HADOOP

Hadoop Map Reduce

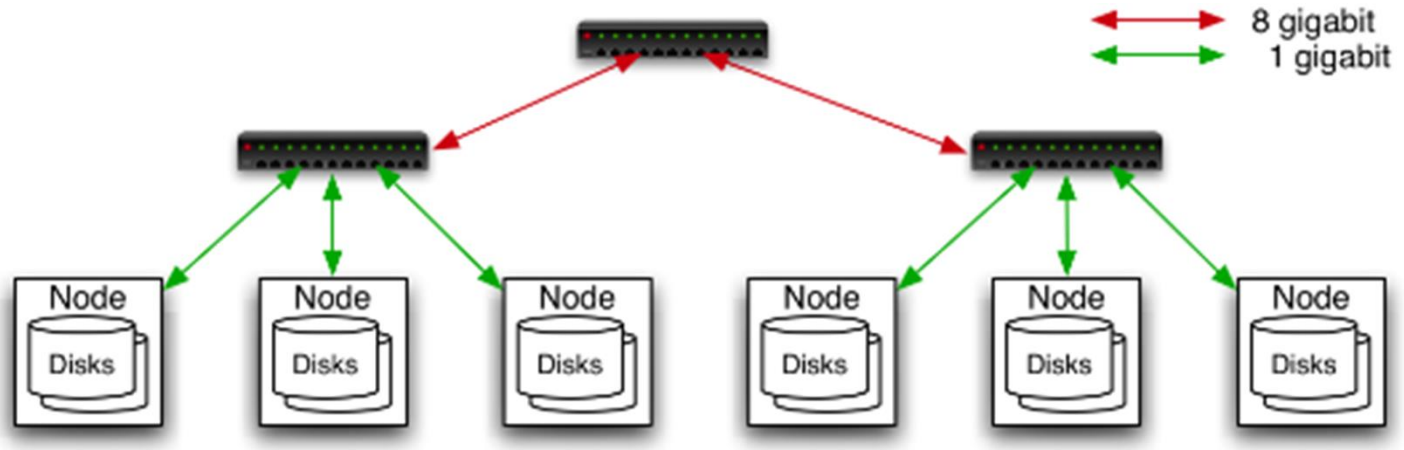
Introduction

- Open source project written in Java
- Large scale distributed data processing
- Based on Google's Map Reduce framework and Google file system
- Works on commodity hardware
- Used by a Google, Yahoo, Facebook, Amazon, and many other startups
  http://wiki.apache.org/hadoop/PoweredBy

Hadoop Core

- Hadoop Distributed File System (HDFS)
  - Distributes and stores data across a cluster (brief intro only)
- Hadoop Map Reduce (MR)
  - Provides a parallel programming model
  - Moves computation to where the data is
  - Handles scheduling, fault tolerance
  - Status reporting and monitoring

Typical Cluster



- Nodes are Linux PCs
- 4-8GB RAM ~100s of GB IDE/SATA drives

GFS

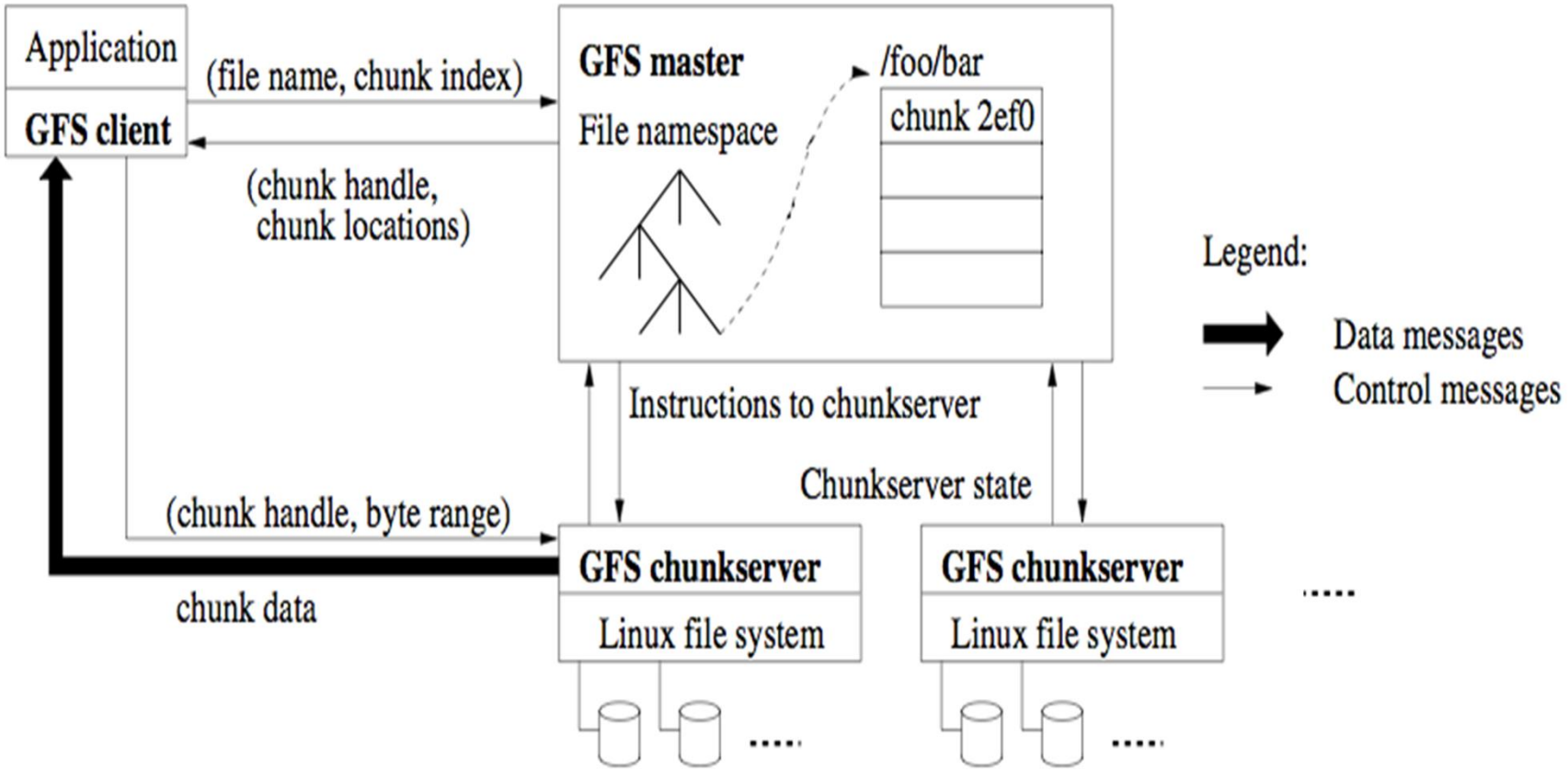
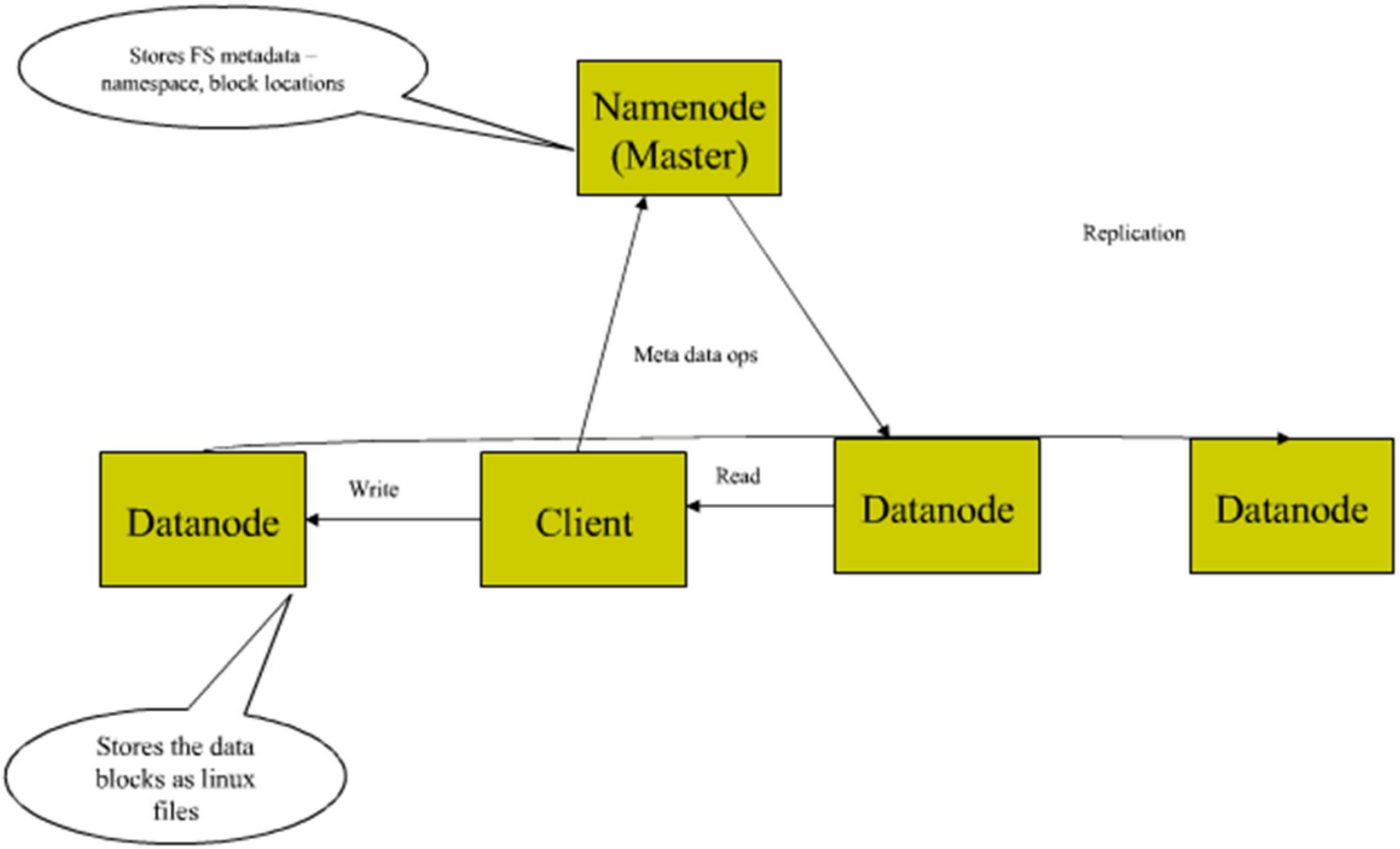


Figure 1: GFS Architecture

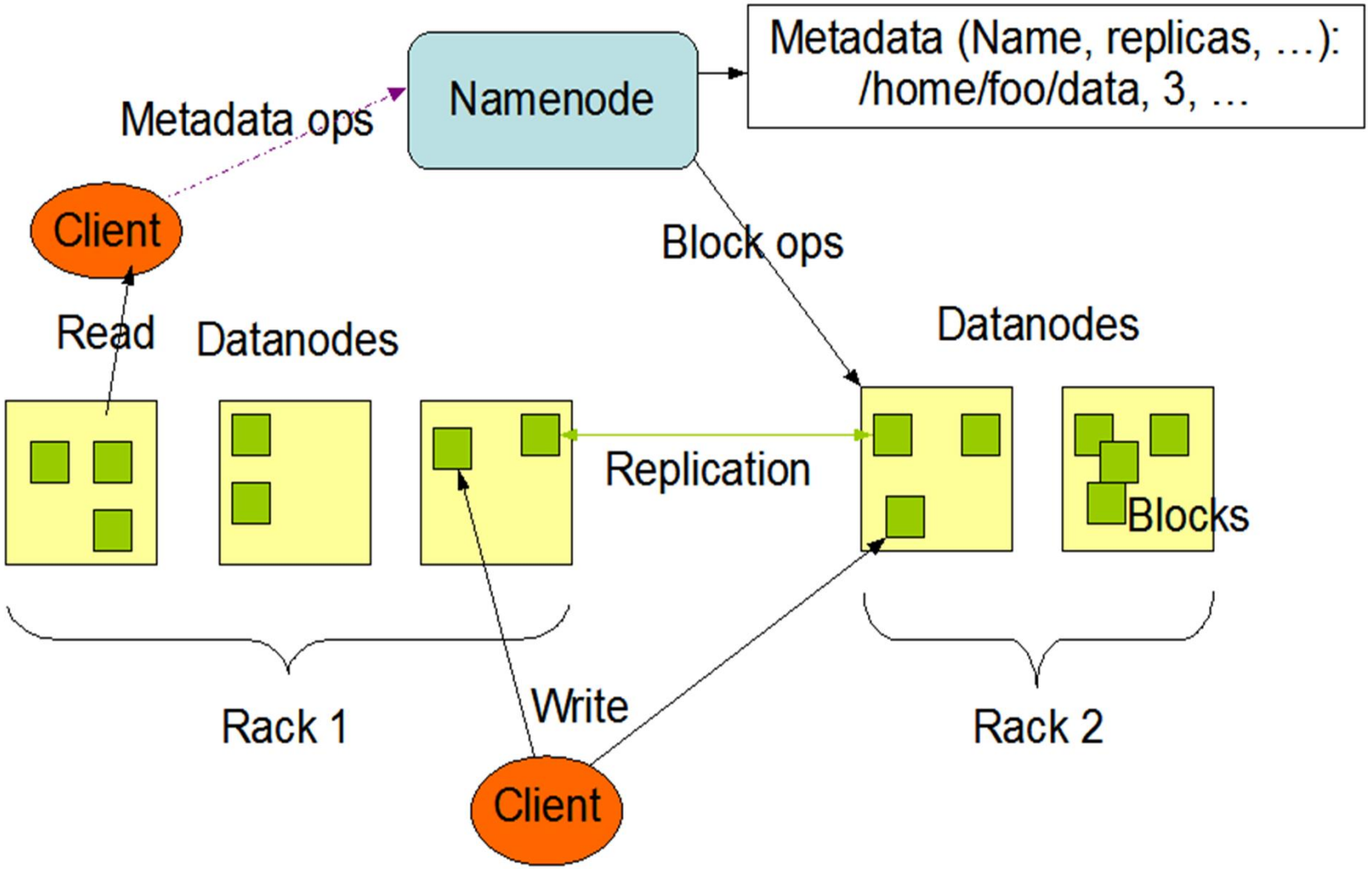
HADOOP Distributed File System

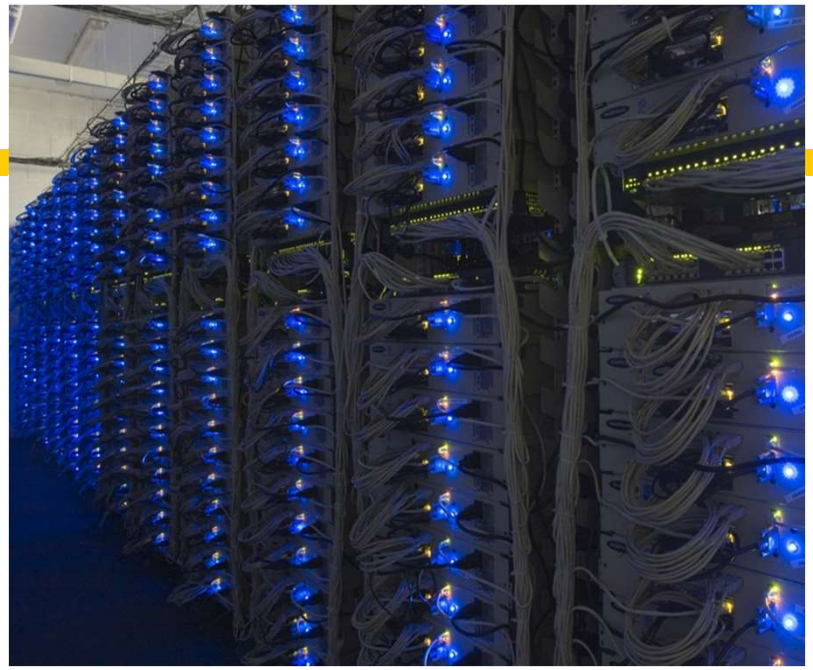
- Scale to Petabytes across 1000s of nodes
- Single namespace for entire cluster
- Files broken into 128MB blocks
- Block level replication handles node failure
- Optimized for single write multiple reads
- Writes are append only

HDFS Architecture



Hadoop Architecture





Word Count Problem

- Find the frequency of each word in a given corpus of documents
- Trivial for small data
- How to process more than a TB of data
- Doing it on one machine is very slow – takes days to finish!
- Good News : It can be parallelized across number of machines

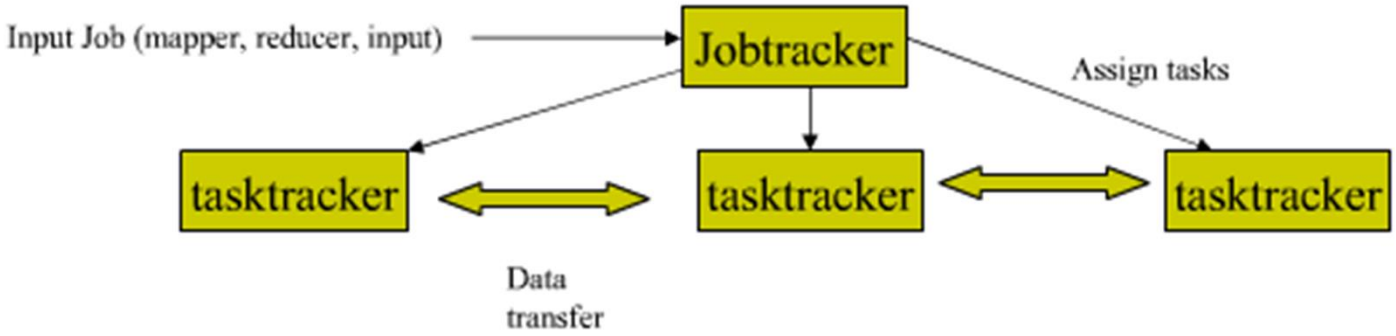
Hadoop Map Reduce

- Why Map Reduce
- Map Reduce Architecture - 1
- Map Reduce Programming Model
- Word count using Map Reduce
- Map Reduce Architecture - 2

Why Map Reduce

- How to scale large data processing applications ?
- Divide the data and process on many nodes
- Each such application has to handle
  - Communication between nodes
  - Division and scheduling of work
  - fault tolerance
  - monitoring and reporting
- Map Reduce handles and hides all these issues
- Provides a clean abstraction for programmer

Map Reduce Architecture



- Each node is part of a HDFS cluster.
- Input data is stored in HDFS spread across nodes and replicated
- Programmer submits job (mapper, reducer, input) to Job tracker
- Job tracker - Master
  - splits input data
  - Schedules and monitors various map and reduce tasks
- Task tracker - Slaves
  - Execute map and reduce tasks

Map Reduce Programming Model

- Inspired by functional language primitives
- ***map*** *f list* : applies a given function *f* to a each element of list and returns a new list

  map *square [1 2 3 4 5] = [1 4 9 16 25]*
- ***reduce*** *g list* : combines elements of list using function *g* to generate a new value

  reduce *sum [1 2 3 4 5] = [15]*
- Map and reduce do not modify input data. They always create new data
- A Hadoop Map Reduce job consists of a mapper and a reducer

Map Reduce Programming Model

- **Mapper**
  - **Input:** <key:, offset, value:line of a document>
  - **Output:** for each word w in input line output<key: w, value:1>

    Input: (2133, *The quick brown fox jumps over the lazy dog.)*

    Output: (the, 1) , (quick, 1), (brown, 1) … (fox,1), (the, 1)
- **Reducer**
  - **Input:** <key: word, value: list<integer>>
  - **Output:** sum all values from input for the given key input list of values

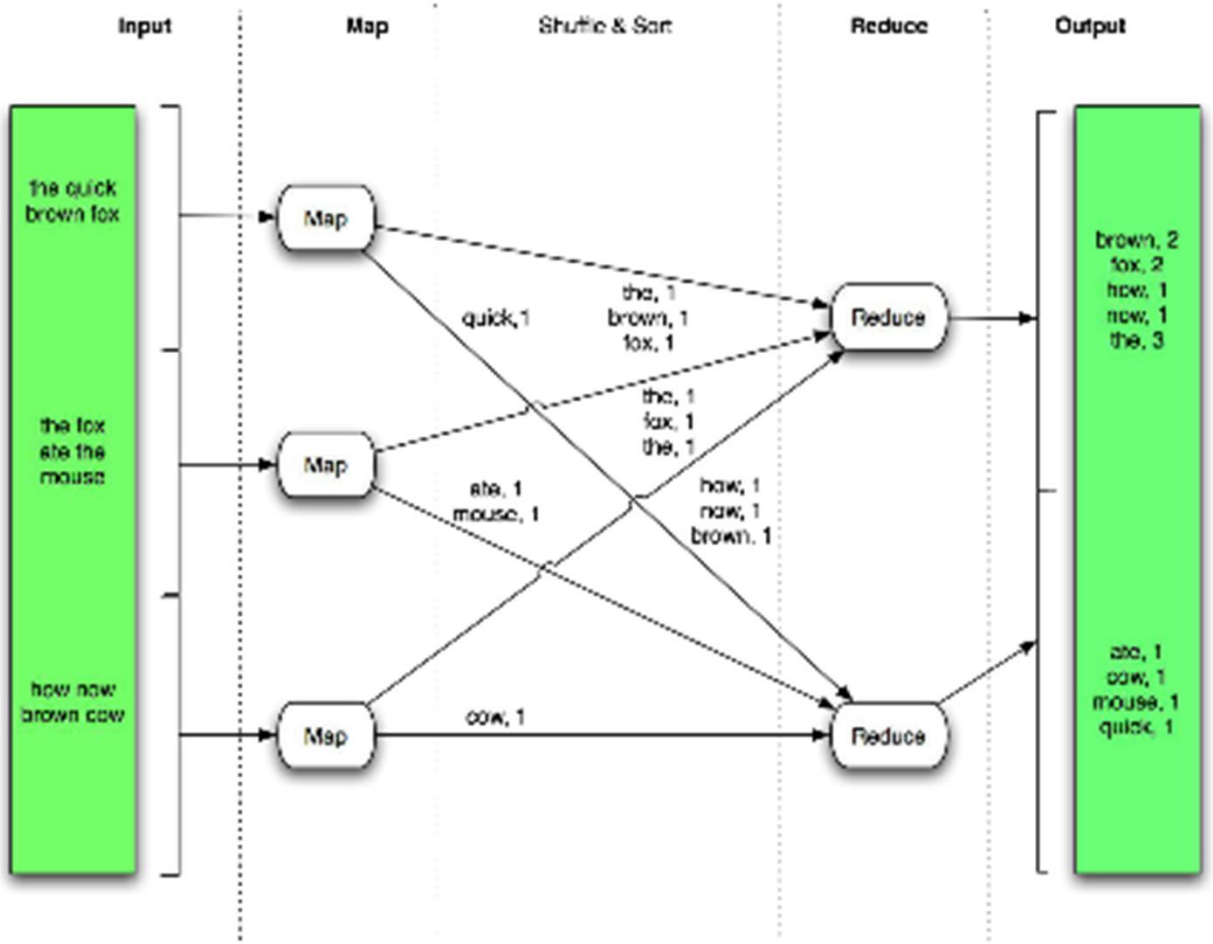
    and output <Key:word value:count>

    Input: (the, [1, 1, 1, 1,1]), (fox, [1, 1, 1]) …

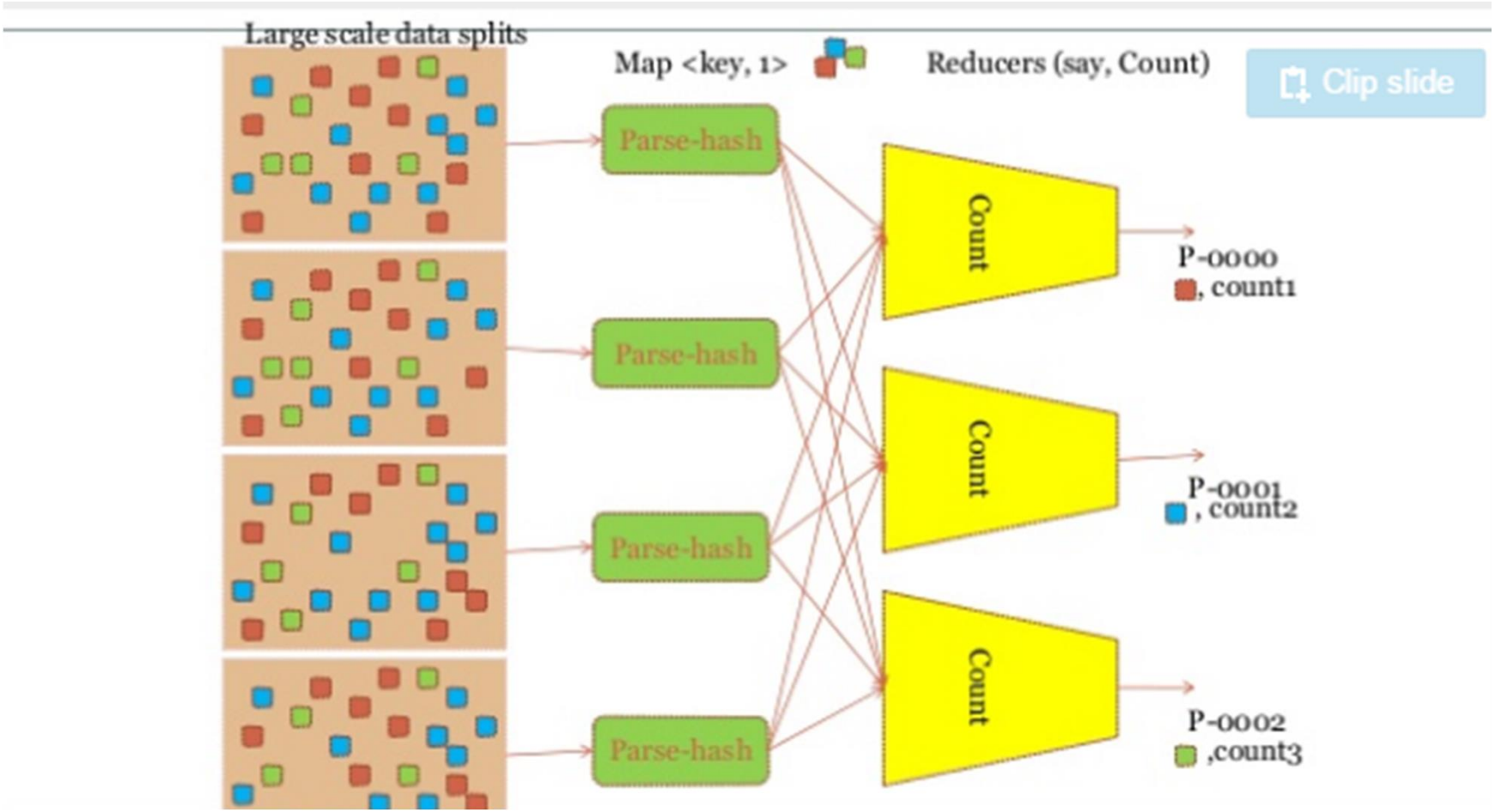
    Output: (the, 5)

    (fox, 3)

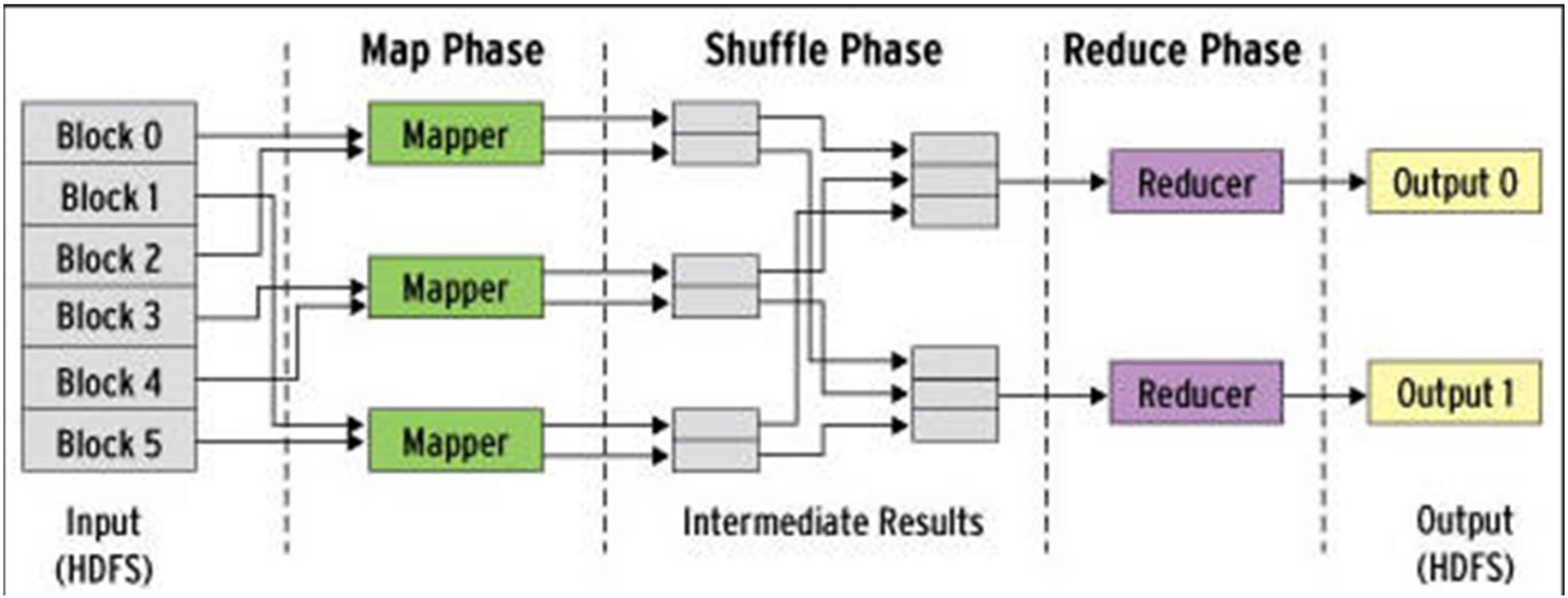
Word Count using Map Reduce



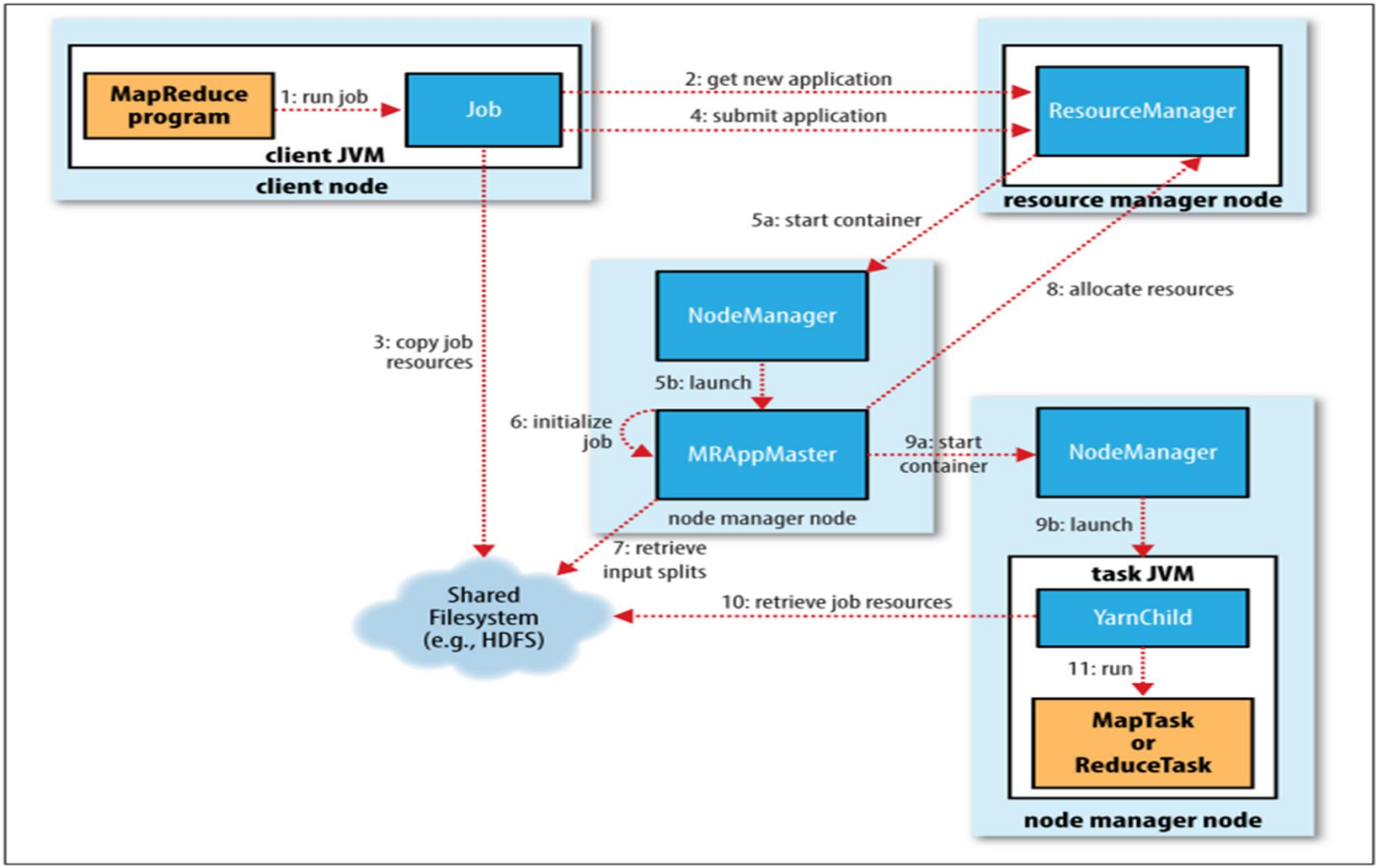
Map Reduce


Large scale data splits
Map <key, 1>
Reducers (say, Count)
Clip slide
Parse-hash
Parse-hash
Parse-hash
Parse-hash
Count
Count
Count
P-0000
, count1
P-0001
, count2
P-0002
,count3

Map Reduce


Map Phase
Shuffle Phase
Reduce Phase
Block 0
Block 1
Block 2
Block 3
Block 4
Block 5
Mapper
Mapper
Mapper
Reducer
Reducer
Output 0
Output 1
Input
(HDFS)
Intermediate Results
Output
(HDFS)

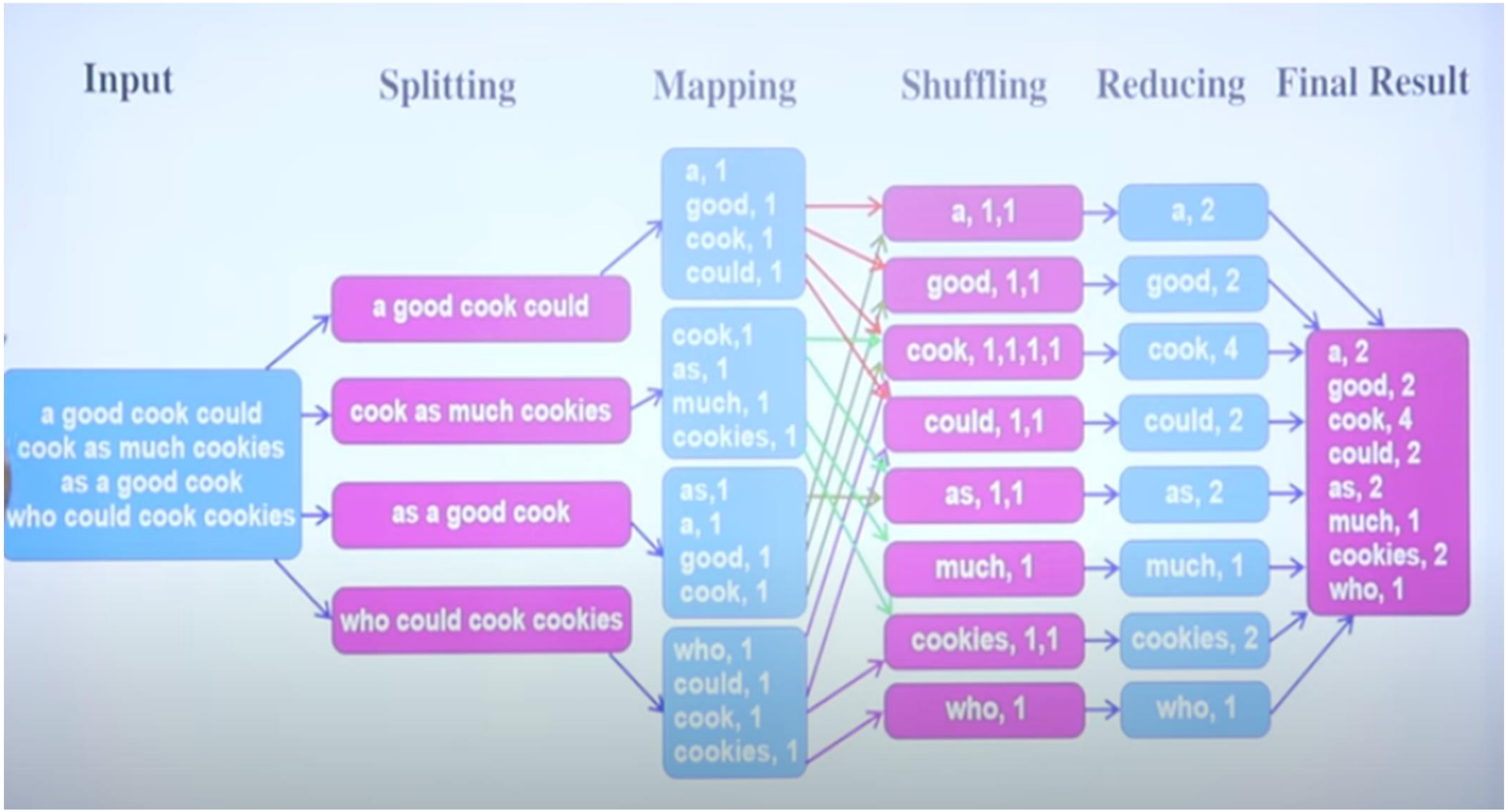
Map Reduce


MapReduce program
1: run job
Job
client JVM
client node
2: get new application
4: submit application
ResourceManager
resource manager node
5a: start container
NodeManager
5b: launch
8: allocate resources
3: copy job resources
6: initialize job
MRAppMaster
node manager node
9a: start container
NodeManager
9b: launch
task JVM
YarnChild
11: run
MapTask or ReduceTask
node manager node
7: retrieve input splits
Shared Filesystem (e.g., HDFS)
10: retrieve job resources

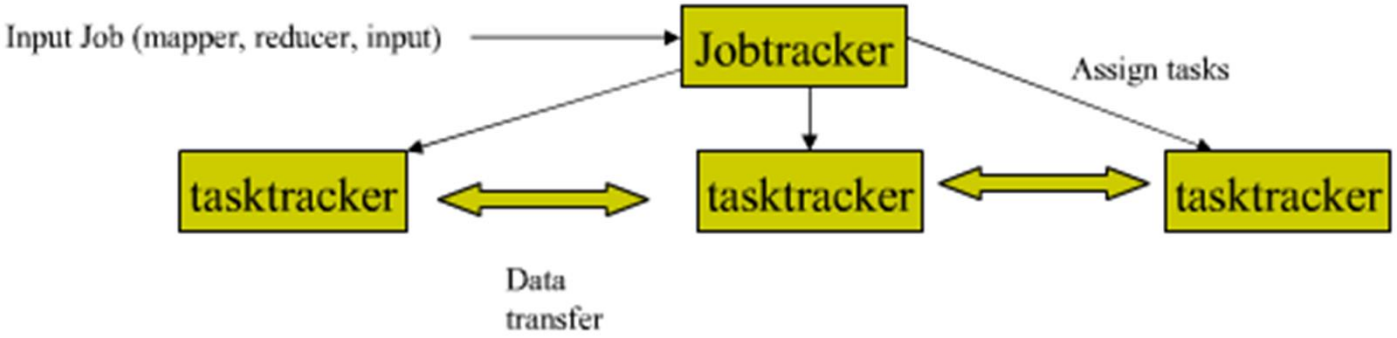
Example: Word Count

http://kickstarthadoop.blogspot.ca/2011/04/word-count-hadoop-map-reduce-example.html

Why Map Reduce



Map Reduce Architecture -2



- Job tracker
  - splits input and assigns to various map tasks
  - Schedules and monitors map tasks (heartbeat)
  - On completion, schedules reduce tasks
- Task tracker
  - Execute map tasks – call mapper for every input record
  - Execute reduce tasks – call reducer for every intermediate key, list of values pair
  - Handle partitioning of map outputs
  - Handle sorting and grouping of reducer input

Map Reduce Advantages

- **Locality**
  - Job tracker divides tasks based on location of data: it tries to schedule map tasks on same machine that has the physical data
- **Parallelism**
  - Map tasks run in parallel working different input data splits
  - Reduce tasks run in parallel working on different intermediate keys
  - Reduce tasks wait until all map tasks are finished
- **Fault tolerance**
  - Job tracker maintains a heartbeat with task trackers
  - Failures are handled by re-execution
  - If a task tracker node fails then all tasks scheduled on it (completed or incomplete) are re-executed on another node