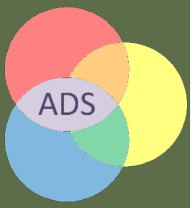
Data Science & Society

Lecture 04: *Spark – Architecture & Transformations*

INFOMDSS 2018 :: Dr. Marco Spruit



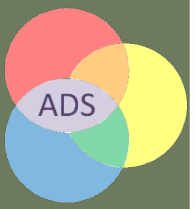
Agenda

- Book review Q/A

- Spark
  - High-level Architecture
  - The concept of Resilient Distributed Datasets (RDDs)
  - Narrow Transformations in Spark
  - ~~Wide Transformations in Spark~~

Architecture of Spark

A high-level outlook onto its components

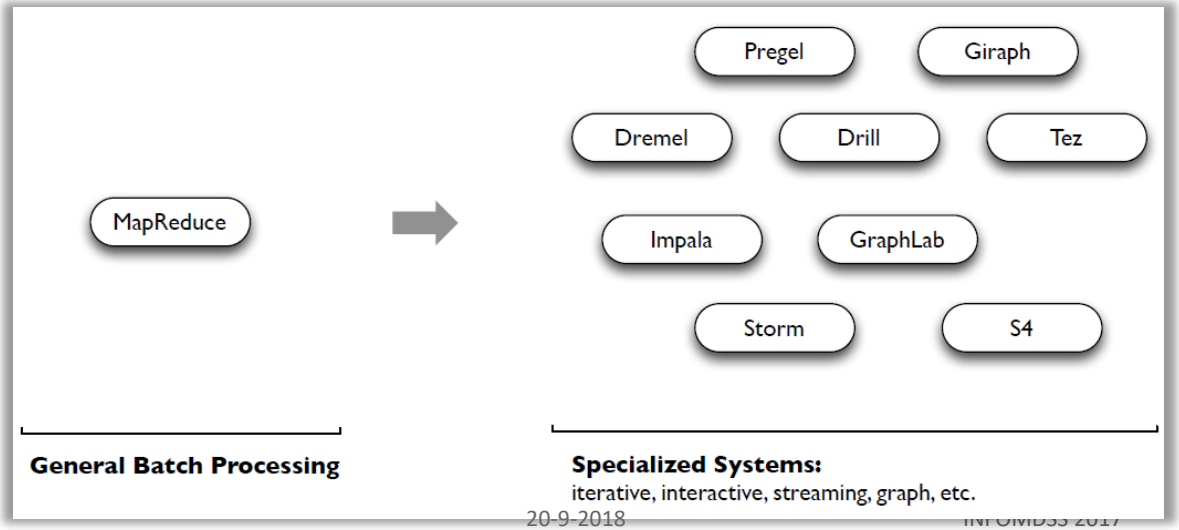


From lecture 03

Recap: Shortcomings of MapReduce

1. Force your pipeline into Map and Reduce steps
   – Other workflows? i.e. join, filter, map-reduce-map
2. Read from disk for each MapReduce job
   – Iterative algorithms? i.e. machine learning
3. Only native JAVA programming interface
   – Other languages? Interactivity?

How Java's Floating-Point Hurts Everyone Everywhere



From lecture 03

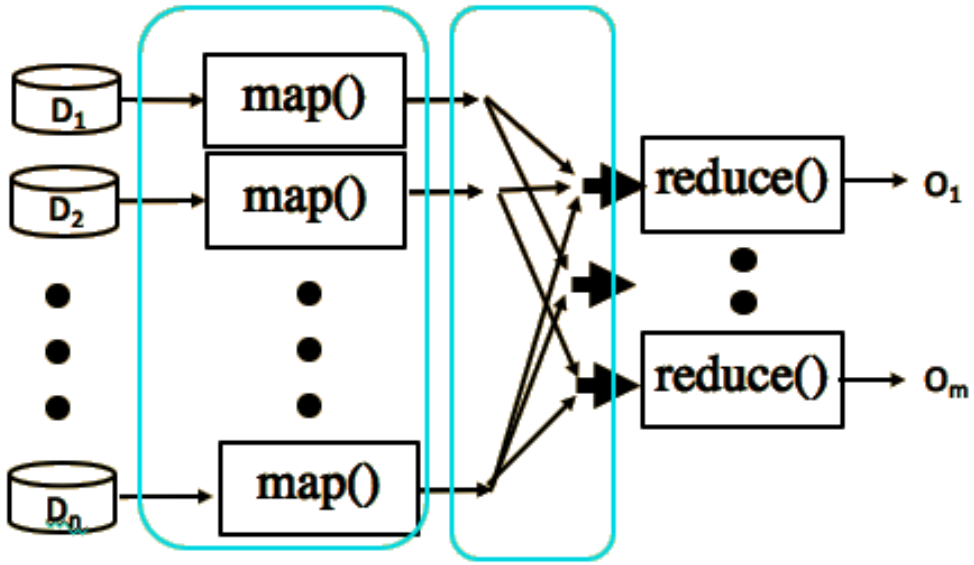
Recap: Solutions by Spark

1. Force your pipeline into Map and Reduce steps
   - Other workflows? i.e. join, filter, map-reduce-map
   - 20 highly efficient distributed operations, any combination of them
2. Read from disk for each MapReduce job
   - Iterative algorithms? i.e. machine learning
   - in-memory caching of data, specified by the user
3. Only native JAVA programming interface
   - Other languages? Interactivity?
   - Native Python, Scala (, R) interface. Interactive shells

From lecture 02

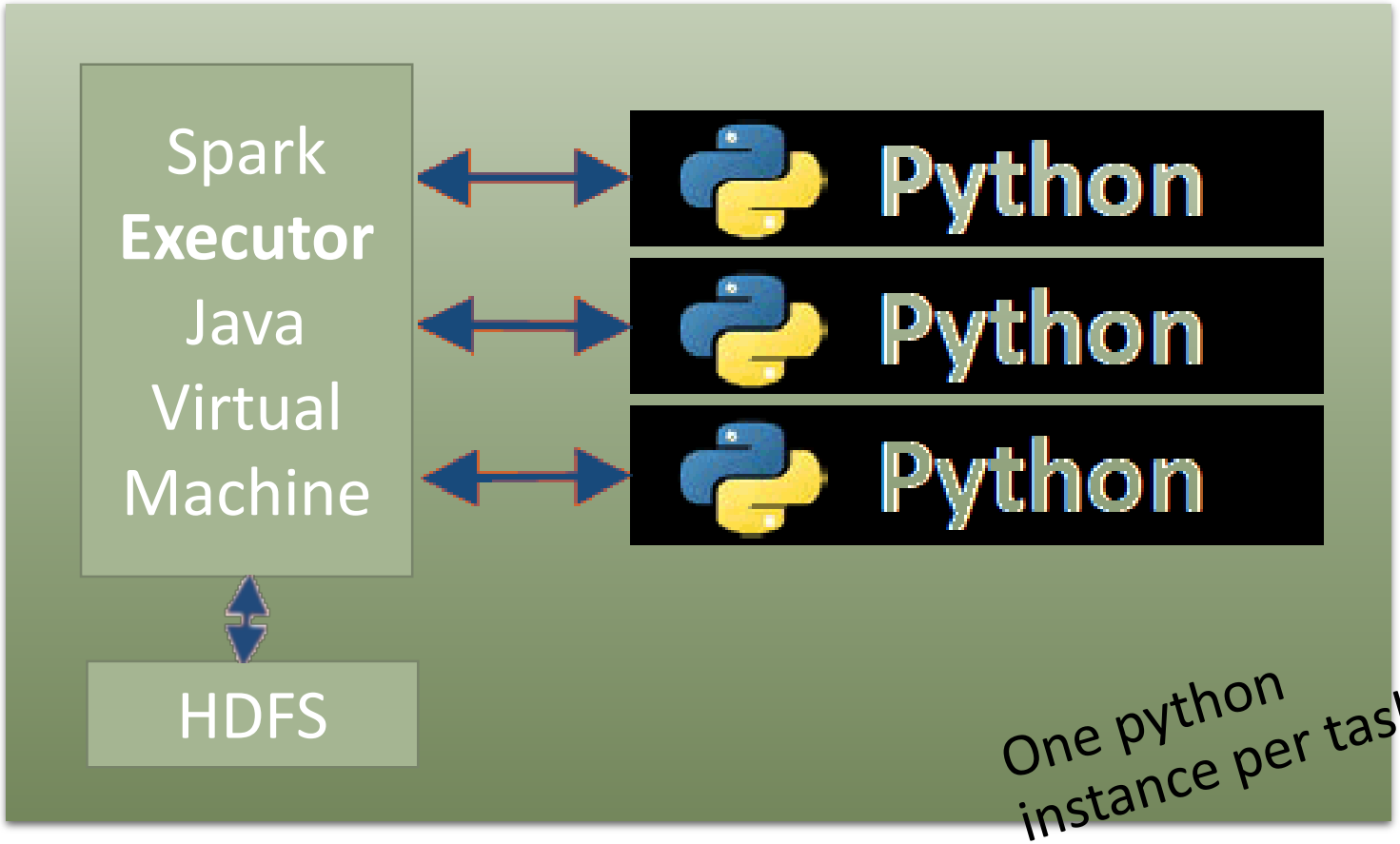
Recap: Original MapReduce Execution Framework

- Software framework which
  - Schedules, monitors, and manages tasks
- Works for Applications that fit MapReduce paradigm

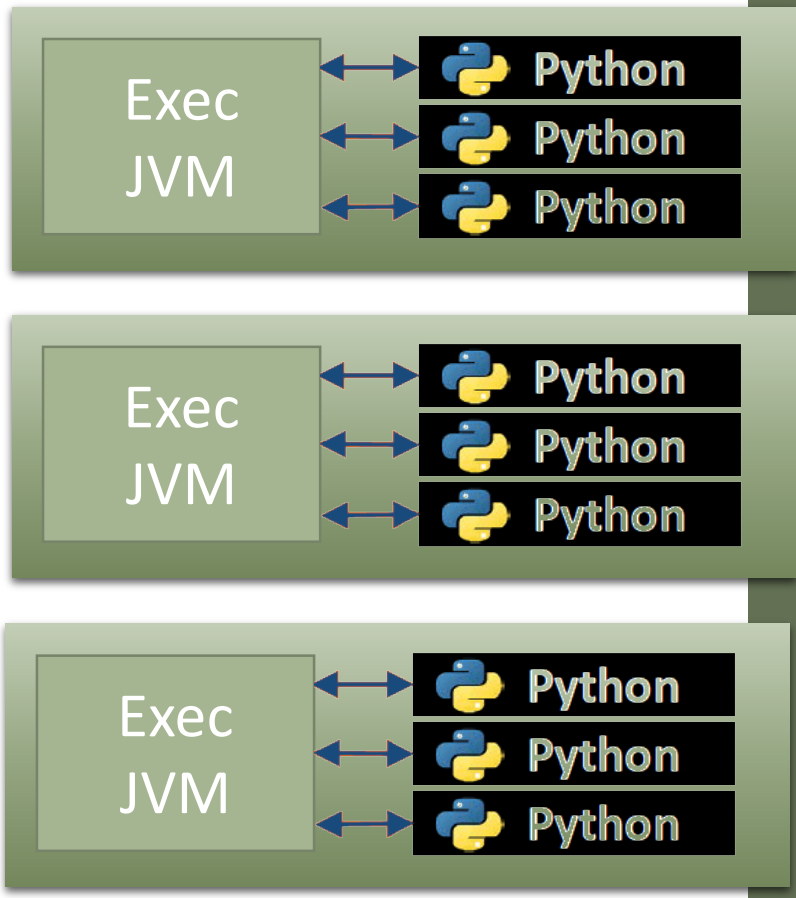


One map() executor per worker node

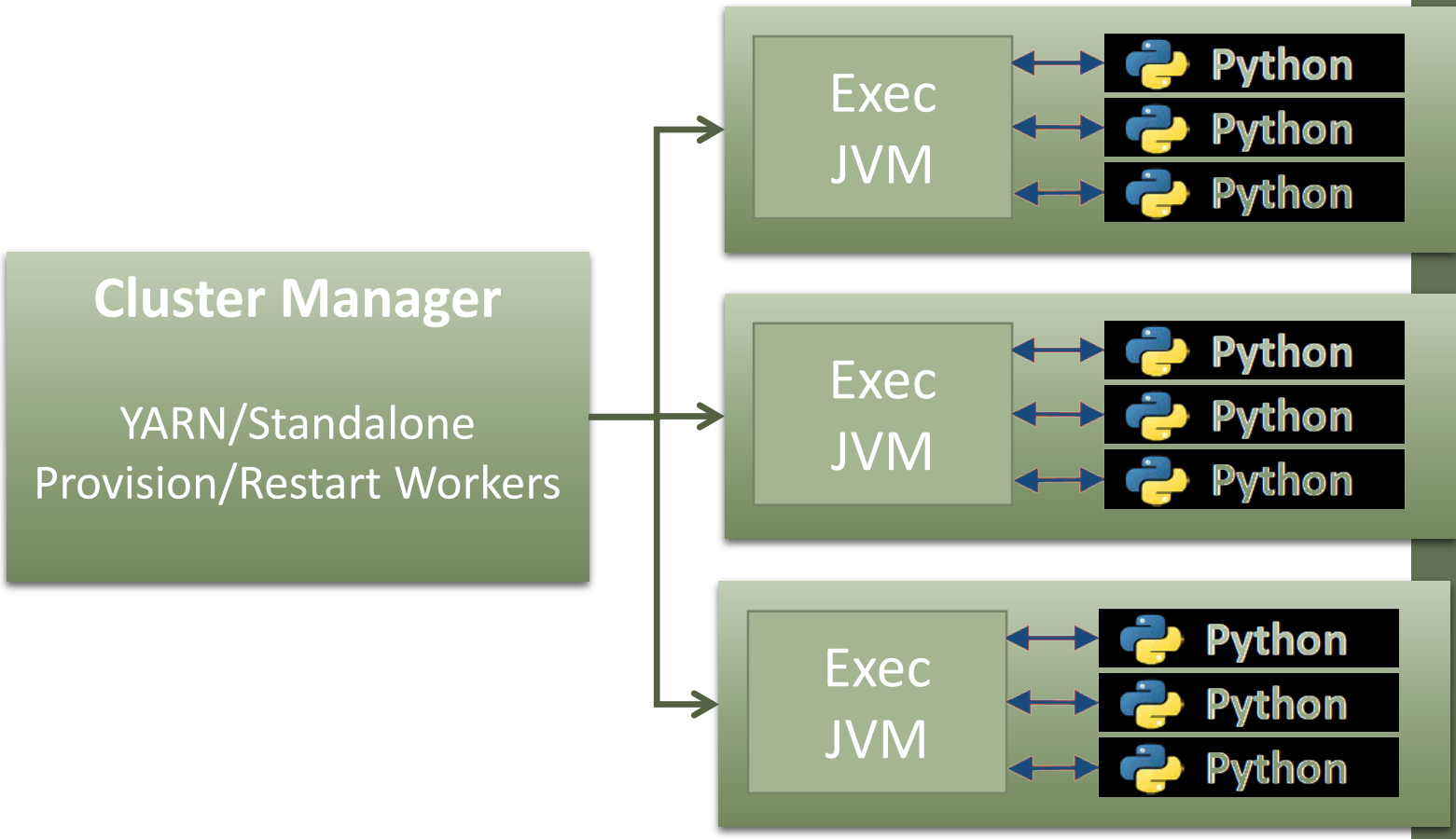
Spark Architecture: One Worker node



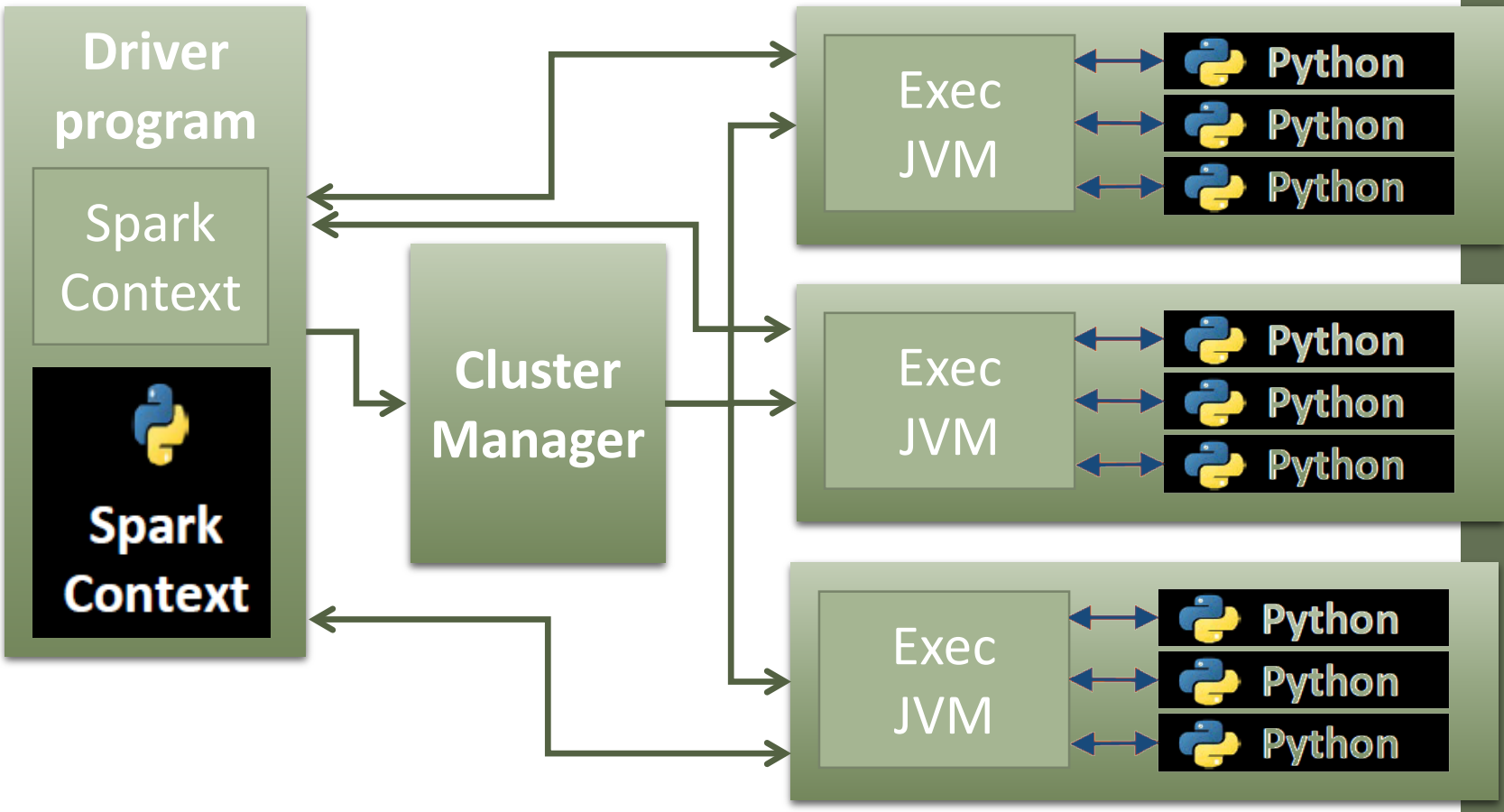
Multiple Worker nodes



Multiple Worker nodes, managed

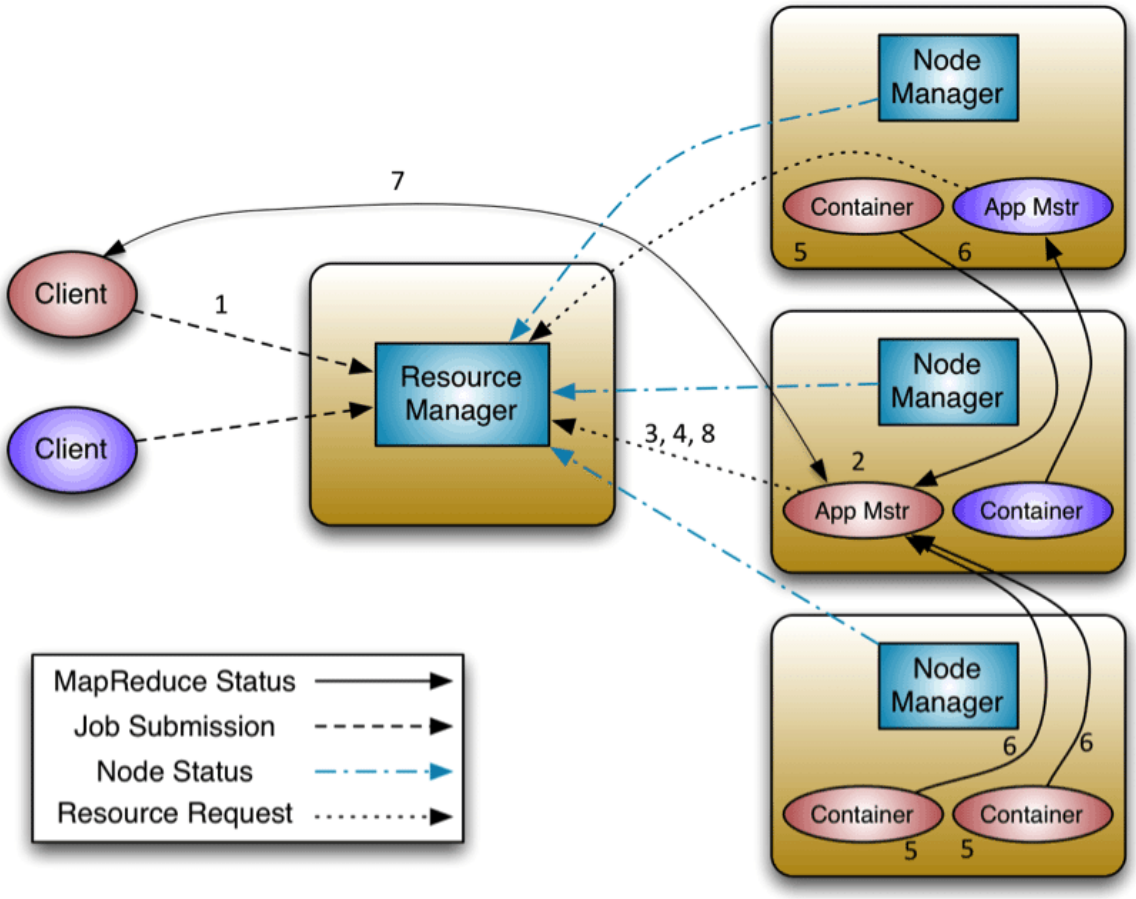


Driver Program



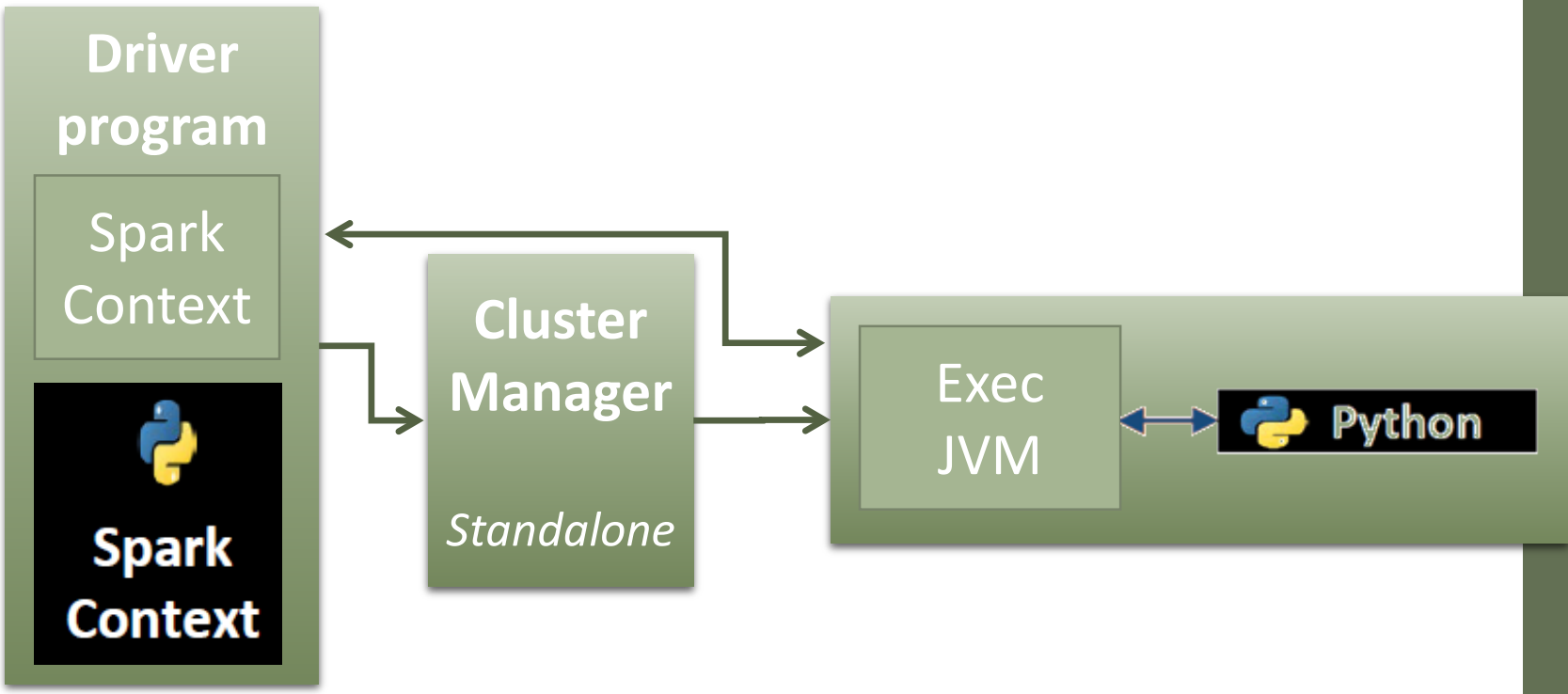
Hadoop's YARN

From lecture 02

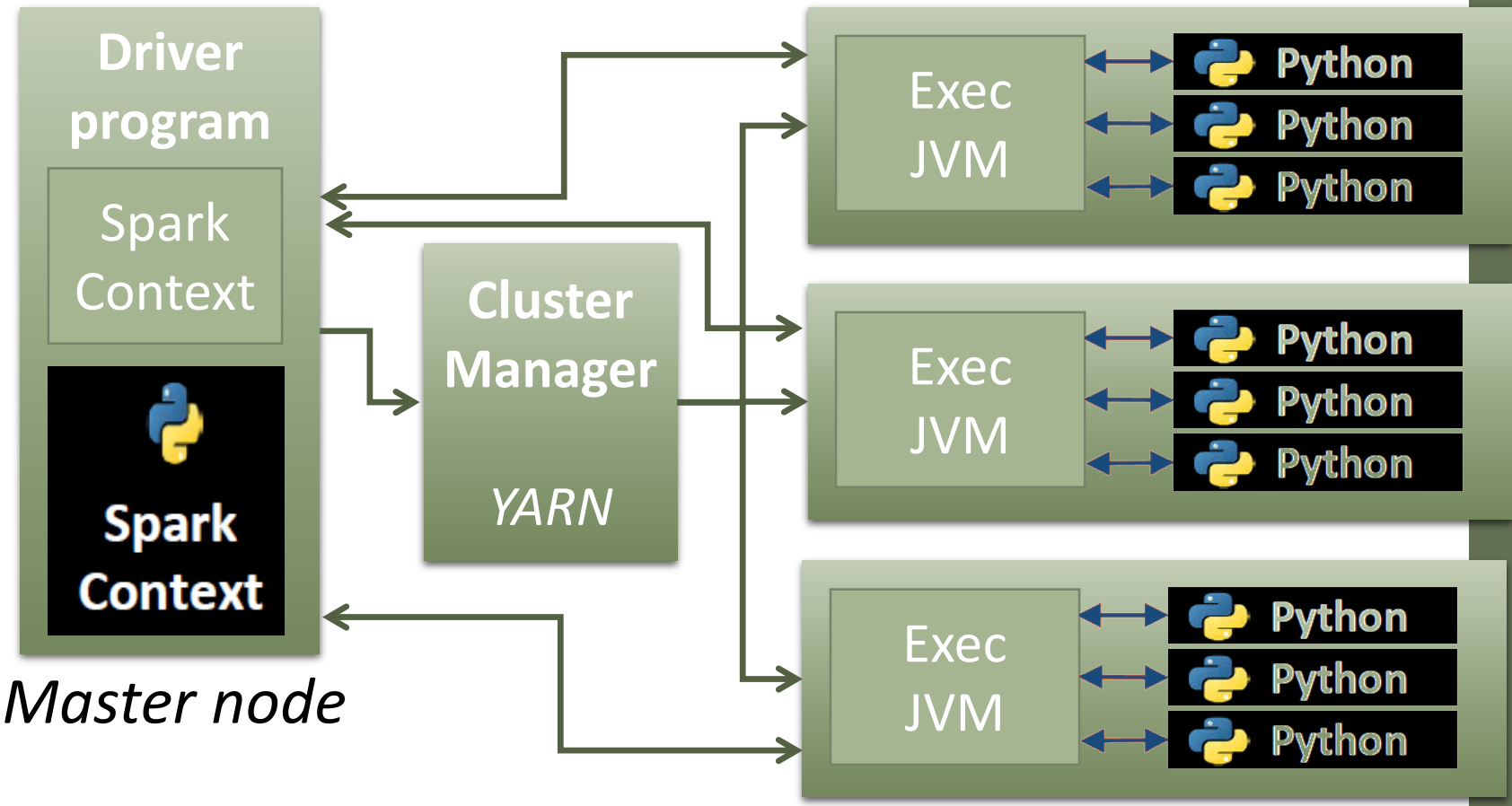



ADS

On a local VM

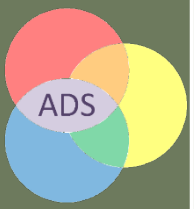

Driver program
Spark Context
Spark Context
Cluster Manager
Standalone
Exec JVM
Python

On Azure, or Amazon Elastic MapReduce (EMR)



Resilient Distributed Datasets

RDDs in Apache Spark



What's in a word? Resilient Distributed Dataset

Dataset
Data storage created from: HDFS, S3, HBase, JSON, text, Local hierarchy of folders
Or created transforming another RDD

*RDDs are **immutable**:*
i.e. You cannot change just a chunk of them

What's in a word? Resilient Distributed Dataset

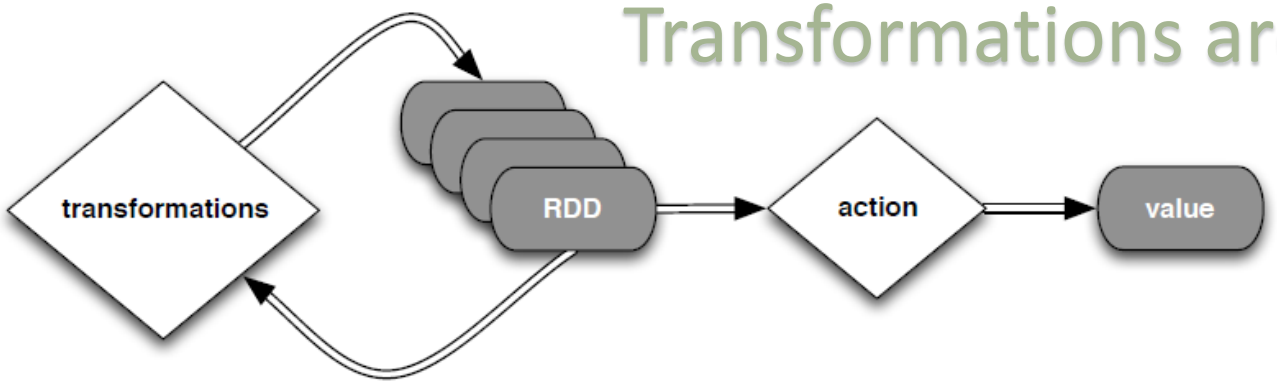
Distributed
Distributed across the cluster of machines
Divided in partitions, atomic chunks of data

Spark works with partitions
i.e. Not just fixed line by line processing

What's in a word? Resilient Distributed Dataset

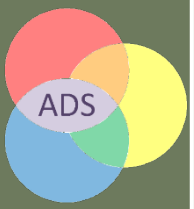
Resilient
Recover from errors, e.g. node failure, slow processes
Track history of each partition, re-run

Transformations are Lazy



(Narrow) Transformations

in Apache Spark



Transformations

- RDD are immutable
- Never modify RDD in place
- Transform RDD to another RDD
- Lazy

Some available transformations

- `map(func)`
  - apply function to each element of RDD
- `flatMap(func)`
  - map then flatten output
- `filter(func)`
  - keep only elements where func is `true`
- `sample(withReplacement,fraction,seed)`
  - get a random data fraction
- `coalesce(numPartitions)`
  - merge partitions to reduce them to numPartitions

Example transformation: `map(func)`

1. Create RDD
   – from local filesystem:

```
text_RDD = sc.textFile("file:///home/cloudera/testfile1")

text_RDD.collect()
    Out[n]: [u'A long time ago in a galaxy far far away']
```

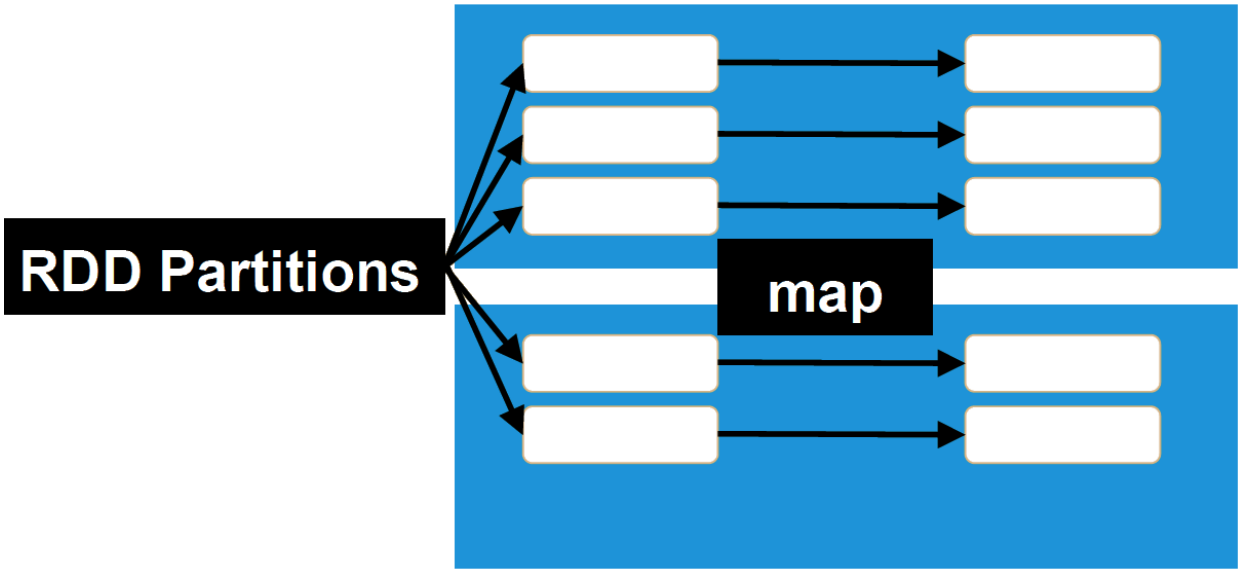
2. Define a mapper function

```
def lower(line):
  return line.lower()
```

3. Invoke the transformation: map

```
lower_text_RDD = text_RDD.map(lower)
```

Example transformation: `map(func)`



› Output size: unchanged

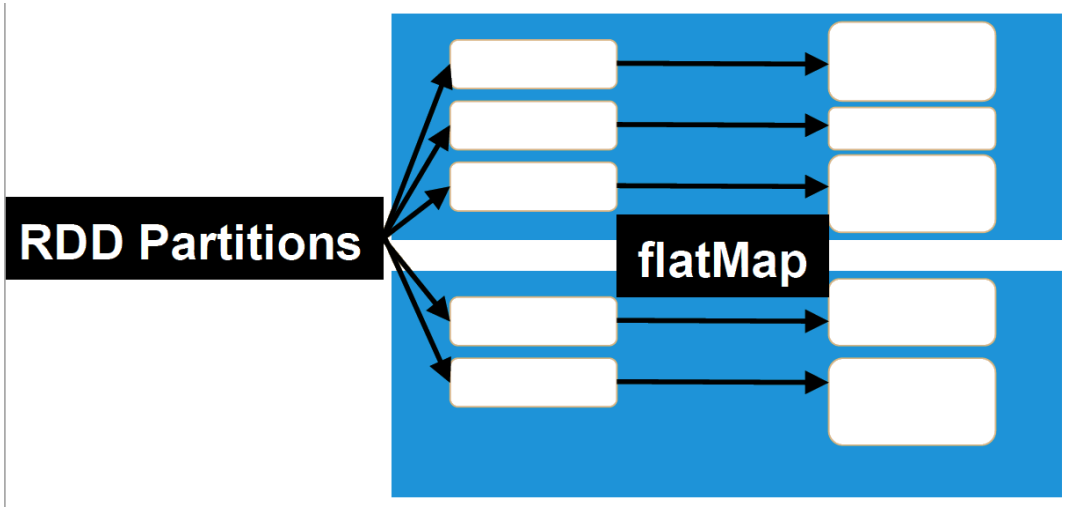
Example transformation: `flatMap(func)`

1. Define a mapper function

```
def split_words(line):
  return line.split()
```

2. Invoke the transformation: flatMap

```
words_RDD = text_RDD.flatMap(split_words)
```



Comparison: testfile1 after split_words with...

› `Map`

```
[[u'A',
 u'long',
 u'time',
 u'ago',
 u'in',
 u'a',
 u'galaxy',
 u'far',
 u'far',
 u'away']]
```

› `FlatMap`

```
[u'A',
 u'long',
 u'time',
 u'ago',
 u'in',
 u'a',
 u'galaxy',
 u'far',
 u'far',
 u'away']
```

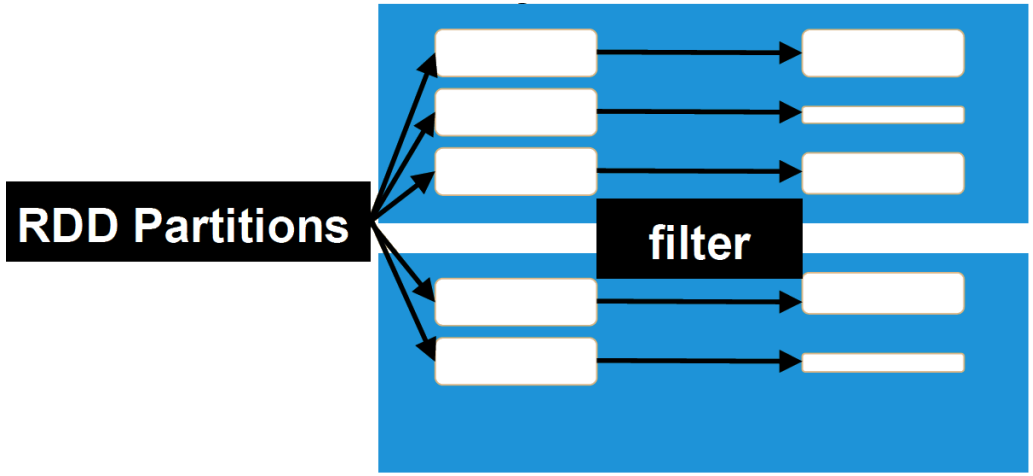
Example transformation: `filter(func)`

1. Define a function to filter out all words which start with an "a"

```
def starts_with_a(word):
    return word.lower().startswith("a")

words_RDD.filter(starts_with_a).collect()

Out[]:[u'A', u'ago', u'a', u'away']
```



filter() can result in unevenly distributed partitions...

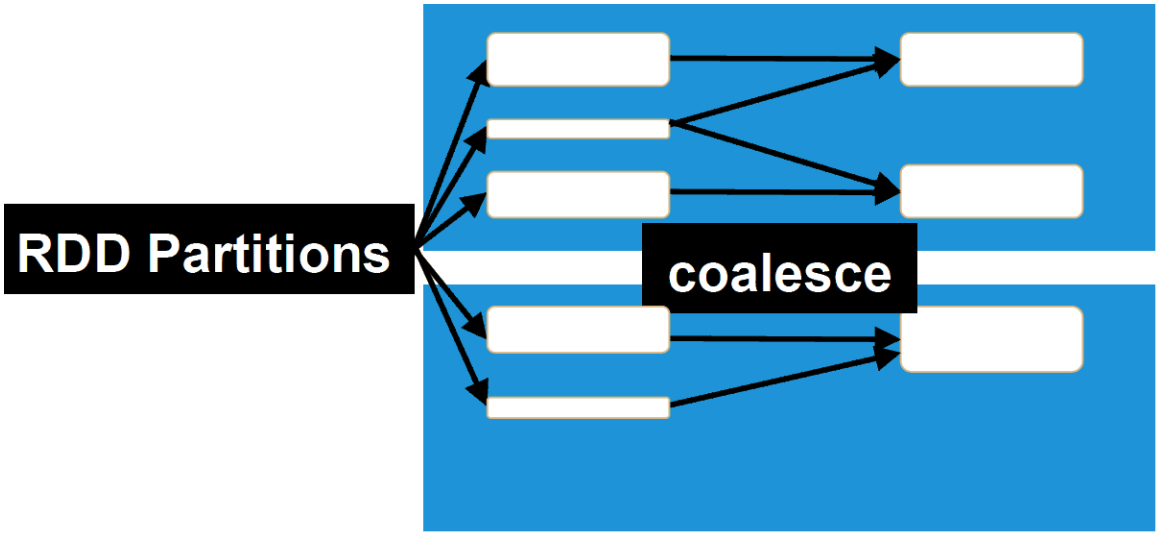
Example transformation: `coalesce(func)`

1. Create a test array with numbers 1-10 in 4 partitions

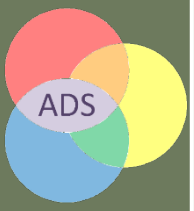
```
sc.parallelize(range(10),4).glom().collect()
Out[]:[[0,1],[2,3],[4,5],[6,7,8,9]]
```

2. Coalesce returns a new RDD that is reduced into 2 partitions

```
sc.parallelize(range(10),4).coalesce(2).glom().collect()
Out[]:[[0,1,2,3],[4,5,6,7,8,9]]
```



Spark Demo



Initial Wordcount code in Python (for PySpark)

Bash

```
def split_words(line):
        return line.split()
def create_pair(word):
        return (word,1)
def sum_counts(a,b):
        return a + b

from pyspark.sql import SparkSession
spark = SparkSession.builder.appName("appName").getOrCreate()
sc = spark.sparkContext

text_RDD = sc.textFile("file:///home/marco/testfile")
pairs_RDD = text_RDD.flatMap(split_words).map(create_pair)
wordcounts_RDD = pairs_RDD.reduceByKey(sum_counts)

wordcounts_RDD.saveAsTextFile("file:///home/marco/output");
```


```
marco@ads:~$ more testfile
A long long long time ago,
in a galaxy far far away...
marco@ads:~$
```

Input > Output

```
marco@ads:~$ ll output/
total 28
drwxrwxr-x  2 marco marco 4096 Sep 19 21:23 ./
drwxr-xr-x 12 marco marco 4096 Sep 19 21:23 ../
-rw-r--r--  1 marco marco   60 Sep 19 21:23 part-00000
-rw-r--r--  1 marco marco   12 Sep 19 21:23 .part-00000.crc
-rw-r--r--  1 marco marco   53 Sep 19 21:23 part-00001
-rw-r--r--  1 marco marco   12 Sep 19 21:23 .part-00001.crc
-rw-r--r--  1 marco marco    0 Sep 19 21:23 _SUCCESS
-rw-r--r--  1 marco marco    8 Sep 19 21:23 ._SUCCESS.crc
marco@ads:~$ more output/part-00000
(u'A', 1)
(u'far', 2)
(u'galaxy', 1)
(u'a', 1)
(u'long', 3)
marco@ads:~$ more output/part-00001
(u'ago,', 1)
(u'away...', 1)
(u'in', 1)
(u'time', 1)
marco@ads:~$
```

Updated Wordcount code in Python (for PySpark)

Bash

```
def split_words(line):
        return line.split()
def create_pair(word):
        return (word,1)
def sum_counts(a,b):
        return a + b

from pyspark.sql import SparkSession
spark = SparkSession.builder.appName("appName").getOrCreate()
sc = spark.sparkContext

text_RDD = sc.textFile("file:///home/marco/testfile")
pairs_RDD = text_RDD.flatMap(split_words).map(create_pair)
wordcounts_RDD = pairs_RDD.reduceByKey(sum_counts, numPartitions=1)

wordcounts_RDD.saveAsTextFile("file:///home/marco/output");
```

Commands used

› vi wordcount_mapreduce.py

› chmod +x wordcount_mapreduce.py

› spark-submit wordcount_mapreduce.py

› more output/part-00000

› rm -r ./output

Optionally: interactive tutorial at:
http://learnpython.org

PySpark reference:
https://spark.apache.org/docs/latest/api/python/pyspark.html

Final Wordcount output

```
marco@ads:~$ ll output/
total 20
drwxrwxr-x  2 marco marco 4096 Sep 19 21:28 ./
drwxr-xr-x 12 marco marco 4096 Sep 19 21:28 ../
-rw-r--r--  1 marco marco  113 Sep 19 21:28 part-00000
-rw-r--r--  1 marco marco   12 Sep 19 21:28 .part-00000.crc
-rw-r--r--  1 marco marco    0 Sep 19 21:28 _SUCCESS
-rw-r--r--  1 marco marco    8 Sep 19 21:28 ._SUCCESS.crc
marco@ads:~$ more output/part-00000
(u'A', 1)
(u'ago,', 1)
(u'far', 2)
(u'in', 1)
(u'long', 3)
(u'away...', 1)
(u'a', 1)
(u'time', 1)
(u'galaxy', 1)
marco@ads:~$
```