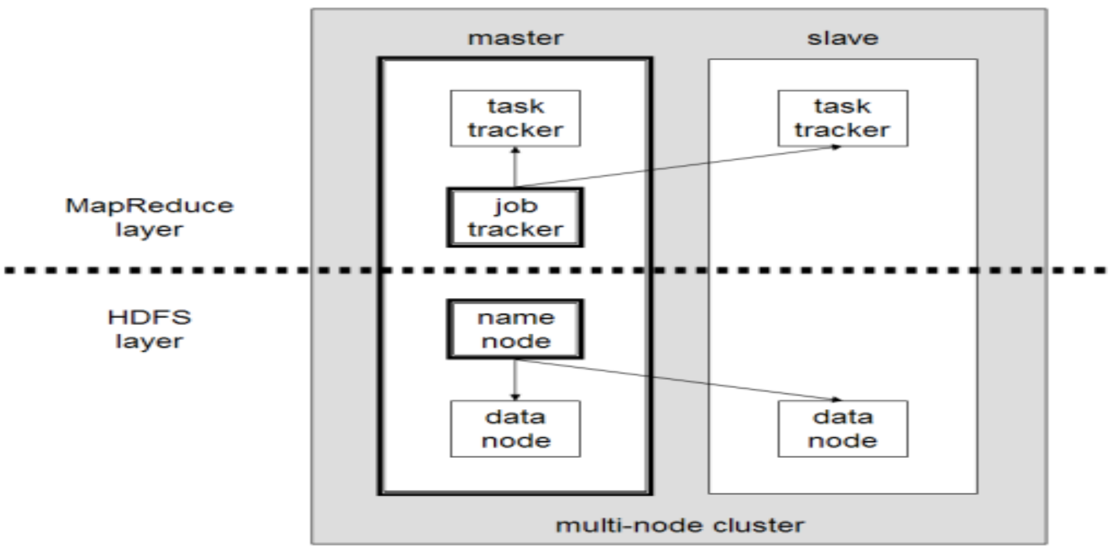
Hadoop: A Framework for Data-Intensive Distributed Computing

CS561-Spring 2012
WPI, Mohamed Y. Eltabakh

What is Hadoop?

- Hadoop is a software framework for *distributed processing* of *large datasets* across *large clusters* of computers
- Hadoop is open-source implementation for Google MapReduce
- Hadoop is based on a simple programming model called *MapReduce*
- Hadoop is based on a simple data model, *any data will fit*
- Hadoop framework consists on two main layers
  - Distributed file system (HDFS)
  - Execution engine (MapReduce)



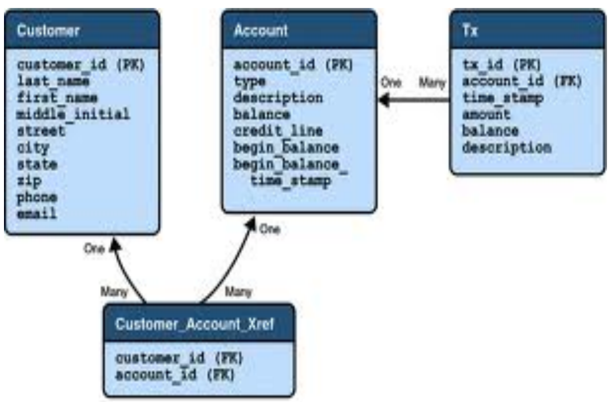
Hadoop Infrastructure

- Hadoop is a ***distributed*** system like ***distributed databases***

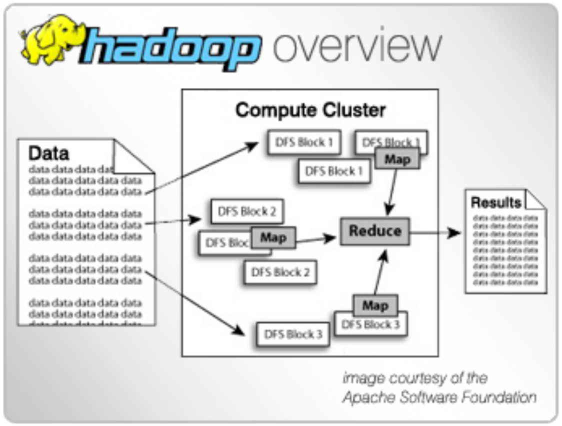
- **However, there are several key differences between the two infrastructures**
  - **Data model**
  - **Computing model**
  - **Cost model**
  - **Design objectives**

How Data Model is Different?

Distributed Databases



Hadoop

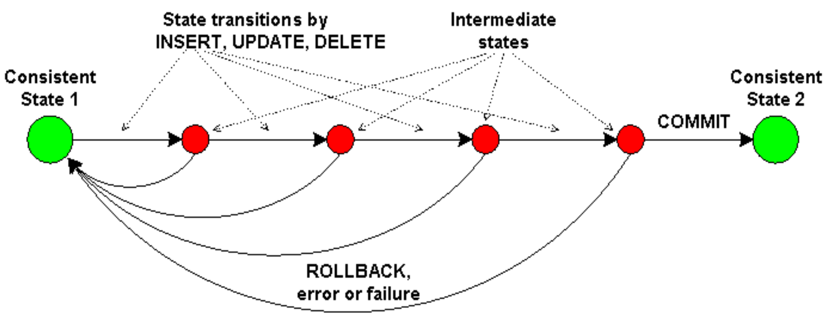


- Deal with tables and relations
- Must have a schema for data
- Data fragmentation & partitioning

- Deal with flat files in any format
- No schema for data
- Files are divide automatically into blocks

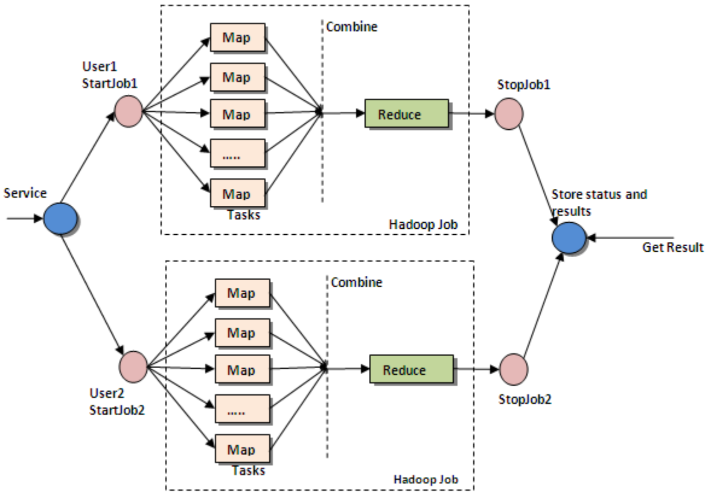
How Computing Model is Different?

Distributed Databases



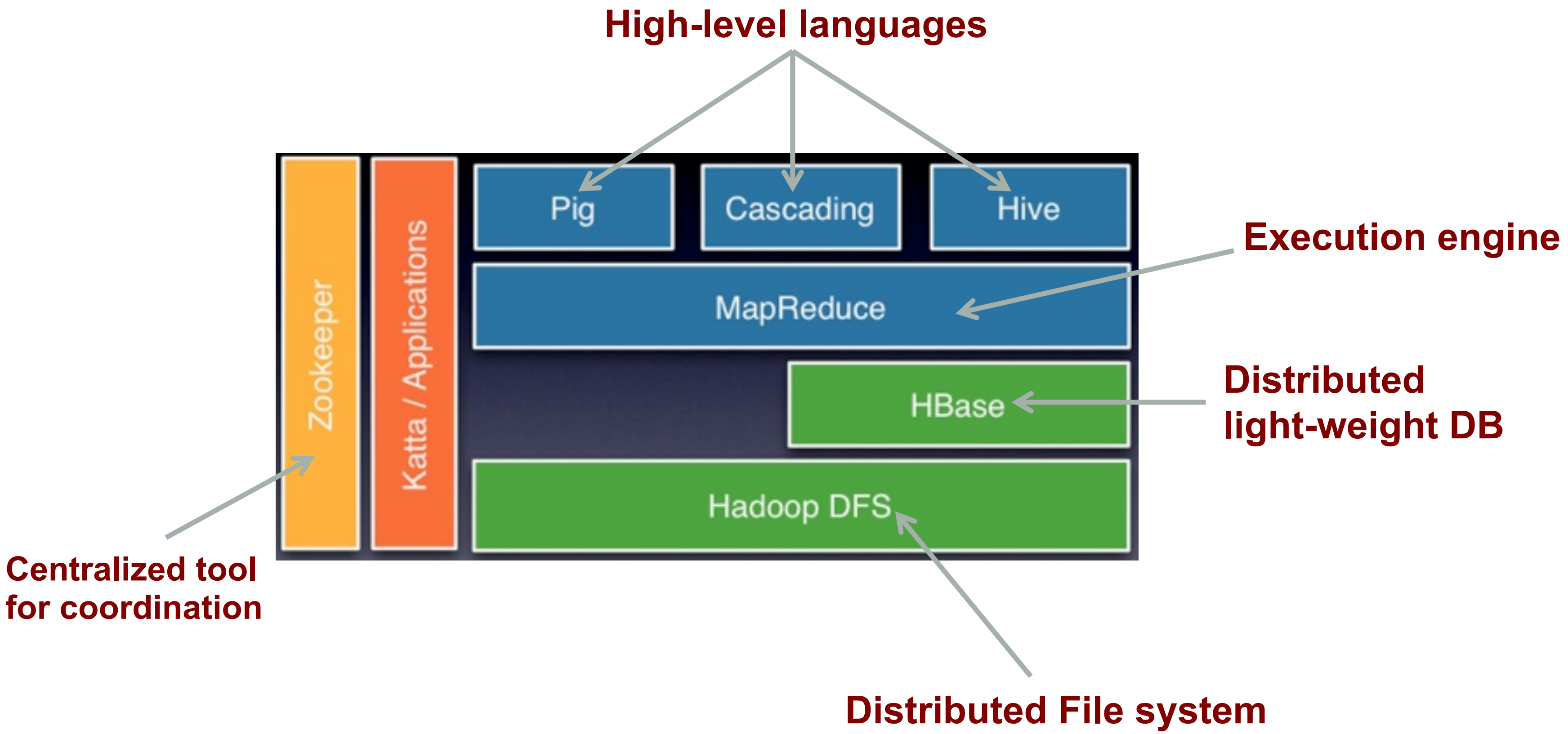
- Notion of a transaction
- Transaction properties ACID
- Distributed transaction

Hadoop



- Notion of a job divided into tasks
- Map-Reduce computing model
- Every task is either a map or reduce

Hadoop: Big Picture

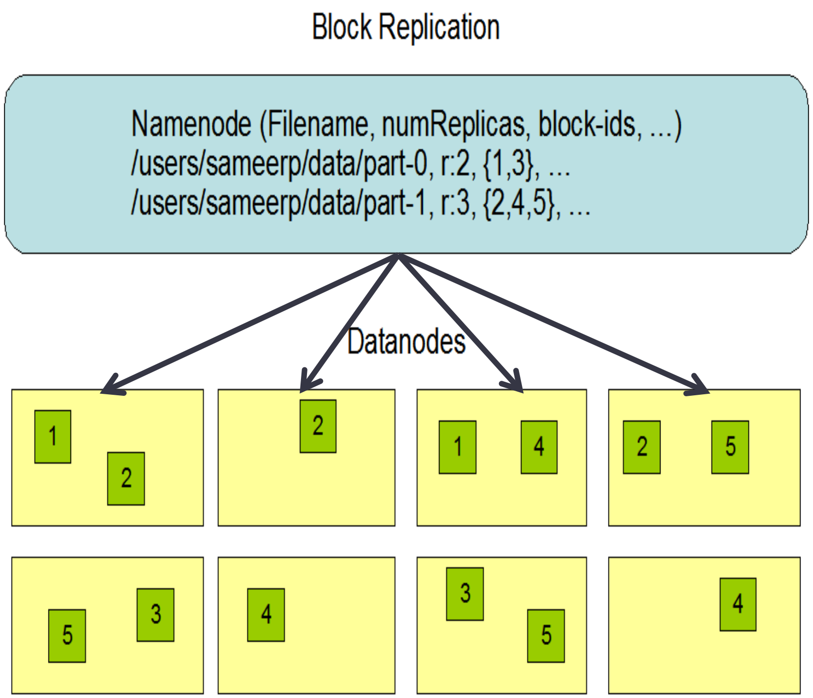


HDFS + MapReduce are enough to have things working

What is Next?

- **Hadoop Distributed File System (HDFS)**
- **MapReduce Layer**
- **Examples**
  - Word Count
  - Join
- **Fault Tolerance in Hadoop**

HDFS: Hadoop Distributed File System

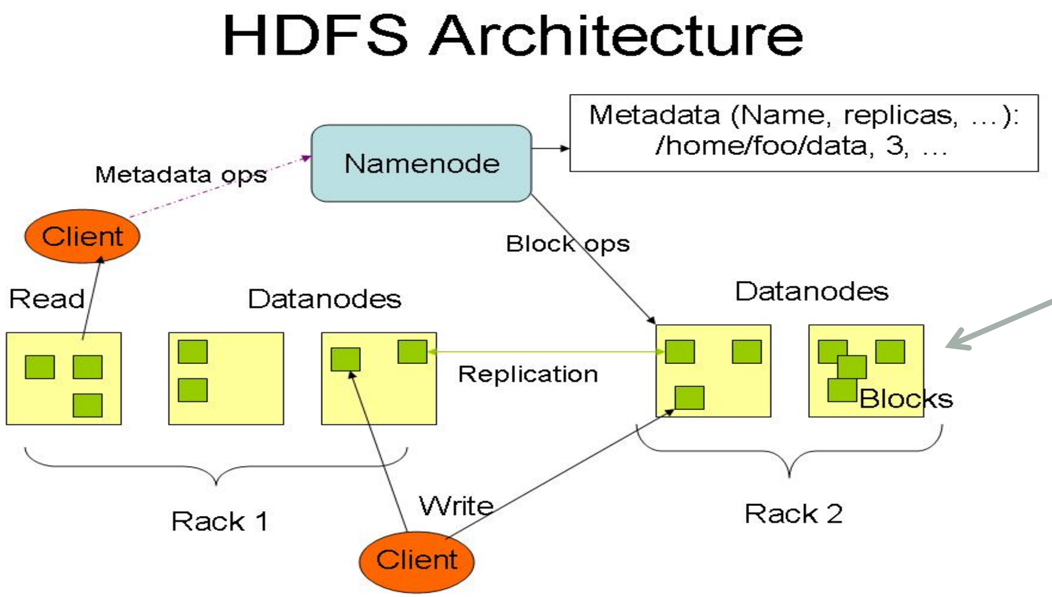


- Single namenode and many datanodes
- Namenode maintains the file system metadata

- Files are split into fixed sized blocks and stored on data nodes (Default 64MB)
- Data blocks are replicated for fault tolerance and fast access (Default is 3)
- Datanodes periodically send heartbeats to namenode

- HDFS is a master-slave architecture
  - Master: namenode
  - Slaves: datanodes (100s or 1000s of nodes)

HDFS: Data Placement and Replication



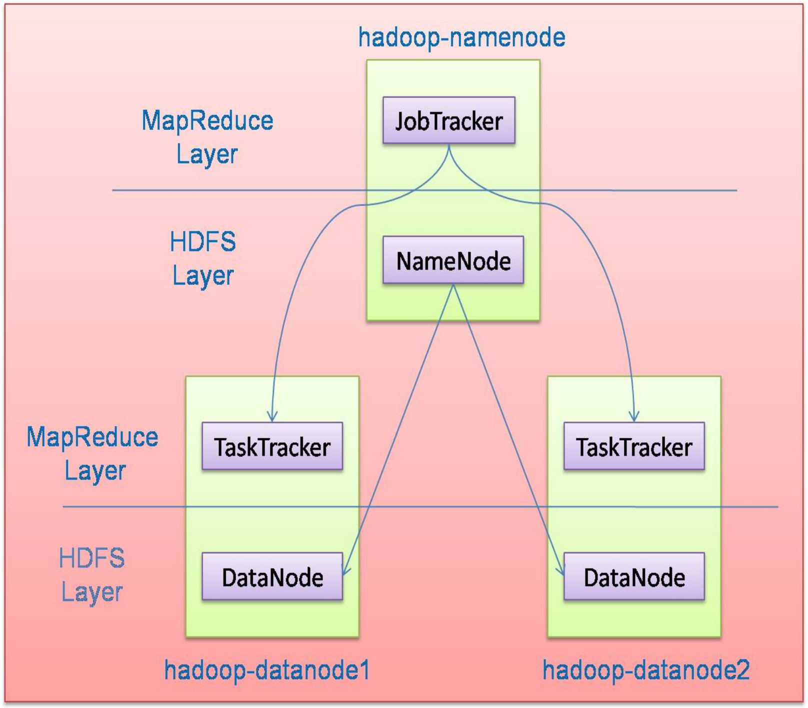
Datanodes can be organized into racks

- Default placement policy: Where to put a given block?
  - **First copy** is written to the node creating the file (write affinity)
  - **Second copy** is written to a data node within the same rack
  - **Third copy** is written to a data node in a different rack
  - **Objectives:** *load balancing, fast access, fault tolerance*

What is Next?

- **Hadoop Distributed File System (HDFS)**
- **MapReduce Layer**
- **Examples**
  - **Word Count**
  - **Join**
- **Fault Tolerance in Hadoop**

MapReduce: Hadoop Execution Layer



- Jobtracker knows everything about submitted jobs
- Divides jobs into tasks and decides where to run each task
- Continuously communicating with tasktrackers

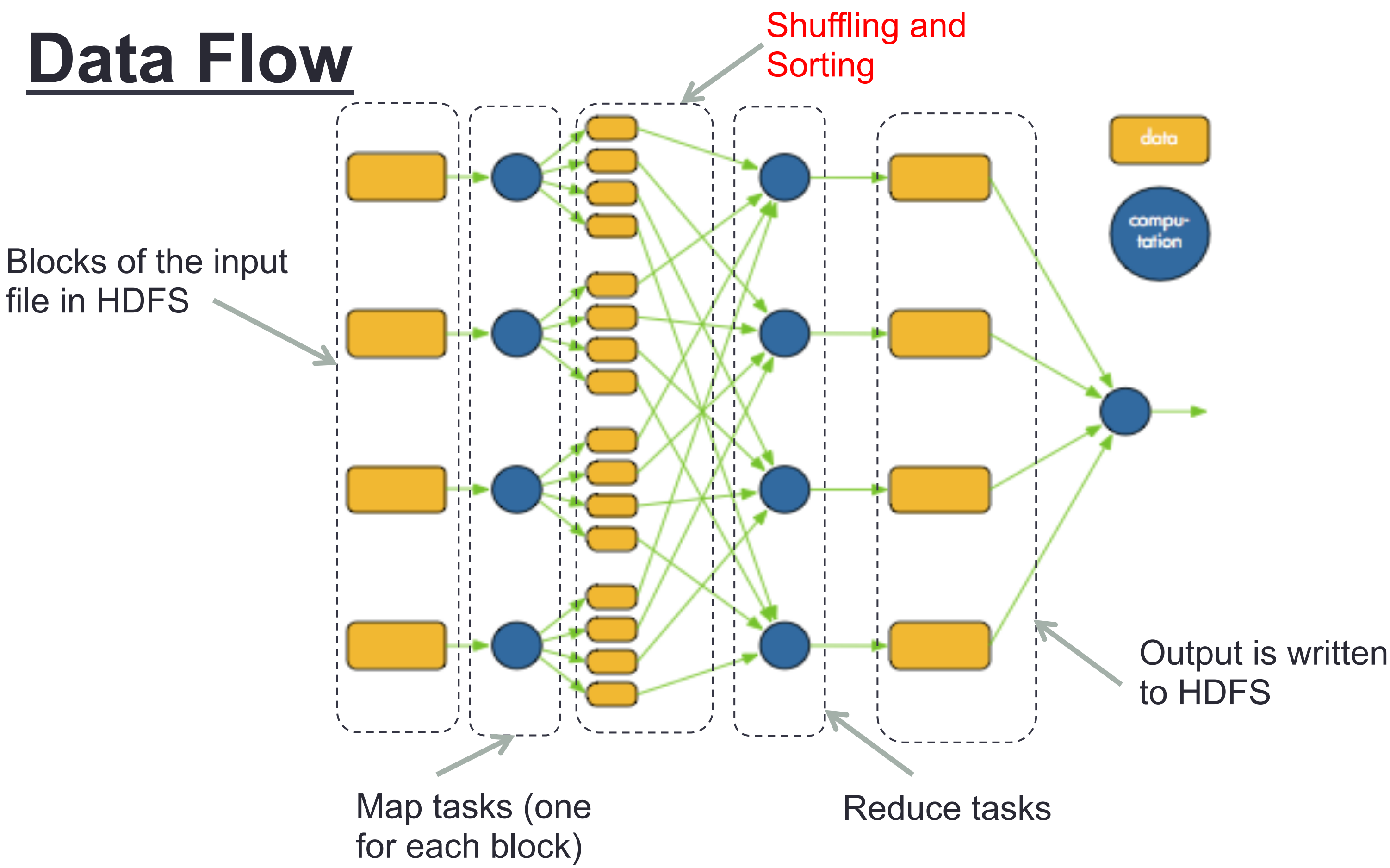
- Tasktrackers execute tasks (multiple per node)
- Monitors the execution of each task
- Continuously sending feedback to Jobtracker

- MapReduce is a master-slave architecture
  - Master: JobTracker
  - Slaves: TaskTrackers (100s or 1000s of tasktrackers)
- Every datanode is running a tasktracker

Hadoop Computing Model

- Two main phases: Map and Reduce
- Any job is converted into *map* and *reduce* tasks
- Developers need *ONLY* to implement the *Map* and *Reduce* classes

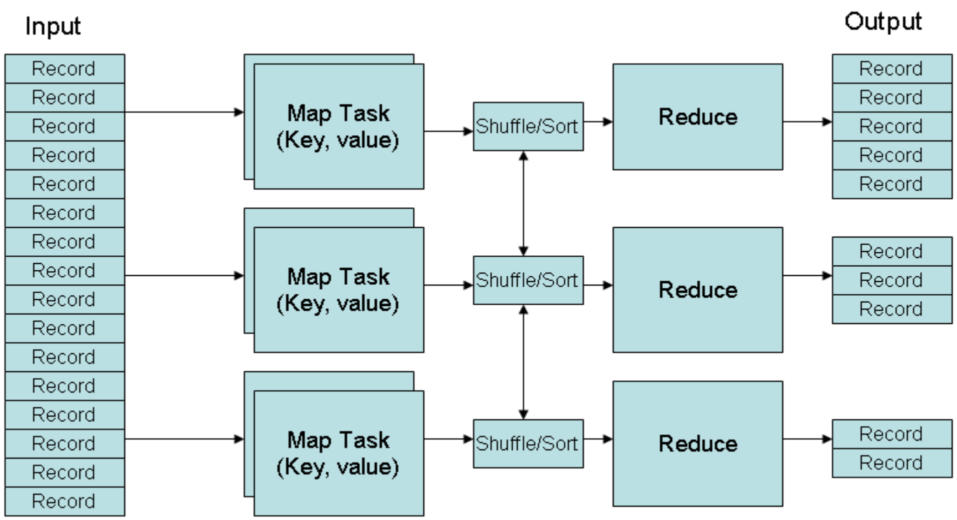
Data Flow



Hadoop Computing Model (Cont'd)

- Mapper and Reducers consume and produce *(Key, Value)* pairs
- Users define the data type of the *Key* and *Value*

- Shuffling & Sorting phase:
  - Map output is *shuffled* such that all same-key records go to the same reducer
  - Each reducer may receive multiple key sets
  - Each reducer *sorts* its records to group similar keys, then process each group

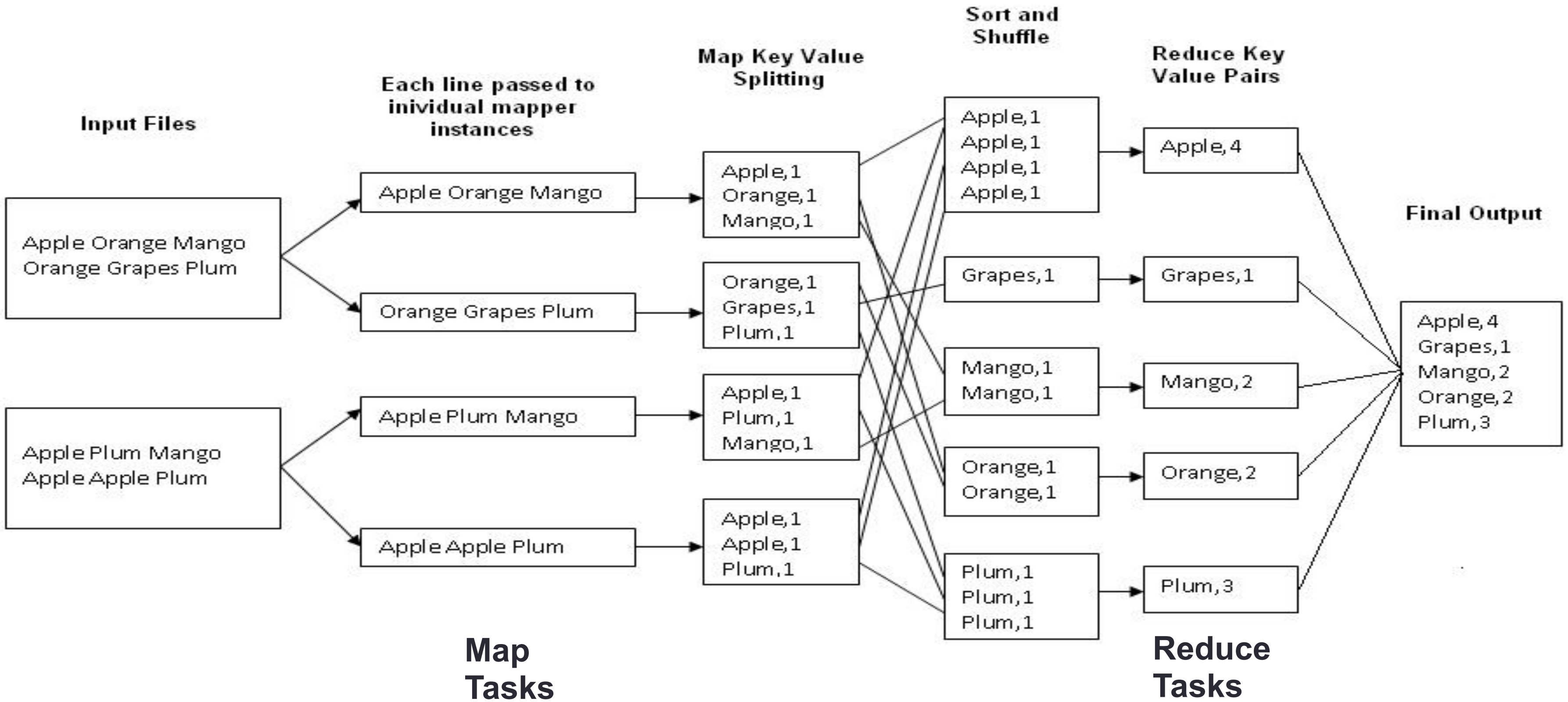


What is Next?

- **Hadoop Distributed File System (HDFS)**
- **MapReduce Layer**
- **Examples**
  - **Word Count**
  - **Join**
- **Fault Tolerance in Hadoop**

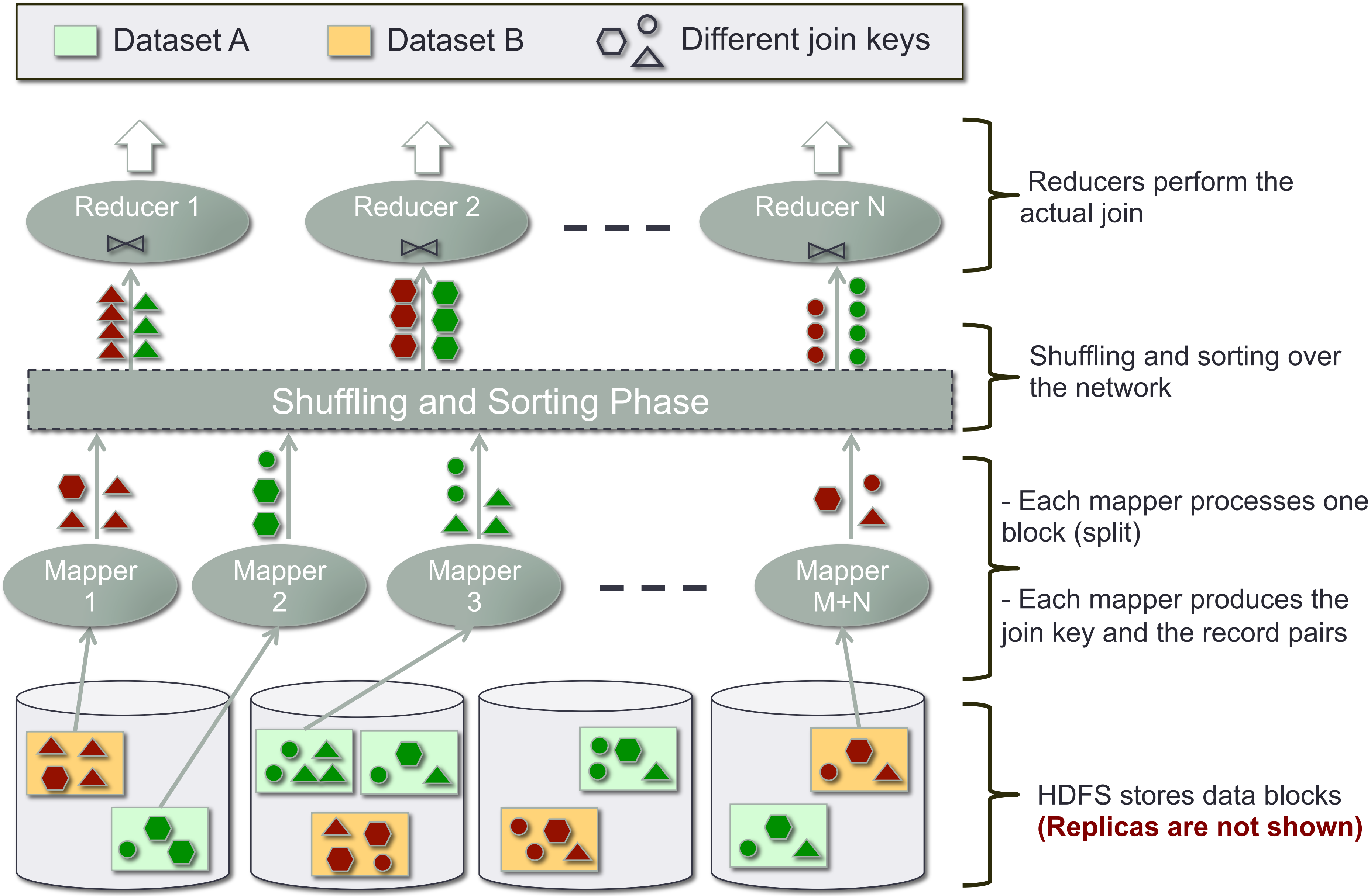
Word Count

- Job: Count the occurrences of each word in a data set



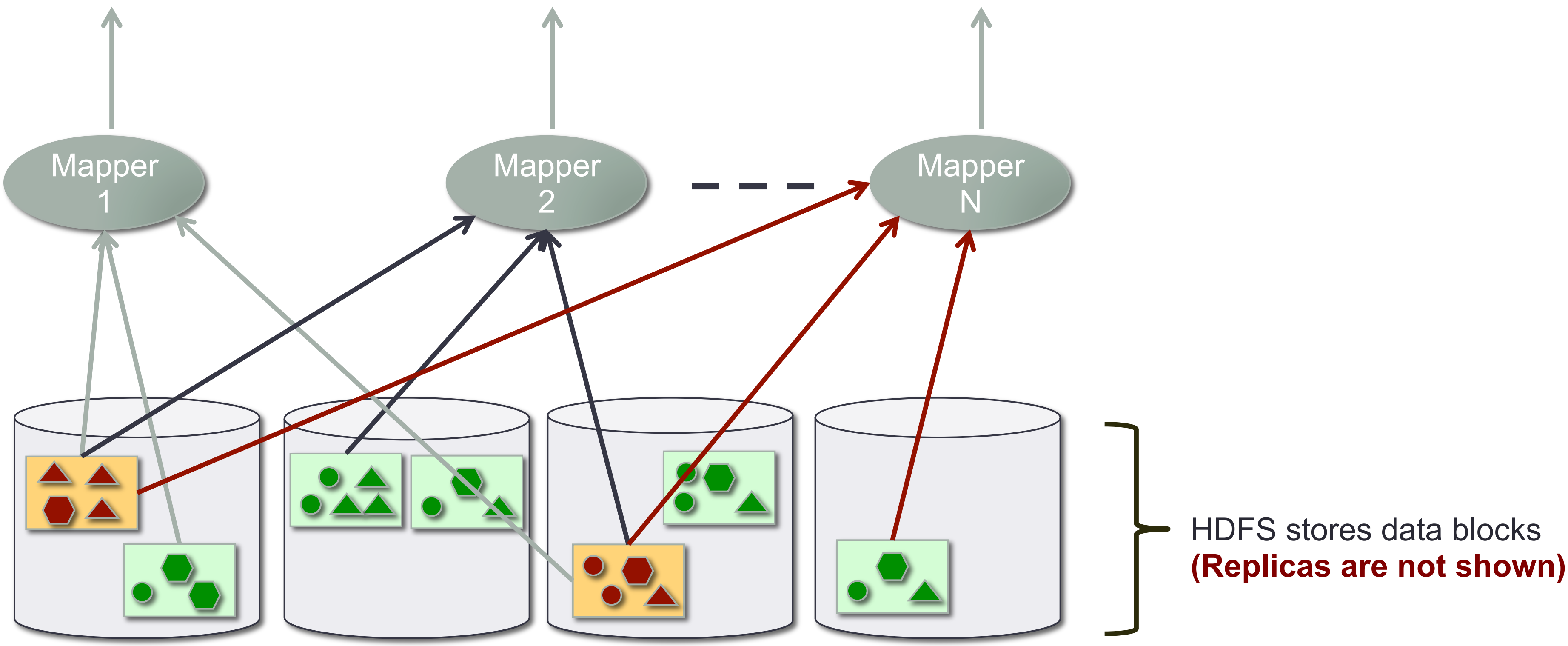
Reduce phase is optional: Jobs can be MapOnly

Joining Two Large Datasets



Joining Large Dataset (A) with Small Dataset (B)

- Every map task processes one block from A and the entire B
- Every map task performs the join ***(MapOnly job)***
- Avoid the shuffling and reduce *expensive* phases



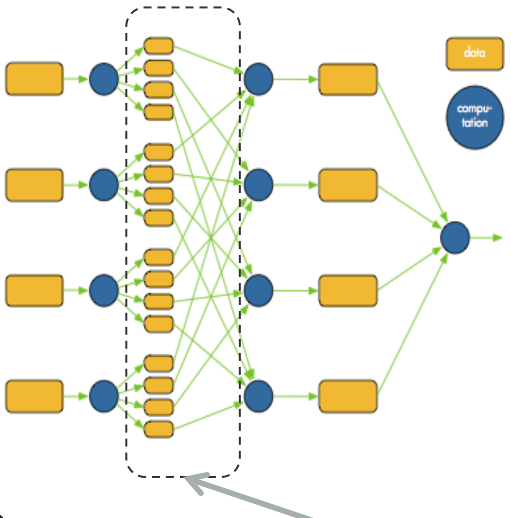
What is Next?

- **Hadoop Distributed File System (HDFS)**
- **MapReduce Layer**
- **Examples**
    - **Word Count**
    - **Join**
- **Fault Tolerance in Hadoop**

Hadoop Fault Tolerance

- Intermediate data between mappers and reducers are *materialized* to simple & straightforward fault tolerance

- **What if a task fails (map or reduce)?**
  - Tasktracker detects the failure
  - Sends message to the jobtracker
  - Jobtracker re-schedules the task



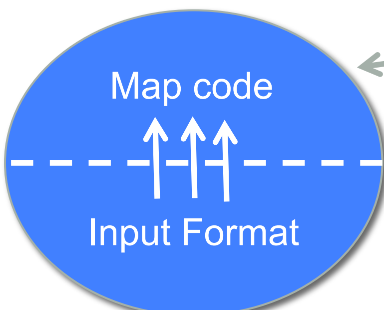
Intermediate data (materialized)

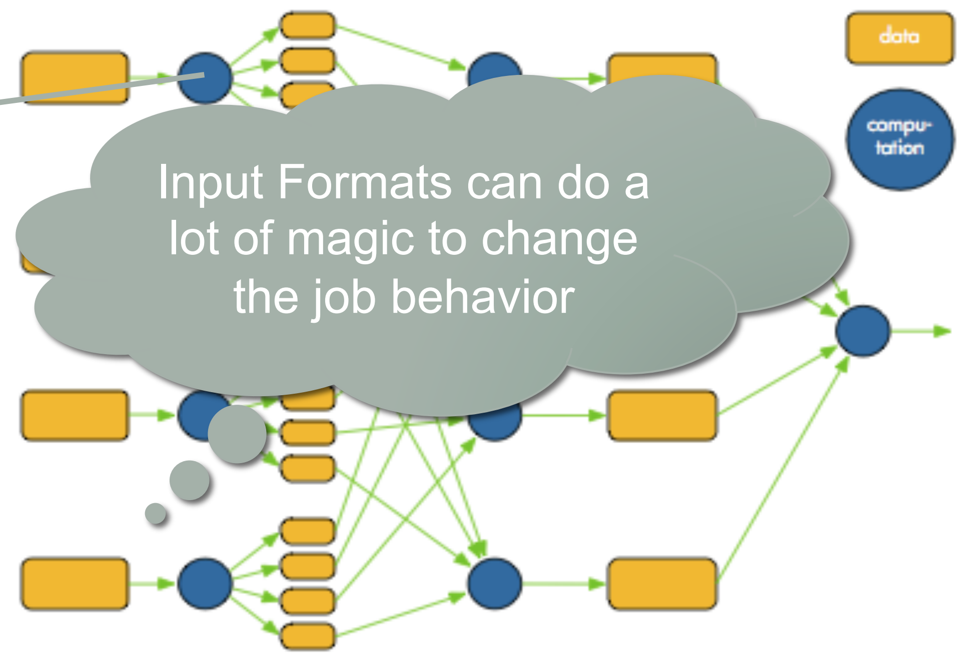
- **What if a datanode fails?**
  - Both namenode and jobtracker detect the failure
  - All tasks on the failed node are re-scheduled
  - Namenode replicates the users' data to another node

- **What if a namenode or jobtracker fails?**
  - The entire cluster is down

Reading/Writing Files

- ***Recall: Any data will fit in Hadoop, so*** **how does Hadoop understand/read the data?**
- User-pluggable class "*Input Format*"
  - Input formats know how to parse and read the data (convert byte stream to records)
  - Each record is then passed to the mapper for processing
- Hadoop provides *built-in* Input Formats for reading *text* & *sequence* files



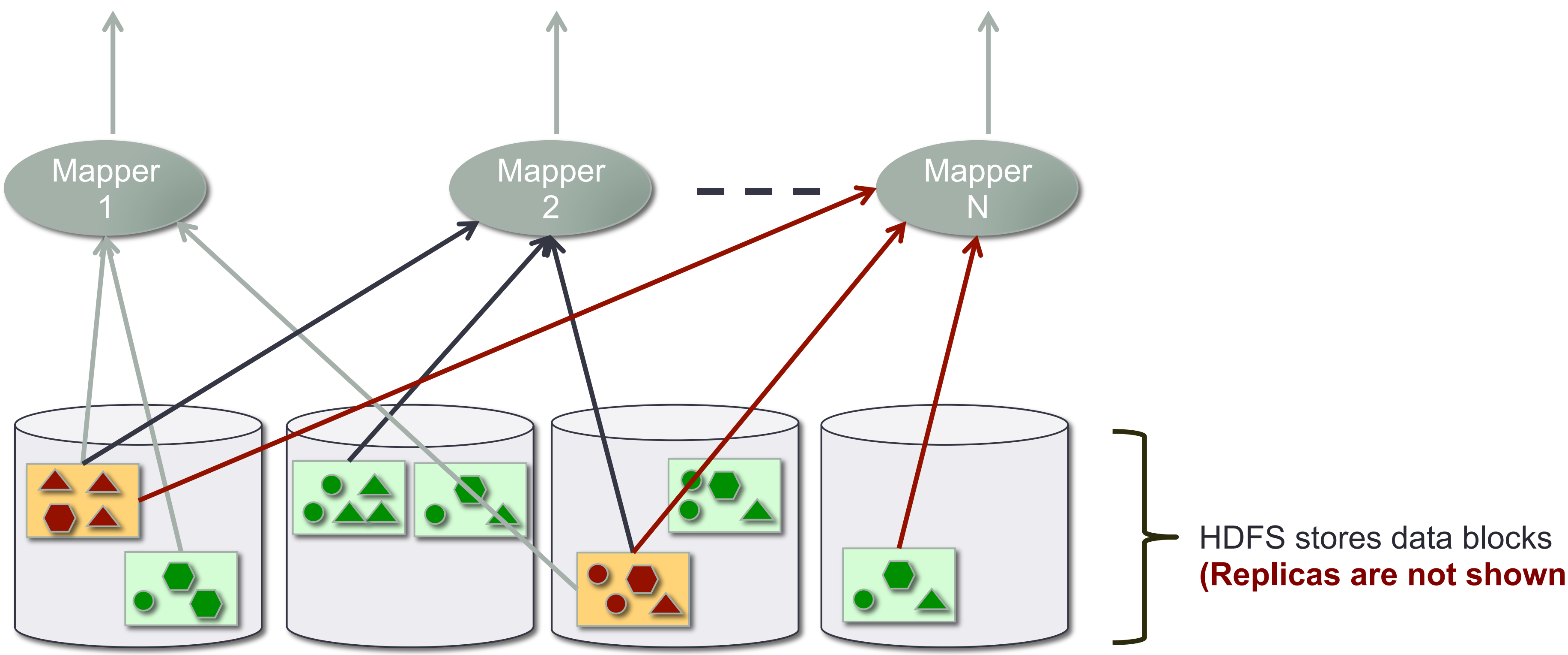


The same for writing →
"Output Formats"

Back to Joining Large & Small Datasets

Dataset A   Dataset B   Different join keys

- Every map task processes one block from A and the entire B
- How does a single mapper reads multiple splits (from different datasets)?
  - Customized input formats



Using Hadoop

- **Java language**

- **High-Level Languages**
  - Hive (Facebook)
  - Pig (Yahoo)
  - Jaql (IBM)

Java Code Example

Source Code

WordCount.java

```
package org.myorg;

import java.io.IOException;
import java.util.*;

import org.apache.hadoop.fs.Path;
import org.apache.hadoop.conf.*;
import org.apache.hadoop.io.*;
import org.apache.hadoop.mapred.*;
import org.apache.hadoop.util.*;

```

Import Hadoop libs

```
public class WordCount {

 public static class Map extends MapReduceBase implements Mapper<LongWritable, Text, Text, IntWritable> {
  private final static IntWritable one = new IntWritable(1);
  private Text word = new Text();

  public void map(LongWritable key, Text value, OutputCollector<Text, IntWritable> output, Reporter reporter) throws IOException {
   String line = value.toString();
   StringTokenizer tokenizer = new StringTokenizer(line);
   while (tokenizer.hasMoreTokens()) {
    word.set(tokenizer.nextToken());
    output.collect(word, one);
   }
  }
 }
```

Map class

```

 public static class Reduce extends MapReduceBase implements Reducer<Text, IntWritable, Text, IntWritable> {
  public void reduce(Text key, Iterator<IntWritable> values, OutputCollector<Text, IntWritable> output, Reporter reporter) throws IOException {
   int sum = 0;
   while (values.hasNext()) {
    sum += values.next().get();
   }
   output.collect(key, new IntWritable(sum));
  }
 }
```

Reduce class

```

 public static void main(String[] args) throws Exception {
  JobConf conf = new JobConf(WordCount.class);
  conf.setJobName("wordcount");

  conf.setOutputKeyClass(Text.class);
  conf.setOutputValueClass(IntWritable.class);

  conf.setMapperClass(Map.class);
  conf.setCombinerClass(Reduce.class);
  conf.setReducerClass(Reduce.class);

  conf.setInputFormat(TextInputFormat.class);
  conf.setOutputFormat(TextOutputFormat.class);

  FileInputFormat.setInputPaths(conf, new Path(args[0]));
  FileOutputFormat.setOutputPath(conf, new Path(args[1]));

  JobClient.runJob(conf);
 }
}
```

Job configuration

Hive Language

- High-level language on top of Hadoop
  - Like SQL on top of DBMSs
- Support structured data, e.g., creating tables, as well as extensibility for un-structured data

page_view

pageid	userid	time
1	111	9:08:01
2	111	9:08:13
1	222	9:08:14

X

user

userid	age	gender
111	25	female
222	32	male

=

pv_users

pageid	age
1	25
2	25
1	32

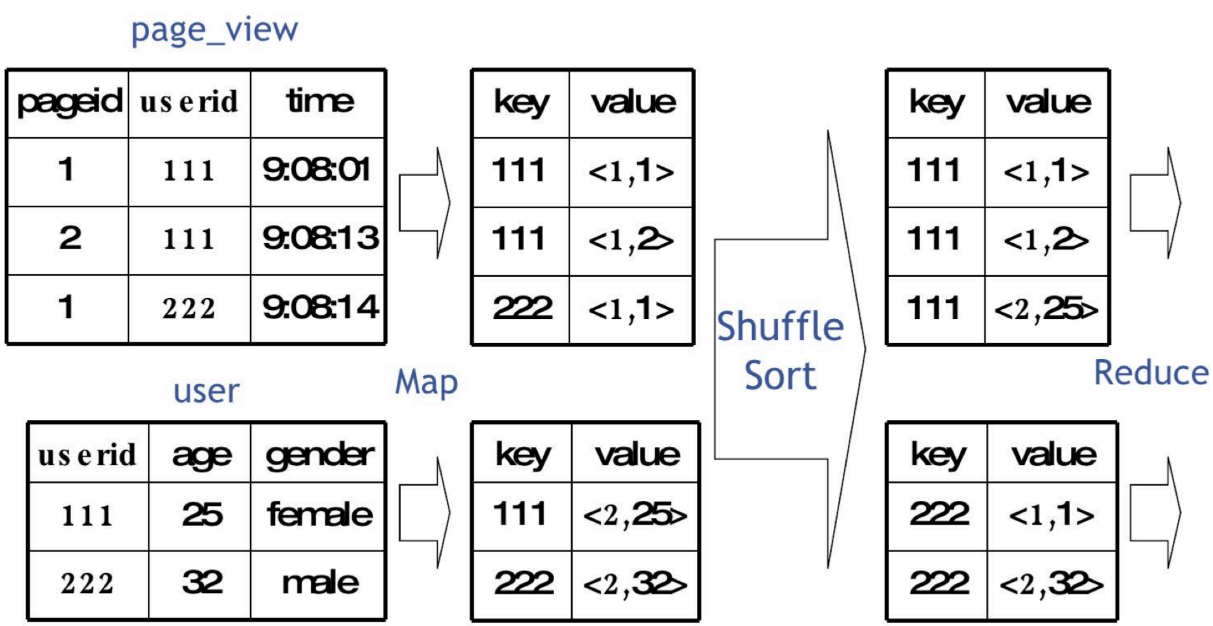
```
INSERT INTO TABLE pv_users
SELECT pv.pageid, u.age
FROM page_view pv JOIN user u ON (pv.userid = u.userid);
```

```
Create Table user (userID int,
                   age int,
                   gender char)
Row Format Delimited Fields;
```

```
Load Data Local Inpath '/user/
local/users.txt' into Table user;
```

From Hive To MapReduce

```
INSERT INTO TABLE pv_users
SELECT pv.pageid, u.age
FROM page_view pv JOIN user u ON (pv.userid = u.userid);
```



Hive: Group By

pv_users

pageid	age
1	25
2	25
1	32
2	25

pageid	age	count
1	25	1
2	25	2
1	32	1

```
SELECT pageid, age, count(1)
FROM pv_users
GROUP BY pageid, age;
```

pv_users

pageid	age
1	25
2	25

pageid	age
1	32
2	25

Map

key	value
<1,25>	1
<2,25>	1

key	value
<1,32>	1
<2,25>	1

Shuffle
Sort

key	value
<1,25>	1
<1,32>	1

key	value
<2,25>	1
<2,25>	1

Reduce

Summary

- **Hadoop is a distributed systems for processing large-scale datasets**
  - Scales to thousands of nodes and petabytes of data
- **Two main layers**
  - HDFS: Distributed file system(NameNode is centralized)
  - MapReduce: Execution engine (JobTracker is centralized)
- **Simple data model, any format will fit**
  - At query time, specify how to read (write) the data using input (output) formats
- **Simple computation model based on Map-Reduce phases**
  - Very efficient in aggregation and joins
- **Higher-level Languages on top of Hadoop**
  - Hive, Jaql, Pig

Summary: Hadoop vs. Other Systems

	Distributed Databases	Hadoop
Computing Model	- Notion of transactions - Transaction is the unit of work - ACID properties, Concurrency control	- Notion of jobs - Job is the unit of work - No concurrency control
Data Model	- Structured data with known schema - Read/Write mode	- Any data will fit in any format - (un)(semi)structured - ReadOnly mode
Cost Model	- Expensive servers	- Cheap commodity machines
Fault Tolerance	- Failures are rare - Recovery mechanisms	- Failures are common over thousands of machines - Simple yet efficient fault tolerance
Key Characteristics	- Efficiency, optimizations, fine-tuning	- Scalability, flexibility, fault tolerance

- **Cloud Computing**
  - A computing model where any computing infrastructure can run on the cloud
  - Hardware & Software are provided as remote services
  - Elastic: grows and shrinks based on the user's demand
  - Example: Amazon EC2