Introduction to Apache Spark

What is Hadoop??

What is Apache Spark??

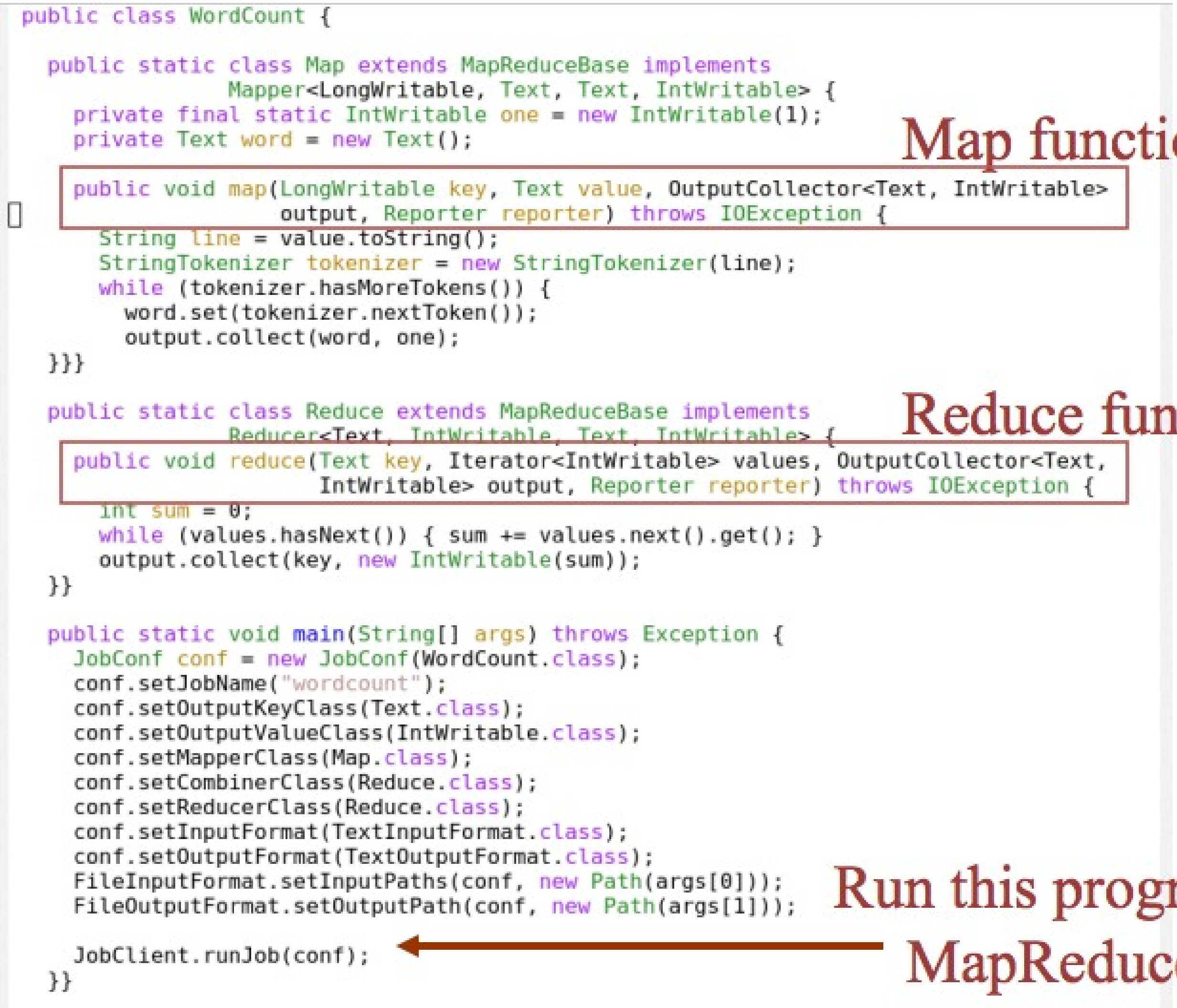
Some History

- Early 2000s - Google needed tools to process very large amounts of data
  - Google File System (GFS)
  - Map Reduce: https://research.google/pubs/pub62/
  - Google BigTable
- On publishing papers about these technologies, development of open-source implementations was taken up by Yahoo, Cloudera, Hortonworks and others
- These projects were eventually donated to the Apache Software Foundation

Some History

- Map Reduce was a ground-breaking programming paradigm for parallel computing
- However, the Hadoop implementation had some drawbacks
  - Very verbose code – lots of boilerplate required
  - Complex for developers to write Jobs
  - Not very fault-tolerant
  - Slow
  - Heavily reliant on disk I/O

Map-Reduce Java



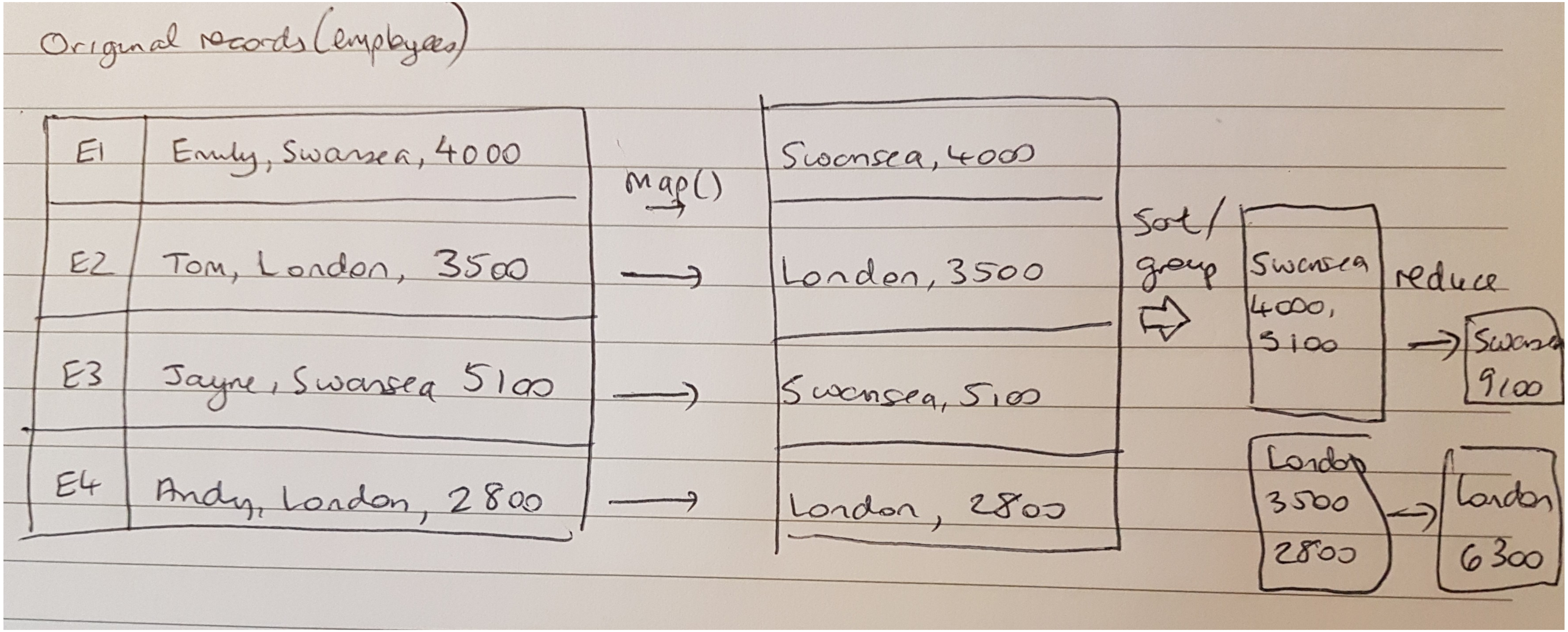
```
public class WordCount {

  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);
  }}}

  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));
  }}

  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);
  }}
```

Map function

Reduce function

Run this program as a MapReduce job

Let's take a simple example



Original records (employees)

E1	Emily, Swansea, 4000
E2	Tom, London, 3500
E3	Jayne, Swansea 5100
E4	Andy, London, 2800

Swansea, 4000
London, 3500
Swansea, 5100
London, 2800

Let's explore this in Python!

Hadoop Core

- Where does the name "Hadoop" come from?

Hadoop Core Principles (1)

- Hadoop was one of the first open-source big data technologies
  - Scalable, fault-tolerant system for processing large datasets…
  - Across a cluster of commodity servers

- Hadoop provides high availability and fault tolerance
  - You don't need to buy expensive hardware
  - Hadoop is well suited for batch processing and ETL (extract transform load) of large-scale data

- Many organizations replaced expensive commercial products with Hadoop
  - Cost benefits - Hadoop is open source, runs on commodity h/w
  - Easily scalable - just add some more (relatively cheap) servers

Hadoop Core Principles (2)

- Hadoop uses a cluster of commodity servers for storing and processing large amounts of data
  - Cheaper than using high-end powerful servers
  - Hadoop uses a scale-out architecture (rather than scale-up)

- Hadoop is designed to work best with a relatively small number of huge files
  - Commonly ,the average file size in Hadoop is > 500MB

Hadoop Core Principles (3)

- Hadoop implements fault tolerance through software
  - Cheaper than implementing fault tolerance through hardware
  - Hadoop doesn't rely on fault-tolerant servers
  - Hadoop assumes servers fail, and transparently handles failures

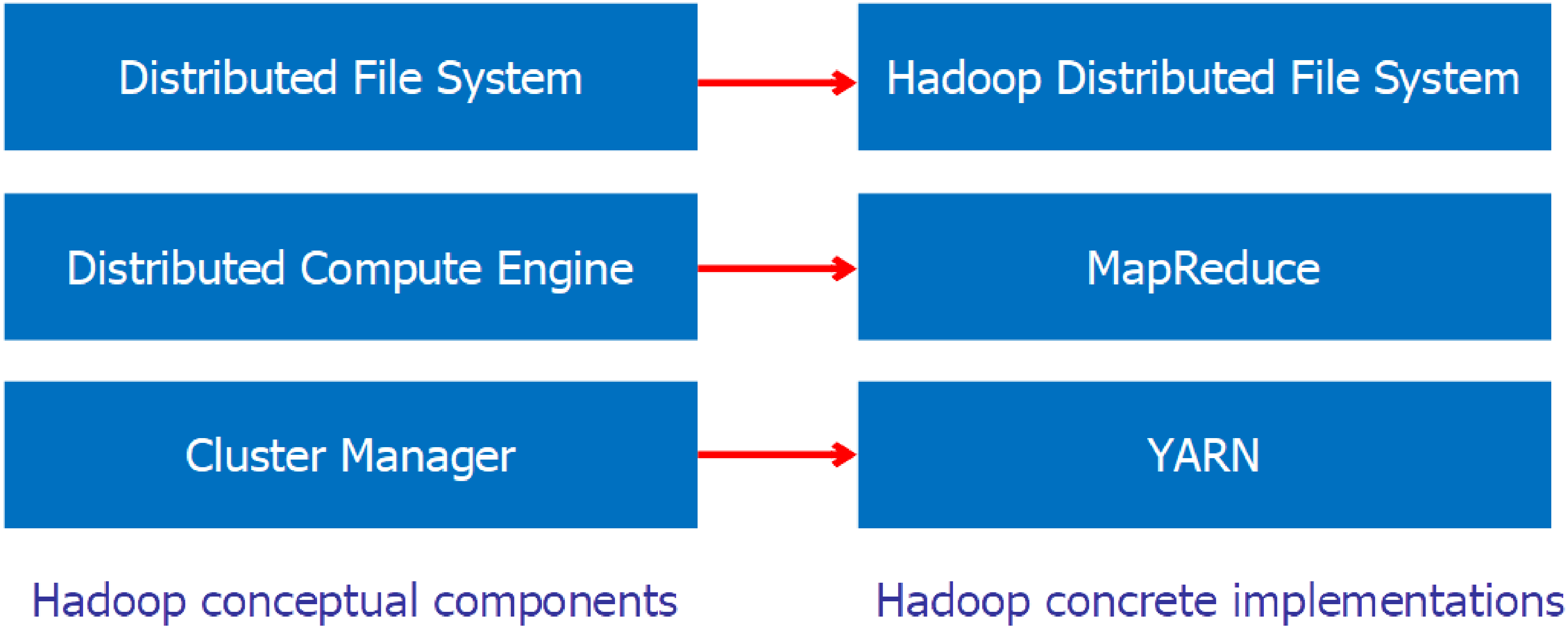
- Developers don't need to worry about handling hardware failures
  - You can leave Hadoop to handle these messy details

Hadoop Core Principles (4)

- Moving code from one computer to another is much faster and more efficient than moving large datasets
  - E.g. imagine you have a cluster of 50 computers with 1TB of data on each computer - what are the options for processing this data?

Hadoop Core Components

- Hadoop isn't really a single product, it's an eco-system
  - At its heart are three key components...



Distributed File System → Hadoop Distributed File System

Distributed Compute Engine → MapReduce

Cluster Manager → YARN

Hadoop conceptual components

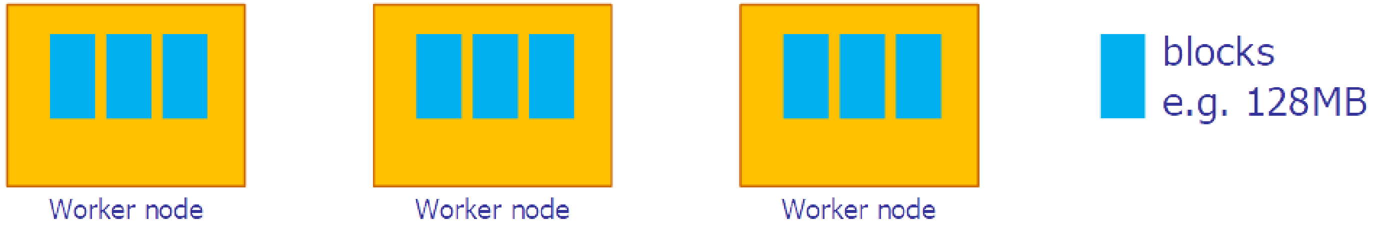
Hadoop concrete implementations

Hadoop Distributed File System (2)

- HDFS is a scalable and fault-tolerant distributed file system
  - Stores a file across a cluster of commodity servers (e.g. 1000s)
  - Aim: to store and allow fast access to big files and large datasets

- HDFS is a block-structured file system
  - Splits a file into fixed-size opaque blocks, aka partitions or slices
  - Default block size 128MB (c.f. ~4KB block size on Linux)

- HDFS spreads file blocks across "worker node" machines
  - Allows file read/write operations to be massively parallelized

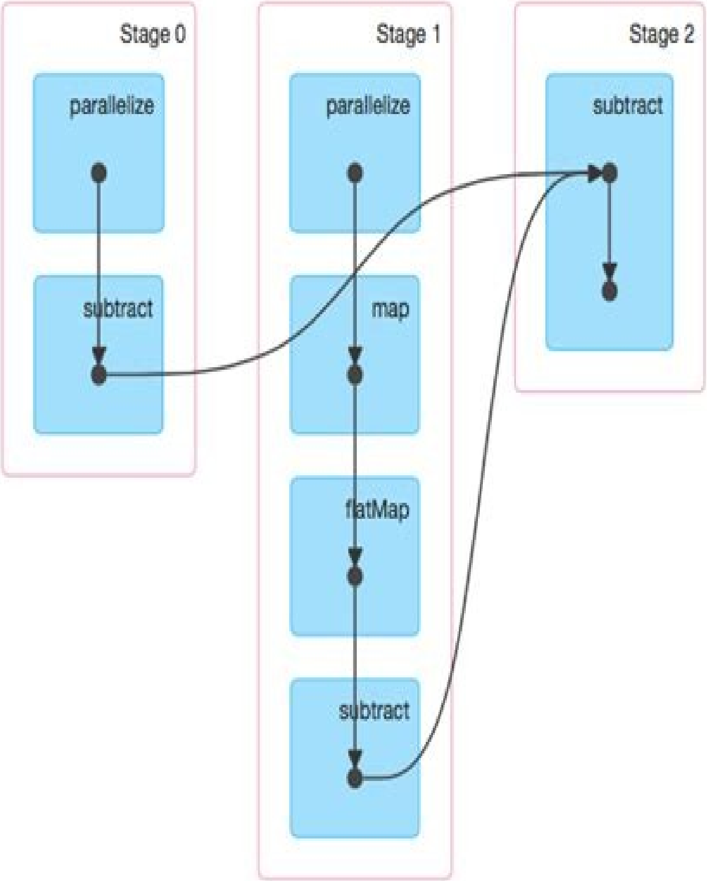


Why Spark?

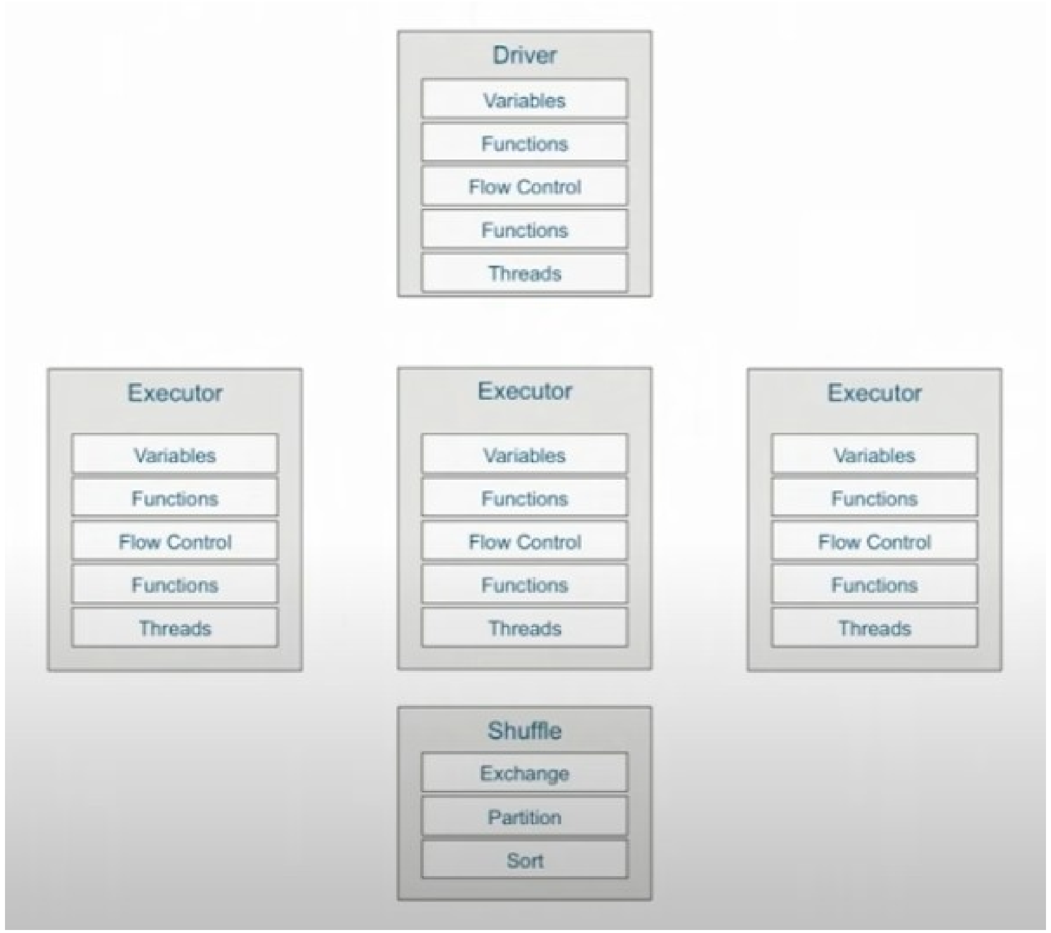
- Spark grew out of the need to have a simpler, faster, more robust way to program with parallelism
- Research groups in UC Berkeley began working on this, with some guiding principles
  - Highly Fault Tolerant
  - 100% Parallel
  - In-memory Intermediate results
  - Easy API
  - Program in multiple languages – e.g. Java, Scala, Python, R

Apache Spark

- "A unified engine for large-scale data analytics"
- Based on the concept of an "RDD"
  - Resilient
  - Distributed
  - Dataset
- Spark creates a Directed Acyclic Graph (DAG) for a job
- Jobs are written through higher-level APIs



Spark is "similar" to MR

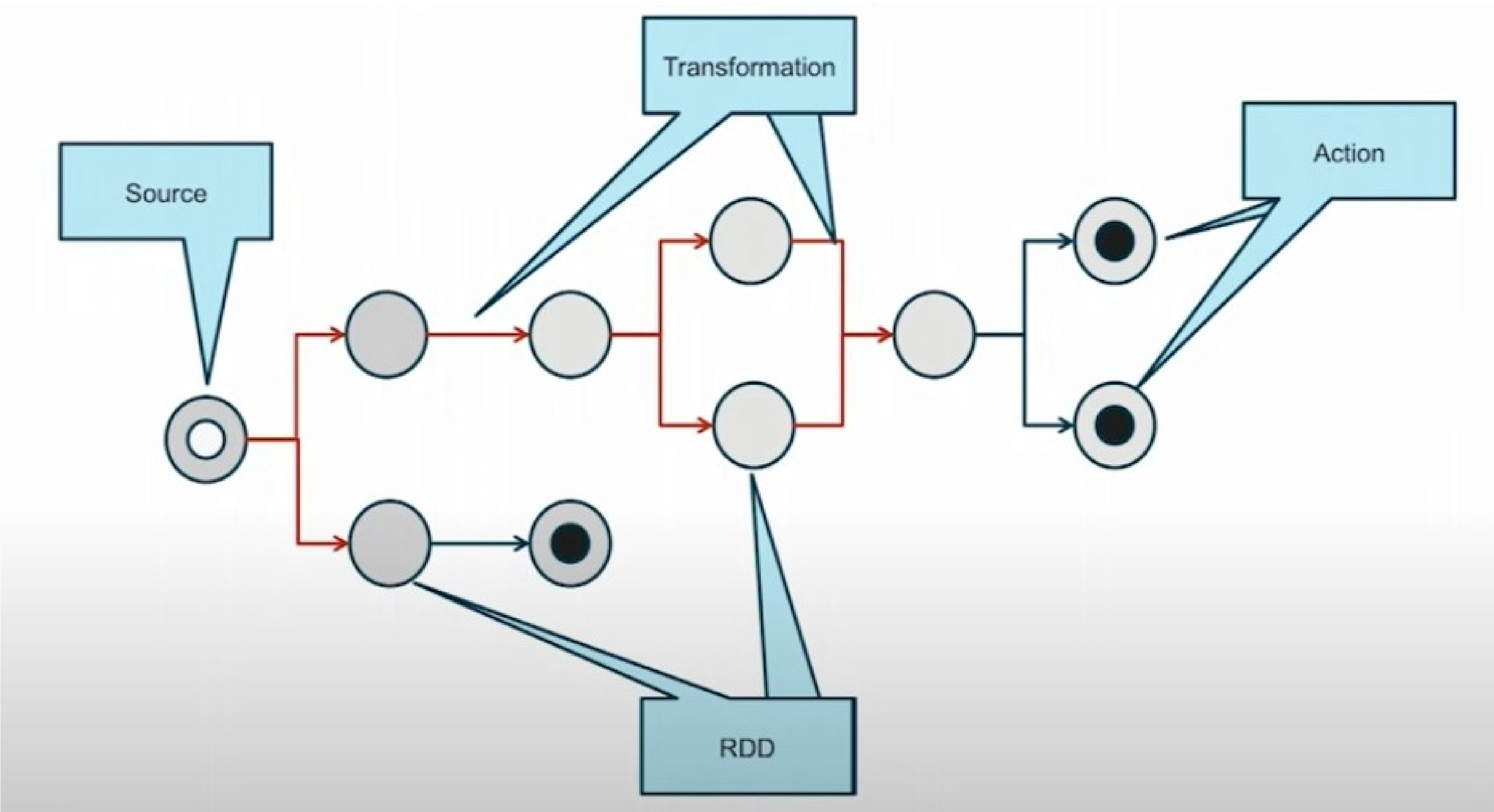


Spark & DAG

- DAG is a finite direct graph with no directed cycles. There are finitely many *vertices* and *edges*
- *vertices* represent the **RDDs** and the *edges* represent the **Operation to be applied on RDD**
- With the original Hadoop MR framework, the programmer would effectively "write" the DAG in his code
- Frameworks like Apache Hive, PIG & Impala, gave a high-level API "on-top" of MR. These tools would create the DAG based on high level instructions

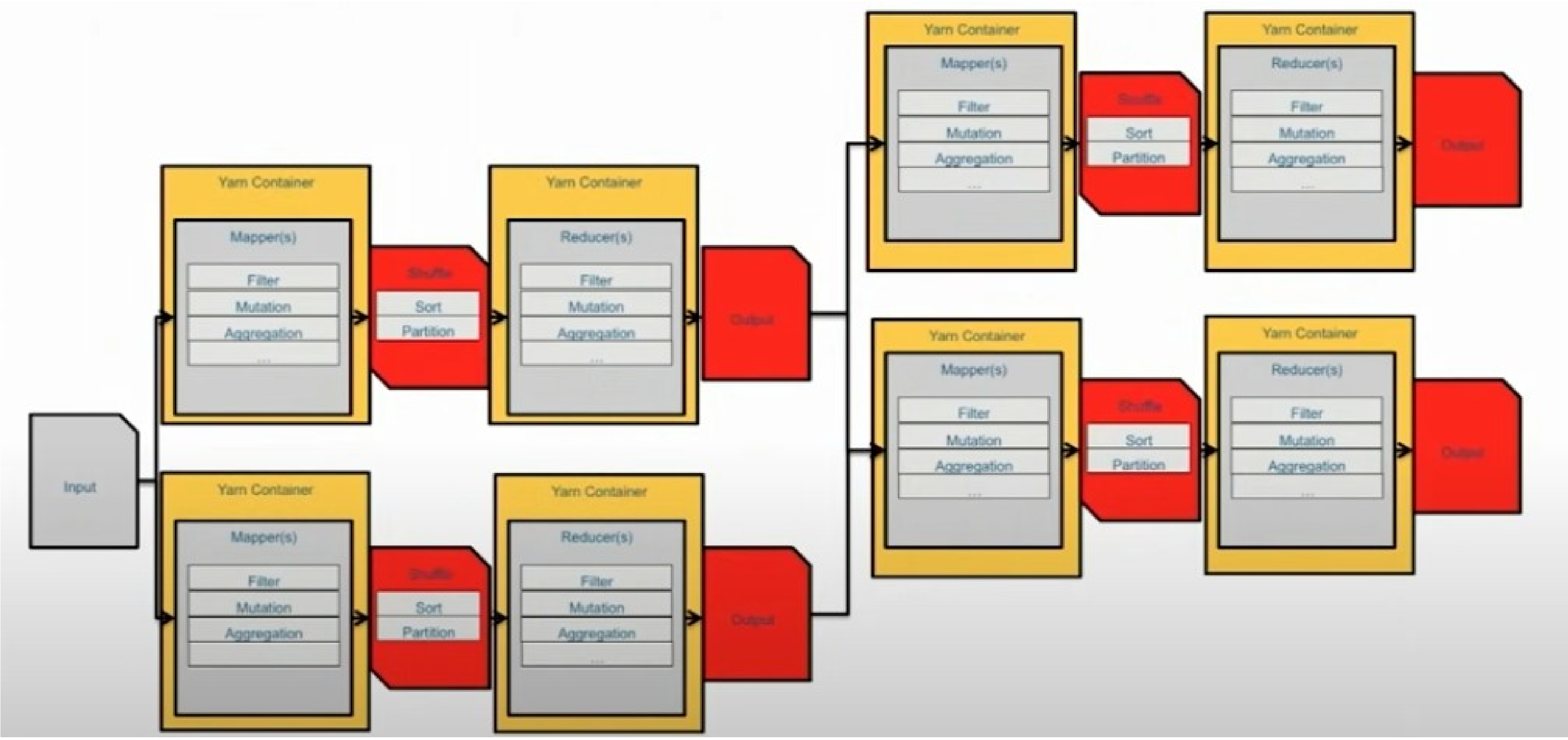
Spark & DAG

- RDDs are **Resilient**
- The DAG contains the instructions to recreate any intermediate RDD



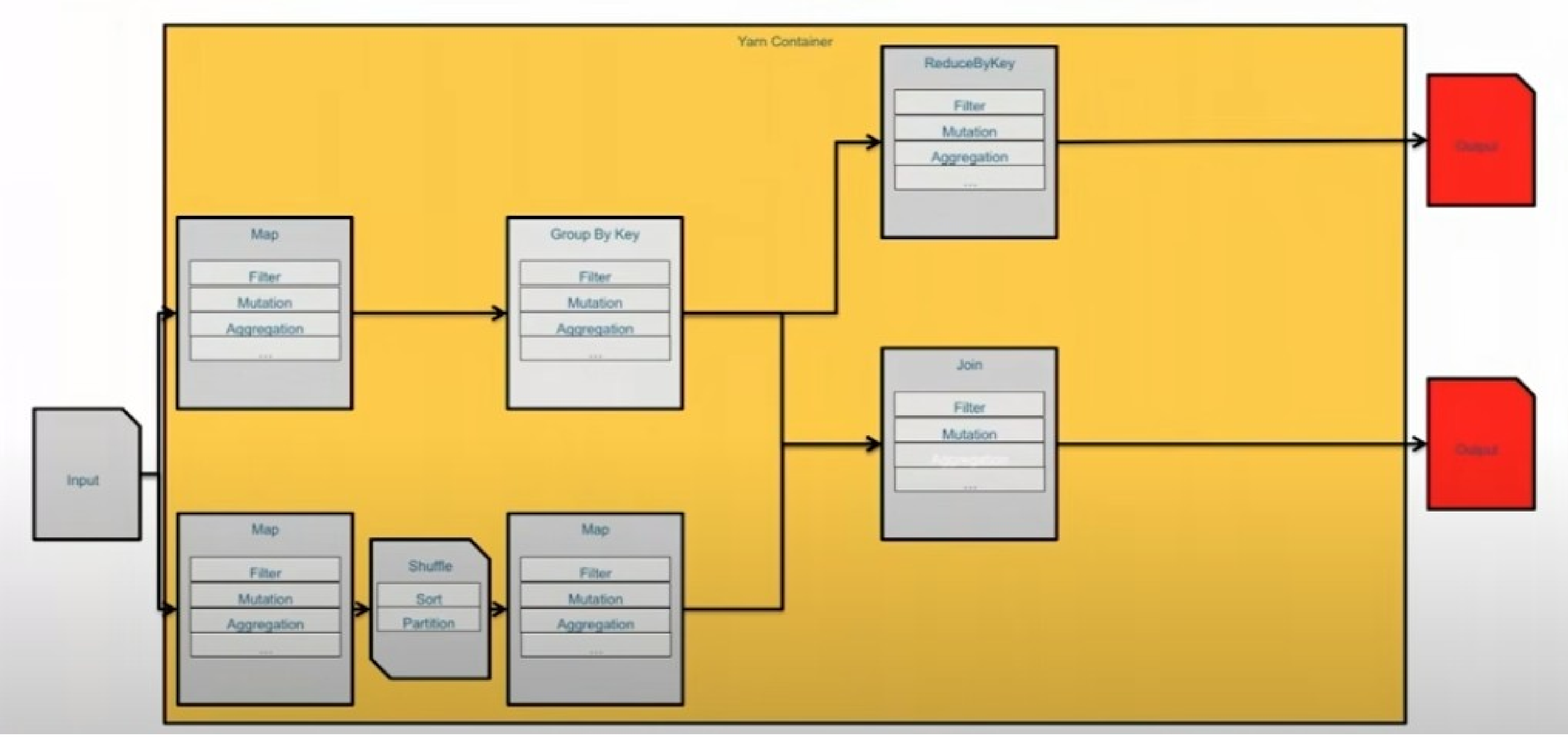
Spark Vs MR

- With MR, intermediate RDDs are saved to disk



Spark Vs MR

- Spark tries to keep intermediate RDDs in memory
- Interactive Sessions
- Long-running jobs
- Streaming Applications



Spark Examples

- Let's take our first steps with Spark in Python & Scala

- References: https://github.com/fcallaly/spark-intro-examples

Questions?