Πανεπιστήμιο Πατρών

Προχωρημένα Θέματα σε Κατανεμημένα Συστήματα

GFS / HDFS

Google File System (GFS)

Overview

- Main functionalities of a distributed filesystem
  - Naming
  - Load Distribution
  - Persistent Storage
  - File operations
- Should provide:
  - Consistency
  - Reliability
  - Availability
  - Scalability
  - Security
  - Transparency
- In the Cloud
  - Storage and management of Big Data!
  - Data produced and consumed at highly diverse geographic locations
  - Hardware failures are very frequent
  - Files are of enormous sizes

GFS: Google File System

- Google's **scalable**, **distributed** filesystem for **large distributed data-intensive applications**.
- It provides **fault tolerance** while running on **inexpensive commodity hardware**.
- It delivers **high aggregate performance** to a **large number of clients**

Design Assumptions

- GFS has been designed with the following assumptions in mind:
  - High frequency of **failures**
  - **Files are huge**, typically multi-GB
  - Two types of reads:
    - **Large streaming reads**
    - **Small random reads**
  - Once written, files are seldom modified
    - Modifications are mostly done by **appending**
    - Random writes are supported, but inefficient

- Targeted towards data analytics
  - Typically huge files
  - **High sustained bandwidth** more important than low latency

Filesystem Interface

- **create/delete**
- **open/close**
- **read/write**
- **Snapshot**
  - Creates a copy of a file or directory almost instantaneously, masking current mutations that may be going on.
- **Record Append**
  - Append some data (a “record”) at the end of a file
  - Many record appends may be taking place concurrently
  - There is no guarantee about the offset where each record will be written
  - The only guarantee is that each record will be written at a contiguous part of the file
  - The actual offset where the record was finally placed is returned at the end of the operation

Architecture

- Three types of nodes:
  - One **master** node
  - Multiple **chunkservers**
  - Multiple **clients**

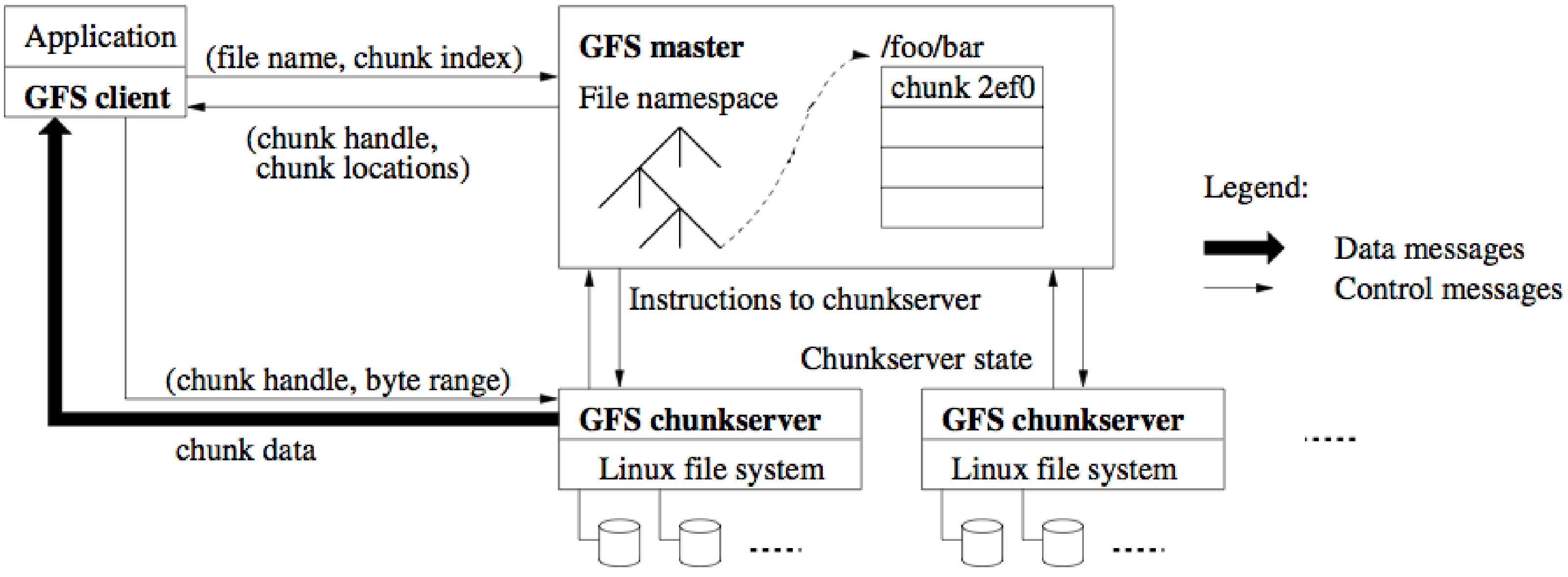


Figure 1: GFS Architecture

Architecture

- Chunks
  - **Fixed size** -- **64MB**
    - Smaller file system structures
    - Lower network load
  - Each chunk has a **unique 64-bit ID**
  - Chunks divided in **1024 blocks** of **64KB** each
    - Each block has a **32-bit checksum**
- No caching below the filesystem interface
  - Most operations **read data once**!

Master Node

- The master nodes maintains:
  - The namespace
  - Access control information
  - Mapping: **file → chunk IDs**
  - Mapping: **chunk ID → chunk location**

- Periodically the master sends **heartbeats** to each chunkserver to receive updates on its state.

- The master also
  - is in charge of **chunk placement** and **chunk replication**
  - keeps track of **chunk leases**

Consistency Model

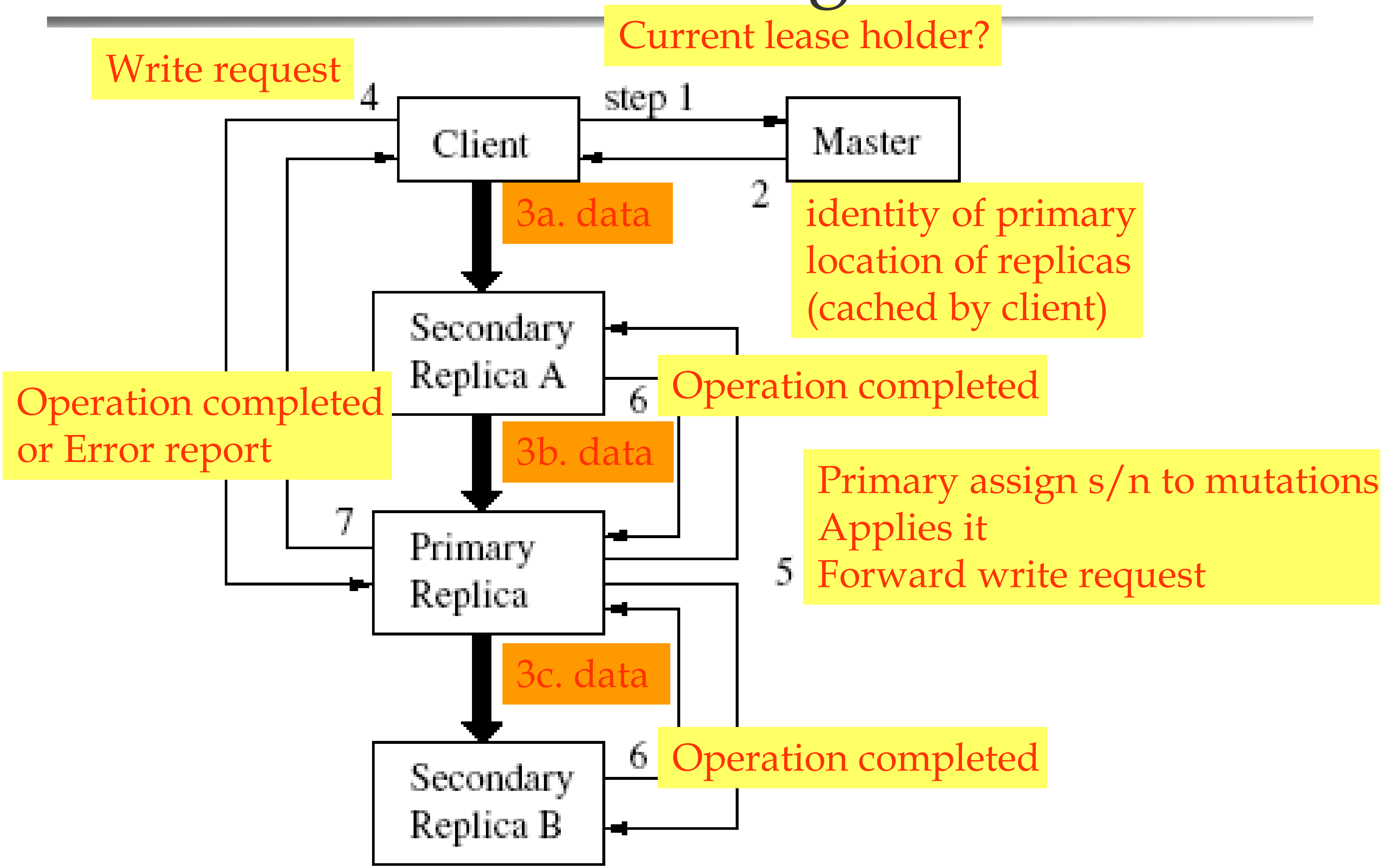
A file can be in one of the following states:

- **Consistent**
  - Data is consistent across all replicas
- **Defined**
  - Data is consistent across all replicas
  - The latest mutation is complete (and available to clients)
- **Inconsistent**
  - When a mutation has failed

Leases and Mutations

- **Mutation** = an update of a file
- Mutations (updates) are applied on all replicas **in the same order**
- One of the replicas of a chunk is given a **lease**, appointing it as the **primary replica**.
- The primary replica decides on the order of the mutations.
- The client copies data to all replicas (including the primary one) in a pipeline route, respecting the relative proximity (for efficiency).

Data flow during a write



Locking

- Read lock
  - Prevents a file/directory from being deleted, renamed, or snapshotted
- Write lock
  - Prevents a file from being modified by someone else

Replica Placement

- The master node is responsible for replica placement
- Replicate data for
  - Reliability / availability
  - Scalability (maximize network bandwidth utilization in reads)
- Number of replicas
  - By default 3
  - User may ask for more

Load balancing

- As we said, the master node is responsible for replica placement
- Allocates a new chunk on a chunkserver whose utilization is below average

Garbage collection

- Mechanism similar to a Recycle Bin.
- A deleted file is renamed to a hidden filename
- Periodically, hidden names are considered for purging (permanent deletion).
  - Purged only after 3 days have passed since deletion.

Hadoop Distributed File System (HDFS)

Hadoop HDFS

- The primary storage FS used by Hadoop applications
- Splits up files in **blocks** (similar to GFS chunks)
- Strong points:
  - Block replication
  - Locality of data access
- Started as an open source implementation of GFS.

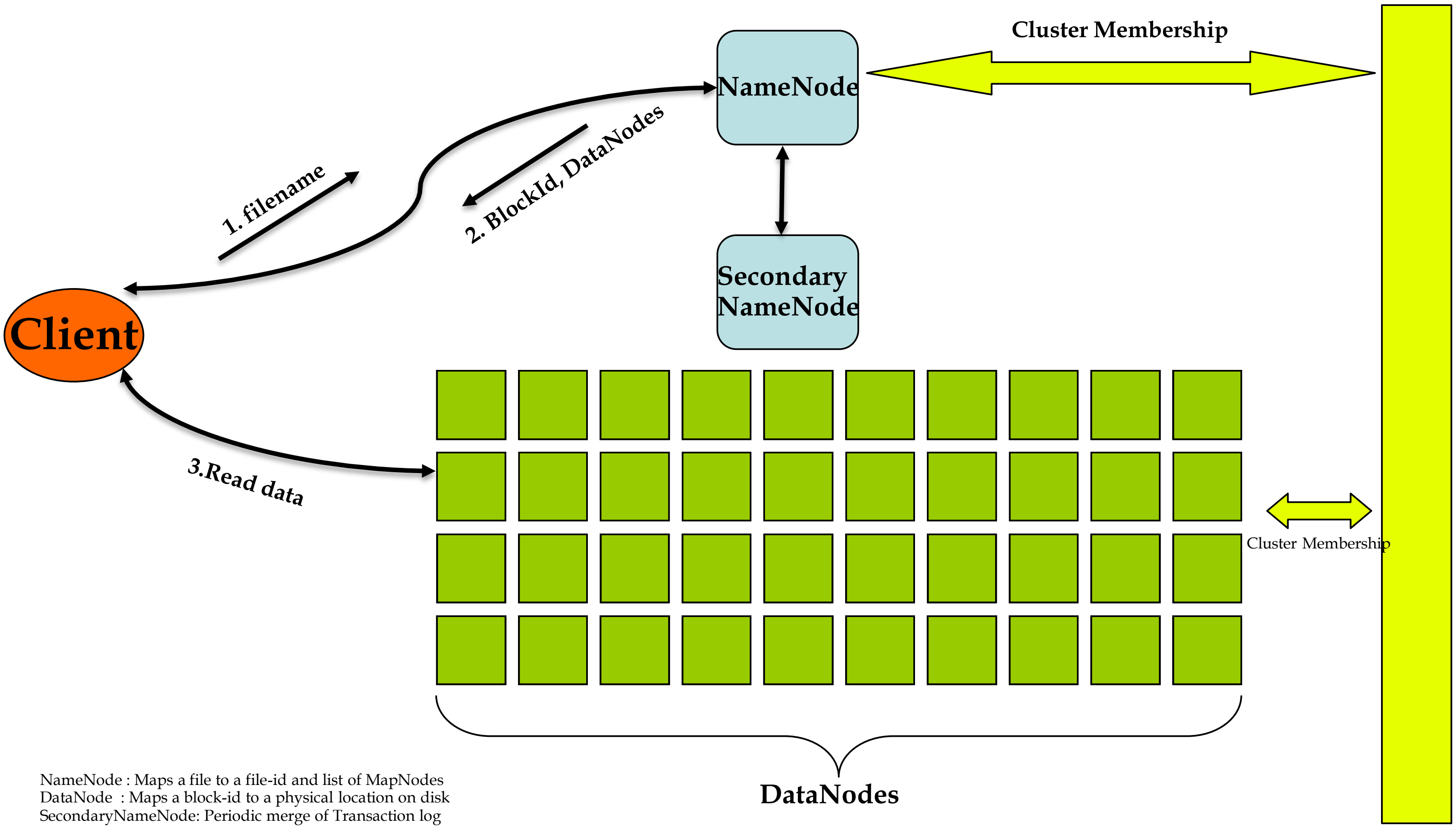
HDFS advantages

- Very large scale storage
  - 10,000 nodes
  - 100,000,000 files
  - 10PB storage space
- Based on inexpensive commodity hardware
  - Replication used to address failures
  - Fault detection
- Optimized for batch processing
  - Data location is visible to the application
  - Computation can be transferred where the data is
  - Very high aggregate bandwidth
- Storage may rely on heterogeneous underlying operating systems

HDFS principles

- Namespace is universal for the whole cluster
- Takes care of consistency
  - Write-once-read-many
  - Append is the only supported write operation
- Files are split into blocks
  - Fixed size – 128 MB
  - Each block replicated on multiple DataNodes
- Based on smart clients
  - Clients know the location of blocks
  - Clients access data directly from Data Nodes

HDFS architecture



NameNode : Maps a file to a file-id and list of MapNodes
DataNode : Maps a block-id to a physical location on disk
SecondaryNameNode: Periodic merge of Transaction log

NameNode - DataNode

NameNode

- Metadata in RAM
  - No paging!
- Metadata types
  - List of files
  - Mapping: file → blocks
  - Mapping: block → DataNode
  - File attributes
- Logging
  - File creations, deletions, etc.

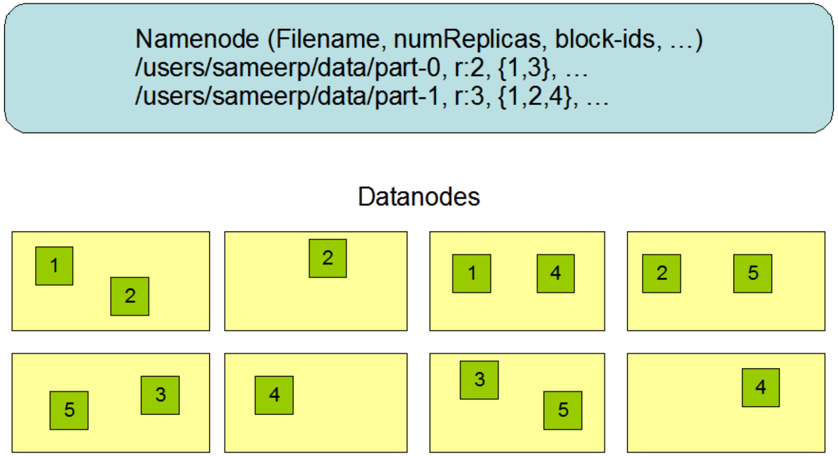
DataNode

- Stores blocks
  - Data + CRC stored on local FS (e.g., EXT3)
  - Serves data to clients
- Block Reports
  - Periodically sends a report on blocks' state to the NameNode
- Helps with pipelining data
  - For transferring data to other nodes

Write Data Pipelining

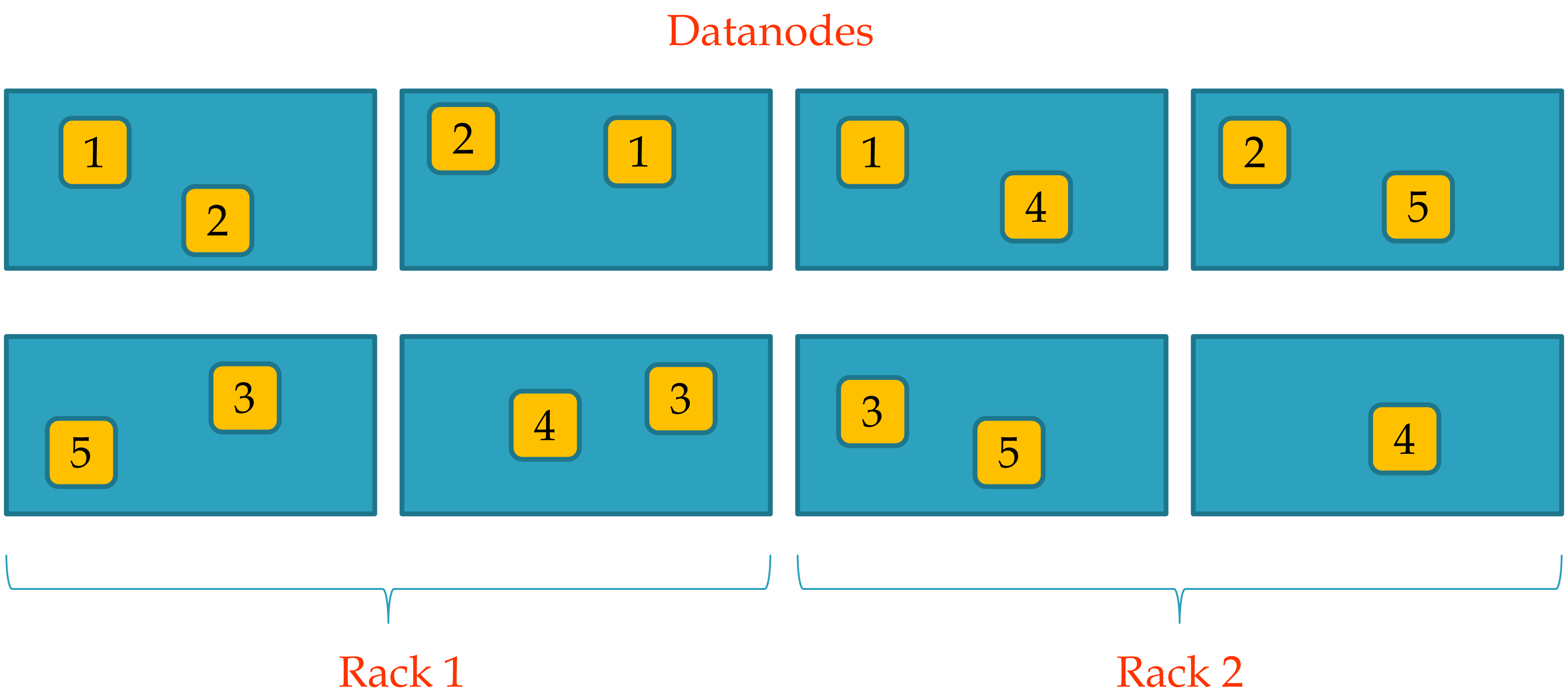
- The Client receives a list of Data Nodes that will serve as replicas of a given block
- Client streams data to the first DataNode
- The first DataNode streams data to the next DataNode, etc.

Block Replication Policy



- Replication Policy
  - A replica on the local node
  - Another replica on another node of the same rack
  - A third replica on a node of a different rack
  - Extra replicas on random other nodes
- Clients read from the closest replica

Replication



Checksums

- Use of CRC32 during file creation
  - CRC32 per 512 bytes
  - DataNodes store the checksums
- When reading data
  - If the client detects a checksum error, it tries a different replica DataNode

NameNode failures

- A single point of failure
- Operation log is stored on different local filesystems
- In case of failure, a new NameNode may be started to continue where the failed NameNode left off.