THE LINUX FOUNDATION
OPEN SOURCE SUMMIT

CASE: Mesos Scheduler for Distributed Training

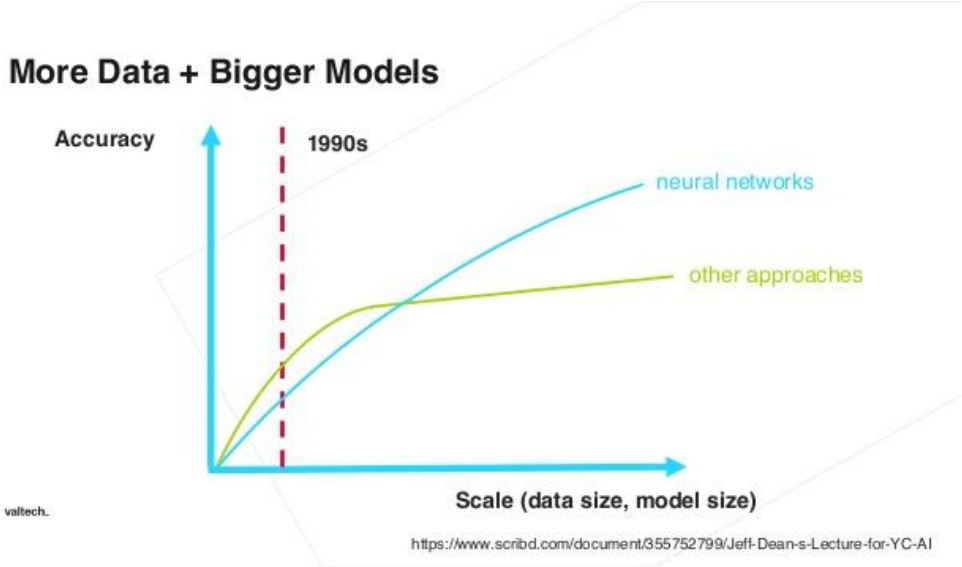
@treadstone90
@BairosNovak

THE LINUX FOUNDATION

Agenda

- Distributed Training
- Mesos and CASE
  - Scheduling Features
    - Implementation in CASE
  - Architecture
- Questions

Machine Learning - Why



ML Infrastructure

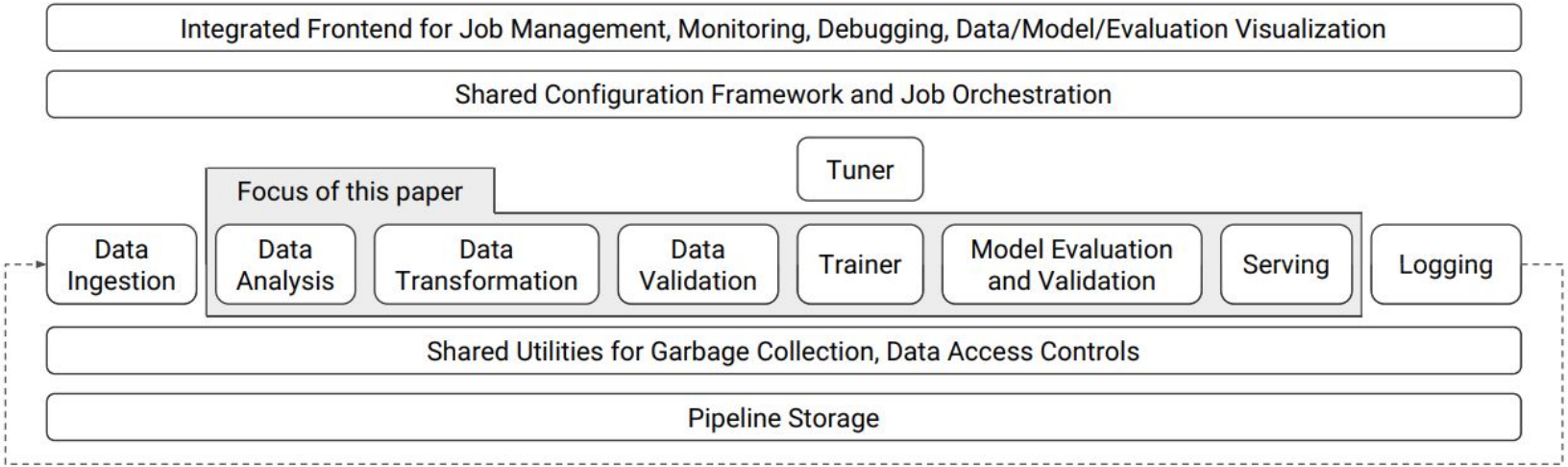


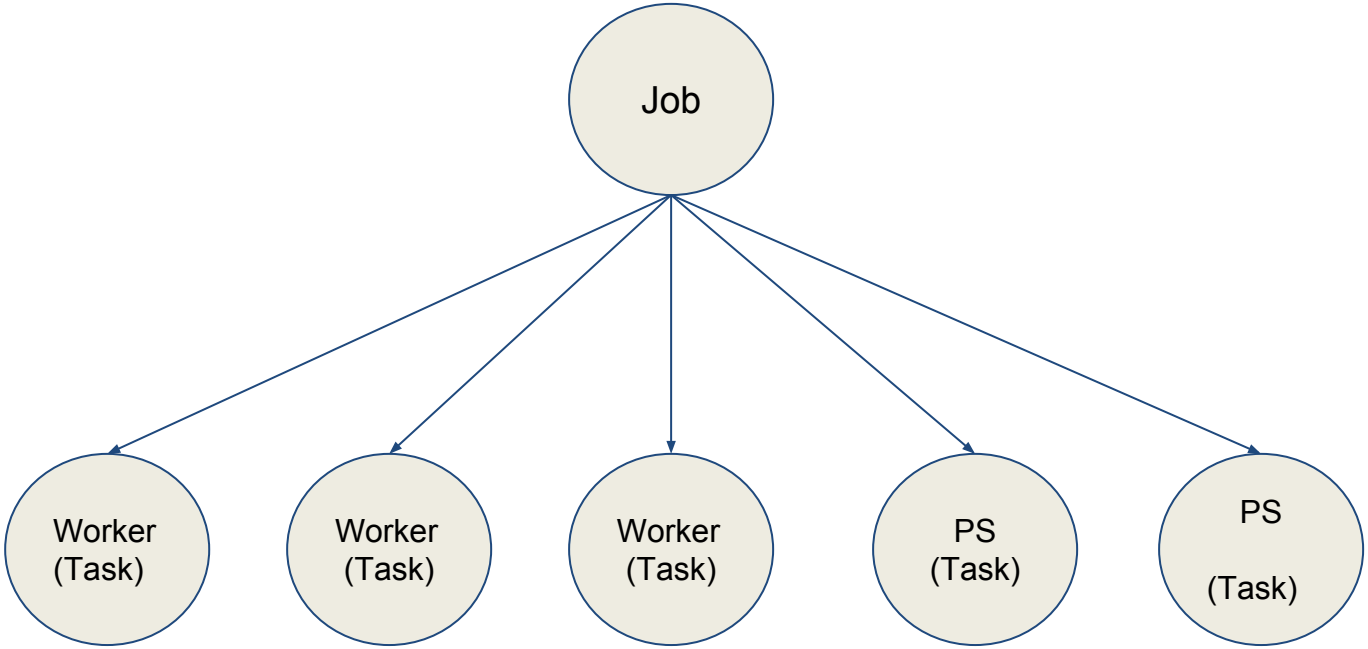
Figure 1: High-level component overview of a machine learning platform.

Baylor, Denis, et al. "Tfx: A tensorflow-based production-scale machine learning platform." *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, 2017.

Distributed Training

- Data does not fit on single box
- Model does not fit on single box
- Cut down training time
  - Training is impractically slow to train on large datasets and large neural networks

Jobs and Tasks



Focus Of This Talk - Job Scheduler

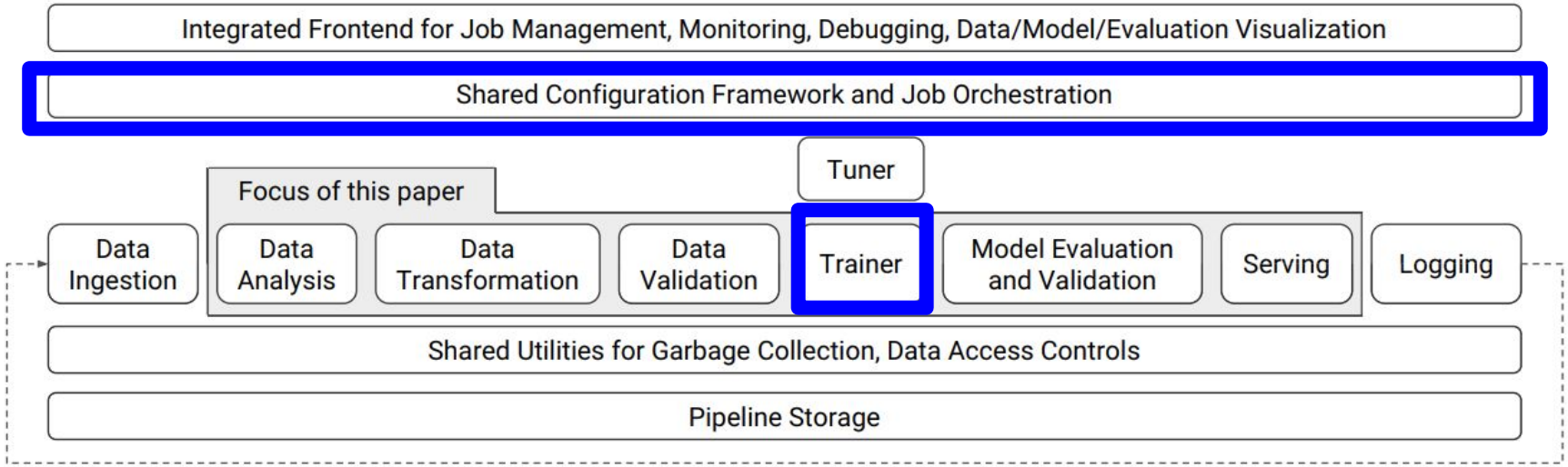


Figure 1: High-level component overview of a machine learning platform.

Baylor, Denis, et al. "Tfx: A tensorflow-based production-scale machine learning platform." *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, 2017.

Job Orchestration Systems

kubernetes

MESOS

HashiCorp
Nomad

Mesos frameworks

- Long Running Services
  - Apache Aurora
  - Marathon
  - Titus
- Batch
  - Cook - More suitable for Spark Jobs
  - Titus
  - Apache Aurora

Orchestration for Distributed Training

- Bootstrap every task with a set of environment variables
  - Rank
  - Locations of other tasks
- Manage lifecycle of tasks
  - How do we deal with failures?
  - When is a job considered complete?
  - Can we restart a task on failure?

Example: TF_CONFIG

```
{
   "cluster":{
      "ps":[
         "host1:2222",
         "host2:2222"
      ],
      "worker":[
         "host3:2222",
         "host4:2222",
         "host5:2222"
      ]
   },
   "task":{
      "type":"worker",
      "index":1
   }
```

Example: PyTorch

- `MASTER_PORT`: A free port on the machine that will host the process with rank 0.
- `MASTER_ADDR`: IP address of the machine that will host the process with rank 0.
- `WORLD_SIZE`: The total number of processes, so that the master knows how many workers to wait for.
- `RANK`: Rank of each process, so they will know whether it is the master of a worker.

Orchestration for Distributed Training

- Bootstrap every task with a set of environment variables
  - Rank
  - Locations of other tasks
- Manage lifecycle of tasks
  - How do we deal with failures?
  - When is a job considered complete?
  - Can we restart a task on failure?

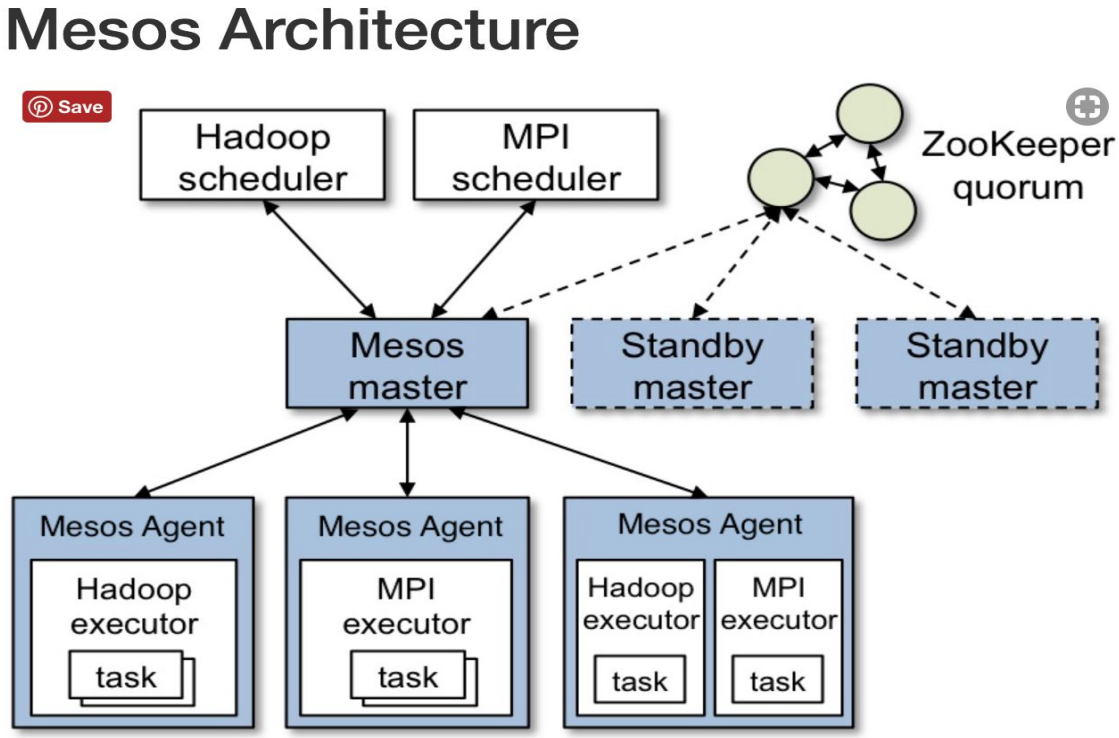
CASE

- Schedules jobs in a resource constrained cluster
  - How should we prioritize training jobs among different users?
- Scheduling features specific to distributed ML training.
- Module architecture with support for multiple frameworks.

Agenda

- Distributed Training
- **Mesos and CASE**
  - Scheduling Features
    - Implementation in CASE
  - Architecture
- Questions

Mesos Architecture

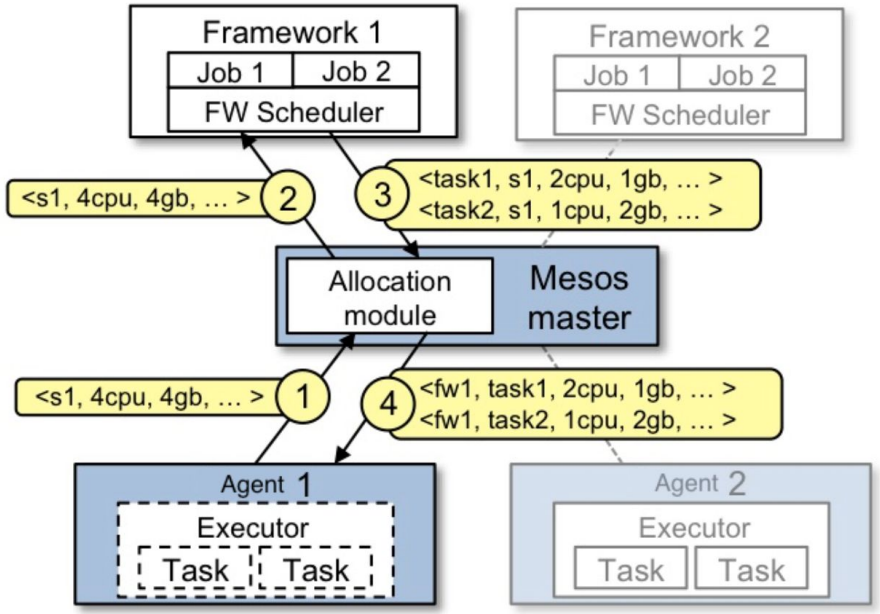


Hindman, Benjamin, et al. "Mesos: A Platform for Fine-Grained Resource Sharing in the Data Center." *NSDI*. Vol. 11. No. 2011. 2011.

Mesos Offer Cycle

Example of resource offer

The figure below shows an example of how a framework gets scheduled to run a task.



Hindman, Benjamin, et al. "Mesos: A Platform for Fine-Grained Resource Sharing in the Data Center." *NSDI*. Vol. 11. No. 2011. 2011.

Agenda

- Distributed Training
- Mesos and CASE
  - **Scheduling Features**
    - Implementation in CASE
  - Architecture
- Questions

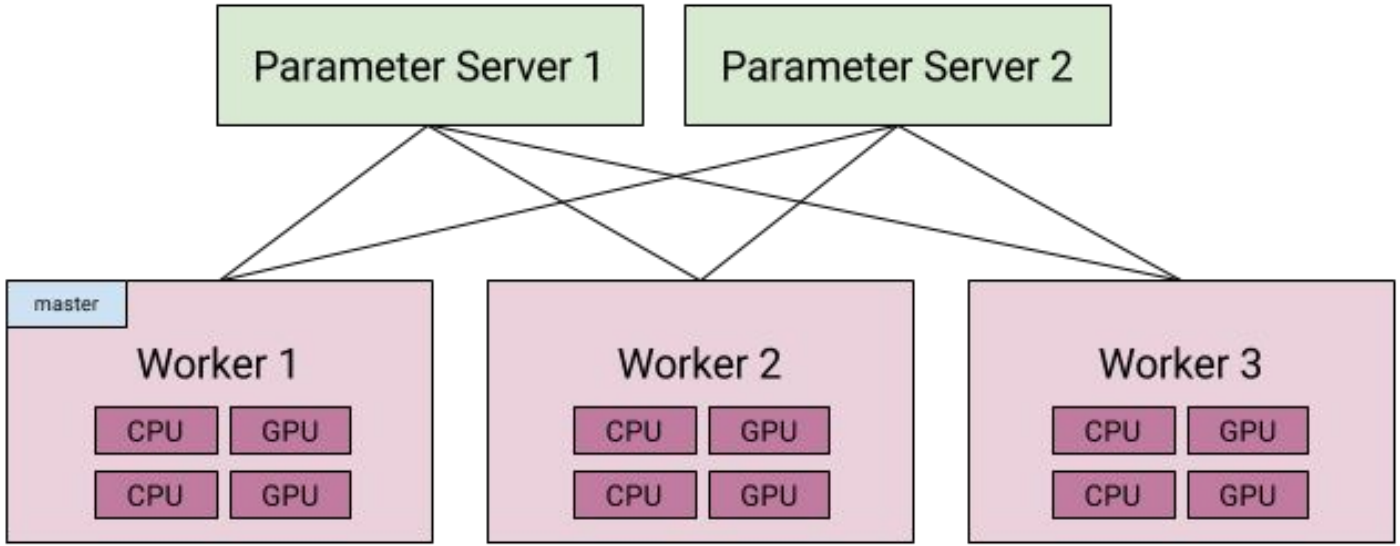
Scheduling Features

1. Gang scheduling
2. Job Fairness
3. Spread vs Pack

1. Gang Scheduling

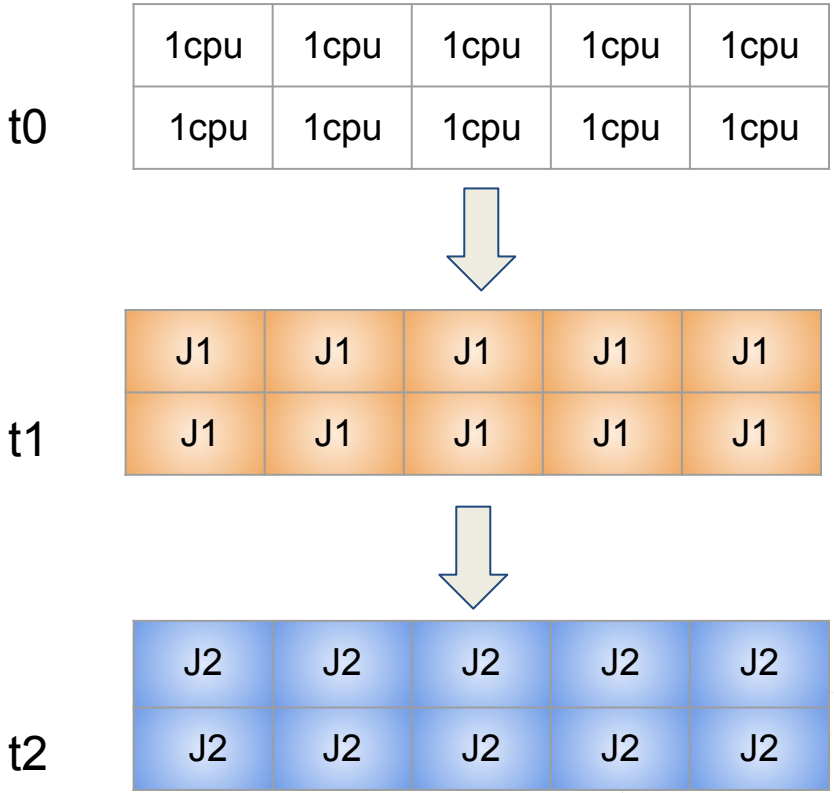
- Schedule related processes to run simultaneously
- Requires that all process be running so as to make progress
- Make the tasks discoverable to each other

Tensorflow Example



Example:

- 10 cores in cluster
- Job1
  - 10 tasks, 1 core each
- Job2
  - 10 tasks, 1 core each



Deadlock

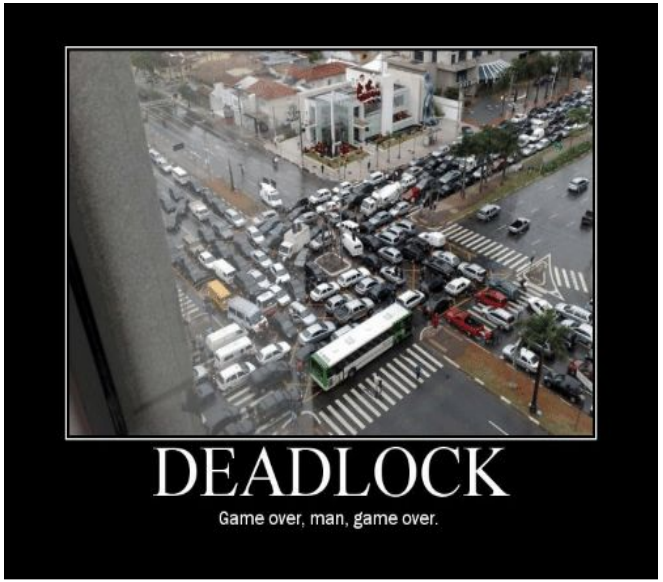
- If not scheduled “correctly” can result in deadlock

t0

J1	J1	J1	J2	J2
J1	J1	J1	J2	J2

t1

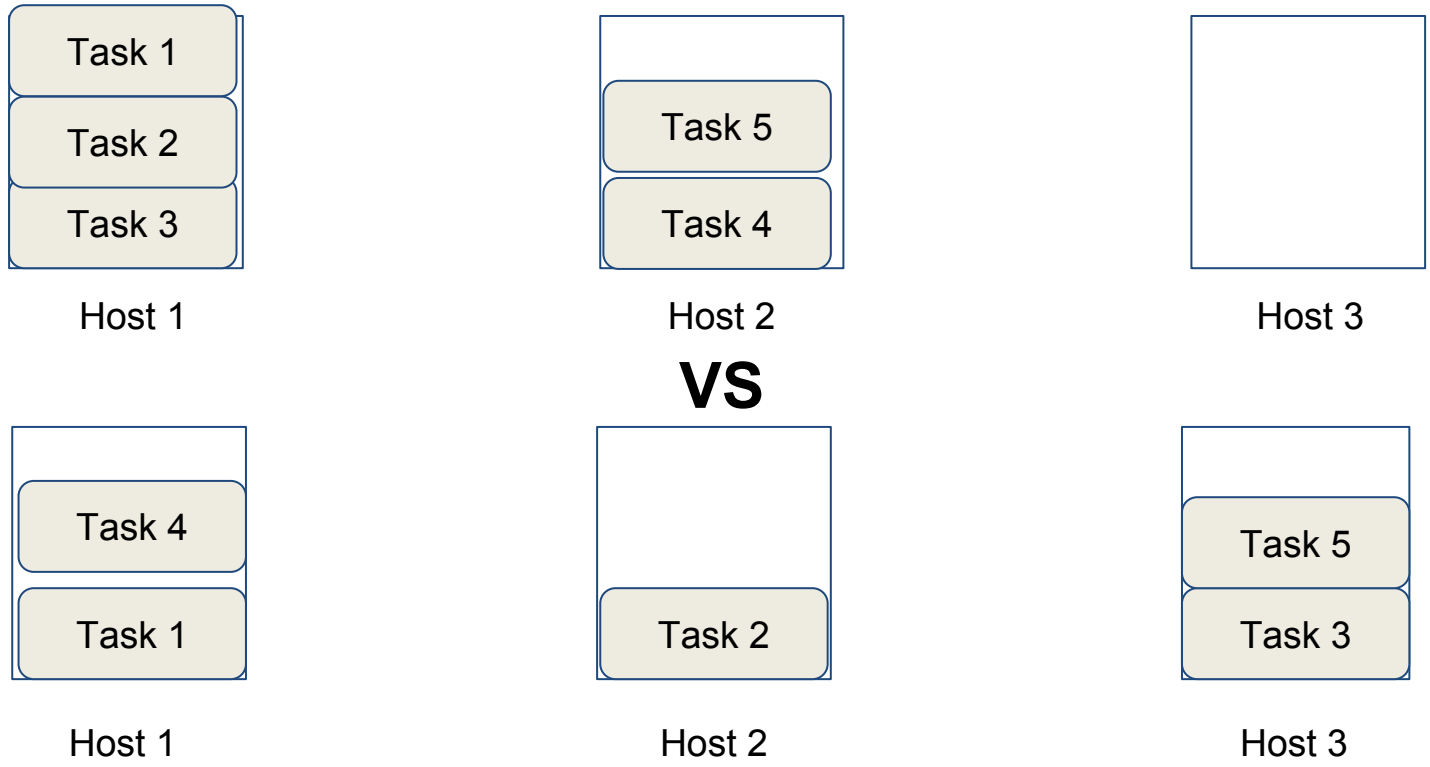
J1	J1	J1	J2	J2
J1	J1	J1	J2	J2



2. Job Fairness

- We want some measure of fairness
- Can’t have a single user monopolize the cluster

3. Pack vs Spread



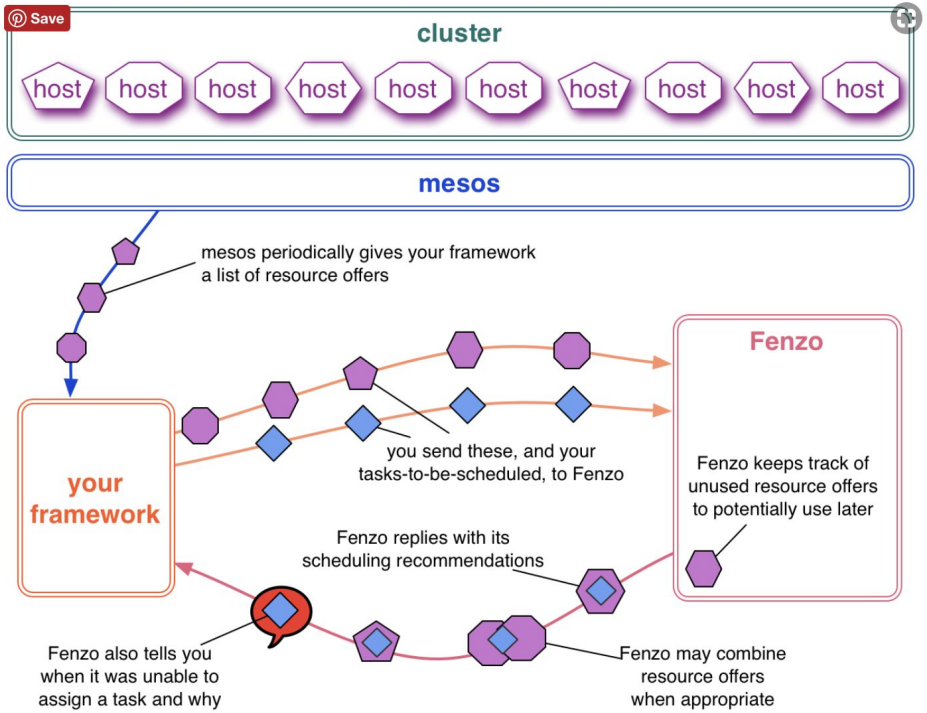
Pack vs Spread

- Pack tasks from a single job
  - Minimize network overhead
- Spread jobs to avoid interference
  - Mesos supports these tasks as containers
  - So we have isolation at cpu, mem, gpu, network disk etc.
  - But PCI Express (PCIe) switch bus for cpu-gpu is shared
  - Non-Prod can affect Prod job

Agenda

- Distributed Training
- Mesos and CASE
  - Scheduling Features
    - **Implementation in CASE**
  - Architecture
- Questions

Fenzo - Scheduling Library



https://github.com/Netflix/Fenzo/wiki

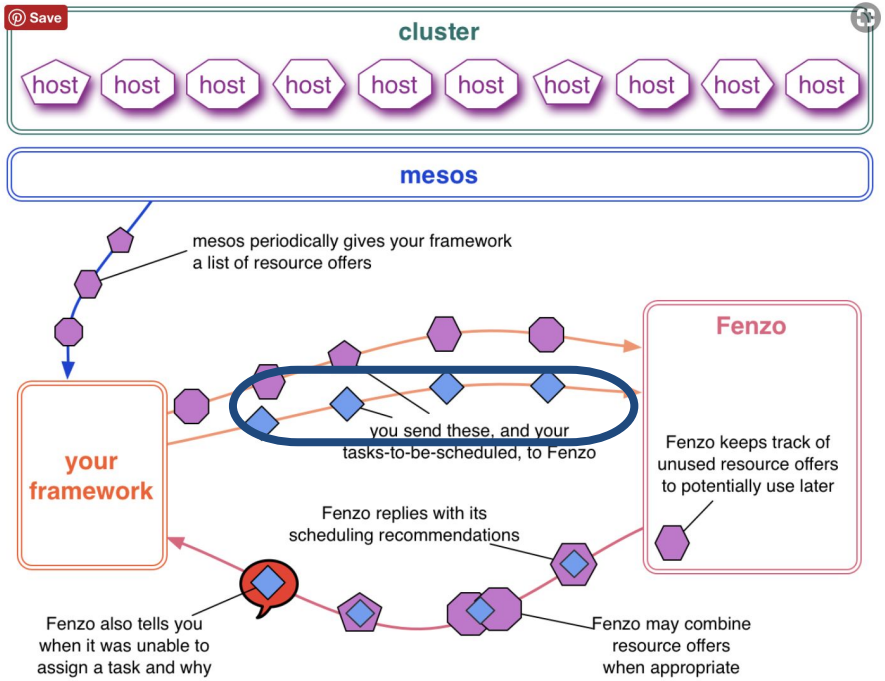
Gang Scheduling in CASE

Total Ordering On Tasks

+

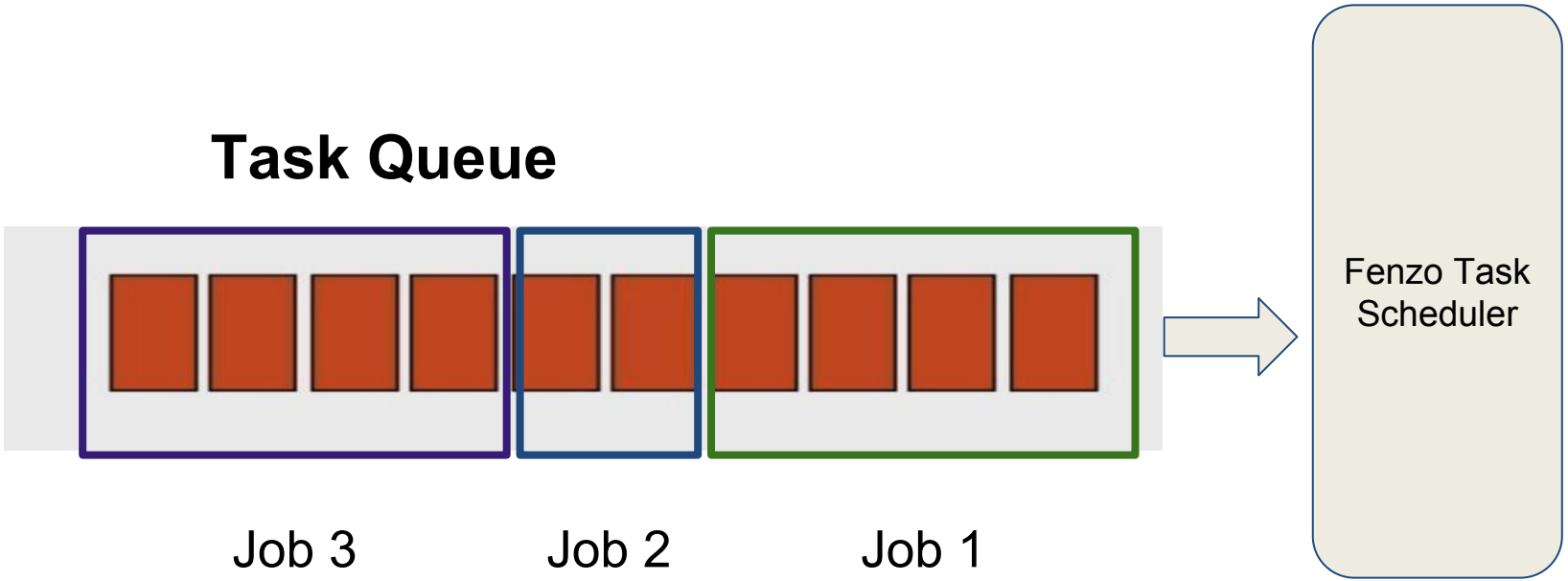
Greedy

Fenzo - Scheduling Library



https://github.com/Netflix/Fenzo/wiki

Order Tasks



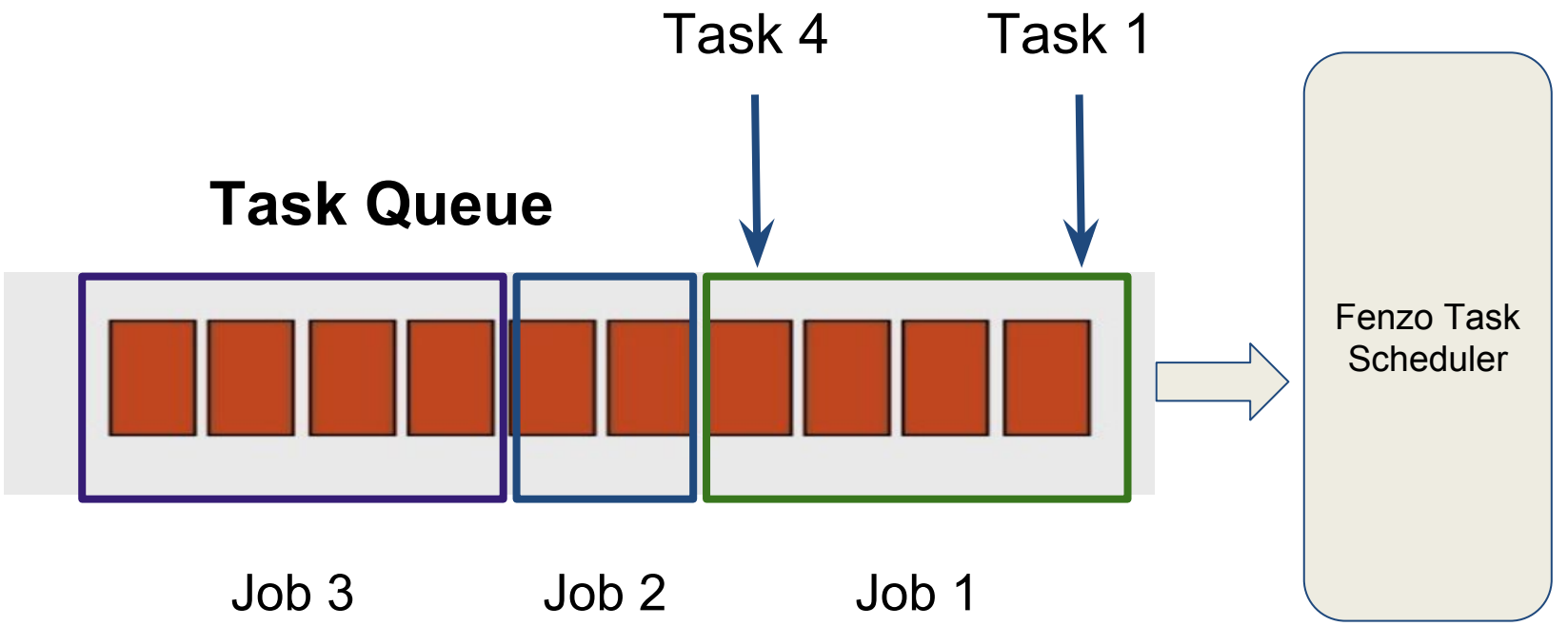
Task Queue

```
public interface TaskQueue extends TaskIterator {

    /**
     * Tasks in a queue are said to be in one of two states. The {@link #QUEUED} state repr
     * resource assignment. Where as, the {@link #LAUNCHED} state represents tasks that hav
     * such tasks may be either already executing or pending launch. This is used primarily
     * the tasks via the callback passed to {@link com.netflix.fenzo.TaskSchedulingService#
     */
    enum TaskState { QUEUED, LAUNCHED }

    /**
     * Add a task to the queue. Duplicates are not allowed, as in, a task request that has
     * existing element will be rejected. The added task will be assigned resources by a sc
     * into Fenzo that is already running from before, use
     * {@link com.netflix.fenzo.TaskSchedulingService#initializeRunningTask(QueuableTask, S
     * <P>
     * This operation is designed to be performed asynchronously, when it is safe to modify
     * implementations generally do not modify the queue while a scheduling iteration is in
     * @param task A task to add to the queue.
     */
    void queueTask(QueuableTask task);

    /**
     * Set SLA for the queue. The queue implementation determines the implementation of {@l
     * accepted.
     * @param sla The SLA to set for the queue.
     * @throws IllegalArgumentException if the implementation of the {@link TaskQueueSla} i
     * queue implementation.
     */
    void setSla(TaskQueueSla sla) throws IllegalArgumentException;
}
```

Task Queue

```
package com.netflix.fenzo;

import ...

public interface TaskIterator {
    /**
     * Get the next task from queue, or {@code null} if no more tasks exist.
     * @return The next task or a task with an assignment failure, if the task cannot be schedu
     *         internal constraints (for example exceeds allowed resource usage for a queue).
     *         Returns {@code null} if there are no tasks left to assign resources to.
     * @throws TaskQueueException if there were errors retrieving the next task from the queue.
     */
    Assignable<? extends TaskRequest> next() throws TaskQueueException;
}
```

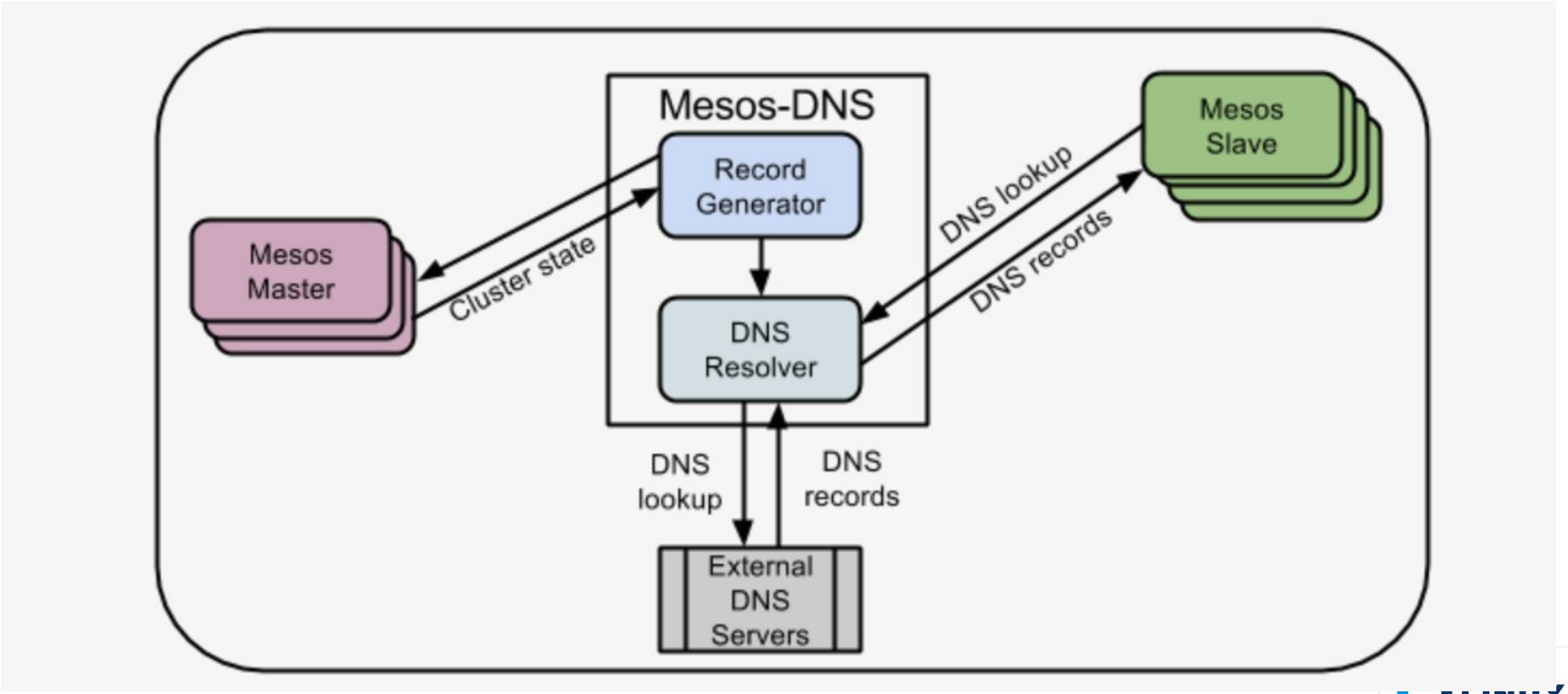
Order Tasks



How do you identify where the workers are running if tasks are launched one at a time?

This becomes a service discovery problem!!

Service Discovery - Mesos DNS



Discovering Other Tasks

- Poll DNS and wait until all tasks have been discovered. This is done inside the Mesos executor.
- Runs on every worker and ps before starting the actual training job
- Once discovered set environment variables to bootstrap the task

Example: TF_CONFIG

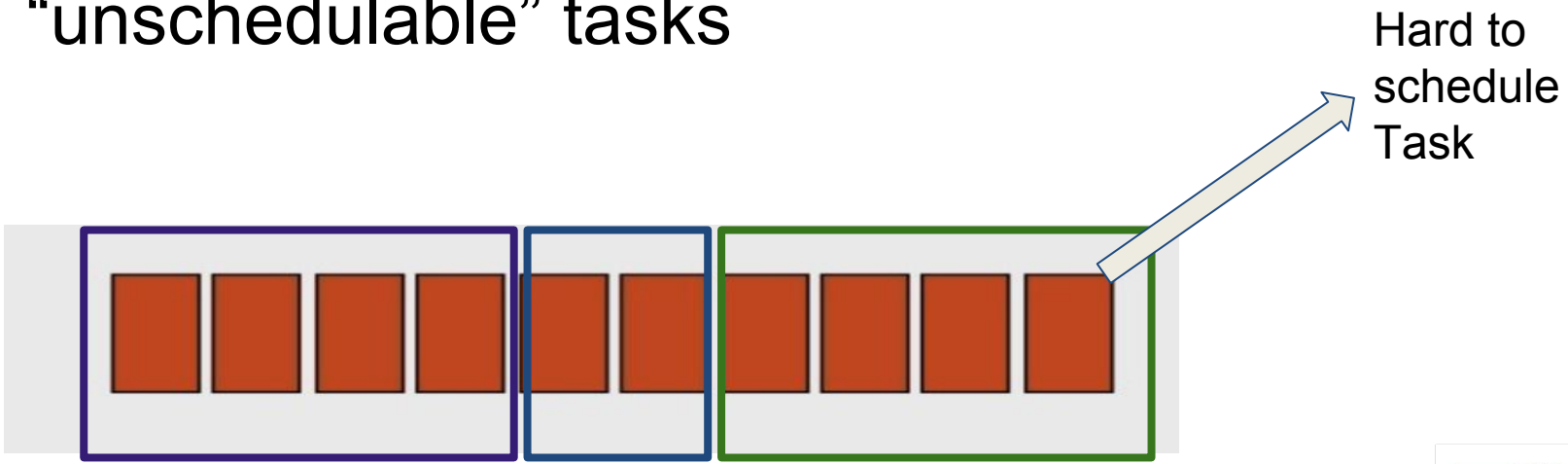
```
{
   "cluster":{
      "ps":[
         "host1:2222",
         "host2:2222"
      ],
      "worker":[
         "host3:2222",
         "host4:2222",
         "host5:2222"
      ]
   },
   "task":{
      "type":"worker",
      "index":1
   }
}
```

Example: LIGHTGBM

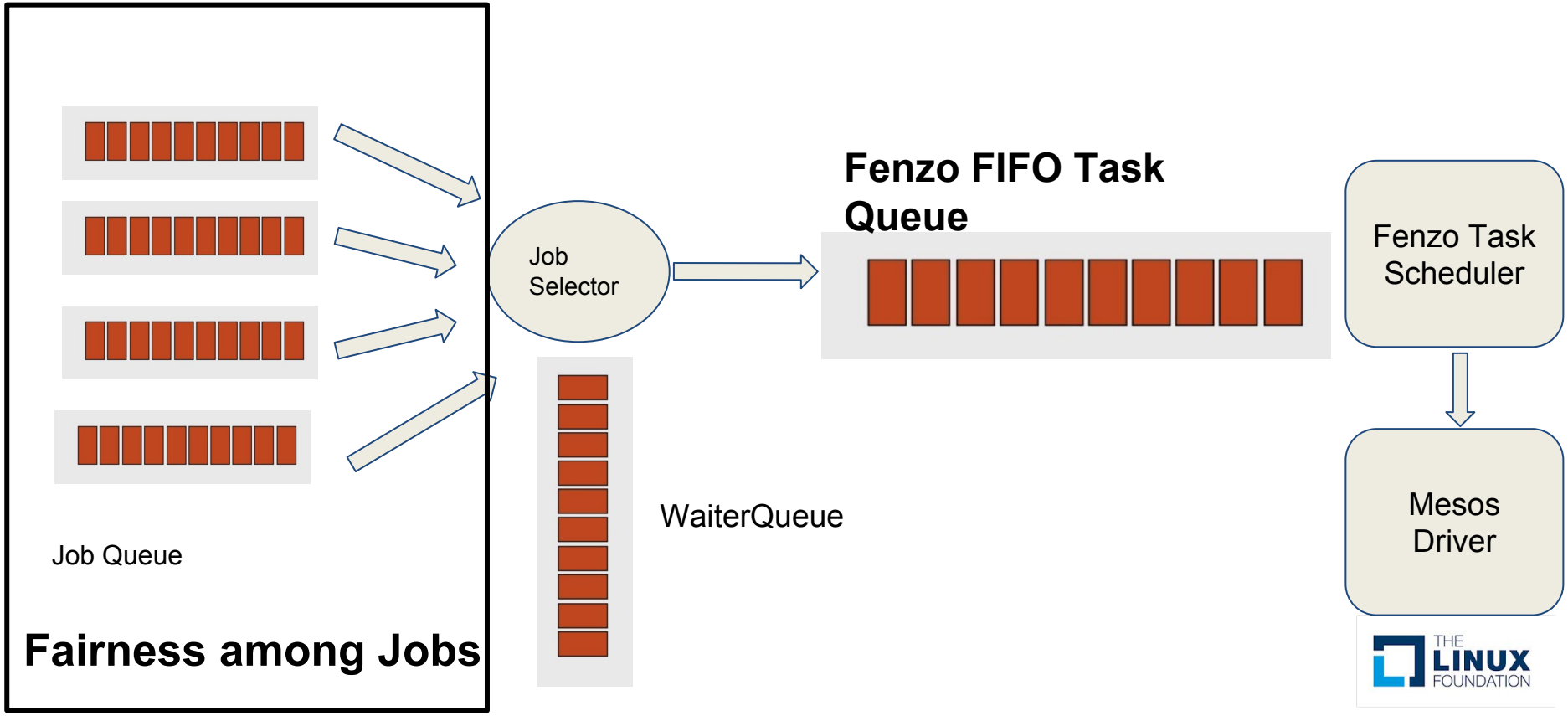
```
export PORT="31551"
export NUM_MACHINES="3"
export
MACHINES="x.x.x.x:31771,x.y.z.a:31811,x.y.z.a
:31551"
export RANK=0
```

Issue With a FIFO Queue

- Ordering Tasks is a problem as well can result in tasks being blocked by “unschedulable” tasks



WaiterQueue



Estimate Job Schedulability

- Whether a job with tasks having resource vector [cpus, mem, disk, gpu] can be scheduled
- Mesos Operator API

Mesos Operator API

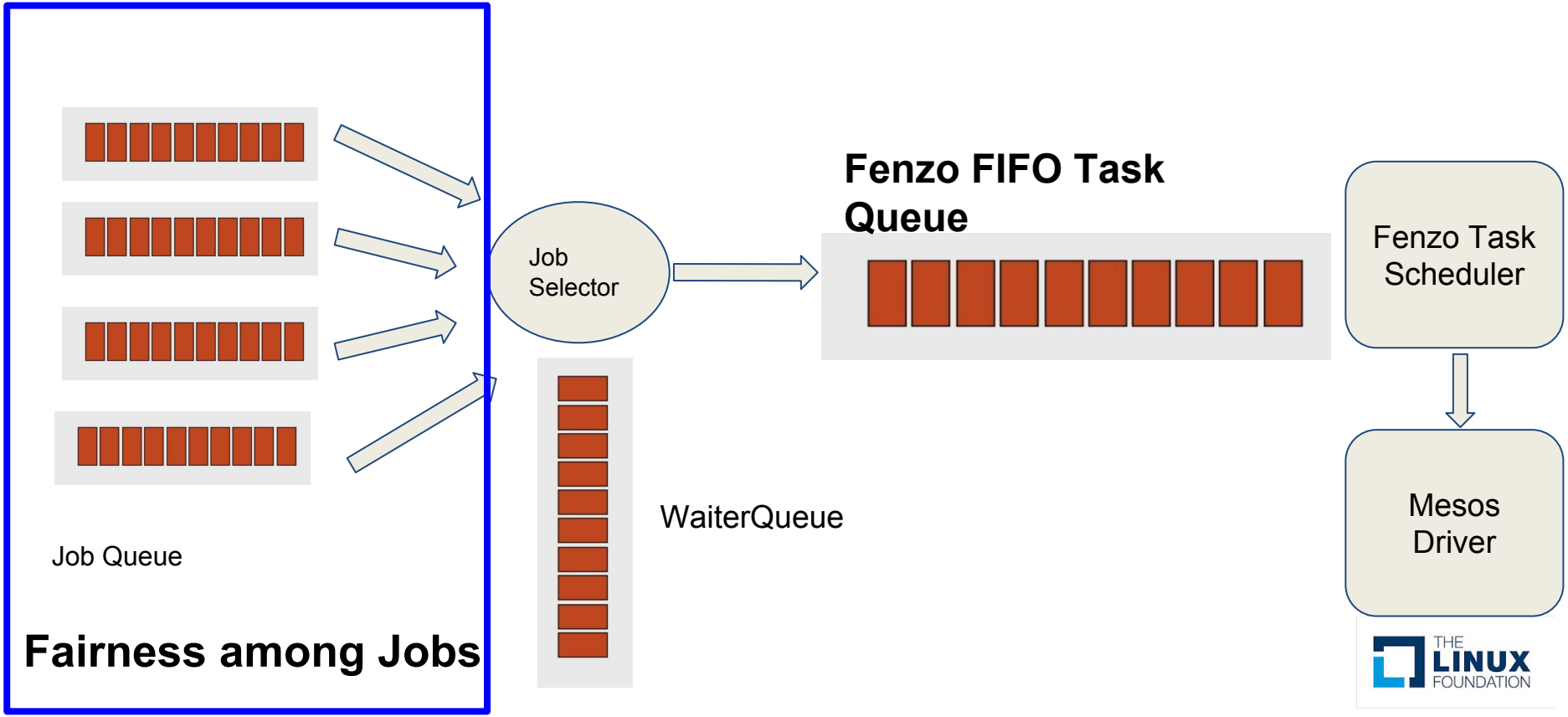
TASK_UPDATED Event (JSON)

```
<event-length>
{
  "type": "TASK_UPDATED",

  "task_updated": {
    "task_id": {
        "value": "42154f1b-adcd-4421-bf13-8bd11adfafaf"
    },

    "framework_id": {
        "value": "49154f1b-8cf6-4421-bf13-8bd11dccd1f1"
    },

    "agent_id": {
        "value": "2915adf-8aff-4421-bf13-afdafaf1f1"
    },

    "executor_id": {
        "value": "adfaf-adff-2421-bf13-adf23tafa21"
    },

    "state" : "TASK_RUNNING"
  }
}
```

Mesos Operator API

- Listen to TaskUpdate events
  - Construct rough picture of the cluster
  - Map[Host, Seq[Slot]]
  - Slot - [cpus, mem, disk, gpu]

Job Fairness



Order of Scheduling Jobs

- Have multiple queues
  - One per team/role
  - Compute DRF (Dominant Resource Fairness) score for every queue
  - Sort queues and go in order of DRF score
- Within a queue
  - Order jobs by FIFO per role
  - Give preference to prod jobs

How Dominant Resource Fairness Works

[8, 50, 2]

Q1

CPU - 2/5
Mem - 1/2
GPU- 1/5

[5, 20, 2]

Q2

CPU - 1/4
Mem - 1/5
GPU- 1/5

[2, 30, 5]

Q3

CPU - 1/10
Mem - 3/10
GPU- 1/2

Total Cluster Resources

CPU - 20
Mem - 100
GPU - 10

How Dominant Resource Fairness Works

[8, 50, 2]

Q1

CPU - 2/5
Mem - 1/2
GPU- 1/5

[5, 20, 2]

Q2

CPU - 1/4
Mem - 1/5
GPU- 1/5

[2, 30, 5]

Q3

CPU - 1/10
Mem - 3/10
GPU- 1/2

Total Cluster Resources

CPU - 20
Mem - 100
GPU - 10

Q2 << Q1 == Q3

Implementing Packing and Spread

- Fitness Function
  - Score between 0 and 1 indicating how well a task fits on a particular host
    - 1 is a perfect fit
    - 0 is an abysmal fit
  - These are only a preference and not strongly enforced

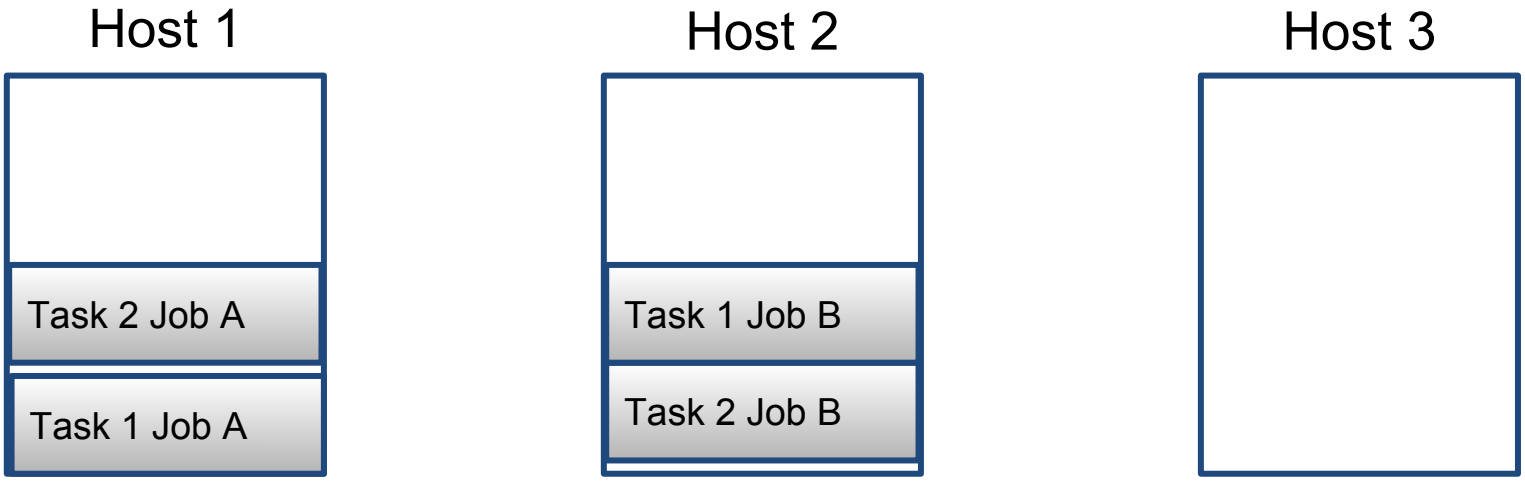
Fenzo Fitness Functions

```java
/**
 * Interface representing a task fitness calculator, or scheduling optimization plugin. A task may fit on
 * multiple hosts. Use a fitness calculator to determine how well a task fits on a particular host given the
 * current state of task assignments and running tasks throughout the system.
 */
public interface VMTaskFitnessCalculator {
    /**
     * Get the name of this fitness calculator.
     *
     * @return Name of the fitness calculator.
     */
    public String getName();

    /**
     * Calculates how well the task fits on the host. This method does not have to check to see that the
     * proposed host has sufficient resources for the proposed task. It can assume that this has already been
     * done.
     *
     * @param taskRequest      the task whose resource requirements can be met by the Virtual Machine
     * @param targetVM         the prospective target host (VM) for given {@code taskRequest}
     * @param taskTrackerState state of the task tracker that contains all tasks currently running and assigned
     * @return a value between 0.0 and 1.0, with higher values representing better fit of the task on the host
     */
    public double calculateFitness(TaskRequest taskRequest, VirtualMachineCurrentState targetVM,
                                   TaskTrackerState taskTrackerState);
}
```

Packing Fitness Example

Task 3 , Job A (Prod)

Job B = Prod

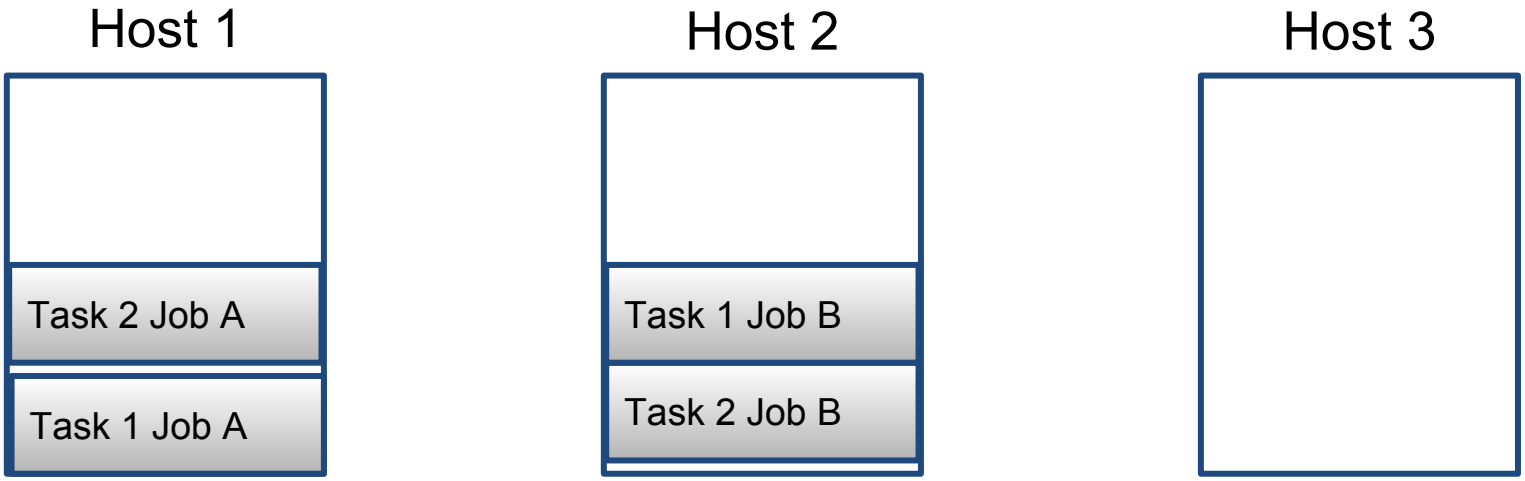


F(Host1, T3JA) > F(Host3, T3JA) > F(Host2, T3JA)

Packing Fitness Example

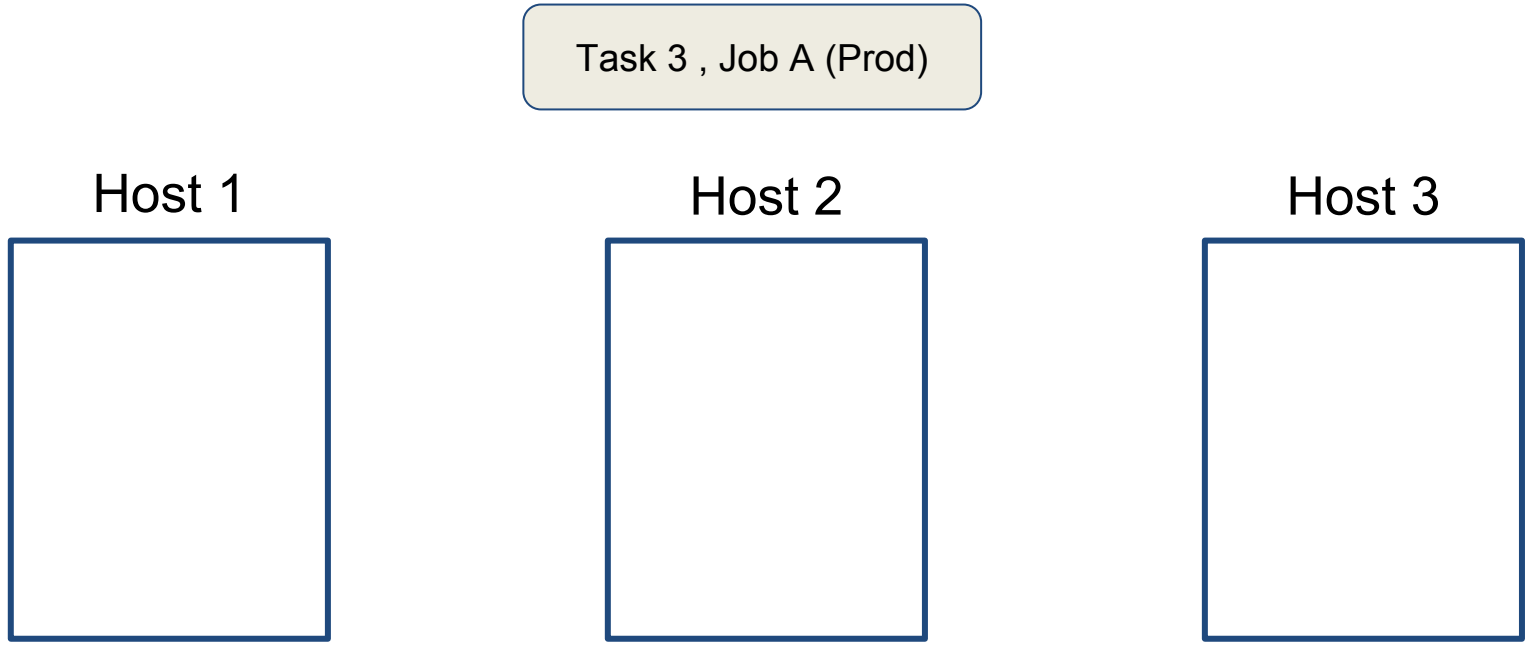
Task 3 , Job A
(Non-Prod)

Job B = Non-Prod

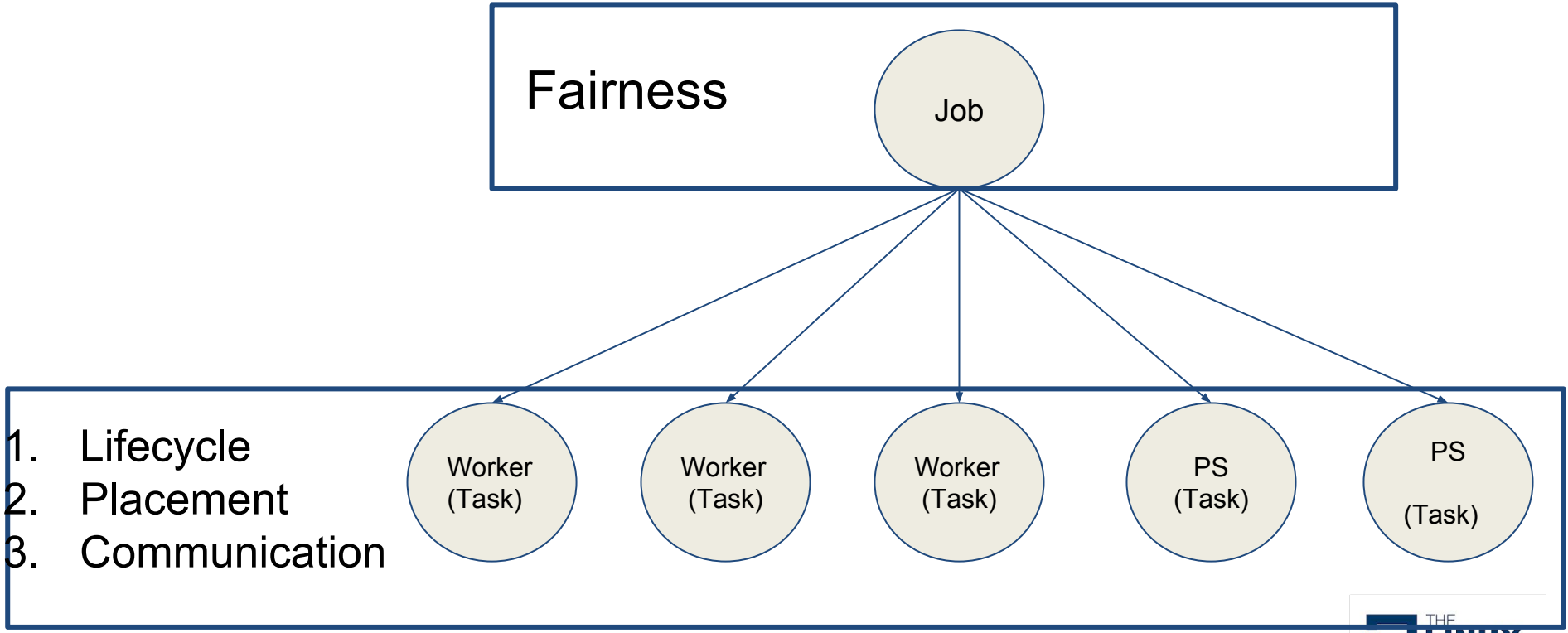


F(Host1, T3JA) > F(Host2, T3JA) > F(Host3, T3JA)

Fitness Example



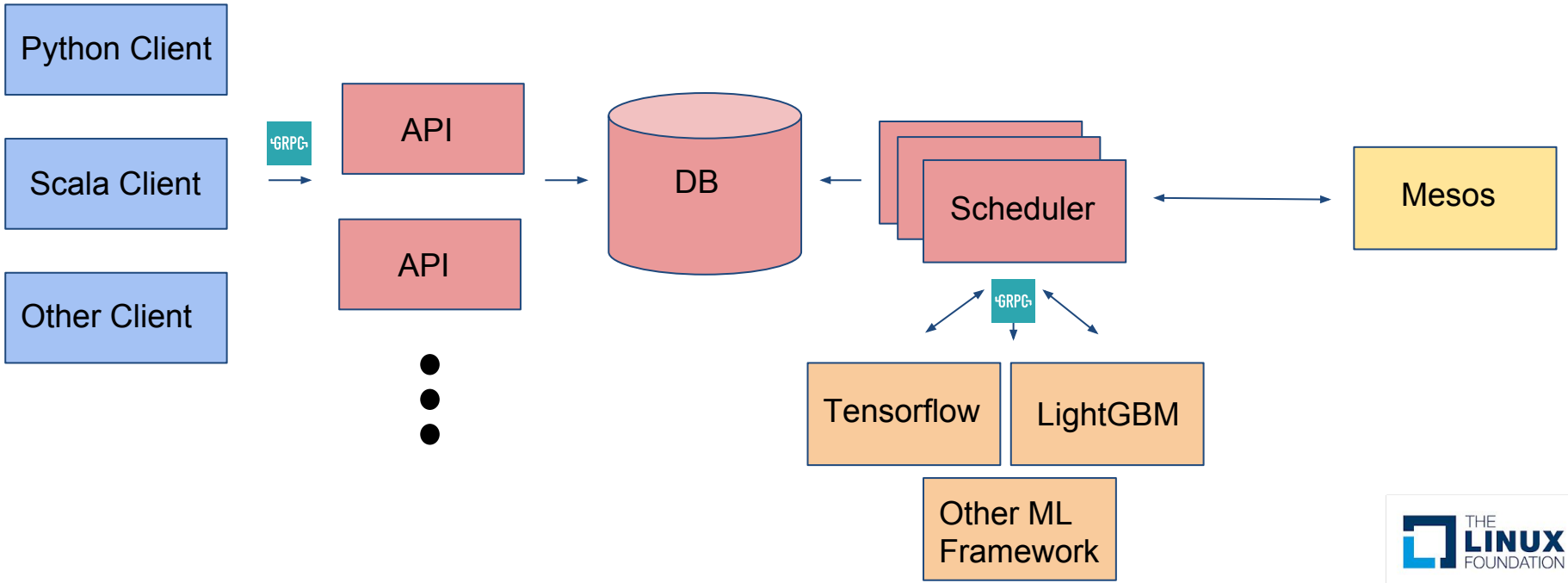
F(Host1, T3JA) = F(Host3, T3JA) = F(Host2, T3JA)


Fairness
Job
1. Lifecycle
2. Placement
3. Communication
Worker (Task)
Worker (Task)
Worker (Task)
PS (Task)
PS (Task)

Agenda

- Distributed Training
- Mesos and CASE
  - Scheduling Features
    - Implementation in CASE
  - **Architecture**
- Questions

Architecture



Job

- Role
- ID
- Labels
- Environment (Dev/Prod)
- Config

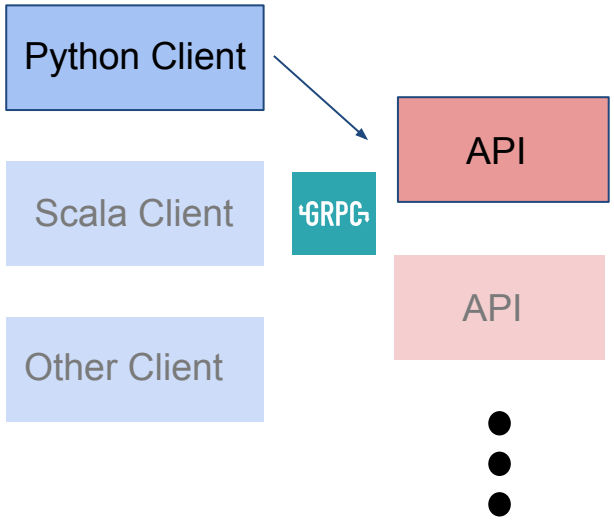
Architecture - Clients

Python Client

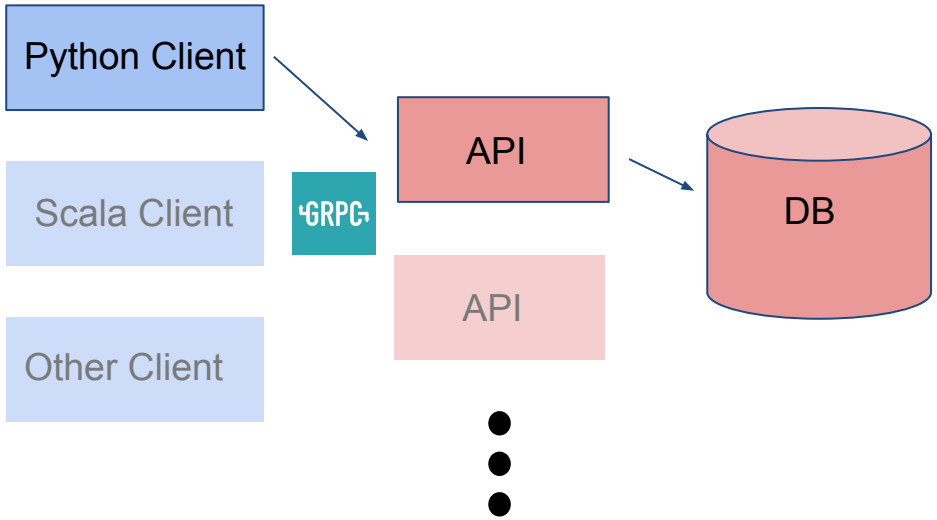
Scala Client

Other Client

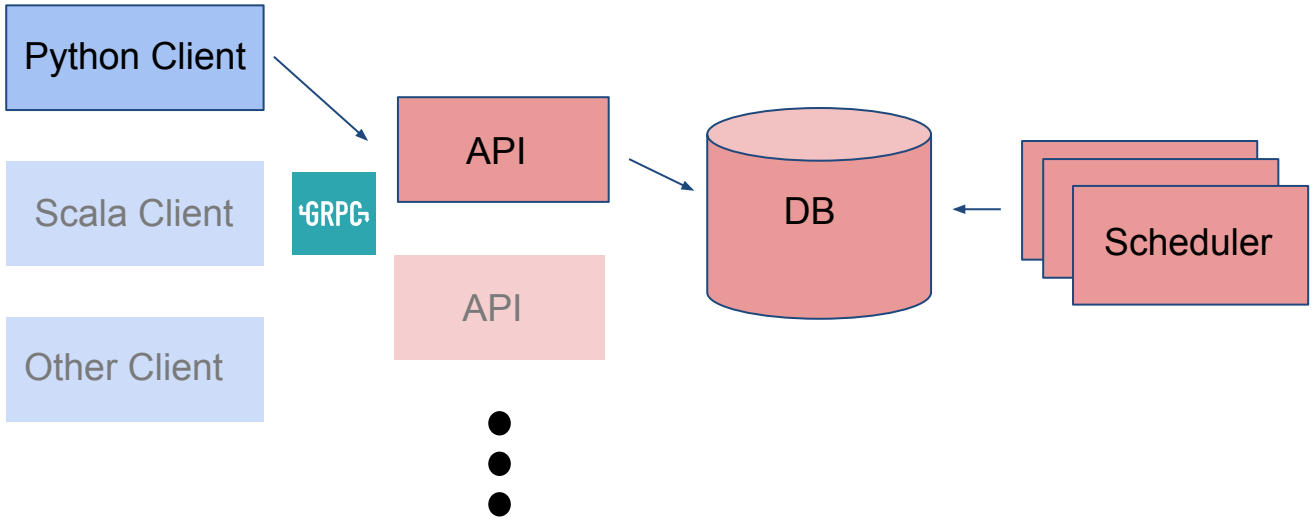
Architecture - API



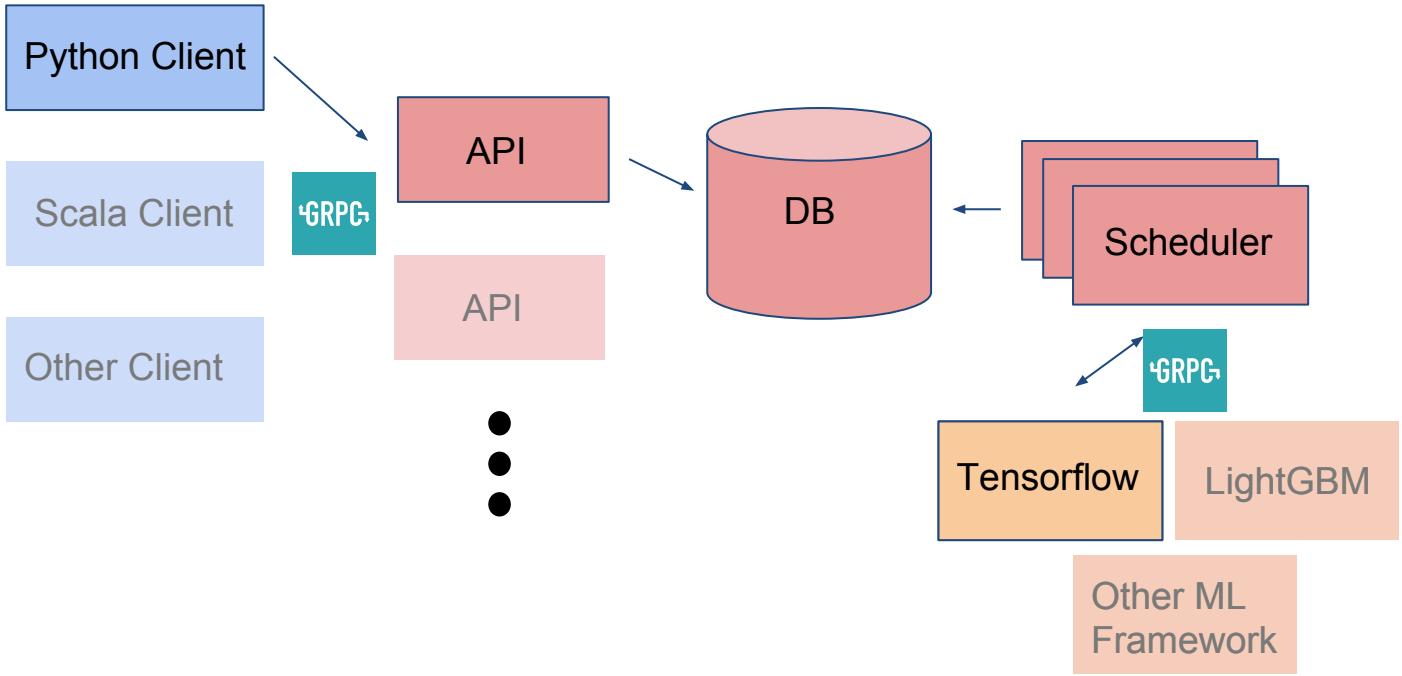
Architecture - Database



Architecture - Scheduler



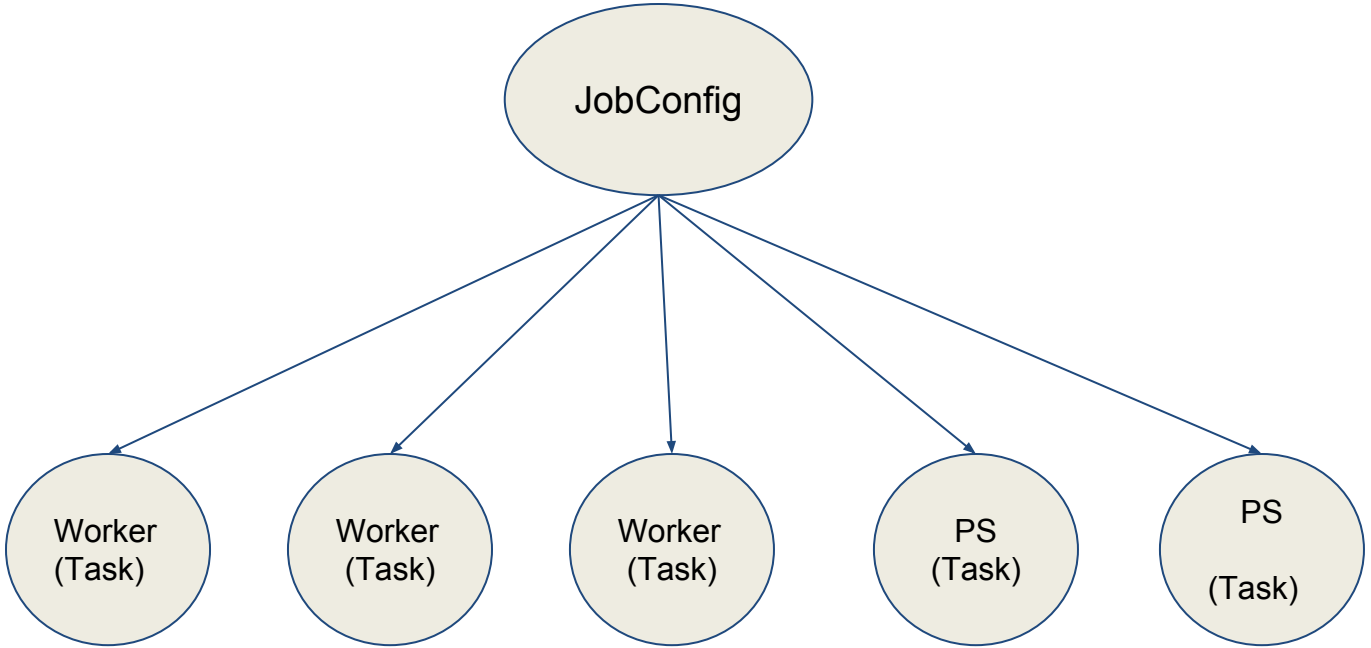
Architecture - Framework Operators



Framework Operator API

- F: Config -> Seq[TaskInfo]
- F: Seq[TaskState] -> JobState

State Management of Tasks



Job Config

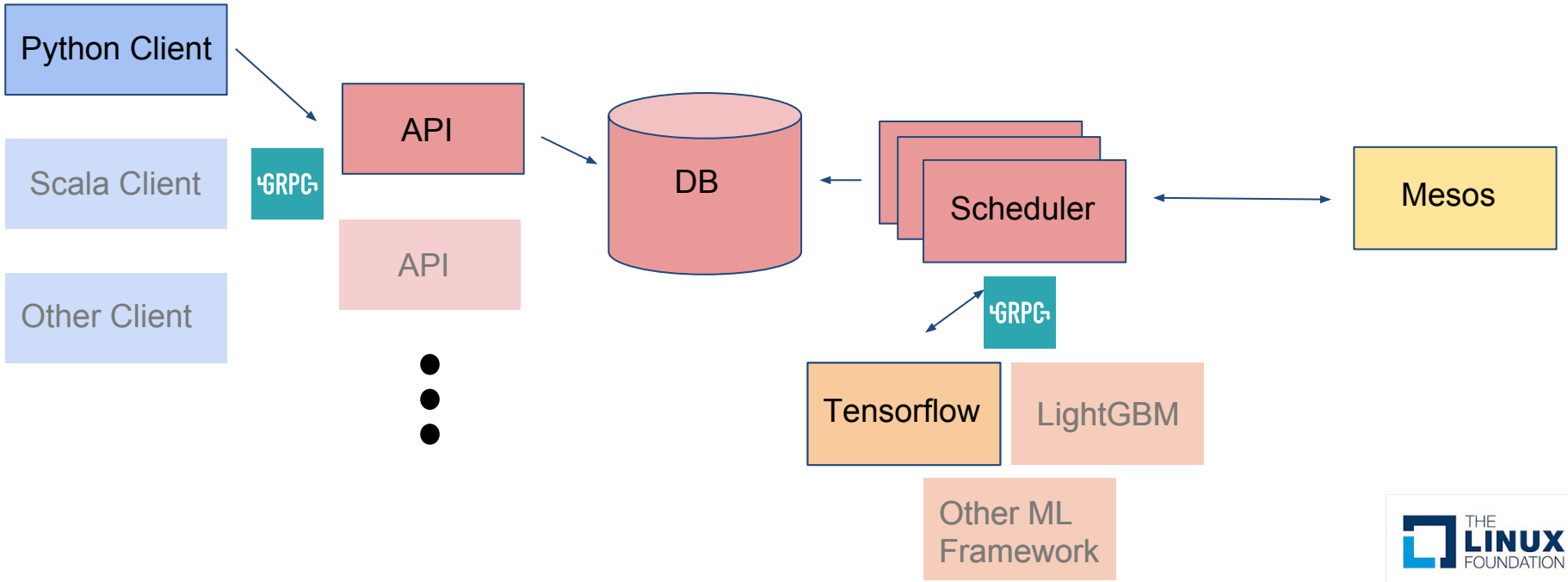
Tensorflow Config

- Command for PS/worker
- Docker image
- Tensorboard log directory
- Tensorflow version
- Plus all of these can be specified for the workers or the PS or both

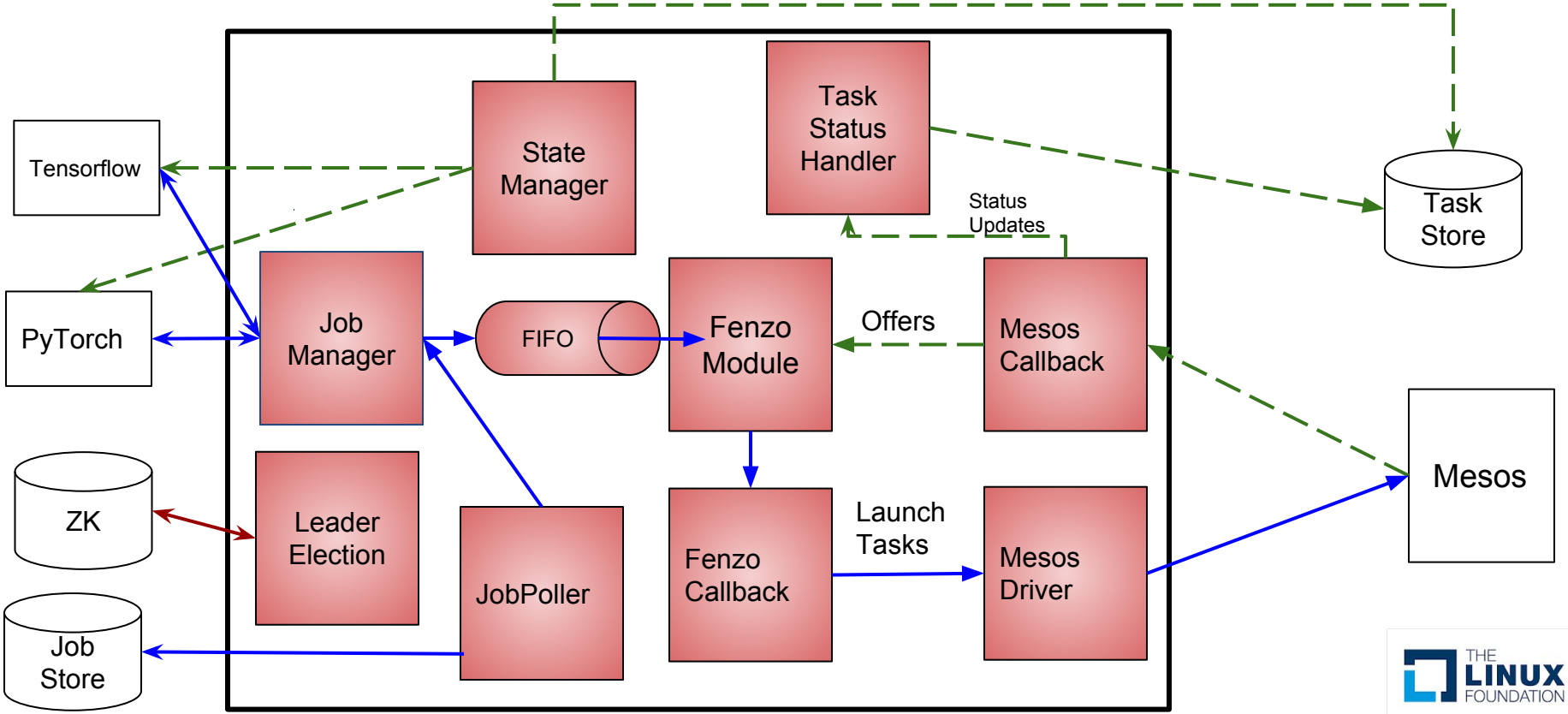
LightGBM Config

- Command
- Docker image
- Number of machines (workers)

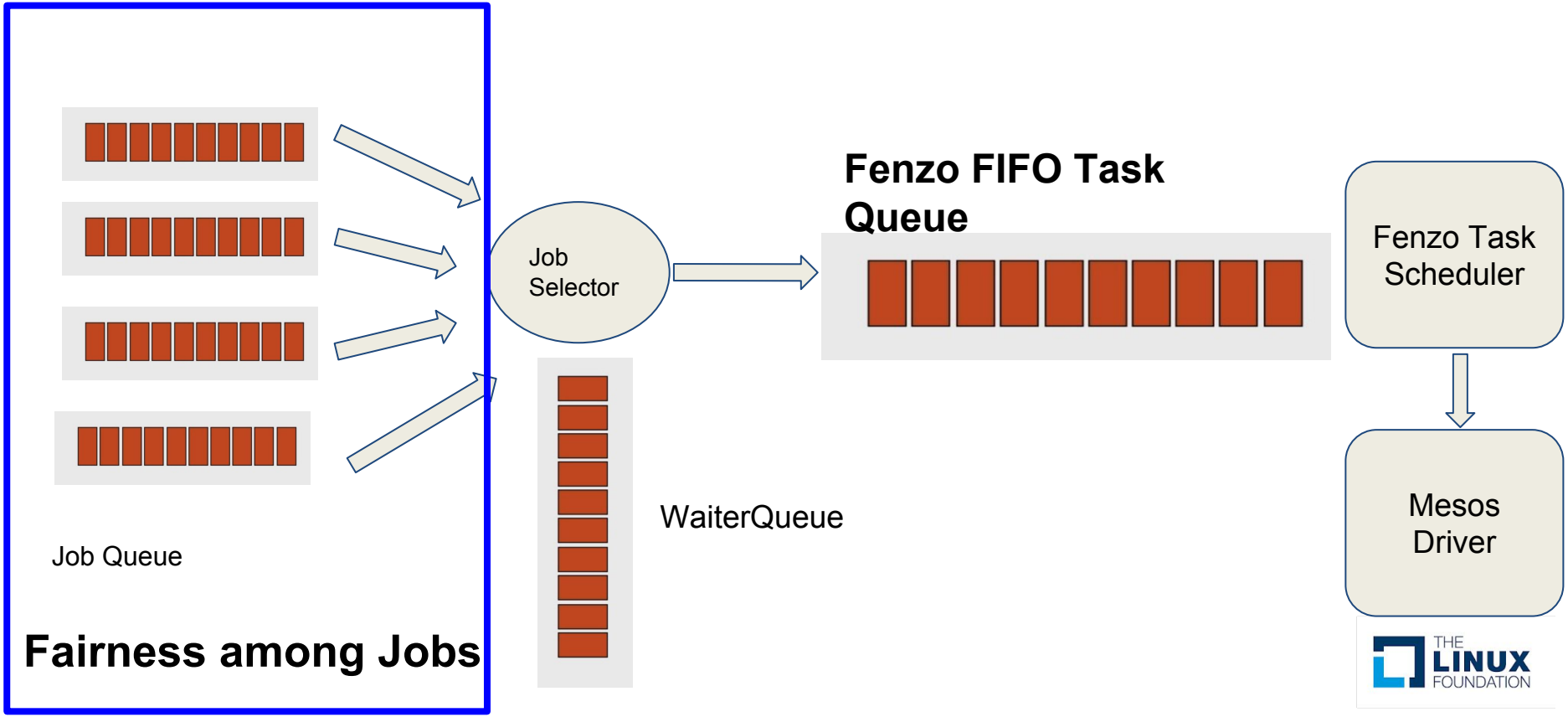
Architecture - Mesos



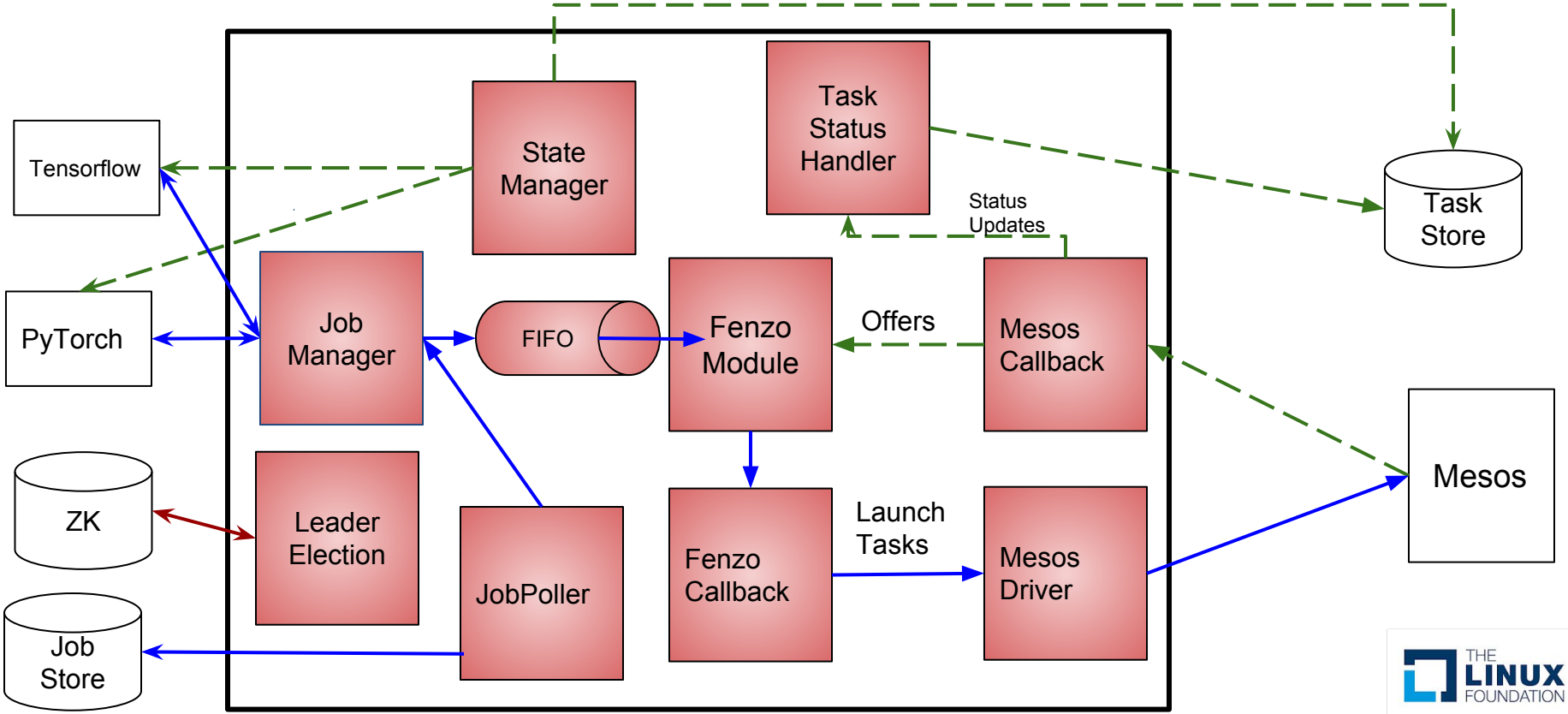
Architecture - Scheduler



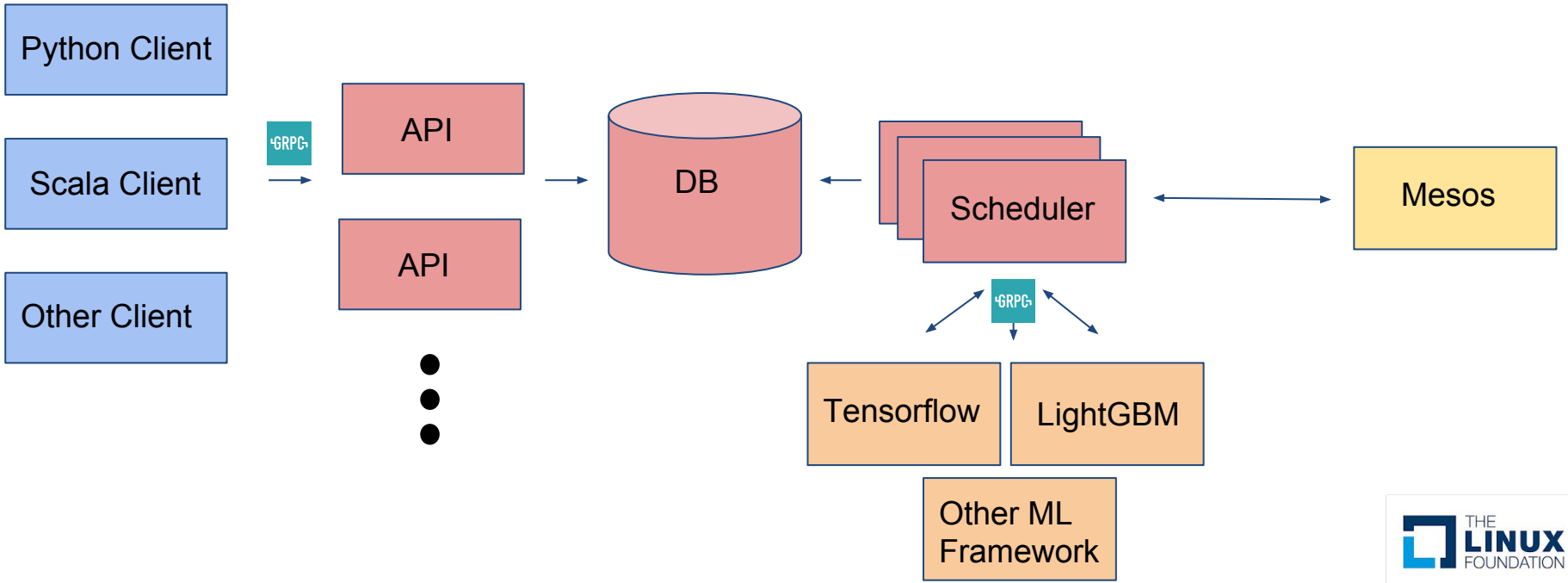
Job Fairness



Architecture - Scheduler



Architecture



Frameworks Supported

- Tensorflow
- LightGBM
- PyTorch
- Horovod

More To Come

- Support for other frameworks
- Preemption
- SLA based scheduling
- Job fairness

Questions ?

THE LINUX FOUNDATION

THE LINUX FOUNDATION

OPEN SOURCE SUMMIT

THE LINUX FOUNDATION