CSCI 381/780
Cloud Computing

Hadoop

Jun Li
Queens College

Sense of data size

- 1 Byte=8 bits
- Kilobyte (1000 Bytes) = 10^3 bytes
- Megabyte (MB) = 10^6 bytes. One photo image (iPhone) ~2 MB
- Gigabyte(GB)=10^9 bytes. 8/16 GB DRAM in your MacBook
- Terabyte (TB)= 10^{12} bytes. Your hard disk: 1-3 TB
- Petabyte (PB)= 10^{15} bytes.
  - 223,000 DVDs (4.7Gb each) to hold 1PB.
  - Over 2.5 years of 24/7 4k video recording = 1 PB.

Google (2008): processed over **20 petabytes** per day, **100,000** MapReduce jobs.

Yahoo! (2011): 42,000 nodes, holding **180-200 petabytes** of data

FaceBook (2010): the largest HDFS cluster in the world with **21 PB** of storage

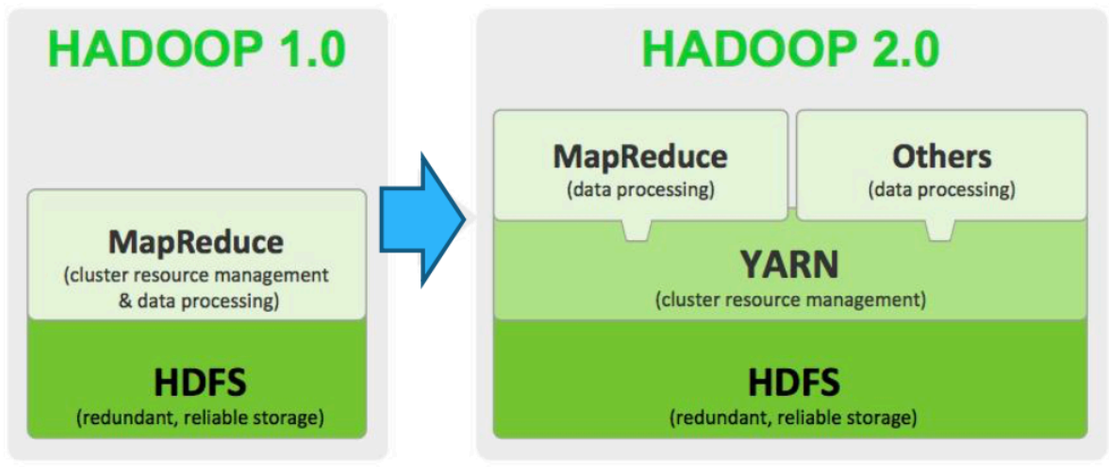
Facebook (Jan 2013): users had uploaded over 240 billion photos 357 PB of storage.

Apache Hadoop

- Composed of the following modules :
  - Hadoop Common - contains libraries and utilities needed by other Hadoop modules.
  - **Hadoop Distributed File System (HDFS)** - a distributed file system for storing data.
  - **Hadoop MapReduce** - a programming model for large scale data processing.
  - **Hadoop YARN** - resource-management and task scheduling
  - Hadoop Ozone (new in Hadoop 3.2): a distributed key-value store for Hadoop (similar to Amazon S3)
  - Hadoop Submarine (new in Hadoop 3.2): A machine learning engine for Hadoop; support Tensorflow/MXNet, GPU, Docker container.

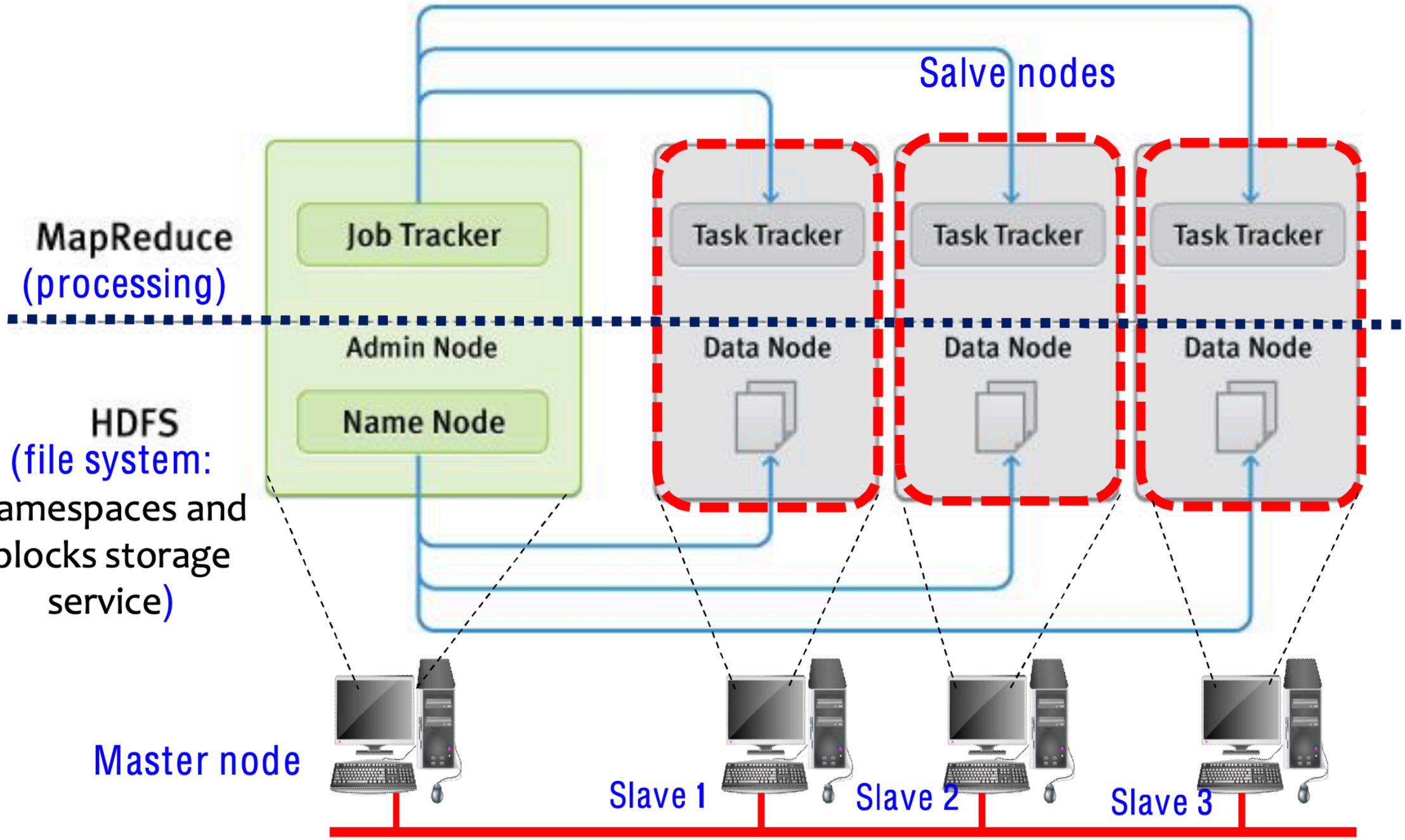
A brief history

Jan 2008: top-level project at Apache
27 Dec, 2011: Release 1.0.0
23 May 2012: Release 2.0.0-alpha
  YARN (aka NextGen MapReduce)
14 Dec., 2017: Release 2.7.5
13 Dec., 2017: Hadoop 3.0.0
Apr. 2018: Hadoop 3.1: Docker in YARN + GPU
23 Jan., 2019: Hadoop 3.2.0 (Ozone, Submarine)



Hadoop 2.x takes Hadoop beyond batch applications: Allows for multiple ways to interact with data stored in HDFS : (1) batch with MapReduce (MR), (2) streaming with Spark/Storm, (3) interactive SQL with Spark SQL, Hive.

Hadoop 1.x Architecture



Hadoop 1.x key components

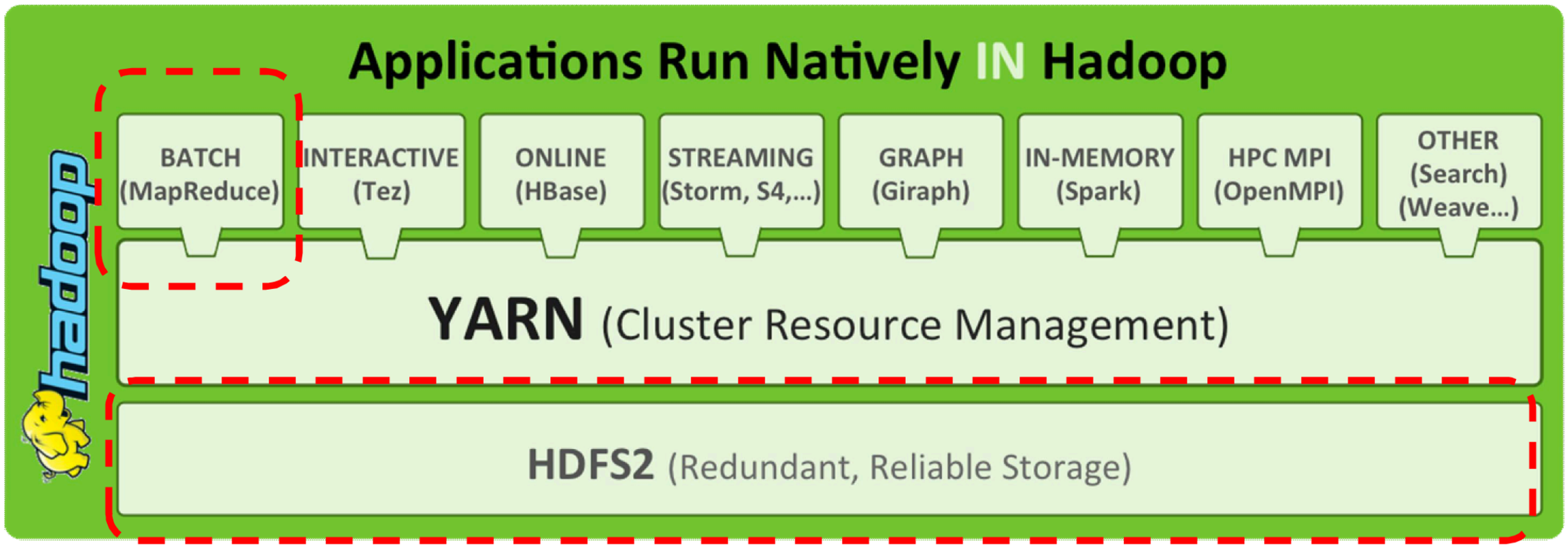
- Name Node @ master node
  - Stores metadata : file/chunk namespaces, file-to-chunk mapping, location of each chunk's replicas. All in memory!
  - Oversees and coordinates the data storage function.
- Data Node @ each slave node
  - a slave to the Name Node
  - stores data on the local file system (e.g., Linux ext3/ext4).

Hadoop 1.x key components

- Job Tracker @ master node
  - Keeps track of all the MapReduce jobs that are running on various nodes.
- Task Tracker @ each slave node
  - a slave to the Job Tracker.
  - Launches child processes (JVMs) to execute the map or reduce tasks

Hadoop 2.x architecture

- YARN supports multiple ways to interact with the data in HDFS: Spark for real-time processing, Hive for SQL, HBase for NoSQL, Pig for ETL data pipelines, and others.

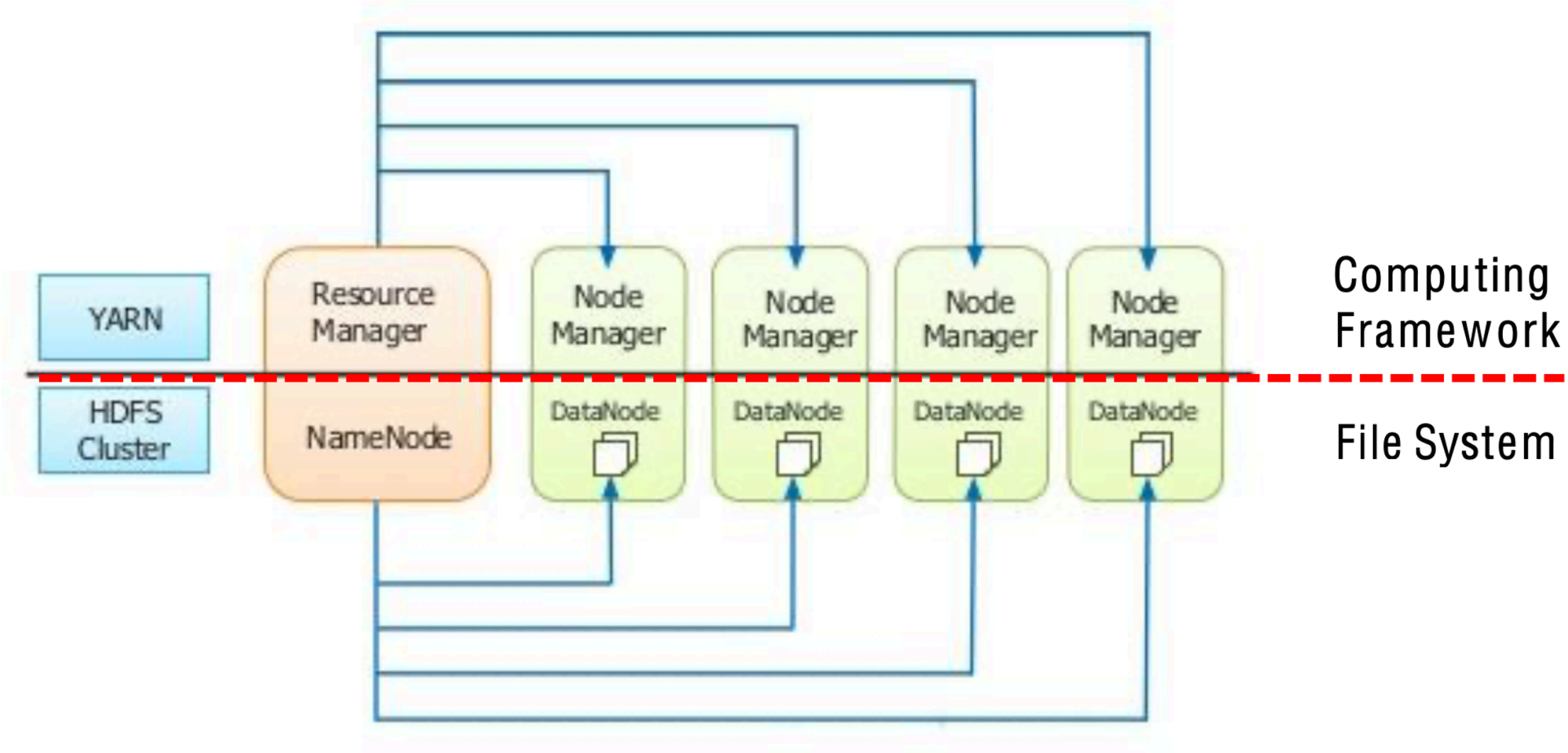


Hadoop 2.x key components

- In Hadoop 2.x, the JobTracker and TaskTracker no longer exist and have been replaced by 3 components: (1) Resource Manager, (2) NodeManager, (3) ApplicationMaster (more in Yarn)

- Roles of the cluster nodes:
  - Master Node(s): Typically one machine in the cluster is designated as the NameNode (NN) and another machine as the Resource Manager (RM), exclusively.
    - For simplicity, we can put NN and RM at the same VM
  - Slave Nodes: The rest of the machines in the cluster act as both DataNode (DN) and Node Manager (NM). These are the slaves.

Hadoop 2.x: HDFS + Yarn



Hadoop in the cloud

- Microsoft Azure: Azure HDInsight
- Amazon Elastic MapReduce (EMR)
- Google Cloud Platform
  - Google Cloud Dataproc: PaaS running Apache Spark and Apache Hadoop clusters
- Oracle Cloud Platform
  - Oracle Big Data SQL Cloud Service

Hadoop Distributed File System (HDFS)

Google File System

- Goal: a global (distributed) file system that stores data across many machines
  - Need to handle 100's TBs

- Google published details in 2003

- Open source implementation:
  Hadoop Distributed File System (HDFS)

Workload-driven design

- Google workload characteristics – Huge files (GBs)
  - Almost all writes are appends
  - Concurrent appends common
  - High throughput is valuable
  - Low latency is not

Workload examples

- Read entire dataset, do computation over it
- Producer/consumer: many producers append work to file concurrently; one consumer reads and does work

Workload-driven design

- Build a global (distributed) file system that incorporates all these application properties
- Only supports features required by applications
- Avoid difficult local file system features, e.g.:
  - rename dir
  - links

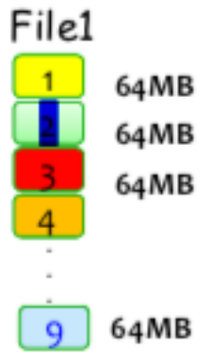
Design details

- Files stored as blocks

  Why is a data chunk in HDFS so large (64 MB or 128MB)?

  - an HDFS file is chopped up into 64MB/128MB blocks

  - each block will reside on a different datanode.



- Single master to coordinate access, keep metadata

  - Simple centralized master per Hadoop cluster

  - Manages metadata (doesn't store the actual data chunks)

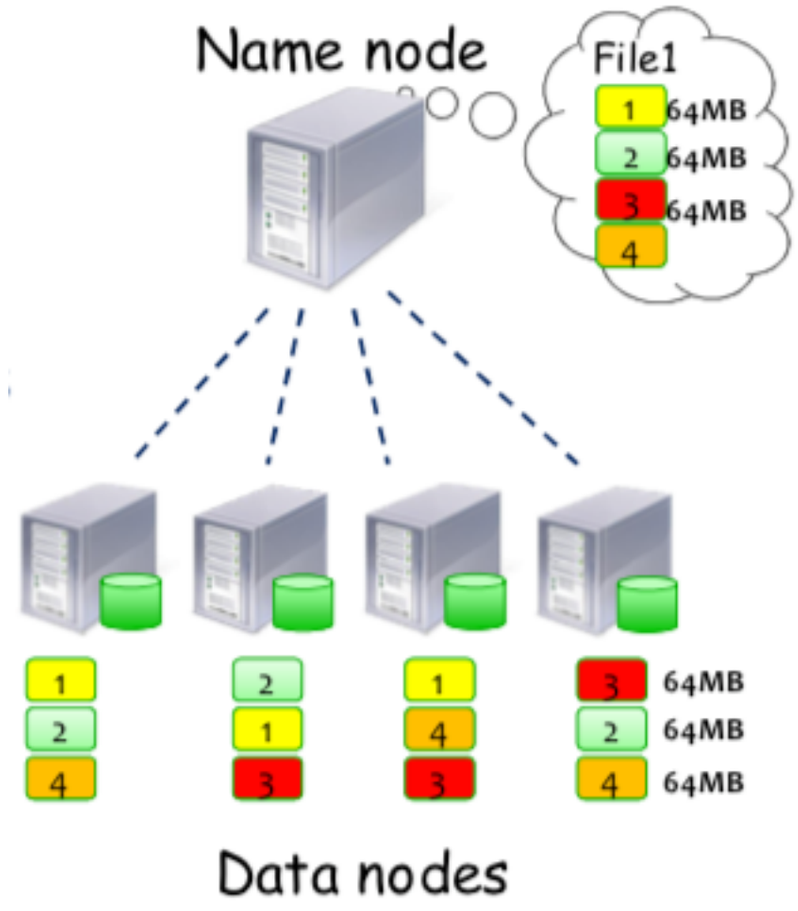
  - Periodic heart beat messages to check up on slave servers

Reliability

- “Hardware failure is the norm rather than the exception.”
  - Hundreds of thousands of machines / disks (cheap but unreliable)
  - Each component has a non-trivial probability of failure
    - 100K drives (MTBF=3 years). one “failure” every 15 minutes! (Google)
    - Add in H/W failures for network, memory, power, etc.
- Reliability through replication
  - Each block is replicated across 3+ datanodes
  - Stored as local files on Linux file system (Ext3/Ext4)

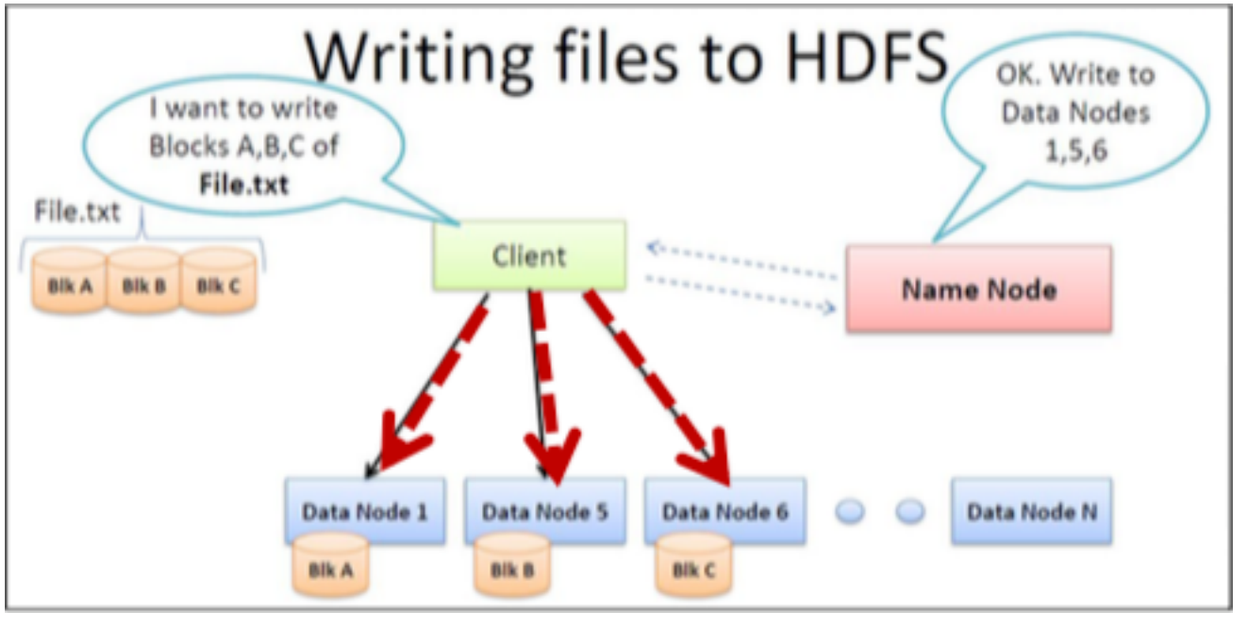
HDFS — A Quick Summary

- HDFS is written in Java.
- Scale to tens of petabytes of storage.
- Files split into blocks (default 64 or 128 MB), replicated across several data nodes (default 3) for fault tolerance.
- Single name node stores metadata (file names, block locations, etc.)
- Optimized for large files, sequential reads (or append-only).
- Files in HDFS are write-once and have strictly one writer at any time (different from GFS).
- Emphasis is on high throughput of data access rather than low latency of data access.



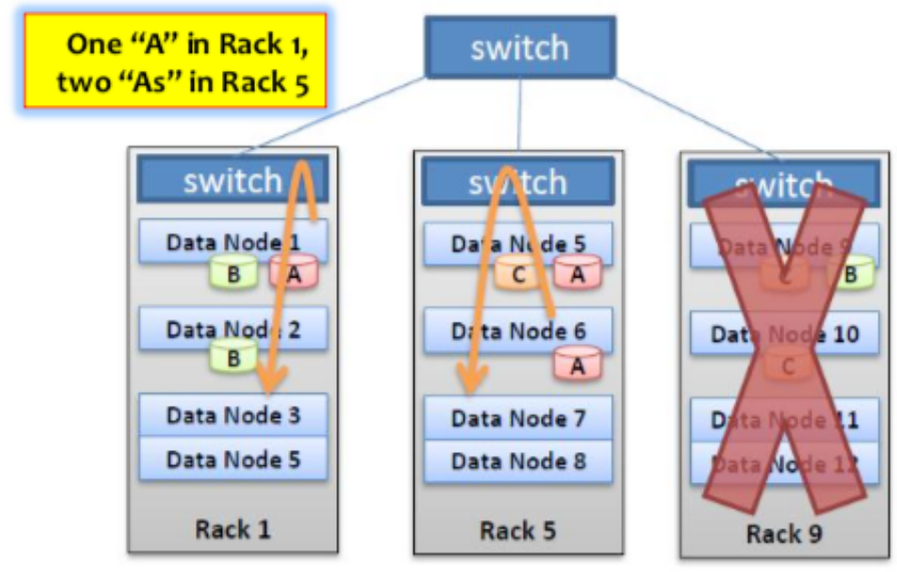
Writing files to HDFS

- The Client breaks File.txt into Blocks (3 blocks: A, B, C)
  - For each block, Client consults NameNode.
  - Client writes block directly to one DataNode.
  - DataNode replicates block (not shown, more to come).



Replica placement policy

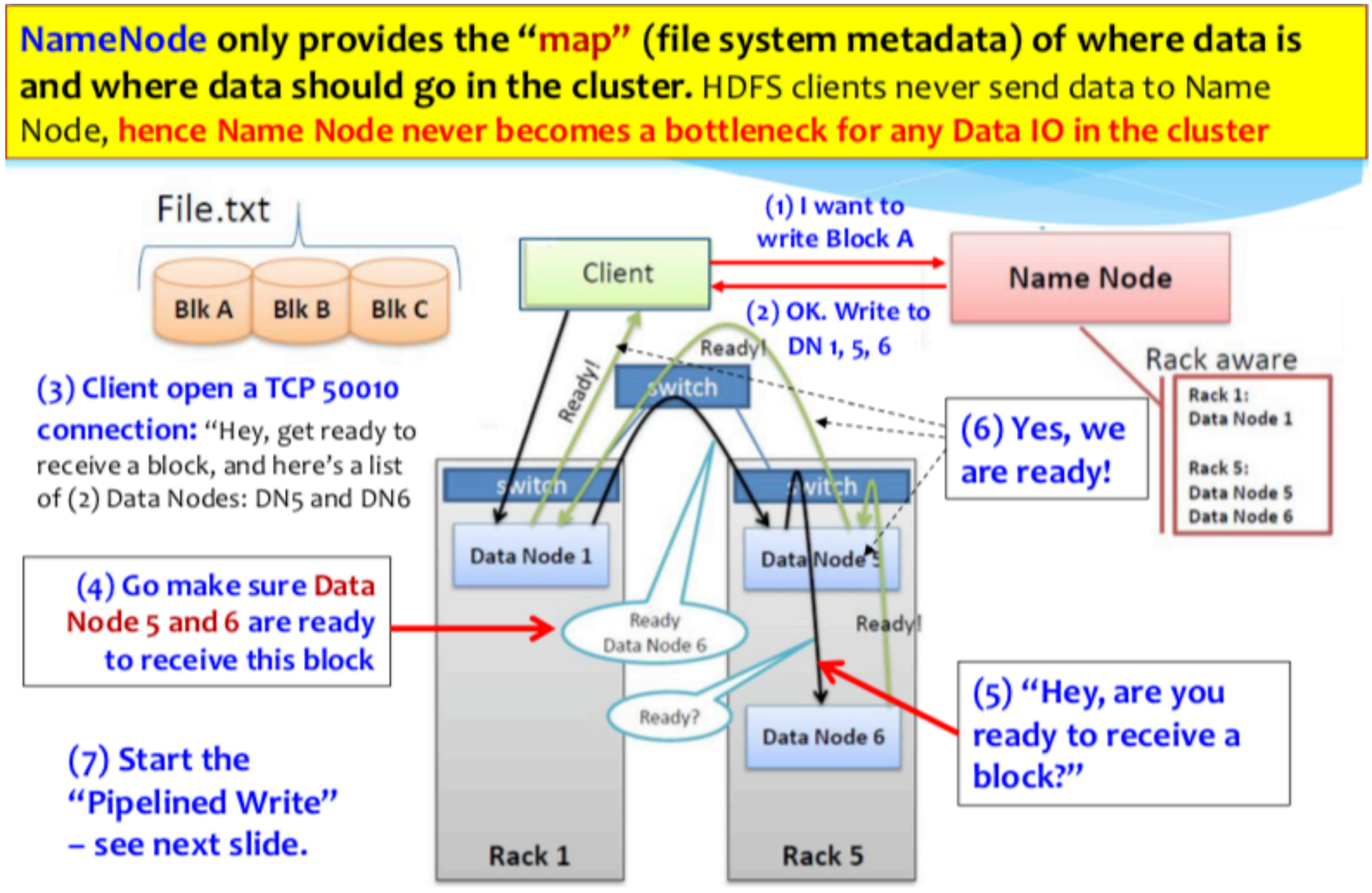
- (1) Network performance issue:
  - Communication in-rack: higher bandwidth, lower latency (good for performance)
  - Keep bulky flows in-rack when possible
- (2) Data loss prevention
  - Never lose all data even the entire rack fails
- Improve data reliability, availability, and network bandwidth utilization



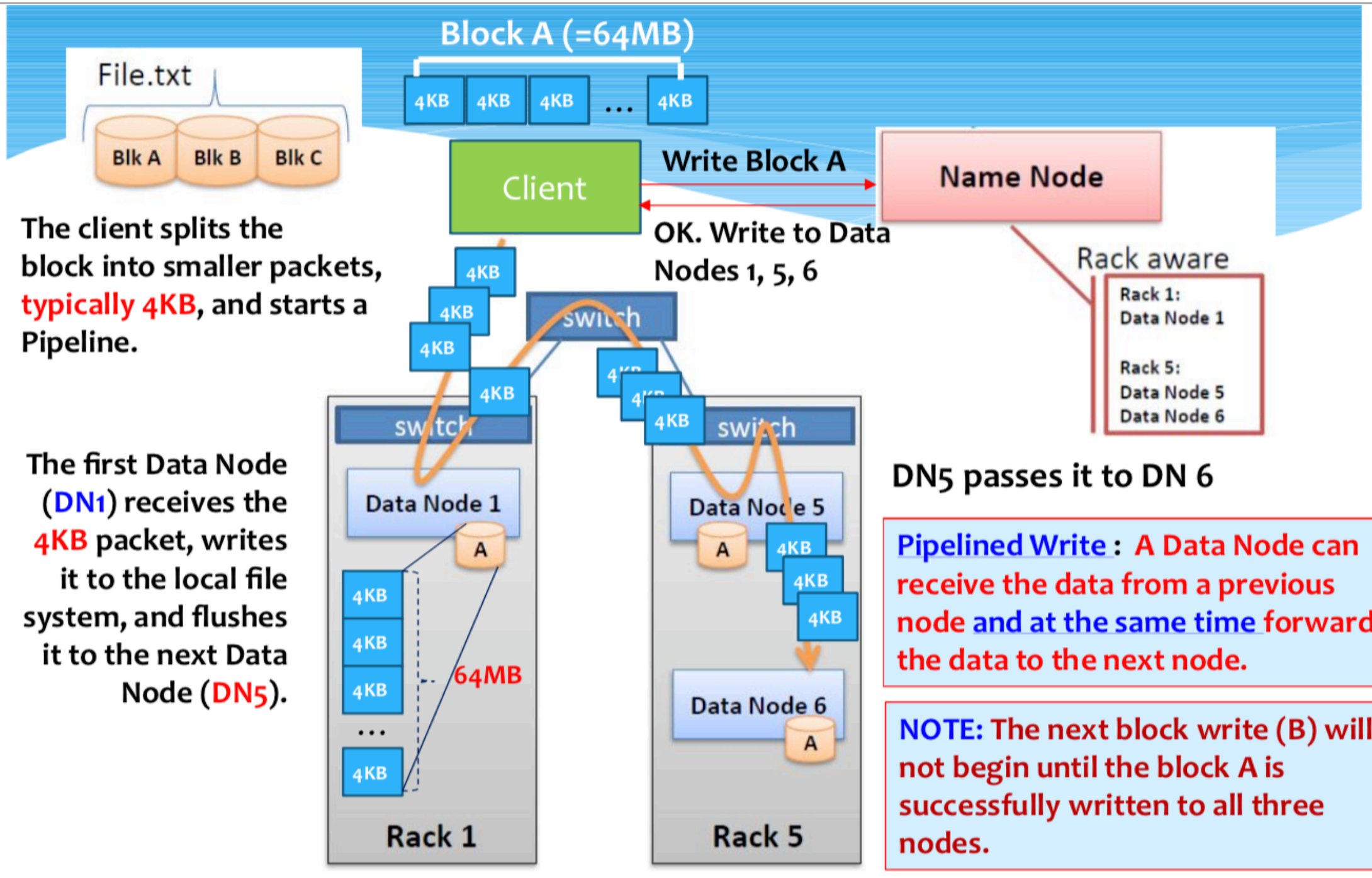
Replica placement policy

- Replication factor = 3
  - 1st one on the same node as the client (the writer), otherwise on a random datanode.
  - 2nd on a node in a different rack (off-rack)
  - 3rd on the same rack as the 2nd one, but on a different node
- Why so?
  - Tradeoff between reliability and write bandwidth and read bandwidth (discussed next slide)

Writing replicas



Writing replicas



The client splits the block into smaller packets, typically 4KB, and starts a Pipeline.

The first Data Node (DN1) receives the 4KB packet, writes it to the local file system, and flushes it to the next Data Node (DN5).

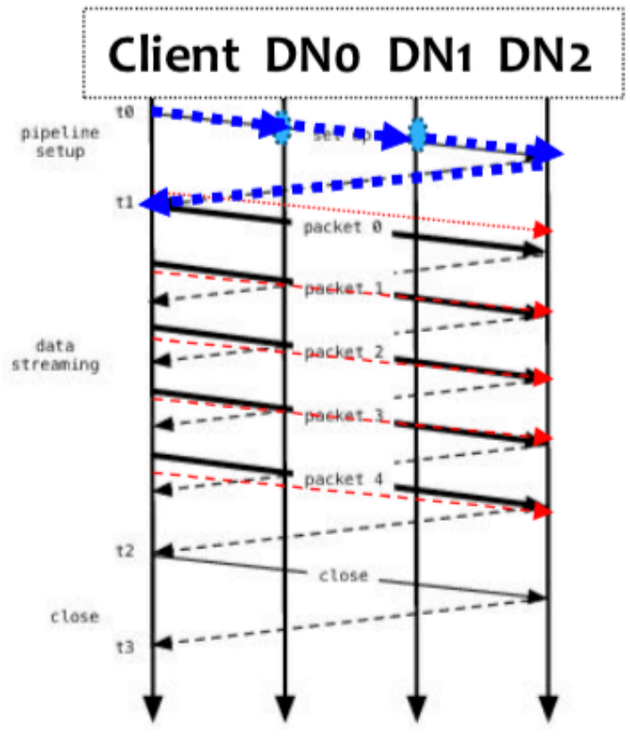
DN5 passes it to DN 6

Pipelined Write : A Data Node can receive the data from a previous node and at the same time forward the data to the next node.

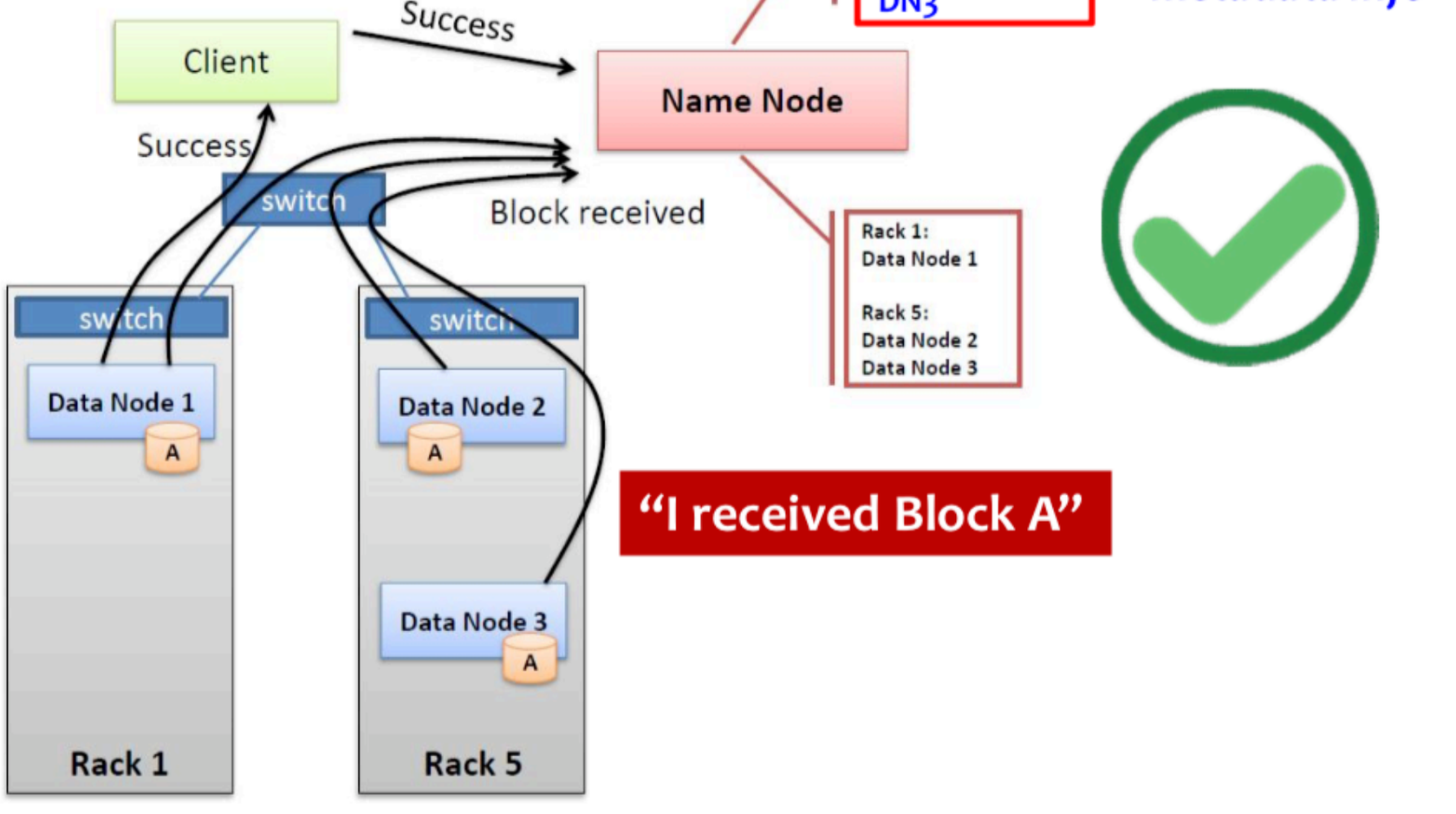
NOTE: The next block write (B) will not begin until the block A is successfully written to all three nodes.

Writing replicas

- When completed, each datanode reports to namenode "block received" with block info.

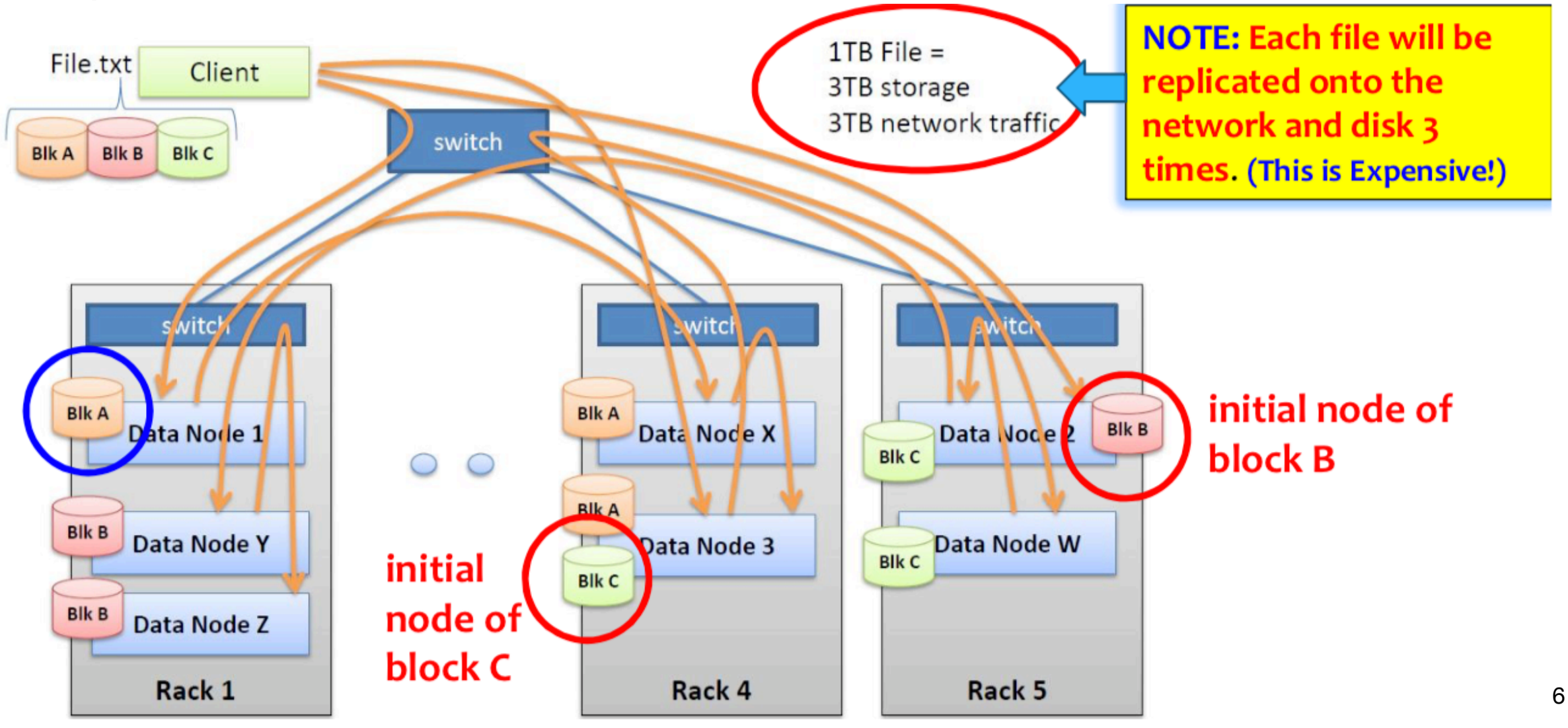


Data Pipeline While Writing a Block



Writing replicas

- Note: The initial node of the subsequent blocks of File.txt will vary for each block (why?) Spreading around the hot spots of in-rack and cross-rack traffic.

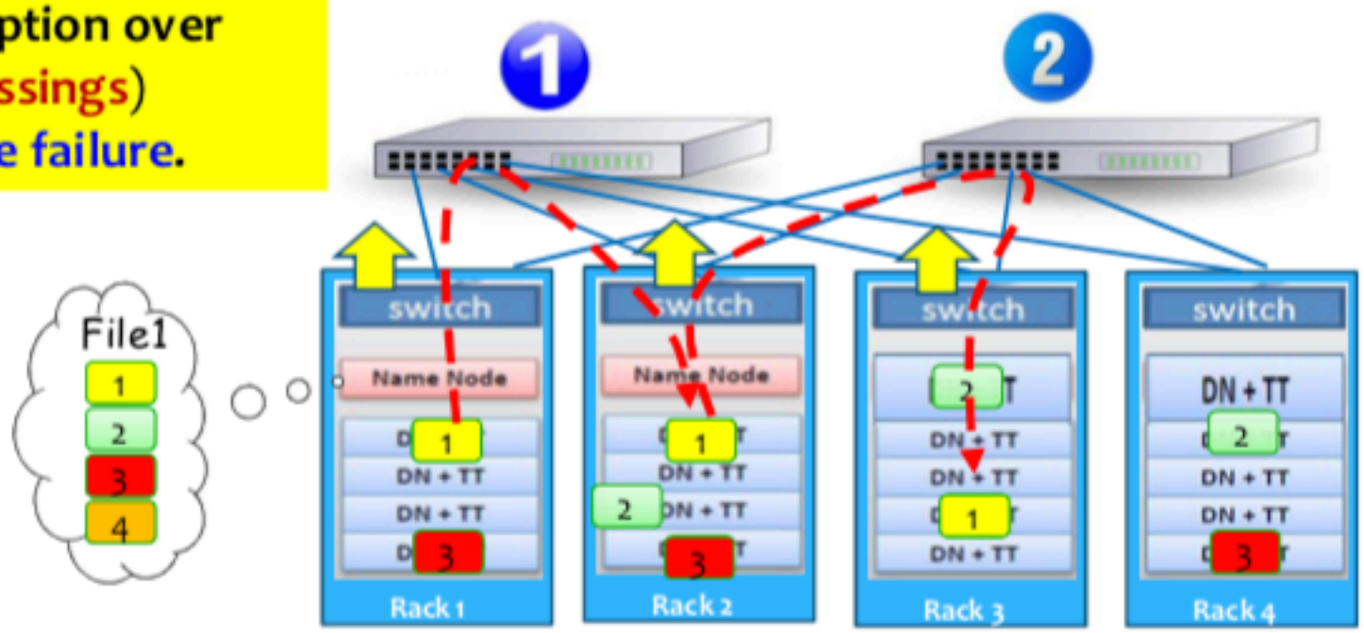


Discussion

- Why not put them in THREE nodes located at three different racks?

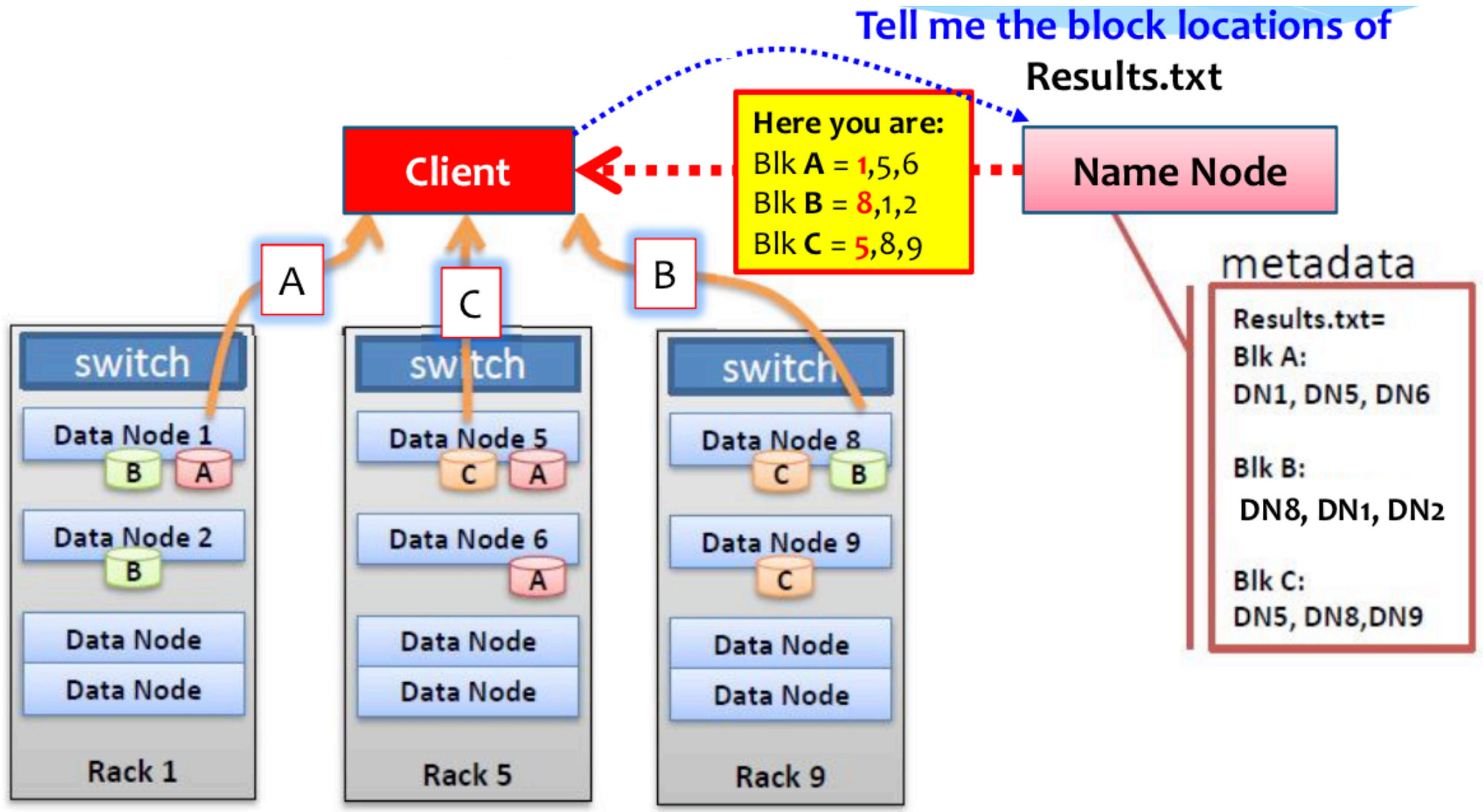
- Seem Good: This maximizes redundancy (better fault tolerance).

But at the cost of higher bandwidth consumption over ToR switches (2 crossings)
Rack failure << node failure.



Reading files from HDFS

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



Heartbeats & block reports

- Data Node sends Heartbeats to Name Node every 3 seconds

- Every 6 hours is a (full) block report (like “I have block A & C”)

  - Block reports: provide the NameNode with an up-to-date view of where block replicas are located on the cluster.

- NameNode builds meta data from Block reports