

# BIG DATA – HADOOP

# Outline

- ▶ Introduction to BigData / Hadoop
- ▶ HDFS Overview
- ▶ MapReduce Framework
- ▶ YARN Overview

# Introduction to Big Data / Hadoop

- ▶ What is BigData?
- ▶ 3Vs – Volume, Velocity, Variety
- ▶ Big Data in Industry
- ▶ Hadoop Overview
- ▶ Hadoop Characteristics and Benefits
  - ▶ Accessible, Robust, Scalable, Simple
  - ▶ Economical, Reliable, Flexible
- ▶ Hadoop Architecture
- ▶ Hadoop Ecosystem
- ▶ Hadoop 1.x vs Hadoop 2.x vs Hadoop 3.x

# What is Big Data?



- ▶ Huge Amount of Data (Terabytes or Petabytes)
- ▶ Big data is the term for a collection of data sets so **large and complex** that it becomes **difficult** to process using on-hand database management tools or traditional data processing applications
- ▶ The challenges include capture, curation, storage, search, sharing, transfer, analysis, and visualization

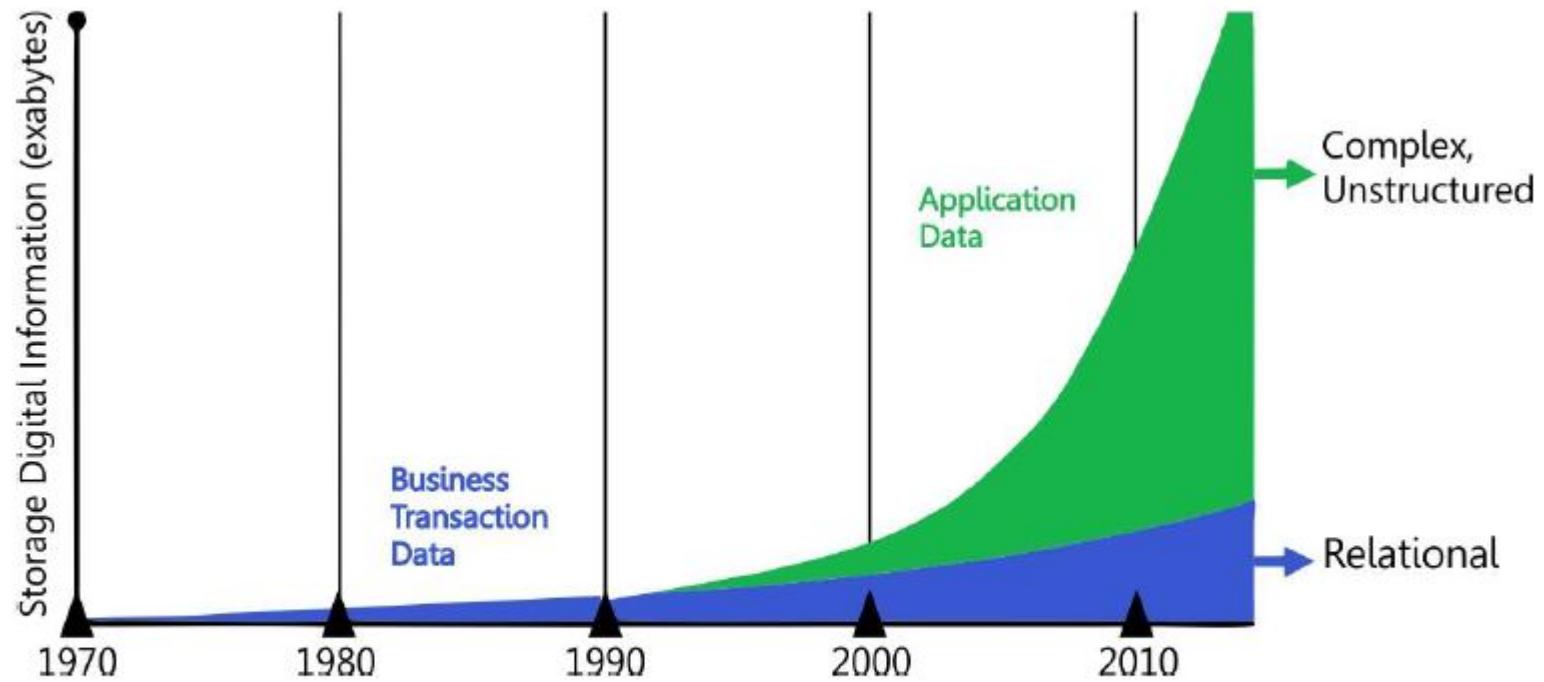
# Sources of Big Data Generation

- ▶ Systems / Enterprises generate huge amount of data from Terabytes to Petabytes of information



NYSE generates about one terabyte of new trade data per day to perform stock trading analytics to determine trends for optimal trades

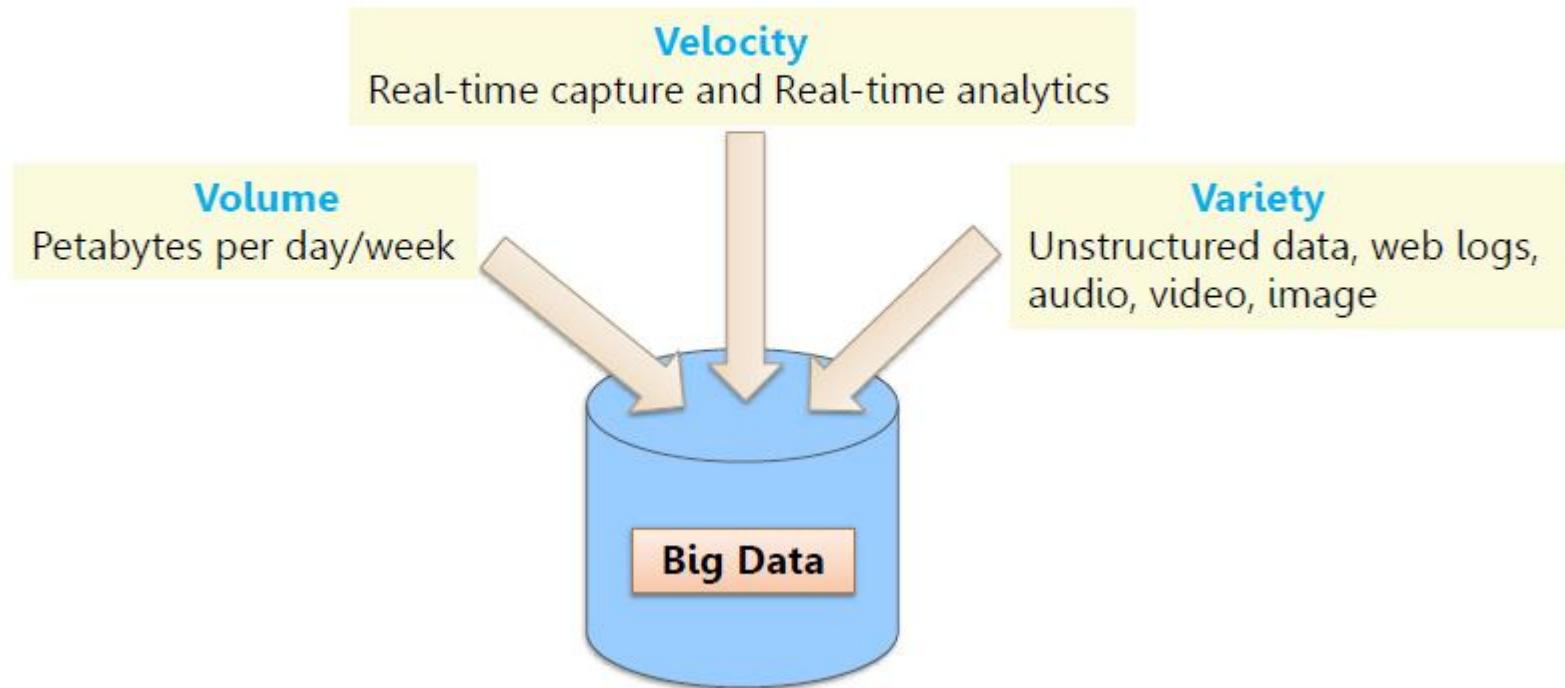
# Challenges of Big Data World



- ▶ 2,500 exabytes of new information in 2012 with interest as primary driver
- ▶ Digital universe grew by 62% last year to 800K petabytes and will go to 1.2 "Zettabytes" this year

# IBM's Definition of Big Data

- ▶ IBM's Definition – Big Data Characteristics
- ▶ <http://www-01.ibm.com/software/data/bigdata/>



# Big Data in Industry



## Web and e-tailing

- ▶ Recommendation Engines
- ▶ Ad Targeting
- ▶ Search Quality
- ▶ Abuse and Click Fraud Detection



中国移动通信  
CHINA MOBILE

## Telecommunications

- ▶ Customer Churn Prevention
- ▶ Network Performance Optimization
- ▶ Calling Data Record (CDR) Analysis
- ▶ Analyzing Network to Predict Failure



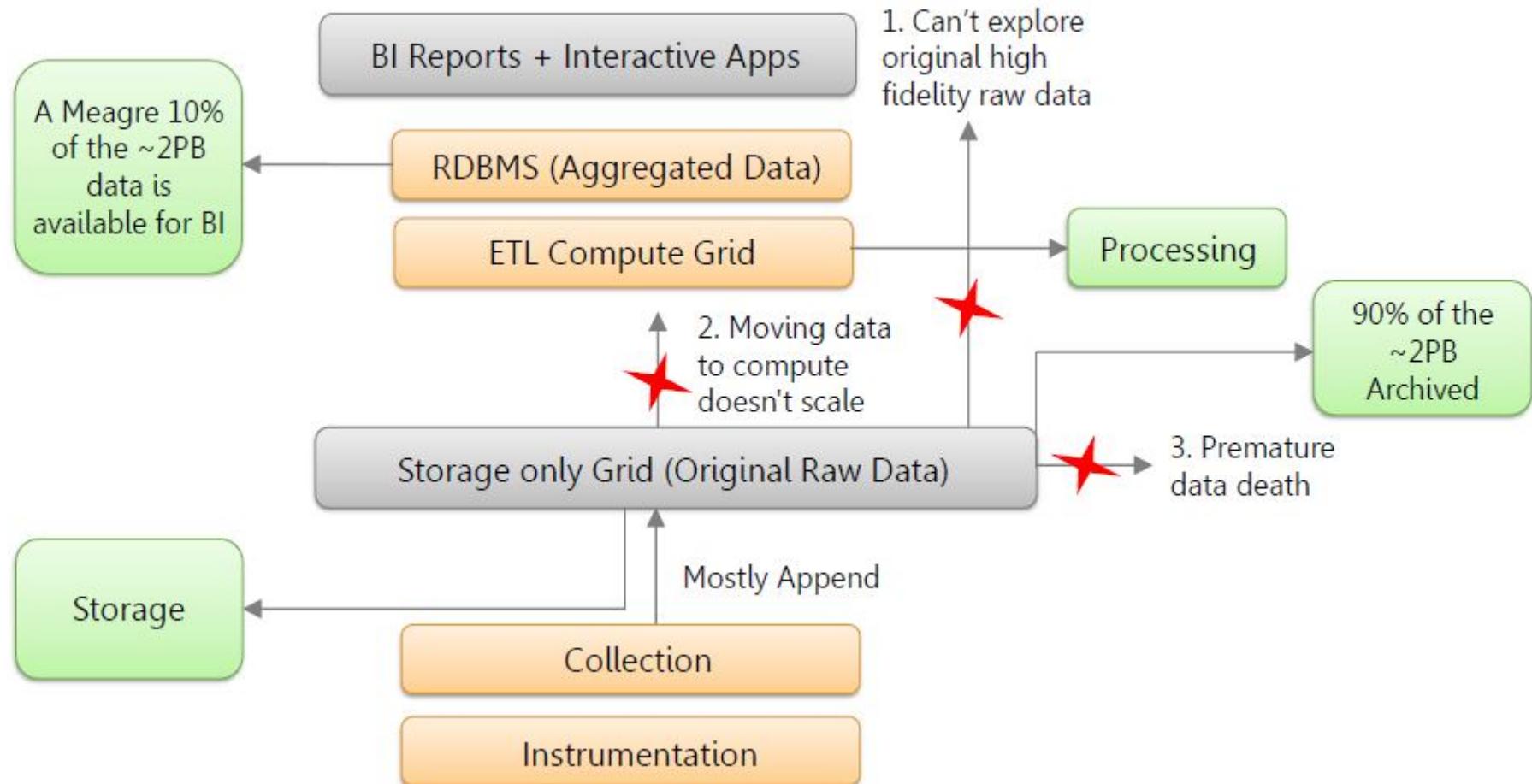
## Government

- ▶ Fraud Detection and Cyber Security
- ▶ Welfare Schemes
- ▶ Justice

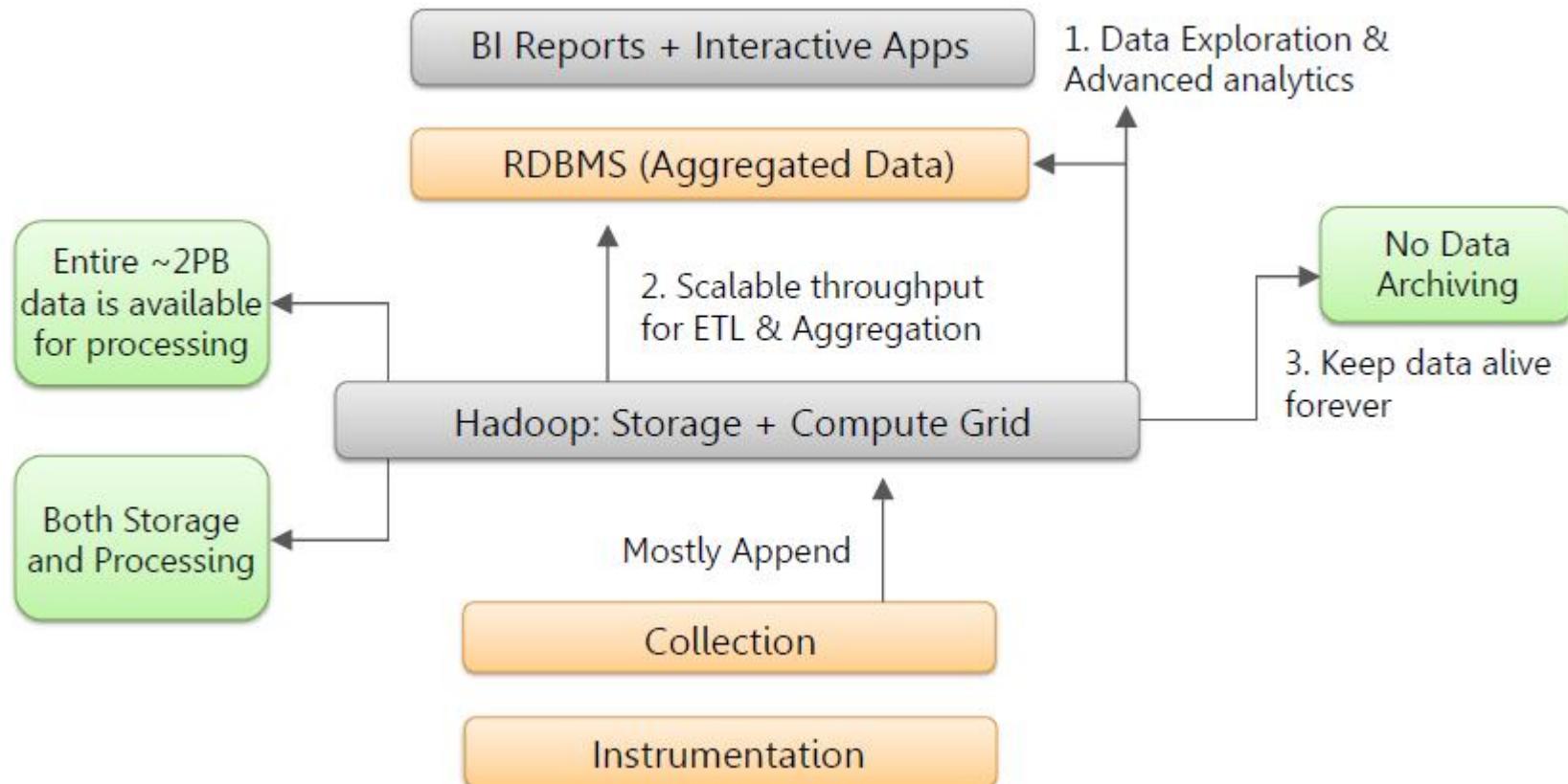
## NEXTBIO) Healthcare and Life Sciences

- ▶ Health Information Exchange
- ▶ Gene Sequencing
- ▶ Serialization
- ▶ Healthcare Service Quality Improvements
- ▶ Drug Safety

# Limitations of Traditional BI Architecture



# Solution = Storage + Processing Layer

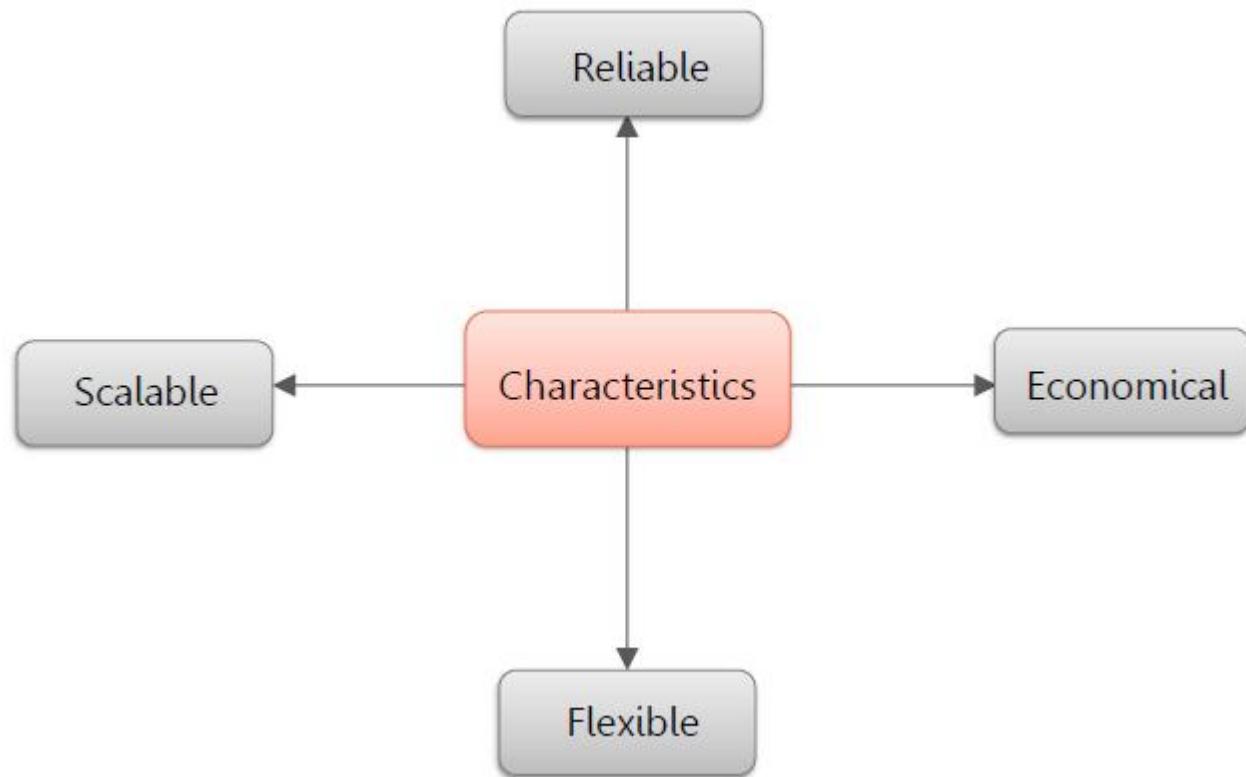


# What is Hadoop?

Apache Hadoop is a [framework](#) that allows the distributed processing of large data sets across clusters of commodity computers using a simple programming model

It is an [Open-source Data Management](#) with scale-out storage and distributed processing

# Hadoop Key Characteristics



# Hadoop Core Components

Hadoop is a system for large scale data processing. It has two main components:

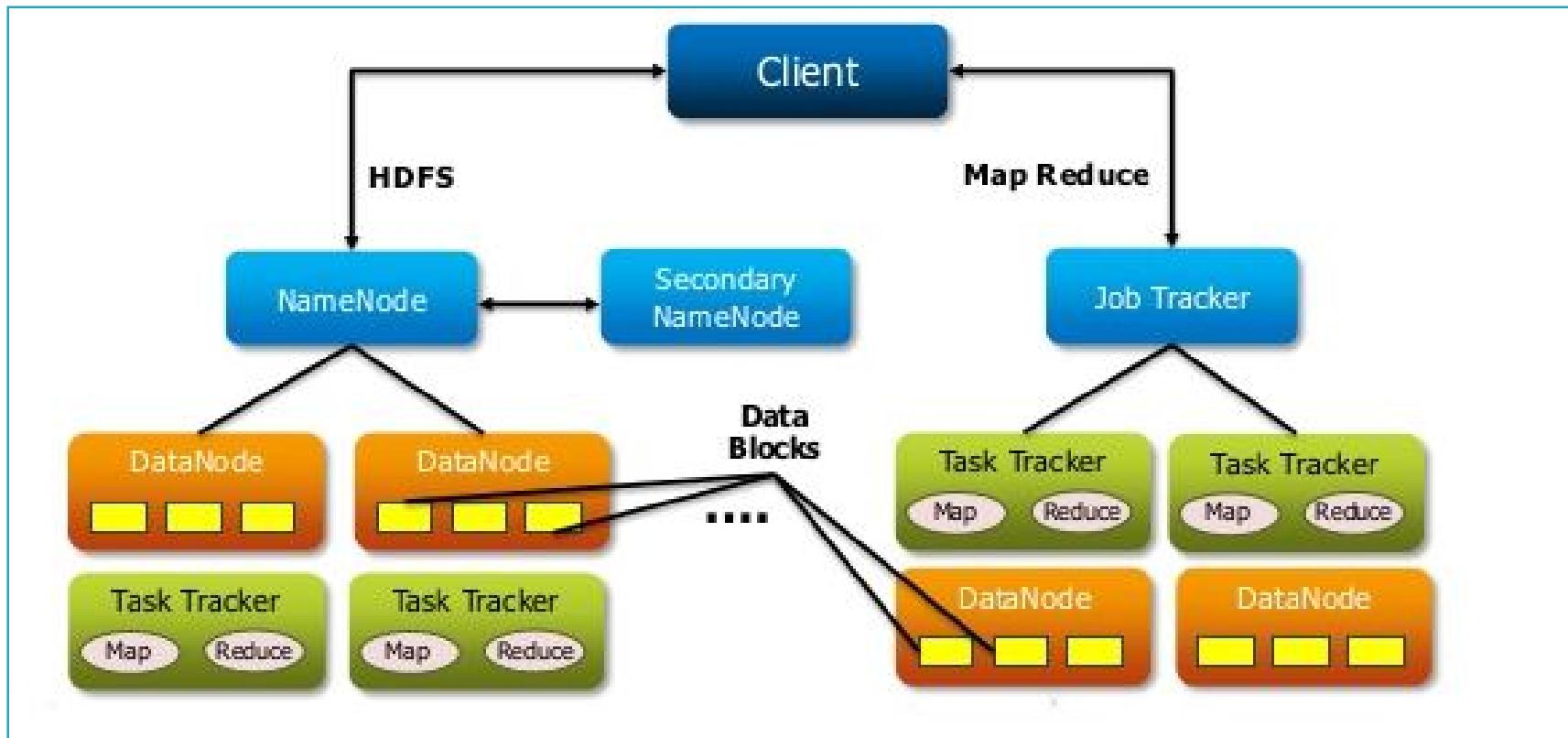
## HDFS

- Distributed across "nodes"
- Natively redundant
- NameNode tracks locations

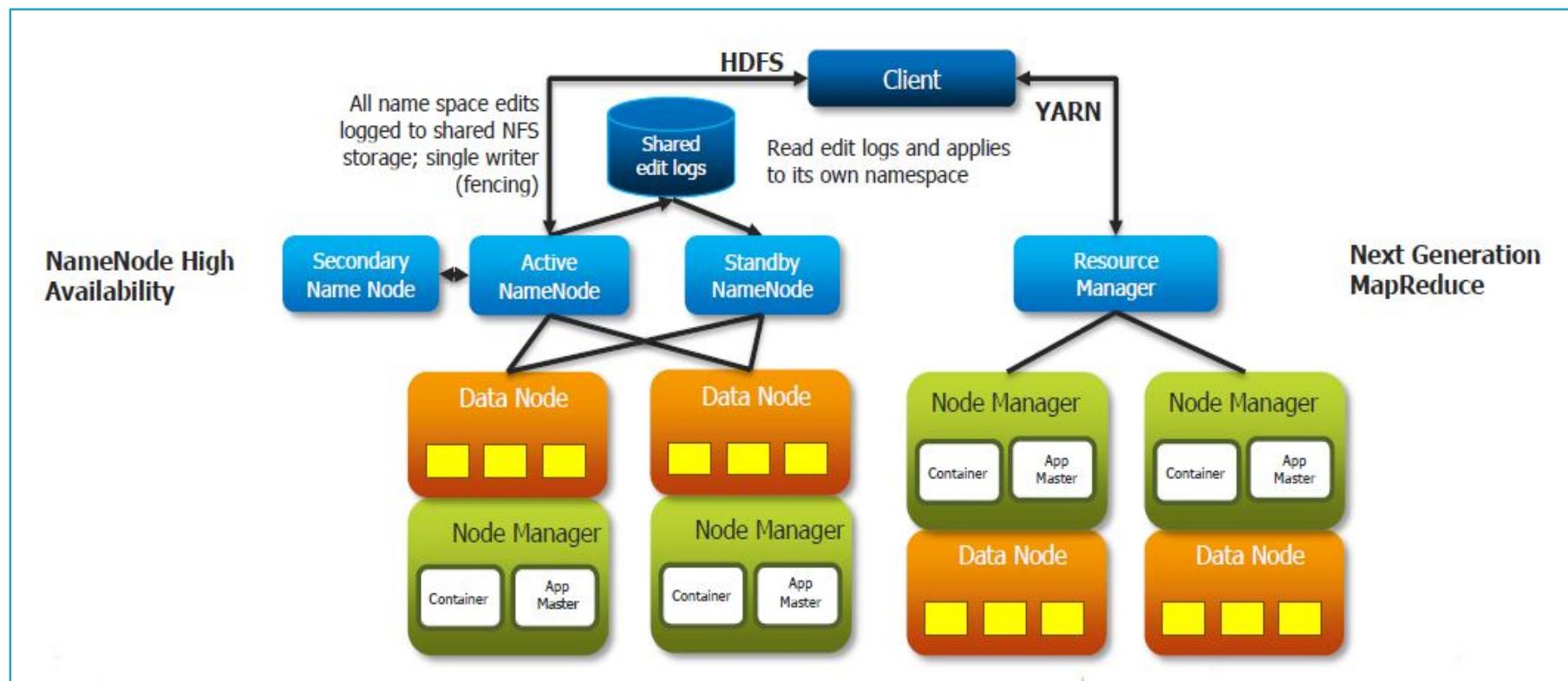
## MapReduce

- Splits a task across processors
- "near" the data & assembles results
- Self-Healing, High Bandwidth
- Clustered storage
- JobTracker manages the TaskTrackers

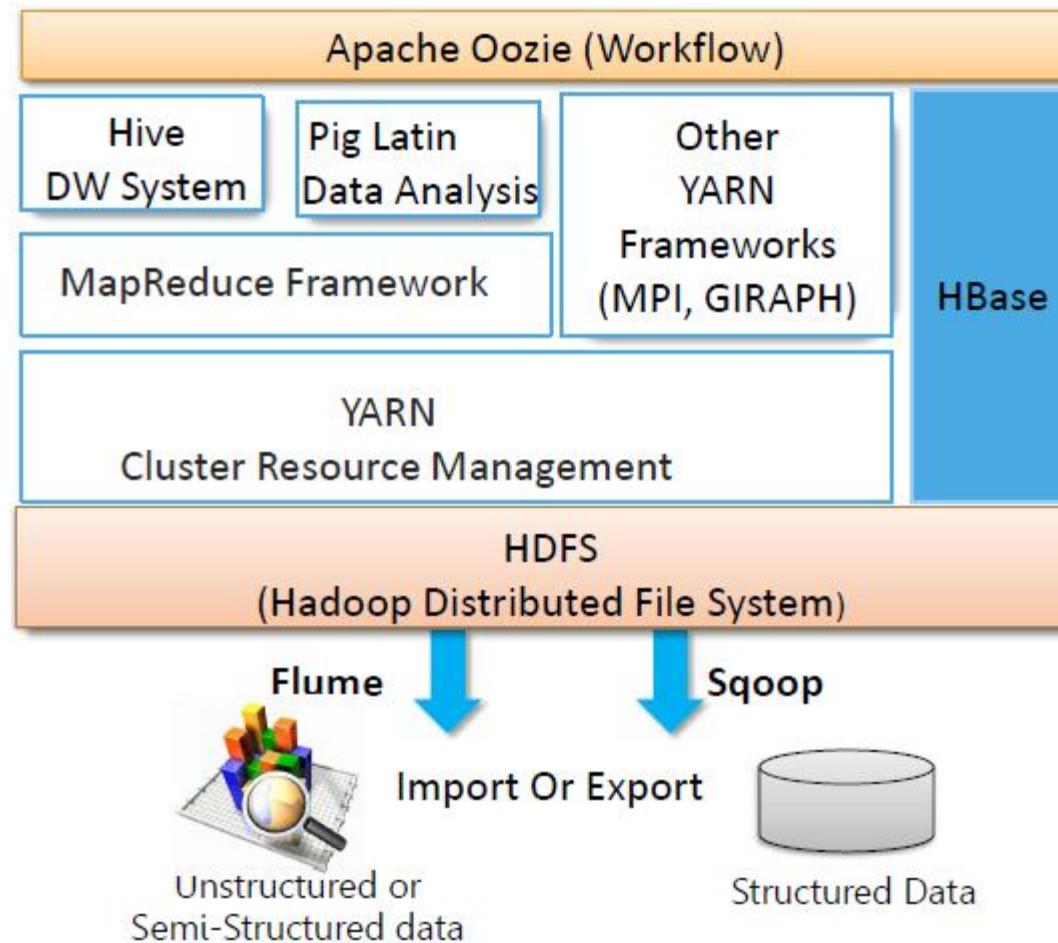
# Hadoop Architecture (1.x)



# Hadoop Architecture (2.x)



# Hadoop EcoSystem



# Different Hadoop Modes

You can use Hadoop in following modes:

## Standalone (or Local) Mode

- ▶ No Hadoop daemons, entire process runs in a single JVM
- ▶ Suitable for running MapReduce programs during development
- ▶ Has no DFS access

## Pseudo-Distributed Mode

- ▶ Hadoop daemons up, but on a single machine

## Fully-Distributed/Clustered/Prod Mode

- ▶ Hadoop daemons run on a cluster of machines

Hadoop 1.X	Hadoop 2.X
Limited to 4,000 nodes per cluster	Up to 10,000 nodes per cluster
Jobtracker bottleneck	Efficient cluster utilization - YARN
Has only one namespace for handling HDFS	Supports multiple namespace for handling HDFS
Map and Reduce slots are static	Not restricted to Java
Has only one job - to run MapReduce	Any application can integrate with Hadoop

# Hadoop 2.x vs Hadoop 3.x

Attributes	Hadoop 2.x	Hadoop 3.x
Handling Fault-tolerance	Through replication	Through erasure coding
Storage	Consumes 200% in HDFS	Consumes just 50%
Scalability	Limited	Improved
File System	DFS, FTP and Amazon S3	All features plus Microsoft Azure Data Lake File System
Manual Intervention	Not needed	Not needed
Scalability	Up to 10,000 nodes in a cluster	Over 10,000 nodes in a cluster
Cluster Resource Management	Handled by YARN	Handled by YARN
Data Balancing	Uses HDFS balancer for this purpose	Uses Intra-data node balancer

# Hadoop 2.x vs Hadoop 3.x

Attributes	Hadoop 2.x	Hadoop 3.x
Handling Fault-tolerance	Through replication	Through erasure coding
Storage	Consumes 200% in HDFS	Consumes just 50%
Scalability	Limited	Improved
File System	DFS, FTP and Amazon S3	All features plus Microsoft Azure Data Lake File System
Manual Intervention	Not needed	Not needed
Scalability	Up to 10,000 nodes in a cluster	Over 10,000 nodes in a cluster
Cluster Resource Management	Handled by YARN	Handled by YARN
Data Balancing	Uses HDFS balancer for this purpose	Uses Intra-data node balancer

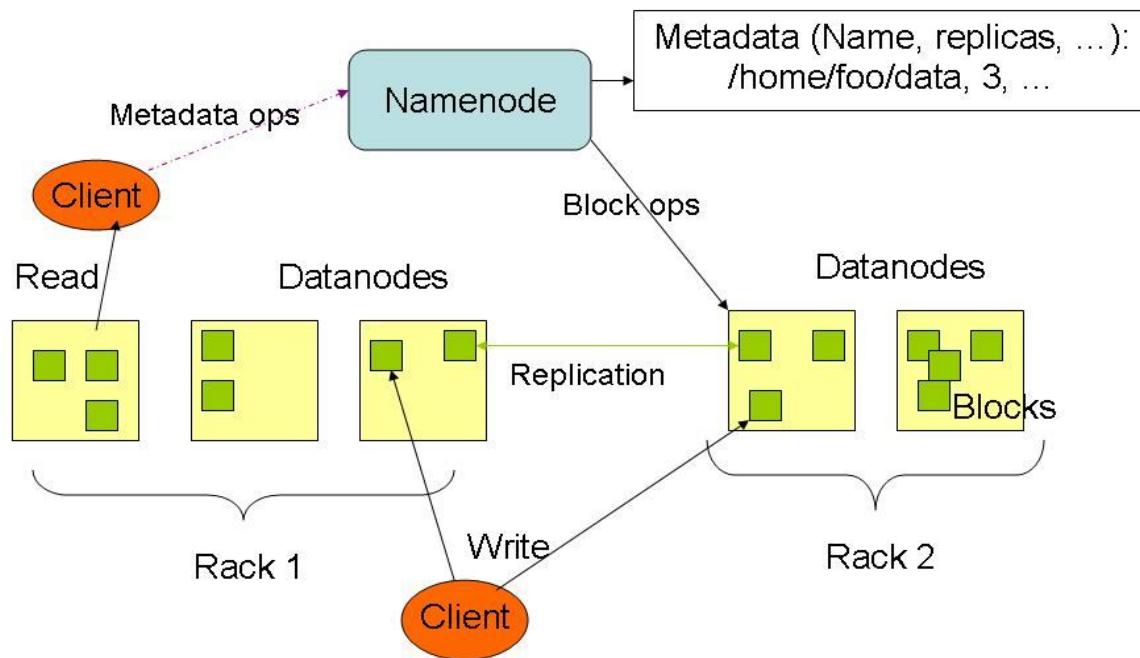
# Hadoop Configuration Files

Configuration Filenames	Description of Log Files
hadoop-env.sh	Environment variables that are used in the scripts to run Hadoop
core-site.xml	Core Hadoop Configuration settings which are common to HDFS and MapReduce
hdfs-site.xml	HDFS Configuration settings for HDFS daemons, the NameNode, the secondary NameNode and the data nodes
mapred-site.xml	MapReduce specific Configuration settings
yarn-site.xml	Configuration settings for ResourceManager and NodeManager
masters	A list of machines (one per line) that each run a secondary NameNode
slaves	A list of machines (one per line) that each run a DataNode and a NodeManager

# HDFS (Hadoop Distributed File System)

- ▶ HDFS Overview
- ▶ HDFS Architecture
- ▶ HDFS Components
  - ▶ Name Node
  - ▶ Secondary Name Node
  - ▶ Data Node
- ▶ HDFS File Write Anatomy
- ▶ HDFS File Read Anatomy
- ▶ Rack Awareness
- ▶ Hadoop File Formats
  - ▶ Text, Sequence, Avro, Parquet, ORC, etc.
- ▶ Hadoop Compression Techniques
  - ▶ Snappy, gzip, bgzip2, LZO, etc.
- ▶ Data Replication, Fault Tolerance, Resiliency, HA

# HDFS Architecture



# Name Node

## Keeps Meta data in Main Memory

- ▶ The entire metadata is in main memory
- ▶ FS meta-data is not loaded from hard disk

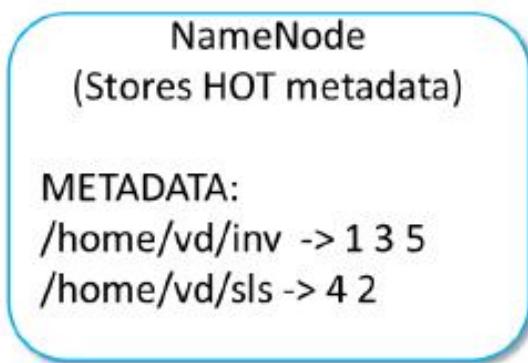
## Metadata type

- ▶ Files in HDFS
- ▶ Data Blocks for each file
- ▶ DataNodes for each block
- ▶ File attributes, e.g. access time, replication factor, access control

# Name Node (contd.)

## Transaction Log

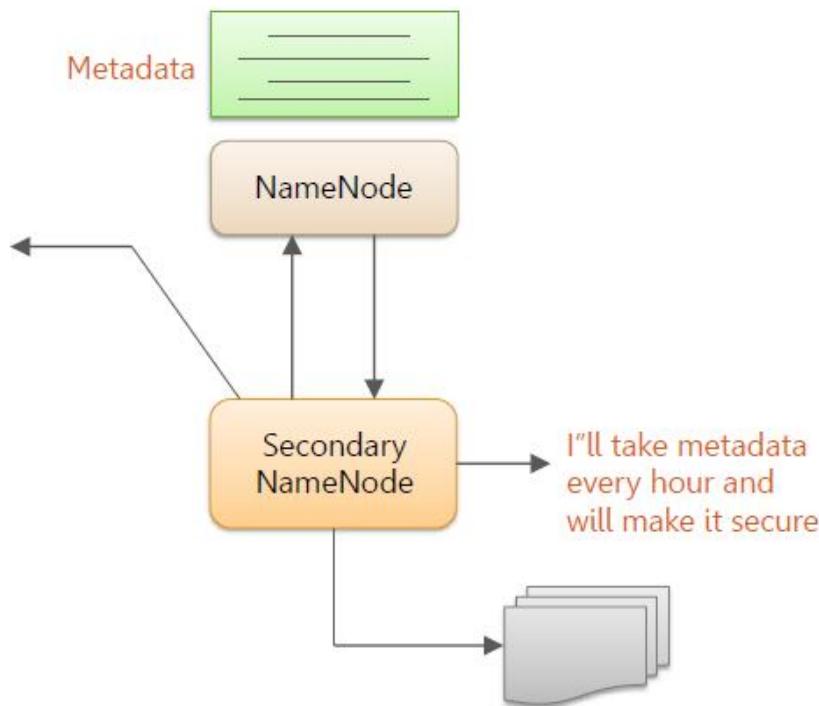
- Records file creations, file deletions, etc



## NameNode:

- Keeps track of overall file directory structure and the placement of Data Block

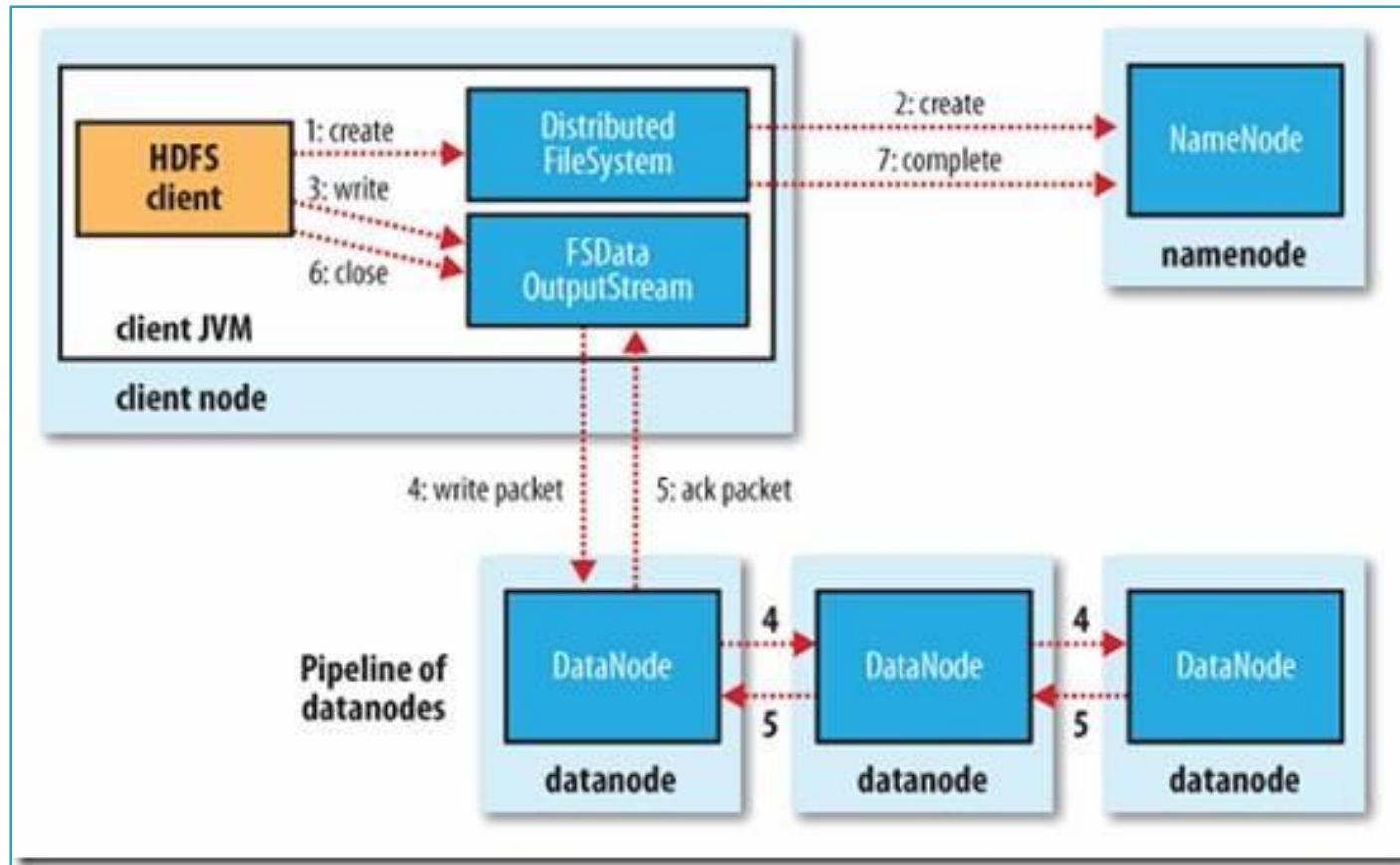
# Secondary Name Node



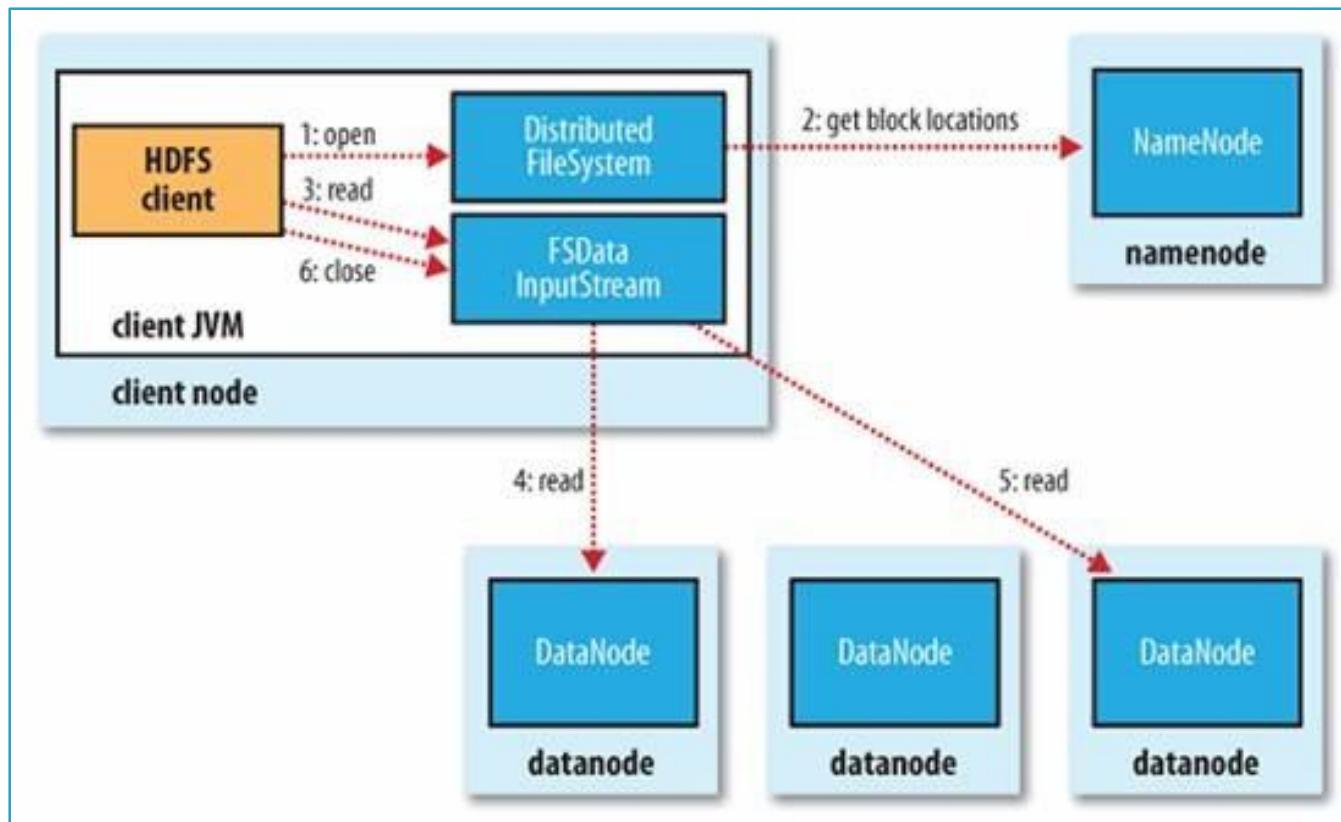
## Secondary NameNode:

- ▶ In HDFS 1.0, not a hot standby for the NameNode
- ▶ By Default connects to NameNode every hour\*
- ▶ Housekeeping, backup of NameNode metadata
- ▶ Saved metadata is used to bring up the Secondary NameNode

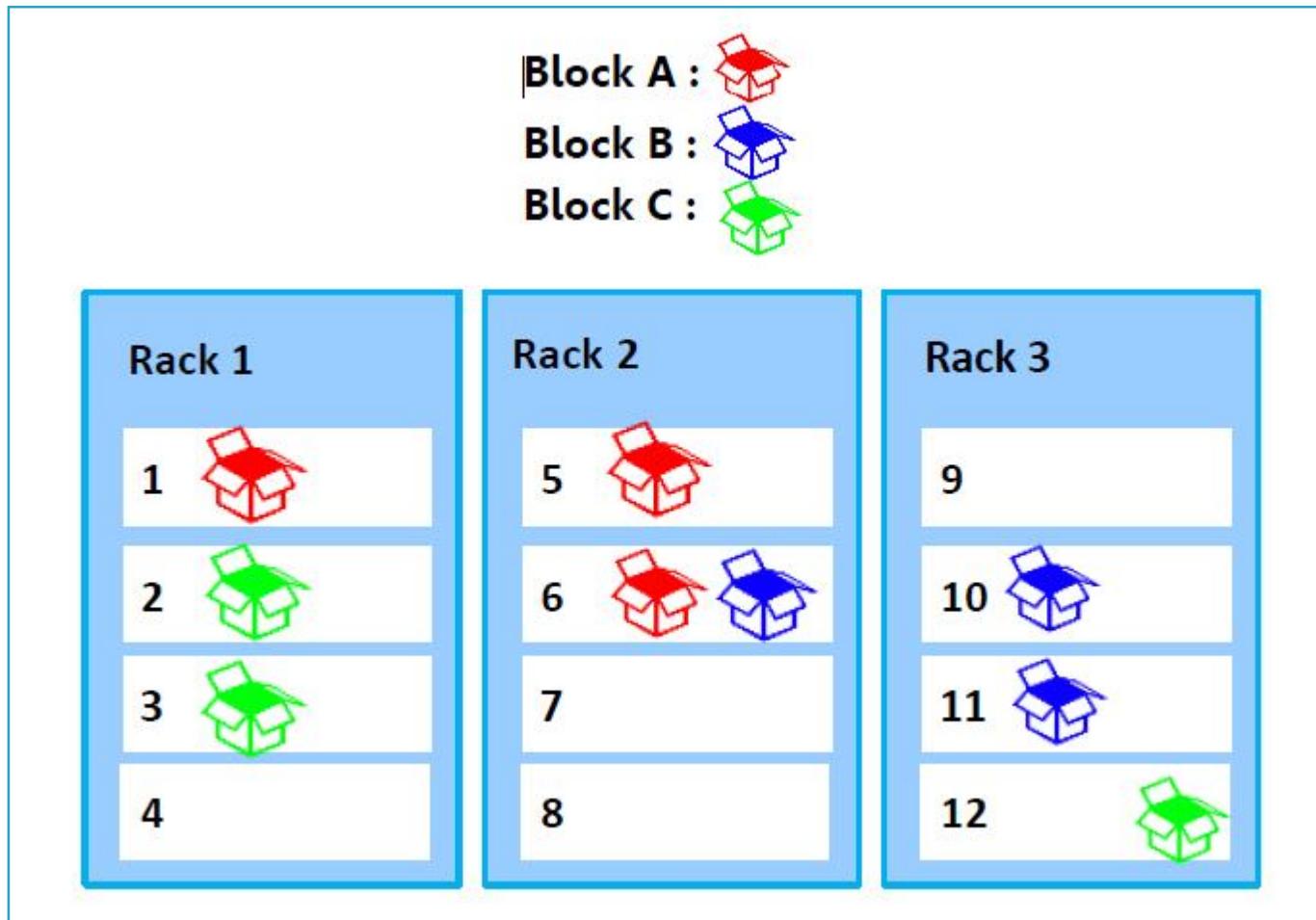
## HDFS File Write Operation



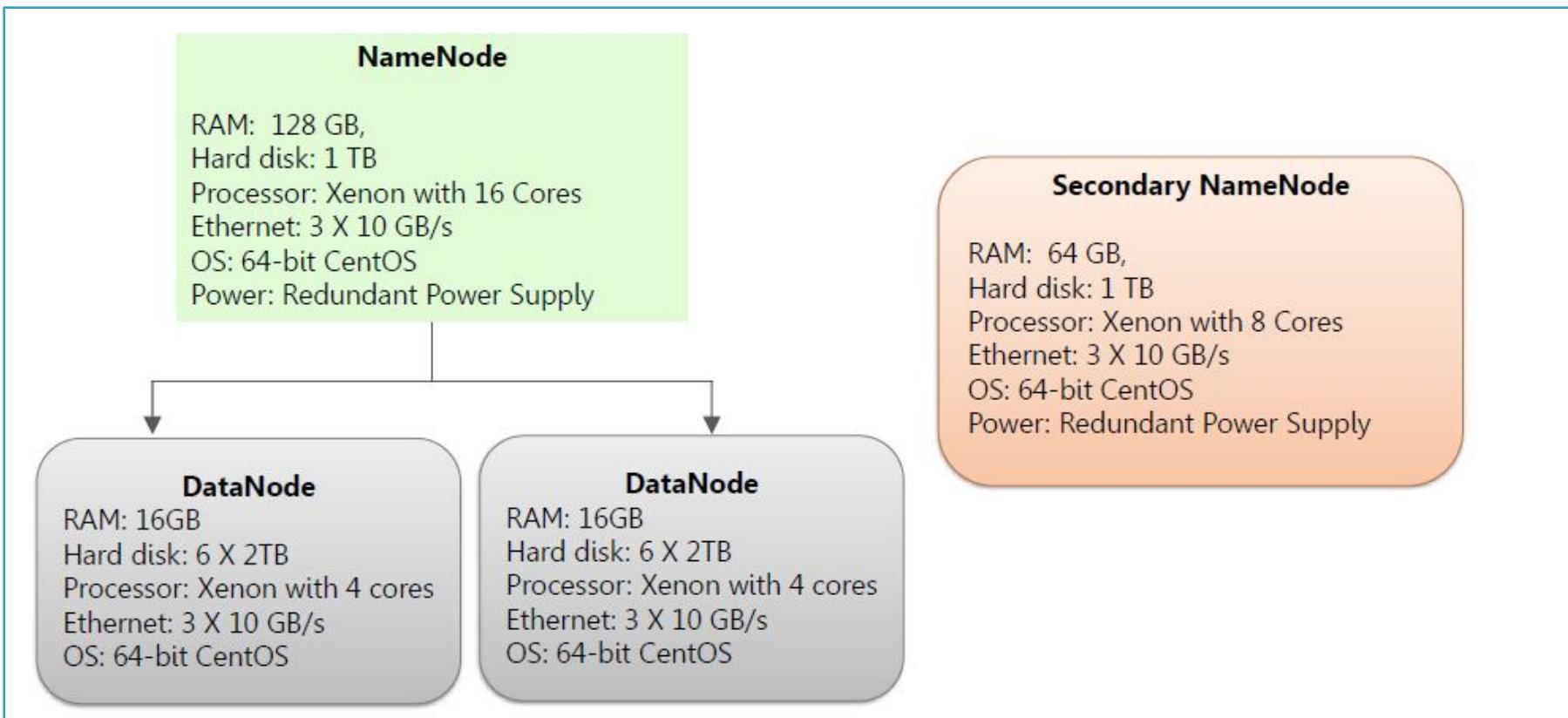
## HDFS File Read Operation



## Rack Awareness



# Typical Production Hadoop Cluster



# MapReduce Framework

- ▶ MapReduce Overview
- ▶ MapReduce Architecture
- ▶ MapReduce Concepts
  - ▶ Splits, Mappers, Reducers, Partitioners, Combiners and Counters
- ▶ Input / Output File Formats
- ▶ Map Side Join / Reduce Side Join
- ▶ YARN Overview
- ▶ YARN Components
  - ▶ Resource Manager, Node Manager, Container, App Master
- ▶ Job Scheduler
  - ▶ Fair Scheduler, Capacity Scheduler, etc.
- ▶ MR1 vs MR2
- ▶ MapReduce Job Execution
- ▶ Fault Tolerance, Resiliency, Reliability, HA

# MapReduce UseCases



## Problem Statement:

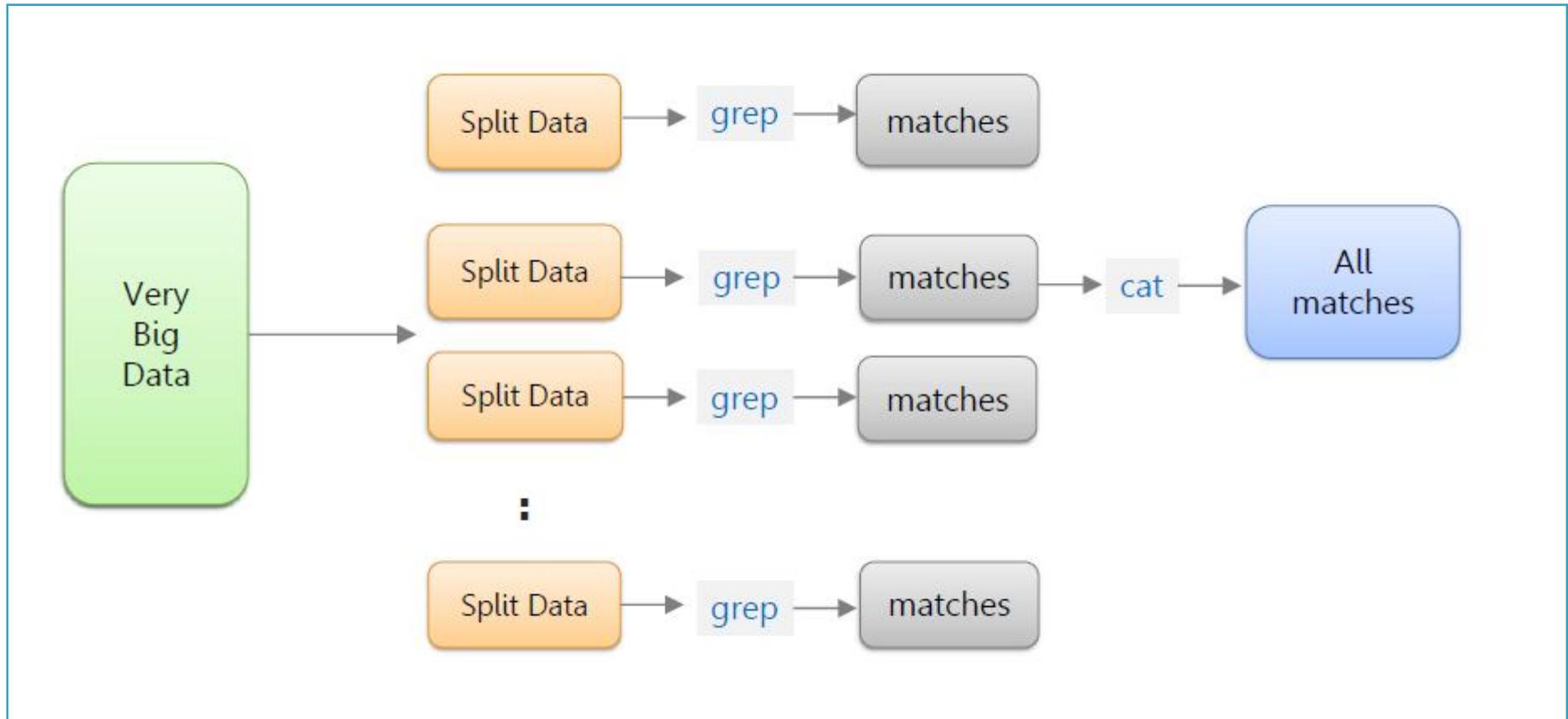
Find maximum stock market levels recorded in a span of 5 years



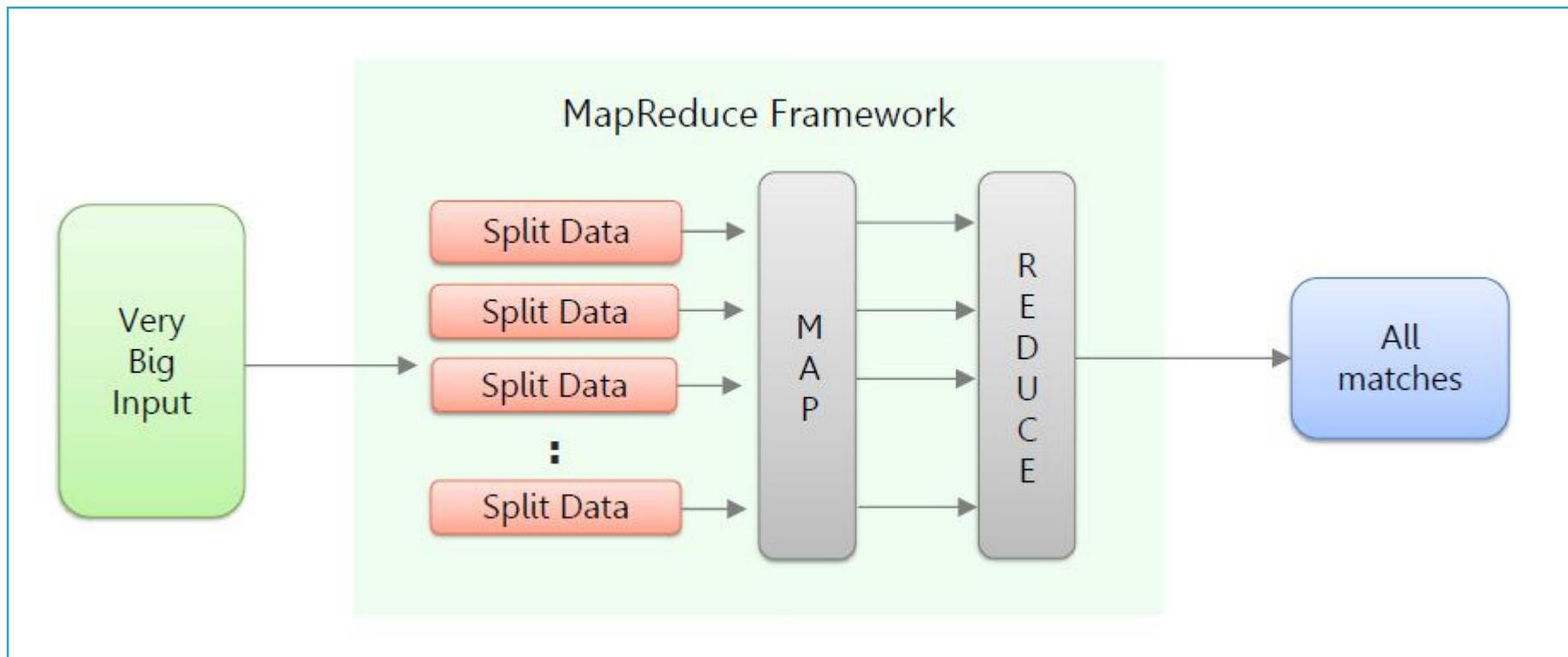
## Problem Statement:

De-identify personal identifier information

# Traditional Solution



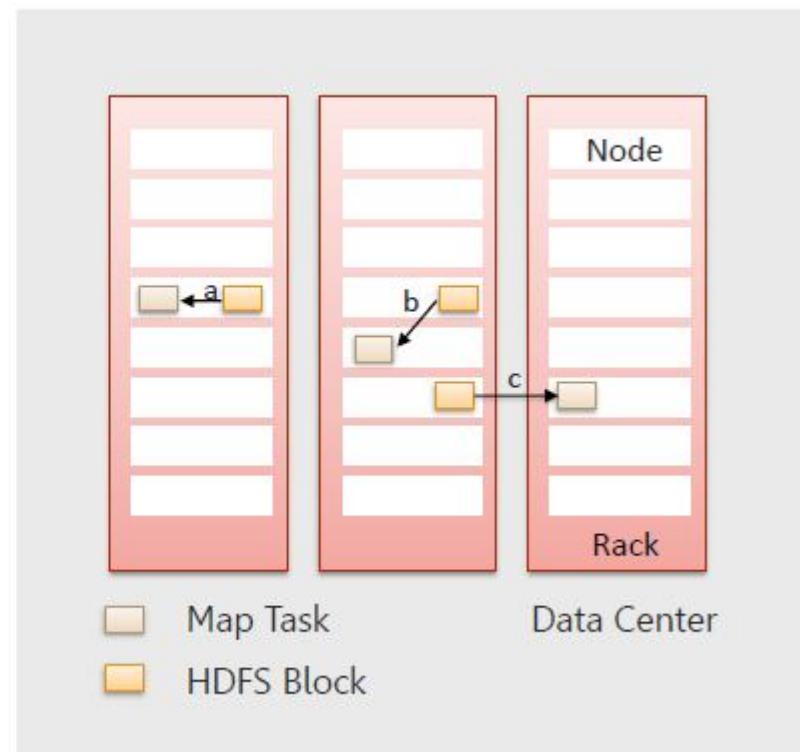
# MapReduce Solution



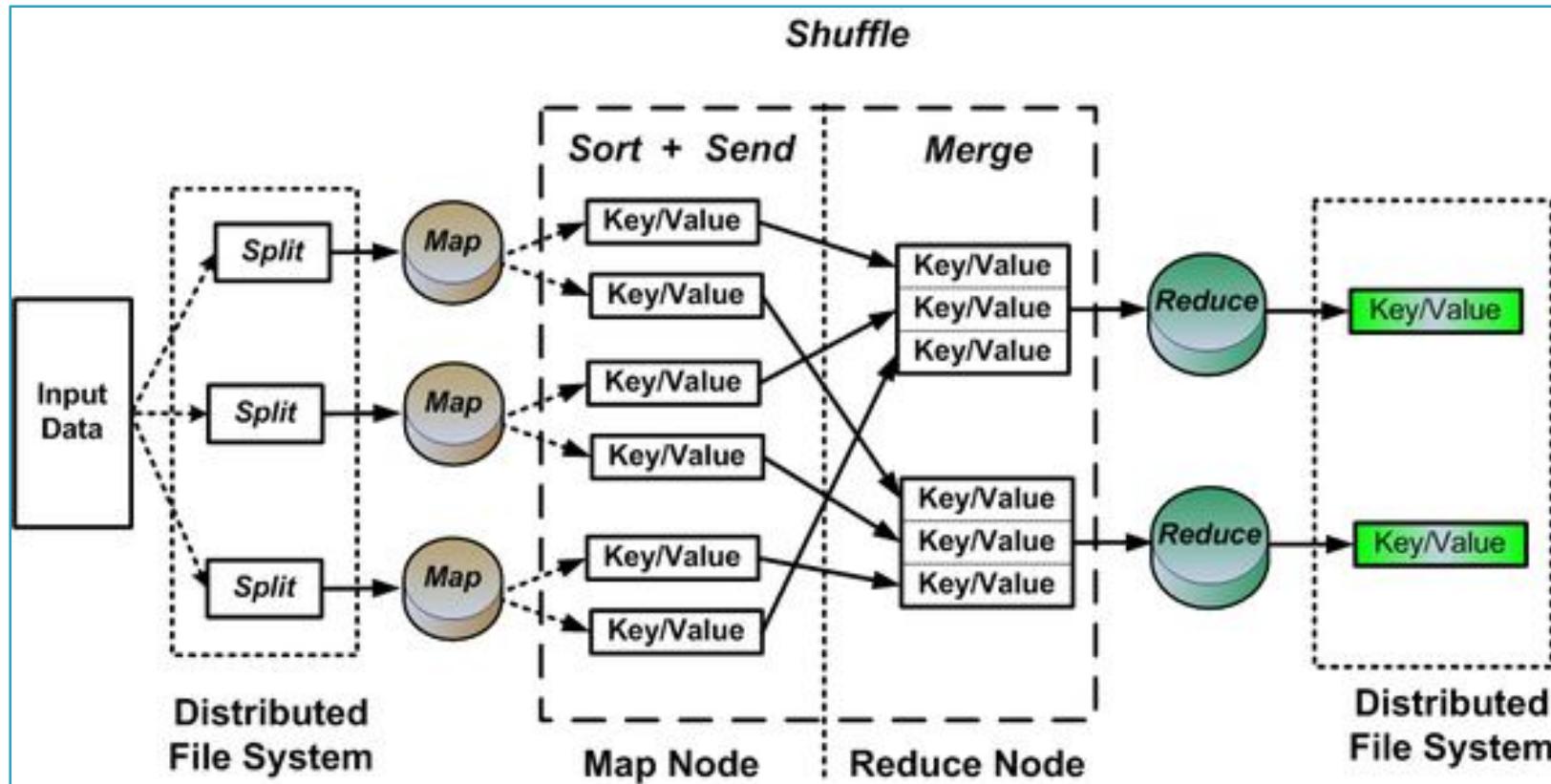
# MapReduce Advantages

The two biggest advantages of Map Reduce are:

- ▶ It takes processing to the data
- ▶ It allows processing of data in parallel



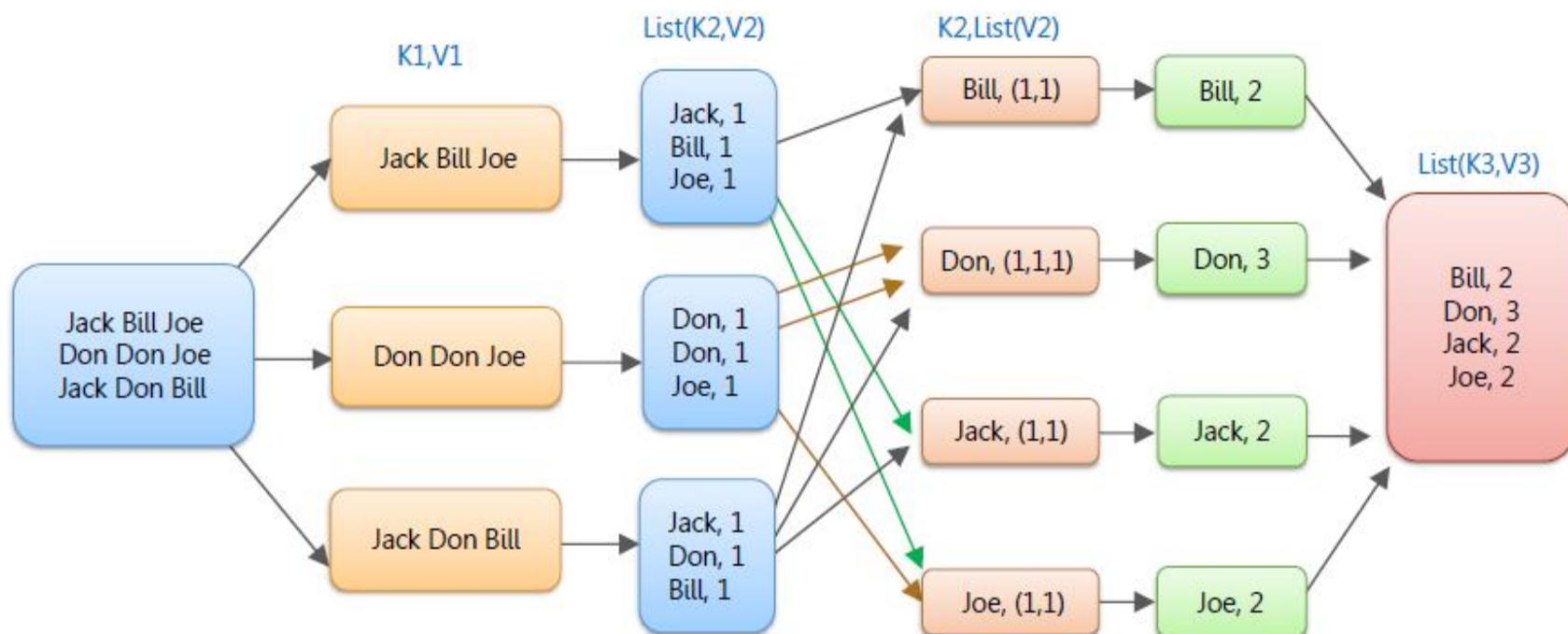
# MapReduce Architecture



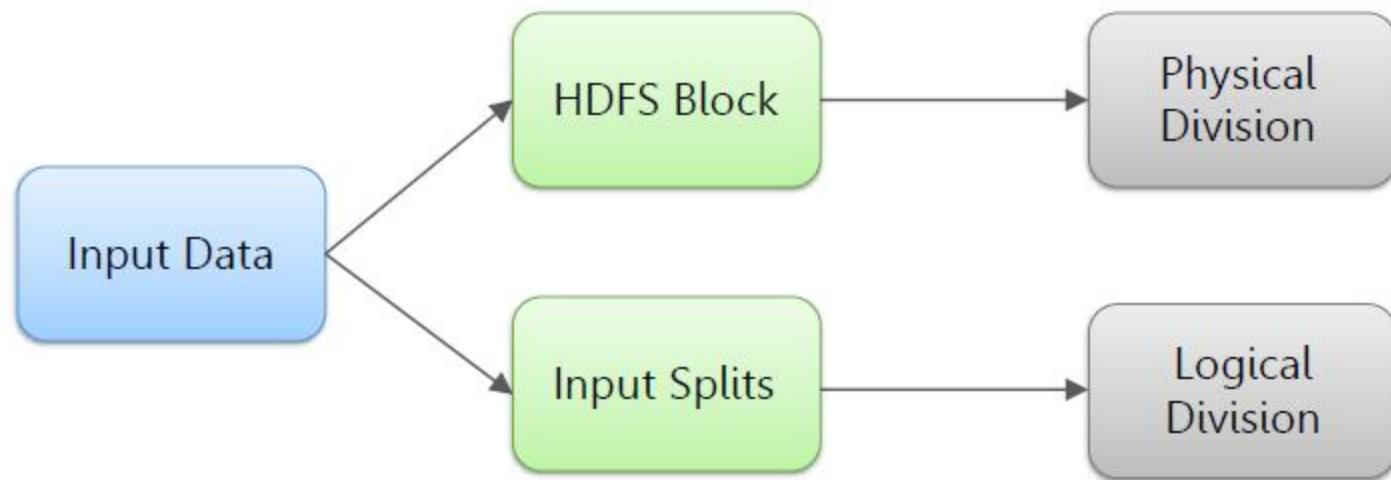
# MapReduce Process Flow

MapReduce Word Count Process Flow

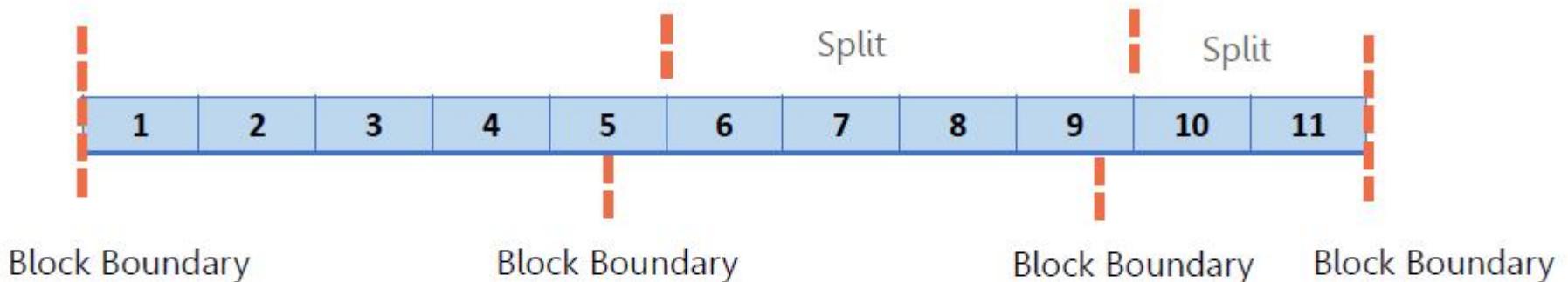
Input      Splitting      Mapping      Shuffling      Reducing      Final Result



# Input Splits in MapReduce

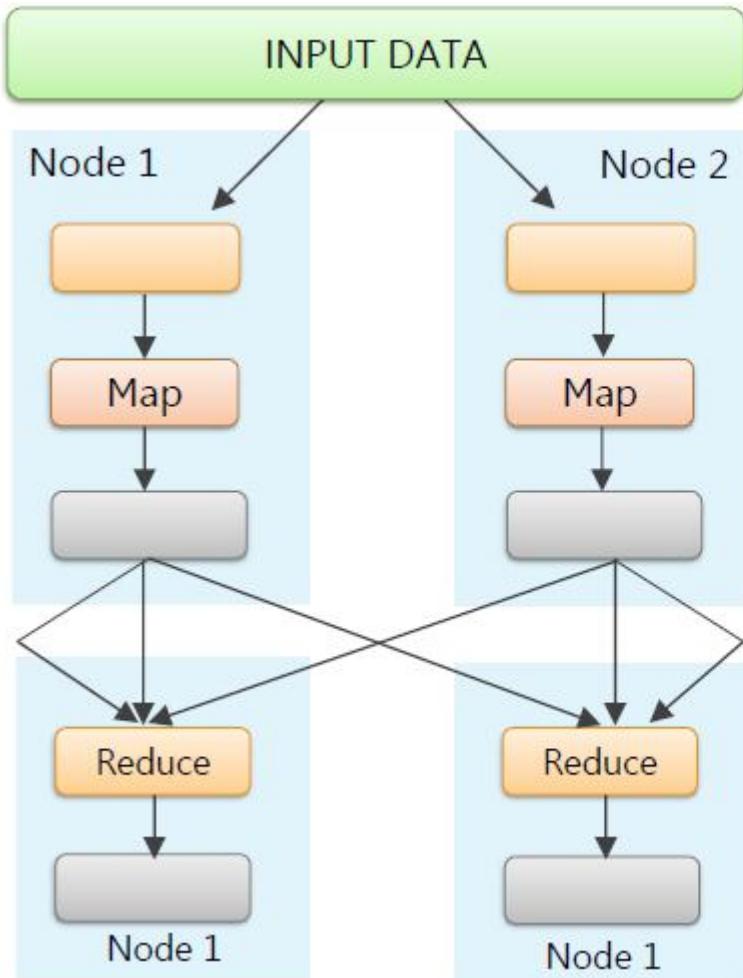


# HDFS Blocks vs Input Splits



- ▶ In an ideal scenario, Block Size == Input Split Size
- ▶ However, Logical records rarely fit into HDFS blocks
- ▶ Logical Records may cross the block boundaries
- ▶ Deltas are read remotely

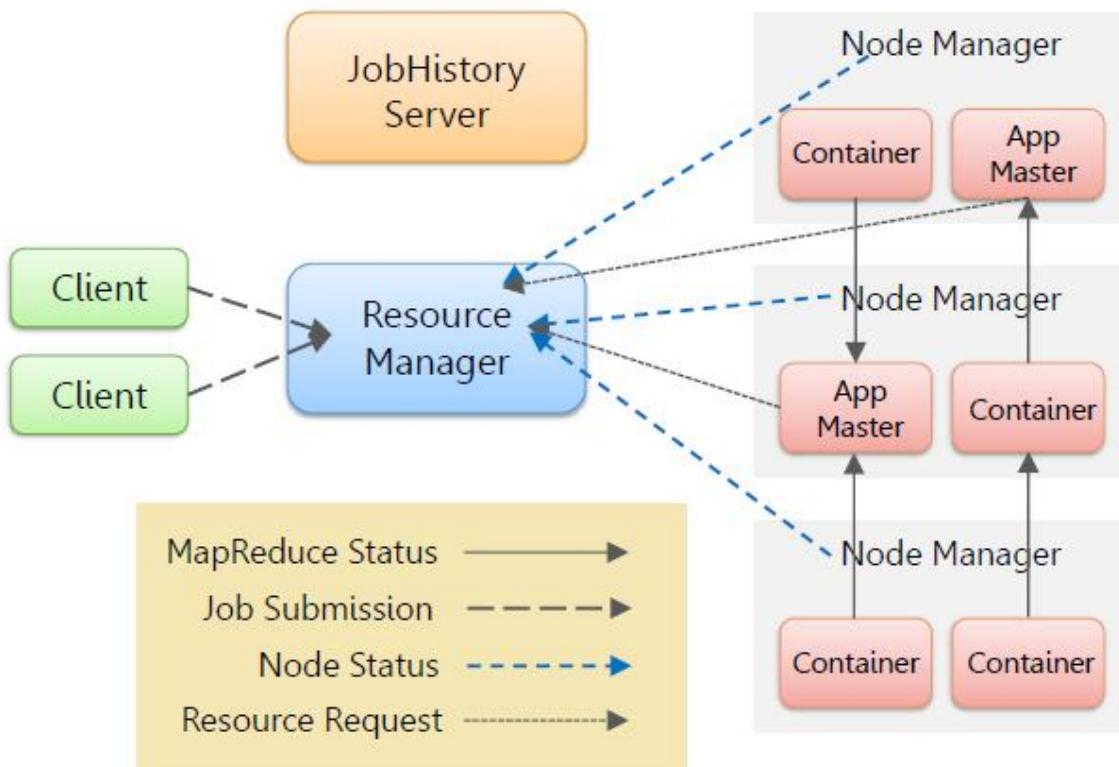
# MapReduce Flow



1. Input data is present in data nodes
2. Map tasks = Input Splits
3. Mappers produce intermediate data
4. Data is exchanged among nodes in "shuffling"
5. All data of same key goes to same reducer
6. Reducer output is stored at output location

# YARN

YARN = Yet Another Resource Negotiator



Resource Manager

- ▶ Cluster Level Resource Manager
- ▶ Long life, High Quality Hardware

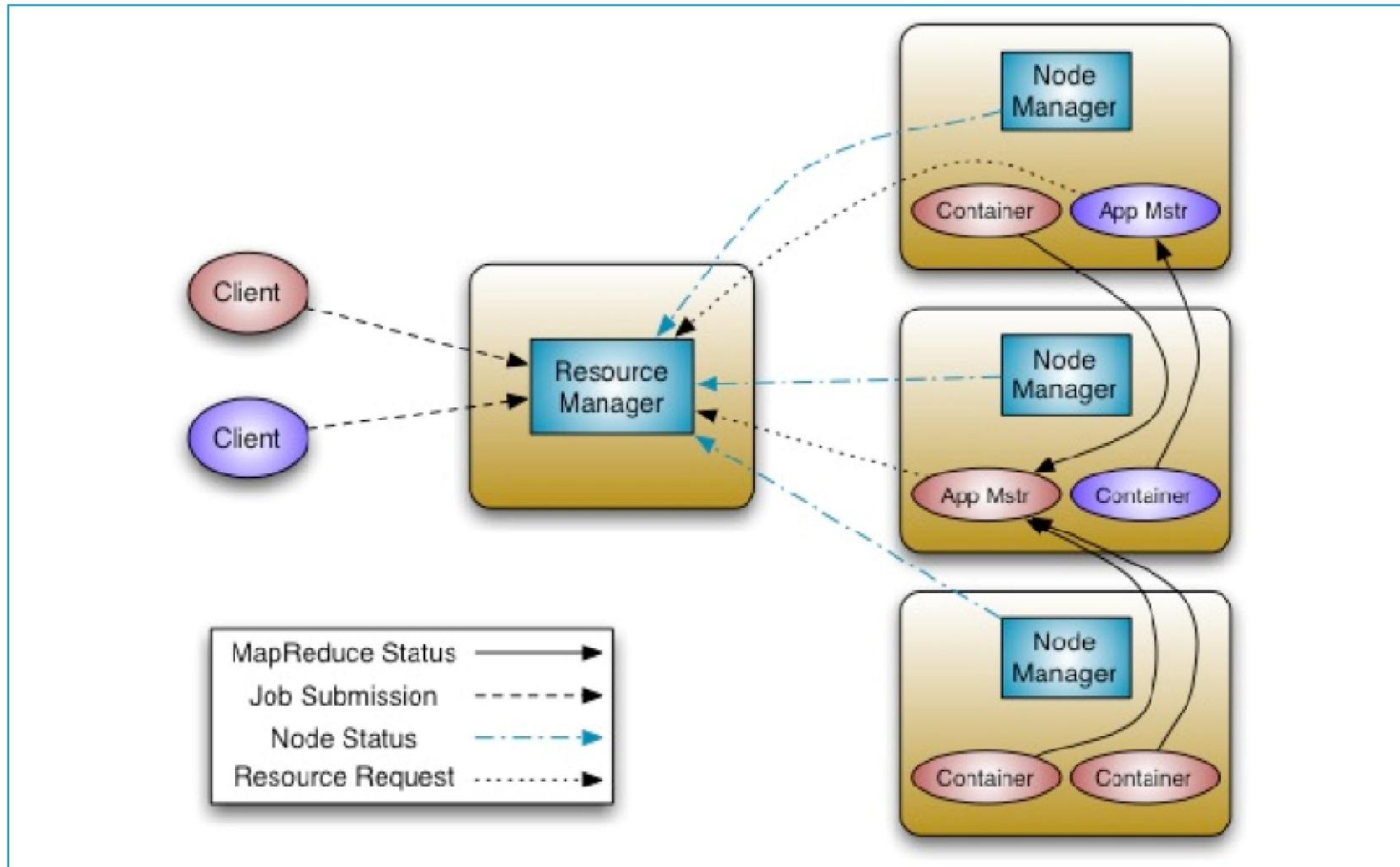
Node Manager

- ▶ One per Data Node
- ▶ Monitors Resources on Data Node

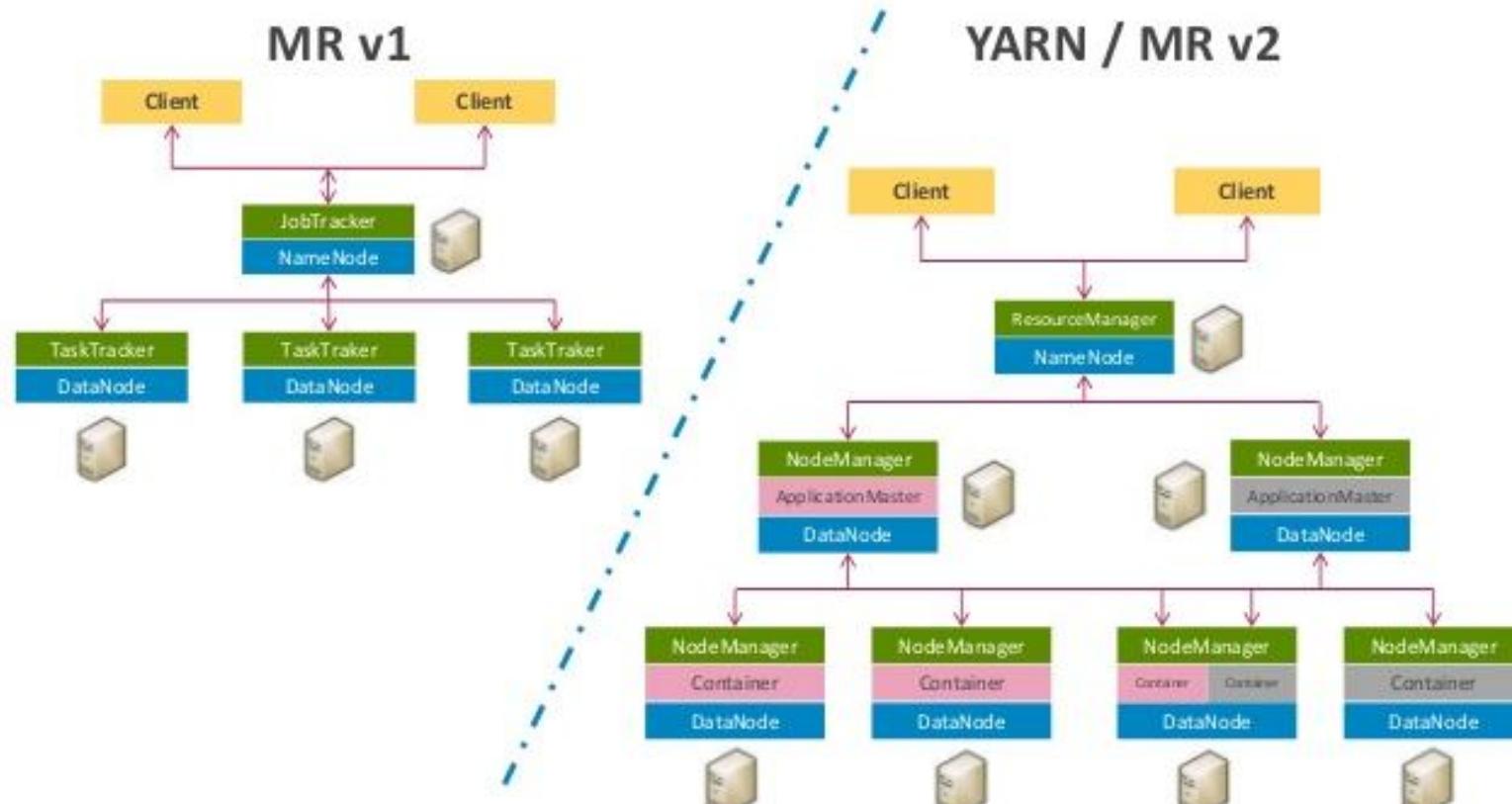
Application Master

- ▶ One per application
- ▶ Short life
- ▶ Manages task/scheduling

# YARN Architecture

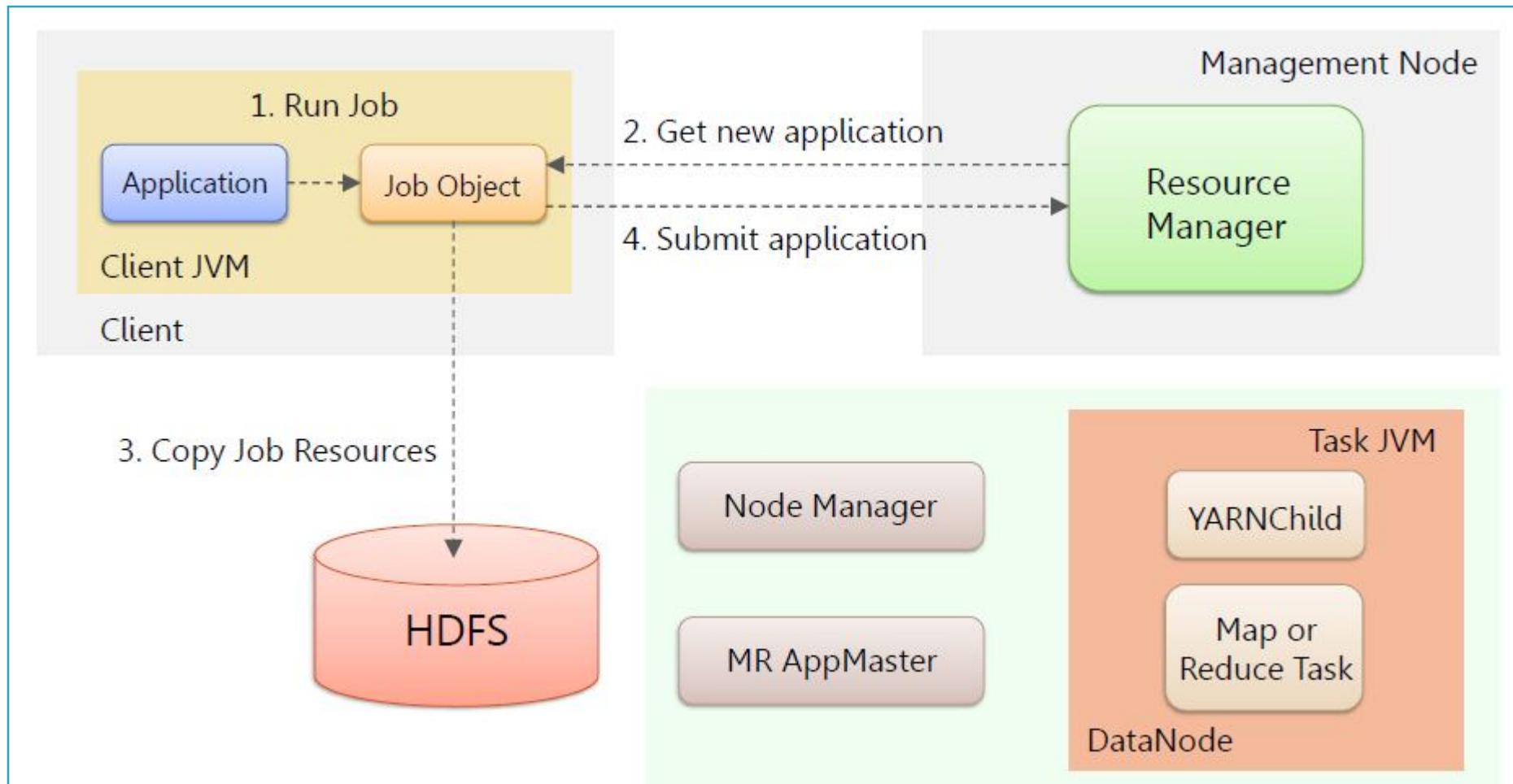


# MR1 vs YARN/MR2 Architecture

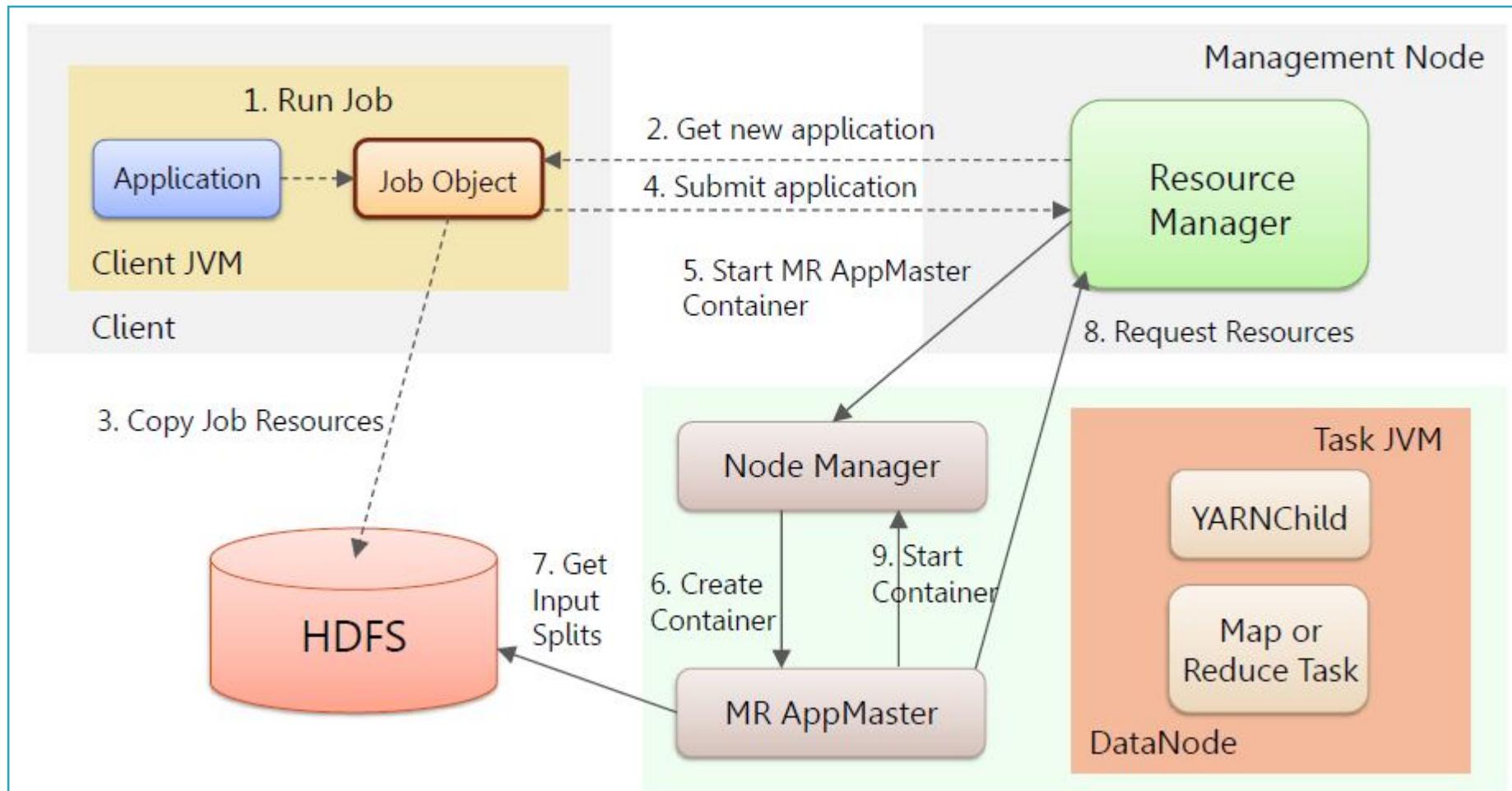


- YARN : Yet Another Resource Negotiator
- MR : MapReduce

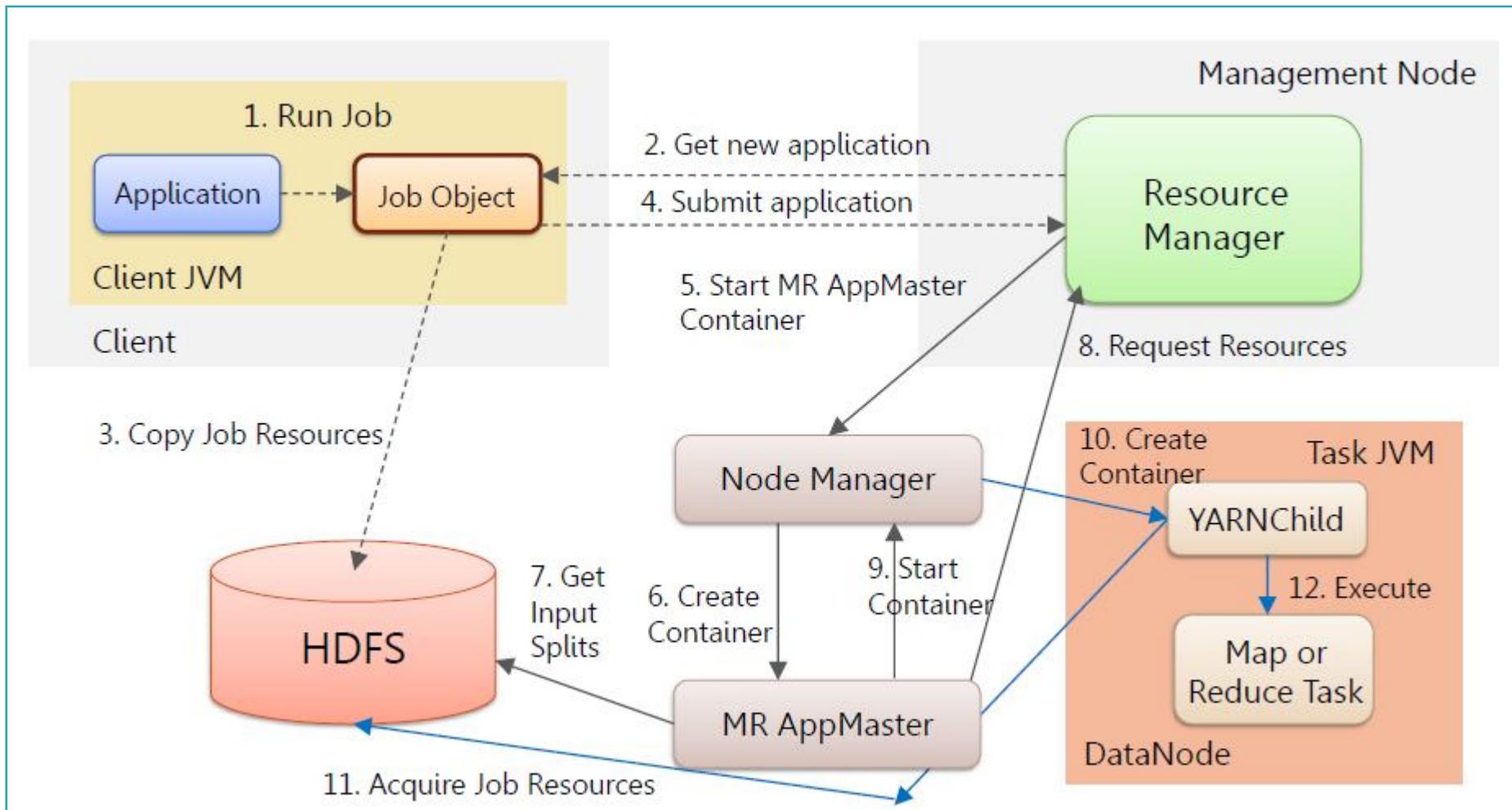
# MR Job Execution YARN Flow



# MR Job Execution YARN Flow

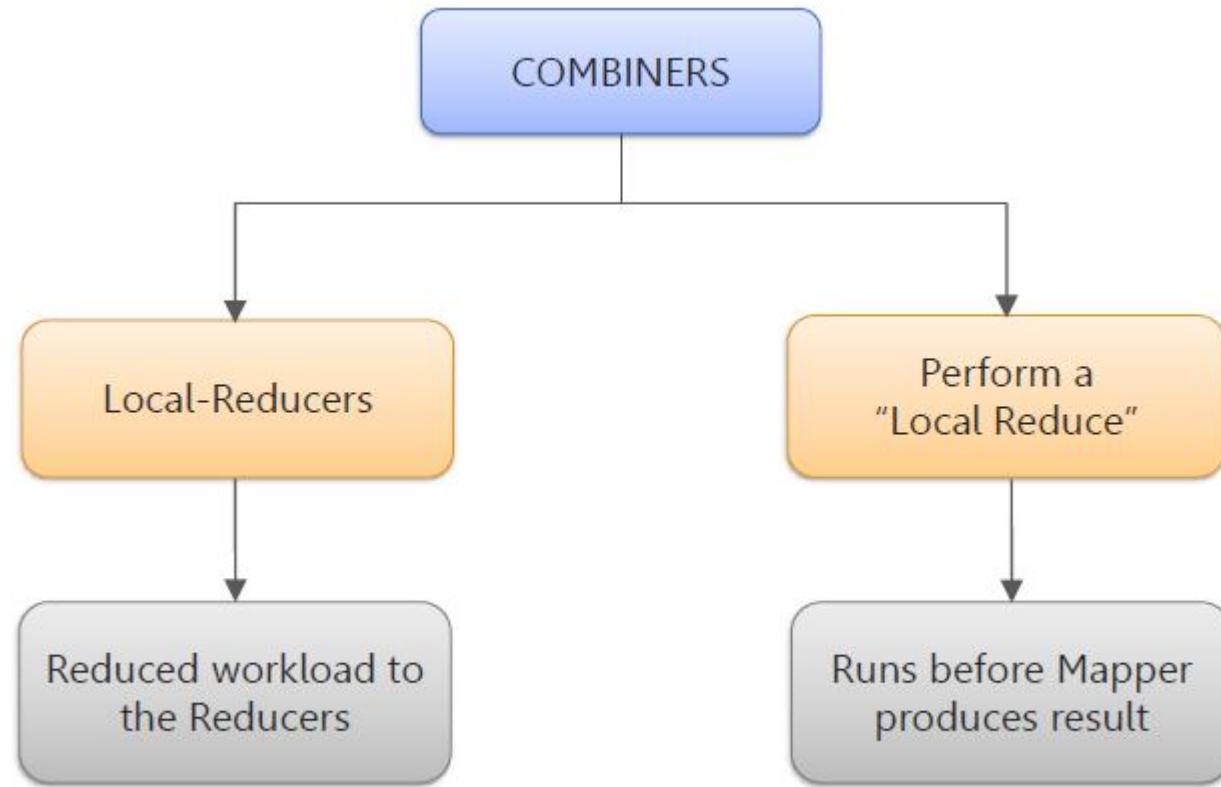


# MR Job Execution YARN Flow

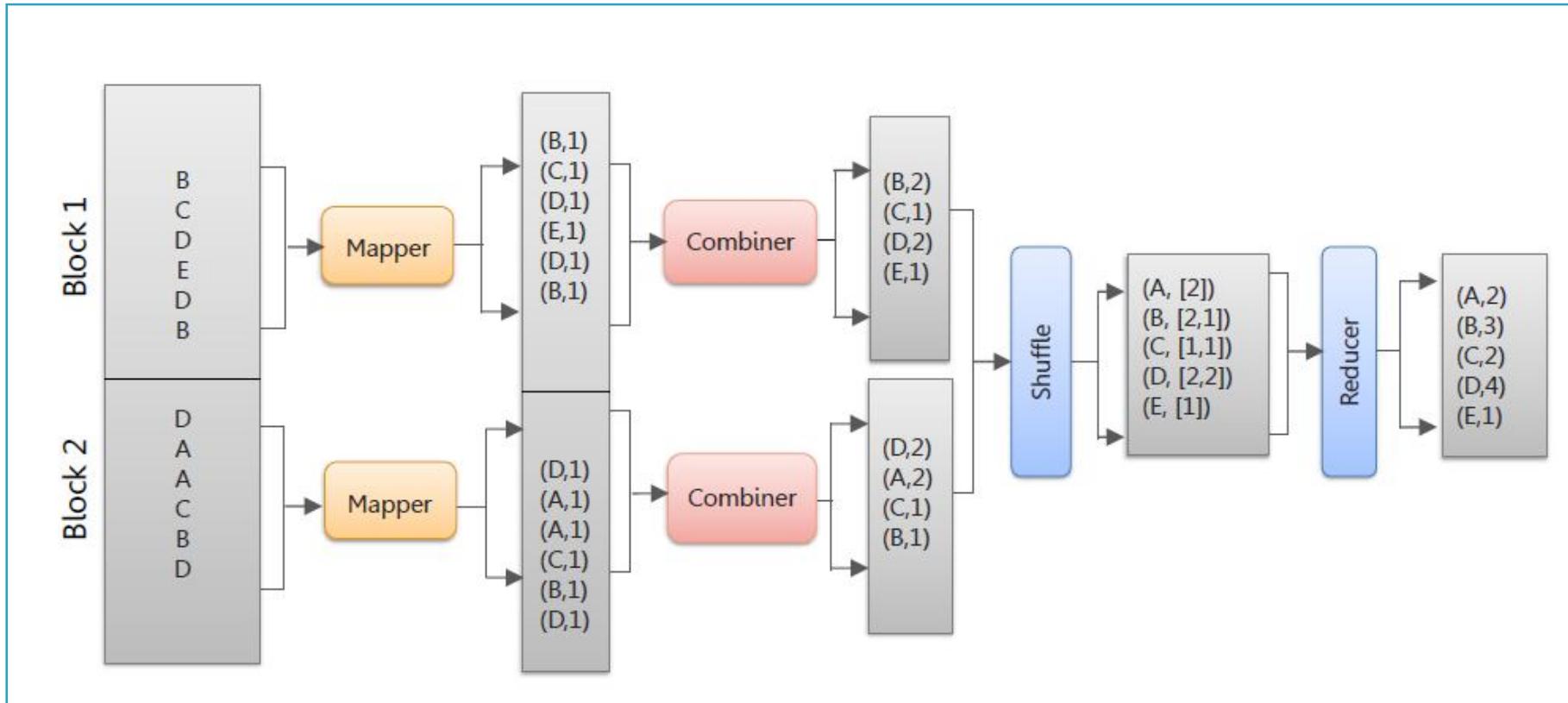


# MR Advanced Concepts

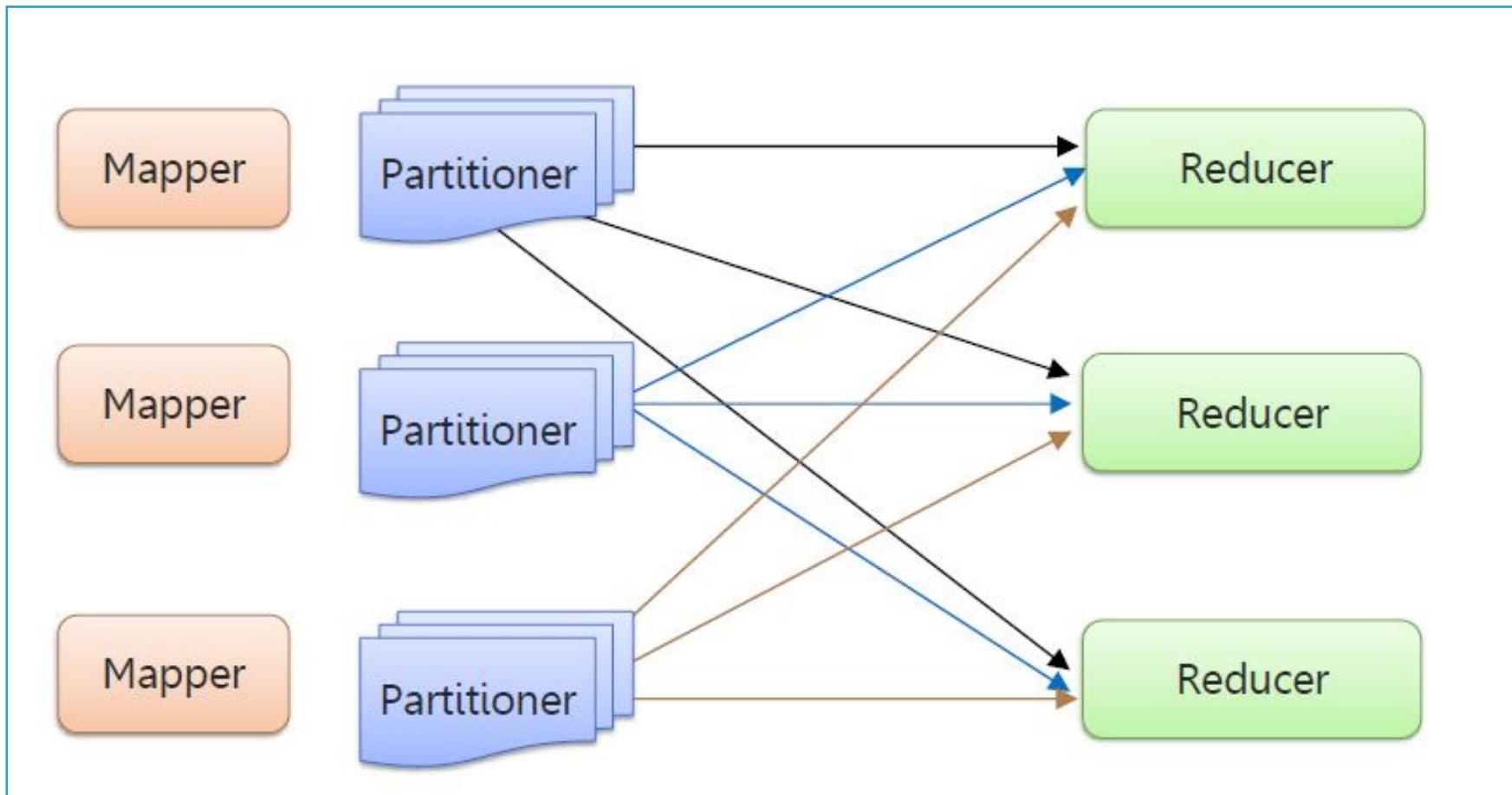
# Combiner



# Combiner in Action

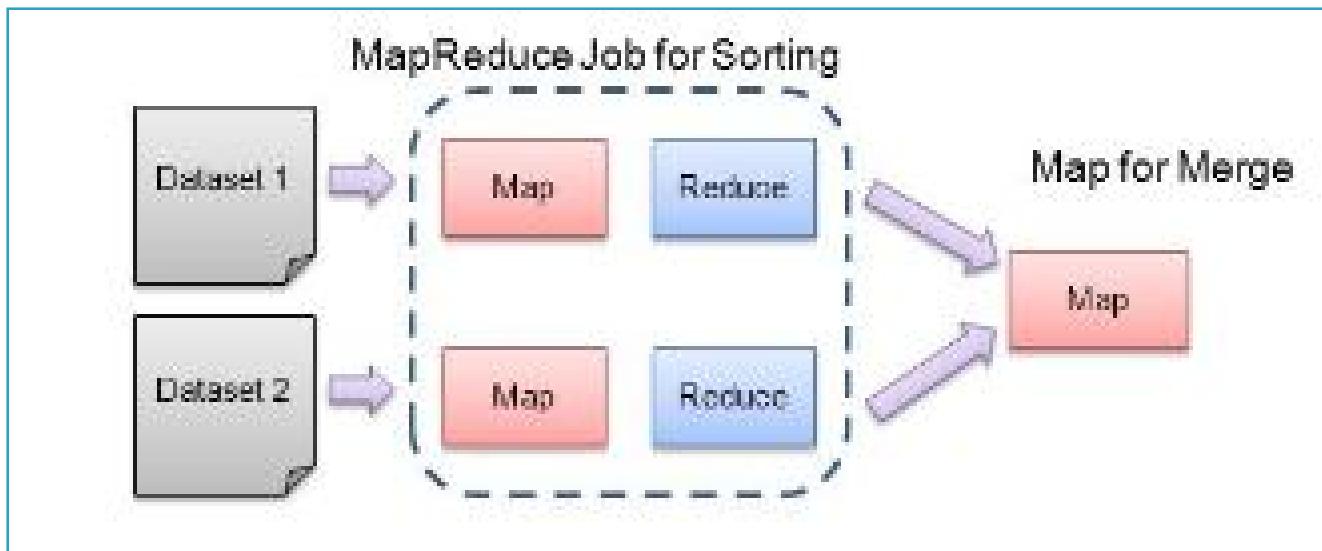


# Partitioner

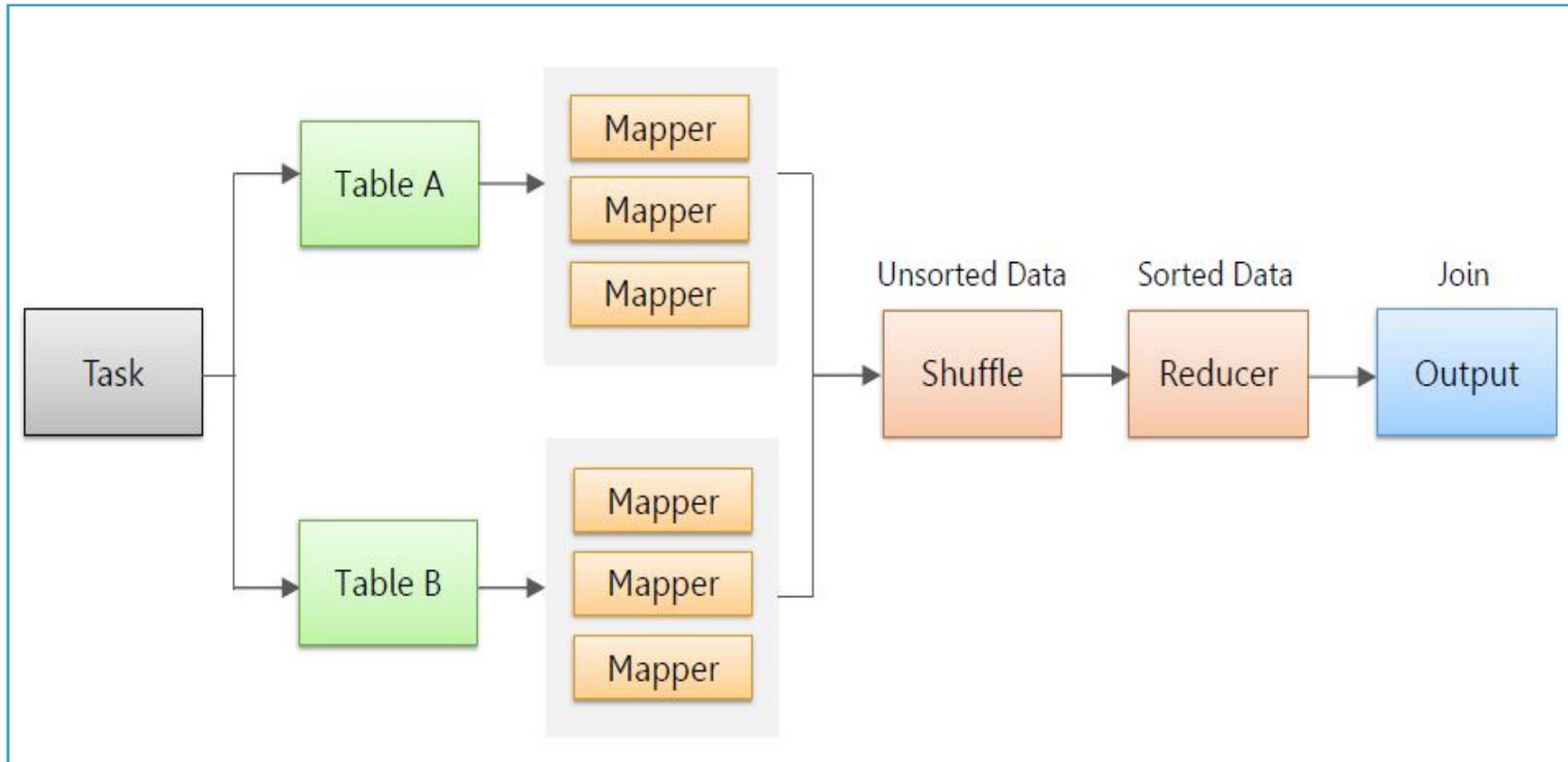


# Map Side Join

- A map-side join works by performing the join before the data reaches the map function
- Requirements
  - Each input dataset must be divided into the same number of partitions
  - It must be sorted by the same key (the join key) in each source
  - All the records for a particular key must reside in the same partition
- Above requirements actually fit the description of the output of a MapReduce job
  - A map-side join can be used to join the outputs of several jobs that had the same number of reducers, the same keys, and output files that are not splittable



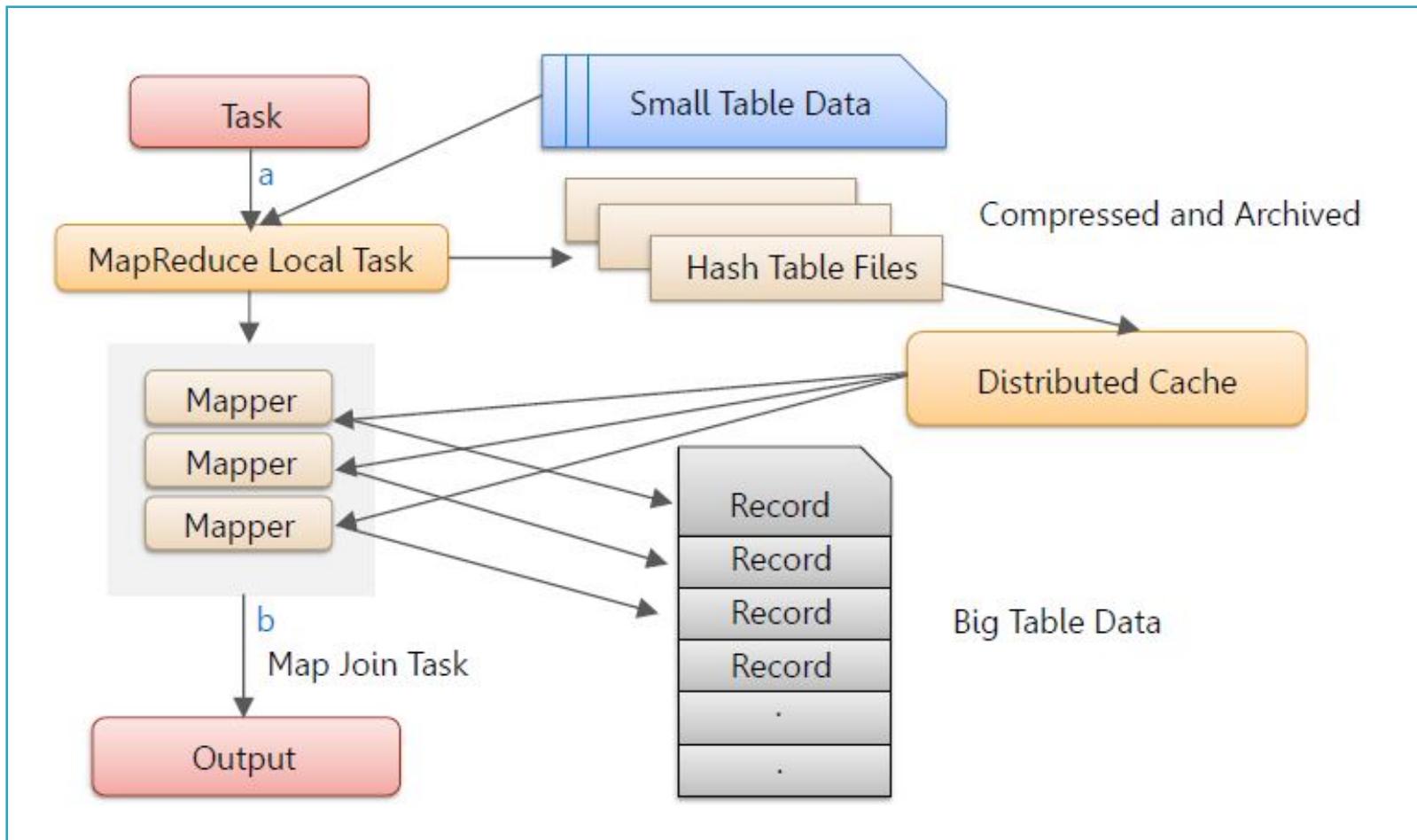
# Reduce Side Join



# Distributed Cache

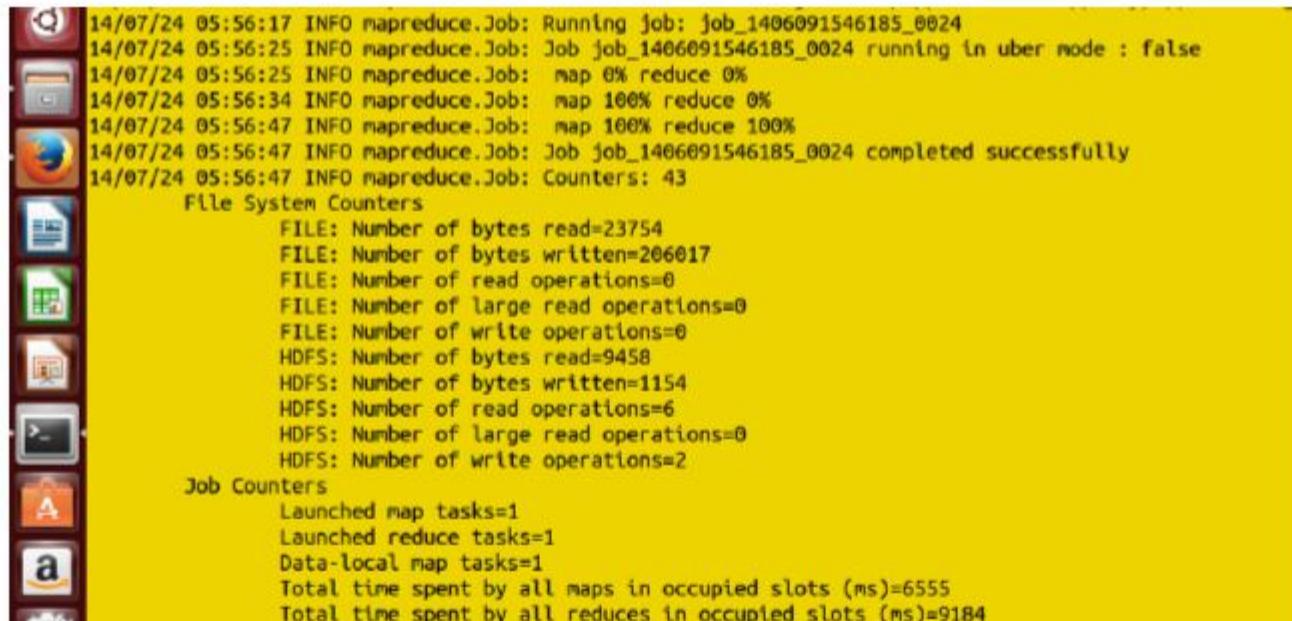
- ▶ At times, we might need to refer a relatively smaller amount of data in Map Reduce jobs  
Examples: Master Files, Lookup files, Cross Reference Files etc.
- ▶ Map Reduce provides Distributed Cache facility. This allows to cache files (text, archives, jars etc.) needed by applications
- ▶ Files are copied only once per job and should not be modified by the application or externally while the job is executing
- ▶ Both Mappers and reducers can access these files
- ▶ DistributedCache can be used to distribute simple, read-only data/text files and/or more complex types such as archives, jars etc via the JobConf
- ▶ Programmers need to be careful to not use too many cache files
- ▶ The ideal size for distributed cache file is <= 100 to 150 MB

# Distributed Cache (contd.)



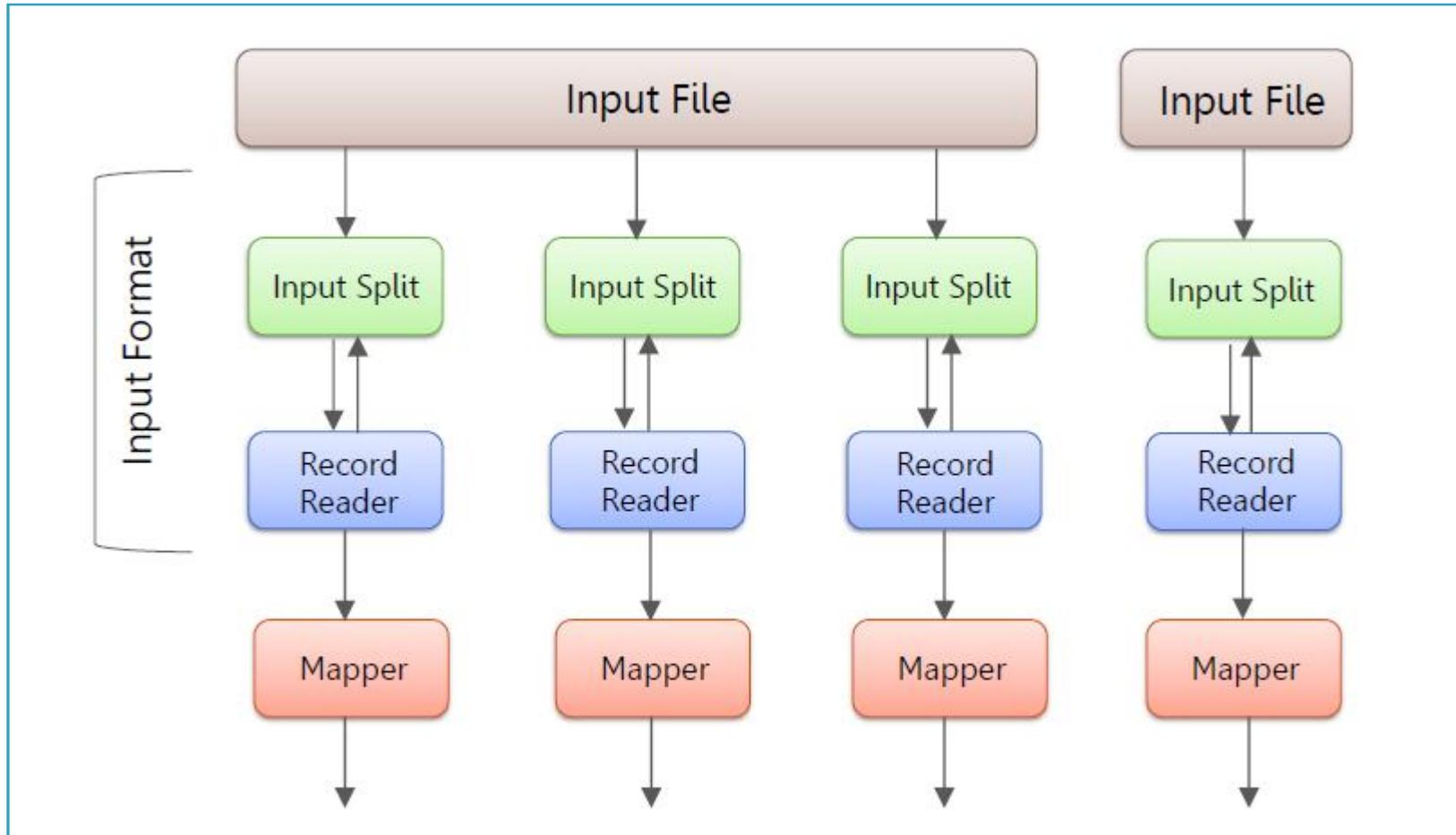
# Counters

- ▶ **Counters** are lightweight objects in Hadoop that allow you to keep track of system progress in both the map and reduce stages of processing
- ▶ Counters are used to gather information about the data we are analysing, like how many types of records were processed, how many invalid records were found while running the job, and so on

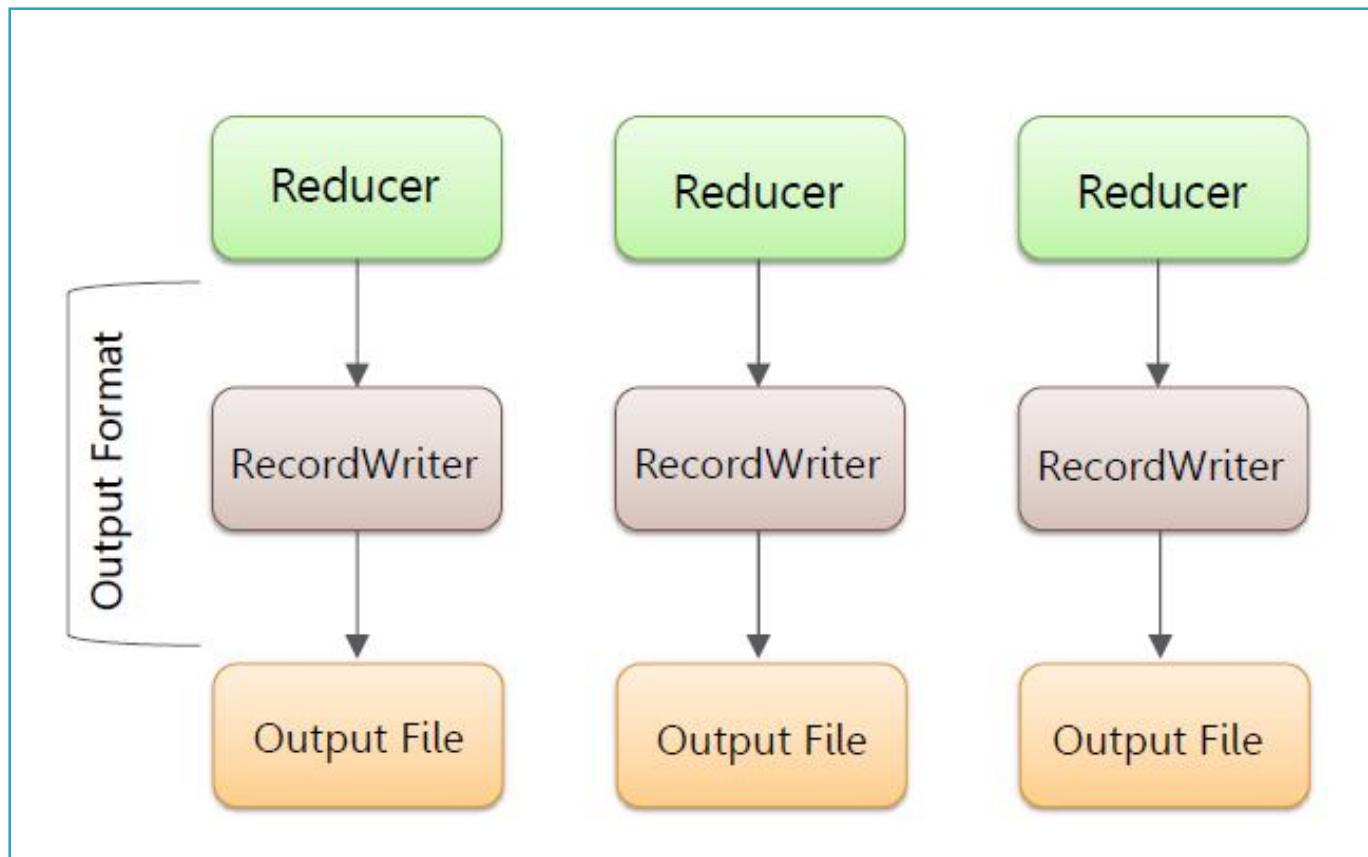


14/07/24 05:56:17 INFO mapreduce.Job: Running job: job\_1406091546185\_0024  
14/07/24 05:56:25 INFO mapreduce.Job: Job job\_1406091546185\_0024 running in uber mode : false  
14/07/24 05:56:25 INFO mapreduce.Job: map 0% reduce 0%  
14/07/24 05:56:34 INFO mapreduce.Job: map 100% reduce 0%  
14/07/24 05:56:47 INFO mapreduce.Job: map 100% reduce 100%  
14/07/24 05:56:47 INFO mapreduce.Job: Job job\_1406091546185\_0024 completed successfully  
14/07/24 05:56:47 INFO mapreduce.Job: Counters: 43  
File System Counters  
FILE: Number of bytes read=23754  
FILE: Number of bytes written=206017  
FILE: Number of read operations=0  
FILE: Number of large read operations=0  
FILE: Number of write operations=0  
HDFS: Number of bytes read=9458  
HDFS: Number of bytes written=1154  
HDFS: Number of read operations=6  
HDFS: Number of large read operations=0  
HDFS: Number of write operations=2  
Job Counters  
Launched map tasks=1  
Launched reduce tasks=1  
Data-local map tasks=1  
Total time spent by all maps in occupied slots (ms)=6555  
Total time spent by all reduces in occupied slots (ms)=9184

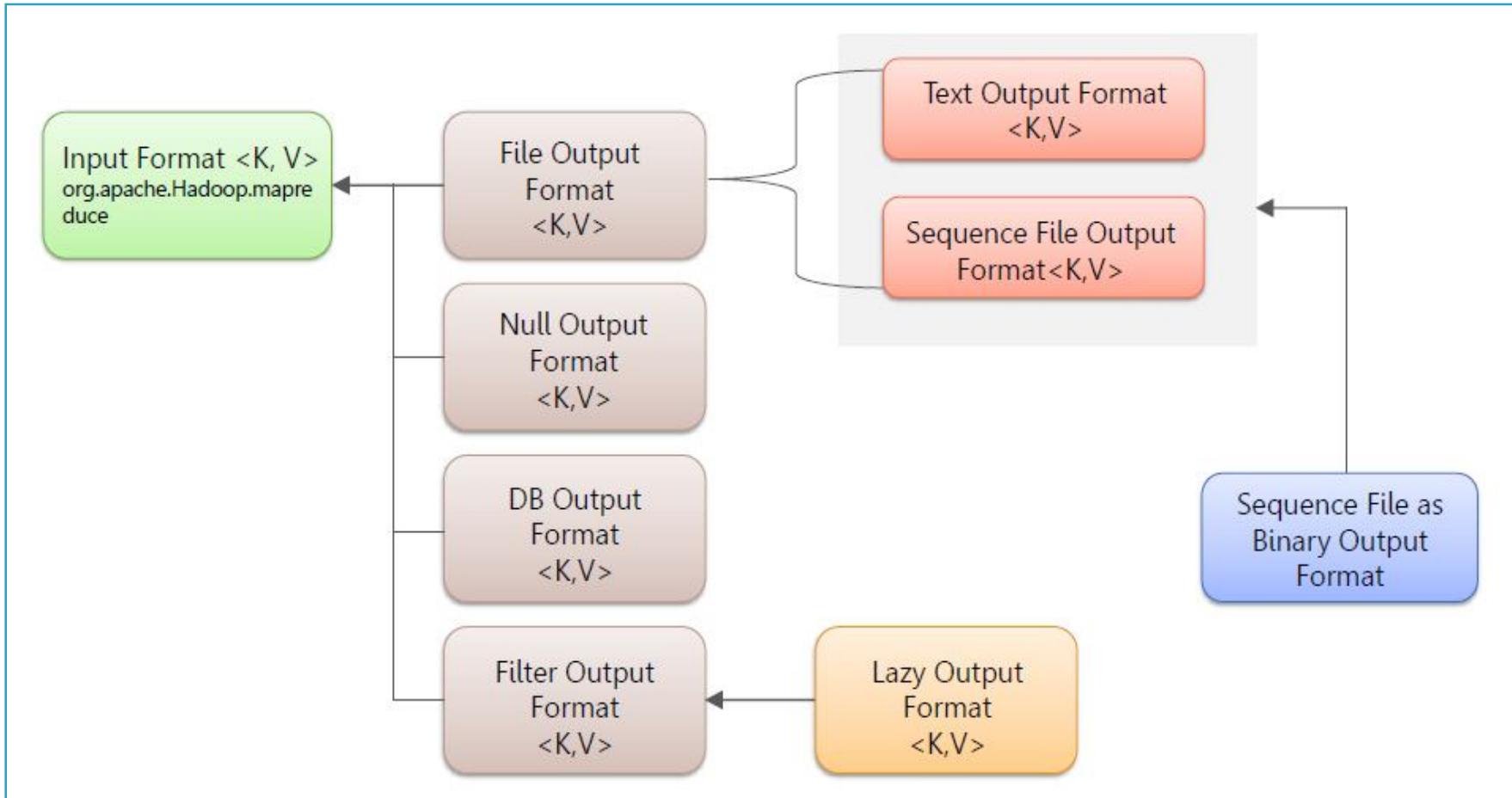
# Hadoop Input Formats



# Hadoop Output Formats



# Hadoop Output Formats – Class Hierarchy



# HDFS File Formats – Sequence File

## In HDFS

- SequenceFile is one of the solutions to small file problem in Hadoop.
- Small file is significantly smaller than the HDFS block size(128MB).
- Each file, directory, block in HDFS is represented as object and occupies 150 bytes.
- 10 million files, would use about 3 gigabytes of memory of NameNode.
- A billion files is not feasible.

## In MapReduce

- Map tasks usually process a block of input at a time (using the default FileInputFormat).
- The more the number of files is, the more number of Map task need and the job time can be much more slower.

## Small file scenarios

- The files are pieces of a larger logical file.
- The files are inherently small, for example, images.

# HDFS File Formats – Sequence File

- The concept of SequenceFile is to put each small file to a larger single file.
- For example, suppose there are 10,000 100KB files, then we can write a program to put them into a single SequenceFile like below, where you can use filename to be the key and content to be the value.

## SequenceFile File Layout

Data	Key	Value	Key	Value	Key	Value	Key	Value
------	-----	-------	-----	-------	-----	-------	-----	-------

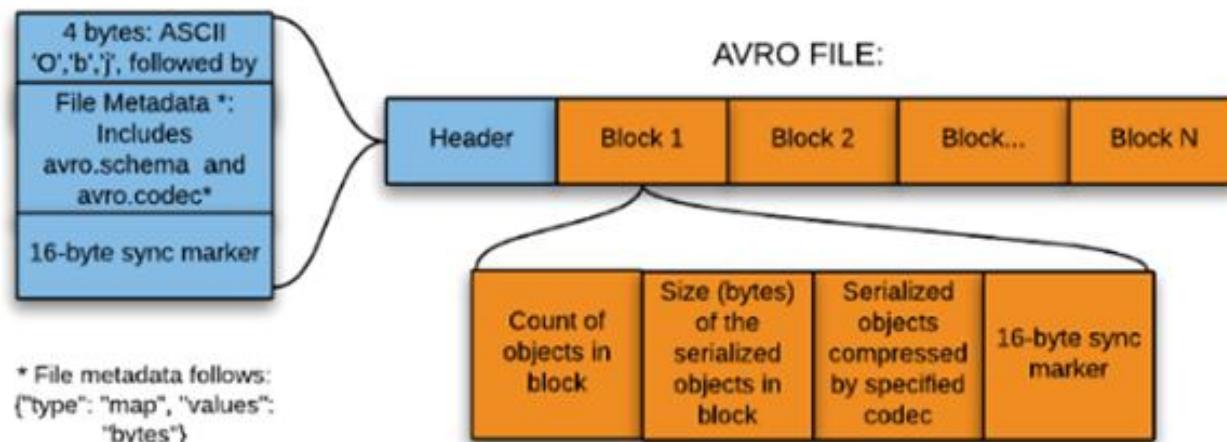
(source: [csdn.net](http://csdn.net))

- Some benefits:
  1. A smaller number of memory needed on NameNode. Continue with the 10,000 100KB files example,
    - Before using SequenceFile, 10,000 objects occupy about 4.5MB of RAM in NameNode.
    - After using SequenceFile, 1GB SequenceFile with 8 HDFS blocks, these objects occupy about 3.6KB of RAM in NameNode.
  2. SequenceFile is splittable, so is suitable for MapReduce.

# HDFS File Formats – Avro

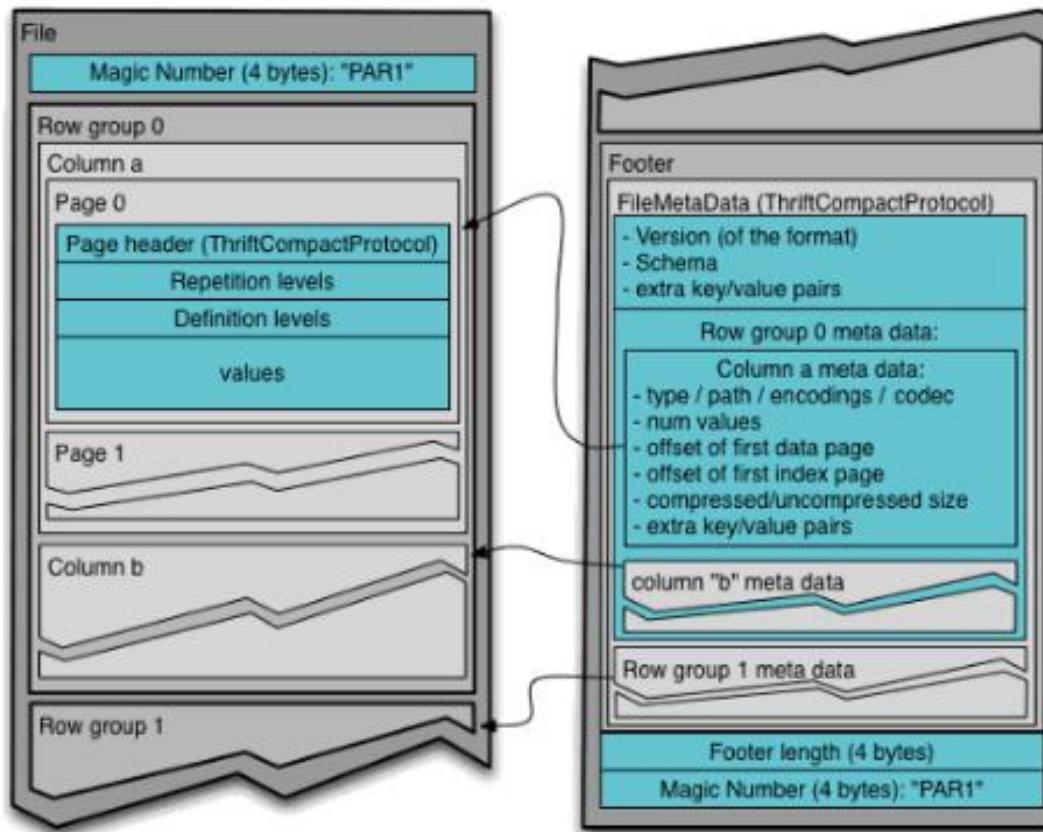
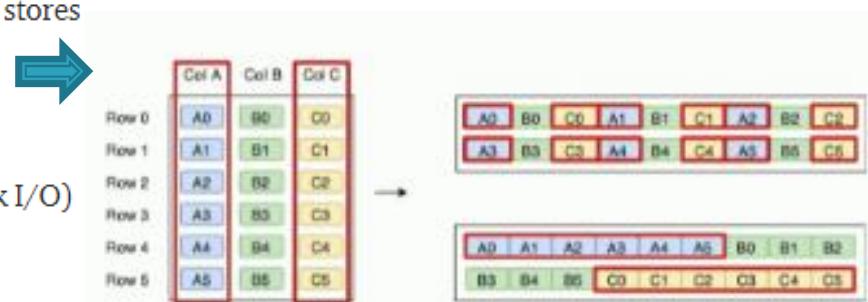
- Row-based (store data in rows): row-based databases are best for write-heavy transactional workloads
- Support serialization
- Fast binary format
- Support block compression and splittable
- Support schema evolution (the use of JSON to describe the data, while using binary format to optimize storage size)
- Stores the schema in the header of file so data is *self-describing*.

```
{  
  "namespace": "customer.avro",  
  "type": "record",  
  "name": "customer",  
  "fields": [  
    {"name": "name", "type": "string"},  
    {"name": "age", "type": "int"},  
    {"name": "address", "type": "string"},  
    {"name": "phone", "type": "string"},  
    {"name": "email", "type": "string"},  
  ]  
}
```



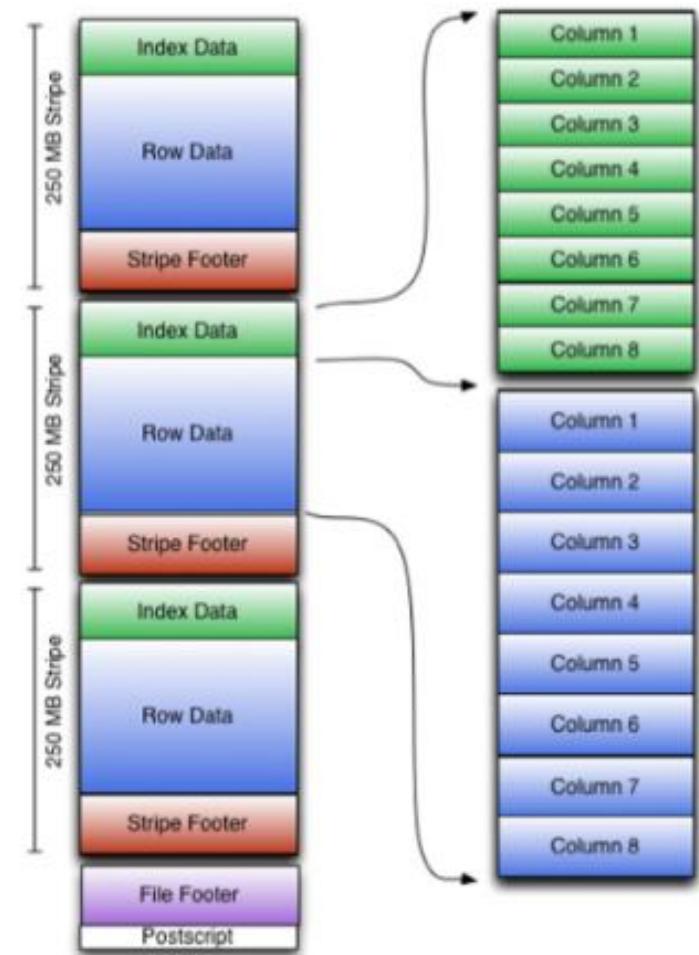
# HDFS File Formats – Parquet

- Column-oriented (store data in columns): column-oriented data stores are optimized for read-heavy analytical workloads
- High compression rates (up to 75% with Snappy compression)
- Only required columns would be fetched/read (reducing the disk I/O)
- Can be read and write using Avro API and Avro Schema
- Support predicate pushdown (reducing disk I/O cost)

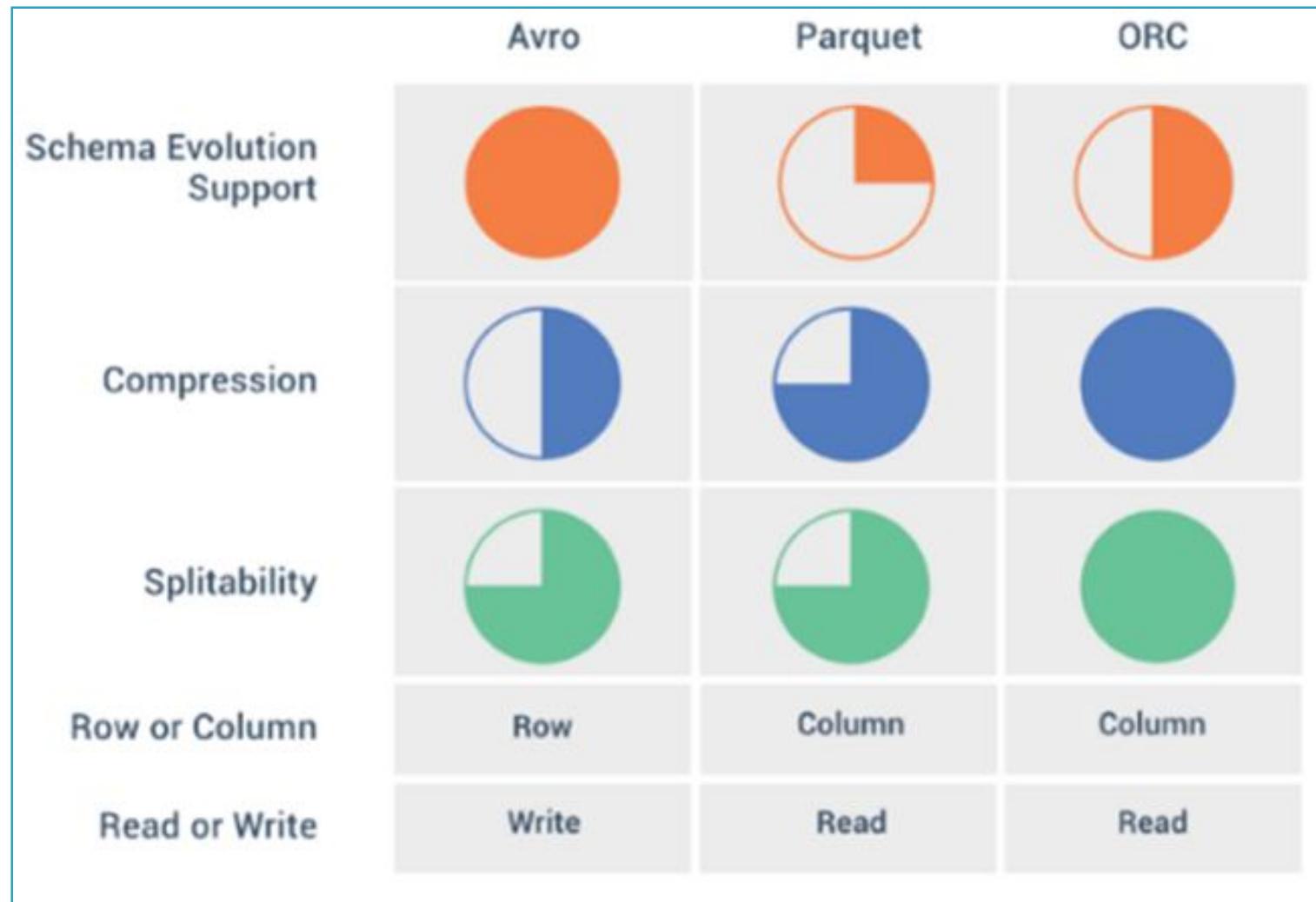


# HDFS File Formats – ORC (Optimized Row Columnar)

- Column-oriented (store data in columns): column-oriented data stores are optimized for read-heavy analytical workloads
- High compression rates (ZLIB)
- Hive type support (datetime, decimal, and the complex types like struct, list, map, and union)
- Metadata stored using Protocol Buffers, which allows addition and removal of fields
- Compatible on HiveQL
- Support serialization



# File Formats Comparison



# Thank You!