COSI 129a

Introduction to Big Data Analysis

Fall 2016

Map Reduce - Hadoop

- A Java-based open-source implementation of MapReduce.

- Comes with an underlying distributed file system and various other tools for monitoring, etc.

- You will use Hadoop as your development platform in your assingments.

Why is it useful?

- **Scalable:** It can reliably store and process petabytes.
- **Economical:** It distributes the data and processing across clusters of 1000s commonly available computers.
- **Efficient:** By distributing the data, it can process it in parallel **on the nodes where the data is located.**
- **Reliable:** It automatically maintains multiple copies of data and automatically redeploys computing tasks based on failures.

Hadoop Map Reduce – What is it?

- Processing engine of Hadoop
- Developers create Map and Reduce Jobs
- Used for big data batch processing
- Parallel processing of huge data volumes
- Fault tolerant
- Scalable

Hadoop Map Reduce – Why use it?

- Your data is Terabyte/Petabyte scale
- You have huge I/O
- Hadoop framework takes care of:
  - Job and task management
  - Failures
  - Storage
  - Replication
- You just write Map and Reduce jobs

Related Projects

- Pig: for analyzing large data sets
- Hive: data warehouse system for Hadoop
- Mahout: machine learning and data mining
- Avro: a data serialization system
- Zoo Keeper: helps build distributed applications
- Hue: Hadoop user interface
- Hbase: Non relational database

Large Users

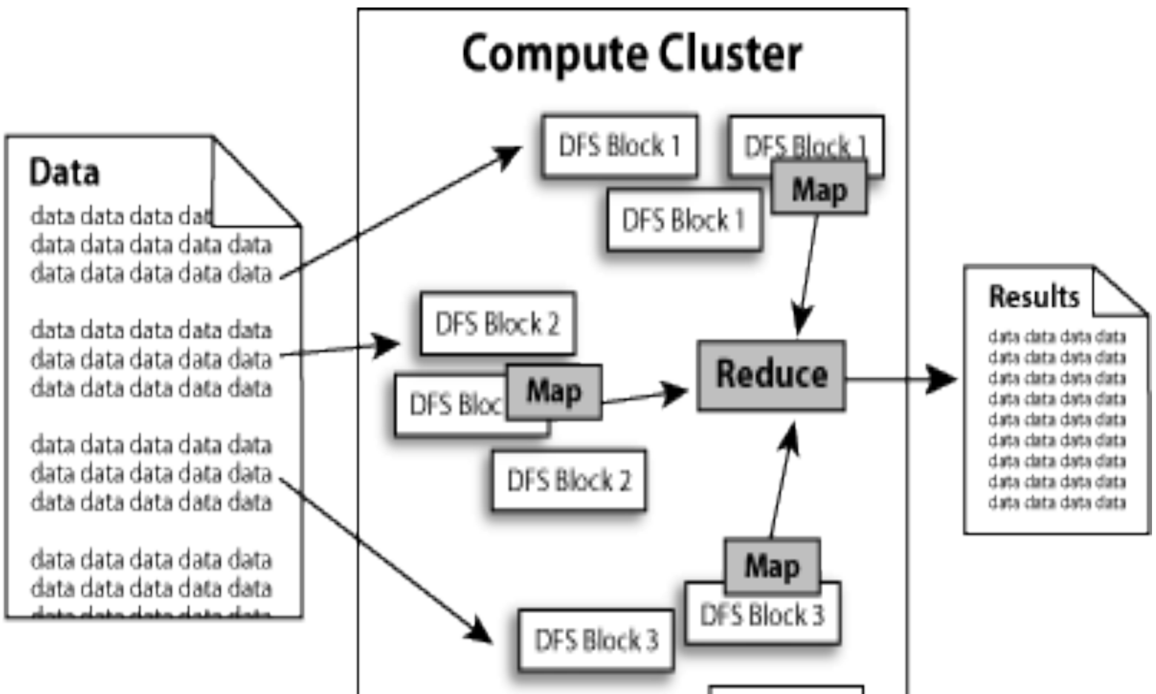
- Yahoo
  - 10,000 core Linux cluster
- Facebook
  - 100 Petabytes, growing at 0.5 Petabytes a day
- Amazon
  - Its possible to run Hadoop on Amazon's EC2 and S3

Hadoop Consists Of:

- Hadoop MapReduce
  - Parallel processing of Hadoop data
- Hadoop Distributed File System (HDFS)
  - A master/slave file system
- Hadoop Yarn
  - Scheduler and Resource Manager
  - Added in Hadoop 2.0

Hadoop-What does it do?

- Hadoop implements Google's MapReduce, using HDFS
- MapReduce divides applications into many small blocks of work.
- HDFS creates multiple replicas of data blocks for reliability, placing them on compute nodes around the cluster.
- MapReduce can then process the data where it is located.



Old Hadoop Architecture

Software Architecture

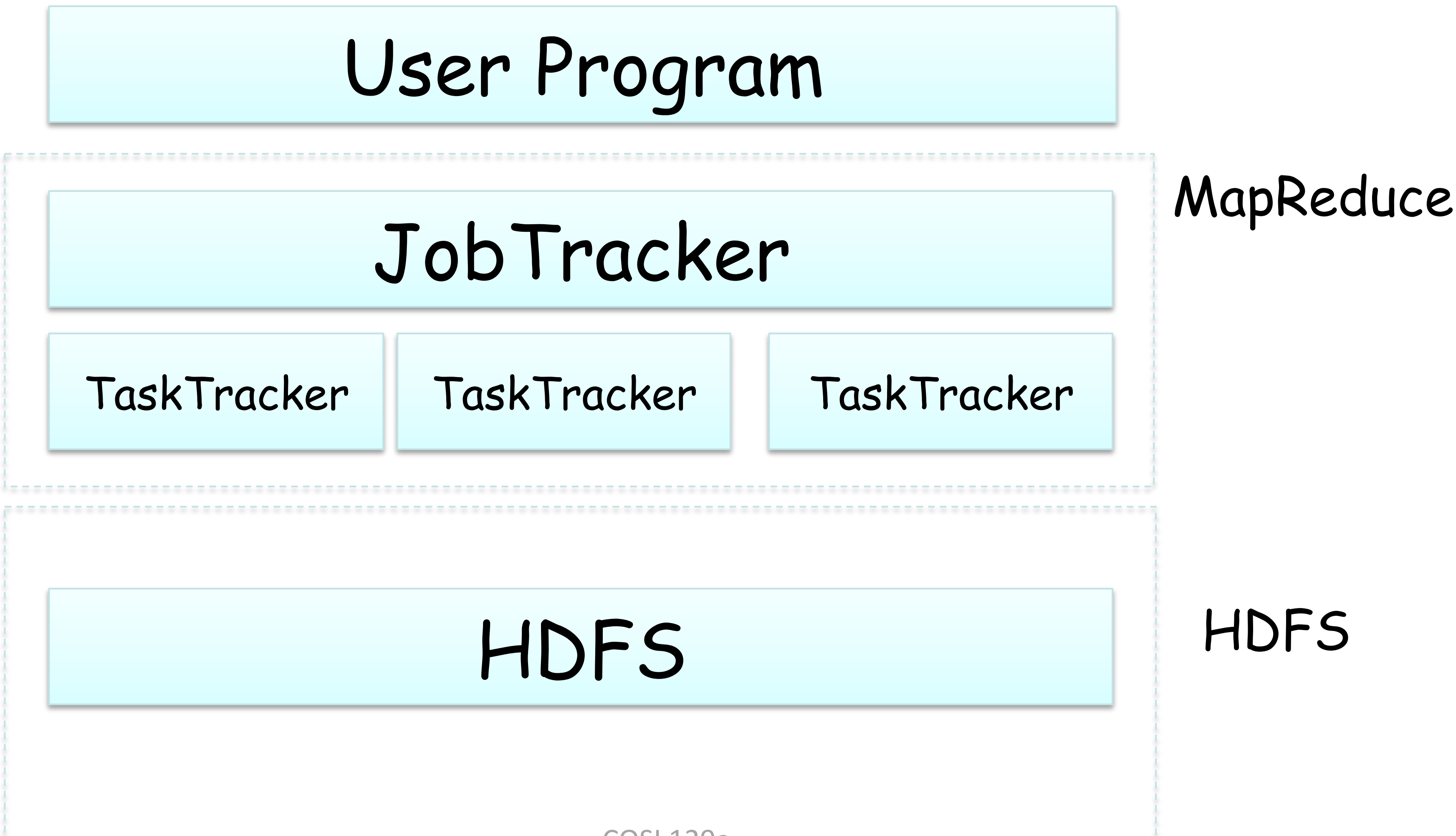
User Program

Hadoop

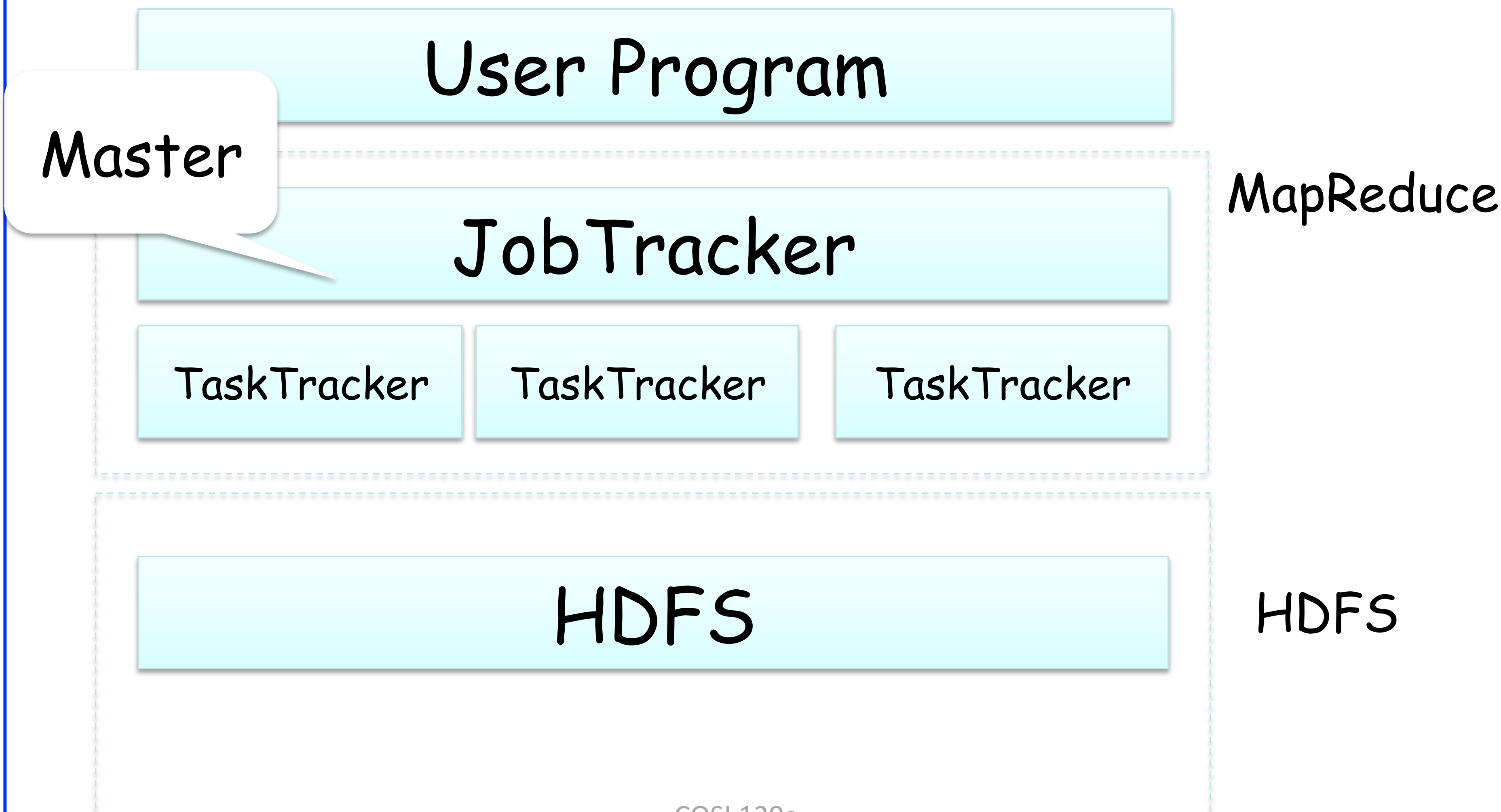
MapReduce

HDFS

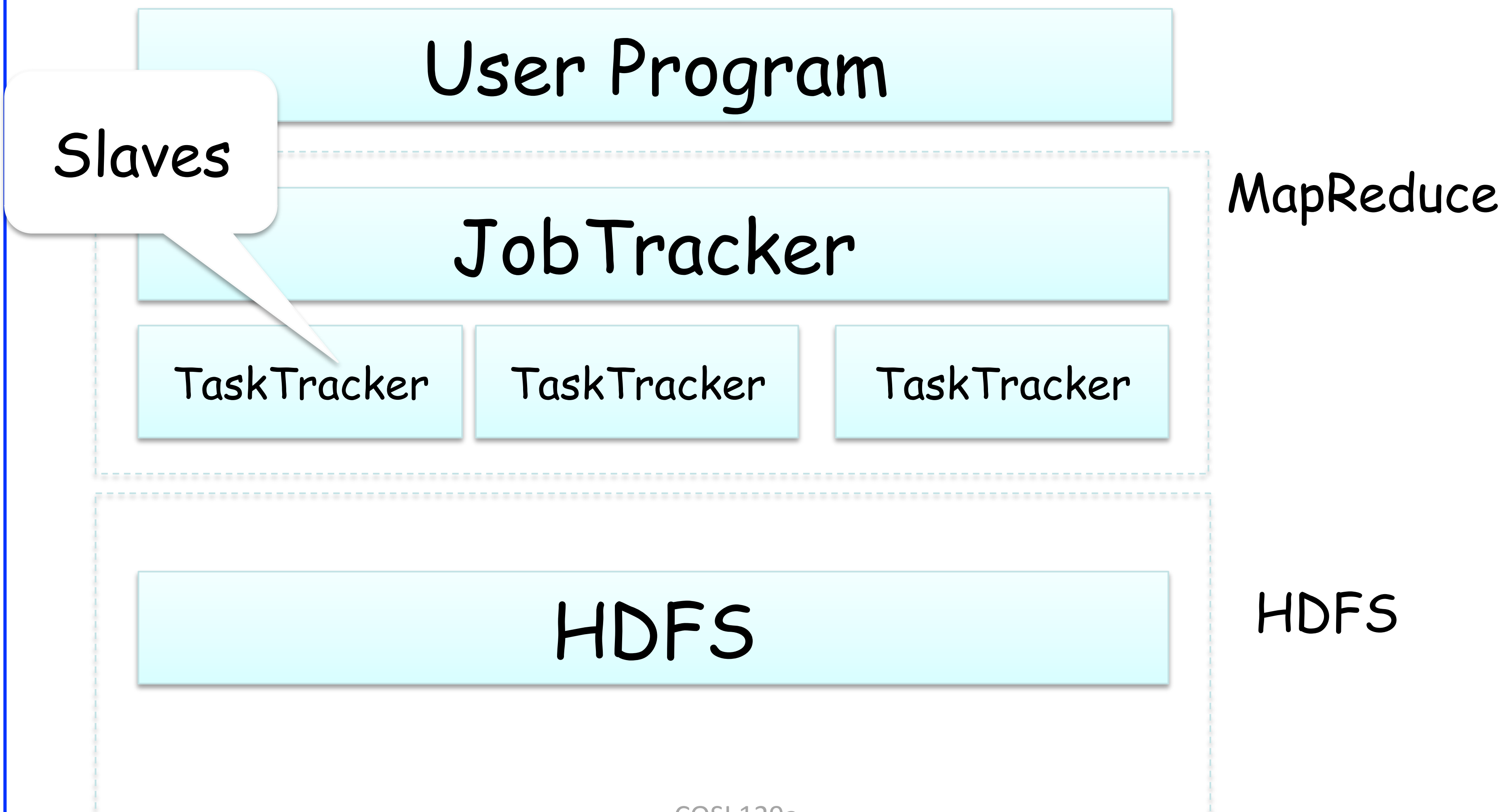
Software Architecture



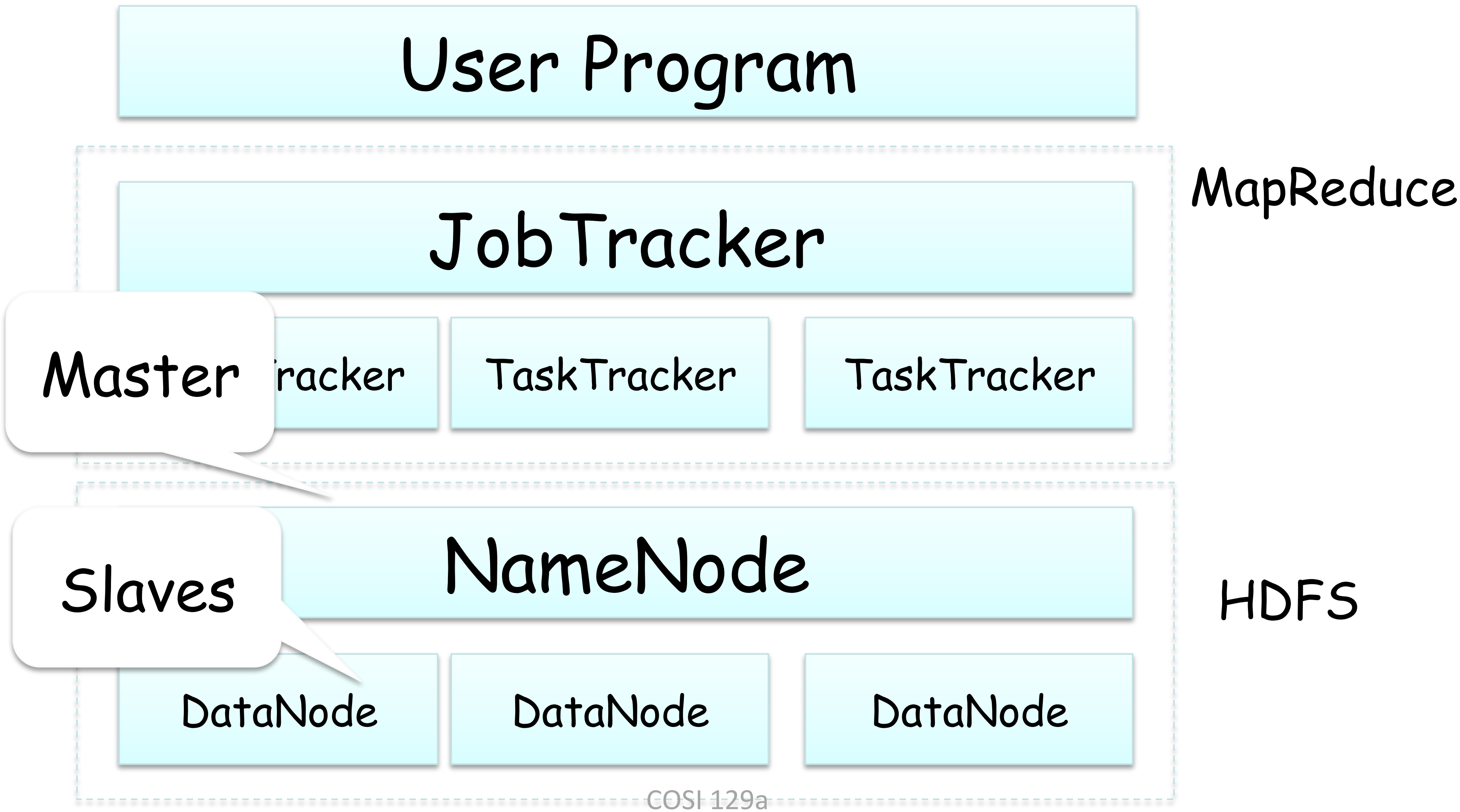
Software Architecture



Software Architecture



Maste-Slave Architecture



Daemons

- A running Hadoop instance is comprised of multiple Hadoop *daemons threads*
  - One thread is the master, the rest are workers/slaves
- Each daemon serves a single role, and communicates only with its master
- Hadoop daemons may run in separate processes on the same node or distributed over multiple nodes

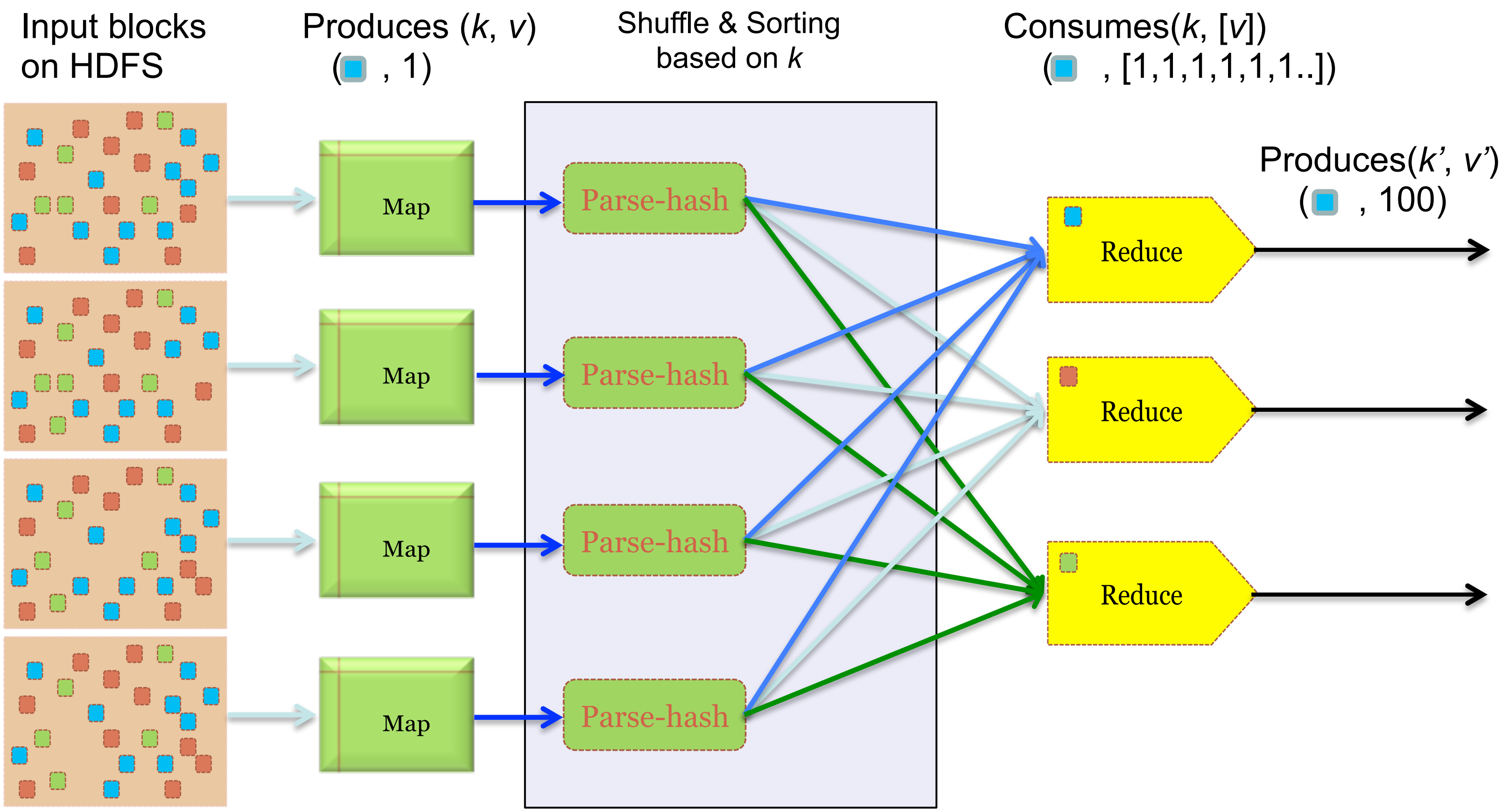
Daemons

- When you run a MapReduce job, it is managed by the **JobTracker** daemon (one per cluster):

- Each job is comprised of multiple tasks (map tasks, reduce tasks, etc.), each is given to a **TaskTracker** daemon (many per cluster)

- Data comes from the Hadoop Distributed File System (HDFS);
    - we'll get to HDFS later, but basically it is comprised of a NameNode and many DataNodes

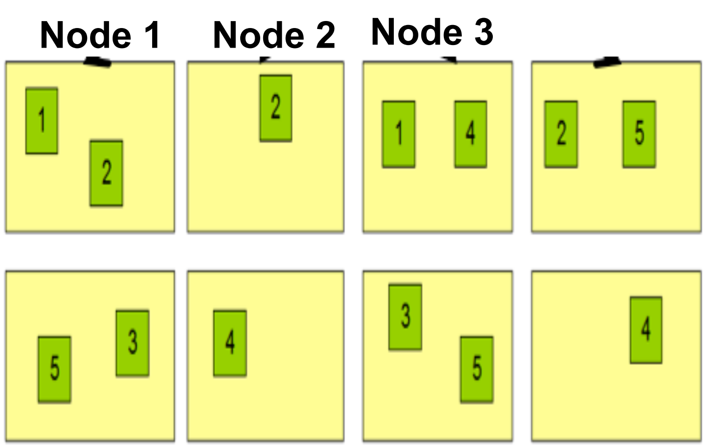
Map-Reduce Execution Engine (Example: Color Count)



Users only provide the "Map" and "Reduce" functions

Properties of MapReduce Engine

- **Job Tracker is the master node (runs with the NameNode)**
  - Receives the user's job
  - Decides on how many tasks will run (number of mappers/reducers)
  - Decides on where to run each mapper (concept of locality)

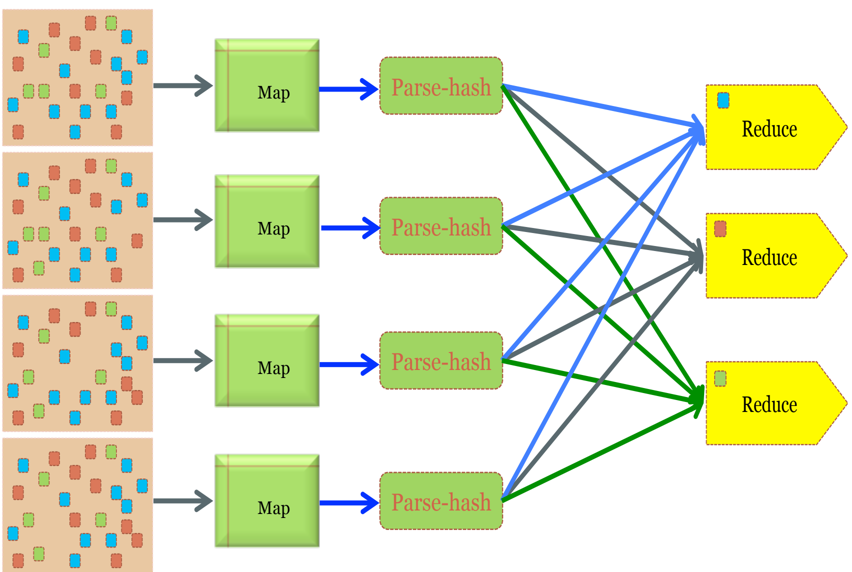


Q: Where to run the task reading block"1" ?

A: Try to run it on Node 1 or Node 3

Properties of MapReduce Engine (Cont'd)

- **Task Tracker is the slave node (runs on each DataNode)**
  - Receives the task from Job Tracker
  - Runs the task until completion (either map or reduce task)
  - Always in communication with the Job Tracker reporting progress



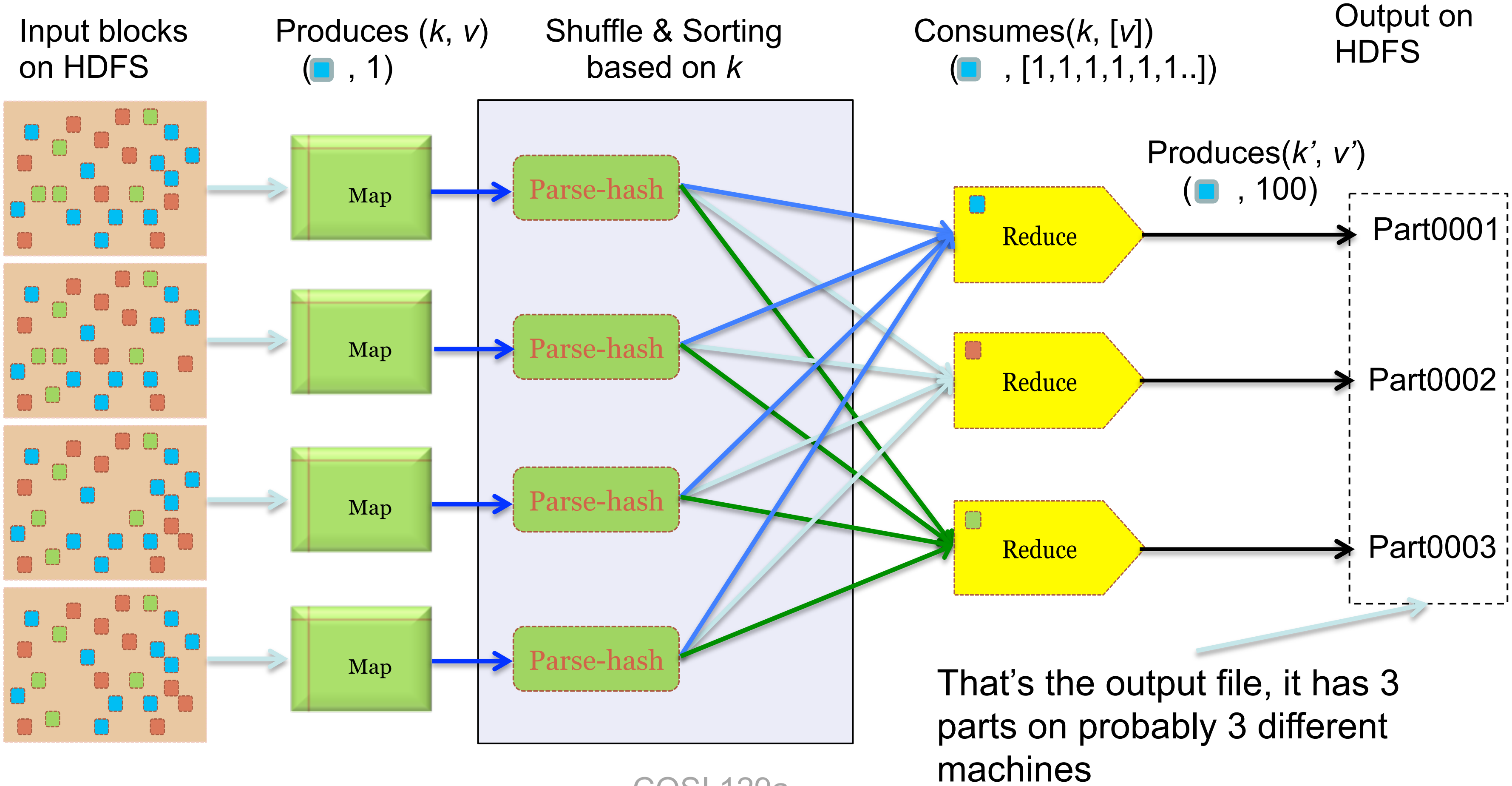
In this example, 1 map-reduce job consists of 4 map tasks and 3 reduce tasks

Key-Value Pairs

- Mappers and Reducers are users' code (provided functions)
- Just need to obey the Key-Value pairs interface
- **Mappers:**
  - Consume <in_key, value> pairs
  - Produce <**out_key**, value> pairs
- **Reducers:**
  - Consume <**out_key**, <list of values>>
  - Produce <final_key, value>
- **Shuffling and Sorting:**
  - Hidden phase between mappers and reducers
  - Groups all similar keys from all mappers, sorts and passes them to a certain reducer in the form of <key, <list of values>>

Example 2: Color Count

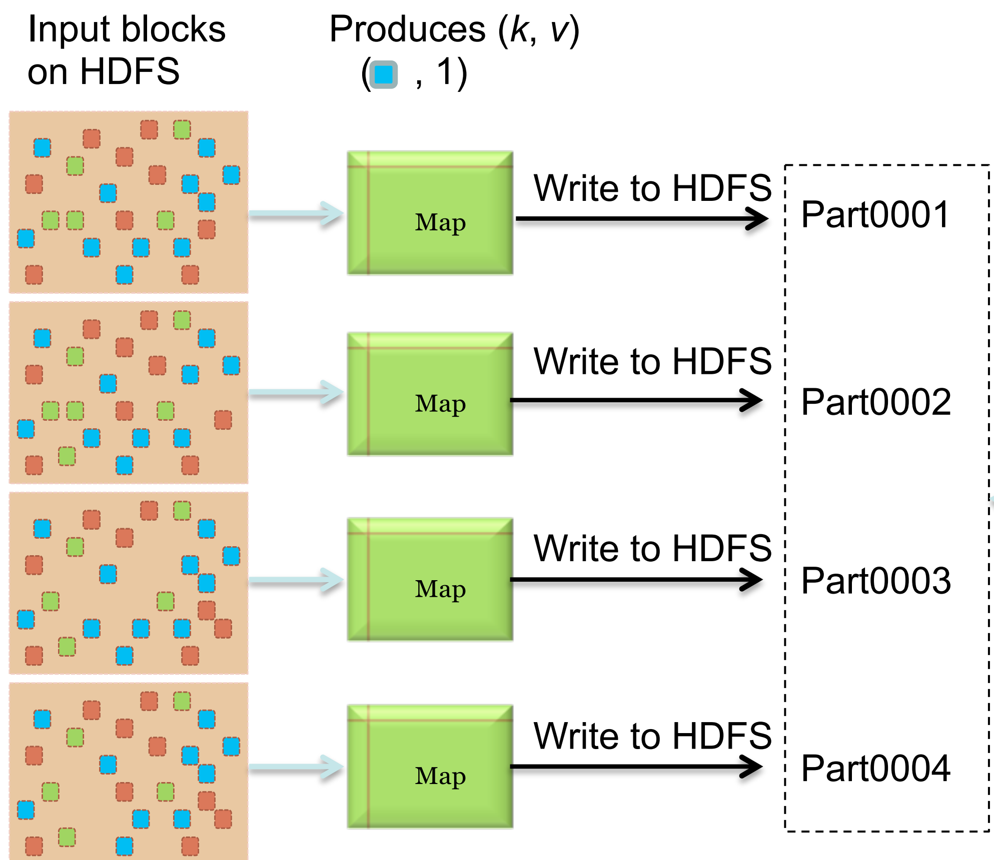
Job: Count the number of each color in a data set



Example 3: Color Filter

Job: Select only the blue and the green colo

- Each map task will select only the blue or green colors
- No need for reduce phase



That's the output file, it has 4 parts on probably 4 different machines

Hadoop Distributed File System (HDFS)

HDFS – What is it?

- It is a distributed file system
- Runs on low cost hardware
- It is open source
- Written in Java
- Fault tolerant
- Designed for large data sets
- Tuned for high throughput

HDFS – What is it for?

- Designed for batch processing
- Streaming access to data
- Large data sizes, i.e., Terabytes
- Highly reliable using data replication
- Supports very large node clusters
- Supports large files

HDFS- Architecture

- A block-structured file system:
  - Back-end: data stored in fixed size blocks
  - Front-end: uses an abstraction called "files" and a way to organize those files by their names
- HDFS stores its blocks on one or more DataNodes
- DataNodes store blocks of big size
  - Typically 64 MB (configurable)
  - This decreases the amount of metadata per file
  - Allows streaming access: large amounts of data are sequentially laid out to disk
- NameNode(s): name service that stores location of file blocks

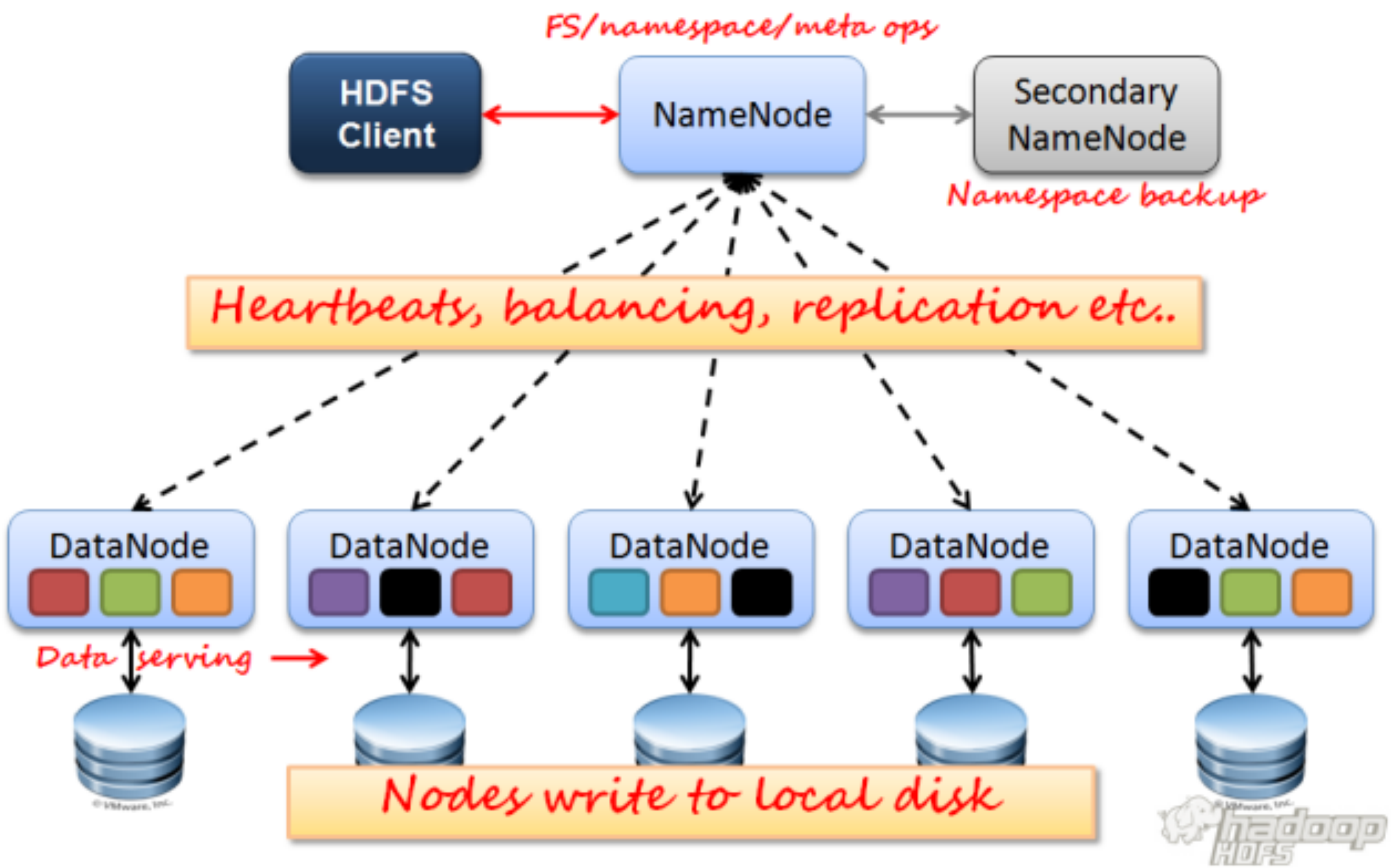
HDFS - Architecture

- Most modern file systems organize names hierarchically using directories and subdirectories
- The leaf elements in the file system hierarchy are files
  - /home/olga/courses/cs12/project.pdf
- HDFS uses the same naming scheme
- Files in HDFS are read-only

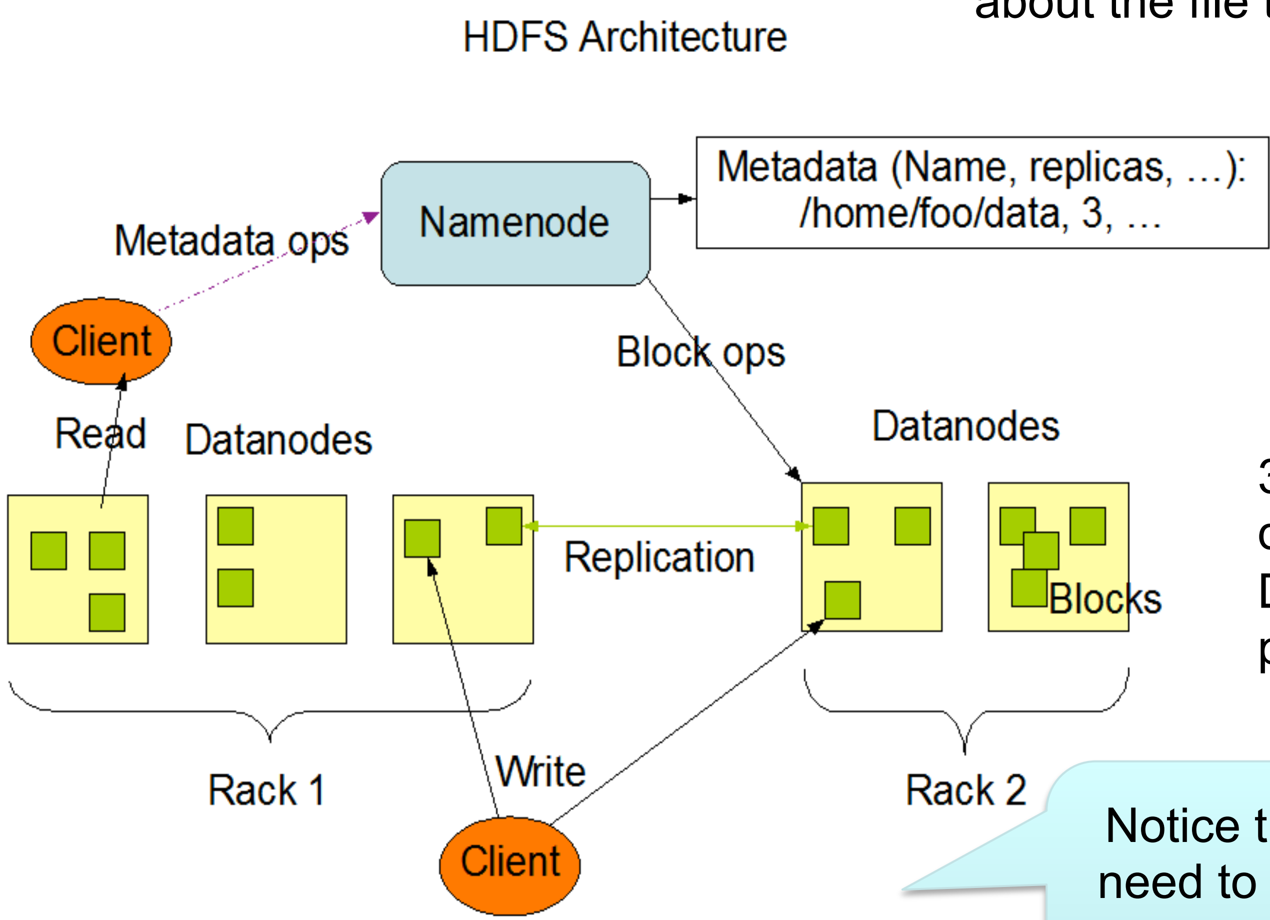
HDFS-Architecture

- Replication on the block level
  - That means the same block is stored in multiple places
  - The NameNode tracks all those places
- Offers fault tolerance:
  - DataNodes can crash without preventing access to files
- Offers data locality (and better I/O performance)
  - The computation is moved to the data when possible

HDFS Architecture



HDFS Architecture



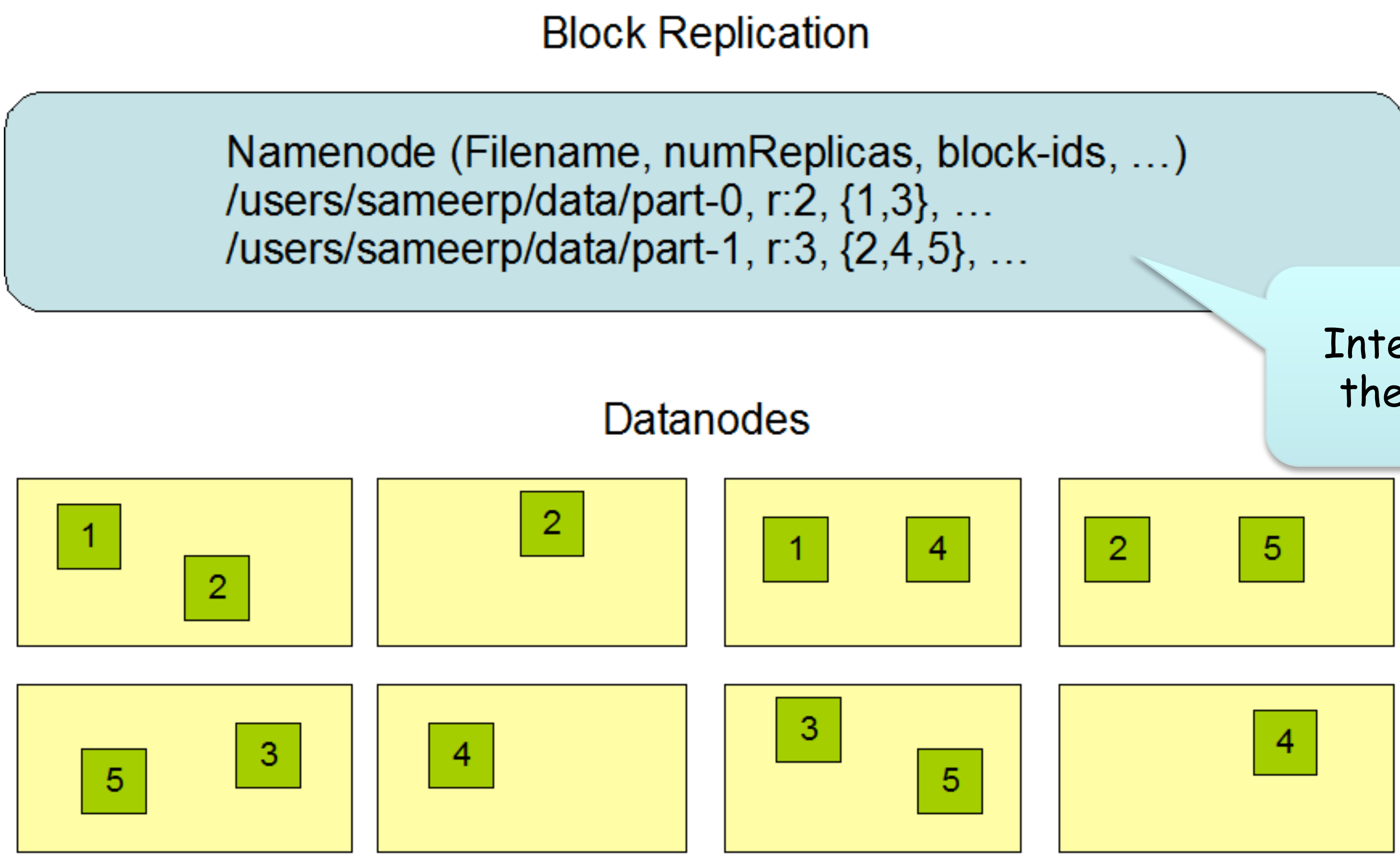
1. Client contacts NameNode about the file to open

2. NameNode returns locations of the file's blocks (e.g., DataNodes)

3. Clients contact directly the DataNodes in parallel

Notice that the client doesn't need to go to the NameNode to read blocks, only to find where they are

Replication in HDFS



Internal state in the NameNode

HDFS – Master/Slave Architecture

- A master NameNode
  - Stores metadata (file names, permission and block locations per file)
  - Metadata stored in memory for fast access
  - Controls file system operations
  - Maps data blocks to DataNodes
  - Logs all changes: always synchronized metadata
- Slave DataNodes
  - Store file blocks
  - Store replicated data

HDFS - Resilience

- Data replicated across the Data Nodes
- Nodes may fail but data still available
- DataNodes indicate their state via heartbeats
- Single point of failure in master NameNode
  - Redundant NameNodes are maintained

Hadoop YARN
(Yet Another Resource Negotiator)

YARN- What is it?

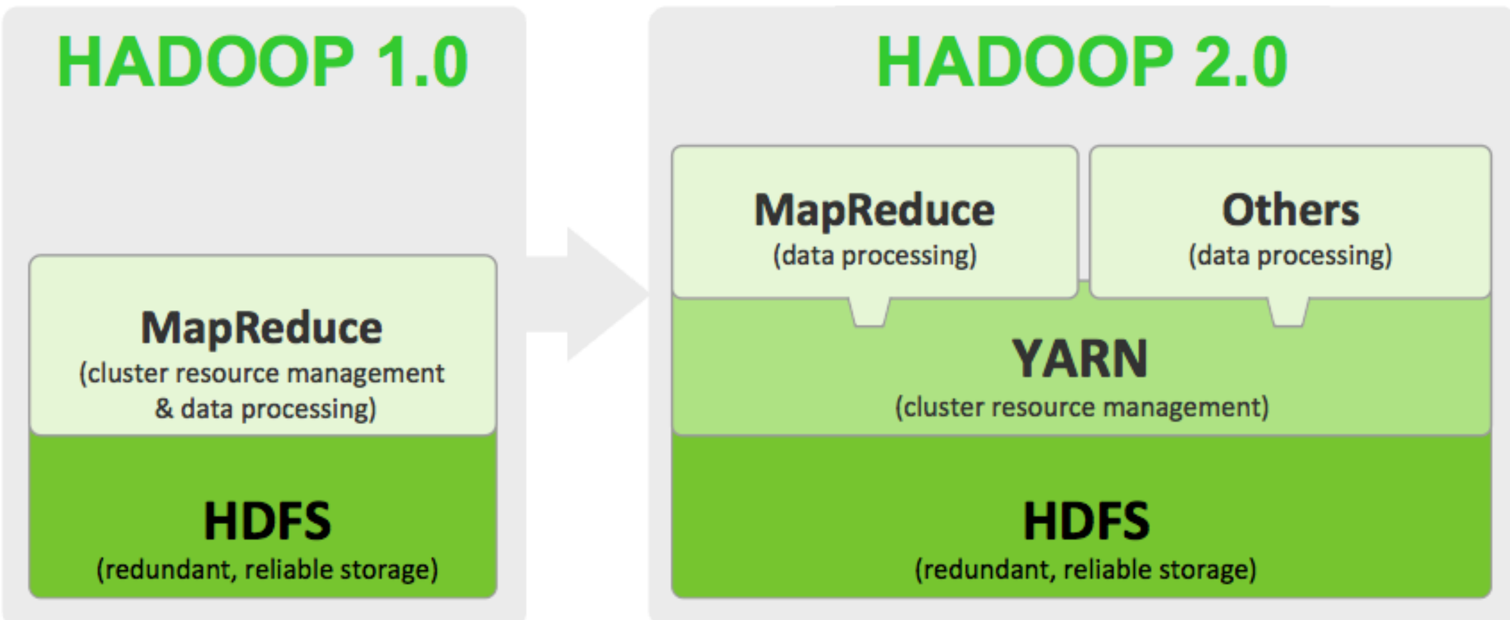
- Next generation MapReduce MRv2
- Splits Job Tracker to
  - Resource Manager
  - Scheduling/Monitoring
- Improves scaling
- Improves resource management
- Already used by Yahoo

Problems with Hadoop 1.0

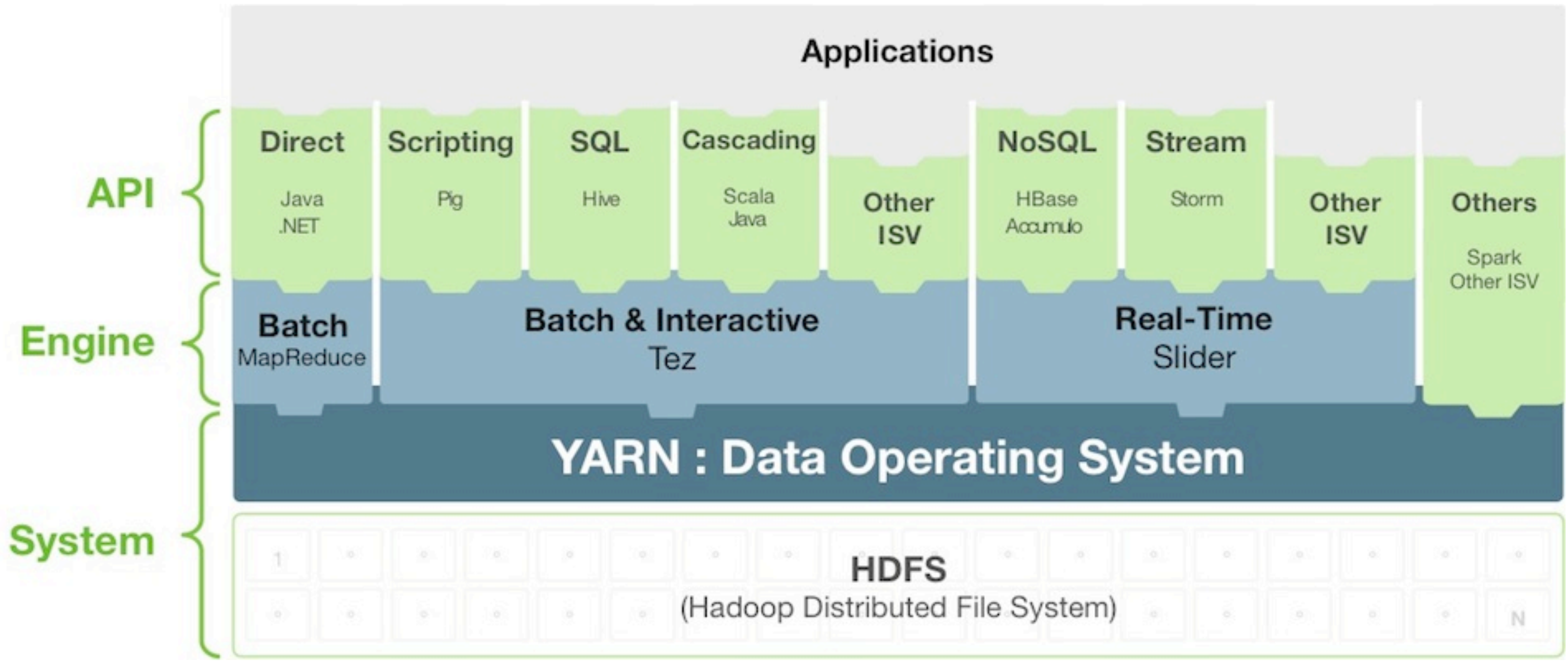
- Problems with large scaling
  - >4,000 nodes
  - >40K concurrent tasks
- Problems with resource utilization
- Slots only for Map or Reduce
- Single NameNode, single point of failure
- Clients and Cluster must be at the same version

What does Yarn do?

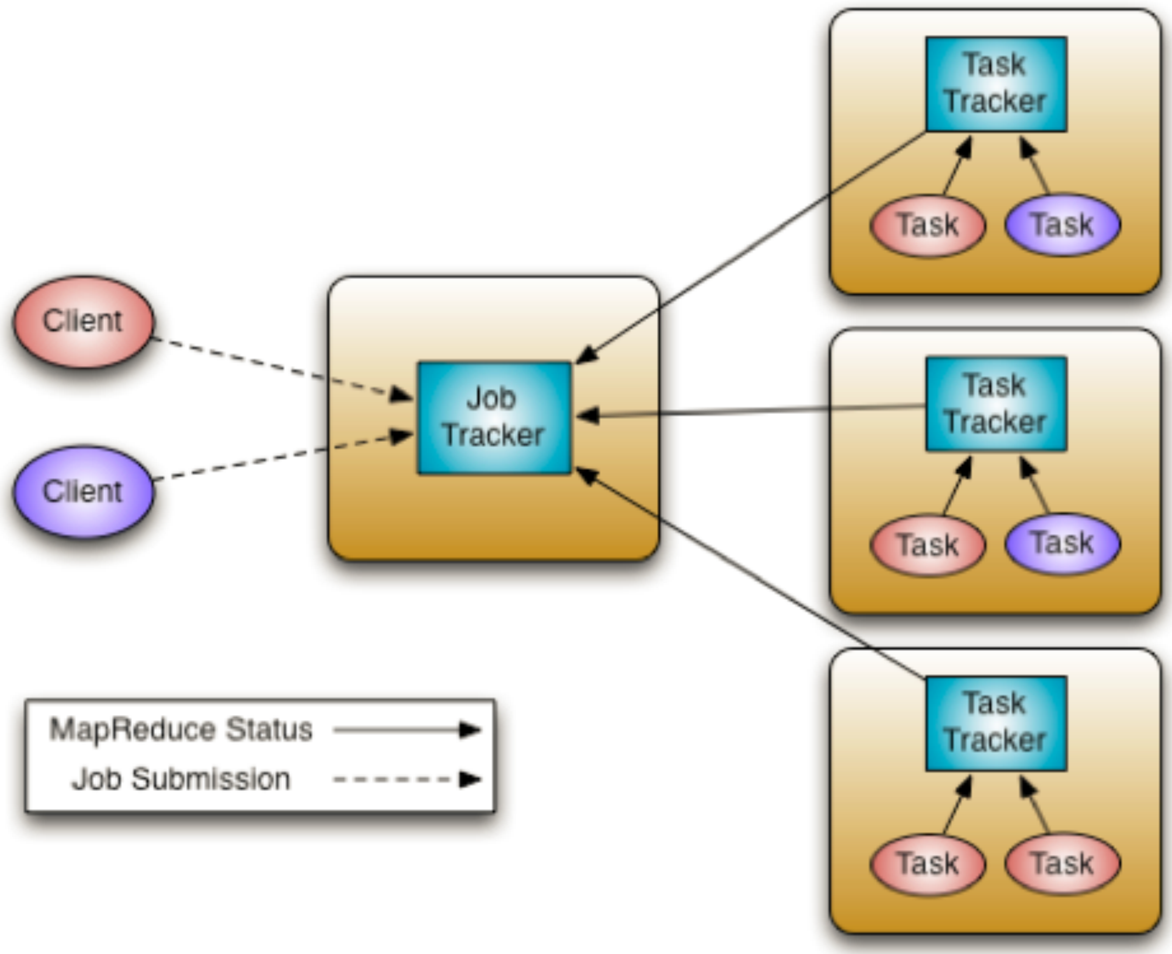
- Provides a cluster level resource manager
- Adds application level resource manager
- Provides slots for jobs other than Map/Reduce
- Improves resource utilization



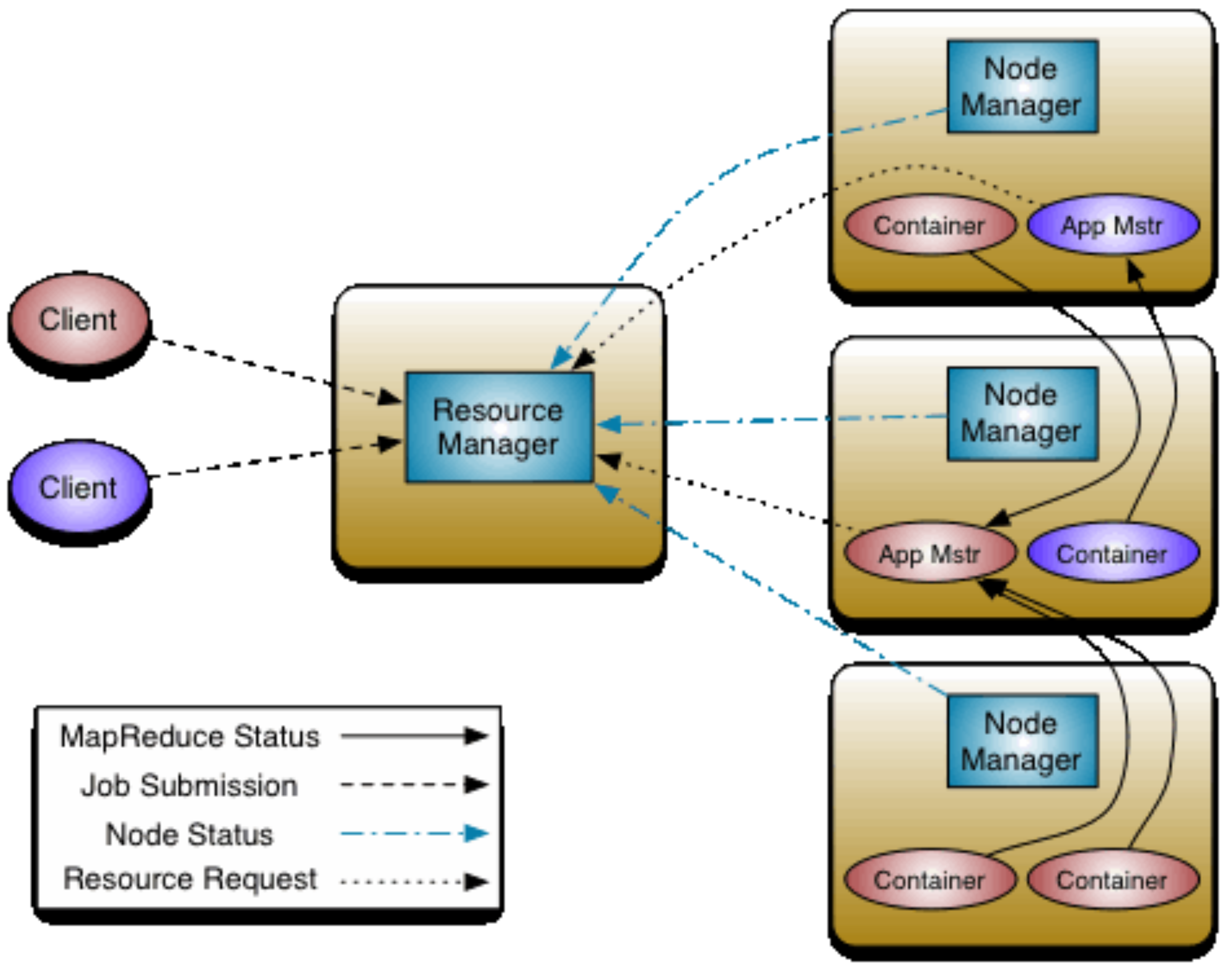
Other Applications on HDFS



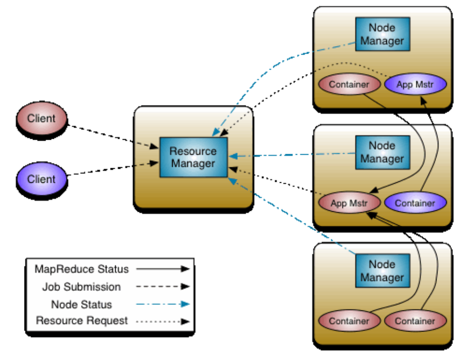
Old Architecture



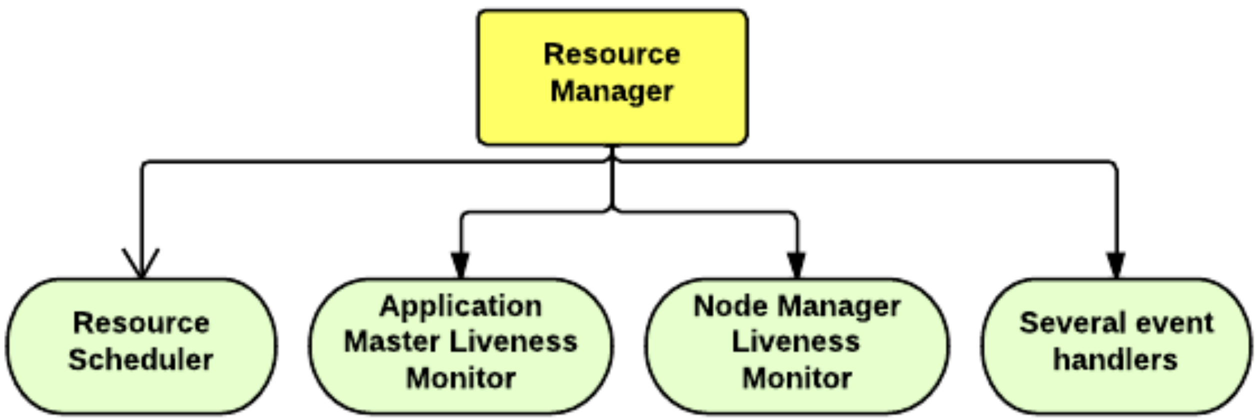
New Architecture



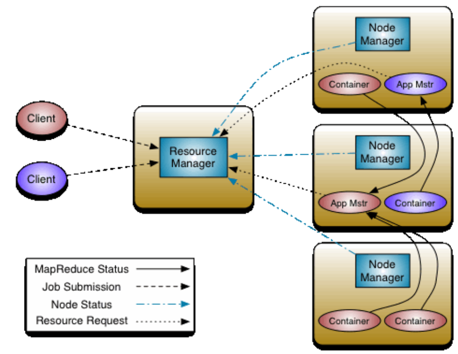
New Architecture



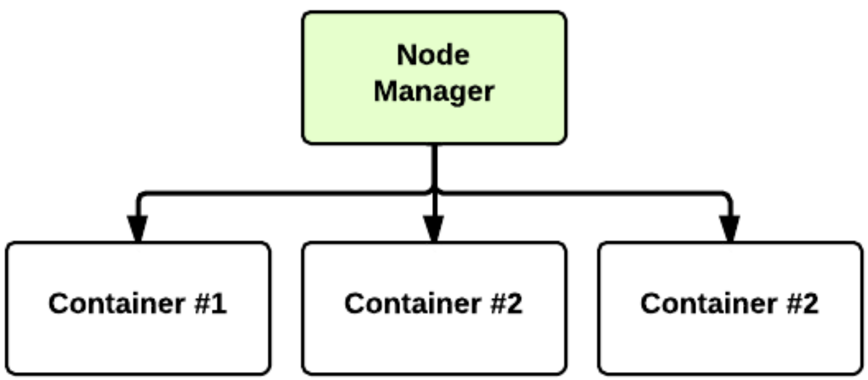
- Resource Manager: the master
  - One per cluster
  - Knows location of slaves (nodes) & their resources
  - Decides how to assign resources



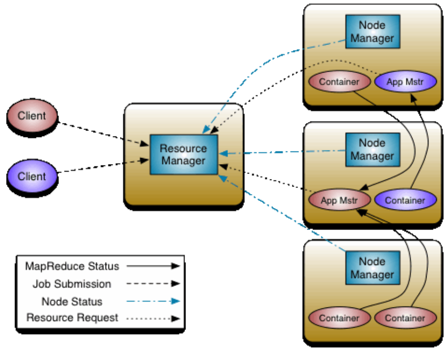
New Architecture



- Node Manager: the slave
  - One per data server (DataNode)
  - Offers resources to the cluster
  - Monitors resources on node (cpu, memory, disk, network)
  - Sends heartbeat and resource reports to the resource manager
  - **Container**: fraction on NM capacity used to run a program



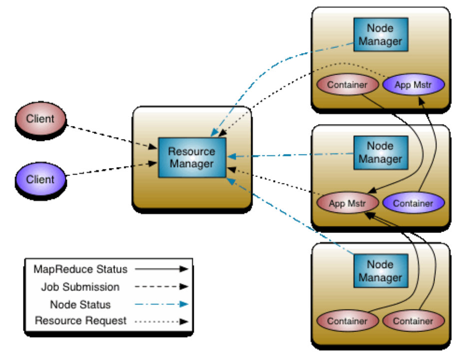
New Architecture

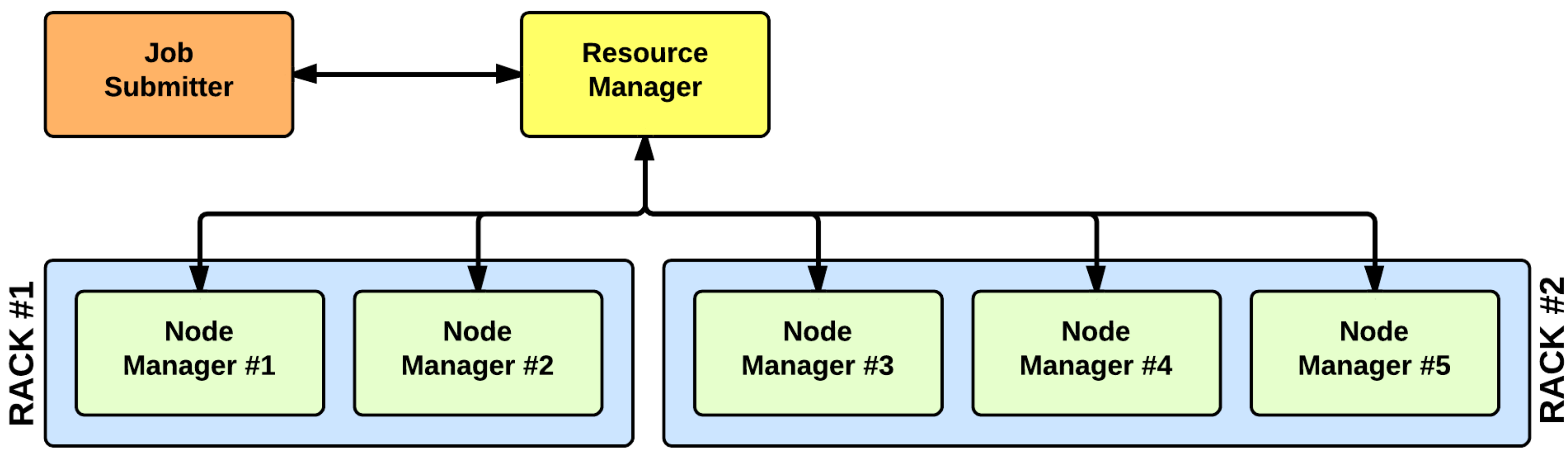


- Application Master
  - One per application
  - Responsible for executing an application
  - Asks for containers from the resource manager
  - Executes specific programs on the obtained containers
  - Monitors the status of the app's containers
  - MapReduce provides its own implementation of an Application Master

YARN Application Setup

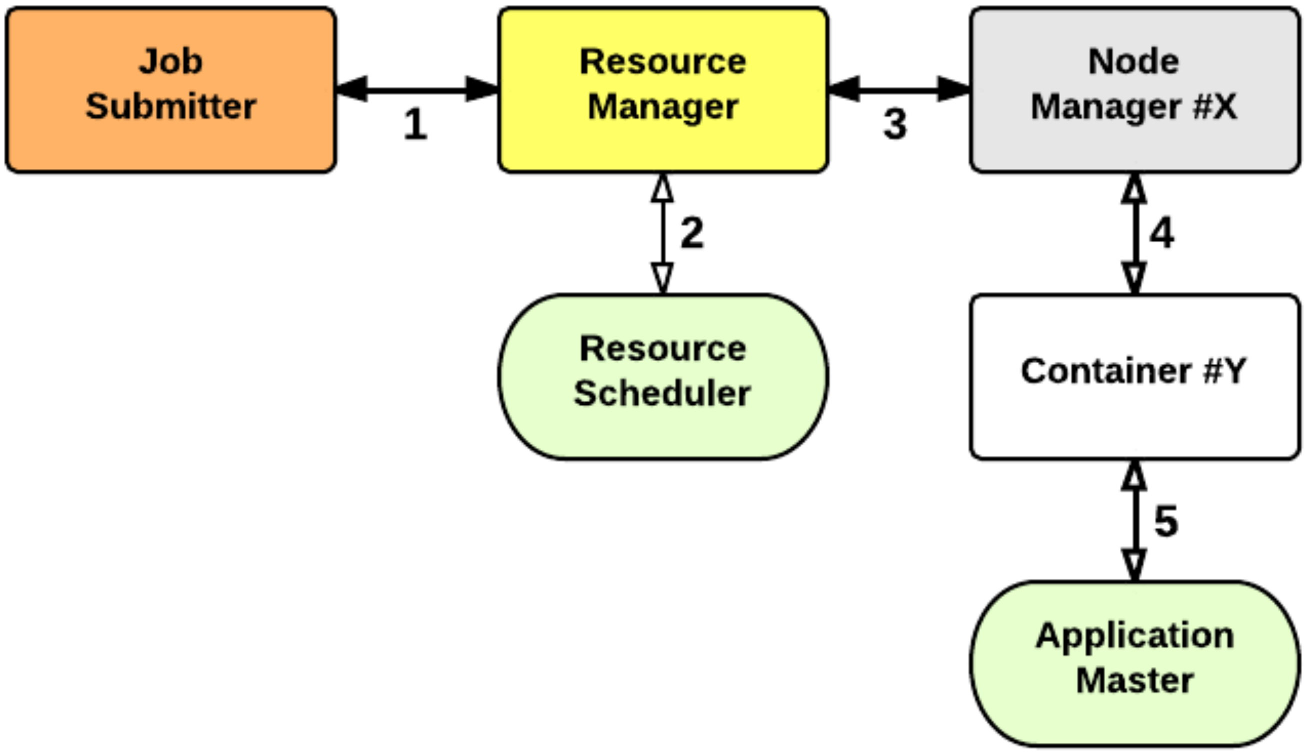
- Three actors:
  - Job Submitter (the client)
  - Resource Manager (the master)
  - Node Manager (the slave)





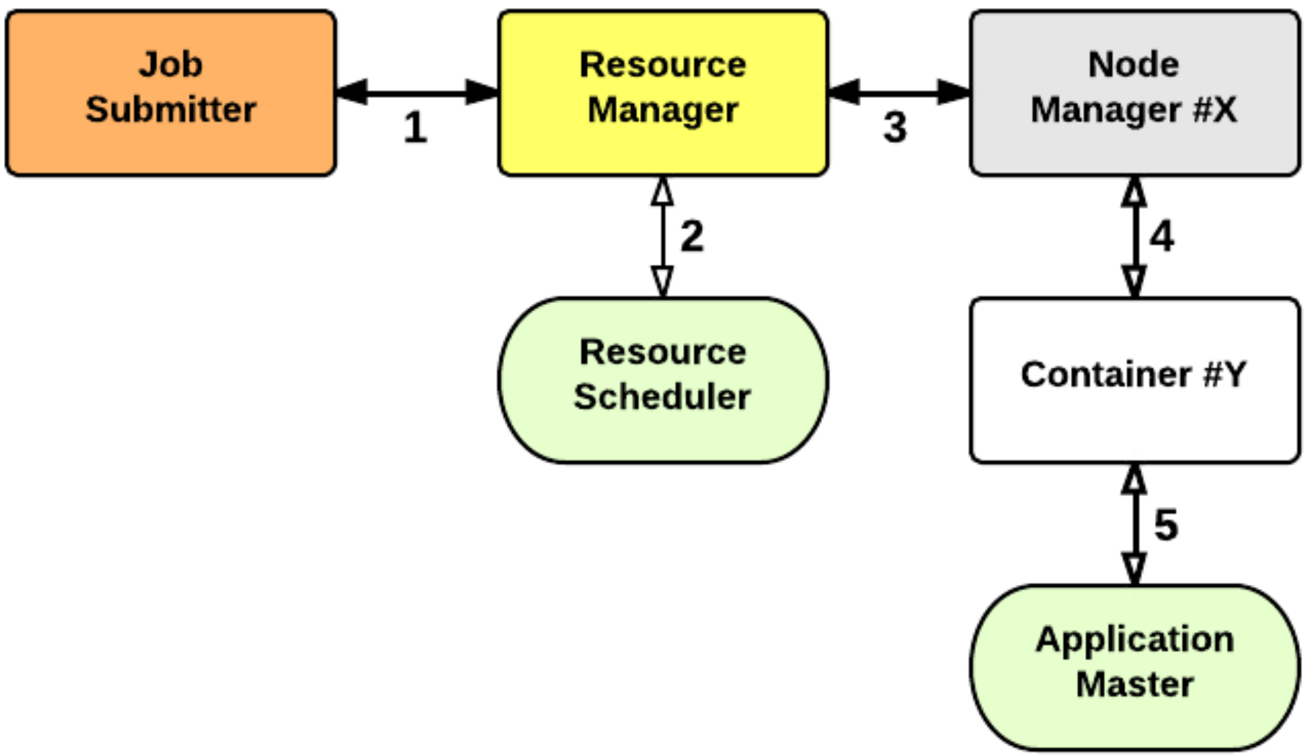
Startup Process

1. Client submits an application to the Resource Manager



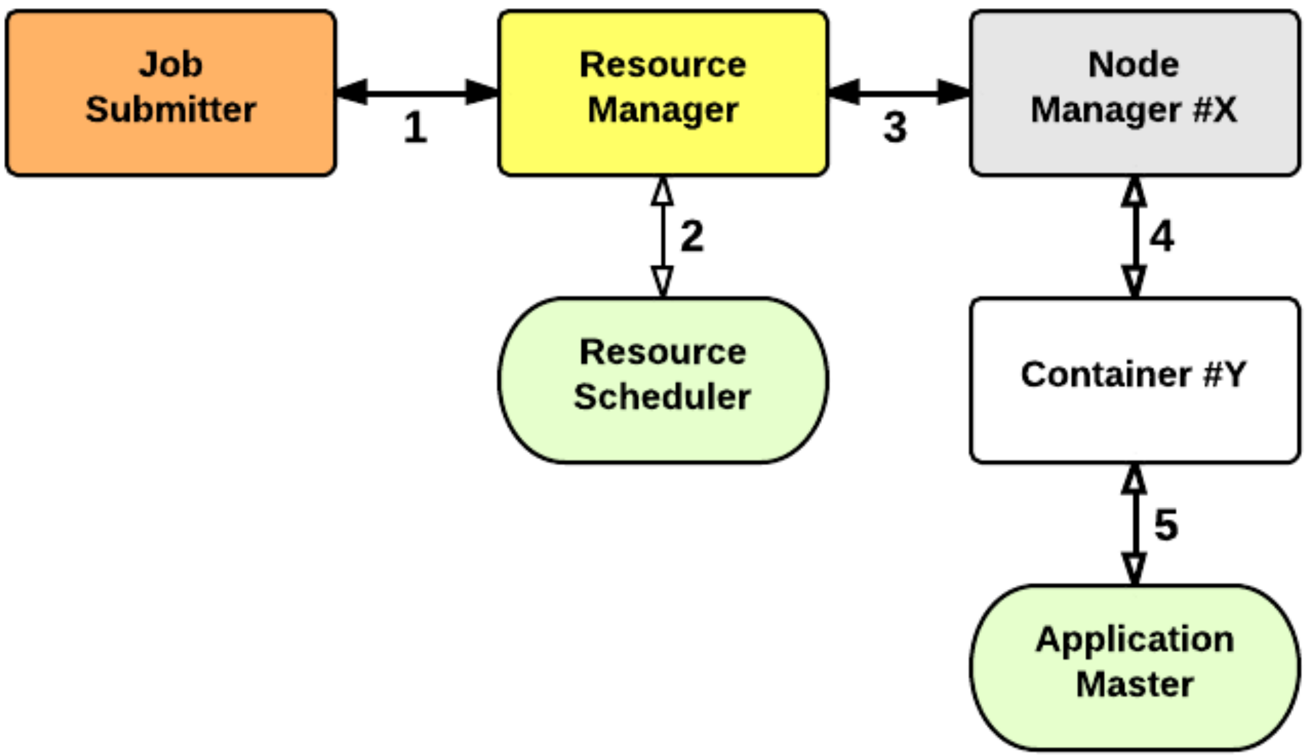
Startup Process

2. Resource Manager allocates a Container



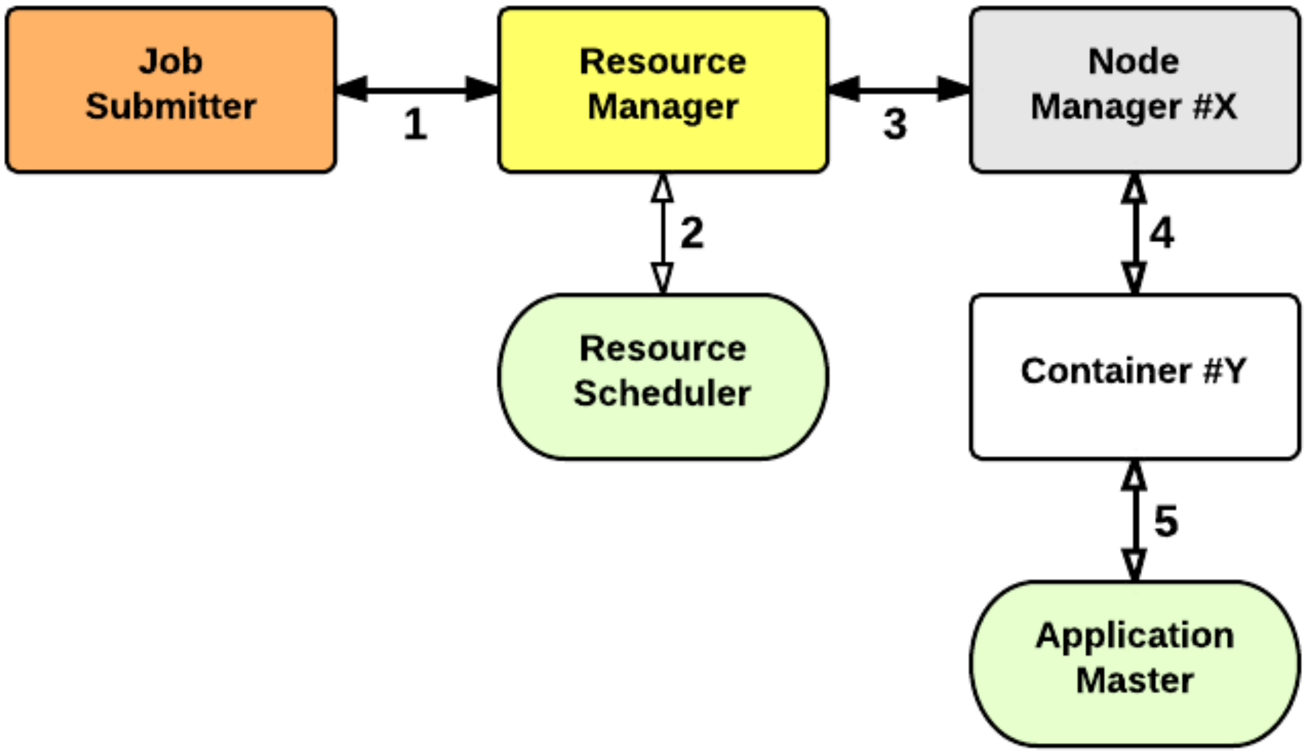
Startup Process

3. Resource Manager contacts the related Node Manager



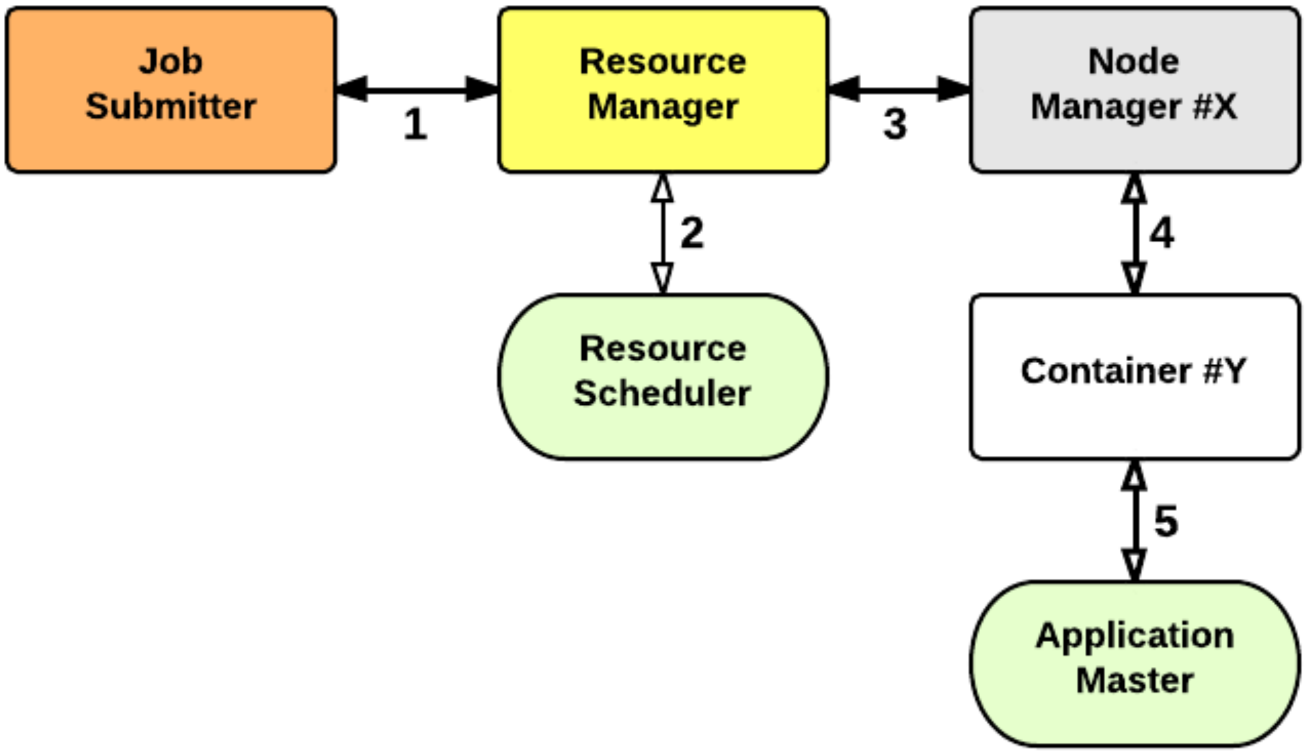
Startup Process

4. Node Manager launches the Container



Startup Process

5. The Container executes the application (in our case M/R job)



Resource Schedulers

- Capacity Scheduler (used in our cluster)
  - Relies on queues with certain capacity guarantees
    - Set up by administrators
  - Apps submit jobs to a queue and have access to the resources of that queue
  - Elastic resource allocation: free resources can be assigned to any queue running above its capacity but will also be given back if needed
- Fair Scheduler
  - All apps get an equal share of resources over times
  - Fairness decisions based on memory
  - Singe app gets all resources
  - Multiples apps share equally the resources