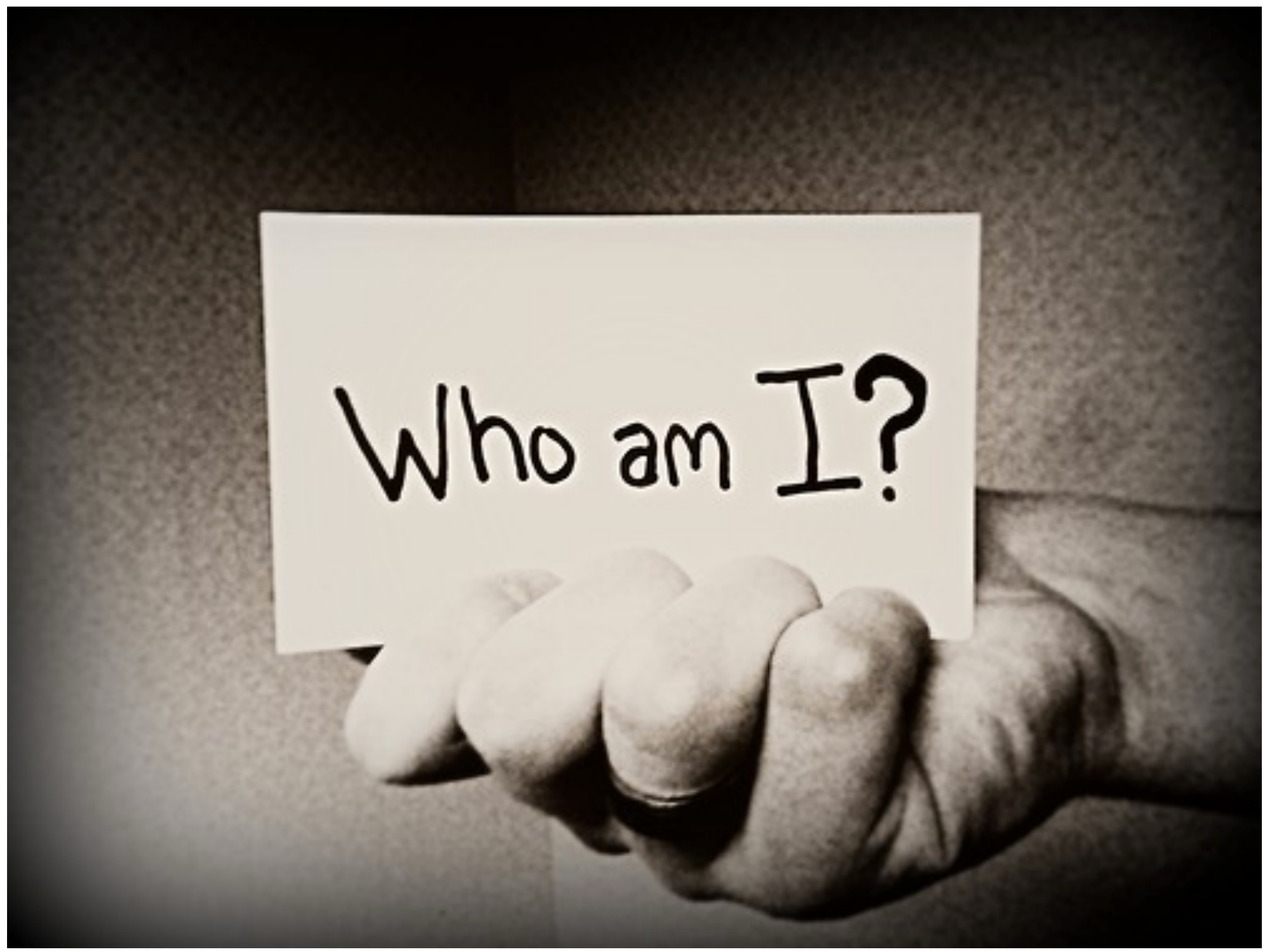
Spark Runtime Internal

Nan Zhu (McGill University & Faimdata)


Who am I?

WHO AM I

- Nan Zhu, PhD Candidate in School of Computer Science of McGill University
    - Work on computer networks (Software Defined Networks) and large-scale data processing
    - Work with Prof. Wenbo He and Prof. Xue Liu
    - PhD is an awesome experience in my life
        - Tackle real world problems
        - Keep thinking ! Get insights !

WHO AM I

- Nan Zhu, PhD Candidate in School of Computer Science of McGill University
  - Work on computer networks (Software Defined Networks) and large-scale data processing
  - Work with Prof. Wenbo He and Prof. Xue Liu

When will I graduate ?

  - PhD is an awesome experience in my life
    - Tackle real world problems
    - Keep thinking ! Get insights !

WHO AM I

- Do-it-all Engineer in Faimdata (http://www.faimdata.com)
- Faimdata is a new startup located in Montreal
    - Build Customer-centric analysis solution based on Spark for retailers
- My responsibility
    - Participate in everything related to data
    - Akka, HBase, Hive, Kafka, Spark, etc.

FAiMDATA

WHO AM I

- My Contribution to Spark
  - 0.8.1, 0.9.0, 0.9.1, 1.0.0
  - 1000+ code, 30 patches
    - Two examples:
      - YARN-like architecture in Spark
      - Introduce Actor Supervisor mechanism to DAGScheduler

WHO AM I

- My Contribution to Spark
  - 0.8.1, 0.9.0, 0.9.1, 1.0.0
  - 1000+ code, 30 patches

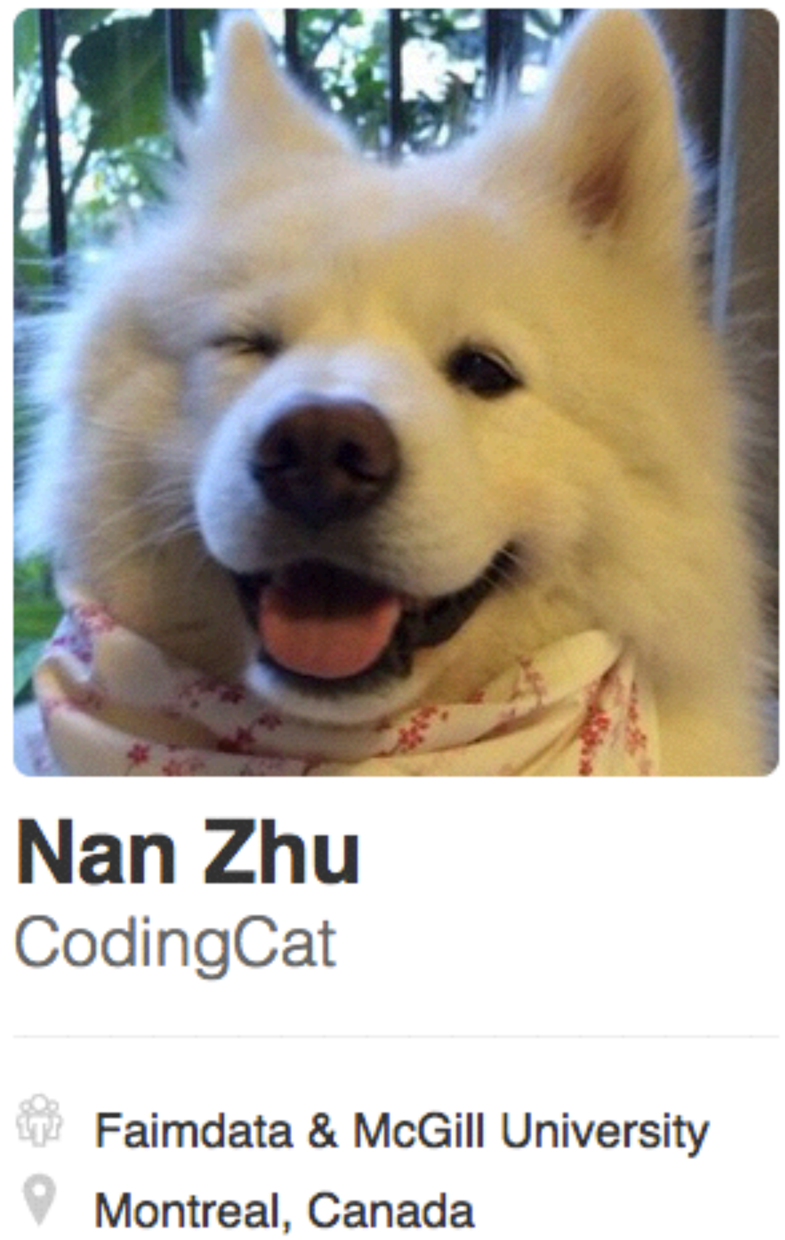
I’m CodingCat@GitHub !!!!

  - Two examples:
    - YARN-like architecture in Spark
    - Introduce Actor Supervisor mechanism to DAGScheduler

WHO AM I

- My Contribution
  - 0.8.1, 0.9.0, (
  - 1000+ code,
    - Two examp
      - YARN-lik   rk
      - Introduc   echanism to DAGScheduler

I'm Cod   GitHub !!!!



What is Spark?

What is Spark?

- A distributed computing framework
    - Organize computation as concurrent tasks
    - Schedule tasks to multiple servers
    - Handle fault-tolerance, load balancing, etc, in automatic (and transparently)

Advantages of Spark

- More Descriptive Computing Model
- Faster Processing Speed
- Unified Pipeline

Mode Descriptive Computing Model (1)

- WordCount in Hadoop (Map & Reduce)

```
public class WordCount {

 public static class Map extends 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, Context context) throws IOException, InterruptedException {
        String line = value.toString();
        StringTokenizer tokenizer = new StringTokenizer(line);
        while (tokenizer.hasMoreTokens()) {
            word.set(tokenizer.nextToken());
            context.write(word, one);
        }
    }
 }

 public static class Reduce extends Reducer<Text, IntWritable, Text, IntWritable> {

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

 public static void main(String[] args) throws Exception {
    Configuration conf = new Configuration();

        Job job = new Job(conf, "wordcount");

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

    job.setMapperClass(Map.class);
    job.setReducerClass(Reduce.class);

    job.setInputFormatClass(TextInputFormat.class);
    job.setOutputFormatClass(TextOutputFormat.class);

    FileInputFormat.addInputPath(job, new Path(args[0]));
    FileOutputFormat.setOutputPath(job, new Path(args[1]));

    job.waitForCompletion(true);
 }

}
```

Mode Descriptive Computing Model (1)

- WordCount in Hadoop (Map & Reduce)

```
public class WordCount {

  public static class Map extends 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, Context context) throws IOException, InterruptedException {
         String line = value.toString();
         StringTokenizer tokenizer = new StringTokenizer(line);
         while (tokenizer.hasMoreTokens()) {
             word.set(tokenizer.nextToken());
             context.write(word, one);
         }
     }
  }

  public static class Reduce extends Reducer<Text, IntWritable, Text, IntWritable> {

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

  public static void main(String[] args) throws Exception {
     Configuration conf = new Configuration();

         Job job = new Job(conf, "wordcount");

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

     job.setMapperClass(Map.class);
     job.setReducerClass(Reduce.class);

     job.setInputFormatClass(TextInputFormat.class);
     job.setOutputFormatClass(TextOutputFormat.class);

     FileInputFormat.addInputPath(job, new Path(args[0]));
     FileOutputFormat.setOutputPath(job, new Path(args[1]));

     job.waitForCompletion(true);
  }

}
```

Mode Descriptive Computing Model (1)

- WordCount in Hadoop (Map & Reduce)

Map function, read each line of the input file and transform each word into <word, 1> pair

```
public class WordCount {

  public static class Map extends 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, Context context) throws IOException, InterruptedException {
         String line = value.toString();
         StringTokenizer tokenizer = new StringTokenizer(line);
         while (tokenizer.hasMoreTokens()) {
             word.set(tokenizer.nextToken());
             context.write(word, one);
         }
     }
  }

  public static class Reduce extends Reducer<Text, IntWritable, Text, IntWritable> {

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

  public static void main(String[] args) throws Exception {
     Configuration conf = new Configuration();

         Job job = new Job(conf, "wordcount");

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

     job.setMapperClass(Map.class);
     job.setReducerClass(Reduce.class);

     job.setInputFormatClass(TextInputFormat.class);
     job.setOutputFormatClass(TextOutputFormat.class);

     FileInputFormat.addInputPath(job, new Path(args[0]));
     FileOutputFormat.setOutputPath(job, new Path(args[1]));

     job.waitForCompletion(true);
  }

}
```

Mode Descriptive Computing Model (1)

- WordCount in Hadoop (Map & Reduce)

Map function, read each line of the input file and transform each word into <word, 1> pair

```java
public class WordCount {

  public static class Map extends 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, Context context) throws IOException, InterruptedException {
         String line = value.toString();
         StringTokenizer tokenizer = new StringTokenizer(line);
         while (tokenizer.hasMoreTokens()) {
             word.set(tokenizer.nextToken());
             context.write(word, one);
         }
     }
  }

  public static class Reduce extends Reducer<Text, IntWritable, Text, IntWritable> {

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

  public static void main(String[] args) throws Exception {
     Configuration conf = new Configuration();

         Job job = new Job(conf, "wordcount");

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

     job.setMapperClass(Map.class);
     job.setReducerClass(Reduce.class);

     job.setInputFormatClass(TextInputFormat.class);
     job.setOutputFormatClass(TextOutputFormat.class);

     FileInputFormat.addInputPath(job, new Path(args[0]));
     FileOutputFormat.setOutputPath(job, new Path(args[1]));

     job.waitForCompletion(true);
  }

}
```

Mode Descriptive Computing Model (1)

- WordCount in Hadoop (Map & Reduce)

Map function, read each line of the input file and transform each word into <word, 1> pair

```java
public class WordCount {

   public static class Map extends 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, Context context) throws IOException, InterruptedException {
          String line = value.toString();
          StringTokenizer tokenizer = new StringTokenizer(line);
          while (tokenizer.hasMoreTokens()) {
              word.set(tokenizer.nextToken());
              context.write(word, one);
          }
      }
   }

   public static class Reduce extends Reducer<Text, IntWritable, Text, IntWritable> {

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

   public static void main(String[] args) throws Exception {
      Configuration conf = new Configuration();

          Job job = new Job(conf, "wordcount");

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

      job.setMapperClass(Map.class);
      job.setReducerClass(Reduce.class);

      job.setInputFormatClass(TextInputFormat.class);
      job.setOutputFormatClass(TextOutputFormat.class);

      FileInputFormat.addInputPath(job, new Path(args[0]));
      FileOutputFormat.setOutputPath(job, new Path(args[1]));

      job.waitForCompletion(true);
   }

}
```

Mode Descriptive Computing Model (1)

- WordCount in Hadoop (Map & Reduce)

Map function, read each line of the input file and transform each word into <word, 1> pair

Reduce function, collect the <word, 1> pairs generated by Map function and merge them by accumulation

```
public class WordCount {

  public static class Map extends 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, Context context) throws IOException, InterruptedException {
         String line = value.toString();
         StringTokenizer tokenizer = new StringTokenizer(line);
         while (tokenizer.hasMoreTokens()) {
             word.set(tokenizer.nextToken());
             context.write(word, one);
         }
     }
  }

  public static class Reduce extends Reducer<Text, IntWritable, Text, IntWritable> {

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

  public static void main(String[] args) throws Exception {
     Configuration conf = new Configuration();

         Job job = new Job(conf, "wordcount");

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

     job.setMapperClass(Map.class);
     job.setReducerClass(Reduce.class);

     job.setInputFormatClass(TextInputFormat.class);
     job.setOutputFormatClass(TextOutputFormat.class);

     FileInputFormat.addInputPath(job, new Path(args[0]));
     FileOutputFormat.setOutputPath(job, new Path(args[1]));

     job.waitForCompletion(true);
  }

}
```

Mode Descriptive Computing Model (1)

- WordCount in Hadoop (Map & Reduce)

Map function, read each line of the input file and transform each word into <word, 1> pair

Reduce function, collect the <word, 1> pairs generated by Map function and merge them by accumulation

```
public class WordCount {

  public static class Map extends 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, Context context) throws IOException, InterruptedException {
         String line = value.toString();
         StringTokenizer tokenizer = new StringTokenizer(line);
         while (tokenizer.hasMoreTokens()) {
             word.set(tokenizer.nextToken());
             context.write(word, one);
         }
     }
  }

  public static class Reduce extends Reducer<Text, IntWritable, Text, IntWritable> {

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

  public static void main(String[] args) throws Exception {
     Configuration conf = new Configuration();

         Job job = new Job(conf, "wordcount");

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

     job.setMapperClass(Map.class);
     job.setReducerClass(Reduce.class);

     job.setInputFormatClass(TextInputFormat.class);
     job.setOutputFormatClass(TextOutputFormat.class);

     FileInputFormat.addInputPath(job, new Path(args[0]));
     FileOutputFormat.setOutputPath(job, new Path(args[1]));

     job.waitForCompletion(true);
  }

}
```

Mode Descriptive Computing Model (1)

- WordCount in Hadoop (Map & Reduce)

Map function, read each line of the input file and transform each word into <word, 1> pair

Reduce function, collect the <word, 1> pairs generated by Map function and merge them by accumulation

```java
public class WordCount {

    public static class Map extends 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, Context context) throws IOException, InterruptedException {
            String line = value.toString();
            StringTokenizer tokenizer = new StringTokenizer(line);
            while (tokenizer.hasMoreTokens()) {
                word.set(tokenizer.nextToken());
                context.write(word, one);
            }
        }
    }

    public static class Reduce extends Reducer<Text, IntWritable, Text, IntWritable> {

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

    public static void main(String[] args) throws Exception {
        Configuration conf = new Configuration();

            Job job = new Job(conf, "wordcount");

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

        job.setMapperClass(Map.class);
        job.setReducerClass(Reduce.class);

        job.setInputFormatClass(TextInputFormat.class);
        job.setOutputFormatClass(TextOutputFormat.class);

        FileInputFormat.addInputPath(job, new Path(args[0]));
        FileOutputFormat.setOutputPath(job, new Path(args[1]));

        job.waitForCompletion(true);
    }

}
```

Mode Descriptive Computing Model (1)

- WordCount in Hadoop (Map & Reduce)

Map function, read each line of the input file and transform each word into <word, 1> pair

Reduce function, collect the <word, 1> pairs generated by Map function and merge them by accumulation

Configurate the program

```java
public class WordCount {

public static class Map extends 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, Context context) throws IOException, InterruptedException {
       String line = value.toString();
       StringTokenizer tokenizer = new StringTokenizer(line);
       while (tokenizer.hasMoreTokens()) {
           word.set(tokenizer.nextToken());
           context.write(word, one);
       }
   }
}

public static class Reduce extends Reducer<Text, IntWritable, Text, IntWritable> {

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

public static void main(String[] args) throws Exception {
   Configuration conf = new Configuration();

       Job job = new Job(conf, "wordcount");

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

   job.setMapperClass(Map.class);
   job.setReducerClass(Reduce.class);

   job.setInputFormatClass(TextInputFormat.class);
   job.setOutputFormatClass(TextOutputFormat.class);

   FileInputFormat.addInputPath(job, new Path(args[0]));
   FileOutputFormat.setOutputPath(job, new Path(args[1]));

   job.waitForCompletion(true);
}

}
```

Mode Descriptive Computing Model (1)

- WordCount in Hadoop (Map & Reduce)

Map function, read each line of the input file and transform each word into <word, 1> pair

Reduce function, collect the <word, 1> pairs generated by Map function and merge them by accumulation

Configurate the program

```java
public class WordCount {

  public static class Map extends 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, Context context) throws IOException, InterruptedException {
         String line = value.toString();
         StringTokenizer tokenizer = new StringTokenizer(line);
         while (tokenizer.hasMoreTokens()) {
             word.set(tokenizer.nextToken());
             context.write(word, one);
         }
     }
  }

  public static class Reduce extends Reducer<Text, IntWritable, Text, IntWritable> {

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

  public static void main(String[] args) throws Exception {
     Configuration conf = new Configuration();

         Job job = new Job(conf, "wordcount");

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

     job.setMapperClass(Map.class);
     job.setReducerClass(Reduce.class);

     job.setInputFormatClass(TextInputFormat.class);
     job.setOutputFormatClass(TextOutputFormat.class);

     FileInputFormat.addInputPath(job, new Path(args[0]));
     FileOutputFormat.setOutputPath(job, new Path(args[1]));

     job.waitForCompletion(true);
  }

}
```

DESCRIPTIVE COMPUTING MODEL (2)

- WordCount in Spark

Scala:

```
val file = spark.textFile("hdfs://...")
val counts = file.flatMap(line => line.split(" "))
                 .map(word => (word, 1))
                 .reduceByKey(_ + _)
counts.saveAsTextFile("hdfs://...")
```

Java:

```
JavaRDD<String> file = spark.textFile("hdfs://...");
JavaRDD<String> words = file.flatMap(new FlatMapFunction<String, String>() {
  public Iterable<String> call(String s) { return Arrays.asList(s.split(" ")); }
});
JavaPairRDD<String, Integer> pairs = words.map(new PairFunction<String, String, Integer>() {
  public Tuple2<String, Integer> call(String s) { return new Tuple2<String, Integer>(s, 1); }
});
JavaPairRDD<String, Integer> counts = pairs.reduceByKey(new Function2<Integer, Integer>() {
  public Integer call(Integer a, Integer b) { return a + b; }
});
counts.saveAsTextFile("hdfs://...");
```

DESCRIPTIVE COMPUTING MODEL (2)

- Closer look at WordCount in Spark

  Organize Computation into Multiple Stages in a Processing Pipeline:
  transformation to get the intermediate results with expected schema
  action to get final output

Scala:

```
val file = spark.textFile("hdfs://...")
val counts = file.flatMap(line => line.split(" "))
                 .map(word => (word, 1))
                 .reduceByKey(_ + _)
counts.saveAsTextFile("hdfs://...")
```

Computation is expressed with more high-level APIs, which simplify the logic in original Map & Reduce and define the computation as a processing pipeline

DESCRIPTIVE COMPUTING MODEL (2)

- Closer look at WordCount in Spark

  Organize Computation into Multiple Stages in a Processing Pipeline:
  transformation to get the intermediate results with expected schema
  action to get final output

Scala:

```
val file = spark.textFile("hdfs://...")
val counts = file.flatMap(line => line.split(" "))
                 .map(word => (word, 1))
                 .reduceByKey(_ + _)
counts.saveAsTextFile("hdfs://...")
```

Transformation

Computation is expressed with more high-level APIs, which simplify the logic in original Map & Reduce and define the computation as a processing pipeline

DESCRIPTIVE COMPUTING MODEL (2)

- Closer look at WordCount in Spark

  Organize Computation into Multiple Stages in a Processing Pipeline:
  transformation to get the intermediate results with expected schema
  action to get final output

Scala:

```scala
val file = spark.textFile("hdfs://...")
val counts = file.flatMap(line => line.split(" "))
                 .map(word => (word, 1))
                 .reduceByKey(_ + _)
counts.saveAsTextFile("hdfs://...")
```

Transformation

Action

Computation is expressed with more high-level APIs, which simplify the logic in original Map & Reduce and define the computation as a processing pipeline

MUCH BETTER PERFORMANCE

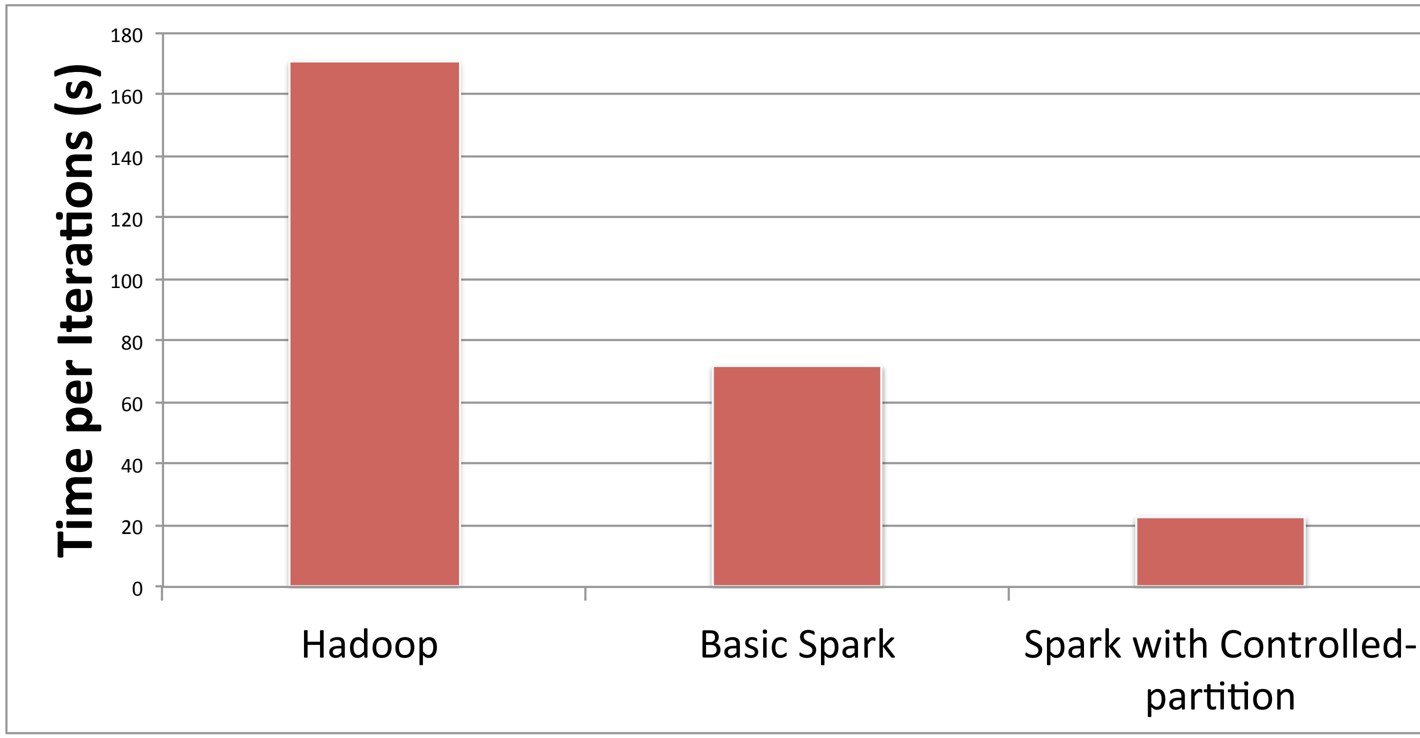
- PageRank Algorithm Performance Comparison

1. Start each page with a rank of 1
2. On each iteration, update each page's rank to

$$\Sigma_{i \in neighbors} rank_i / |neighbors_i|$$

```
links = // RDD of (url, neighbors) pairs
ranks = // RDD of (url, rank) pairs

for (i <- 1 to ITERATIONS) {
  ranks = links.join(ranks).flatMap {
    (url, (links, rank)) =>
      links.map(dest => (dest, rank/links.size))
  }.reduceByKey(_ + _)
}
```



Matei Zaharia, et al, Resilient Distributed Datasets: A Fault-Tolerant Abstraction for In-Memory Cluster Computing, NSDI 2012

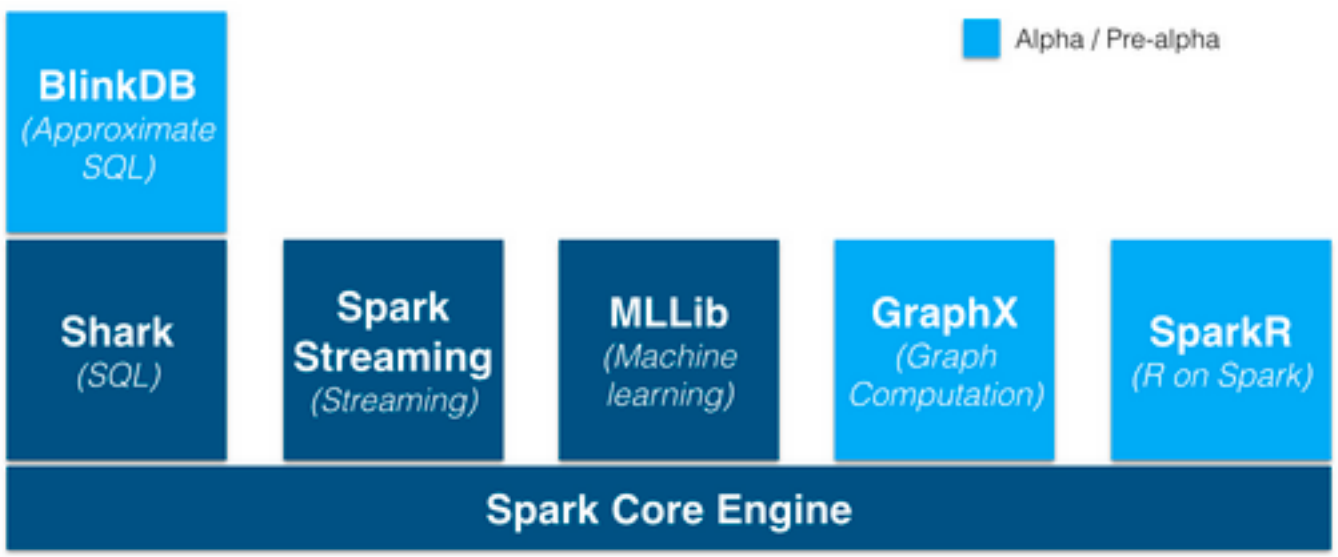
Unified pipeline

General Batching	Specialized systems			
	Streaming	**Iterative**	**Ad-hoc / SQL**	**Graph**
MapReduce	Storm	Mahout	Pig	Giraph
	S4		Hive	
	Samza		Drill	
			Impala	

Diverse APIs, Operational Cost, etc.

Unified pipeline

Unified pipeline



- With a Single Spark Cluster
  - Batching Processing: Spark Core
  - Query: Shark & Spark SQL & BlinkDB
  - Streaming: Spark Streaming
  - Machine Learning: MLlib
  - Graph: GraphX

Understanding Distributed Computing Framework

Understand a distributed computing framework

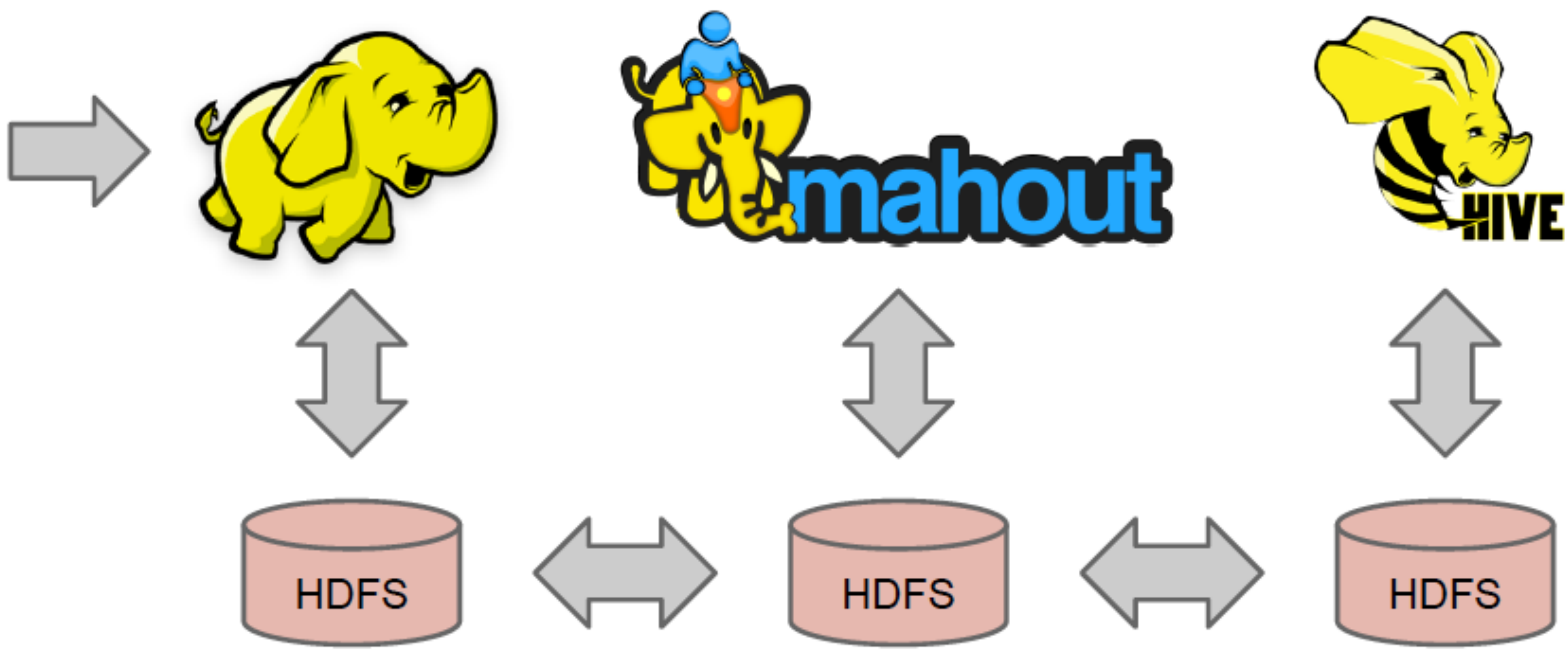
- DataFlow
  - e.g. Hadoop family utilizes HDFS to transfer data within a job and share data across jobs/applications

Understand a distributed computing framework

- DataFlow
  - e.g. Hadoop family utilizes HDFS to transfer data within a job and share data across jobs/applications

Understand a distributed computing framework

- DataFlow
  - e.g. Hadoop family utilizes HDFS to transfer data within a job and share data across jobs/applications



Understanding a distributed computing engine

- Task Management
  - How the computation is executed within multiple servers
    - How the tasks are scheduled
    - How the resources are allocated

Spark Data Abstraction Model

Basic Structure of Spark program

- A Spark Program

```
val sc = new SparkContext(…)

val points = sc.textFile("hdfs://...")
    .map(_.split.map(_.toDouble)).splitAt(1)
    .map { case (Array(label), features) =>
        LabeledPoint(label, features)
    }

val model = Model.train(points)
```

Basic Structure of Spark program

- A Spark Program

```
val sc = new SparkContext(…)

val points = sc.textFile("hdfs://…")
    .map(_.split.map(_.toDouble)).splitAt(1)
    .map { case (Array(label), features) =>
        LabeledPoint(label, features)
    }

val model = Model.train(points)
```

Includes the components driving the running of computing tasks (will introduce later)

Basic Structure of Spark program

- A Spark Program

```
val sc = new SparkContext(…)

val points = sc.textFile("hdfs://…")
      .map(_.split.map(_.toDouble)).splitAt(1)
      .map { case (Array(label), features) =>
            LabeledPoint(label, features)
      }

val model = Model.train(points)
```

Includes the components driving the running of computing tasks (will introduce later)

Load data from HDFS, forming a **RDD (Resilient Distributed Datasets) object**

Basic Structure of Spark program

- A Spark Program

```
val sc = new SparkContext(…)

val points = sc.textFile("hdfs://…")
    .map(_.split.map(_.toDouble)).splitAt(1)
    .map { case (Array(label), features) =>
        LabeledPoint(label, features)
    }

val model = Model.train(points)
```

Includes the components driving the running of computing tasks (will introduce later)

Load data from HDFS, forming a **RDD (Resilient Distributed Datasets) object**

Transformations to generate **RDDs** with expected element(s)/ format

Basic Structure of Spark program

- A Spark Program

```
val sc = new SparkContext(…)

val points = sc.textFile("hdfs://…")
    .map(_.split.map(_.toDouble)).splitAt(1)
    .map { case (Array(label), features) =>
        LabeledPoint(label, features)
    }

val model = Model.train(points)
```

Includes the components driving the running of computing tasks (will introduce later)

Load data from HDFS, forming a **RDD (Resilient Distributed Datasets) object**

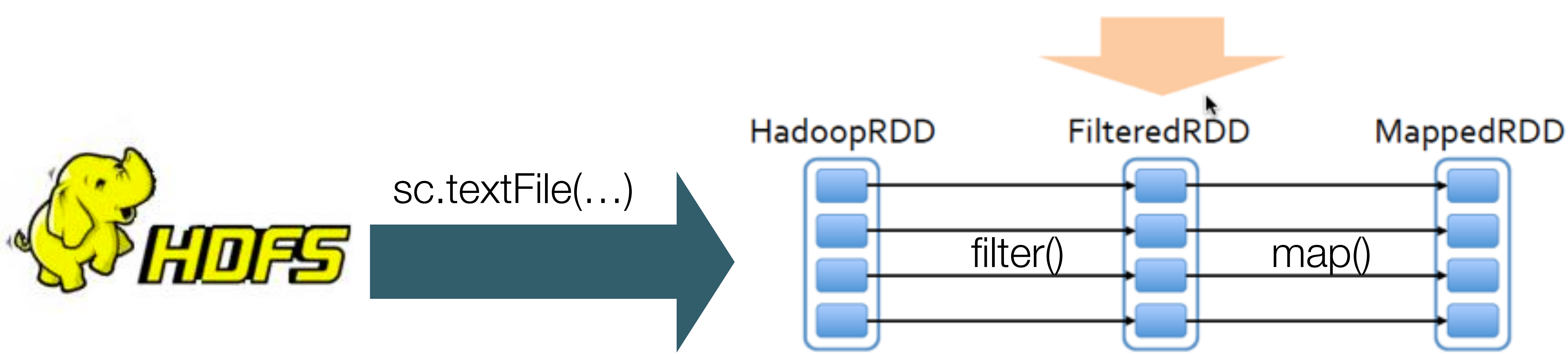
Transformations to generate **RDDs** with expected element(s)/ format

All Computations are around RDDs

Resilient Distributed Dataset

- RDD is a distributed memory abstraction which is
  - data collection
  - immutable
    - created by either loading from stable storage system (e.g. HDFS) or through transformations on other RDD(s)
  - partitioned and distributed

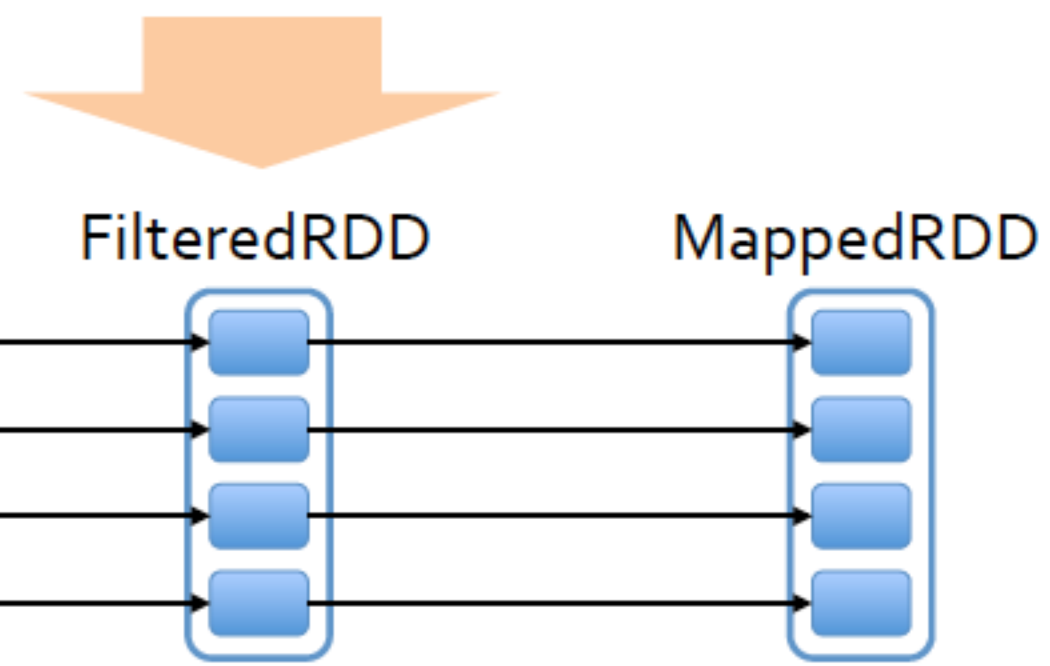
E.g.:
```
messages = textFile(...).filter(_.contains("error"))
                        .map(_.split('\t')(2))
```



From data to computation

- Lineage

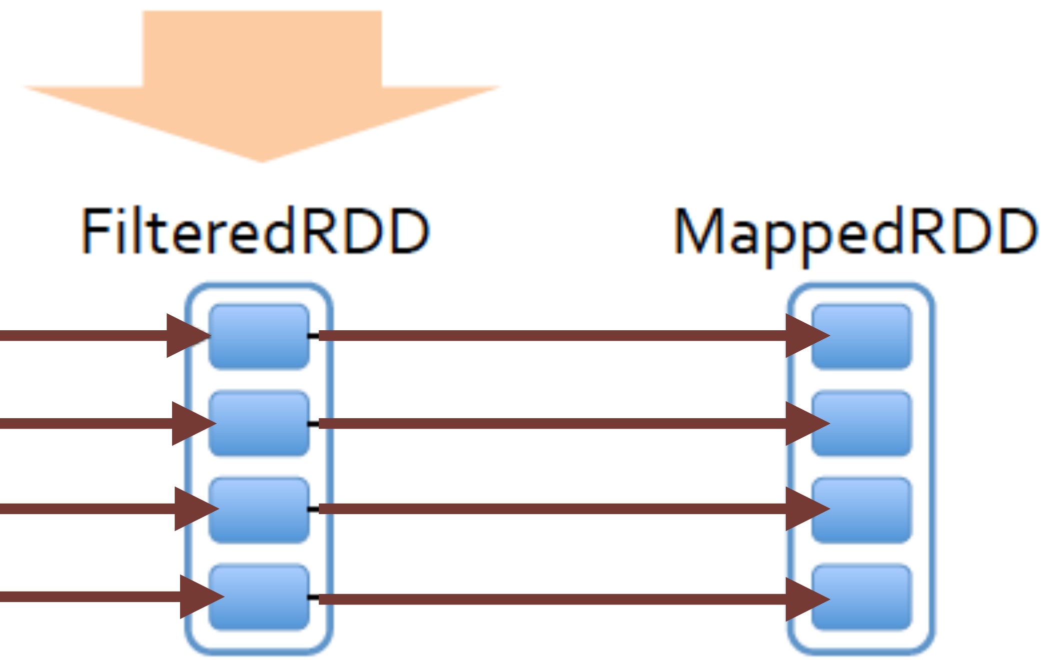
E.g.:
```
messages = textFile(...).filter(_.contains("error"))
                        .map(_.split('\t')(2))
```



From data to computation

- Lineage

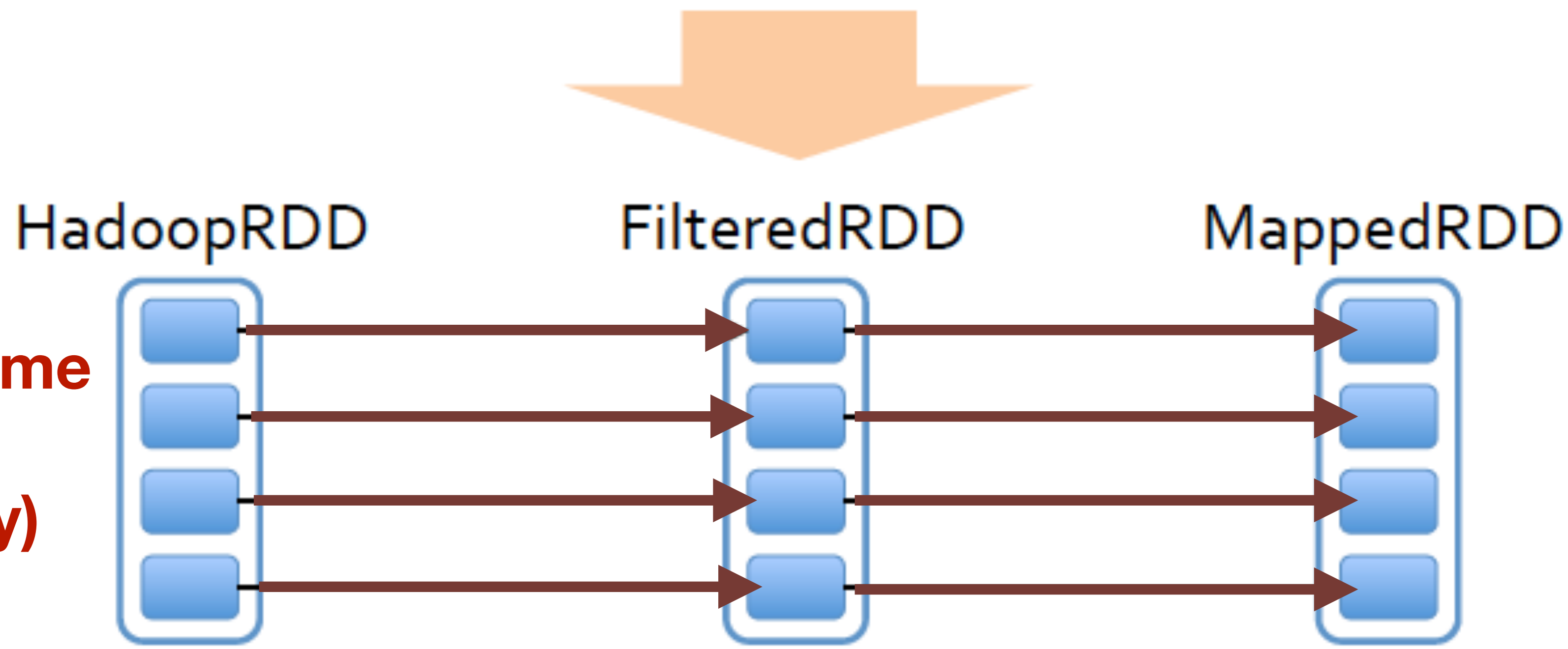
E.g.: `messages = textFile(...).filter(_.contains("error"))`
`.map(_.split('\t')(2))`



From data to computation

- Lineage

E.g.:
```
messages = textFile(...).filter(_.contains("error"))
                        .map(_.split('\t')(2))
```

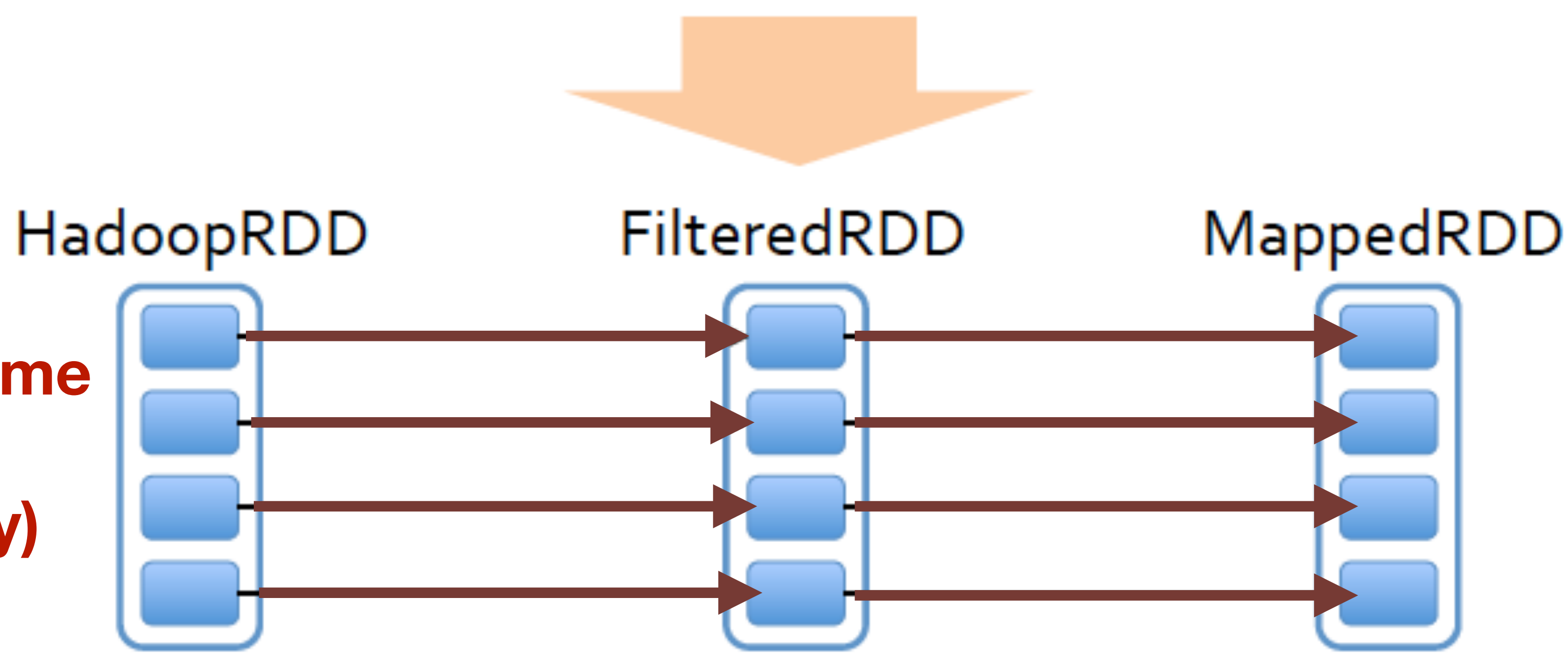


Where do I come from? (dependency)

From data to computation

- Lineage

E.g.:
```
messages = textFile(...).filter(_.contains("error"))
                        .map(_.split('\t')(2))
```



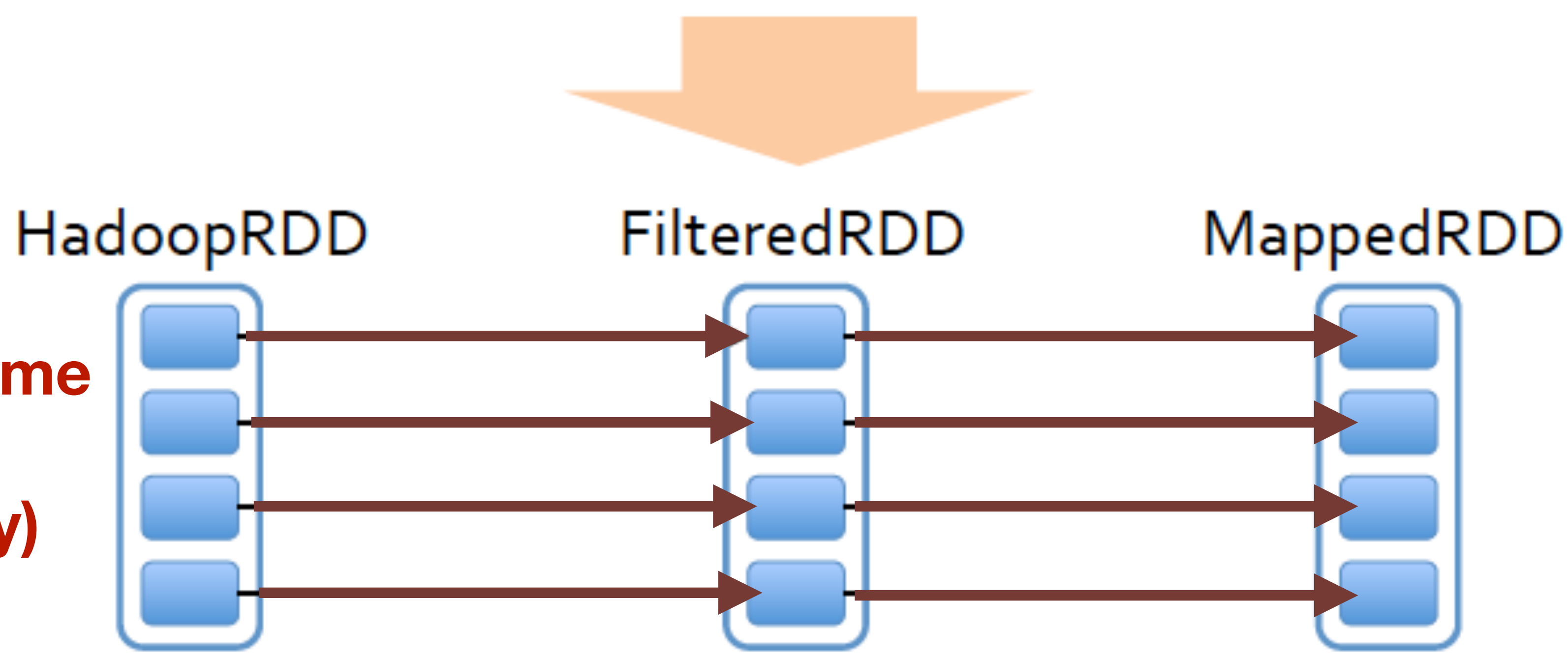
Where do I come from? (dependency)

How do I come from? (save the functions calculating the partitions)

From data to computation

- Lineage

E.g.: `messages = textFile(...).filter(_.contains("error"))`
`.map(_.split('\t')(2))`



Where do I come from? (dependency)

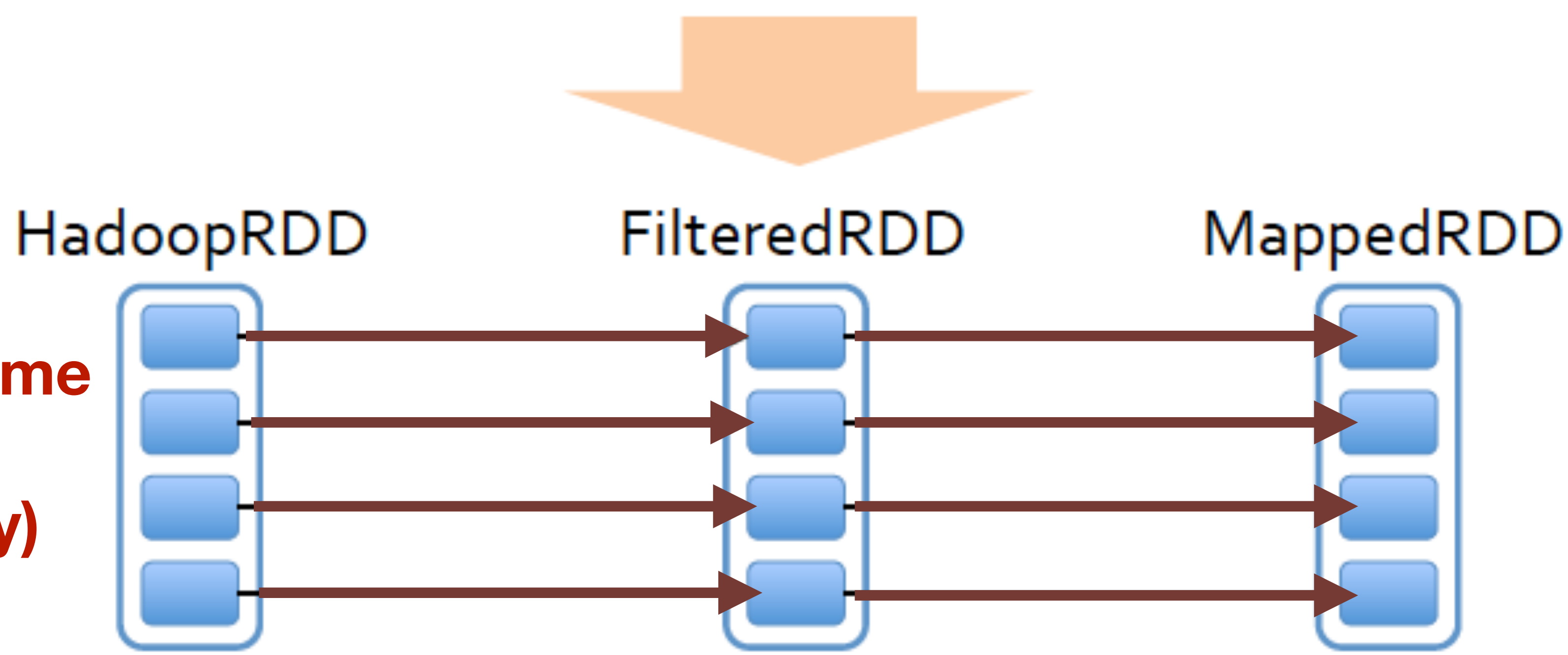
Computation is organized as a DAG (Lineage)

How do I come from? (save the functions calculating the partitions)

From data to computation

- Lineage

E.g.:
```
messages = textFile(...).filter(_.contains("error"))
                        .map(_.split('\t')(2))
```



Where do I come from? (dependency)

How do I come from? (save the functions calculating the partitions)

Computation is organized as a DAG (Lineage)

Lost data can be recovered in parallel with the help of the lineage DAG

Cache

- Frequently accessed RDDs can be materialized and cached in memory
- Cached RDD can also be replicated for fault tolerance (Spark scheduler takes cached data locality into account)
- Manage the cache space with LRU algorithm

Benefits Brought Cache

- Example (Log Mining)

```
lines = spark.textFile("hdfs://...")
errors = lines.filter(_.startsWith("ERROR"))
messages = errors.map(_.split('\t')(2))
cachedMsgs = messages.cache()

cachedMsgs.filter(_.contains("foo")).count
cachedMsgs.filter(_.contains("bar")).count
. . .
```

Benefits Brought Cache

- Example (Log Mining)

```
lines = spark.textFile("hdfs://...")
errors = lines.filter(_.startsWith("ERROR"))
messages = errors.map(_.split('\t')(2))
cachedMsgs = messages.cache()

cachedMsgs.filter(_.contains("foo")).count
cachedMsgs.filter(_.contains("bar")).count
. . .
```

Count is an action, for the first time, it has to calculate from the start of the DAG Graph (textFile)

Benefits Brought Cache

- Example (Log Mining)

```
lines = spark.textFile("hdfs://...")
errors = lines.filter(_.startsWith("ERROR"))
messages = errors.map(_.split('\t')(2))
cachedMsgs = messages.cache()

cachedMsgs.filter(_.contains("foo")).count
cachedMsgs.filter(_.contains("bar")).count
. . .
```

Count is an action, for the first time, it has to calculate from the start of the DAG Graph (textFile)

Because the data is cached, the second count does not trigger a "start-from-zero" computation, instead, it is based on "cachedMsgs" directly

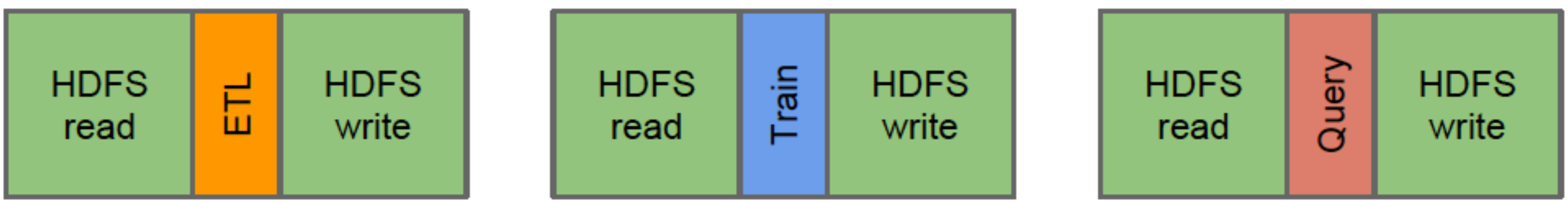
Summary

- Resilient Distributed Datasets (RDD)
  - Distributed memory abstraction in Spark
  - Keep computation run in memory with best effort
- Keep track of the “lineage” of data
  - Organize computation
  - Support fault-tolerance
- Cache

RDD brings much better performance by simplifying the data flow

- Share Data among Applications

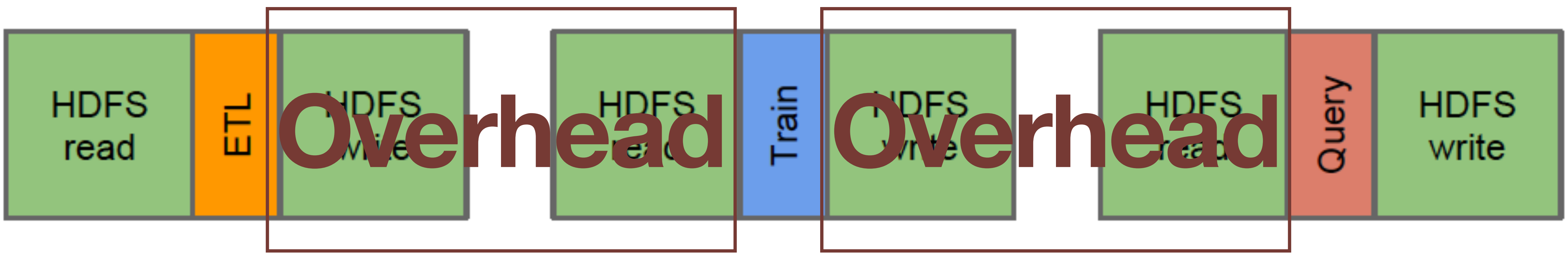
A typical data processing pipeline



RDD brings much better performance by simplifying the data flow

- Share Data among Applications

A typical data processing pipeline



RDD brings much better performance by simplifying the data flow

- Share Data among Applications

A typical data processing pipeline

Overhead **Overhead**

RDD brings much better performance by simplifying the data flow

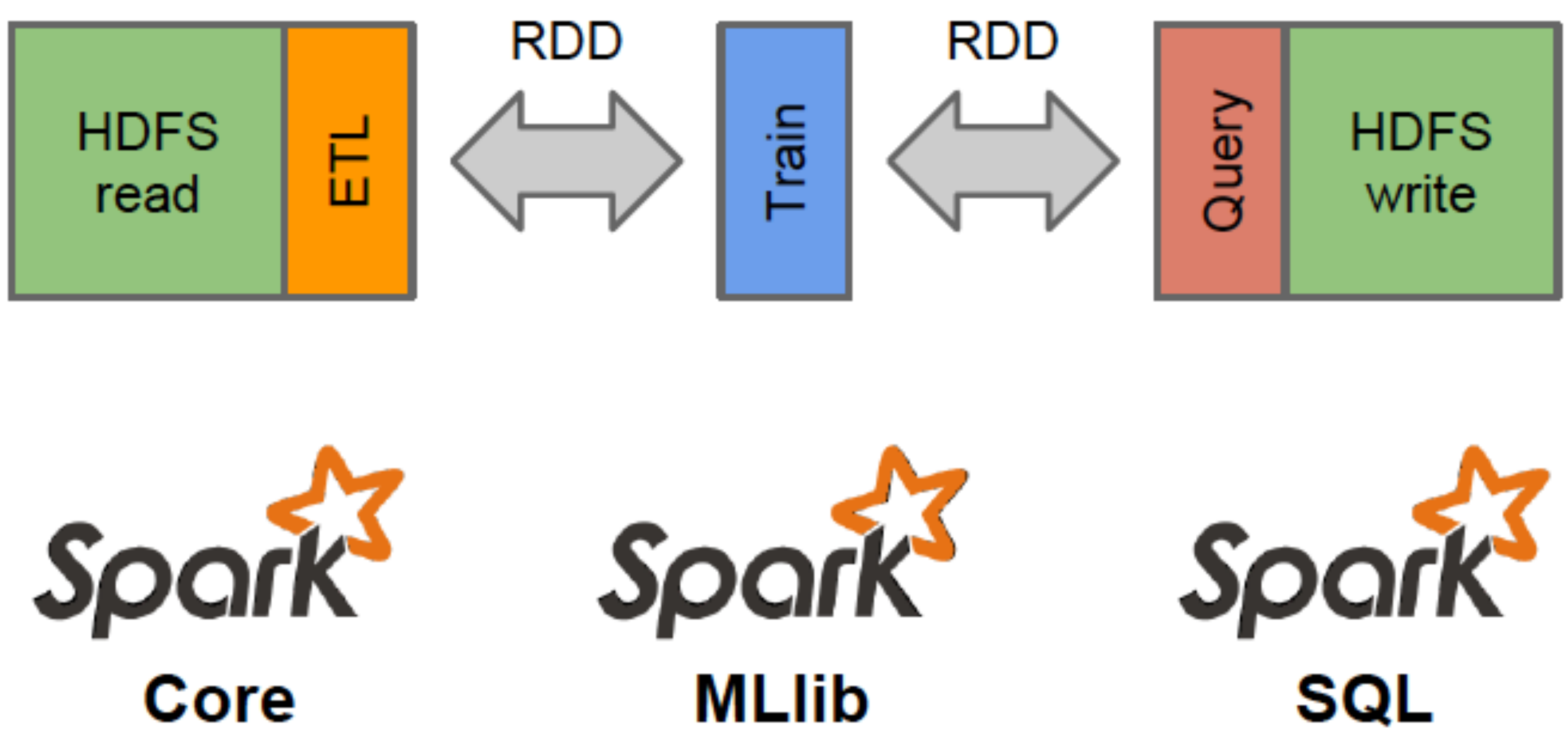
- Share Data among Applications

A typical data processing pipeline

RDD brings much better performance by simplifying the data flow

- Share Data among Applications

A typical data processing pipeline



RDD brings much better performance by simplifying the data flow

- Share data in Iterative Algorithms
  - Certain amount of predictive/machine learning algorithms are iterative
  - e.g. K-Means

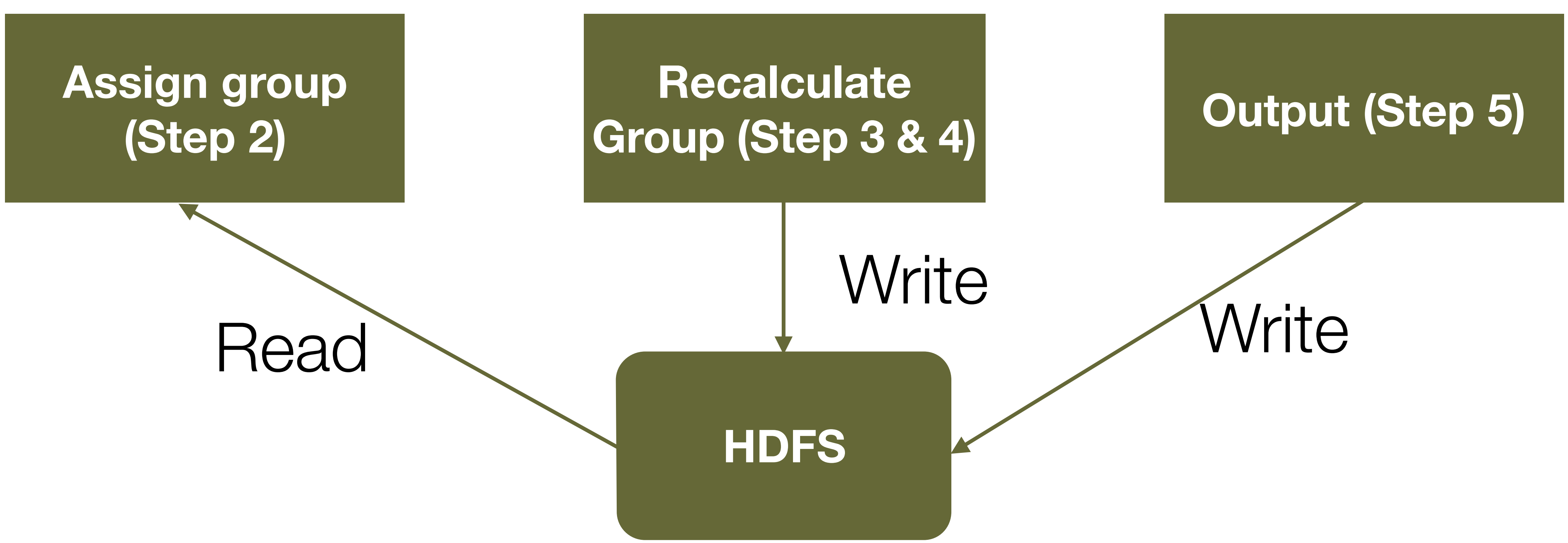
Step 1: Place randomly initial group centroids into the space.
Step 2: Assign each object to the group that has the closest centroid.
Step 3: Recalculate the positions of the centroids.
Step 4: If the positions of the centroids didn't change go to the next step, else go to Step 2.
Step 5: End.

RDD brings much better performance by simplifying the data flow

- Share data in Iterative Algorithms
  - Certain amount of predictive/machine learning algorithms are iterative
  - e.g. K-Means

RDD brings much better performance by simplifying the data flow

- Share data in Iterative Algorithms
  - Certain amount of predictive/machine learning algorithms are iterative
  - e.g. K-Means

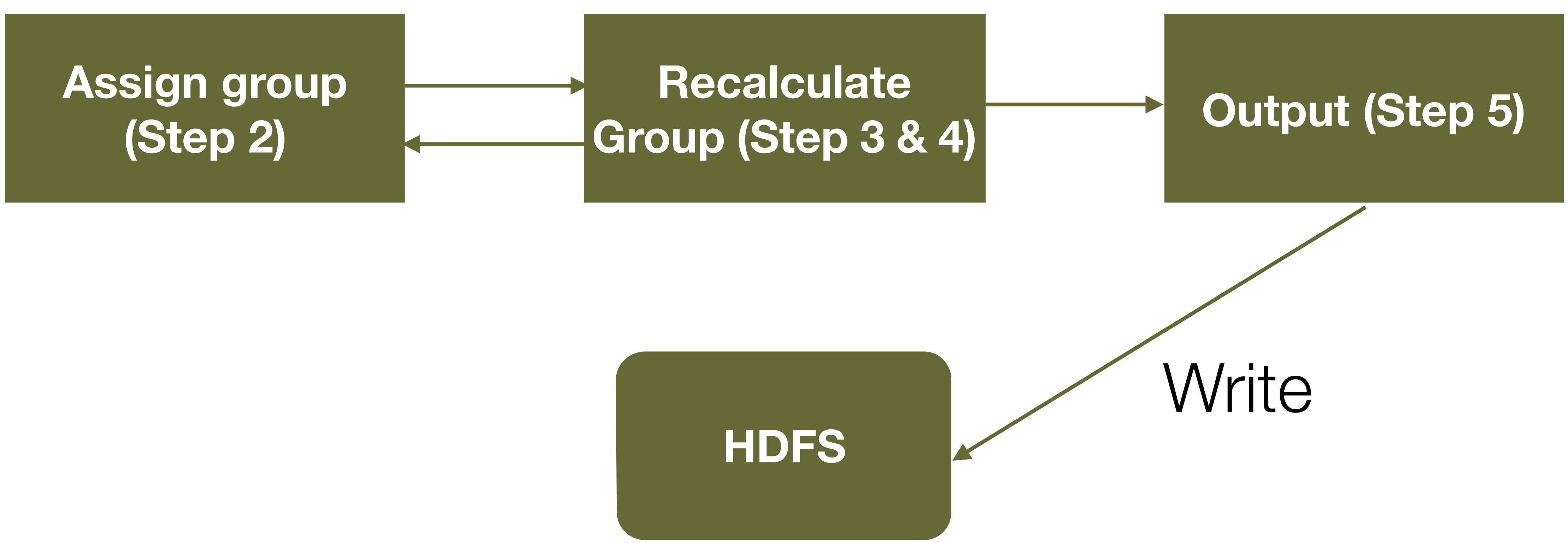


RDD brings much better performance by simplifying the data flow

- Share data in Iterative Algorithms
  - Certain amount of predictive/machine learning algorithms are iterative
  - e.g. K-Means

RDD brings much better performance by simplifying the data flow

- Share data in Iterative Algorithms
  - Certain amount of predictive/machine learning algorithms are iterative
  - e.g. K-Means

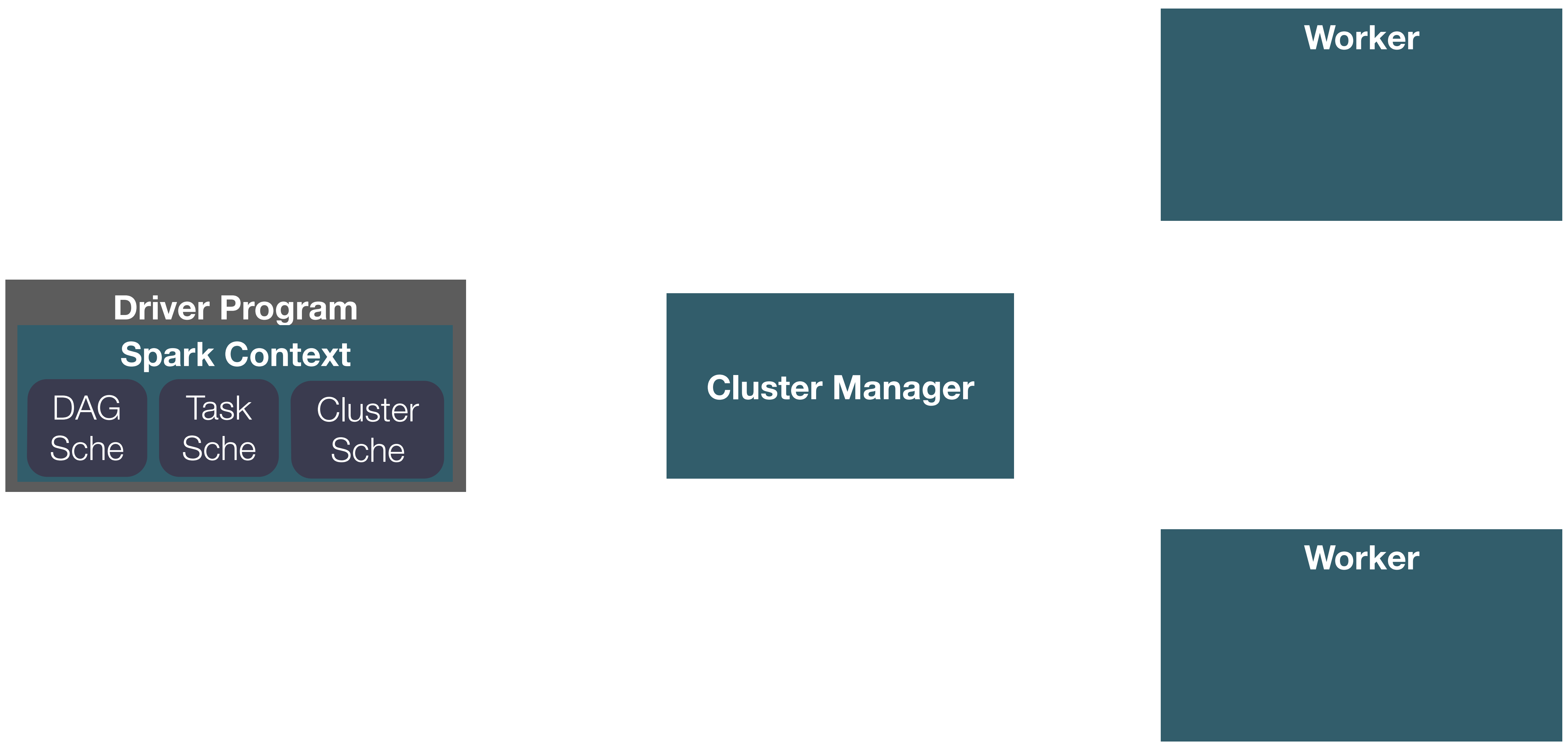


RDD brings much better performance by simplifying the data flow

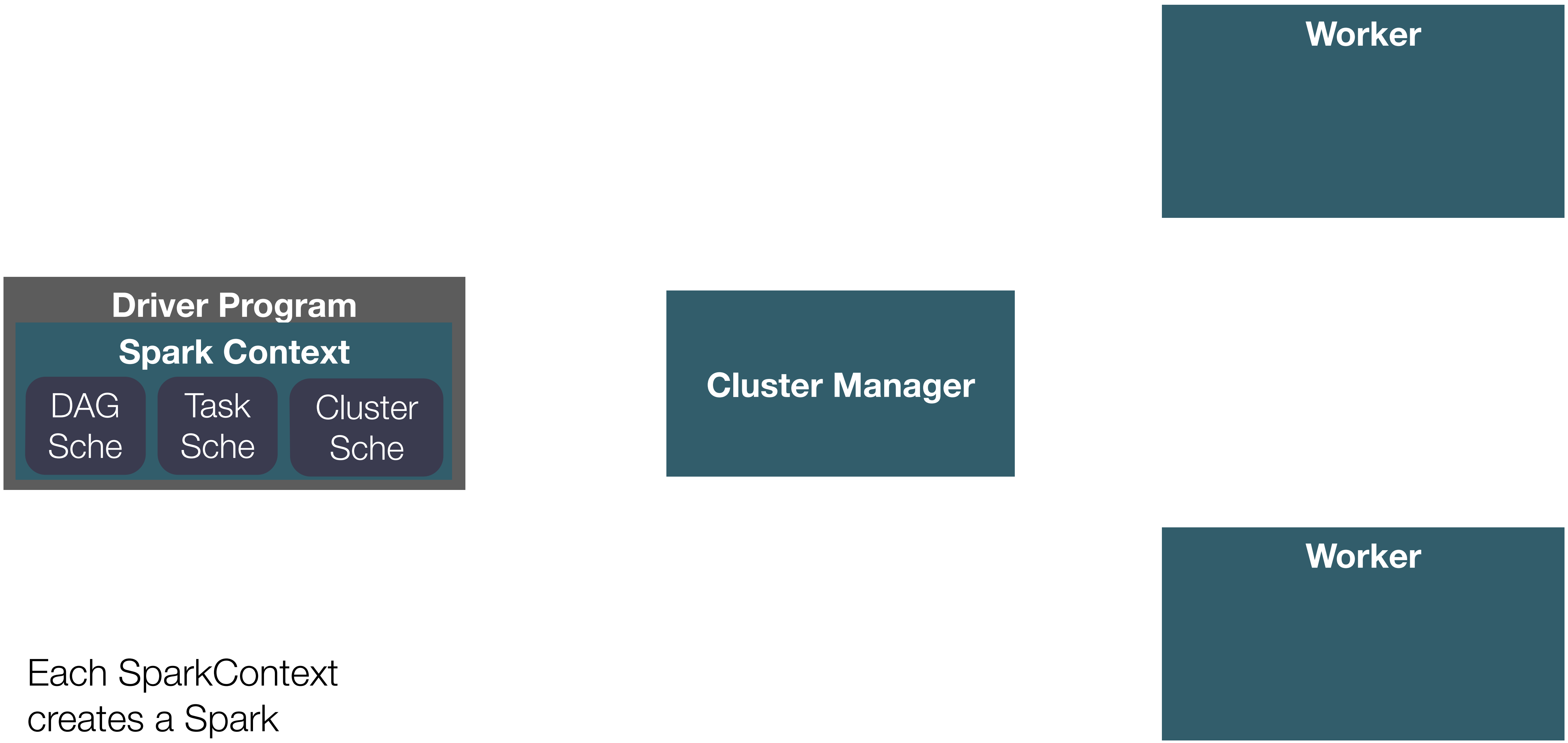
- Share data in Iterative Algorithms
  - Certain amount of predictive/machine learning algorithms are iterative
  - e.g. K-Means

Spark Scheduler

The Structure of Spark Cluster

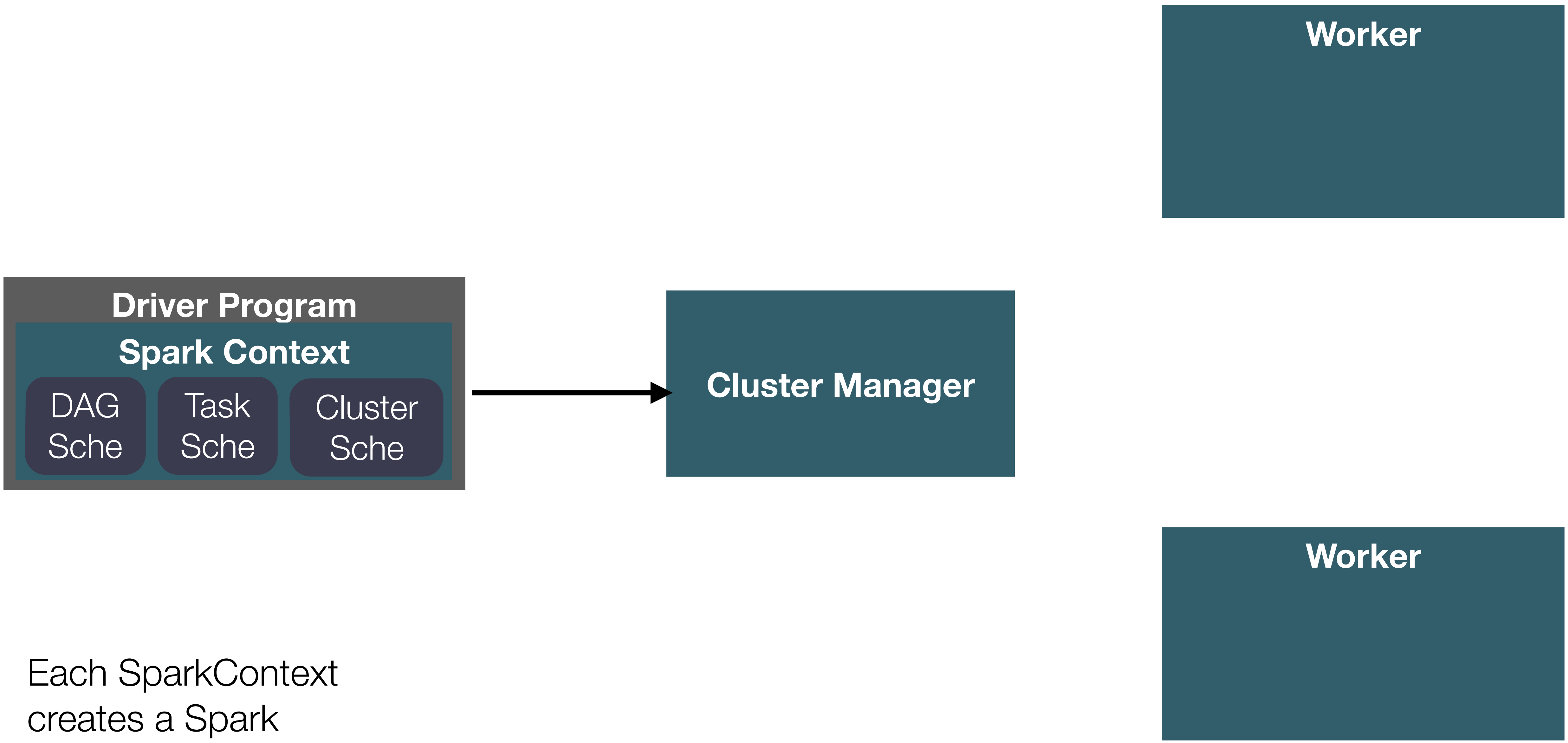


The Structure of Spark Cluster



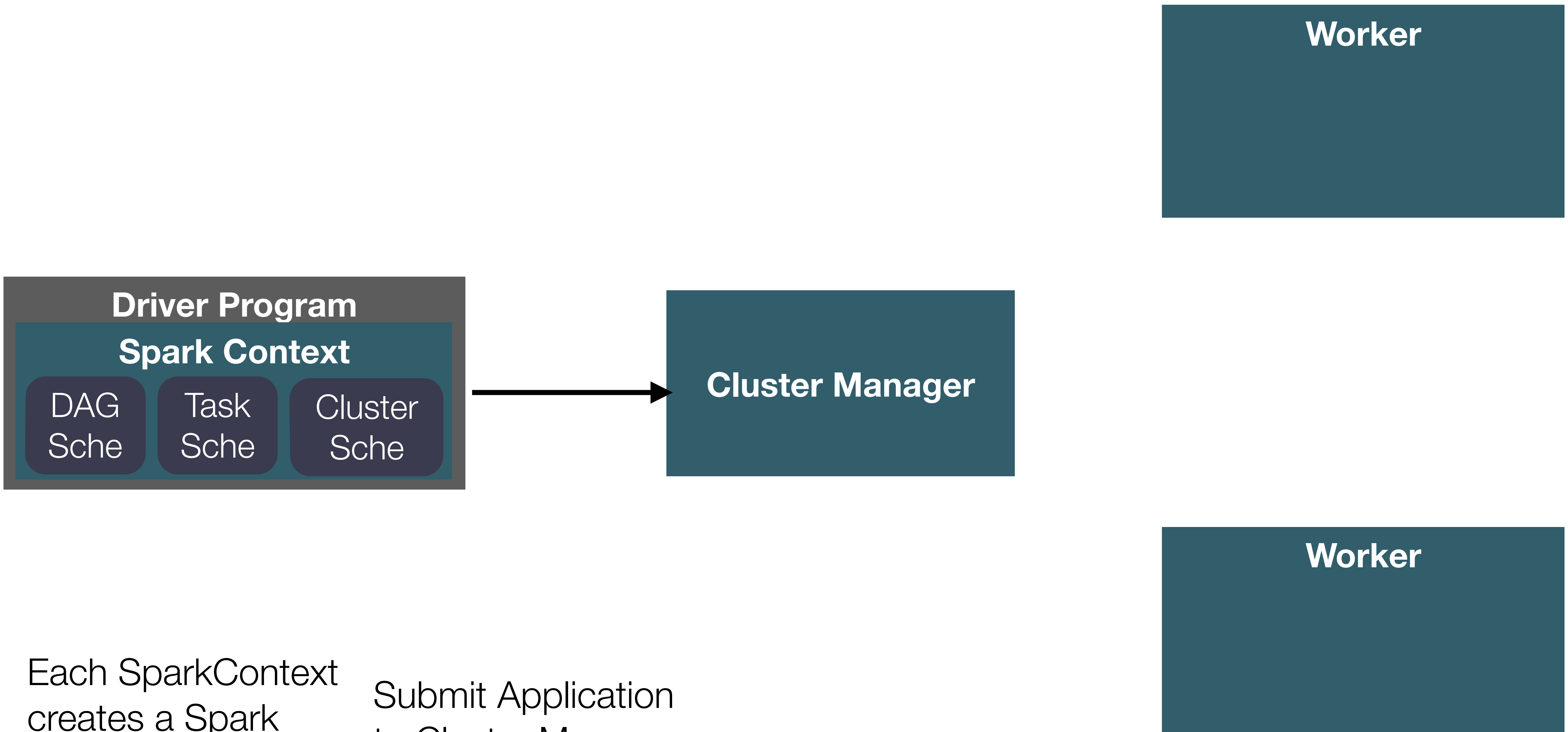
Each SparkContext creates a Spark application

The Structure of Spark Cluster



Each SparkContext creates a Spark application

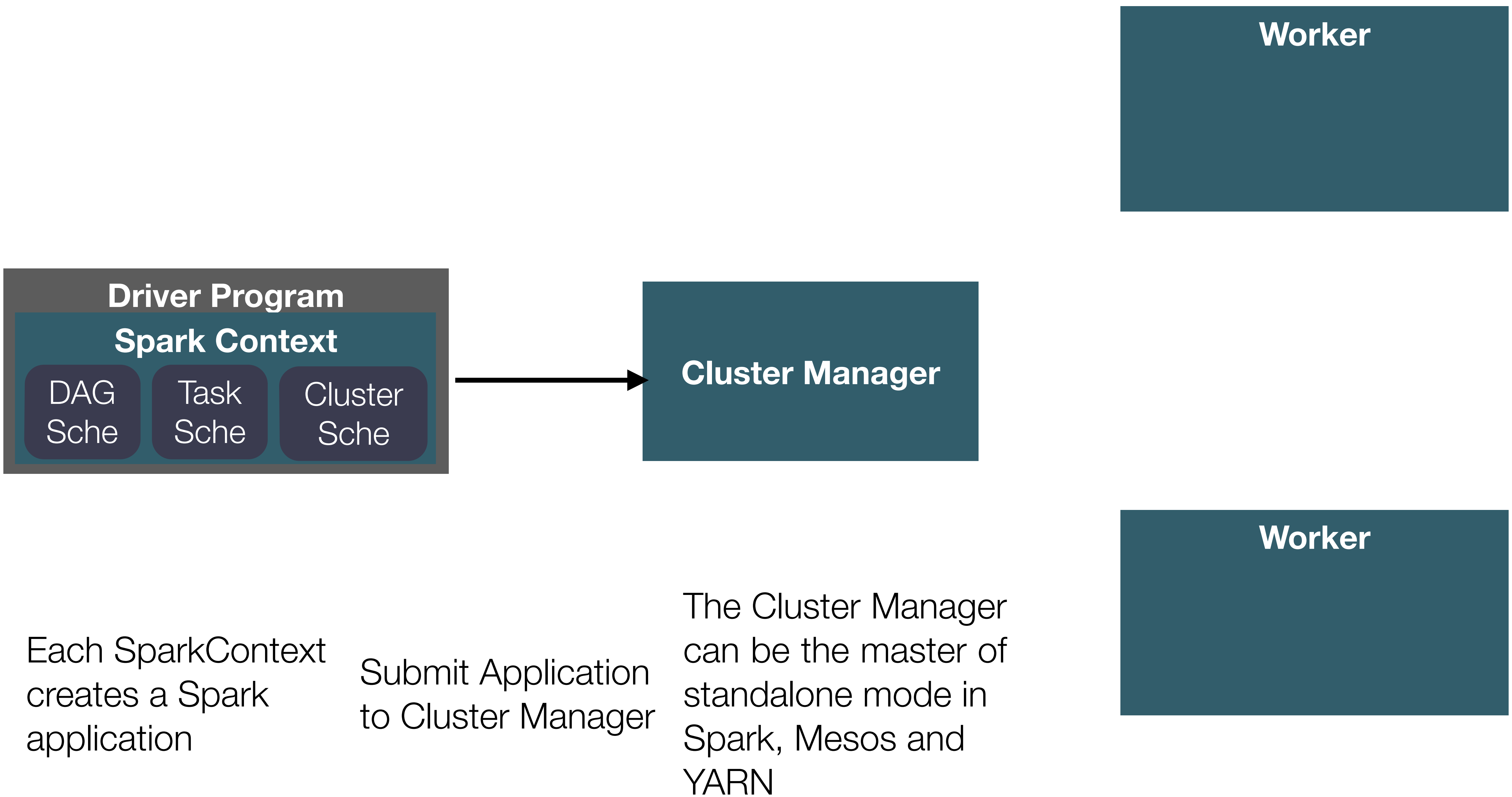
The Structure of Spark Cluster



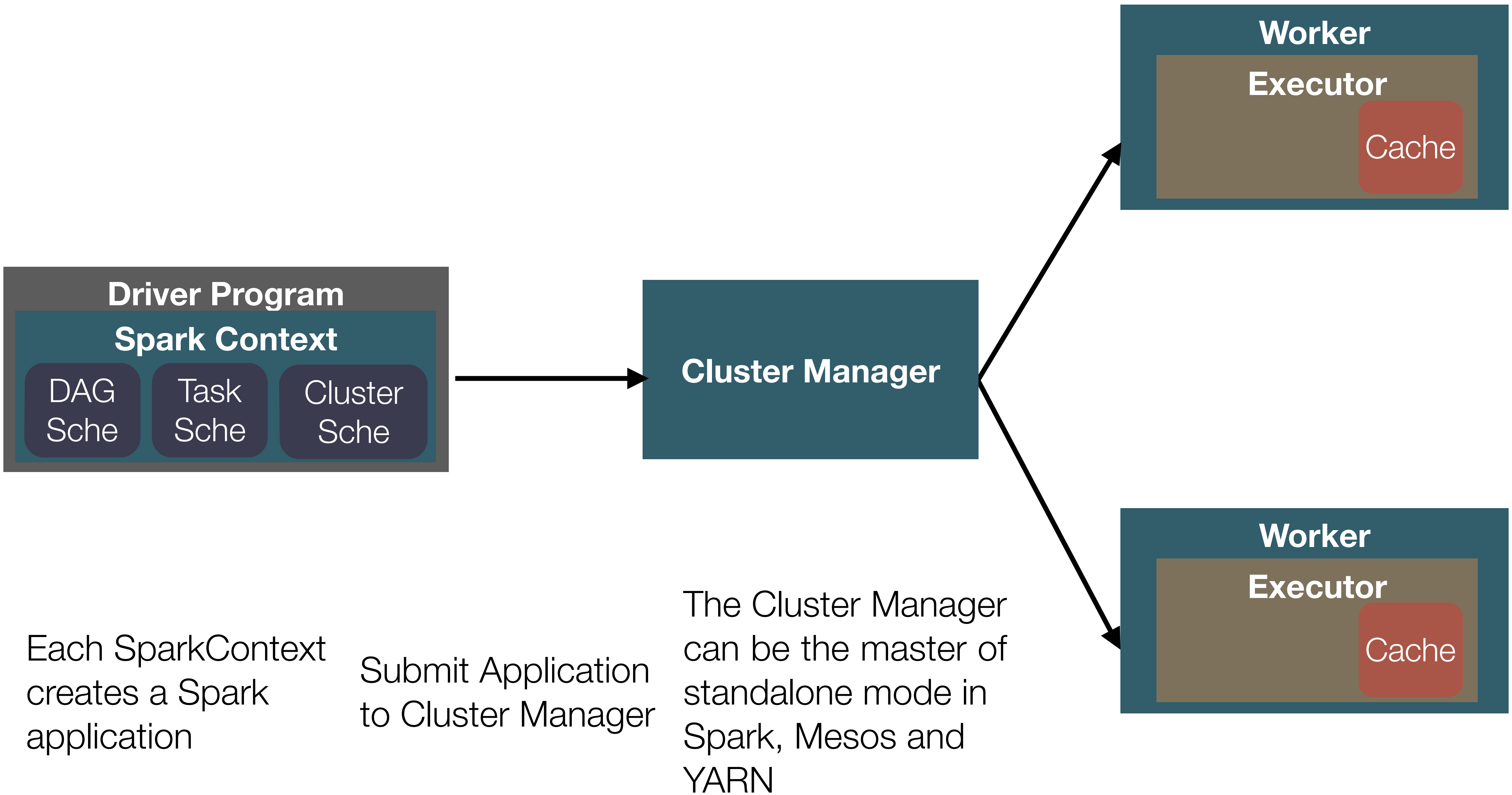
Each SparkContext creates a Spark application

Submit Application to Cluster Manager

The Structure of Spark Cluster

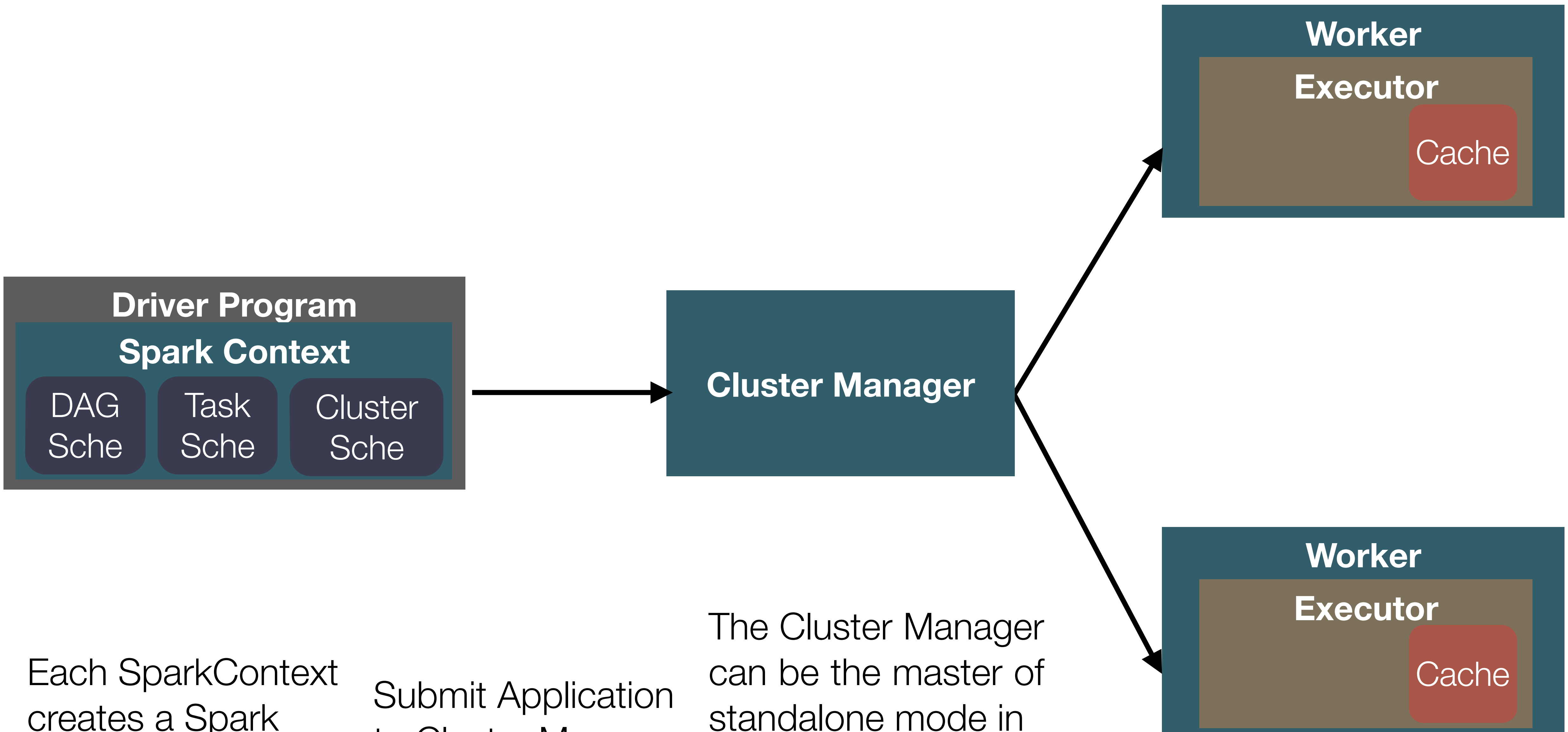


The Structure of Spark Cluster



The Structure of Spark Cluster

Start Executors for the application in Workers; Executors registers with ClusterScheduler;



Each SparkContext creates a Spark application

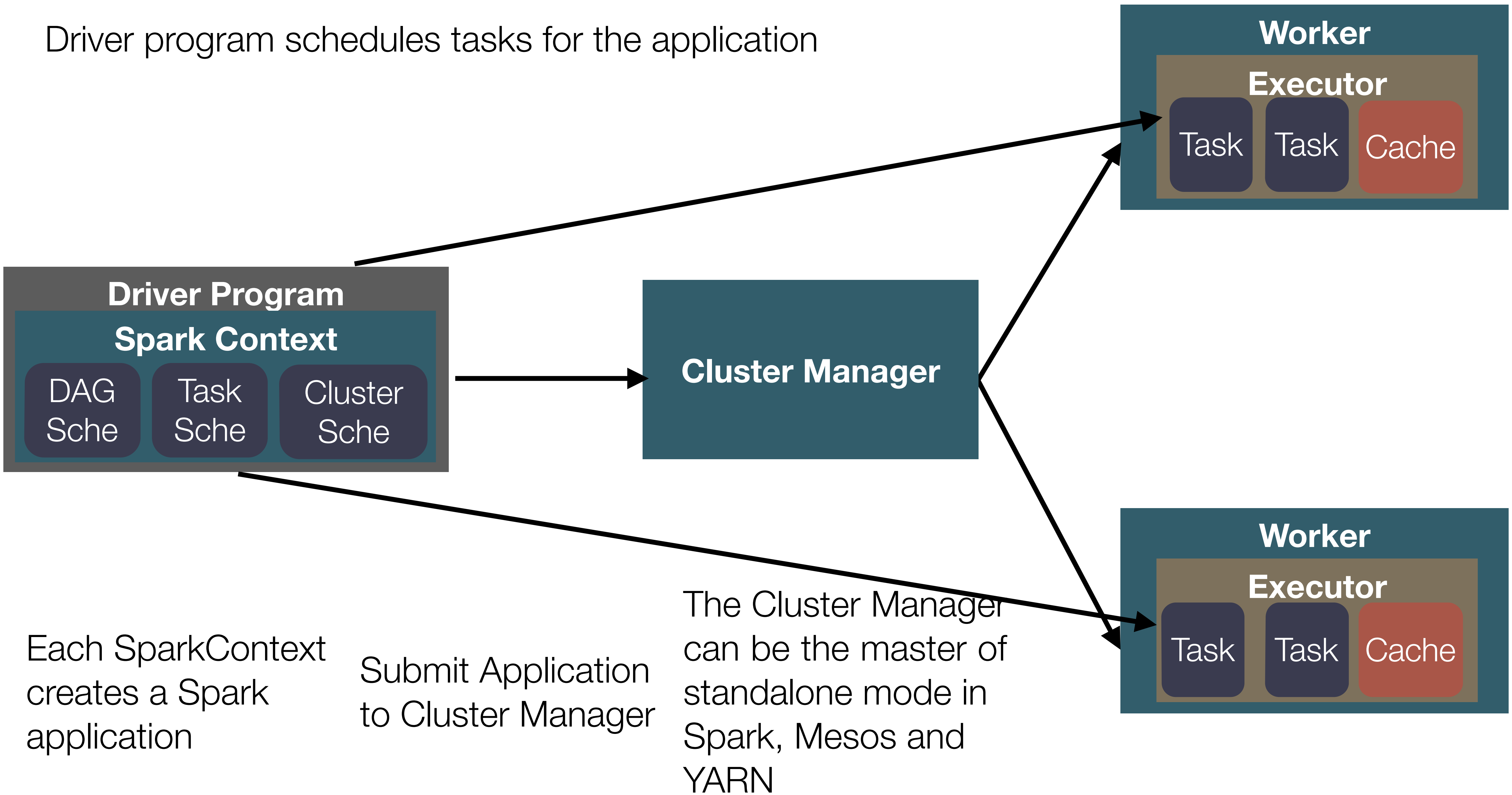
Submit Application to Cluster Manager

The Cluster Manager can be the master of standalone mode in Spark, Mesos and YARN

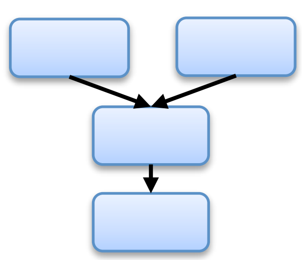
The Structure of Spark Cluster

Start Executors for the application in Workers; Executors registers with ClusterScheduler;

Driver program schedules tasks for the application

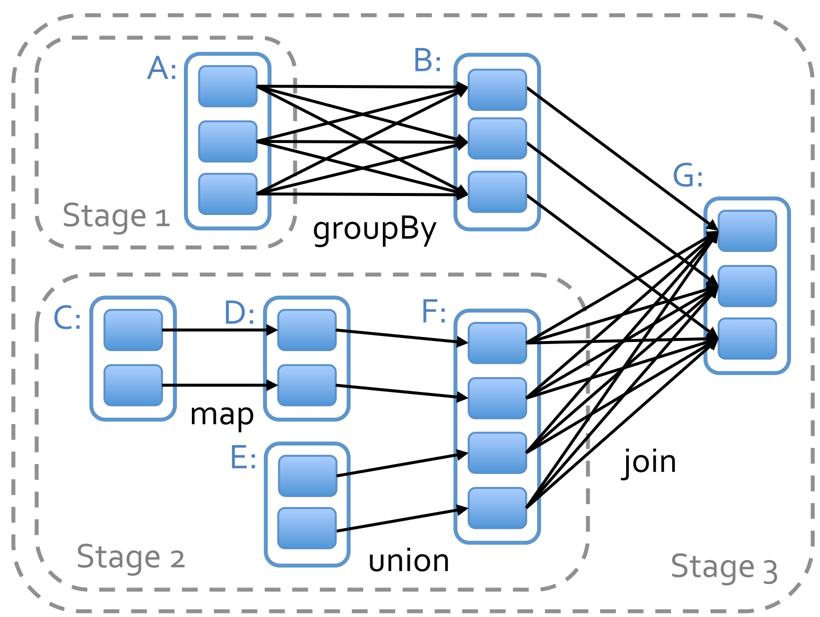


Scheduling Process



RDD objects are connected together with a DAG

DAGScheduler



Split DAG

Submit each stage as a TaskSet

TaskScheduler

TaskSetManagers

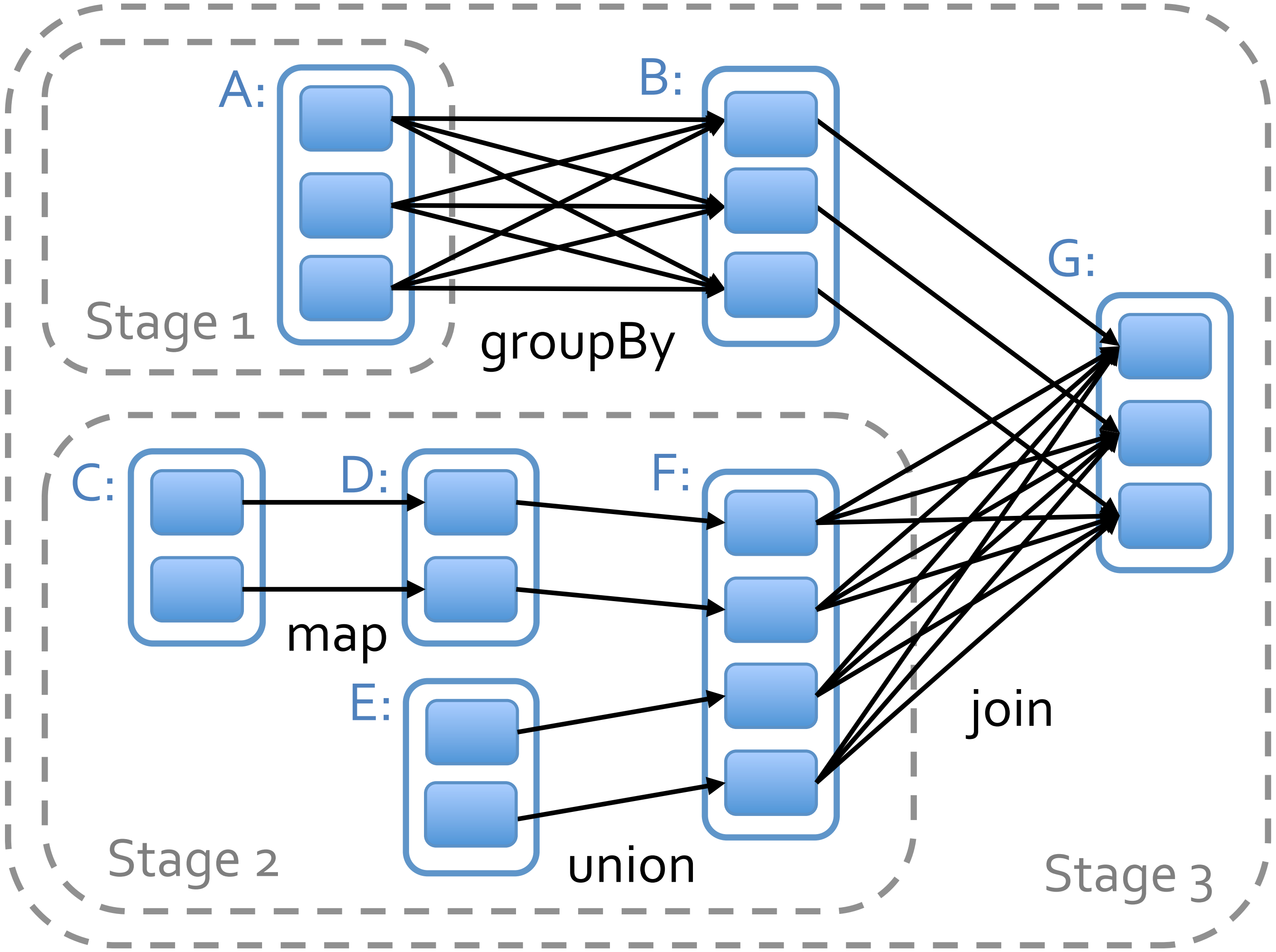
(monitor the progress of tasks and handle failed stages)

Failed Stages

ClusterScheduler

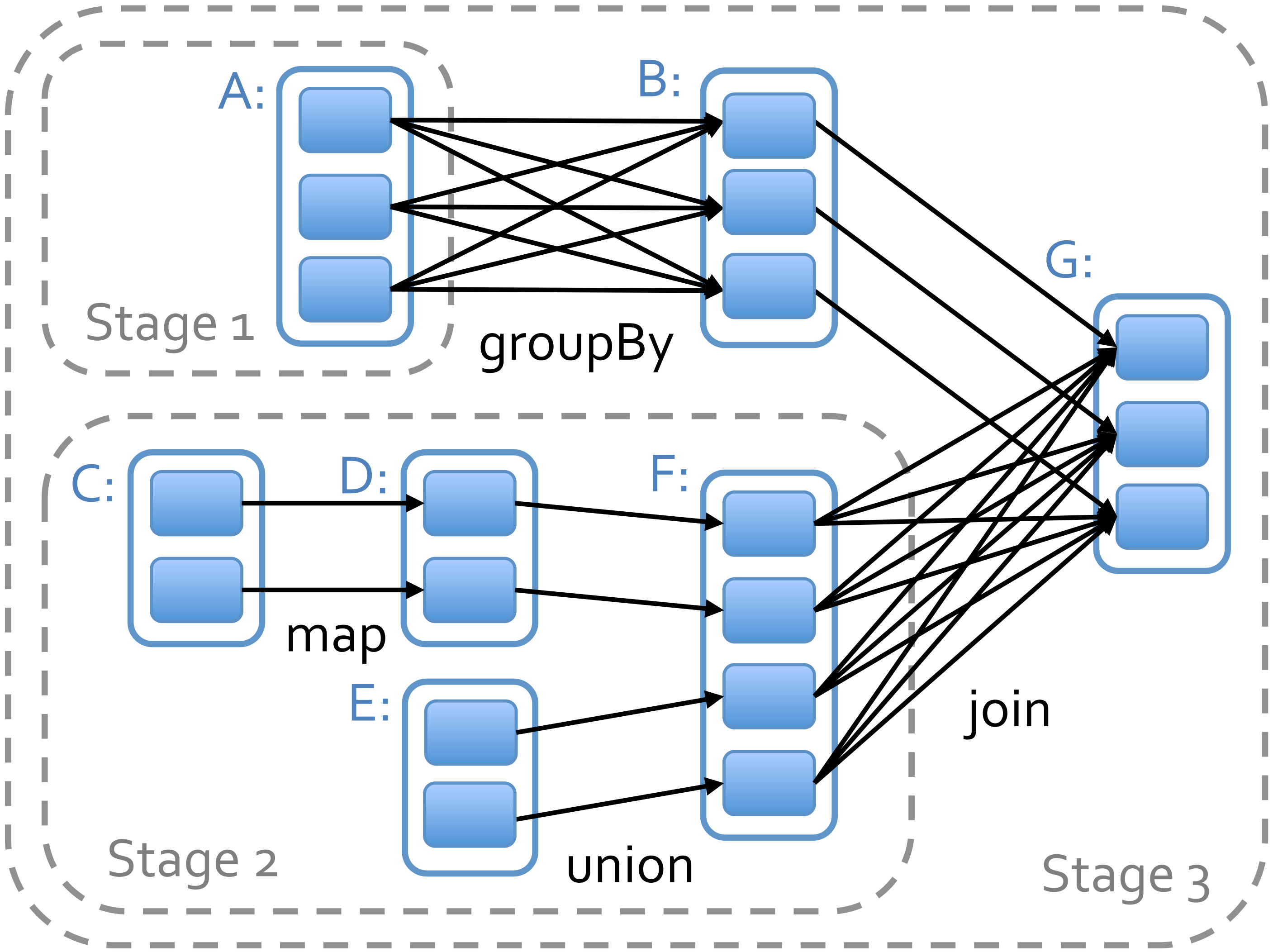
Submit Tasks to Executors

Scheduling Optimization



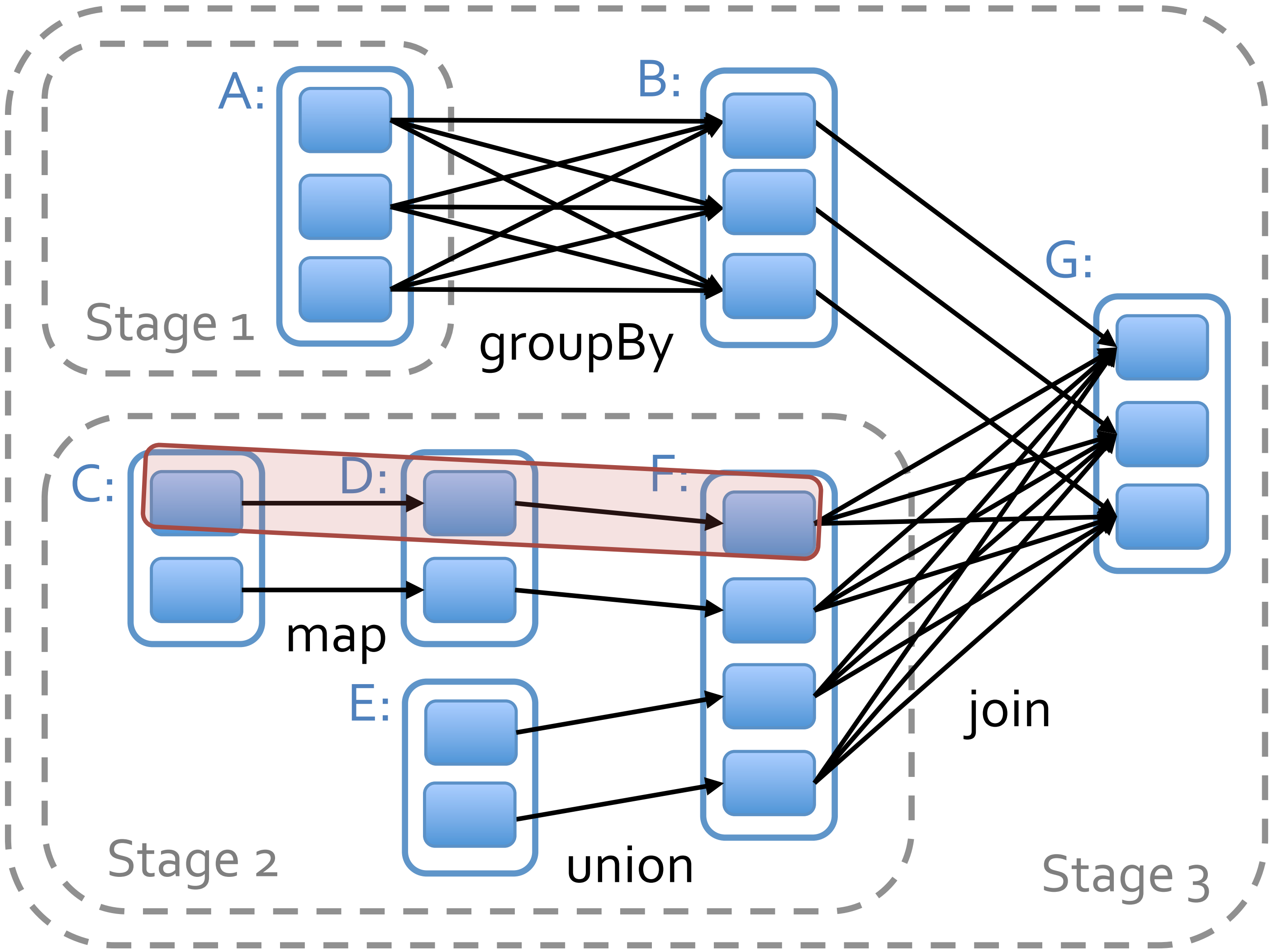
Scheduling Optimization

Within Stage Optimization, Pipelining the generation of RDD partitions when they are in narrow dependency



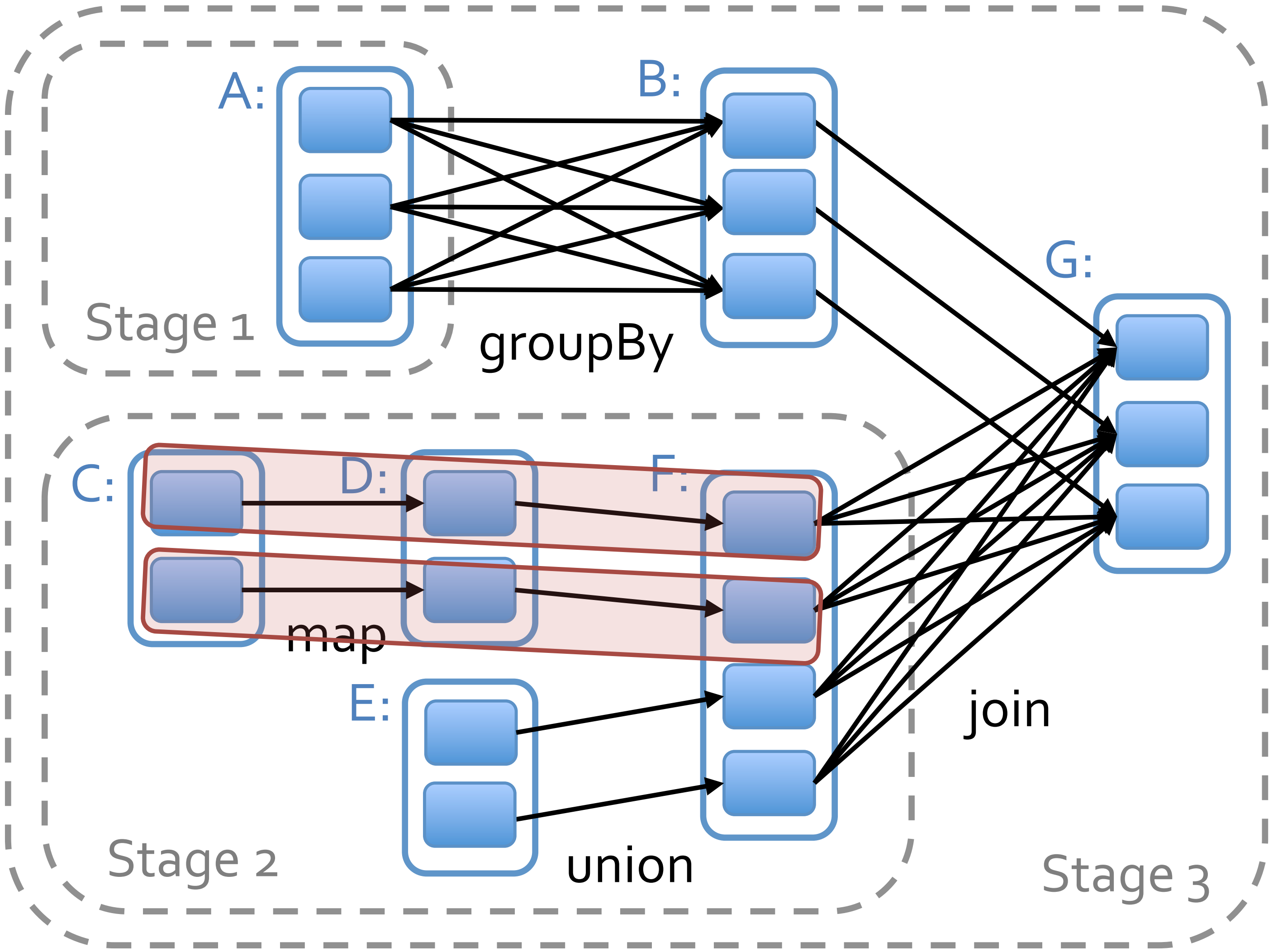
Scheduling Optimization

Within Stage Optimization, Pipelining the generation of RDD partitions when they are in narrow dependency



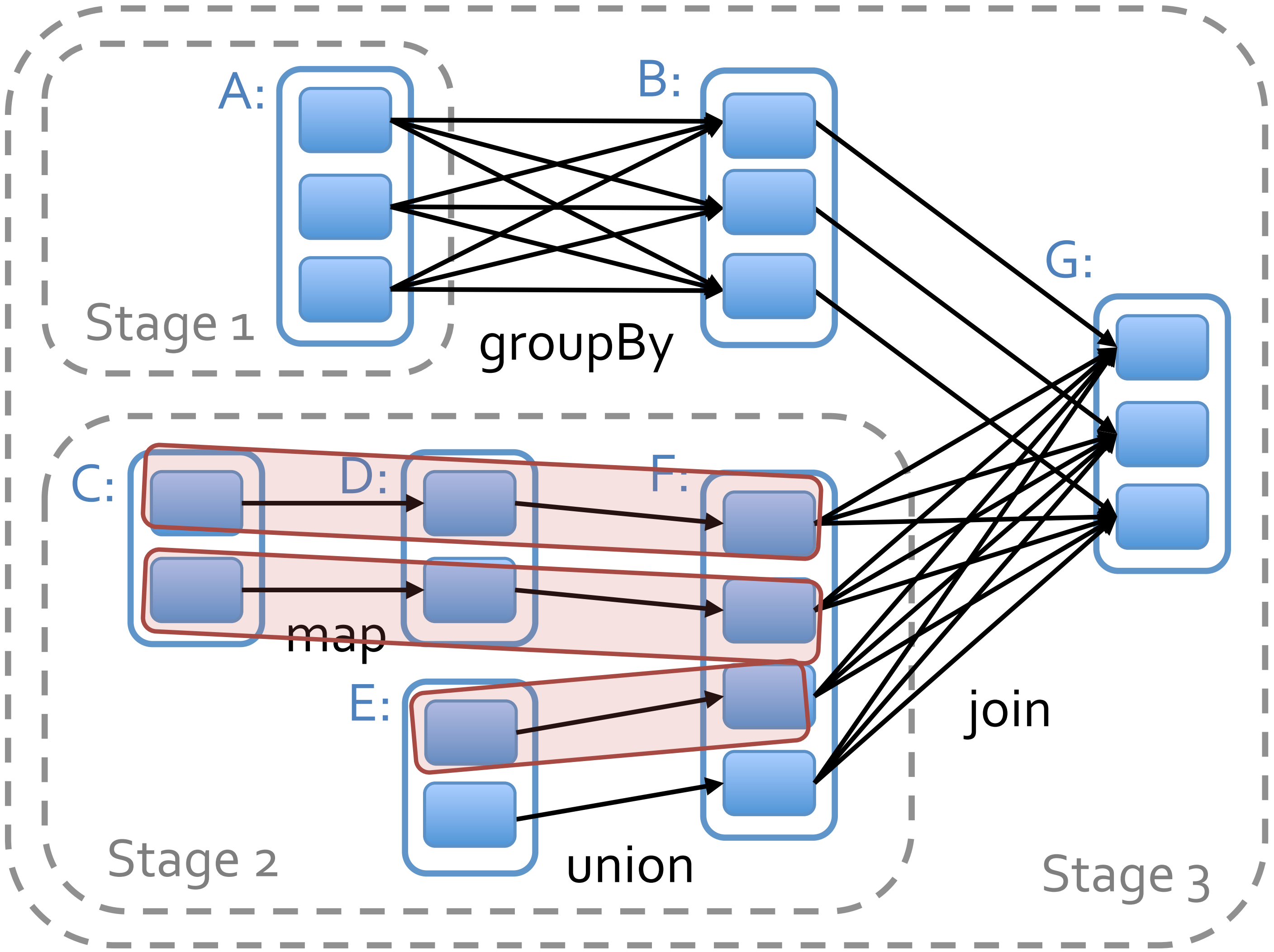
Scheduling Optimization

Within Stage Optimization, Pipelining the generation of RDD partitions when they are in narrow dependency



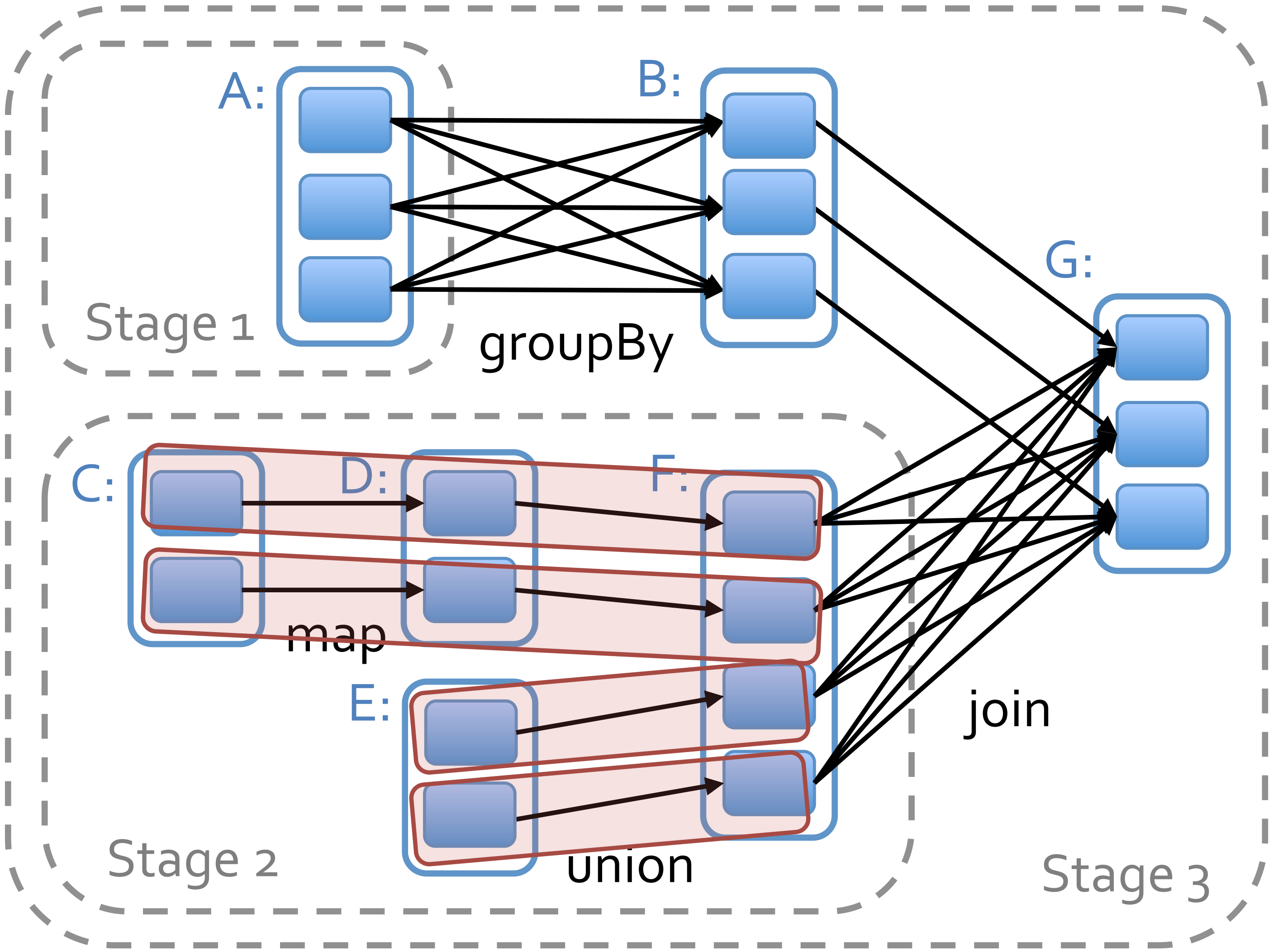
Scheduling Optimization

Within Stage Optimization, Pipelining the generation of RDD partitions when they are in narrow dependency



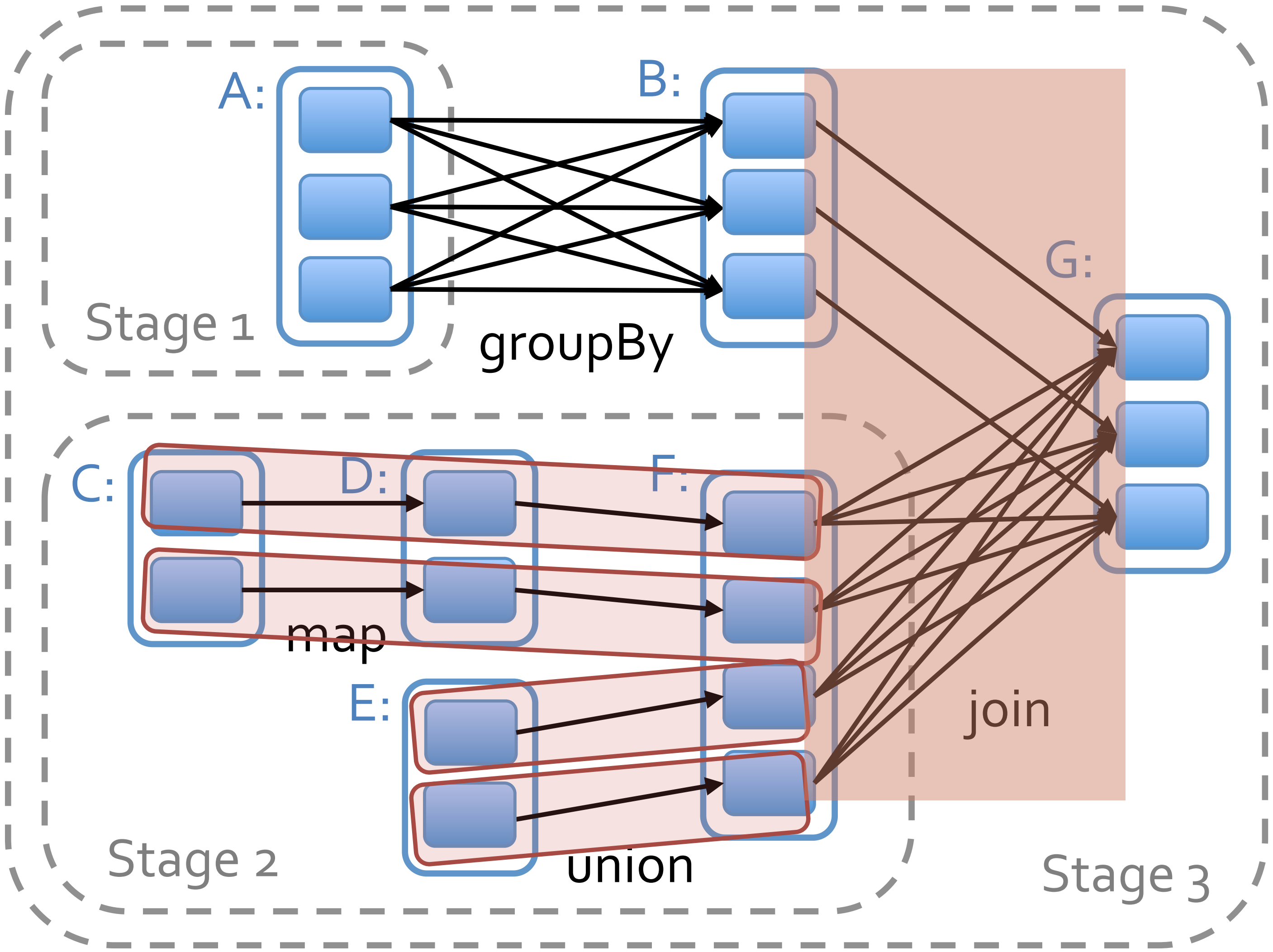
Scheduling Optimization

Within Stage Optimization, Pipelining the generation of RDD partitions when they are in narrow dependency



Scheduling Optimization

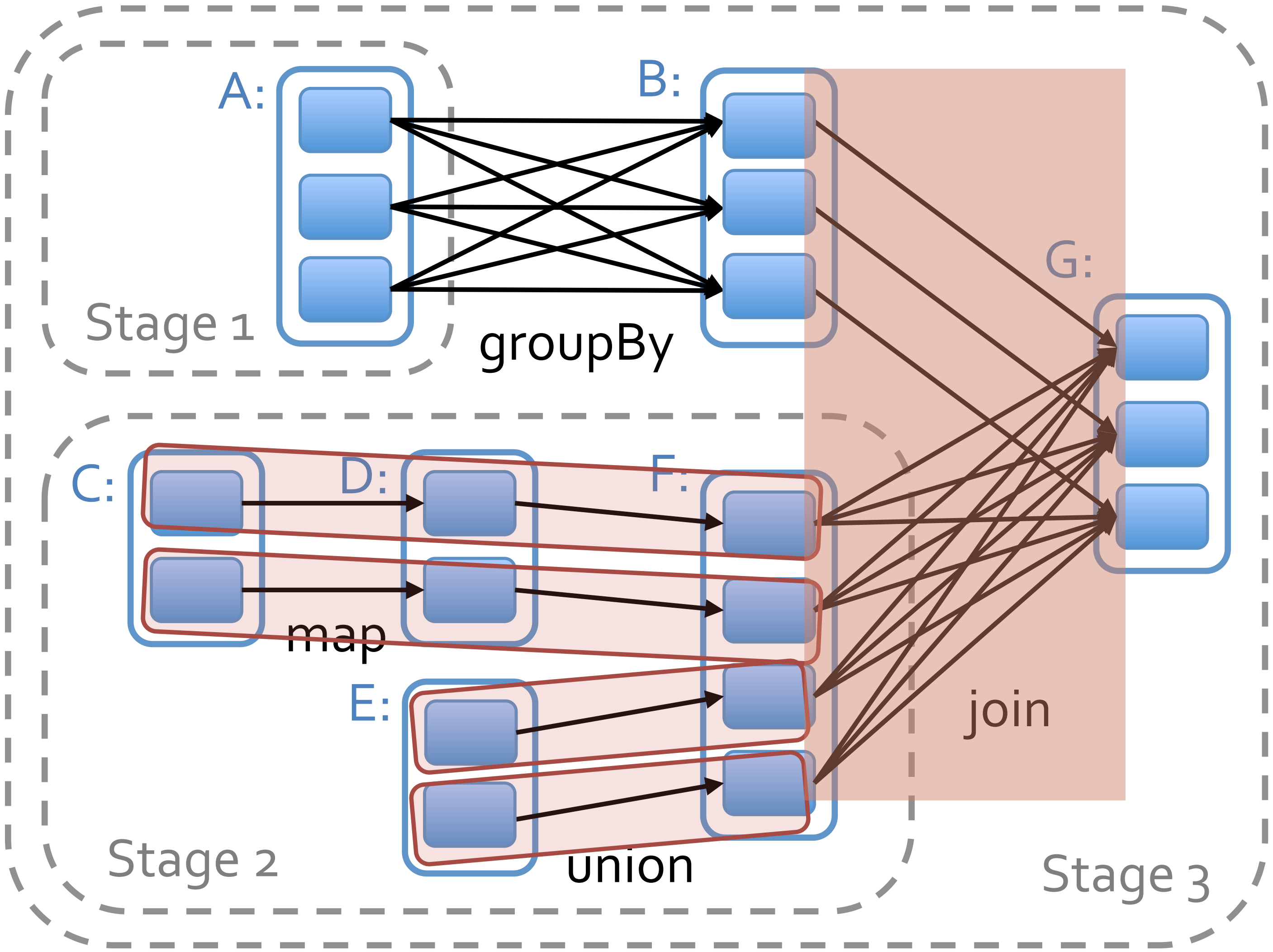
Within Stage Optimization, Pipelining the generation of RDD partitions when they are in narrow dependency



Scheduling Optimization

Within Stage Optimization, Pipelining the generation of RDD partitions when they are in narrow dependency

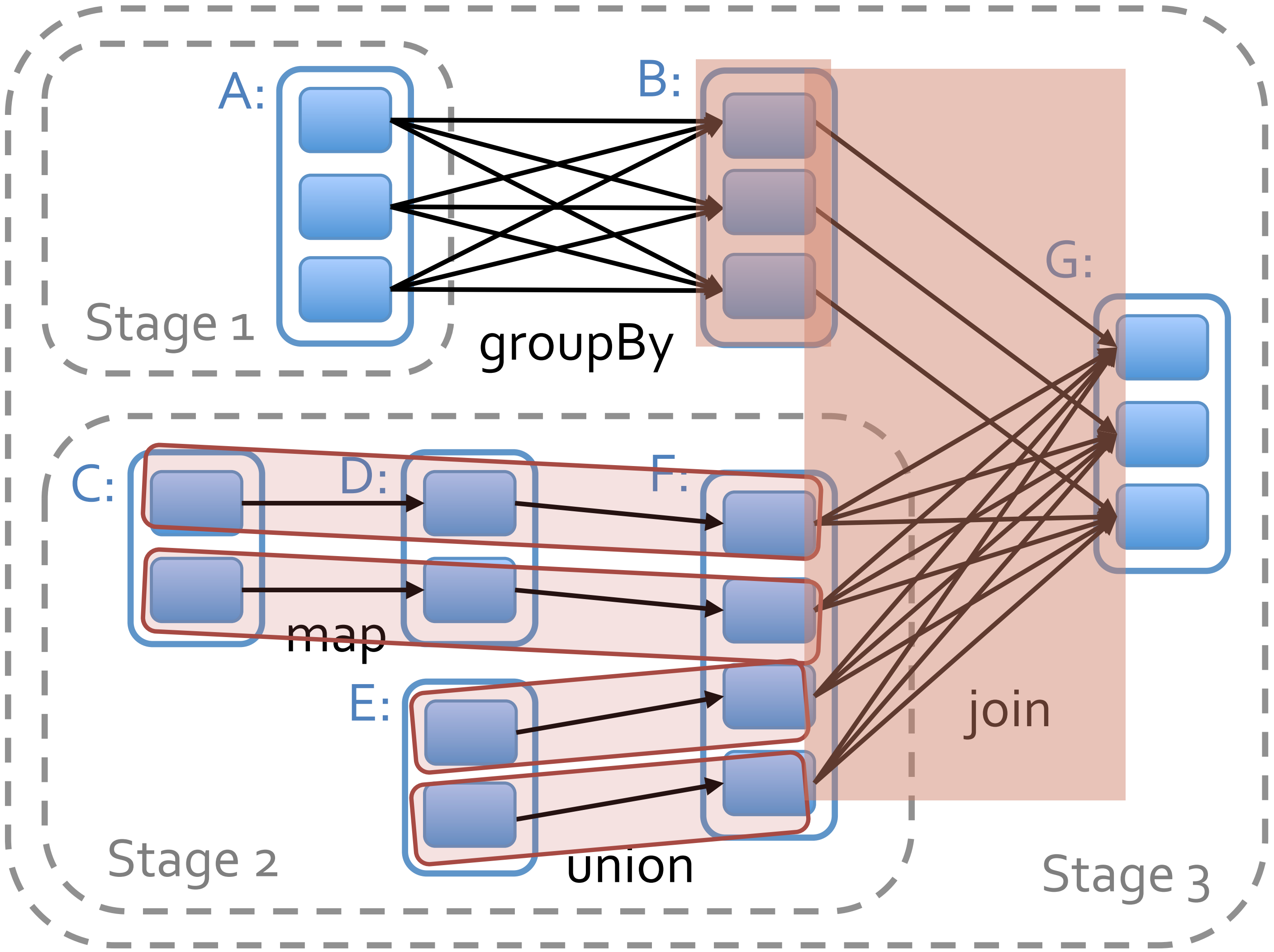
Partitioning-based join optimization, avoid whole-shuffle with best-efforts



Scheduling Optimization

Within Stage Optimization, Pipelining the generation of RDD partitions when they are in narrow dependency

Partitioning-based join optimization, avoid whole-shuffle with best-efforts

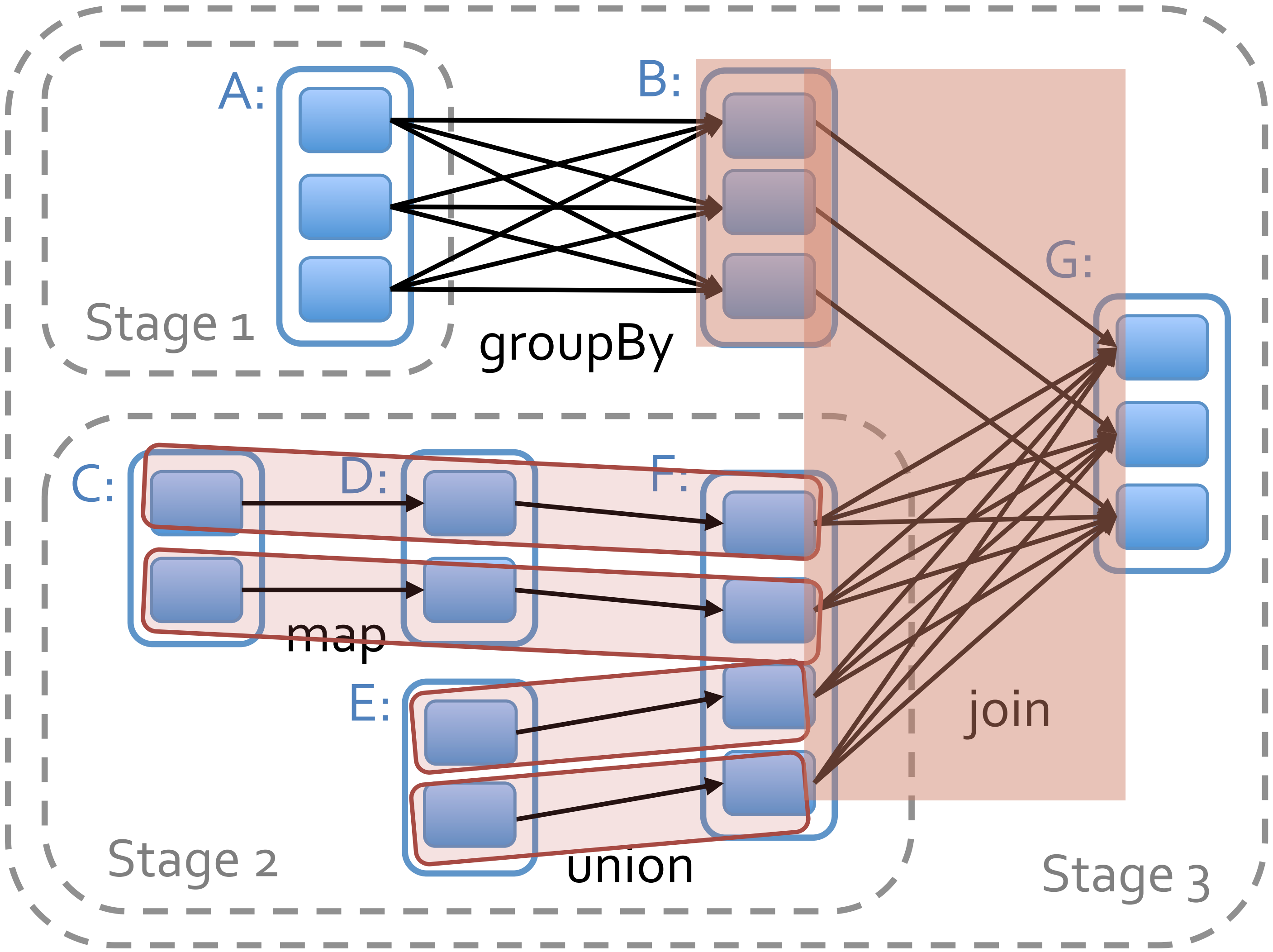


Scheduling Optimization

Within Stage Optimization, Pipelining the generation of RDD partitions when they are in narrow dependency

Partitioning-based join optimization, avoid whole-shuffle with best-efforts

Cache-aware to avoid duplicate computation



Summary

- No centralized application scheduler
  - Maximize Throughput
  - Application specific schedulers (DAGScheduler, TaskScheduler, ClusterScheduler) are initialized within SparkContext
- Scheduling Abstraction (DAG, TaskSet, Task)
  - Support fault-tolerance, pipelining, auto-recovery, etc.
- Scheduling Optimization
  - Pipelining, join, caching

FAIMDATA

We are hiring !

http://www.faimdata.com

nanzhu@faimdata.com

jobs@faimdata.com

Thank you!

Q & A

Credits to my friend, LianCheng@Databricks, his slides inspired me a lot