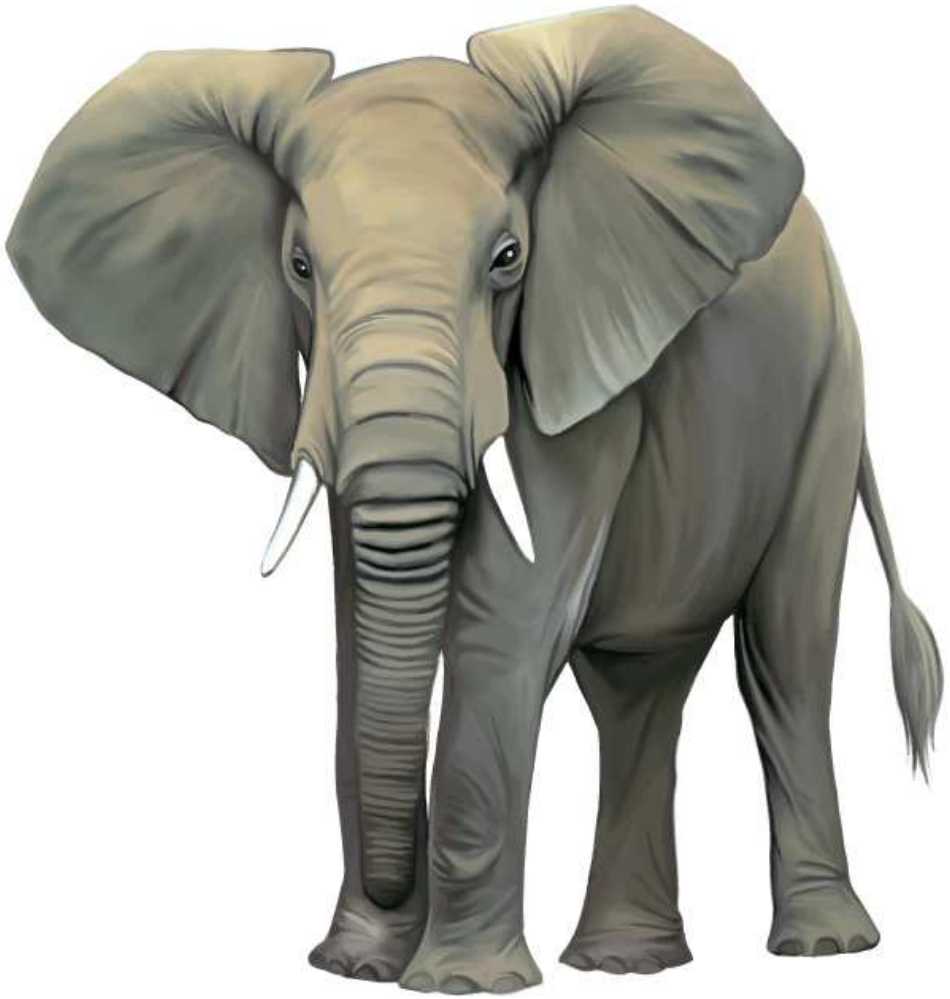
Hadoop



Sudsanguan Ngamsuriyaroj
Ekasit Kijsipongse
Putt Sakdhnagool

Semester 1/2019

Topics

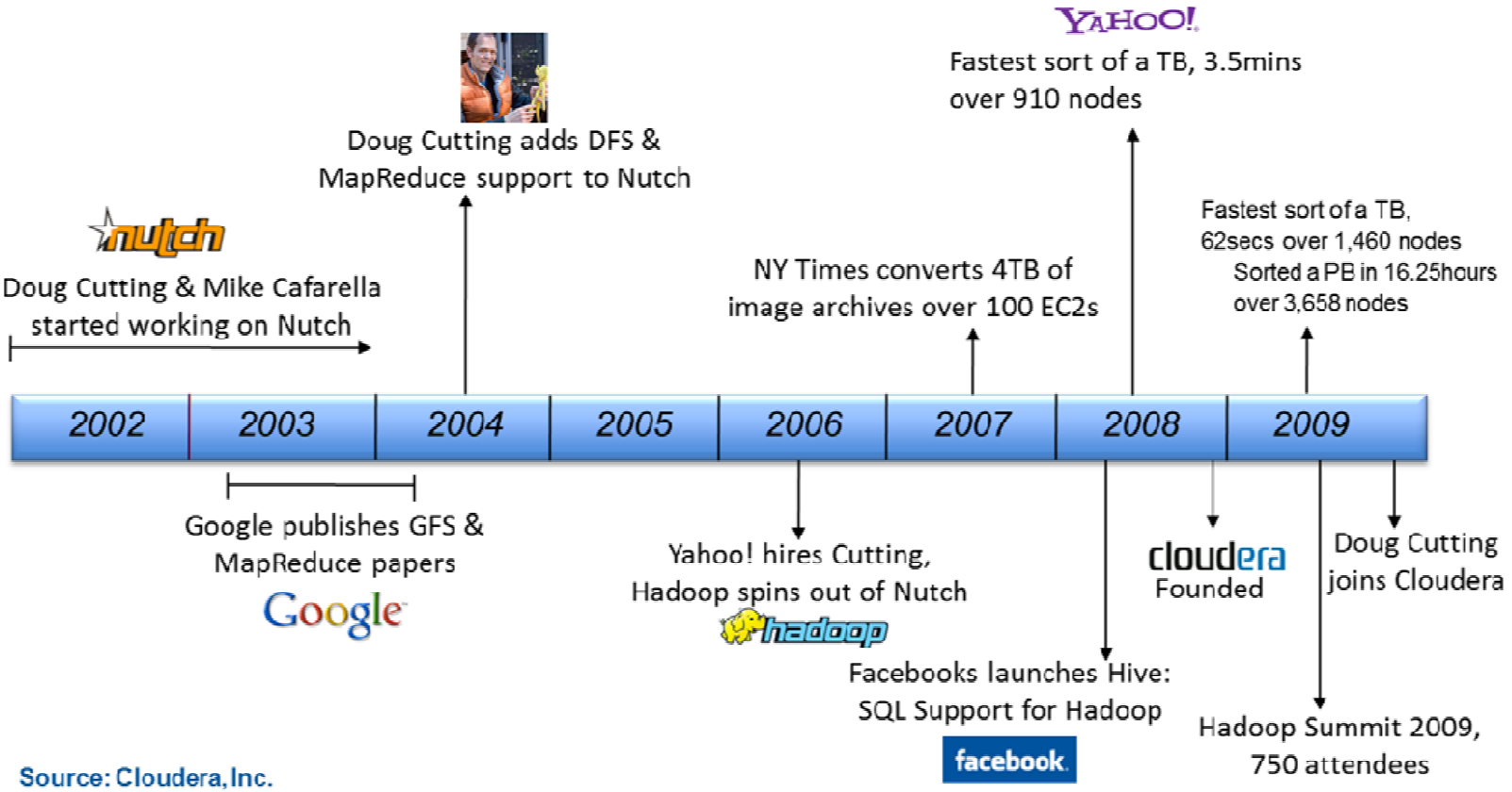
- What is Hadoop
- Hadoop Distributed File System
- Hadoop MapReduce
- Programming in MapReduce
- Hadoop Ecosystem

What is 

- How to process Big data with reasonable cost and time?
- Hadoop is an open-source software that supports Big data applications, licensed under the Apache v2 license
- Design Principles
  - Facilitate the storage and processing of large and/or rapidly growing data sets, both structured and non-structured data
  - Simple programming models
  - Optimized for large and very large data
  - Highly scalable
  - Use commodity (cheap!) hardware
  - Fault-tolerance
  - Move computation rather than data

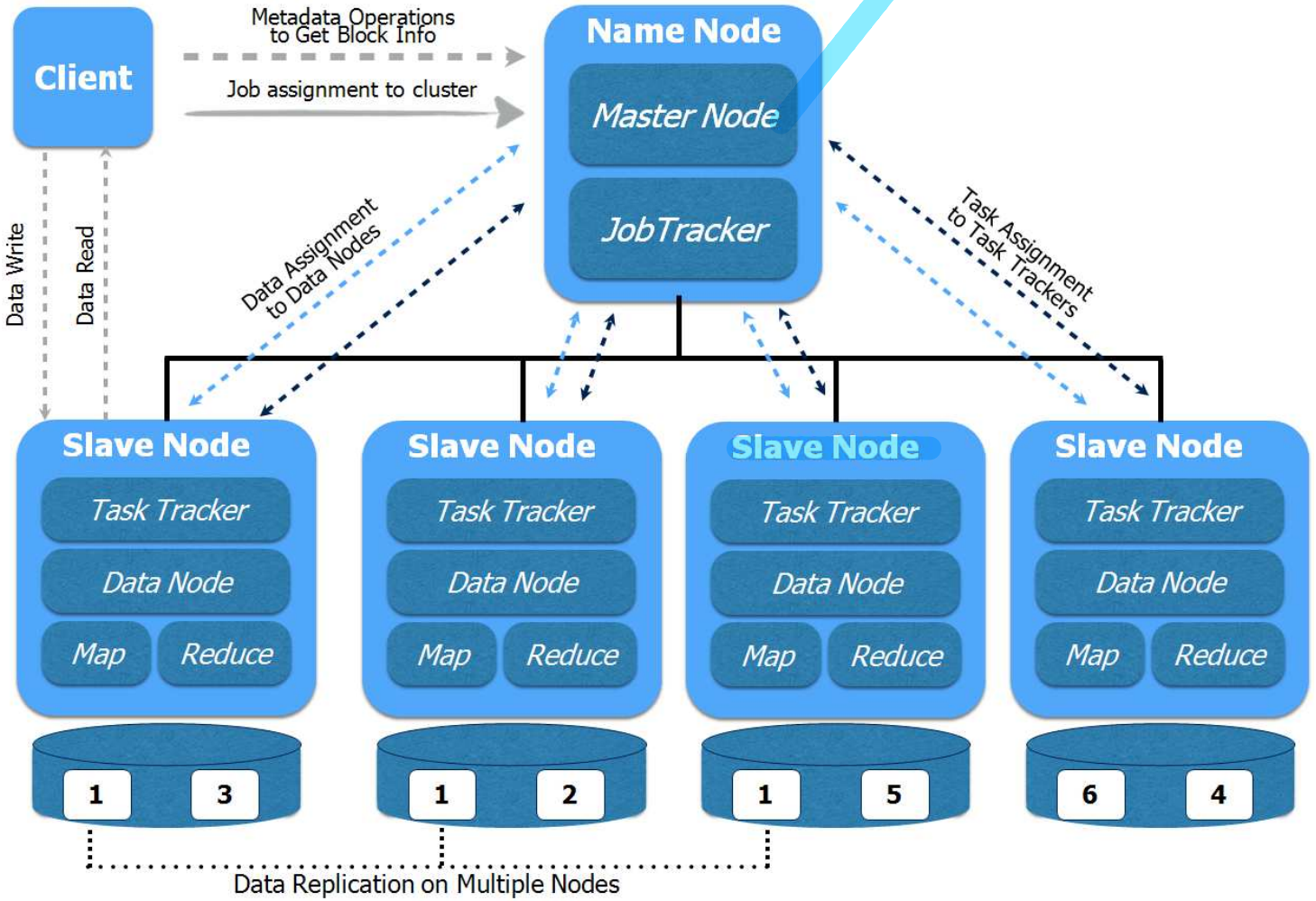
Hadoop History

- Starting by Cutting at Yahoo based on Google's publication in 2003 and 2004
- Developers works at several organizations, e.g facebook, Cloudera, Yahoo

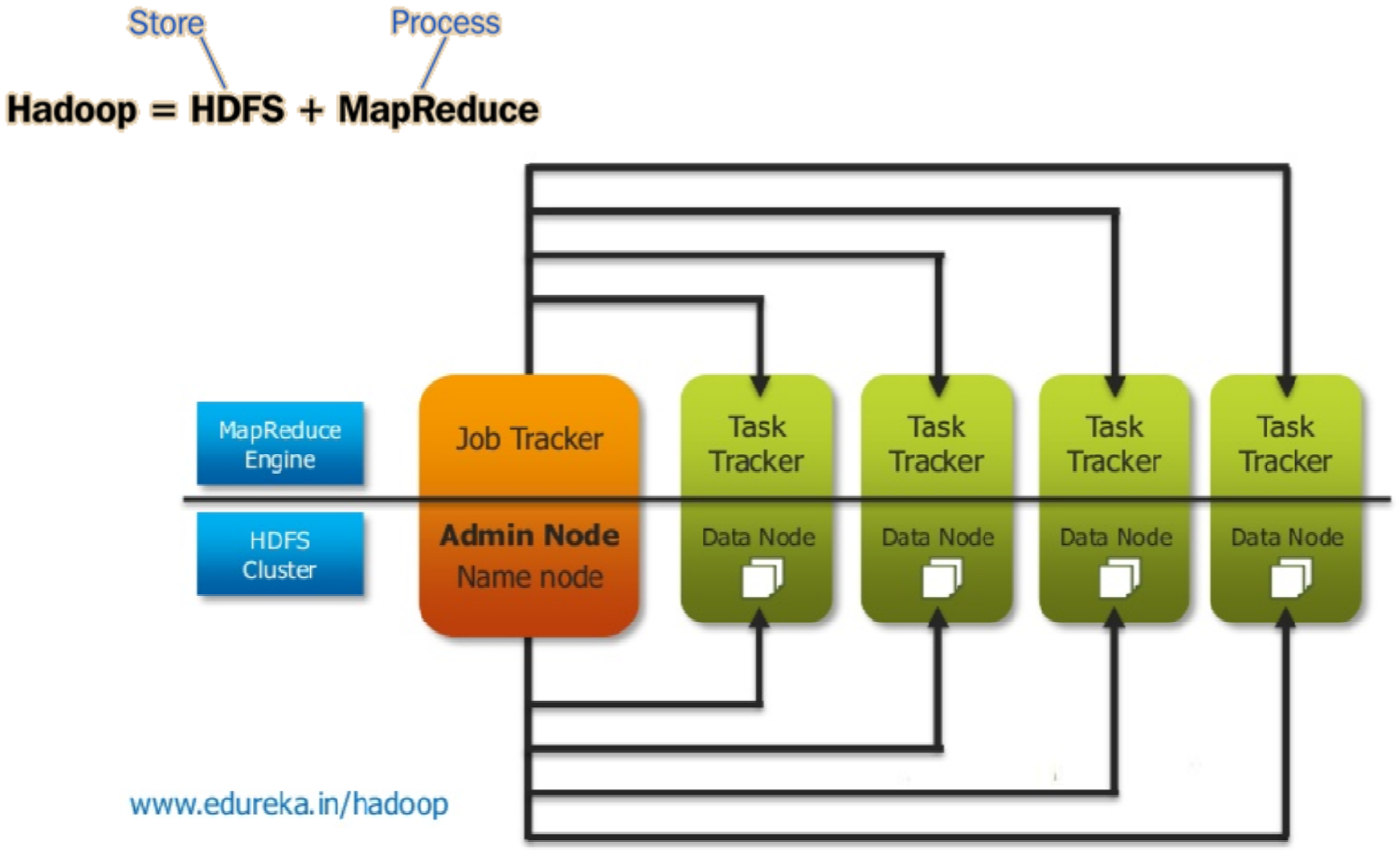


Source: Cloudera, Inc.

Batch Processing in Hadoop



Hadoop Core Architecture

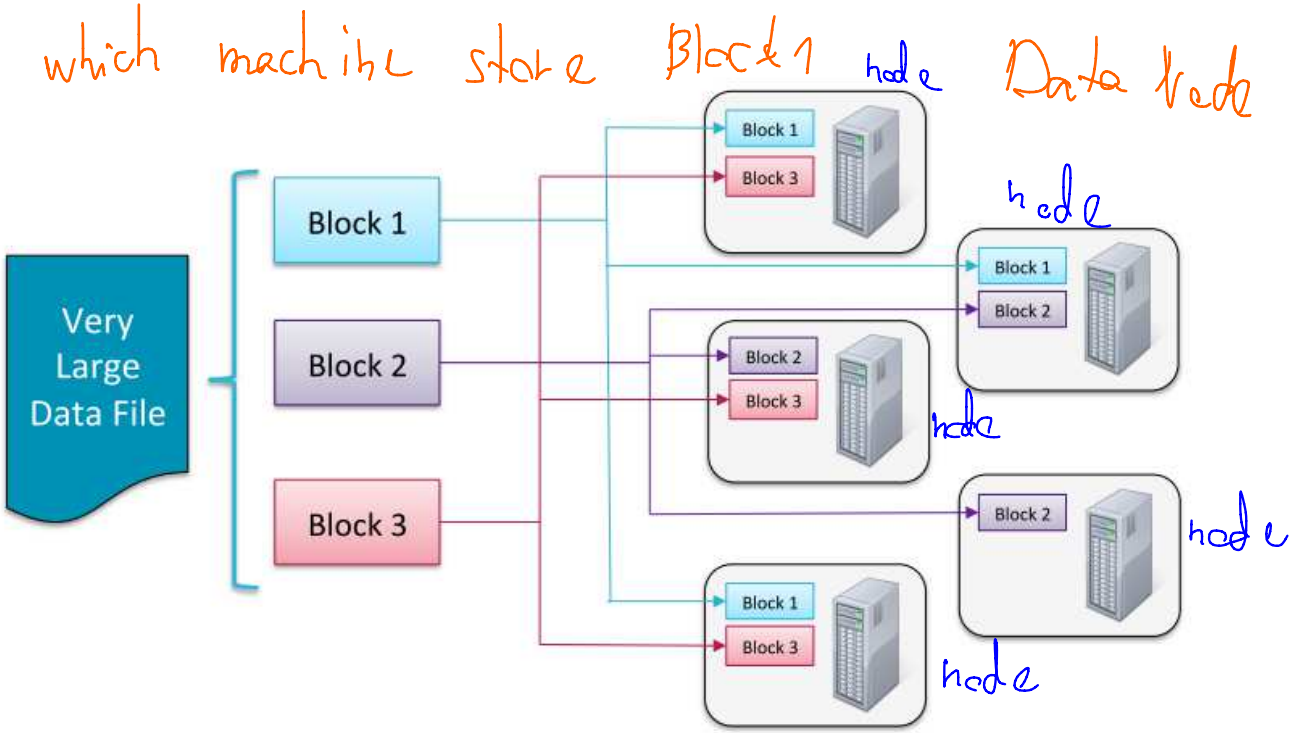


Hadoop Distributed File System (HDFS)

- Very Large Distributed File System (not Databases)
  - 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 Large Streaming Reads of Files
  - Provides very high aggregate bandwidth
- Files are broken up into blocks
  - Typically 64MB block size
- Each block replicated on multiple DataNodes
- Not support random write! (only created and appended)

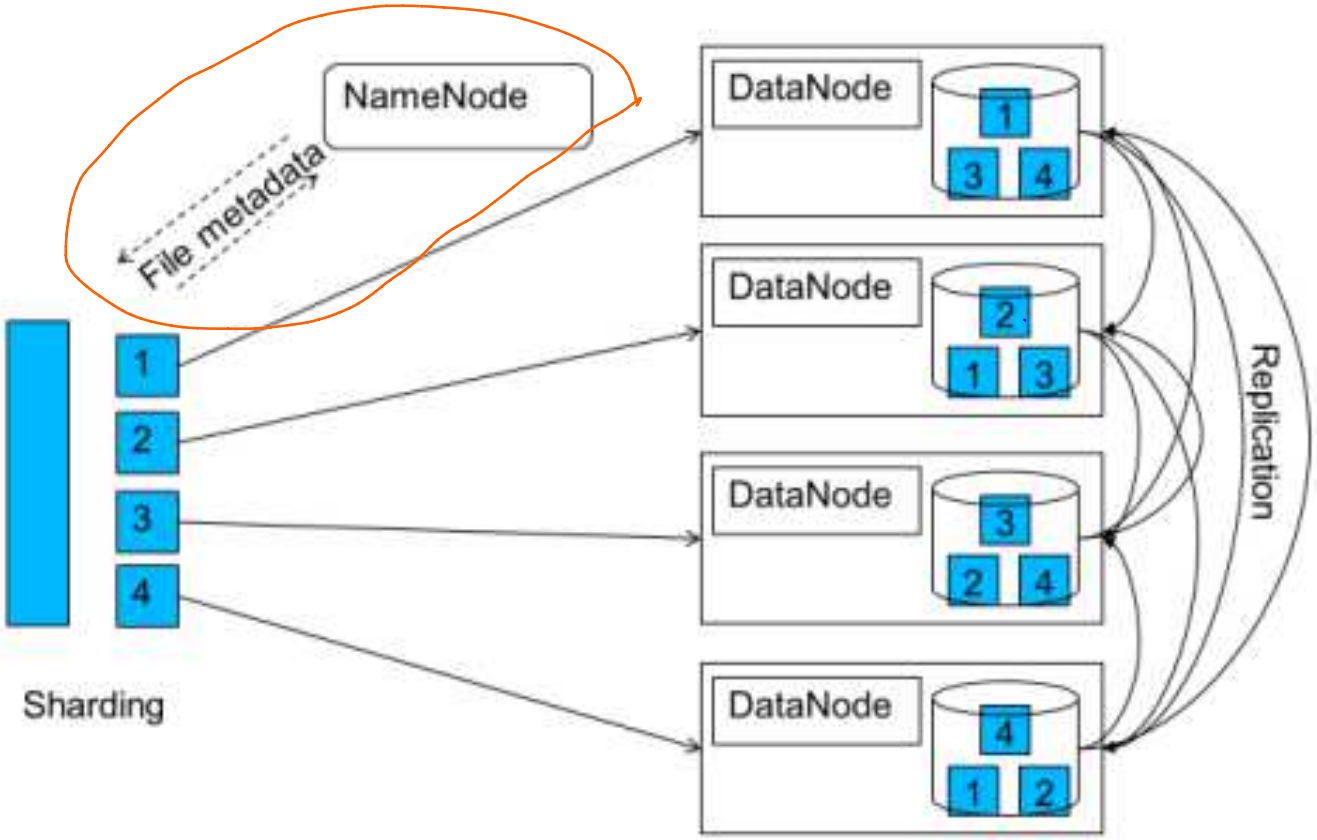
How Files are Stored

- Files are split into blocks and distributed to datanodes
- Each block is replicated to multiple nodes (default to 3)



HDFS Architecture

- NameNode + DataNode



Namenode

- Maintains the HDFS namespace, filesystem tree and metadata
- Maintains the mapping from each file to the list of blockIDs where the file is
- Metadata mapping is maintained in memory as well as persisted on disk
- Maintains in memory the locations of each block. (Block to datanode mapping)
- Memory requirement: ~150 bytes/file
- Issues instructions to datanode to create/replicate/delete blocks

•การทำแผนที่ข้อมูลเมตาจะถูกเก็บไว้ในหน่วยความจำเช่นกัน คงอยู่บนดิสก์
•รักษาหน่วยความจำตำแหน่งของแต่ละบล็อก (การแมปบล็อกกับดาต้าโหนด)
•ข้อกำหนดหน่วยความจำ: ~ 150 ไบต์ / ไฟล์
•ปัญหาคำแนะนำในการ datanode สร้าง / ทำซ้ำ / ลบบล็อก

Datanodes

Data Not store metadata.

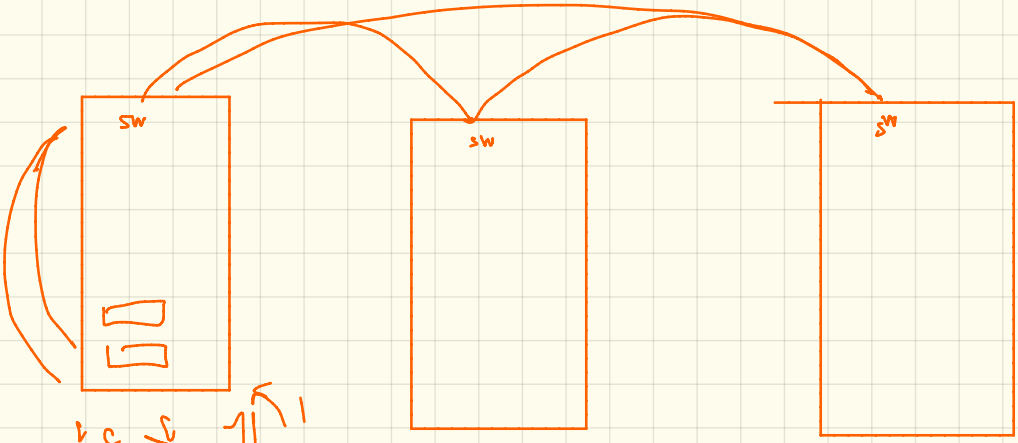
- Serve as storage for data blocks
- No metadata
- Report all blocks to namenode at startup (BlockReport)
- Sends periodic "heartbeat" to Namenode
- Serves read, write requests, performs block creation, deletion, and replication upon instruction from Namenode
- User data never flows through the NameNode

ใช้เป็นที่เก็บข้อมูลสำหรับบล็อกข้อมูล
•ไม่มีข้อมูลเมตา
•รายงานบล็อกทั้งหมดเพื่อ *namenode* เมื่อเริ่มต้น *(BlockReport)*
•ส่ง *"heartbeat"* เป็นระยะถึง *Namenode*
•ทำหน้าที่อ่านเขียนร้องขอทำการสร้างบล็อก การลบและการจำลองแบบตามคำสั่งจาก *Namenode*
•ข้อมูลผู้ใช้จะไม่ไหลผ่าน *NameNode*

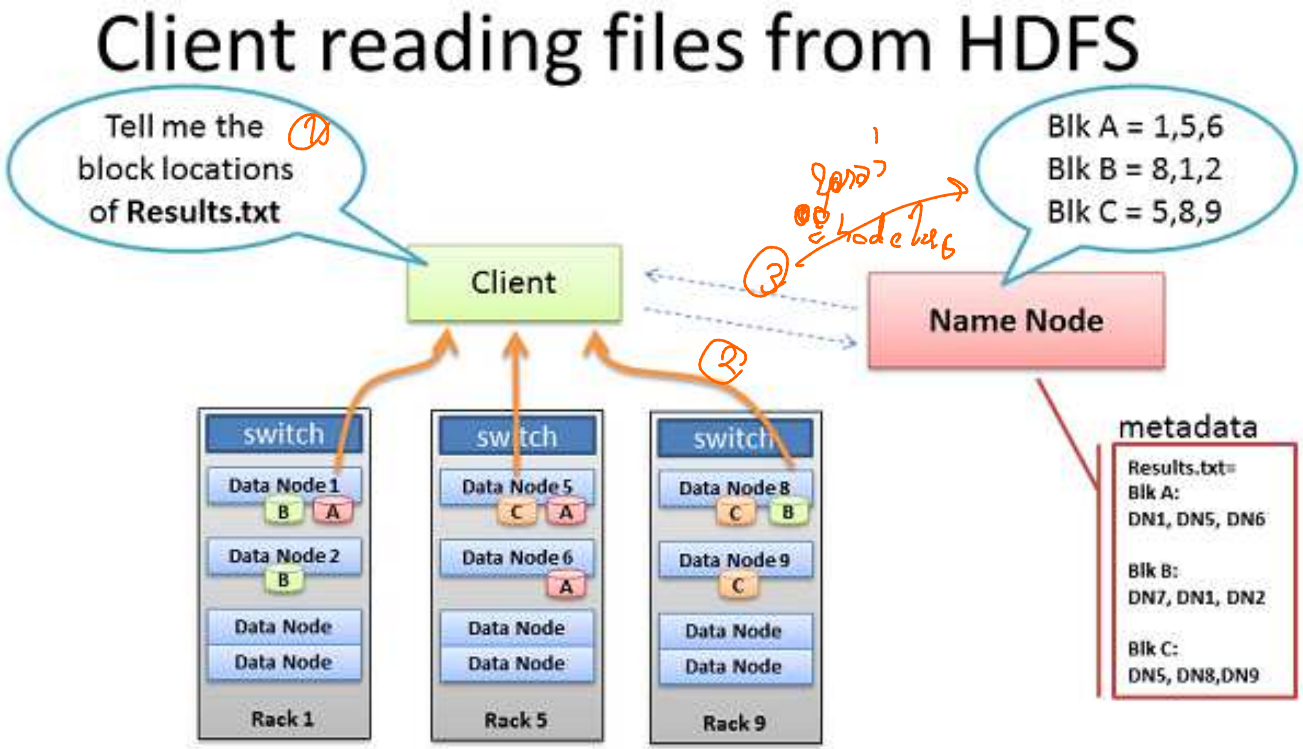
Replication and Rack-awareness

- Replication in Hadoop is at the block level
- Each block of data will be replicated to multiple machines to prevent the failure of one machine from losing all copies of data
- Unfortunate if all copies of data happened to be located on machines in the same rack, and that rack experiences a failure?
- Replication is "Rack-aware"
- Reading and writing on HDFS also makes use of rack-awareness

•โชคร้ายถ้าสำเนาของข้อมูลทั้งหมดเกิดขึ้น
ตั้งอยู่บนเครื่องจักรในชั้นวางเดียวกันและชั้นวางนั้น
ประสบกับความล้มเหลว?
•การจำลองแบบคือ *"Rack-aware"*
•การอ่านและการเขียนบน *HDFS* ยังใช้ประโยชน์จากชั้นวาง
ความตระหนัก


sw
sw
sw

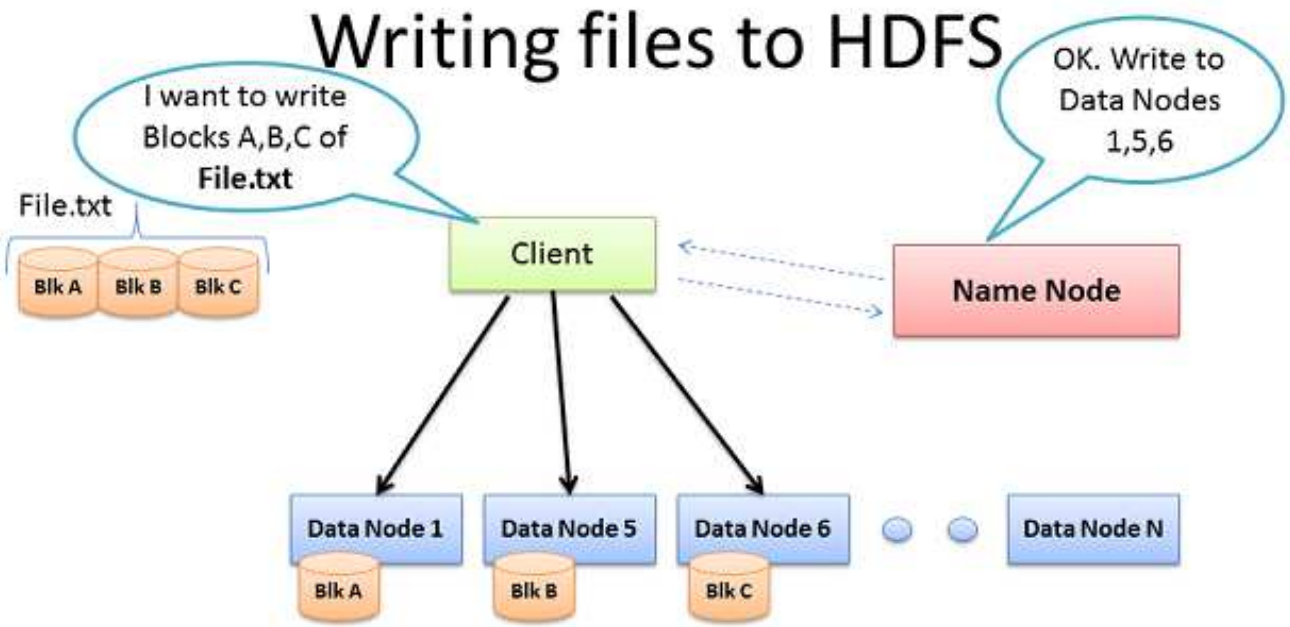
Reading Files from HDFS



- Client receives Data Node list for each block
- Client picks first Data Node for each block
- Client reads blocks sequentially

BRAD HEDLUND .com

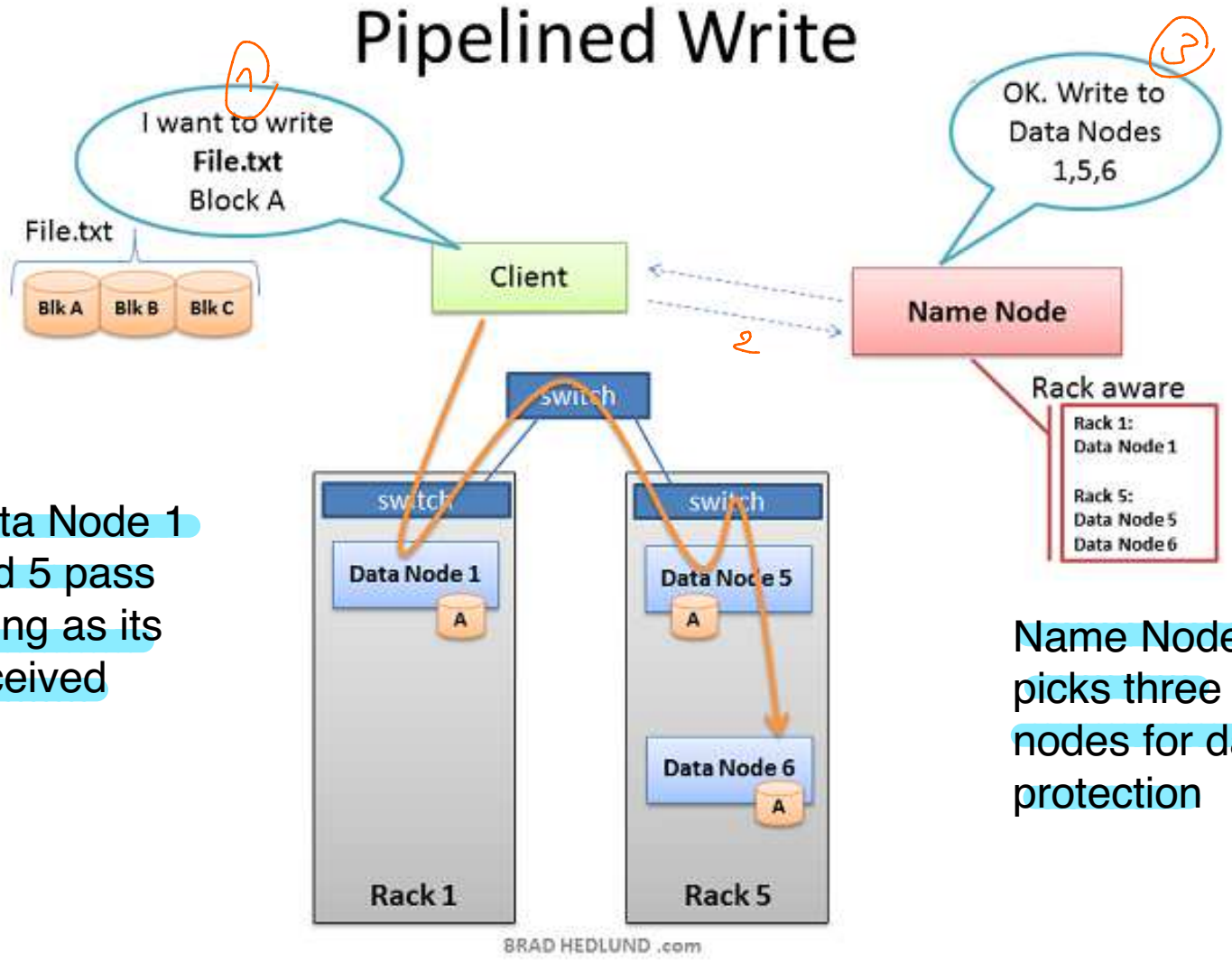
Writing Files to HDFS



- Client consults Name Node
- Client writes block directly to one Data Node
- Data Nodes replicates block
- Cycle repeats for next block

BRAD HEDLUND .com

Pipelined Write



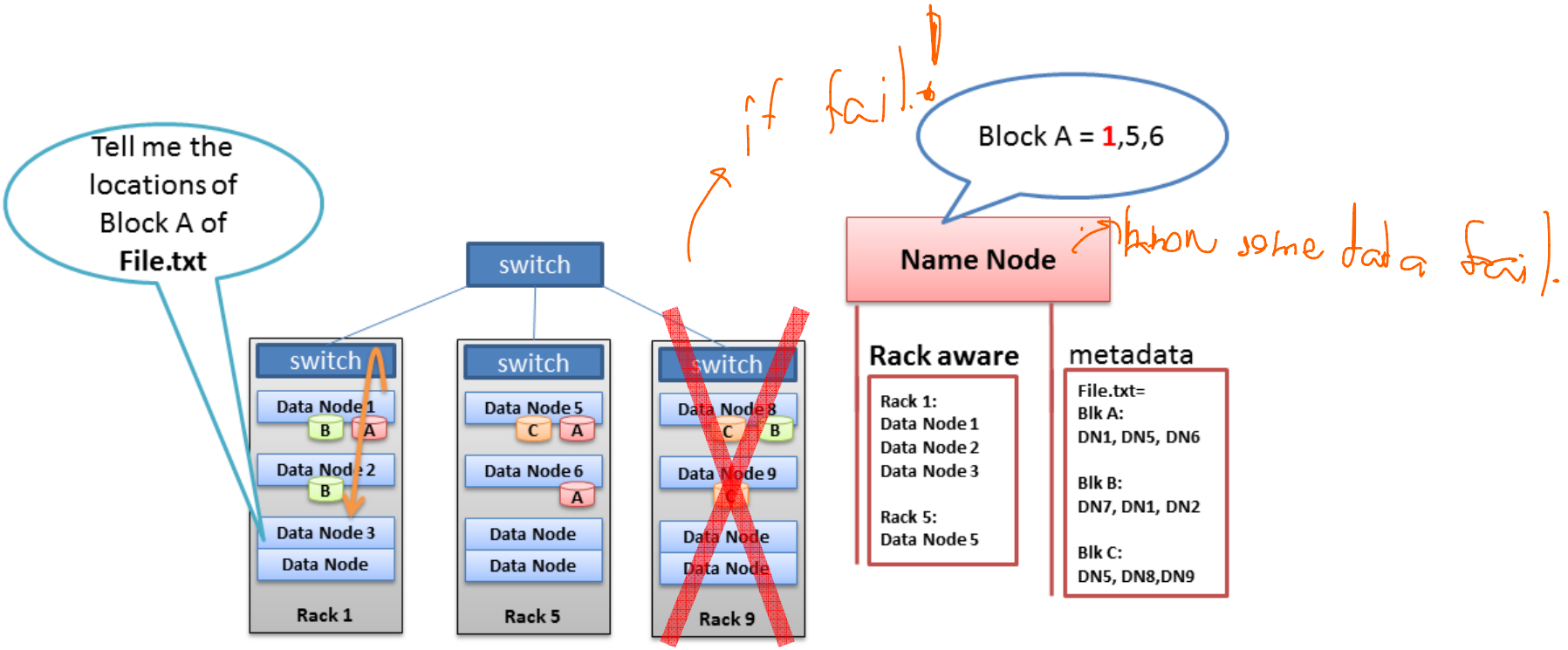
Data Node 1 and 5 pass along as its received

Name Node picks three nodes for data protection

Fault-Tolerance

ไม่สูญเสียข้อมูลแม้ว่าแร็คทั้งหมดจะล้มเหลว

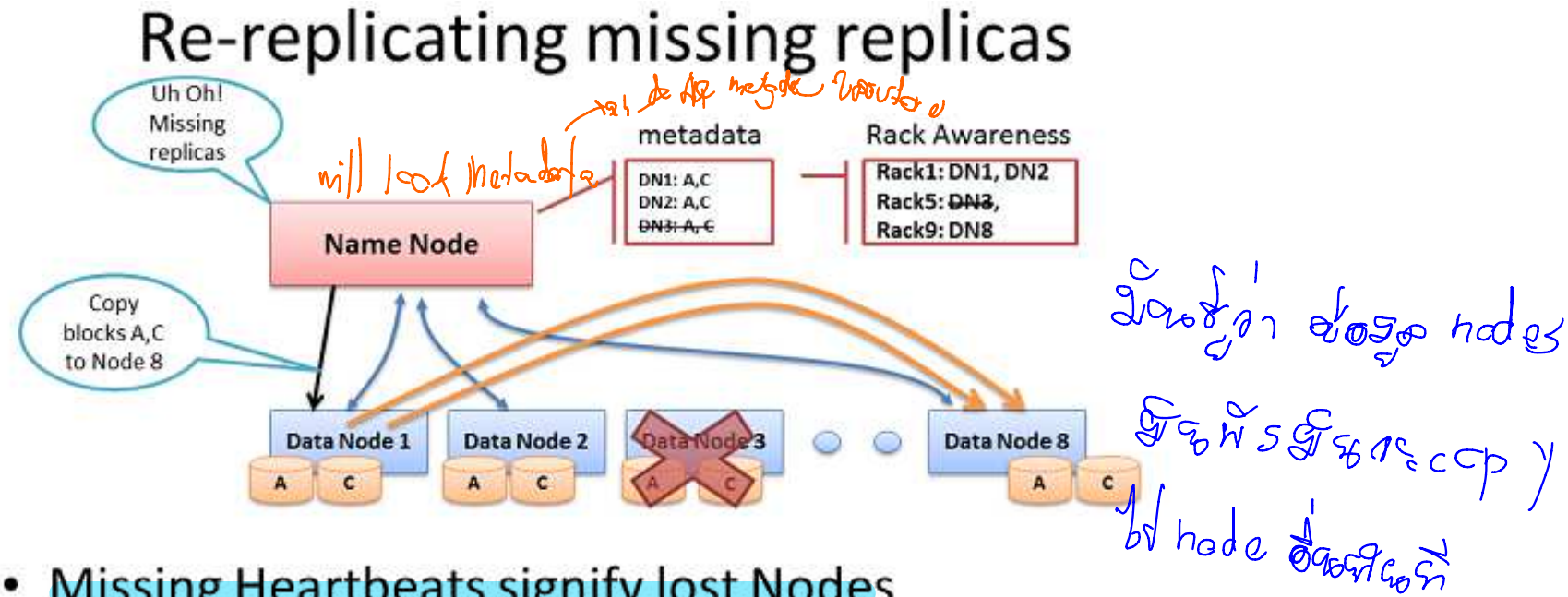
- Never lose data even if an entire rack fails



- Name Node provides rack local Nodes first
- Leverage in-rack bandwidth, single hop

BRAD HEDLUND .com

Recover from Fault



- Missing Heartbeats signify lost Nodes
- Name Node consults metadata, finds affected data
- Name Node consults Rack Awareness script
- Name Node tells a Data Node to re-replicate

need specify two function.

MapReduce

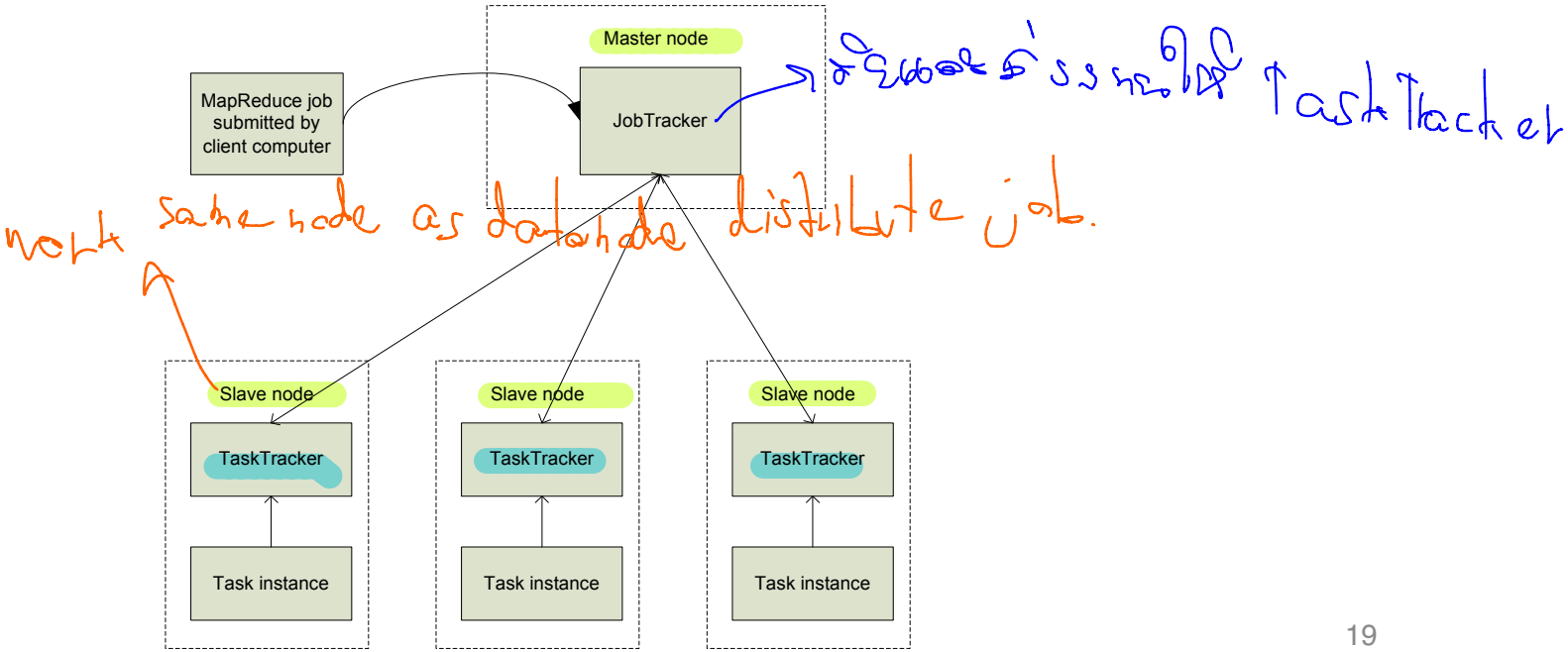
- Pioneered by Google to processes 20 petabytes of data per day
- Simple parallel programming model designed for processing **large datasets** on a large cluster of machines
  - scalability and fault-tolerance
- Programmer specifies two functions: **Map** and **Reduce** functions.
- Input & Output: each a set of **key/value** pairs
- Many real world tasks are expressible in this model

งานในโลกแห่งความเป็นจริงจำนวนมากสามารถแสดงออกได้ในรูปนี้

- MPI is designed for numerical applications

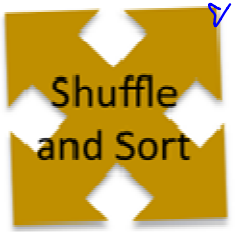
MapReduce Architecture

- **Master** node runs **JobTracker**, which accepts Job requests from clients and sends tasks to **TaskTrackers**
- TaskTracker runs on **slave** nodes
- TaskTracker forks separate process for task instances

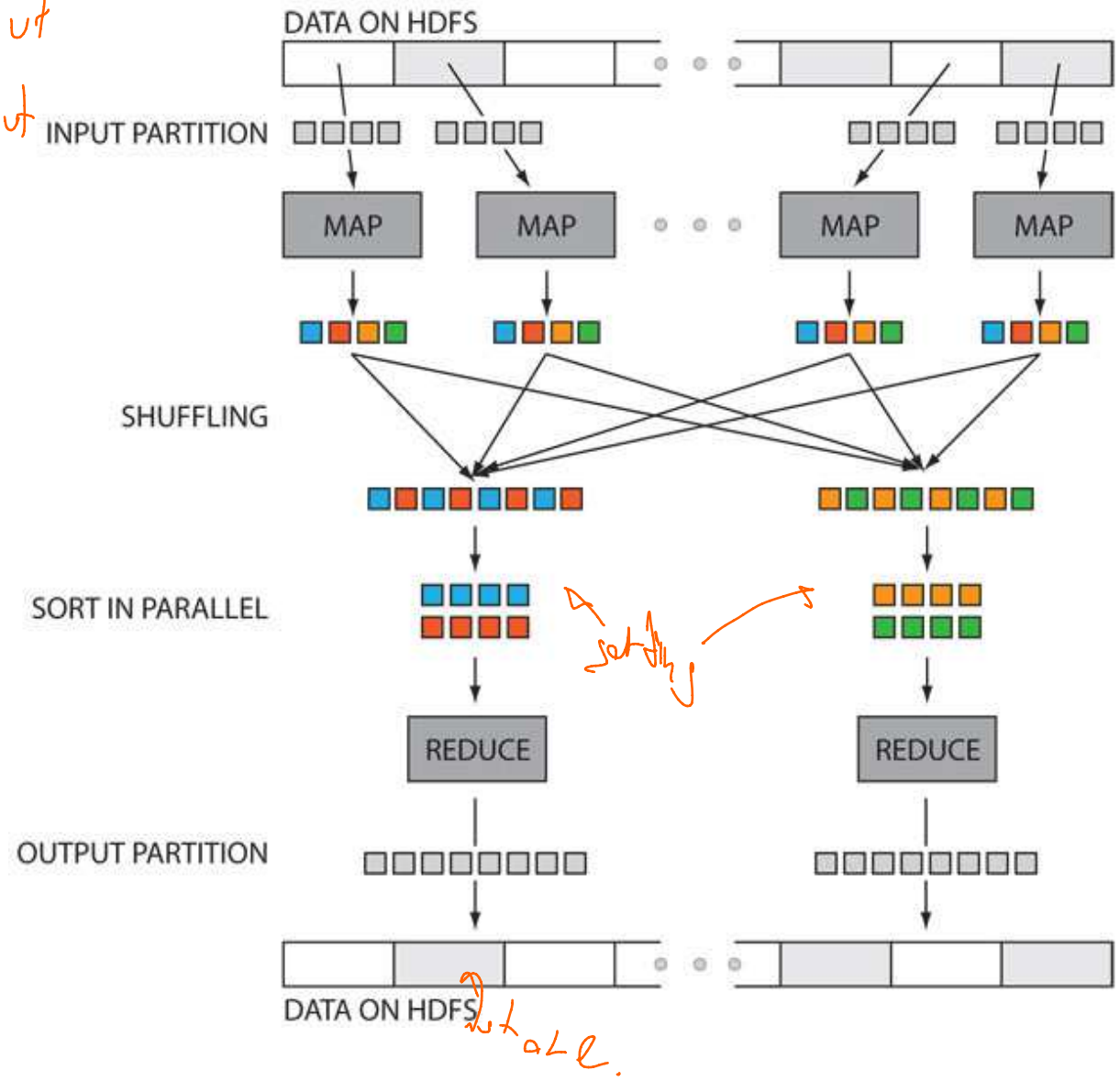


How Jobs are Executed

- **The Mapper**
  - Each Map task (typically) operates on a single HDFS block
  - Map tasks(usually) run on the node where the block is stored
- **Shuffle and Sort**
  - Intermediate data from all mappers are split
  - Then, consolidate and sort intermediate data to reducers
  - Happens after all Map tasks are complete and before Reduce tasks start
- **The Reducer**
  - Operates on shuffled/sorted intermediate data (Map task output)
  - Produces final output



How Jobs are Executed



MapReduce Input/Output

- Input is a set of key/value (k1,v1) pairs
- Map converts each input to intermediate data

  map(k1,v1) → list (k2,v2)
- Intermediate data are a list of (k2,v2) pairs
- Then, intermediate data are grouped by key
- Reduce collects and converts intermediate data into output

  reduce(k2, list (v2)) → (k3,v3)
- Output is the list of (k3,v3) pairs
- Key cannot be null, but value can

- Example: Count word frequency in text files

Wordcount Example

Input

txt. file.

Line1 Hadoop uses MapReduce
Line2 There is a Map phrase
Line3
Line4 There is a Reduce phrase

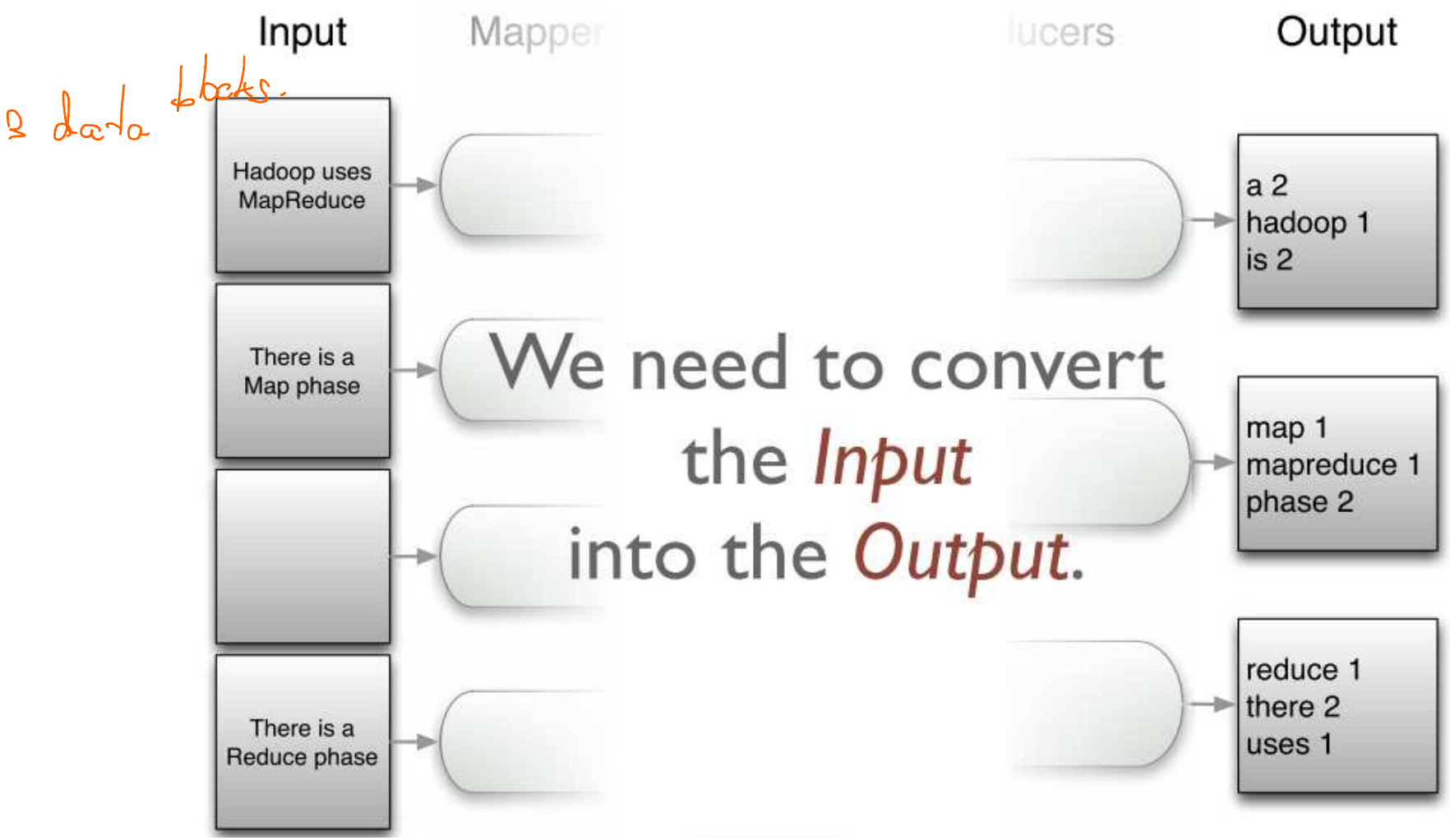
each line contain one line.

Output

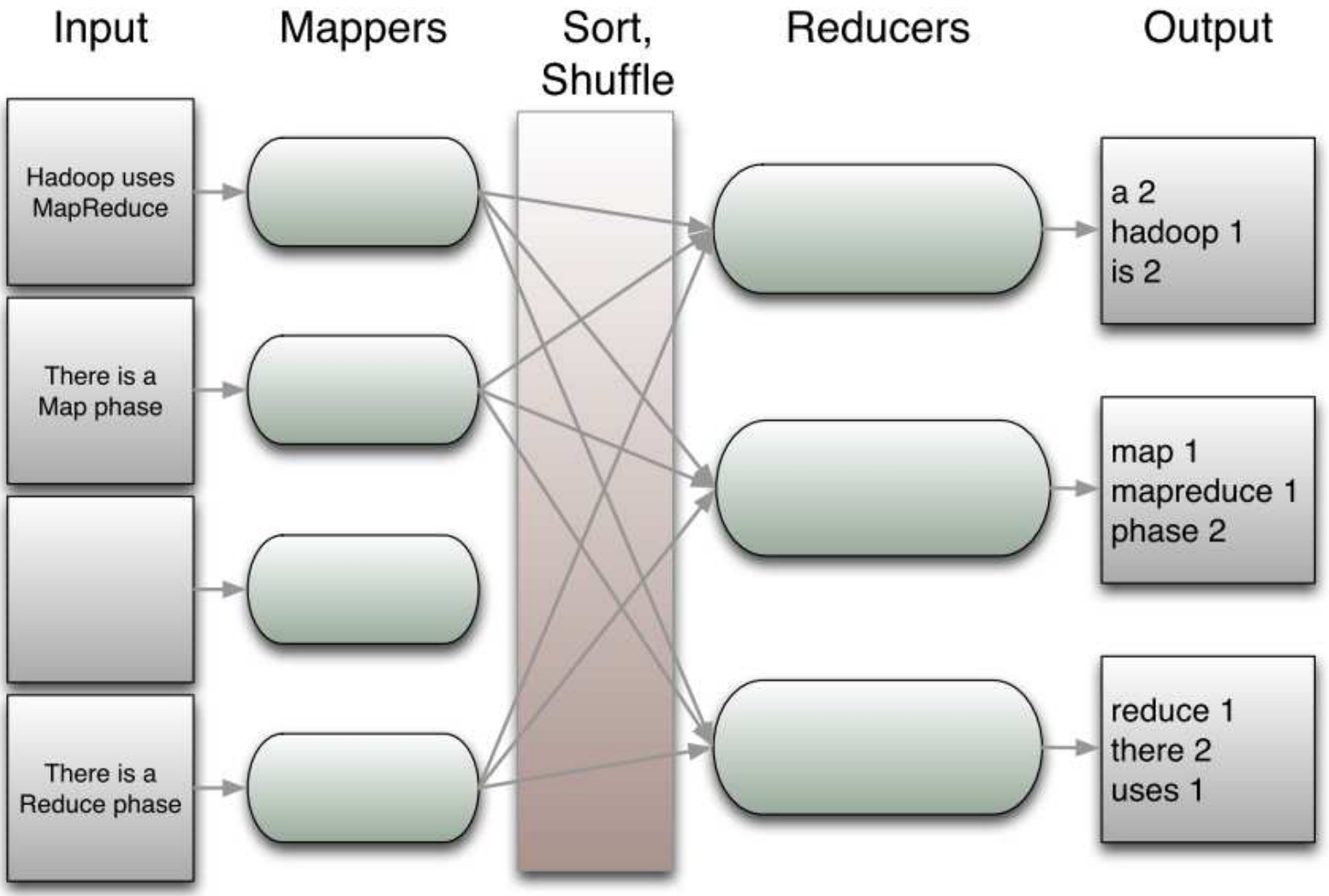
a 2
hadoop 1
is 2
map 1
mapreduce 1
phrase 2
reduce 1
there 2
uses 1

(This example ignores case sensitive)

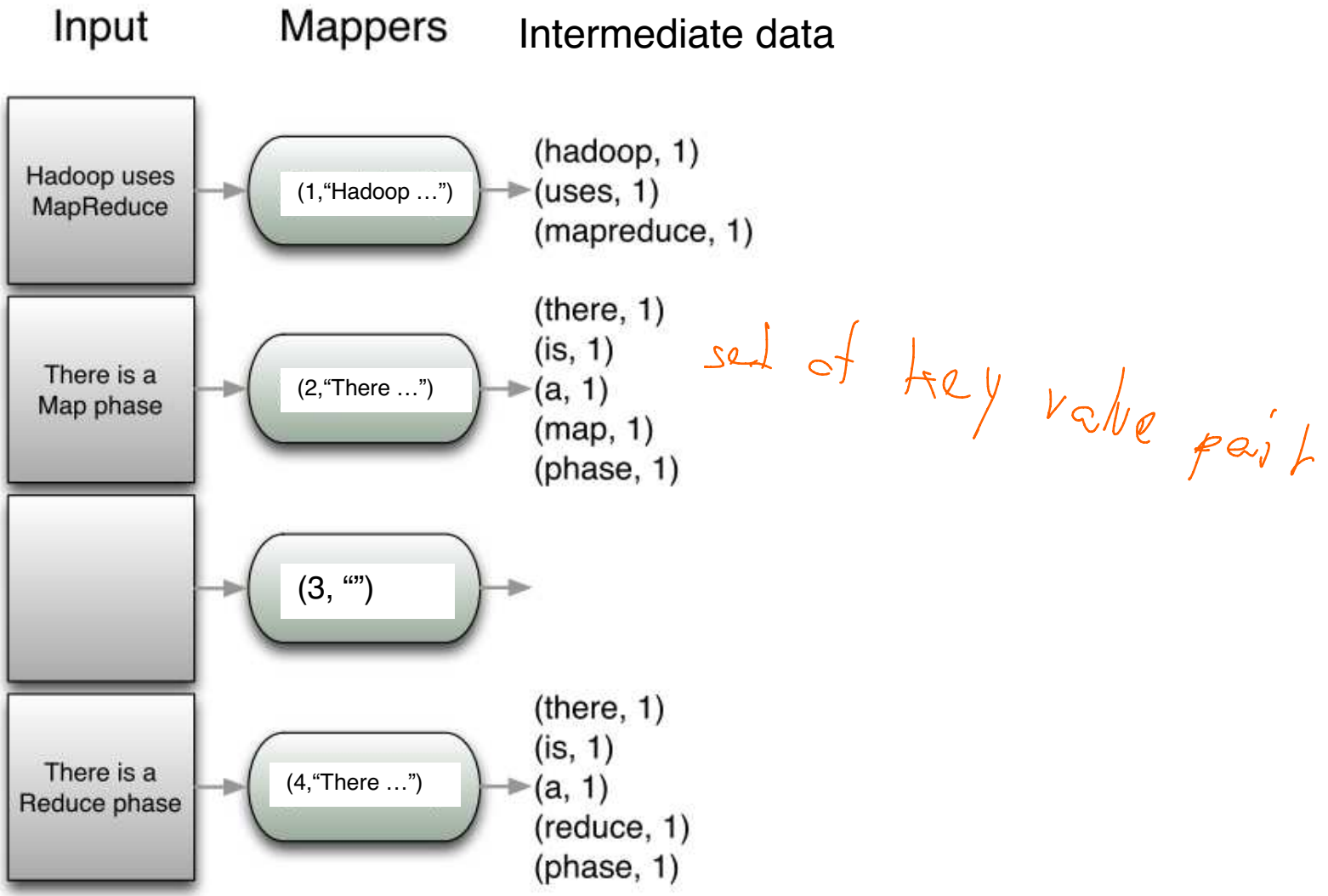
Wordcount Example



Wordcount Example

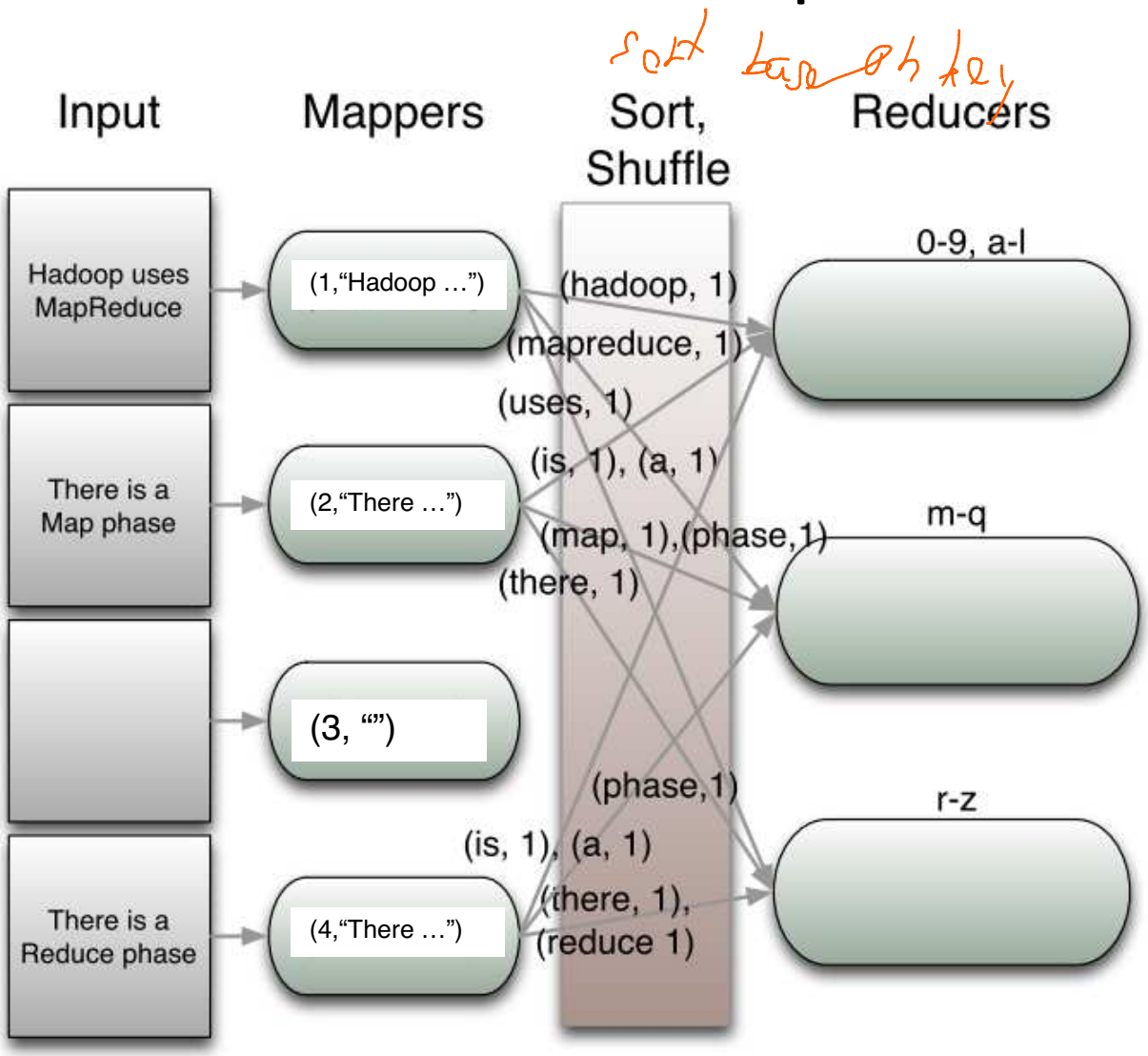


Wordcount Example

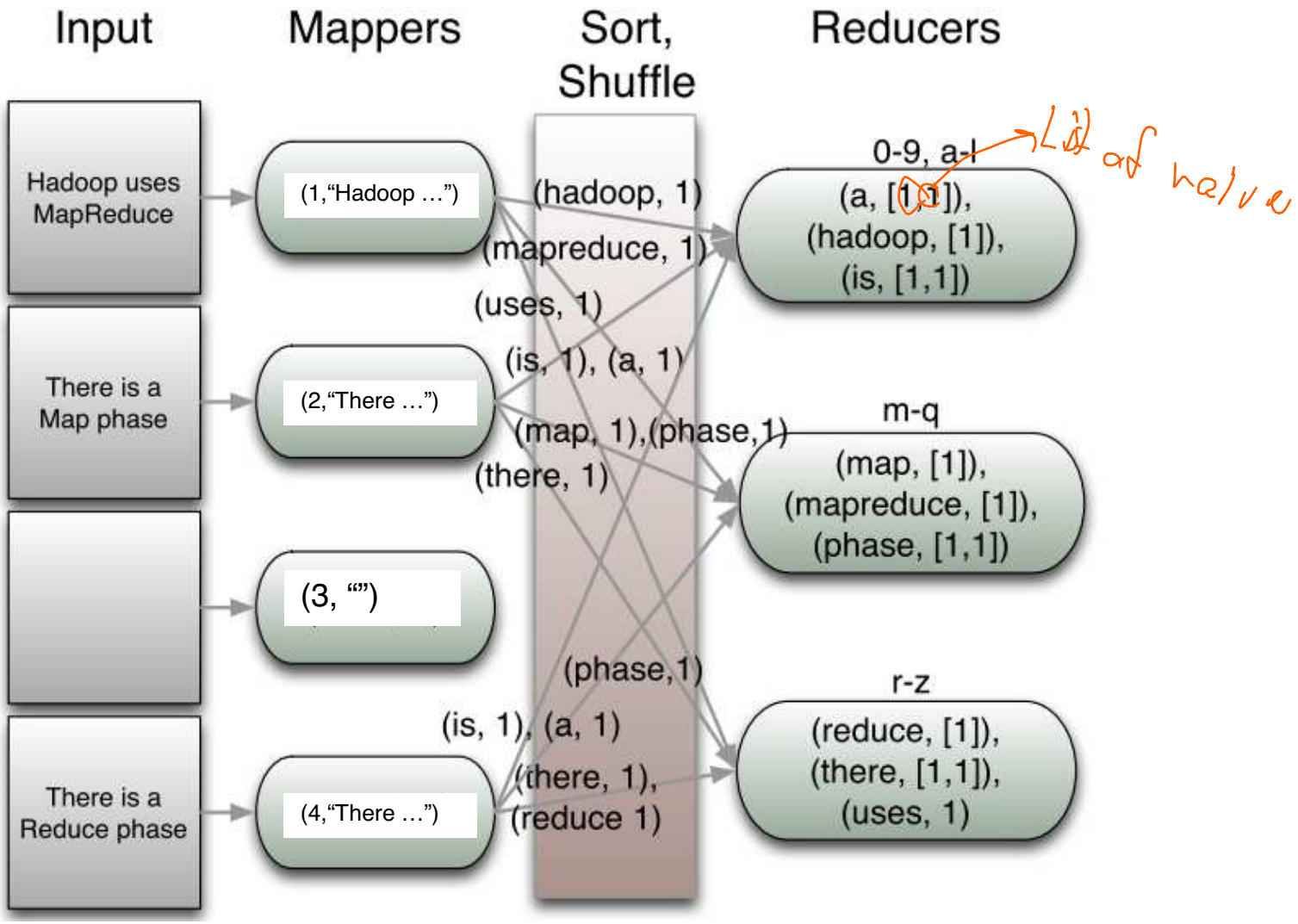


set of key value pair

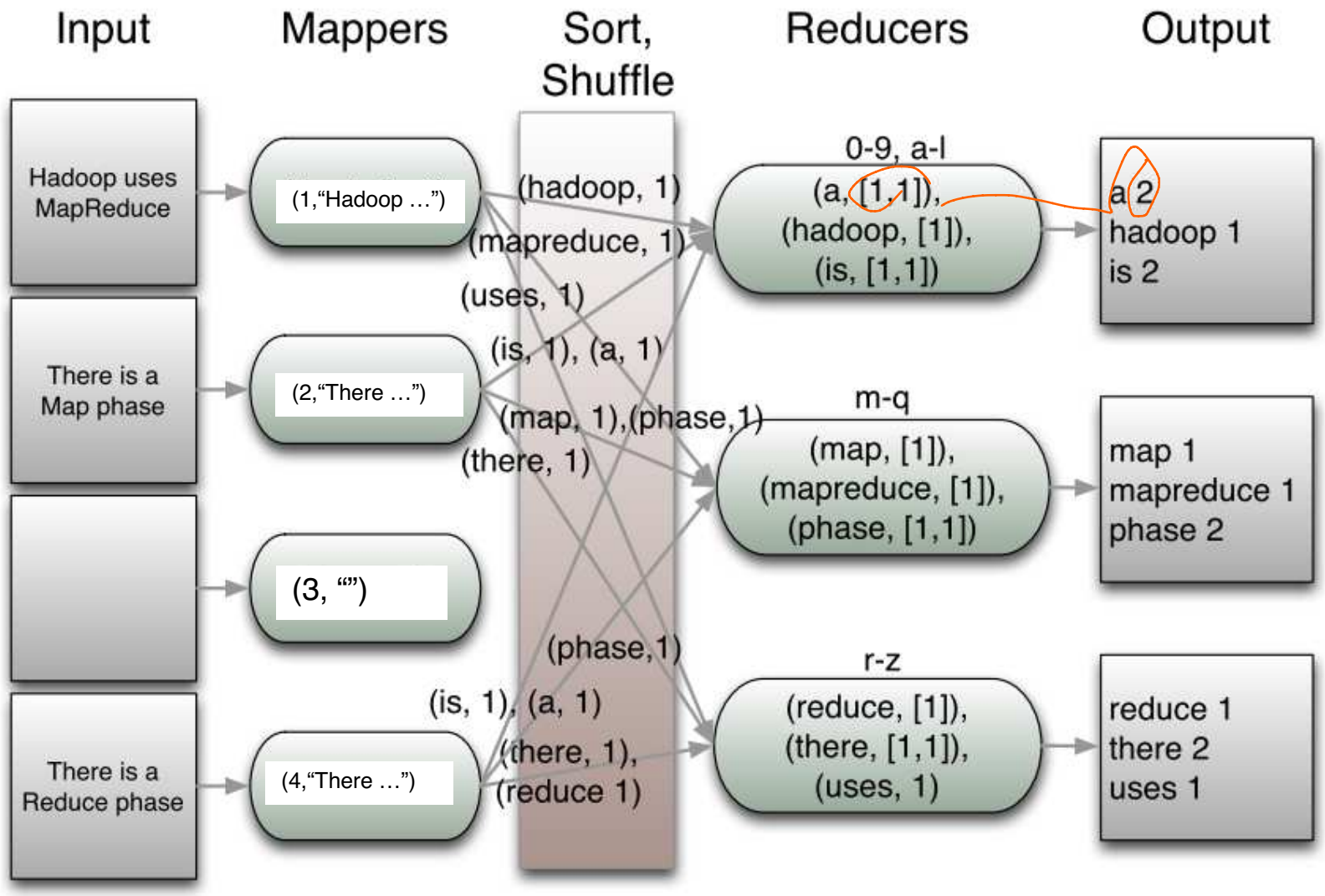
Wordcount Example



Wordcount Example



Wordcount Example



MapReduce Programming Model

For every input line (record), call a map function on the line.

Map:

- Accepts *input* key/value pair
- Emits *intermediate* key/value pairs

For every distinct intermediate key, call a reduce function

Reduce :

- Merges all *intermediate* key/value pairs with the same intermediate key Same key :merge
- Emits *output* key/value pairs (usually one)

Wordcount's Map and Reduce Functions (Pseudo Code)

```
map(key, value):
        // key: line number
        // value: line content
        for each word w in split(value," ")
              Emit(w, 1);
        end for
```

Split returns an array of words

```
reduce(key, values[ ]):
        // key: a word
        // values: a list of 1 for the same word
        sum = 0;
        for each v in values
              sum += v;
        end for
        Emit(key, sum);
```

Mean Temperature Example

- From data collected since 2010 until now, find the mean temperature of each month
- Input data has two columns: YYYYMMDDhhmm, Celsius
- Output has two columns and 12 rows each for a month

```
201001010000,25.0
201001010015,24.5
201001010030,24.0
201001010045,24.0
201001010100,23.5
201001010115,23.0
      …
201512312345,27.0
```

Mean's Map and Reduce Functions

```
map(key, value):
        // key: line number
        // value: line of datetime and degree
        (datetime,degree) = split(value,",")
        month = substring(datetime,4,2)
        Emit(month, degree);
```

Assign the first and second elements to tuple (datetime,degree)

```
reduce(key, values[ ]):
        // key: a month
        // values: a list of temperatures in the same month
        sum = 0, count=0;
        for each v in values
                sum += v;
                count += 1;
        end for
        Emit(key, sum/count);
```

Searching Keyword Example

- Example: Check if a specific keyword "*refund*" exists in a large text file

Some people believe that getting a large tax **refund** is not as desirable as more accurate withholding throughout the year, as a large **refund** represents a loan paid back by the government interest-free. Optimally, a return should result in a payment owed of just less than would cause a penalty charge, which is 100% of the prior year's tax (110% for high income individuals), 90% of the current year's tax, or $1,000 for individuals who have direct withholding and do not pay estimated tax. In order to decrease the amount of the tax **refund** which has to be received by taxpayers, they can turn to one or several of the following methods …

Searching's Map and Reduce Functions

```
map(key, value):
        // key: line number
        // value: line of text
        count = 0
        for each word w in split(value," ")
              if (w equals "refund") count += 1
        end for
        Emit(w,count)        // 0 means not found
```

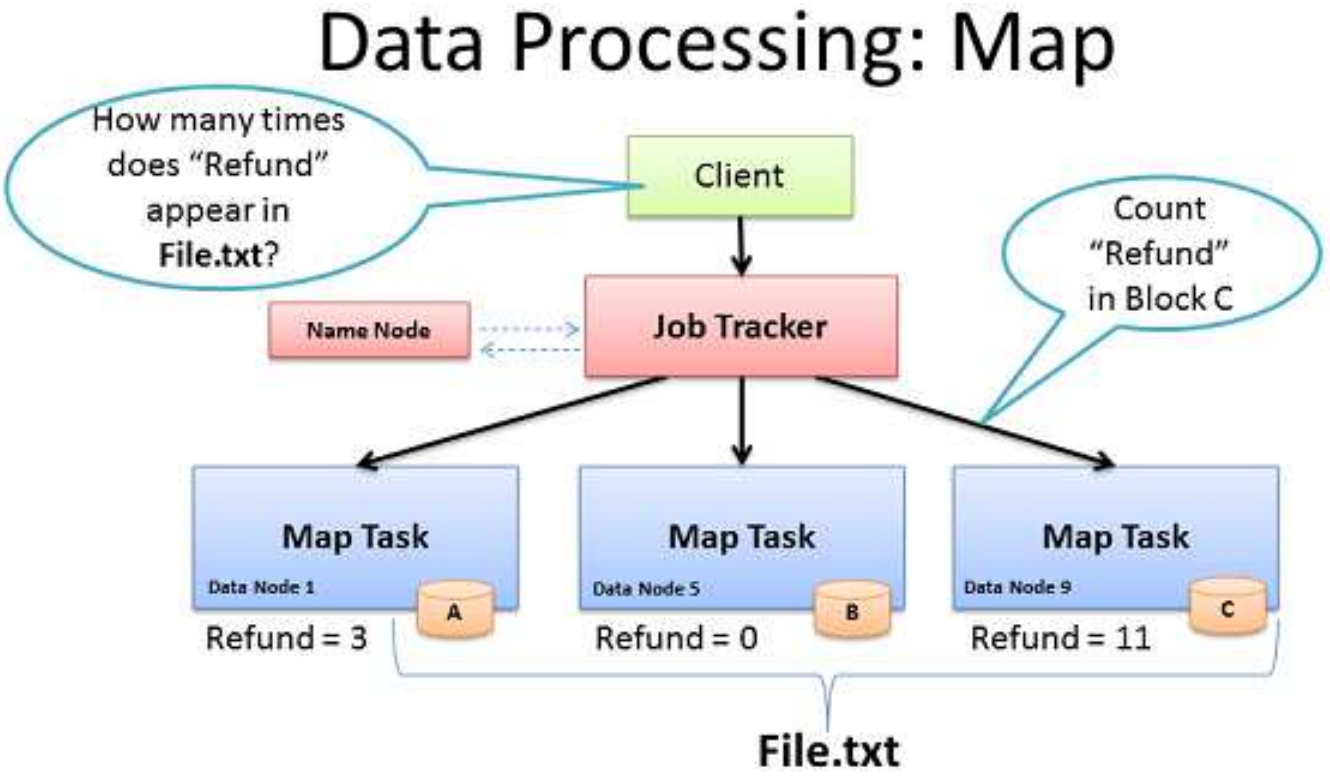
refund.

```
reduce(key, values[ ]):
       // key: word "refund"
       // values: a list of frequencies for the word "refund" in each line
       sum = 0;
       for each v in values
             sum += v;
       end for
       Emit(key, sum);   // If output has sum > 0, found
```

Hadoop Job Scheduling

- Input files are divided into blocks (default to HDFS blocks)
- A map task is created for each block
- JobTracker finds the location of block from NameNode
- JobTracker selects a node to run a map task
- Data aware scheduling
  - A map task is scheduled to run at the node where data block is located
  - Otherwise, a map task is scheduled to run at the node close to the data block

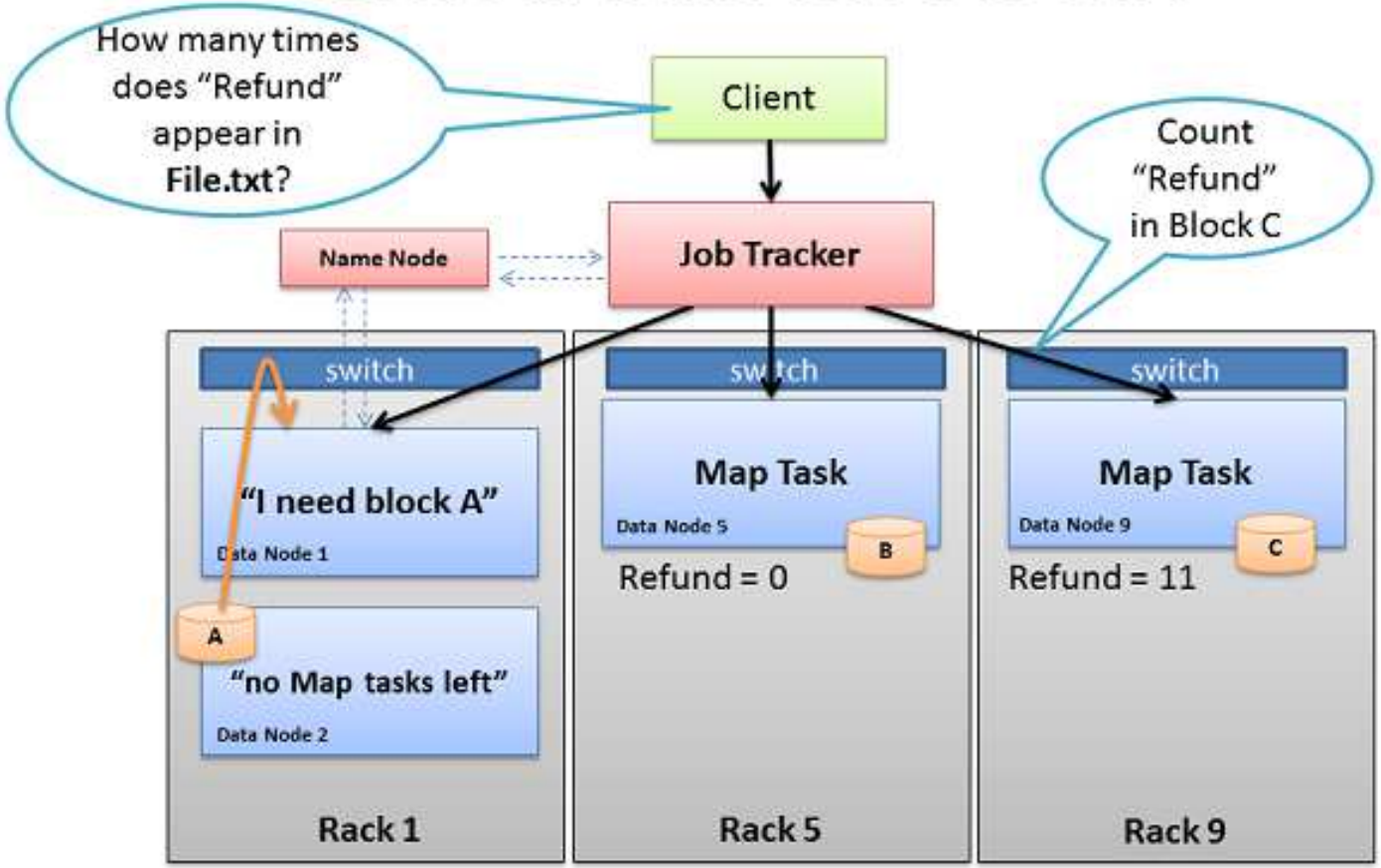
Data Aware Scheduling in Map Tasks



- **Map:** "Run this computation on your local data"
- Job Tracker delivers Java code to Nodes with local data

BRAD HEDLUND .com

What if data isn't local?



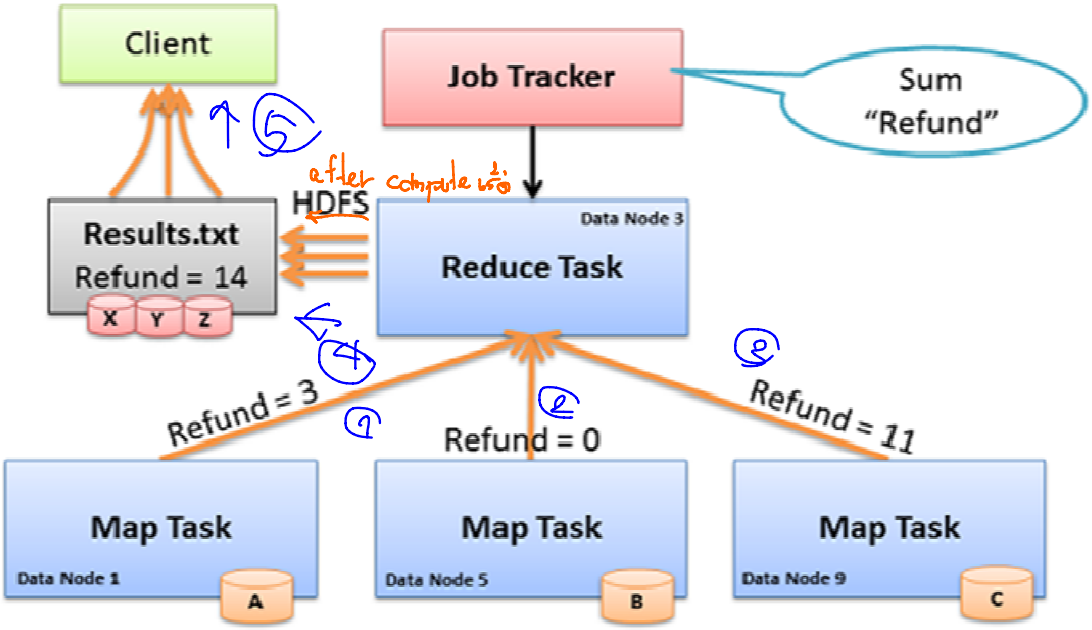
- Job Tracker tries to select Node in same rack as data
- Name Node rack awareness

BRAD HEDLUND .com

Reduce Task Scheduling

- Reduce tasks are placed at random nodes
- The intermediate data from mappers are split to each reducer based on the **hash or range partition** of the intermediate key
- For wordcount example, with 2 reducers, using range partition, words begin with a-m is assigned to reducer1, and n-z is assigned to reducer2
- Each reduce task collects intermediate data of the assigned keys from all mappers

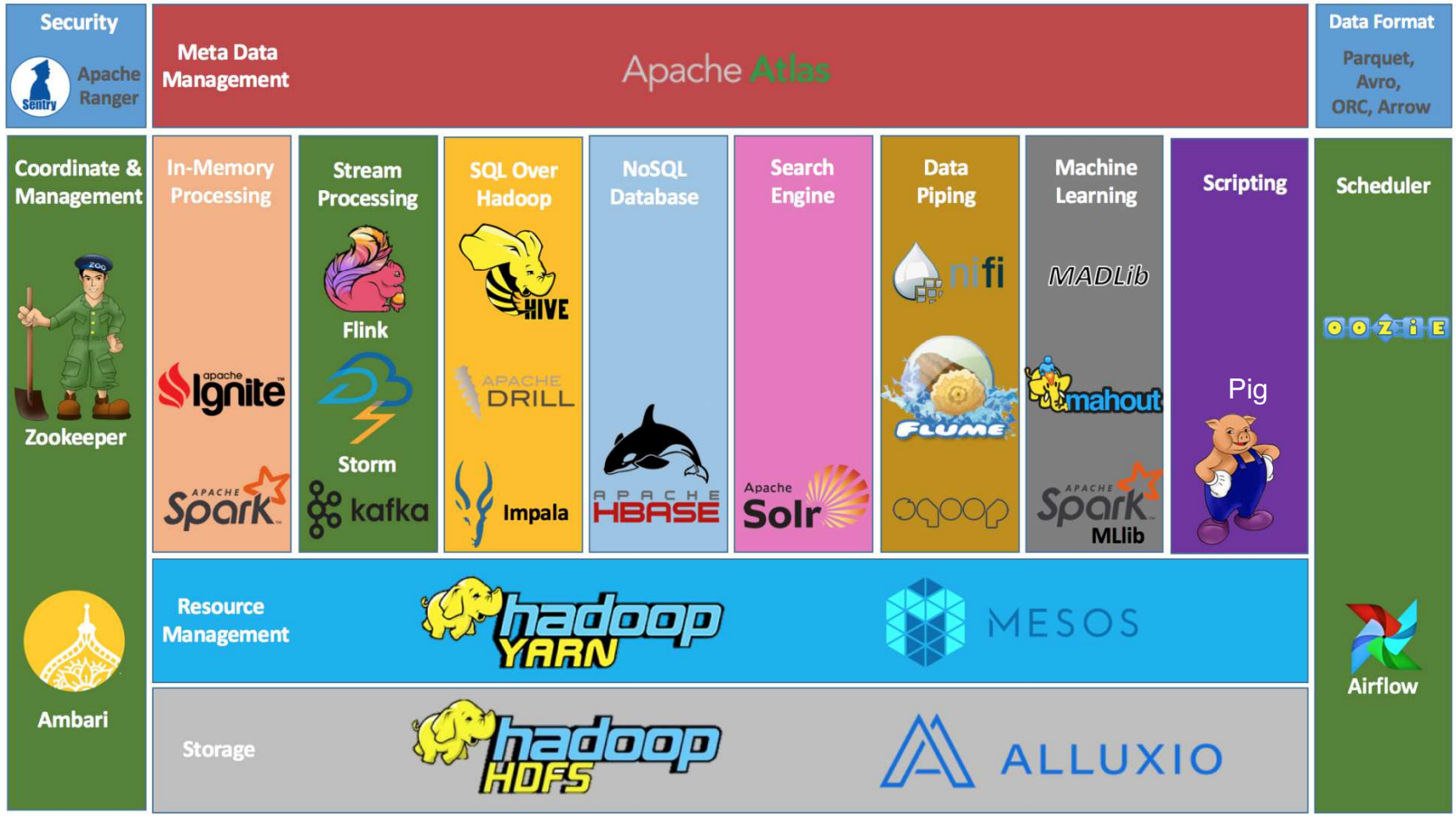
Data Processing: Reduce



- **Reduce:** "Run this computation across Map results"
- Map Tasks send output data to Reducer over the network
- Reduce Task data output written to HDFS

BRAD HEDLUND .com

Hadoop Ecosystem



HBASE

- A NoSQL datastore

- Clone of Big Table (Google)
- Implemented in Java (Clients : Java, C++, Ruby...)
- Data is stored "Column-oriented"
- Distributed over many servers
- Tolerant of machine failure
- Layered over HDFS
- Strong consistency

- It's not a relational database (No joins)
- Sparse data – nulls are stored for free
- Semi-structured or unstructured data
- Data changes through time
- Versioned data
- Scalable – Goal of billions of rows x millions of columns

Table - Example

Row	Timestamp	Animal		Repair
		Type	Size	Cost
Enclosure1	12	Zebra	Medium	1000€
	11	Lion	Big	
Enclosure2	13	Monkey	Small	1500€

Region

Key

Column

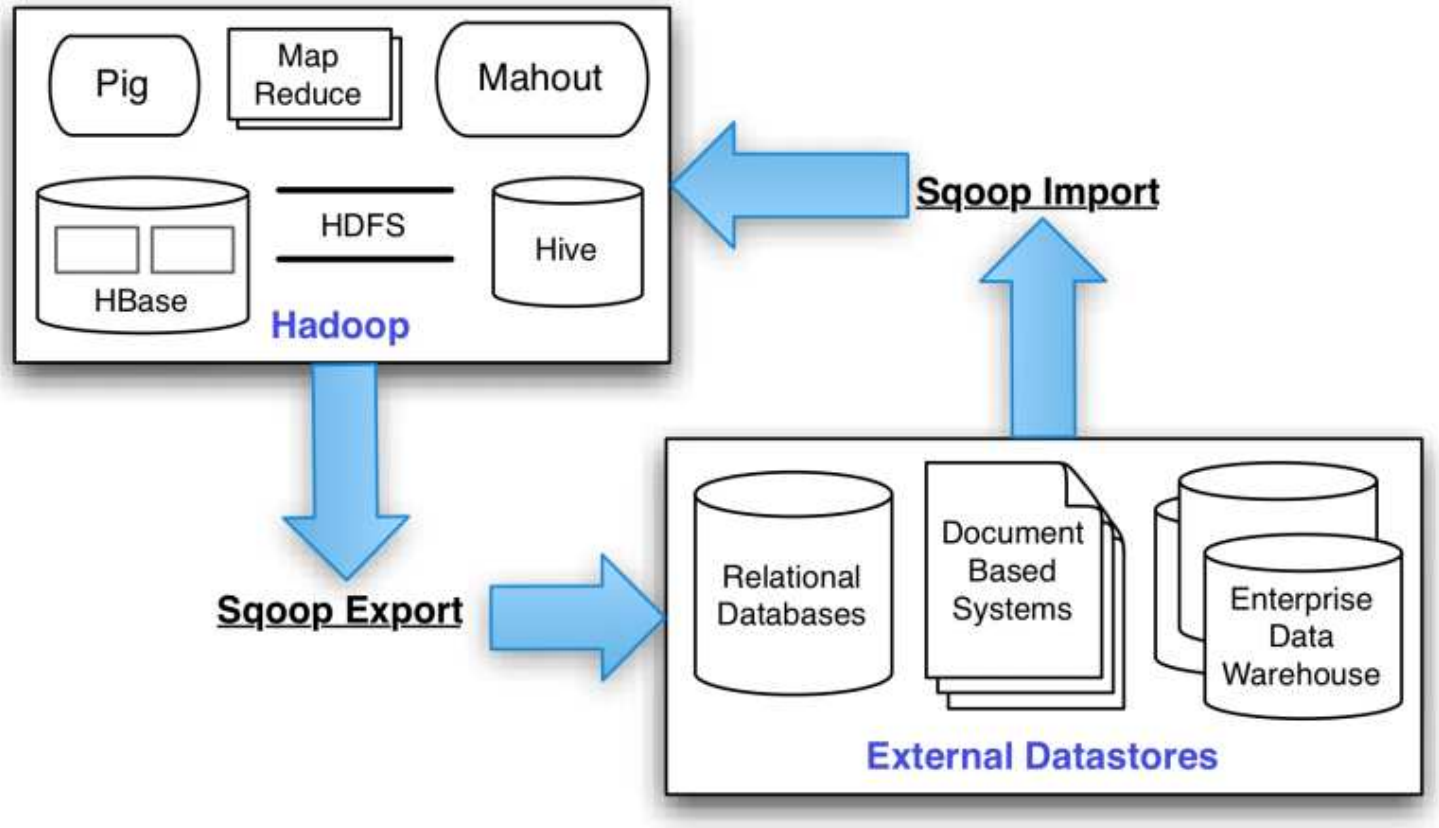
Family

Cell

(Table, Row_Key, Family, Column, Timestamp) = Cell (Value)

Sqoop

- A tool for data import/export between Hadoop and RDBMS



Flume

- Move large amount of streaming event (log data) into Hadoop

Access log:
127.0.0.1 - frank [10/Oct/2000:13:55:36 -0700] "GET /apache_pb.gif HTTP/1.0" 200 2326

Error log:
[Sun Mar 7 20:58:27 2004] [info] [client 64.242.88.10] (104)Connection reset by peer: client stopped connection before send body completed
[Sun Mar 7 21:16:17 2004] [error] [client 24.70.56.49] File does not exist: /home/httpd/twiki/view/Main/WebHome

Vmstat

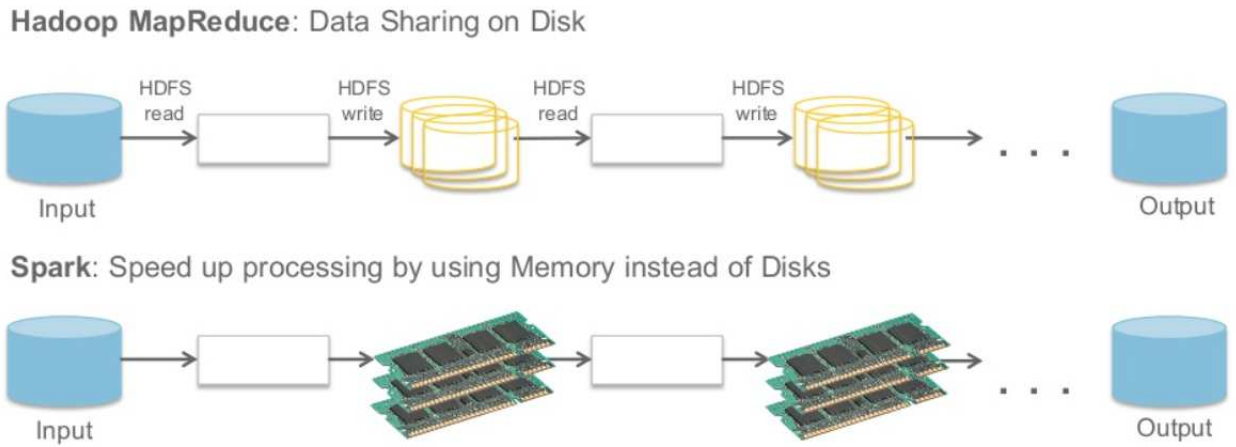
```
procs -----------memory---------- ---swap-- -----io---- --system-- -----cpu------
 r  b   swpd   free   buff  cache   si   so    bi    bo   in   cs us sy id wa st
 0  0 305416 260688  29160 2356920    2    2     4     1    0    0  6  1 92  2  0
```

iostat

```
Linux 2.6.32-100.28.5.el6.x86_64 (dev-db)       07/09/2011

avg-cpu:  %user   %nice %system %iowait  %steal   %idle
           5.68    0.00    0.52    2.03    0.00   91.76
```

- Supersede Hadoop MapReduce
- Can utilize both in-memory and disk storage

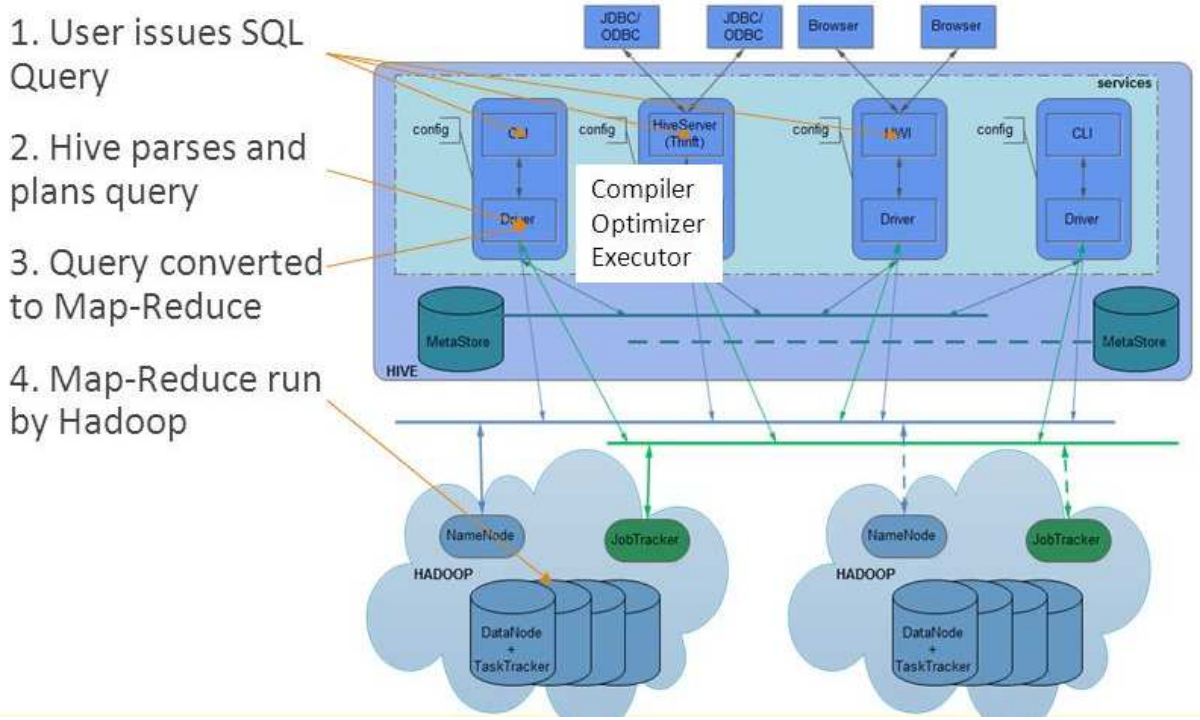


- More operations than map/reduce
- Support Scala, Java, Python, R
- Scale from a single node to a cluster

Hive

- Hive is an SQL-like interface to Hadoop

```
SELECT * FROM purchases WHERE price > 10000 ORDER BY storeid
```



Additional References

- https://hadoop.apache.org/
- http://biforbeginners.blogspot.com/2012/08/hadoop-basics.html

Q & A