Basic Spark 2 – Part 1

DSC 232R

1.1 Chaining

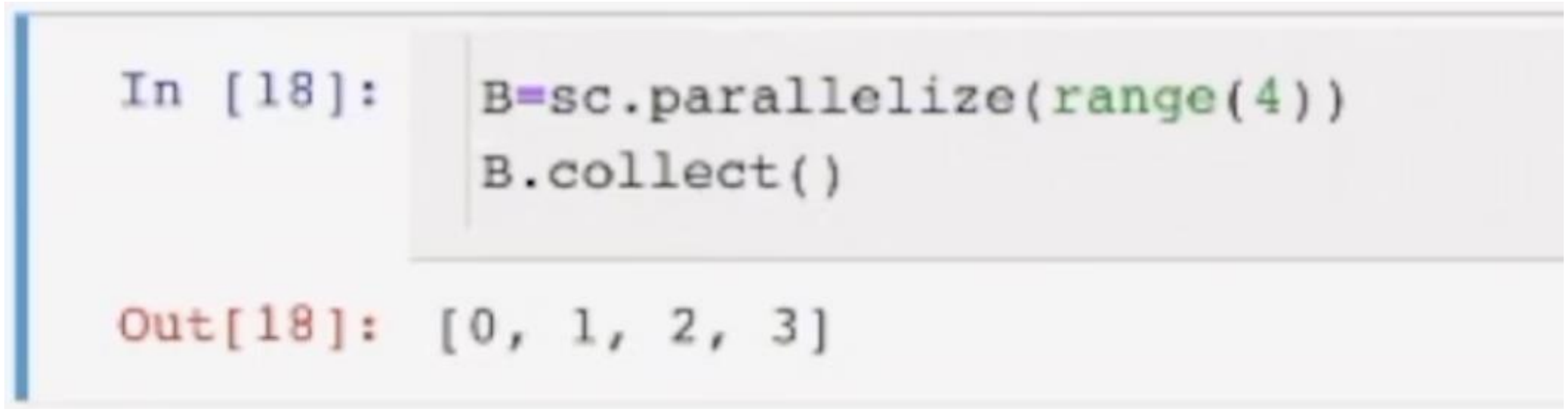
We can **chain** transformations and actions to create a computation **pipeline**

Suppose we want to compute the sum of the squares

$$\sum_{i=1}^{n} {x^2}_i$$

where the elements x_i are stored in an RDD.

1.1.1 Create an RDD

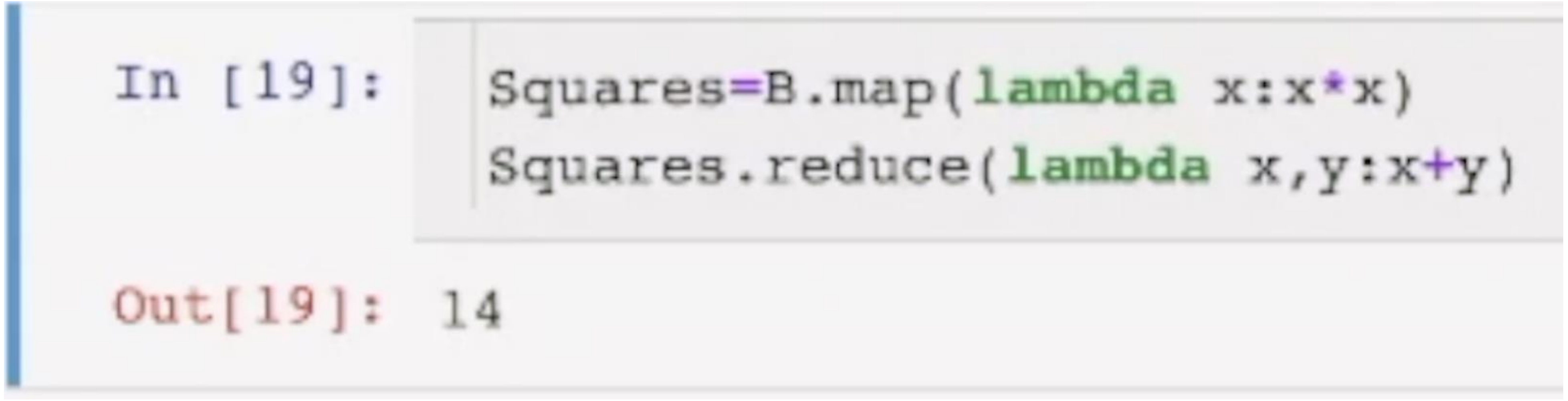


```
In [18]:  B=sc.parallelize(range(4))
          B.collect()

Out[18]: [0, 1, 2, 3]
```

1.1.2 Sequential syntax for chaining

Perform assignment after each computation

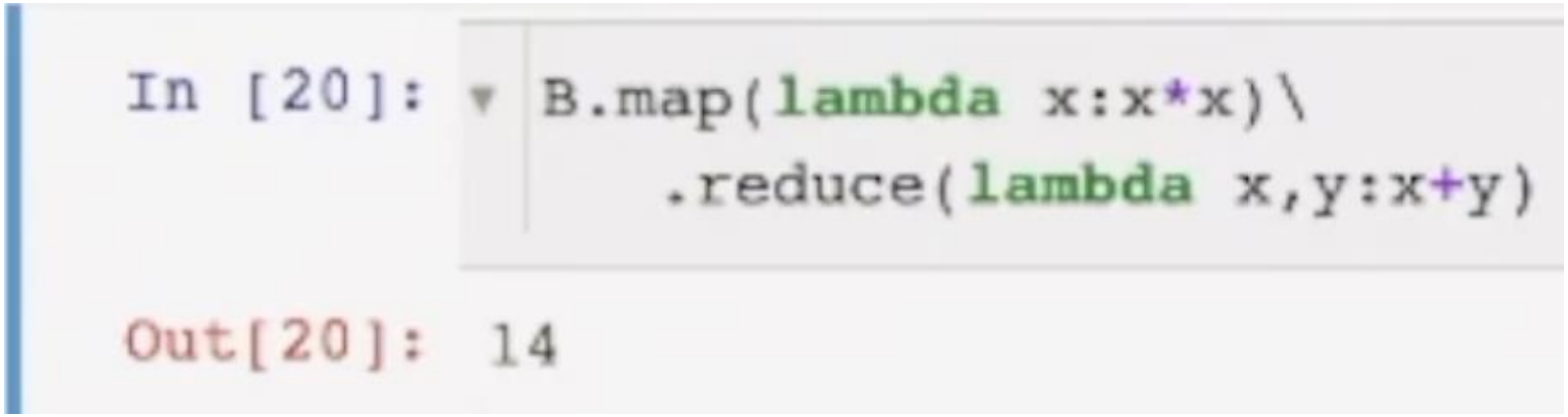


```
In [19]:  Squares=B.map(lambda x:x*x)
          Squares.reduce(lambda x,y:x+y)

Out[19]:  14
```

1.1.3 Cascaded syntax for chaining

Combine computations into a single cascaded command



```
In [20]:  B.map(lambda x:x*x)\
               .reduce(lambda x,y:x+y)

Out[20]:  14
```

1.1.4 Both syntaxes mean exactly the same thing

The only difference:

- In the sequential syntax the intermediate RDD has a name `Squares`
- In the cascaded syntax the intermediate RDD is *anonymous*

The execution is identical!

1.1.5 Sequential execution

The way MapReduce are executed by a standard system (not Spark!)

- perform the map
- store the resulting RDD in memory
- perform the reduce

1.1.6 Disadvantages of Sequential execution

1. Intermediate result (`Squares`) requires memory space.
2. Two scans of memory (of `B`, then of `Squares`) – double the cache-misses.

1.1.7 Pipelined execution

Spark performs the whole computation in a single pass. For each element of `B`

1. Compute the square.
2. Enter the square as input to the `reduce` operation.

1.1.8 Advantages of Pipelined execution

1. Less memory required – intermediate result is not stored.
2. Faster – only one pass through the Input RDD.

1.1.9 Lazy Evaluation

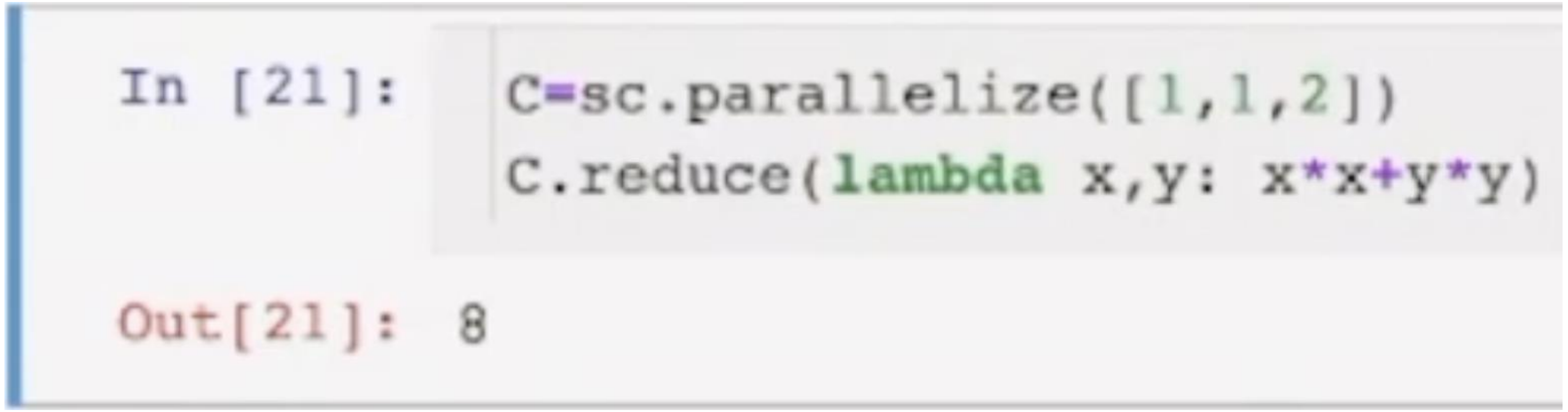
This type of pipelined evaluation is related to **Lazy Evaluation**. The word **Lazy** is used because the first command (computing the square) is not executed immediately. Instead, the execution is delayed as long as possible so that several commands are executed in a single pass.

The delayed commands are organized in an **Execution plan**.

For more on Pipelined execution, Lazy evaluation and Execution Plans see spark programming guide/RDD operations

1.1.10 An instructive mistake

Here is another way to compute the sum of the squares using a single reduce command. Can you figure out how it comes up with this unexpected result?



```
In [21]:  C=sc.parallelize([1,1,2])
          C.reduce(lambda x,y: x*x+y*y)

Out[21]:  8
```

1.1.10.1 Answer:

1. `reduce` first operates on the pair (1,1), replacing it with $1^2 + 1^2 = 2$
2. `reduce` then operates on the pair (2,2), giving the final result $2^2 + 2^2 = 8$