Hadoop - HDFS and MapReduce

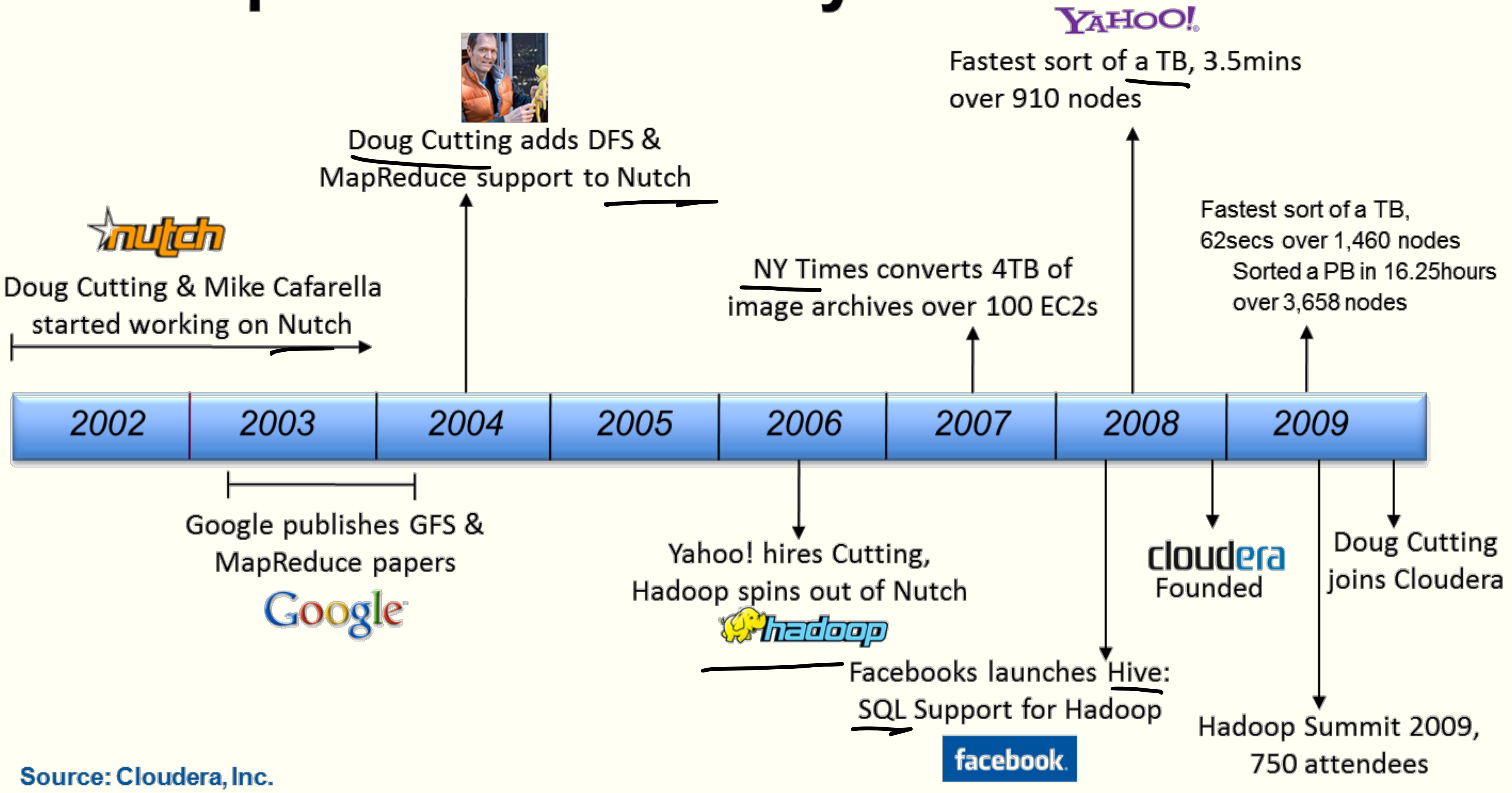
Srini Badri

Hadoop

- Apache top level project, open-source implementation of frameworks for reliable, scalable, distributed computing and data storage.
- A flexible and highly-available architecture for large scale computation and data processing on a network of commodity hardware.
  - open-source implementation for Google MapReduce
  - based on MapReduce
  - based on a simple data model for any data

Hadoop History

Hadoop Creation History



Doug Cutting & Mike Cafarella started working on Nutch

Google publishes GFS & MapReduce papers

Doug Cutting adds DFS & MapReduce support to Nutch

Yahoo! hires Cutting, Hadoop spins out of Nutch

NY Times converts 4TB of image archives over 100 EC2s

Facebooks launches Hive: SQL Support for Hadoop

Fastest sort of a TB, 3.5mins over 910 nodes

Founded

Fastest sort of a TB, 62secs over 1,460 nodes
Sorted a PB in 16.25hours over 3,658 nodes

Doug Cutting joins Cloudera

Hadoop Summit 2009, 750 attendees

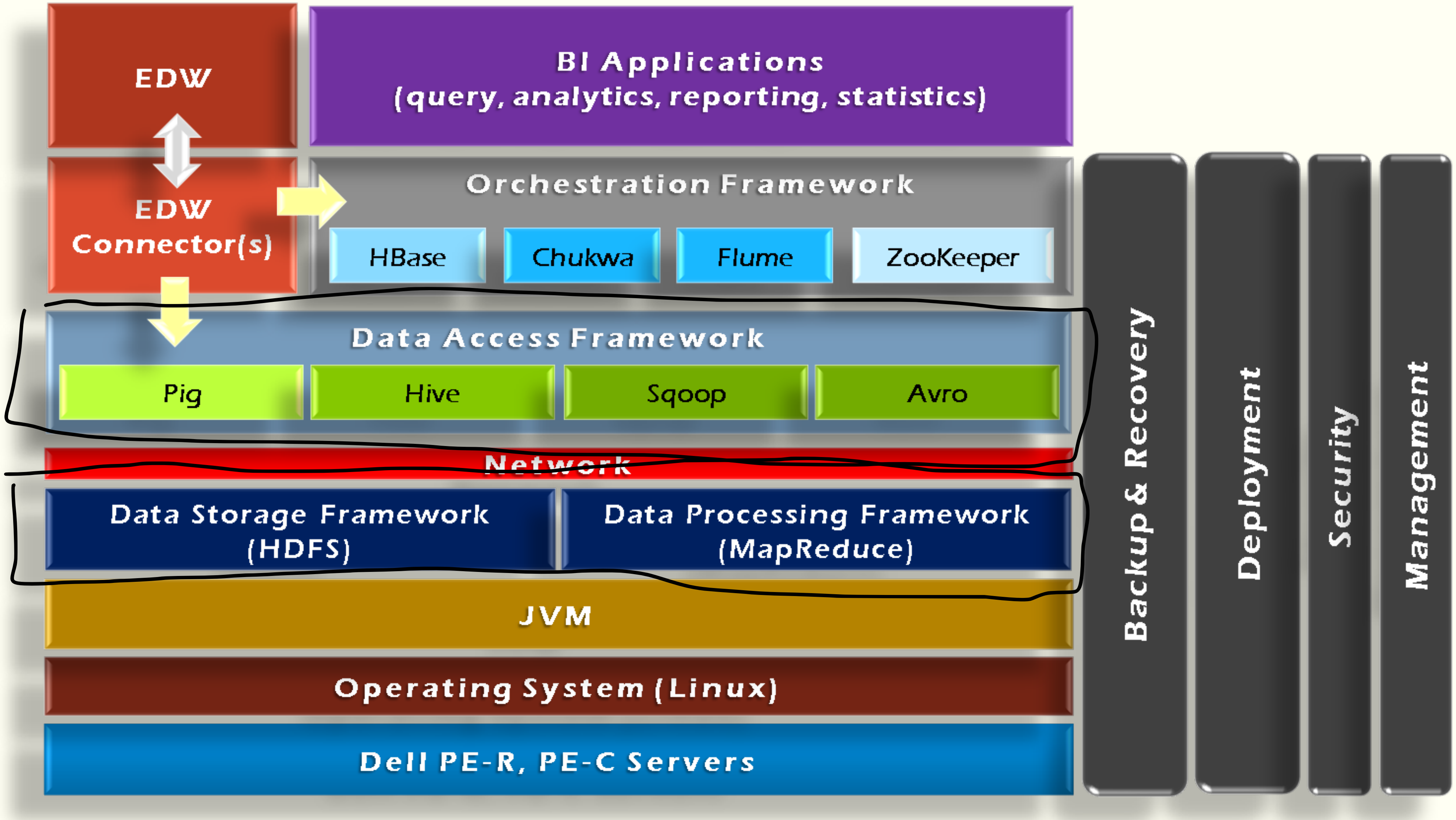
Source: Cloudera, Inc.

Hadoop Vs. DBMS

- Scalability (petabytes of data, thousands of machines)
- Flexibility in accepting all data formats (no schema)
- Simple fault-tolerant mechanism
- Commodity inexpensive hardware

- Performance (indexing, tuning, data organization tech.)
- Features:
  - Provenance tracking
  - Annotation management
  - ….

Hadoop Framework Tools



Design Principles of Hadoop

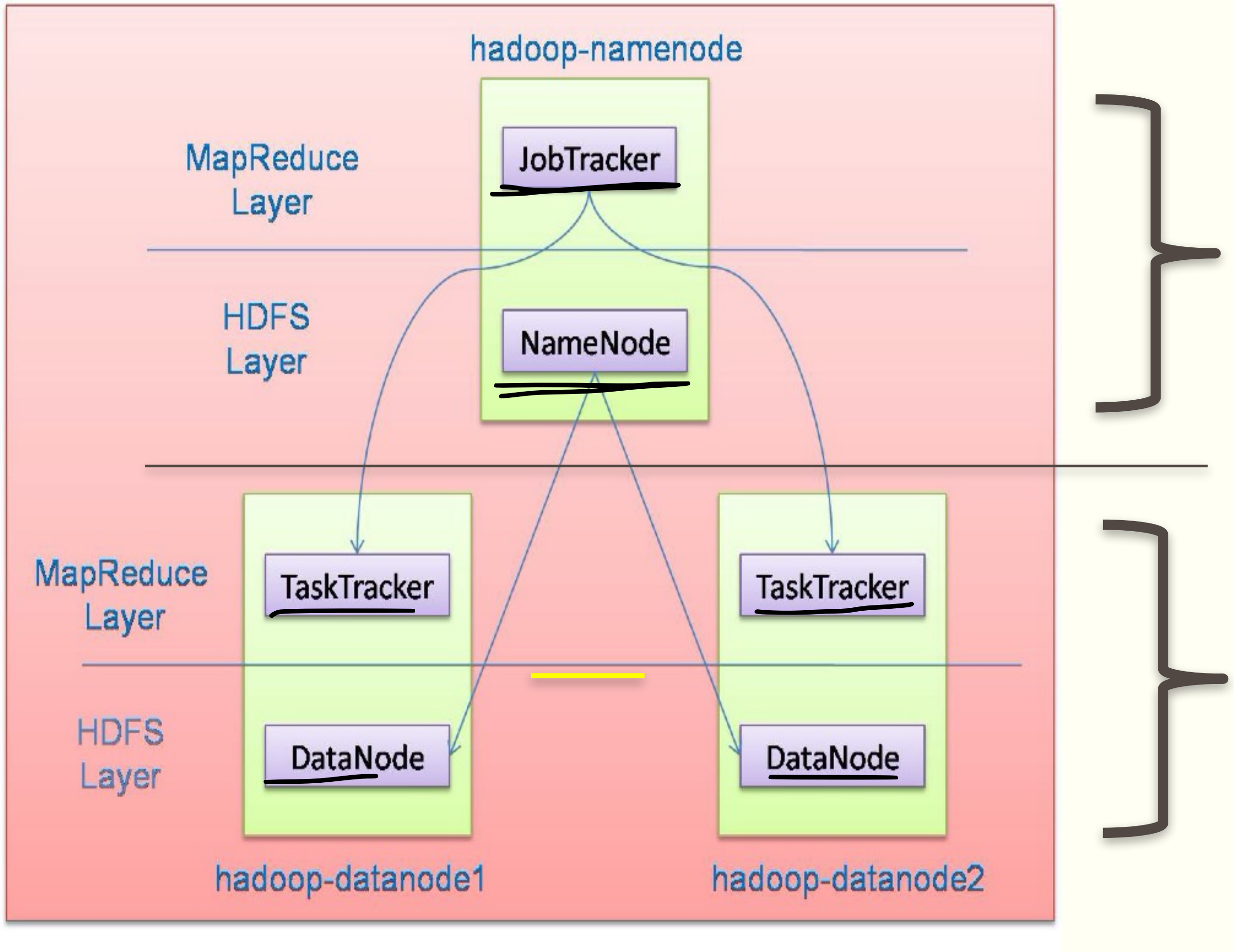
- Need to parallelize computation across thousands of nodes
- Commodity hardware
  - Large number of low-end cheap machines working in parallel to solve a computing problem
  - in contrast to Parallel DBs: Small number of high-end expensive machines
- Automatic parallelization & distribution
  - Hidden 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"

HDFS

- Hadoop Distributed File System
- 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)
- Fault Tolerance: Detection of faults and quick, automatic recovery from them is a core architectural goal of HDFS
  - Namenode is consistently checking Datanodes: The Namenode receives a Heartbeat and a BlockReport from each DataNode in the cluster.

Hadoop: Master/Save Architecture

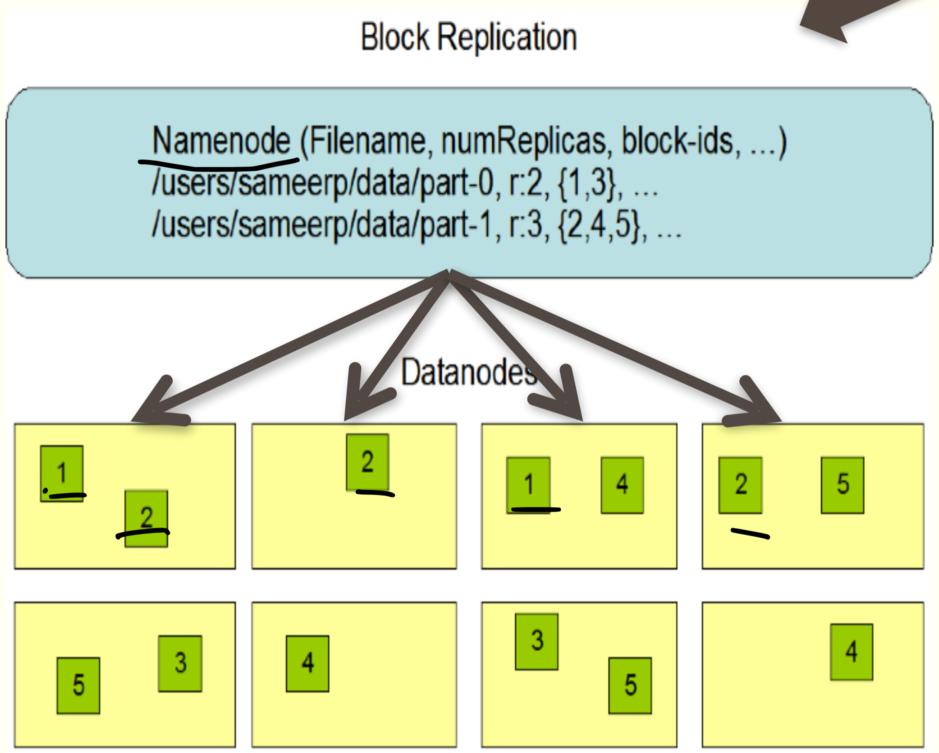
- Hadoop is designed as a master-slave shared-nothing architecture
  - Distributed file system (HDFS)
  - Execution engine (MapReduce)



Master node (single node)

Many slave nodes

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}, ...

Centralized namenode

- Maintains metadata info about files
  FsImage + EditLog

File *F* 1 2 3 4 5

Blocks (64 MB)

Many datanode (1000s)

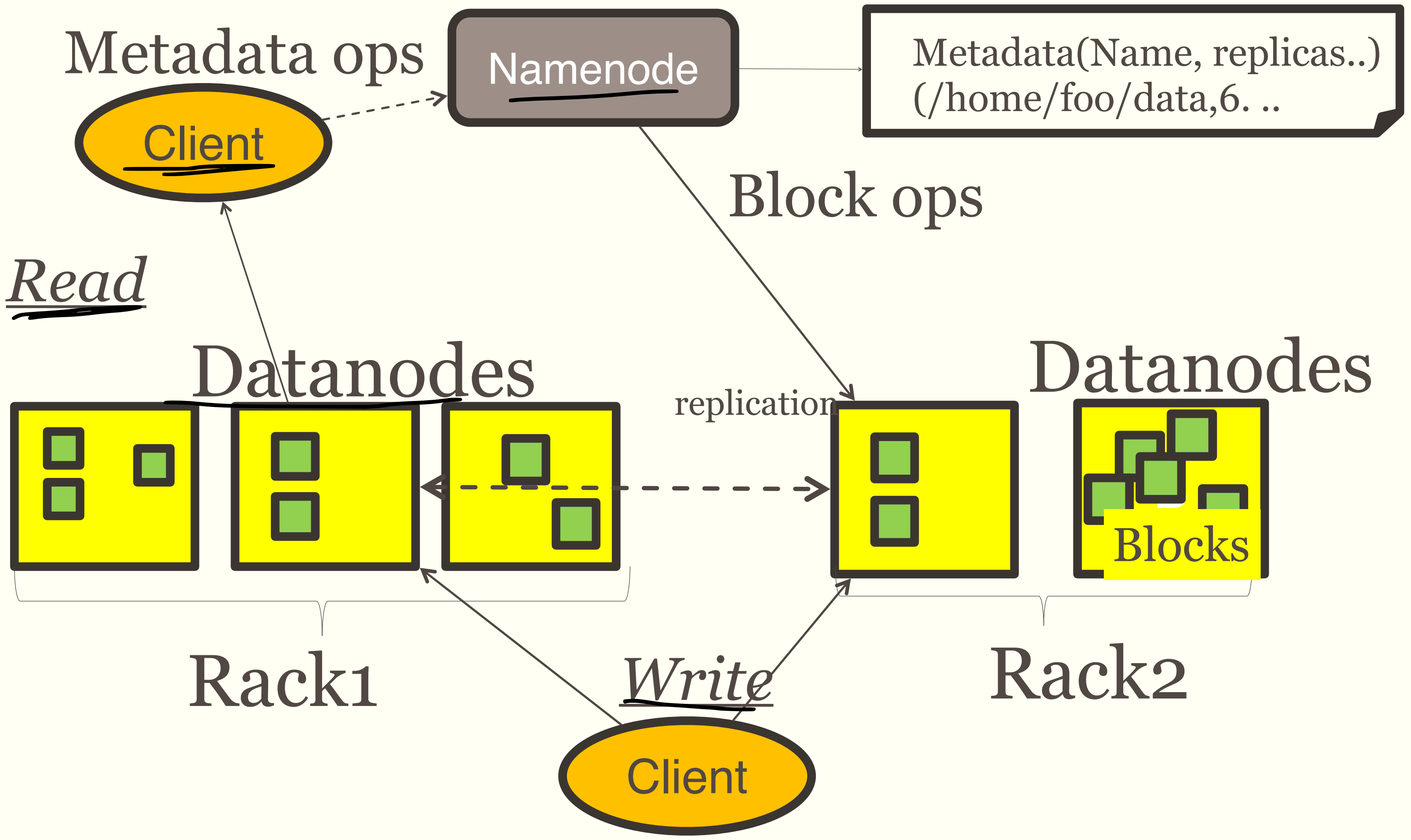
- Store the actual data
- Files are divided into blocks
- Each block is replicated *N* times

(Default = 3)

Functions of “nodes”

- NameNode
    - Manages File System Namespace
        - Maps a file name to a set of blocks
        - Maps a block to the DataNodes where it resides
        - FsImage + EditLog
    - Cluster Configuration Management
    - Replication Engine for Blocks
- DataNode
    - A Block Server: Stores data in the local file system (e.g. ext3); Stores metadata of a block; Serves data and metadata to Clients
    - Block Report
        - Periodically sends a report of all existing blocks to the NameNode
    - Facilitates Pipelining of Data
        - Forwards data to other specified DataNodes

HDFS Architecture

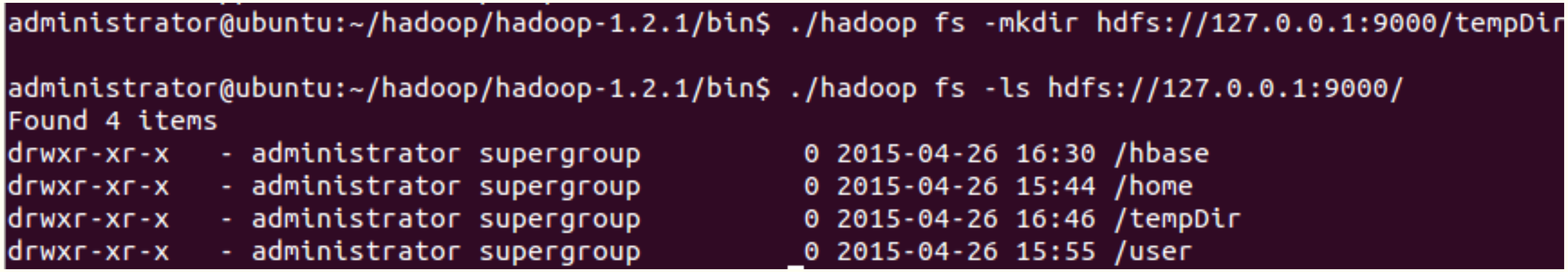


HDFS command

- Shell command: most common: fs

hadoop fs [genericOptions] [commandOptions]

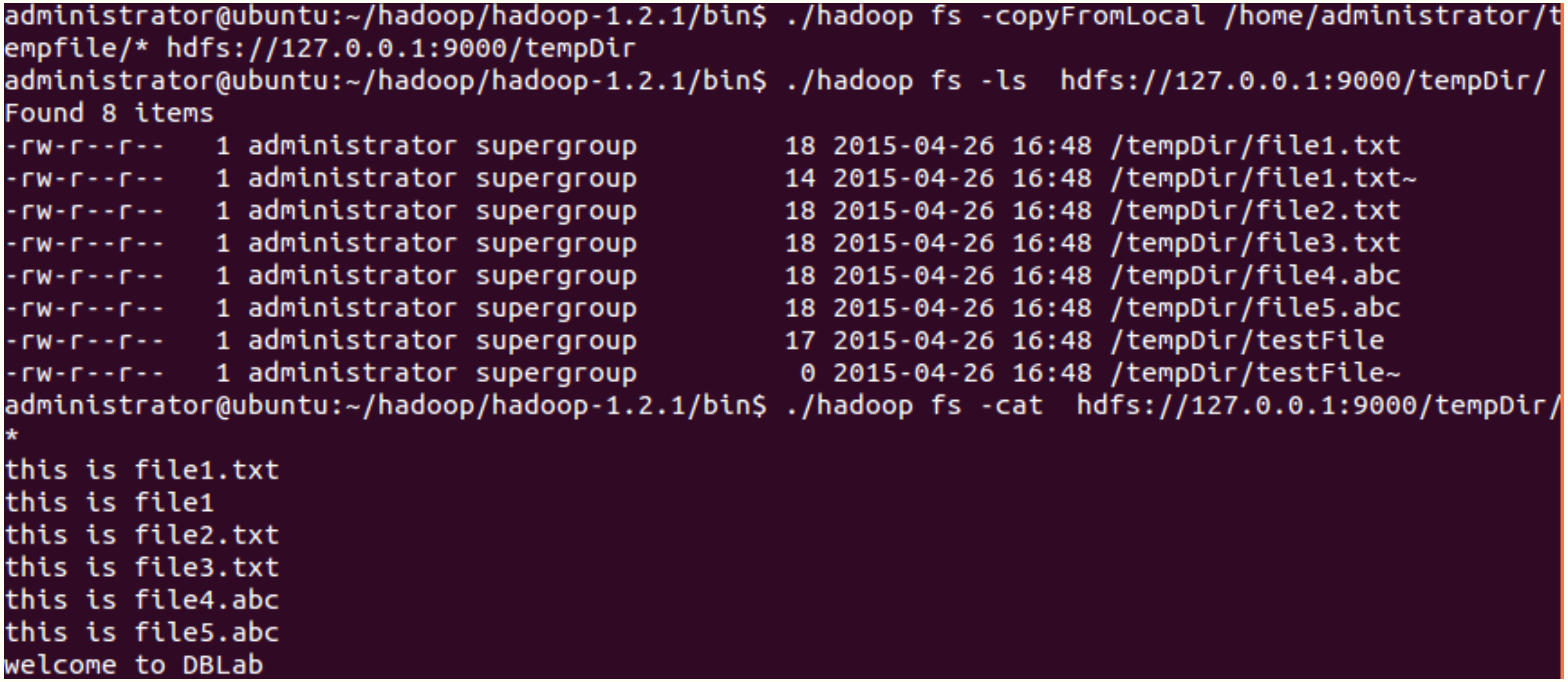
- hadoop fs -ls <path>: display detailed file info specified by path
- hadoop fs -mkdir <path>: create folder



```
administrator@ubuntu:~/hadoop/hadoop-1.2.1/bin$ ./hadoop fs -mkdir hdfs://127.0.0.1:9000/tempDir

administrator@ubuntu:~/hadoop/hadoop-1.2.1/bin$ ./hadoop fs -ls hdfs://127.0.0.1:9000/
Found 4 items
drwxr-xr-x   - administrator supergroup          0 2015-04-26 16:30 /hbase
drwxr-xr-x   - administrator supergroup          0 2015-04-26 15:44 /home
drwxr-xr-x   - administrator supergroup          0 2015-04-26 16:46 /tempDir
drwxr-xr-x   - administrator supergroup          0 2015-04-26 15:55 /user
```

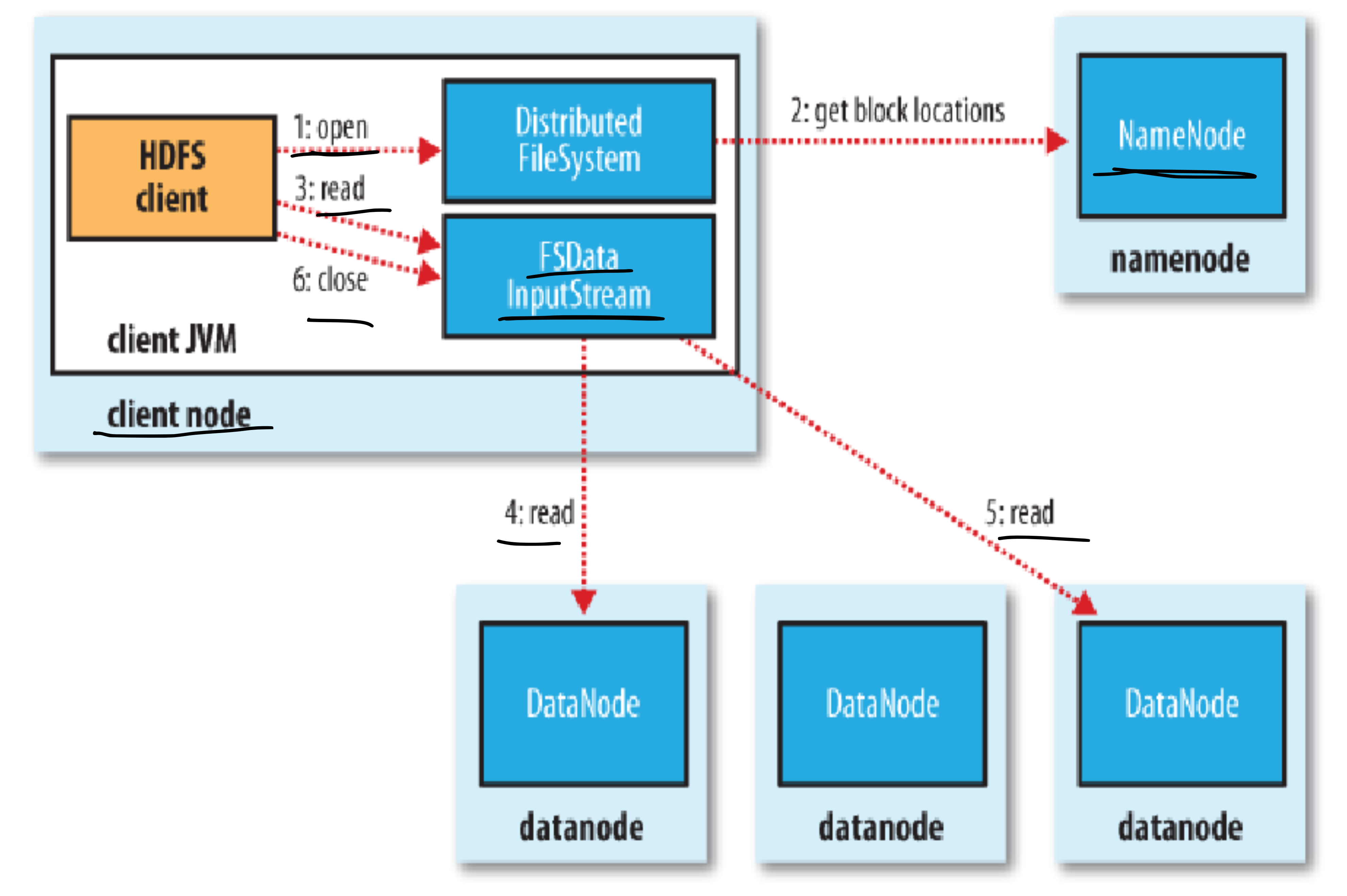
HDFS command (cont.)

- Shell command:
  - hadoop fs -cat <path>: stdout file content
  - hadoop fs –copyFromLocal <localsrc> <dst>: copy file



```
administrator@ubuntu:~/hadoop/hadoop-1.2.1/bin$ ./hadoop fs -copyFromLocal /home/administrator/t
empfile/* hdfs://127.0.0.1:9000/tempDir
administrator@ubuntu:~/hadoop/hadoop-1.2.1/bin$ ./hadoop fs -ls  hdfs://127.0.0.1:9000/tempDir/
Found 8 items
-rw-r--r--   1 administrator supergroup         18 2015-04-26 16:48 /tempDir/file1.txt
-rw-r--r--   1 administrator supergroup         14 2015-04-26 16:48 /tempDir/file1.txt~
-rw-r--r--   1 administrator supergroup         18 2015-04-26 16:48 /tempDir/file2.txt
-rw-r--r--   1 administrator supergroup         18 2015-04-26 16:48 /tempDir/file3.txt
-rw-r--r--   1 administrator supergroup         18 2015-04-26 16:48 /tempDir/file4.abc
-rw-r--r--   1 administrator supergroup         18 2015-04-26 16:48 /tempDir/file5.abc
-rw-r--r--   1 administrator supergroup         17 2015-04-26 16:48 /tempDir/testFile
-rw-r--r--   1 administrator supergroup          0 2015-04-26 16:48 /tempDir/testFile~
administrator@ubuntu:~/hadoop/hadoop-1.2.1/bin$ ./hadoop fs -cat  hdfs://127.0.0.1:9000/tempDir/
*
this is file1.txt
this is file1
this is file2.txt
this is file3.txt
this is file4.abc
this is file5.abc
welcome to DBLab
```

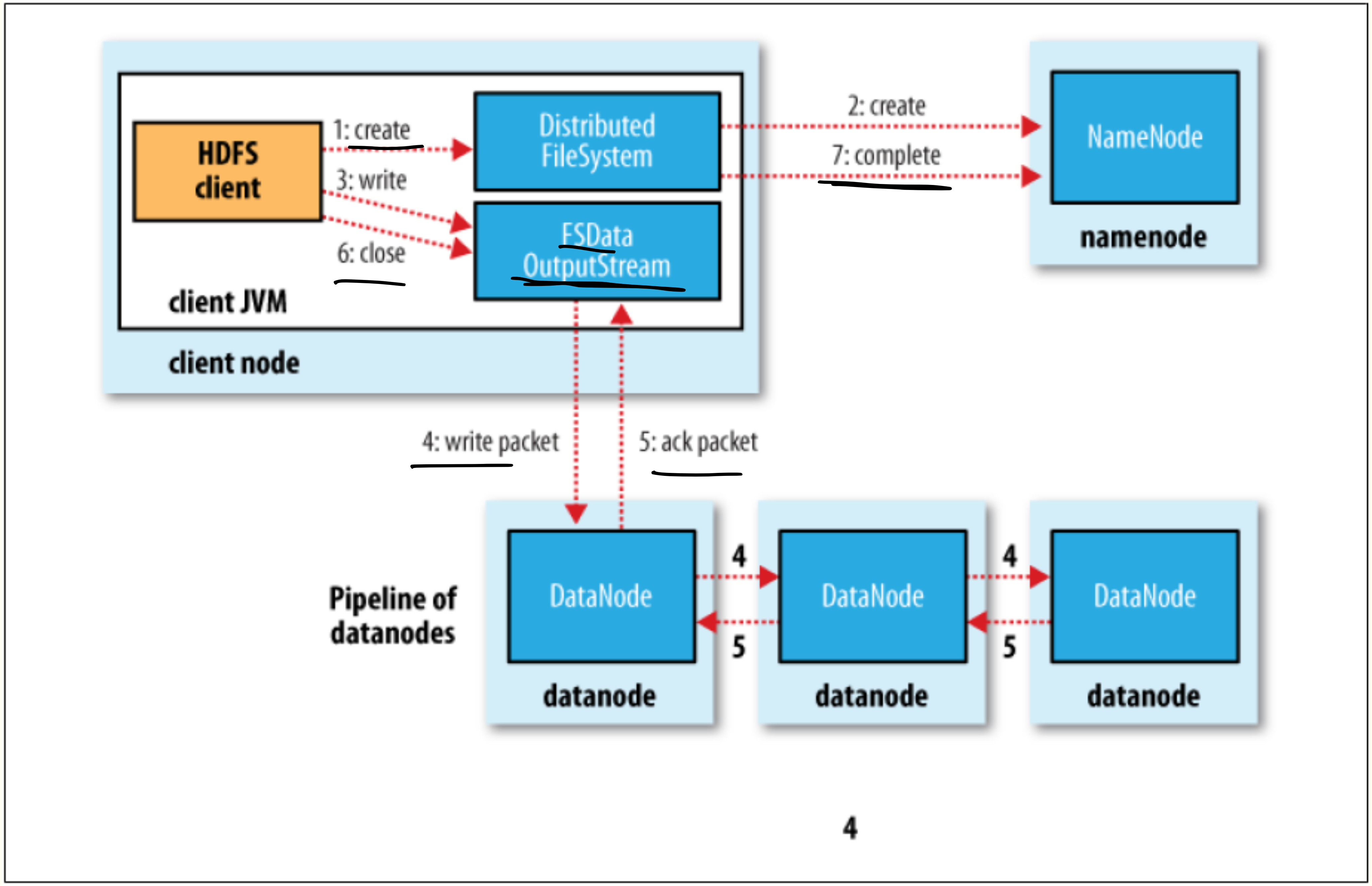
Read from HDFS



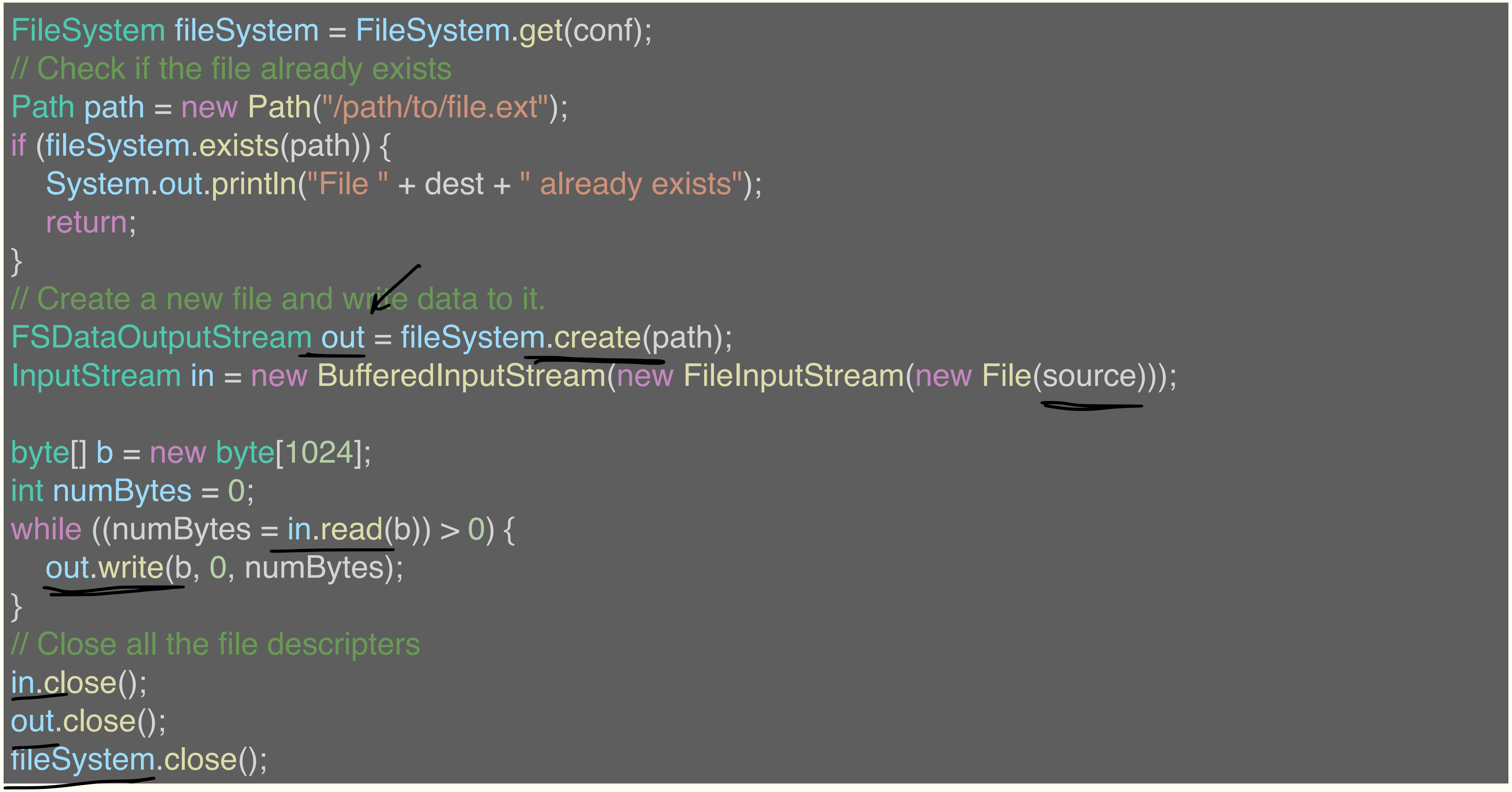
Read from HDFS

```
FileSystem fileSystem = FileSystem.get(conf);
Path path = new Path("/path/to/file.ext");
if (!fileSystem.exists(path)) {
    System.out.println("File does not exists");
    return;
}

FSDataInputStream in = fileSystem.open(path);
int numBytes = 0;
while ((numBytes = in.read(b)) > 0) {
    System.out.println((char) numBytes);
    // code to manipulate the data which is read

}
in.close();
out.close();
fileSystem.close();
```

HDFS Write


HDFS client
1: create
3: write
6: close
Distributed FileSystem
FSData OutputStream
client JVM
client node
2: create
7: complete
NameNode
namenode
4: write packet
5: ack packet
Pipeline of datanodes
DataNode
datanode
4
5
DataNode
datanode
4
5
DataNode
datanode
4

HDFS Write



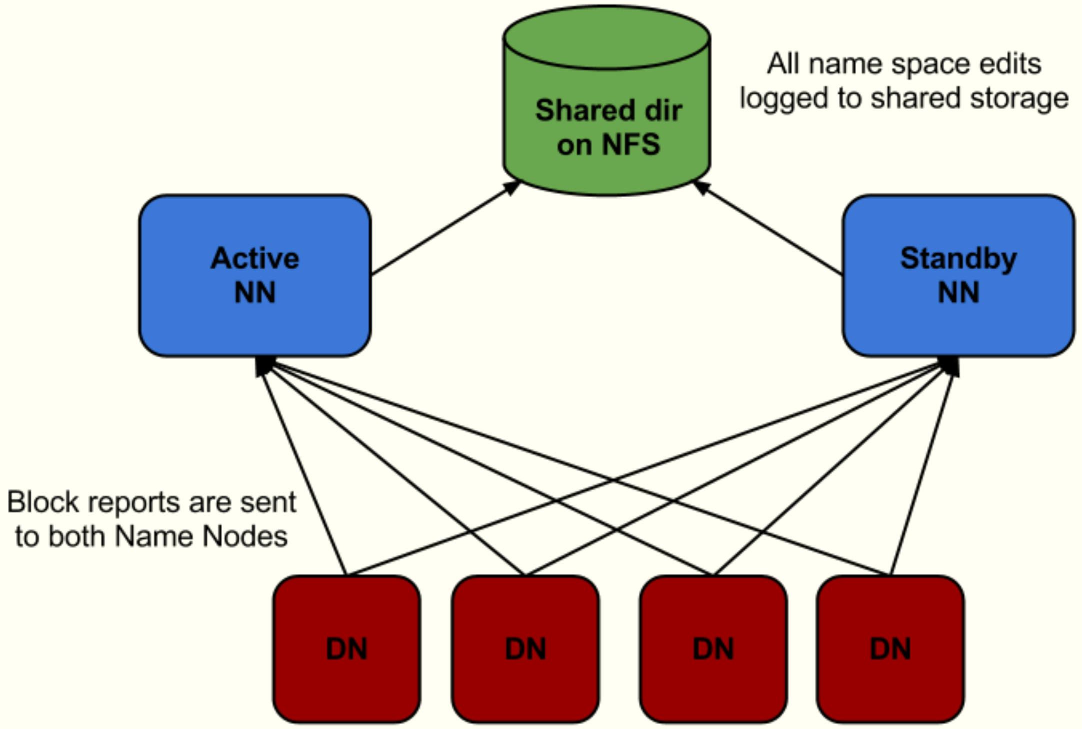
```
FileSystem fileSystem = FileSystem.get(conf);
// Check if the file already exists
Path path = new Path("/path/to/file.ext");
if (fileSystem.exists(path)) {
    System.out.println("File " + dest + " already exists");
    return;
}
// Create a new file and write data to it.
FSDataOutputStream out = fileSystem.create(path);
InputStream in = new BufferedInputStream(new FileInputStream(new File(source)));

byte[] b = new byte[1024];
int numBytes = 0;
while ((numBytes = in.read(b)) > 0) {
    out.write(b, 0, numBytes);
}
// Close all the file descripters
in.close();
out.close();
fileSystem.close();
```

System Issues

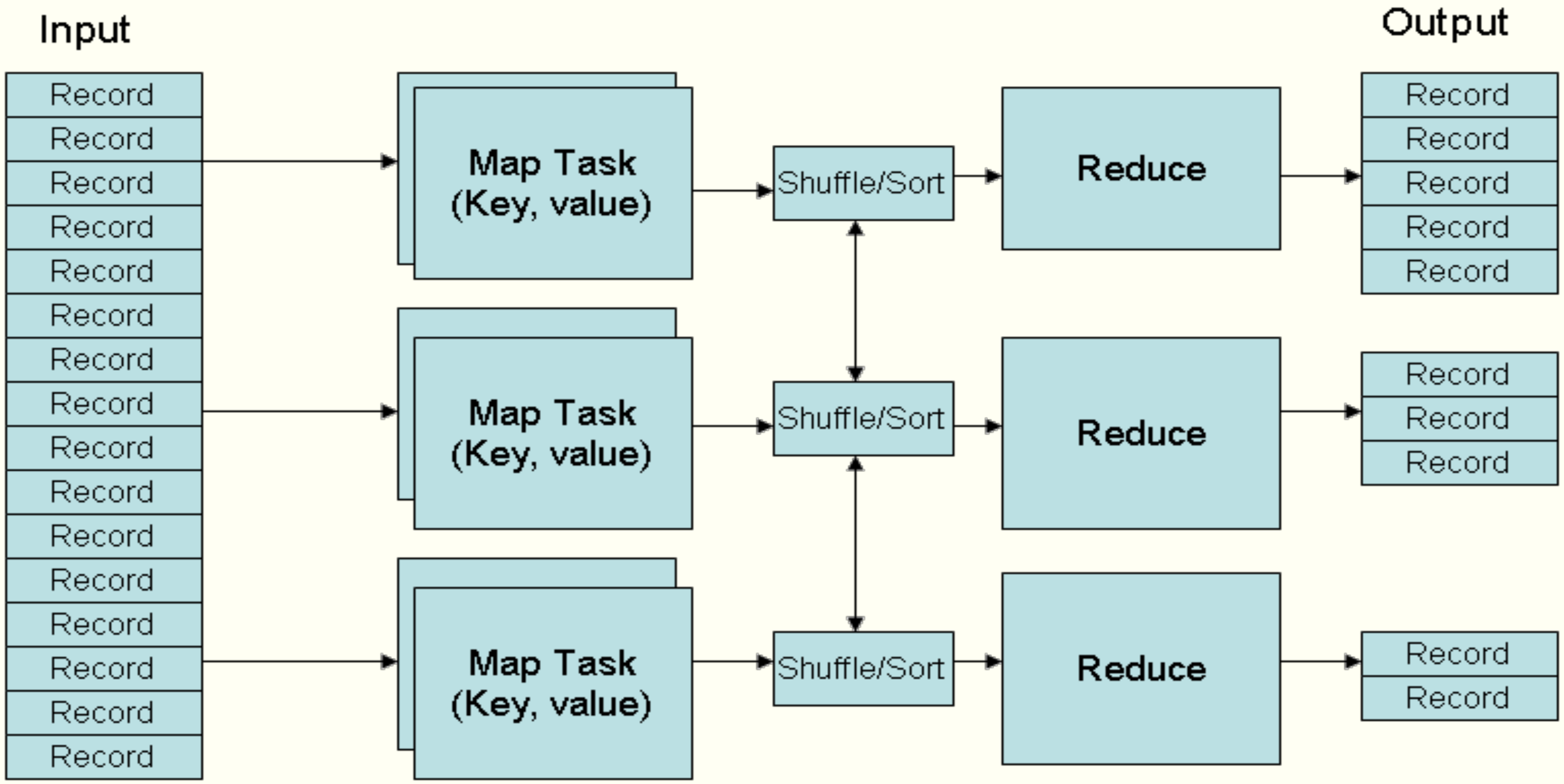
- Block Placement: How to place data blocks?
    - One replica on local node, second/third on same remote rack, additional replica randomly placed
    - Clients read from nearest replicas
- Replication Engine
    - NameNode detects DataNode failures
        - Chooses new DataNodes for new replicas
        - Balances disk usage
        - Balances communication traffic to DataNodes
- Rebalancer: % of disk full on DataNodes should be similar
    - Run when new datanodes are added

Data Recovery And Error Tolerance

- HDFS treats fault as norm not exception
  - Namenode failure
  - Datanode failure
  - Data error
- Heartbeats
  - DataNodes send hearbeat to the NameNode
    - Once every 3 seconds
  - NameNode uses heartbeats to detect DataNode failure
- Namenode failure:
  - FsImage, Editlog -> SecondaryNameNode
  - Transaction Log + standby NN
- Data error
  - md5/sha1 validation
  - client check/report -> namenode replication

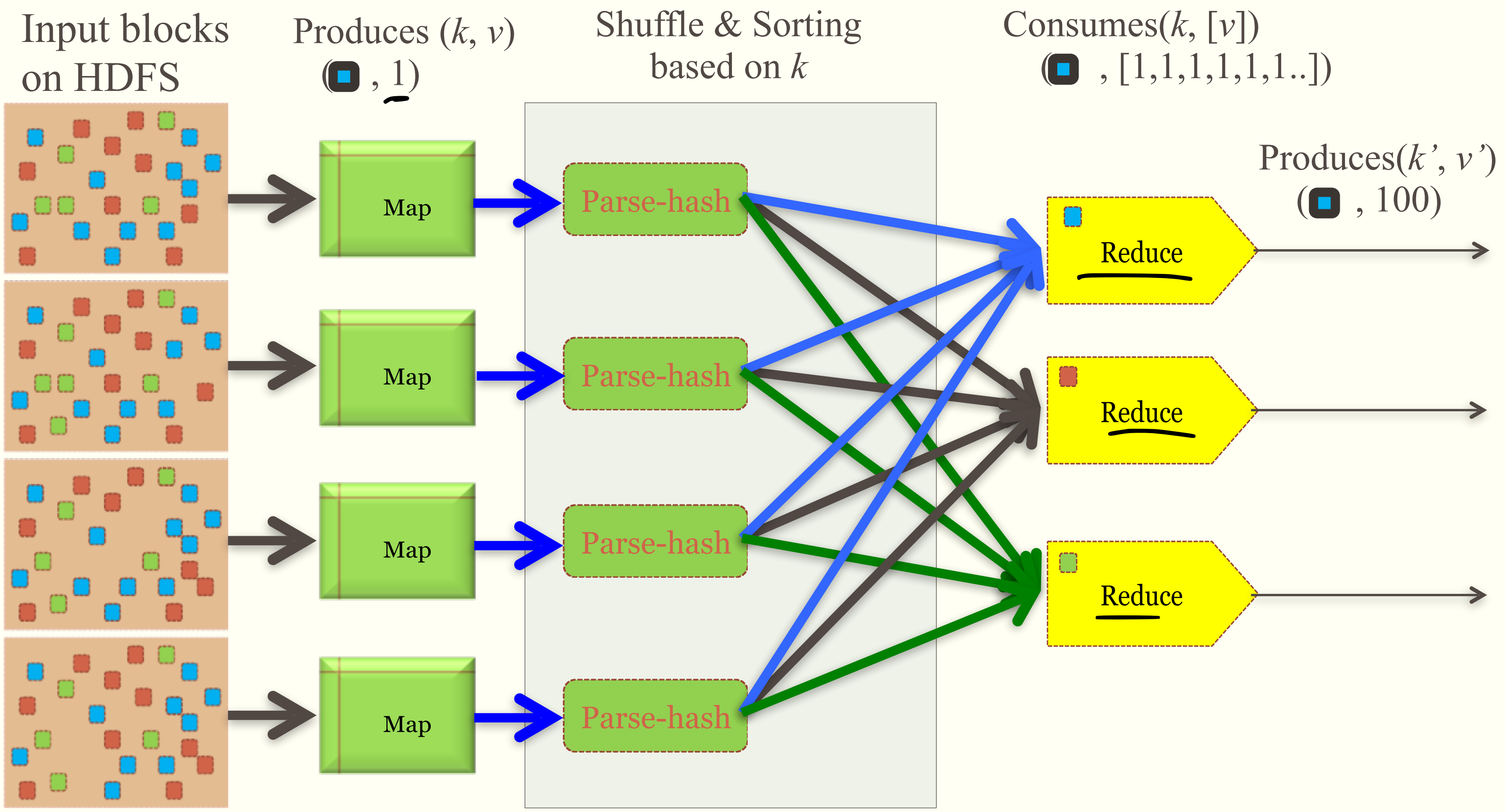

Shared dir on NFS
All name space edits logged to shared storage
Active NN
Standby NN
Block reports are sent to both Name Nodes
DN
DN
DN
DN

MapReduce Layer



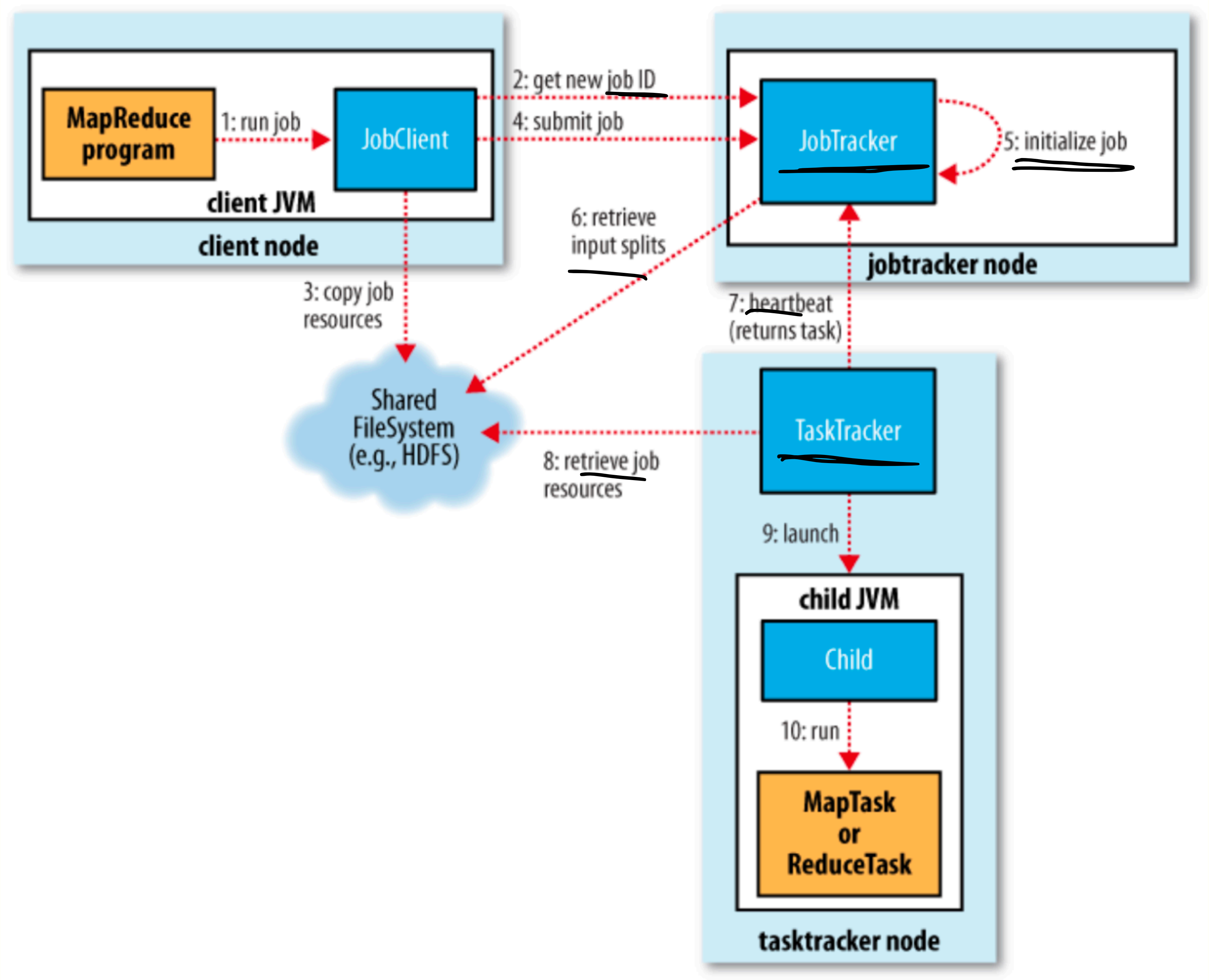
Deciding on what will be the key and what will be the value ➔ developer's responsibility

Example: Color Count



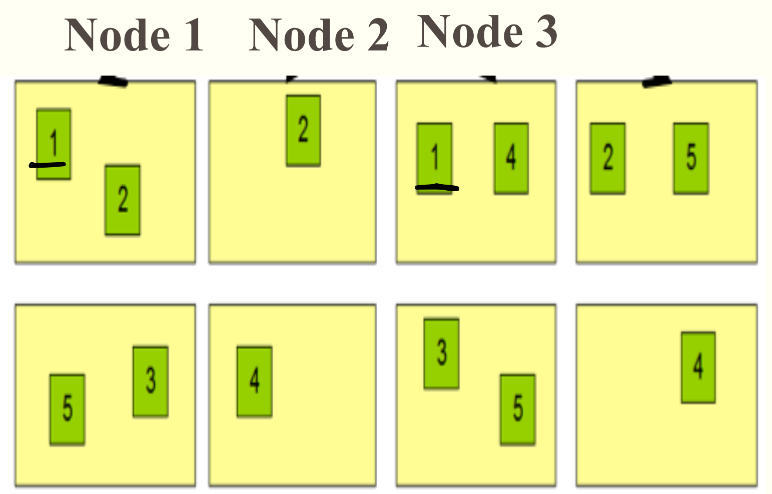
Users only provide the "Map" and "Reduce" functions

MapReduce Framework Details



Properties of MapReduce Engine

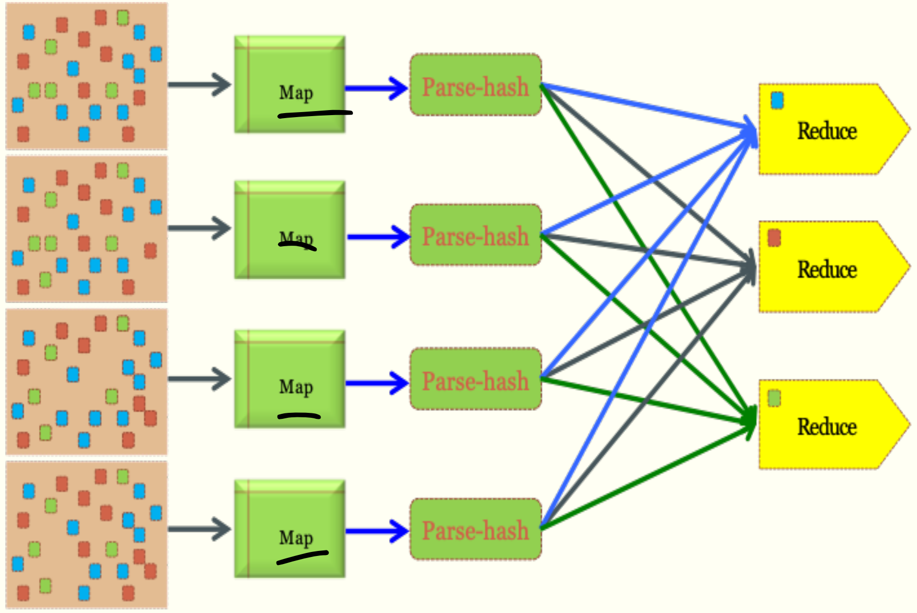
- Job Tracker is the master node (runs with the namenode)
  - Receives the user's job
  - Decides on how many tasks will run (number of mappers)
  - Decides on where to run each mapper (concept of locality)



- 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 (Cont'd)

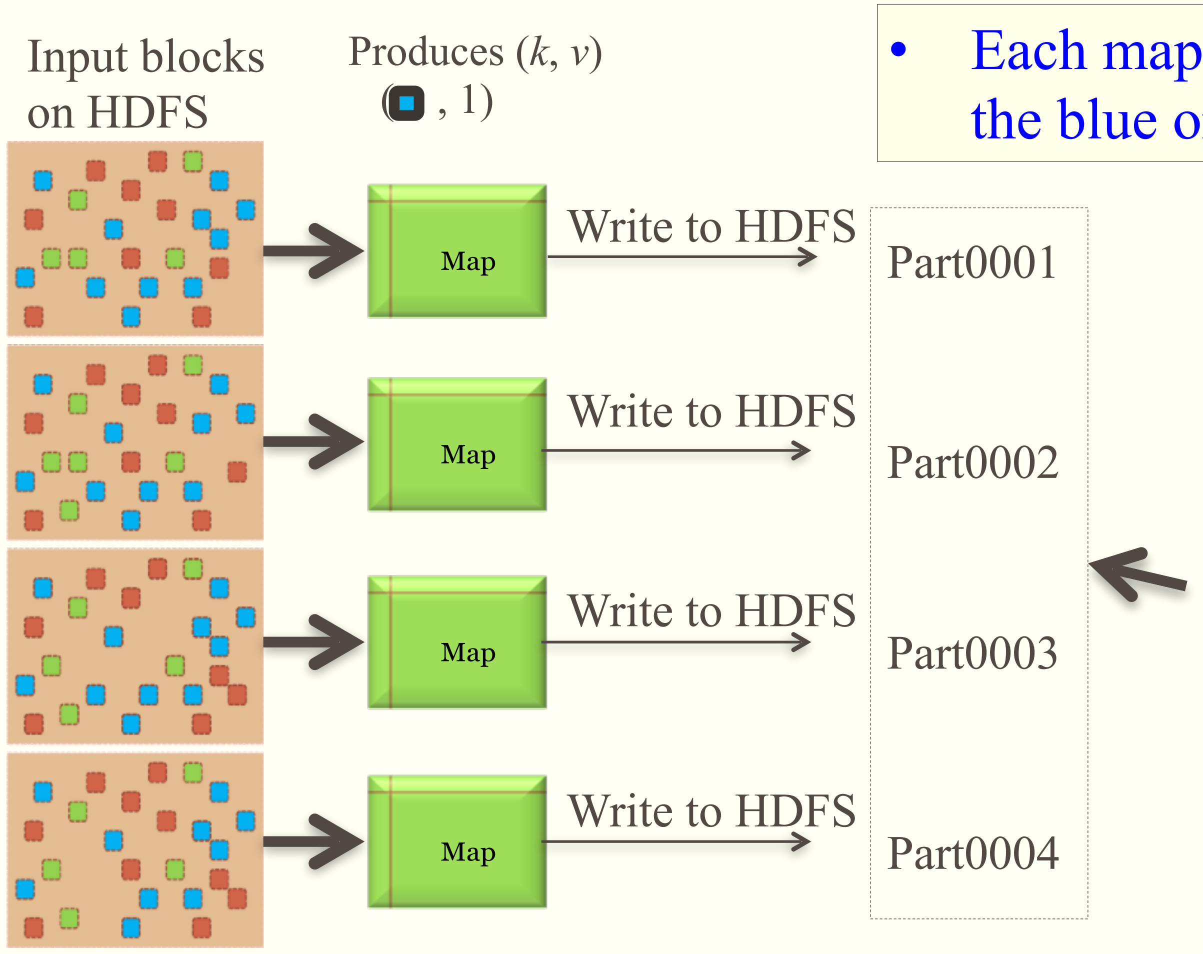
- Task Tracker is the slave node (runs on each datanode)
  - Receives the task from Job Tracker
  - Runs the task until completion (either map or reduce task)
  - Always in communication with the Job Tracker reporting progress



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

Example: Color Filter

- Job: Select only the blue and the green colors



- Each map task will select only the blue or green colors

the output file, has 4 parts on probably 4 different machines

Putting it all together

- Create a launching program for your application

- The launching program configures:
    - The Mapper and Reducer to use
    - The output key and value types (input types are inferred from the InputFormat)
    - The locations for your input and output

- The launching program then submits the job and typically waits for it to complete