Hadoop Distributed File System

By : Peyman Navidi

Introduction

- Hadoop Distributed File System (HDFS)
  - is the file system component of Hadoop. It is designed to store **very large data sets** reliably, and to **stream those data sets** at high bandwidth to user applications. These are achieved by **replicating file** content on **multiple machines** (DataNodes).
  - An open-source implementation of Google File System (GFS).
  - The origin comes from Google: (95% of architecture is implemented)

- Why HDFS is required ?
  - **Big Data** : “when a dataset outgrows the storage capacity of a single physical machine, it becomes necessary to partition it across a number of separate machines.”
  - Since it is a network-based, the challenge is to make the file system adapted to bear the Node failure without any data loss.

The Design of HDFS

- Hadoop Distributed File System (HDFS)
  - "HDFS is filesystem designed for storing **very large** files with **streaming data** access patterns, running on clusters of **commodity hardware**."
- Very large files (**GB**s,**TB**s and **PB**s)
  - Hadoop clusters stores petabytes of data today.
- Streaming data access (**write once read many times**)
  - A dataset is generated or copied from a source.
  - And various analysis are performed over time.
- Commodity hardware (clusters of commodity hardware)
  - doesn't require expensive hardware
  - without noticeable interruption.

Applications for which HDFS doesn't work:

- Lots of small files:
  - The limit to the number of files in a file system is governed by the amount of memory in the namenode. "150 bytes for each file, example"
- Low-latency data access:
  - Applications that requires low-latency access will not work with HDFS, as it is optimized to deliver the a high throughput.
- Multiple writers, arbitrary file modifications :
  - No support for the multiple writers. (Random data access)

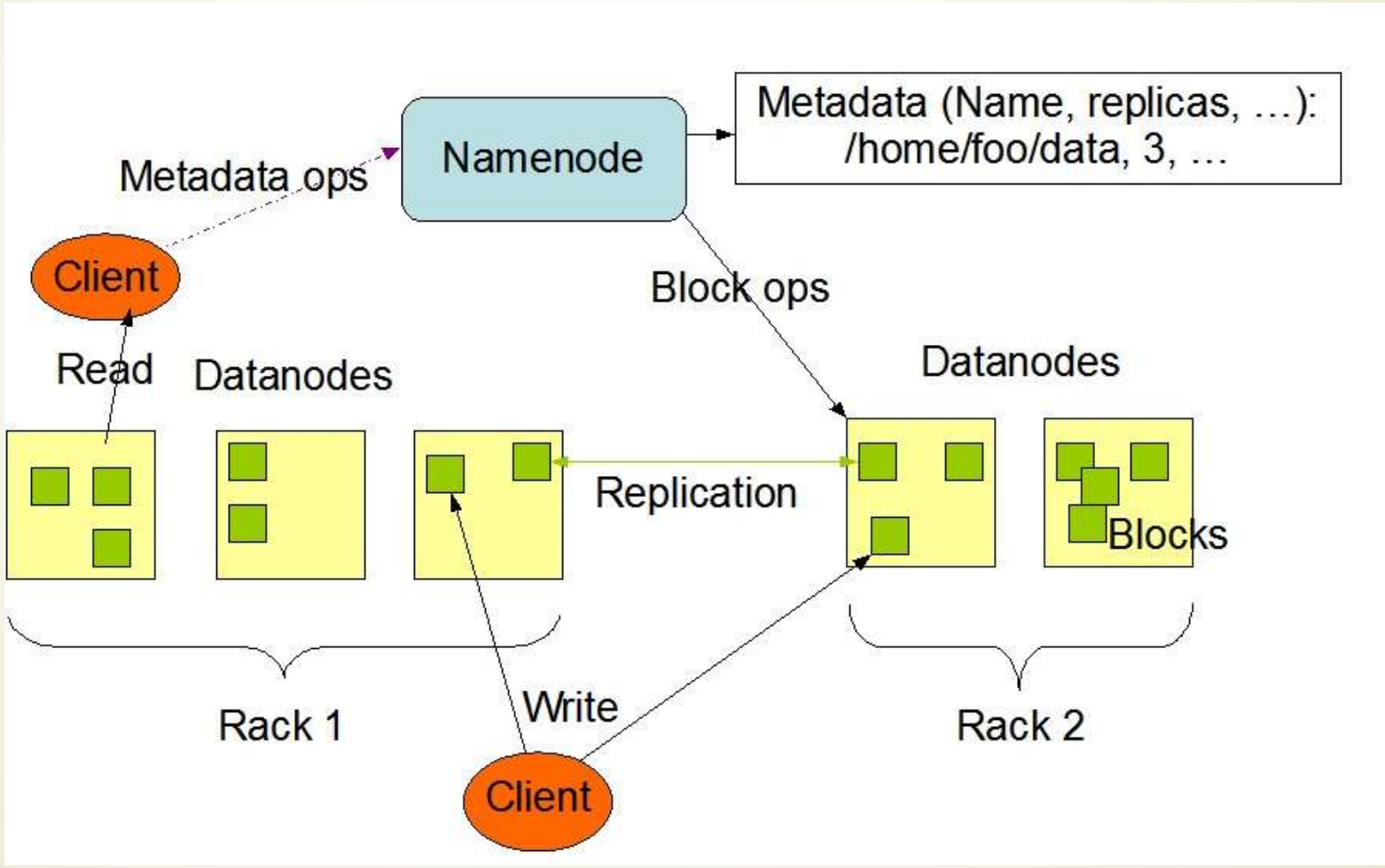
The Architecture of HDFS

- HDFS is a block-structured file system:
  - " A disk has block size, which is the minimum amount of data that it can read or write or replicate."
- Here HDFS , blocks are much larger (64 MB by default) :
  - Unlike a file system files in HDFS are broken into block-sized **chunks**, which are stored as independent units.
- 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.

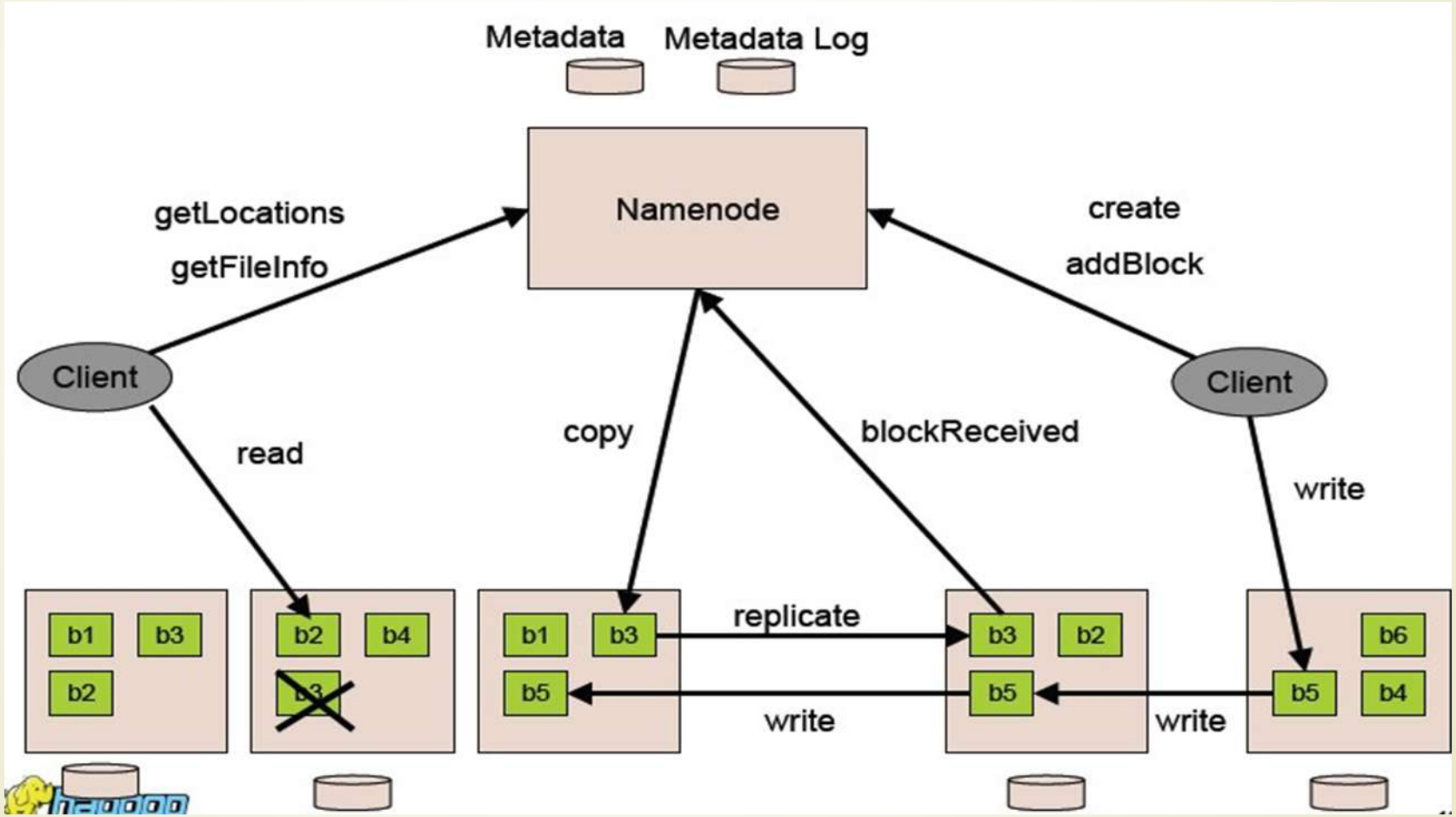
HDFS Nodes

- An HDFS cluster has two types of node operating in a master-worker pattern:
- **Namenode** (Master) :
  - 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.
- **Datanode** (Worker) :
  - 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.

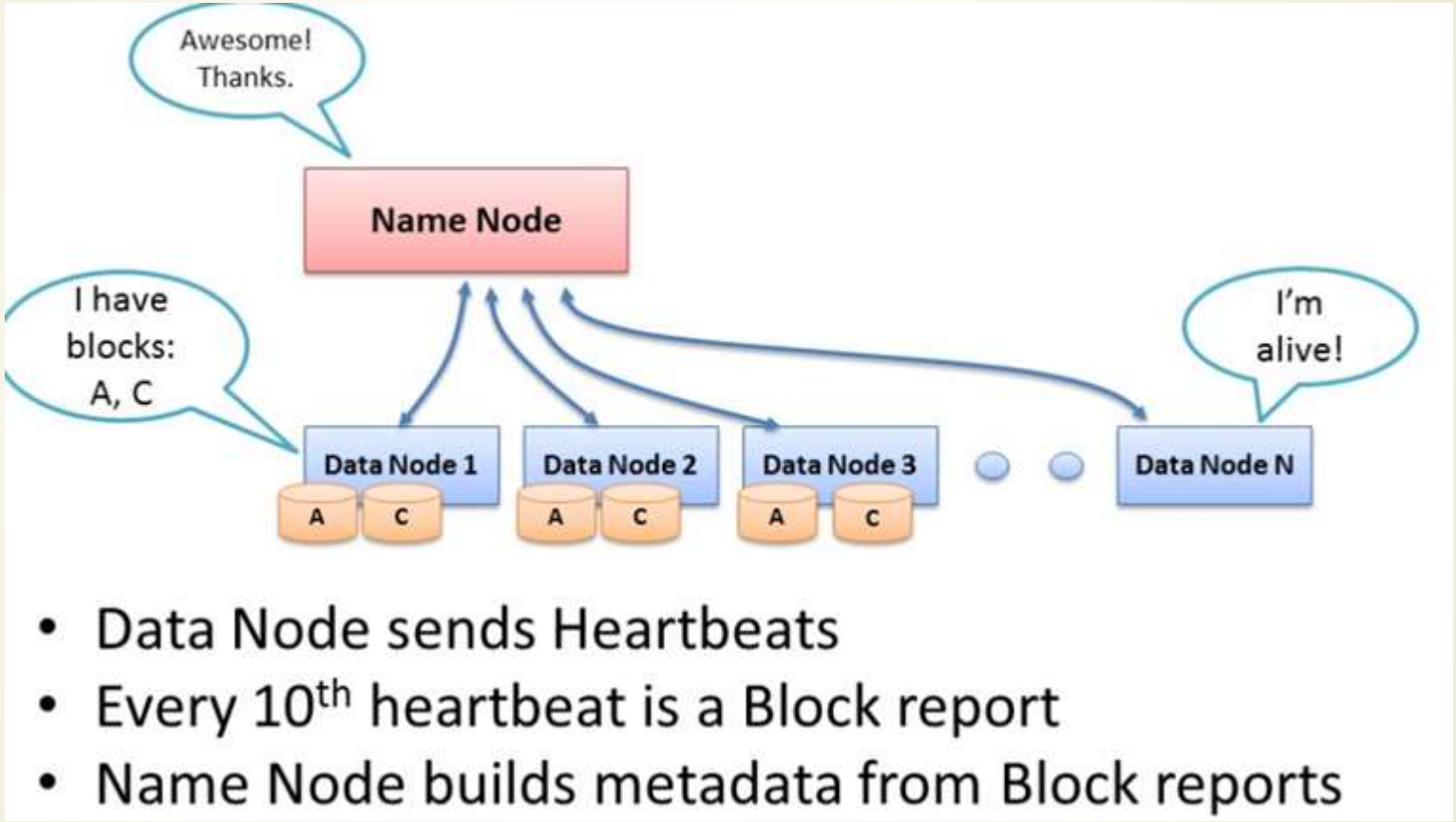
The Architecture of HDFS Task Flow



The Architecture of HDFS

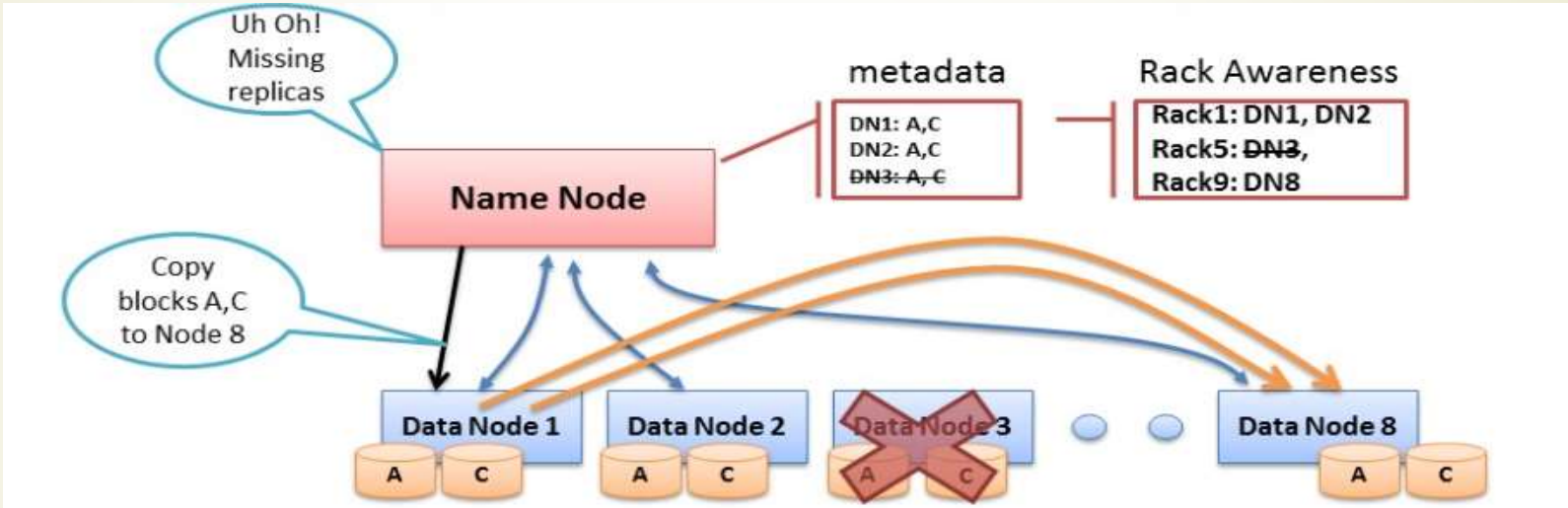


The Architecture of HDFS



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

Re-replicating Missing Replicas

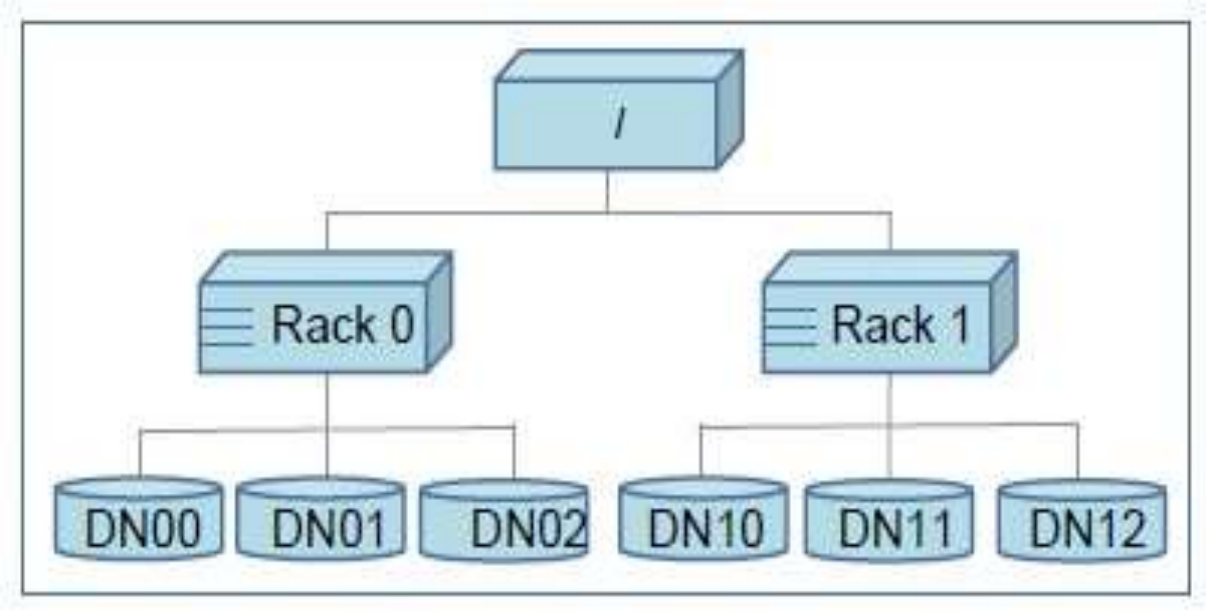


- Missing Heartbeats signify lost Nodes
- Name Node consults metadata, finds affected data
- Name Node consults Rack Awareness script
- Name Node tells a Data Node to re-replicate

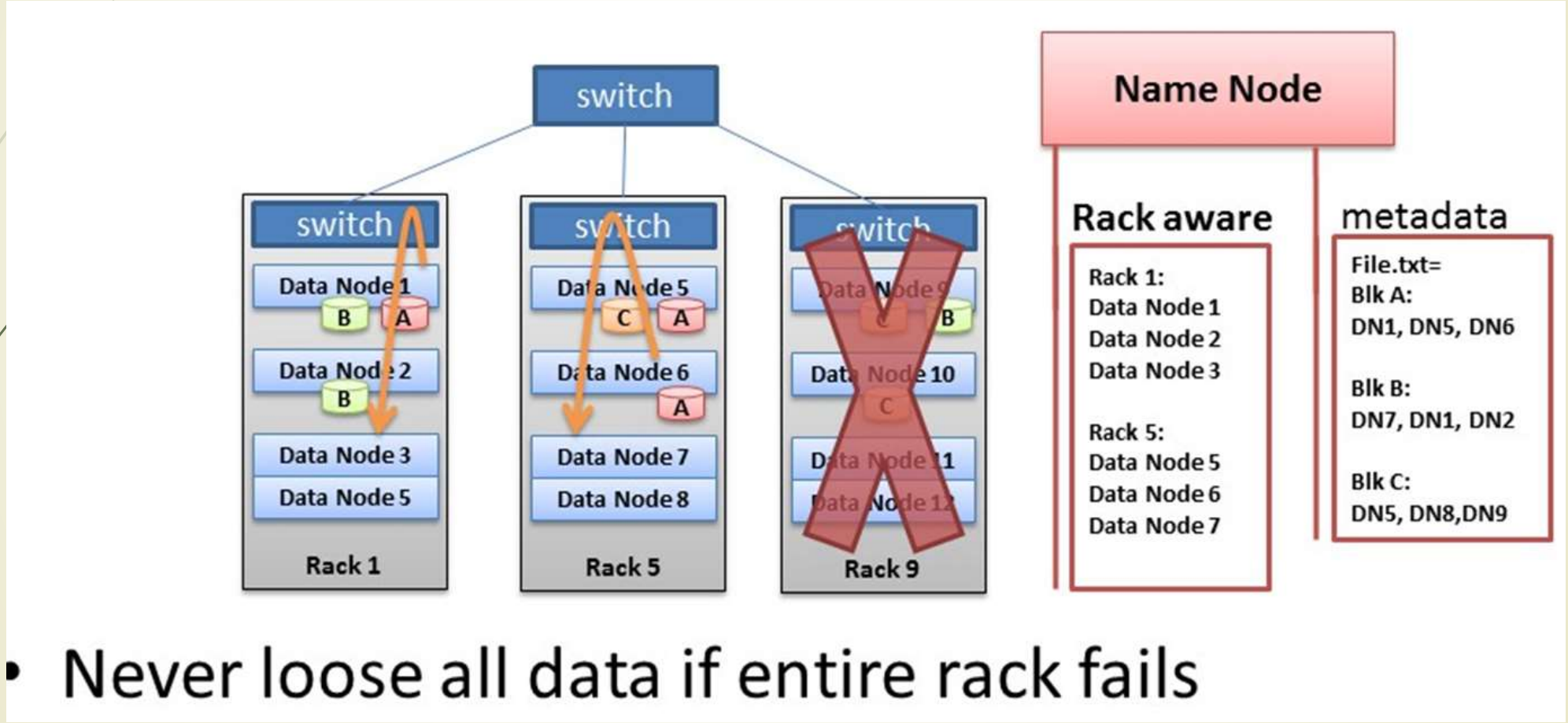
File I/O Operations and Replica Management

- 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,

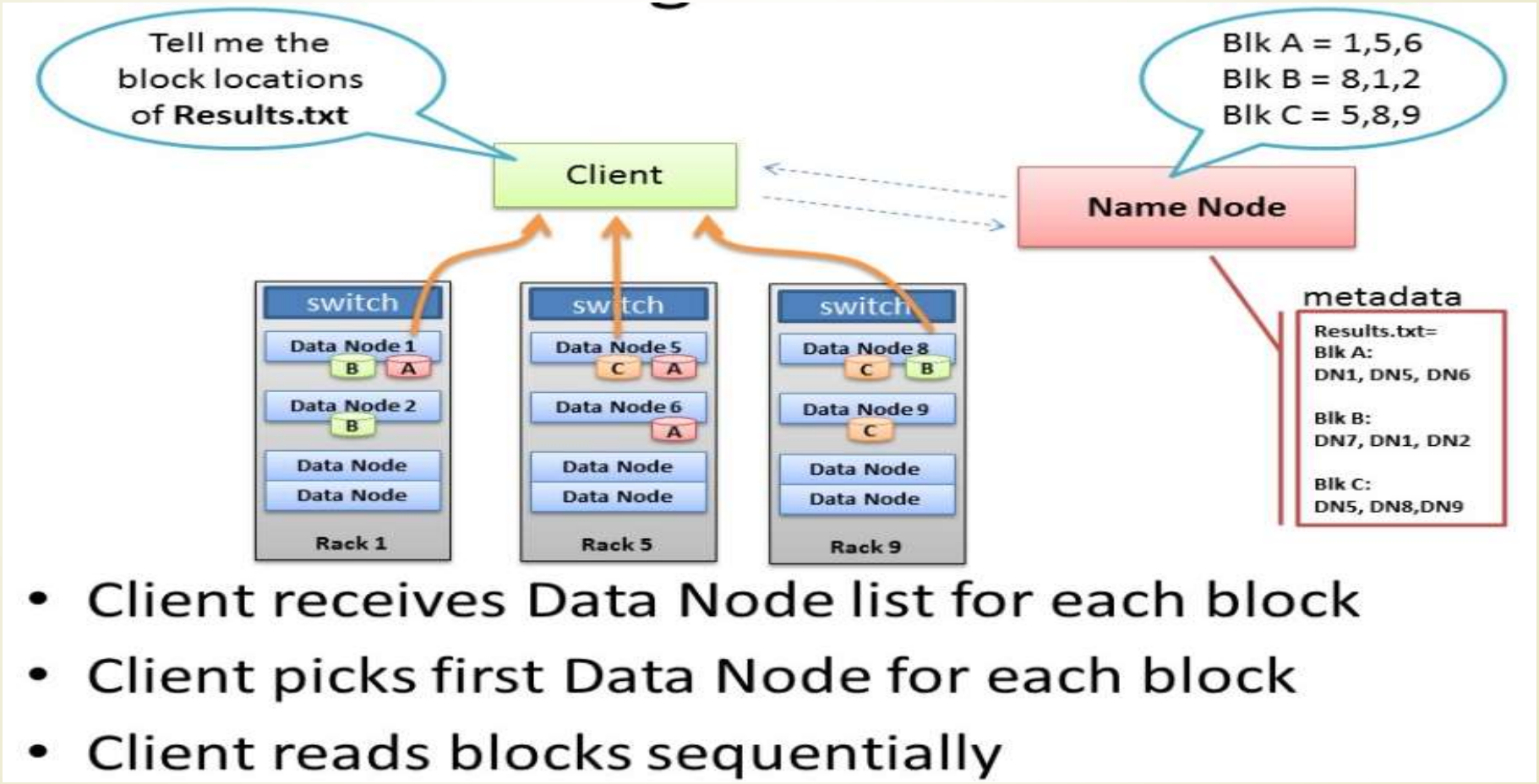


Hadoop Rack Awareness



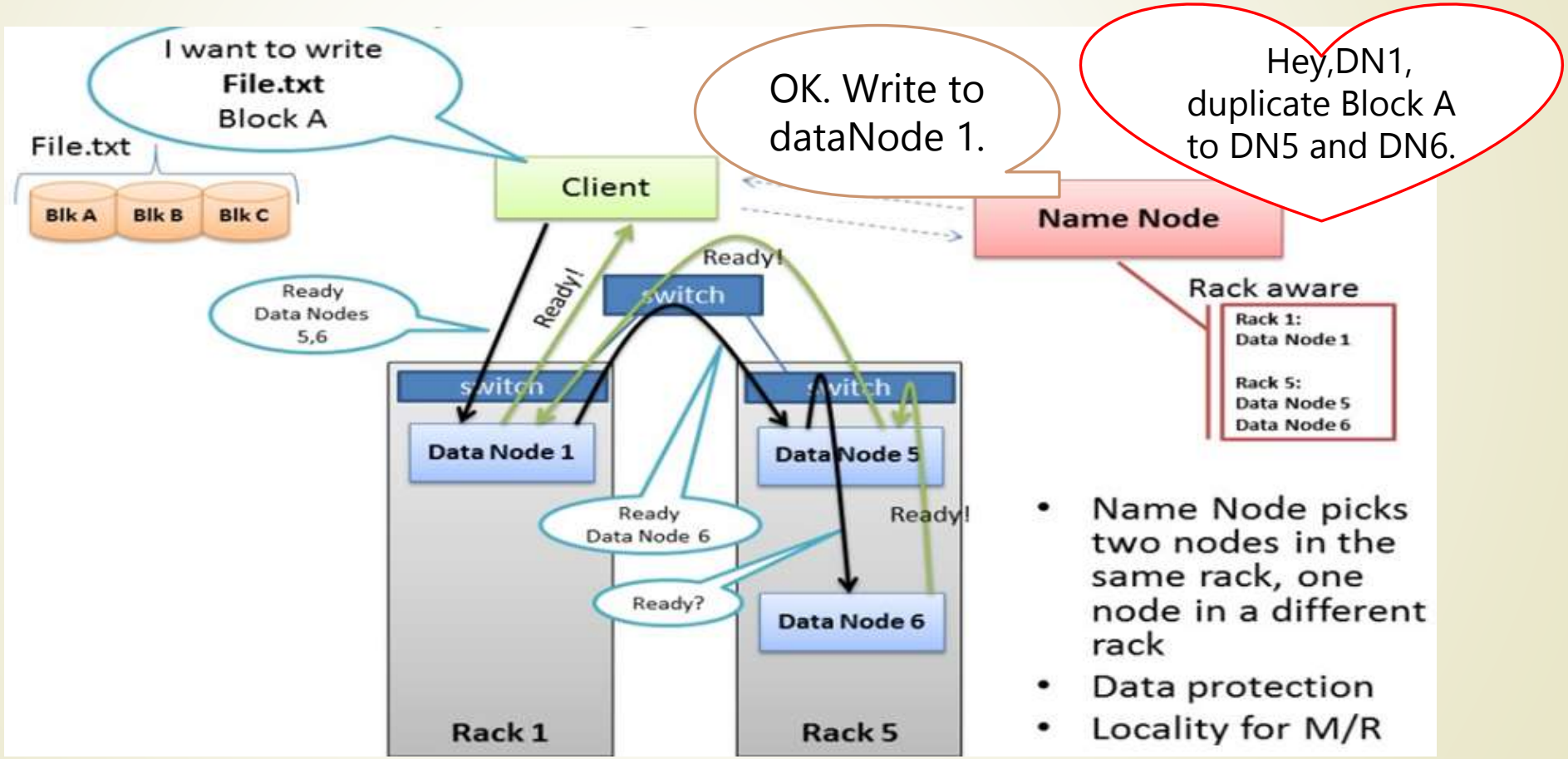
- Never loose all data if entire rack fails

Reading File from HDFS



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

Preparing HDFS Write



Basic Filesystem Operations

- We can do all of the usual filesystem operations such as **reading** files, **creating** directories, **moving** files, **deleting** data, and **listing** directories.

- Copying a file from the local filesystem to HDFS:
  - `% hadoop fs -copyFromLocal input/docs/quangle.txt hdfs://localhost/user/tom/quangle.txt`
- Copying the file back to the local filesystem:
  - `% hadoop fs -copyToLocal hdfs://localhost/user/tom/quangle.txt quangle.copy.txt`
- Creating a directory:
  - `% hadoop fs -mkdir books`
  - `% hadoop fs -ls .`

```
Found 2 items
drwxr-xr-x - tom supergroup   0 2009-04-02 22:41 /user/tom/books
-rw-r--r-- 1 tom supergroup 118 2009-04-02 22:29 /user/tom/quangle.txt
```

- Sample Run
  - `% hadoop URLCat hdfs://localhost/user/tom/quangle.txt`

Hadoop Filesystems

- HDFS is just one implementation of Hadoop filesystem.
- You can run MapReduce programs on any of these filesystems
  - But, DFS (e.g., HDFS, KFS) is better to process large volumes of data

Filesystem	URI scheme	Java implementation (all under org.apache.hadoop)
Local	*file*	fs.LocalFileSystem
HDFS	*hdfs*	hdfs.DistributedFileSystem
HFTP	*hftp*	hdfs.HftpFileSystem
HSFTP	*hsftp*	hdfs.HsftpFileSystem
HAR	*har*	fs.HarFileSystem
KFS (Cloud-Store)	*kfs*	fs.kfs.KosmosFileSystem
FTP	*ftp*	fs.ftp.FTPFileSystem
S3 (native)	*s3n*	fs.s3native.NativeS3FileSystem
S3 (block-based)	*s3*	fs.s3.S3FileSystem

Failure Recovery

- So when dataNode died, NameNode will notice and instruct other dataNode to replicate data to new dataNode.
  - Keep journal (the modification log of metadata).
  - Checkpoint: The persistent record of the metadata stored in the local host's native files system.

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

- **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.

References

- Hadoop: The Definitive Guide, Third Edition, Mike Loukides and Meghan Blanchette
- http://hadoop.apache.org/hdfs/
- Shvachko, K.; Hairong Kuang; Radia, S.; Chansler, R.; , "The Hadoop Distributed File System," Mass Storage Systems and Technologies (MSST), 2010 IEEE 26th Symposium on , vol., no., pp.1-10, 3-7 May 2010