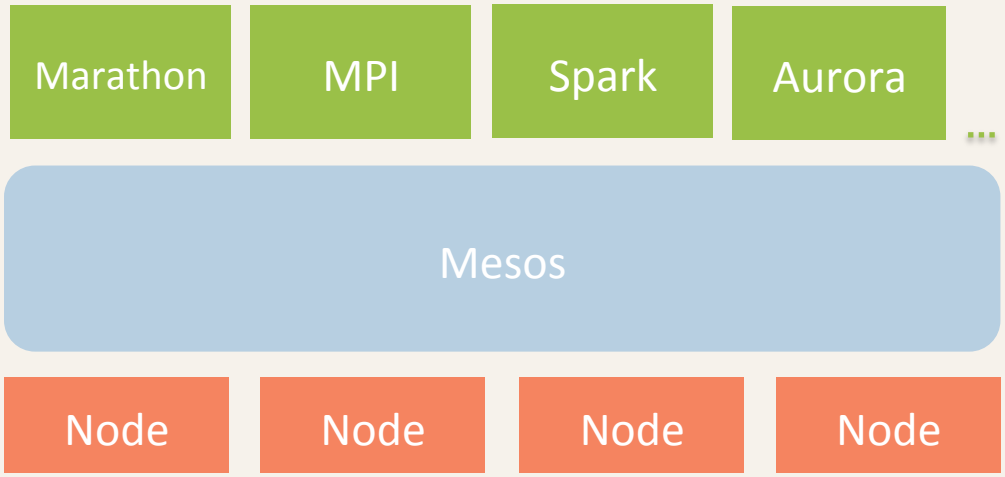
Running YARN alongside Mesos

Mohit Soni
ebay inc

Renan DelValle
SUNY Binghamton

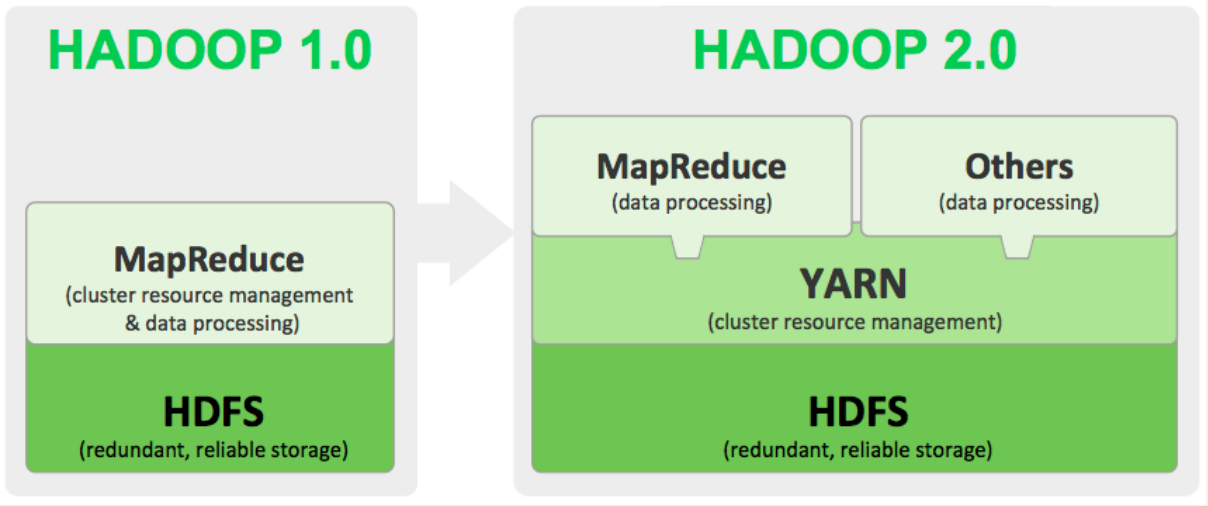
About Mesos

- Cluster manager
- Two Level Scheduler
- Supports service and analytical jobs



About YARN

- Resource Negotiator
- Single level scheduler
- Supports different types of analytical jobs



Problem Statement

- Independent resource managers statically partitions datacenter
- Mesos supports long running services and analytics workloads
- YARN ecosystem is currently around analytics/data processing

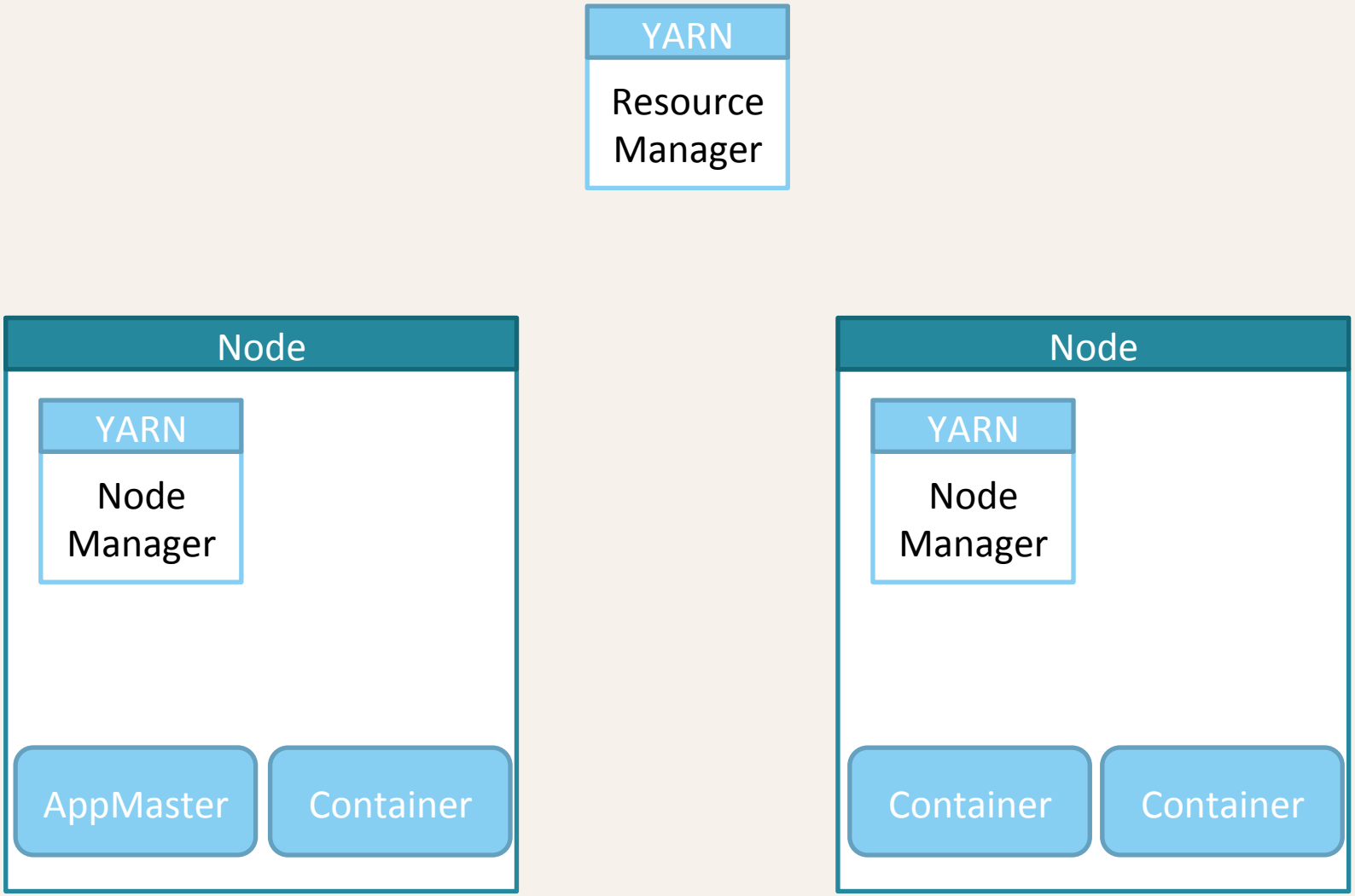
Goal

Share resources between YARN and Mesos, with Mesos being the resource manager for the data center.

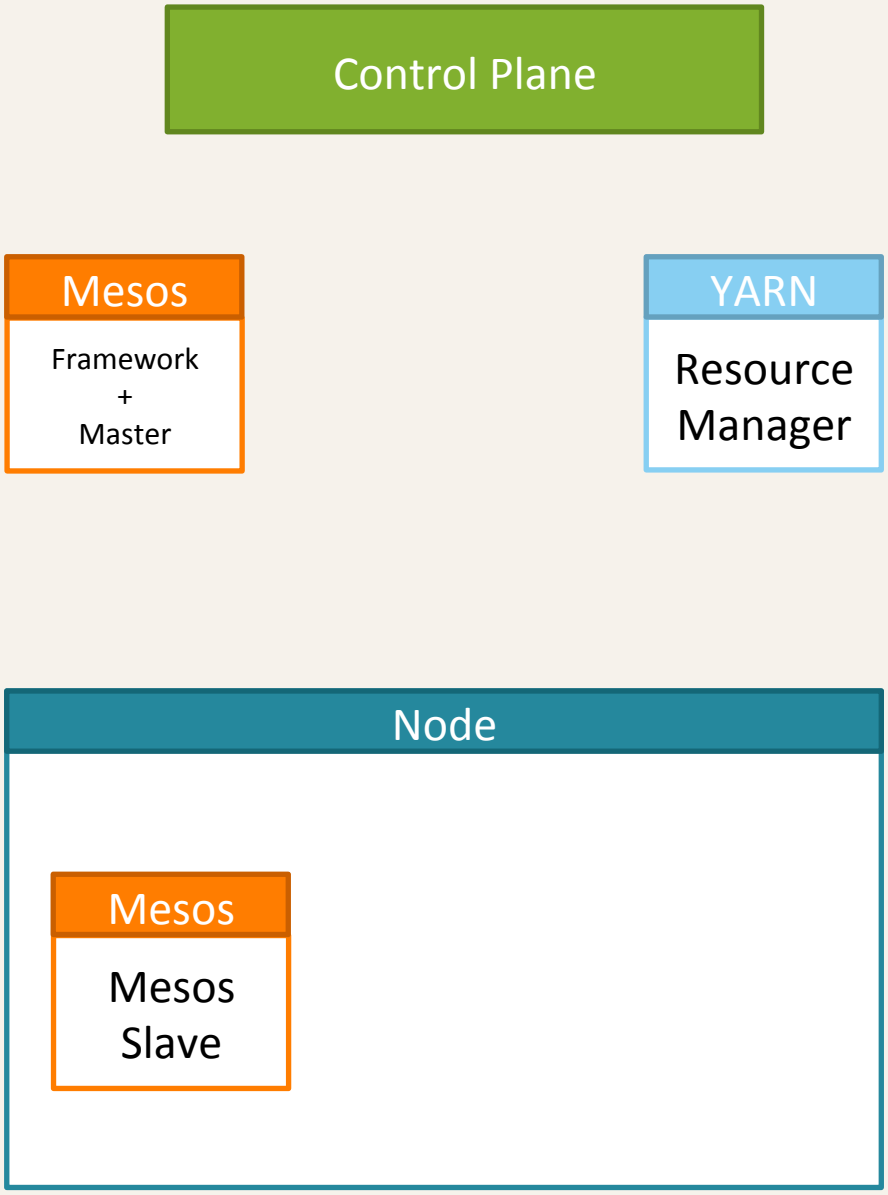
Solution Characteristics

- Non-intrusive, avoids modifying Mesos or YARN protocols
  - Easy future upgrades
  - Easier certification path
- Use YARN's scheduling data, for providing & rescinding resources to YARN

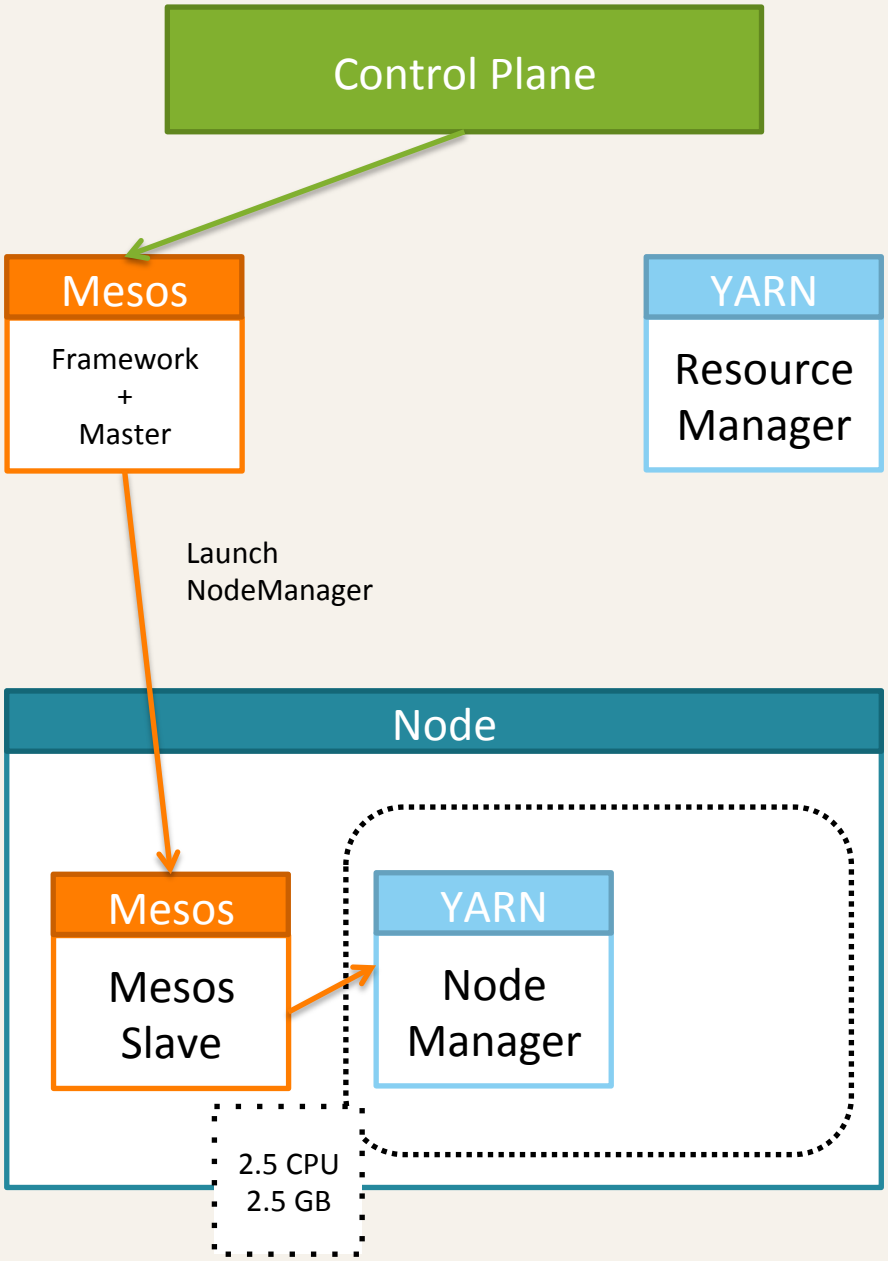
YARN Architecture Overview



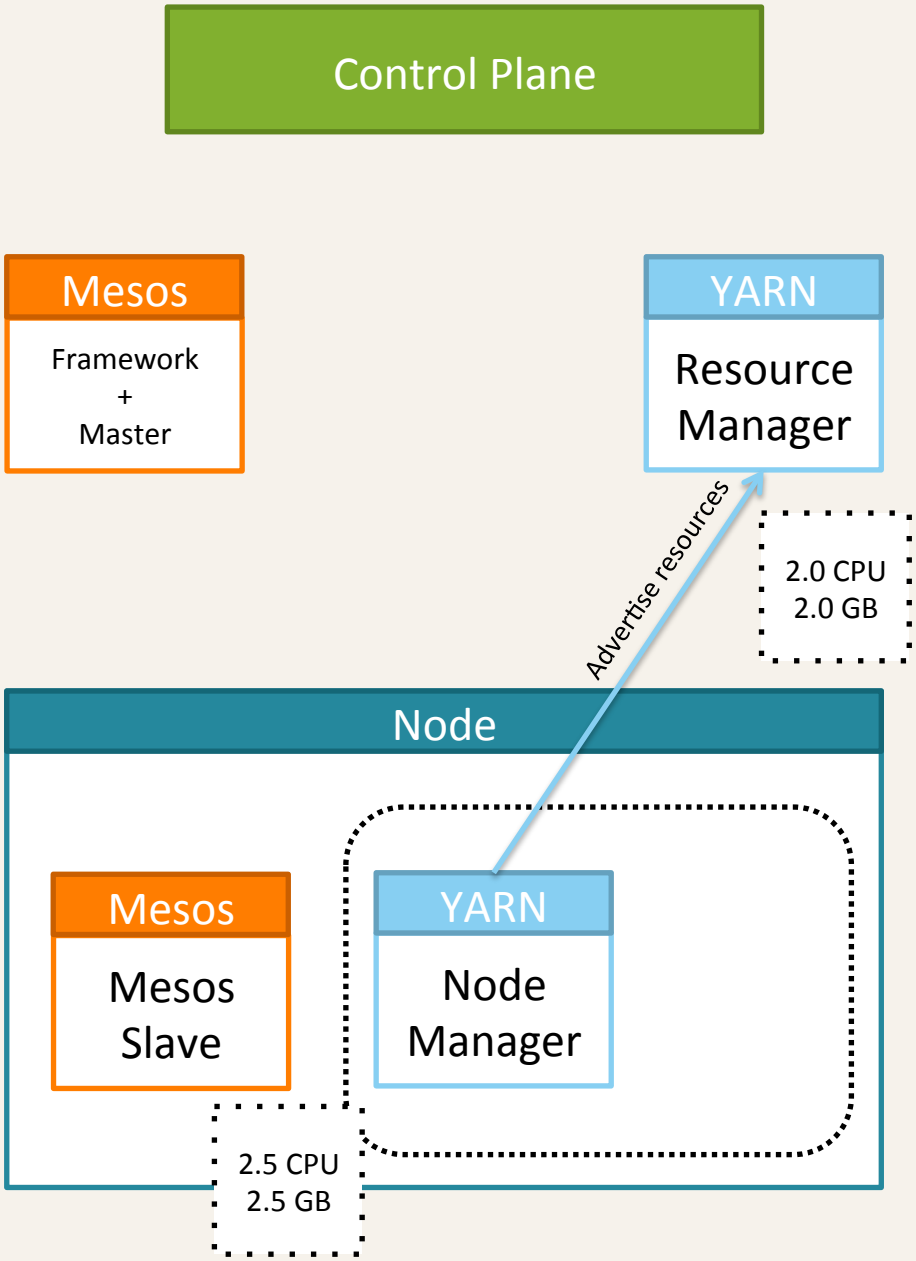
How it works


Control Plane
Mesos
Framework
+
Master
YARN
Resource
Manager
Node
Mesos
Mesos
Slave

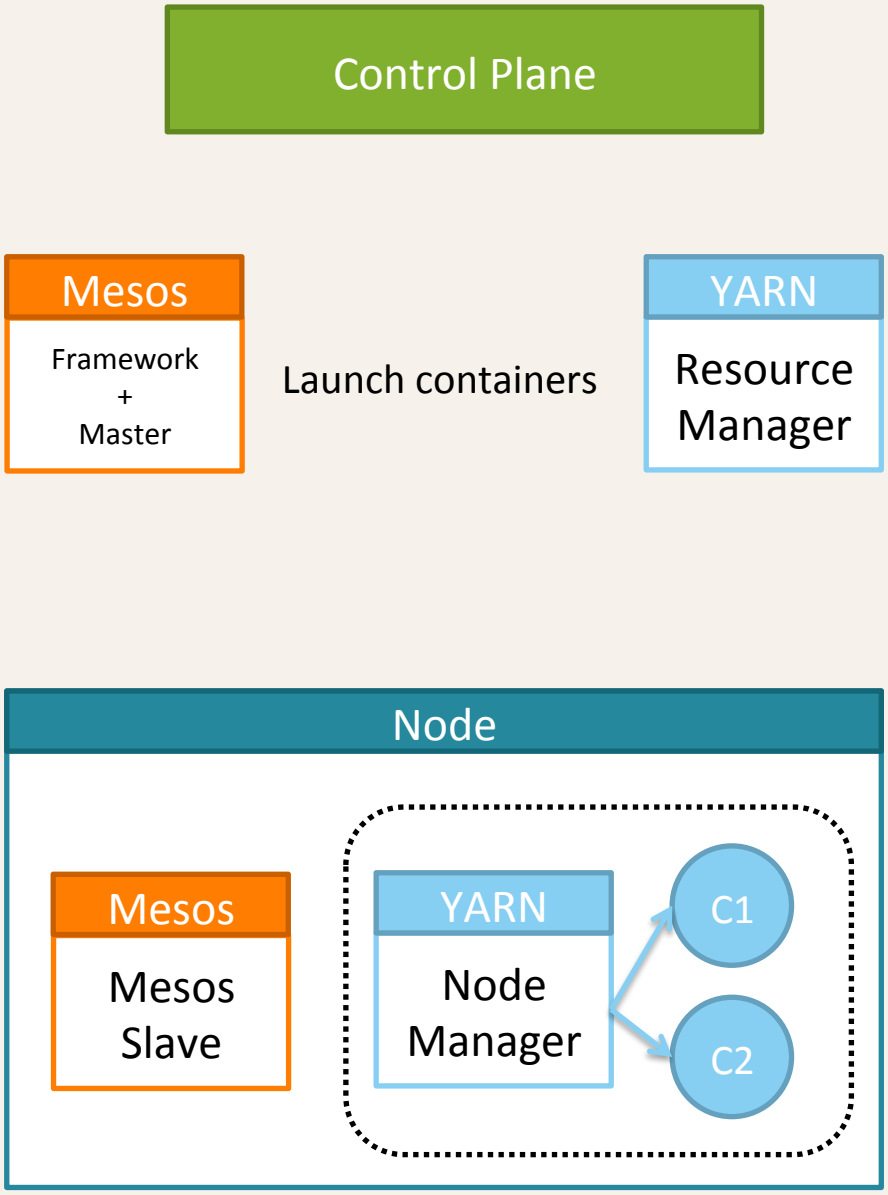
How it works


Control Plane
Mesos
Framework
+
Master
YARN
Resource
Manager
Launch
NodeManager
Node
Mesos
Mesos
Slave
YARN
Node
Manager
2.5 CPU
2.5 GB

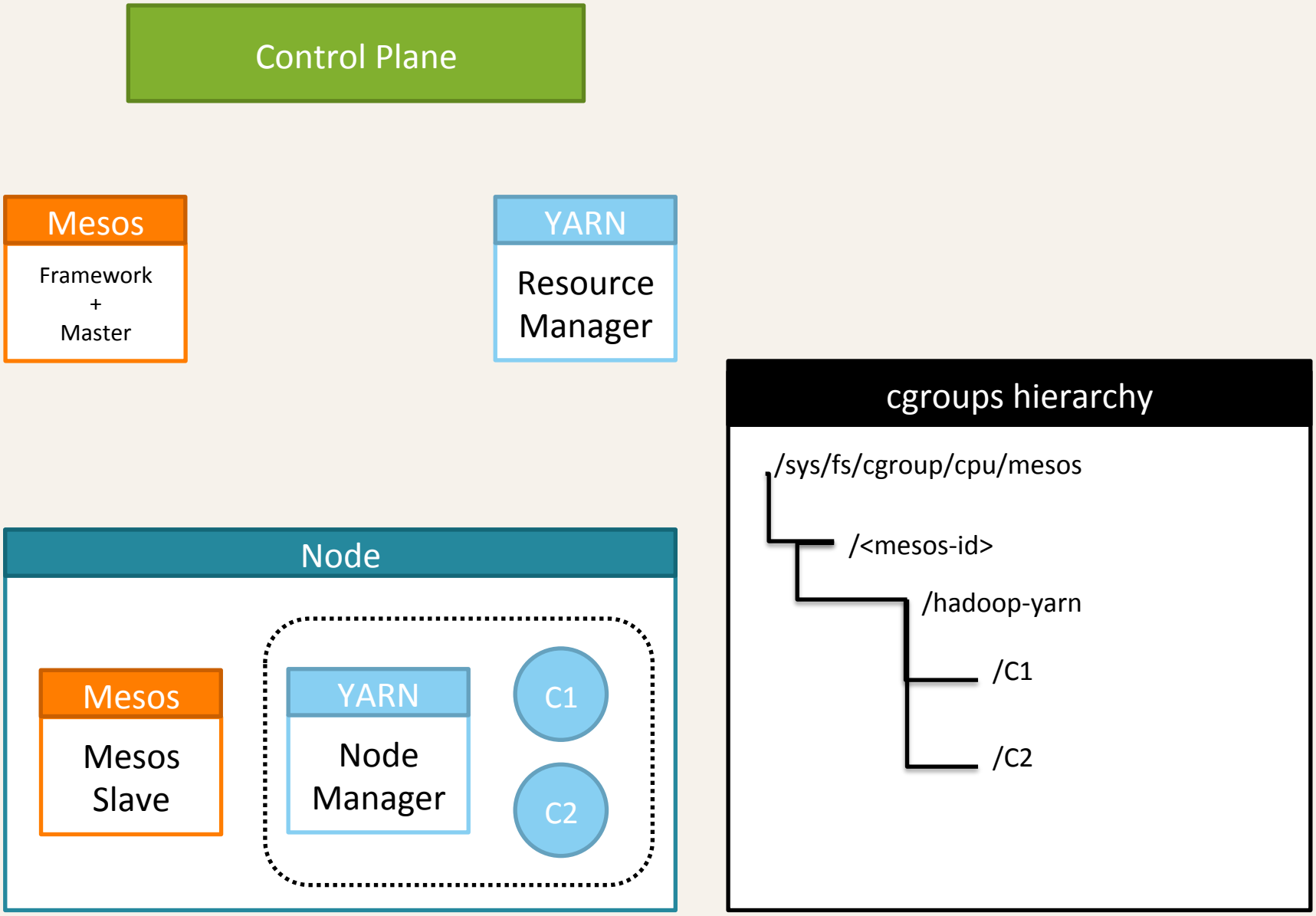
How it works


Control Plane
Mesos
Framework
+
Master
YARN
Resource
Manager
Advertise resources
2.0 CPU
2.0 GB
Node
Mesos
Mesos
Slave
YARN
Node
Manager
2.5 CPU
2.5 GB

How it works


Control Plane
Mesos
Framework
+
Master
Launch containers
YARN
Resource
Manager
Node
Mesos
Mesos
Slave
YARN
Node
Manager
C1
C2

How it works


Control Plane
Mesos
Framework
+
Master
YARN
Resource
Manager
cgroups hierarchy
/sys/fs/cgroup/cpu/mesos
/<mesos-id>
/hadoop-yarn
/C1
/C2
Node
Mesos
Mesos
Slave
YARN
Node
Manager
C1
C2

Scenario 1: Handling Traffic Spikes

Traffic spike for
ebay.com/sch

Control Plane

Mesos
Framework
+
Master

YARN
Resource
Manager

Hadoop cluster Node 1

Mesos
Slave

YARN
Node
Manager

8 CPU
8 GB

8.5 CPU / 8.5 GB

Hadoop cluster Node 2

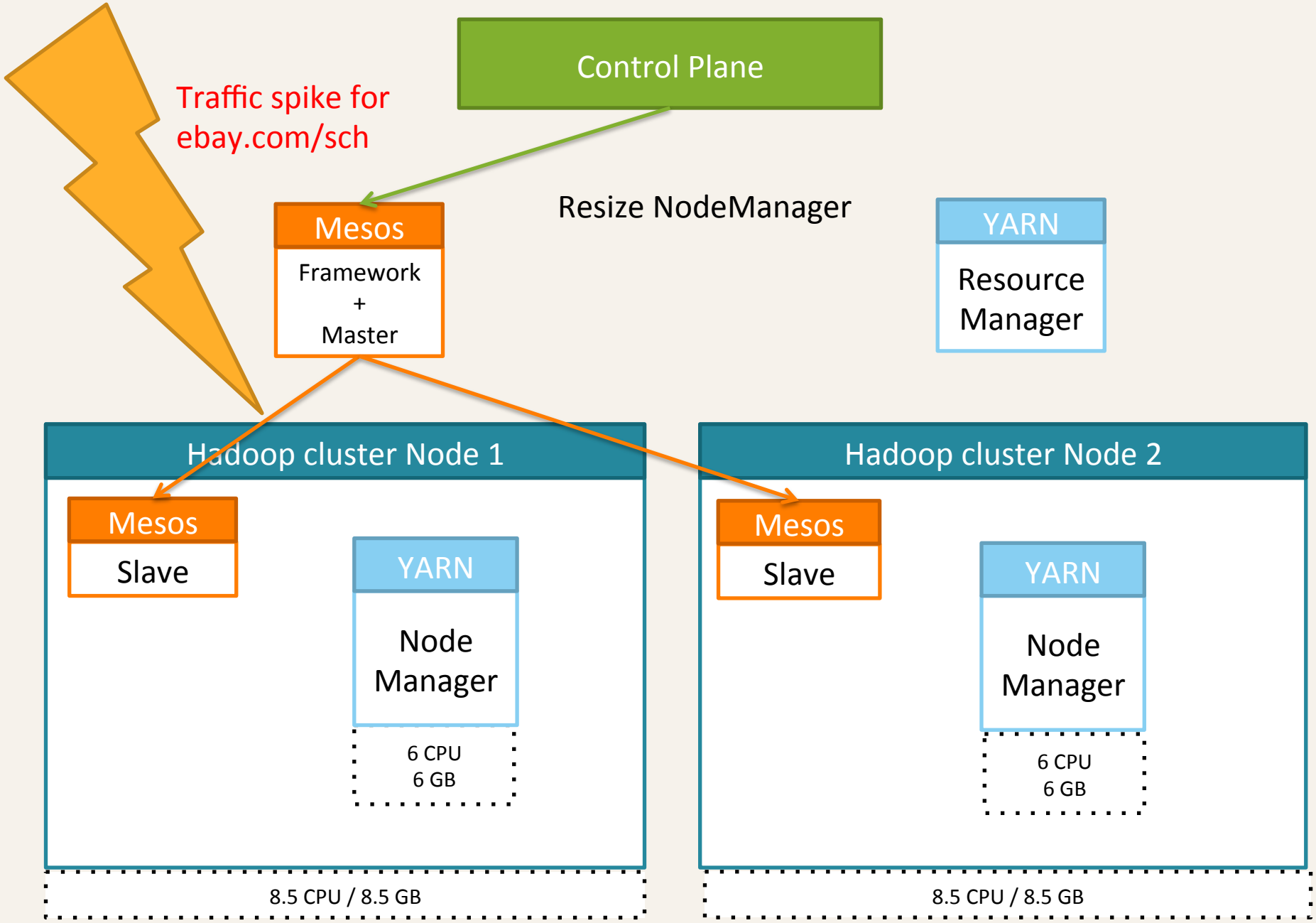
Mesos
Slave

YARN
Node
Manager

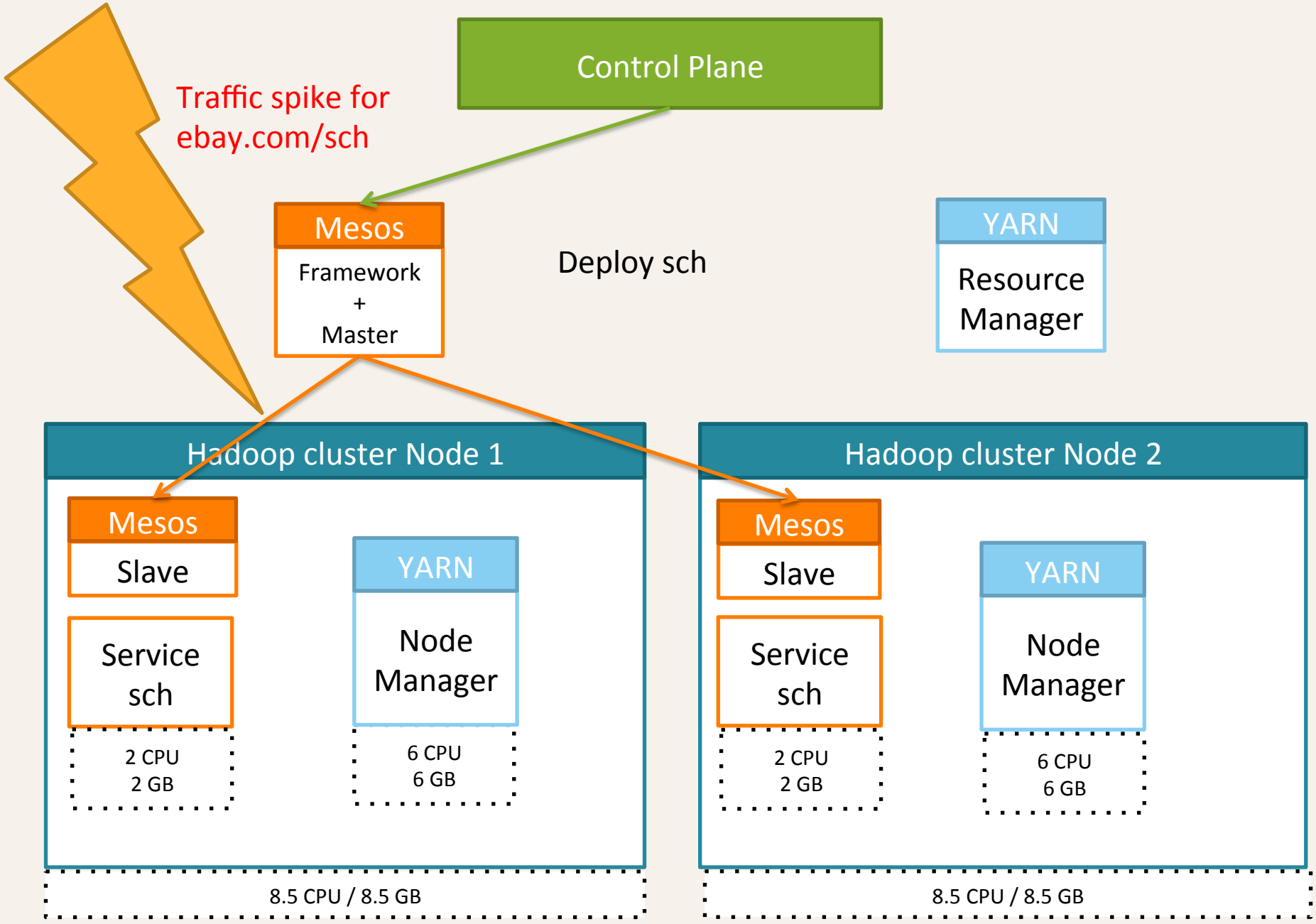
8 CPU
8 GB

8.5 CPU / 8.5 GB

Scenario 1: Handling Traffic Spikes



Scenario 1: Handling Traffic Spikes



Scenario 1: Handling Traffic Spikes

ebay.com/sch
survives

Control Plane

Mesos
Framework
+
Master

YARN
Resource
Manager

Hadoop cluster Node 1

Mesos
Slave

Service
sch

2 CPU
2 GB

YARN
Node
Manager

6 CPU
6 GB

8.5 CPU / 8.5 GB

Hadoop cluster Node 2

Mesos
Slave

Service
sch

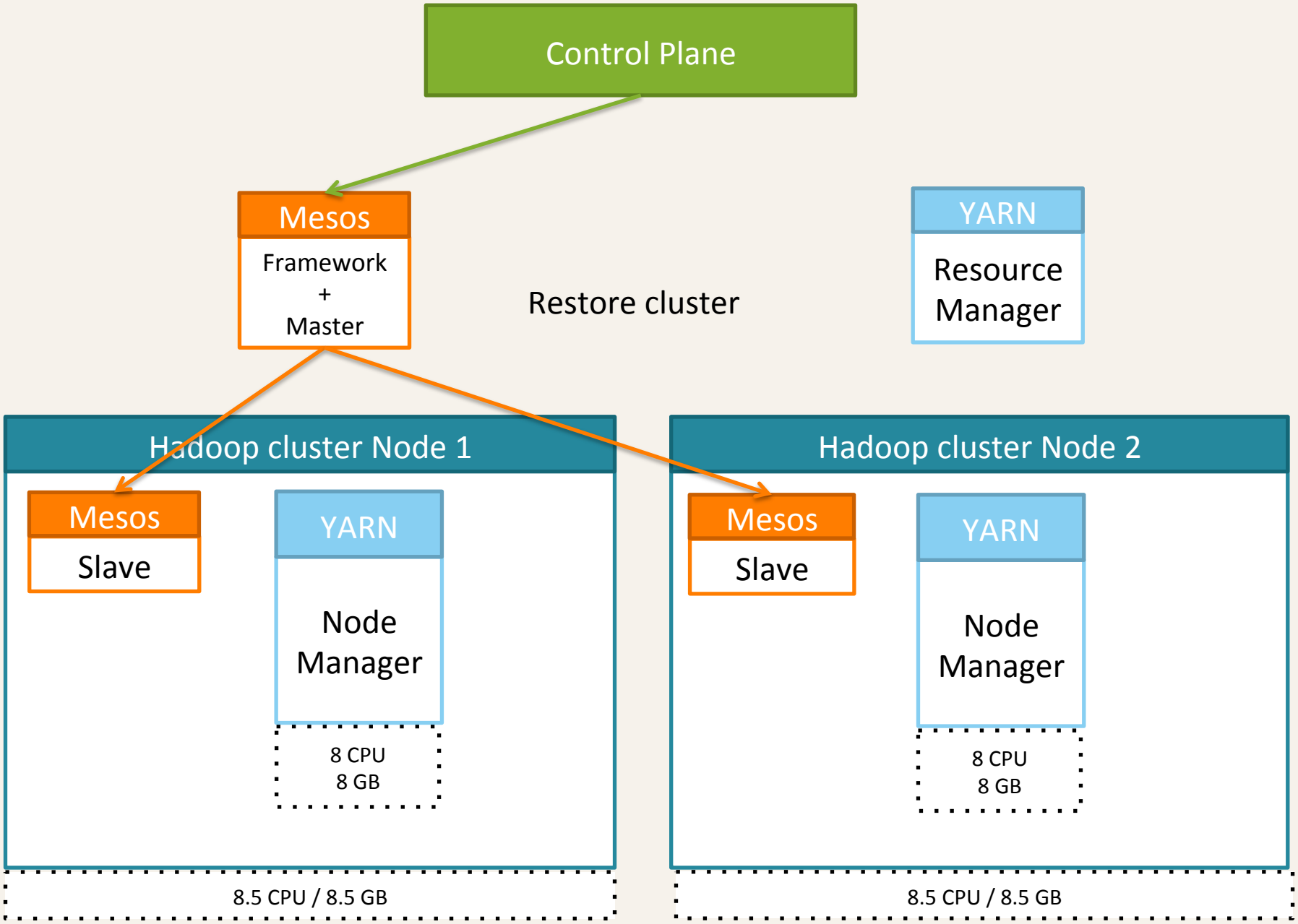
2 CPU
2 GB

YARN
Node
Manager

6 CPU
6 GB

8.5 CPU / 8.5 GB

Scenario 1: Handling Traffic Spikes

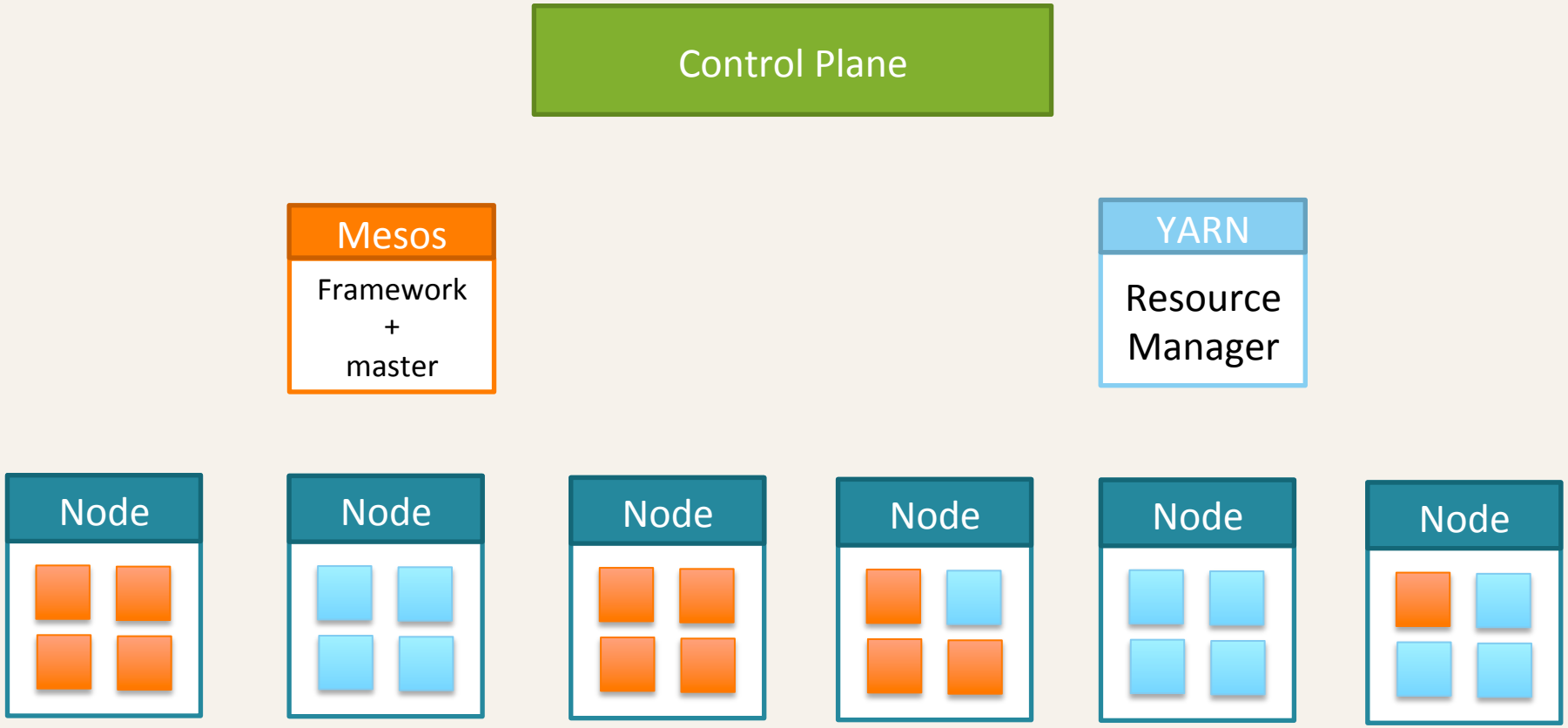


YARN Pending Improvements

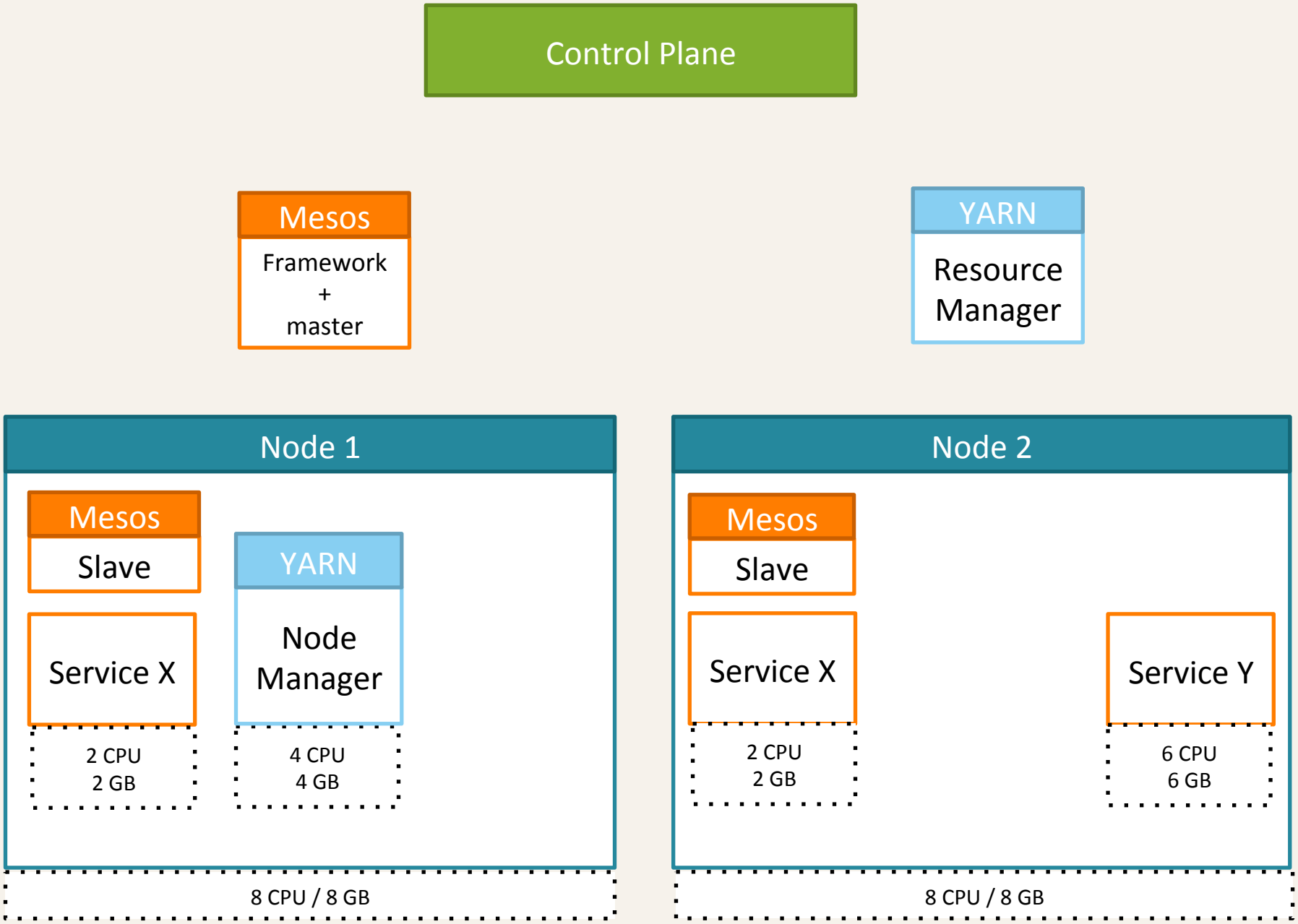
- Restarting NodeManager, kills child containers (YARN-1336)
- Restarting AppMaster, kills child containers across all nodes (YARN-1489)
- NodeManager's pending support for cgroups memory subsystem
- Getting richer scheduling information from ResourceManager

Future Vision

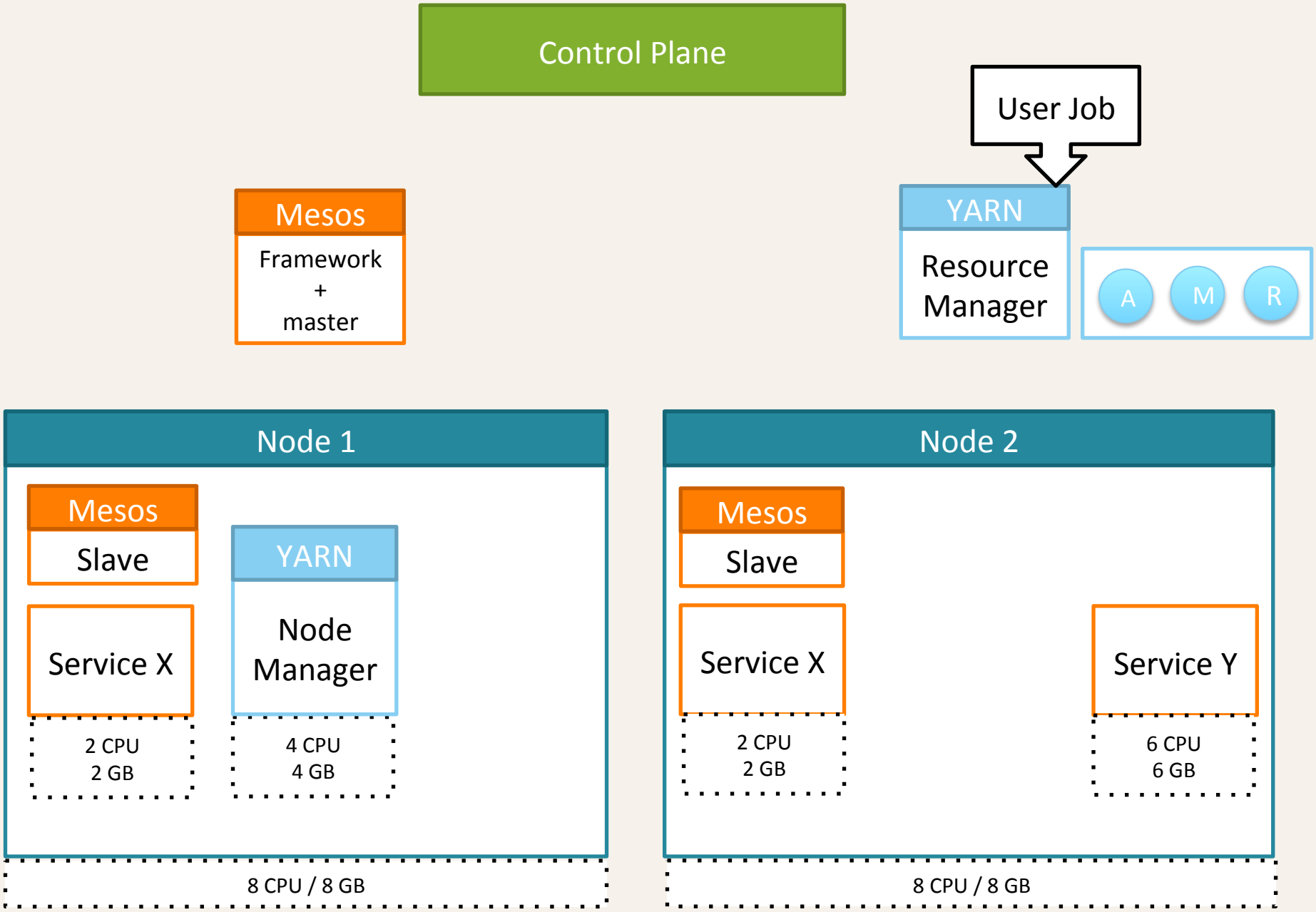
- One unified cluster per datacenter
- YARN and Mesos tasks co-exist on Nodes
- Provision resources on demand.



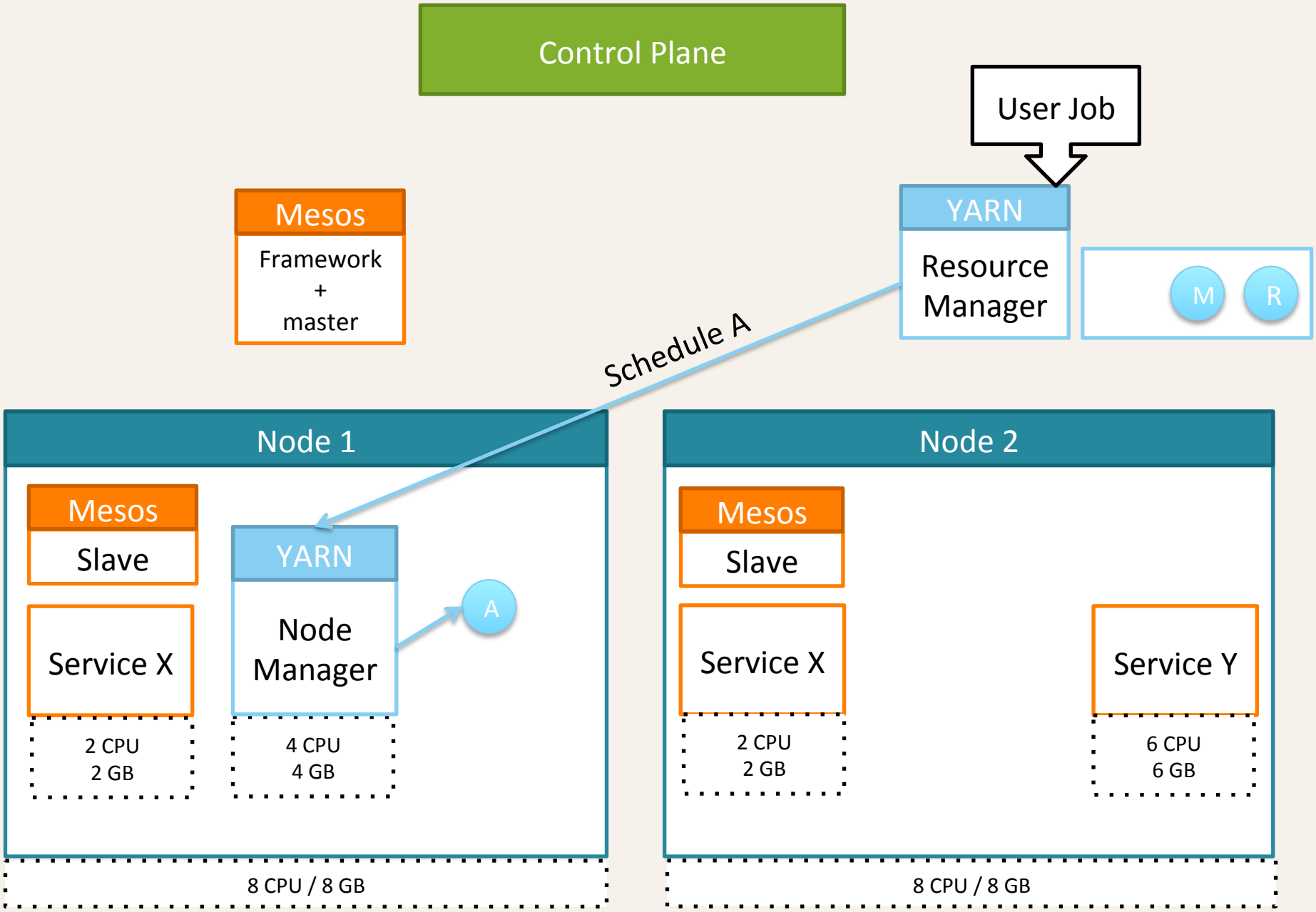
Scenario 2: Scaling YARN



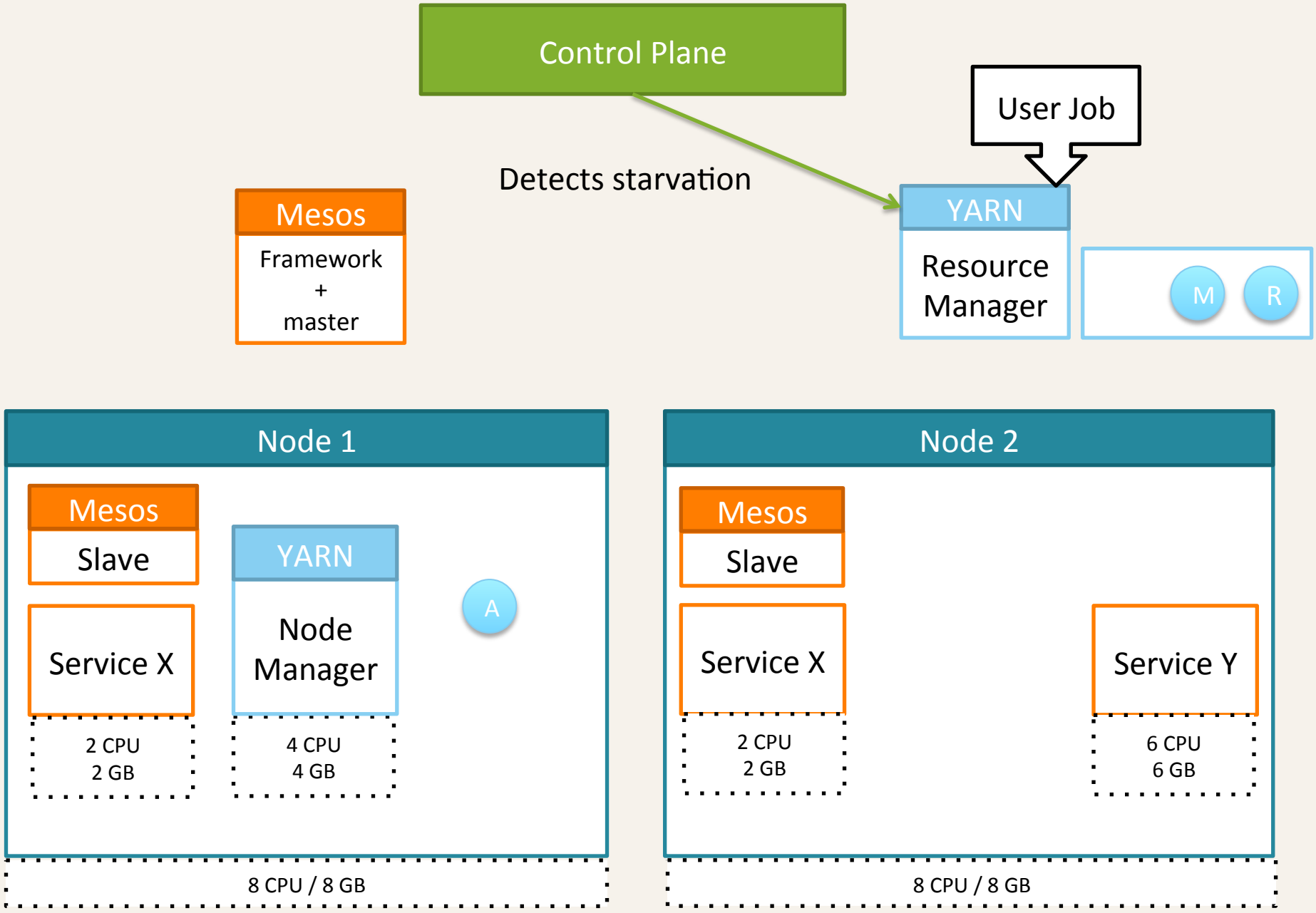
Scenario 2: Scaling YARN



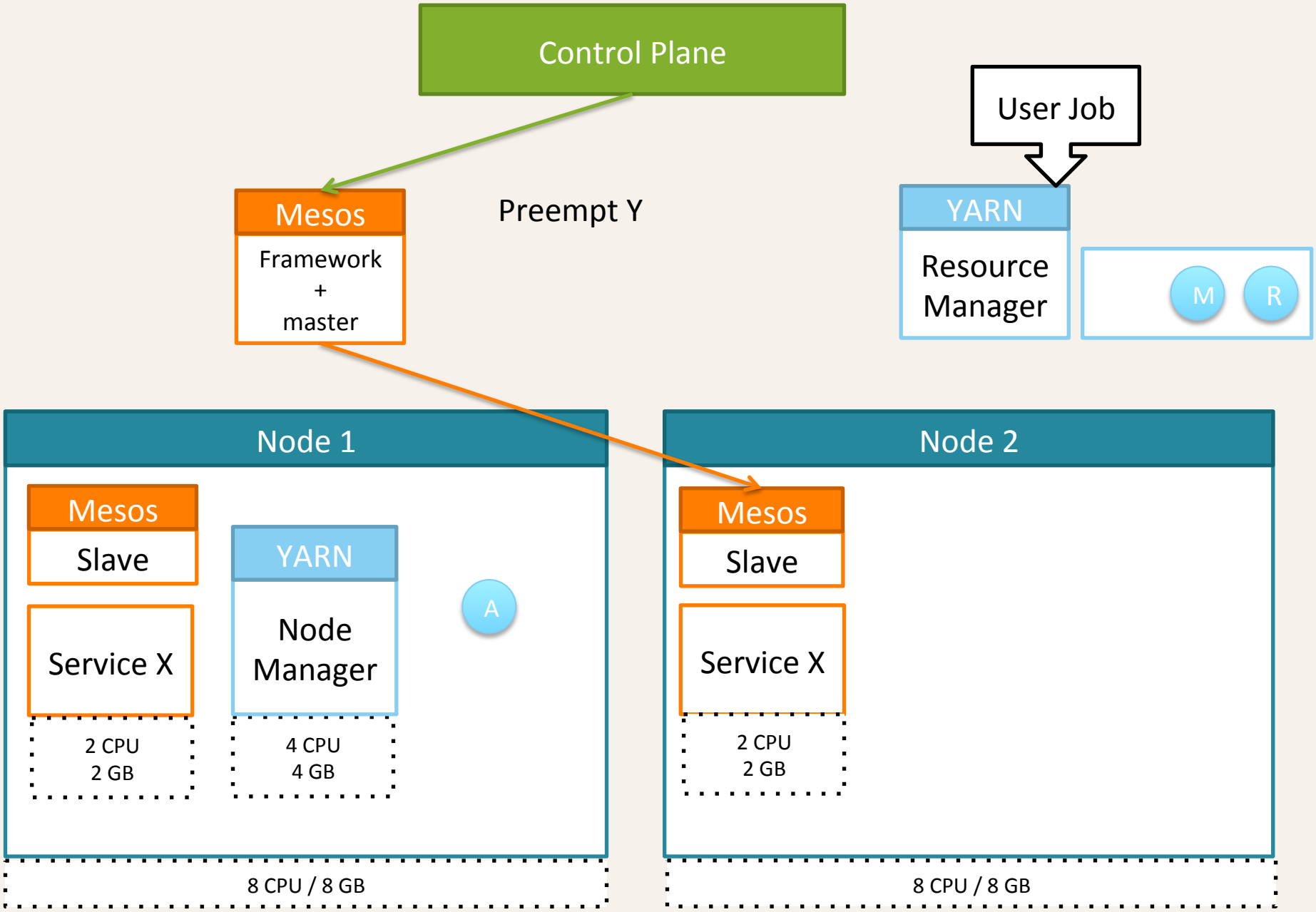
Scenario 2: Scaling YARN



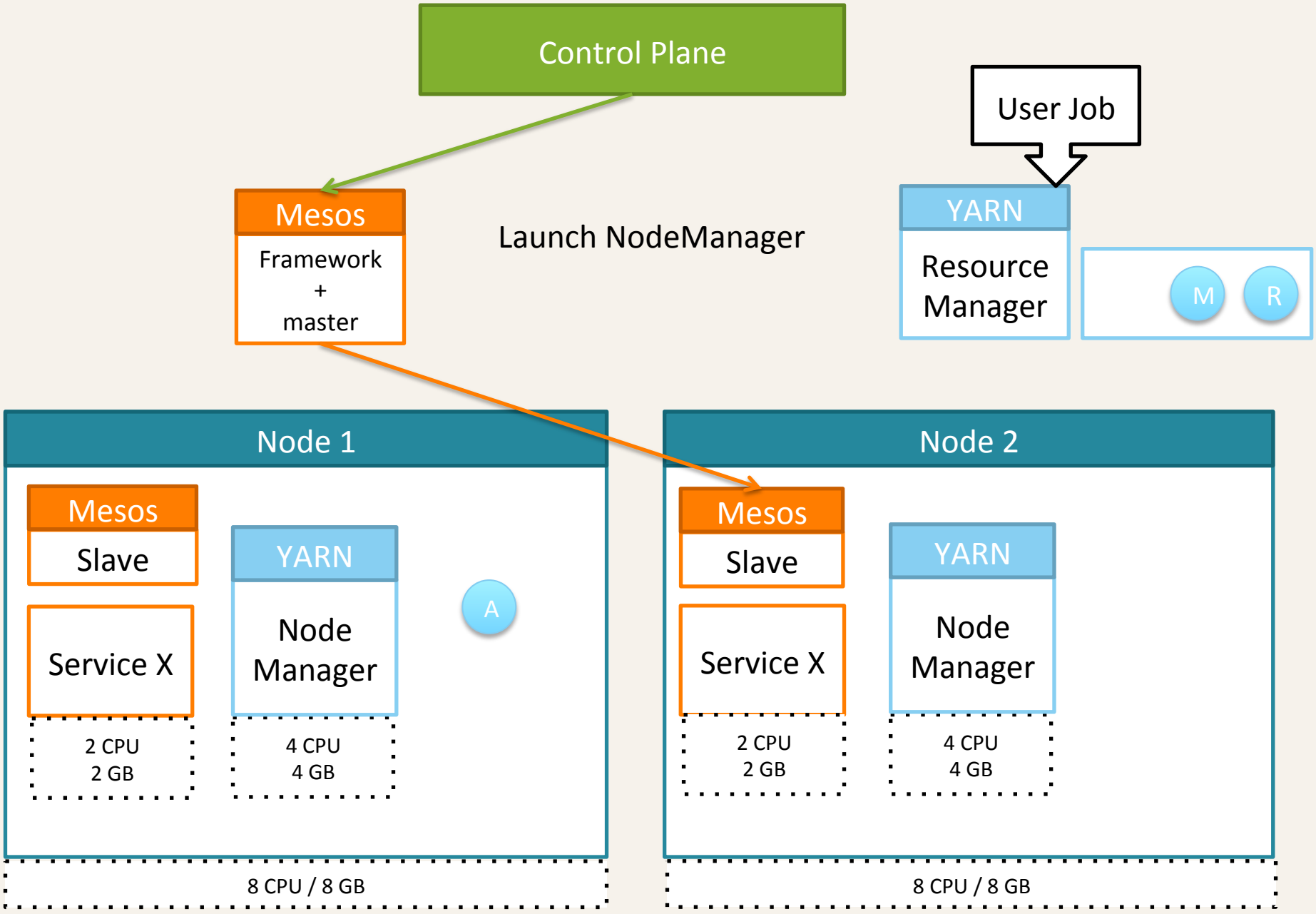
Scenario 2: Scaling YARN



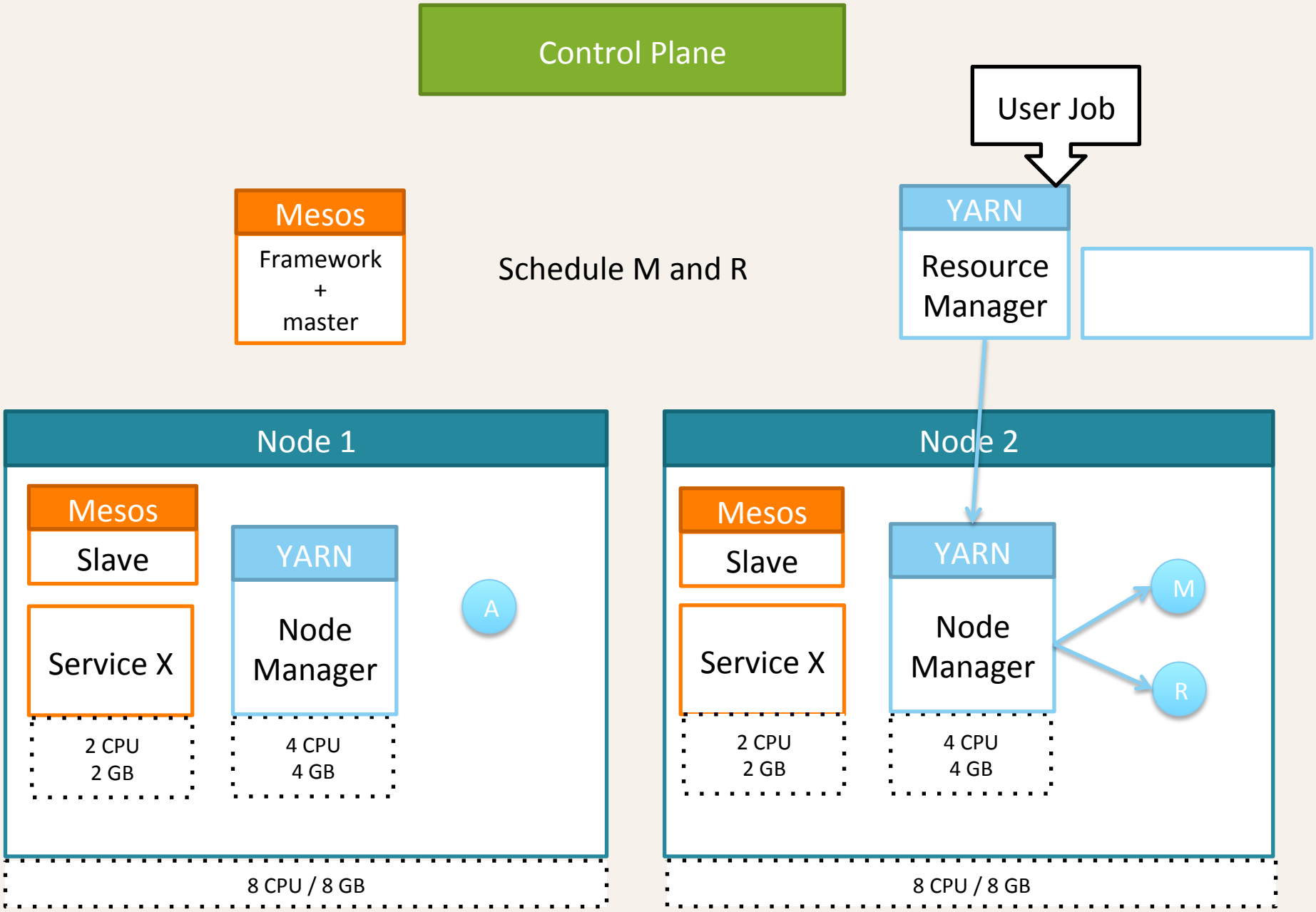
Scenario 2: Scaling YARN



Scenario 2: Scaling YARN



Scenario 2: Scaling YARN



New YARN API (YARN-2408)

```
<resourceRequests>
  <MB>96256</MB>
  <VCores>94</VCores>
  <appMaster>
    <applicationId>application_</applicationId>
    <applicationAttemptId>appattempt_</applicationAttemptId>
    <queueName>default</queueName>
    <totalPendingMB>96256</totalPendingMB>
    <totalPendingVCores>94</totalPendingVCores>
    <numResourceRequests>3</numResourceRequests>
    <resourceRequests>
      <request>
        <MB>1024</MB>
        <VCores>1</VCores>
        <resourceName>/default-rack</resourceName>
        <numContainers>94</numContainers>
        <relaxLocality>true</relaxLocality>
        <priority>20</priority>
      </request>
      <request>
        <MB>1024</MB>
        <VCores>1</VCores>
        <resourceName>*</resourceName>
        <numContainers>94</numContainers>
        <relaxLocality>true</relaxLocality>
        <priority>20</priority>
      </request>
      <request>
        <MB>1024</MB>
        <VCores>1</VCores>
        <resourceName>master</resourceName>
        <numContainers>94</numContainers>
        <relaxLocality>true</relaxLocality>
        <priority>20</priority>
      </request>
    </resourceRequests>
  </appMaster>
</resourceRequests>
```

- Resource Requests snapshot API:
  - Memory
  - Virtual Cores
  - Locality constraint
- REST API with JSON & XML output
- Non-intrusive – simply exposes more information from the Resource Manager
- Helps control plane decide NodeManager sizing

Control Plane (Mesos Framework?)

Design scope:

- Flex up or down, vertically or horizontally?
- Determining NodeManager profile for flex Up
  - Small (2 CPU, 4 GB RAM), **OR** Large (8 CPU, 24 GB RAM)
- Choosing NodeManager(s) to flex down, avoiding ones
  - which runs AppMaster container
  - whose child containers are critical (ex: HBase zone servers)

Thanks!

Sample Aurora Job

```
#imports
pre_cleanup = Process(...)
make_cgroups_dir = Process(...)

configure_cgroups = Process(name = 'configure_cgroups', cmdline = "MY_TASK_ID=`pwd | awk -F'/' '{ print $
(NF-1) }'` && echo 'hadoop' | sudo -S sed -i \"s@mesos.*/hadoop-yarn@mesos/$MY_TASK_ID/hadoop-yarn@g\" /usr/
local/hadoop/etc/hadoop/yarn-site.xml")

start = Process(name = 'start', cmdline = "source %s; %s start nodemanager; sleep 10;" % (BASHRC,
YARN_DAEMONS))

monitor = Process(name = 'monitor', cmdline = "sleep 10; PID=`cat /tmp/yarn-hduser-nodemanager.pid`; echo
'Monitoring nodemanager pid: ' ${PID}; while [ -e /proc/${PID} ]; do sleep 1; done")

stop = Process(name = 'stop',final = True, cmdline = "source %s; %s stop nodemanager" % (BASHRC,
YARN_DAEMONS))

template_task = Task(
  processes = [pre_cleanup, make_cgroups_dir, configure_cgroups, start, monitor, stop],
  constraints = order(pre_cleanup, make_cgroups_dir, configure_cgroups, start, monitor) + order(stop)
)

small_task = template_task(name = 'small_task', resources = Resources(cpu=1.0, ram=512*MB, disk=2048*MB))

large_task = template_task(name = 'large_task', resources = Resources(cpu=2.0, ram=2048*MB, disk=2048*MB))

jobs = [Service(task = large_task, instances = instances, cluster = 'devcluster',
    role = ROLE, environment = 'devel', name = 'yarnlarge')]

# Job config, for a small task.
#small_jobs = [Service(task = small_task, instances = instances, cluster = 'devcluster',
    role = ROLE, environment = 'devel', name = 'yarnsmall')]
```