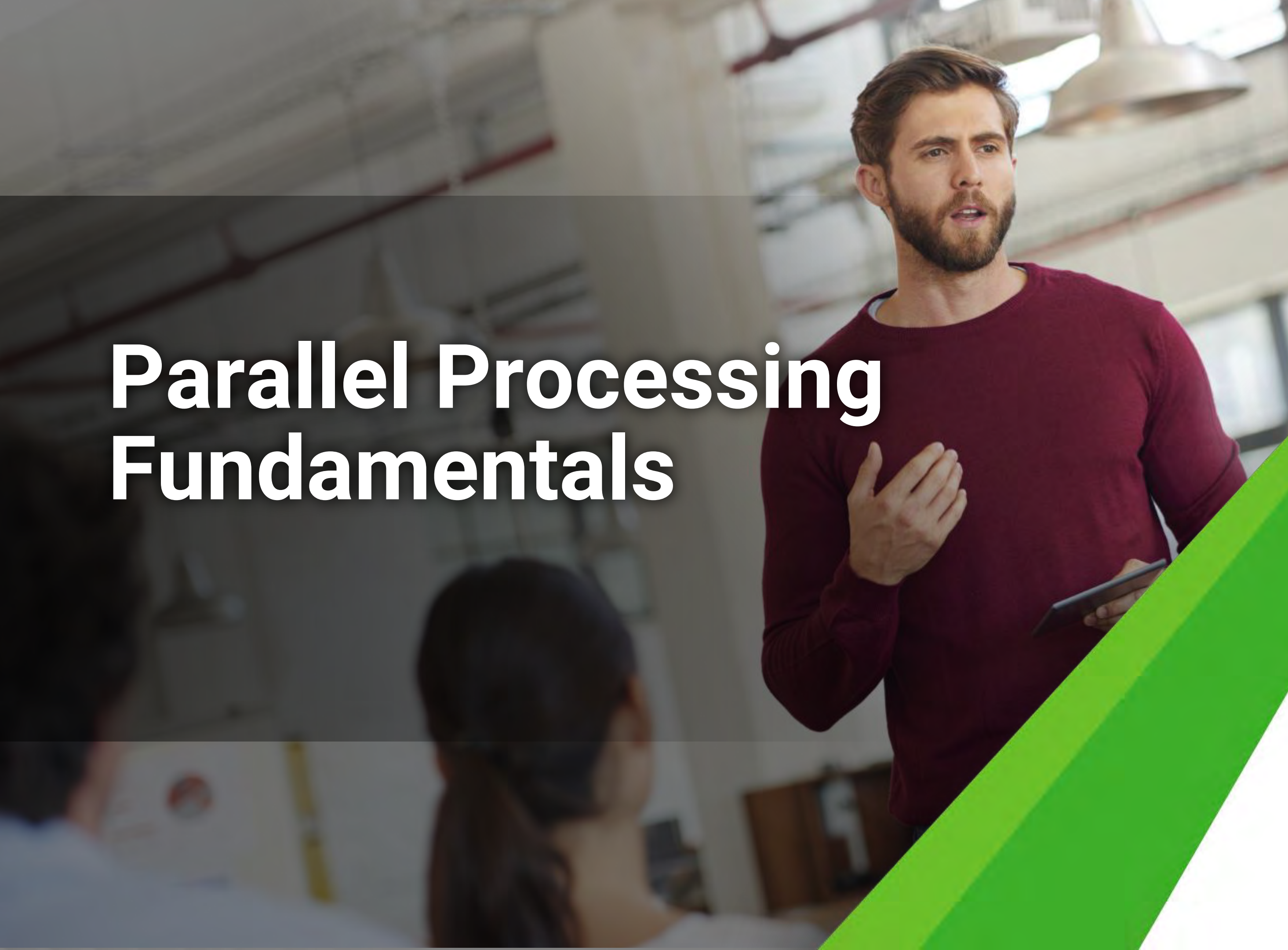
Parallel Processing Fundamentals



Lesson Objectives

After completing this lesson, students should be able to:

- Describe how MapReduce works
  - Explain the reliance on the Key Value Pair (KVP) paradigm
  - Illustrate the MapReduce framework with simple examples

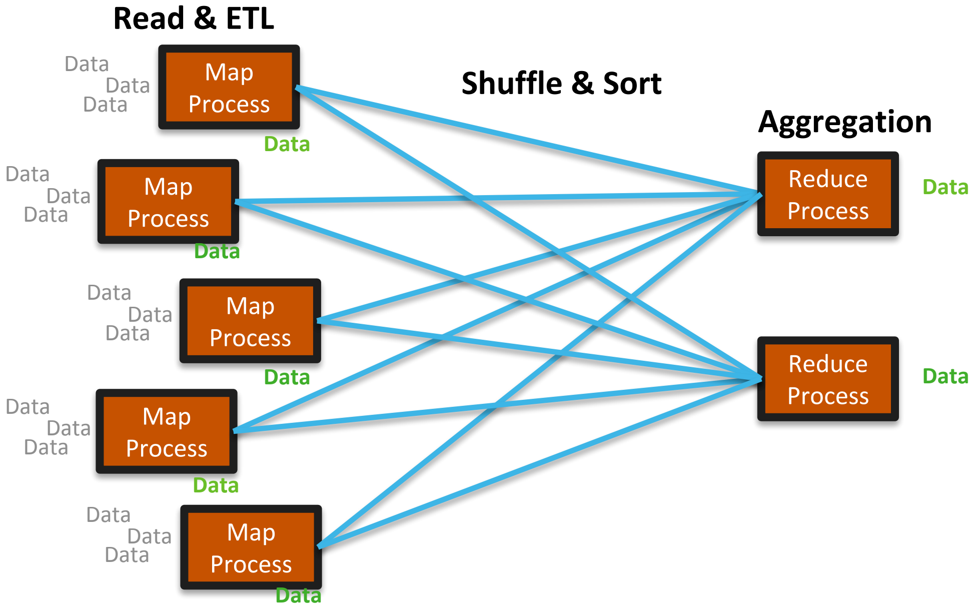
The MapReduce Framework

HORTONWORKS

UNIVERSITY

What is MapReduce?

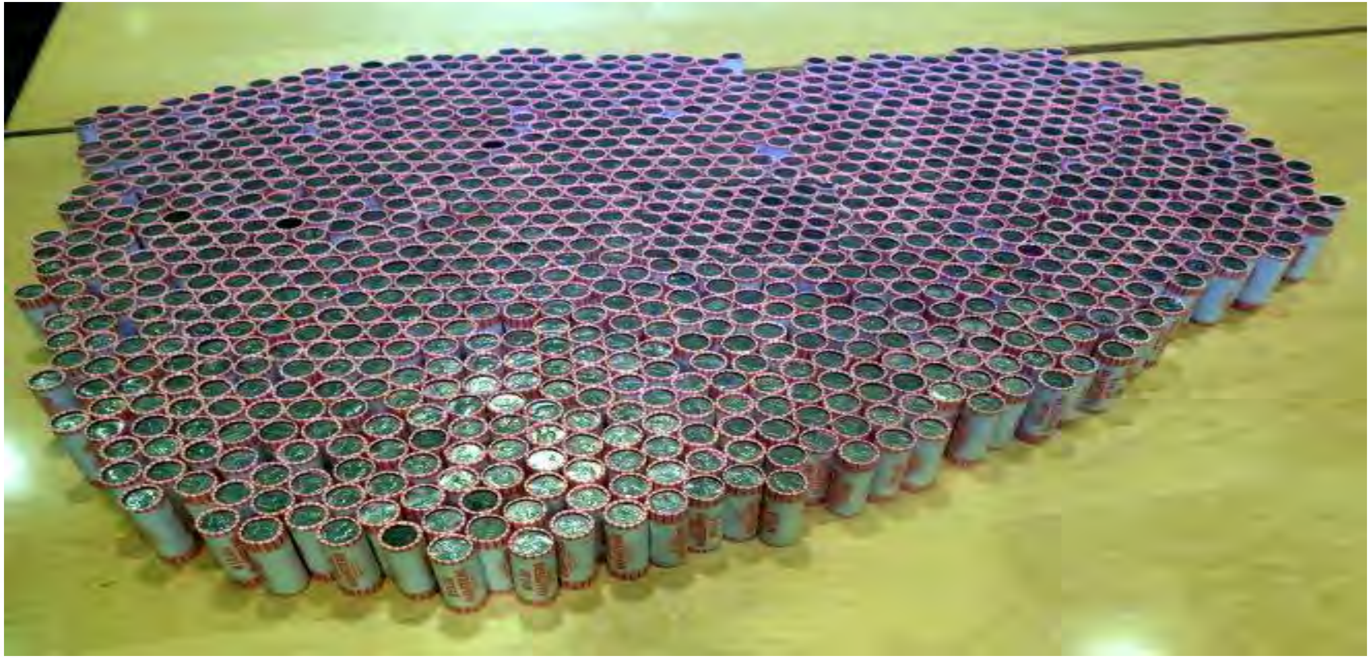
Breaking a large problem into sub-solutions



Simple Algorithm

1. Review stack of quarters
2. Count each year that ends in an even number

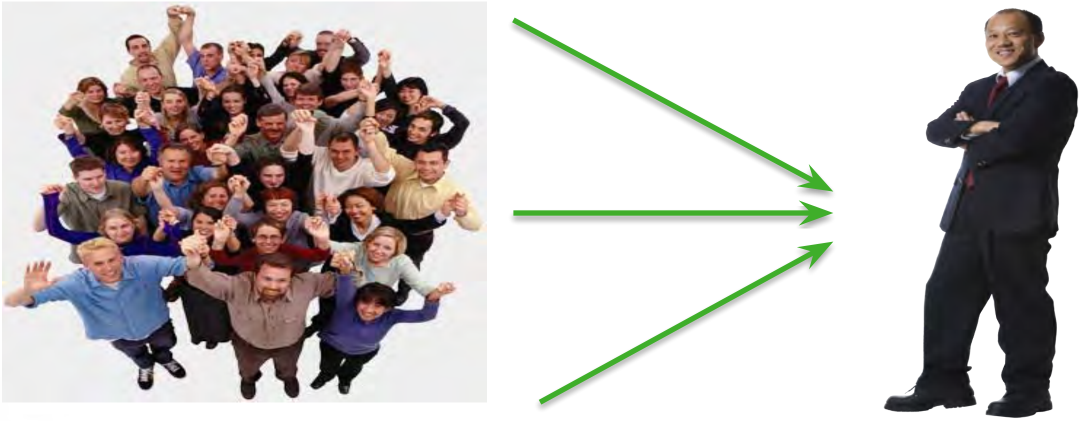
Processing at Scale



Distributed Algorithm – MapReduce

Map
(total number of quarters)

Reduce
(sum each person’s total)



The Mapper

- **The Mapper reads data in the form of key/value pairs (KVPs)**
- **It outputs zero or more KVPs**
- **The Mapper may use or completely ignore the input key**
  - For example, a standard pattern is to read a line of a file at a time
    - The key is the byte offset into the file at which the line starts
    - The value is the contents of the line itself
    - Typically the key is considered irrelevant with this pattern
- **If the Mapper writes anything out, it must in the form of KVPs**
  - This “intermediate data” is NOT stored in HDFS (local storage only without replication)

MapReduce Example – Map Phase

Input to Mapper

```
(8675, ‘I will not eat green eggs and ham’)
(8709, ‘I will not eat them Sam I am’)
…
```

- Ignoring the key
  - It is just and offset

Output from Mapper

```
(‘I’, 1), (‘will’, 1), (‘not’, 1), (‘eat’, 1),
(‘green’, 1),
(‘eggs’, 1), (‘and’, 1),
(‘ham’, 1), (‘I’, 1), (‘will’, 1),
(‘not’, 1), (‘eat’, 1),
(‘them’, 1), (‘Sam’, 1),
(‘I’, 1), (‘am’, 1)
```

- In this example
  - The size of the output > size of the input
  - No attempt is made to optimize within a record in this example
    - This is a great use case for a “Combiner”

The Shuffle

- After the Map phase is over, all the outputs from the mappers are sent to reducers
- KVPs with the same key will be sent to the same reducer
  - By default (k,v) will be sent to the reducer number hash(k) % numReducers
- **This can potentially generate a lot of network traffic on your cluster**
  - In our word count example the size of the output data is of the same order of magnitude as our input data
- Some very common operations like join, or group by require a lot of shuffle by design
- Optimizing these operations is an important part of mastering distributed processing programming
- CPU and RAM can scale by adding worker nodes, network can't

MapReduce Example – Reduce Phase

Input to Reducer

```
(‘I’, [1, 1, 1])
(‘Sam’, [1])
(‘am’, [1])
(‘and’, [1])
(‘eat’, [1, 1])
(‘eggs’, [1])
(‘green’, [1])
(‘ham’, [1])
(‘not’, [1, 1])
(‘them’, [1])
(‘will’, [1, 1])
```

- Notice keys are sorted and associated values for same key are in a single list
  - Shuffle & Sort did this for us

Output from Reducer

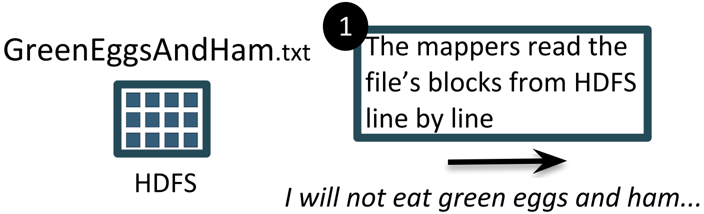
```
(‘I’, 3)
(‘Sam’, 1)
(‘am’, 1)
(‘and’, 1)
(‘eat’, 2)
(‘eggs’, 1)
(‘green’, 1)
(‘ham’, 1)
(‘not’, 2)
(‘them’, 1)
(‘will’, 2)
```

- All done!

The Reducer

- **After the Shuffle phase is over, all the intermediate values for a given intermediate key are sorted and combined together into a list**
- **This list is given to a Reducer**
  - There may be a single Reducer, or multiple Reducers
  - All values associated with a particular intermediate key are guaranteed to go to the same Reducer
  - The intermediate keys, and their value lists, are passed in sorted order
- **The Reducer outputs zero or more KVPs**
  - These are written to HDFS
  - In practice, the Reducer often emits a single KVP for each input key

Recap of Word Count



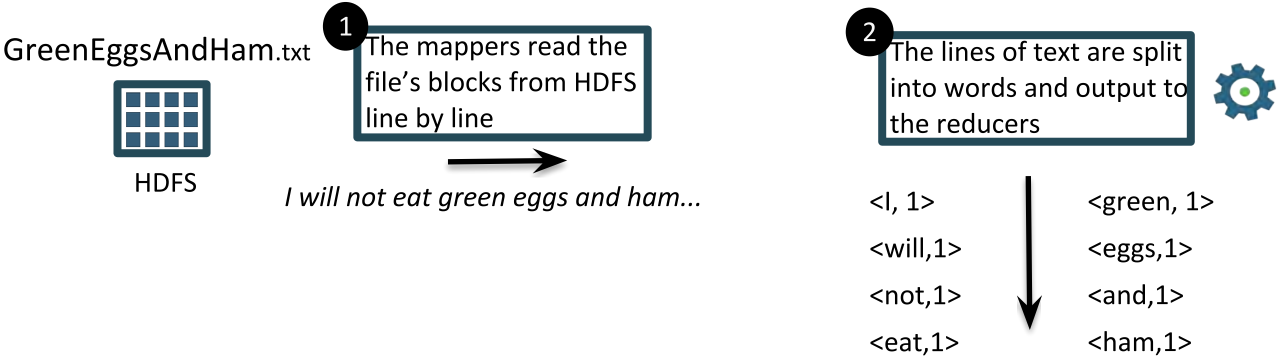
GreenEggsAndHam.txt

HDFS

1 The mappers read the file's blocks from HDFS line by line

I will not eat green eggs and ham...

Recap of Word Count



GreenEggsAndHam.txt

HDFS

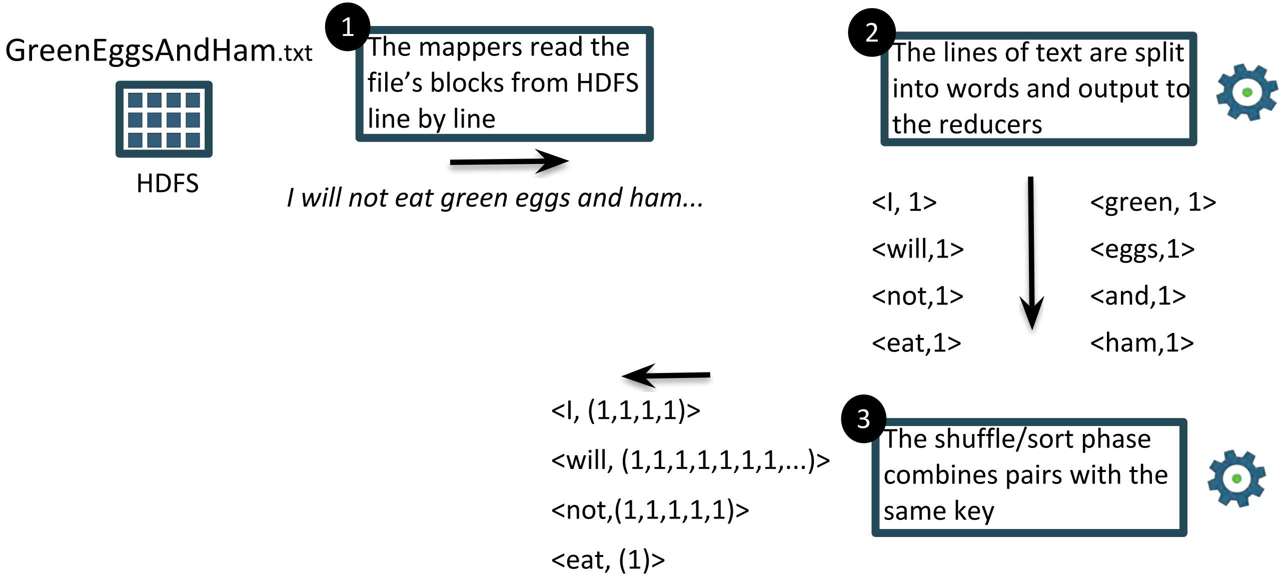
1 The mappers read the file's blocks from HDFS line by line

I will not eat green eggs and ham…

2 The lines of text are split into words and output to the reducers

<I, 1>
<will,1>
<not,1>
<eat,1>
<green, 1>
<eggs,1>
<and,1>
<ham,1>

Recap of Word Count



GreenEggsAndHam.txt

HDFS

1 The mappers read the file’s blocks from HDFS line by line

I will not eat green eggs and ham...

2 The lines of text are split into words and output to the reducers

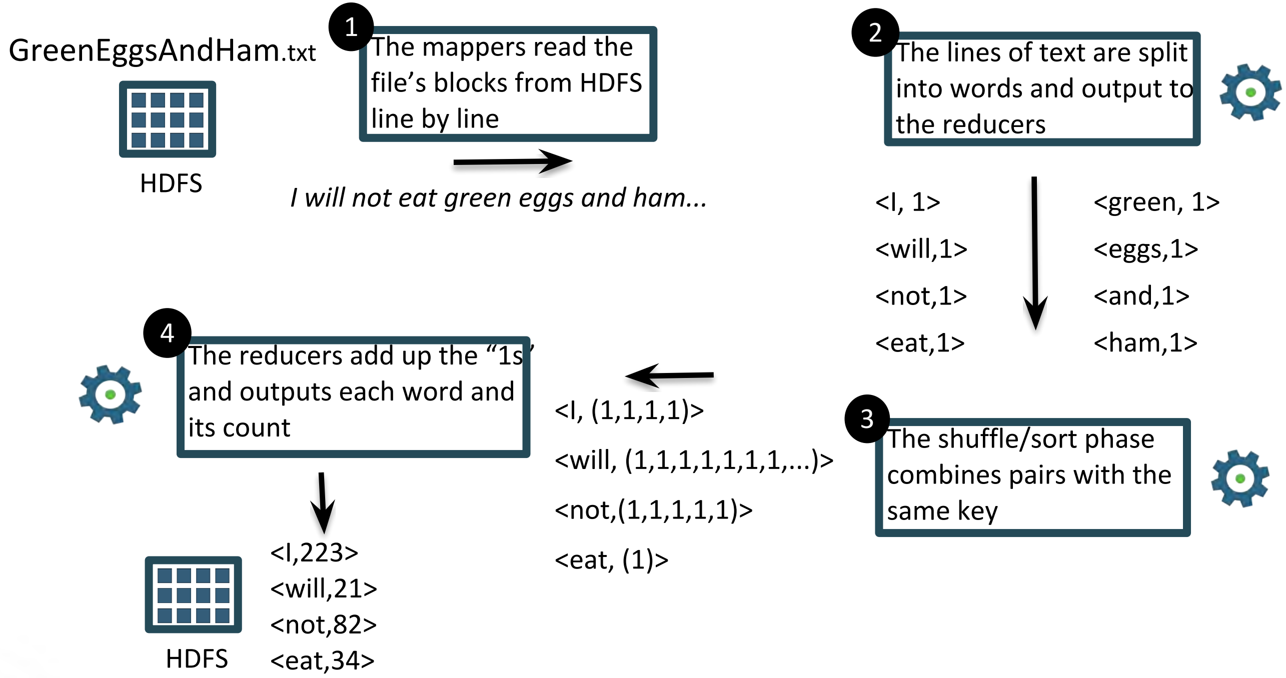
<I, 1>
<will,1>
<not,1>
<eat,1>

<green, 1>
<eggs,1>
<and,1>
<ham,1>

3 The shuffle/sort phase combines pairs with the same key

<I, (1,1,1,1)>
<will, (1,1,1,1,1,1,1,...)>
<not,(1,1,1,1,1)>
<eat, (1)>

Recap of Word Count



GreenEggsAndHam.txt

HDFS

1 The mappers read the file’s blocks from HDFS line by line

I will not eat green eggs and ham...

2 The lines of text are split into words and output to the reducers

<I, 1>
<will,1>
<not,1>
<eat,1>

<green, 1>
<eggs,1>
<and,1>
<ham,1>

3 The shuffle/sort phase combines pairs with the same key

<I, (1,1,1,1)>
<will, (1,1,1,1,1,1,1,...)>
<not,(1,1,1,1,1)>
<eat, (1)>

4 The reducers add up the “1s” and outputs each word and its count

<I,223>
<will,21>
<not,82>
<eat,34>

HDFS

MapReduce Example – Word Count

The mapper

```
public class WordCount {

  public static class TokenizerMapper
       extends Mapper<Object, Text, Text, IntWritable>{

    private final static IntWritable one = new IntWritable(1);
    private Text word = new Text();

    public void map(Object key, Text value, Context context
                    ) throws IOException, InterruptedException {
      StringTokenizer itr = new StringTokenizer(value.toString());
      while (itr.hasMoreTokens()) {
        word.set(itr.nextToken());
        context.write(word, one);
      }
    }
  }
```

MapReduce Example – Word Count

The reducer

```
public static class IntSumReducer
      extends Reducer<Text,IntWritable,Text,IntWritable> {
   private IntWritable result = new IntWritable();

   public void reduce(Text key, Iterable<IntWritable> values,
                      Context context
                      ) throws IOException, InterruptedException {
     int sum = 0;
     for (IntWritable val : values) {
       sum += val.get();
     }
     result.set(sum);
     context.write(key, result);
   }
 }
```

MapReduce Example – Word Count

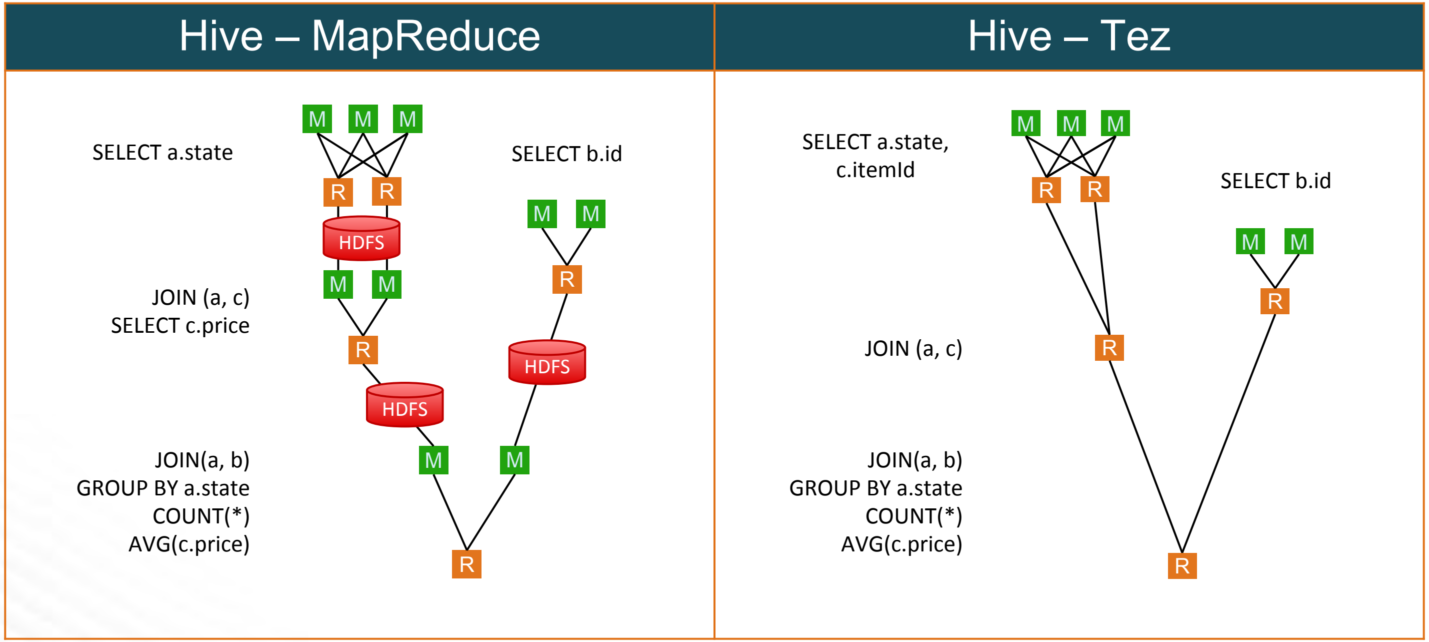
The main

```
public static void main(String[] args) throws Exception {
   Configuration conf = new Configuration();
   Job job = Job.getInstance(conf, "word count");
   job.setJarByClass(WordCount.class);
   job.setMapperClass(TokenizerMapper.class);
   job.setCombinerClass(IntSumReducer.class);
   job.setReducerClass(IntSumReducer.class);
   job.setOutputKeyClass(Text.class);
   job.setOutputValueClass(IntWritable.class);
   FileInputFormat.addInputPath(job, new Path(args[0]));
   FileOutputFormat.setOutputPath(job, new Path(args[1]));
   System.exit(job.waitForCompletion(true) ? 0 : 1);
 }
```

‘Complex’ processing required chaining MapReduce jobs

```
SELECT a.state, COUNT(*), AVG(c.price) FROM a
JOIN b ON (a.id = b.id)
JOIN c ON (a.itemId = c.itemId)
GROUP BY a.state
```

Tez avoids unneeded writes to HDFS



Knowledge Check

Questions

1. What are the two primary phases of the MapReduce framework? *No, this is not a trick question.*

Questions

1. What are the two primary phases of the MapReduce framework? *No, this is not a trick question.*
2. What dictates the number of Mappers that are run? Same question for the Reducers.

Questions

1. What are the two primary phases of the MapReduce framework? *No, this is not a trick question.*
2. What dictates the number of Mappers that are run? Same question for the Reducers.
3. How many input & output KVPs are passed into, and emitted out of, the Mappers? Same question for the Reducers.

Questions

1. What are the two primary phases of the MapReduce framework? *No, this is not a trick question.*
2. What dictates the number of Mappers that are run? Same question for the Reducers.
3. How many input & output KVPs are passed into, and emitted out of, the Mappers? Same question for the Reducers.
4. True/False? It is possible to have a Reducer-only job.

Questions

1. What are the two primary phases of the MapReduce framework? *No, this is not a trick question.*
2. What dictates the number of Mappers that are run? Same question for the Reducers.
3. How many input & output KVPs are passed into, and emitted out of, the Mappers? Same question for the Reducers.
4. True/False? It is possible to have a Reducer-only job.
5. Why were frameworks like Pig and Hive built on top of MapReduce? *Again, not a trick question…*

Summary

Summary

- MapReduce is the foundational framework for processing data at scale because of its ability to break a large problem into any smaller ones
- Mappers read data in the form of KVPs and each call to a Mapper is for a single KVP; it can return 0..m KVPs
- The framework shuffles & sorts the Mappers' outputted KVPs with the guarantee that only one Reducer will be asked to process a given Key's data
- Reducers are given a list of Values for a specific Key; they can return 0..m KVPs
- Due to the fine-grained nature of the framework, many use cases are better suited for higher-order tools