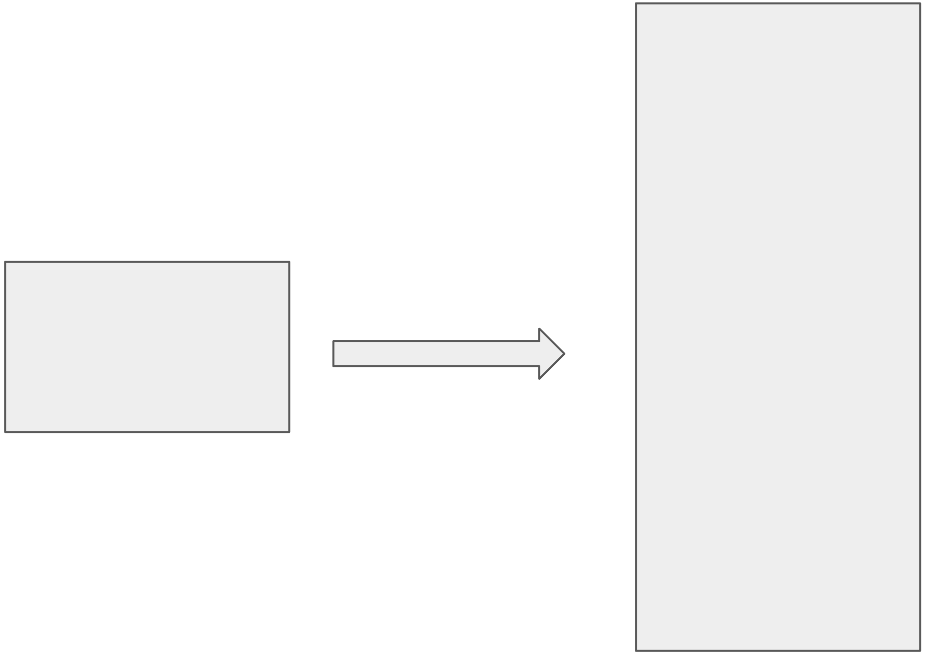
Spark Deep Dive

Partition & Shuffle

https://github.com/scauglog/prez

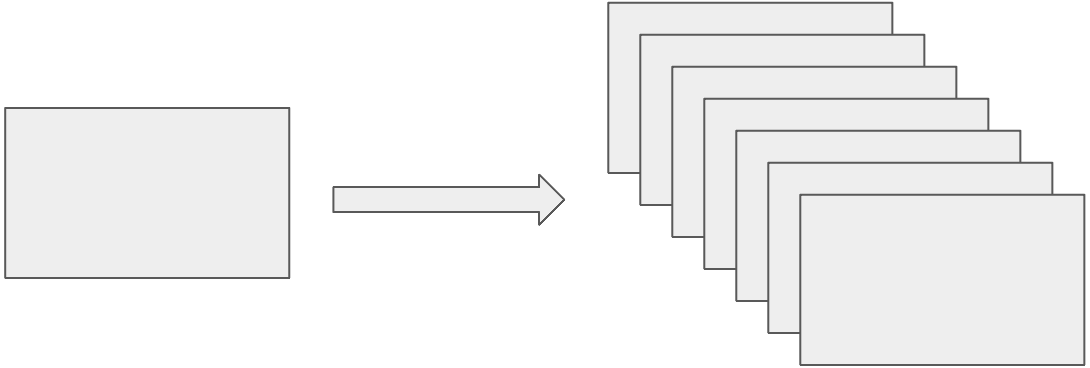
Vertical Approach

- Faster CPU
- More memory
- Simpler Programing
- Hardware Bounded
- Low volume

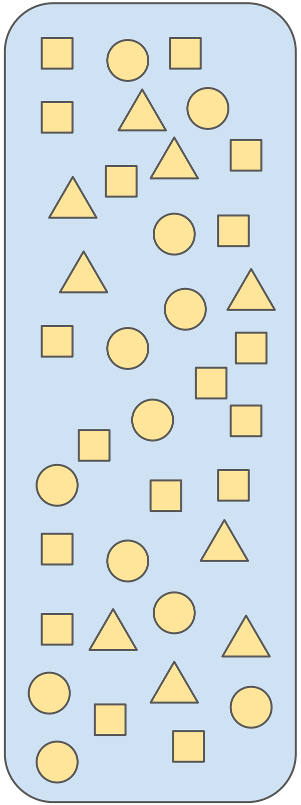


Horizontal Approach

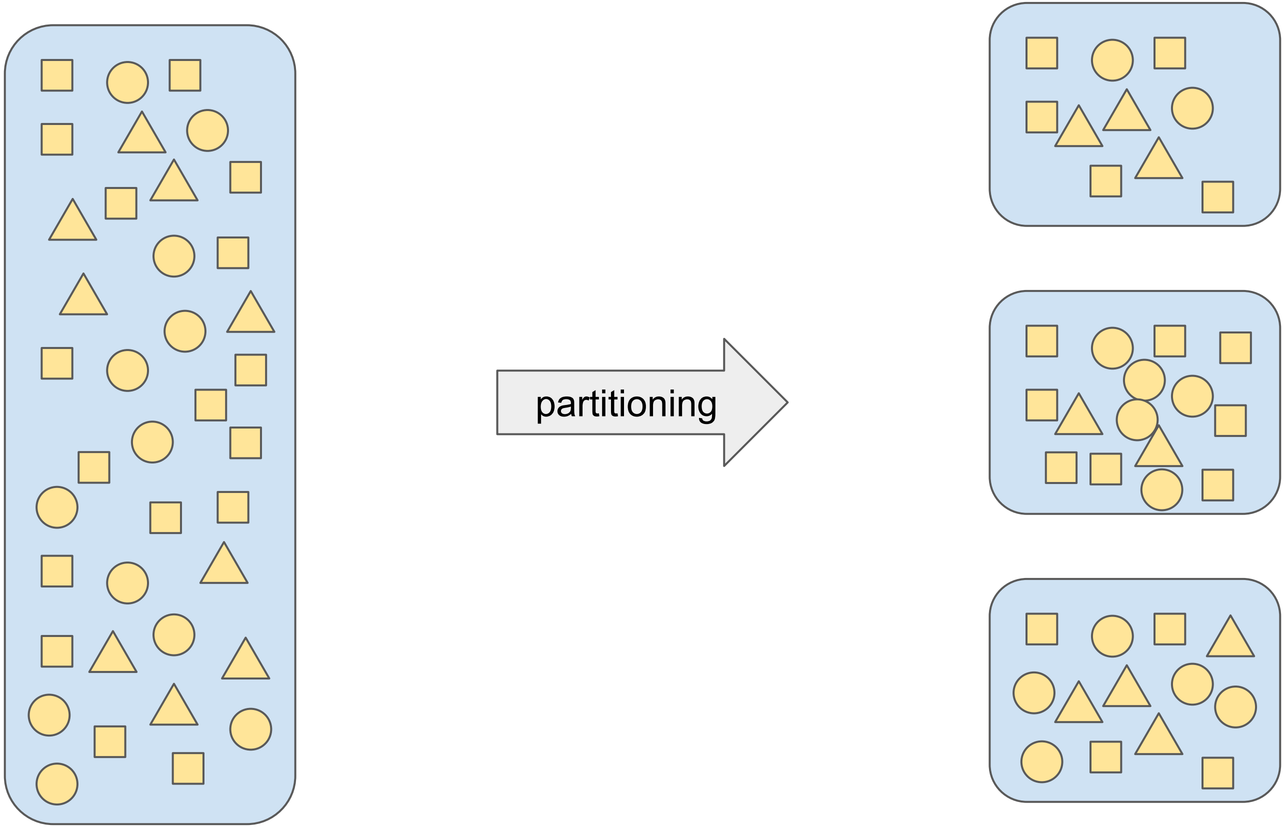
- Big Volume
- Many Server
- Complex programing
  - Failure Handling
  - Distributed Computing

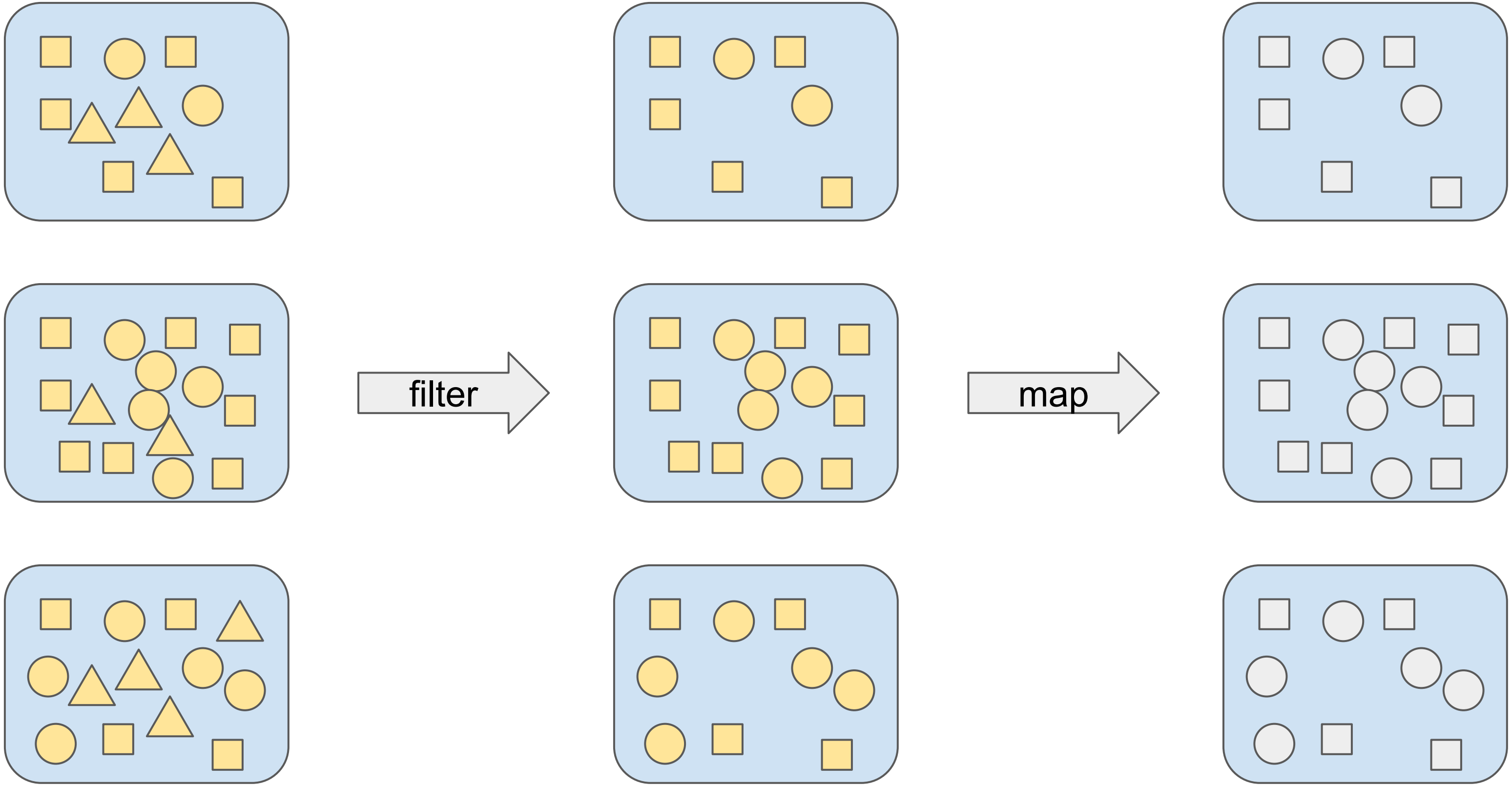


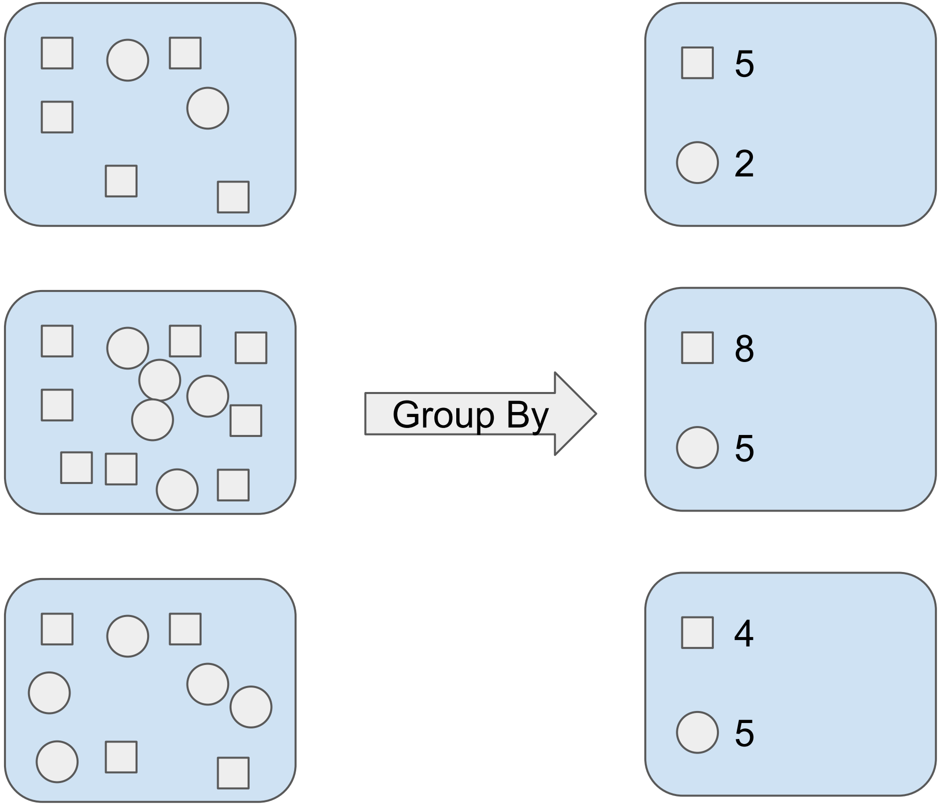
Map Reduce



How many circle and square?


partitioning


filter
map



Group By

□ 5

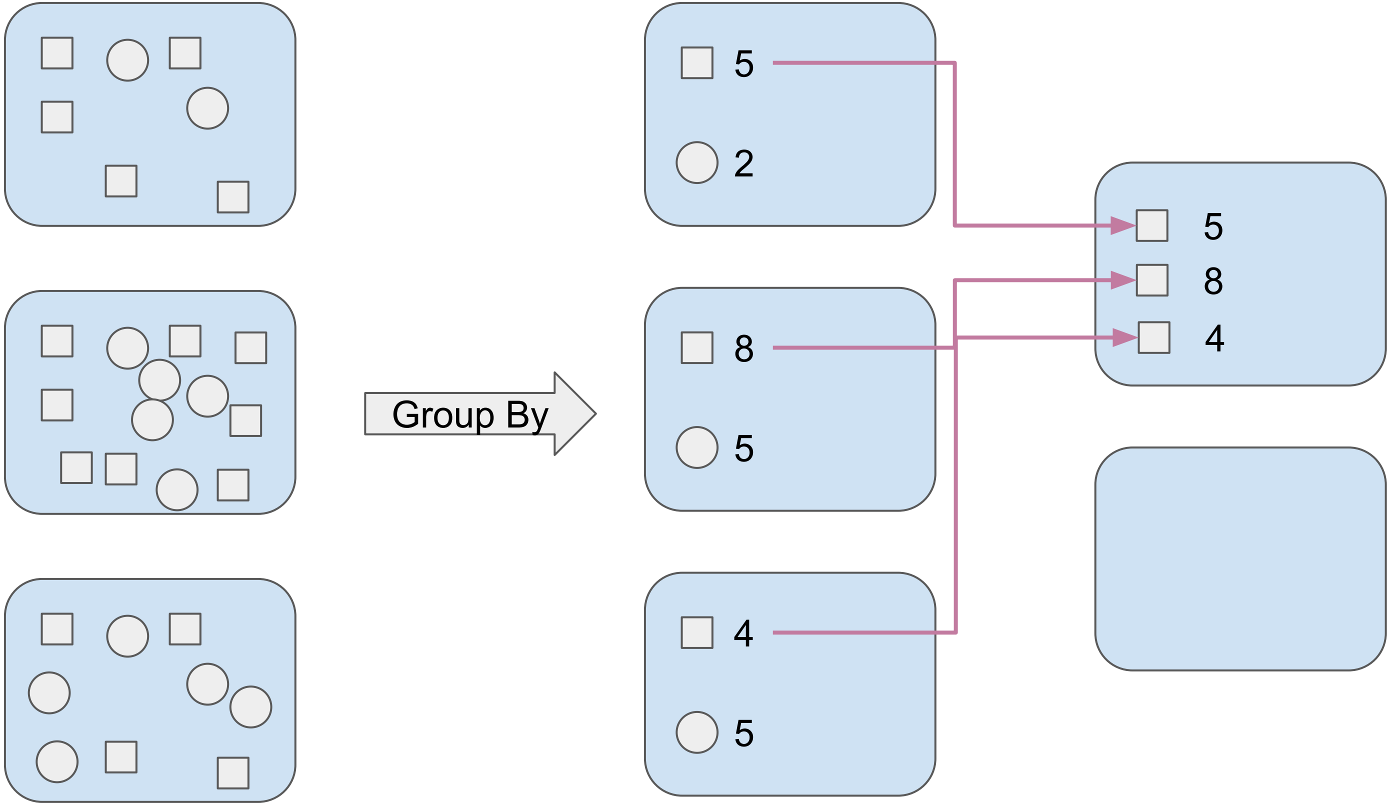
○ 2

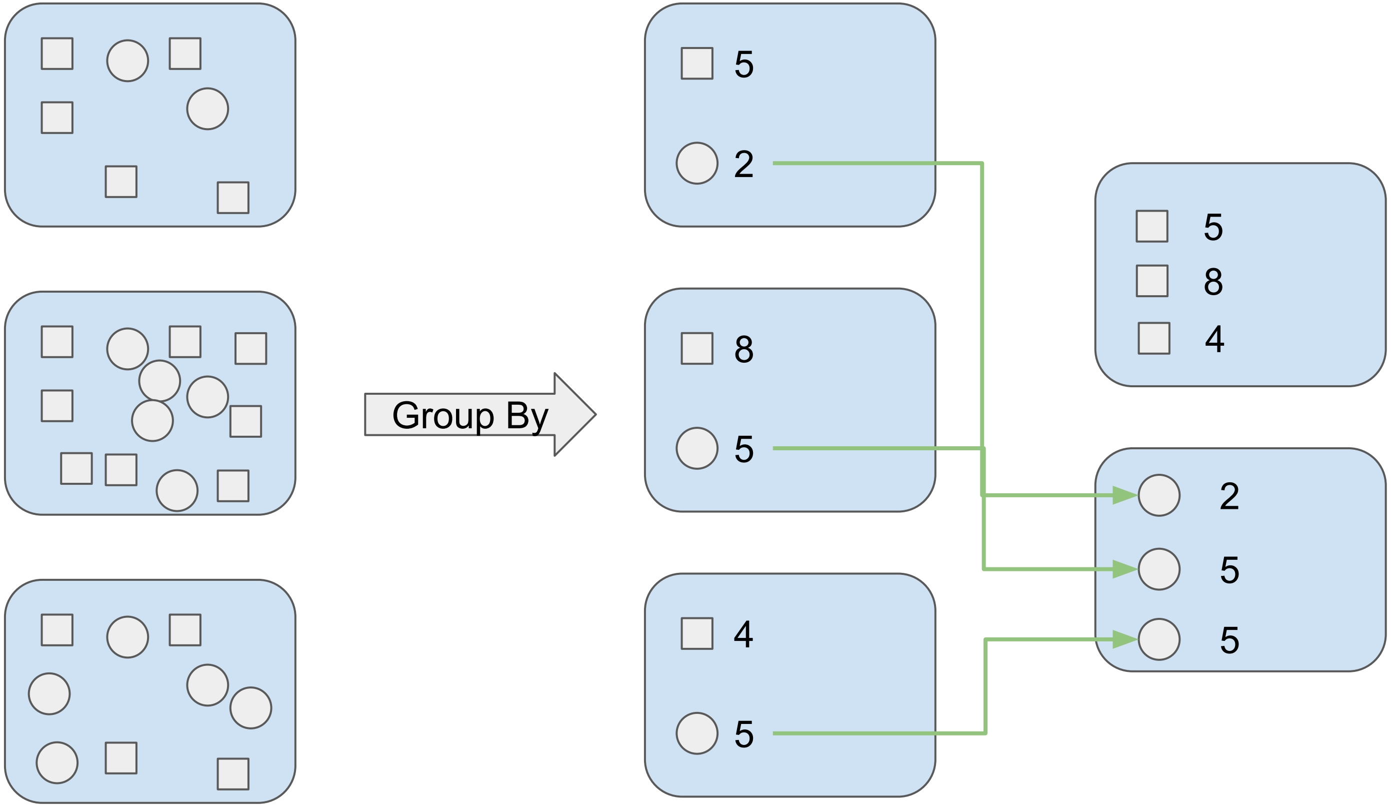
□ 8

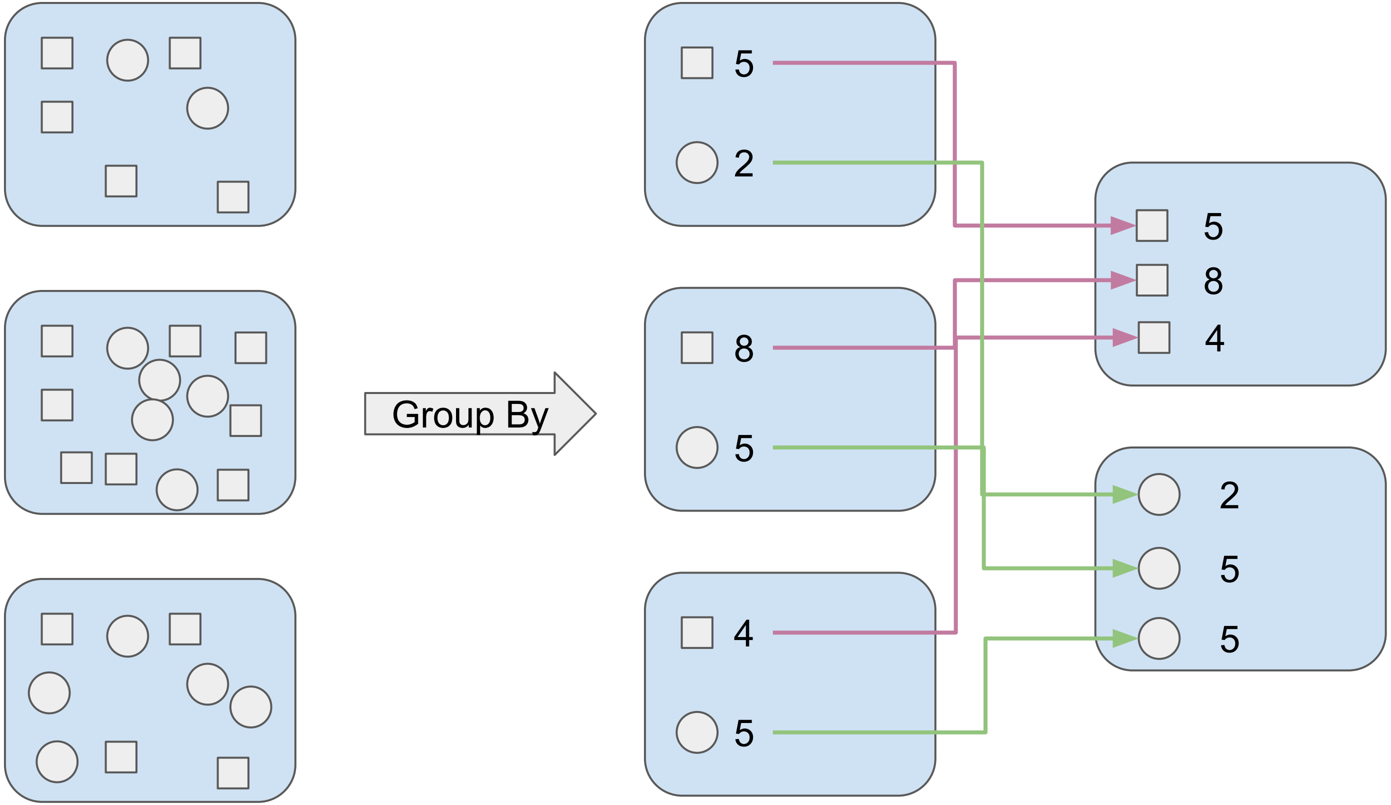
○ 5

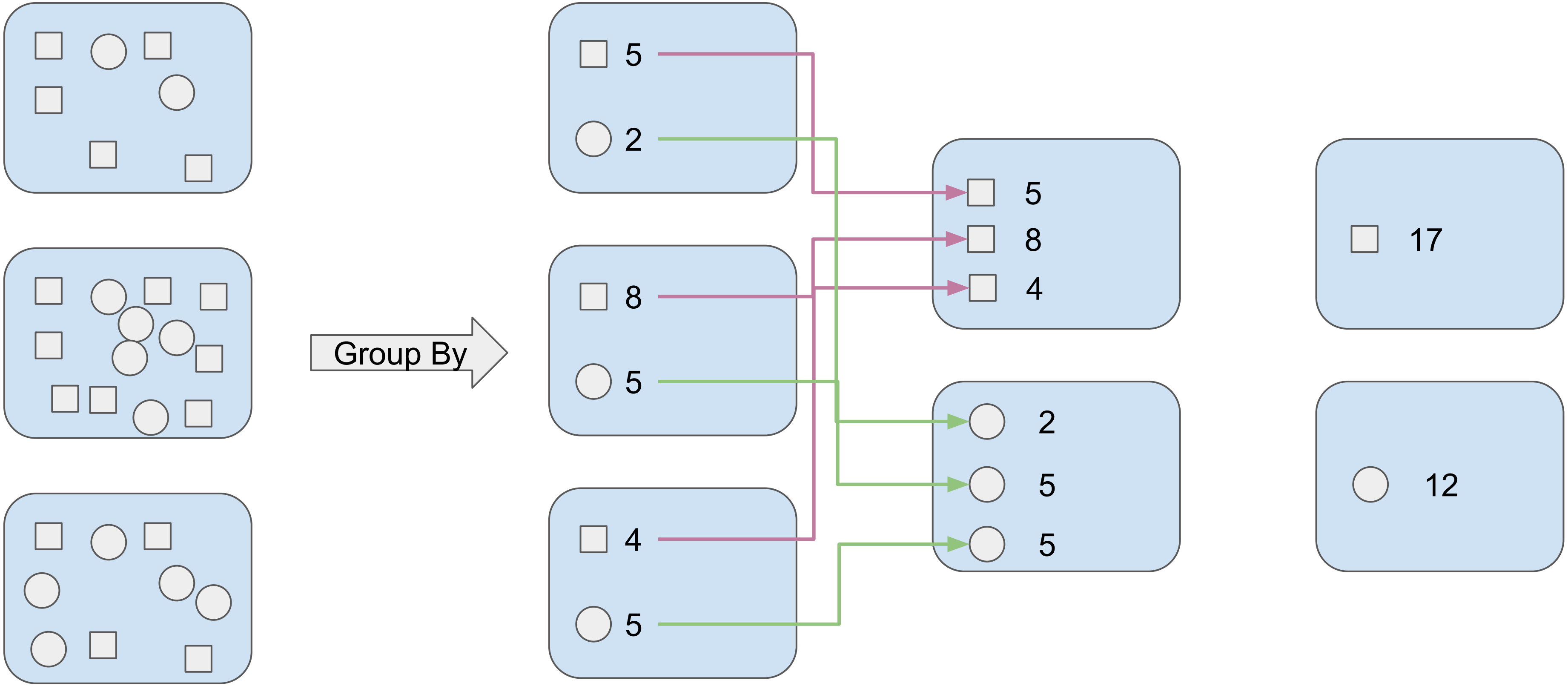
□ 4

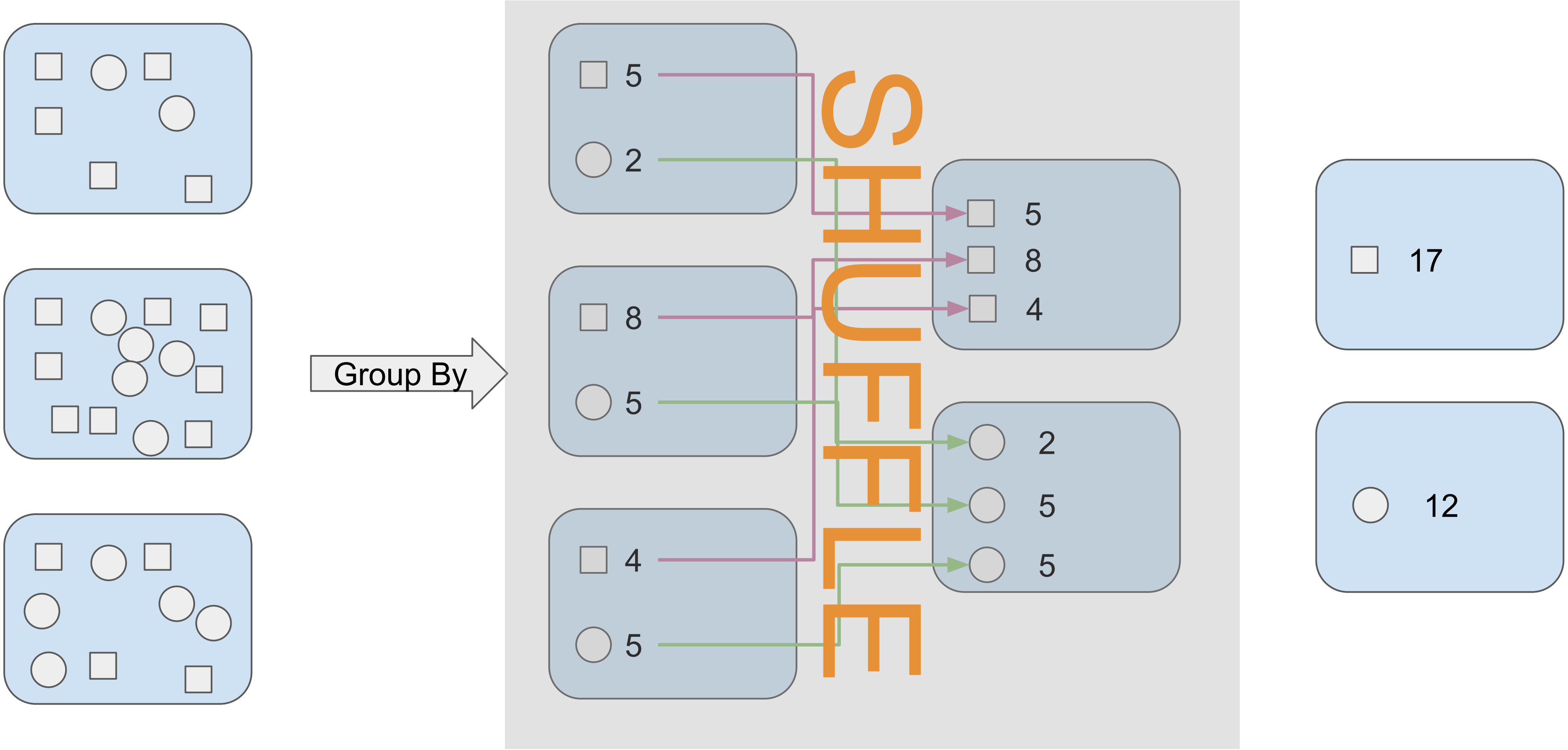
○ 5


Group By
5
2
8
5
4
5
5
8
4


Group By
5
2
8
5
4
5
5
8
4
2
5
5

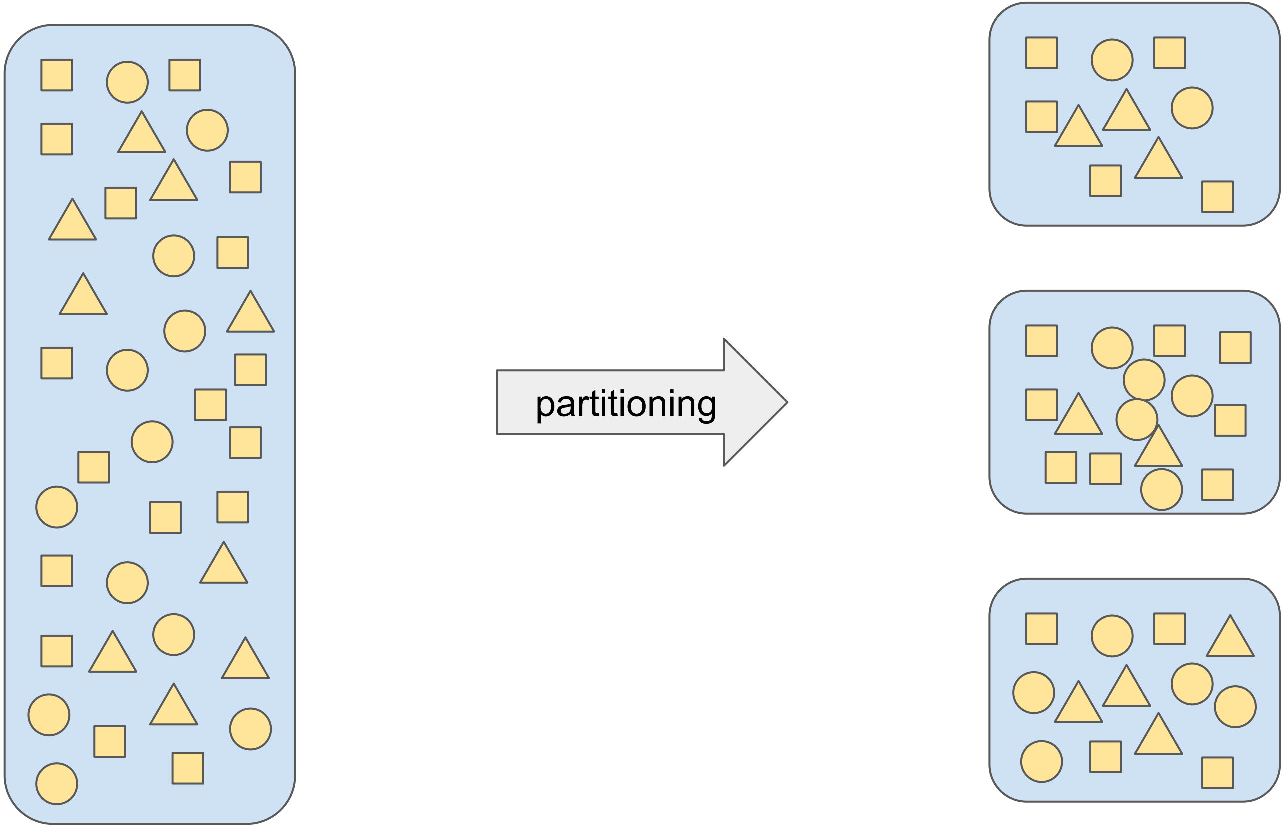

Group By
5
2
8
5
4
5
5
8
4
2
5
5

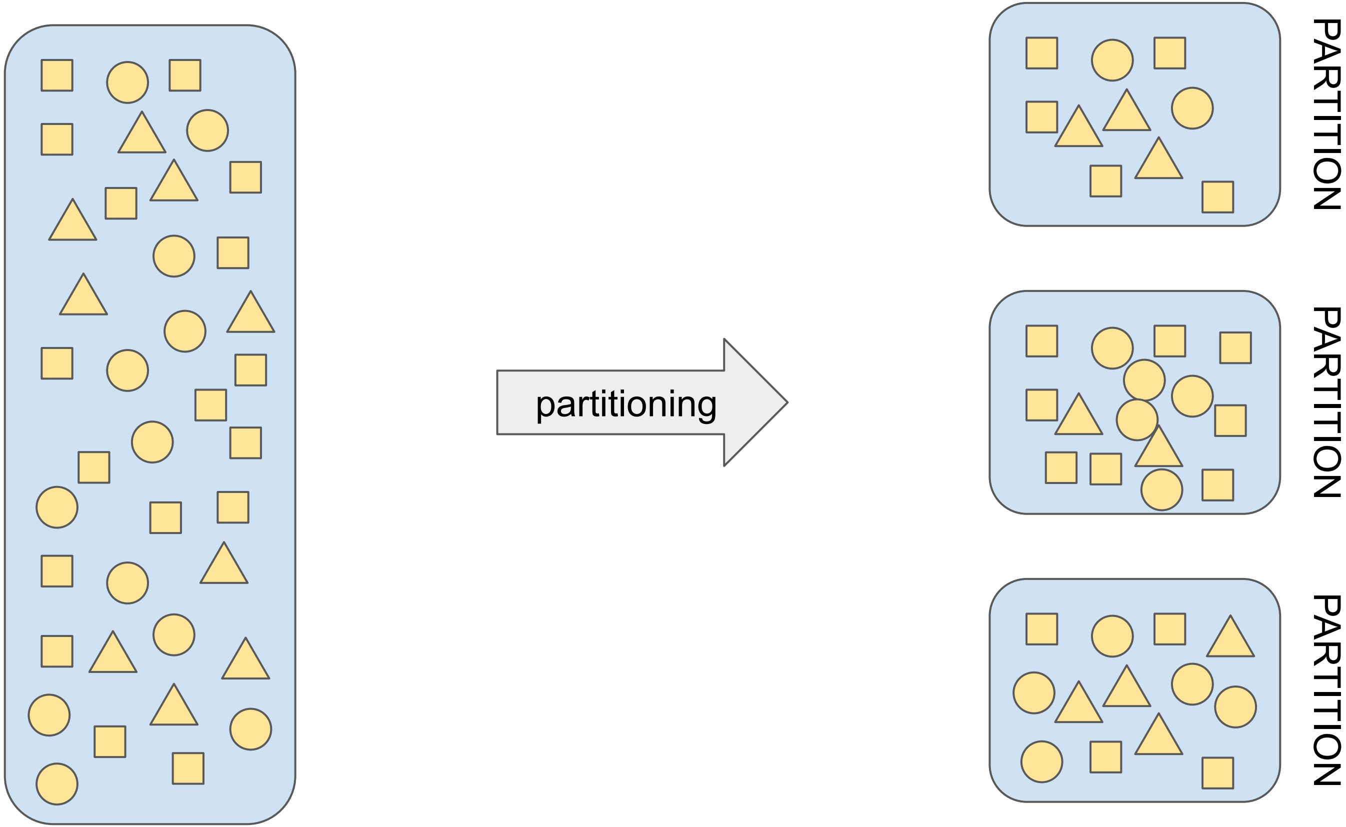

Group By
□ 5
○ 2
□ 8
○ 5
□ 4
○ 5
□ 5
□ 8
□ 4
○ 2
○ 5
○ 5
□ 17
○ 12


Group By
5
2
8
5
4
5
SHUFFLE
5
8
4
2
5
5
17
12

Spark

Partitioning


partitioning



partitioning

PARTITION

PARTITION

PARTITION

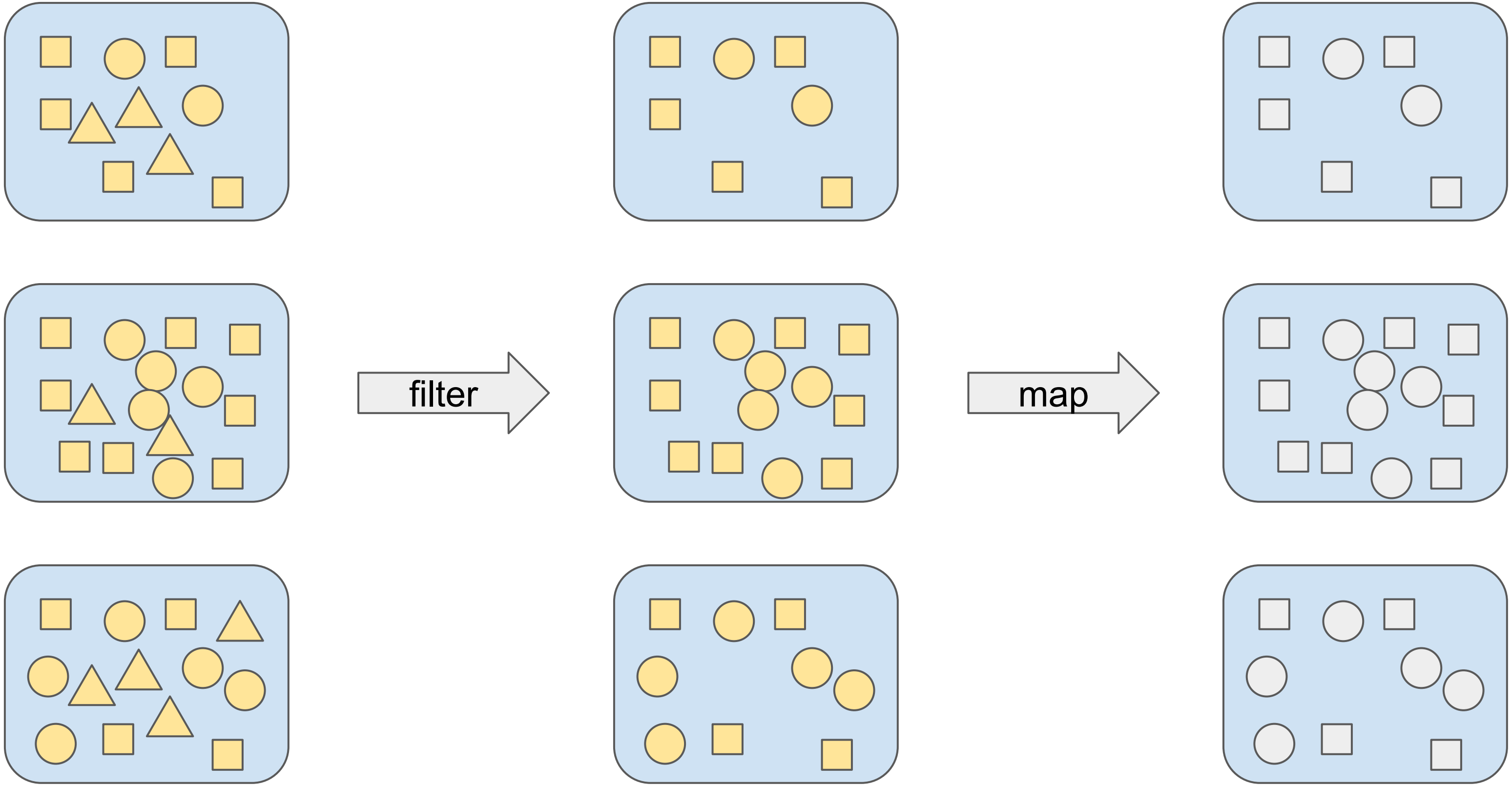
Initial Partitioning

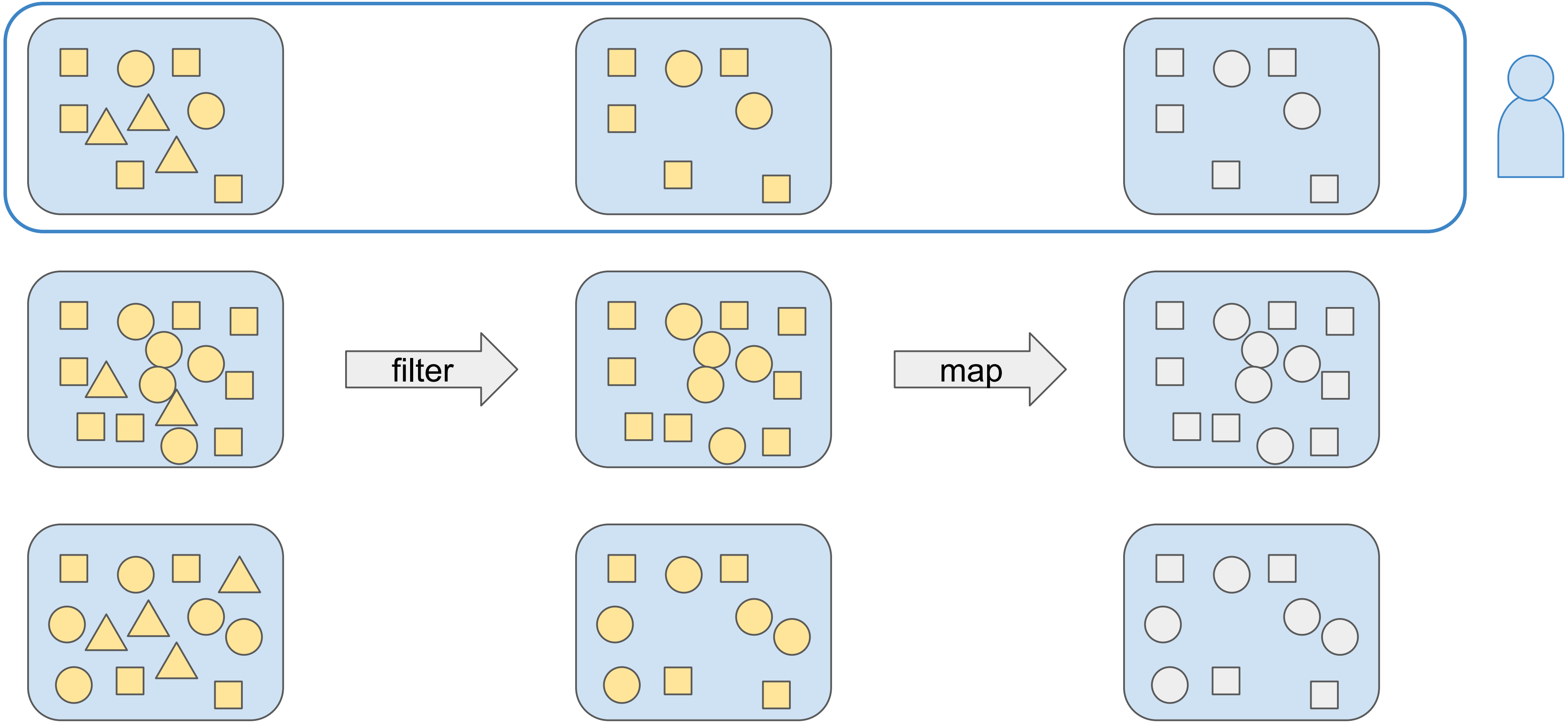
- depends on total size
- depends on parallelism
- small file -> 1 partition per 4 Mo
- intermediate -> 1 partition per task in parallel
- large file -> 1 partition per 128 Mo

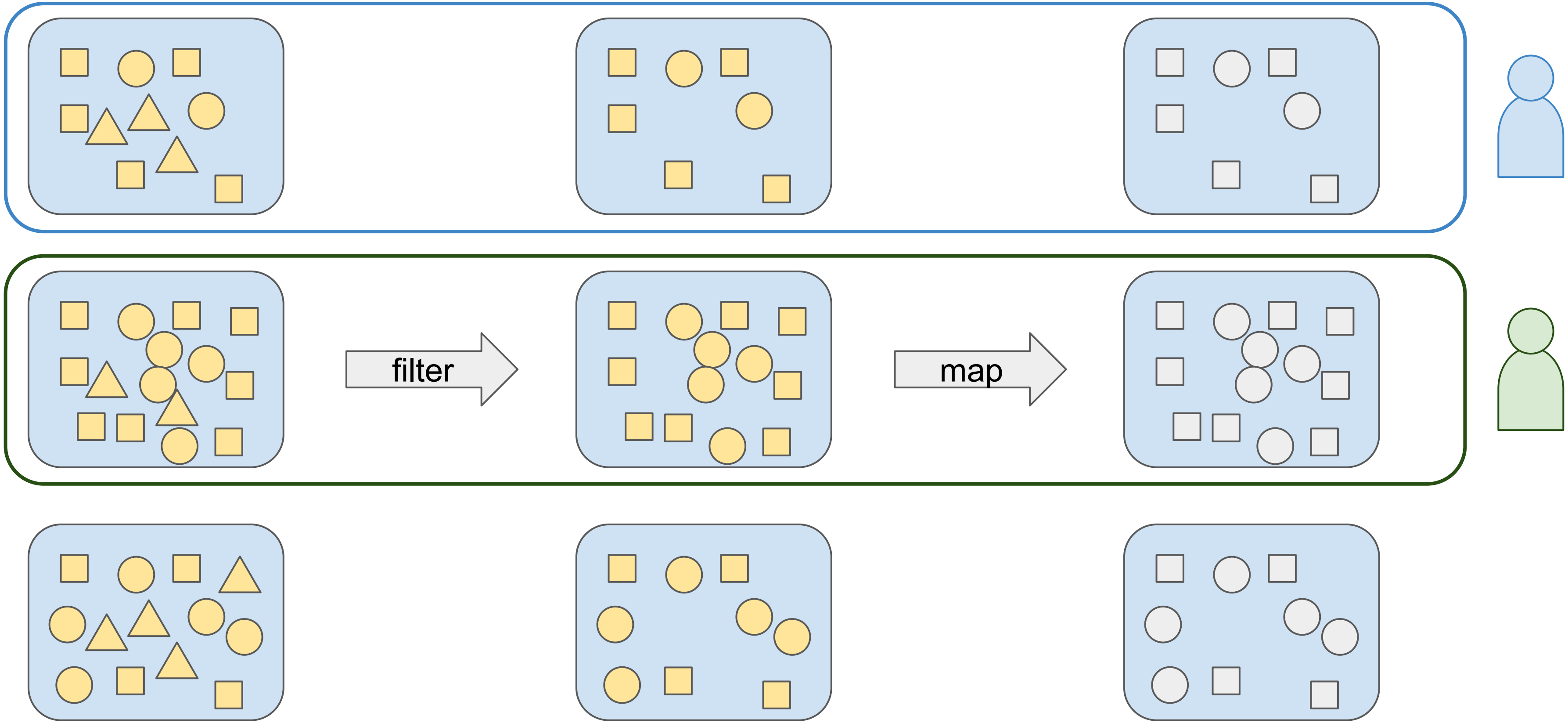
```
Math.min(
      maxPartitionBytes,
      Math.max(openCostInBytes, bytesPerCore)
)
```

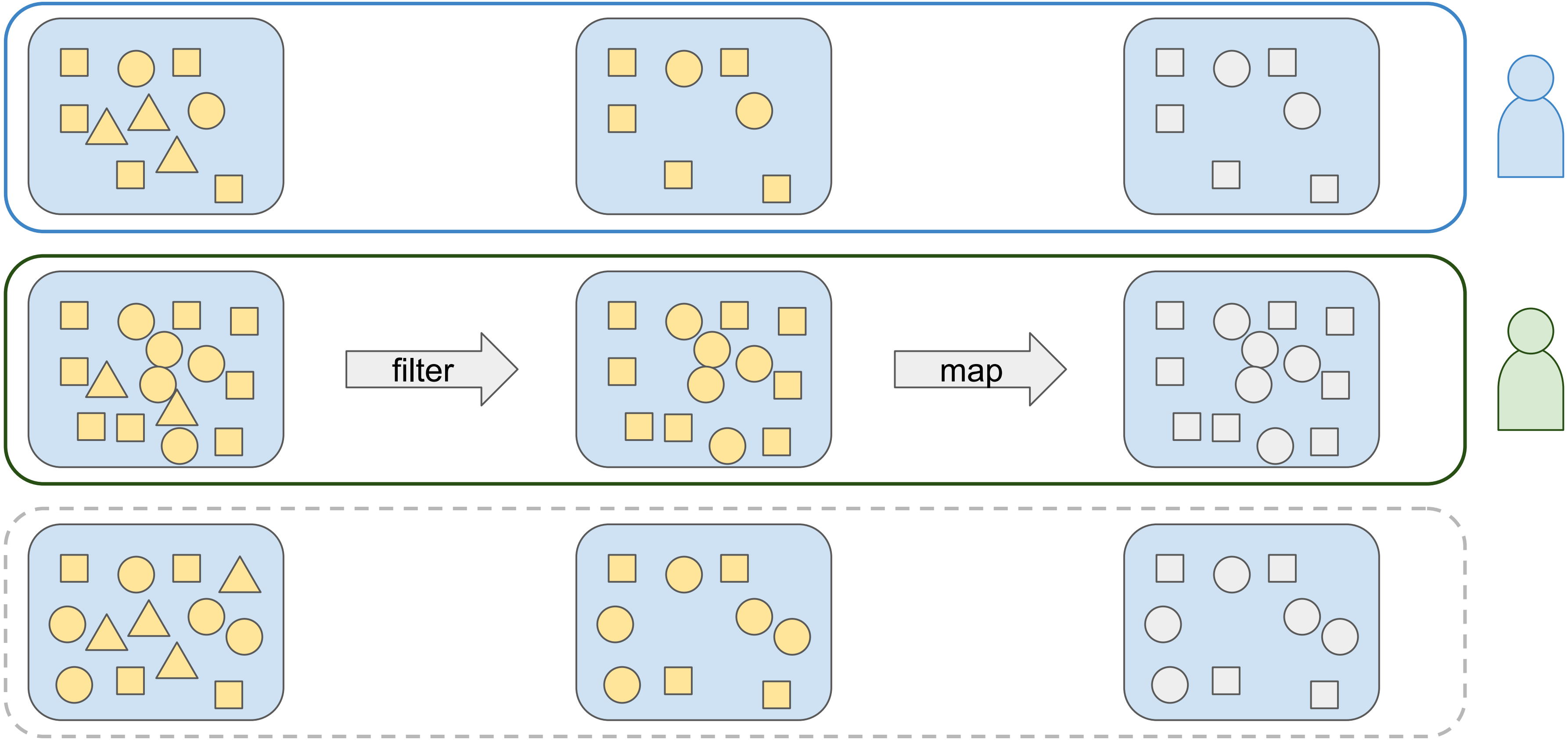
```
spark.sql.files.maxPartitionBytes = 128 Mo

spark.sql.files.openCostInBytes = 4Mo

bytesPerCore = Total size / number of task in parallel
```

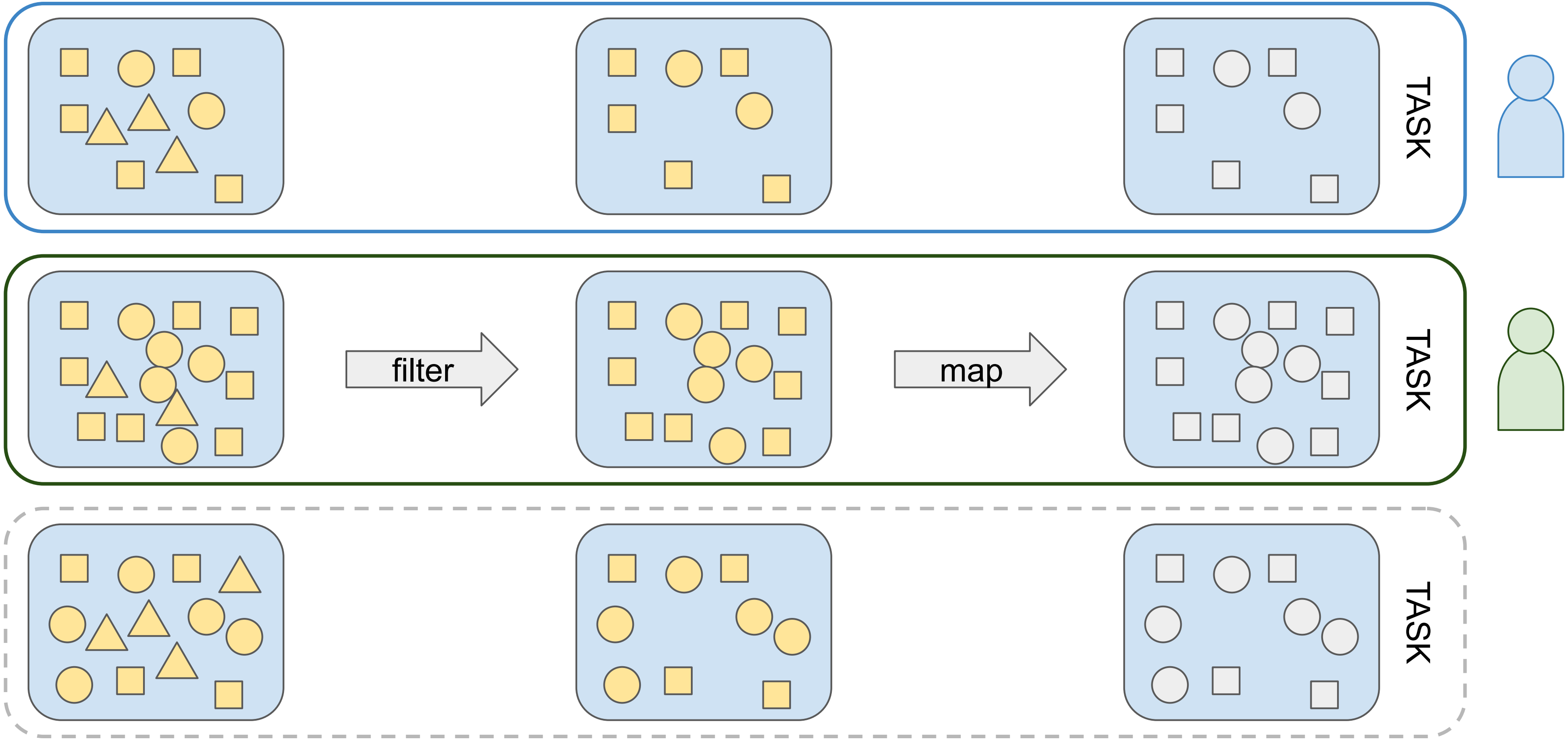
Parallelize


filter
map


filter
map


filter
map


filter
map

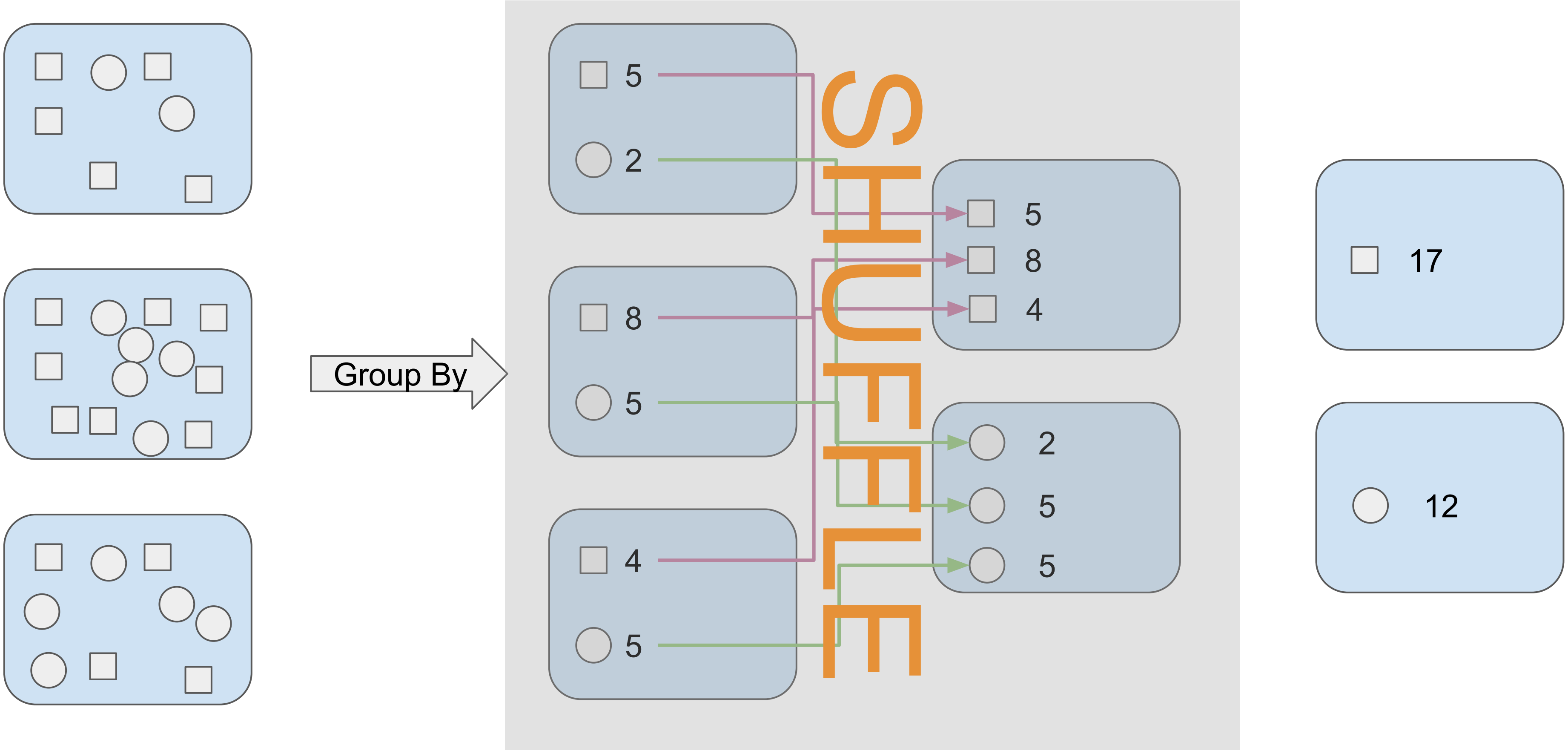

TASK
filter
map
TASK
TASK

Executor parallelism

- one executor process many partition in parallel
  - task in parallel = `spark.executor.cores / spark.task.cpus`
- limit the number of executor
  - less data send in case of broadcast
  - executor need some extra memory to work
- limit the number of cores by executor
  - executor with lots of core can’t be allocated if nodes are too busy
  - performance drawback when reading file

Shuffle

everyday I’m shuffling


Group By
5
2
8
5
4
5
SHUFFLE
5
8
4
2
5
5
17
12

Shuffle

- put all key on the same node
- create File in local FS (spark.shuffle.spill)
- different kind of shuffle
  - hash shuffle
  - consolidate hash shuffle
  - sort shuffle
  - tungsten sort shuffle

Partitioning RDD

- each aggregation operation has a signature with numPartitions
- `spark.default.parallelism`
  - ignored when working with dataframe
  - largest number of partition in a parent RDD
- `.coalesce()`
  - can only reduce number of partition
  - no shuffle involve
  - union of partition
- `.repartition()`
  - shuffle data
  - solve skewed data issue

```
myRdd.reduceByKey(_+_, numPartitions = 14)

myRdd.sortByKey(true, numPartitions = 14)

myRdd.join(myRdd2, numPartitions = 14)

myRdd.coalesce(numPartitions = 5)

myRdd.repartition(numPartitions = 5)
```

Partitioning DataFrame

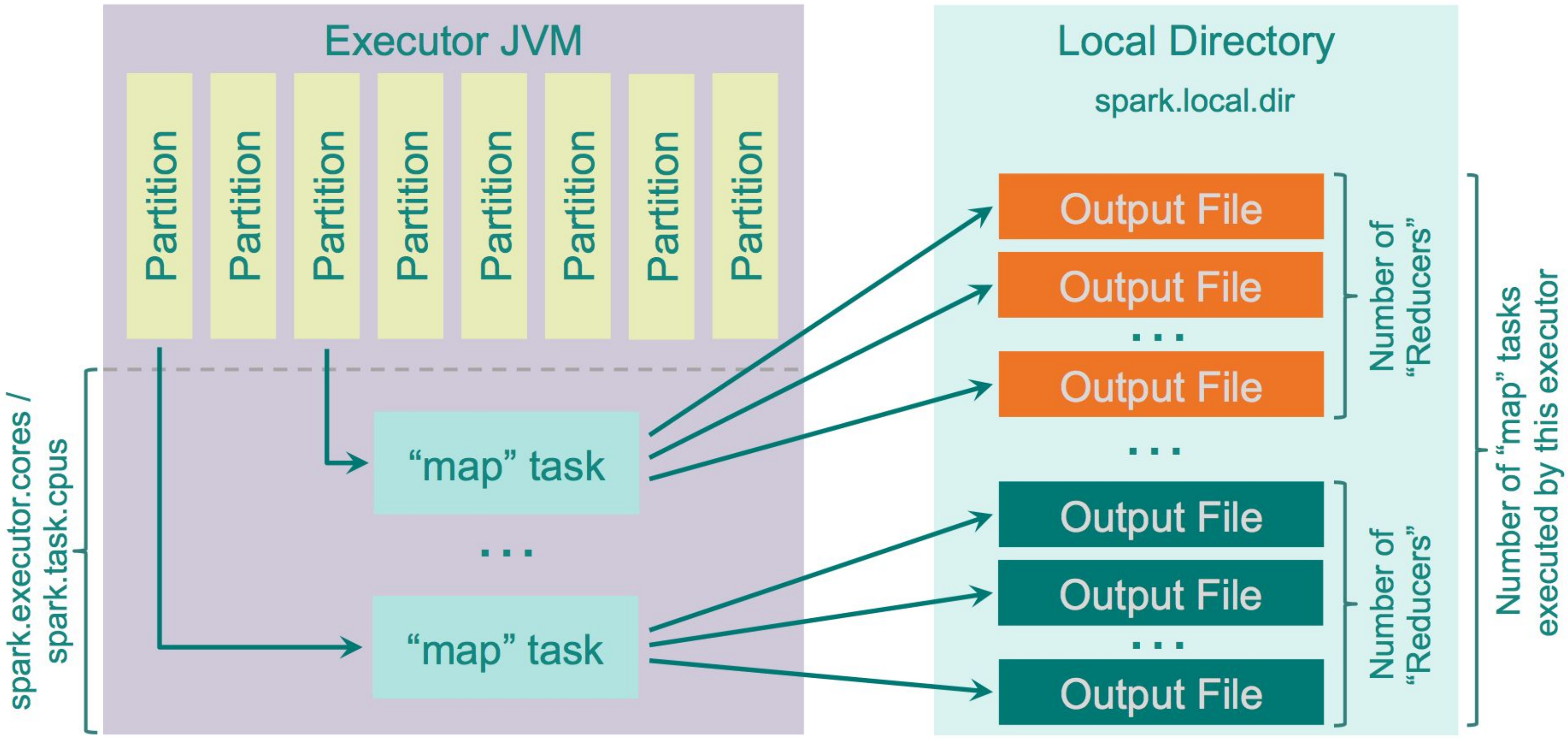
- we can’t set a different numPartitions for shuffle operation
- `spark.sql.shuffle.partitions`
  - global for each shuffle operation
  - should be a multiple of the number of partition computed in parallel (num-executor * executor-cores / task.cores)
  - default = 200

Hash Shuffle

- `spark.shuffle.manager`=hash
- 1 file for each reducer for each mapper
- number of file = M*R
- fast
- big amount of files written to FS
- lot of random IO

M: mapper task
R: reducer task

Hash Shuffle

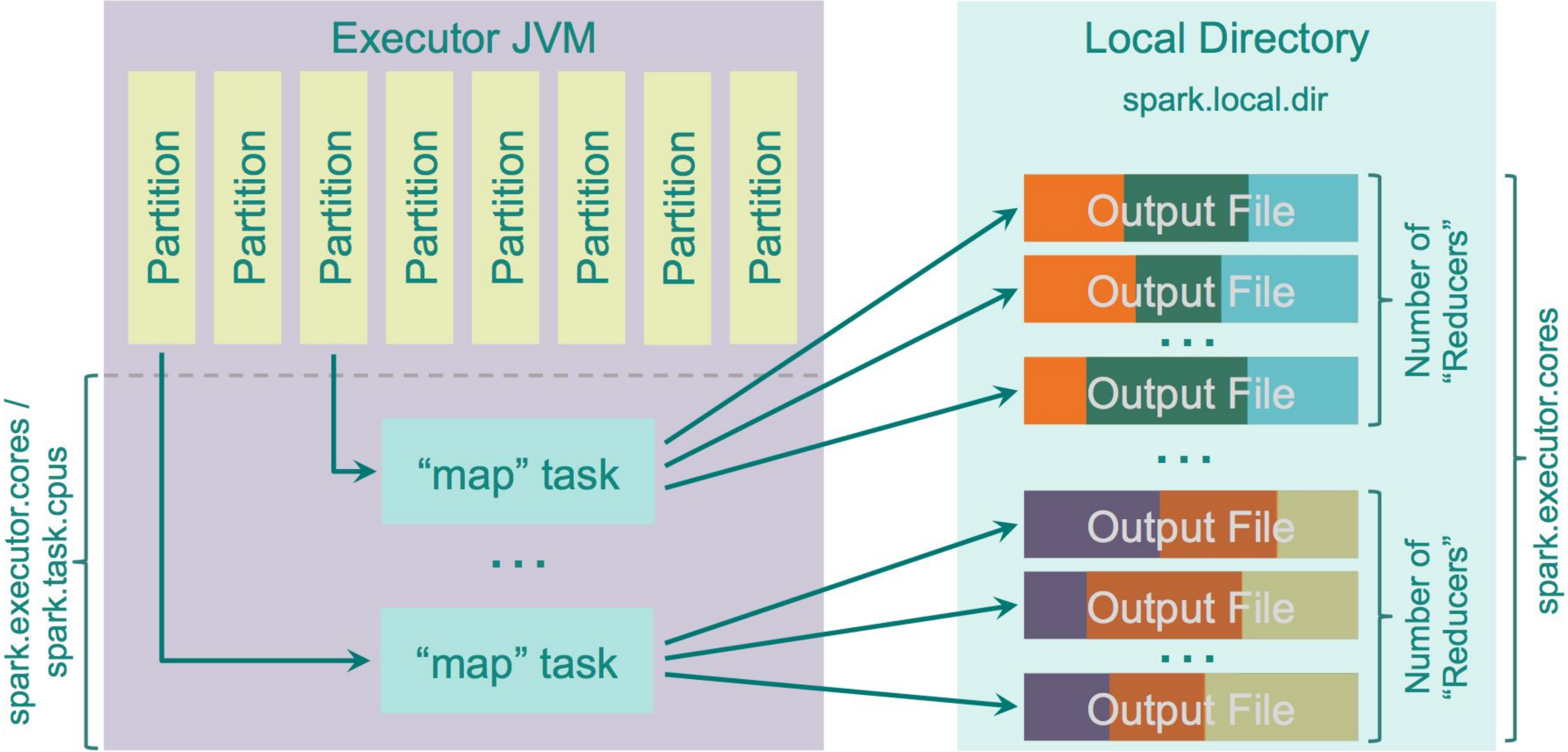


Consolidate Hash Shuffle

- `spark.shuffle.manager`=hash
- `spark.shuffle.consolidateFiles`=true
- 1 file foreach reducer for each task in parallel by executor
- number of files = E * C/T * R
- less file written

M: mapper task
R: reducer task
E: num-executor
C: executor-cores
T: tasks.cpu

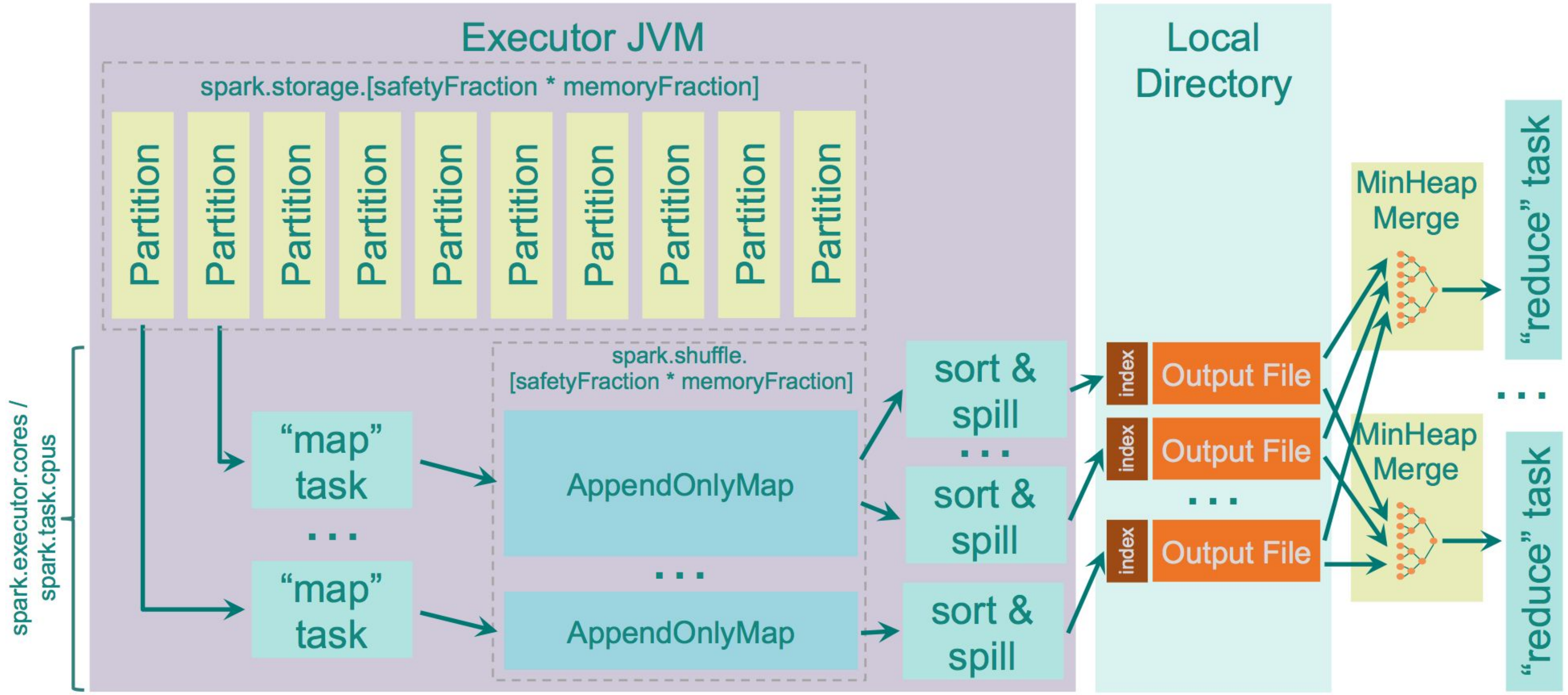
Consolidate Hash Shuffle



Sort Shuffle

- `spark.shuffle.manager`=sort
- 1 file by mapper ordered by reducer and indexed
- if R<200 then hash (`spark.shuffle.sort.bypassMergeThreshold`)
- sort data on map side using TimSort
- merge by reducer before sending to reducer
- sort after shuffle is faster

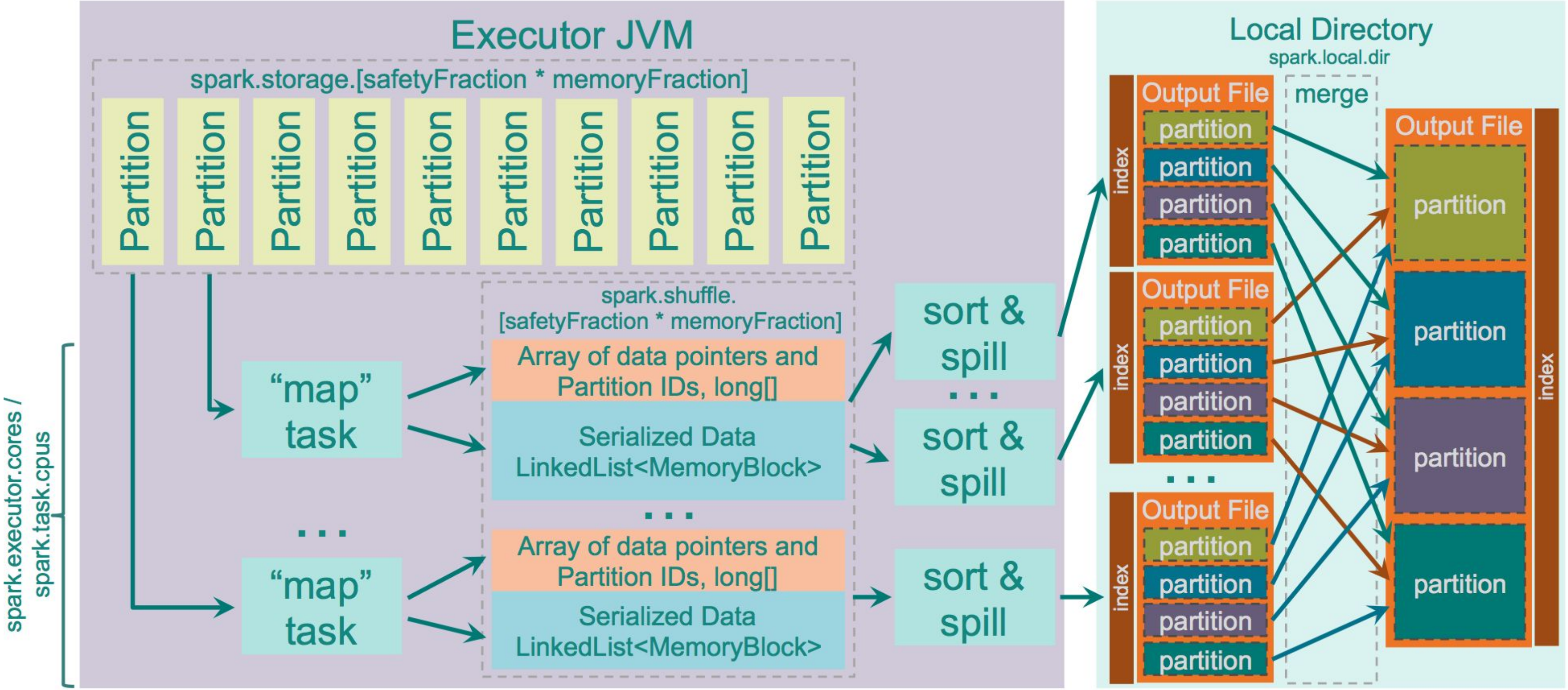
Sort Shuffle



Tungsten Sort Shuffle

- `spark.shuffle.manager`=tungsten-sort
- operate on serialized data
- cache-efficient sorter
- work only if:
  - no aggregation (deserialisation)
  - less than 16 777 216 output partition (2^{24}, 3 octets)
  - row size < 128MB in serialized form
- no more fast sort after shuffle

Tungsten Sort Shuffle

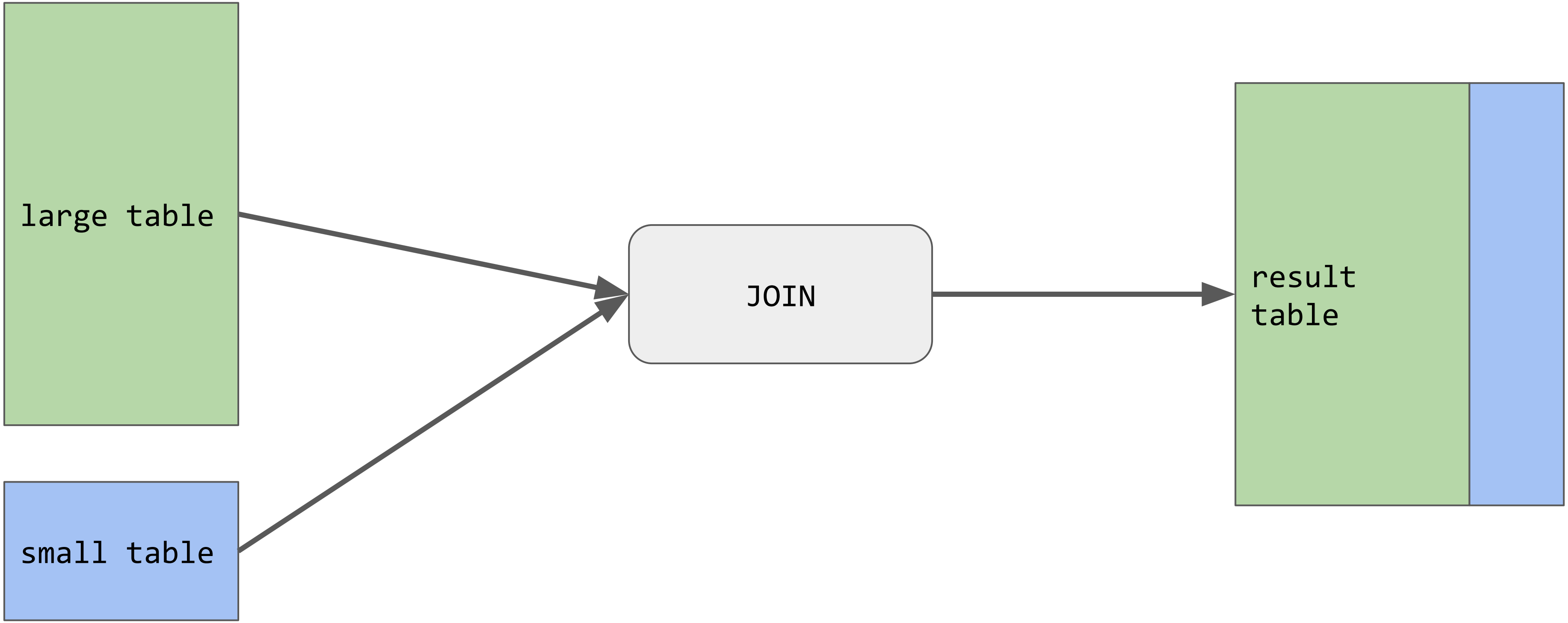


Shuffle take away

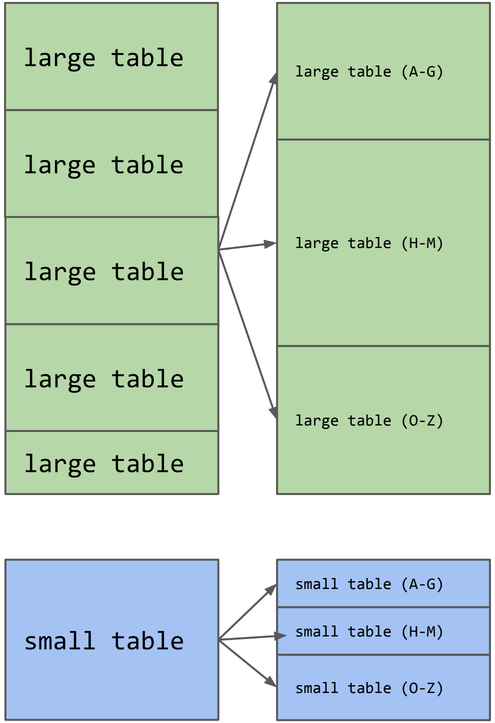
- hash shuffle
  - for small data
  - generate lots of file
- sort shuffle
  - for large data
  - slower due to sort
  - less file generated
- tune with `spark.shuffle.sort.bypassMergeThreshold`

JOIN

Join



Join



large table

large table

large table

large table

large table

large table (A-G)

large table (H-M)

large table (O-Z)

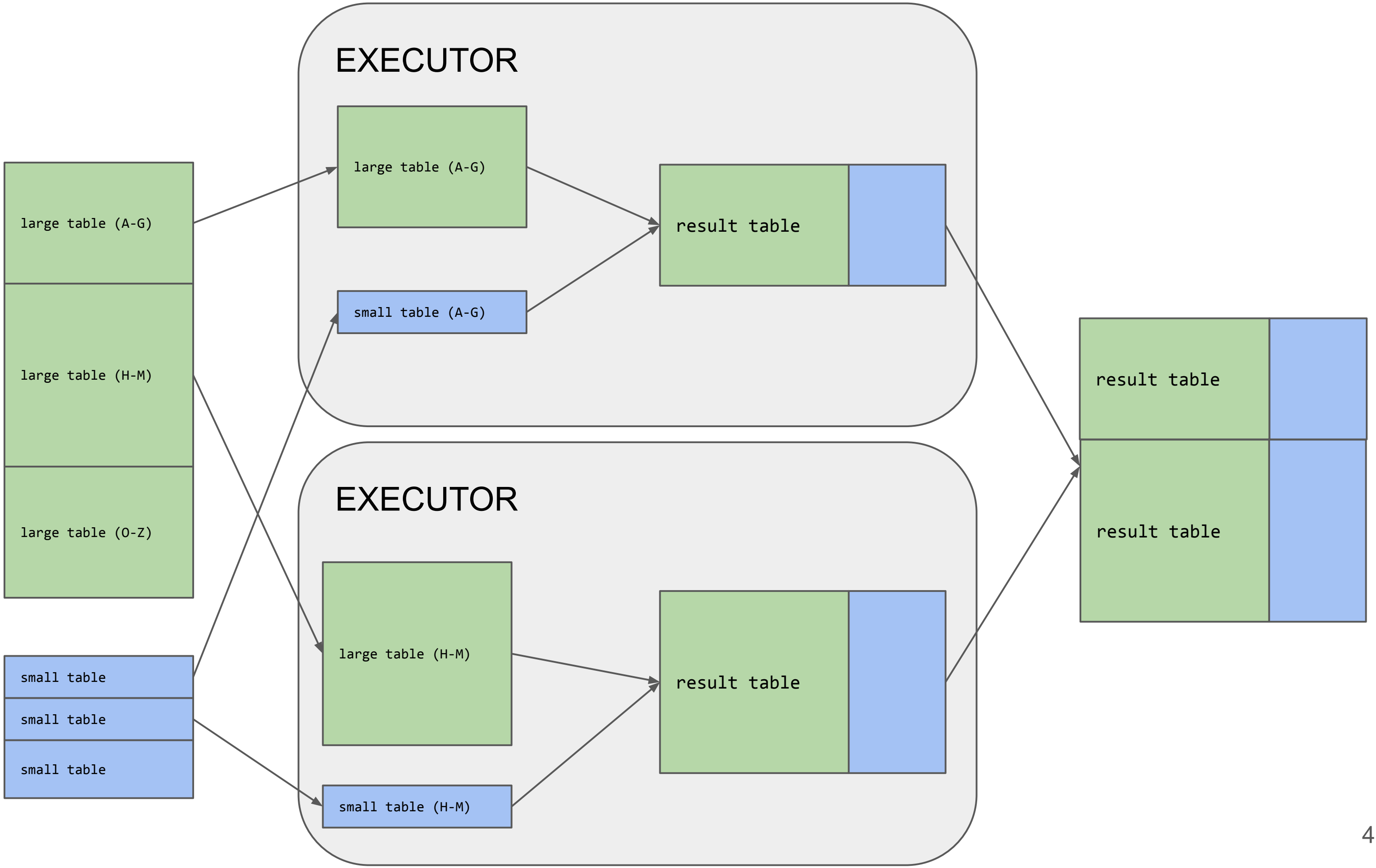
small table

small table (A-G)

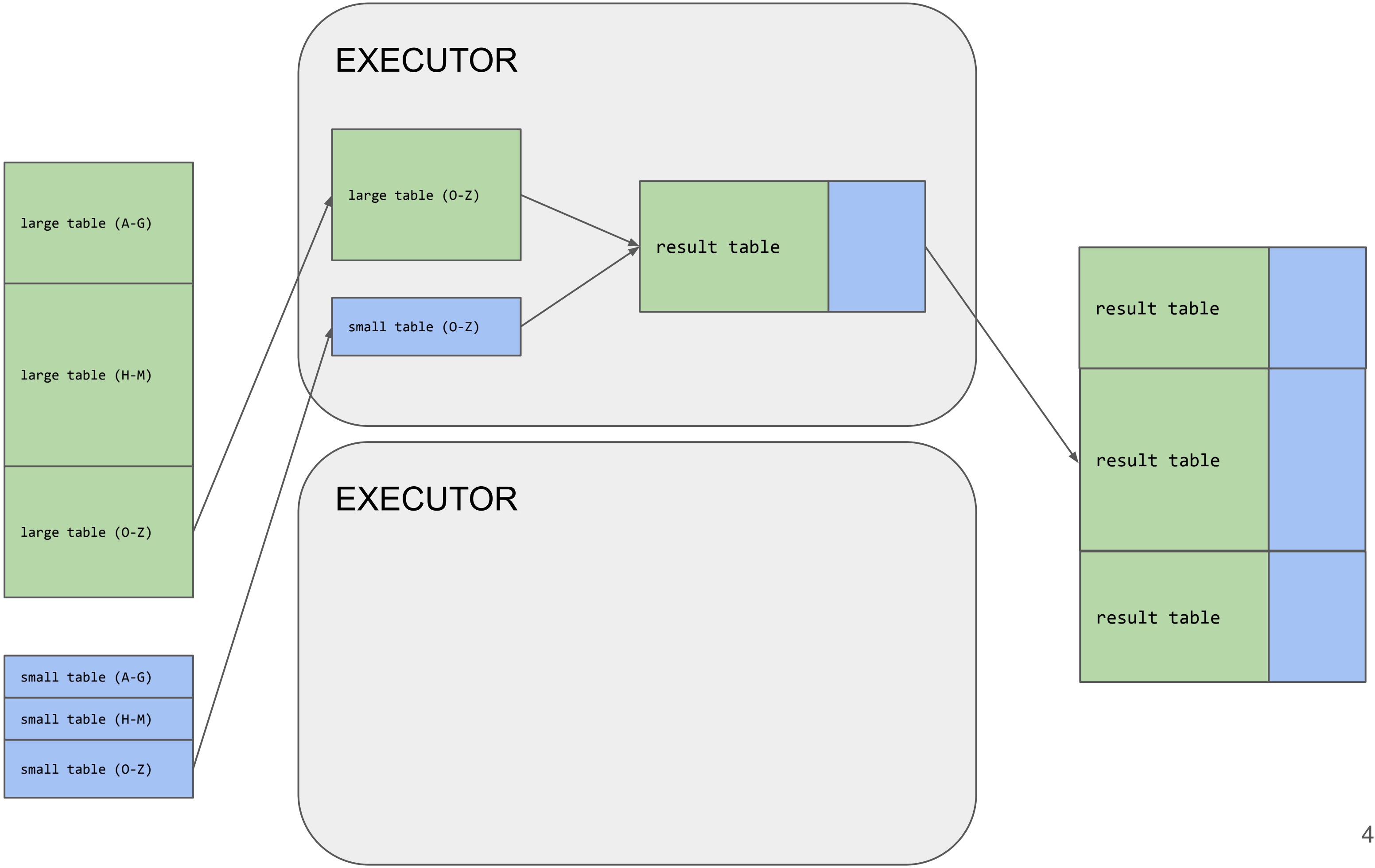
small table (H-M)

small table (O-Z)

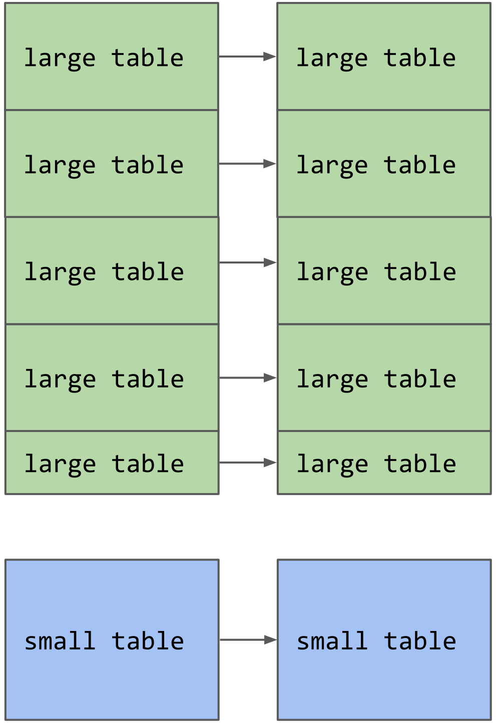
Join


EXECUTOR
large table (A-G)
small table (A-G)
result table
EXECUTOR
large table (H-M)
small table (H-M)
result table
large table (A-G)
large table (H-M)
large table (O-Z)
small table
small table
small table
result table
result table

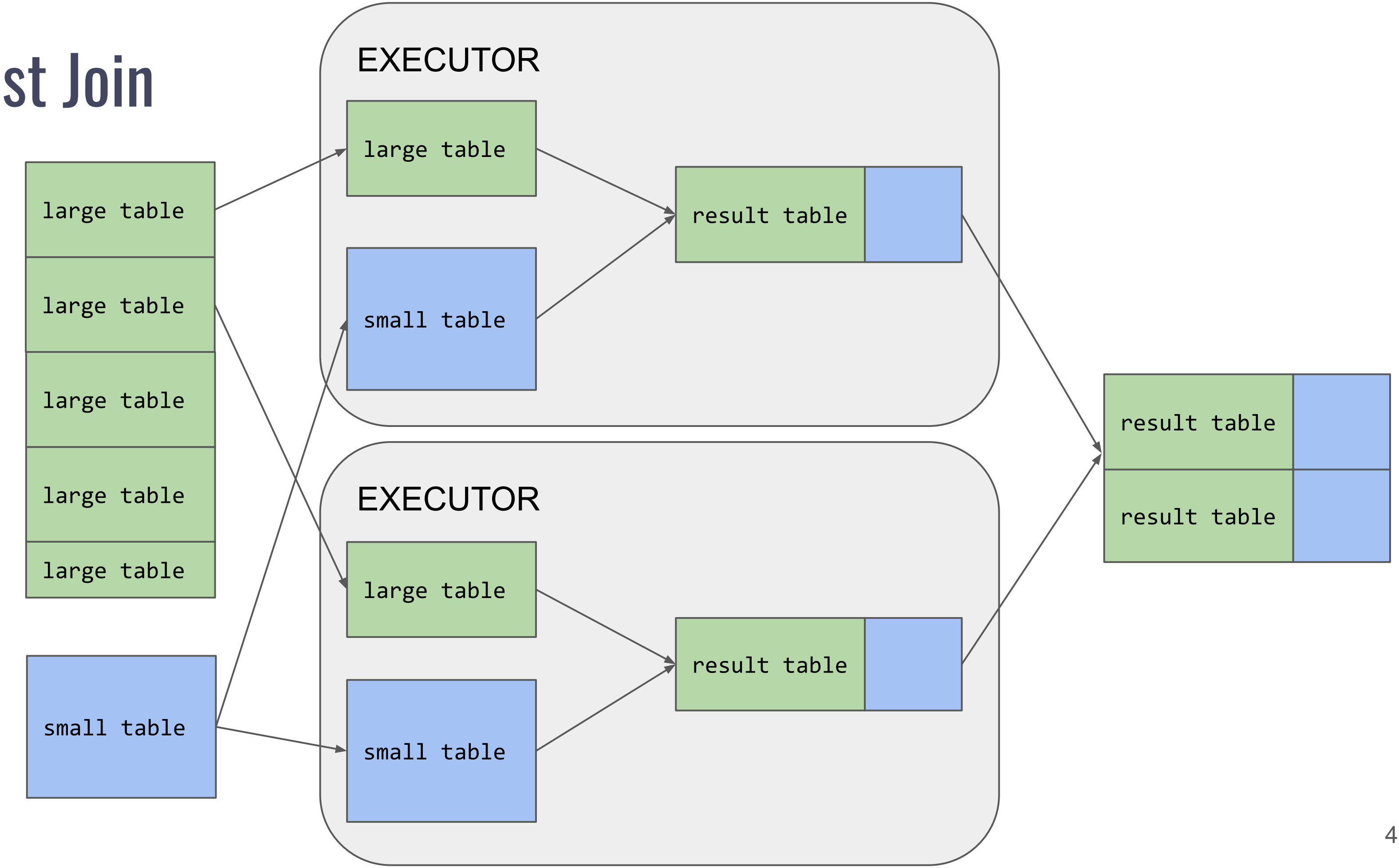
Join


EXECUTOR
large table (O-Z)
small table (O-Z)
result table
EXECUTOR
large table (A-G)
large table (H-M)
large table (O-Z)
small table (A-G)
small table (H-M)
small table (O-Z)
result table
result table
result table

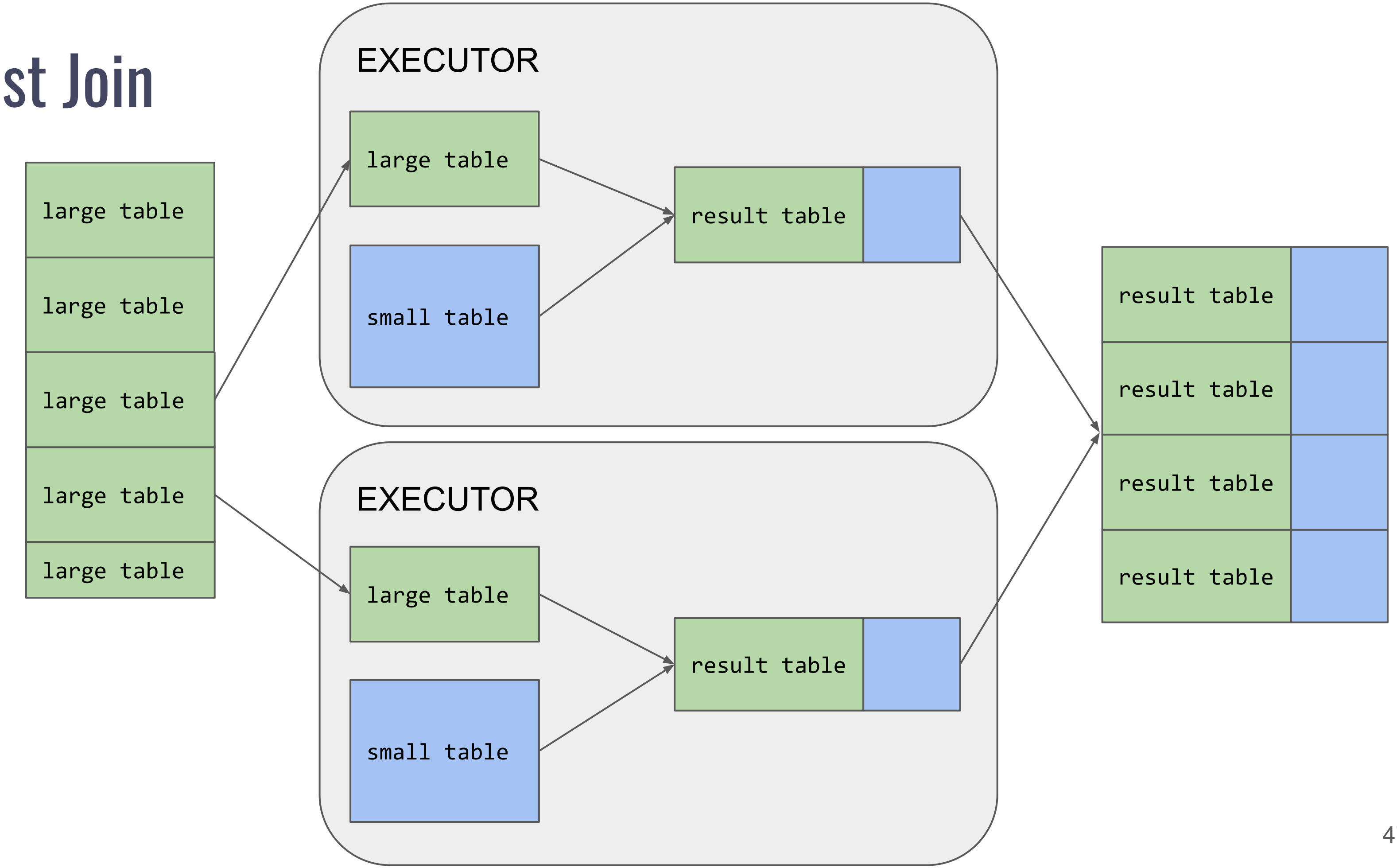
Broadcast Join



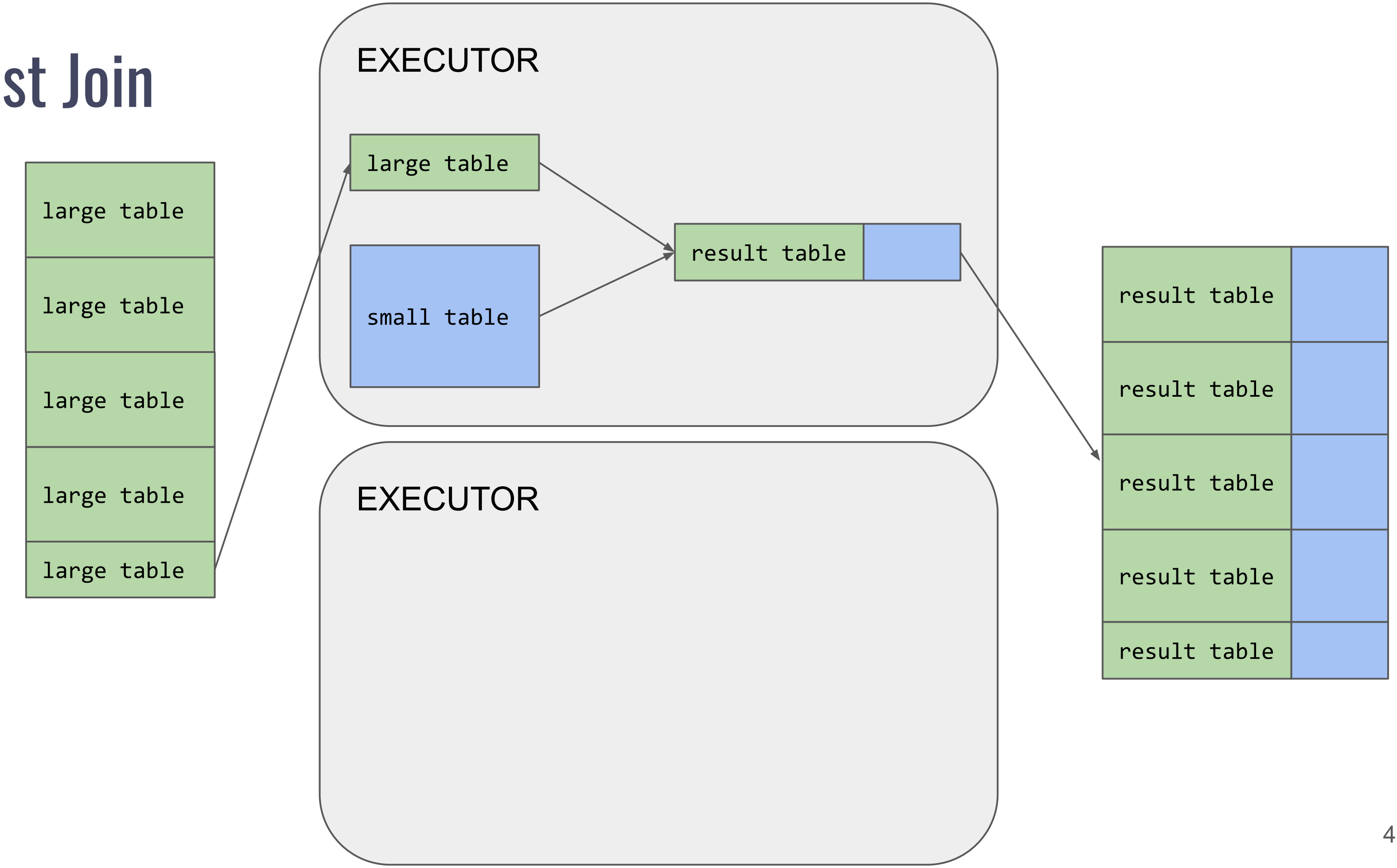
Broadcast Join



Broadcast Join



Broadcast Join



Broadcast join

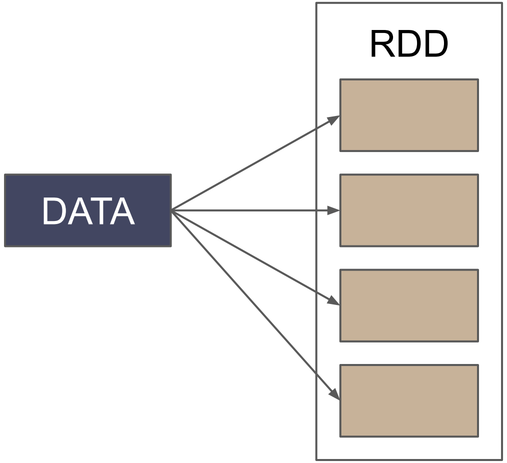
- no shuffle
- the small table is sent (broadcast) to all executor
- the small table must fit in executor memory
- broadcast join is automatic but it’s better to specify explicitly
- spark.sql.autoBroadcastJoinThreshold = 10Mb

```
df1.join(broadcast(df2), Seq("column1"))
```

Tasks, Stages & Jobs

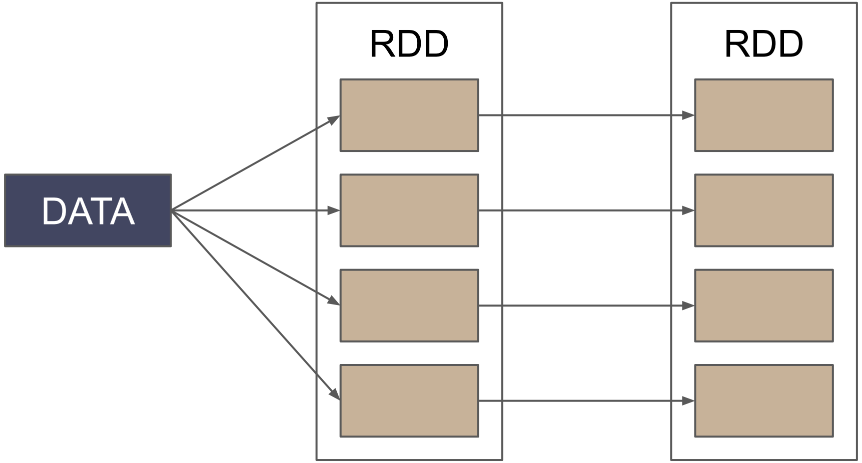
Task and Stage

```
val wc = sc.textFile(myData)
```



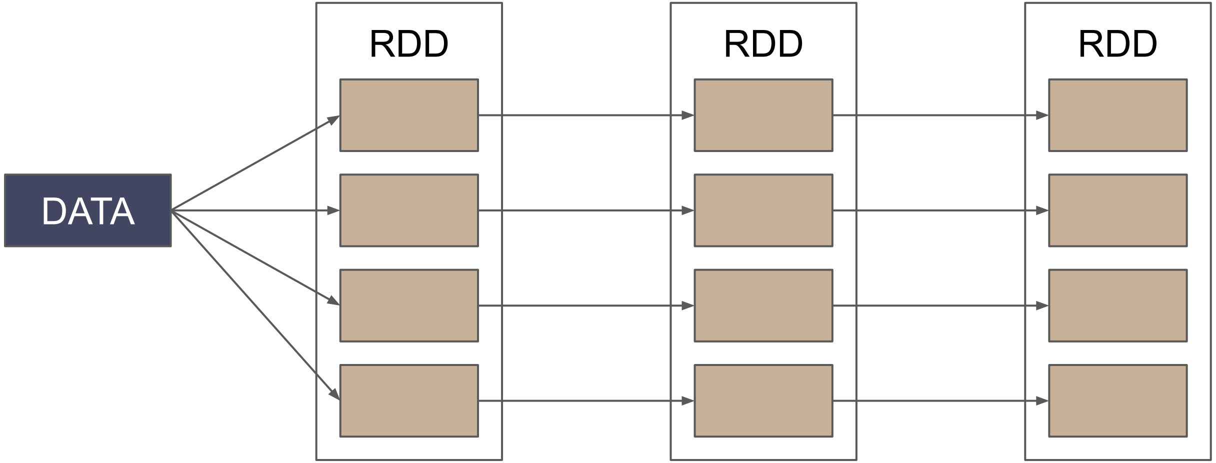
Task and Stage

```
val wc = sc.textFile(myData)
       .map(_.split(“,”))
```



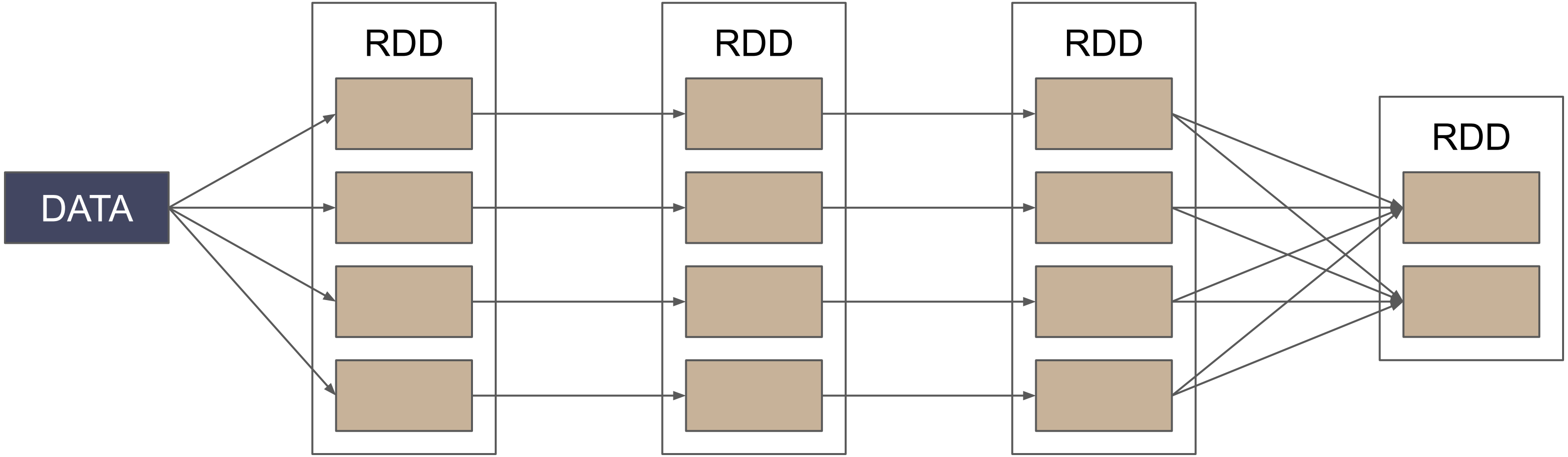
Task and Stage

```
val wc = sc.textFile(myData)
       .map(_.split(“,”))
       .map(row => (row(0), (row(14), 1)))
```



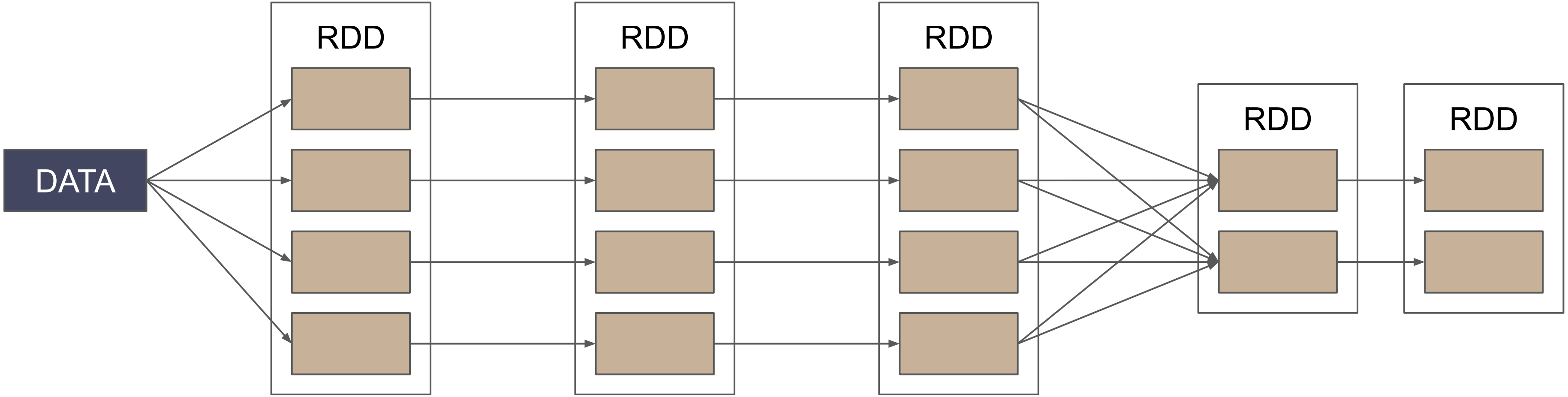
Task and Stage

```
val wc = sc.textFile(myData)
       .map(_.split(“,”))
       .map(row => (row(0), (row(14), 1)))
       .reduceByKey( (x,y) => (x._1 + y._1, x._2 + y._2))
```



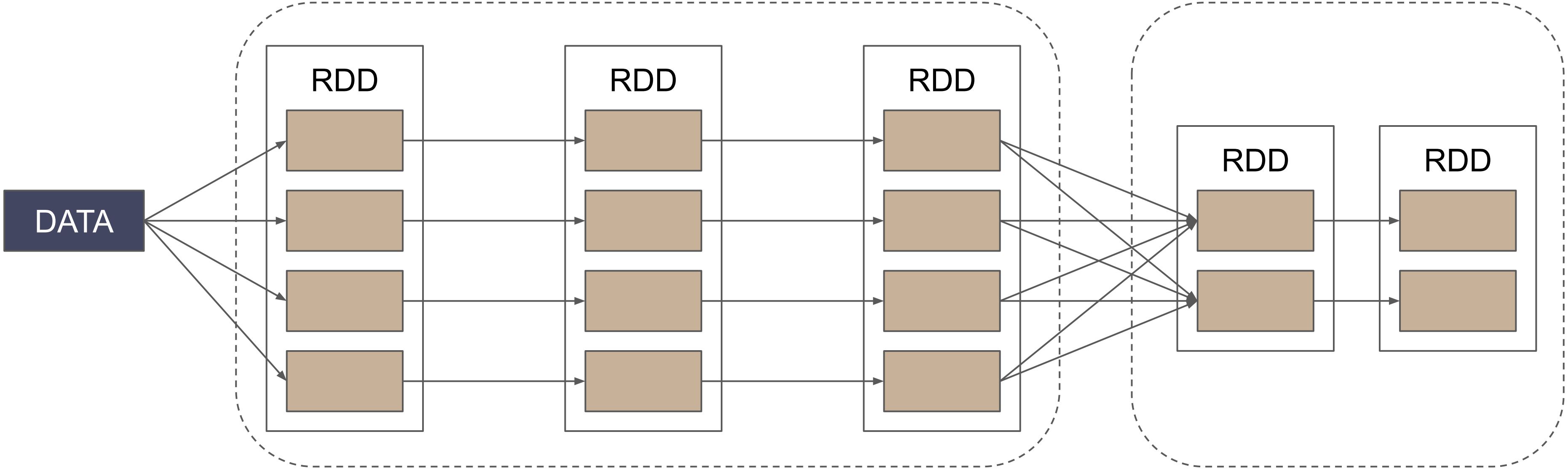
Task and Stage

```
val wc = sc.textFile(myData)
       .map(_.split(“,”))
       .map(row => (row(0), (row(14), 1)))
       .reduceByKey( (x,y) => (x._1 + y._1, x._2 + y._2))
       .mapValues { case (dataSum, cpt) => dataSum/cpt }
```



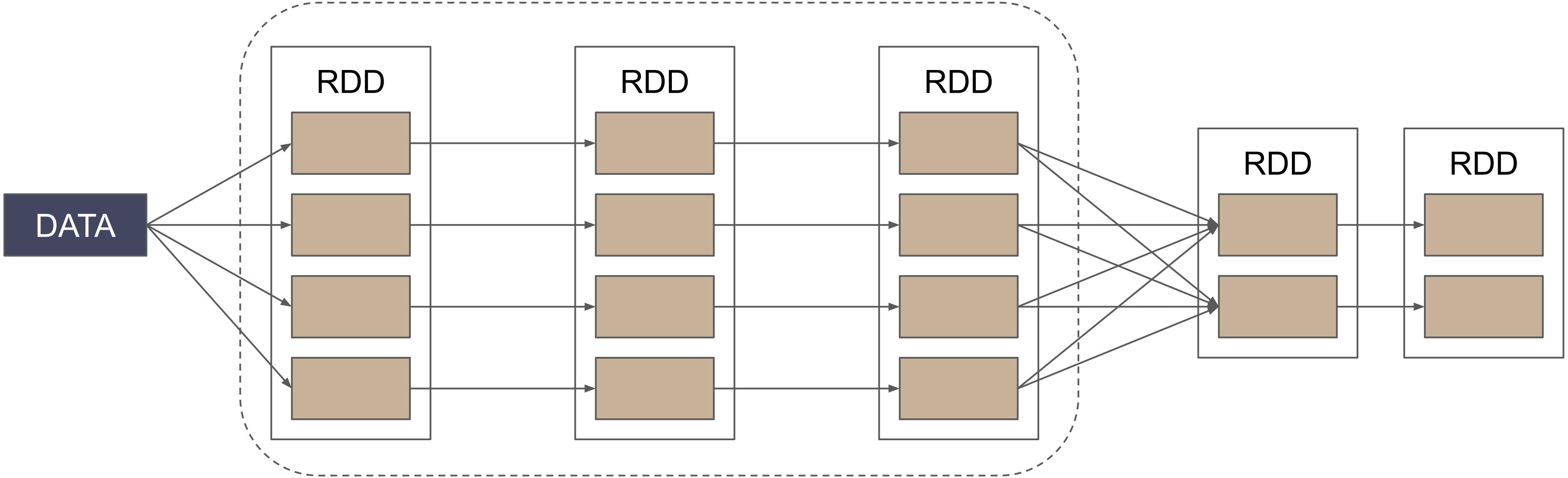
Task and Stage

```
val wc = sc.textFile(myData)
       .map(_.split(“,”))
       .map(row => (row(0), (row(14), 1)))
       .reduceByKey( (x,y) => (x._1 + y._1, x._2 + y._2))
       .mapValues { case (dataSum, cpt) => dataSum/cpt }
```



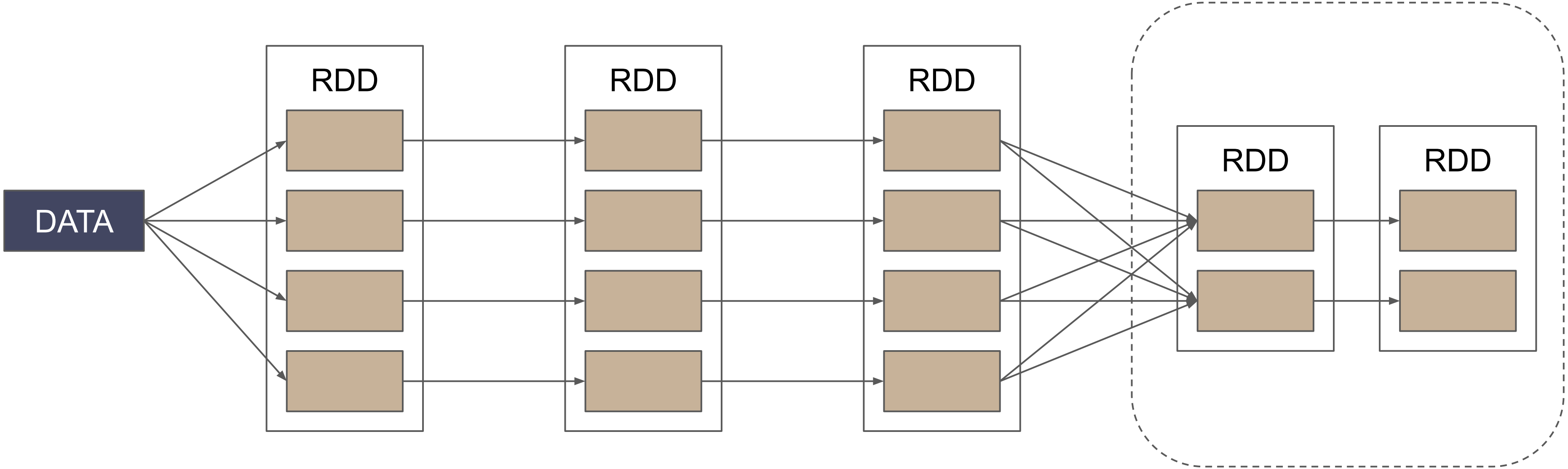
Task and Stage

```
val wc = sc.textFile(myData)
       .map(_.split(“,”))
       .map(row => (row(0), (row(14), 1)))
       .reduceByKey( (x,y) => (x._1 + y._1, x._2 + y._2))
       .mapValues { case (dataSum, cpt) => dataSum/cpt }
```



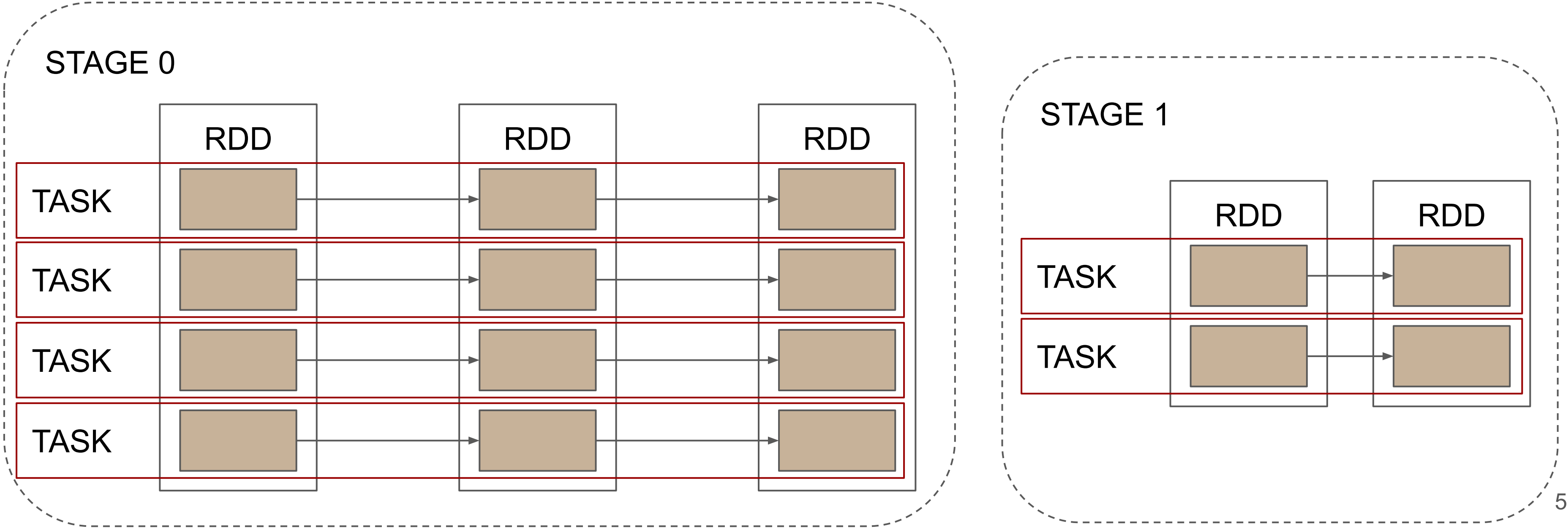
Task and Stage

```
val wc = sc.textFile(myData)
       .map(_.split(“,”))
       .map(row => (row(0), (row(14), 1)))
       .reduceByKey( (x,y) => (x._1 + y._1, x._2 + y._2))
       .mapValues { case (dataSum, cpt) => dataSum/cpt }
```



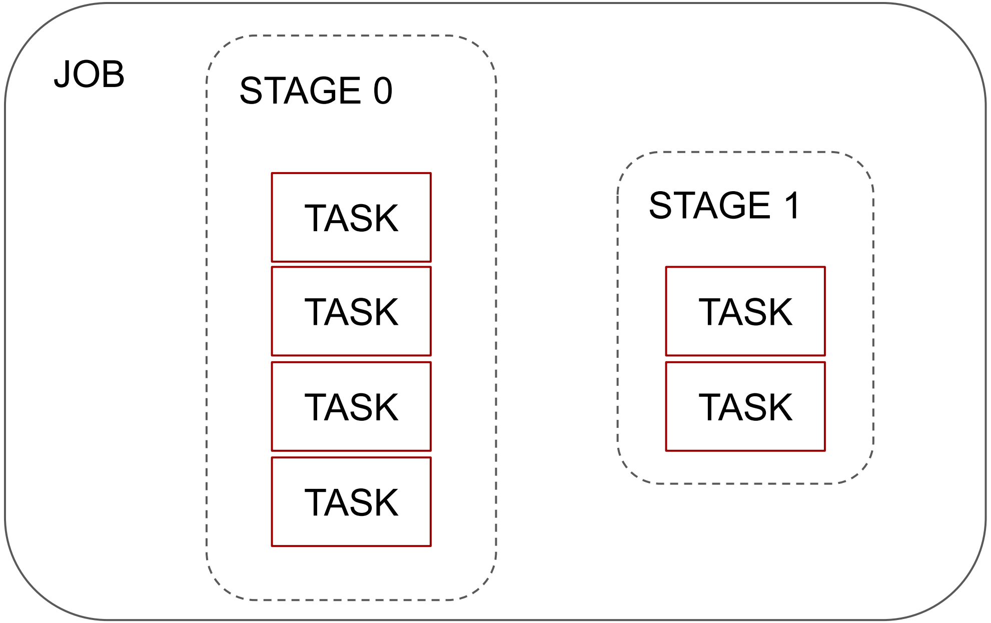
Task and Stage

```
val wc = sc.textFile(myData)
       .map(_.split(“,”))
       .map(row => (row(0), (row(14), 1)))
       .reduceByKey( (x,y) => (x._1 + y._1, x._2 + y._2))
       .mapValues { case (dataSum, cpt) => dataSum/cpt }
```

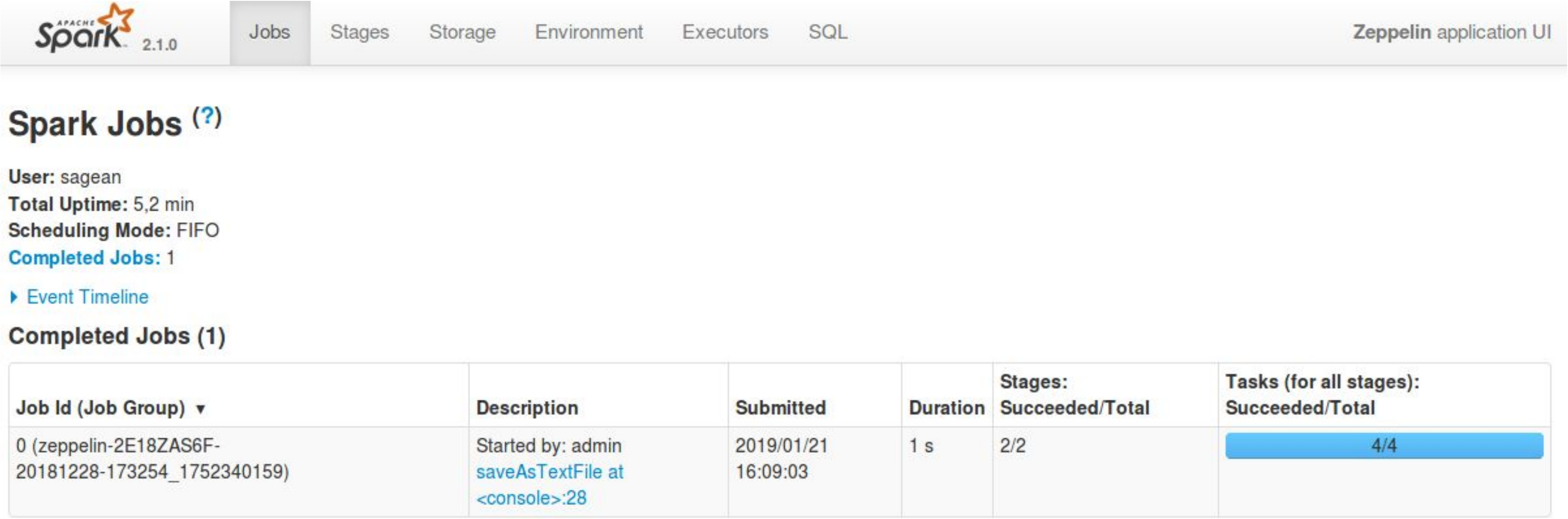


Task and Stage

```
val wc = sc.textFile(myData)
       .map(_.split(“,”))
       .map(row => (row(0), (row(14), 1)))
       .reduceByKey( (x,y) => (x._1 + y._1, x._2 + y._2))
       .mapValues { case (dataSum, cpt) => dataSum/cpt }
```



Spark UI: Jobs



Spark Jobs (?)

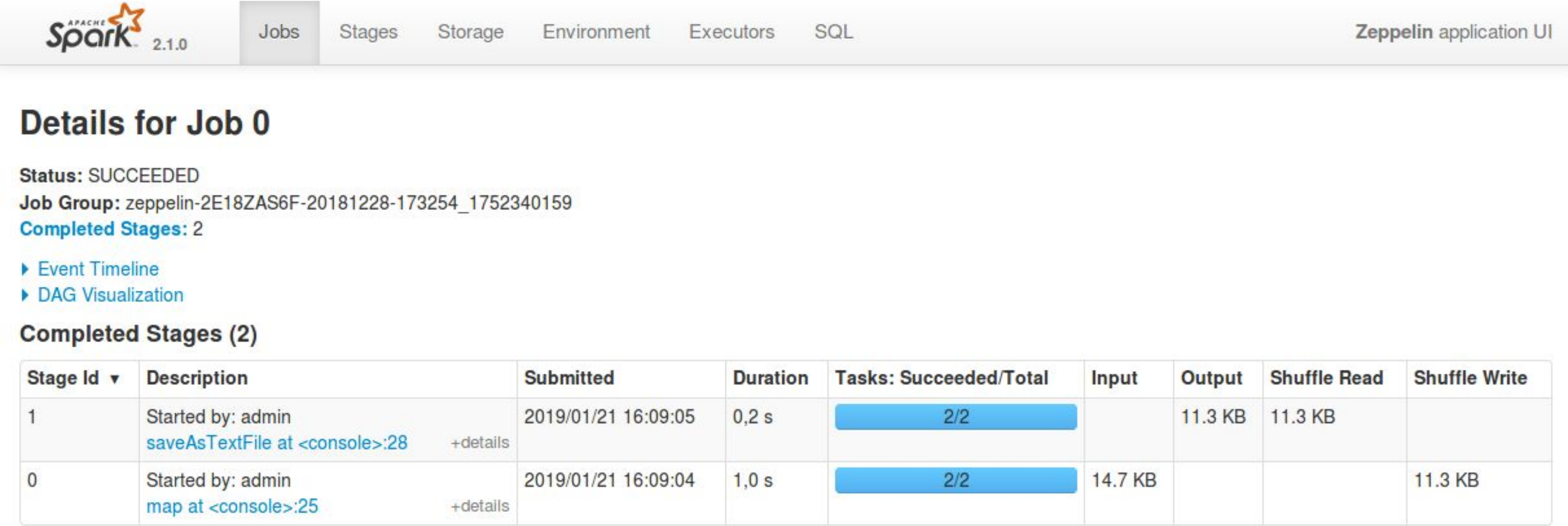
User: sagean
Total Uptime: 5,2 min
Scheduling Mode: FIFO
Completed Jobs: 1

▸ Event Timeline

Completed Jobs (1)

Job Id (Job Group) ▾	Description	Submitted	Duration	Stages: Succeeded/Total	Tasks (for all stages): Succeeded/Total
0 (zeppelin-2E18ZAS6F-20181228-173254_1752340159)	Started by: admin saveAsTextFile at <console>:28	2019/01/21 16:09:03	1 s	2/2	4/4

Spark UI: Stage



Details for Job 0

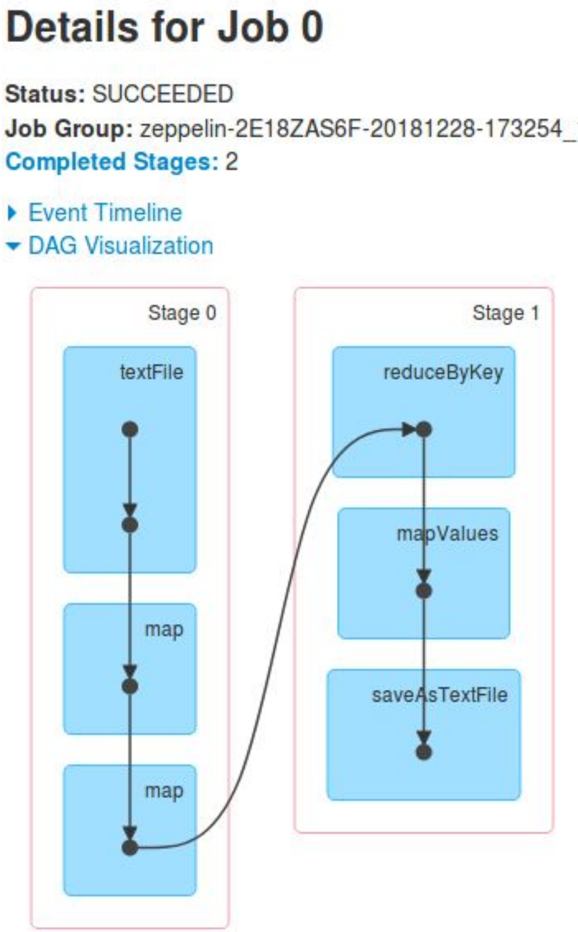
Status: SUCCEEDED
Job Group: zeppelin-2E18ZAS6F-20181228-173254_1752340159
Completed Stages: 2

- Event Timeline
- DAG Visualization

Completed Stages (2)

Stage Id ▾	Description	Submitted	Duration	Tasks: Succeeded/Total	Input	Output	Shuffle Read	Shuffle Write
1	Started by: admin saveAsTextFile at <console>:28 +details	2019/01/21 16:09:05	0,2 s	2/2		11.3 KB	11.3 KB	
0	Started by: admin map at <console>:25 +details	2019/01/21 16:09:04	1,0 s	2/2	14.7 KB			11.3 KB

Spark UI: Stage DAG



Details for Job 0

Status: SUCCEEDED

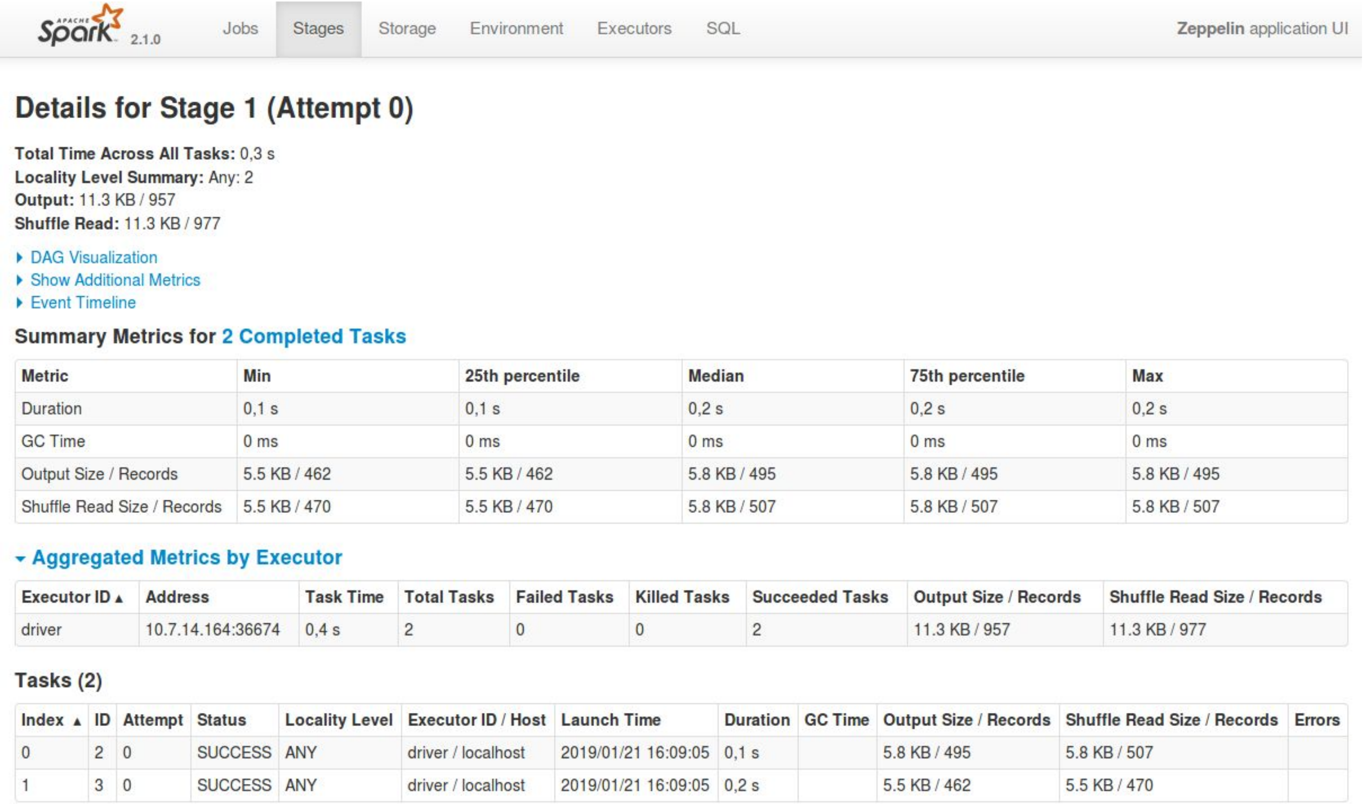
Job Group: zeppelin-2E18ZAS6F-20181228-173254_1752340159

Completed Stages: 2

▸ Event Timeline

▾ DAG Visualization

Spark UI: Tasks



Details for Stage 1 (Attempt 0)

Total Time Across All Tasks: 0,3 s
Locality Level Summary: Any: 2
Output: 11.3 KB / 957
Shuffle Read: 11.3 KB / 977

- DAG Visualization
- Show Additional Metrics
- Event Timeline

Summary Metrics for 2 Completed Tasks

Metric	Min	25th percentile	Median	75th percentile	Max
Duration	0,1 s	0,1 s	0,2 s	0,2 s	0,2 s
GC Time	0 ms	0 ms	0 ms	0 ms	0 ms
Output Size / Records	5.5 KB / 462	5.5 KB / 462	5.8 KB / 495	5.8 KB / 495	5.8 KB / 495
Shuffle Read Size / Records	5.5 KB / 470	5.5 KB / 470	5.8 KB / 507	5.8 KB / 507	5.8 KB / 507

Aggregated Metrics by Executor

Executor ID ▴	Address	Task Time	Total Tasks	Failed Tasks	Killed Tasks	Succeeded Tasks	Output Size / Records	Shuffle Read Size / Records
driver	10.7.14.164:36674	0,4 s	2	0	0	2	11.3 KB / 957	11.3 KB / 977

Tasks (2)

Index ▴	ID	Attempt	Status	Locality Level	Executor ID / Host	Launch Time	Duration	GC Time	Output Size / Records	Shuffle Read Size / Records	Errors
0	2	0	SUCCESS	ANY	driver / localhost	2019/01/21 16:09:05	0,1 s		5.8 KB / 495	5.8 KB / 507	
1	3	0	SUCCESS	ANY	driver / localhost	2019/01/21 16:09:05	0,2 s		5.5 KB / 462	5.5 KB / 470	

The End