MesosCon
NORTH AMERICA

# Idle Resources Oversubscription:

## improving cluster utilization

**Dmitry Zhuk, Sr. Software Engineer, *Twitter***

# Cluster workload

- Cluster size: >30K nodes
- Production Jobs
  - Allocated CPUs: ~80%
  - p95 cluster CPU utilization: >30%
- Non-production Jobs
  - Allocated CPUs: ~10%
  - p95 cluster CPU utilization: ~20%
  - Best effort
  - Can be preempted by production jobs

# Problem

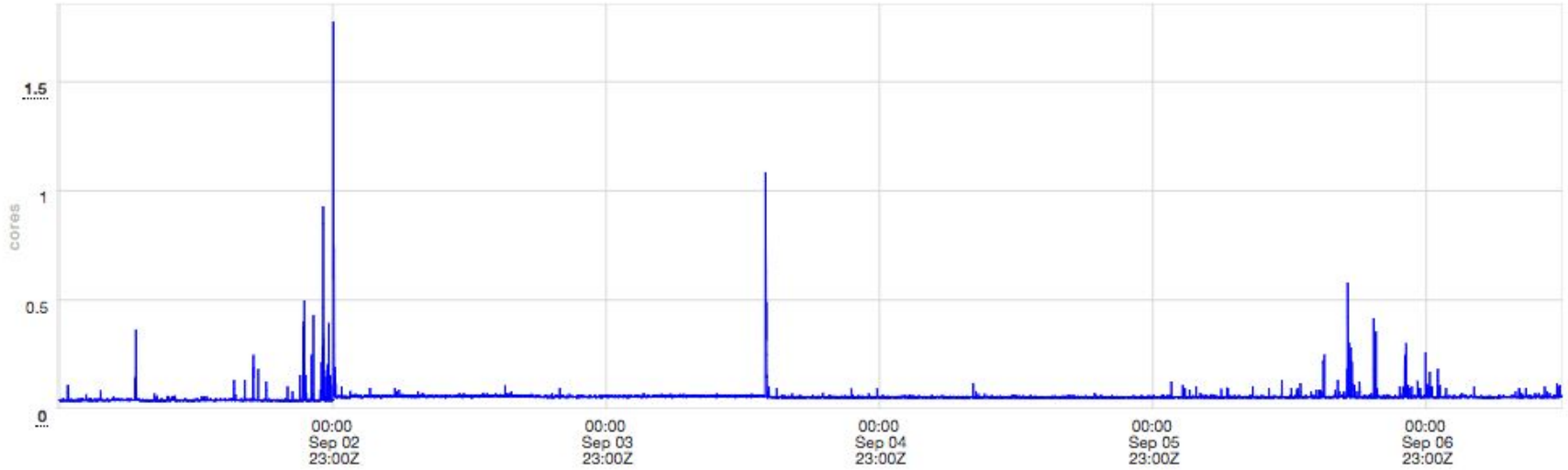
- Most limited cluster resource is CPU
- Non-production resource allocation is growing
- Many non-production jobs are idle, i.e. CPU utilization is significantly below allocation

# Non-production CPU utilization

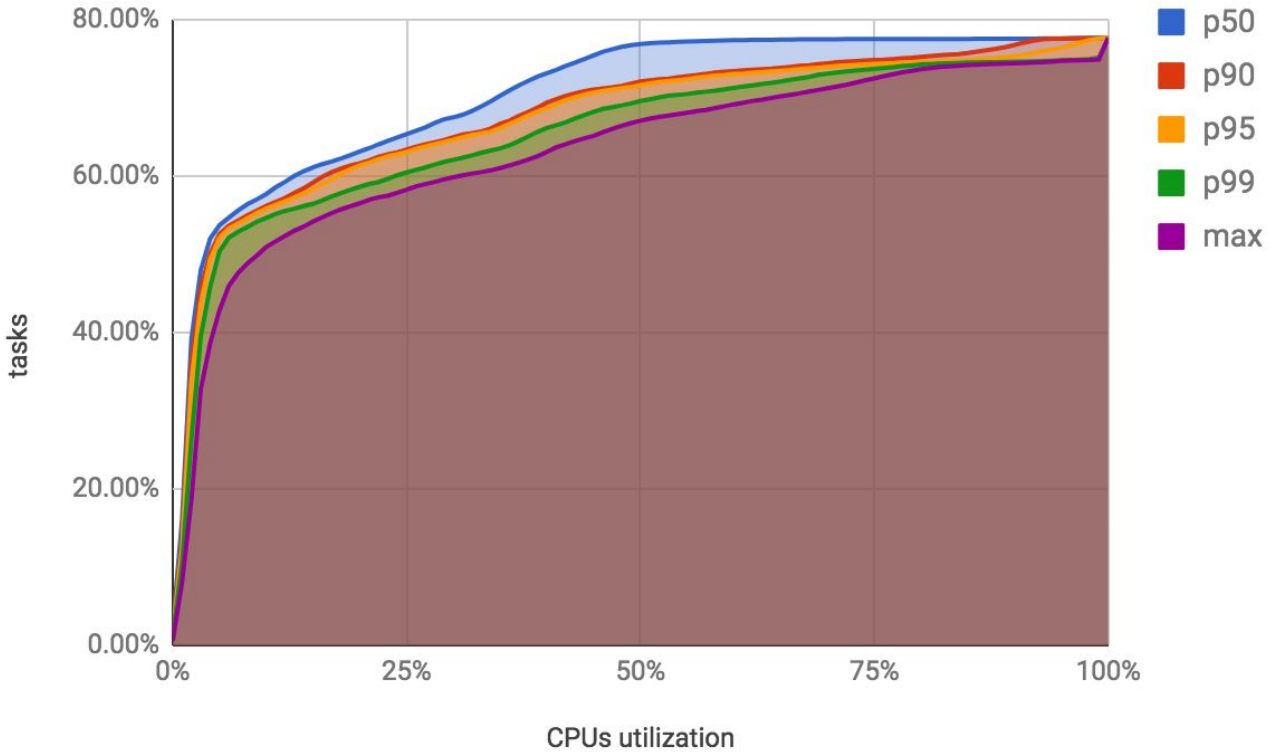
- Examined non-production jobs CPU utilization:
  - most jobs are idle, some with occasional bursts in CPU utilization
- Collected CPU utilization metrics for non-production jobs:
  - includes only long-running jobs (~80% of CPUs allocated for non-production jobs)

# Idle task example

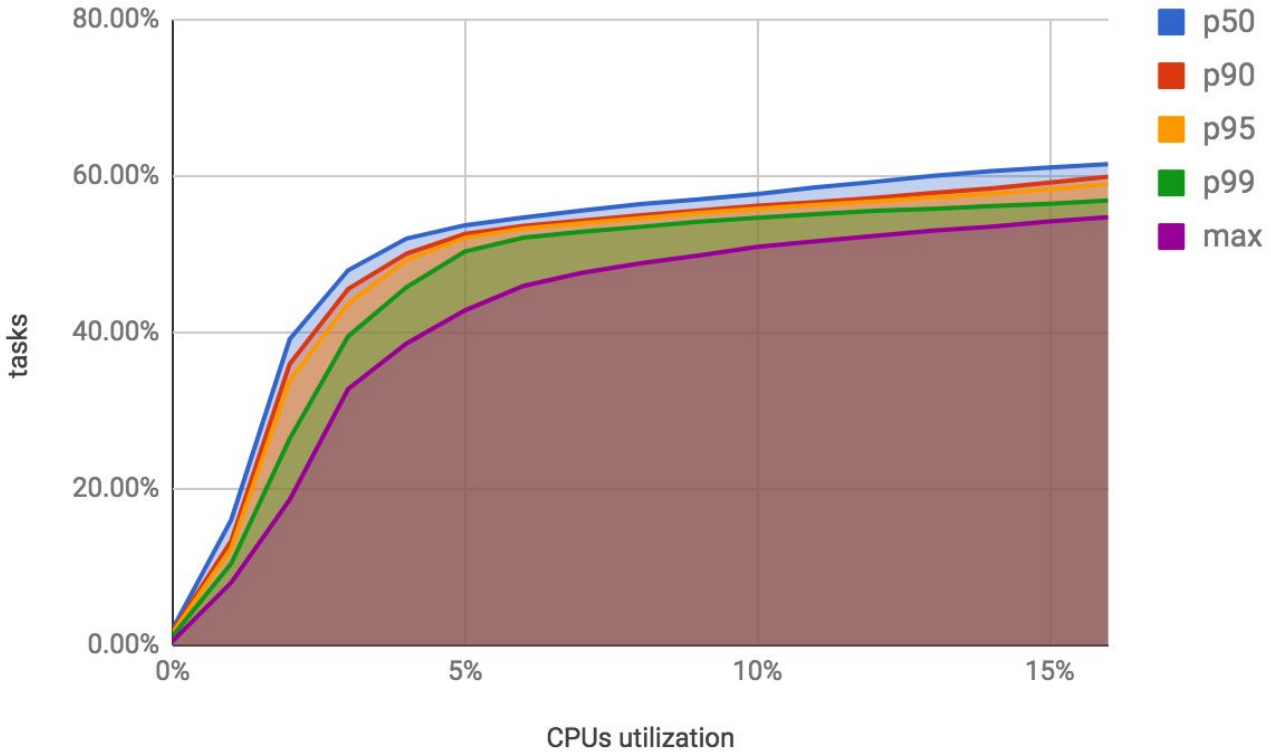
Allocation: 4 CPUs




# Non-production CPU utilization



# Non-production CPU utilization



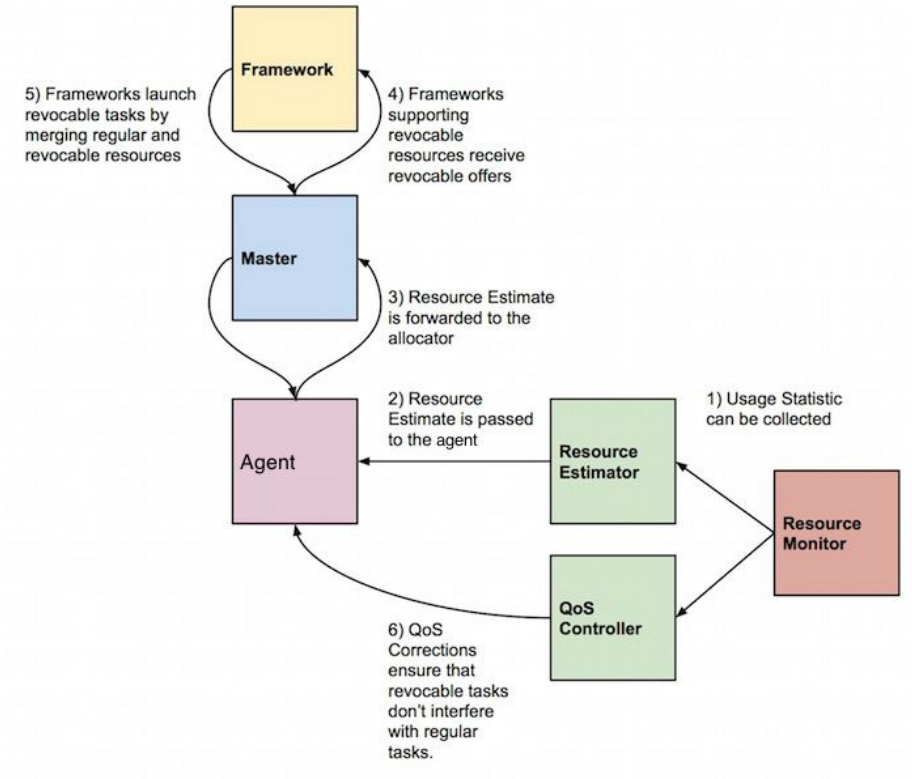
# Solution

- Detect under-utilized CPU resources allocated to non-production jobs
- Offer these resources for oversubscription and use them to launch other non-production jobs
- Production jobs use non-revocable resources
- Non-production jobs use revocable resources, however they may also use non-revocable resources, but can be preempted by scheduler

# Constraints

- Non-production jobs must not affect production jobs
- No preemption of production jobs
- Non-production jobs can be preempted
- Some impact on performance of non-production jobs is expected at timescales relevant to time required to detect possible contention

# Oversubscription in Mesos



# Resource Estimator

```
class ResourceEstimator {
  Try<Nothing> initialize(
    const function<
      Future<ResourceUsage>()>& usage);

  Future<Resources> oversubscribable();
};
```

# QoS Controller

```
class QoSController {
  Try<Nothing> initialize(
    const function<
      Future<ResourceUsage>()>& usage);

  Future<list<QoSCorrection>> corrections();
};
```

# Oversubscription in Mesos

- Framework should opt-in to receive revocable resources with `Capability::REVOCABLE_RESOURCES`

- Enable `ResourceEstimator` and `QoSController` modules on agents:

```
--resource_estimator=...
--oversubscribed_resources_interval=...
--qos_controller=...
--qos_correction_interval_min=...
```

# Idle resources oversubscription

- Resource Estimation
  - Detect unutilized CPUs by non-production jobs:

    allocation slack + under-utilized allocated resources
- QoS Control
  - Kill non-production jobs, if there are not enough available resources
- Isolation
  - Isolate all non-production jobs in a separate cgroup.

# Idleness detection

Container is idle if ***P%*** of samples over window with duration ***D*** show CPU usage less than ***threshold***.

# Estimated savings

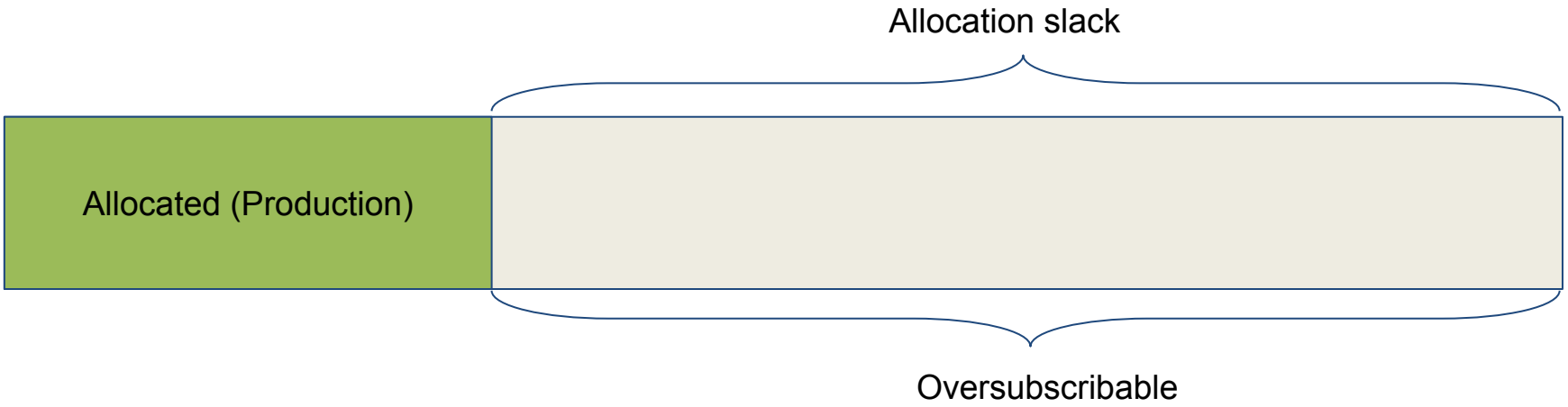
window=1day, threshold=10%

| Metric | Idle tasks | Idle CPUs* |
|---|---|---|
| p90 | 56% | 36 - 48% |
| p95 | 56% | 36 - 47% |
| p99 | 54% | 35 - 46% |
| max | 51% | 30 - 41% |

* lower bound: sum(int(cpus)), upper bound: sum(cpus)

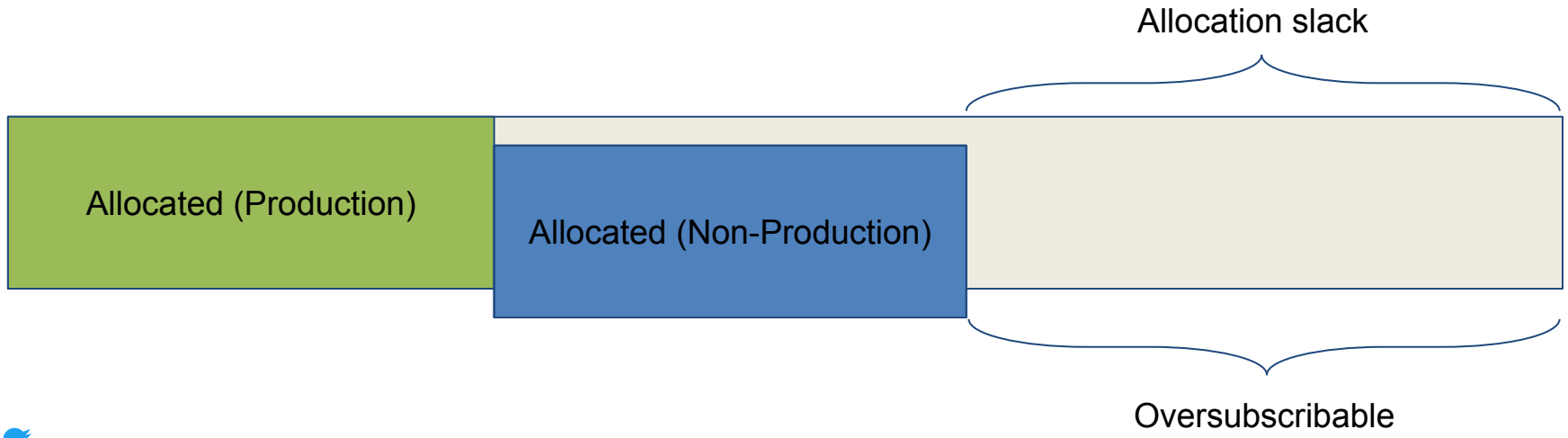
# Oversubscribable resources

Allocation slack is offered as oversubscribable



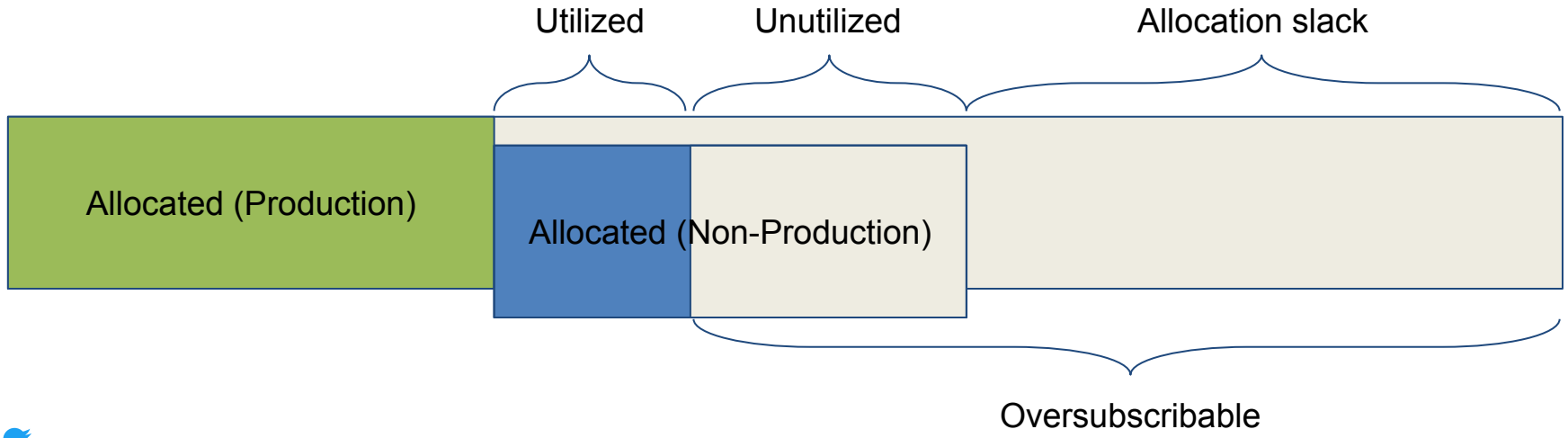
# Oversubscribable resources

Allocation slack is offered as oversubscribable



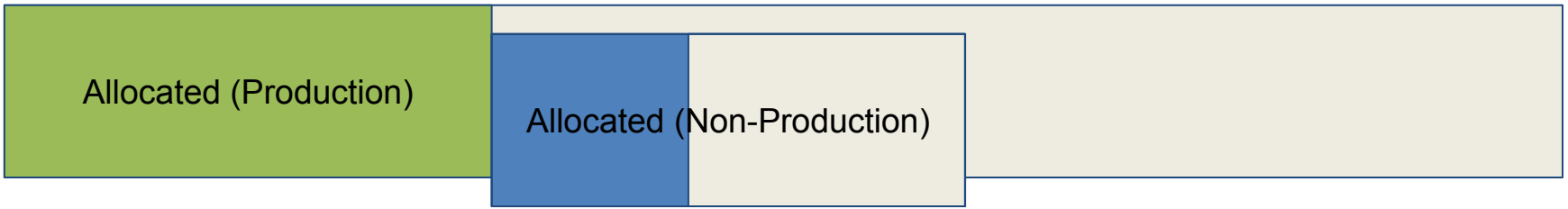
# Oversubscribable resources

Unutilized allocated resources are offered as oversubscribable



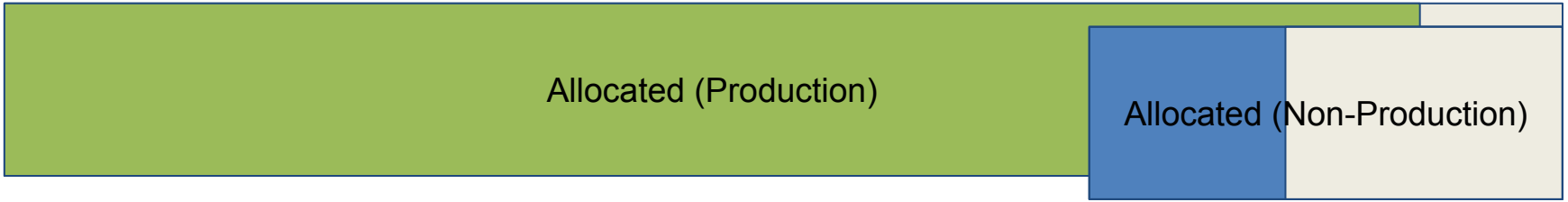
# QoS correction

Launching Production job: Allocation slack is reduced



# QoS correction

Launching Production job: Allocation slack is reduced




# QoS correction

Non-Production job becomes non-idle:
Unutilized is reduced

# QoS correction

Non-Production job becomes non-idle:
Unutilized is reduced


# QoS correction strategy

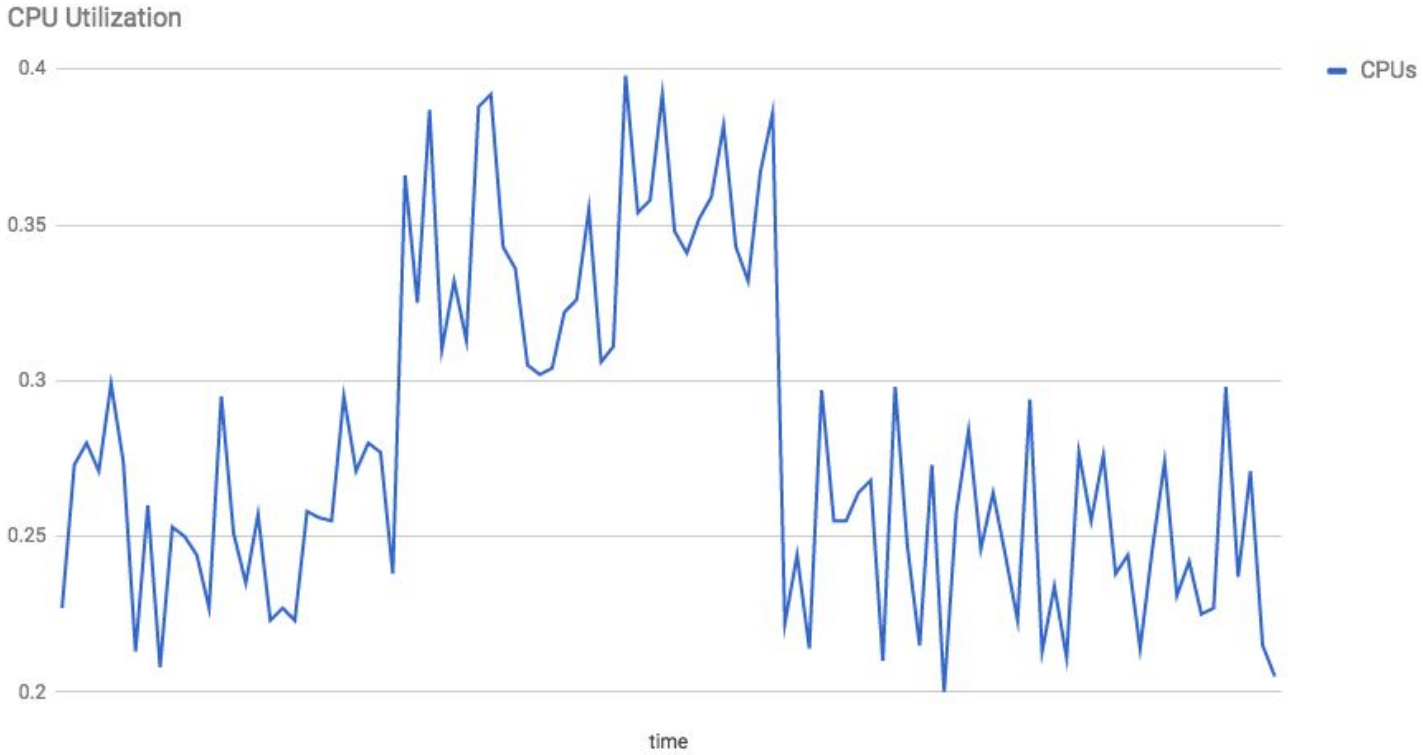
- Kill non-idle tasks, preserve idleness
  - Killing idle task will make scheduler relaunch it elsewhere in non-idle state

# Recover: idleness preservation

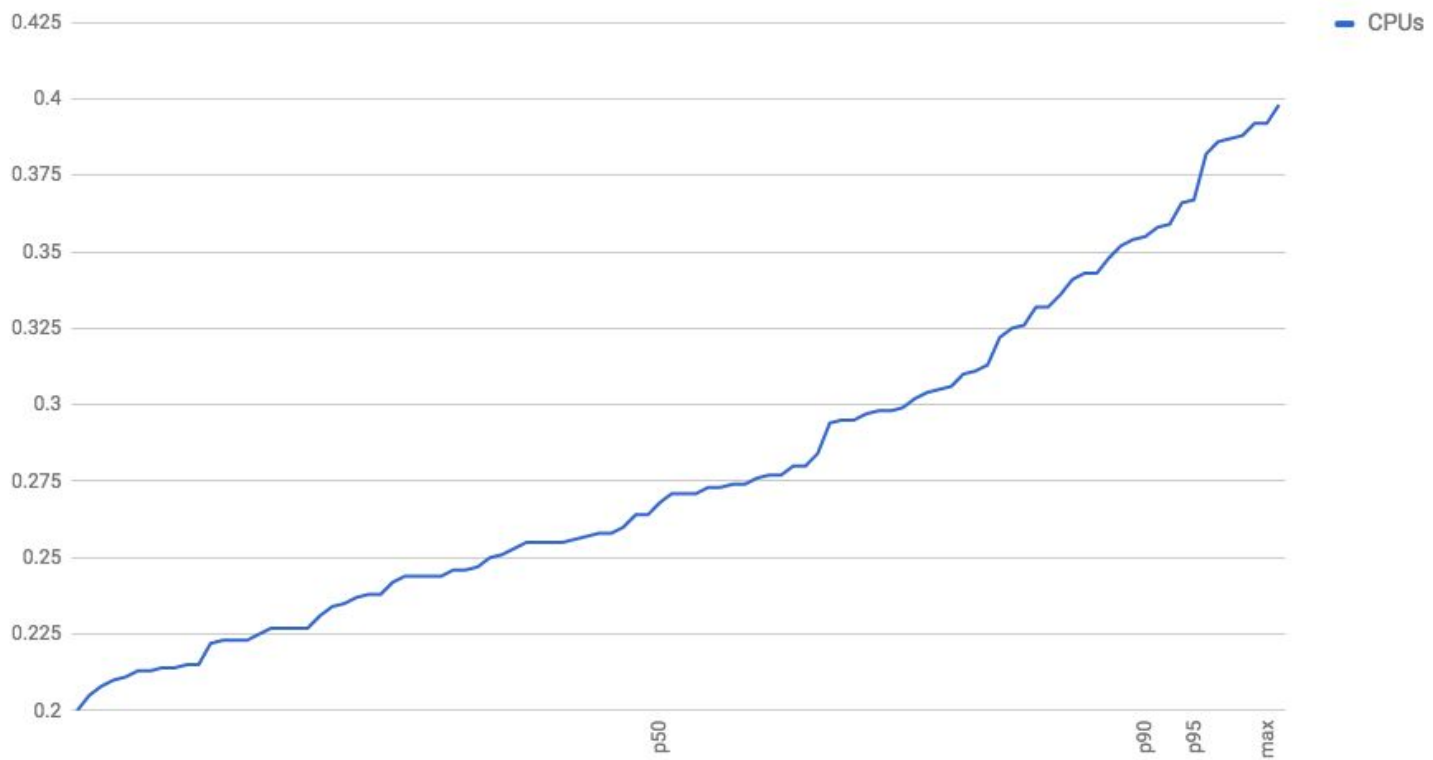
Preserve idleness between agent restarts:

- Checkpoint on each sample:
  - `Statistics` (p50, p90, p95, etc)
  - timestamps of first and last sample in the window
- Rebuild samples from statistics on recover

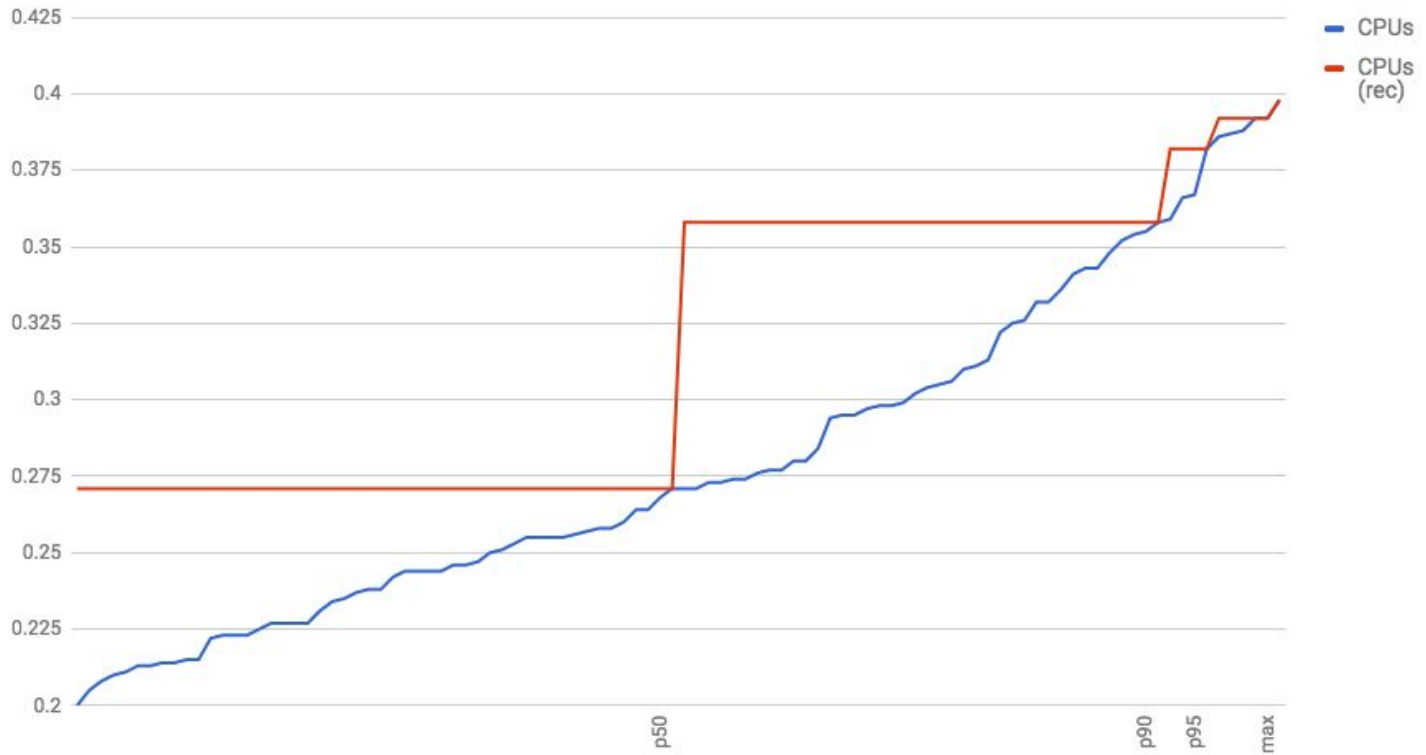
# Reconstruction



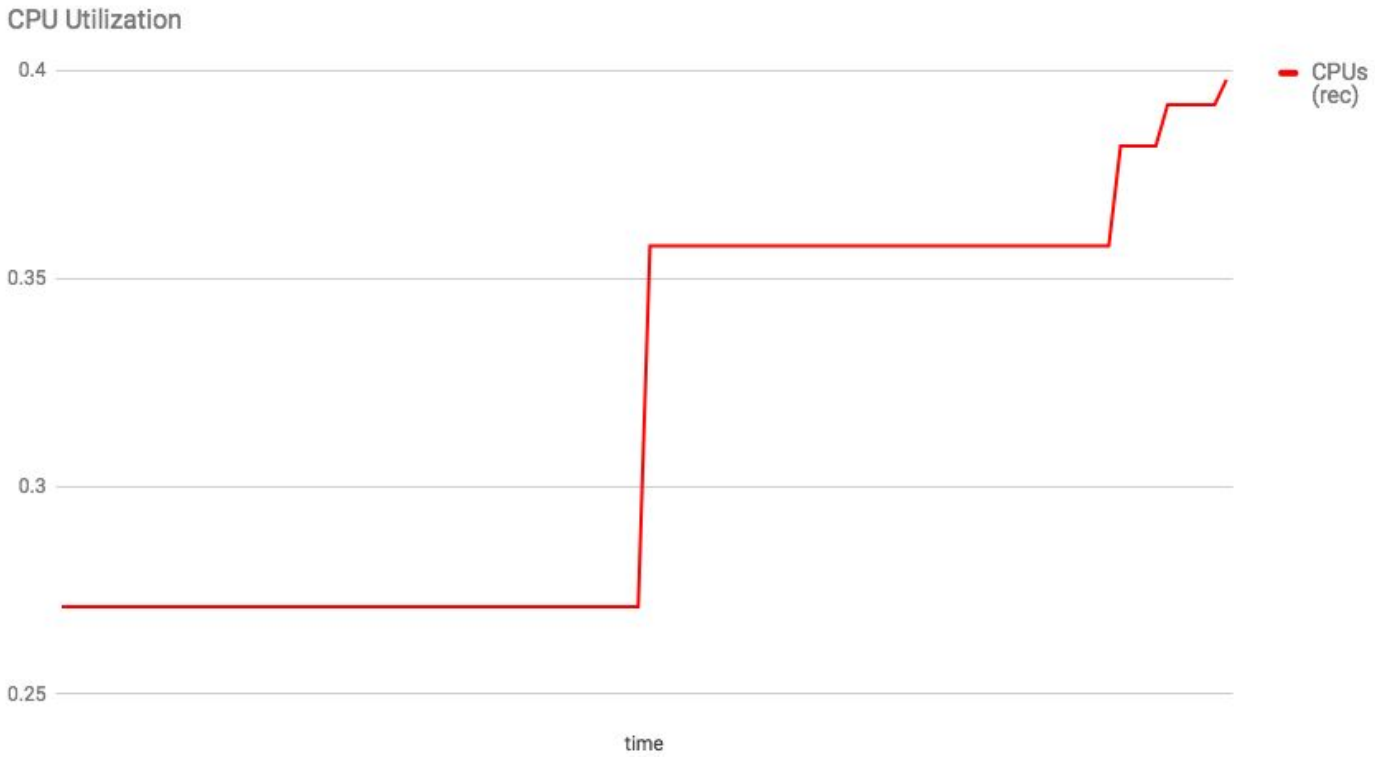
# Reconstruction



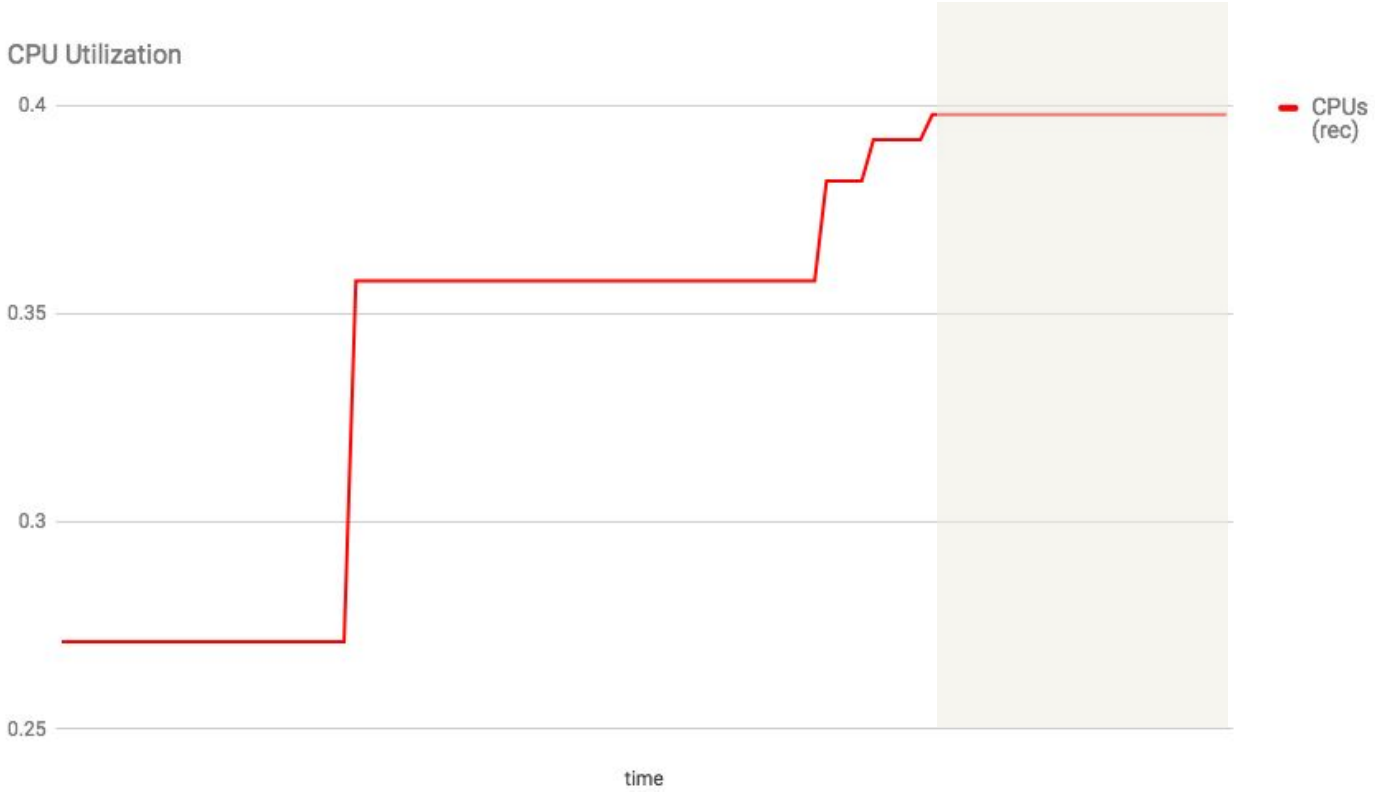
# Reconstruction



# Reconstruction



# Reconstruction



# Reconstruction - features

- Overestimates CPU utilization
- Conservative
  - non-idle jobs remain non-idle
  - idle job may become non-idle
  - job can remain non-idle more after reconstruction

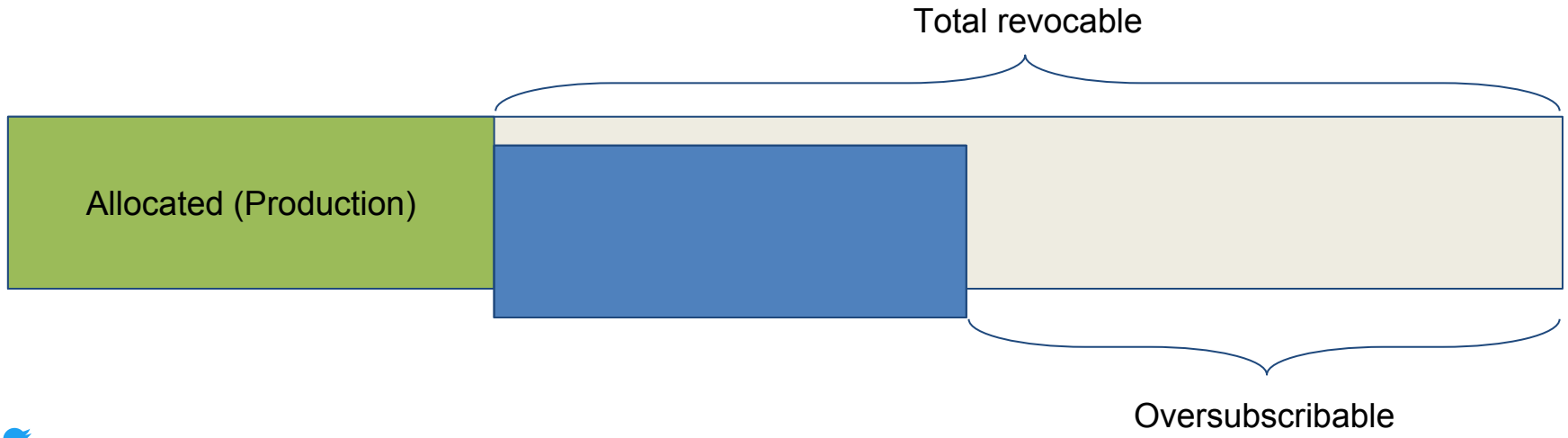
# Issues

Mesos oversubscription model didn’t work well:

- Difficulties with decreasing oversubscribable resources
- Delay between killing Non-Production task and update of oversubscribable resources
- Delay between launching Production task and update of oversubscribable resources

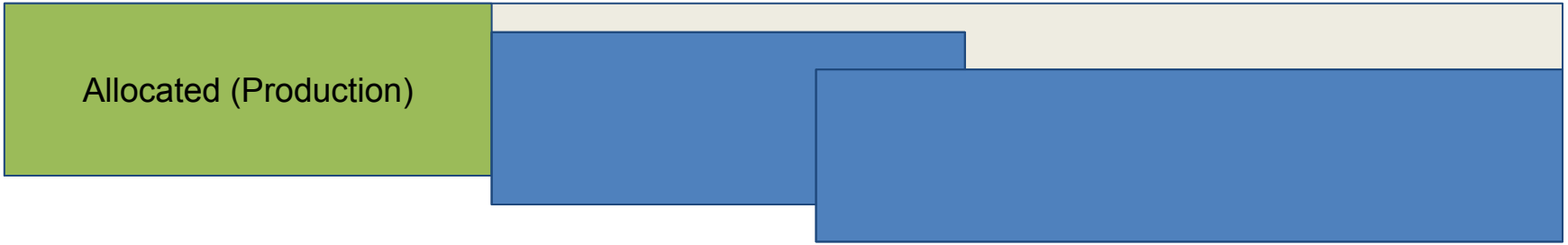
# Issues

**`ResourceEstimator`** must return *additional* amount of oversubscribable resources:




# Issues

**`ResourceEstimator`** must return *additional* amount of oversubscribable resources:

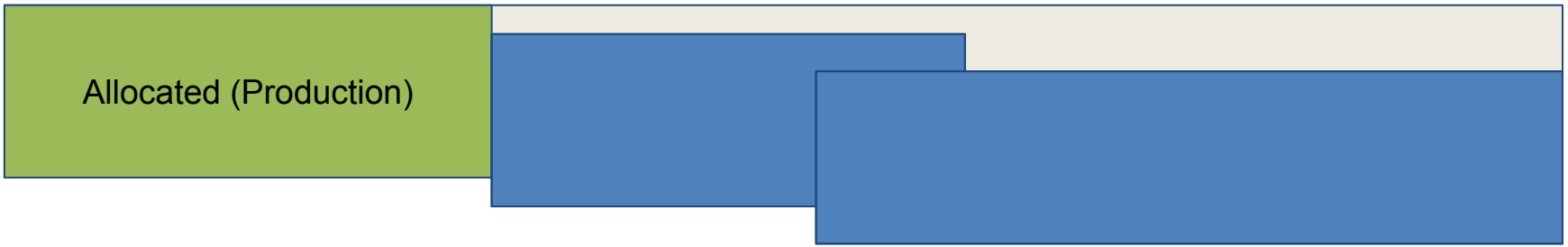


# Issues

To reduce total revocable resources
`QoSController` must kill some container…


# Issues

...and hope **`ResourceEstimator`** runs before scheduler assigns another job in killed job's place




# Workaround

```
class ResourceEstimator {
   Future<Resources> oversubscribable();
};

class QoSController {
  Future<list<QoSCorrection>> corrections();
};
```

# Implementation details

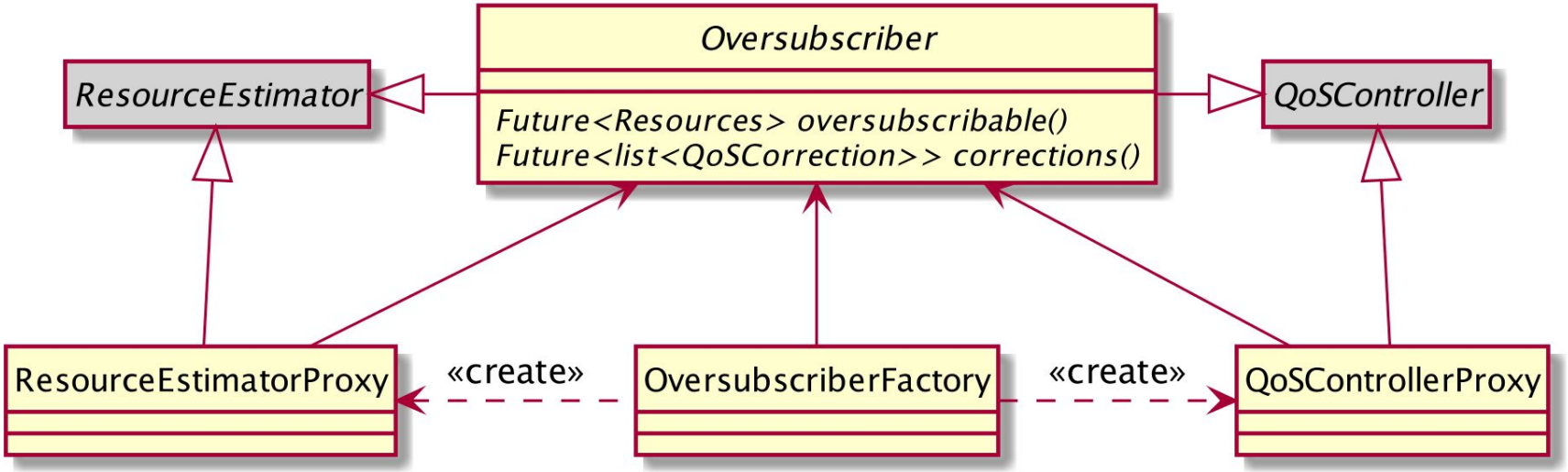
- Control resource estimates and QoS corrections by completing **`Future`** at the right time
- Hook executor and task lifecycle events to react to changes as soon as possible
- Run periodic sampling independently and set **`oversubscribed_resources_interval`** and **`qos_correction_interval_min`** to 0secs.

# Implementation details

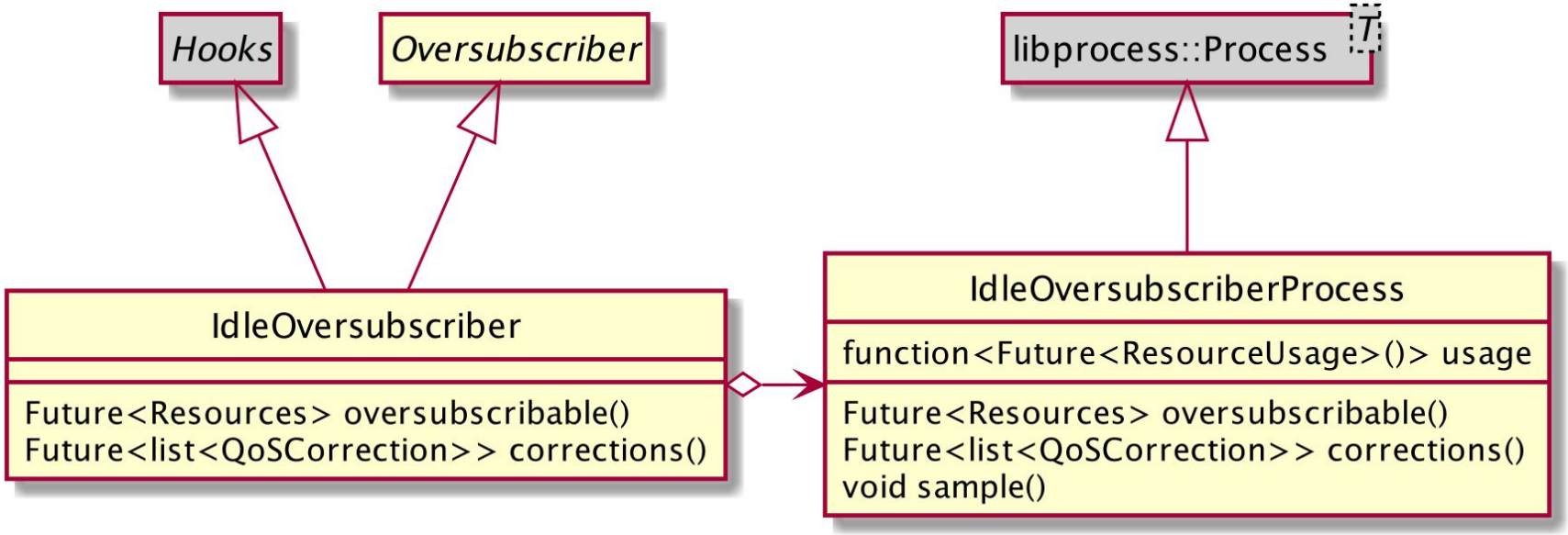
High coupling between **`ResourceEstimator`** and **`QoSController`**:

- Most of the calculations are performed by **`ResourceEstimator`** (detect if container is idle, estimate oversubscribable resources)
- **`QoSController`** relies on data from **`ResourceEstimator`** to kill containers

# Oversubscriber



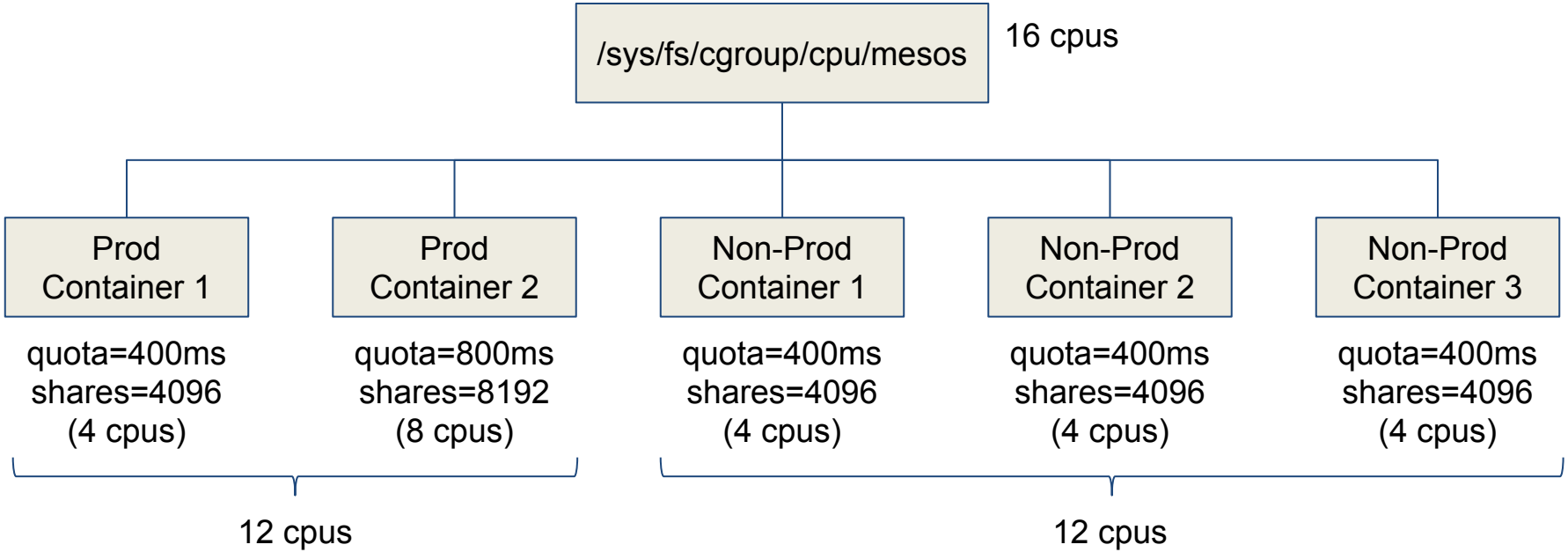

# IdleOversubscriber




# Configuration

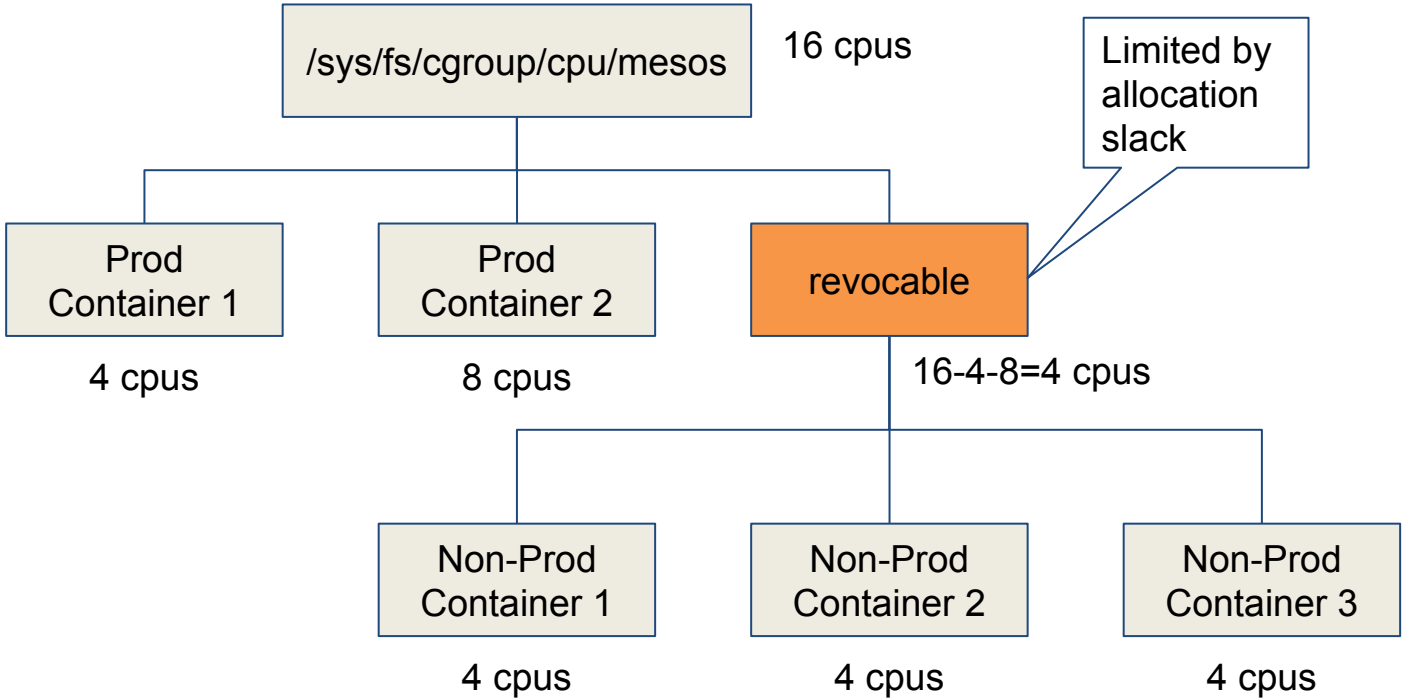
```
"libraries": [
  { "file": ...,
    "modules": [
      { "name": "org_apache_mesos_IdleOversubscriberHook",
        "parameters": [ … ]
      },
      { "name": "org_apache_mesos_IdleQoSController" },
      { "name": "org_apache_mesos_IdleResourceEstimator" }
    ]
  }
]
```

# Isolation




Non-production tasks can impact production tasks

# Isolation


/sys/fs/cgroup/cpu/mesos
16 cpus
Limited by allocation slack
Prod Container 1
4 cpus
Prod Container 2
8 cpus
revocable
16-4-8=4 cpus
Non-Prod Container 1
4 cpus
Non-Prod Container 2
4 cpus
Non-Prod Container 3
4 cpus

# Isolation issues

- Non-production tasks may never become non-idle because of *revocable* cgroup limit:
  - Detect contention between non-prod tasks
  - Ensure $\sum cpus_i \times threshold$ < allocation slack
- Enabling revocable isolation will take effect on new tasks only
- Disabling/Downgrading revocable isolation requires non-production tasks restart

# Scale issues

- Increased frequency of agents resources updates => more traffic between agents and master, more rescinded offers
  - Added tolerance to oversubscribable resource changes: do not update if oversubscribable resources increased by less than 1 cpu

# Scale issues

- Increased number of rescinded offers:
  - Changed scheduler to handle rescinded offers ASAP

- Scheduler unawareness of revocable and non-revocable resources relation

# Current state

- Tested on our scale test cluster
- Not in production yet
  - Few issues remaining in scheduler

- Deployment
  - Need to migrate all non-production jobs to revocable resources. This can be changed in scheduler, but requires rescheduling the jobs.

# Thank you!