UNIT - III BIG DATA PROCESSING

Big Data technologies, Introduction to Google file system, Hadoop Architecture, Hadoop Storage: HDFS, Common Hadoop Shell commands, Anatomy of File Write and Read, NameNode, Secondary NameNode, and DataNode, Hadoop MapReduce paradigm, Map Reduce tasks, Job, Task trackers - Cluster Setup – SSH & Hadoop Configuration, Introduction to: NOSQL, Textual ETL processing.

Distributed Computing

- **Distributed computing**
  - networking several computers together and taking advantage of their individual resources in a collective way.
- Each computer contributes some of its resources (such as memory, processing power and hard drive space) to the overall network.
- It turns the entire network into a massive computer, with each individual computer acting as a processor and data storage device.

- Need for a scalable DFS
- Large distributed data-intensive applications
- High data processing needs
- Performance, Reliability, Scalability and Availability
- More than traditional DFS

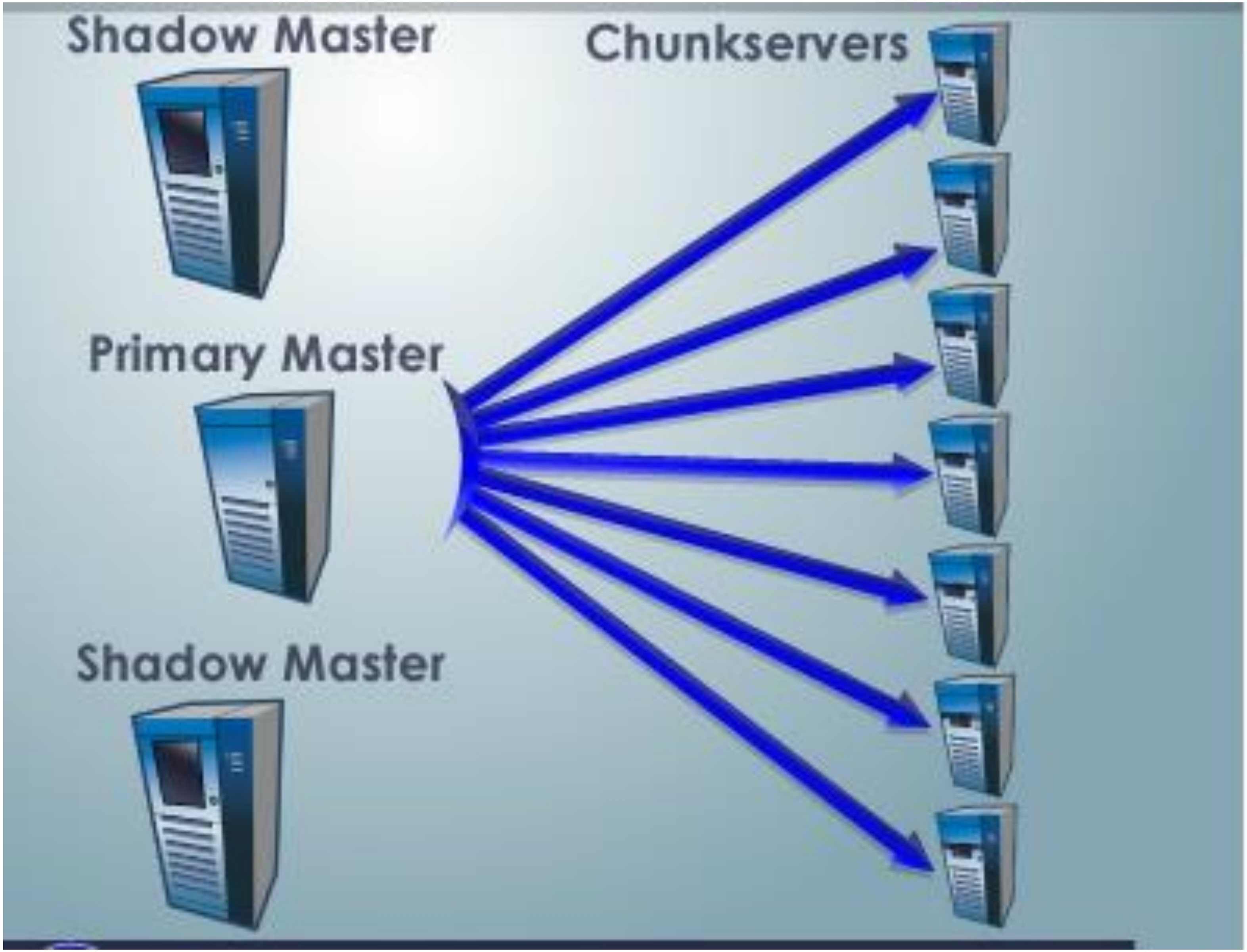
Google File System

Assumptions –Environment

- Commodity Hardware
  - inexpensive
- Component Failure
  - the norm rather than the exception
- TBs of Space
  - must support TBs of space

Architecture

- Files are divided into *chunks*
- Fixed-size chunks (64MB)
- Replicated over *chunkservers,* called *replicas*
- Unique 64-bit *chunk handles*
- Chunks as Linux files

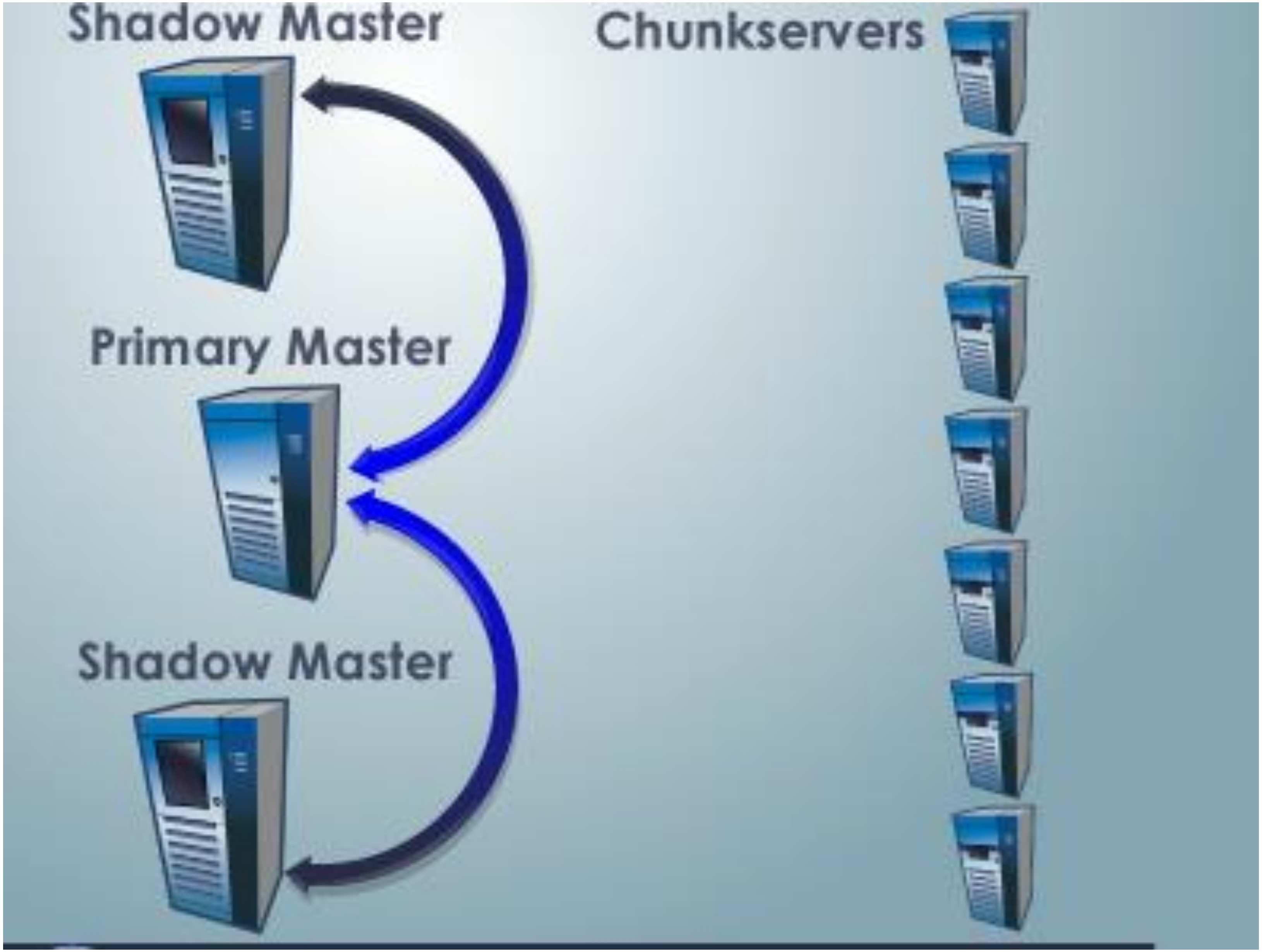


Shadow Master

Chunkservers

Primary Master

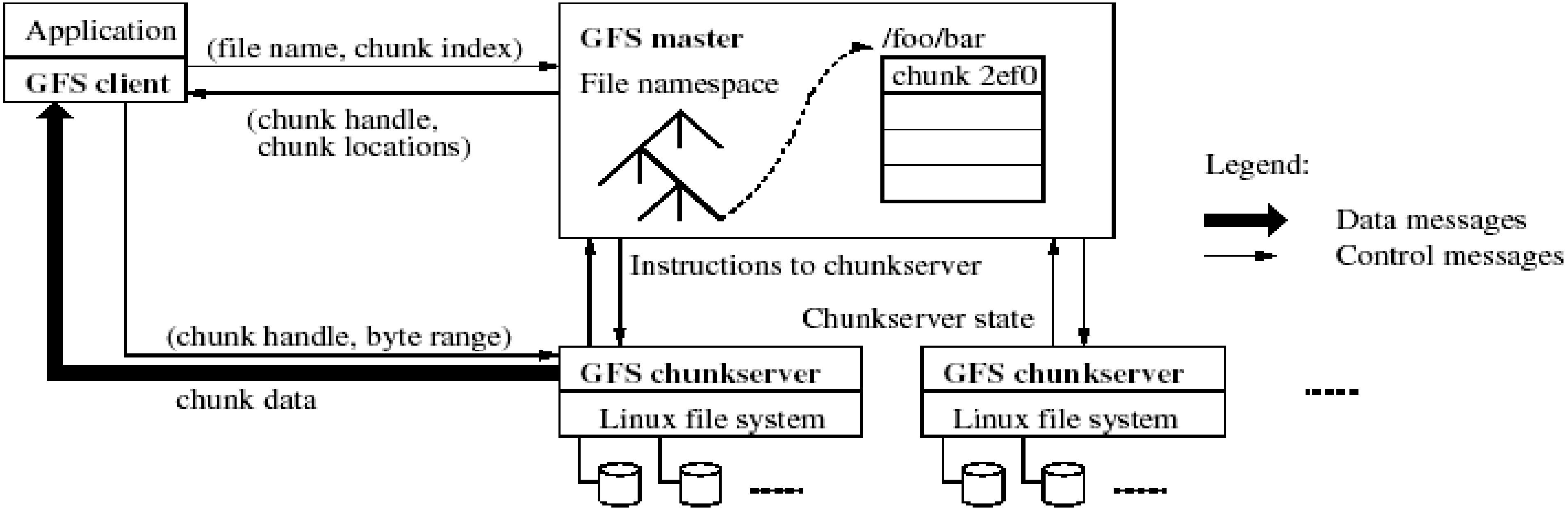
Shadow Master


Shadow Master
Chunkservers
Primary Master
Shadow Master

Architecture

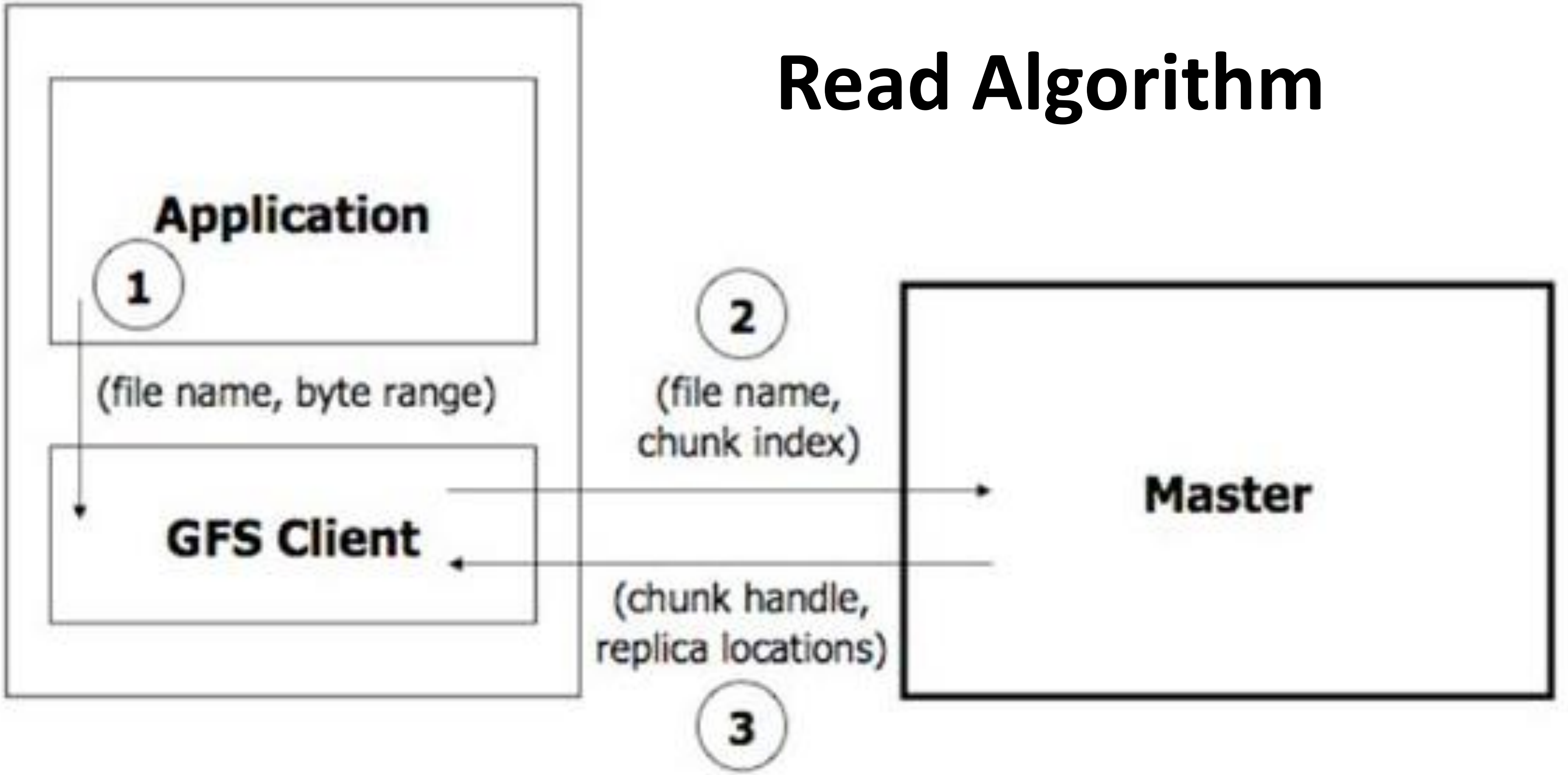
- Single master
- Multiple chunkservers
  - Grouped into Racks
  - Connected through switches
- Multiple clients
- Master/chunkserver coordination
  - HeartBeat messages

Architecture

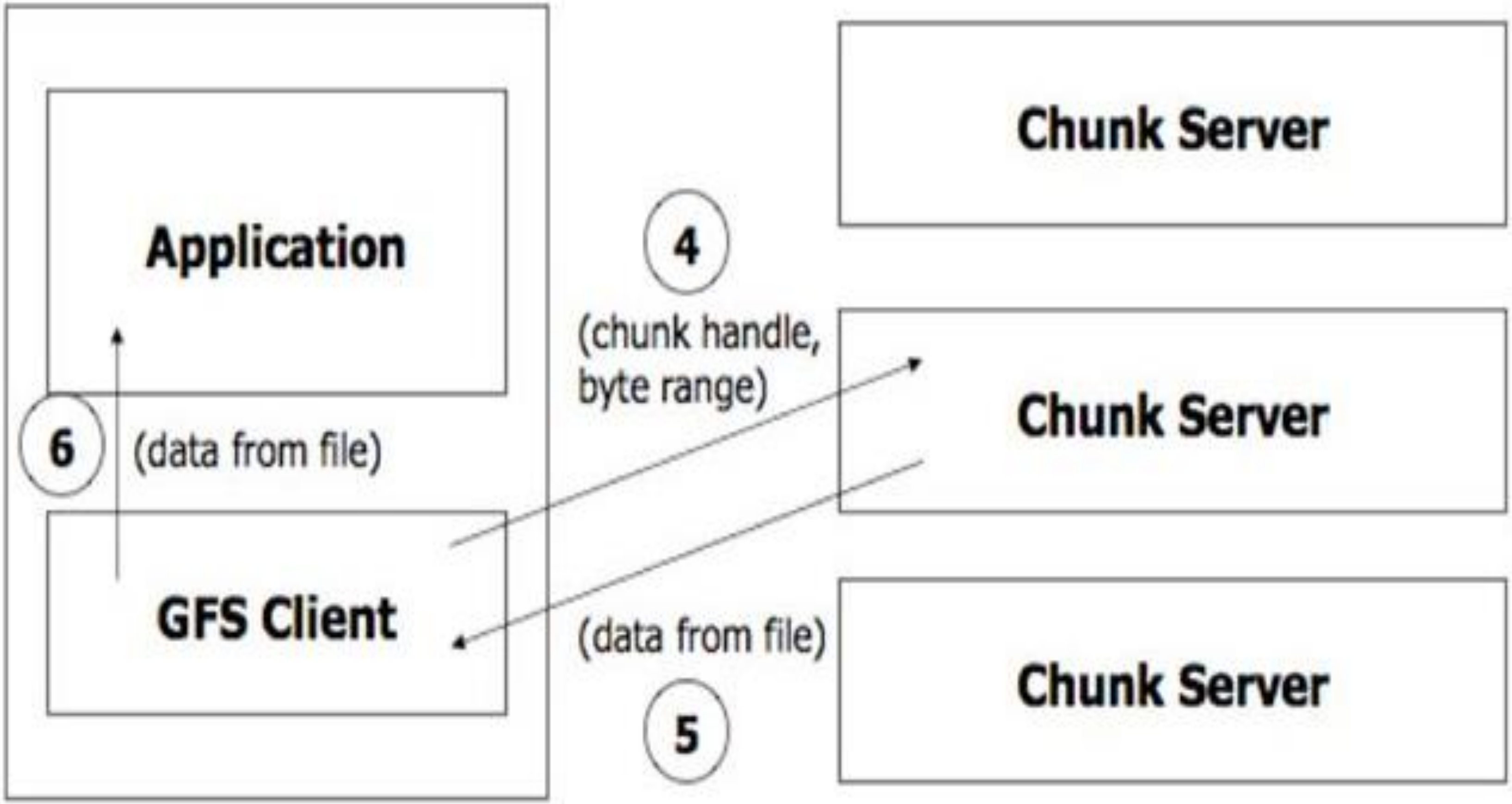


- Contact single *master*
- Obtain chunk locations
- Contact one of chunkservers
- Obtain data

Read Algorithm



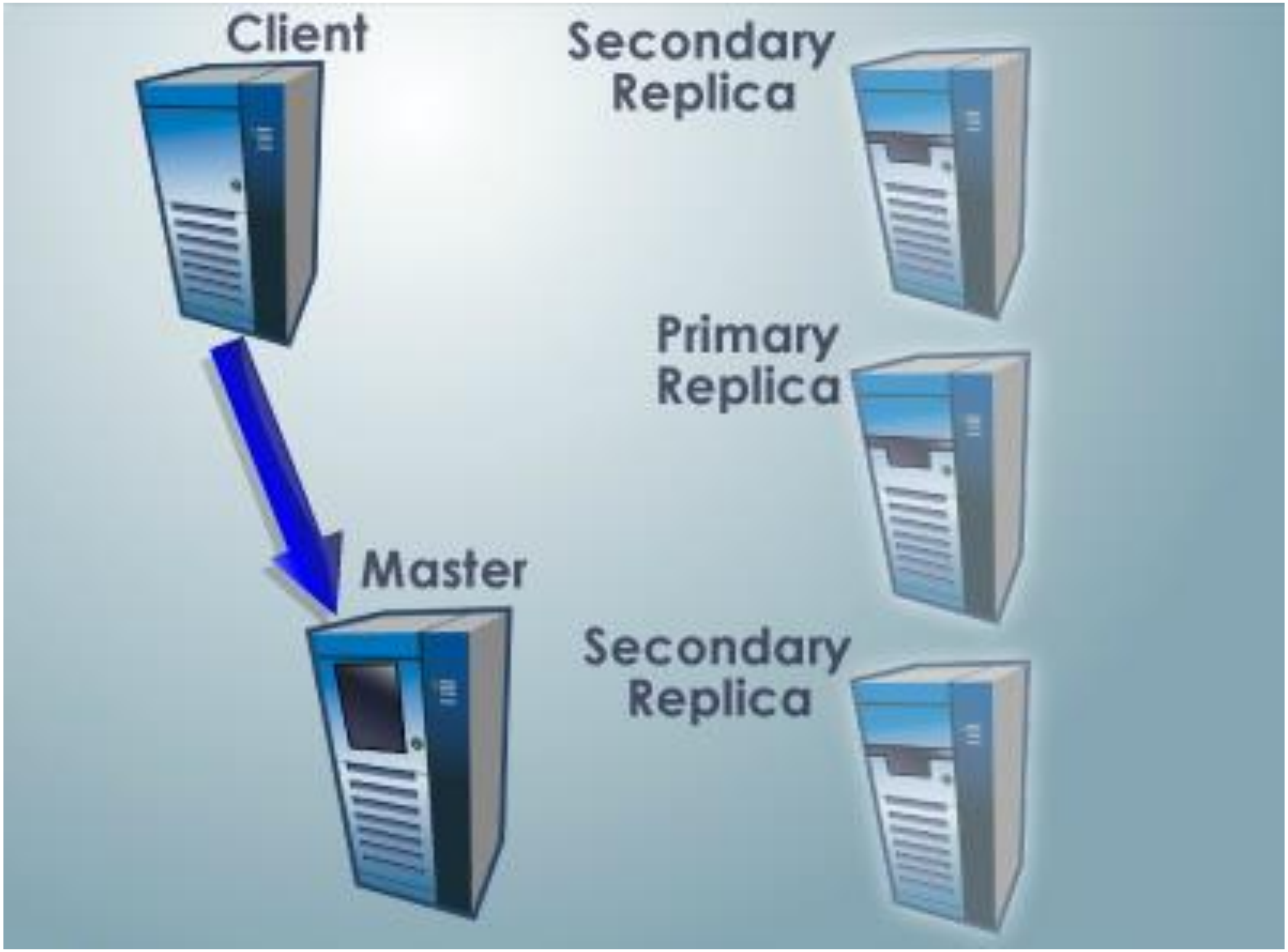
1. Application originates request
2. GFS client translate the request and sends it to master
3. Master Responds with chunk handle and replica locations



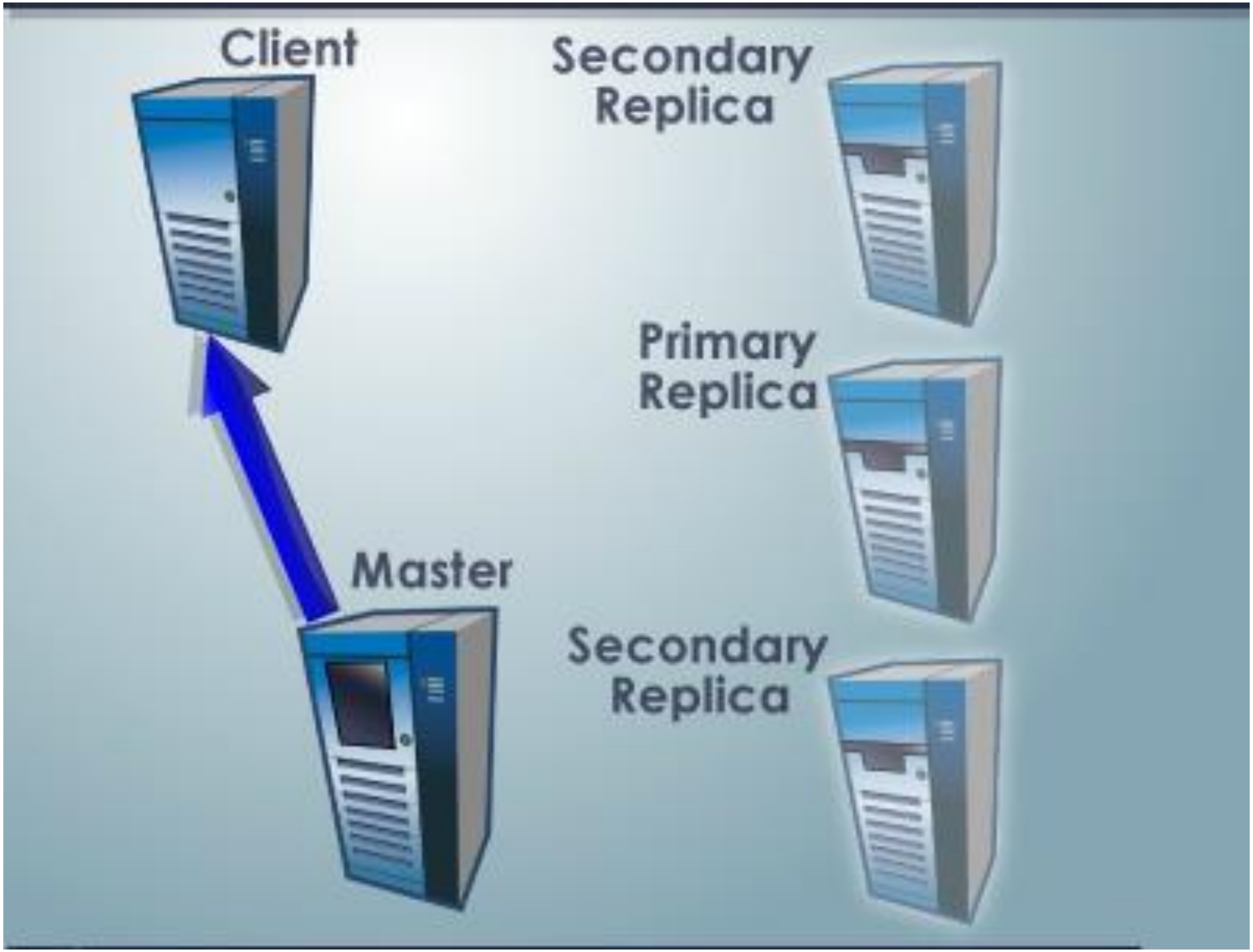
4. Client Picks the location and sends the request to chunk server
5. Chunk server sends the requested data
6. Data forwarded to application

Mutation

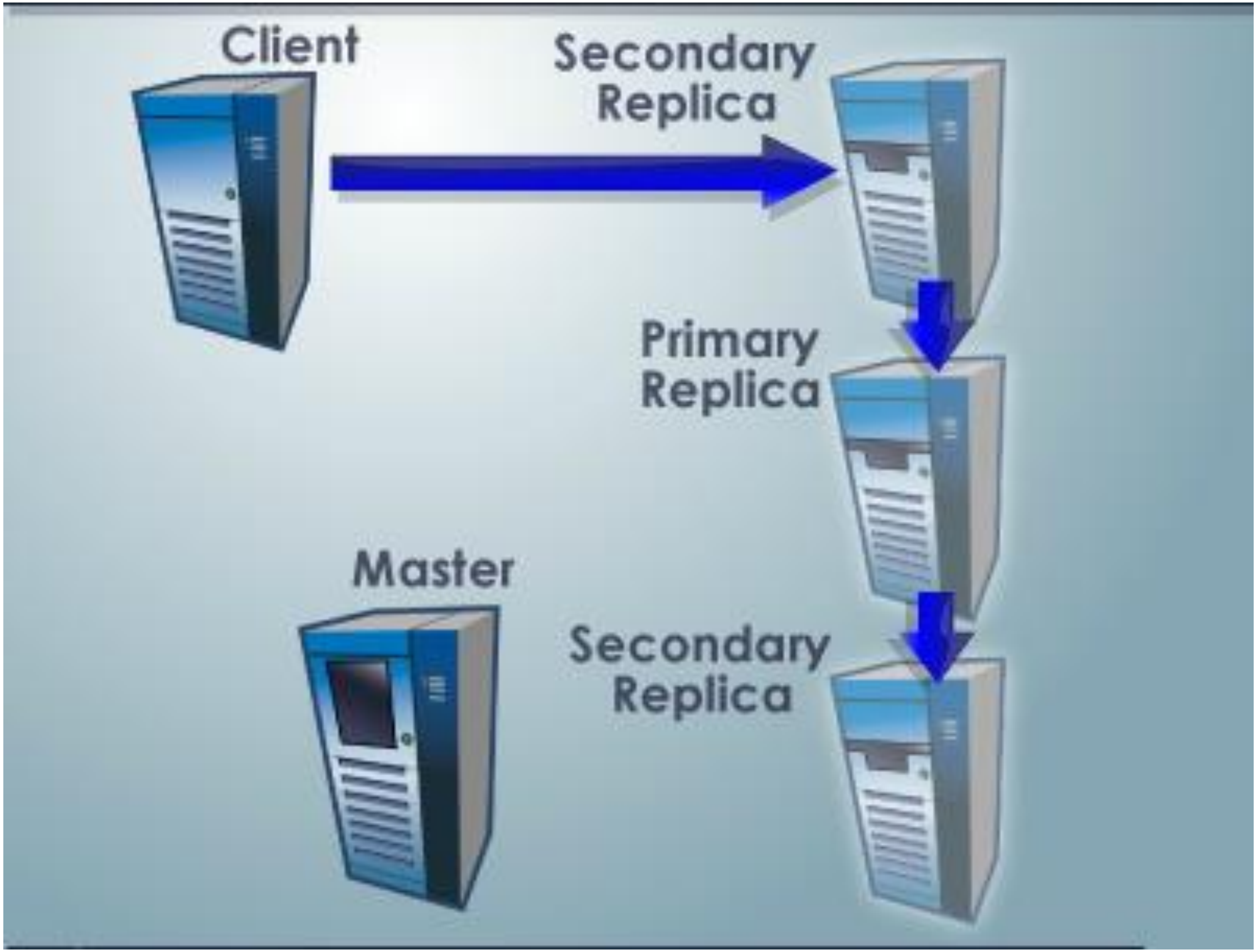
- The changes are called as mutations



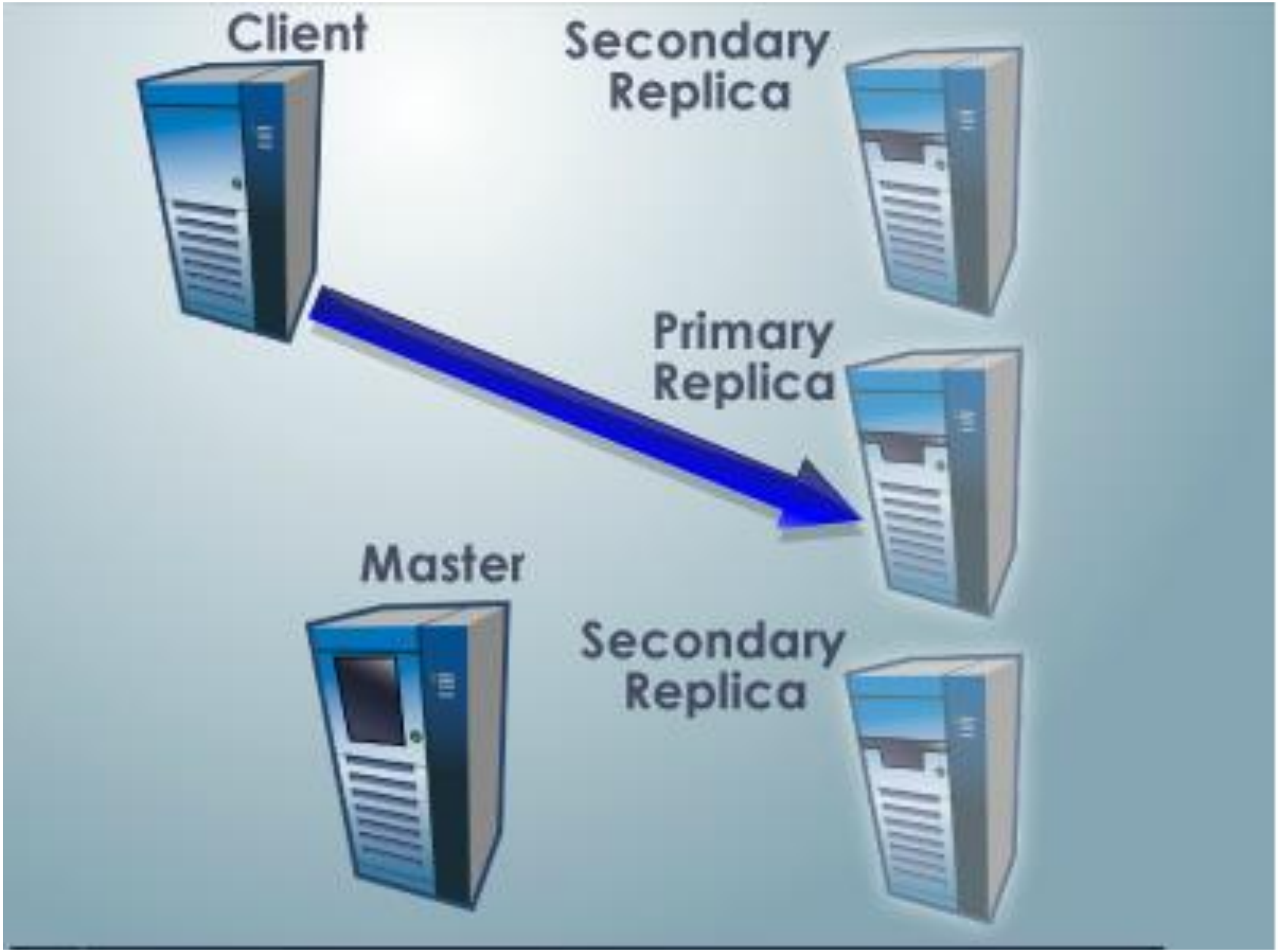
Clients sends the request to the master to find the location of chunkserver acting as primary replica .



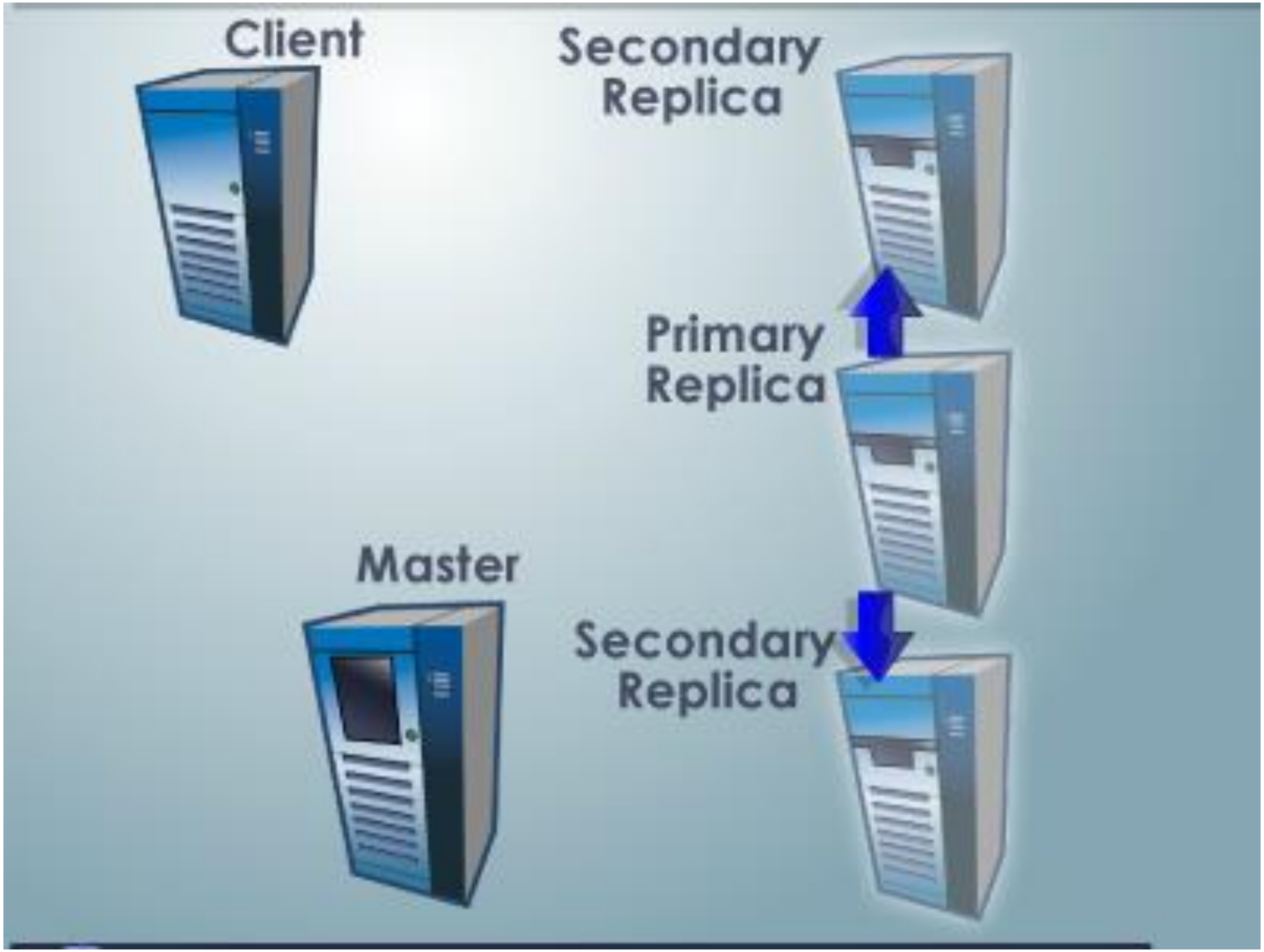
The master sends the location of the chunkserver replicas and indentifies the primary replica



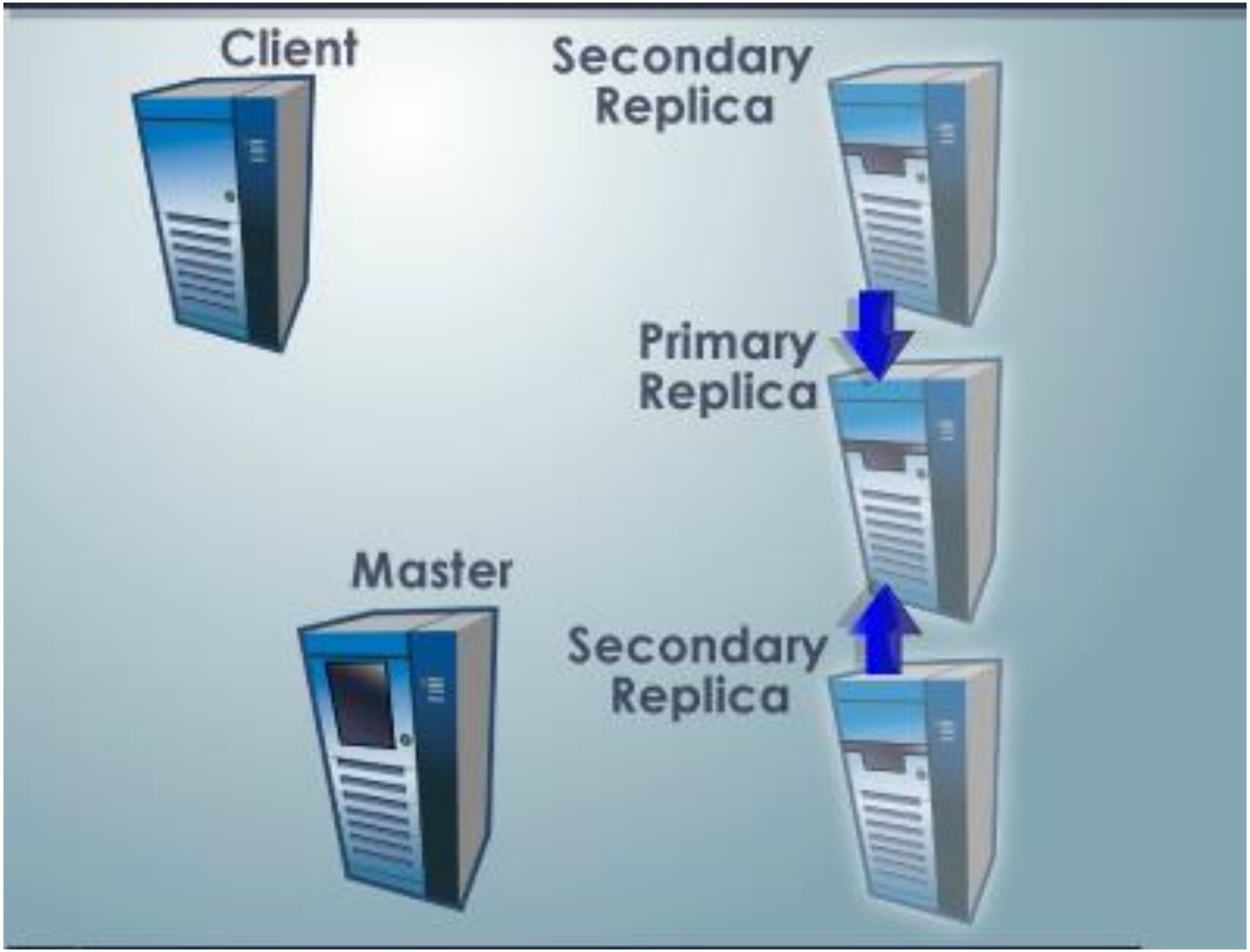
The client sends the write data to all the chunkserver replicas starting with the closest and pipelining the data through



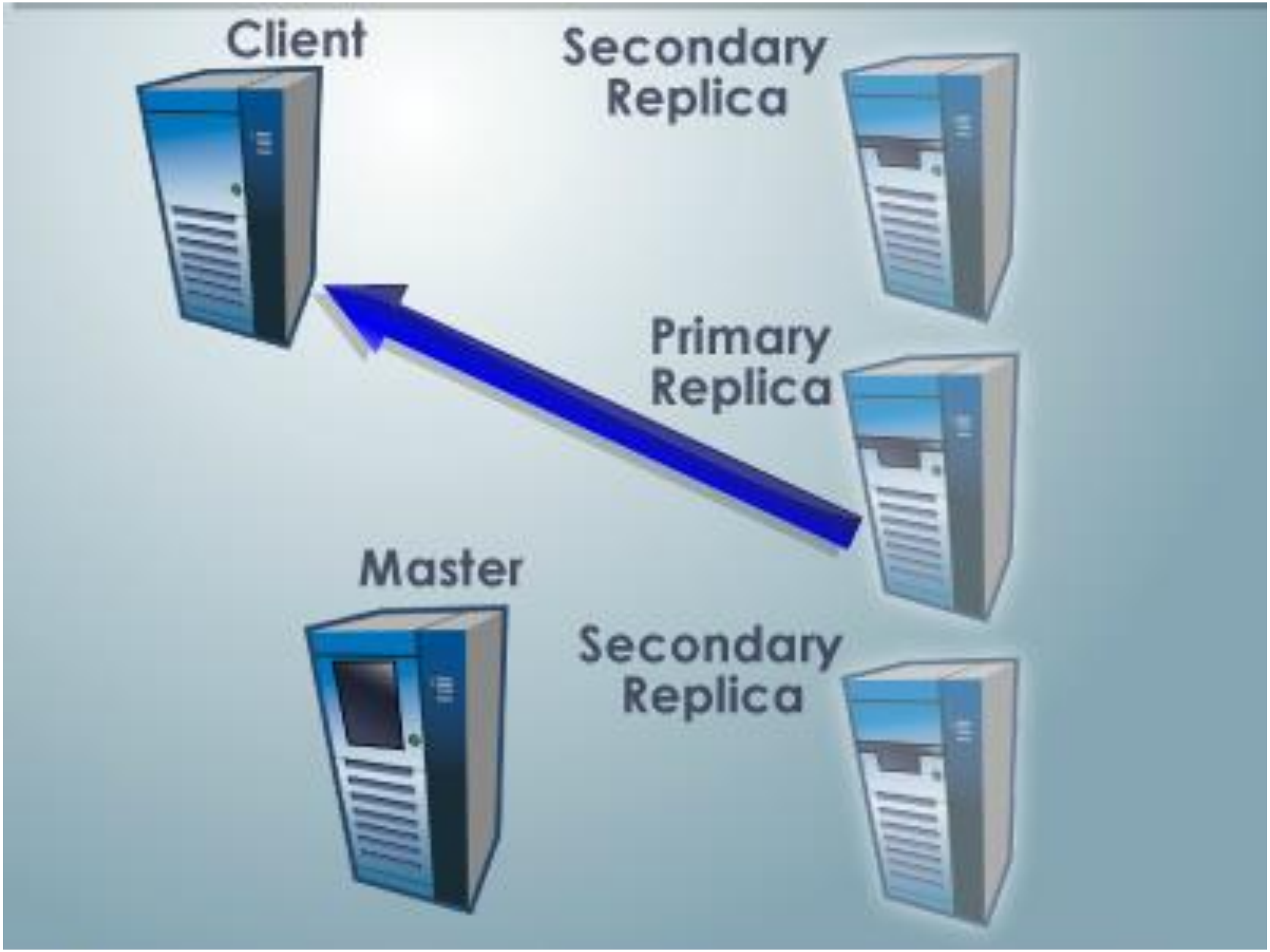
Once the replicas receive the data, the client tells the primary replica to begin the write application



The primary replica writes the data to the appropriate chunk and then tells the secondary replica to do the same

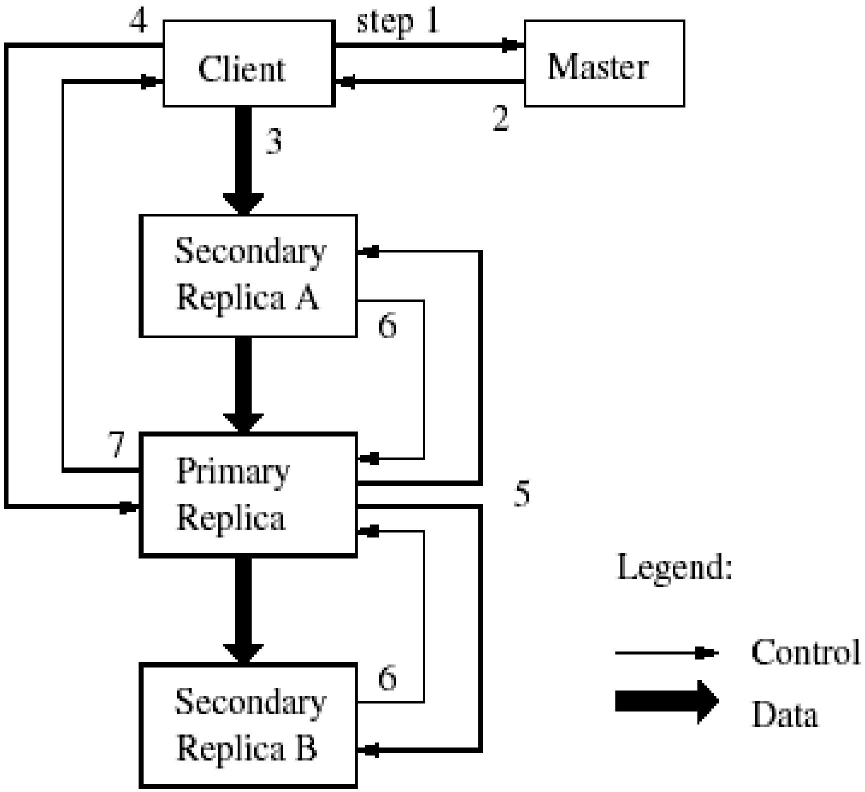


The secondary replicas complete the write function and report back to primary replica



The primary replica sends the confirmation to the client

Implementation – Writes



Mutation Order

→ *identical replicas*

Master

- **Metadata**
  - Three types
    - File & chunk namespaces
    - Mapping from files to chunks
    - Locations of chunks’ replicas
  - Replicated on multiple remote machines
  - Kept in memory

- **Operations**
  - Replica placement
  - New chunk and replica creation
  - Load balancing
  - Unused storage reclaim

Consistency Model

- Two types of mutations
  - **Writes**
    - Cause data to be written at an application-specified file offset
  - **Record appends**
    - Operations that append data to a file
    - Cause data to be appended

- States of a file region after a mutation
  - *Consistent*
    - All clients see the same data, regardless which replicas they read from
  - *Defined*
    - *consistent* + all clients see what the mutation writes in its entirety
  - *Undefined*
    - *consistent* +but it may not reflect what any one mutation has written
  - *Inconsistent*
    - Clients see different data at different times

Leases and Mutation Order

- Master uses *leases* to maintain a consistent mutation order among replicas
- *Primary* is the chunkserver who is granted a chunk lease
- All others containing replicas are *secondary*
- Primary defines a mutation order between mutations
- All *secondary* follows this order

Snapshot

- Goals
  - To quickly create branch copies of huge data sets
  - To easily checkpoint the current state

- Copy-on-write technique
  - Metadata for the source file or directory tree is duplicated
  - Reference count for chunks are incremented
  - Chunks are copied later at the first write

Fault Tolerance and Diagnosis

- Fast Recovery
  - Operation log
  - Check pointing

- Chunk replication
  - Each chunk is replicated on multiple chunkservers on different racks

- Master replication
  - Operation log and check points are replicated on multiple machines

- Data integrity
  - Checksumming to detect corruption of stored data
  - Each chunkserver independently verifies the integrity

- Diagnostic logs
  - Chunkservers going up and down
  - RPC requests and replies

Advantages

- Different than previous file systems
- Satisfies needs of the application
- Fault tolerance

Hadoop

Introduction

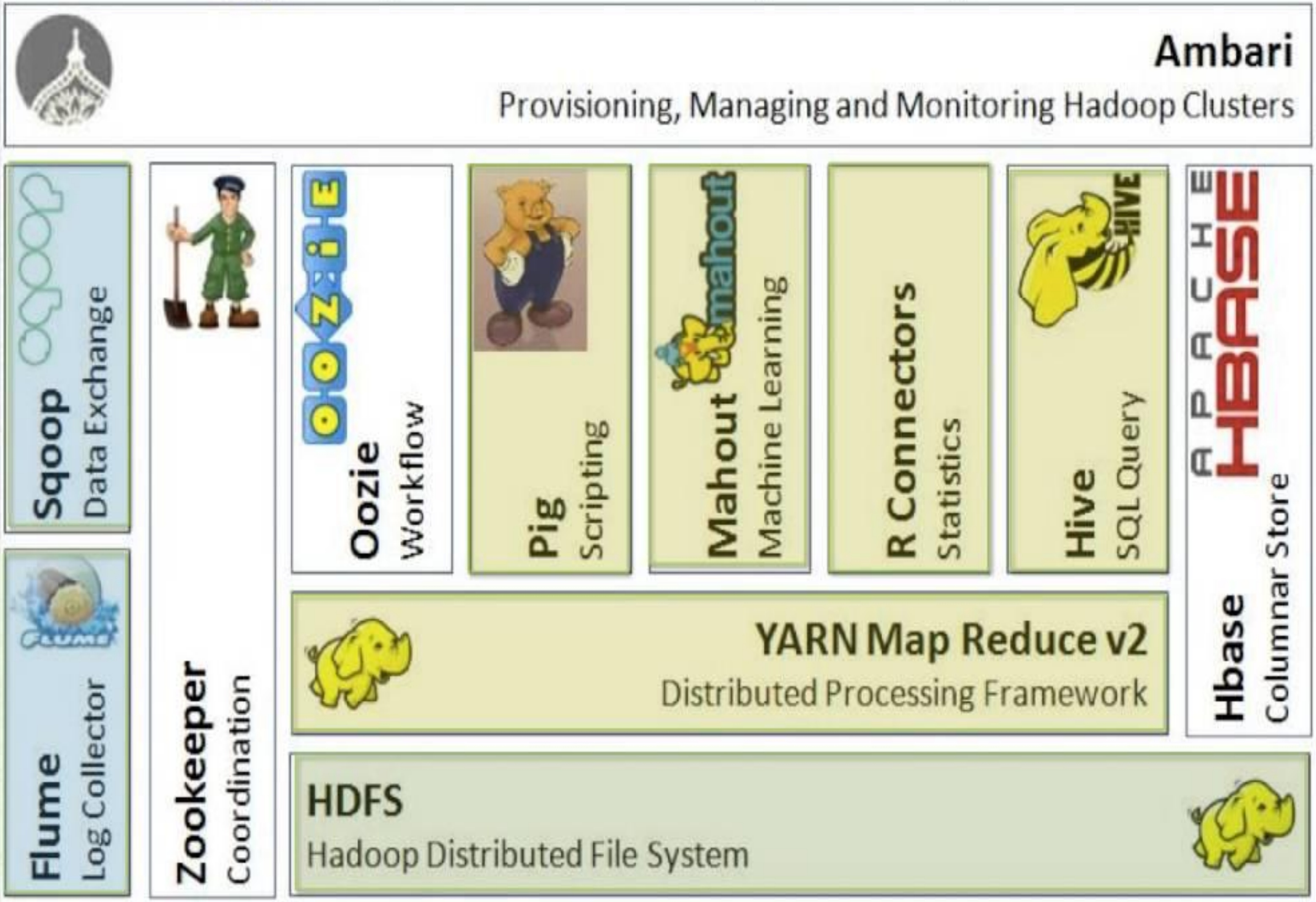
- Hadoop ecosystem
  - A framework of various tools

- Definition
  - Comprehensive collection of tools and technologies that can be effectively implemented and deployed to provide Big Data solutions in cost effective manner

Who uses?

- Amazon/A9
- Facebook
- Google
- New York Times
- Veoh
- Yahoo!
- …. many more

Apache Hadoop Ecosystem



Ambari
Provisioning, Managing and Monitoring Hadoop Clusters

Sqoop
Data Exchange

Flume
Log Collector

Zookeeper
Coordination

Oozie
Workflow

Pig
Scripting

Mahout
Machine Learning

R Connectors
Statistics

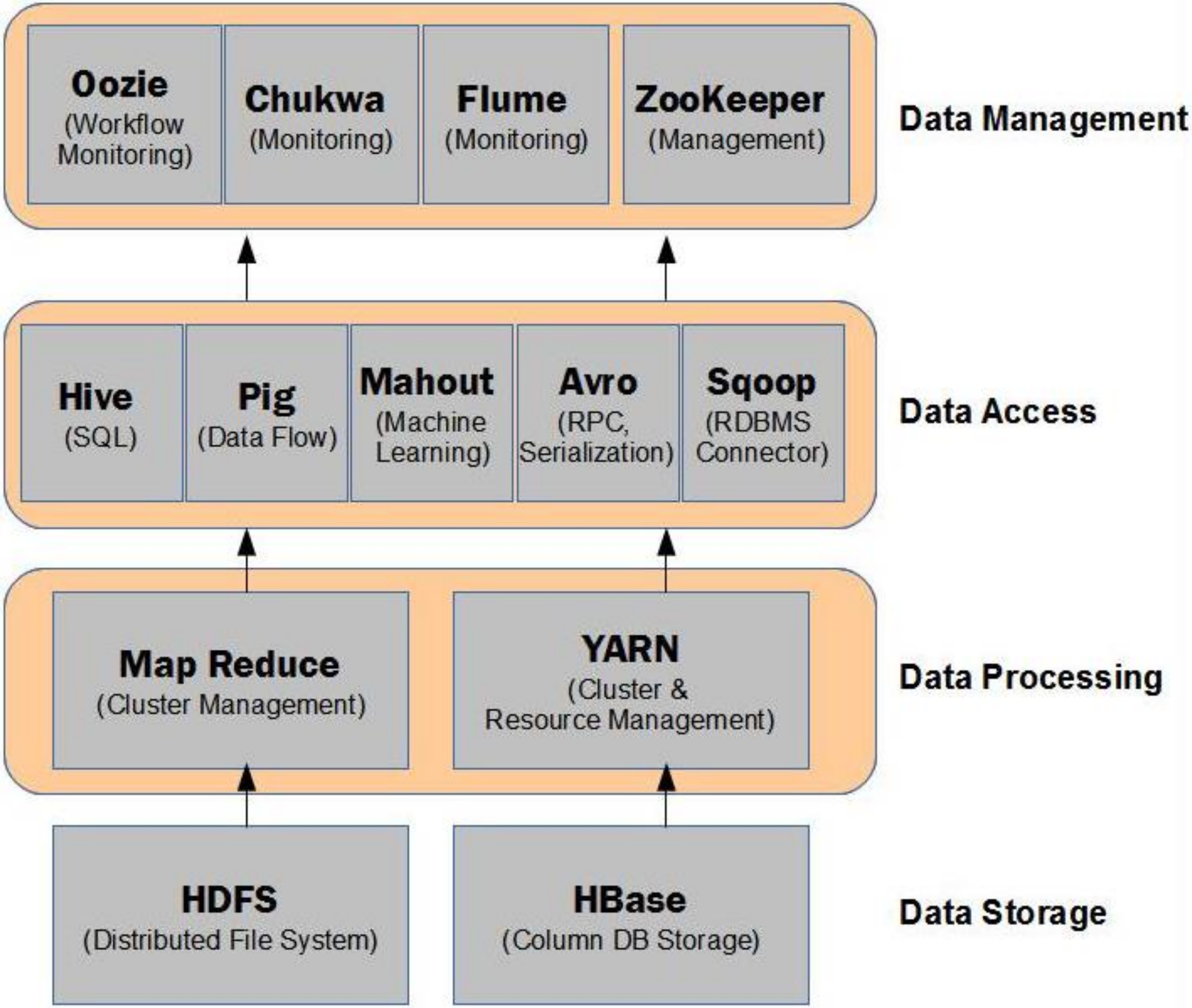
Hive
SQL Query

Hbase
Columnar Store

YARN Map Reduce v2
Distributed Processing Framework

HDFS
Hadoop Distributed File System

Hadoop Ecosystem



Oozie (Workflow Monitoring)

Chukwa (Monitoring)

Flume (Monitoring)

ZooKeeper (Management)

Data Management

Hive (SQL)

Pig (Data Flow)

Mahout (Machine Learning)

Avro (RPC, Serialization)

Sqoop (RDBMS Connector)

Data Access

Map Reduce (Cluster Management)

YARN (Cluster & Resource Management)

Data Processing

HDFS (Distributed File System)

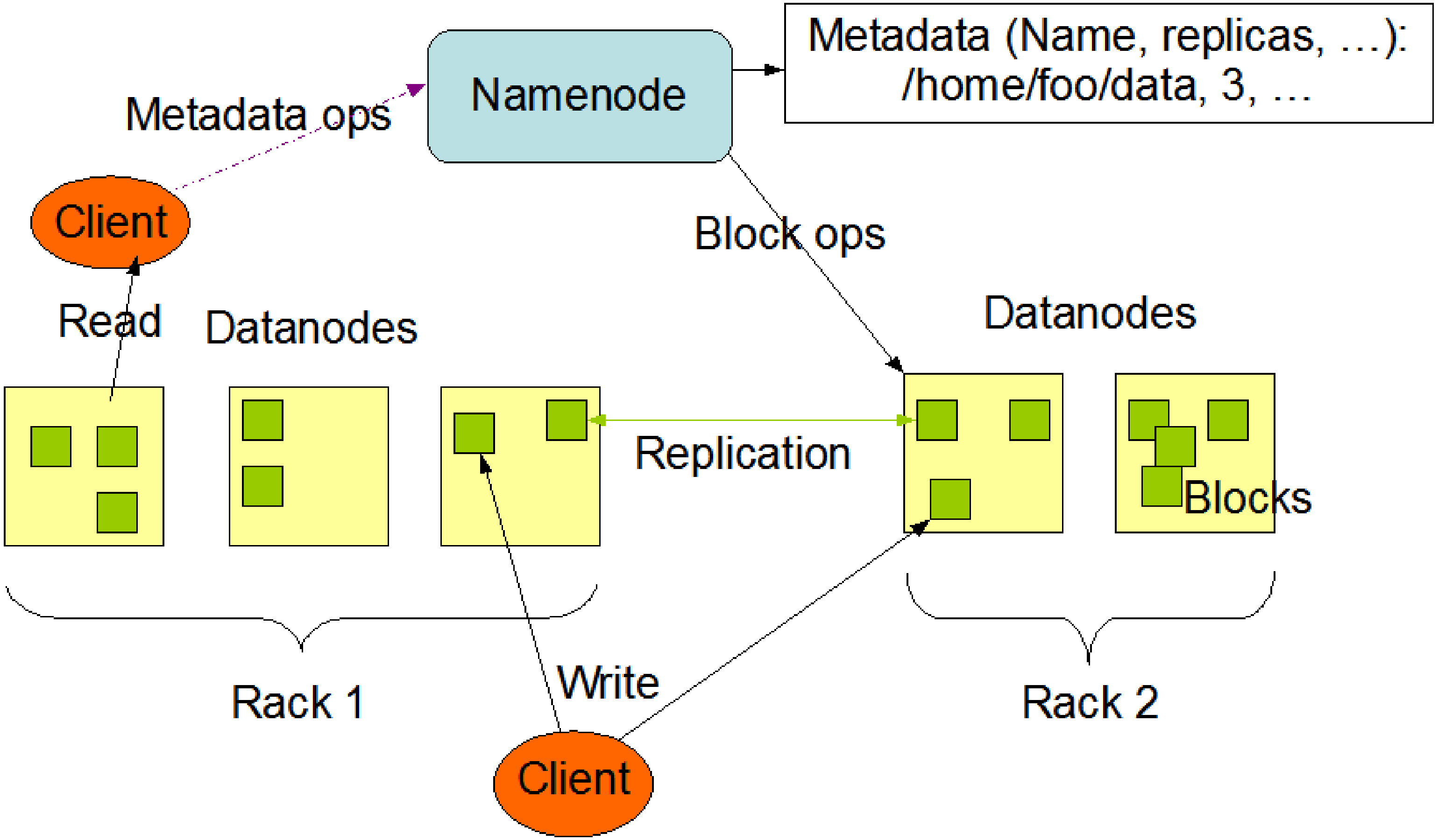
HBase (Column DB Storage)

Data Storage

HDFS

- Hadoop Distributed File System
  - Effective ,Scalable fault tolerant
  - Distributed approach for storing and managing huge volumes of Data
  - Data collected in a cluster is broken down into smaller chunks called as blocks.

HDFS Architecture



Metadata ops

Namenode

Metadata (Name, replicas, …):
/home/foo/data, 3, …

Client

Block ops

Read

Datanodes

Datanodes

Replication

Blocks

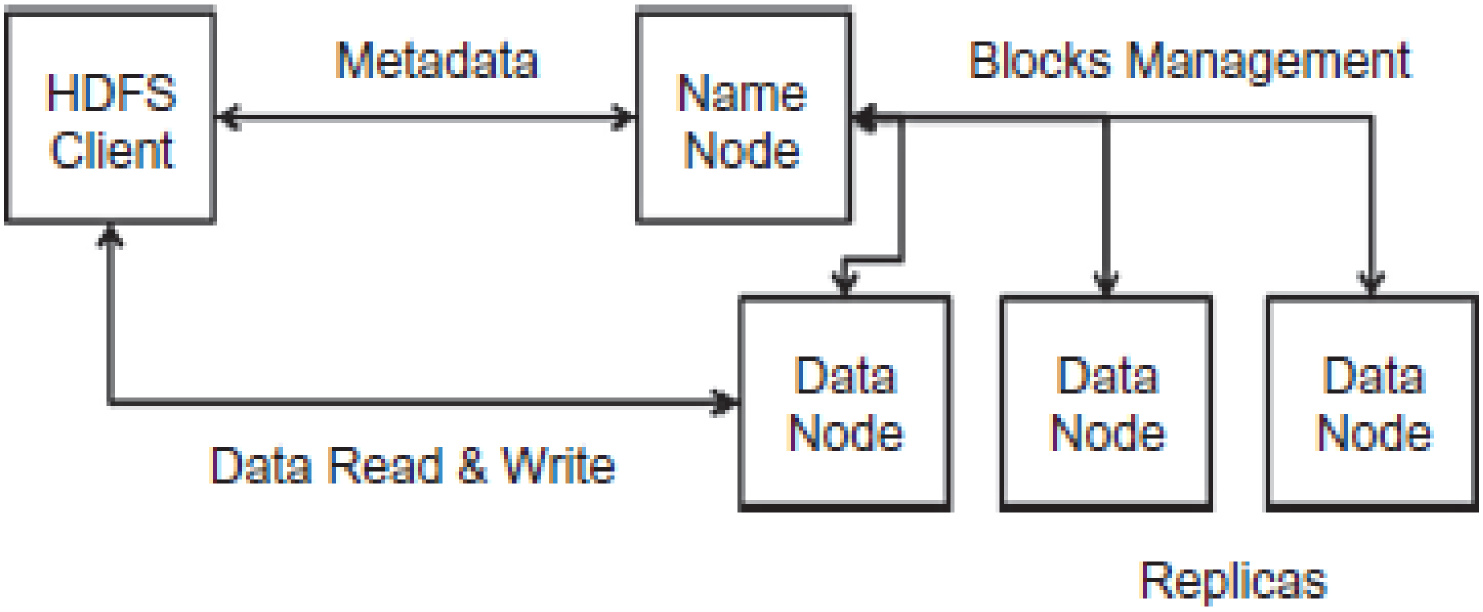
Rack 1

Write

Rack 2

Client

Building Blocks of HDFS



Components

- Namenode
- DataNode
- Image
- Journal
- Checkpoint

NameNode

- ***The NameNode***
  - ***single master server that manages the file system namespace and*** regulates access to files by clients.
  - NameNode manages all the operations like opening , closing, moving, naming, and renaming of files and directories.
  - It also manages the mapping of blocks to DataNodes.

Metadata

- Metadata in Memory
  - The entire metadata is in main memory
  - No demand paging of metadata
- Types of metadata
  - List of files
  - List of Blocks for each file
  - List of DataNodes for each block
  - File attributes, e.g. creation time, replication factor
- A Transaction Log
  - Records file creations, file deletions etc

Functions

- Manages File System Namespace
  - Maps a file name to a set of blocks
  - Maps a block to the DataNodes where it resides
- Cluster Configuration Management
- Replication Engine for Blocks

- fsImage :
  - Stores the complete state of the file system at a gpoint
  - FsImage is a snapshot of the HDFS file system metadata at a certain point of time
- edit log
  - EditLog is a transaction log which contains records for every change that occurs to file system metadata
  - File that contains a log of each file system change
  - Namenode stores modification to the file system as a log appended to native file system ,edits

- At the time of startup the namenode merges the fsimage and edit logs .
- If the edit logs file is very large the Namenode startup will require more time
- So secondaryNameNode comes into picture

Secondary Namenode

- Dedicated node whose function is to take checkpoints of metadata present on namenode
- Helper for namenode but not replacer.

Secondary NameNode

- stores a copy of FsImage file and edit log file
- merges the fsimage and edit logs periodically and keeps the edit log size within a limit.
- Whenever a NameNode is restarted, the latest status of FsImage is built by applying edits records on last saved copy of FsImage.

- NameNode merges FsImage and EditLog files only at start up, the edits log file might get very large over time on a busy cluster.
- If the EditLog is very large, NameNode restart process result in some considerable delay in the availability of file system.
- So, it is important keep the edits log as small as possible which is one of the main functions of Secondary NameNode.

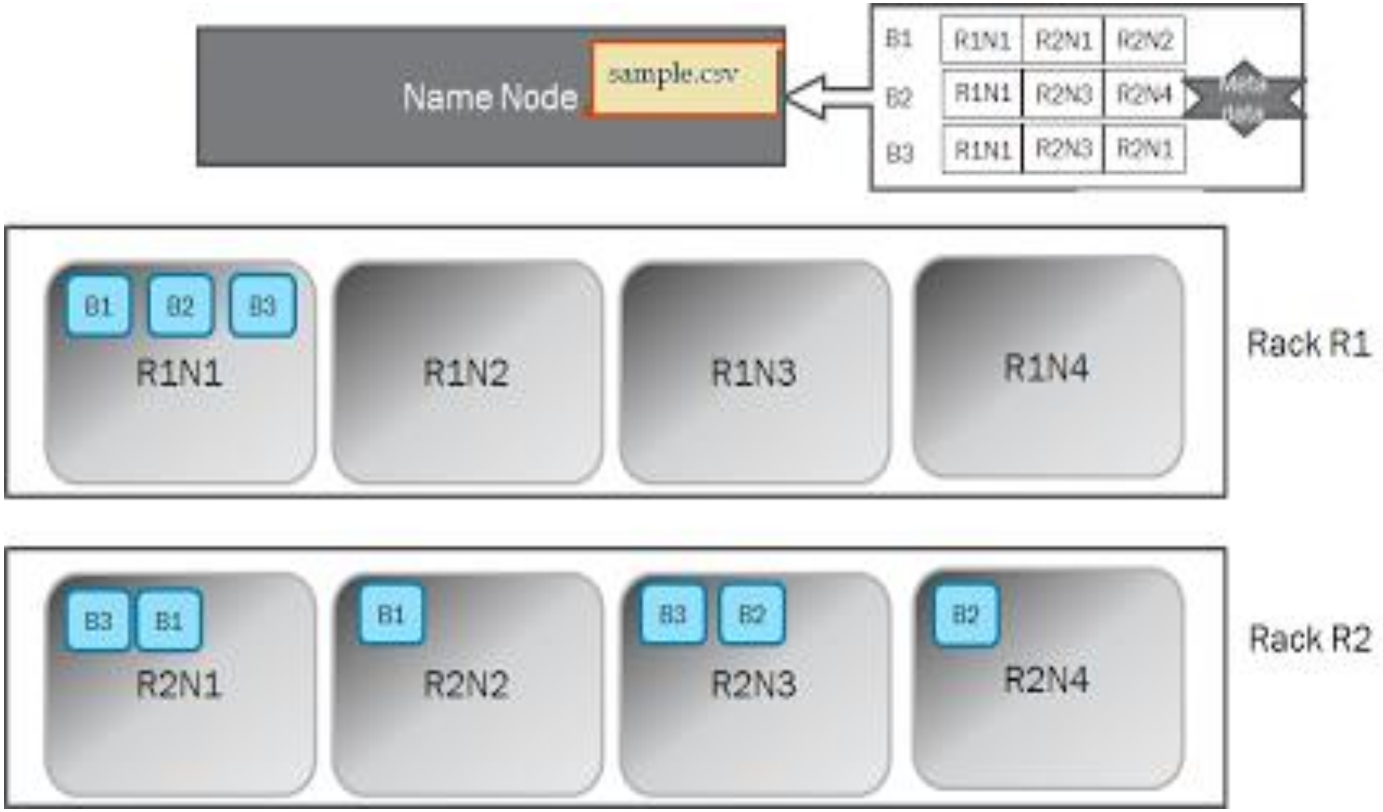
- Secondary NameNode
  - not a true backup of Namenode
  - usually runs on a different machine than the primary NameNode due to its memory requirements
- **Secondary NameNode Functions:**
  - Stores a copy of FsImage file and edits log.
  - Periodically applies edits log records to FsImage file and refreshes the edits log. And sends this updated FsImage file to NameNode so that NameNode doesn't need to re-apply the EditLog records during its start up process. Thus Secondary NameNode makes NameNode start up process fast.
  - If NameNode is failed, File System metadata can be recovered from the last saved FsImage on the Secondary NameNode but Secondary NameNode can't take the primary NameNode's functionality.

DataNodes

- *DataNodes represent the slaves in the architecture that manage data and the storage* attached to the data.

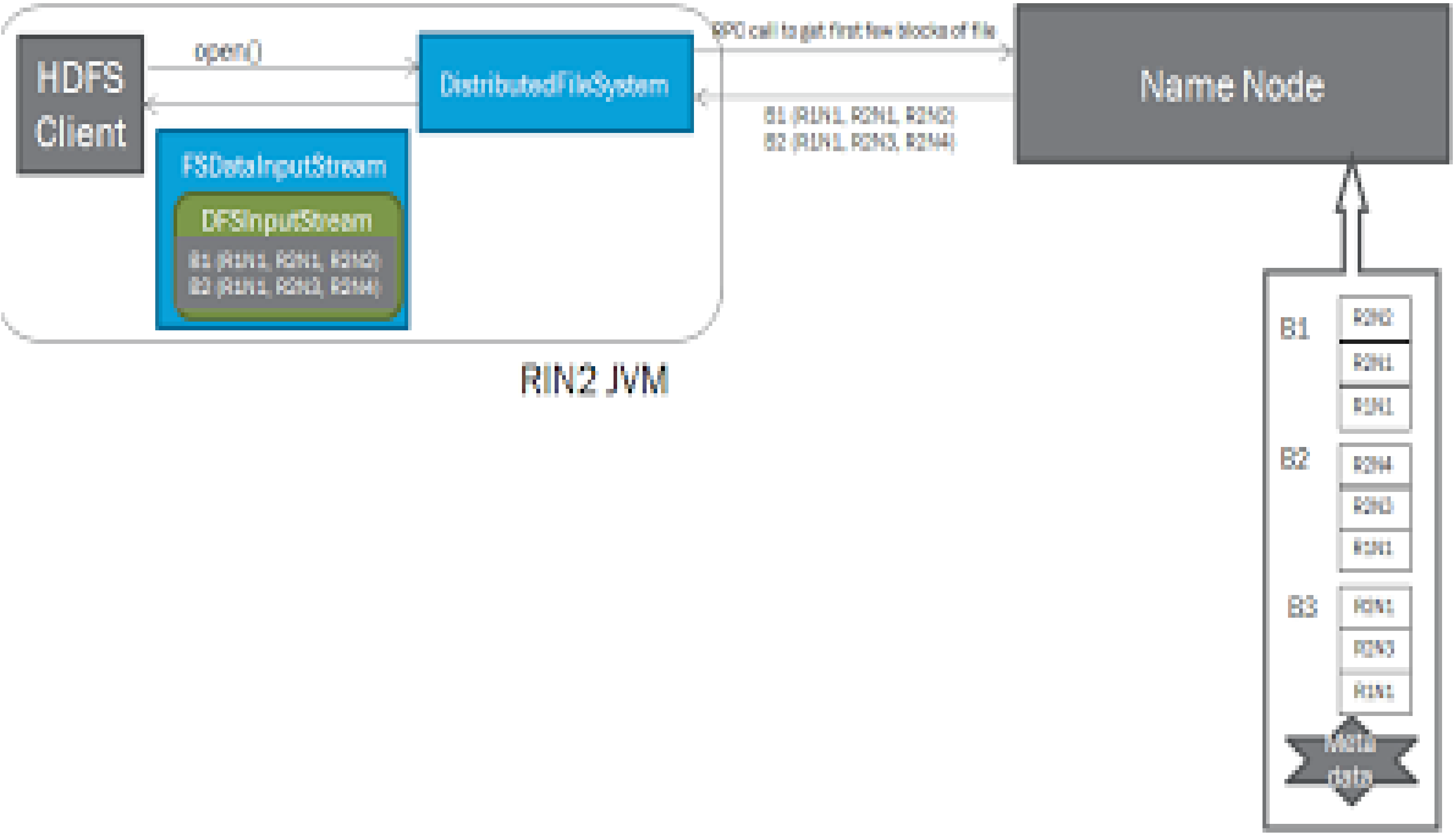
- HDFS exposes a file system namespace and allows user data to be stored in files.
- Internally, a file is split into one or more blocks and these blocks are stored in a set of DataNodes.
- The DataNodes are responsible for serving read and write requests from the file system’s clients.
- The DataNodes also perform block creation, deletion, and replication upon instruction from the NameNode.

File Blocks



B1	R1N1	R2N1	R2N2
B2	R1N1	R2N3	R2N4
B3	R1N1	R2N3	R2N1

In Hadoop V1,sample.csv size say 192 MB then it is divided into 3 blocks of 64 MB each(B1,B2,B3)



HDFS Client

open()

DistributedFileSystem

RPC call to get first few blocks of file

Name Node

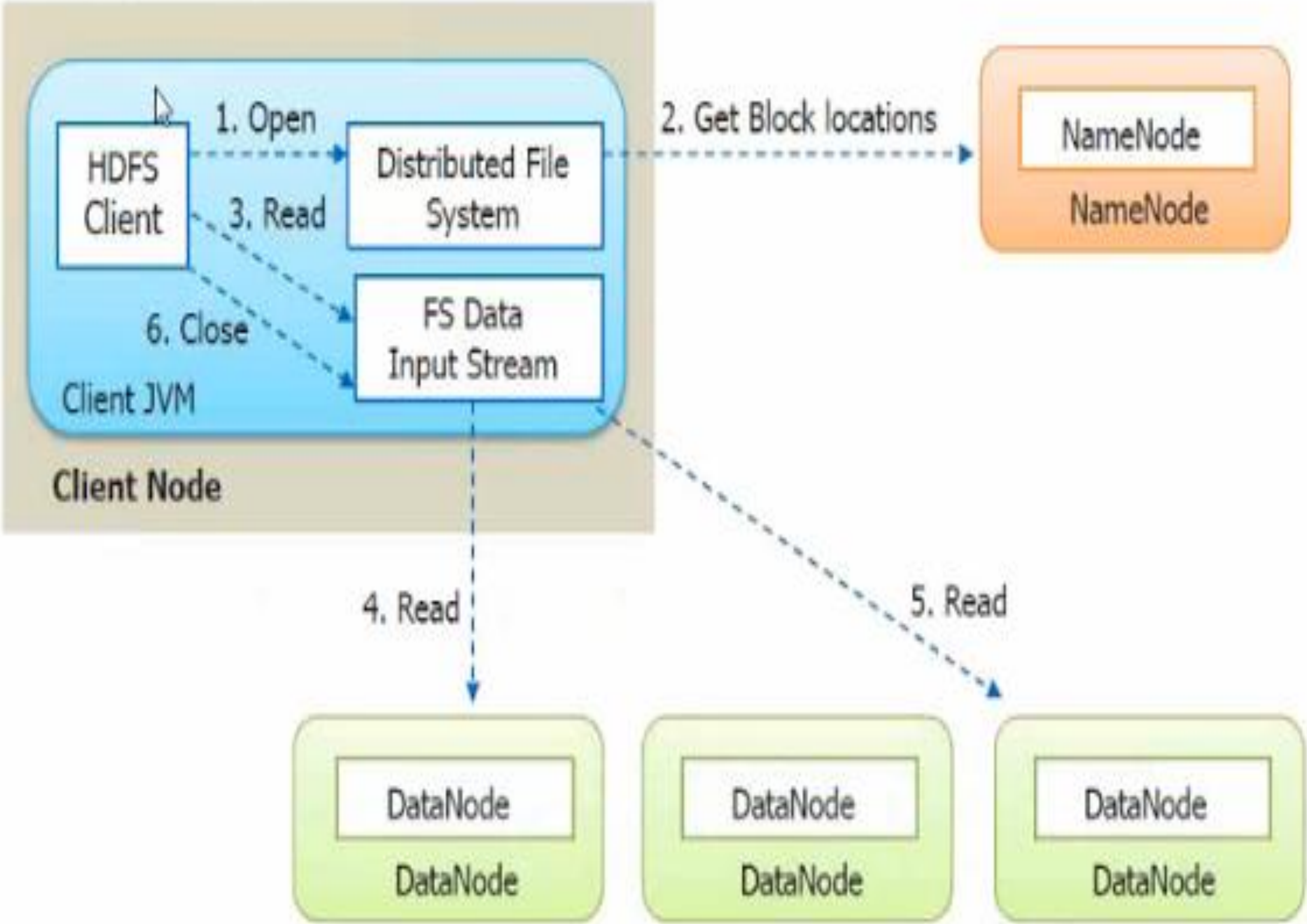
B1 (R1N1, R2N1, R2N2)
B2 (R1N1, R2N3, R2N4)

FSDataInputStream

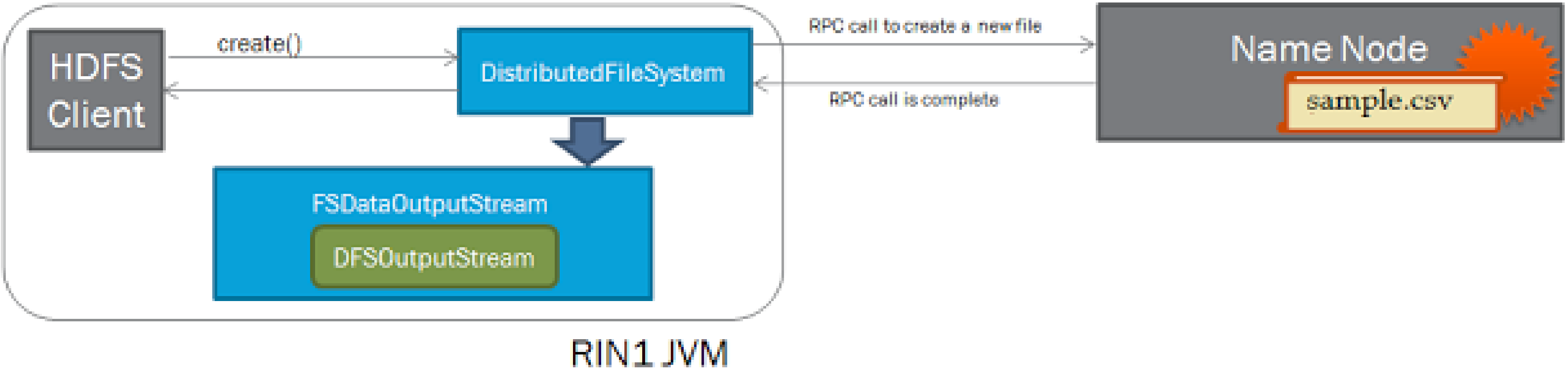
DFSInputStream

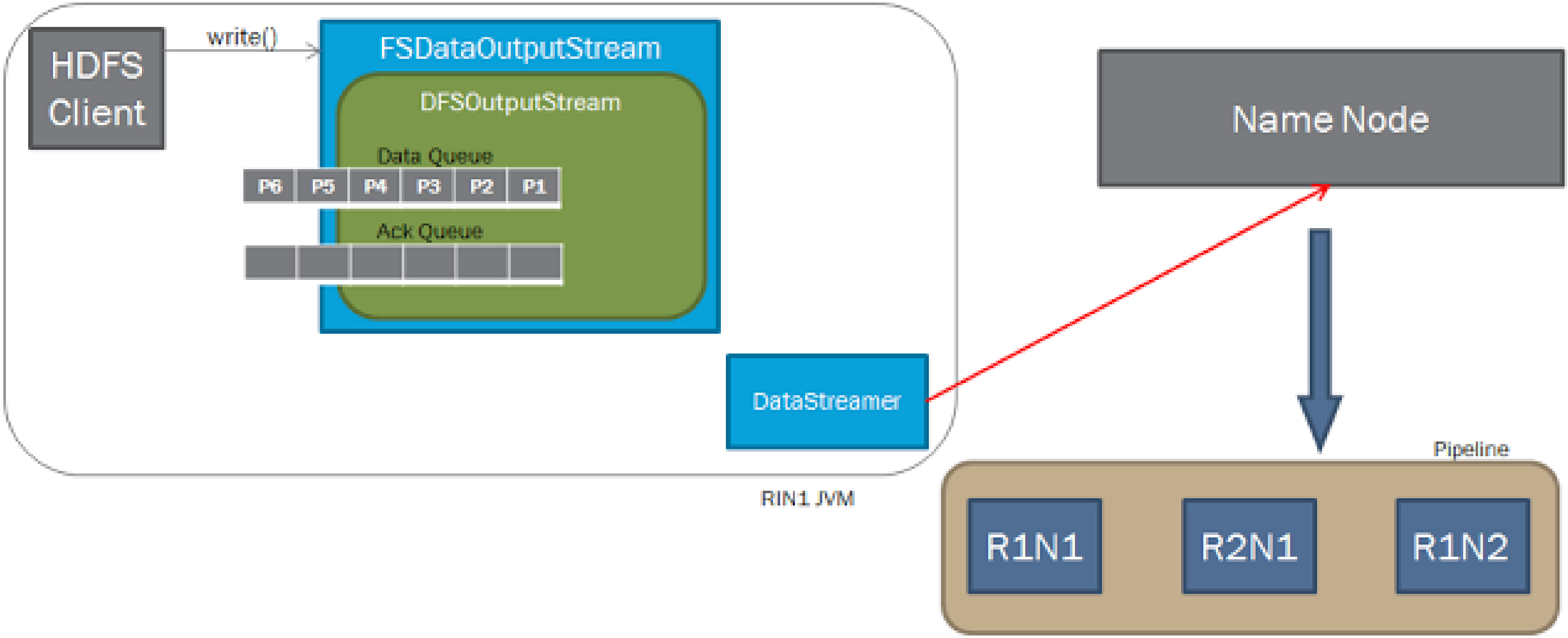
B1 (R1N1, R2N1, R2N2)
B2 (R1N1, R2N3, R2N4)

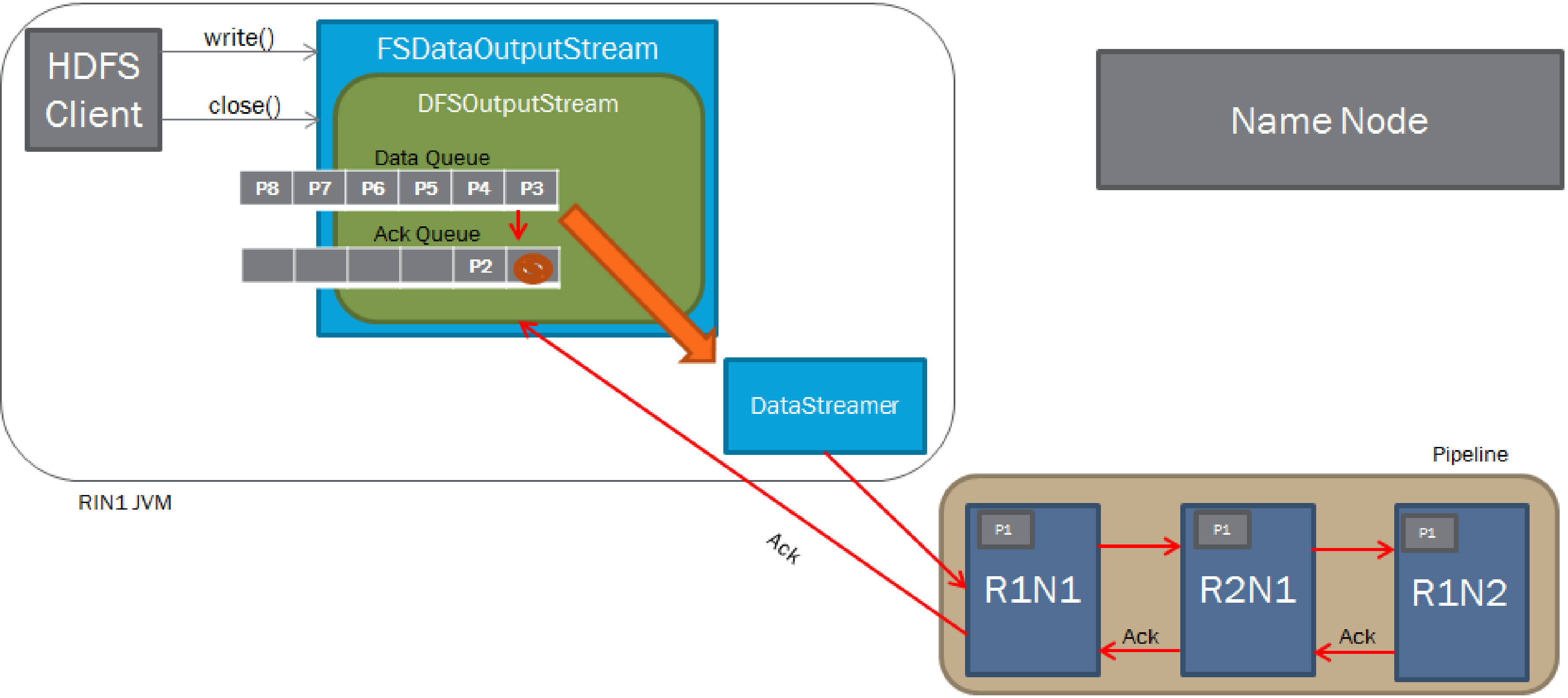
RIN2 JVM

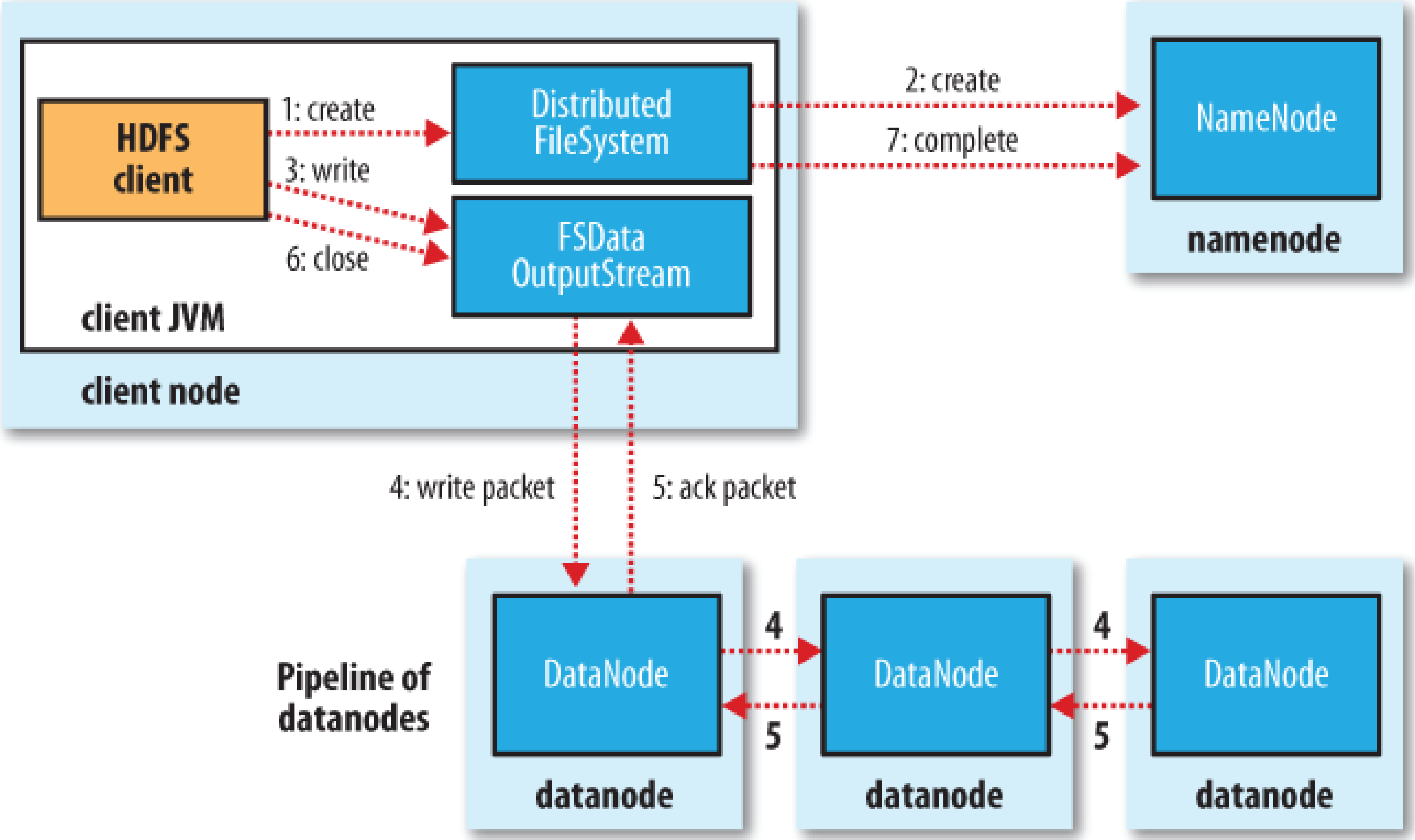

1. Open
2. Get Block locations
HDFS Client
Distributed File System
NameNode
NameNode
3. Read
FS Data Input Stream
6. Close
Client JVM
Client Node
4. Read
5. Read
DataNode
DataNode
DataNode
DataNode
DataNode
DataNode

Write


HDFS Client
create()
DistributedFileSystem
RPC call to create a new file
RPC call is complete
Name Node
sample.csv
FSDataOutputStream
DFSOutputStream
RIN1 JVM


HDFS Client
write()
FSDataOutputStream
DFSOutputStream
Data Queue
P6
P5
P4
P3
P2
P1
Ack Queue
DataStreamer
RIN1 JVM
Name Node
Pipeline
R1N1
R2N1
R1N2


HDFS Client
write()
close()
FSDataOutputStream
DFSOutputStream
Data Queue
P8
P7
P6
P5
P4
P3
Ack Queue
P2
DataStreamer
RIN1 JVM
Ack
Name Node
Pipeline
P1
R1N1
P1
R2N1
P1
R1N2
Ack
Ack

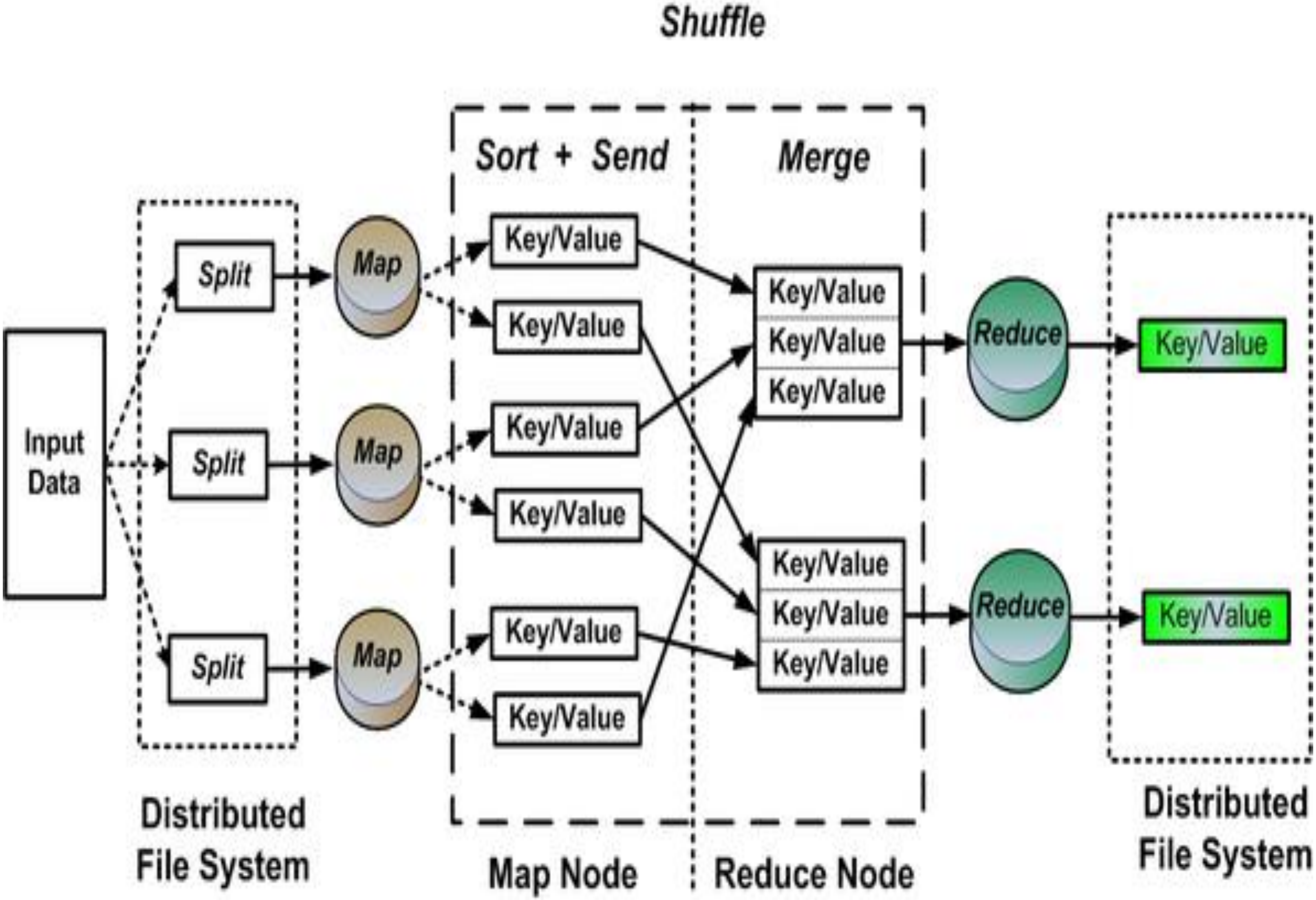

HDFS client
1: create
3: write
6: close
Distributed FileSystem
FSData OutputStream
client JVM
client node
2: create
7: complete
NameNode
namenode
4: write packet
5: ack packet
Pipeline of datanodes
DataNode
4
5
DataNode
4
5
DataNode
datanode
datanode
datanode

Mapreduce

- A parallel programming framework
- Simplifies data processing across massive data sets.
- A tool to handle both unstructured and semi-structured data

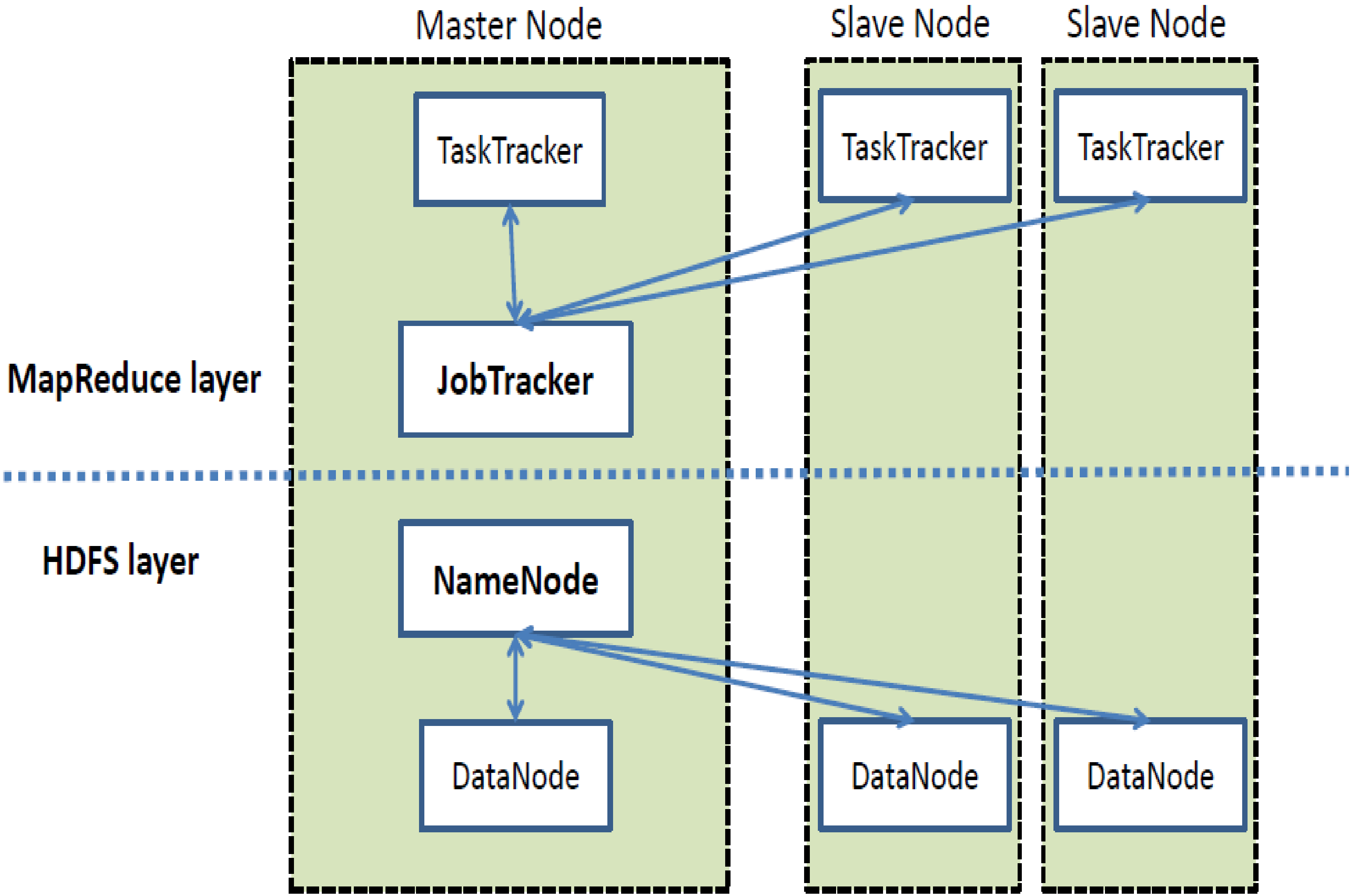
How Mapreduce works?

- Consists of 2 Primary processes
  - Map
  - Reduce
- Both the processes run in parallel
- Follows the master –worker approach

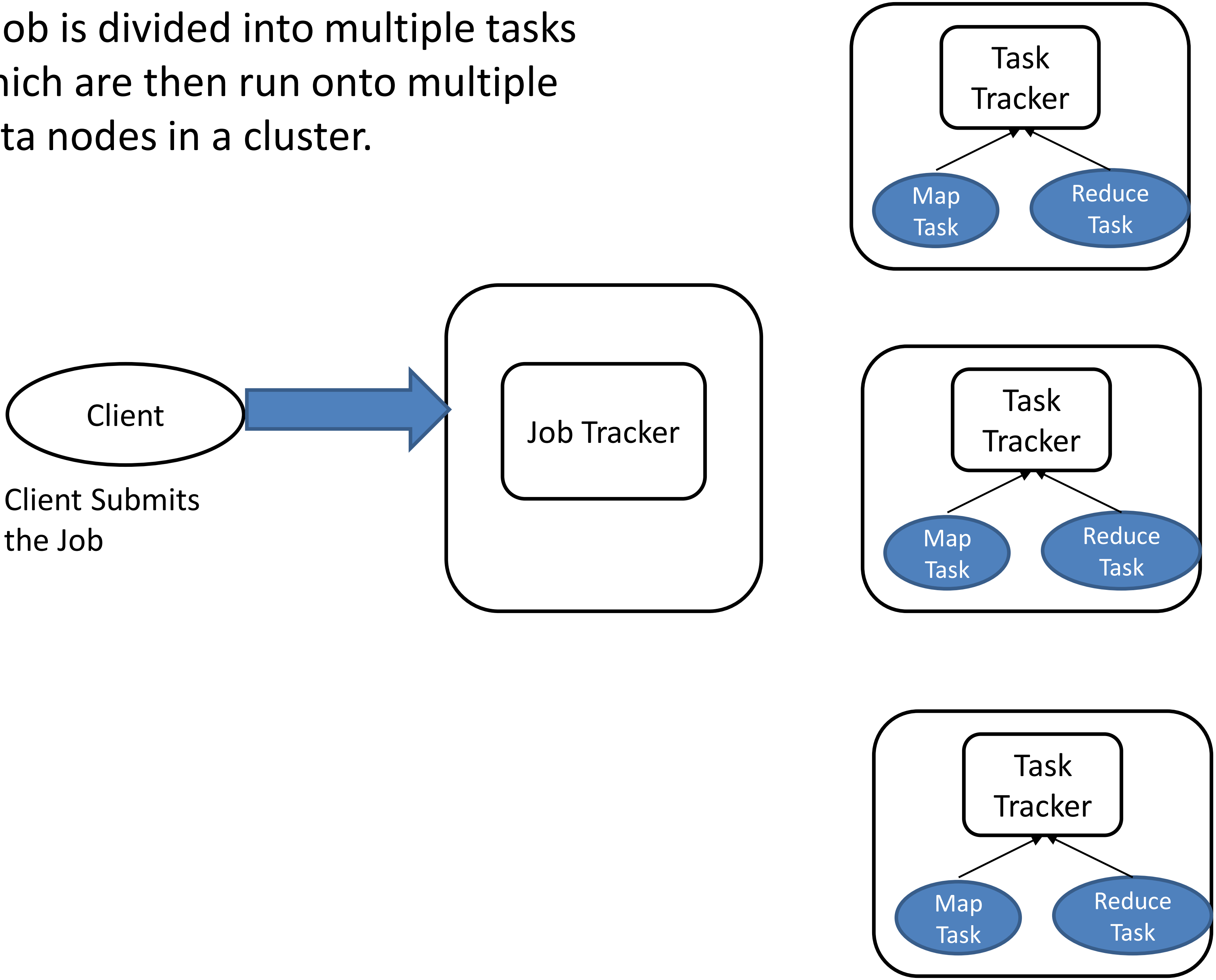

Shuffle
Sort + Send
Merge
Input Data
Split
Split
Split
Map
Map
Map
Key/Value
Key/Value
Key/Value
Key/Value
Key/Value
Key/Value
Key/Value
Key/Value
Key/Value
Key/Value
Key/Value
Key/Value
Reduce
Reduce
Key/Value
Key/Value
Distributed File System
Map Node
Reduce Node
Distributed File System

1. Split the input into multiple pieces of data
2. Create master and workers and execute processes remotely
3. Different map tasks simultaneously work and read pieces of data . And map functions extracts the required data in key, value pair format.
4. After map workers finish the job master instructs reducer to start their work. Reducers in turn contact the map workers to get the data.
5. The data received is sorted as per keys. This is called as shuffling .

6. After sorting ,the reduce function is called for every key .And writes the output to the file.
7. After all reducers complete their work the control is transferred to the master.

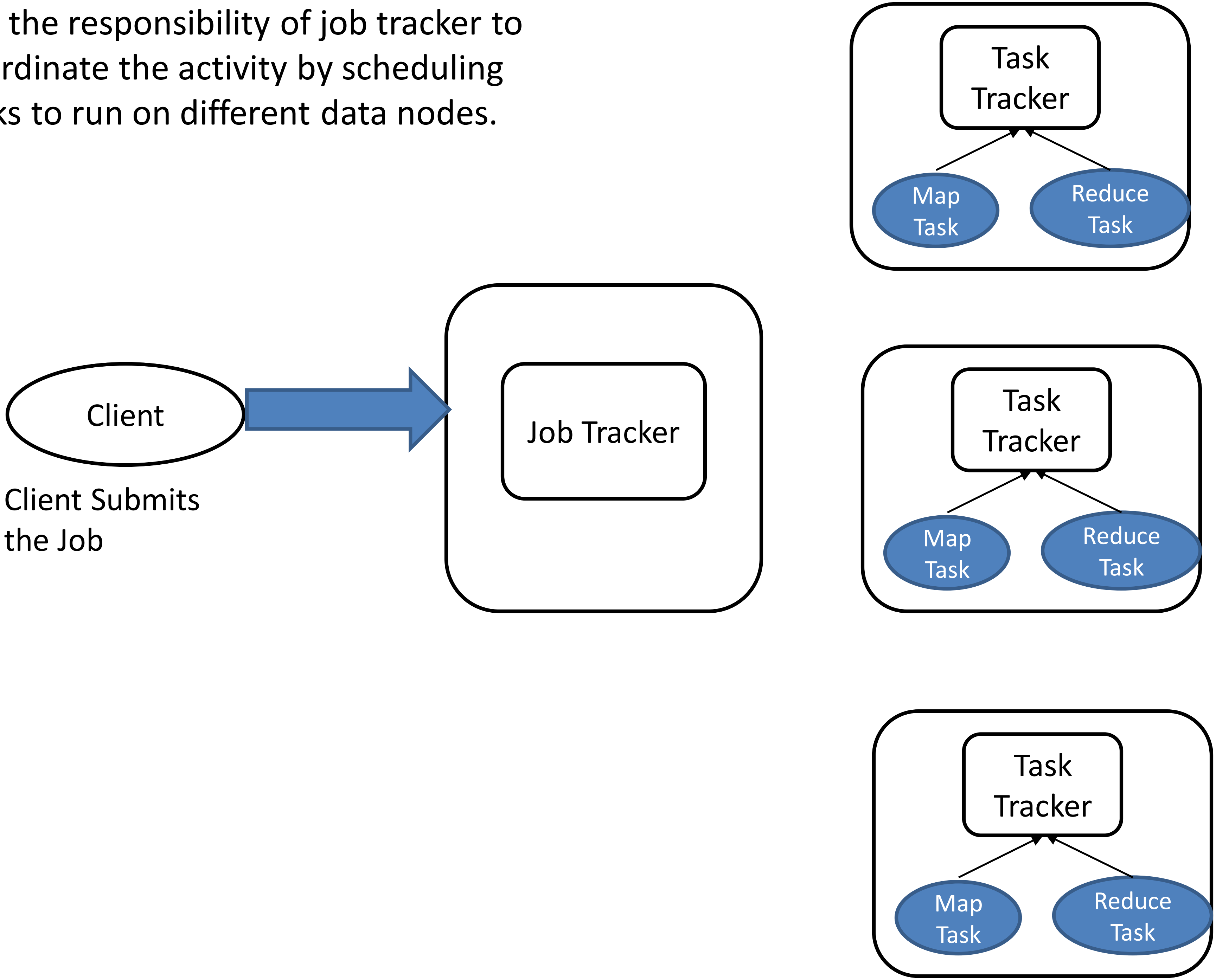

Master Node
Slave Node
Slave Node
TaskTracker
TaskTracker
TaskTracker
MapReduce layer
JobTracker
HDFS layer
NameNode
DataNode
DataNode
DataNode

A job is divided into multiple tasks which are then run onto multiple data nodes in a cluster.



Client Submits the Job

It is the responsibility of job tracker to coordinate the activity by scheduling tasks to run on different data nodes.



Client

Client Submits the Job

Job Tracker

Task Tracker

Map Task

Reduce Task

Task Tracker

Map Task

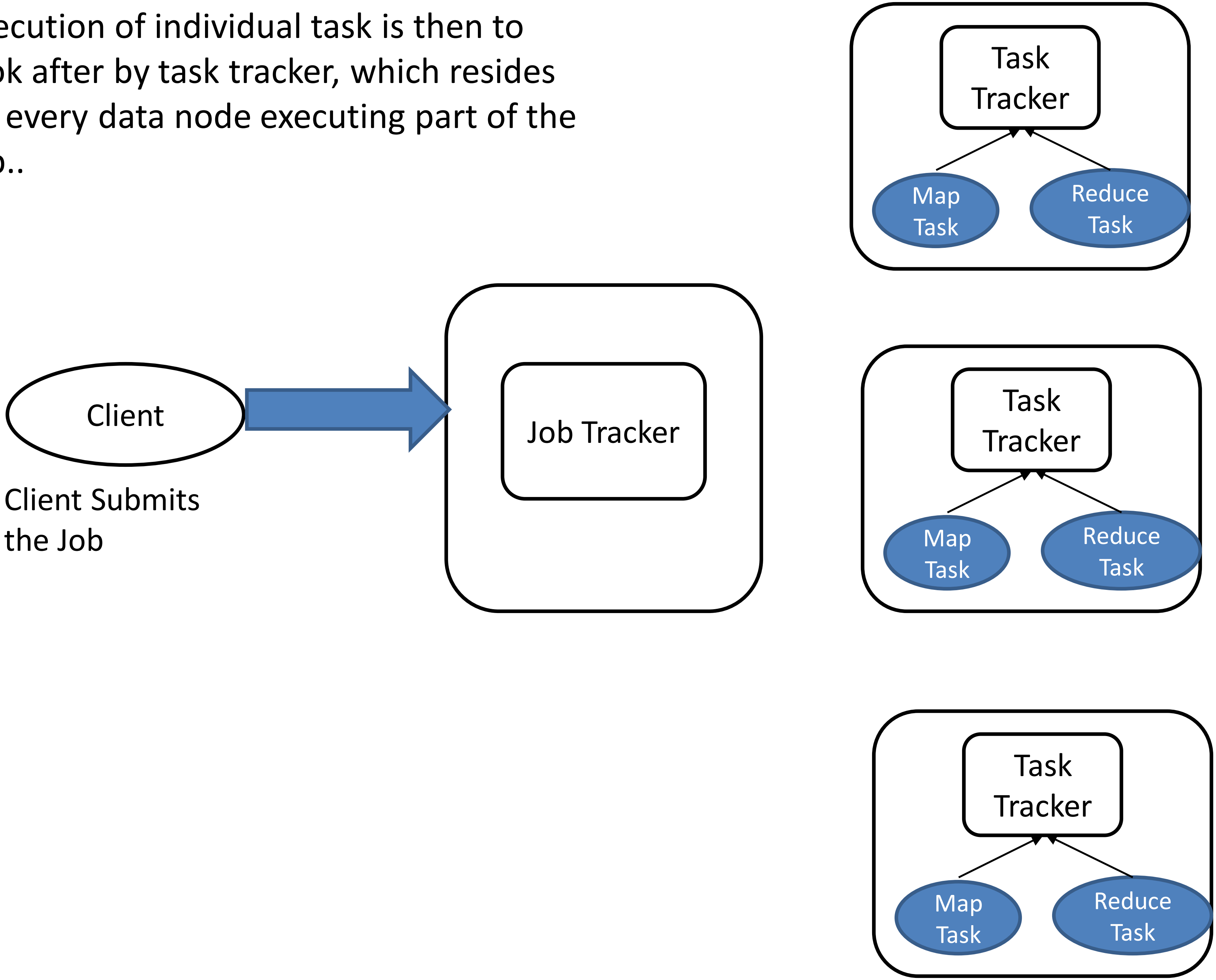
Reduce Task

Task Tracker

Map Task

Reduce Task

Execution of individual task is then to look after by task tracker, which resides on every data node executing part of the job..



Task Tracker

Map Task

Reduce Task

Client

Job Tracker

Task Tracker

Map Task

Reduce Task

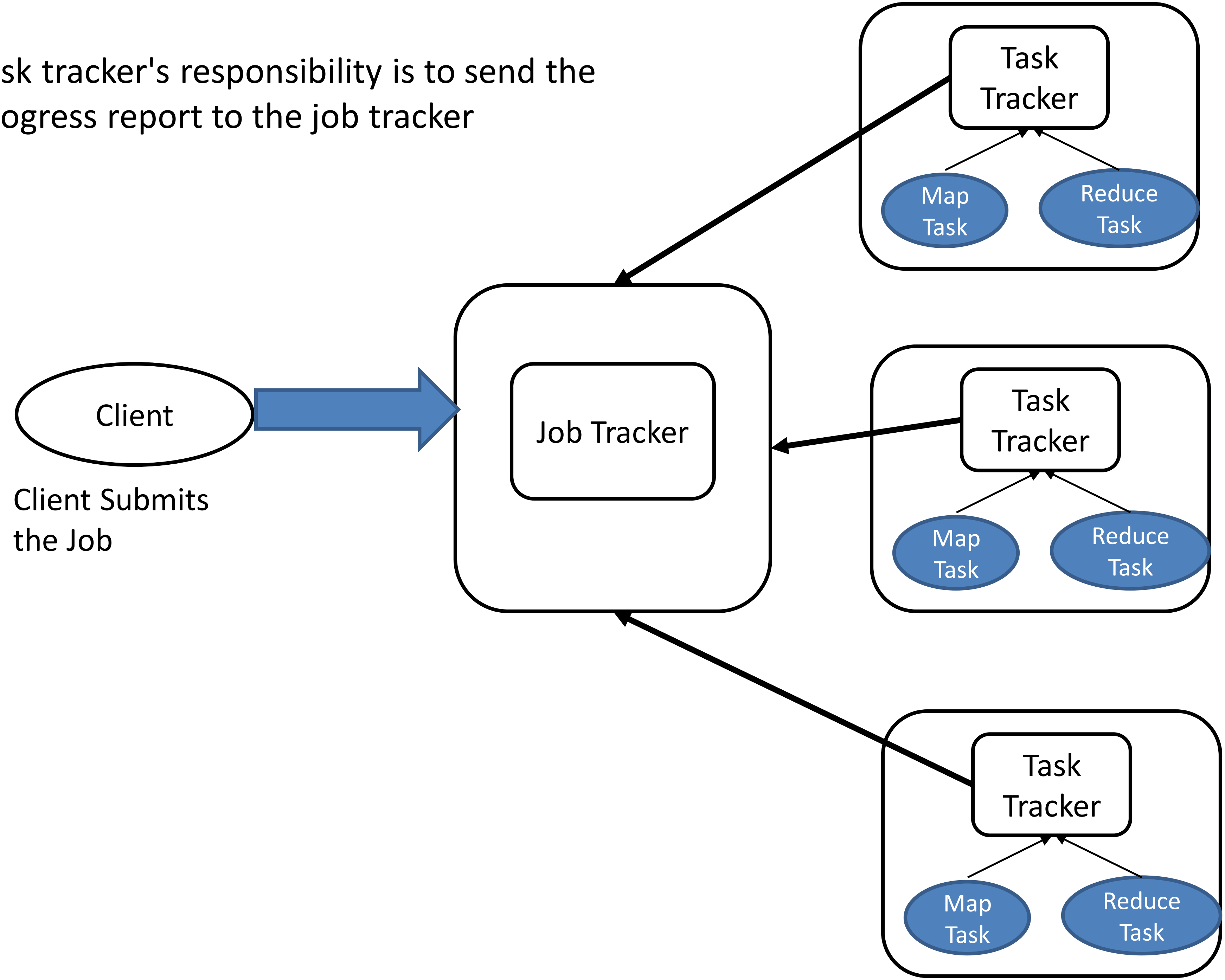
Task Tracker

Map Task

Reduce Task

Client Submits the Job

Task tracker's responsibility is to send the progress report to the job tracker



Client

Client Submits the Job

Job Tracker

Task Tracker

Map Task

Reduce Task

Task Tracker

Map Task

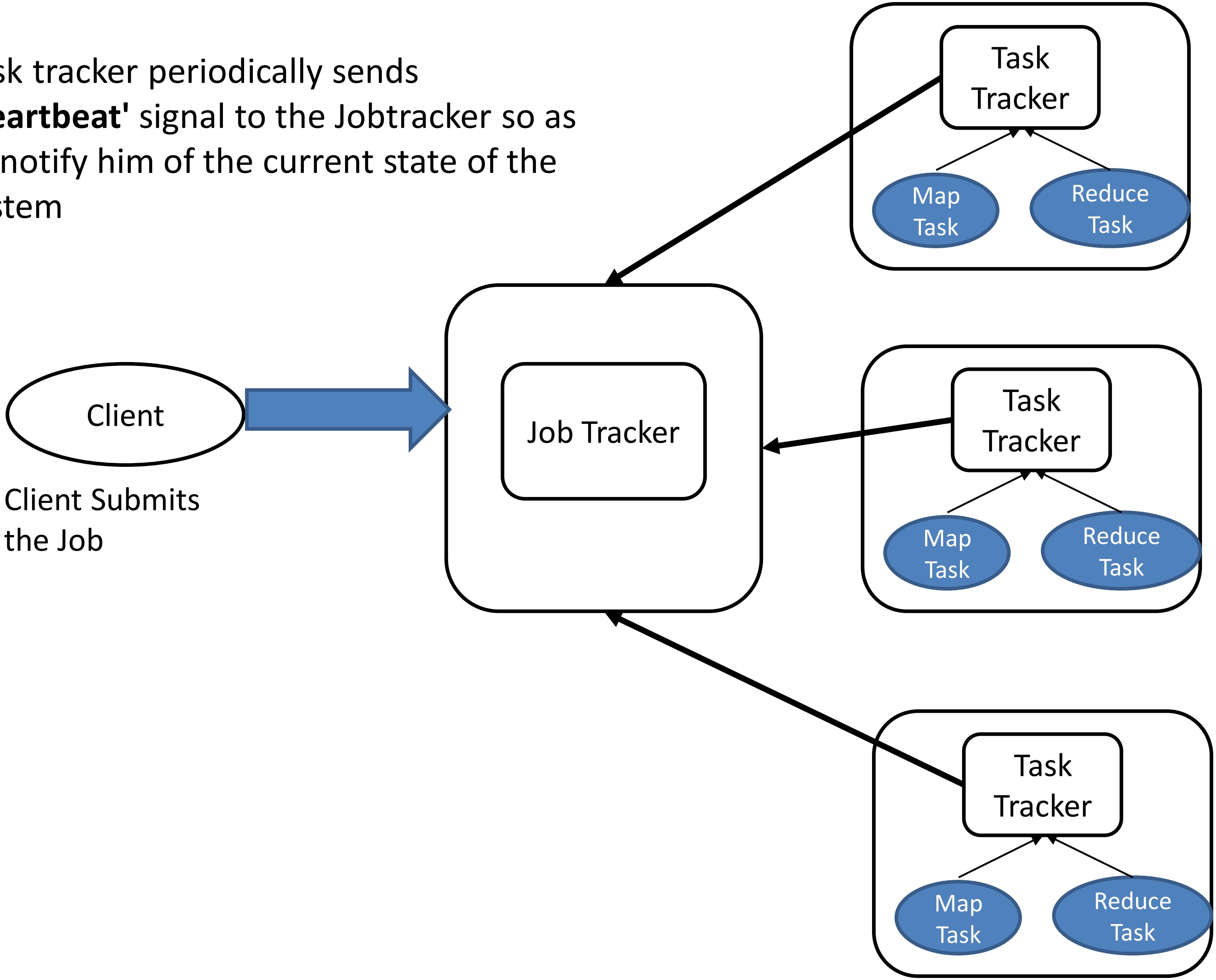
Reduce Task

Task Tracker

Map Task

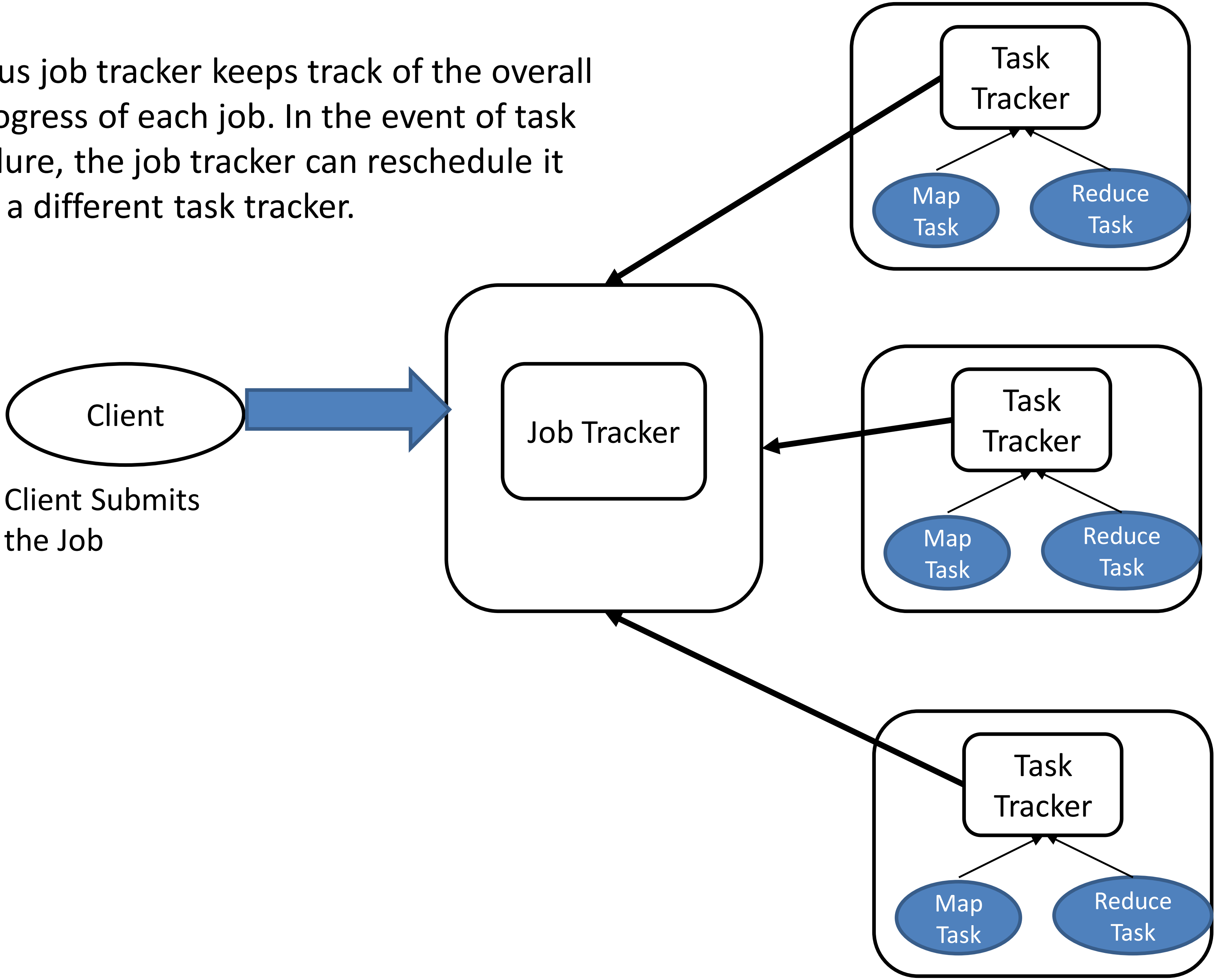
Reduce Task

Task tracker periodically sends **'heartbeat'** signal to the Jobtracker so as to notify him of the current state of the system



Client Submits the Job

Thus job tracker keeps track of the overall progress of each job. In the event of task failure, the job tracker can reschedule it on a different task tracker.



Client

Client Submits the Job

Job Tracker

Task Tracker

Map Task

Reduce Task

Task Tracker

Map Task

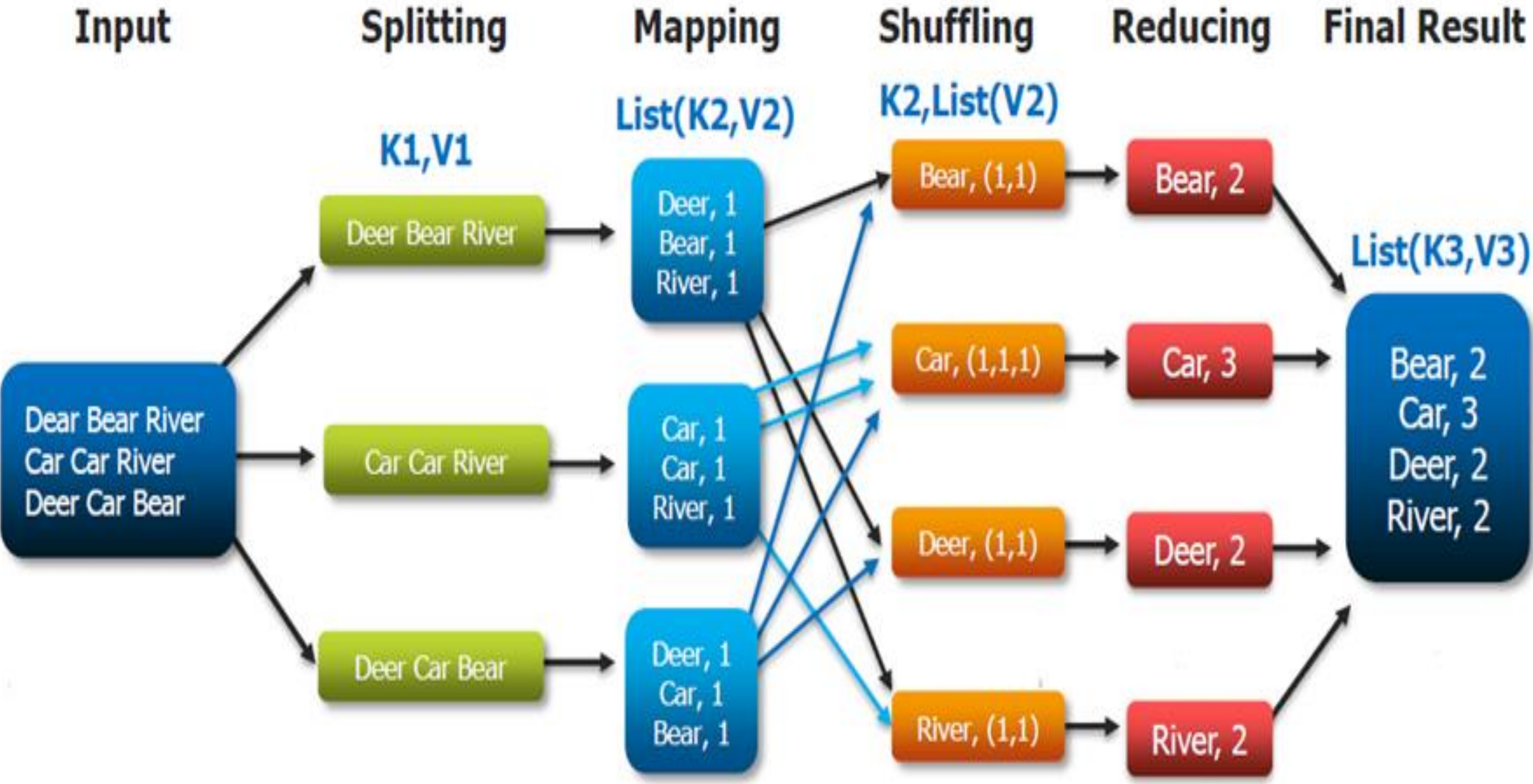
Reduce Task

Task Tracker

Map Task

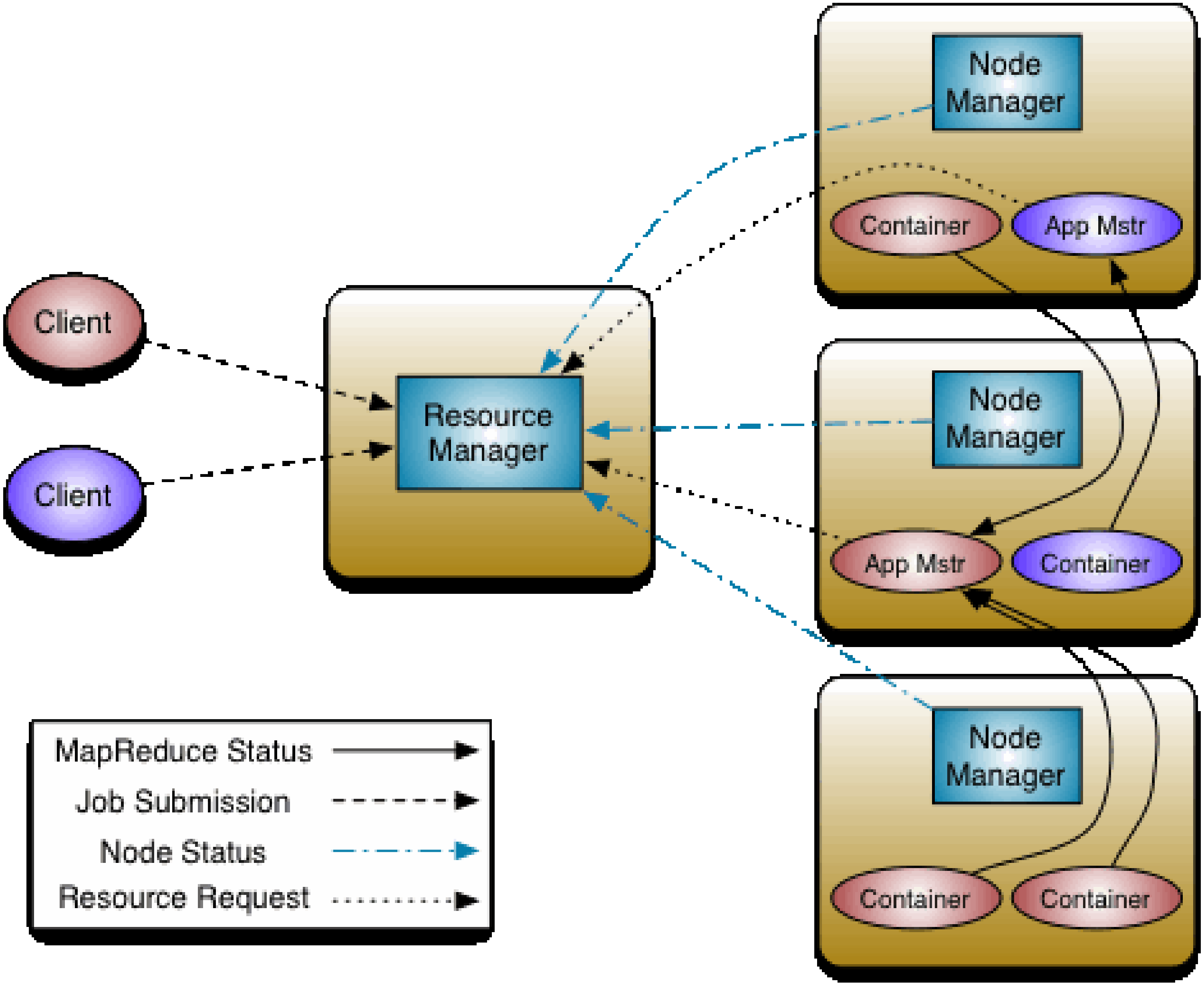
Reduce Task

The Overall MapReduce Word Count Process



YARN

- Yet Another Resource Negotiator
- The fundamental idea of YARN is to split up the functionalities of resource management and job scheduling/monitoring into separate daemons.
- The idea is to have a global ResourceManager (*RM*) and per-application ApplicationMaster (*AM*


Node
Manager
Container
App Mstr
Client
Resource
Manager
Client
Node
Manager
App Mstr
Container
Node
Manager
Container
Container
MapReduce Status
Job Submission
Node Status
Resource Request

- MRV1 vs Hadoop2.0
- the core component of Hadoop v2.0,

- YARN enabled the users to perform operations as per requirement by using a variety of tools like ***Spark*** for real-time processing, ***Hive*** for SQL, ***HBase*** for NoSQL and others.

Components of YARN

- **Resource Manager:** Runs on a master daemon and manages the resource allocation in the cluster.
- **Node Manager:** They run on the slave daemons and are responsible for the execution of a task on every single Data Node.
- **Application Master:** Manages the user job lifecycle and resource needs of individual applications. It works along with the Node Manager and monitors the execution of tasks.
- **Container:** Package of resources including RAM, CPU, Network, HDD etc on a single node.

- **Resource Manager**
- The ultimate authority in resource allocation.
- passes parts of requests to corresponding node managers
- the arbitrator of the cluster resources and decides the allocation of the available resources for competing applications.
- Optimizes the cluster utilization.
- It has two major components:
  - a) Scheduler b) Application Manager

Scheduler

- responsible for allocating resources to the various running applications
- It is called a pure scheduler in ResourceManager,
- Performs scheduling based on the resource requirements of the applications.

Node Manager

- takes care of individual nodes in a Hadoop cluster and manages user jobs and workflow on the given node.
- registers with the Resource Manager and sends heartbeats with the health status of the node.
- manages application containers assigned to it by the resource manager.
- creates the requested container process and starts it.
- Monitors resource usage (memory, CPU) of individual containers.
- Performs Log management.
- kills the container as directed by the Resource Manager.

Application Master

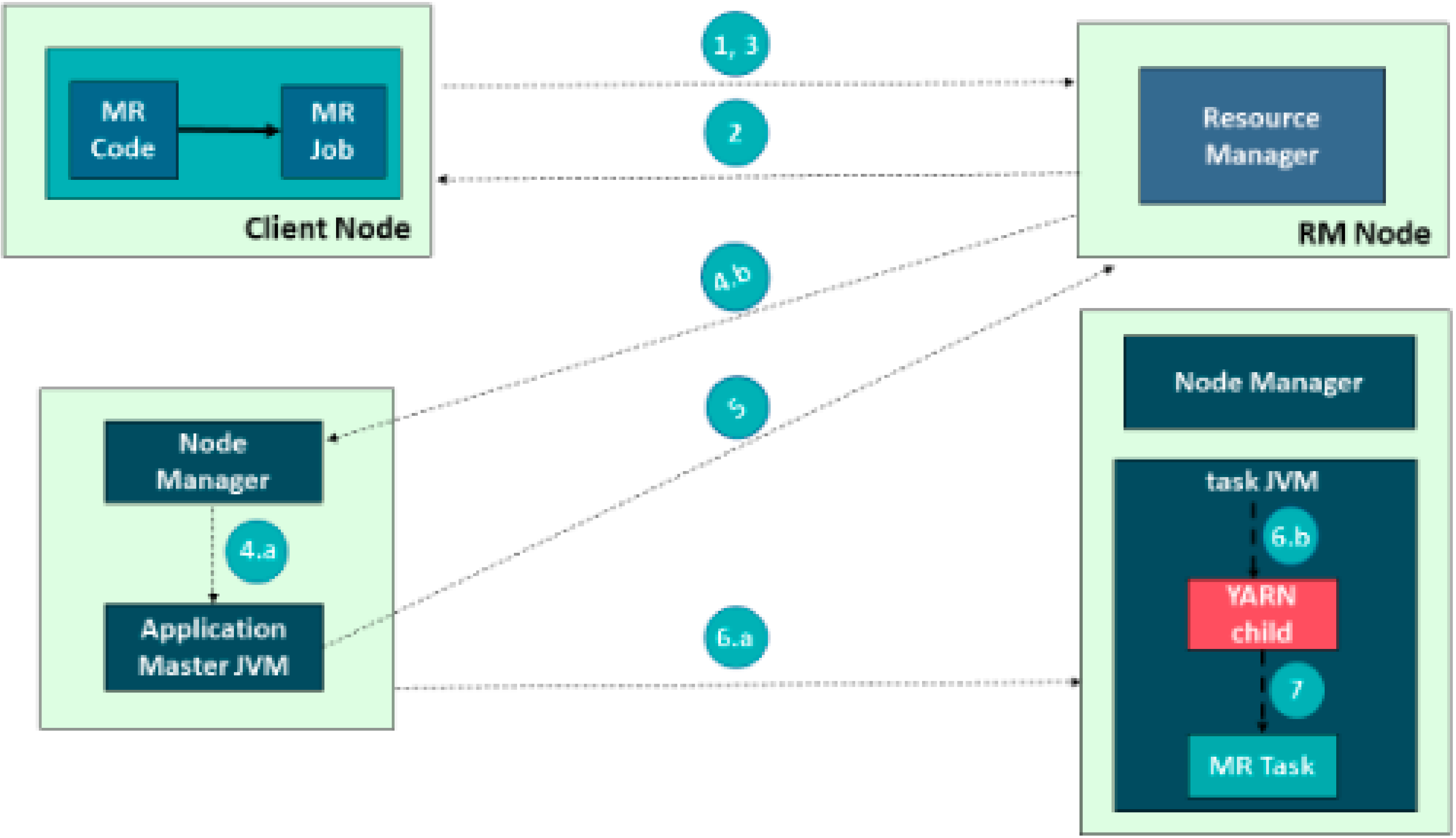
- An application is a single job submitted to the framework. Each such application has a unique Application Master associated with it which is a framework specific entity.
- the process that coordinates an application’s execution in the cluster and also manages faults.
- Negotiates resources from the Resource Manager and work with the Node Manager to execute and monitor the component tasks.
- tracking their status and monitoring progress.
- periodically sends heartbeats to the Resource Manager

Container

- collection of physical resources such as RAM, CPU cores, and disks on a single node.
- YARN containers are managed by a container launch context which is container life-cycle(CLC). This record contains a map of environment variables, dependencies stored in a remotely accessible storage, security tokens, payload for Node Manager services and the command necessary to create the process.
- It grants rights to an application to use a specific amount of resources (memory, CPU etc.) on a specific host.

Application Manager

- It is responsible for accepting job submissions.
- Negotiates the first container from the Resource Manager for executing the application specific Application Master.
- Manages running the Application Masters in a cluster and provides service for restarting the Application Master container on failure.

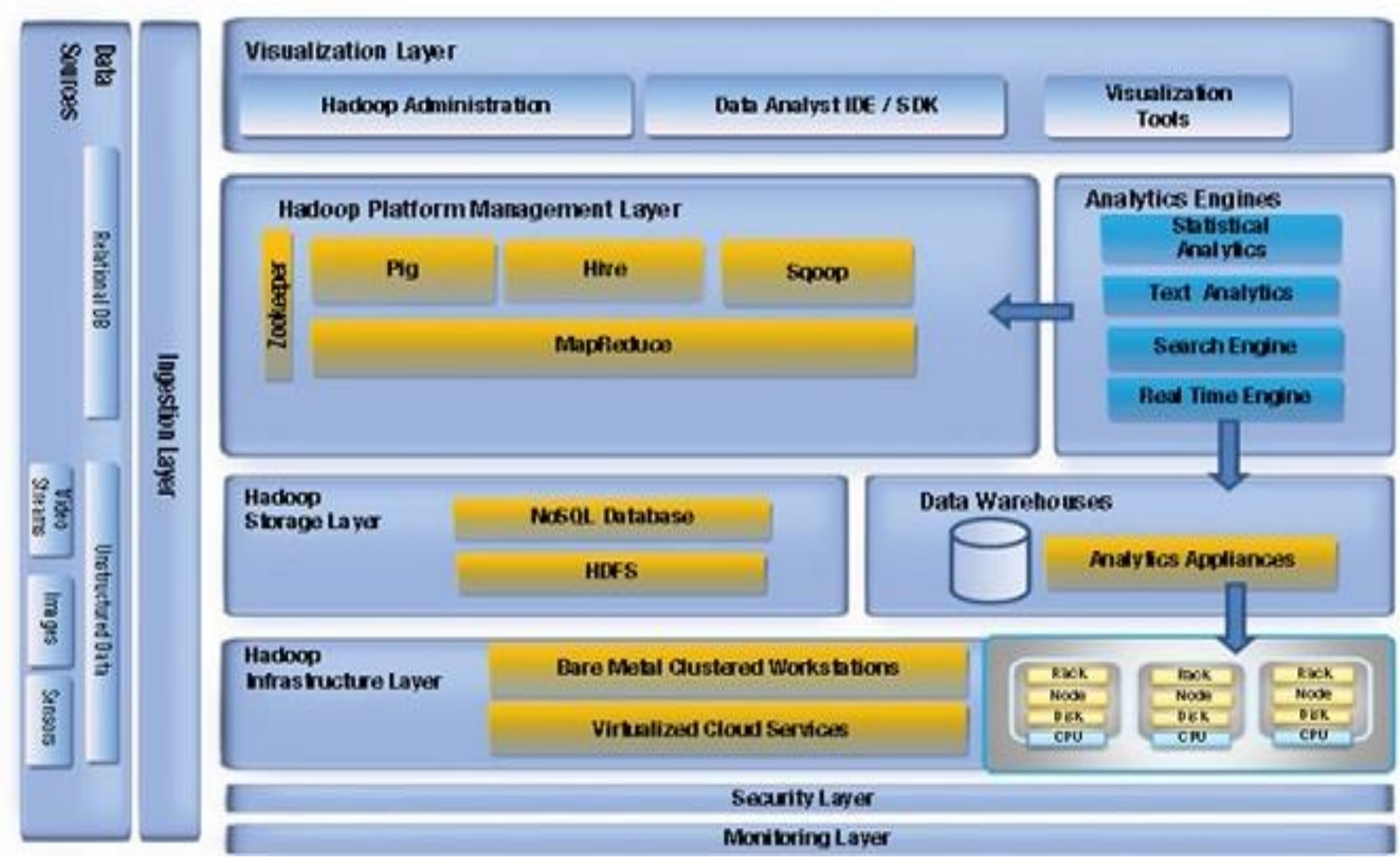

1, 3
2
MR Code
MR Job
Client Node
Resource Manager
RM Node
4.b
5
Node Manager
4.a
Application Master JVM
6.a
Node Manager
task JVM
6.b
YARN child
7
MR Task

Hadoop

- Hadoop was started by Doug Cutting and Mike Cafarella in 2002..
- **Apache Nutch :** It is an open source web crawler software project.
- they were dealing with big data.
- Doug Cutting gave named his project Hadoop after his son's toy elephant.

2003	Google released the paper, Google File System (GFS).
2004	Google released a white paper on Map Reduce.
2006	•Hadoop introduced. •Hadoop 0.1.0 released. •Yahoo deploys 300 machines and within this year reaches 600 machines.
2007	•Yahoo runs 2 clusters of 1000 machines. •Hadoop includes HBase.
2008	•YARN JIRA opened •Hadoop becomes the fastest system to sort 1 terabyte of data on a 900 node cluster within 209 seconds. •Yahoo clusters loaded with 10 terabytes per day. •Cloudera was founded as a Hadoop distributor.
2009	•Yahoo runs 17 clusters of 24,000 machines. •Hadoop becomes capable enough to sort a petabyte. •MapReduce and HDFS become separate subproject.
2010	•Hadoop added the support for Kerberos. •Hadoop operates 4,000 nodes with 40 petabytes. •Apache Hive and Pig released.
2011	•Apache Zookeeper released. •Yahoo has 42,000 Hadoop nodes and hundreds of petabytes of storage.
2012	Apache Hadoop 1.0 version released.
2013	Apache Hadoop 2.2 version released.
2014	Apache Hadoop 2.6 version released.

Big Data Stack



Data Sources

Relational DB

Video Streams

Images

Sensors

Unstructured Data

Ingestion Layer

Visualization Layer

Hadoop Administration

Data Analyst IDE / SDK

Visualization Tools

Hadoop Platform Management Layer

Zookeeper

Pig

Hive

Sqoop

MapReduce

Analytics Engines

Statistical Analytics

Text Analytics

Search Engine

Real Time Engine

Hadoop Storage Layer

NoSQL Database

HDFS

Data Warehouses

Analytics Appliances

Hadoop Infrastructure Layer

Bare Metal Clustered Workstations

Virtualized Cloud Services

Rack

Node

Disk

CPU

Rack

Node

Disk

CPU

Rack

Node

Disk

CPU

Security Layer

Monitoring Layer

Big data stack Layers

- Data Source layer
- Ingestion layer
- Storage layer
- Physical infrastructure layer
- Platform management layer
- Security Layer
- Monitoring Layer
- Analytics Engine
- Visualization Layer
- Big Data Applications

- Data Source Layer
  - Absorb and integrate the data coming form various sources at varying velocity and in different formats.

- Ingestion Layer
    - Absorb the huge inflow and sort it into different categories
    - Separates noise
    - Validates ,cleanses ,transforms reduces and integrates unstructured data

- Storage layer
  - HDFS
    - Blocks of Files
    - Write once read many times model
  - NoSQL Databases
    - Key –value pair
    - Document based
    - Graph based
    - Column oriented

- Physical Infrastructure Layer
  - Hardware and Network requirements
- Platform Management Layer
  - Provides tools and query languages for the accessing data
  - Mapreduce ,Hive ,Pig ,etc

- Security Layer
  - Authentication of node
  - Enable file layer encryption
  - Maintain logs of communication
  - Secure communication between nodes by using SSL

- Monitoring Layer
  - Monitoring systems
  - Ganglia, Nagios

- Analytics Engine
  - Analyze huge amount of unstructured data
  - NLP, Text mining ,Machine Learning
  - Search Engines, Real time engines

- Visualization Layer
  - Data Interpretation
  - Tableau ,Clickview ,Sportfire,MapR

NoSQL

Classical relation database follow the ACID Rules

- **Atomic**
- **Consistent**
- **Isolated**
- **Durable**

Distributed Systems

- **Reliability (fault tolerance)**
- **Scalability**
- **Sharing of Resources**
- **Flexibility**
- **Speed**
- **Open system**
- **Performance**

Scalability

- scalability is the ability of a system to expand to meet your business needs
- Two ways of scaling
  - Vertical
  - Horizontal

Vertical scaling

- add resources within the same logical unit to increase capacity
  - CPUs to an existing server
  - increase memory in the system
  - expanding storage by adding hard drive.

Horizontal scaling

- add more nodes to a system, such as adding a new computer to a distributed software application
- "scaling out"
  - to add more nodes to a system and distribute the load over those nodes.

NoSQL

- NoSQL is a non-relational database management systems
- Not Only SQL
- designed for distributed data stores where very large scale of data storing needs

Features

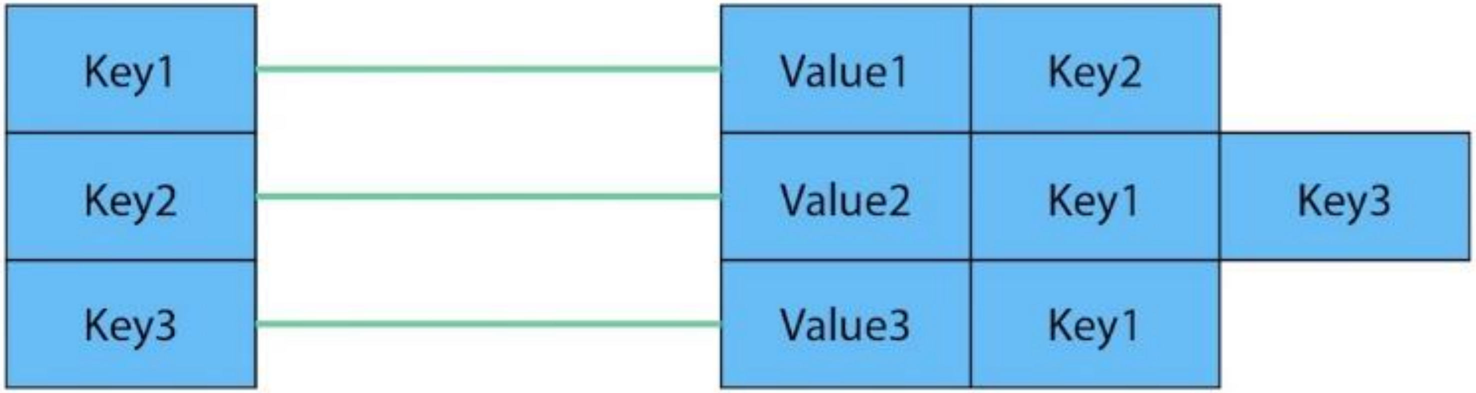
- Non-relational data model.
- Runs well on clusters.
- Mostly open-source.
- Built for the new generation Web applications.
- Is schema-less.

NOSQL

- Most popular Types
  - Key-value stores
  - Column family stores
  - Document databases
  - Graph databases

Key-value store

- Data is represented as collection of key-value pair
- These databases store the data as a hash table with a unique key and a pointer to a particular item of data.
- used whenever the data would be queried by precise parameters and needs to be retrieved really fast.



Key1

Key2

Key3

Value1 Key2

Value2 Key1 Key3

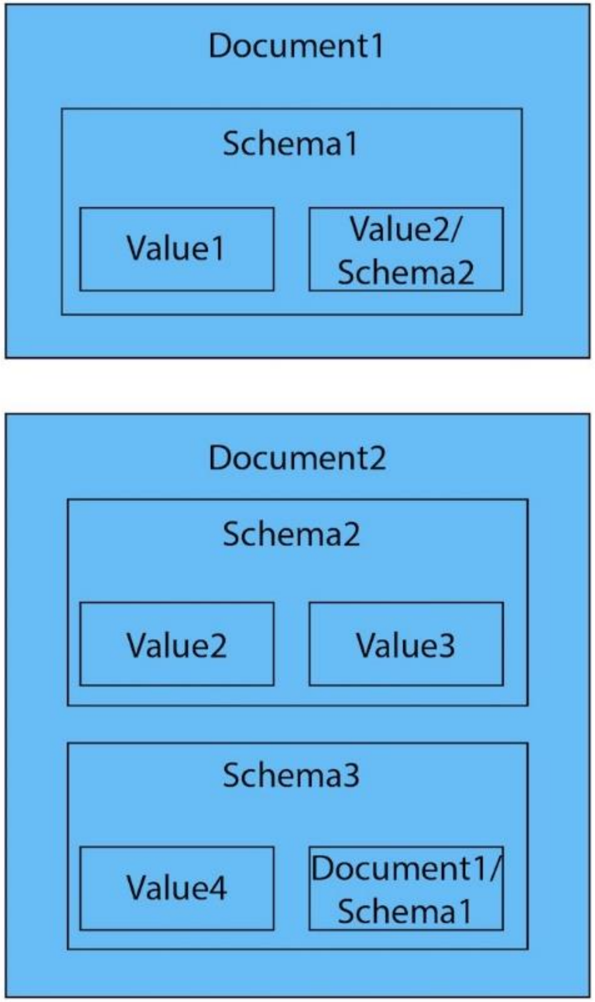
Value3 Key1

- key-value pair the key is represented by an arbitrary string such as a filename, URI or hash.
- The value can be any kind of data like an image, user preference file or document.
- Key-value stores have no query language

- well suited to storing things like
  - user profiles
  - session info on a website,
  - telecom directories
  - shopping cart contents on e-commerce sites, and more.
  - Examples of key-store database management systems include:
    - Redis
    - Oracle NoSQL Database
    - Voldemorte
    - Aerospike
    - Oracle Berkeley DB

Document store/ Document database

- The document databases are typically organized in form of a collection of documents, where one domain entity forms one document.
- It is similar to a key-value database in that it uses a key-value approach. The difference is that, the value in a document store database consists of semi-structured data.
- The major advantage of the document-oriented concept; every document in the database can have a different format.



Document1

Schema1

Value1

Value2/ Schema2

Document2

Schema2

Value2

Value3

Schema3

Value4

Document1/ Schema1

- Thus document stores typically work best with semistructured data or document formats containing some meta data e.g. xml, json, etc.
- They are ideal for content management systems, blogging platforms, and other web applications.
- well suited for user generated content such as blog comments, chat sessions, tweets, ratings, etc.
- Document stores are also great for providing real time analytics and other reporting features.
- Here are examples of document store DBMSs.
  - MongoDB
  - DocumentDB
  - CouchDB
  - MarkLogic
  - OrientDB

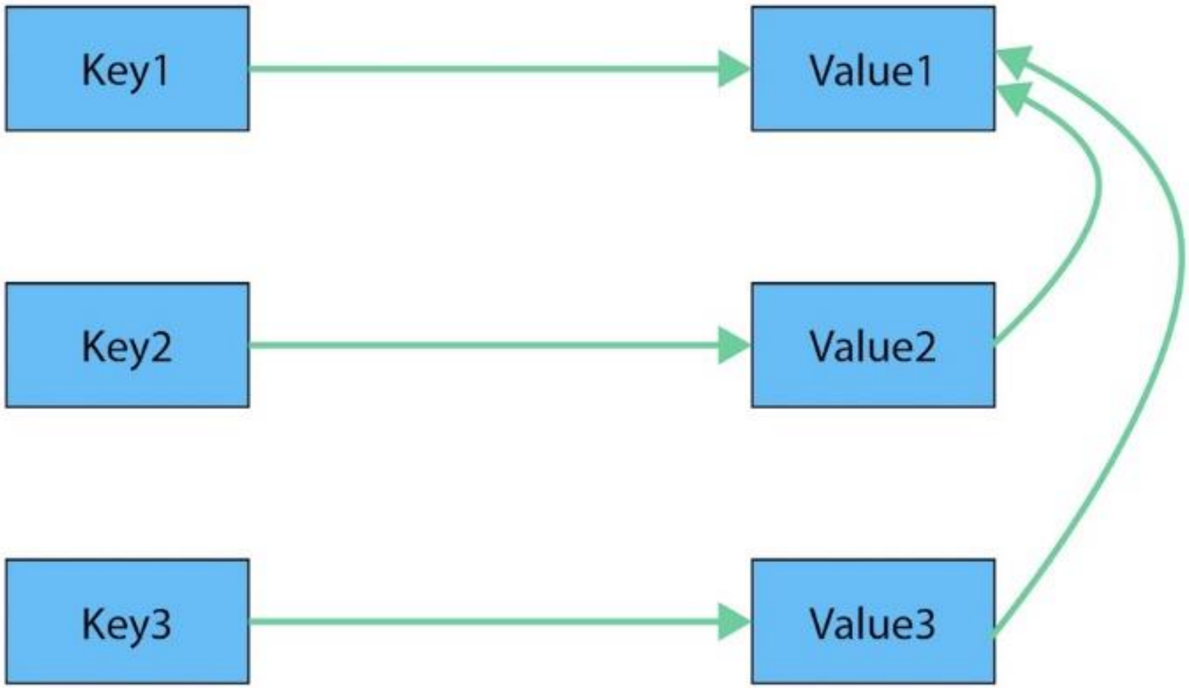
Column Store

- database that stores data using a column oriented model.
- instead of storing data in rows,
  these databases are designed for storing data tables as sections of columns of data
- Examples : HBase, BigTable and HyperTable.

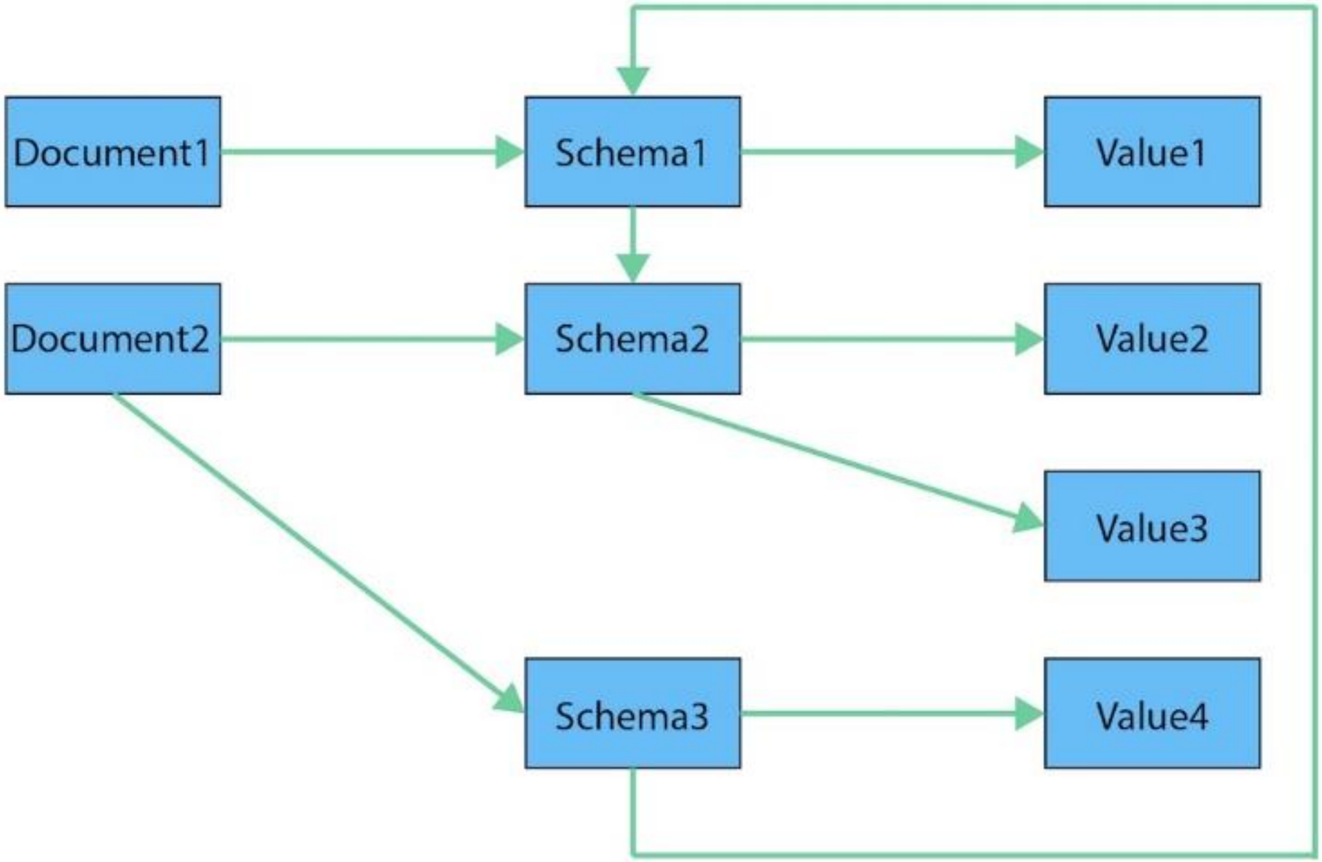
Graph database

- Based on graph theory, these databases are designed for data whose relations are well represented as a graph and has elements which are interconnected, with an undetermined number of relations between them.
- Examples include: Neo4j

Key Value as Graph



Document as Graph

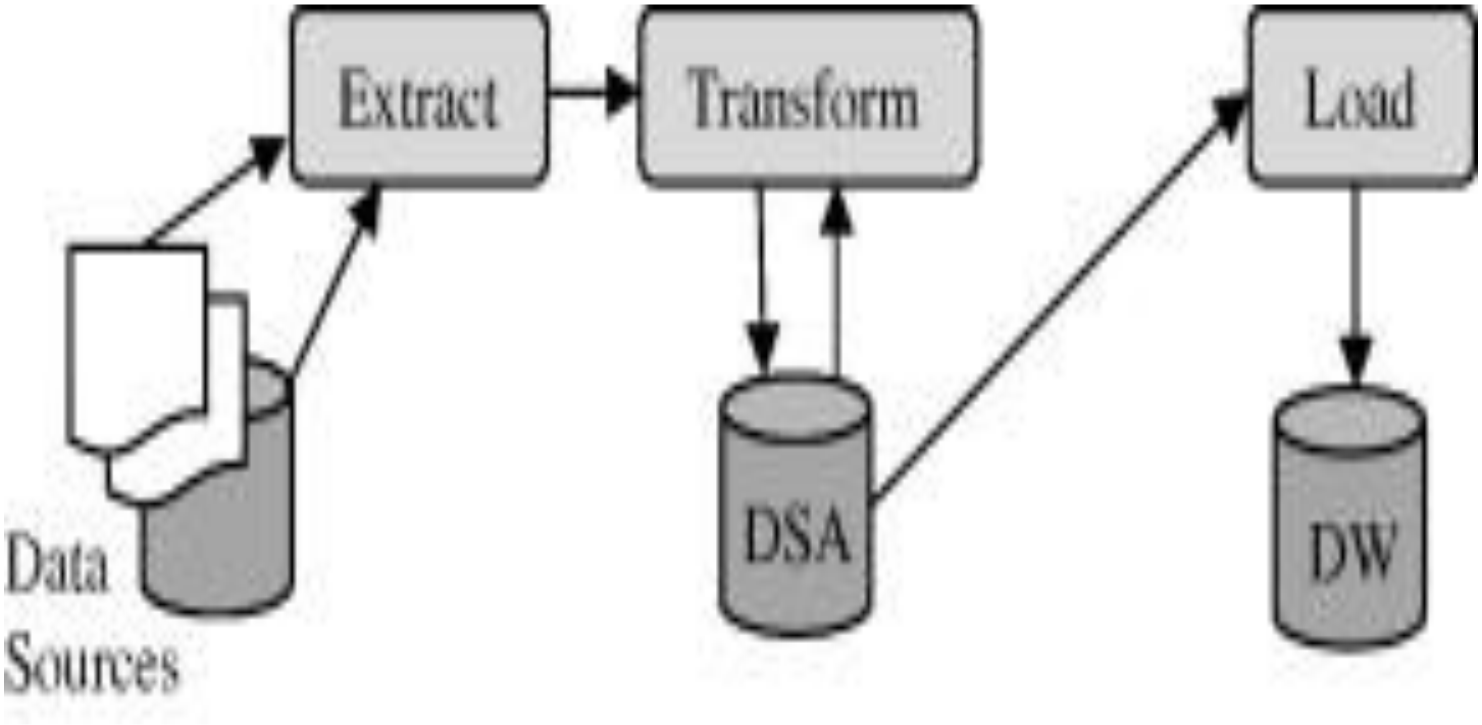


ETL

- ETL : Extract Transform and Load
- ETL is the process used to turn raw data into information that can be used for actionable BI
- Allows to gather data from different sources and consolidate into single ,centralized location
- Collects and refines different types of data

ETL

- Extract , Transform and Load



Extract

- Data extraction is collecting different types of data from variety of sources
- Locates and identifies data from one or more sources
- Identifies relevant data
- Prepares it fro Processing or Transoformation

Extract

- The Extract step covers the data extraction from the source system and makes it accessible for further processing.
- The main objective of the extract step is to retrieve all the required data from the source system with as little resources as possible.
- The extract step should be designed in a way that it does not negatively affect the source system in terms or performance, response time or any kind of locking.

Extract

- Raw data can be extracted from
  - Existing Databases
  - Cloud
  - Sales and marking applications
  - Mobile devices and apps
  - CRM Systems
  - Data Warehouses

Transform

- Data transformation is the process of converting data from one format to another
- Rules are applied to ensure data quality and accessibility

Transform

- The transform step applies a set of rules to transform the data from the source to the target.
- Converts any measured data to the same dimension (i.e. conformed dimension) using the same units so that they can later be joined.
- Joins data from several sources, generating aggregates, generating surrogate keys, sorting, deriving new calculated values, and applying advanced validation rules.

Transform

- Transformation includes
  - Cleansing
    - Inconsistencies and missing values are resolved
  - Standardization
    - Formatting rules are applied to data set
  - Deduplication
    - Redundant data is discarded
  - Verification
    - Unused data is removed
  - Sorting
    - Data is organized according to type
  - Other Tasks
    - Additional rules are applied to improve data quality

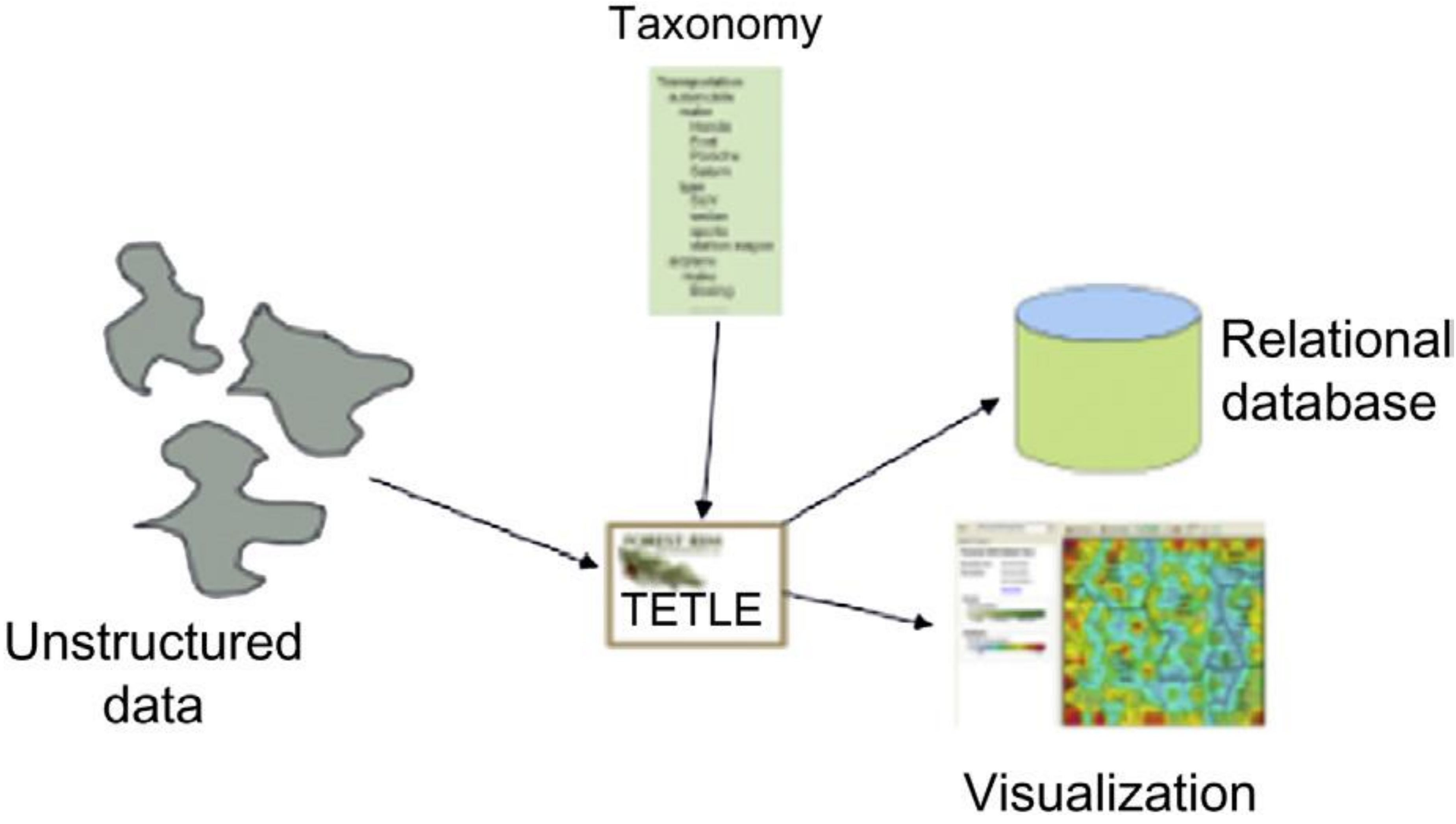
Clean

- The cleaning step is one of the most important as it ensures the quality of the data in the data warehouse. Cleaning should perform basic data unification rules, such as:
- Making identifiers unique (sex categories Male/Female/Unknown, M/F/null, Man/Woman/Not Available are translated to standard Male/Female/Unknown)
- Convert null values into standardized Not Available/Not Provided value
- Convert phone numbers, ZIP codes to a standardized form
- Validate address fields, convert them into proper naming, e.g. Street/St/St./Str./Str
- Validate address fields against each other (State/Country, City/State, City/ZIP code, City/Street).

Load

- Final Step in ETL process
- Load the newly transformed data into new destination
- Full Load
- Incremental Loading

Textual ETL


Taxonomy
Transportation
automobile
make
Honda
Ford
Porsche
Saturn
type
SUV
sedan
sports
station wagon
airplane
make
Boeing
Unstructured data
FOREST RIM
TETLE
Relational database
Visualization

Textual ETL

Components

- *Textual ETL rules engine. This is the core processing engine to parse large unstructured data* and extract value from the parsing for integration.

  The rules engine executes a series of data processing steps and processes the data.

  The following are the algorithms that are incorporated into the rules engine:

  Classification

  Clustering

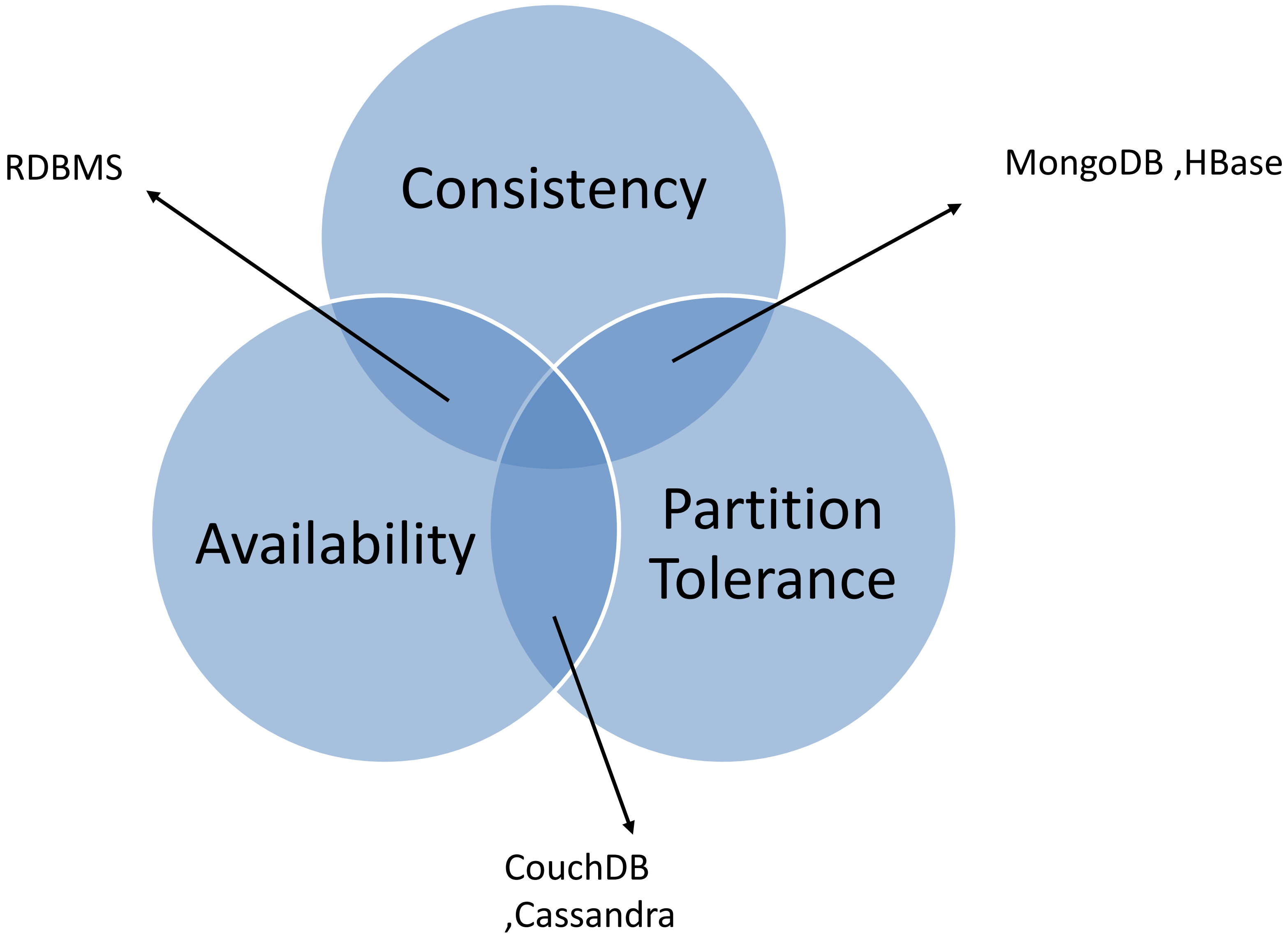
  The additional integration processing techniques include taxonomy integration, metadata integration, and master data integration.

- *User interface. The business users can create data processing rules in a drag-and-drop interface* and in an free-form text interface that supports 16 languages for processing.
- *Taxonomies. There are several categories of data that can be processed by any enterprise, and* the most effective way to processes a multistructured and multihierarchical data is by integrating third-party taxonomy libraries. Textual ETL supports over 40,000 known taxonomies in 16 languages for data enrichment and processing.
- *Output database. Textual ETL can write the output to any RDBMS or NoSQL databases. The* result sets are key-value pairs that can be used to integrate the structured and unstructured databases.

Goals

- Scalability
- Flexibility
- Naturalness
- Distribution
- Performance

CAP Theorem


RDBMS

Consistency

MongoDB ,HBase

Availability

Partition Tolerance

CouchDB ,Cassandra

- NOT all C,A,P can be satisfied simultaneously

BASE

- Basically Available
  - Guarantees availability
- Soft State
- Eventual Consistency