Big Data Technologies

MapReduce - YARN

Lionel Fillatre

Polytech Nice Sophia

lionel.fillatre@univ-cotedazur.fr

Outlines

- Basic definitions
- Some Handy Tools
- YARN
- YARN and MapReduce
- First MapReduce Job
- Hadoop 3
- Conclusion

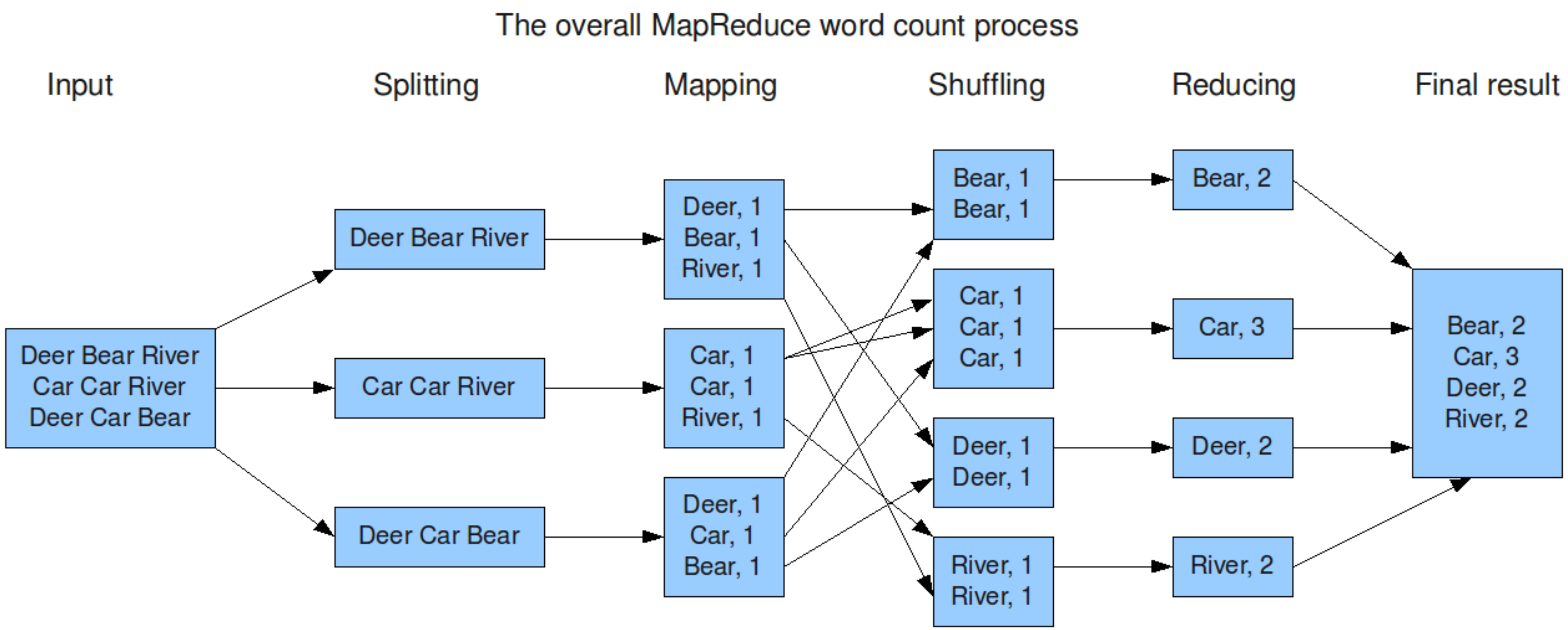
- Appendix: YARN Commands (for your information only)

Basic Definitions and Principles

Parallel and Distributed Programming Paradigms

- A distributed computing system consisting of a set or networked nodes or workers.
- The system issues for running a typical parallel program in either a parallel or a distributed manner would include the following:
  - **Computation partitioning** (program splitted in smaller tasks)
  - **Data partitioning** (data splitted in smaller pieces)
  - **Mapping** (small task mapped to a data piece)
  - **Synchronization** (synchronize the workers w.r.t. tasks)
  - **Communication** (communication between workers)
  - **Scheduling** (sequence of tasks/pieces of data assigned to a worker)

WordCount example



The couple MapReduce and YARN

- MapReduce is a programming model and an associated implementation for processing and generating large data sets.

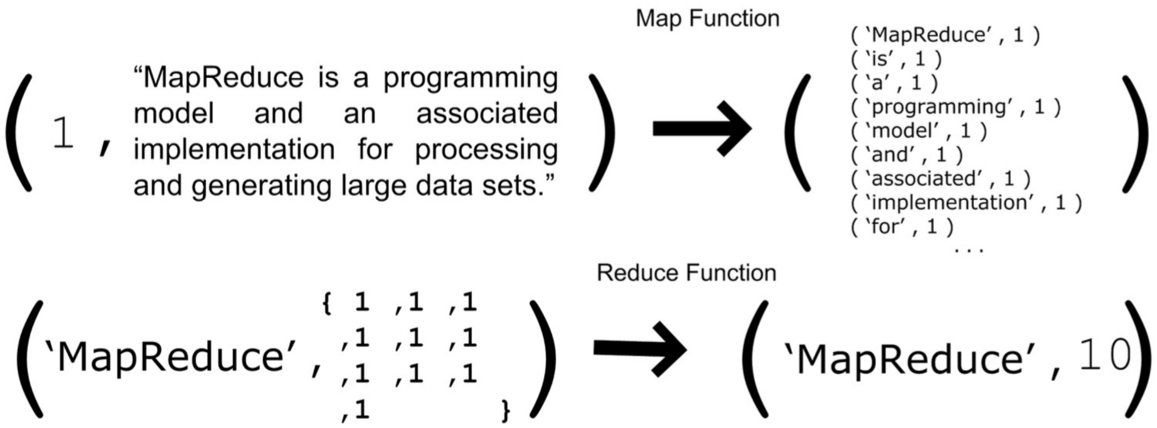
- Users specify a map function that processes a key/value pair to generate a set of intermediate key/value pairs, and a reduce function that merges all intermediate values associated with the same intermediate key.

- Programs written in this functional style are automatically parallelized and executed on a large cluster of machines.

- The run-time system, YARN in case of Hadoop, takes care of the details of partitioning the input data, scheduling the program execution across a set of machines, handling machine failures, and managing the required inter-machine communication.

Main Advantages

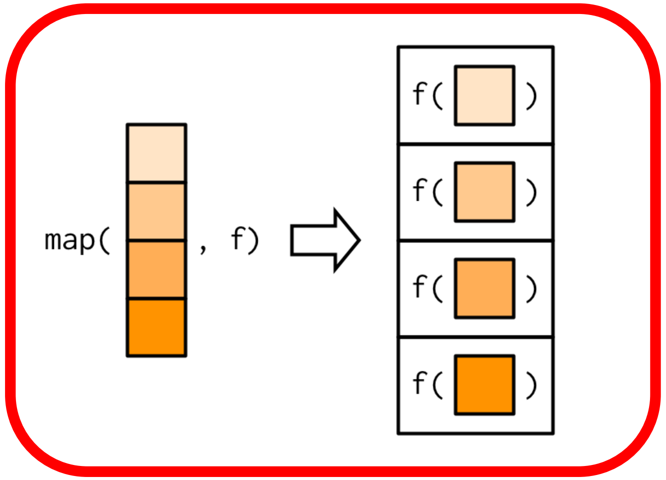
- Many real world tasks are expressible in this model.
- This model allows programmers without any experience with parallel and distributed system to easily utilize the resources of a large distributed system.
- It is inspired by the map and reduce functions commonly used in functional programming



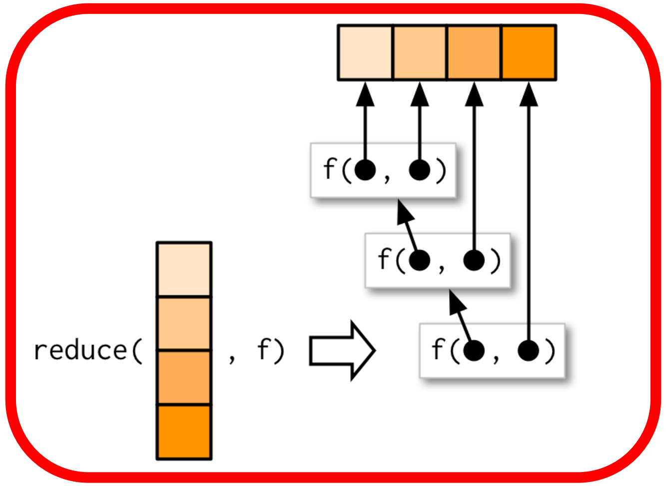
https://adv-r.hadley.nz/functionals.html

Main principle of MapReduce

- Derived from functional programming (no global variables, no side effects)
- Can be implemented in multiple languages (Java, Scala, C++, Python, etc.)
- **Map**: (f, [a, b, c, ...]) -> [f(a), f(b), f(c), ...]
  - Apply a function to all the elements of a list
  - ex.: map((f: x->x + 1), [1, 2, 3]) = [2, 3, 4]
  - Intrinsically **parallel**



- **Reduce**: (f, [a, b, c, ...]) -> f(a, f(b, f(c, ...)))
  - Apply a function to a list recursively
  - ex.: reduce((f: (x,y)->(x + y)), [1, 2, 3, 4]) = (1+(2+(3+4)))=10



MapReduce Programming Paradigm

- **Input of MapReduce:** the basic unit of information is a <key; value> pair.

- A program in the MapReduce paradigm can consist of many rounds of different map and reduce functions, performed one after another.

1- Split stage:

- The MapReduce framework transforms the files in a set of <key; value> pairs
- Provides a mapper with all input data associated with a given key

split: data → list (K1,V1)

Example: word count

(deer, bear, river, car, car, river, deer, car, bear)	→	(line1, (deer, bear, river)) (line2, (car, car, river)) (line3, (deer, car, bear))

MapReduce Programming Paradigm

2- Map stage:

- The mapper takes as input a single <key; value> pair, and produces as output any number of new <key; value> pairs.
- The map operation is stateless: it operates on one pair at a time. This allows for easy parallelization: different inputs for the map can be processed by different machines

map: (K1,V1) → list (K2,V2)

Example: word count

(line1, (deer, bear, river))
(line2, (car, car, river))
(line3, (deer, car, bear))

→

(deer, 1)
(bear, 1)
(river, 1)
(car, 1)
(car, 1)
(river, 1)
(deer, 1)
(car, 1)
(bear, 1)

MapReduce Programming Paradigm

3- Shuffle stage:

- All of the values that are associated with the same key are sent to the same machine.
- This is the possible bottleneck of the process
- Intermediate key-values are sorted and grouped by key
- This occurs automatically and is seamless to the programmer

shuffle: list (K2,V2) →(K2,list(V2))

Example: word count

(deer, 1) (bear, 1) (river, 1) (car, 1) (car, 1) (river, 1) (deer, 1) (car, 1) (bear, 1)	→	(bear, (1, 1)) (car, (1, 1, 1)) (deer, (1, 1)) (river, (1, 1))

MapReduce Programming Paradigm

4- Reduce stage:

- The reducer takes all of the values associated with a single key *k*, and outputs a multiset of <key; value> pairs with the same key *k*.
- Since the reducer has access to all the values with the same key, it can perform sequential computations on these values.
- The parallelism is exploited by observing that reducers operating on different keys can be executed simultaneously.
- Remark: all of the maps need to finish before the reduce stage can begin.

reduce: (K2,list(V2)) → list (K3,V3)

Example: word count

(bear, (1, 1))		(bear, 2)
(car, (1, 1, 1))	→	(car, 3)
(deer, (1, 1))		(deer, 2)
(river, (1, 1))		(river, 2)

MapReduce Programming Paradigm

5- Final output stage:

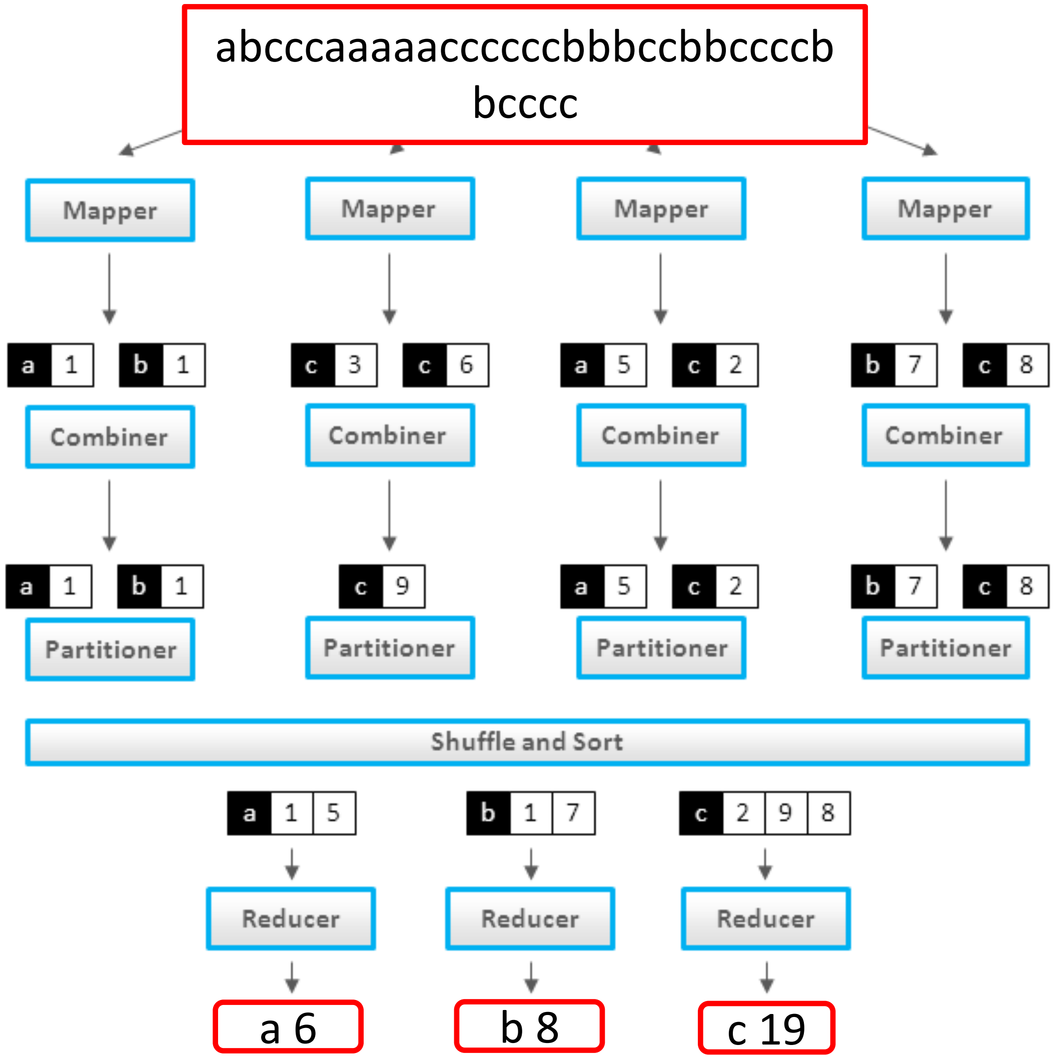
- The MapReduce framework collects all the Reducer outputs, and sorts them by key to produce the final outcome.

Example: word count

(bear, 2)
(car, 3)
(deer, 2)
(river, 2)

Some Handy Tools

Full MapReduce Job



Partitioners

- Partitioners are application code that define how keys are assigned to reducers

- Default partitioning spreads keys evenly, but randomly
  - Uses *key.hashCode() % num_reduces*

- Custom partitioning is often required, for example, to produce a total order in the output
  - To get a total order, sample the map output keys and pick values to divide the keys into roughly equal buckets and use that in your partitioner

Combiners

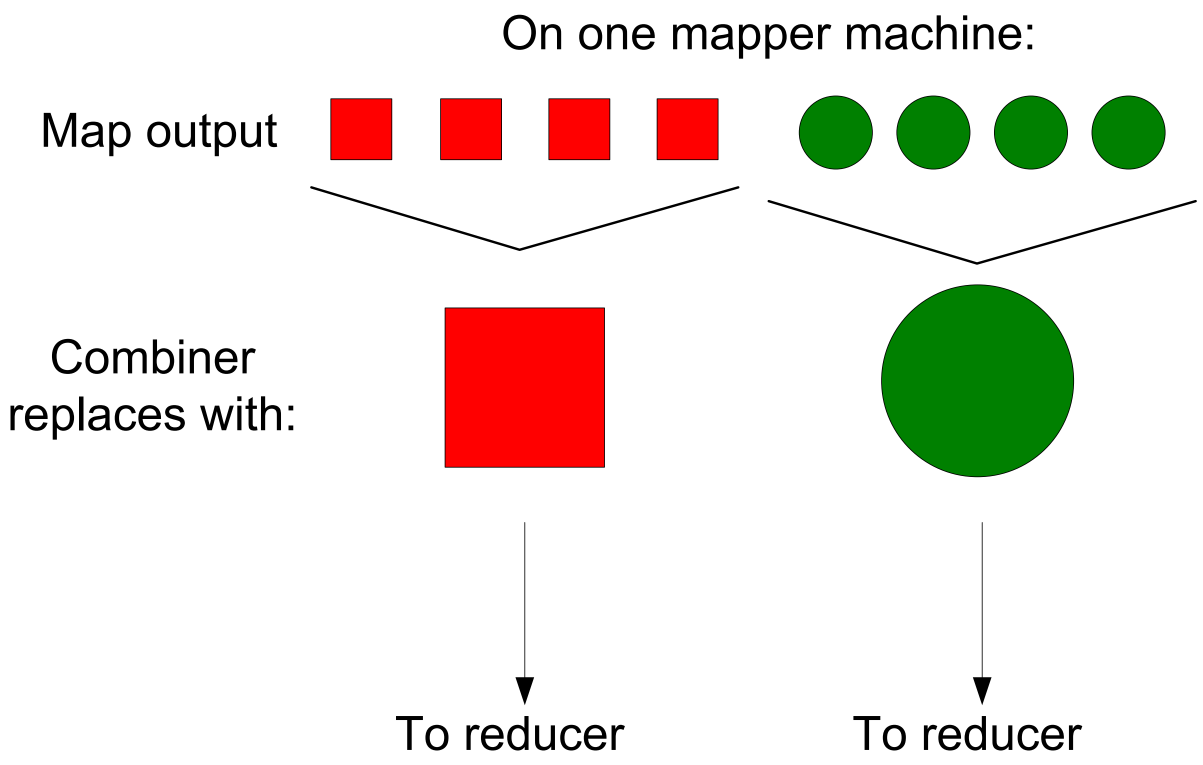
- “Mini-reducer,” only on local map output

- Run on map machines after map phase

- When maps produce many repeated keys
  - It is often useful to do a local aggregation following the map
  - Done by specifying a Combiner
  - Goal is to decrease size of the transient data
  - Used to save bandwidth before sending data to full reduce tasks

- Reduce tasks can be combiner if commutative and associative
  - Combiners have the same interface as Reduces, and often are the same class

Combiners, graphically



Compression

- Compressing the outputs and intermediate data will often yield huge performance gains
  - Can be specified via a configuration file or set programmatically
  - Set mapred.output.compress to true to compress job output
  - Set mapred.compress.map.output to true to compress map outputs

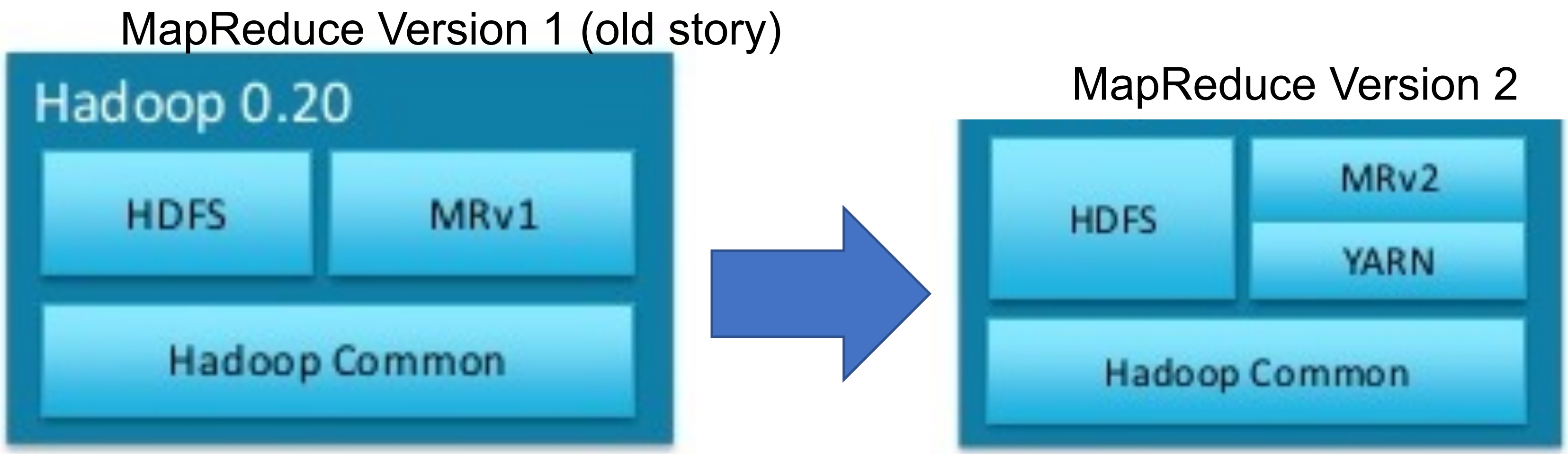
- Compression Types
  - “block” - Group of keys and values are compressed together
  - “record” - Each value is compressed individually

- Compression Codecs
  - Default (zlib) - slower, but more compression
  - LZO - faster, but less compression

YARN

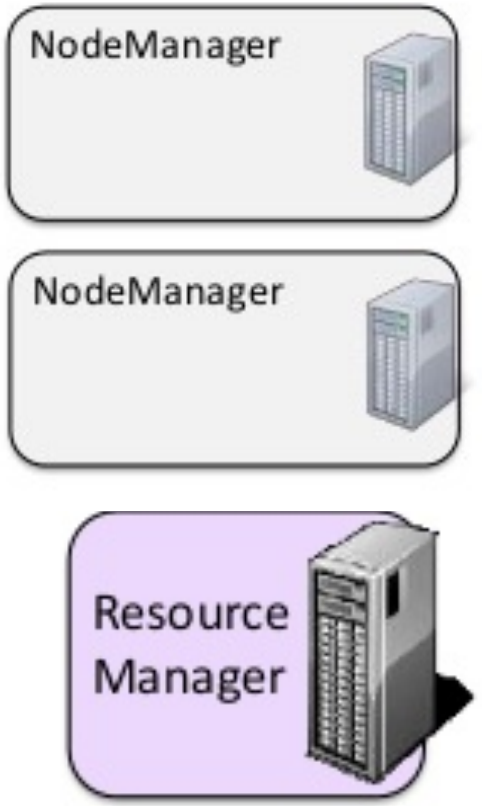
MapReduce 2.0 on YARN

- **Yet Another Resource Negotiator (YARN)**
- **Various applications can run on YARN**
  - MapReduce is just one choice (the main choice at this point)
  - http://wiki.apache.org/hadoop/PoweredByYarn



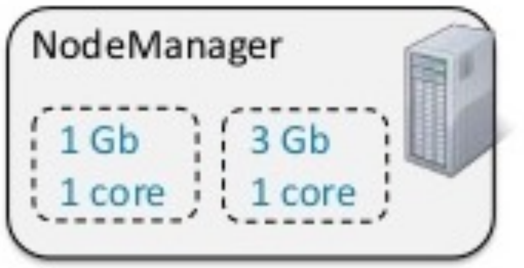
Daemons

- **YARN Daemons**
    - Node Manager
        - Manages resources of a single node
        - There is one instance per node in the cluster
        - Communicates with Resource Manager
        - Runs on slave mode
    - Resource Manager (RM)
        - Manages Resources for a Cluster
        - Instructs Node Manager to allocate resources
        - Application negotiates for resources with Resource Manager
        - There is only one instance of Resource Manager
- **MapReduce Specific Daemon**
    - MapReduce History Server
        - Archives Jobs' metrics and meta-data



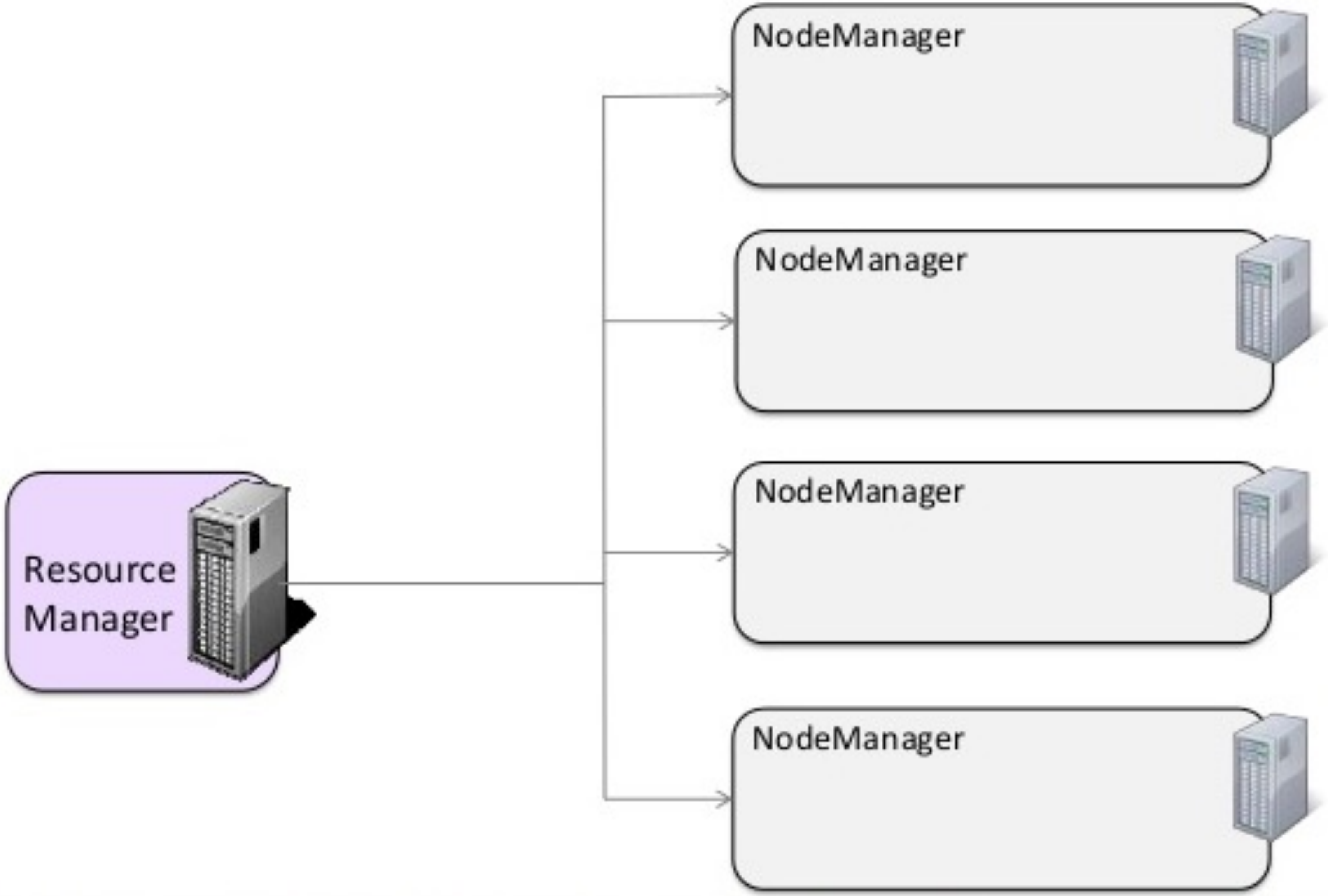
Daemons

- **Containers**
  - Created by the Ressource Manager upon request
  - Allocate a certain amount of resources (memory, CPU) on a slave node
  - Applications run in one or more container

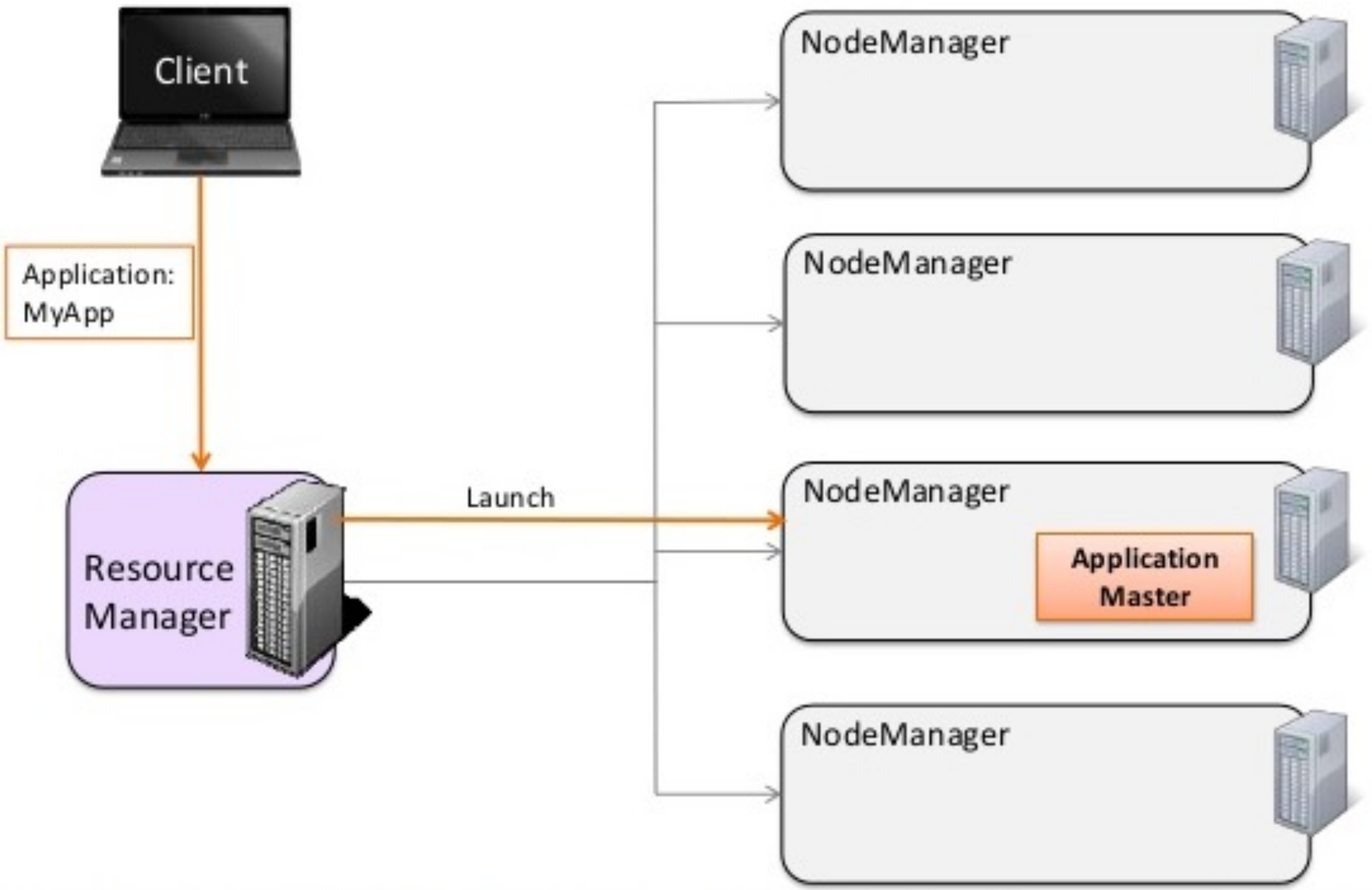


- **Application Master (AM)**
  - One per application
  - Framework/application specific
  - Runs in a container
  - Requests more containers to run application tasks

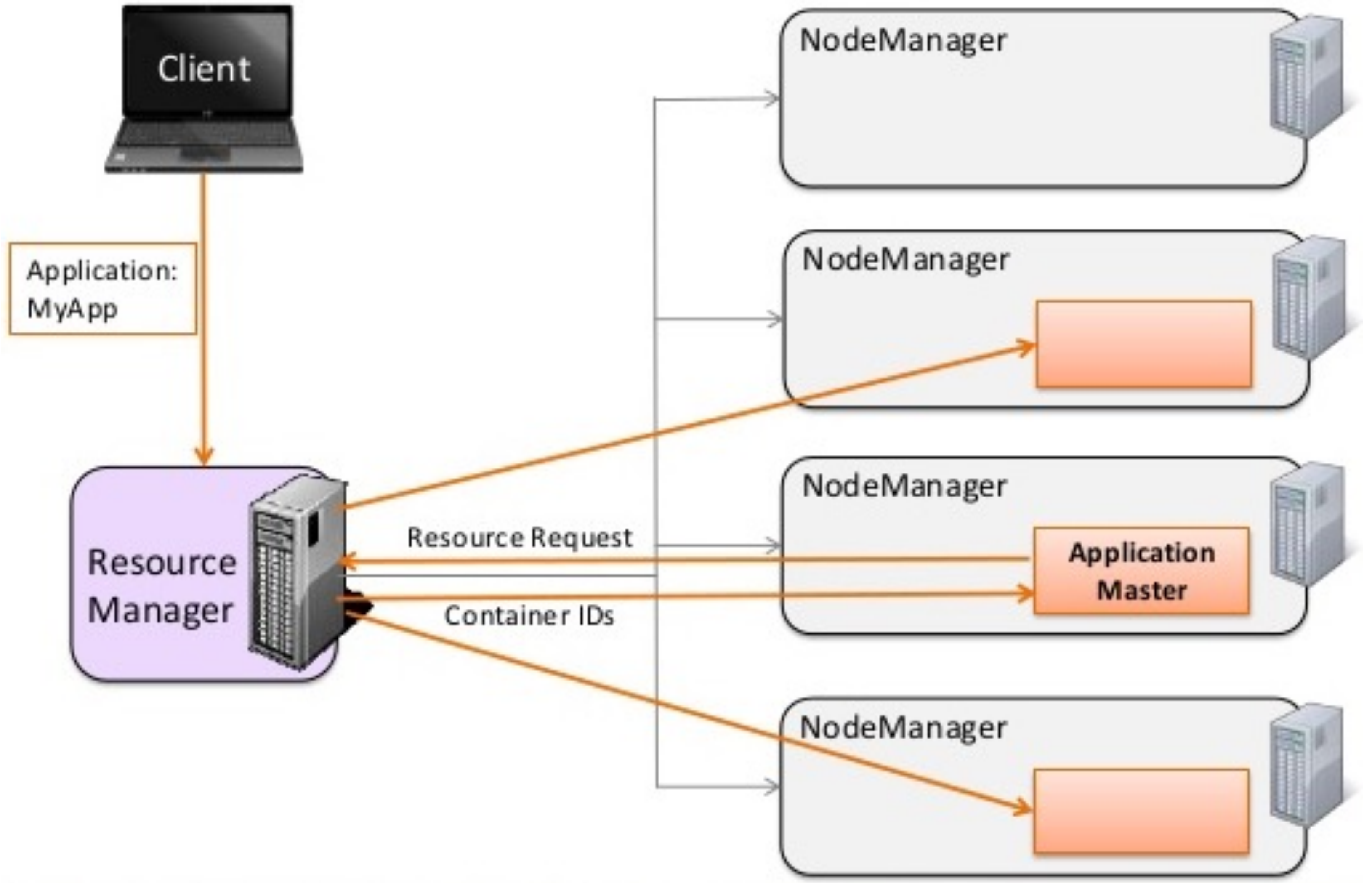
YARN Cluster



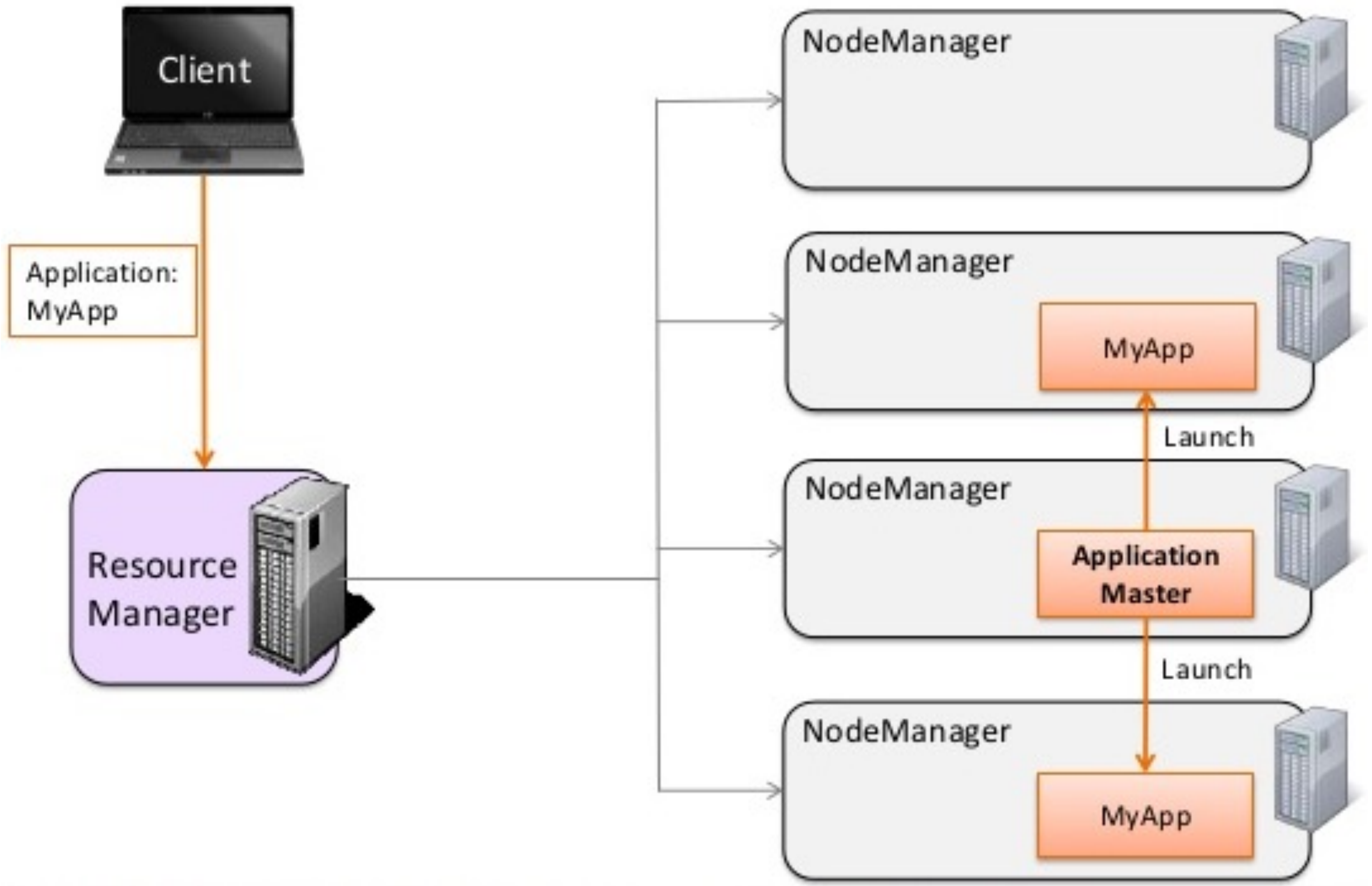
YARN: Running an Application



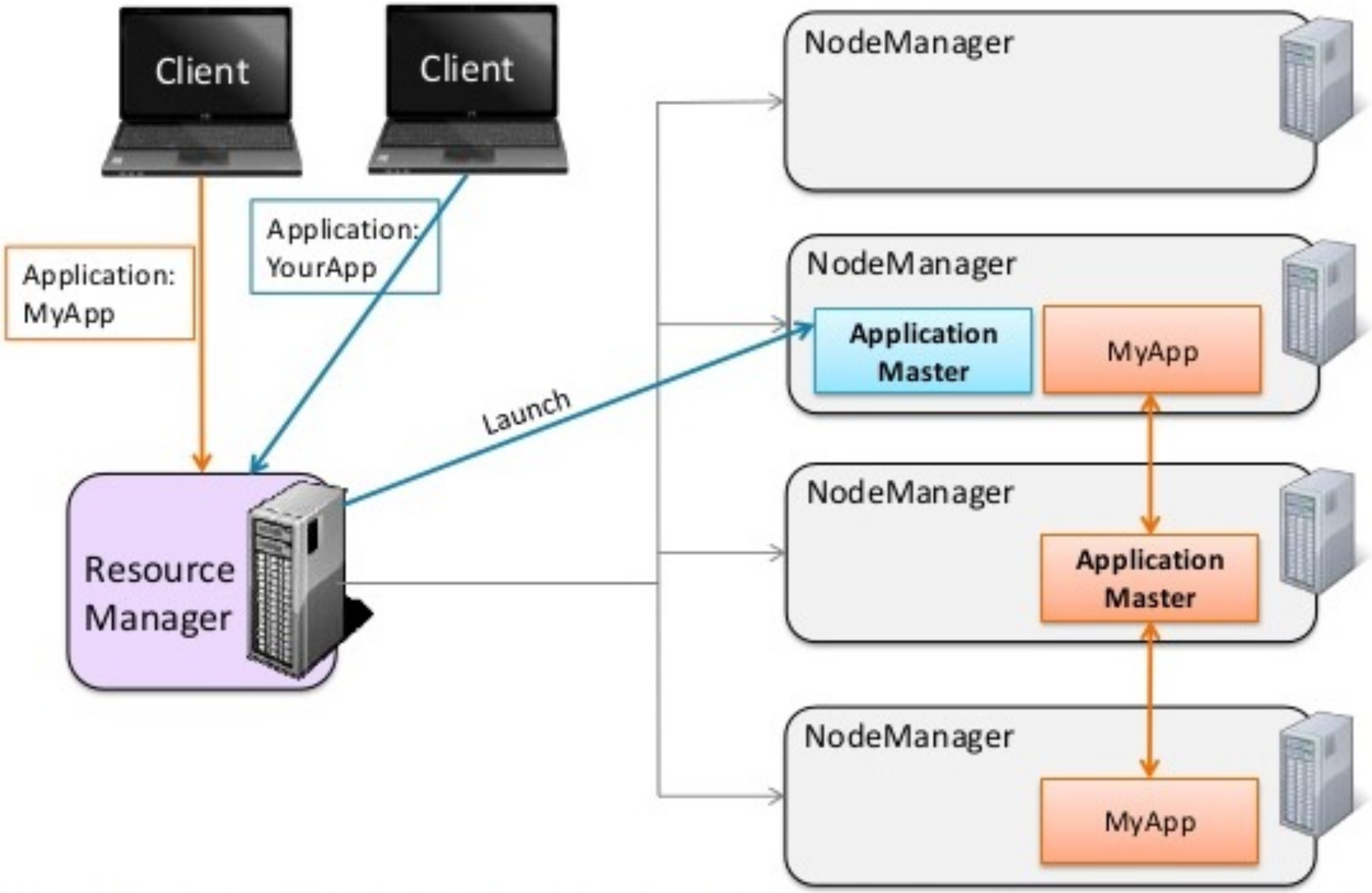
YARN: Running an Application



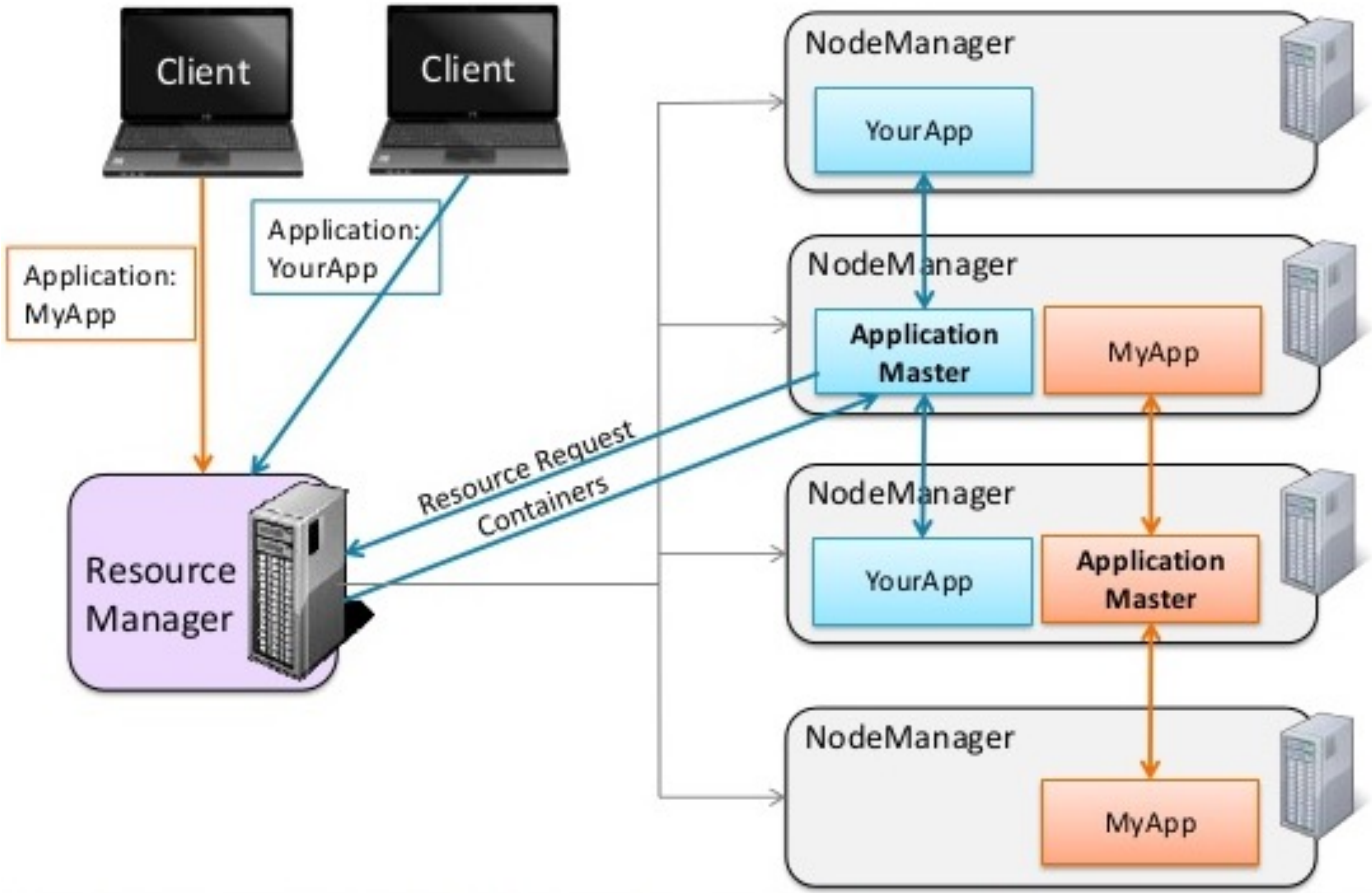
YARN: Running an Application



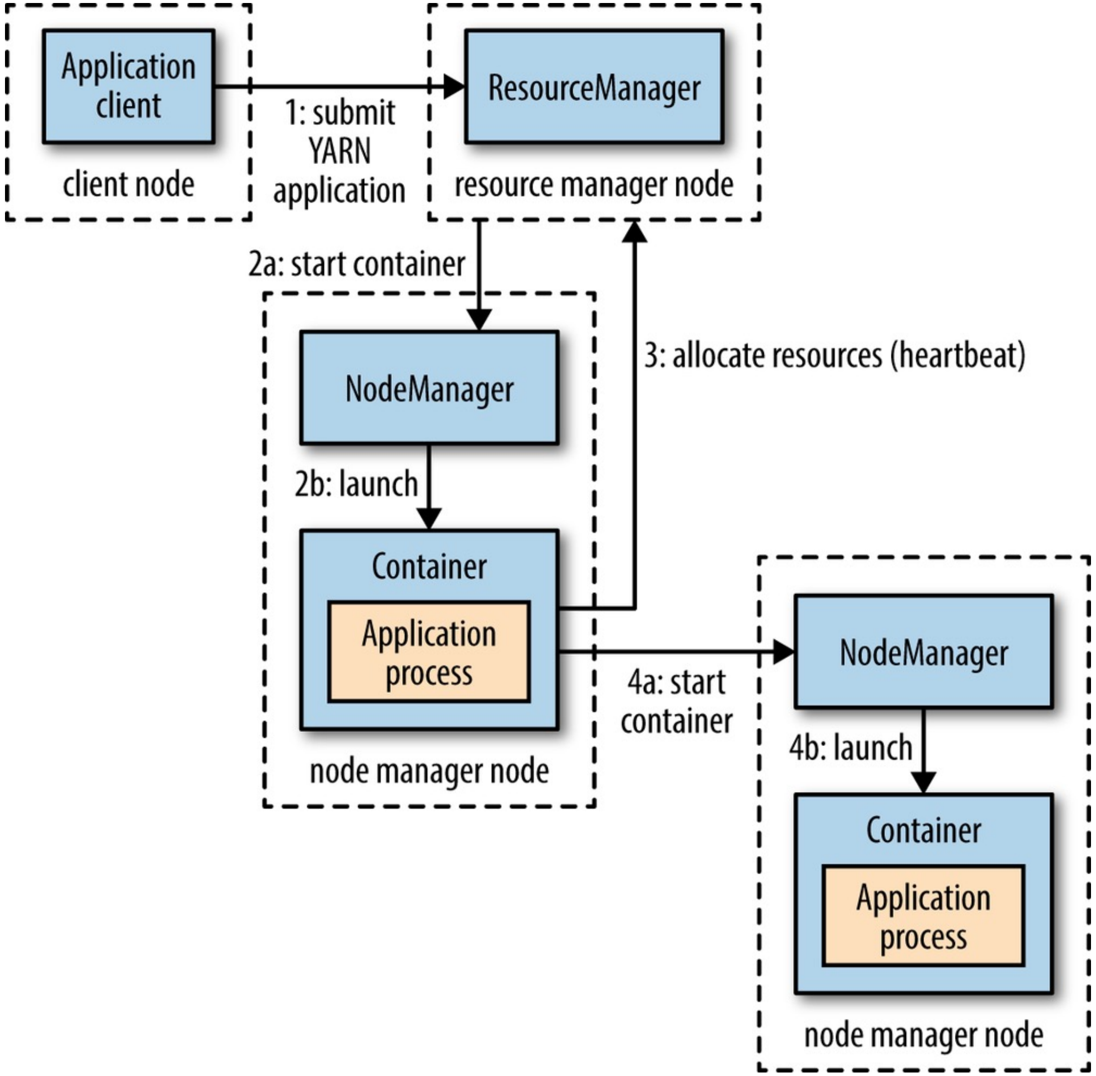
YARN: Running an Application



YARN: Running an Application



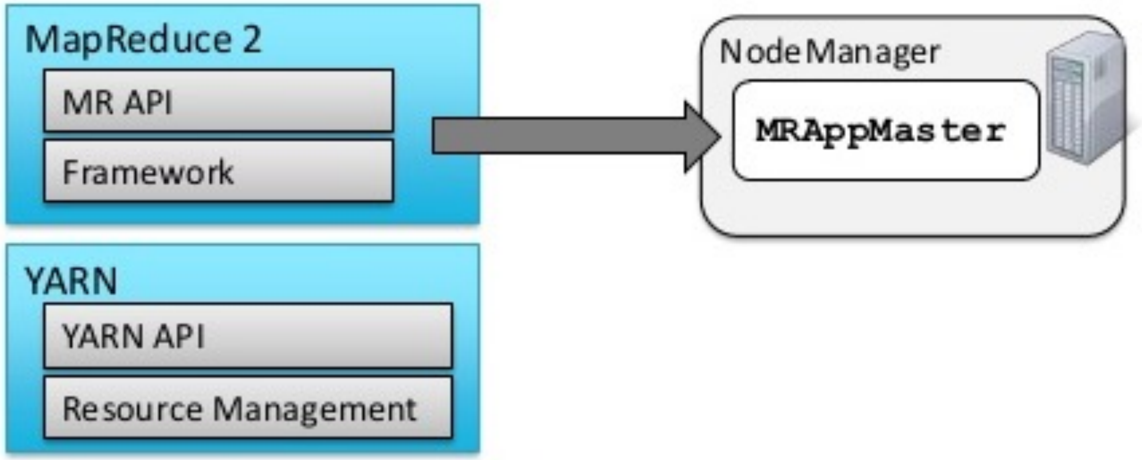
Anatomy of a YARN Application Run



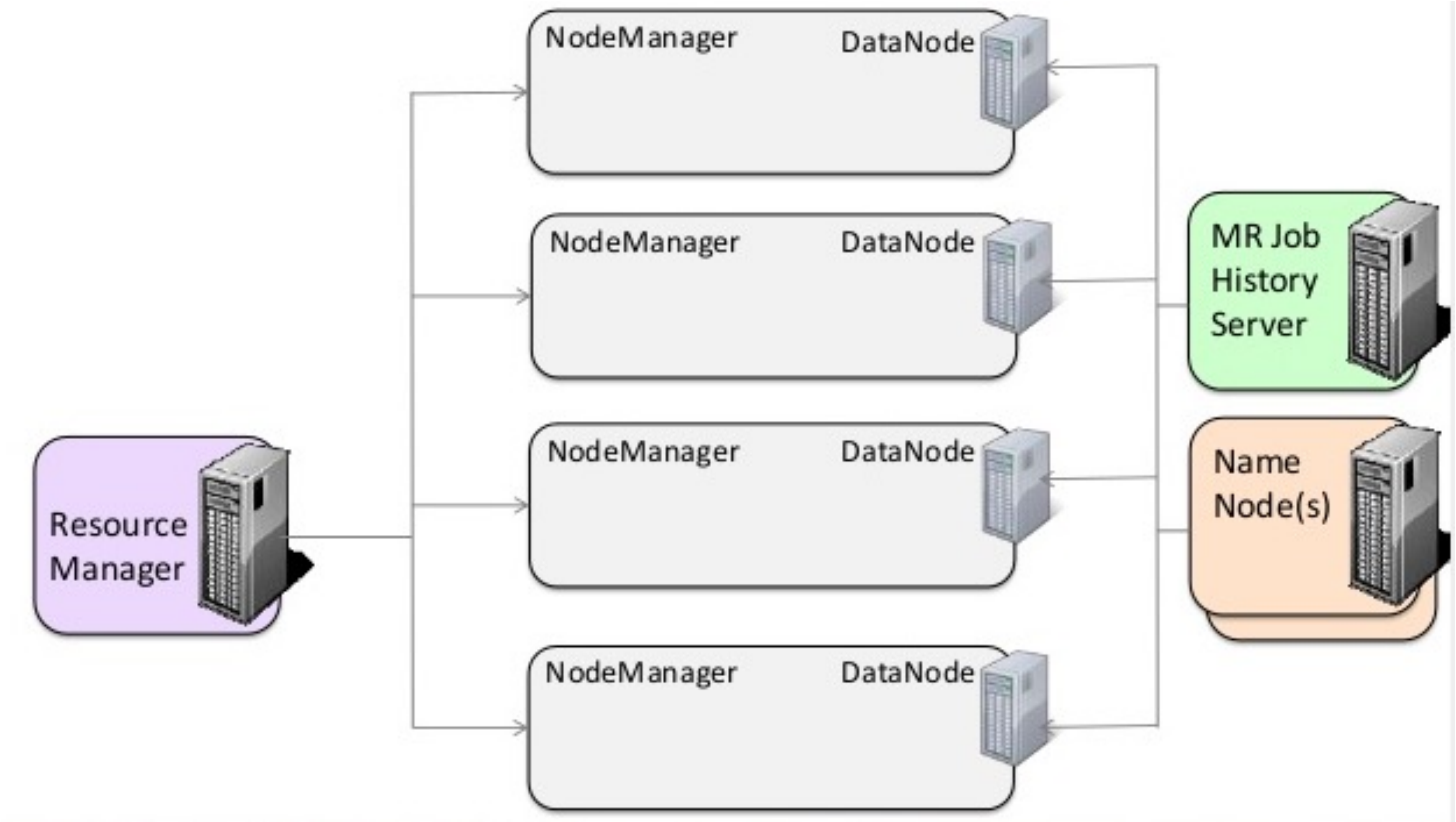
YARN and MapReduce

YARN and MapReduce

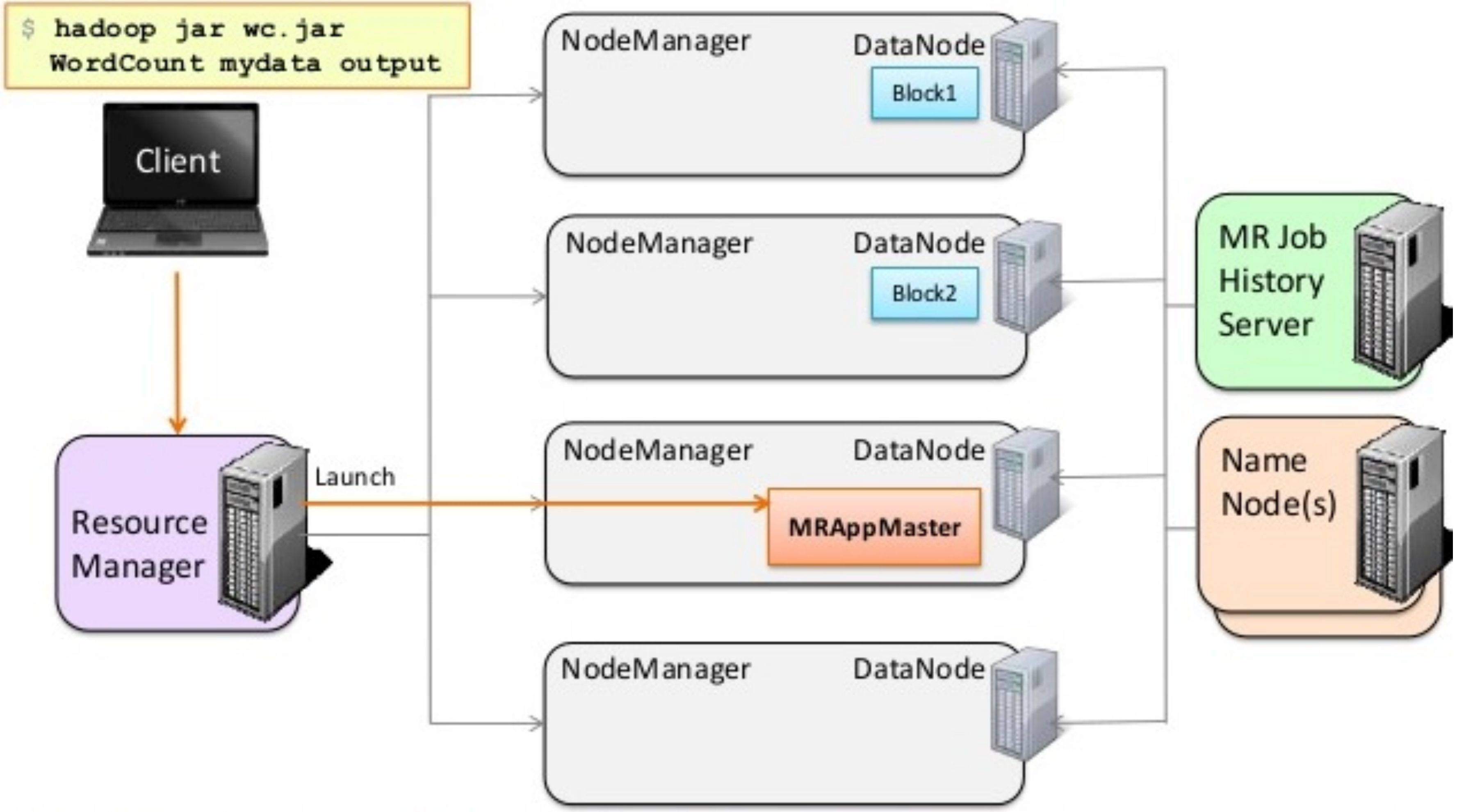
- YARN does not know or care what kind of application it is running
- MapReduce uses YARN
  - Hadoop includes a MapReduce ApplicationMaster to manage MapReduce jobs
  - Each MapReduce job is an instance of an application



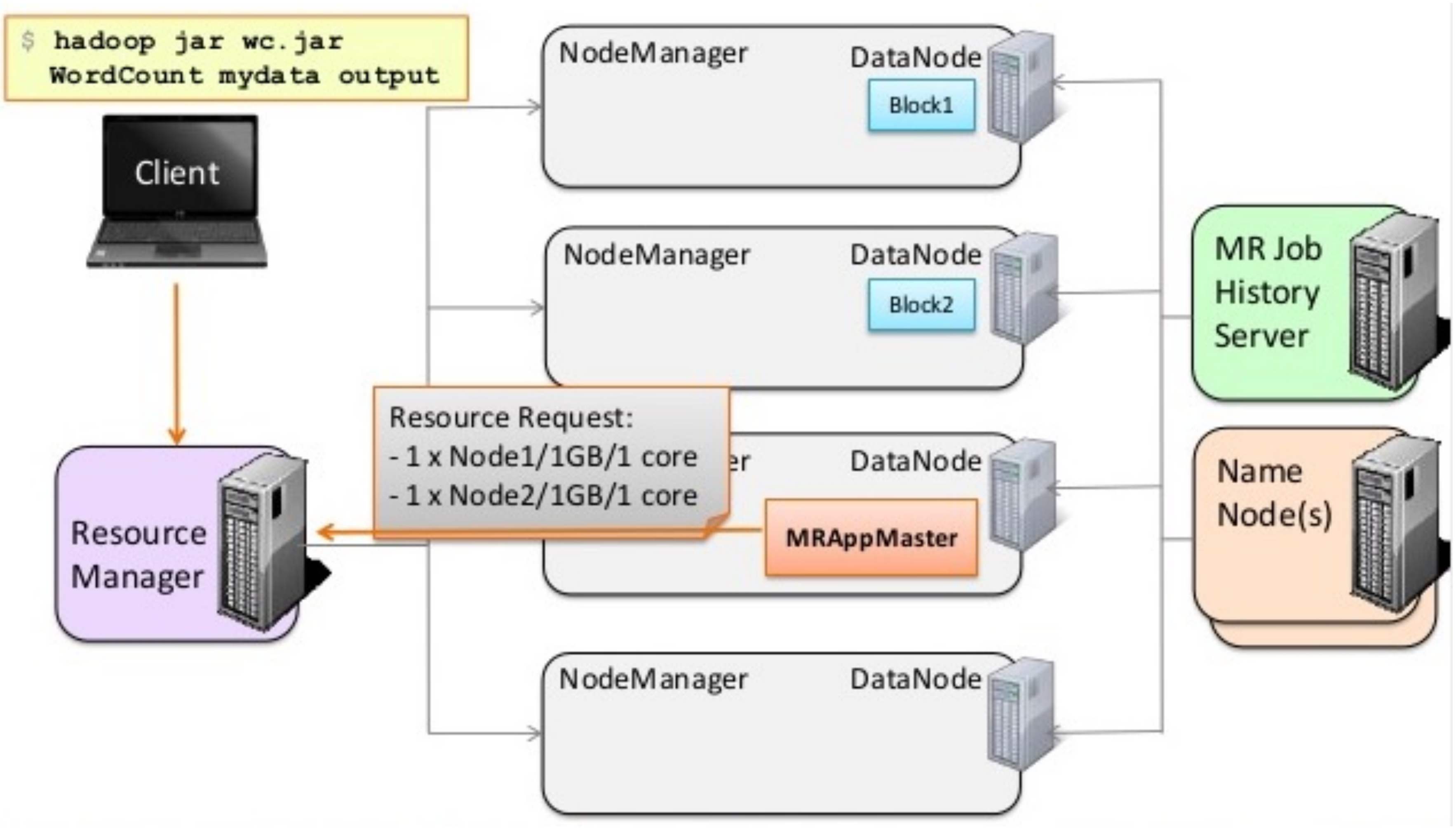
Running a MapReduce2 Application



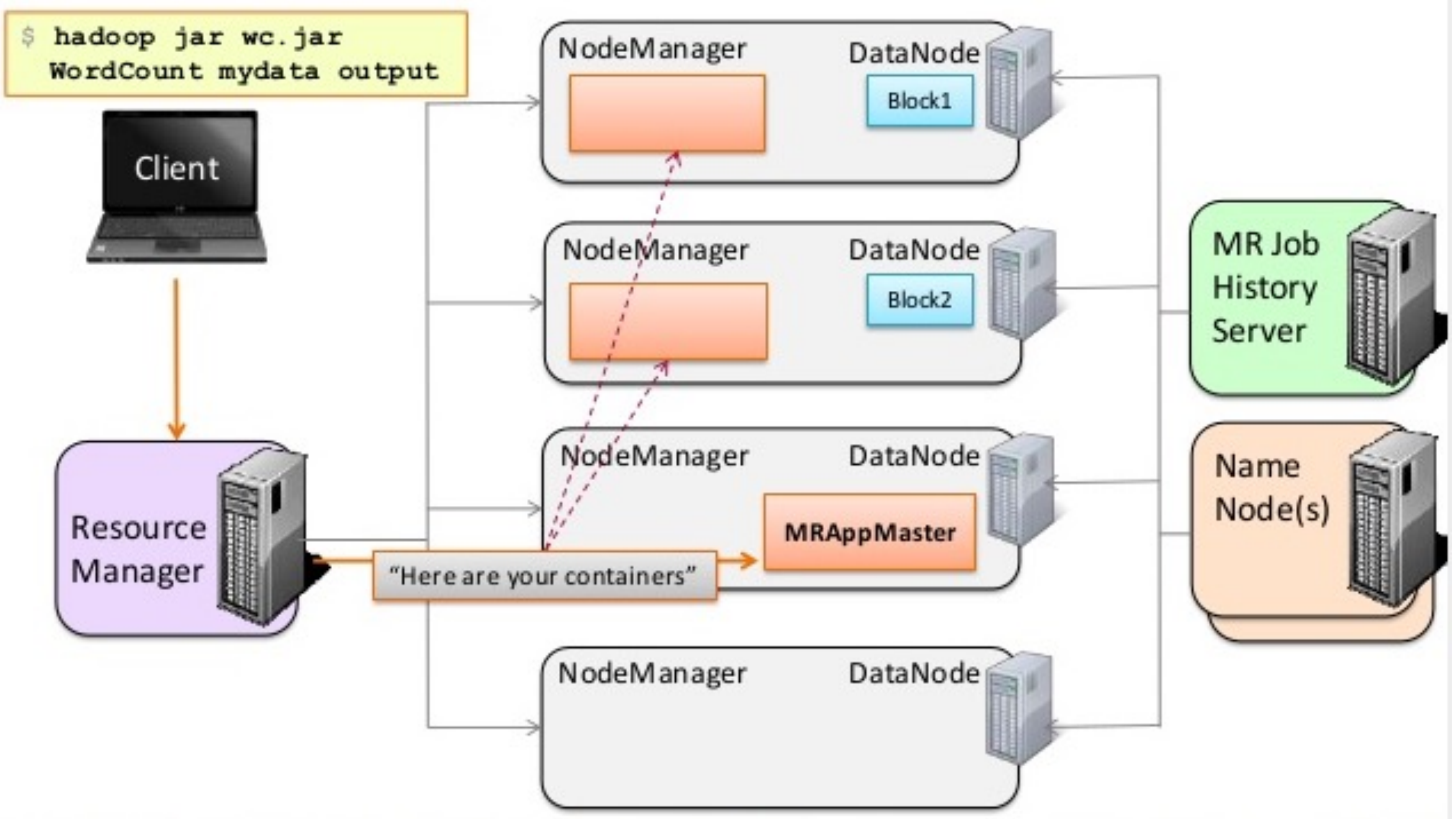
Running a MapReduce2 Application



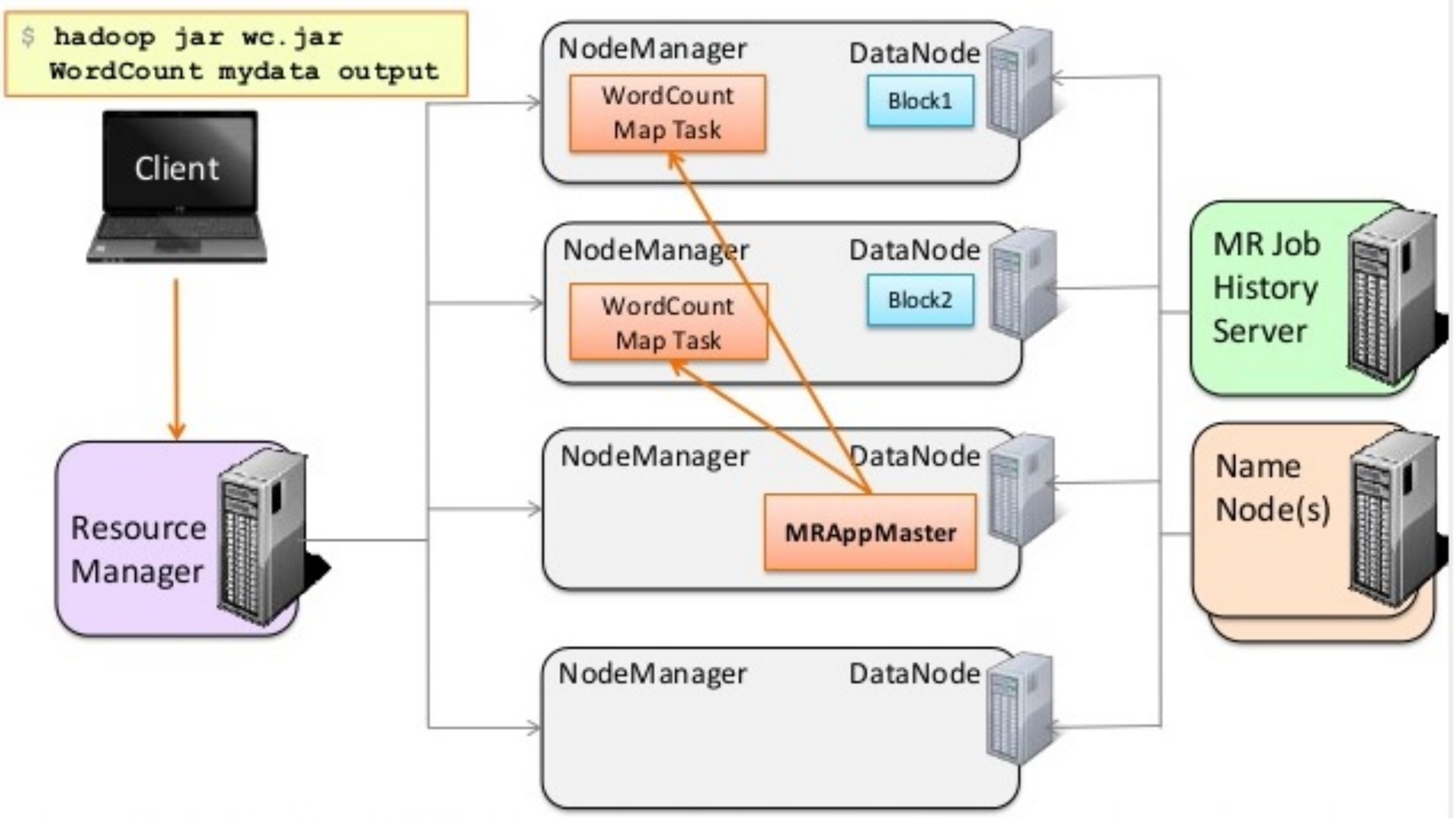
Running a MapReduce2 Application



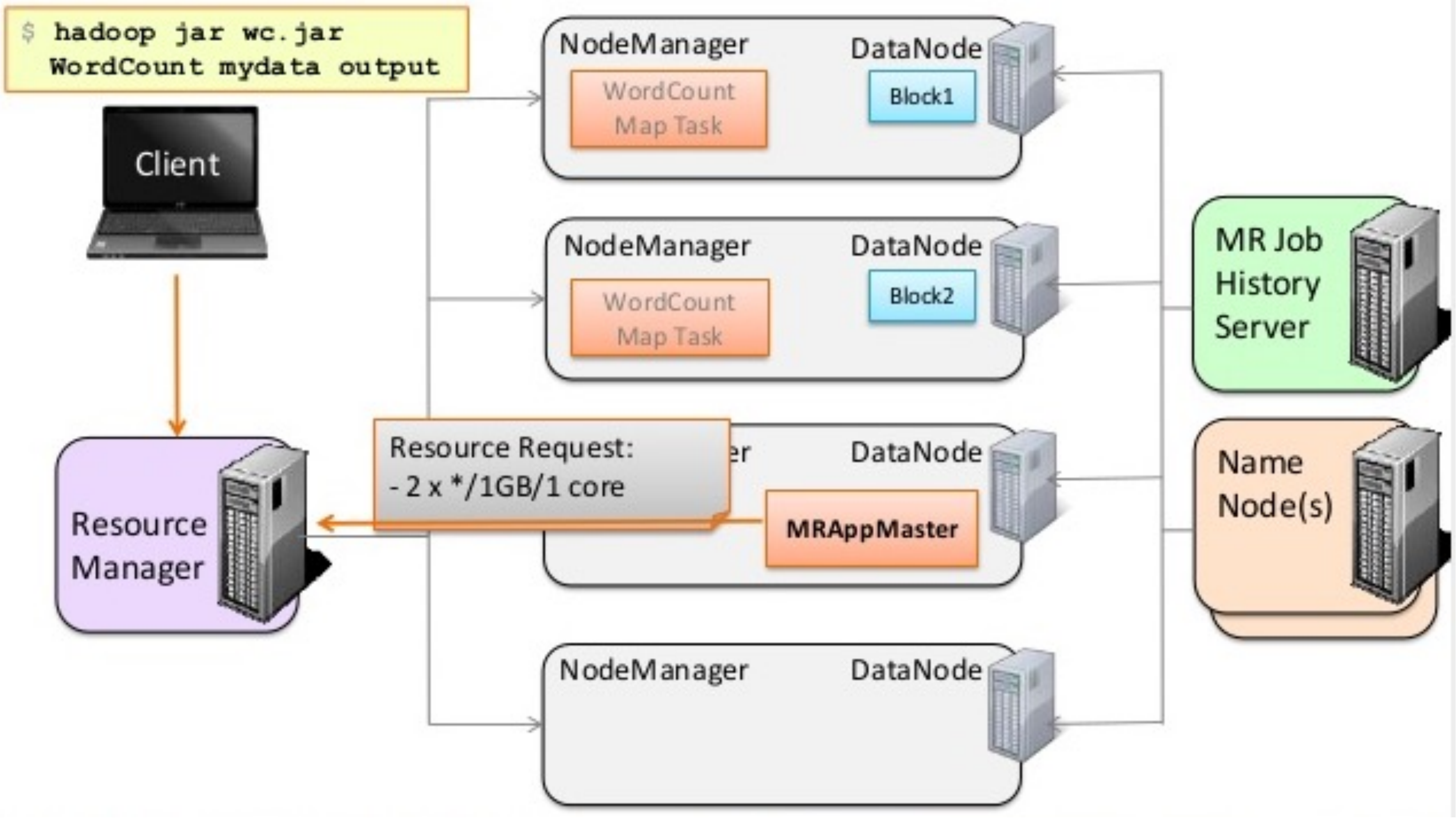
Running a MapReduce2 Application



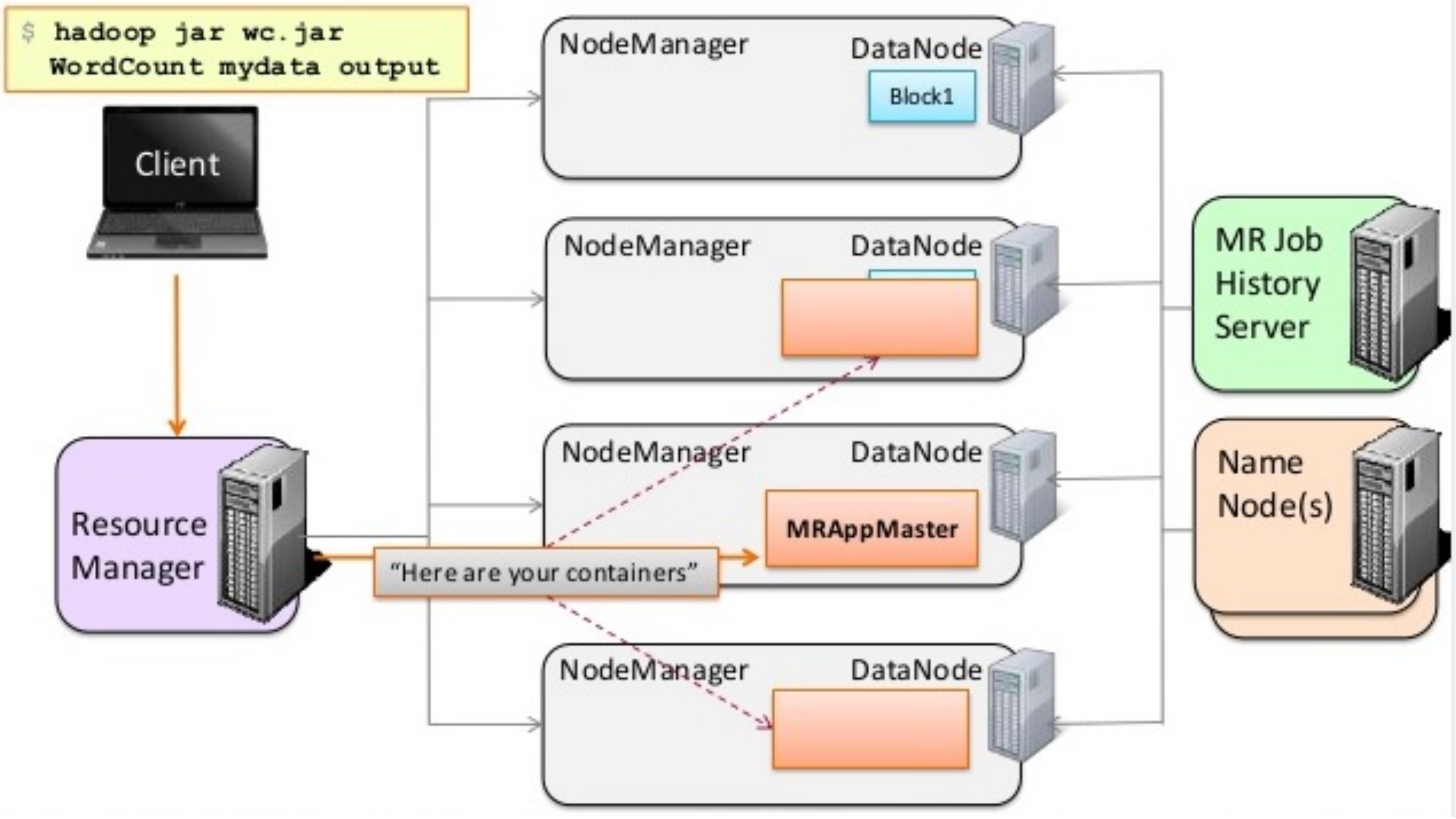
Running a MapReduce2 Application



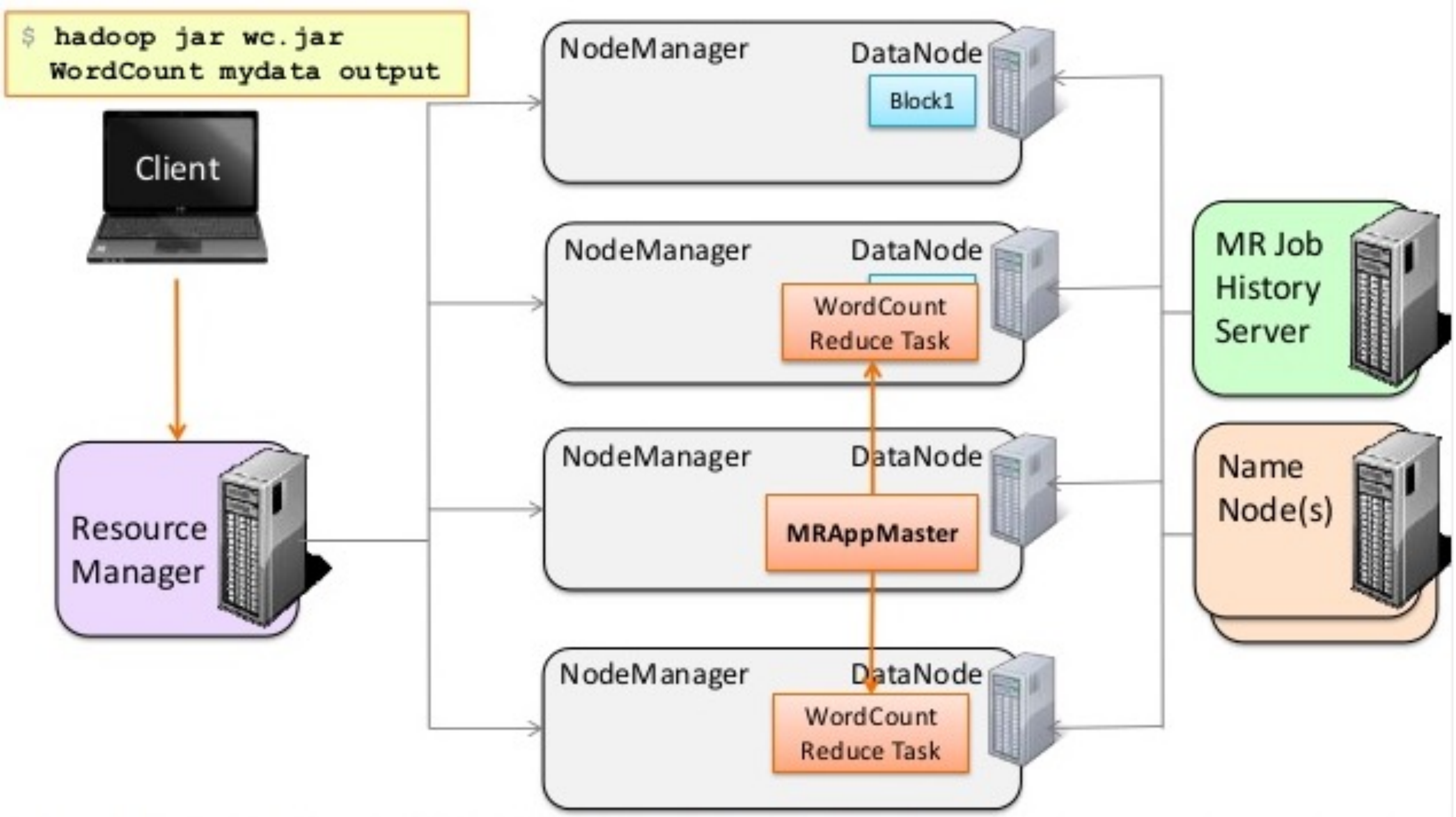
Running a MapReduce2 Application



Running a MapReduce2 Application

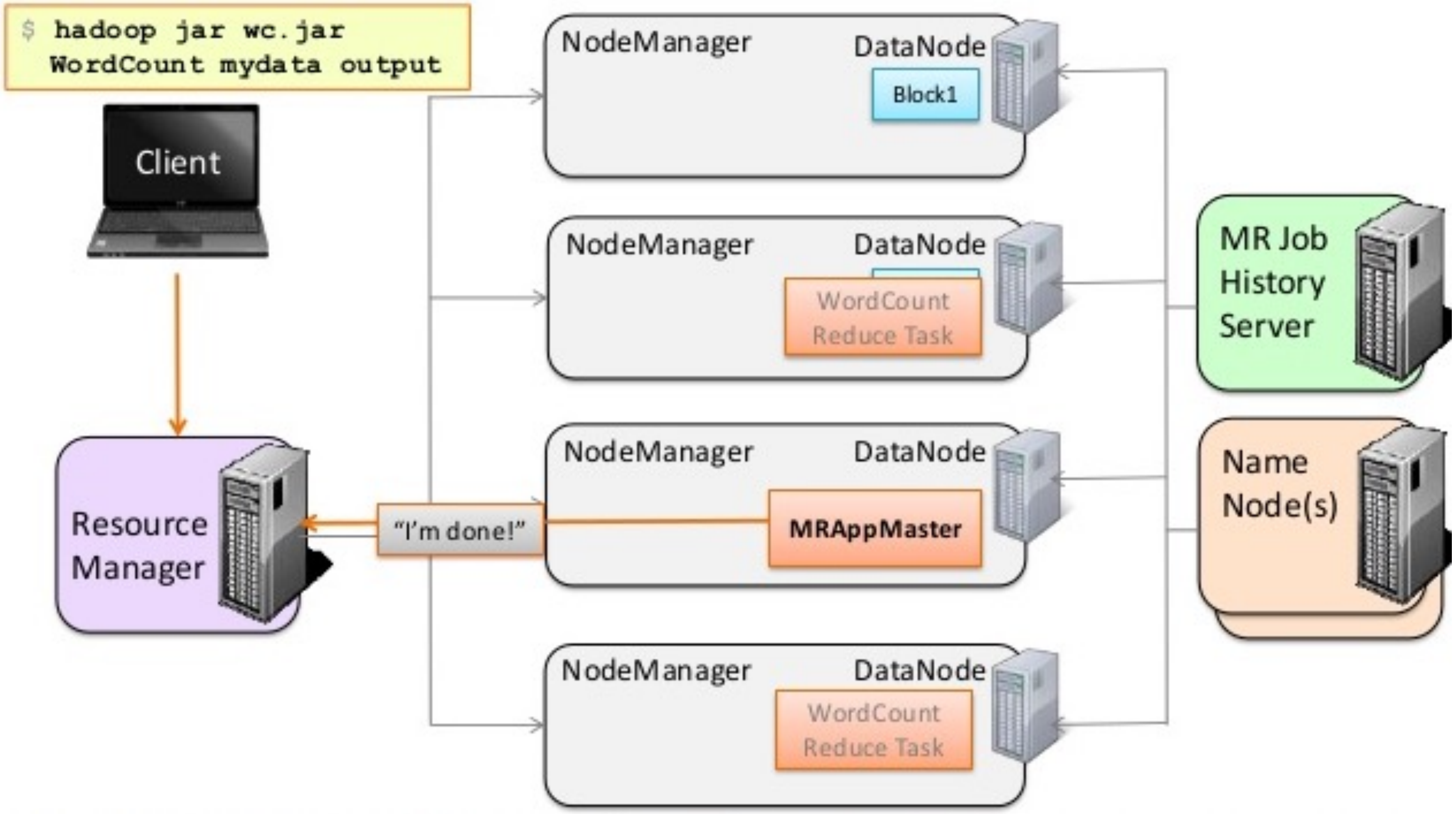


Running a MapReduce2 Application



An output of mapper is not stored on HDFS as this is temporary data and writing on HDFS will create unnecessary many copies.

Running a MapReduce2 Application



YARN: Fault Tolerance

- Task (Container)
  - MRAppMaster will re-attempt tasks that complete with exceptions or stop responding (4 times by default)
  - Applications with too many failed tasks are considered failed

- Application Master
  - If application fails or if AM stops sending heartbeats, RM will re-attempt the whole application (2 times by default)
  - MRAppMaster optional seeting: Job recovery
    - If false, all tasks will re-run
    - If true, MRAppMaster retrieves state of tasks when it restarts; only incomplete tasks will re-run

YARN: Fault Tolerance

- Ressource Manager (RM)
  - No applications or tasks can be launched if RM is unavailable
  - Can be configured with High Availibility

- NodeManager (NM)
  - If NM stops sending heartbeats to RM, it is removed from list of active nodes
  - Tasks on the node will be treated as failed by MRAppMaster
  - If the MRAppMaster node fails, it will treated as a failed application

First MapReduce Job

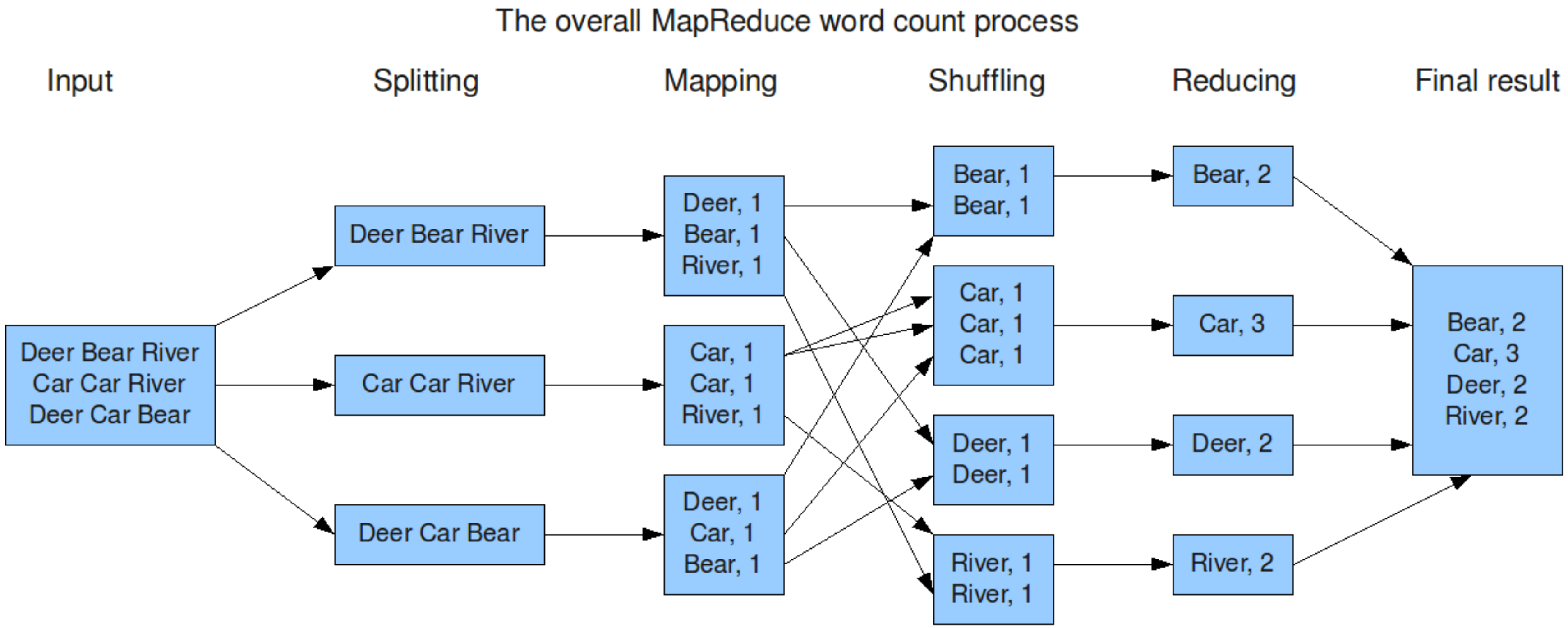
MapReduce in practice

- Job - execution of map and reduce functions to accomplish a task
  - Equal to Java’s main
- Task - single Mapper or Reducer
  - Performs work on a fragment of data

First Map Reduce Job

- **WordCount Job**
  - Input is a body of text from HDFS
  - Split text into tokens
  - For each word sum up all occurrences
  - Output to HDFS

WordCount example (recall)



WordCount Job in JAVA

- **Configure the Job**
  - Specify Input, Output, Mapper, Reducer and Combiner
- **Implement Mapper**
  - Input is text – a line from the text file
  - Tokenize the text and emit each word with a count of 1
    <token, 1>
- **Implement Reducer**
  - Sum up counts for each word
  - Write out the result to HDFS
- **Run the job**

WordCount Job in JAVA

```
package org.myorg;

import java.io.IOException;
import java.util.*;

import org.apache.hadoop.fs.Path;
import org.apache.hadoop.conf.*;
import org.apache.hadoop.io.*;
import org.apache.hadoop.mapreduce.*;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.input.TextInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
import org.apache.hadoop.mapreduce.lib.output.TextOutputFormat;

```

WordCount Job in JAVA

```
public class WordCount {

 public static class Map extends Mapper<LongWritable, Text, Text, IntWritable> {
   private final static IntWritable one = new IntWritable(1);
   private Text word = new Text();

   public void map(LongWritable key, Text value, Context context) throws IOException, InterruptedException {
       String line = value.toString();
       StringTokenizer tokenizer = new StringTokenizer(line);
       while (tokenizer.hasMoreTokens()) {
           word.set(tokenizer.nextToken());
           context.write(word, one);
       }
   }
 }

```

WordCount Job in JAVA

```
public static class Reduce extends Reducer<Text, IntWritable, Text, IntWritable> {

   public void reduce(Text key, Iterable<IntWritable> values, Context context)
     throws IOException, InterruptedException {
       int sum = 0;
       for (IntWritable val : values) {
           sum += val.get();
       }
       context.write(key, new IntWritable(sum));
   }
}

```

WordCount Job in JAVA

```
public static void main(String[] args) throws Exception {
  Configuration conf = new Configuration();

    Job job = new Job(conf, "wordcount");

  job.setOutputKeyClass(Text.class);
  job.setOutputValueClass(IntWritable.class);

  job.setMapperClass(Map.class);
  job.setReducerClass(Reduce.class);

  job.setInputFormatClass(TextInputFormat.class);
  job.setOutputFormatClass(TextOutputFormat.class);

  FileInputFormat.addInputPath(job, new Path(args[0]));
  FileOutputFormat.setOutputPath(job, new Path(args[1]));

  job.waitForCompletion(true);
}
} // end of class
```

WordCount Job in SCALA/SPARK

```
val textFile = spark.textFile("hdfs://...")
val counts = textFile.flatMap(line => line.split(" "))
             .map(word => (word, 1))
             .reduceByKey(_ + _)
counts.saveAsTextFile("hdfs://...")
```

What is flatMap?

```
scala> val list = List(1,2,3,4,5)
list: List[Int] = List(1, 2, 3, 4, 5)

scala> def g(v:Int) = List(v-1, v, v+1)
g: (v: Int)List[Int]

scala> list.map(x => g(x))
res0: List[List[Int]] = List(List(0, 1, 2), List(1, 2, 3), List(2, 3, 4), List(3, 4, 5), List(4, 5, 6))

scala> list.flatMap(x => g(x))
res1: List[Int] = List(0, 1, 2, 1, 2, 3, 2, 3, 4, 3, 4, 5, 4, 5, 6)
```

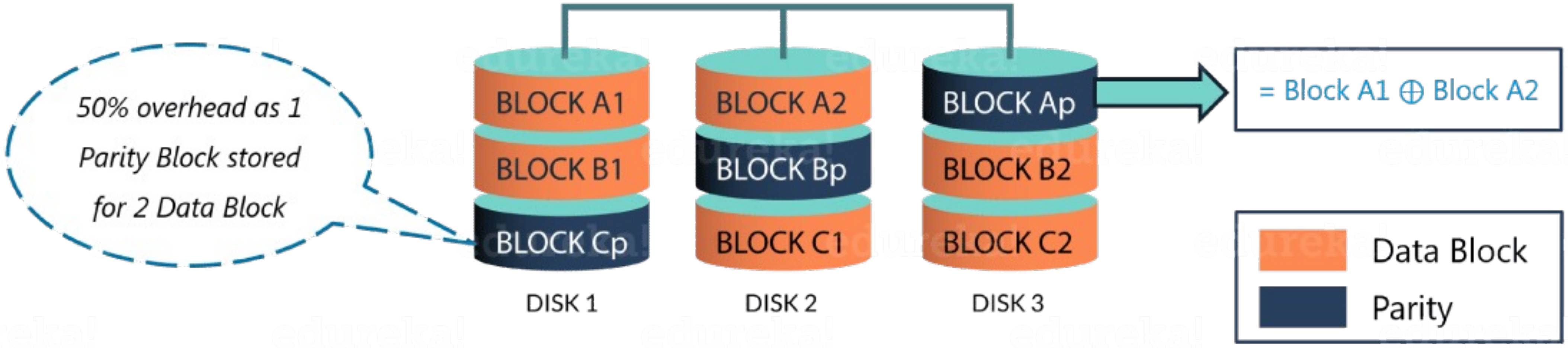
Hadoop 3

What’s New in Hadoop 3?

1- Java 8

- All the Hadoop jar files have been compiled using Java 8 run time version.
- The user now has to install Java 8 to use Hadoop 3.0.

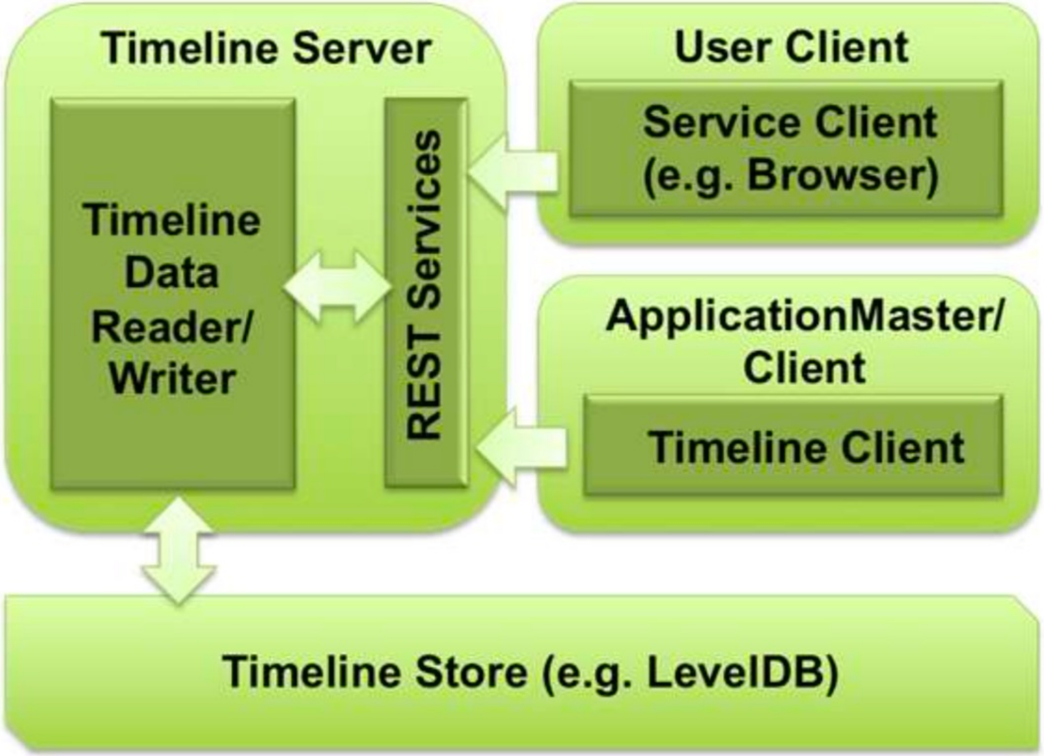
2- Erasure coding



3- YARN Timeline Service

- The YARN Timeline Service is a web application which can be deployed as part of a YARN cluster.
- It logs events from sources in the cluster, including for example Hadoop MapReduce.
- For Apache Spark, it can store the lifecycle events normally logged to a file, then replay them later in the Spark History Server Web UI.

Application Timeline Service V1



- REST is an acronym for **RE**presentational **S**tate **T**ransfer and an architectural style for **distributed hypermedia systems**.
- A Web API (or Web Service) conforming to the REST architectural style is a REST API.

The YARN Timeline server

- It keeps the information for current and historic applications executed on the YARN cluster.
- It provides metrics visibility for YARN applications, similar to the functionality the Job History Server provides for MapReduce.
- It keeps traces of failures of Application Master and Data for current running application.

YARN Timeline Service v.2

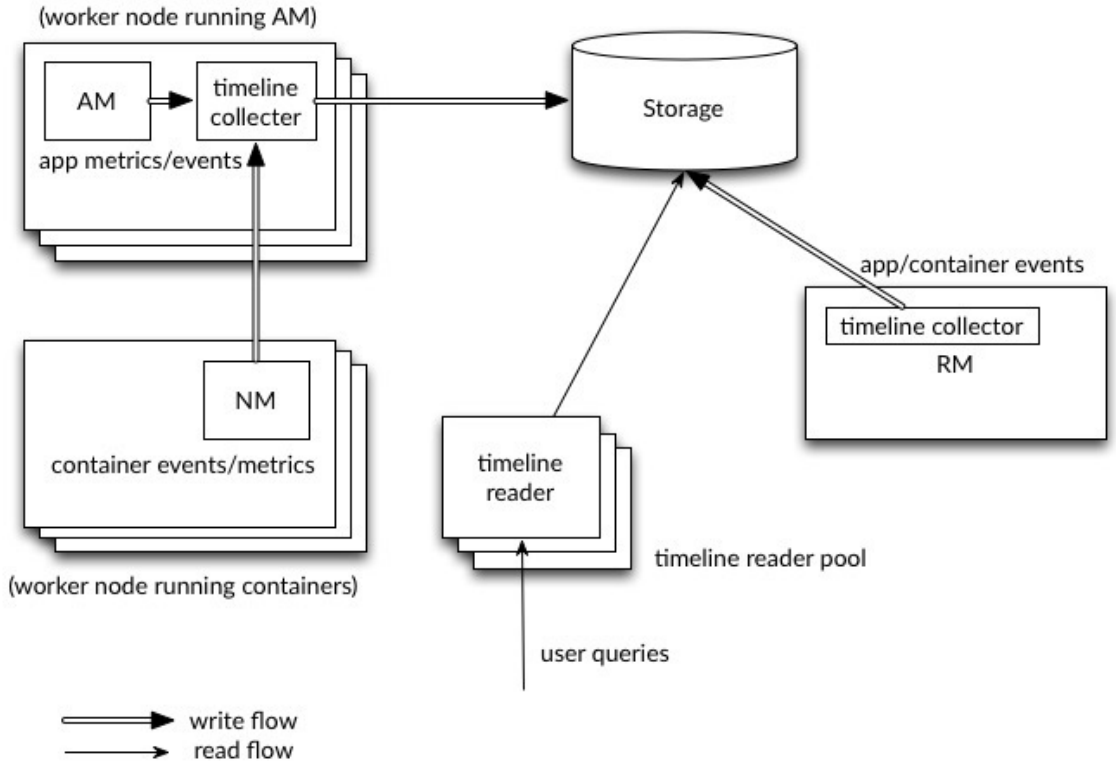
- It is developed to address two major challenges:
  1. Improving scalability and reliability of Timeline Service
  2. Enhancing usability by introducing flows and aggregation

- In version v.2 Timeline server has a distributed writer architecture and scalable backend storage. It separates collection (writer) of data from serving (read) of data.
- Also, it uses one collector per YARN application. It has a reader as a separate instance which servers query request via REST API.
- Timeline server v.2 uses HBase for storage which can get scaled to huge size giving good response time for reads and writes.

Architecture

- For a given application, the application master can write data for the application to the co-located timeline collectors.
- Node managers of other nodes that are running the containers for the application also write data to the timeline collector on the node that is running the application master.
- The resource manager also maintains its own timeline collector.
- The timeline readers are separate daemons separate from the timeline collectors, and they are dedicated to serving queries via REST API.

- AM: Application Master
- NM: Node Manager
- RM: Ressource Manager



4- Support for Opportunistic Containers

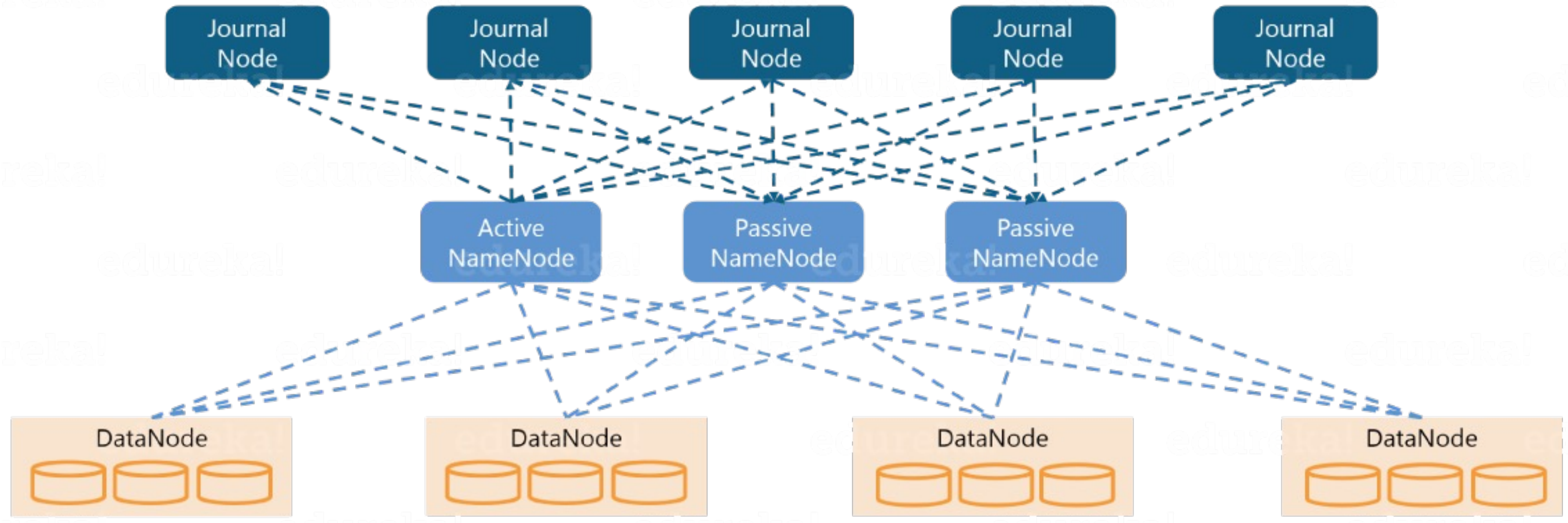
- **Guaranteed containers** correspond to the existing YARN containers.

- **Opportunistic containers** can be dispatched for execution at a NodeManager even if there are no resources available at the moment of scheduling.
  - In such a case, these containers will be queued at the NM, waiting for resources to be available for it to start.
  - Opportunistic containers are of lower priority than the default Guaranteed containers and are therefore preempted, if needed, to make room for Guaranteed containers.

- This should improve cluster utilization.

5. Support for More Than Two NameNodes

- Hadoop 2.0 supported single active NameNode and single standby NameNode.
  - This architecture allowed for the failure of one NameNode.
- Hadoop 3.0 tolerates the failure of two NameNodes.
  - Hadoop 3.0 has made the system more highly available.

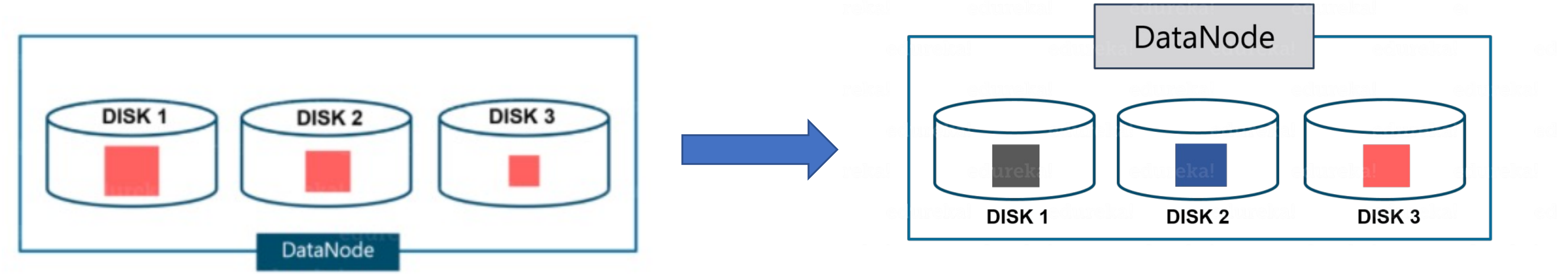


6- Intra-DataNode Balancer

- A single DataNode manages multiple disks. During a normal write operation, data is divided evenly and thus, disks are filled up evenly.

- But adding or replacing disks leads to skew within a DataNode. This situation was not handled by the existing HDFS balancer.

- Now Hadoop 3 handles this situation by the new intra-DataNode balancing functionality, which is invoked via the hdfs diskbalancer.



7. Reworked Daemon and Task Heap Management

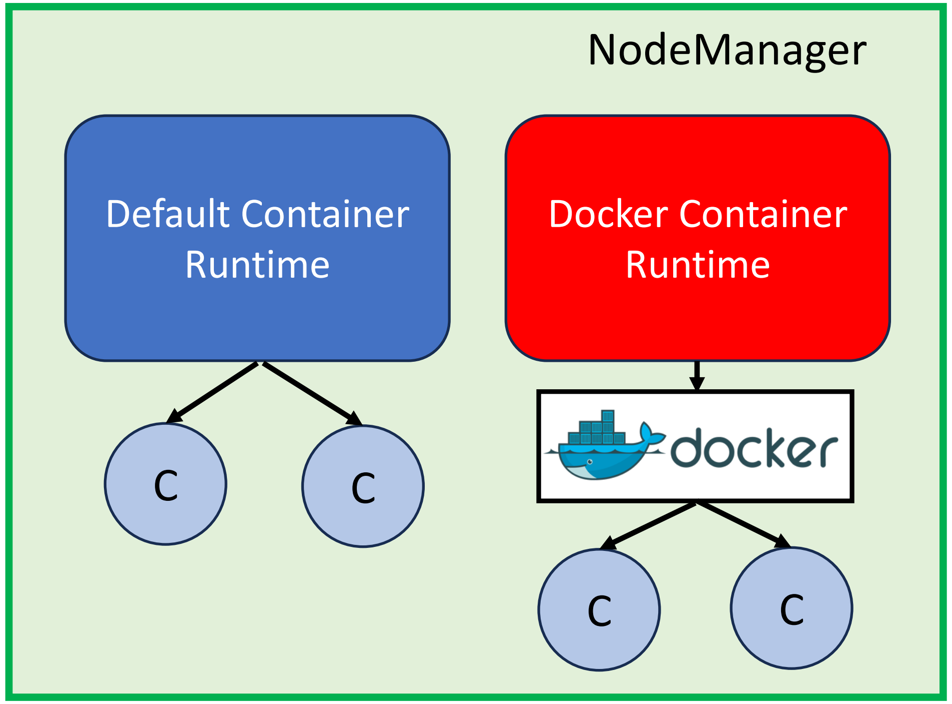
- A series of changes have been made to heap management for Hadoop daemons as well as MapReduce tasks.

8- Generalization of Yarn Resource Model

- The Linux Container Executor (LCE) allows the YARN NodeManager to launch YARN containers to run either directly on the host machine or inside Docker containers.

- The application requesting the resources can specify for each container how it should be executed.

- When the LCE launches a YARN container to execute in a Docker container, the application can specify the Docker image to be used.



Docker Containers

- Docker combines an easy-to-use interface to Linux containers with easy-to-construct image files for those containers.
- In short, Docker enables users to bundle an application together with its preferred execution environment to be executed on a target machine.
- Docker for YARN provides both **consistency** (all YARN containers will have the same software environment) and **isolation** (no interference with whatever is installed on the physical machine).
- Docker containers provide a custom execution environment in which the application's code runs, isolated from the execution environment of the NodeManager and other applications.
- These containers can include special libraries needed by the application, and they can have different versions of native tools and libraries including Perl, Python, and Java.

Container with GPU Ressources

- GPU scheduling: containers with GPU request can be placed to machines with enough available GPU resources.
- GPU isolation: when multiple applications use GPU resources on the same machine, they should not affect each other.
- GPU discovery: For properly doing scheduling and isolation, we need to know how many GPU devices are available in the system.
- Web UI: GPU information were added to the new YARN web UI.
- Note: FPGA ressource is also supported.

Conclusion

Conclusion

- Global resource management with YARN (or alternative solutions)
- MapReduce is the heart of Hadoop-based data processing
- The MapReduce concept is fairly simple to understand for those who are familiar with clustered scale-out data processing solutions
- Designing an algorithm based on MapReduce is not always easy
- Tools exploiting the MapReduce programming model: SPARK, etc.
- Significant evolution towards data science, high performance computing, artificial intelligence

YARN Commands

Configure YARN

Config File	Description
yarn-env.sh	A bash script where YARN environment variables are specified. For example, configure log directory here.
yarn-site.xml	Hadoop configuration file where majority of properties are specified for YARN daemons. Configures Resource Manager, Node Manager and History Server.
slaves	A list of nodes where Node Manager daemons are started; one host per line.
mapred-site.xml	MapReduce specific properties go here. This is the application specific configuration file; an application is MapReduce in this case.

- Note: YARN will also utilize core-site.xml and hadoop-env.sh which were covered in HDFS lecture.

Example - yarnsite.xml

<property>

<name>**yarn.resourcemanager.address**</name>

<value>localhost:10040</value>

<description>In Server specified the port that Resource Manager will run on. In client is used for connecting to Resource Manager</description>

</property>

<property>

<name>**yarn.nodemanager.local-dirs**</name>

<value>/home/hadoop/Training/hadoop_work/mapred/nodemanager</value>

<final>true</final>

<description>Comma separated list of directories, where local data is persisted by Node Manager</description>

</property>

YARN Web-UI

- **Resource Manager Web-UI**
  - Cluster resource usage, job scheduling, and current running jobs
  - Runs on port 8088 by default
- **Application Proxy Web-UI**
  - Provides information about the current job
  - Runs as a part of Resource Manager Web-UI by default
  - After completion, jobs get exposed by History Server
- **Node Manager Web-UI**
  - Single Node information and current containers being executed
  - Runs on port 8042 by default
- **MapReduce History Server Web-UI**
  - Provides history and details of past MapReduce jobs
  - Runs on port 19888 by default

Command Line Tools

- **<hadop_install>/bin/yarn**
  - Execute code with a jar
  - $yarn jar jarFile [mainClass] args...
  - Print out CLASSPATH: $yarn classpath
  - Resource Manager admin: $yarn rmadmin

- **<hadop_install>/bin/mapred**
  - $mapred job
    - Get information about jobs
    - Kill Jobs

$ yarn jar jarFile [mainClass] args...

- Exec

```
$ yarn jar
$HADOOP_HOME/share/hadoop/mapreduce/hadoop-
mapreduce-examples-2.0.0-cdh4.0.0.jar pi 5 5
```

Examples jar files shipped with hadoop

pi is the program that computes pi

Specify number of mappers

Number of samples; artifact of pi application

$ yarn rmadmin

- Runs ResourceManager admin client
- Allows to refresh and clear resources

```
$ yarn rmadmin -refreshNodes
```

Resource Manager will refresh its information
about all the Node Managers

$mapred job

- Command line interface to view job's attributes
- Most of the information is available on Web-UI

List Jobs that are currently running

```
$ mapred job -list

$ mapred job -status job_1340417316008_0001
```

Retrieve job's status by Job ID