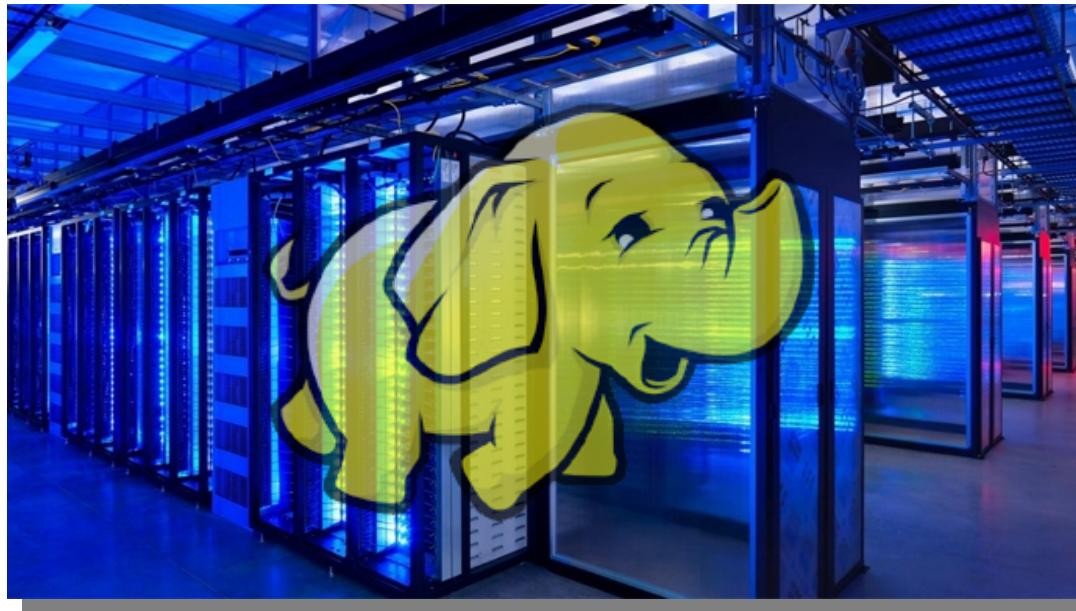


Big Data Analytics and Cluster Computing with Apache Hadoop

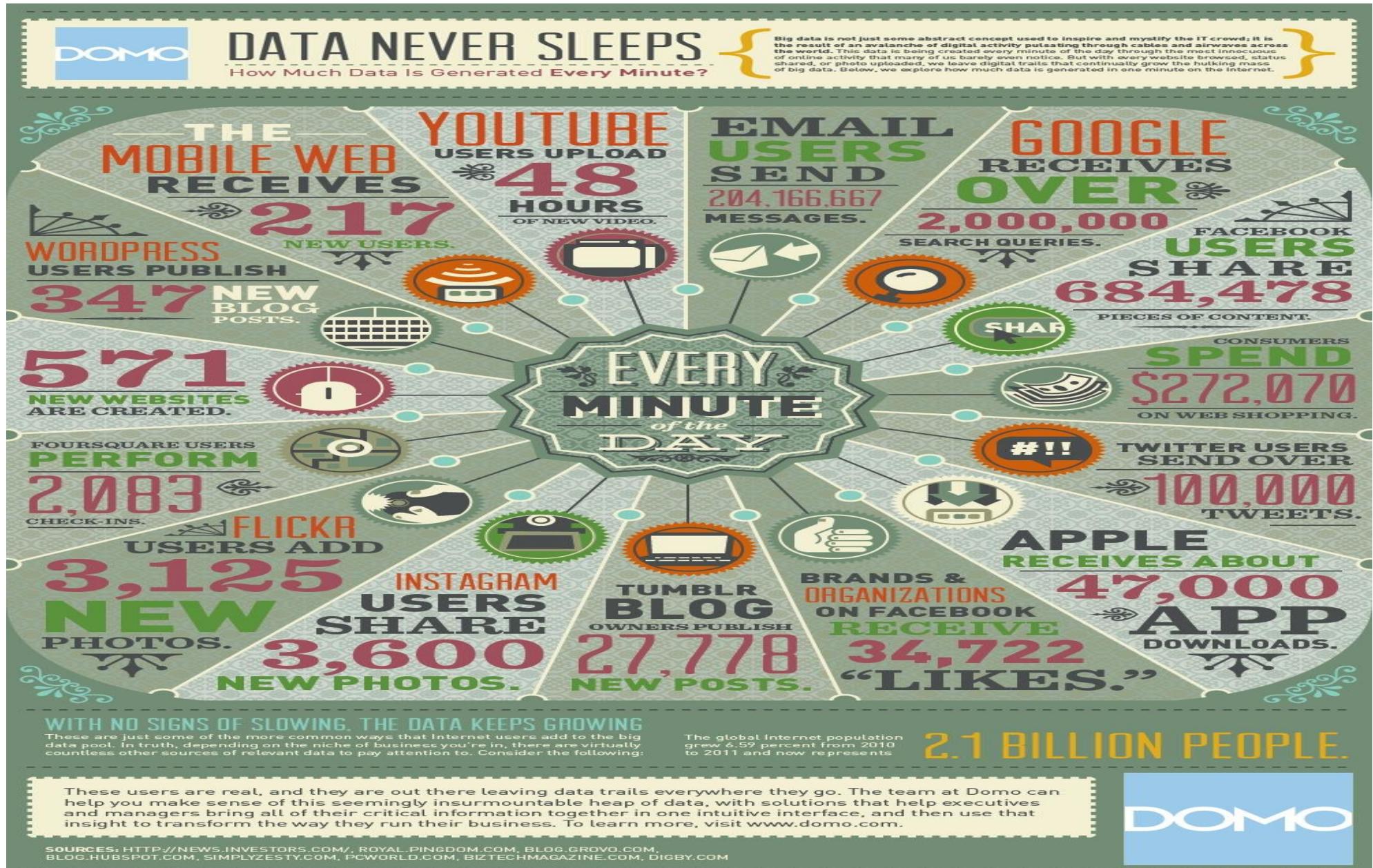


Apostolos N. Papadopoulos Christos Giatsidis

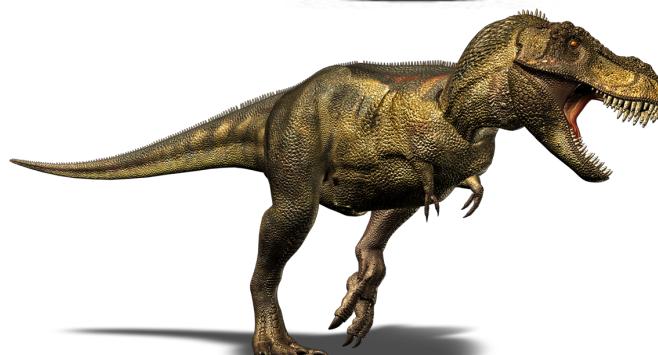
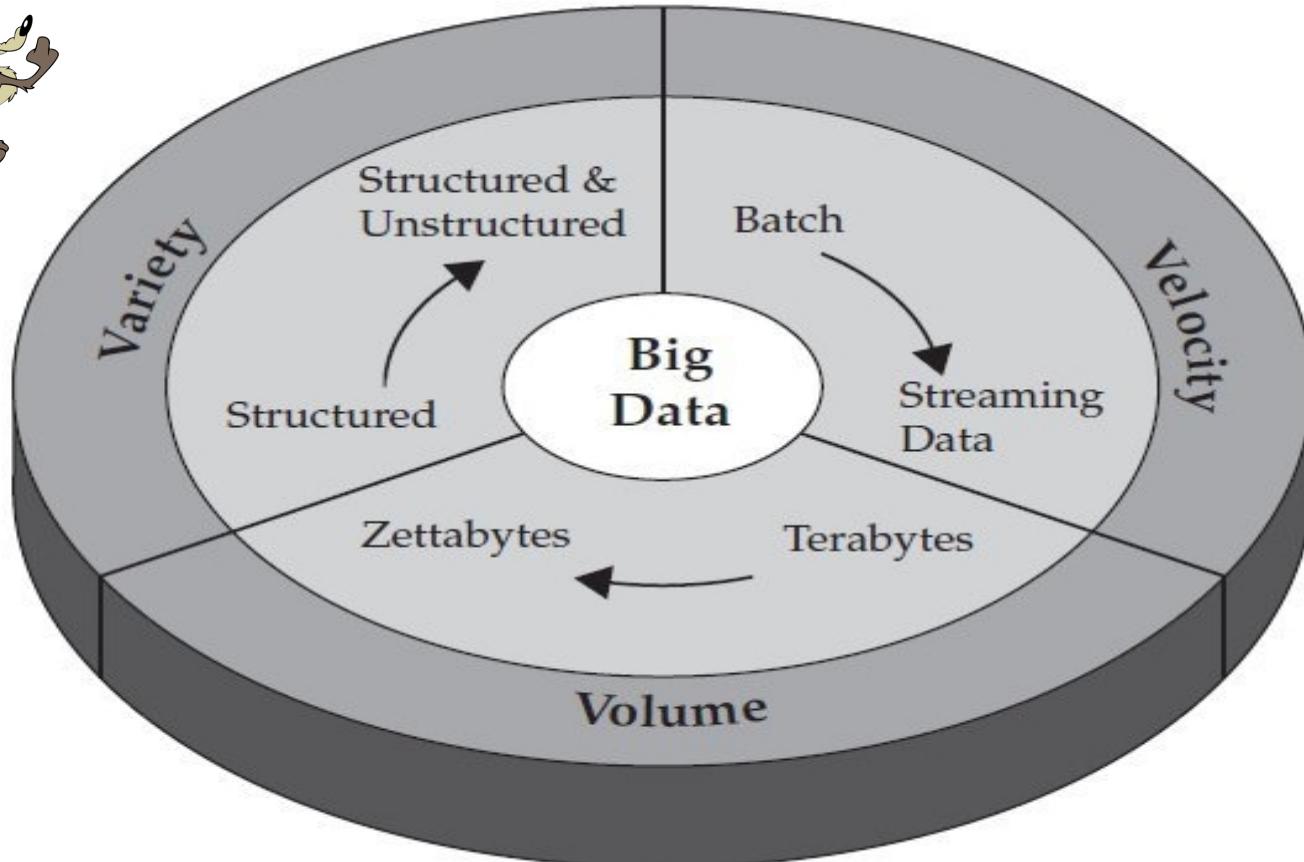
Outline

- Big Data everywhere
- Why one machine is not enough ?
- Parallel architectures
- Important issues in cluster computing
- Hadoop MapReduce
- Architecture of HDFS
- Architecture of YARN cluster manager
- MapReduce limitations

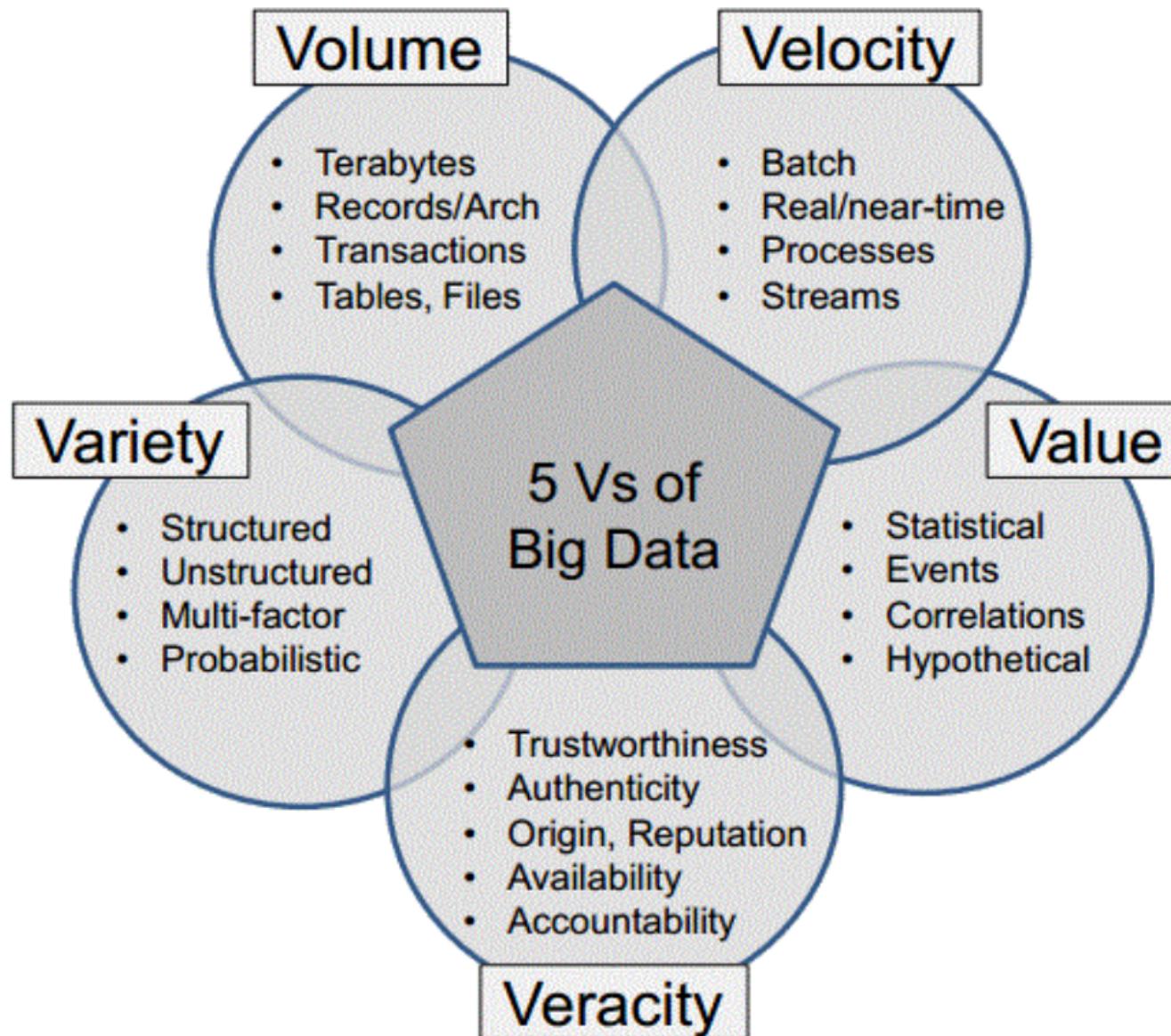
Big Data Everywhere



The 3 V's of Big Data

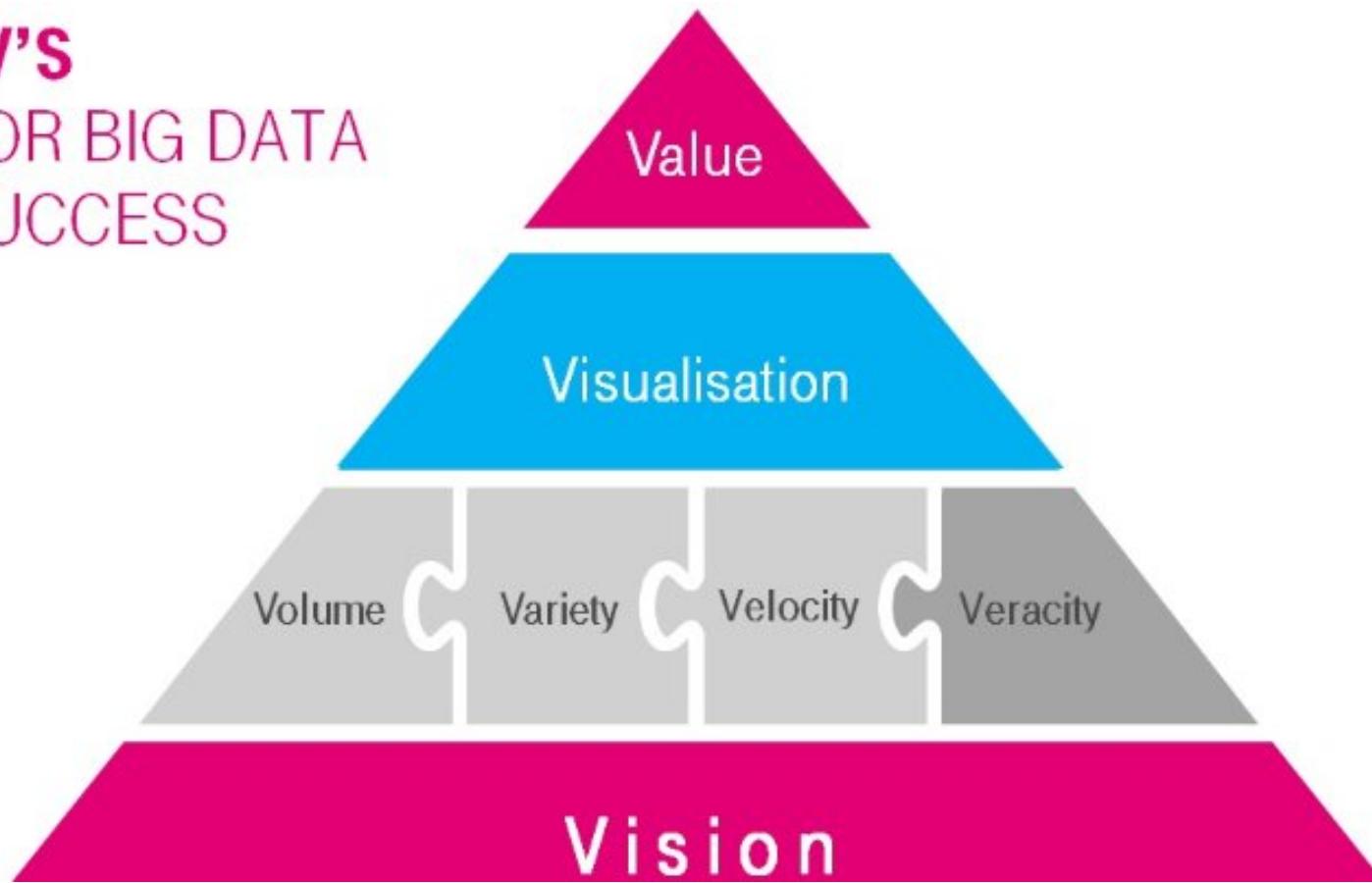


The 5 V's of Big Data

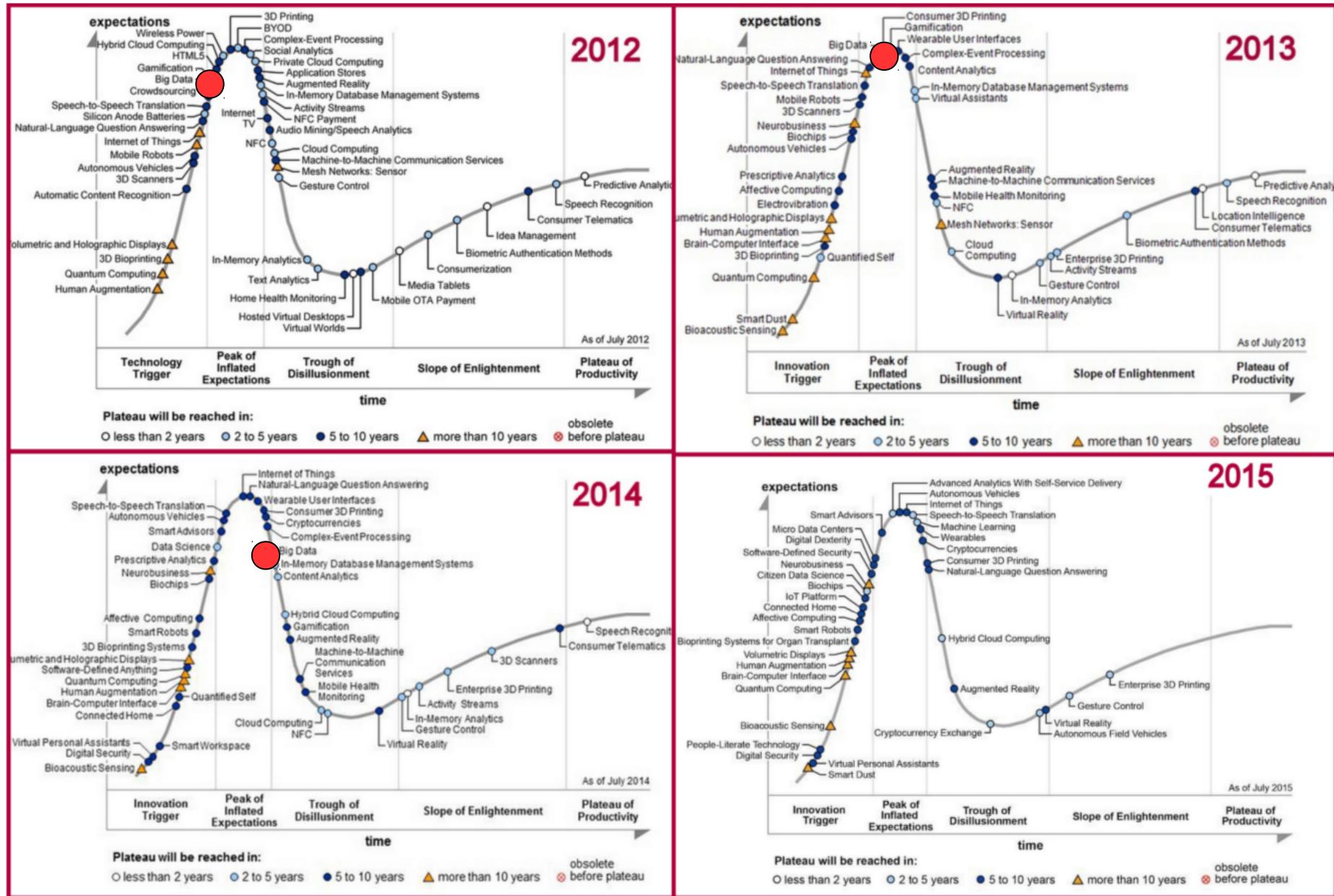


The 7 V's of Big Data

7V'S
FOR BIG DATA
SUCCESS



Big Data is not a Hype



Motivation

We need **more CPUs** because:

- we can run programs faster

We need **more disks** because:

- modern applications require huge amounts of data
- with many disks we can perform I/O in parallel

Assume that we are able to build a single disk with **500 TB** capacity. This is enough to store more than **20 billion webpages** (assuming an average size per page of **20KB**).

However, **just to scan** these 500 TB we need **more than 4 months** if the disk can bring **40 MB/sec**. Imagine the time required to process the data !

In the Near Future

“IBM Research and Dutch astronomy agency Astron work on new technology to handle one exabyte of raw data per day that will be gathered by the world largest radio telescope, the Square Kilometer Array, when activated in 2024.”

Some Challenges

- Scalability
- Load balancing
- Fault Tolerance
- Efficiency
- Data Stream processing
- Support for complex objects
- Accuracy/Speed tradeoffs (with performance guarantees)

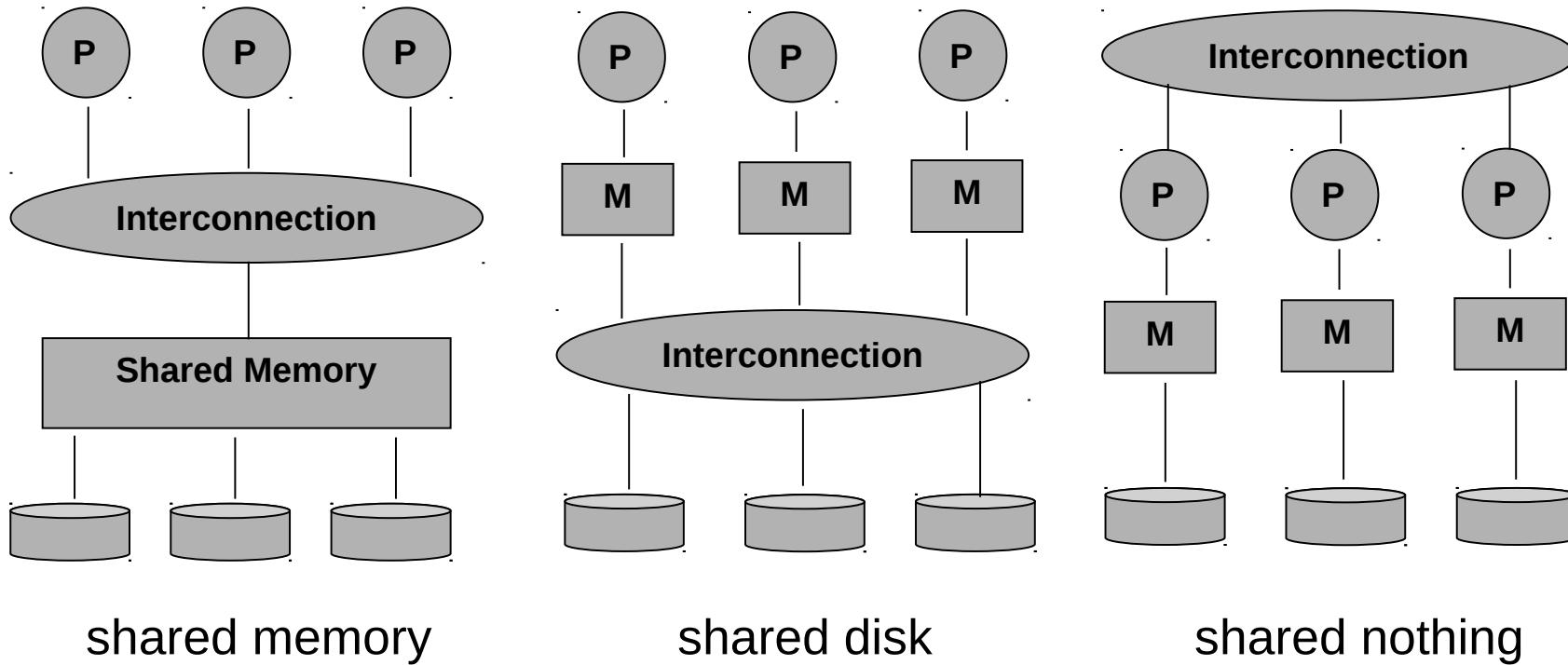
Parallel Architectures

Shared Memory: processors share a common main memory and also share secondary storage (e.g., disks)

Shared Disk: processors share only secondary storage, whereas each processor has its own private memory

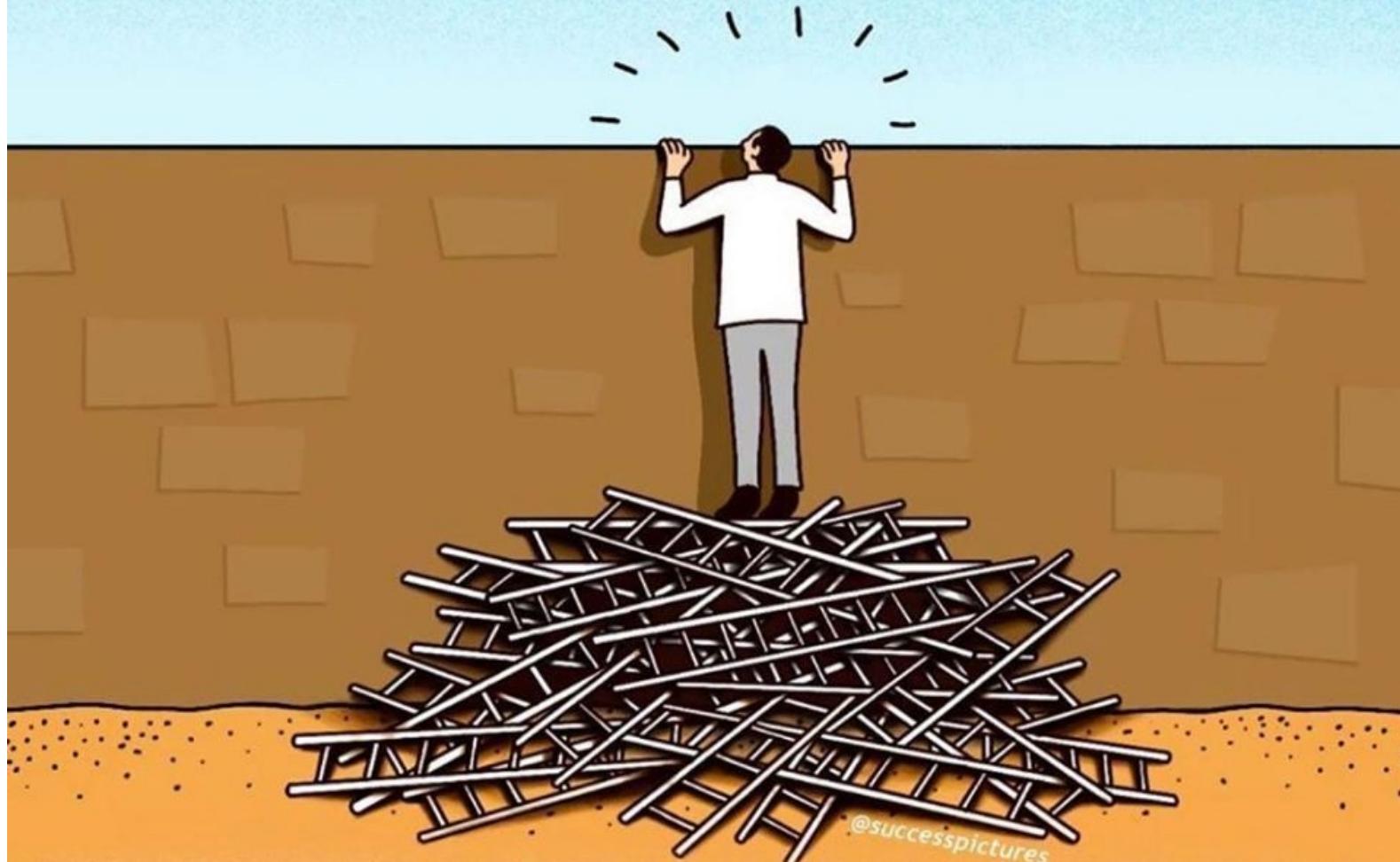
Shared Nothing: processors do not share anything, each one has private secondary storage and memory

Parallel Architectures



**IT DOESN'T MATTER HOW MANY
RESOURCES YOU HAVE.**

@successpictures



**IF YOU DON'T KNOW HOW TO USE
THEM, IT WILL NEVER BE ENOUGH.**

Scalability

Scale-Up: put more resources into the system to make it bigger and more powerful



Scale-Out: connect a large number of “ordinary” machines and create a cluster



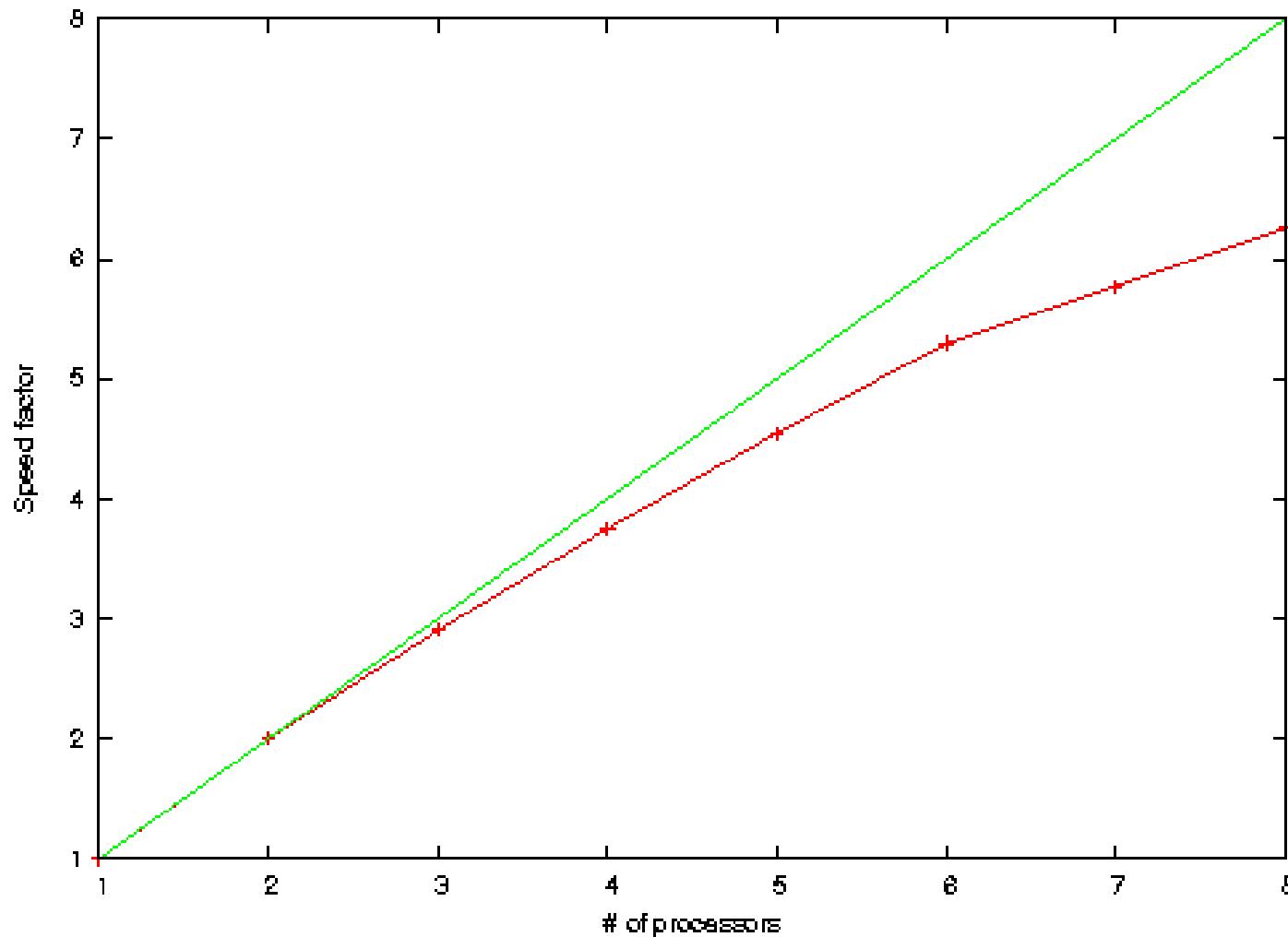
Scale-Out is **more powerful** than Scale-Up, and also **less expensive**

Scalability: measures

Among the three parallel architectures, shared-nothing is the one that **scales best**. This is the main reason for being adopted for building massively parallel systems (thousands of processors)

- **Speedup**: monitor performance by increasing the number of processors
- **Sizeup**: monitor performance by increasing only the dataset size
- **Scaleup**: monitor performance by increasing both the number of processors and the dataset size

The Speedup Curve



Real Curves are Non-Linear

Why ?

Start-up costs: cost for starting an operation in a processor

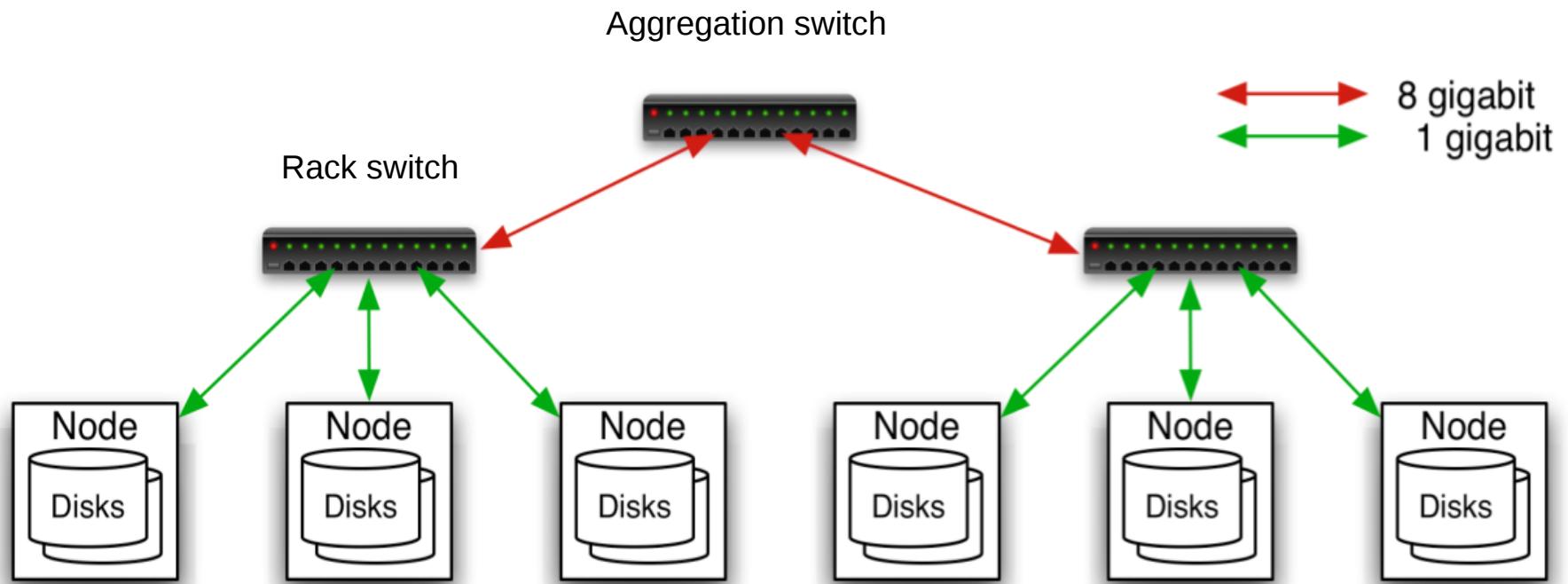
Interference: cost for communication among processors and resource congestion

Skew: either in data or tasks → the slowest processor becomes the bottleneck

Result formation: partial results from each processor must be combined to provide the final result.

Failures: in case of failures, some time must be devoted for recovery from the failure.

Cluster Configuration Example



- 40 nodes/rack, 1000-4000 nodes in cluster
- 1 Gbps bandwidth within rack, 8 Gbps out of rack
- 8 x 2GHz cores, 8 GB RAM, 4 disks (= 4 TB?)

Fault Tolerance

Failures are very common in massively parallel systems

Let P the probability that a disk will fail in the next month. If we have D disks in total, the probability that at least one disk will fail is given by:

$$\text{Prob}\{\text{at least one disk failure}\} = 1 - (1 - P)^D$$

e.g., $D = 10000$, $P = 0.0001$

$$\text{Prob}\{\text{at least one disk failure}\} = 0.63$$

Fault Tolerance

Failures may happen because of:

Hardware not working properly

- Disk failure

- Memory failure (8% of DIMMs have problems)

- Inadequate cooling (CPU overheating)

Resource unavailability

- Due to overload

We must provide fault tolerance in the cluster!

Fault Tolerance

Simplest protocol: if there is a failure, restart the job.

Assume a job that requires 1 week of processing.
If there is a failure once per week, the job will never finish!

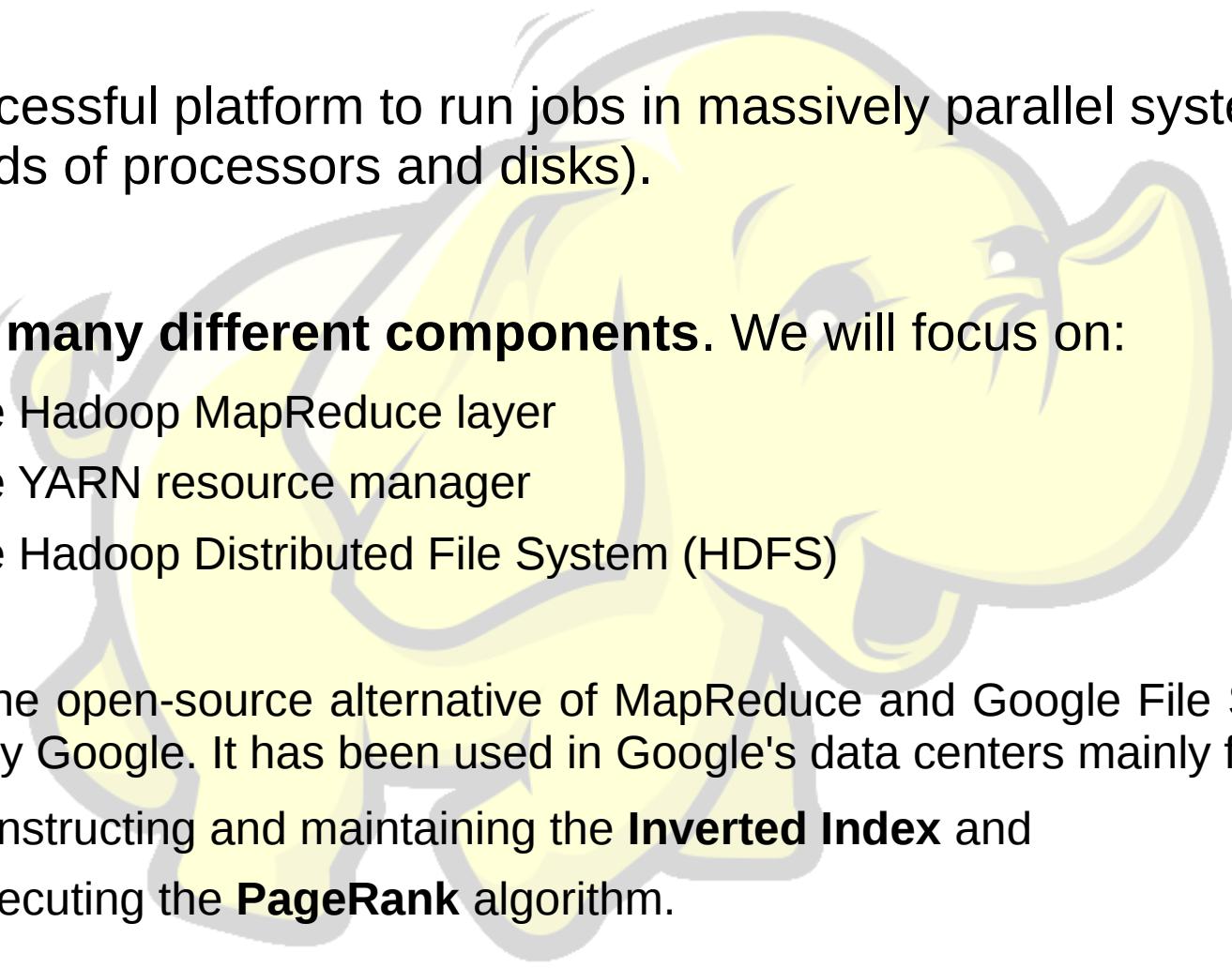
Fault Tolerance

A better protocol:

Replicate the data and also split the job in parts and replicate them as well. As an alternative, submit a smaller job (task) and if it fails then start another one.

A large job must be decomposed to simpler ones.

Meet Hadoop



A very successful platform to run jobs in massively parallel systems (thousands of processors and disks).

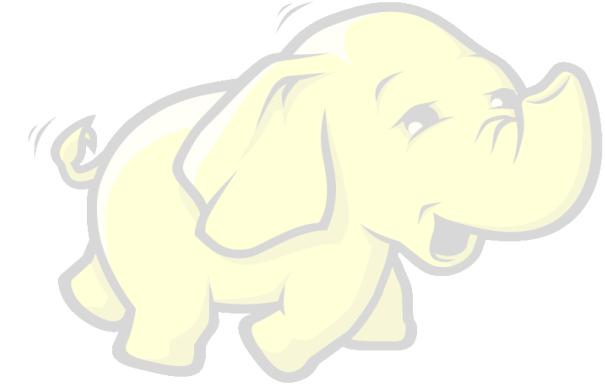
It contains **many different components**. We will focus on:

- the Hadoop MapReduce layer
- the YARN resource manager
- the Hadoop Distributed File System (HDFS)

Hadoop is the open-source alternative of MapReduce and Google File System (GFS) invented by Google. It has been used in Google's data centers mainly for:

- constructing and maintaining the **Inverted Index** and
- executing the **PageRank** algorithm.

Meet Hadoop



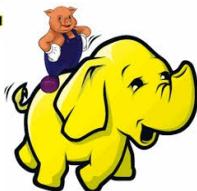
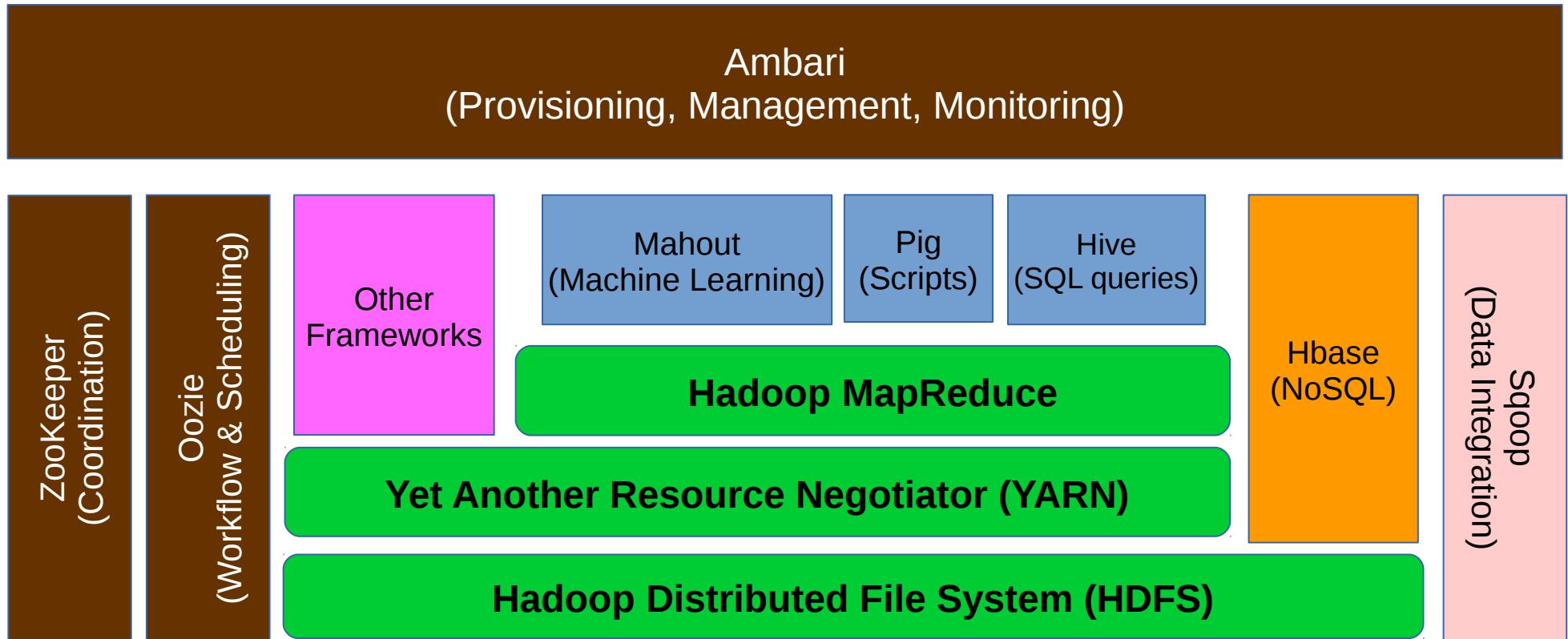
Doug Cutting, explains how the name came about



From Wikipedia: Douglass Read "Doug" Cutting is an advocate and creator of open-source search technology. He originated **Lucene** and, with Mike Cafarella, **Nutch**, both open-source search technology projects which are now managed through the Apache Software Foundation. He is also the creator of **Hadoop** (Yahoo!, Cloudera).

"The name my kid gave a **stuffed yellow elephant**. Short, relatively easy to spell and pronounce, meaningless, and not used elsewhere: those are my naming criteria. Kids are good at generating such. Googol is a kid's term"

The Hadoop Ecosystem



yarn

The Hadoop Ecosystem

MRv1

vs

MRv2

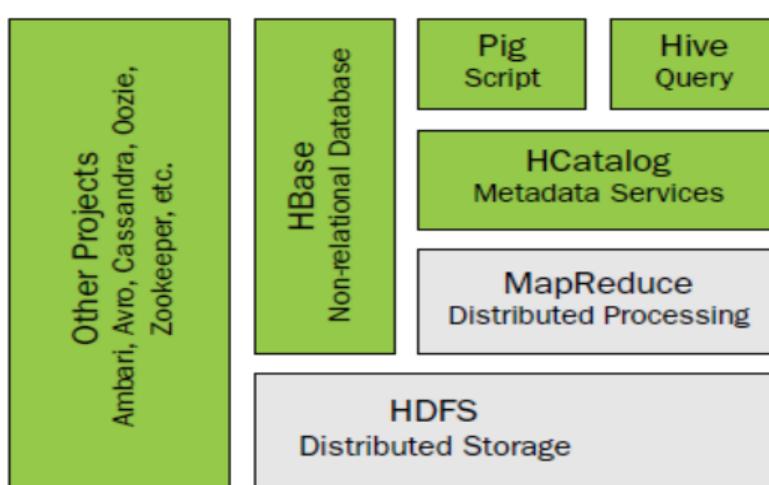


Figure 3.1 The Hadoop 1.0 ecosystem. MapReduce and HDFS are the core components, while other components are built around the core.

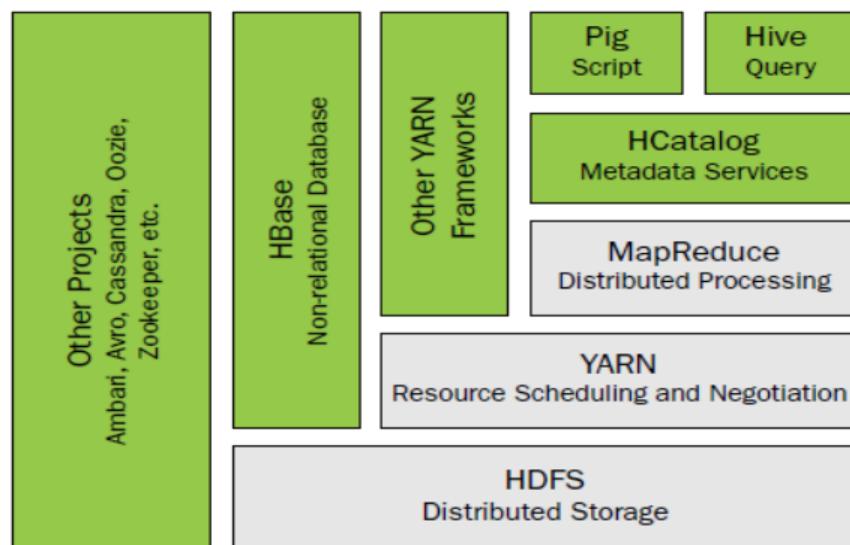
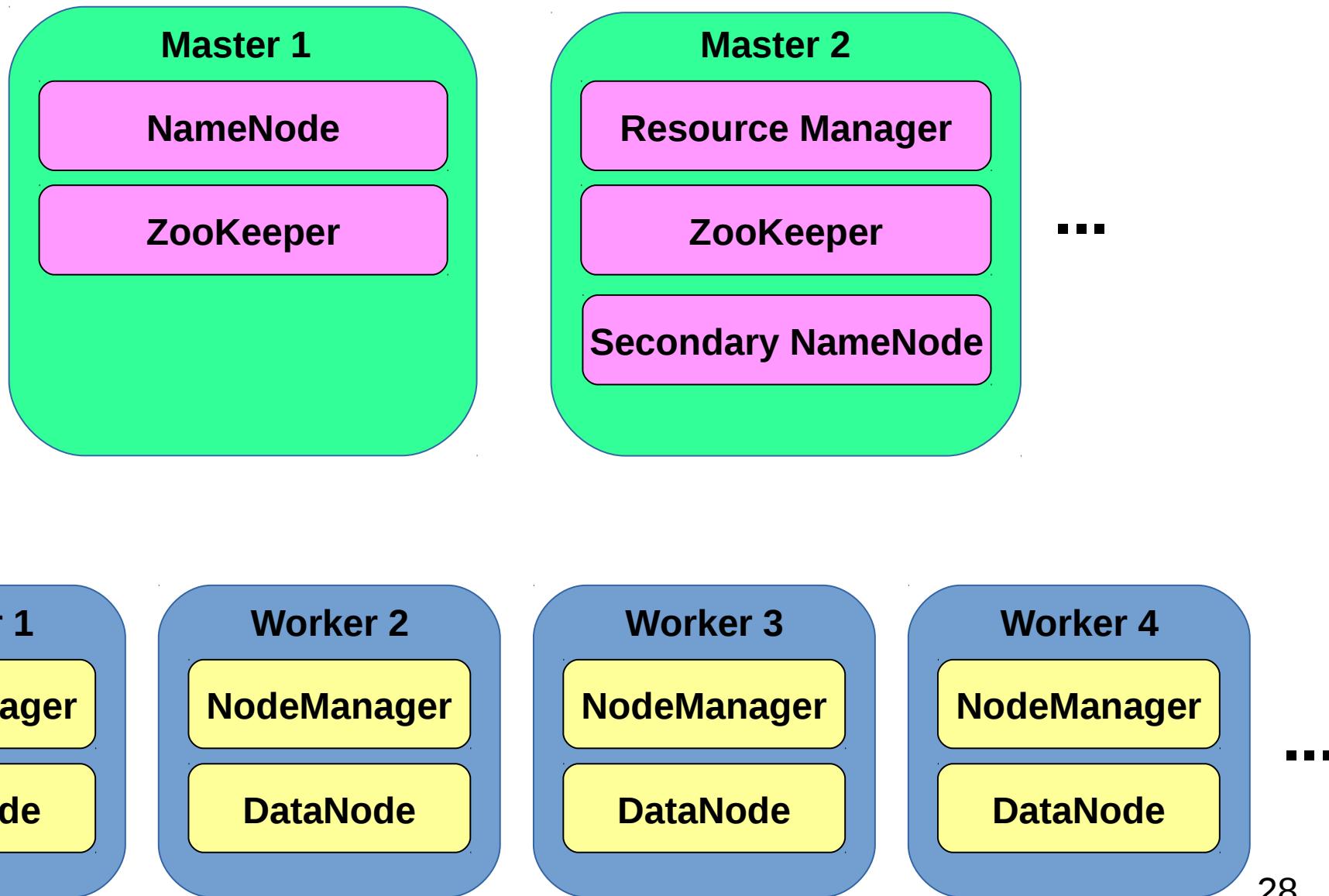


Figure 3.2 YARN adds a more general interface to run non-MapReduce jobs within the Hadoop framework

From “Apache Hadoop™ YARN Moving beyond MapReduce and Batch Processing with Apache Hadoop™ 2”, by Arun C. Murthy, Vinod Kumar Vavilapalli, Doug Eadline, Joseph Niemiec, Jeff Markham

Hadoop Cluster Architecture



HDFS Architecture

HDFS stands for **Hadoop Distributed File System**

It is a File System that lives across the nodes of a cluster. It stores files, each file has a filename and is located in a specific directory.

It supports most of the operations supported by an ordinary File System.

Every HDFS cluster is comprised of:

- one or two **NameNodes** and
- many **DataNodes**

HDFS Architecture

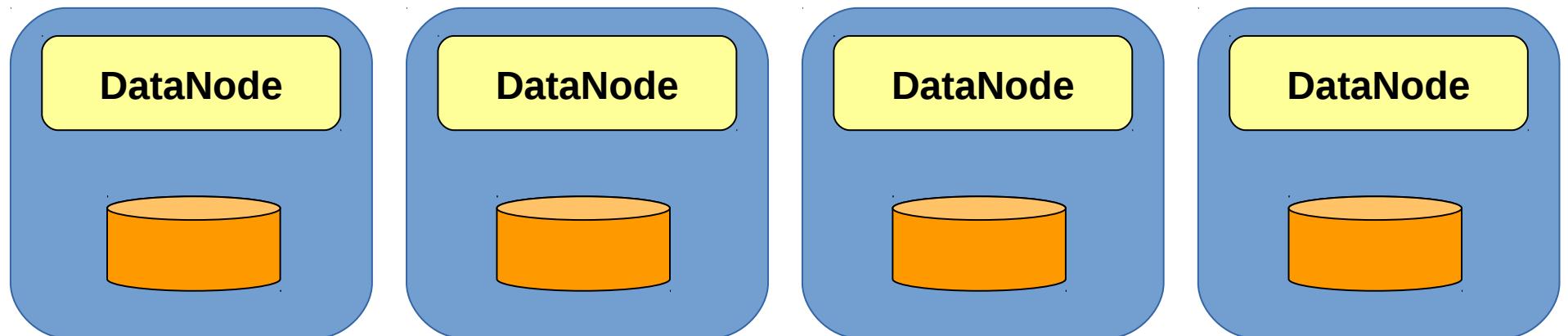
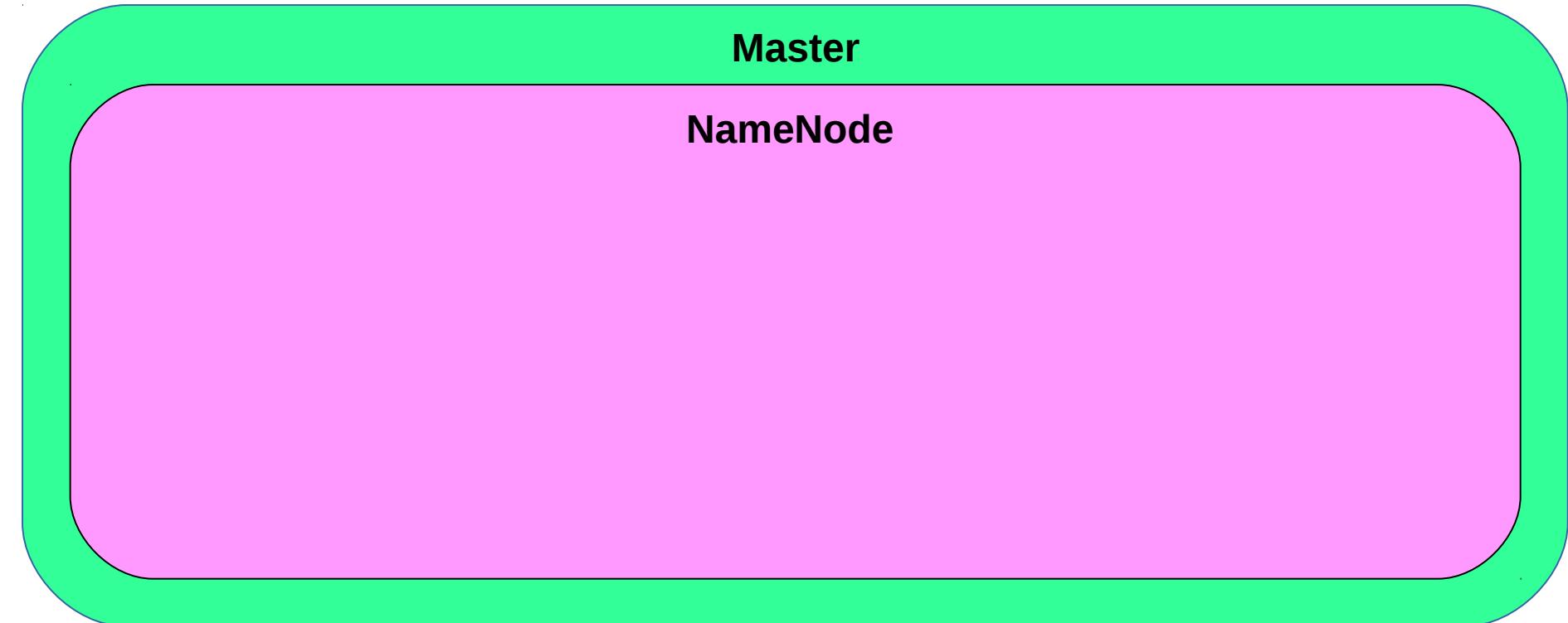
NameNode (one or two per cluster)

- Represents a single filesystem namespace rooted at /
- Is the master service of HDFS
- Determines and maintains how the chunks of data are distributed across the DataNodes
- Actual data never resides here, only metadata (e.g., maps of where blocks are distributed).

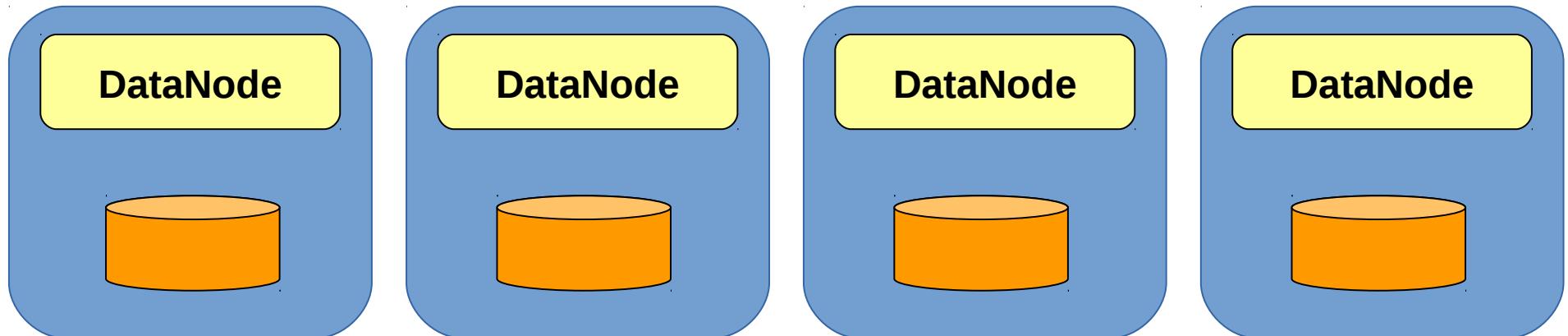
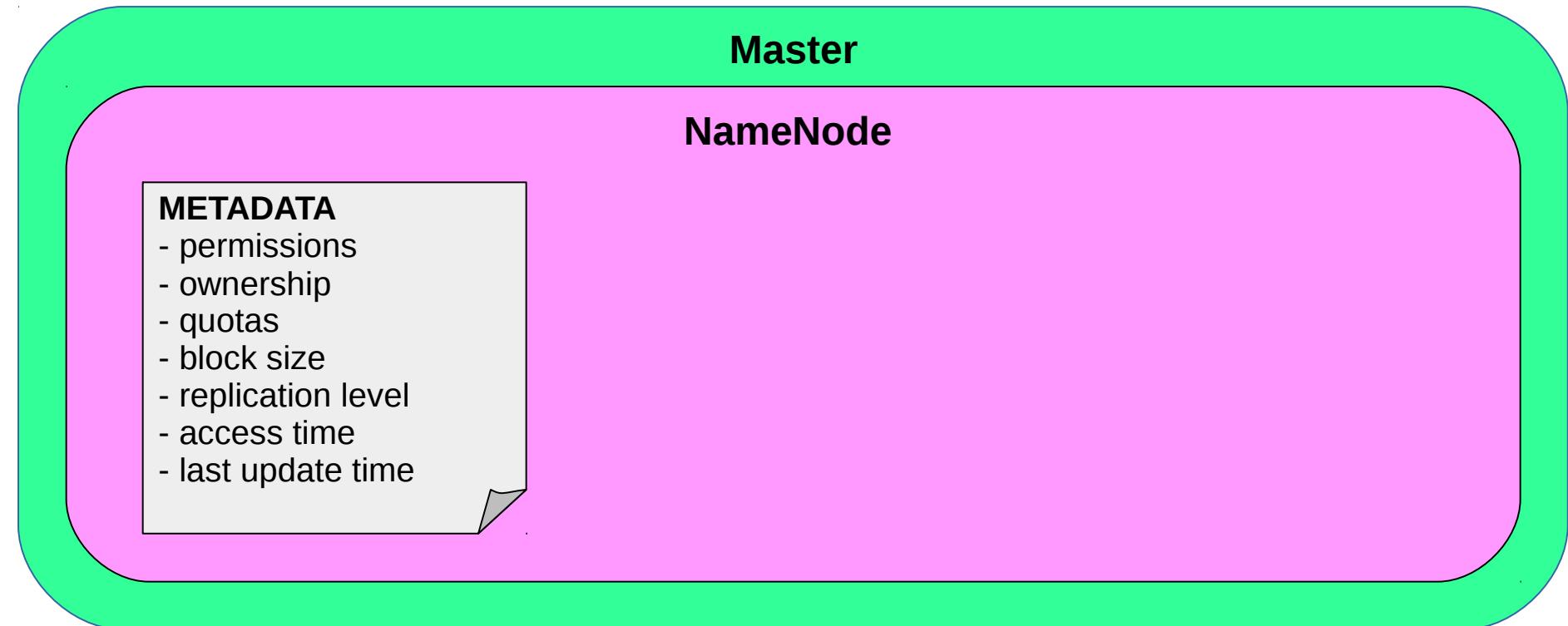
DataNode (as many as you want per cluster)

- Stores the chunks of data, and is responsible for **replicating** the chunks across other DataNodes
- Default number of replicas on most clusters is 3 (but it can be changed on a per-file basis)
- Default block size on most clusters is 128MB.

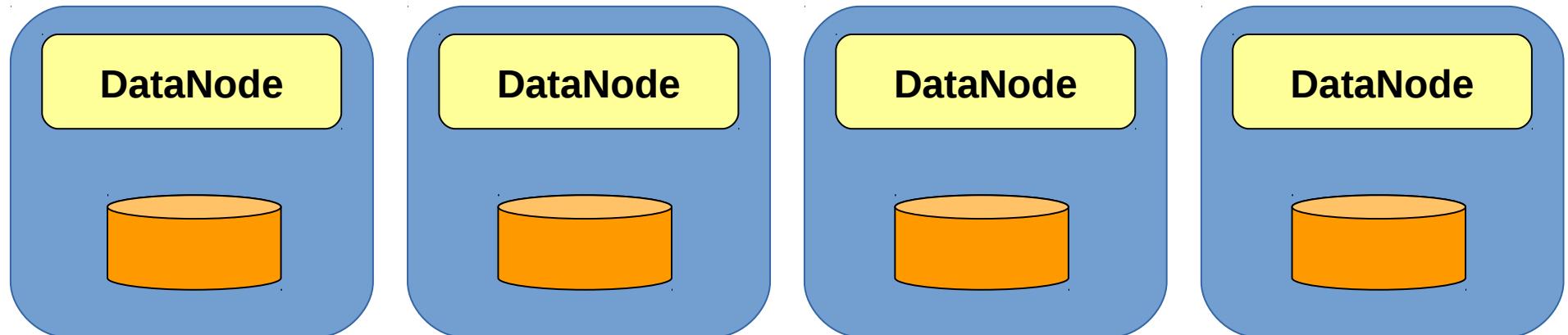
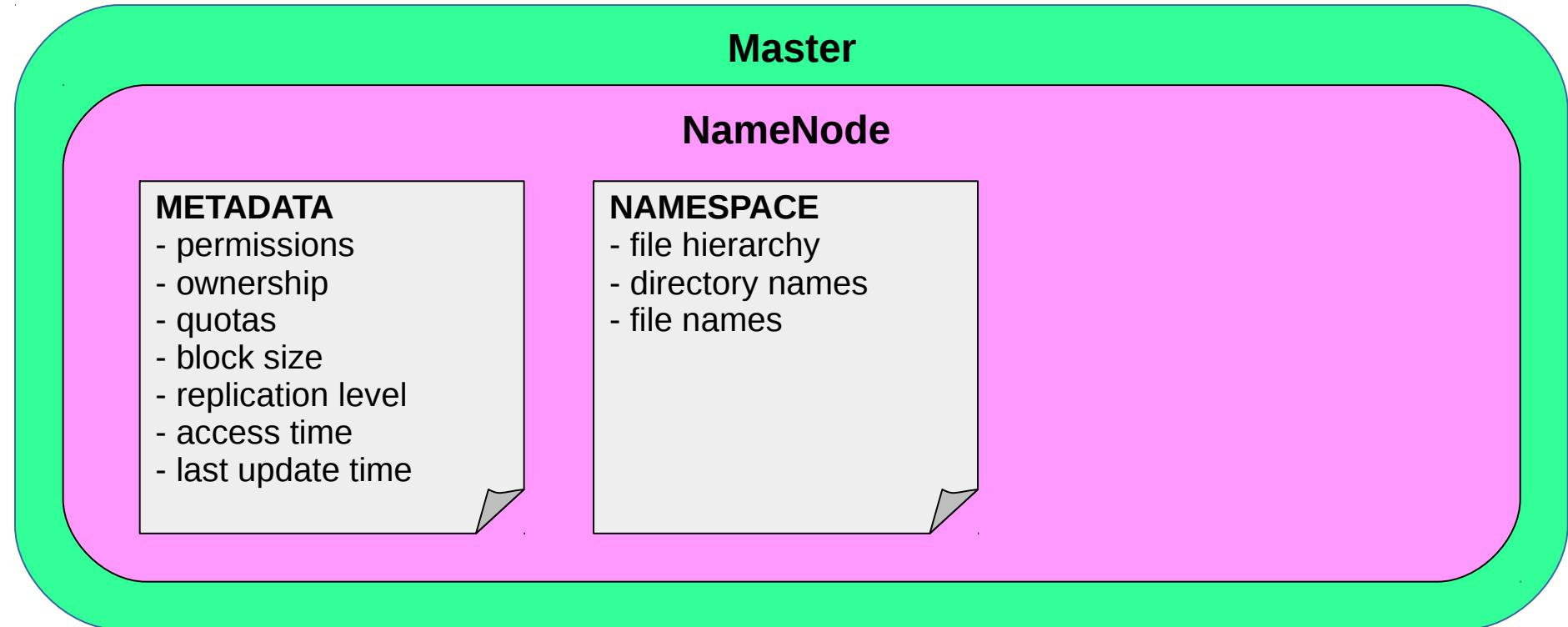
HDFS Architecture: NameNode



HDFS Architecture: NameNode



HDFS Architecture: NameNode



HDFS Architecture: NameNode

Master

NameNode

METADATA

- permissions
- ownership
- quotas
- block size
- replication level
- access time
- last update time

NAMESPACE

- file hierarchy
- directory names
- file names

BLOCK MAPPING

For every file name it keeps track of the block identifiers (IDs)

DataNode



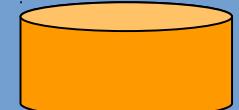
DataNode



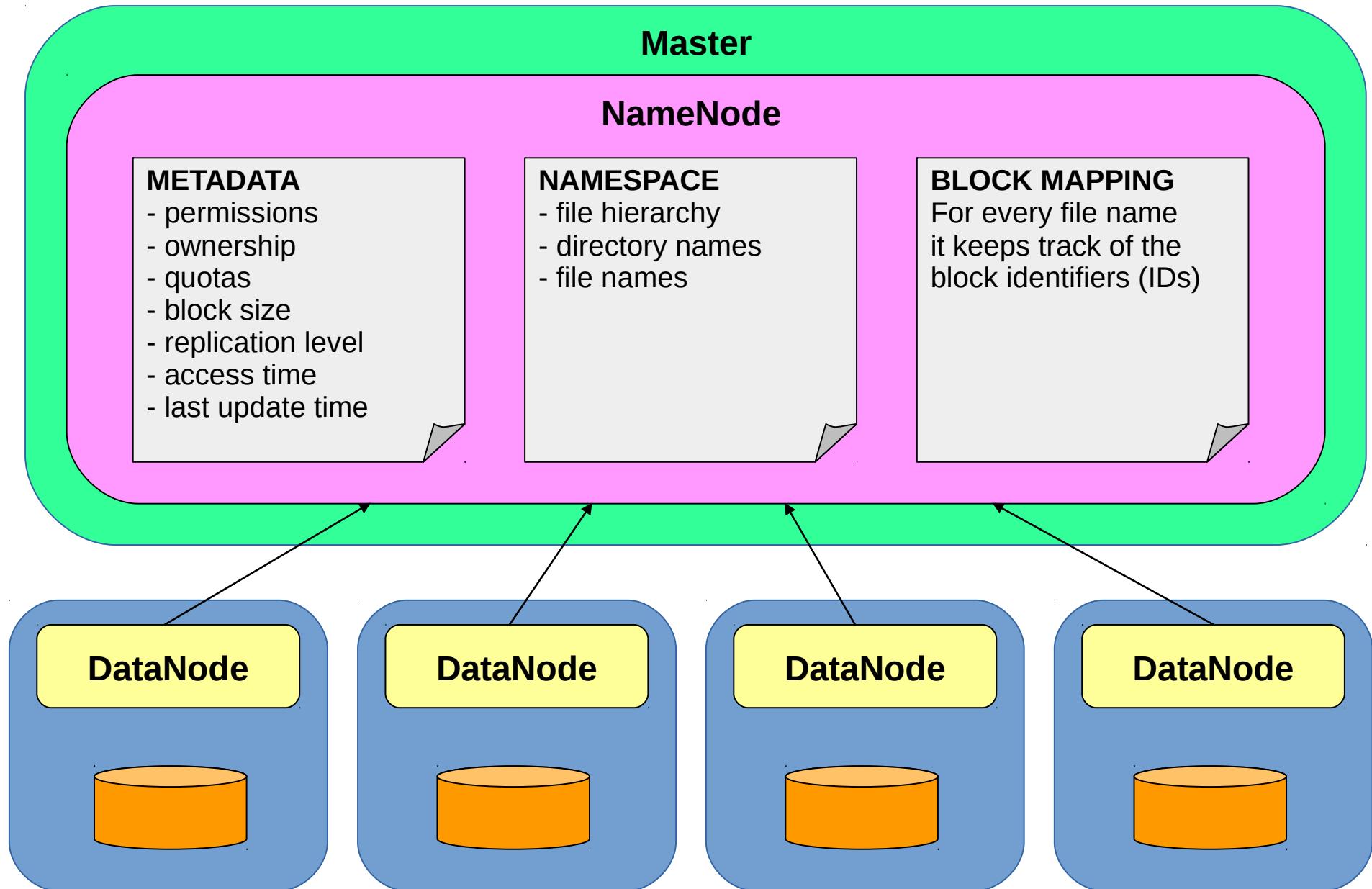
DataNode



DataNode

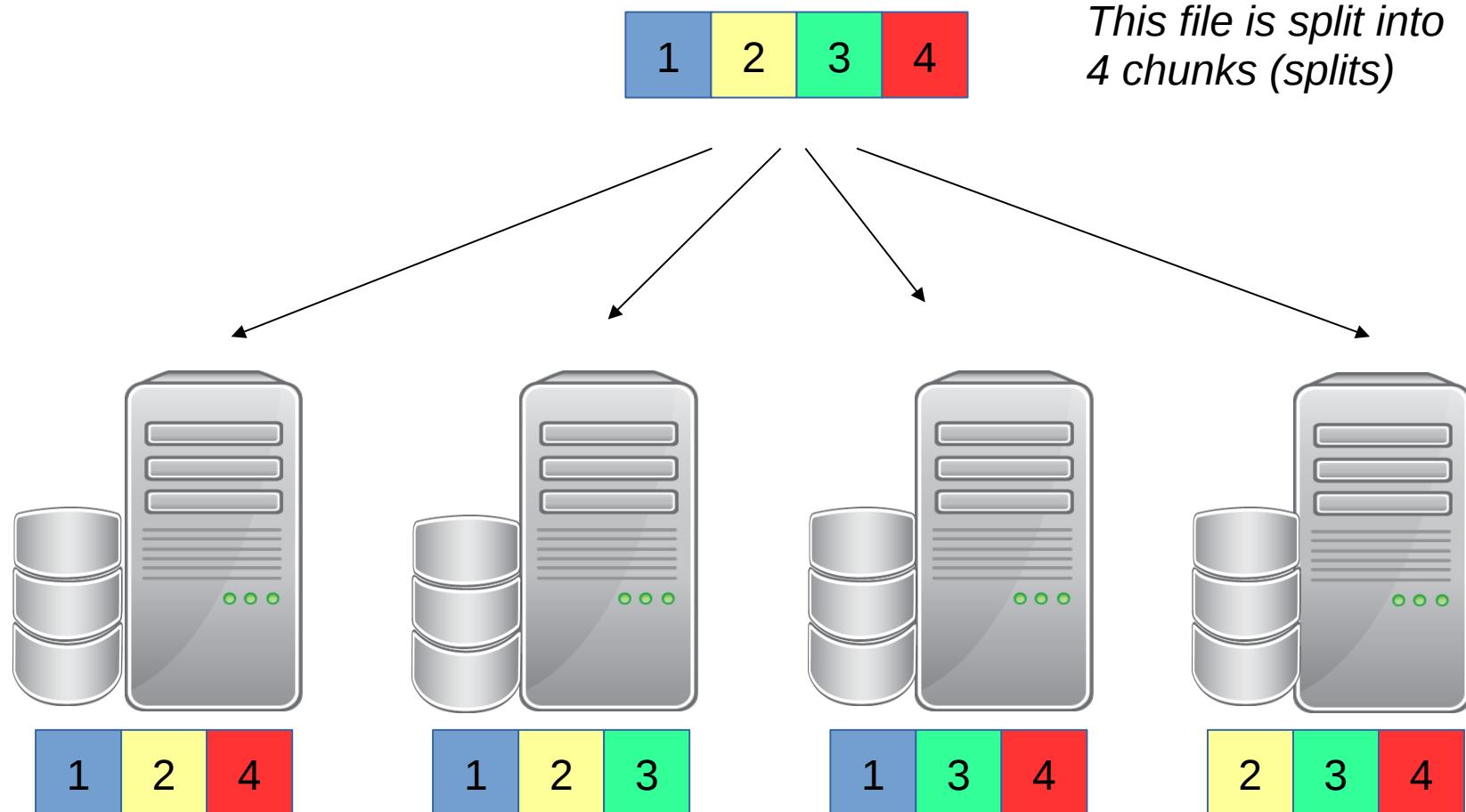


HDFS Architecture: NameNode



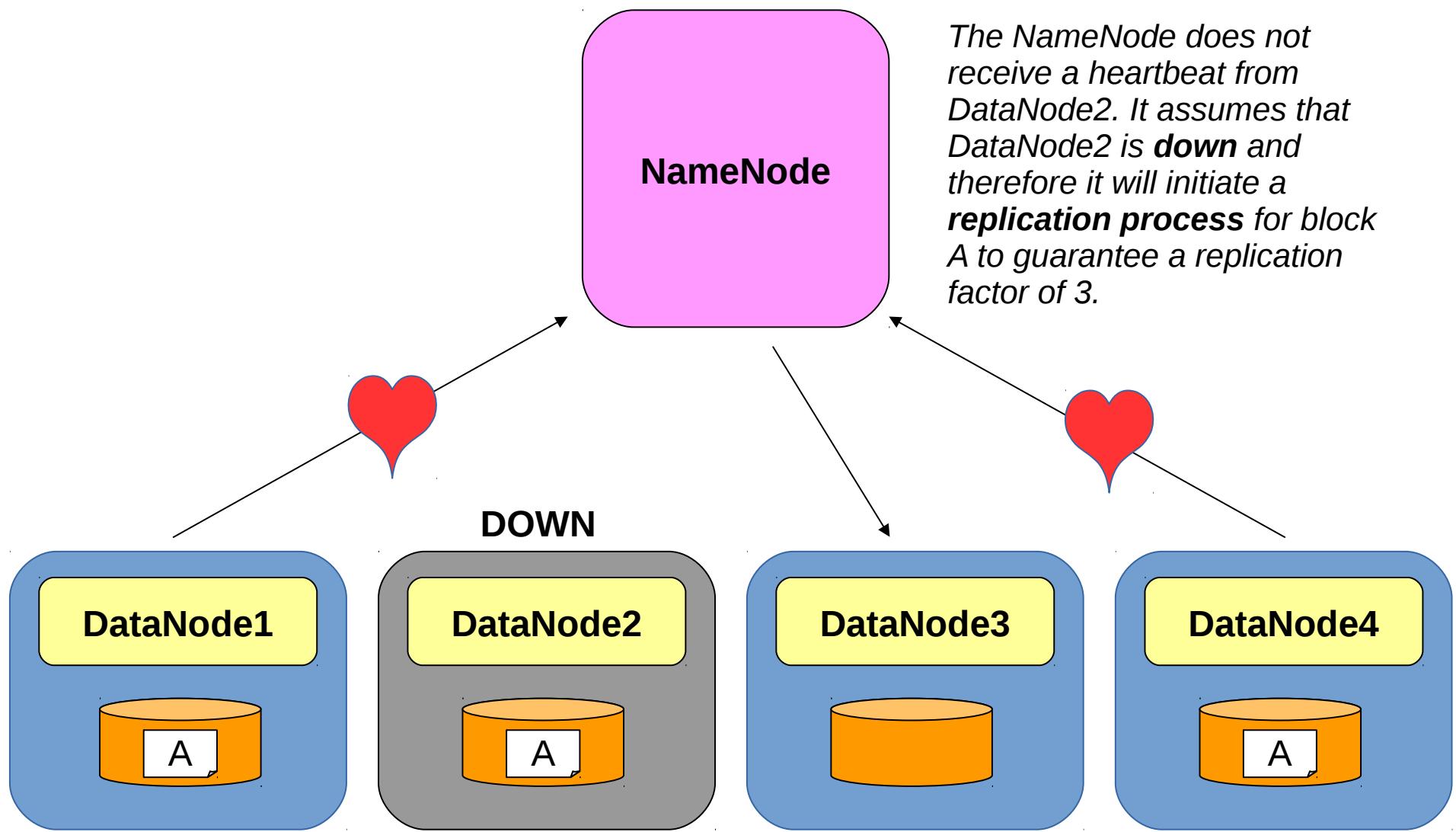
HDFS Architecture: replication

Replication in HDFS

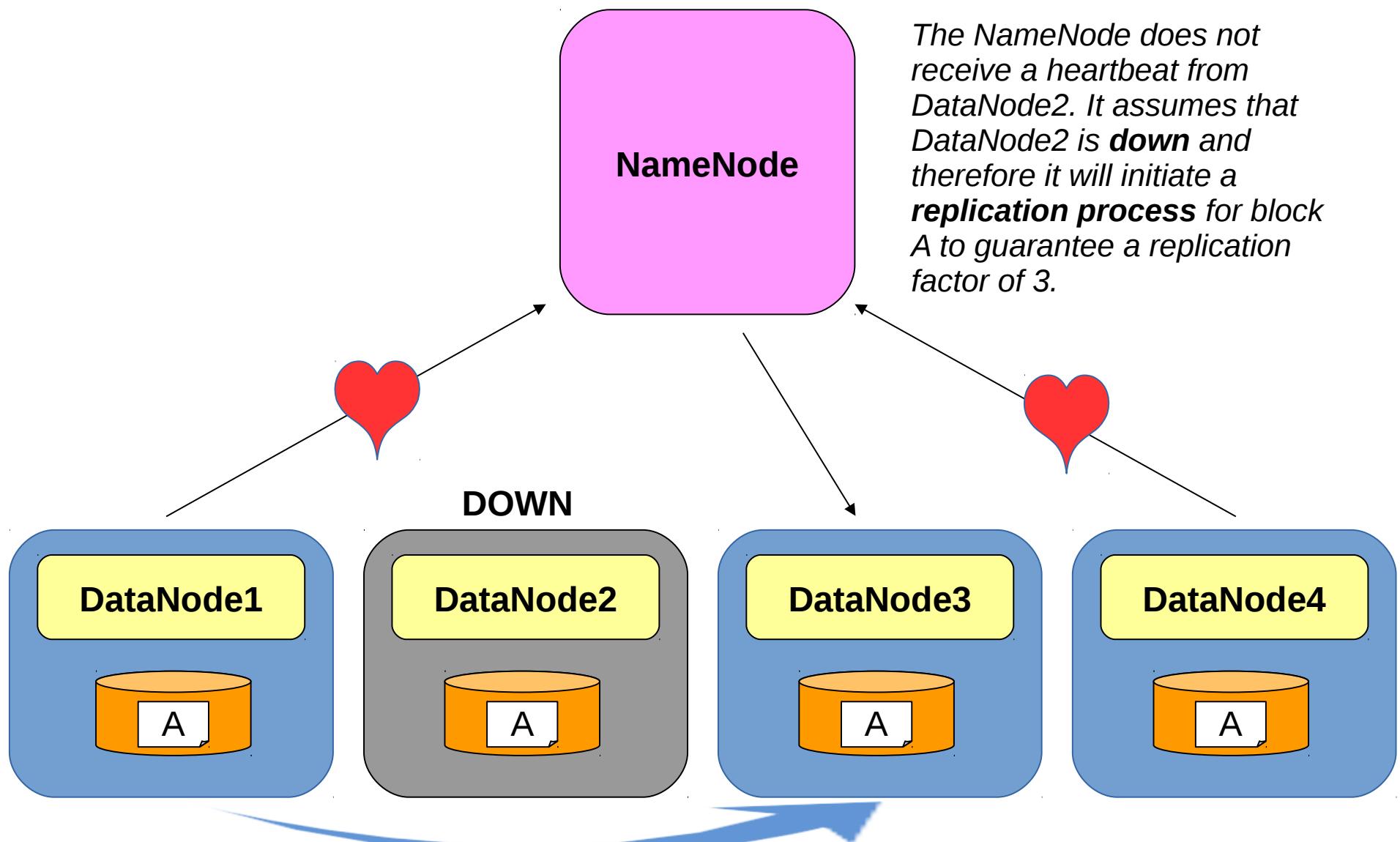


The file is split in chunks. Each is replicated three times in this example. One of the chunk is located at a different rack for increased fault tolerance.

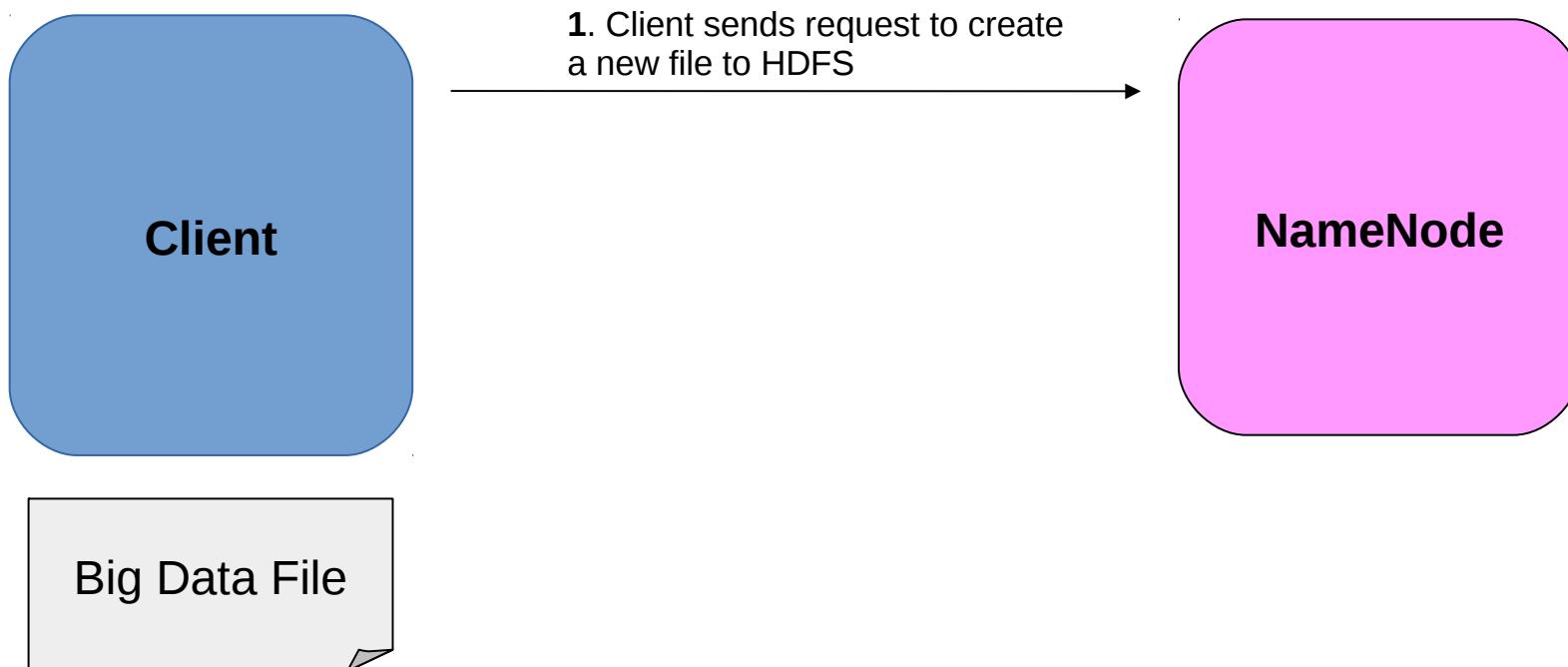
HDFS Architecture: replication



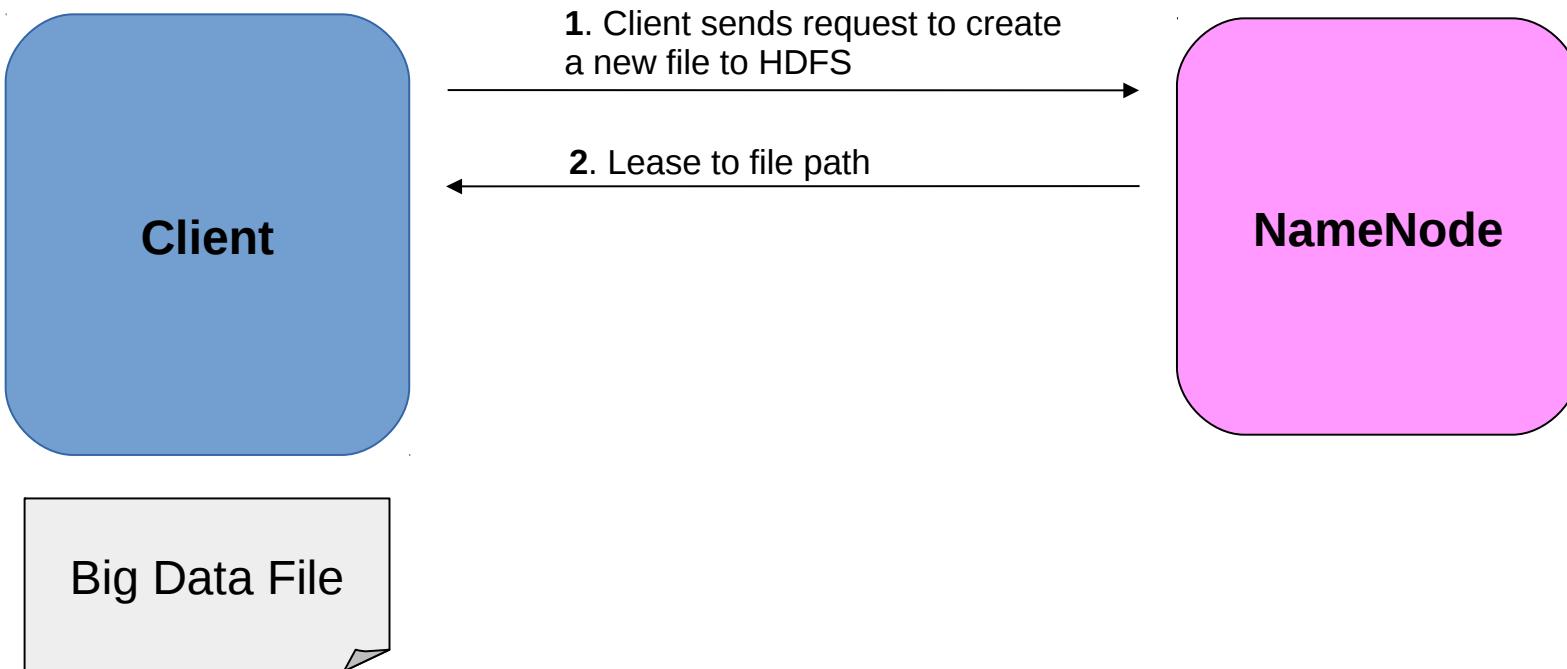
HDFS Architecture: replication



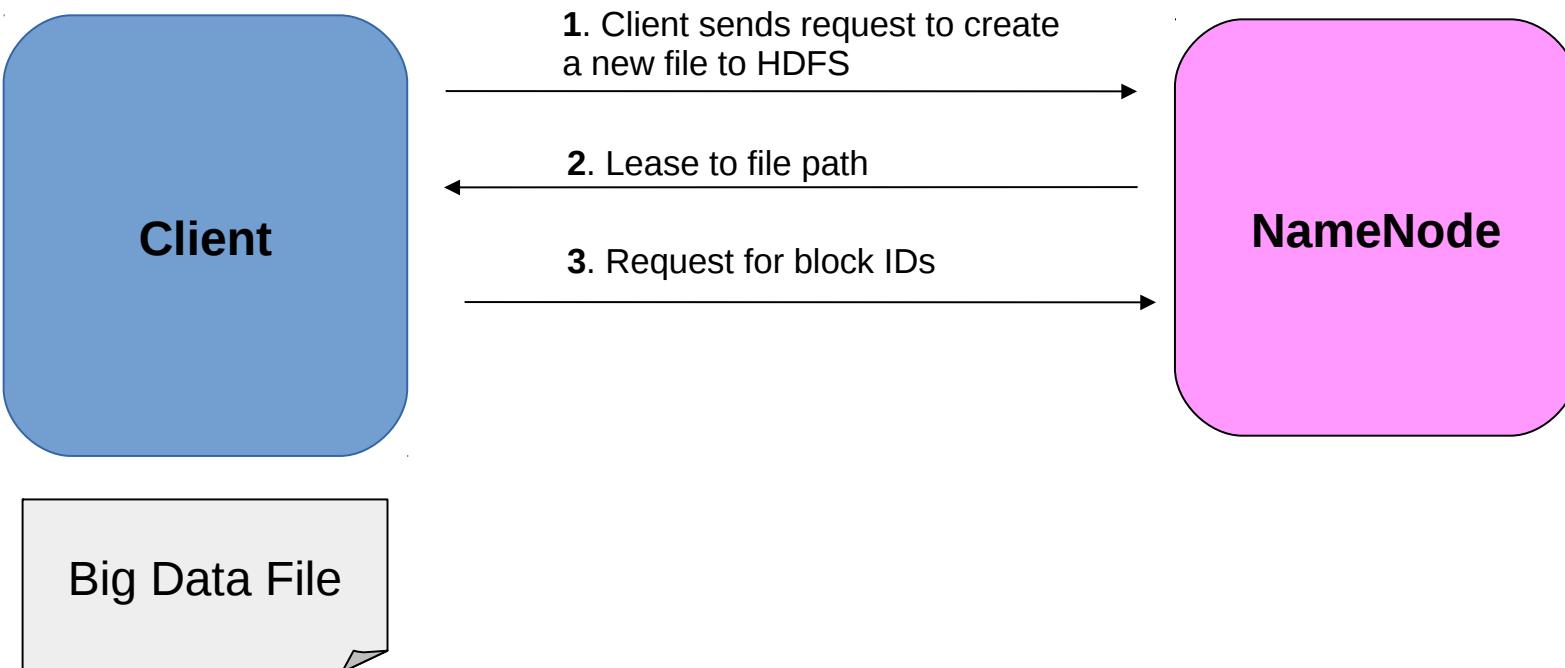
HDFS Architecture: writing a file



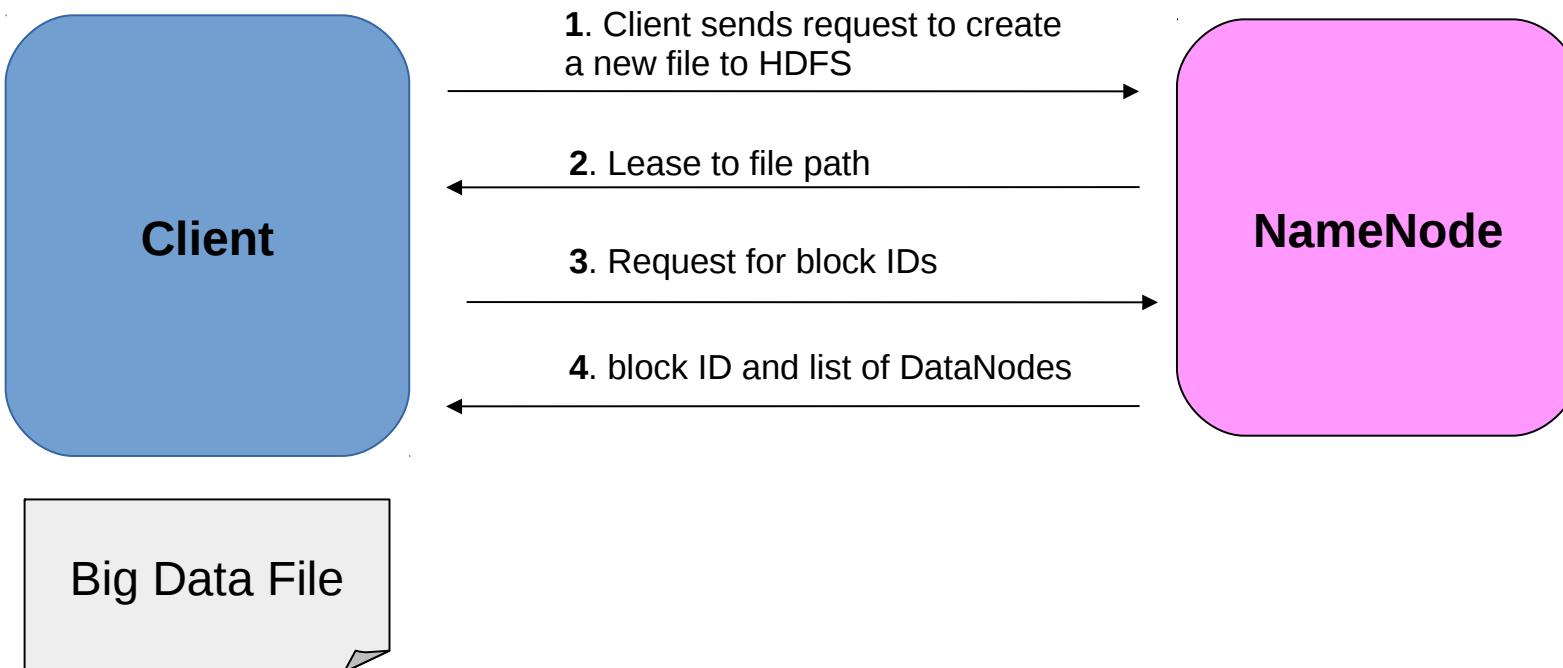
HDFS Architecture: writing a file



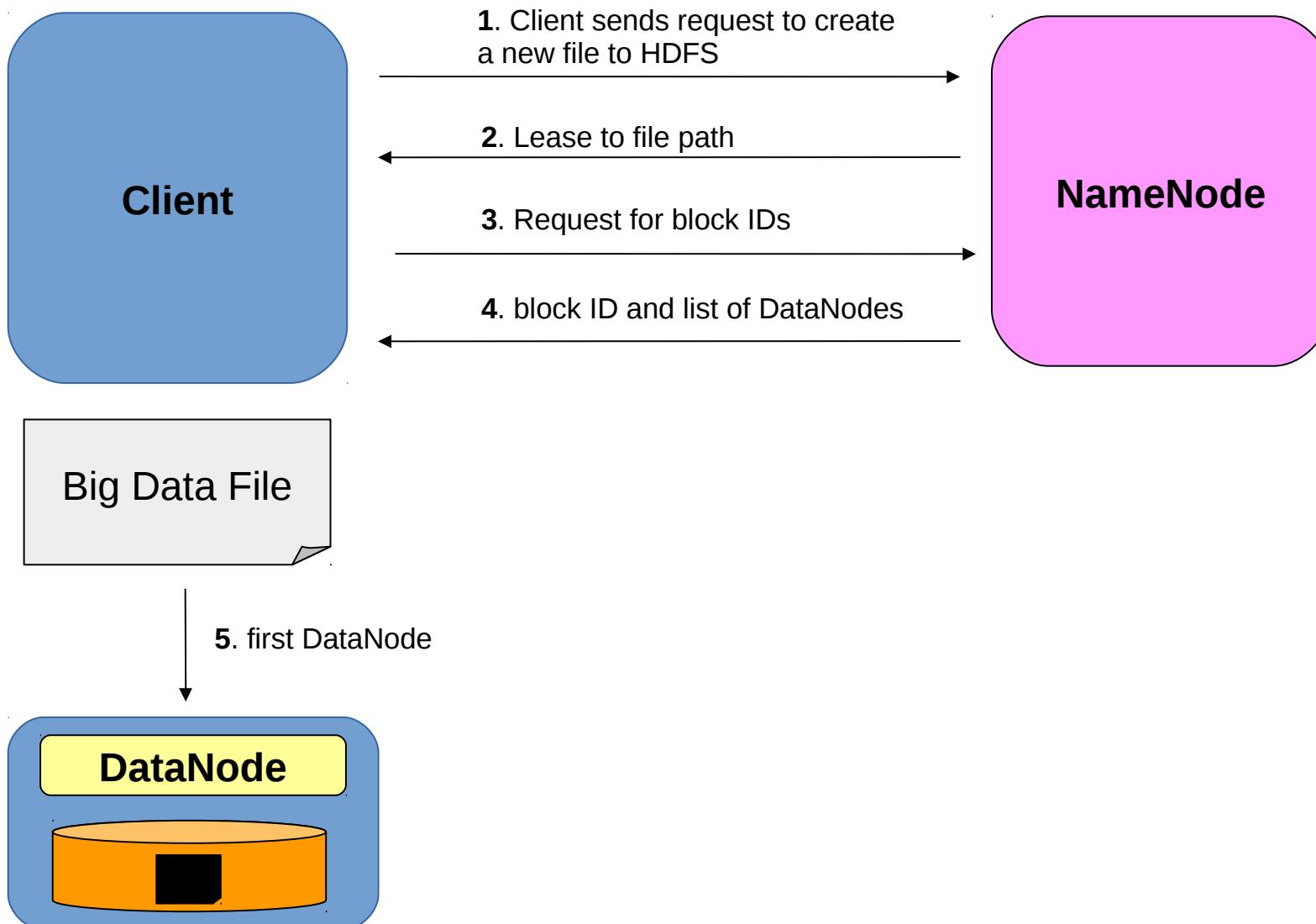
HDFS Architecture: writing a file



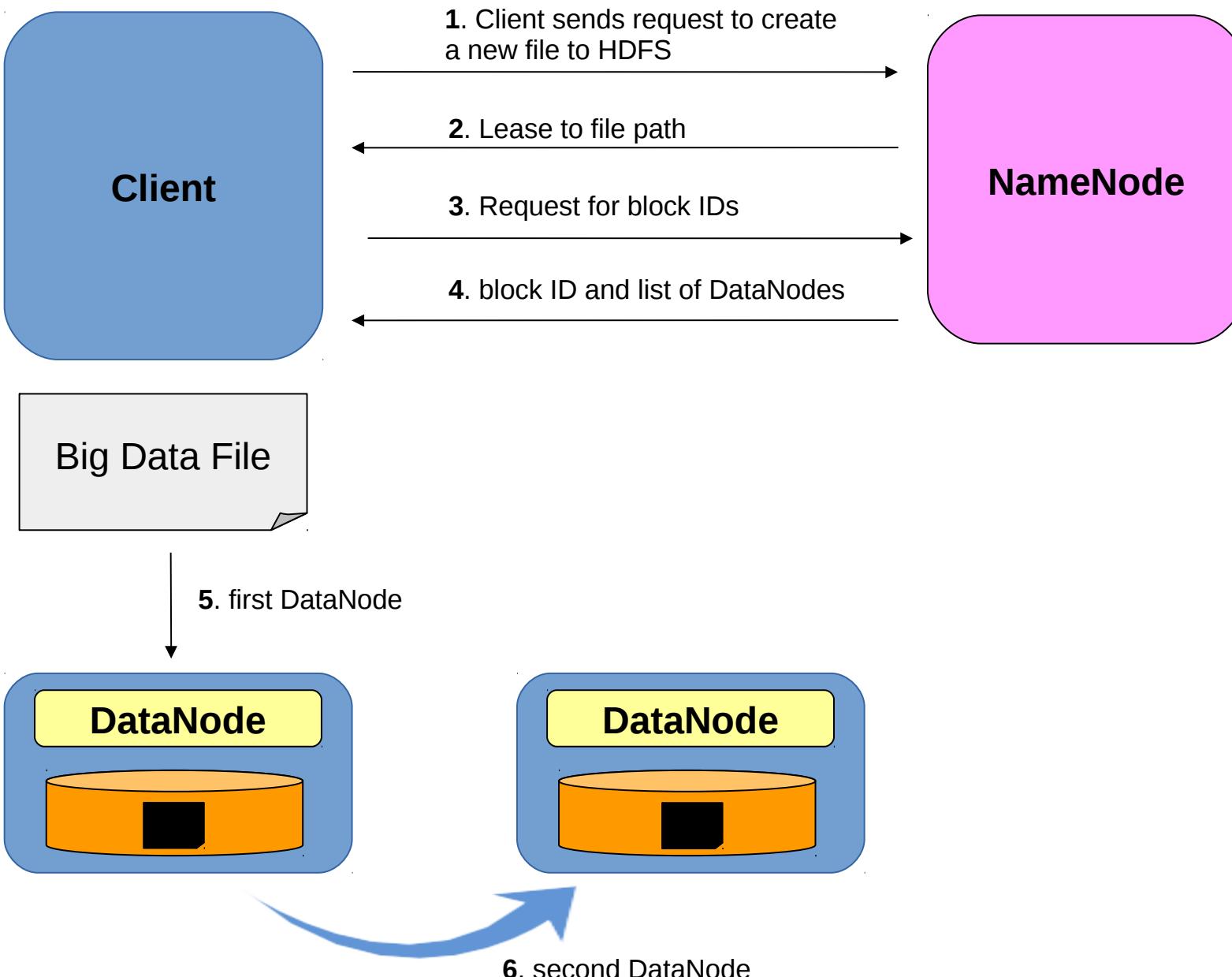
HDFS Architecture: writing a file



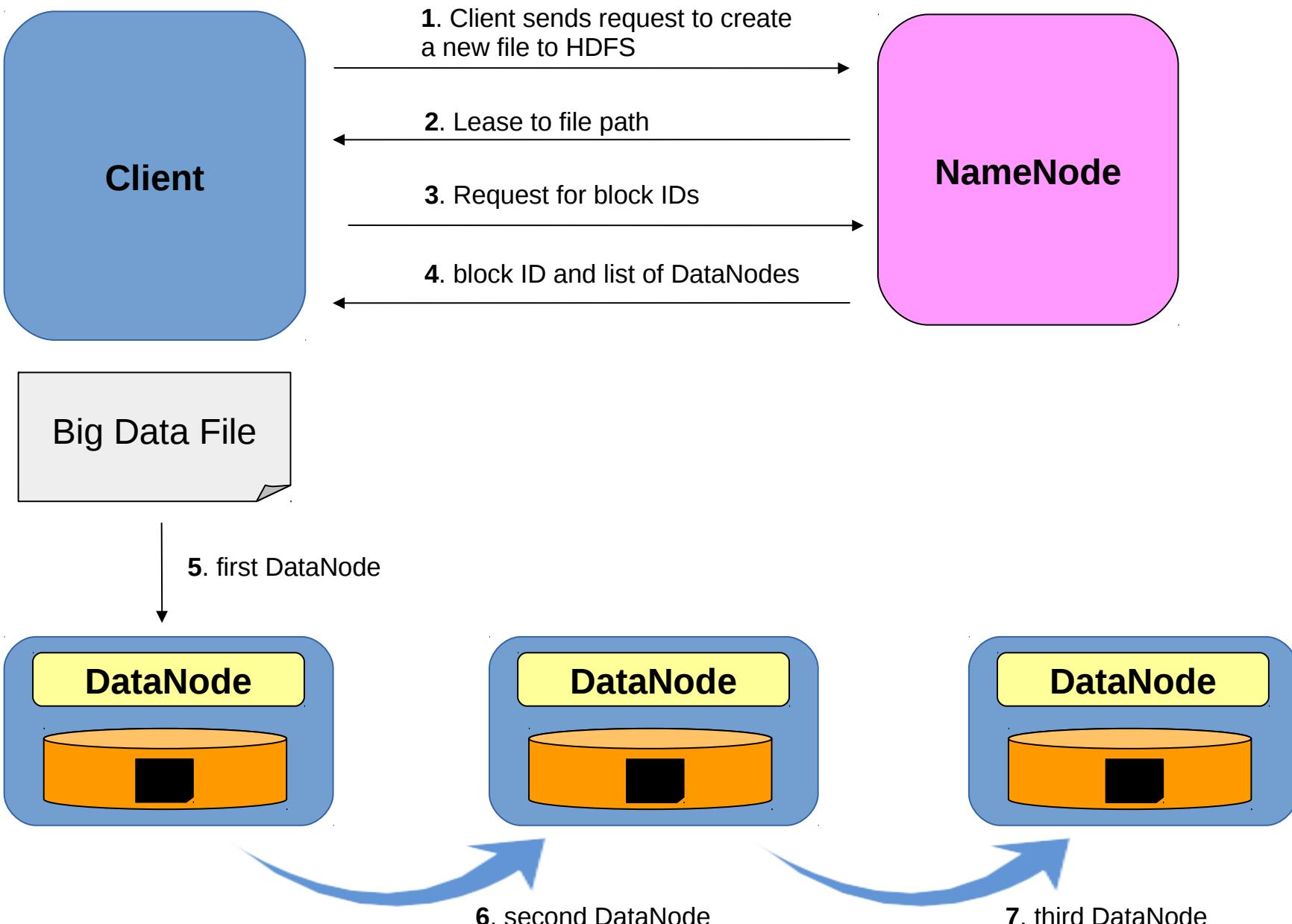
HDFS Architecture: writing a file



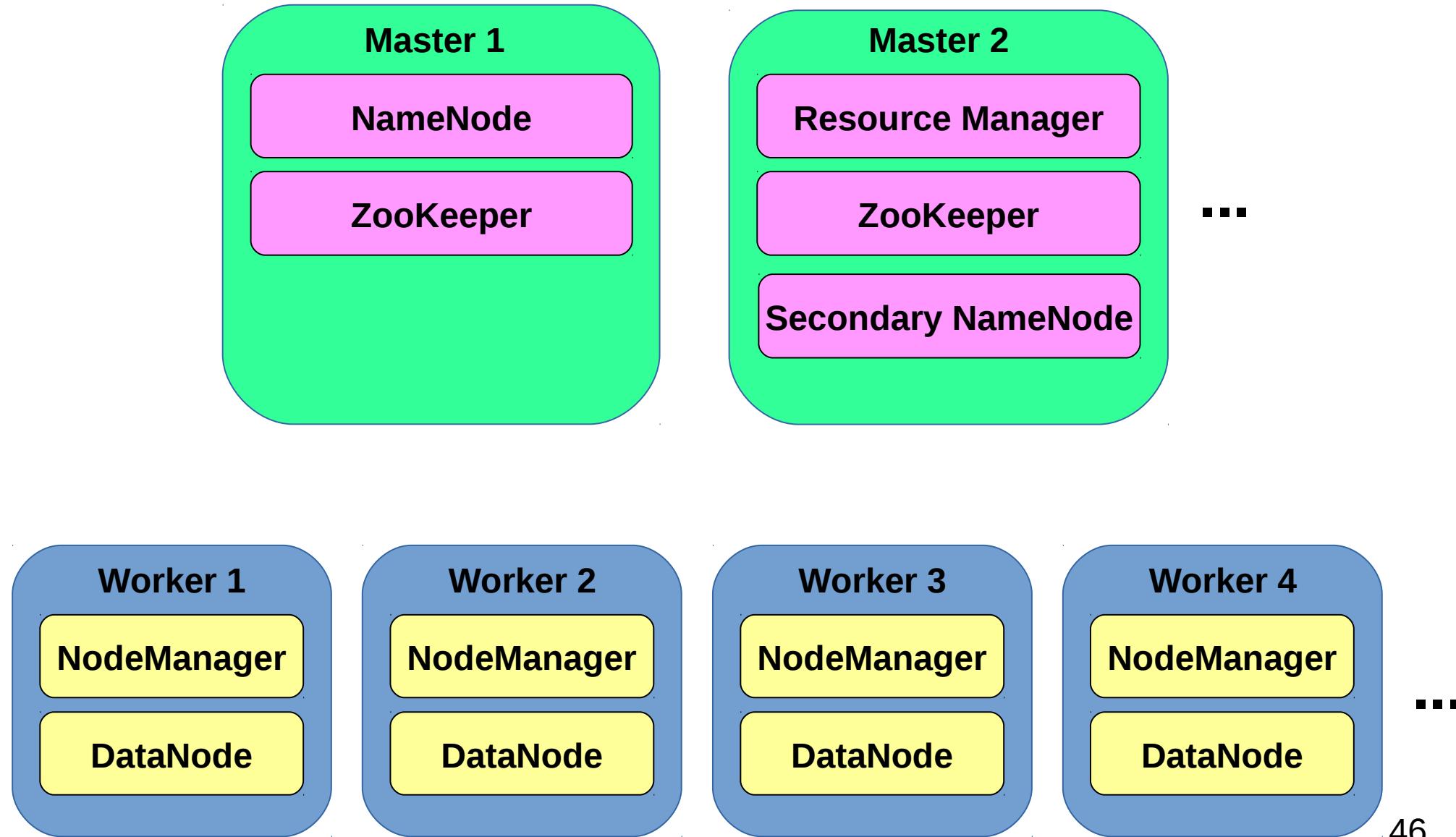
HDFS Architecture: writing a file



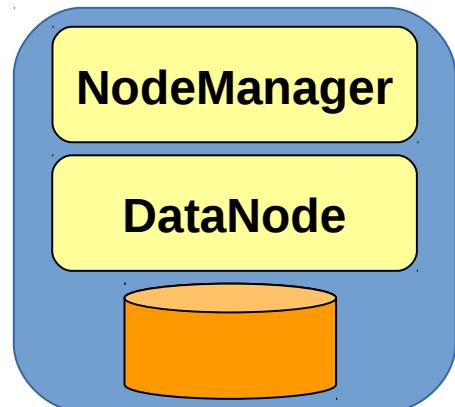
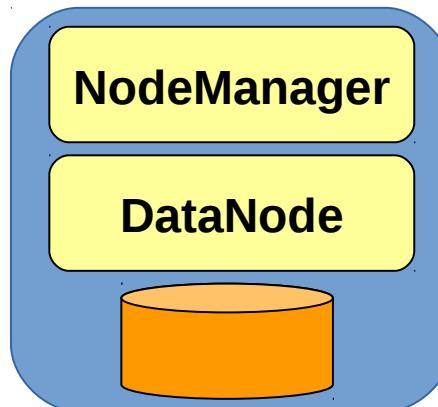
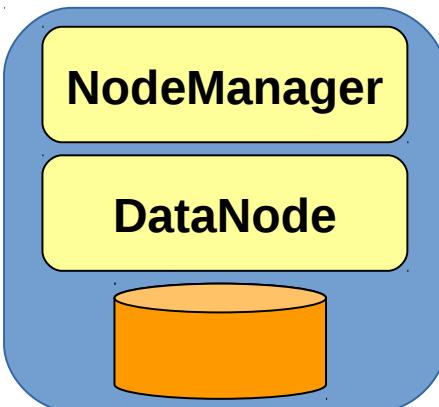
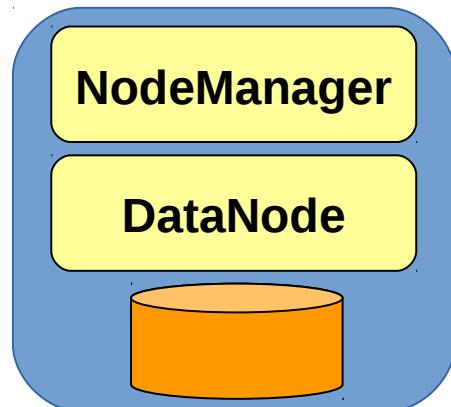
HDFS Architecture: writing a file



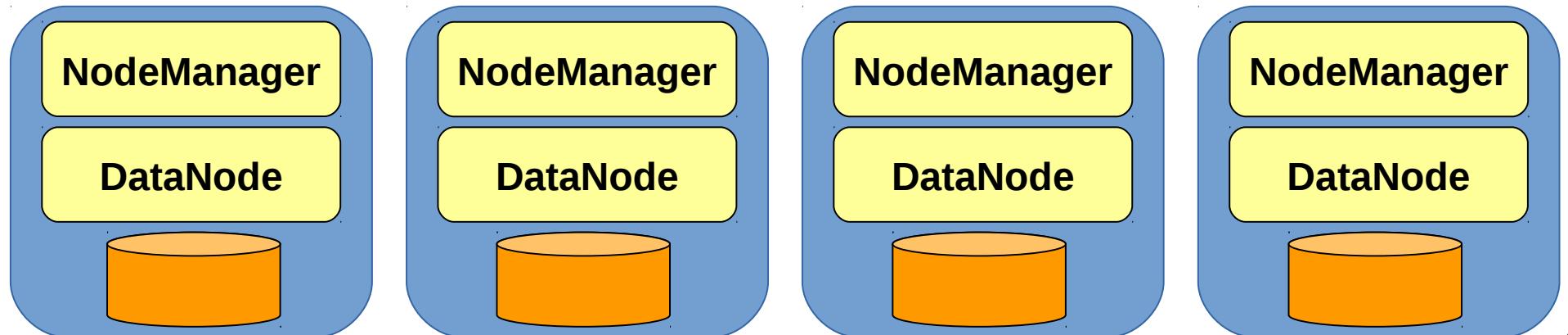
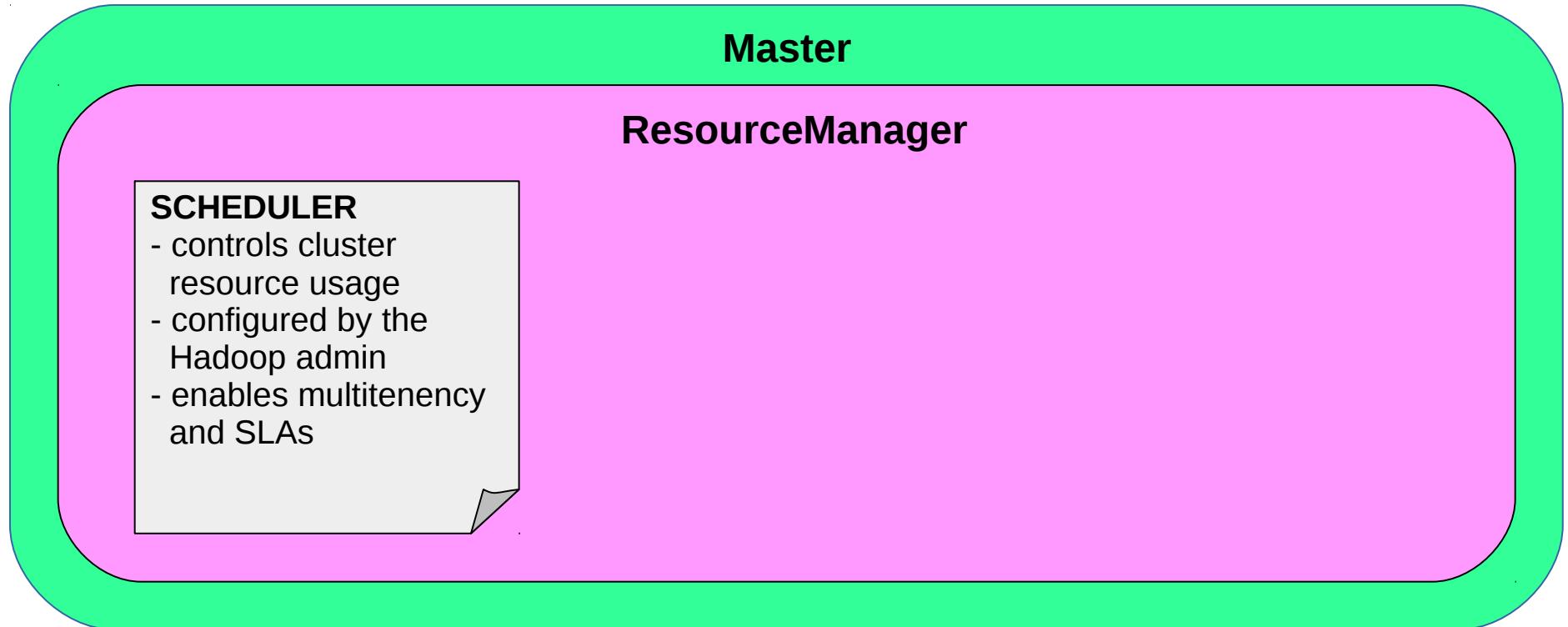
Hadoop Cluster Architecture



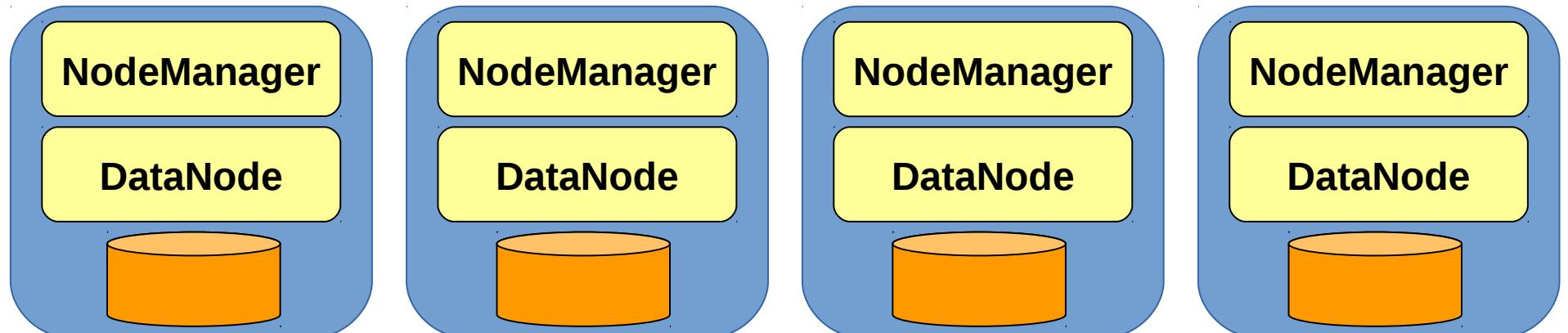
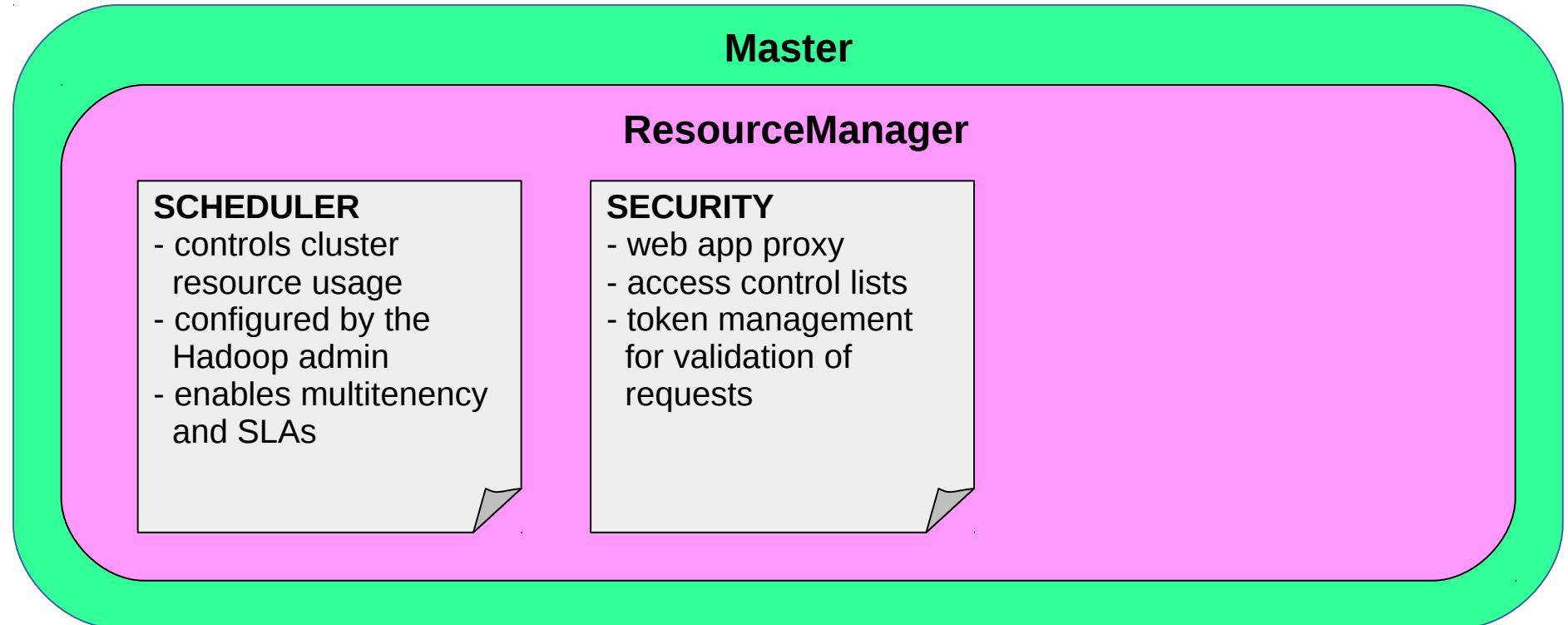
YARN Architecture: ResourceManager



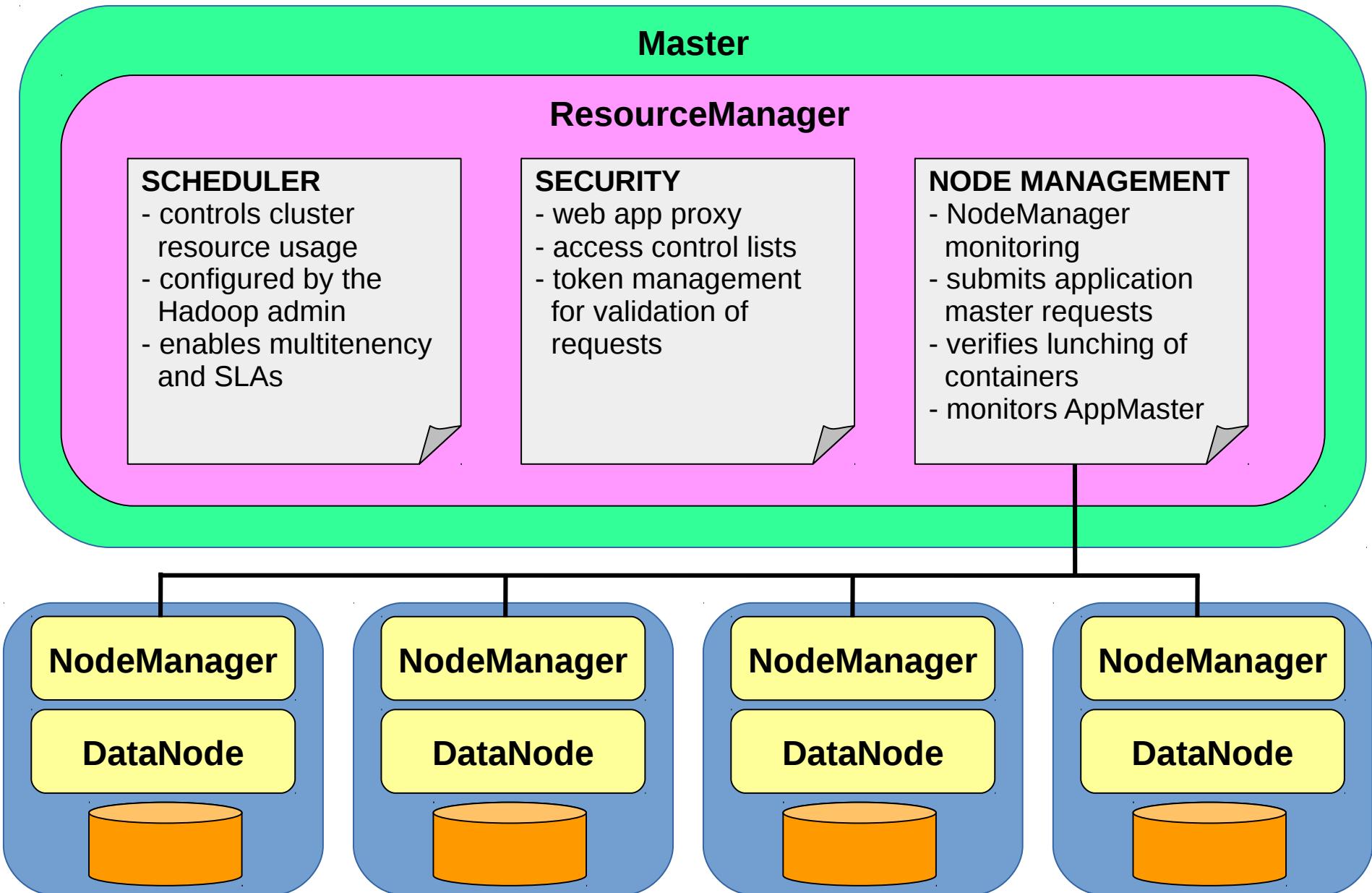
YARN Architecture: ResourceManager



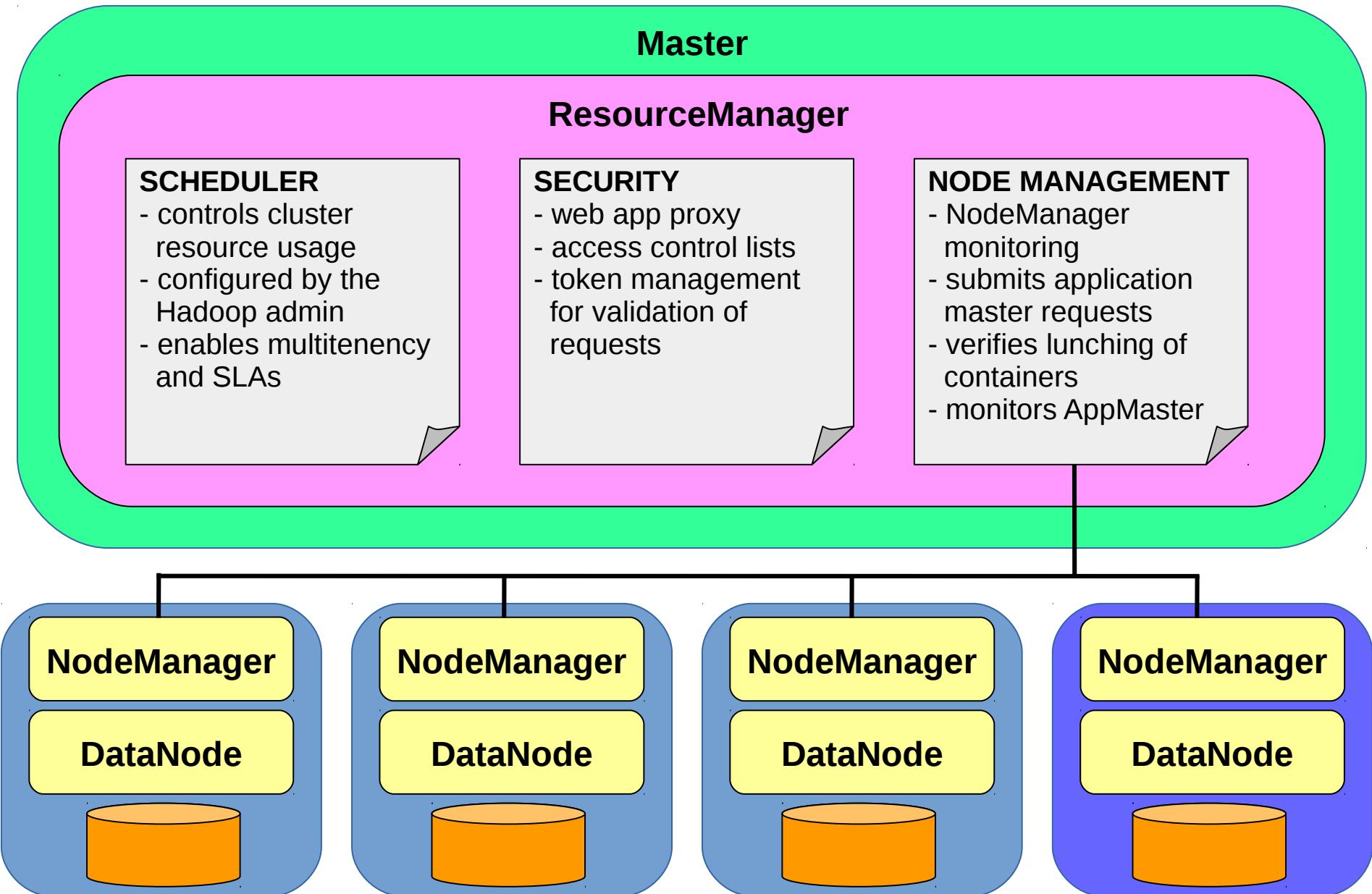
YARN Architecture: ResourceManager



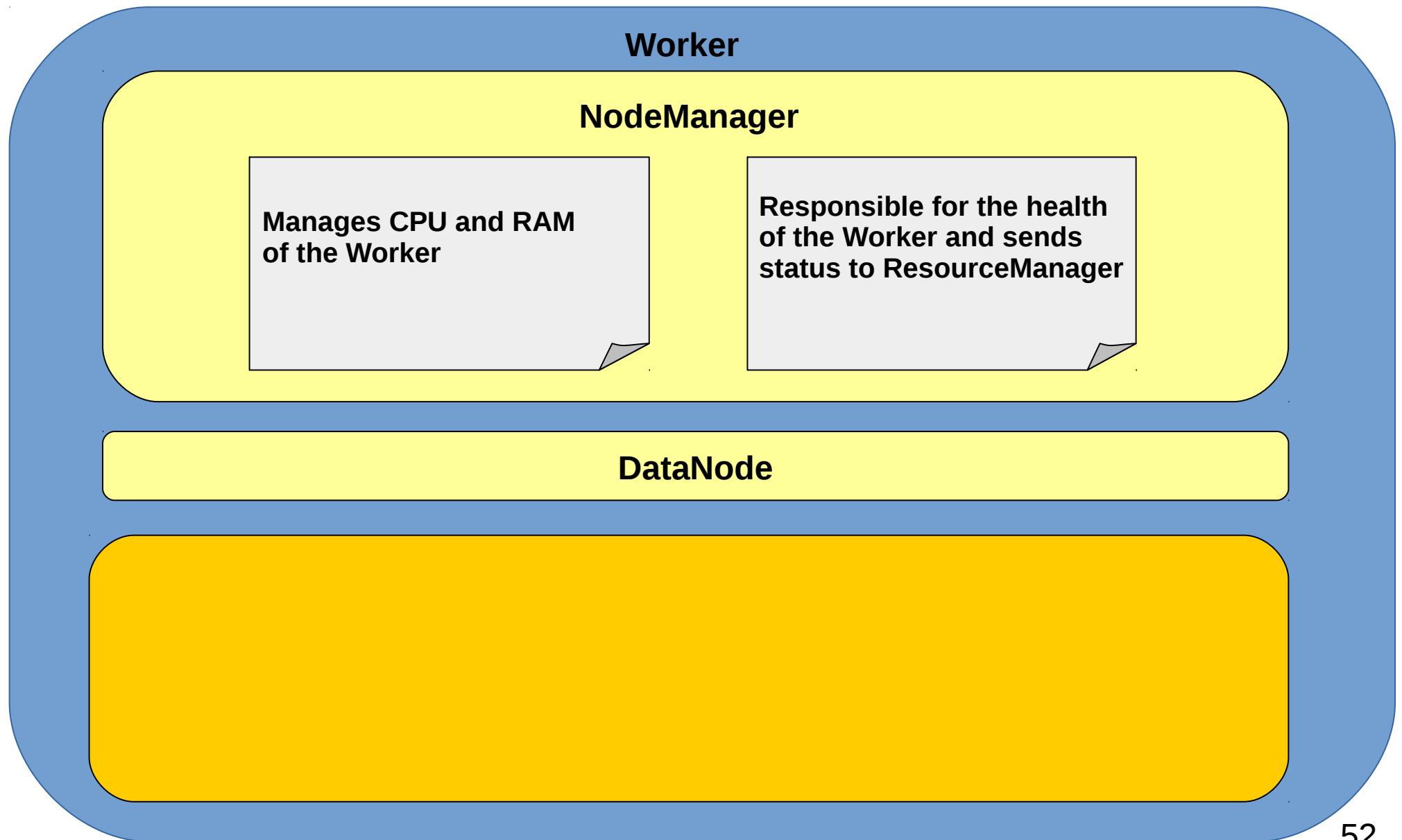
YARN Architecture: ResourceManager



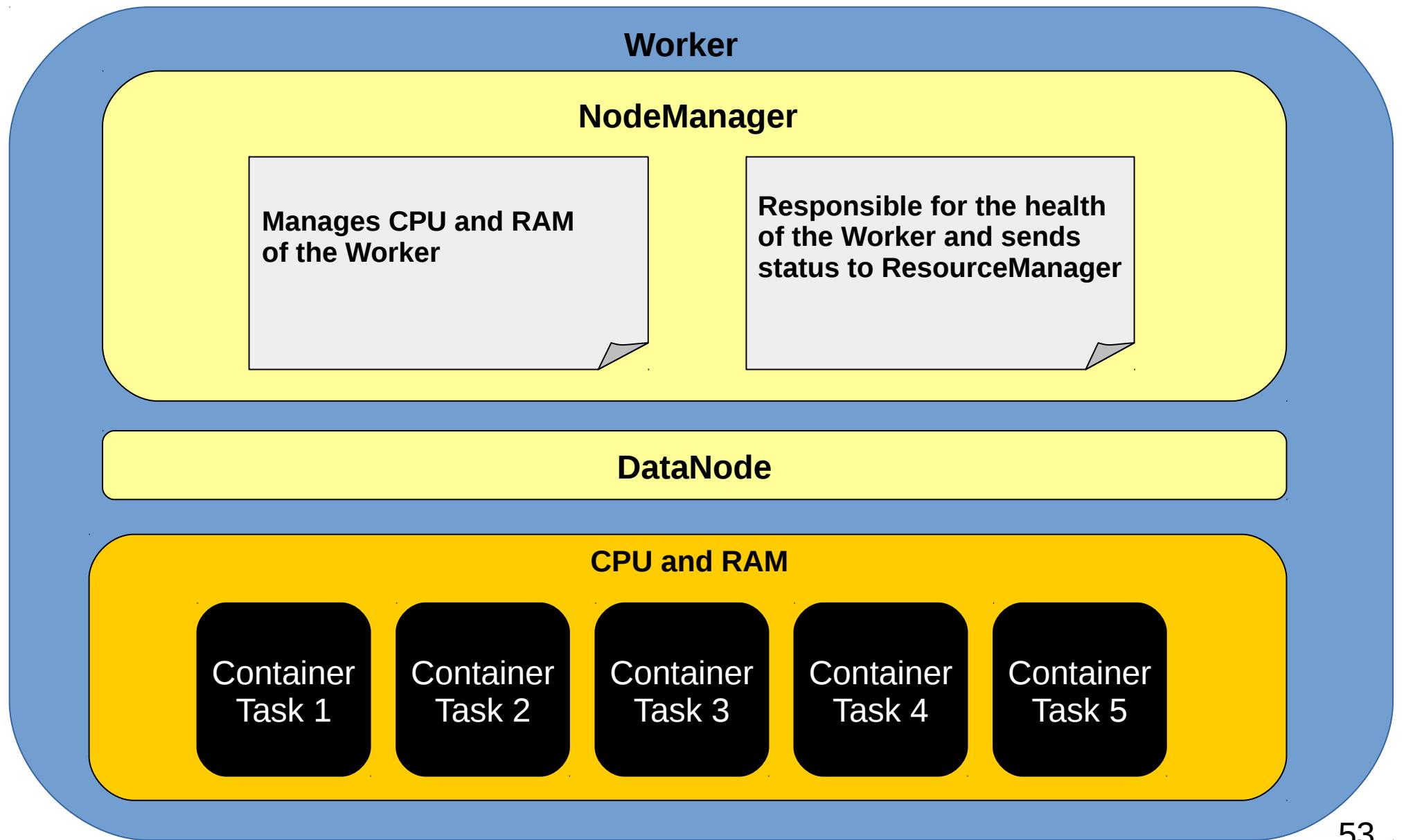
YARN Architecture: ResourceManager



YARN Architecture: NodeManager



YARN Architecture: NodeManager



Writing a MapReduce Program

Based on key-value pairs

Each job is composed of one or more MR stages

Each MR stage comprises:

- the **map** phase
- the **shuffle-and-sort** phase
- the **reduce** phase

The programmer **focuses on the problem**. Replication, fault tolerance, scheduling, re-scheduling and other low level procedures are handled by HDFS or YARN.

Complete MapReduce API

The programmer must implement the following functions:

map(): accepts a set of key-value pairs and generates another list of key-value pairs.

combine(): performs an aggregation before sending the data to reducers (reduces network traffic).

partition(): uses a hash function to distribute data to reducers (load balancing, avoids hotspots).

reduce(): accepts a key and a list of values for this specific key and performs an aggregation.

Note: combine() and partition() are optional

Simple Example: WordCount

Given a potentially massive txt file, compute the number of occurrences of every word. For every word, output a pair
(word, #occurrences)

The number of pairs in the output equals the number of unique words in the file.

E.g.,

input: **can you** see the real me? **can you?** **can you?**

output: **(can,3), (you,3), (see,1), (the,1), (real,1), (me,1)**

WordCount Example

INPUT
(from HDFS)

hello there

hello again

hello to the world

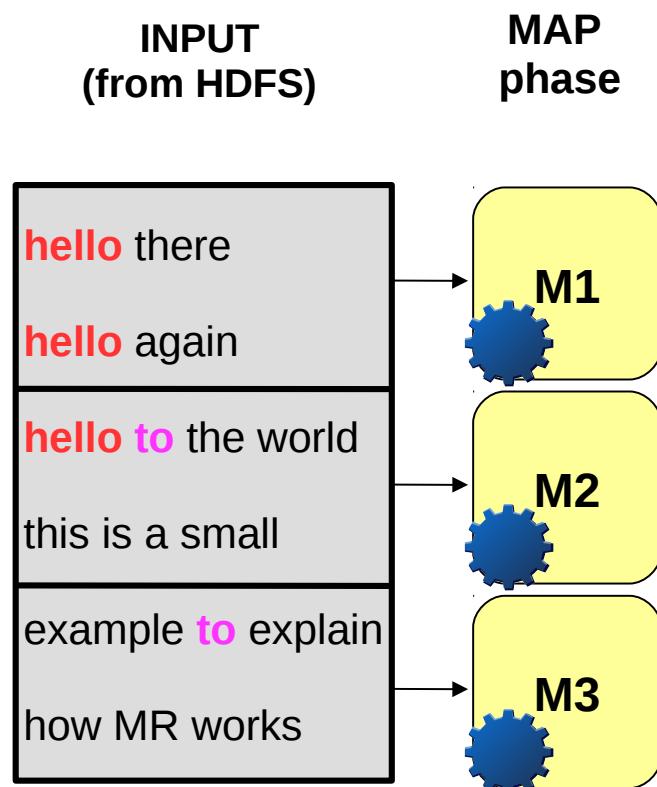
this is a small

example to explain

how MR works

3 splits
(HDFS)

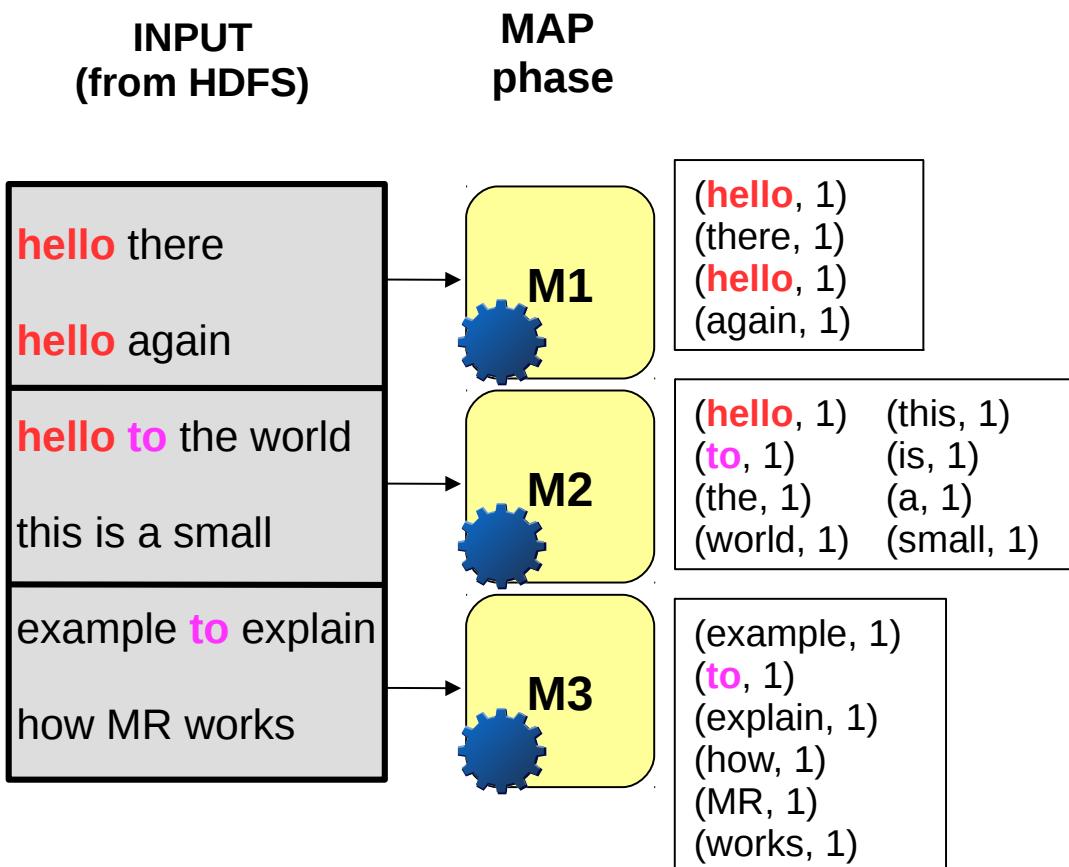
WordCount Example



3 splits
(HDFS)

3 mappers

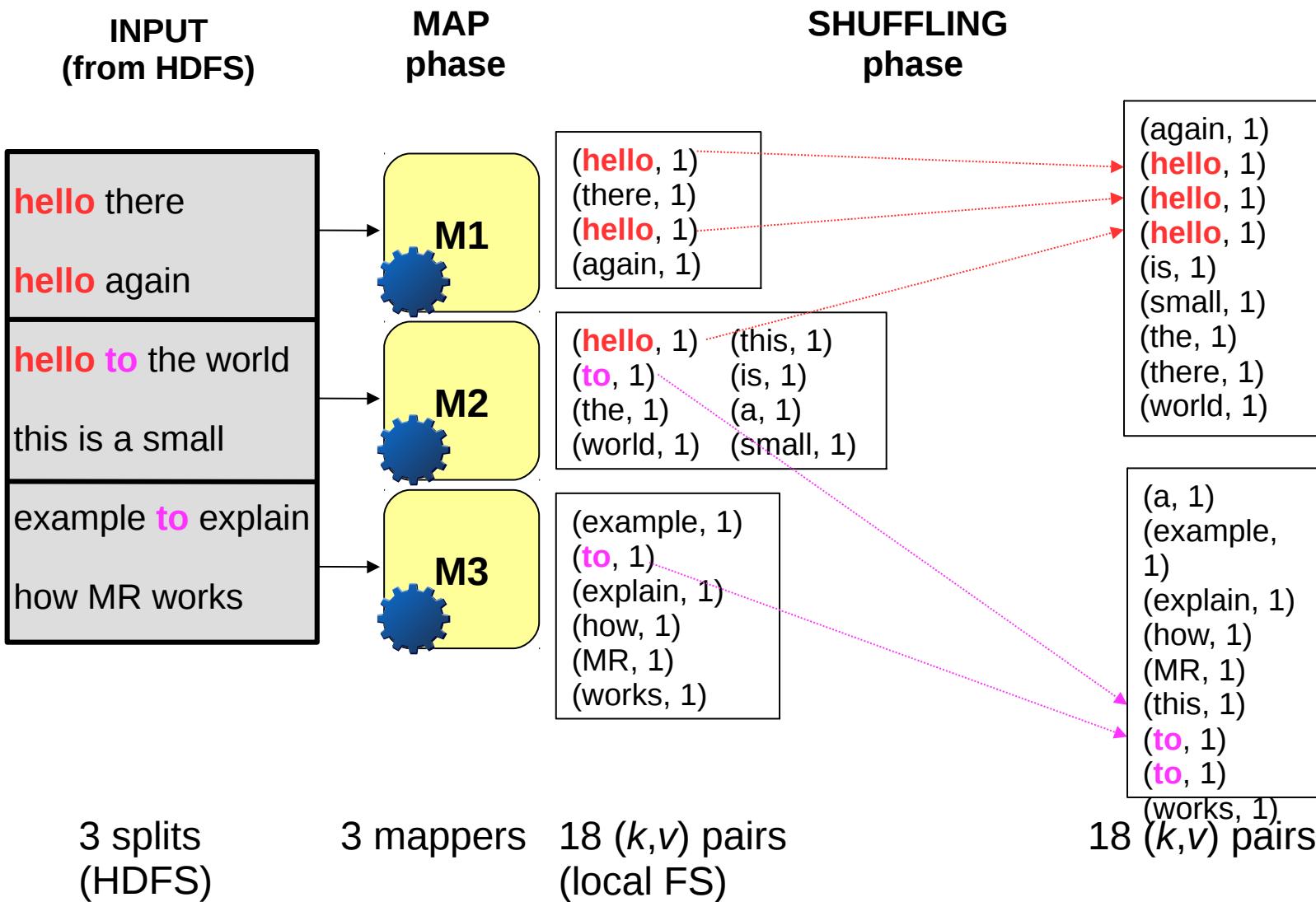
WordCount Example



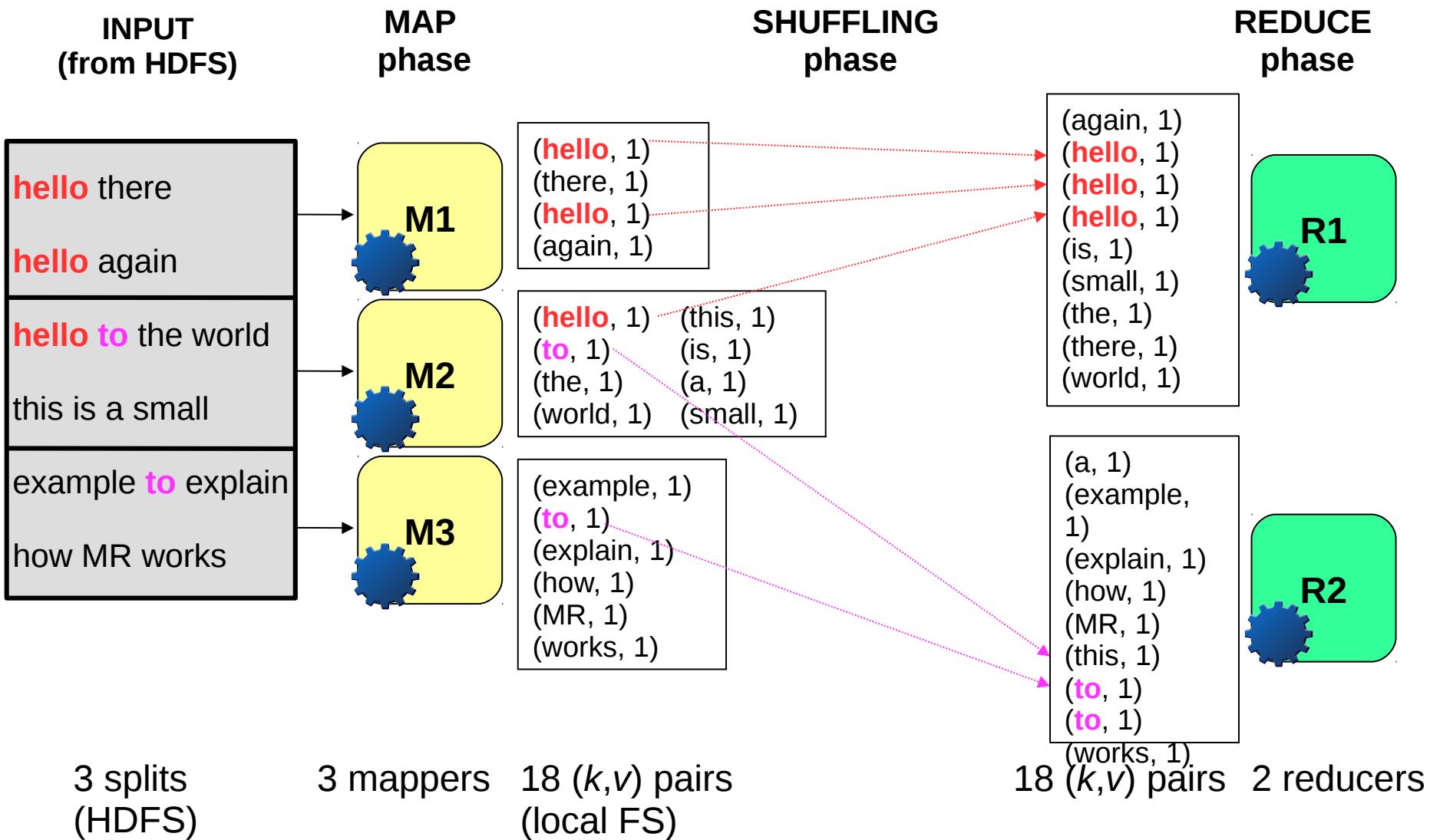
3 splits
(HDFS)

3 mappers
18 (k,v) pairs
(local FS)

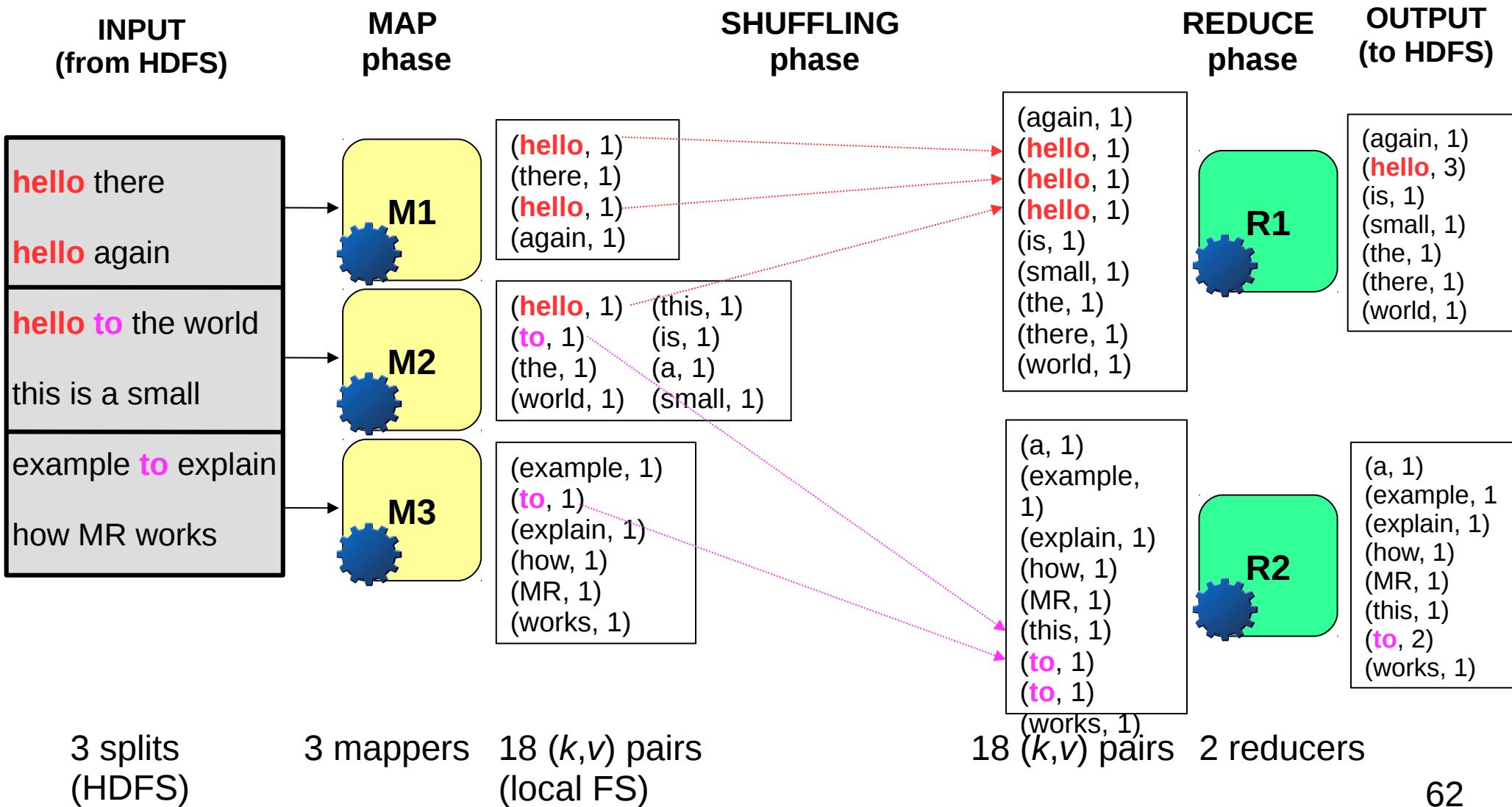
WordCount Example



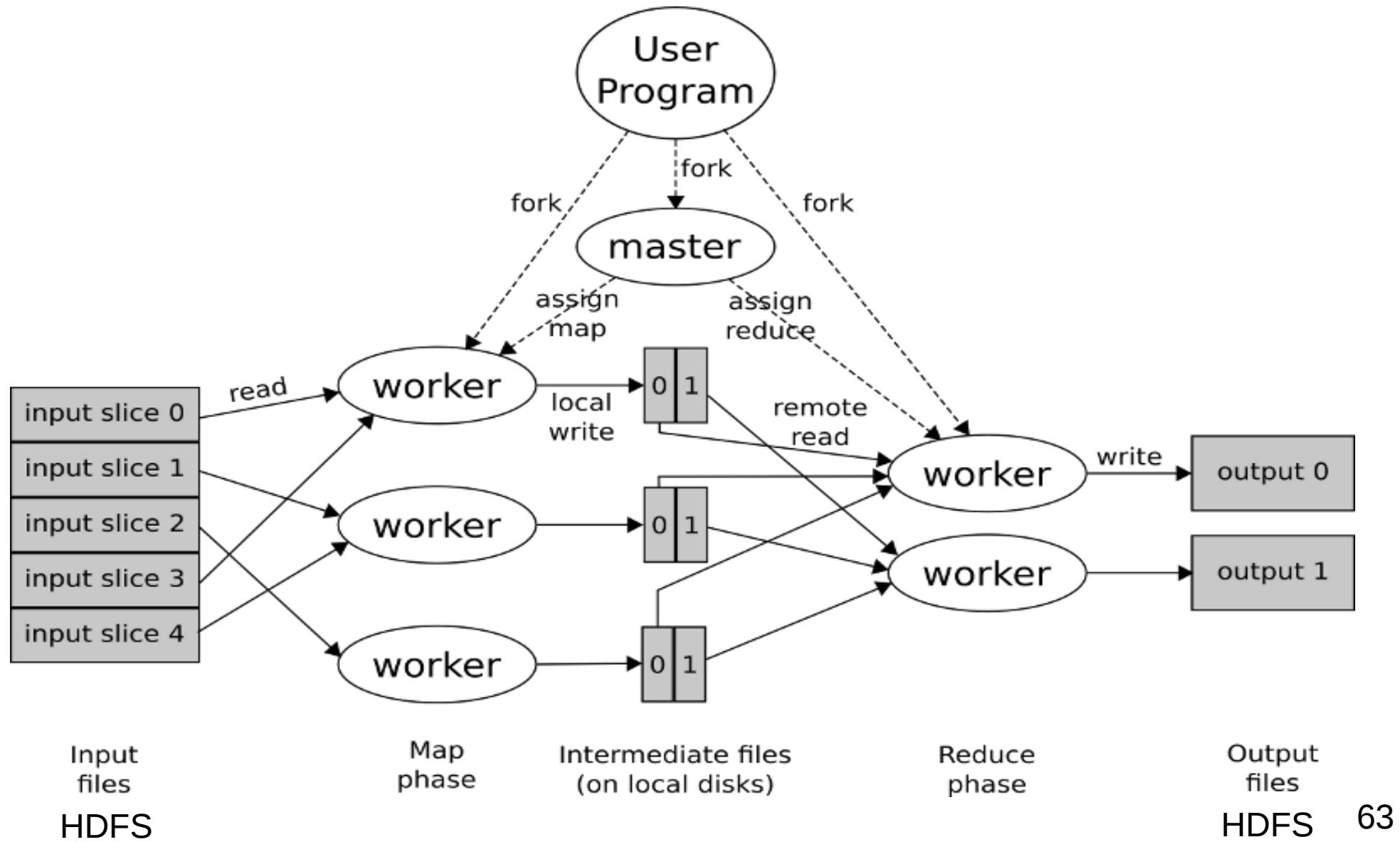
WordCount Example



WordCount Example



Workflow in Hadoop



WordCount Source Code

```
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;

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);
            }
        }
    }

    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));
        }
    }
}

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);
}
```

WordCount: the driver program

```
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);  
}
```

WordCount: the `map()` function

```
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: the `reduce()` function

```
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));  
    }  
}
```

Theoretical Issues in MapReduce

Paper titles:

- “On the Computational Complexity of MapReduce”
- “A new Computation Model for Cluster Computing”
- “Fast Greedy Algorithms in MapReduce and Streaming”
- “Minimal MapReduce Algorithms”
- “Filtering: A Method for Solving Graph Problems in MapReduce”
- “A Model of Computation for MapReduce” (**SODA 2010**)

MapReduce Limitations

- Difficult to design efficient/optimal algorithms (everything must be expressed in key-value pairs)
- A lot of disk I/Os (mappers reading HDFS and writing local data)
- A lot of network traffic (shuffling is expensive)
- Difficult to handle data skew (the curse of the last reducer!)
- Not very good for iterative processing (requires many MR rounds, memory contents are not available across rounds)
- Not very good for **streaming applications**

Case Studies

Nokia

Nokia collects and analyzes vast amounts of data from mobile phones

Problem:

- (1) Dealing with 100TB of structured data and 500TB+ of semi-structured data
- (2) 10s of PB across Nokia, 1TB / day

Solution: HDFS data warehouse allows storing all the semi/multi structured data and offers processing data at peta byte scale

Hadoop Vendor: Cloudera

Cluster/Data size:

- (1) 500TB of data
- (2) 10s of PB across Nokia, 1TB / day

Links:

- (1) Cloudera case study (Published Apr 2012)
(http://hadoopilluminated.com/hadoop_illuminated/cached_reports/Cloudera_Nokia_Case_Study_Hadoop.pdf)
- (2) strata NY 2012 presentation slides (http://hadoopilluminated.com/hadoop_illuminated/cached_reports/Nokia_Bigdata.pdf)
Strata NY 2012 presentation

Case Studies

China Mobil Guangdong

Problem: Storing billions of mobile call records and providing real time access to the call records and billing information to customers. Traditional storage/database systems couldn't scale to the loads and provide a cost effective solution

Solution: HBase is used to store billions of rows of call record details. 30TB of data is added monthly

Hadoop vendor: Intel

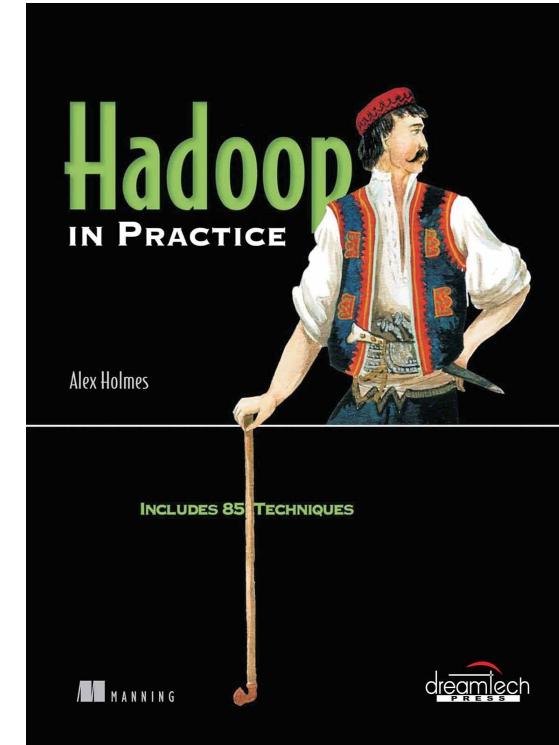
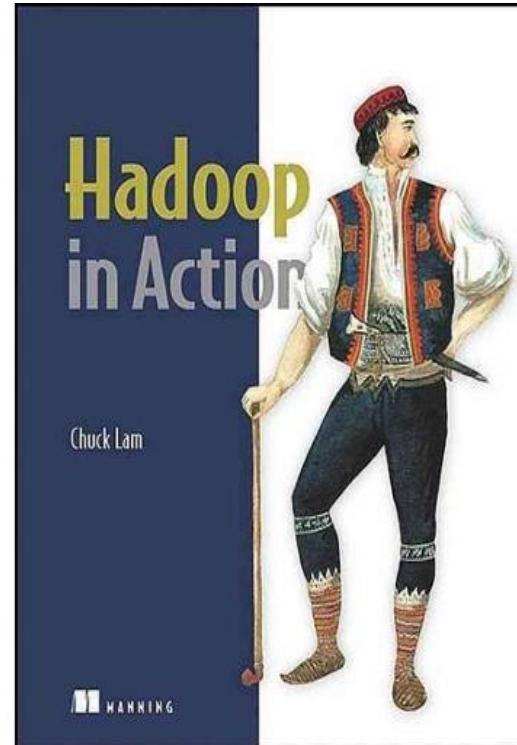
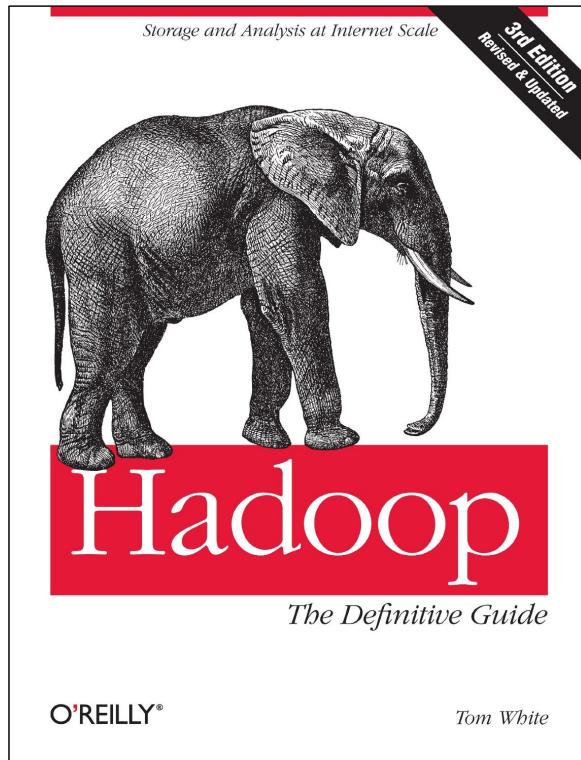
Hadoop cluster size: 100+ nodes

Links:

China Mobil Quangdong (<http://gd.10086.cn/>)

Intel APAC presentation (<http://www.slideshare.net/IntelAPAC/apac-big-data-dc-strategy-update-for-idh-launch-rk>)

Related Books



Related MOOCs

Introduction to Apache Hadoop (edX)

<https://www.edx.org/course/introduction-apache-hadoop-linuxfoundationx-lfs103x>

Hadoop platform and application framework (Coursera)

<https://www.coursera.org/learn/hadoop>

Introduction to Big Data (Coursera)

<https://www.coursera.org/learn/big-data-introduction>

Thank You

Questions ?