Introduction to the **Spark ecosystem**

Federico Feroldi

@cloudify

LambdaCon 2015

How **big** is your **data**?

SAY BIG DATA
ONE MORE TIME
memegenerator.n

How ~~big is~~ your ~~data~~?

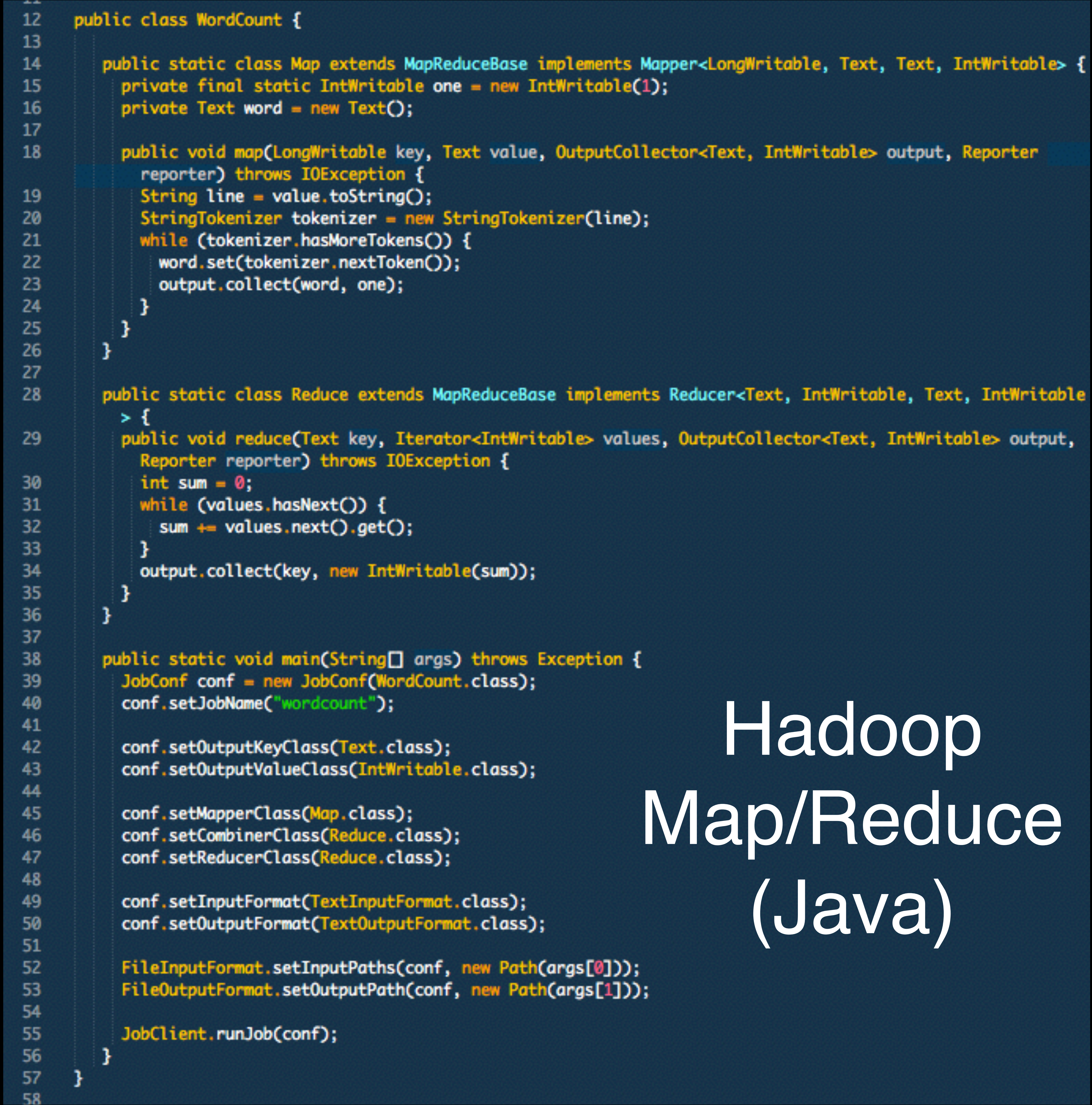
fast is computer

Map/Reduce

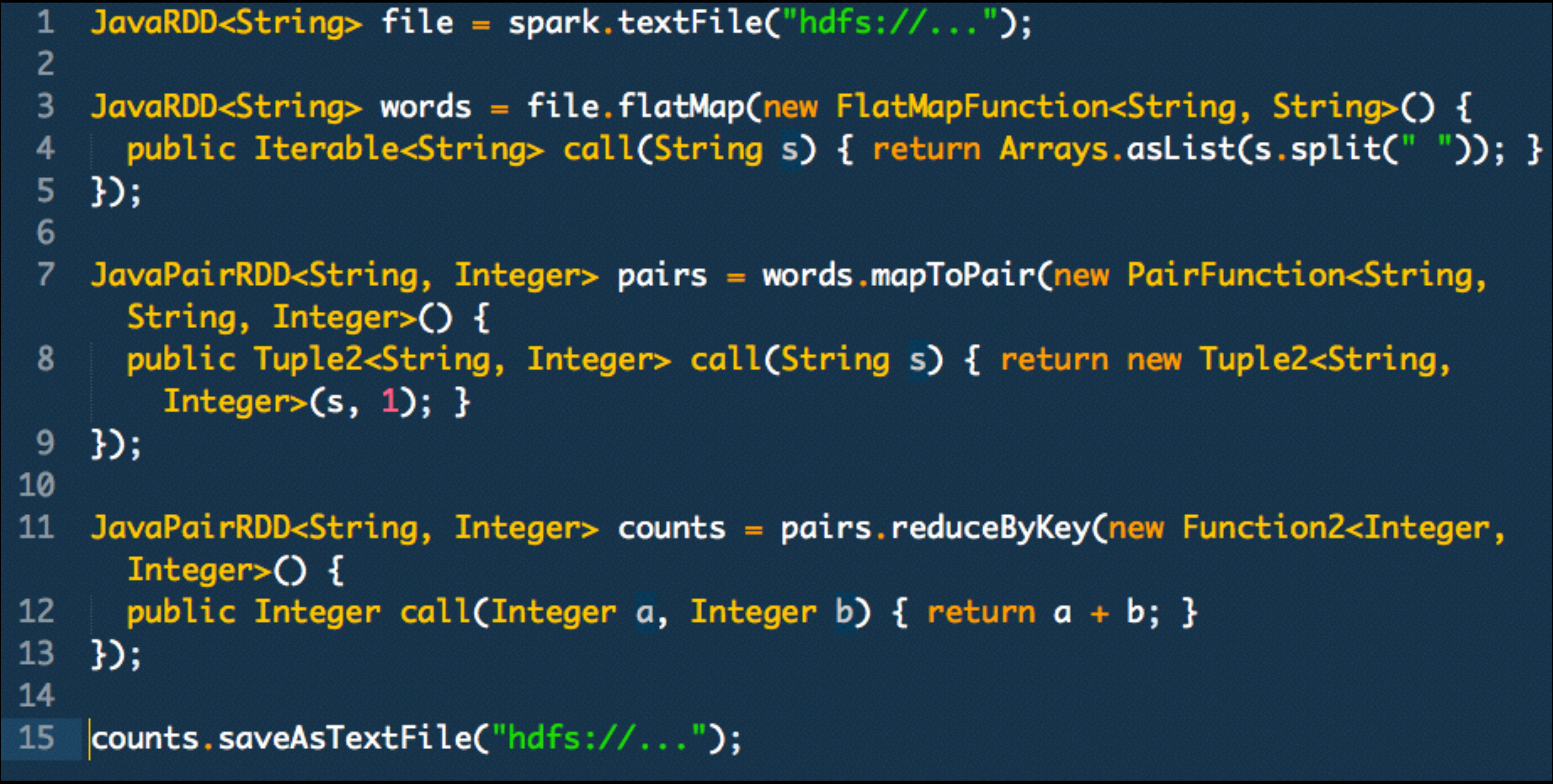
How ~~big is~~ your ~~data~~?
~~fast is~~ ~~computer~~
productive is team

Why Spark?

Hadoop Map/Reduce (Java)

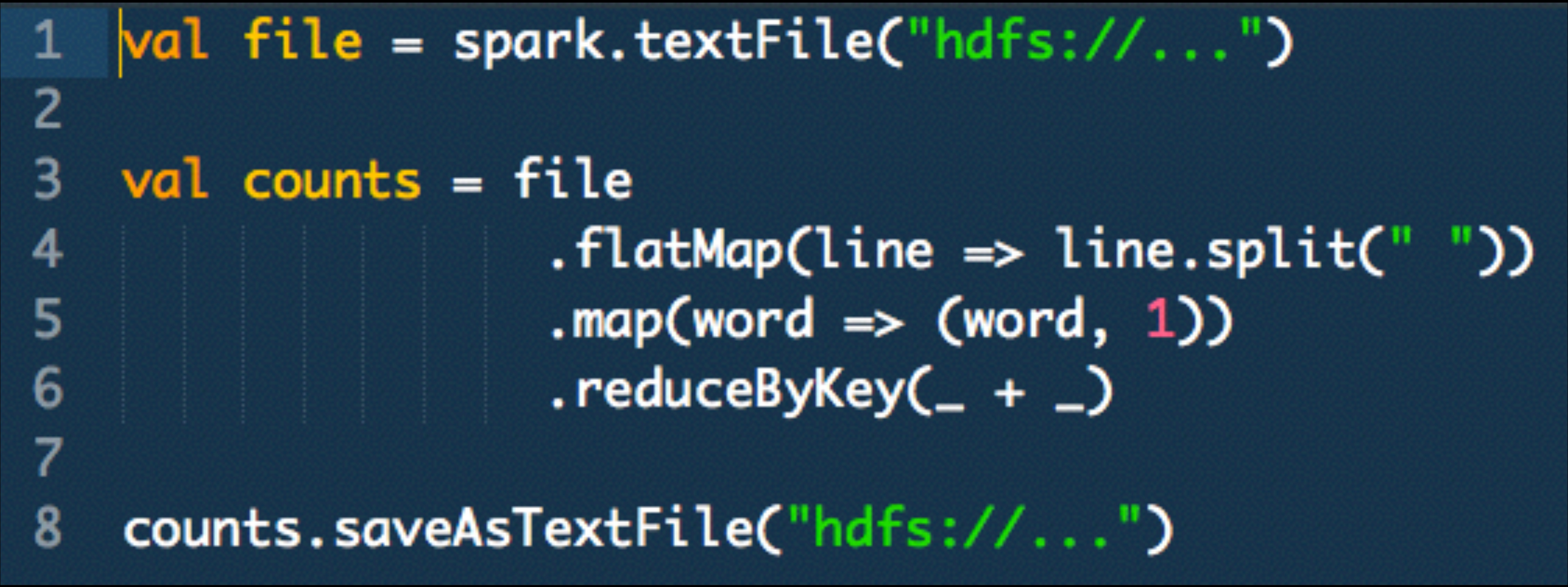


```
public class WordCount {

  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);
  }
}
```



```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.mapToPair(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://...");
```

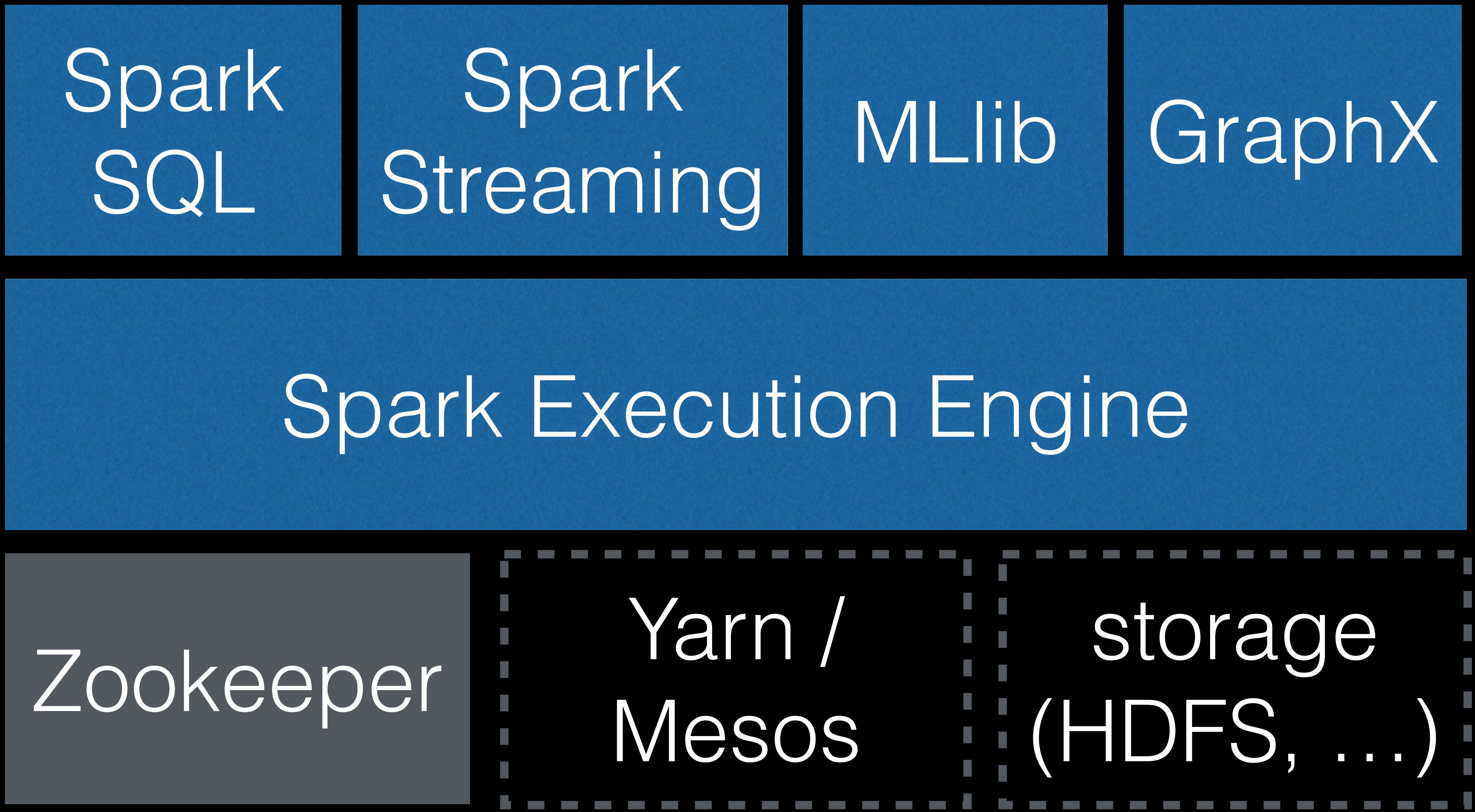
Spark (Java)



```scala
val file = spark.textFile("hdfs://...")

val counts = file
              .flatMap(line => line.split(" "))
              .map(word => (word, 1))
              .reduceByKey(_ + _)

counts.saveAsTextFile("hdfs://...")
```

Spark (Scala)

Spark Architecture



Spark SQL

Spark Streaming

MLlib

GraphX

Spark Execution Engine

Zookeeper

Yarn / Mesos

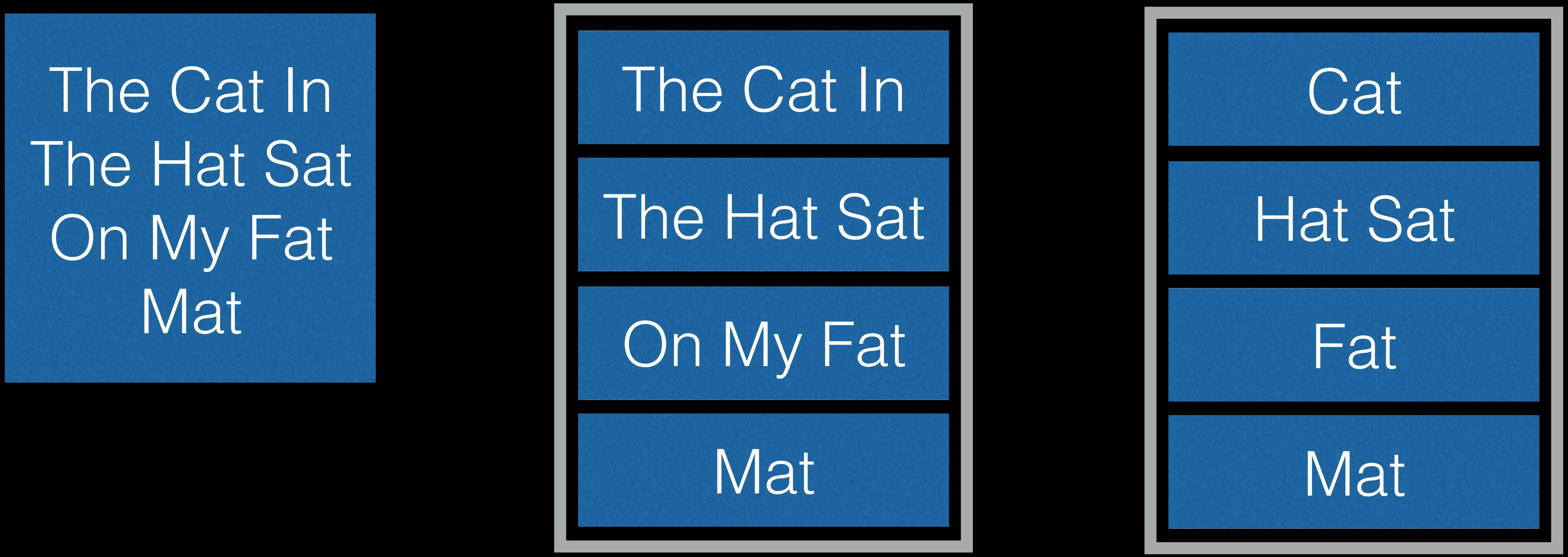
storage (HDFS, …)

Resilient Distributed Dataset

```
words =
"…".split(" ")
```

```
rdd1 = sc.
parallelize(words)
```

```
rdd2 = rdd1.filter(
_.contains("at"))
```



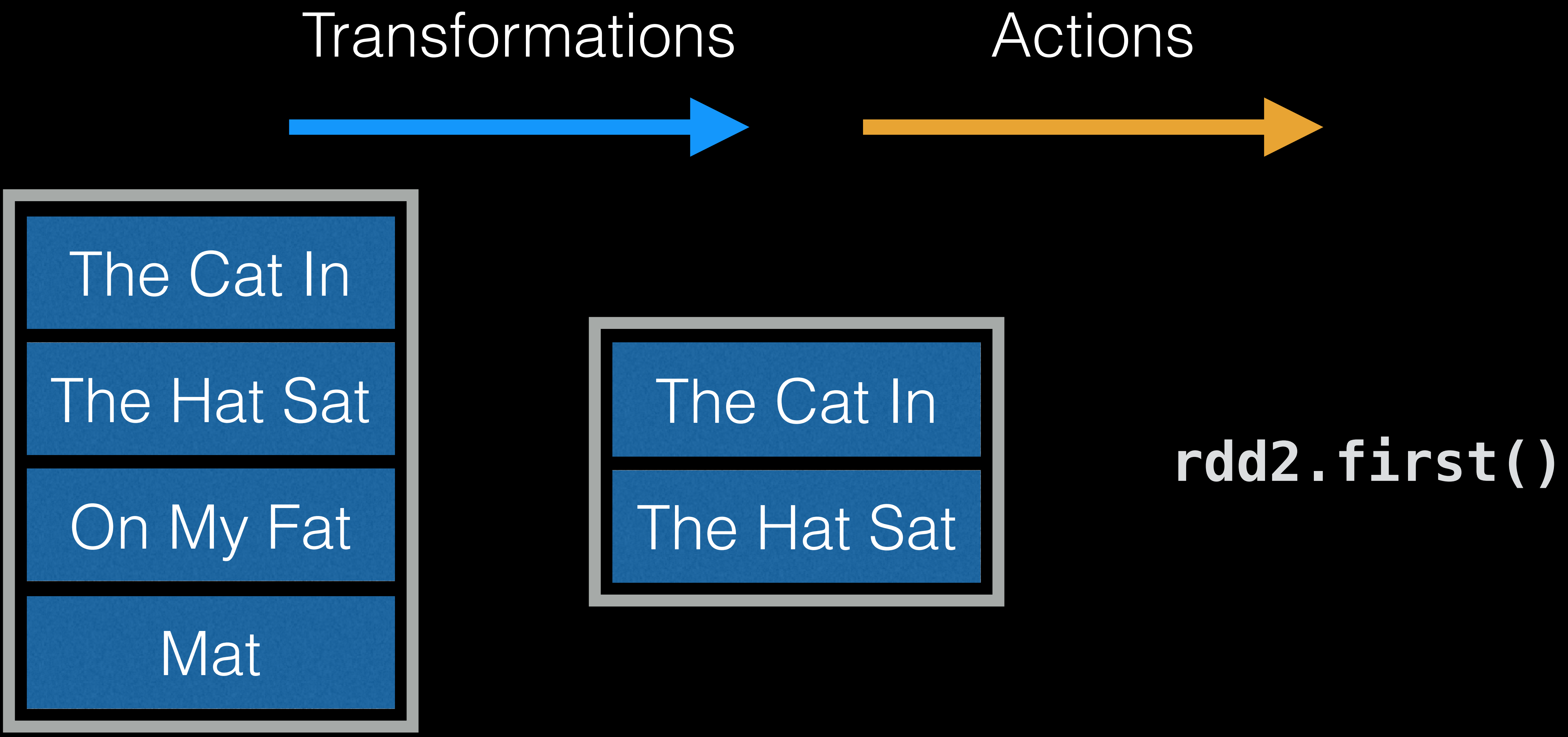
The Cat In The Hat Sat On My Fat Mat

The Cat In
The Hat Sat
On My Fat
Mat

Cat
Hat Sat
Fat
Mat

`Seq[String]` `RDD[String]` `RDD[String]`

RDD Operations



Transformations

Actions

The Cat In

The Hat Sat

On My Fat

Mat

The Cat In

The Hat Sat

```
rdd2.first()
```

Transformations

map

filter

flatMap

mapPartitions

mapPartitionsWithIndex

sample

union

intersection

distinct

groupByKey

aggregateByKey

sortByKey

join

cogroup

cartesian

Actions

reduce

collect

count

first

take

takeSample

takeOrdered

saveAsTextFile

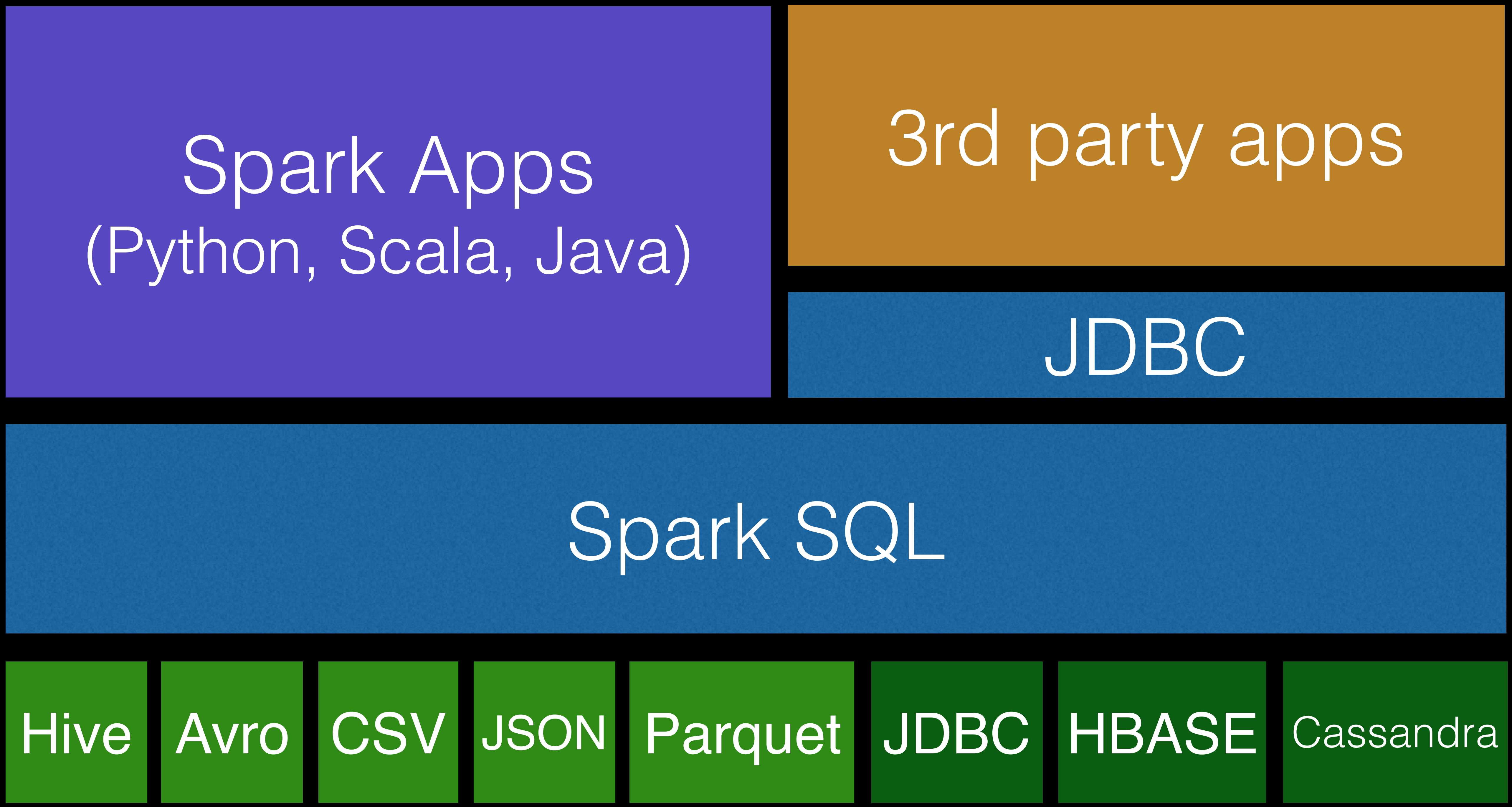
saveAsSequenceFile

saveAsObjectFile

countByKey

foreach

Spark SQL



Spark Apps
(Python, Scala, Java)

3rd party apps

JDBC

Spark SQL

Hive

Avro

CSV

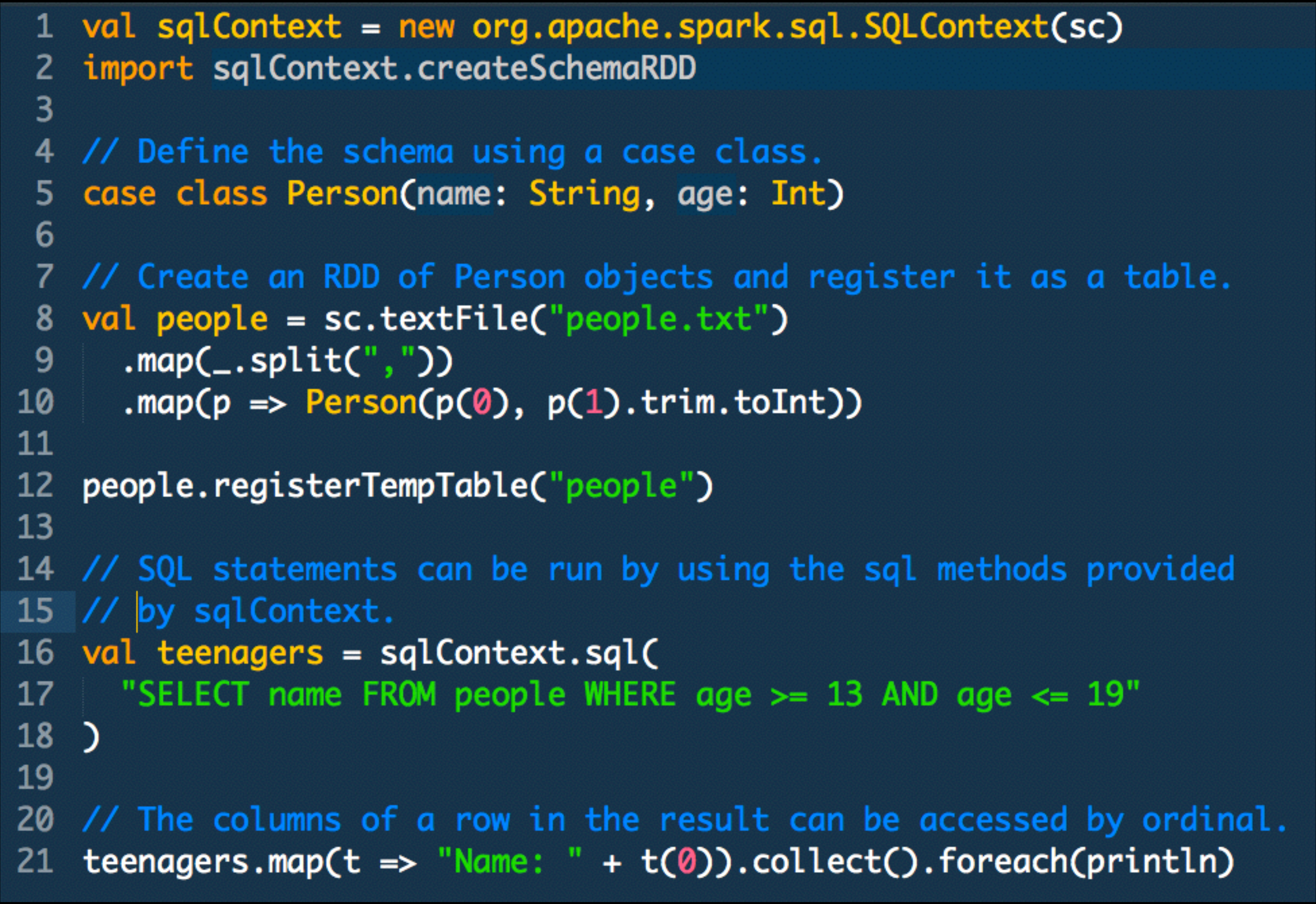
JSON

Parquet

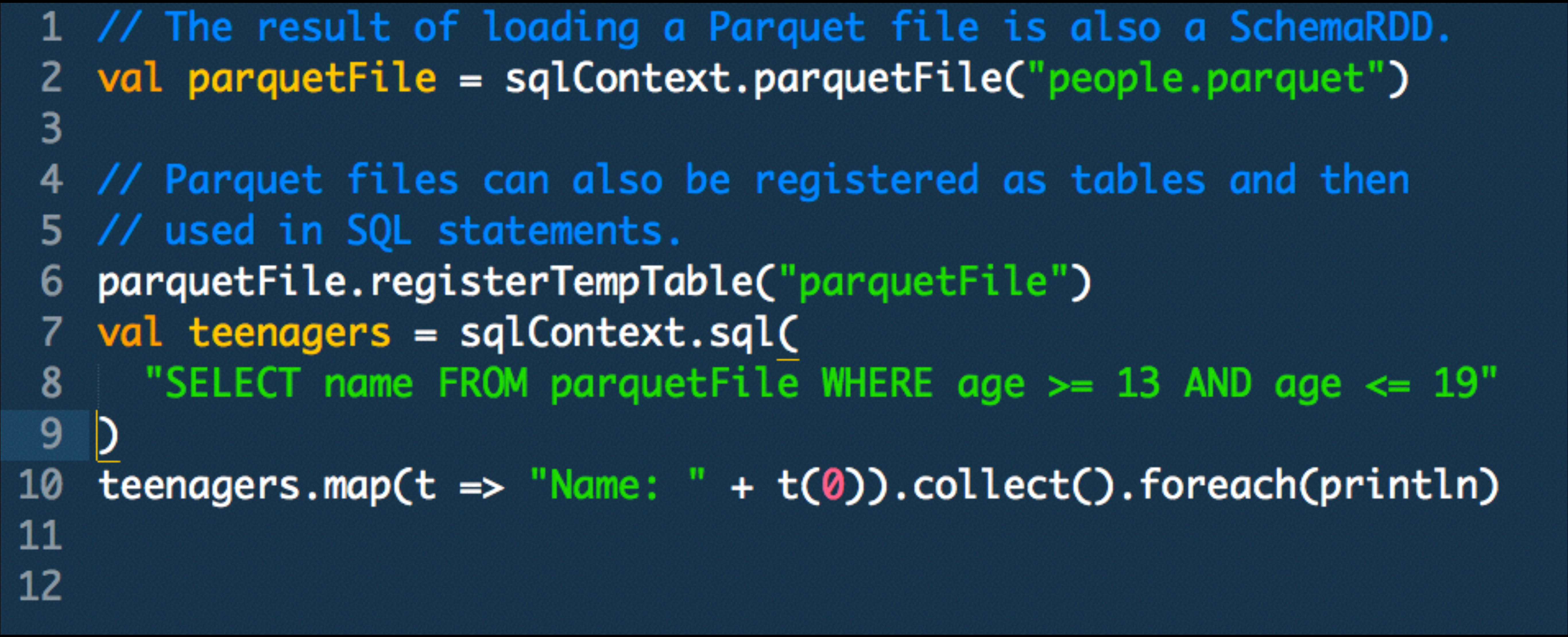
JDBC

HBASE

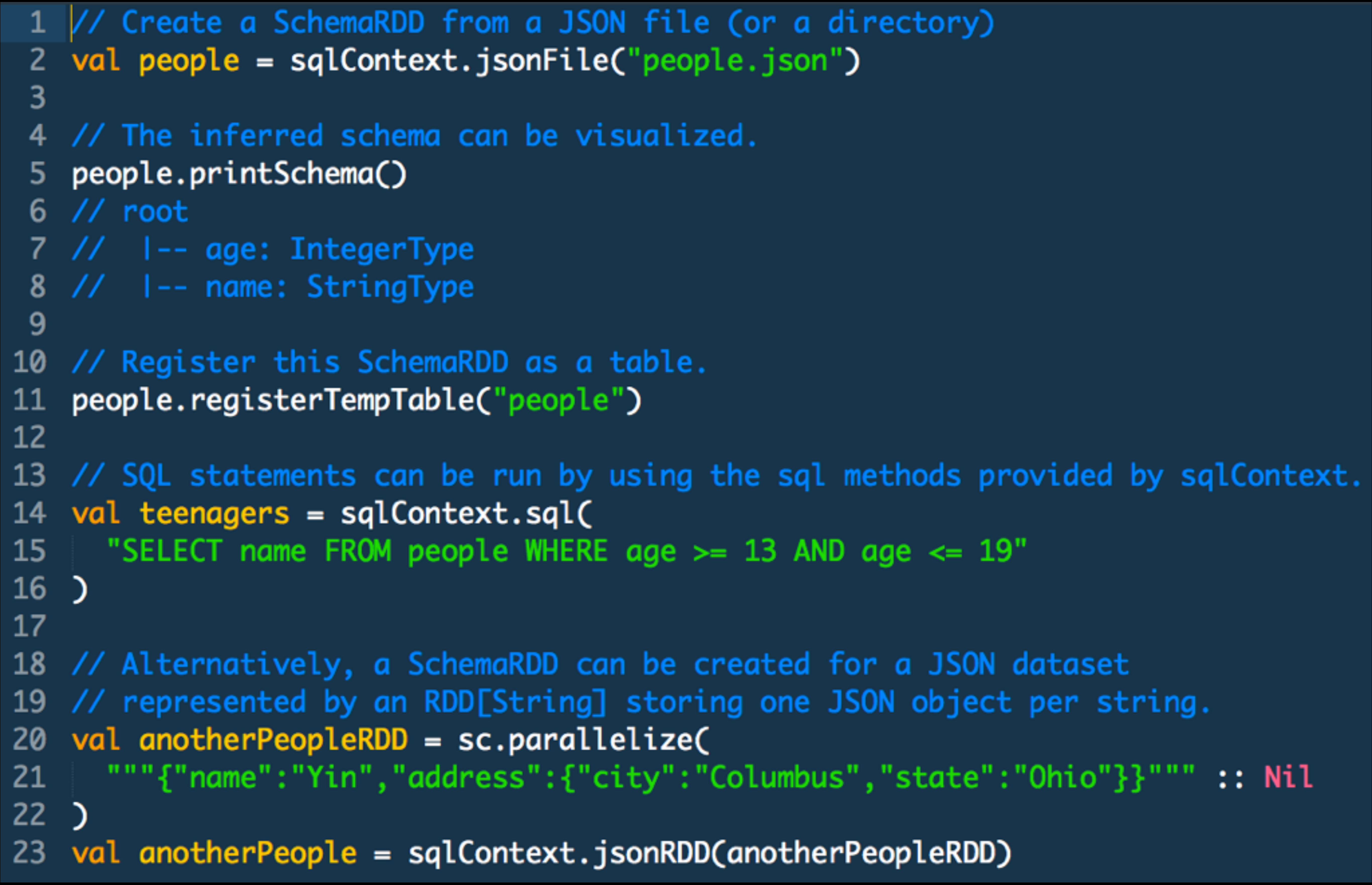
Cassandra



```scala
val sqlContext = new org.apache.spark.sql.SQLContext(sc)
import sqlContext.createSchemaRDD

// Define the schema using a case class.
case class Person(name: String, age: Int)

// Create an RDD of Person objects and register it as a table.
val people = sc.textFile("people.txt")
  .map(_.split(","))
  .map(p => Person(p(0), p(1).trim.toInt))

people.registerTempTable("people")

// SQL statements can be run by using the sql methods provided
// by sqlContext.
val teenagers = sqlContext.sql(
  "SELECT name FROM people WHERE age >= 13 AND age <= 19"
)

// The columns of a row in the result can be accessed by ordinal.
teenagers.map(t => "Name: " + t(0)).collect().foreach(println)
```

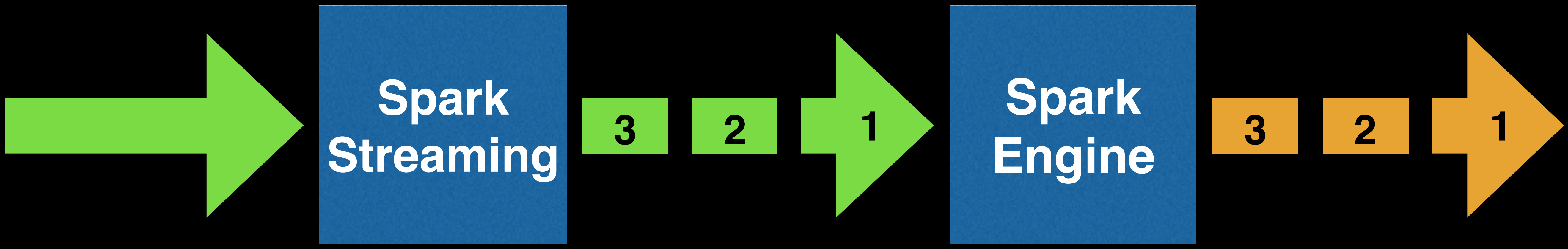


```scala
// The result of loading a Parquet file is also a SchemaRDD.
val parquetFile = sqlContext.parquetFile("people.parquet")

// Parquet files can also be registered as tables and then
// used in SQL statements.
parquetFile.registerTempTable("parquetFile")
val teenagers = sqlContext.sql(
  "SELECT name FROM parquetFile WHERE age >= 13 AND age <= 19"
)
teenagers.map(t => "Name: " + t(0)).collect().foreach(println)
```



```scala
// Create a SchemaRDD from a JSON file (or a directory)
val people = sqlContext.jsonFile("people.json")

// The inferred schema can be visualized.
people.printSchema()
// root
//  |-- age: IntegerType
//  |-- name: StringType

// Register this SchemaRDD as a table.
people.registerTempTable("people")

// SQL statements can be run by using the sql methods provided by sqlContext.
val teenagers = sqlContext.sql(
  "SELECT name FROM people WHERE age >= 13 AND age <= 19"
)

// Alternatively, a SchemaRDD can be created for a JSON dataset
// represented by an RDD[String] storing one JSON object per string.
val anotherPeopleRDD = sc.parallelize(
  """{"name":"Yin","address":{"city":"Columbus","state":"Ohio"}}""" :: Nil
)
val anotherPeople = sqlContext.jsonRDD(anotherPeopleRDD)
```

Spark Streaming



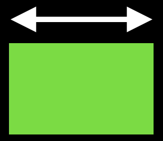
Input data stream

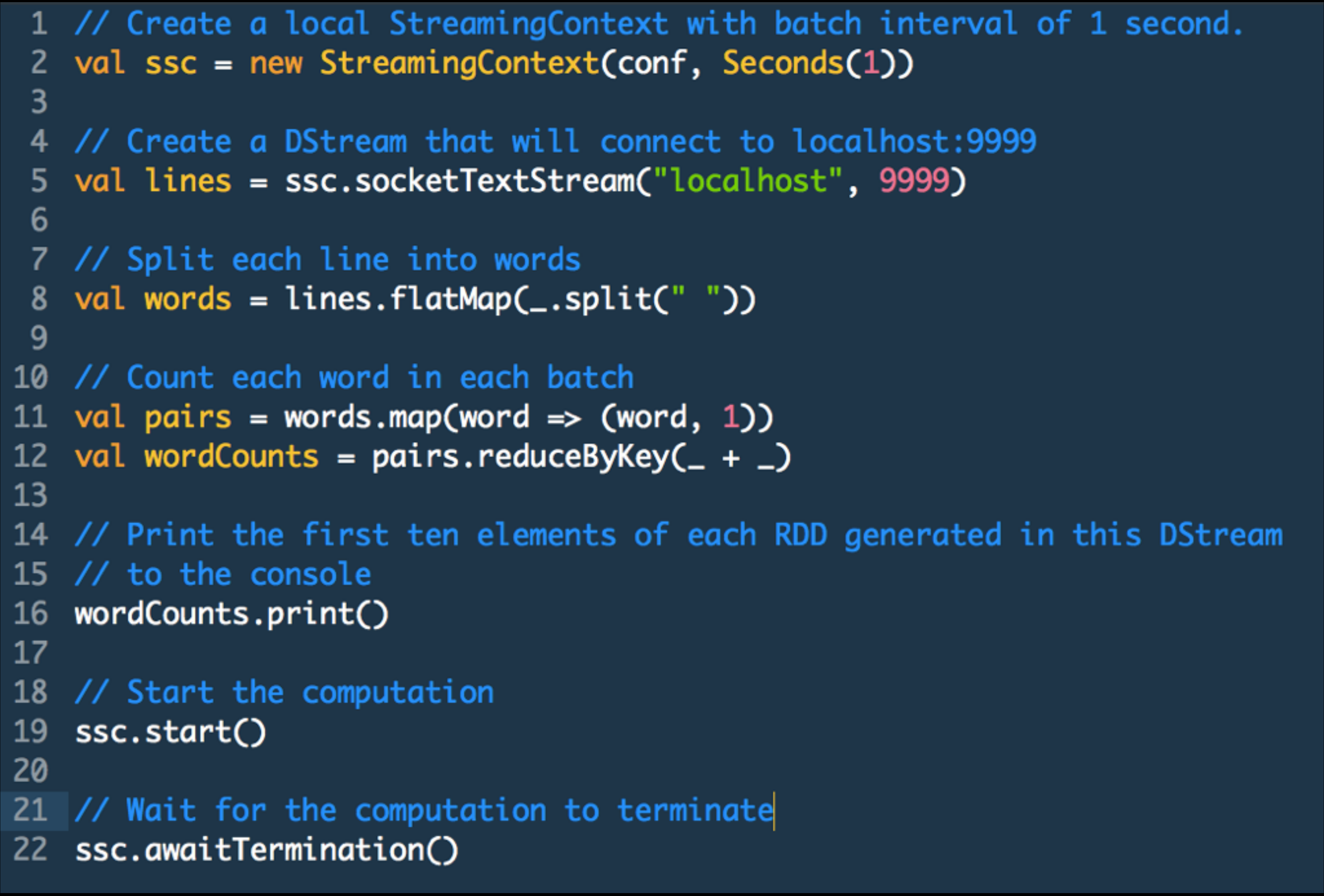
Kafka
Flume
Kinesis
ZeroMQ
MQTT
Twitter
…

Batches of input data

Batches of processed data

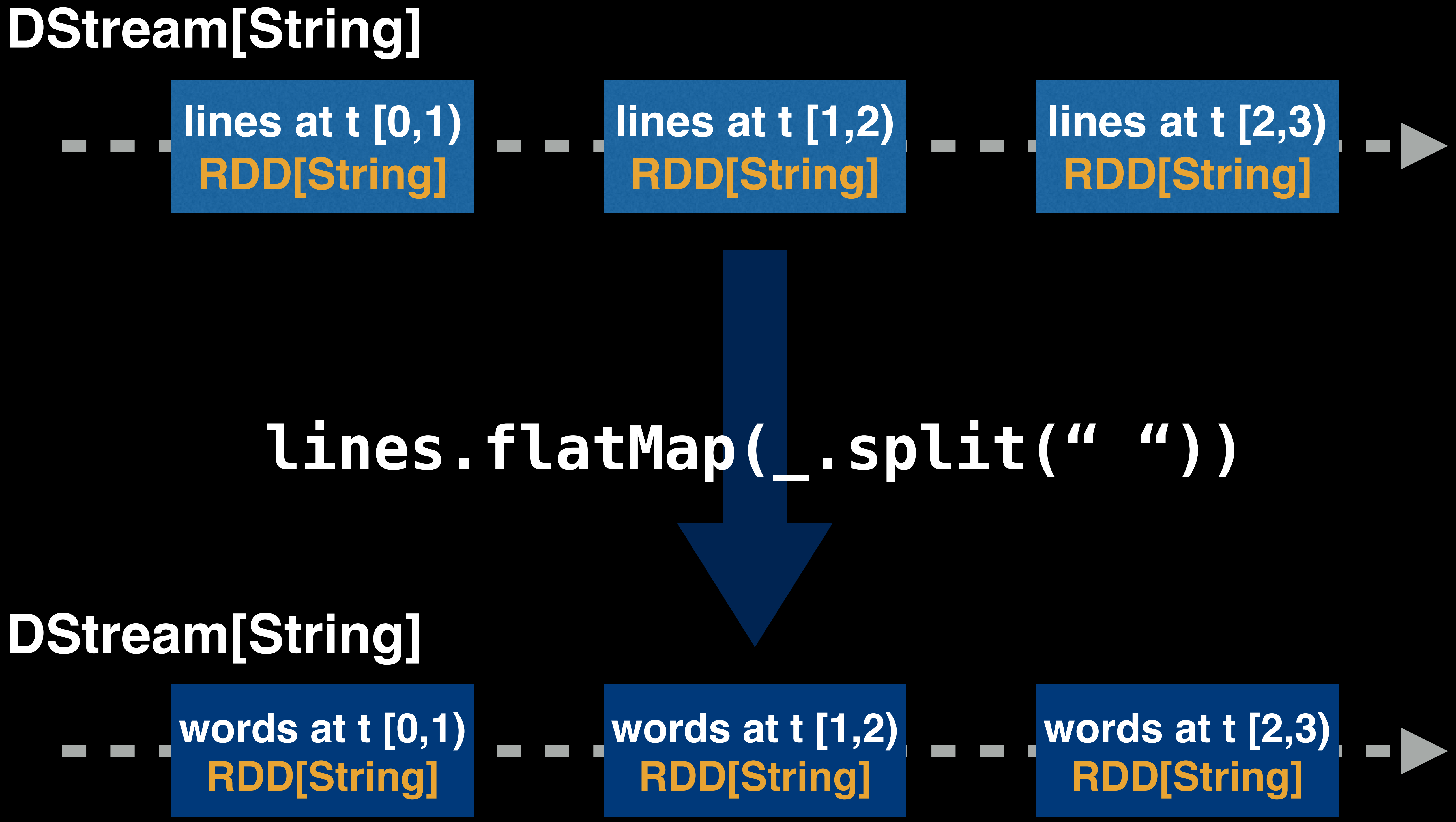
Batch size (s)





```scala
// Create a local StreamingContext with batch interval of 1 second.
val ssc = new StreamingContext(conf, Seconds(1))

// Create a DStream that will connect to localhost:9999
val lines = ssc.socketTextStream("localhost", 9999)

// Split each line into words
val words = lines.flatMap(_.split(" "))

// Count each word in each batch
val pairs = words.map(word => (word, 1))
val wordCounts = pairs.reduceByKey(_ + _)

// Print the first ten elements of each RDD generated in this DStream
// to the console
wordCounts.print()

// Start the computation
ssc.start()

// Wait for the computation to terminate
ssc.awaitTermination()
```

Stateless ops



DStream[String]

lines at t [0,1)
RDD[String]

lines at t [1,2)
RDD[String]

lines at t [2,3)
RDD[String]

```
lines.flatMap(_.split(“ “))
```

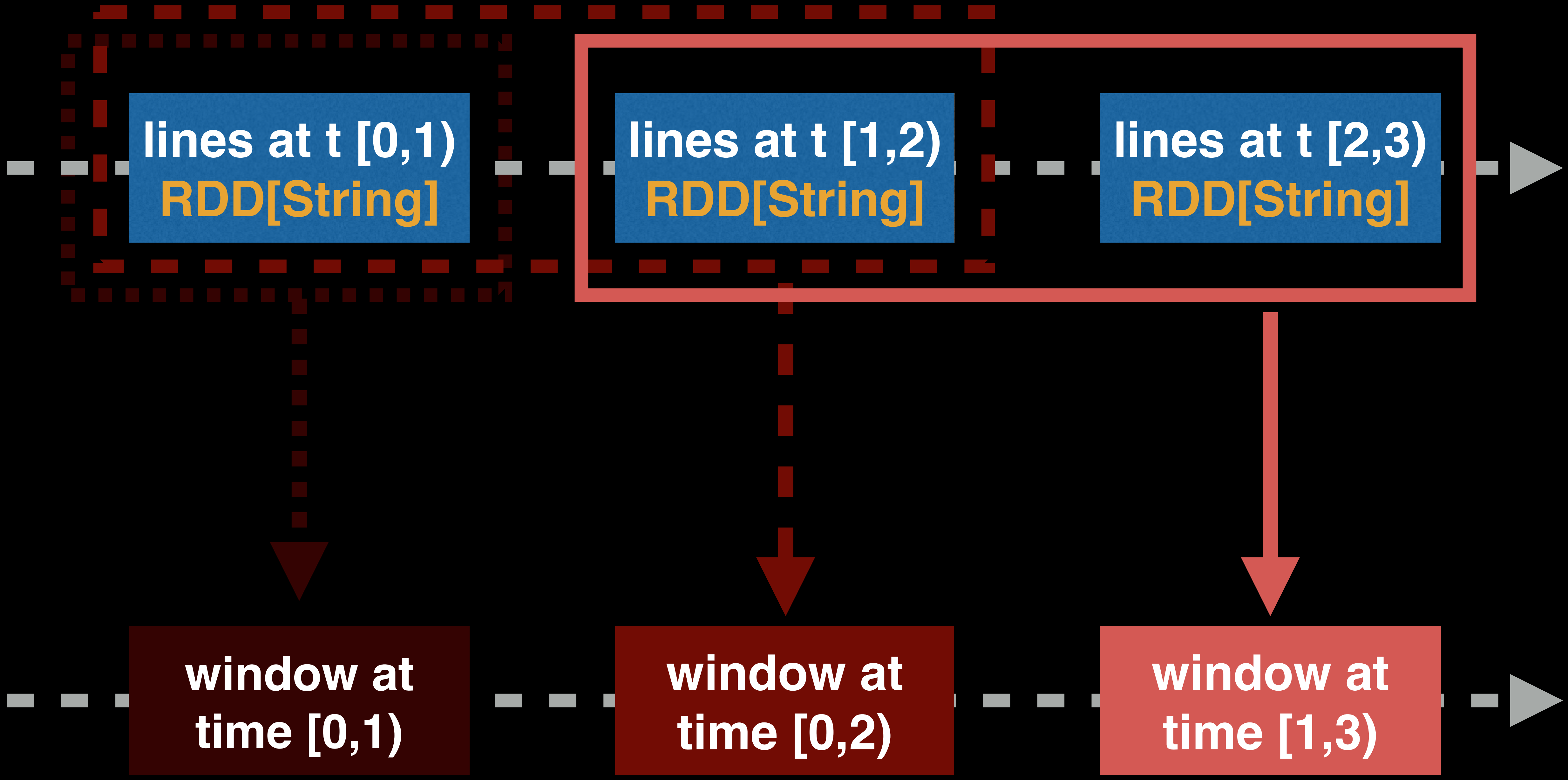
DStream[String]

words at t [0,1)
RDD[String]

words at t [1,2)
RDD[String]

words at t [2,3)
RDD[String]

Windowed ops (stateful)



lines at t [0,1)
RDD[String]

lines at t [1,2)
RDD[String]

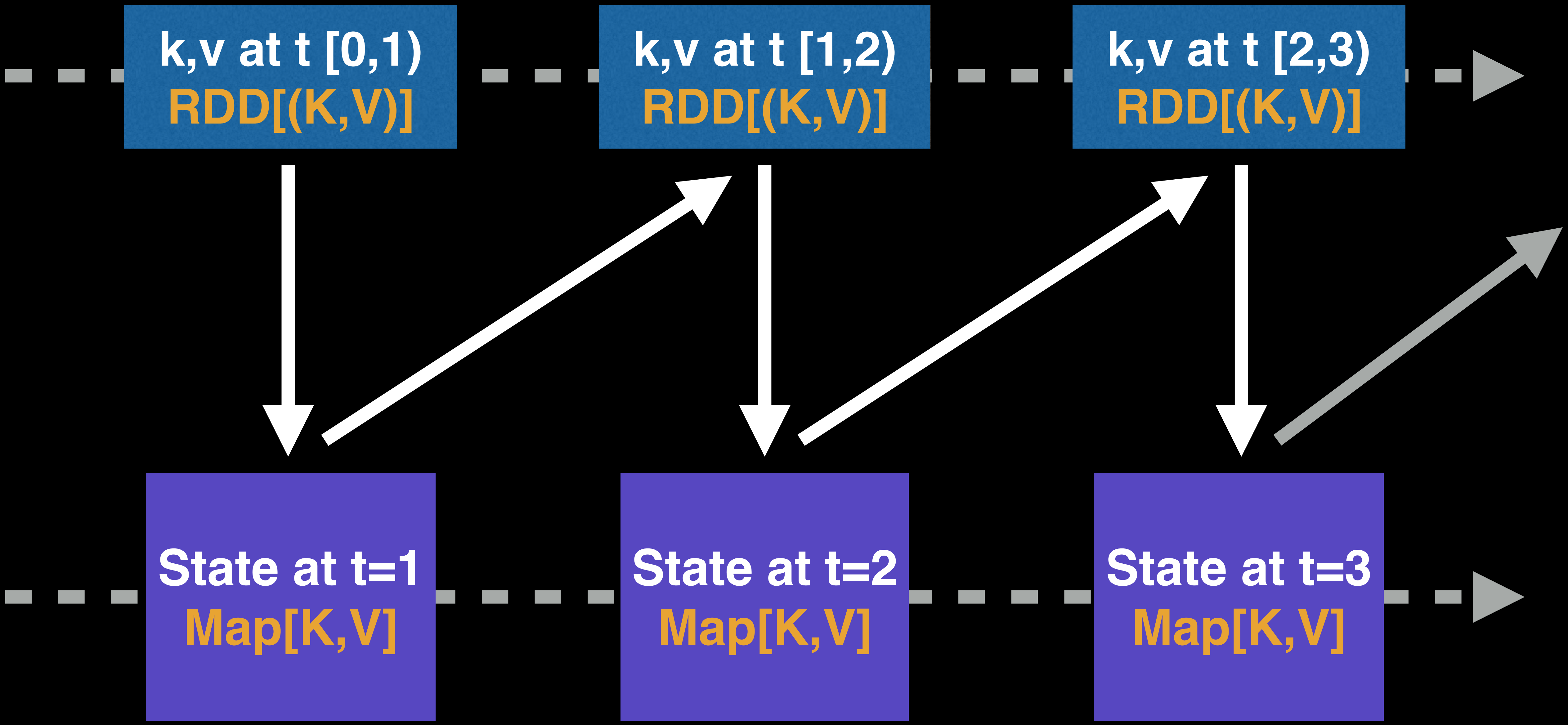
lines at t [2,3)
RDD[String]

window at time [0,1)

window at time [0,2)

window at time [1,3)

UpdateStateByKey (stateful)



k,v at t [0,1)
RDD[(K,V)]

k,v at t [1,2)
RDD[(K,V)]

k,v at t [2,3)
RDD[(K,V)]

State at t=1
Map[K,V]

State at t=2
Map[K,V]

State at t=3
Map[K,V]

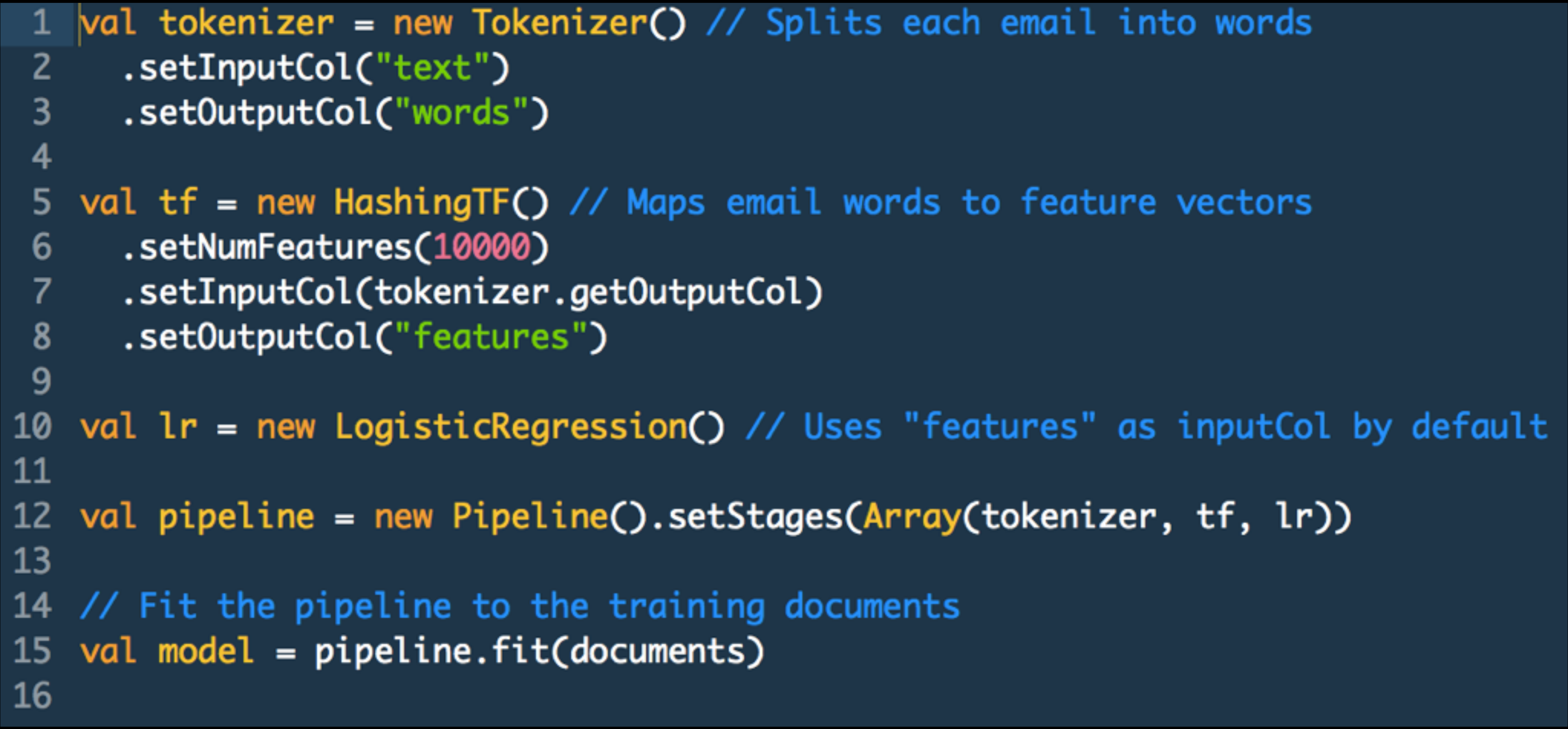
MLlib

RDD based implementation of:
Feature extraction, **Statistics**, **Classification**, **Regression**, **Clustering**, **Collaborative filtering**, **Recommendation**, **Dimensionality reduction**.

Focused on **parallel algorithms that run well on clusters**.

Best suited for **running each algorithm on a large dataset**.

ML Pipeline API



```scala
val tokenizer = new Tokenizer() // Splits each email into words
  .setInputCol("text")
  .setOutputCol("words")

val tf = new HashingTF() // Maps email words to feature vectors
  .setNumFeatures(10000)
  .setInputCol(tokenizer.getOutputCol)
  .setOutputCol("features")

val lr = new LogisticRegression() // Uses "features" as inputCol by default

val pipeline = new Pipeline().setStages(Array(tokenizer, tf, lr))

// Fit the pipeline to the training documents
val model = pipeline.fit(documents)

```

GraphX

RDD based **Vertexes** + **Edges**

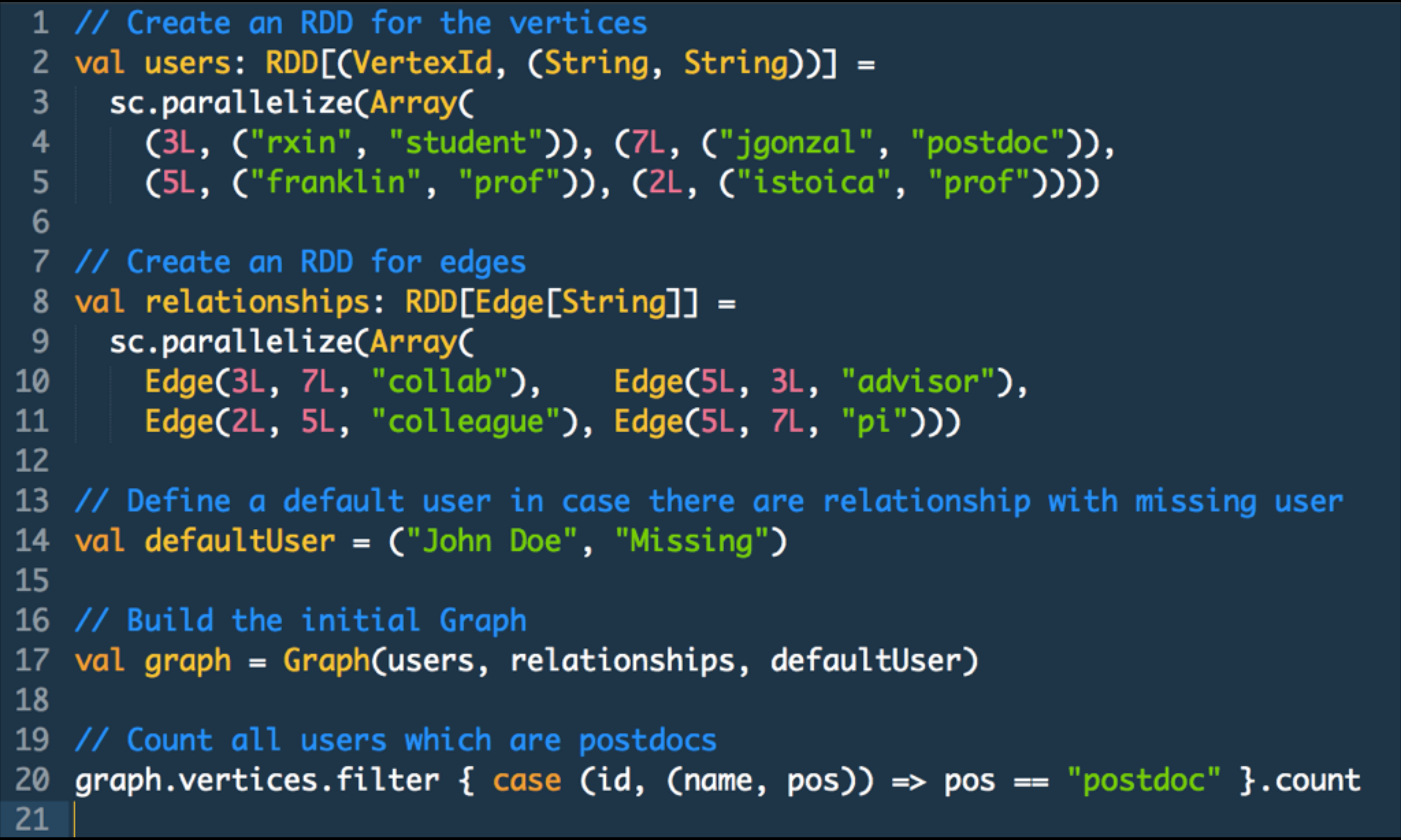
Property Operators (mapVertices, mapEdges, mapTriplets)

Structural Operators (reverse, subgraph, mask, groupEdges)

Join Operators (joinVertices, outerJoinVertices)

Neighborhood Aggregation

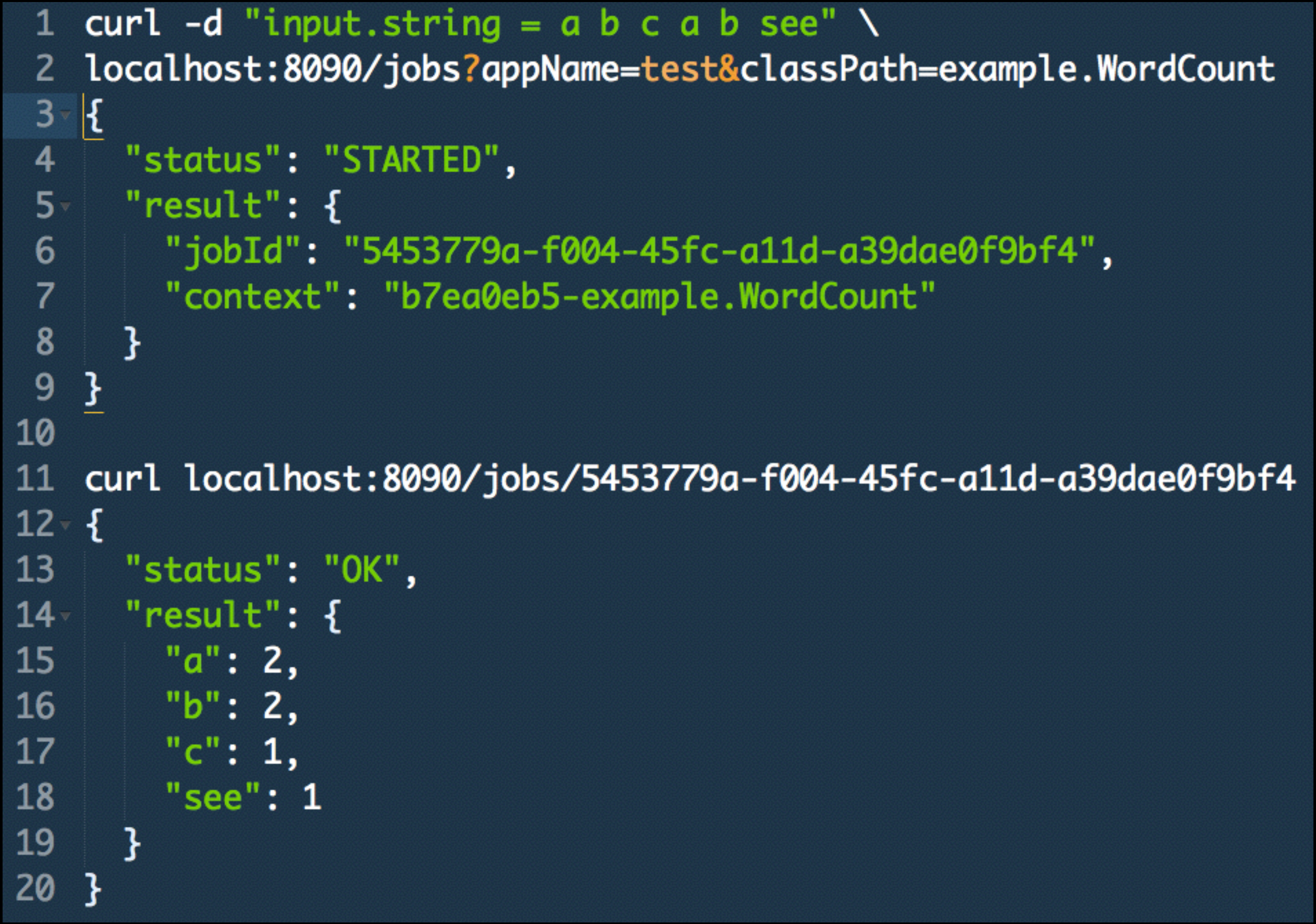
Pregel API



```scala
// Create an RDD for the vertices
val users: RDD[(VertexId, (String, String))] =
  sc.parallelize(Array(
    (3L, ("rxin", "student")), (7L, ("jgonzal", "postdoc")),
    (5L, ("franklin", "prof")), (2L, ("istoica", "prof"))))

// Create an RDD for edges
val relationships: RDD[Edge[String]] =
  sc.parallelize(Array(
    Edge(3L, 7L, "collab"),    Edge(5L, 3L, "advisor"),
    Edge(2L, 5L, "colleague"), Edge(5L, 7L, "pi")))

// Define a default user in case there are relationship with missing user
val defaultUser = ("John Doe", "Missing")

// Build the initial Graph
val graph = Graph(users, relationships, defaultUser)

// Count all users which are postdocs
graph.vertices.filter { case (id, (name, pos)) => pos == "postdoc" }.count
```

Deployment Options

Spark Job Server



```
curl -d "input.string = a b c a b see" \
localhost:8090/jobs?appName=test&classPath=example.WordCount
{
  "status": "STARTED",
  "result": {
    "jobId": "5453779a-f004-45fc-a11d-a39dae0f9bf4",
    "context": "b7ea0eb5-example.WordCount"
  }
}

curl localhost:8090/jobs/5453779a-f004-45fc-a11d-a39dae0f9bf4
{
  "status": "OK",
  "result": {
    "a": 2,
    "b": 2,
    "c": 1,
    "see": 1
  }
}
```

Takeaways

- Spark is powerful yet flexible
- Improves team productivity and (faster results!)
- Easy learning curve (Python, Java and Scala)
- SparkSQL and JDBC API makes integrations with existing BI tools a breeze
- Flexible deployment options (happy sysadmins)
- Try Spark today! Run **spark-shell** on your laptop