DSC650: Data Technology and Future Emergence

Lecture 3: **Data Storage Technology**

Lecturer: Dr. Khairul Anwar Hj. Sedek

Lecture Outlines

- Evolution of Data Storage:
- On-Disk Storage,
- Distributed File System,
- RDBMS,
- NoSQL
- Comparison between SQL and NoSQL Database
- Hadoop Distributed File System (HDFS)

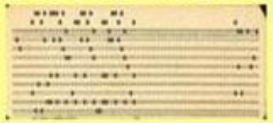
At the end of the lecture, students should be able to;

Demonstrate an understanding on the basic **concepts and practices** of big data technology (CLO1)

Evolution of Data Storage

EVOLUTION OF DATA STORAGE FROM PUNCH CARDS TO THE CLOUD

Punch Cards
1837 to mid-1980s



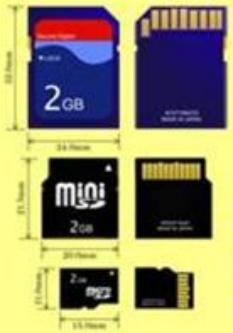
A 12-row/80-column IBM punched card from the mid-twentieth century
Pete Birkinshaw, CC BY 2.0, via Wikipedia
https://en.wikipedia.org/wiki/Punched_card#/media/File:Used_Punchcard_(5151286161).jpg

Floppy Disks
1967 to mid-1990s

8-inch, 5 ¼-inch, and 3 ½-inch floppy disks
George Chernilevsky, Wikimedia Commons, via Wikipedia
https://commons.wikimedia.org/wiki/File:Floppy_disk_2009_G1.jpg

Flash Drives
Available: 1999

A SanDisk-brand USB thumb drive, SanDisk Cruzer Micro, 4GB
Evan-Amos, Wikimedia Commons, via Wikipedia
https://en.wikipedia.org/wiki/File:SanDisk-Cruzer-USB-4GB-ThumbDrive.jpg



Secure Digital (SD) Cards
Available: 1999

SD card, miniSD card and microSD card
(Original Work: 毛抜き Derivative Work: Tkgd2007)
Tkgd2007, Wikimedia Commons, via Wikipedia
https://en.wikipedia.org/wiki/File:SD_Cards.svg

Cloud Storage

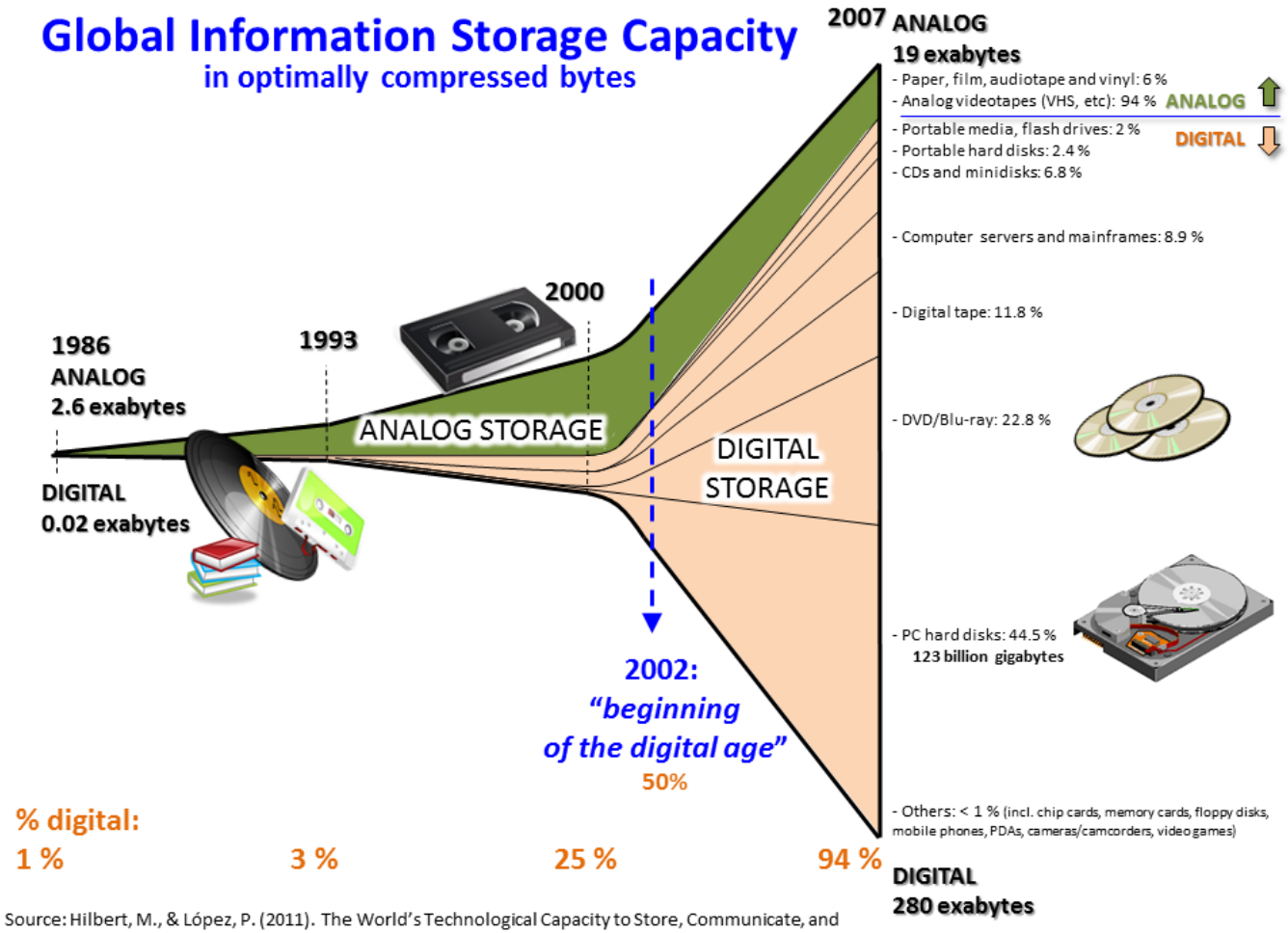
This was invented in the 1960s but was only commercially available in 1983.

Nathan Dumlao, CC0, via Unsplash
https://unsplash.com/photos/OHzkfrv9Ycw

Evolution of Data Storage

1920s-1950s		1960s-1970s		1980s-Mid 1990s		Mid 1990s-Today	
1920s Magnetic Tape		1960s Music Tape		1981 3.5" Floppy		1994 Zip Drive	
1930s Magnetic Drum		1960s DRAM		1984 CD Rom		1995 DVD	
1940s Williams Tube		1960s Twistor Memory		1989 Digital Data Storage		1995 Smart Media	
1940s Selectron Tube		1970s Bubble Memory		1990 Magneto Optical Disc		1997 Multimedia Card	
1940s Delay Line Memory		1971 8" Floppy		1992 Mini Disc		1999 Microdrive	
1950s Magnetic Core		1975 5.25" Floppy		1993 Digital Linear Tape		2003 xD-Picture Card	
1950s Hard Disk		1980s CD		1994 Compact Flash		Future Holographic Memory	

Evolution of Data Storage



Source: Hilbert, M., & López, P. (2011). The World's Technological Capacity to Store, Communicate, and Compute Information. *Science*, 332(6025), 60 –65. http://www.martinhilbert.net/WorldInfoCapacity.html

On-Disk Storage

On-disk storage utilizes low cost hard-disk drives for long-term storage.

Implemented via a distributed file system or a database.

Distributed File Systems

- Support schema-less data storage.
- DFS storage device provides out of box redundancy and high availability by copying data to multiple locations via replication.
- Simple and fast access data storage, non-relational in nature, fast read/write capability.
- Multiple smaller files are generally combined into a single file to enable optimum storage and processing.

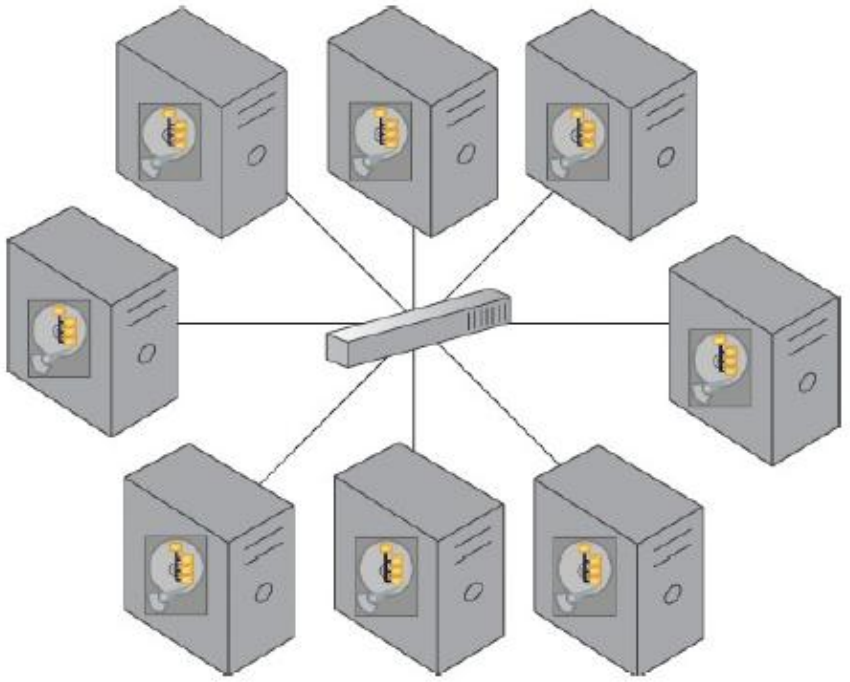
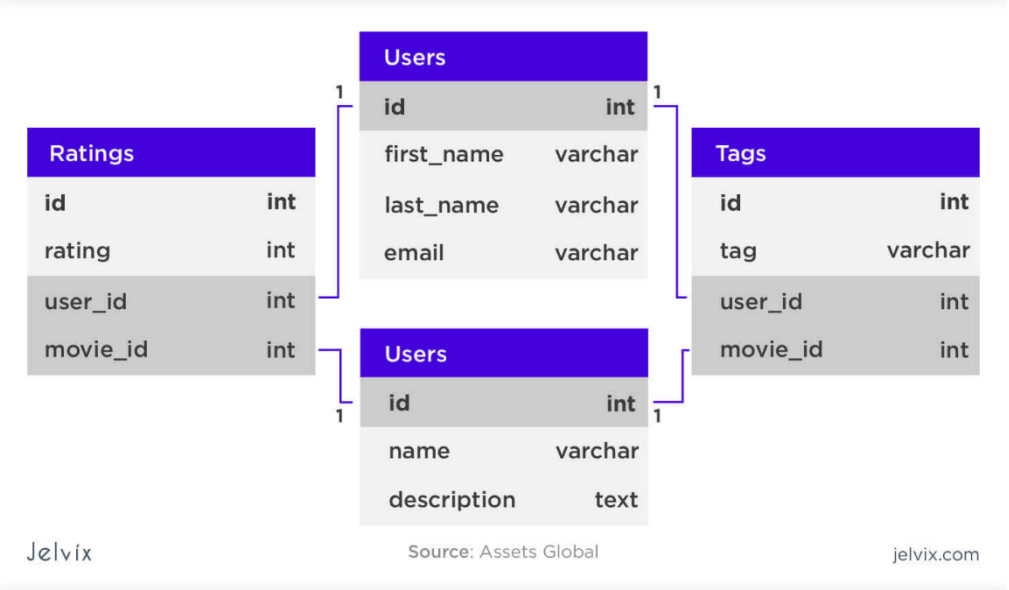


Figure 5.3 The symbol used to represent distributed file systems.

Relational DBMS

- ACID-compliant – restricted to a single node.
- Do not provide out-of-the-box redundancy and fault tolerance.
- Less ideal for long-term storage of data that accumulates over time.
- Manually sharded – complicates data processing when data from multiple shards is required.
- Schema-based, not suitable for semi- & un-structured data
- Data need to be checked against schema constraints – creates latency.

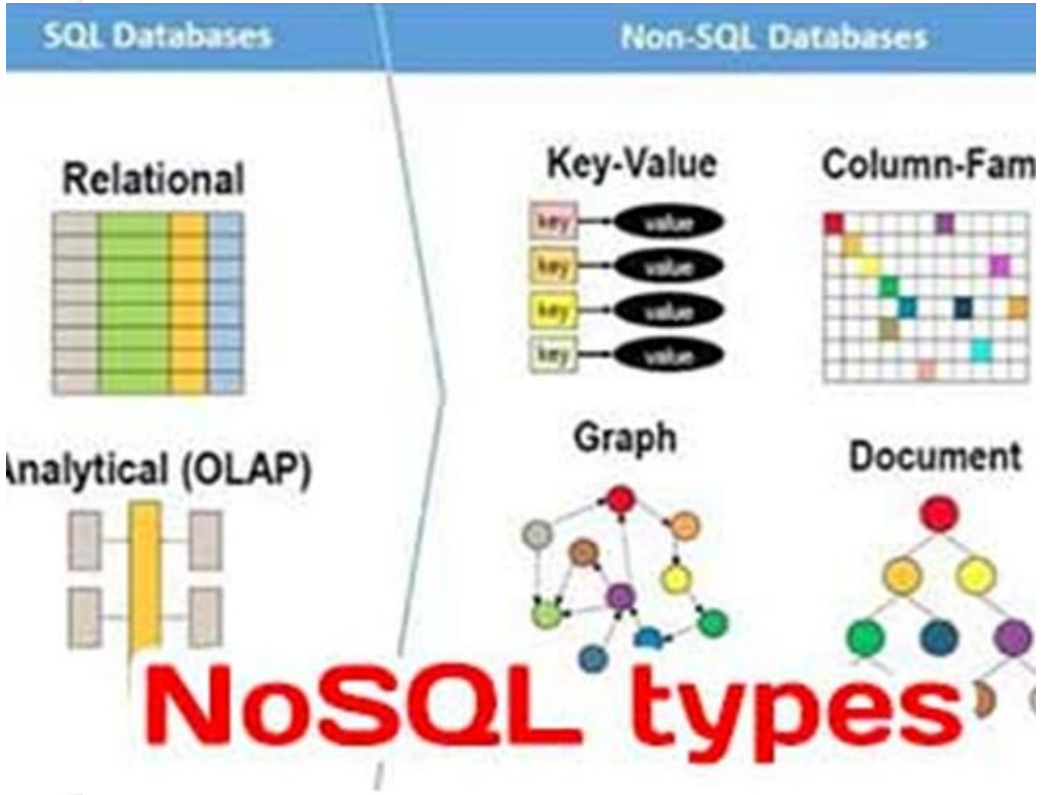


Relational DBMS

- Transaction management style that leverages **pessimistic concurrency controls** to ensure consistency is maintained through the application of record locks.
- **ACID** (atomicity, consistency, isolation, durability)
  - **Atomicity** ensures that all **operations will always succeed or fail completely**
  - **Consistency** ensures that the **database will always remain in a consistent state** by ensuring that only data that conforms to the constraints of the database schema.
  - **Isolation** ensures that the results of a **transaction are not visible to other operations** until it is complete.
  - **Durability** ensures that the **results of an operation are permanent**.

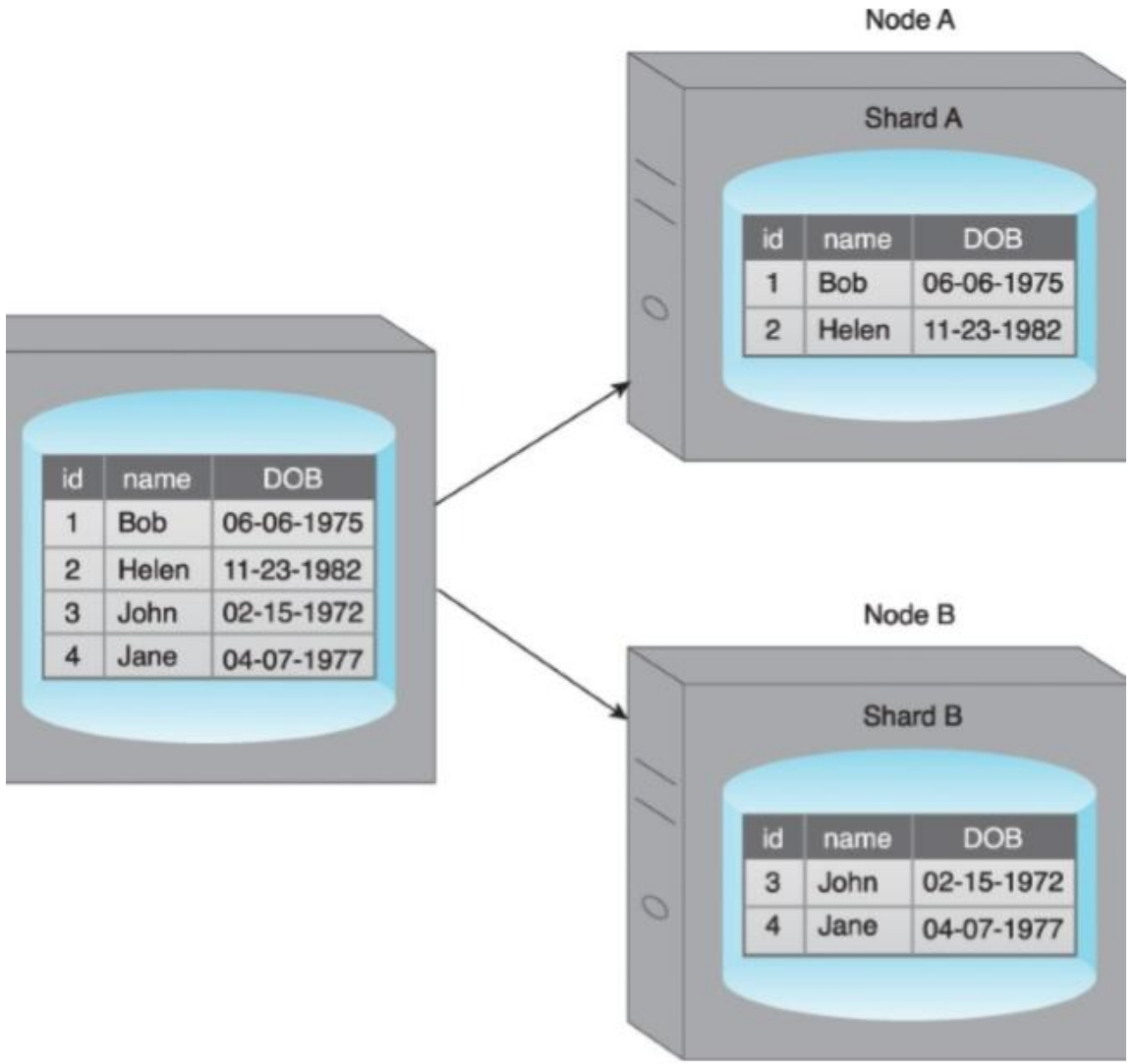
NoSQL Database

- NoSQL means Not Only SQL
- NoSQL database - non-relational database that is highly scalable, fault-tolerant and specifically designed to house semi-structured and unstructured data.
- A NoSQL database often provides an API-based query interface that can be called from within an application.



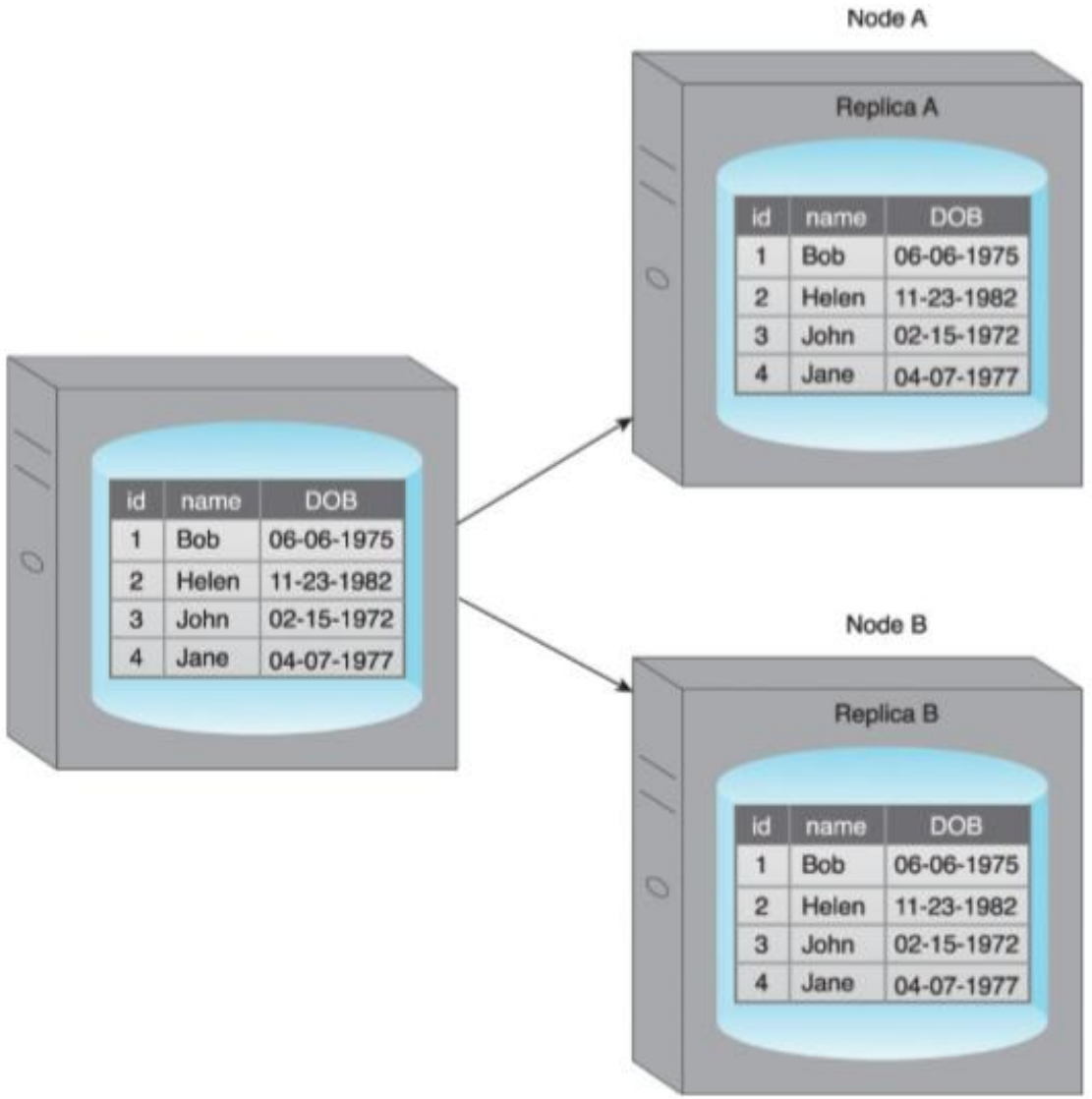
NoSQL Database: **Sharding**

- **Sharding**– process of horizontally partitioning a large dataset into a collection of smaller, more manageable datasets called shards.
- The shards are distributed across multiple nodes, where a node is a server or a machine.
- Each shard is stored on a separate node and each node is responsible for only the data stored on it.
- Each shard shares the same schema, and all shards collectively represent the complete dataset.



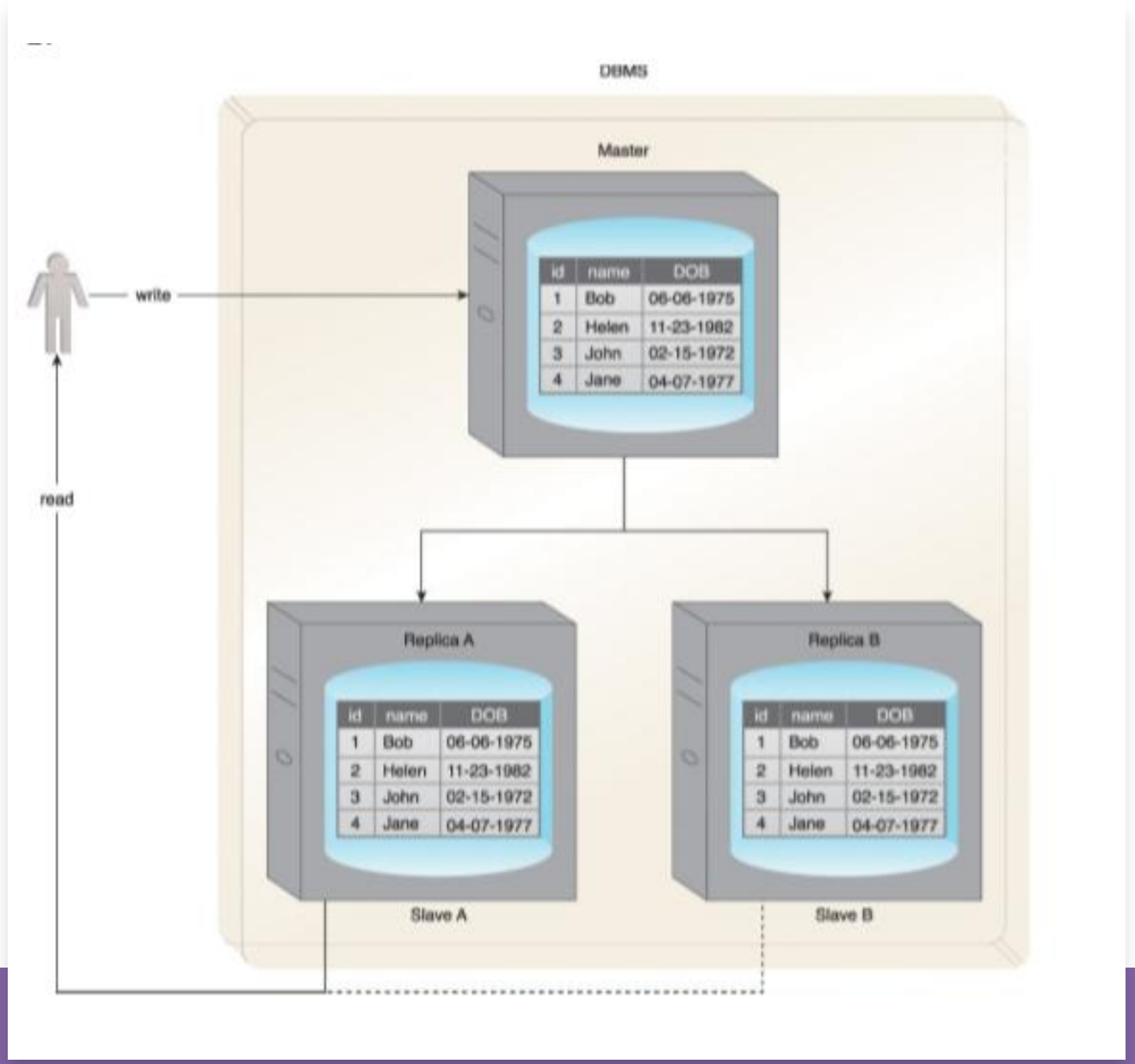
NoSQL Database: **Replication**

- **Replication** stores multiple copies of a dataset, replicas, on multiple nodes
- Provides scalability and availability due to the fact that the same data is replicated on various nodes.
- Fault tolerance is achieved since data redundancy ensures that data is not lost when an individual node fails.
- Two different methods used:
  - master-slave
  - peer-to-peer

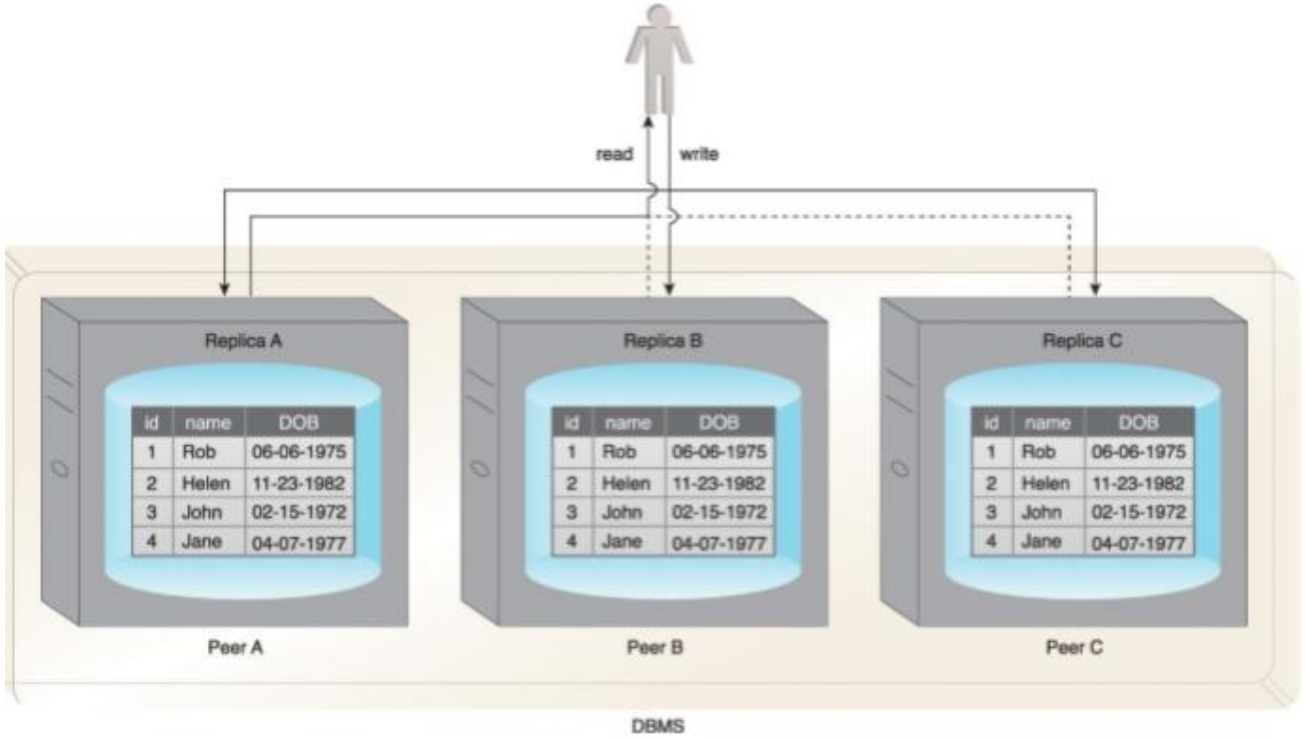


NoSQL Database: **Master-Slave Replication**

- During master-slave replication, nodes are arranged in a master-slave configuration, and all data is written to a master node.
- Once saved, the data is replicated over to multiple slave nodes.
- All external write requests, including **insert, update and delete,** occur on the **master node**.
- **Read requests** can be fulfilled by any **slave node.**
- Problem: read inconsistency
- Solution: voting system



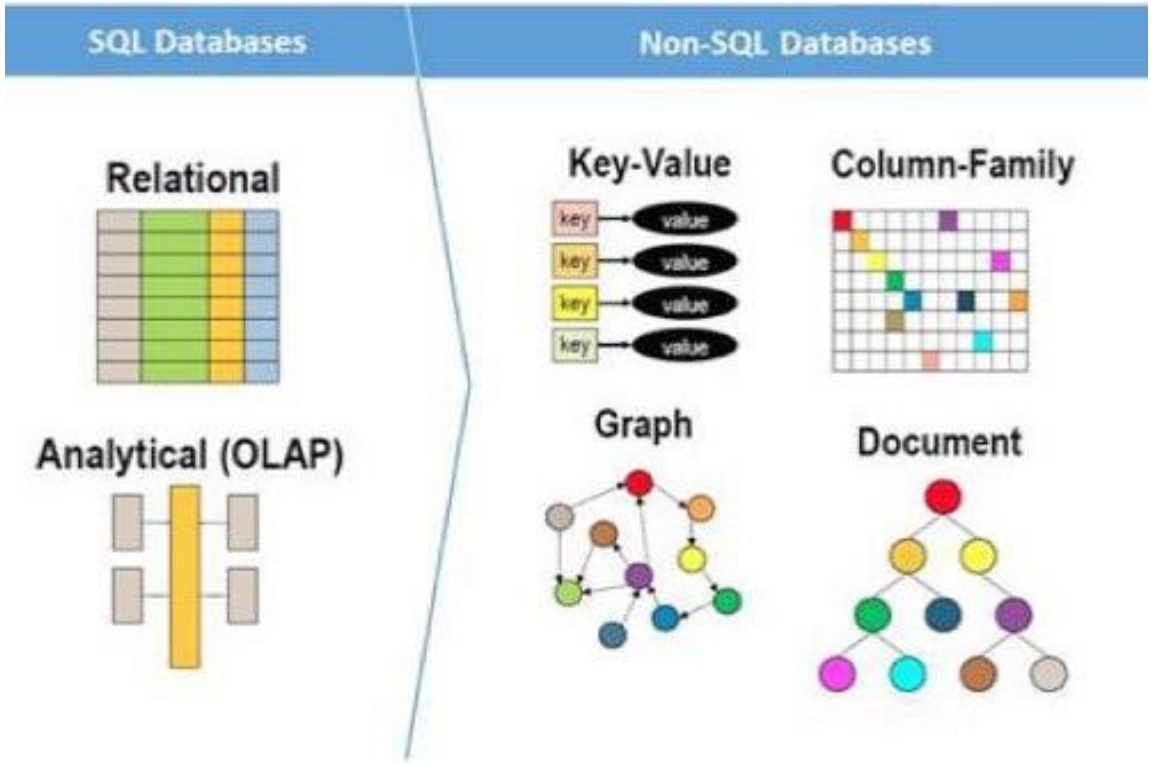
NoSQL Database **Peer-to-Peer Replication**



- All nodes operate at the same level.
- Each node, a peer, is equally capable of handling reads and writes.
- Each write is copied to all peers.
- Prone to write inconsistencies – simultaneous update

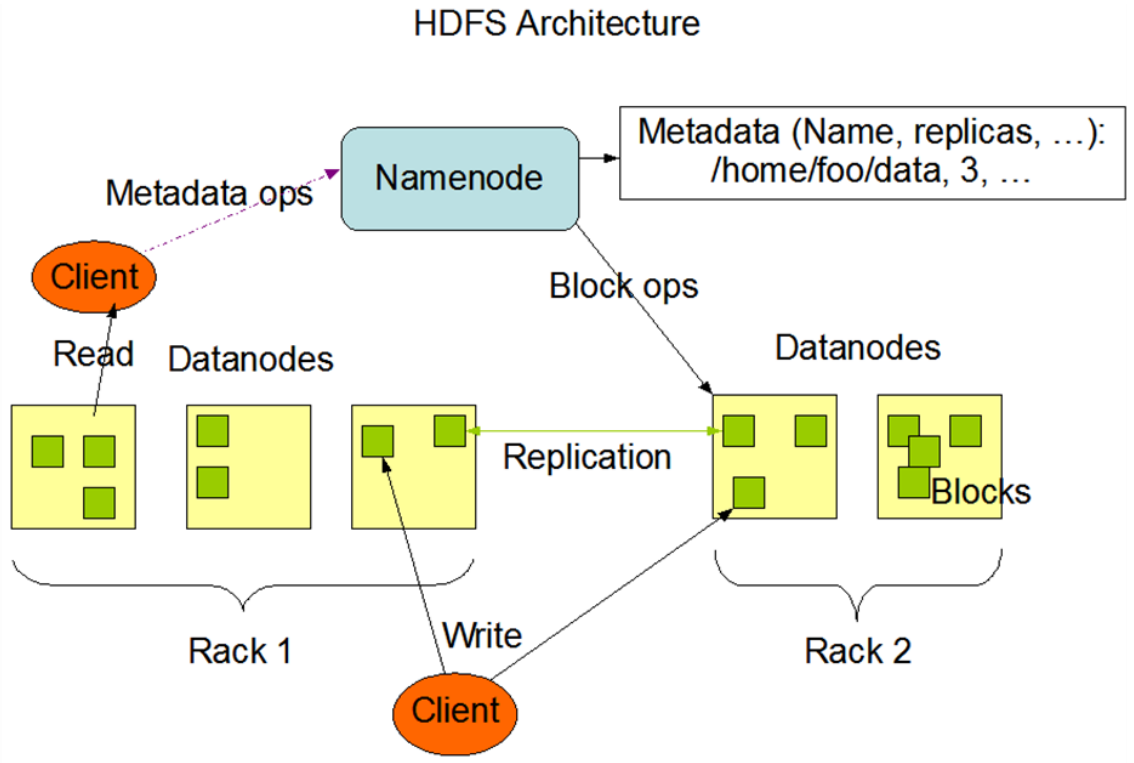
NoSQL vs SQL

SQL	NoSQL
1. Row Oriented	1. Column oriented
2. Fixed schema	2. Flexible schema, add column later
3. Not optimized for sparse matrix	3. Good with sparse table
4. Optimized for join operations	4. Join by using MR- not optimized
5. Not integrated	5. Tight integration with key-value system
6. Hard to shard and scale	6. Horizontal scalability
7. Only for structured data	7. Good for semi structured, unstructured and structured data



Hadoop Distributed File System (HDFS)

- HDFS
  - Is a versatile, resilient, clustered approach to managing files in a big data environment.
- HDFS is NOT the final destination for files.
- HDFS is a data service that offers a unique set of capabilities needed when data volumes and velocity are high.
- Because the data is written once and then read many times thereafter, rather than the constant read-writes of other file systems, HDFS is an excellent choice for supporting big data analysis.

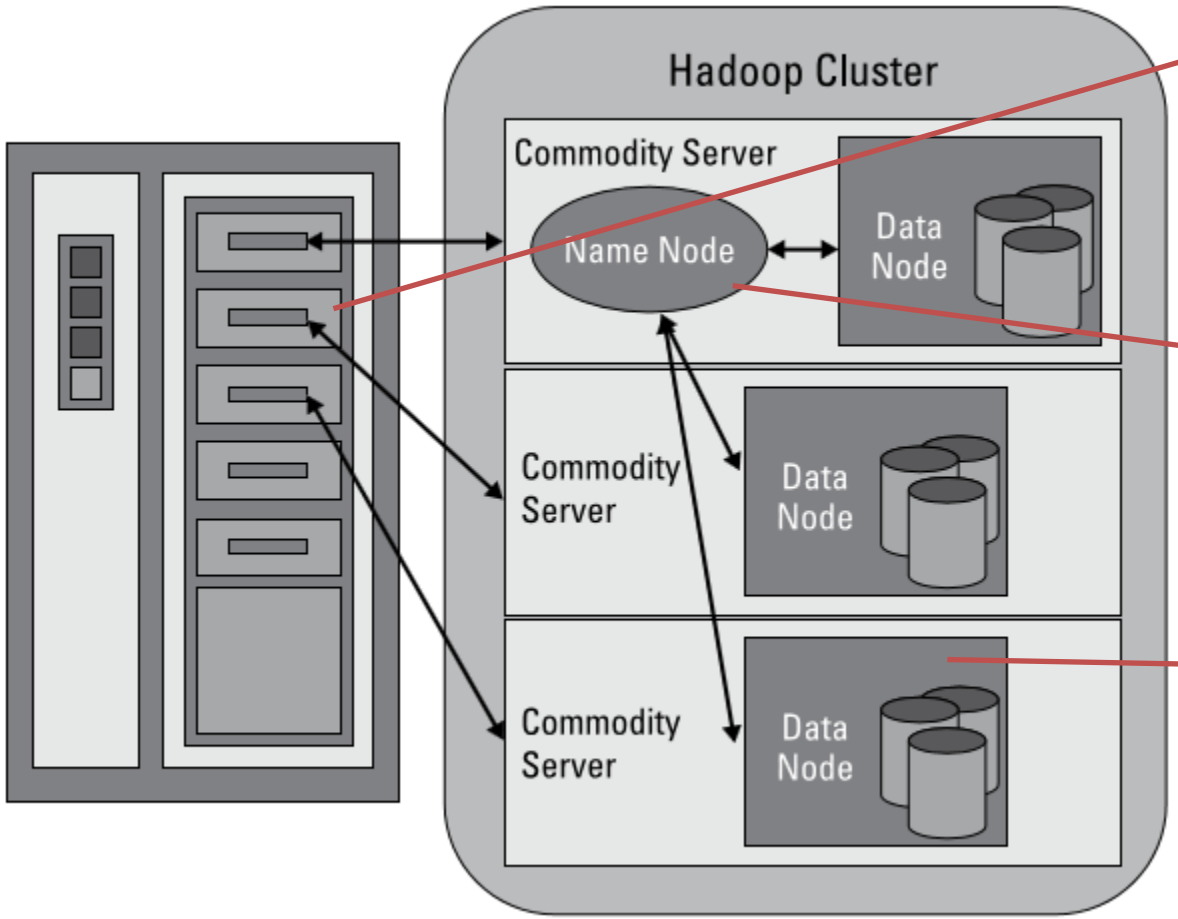


Hadoop Distributed File System

Motivations for developing HDFS - to deal with the following challenges:

- Hardware failure
- The need for streaming access
- Large datasets
- Data coherency issue
- Cheaper in computation
- heterogeneous platforms

Hadoop Distributed File System



HDFS works by breaking large files into smaller pieces called *blocks.*

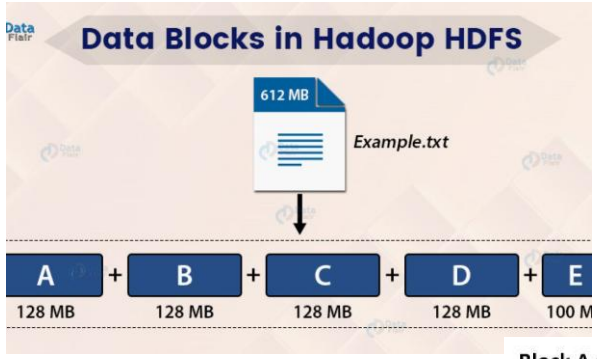
The NameNode also acts as a “traffic cop,” managing all access to the files.

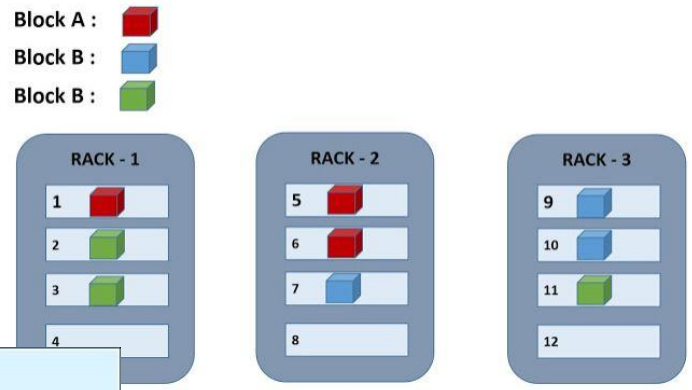
The blocks are stored on data nodes,

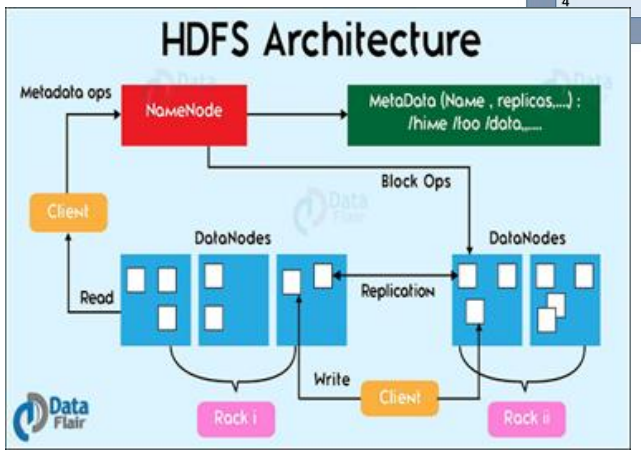
How a Hadoop cluster is mapped to hardware

Hadoop Distributed File System

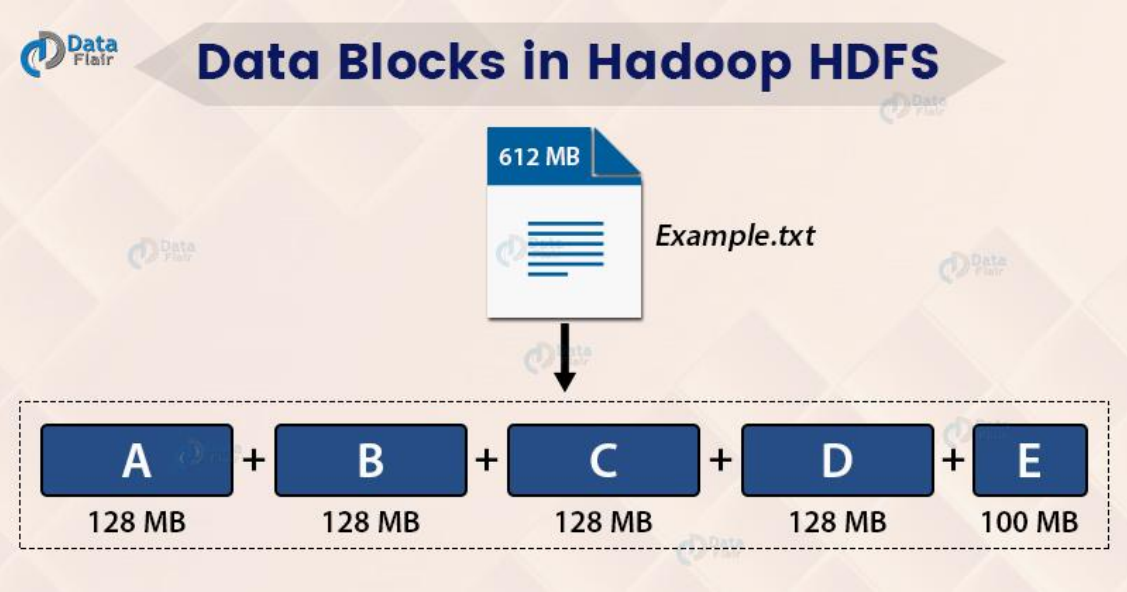
- **Files are broken into large blocks.**
  - – Typically, 128 MB block size
  - – Blocks are replicated for reliability
  - - One replica on local node,
    another replica on a remote rack,
    Third replica on local rack,
    Additional replicas are randomly placed
- **• Understands rack locality**
  - – Data placement exposed so that computation can be migrated to data
- **• Client talks to both NameNode and DataNodes**
  - – Data is not sent through the namenode, clients access data directly from DataNode
  - – Throughput of file system scales nearly linearly with the number of nodes.





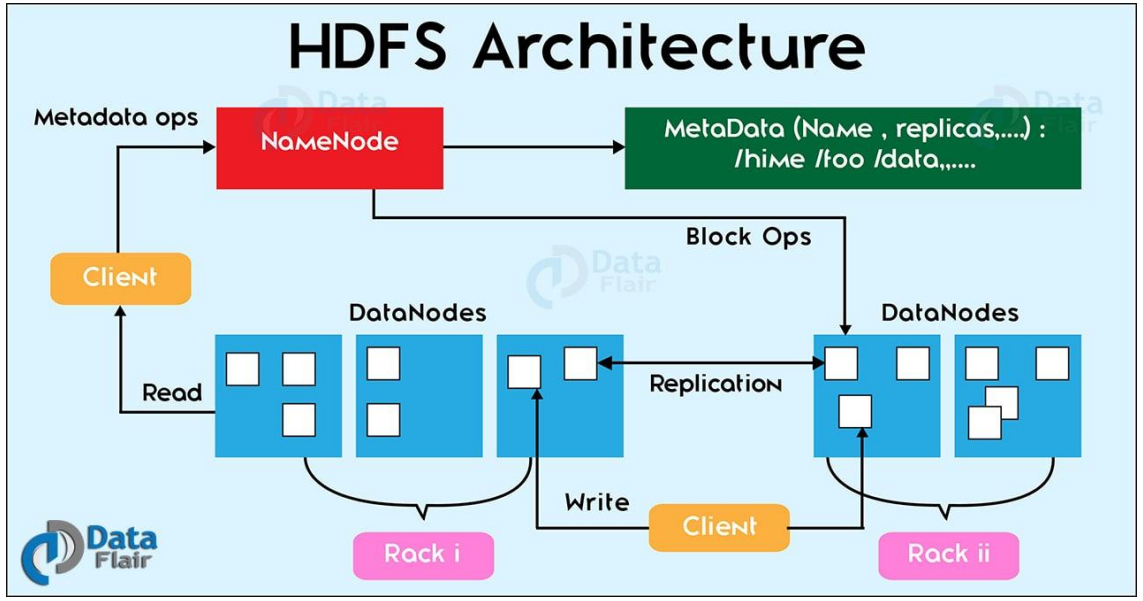


Data block in Hadoop HDFS



- Internally, HDFS split the file into block-sized chunks called a block. The size of the block is **128 Mb** by default. One can configure the block size as per the requirement.
- For example, if there is a file of size 612 Mb, then HDFS will create four blocks of size 128 Mb and one block of size 100 Mb.

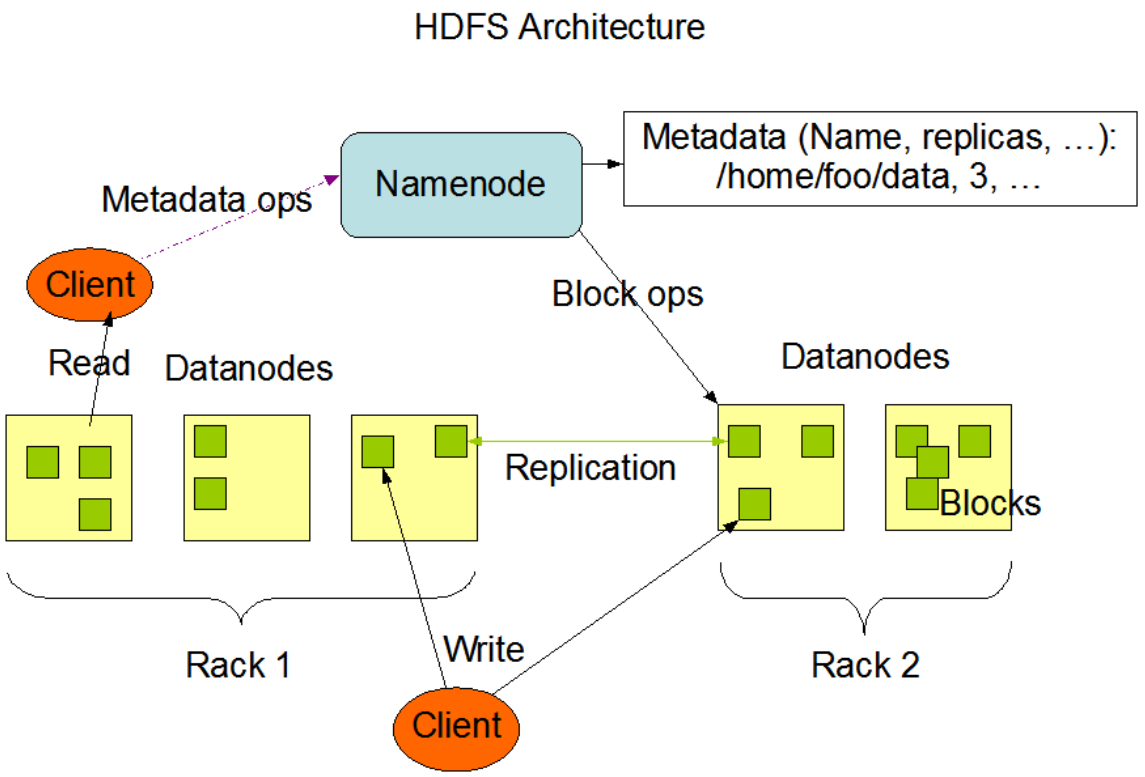
HDFS Architecture



- Hadoop Distributed File System follows the **master-slave architecture**. Each cluster comprises a **single master node** and **multiple slave nodes**.
- Internally, the files get divided into one or more **blocks**, and each block is stored on different slave machines depending on the **replication factor**
- The master node stores and manages the file system namespace, that is information about blocks of files like block locations, permissions, etc. The slave nodes store data blocks of files.

https://data-flair.training/blogs/hadoop-hdfs-architecture/

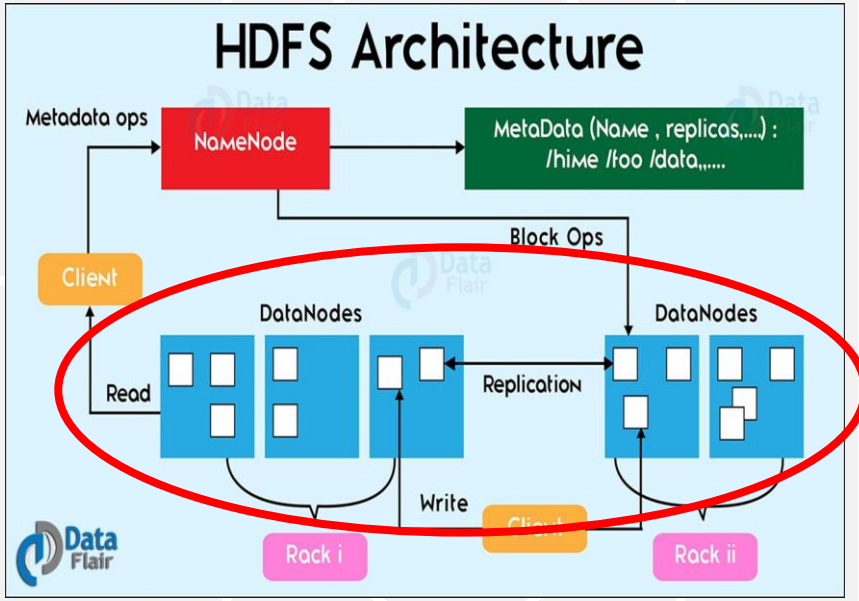
HDFS NameNodes



- NameNode is the centerpiece of the Hadoop Distributed File System. It **maintains and manages the file system namespace** and **provides the right access permission** to the clients.
- HDFS works by breaking large files into smaller pieces called blocks.
- The blocks are stored on data nodes.
- It is the responsibility of the NameNode to know what blocks on which data nodes make up the complete file.
- The NameNode manages all access to the files, including reads, writes, creates, deletes, and replication of data blocks on the data nodes.

HDFS DataNode

- DataNodes are the slave nodes in Hadoop HDFS. They store blocks of a file.
- Data nodes are not smart, but they are resilient.
- Within the HDFS cluster, data blocks are replicated across multiple data nodes and access is managed by the NameNode.
- The replication mechanism is designed for optimal efficiency when all the nodes of the cluster are collected into a rack.
- In fact, the NameNode uses a “rack ID” to keep track of the data nodes in the cluster.



Hadoop Distributed File System

HDFS supports a number of capabilities designed to provide data integrity - when files are broken into blocks and then distributed across different servers in the cluster, any variation in the operation of any element could affect data integrity.

HDFS uses transaction logs and checksum validation.

Transaction logs are a very common practice in file system and database design.

They keep track of every operation and are effective in auditing or rebuilding of the file system should something untoward occur.

Checksum validations are used to guarantee the contents of files in HDFS.

When a client requests a file, it can verify the contents by examining its checksum. If the checksum matches, the file operation can continue. If not, an error is reported. Checksum files are hidden to help avoid tampering.

HDFS Key Features

Rack awareness

High availability

data blocks

replication management

data read and write operations

HDFS Key Features: Rack Awareness

Key Idea:

- NameNode has the capability to access rack information (rack awareness) from metadata for deciding where to locate data blocks on the DataNodes.

Purpose:

- to achieve fault-tolerance and to minimize latency (time taken to read/write data)

How to achieve fault-tolerance?

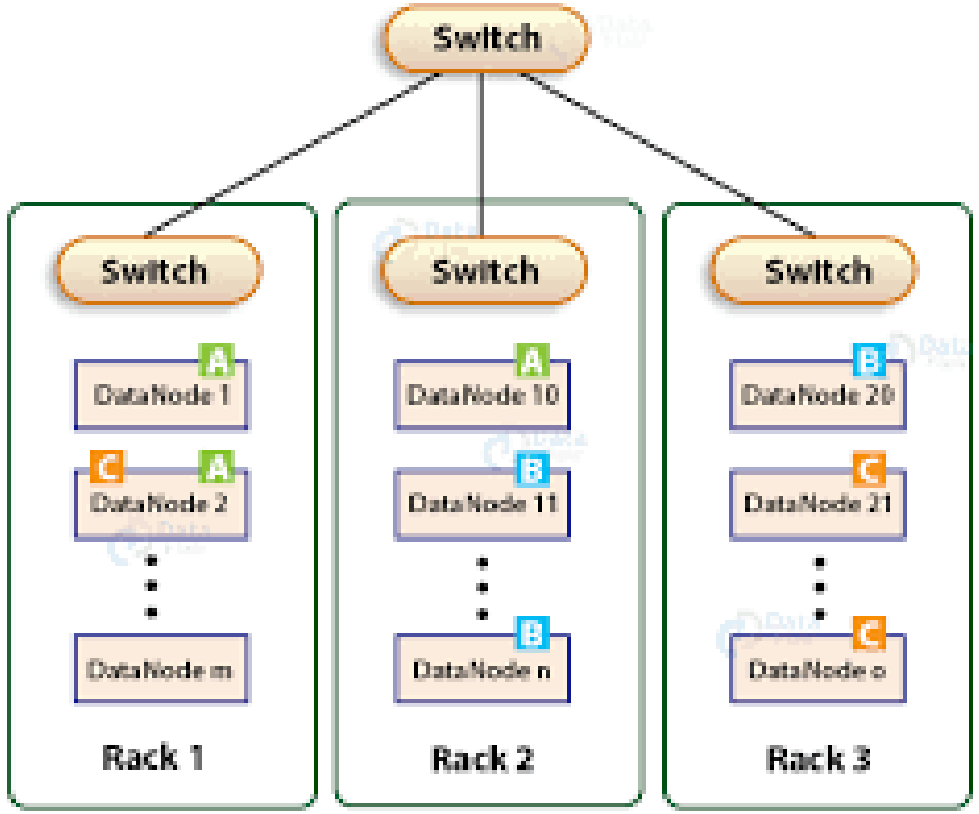
- NameNode perform a specific replication policy (the default is 3 replication factor)
  - the first block replica is stored on the same node.
  - the second block replica is stored on a different rack.
  - the third block replica is located on different node of the same rack.

How to minimize latency?

- When choosing different rack, NameNode chooses the closest one.

Benefits:

- Increase data availability and reliability via fault tolerance
- Improve network bandwidth via closet replication strategy



HDFS Key Features: High availability

- With Hadoop 2.0, we have support for multiple NameNodes and with Hadoop 3.0 we have standby nodes.
- This overcomes the SPOF (Single Point Of Failure) issue using an extra NameNode (Passive Standby NameNode) for automatic **failover**.
- This is the high availability in Hadoop.
- **Failover** is a process in which the system transfers control to a secondary system in an event of failure.
- There are two types of failover:
  - **Graceful Failover –** In this type of failover the administrator manually initiates it.
  - **Automatic Failover –** In **Automatic Failover**, the system automatically transfers the control to standby NameNode without manual intervention