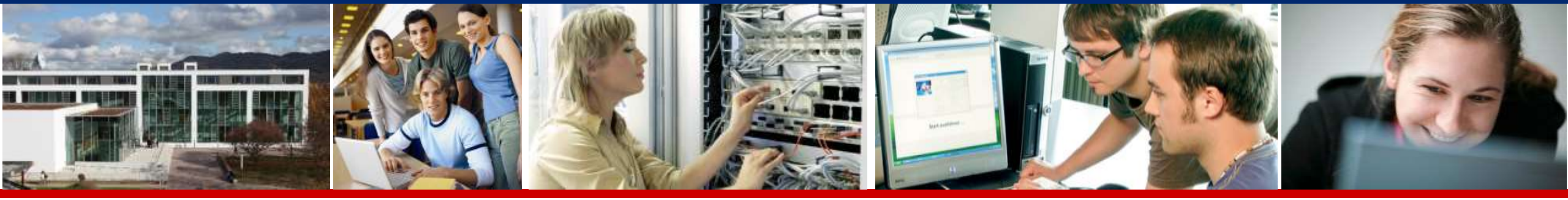


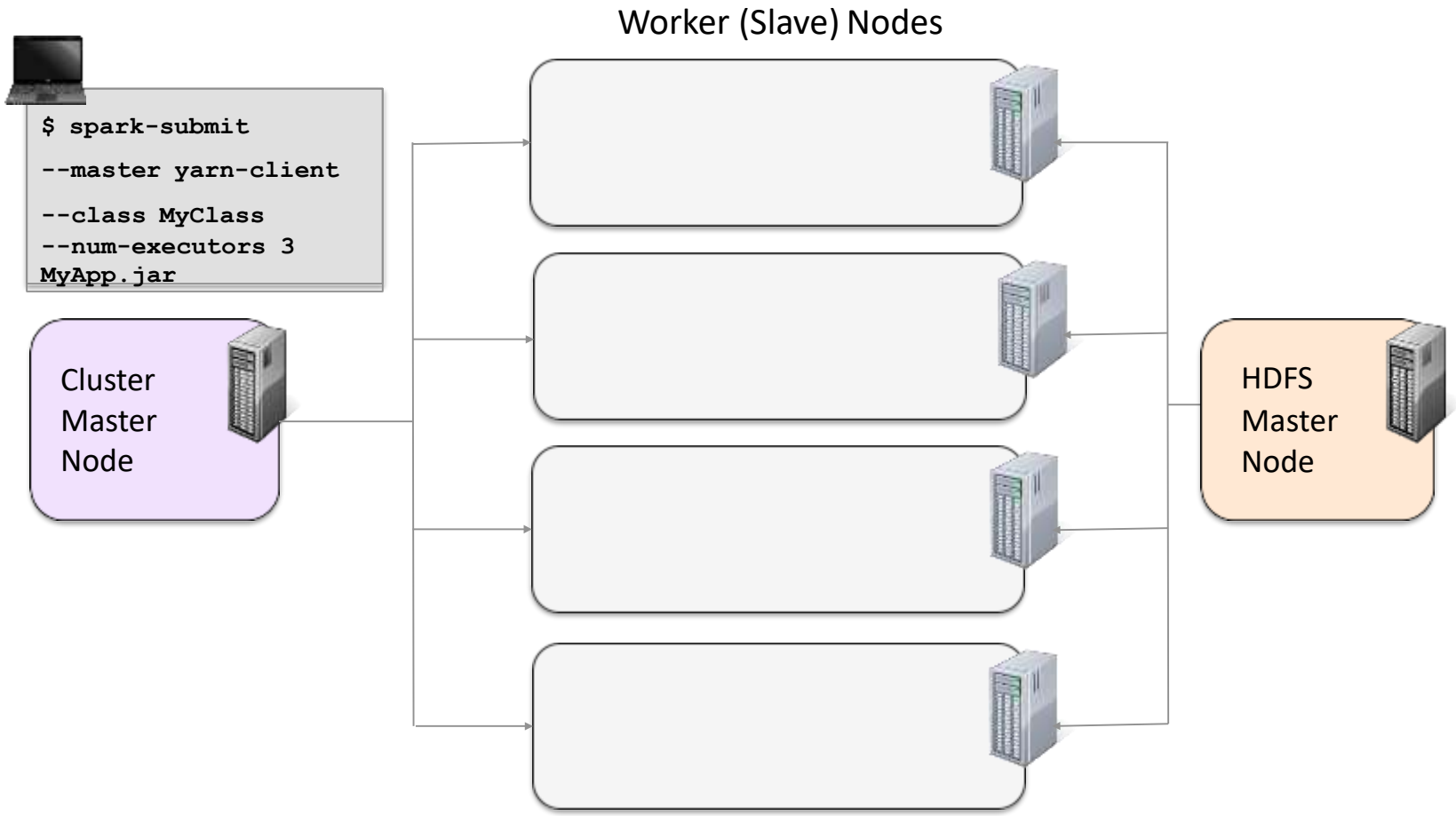
Parallel Processing in Spark

Based on Slides of Cloudera. Copyrigth Cloudera.

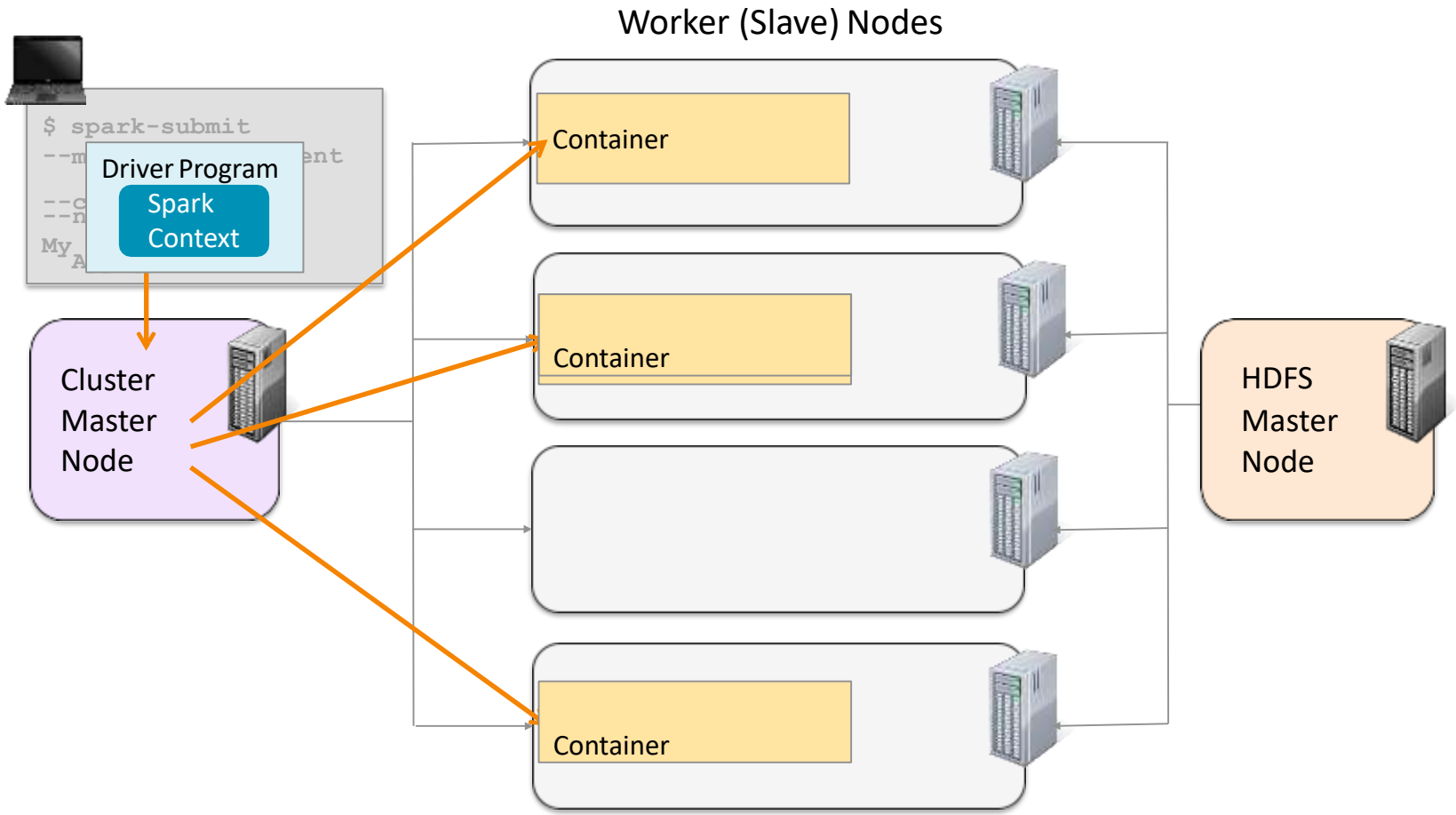
Outline

- RDD Partions
- Partioning of File-based RDDs
- HDFS and Data Locality
- Executing Parallel Operations
- Stages and Tasks
- Conclusion

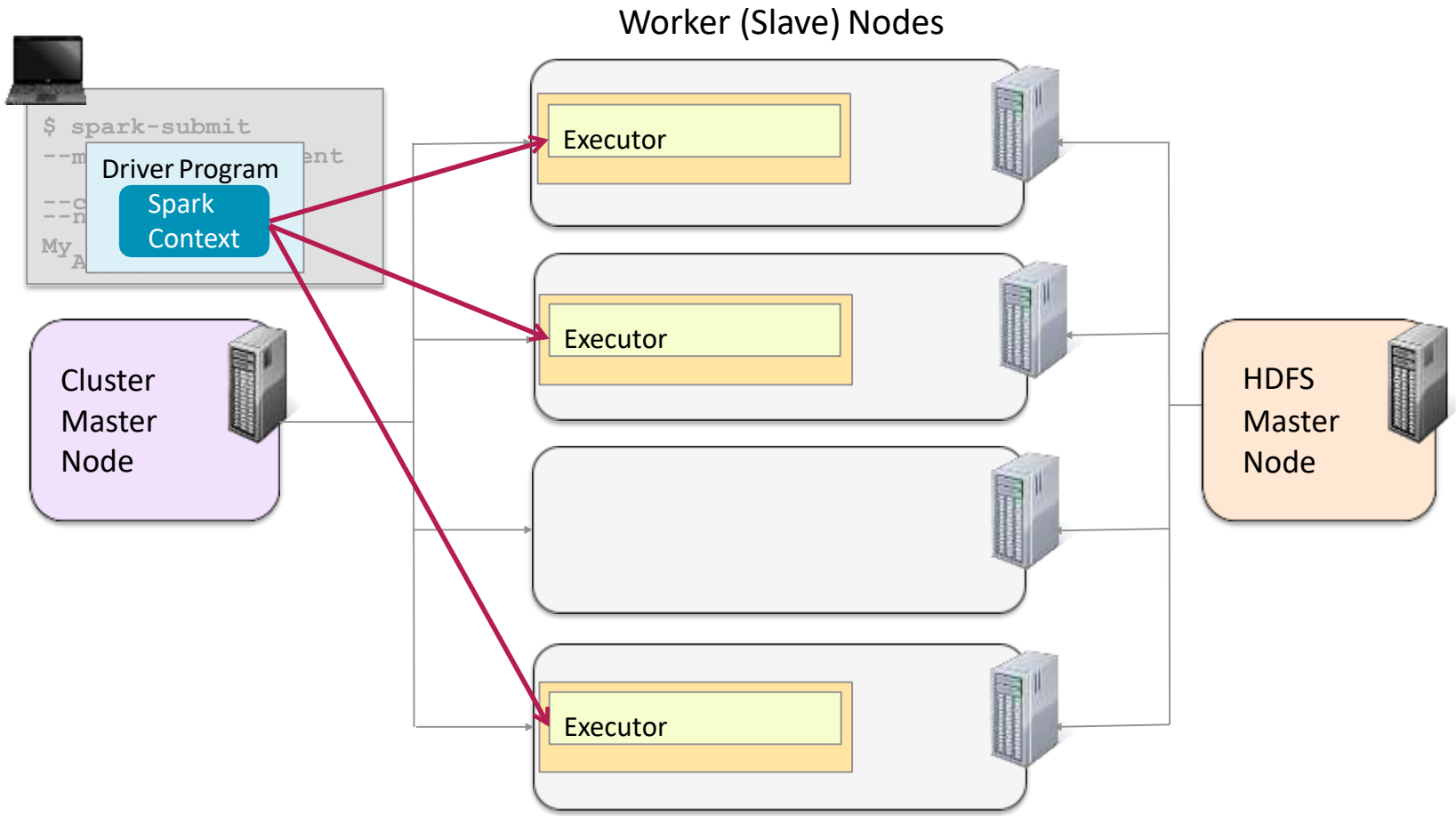
Spark Cluster Review



Spark Cluster Review



Spark Cluster Review



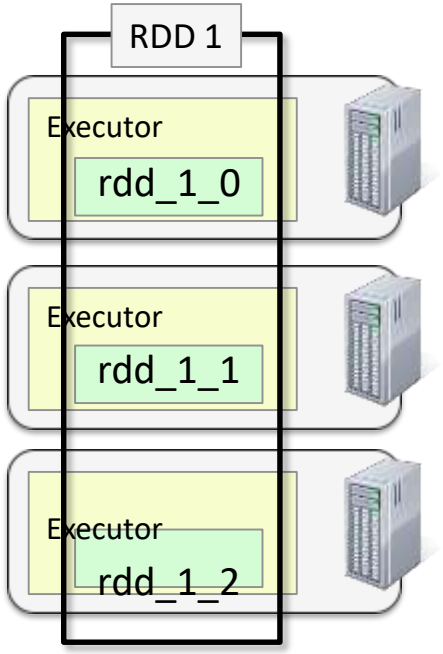
RDDs on a Cluster

Resilient Distributed Datasets

Data is partioned across worker nodes

Partioning is done automatically by Spark

Optionally, you can control how many Partions are created



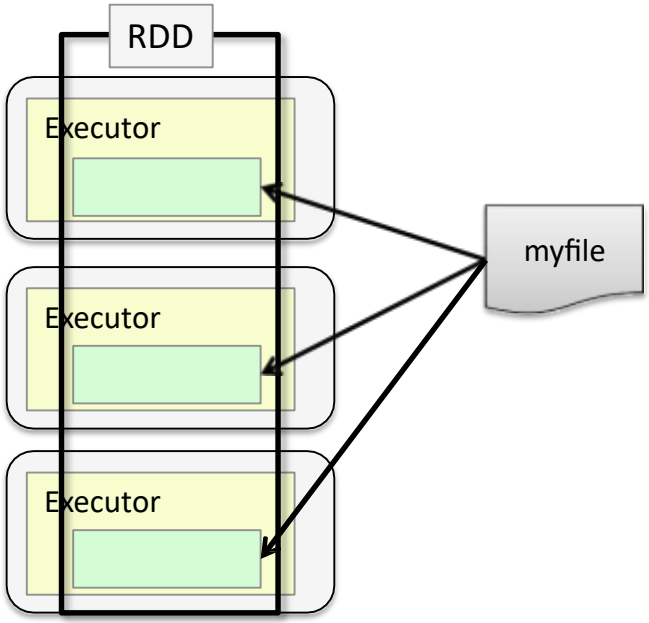
Outline

- RDD Partions
- Partioning of File-based RDDs
- HDFS and Data Locality
- Executing Parallel Operations
- Stages and Tasks
- Conclusion

File Partioning: Single Files

- Partions based on size
- You can optionally specify a minimum number of Partions
- textFile(file, minPartitions)
- Default is 2
- More Partions = more parallelization

```
sc.textFile("myfile",3)
```



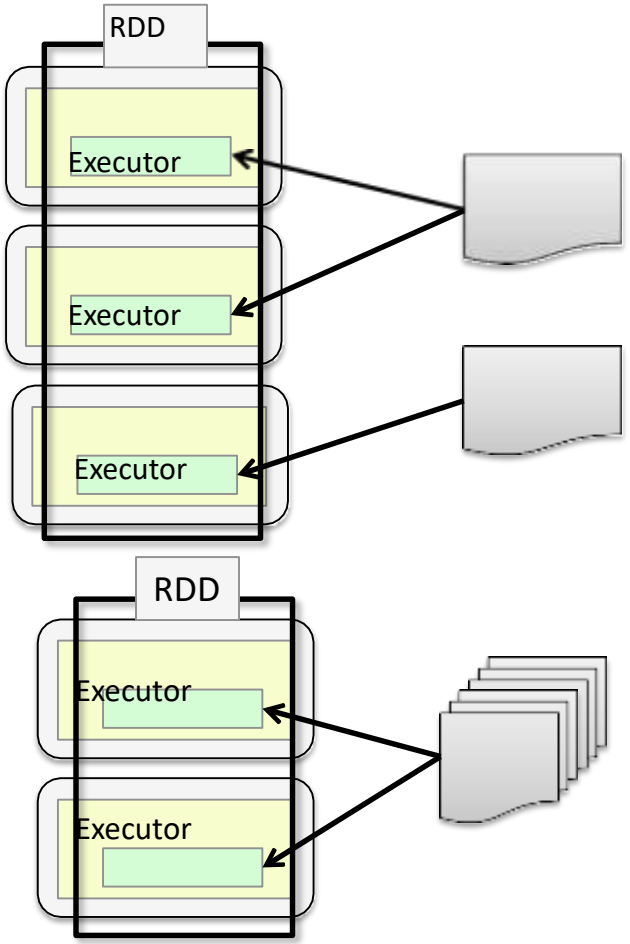
File Partioning: Multiple Files

sc.textFile("mydir/*")

- Each file becomes (at least) one Partion
- File-based operations can be done per-Partion, for example parsing XML

sc.wholeTextFiles("mydir")

- For many small files
- Creates a key-value PairRDD
- key = file name
- value = file contents



Operating on Partions

Most RDD operations work on each element of an RDD

A few work on each partion

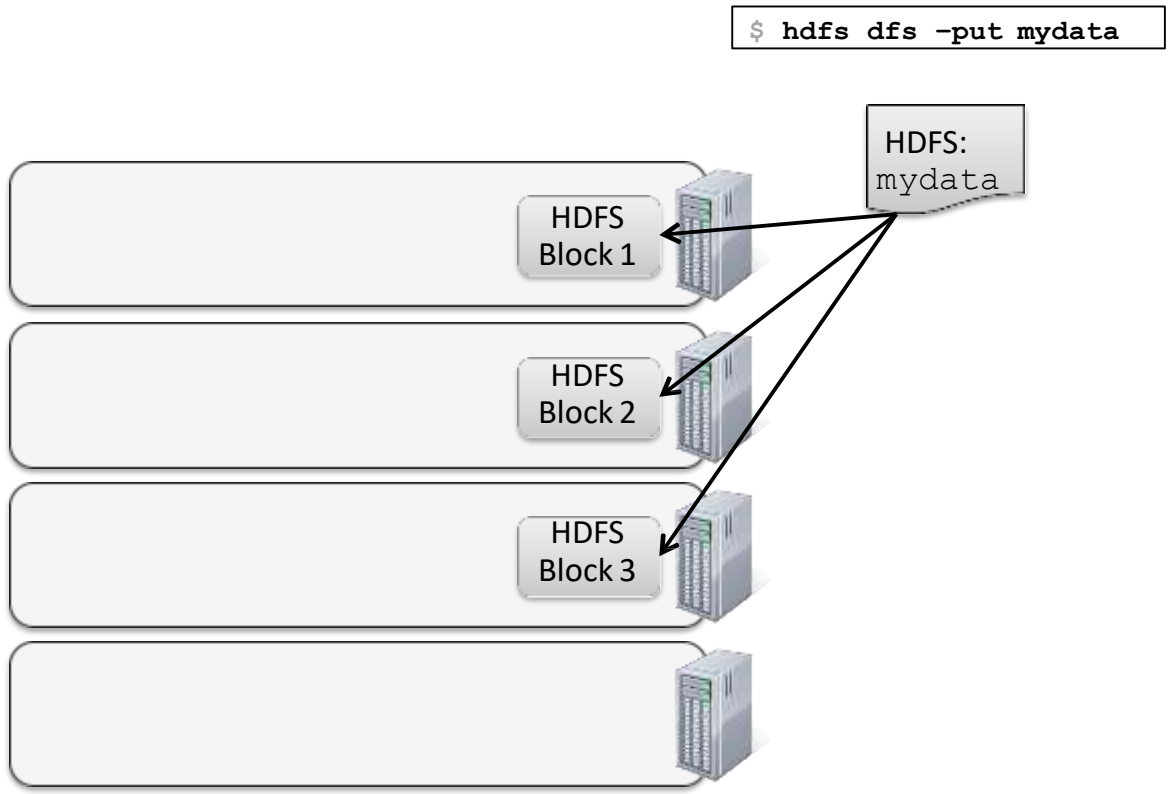
- foreachPartition – call a function for each Partion
- mapPartitions – create a new RDD by executing a function on each Partion in the current RDD
- mapPartitionsWithIndex – same as mapPartitions but includes index of the Partion Functions for partion operations take iterators

Outline

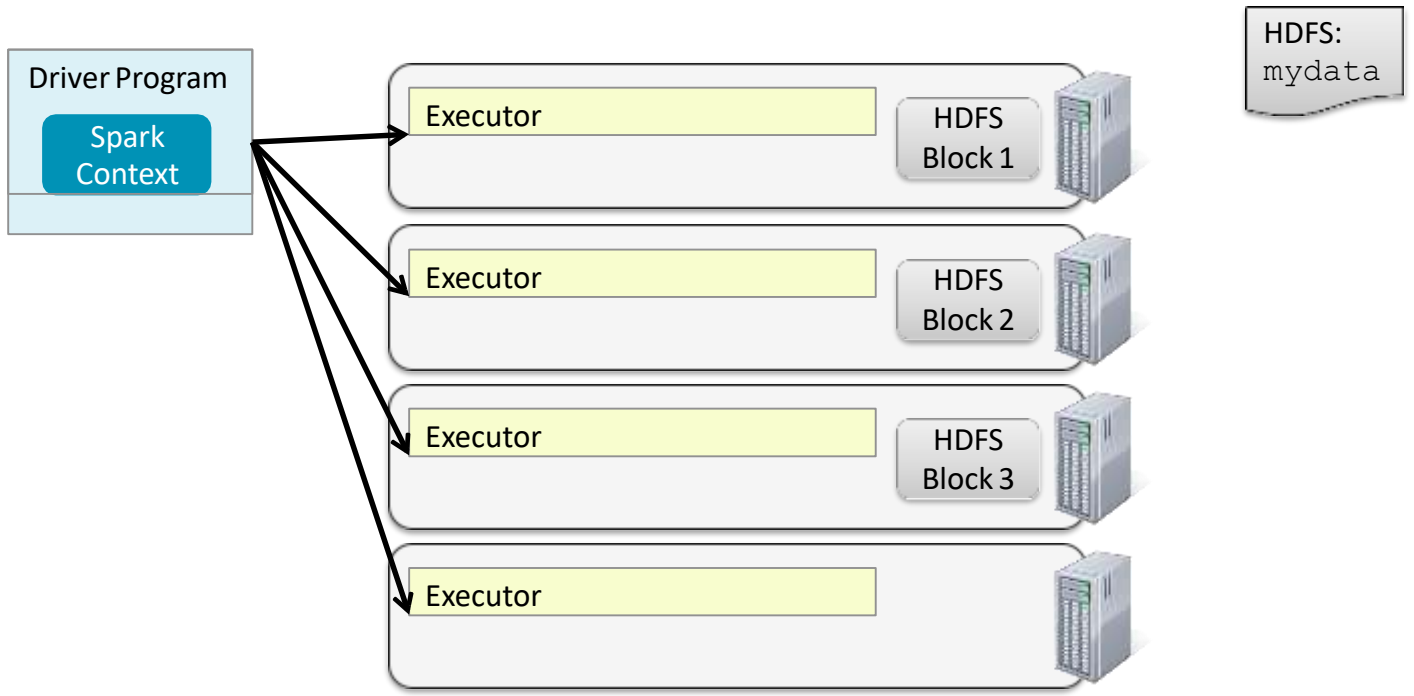
- RDD Partions
- Partioning of File-based RDDs
- HDFS and Data Locality
- Executing Parallel Operations
- Stages and Tasks
- Conclusion

HDFS and Data Locality (1)

HDFS and Data Locality (2)



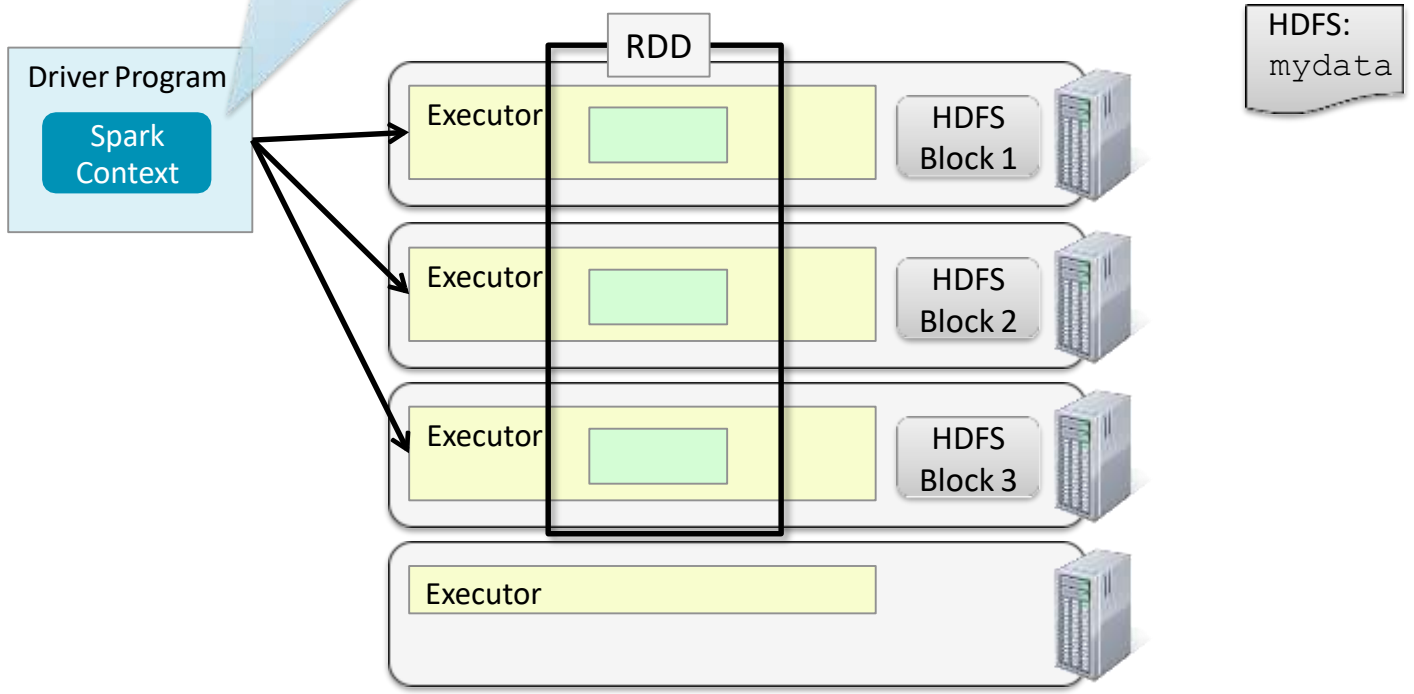
HDFS and Data Locality (3)



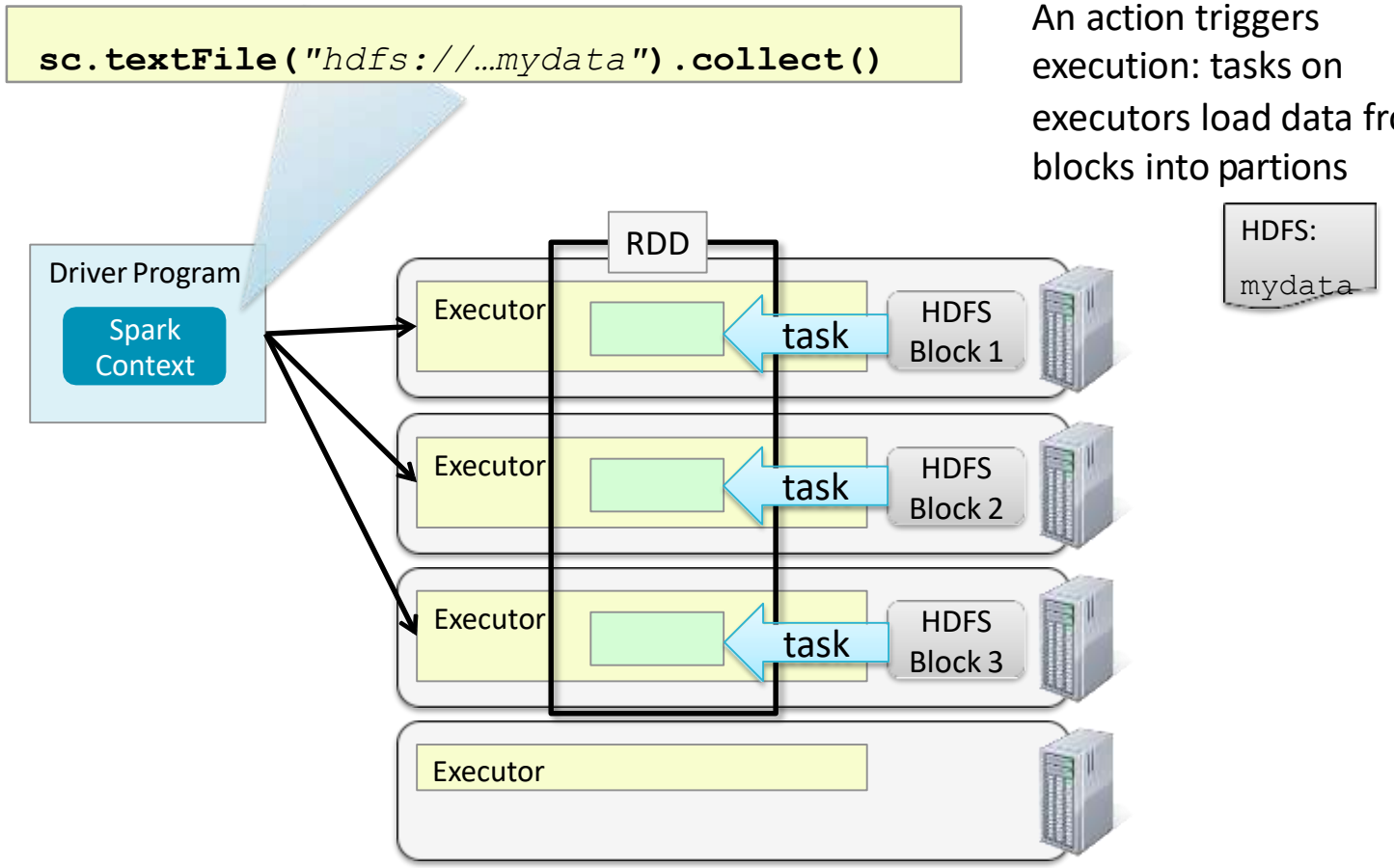
HDFS and Data Locality (4)

```
sc.textFile("hdfs://…mydata").collect()
```

By default, Spark partions file-based RDDs by block.
Each block loads into a single Partion.



HDFS and Data Locality (5)



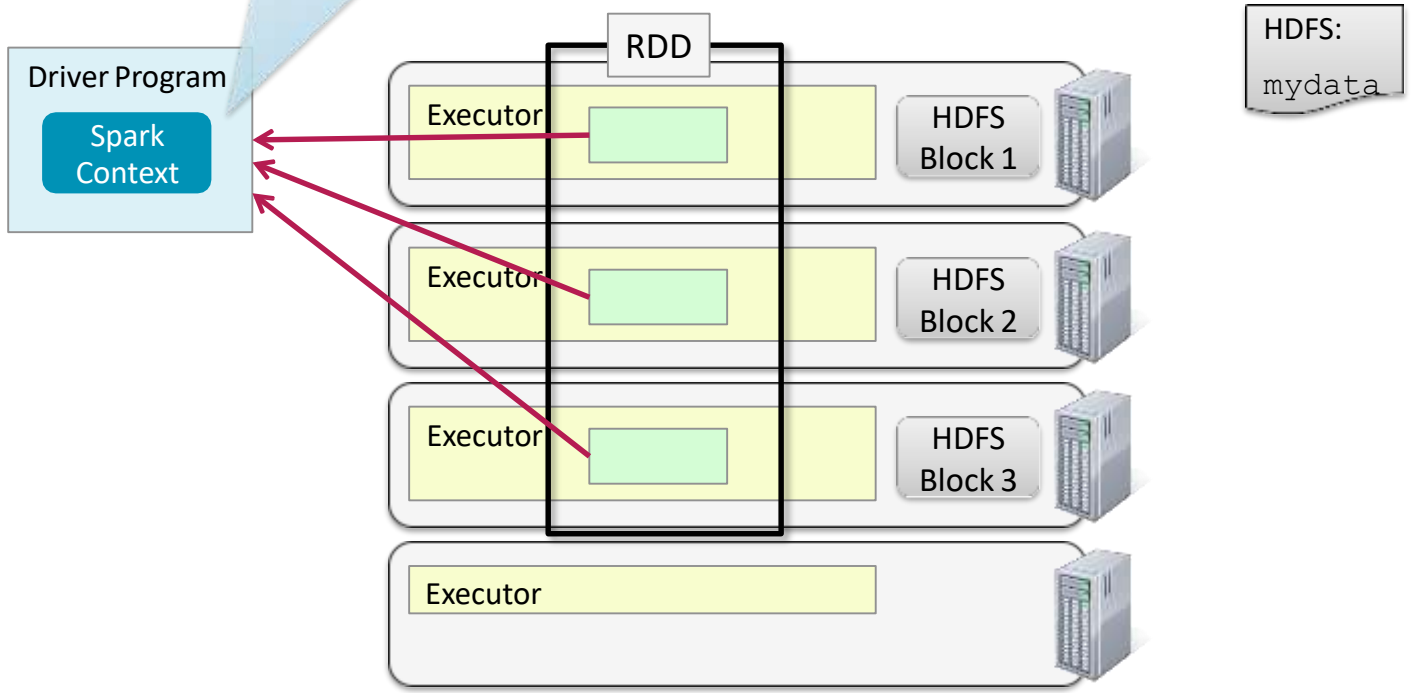
An action triggers execution: tasks on executors load data from blocks into partions

HDFS:
mydata

HDFS and Data Locality (6)

```
sc.textFile("hdfs://…mydata").collect()
```

Data is distributed across executors until an action returns a value to the driver



Outline

- RDD Partions
- Partioning of File-based RDDs
- HDFS and Data Locality
- Executing Parallel Operations
- Stages and Tasks
- Conclusion

Parallel Operations on Partions

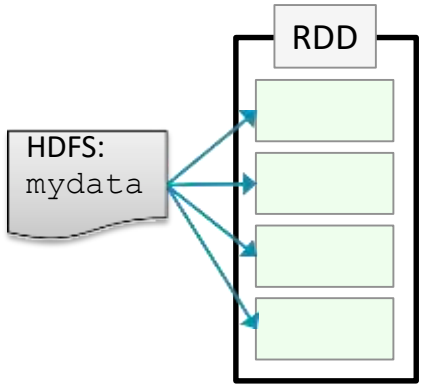
- RDD operations are executed in parallel on each partion
- When possible, tasks execute on the worker nodes where the data is in memory

- Some operations preserve Partioning e.g., map, flatMap, filter

- Some operations reparation e.g., reduce, sort, group

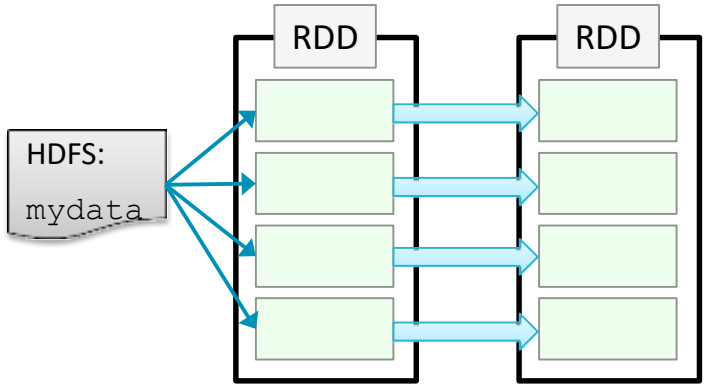
Example: Average Word Length by Letter (1)

```
> avglens = sc.textFile(file)
```



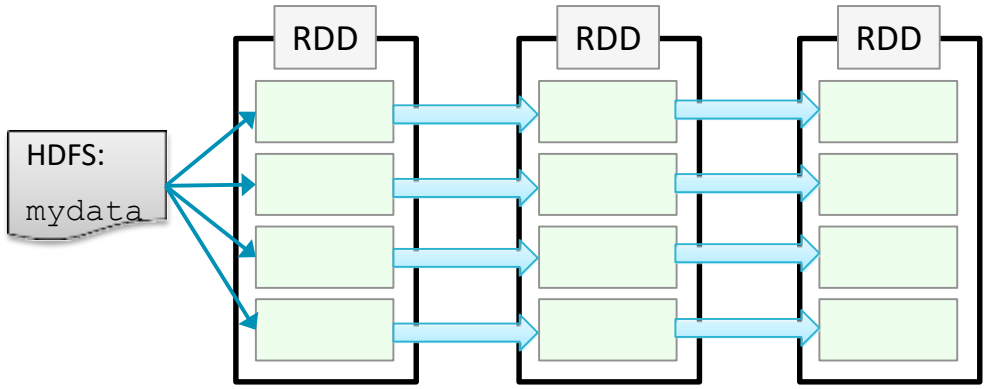
Example: Average Word Length by Letter (2)

```
> avglens = sc.textFile(file) \
  .flatMap(lambda line: line.split())
```



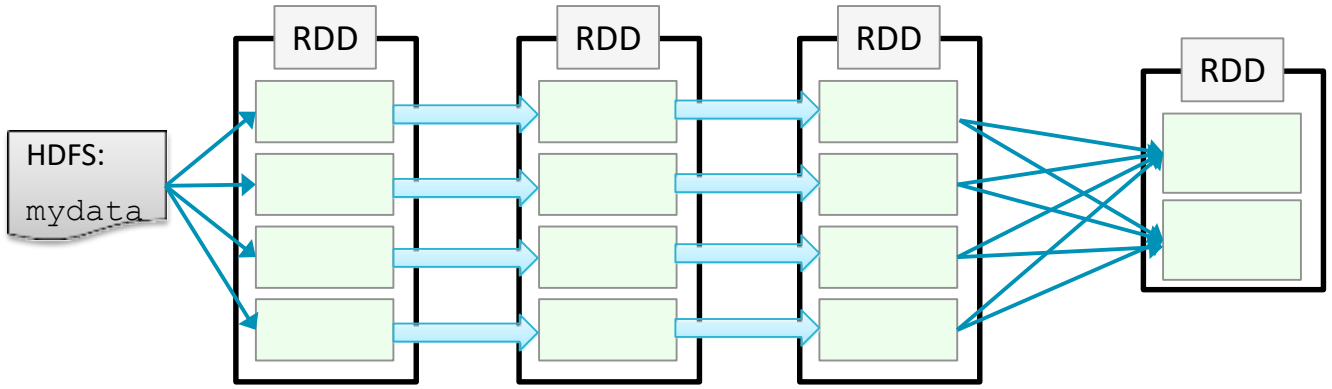
Example: Average Word Length by Letter (3)

```
> avglens = sc.textFile(file) \
  .flatMap(lambda line: line.split()) \
  .map(lambda word: (word[0],len(word)))
```



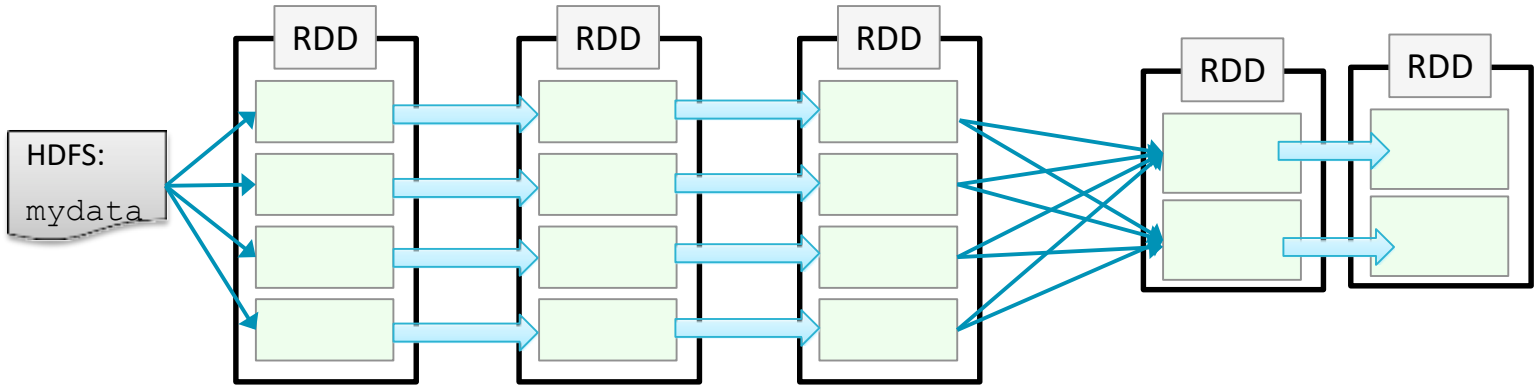
Example: Average Word Length by Letter (4)

```
> avglens = sc.textFile(file) \
  .flatMap(lambda line: line.split()) \
  .map(lambda word: (word[0],len(word))) \
  .groupByKey()
```



Example: Average Word Length by Letter (5)

```
> avglens = sc.textFile(file) \
  .flatMap(lambda line: line.split()) \
  .map(lambda word: (word[0],len(word))) \
  .groupByKey() \
  .map(lambda (k, values): \
      (k, sum(values)/len(values)))
```



Outline

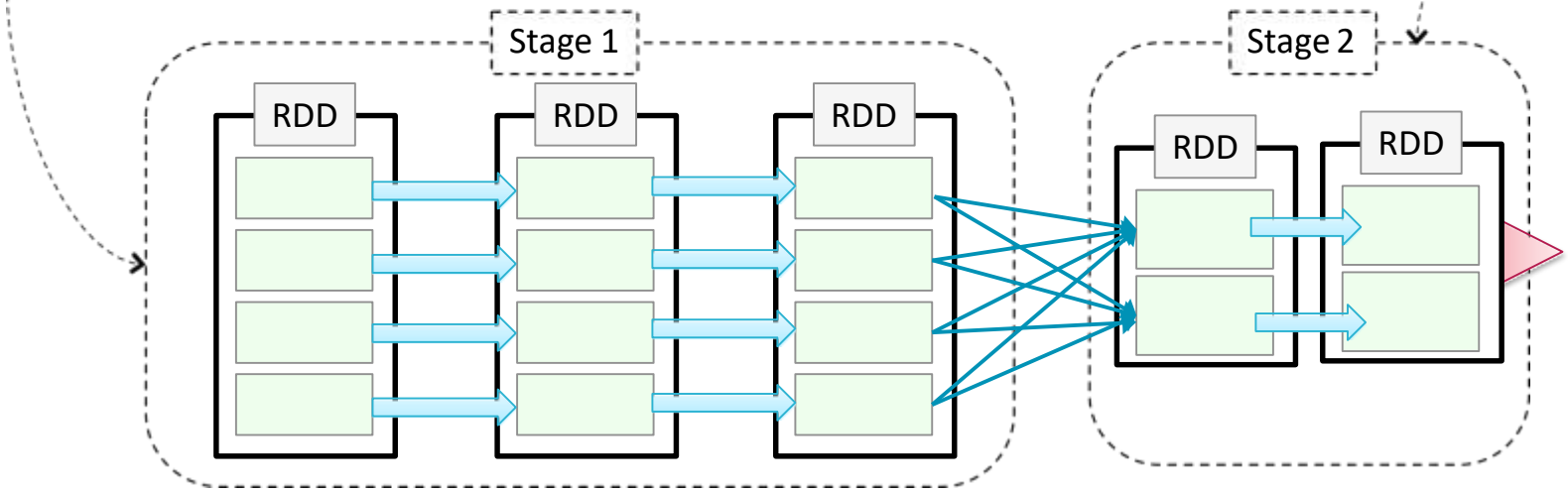
- RDD Partions
- Partioning of File-based RDDs
- HDFS and Data Locality
- Executing Parallel Operations
- Stages and Tasks
- Conclusion

Stages

- Operations that can run on the same partion are executed in stages
- Tasks within a stage are pipelined together
- Developers should be aware of stages to improve performance

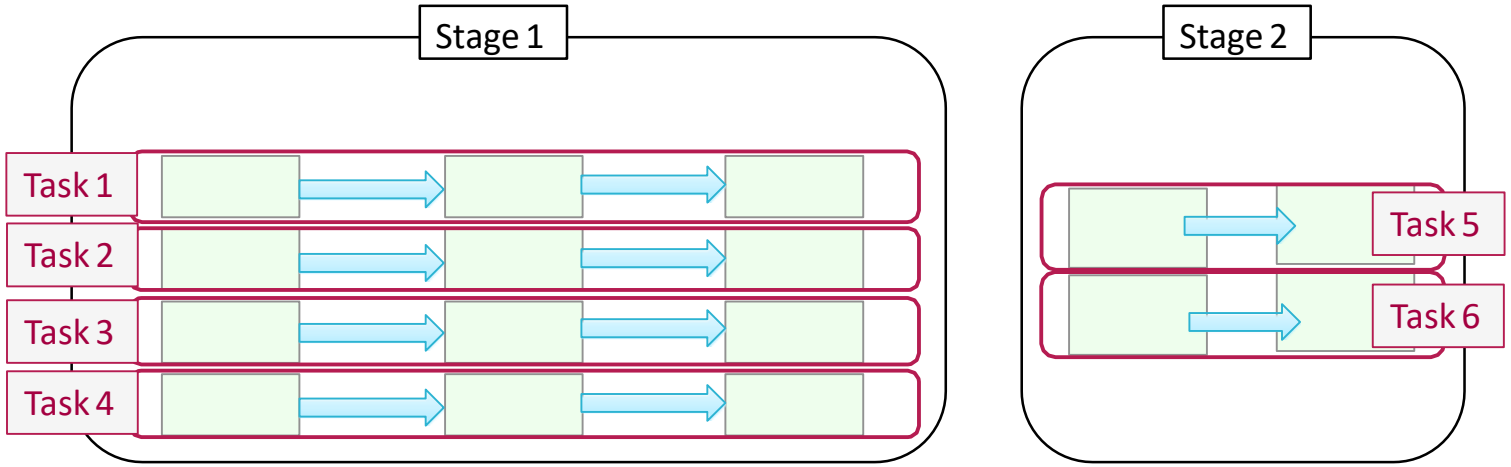
Spark Execution: Stages (1)

```
> val avglens = sc.textFile(myfile).
    flatMap(line => line.split("\\W")).
    map(word => (word(0),word.length)).
    groupByKey().
    map(pair => (pair._1, pair._2.sum/pair._2.size.toDouble))

> avglens.saveAsTextFile("avglen-output")
```



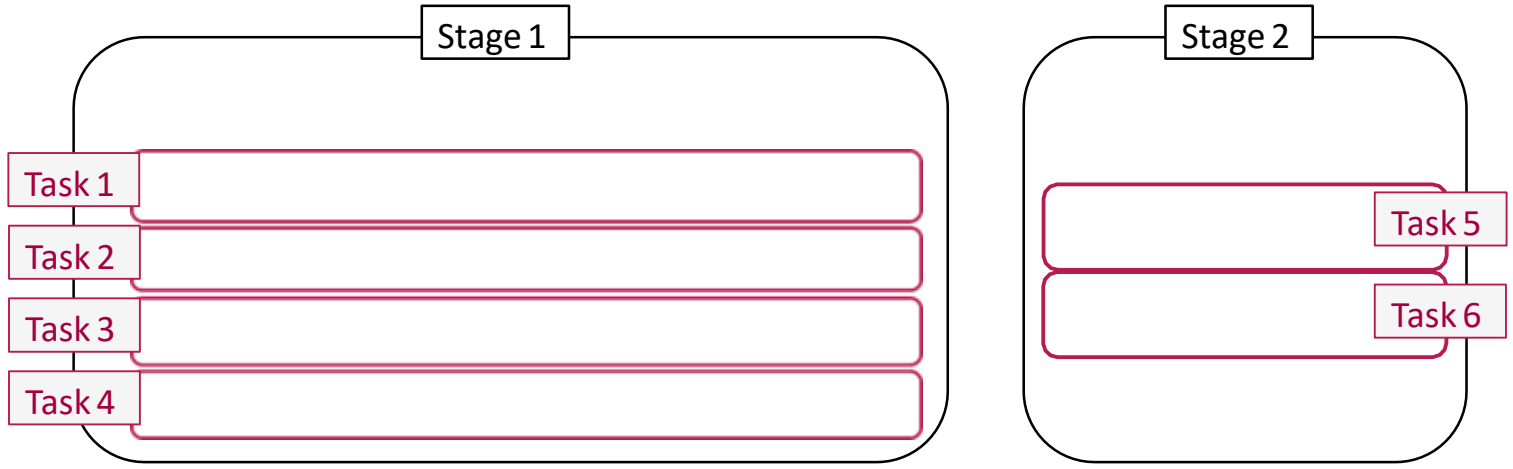
Spark Execution: Stages (2)

```
> val avglens = sc.textFile(myfile).
    flatMap(line => line.split("\\W")).
    map(word => (word(0),word.length)).
    groupByKey().
    map(pair => (pair._1, pair._2.sum/pair._2.size.toDouble))

> avglens.saveAsTextFile("avglen-output")
```



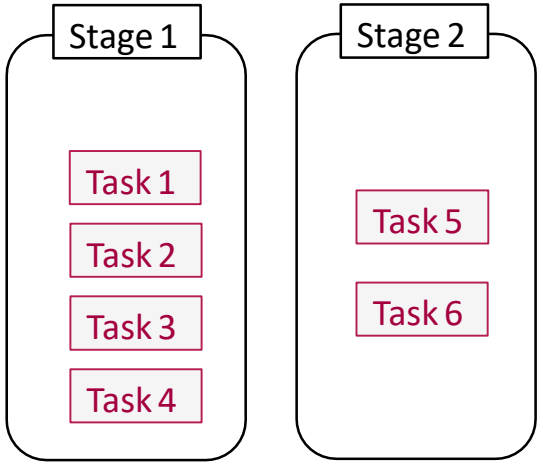
Spark Execution: Stages (3)

```
> val avglens = sc.textFile(myfile).
    flatMap(line => line.split("\\W")).
    map(word => (word(0),word.length)).
    groupByKey().
    map(pair => (pair._1, pair._2.sum/pair._2.size.toDouble))

> avglens.saveAsTextFile("avglen-output")
```



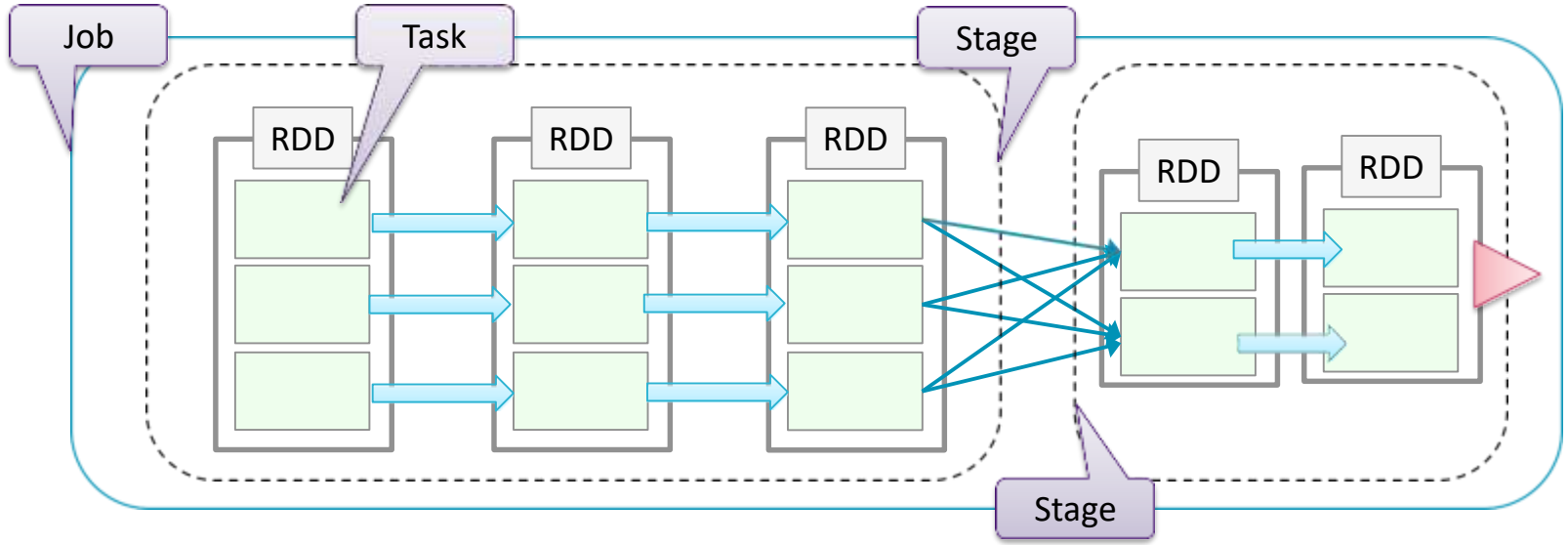
Spark Execution: Stages (4)

```
> val avglens = sc.textFile(myfile).
    flatMap(line => line.split("\\W")).
    map(word => (word(0),word.length)).
    groupByKey().
    map(pair => (pair._1, pair._2.sum/pair._2.size.toDouble))

> avglens.saveAsTextFile("avglen-output")
```



Summary of Spark Terminology

- Job – a set of tasks executed as a result of an action
- Stage – a set of tasks in a job that can be executed in parallel
- Task – an individual unit of work sent to one executor
- Application – can contain any number of jobs managed by a single driver



How Spark Calculates Stages

Spark constructs a DAG (Directed Acyclic Graph) of RDD dependencies

Narrow dependencies

- Only one child depends on the RDD
- No shuffe required between nodes
- Can be collapsed into a single stage
  e.g., map, filter, union

- **Wide (or shuffle) dependencies**
- Multiple children depend on the RDD
- Defines a new stage
  e.g., reduceByKey, join, groupByKey

Viewing the Stages using `toDebugString` (Scala)

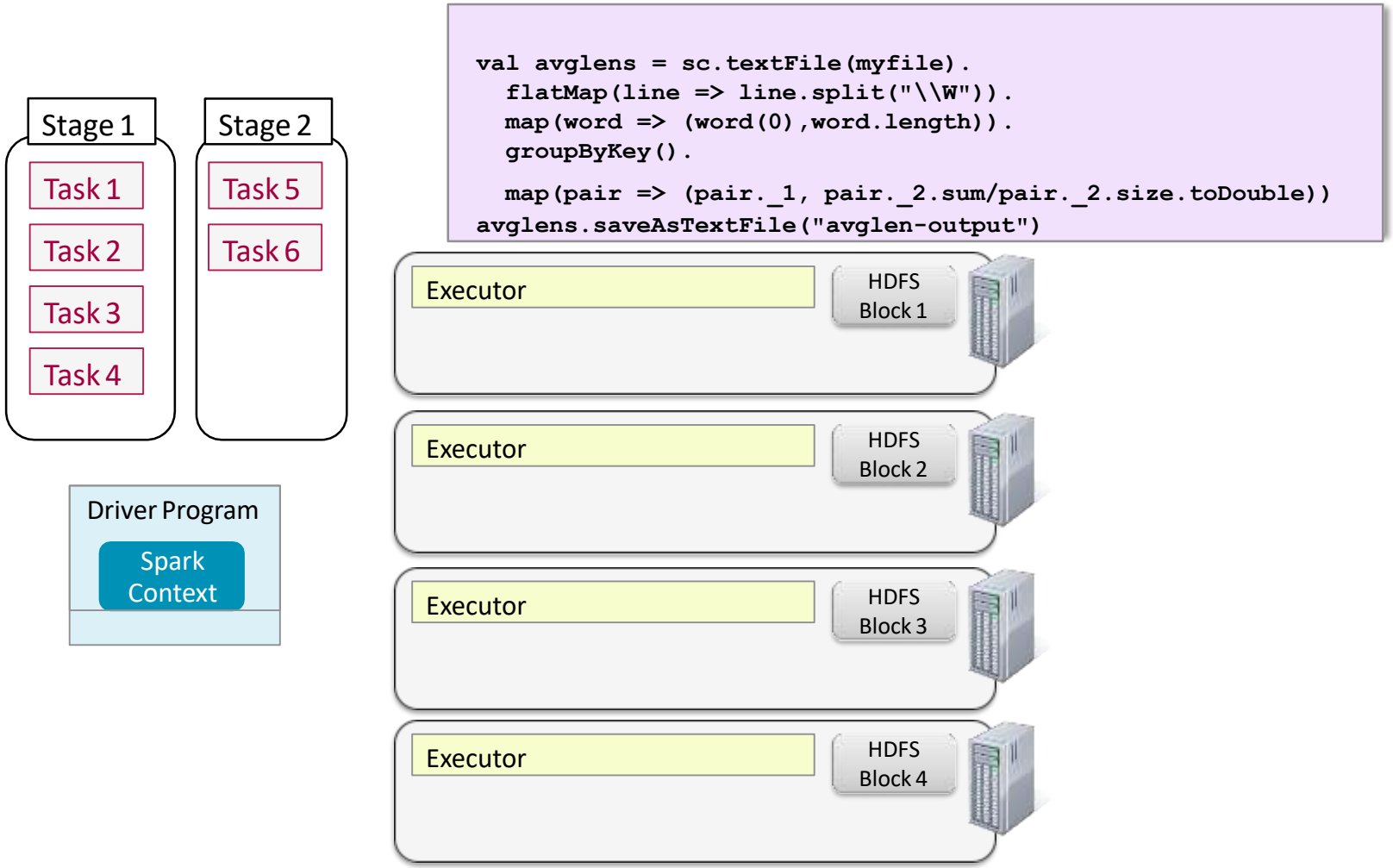
```
> val avglens = sc.textFile(myfile).
    flatMap(line => line.split("\\W")).
    map(word => (word(0),word.length)).
    groupByKey().
    map(pair => (pair._1, pair._2.sum/pair._2.size.toDouble))

> avglens.toDebugString()

(2) MappedRDD[5] at map at …
 |  ShuffledRDD[4] at groupByKey at …
 +-(4) MappedRDD[3] at map
    |  FlatMappedRDD[2] at flatMap at …
    |  myfile MappedRDD[1] at textFile at
    |  myfile HadoopRDD[0] at textFile at
```

Stage 2

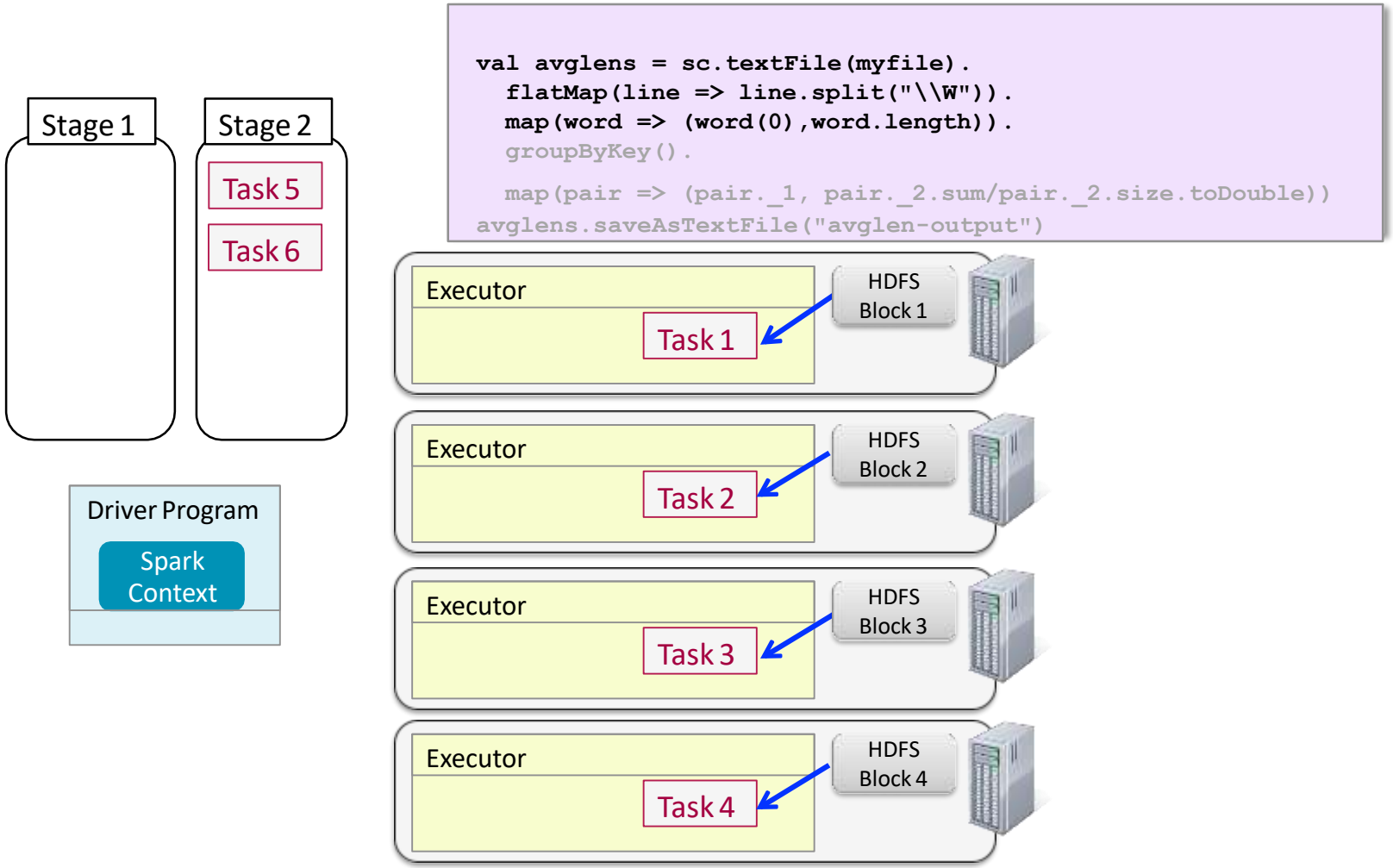
Stage 1

Indents indicate stages (shuffle boundaries)

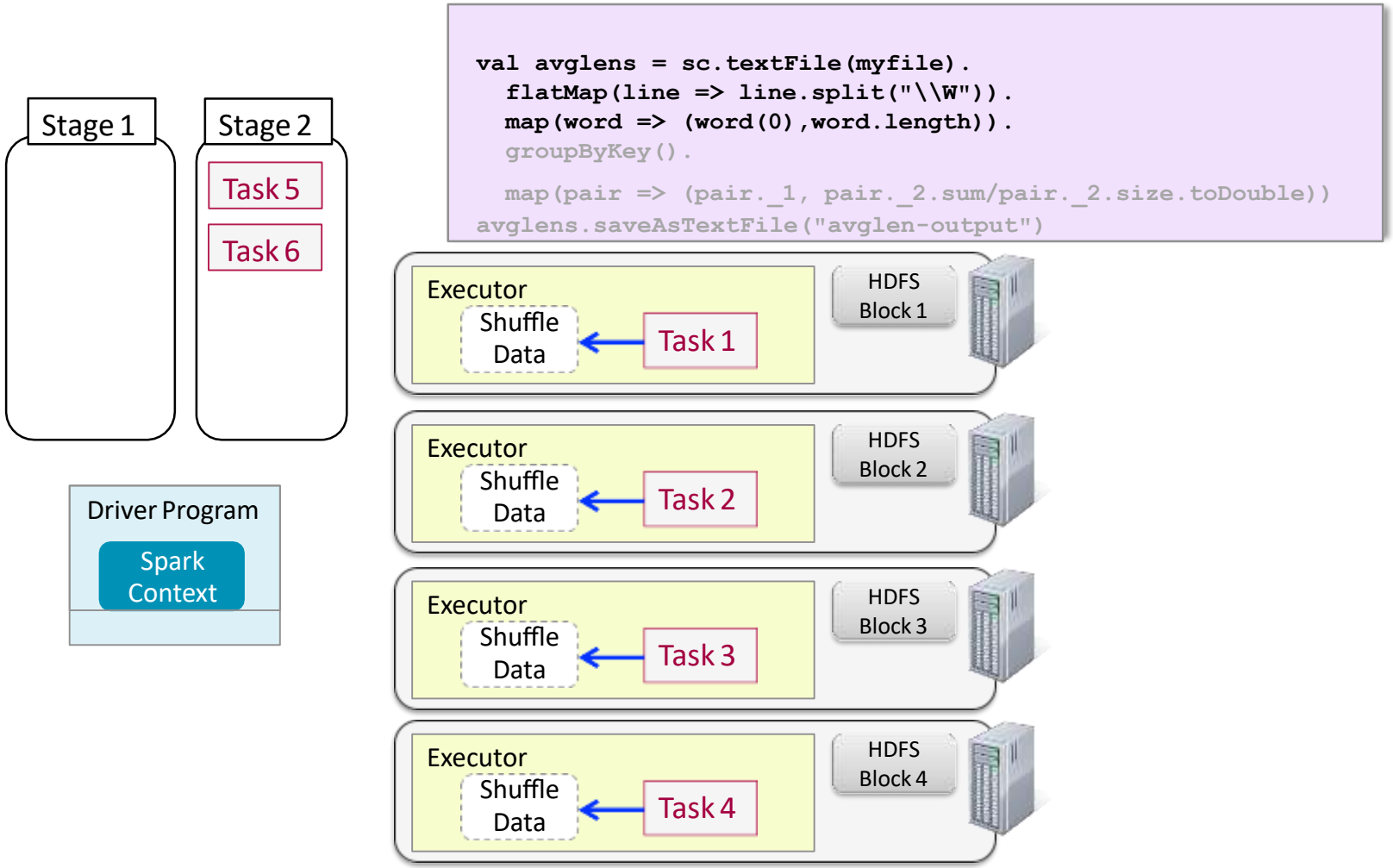
Spark Task Execution (1)



Spark Task Execution (2)

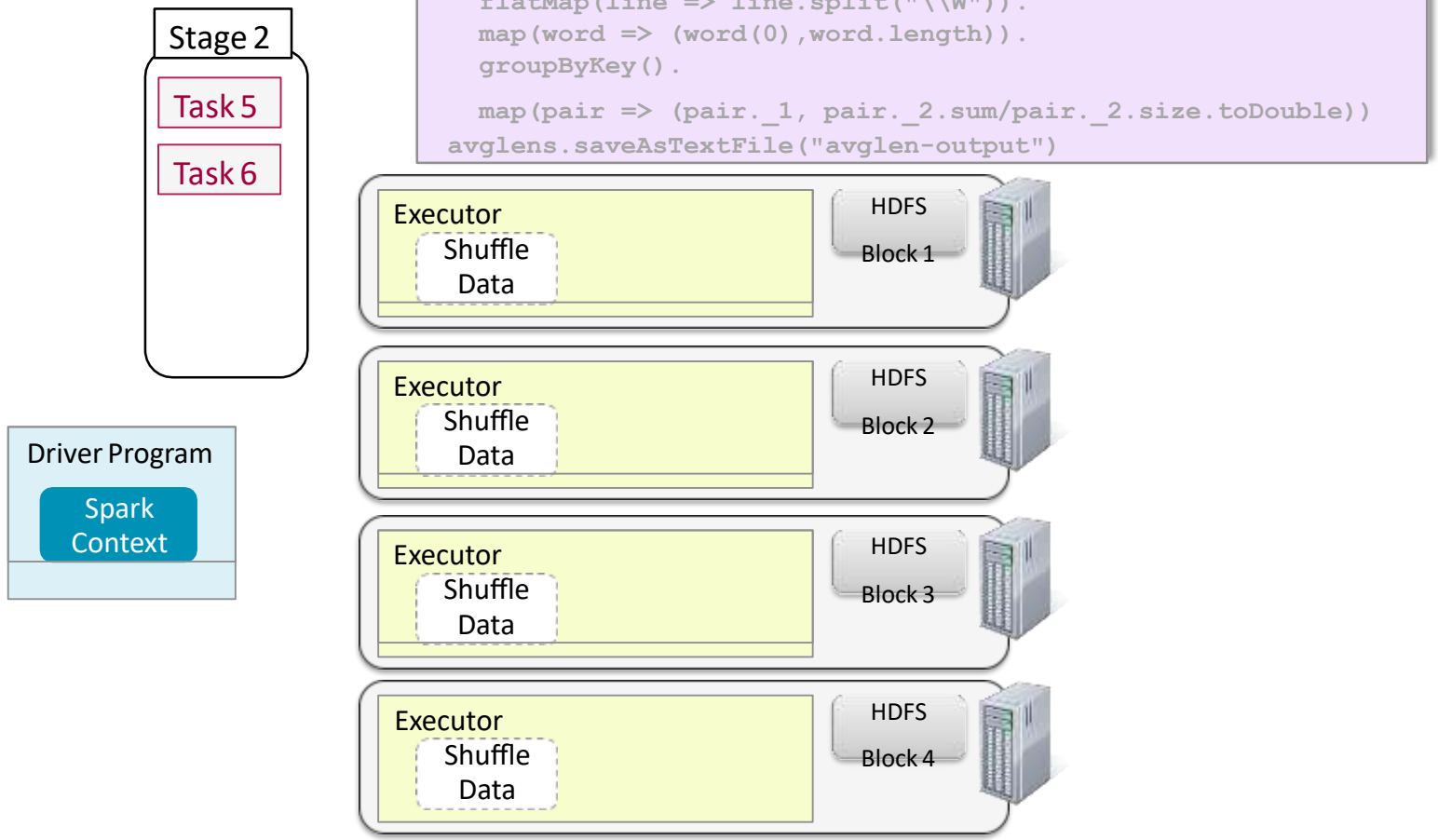


Spark Task Execution (3)



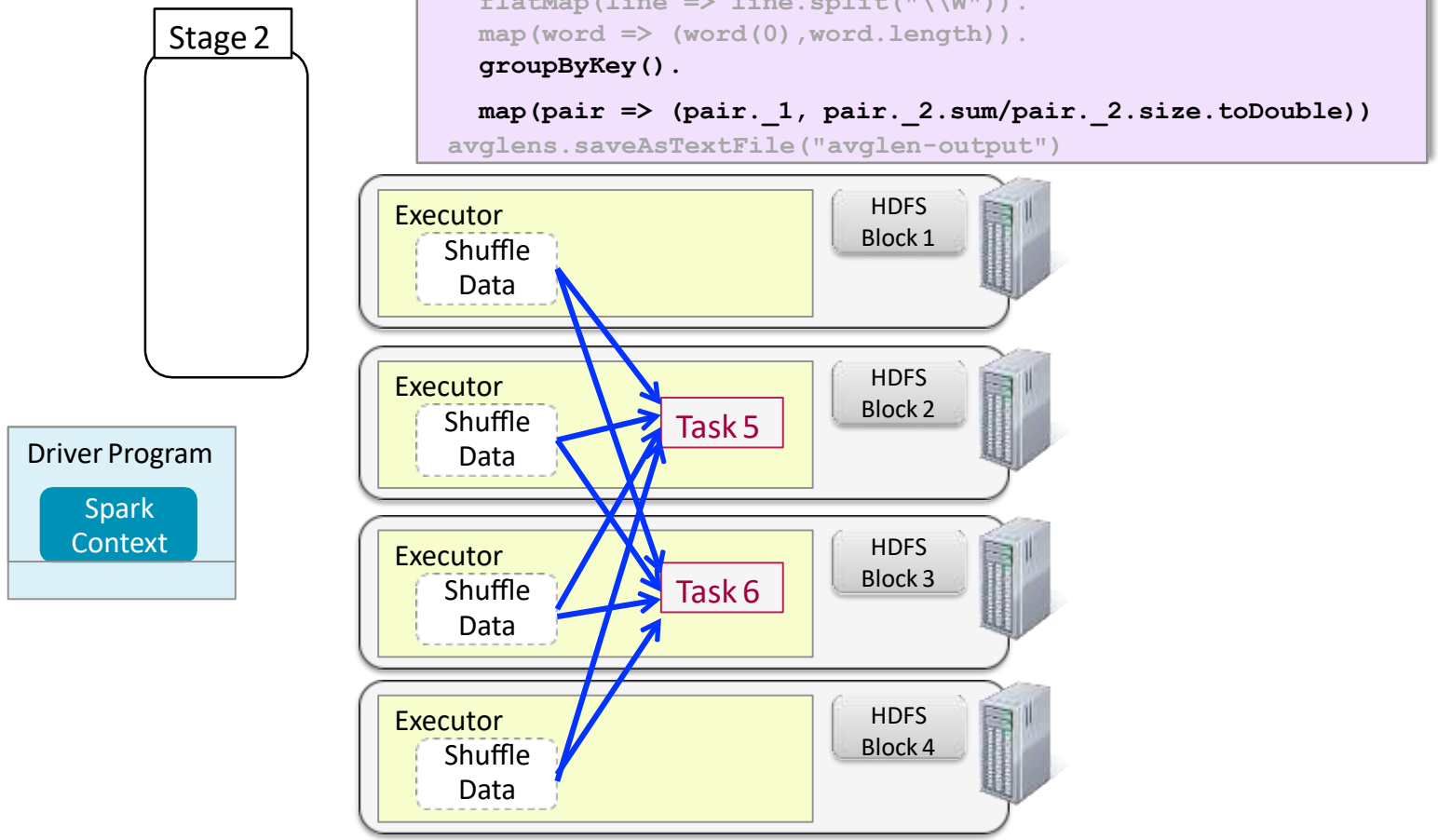
Spark Task Execution (4)

```
val avglens = sc.textFile(myfile).
  flatMap(line => line.split("\\W")).
  map(word => (word(0),word.length)).
  groupByKey().
  map(pair => (pair._1, pair._2.sum/pair._2.size.toDouble))
avglens.saveAsTextFile("avglen-output")
```



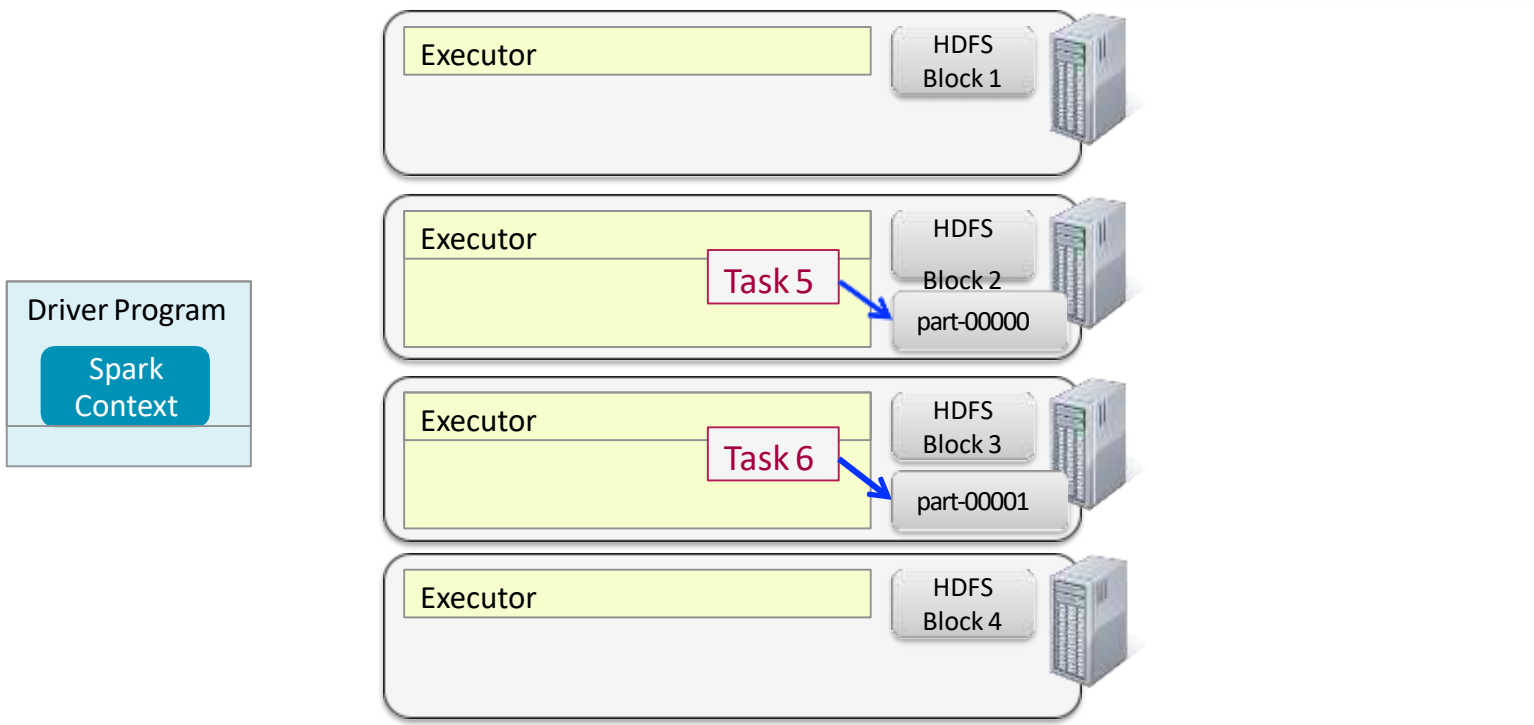
Spark Task Execution (5)

```
val avglens = sc.textFile(myfile).
  flatMap(line => line.split("\\W")).
  map(word => (word(0),word.length)).
  groupByKey().
  map(pair => (pair._1, pair._2.sum/pair._2.size.toDouble))
avglens.saveAsTextFile("avglen-output")
```



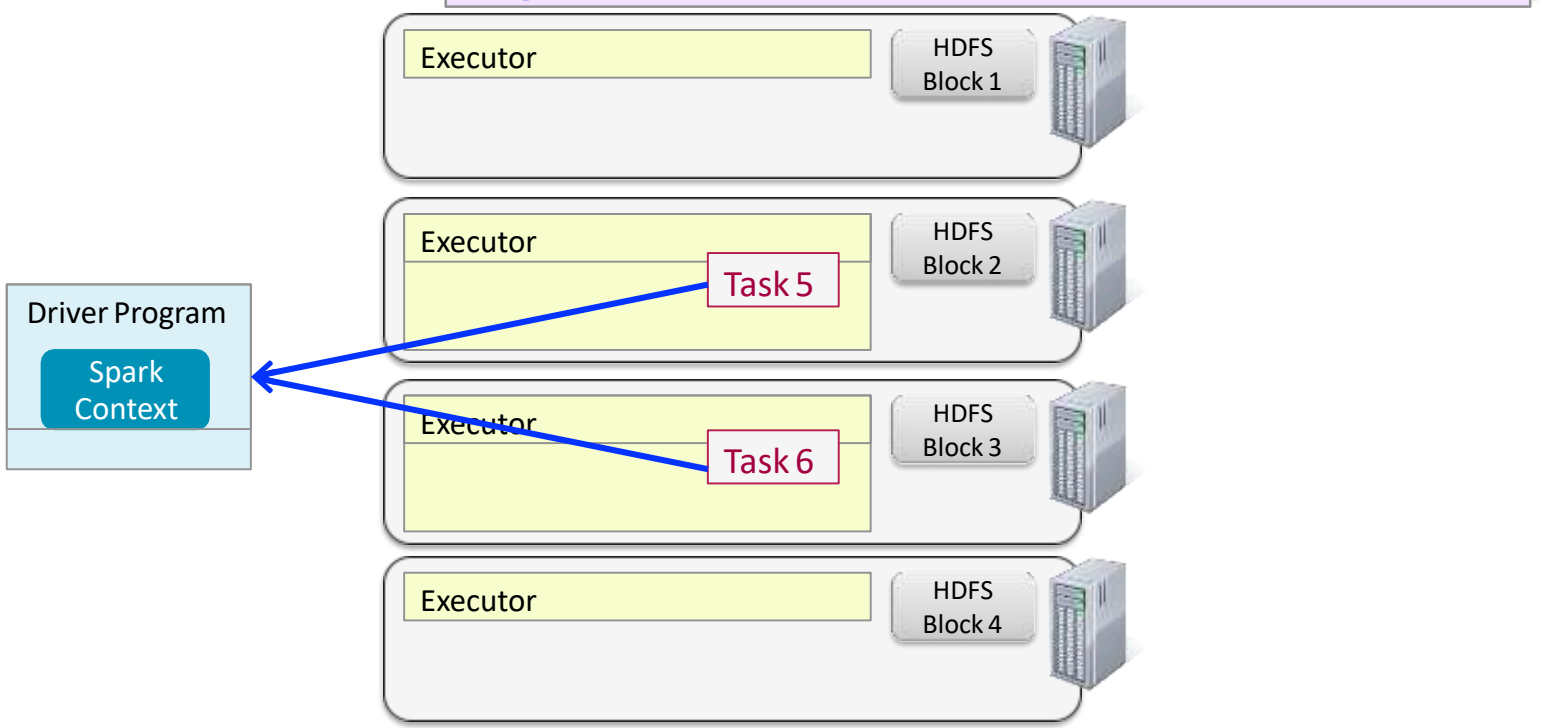
Spark Task Execution (6)

```
val avglens = sc.textFile(myfile).
  flatMap(line => line.split("\\W")).
  map(word => (word(0),word.length)).
  groupByKey().
  map(pair => (pair._1, pair._2.sum/pair._2.size.toDouble))
avglens.saveAsTextFile("avglen-output")
```



Spark Task Execution (alternate ending)

```
val avglens = sc.textFile(myfile).
  flatMap(line => line.split("\\W")).
  map(word => (word(0),word.length)).
  groupByKey().
  map(pair => (pair._1, pair._2.sum/pair._2.size.toDouble))
avglens.collect()
```



Controlling the Level of Parallelism

- "Wide" operations (e.g., reduceByKey) partion result RDDs
  - More Partions = more parallel tasks
  - Cluster will be under-utilized if there are too few Partions

- You can control how many Partions
  - Configure with the spark.default.parallelism property
    spark.default.parallelism 10
  - Optional numPartitions parameter in function call

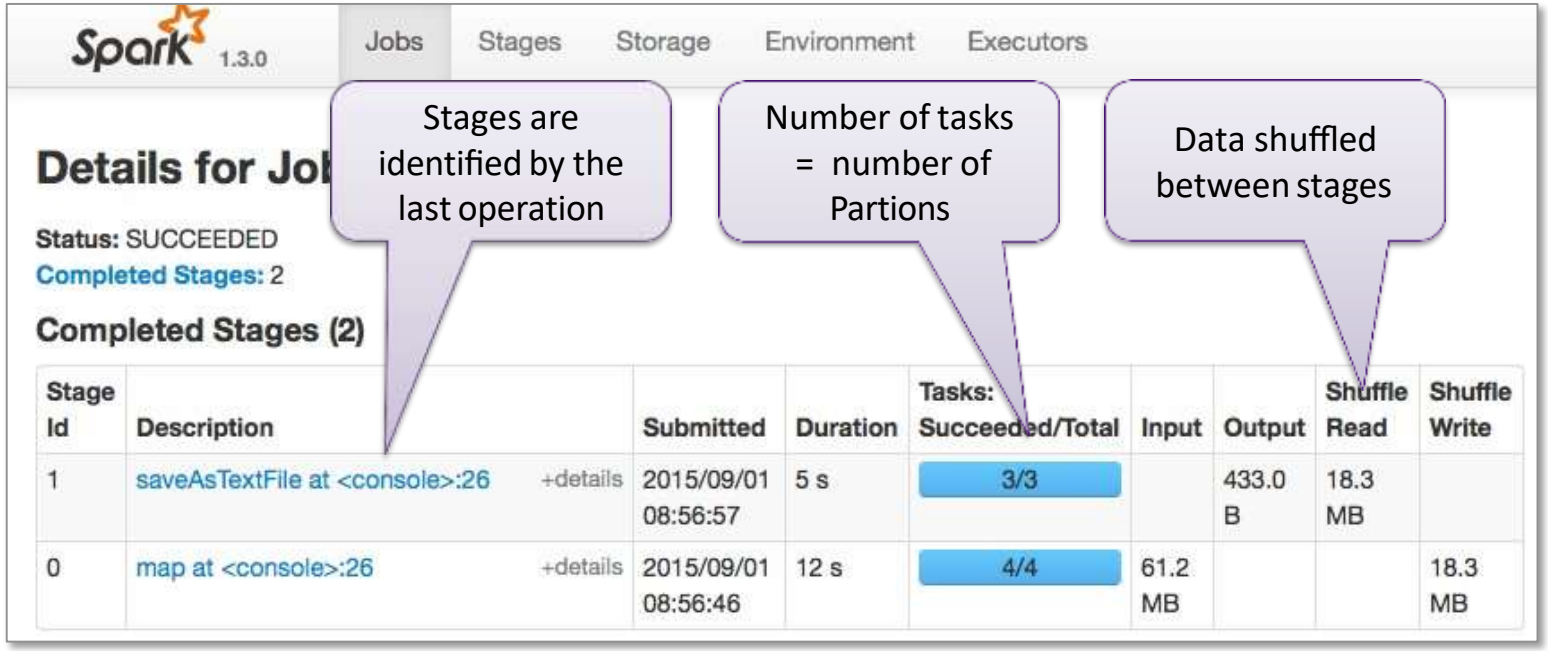
```
> words.reduceByKey(lambda v1, v2: v1 + v2, 15)
```

Viewing Stages in the Spark Application UI (1)

You can view jobs and stages in the Spark Application UI

Viewing Stages in the Spark Application UI (2)

Select the job to view execution stages



Outline

- RDD Partions
- Partioning of File-based RDDs
- HDFS and Data Locality
- Executing Parallel Operations
- Stages and Tasks
- Conclusion

Essential Points

- RDDs are stored in the memory of Spark executor JVMs
- Data is split into Partions – each partion in a separate executor
- RDD operations are executed on Partions in parallel
- Operations that depend on the same partion are pipelined together in stages
  e.g., map, filter
- Operations that depend on multiple Partions are executed in separate stages
  e.g., join, reduceByKey