Cloud Computing (INFS3208)

Lecture 11: Hadoop Ecosystem

Lecturer: Dr Sen Wang

School of Electrical Engineering and Computer Science

Faculty of Engineering, Architecture and Information Technology

The University of Queensland

Caution

- If your GCP credit runs out, please contact Mr Jason Liang;
- All assignments/practical materials can be implemented/deployed locally;
- No extension request will be considered by the school due to the insufficient GCP credit.
- Read more details about the extension policy and late submission policy

Submission guidelines

Online Submission.

Deferral or extension

You may be able to apply for an extension.

The maximum extension allowed is 7 days. Extensions are given in multiples of 24 hours.

Marked assignments with feedback and/or detailed solutions with feedback will be released to students within 14-21 days, where the earlier time frame applies if there are no extensions.

Late submission

A penalty of 10% of the maximum possible mark will be deducted per 24 hours from time submission is due for up to 7 days. After 7 days, you will receive a mark of 0.

Recap

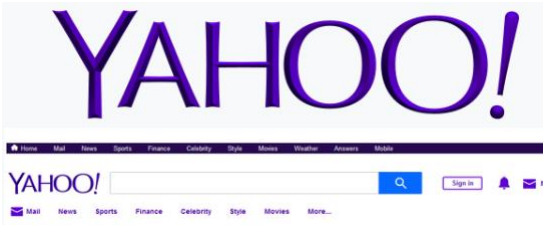
- Background of Distributed File Systems
  - Big Data, Data Centre Technology, and Storage Hardware
- Distributed File System
  - File System
  - Server/Client System
  - Sun's Network File System (NFS)
- Clustered File System (CFS)
  - Google File System (GFS)
  - Hadoop Distributed File System (HDFS)
  - HDFS Shell Commands

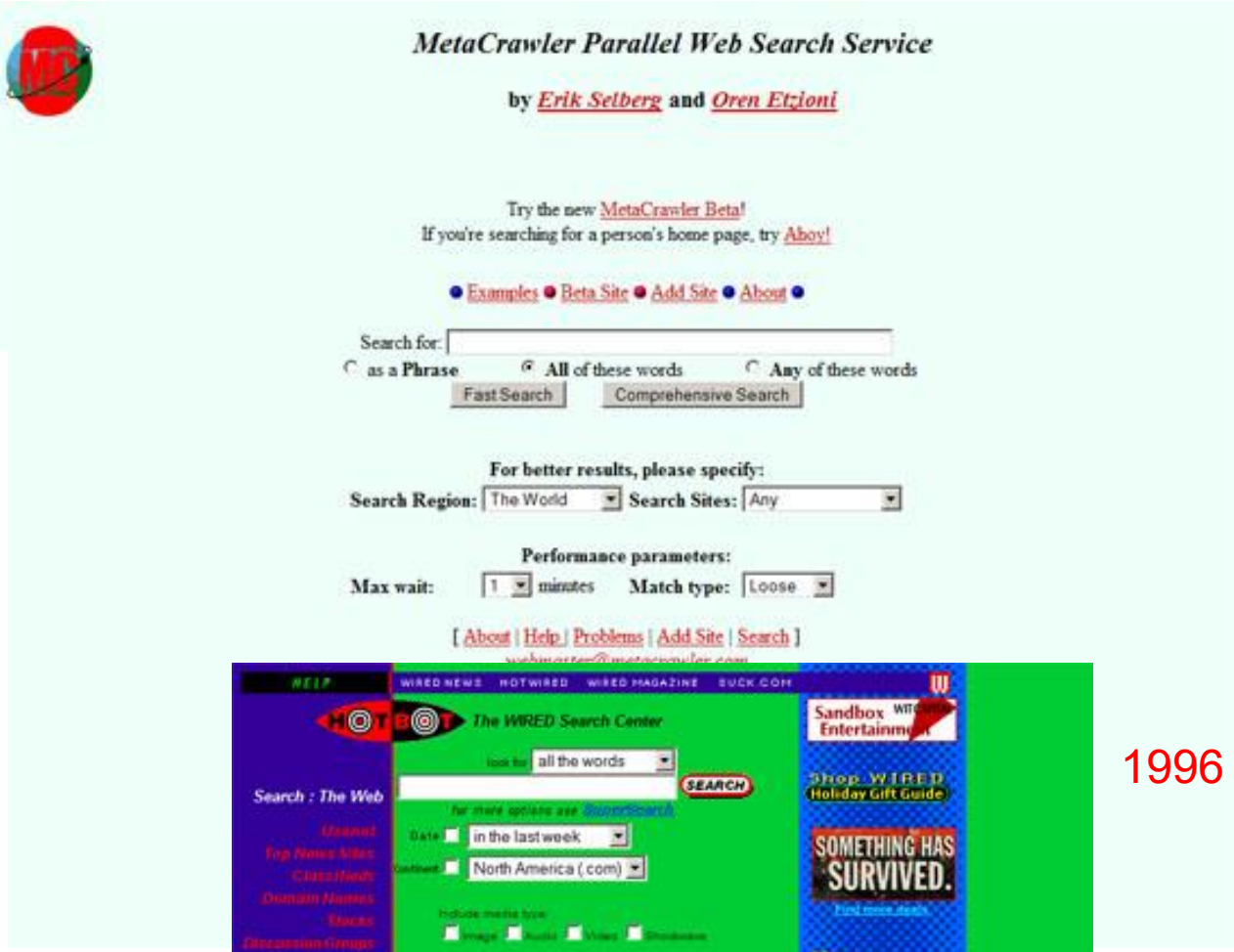
Outline

- Introduction to Hadoop
  - What is Hadoop & History
  - Hadoop Ecosystem
- Hadoop Computation Model: MapReduce
  - MapReduce Components & Paradigm
  - MapReduce Workflow
  - MapReduce Examples:
    - Word Count
    - Find the Highest/Averaged Temperature
    - Find Word Length Distribution
    - Find Mutual Friends
    - Inverted Indexing
- Besides the Core: Hive and Pig

Search Engines in 1990s

1995

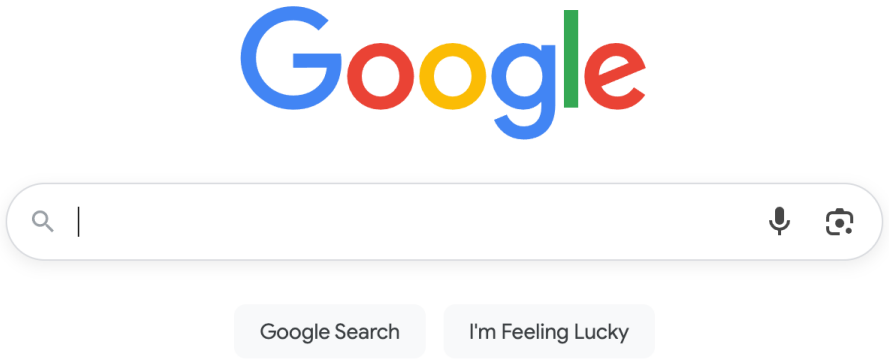




1996

Google search engines

1998



2025

Google Search works in three stages:

1. **Crawling:** Google downloads text, images, and videos from pages it found on the internet with automated programs called crawlers.
2. **Indexing:** Google analyzes the text, images, and video files on the page, and stores the information in the Google index, which is a large database.
3. **Serving search results:** When a user searches on Google, Google returns information that's relevant to the user's query.

Distributed File Systems (DFSs)
Big Data Computation Models
(MapReduce)

https://developers.google.com/search/docs/fundamentals/how-search-works

What is Hadoop?

- Apache Hadoop's MapReduce and HDFS components were inspired by Google papers on MapReduce and Google File System (GFS).
- is a collection of open-source software utilities that facilitate using a network of many computers to solve problems involving massive amounts of data and computation.
- provides a software framework for distributed storage and processing of big data using the MapReduce programming model.
- originally designed for clusters of cheap commodity hardware with fundamental assumptions:
  - Abstract and facilitate the storage and processing of large and/or rapidly growing data sets
    - Structured and non-structured data
    - Simple programming models
  - High scalability and availability
  - Fault-tolerance (failures are common)
  - Move computation rather than data

Dean, Jeffrey, and Sanjay Ghemawat. "MapReduce: Simplified data processing on large clusters." (2004).
Ghemawat, Sanjay, Howard Gobioff, and Shun-Tak Leung. "The Google file system." (2003).

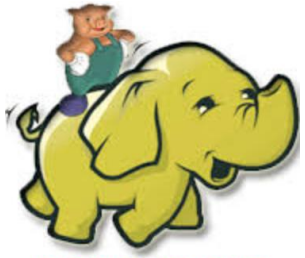
Hadoop Core and Sub-modules

- The core of Apache Hadoop consists of:
  - Storage: Hadoop Distributed File System (HDFS),
  - Computation model: MapReduce programming model.
- The base Apache Hadoop framework is composed of the following modules:
  - Hadoop Common – contains libraries and utilities needed by other Hadoop modules;
  - Hadoop Distributed File System (HDFS) – a distributed file-system;
  - Hadoop MapReduce – an implementation of the MapReduce programming model for big data processing.
  - Hadoop YARN – a cluster manager;
    Hadoop HBase - a NoSQL
  - Hadoop Hive - a data warehouse that supports high-level SQL-like query language

Dean, Jeffrey, and Sanjay Ghemawat. "MapReduce: Simplified data processing on large clusters." (2004).
Ghemawat, Sanjay, Howard Gobioff, and Shun-Tak Leung. "The Google file system." (2003).

Additional Packages in Hadoop

- The term Hadoop is often used for both base modules and sub-modules and also the ecosystem,
- Some additional software packages that can be installed on top of or alongside Hadoop are also included:
  - Apache Pig: is a high-level platform for creating programs that run on Apache Hadoop
  - Apache Hive: is a data warehouse software project built on top of Apache Hadoop for providing data query and analysis
  - Apache HBase: is an open-source, distributed, versioned, non-relational database inspired by Google BigTable
  - Apache Phoenix: is an open source, massively parallel, relational database engine supporting OLTP for Hadoop using Apache HBase as its backing store.
  - Apache Spark: is an in-memory computing platform
  - etc.



APACHE PIG

Why use Hadoop?

- Need to process Multi Petabyte Datasets
- Data may not have strict schema
- Expensive to build reliability in each application
- Nodes fails everyday
- Need common infrastructure
- Very Large Distributed File System
- Assumes Commodity Hardware on heterogeneous OS
- Optimized for Batch Processing

Brief History of Hadoop

Designed to answer the question: **"How to process big data with reasonable cost and time?"**

Hadoop was created by Doug Cutting and has its origins in Apache Nutch, an open-source web search engine.

Inspired by Google's GFS and MapReduce papers, development started and then moved to the new Hadoop subproject in Jan 2006.

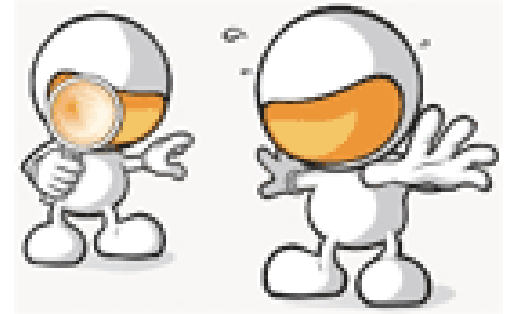
The name of Hadoop was named after a toy elephant of Doug Cutting's son.

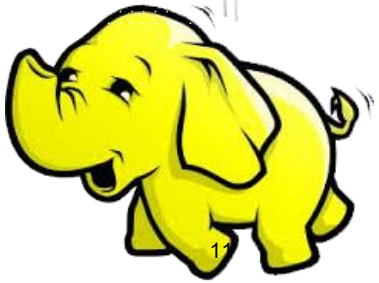
The initial code just includes 5,000 lines of code for HDFS and about 6,000 lines for MapReduce.

Milestones:

- 2006/April – Hadoop won the sorting competition (1.8T on 188 nodes in 47.9 hours)
- 2013 – Hadoop 1.1.2 and Hadoop 2.0.3 alpha.
- 2014 – Hadoop 2.3 and became top-level Apache Project
- 2019 – Apache Hadoop 3.2
- 2022 – Apache Hadoop 3.3.4
- 2023 – Apache Hadoop 3.3.6
- 2024 – Apache Hadoop 3.4.0
- 2025 – Apache Hadoop 3.4.2

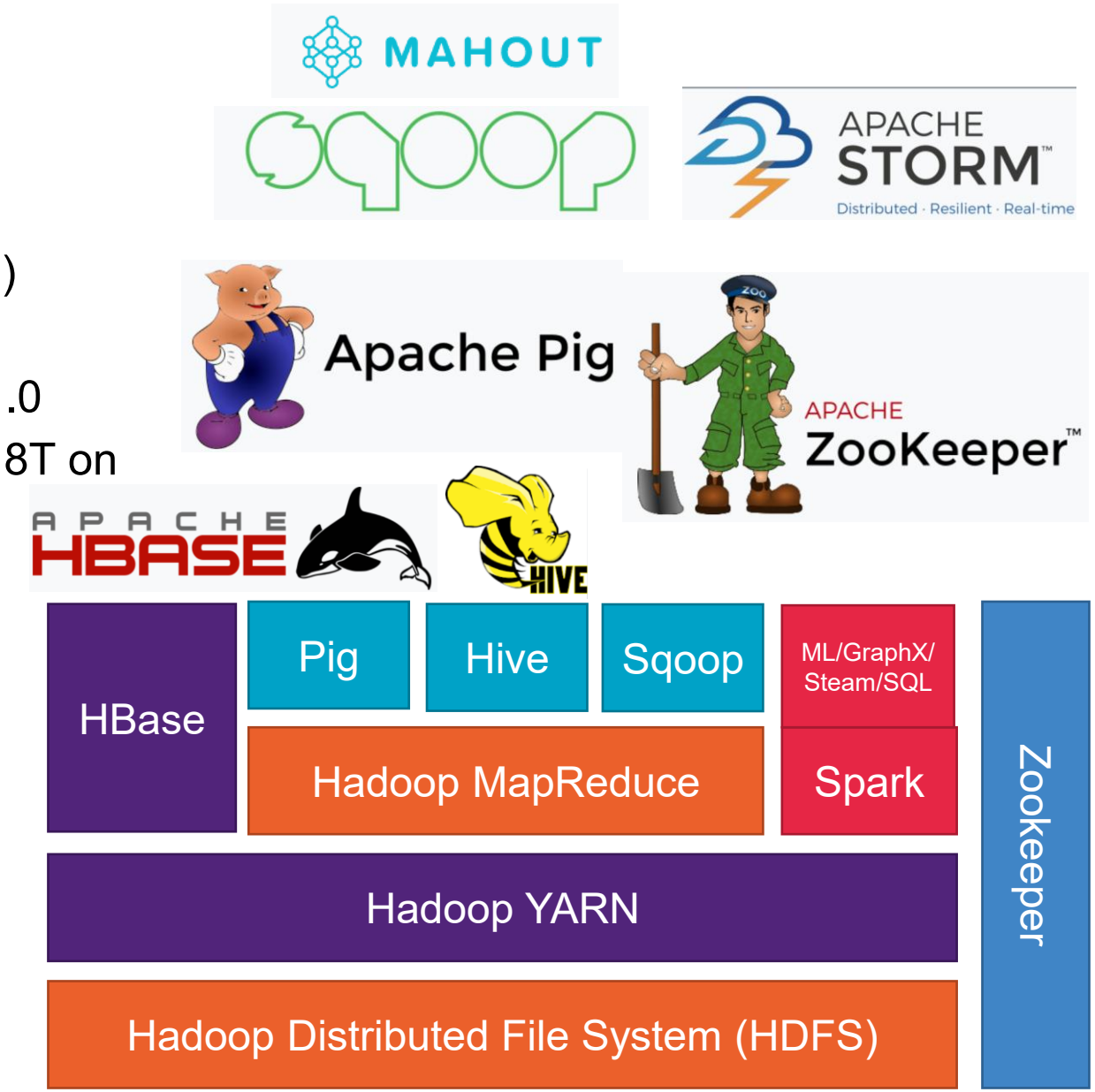
Doug Cutting





Hadoop Ecosystem

- 2003 – Google File System (GFS)
- 2004 – MapReduce computation model (Google)
- 2006 – Hadoop was born
- 2006/April – HDFS + MapReduce in Hadoop 0.1.0
- 2006/April – Hadoop won sorting competition (1.8T on 188 nodes in 47.9 hours)
- 2006 – 2007 – Yahoo contributed to Hadoop
- 2008/March – HBase released
- 2008/Sept – Pig released
- 2009/June – Sqoop released
- 2010/Oct – Hive released
- 2011/Feb – Zookeeper released
- 2012 – Hadoop YARN released
- 2014/May – Spark released

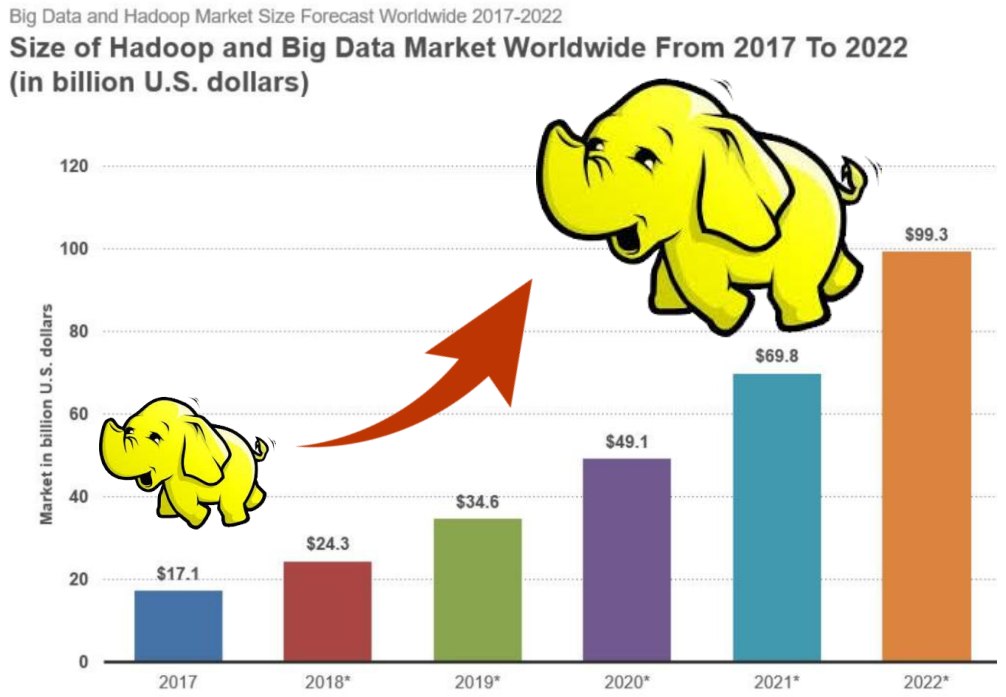


Hadoop in the Wild

- Hadoop is in use at most organizations that handle big data:
  - Yahoo!
  - Facebook
  - Amazon
  - Netflix
  - and more…
- Main applications using Hadoop:
  - Advertisement (Mining user behavior to generate recommendations)
  - Searches (group related documents)
  - Security (search for uncommon patterns)

https://www.statista.com/statistics/593479/worldwide-hadoop-bigdata-market/

- *Adobe*
  - *We use Apache Hadoop and Apache HBase in several areas from social services to structured data storage and processing for internal use.*
  - *We currently have about 30 nodes running HDFS, Hadoop and HBase in clusters ranging from 5 to 14 nodes on both production and development. We plan a deployment on an 80 nodes cluster.*
  - *We constantly write data to Apache HBase and run MapReduce jobs to process then store it back to Apache HBase or external systems.*
  - *Our production cluster has been running since Oct 2008.*

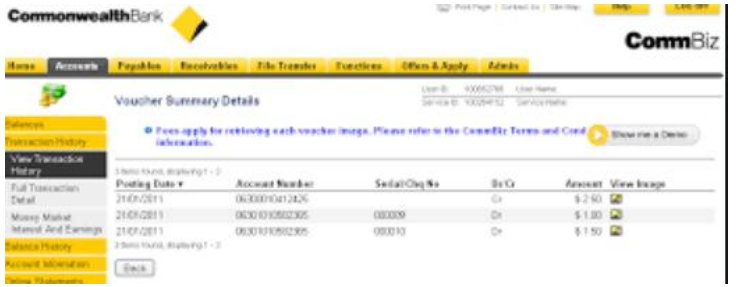


Outline

- Introduction to Hadoop
  - What is Hadoop & History
  - Hadoop Ecosystem
- Hadoop Computation Model: MapReduce
  - MapReduce Components & Paradigm
  - MapReduce Workflow
  - MapReduce Examples:
    - Word Count
    - Find the Highest/Averaged Temperature
    - Find Word Length Distribution
    - Find Mutual Friends
    - Inverted Indexing
- Besides the Core: Hive and Pig

MapReduce: A Real-World Analogy

Coins Deposit



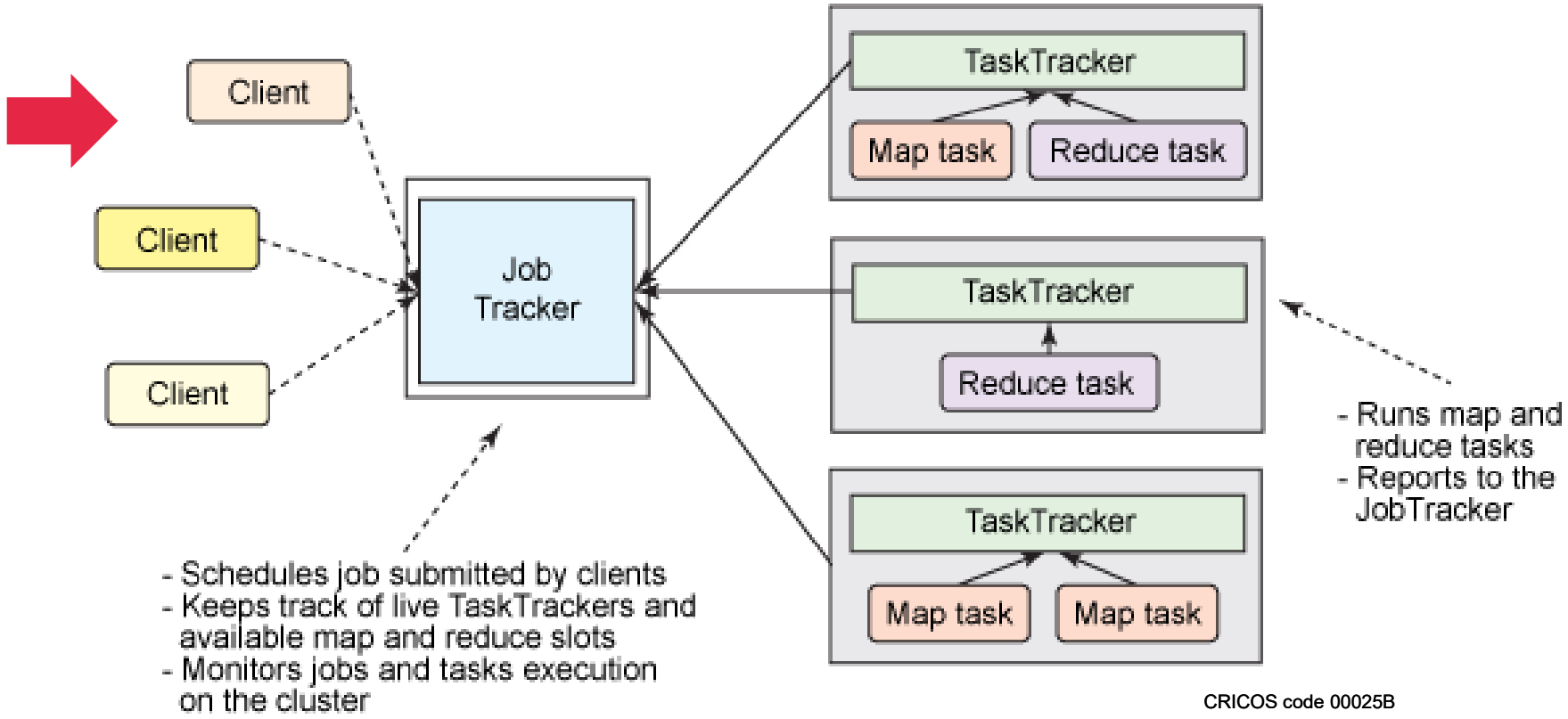
Mapper: Categorize coins by their face values
Reducer: Count the coins in each face value ***in parallel***

MapReduce Architecture: Components

- Job Client:
  - MapReduce program by users will be submitted to JobTracker via Client
  - Users can display job running status through interfaces in Client
- JobTracker:
  - Monitor resources and coordinate jobs
  - Monitor health of all the TaskTrackers (transfer jobs to other nodes once failure found)
  - Monitor execution percentage of jobs and resources availability
- TaskTracker:
  - Periodically heartbeat with resource information job execution status to JobTracker
  - Receive and execute commands from JobTracker (start new tasks or kill existing tasks)
- Task:
  - Map Task and Reduce Task
  - Initiated by TaskTracker

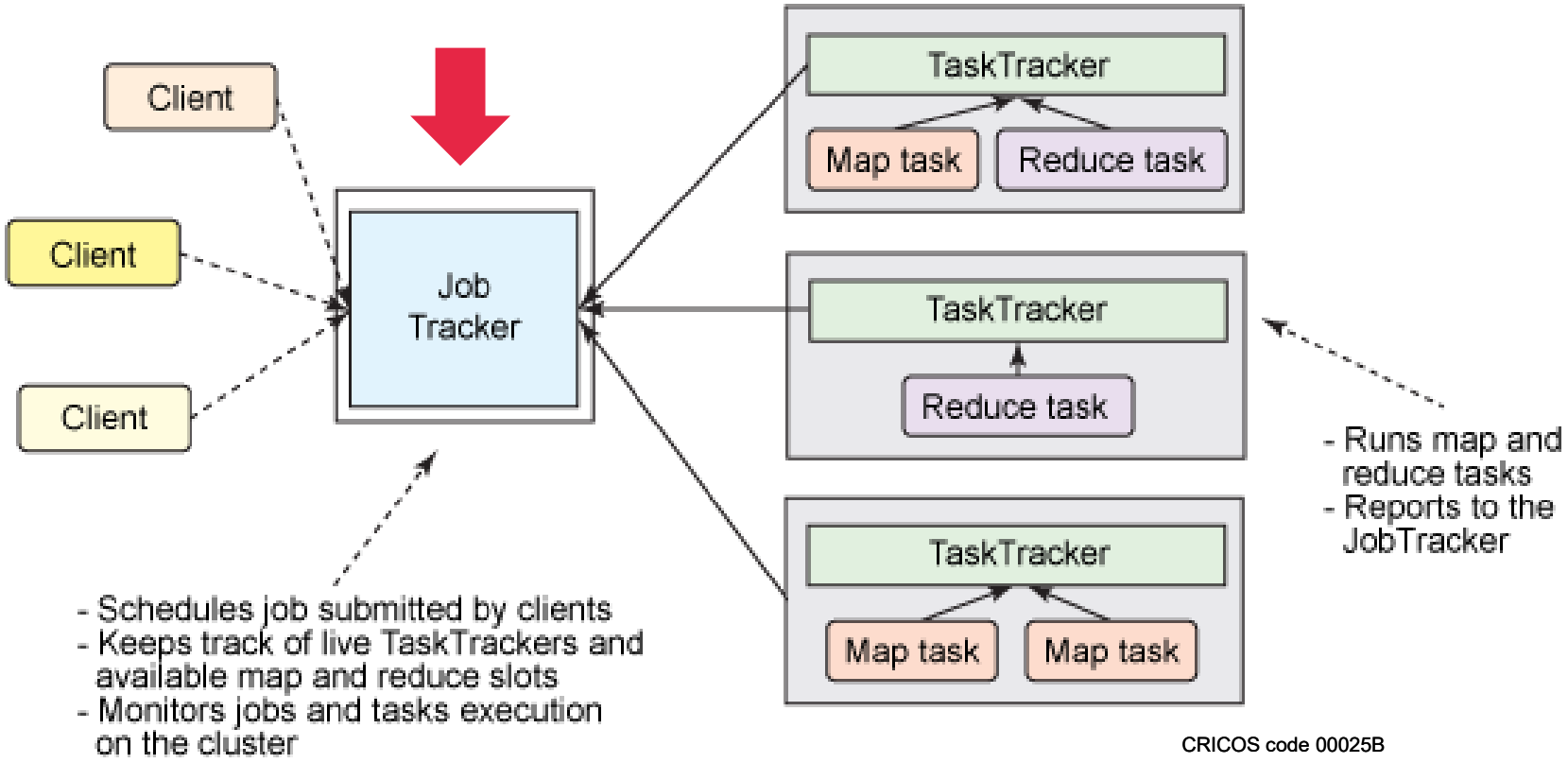
MapReduce Architecture: Components

- Client:
  - MapReduce program by users will be submitted to JobTracker via Client
  - Users can display job running status through interfaces in Client



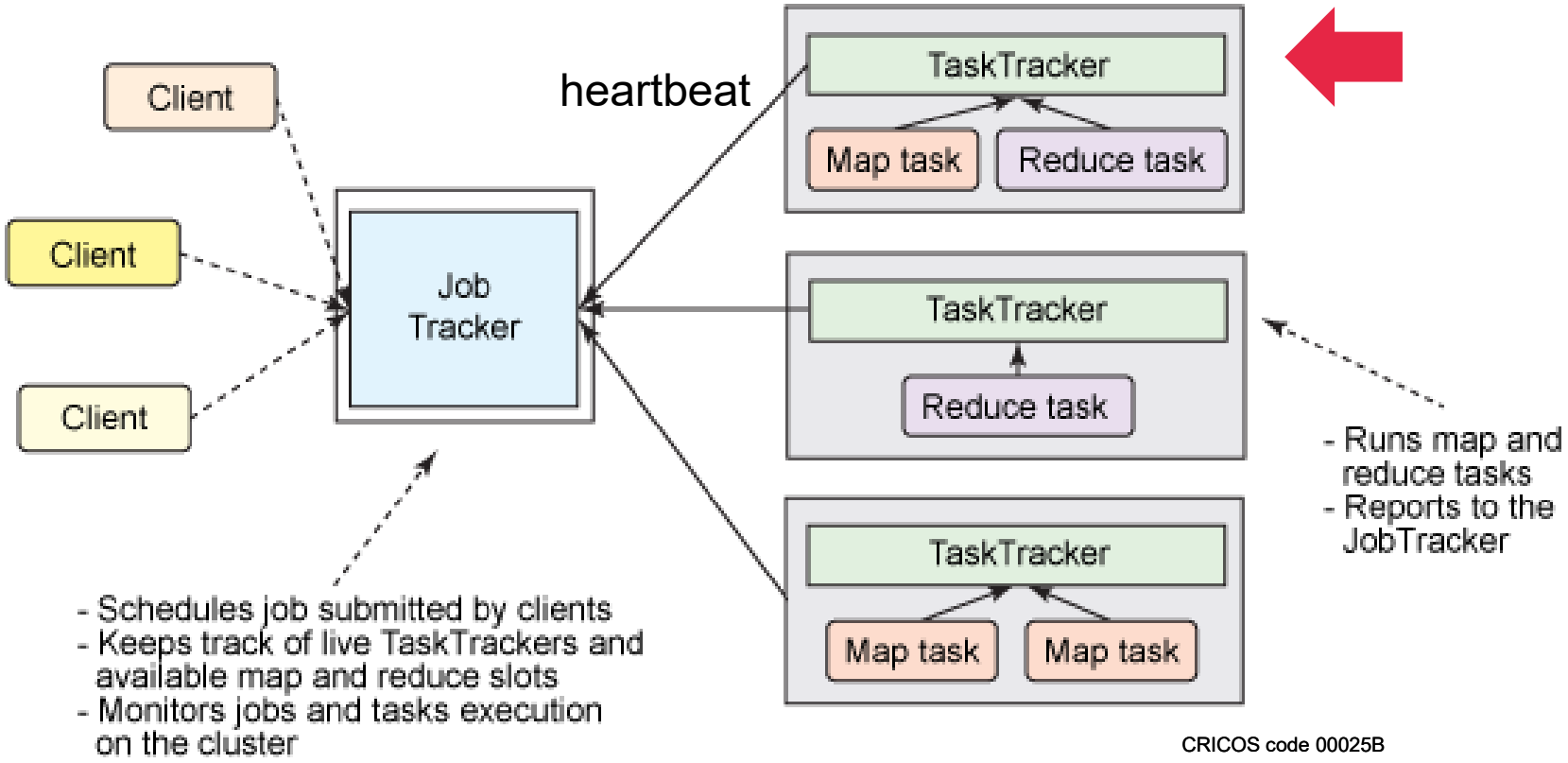
MapReduce Architecture: Components

- JobTracker:
  - Monitor resources and coordinate jobs
  - Monitor health of all the TaskTrackers (transfer jobs to other nodes once failure found)



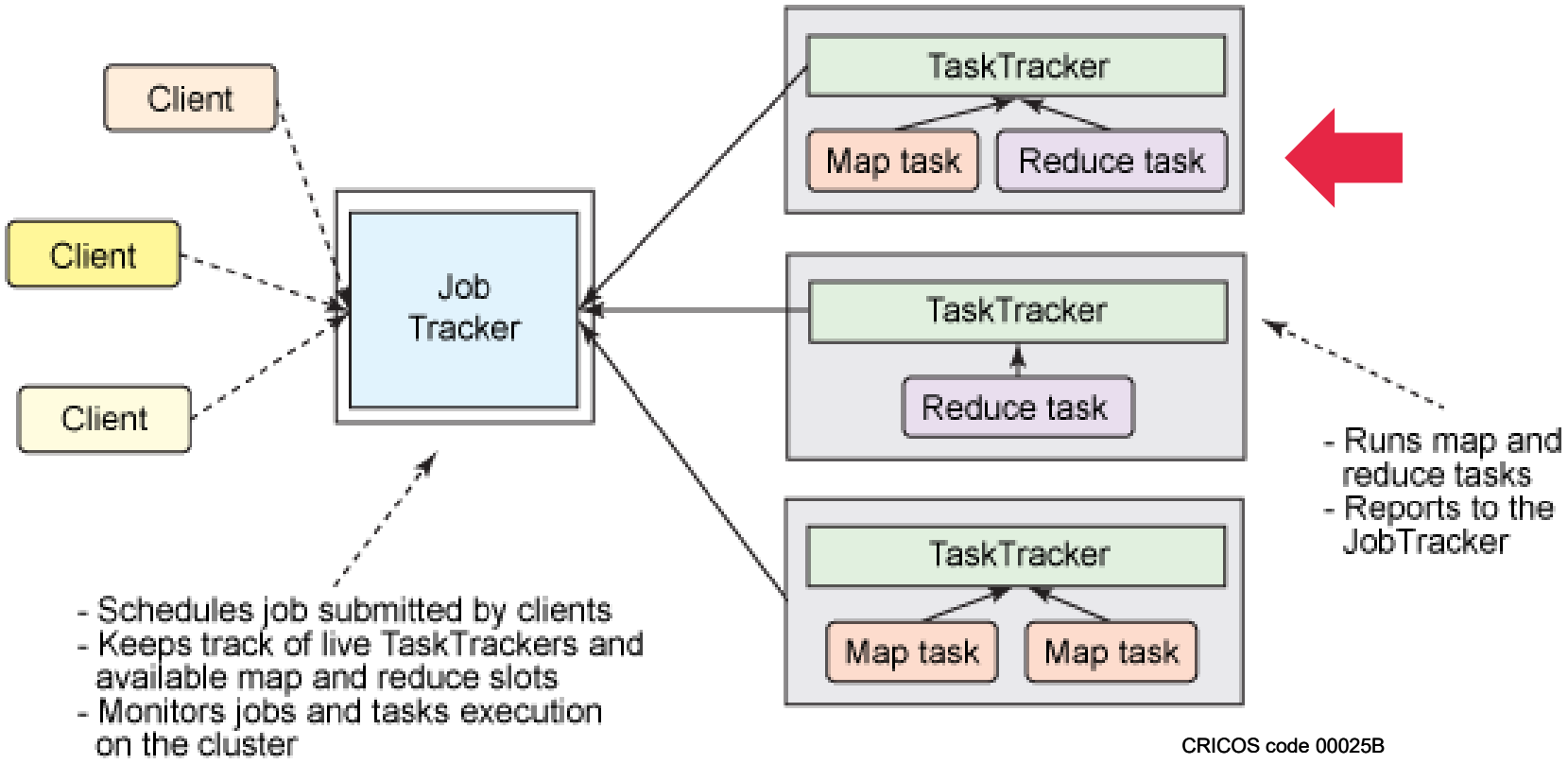
MapReduce Architecture: Components

- TaskTracker:
  - Periodically heartbeat with resource information job execution status to JobTracker
  - Receive and execute commands from JobTracker (start new tasks or kill existing tasks)

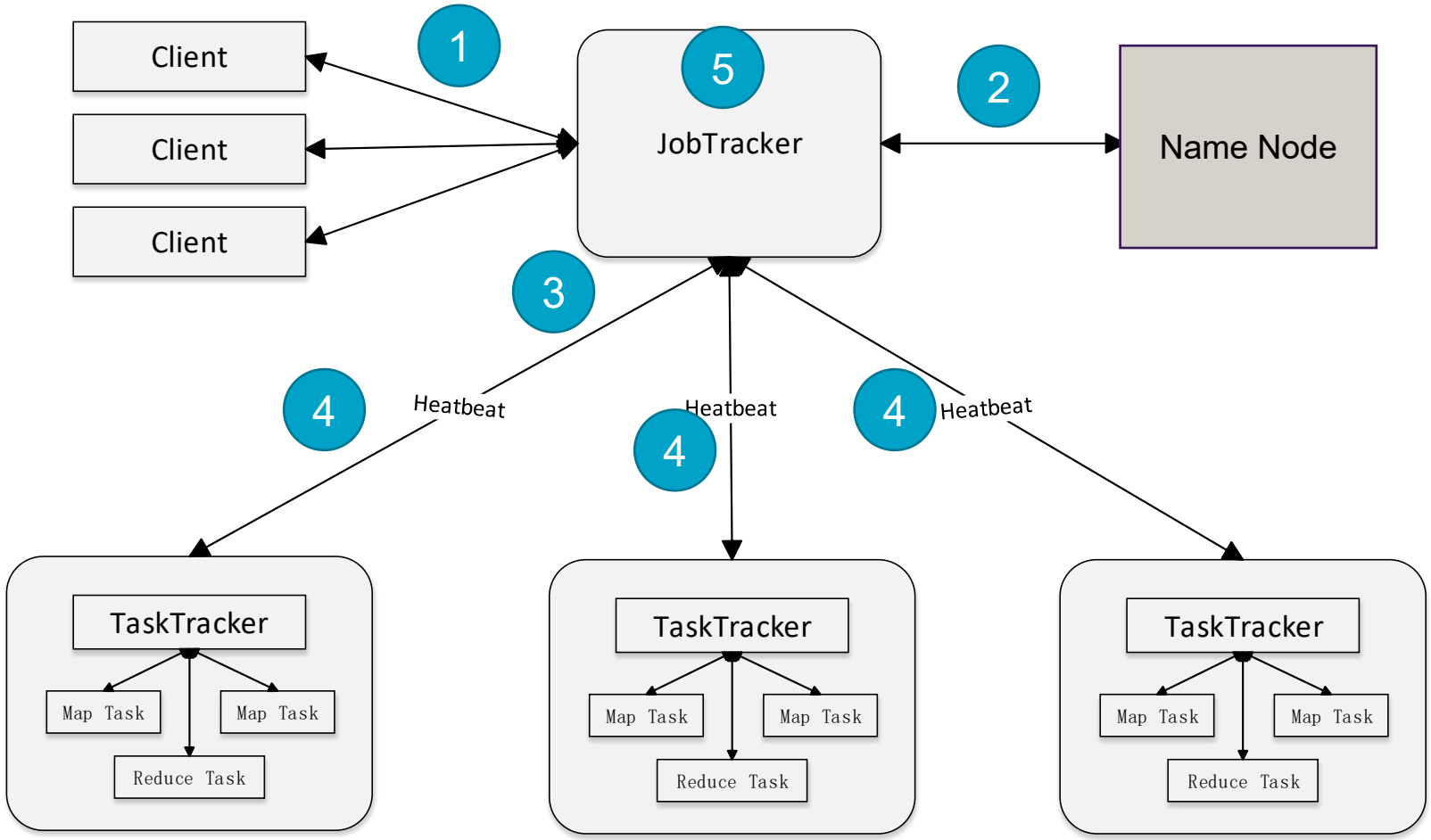


MapReduce Architecture: Components

- Task:
  - Map Task and Reduce Task
  - Initiated by TaskTracker

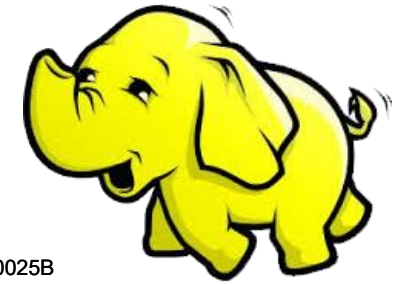


MapReduce Architecture Ver 1: Workflow



MapReduce Paradigm

- Map-Reduce is the data processing component of Hadoop.
- Map-Reduce programs transform lists of input data elements into lists of output data elements.
- A Map-Reduce program will do map and reduce tasks asynchronously
- MapReduce Job (an execution of a Mapper and Reducer across a data set) consists of
  - the input data (submitted by users),
  - the MapReduce Program (program logic written by users),
  - and configuration info.



Inputs and Outputs

- The MapReduce framework operates exclusively on **<key, value>** pairs;
- The framework views the input to the job as a set of **<key, value>** pairs and produces a set of **<key, value>** pairs as the output of the job.
- Input and Output types of a MapReduce job:

Function	Input	Output	Note
Map	<k_1, v_1> e.g. <lineNum, "a b c b c a a c c">	List(<k_2, v_2>) e.g. (<"a", 1>, <"b", 1>, <"c", 1>, <"b", 1>, <"c",1>, <"a",1>, <"a",1>, <"c",1>, <"c",1>)	1. Convert splits of data into a list of <key, value> pairs. 2. Each input <k_1, v_1> will output a list of key/value pairs as intermediate results
Reduce	<k_2, List(v_2)> e.g. <"a", <1, 1, 1> >	<k_3, v_3> e.g. <"a", 3>, <"b", 2>, <"c", 4>	The value of <k_2, List(v_2)> in the intermediate result, List(v_2), represents the values of the same key k_2.

Inputs and Outputs

Map takes one pair of data with a type in one data domain, and returns a list of pairs in a different domain: Map(k_1,v_1) → list(k_2,v_2)

- k_1 stands for line number while v_1 stands for contents
- E.g. (1, “Hello, world. Welcome to MapReduce world!”) → (”Hello”, 1), (“world”, 1), (“Welcome”, 1), (“to”, 1), (“MapReduce”,1), (“world”,1)
- The Map function is applied in parallel to every pair (keyed by k1) in the input dataset
  - Each line will be applied with Map function
  - (2, “Hello, MapReduce. Welcome to world!”) → (”Hello”, 1), (“world”, 1) …
  - (3, “Hello, Spark. Spark is better than MapReduce.”) → (”Hello”, 1), (“Spark”, 1) …
  - (…)
- a list of pairs (keyed by k_2) will be generated. k_2 here is a word not a line number.
- After that, the MapReduce framework collects all pairs with the same key (k_2) from all lists and groups them together, creating one group for each key.
  - (“Hello”, <1, 1, 1>), (“world”, <1, 1, 1, 1>), etc.

Inputs and Outputs

The Reduce function is then applied in parallel to each group, which in turn produces a collection of values in the same domain: Reduce(k_2, list (v_2)) → (k_3, v_3)

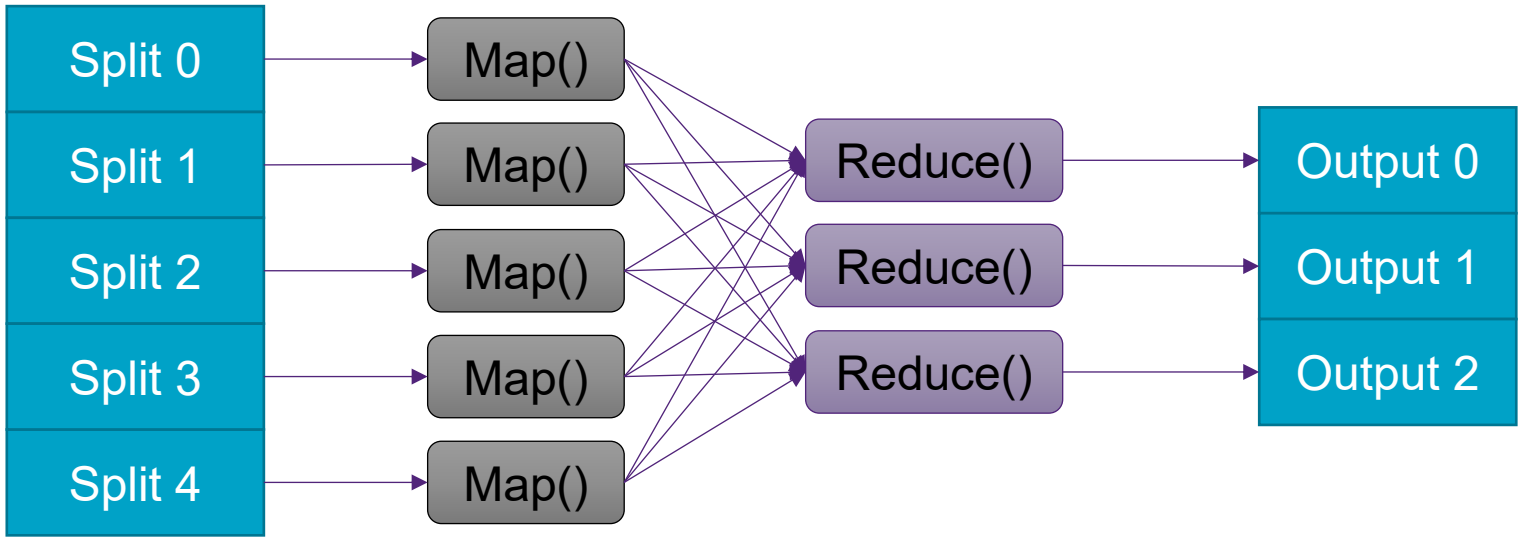
- E.g. (“Hello”, <1, 1, 1>) → (”Hello”, 3), (“world”, <1, 1, 1, 1>) → (”world”, 4)
- Each Reduce call typically produces either one value v_3 or an empty return.

```
function map(String name, String document):
  // name: document name
  // document: document contents
  for each word w in document:
    emit (w, 1)

function reduce(String word, Iterator partialCounts):
  // word: a word
  // partialCounts: a list of aggregated partial counts
  sum = 0
  for each pc in partialCounts:
    sum += pc
  emit (word, sum)
```

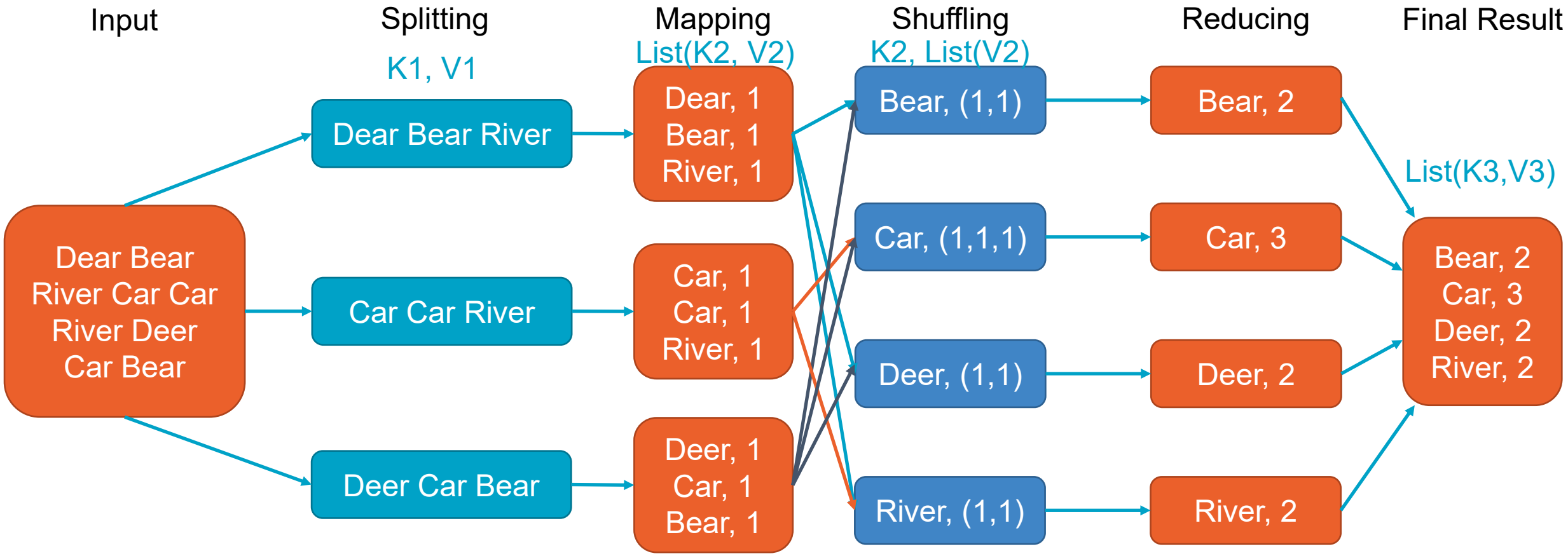
Divide and Conquer

- No communications between Map tasks
- No communications between Reduce tasks
- Need shuffle process to transfer data to reducer



Example I – Word Count

- Given a file consists of the following words: "**Dear, Bear, River, Car, Car, River, Deer, Car and Bear**" and The overall MapReduce process will look like:

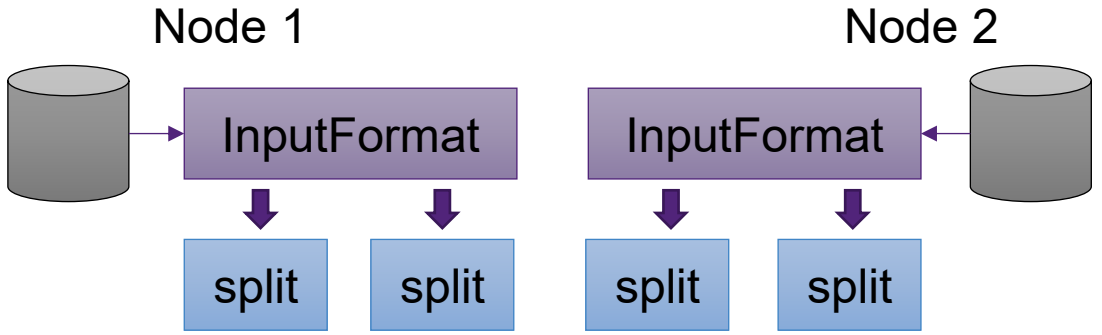


Example I – word count

- The input is split into three different data-sets that are distributed among the map nodes.
- The words are tokenized and a value of 1 is assigned to each word which is hardcoded. This is done assuming that each word will appear only once.
- After this list of key-value pairs is created where the key is the word and value is the number one. For e.g. the first data-set has three key-value pairs: Bear, 1; River, 1; Dear, 1.
- After the mapping process is completed the shuffling and sorting tasks are executed so that all the tuples that have the same key are combined together and sent to the reducer.
- On completion of the sorting and shuffling tasks, each reducer will have a unique key and the corresponding list of values. For example, Car, [1,1,1]; Deer, [1,1]… etc.
- The reducer will now count the values present in the list of each unique key and generate the key, value pair for that key, where the value will now correspond to the count.
- The output is written in the key, value format to the output file.

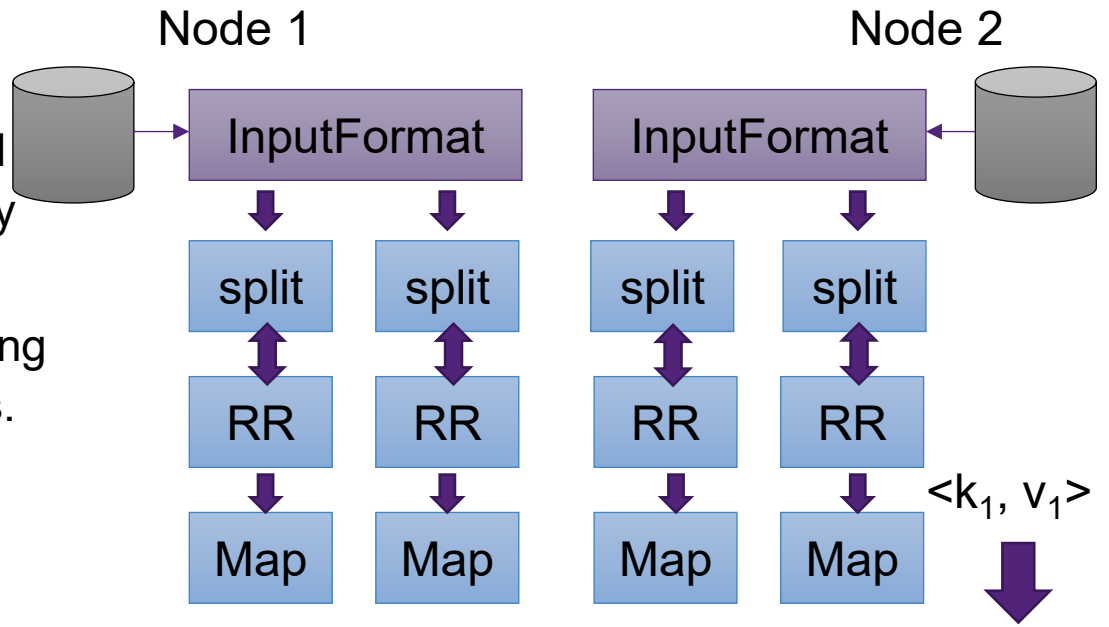
DataFlow

- Data are stored in HDFS across two nodes as input files.
- InputFormat defines how these input files are split and read. It selects the files or other objects that are used for input. InputFormat creates InputSplit.
- InputSplit logically represents the data which will be processed by an individual **Mapper**.
- One map task is created for each split; thus the number of map tasks will be equal to the number of InputSplits.
- The split is divided into records and each record will be processed by the **mapper**.



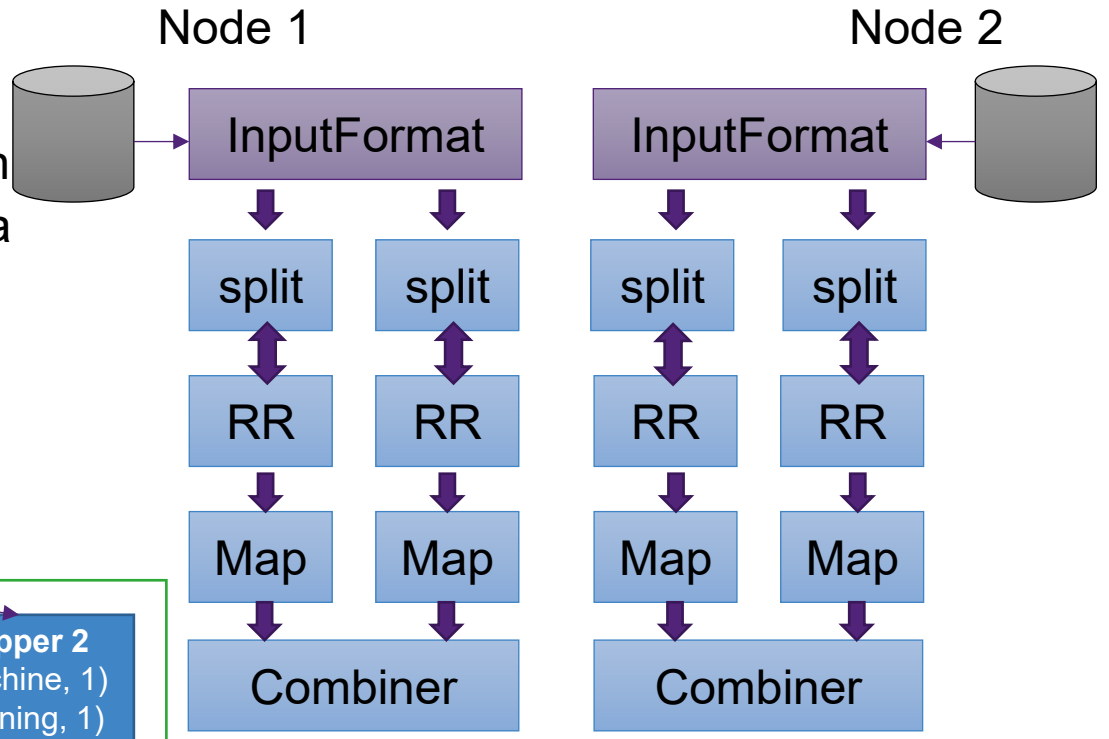
DataFlow

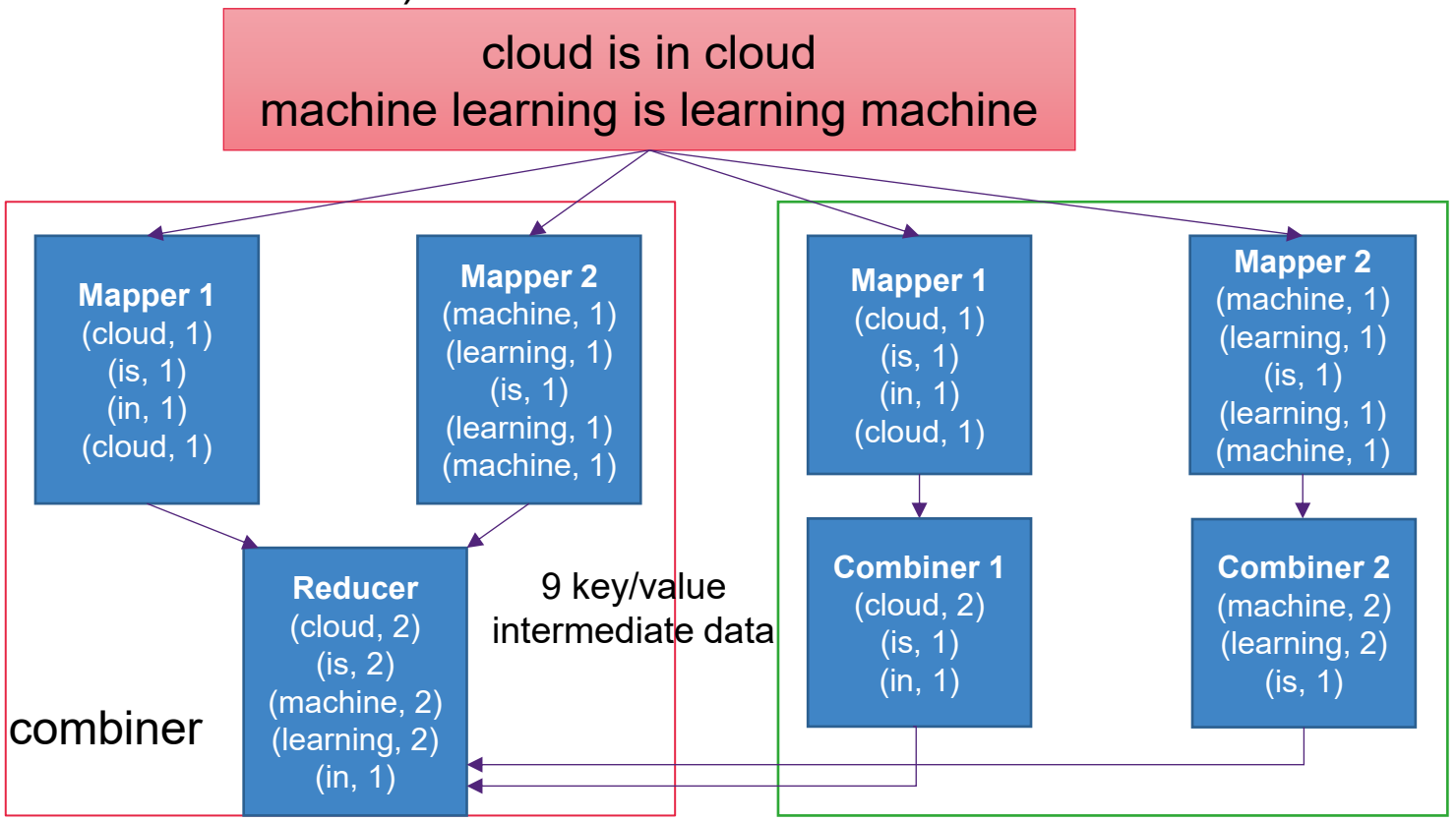
- **RecordReader (RR)** communicates with the **InputSplit** and converts the data into key-value pairs suitable for reading by the mapper.
- RecordReader (RR) will actually read data in HDFS according to the split information and pass key-value pairs to mappers.
- **Mapper** processes each input record (from RecordReader) and generates new key-value pair, which is completely different from the input pair.
- The output of Mapper is also known as intermediate output which is written to the local disk.
- The output of the Mapper is not stored on HDFS as this is temporary data and writing on HDFS will create unnecessary copies (due to replication setting, e.g. replicas=3).
- Mappers output is passed to the combiner for further process



DataFlow

- **Combiner** (aka mini-reducer) performs local aggregation on the mappers' output, which helps to minimize the data transfer between **mapper** and **reducer** (we will see reducer below).





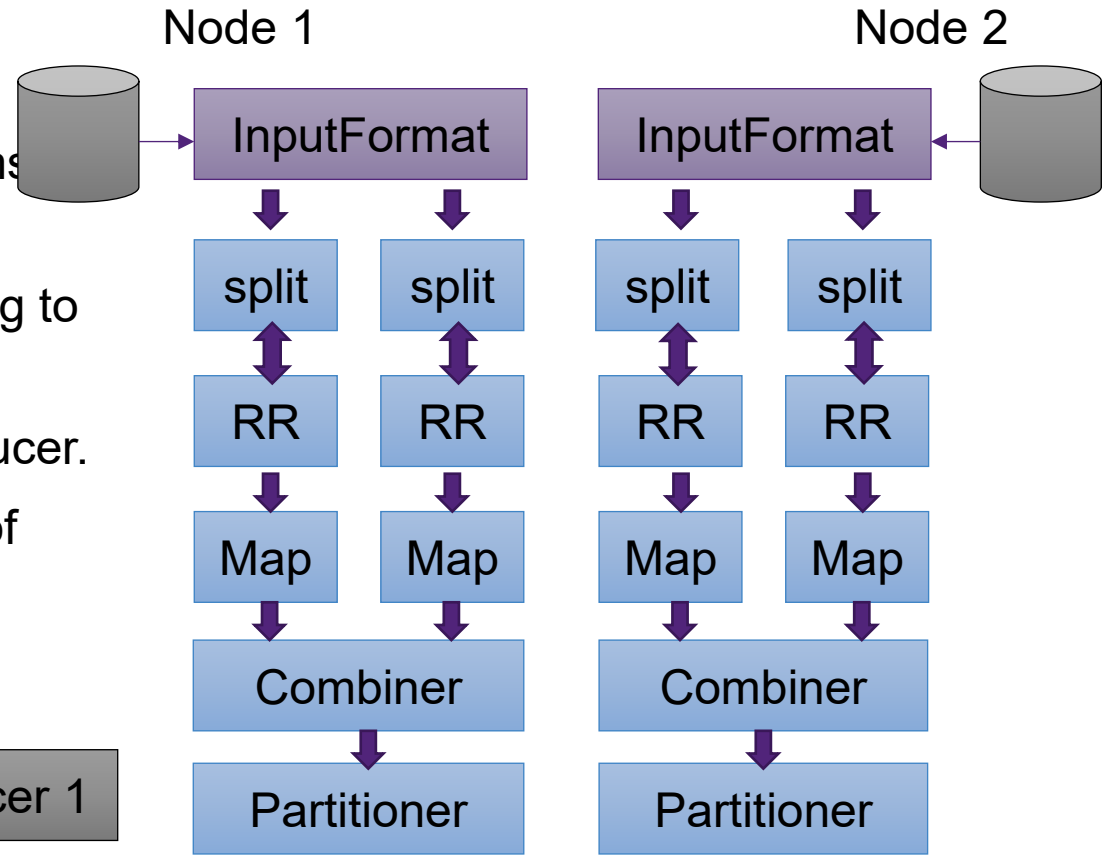
With combiners

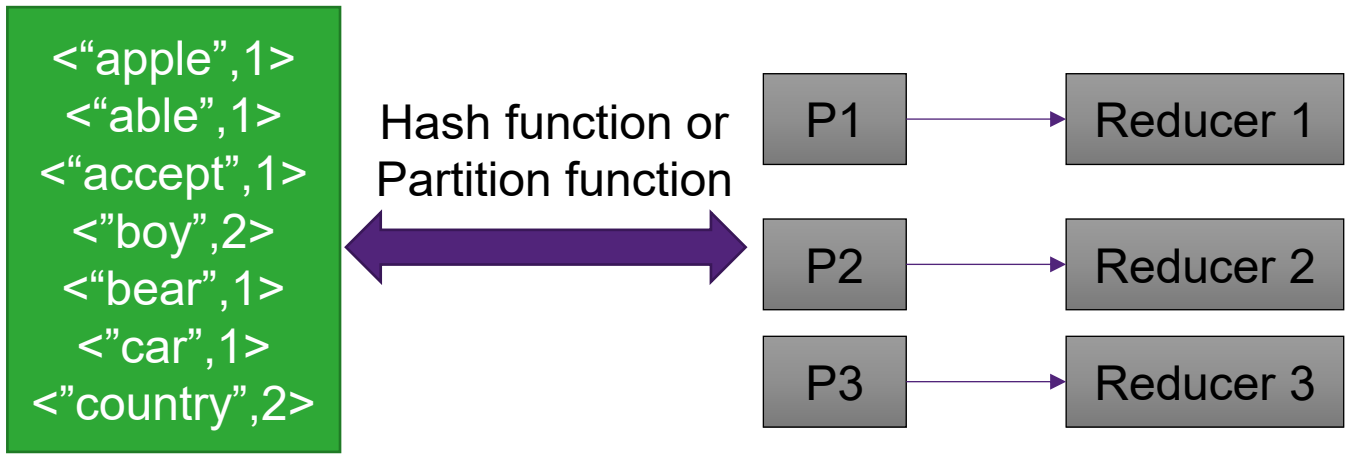
6 key/value
intermediate data

Improved Overall Performance

DataFlow

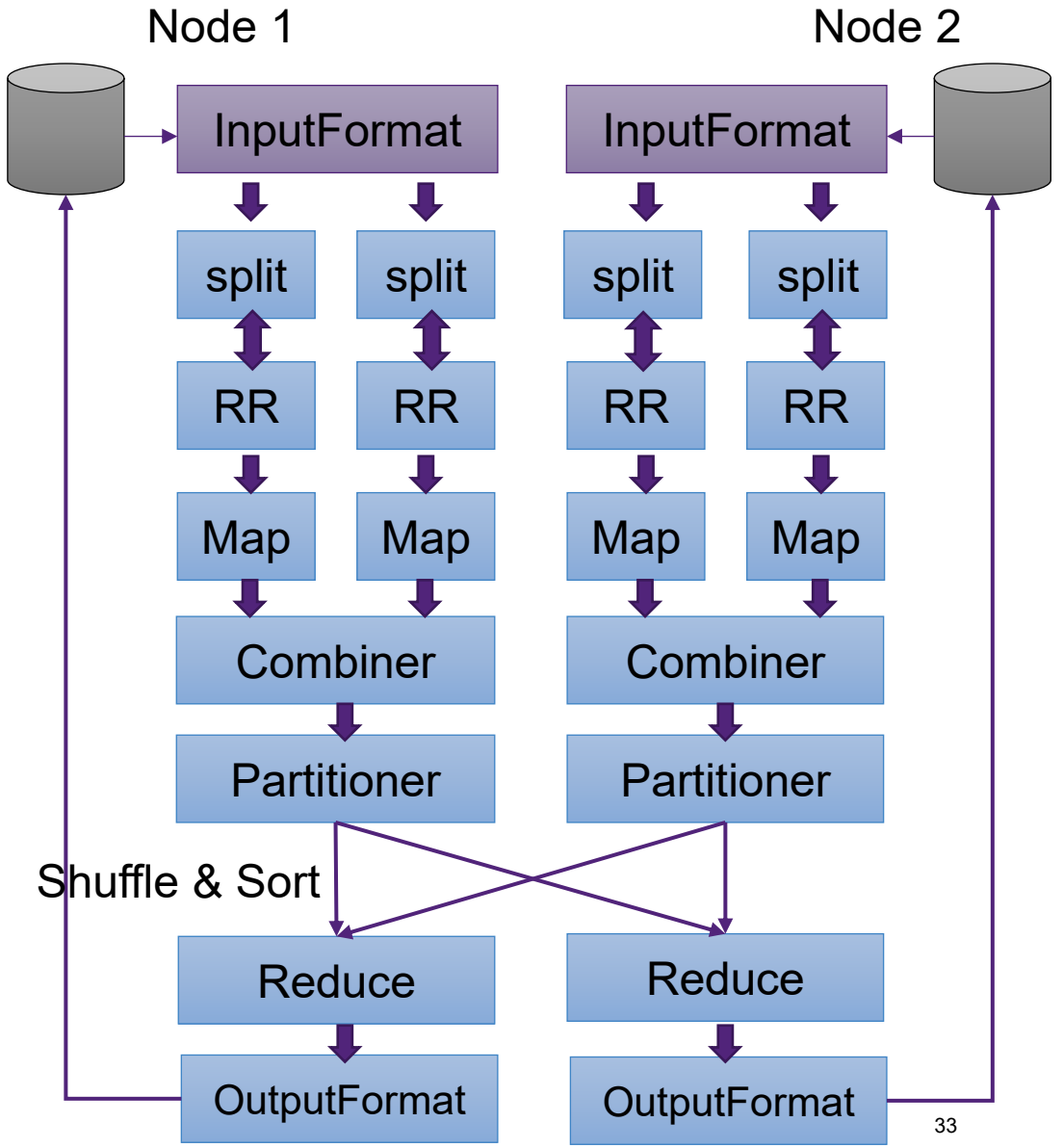
- **Partitioner** takes the output from combiners and performs partitioning.
- Keys of the intermediate data will be partitioned according to hash function.
- All keys within the same partition will go to the same reducer.
- The total number of Partitioner depends on the number of reducers.





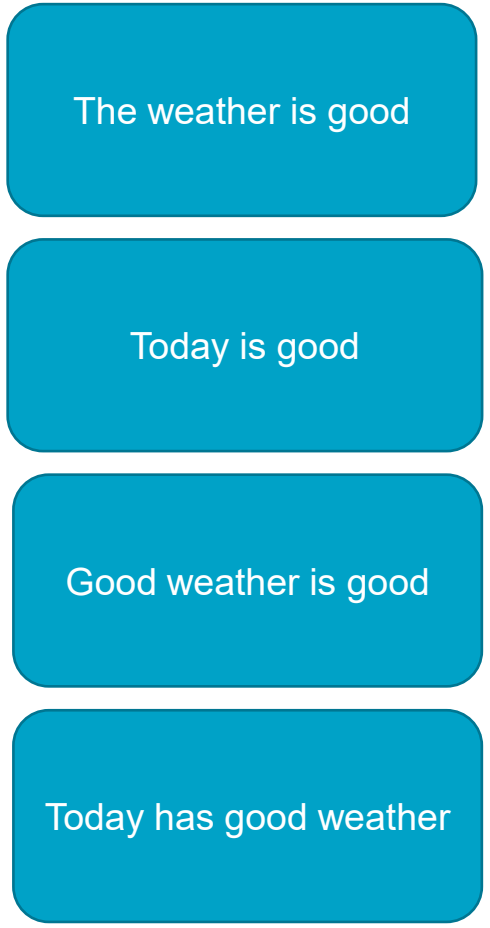
DataFlow

- **Shuffling** is the process by which the intermediate output from **mappers** is transferred to the **reducer**.
- Reducer has three primary phases: **shuffle** & **sort** and **reduce**.
- OutputFormat determines the way these output key-value pairs are written in output files.
- The final results will be written in HDFS.
  - Note that the intermediate data from mappers are not immediately written into local storage, but into memory.
  - There is a mechanism, call **spill**, that periodically write intermediate data in memory into disk.

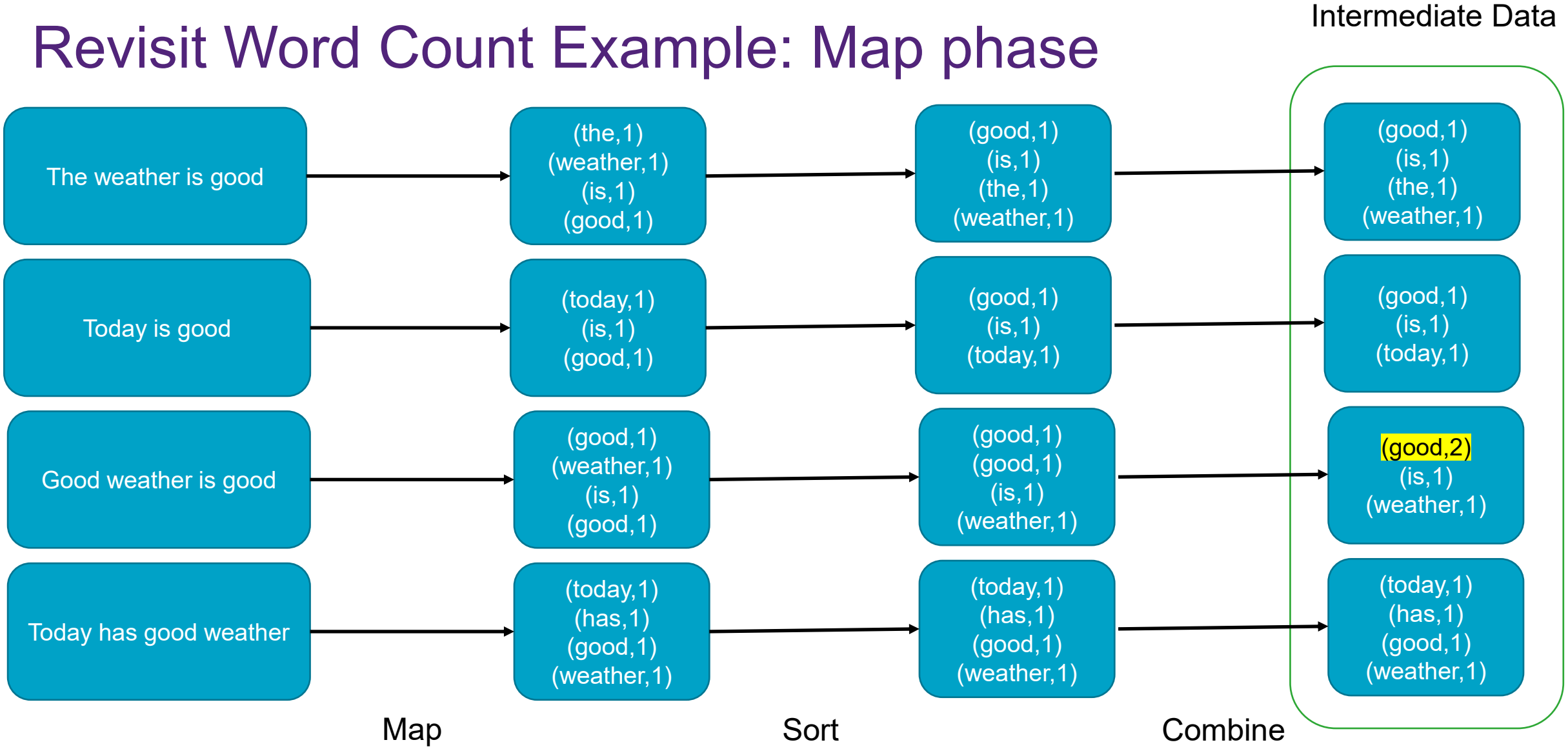


Revisit Word Count Example:

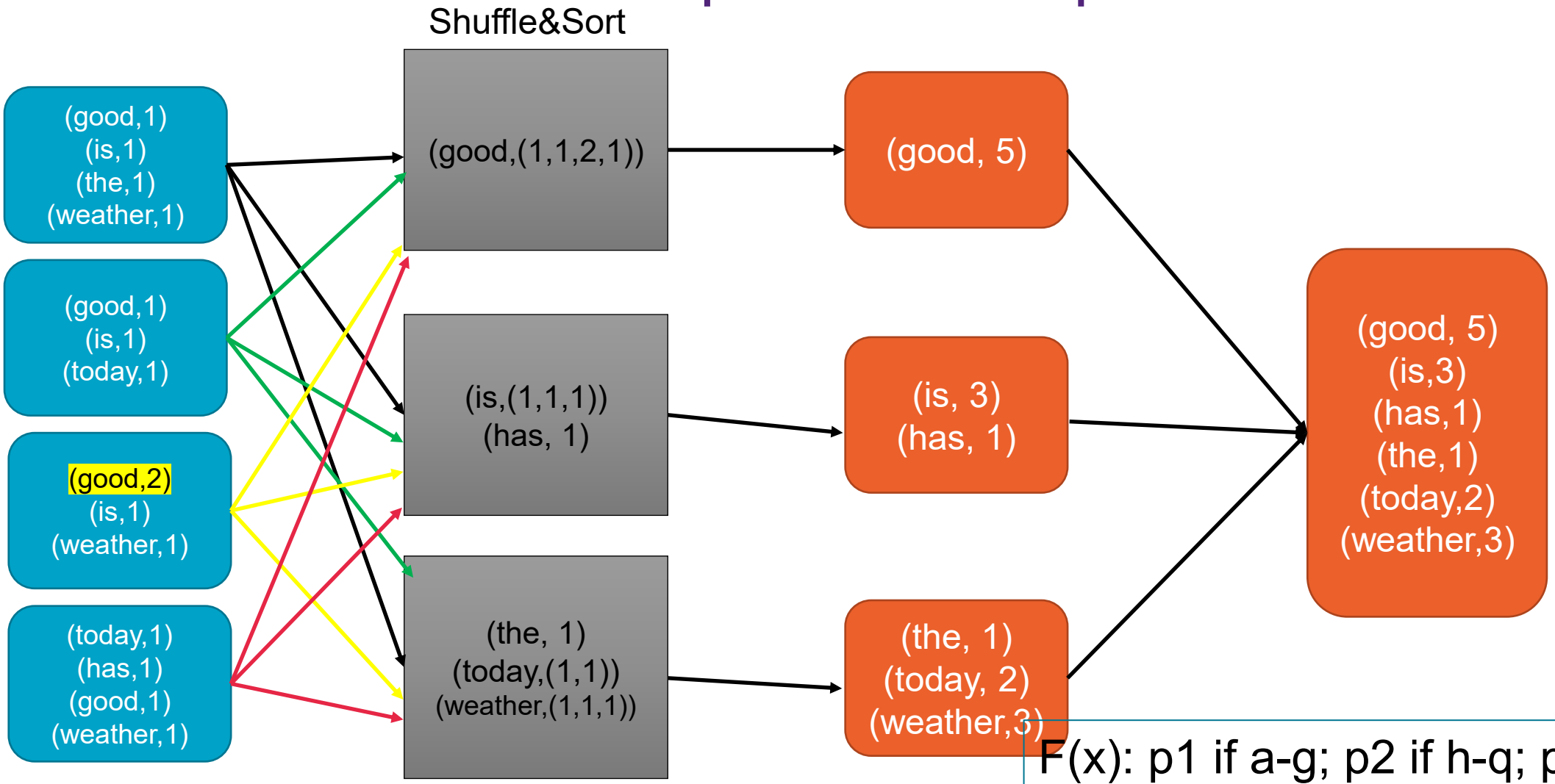
- Assuming that there are four text files on different nodes, you want to count the word frequency using MapReduce model.
- Text 1 (node 1): the weather is good
- Text 2 (node 2): today is good
- Text 3 (node 3): good weather is good
- Text 4 (node 4): today has good weather



Revisit Word Count Example: Map phase



Revisit Word Count Example: Reduce phase



F(x): p1 if a-g; p2 if h-q; p3 if r-z

Hash function: need to distribute records evenly over the partitions

Example II: Find the Highest Temperature

Problem: Find the maximum monthly temperature for **each year** from weather reports

Input: A set of records with format as:

<Year/Month, Average Temperature of that month>

- **(201707,37.8), (201706,32.2)**
- **(201508, 32.2), (201607,37.8)**
- **(201708, 26.7), (201606,26.7)**

Question: write down the Map and Reduce function to solve this problem

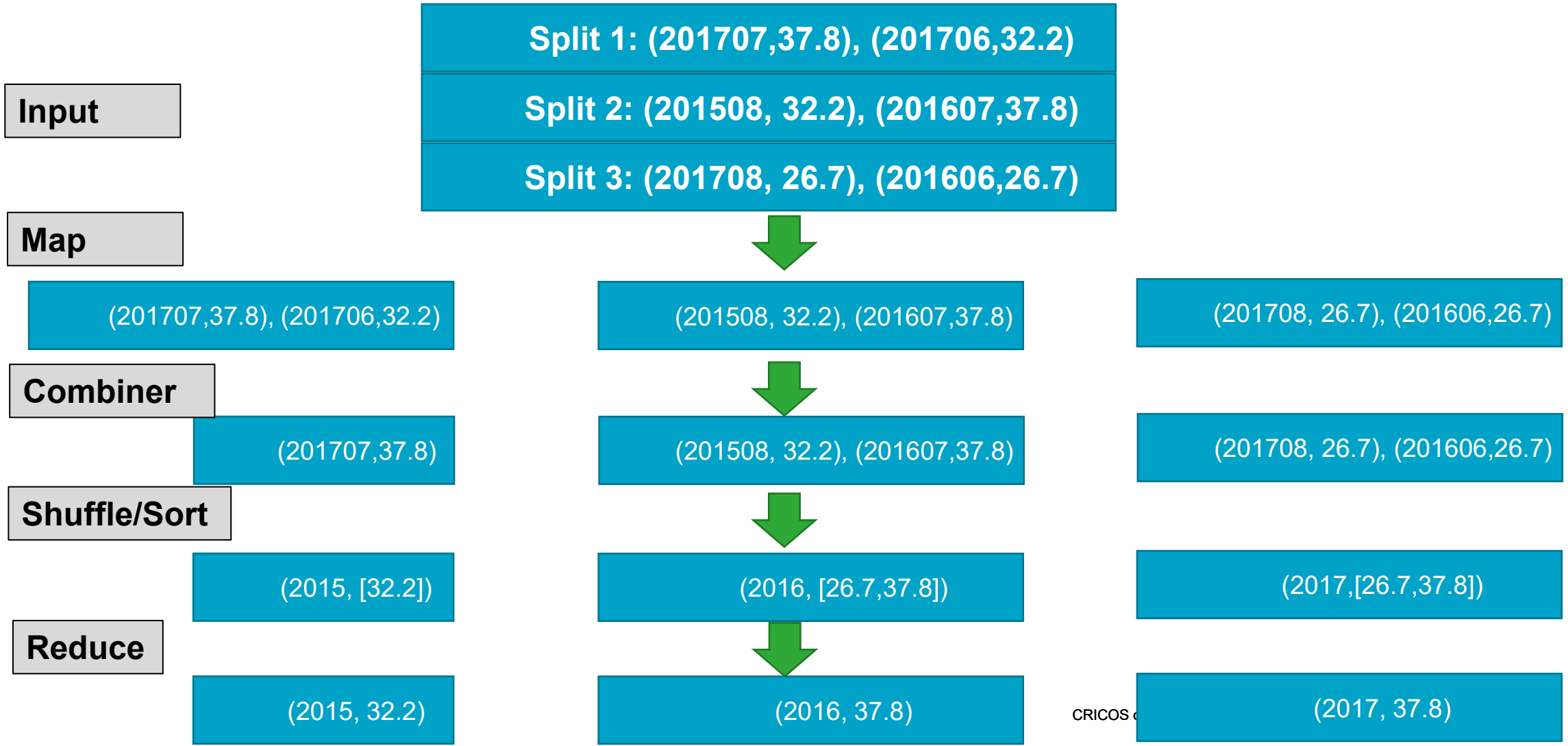
- Assume we split the input by line:

Split 1: (201707,37.8), (201706,32.2)

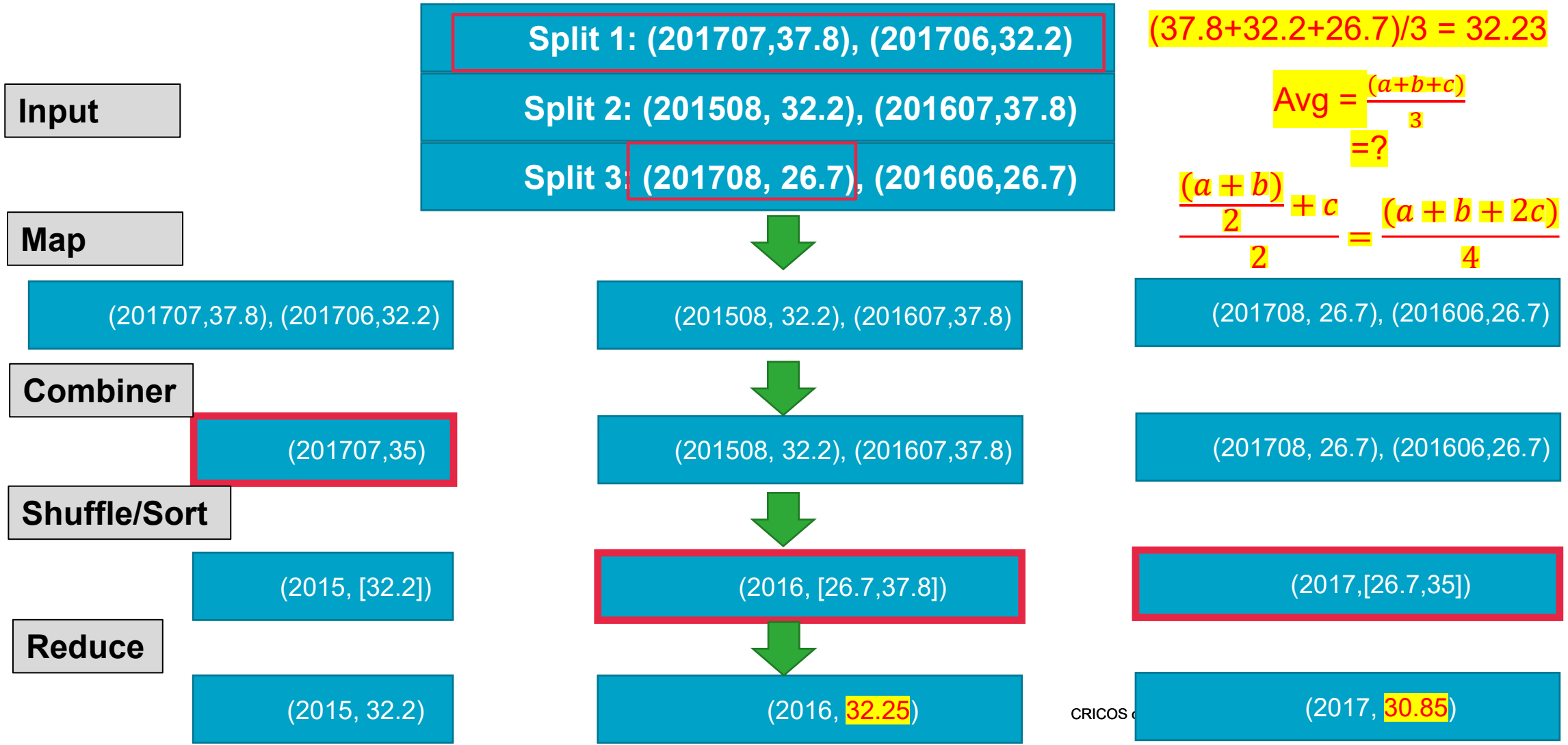
Split 2: (201508, 32.2), (201607,37.8)

Split 3: (201708, 26.7), (201606,26.7)

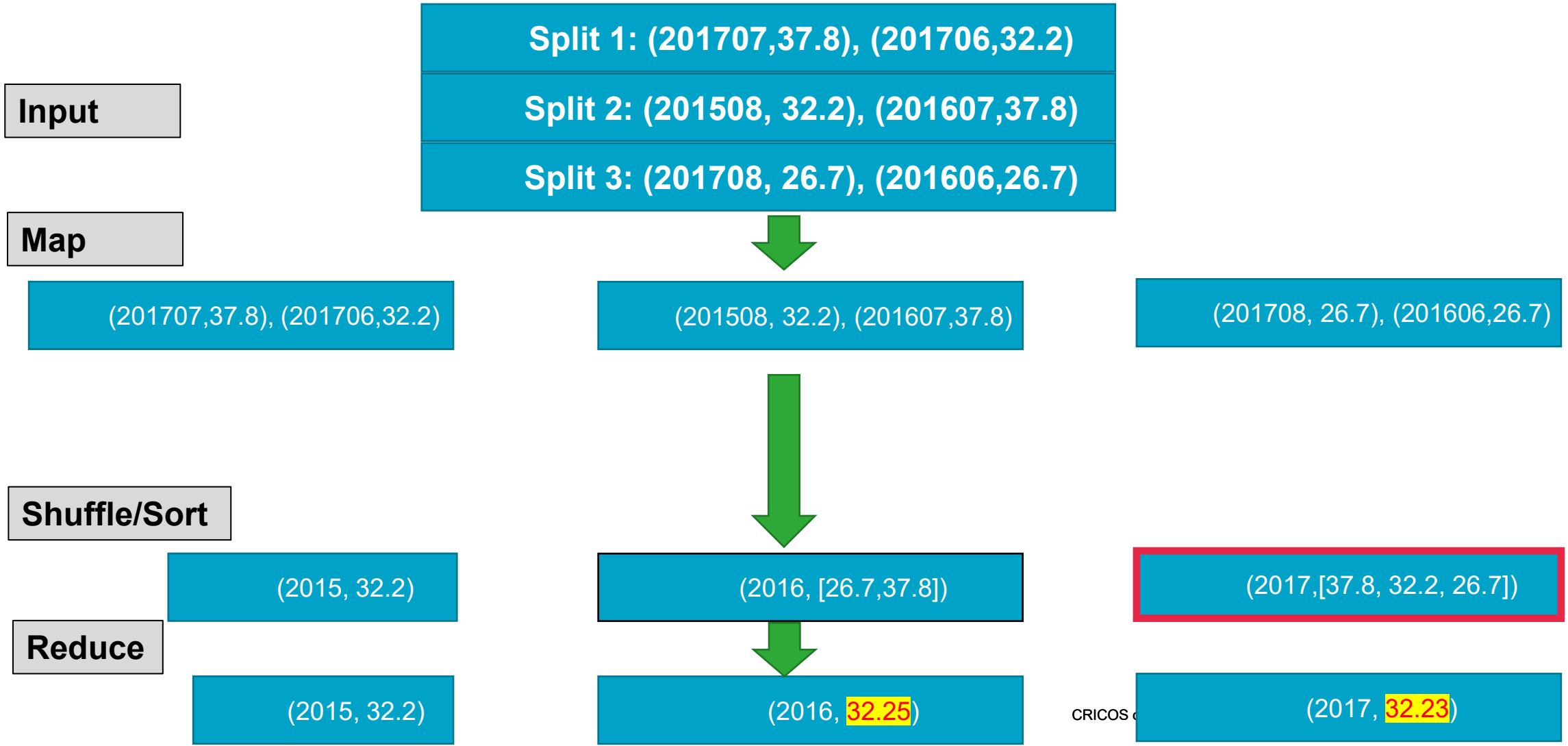
Example II: Find the Highest Temperature



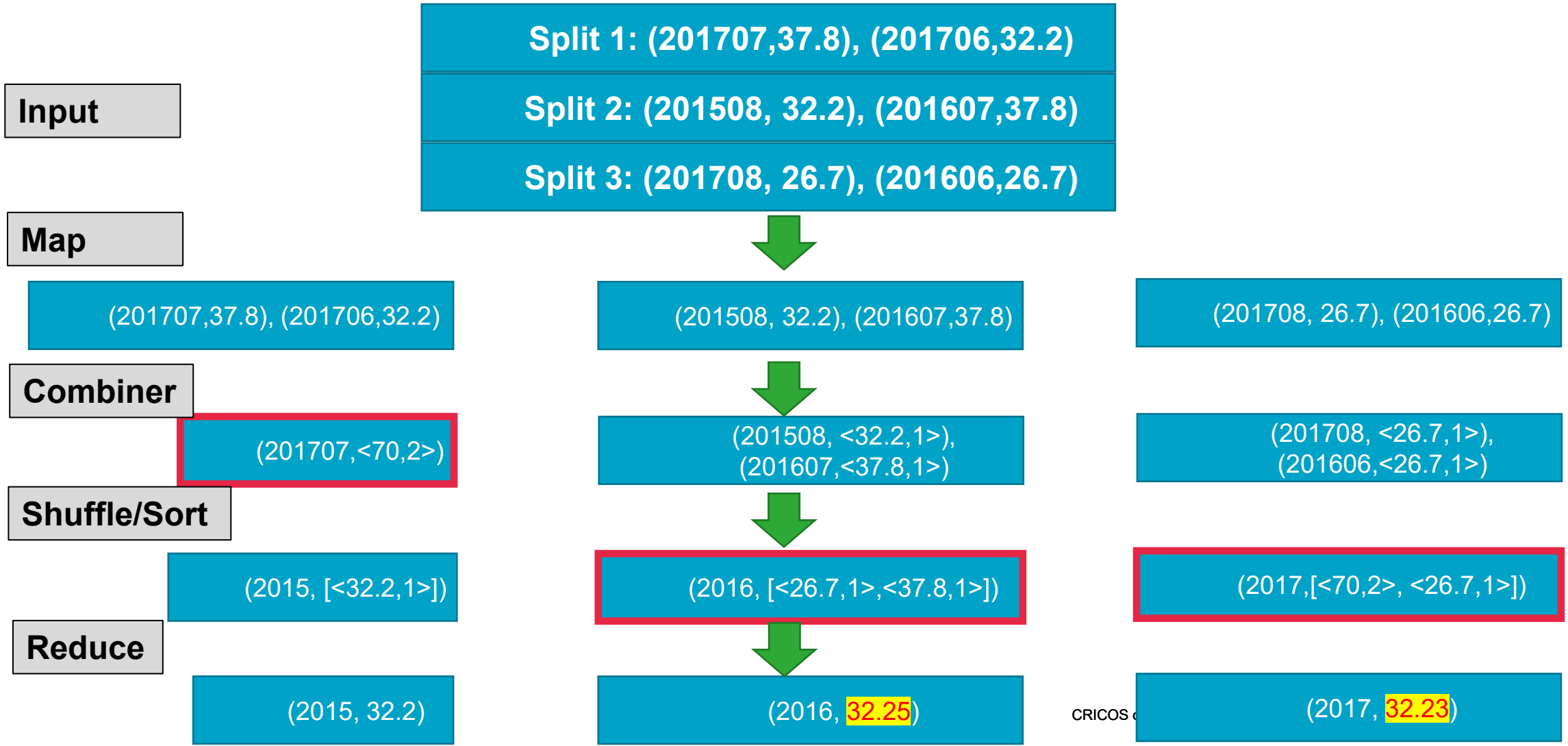
Example III: Find the Average Temperature (1)



Example III: Find the Average Temperature (2)



Example III: Find the Average Temperature (3)



Example III: Find the Average Temperature (Pseudo code)

- **Map(key, value)**{

```
// key: line number
// value: tuples in a line
for each tuple t in value:
        Emit(t->year, t->temperature);}
```

- **Combiner(key, list of values)**{

```
// key: year
// list of values: a list of monthly temperature
int total_temp = 0;
for each v in values:
      total_temp= total_temp+v;
Emit(key,<total_temp,size_of(values)>);}
```

- **Reduce (key, list of values)**{

```
// key: year
// list of values: a list of <temperature sums, counts> tuples
int total_temp = 0;
int total_count=0;
for each v in values:
      total_temp= total_temp+v->sum;
      total_count=total_count+v->count;
Emit(key,total_temp/total_count);}
```

Example IV: Find Word Length Distribution

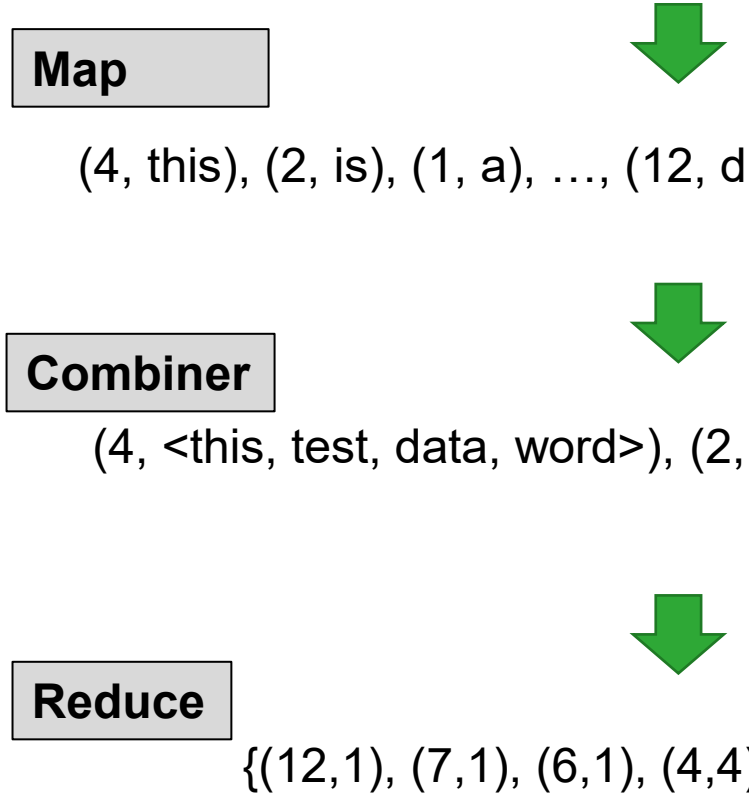
- Given a set of documents, use Map-Reduce model to find the length distribution of all words contained in the documents.
- Example:
  - Input Doc: {This is a test data for the word length distribution problem}
  - Output: {(12,1), (7,1), (6,1), (4,4), (3,2), (2,1), (1,1)}

Word	Length	Word	Length
this	4	the	3
is	2	word	4
a	1	length	6
test	4	distribution	12
data	4	problem	7
for	3		

Example IV: Find Word Length Distribution

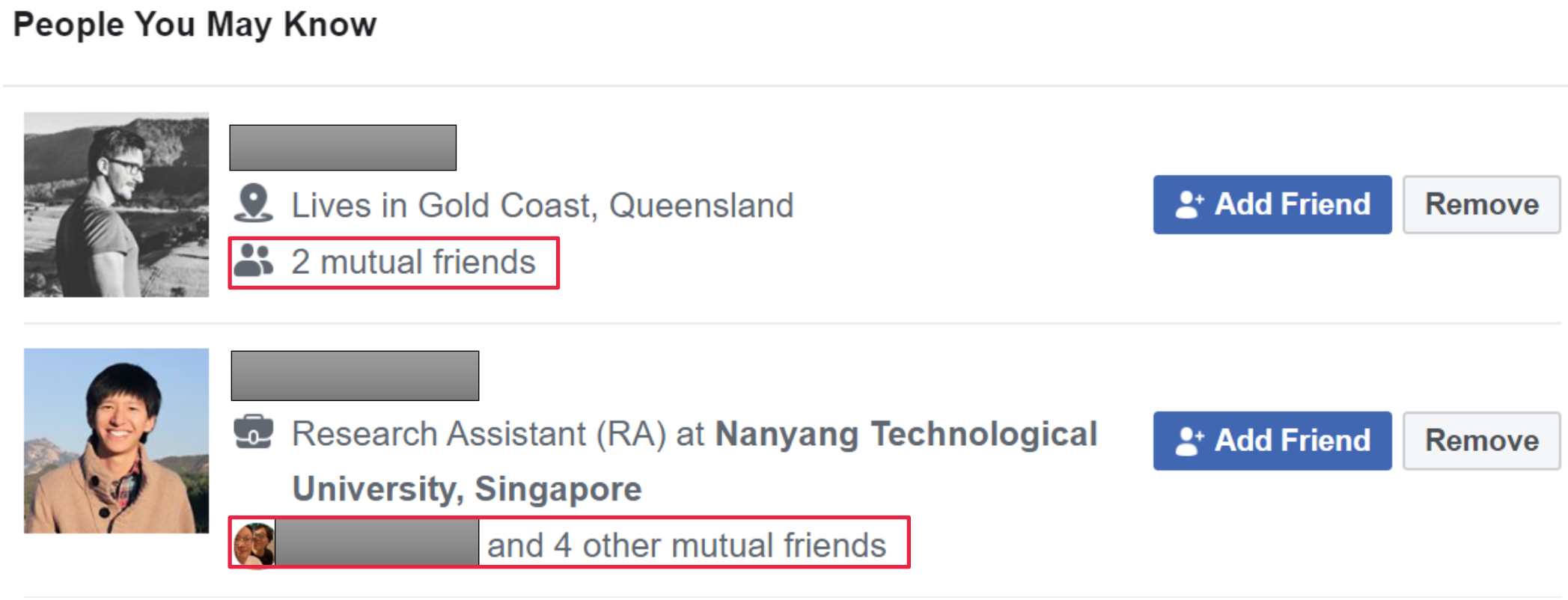
```
Map(key, value){
        // key: document name
        // value: words in a document
        for each word w in value:
                Emit(length(w), w);}
Reduce(key, list of values){
        // key: length of a word
        // list of values: a list of words
with the same length
         Emit(key, size_of(values));}
```

{This is a test data for the word length distribution problem}



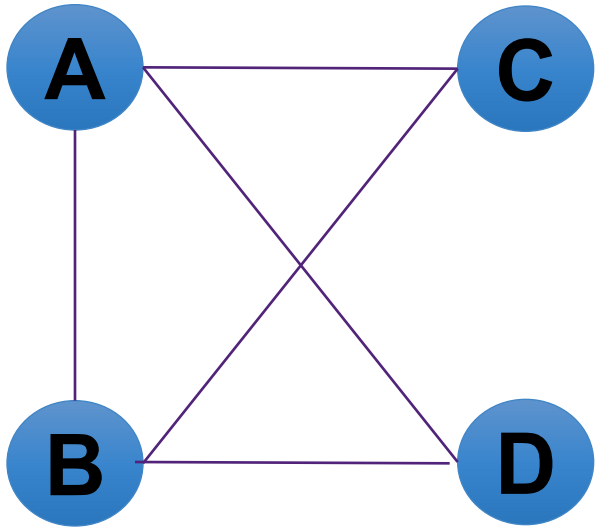
Example V: Find Mutual Friends

Given a group of people on online social media (e.g., Facebook), each has a list of friends, use Map-Reduce to find *mutual friends* between two persons



Example V: Find Mutual Friends

Simple example:



Input:

A -> B,C,D

B-> A,C,D

C-> A,B

D->A,B

MapReduce

Output:

(A ,B) -> C,D

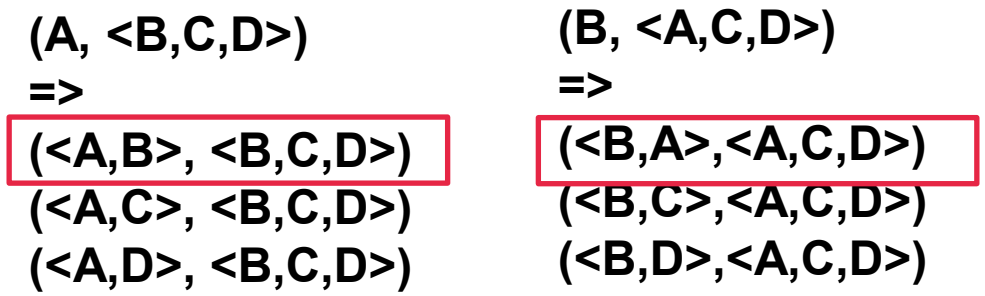
(A,C) -> B

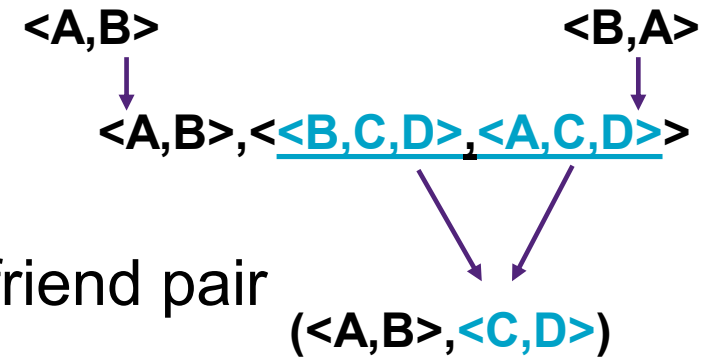
(A,D) -> ..

….

Example V: Find Mutual Friends

```
Map(key, value){
        // key: person_id
        // value: the list of friends of the person
        for each friend f_id in value:
                Emit(<person_id, f_id>, value);}
Reduce(key, list of values){
        // key: <friend pair>
        // list of values: a set of friend lists related with the friend pair
        for  v1, v2 in values:
                mutual_friends = v1 intersects v2;
    Emit(key, mutual_friends);}
```





Example V: Find Mutual Friends

Input:

A -> B,C,D
B-> A,C,D
C-> A,B
D->A,B

Map:

(A,B) -> B,C,D
(A,C) -> B,C,D
(A,D) -> B,C,D
(A,B) -> A,C,D
(B,C) -> A,C,D
(B,D) -> A,C,D
(C,A) -> A,B
(C,B) -> A,B
(D,A) -> A,B
(D,B) -> A,B

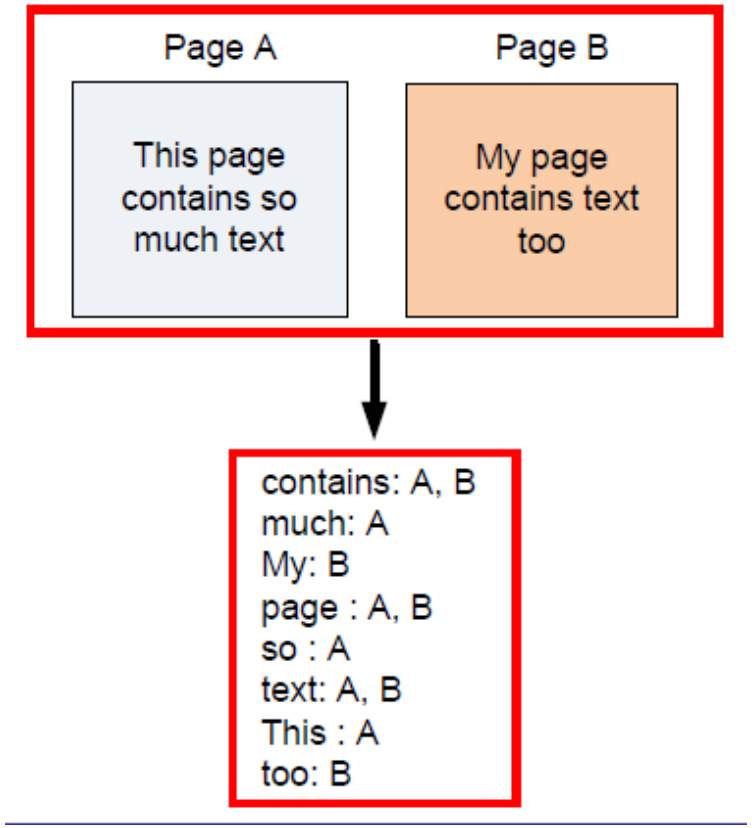
Reduce:

(A,B) -> C,D
(A,C) -> B
(A,D) -> B
(B,C) -> A
(B,D) -> A

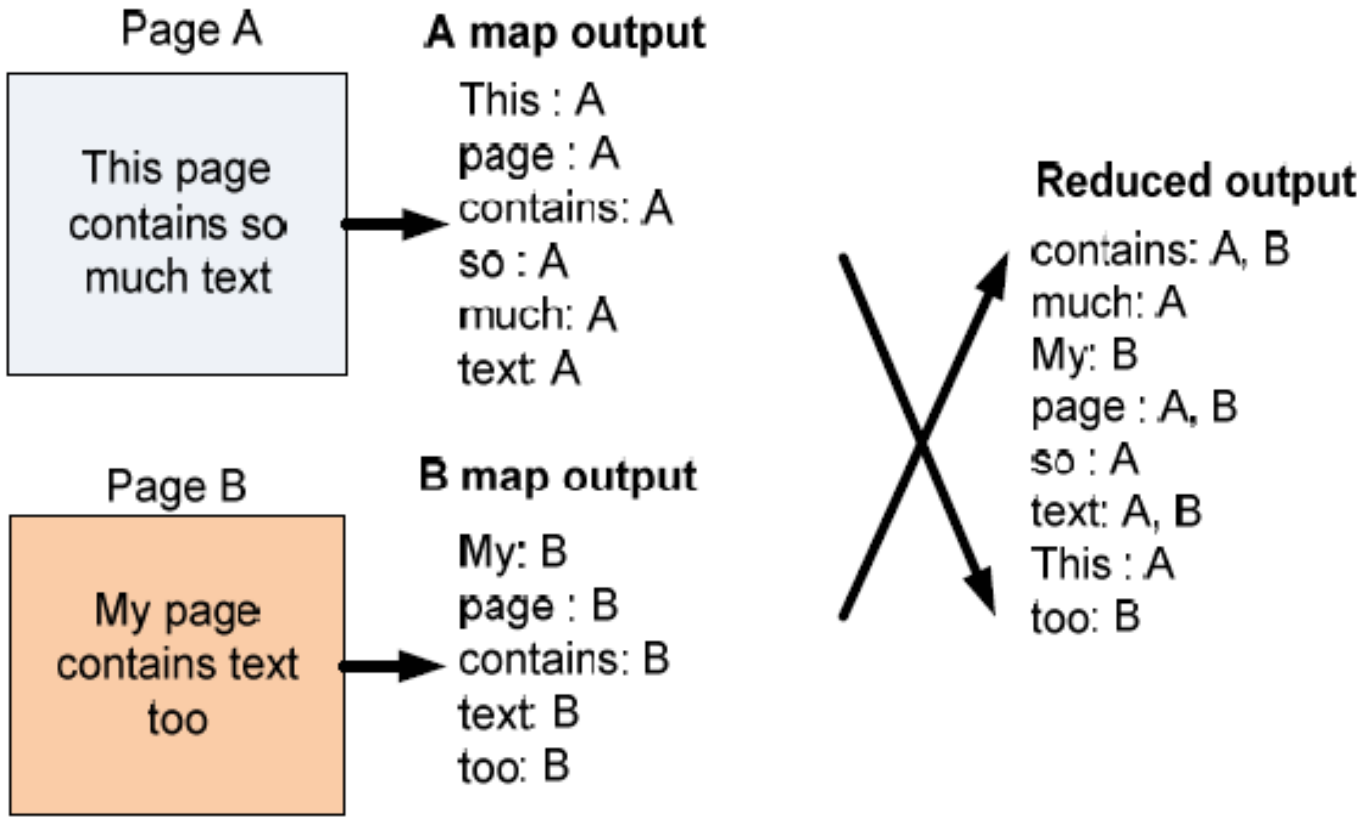
Suggest C&D to be Friends ☺

Example VI: Inverted Index for Large Collection of Documents

- Inverted index is a database index storing a mapping from content (e.g. words) to its locations (e.g. pages) in a document or a set of documents
- The purpose of an inverted index is to allow fast full-text searches.
- Inverted index algorithm becomes very challenging for documents on Internet:
  - Documents are in all different formats
  - Millions of Words and Short Phrases in Documents
  - Billions of short or long documents



Example VI: Inverted Index for Large Collection of Documents



Algorithm:

Step 1. For each page, get a list of all words on that page.

Step 2. For each word from Step 1, get a list of pages on which it appears.

Legacy vs. Modern: MapReduce and Its Successors

MapReduce is still in use in 2025, but

- MR is now **largely confined** to legacy systems and specific archival batch-processing contexts, rather than as the backbone for new, modern data analytics pipelines.
- MR strengths **remain relevant** for some organisations, especially those that have long-established MapReduce workflows or operate within sectors slow to migrate (e.g., traditional enterprises, government).
- MR has been increasingly replaced by faster, in-memory, stream-first processing frameworks like **Apache Spark**, which offer greater speed, flexibility, and cloud-readiness for real-time and AI workloads.

Dominant **Modern Alternatives**

- **Apache Spark** is now the ***primary platform for distributed data processing***, favoured for its in-memory operations, speed, and comprehensive ecosystem supporting batch, streaming, ML, and interactive analytics.
- **Apache Flink** and **Apache Beam** are used for event-driven and streaming-first workloads.
- **Databricks Delta Engine** and **Snowflake** offer transactional consistency, concurrent access, and high-speed analytics beyond what MapReduce can provide.

Outline

- Introduction to Hadoop
  - What is Hadoop & History
  - Hadoop Ecosystem
- Hadoop Computation Model: MapReduce
  - MapReduce Components & Paradigm
  - MapReduce Workflow
  - MapReduce Examples:
    - Word Count
    - Find the Highest/Averaged Temperature
    - Find Word Length Distribution
    - Find Mutual Friends
    - Inverted Indexing
- Besides the Core: Hive and Pig

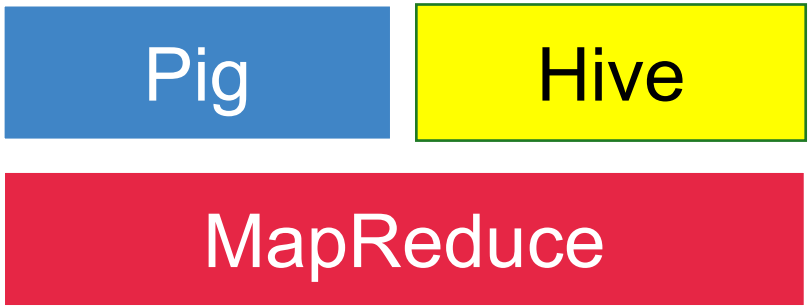
Hive and Pig

Hadoop is great for large-data processing!

- But writing Java programs for everything is verbose and slow
- Not everyone wants to (or can) write Java code

Solution: develop higher-level data processing languages and convert to Hadoop jobs

- Hive: HQL is like SQL
  - Query language is HQL, a variant of SQL
  - Tables stored on HDFS as flat files
  - Developed by Facebook, now open-source
- Pig:
  - Pig Latin is a bit like Perl
  - Scripts are written in Pig Latin, a dataflow language
  - Developed by Yahoo!, now open-source
  - Roughly 1/3 of all Yahoo! internal jobs



Hive

- Apache Hive supports analysis of large datasets stored in Hadoop's HDFS and compatible file systems such as Amazon S3 filesystem.
- It provides a SQL-like query language called HiveQL with schema on read and transparently converts queries to MapReduce
- HiveQL does not strictly follow the full SQL-92 standard but offers some extensions not found in SQL, such as multitable inserts.
- HiveQL has full ACID properties.
- Hive is a data warehouse that is good for OLAP rather than OLTP

```
1 DROP TABLE IF EXISTS docs;
2 CREATE TABLE docs (line STRING);
3 LOAD DATA INPATH 'input_file' OVERWRITE INTO TABLE docs;
4 CREATE TABLE word_counts AS
5 SELECT word, count(1) AS count FROM
6  (SELECT explode(split(line, '\s')) AS word FROM docs) temp
7 GROUP BY word
8 ORDER BY word;
```

Online Transactional Processing (OLTP) vs Online Analytical Processing (OLAP)

- **Online transaction processing**, or **OLTP**, is a class of information systems that facilitate and manage transaction-oriented applications, typically for data entry and retrieval transaction processing.
  - Processes operational data
  - Quick responses to user requests
  - Concurrently used by many users
  - Frequent updates (UPDATE, INSERT, DELETE) e.g., booking airline tickets.
  - HBase is a NoSQL database for OLTP
- **Online Analytical Processing,** or **OLAP**, is a class of systems that is designed to response to multi-dimensional analytical (MDA) queries.
  - No need to promptly respond
  - Hive is a data warehouse suitable for OLAP

Dimension: Product, Region, Year

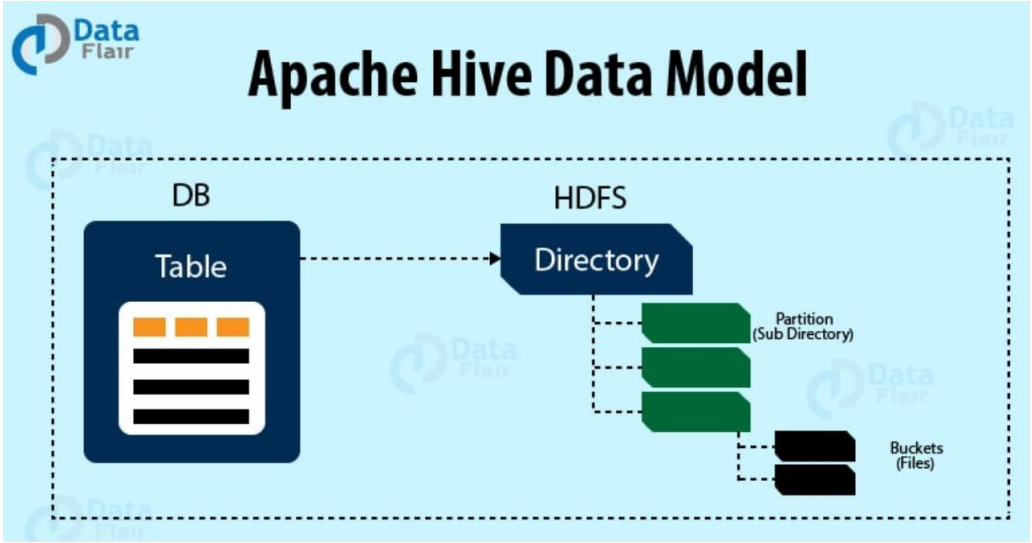
MDA example:

Show the total sales amount for each **product category** in **each region** for **2024**.

Hive Data Model

Data in Hive organized into :

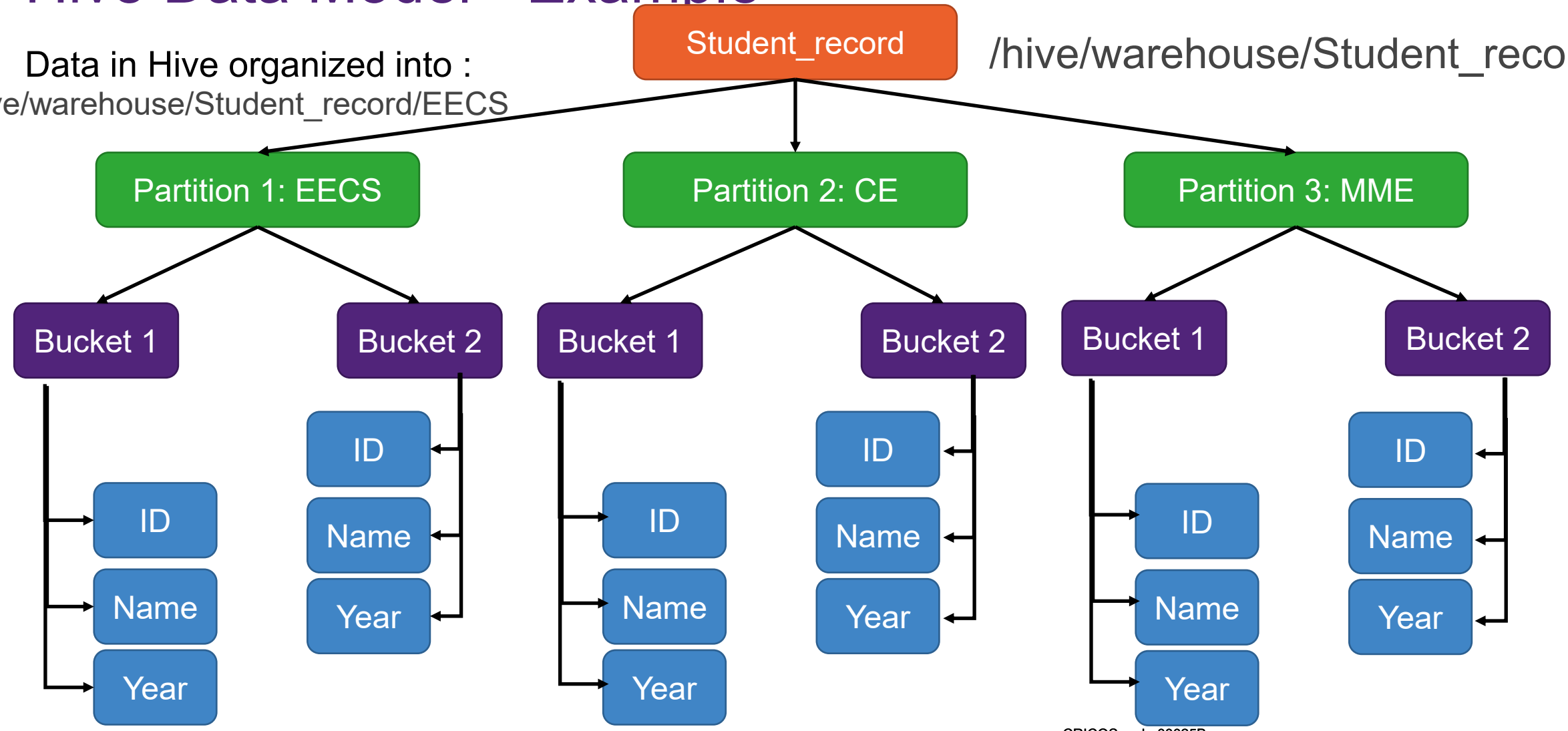
- Tables
  - Analogous to relational tables
  - Each table has a corresponding directory in HDFS
  - Data serialized and stored as files within that directory
- Partitions
  - Each table can be broken into partitions
  - Partitions determine distribution of data within subdirectories
- Buckets
  - Data in each partition divided into buckets
  - Based on a hash function of the column
  - Each bucket is stored as a file in partition directory



https://data-flair.training/blogs/hive-data-model/

Hive Data Model - Example

Data in Hive organized into :
/hive/warehouse/Student_record/EECS



https://data-flair.training/blogs/hive-data-model/

Apache Pig

- Apache Pig is a high-level platform for creating programs that run on Apache Hadoop.
- The language for this platform is called Pig Latin.
- Pig can execute its Hadoop jobs in MapReduce or Apache Spark.
- Pig Latin can be regarded as
  - High-level Map/Reduce commands pipeline
  - Very intuitive and friendly to users who dislike Java
- Pig Latin can be extended using user-defined functions (UDFs)
  - the user can write in Java, Python, JavaScript, etc
  - then call directly from the language.
- Limitations:
  - No loops and conditions (IF.. THEN)

Pig Slides adapted from Olston et al.

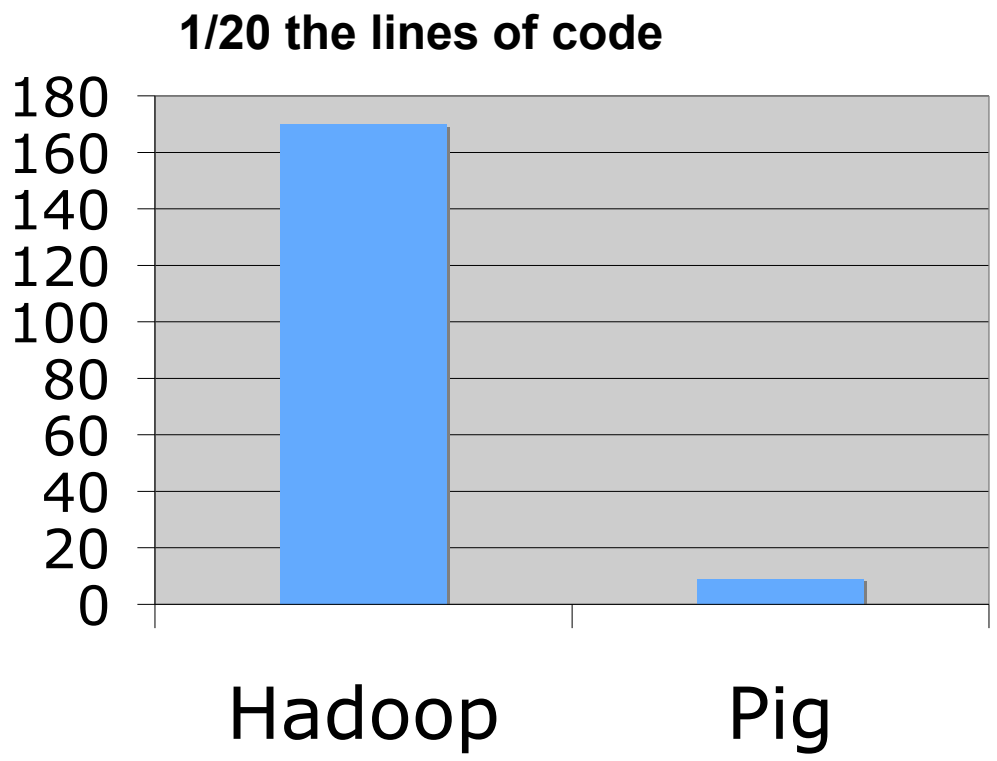
Pig Latin Script

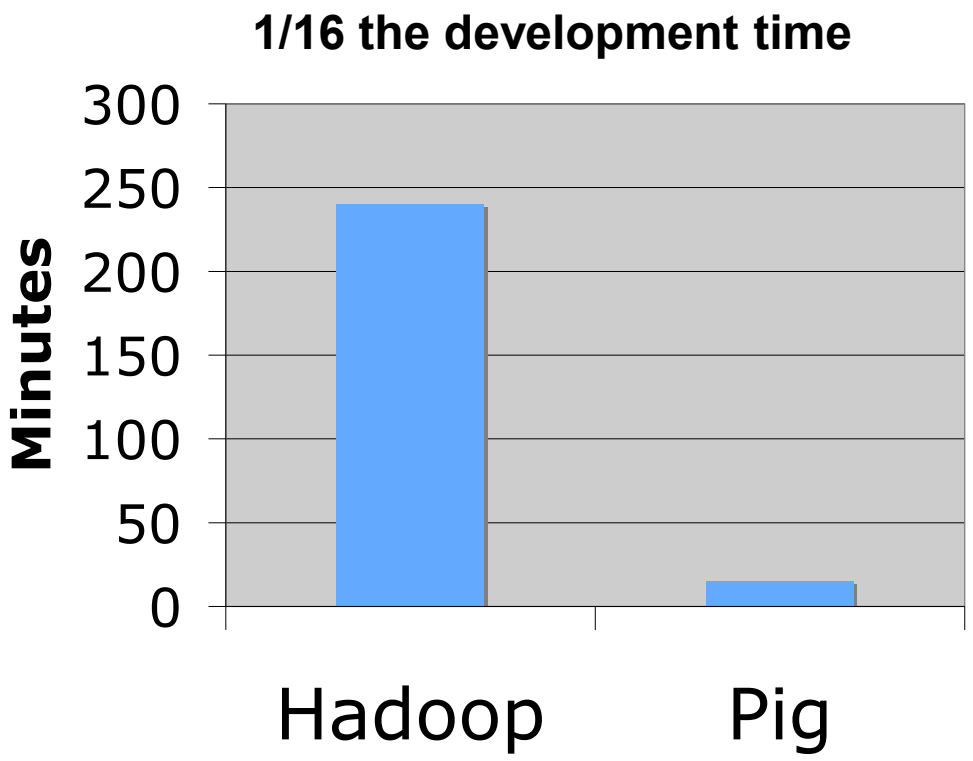
```
-- inner join
A = LOAD 'student_data' AS (name: chararray, age: int, gpa: float);
B = LOAD 'voter_data' AS (name: chararray, age: int, registration: chararay, contributions: float);
C = COGROUP A BY name, B BY name;
D = FILTER C BY not IsEmpty(A);
E = FILTER D BY not IsEmpty(B);
F = FOREACH E GENERATE flatten(A), flatten(B);
DUMP F;
```

```
 */
public class IsEmpty extends FilterFunc {

    @Override
    public Boolean exec(Tuple input) throws IOException {
        try {
            Object values = input.get(0);
            if (values instanceof DataBag)
                return ((DataBag)values).size() == 0;
            else if (values instanceof Map)
                return ((Map)values).size() == 0;
            else {
                int errCode = 2102;
                String msg = "Cannot test a " +
                DataType.findTypeName(values) + " for emptiness.";
                throw new ExecException(msg, errCode, PigException.BUG);
            }
        } catch (ExecException ee) {
            throw ee;
        }
    }
}
```

Java vs. Pig Latin





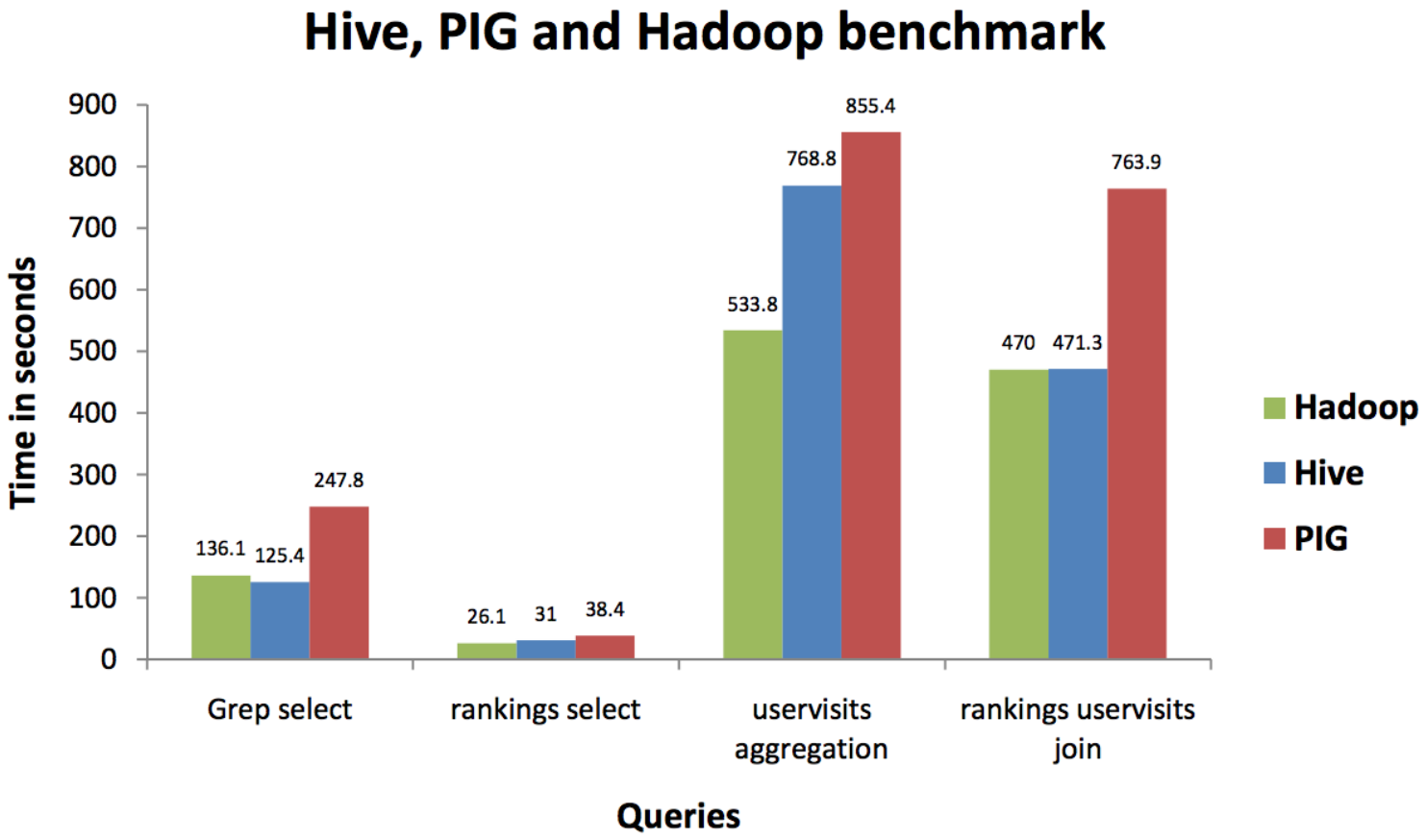
Performance on par with raw Hadoop!

Pig Slides adapted from Olston et al.

Hive v/s Pig

- **Similarities**:
  - Both High level Languages which work on top of map reduce framework
  - Can coexist since both use the under lying HDFS and map reduce
- **Differences**:
  - Language
    - Pig is a procedural ; (A = load ‘mydata’; dump A)
    - Hive is Declarative (select * from A)
  - Work Type
    - Pig more suited for adhoc analysis (on demand analysis of click stream search logs)
    - Hive a reporting tool (e.g. weekly Business Intelligence reporting)
  - Users
    - Pig – Researchers, Programmers (build complex data pipelines, machine learning)
    - Hive – Business Analysts

Head-to-Head (the Bee, the Pig, the Elephant)



Version: Hadoop – 0.18x, Pig:786346, Hive:786346

Guest Lecture in Week 12

Cloud Computing (INFS3208)

Lecture 12: Security & Privacy

Guest Lecturer

Cheng Jiang He/Him · 2nd
Lead Platform Engineer @ Dabble
Greater Brisbane Area · **Contact info**

Part I

- Guest Lectures' exclusive content

Part II

- Security in Cloud Computing
- Privacy in Cloud Computing

References

1.https://data-flair.training/blogs/hadoop-mapreduce-tutorial/

2. CSE 40822/60822, Cloud Computing, https://www3.nd.edu/~dthain/courses/cse40822/fall2018/

3.MapReduce: Simplified Data Processing on Large Clusters

4.Hadoop Tutorial (https://www.tutorialspoint.com/hadoop/index.htm)