

Word Count Example in Spark

The following slides show how to implement the MapReduce Word count algorithm, both in **Java** and **Python**

To understand the code, refer to

- The standard Java and Python APIs and reference manuals.
- Introduction to Programming in Spark (*for this course*)
- Spark RDD Programming guide.
- Spark Java API.
- Spark Python API.

All links and the complete code (**WordCountExample.Java** and **WordCountExample.py**) can be found in the course Moodle.

OUTLINE

- ① Functional programming
- ② 1-Round Word Count in Spark
- ③ 2-Round Word Count in Spark

Functional programming

Functional programming

One of the core ideas of functional programming is that **functions can be arguments to other functions**.

- In a MapReduce algorithm, the map and reduce functions used in each round can be regarded as "arguments" for the round.
- Spark's API relies heavily on passing functions. For example, many methods (e.g., RDD operations) require functions as arguments.

In Spark there are essentially **2 ways to pass functions as arguments** to a method.

- as **anonymous functions** or **lambdas** which are defined inline without a name, where the actual argument is expected.
- as **named functions** defined outside the method.

Functional programming: Java example

Let `values` be an `RDD of doubles` (i.e., of type `JavaRDD<Double>`)

USE of ANONYMOUS FUNCTIONS

- **Single statement:**

```
JavaRDD<Double> squaredValues = values.map((x) -> x*x);
```

- **Multiple statements:**

```
double fixed = 1.5;
JavaRDD<Double> normValues = values.map((x) -> {
    double diff = fixed - x;
    return diff;
});
```

Terminology: variable `fixed` is "captured" by the anonymous function

Functional programming: Java example

USE of NAMED FUNCTIONS

Create a separate class such as:

```
class myMethods {  
    public static double mySquare(double x) {  
        return x * x;  
    }  
}
```

Then, in the class containing the main program write:

```
JavaRDD<Double> squaredValues = values.map(myMethods::mySquare);
```

Functional programming: Python example

Let `values` be an RDD of double

USE of ANONYMOUS FUNCTIONS

```
squaredValues = values.map(lambda x: x * x)
```

Obs: Python does not allow multiple statements in anonymous functions.

USE of NAMED FUNCTIONS

```
def mySquare(x):  
    return x * x
```

```
squaredValues = values.map(mySquare)
```


1-Round Word Count in Spark

JAVA

Java: initialization

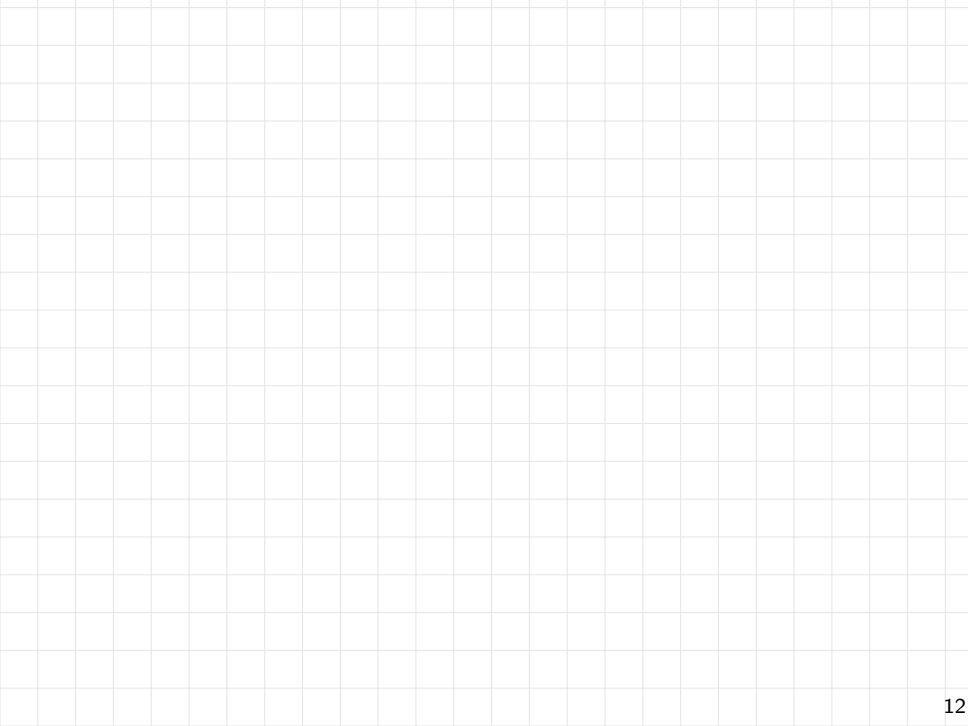
```
import ....

public class WordCountExample{

    public static void main(String[] args) throws IOException {

        // SPARK SETUP
        SparkConf conf = new SparkConf(true).setAppName("WordCount");
        JavaSparkContext sc = new JavaSparkContext(conf);

        // INPUT READING
        int K = Integer.parseInt(args[0]);
        JavaRDD<String> docs = sc.textFile(args[1]).repartition(K).cache();
```



Java: implementation of the round

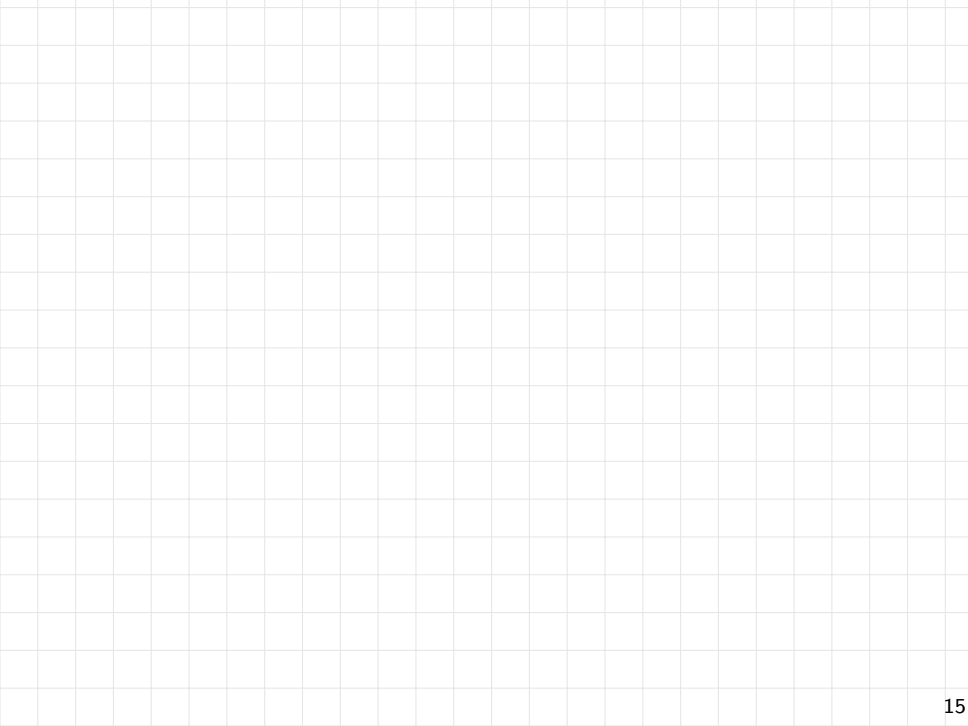
The full code implementing the 1-round algorithm is:

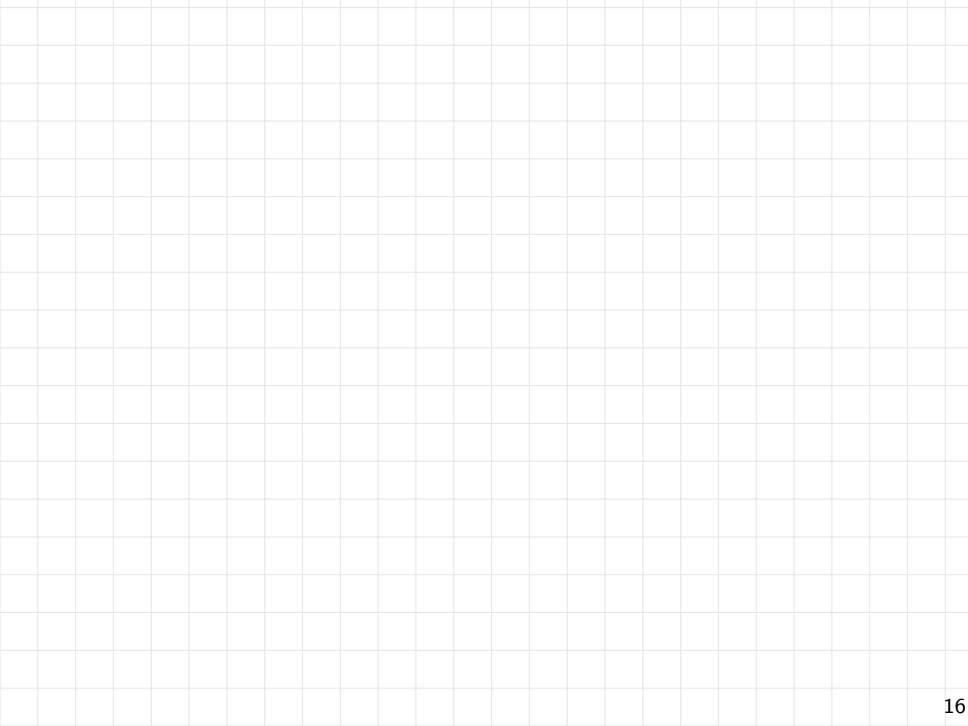
```
JavaPairRDD<String, Long> wordCounts = docs
  .flatMapToPair(myMethods::wordCountPerDoc) // <-- MAP PHASE
  .groupByKey() // SHUFFLING+GROUPING
  .mapValues((it) -> { // <-- REDUCE PHASE
    long sum = 0;
    for (long c : it) sum += c;
    return sum;
  })
```

Observation: to illustrate the various possibilities to pass functions, we made `flatMapToPair` receive in input a named function, and made `mapValues` receive in input an anonymous function

Java: implementation of the round

```
class myMethods {  
    public static Iterator<Tuple2<String,Long>>  
        wordCountPerDoc(String document) {  
        String[] tokens = document.split(" ");  
        HashMap<String, Long> counts = new HashMap<>();  
        ArrayList<Tuple2<String, Long>> pairs = new ArrayList<>();  
        for (String token : tokens) {  
            counts.put(token, 1L + counts.getDefault(token, 0L));  
        }  
        for (Map.Entry<String, Long> e : counts.entrySet()) {  
            pairs.add(new Tuple2<>(e.getKey(), e.getValue()));  
        }  
        return pairs.iterator();  
    }  
    .....  
}
```

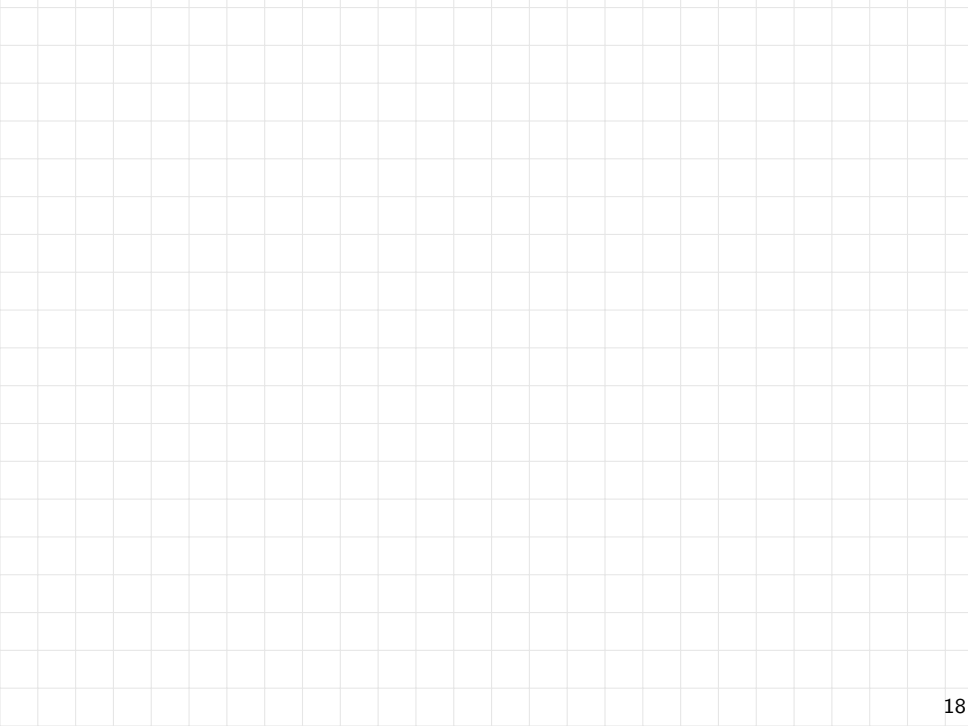


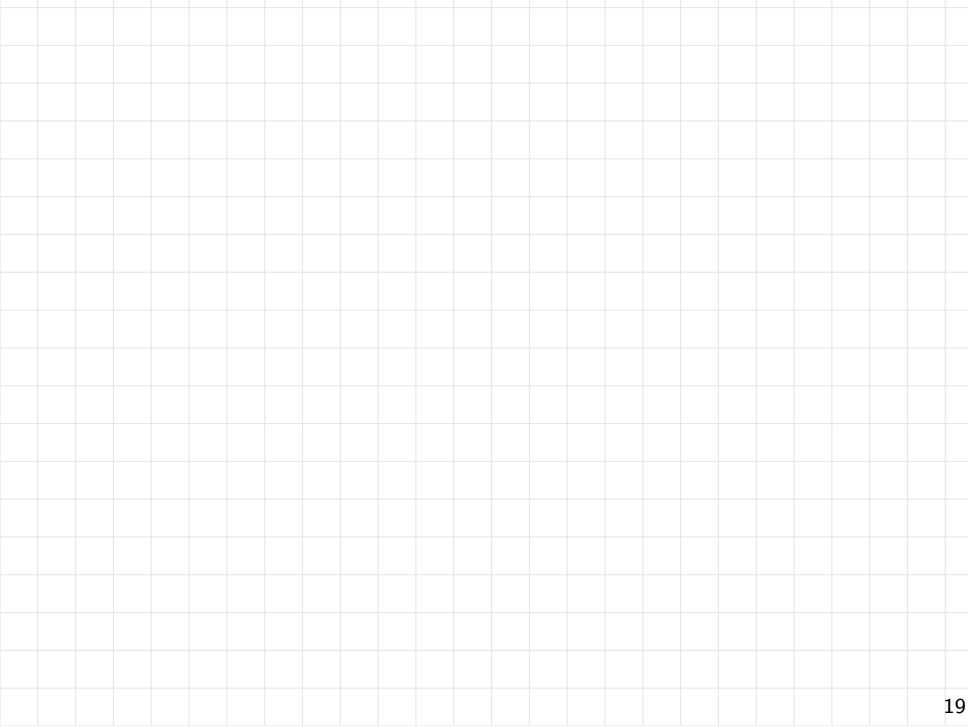


Java: implementation of the round

Alternatively, the combination of `groupByKey` and `mapValues` can be replaced by `reduceByKey` as follows:

```
JavaPairRDD<String, Long> wordCounts = docs
  .flatMapToPair(myMethods::wordCountPerDoc) // <-- MAP PHASE
  .reduceByKey((x, y) -> x+y);    // <-- REDUCE PHASE
```





PYTHON

Python: main function

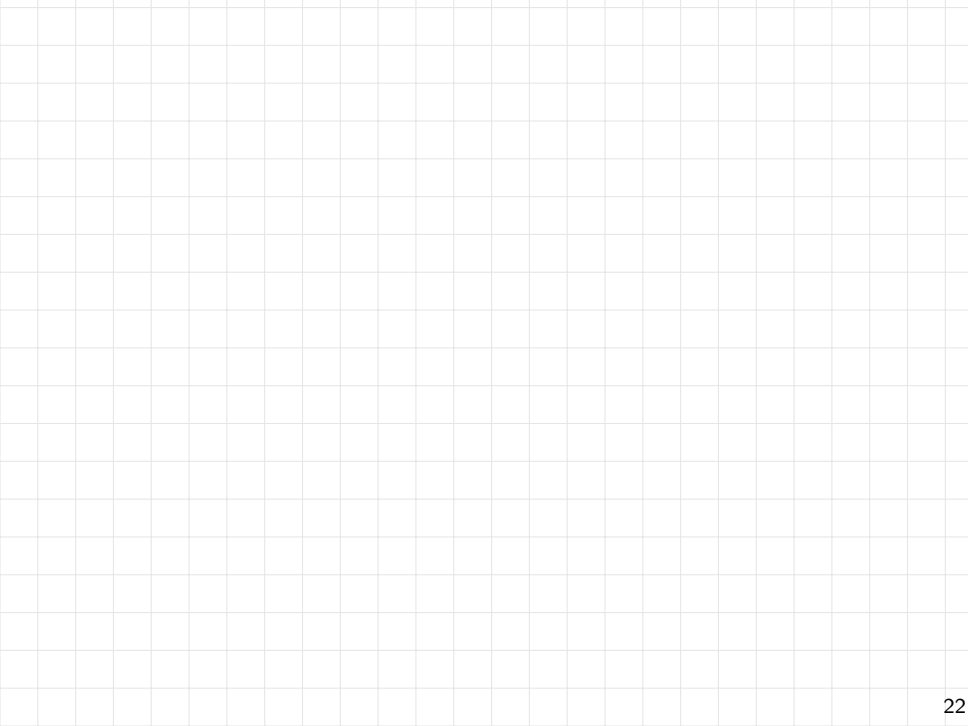
```
def main():

    # SPARK SETUP
    conf = SparkConf(true).setAppName('WordCount')
    sc = SparkContext(conf=conf)

    # INPUT READING
    K = sys.argv[1]
    K = int(K)
    data_path = sys.argv[2]
    docs = sc.textFile(data_path).repartition(K).cache()

    # COMPUTATION OF WORD COUNTS
    wordCounts = word_count_1(docs)

    if __name__ == "__main__":
        main()
```



Python: relevant functions

```
import ...

def word_count_per_doc(document):
    pairs_dict = {}
    for word in document.split(' '):
        if word not in pairs_dict.keys():
            pairs_dict[word] = 1
        else:
            pairs_dict[word] += 1
    return [(key, pairs_dict[key]) for key in pairs_dict.keys()]

def word_count_1(docs):
    word_count = (docs.flatMap(word_count_per_doc) # <-- MAP PHASE
                  .groupByKey()                    # <-- SHUFFLE+GROUPING
                  .mapValues(lambda vals: sum(vals))) # <-- REDUCE PHASE
    return word_count
```

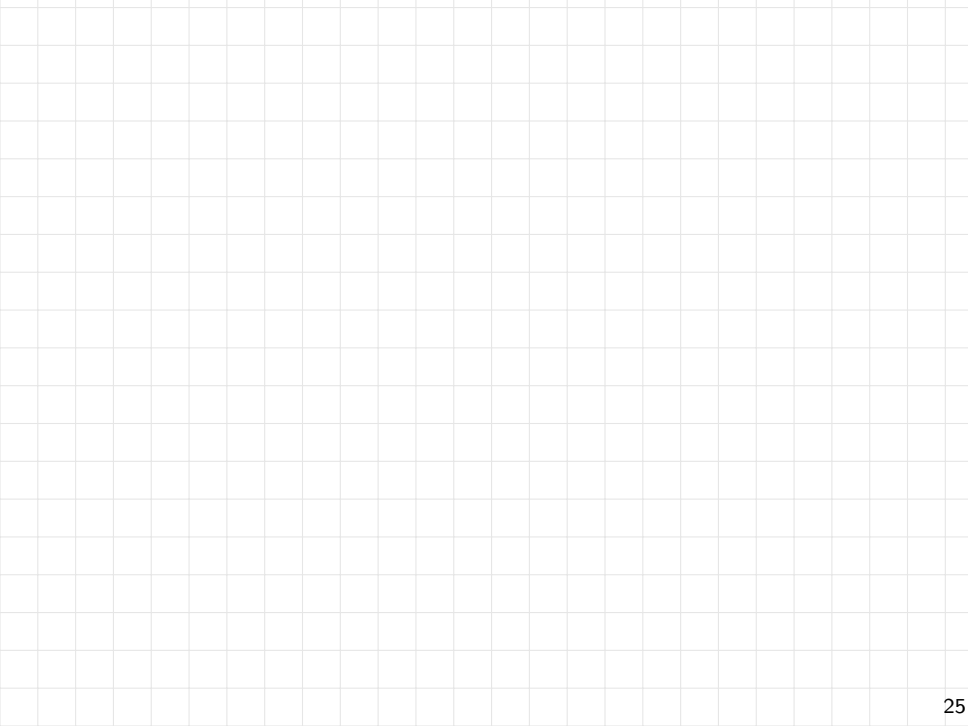
Python: relevant functions

Alternatively:

```
import ...

def word_count_per_doc(document):
    pairs_dict = {}
    for word in document.split(' '):
        if word not in pairs_dict.keys():
            pairs_dict[word] = 1
        else:
            pairs_dict[word] += 1
    return [(key, pairs_dict[key]) for key in pairs_dict.keys()]

def word_count_1(docs):
    word_count = (docs.flatMap(word_count_per_doc) # <-- MAP PHASE
                  .reduceByKey(lambda x, y: x + y)) # <-- REDUCE PHASE
    return word_count
```

2-Round Word Count in Spark

2-Round Word Count in Spark

General structure of the 2-Round Word Count algorithm

Round 1

- **Map Phase:** for each document D_i compute local count $c_i(w)$ of each word w and partition $(w, c_i(w))$ pairs among K partitions.
- **Reduce Phase:** for each word w aggregate local counts separately for each partition producing (at most) K pairs $(w, c(x, w))$, one for every partition x , with $0 \leq x < K$

Round 2

- **Map Phase:** empty.
- **Reduce Phase:** for each word w aggregate the $c(x, w)$ counts into the final count $c(w)$ and produce the pair $(w, c(w))$.

2-Round Word Count in Spark

We see 3 alternative implementations which use partitions:

- ① defined by random keys assigned in the Map phase of R1
- ② defined by random keys assigned on the fly during the group-by of R1
- ③ provided by Spark.

JAVA

Java: random keys assigned in the Map phase

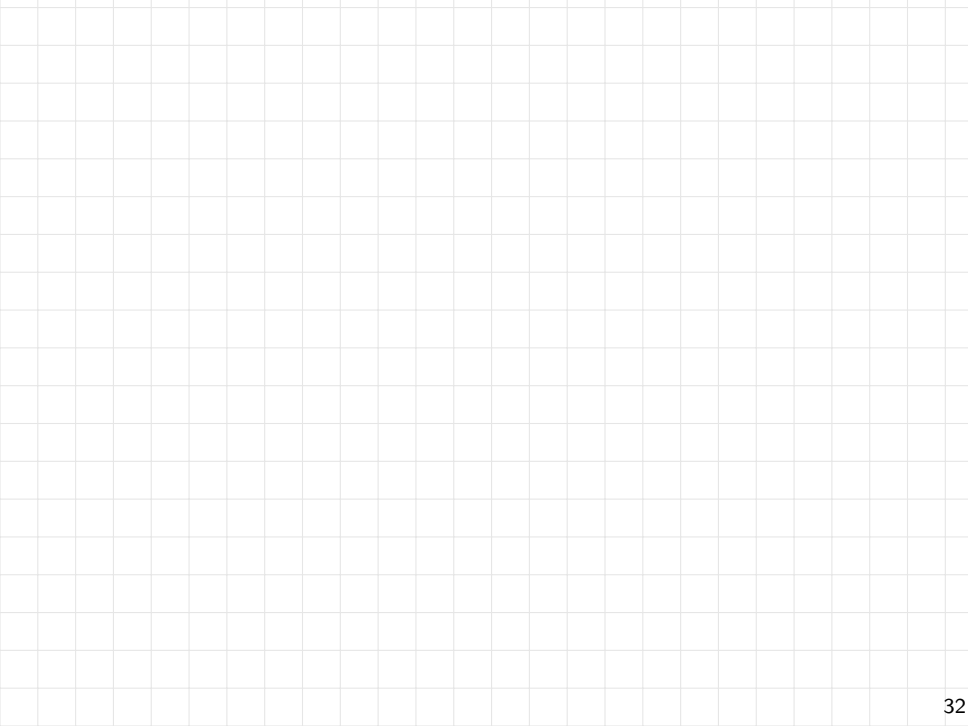
```
JavaPairRDD<String, Long> wordCounts = docs
    .flatMapToPair((document) -> {    // <-- MAP PHASE (R1)
        String[] tokens = document.split(" ");
        HashMap<String, Long> counts = new HashMap<>();
        ArrayList<Tuple2<Integer, Tuple2<String, Long>>>
            pairs = new ArrayList<>();
        for (String token : tokens) {
            counts.put(token, 1L + counts.getOrDefault(token, 0L));
        }
        for (Map.Entry<String, Long> e : counts.entrySet()) {
            pairs.add(new Tuple2<>(randomGenerator.nextInt(K),
                                   new Tuple2<>(e.getKey(), e.getValue())));
        }
        return pairs.iterator();
    })
    .groupByKey()    // <-- SHUFFLE AND GROUPING
```

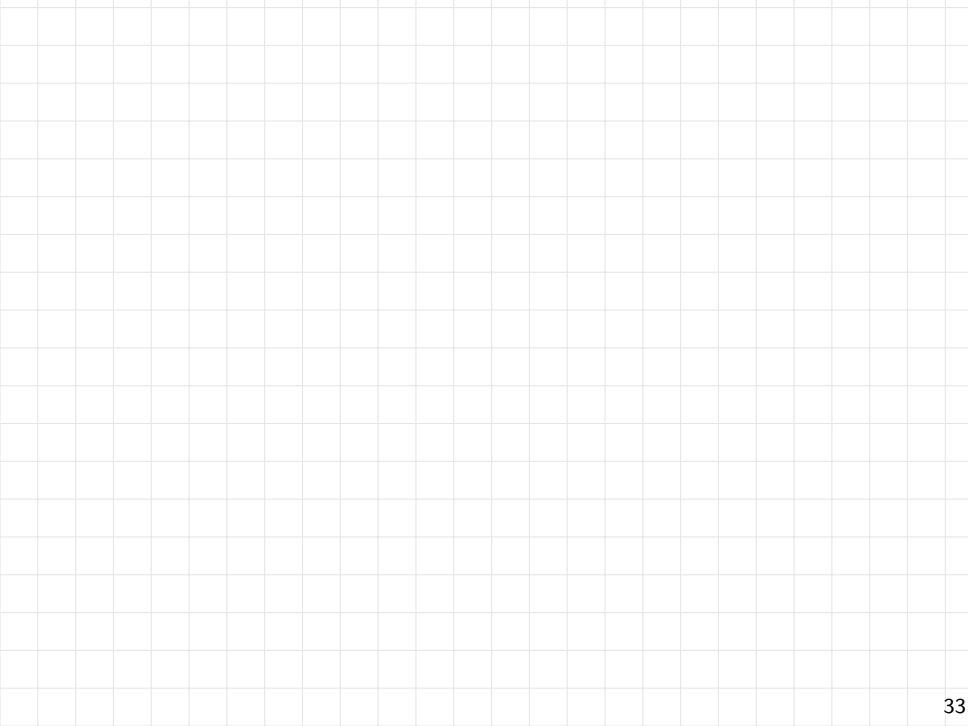
```

.flatMapToPair(myMethods::gatherPairs) <-- REDUCE PHASE (R1)
.reduceByKey((x, y) -> x+y); // <-- REDUCE PHASE (R2)

class myMethods {
    .....
    public static Iterator<Tuple2<String,Long>> gatherPairs
        (Tuple2<Integer,Iterable<Tuple2<String, Long>>> element) {
        HashMap<String, Long> counts = new HashMap<>();
        for (Tuple2<String, Long> c : element._2()) {
            counts.put(c._1(),c._2()+counts.getDefault(c._1(),0L));
        }
        ArrayList<Tuple2<String, Long>> pairs = new ArrayList<>();
        for (Map.Entry<String, Long> e : counts.entrySet()) {
            pairs.add(new Tuple2<>(e.getKey(), e.getValue()));
        }
        return pairs.iterator();
    }
}

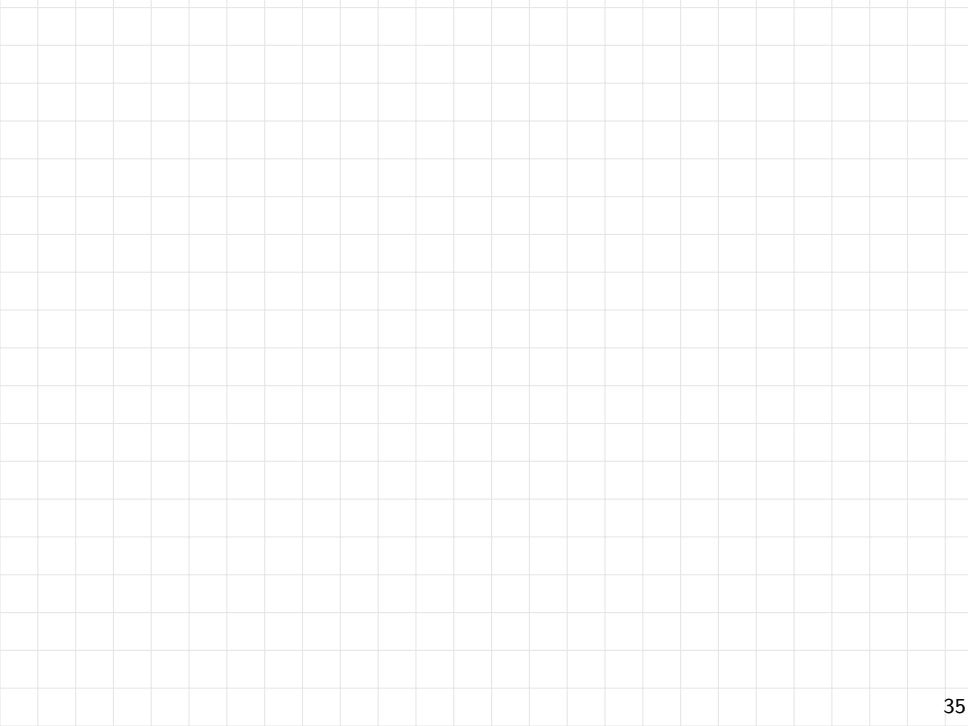
```





Java: random keys assigned on-the-fly

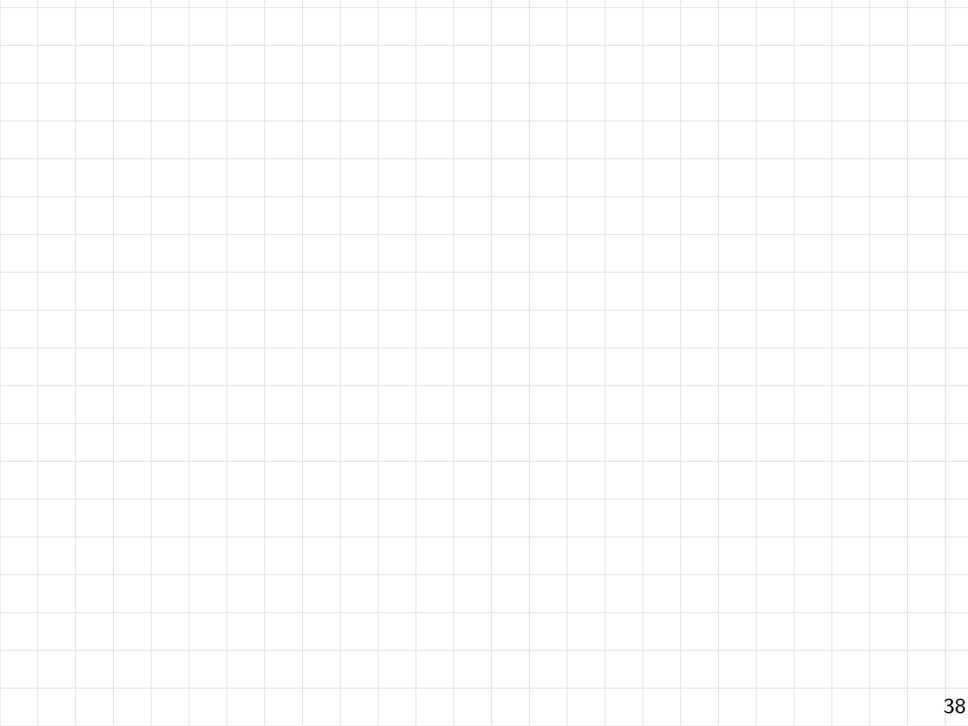
```
JavaPairRDD<String, Long> wordCounts = docs
    .flatMapToPair((document) -> {    // <-- MAP PHASE (R1)
        String[] tokens = document.split(" ");
        HashMap<String, Long> counts = new HashMap<>();
        ArrayList<Tuple2<String, Long>> pairs = new ArrayList<>();
        for (String token : tokens) {
            counts.put(token, 1L + counts.getOrDefault(token, 0L));
        }
        for (Map.Entry<String, Long> e : counts.entrySet()) {
            pairs.add(new Tuple2<>(e.getKey(), e.getValue()));
        }
        return pairs.iterator();
    })
    .groupBy((wordcountpair) -> randomGenerator.nextInt(K))
    // IMPLEMENTS RANDOM KEY ASSIGNMENT, SHUFFLE AND GROUPING
    .flatMapToPair(myMethods::gatherPairs) <-- REDUCE PHASE (R1)
    .reduceByKey((x, y) -> x+y); // <-- REDUCE PHASE (R2)
```



Java: use of Spark partitions

```
JavaPairRDD<String, Long> wordCounts = docs
    .flatMapToPair(myMethods::wordCountPerDoc)
    .mapPartitionsToPair((element) -> {    // <-- REDUCE PHASE (R1)
        HashMap<String, Long> counts = new HashMap<>();
        while (element.hasNext()){
            Tuple2<String, Long> tuple = element.next();
            counts.put(tuple._1(), tuple._2() +
                        counts.getDefault(tuple._1(), 0L));
        }
        ArrayList<Tuple2<String, Long>> pairs = new ArrayList<>();
        for (Map.Entry<String, Long> e : counts.entrySet()) {
            pairs.add(new Tuple2<>(e.getKey(), e.getValue()));
        }
        return pairs.iterator();
    })
```

```
.groupByKey()          // <-- SHUFFLE+GROUPING
.mapValues((it) -> { // <-- REDUCE PHASE (R2)
    long sum = 0;
    for (long c : it) sum += c;
    return sum;
});
// reduceByKey CAN BE USED IN PLACE OF groupByKey AND mapValues
```



PYTHON

Python: relevant functions

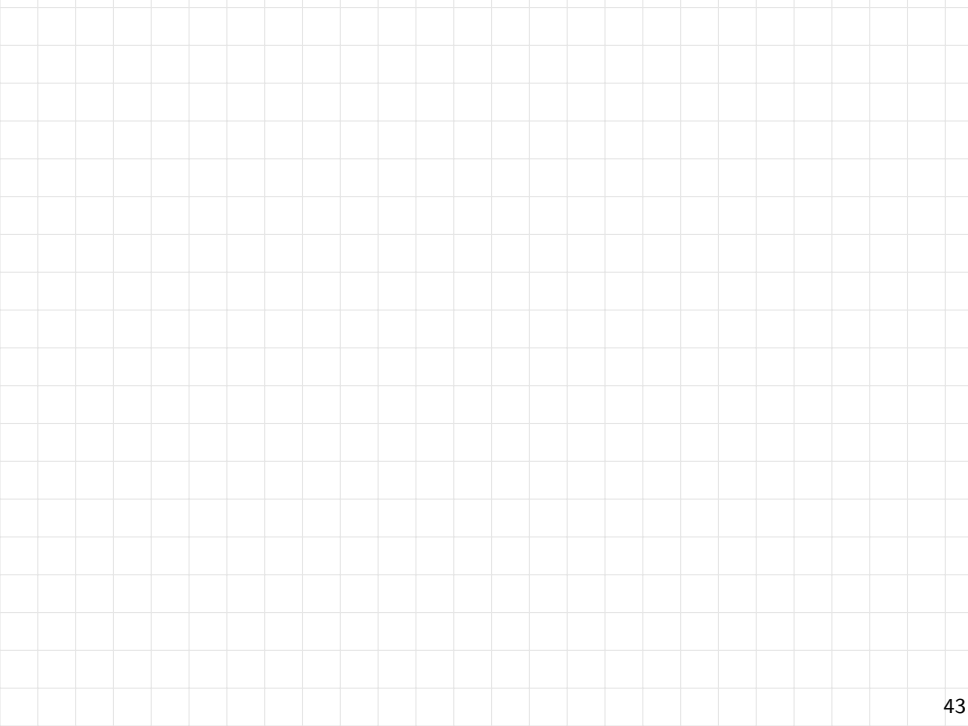
```
def word_count_per_doc(document,F=-1):
    pairs_dict = {}
    for word in document.split(' '):
        if word not in pairs_dict.keys():
            pairs_dict[word] = 1
        else:
            pairs_dict[word] += 1
    if F == -1:
        return [(key, pairs_dict[key]) for key in pairs_dict.keys()]
    else:
        return [(rand.randint(0,F-1),(key, pairs_dict[key]))
                for key in pairs_dict.keys()]
```


Python: relevant functions

```
def gather_pairs(pairs):  
    pairs_dict = {}  
    for p in pairs[1]:  
        word, occurrences = p[0], p[1]  
        if word not in pairs_dict.keys():  
            pairs_dict[word] = occurrences  
        else:  
            pairs_dict[word] += occurrences  
    return [(key, pairs_dict[key]) for key in pairs_dict.keys()]
```

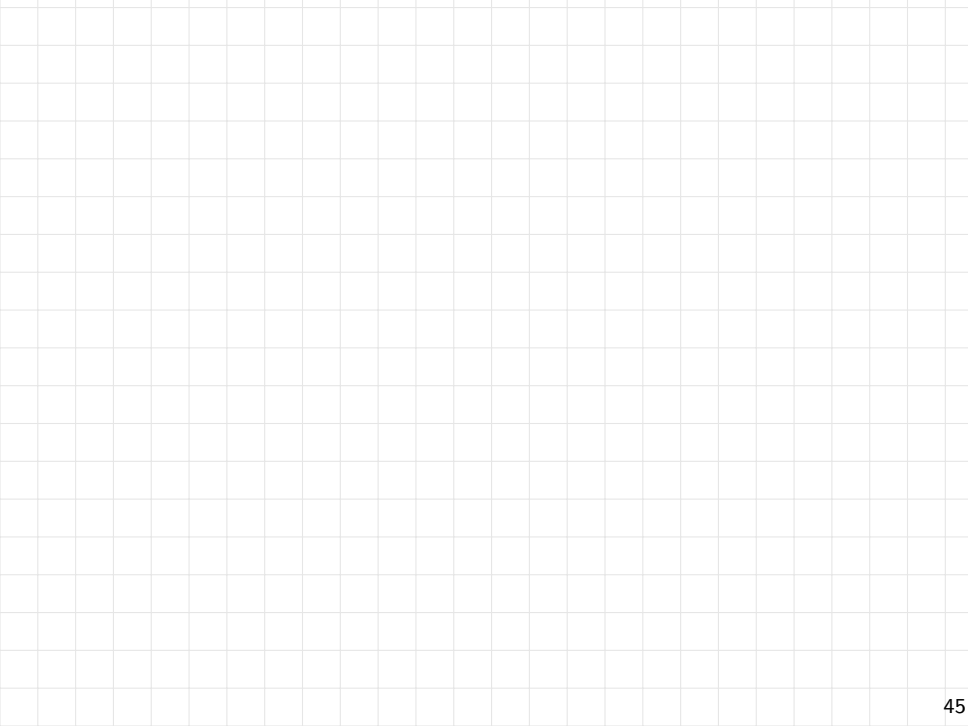
Python: random keys assigned in the Map phase

```
def word_count_2(docs, K):  
    word_count = (docs.flatMap(lambda x: word_count_per_doc(x, K))  
                  # <-- MAP PHASE (R1)  
                  .groupByKey() # <-- SHUFFLE+GROUPING  
                  .flatMap(gather_pairs) # <-- REDUCE PHASE (R1)  
                  .reduceByKey(lambda x, y: x + y)) # <-- REDUCE PHASE (R2)  
    return word_count
```



Python: random keys assigned on-the-fly

```
def word_count_3(docs, K):  
    word_count = (docs.flatMap(word_count_per_doc) # <-- MAP PHASE (R1)  
        .groupBy(lambda x: (rand.randint(0,K-1))) # <-- SHUFFLE+GROUPING  
        .flatMap(gather_pairs) # <-- REDUCE PHASE (R1)  
        .reduceByKey(lambda x, y: x + y)) # <-- REDUCE PHASE (R2)  
    return word_count
```



Python: use of Spark partitions

```
def gather_pairs_partitions(pairs):
    pairs_dict = {}
    for p in pairs:
        word, occurrences = p[0], p[1]
        if word not in pairs_dict.keys():
            pairs_dict[word] = occurrences
        else:
            pairs_dict[word] += occurrences
    return [(key, pairs_dict[key]) for key in pairs_dict.keys()]

def word_count_with_partition(docs):
    word_count = (docs.flatMap(word_count_per_doc) # <-- MAP PHASE (R1)
        .mapPartitions(gather_pairs_partitions) # <-- REDUCE PHASE (R1)
        .groupByKey() # <-- SHUFFLE+GROUPING
        .mapValues(lambda vals: sum(vals))) # <-- REDUCE PHASE (R2)
    return word_count
```

