Hadoop Distributed File System

File System

- A subsystem of the OS that performs file management activities such as organization, sorting, retrieval, naming, sharing and protection of files
- Frees the programmer from concerns about the details of space allocation and layout of the secondary storage devices

Distributed File System

- Method of storing and accessing files in a client-server architecture
- Data is stored on the servers
- Accessed by clients with proper authorization rights

Distributed File System - Requirements

- **Transparency**
  - *Access transparency* – local and remote files are accessed in the same way
  - *Location transparency* – files can be relocated without affecting their path names
  - *Performance transparency* – client programs should not be affected when the load of server varies within a specified range
  - *Scaling transparency* – service can be expanded easily

Distributed File System - Requirements

- **Concurrent file updates**
  - Changes of file made by one client program should not interfere with the operation of other programs accessing the same file
- **File Replication**
  - Several copies of the same file are stored at different locations

Distributed File System - Requirements

- **Management of Hardware and OS Heterogeneity**
  - Service interface must abstract the heterogeneity of the underlying hardware and OS
- **Fault Tolerance**
  - Must handle failure of nodes and communication links

Distributed File System - Requirements

- **Consistency**
  - Different copies of the same file must have consistent data
- **Security**
  - Client requests need to be authenticated
- **Efficiency**
  - Services offered must be at least as powerful as a conventional file system

Examples of DFS

- Google File System (GFS)
- Hadoop Distributed File System (HDFS)
- Windows Distributed File System
- XtreemFS

Google File System

- Proprietary distributed file system developed by Google for its own use
- Provides efficient and reliable access to data using large clusters of commodity hardware
- Not for sale, but serves as a model for file systems for organizations with similar needs

GFS - Architecture

- Consists of a single **master** and multiple **chunk servers**
- Accessed by multiple **clients**
- Files are divided into fixed-size (64 MB) **chunks** (each chunk identified by a unique 64 bit **chunk handle**)

GFS - Architecture

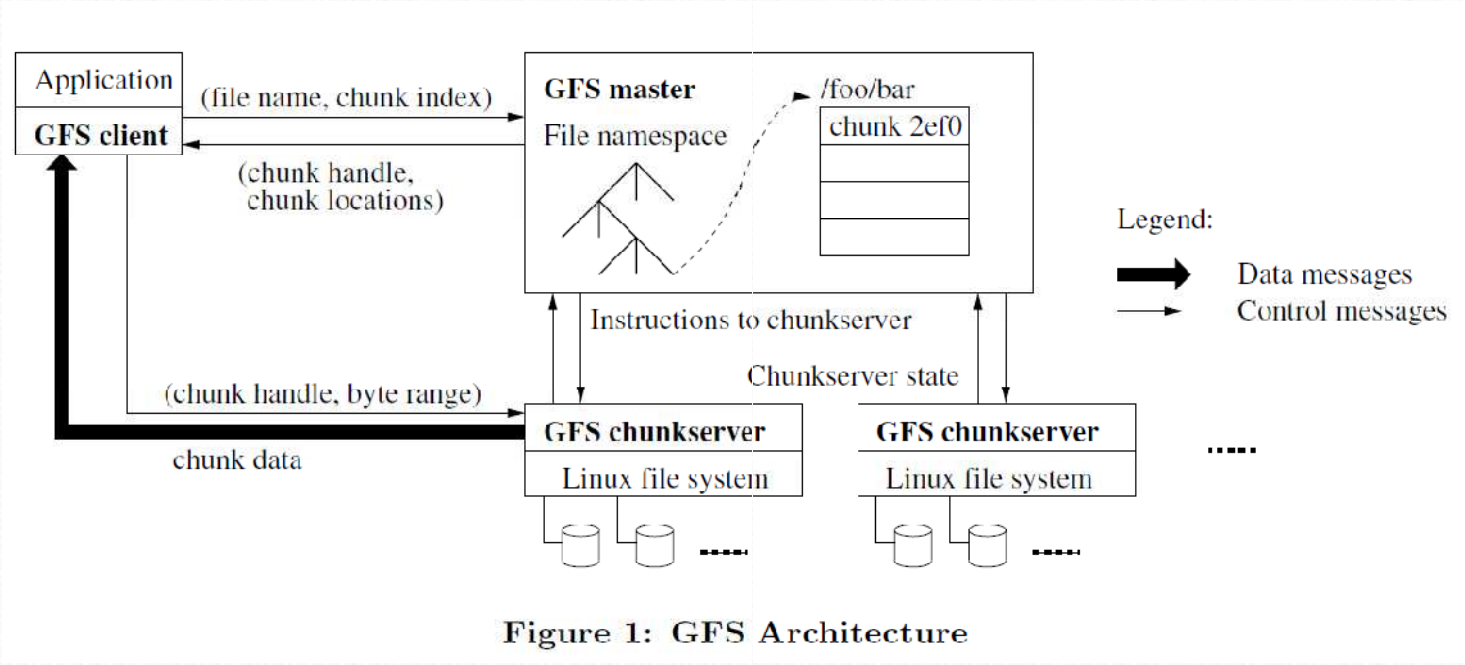


Figure 1: GFS Architecture

Hadoop Distributed File System

- Implemented for running MapReduce applications
- Differences from other distributed file sytems
  - Highly fault-tolerant
  - Designed to run on commodity hardware
  - High throughput access to application data
  - Suitable for large data sets

HDFS - Architecture

- **Master-slave** architecture
- Single **NameNode**
- Multipile **DataNodes**, usually one per node in a cluster
- File is split into one or more blocks and set of blocks are stored in DataNodes

HDFS - Architecture

- **File System Namespace**
  - Hierarchical file organization
  - User can create directories and store files in them
  - User can create, remove, move files from one directory to another, rename a file

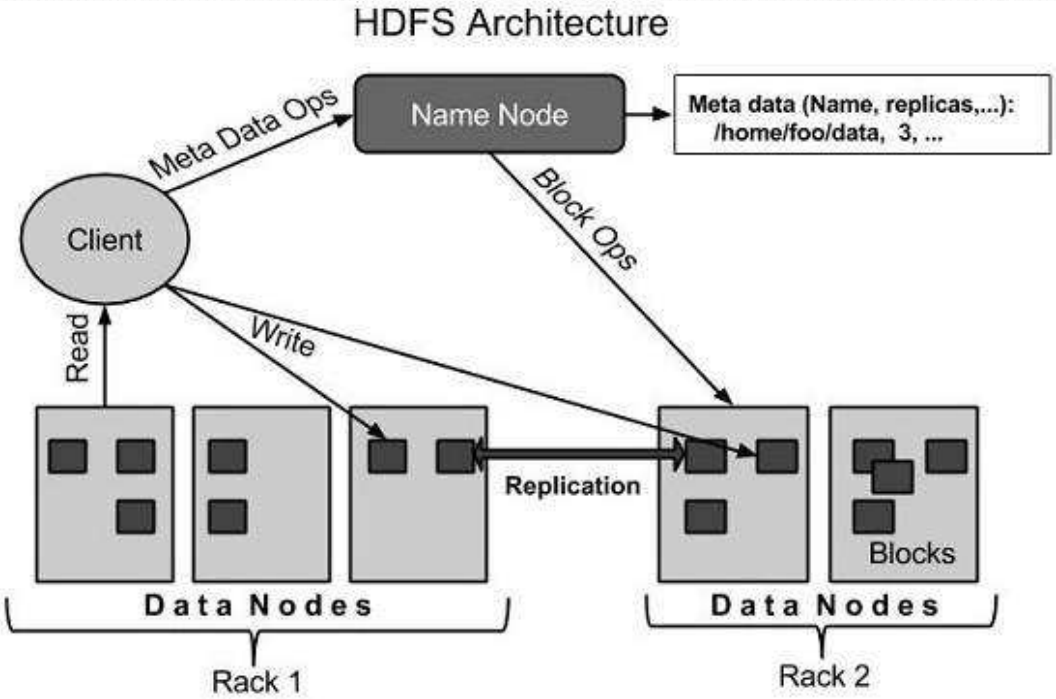
HDFS - Architecture

- **NameNode**
  - Manages the file system namespace. Any changes to file system namespace or its properties is recorded by NameNode
  - Contains meta-data
  - Regulates client's access to files
  - Executes file system operations such as renaming, closing, opening of files and directories
  - Stores replication factor as specified by application

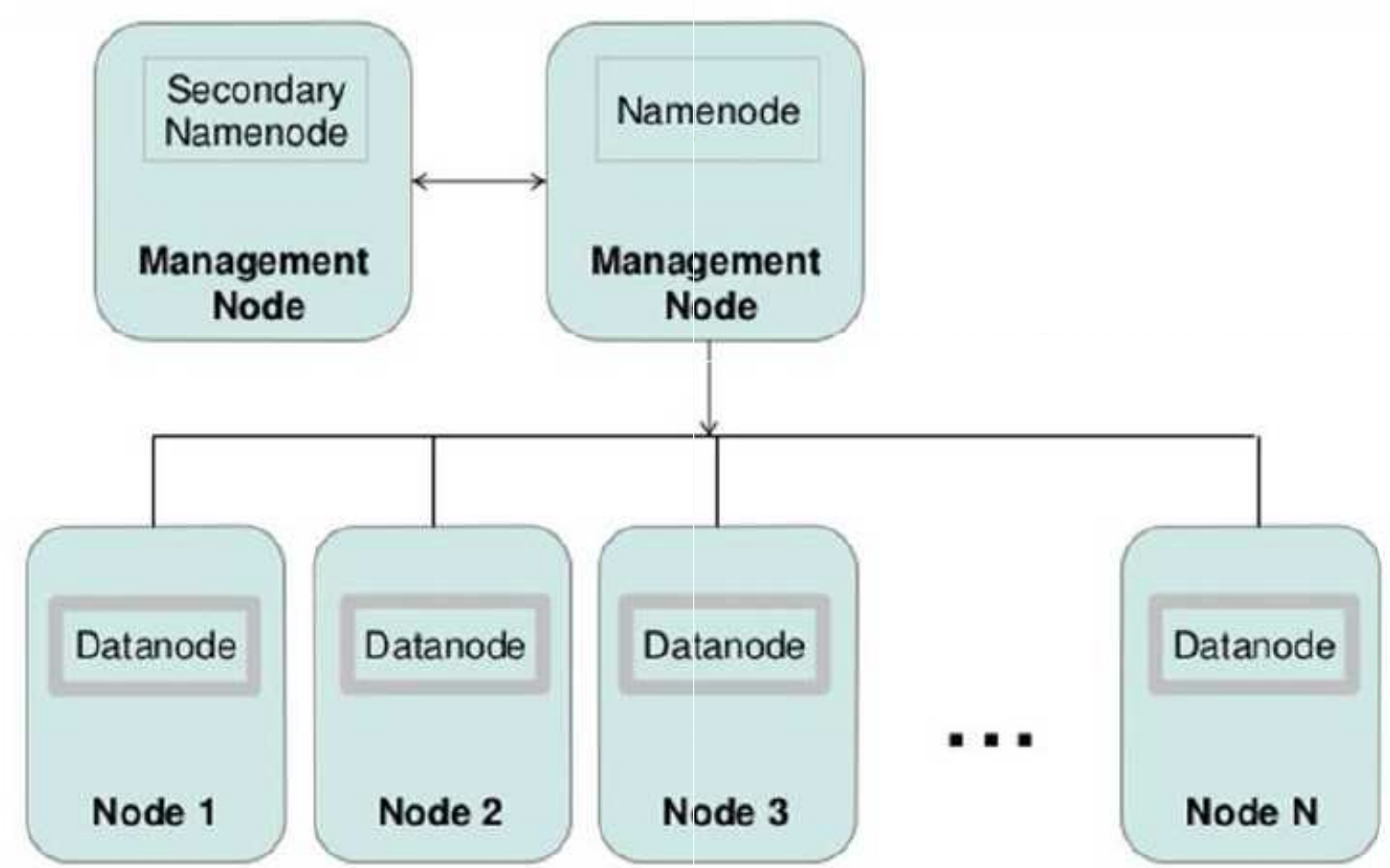
HDFS - Architecture

- **DataNode**
  - Perform read-write operations on the file systems
  - Perform block creation, deletion, and replication according to instructions of NameNode

HDFS - Architecture



HDFS Environment



Fault Tolerance

- HDFS assumes that failure is the norm rather than an exception
- Ways to fulfil reliability requirements
  - Block Replication
  - Replica placement
  - Heartbeat and Block report messages

Fault Tolerance

- **Block Replication**
  - Each block is replicated and distributed across the whole cluster
  - Replication factor – specified by the user (usually three)
- **Replica Placement**
  - DataNodes organized into “racks” –
  - Communication within a rack more efficient than communication between racks

Fault Tolerance

- **Replica Placement**
  - Optimizing replica placement distinguishes HDFS from other distributed file systems
  - **Three copies**
    - One replica in same node as original data
    - One replica in a different node but in the same rack
    - One replica in a different node in a different rack

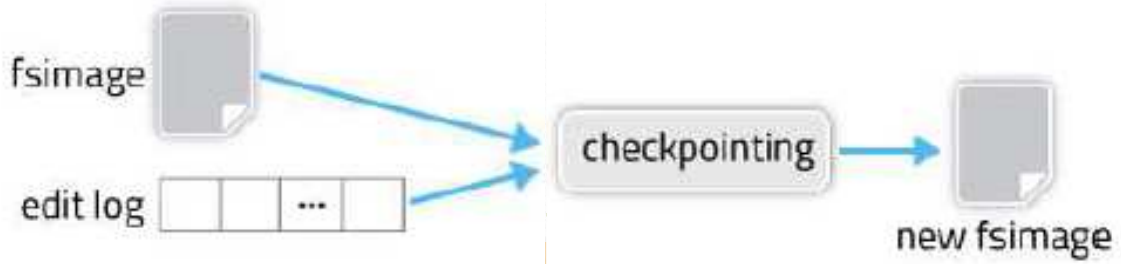
Fault Tolerance

- **Heartbeat and Block report messages**
  - Heartbeats and Block reports are periodic messages sent to the NameNode by each DataNode every 3 seconds
  - Receipt of heartbeat implies that DataNode is functioning properly
  - Block report contains a list of all blocks in a DataNode
  - NameNode receives such messages because it is the sole decision maker of all replicas in the system

Fault Tolerance

Secondary Name node

- Copies FsImage and Transaction Log from Name node to a temporary directory
- Merges FSImage and Transaction Log into a new FSImage in temporary directory
- Uploads new FSImage to the Name node
  - Transaction Log on Name node is purged



HDFS High – Throughput Access to Large Data Sets (Files)

- HDFS primarily designed for **batch processing** rather than **interactive processing**
- Individual files are broken into large blocks (ex: 64 MB) to allow HDFS to decrease the amount of metadata storage required per file
- 2 Advantages
  - List of blocks per file will shrink as the size of individual blocks increases
  - Keeping large amounts of data sequentially within a block, HDFS provides fast streaming reads of a data

Streaming Data

- Streaming access enables data to be transferred in the form of a steady and continuous stream
- HDFS *starts sending the data as it reads the file;* does not wait for the entire file to be read
- Client consuming this data starts processing immediately
- This makes data processing really first

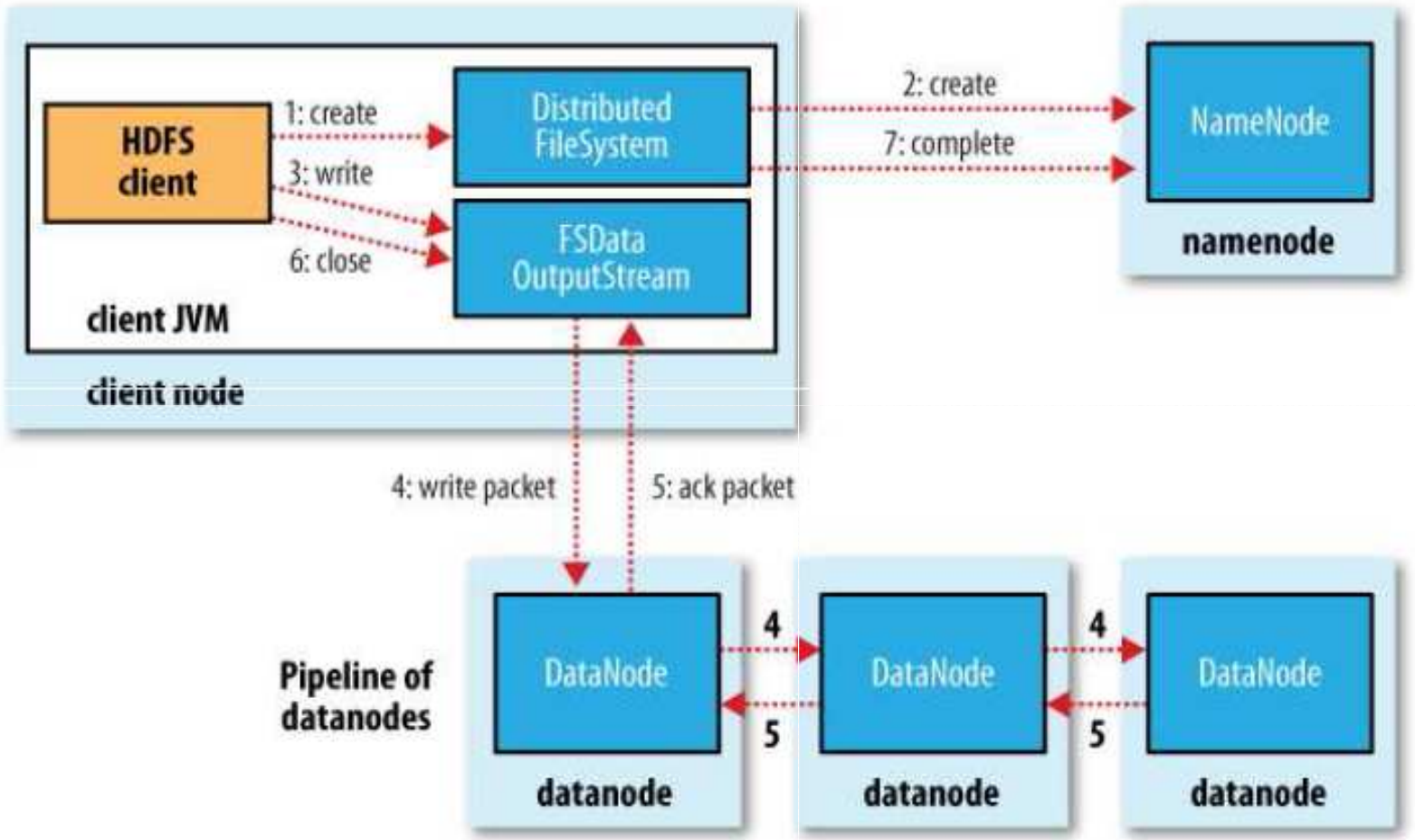
Writing a file into HDFS

- To write a file in HDFS, a user sends a **"create"** request to the **NameNode** to create a new file in the file system namespace.
- If the file does not exist, the NameNode notifies the user and allows him to start writing data to the file by calling the **write** function.
- The first block of the file is written to an internal queue termed the **data queue** while a **data streamer** monitors its writing into a **DataNode**.
- Since each file block needs to be **replicated** by a predefined factor, the data streamer first sends a request to the NameNode to get a list of **suitable** DataNodes to store replicas of the first block.

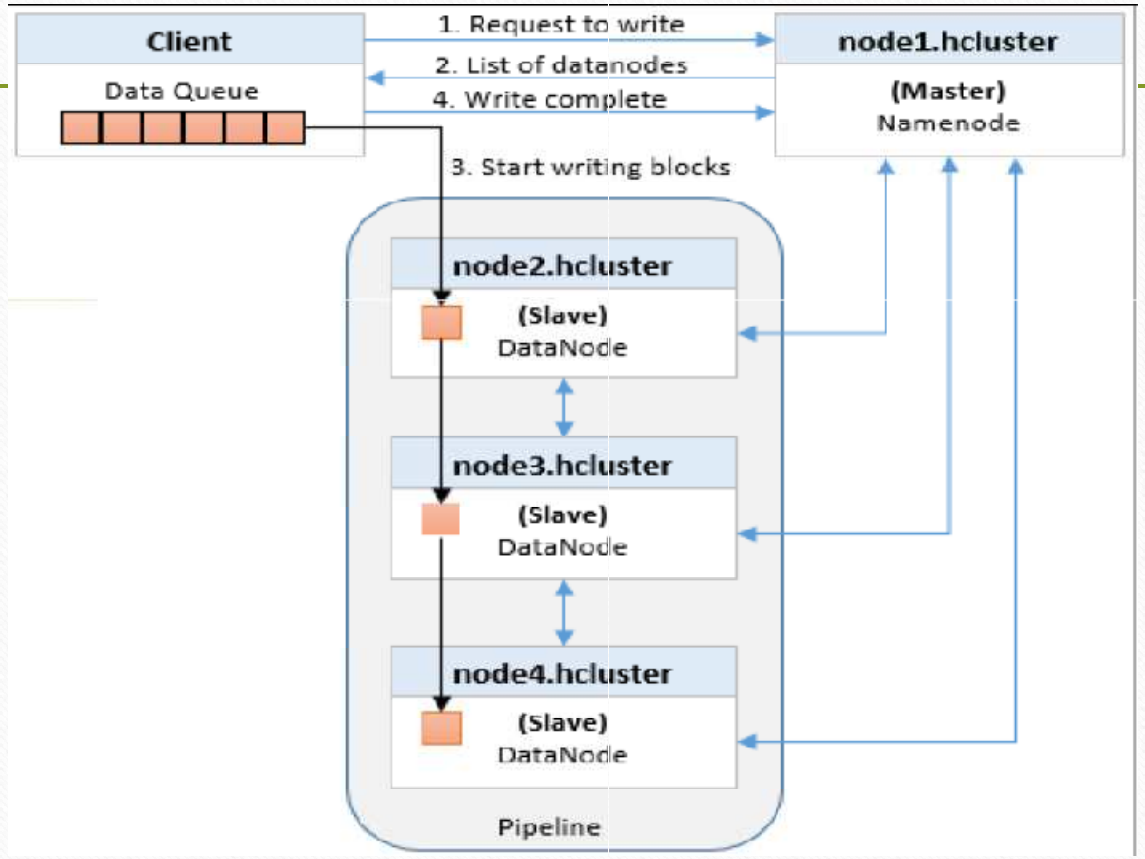
Writing a file into HDFS

- The steamer then stores the block in the **first** allocated DataNode.
- Afterward, the block is **forwarded** to the second DataNode by the first DataNode.
- The process continues until all allocated DataNodes receive a replica of the first block from the previous DataNode.
- Once this replication process is finalized, the same process starts for the second block and continues until all blocks of the file are stored and replicated on the file system.

HDFS Data I/O-Write



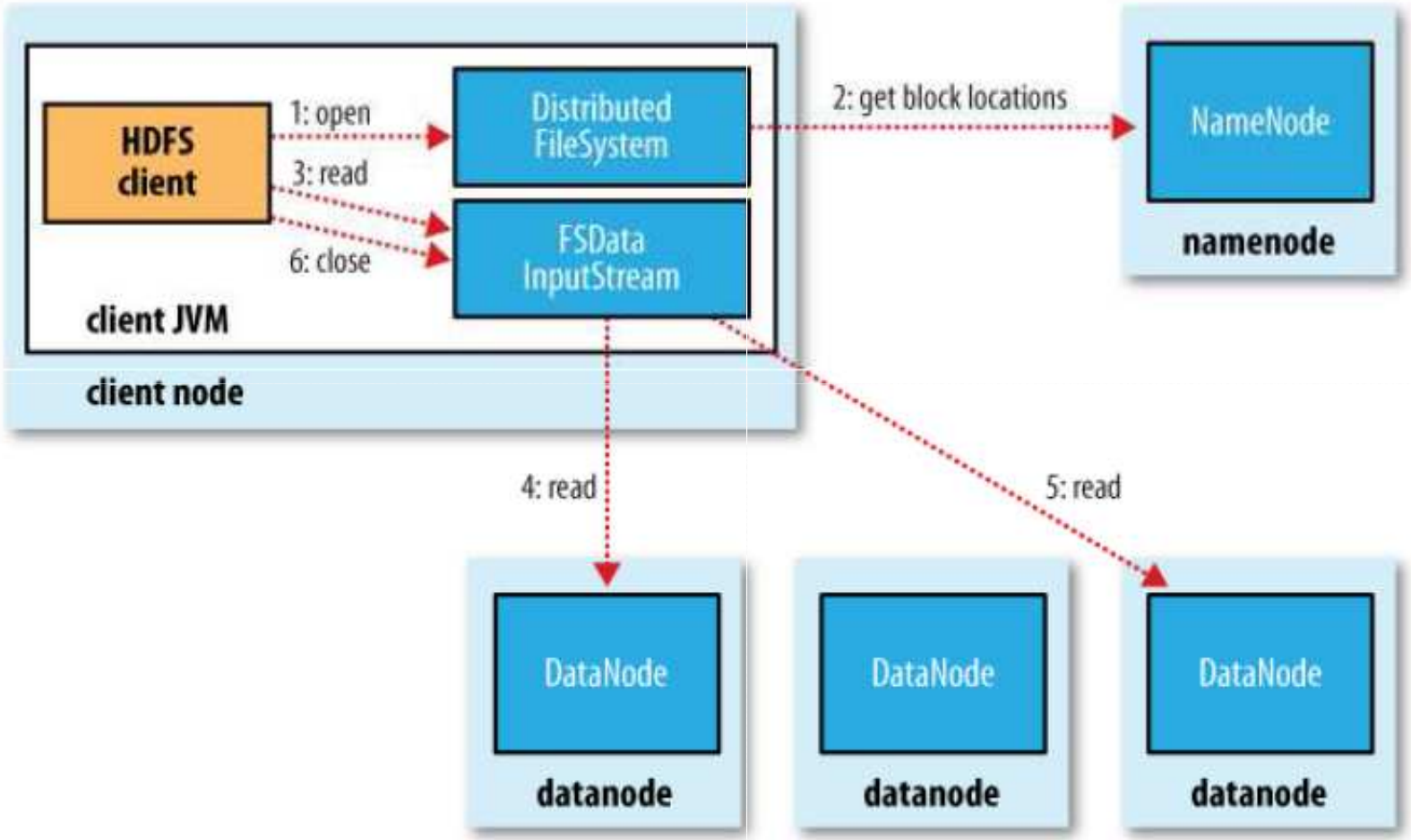
Writing a file into HDFS



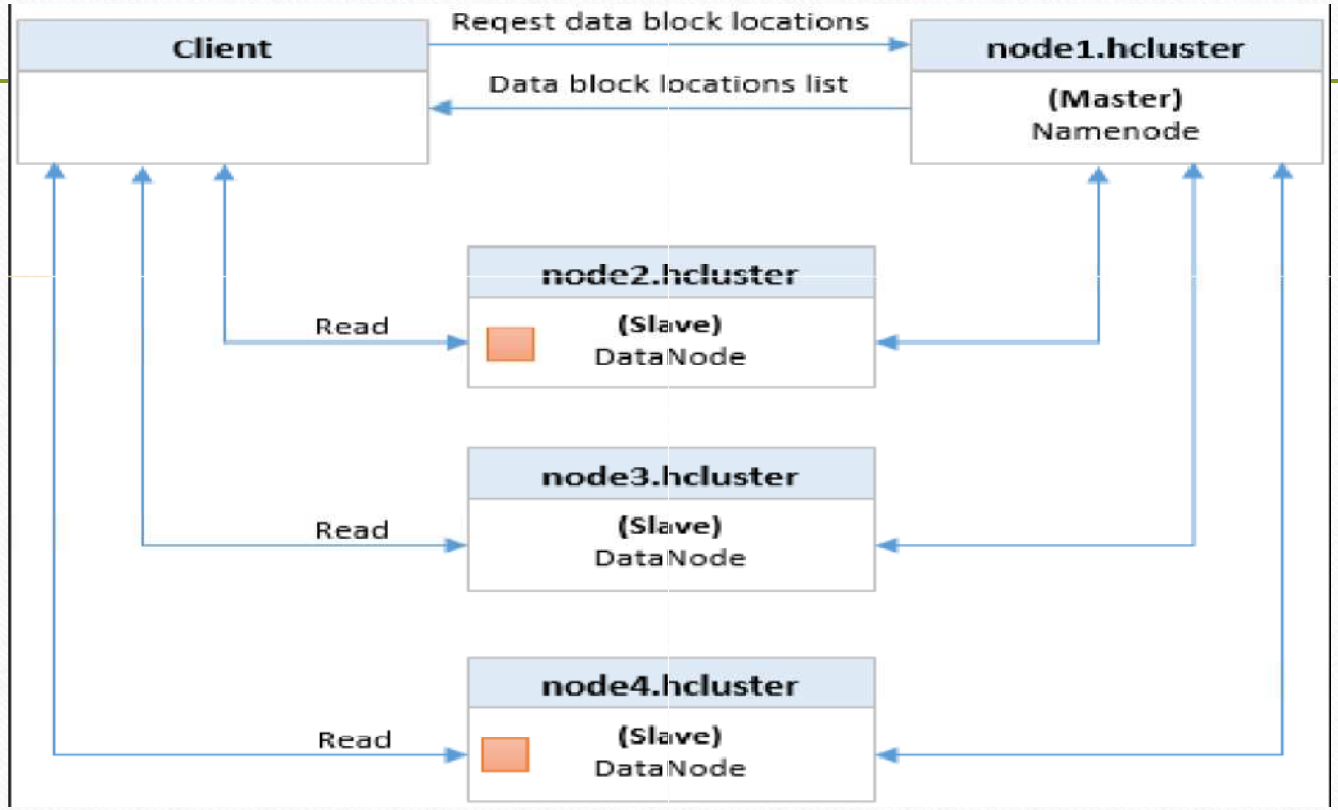
Reading a file from HDFS

- To read a file in HDFS, a user sends an **"open"** request to the **NameNode** to get the location of file blocks.
- For each file block, the NameNode returns the address of a set of **DataNodes** containing replica information for the requested file. The number of addresses depends on the number of block replicas.
- Upon receiving such information, the user calls the **read** function to connect to the **closest** DataNode containing the first block of the file.
- After the first block is streamed from the respective DataNode to the user, the established connection is terminated
- The same process is repeated for all blocks of the requested file until the whole file is streamed to the user.

HDFS Data I/O-Read



Reading a file from HDFS



Thank You!