

# Hadoop YARN

YARN is the resource management system in Hadoop 2.0. It provides a distributed resource manager and a scheduler.

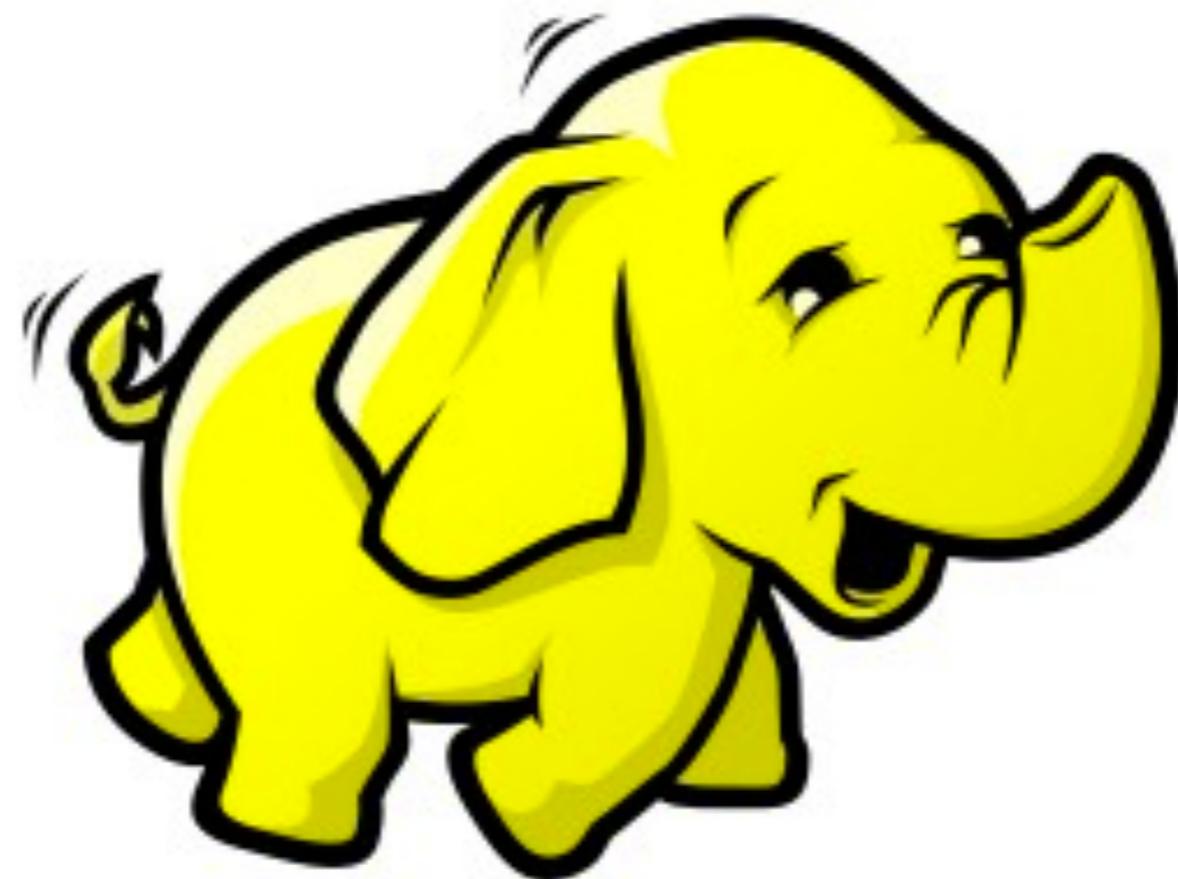
YARN is designed to support multiple applications running on the same cluster. It provides a common interface for managing resources and scheduling tasks across different applications.

YARN consists of two main components: the ResourceManager (RM) and the NodeManager (NM). The RM is responsible for managing the cluster's resources and scheduling tasks. The NM is responsible for managing the local resources on each node and executing tasks.

YARN supports a variety of applications, including MapReduce, Tez, and HDFS. It also provides a framework for developing new applications. YARN is designed to be highly fault-tolerant and can handle large-scale workloads.

YARN is a key component of the Hadoop ecosystem and is used in many large-scale data processing and analysis applications. It has been adopted by many organizations and is available as open source software.

let's start by reviewing  
Map/Reduce

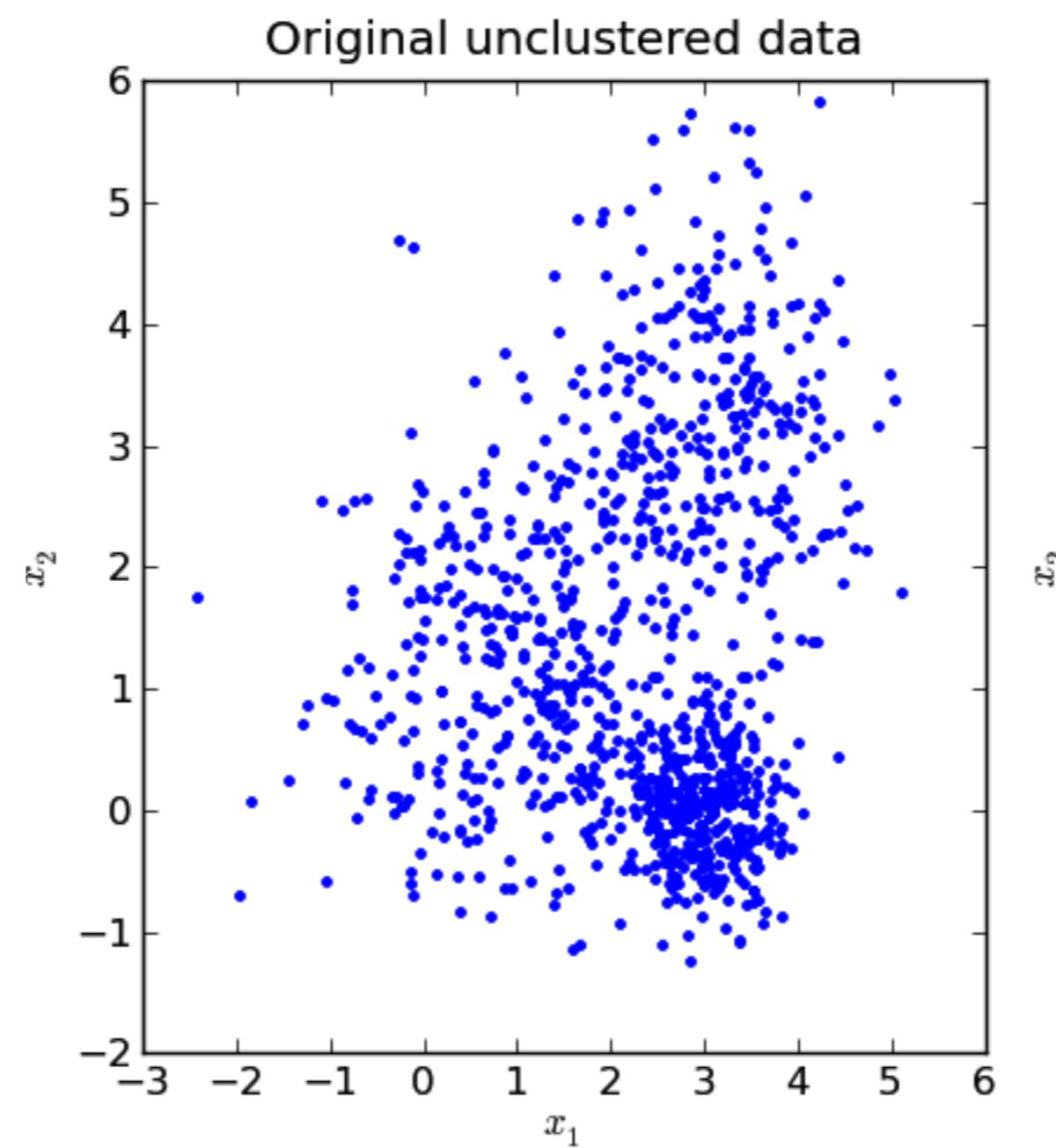


# SAY "WORD COUNT"

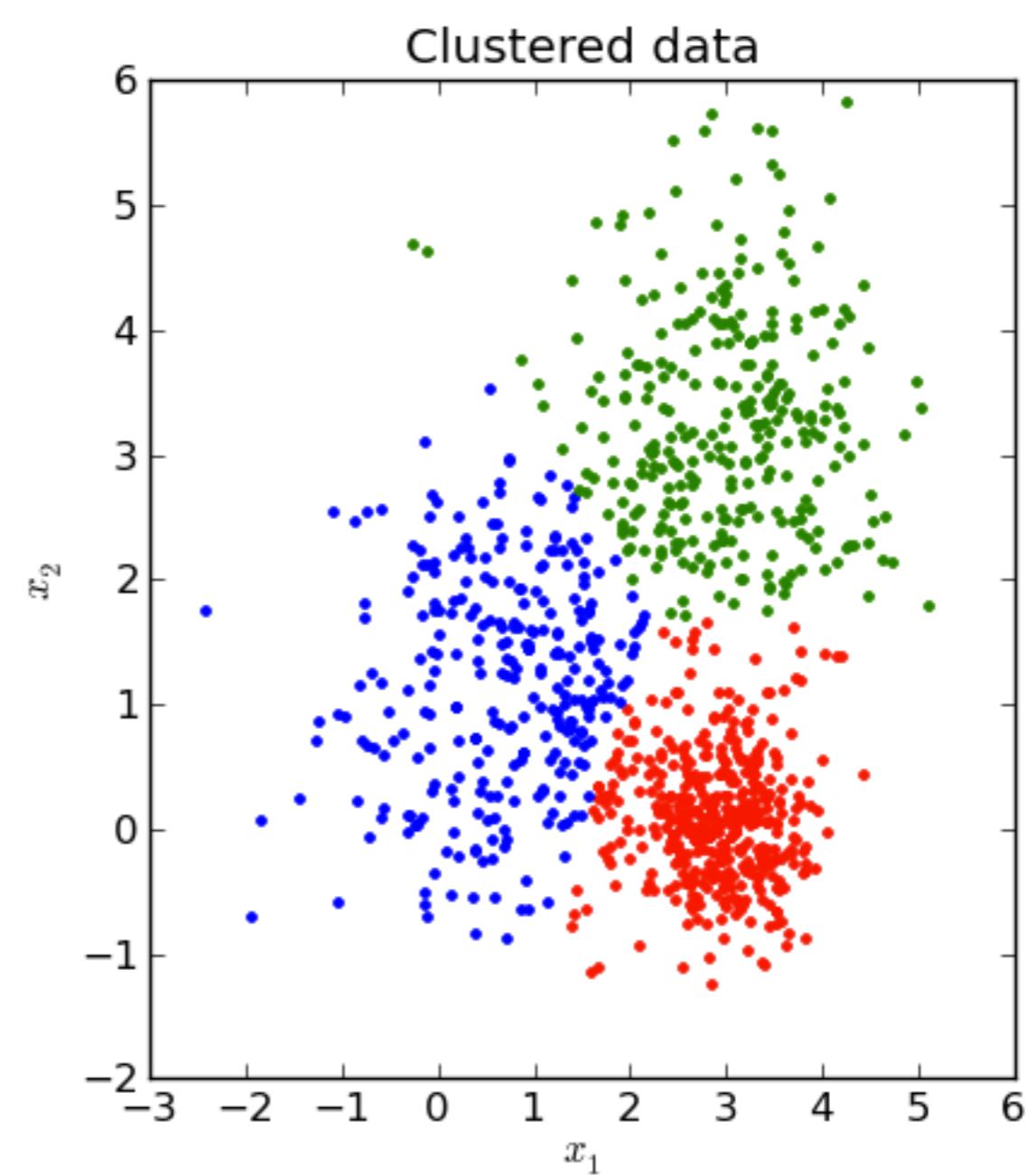
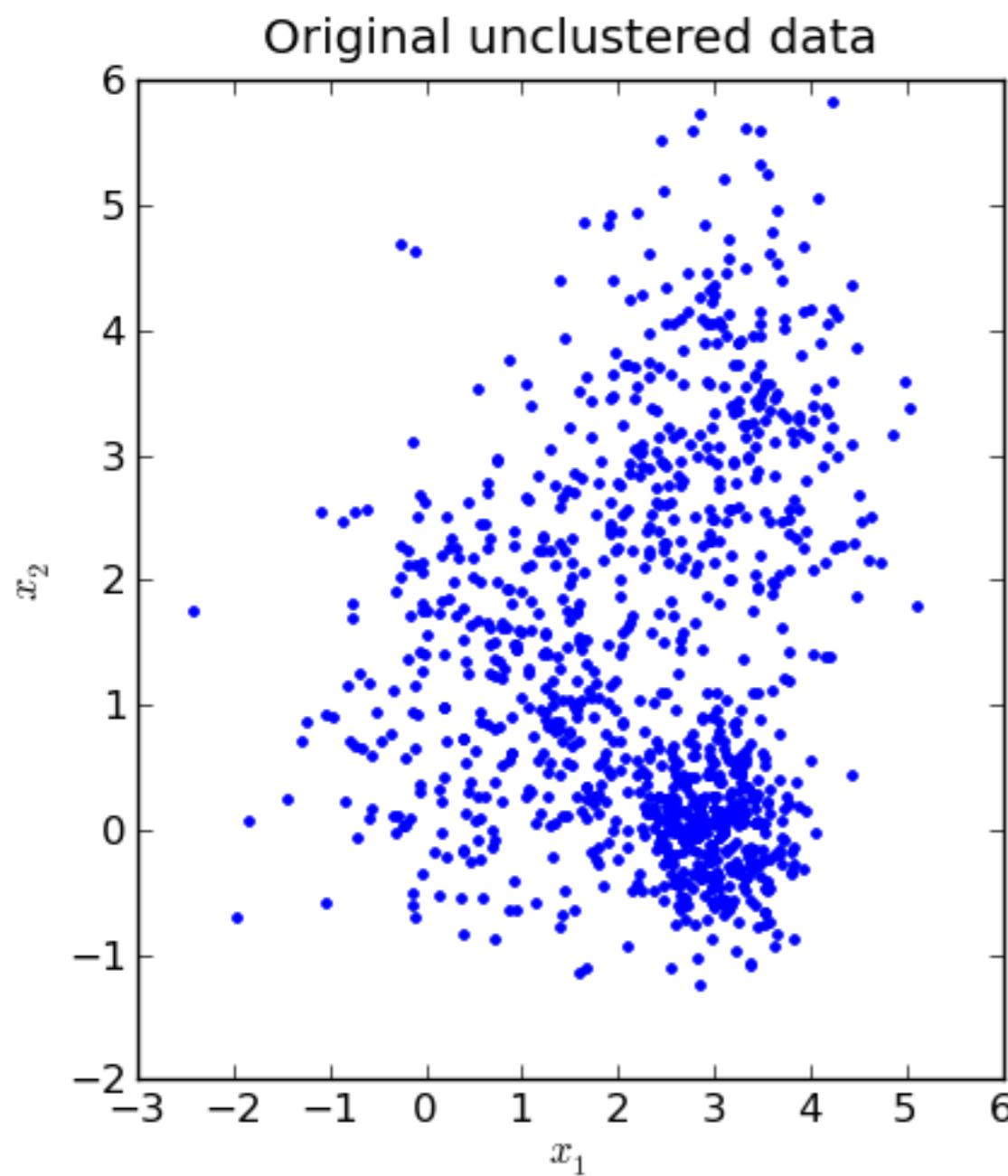


## ONE MORE TIME...

# ok, ok - K-Means



K=3



```
def perform_kmeans():
    isStillMoving = 1

    initialize_centroids()

    while(isStillMoving):
        recalculate_centroids()
        isStillMoving = update_clusters()

    return
```

# put differently

## In the map phase

- Read the cluster centers into memory from a File/HBase
- Iterate over each cluster center for each input key/value pair.
- Measure the distances and save the nearest center which has the lowest distance to the vector
- Write the clustercenter with its vector to the context.

## In the reduce phase

- Iterate over each value vector and calculate the average vector. (Sum each vector and devide each part by the number of vectors we received).
- This is the new center, save it into a File/HBase.
- Check if we need the different between runs is  $< \text{epsilon}$

**Run this whole damn thing again and again  
until  $\text{diff} < \text{epsilon}$**

So you want to implement it  
differently something that  
doesn't rely on map/reduce



# bivm Hadoop Map/Reduce Administration

**State:** RUNNING

**Started:** Thu Oct 10 02:57:48 EDT 2013

**Version:** 1.1.1, rf0025c9fd25730e3c1bfebceeeeb50d930b4fbbaa

**Compiled:** Fri Aug 9 17:06:14 PDT 2013 by jenkins

**Identifier:** 201310100257

**SafeMode:** OFF

## Cluster Summary (Heap Size is 28.52 MB/2.02 GB)

Running Map Tasks	Running Reduce Tasks	Total Submissions	Nodes	Occupied Map Slots	Occupied Reduce Slots	Reserved Map Slots	Reserved Reduce Slots	Map Task Capacity	Reduce Task Capacity	Avg Tasks/L
0	0	0	1	0	0	0	0	2	1	3.00

## Scheduling Information

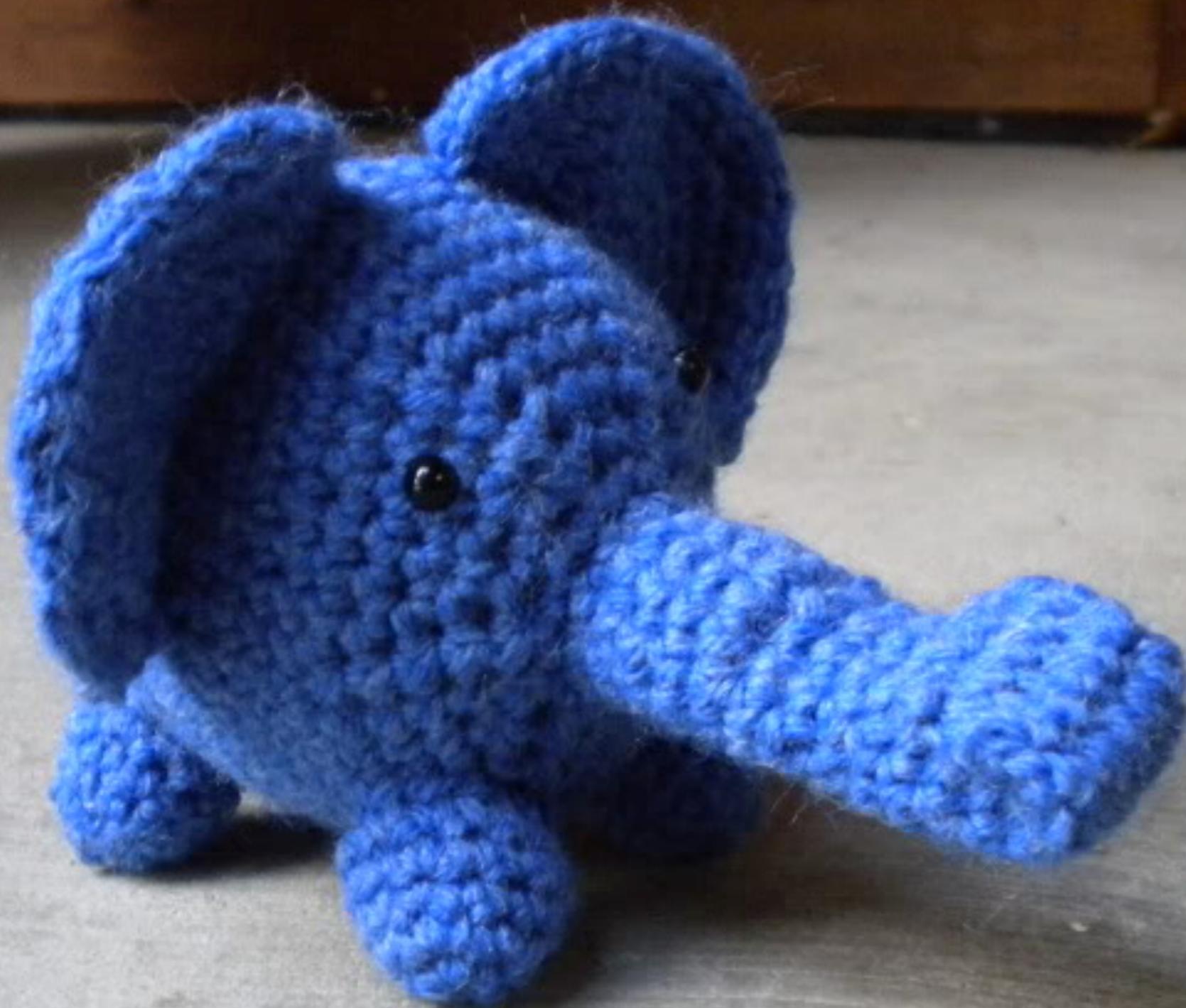
Queue Name	State	Scheduling Information
<a href="#">default</a>	running	N/A

**Filter (Jobid, Priority, User, Name)**

Example: 'user:smith 3200' will filter by 'smith' only in the user field and '3200' in all fields

## Running Jobs

Resource management is tied to Map/Reduce

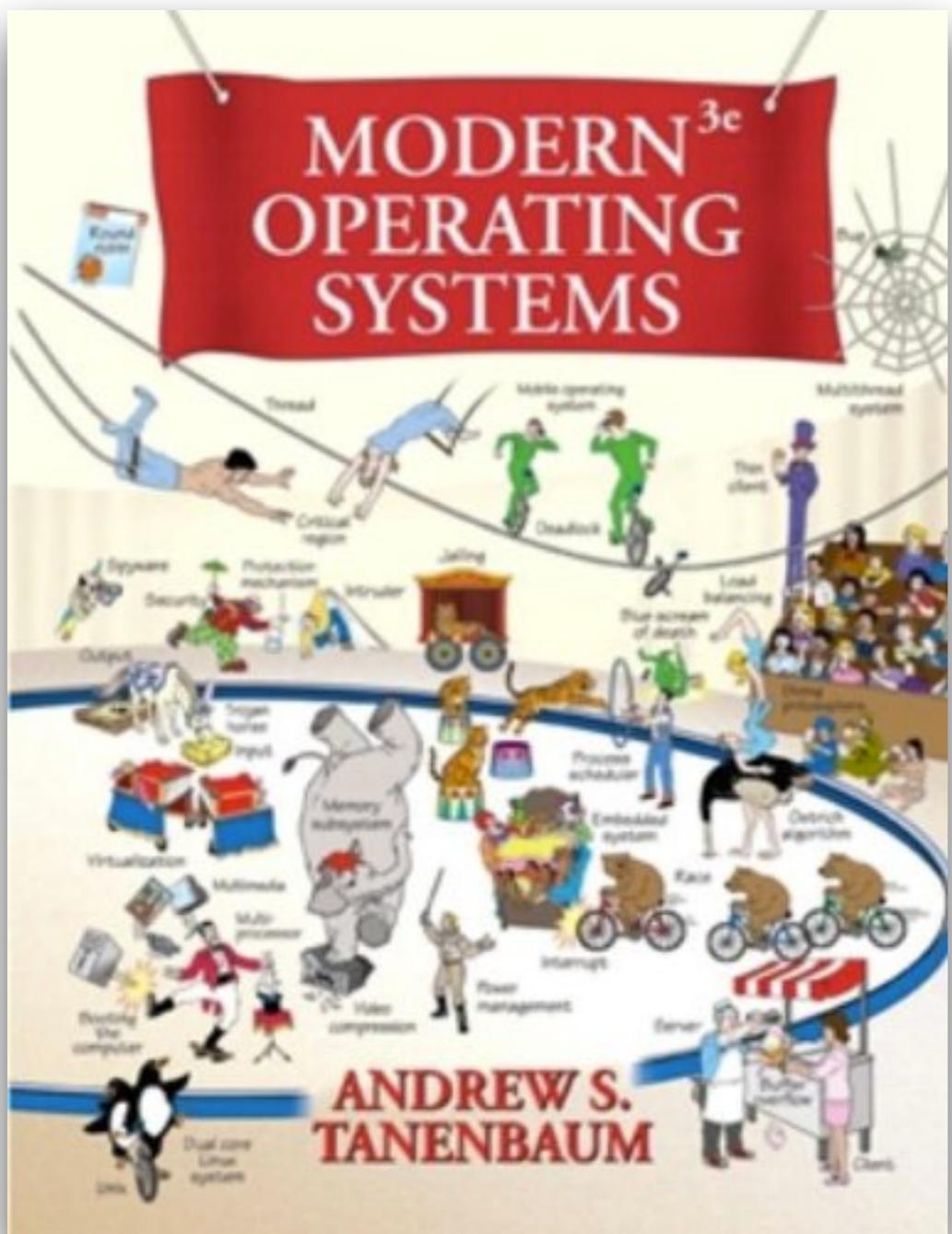


Yet Another Resource Negotiator

## Applications Run Natively IN Hadoop



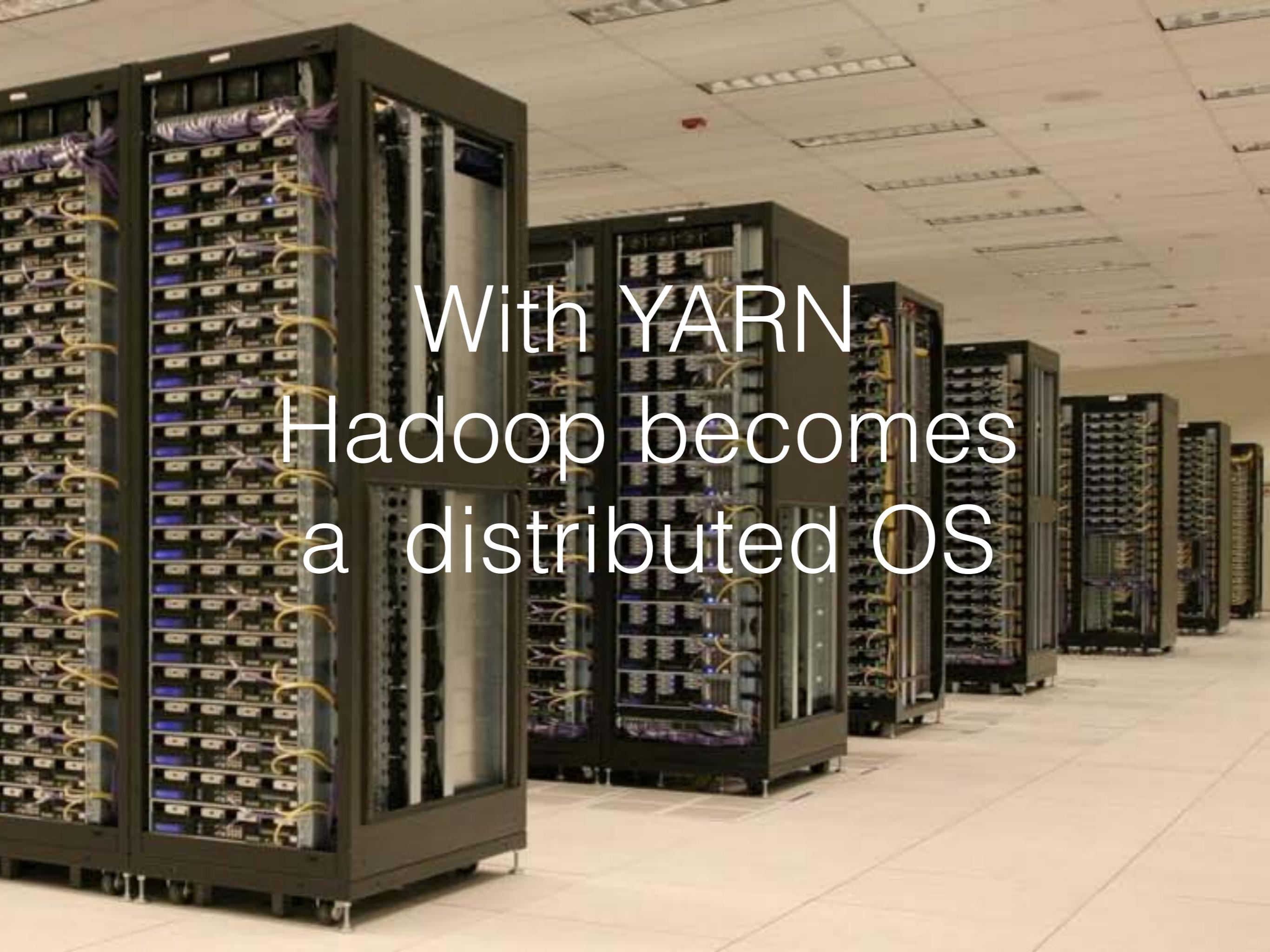
YARN takes Hadoop  
beyond Map/Reduce



# What is an OS?

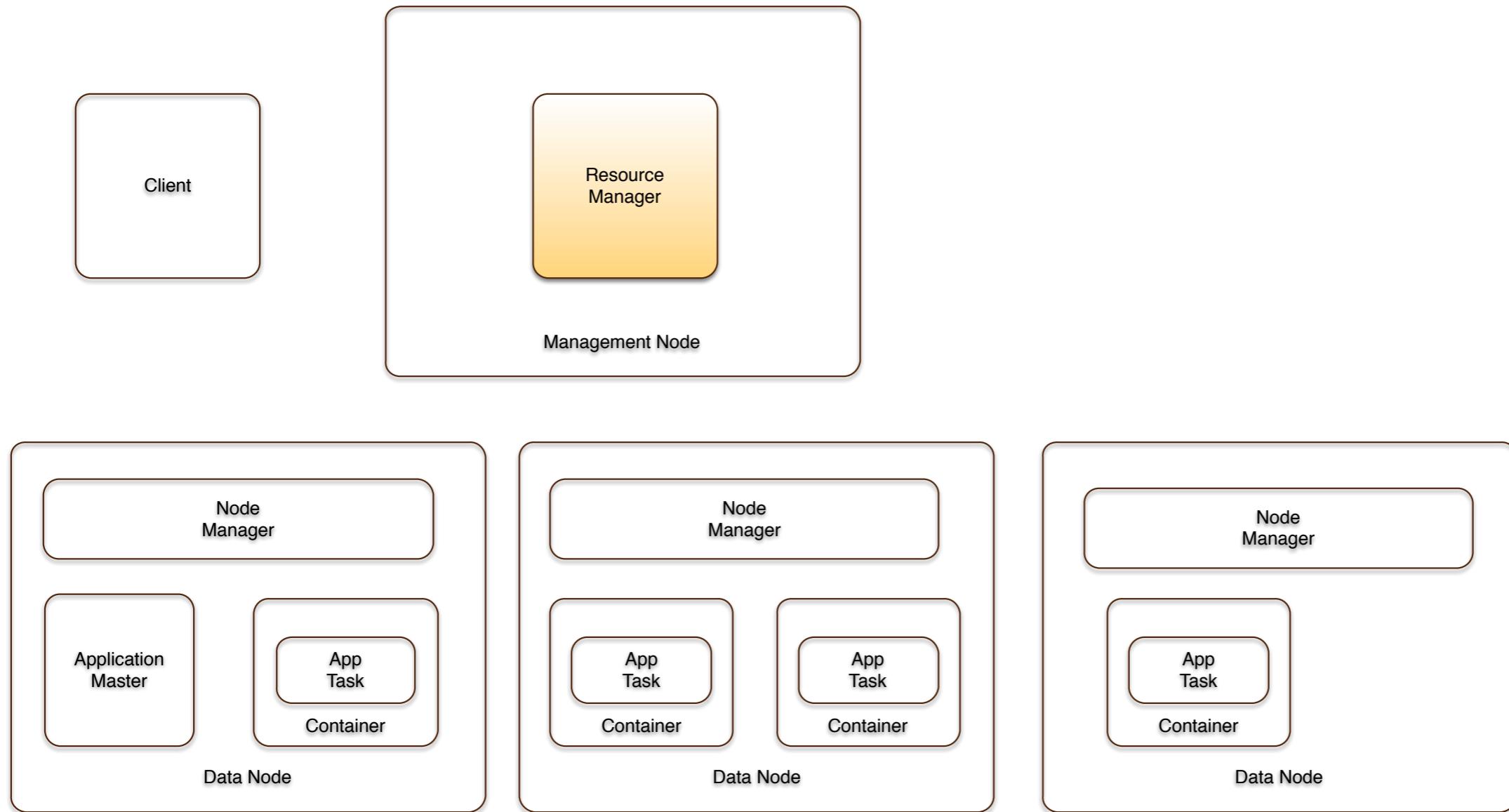
“...the function of the operating system is to present the user with the equivalent of an extended machine or **virtual machine** that is easier to program than the underlying hardware”

“...The Operating System as a  
**Resource Manager**... the job of the  
operating system is to provide for an orderly and  
controlled allocation of the processors, memories,  
and/or devices among the various programs  
competing for them”

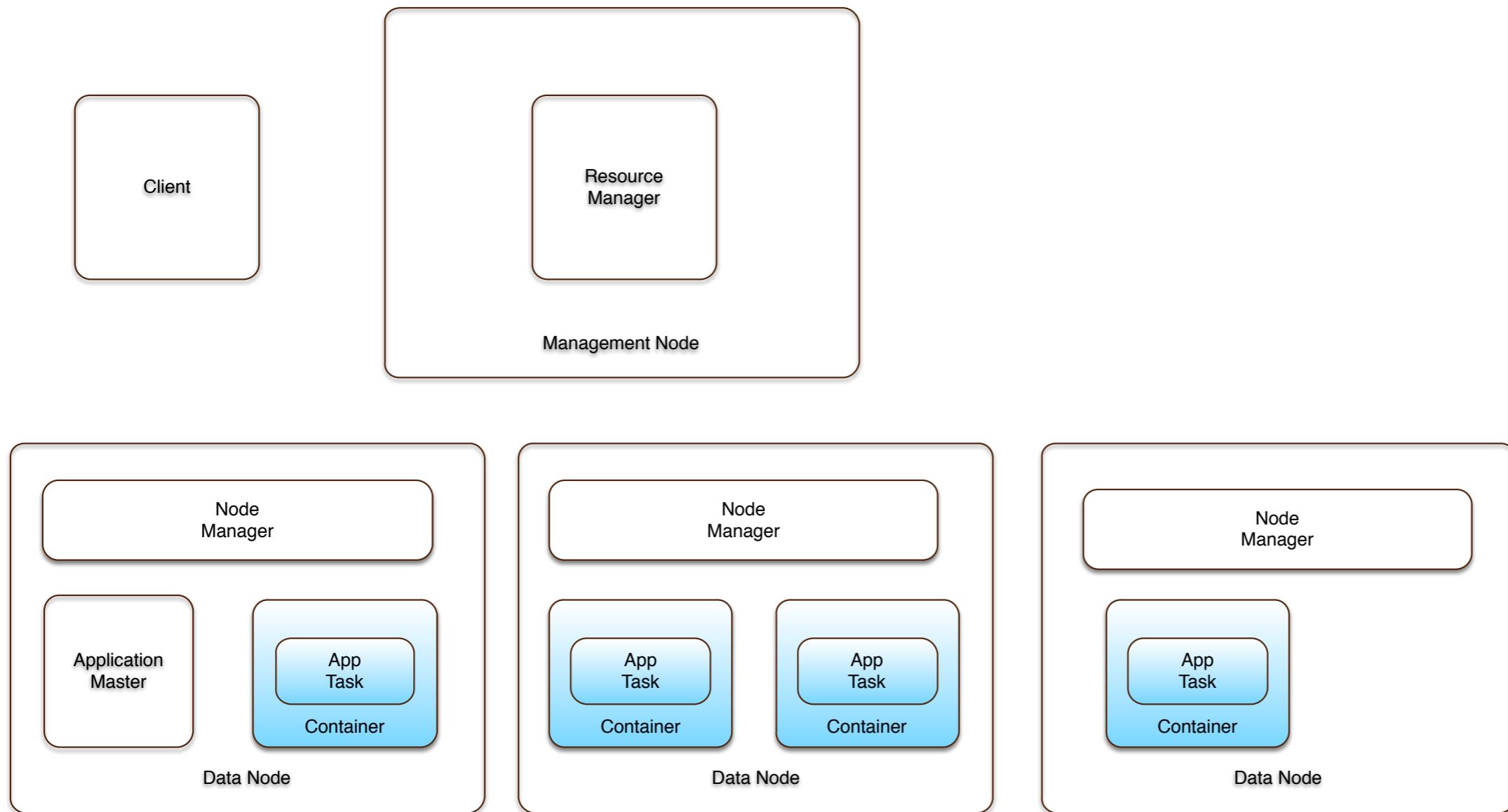
A photograph of a data center floor showing several rows of server racks. The racks are dark-colored and filled with various electronic components and cables. The floor is made of light-colored tiles, and the ceiling above is white with some visible infrastructure.

With YARN  
Hadoop becomes  
a distributed OS

# The Resource Manager is essentially a scheduler

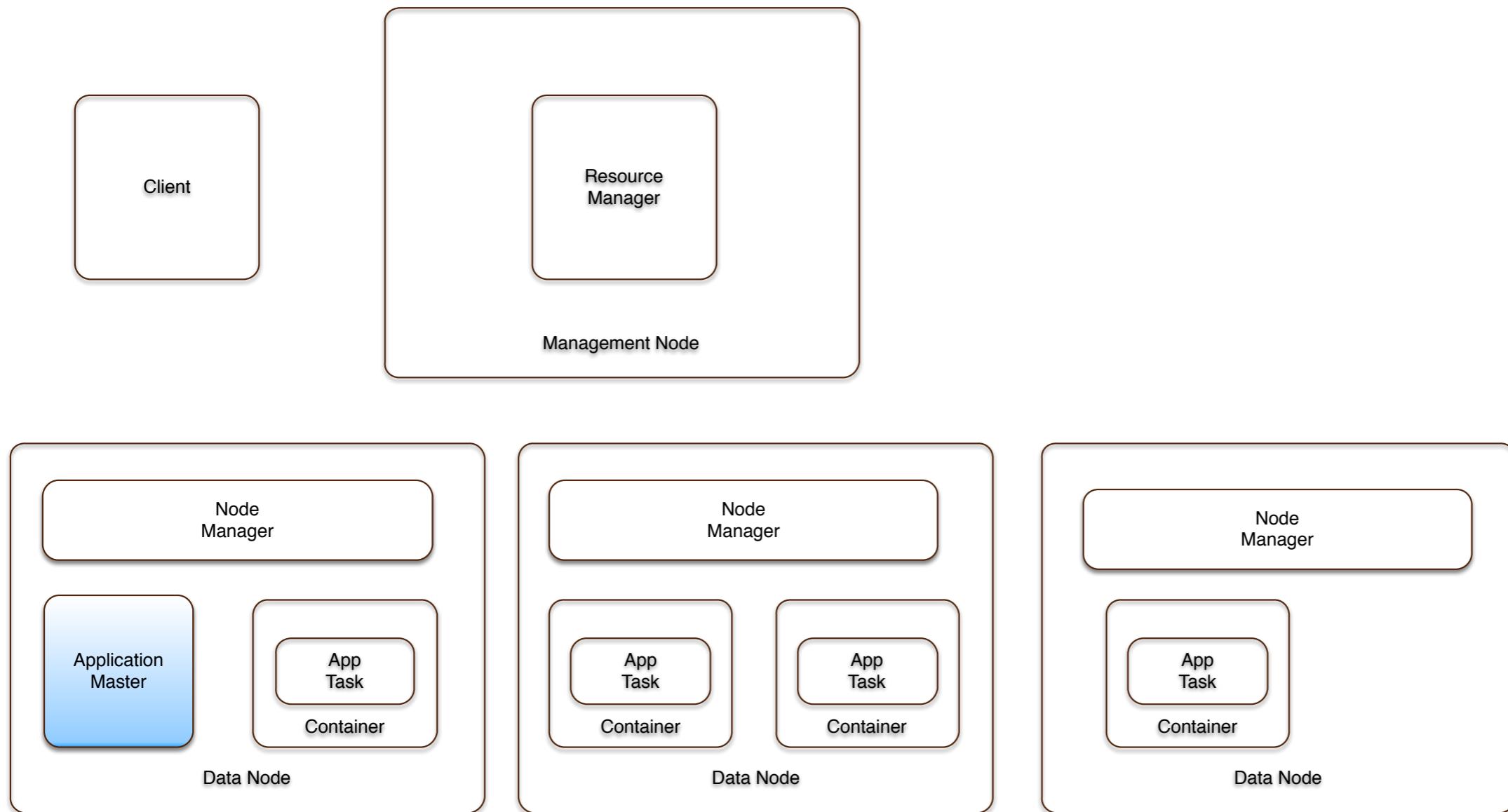


# Containers are allocations of physical resources

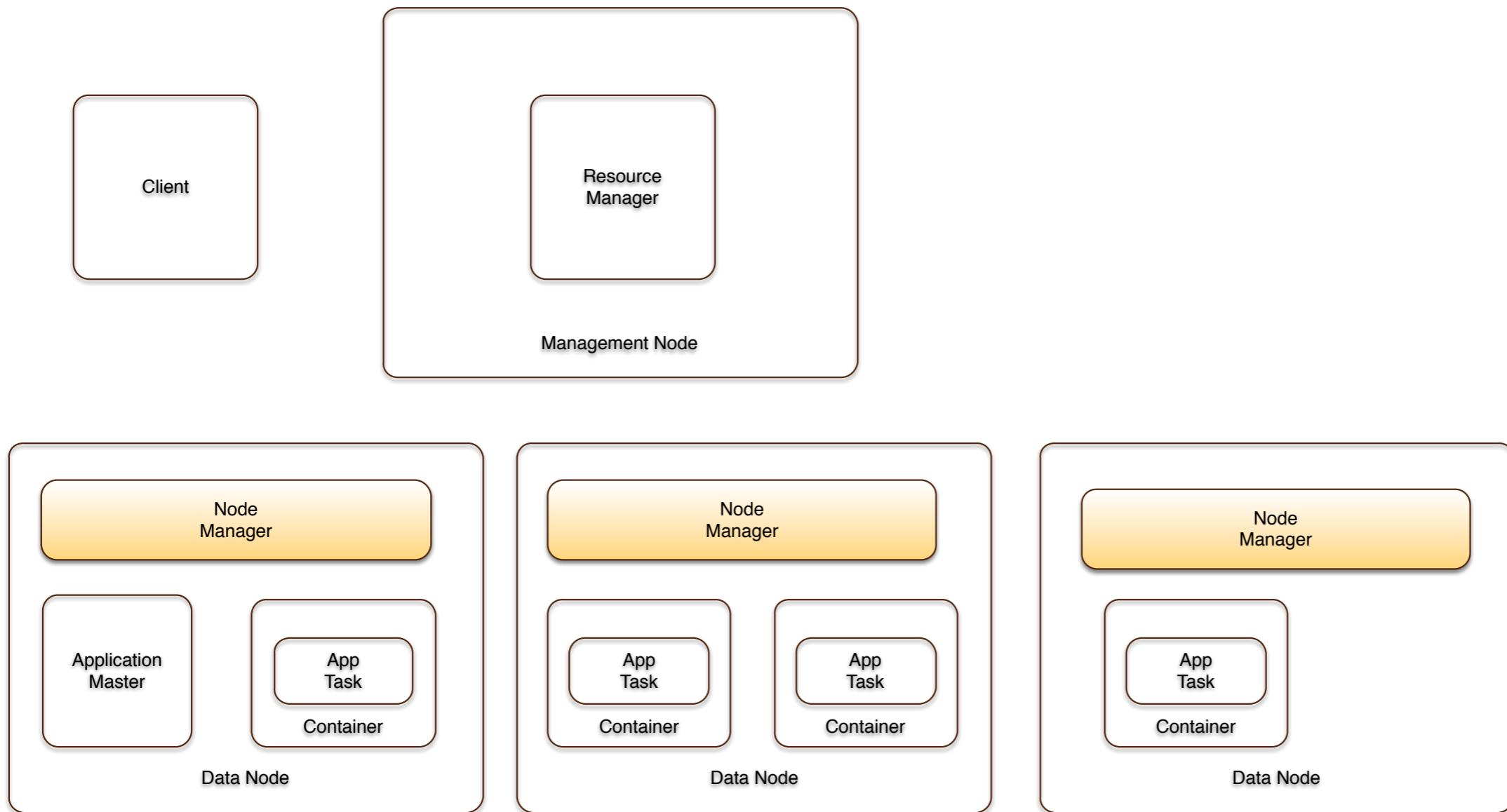


# Each app instance spawns an **application manager** (container 0)

- to negotiate resource and monitor app progress (tasks)

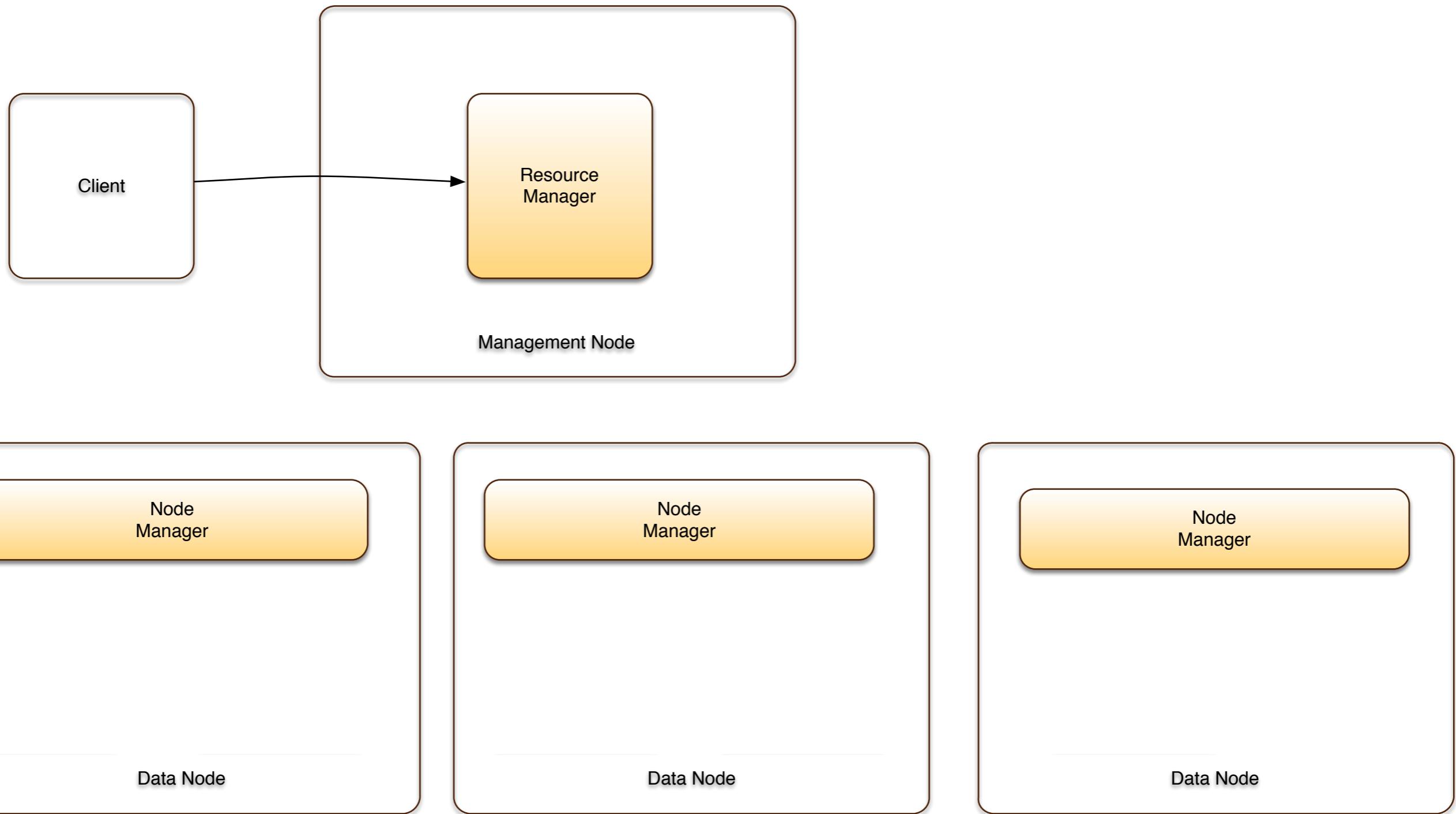


# Node managers monitor nodes and manage containers lifecycle

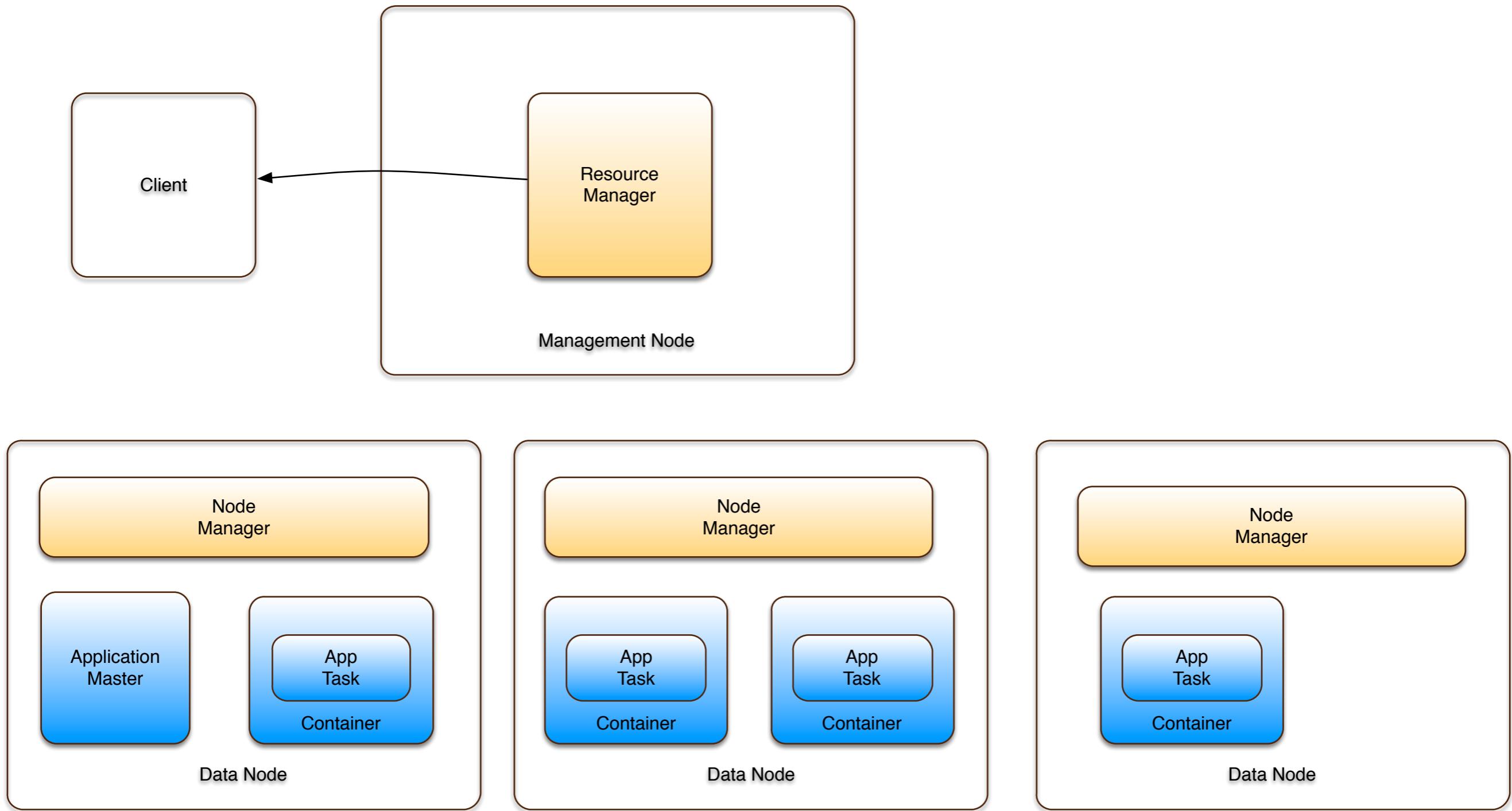


# Application Initiation

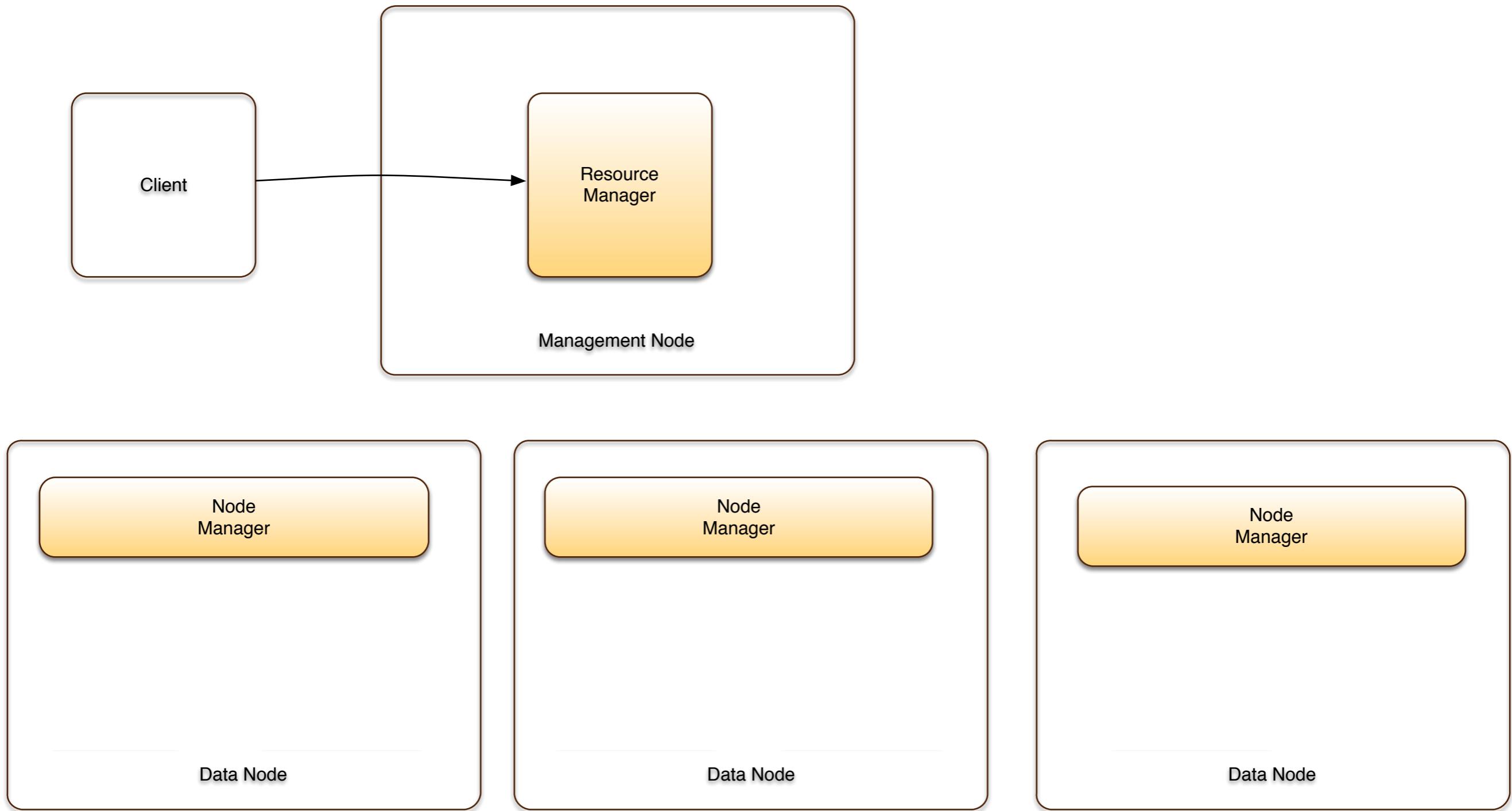
(or “how to get an App running in 11 easy steps”)



1. Client submits a job/app

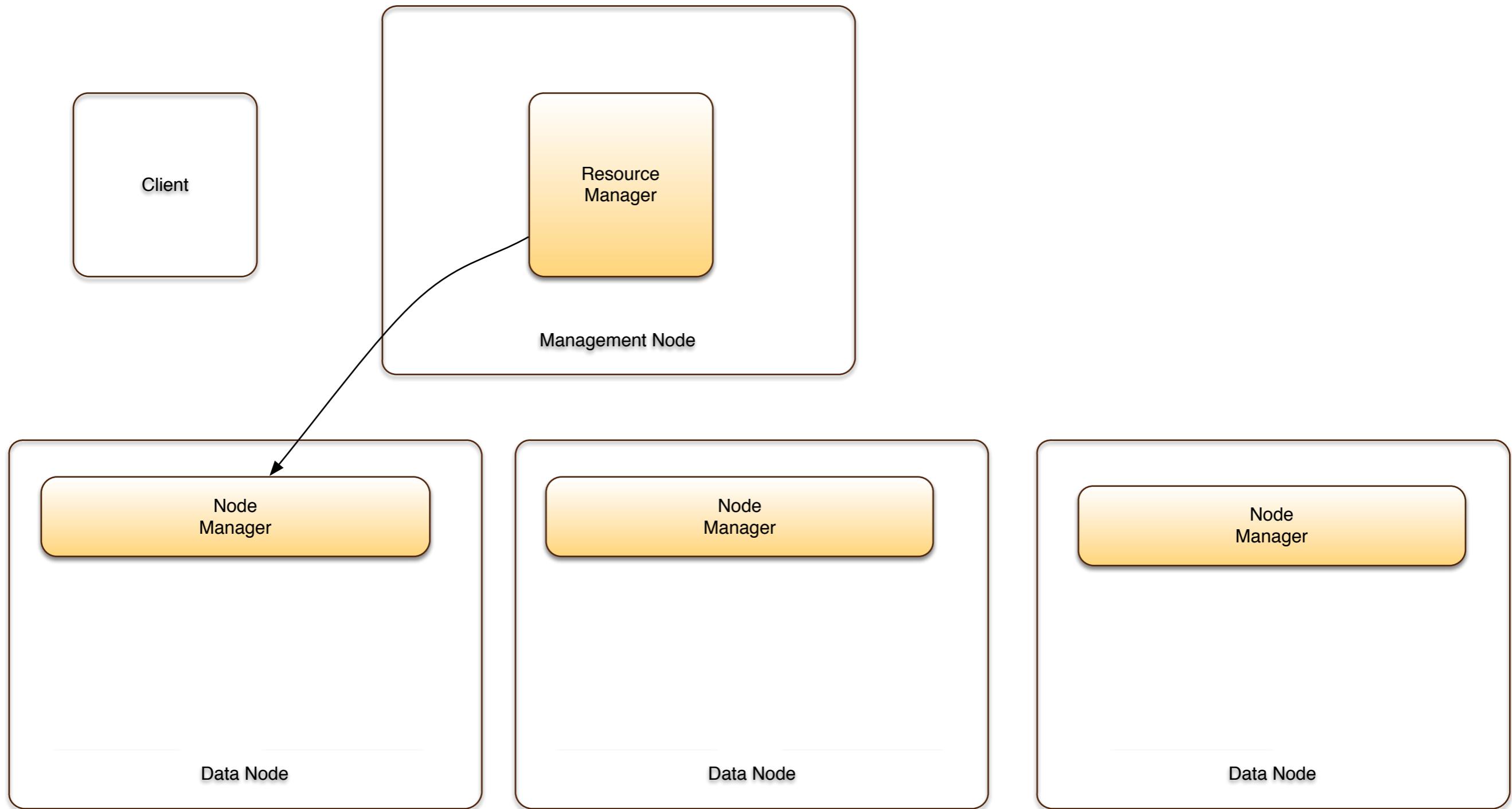


## 2. Resource Manager (RM) provides Application Id

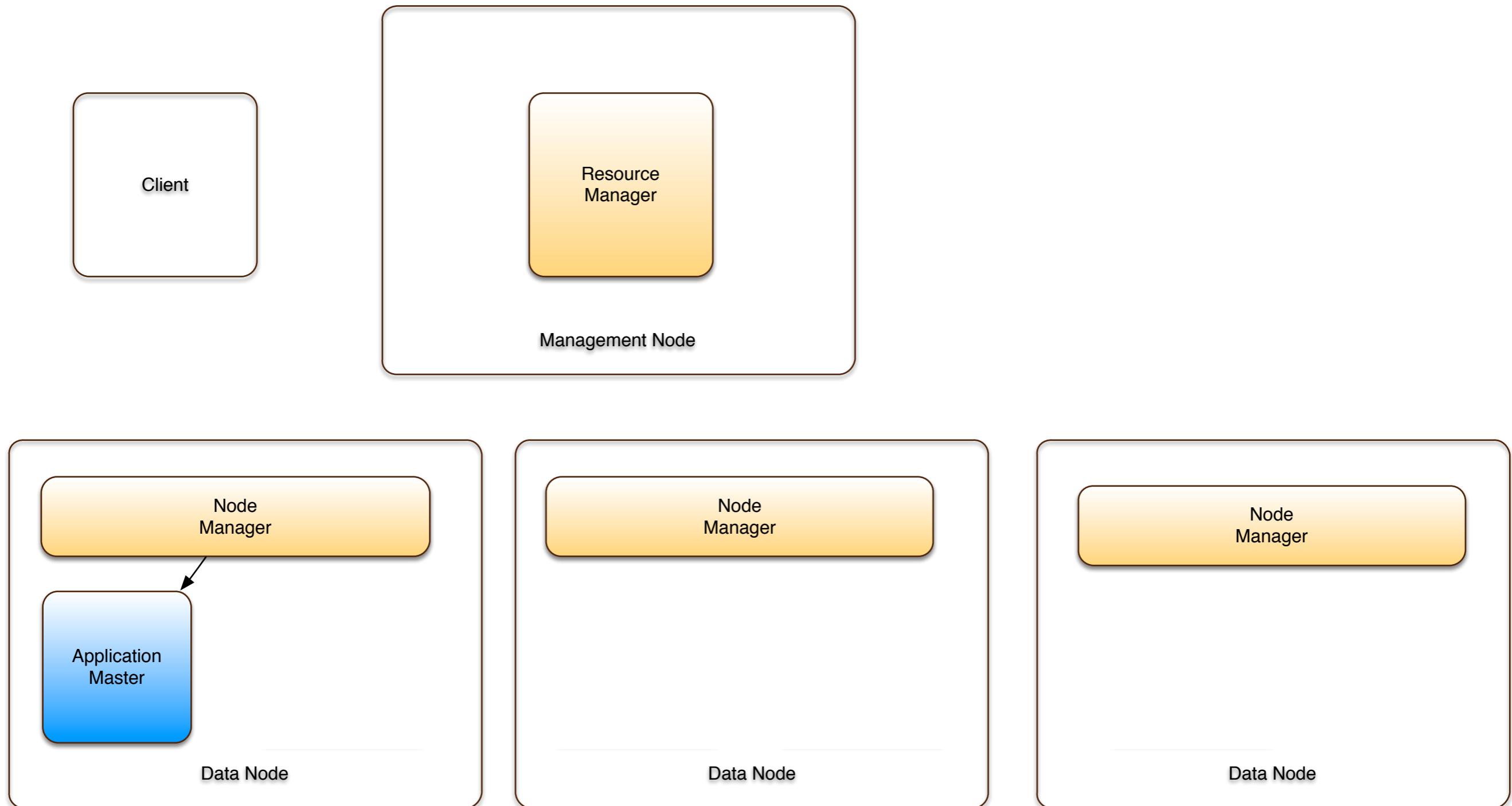


### 3. Client provides context

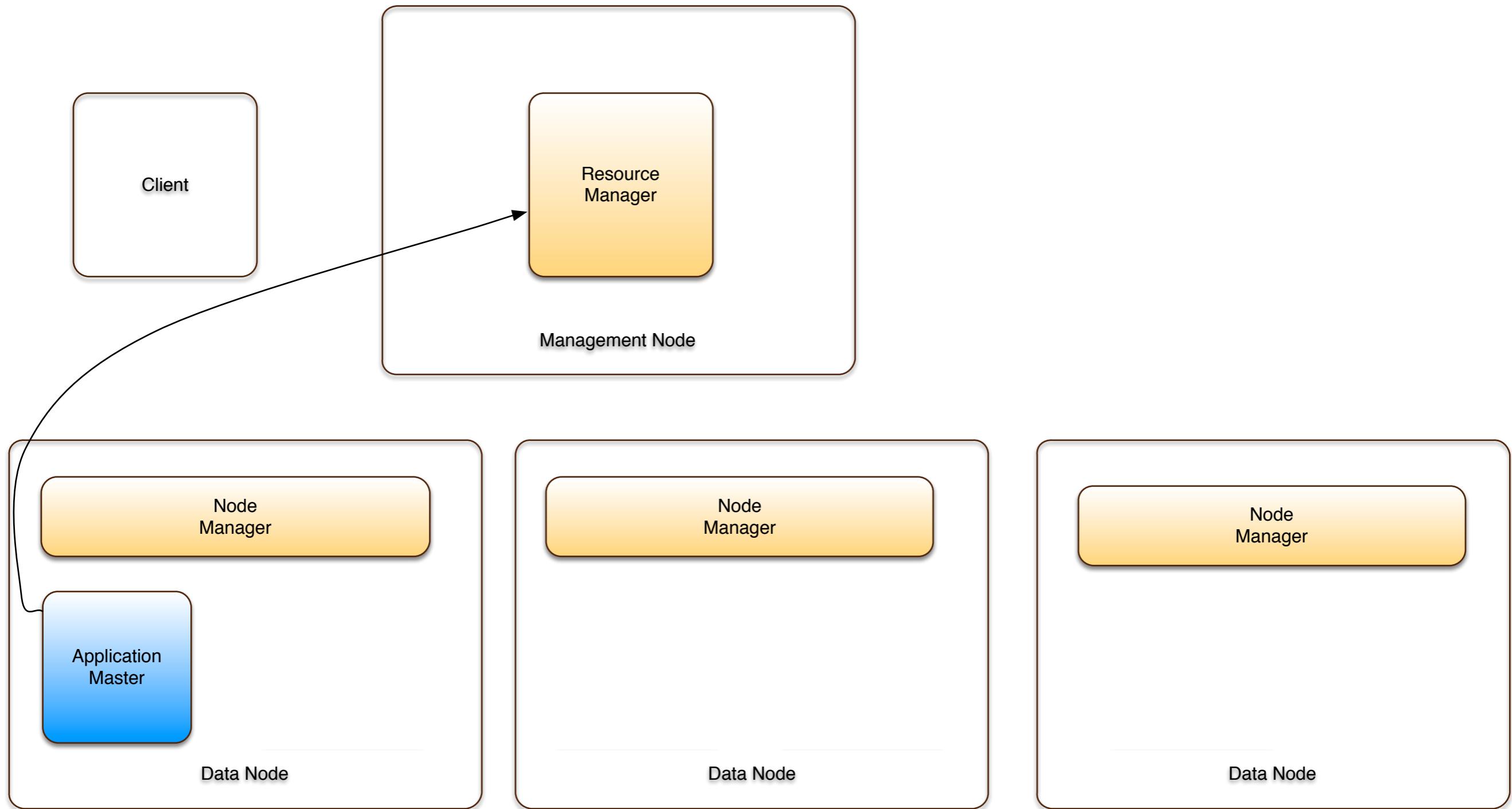
(queue, resource requirements, files, security tokens etc.)



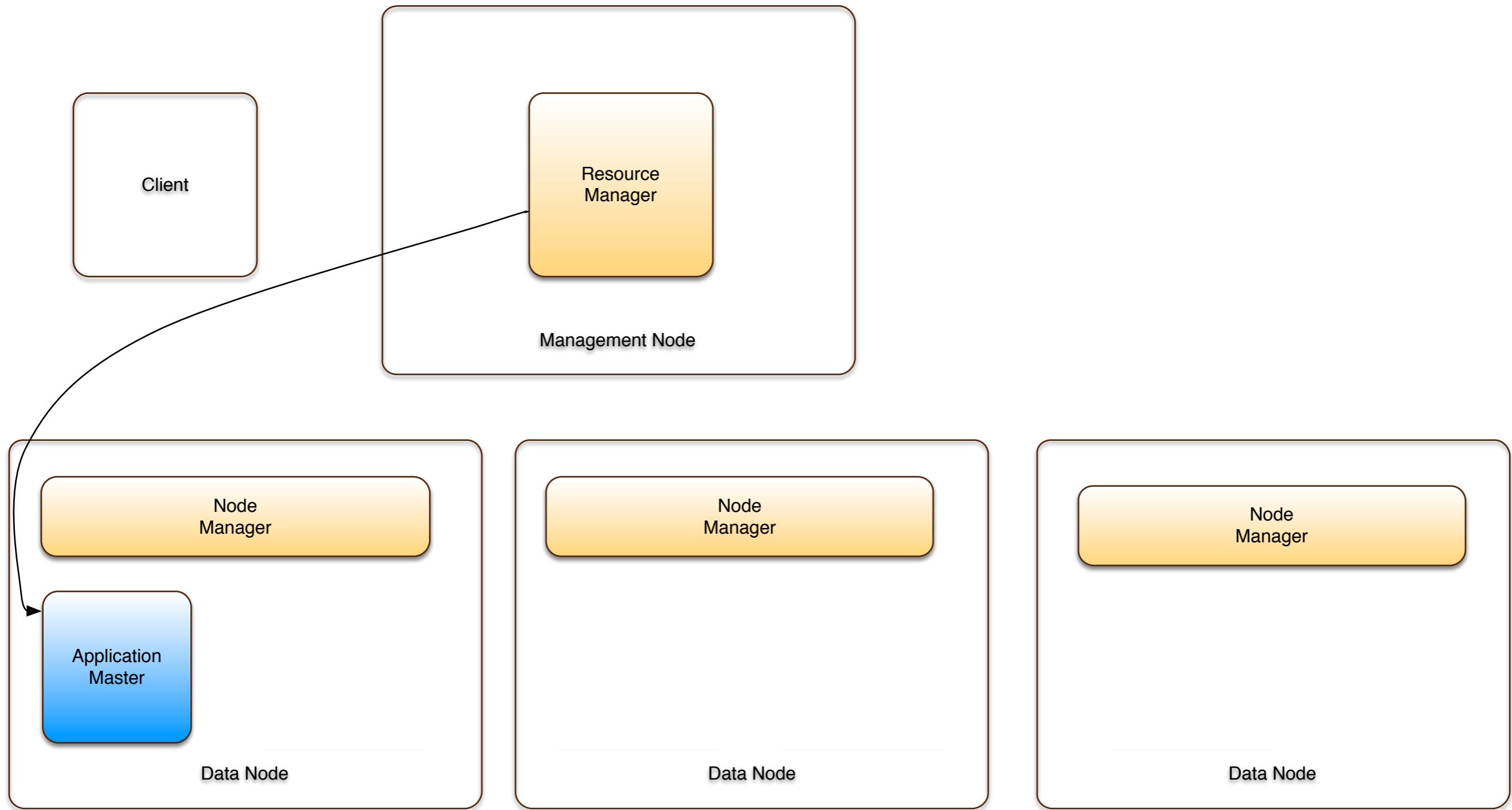
4. RM asks Node Manager  
to launch Application Master



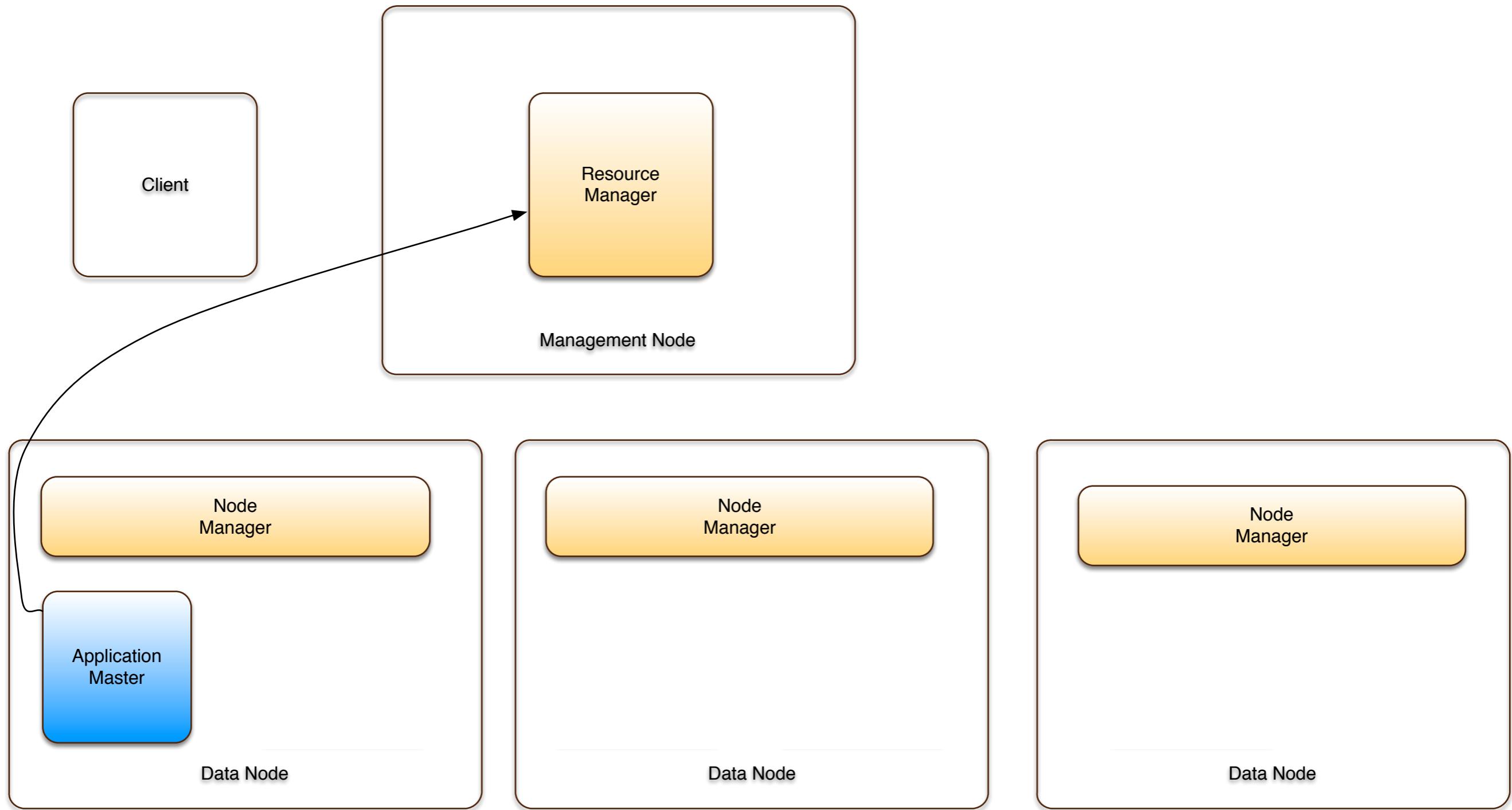
## 5. Node Manager launches Application Master



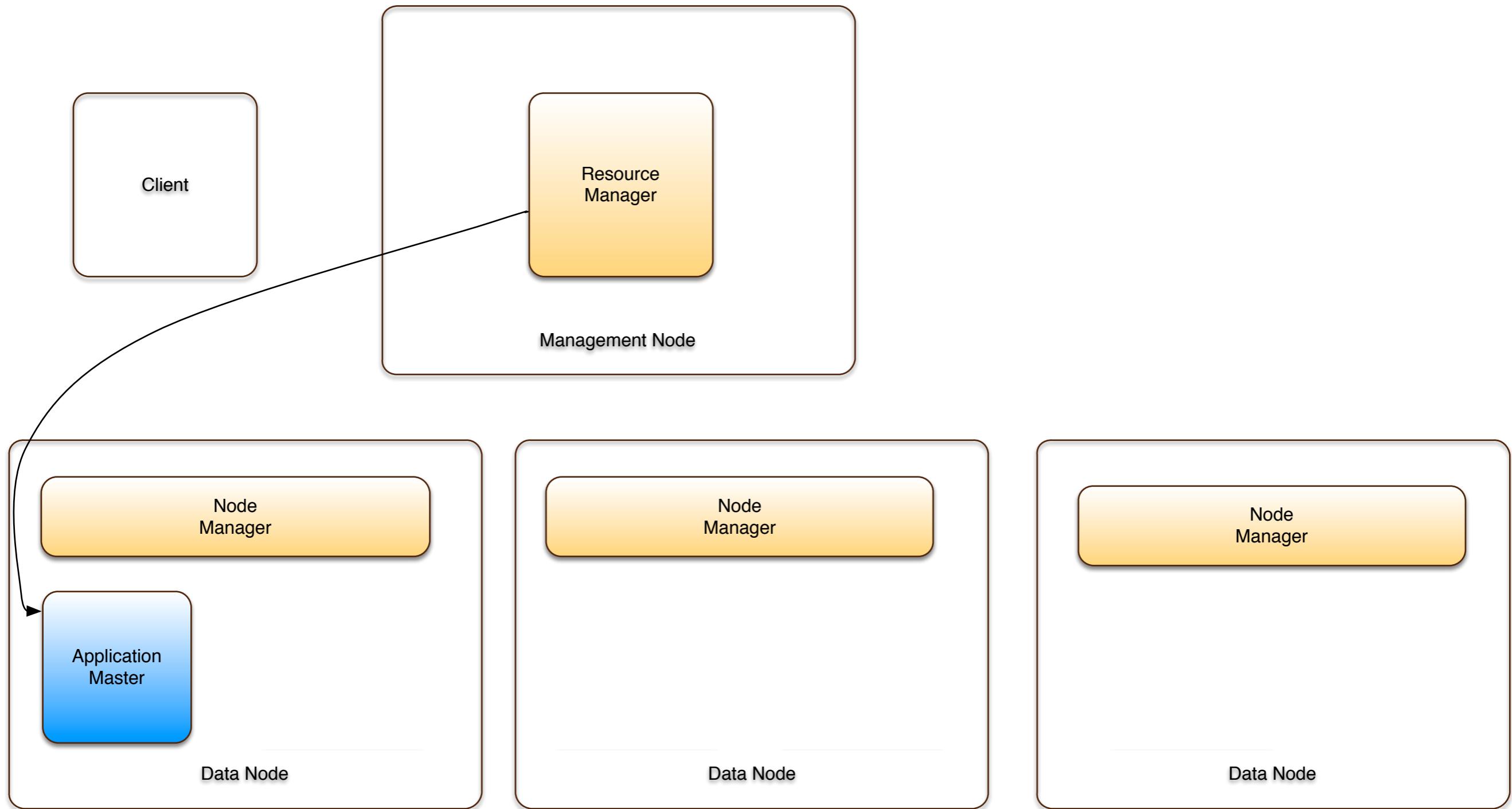
# 6. Application Master registers with RM



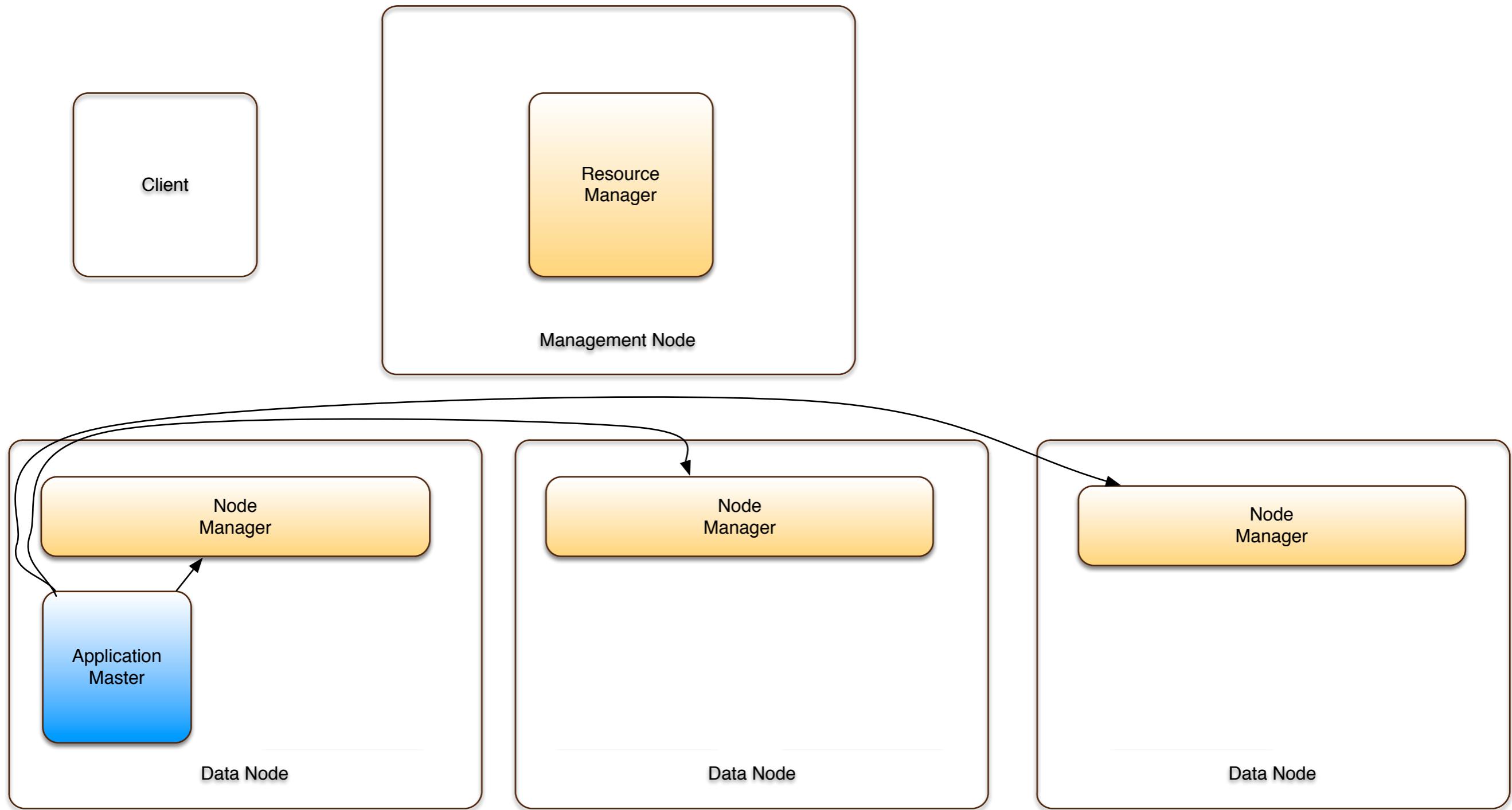
7. RM shares resource capabilities  
with Application Master



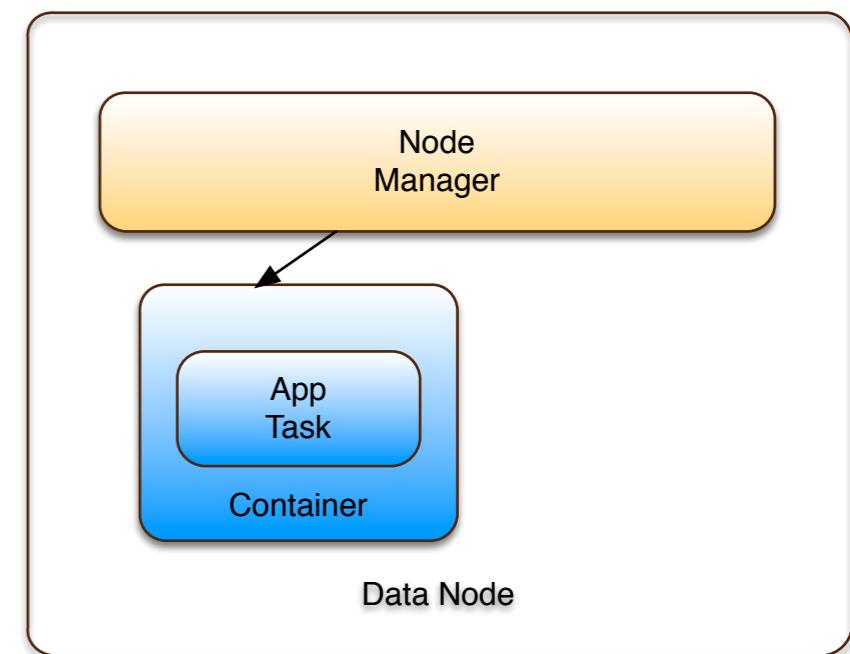
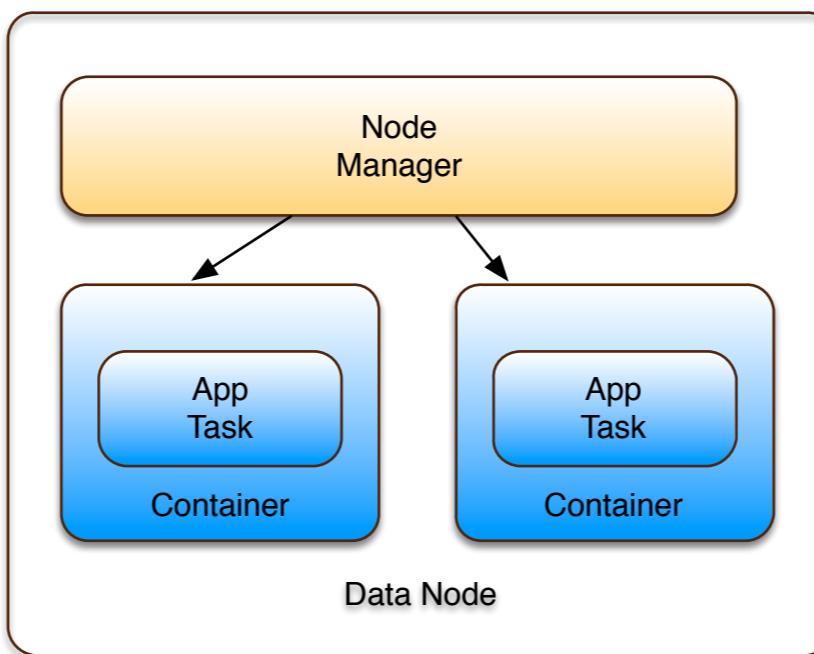
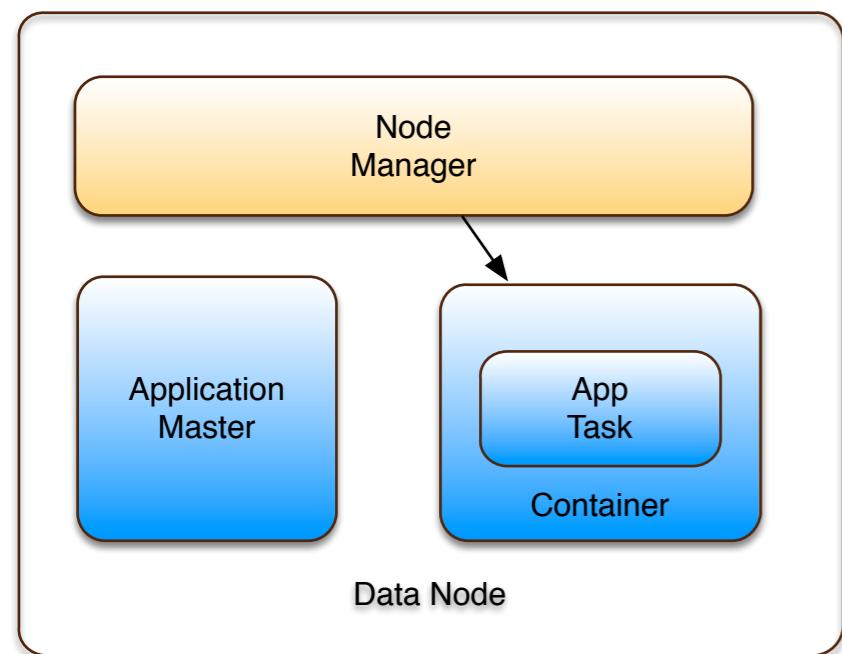
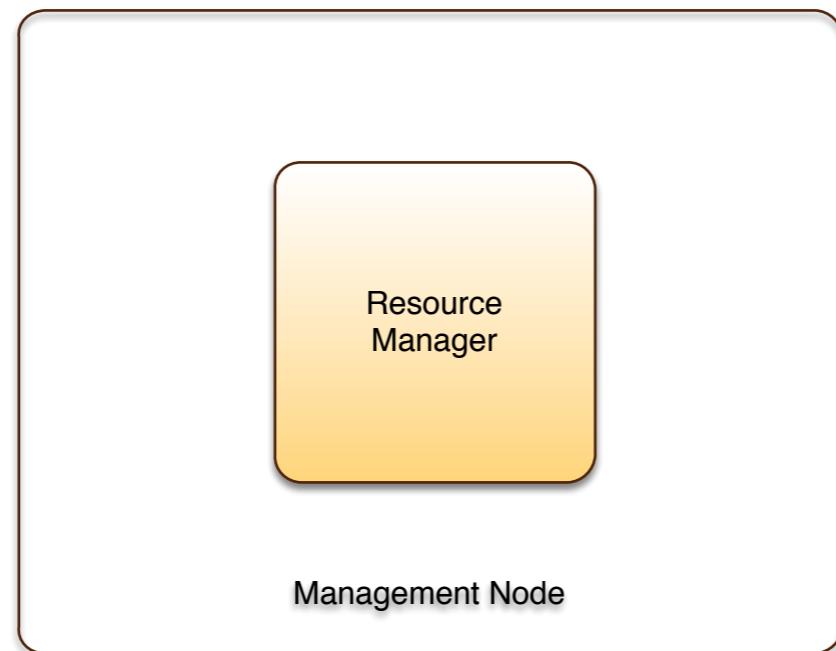
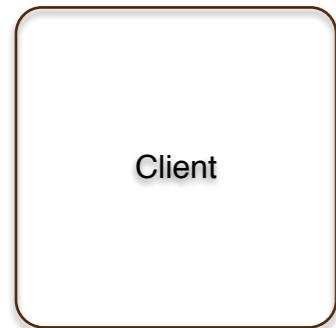
# 8. Application Master requests containers



9. RM assigns containers based on policies and available resources



10. Application Master contacts assigned node mageres to instantiate containers  
(passing container contexts)



# 11. Node Manager initiate container(s)

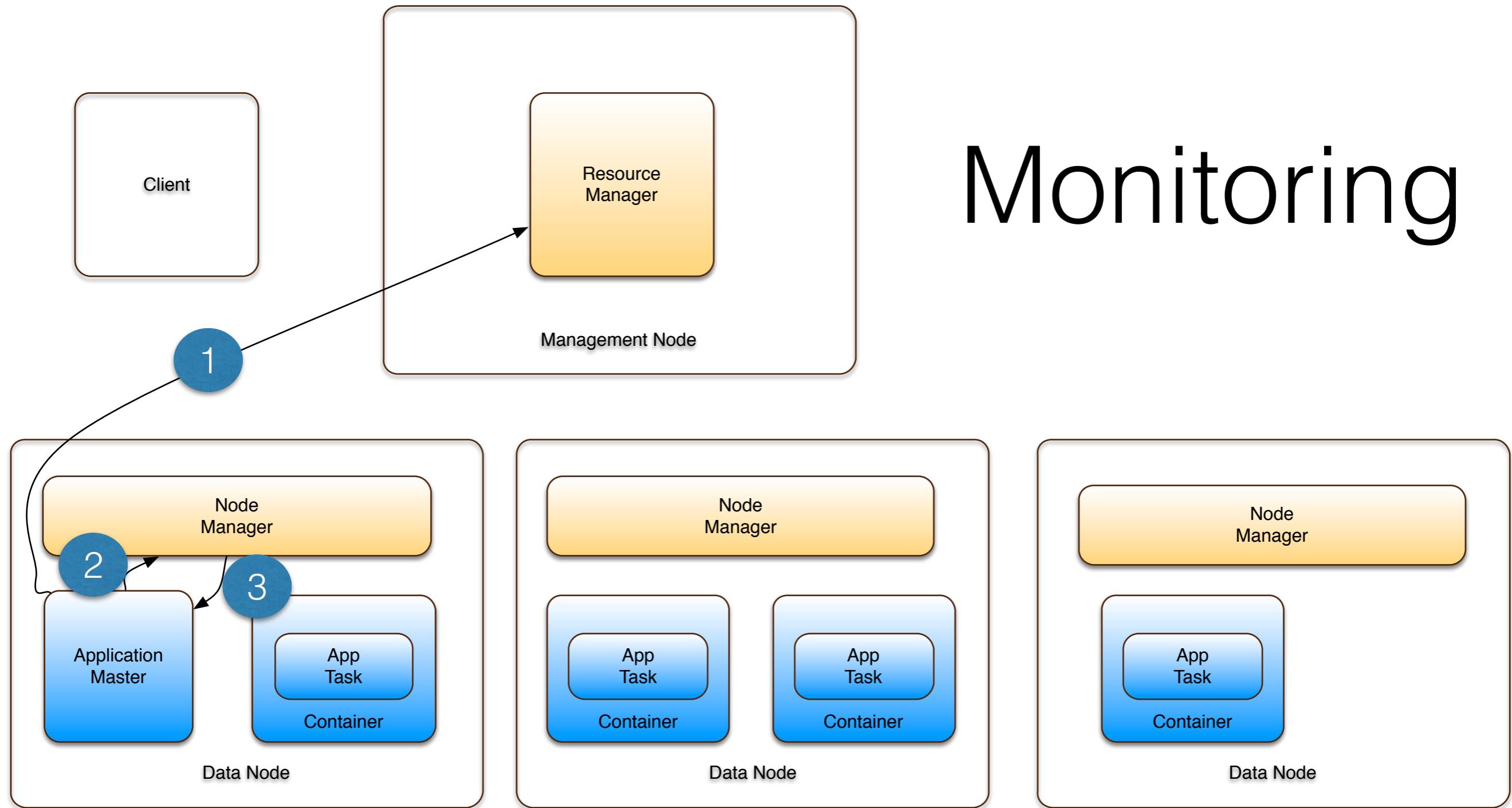
A vibrant fireworks display against a dark sky. The fireworks are in various colors including red, green, pink, yellow, blue, and orange. They are exploding in different patterns, some as large spherical bursts and others as smaller, more scattered sparks. The overall effect is a festive and celebratory atmosphere.

Congratulations your  
Application is now  
running

# Application Progress

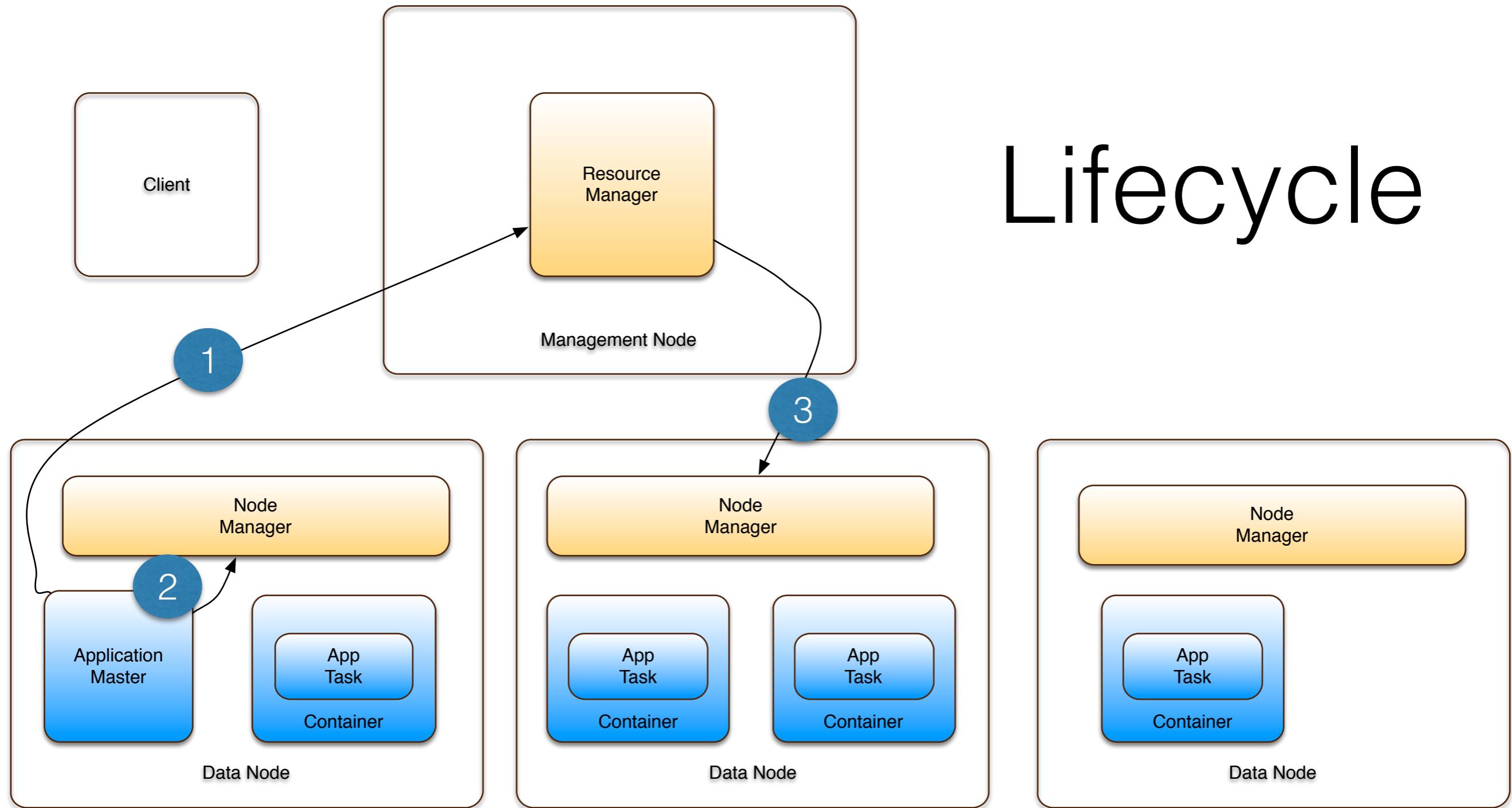
(or “it doesn’t end there”)

# Monitoring



1. continuous heartbeat & progress report
2. Request container status
3. Status response

# Lifecycle



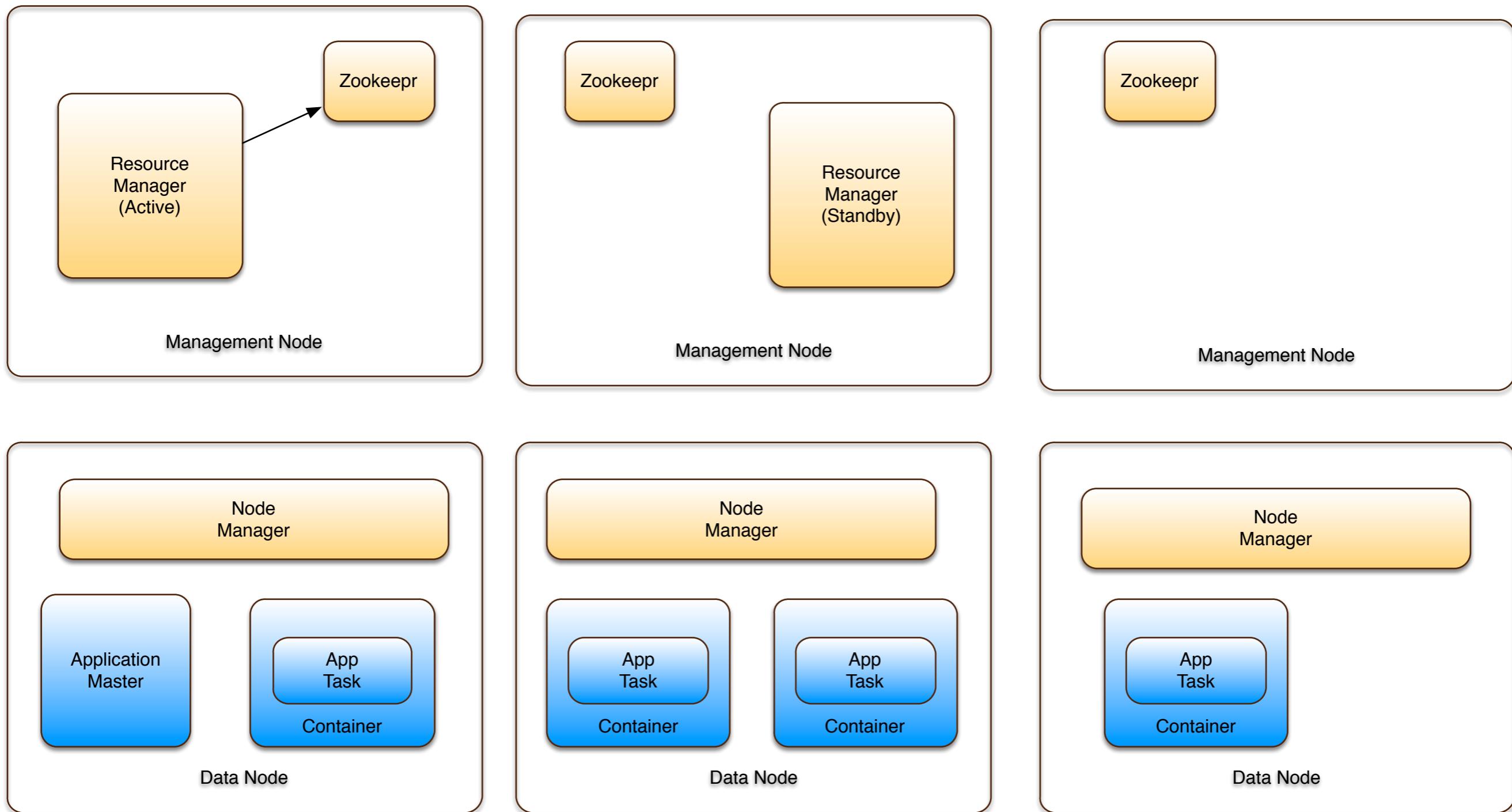
1. Heartbeat also carries request for new container allocations / container releases
2. Application master connects to node manger to activate allocated containers
3. Container releases go through the Resource Manager

# YARN HA

(or “you really didn’t think that was all, did ya?”)

# Still a work in progress

- Resource Manager - YARN 149 (patch available)
- Application Manager - YARN 1489
- Node Manager - YARN 1336



Active/Standby  
Store state in ZooKeeper  
Use ZooKeeper for failover management

YARN Limitation:  
Manages memory & CPU  
but not Disk IO or Network

# YARN Limitations: Batch Focus



**Small Batch**



**Large Batch**

# Black Legion



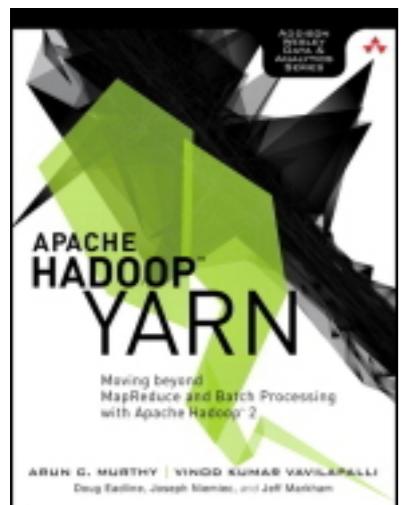
Daemon Prince

YARN Limitation: Relatively complex to develop for

# Further Reading



**YARN Source code** <https://github.com/apache/hadoop-common/tree/trunk/hadoop-yarn-project/hadoop-yarn>



**Apache Hadoop YARN: Moving beyond MapReduce and Batch Processing with Hadoop 2** by Arun C. Murthy, Vinod Kumar Vavilapalli, Doug Eadline, Joseph Niemiec & Jeff Markham

**Apache YARN site** <http://hadoop.apache.org/docs/r2.3.0/hadoop-yarn/hadoop-yarn-site/YARN.html>



# Image attributions:

slide 1 - Hadoop logo - <http://hadoop.apache.org/docs/current/>

slide 3 - adopted from pulp fiction

slide 4-5 <http://pypr.sourceforge.net/kmeans.html>

slide 10 - Elizabeth Moreau <http://bornlibrarian.blogspot.co.il/2011/01/christmas-knitting.html>

slide 11 <http://hortonworks.com/hadoop/yarn/>

slide 14 <http://developer.yahoo.com/blogs/ydn/posts/2007/07/yahoo-hadoop/>

slide 32 <https://upload.wikimedia.org/wikipedia/commons/a/a7/ColorfulFireworks.png>

slide 39 <http://justdan93.wordpress.com/2012/04/22/resource-categories/>

slide 40 - <http://dev2ops.org/2012/03/devops-lessons-from-lean-small-batches-improve-flow/>

slide 41 [http://fc00.deviantart.net/fs71/i/2012/064/1/0/black\\_legion\\_daemon\\_prince\\_by\\_knyghtos-d4rtwhr.png](http://fc00.deviantart.net/fs71/i/2012/064/1/0/black_legion_daemon_prince_by_knyghtos-d4rtwhr.png)

Slide 42 By Christina Quinn <https://www.flickr.com/photos/chrisser/7909860048/>