



INDIAN INSTITUTE OF  
INFORMATION  
TECHNOLOGY

# Distributed Computing Systems



Dr. Animesh Chaturvedi

Assistant Professor: **IIIT Dharwad**

Young Researcher: **Heidelberg Laureate Forum**  
and **Pingala Interaction in Computing**

Young Scientist: **Lindau Nobel Laureate Meetings**

Postdoc: **King's College London & The Alan Turing Institute**

PhD: **IIT Indore** MTech: **IIITDM Jabalpur**



Indian Institute of Technology Indore  
भारतीय प्रौद्योगिकी संस्थान इंदौर

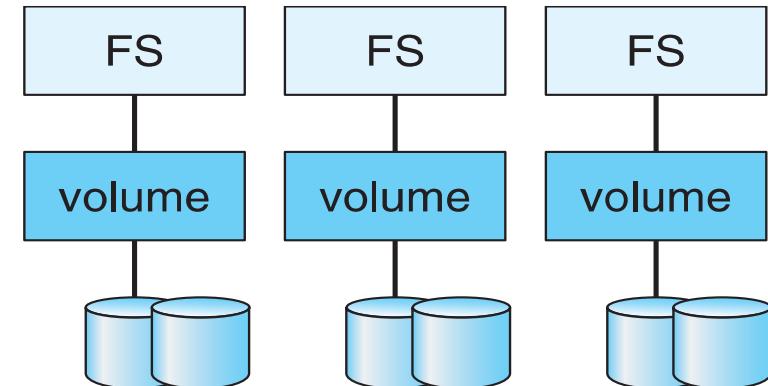


PDPM  
Indian Institute of Information Technology,  
Design and Manufacturing, Jabalpur

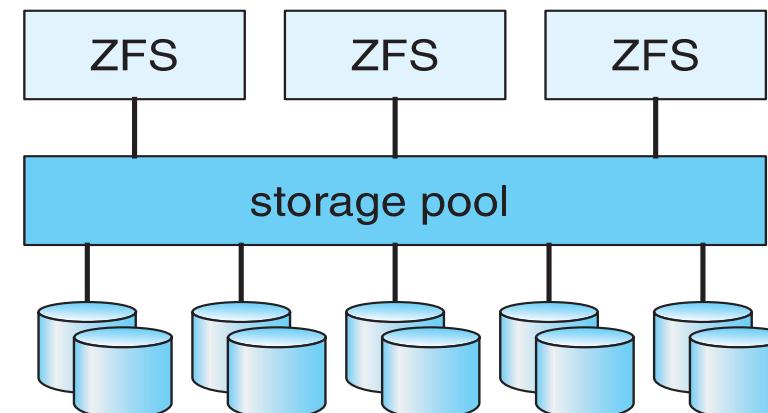
# Distributed File System

# Traditional and Pooled Storage

- File System (FS)
- Example: Hadoop Distributed File System (HDFS)
- **Horizontally scalable**



(a) Traditional volumes and file systems.



(b) ZFS and pooled storage.

# Distributed File System

- Don't move data to workers... move workers to the data!
  - Store data on the local disks of nodes in the cluster
  - Start up the workers on the node that has the data local
- Why?
  - Network bisection bandwidth is limited
  - Not enough RAM to hold all the data in memory
  - Disk access is slow, but disk throughput is reasonable
- A distributed file system is the answer
  - GFS (Google File System) for Google's MapReduce
  - HDFS (Hadoop Distributed File System) for Hadoop

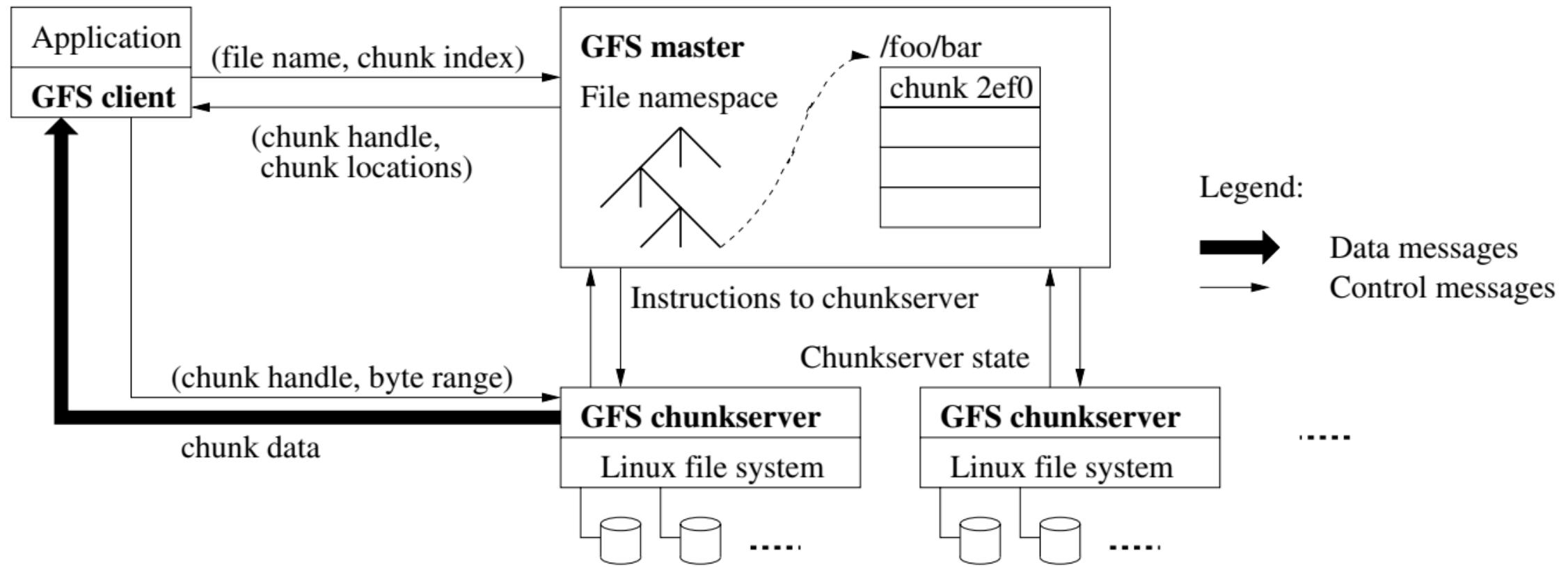
# Big Files to Google File System (GFS)

- Earlier Google effort, "BigFiles", developed by Larry Page and Sergey Brin.
  - Supervisors: Hector Garcia-Molina, Rajeev Motwani, Jeff Ullman, and Terry Winograd
- “Big File” was regenerated as “Google File System” by Sanjay Ghemawat, et al.
- Google File System (GFS)
  - “It is widely deployed within Google as the storage platform for the generation and processing of data used by Google service as well as research and development efforts that require large data sets.” 2003
  - “The largest cluster to date provides hundreds of terabytes of storage across thousands of disks on over a thousand machines, and it is concurrently accessed by hundreds of clients.” 2003

<http://infolab.stanford.edu/~backrub/google.html>

Sanjay Ghemawat, Howard Gobioff, and Shun-Tak Leung. "The Google file system." Proceedings of the nineteenth ACM symposium on Operating systems principles. 2003.

# Google File System (GFS)



[https://en.wikipedia.org/wiki/Google\\_File\\_System](https://en.wikipedia.org/wiki/Google_File_System)

<https://sites.google.com/site/gfsassignmentwiki/home>

Sanjay Ghemawat, Howard Gobioff, and Shun-Tak Leung. "The Google file system." Proceedings of the nineteenth ACM symposium on Operating systems principles. 2003.

# GFS: Assumptions

- Choose commodity hardware over “exotic” hardware
  - Scale “out”, not “up”
- High component failure rates
  - Inexpensive commodity components fail all the time
- “Modest” number of huge files
  - Multi-gigabyte files are common, if not encouraged
- Files are write-once, mostly appended to
  - Perhaps concurrently
- Large streaming reads over random access
  - High sustained throughput over low latency

[https://en.wikipedia.org/wiki/Google\\_File\\_System](https://en.wikipedia.org/wiki/Google_File_System)

<https://sites.google.com/site/gfsassignmentwiki/home>

Sanjay Ghemawat, Howard Gobioff, and Shun-Tak Leung. "The Google file system." Proceedings of the nineteenth ACM symposium on Operating systems principles. 2003.

# GFS: Design Decisions

- Files stored as chunks
  - Fixed size (64MB)
- Reliability through replication
  - Each chunk replicated across 3+ chunkservers
- Single master to coordinate access, keep metadata
  - Simple centralized management
- No data caching
  - Little benefit due to large datasets, streaming reads
- Simplify the API
  - Push some of the issues onto the client (e.g., data layout)

HDFS = GFS clone (same basic ideas implemented in Java)

# GFS to HDFS

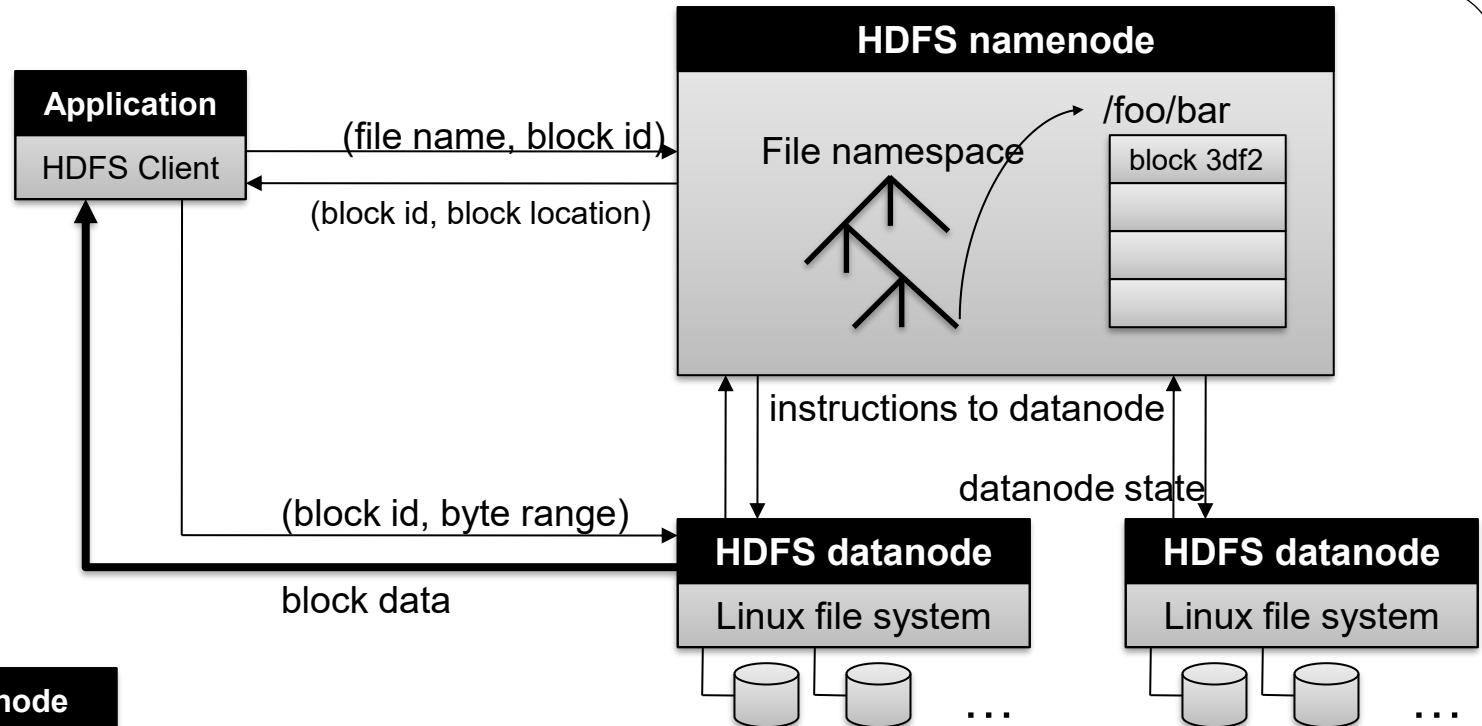
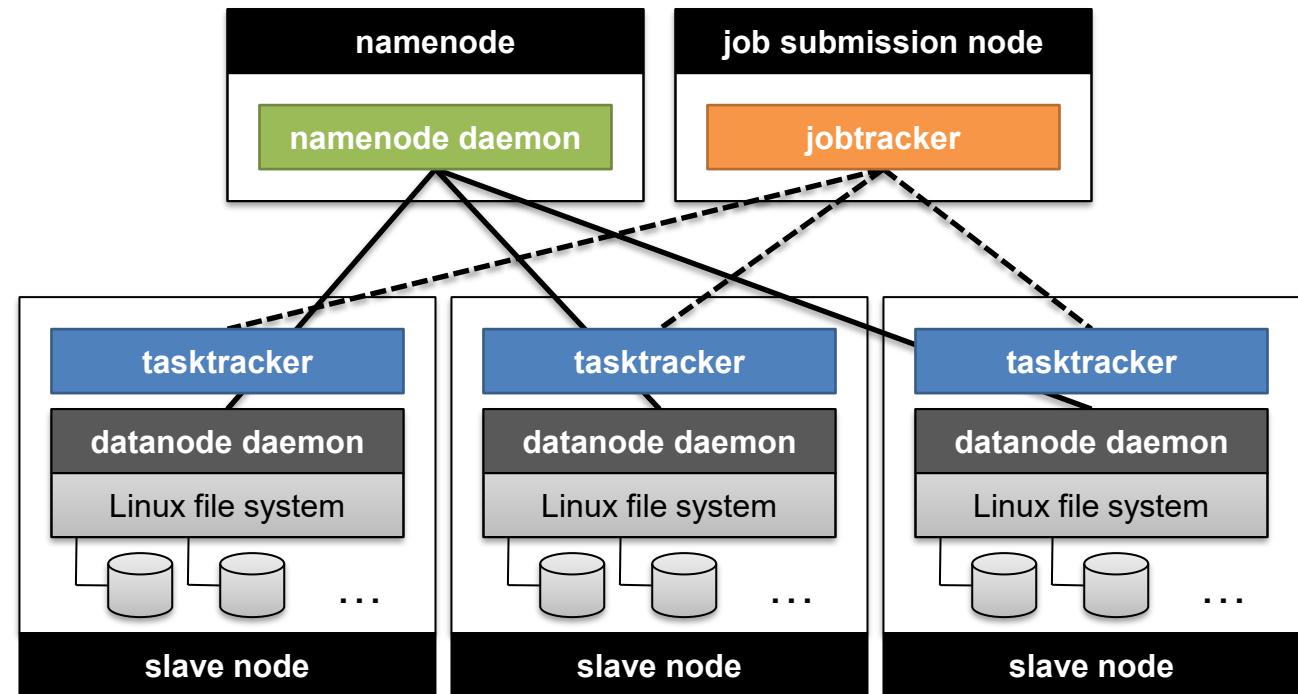
- Google File System (GFS) has similar open-source
  - “Hadoop Distributed File System (HDFS)”
- GFS and HDFS are distributed computing environment to process “Big Data”.
- GFS and HDFS are not implemented in the kernel of an operating system, but they are instead provided as a userspace library.
- GFS and HDFS properties
  - Files are divided into fixed-size chunks of 64 megabytes.
  - Scalable distributed file system for large distributed data intensive applications.
  - Provides fault tolerance.
  - High aggregate performance to a large number of clients.

[https://en.wikipedia.org/wiki/Google\\_File\\_System](https://en.wikipedia.org/wiki/Google_File_System)

Sanjay Ghemawat, Howard Gobioff, and Shun-Tak Leung. "The Google file system." Proceedings of the nineteenth ACM symposium on Operating systems principles. 2003.

# GFS to HDFS

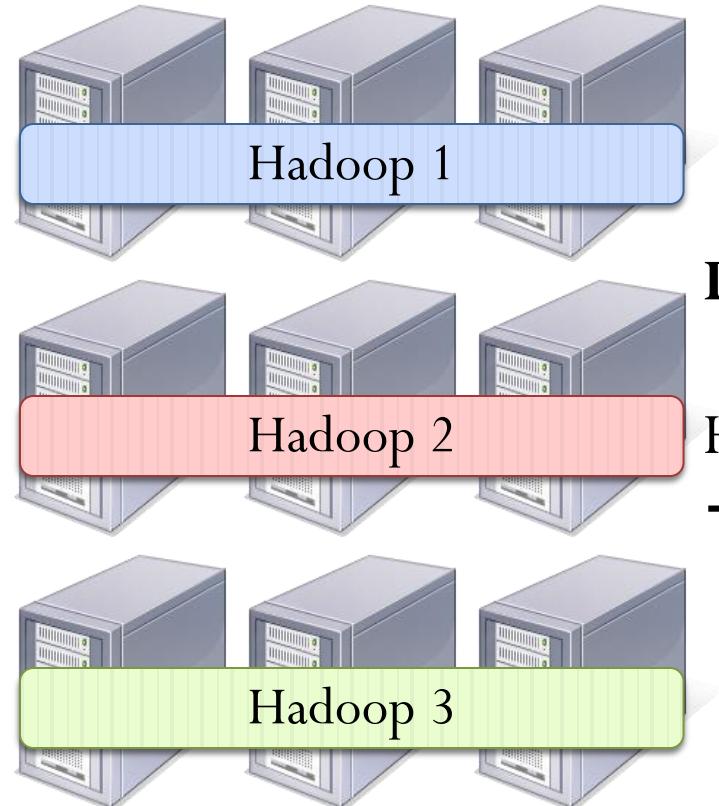
- GFS master
  - Hadoop namenode
- GFS chunkservers
  - Hadoop datanodes



# Coarse-grained sharing

Option: Coarse-grained sharing

- Give framework a (slice of) machine for its entire duration

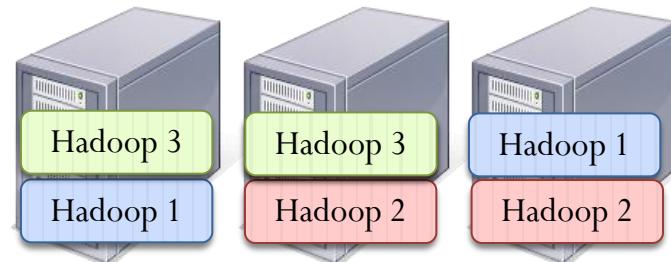


**Data locality compromised** if machine held for long time

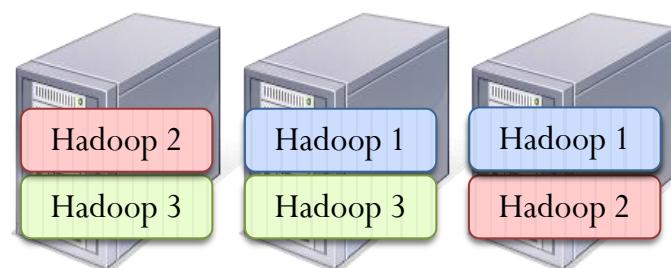
Hard to account for new frameworks and changing demands  
→ **hurts utilization and interactivity**

# Fine-grained sharing

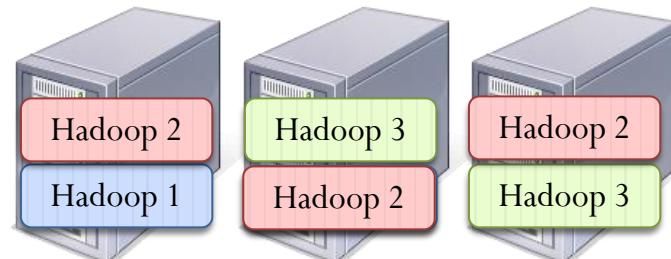
- Support frameworks that use smaller *tasks* (in time and space) by multiplexing them across all available resources



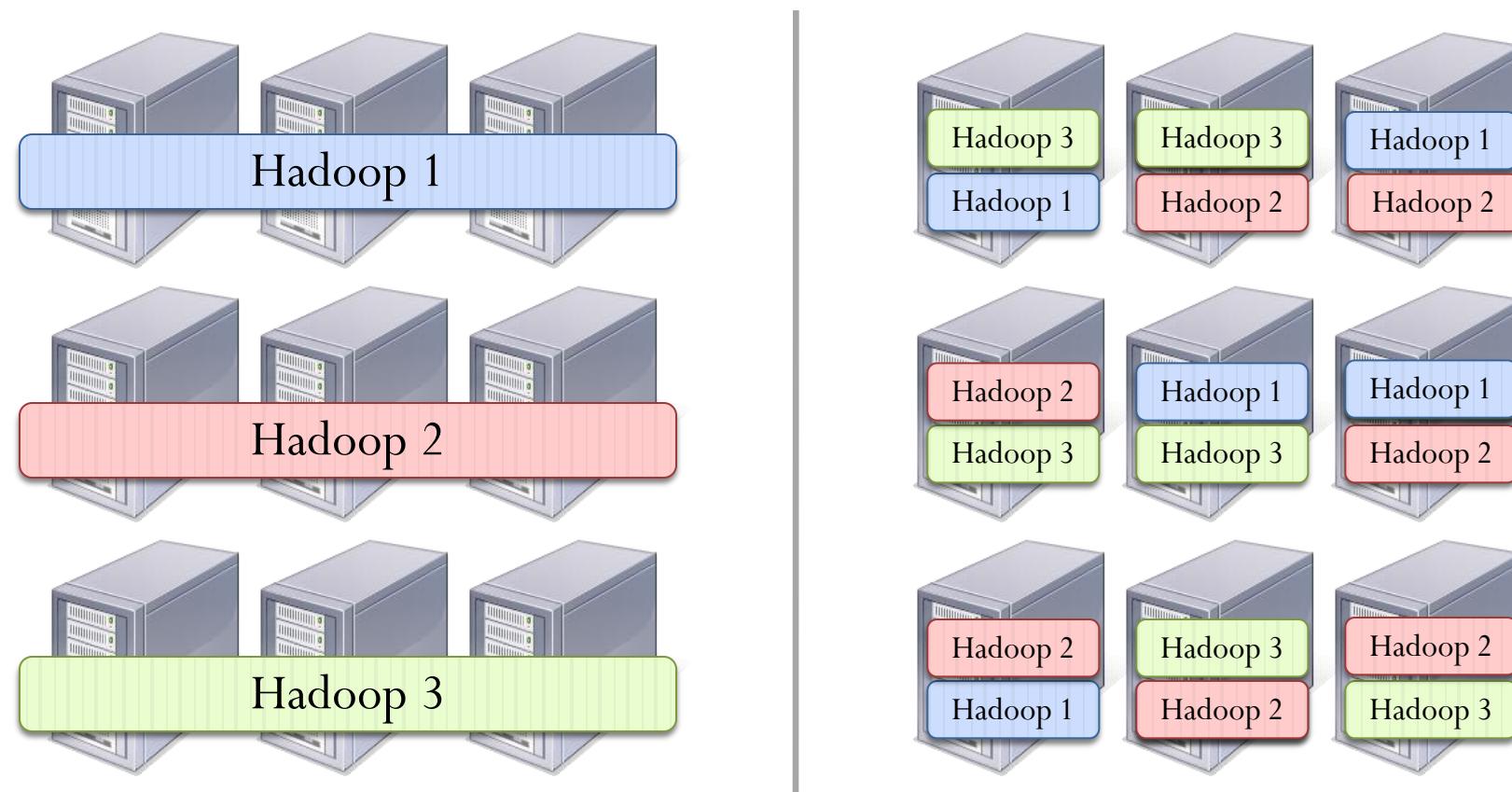
Frameworks can take turns accessing data on each node



Can resize frameworks shares to get utilization & interactivity



# Multiple Hadoops Experiment





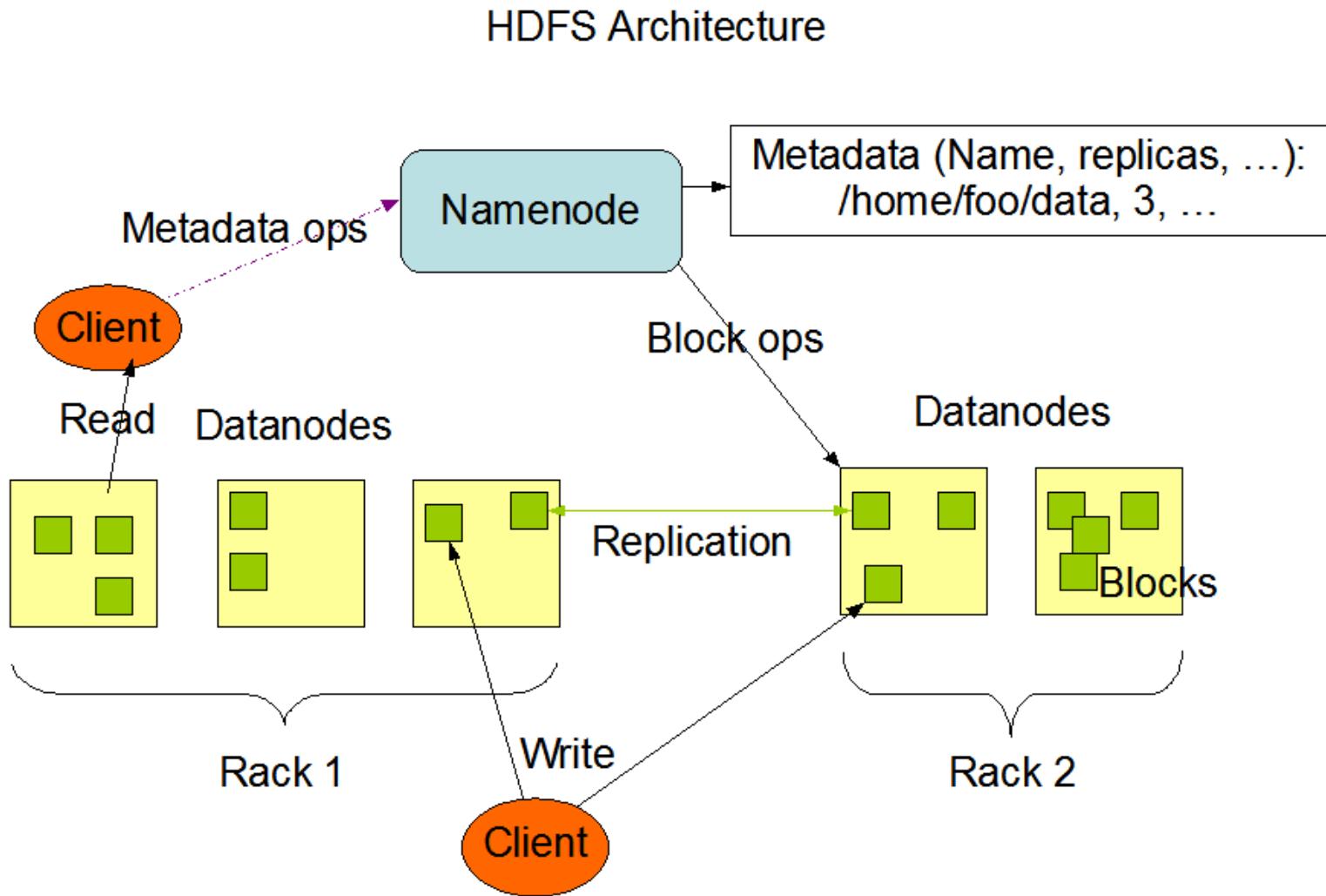
# Hadoop

- “The Apache Hadoop project develops open source software for reliable, scalable, distributed computing.” Software library and a framework.
- Created by Doug Cutting Named on his son's stuffed elephant
- For distributed processing of large data sets across clusters of computers using simple programming models.
- Locality of reference
- Scalability: Scale up from single servers to thousands of machines,
  - Each offering local computation and storage
  - Program remains same for 10, 100, 1000,... nodes
  - Corresponding performance improvement
- Fault-tolerant file system:
  - Detect and handle failures and Delivering a highly-available.
- Hadoop Distributed File System (HDFS) Modeled on Google File system
- MapReduce for Parallel computation using
- Components – Pig, Hbase, HIVE, ZooKeeper



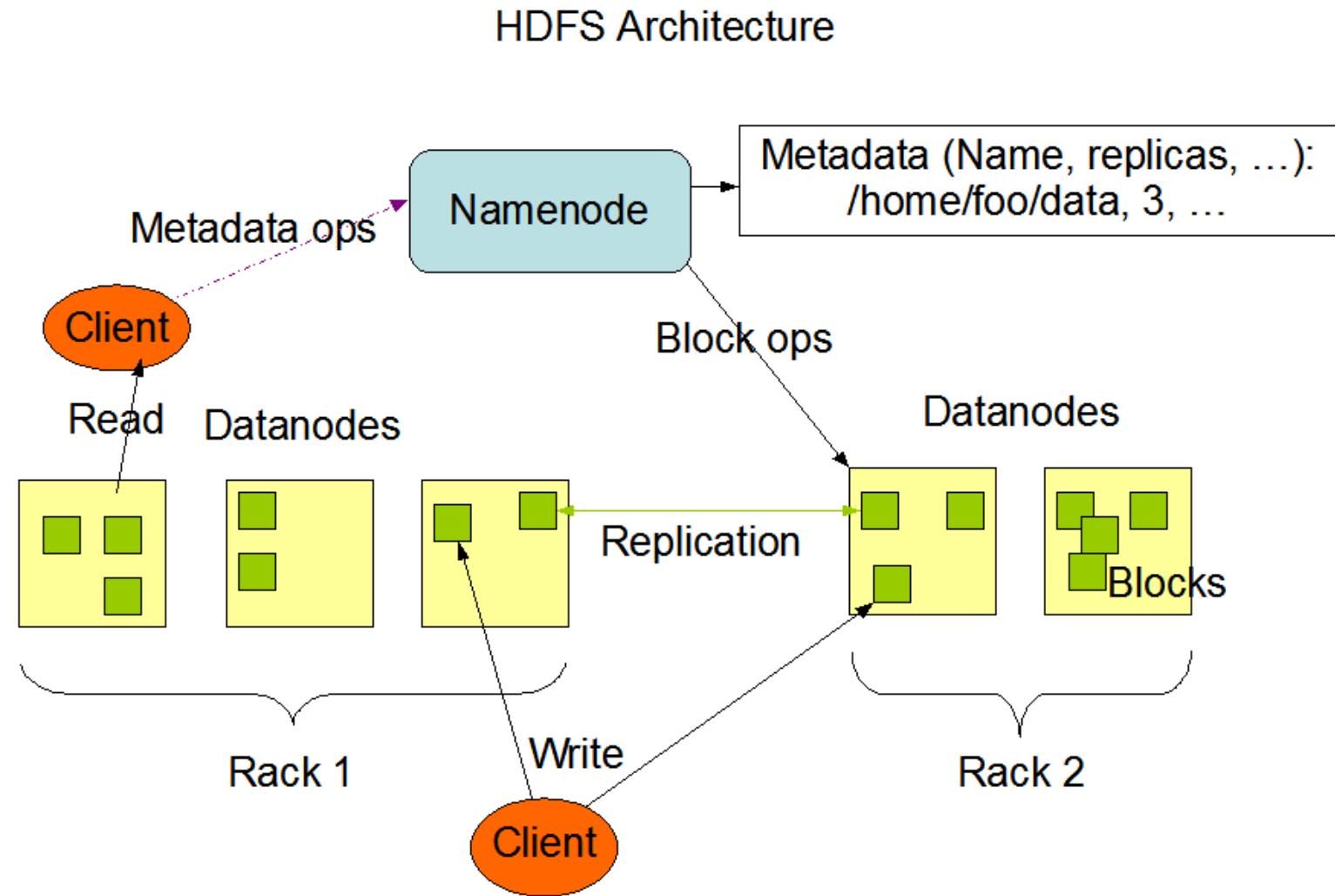
# Hadoop Distributed File System (HDFS)

- HDFS has a master/slave architecture. An HDFS cluster consists of a single NameNode, a master server that manages the file system namespace and regulates access to files by clients.
- There are a number of DataNodes, usually one per node in the cluster, which manage storage attached to the nodes that they run on.
- HDFS exposes a file system namespace and allows user data to be stored in files.
- A file is split into one or more blocks and these blocks are stored in a set of DataNodes.



# Hadoop Distributed File System (HDFS)

- The NameNode executes file system namespace operations like opening, closing, and renaming files and directories. It also determines the mapping of blocks to 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.



# Namenode Responsibilities

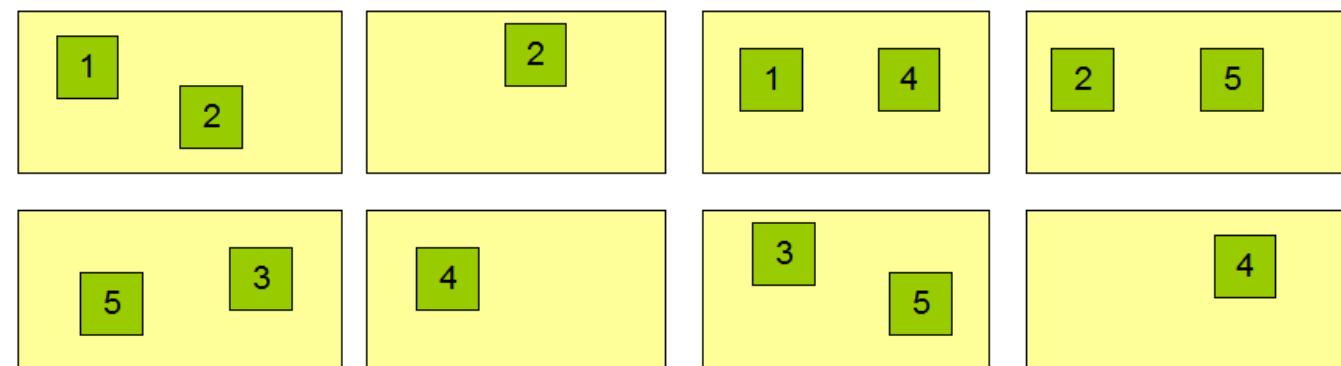
- **Manages File System namespace**

- mapping files to blocks and blocks to data nodes.
- Maintains status of data nodes
- holds file/directory structure, metadata, file-to-block mapping, access permissions, etc.

## Block Replication

Namenode (Filename, numReplicas, block-ids, ...)  
/users/sameerp/data/part-0, r:2, {1,3}, ...  
/users/sameerp/data/part-1, r:3, {2,4,5}, ...

## Datanodes



- **Maintaining overall health:**

- Periodic communication with the Datanodes
- Block re-replication and rebalancing
- Garbage collection

# Namenode Responsibilities

- **Heartbeat:** Datanode sends heartbeat at regular intervals, if heartbeat is not received, Datanode is declared to be dead

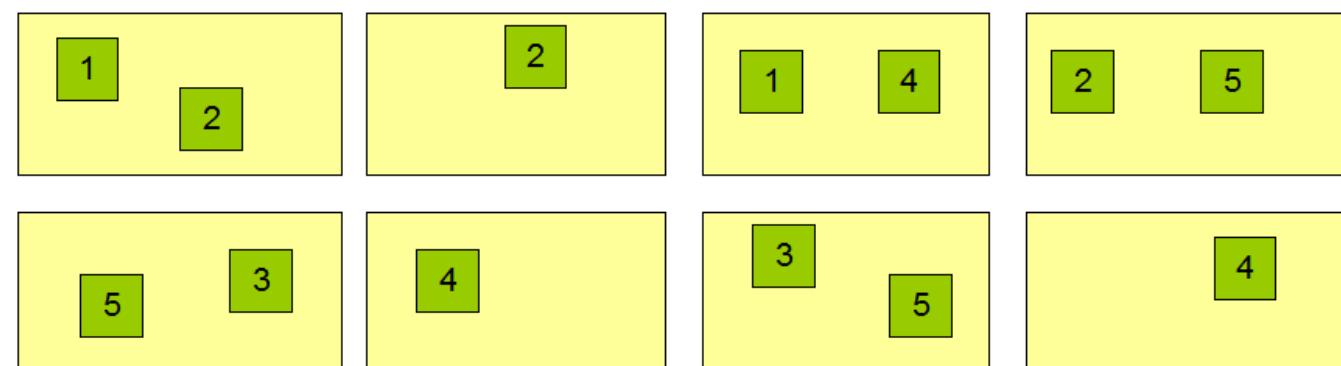
Block Replication

Namenode (Filename, numReplicas, block-ids, ...)  
/users/sameerp/data/part-0, r:2, {1,3}, ...  
/users/sameerp/data/part-1, r:3, {2,4,5}, ...

- **Coordinating file operations:**

- Directs clients to Datanodes for reads and writes
- No data is moved through the Namenode

Datanodes



- **Blockreport:** Datanode sends list of blocks on it. Used to check health of HDFS

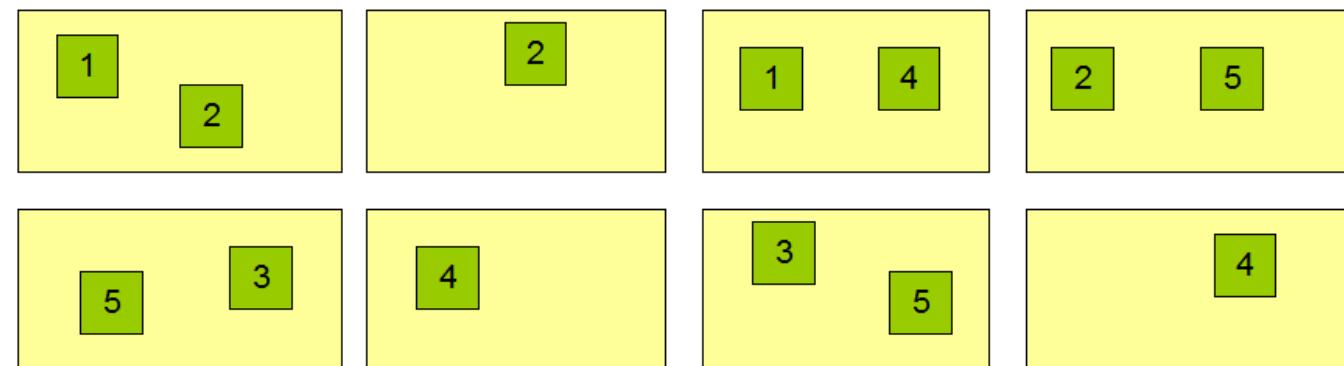
# Datanodes: Responsibilities

- Replicates
  - On Datanode failure,
  - On Disk failure,
  - On Block corruption
- Data integrity
  - Checksum for each block,
  - Stored in hidden file
- Rebalancing - balancer tool
  - Provisioning: addition of new nodes,
  - Decommissioning: remove node,
  - Deletion of some files

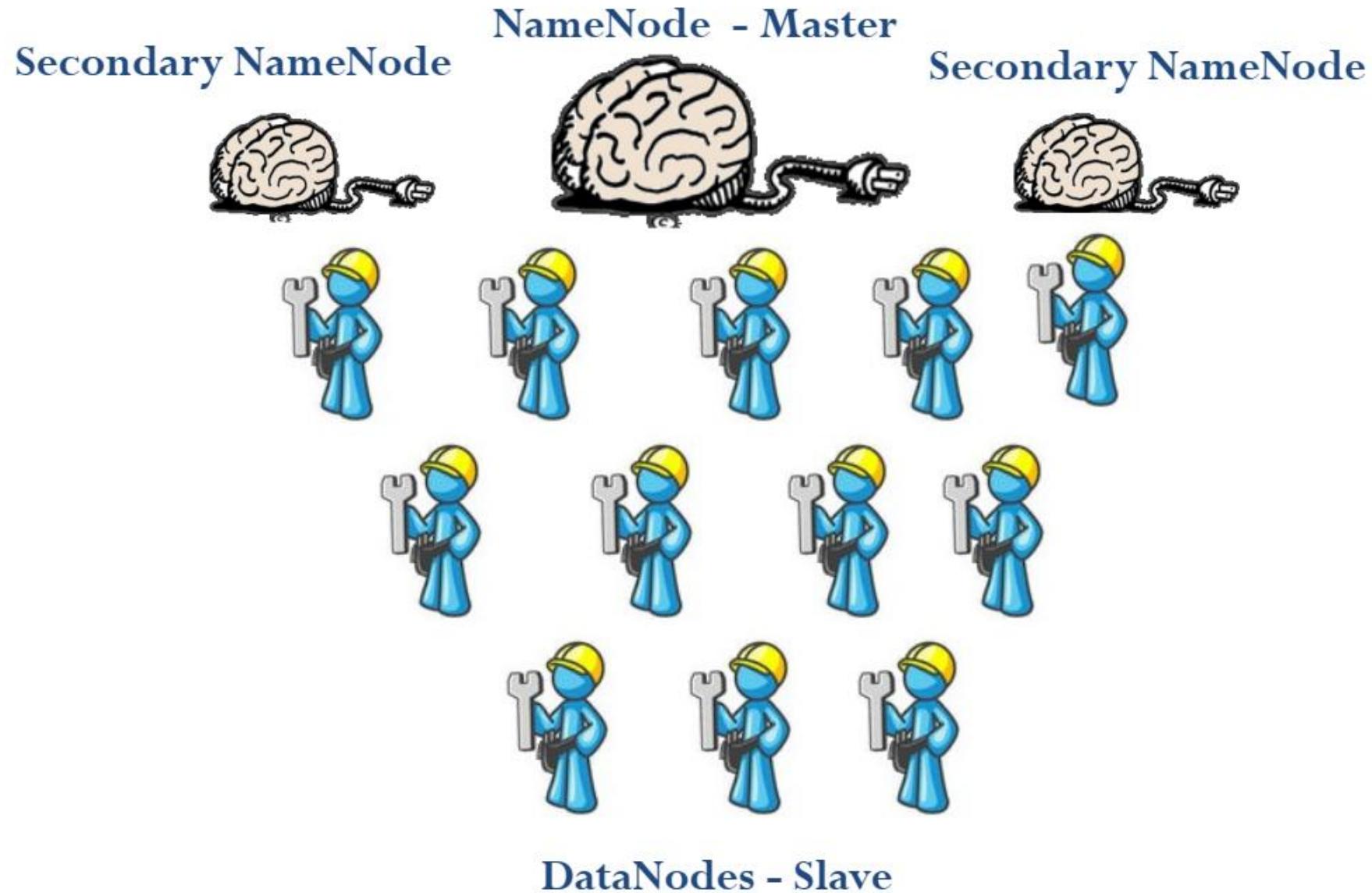
Block Replication

Namenode (Filename, numReplicas, block-ids, ...)  
/users/sameerp/data/part-0, r:2, {1,3}, ...  
/users/sameerp/data/part-1, r:3, {2,4,5}, ...

Datanodes



# HDFS Force



# HDFS Prerequisites & Environment Setup:

- **Install Java:** Hadoop is Java-based. Recommended to use OpenJDK 8 for compatibility.

```
sudo apt update && sudo apt install openjdk-8-jdk
```

- **Create Hadoop User:** Dedicated users improve security.

```
sudo adduser Hadoop
```

```
su - Hadoop
```

- **Configure SSH:** Enable password-less login so Hadoop daemons can communicate with the localhost.

```
ssh-keygen -t rsa -P "" -f ~/.ssh/id_rsa
```

```
cat ~/.ssh/id_rsa.pub >> ~/.ssh/authorized_keys
```

```
chmod 0600 ~/.ssh/authorized_keys
```

```
ssh localhost # Test connection
```

# Hadoop Installation Steps

- **Download & Extract:** Get the latest stable release from the Apache Hadoop Official Page.

```
wget https://dlcdn.apache.org/hadoop/common/hadoop-3.3.6/hadoop-3.3.6.tar.gz
```

```
tar -xvzf hadoop-3.3.6.tar.gz
```

```
mv hadoop-3.3.6 Hadoop
```

- **Set Environment Variables:** Update your `~/.bashrc` file to include Hadoop paths.

```
export HADOOP_HOME=/home/hadoop/hadoop
```

```
export PATH=$PATH:$HADOOP_HOME/bin:$HADOOP_HOME/sbin
```

```
source ~/.bashrc
```

# Hadoop Configuration Files

- Navigate to \$HADOOP\_HOME/etc/hadoop/ and update these files:
- **core-site.xml**: Set the default filesystem URL.

xml

```
<property>
  <name>fs.defaultFS</name>
  <value>hdfs://localhost:9000</value>
</property>
```

- **hdfs-site.xml**: Define storage paths and replication factor (set to 1 for single-node).

```
<property>
  <name>dfs.replication</name>
  <value>1</value>
</property>
<property>
  <name>dfs.namenode.name.dir</name>
  <value>file:///home/hadoop/hadoopdata/hdfs/namenode</value>
</property>
```

# Hadoop Cluster Initialization & Operation

- **Format NameNode:** This initializes the HDFS metadata directory.

*hdfs namenode –format*

- **Start/Stop Services:** Use the provided scripts in the sbin directory.

*start-dfs.sh # Starts NameNode and DataNode*

*jps # Verify processes are running*

*stop-dfs.sh # Stop HDFS*

# Hadoop Common HDFS Commands

- Interact with the filesystem using the *hdfs dfs* prefix:
- List Files: *hdfs dfs -ls /*
- Create Directory: *hdfs dfs -mkdir /test\_dir*
- Upload File: *hdfs dfs -put local\_file.txt /test\_dir*
- Read File: *hdfs dfs -cat /test\_dir/local\_file.txt*
- Download File: *hdfs dfs -get /hdfs\_file.txt local\_path/*

# Map Reduce

# MapReduce is a programming model

- An implementation for processing and generating large data sets.
- Many real world tasks are expressible in this model.
- Users specify
  - a map function that processes a key/value pair to generate a set of intermediate key/value pairs,
  - a reduce function that merges all intermediate values associated with the same intermediate key.
- MapReduce computation processes
  - many terabytes of data
  - on thousands of machines.
- Runs on a large cluster of commodity machines and is highly scalable.

# MapReduce for programmer

- Programs written in this functional style are automatically parallelized and executed on a large cluster of commodity machines.
- Programmers
  - find it easy to use,
  - easily utilize the resources of a large distributed system,
  - without any experience with parallel and distributed systems
- Hundreds of MapReduce programs have been implemented
- Thousands of MapReduce jobs are executed on Google's clusters every day.
- Takes care of
  - partitioning the input data,
  - scheduling the program's execution across a set of machines,
  - handling machine failures, and
  - managing the required inter-machine communication.

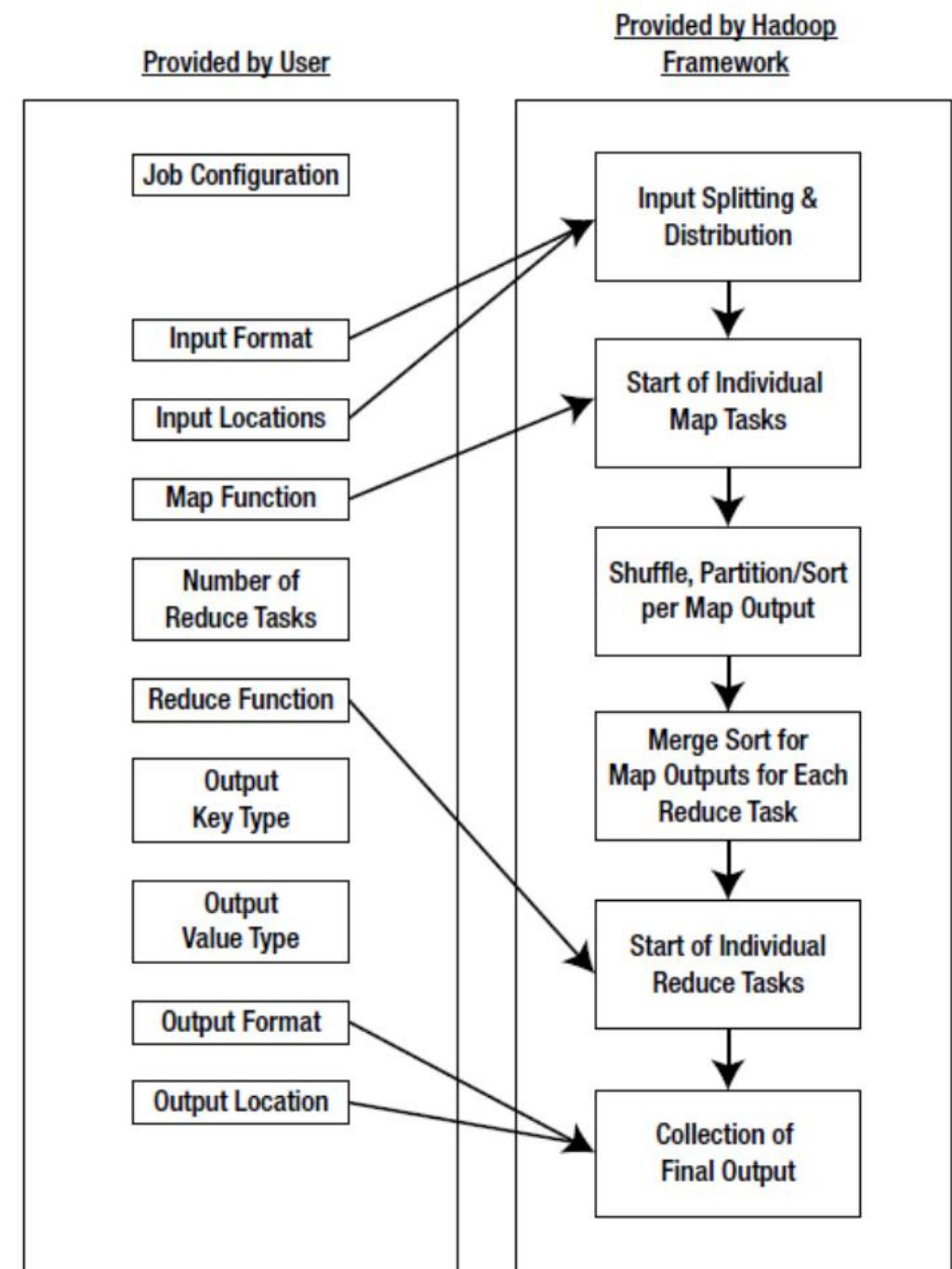
# Typical Large-Data Problem

- Iterate over a large number of records      **Map**
- Extract something of interest from each
- Shuffle and sort intermediate results
- Aggregate intermediate results      **Reduce**
- Generate final output

Key idea: provide a functional abstraction for these two operations – MapReduce

# Map Reduce

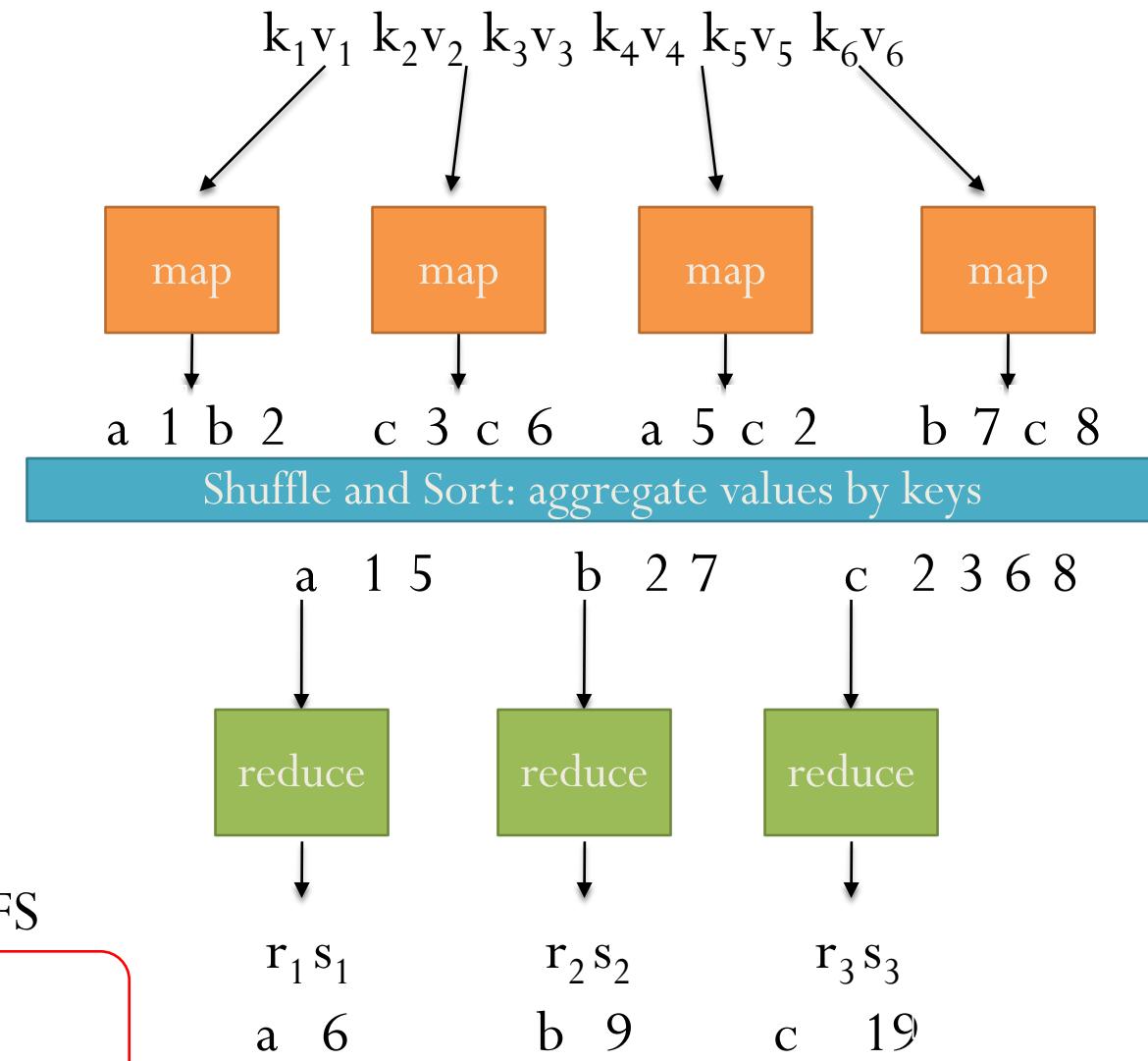
- Format of input- output (key, value)
  - Map:  $(k_1, v_1) \rightarrow \text{list } (k_2, v_2)$
  - Reduce:  $(k_2, \text{list } v_2) \rightarrow \text{list } (k_3, v_3)$
  - All values with the same key are sent to the same reducer
- The execution framework handles everything else
- MapReduce will not work for
  - Inter-process communication
  - Data sharing required
  - Example: Recursive functions



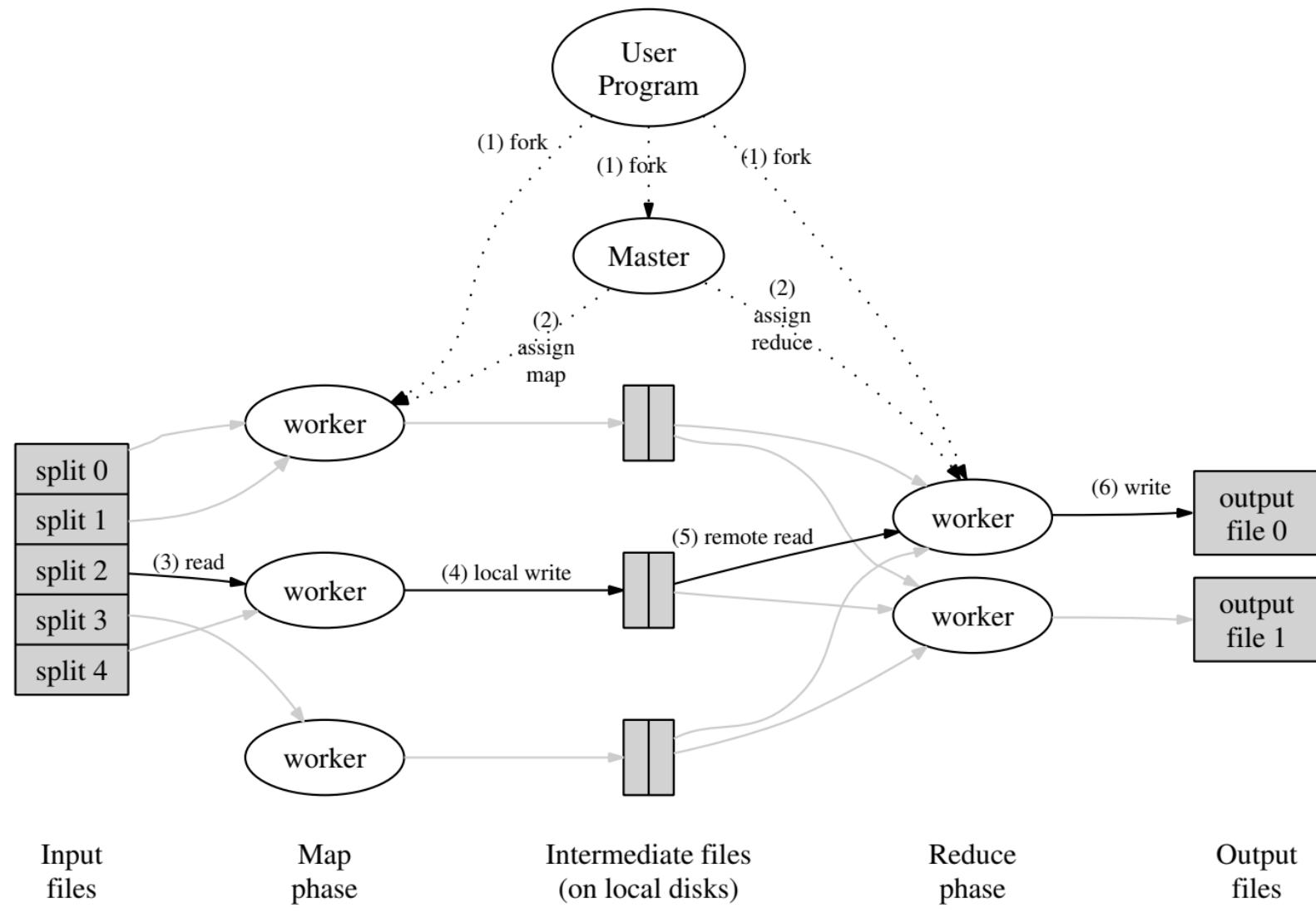
# MapReduce Runtime

- Handles scheduling
  - Assigns workers to map and reduce tasks
- Handles “data distribution”
  - Moves processes to data
- Handles synchronization
  - Gathers, sorts, and shuffles intermediate data
- Handles errors and faults
  - Detects worker failures and automatically restarts
- Handles speculative execution
  - Detects “slow” workers and re-executes work
- Everything happens on top of a Distributed FS

Sounds simple, but many challenges!

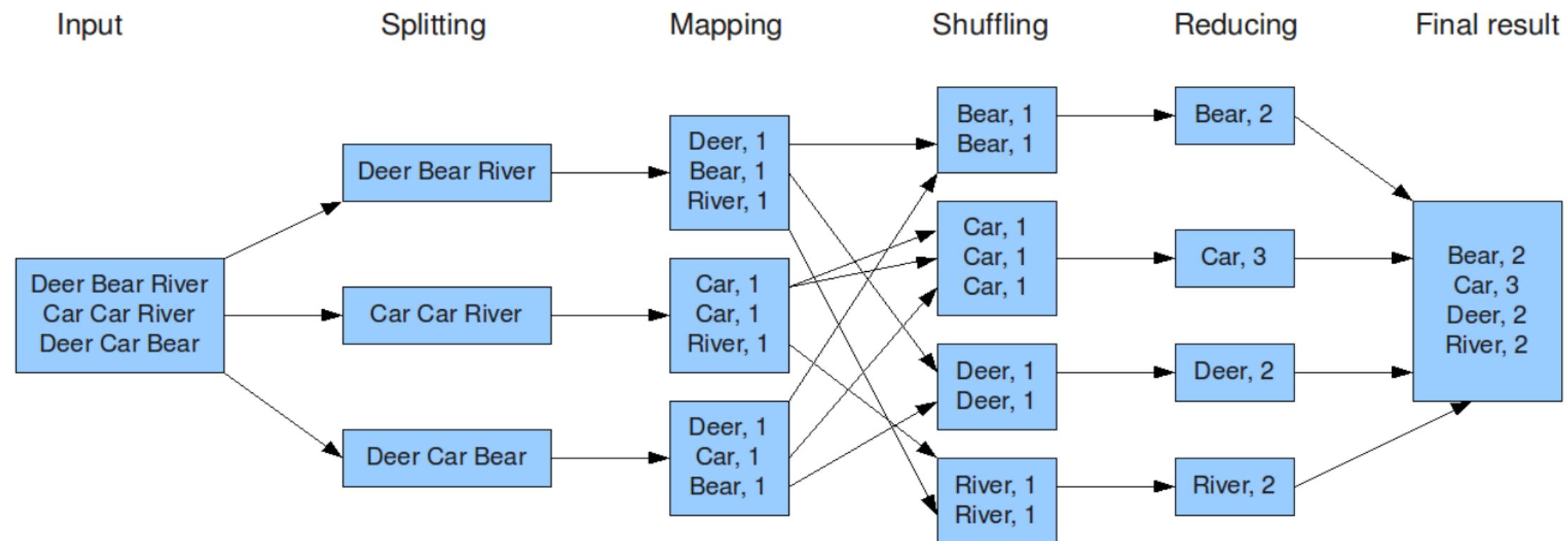


# MapReduce Overall Architecture



# Word Count

- **Map:** Input lines of text to breaks them into words gives outputs for each word  
 $\langle \text{key} = \text{word}, \text{value} = 1 \rangle$
- **Reduce:** Input  $\langle \text{word}, 1 \rangle$  output  $\langle \text{word}, + \text{value} \rangle$



# “Hello World” Example: Word Count

```
map(String key, String value):  
    // key: document name  
    // value: document contents  
    for each word w in value:  
        EmitIntermediate(w, "1");  
  
reduce(String key, Iterator values):  
    // key: a word  
    // values: a list of counts  
    int result = 0;  
    for each v in values:  
        result += ParseInt(v);  
    Emit(AsString(result));
```

# MapReduce Implementations

- Google has a proprietary implementation in C++
  - Bindings in Java, Python
- Hadoop is an open-source implementation in Java
  - Development led by Yahoo, used in production
  - Now an Apache project
  - Rapidly expanding software ecosystem, but still lots of room for improvement
- Lots of custom research implementations issues

# Map Reduce/GFS Summary

- Simple, but powerful programming model
- Scales to handle petabyte+ workloads
  - Google: six hours and two minutes to sort 1PB (10 trillion 100-byte records) on 4,000 computers
  - Yahoo!: 16.25 hours to sort 1PB on 3,800 computers
- Incremental performance improvement with more nodes
- Seamlessly handles failures, but possibly with performance penalties

# MapReduce WordCount code in Python

```
# Mapper: mapper.py (Reads line by line, emits word, 1)
import sys

def mapper():
    for line in sys.stdin:
        words = line.split()
        for word in words:
            print(f"{word}\t1")

# Reducer: reducer.py (Groups by word, sums counts)
def reducer():
    current_word = None
    current_count = 0

    for line in sys.stdin:
        word, count = line.strip().split('\t')
        count = int(count)

        if current_word == word:
            current_count += count
        else:
            if current_word:
                print(f'{current_word}\t{current_count}')
            current_word = word
            current_count = count

    if current_word:
        print(f'{current_word}\t{current_count}'')
```

# MapReduce WordCount code in Java

```
public static class TokenizerMapper extends Mapper<Object, Text, Text, IntWritable> {
    private final static IntWritable one = new IntWritable(1);
    private Text word = new Text();

    public void map(Object key, Text value, Context context) throws IOException,
        StringTokenizer itr = new StringTokenizer(value.toString());
        while (itr.hasMoreTokens()) {
            word.set(itr.nextToken());
            context.write(word, one); // output: <word, 1>
        }
    }

    public static class IntSumReducer extends Reducer<Text, IntWritable, Text, IntWritable> {
        private IntWritable result = new IntWritable();

        public void reduce(Text key, Iterable<IntWritable> values, Context context) {
            int sum = 0;
            for (IntWritable val : values) {
                sum += val.get();
            }
            result.set(sum);
            context.write(key, result); // output: <word, total_count>
        }
    }
}
```

# References

- Cloud Computing: Past, Present, and Future, Professor Anthony D. Joseph, UC Berkeley Reliable Adaptive Distributed systems Lab (RAD lab) UC Berkley <http://abovetheclouds.cs.berkeley.edu/>
- [https://en.wikipedia.org/wiki/Google\\_File\\_System](https://en.wikipedia.org/wiki/Google_File_System)
- <https://sites.google.com/site/gfsassignmentwiki/home>
- Sanjay Ghemawat, Howard Gobioff, and Shun-Tak Leung. "The Google file system." Proceedings of the nineteenth ACM symposium on Operating systems principles. 2003.
- Jeffrey Dean, and Sanjay Ghemawat. "MapReduce: simplified data processing on large clusters." *Communications of the ACM* 51.1 (2008): 107-113.

תודה רבה

Hebrew

Danke

German

Merci

French

Grazie

Italian

Gracias

Spanish

Obrigado

Portuguese

Ευχαριστώ

Greek

Спасибо

Russian

ধন্যবাদ

Bangla

ಧನ್ಯವಾದಗಳು

Kannada

ధన్యవాదాలు

Telugu

ਧੰਨਵਾਦ

Punjabi

धन्यवादः

Sanskrit

*Thank You*

English

நன்றி

Tamil

മന്ത്രി

Malayalam

આમાર

Gujarati

ありがとうございました

Japanese

多謝

Traditional Chinese

多谢

Simplified Chinese

ຂອບຄຸມ

Thai

감사합니다

Korean