LEARN.
CONNECT.
ELEVATE.

YCBS 257 - Data at Scale (Winter 2019)

Instructor: Khaled Tannir

---

McGill | School of Continuing Studies

mcgill.ca
/continuingstudies

School of Continuing Studies

YCBS 257-256 / 257 - Data at Scale (BIG DATA)

# Course 3

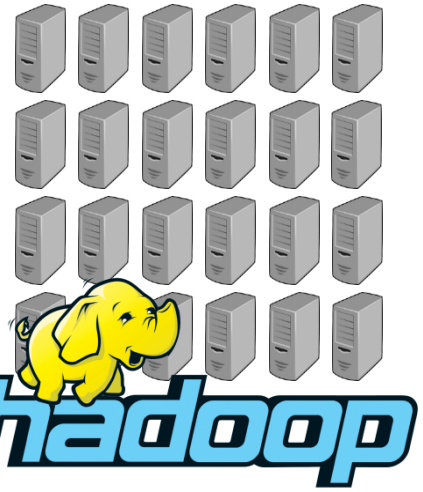
## *Hadoop Core*

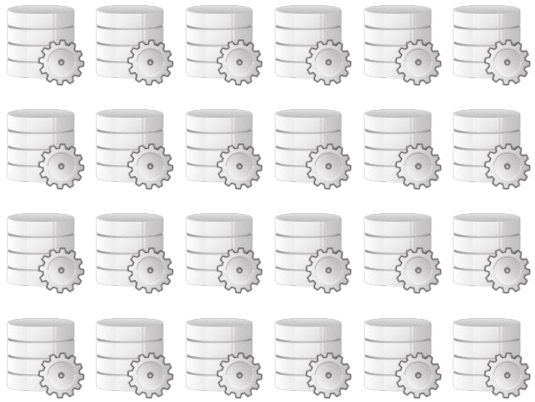
Khaled Tannir

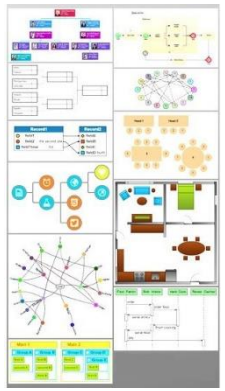
McGill

Montréal – Winter 2019

# Machine Learning at Scale







Analyze

Ingest






# Theme of this Course

## *Hadoop Core*

- *The Hadoop Distributed File System*
  - *Core Concepts*
  - *HDFS Usage*


- *The MapReduce Model*
  - *MapReduce anatomy*
  - *Writing MapReduce jobs (Java)*


# Hadoop HDFS

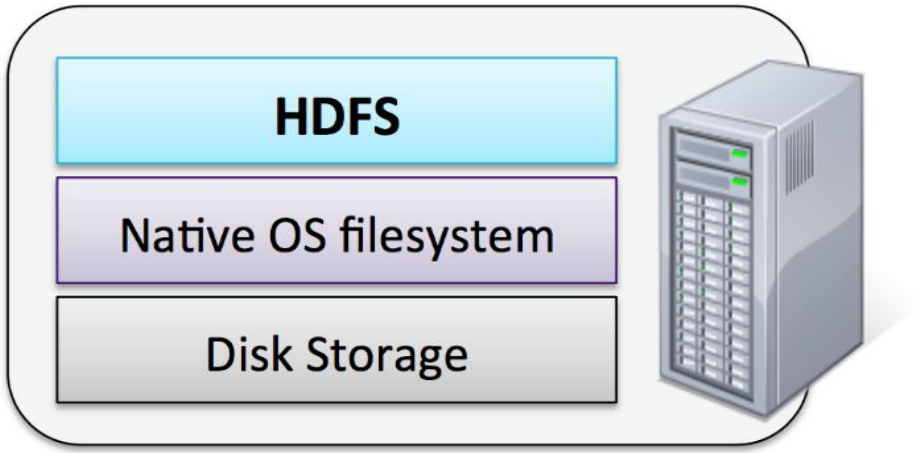
*Diving into the Hadoop core*

McGill

FALL 2018

# Hadoop Distributed File System (*HDFS*)

- HDFS is a the Hadoop Distributed Storage system
- Designed for batch processing to work closely with MapReduce
- HDFS can handle very large data file (TB, PB +)
- Write-Once / Read many access model (Append only)
- Sequential reading pattern
- Written in Java (part of Hadoop)
- Runs on commodity hardware



# HDFS Origin

- Based on *Google File System (GFS)*
- Developed by *Doug Cutting* and *Mike Cafarella* to support Nutch, open source web search engine
- Many abstractions build over HDFS for specific cases, like Hive, HBase, etc.
- Hundreds of companies use it today, including Facebook, Yahoo, Netflix, Twitter, Amazon, etc.

# HDFS Features

- **HDFS is Distributed**

  *Scale of data growing at higher pace than single storage disk capacity growth.*

- **HDFS is Scalable**

  *Extends to handle growing data requirement.*

- **HDFS is Reliable**

  *Safeguards against data lose, achieved by data replication.*

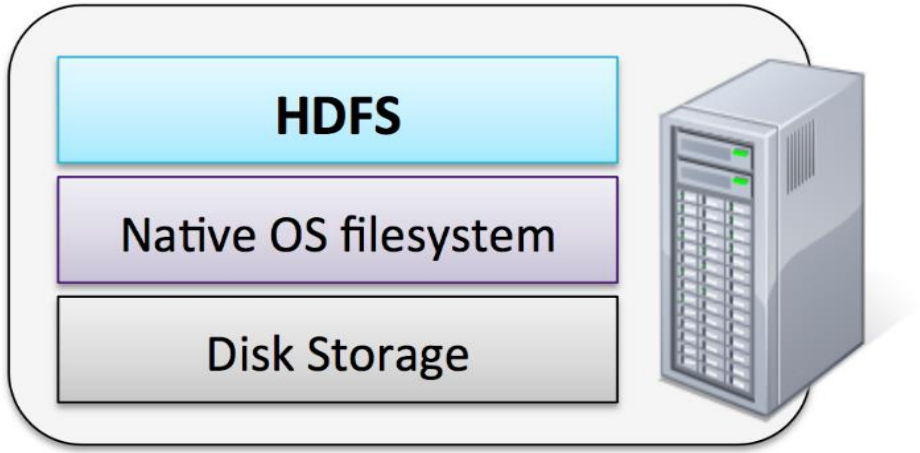
- **HDFS is Fault-Tolerant**

  *Protects against increased failure probability due to large number of disks.*

# HDFS Data Organization

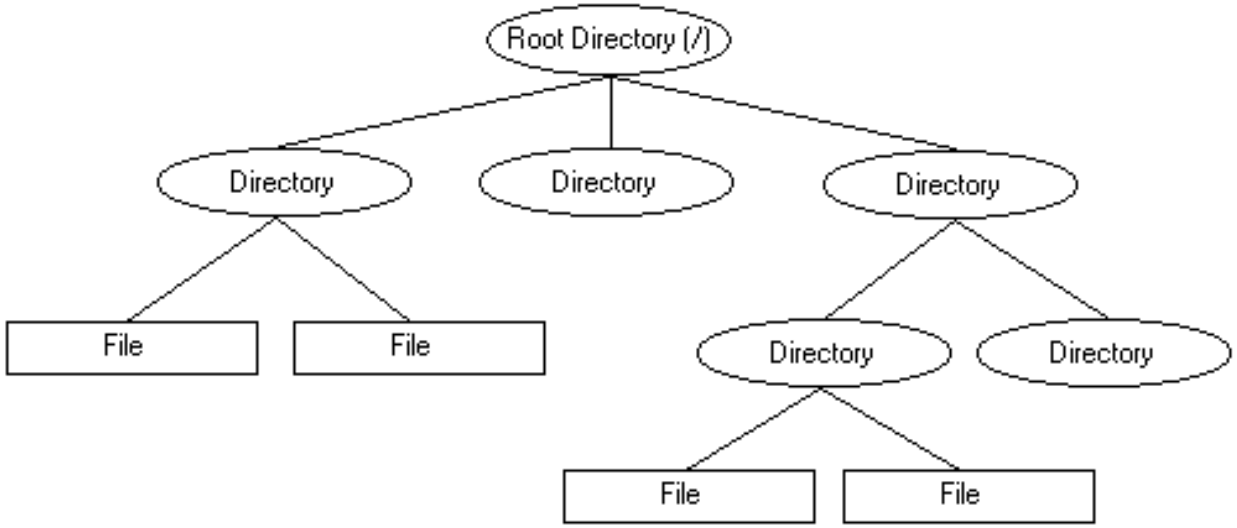
- **Files are divided into uniform sized data blocks *(legacy default 64 MB, new default 128 MB)* and distributed across cluster nodes**


- **Patterned after the UNIX file system**
  - Not fully POSIX compliant
  - Standards sacrificed for performance, space-efficiency and fault-tolerance



# HDFS Data Organization

- Data is organized into files and directories
- HDFS exposes block placement so that computation can be migrated to data
- Data is replicated to prevent hardware failure (default = 3)



# HDFS Main Components

- ***NameNode***
  - *Master of the system*
  - *Maintains and manages the blocks which are present on the **DataNodes***
- ***DataNodes***
  - *Slaves which are deployed on each machine and provide the actual storage*
  - *Responsible for serving read and write requests for the clients*
- ***Secondary NameNode***
  - *Take checkpoints of the file system metadata present on **NameNode**. **It is not** a backup NameNode*

# HDFS High Level Architecture

- **Master / Slave paradigm Architecture** *(Metadata Only)*
  - *Master : NameNode – single machine per cluster*
  - *Slaves (or Workers) : DataNodes many machines per cluster (thousands)*

NameNode - Centralized Metadata

DataNodes

- **Multi-threaded system, serves requests from multiple clients simultaneously**
- **Each request is logged in journal**

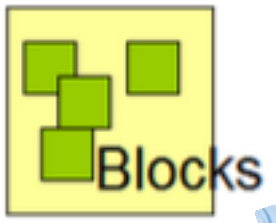
# HDFS Master : NameNode Server

- **Single** NameNode for each cluster.
  - *with a backup (Secondary NameNode)*
- Contains namespace in **RAM**
- Manages filesystem space and files meta-data *(e.g "inodes")*
- Manages checkpoints & journal namespace changes

  *(Checkpoints file: fsimage / Journal of modifications file: fsedits)*

# HDFS Slaves : DataNodes Servers

- **Contain all data blocks and replicas**
- **Handshake with NameNode on startup**
  - *Ensures correct namespace and version*
  - *Automatic shutdown on bad handshake*

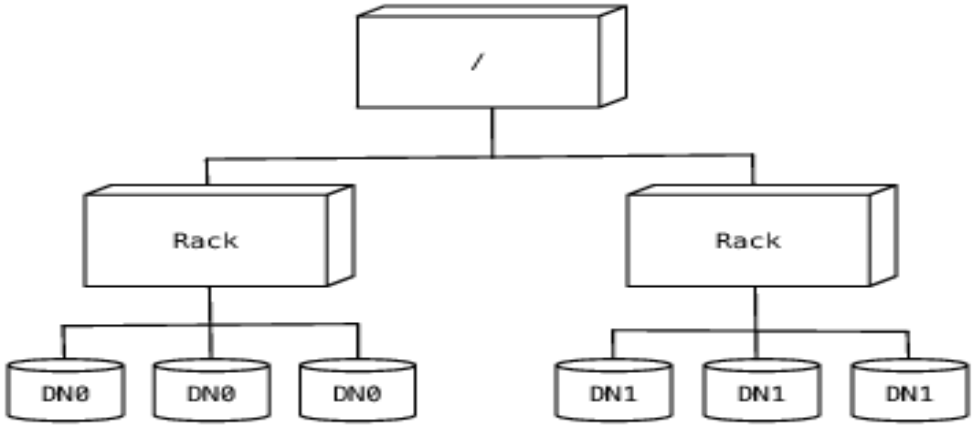



- **Sends a block report upon joining the cluster**
  - *Thereafter sent each hour*
- **Sends heartbeats to NameNode**
  - *Piggybacks stats on heartbeat*
  - *NameNode instruction in heartbeat reply*


# HDFS Slaves : DataNodes Servers

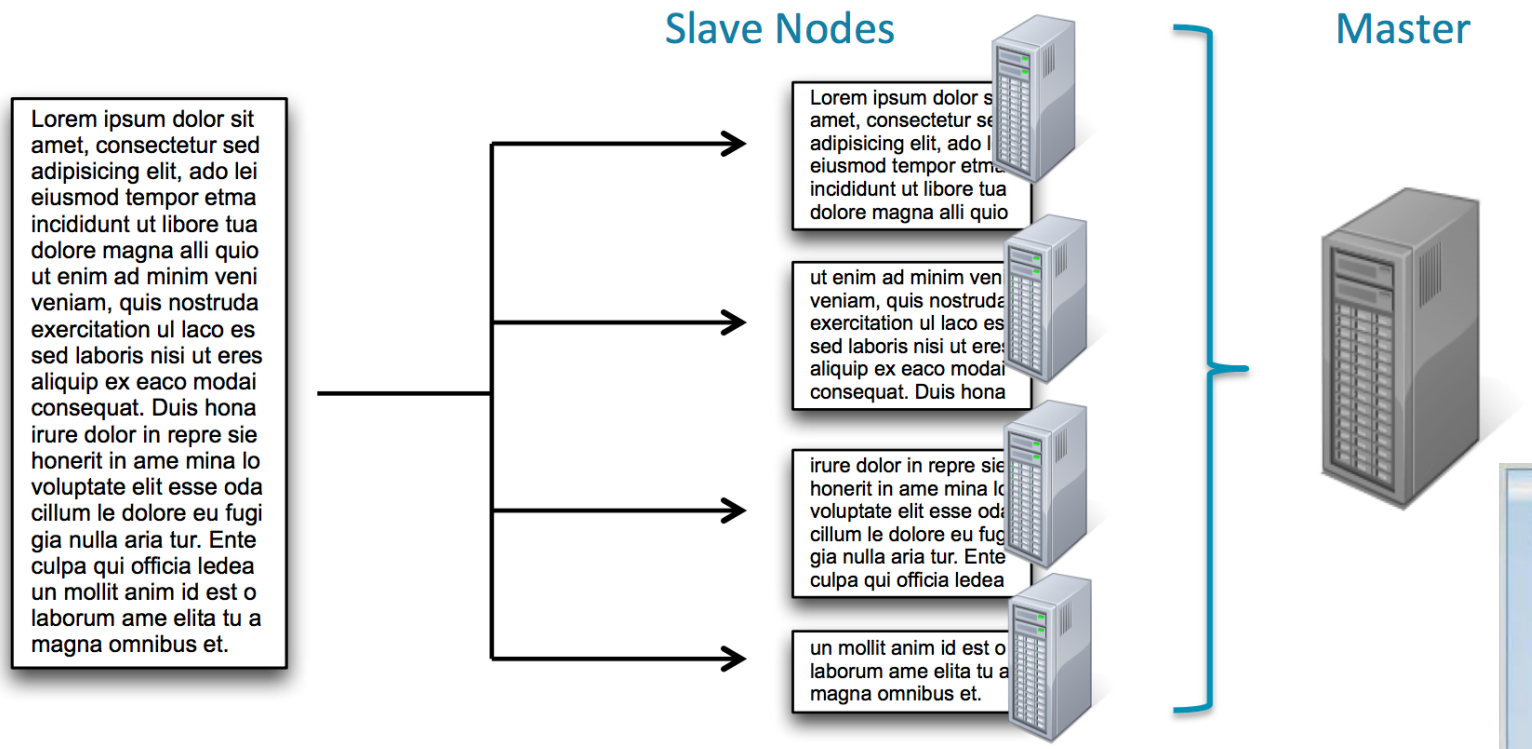
- **Files are stored as collection of data blocks** *(default size - 128MB, legacy default size - 64MB)*
- **Data block is represented by two files :**
  - *one containing data itself,*
  - *other containing metadata like checksum*
- **Blocks are stored underlying OS's files**
- **Clients access the block directly from data nodes**

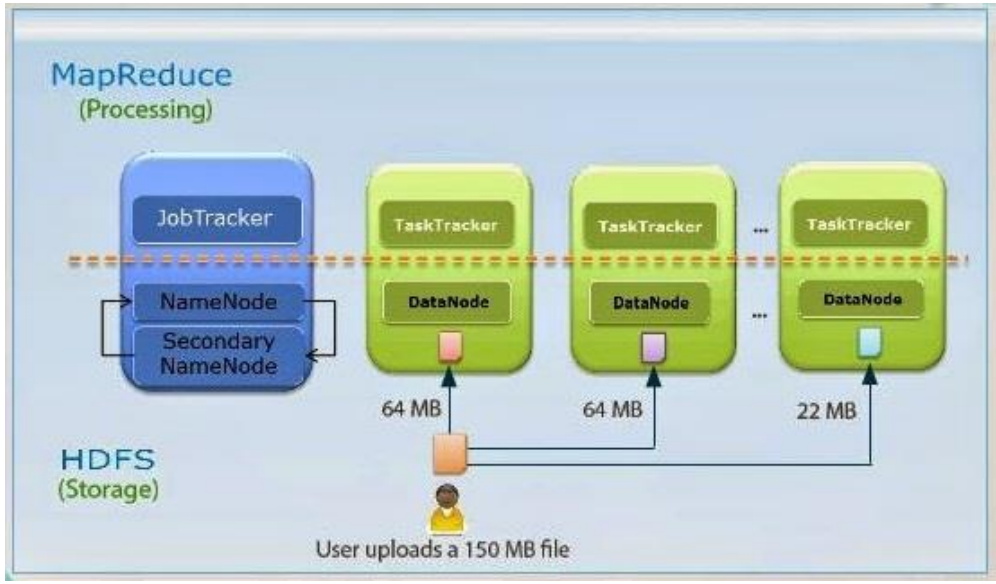
**Spread data across racks**




Hadoop – The Definitive Guide

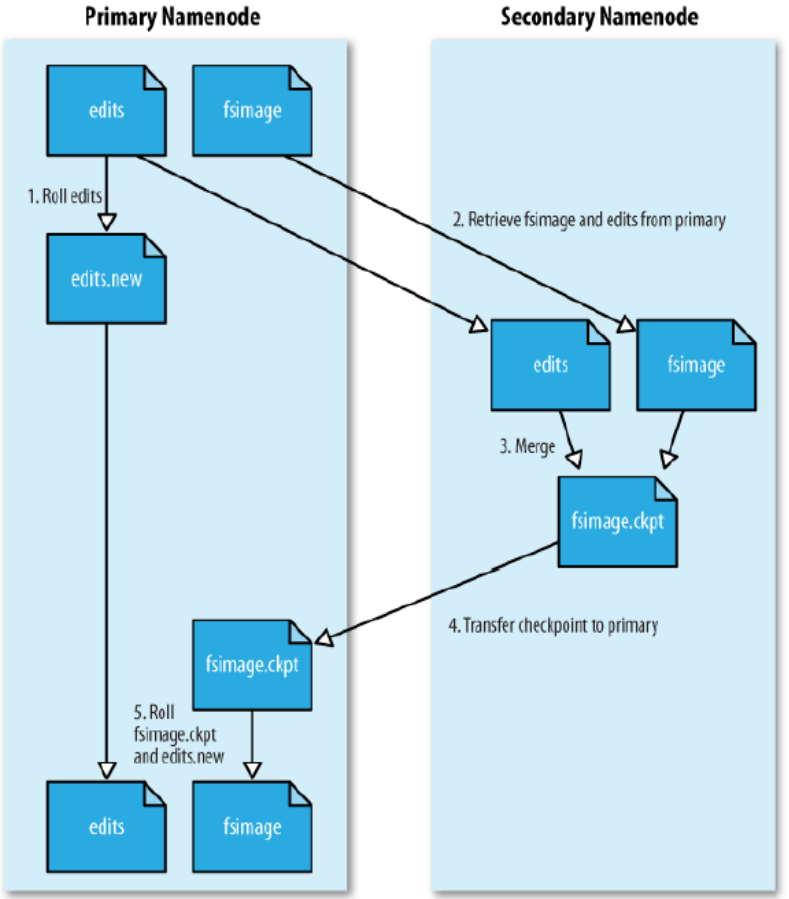
# HDFS : Blocks Storage



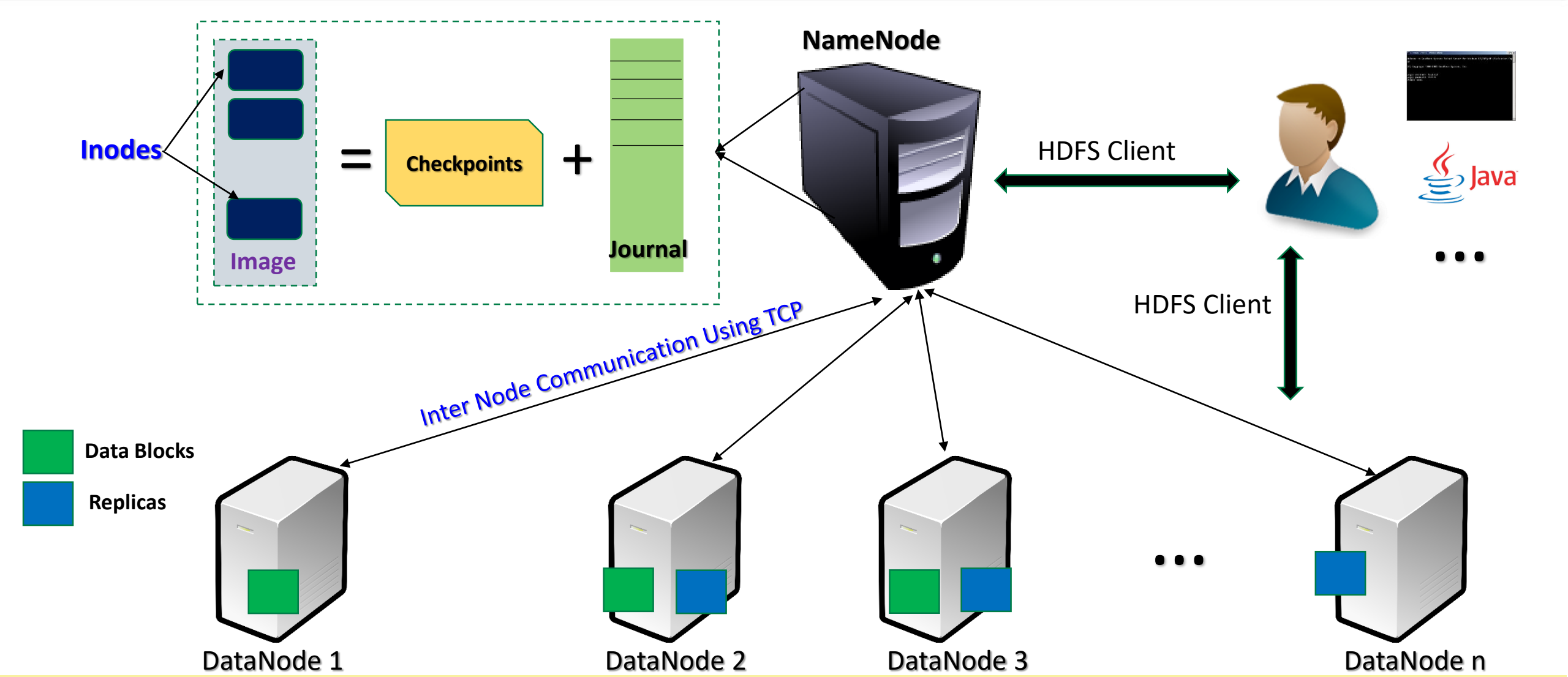
# HDFS Masters : Secondary NameNode

- **Cannot take over for the NameNode**
- **Maintains namespace image from transaction log**
- **Occasionally combine the checkpoint and journal on NameNode to create a new checkpoint and empty journal**
- **Checkpoint is never overwritten, but replaced with newly created one**
- **Used to recover the namenode on reboot**



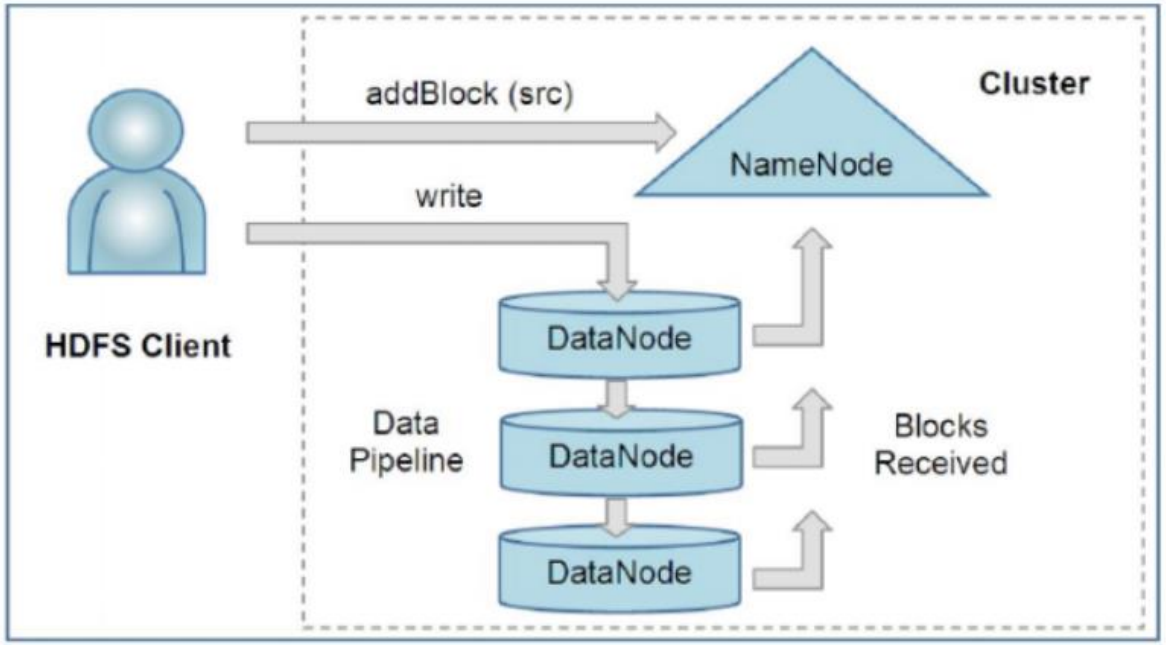

*Journal size can grow very large*

# HDFS Masters : Architecture Overview



# HDFS Client

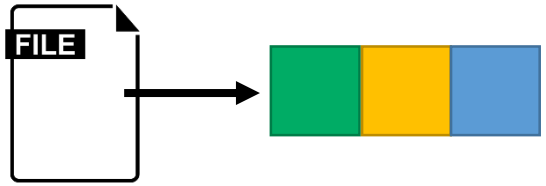
- **Provides API to read/write/delete file:**
  - **READ: Get list of Data Nodes from NameNode in topological sorted order. Then read data directly from Data Nodes.**
  - **WRITE: For each block of data, setup a pipeline of Data Nodes to write to.**

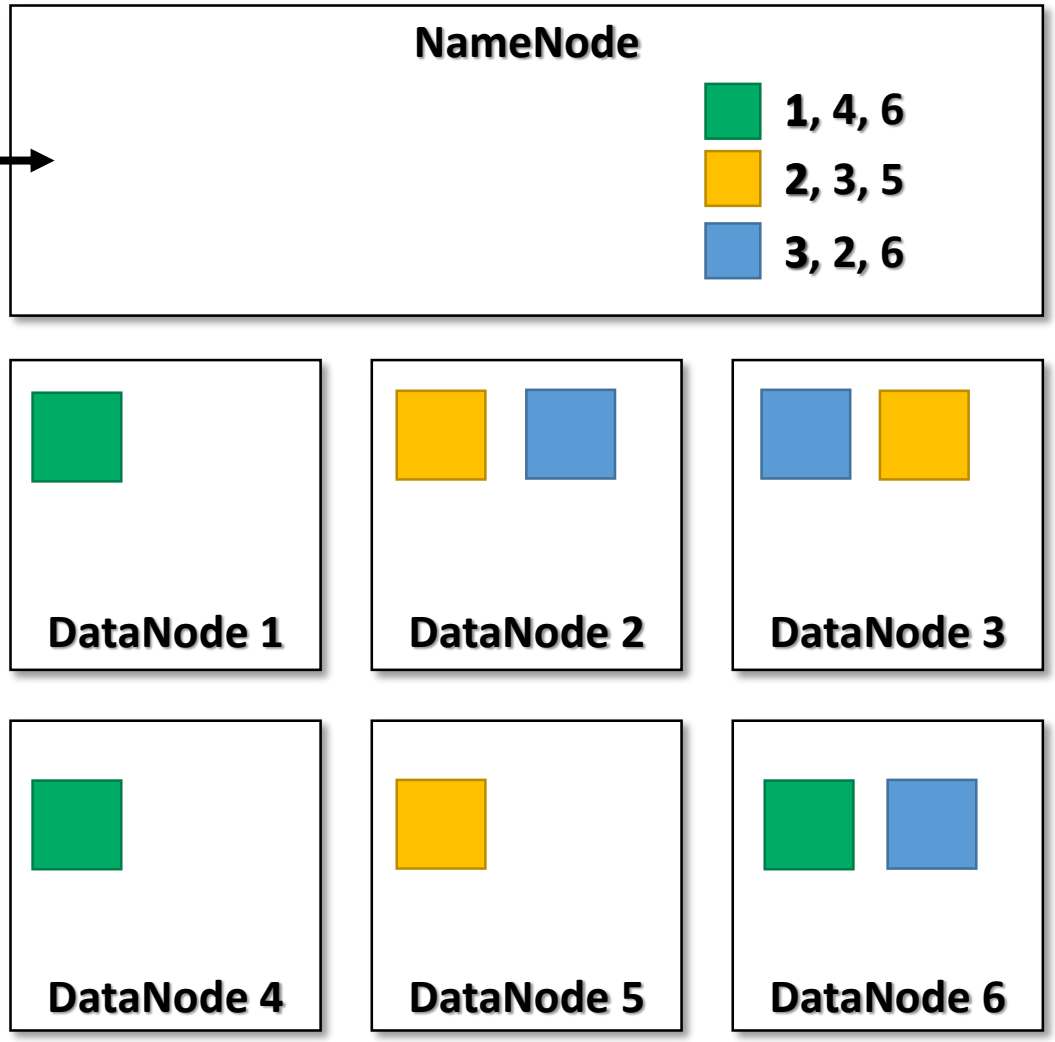



Hadoop – The Definitive Guide

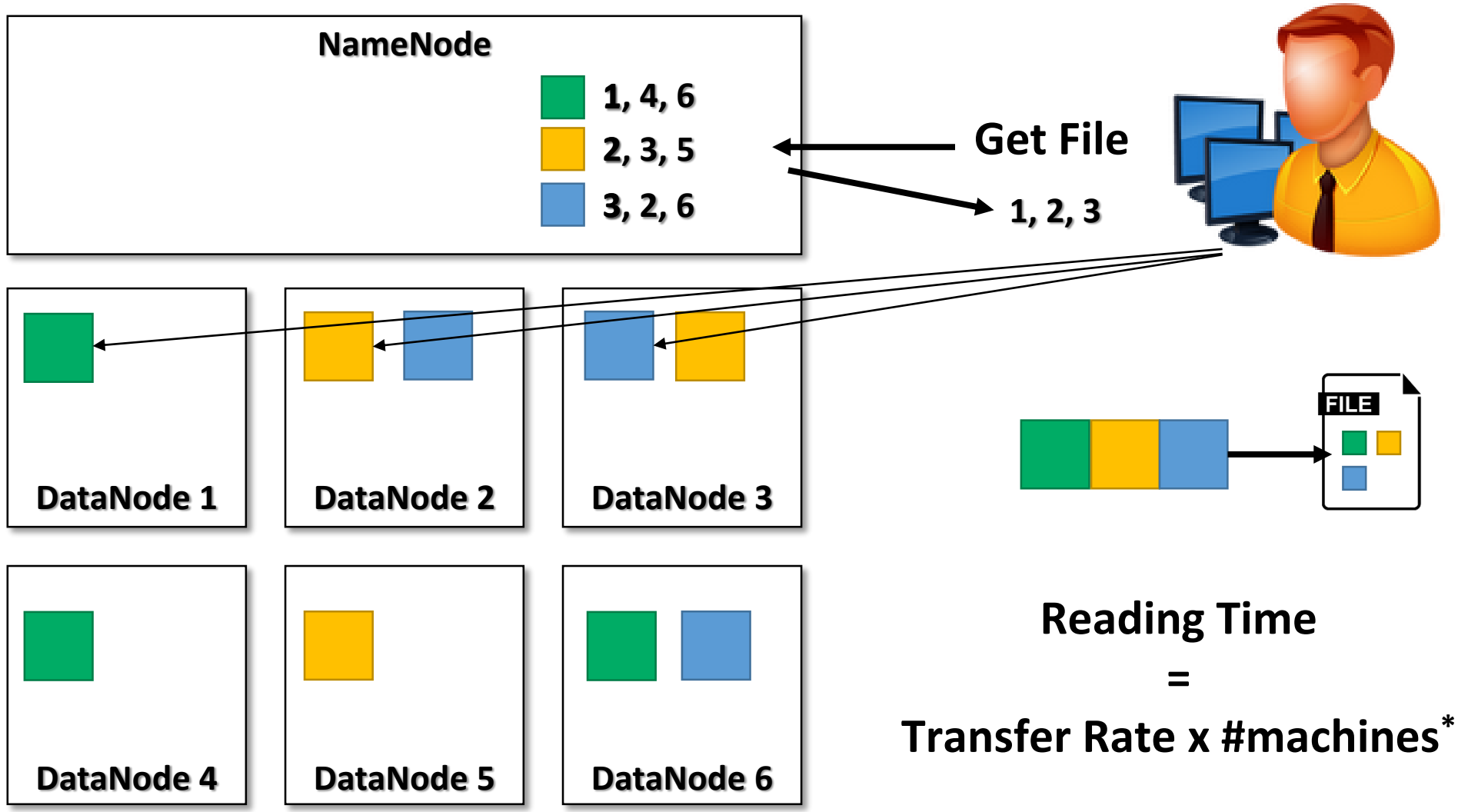
# HDFS : File Writing Sequence

Put File



# HDFS : File Reading Sequence

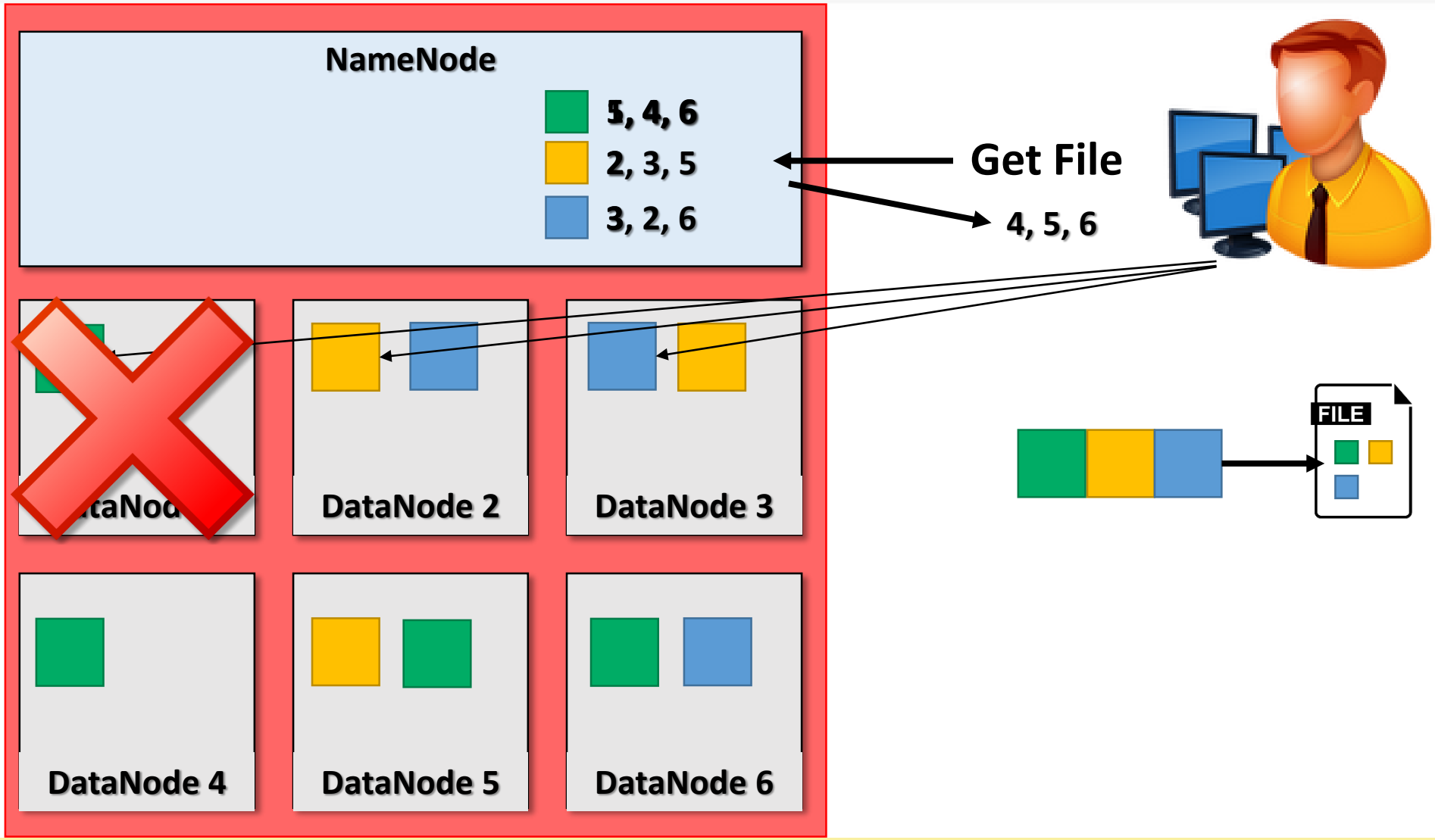



**Reading Time**

**=**

**Transfer Rate x #machines***

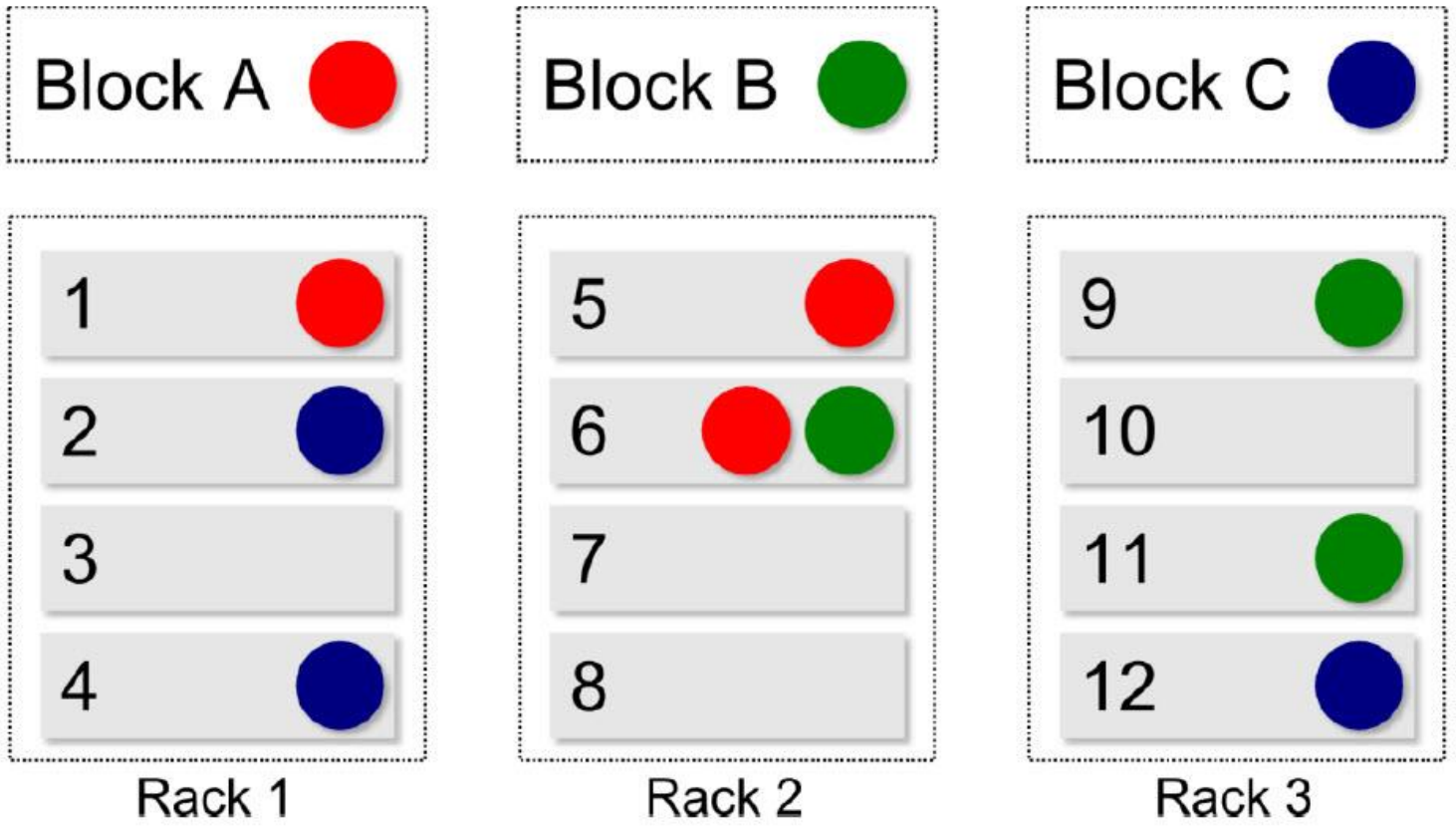
# HDFS : DataNode Failure Case



# HDFS Replication Management

- Data is replicated across multiple nodes which span multiple racks.
- Name Node ensures data block is not over or under replicated. Also ensures all replicas are not on the same rack.
- Extra replicas are deleted ensuring that number of racks remain same.
- Under replicated blocks are put in replication priority queue. Blocks with only one replica left are given highest priority.

# Block Replication Strategy




- 1st replica is placed on a node in the local rack
- The 2nd replica is placed on a different node in a different rack
- The 3rd replica is placed in the same rack of the second replica but on different node
- File replication factor can be defined dynamically
- Rack-awarness
  Optimize inter-rack data transfert

# Other HDFS Concepts

- **Managing Authorizations & Authentications**
  - **Permissions in HDFS are similar to Linux POSIX**

    **Can be defined using these commands:**
    **`hdfs dfs -chmod, -chown, or -chgrp`**


- **Quota: Number of files for a directory**

  *Sets total number of files that can be stored in each directory.*

- **Quota: Total disk space for a directory**

  *Sets total disk space that can be used by each directory.*

# HDFS Shell

- **Easy to use command line interface.**
  - *Create, copy, move, and delete files.*
- **Administrative duties**
  - *chmod, chown, chgrp*
- **Set replication factor for a file.**
- *head, tail, cat* **to view files**
- **…**

```
Usage: hdfs [--config confdir] COMMAND
       where COMMAND is one of:
  dfs                  run a filesystem command on the file systems supported in Hadoop.
  namenode -format     format the DFS filesystem
  secondarynamenode    run the DFS secondary namenode
  namenode             run the DFS namenode
  journalnode          run the DFS journalnode
  zkfc                 run the ZK Failover Controller daemon
  datanode             run a DFS datanode
  dfsadmin             run a DFS admin client
  haadmin              run a DFS HA admin client
  fsck                 run a DFS filesystem checking utility
  balancer             run a cluster balancing utility
  jmxget               get JMX exported values from NameNode or DataNode.
  mover                run a utility to move block replicas across
                       storage types
  oiv                  apply the offline fsimage viewer to an fsimage
  oiv_legacy           apply the offline fsimage viewer to an legacy fsimage
  oev                  apply the offline edits viewer to an edits file
  fetchdt              fetch a delegation token from the NameNode
  getconf              get config values from configuration
  groups               get the groups which users belong to
  snapshotDiff         diff two snapshots of a directory or diff the
                       current directory contents with a snapshot
  lsSnapshottableDir   list all snapshottable dirs owned by the current user
                                              Use -help to see options
  portmap              run a portmap service
  nfs3                 run an NFS version 3 gateway
  cacheadmin           configure the HDFS cache
  crypto               configure HDFS encryption zones
  storagepolicies      get all the existing block storage policies
  version              print the version
```

# Start / Stop NameNode Server

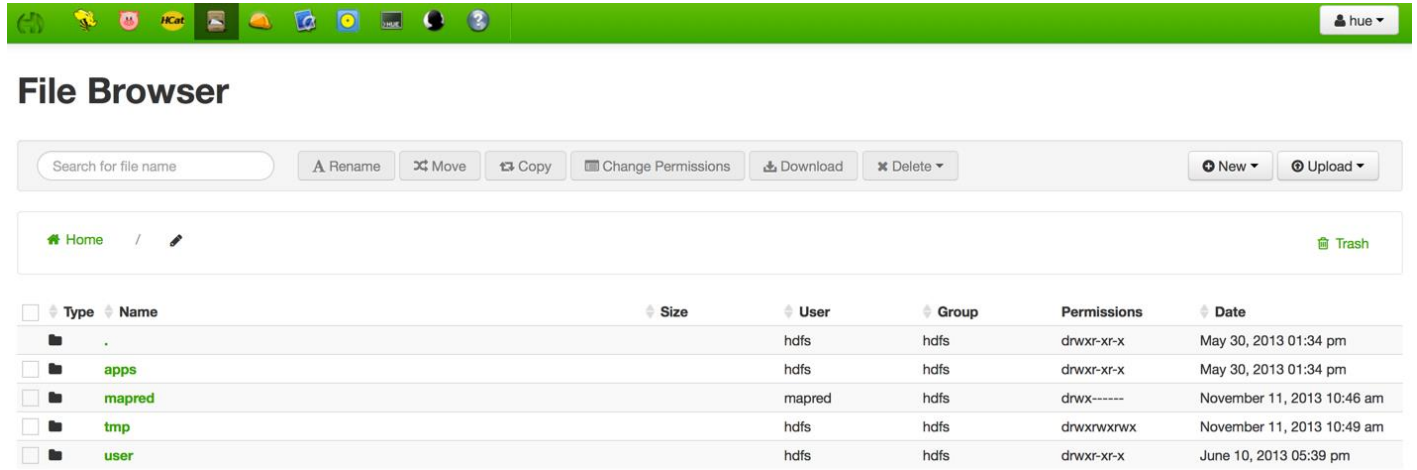
- **To Start HDFS**
  - From the hadoop/sbin folder use the commande start-dfs.sh
  - This commande starts the NameNode, DataNodes and Secondary NameNode


- **To Stop HDFS**
  - From the hadoop/sbin foler use the commande stop-dfs.sh
  - This commande stops the NameNode, DataNodes and Secondary NameNode

# Access to HDFS

- You can access HDFS via:
  - command line (Shell command)
  - the imbedded web interface
  - the Java API

```java
import ...;
public class HDFSwrite {
  public static void main(String[] args) throws IOException
    Configuration conf = new Configuration();
    FileSystem fs = FileSystem.get(conf);
    Path nomcomplet = new Path("apitest.txt");
    if (! fs.exists(nomcomplet)) {
      FSDataOutputStream outStream = fs.create(nomcomplet);
      outStream.writeUTF("Bonjour tout le monde !");
      outStream.close();
    }
    fs.close();
  }
}
```




```
[root@sandbox ~]# hadoop fs -help
Usage: hadoop fs [generic options]
        [-appendToFile <localsrc> ... <dst>]
        [-cat [-ignoreCrc] <src> ...]
        [-checksum <src> ...]
        [-chgrp [-R] GROUP PATH...]
        [-chmod [-R] <MODE[,MODE]... | OCTALMODE> PATH...]
        [-chown [-R] [OWNER][:[GROUP]] PATH...]
        [-copyFromLocal [-f] [-p] [-l] <localsrc> ... <dst>]
        [-copyToLocal [-p] [-ignoreCrc] [-crc] <src> ... <localdst>]
        [-count [-q] [-h] [-v] [-t [<storage type>]] <path> ...]
        [-cp [-f] [-p | -p[topax]] <src> ... <dst>]
        [-createSnapshot <snapshotDir> [<snapshotName>]]
        [-deleteSnapshot <snapshotDir> <snapshotName>]
        [-df [-h] [<path> ...]]
        [-du [-s] [-h] <path> ...]
        [-expunge]
        [-find <path> ... <expression> ...]
```

# HDFS Command Format

**hdfs [--config confdir] COMMAND**

**e.g:**

- **`hdfs --help`**
  (print the help screen)
- **`hdfs dfsadmin -report`**
  (print information about the cluster)
- **`hdfs version`**
  (print the version)

```
Usage: hdfs [--config confdir] COMMAND
       where COMMAND is one of:
  dfs                  run a filesystem command on the file systems supported in Hadoop.
  namenode -format     format the DFS filesystem
  secondarynamenode    run the DFS secondary namenode
  namenode             run the DFS namenode
  journalnode          run the DFS journalnode
  zkfc                 run the ZK Failover Controller daemon
  datanode             run a DFS datanode
  dfsadmin             run a DFS admin client
  haadmin              run a DFS HA admin client
  fsck                 run a DFS filesystem checking utility
  balancer             run a cluster balancing utility
  jmxget               get JMX exported values from NameNode or DataNode.
  mover                run a utility to move block replicas across
                       storage types
  oiv                  apply the offline fsimage viewer to an fsimage
  oiv_legacy           apply the offline fsimage viewer to an legacy fsimage
  oev                  apply the offline edits viewer to an edits file
  fetchdt              fetch a delegation token from the NameNode
  getconf              get config values from configuration
  groups               get the groups which users belong to
  snapshotDiff         diff two snapshots of a directory or diff the
                       current directory contents with a snapshot
  lsSnapshottableDir   list all snapshottable dirs owned by the current user
                                                Use -help to see options
  portmap              run a portmap service
  nfs3                 run an NFS version 3 gateway
  cacheadmin           configure the HDFS cache
  crypto               configure HDFS encryption zones
  storagepolicies      get all the existing block storage policies
  version              print the version
```

# Common HDFS Operations

- Create / Delete  folders
- Copy files from local system to HDFS and vice versa
- List directory contents
- Display a file on stdout
- Allocated space analysis
- Verify read/write permissions
- Define replication factor for a given file





# Basic HDFS Commands

**hdfs dfs -mkdir <path>**
(create a new directory)

**hdfs dfs -rmr <src>**
(delete a folder - recursively)

**hdfs dfs -put <localsrc>…<dest>**
(copy a file from local systel to HDFS)

**hdfs dfs -copyFromLocal<localsrc>…<dest>**
(same as put)

**hdfs dfs -moveFromLocal<localsrc>…<dest>**
(same as put but the source file is deleted after the operation)

**hdfs dfs -ls <path>**
(list all files)

**hdfs dfs -lsr <path>**
(list all files - recursively)

**hdfs dfs -cat <src>**
(display the content of a file on 'stdout')

**hdfs dfs -help**
(display the help screen)

# It’s time for a break

Grab some coffee, We’ll be back in 15min


# Hadoop MapReduce

*Diving into the Hadoop core*

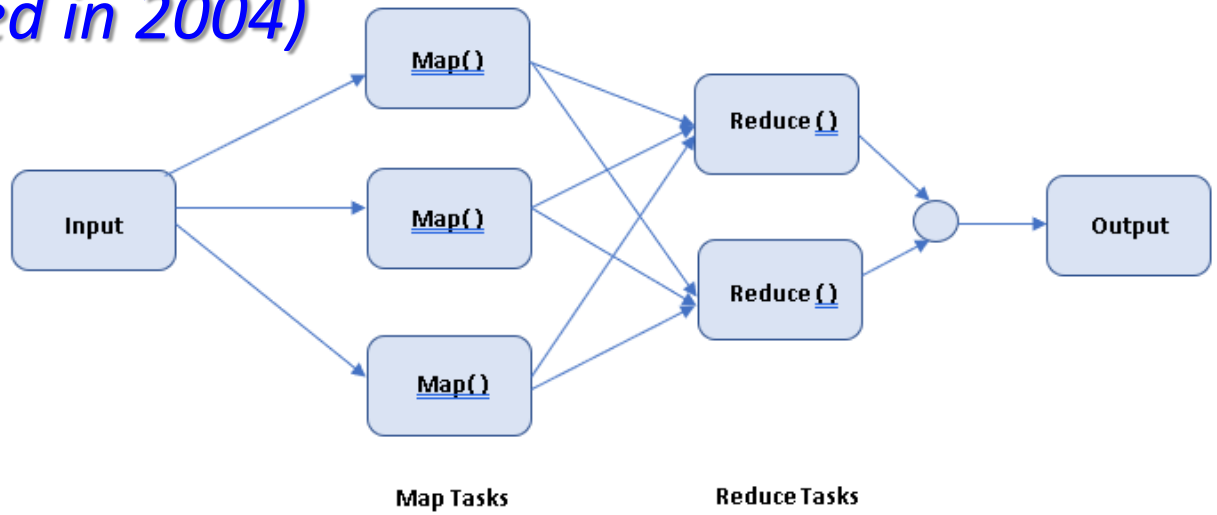
McGill

FALL 2018

# What is MapReduce?

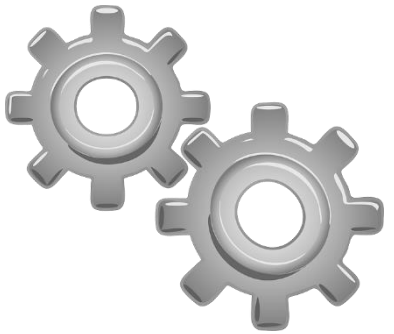
- MapReduce is a parallel programming model designed to process large scale data
- Intended to simplify the processing of vast amounts of data in parallel
- MapReduce was first developed by Google

  *(Google's MapReduce paper submitted in 2004)*



# MapReduce Overview

- **Automatic parallelization and distribution**
- **MapReduce scales out effortlessly from a single machine to thousands**
- **It was designed for Fault tolerant & High performance**
- **MapReduce runs on large clusters of commodity hardware**
- **Provides a clean abstraction for programmers**
  - *Written in JAVA*
  - *Can be written in other languages using "Hadoop Streaming"*



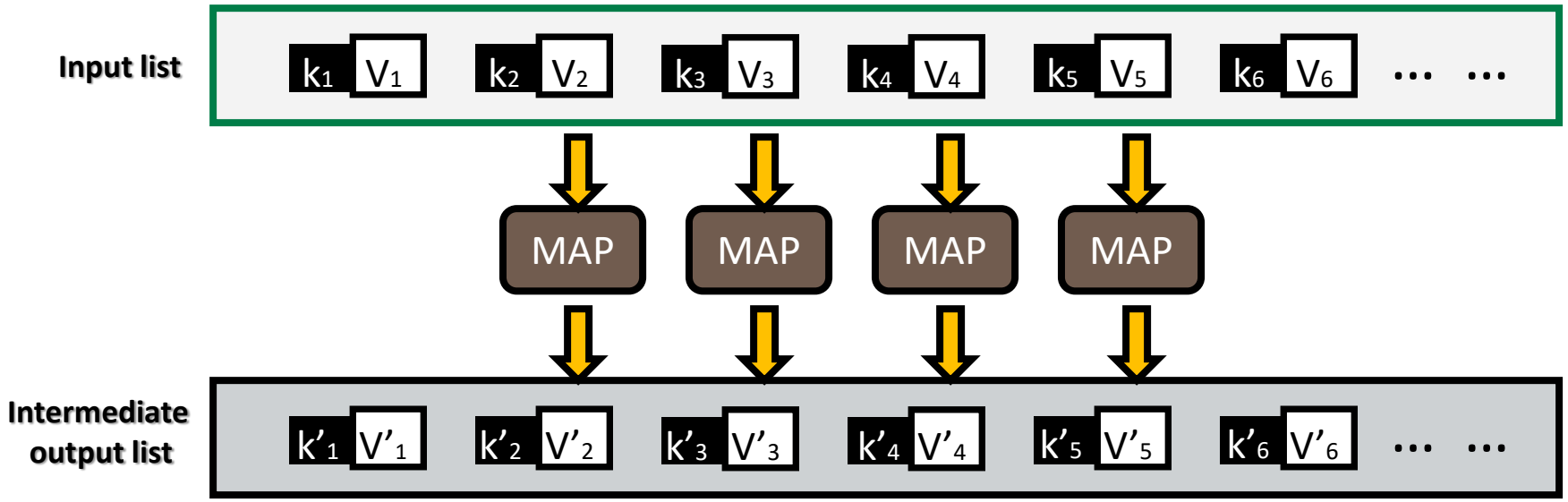
# Hadoop MapReduce Terminology

- **Job:** *MapReduce application*
- **Task:** *A MapReduce job is divided into tasks*
- **JobTracker:** *Master component for scheduling and managing jobs and cluster resources*
- **TaskTrackers:** *Slave component of the JobTracker – execute and manage tasks assigned by the JobTracker*
- **Shard:** *data partition*
- **Application Master (AM)**

  *Run any user code (Dryad, MapReduce, Tez, REEF…etc)*

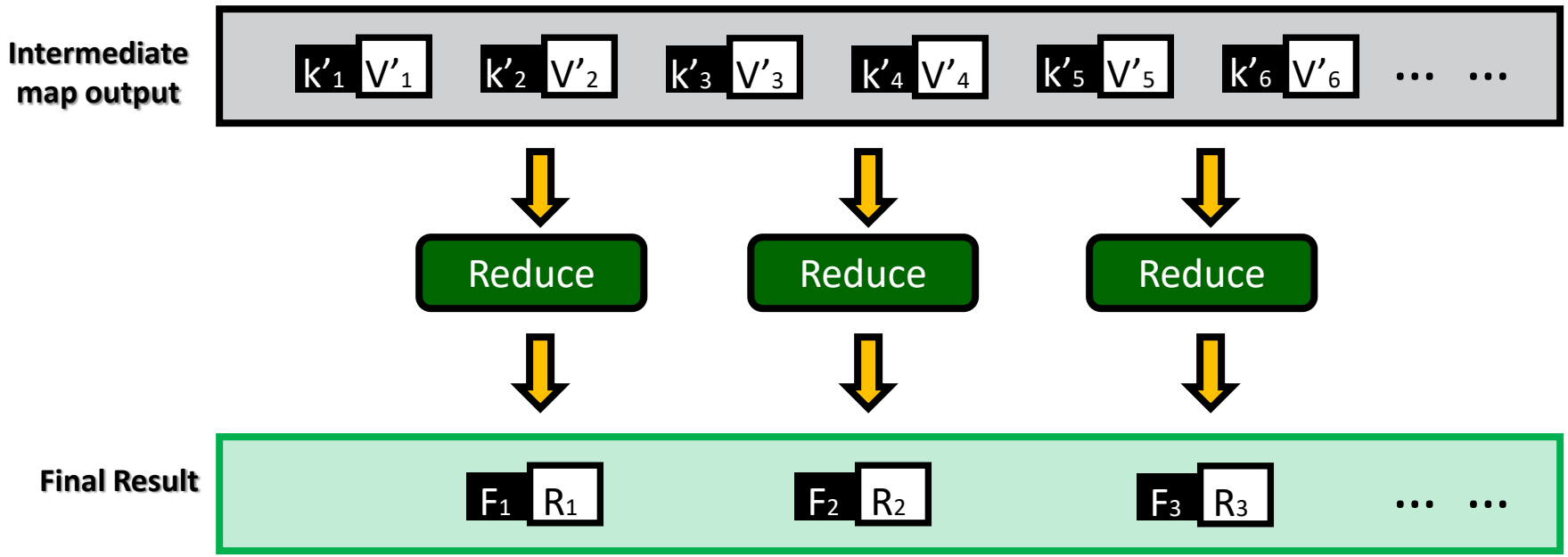
# Map

- A list of data elements are passed, one at a time, to map() functions
- map() functions transform each data element to an individual output data element.
- A map() produces one or more intermediate <key, values> pair(s) from the input list.



# Reduce

- After map phase finish, intermediate values with same output key are reduced into one or more final values



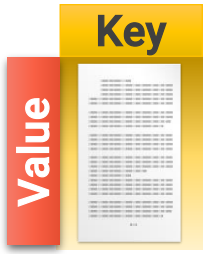
# MapReduce in Apache Hadoop

- **Hadoop MapReduce implementation provides these features:**
  - **Auto parallelization**
  - **Fault-tolerance**
  - **Status monitoring**
  - **Simple programming constructs**


- *Google's MapReduce is written in C++ - not open source*
- *Hadoop MapReduce is written in java – open source*

# Hadoop MapReduce data Flow

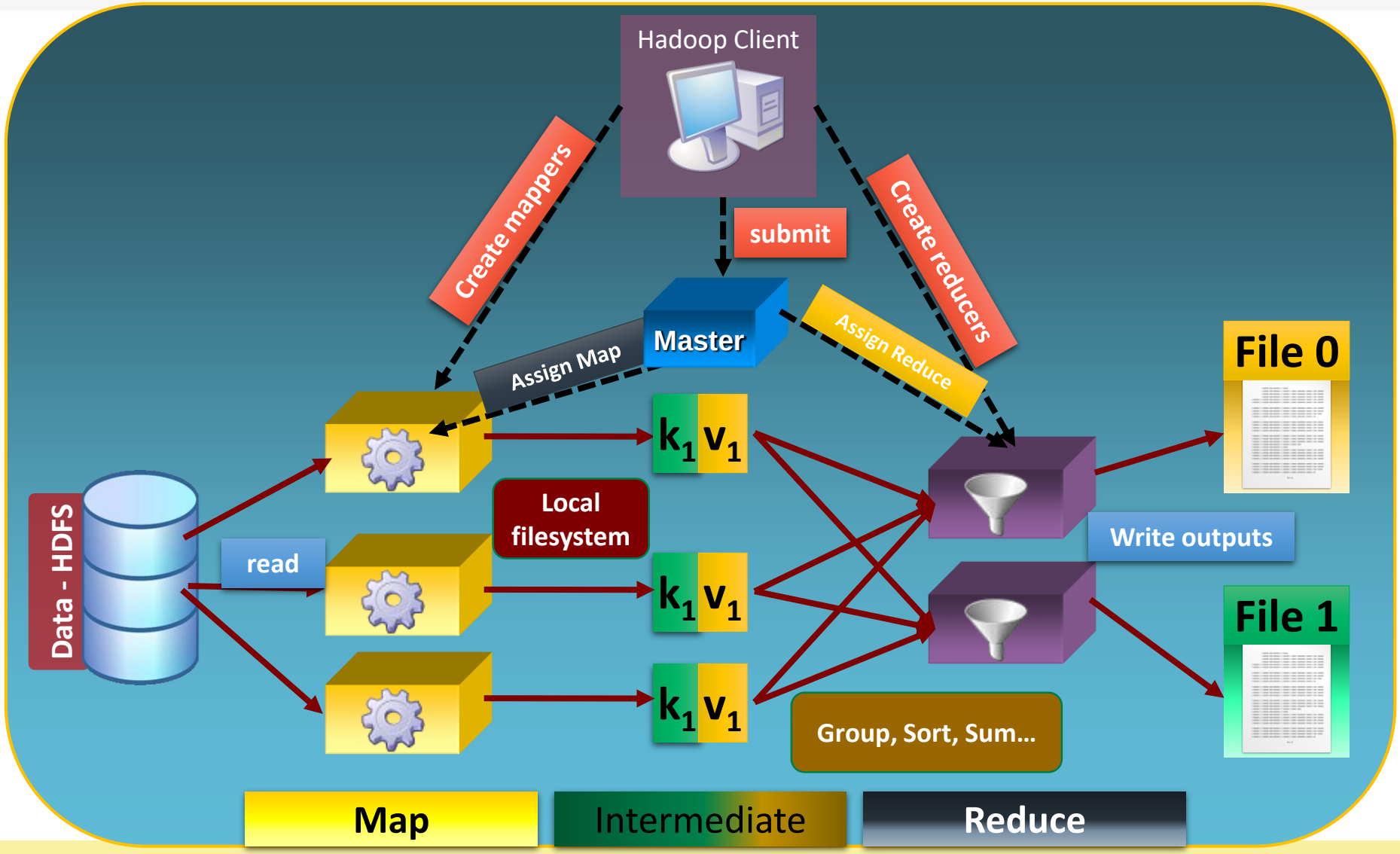

- **The Mapper**
  - Each Map Task operates on a single HDSF block
  - Map tasks run in the node where the is stored



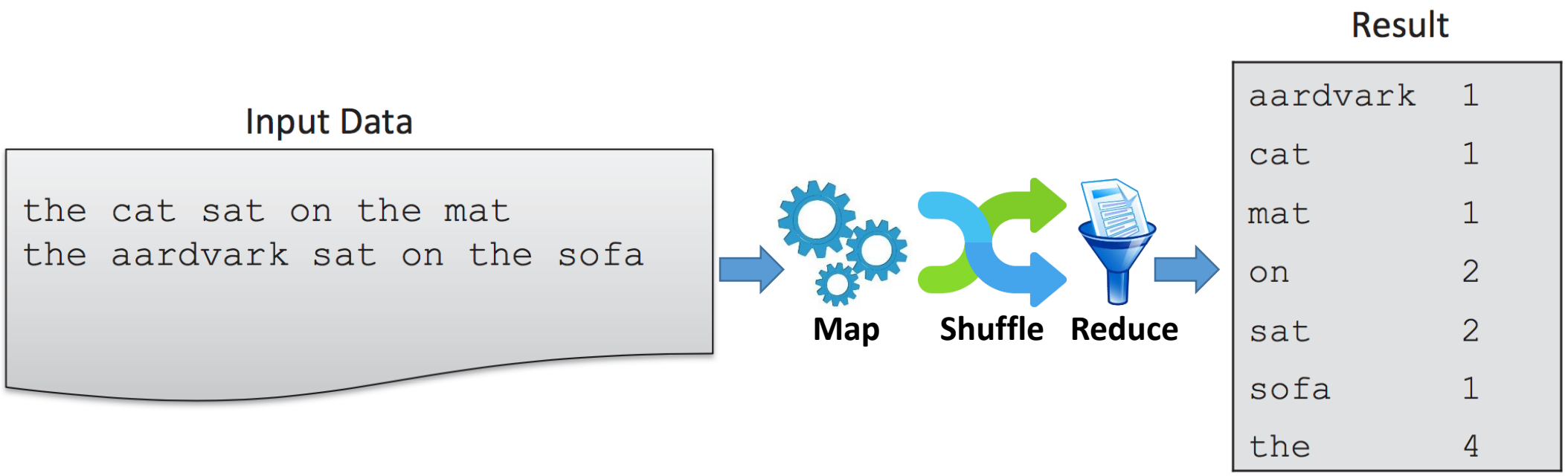

- **Shuffle and Sort**
  - Sorts and consolidates intermediate data form all mappers
  - Happens after all Map tasks are complete and before Reduce tasks start

- **The Reducer**
  - Operates on shufelled/sorted intermediate data (map output)
  - Produce final output

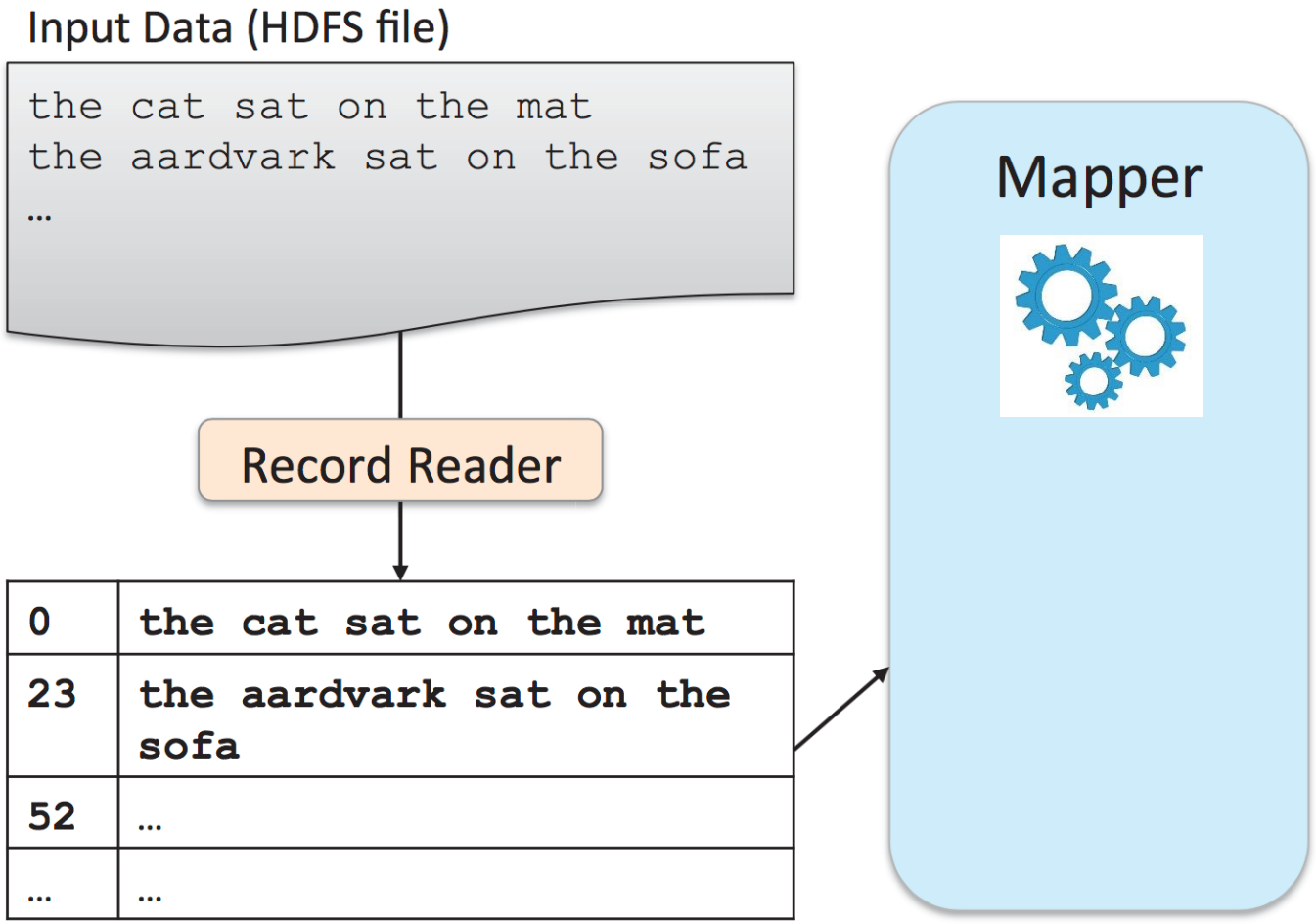
# Submitting MapReduce Job



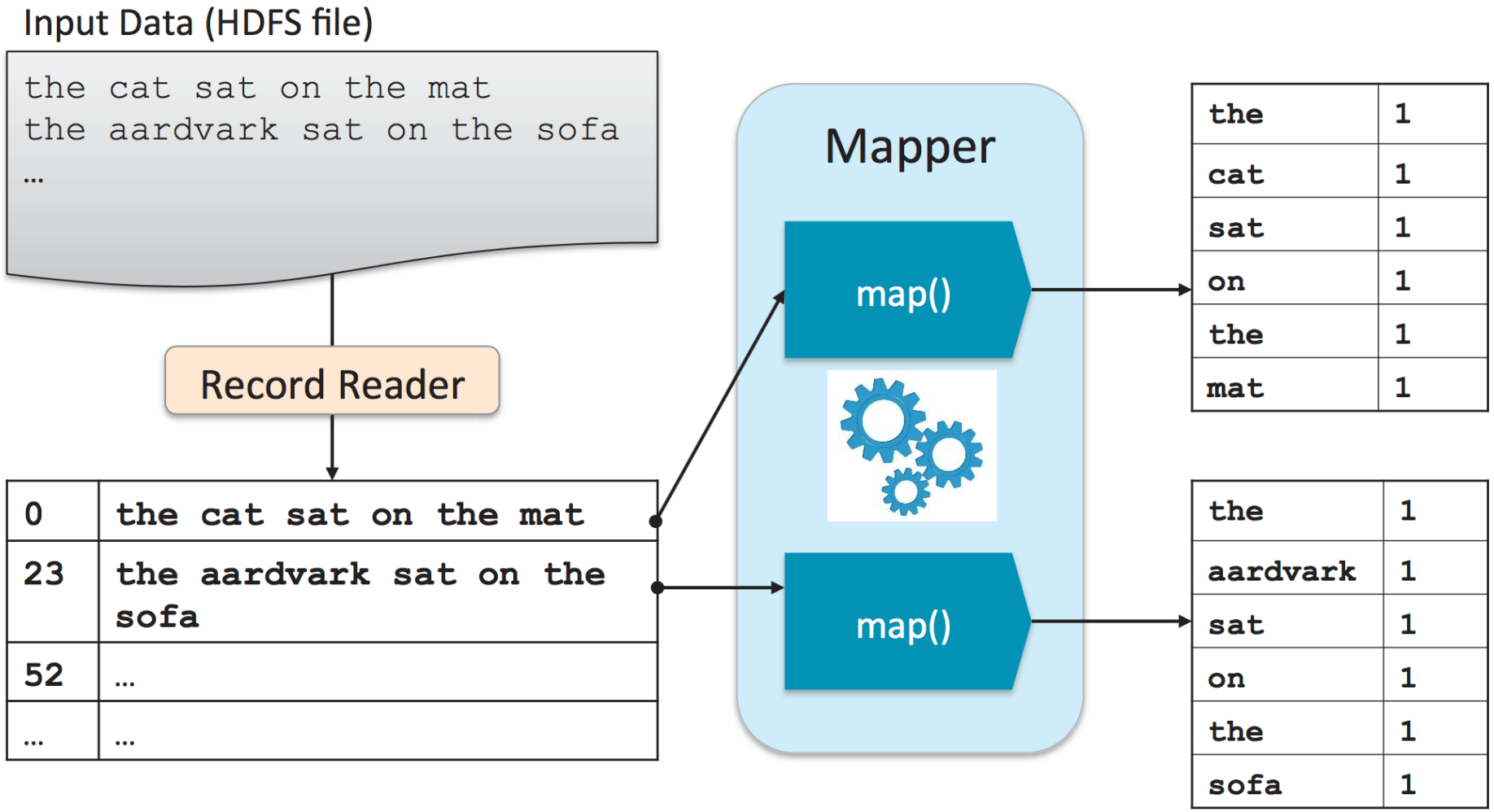
# WordCount Example



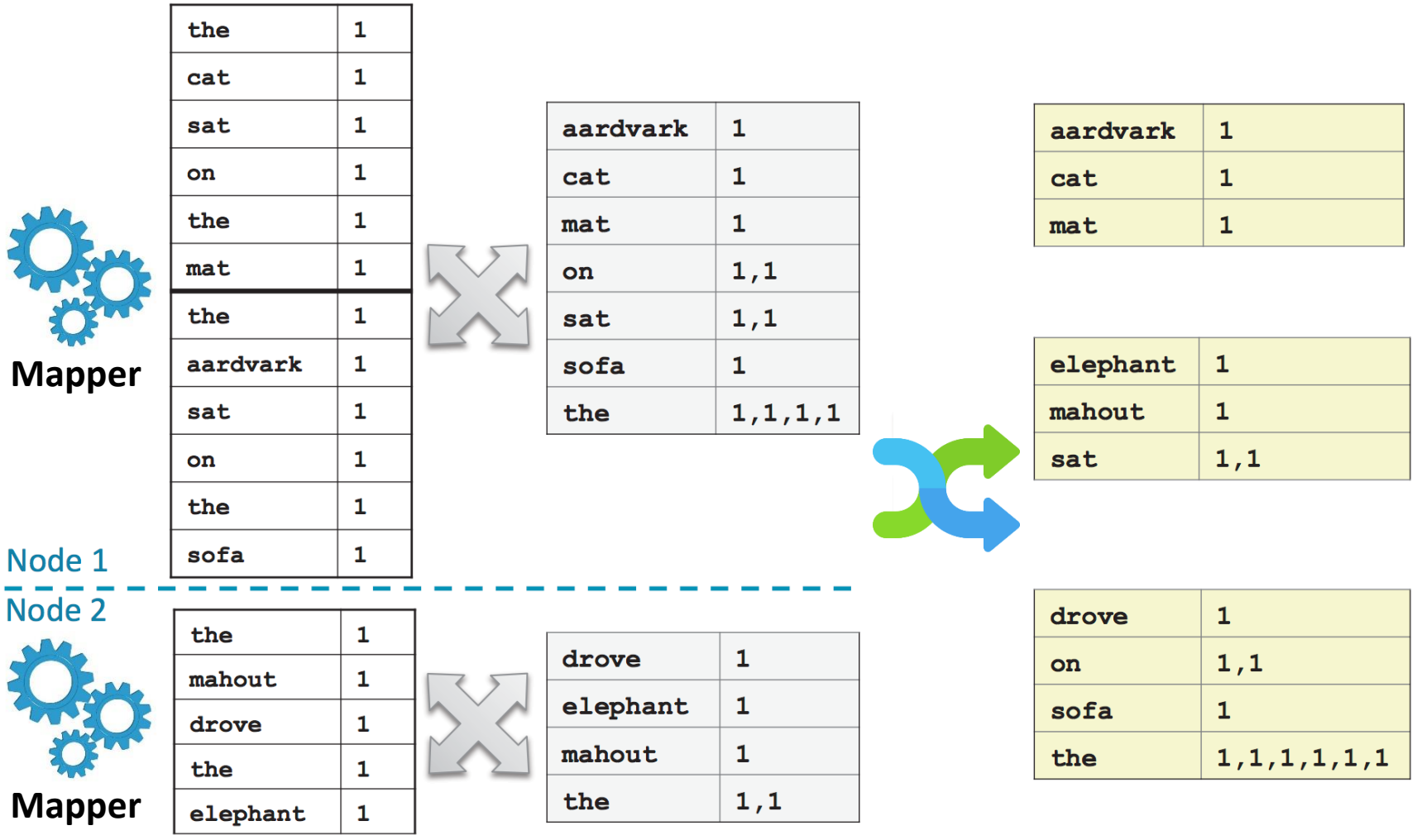
# WordCount Data Reading



# WordCount Mapper



# WordCount Shuffle & Sort




| | |
|---|---|
| the | 1 |
| cat | 1 |
| sat | 1 |
| on | 1 |
| the | 1 |
| mat | 1 |

| | |
|---|---|
| the | 1 |
| aardvark | 1 |
| sat | 1 |
| on | 1 |
| the | 1 |
| sofa | 1 |

| | |
|---|---|
| aardvark | 1 |
| cat | 1 |
| mat | 1 |
| on | 1,1 |
| sat | 1,1 |
| sofa | 1 |
| the | 1,1,1,1 |

| | |
|---|---|
| the | 1 |
| mahout | 1 |
| drove | 1 |
| the | 1 |
| elephant | 1 |

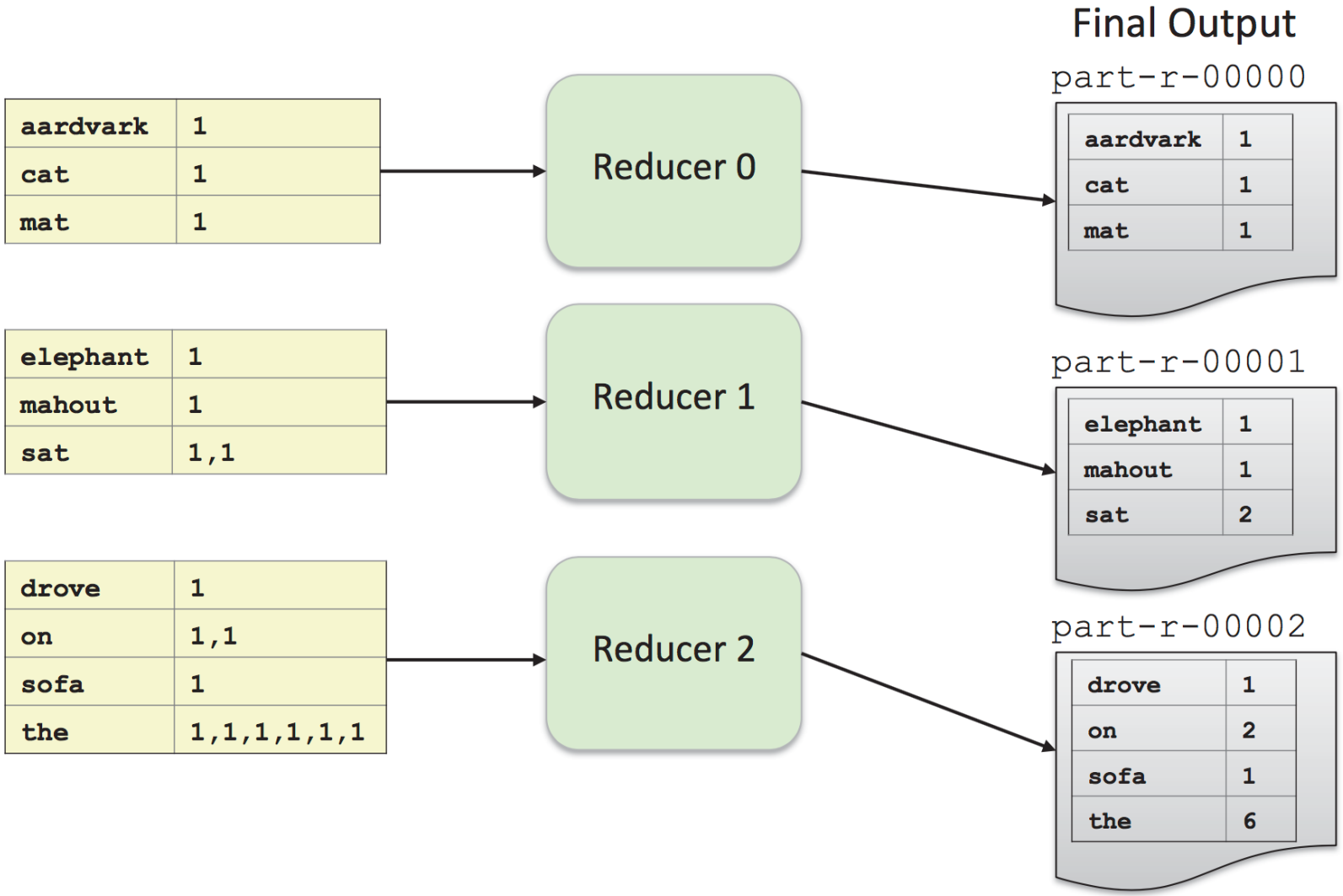
| | |
|---|---|
| drove | 1 |
| elephant | 1 |
| mahout | 1 |
| the | 1,1 |

| | |
|---|---|
| aardvark | 1 |
| cat | 1 |
| mat | 1 |

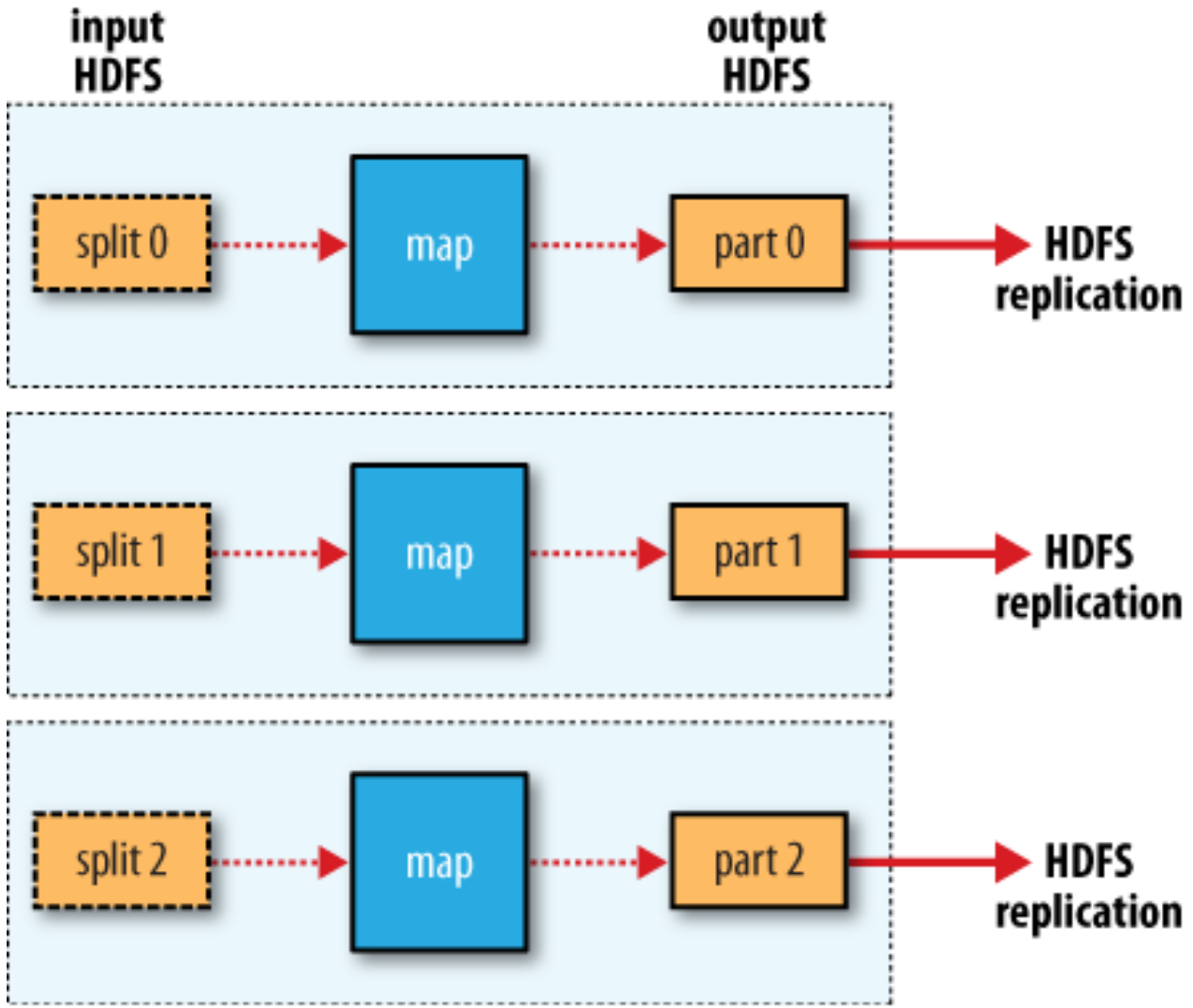
| | |
|---|---|
| elephant | 1 |
| mahout | 1 |
| sat | 1,1 |

| | |
|---|---|
| drove | 1 |
| on | 1,1 |
| sofa | 1 |
| the | 1,1,1,1,1,1 |

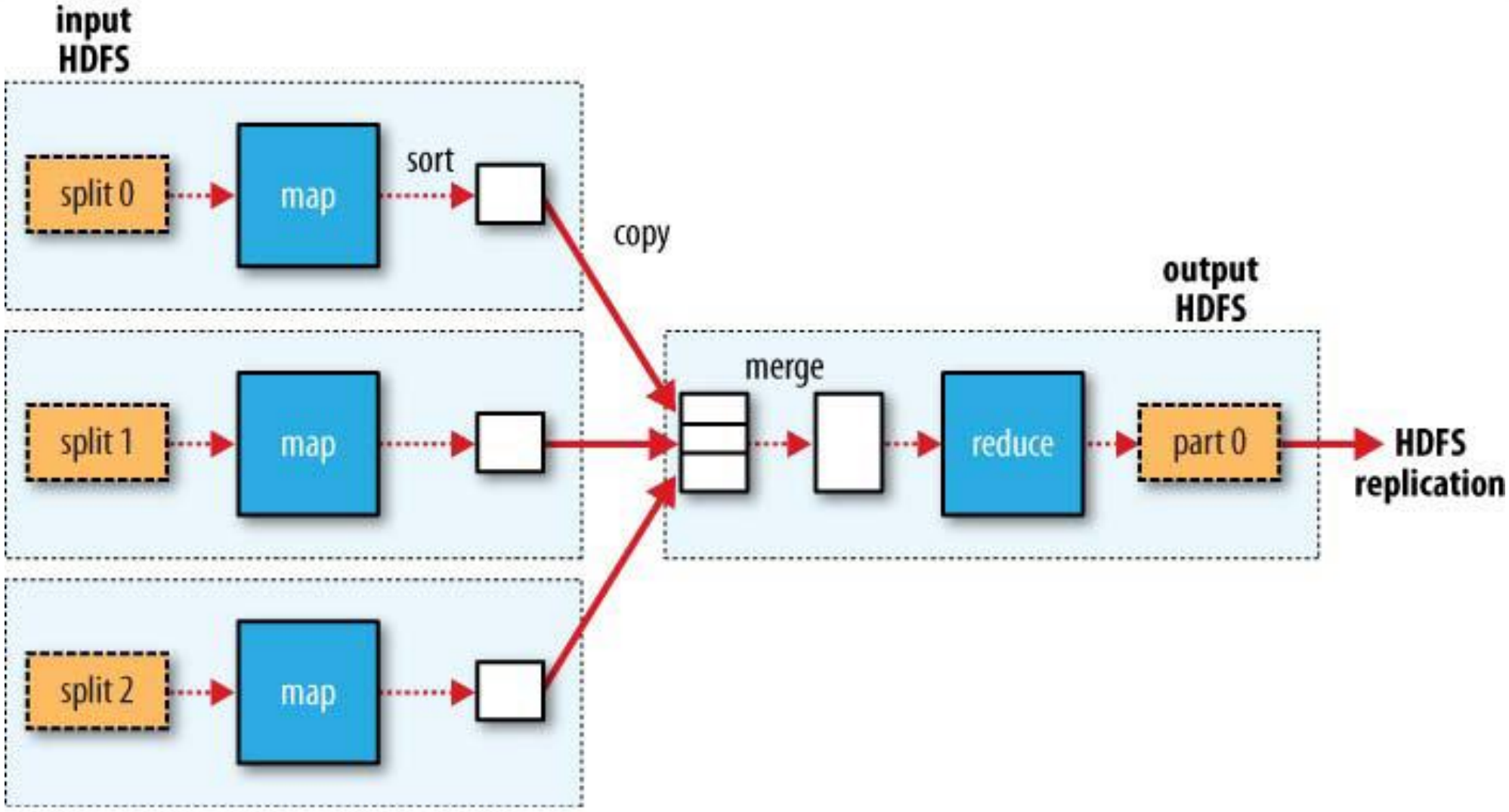
# WordCount Reducer



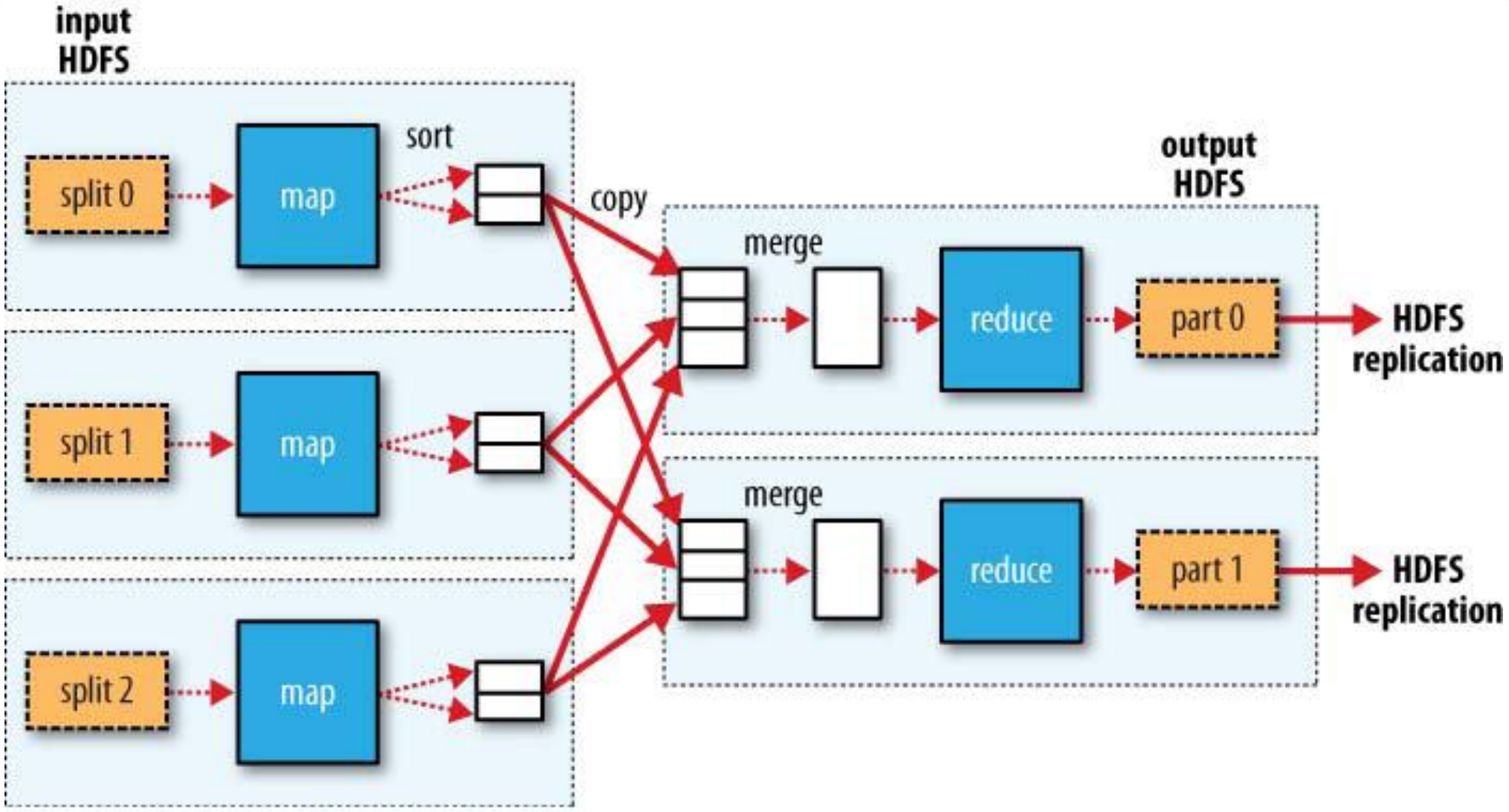
# MapReduce Job: 0 reducer



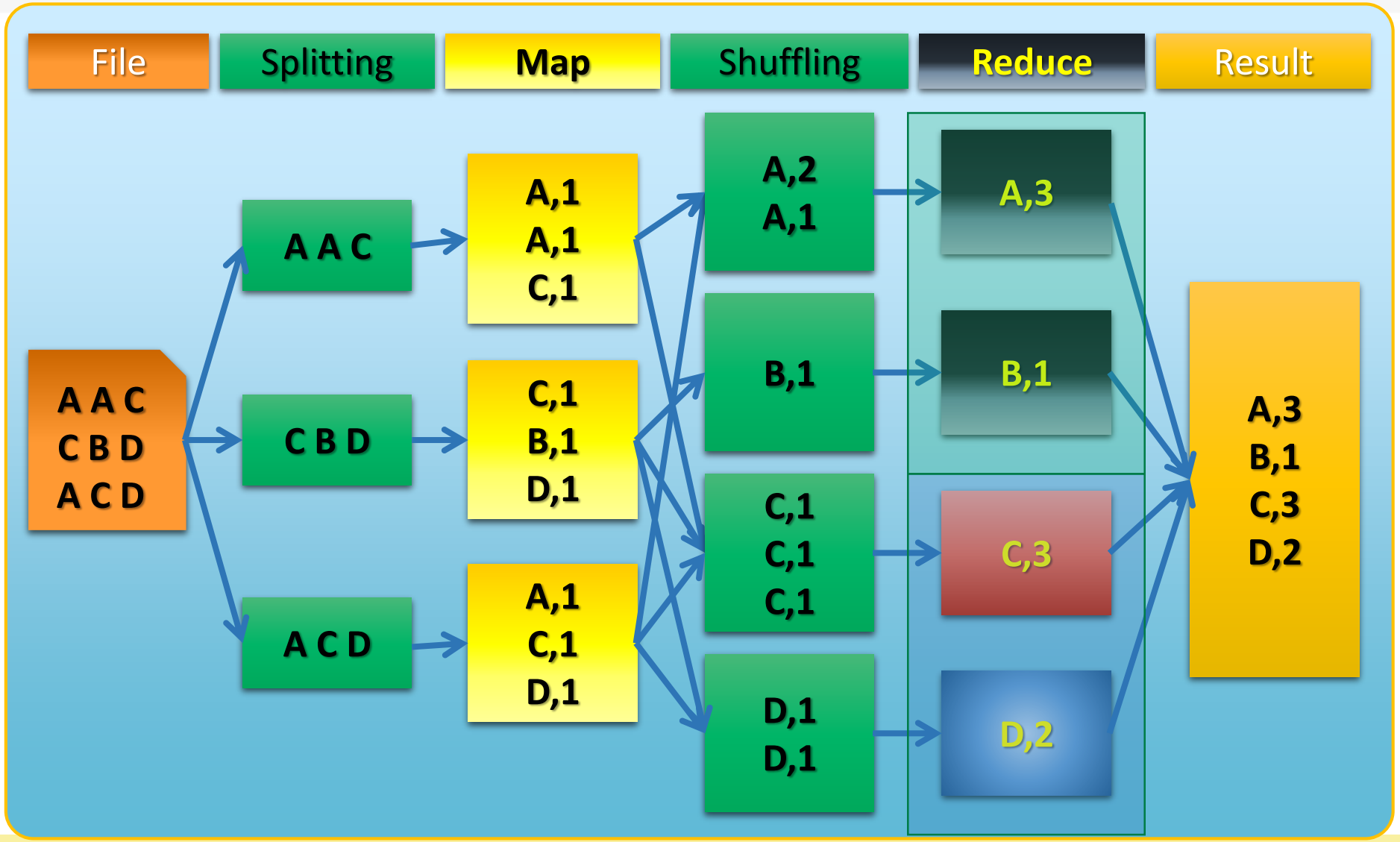
# MapReduce Job: 1 reducer



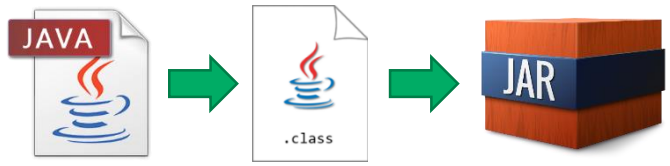
# MapReduce Job: 2 reducer



# MapReduce Combiner Example



# Write a MapReduce Application




Java

Combiner (Reducer Code)
(Optional)

Partitioner Code
(Optional)

Mapper Code
(Mandatory)

Reducer Code
(Optional)

Driver Code

# Programming MapReduce

**Externally: For user**

1. Write a Map program (short), write a Reduce program (short)
2. Specify number of Maps and Reduces (parallelism level)
3. Submit job; wait for result
4. Need to know very little about parallel/distributed programming!

**Internally: For the Paradigm and Scheduler**

1. Parallelize Map
2. Transfer data from Map to Reduce
3. Parallelize Reduce
4. Implement Storage for Map input, Map output, Reduce input, and Reduce output

*(Ensure that no Reduce starts before all Maps are finished. That is, ensure the barrier between the Map phase and Reduce phase)*

# Snippet code example map()

```java
import java.io.IOException;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Mapper;

public class WordCountMapper extends Mapper<Object, Text, Text, IntWritable> {

    private final static IntWritable one = new IntWritable(1);

    @Override
    public void map(Object key, Text value, Context output) throws IOException,
            InterruptedException {

        //Split on whitespace
        String[] words = value.toString().split(" ");
        //Get First Word
        output.write(new Text(words[0]), one);
    }
}
```

# Snippet code example reduce()

```java
import java.io.IOException;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Reducer;

public class WordCountReducer extends Reducer<Text, IntWritable, Text, IntWritable> {

    @Override
    public void reduce(Text key, Iterable<IntWritable> values, Context output)
            throws IOException, InterruptedException {
        int voteCount = 0;
        for(IntWritable value: values){
            voteCount+= value.get();
        }
        output.write(key, new IntWritable(voteCount));
    }
}
```

# Snipp...river()

```java
public class WordCountMain17 extends Configured implements Tool{

    public static void main(String[] args) throws Exception {
        int res = ToolRunner.run(new Configuration(), new WordCountMain17(), args);
        System.exit(res);
    }

    @Override
    public int run(String[] args) throws Exception {
        if (args.length != 2) {
            System.out.println("usage: [input] [output]");
            System.exit(-1);
        }

        Job job = Job.getInstance(new Configuration());
        job.setOutputKeyClass(Text.class);
        job.setOutputValueClass(IntWritable.class);

        job.setMapperClass(WordCountMapper.class);
        job.setReducerClass(WordCountReducer.class);

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

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

        job.setJarByClass(WordCountMain17.class);

        job.submit();
        return 0;
    }
}
```

# Submitting MapReduce Job to Hadoop

- To run a MapReduce jar file use this command

```
hadoop jar examples.jar myjob […]
```


```
[cloudera@quickstart ~]$ hadoop
Usage: hadoop [--config confdir] COMMAND
       where COMMAND is one of:
  fs                   run a generic filesystem user client
  version              print the version
  jar <jar>            run a jar file
  checknative [-a|-h]  check native hadoop and compression libraries availability
  distcp <srcurl> <desturl> copy file or directories recursively
  archive -archiveName NAME -p <parent path> <src>* <dest> create a hadoop archive
  classpath            prints the class path needed to get the
  credential           interact with credential providers
                       Hadoop jar and the required libraries
  daemonlog            get/set the log level for each daemon
 or
  CLASSNAME            run the class named CLASSNAME

Most commands print help when invoked w/o parameters.
[cloudera@quickstart ~]$
```


EXERCICE

# Thank You