Spark Basics

- Spark Context
- Resilient Distributed Dataset (RDD)

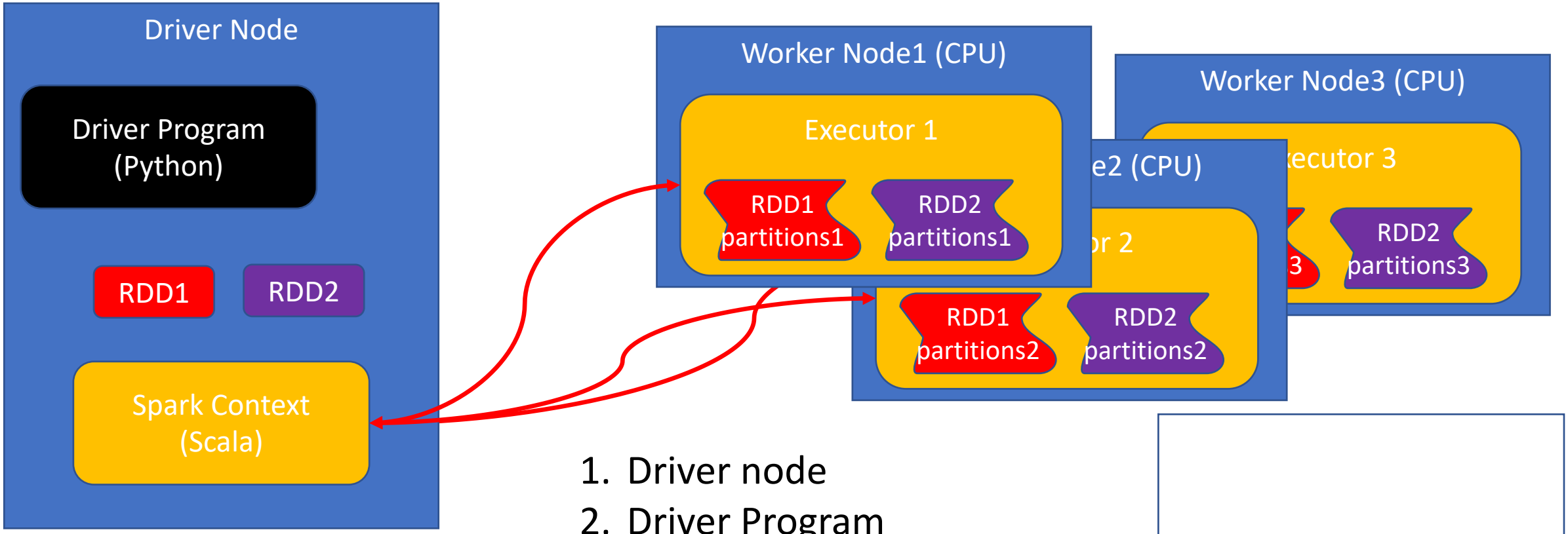
Spark Context

- Spark is complex distributed software.
- The python interface to spark is called **pyspark**
- **SparkContext** is a python class, defined as part of **pyspark** which manages the communication between the user's program and spark.
- We start by creating a **SparkContext** object named **sc**.

```
In [137]: from pyspark import SparkContext
          sc = SparkContext(master="local[3]")
          sc

Out[137]: <pyspark.context.SparkContext at 0x10855f690>
```

Resilient Distributed DataSets (RDDs)



1. Driver node
2. Driver Program
3. Spark Context
4. RDDs
5. Partitions
6. Executors

Parallelize

- Simplest way to create an RDD.
- The method `A=sc.parallelize(L)`, creates an RDD named `A` from list `L`.
- `A` is an RDD of type `ParallelCollectionRDD`.

```
A=sc.parallelize(range(3))
A
```

```
ParallelCollectionRDD[0] at parallelize at PythonRDD.scala:423
```

Collect

- RDD content is distributed among all executors.
- `collect()` is the inverse of `parallelize()'
- collects the elements of the RDD
- Returns a list

In [140]:
```
L=A.collect()
print type(L)
print L
```

```
<type 'list'>
[0, 1, 2]
```

Map

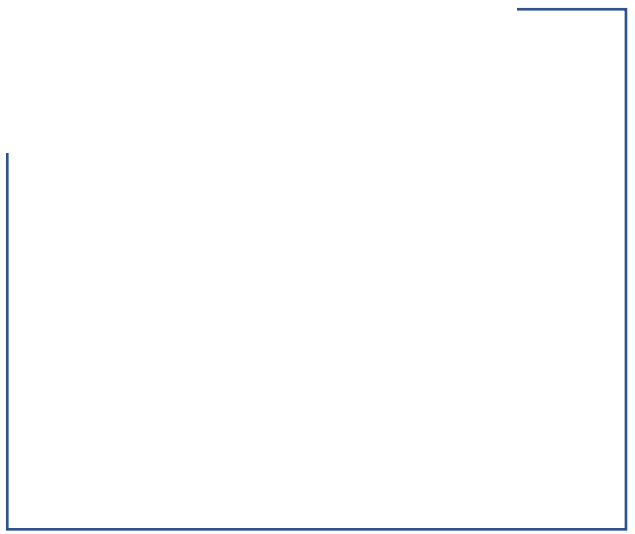
- applies a given operation to each element of an RDD
- parameter is the function defining the operation.
- returns a new RDD.
- Operation performed in parallel on all executors.
- Each executor operates on the data **local** to it.

```
In [141]: A.map(lambda x: x*x).collect()

Out[141]: [0, 1, 4]
```

Reduce

- Takes RDD as input, returns a single value.
- **Reduce operator** takes **two** elements as input returns **one** as output.
- Repeatedly applies a **reduce operator**
- Each executor reduces the data local to it.
- The results from all executors are combined.



The simplest example of a 2-to-1 operation is the sum:

```
In [144]: A.reduce(lambda x,y: x+y)

Out[144]: 3
```

Another example:
Find the shortest word in a list

```
In [145]: words=['this','is','the','best','mac','ever']
          wordRDD=sc.parallelize(words)
          wordRDD.reduce(lambda w,v: w if len(w)<len(v) else v)

Out[145]: 'is'
```

Using regular functions instead of lambda functions

- lambda function are short and sweet.
- but sometimes it's hard to use just one line.
- We can use full-fledged functions instead.

suppose we want to find the

- last word in a lexicographical order
- among
- the longest words in the list.

We could achieve that as follows

In [146]:
```
def largerThan(x,y):
    if len(x)>len(y): return x
    elif len(y)>len(x): return y
    else:  #lengths are equal, compare lexicographically
        if x>y:
            return x
        else:
            return y

wordRDD.reduce(largerThan)
```

Out[146]:
```
'this'
```

Summary

- Spark - Context
- RDDs
- Map
- Reduce
- More details, and excercises, in the jupyter notebook.
- Next time: more about RDDs