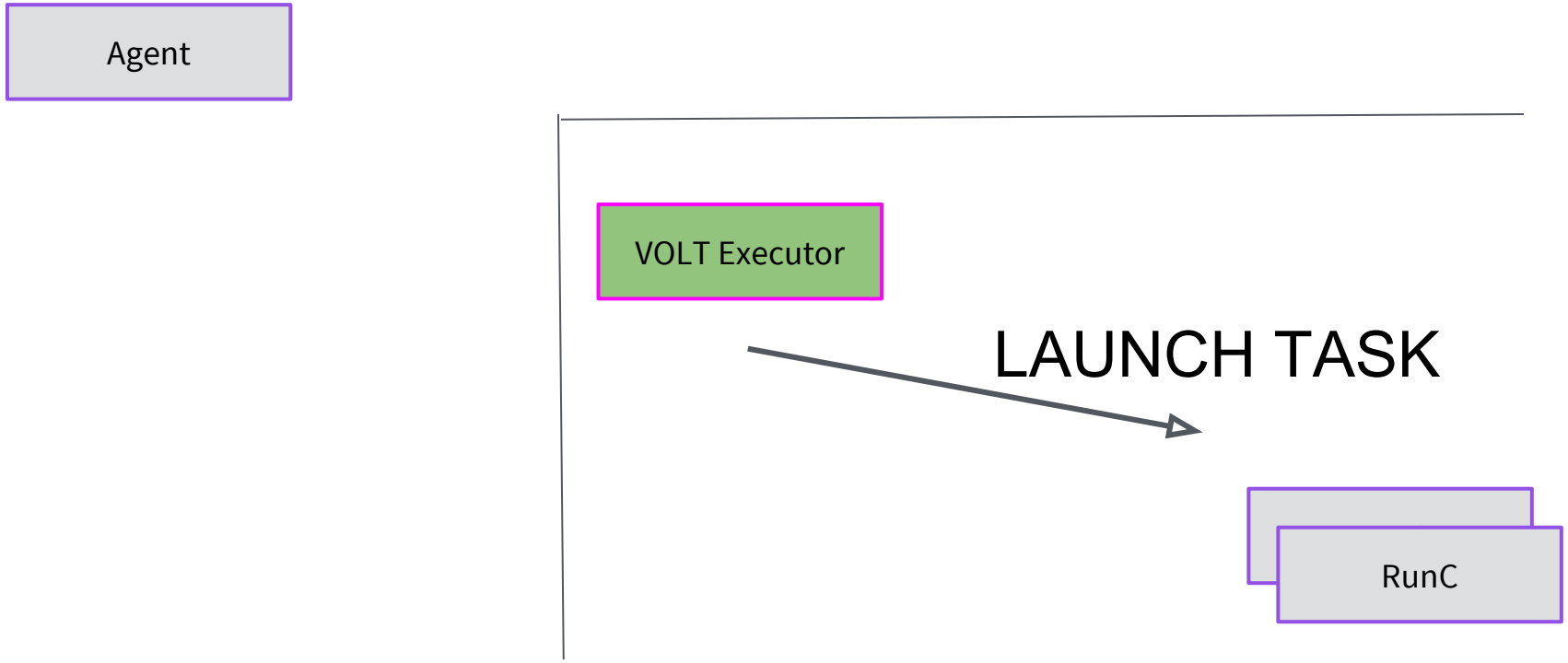
```
{
    "container": {
        "docker": {
            "image": "busybox",
        },
        "type": "DOCKER"
    },
    "cpus": 0.1,
    "id": "demo",
    "executor": volt-executor,
    "mem": 128
}
```



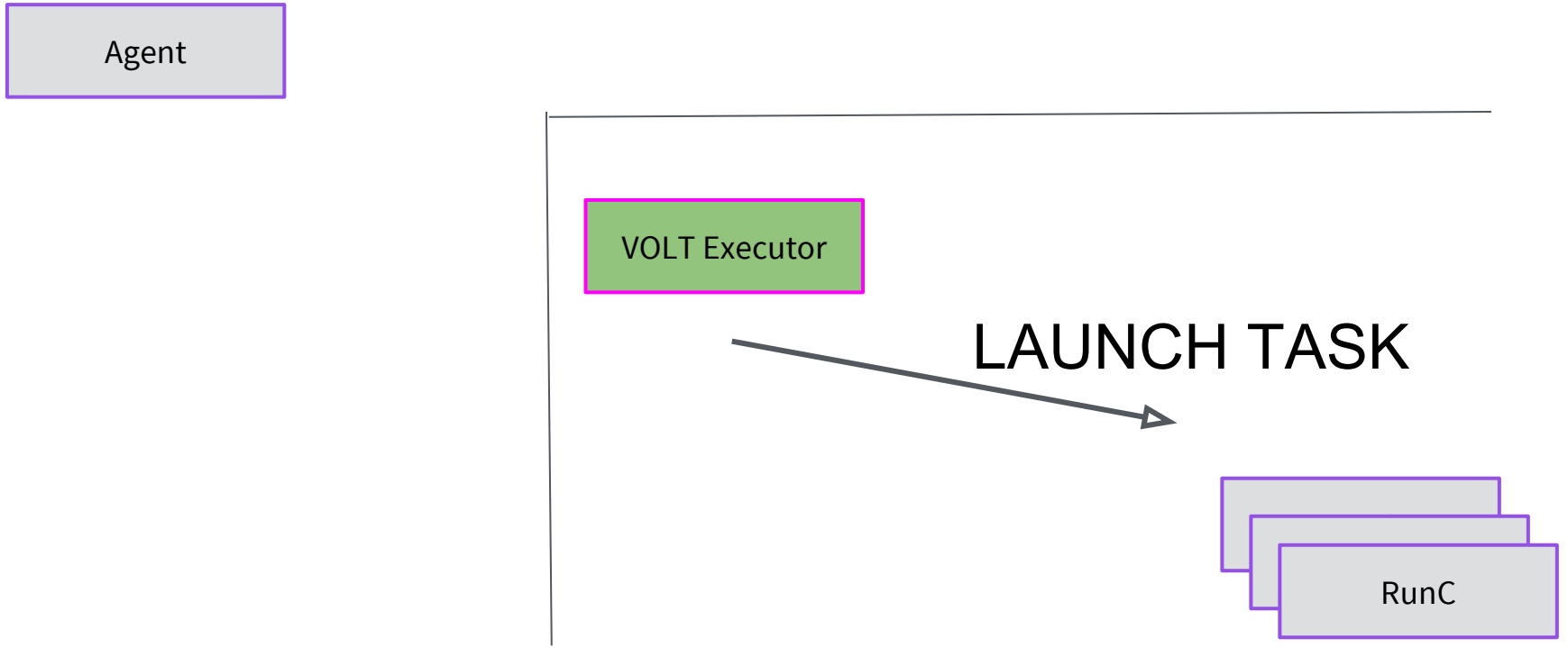
Mesos mechanics



Mesos mechanics



Mesos mechanics



Demo!

Future Work: Checkpointing as a Service

First class integration with Mesos

- Transparent to the scheduler and executor
- New tasks states (CHECKPOINTED, RESTORING, etc)
- Support multiple checkpoint-service providers (DMTCP, CRIU, etc)

THANK YOU!