https://github.com/twitter/scalding
@Scalding

@knutwalker
@avantgardelabs

Agenda

Agenda

- Why **Scala** & **Scalding** ?

Agenda

- Why **Scala** & **Scalding** ?

- **Hadoop** + **Cascading**

Agenda

- Why **Scala** & **Scalding** ?
- **Hadoop** + **Cascading**
- Hello, **Scalding**

Why Scalding

"Hadoop is a distributed system for counting words"

Why Scalding

"Hadoop is a distributed system for counting words"

```
type Word    = String
```

Why Scalding

"Hadoop is a distributed system for counting words"

```
type Word   = String
type Count  = Int
```

Why Scalding

"Hadoop is a distributed system for counting words"

```
type Word   = String
type Count  = Int
```

Why Scalding

"Hadoop is a distributed system for counting words"

```
type Word    = String
type Count   = Int

type WordCount =
```

Why Scalding

"Hadoop is a distributed system for counting words"

```
type Word    = String
type Count   = Int

type WordCount =
  Word => Map[Word, Count]
```

Why Scalding

WordCount in Scala

Why Scalding

WordCount in Scala

```
val wordCount: WordCount = text => text
```

Why Scalding

WordCount in Scala

```
val wordCount: WordCount = text => text
  .split("[,.\\s]+")    // Splitting
```

Why Scalding

WordCount in Scala

```
val wordCount: WordCount = text => text
  .split("[,.\\s]+")    // Splitting
  .map(w => (w, 1))     // Mapping
```

Why Scalding

WordCount in Scala

```
val wordCount: WordCount = text => text
  .split("[,.\\s]+")    // Splitting
  .map(w => (w, 1))     // Mapping
  .groupBy(w => w._1)   // Shuffling
```

Why Scalding

WordCount in Scala

```
val wordCount: WordCount = text => text
  .split("[,.\\s]+")    // Splitting
.map(w => (w, 1))       // Mapping
.groupBy(w => w._1)     // Shuffling
.map { case (word, counts) =>
```

Why Scalding

WordCount in Scala

```
val wordCount: WordCount = text => text
  .split("[,.\\s]+")    // Splitting
.map(w => (w, 1))       // Mapping
.groupBy(w => w._1)     // Shuffling
.map { case (word, counts) =>
  word -> counts.map(_._2).sum) }
```

Why Scalding

WordCount in Scala

```
val wordCount: WordCount = text => text
  .split("[,.\\s]+")    // Splitting
  .map(w => (w, 1))     // Mapping
  .groupBy(w => w._1)   // Shuffling
  .map { case (word, counts) =>
    word -> counts.map(_._2).sum) }
                        // Reducing
```

Why Scalding

```
val text = "Wenn hinter Fliegen
Fliegen fliegen, fliegen Fliegen
Fliegen hinter her."
```

```
assert(wordCount(text) == Map(
  "Wenn"     -> 1,
  "hinter"   -> 2,
  "Fliegen"  -> 4,
  "fliegen"  -> 2,
  "her"      -> 1 ))
```

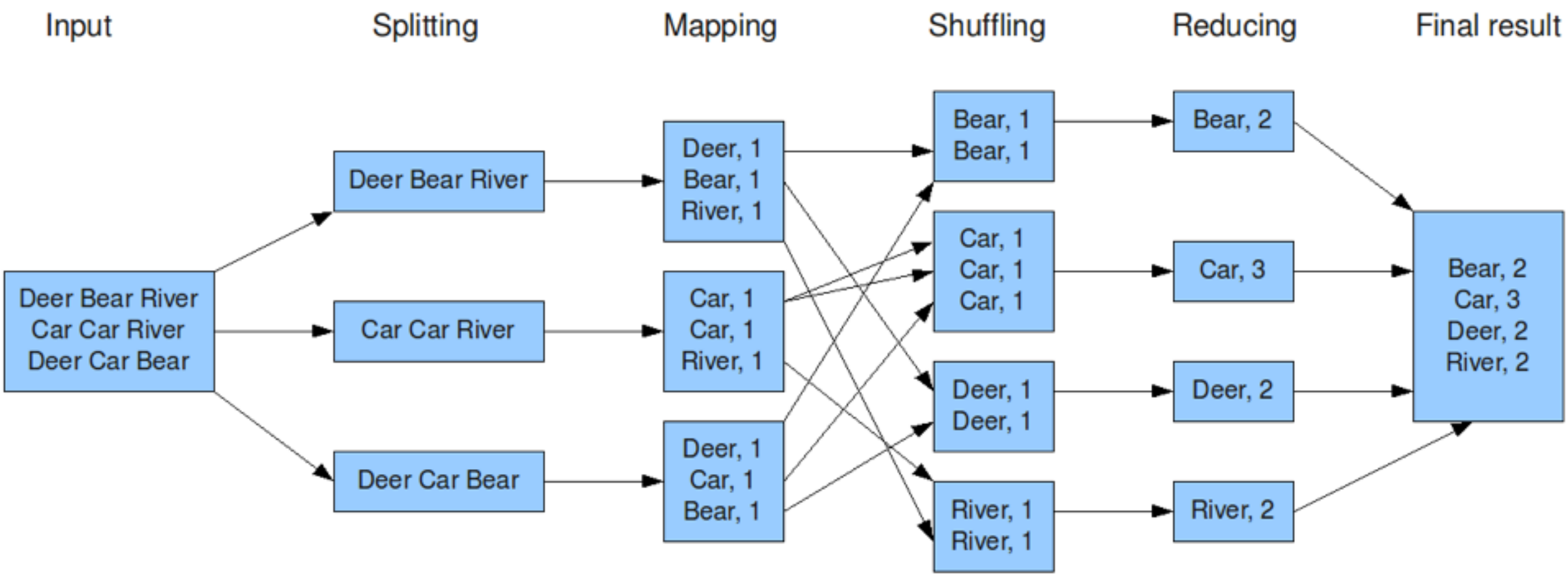
Why Scalding

```
val wordCount: WordCo                                    xt
  .split("[,.\\s]+")
  .map(w => (w, 1))                                  Mapping
.groupBy(w => w._1)                          // Shuffling
.map { case (word, counts) =>
  word -> counts.map(_._2).sum) }
                                             // Reducing
```

In memory!

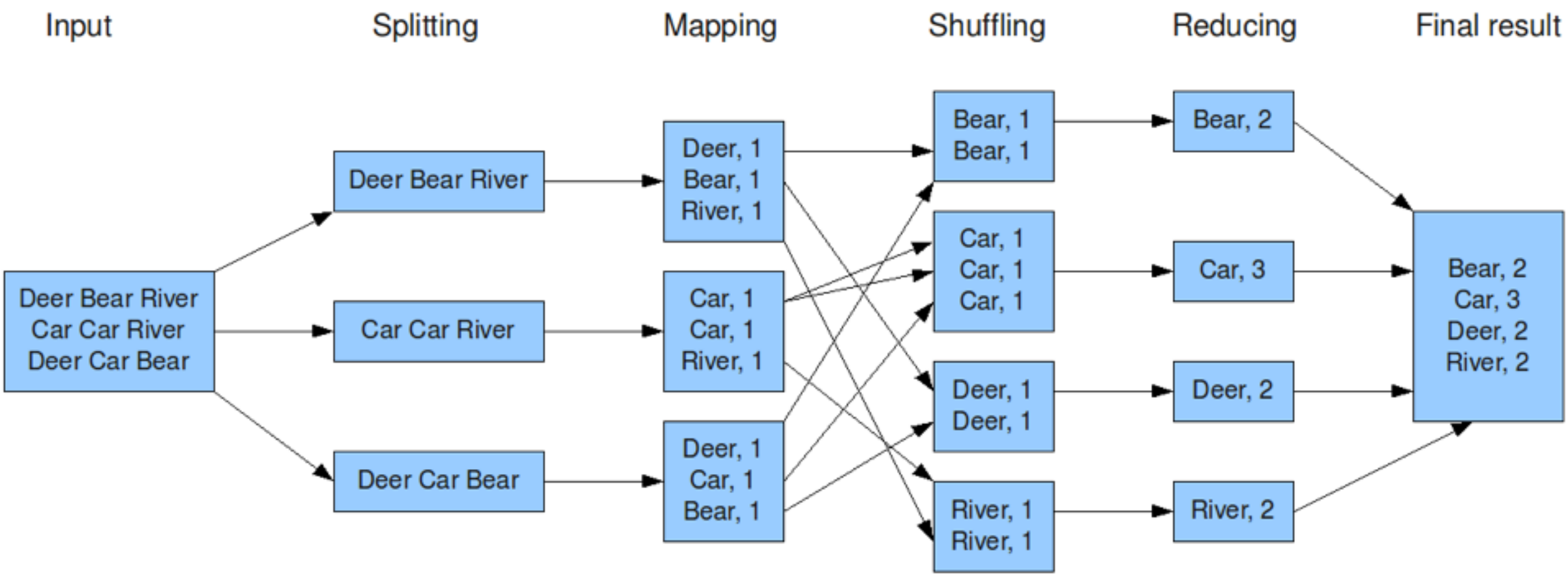
http://hadoop.apache.org/

Hadoop



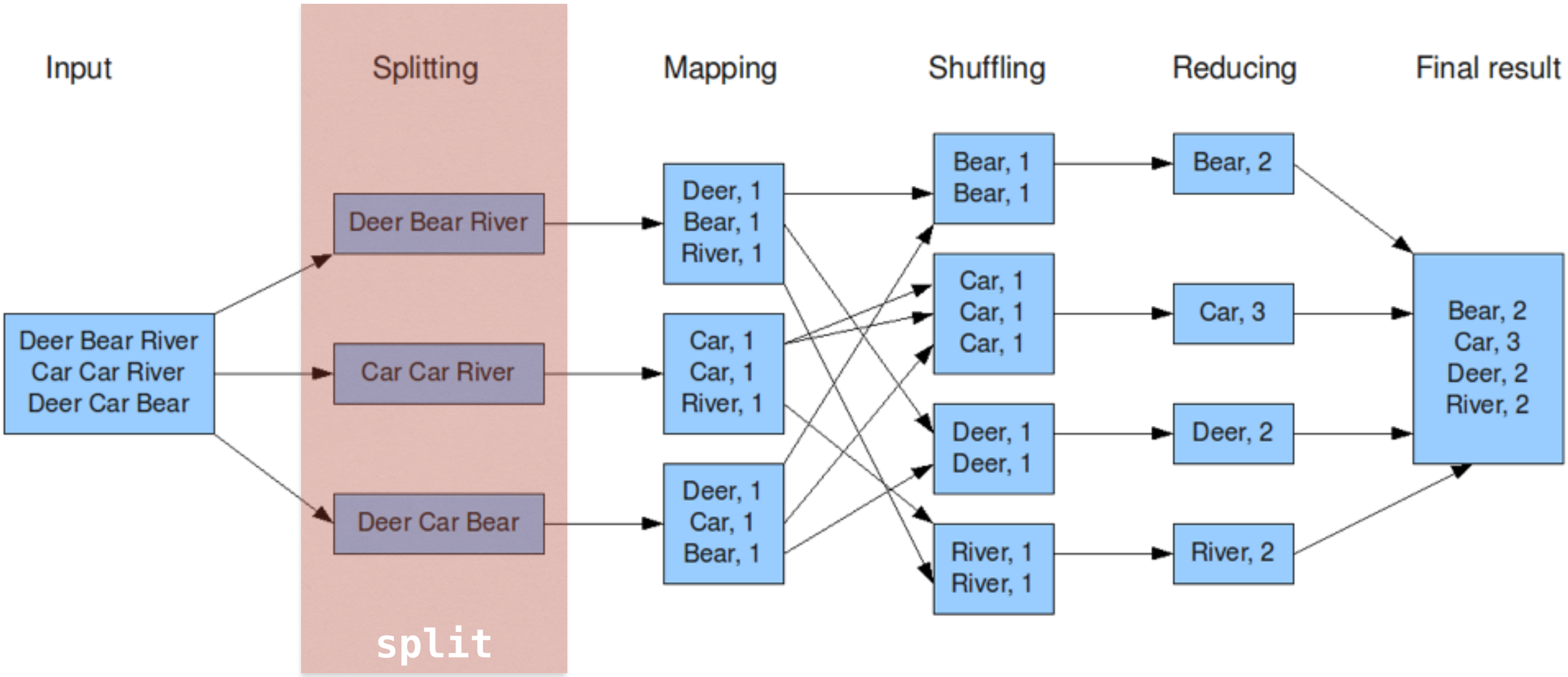
http://xiaochongzhang.me/blog/?p=338

Hadoop



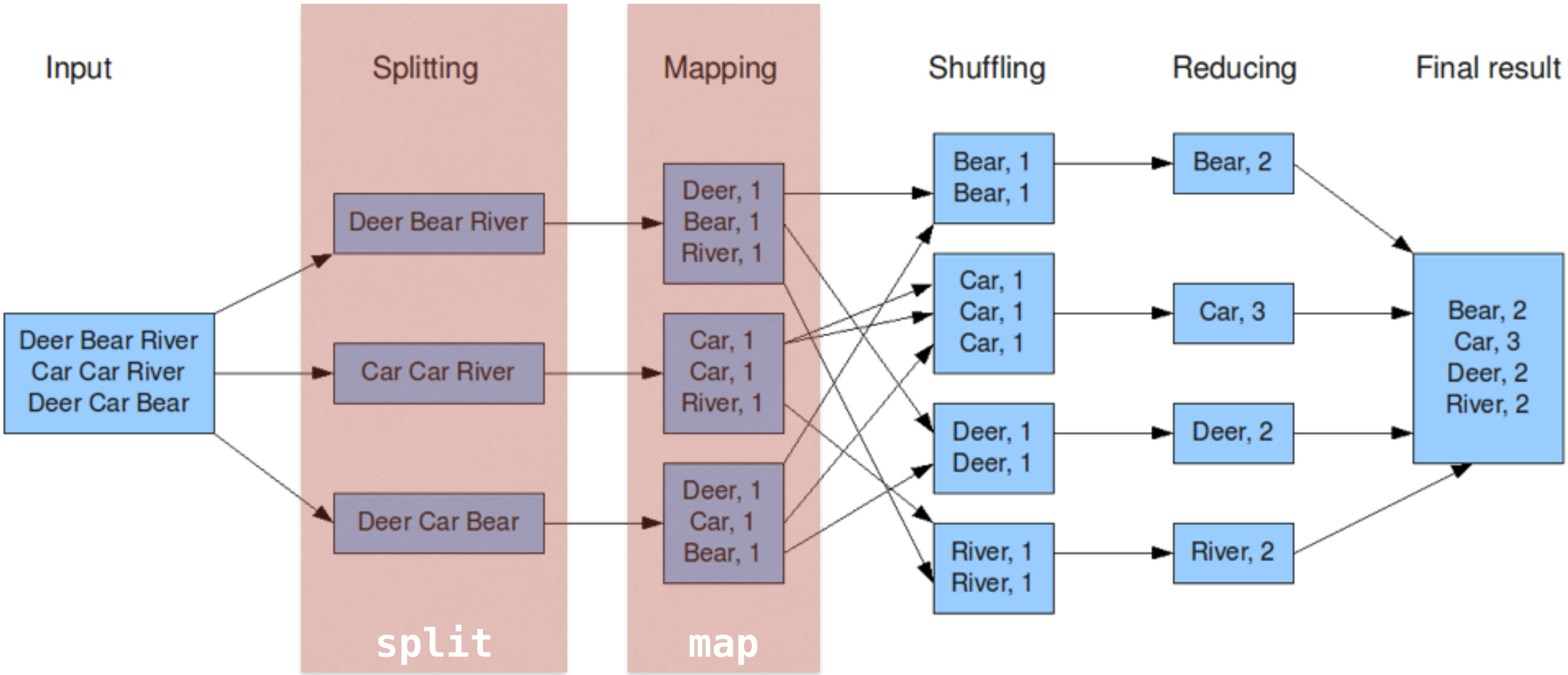
http://xiaochongzhang.me/blog/?p=338

Hadoop



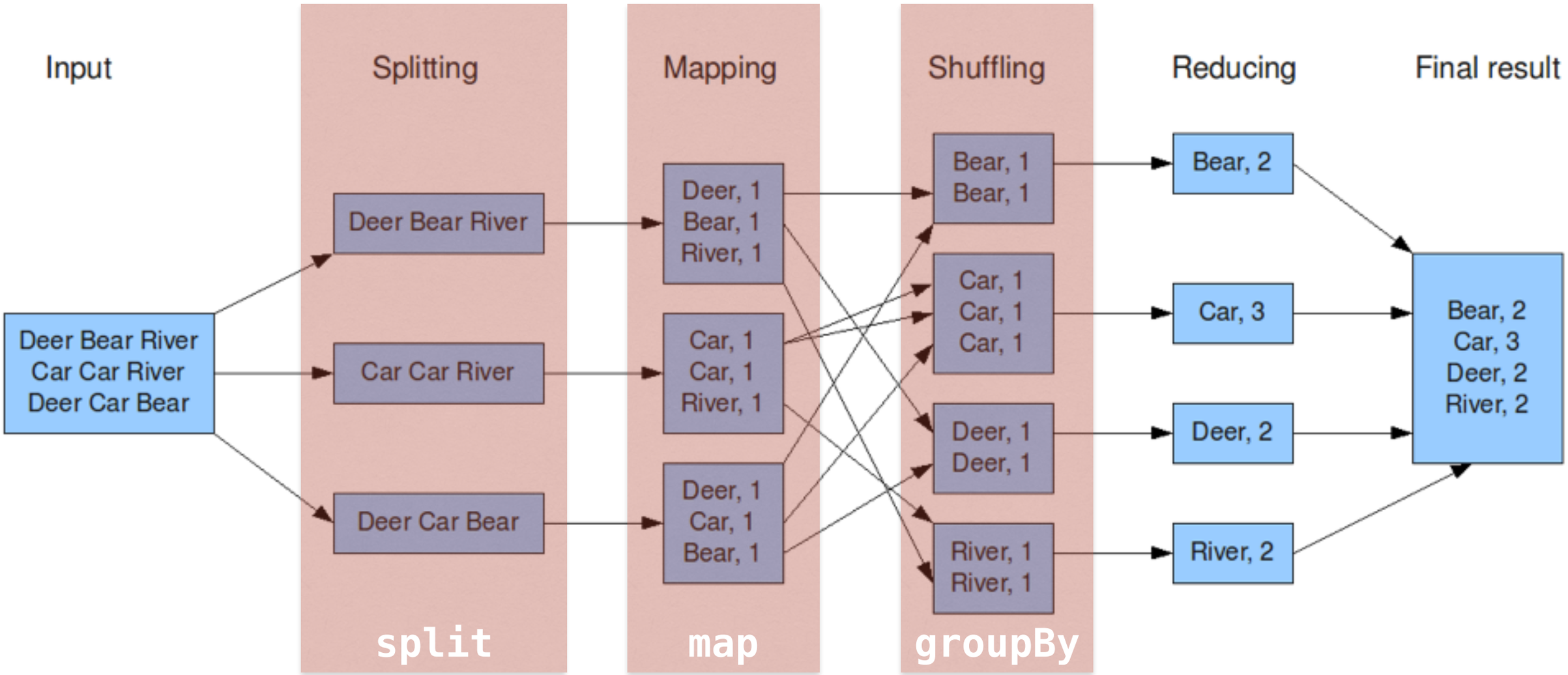
http://xiaochongzhang.me/blog/?p=338

Hadoop



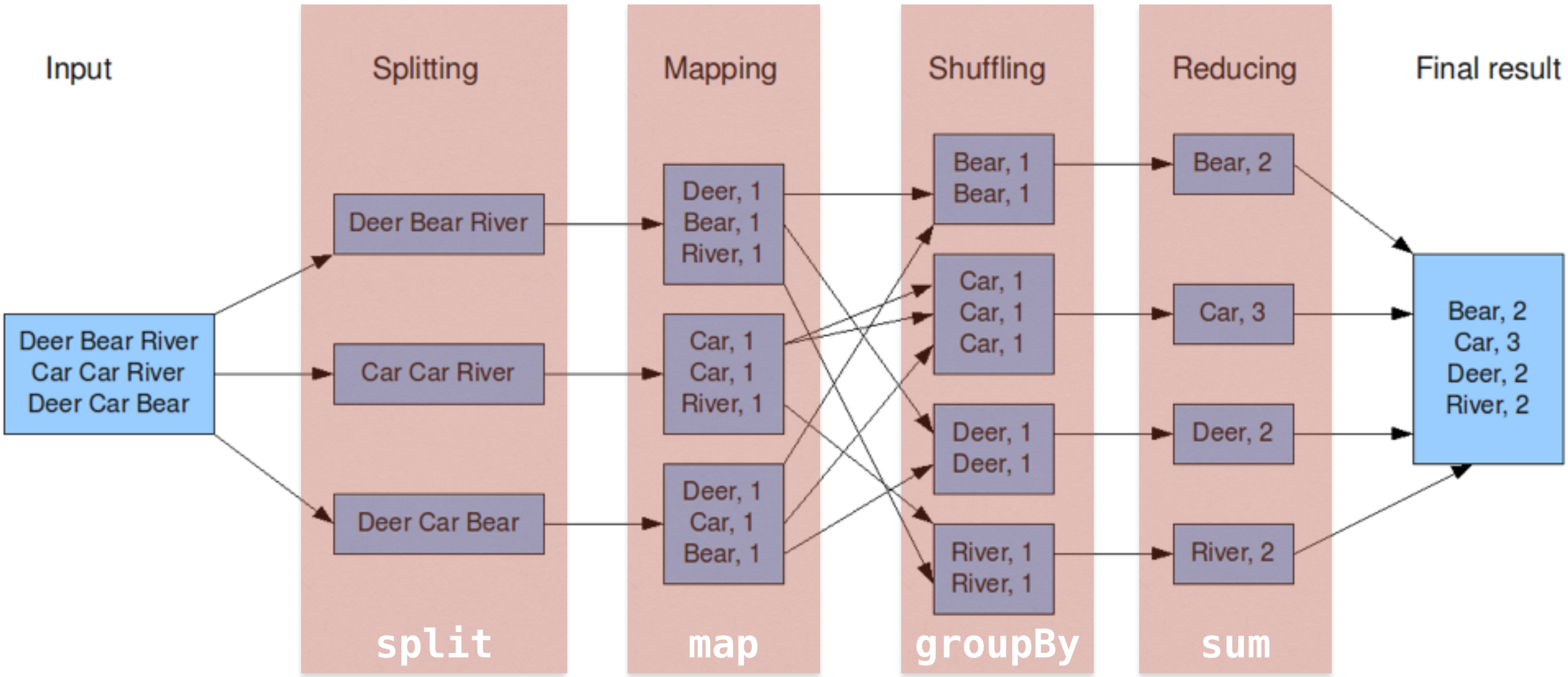
http://xiaochongzhang.me/blog/?p=338

Hadoop



http://xiaochongzhang.me/blog/?p=338

Hadoop



http://xiaochongzhang.me/blog/?p=338

```java
package org.apache.hadoop.examples;

import java.io.IOException;
import java.util.StringTokenizer;

import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.Mapper;
import org.apache.hadoop.mapreduce.Reducer;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
import org.apache.hadoop.util.GenericOptionsParser;

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);
      }
    }
  }

  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);
    }
  }

  public static void main(String[] args) throws Exception {
    Configuration conf = new Configuration();
    String[] otherArgs = new GenericOptionsParser(conf, args).getRemainingArgs();
    if (otherArgs.length != 2) {
      System.err.println("Usage: wordcount <in> <out>");
      System.exit(2);
    }
    Job job = new Job(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(otherArgs[0]));
    FileOutputFormat.setOutputPath(job, new Path(otherArgs[1]));
    System.exit(job.waitForCompletion(true) ? 0 : 1);
  }
}
```

```java
package org.apache.hadoop.examples;

import java.io.IOException;
import java.util.StringTokenizer;

import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.Mapper;
import org.apache.hadoop.mapreduce.Reducer;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
import org.apache.hadoop.util.GenericOptionsParser;

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);
      }
    }
  }

  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);
    }
  }

  public static void main(String[] args) throws Exception {
    Configuration conf = new Configuration();
    String[] otherArgs = new GenericOptionsParser(conf, args).getRemainingArgs();
    if (otherArgs.length != 2) {
      System.err.println("Usage: wordcount <in> <out>");
      System.exit(2);
    }
    Job job = new Job(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(otherArgs[0]));
    FileOutputFormat.setOutputPath(job, new Path(otherArgs[1]));
    System.exit(job.waitForCompletion(true) ? 0 : 1);
  }
}
```

Splitting

```java
package org.apache.hadoop.examples;

import java.io.IOException;
import java.util.StringTokenizer;

import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.Mapper;
import org.apache.hadoop.mapreduce.Reducer;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
import org.apache.hadoop.util.GenericOptionsParser;

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);
      }
    }
  }

  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);
    }
  }

  public static void main(String[] args) throws Exception {
    Configuration conf = new Configuration();
    String[] otherArgs = new GenericOptionsParser(conf, args).getRemainingArgs();
    if (otherArgs.length != 2) {
      System.err.println("Usage: wordcount <in> <out>");
      System.exit(2);
    }
    Job job = new Job(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(otherArgs[0]));
    FileOutputFormat.setOutputPath(job, new Path(otherArgs[1]));
    System.exit(job.waitForCompletion(true) ? 0 : 1);
  }
}
```

Mapping

Splitting

Mapping

Splitting

Reducing

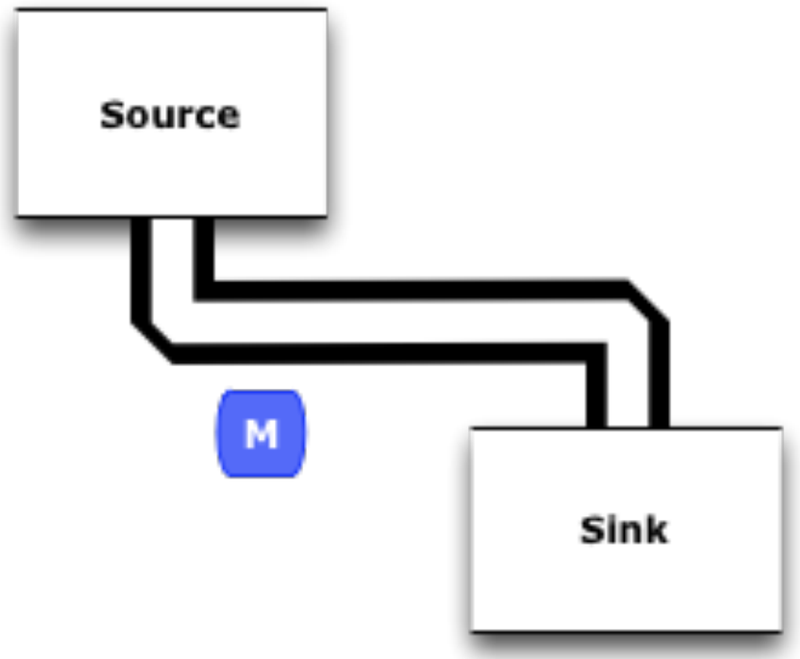
```java
package org.apache.hadoop.examples;

import java.io.IOException;
import java.util.StringTokenizer;

import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.Mapper;
import org.apache.hadoop.mapreduce.Reducer;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
import org.apache.hadoop.util.GenericOptionsParser;

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);
      }
    }
  }

  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);
    }
  }

  public static void main(String[] args) throws Exception {
    Configuration conf = new Configuration();
    String[] otherArgs = new GenericOptionsParser(conf, args).getRemainingArgs();
    if (otherArgs.length != 2) {
      System.err.println("Usage: wordcount <in> <out>");
      System.exit(2);
    }
    Job job = new Job(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(otherArgs[0]));
    FileOutputFormat.setOutputPath(job, new Path(otherArgs[1]));
    System.exit(job.waitForCompletion(true) ? 0 : 1);
  }
}
```

http://www.cascading.org/

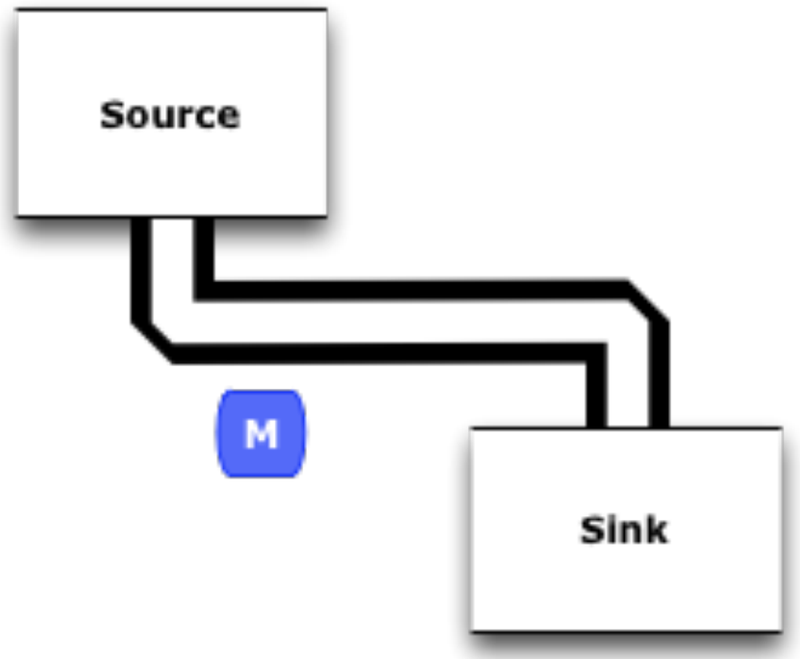
Cascading

Cascading

Cascading

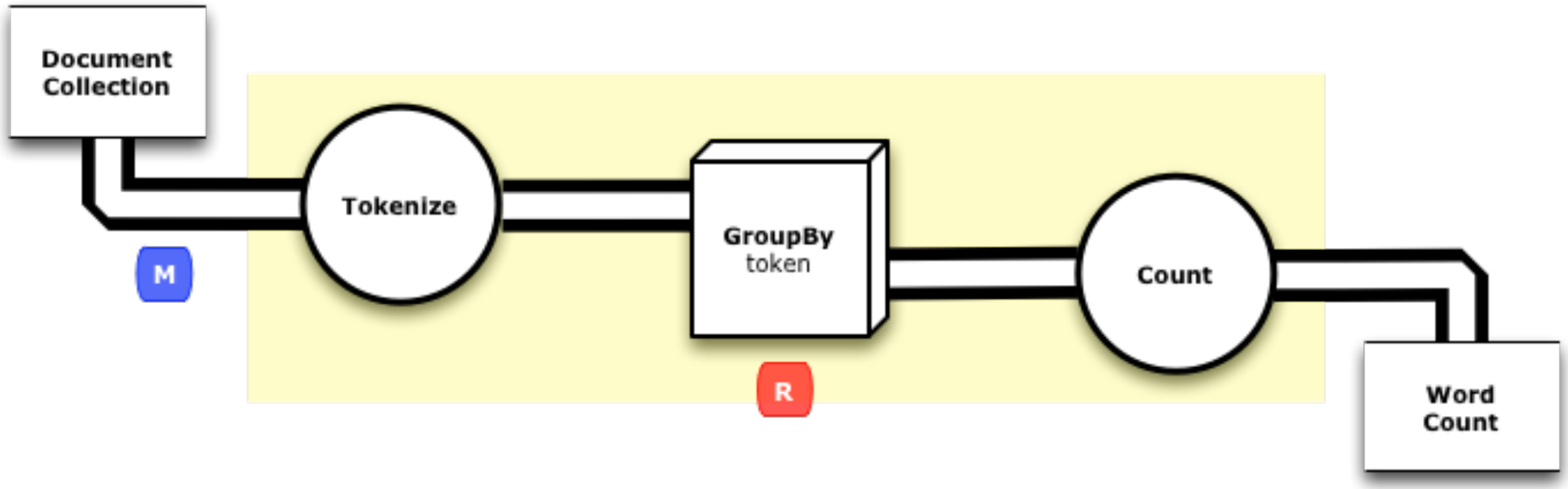


Cascading



Cascading

2: Word Count



Cascading

```
Properties properties = new Properties();
AppProps.setApplicationJarClass( properties, Main.class );
HadoopFlowConnector flowConnector = new HadoopFlowConnector( properties );

// create source and sink taps
Tap docTap = new Hfs( new TextDelimited( true, "\t" ), docPath );
Tap wcTap = new Hfs( new TextDelimited( true, "\t" ), wcPath );

// specify a regex operation to split the "document" text lines into a token stream
Fields token = new Fields( "token" );
Fields text = new Fields( "text" );
RegexSplitGenerator splitter = new RegexSplitGenerator( token, "[ \\[\\]\\(\\),.]" );
// only returns "token"
Pipe docPipe = new Each( "token", text, splitter, Fields.RESULTS );
```

Cascading

```
Properties properties = new Properties();
AppProps.setApplicationJarClass( properties, Main.class );
HadoopFlowConnector flowConnector = new HadoopFlowConnector( properties );

// create source and sink taps
Tap docTap = new Hfs( new TextDelimited( true, "\t" ), docPath );
Tap wcTap = new Hfs( new TextDelimited( true, "\t" ), wcPath );
```

Taps

```
// specify a regex operation to split the "document" text lines into a token stream
Fields token = new Fields( "token" );
Fields text = new Fields( "text" );
RegexSplitGenerator splitter = new RegexSplitGenerator( token, "[ \\[\\]\\(\\),.]" );
// only returns "token"
Pipe docPipe = new Each( "token", text, splitter, Fields.RESULTS );
```

Cascading

```
Properties properties = new Properties();
AppProps.setApplicationJarClass( properties, Main.class );
HadoopFlowConnector flowConnector = new HadoopFlowConnector( properties );

// create source and sink taps
Tap docTap = new Hfs( new TextDelimited( true, "\t" ), docPath );
Tap wcTap = new Hfs( new TextDelimited( true, "\t" ), wcPath );

// specify a regex operation to split the "document" text lines into a token stream
Fields token = new Fields( "token" );
Fields text = new Fields( "text" );
RegexSplitGenerator splitter = new RegexSplitGenerator( token, "[ \\[\\]\\(\\),.]" );
// only returns "token"
Pipe docPipe = new Each( "token", text, splitter, Fields.RESULTS );
```

Splitting

Cascading

```
Properties properties = new Properties();
AppProps.setApplicationJarClass( properties, Main.class );
HadoopFlowConnector flowConnector = new HadoopFlowConnector( properties );

// create source and sink taps
Tap docTap = new Hfs( new TextDelimited( true, "\t" ), docPath );
Tap wcTap = new Hfs( new TextDelimited( true, "\t" ), wcPath );

// specify a regex operation to split the "document" text lines into a token stream
Fields token = new Fields( "token" );
Fields text = new Fields( "text" );
RegexSplitGenerator splitter = new RegexSplitGenerator( token, "[ \\[\\]\\(\\),.]" );
// only returns "token"
Pipe docPipe = new Each( "token", text, splitter, Fields.RESULTS );
```

Splitting

Cascading

```
// determine the word counts
Pipe wcPipe = new Pipe( "wc", docPipe );
wcPipe = new GroupBy( wcPipe, token );
wcPipe = new Every( wcPipe, Fields.ALL, new Count(), Fields.ALL );

// connect the taps, pipes, etc., into a flow
FlowDef flowDef = FlowDef.flowDef()
 .setName( "wc" )
 .addSource( docPipe, docTap )
 .addTailSink( wcPipe, wcTap );

// write a DOT file and run the flow
Flow wcFlow = flowConnector.connect( flowDef );
wcFlow.writeDOT( "dot/wc.dot" );
wcFlow.complete();
```

Cascading

```
// determine the word counts
Pipe wcPipe = new Pipe( "wc", docPipe );
wcPipe = new GroupBy( wcPipe, token );
wcPipe = new Every( wcPipe, Fields.ALL, new Count(), Fields.ALL );
```

Mapping + Shuffling

```
// connect the taps, pipes, etc., into a flow
FlowDef flowDef = FlowDef.flowDef()
  .setName( "wc" )
  .addSource( docPipe, docTap )
  .addTailSink( wcPipe, wcTap );

// write a DOT file and run the flow
Flow wcFlow = flowConnector.connect( flowDef );
wcFlow.writeDOT( "dot/wc.dot" );
wcFlow.complete();
```

Cascading

```
// determine the word counts
Pipe wcPipe = new Pipe( "wc", docPipe );
wcPipe = new GroupBy( wcPipe, token );
wcPipe = new Every( wcPipe, Fields.ALL, new Count(), Fields.ALL );
```

Reducing

```
// connect the taps, pipes, etc., into a flow
FlowDef flowDef = FlowDef.flowDef()
  .setName( "wc" )
  .addSource( docPipe, docTap )
  .addTailSink( wcPipe, wcTap );

// write a DOT file and run the flow
Flow wcFlow = flowConnector.connect( flowDef );
wcFlow.writeDOT( "dot/wc.dot" );
wcFlow.complete();
```

Cascading

```
// determine the word counts
Pipe wcPipe = new Pipe( "wc", docPipe );
wcPipe = new GroupBy( wcPipe, token );
wcPipe = new Every( wcPipe, Fields.ALL, new Count(), Fields.ALL );
```

Reducing

```
// connect the taps, pipes, etc., into a flow
FlowDef flowDef = FlowDef.flowDef()
  .setName( "wc" )
  .addSource( docPipe, docTap )
  .addTailSink( wcPipe, wcTap );

// write a DOT file and run the flow
Flow wcFlow = flowConnector.connect( flowDef );
wcFlow.writeDOT( "dot/wc.dot" );
wcFlow.complete();
```

=

+

http://github.com/twitter/scalding

Scalding

```
IterableSource(List(text), 'line)
    .flatMap('line -> 'word) {
      line: String =>
        line.split("[,.\\s]+") }
  .groupBy('word) {
      group => group.size('count) }
  .write(Tsv("output.txt"))
```

Scalding

```
val text = "Wenn hinter Fliegen
Fliegen fliegen, fliegen Fliegen
Fliegen hinter her."
```

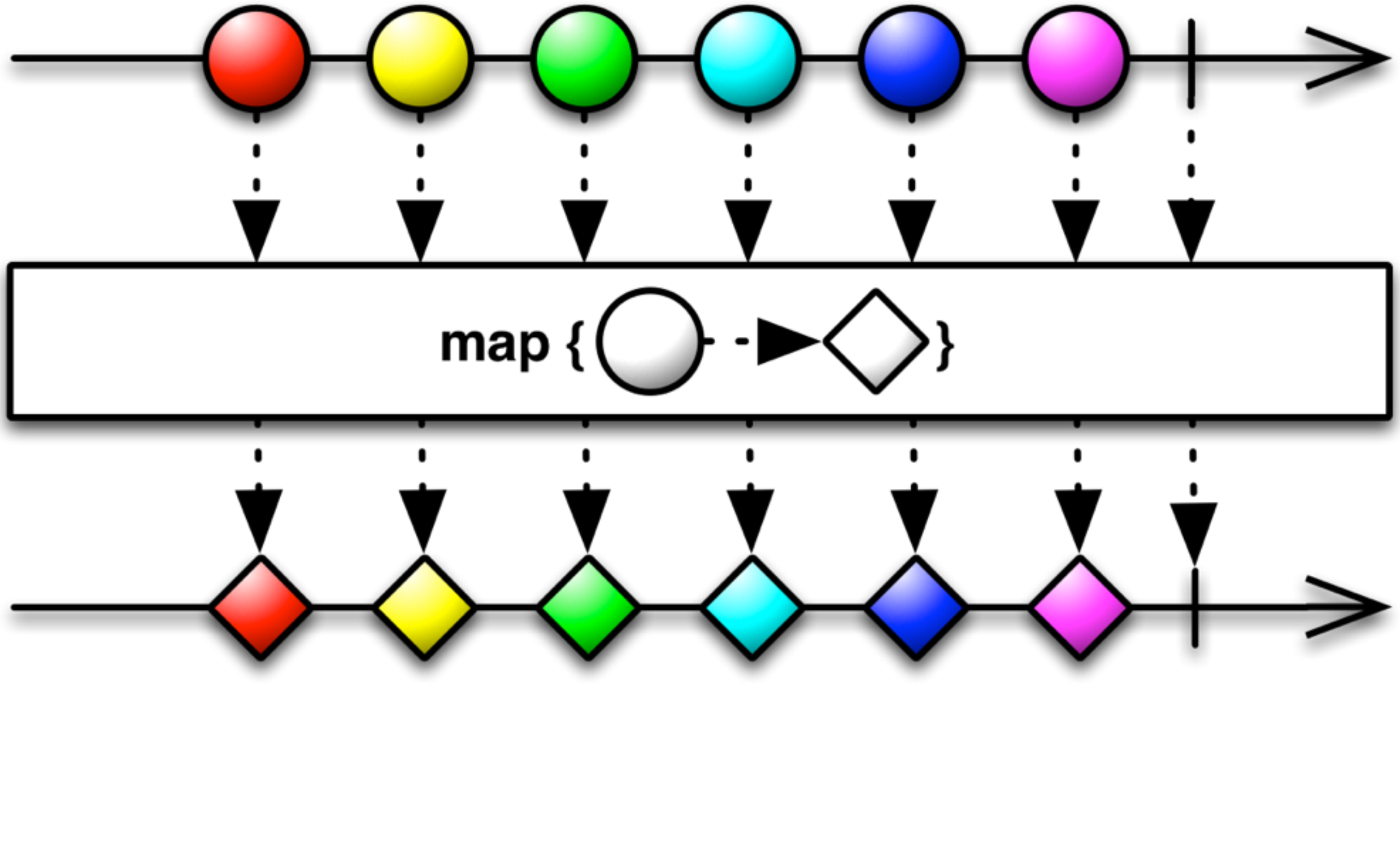
```
Wenn      -> 1
hinter    -> 2
Fliegen   -> 4
fliegen   -> 2
her       -> 1
```

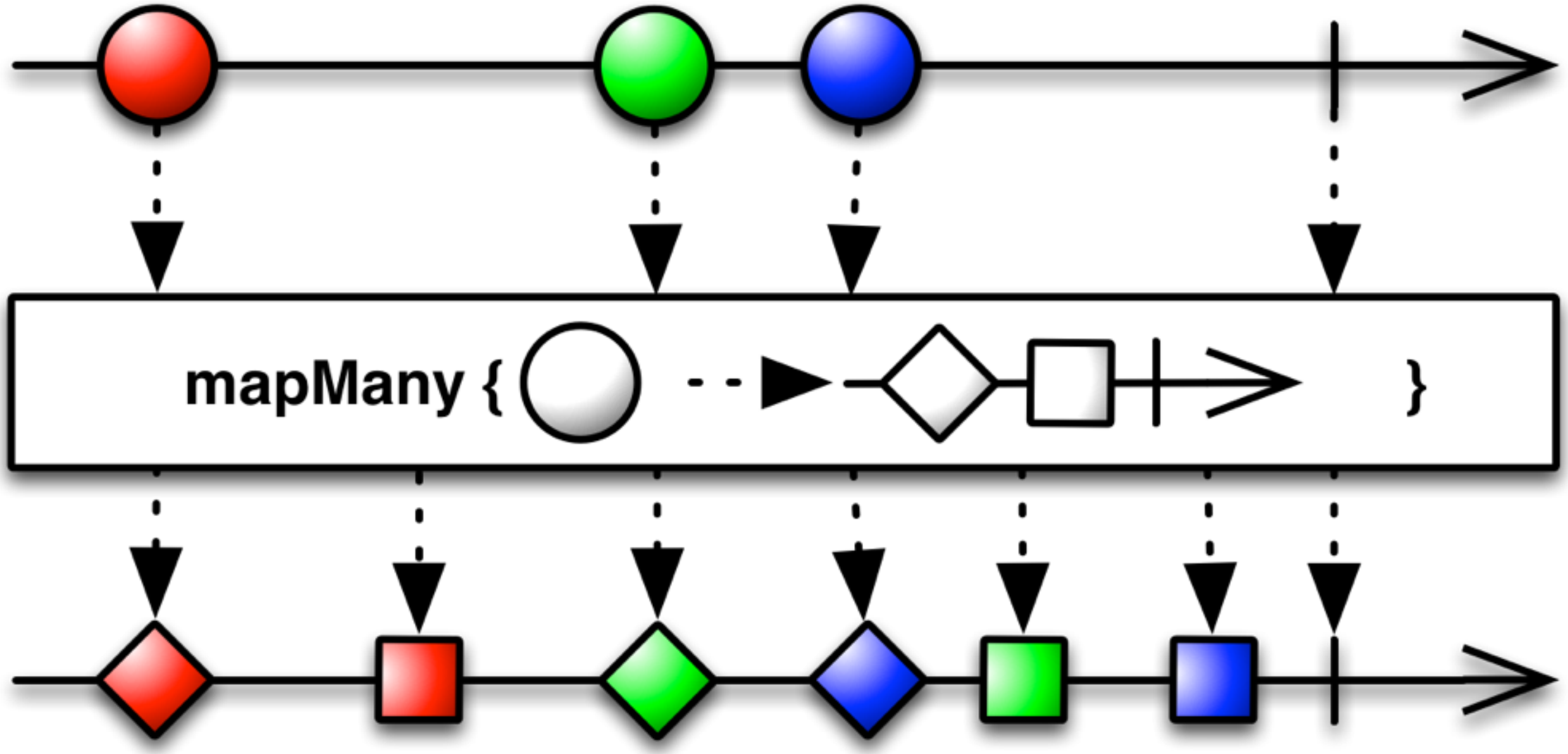
Scalding

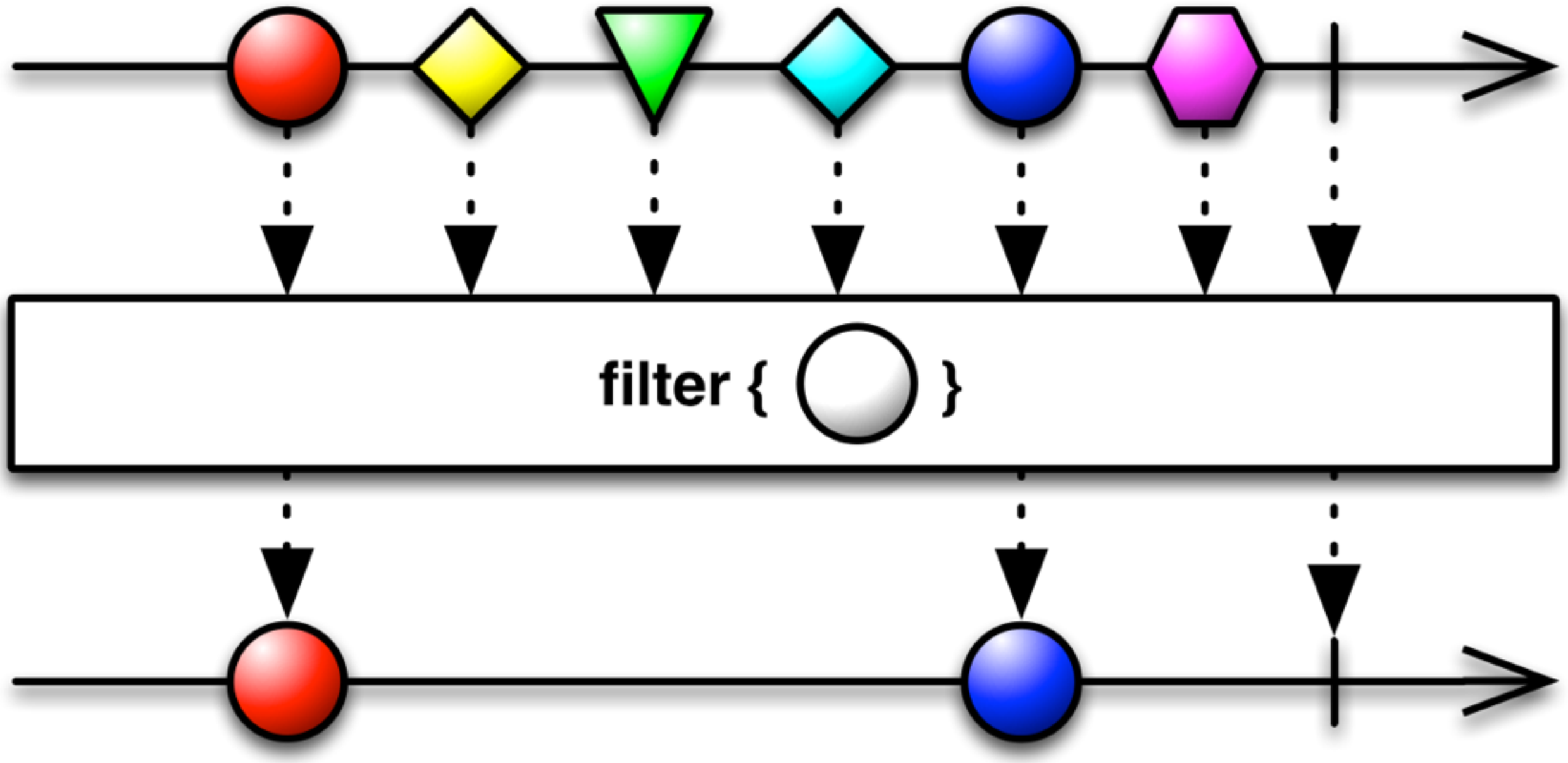
```
class CountWords(args: Args) extends
Job(args) {

  TextLine(args("input"))
     .read
     .flatMap('line -> 'word) { line:
String => tokenize(line) }
     .groupBy('word){ _.size }
     .write(Tsv(args("output")))

}
```

Scalding Functions

- **Map**-like
  - `map(To), flatMap(To)`
  - `project, discard`
  - `insert, rename, limit`
  - `filter, unique`

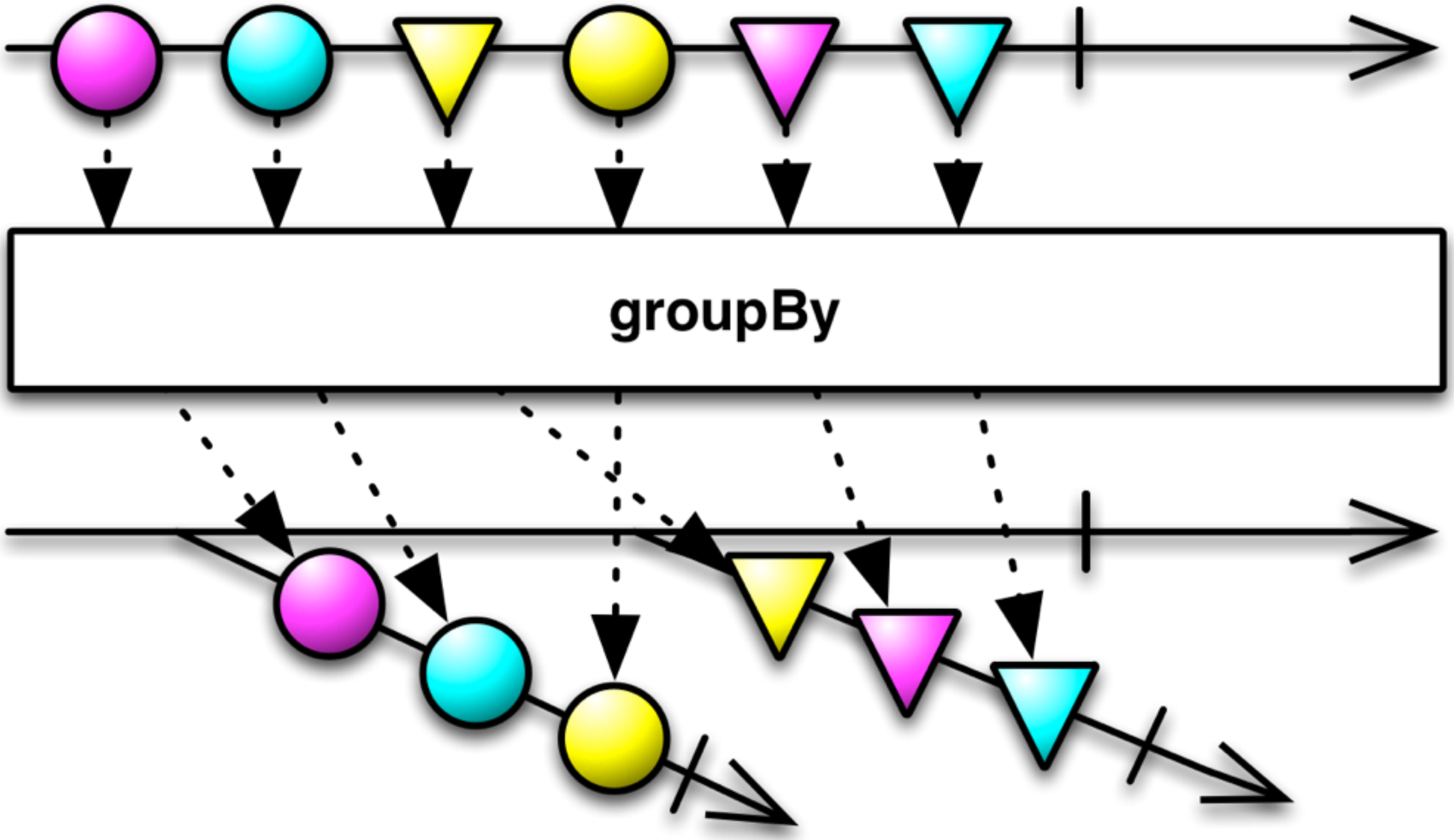

map {


mapMany {
}


filter { }

Scalding Functions

- **Grouping** functions
  - `groupBy`
  - `groupAll`
  - *`GroupBuilder`*


groupBy

Scalding Functions

- **Grouping** operations
  - size, average, sizeAveStdev
  - sum, max, min, count
  - sortBy, take, drop, sortWithTake


JOINS

Scalding Functions

- **Join** operations
  - `joinWithSmaller`
  - joinWithLarger
  - joinWithTiny
  - *Inner/Left/Right/Outer*

```
// `people` is a large pipe with a "birthCityId" field.
// Join it against the smaller `cities` pipe, which contains an "id" field.
val peopleWithBirthplaces = people.joinWithSmaller('birthCityId -> 'id, cities)

// Join on both city.id and state.id
val peopleWithBirthplaces = people.joinWithSmaller( ('birthCityId , 'birthStateID) ->
    ('id,'StateID) , cities)


// `cities` is a small pipe with an "id" field.
// Join it against the larger `people` pipe, which contains a "birthCityId" field.
val peopleWithBirthplaces = cities.joinWithLarger('id -> 'birthCityId, people)



import cascading.pipe.joiner._

people.joinWithSmaller('birthCityId -> 'id, cities, joiner = new LeftJoin)
people.joinWithSmaller('birthCityId -> 'id, cities, joiner = new RightJoin)
people.joinWithSmaller('birthCityId -> 'id, cities, joiner = new OuterJoin)



// This is allowed. Only a single "ssn" field will be left in the resulting merged
pipe.
people.joinWithSmaller('ssn -> 'ssn, teachers)
// Instead
people.joinWithSmaller('ssn_left -> 'ssn_right, teachers)
// Both fields are kept after the join.
```

???

https://github.com/avantgarde-labs/bdug-dd-scalding