ITU INFORMATION TECHNOLOGY UNIVERSITY

DATA SCIENCE LAB

Lecture 9

CS 537- Big Data Analytics

Dr. Faisal Kamiran

Safemode Startup

- On startup Namenode enters Safemode.
- Replication of data blocks do not occur in Safemode.
- Each DataNode checks in with Heartbeat and BlockReport.
- Namenode verifies that each block has acceptable number of replicas
- After a configurable percentage of safely replicated blocks check in with the Namenode, Namenode exits Safemode.
- It then makes the list of blocks that need to be replicated.
- Namenode then proceeds to replicate these blocks to other Datanodes.

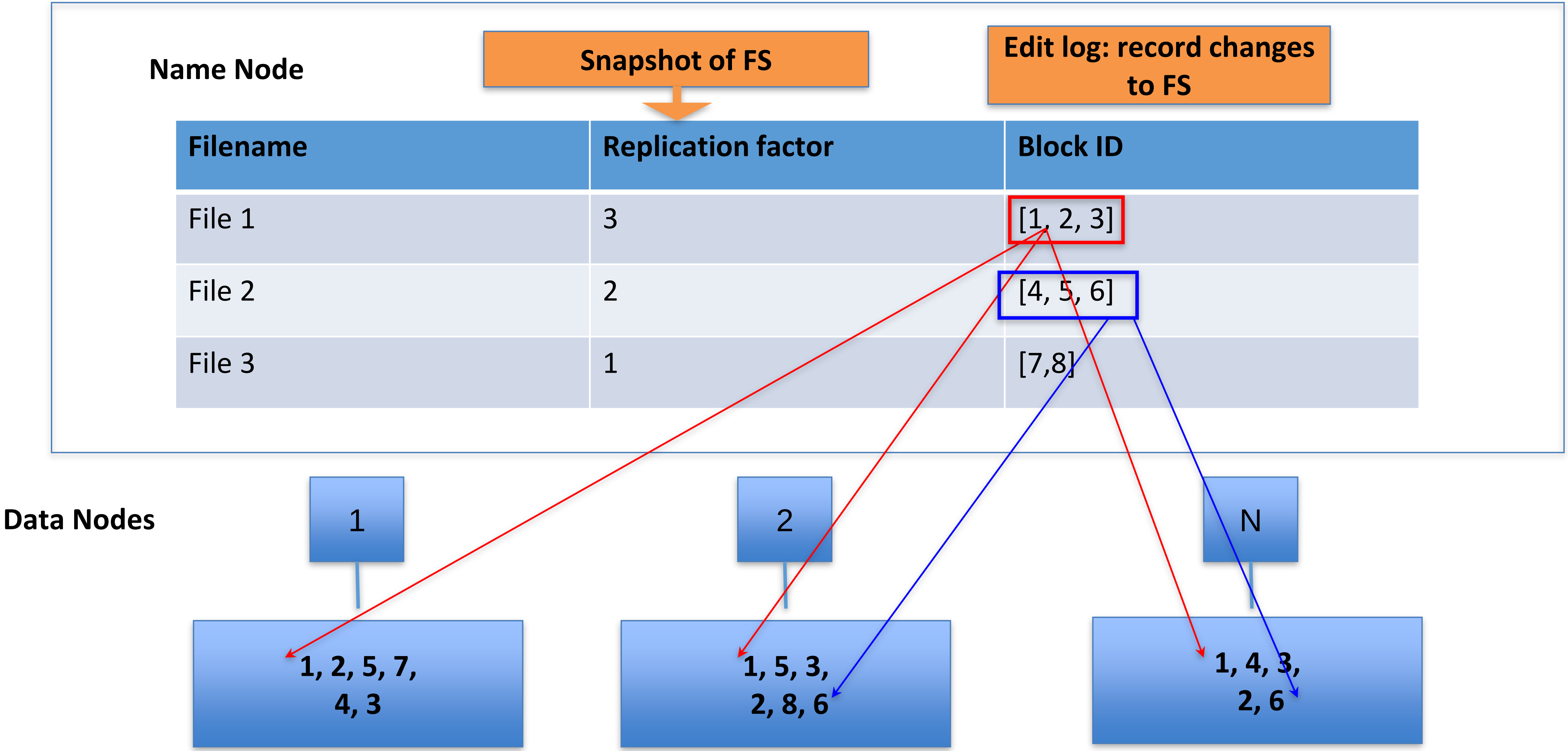
Filesystem Metadata

- The HDFS namespace is stored by Namenode.
- Namenode uses a transaction log called the EditLog to record every change that occurs to the filesystem meta data.
  - For example, creating a new file.
  - Change replication factor of a file
  - EditLog is stored in the Namenode’s local filesystem
- Entire filesystem namespace including mapping of blocks to files and file system properties is stored in a file FsImage. Stored in Namenode’s local filesystem.

Namenode

- Keeps image of entire file system namespace and file Blockmap in memory.
- When the Namenode starts up it gets the **FsImage** and **Editlog** from its local file system, update FsImage with EditLog information and then stores a copy of the FsImage on the filesytstem as a checkpoint.
- Periodic checkpointing is done. So that the system can recover back to the last checkpointed state in case of a crash.

HDFS Inside: Name Node



Name Node

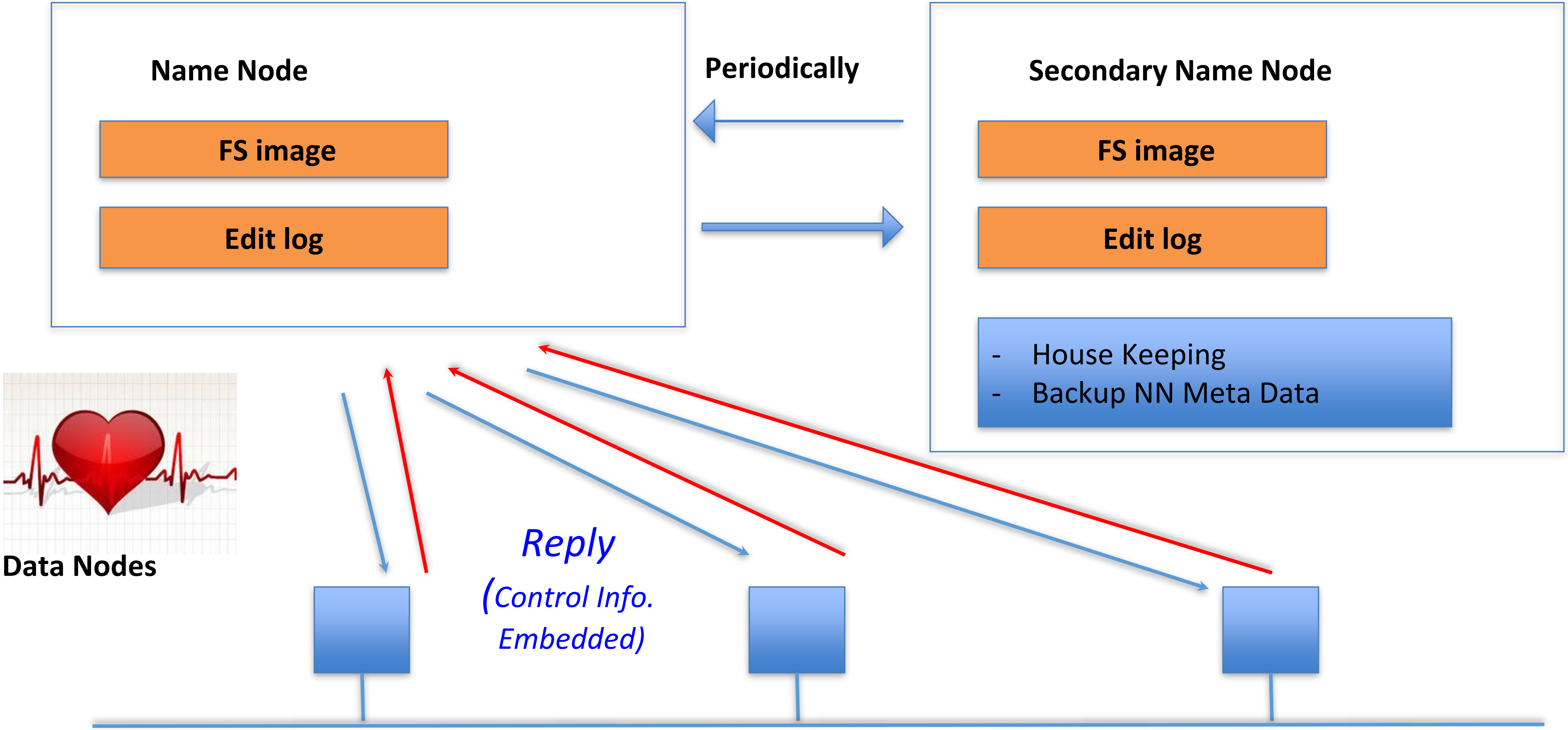
Snapshot of FS

Edit log: record changes to FS

Filename	Replication factor	Block ID
File 1	3	[1, 2, 3]
File 2	2	[4, 5, 6]
File 3	1	[7,8]

Data Nodes

HDFS Inside: Name Node



Name Node

FS image

Edit log

Periodically

Secondary Name Node

FS image

Edit log

- House Keeping
- Backup NN Meta Data

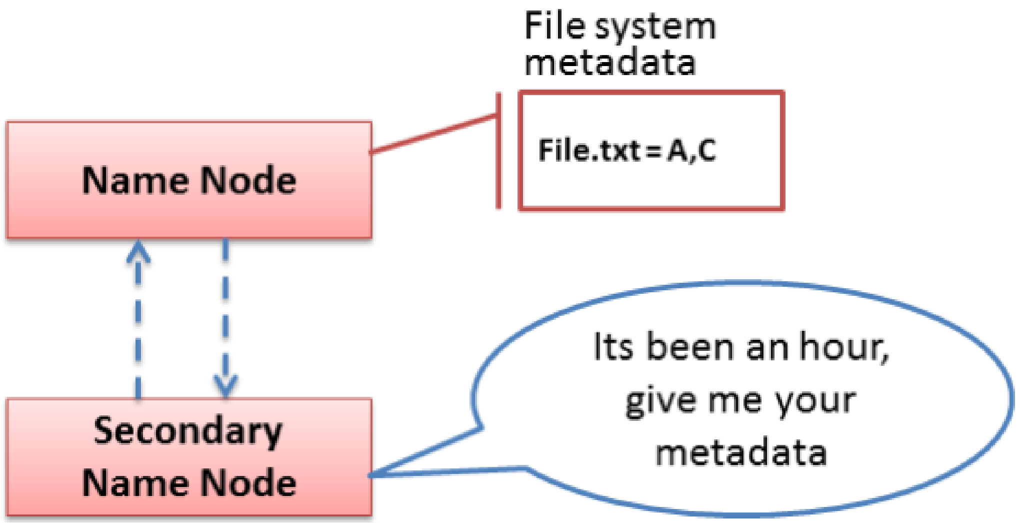
Data Nodes

Reply

(Control Info. Embedded)

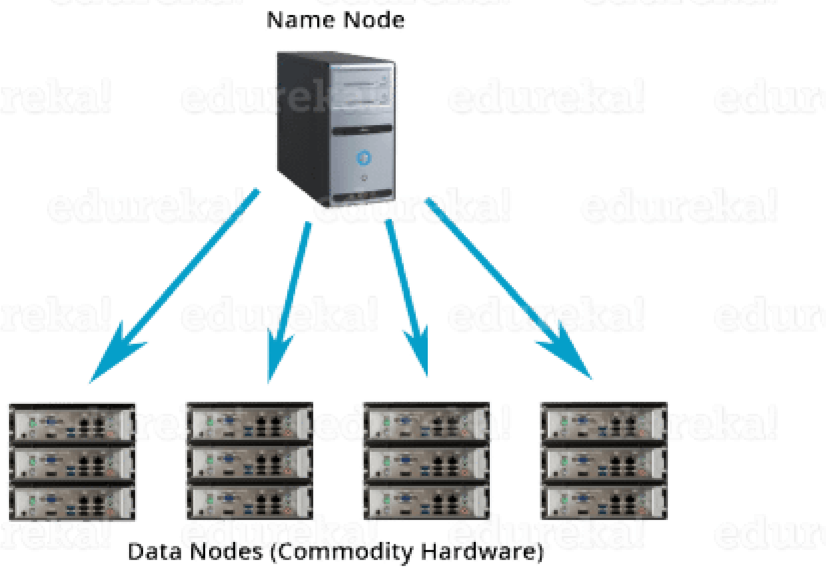
Secondary Name Node

- NOT a backup Name Node (We can call it a helper node)
- Merging FsImage and EditLog is memory intensive
- Secondary Name Node periodically downloads current Name Node FsImage and EditLog, merges them and uploads the new image back



Datanode

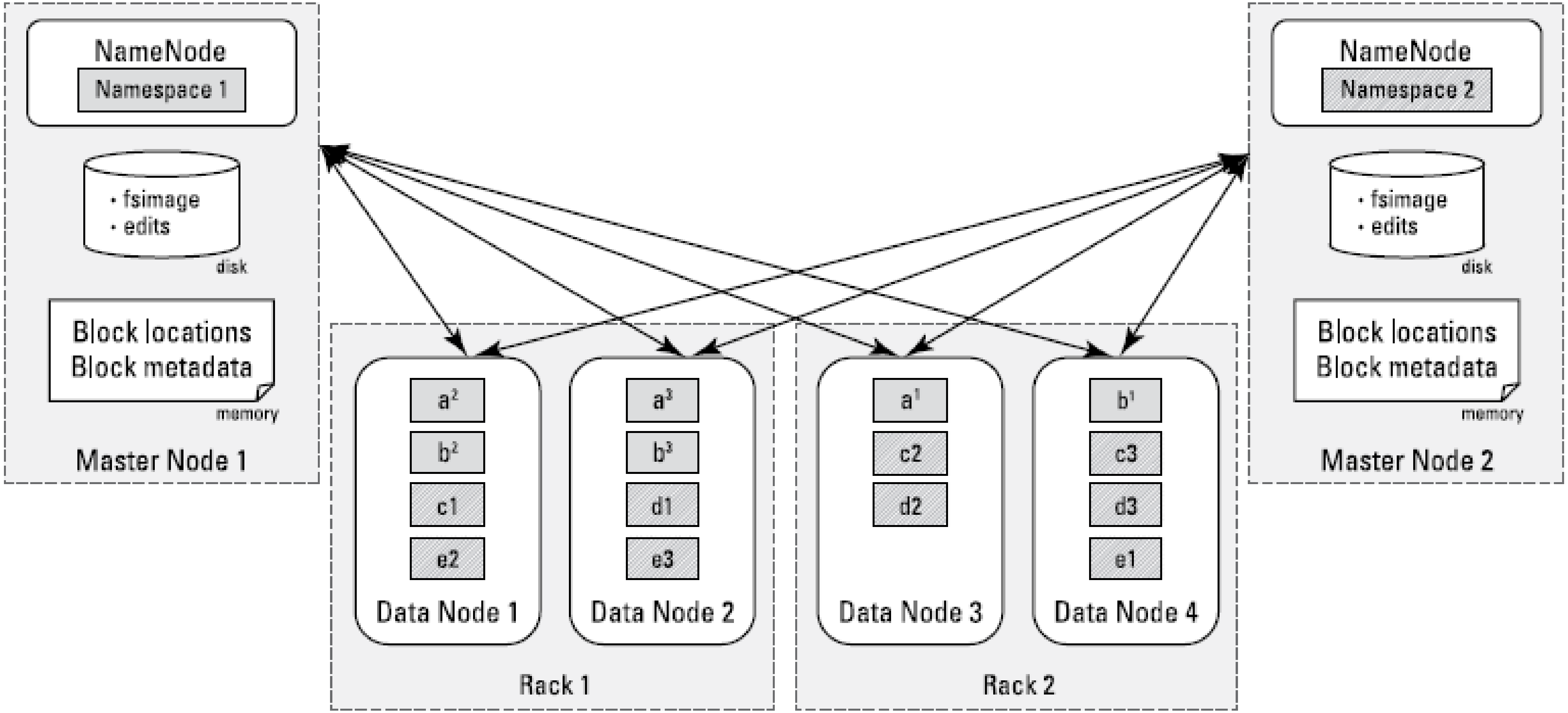
- A Datanode stores data in files in its local file system.
- Datanode has no knowledge about HDFS filesystem
- When the filesystem starts up it generates a list of all HDFS blocks and send this report to Namenode: Blockreport.



HDFS Federation

- Before Hadoop 2, a single Name Node had to store metadata for every block of data
- Difficult to manage increasing number of blocks
- HDFS Federation –
    - Multiple Name Nodes
    - Each Name Node responsible for a particular Namespace (set of blocks)
    - Only individual blocks are partitioned among Name Nodes. Data Nodes can be shared.

HDFS Federation

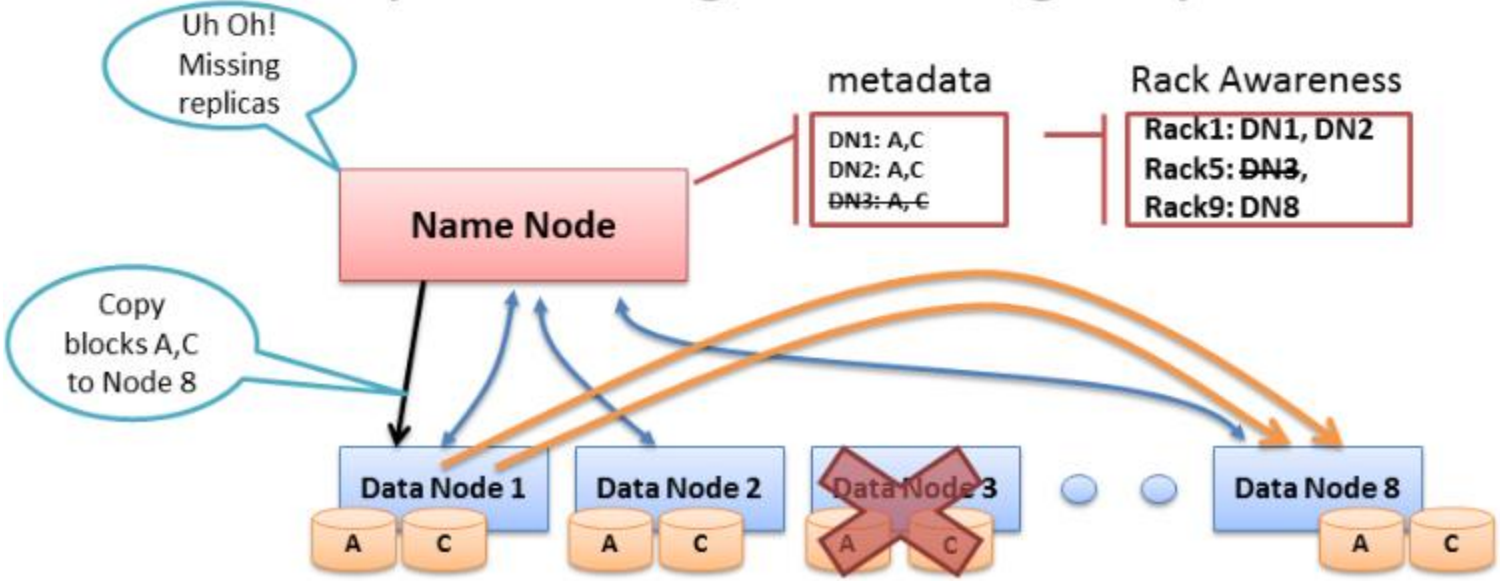


DataNode failure and heartbeat

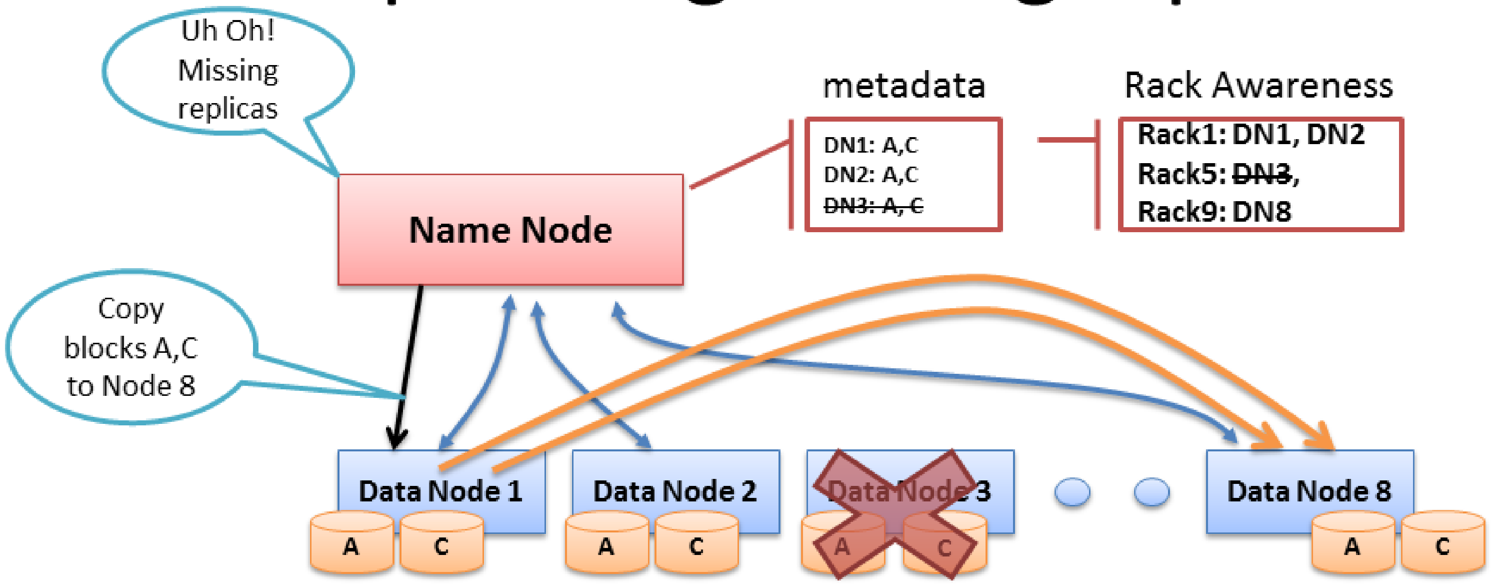
- A network partition can cause a subset of Datanodes to lose connectivity with the Namenode.
- Namenode detects this condition by the absence of a Heartbeat message.
- Namenode marks **Datanodes without Hearbeat** and does not send any IO requests to them.
- Any data registered to the failed Datanode is not available to the HDFS.
- Also the death of a Datanode may cause replication factor of some of the blocks to fall below their specified value.

Re-replication

- The necessity for re-replication may arise due to:
  - A Datanode may become unavailable,
  - A replica may become corrupted,
  - A hard disk on a Datanode may fail, or
  - The replication factor on the block may be increased.



Re-replicating missing replicas



Uh Oh! Missing replicas

Name Node

Copy blocks A,C to Node 8

metadata

DN1: A,C
DN2: A,C
~~DN3: A, C~~

Rack Awareness

Rack1: DN1, DN2
Rack5: ~~DN3~~,
Rack9: DN8

Data Node 1

Data Node 2

Data Node 3

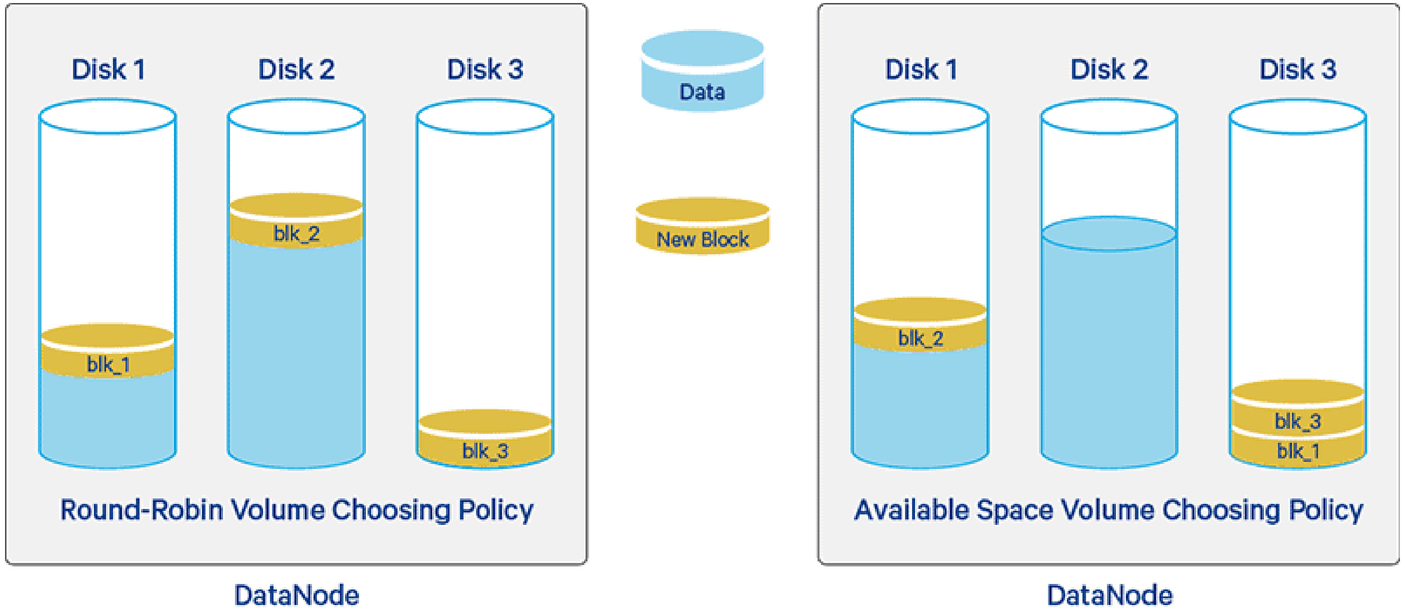
Data Node 8

- 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

BRAD HEDLUND .com

Cluster Rebalancing

- HDFS architecture is compatible with data rebalancing schemes.
- A scheme might move data from one Datanode to another if the free space on a Datanode falls below a certain threshold.
- In the event of a sudden high demand for a particular file, a scheme might dynamically create additional replicas and rebalance other data in the cluster.



Data Integrity

- Consider a situation: a block of data fetched from Datanode arrives corrupted.
- This corruption may occur because of faults in a storage device, network faults, or buggy software.
- A HDFS client creates the **checksum** of every block of its file and stores it in hidden files in the HDFS namespace.
- When a clients retrieves the contents of file, it verifies that the corresponding checksums match.
- If does not match, the client can retrieve the block from a replica.

Metadata Disk Failure

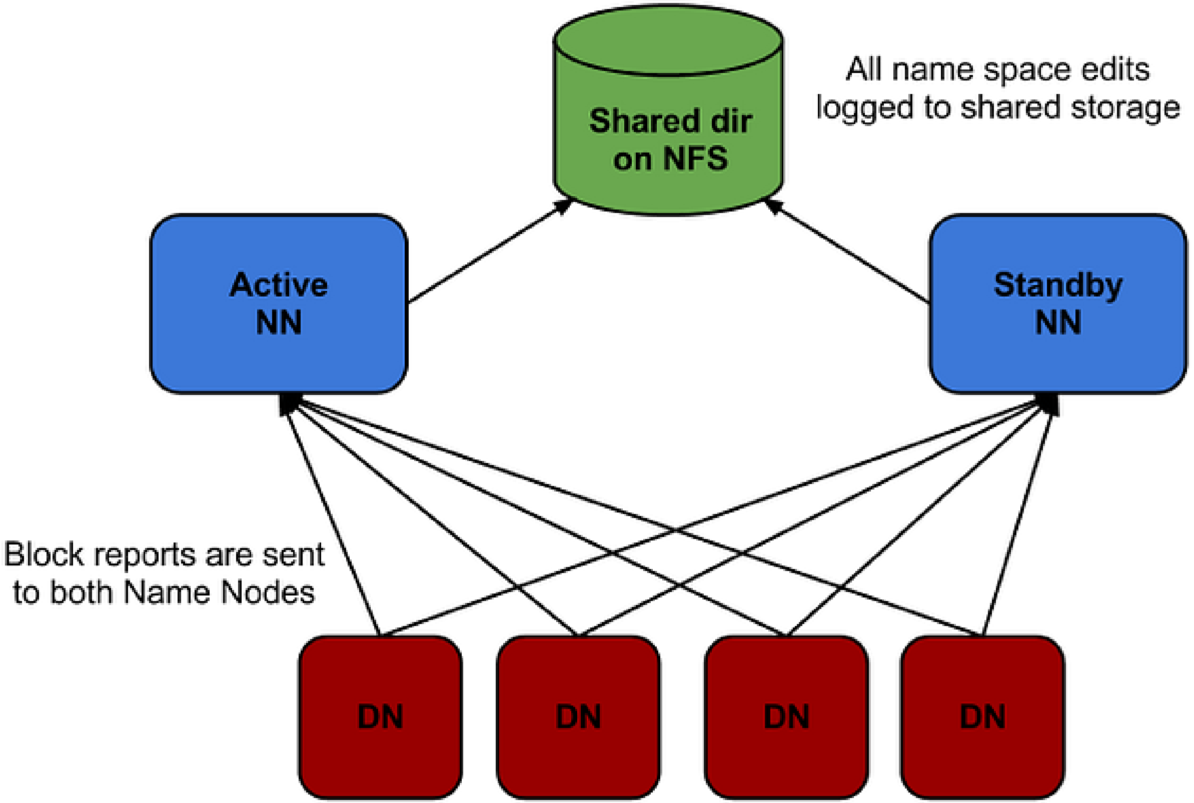
- **FsImage and EditLog** are central data structures of HDFS.
- A corruption of these files can cause a HDFS instance to be non-functional.
- For this reason, a Namenode can be configured to maintain multiple copies of the FsImage and EditLog.
- Multiple copies of the FsImage and EditLog files are updated synchronously.
- Meta-data is not data-intensive.

HDFS High Availability

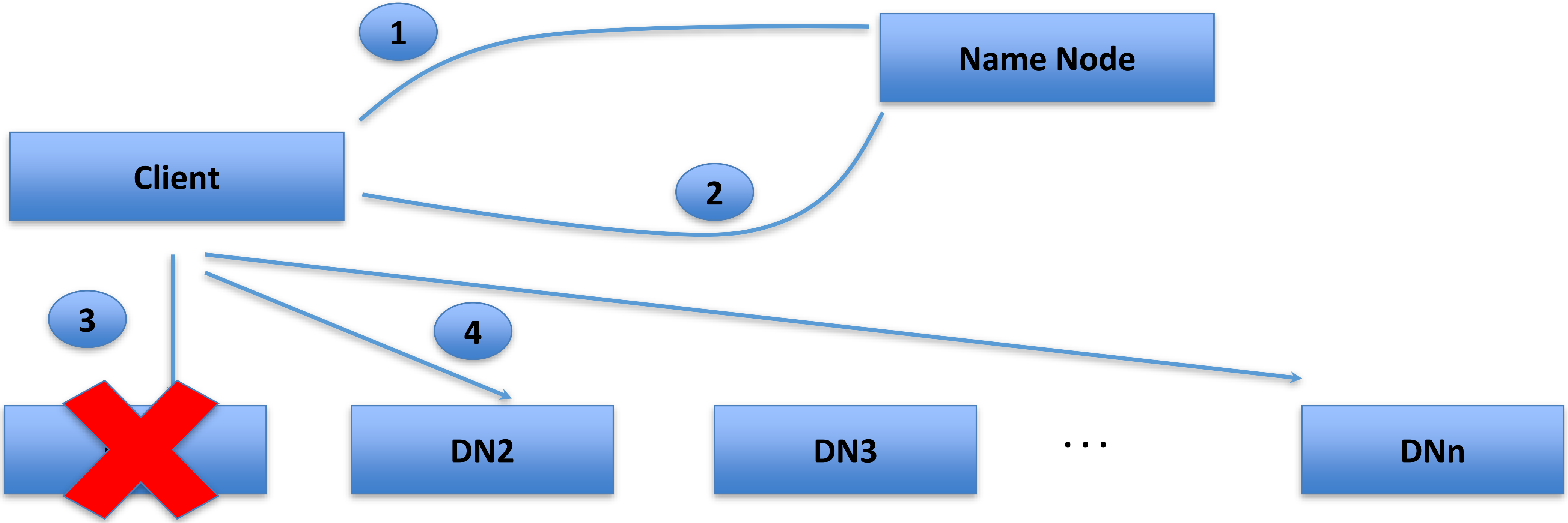
- Originally, Name Node was single point of failure.
- HDFS High Availability introduced in Hadoop 2.
- A dedicated Standby Name Node is created which has access to the up-to-date version of the current state (FsImage and EditLogs) of the file system.
- In case of Active Name Node failure, the standby Name Node takes over.
- Apache Zookeeper daemon detects Active Name Node failure and coordinates the shift.

HDFS High Availability

- Hadoop 2 only allowed one standby Name Node.
- In Hadoop 3, multiple standby Name Nodes can be created.

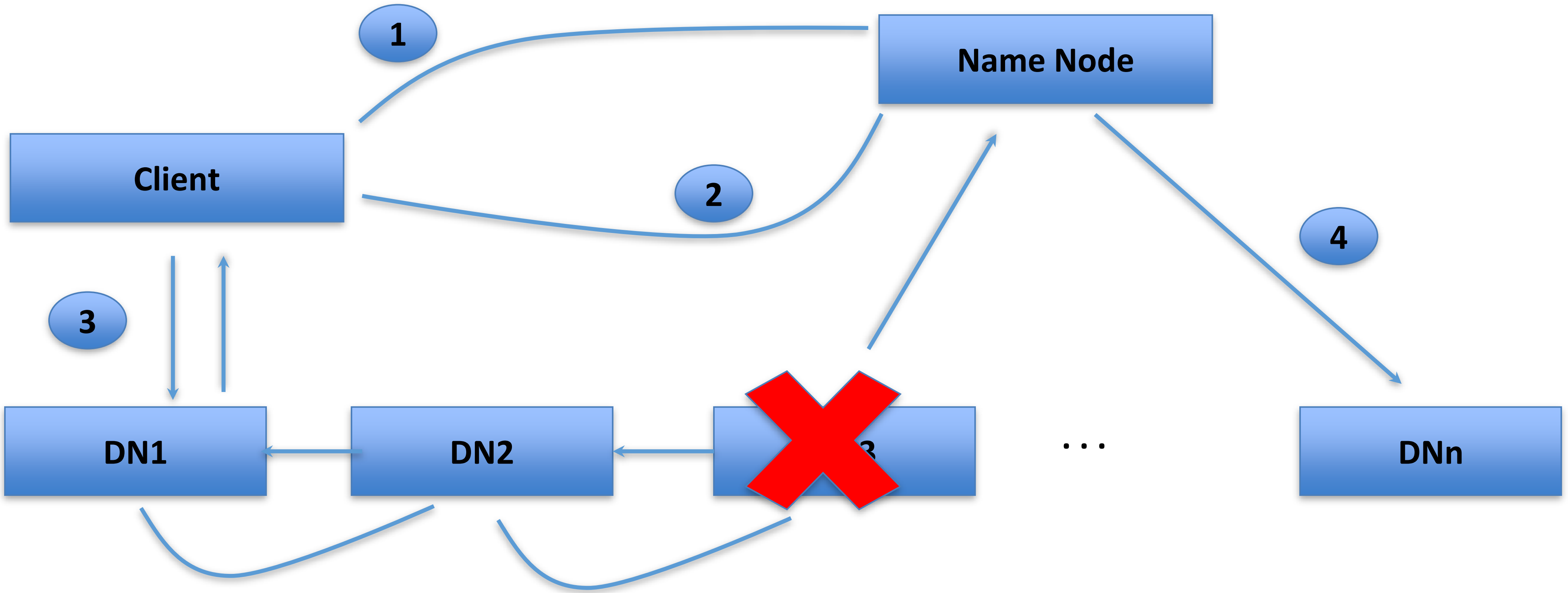

Shared dir on NFS
All name space edits logged to shared storage
Active NN
Standby NN
Block reports are sent to both Name Nodes
DN
DN
DN
DN

HDFS Inside: Read Workflow



1. Client connects to NN to read data
2. NN tells client where to find the data blocks
3. Client reads blocks directly from data nodes (without going through NN)
4. In case of node failures, client connects to another node that serves the missing block

HDFS Inside: Write Workflow



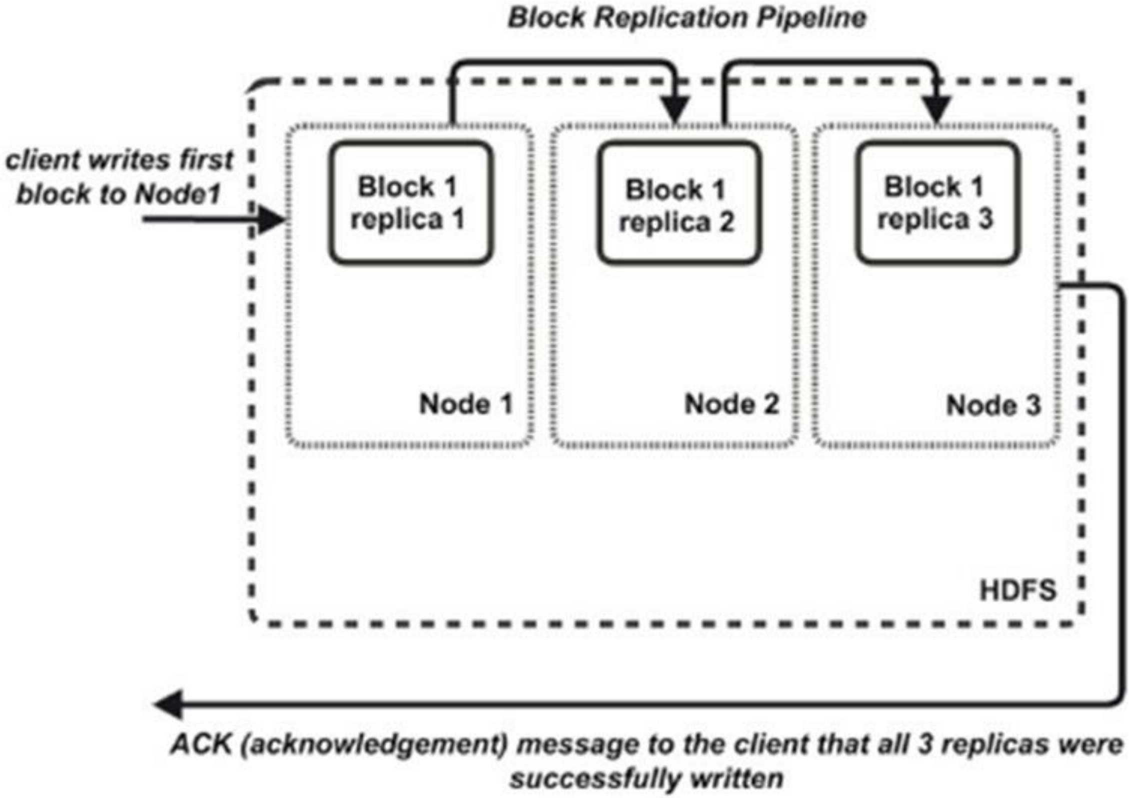
1. Client connects to NN to write data
2. NN tells client write these data nodes
3. Client writes blocks directly to data nodes with desired replication factor
4. In case of node failures, NN will figure it out and replicate the missing blocks

Staging

- A client request to create a file does not reach Namenode immediately.
- HDFS client caches the data into a temporary file. When the data reached a HDFS block size the client contacts the Namenode.
- Namenode inserts the filename into its hierarchy and allocates a data block for it.
- The Namenode responds to the client with the identity of the Datanode and the destination of the replicas (Datanodes) for the block.
- Then the client flushes it from its local memory.
- The client sends a message that the file is closed.
- Namenode proceeds to commit the file for creation operation into the persistent store.
- If the Namenode dies before file is closed, the file is lost.
- This client side caching is required to avoid network congestion

Replication Pipelining

- When the client receives response from Namenode, it flushes its block in small pieces (4K) to the first replica, that in turn copies it to the next replica and so on.
- Thus data is pipelined from Datanode to the next.

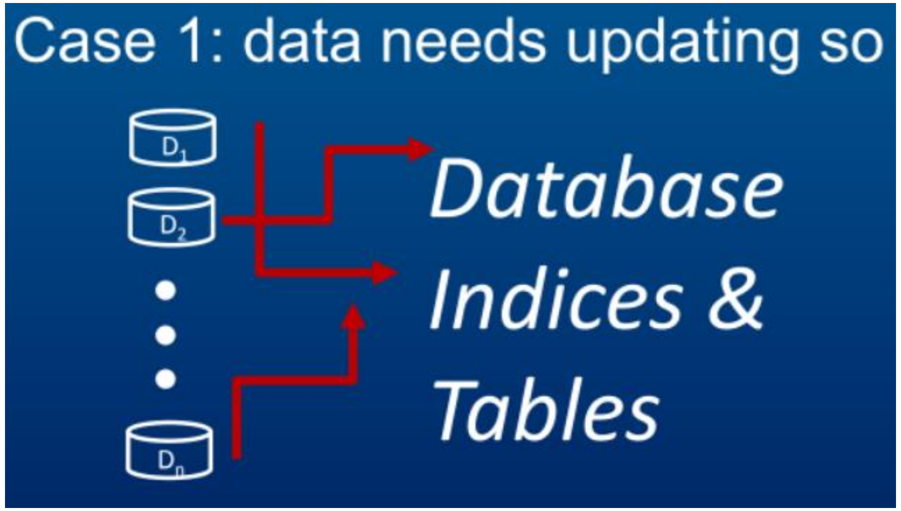

Block Replication Pipeline
client writes first block to Node1
Block 1 replica 1
Block 1 replica 2
Block 1 replica 3
Node 1
Node 2
Node 3
HDFS
ACK (acknowledgement) message to the client that all 3 replicas were successfully written

Map Reduce

Scheduling and Data Flow

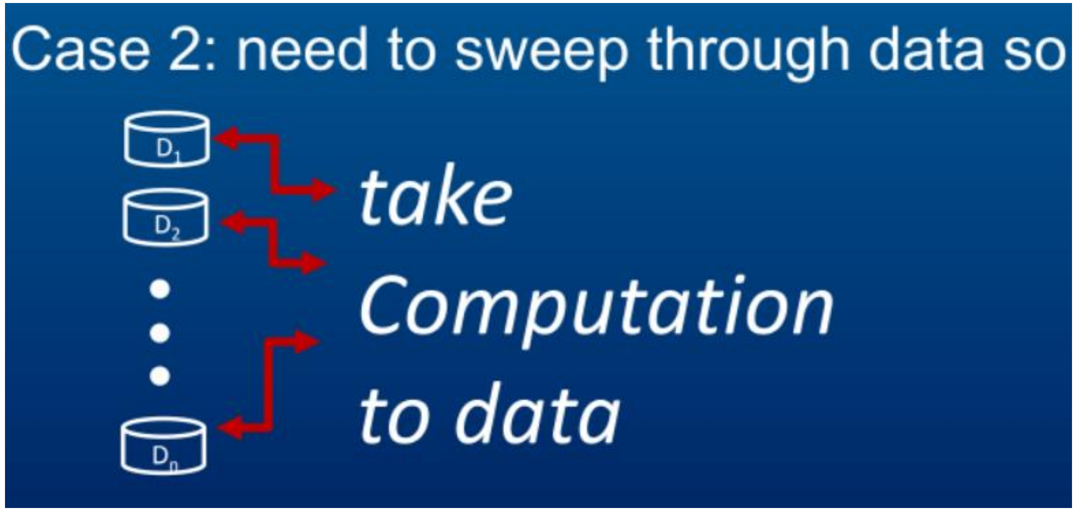
The Problem and The Solution

- **Problem:**
  - Big Data -> Large amount of data stored in large amount of devices
- **Solution:**
  - Bring computations to data
- **Possibilities:**



Case 1: data needs updating so

Database Indices & Tables

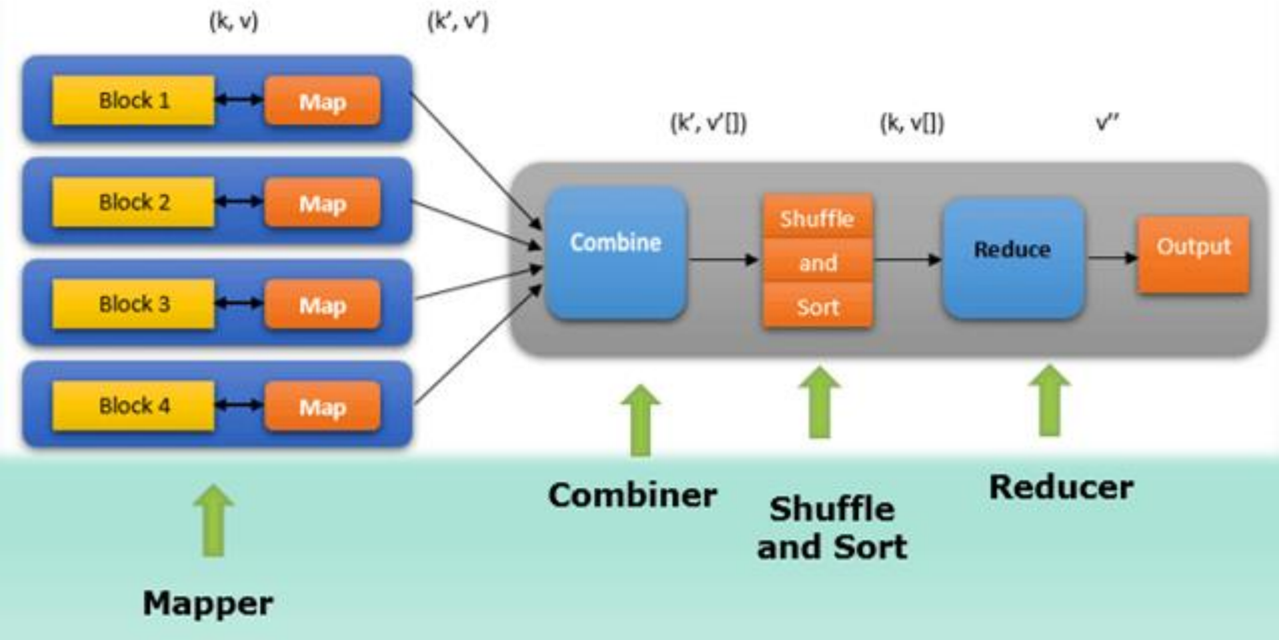


Case 2: need to sweep through data so

take Computation to data

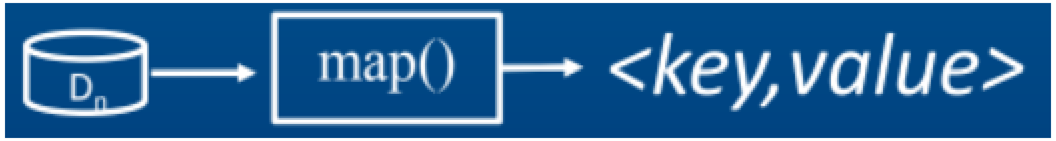
MapReduce Framework

- User defines
  - <key, value>
  - mapper & reducer functions
- Logistics:
  - Hadoop handles the distribution and execution

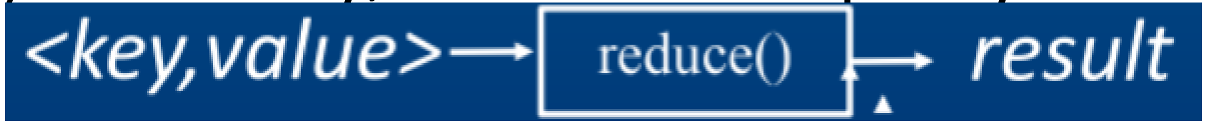


MapReduce Flow

- User defines a map function
  - map() reads data and outputs <key,value>



- User defines a reduce function
  - reduce() reads <key,value> and outputs your result

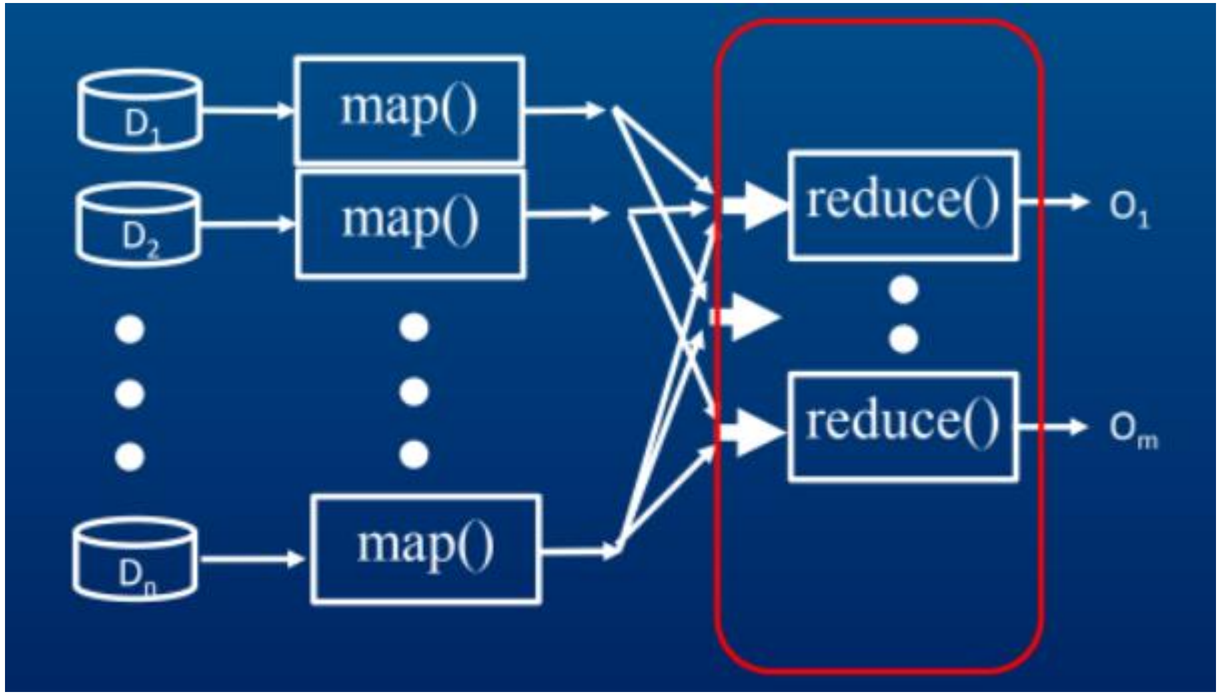


Hadoop Rule of Thumb:

- 1 mapper per data split (typically)
- 1 reducer per computer node (best parallelism)

MapReduce Flow

- Hadoop distributes map() to data
- Hadoop groups <key,value> data
- Hadoop distributes groups to reducers()

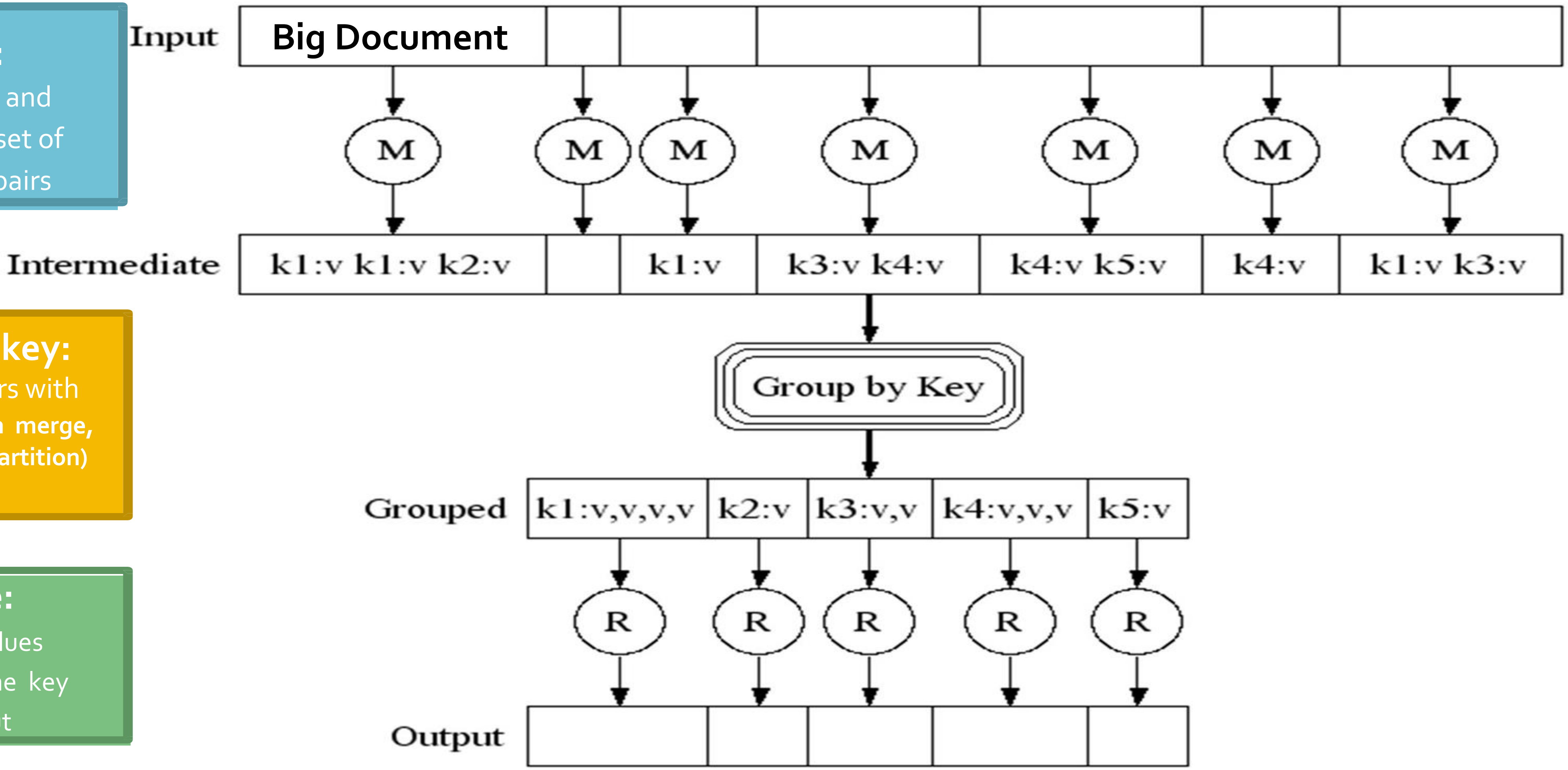


MapReduce Working Diagram

MAP:
Read input and produces a set of key-value pairs

Group by key:
Collect all pairs with same key **(Hash merge, Shuffle, Sort, Partition)**

Reduce:
Collect all values belonging to the key and output



Input: Big Document

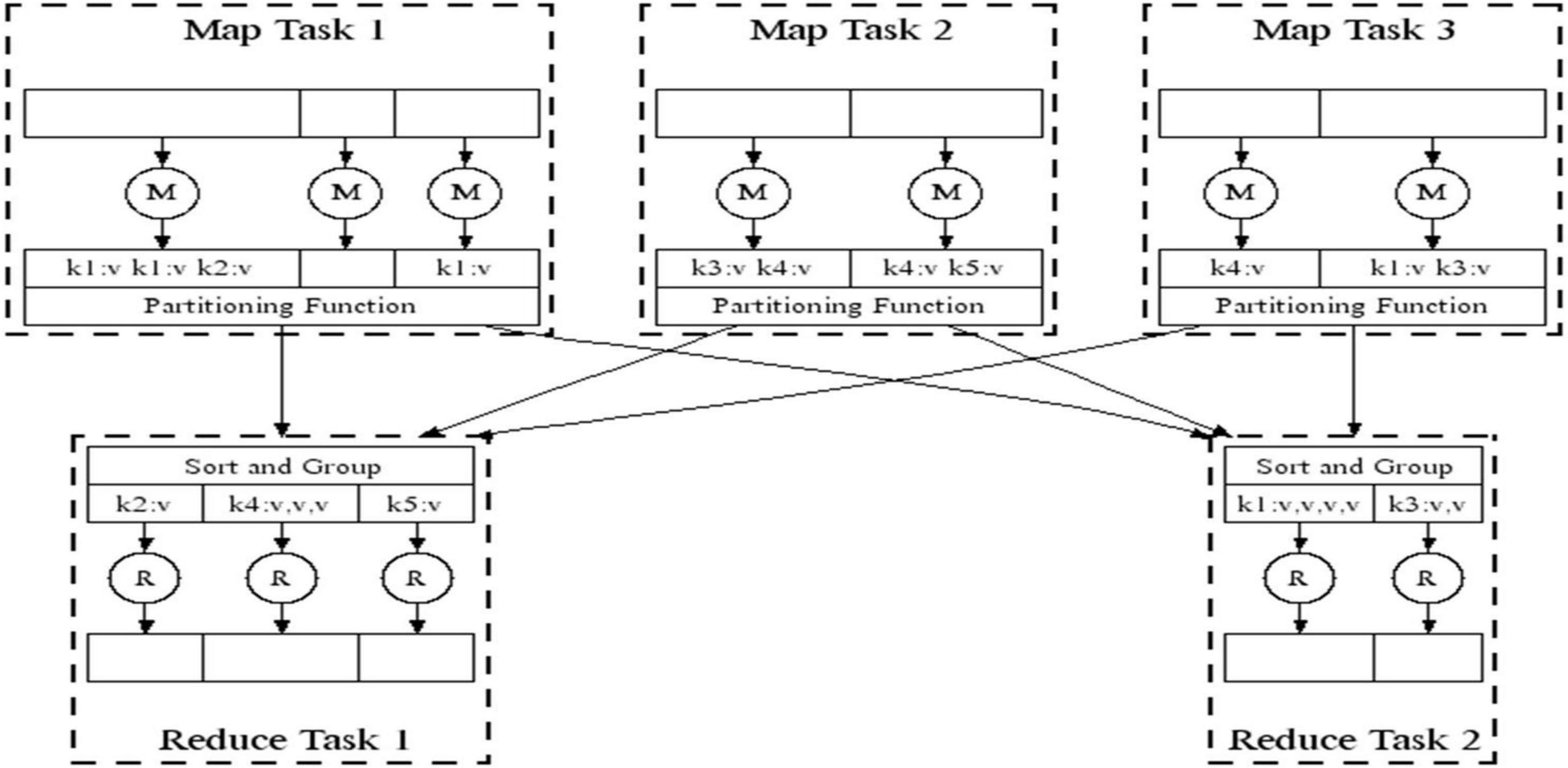
Intermediate: k1:v k1:v k2:v | | k1:v | k3:v k4:v | k4:v k5:v | k4:v | k1:v k3:v

Group by Key

Grouped: k1:v,v,v,v | k2:v | k3:v,v | k4:v,v,v | k5:v

Output

MapReduce: Parallel Processing



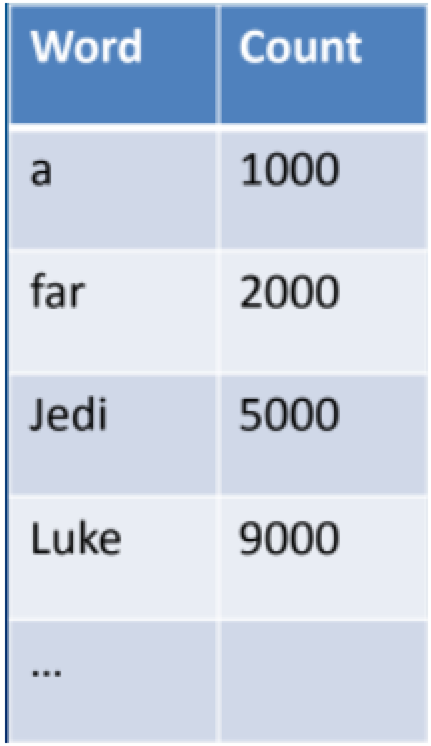
All phases are distributed with many tasks doing the work

Word Count Example

- Count word frequencies
- How would you count all the words in Star Wars?
- In a nutshell:
    - Get word
    - Look up word in table
    - Add 1 to count
- How would you count all the words in all the Star Wars scripts and books, blogs, and fan-fiction?

Episode IV

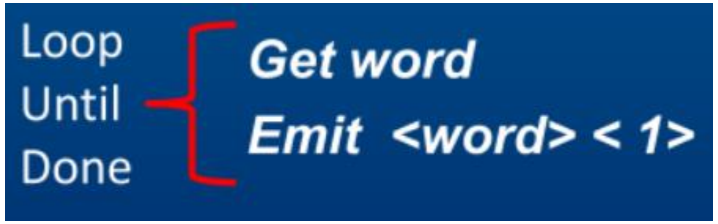
A long
time ago,
in a galaxy
far, far,
away ...



Word	Count
a	1000
far	2000
Jedi	5000
Luke	9000
…	

Strategy: Word Count

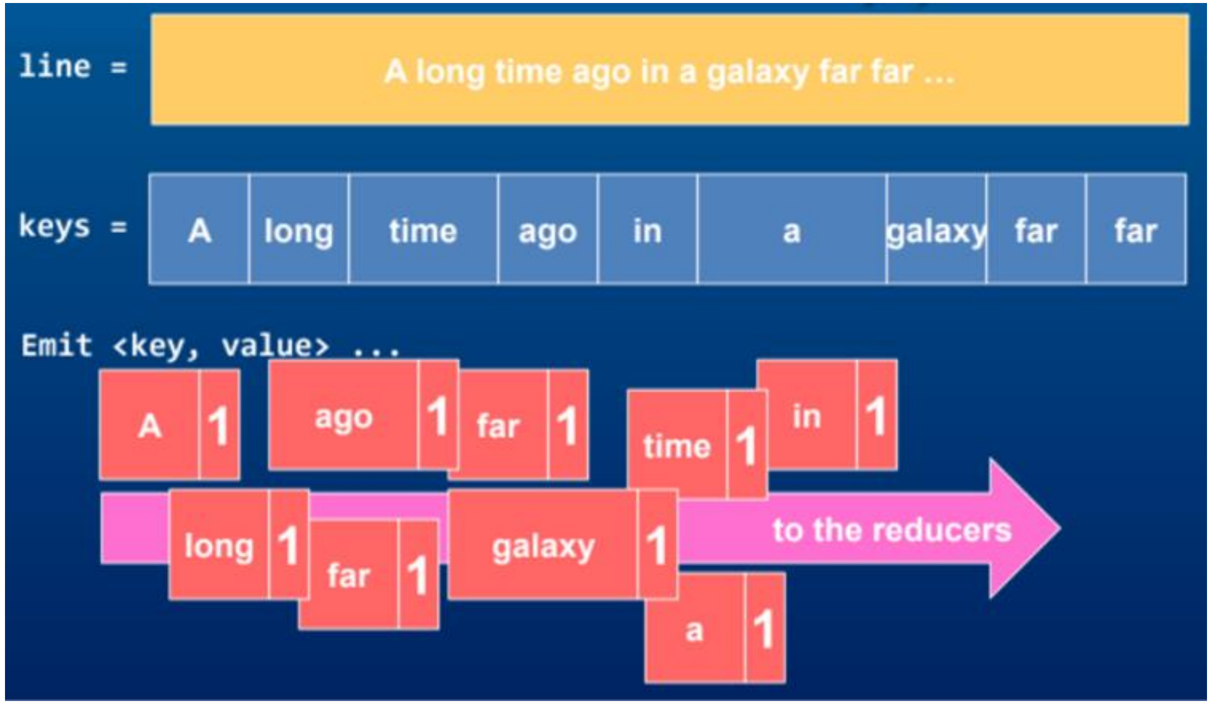
- Keep it simple (remember big data and simple aggregations)
  - Let <word, 1> be the <key,value>
  - The mapper:



```
Loop       Get word
Until  {   Emit  <word> < 1>
Done
```

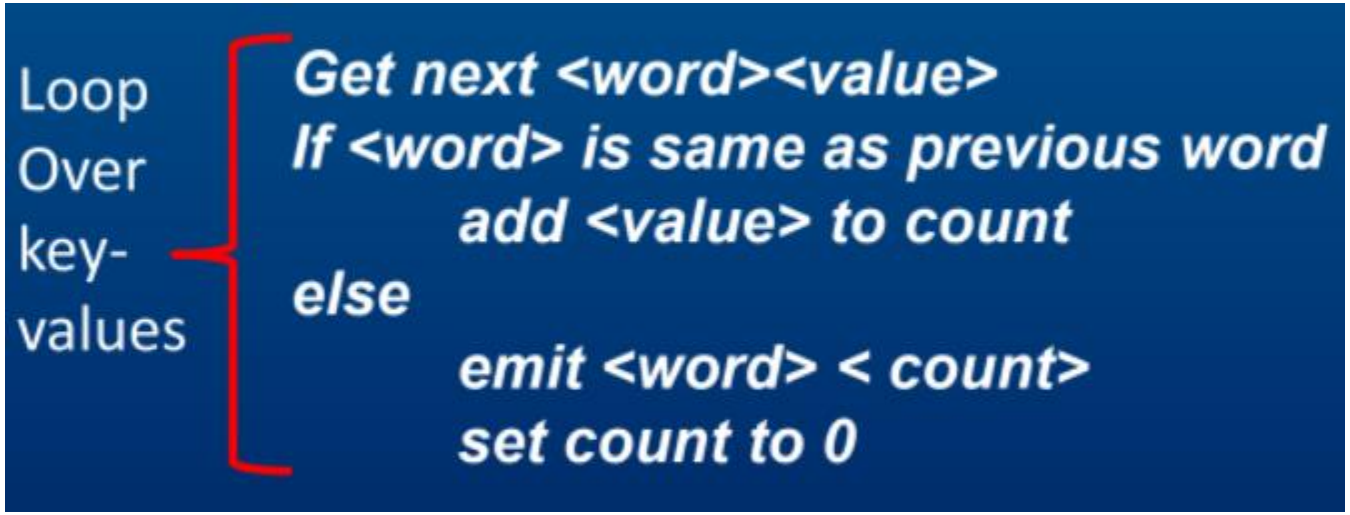
Mappers are separate and independent

Mappers work on data parts



Strategy: Word Count

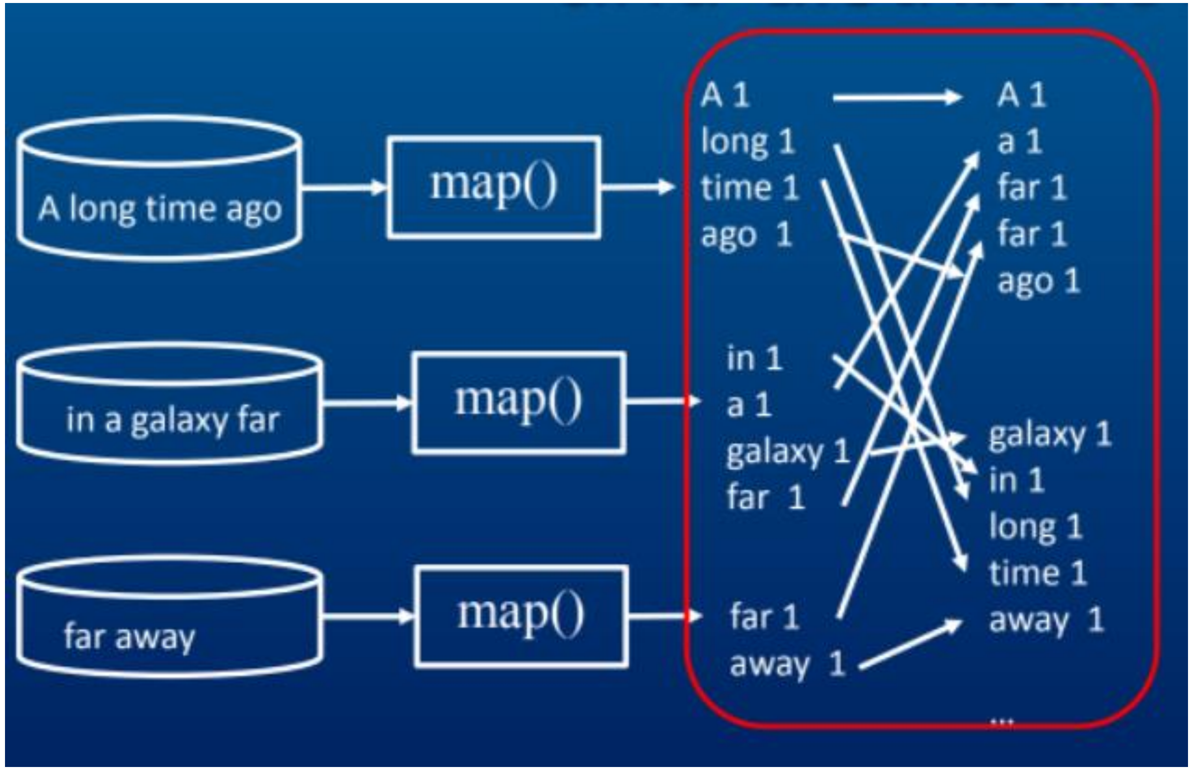
- Lets Hadoop do the hard work
  - The reducer:



```
Loop        Get next <word><value>
Over        If <word> is same as previous word
key-                add <value> to count
values      else
                    emit <word> < count>
                    set count to 0
```

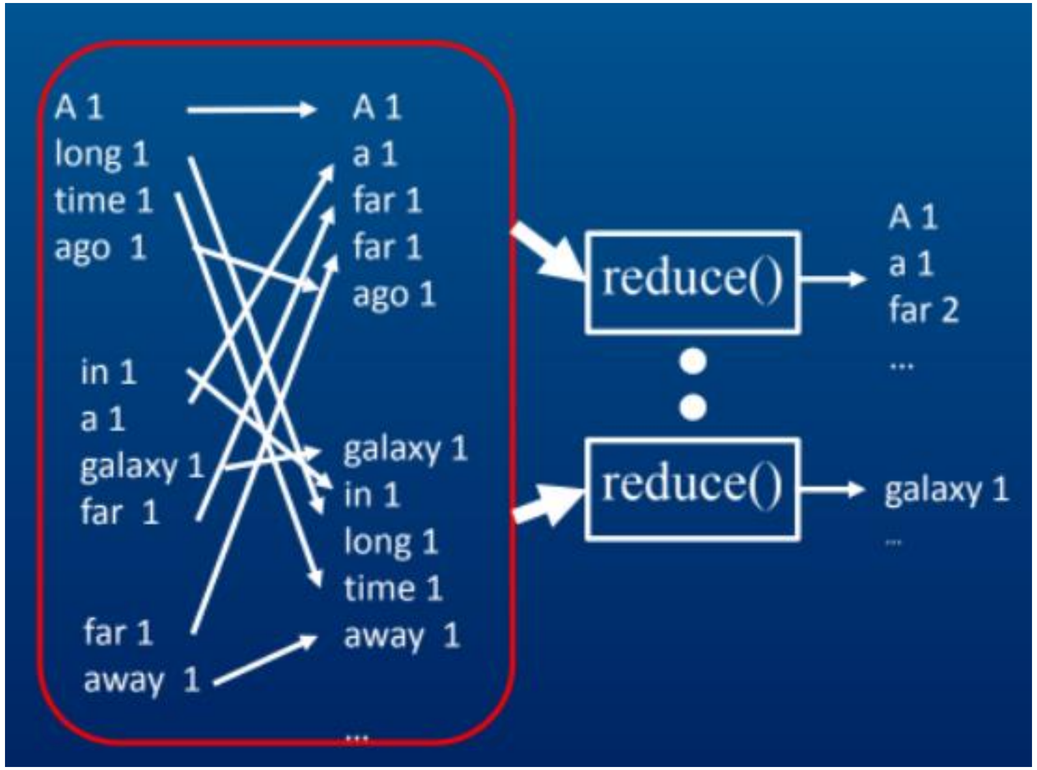
Strategy: Word Count

- Hadoop shuffles, groups, and distributes:



Strategy: Word Count

- reduce() aggregates



That’s all for today.