Hadoop Distributed File System

Why Hadoop ?

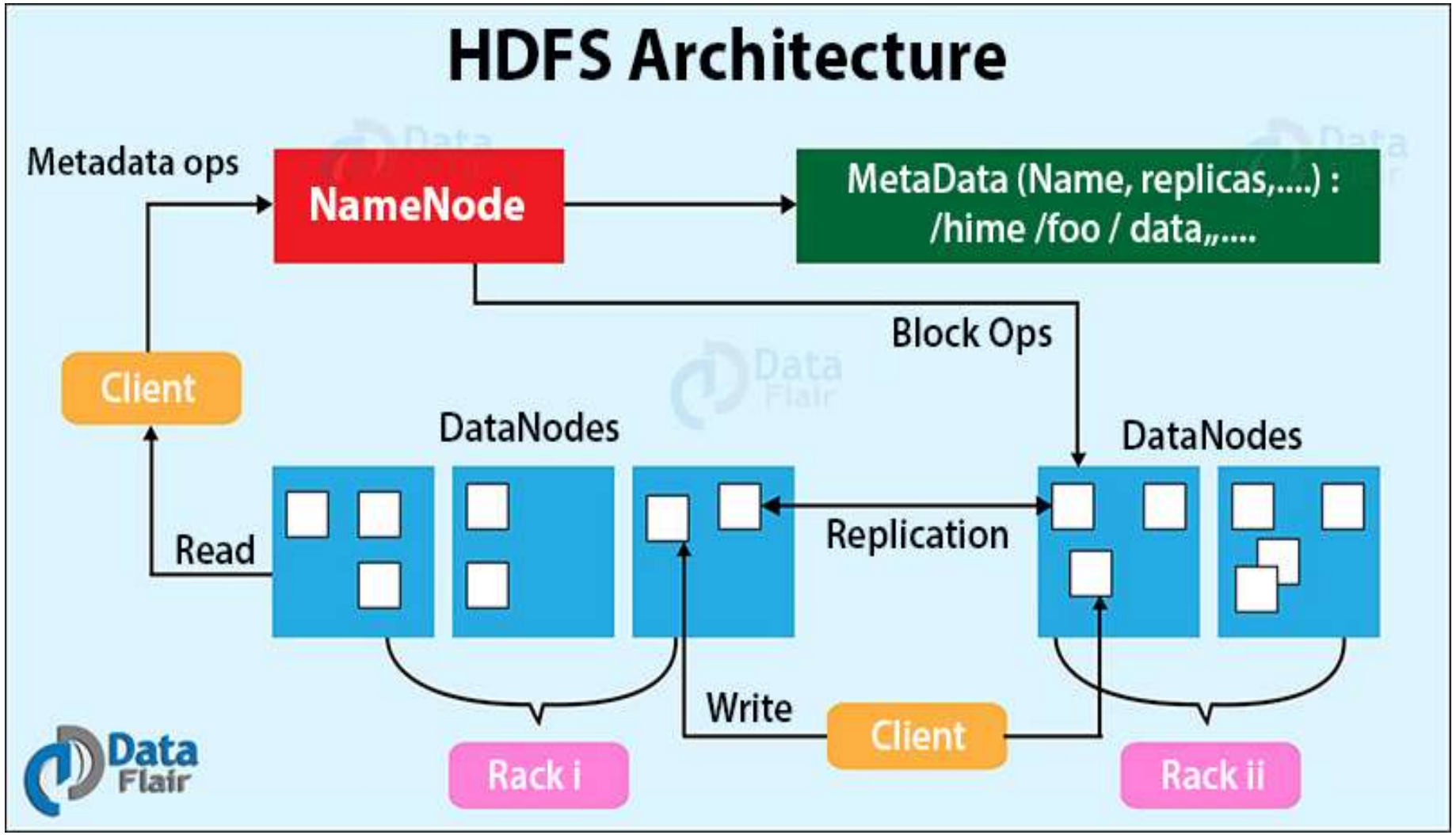
- Need to process huge datasets on large clusters of computers.
- Very expensive to build reliability into each application.
- Nodes fail every day
  - *Failure is expected, rather than exceptional*
  - *The number of nodes in a cluster is not constant*
- Need a common infrastructure
  - *Efficient, reliable, easy to use*
  - *Open Source, Apache Licence*

Introduction

- The Hadoop Distributed File System (HDFS) is a distributed file system designed to run on commodity hardware.
- HDFS is highly fault-tolerant and is designed to be deployed on low-cost hardware.
- HDFS provides high throughput access to application data and is suitable for applications that have large data sets.

Assumptions & Goals

1. Hardware Failure: detection of faults and quick automatic recovery from them is a core architectural goal of HDFS.
2. Streaming Data Access: data is read continuously with a constant bitrate.
3. Large Data Sets
4. Simple Coherency Model: HDFS applications need a write-once-read-many access model for files.
5. "Moving Computation is Cheaper than Moving Data": A computation requested by an application is much more efficient if it is executed near the data it operates on. HDFS provides interfaces for applications to move themselves closer to where the data is located.
6. Portability Across Heterogeneous Hardware and Software Platforms: HDFS has been designed to be easily portable from one platform to another.


HDFS Architecture
Metadata ops
NameNode
MetaData (Name, replicas,....) :
/hime /foo / data„....
Block Ops
Client
DataNodes
DataNodes
Read
Replication
Write
Client
Rack i
Rack ii
Data Flair

HDFS Architecture

- NameNode
- DataNodes
- HDFS Client
- Image Journal
- CheckpointNode
- BackupNode
- Upgrade, File System Snapshots

NameNode – one per cluster

- Maintain The HDFS namespace, a hierarchy of files and directories represented by inodes
- Maintain the mapping of file blocks to DataNodes
  - Read: ask NameNode for the location
  - Write: ask NameNode to nominate DataNodes
- Checkpoint: native files store persistent record of images (no location)
- Stores metadata – filename, location of blocks etc.
- Maintains metadata in memory

Inode: ***a data structure that stores various information about a file in Linux****, such as the access mode (read, write, execute permissions), ownership, file type, file size, group, number of links, etc. Each inode is identified by an integer number. An inode is assigned to a file when it is created.*

DataNodes

- Stores file contents as blocks
- Different blocks of same file are on different data nodes
- Same block is replicated across data nodes
- **Handshake** when connect to the NameNode
  - Verify namespace ID and software version
  - New DN can get one namespace ID when join
- **Register** with NameNode
  - Storage ID is assigned and never changes
  - Storage ID is a unique internal identifier

Contd…

- **Block report:** identify block replicas
  - Block ID, the generation stamp, and the length
  - Send first when register and then send per hour
- **Heartbeats:** message to indicate availability
  - Default interval is three seconds
  - DN is considered “dead” if not received in 10 mins
  - Contains Information for space allocation and load balancing
    - Storage capacity
    - Fraction of storage in use
    - Number of data transfers currently in progress
  - NN replies with instructions to the DN
  - Keep frequent. Scalability

HFDS Client

- A code library exports HDFS interface
- Read a file
  - Ask for a list of DN host replicas of the blocks
  - Contact a DN directly and request transfer
- Write a file
  - Ask NN to choose DNs to host replicas of the first block of the file
  - Organize a pipeline and send the data
  - Iteration
- Delete a file and create/delete directory
- Various APIs
  - Schedule tasks to where the data are located
  - Set replication factor (number of replicas)

Contd…

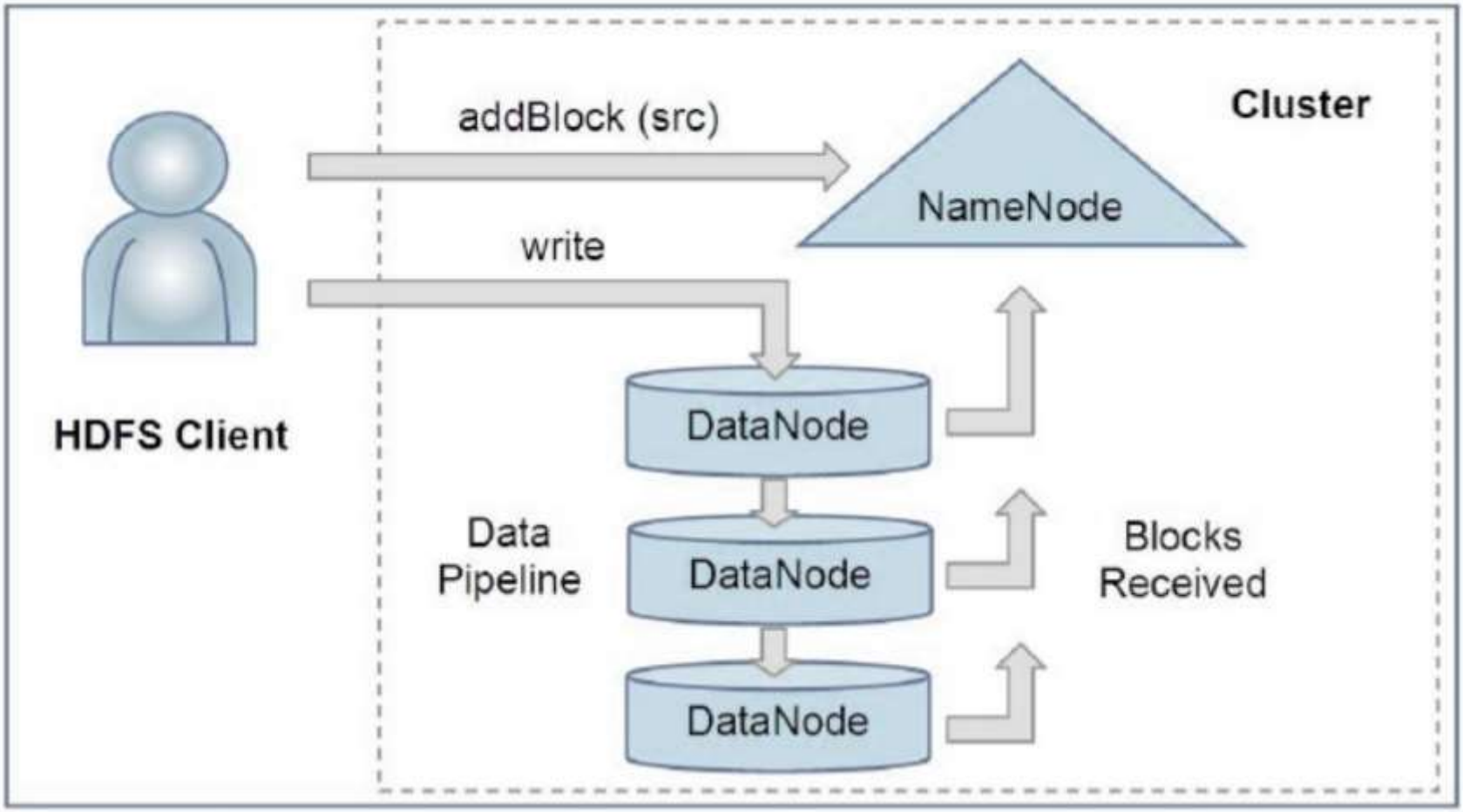

addBlock (src)
Cluster
NameNode
write
HDFS Client
DataNode
Data
Pipeline
DataNode
Blocks
Received
DataNode

Image and Journal

- **Image:** metadata describe organization
  - Persistent record is called checkpoint
  - Checkpoint is never changed, and can be replaced
- **Journal:** log for persistence changes
  - Flushed and synched before change is committed
- Store in multiple places to prevent missing
  - NN shut down if no place is available
- **Bottleneck:** threads wait for flush-and-sync
  - Solution: batch

CheckpointNode

- CheckpointNode is NameNode
- Runs on different host
- Create new checkpoint
  - Download current checkpoint and journal
  - Merge
  - Create new and return to NameNode
  - NameNode truncate the tail of the journal
- **Challenge:** large journal makes restart slow
  - Solution: create a daily checkpoint

BackupNode

- Recent feature
- Similar to CheckpointNode
- Maintain an in memory, up-to-date image
  - Create checkpoint without downloading
- Journal store
- Read-only NameNode
  - All metadata information except block locations
  - No modification

Upgrades, File System and Snapshots

- Minimize damage to data during upgrade
- **NameNode**
  - Merge current checkpoint and journal in memory
  - Create new checkpoint and journal in a new place
  - Instruct DataNodes to create a local snapshot
- **DataNode**
  - Create a copy of storage directory
  - Hard link (mirror copy) existing block files

Some More Points: HDFS Architecture

- 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.
- In addition, there are a number of DataNodes 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.
- Internally, a file is split into one or more blocks and these blocks are stored in a set of DataNodes.
- The NameNode executes file system namespace operations like opening, closing, and renaming files and directories.

Contd…

- 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.

Important Points:

- Operates on top of an existing file system.
- Files are stored as blocks
  - *Default size is 64 MB*
- Reliability through replication.
- NameNode stores metadata and manages access.
- No caching due to large file sizes.

Failure and Recovery

- NameNodes keep track of DataNodes through periodic HeartBeat messages
- If no heartbeat is received for a certain duration, DataNode is assumed to be lost
  - NameNode determines which blocks were lost
  - Replicates the same on other DataNodes
- NameNode failure = File system failure
- Two options
  - Persistent backup and checkpointing
  - Secondary/backup NameNode

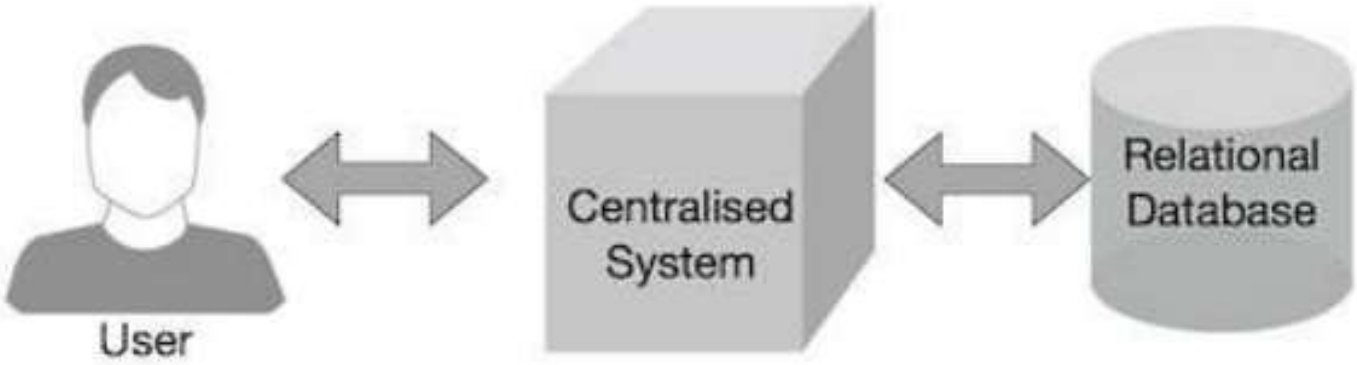
Replication Policy

- **Rule1:** No DataNode contains more than one replica of any block
- **Rule2:** No rack contains more than two replicas of the same block, provided there are sufficient racks on the cluster

Introduction to MapReduce

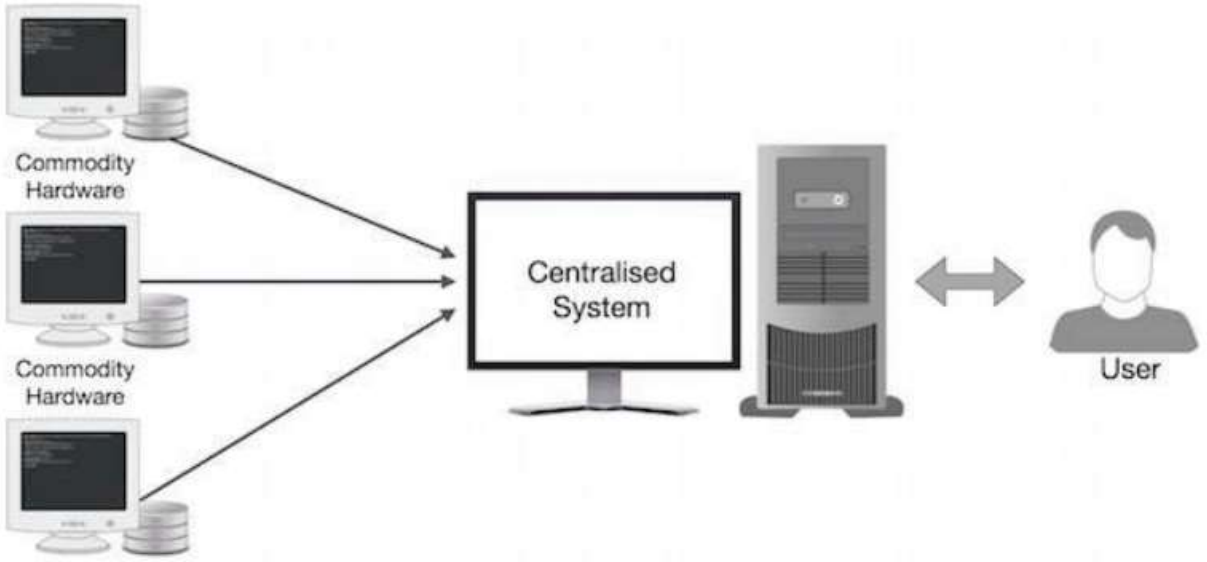
Why MapReduce?

- Traditional Enterprise Systems normally have a centralized server to store and process data.
- Moreover, the centralized system creates too much of a bottleneck while processing multiple files simultaneously.



Contd…

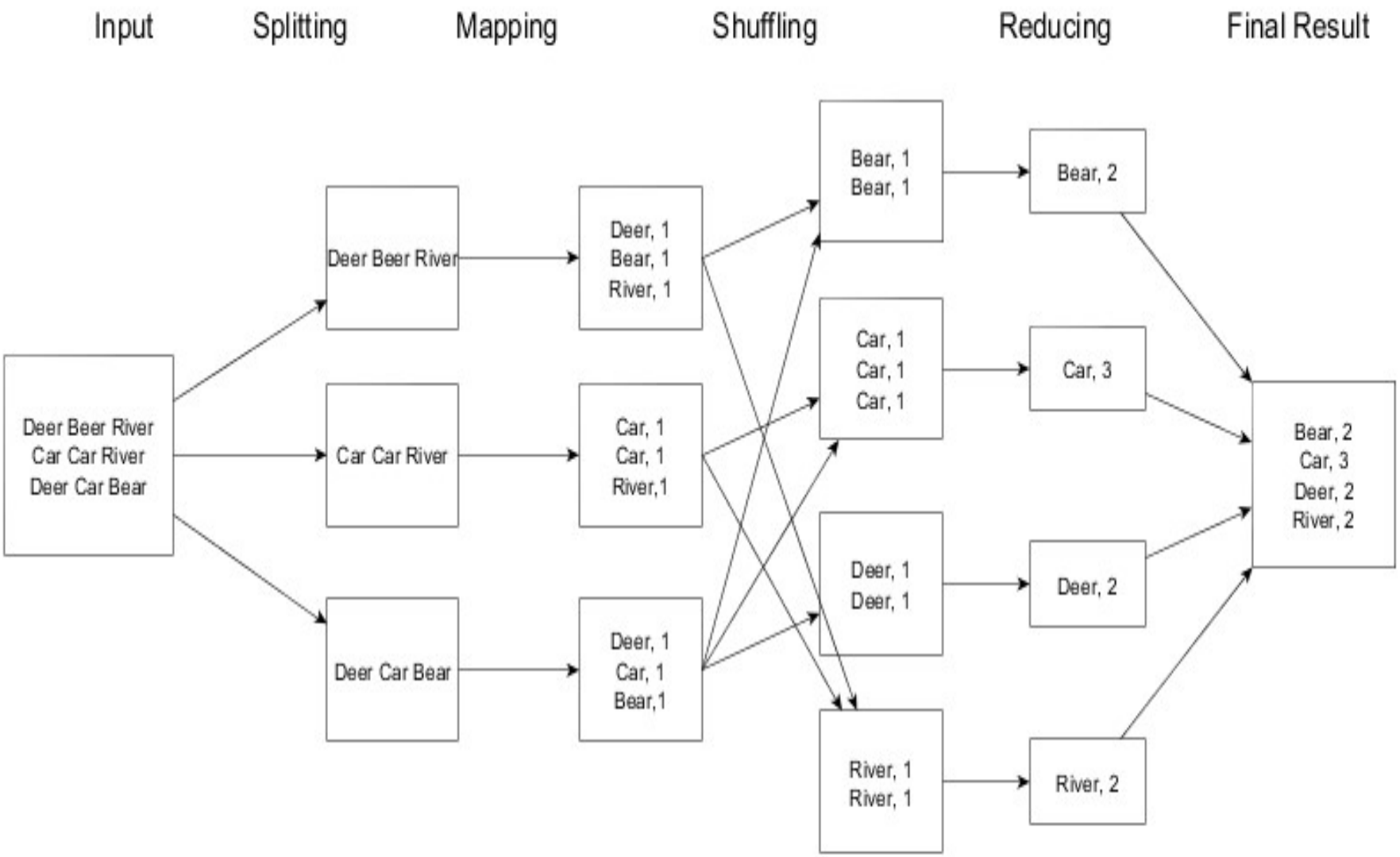
- Google solved this bottleneck issue using an algorithm called MapReduce.
- MapReduce divides a task into small parts and assigns them to many computers.
- Later, the results are collected at one place and integrated to form the result dataset.



How MapReduce Works?

- MapReduce is a software framework for processing large datasets in a distributed fashion over several machines.
  - Map data into multiple <key, value> pairs
  - Reduce over all pairs with the same key.
- Consider a simple example of finding the count of every word present in a document.

Word Count using MapReduce



References

[1] https://cs.uwaterloo.ca/~david/cs848s13/alex-presentation.pdf

[2] https://hadoop.apache.org/docs/r1.2.1/hdfs_design.html

[3]https://hci.stanford.edu/courses/cs448g/a2/files/map_reduce_tutorial.pdf