

Map Reduce and Hadoop



In this Lecture:

- Outline:
 - What is Big Data?
 - Issues with Big Data
 - What is Hadoop ?
 - What is Map Reduce ?
 - Example Map Reduce program.



Motivation: Google Example

- 20+ billion web pages x 20KB = 400+ TB
- 1 computer reads 30-35 MB/sec from disk
 - ~4 months to read the web
- ~1,000 hard drives to store the web
- Takes even more to do something useful with the data!
- Today, a standard architecture for such problems is emerging:
 - Cluster of commodity Linux nodes
 - Commodity network (ethernet) to connect them

Large-scale Computing

- Large-scale computing for data mining problems on commodity hardware
- Challenges:
 - How do you distribute computation?
 - How can we make it easy to write distributed programs?
 - Machines fail:
 - One server may stay up 3 years (1,000 days)
 - If you have 1,000 servers, expect to loose 1/day
 - People estimated Google had ~1M machines in 2011
 - 1,000 machines fail every day!



Big Data Challenges

- Scalability: processing should scale with increase in data.
- Fault Tolerance: function in presence of hardware failure
- Cost Effective: should run on commodity hardware
- Ease of use: programs should be small
- Flexibility: able to process unstructured data
- **Solution: Map Reduce !**

Idea and Solution

- **Issue:** Copying data over a network takes time
- **Idea:**
 - Bring computation close to the data
 - Store files multiple times for reliability
- Map-reduce addresses these problems
 - Elegant way to work with big data
 - Storage Infrastructure – File system
 - Google: GFS. Hadoop: HDFS
 - Programming model
 - Map-Reduce

Storage Infrastructure

- **Problem:**
 - If nodes fail, how to store data persistently?
- **Answer:**
 - Distributed File System:
- Provides global file namespace
- Google GFS; Hadoop HDFS;
- Typical usage pattern
 - Huge files (100s of GB to TB)
 - Data is rarely updated in place
 - Reads and appends are common

What is Hadoop ?

- A scalable fault-tolerant distributed system for data storage and processing.
- Core Hadoop:
 - Hadoop Distributed File System (HDFS)
 - Hadoop YARN: Job Scheduling and Cluster Resource Management
 - Hadoop Map Reduce: Framework for distributed data processing.
- Open Source system with large community support.
 - <https://hadoop.apache.org/>

What is Map Reduce ?

- Method for distributing a task across multiple servers.
- Proposed by Dean and Ghemawat, 2004.
- Consists of two developer created phases:
 - Map
 - Reduce
- In between Map and Reduce is the Shuffle and Sort phase.
- User is responsible for casting the problem into map – reduce framework.
- Multiple map-reduce jobs can be “chained”



Programming Model: MapReduce

- Warm-up task:
- We have a huge text document
- Count the number of times each distinct word appears in the file
- Sample application:
 - Analyze web server logs to find popular URLs

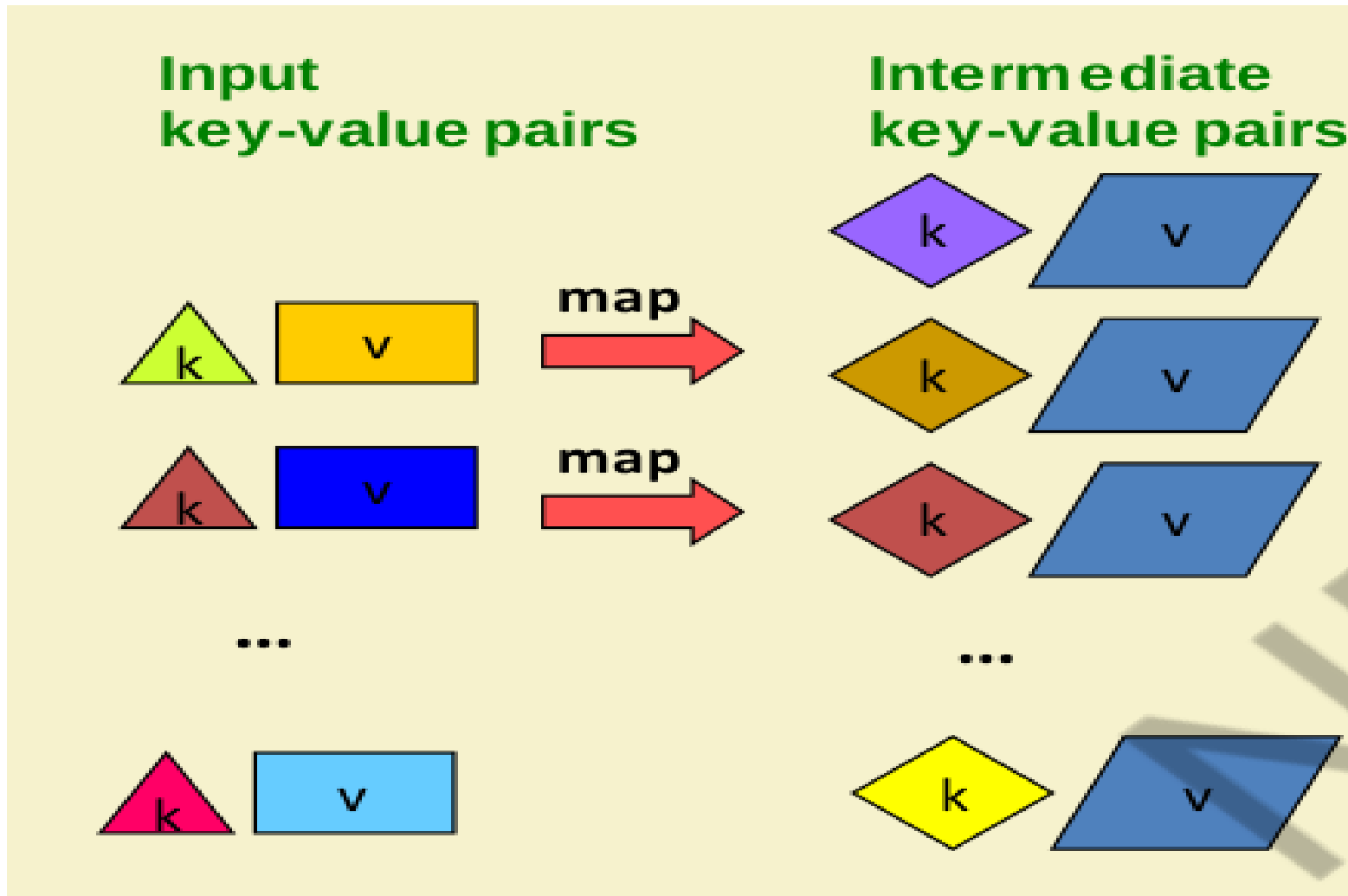
Task: Word Count

- Case 1:
 - File too large for memory, but all `<word, count>` pairs fit in memory
- Case 2:
- Count occurrences of words:
 - `words(doc.txt) | sort | uniq -c`
 - where `words` takes a file and outputs the words in it, one per a line
- Case 2 captures the essence of MapReduce
 - Great thing is that it is naturally parallelizable

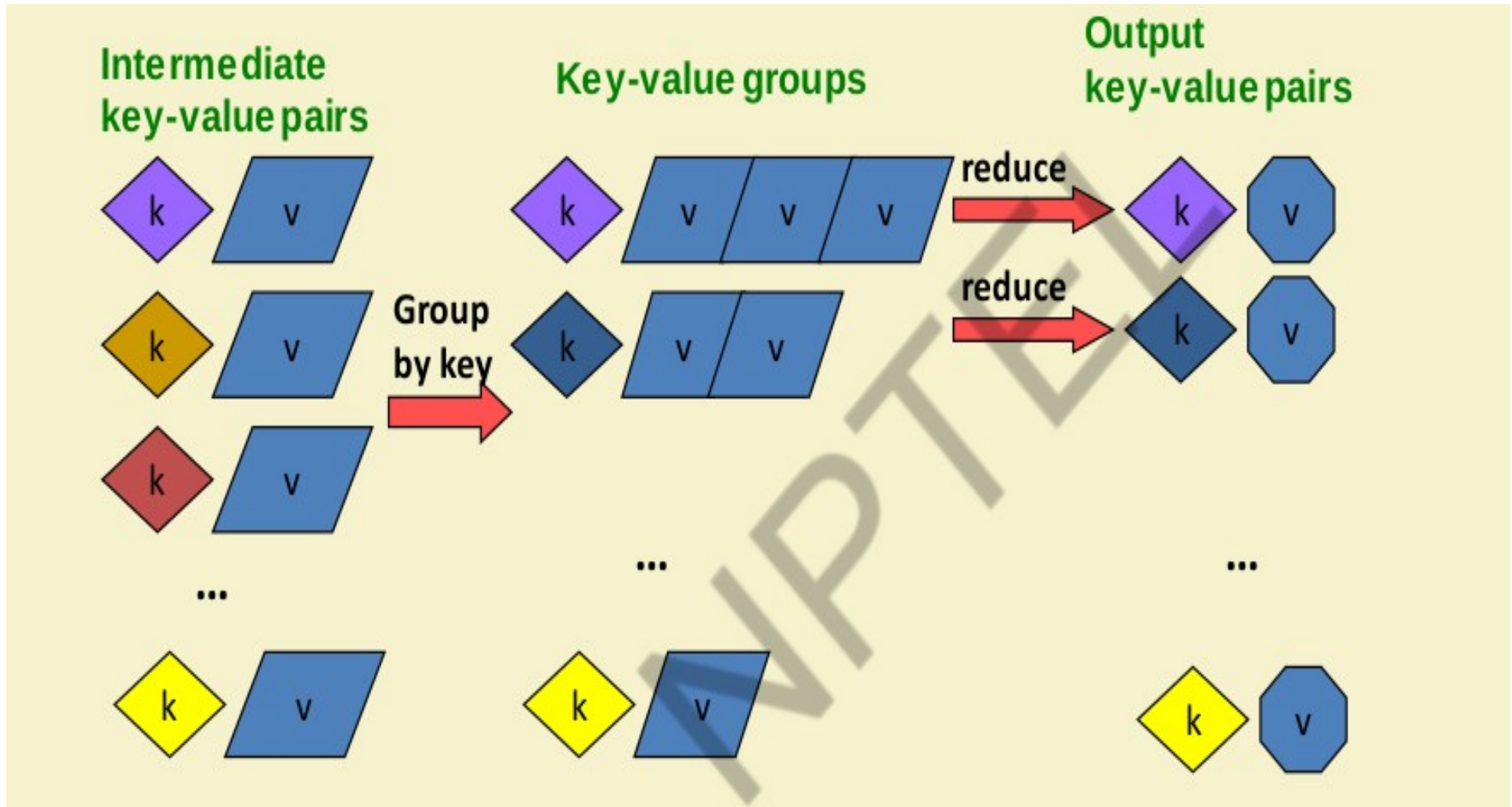
MapReduce: Overview

- Sequentially read a lot of data
- Map:
 - Extract something you care about
- Group by key: Sort and Shuffle
- Reduce:
 - Aggregate, summarize, filter or transform
- Write the result
- **Outline stays the same, Map and Reduce change to fit the problem**

MapReduce: The Map Step



MapReduce: The Reduce Step



More Specifically

- **Input:** a set of key-value pairs
- Programmer specifies two methods:
 - $\text{Map}(k, v) \rightarrow \langle k', v' \rangle^*$
 - Takes a key-value pair and outputs a set of key-value pairs
 - E.g., key is the filename, value is a single line in the file
 - There is one Map call for every (k, v) pair
 - $\text{Reduce}(k', \langle v' \rangle^*) \rightarrow \langle k', v'' \rangle^*$
 - **All values v' with same key k' are reduced together and processed in v' order**
 - There is one Reduce function call per unique key k'

MapReduce: Word Counting

Provided by the
programmer

MAP:

Read input and
produces a set of
key-value pairs

Group by key:

Collect all pairs
with same key

Provided by the
programmer

Reduce:

Collect all values
belonging to the
key and output

The crew of the space shuttle Endeavor recently returned to Earth as ambassadors, harbingers of a new era of space exploration. Scientists at NASA are saying that the recent assembly of the Dextre bot is the first step in a long-term space-based man/machine partnership. "The work we're doing now -- the robotics we're doing -- is what we're going to need

(The, 1)

(crew, 1)

(of, 1)

(the, 1)

(space, 1)

(shuttle, 1)

(Endeavor, 1)

(recently, 1)

....

(crew, 1)

(crew, 1)

(space, 1)

(the, 1)

(the, 1)

(the, 1)

(shuttle, 1)

(recently, 1)

...

(crew, 2)

(space, 1)

(the, 3)

(shuttle, 1)

(recently, 1)

...

Big document

(key, value)

(key, value)

(key, value)

Only sequential reads

Word Count Using MapReduce

- `map(key, value):`
 - `// key: document name; value: text of the document`
for each word `w` in `value`:
 `emit(w, 1)`
- `reduce(key, values):`
 - `// key: a word; value: an iterator over counts`
 `result = 0`
for each count `v` in `values`:
 `result += v`
 `emit(key, result)`

Map Phase

- User writes the mapper method.
- Input is an unstructured record:
 - E.g. A row of RDBMS table,
 - A line of a text file, etc
- Output is a set of records of the form: <key, value>
 - Both key and value can be anything, e.g. text, number, etc.
 - E.g. for row of RDBMS table: <column id, value>
 - Line of text file: <word, count>

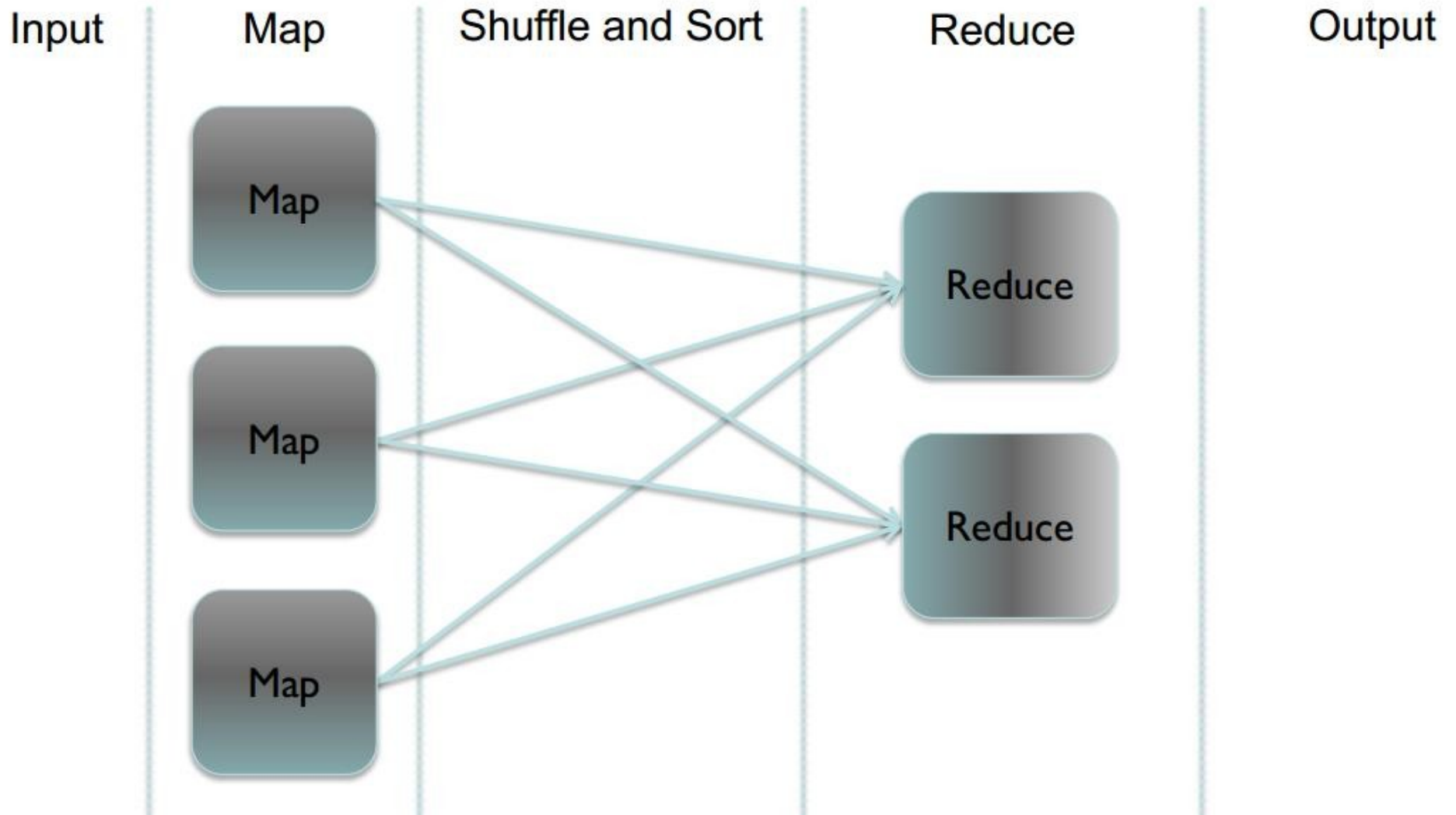
Shuffle/Sort phase

- Shuffle phase ensures that all the mapper output records with the same key value, goes to the same reducer.
- Sort ensures that among the records received at each reducer, records with same key arrives together.

Reduce phase

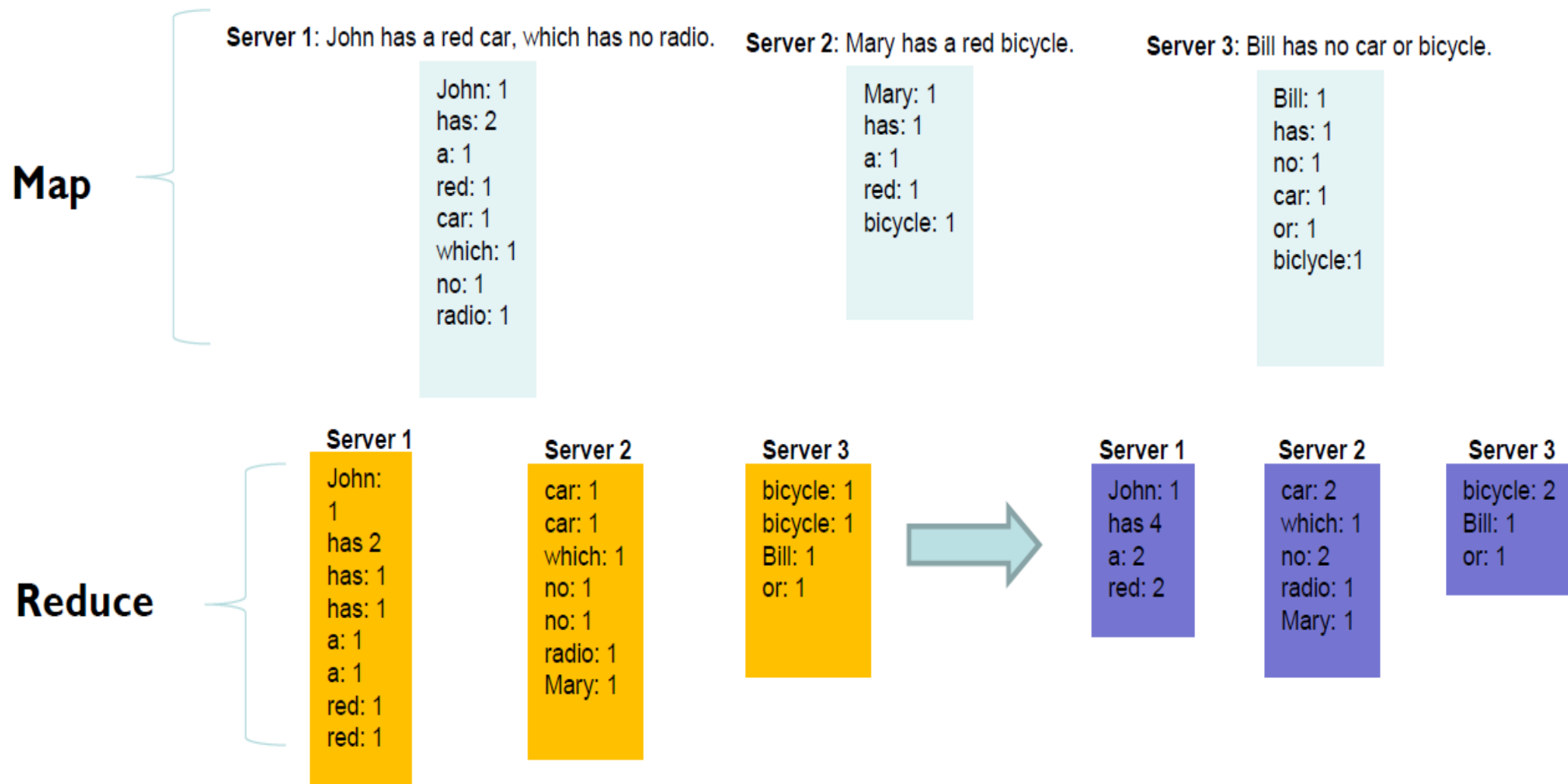
- Reducer is a user defined function which processes mapper output records with some of the keys output by mapper.
- Input is of the form <key, value>
 - All records having same key arrive together.
- Output is a set of records of the form <key, value>
 - Key is not important

Parallel picture

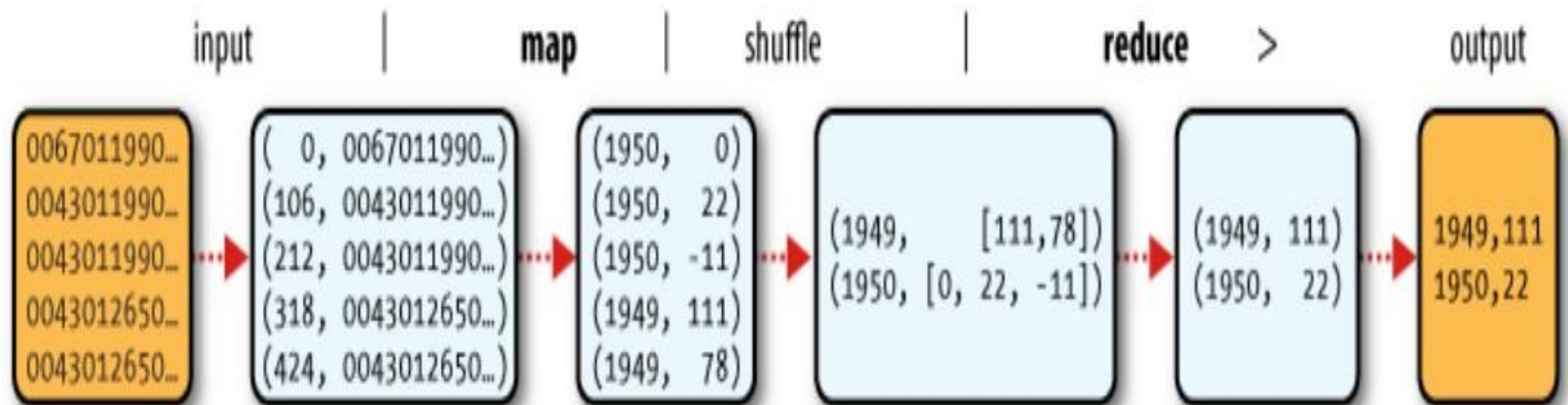


Example

- Word Count: Count the total no. of occurrences of each word



Map Reduce - Example



What was the max/min temperature for the last century ?



Hadoop Map Reduce

Provides:

- Automatic parallelization and Distribution
- Fault Tolerance
- Methods for interfacing with HDFS for colocation of computation and storage of output.
- Status and Monitoring tools
- API in Java
- Ability to define the mapper and reducer in many languages through Hadoop streaming.

What is Hadoop ?

- A scalable fault-tolerant distributed system for data storage and processing.
- Core Hadoop:
 - Hadoop Distributed File System (HDFS)
 - Hadoop YARN: Job Scheduling and Cluster Resource Management
 - Hadoop Map Reduce: Framework for distributed data processing.
- Open Source system with large community support.
 - <https://hadoop.apache.org/>

What's HDFS

- HDFS is a distributed file system that is fault tolerant, scalable and extremely easy to expand.
- HDFS is the primary distributed storage for Hadoop applications.
- HDFS provides interfaces for applications to move themselves closer to data.
- HDFS is designed to 'just work', however a working knowledge helps in diagnostics and improvements.



HDFS

- Design Assumptions
 - Hardware failure is the norm.
 - Streaming data access.
 - Write once, read many times.
 - High throughput, not low latency.
 - Large datasets.
- Characteristics:
 - Performs best with modest number of large files
 - Optimized for streaming reads
 - Layer on top of native file system.

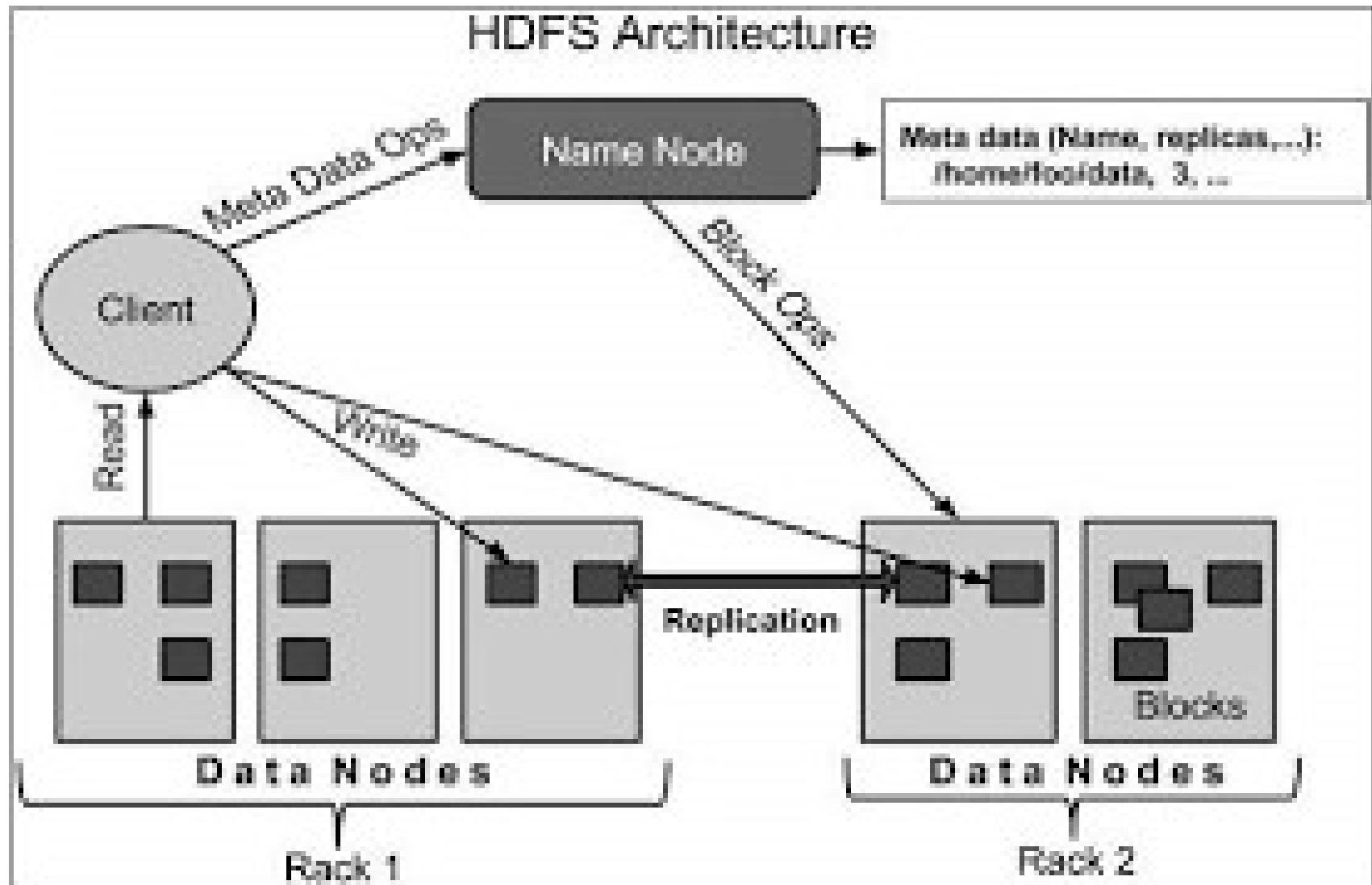
HDFS

- Data is organized into file and directories.
- Files are divided into blocks and distributed to nodes.
- Block placement is known at the time of read
 - Computation moved to same node.
- Replication is used for:
 - Speed
 - Fault tolerance
 - Self healing.

Components of HDFS

- There are two (and a half) types of machines in a HDFS cluster
 - **NameNode** :- is the heart of an HDFS filesystem, it maintains and manages the file system metadata. E.g; what blocks make up a file, and on which datanodes those blocks are stored.
 - **DataNode** :- where HDFS stores the actual data, there are usually quite a few of these.

HDFS Architecture



Goals of HDFS

- Very Large Distributed File System
 - 10K nodes, 100 million files, 10 PB
- Assumes Commodity Hardware
 - Files are replicated to handle hardware failure
 - Detect failures and recovers from them
- Optimized for Batch Processing
 - Data locations exposed so that computations can move to where data resides
 - Provides very high aggregate bandwidth User Space, runs on heterogeneous OS



Distributed File System

- Single Namespace for entire cluster
- Data Coherency
 - Write-once-read-many access model
 - Client can only append to existing files
- Files are broken up into blocks
 - Typically 128 MB block size
 - Each block replicated on multiple DataNodes
- Intelligent Client
 - Client can find location of blocks
 - Client accesses data directly from DataNode

NameNode Metadata

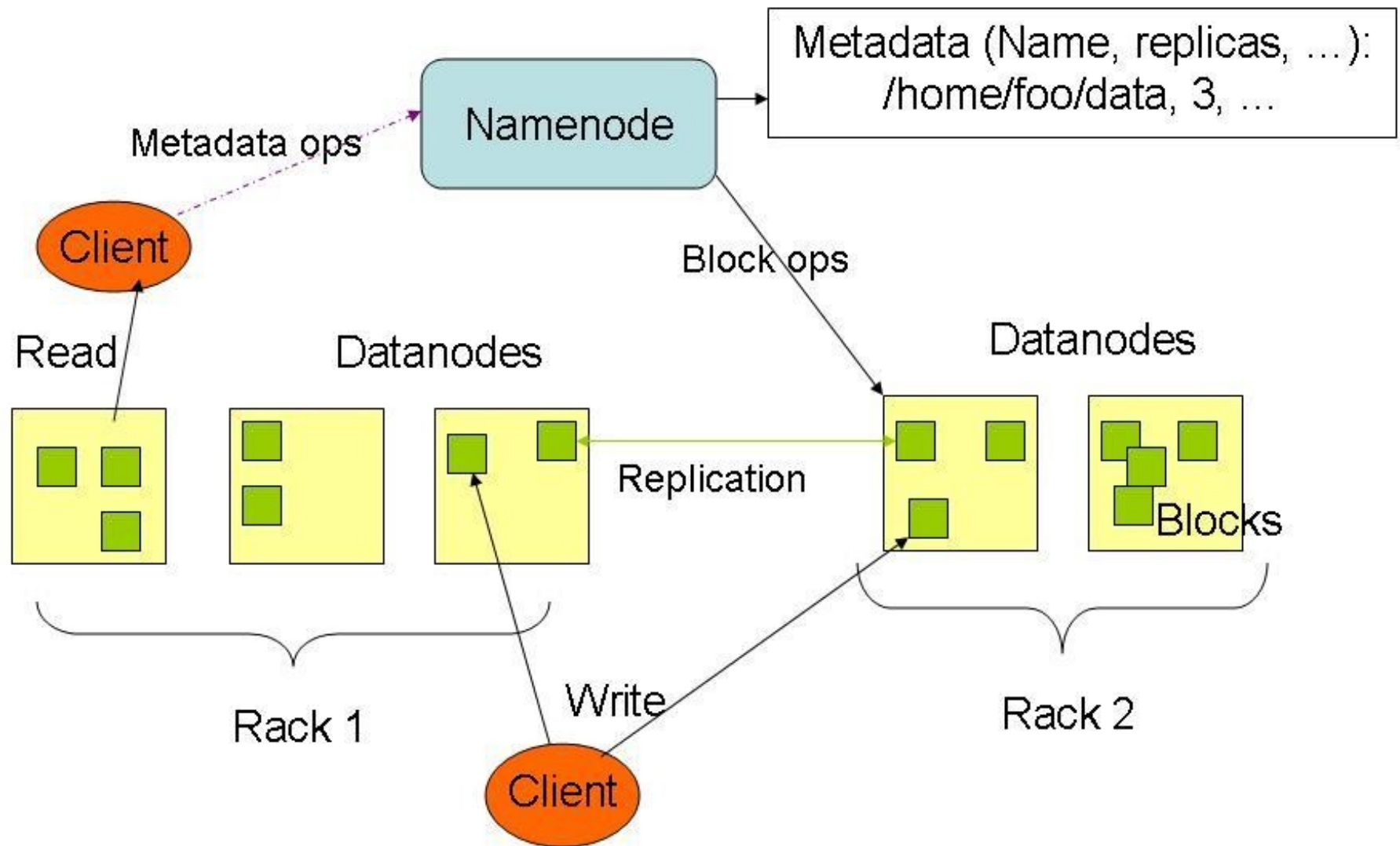
- Meta-data in Memory
 - The entire metadata is in main memory
 - No demand paging of meta-data
- Types of Metadata
 - List of files
 - List of Blocks for each file
 - List of DataNodes for each block
 - File attributes, e.g creation time, replication factor
- A Transaction Log
 - Records file creations,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



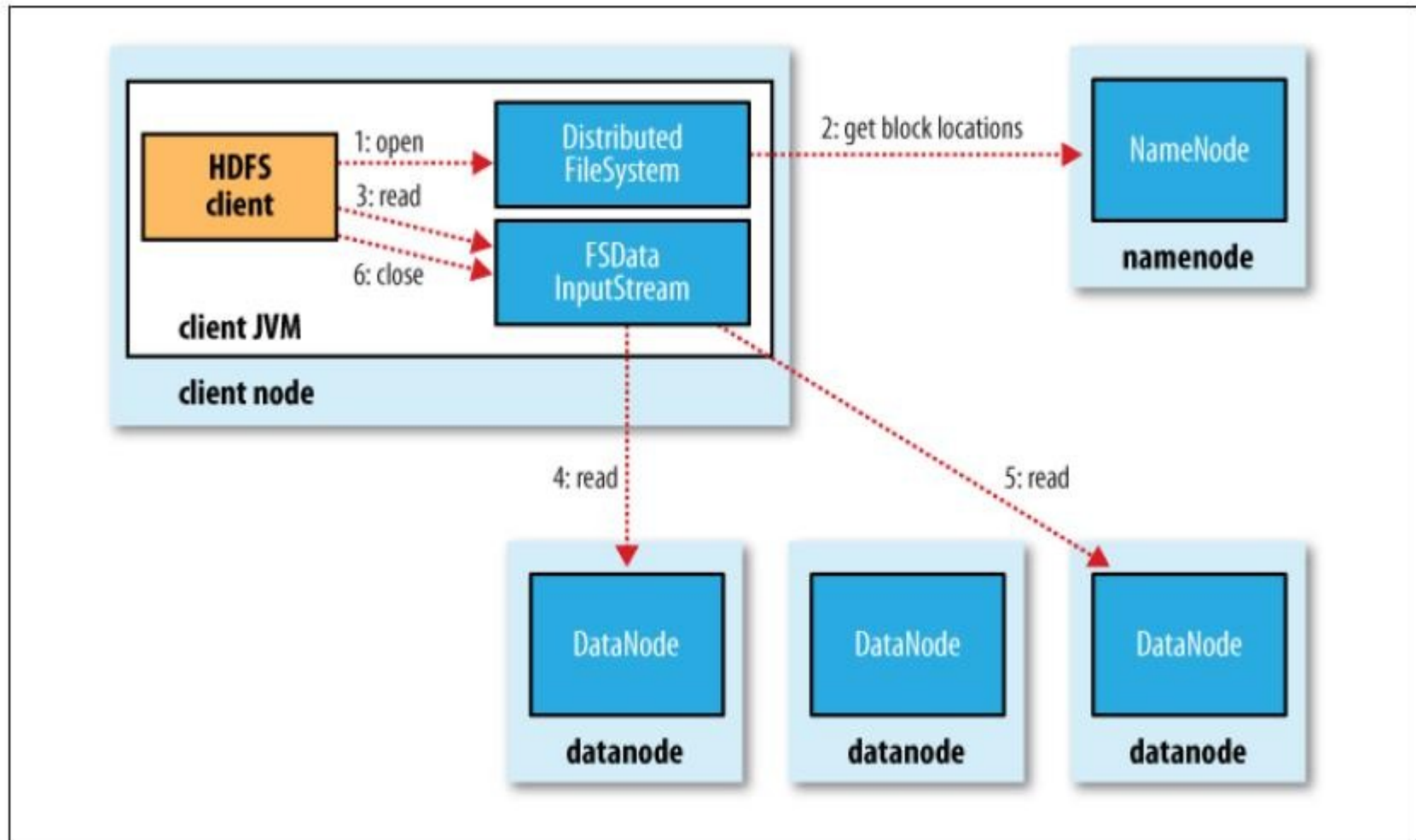
DataNode

- A Block Server
 - Stores data in the local file system (e.g. ext3)
 - Stores meta-data of a block (e.g. CRC)
 - Serves data and meta-data to Clients
- Block Report
 - Periodically sends a report of all existing blocks to the NameNode
- Facilitates Pipelining of Data
 - Forwards data to other specified DataNodes

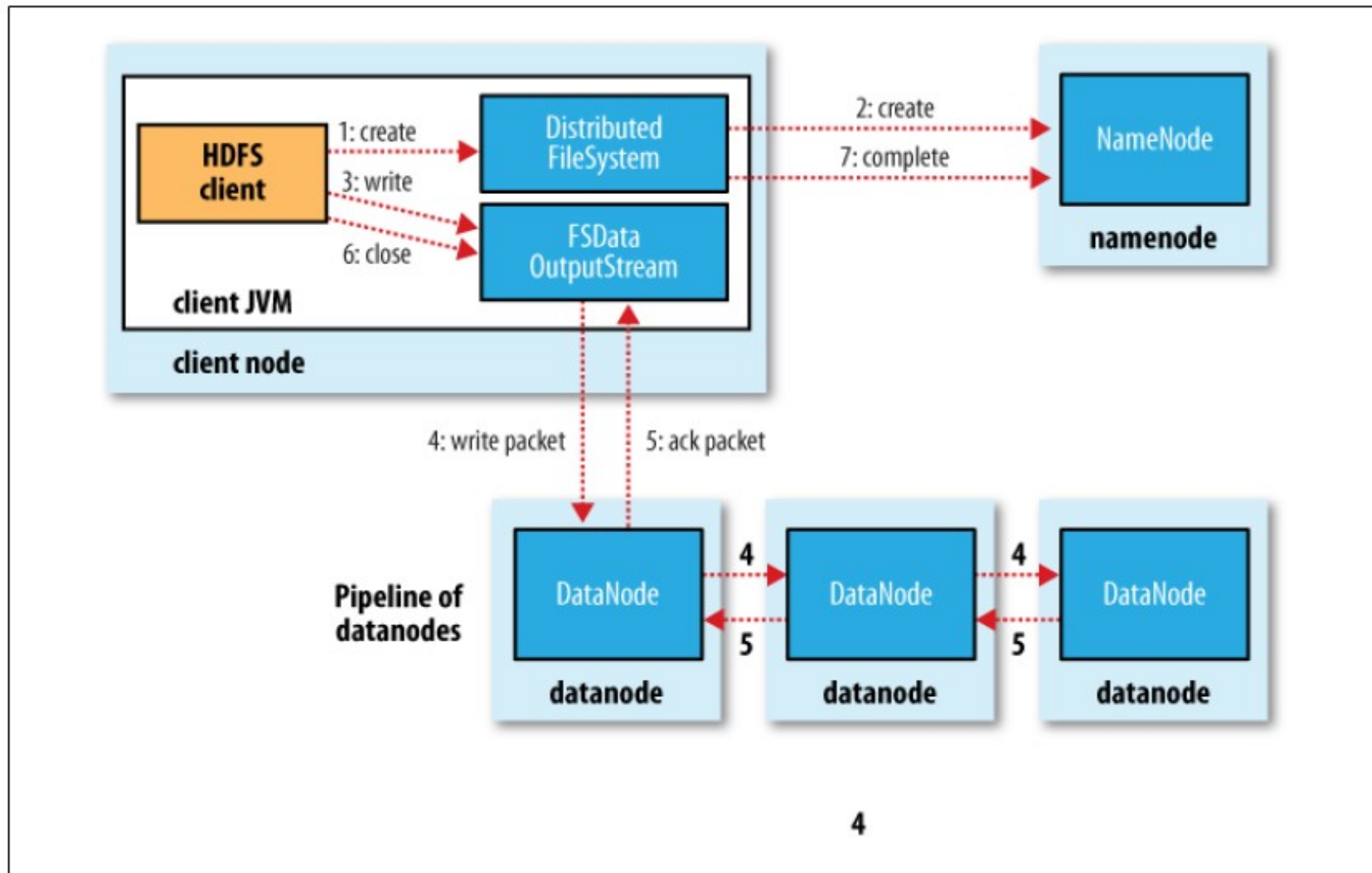
HDFS Architecture



HDFS read client



HDFS write Client





Block Placement

- Current Strategy
 - One replica on local node
 - Second replica on a remote rack
 - Third replica on same remote rack
 - Additional replicas are randomly placed
- Clients read from nearest replica
- Would like to make this policy pluggable



NameNode Failure

- A single point of failure
- Transaction Log stored in multiple directories
 - A directory on the local file system
 - A directory on a remote file system (NFS/CIFS)
- Need to develop a real HA solution



Data Pipelining

- Client retrieves a list of DataNodes on which to place replicas of a block
- Client writes block to the first DataNode
- The first DataNode forwards the data to the next DataNode in the Pipeline
- When all replicas are written, the Client moves on to write the next block in file



Conclusion:

- We have seen:
 - The structure of HDFS.
 - The architecture of HDFS system.
 - Internal functioning of HDFS.