Little data BIG DATA

Summary

Introduction to Hadoop
https://hadoop.apache.org

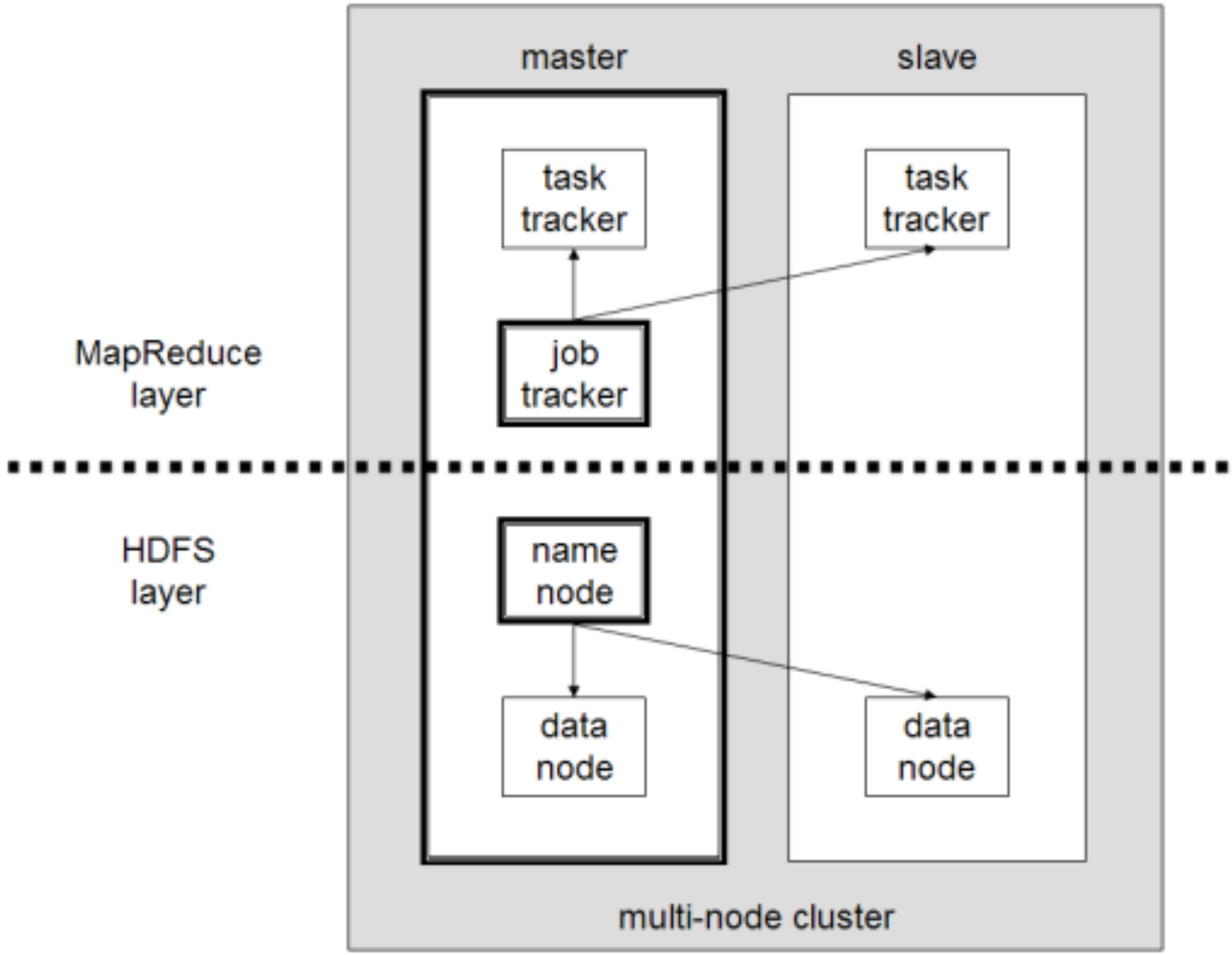
Overview of HDFS and MapReduce

Suitable problems

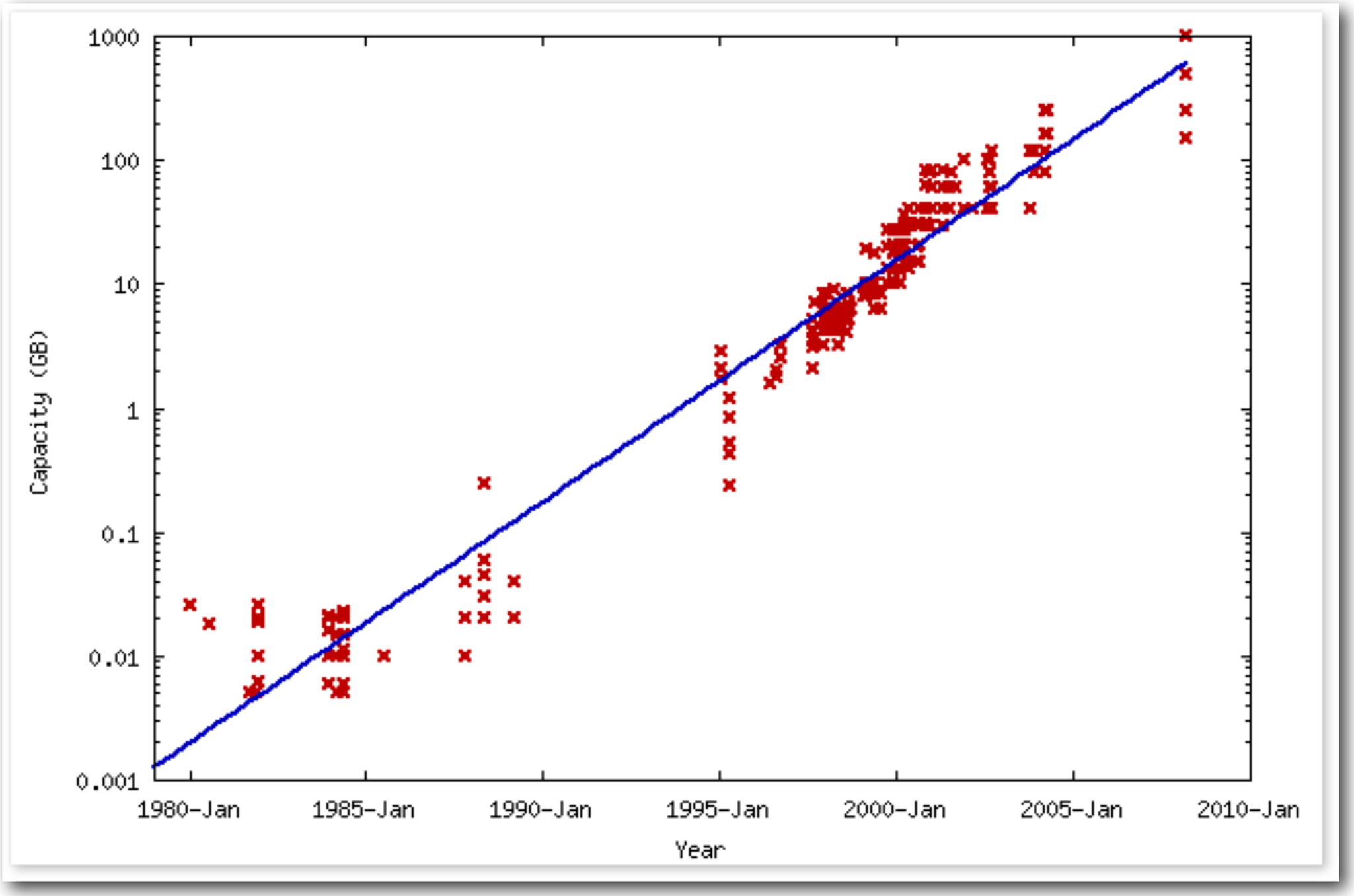
How the Mapper and Reducer work

Shuffle & Sort

Pseudocode for a simple Hadoop programme


master
slave
task tracker
task tracker
job tracker
MapReduce layer
HDFS layer
name node
data node
data node
multi-node cluster

Storage Capacity of Hard Drives



https://en.wikipedia.org/wiki/History_of_hard_disk_drives

Access Speeds?

1990:
Typical drive ~1370MB
Transfer speed ~ 4.4MB/s

⇒ read drive in 5 mins

Access Speeds?

1990:
Typical drive ~1370MB
Transfer speed ~ 4.4MB/s

⇒ read drive in 5 mins

2010:
Typical drive ~1TB
Transfer speed ~ 100MB/s

⇒ read drive in 2.5 hrs

In pioneer days they used oxen for heavy pulling, and when one ox couldn't budge a log, they didn't try to grow a **larger ox**.

We shouldn't be trying for bigger computers, **but for more systems of computers**.

- Grace Hopper (1906 - 1992)

Access Speeds?

1990:
Typical drive ~1370MB
Transfer speed ~ 4.4MB/s

⇒ read drive in 5 mins

1TB =100 x10GB hard drives

Transfer speed ~ 100MB/s

⇒ read data in < 2 mins

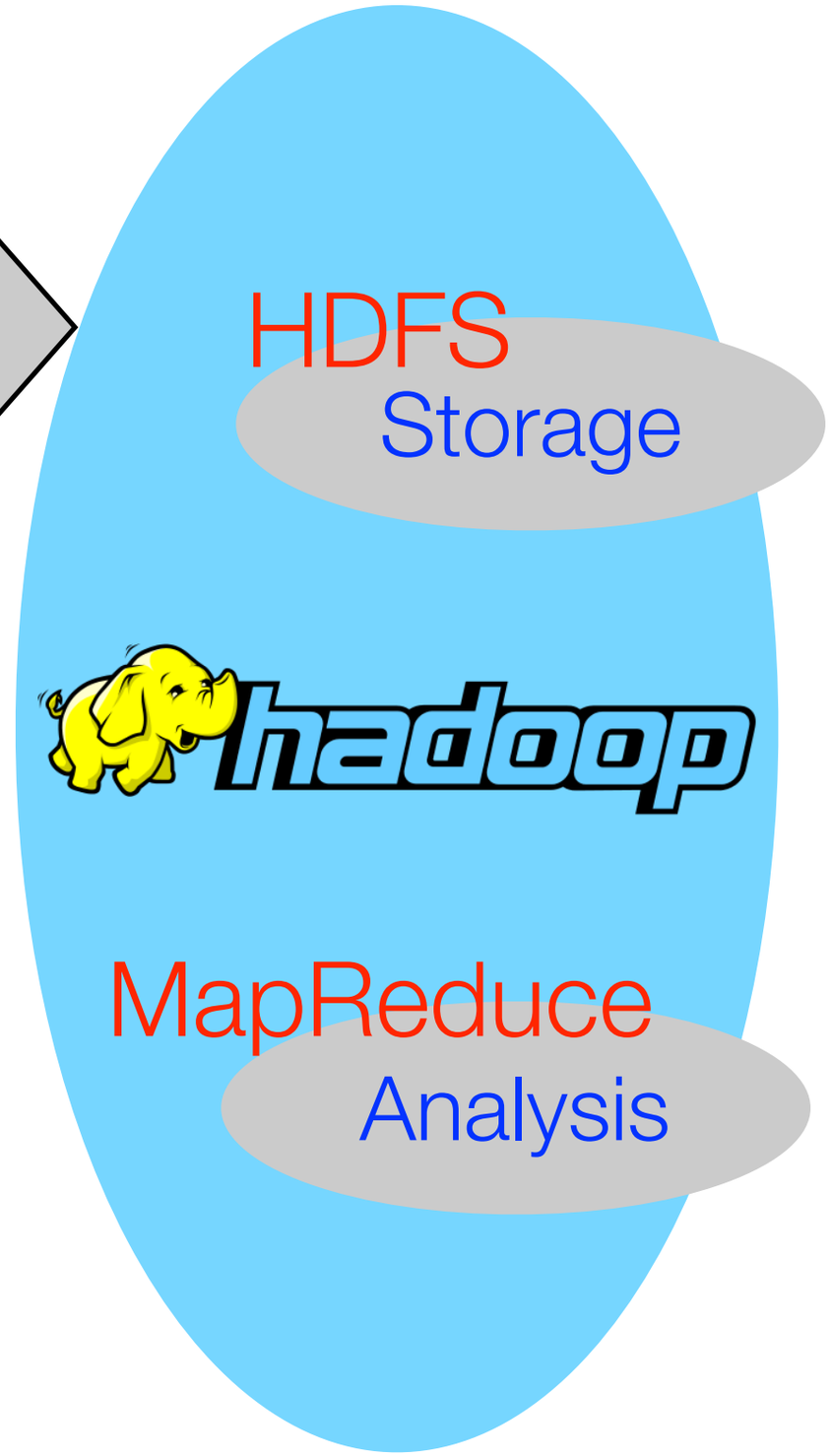
2010:
Typical drive ~1TB
Transfer speed ~ 100MB/s

⇒ read drive in 2.5 hrs

Challenges

Hardware Failure → Replication → HDFS Storage

How to combine data from multiple disks? → MapReduce Analysis

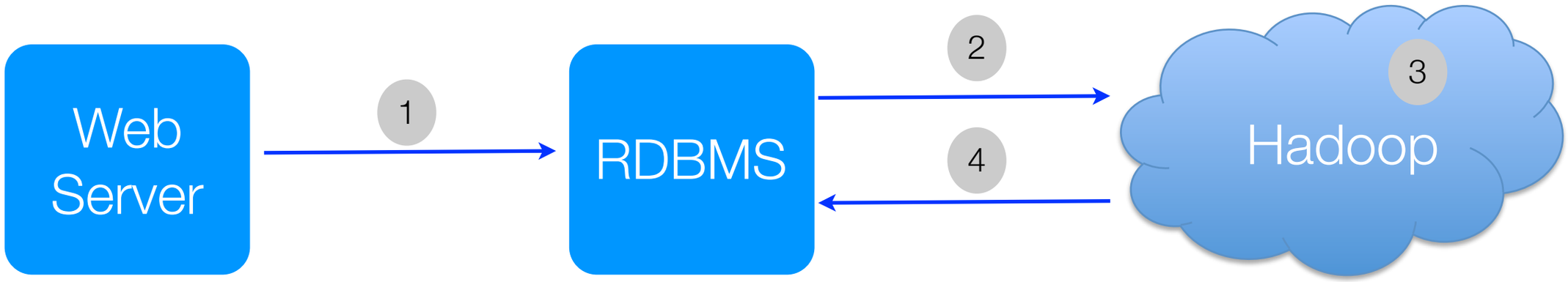


	RDBMS	**Hadoop**
Data size	Gigabytes	Petabytes
Access	Interactive & Batch	Batch
Updates	Read & write many times	Write once, read many times
Integrity	High	Low
Scaling	Non Linear	Linear
Data representation	Structured	Unstructured, semi-structured

RDBMS vs Hadoop

Frequently complement each other

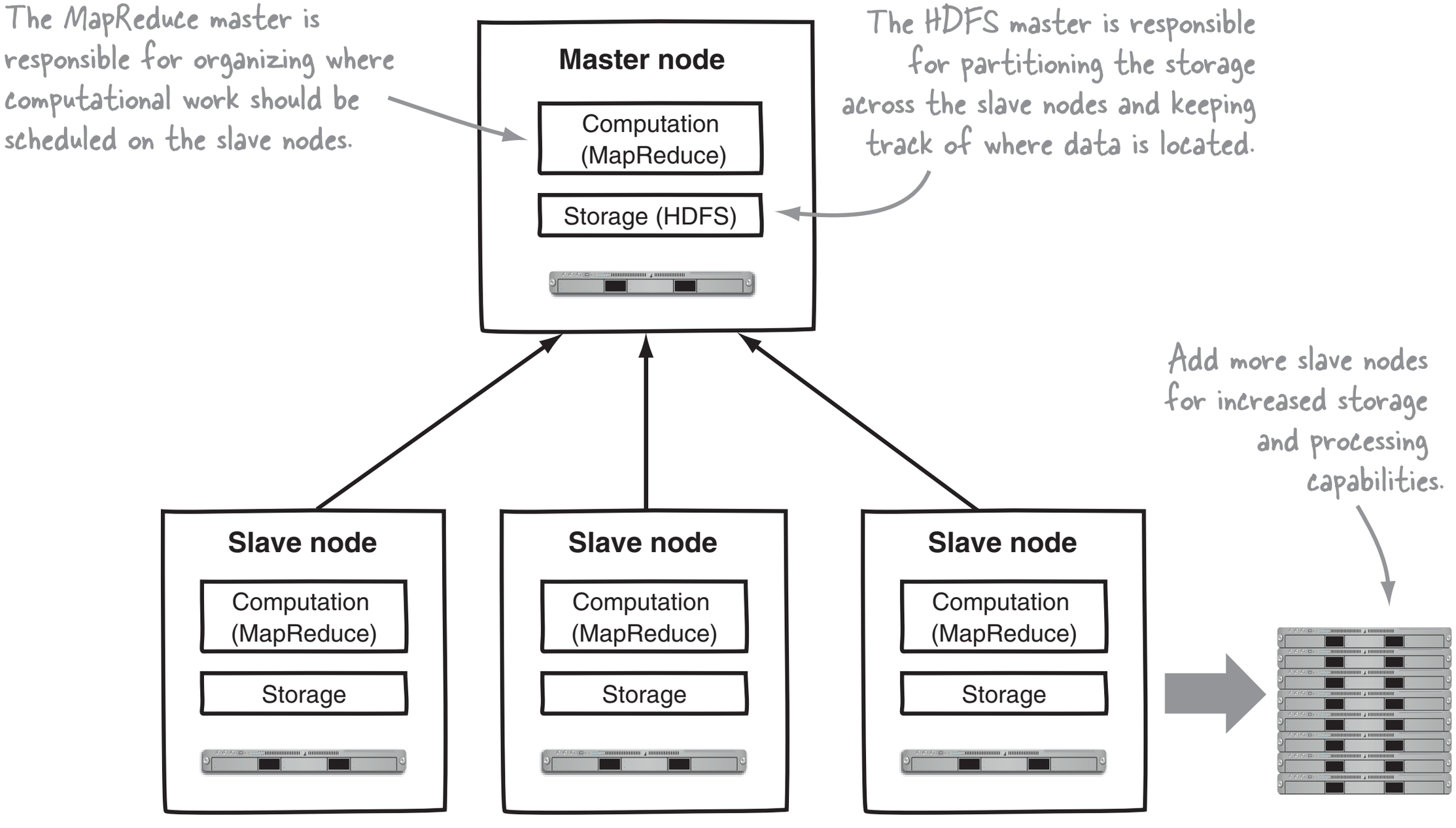
e.g. small number of users that produces a large number of logs



1. Use RDBMS to enforce data integrity on UI
2. Logs are moved periodically to cluster
3. Analytics performed as a batch on logs in Hadoop
4. Results moved back to RDBMS to enhance website, e.g. suggestions based on audit history

Hadoop Master-Slave Architecture

Hadoop Distributed File Storage (HDFS) for storage and MapReduce for computational capabilities



[from Hadoop in Practice, Alex Holmes]

Hadoop Architecture

A set of machines running HDFS and MapReduce is known as a **Hadoop Cluster**

- Individual machines are known as ***nodes***
- A cluster can have as few as **one node** and as many as **several thousand nodes**

Hadoop - "*move-code-to-data*" approach

Data is distributed among the nodes as it is initially stored

Data is replicated multiple times on the system for increased reliability & availability

Master allocates work to nodes

Computation happens on nodes where data is stored - *data locality*

Nodes work **in parallel** each on their own part of the overall dataset

Nodes independent and self-sufficient

If a node fails, master detects the failure and re-assigns work to other nodes

If a failed node restarts, it is automatically added back into the system and assigned new tasks

HDFS

Distributed file system modelled on Google File System (GFS)
[http://research.google.com/archive/gfs.html]

Data is split into blocks, typically 64MB or 128MB in size, spread across many nodes
- Works better on large files **>= 1 HDFS block in size**

Each block is **replicated** to a number of nodes (typically 3)
- ensures reliability and availability

Files in HDFS are write once - no random writes to files allowed

HDFS is optimised for **large streaming reads of files - no random access to files allowed**

HDFS Nodes

Two types of nodes in a HDFS cluster

NameNode - the master node

DataNodes - slave or worker nodes

NameNode manages the file system

- keeps track of the **metadata** - which blocks make up a file
- knows on which DataNodes the blocks are **stored**

DataNodes do the work

- store the blocks
- retrieve blocks when requested to (by the client or the NameNode)
- poll and report back to the NameNode periodically with the list of blocks that they are storing

HDFS

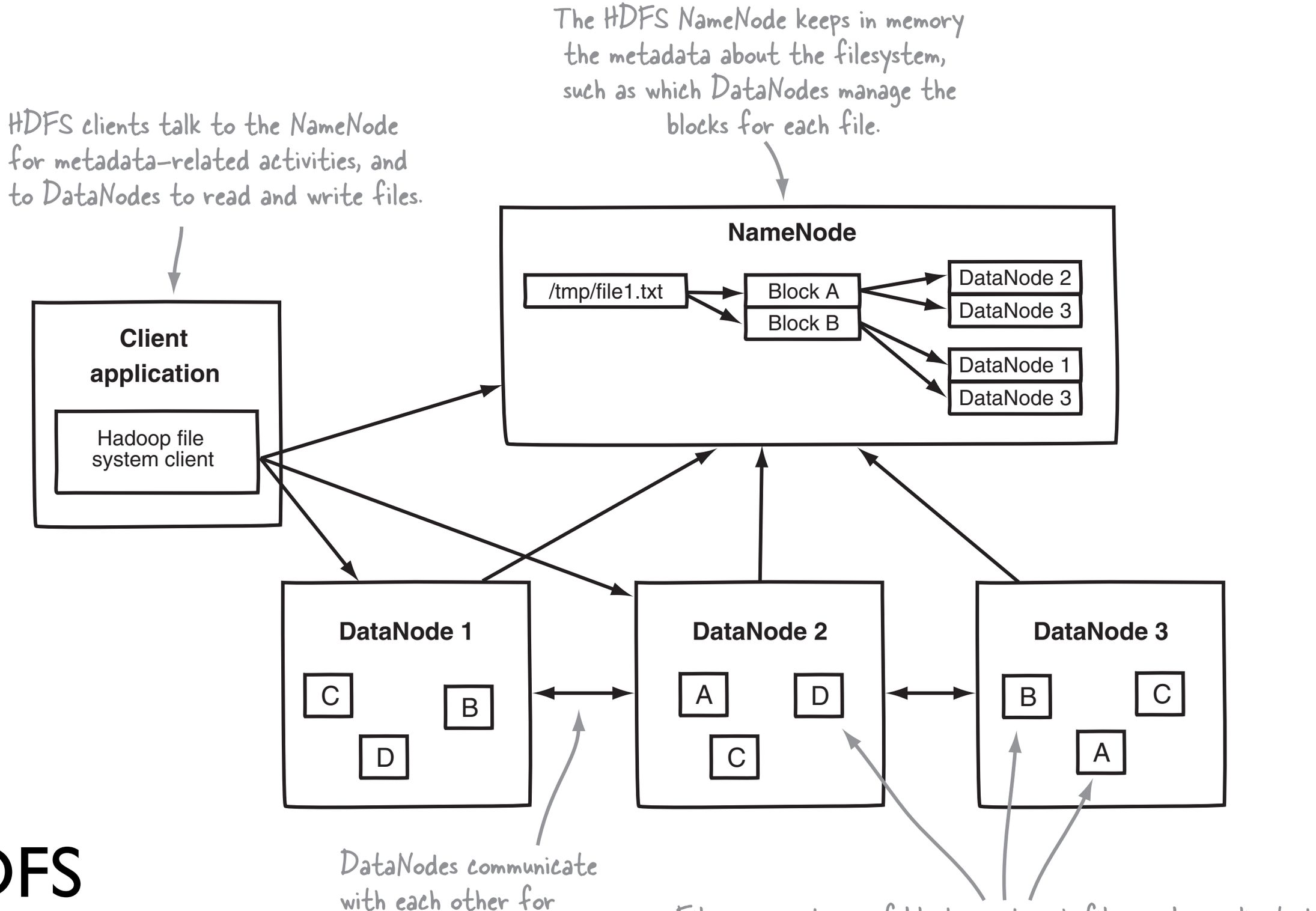
When a client application wants to read a file...

- it communicates with the **NameNode** to determine which blocks make up the file, and on which **DataNodes** the block reside
- it then communicates **directly** with the **DataNodes**

NameNode is the single point of failure of a Hadoop system

- **backup** periodically to remote **Network File System** (setup as part of Hadoop configuration)
- **use Secondary NameNode**
  - not the same as the NameNode
  - a slightly out of date copy

HDFS Architecture



[from Hadoop in Practice, Alex Holmes]

Quiz

1. A distributed file system behaves similar to which of the following?

A - RAID-0 File system

B - RAID-1 Filesystem

2. Which one of the following stores data?

A - Name node

B - Data node

C - Master node

D - None of these

3. In the secondary namenode the amount of memory needed is

A - Similar to that of primary node

B - Should be at least half of the primary node

C - Must be double of that of primary node

D - Depends only on the number of data nodes it is going to handle

4. With Hadoop, what is true for a HDFS block that is lost due to disk corruption or machine failure?

A - It is lost for ever

B - It can be replicated form its alternative locations.

C - The namenode allows new client request to keep trying to read it.

D - The Mapreduce job process runs ignoring the block and the data stored in it.

5. How does Hadoop process large volumes of data?

A - Hadoop uses a lot of machines in parallel. This optimizes data processing.

B - Hadoop was specifically designed to process large amount of data by taking advantage of MPP hardware.

C - Hadoop ships the code to the data instead of sending the data to the code.

D - Hadoop uses sophisticated caching techniques on name node to speed processing of data.

Quiz

1. A distributed file system behaves similar to which of the following?

A - RAID-0 File system

B - RAID-1 Filesystem

2. Which one of the following stores data?

A - Name node

B - Data node

C - Master node

D - None of these

3. In the secondary namenode the amount of memory needed is

A - Similar to that of primary node

B - Should be at least half of the primary node

C - Must be double of that of primary node

D - Depends only on the number of data nodes it is going to handle

4. With Hadoop, what is true for a HDFS block that is lost due to disk corruption or machine failure?

A - It is lost for ever

B - It can be replicated form its alternative locations.

C - The namenode allows new client request to keep trying to read it.

D - The Mapreduce job process runs ignoring the block and the data stored in it.

5. How does Hadoop process large volumes of data?

A - Hadoop uses a lot of machines in parallel. This optimizes data processing.

B - Hadoop was specifically designed to process large amount of data by taking advantage of MPP hardware.

C - Hadoop ships the code to the data instead of sending the data to the code.

D - Hadoop uses sophisticated caching techniques on name node to speed processing of data.

MapReduce

A batch based, distributed computing framework modelled on Google's paper on MapReduce [http://research.google.com/archive/mapreduce.html]

MapReduce decomposes work into small parallelised map and reduce tasks which are scheduled for remote execution on slave nodes

Terminology

- A *job* is a full programme
- A *task* is the execution of a single map or reduce task over a slice of data called a *split*
- A *Mapper* is a map task
- A *Reducer* is a reduce task

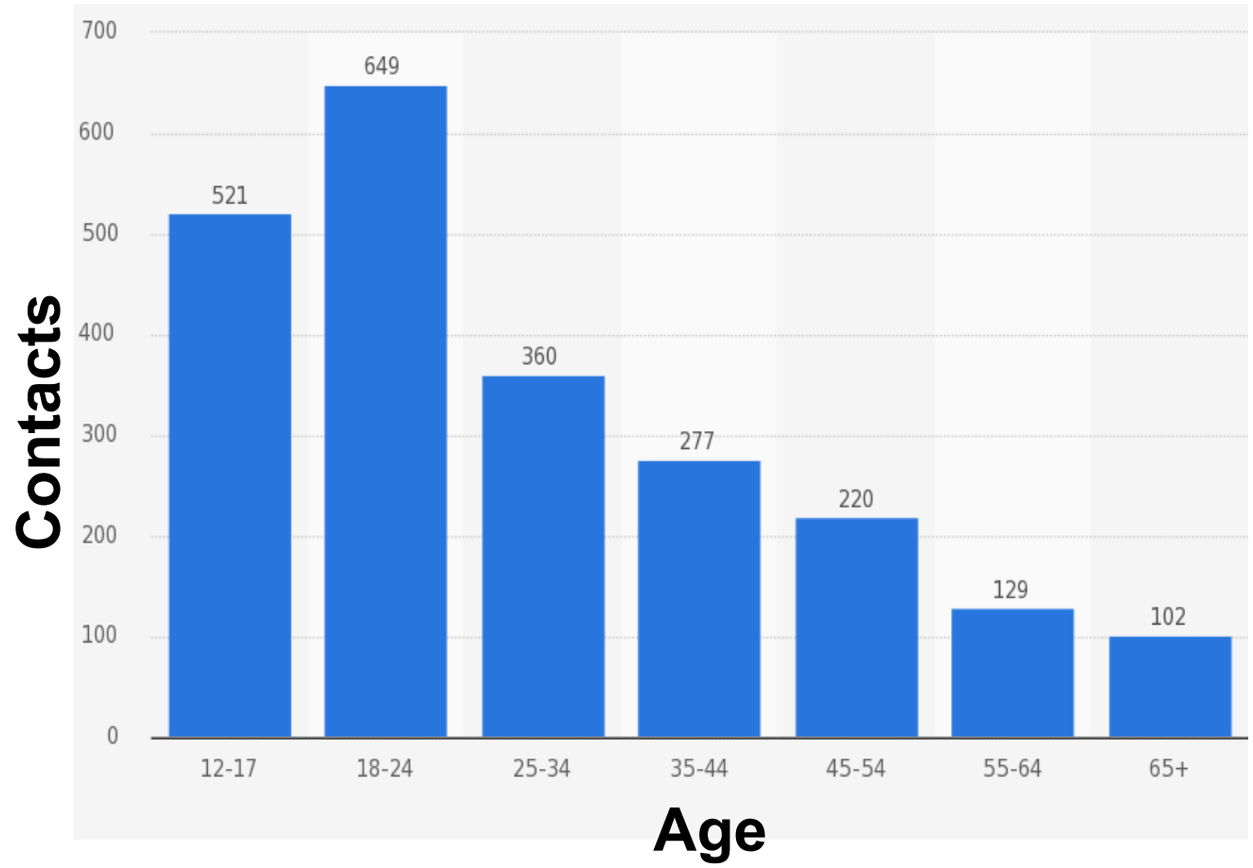
MapReduce works by manipulating key/value pairs in the general format

```
map(key1,value1)→ list(key2,value2)
reduce(key2,list(value2)) → (key3, value3)
```

2 MapReduce Examples

Imagine that for a database of 1.1 billion people, one would like to compute the average number of social contacts a person has according to age.

Aggregate stats over 1.1b records



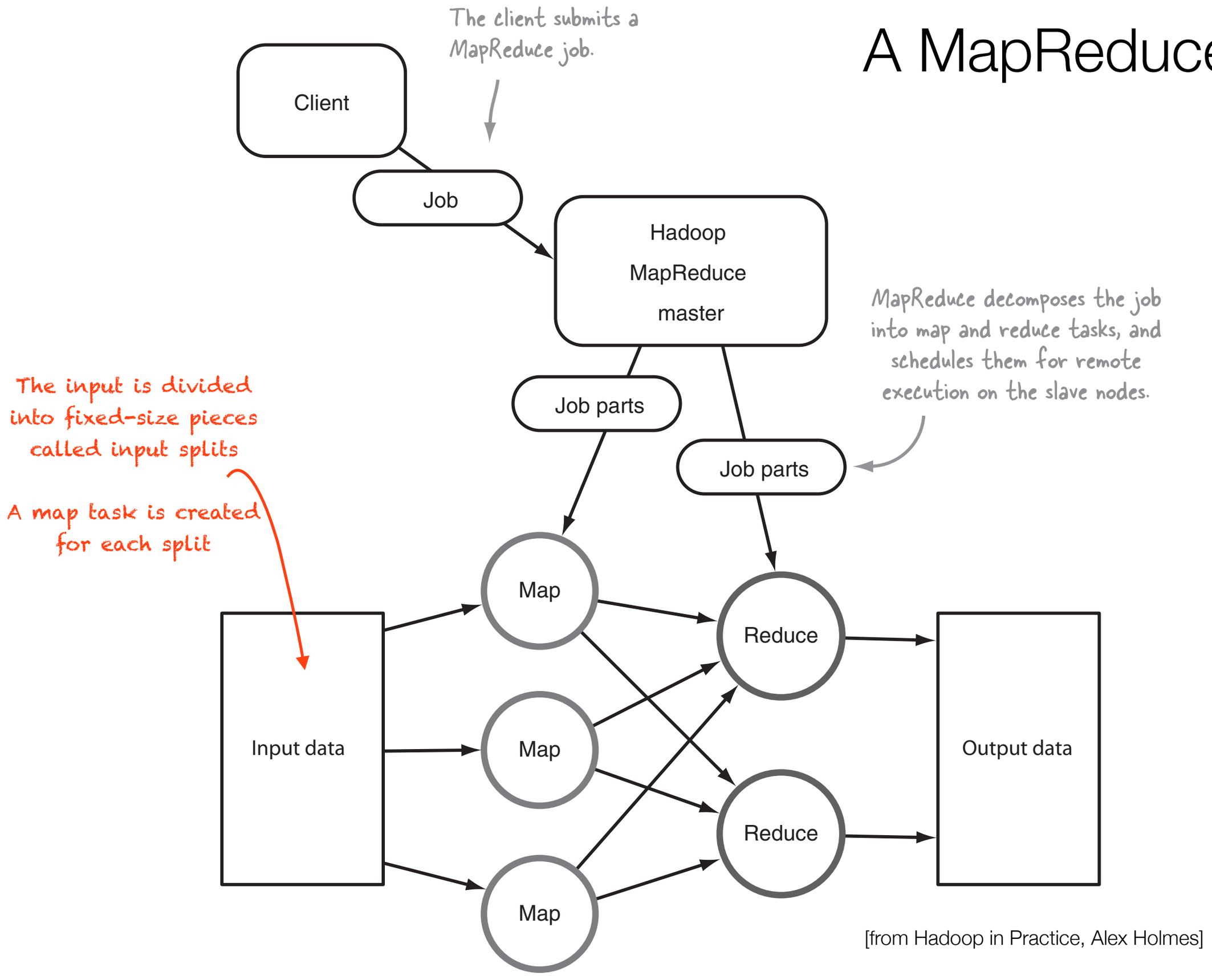
Word count: count occurrences of all words is corpus of docs,

e.g. 3 docs

```
"this one I think is called a yink"
"he likes to wink, he likes to drink"
"he likes to drink and drink and drink"
```

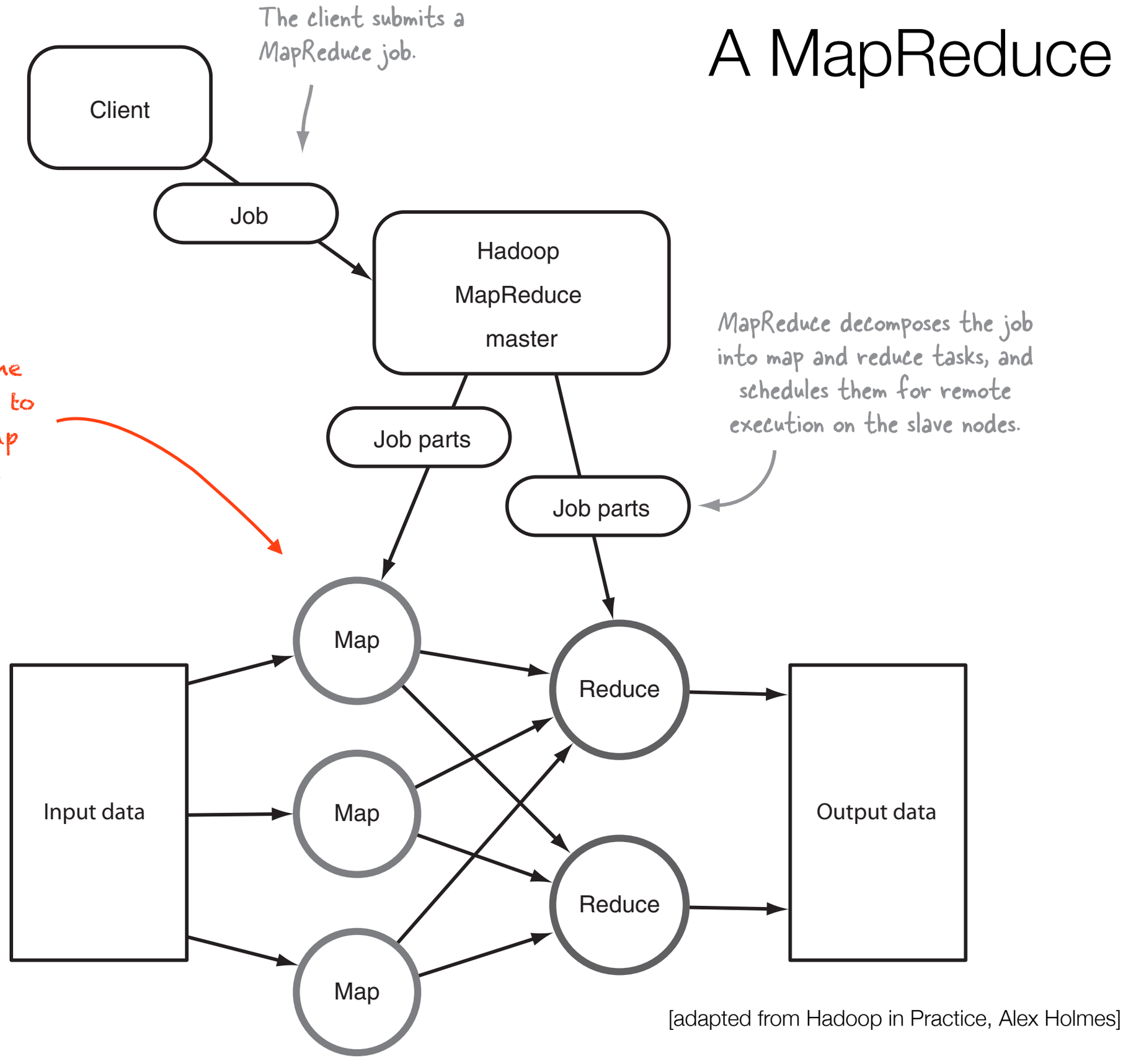
Jump ahead and look at solution: Slides 30-32

A MapReduce Job



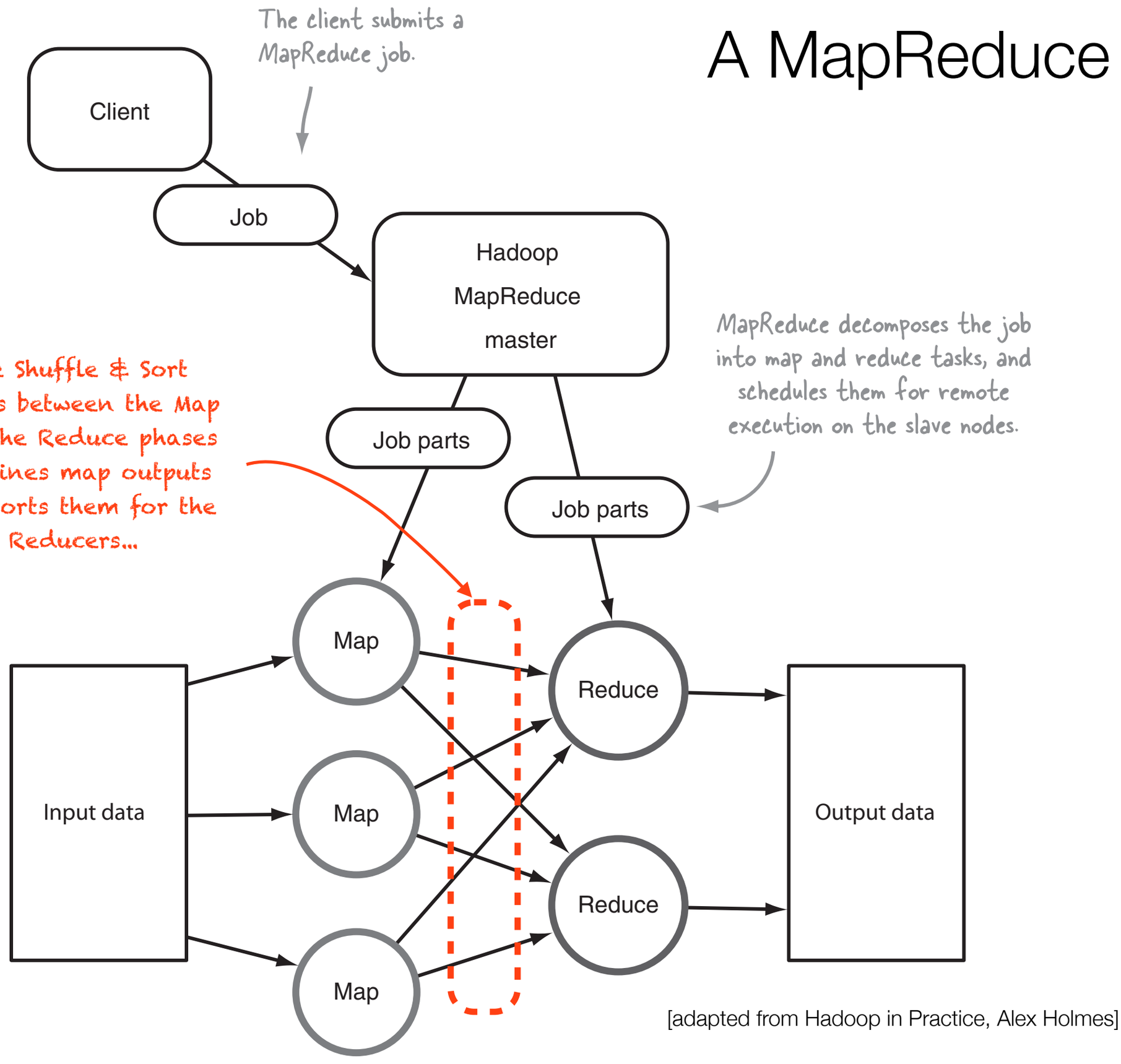
[from Hadoop in Practice, Alex Holmes]

A MapReduce Job



[adapted from Hadoop in Practice, Alex Holmes]

A MapReduce Job



[adapted from Hadoop in Practice, Alex Holmes]

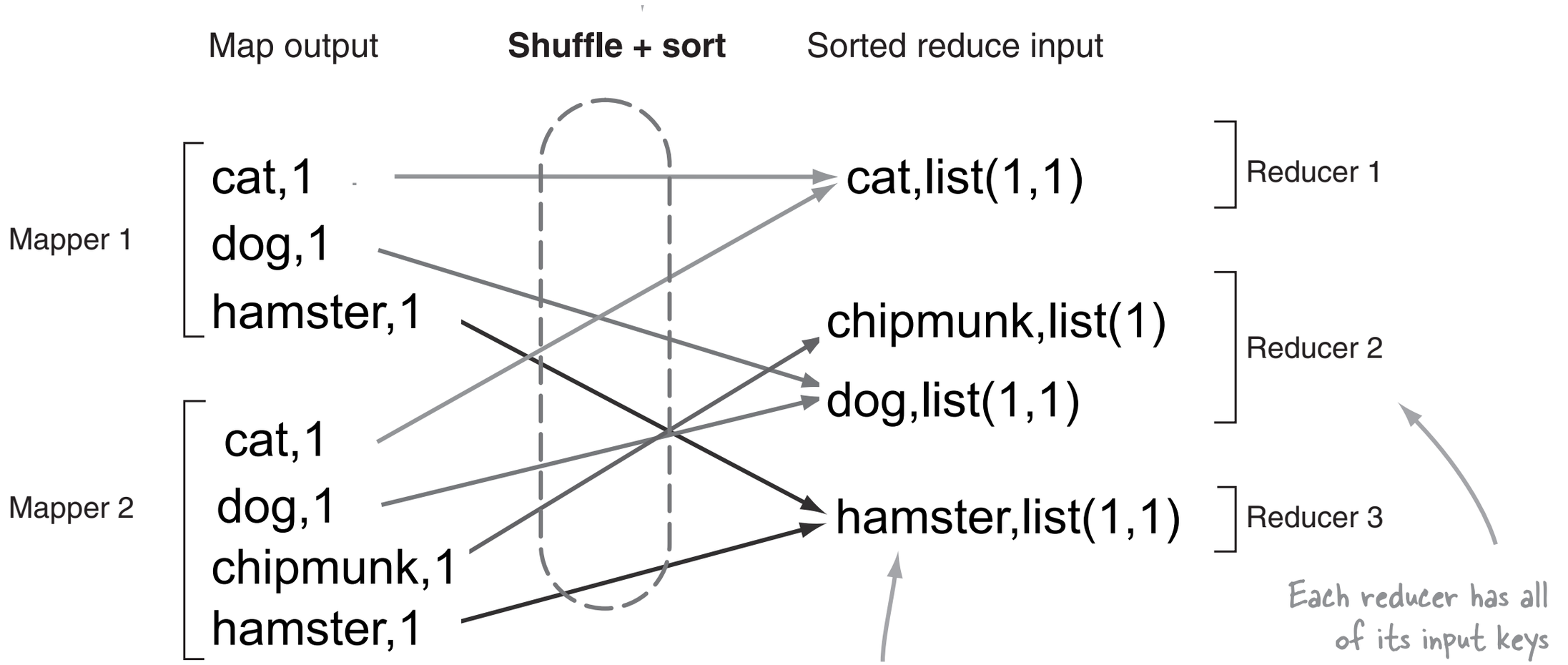
Shuffle & Sort

Shuffle & Sort phases (word count example)

Combines all the map output key/value pairs and sorts within key

Determines which subset goes to which Reducer

- doc1: cat dog hamster
- doc2: cat dog chipmunk hamster



Reduce

The Reducer(s)

A single Reducer handles all the map output for a unique map output key

A Reducer outputs zero to many key/value pairs

The output is written to HDFS files, to external DBs, or to any data sink...

```
reduce(temp_key,list(temp_values) → list(out_key, out_value)
```

MapReduce

JobTracker - (Master)

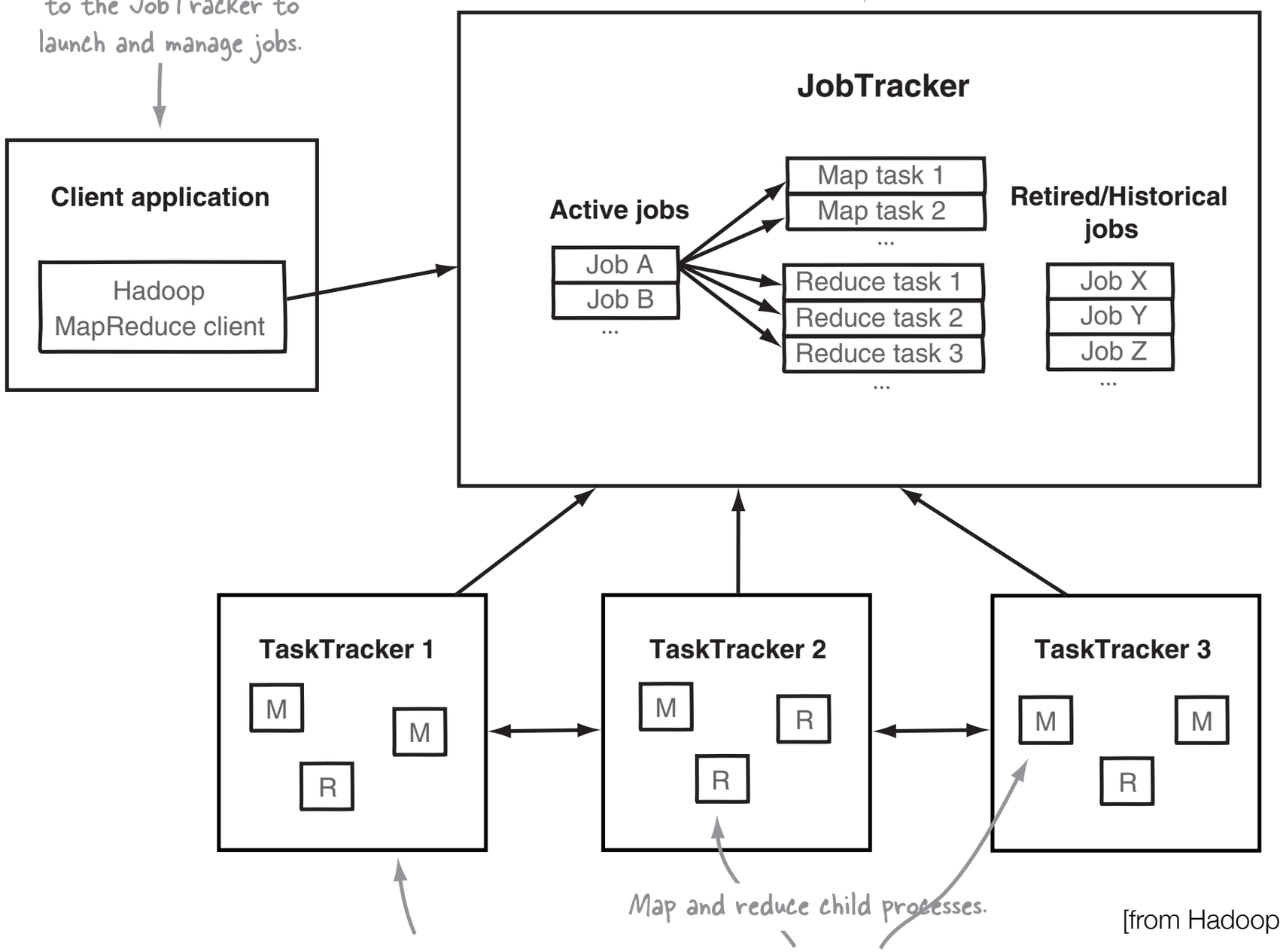
- **Controls MapReduce jobs**
- **Assigns Map & Reduce tasks** to the other nodes on the cluster
- Monitors the tasks as they are running
- Relaunches failed tasks on other nodes in the cluster

TaskTracker - (Slave)

- **A single TaskTracker per slave node**
- Manage the execution of the individual tasks on the node
- Can instantiate many Java VMs to handle tasks in parallel
- Communicates back to the JobTracker

The JobTracker coordinates activities across the slave TaskTracker processes. It accepts MapReduce job requests from clients and schedules map and reduce tasks on TaskTrackers to perform the work.

MapReduce clients talk to the JobTracker to launch and manage jobs.



Map and reduce child processes.

The TaskTracker is a daemon process that spawns child processes to perform the actual map or reduce work. Map tasks typically read their input from HDFS, and write their output to the local disk. Reduce tasks read the map outputs over the network and write their outputs back to HDFS.

[from Hadoop in Practice, Alex Holmes]

Mapper

The Mapper takes as input a key/value pair which represents a logical record from the input data source (e.g. a line in a file)

The Mapper may use or ignore the input key

- E.g. a standard pattern is to read a file one line at a time
- Key = byte offset into the file where the line starts
- Value = contents of the line in the file
- Typically the key can be considered irrelevant

It produces zero or more outputs key/value pairs for each input pair

e.g. a filtering function may only produce output if a certain condition is met

e.g. a counting function may produce multiple key/value pairs, one per element being counted

What does the Mapper produce for:

1. The Word Count example:
2. The Social Connections (friends/followers) example?

Reducer

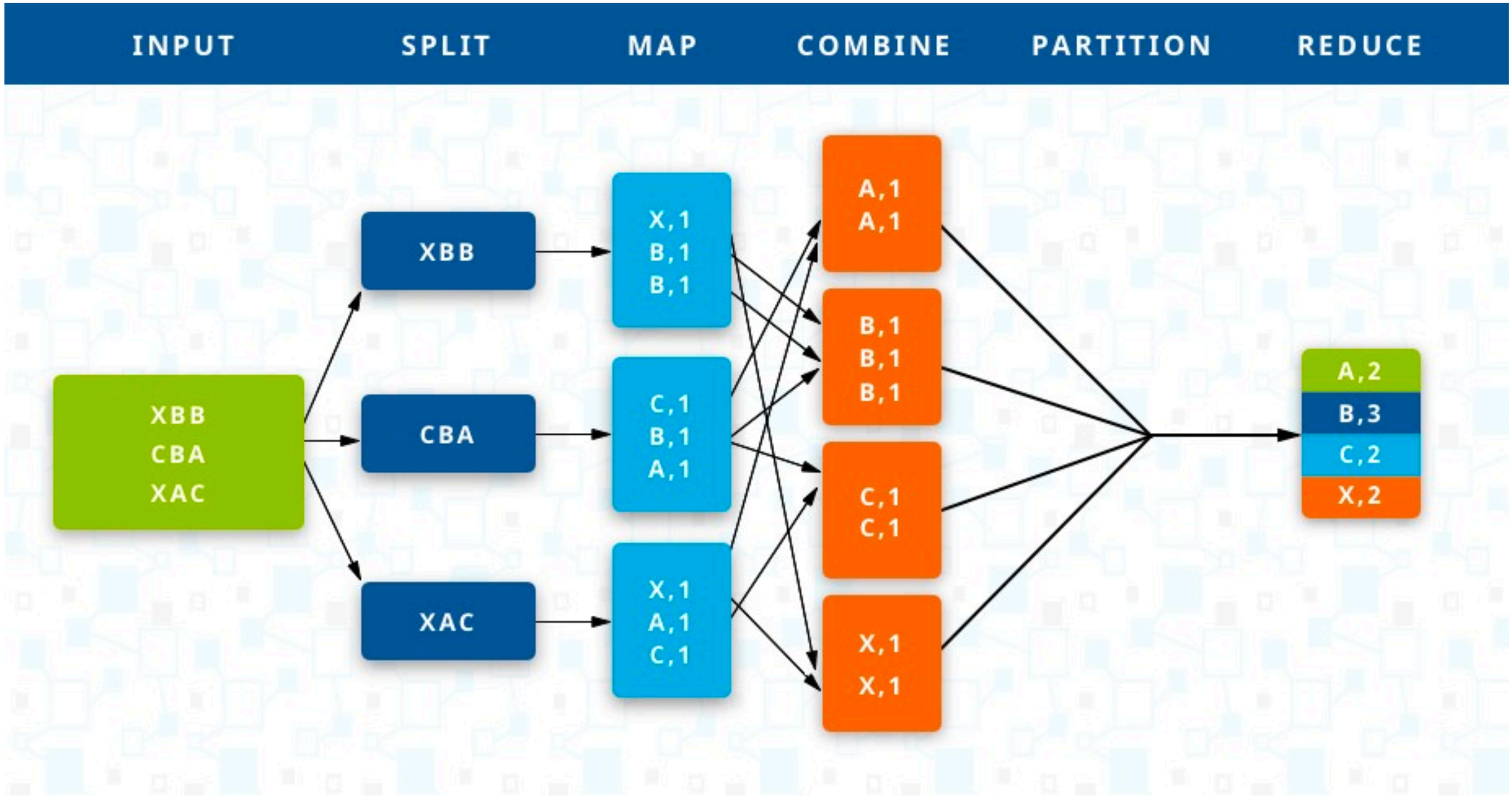
The Reducer(s)

- A single Reducer handles all the map output for a unique map output key
- These are combined by the Shuffle & Sort phase
- **All values for a unique map output key are guaranteed to go to the same Reducer**
- **A Reducer can output zero to many key/value pairs**
- The output is written to HDFS files, to external DBs, or to any data sink...

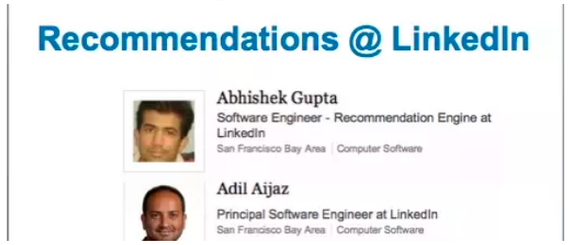
The number of Reducers can vary

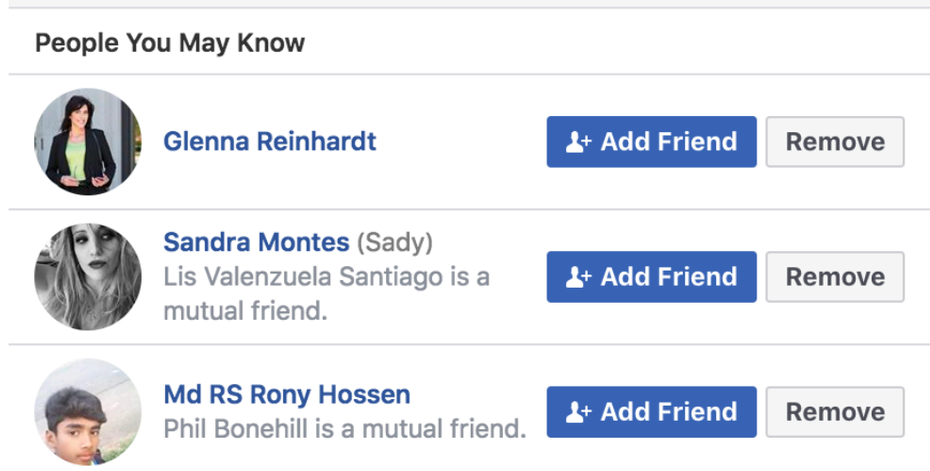
- Set as part of the job configuration
- In practice, the Reducer outputs a key/value pair for each reducer input key so the number of Reducers needed depends on the output.

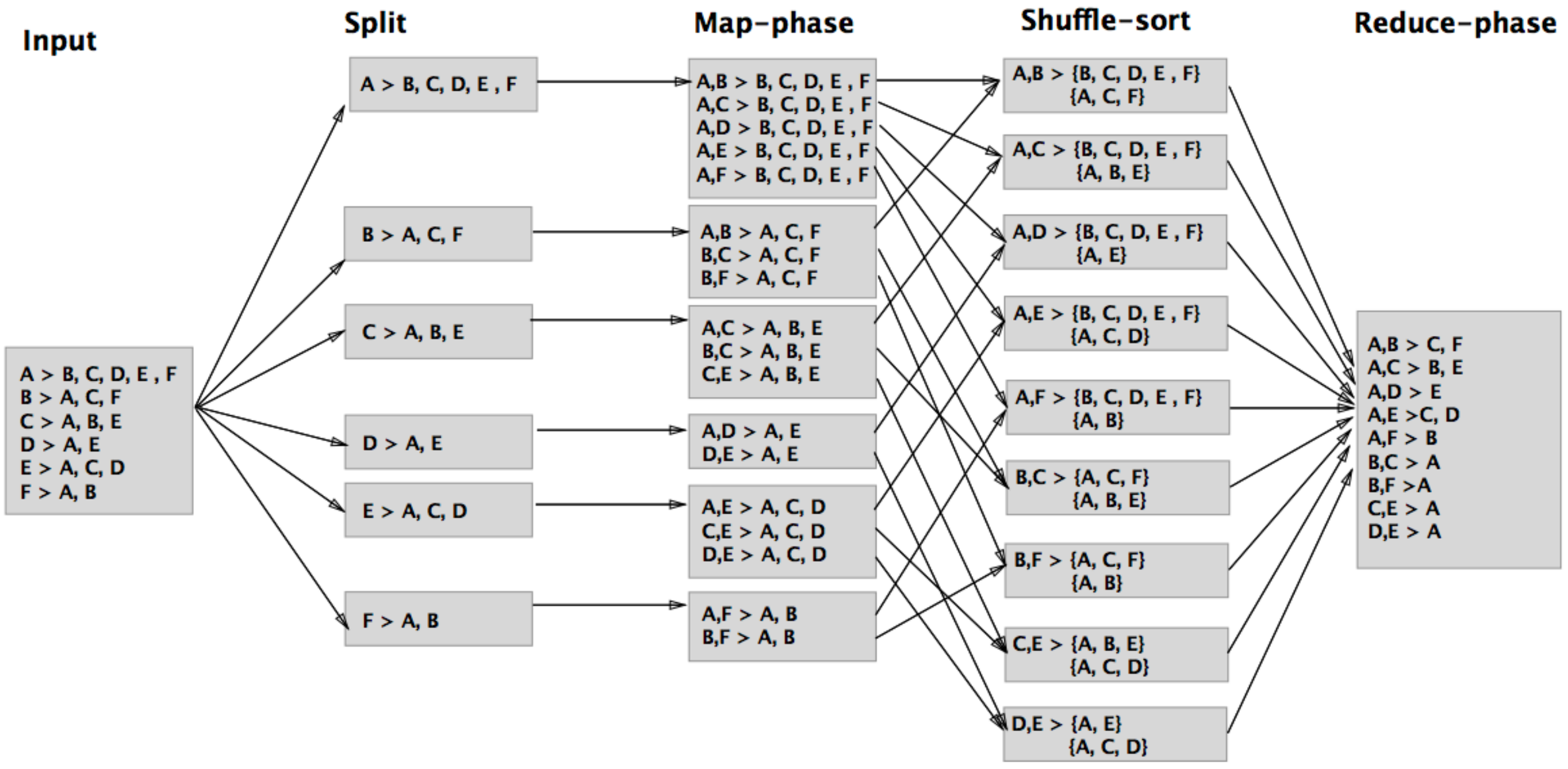
MapReduce process…



Friends in common example

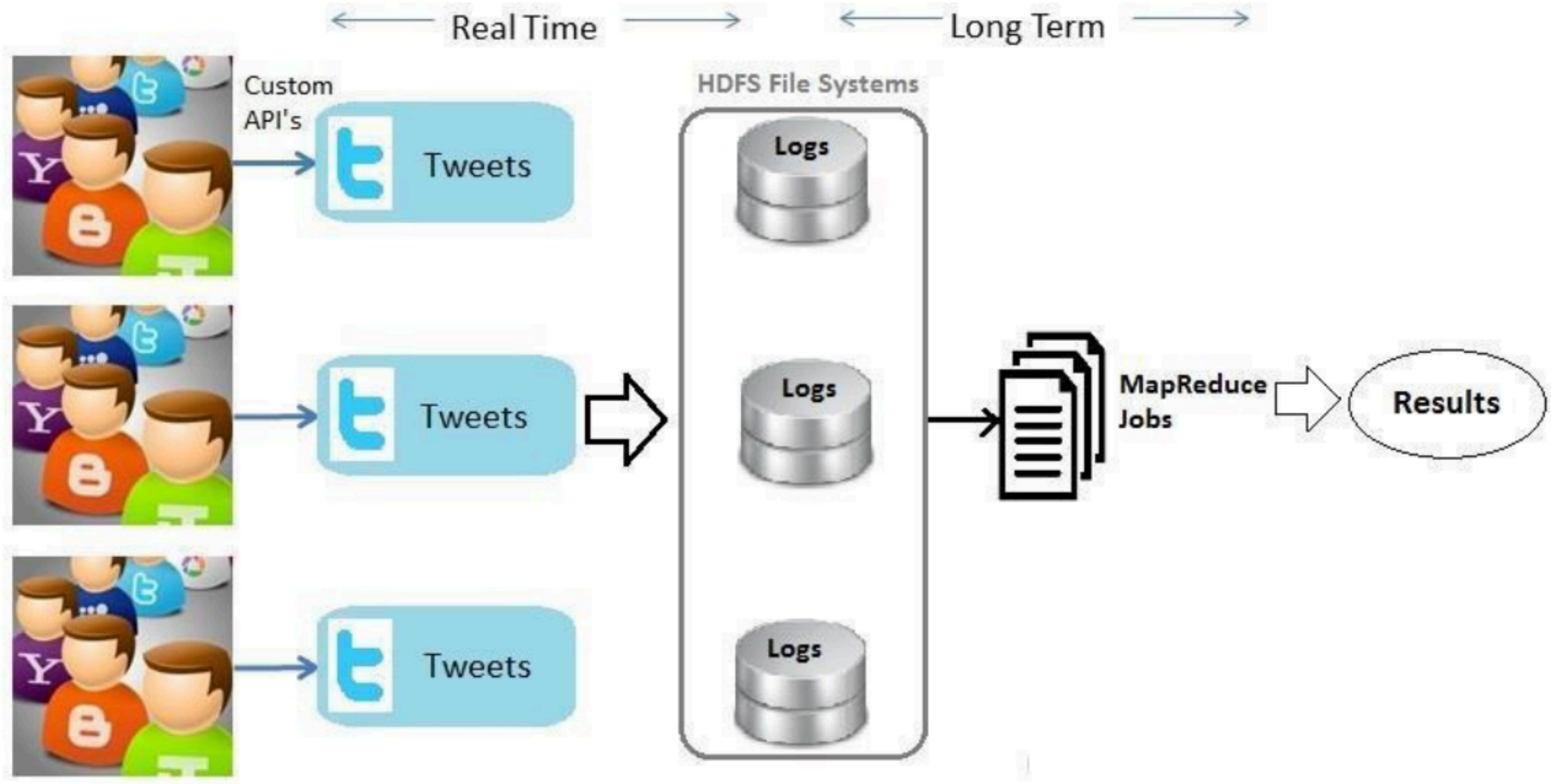






https://opensourceconnections.com/blog/2013/07/04/friend-recommendations-

Twitter monitor example



https://blog.cloudera.com/blog/2012/09/analyzing-twitter-data-with-hadoop/

Example MapReduce Job

Job: Count the occurrences of each word in a large amount of input data

```
map(String inputKey, String inputValue)
  foreach(word w in inputValue)
    output (w,1)
```

- Input to the Mapper:

```
(124, “this one I think is called a yink”)
(158, “he likes to wink, he likes to drink”)
(195, “he likes to drink and drink and drink”)
```

- Output from the Mapper:

```
(this, 1) (one, 1) (I, 1) (think, 1) (is, 1) (called,1)(a, 1)
(yink,1) (he, 1) (likes,1) (to,1) (wink,1) (he,1) (likes,1)
(to,1) (drink,1) (he,1) (likes,1) (to,1) (drink 1) (and,1)
(drink,1) (and,1) (drink,1)
```

Example Job

Intermediate data sent to the Reducer

```
(a, [1])
(and,[1,1])
(called, [1])
(drink, [1,1,1,1])
(he, [1,1,1])
(I, [1])
(is, [1])
(likes, [1,1])
(one, [1])
(think, [1])
(this, [1])
(to, [1,1,1])
(wink, [1])
(yink, [1] )
```

Example Job

```
reduce(String outputKey, List<int> intermediateValues)
  set count = 0
  foreach(value v in intermediateValues)
    count=count+1
  output(outputKey, count)
```

- Output from the Reducer:

```
(a,1)                (likes,2)
(and,2)              (one,1)
(called,1)           (think,1)
(drink,4)            (this,1)
(he,3)               (to,3)
(I,1)                (wink,1)
(is,1)               (yink,1)
```

Jump back to slide 22

Summary

Introduction to Hadoop
https://hadoop.apache.org

Overview of HDFS and MapReduce

Suitable problems

How the Mapper and Reducer work

Shuffle & Sort

Pseudocode for a simple Hadoop programme