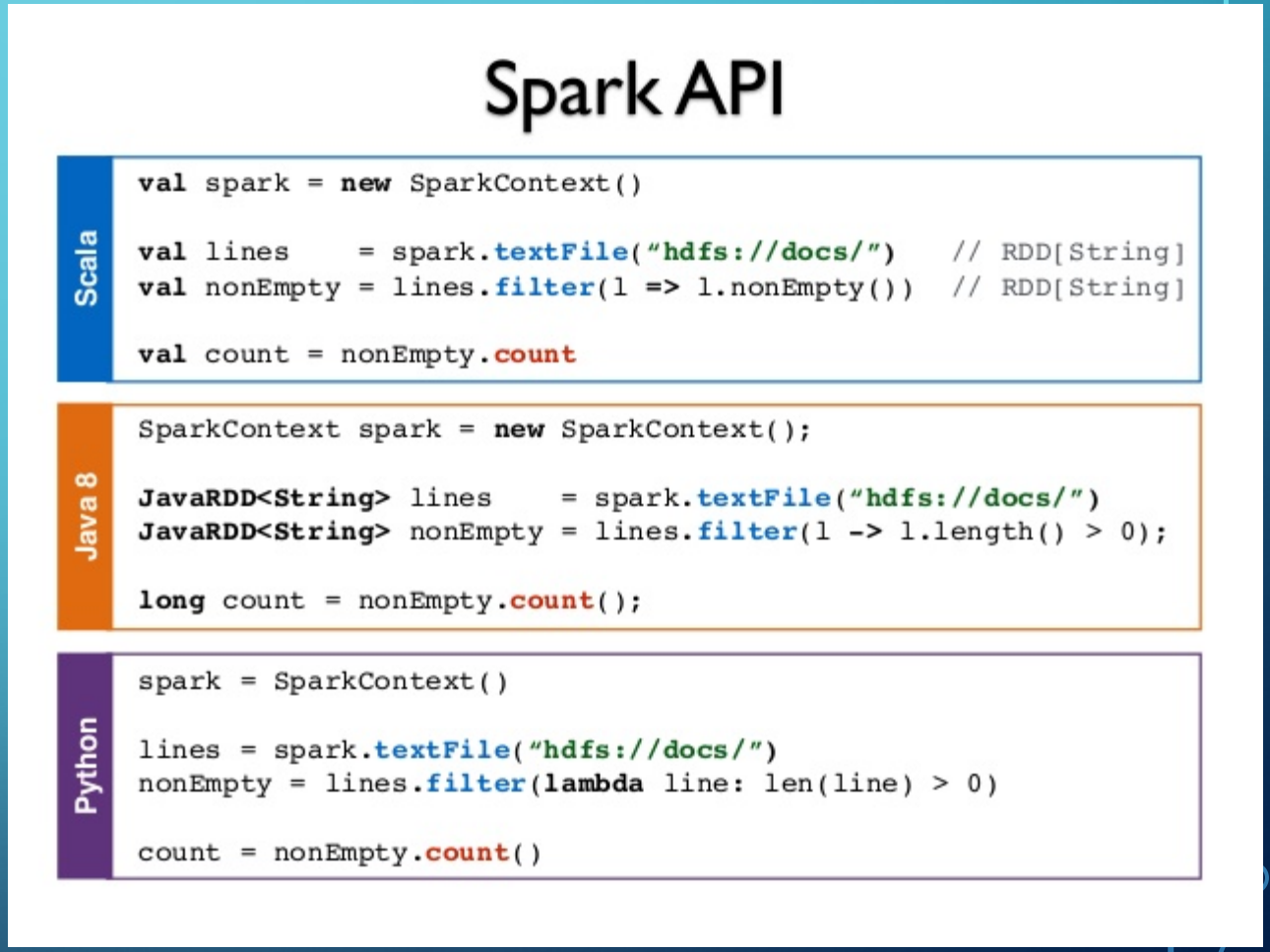
DISTRIBUTED AND CLOUD COMPUTING

LAB10 SPARK APPLICATION PROGRAMMING

SPARK API

- Spark is coded in **Scala**
- Scala is a general-purpose, high-level programming language that combines **object-oriented** and **functional programming** concepts.
- Spark provides API in **Scala, Java and Python**



SPARK API: JAVA V.S. SCALA

- Scala
  - **spark-shell** could do interactive query and testing
  - Scala supports more functional programming syntax, thus can reduce total amount of code
- Java
  - You already know how things work in Java

- But since they are all JVM language (meaning that they all compile to Java .class bytecode), Java and Scala could mix well in your application.

- SparkConf, JavaSparkContext

Java

```java
String appName="Demo";
String master="local[*]"
SparkConf conf = new SparkConf().setAppName(appName).setMaster(master);
JavaSparkContext sc = new JavaSparkContext(conf);
```

Scala

```scala
val appName="Demo"
val master="local[*]"
val conf = new SparkConf().setAppName(appName).setMaster(master)
new SparkContext(conf)
```

- Master could be:
  - Local: `local`, `local[k]`, `local[*]`: use single, k or <num of core> threads
  - Standalone: master url, such as `spark://host:port`
  - Yarn: Just use `"yarn-cluster"`

Transformation

- JavaRDD<class>, parallelize, saveAsTextFile(action)

Java

```java
List<String> dataList=new ArrayList<>(Arrays.asList("1,2,3,4,5", "a,b,c,d,e"));
JavaRDD<String> dataRdd=sc.parallelize(dataList);
dataRdd.saveAsTextFile("output/");
```

Scala

```scala
val dataList = Array("1,2,3,4,5", "a,b,c,d,e")
val dataRdd = sc.parallelize(dataList)
dataRdd.saveAsTextFile("output/")
```

- JavaRDD is the representation of RDD in Java API
- `parallelize` converts the data to RDD
- `saveAsTextFile` saves the RDD's content to a text file, is an action

Transformation

- map

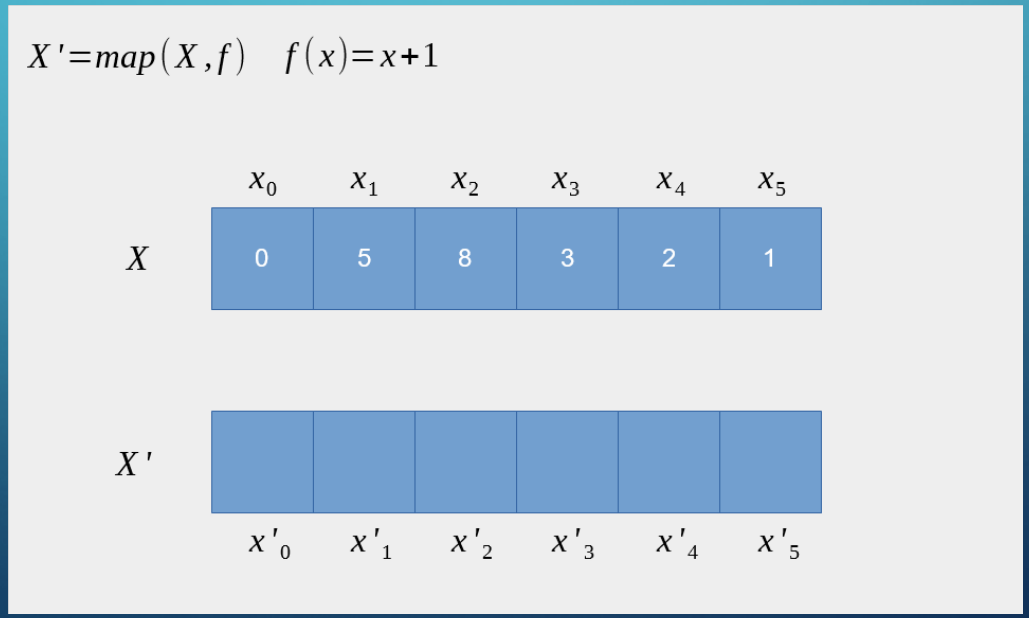
Java

```
// ["1,2,3,4,5", "a,b,c,d,e"]
JavaRDD<String []> mapRdd = dataRdd.map(l -> l.split(","));
// [["1","2","3","4","5"],["a","b","c","d","e"]]
```

Scala

```
val mapRdd = dataRdd.map(s => s.split(“,”))
```

- map applies the operation to each element in the RDD



Transformation

- `flatMap`

Java

```java
JavaRDD<String> flatMapRdd = dataRdd.flatMap(l ->
                        Arrays.asList(l.split(",")).iterator());
// ["1","2","3","4","5","a","b","c","d","e"]
```

Scala

```scala
val flatMapRdd = dataRdd.flatMap(s => s.split(","))
```

- Same as `map` but the result is *flat*

```
// not flat [["1","2","3","4","5"],["a","b","c","d","e"]]
// flat     ["1","2","3","4","5","a","b","c","d","e"]
```

Transformation

- `filter`

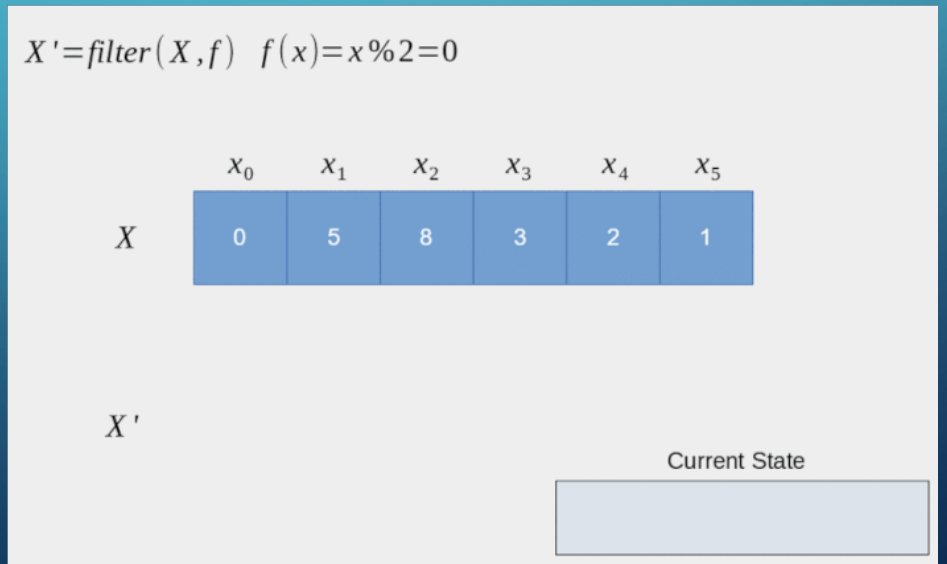
Java

```
// ["1","2","3","4","5","a","b","c","d","e"]
JavaRDD<String> filterRdd = flatMapRdd.filter(l -> Character.isDigit(l.charAt(0)));
// ["1","2","3","4","5"]
```

Scala

```
val filterRdd = flatMapRdd.filter(s => s.charAt(0).isDigit)
```

- `filter` applies the function to each element in the RDD. If result is true, the element is preserved. If false, then element is discarded.



Transformation

- union

Java

```
// dataRdd1: [1,2,3] dataRdd2: [4,5,6]
JavaRDD<Integer> unionRdd = dataRdd1.union(dataRdd2);
// [1,2,3,4,5,6]
```

Scala

```
val unionRdd = dataRdd1.union(dataRdd2)
```

- union just merge two RDDs together, like the union in SQL

Transformation

- groupByKey, JavaPairRDD<class, class>

Java

```java
List<Tuple2<String, Integer>> list = Arrays.asList(
        new Tuple2<>("a", 1),
        new Tuple2<>("b", 2),
        new Tuple2<>("a", 3),
        new Tuple2<>("b", 4)
);
JavaPairRDD<String, Integer> pairRdd = sc.parallelizePairs(list);

// pairRdd: [("a",1),("b",2),("a",3),("b",4)]
JavaPairRdd<String, List<Integer>> groupByKeyRdd = pairRdd.groupByKey();
// [("a",[1,3]),("b",[2,4])]
```

Scala

```scala
val list = List(("a", 1), ("b", 2), ("a", 3), ("b", 4))
val pairRdd = sc.parallelize(list)
val groupByKeyRdd = pairRdd.groupByKey()
```

- JavaPairRDD is the key-value pair RDD
- groupByKey groups a pair by their keys

Transformation

- reduceByKey

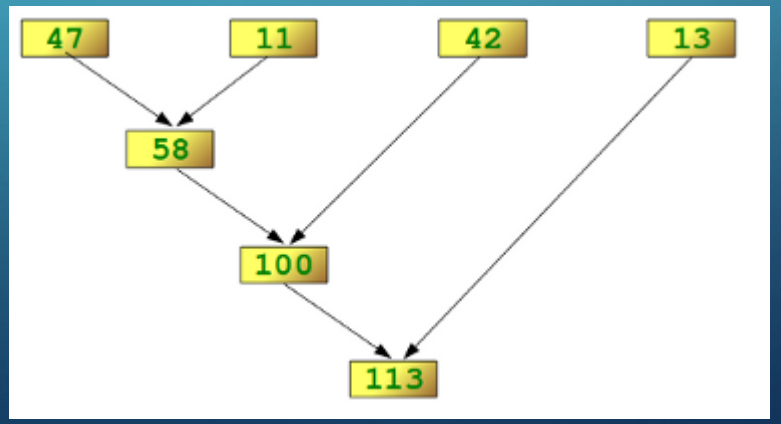
Java

```java
// pairRdd: [("a",1),("b",2),("a",3),("b",4)]
JavaPairRdd<String, Integer> reduceByKeyRdd = pairRdd.reduceByKey(
    (n,m) -> n+m;
);
// [("a",4),("b",6)]
```

Scala

```scala
val reduceByKeyRdd = pairRdd.reduceByKey((n,m) => n+m)
```

- reduceByKey performs groupBy first, then perform reduce



Transformation

- mapValues

Java

```java
// pairRdd: [("a",1),("b",2),("a",3),("b",4)]
JavaPairRdd<String, Integer> mapValuesRdd = pairRdd.mapValues(
    e -> e+1;
);
// [("a",2),("b",3),("a",4),("b",5)]
```

Scala

```scala
val mapValuesRdd = pairRdd.mapValues(e => e+1)
```

- mapValues performs map to value only, while map applies to the element (k-v pair)

Action

- count

Java

```
// filterRdd: ["1","2","3","4","5"]
int count = filterRdd.count();
// 5
```

Scala

```
val count = filterRdd.count()
```

- count counts the number of elements in an RDD (number of lines)

Action

- collect

Java

```
// filterRdd: ["1","2","3","4","5"]
List<String> list = filterRdd.collect();
// ["1","2","3","4","5"] (Java List)
```

Scala

```
val list = filterRdd.collect()
```

- collect converts the RDD back to Java's list (Since RDD is distributed)
- This action should be avoided for large dataset, since it will send all data to the local node

SPARK WORDCOUNT

Java

```java
JavaRDD<String> textFile = sc.textFile("hdfs://...");
JavaPairRDD<String, Integer> counts = textFile
    .flatMap(s -> Arrays.asList(s.split(" ")).iterator())
    .mapToPair(word -> new Tuple2<>(word, 1))
    .reduceByKey((a, b) -> a + b);
counts.saveAsTextFile("hdfs://...");
```

Scala

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

https://spark.apache.org/examples.html

PRACTICE

- A dataset of parking lots in Shenzhen is provided
- You need to do some analytics using this dataset
  - Output the total amount of parking lots and all unique ids, associated with their section
  - Output the average parking time for each section
  - Output the average parking time for each parking lot, sort from highest to lowest
  - Output the total number of parking lots in use and the percentage, for each section, in a 30-minute interval

DATASET EXAMPLE

- Dataset is a csv (comma separated value) file, with the following headers:
  - section: section where the parking lot is in
  - berthage: id of the parking lot
  - admin_region: district
  - out_time: time when the car goes out of the parking lot
  - in_time: time when the car goes into the parking lot

```
out_time,admin_region,in_time,berthage,section
"2018-09-01 12:00:00","南山区","2018-09-01 10:10:00","201091","荔园路(蛇口西段)"
"2018-09-01 14:29:35","南山区","2018-09-01 13:43:35","201091","荔园路(蛇口西段)"
"2018-09-01 16:08:54","南山区","2018-09-01 15:10:54","201091","荔园路(蛇口西段)"
"2018-09-01 17:56:03","南山区","2018-09-01 16:34:03","201091","荔园路(蛇口西段)"
"2018-09-01 20:00:20","南山区","2018-09-01 18:40:20","201091","荔园路(蛇口西段)"
"2018-09-02 12:51:00","南山区","2018-09-02 10:10:00","201091","荔园路(蛇口西段)"
"2018-09-02 17:31:15","南山区","2018-09-02 16:27:15","201091","荔园路(蛇口西段)"
"2018-09-02 20:00:58","南山区","2018-09-02 19:04:58","201091","荔园路(蛇口西段)"
"2018-09-03 09:00:00","南山区","2018-09-03 07:40:00","201091","荔园路(蛇口西段)"
"2018-09-03 12:05:47","南山区","2018-09-03 11:09:47","201091","荔园路(蛇口西段)"
"2018-09-03 15:10:40","南山区","2018-09-03 13:44:40","201091","荔园路(蛇口西段)"
"2018-09-03 16:02:30","南山区","2018-09-03 15:20:30","201091","荔园路(蛇口西段)"
"2018-09-04 11:34:48","南山区","2018-09-04 09:49:48","201091","荔园路(蛇口西段)"
"2018-09-04 14:25:16","南山区","2018-09-04 13:41:16","201091","荔园路(蛇口西段)"
"2018-09-04 16:53:17","南山区","2018-09-04 15:42:17","201091","荔园路(蛇口西段)"
```

TIPS FOR PRACTICE

- Tips:
  - Filter out invalid data first, e.g. out_time earlier than or equal to in_time
  - Some optimization could be done, like converting the data (in_time, out_time) to (in_time, parking_time_length) before doing further computations
  - https://spark.apache.org/docs/latest/rdd-programming-guide.html