YARN

Yet Another Resource Negotiator

YARN was introduced in Hadoop 2.0 to separately handle

the management of resources on the Hadoop cluster

YARN

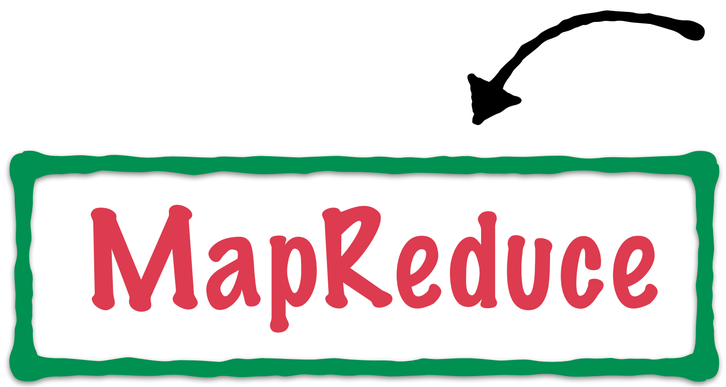
Yet Another Resource Negotiator

YARN co-ordinates all the different MapReduce tasks running on the cluster

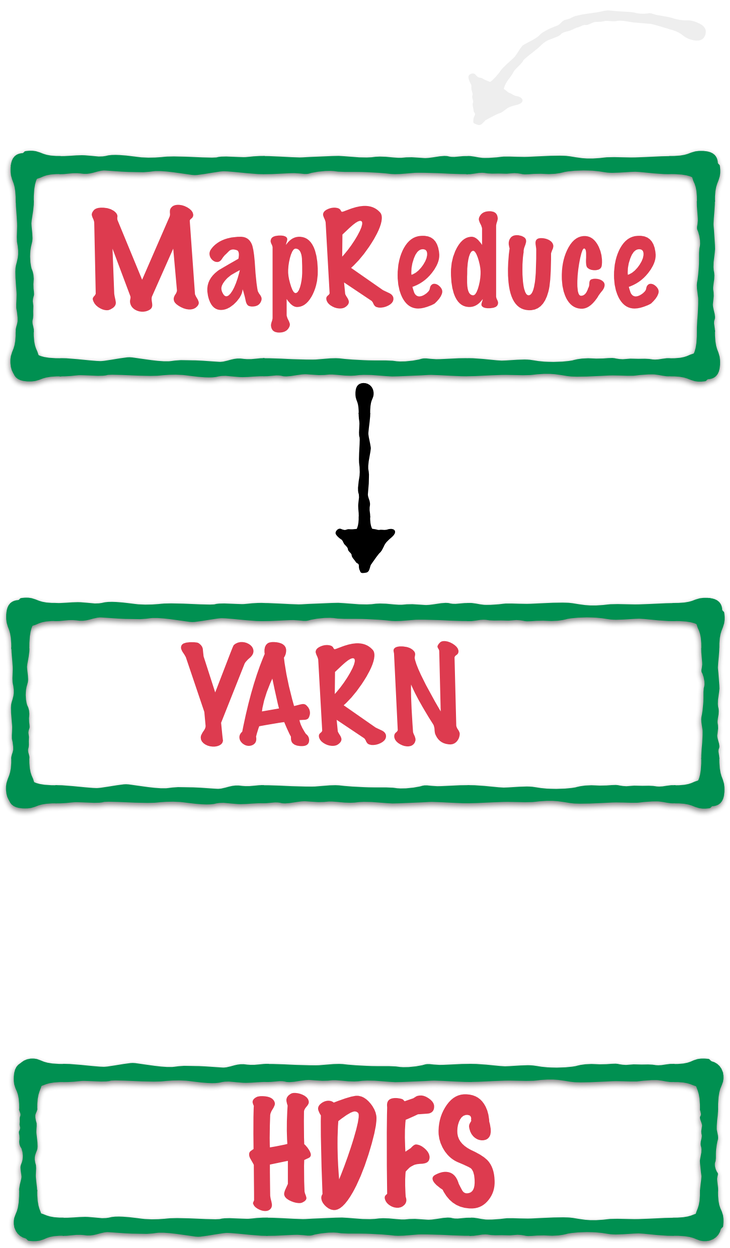
YARN

Yet Another Resource Negotiator

YARN also monitors for failures and assigns new nodes when others fail

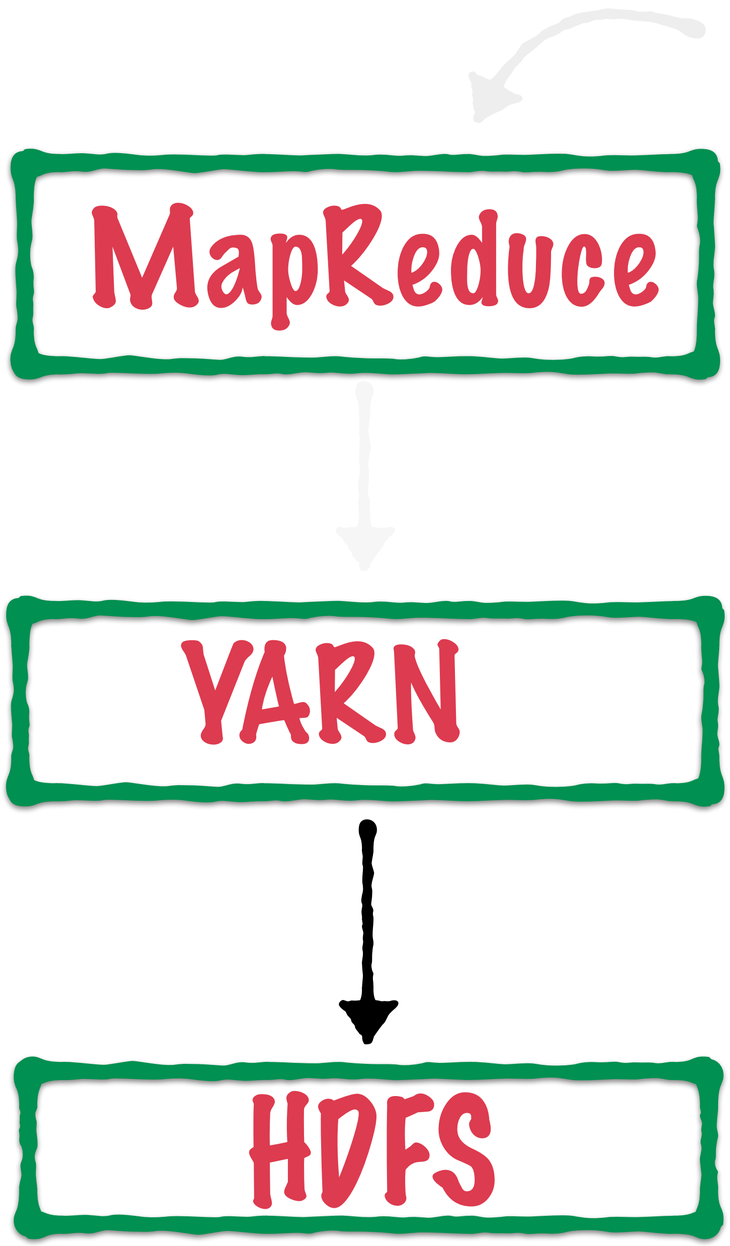


User defines map and reduce tasks using the MapReduce API



User defines map and reduce tasks using the MapReduce API

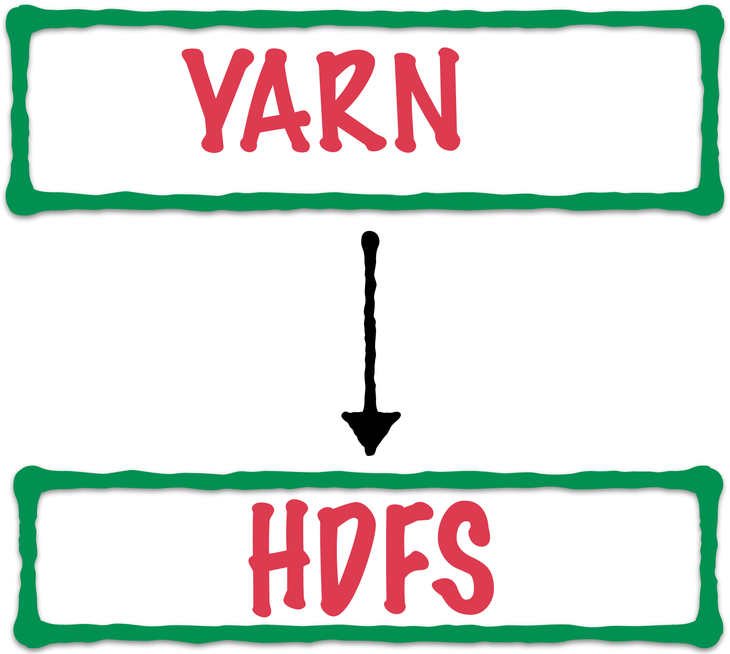
A job is triggered on the cluster



User defines map and reduce tasks using the MapReduce API

A job is triggered on the cluster

YARN figures out where and how to run the job, and stores the result in HDFS



YARN does this using 2 services

Resource manager

Node manager

Resource manager

There is 1 Resource Manager for a Hadoop cluster

Resource manager

The ResourceManager service runs on a single node - usually the same node as HDFS name node

Resource manager

The ResourceManager launches tasks that are submitted to YARN

Resource manager

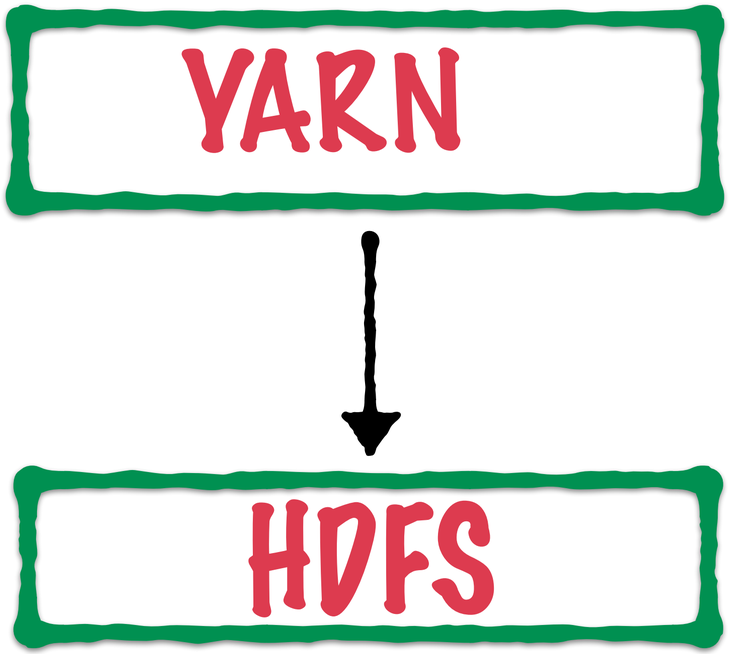
It arbitrates the available resources on the cluster among competing applications

Resource manager

It optimizes for cluster utilization based on constraints such as capacity guarantees, fairness and SLAs

Resource manager

It has a pluggable scheduler using which different scheduling policies can be utilized



YARN does this using 2 services

Resource manager

Node manager

Node manager

A NodeManager service runs on each node in the cluster i.e. all the data nodes

Node manager

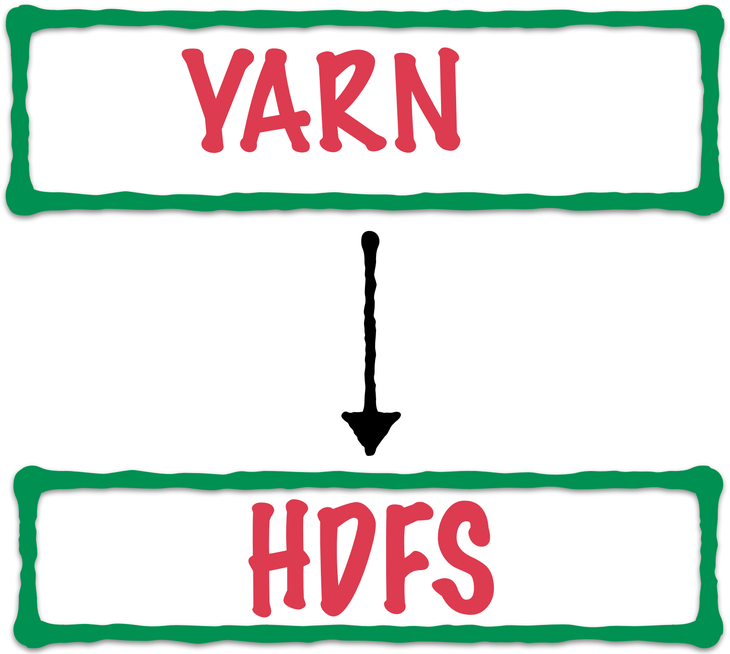
The NodeManager launches and monitors all tasks running on that node

Node manager

It coordinates with the ResourceManager in order to perform its tasks

Node manager

It monitors resources, logs, tracks the health of the node etc. everything related to the one node that is in its charge

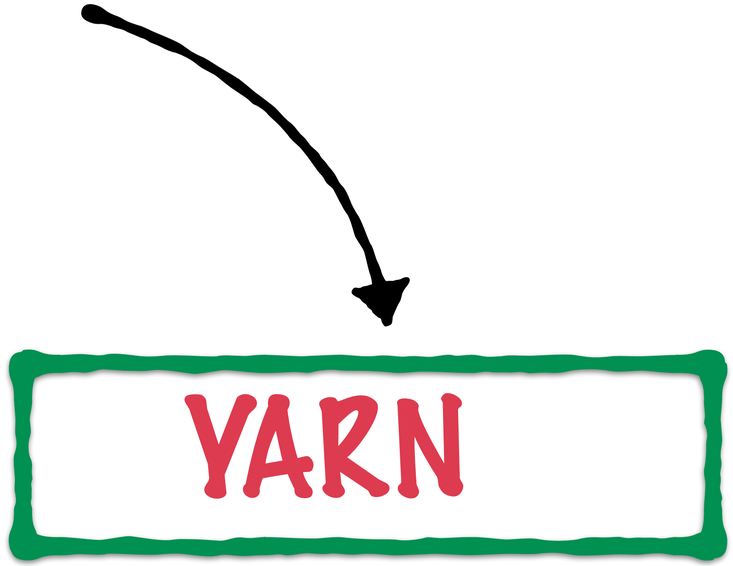


Let's see how these services co-ordinate tasks

Resource manager ✓

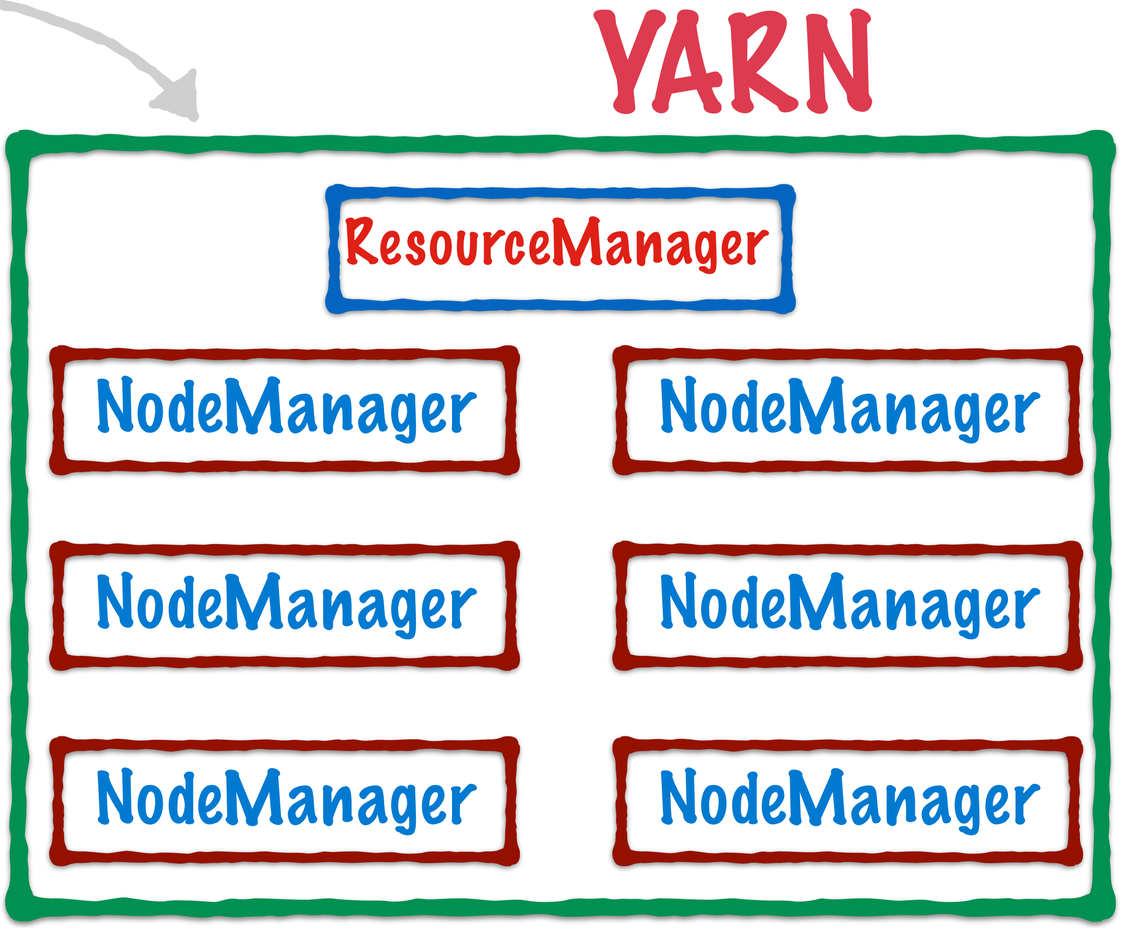
Node manager ✓

A job is submitted to YARN



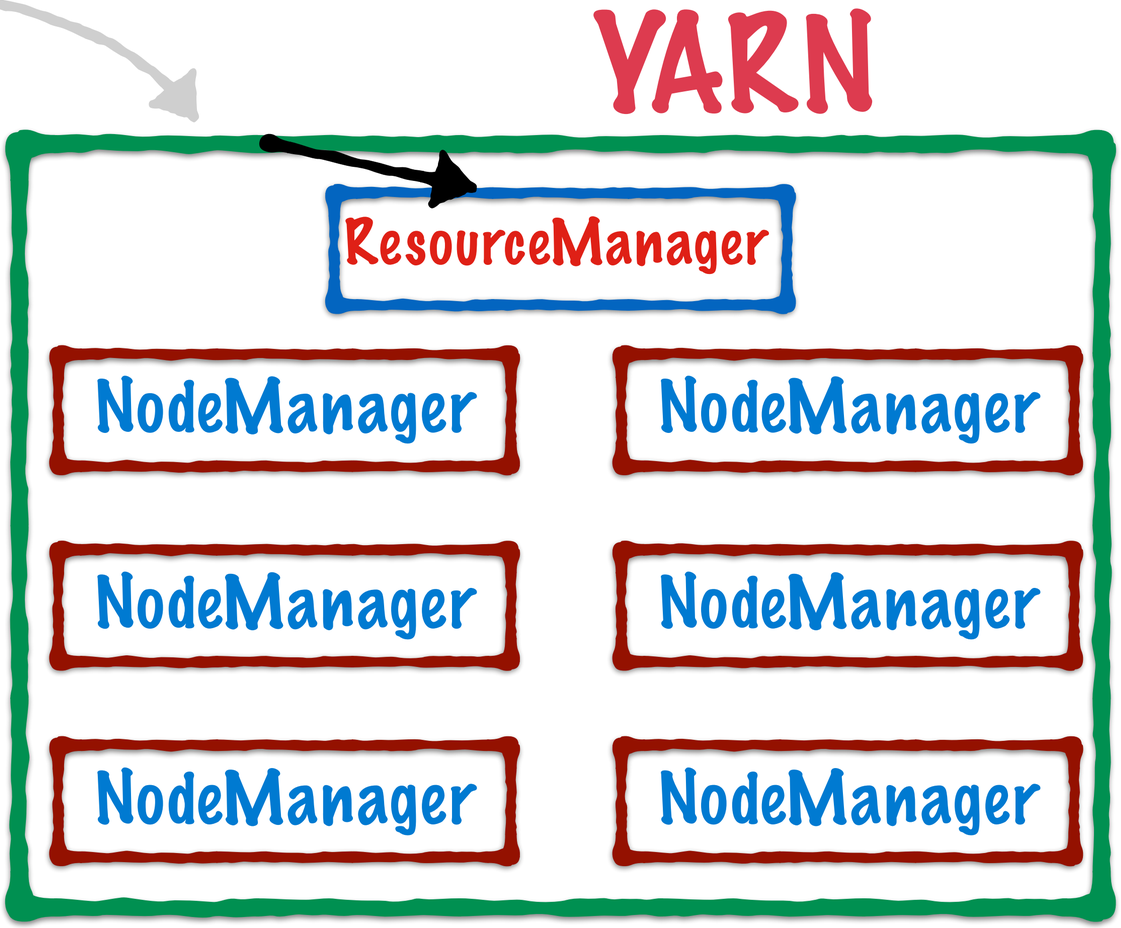
A job is submitted to YARN

YARN has a ResourceManager running on 1 node and NodeManagers on all others



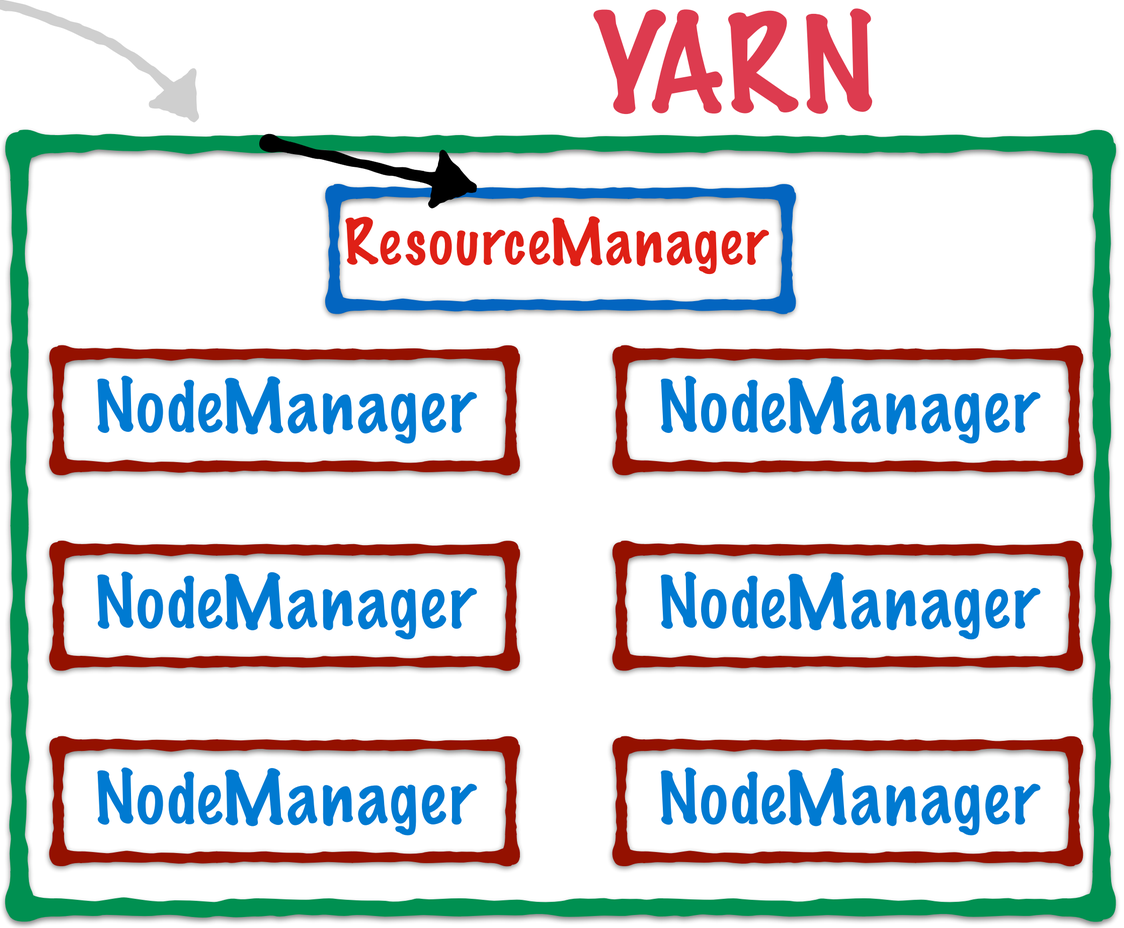
A job is submitted to YARN

The job is first submitted to the ResourceManager



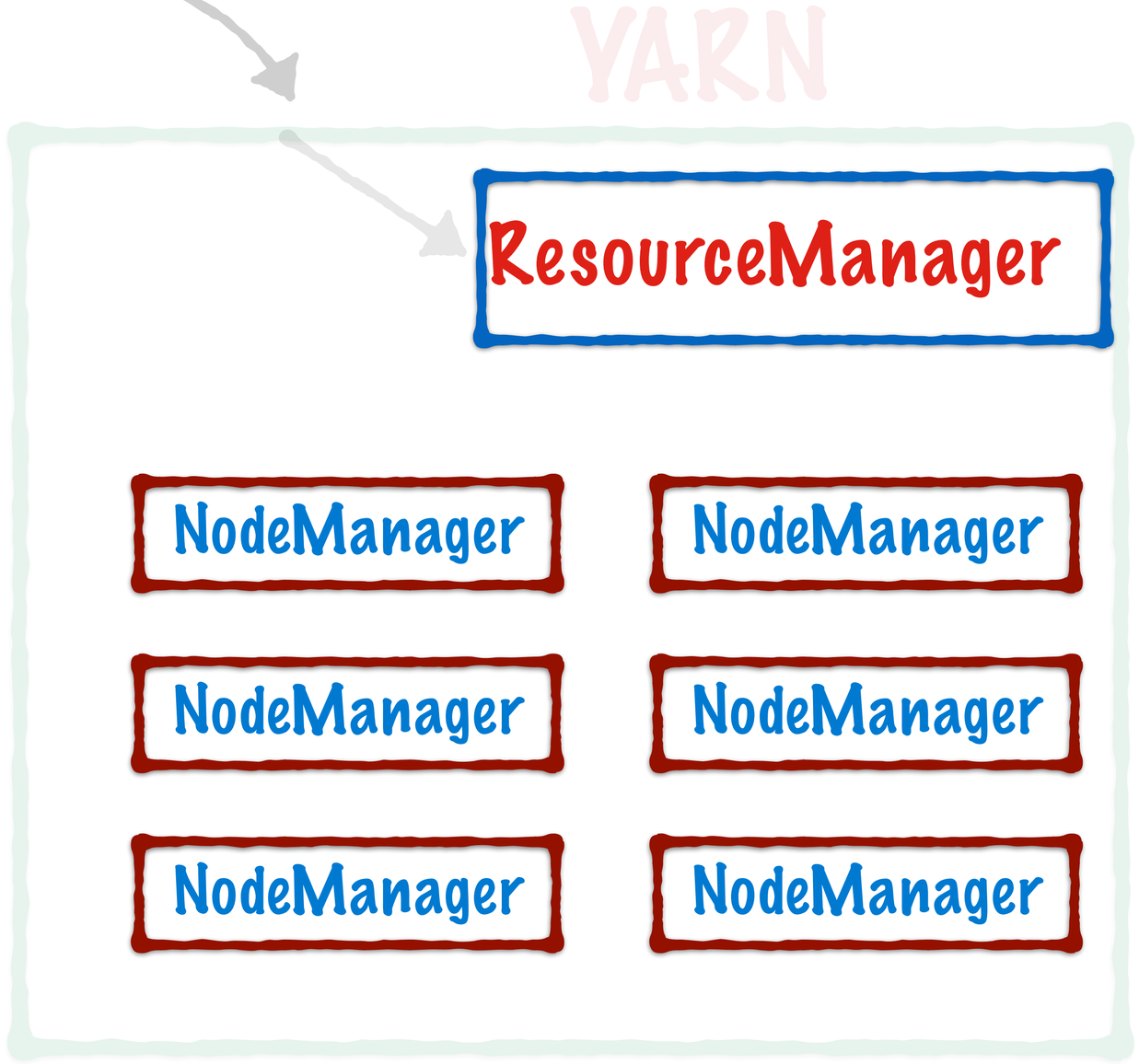
A job is submitted to YARN

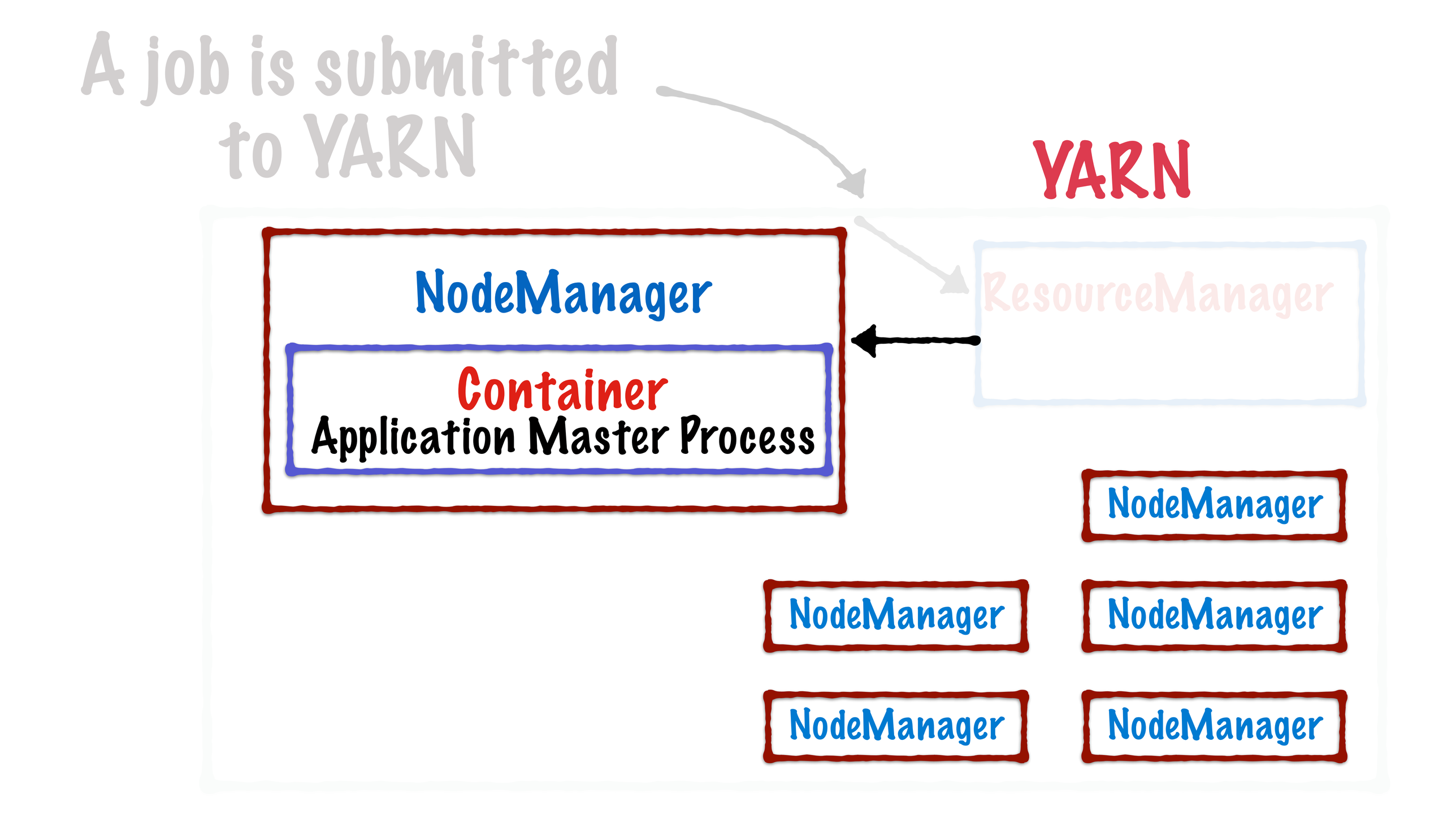
The ResourceManager will schedule the job based on the constraints specified and capacity available



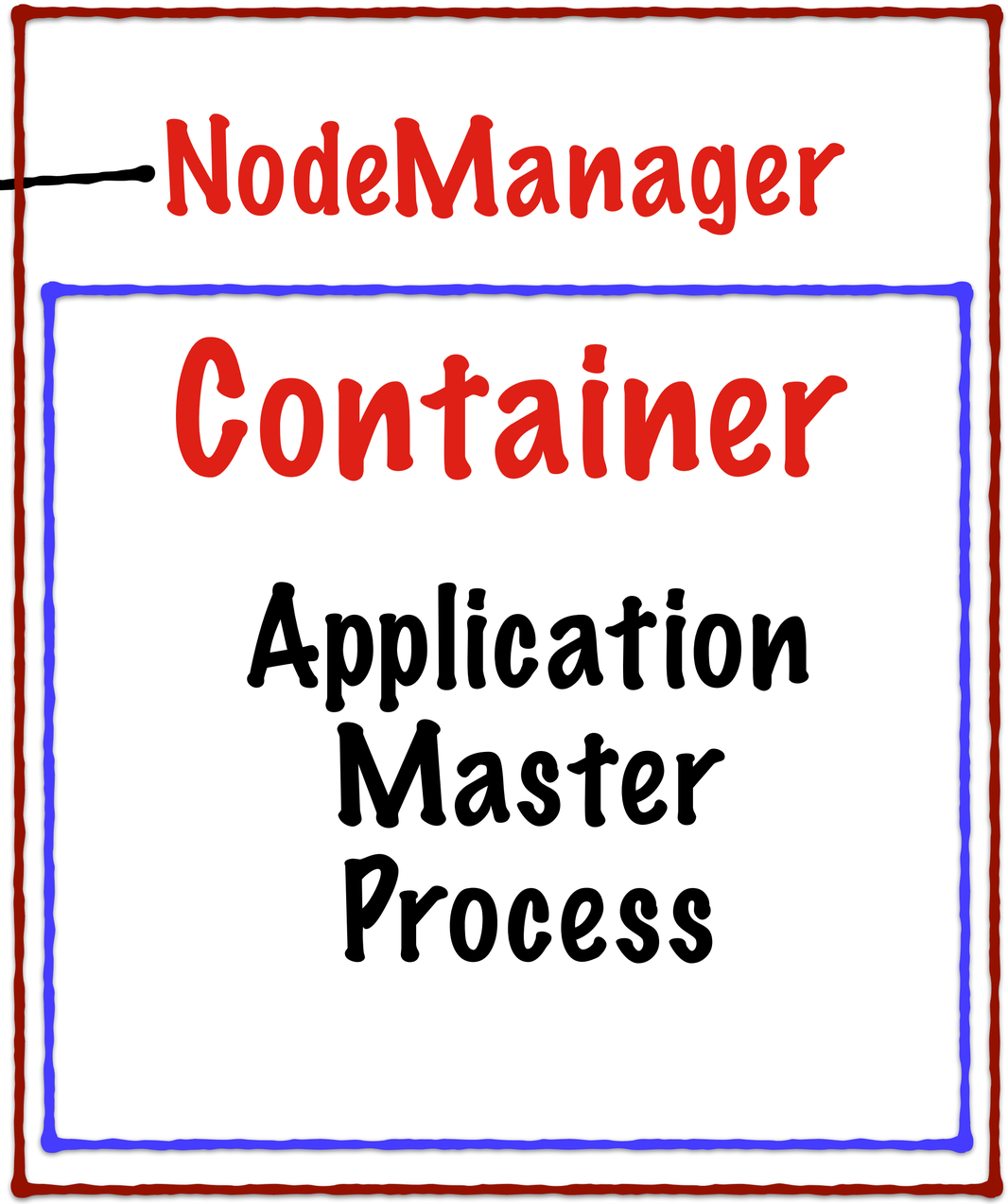
A job is submitted to YARN

The Resource Manager will in turn find a Node Manager on one of the nodes to launch an Application Master Process

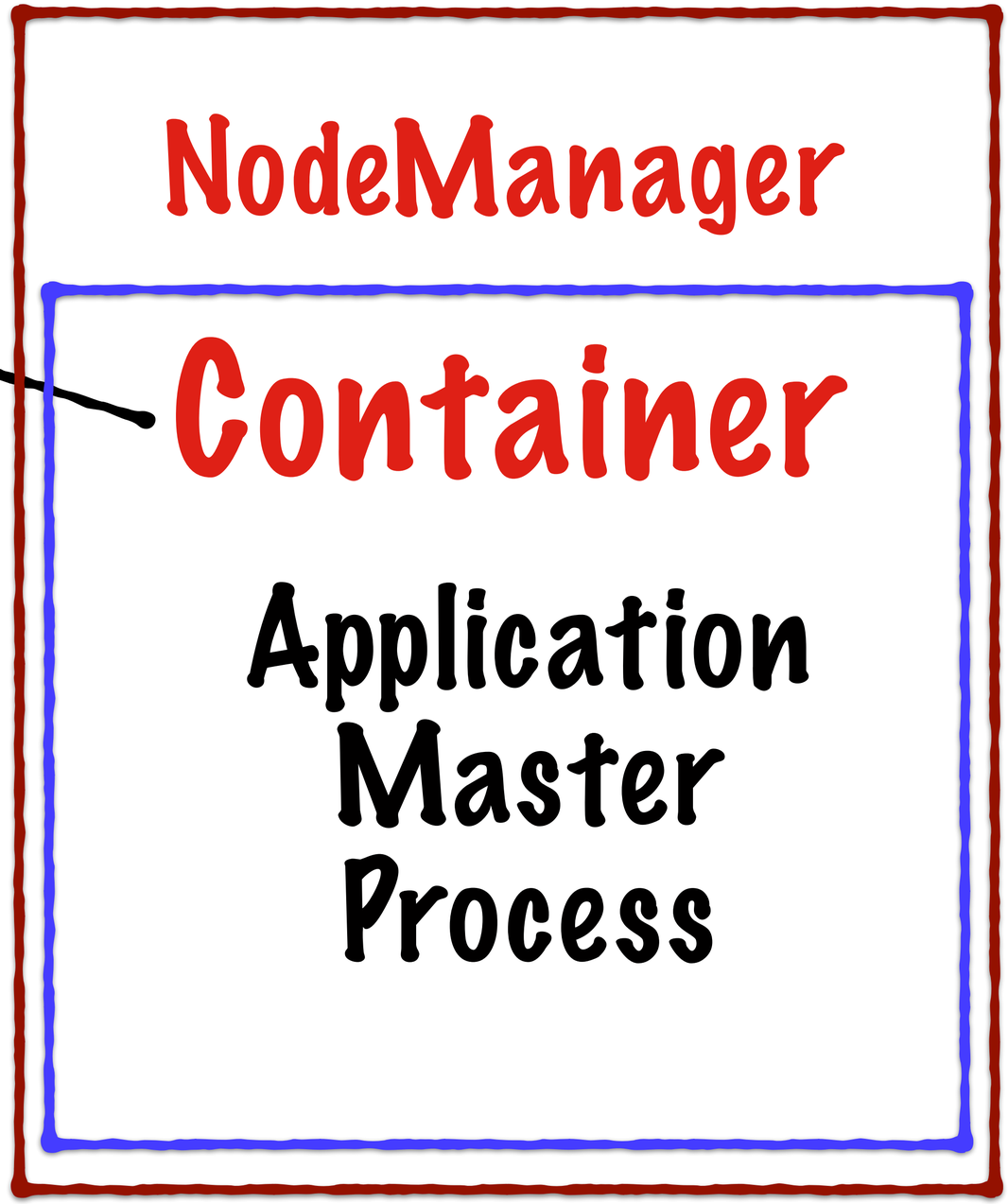



A job is submitted to YARN
YARN
NodeManager
Container
Application Master Process
ResourceManager
NodeManager
NodeManager
NodeManager
NodeManager
NodeManager
NodeManager

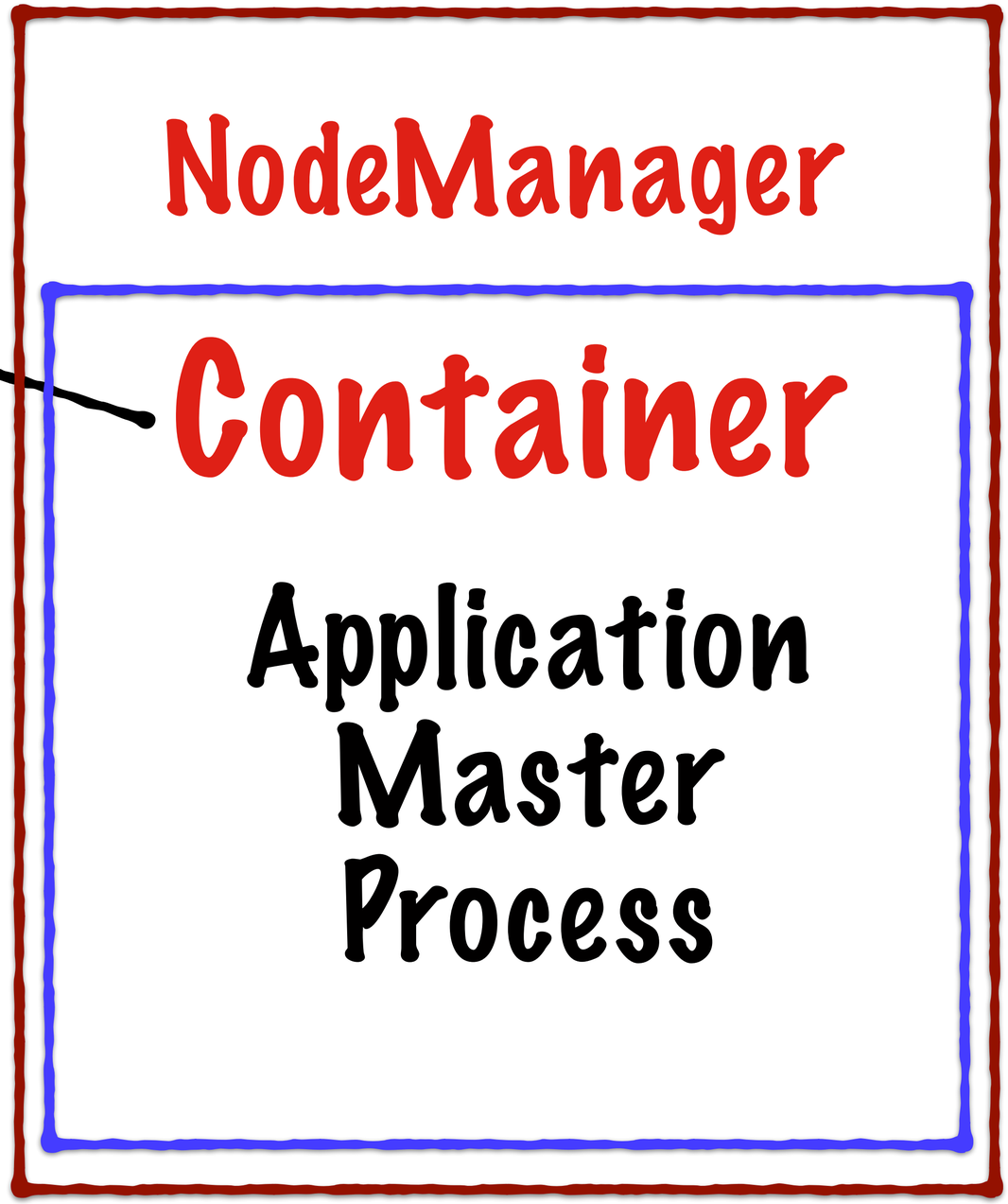
This NodeManager is located on some node that has the necessary resources to process the task



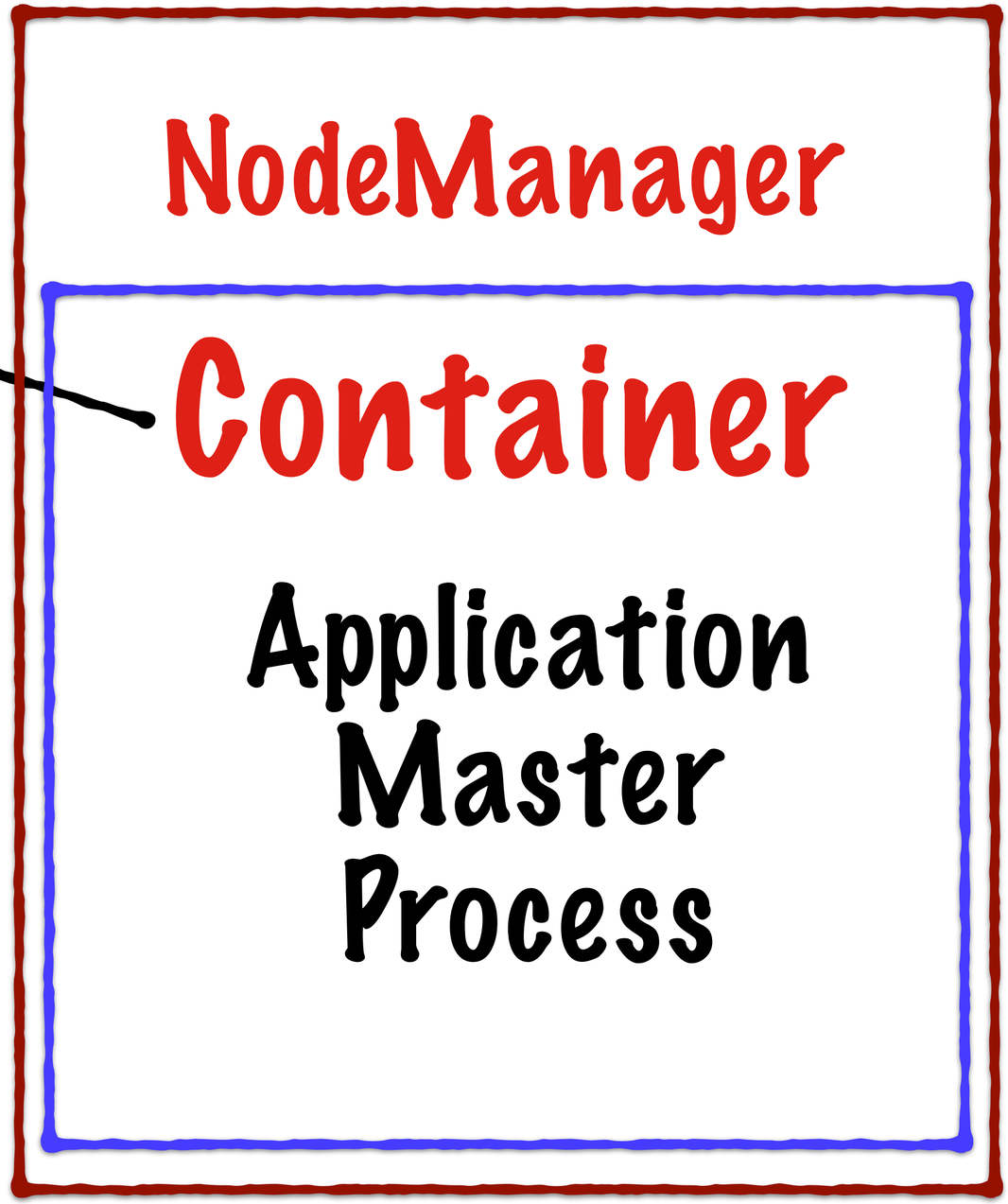
A container is the resource that is allocated to a specific task/ application



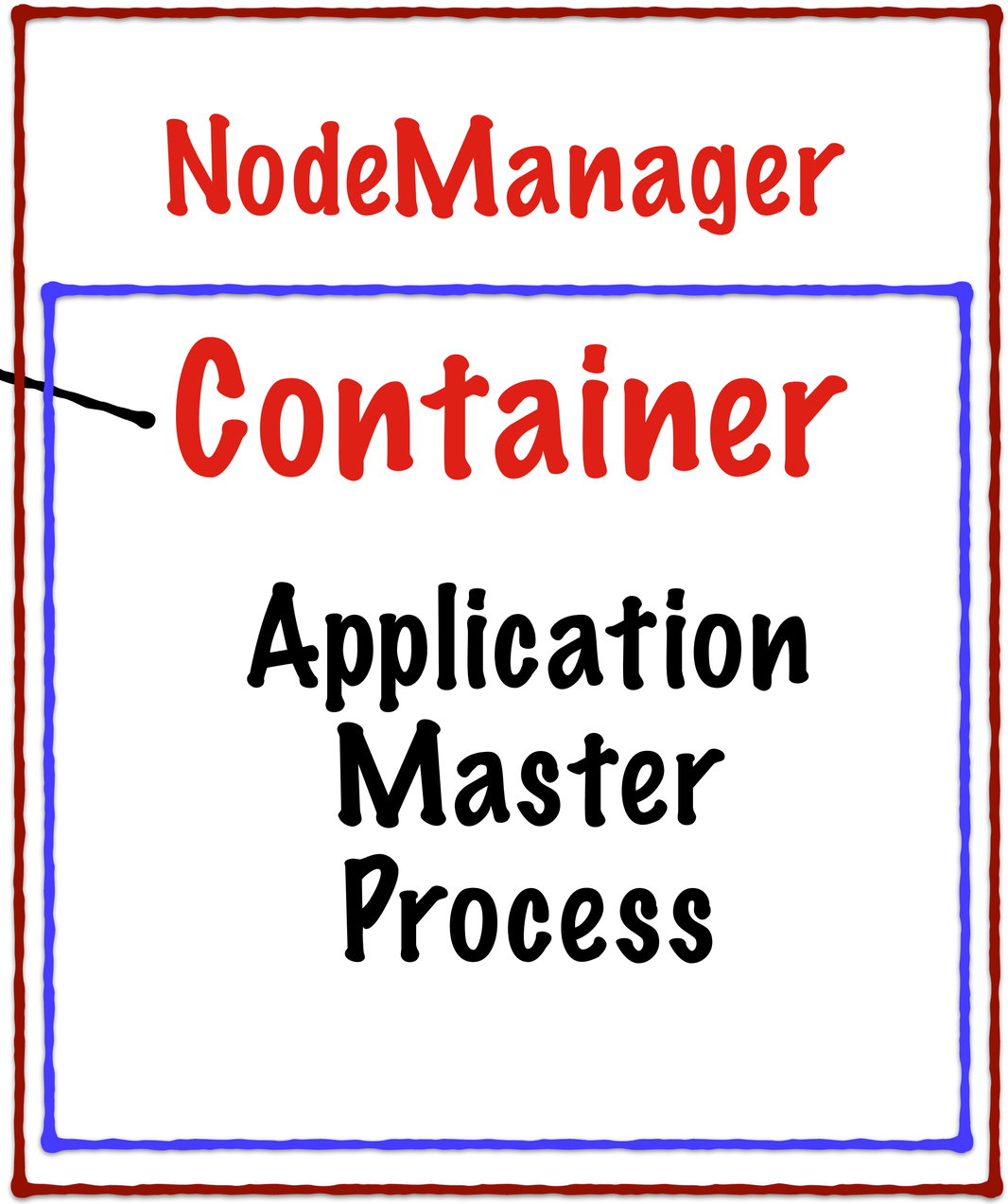
A Container grants the rights to an application to use a specified amount of resources on a specific host



The actual resources specified by the container can only be allocated by the NodeManager

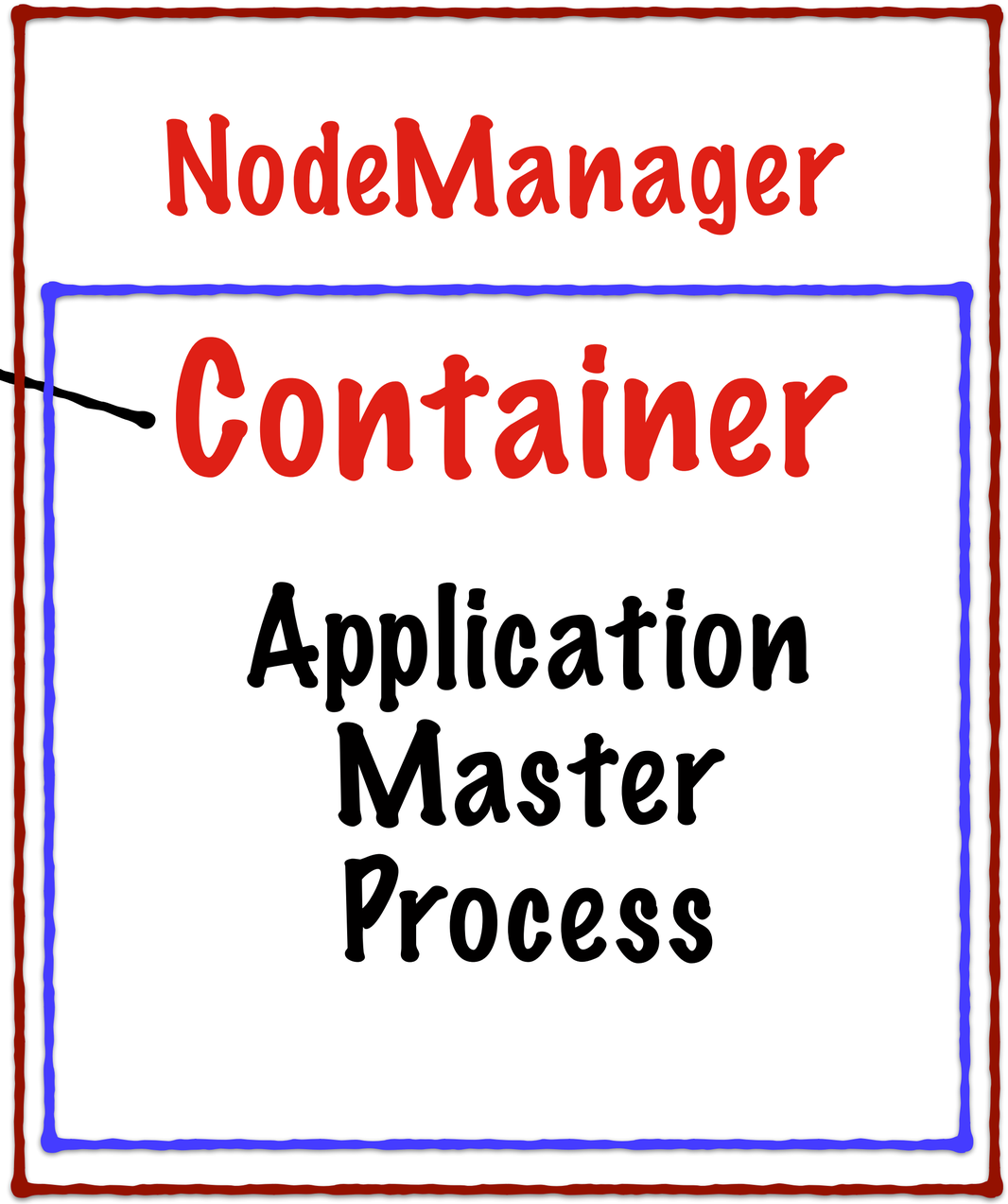


A NodeManager could be managing many containers, 1 for each task



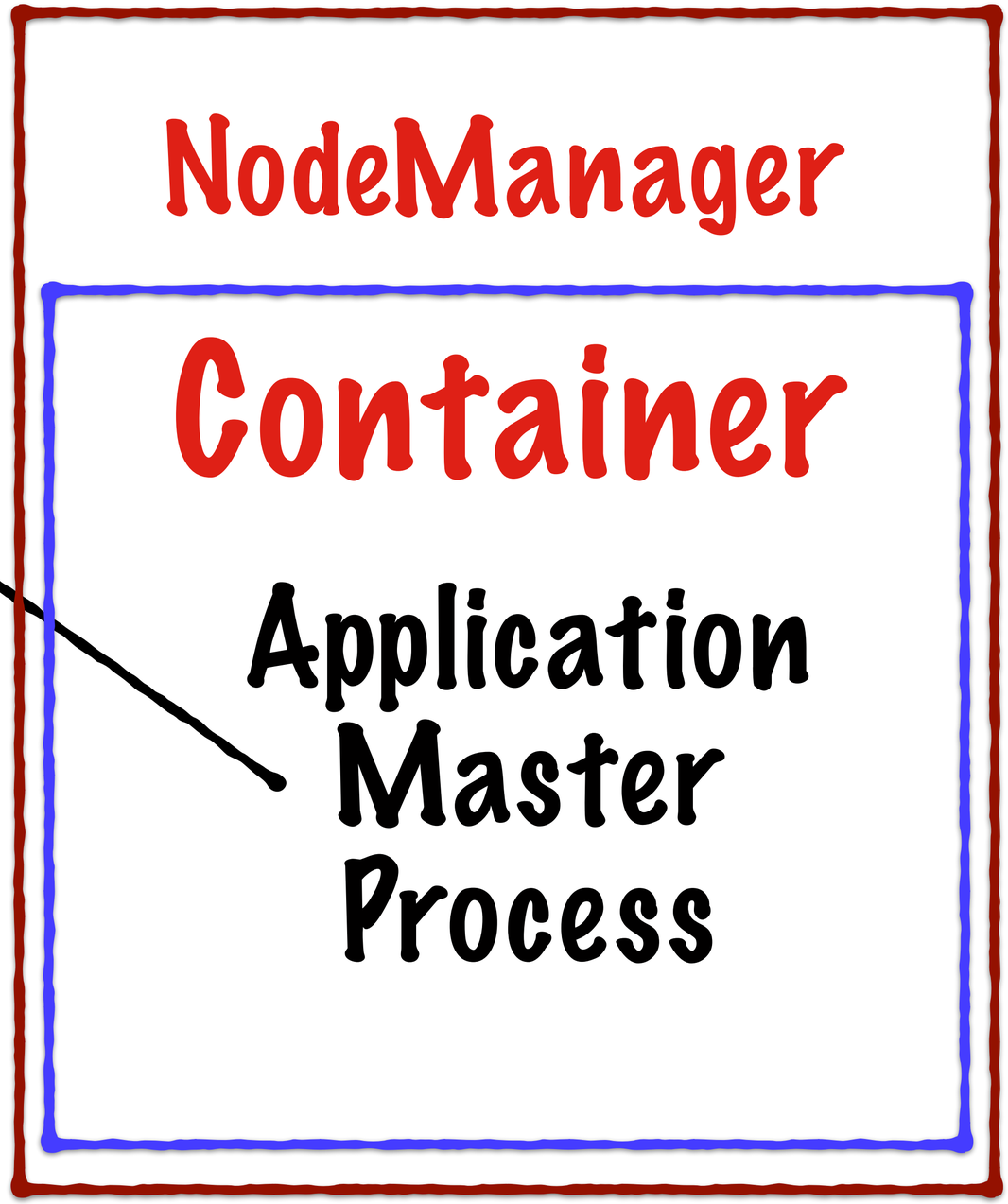
Each container has a specific amount of resources allocated to it - memory, CPU etc

Resources are allocated by the NodeManager

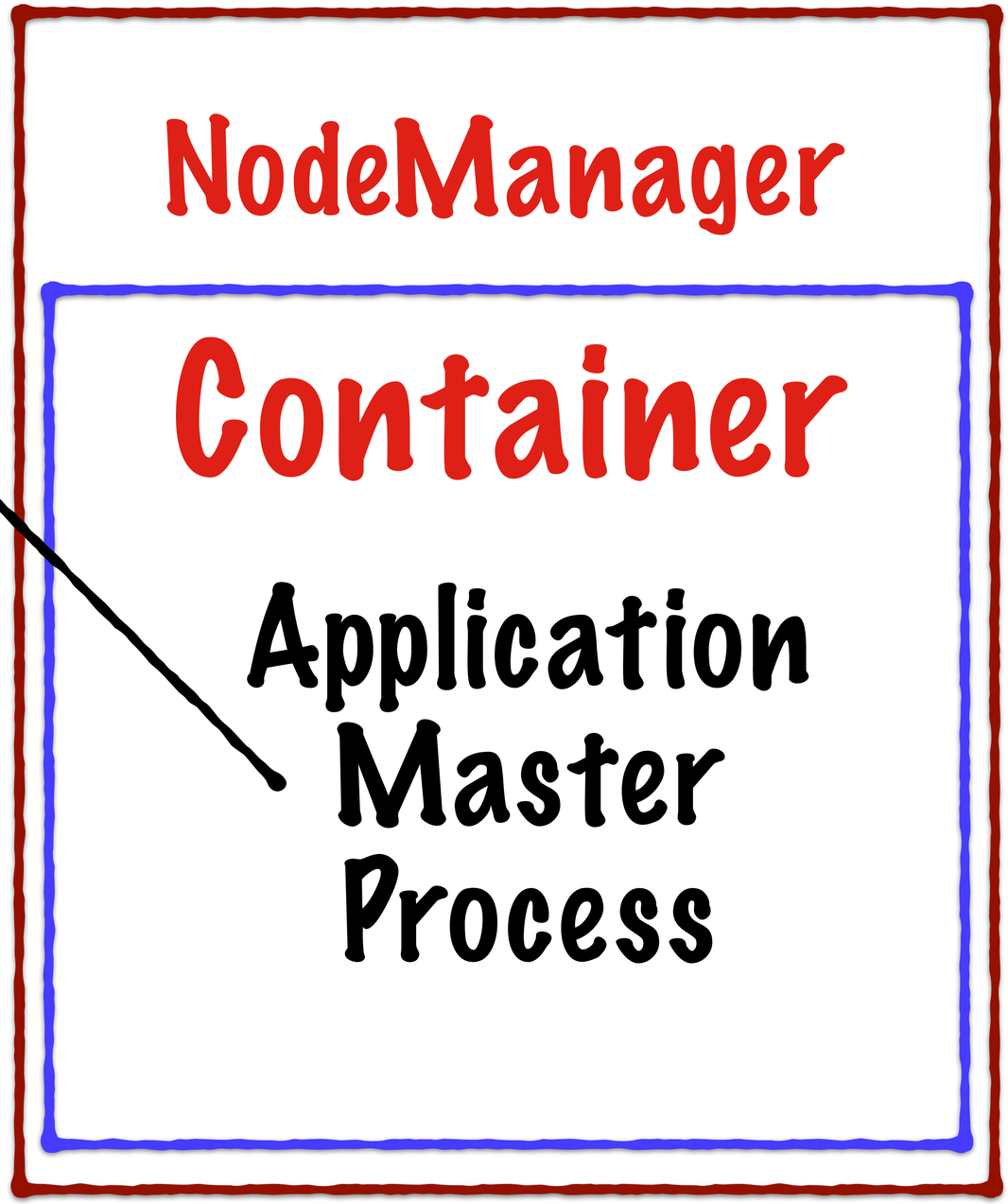


This does a bunch of stuff:

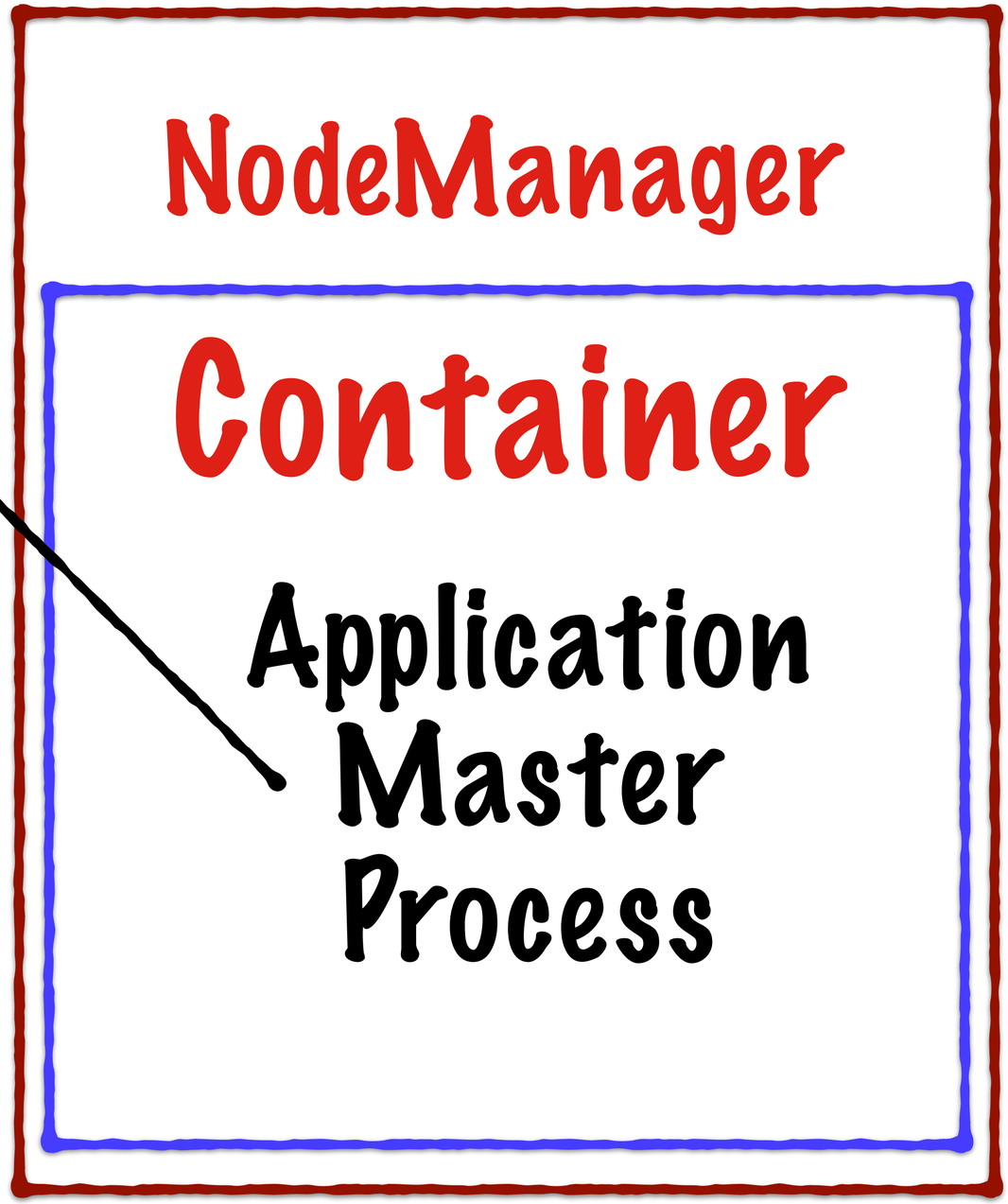
1. Negotiates resources from the ResourceManager
2. Works with the NodeManagers to execute and monitor containers



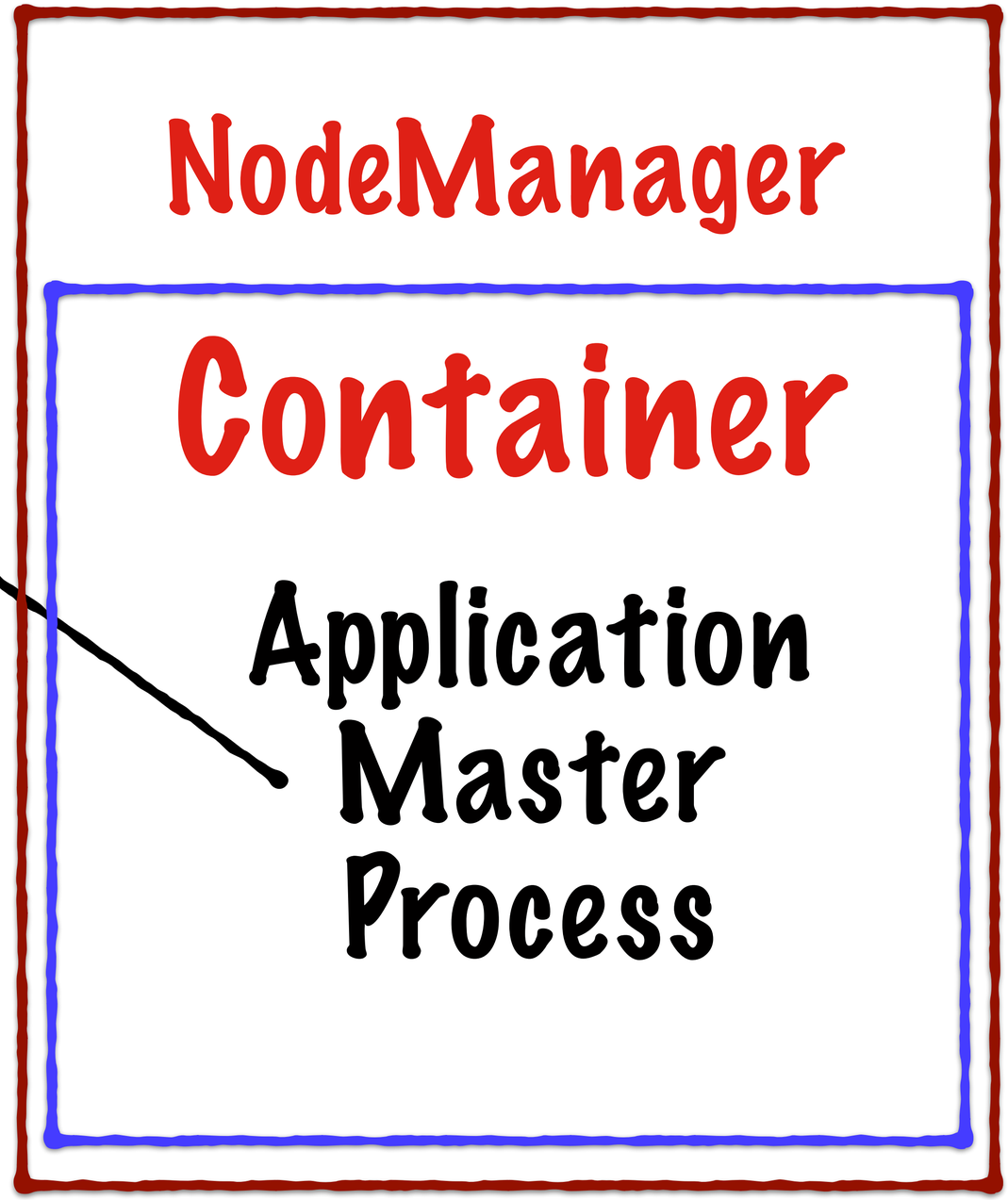
Moving responsibility for the monitoring and execution of tasks to the ApplicationMaster allows scale in computing



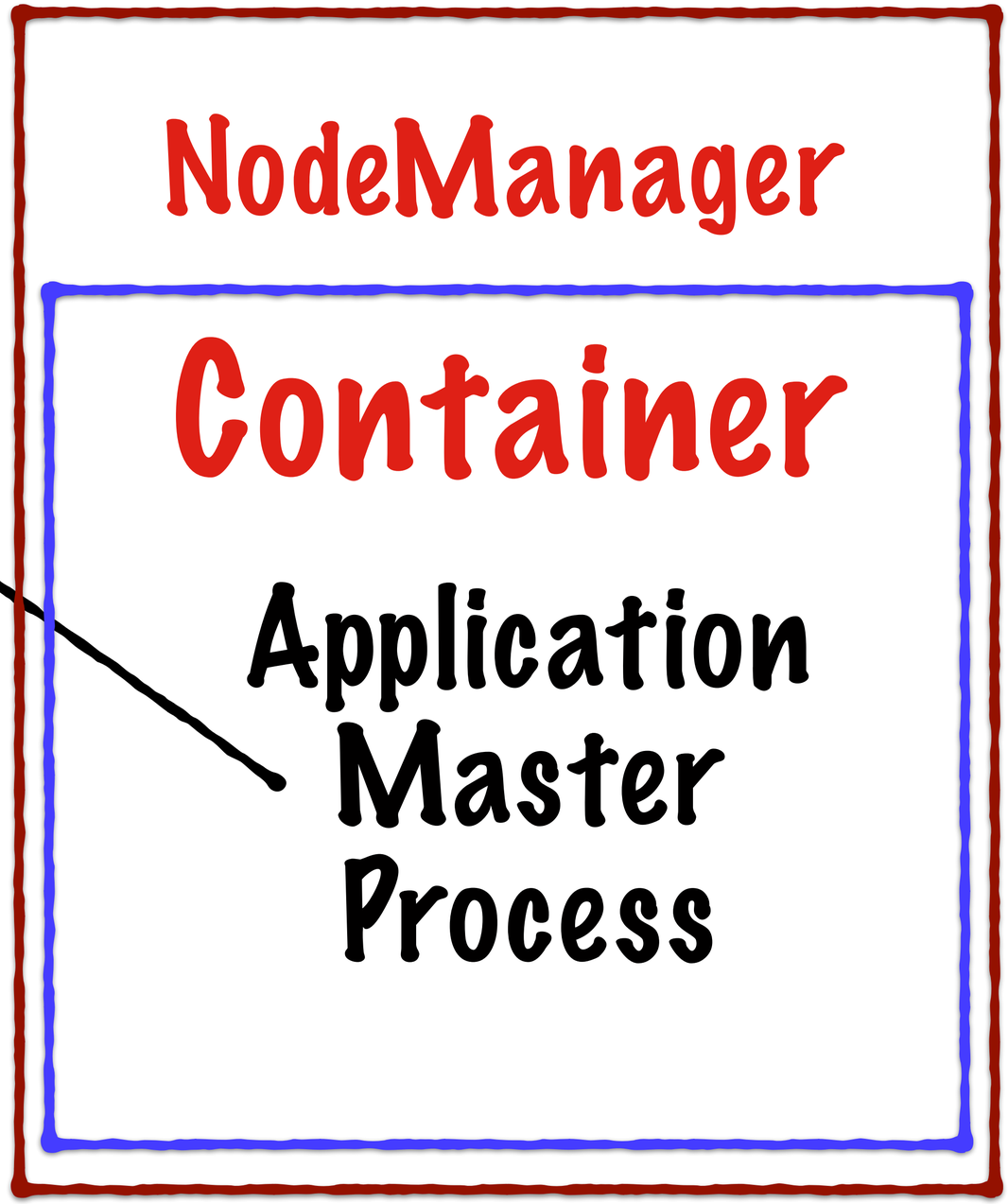
The ApplicationMaster is per application so is not a bottle neck for the infrastructure as a whole



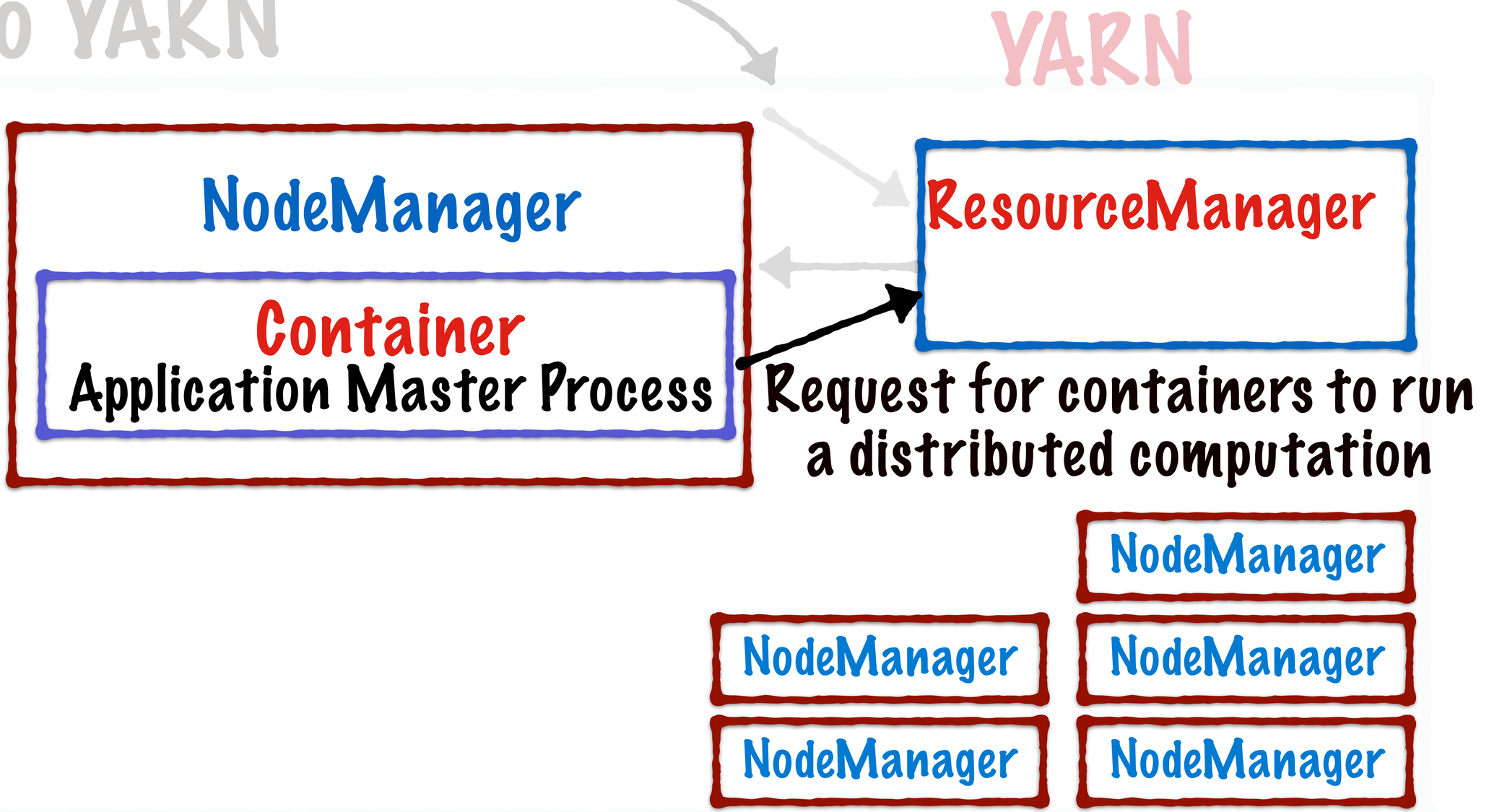
It has to perform the task with the resources allocated to the container



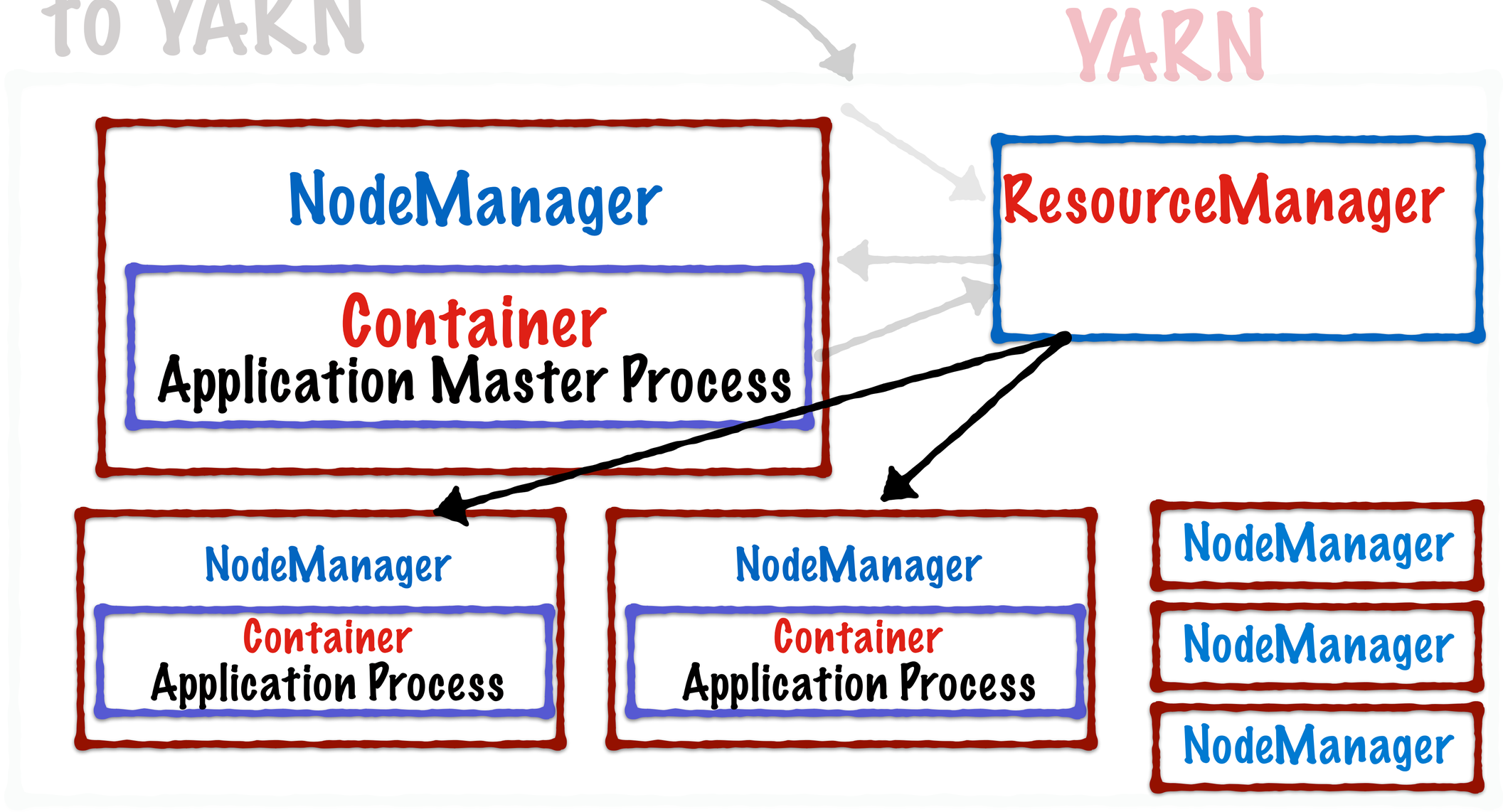
In case we are running a distributed computation - like MapReduce, this process will launch requests for more containers on other nodes



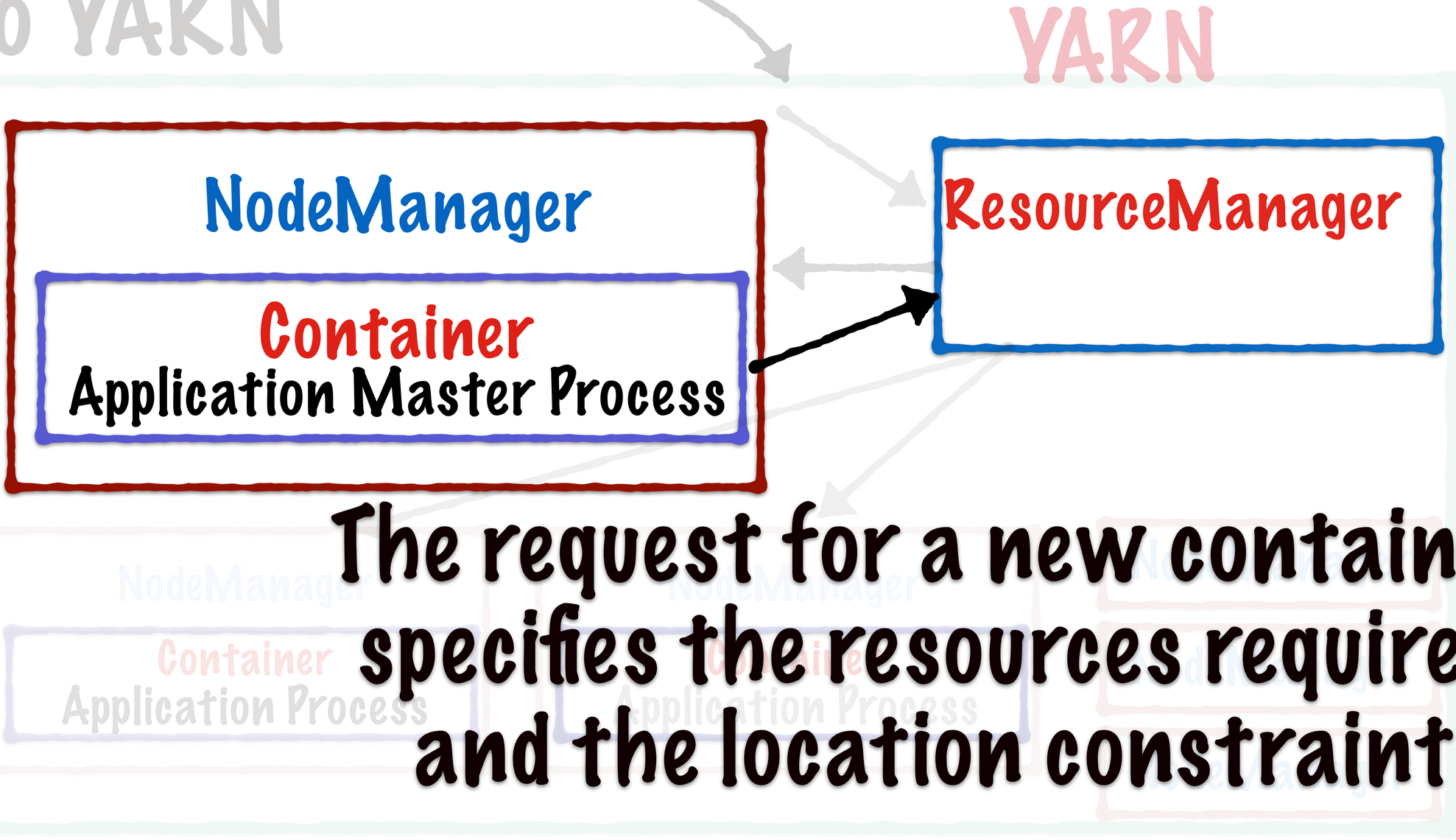
A job is submitted to YARN



A job is submitted to YARN

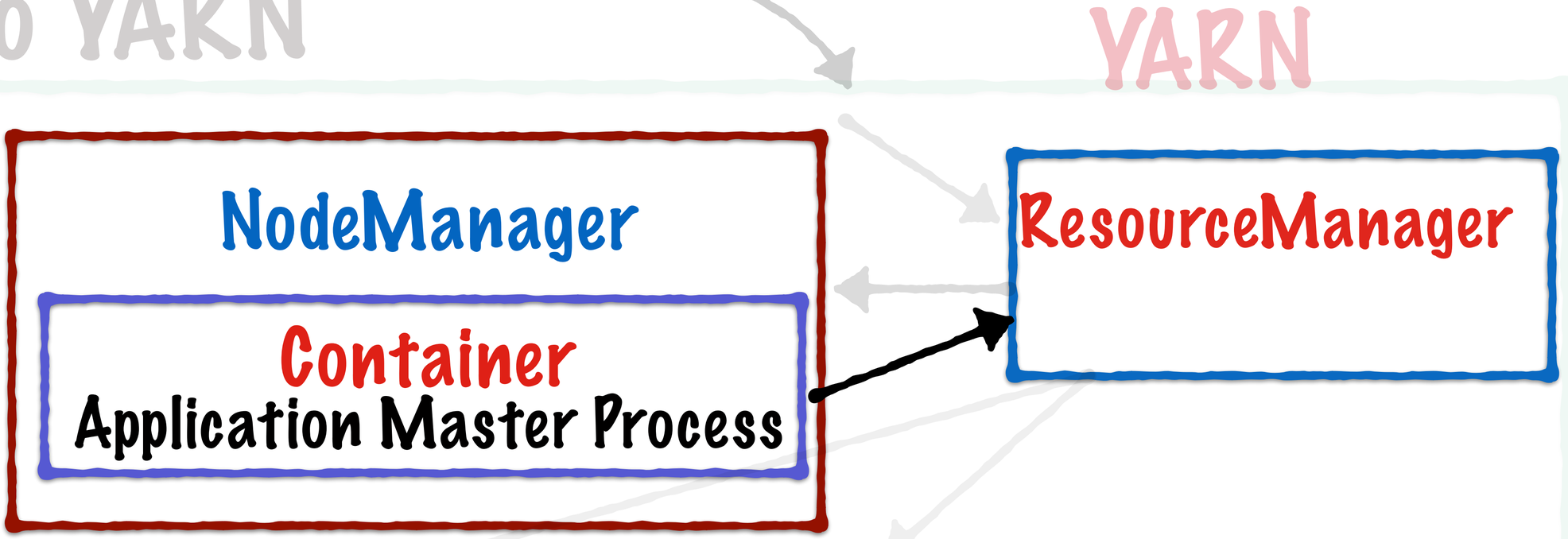


A job is submitted to YARN



The request for a new container specifies the resources required and the location constraint

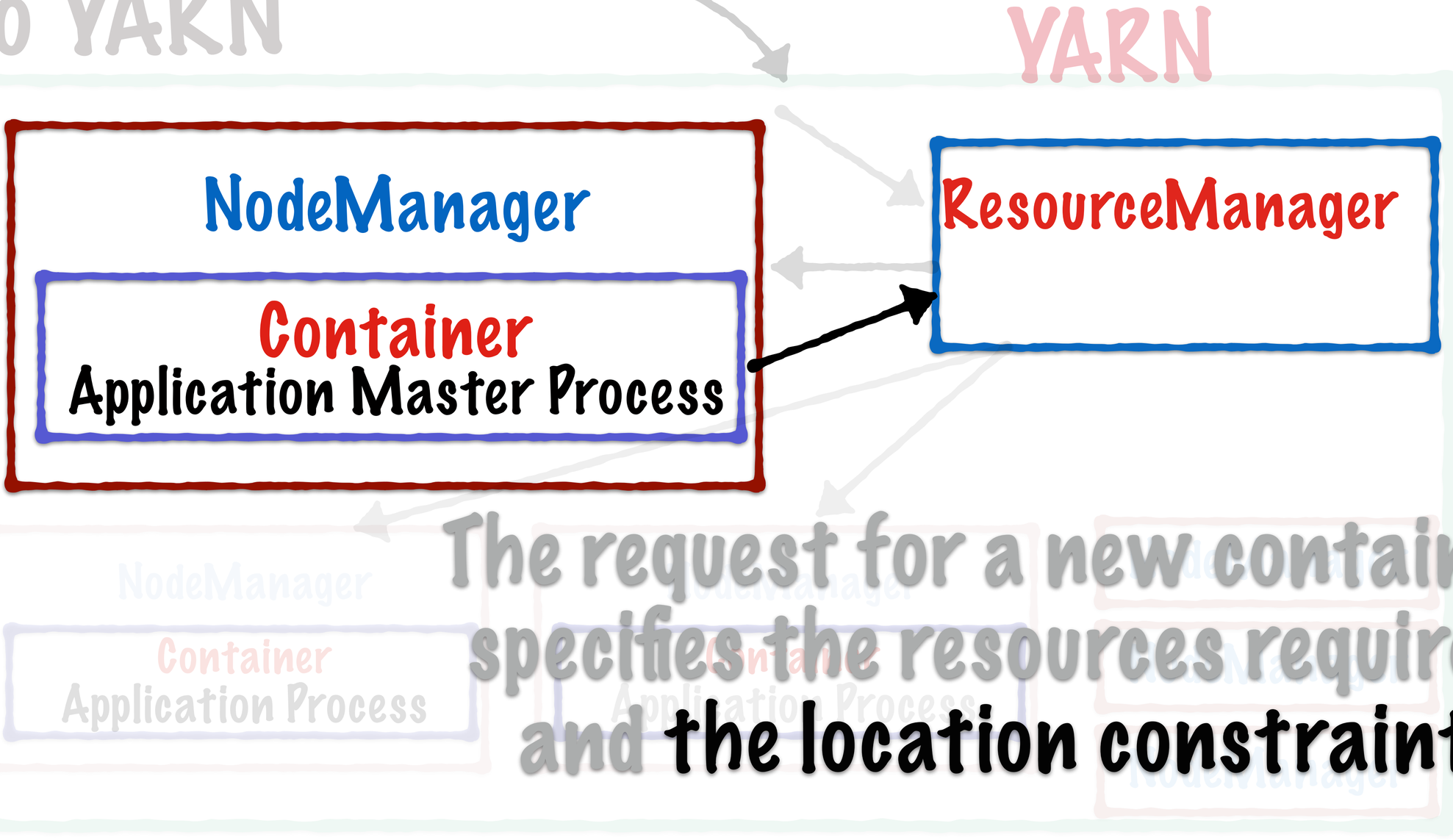
A job is submitted to YARN



Memory, CPU requirements

The request for a new container specifies the resources required and the location constraint

A job is submitted to YARN



The request for a new container specifies the resources required and the location constraint

the location constraint

In order to conserve network bandwidth, YARN will try to perform the computation on the same node as the data resides

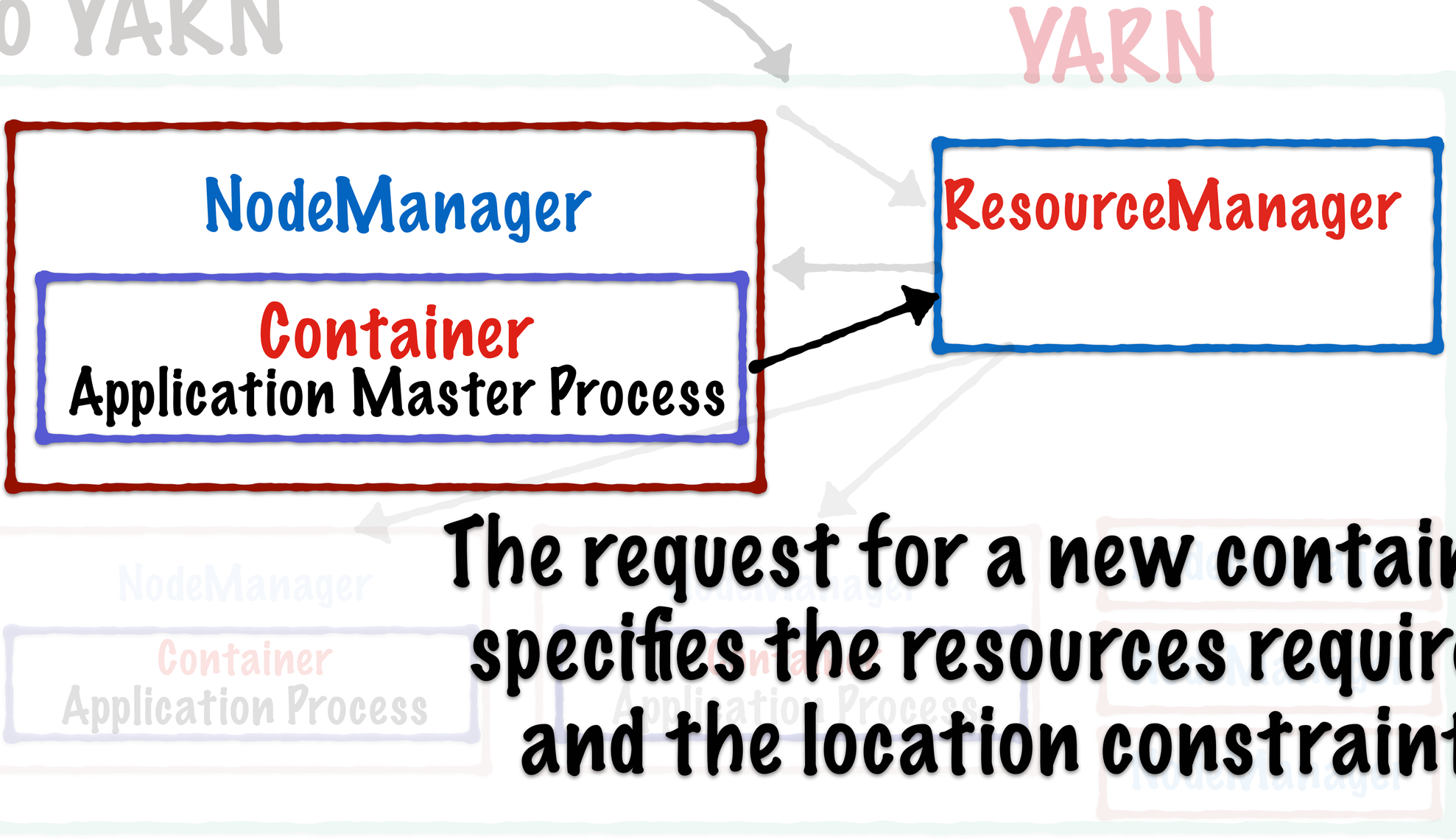
the location constraint

If there are no resources available on the data node, then YARN will first wait for some time for resources to free up

the location constraint

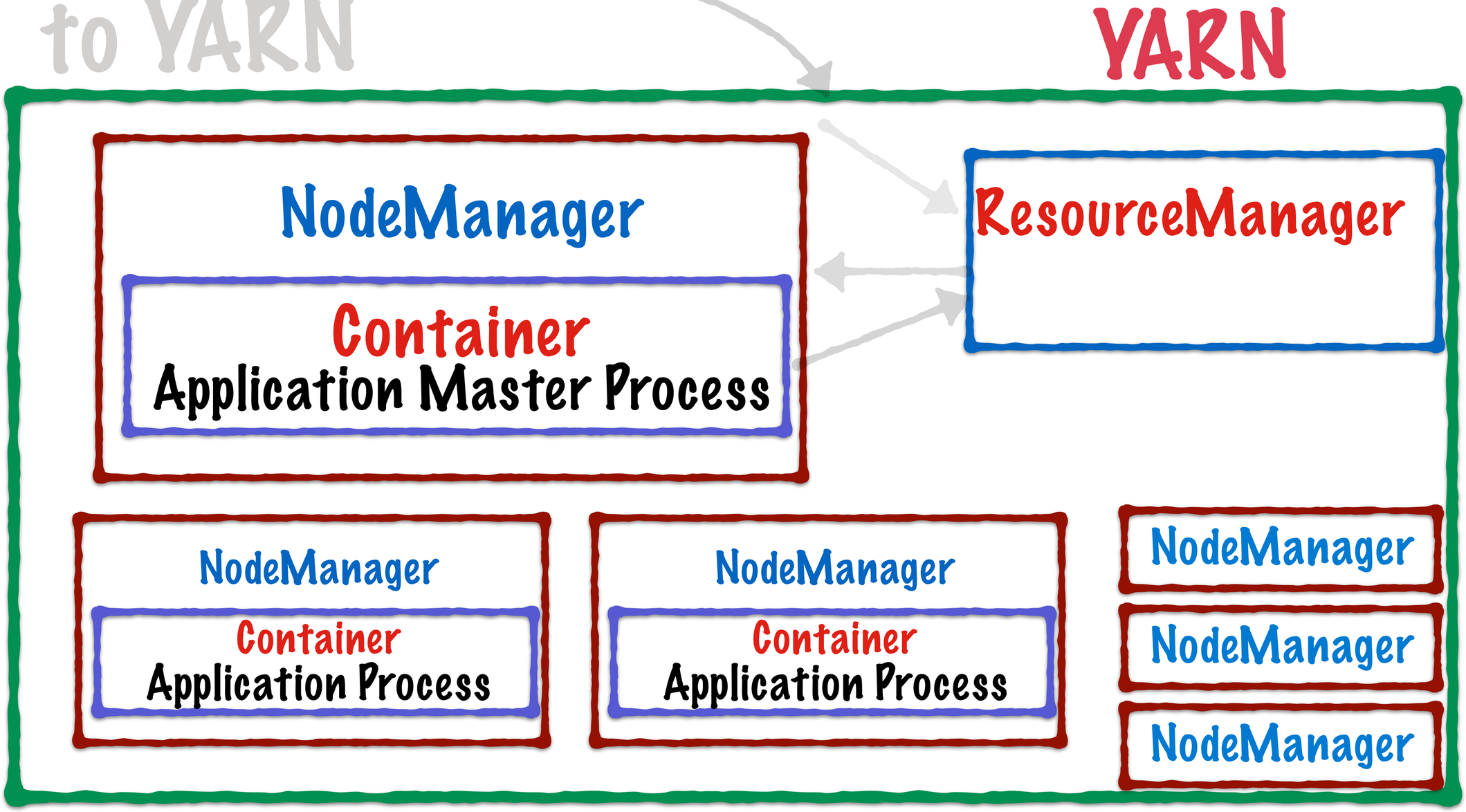
If that fails, the container is requested on the same rack as the concerned data node

A job is submitted to YARN



The request for a new container specifies the resources required and the location constraint

A job is submitted to YARN



Scheduling in YARN

Scheduling in YARN

YARN manages the resources which are allocated to different tasks

Scheduling in YARN

When there are multiple resource requests, YARN follows defined policies to allocate resources

Scheduling in YARN

There are 3 scheduling policies available

- FIFO Scheduler
- Capacity Scheduler
- Fair Scheduler

Scheduling in YARN

FIFO Scheduler

Resources are allocated on First In First Out basis

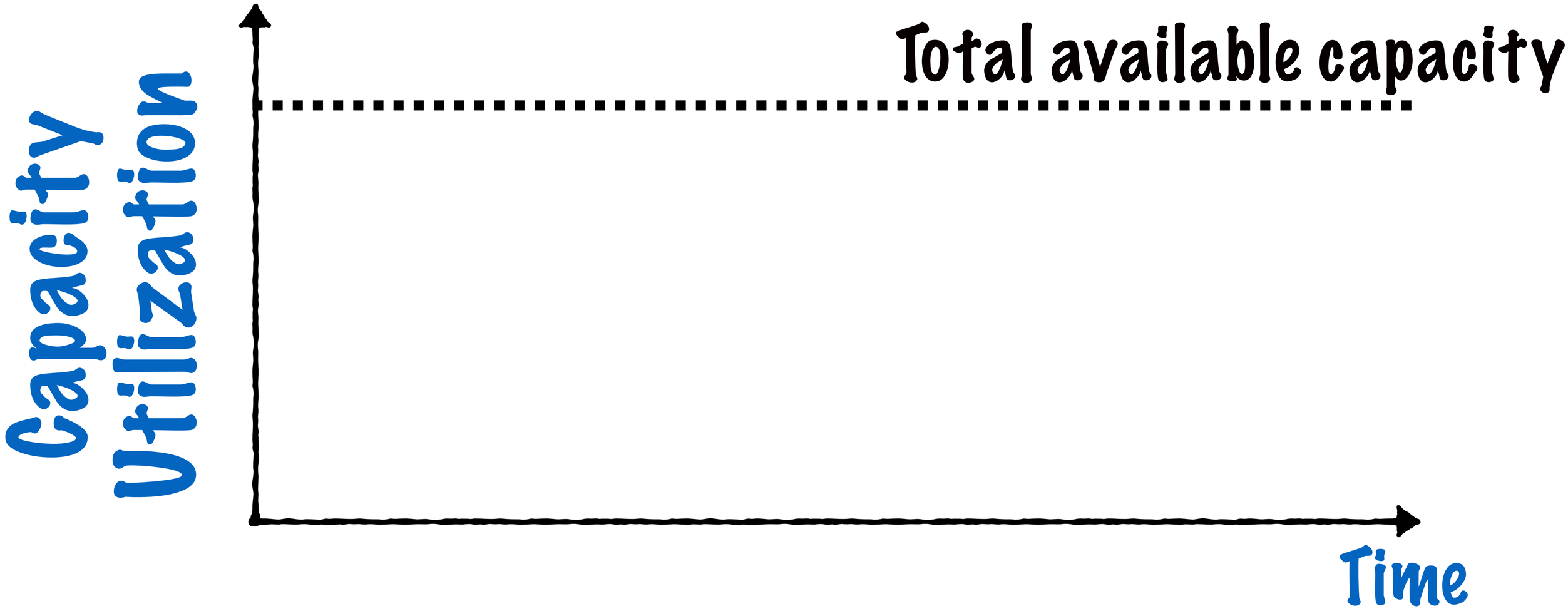
Scheduling in YARN

FIFO Scheduler

Requests made first are satisfied first before resources are allocated to other

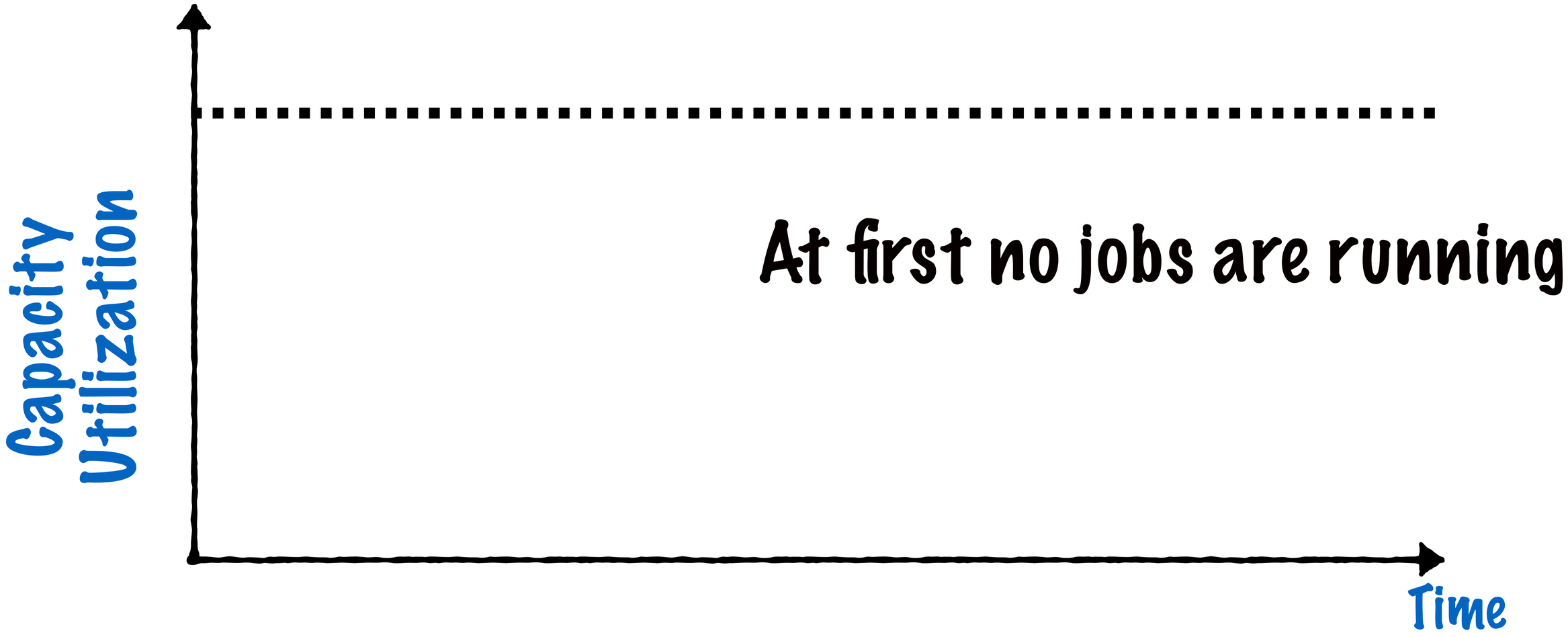
Scheduling in YARN

FIFO Scheduler



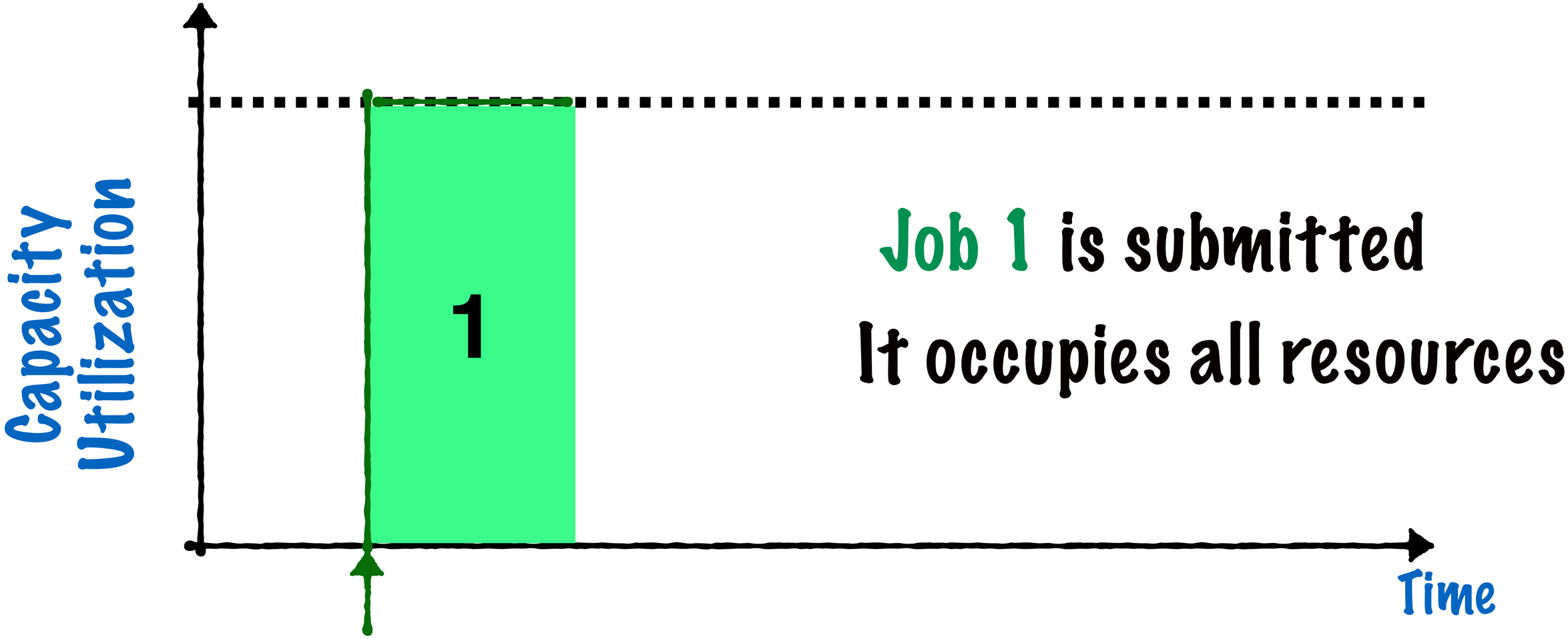
Scheduling in YARN

FIFO Scheduler



Scheduling in YARN

FIFO Scheduler

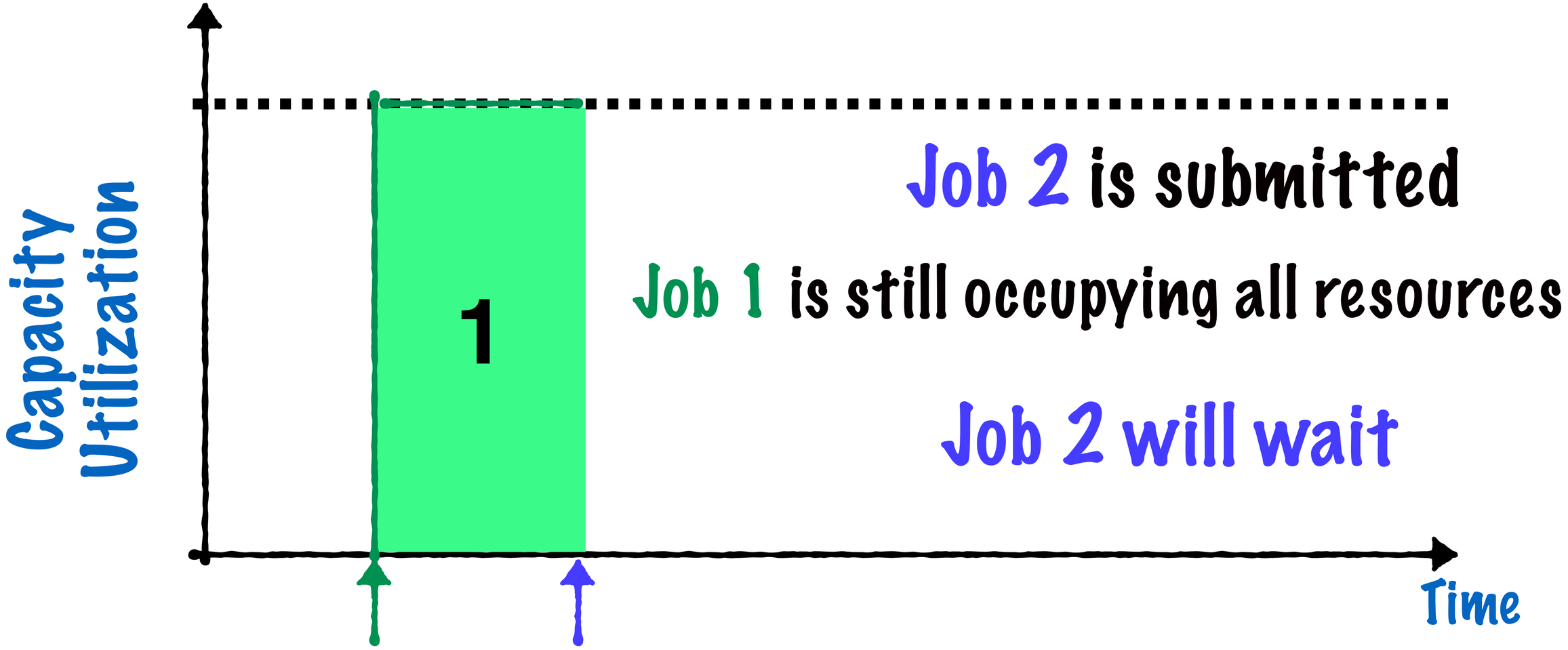


Job 1 is submitted

It occupies all resources

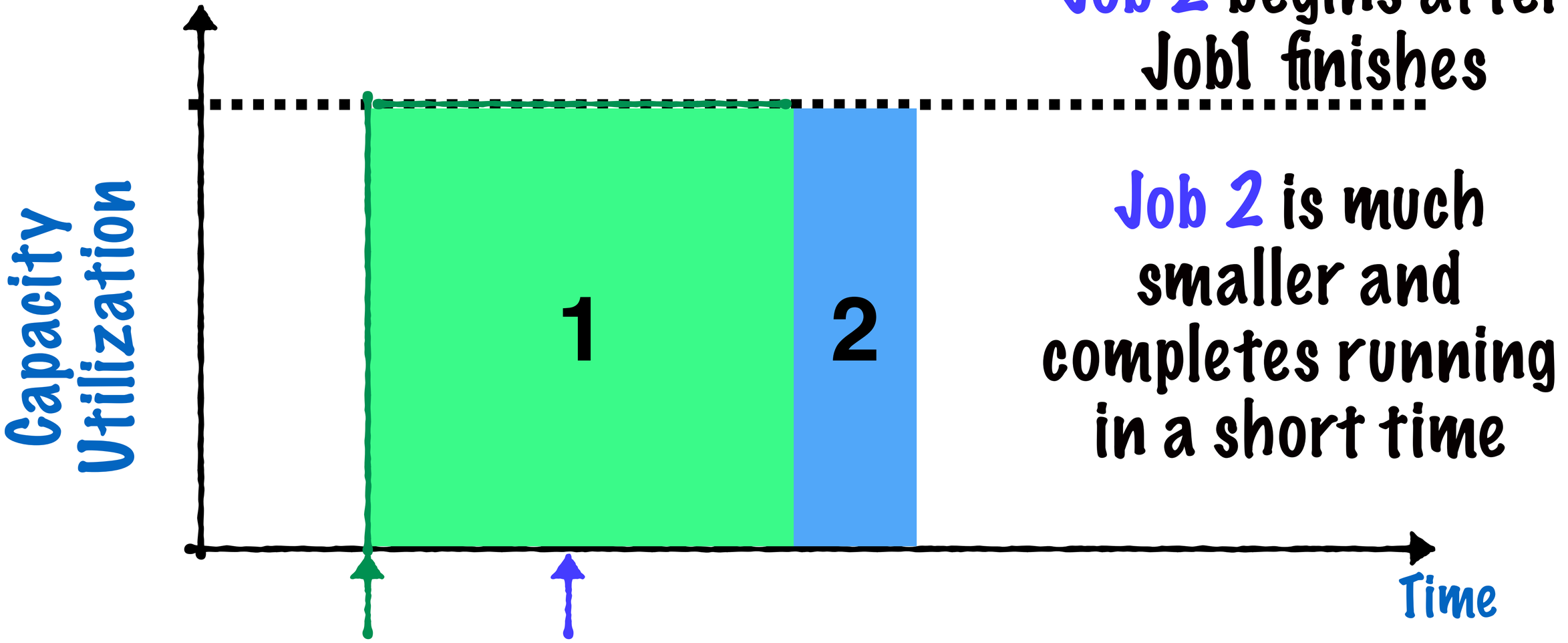
Scheduling in YARN

FIFO Scheduler



Scheduling in YARN

FIFO Scheduler

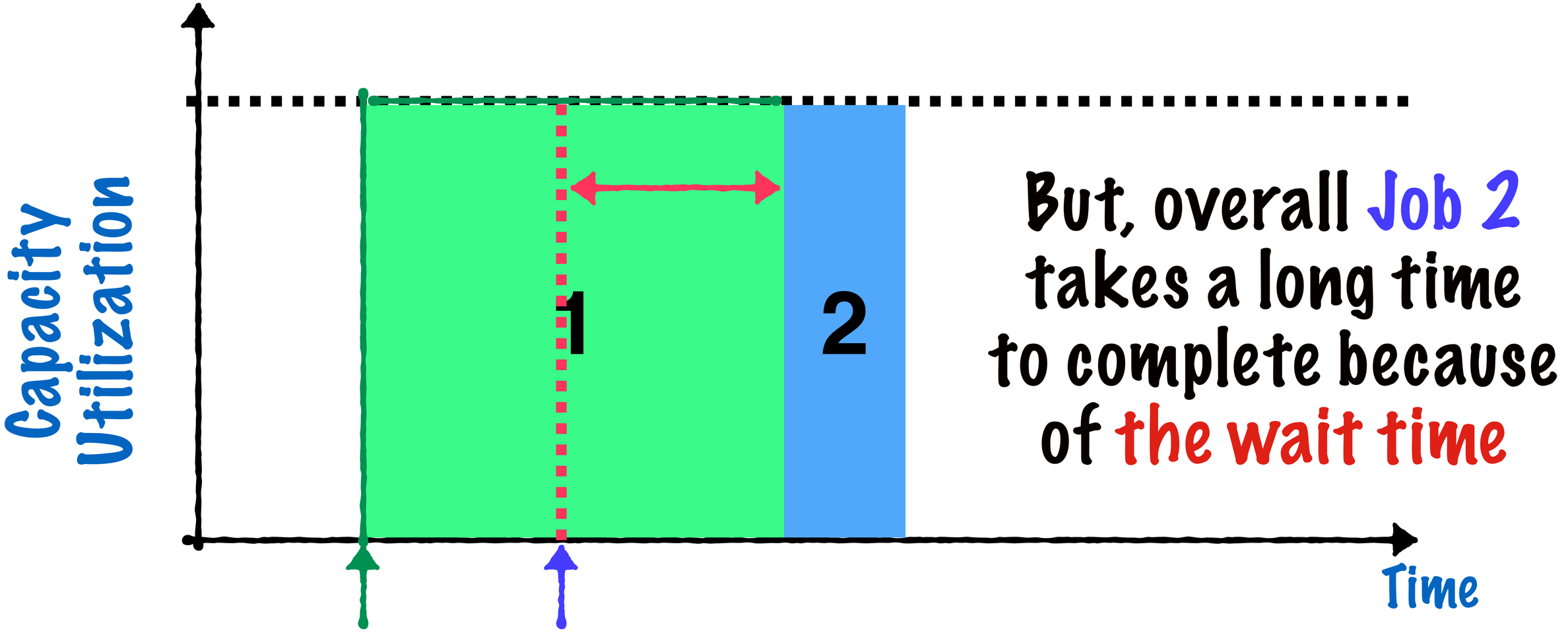


Job 2 begins after Job1 finishes

Job 2 is much smaller and completes running in a short time

Scheduling in YARN

FIFO Scheduler



But, overall Job 2 takes a long time to complete because of the wait time

Scheduling in YARN

FIFO Scheduler

On a cluster shared by many applications, FIFO will cause huge wait times

For this reason, FIFO scheduler is rarely used

Scheduling in YARN

There are 3 scheduling policies available

- FIFO Scheduler 
- Capacity Scheduler
- Fair Scheduler

Scheduling in YARN

Capacity Scheduler

In a Capacity Scheduler, Capacity is distributed to different queues

Scheduling in YARN

Capacity Scheduler

Each queue is allocated a share of the cluster resources

Scheduling in YARN

Capacity Scheduler

Jobs can be submitted to a specific queue

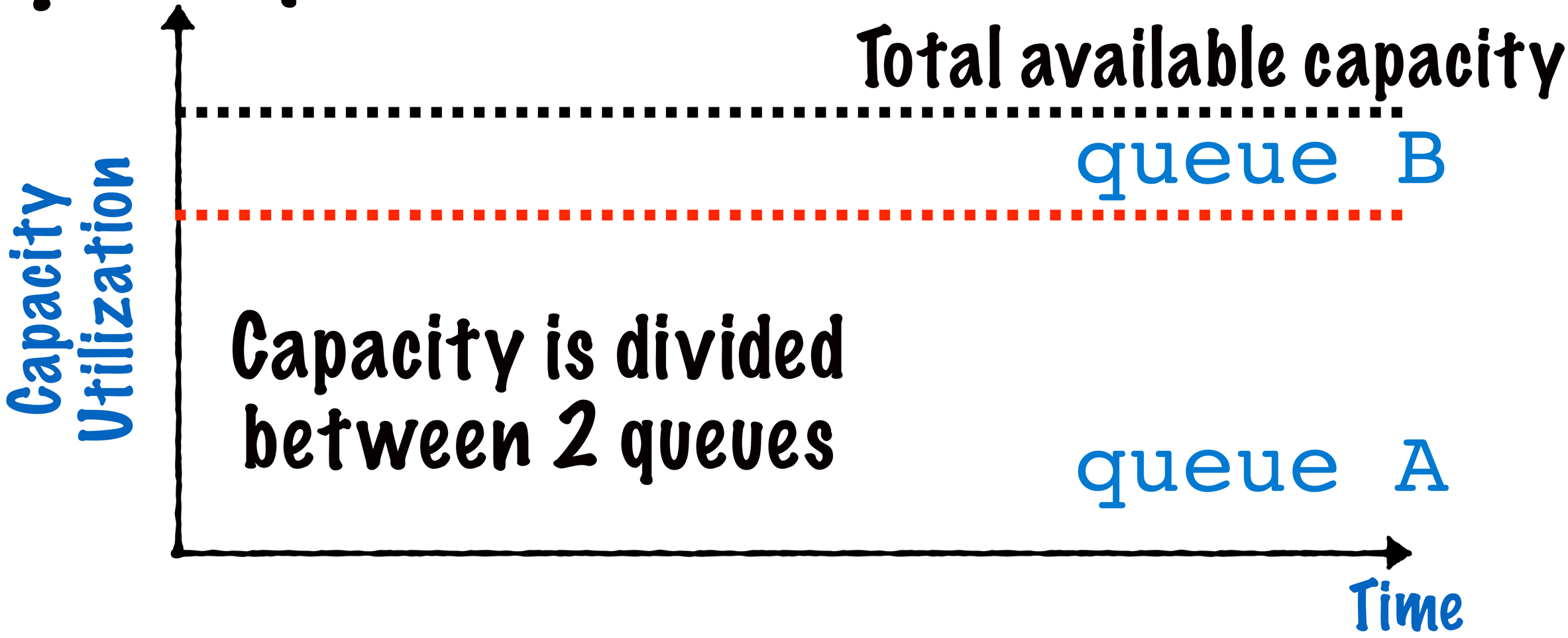
Scheduling in YARN

Capacity Scheduler

Within a queue, FIFO scheduling is followed

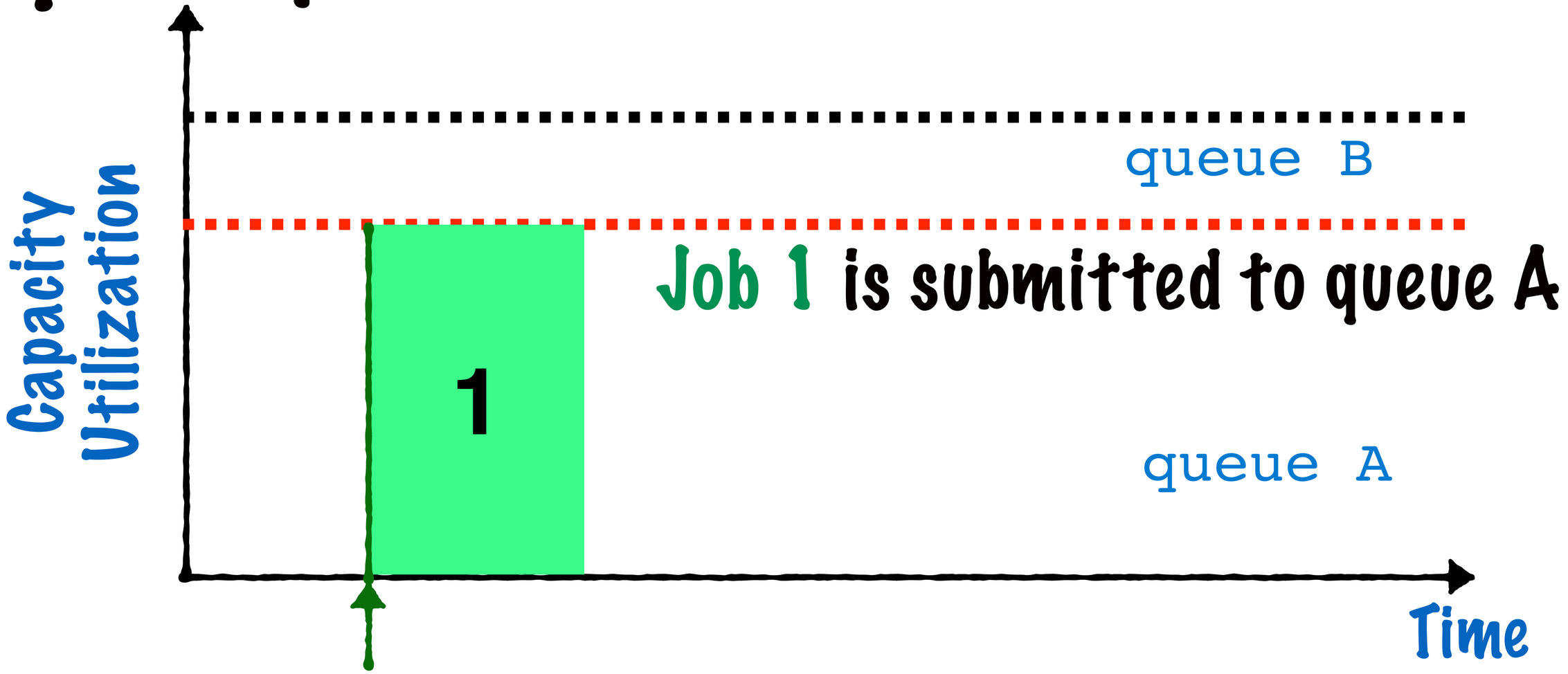
Scheduling in YARN

Capacity Scheduler



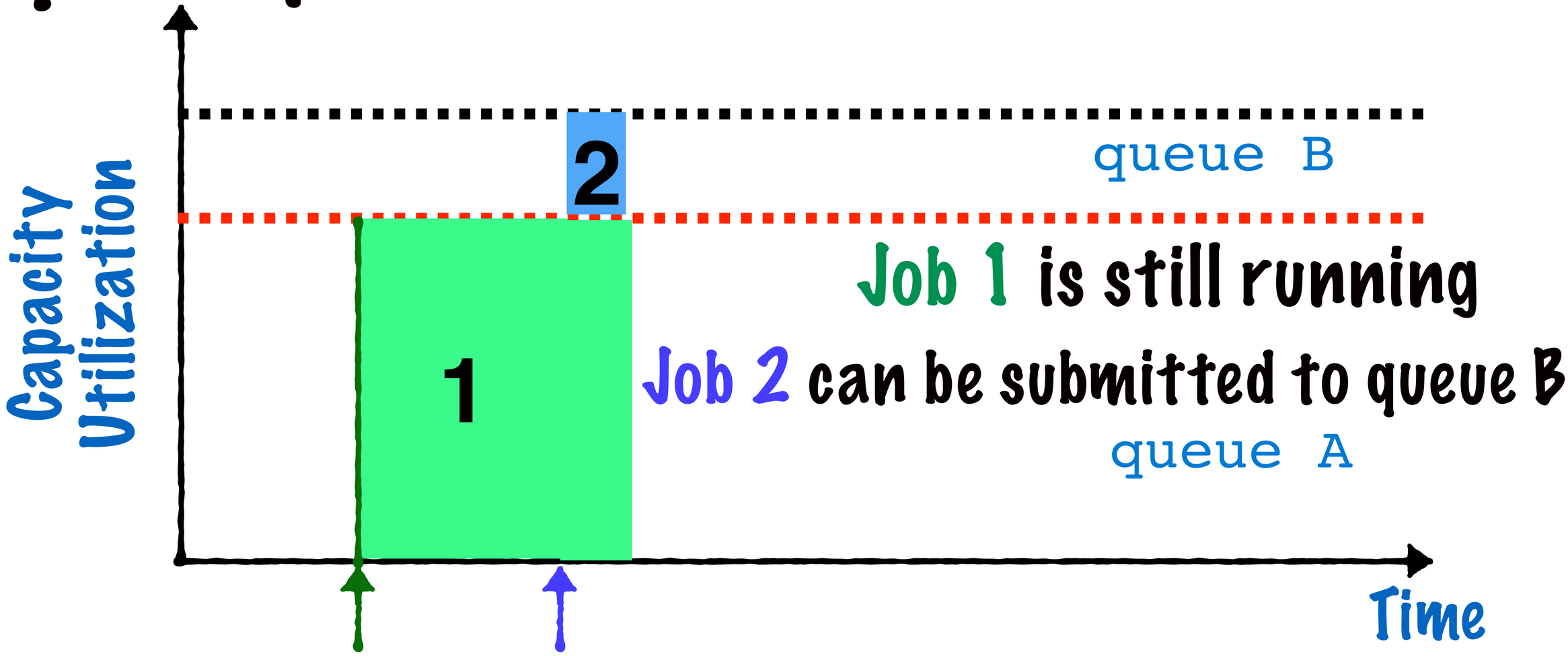
Scheduling in YARN

Capacity Scheduler



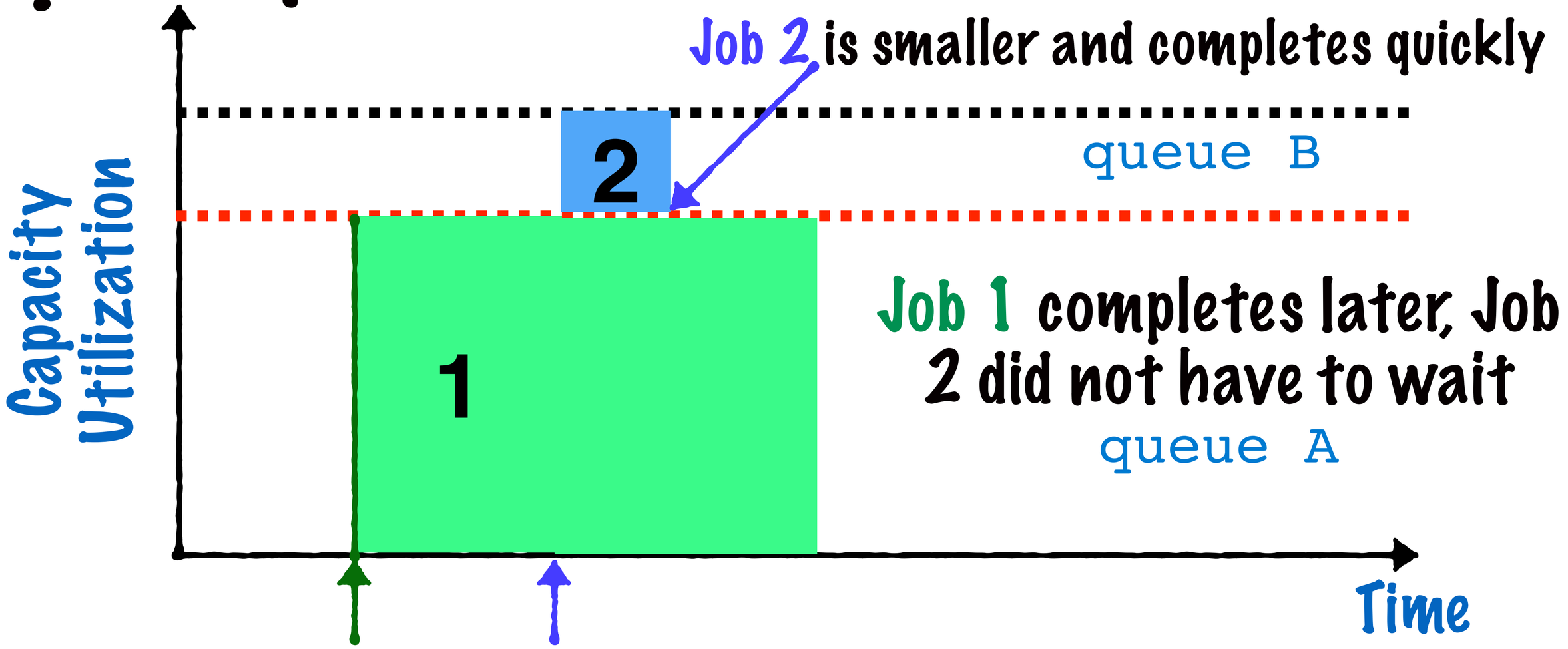
Scheduling in YARN

Capacity Scheduler



Scheduling in YARN

Capacity Scheduler



Scheduling in YARN

Capacity Scheduler

Capacity Scheduler allows small jobs to complete without getting stuck due to large ones

Scheduling in YARN

Capacity Scheduler

The cluster might be underutilized since capacity is reserved for queues

Scheduling in YARN

There are 3 scheduling policies available

- FIFO Scheduler ✓
- Capacity Scheduler ✓
- Fair Scheduler

Scheduling in YARN

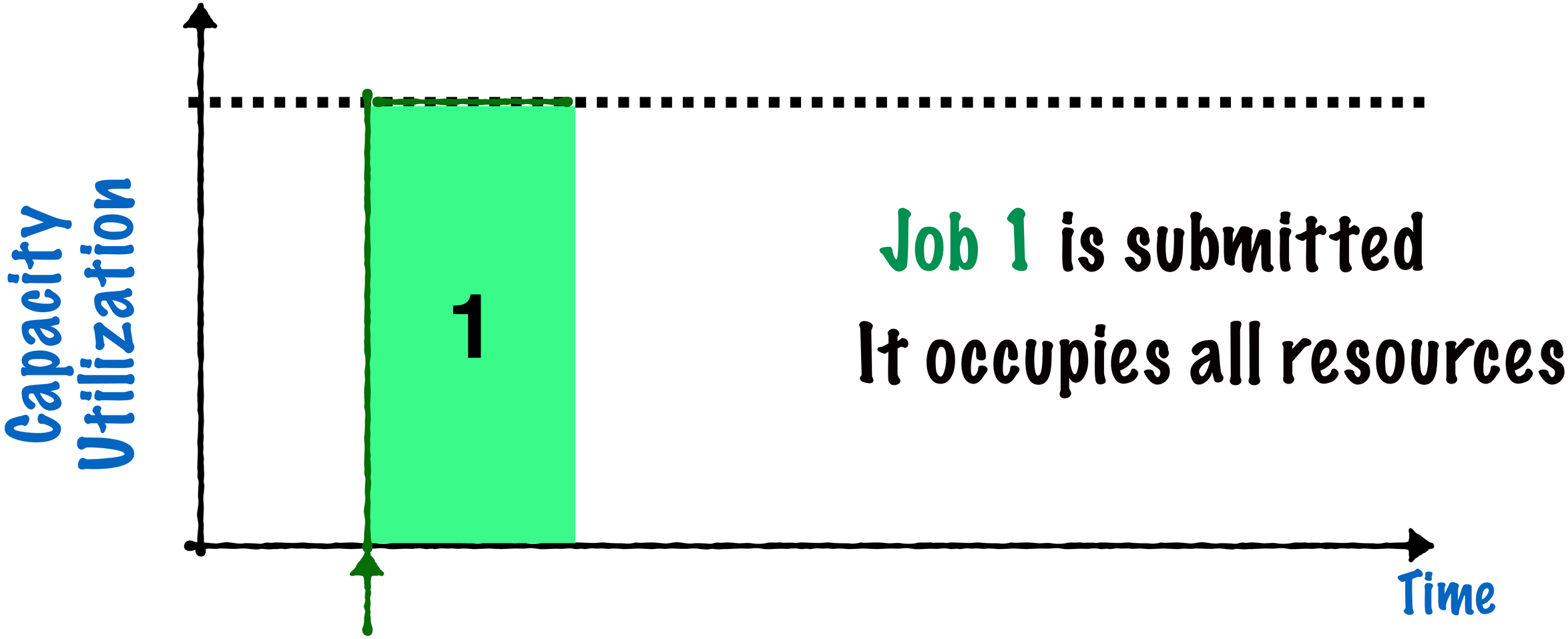
Fair Scheduler

Resources are always proportionally allocated to all jobs

There is no wait time

Scheduling in YARN

Fair Scheduler

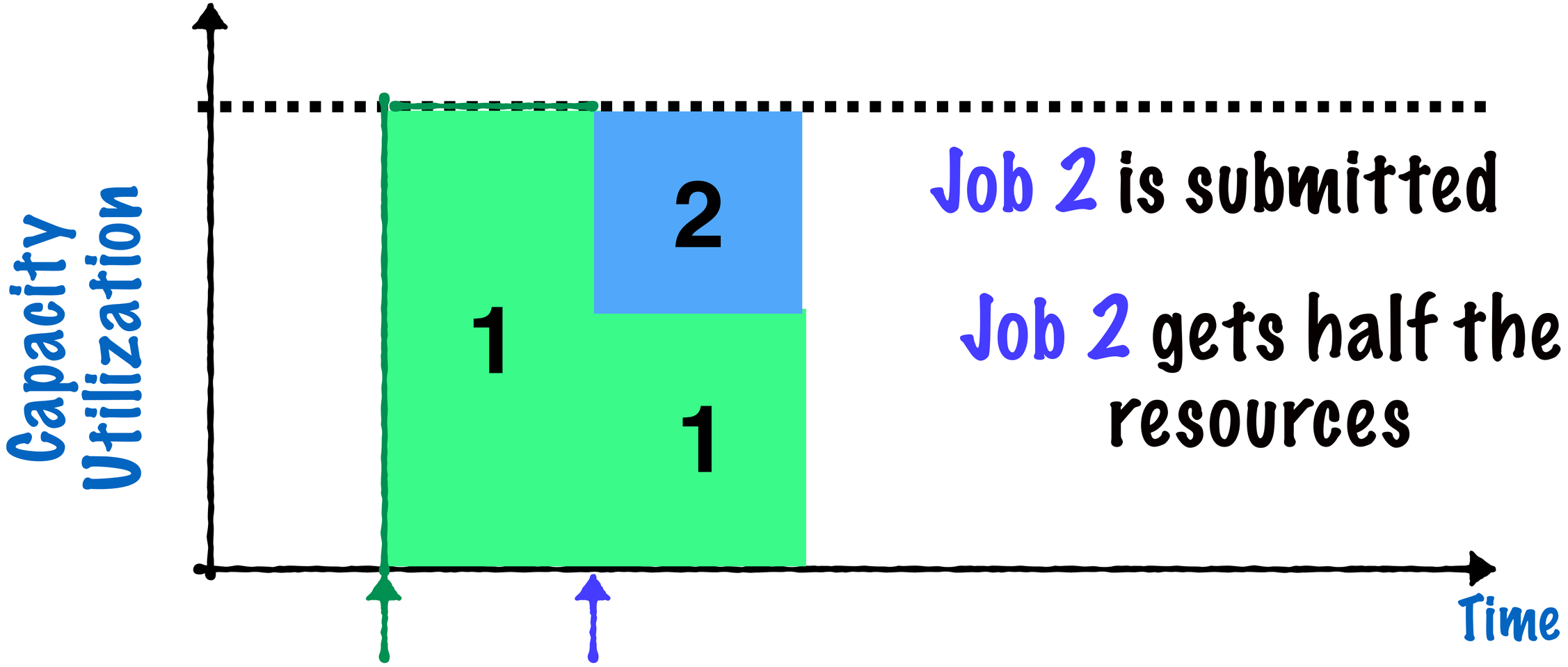


Job 1 is submitted

It occupies all resources

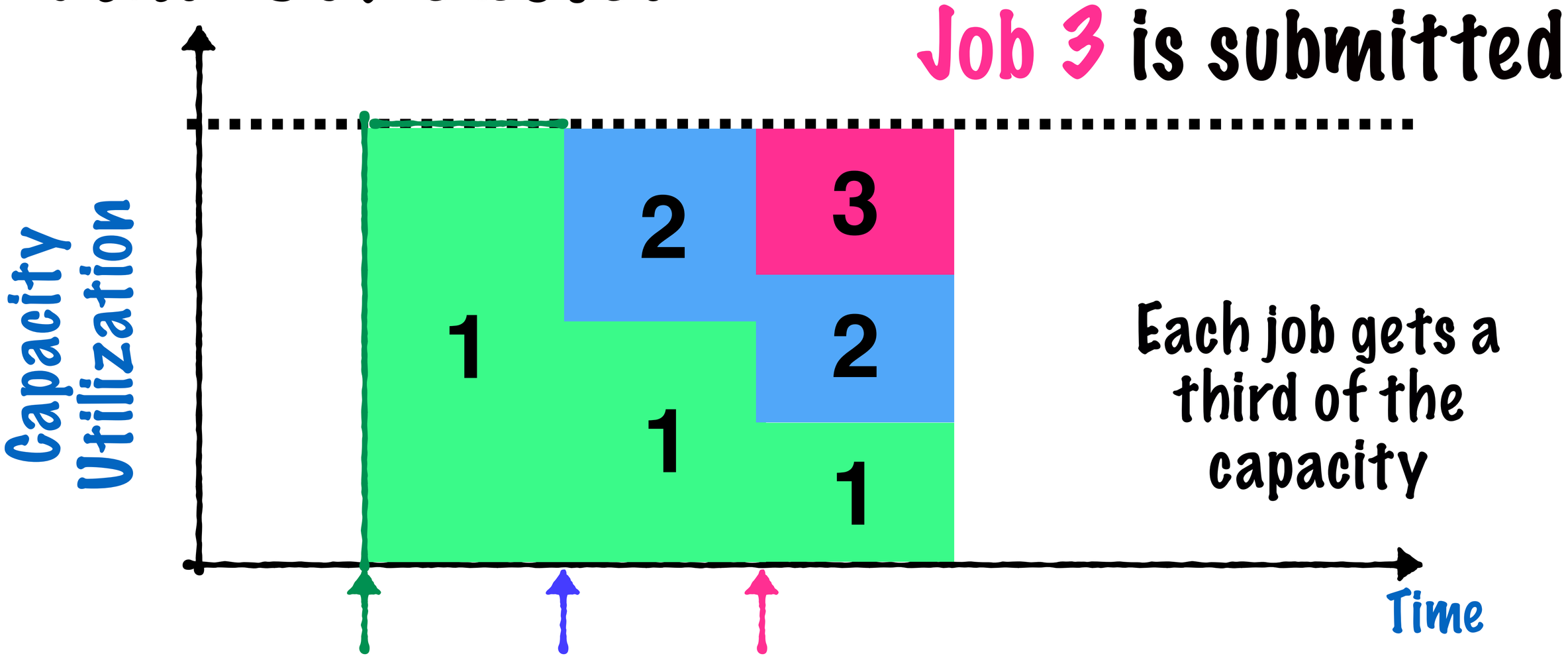
Scheduling in YARN

Fair Scheduler



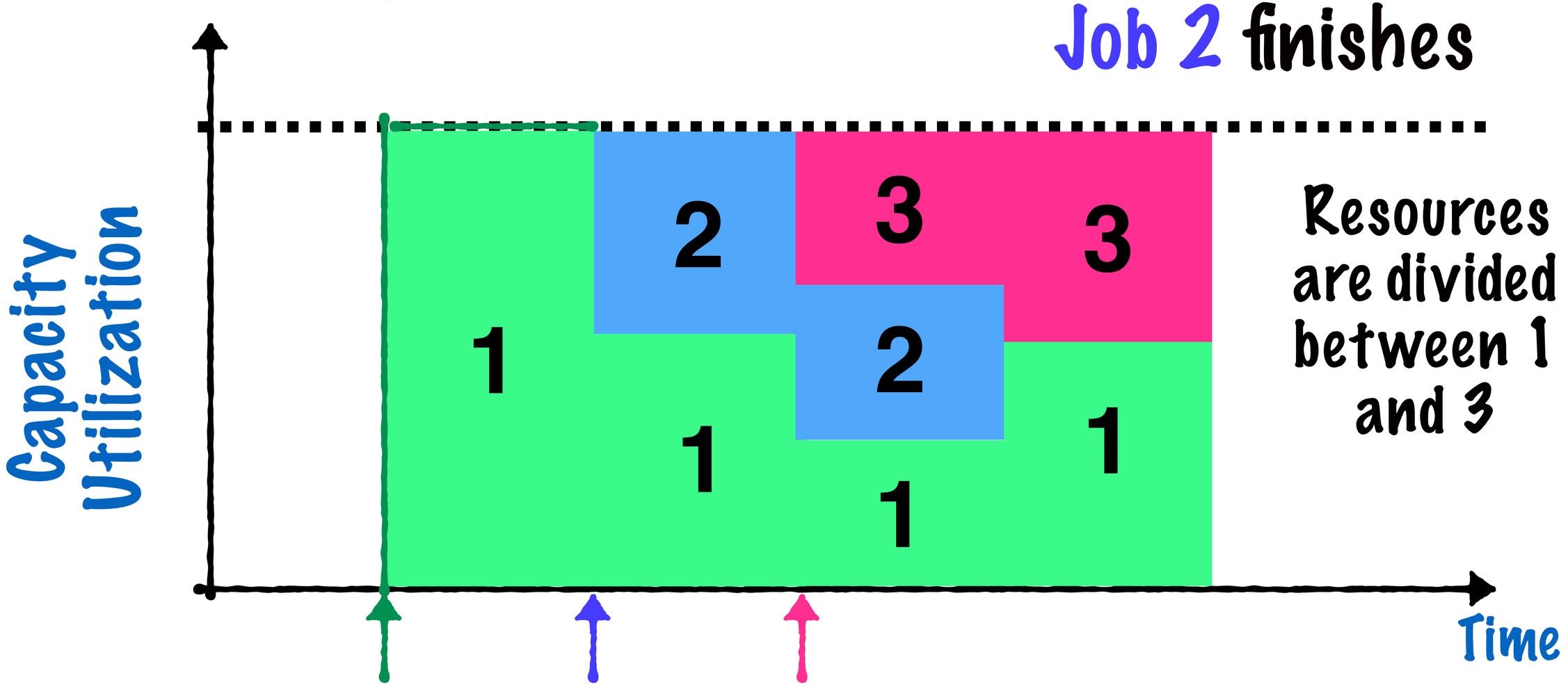
Scheduling in YARN

Fair Scheduler



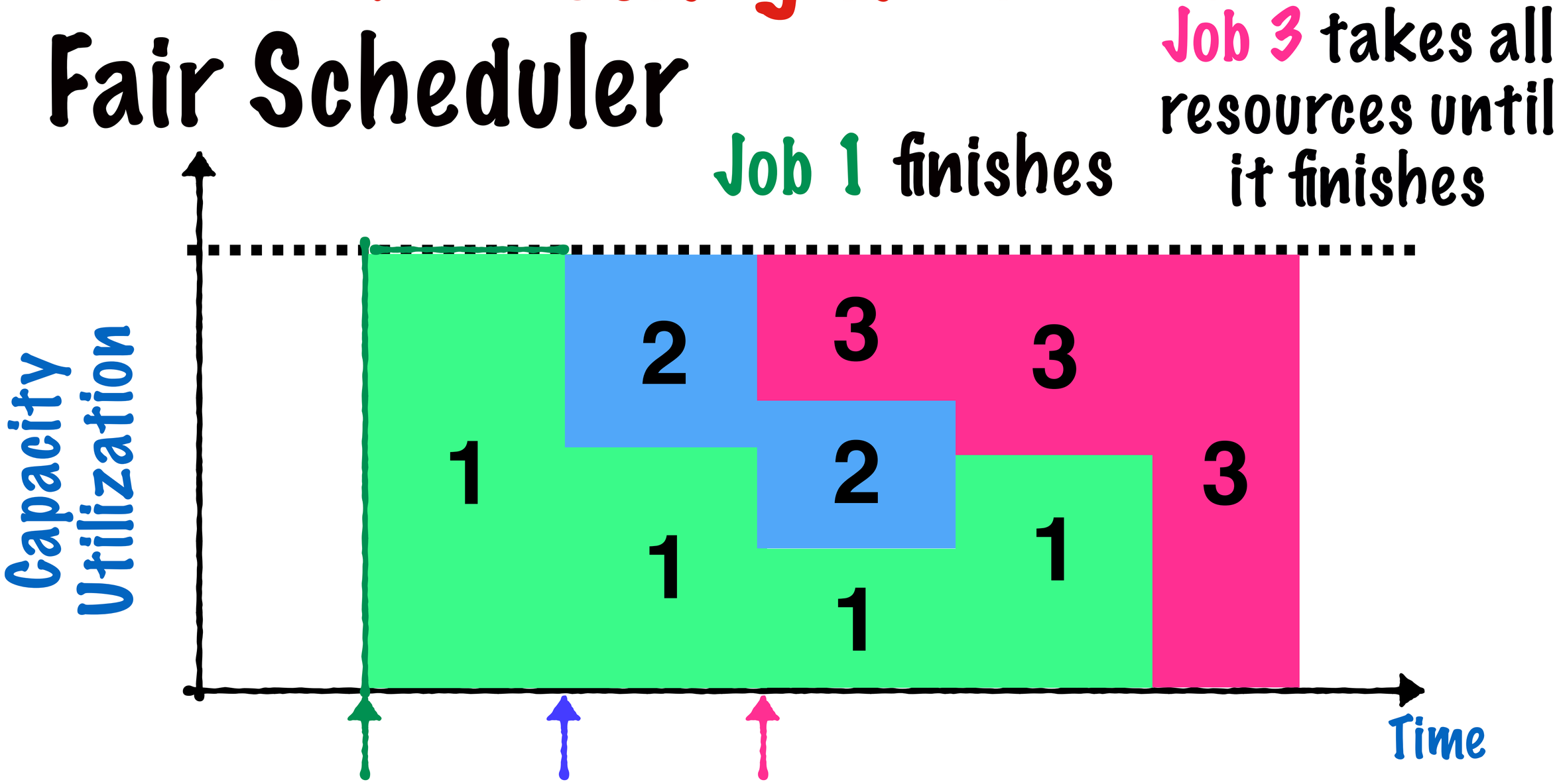
Scheduling in YARN

Fair Scheduler



Scheduling in YARN

Fair Scheduler



Scheduling in YARN

There are 3 scheduling policies available

- FIFO Scheduler ✓
- Capacity Scheduler ✓
- Fair Scheduler ✓

Scheduling in YARN

Configuring the Scheduling policy

Scheduling in YARN

By default, Capacity Scheduling is used

This can be overridden manually using the configuration file

yarn-site.xml

Scheduling in YARN

This is the configuration for Fair Scheduler

```
<configuration>
<property>
<name>yarn.resourcemanager.scheduler.class</name>
<value>org.apache.hadoop.yarn.server.resourcemanager.scheduler.fair.FairScheduler</value>
</property>
</configuration>
```

The complete class name for the FairScheduler has to be used to set the property

Scheduling in YARN

The queue configurations for Capacity Scheduler are stored in

capacity-scheduler.xml

Scheduling in YARN

capacity-scheduler.xml

```
<property>
  <name>yarn.scheduler.capacity.root.queues</name>
  <value>default</value>
  <description>
    The queues at the this level (root is the root queue).
  </description>
</property>

<property>
  <name>yarn.scheduler.capacity.root.default.capacity</name>
  <value>100</value>
  <description>Default queue target capacity.</description>
</property>
```

The list of queues can be set using this property

Scheduling in YARN

capacity-scheduler.xml

```
<property>
  <name>yarn.scheduler.capacity.root.queues</name>
  <value>default</value>
  <description>
    The queues at the this level (root is the root queue).
  </description>
</property>

<property>
  <name>yarn.scheduler.capacity.root.default.capacity</name>
  <value>100</value>
  <description>Default queue target capacity.</description>
</property>
```

By default there is a single queue called default

Scheduling in YARN

capacity-scheduler.xml

```
<property>
  <name>yarn.scheduler.capacity.root.queues</name>
  <value>default</value>
  <description>
    The queues at the this level (root is the root queue).
  </description>
</property>

<property>
  <name>yarn.scheduler.capacity.root.default.capacity</name>
  <value>100</value>
  <description>Default queue target capacity.</description>
</property>
```

The capacity % allocated to the default queue

Scheduling in YARN

capacity-scheduler.xml

```
<property>
  <name>yarn.scheduler.capacity.root.queues</name>
  <value>default</value>
  <description>
    The queues at the this level (root is the root queue).
  </description>
</property>

<property>
  <name>yarn.scheduler.capacity.root.default.capacity</name>
  <value>100</value>
  <description>Default queue target capacity.</description>
</property>
```

By default it is 100%

Scheduling in YARN

capacity-scheduler.xml

```
<property>
  <name>yarn.scheduler.capacity.root.queues</name>
  <value>default</value>
  <description>
    The queues at the this level (root is the root queue).
  </description>
</property>

<property>
  <name>yarn.scheduler.capacity.root.default.capacity</name>
  <value>100</value>
  <description>Default queue target capacity.</description>
</property>
```

You can replace this with a list of queue names

Scheduling in YARN
capacity-scheduler.xml

```
<property>
  <name>yarn.scheduler.capacity.root.queues</name>
  <value>default</value>
  <description>
    The queues at the this level (root is the root queue).
  </description>
</property>

<property>
  <name>yarn.scheduler.capacity.root.default.capacity</name>
  <value>100</value>
  <description>Default queue target capacity.</description>
</property>
```

Replace this with the queue name to set the capacity % for a specific queue

Scheduling in YARN

capacity-scheduler.xml

```
<property>
    <name>yarn.scheduler.capacity.root.queues</name>
    <value>dev,prod</value>
    <description>
      The queues at the this level (root is the root queue).
    </description>
  </property>

  <property>
    <name>yarn.scheduler.capacity.root.dev.capacity</name>
    <value>30</value>
    <description>Default queue target capacity.</description>
  </property>

  <property>
    <name>yarn.scheduler.capacity.root.prod.capacity</name>
    <value>70</value>
    <description>Default queue target capacity.</description>
  </property>
```

Here is an example to create 2 queues

Scheduling in YARN

capacity-scheduler.xml

```
<property>
    <name>yarn.scheduler.capacity.root.queues</name>
    <value>dev,prod</value>
    <description>
      The queues at the this level (root is the root queue).
    </description>
  </property>

  <property>
    <name>yarn.scheduler.capacity.root.dev.capacity</name>
    <value>30</value>
    <description>Default queue target capacity.</description>
```

Two queues, dev and prod

```
    <value>dev,prod</value>
    <description>
      The queues at the this level (root is the root queue).
    </description>
  </property>

  <property>
    <name>yarn.scheduler.capacity.root.dev.capacity</name>
    <value>30</value>
    <description>Default queue target capacity.</description>
  </property>

  <property>
    <name>yarn.scheduler.capacity.root.prod.capacity</name>
    <value>70</value>
    <description>Default queue target capacity.</description>
  </property>
```

30% reserved for dev queue

```
<property>
  <name>yarn.scheduler.capacity.root.dev.capacity</name>
  <value>30</value>
  <description>Default queue target capacity.</description>
</property>

<property>
  <name>yarn.scheduler.capacity.root.prod.capacity</name>
  <value>70</value>
  <description>Default queue target capacity.</description>
</property>
```

70% reserved for prod queue

Scheduling in YARN

capacity-scheduler.xml

```
<property>
    <name>yarn.scheduler.capacity.root.queues</name>
    <value>dev,prod</value>
    <description>
      The queues at the this level (root is the root queue).
    </description>
  </property>

  <property>
    <name>yarn.scheduler.capacity.root.dev.capacity</name>
    <value>30</value>
    <description>Default queue target capacity.</description>
  </property>

  <property>
    <name>yarn.scheduler.capacity.root.prod.capacity</name>
    <value>70</value>
    <description>Default queue target capacity.</description>
  </property>
```

You can even create a hierarchy of queues

Scheduling in YARN

capacity-scheduler.xml

```
<property>
    <name>yarn.scheduler.capacity.root.queues</name>
    <value>dev,prod</value>
    <description>
      The queues at the this level (root is the root queue).
    </description>
  </property>

  <property>
    <name>yarn.scheduler.capacity.root.dev.capacity</name>
    <value>30</value>
    <description>Default queue target capacity.</description>
  </property>

  <property>
    <name>yarn.scheduler.capacity.root.prod.capacity</name>
    <value>70</value>
```

Replace root with root.dev to create some queues within the dev queue

Scheduling in YARN

capacity-scheduler.xml

```
<property>
    <name>yarn.scheduler.capacity.root.queues</name>
    <value>dev,prod</value>
    <description>
      The queues at the this level (root is the root queue).
    </description>
  </property>

  <property>
    <name>yarn.scheduler.capacity.root.dev.capacity</name>
    <value>30</value>
    <description>Default queue target capacity.</description>
  </property>

  <property>
    <name>yarn.scheduler.capacity.root.prod.capacity</name>
    <value>70</value>
```

Replace root.dev with root.dev.<queuename> to set the capacity

Scheduling in YARN

When you submit a job, you can specify the queue using **mapred.job.queue.name** property

```
$ hadoop jar <jarPath> -D mapred.job.queue.name=root.prod
```

Scheduling in YARN

capacity-scheduler.xml

Queues should be configured based on the type of tasks / type of users

Scheduling in YARN

capacity-scheduler.xml

For instance, you can have separate queues for

1. Daily runs vs Ad-hoc runs
2. Data Analysts vs Engineers
3. Production jobs vs Test runs

Scheduling in YARN

Delay Scheduling

Scheduling in YARN

In the capacity-scheduler.xml, there is a property called node-locality-delay

```
<property>
  <name>yarn.scheduler.capacity.node-locality-delay</name>
  <value>40</value>
  <description>
    Number of missed scheduling opportunities after which the CapacityScheduler
    attempts to schedule rack-local containers.
    Typically this should be set to number of nodes in the cluster, By default is setting
    approximately number of nodes in one rack which is 40.
  </description>
</property>
```

Scheduling in YARN

```
<property>
  <name>yarn.scheduler.capacity.node-locality-delay</name>
  <value>40</value>
  <description>
    Number of missed scheduling opportunities after which the CapacityScheduler
    attempts to schedule rack-local containers.
    Typically this should be set to number of nodes in the cluster, By default
    approximately number of nodes in one rack which is 40.
  </description>
</property>
```

Let's parse this

Scheduling in YARN

Sometimes YARN resource manager gets a request with a location constraint

This constraint specifies a particular node on which the computation is required to run

Scheduling in YARN

Here we are delaying the scheduling of a job because a specific location is not available

This is usually better than transferring the data from the specified location to a free node

Scheduling in YARN

```
<property>
  <name>yarn.scheduler.capacity.node-locality-delay</name>
  <value>40</value>
  <description>
    Number of missed scheduling opportunities after which the CapacityScheduler
    attempts to schedule rack-local containers.
    Typically this should be set to number of nodes in the cluster, By default i
    approximately number of nodes in one rack which is 40.
  </description>
</property>
```

By default Delay Scheduling is turned on