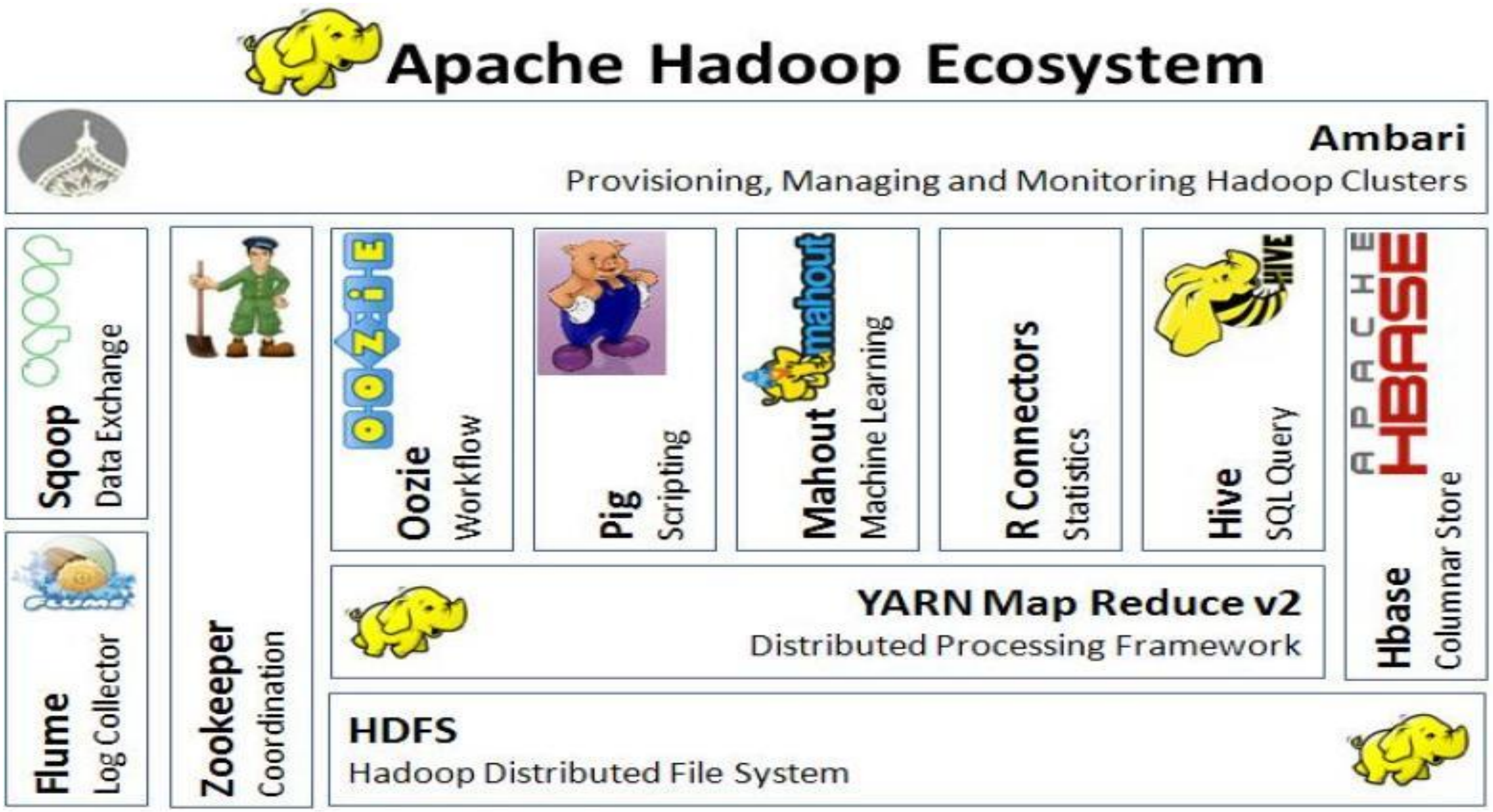
Introduction

The Hadoop Distributed File System (HDFS) is the file system component of Hadoop. It is designed to store very large data sets (1) *reliably*, and to stream those data sets (2) at *high bandwidth* to user applications. These are achieved by **replicating file content** on multiple machines(DataNodes).



Architecture

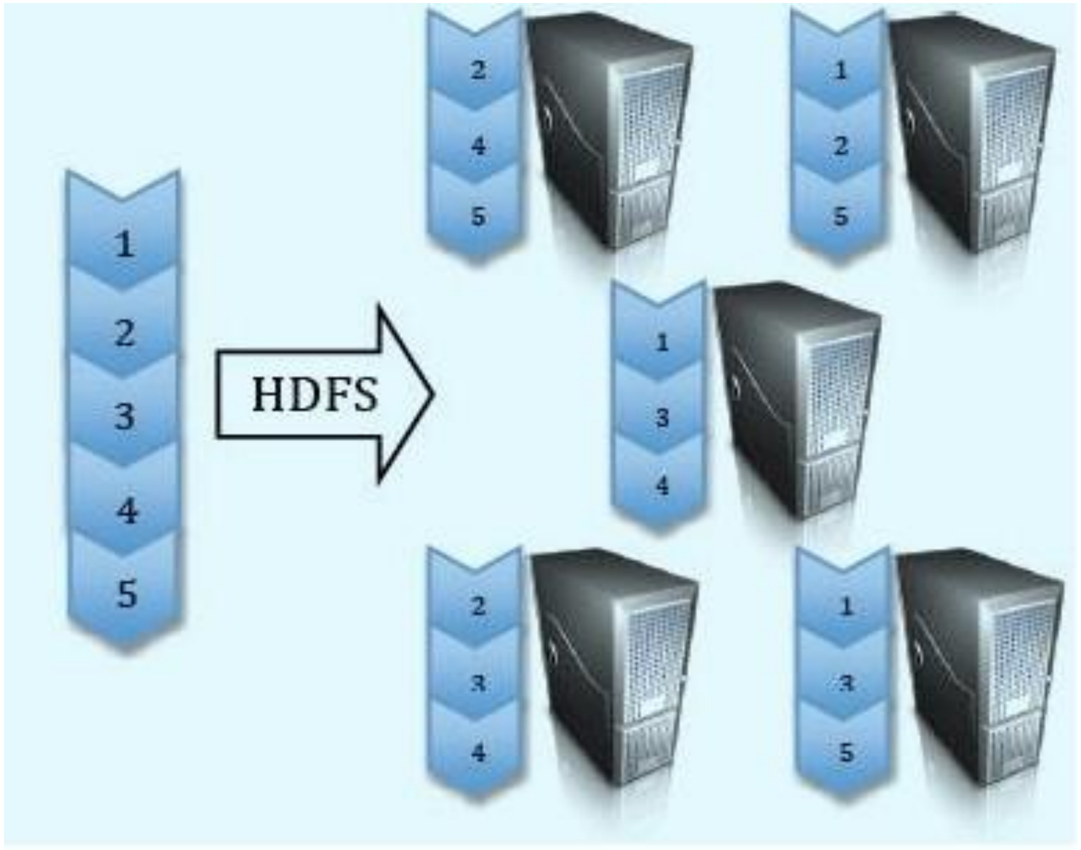
- HDFS is a block-structured file system: Files broken into blocks of 128MB (per-file configurable).

- A file can be made of several blocks, and they are stored across a cluster of one or more machines with data storage capacity.

- Each block of a file is replicated across a number of machines, To prevent loss of data.

Cluster HDFS

Architecture



Architecture

NameNode and DataNodes

- HDFS stores file system metadata and application data separately.
- **Metadata** refers to file metadata(attributes such as permissions, modification, access times, namespace and disk space quotas.
-)called “inodes”+list of blocks belong to the file.

- HDFS stores metadata on a dedicated server, called the NameNode.(Master) Application data are stored on other servers called DataNodes.(Slaves)

- All servers are fully connected and communicate with each other using TCP-based protocols.(RPC)

Architecture

- Single Namenode:

- Maintain the namespace tree(a hierarchy of files and directories) operations like opening, closing, and renaming files and directories.
- Determine the mapping of file blocks to DataNodes (the physical location of file data).
- File metadata (i.e. “inode”) .
- Authorization and authentication.
- Collect block reports from Datanodes on block locations.
- Replicate missing blocks.

- HDFS keeps the entire namespace in RAM, allowing fast access to the metadata.

Architecture

- DataNodes:

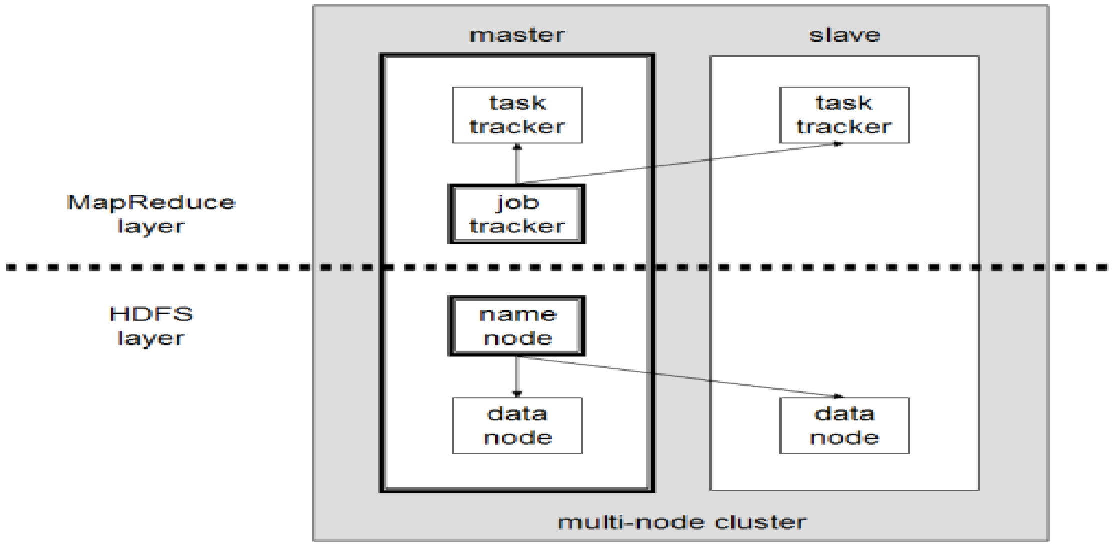
- The DataNodes are responsible for serving read and write requests from the file system's clients.

- The DataNodes also perform block creation, deletion, and replication upon instruction from the NameNode.

- Data nodes periodically send block reports to Namenode.

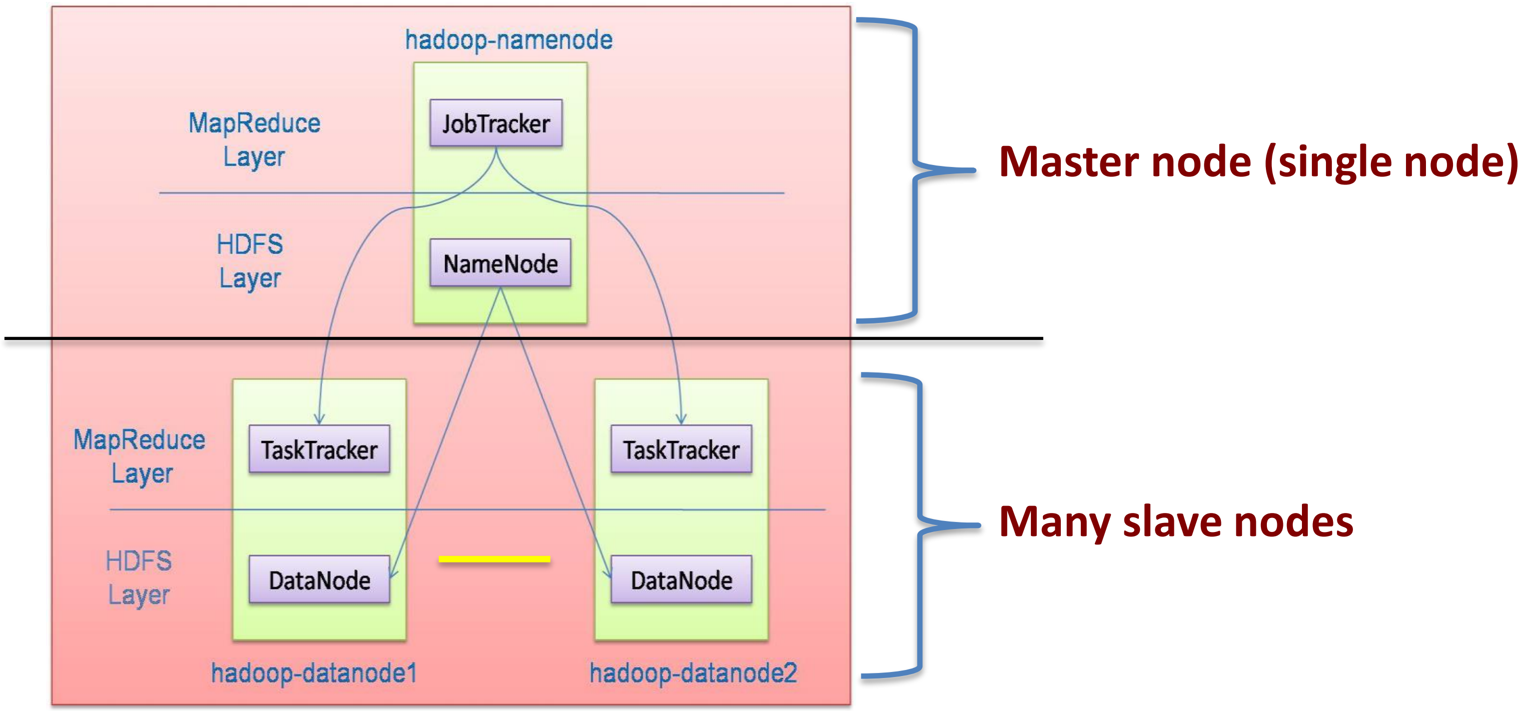
What is Hadoop

- **Hadoop framework consists on two main layers**
  - Distributed file system (HDFS)
  - Execution engine (MapReduce)



Hadoop Master/Slave Architecture

- Hadoop is designed as a *master-slave* *shared-nothing* architecture



Design Principles of Hadoop

- Need to process big data
- Need to parallelize computation across thousands of nodes
- **Commodity hardware**
  - Large number of low-end cheap machines working in parallel to solve a computing problem
- This is in contrast to **Parallel DBs**
  - Small number of high-end expensive machines

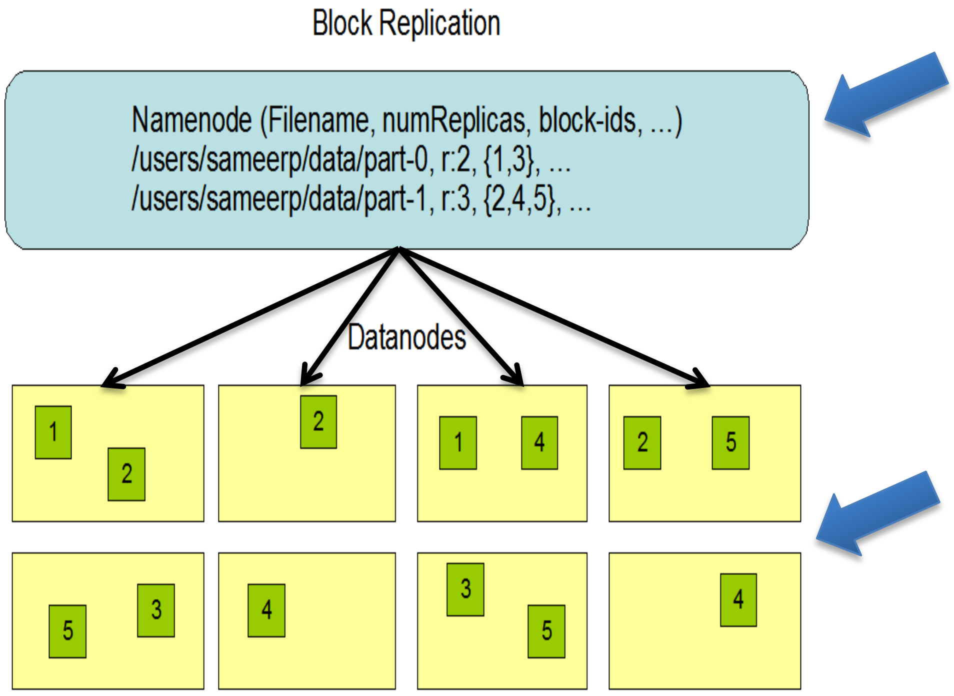
Design Principles of Hadoop

- **Automatic parallelization & distribution**
  - Hidden from the end-user
- **Fault tolerance and automatic recovery**
  - Nodes/tasks will fail and will recover automatically
- **Clean and simple programming abstraction**
  - Users only provide two functions “map” and “reduce”

How Uses MapReduce/Hadoop

- Google: Inventors of MapReduce computing paradigm
- Yahoo: Developing Hadoop open-source of MapReduce
- IBM, Microsoft, Oracle
- Facebook, Amazon, AOL, NetFlex
- Many others + universities and research labs

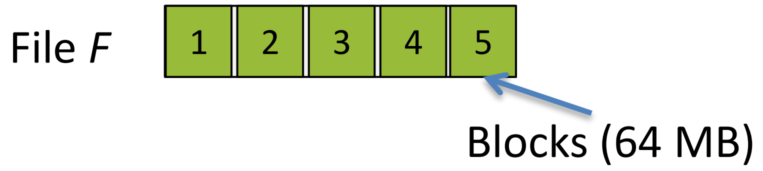
Hadoop Distributed File System (HDFS)



Block Replication
Namenode (Filename, numReplicas, block-ids, ...)
/users/sameerp/data/part-0, r:2, {1,3}, ...
/users/sameerp/data/part-1, r:3, {2,4,5}, ...
Datanodes
1
2
2
1
4
2
5
5
3
4
3
5
4

Centralized namenode

- Maintains metadata info about files



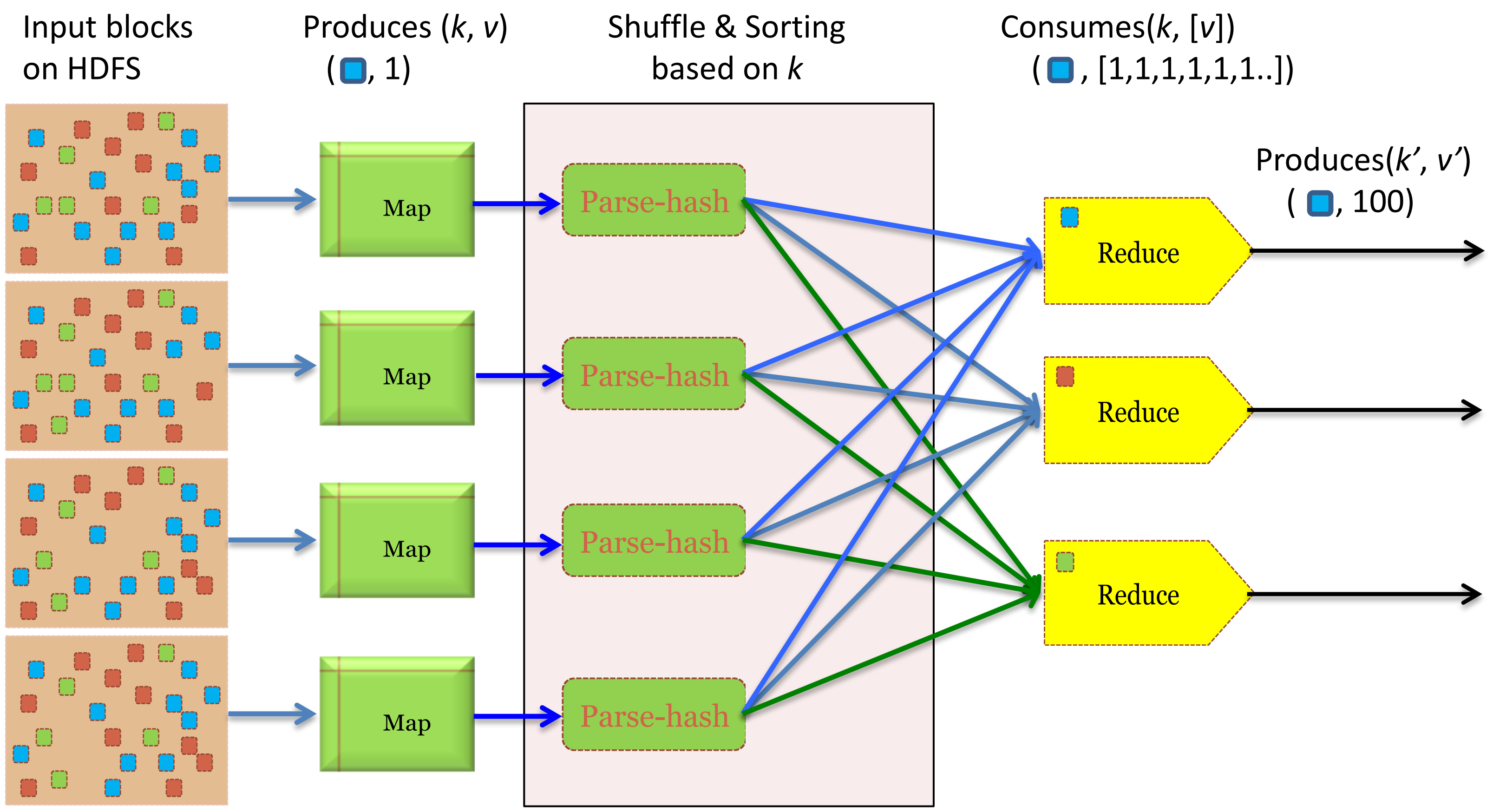
Many datanode (1000s)

- Store the actual data
- Files are divided into blocks
- Each block is replicated *N* times (Default = 3)

Main Properties of HDFS

- ***Large:*** A HDFS instance may consist of thousands of server machines, each storing part of the file system's data
- ***Replication:*** Each data block is replicated many times (default is 3)
- ***Failure:*** Failure is the norm rather than exception
- ***Fault Tolerance:*** Detection of faults and quick, automatic recovery from them is a core architectural goal of HDFS
  - Namenode is consistently checking Datanodes

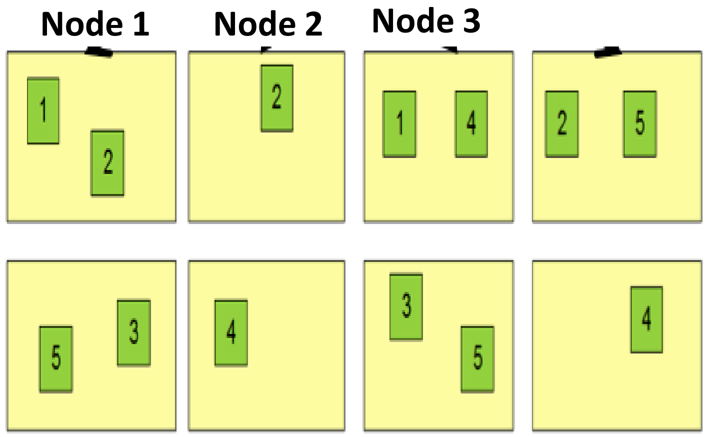
Map-Reduce Execution Engine (Example: Color Count)



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

Properties of MapReduce Engine

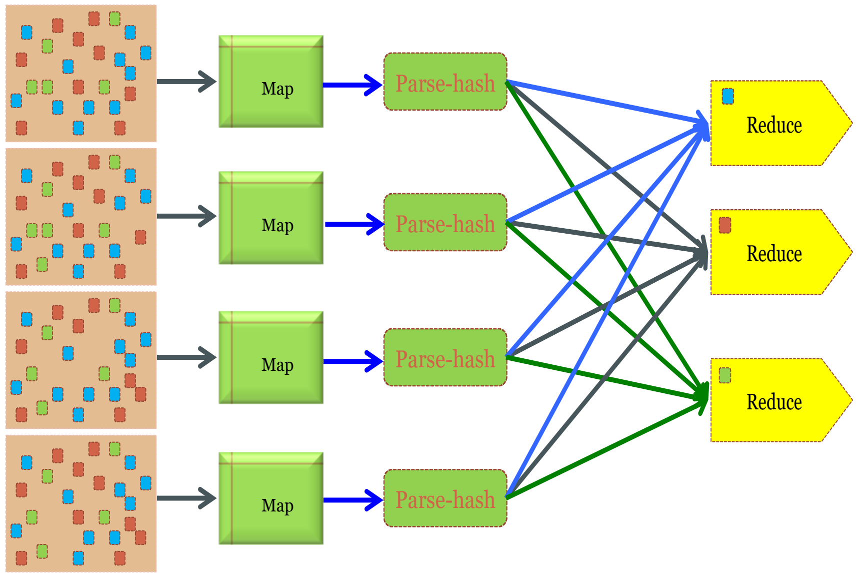
- **Job Tracker is the master node (runs with the namenode)**
  - Receives the user's job
  - Decides on how many tasks will run (number of mappers)
  - Decides on where to run each mapper (concept of locality)



- This file has 5 Blocks → run 5 map tasks
- Where to run the task reading block "1"
  - *Try to run it on Node 1 or Node 3*

Properties of MapReduce Engine

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

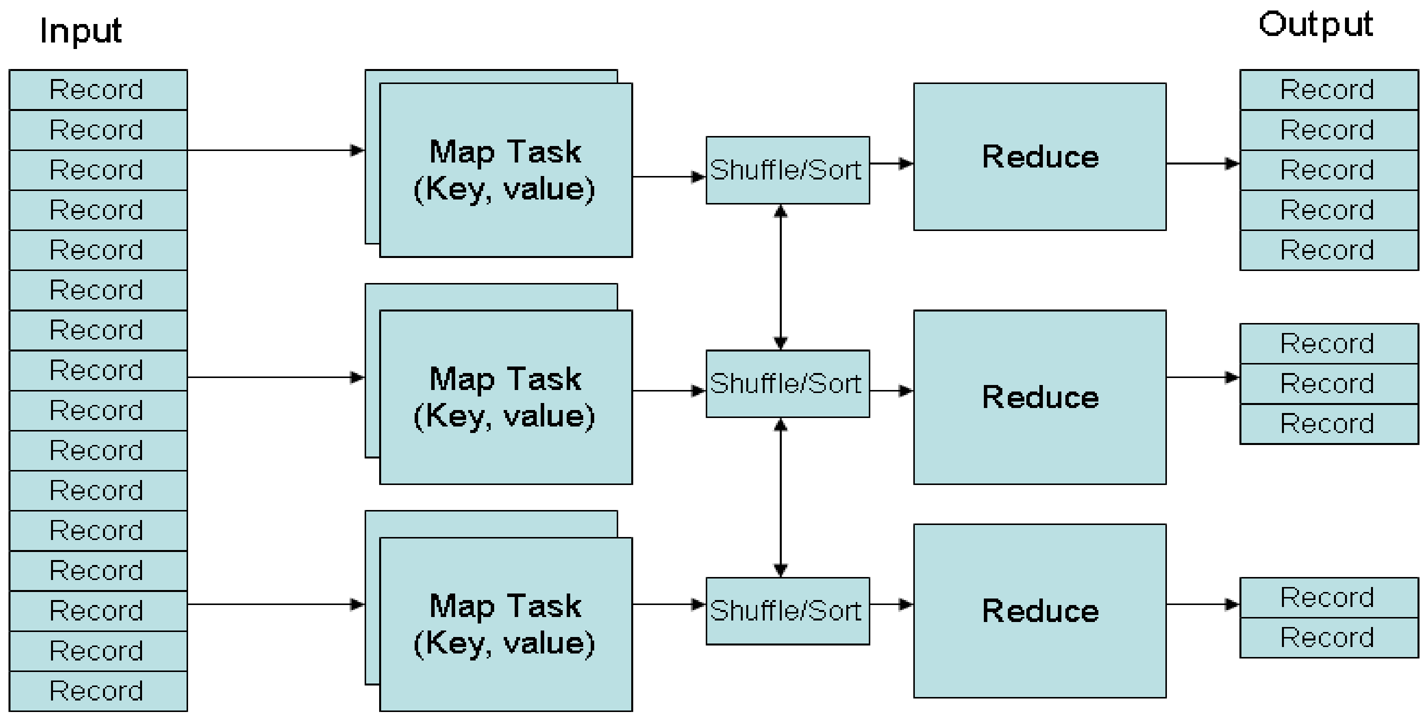


In this example, 1 map-reduce job consists of 4 map tasks and 3 reduce tasks

Key-Value Pairs

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

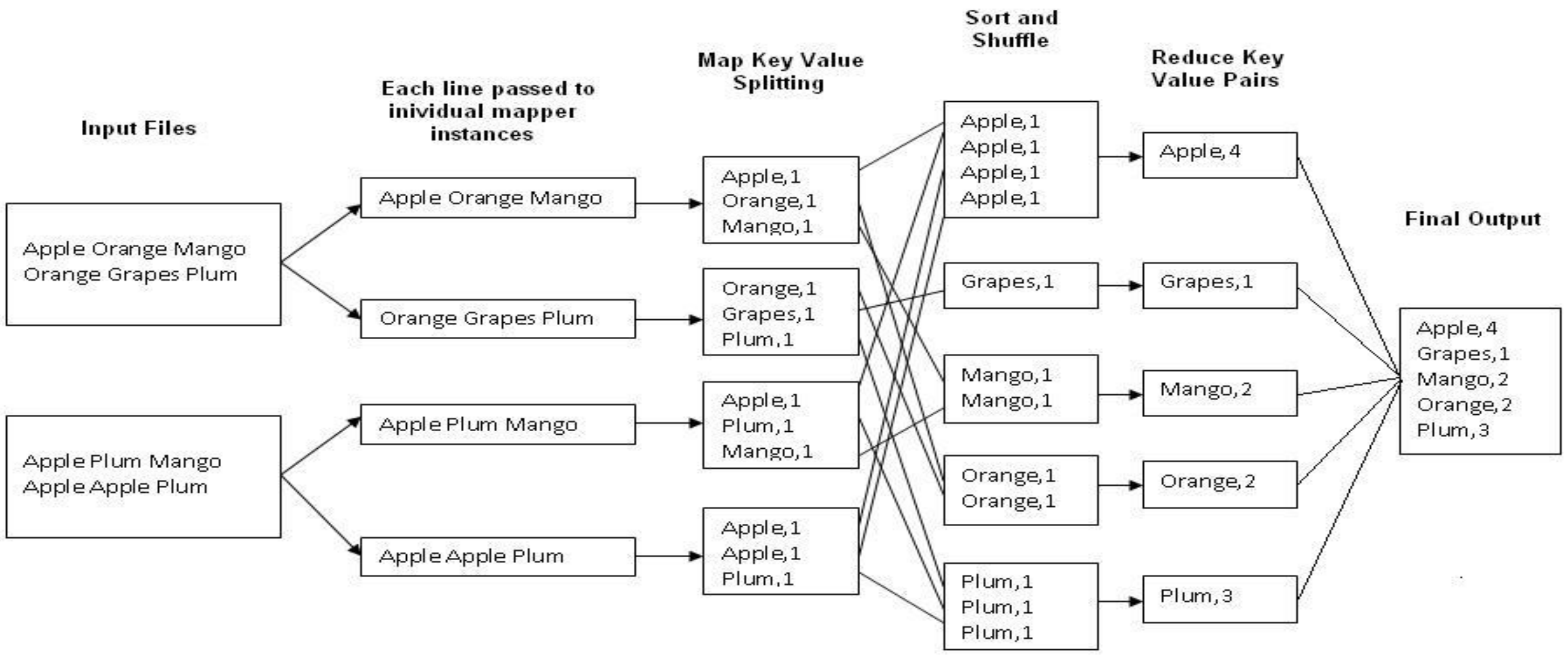
MapReduce Phases



Deciding on what will be the key and what will be the value → developer's responsibility

Example 1: Word Count

- **Job: Count the occurrences of each word in a data set**

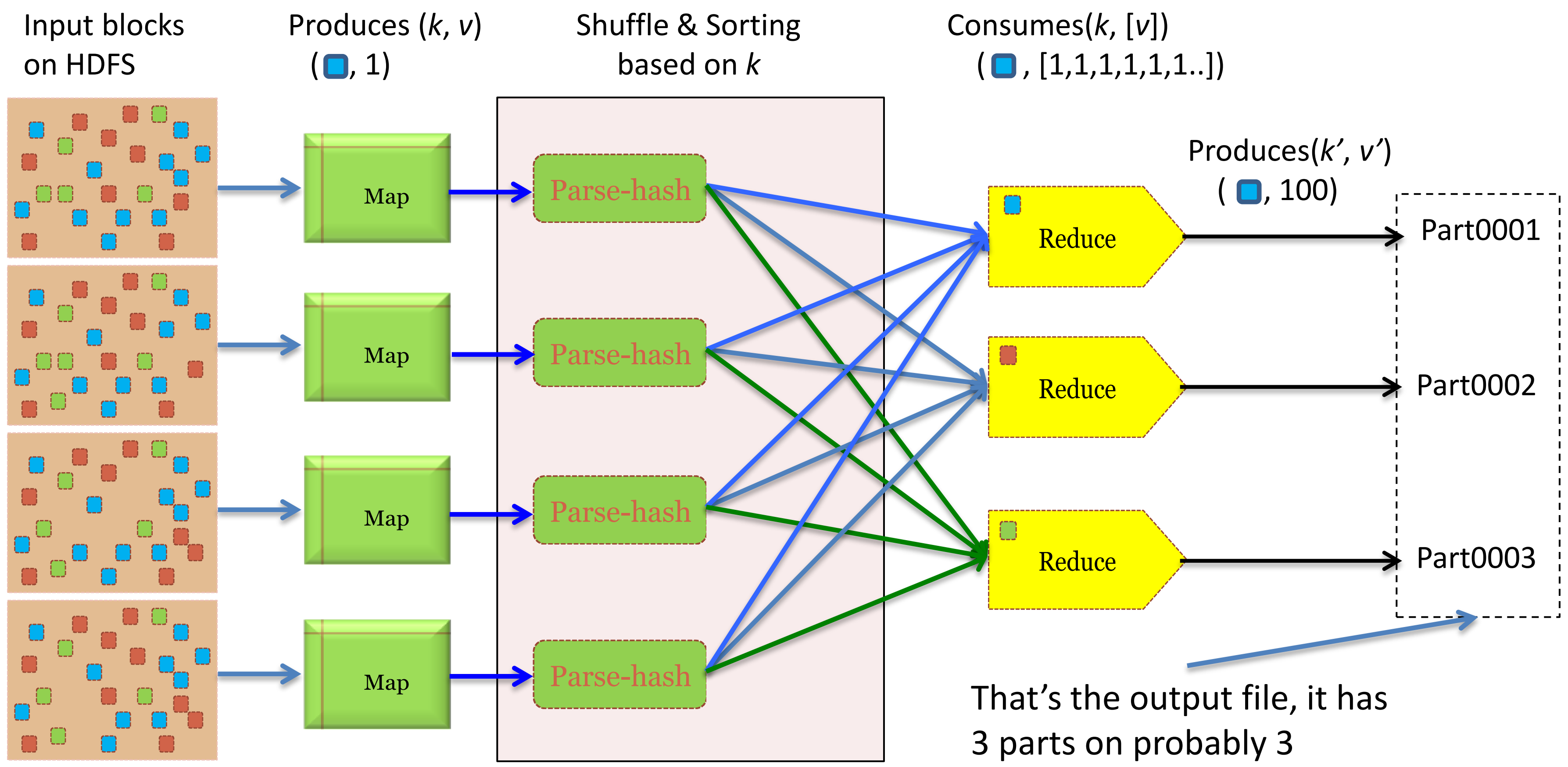


Map Tasks

Reduce Tasks

Example 2: Color Count

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

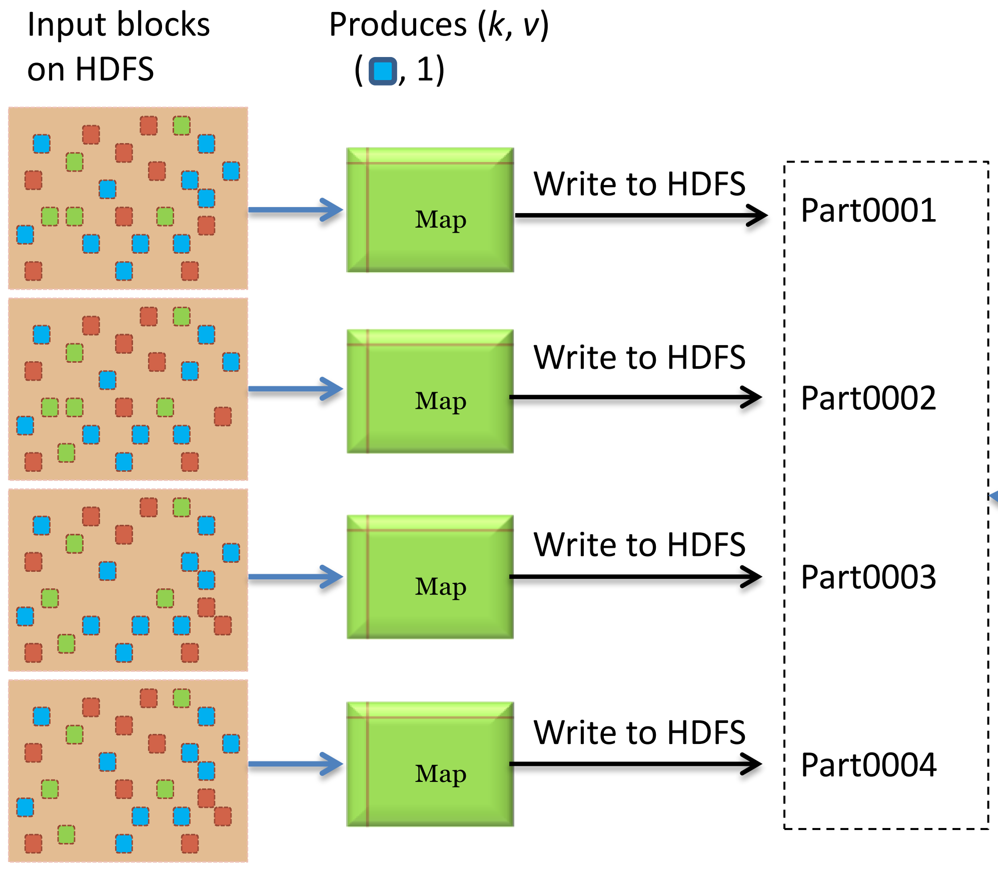


That’s the output file, it has 3 parts on probably 3 different machines

Example 3: Color Filter

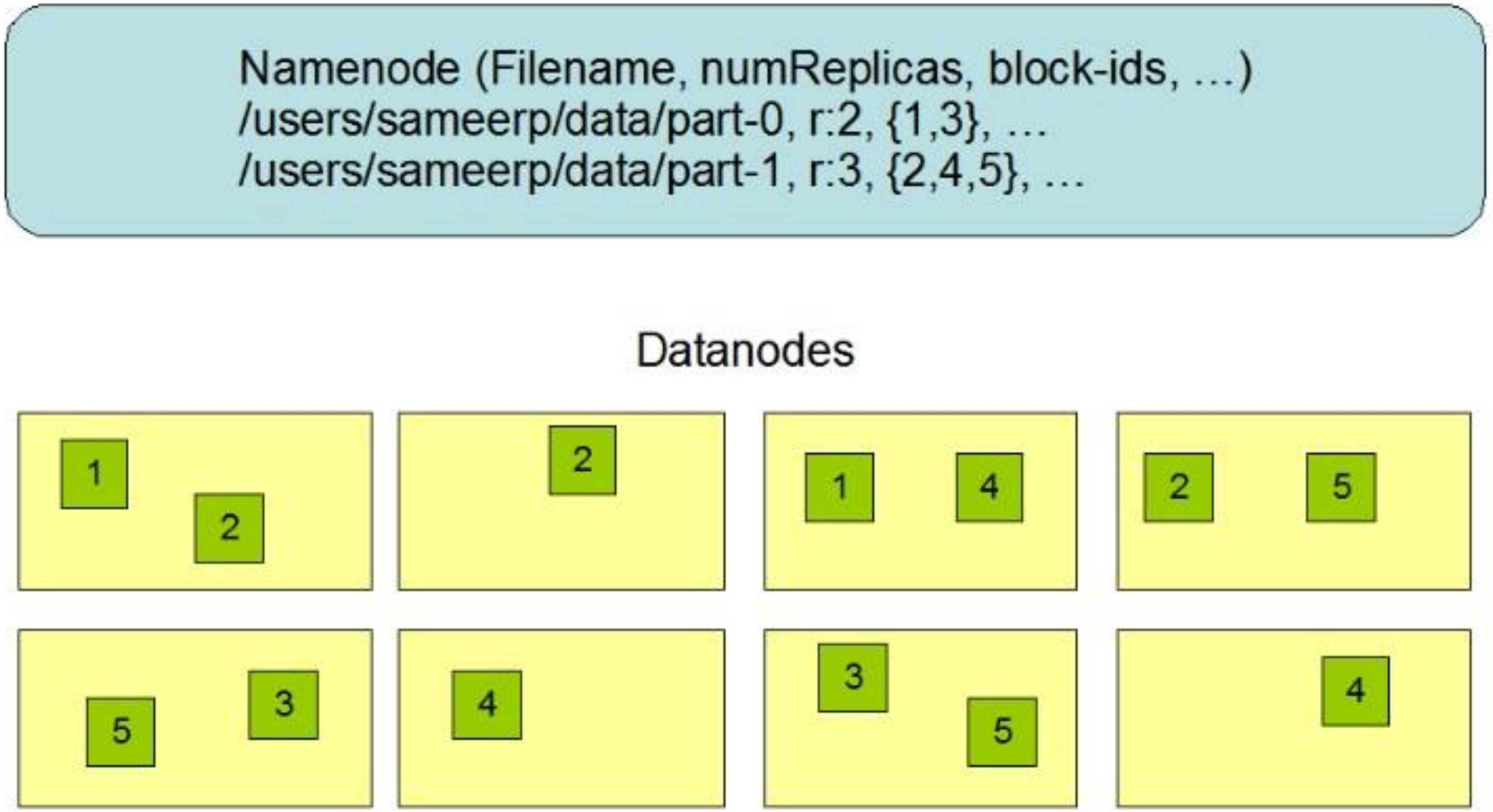
Job: Select only the blue and the green colors

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

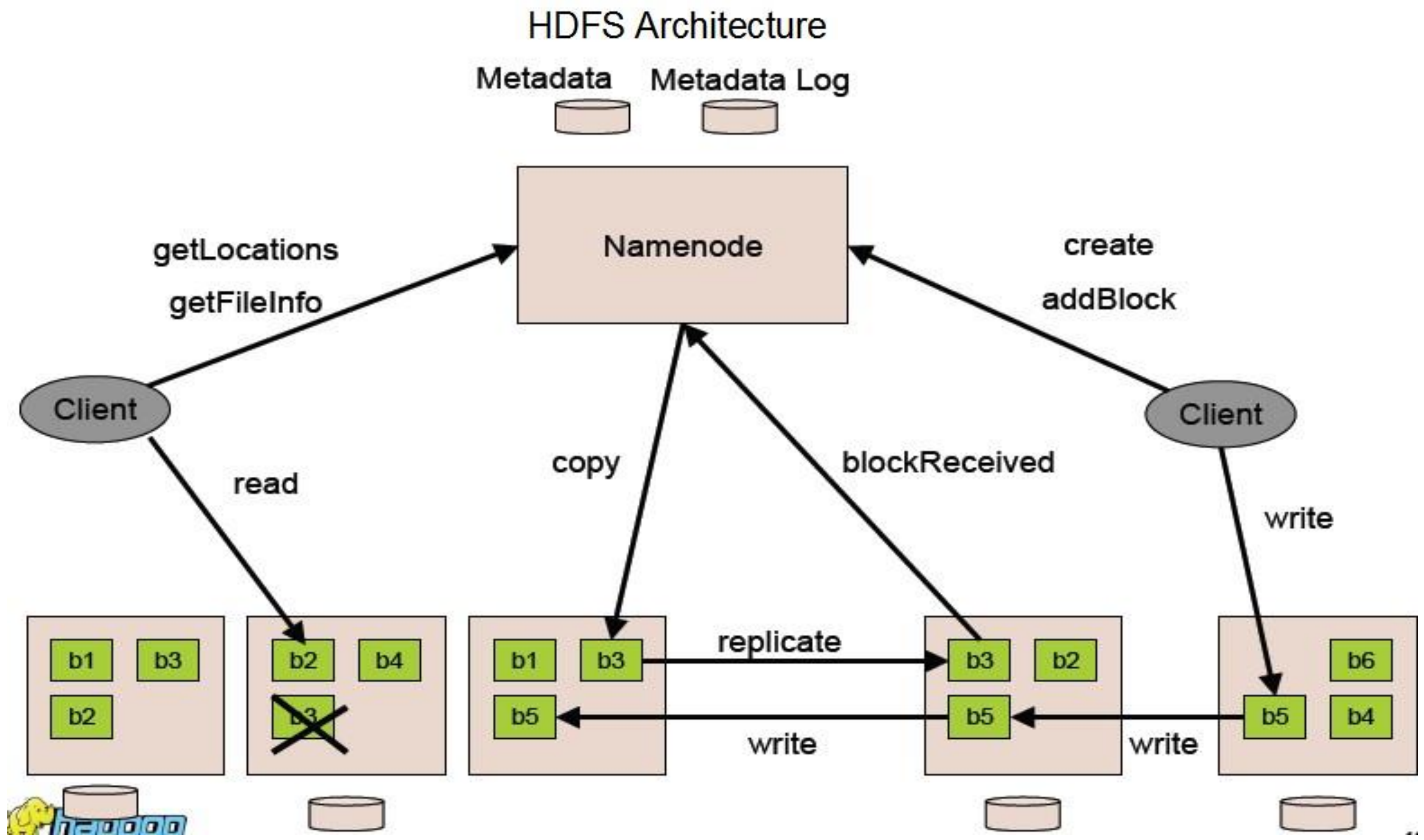


That's the output file, it has 4 parts on probably 4 different machines

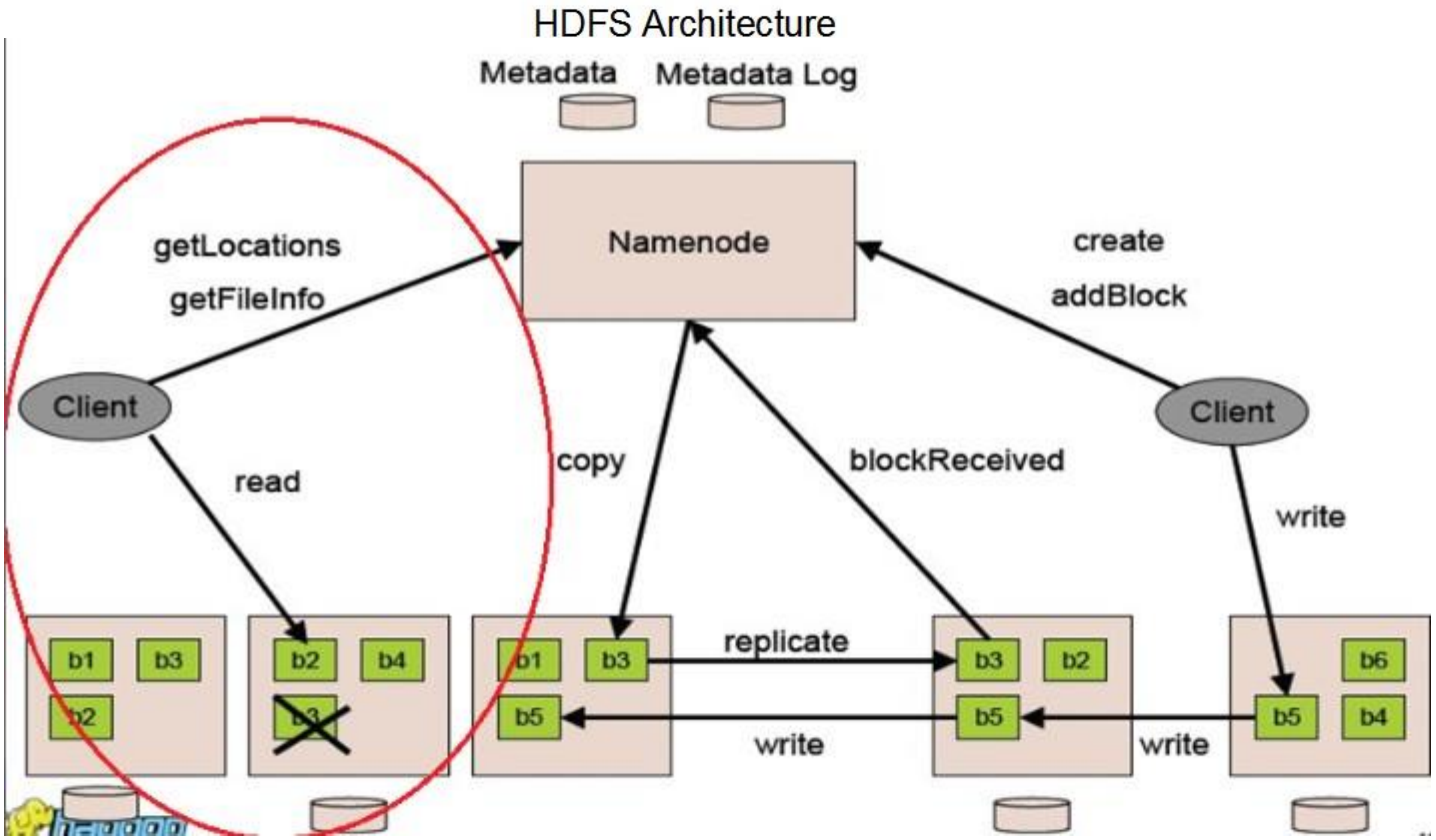
Architecture


Namenode (Filename, numReplicas, block-ids, …)
/users/sameerp/data/part-0, r:2, {1,3}, …
/users/sameerp/data/part-1, r:3, {2,4,5}, …
Datanodes
1
2
2
1
4
2
5
5
3
4
3
5
4

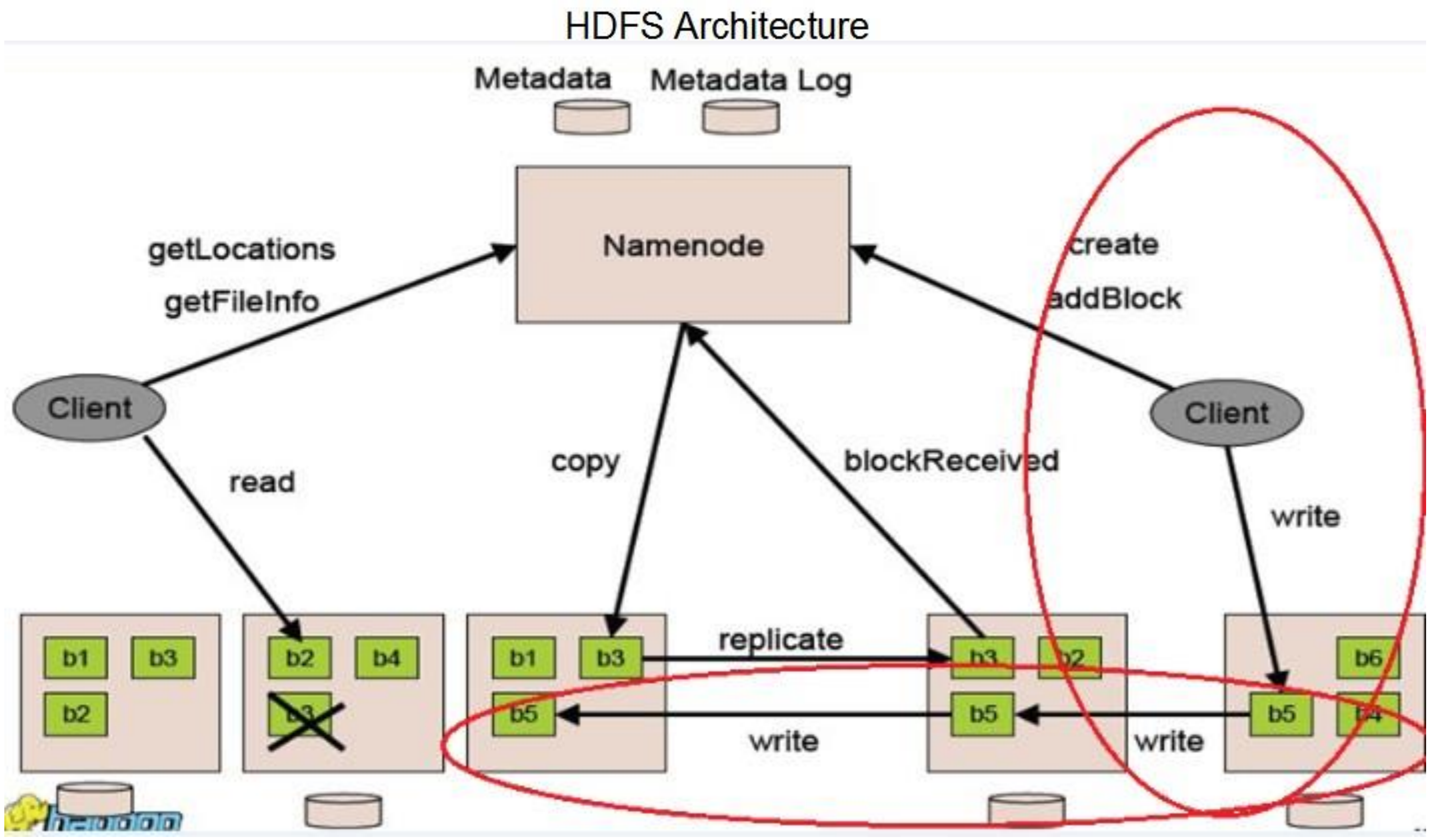
Architecture



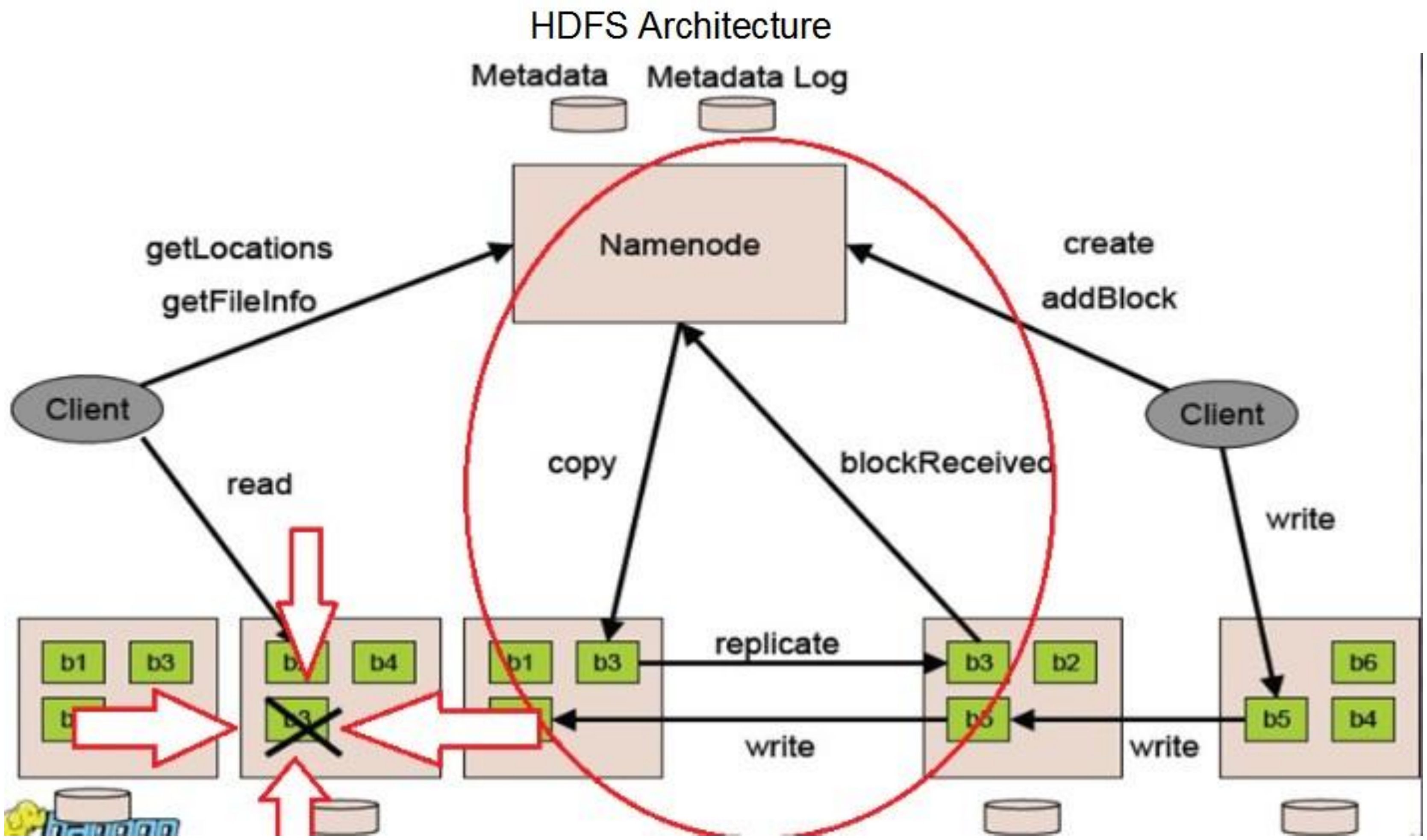
Architecture



Architecture



Architecture

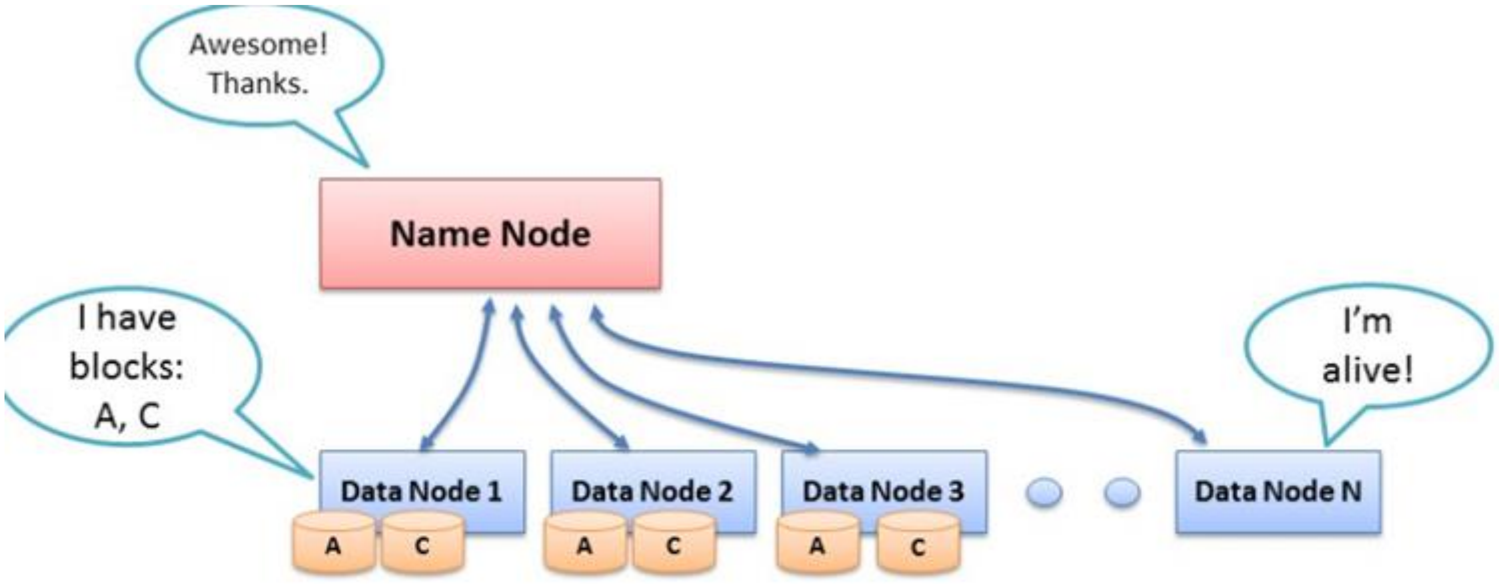


Architecture

- NameNode and DataNode communication: *Heartbeats.*

- DataNodes send *heartbeats* to the NameNode to confirm that **the DataNode is operating** and **the block replicas it hosts are available.**

Architecture



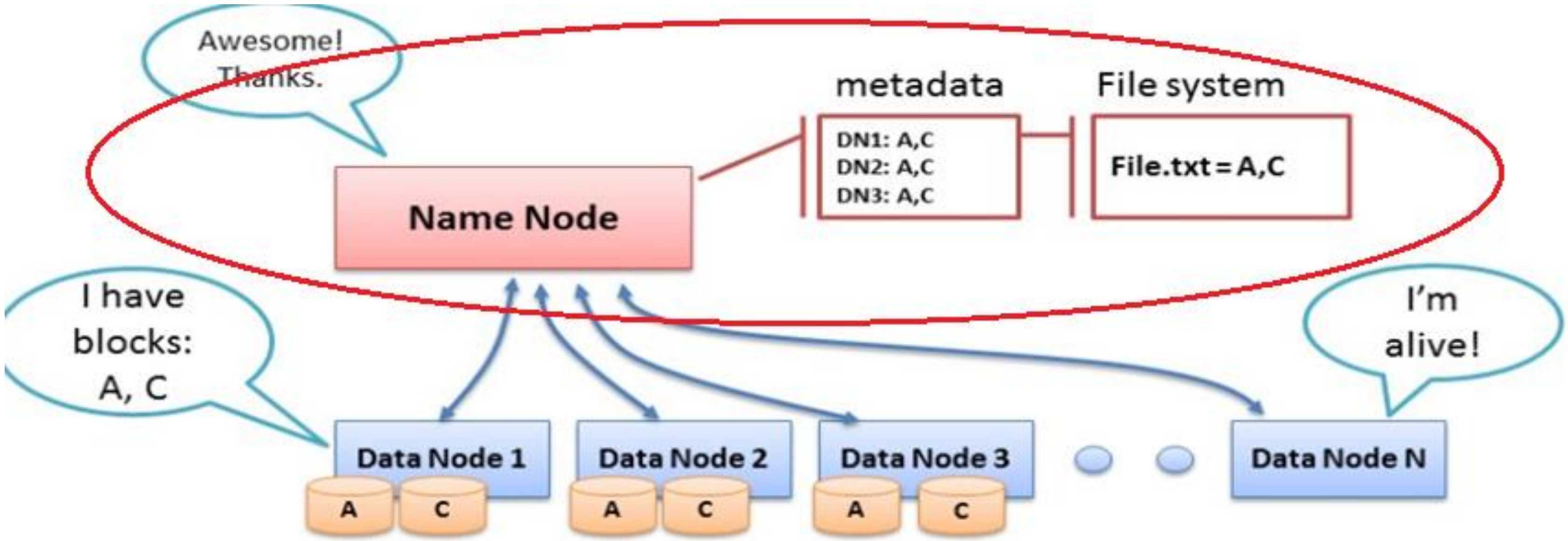
- Data Node sends Heartbeats
- Every 10th heartbeat is a Block report
- Name Node builds metadata from Block reports
- TCP – every 3 seconds

Architecture

- Blockreports:

- A DataNode identifies block replicas in its possession to the NameNode by sending a *block report*. A block report contains the ***block id***, the ***generation stamp*** and the **length for each block replica** the server hosts.
- Blockreports provide the NameNode *with an up-to-date view* of where *block replicas* are located on the cluster and nameNode constructs and maintains latest metadata from blockreports.

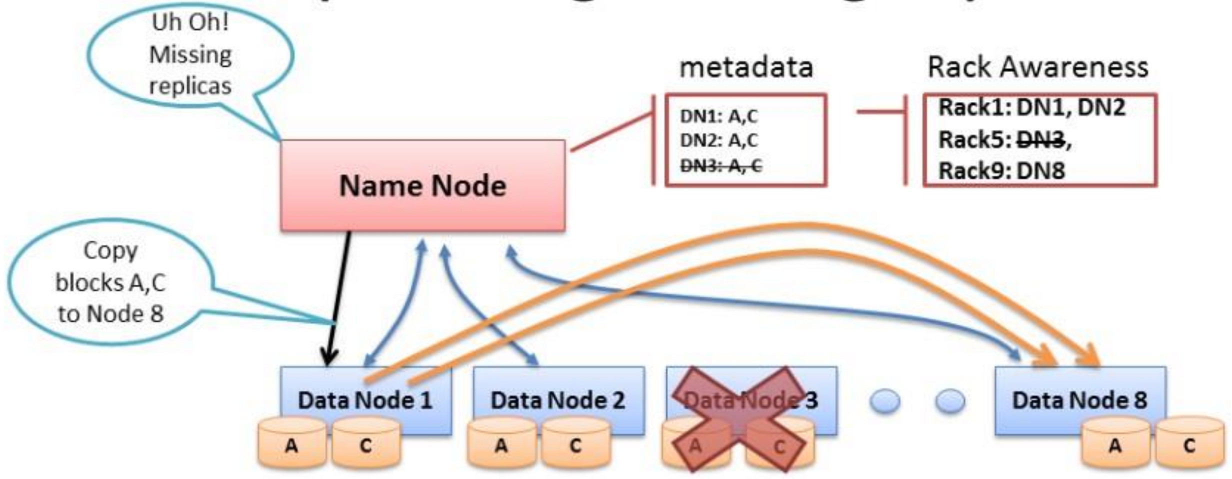
Architecture



- Data Node sends Heartbeats
- Every 10th heartbeat is a Block report
- Name Node builds metadata from Block reports
- TCP – every 3 seconds

Architecture

Re-replicating missing replicas



- Missing Heartbeats signify lost Nodes
- Name Node consults metadata, finds affected data
- Name Node consults Rack Awareness script
- Name Node tells a Data Node to re-replicate

Architecture

- failure recovery

So when dataNode died, NameNode will notice and instruct other dataNode to replicate data to new dataNode. What if NameNode died?

Architecture

- failure recovery

- Keep journal (the modification log of metadata).
- Checkpoint: The persistent record of the metadata stored in the local host’s native files system.

For example:

During restart, the NameNode initializes the namespace image from the checkpoint, and then replays changes from the journal until the image is up-to-date with the last state of the file system.

Architecture

- failure recovery
- **CheckpointNode** and **BackupNode**--two other roles of NameNode

- **CheckpointNode:**
- When journal becomes too long, checkpointNode combines the existing checkpoint and journal to create a new checkpoint and an empty journal.

Architecture

- failure recovery
- **CheckpointNode** and **BackupNode**--two other roles of NameNode

- **BackupNode:** A read-only NameNode
- it maintains an *in-memory*, up-to-date image of the file system namespace that is always synchronized with the state of the NameNode.
- If the NameNode fails, the BackupNode's image in memory and the checkpoint on disk is a record of the latest namespace state.

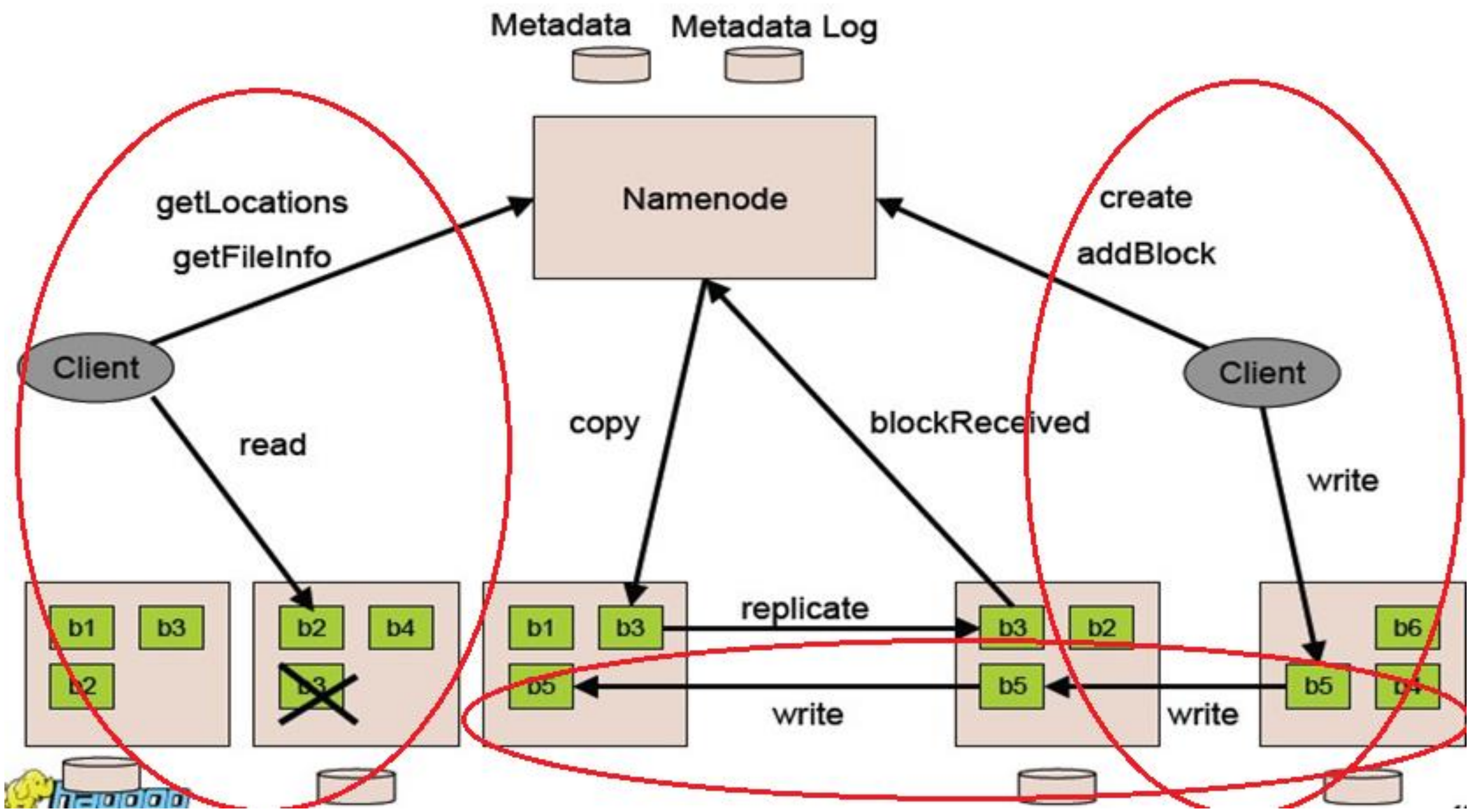
Architecture

- failure recovery

- Upgrades, File System Snapshots
- **The purpose of creating snapshots** in HDFS is to minimize potential damage to the data stored in the system during upgrades. During software upgrades the possibility of corrupting the system due to software bugs or human mistakes increases.
- The snapshot mechanism lets administrators **persistently**

save the current state of the file system(both data and metadata), so that if the upgrade results in data loss or corruption, it is possible to **rollback the upgrade** and return HDFS to the namespace and storage state as they were at the time of the snapshot.

File I/O Operations and Replica Management



File I/O Operations and Replica Management

- Hadoop has the concept of “Rack Awareness”.

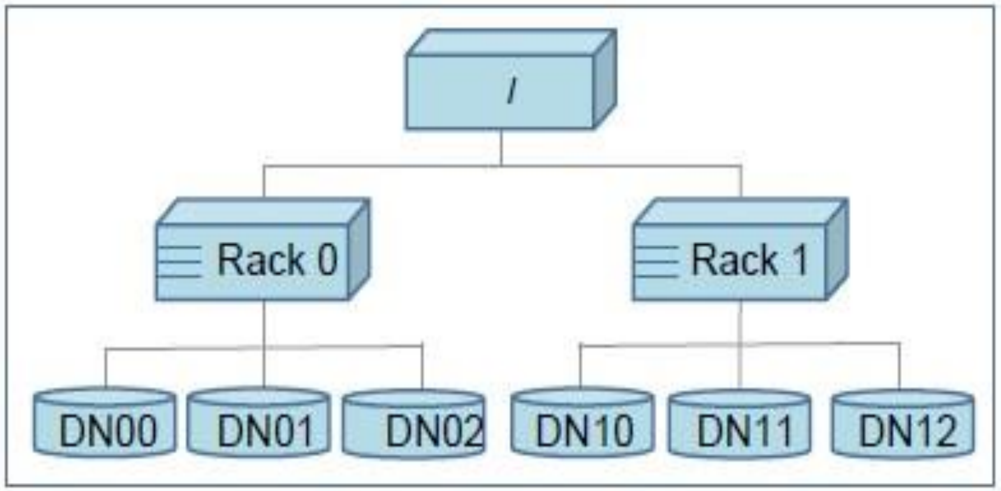


Figure 3. Cluster topology example

File I/O Operations and Replica Management

- Hadoop has the concept of “Rack Awareness”.

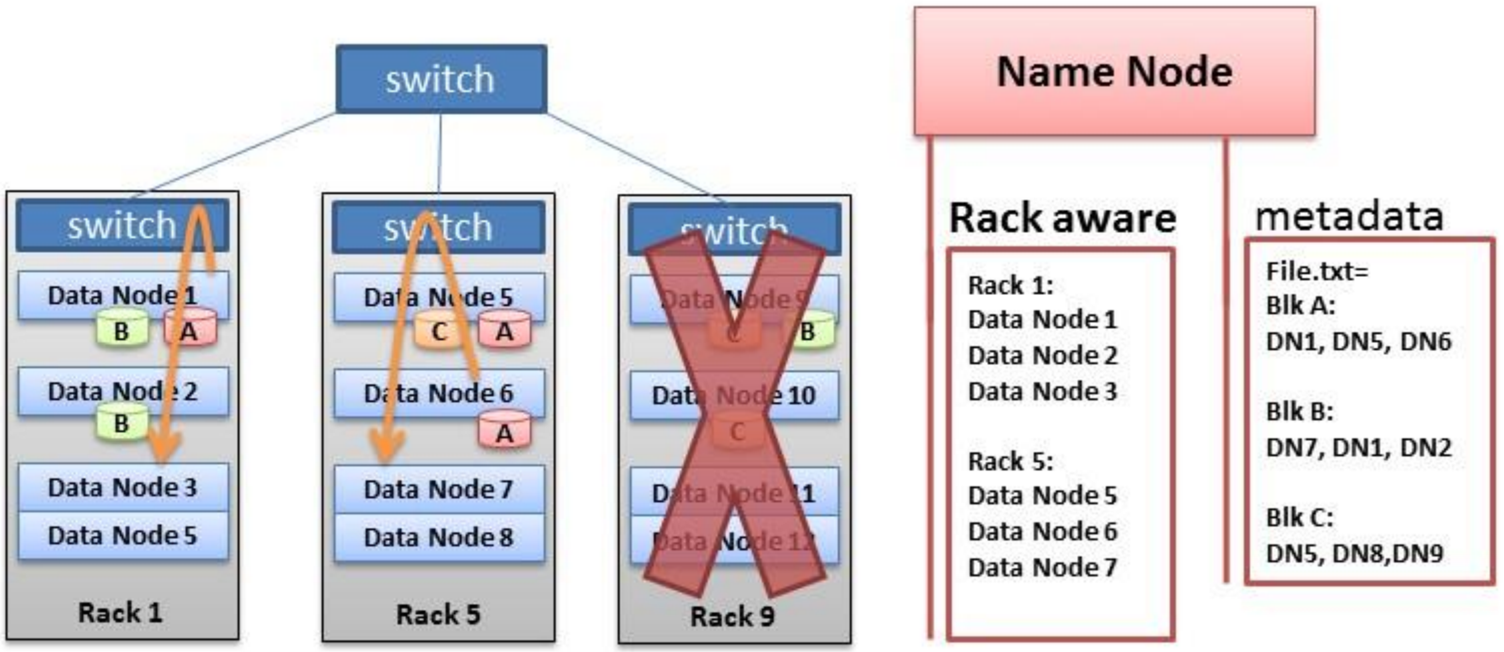
- The default HDFS replica placement policy can be summarized as follows:

  1. No Datanode contains more than one replica of any block.

  2. No rack contains more than two replicas of the same block, provided there are sufficient racks on the cluster.

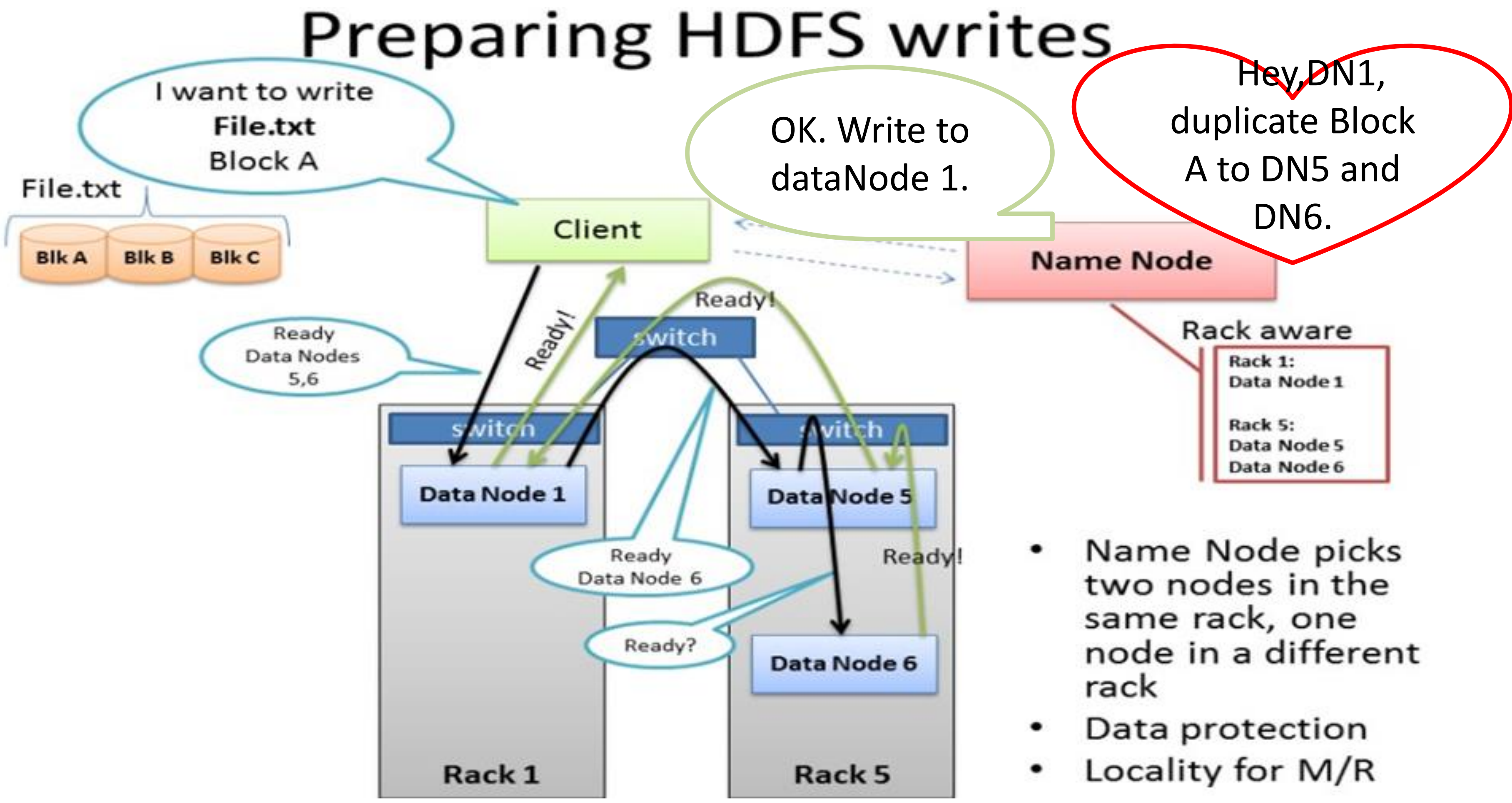
File I/O Operations and Replica Management

Hadoop Rack Awareness – Why?



- Never loose all data if entire rack fails

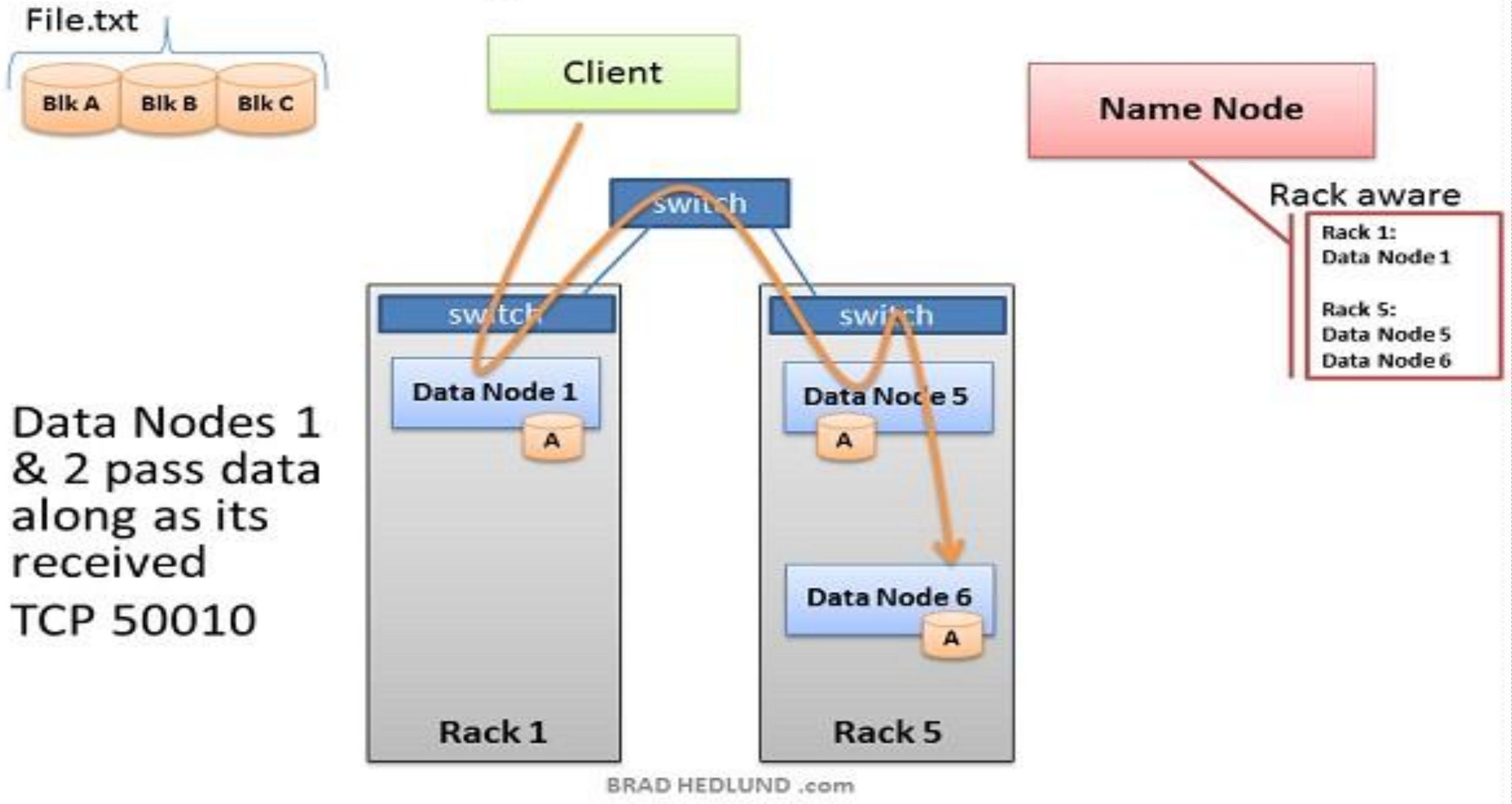
File I/O Operations and Replica Management



- Name Node picks two nodes in the same rack, one node in a different rack
- Data protection
- Locality for M/R

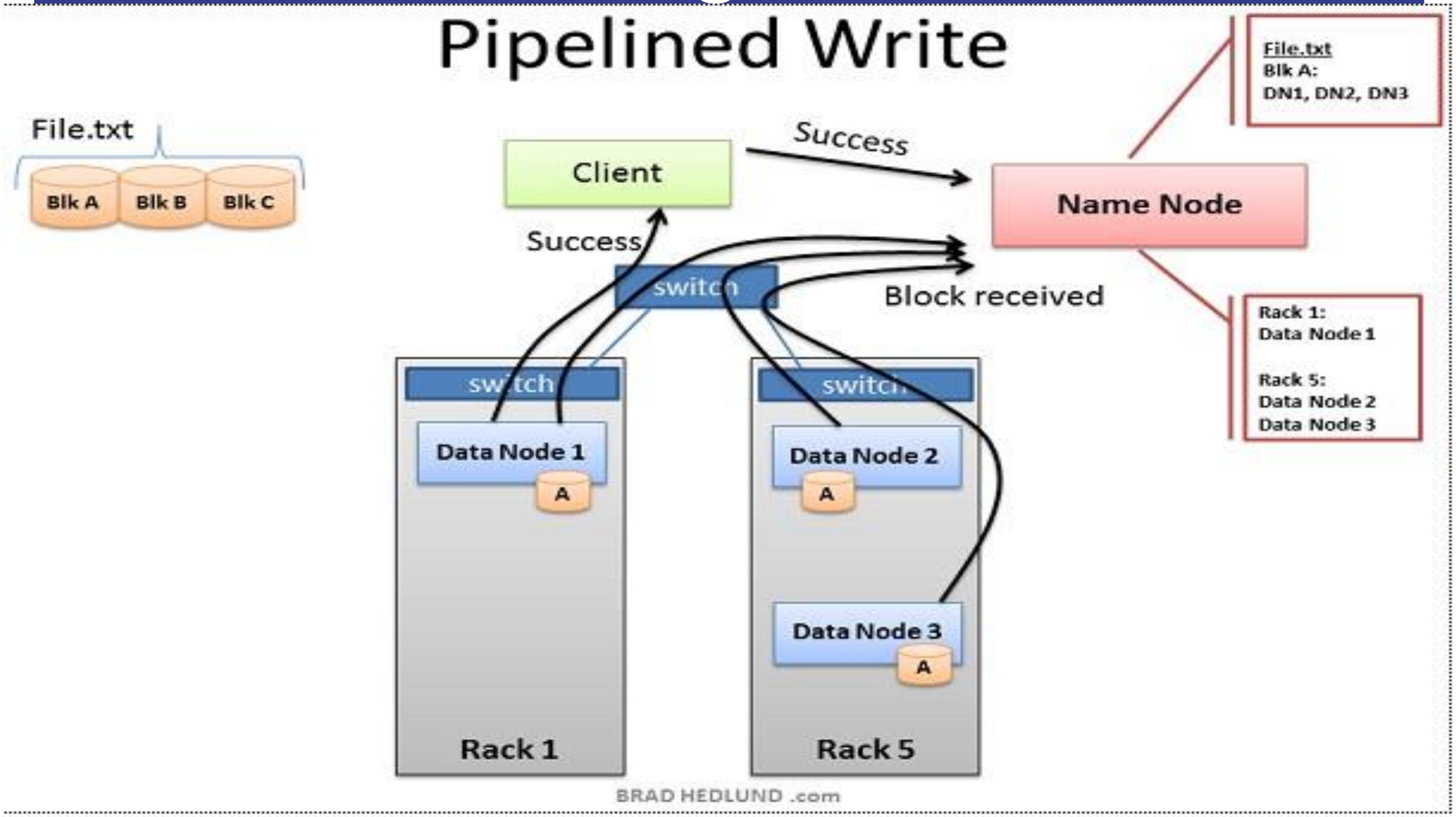
File I/O Operations and Replica Management

Pipelined Write

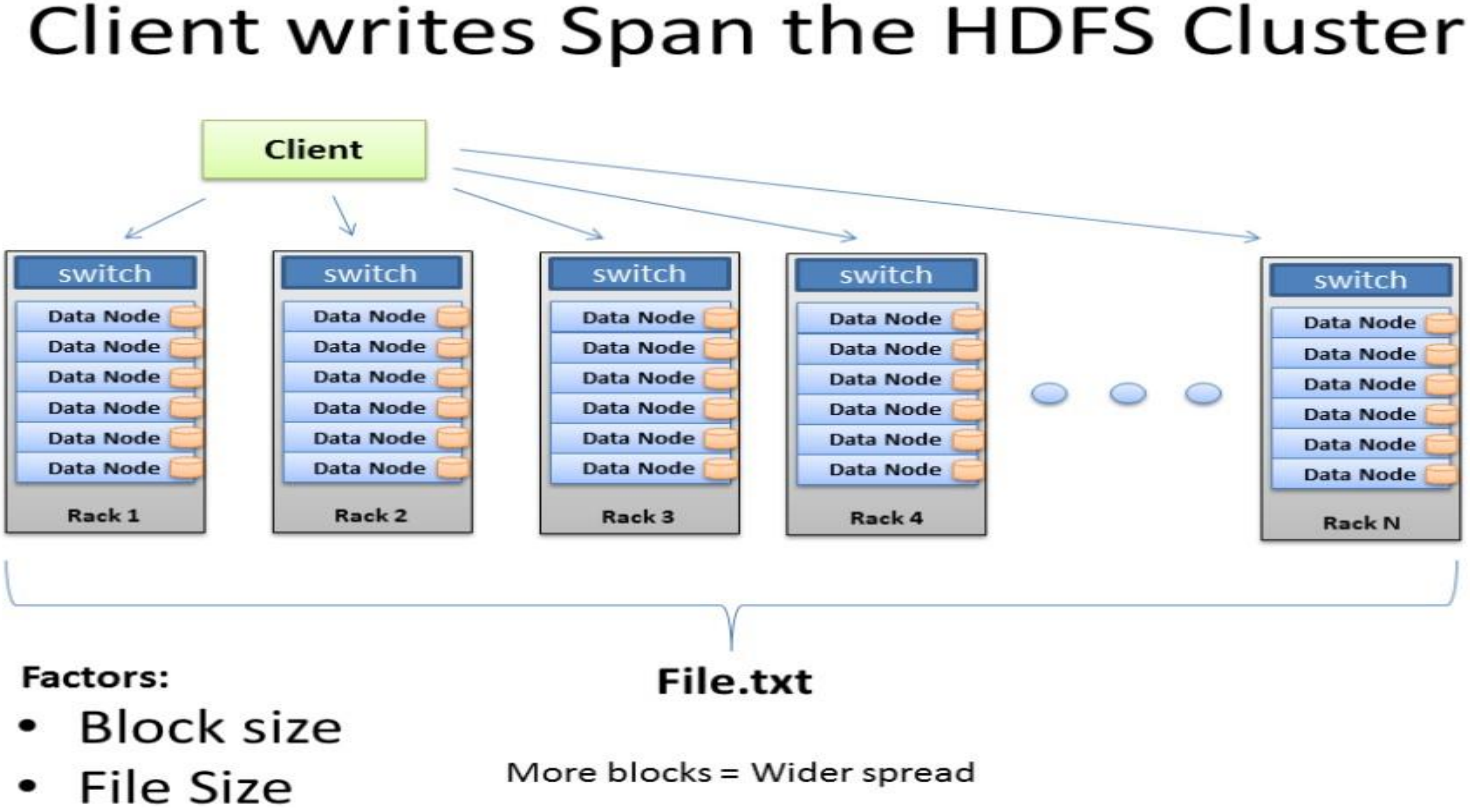


- Data Nodes 1 & 2 pass data along as its received
- TCP 50010

File I/O Operations and Replica Management

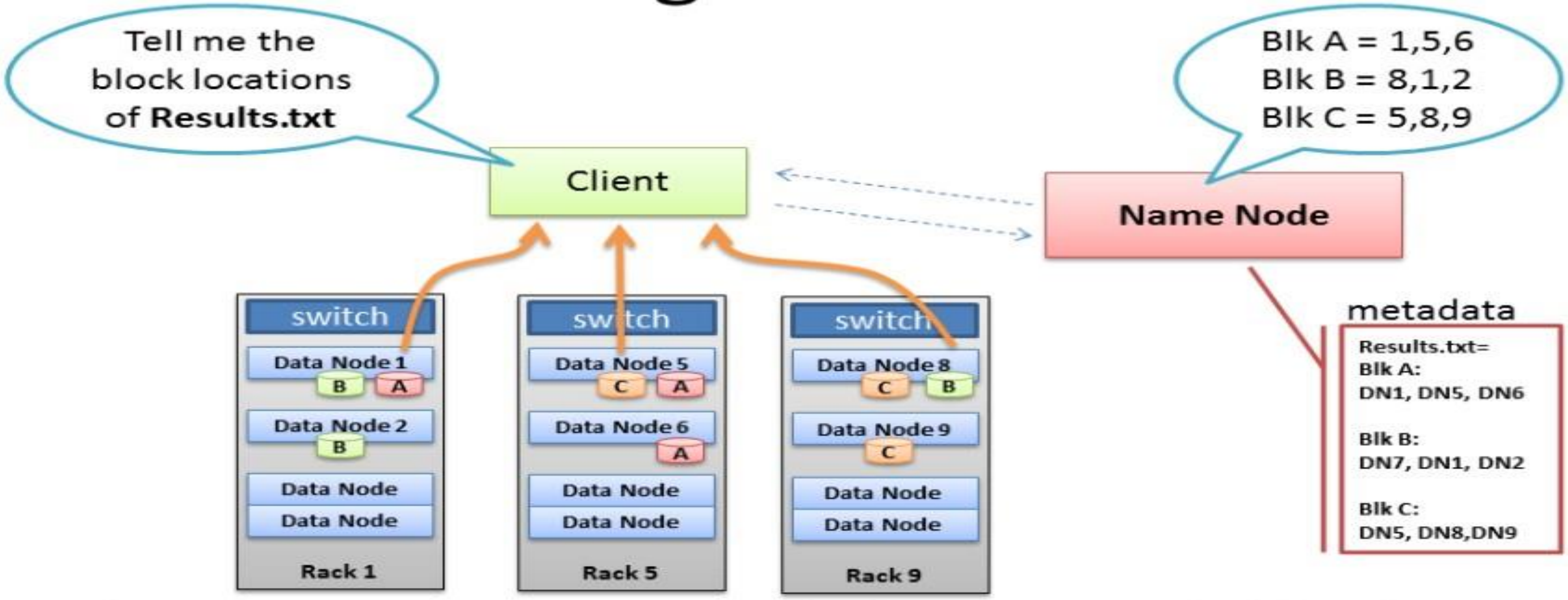


File I/O Operations and Replica Management



File I/O Operations and Replica Management

Client reading files from HDFS

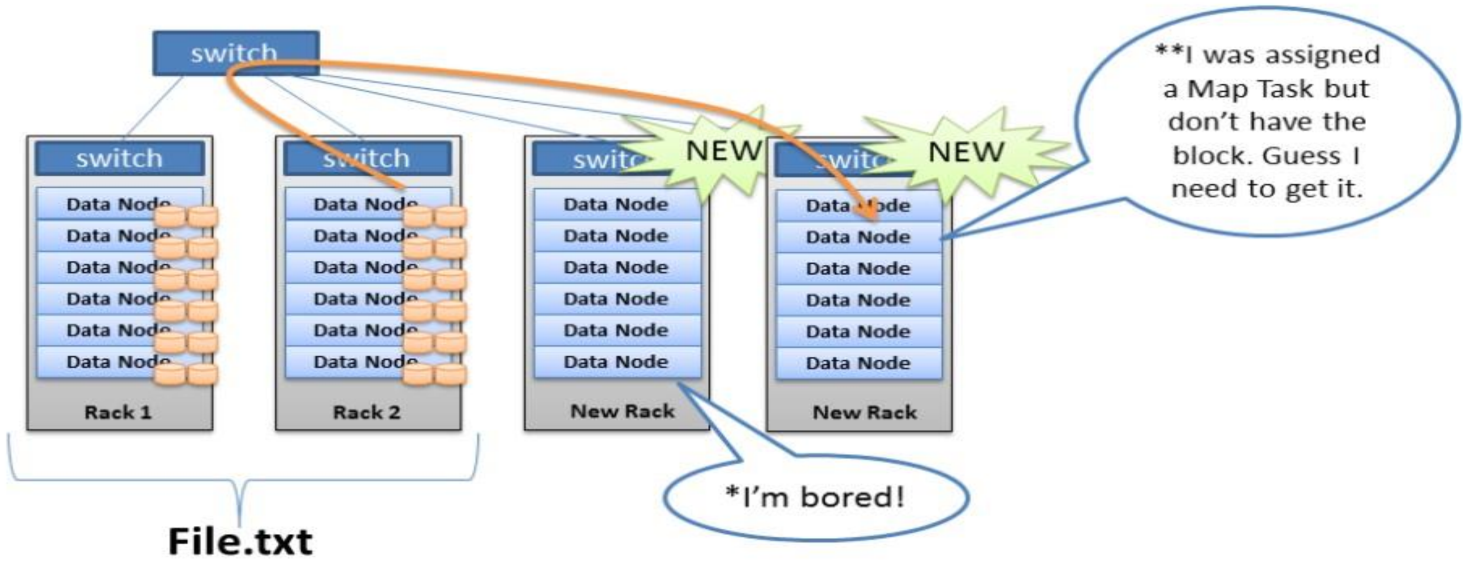


- Client receives Data Node list for each block
- Client picks first Data Node for each block
- Client reads blocks sequentially

File I/O Operations and Replica Management

- Balancer

Unbalanced Cluster

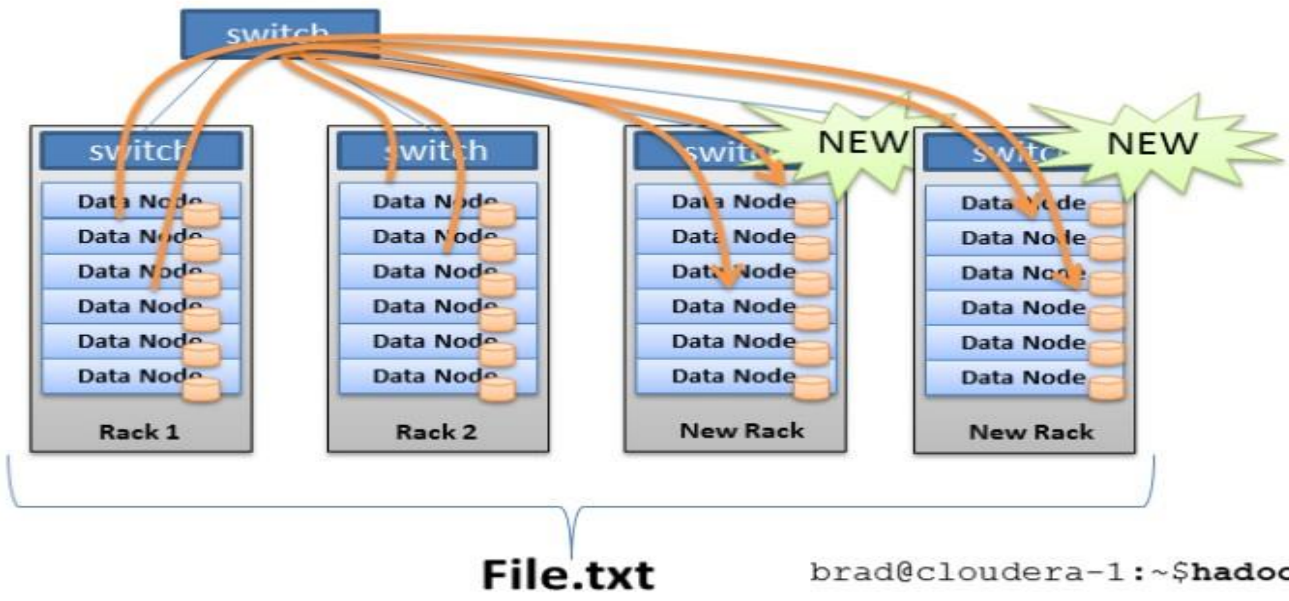

switch
switch
Data Node
Rack 1
Rack 2
New Rack
NEW
NEW
**I was assigned a Map Task but don't have the block. Guess I need to get it.
*I'm bored!
File.txt

- Hadoop prefers local processing <u>if possible</u>
- New servers underutilized for Map Reduce, HDFS*
- More network bandwidth, slower job times**

File I/O Operations and Replica Management

- Balancer

Cluster Balancing



```
brad@cloudera-1:~$hadoop balancer
```

- Balancer utility (if used) runs in the background
- Does not interfere with Map Reduce or HDFS
- Default rate limit 1 MB/s

Practice at YAHoo!

- **HDFS clusters at Yahoo! include about 3500 nodes**
- **A typical cluster node has:**

- · 2 quad core Xeon processors @ 2.5ghz
- · Red Hat Enterprise Linux Server Release 5.1
- · Sun Java JDK 1.6.0_13-b03
- · 4 directly attached SATA drives (one terabyte each)
- · 16G RAM
- · 1-gigabit Ethernet

Practice at YAHoo!

- 70 percent of the disk space is allocated to HDFS. The remainder is reserved for the operating system (Red Hat Linux), logs, and space to spill the *output of map tasks*. **(MapReduce intermediate data are not stored in HDFS.)**

- For each cluster, the NameNode and the BackupNode hosts are specially provisioned with up to 64GB RAM; application tasks are never assigned to those hosts.

- In total, a cluster of 3500 nodes has 9.8 PB of storage available as blocks that are replicated three times yielding a net 3.3 PB of storage for user applications. As a convenient approximation, one thousand nodes represent one PB of application storage.

Practice at YAHoo!

- **Durability of Data**

➢ uncorrelated node failures

Replication of data *three times* is a robust guard against loss of data due to uncorrelated node failures.

➢ correlated node failures, the failure of a rack or core switch.

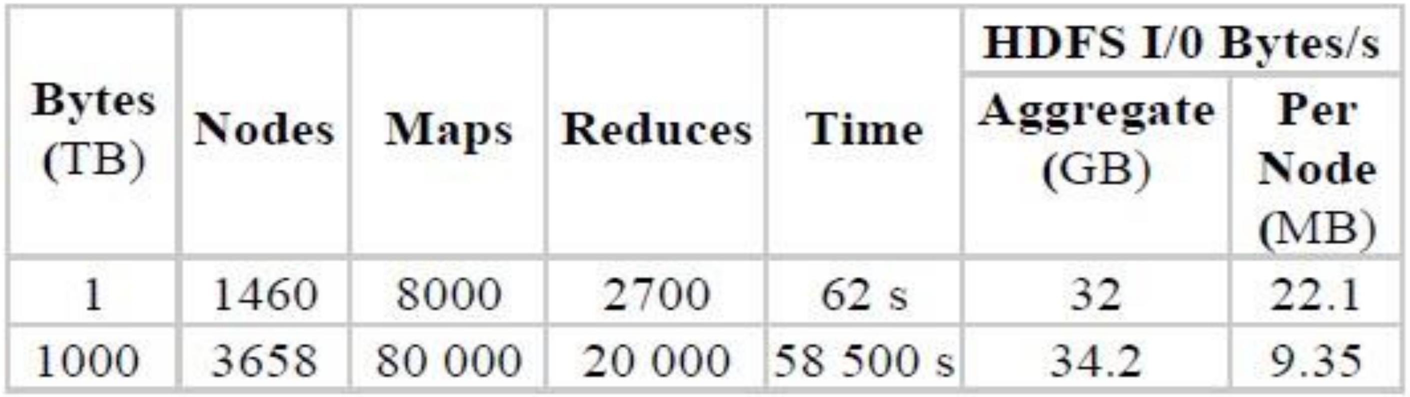
HDFS can tolerate losing a rack switch (each block has a replica on some other rack).

➢ loss of electrical power to the cluster

a large cluster will lose a handful of blocks during a power-on restart.

Practice at YAHoo!

- Benchmarks

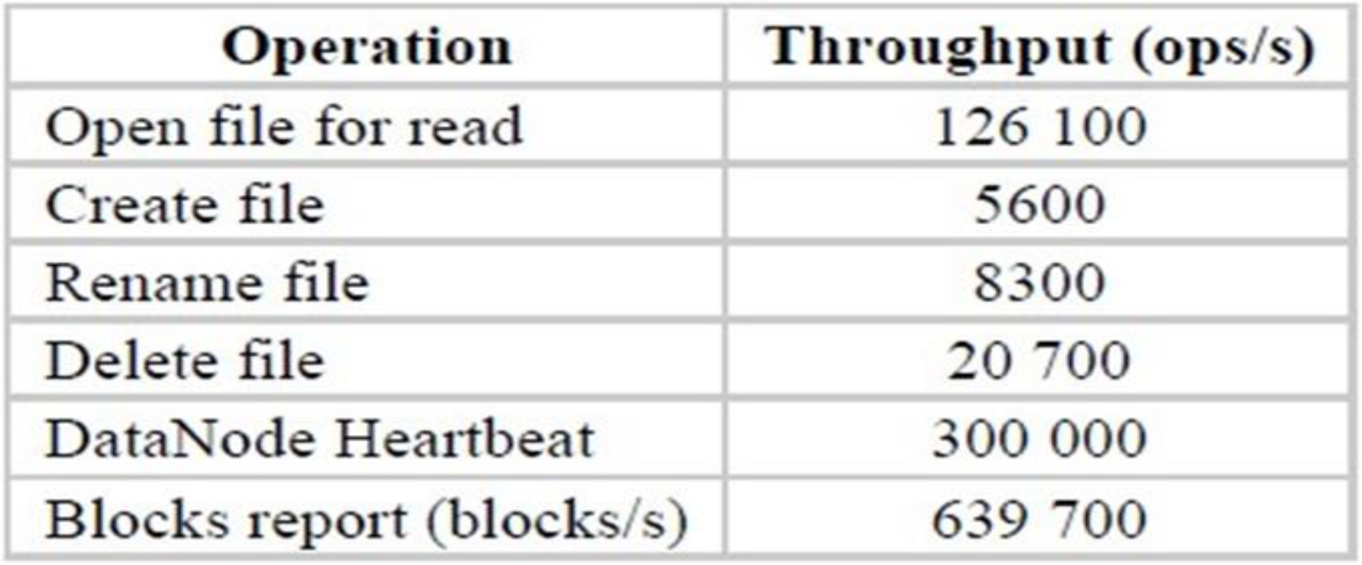


Bytes (TB)	Nodes	Maps	Reduces	Time	HDFS I/0 Bytes/s	
					Aggregate (GB)	Per Node (MB)
1	1460	8000	2700	62 s	32	22.1
1000	3658	80 000	20 000	58 500 s	34.2	9.35

Table 2. Sort benchmark for one terabyte and one petabyte of data. Each data record is 100 bytes with a 10-byte key. The test program is a general sorting procedure that is not specialized for the record size. In the terabyte sort, the block replication factor was set to one, a modest advantage for a short test. In the petabyte sort, the replication factor was set to two so that the test would confidently complete in case of a (not unexpected) node failure.

Practice at YAHoo!

- Benchmarks



Operation	Throughput (ops/s)
Open file for read	126 100
Create file	5600
Rename file	8300
Delete file	20 700
DataNode Heartbeat	300 000
Blocks report (blocks/s)	639 700

NameNode Throughput benchmark

FUTURE WORK

- Automated failover

plan: Zookeeper, Yahoo’s distributed consensus technology to build an automated failover solution

- Scalability of the NameNode

Solution: Our near-term solution to scalability is to allow multiple namespaces (and NameNodes) to share the physical storage within a cluster.

Drawbacks: The main drawback of multiple independent namespaces is the cost of managing them.