
# Apache Spark - if only it worked

Marcin Szymaniuk
TantusData



# About

- Data Engineer and Consultant @TantusData
- Almost a decade of experience
- Have worked for Spotify, Apple, small startups

blog.explainmydata.com/2014/05/

# explain my data

Wednesday, May 28, 2014

## Spark should be better than MapReduce (if only it worked)

Spark is the user-friendly face of Big Data: a distributing programming framework which lets you write collection oriented algorithms in Scala that (theoretically) execute seamlessly across many machines. Spark has an elegant API (parallel collections with methods like map/reduce/groupByKey) that feels like programming

Data Wranglers

- Alex Rubinsteyn
- Brooks Paige
- Eason Liao
- Eric Garcia
- Russell Power
- Sergey Feldman



# Agenda

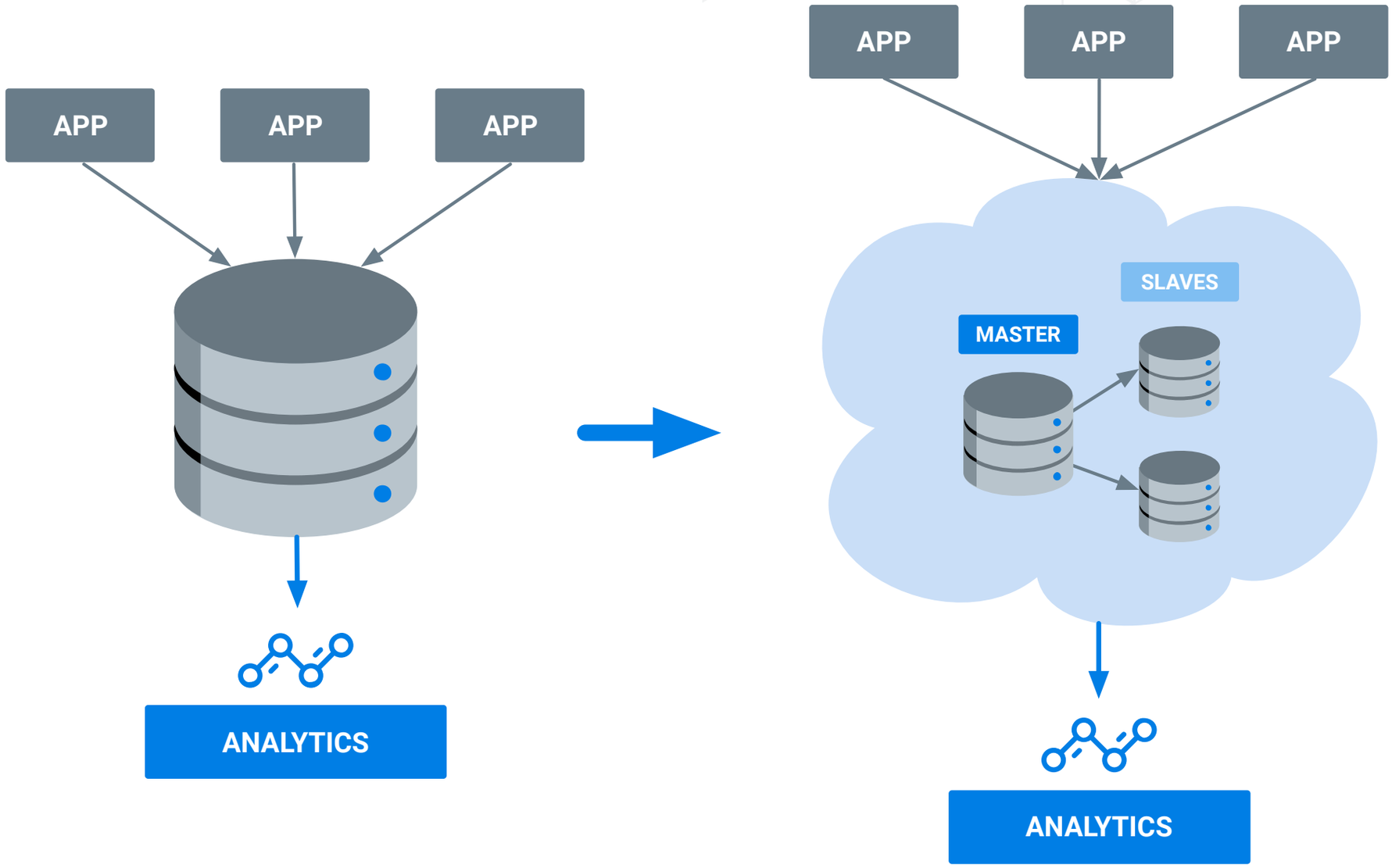
- Execution model
- Sizing executors
- Skewed data
- Locality
- Caching
- Testing and Debugging



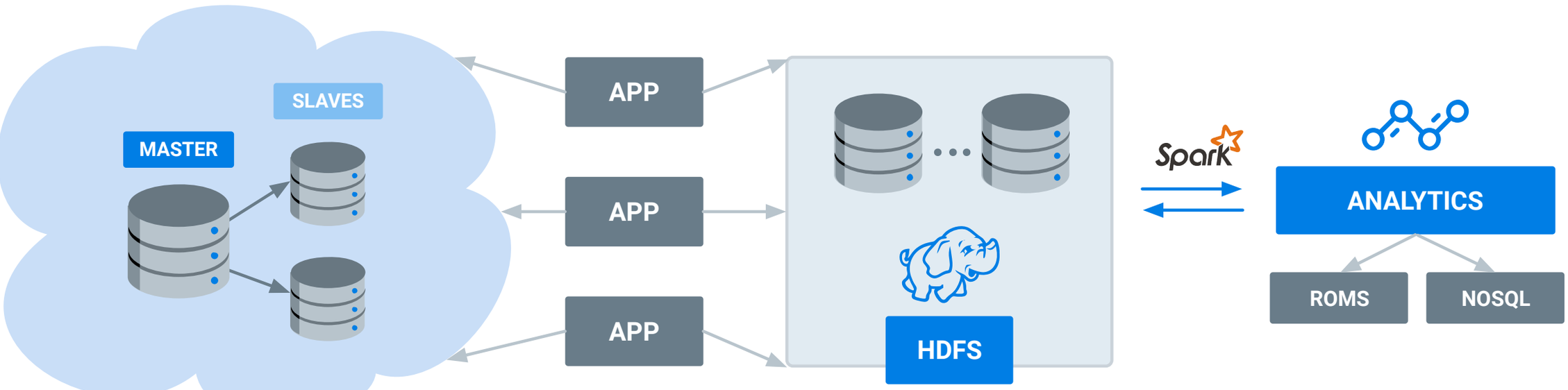
# What is Spark?

- Engine for distributed data processing
- Java, Scala, R, Python API
- SQL, Streaming, Machine Learning

# What is Spark? - use case



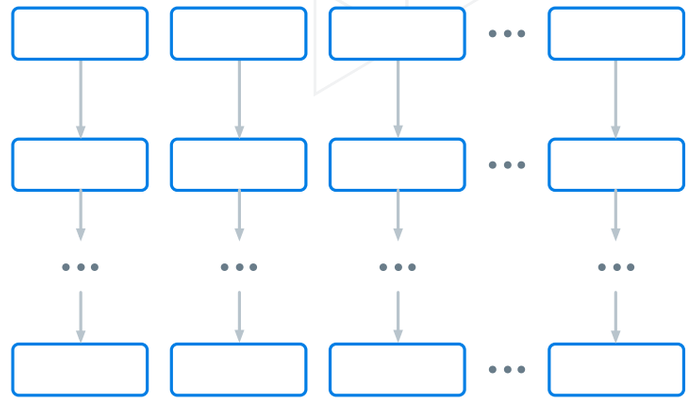
# What is Spark? - use case

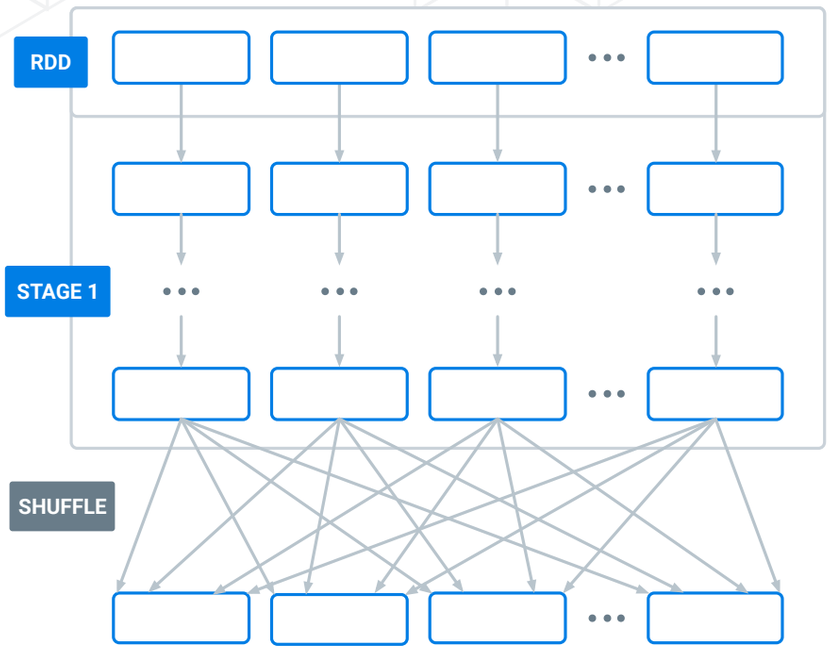


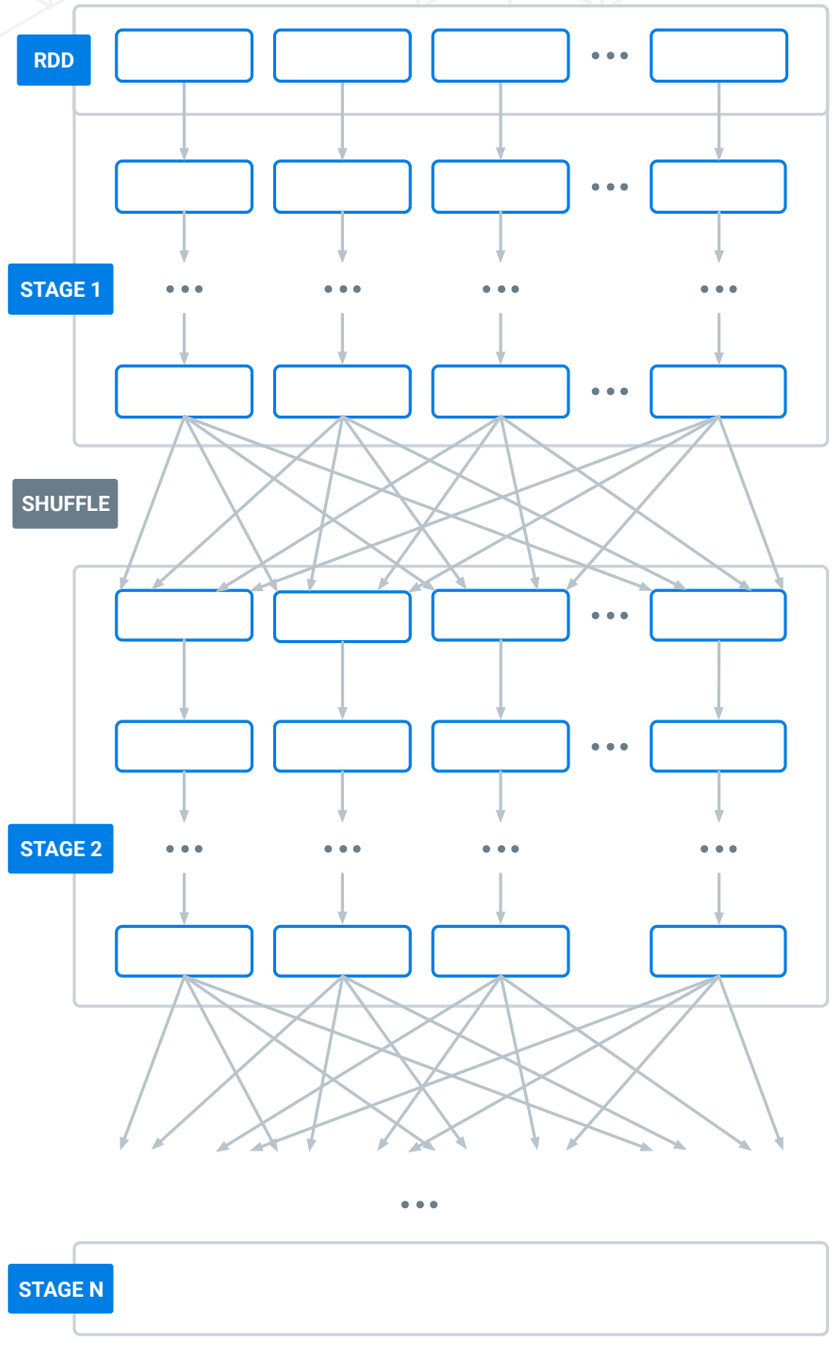


# Spark Execution Model


RDD




RDD
STAGE 1
SHUFFLE

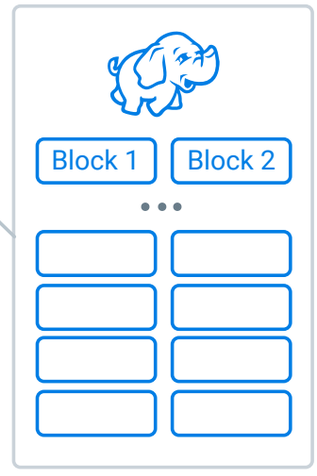

RDD
STAGE 1
SHUFFLE
STAGE 2
STAGE N

# Execution units - Tasks

STAGE 1

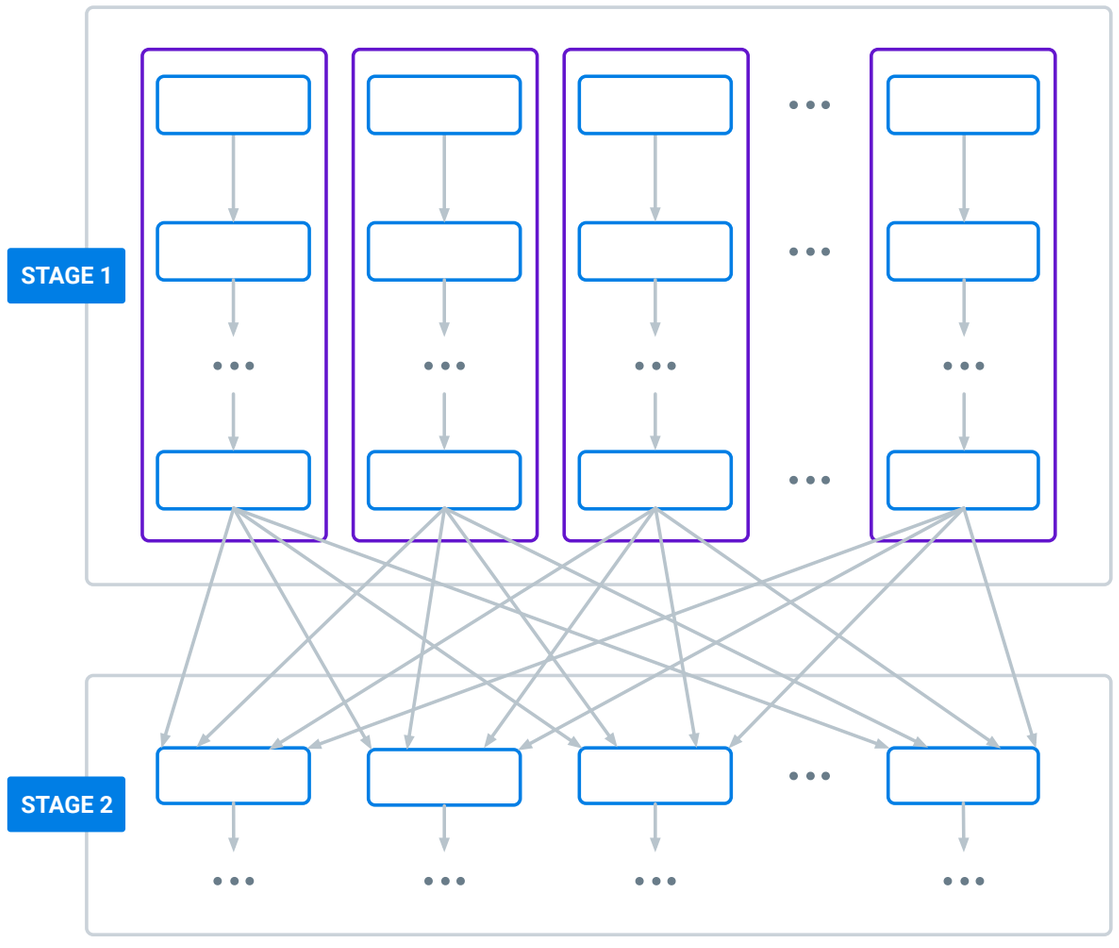
```
sc.textFile("/input/text/")
  .flatMap(line⇒line.split(" "))
  .map(word ⇒ (word, 1))
```

TASK 1




STAGE 2

```
.reduceByKey{case (x, y) ⇒ x + y}
.saveAsTextFile(output)
```





# Executors


Executor - JVM process able to run one or more tasks

Example config:

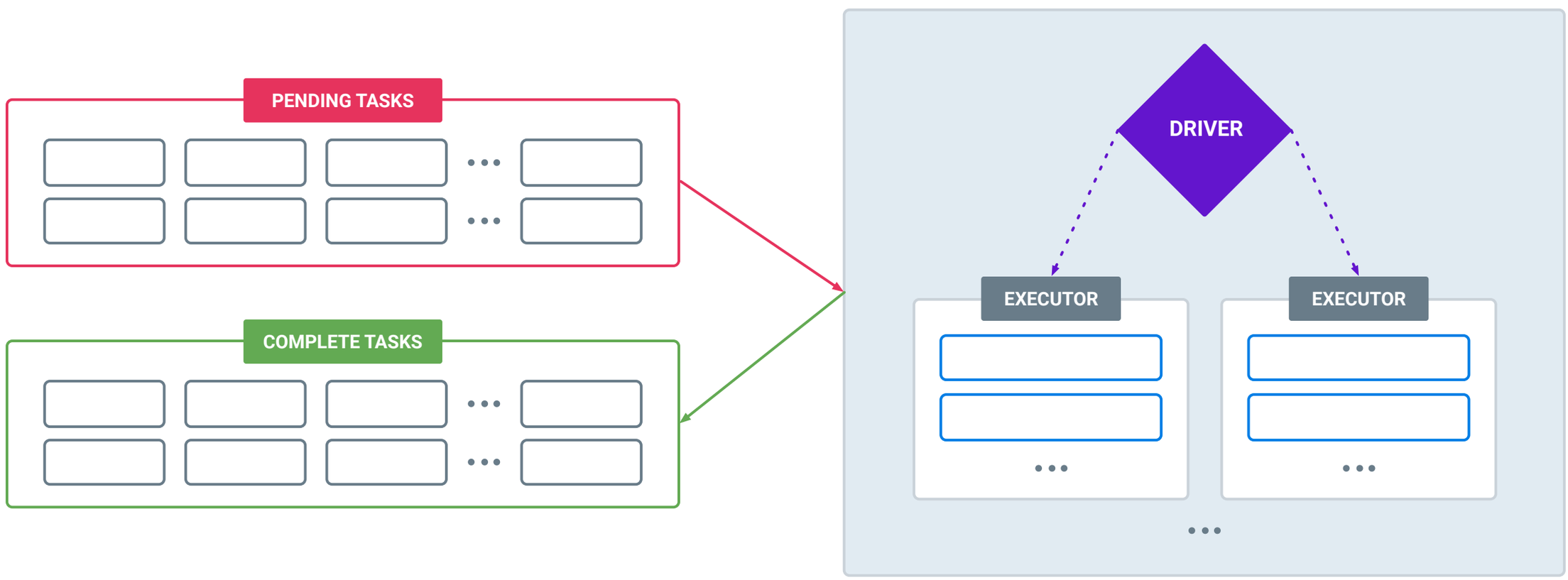
```
–executor-cores 3 –executor-memory 10g
```

# Executors

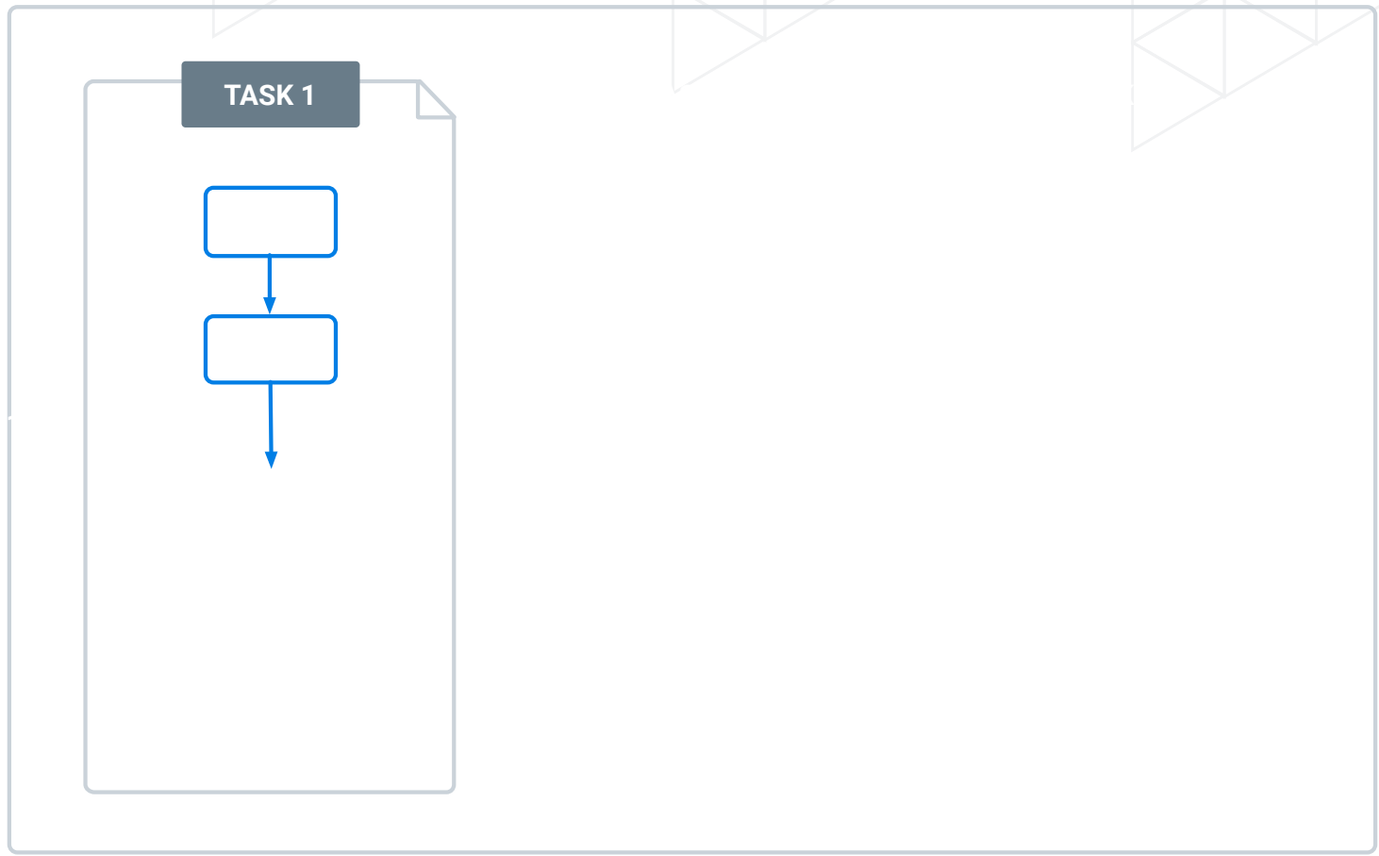

Executor - JVM process able to run one or more tasks

Example config:

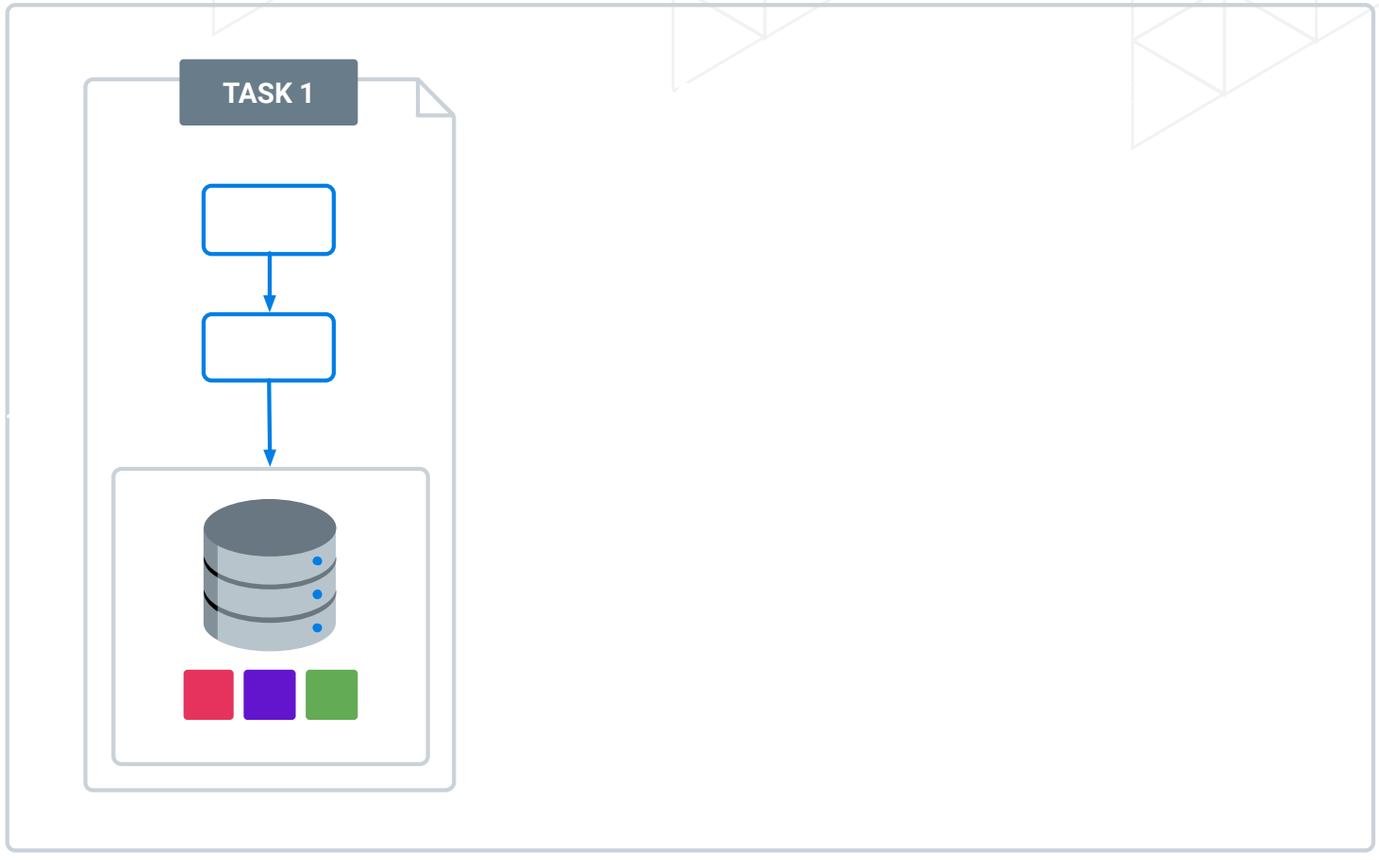
```
—executor-cores 3 —executor-memory 10g
```



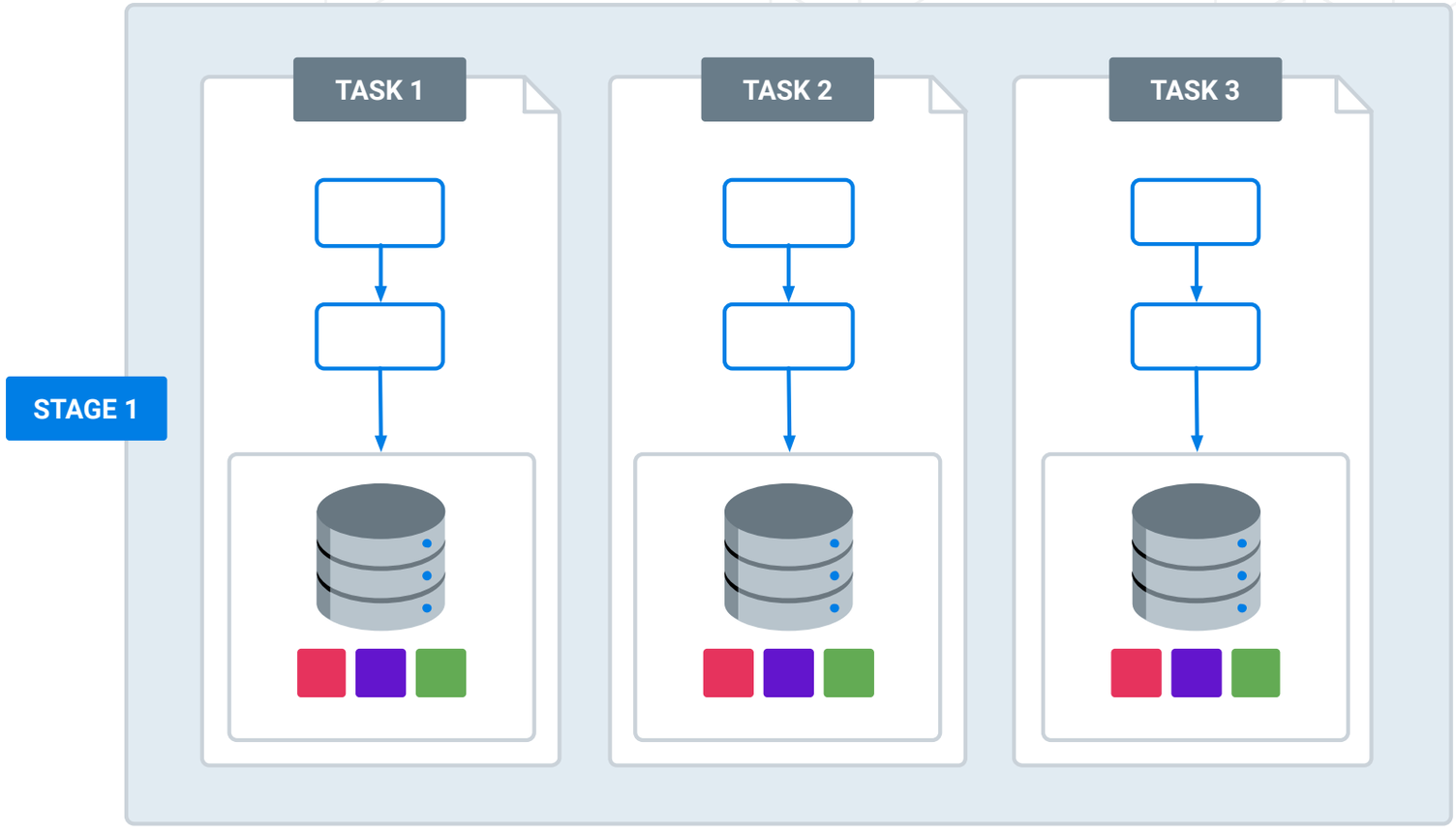
# Shuffle zoom-in



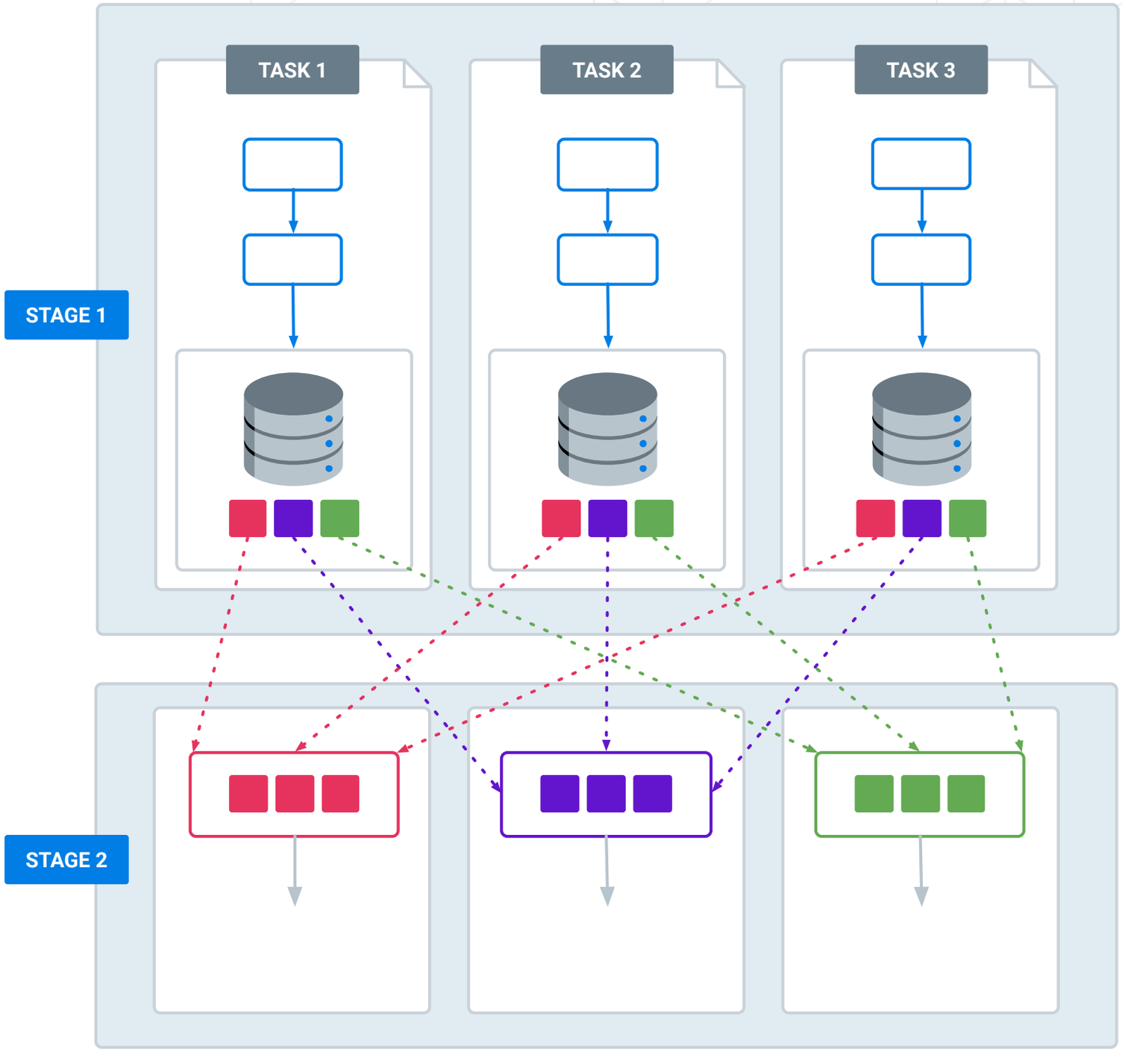
# Shuffle zoom-in



# Shuffle zoom-in



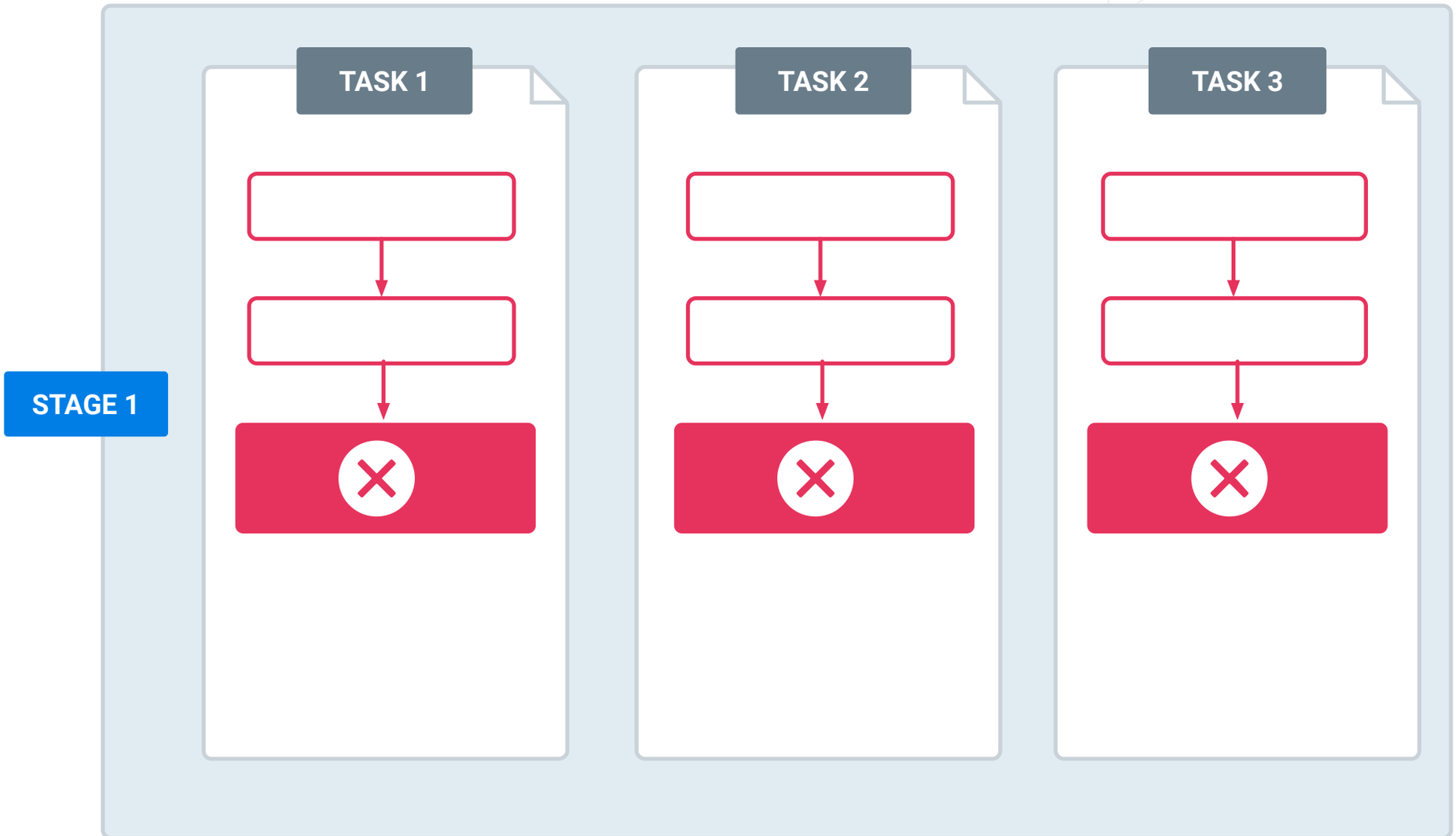
# Shuffle zoom-in



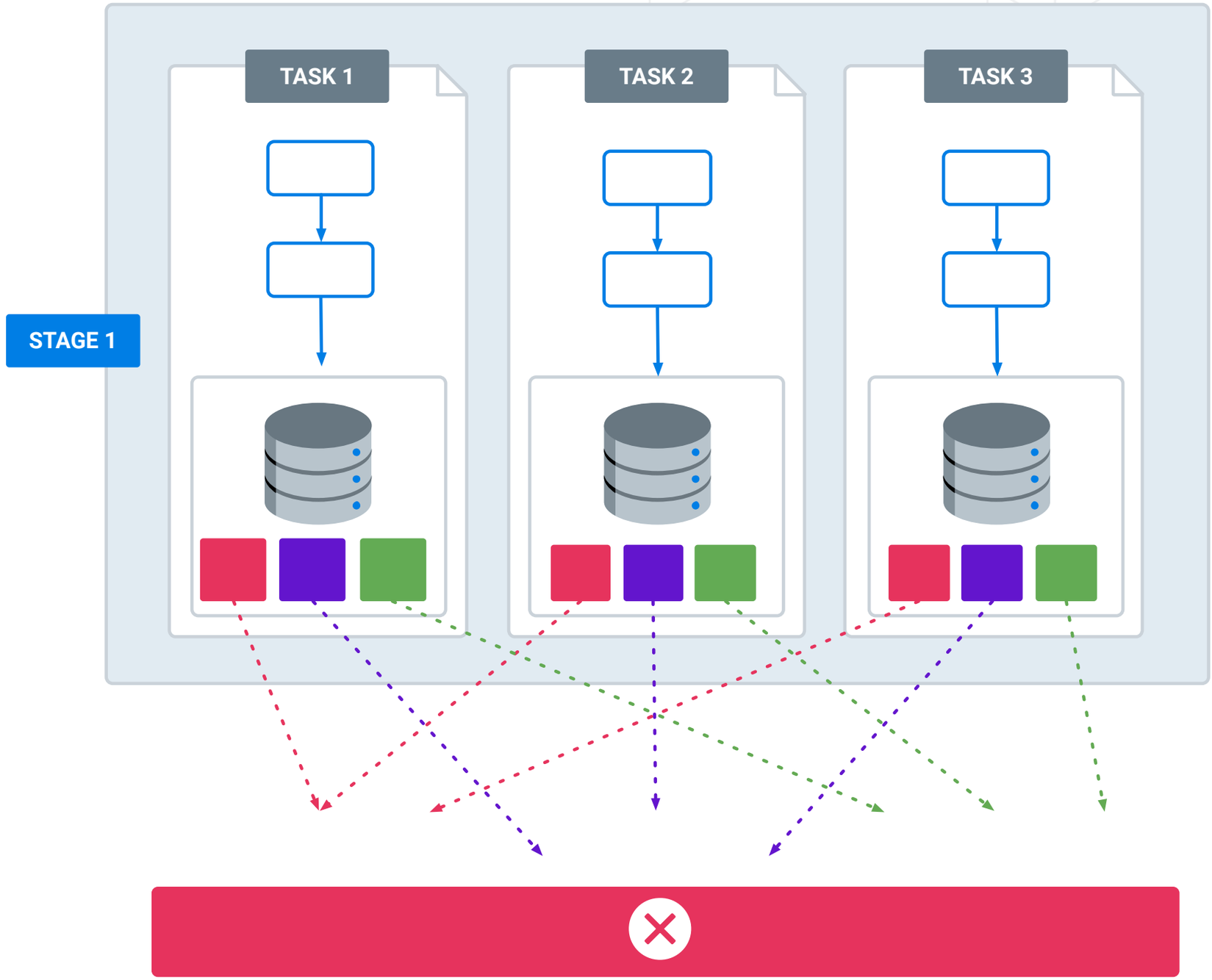
# We are up & running

- 2G block limit
- Timeouts
- GC overhead limit exceeded
- OOM
- ExecutorLostFailure

# Shuffle problems



# Shuffle problems





# We are up & running

What to watch out for:

- Failing tasks
- GC heavy tasks
- Shuffle read/write sizes
- Long running tasks

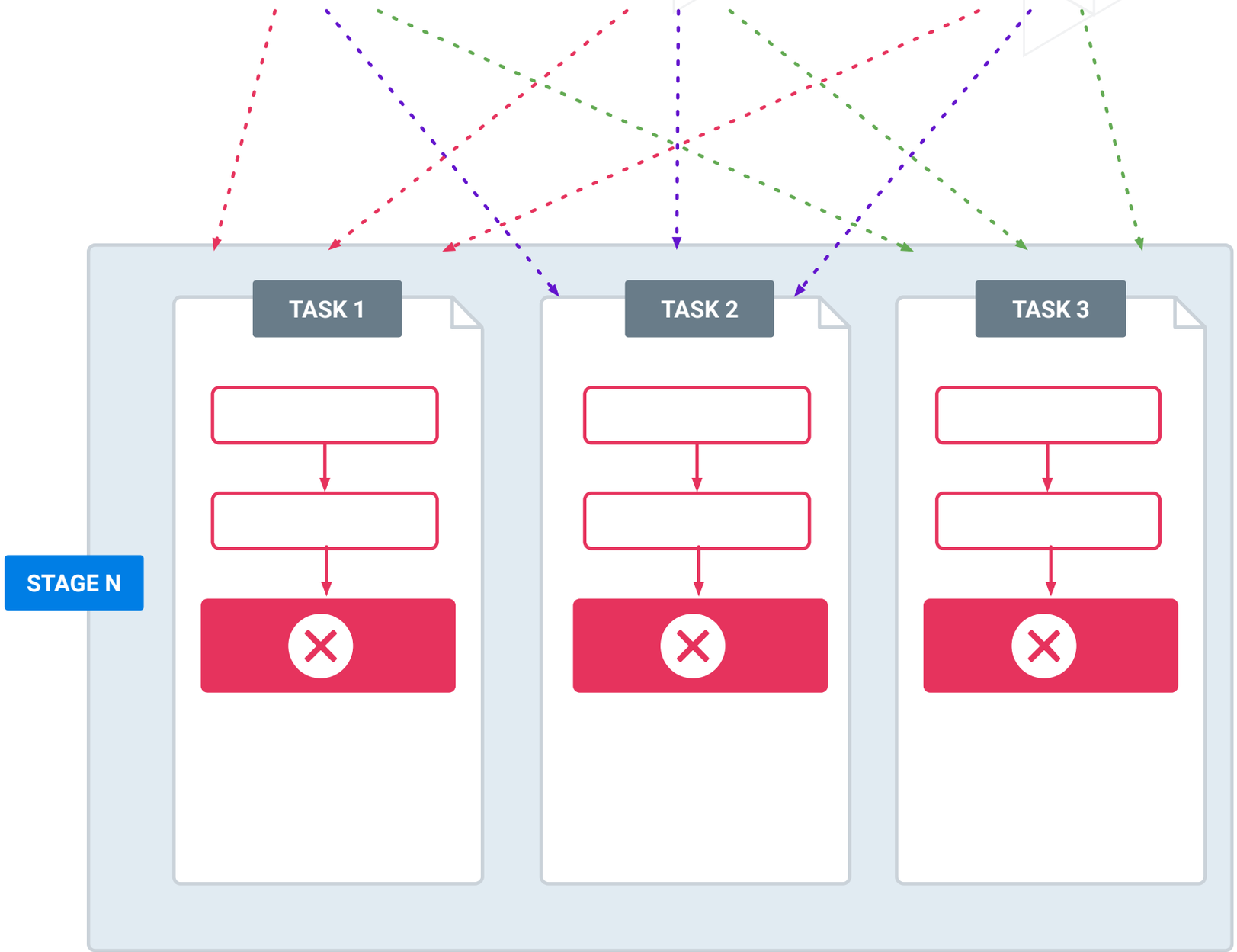
# Level of parallelism

- Too much data per task?
- Is parallelism level large enough?

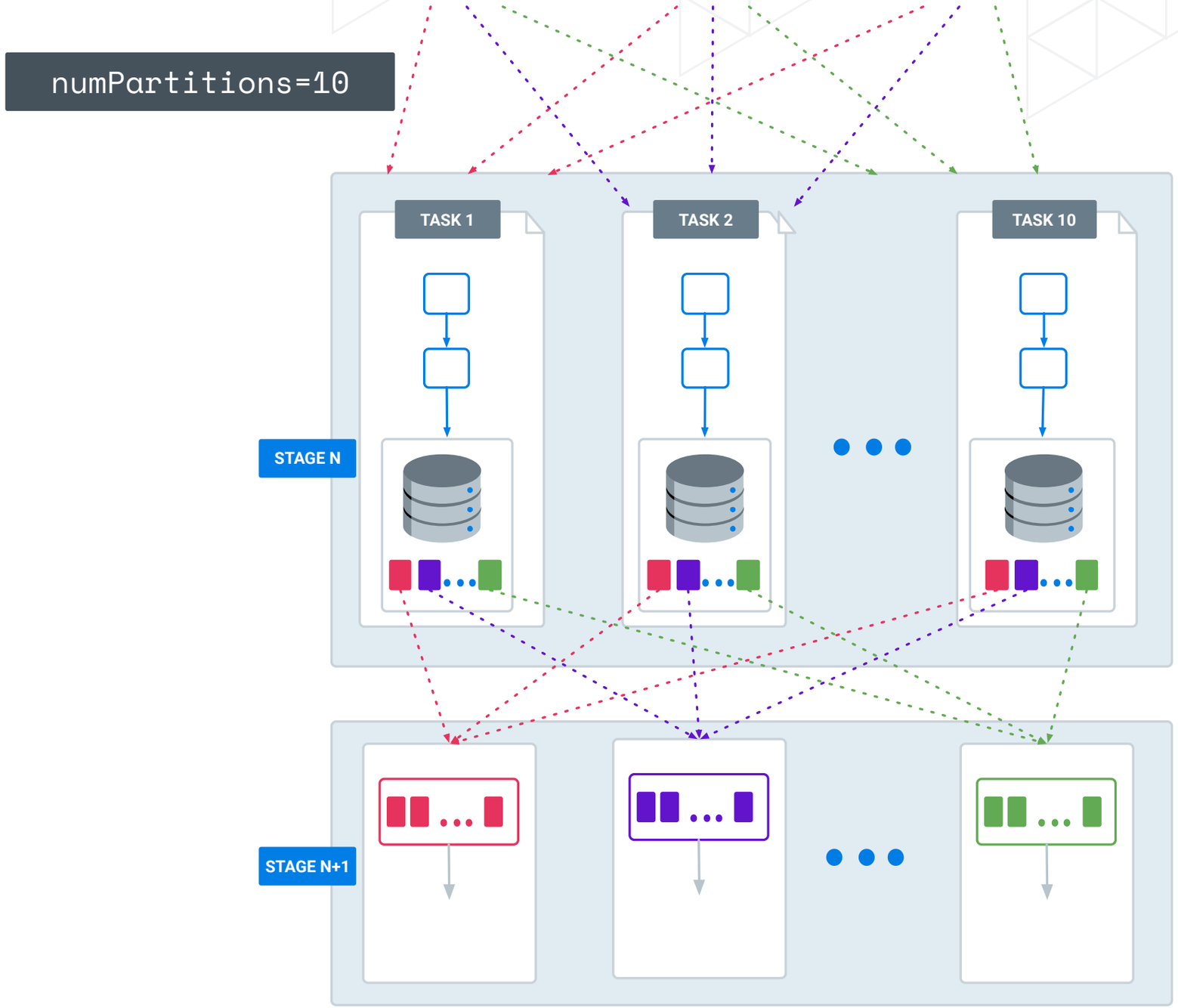
```
def groupByKey(numPartitions: Int)
def repartition(numPartitions: Int)
```

- More memory for executor

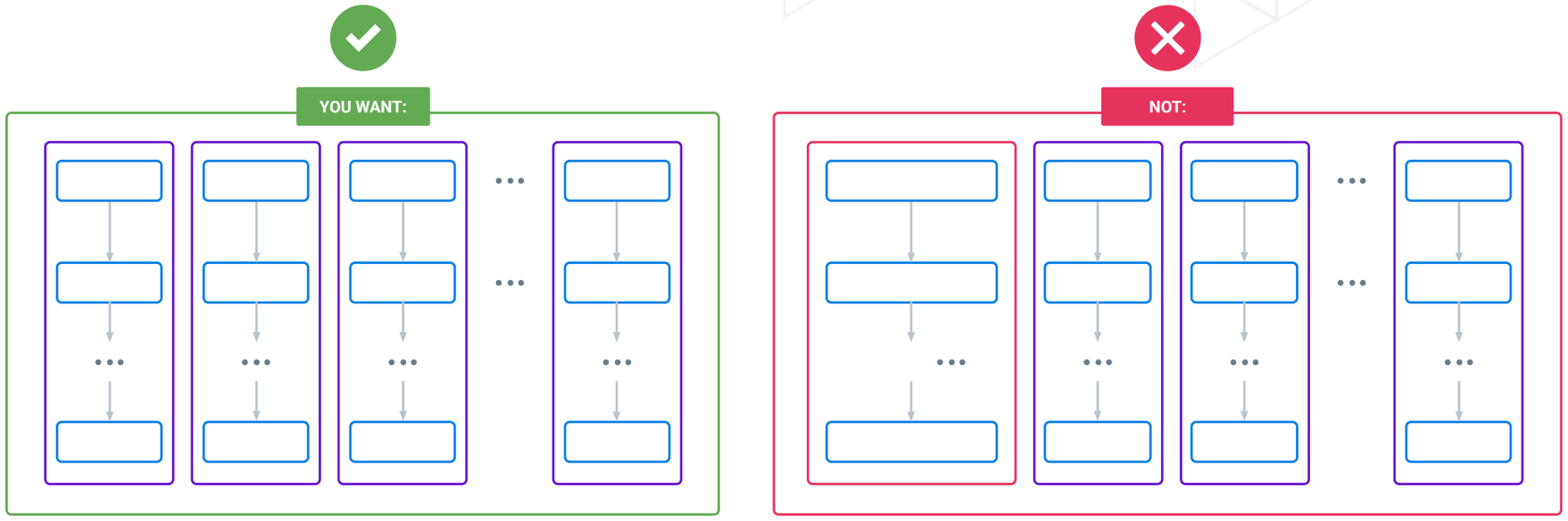
# Level of parallelism

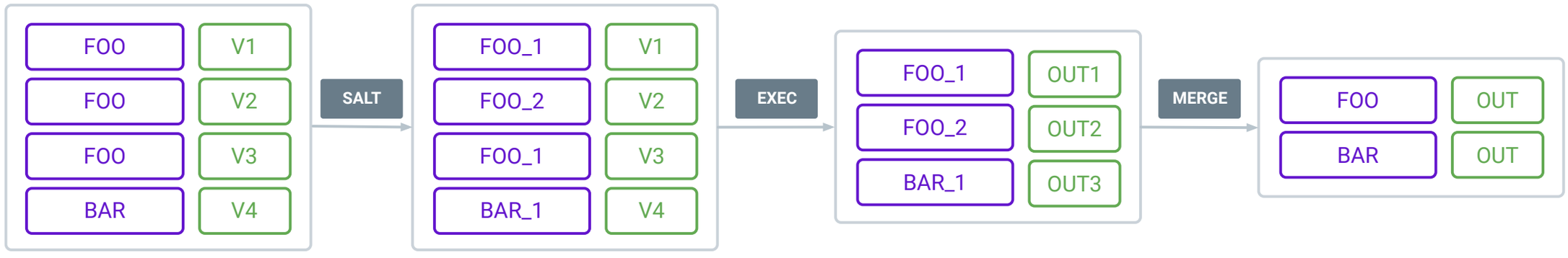


# Level of parallelism

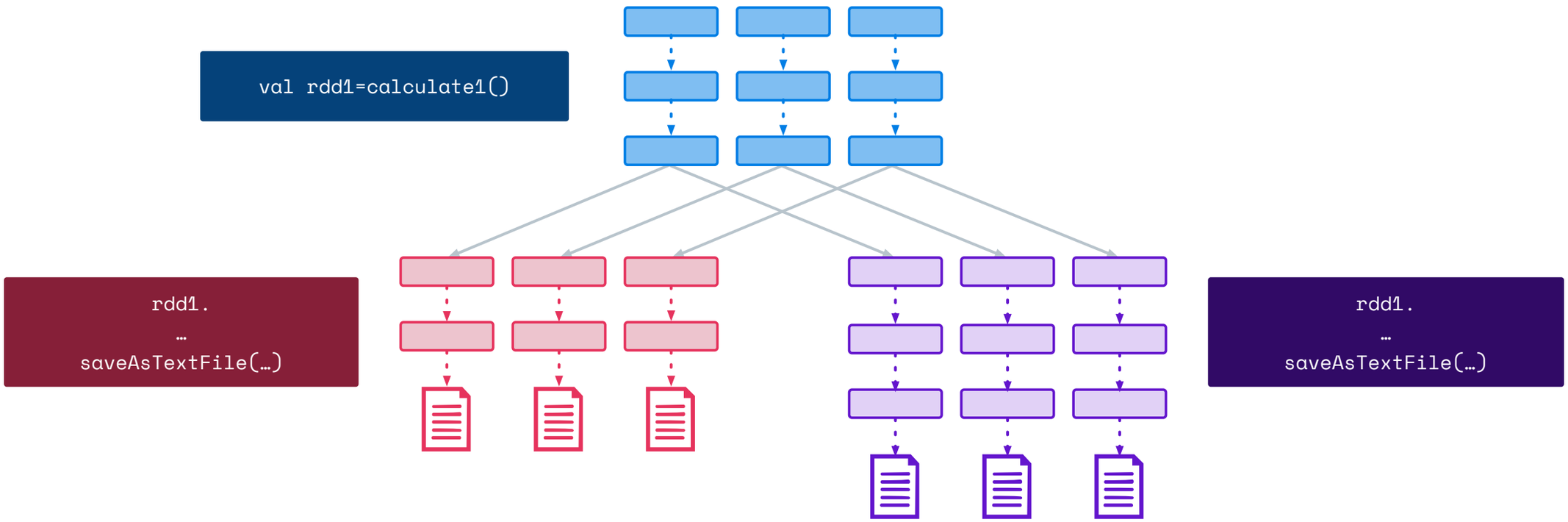


# Skewed data

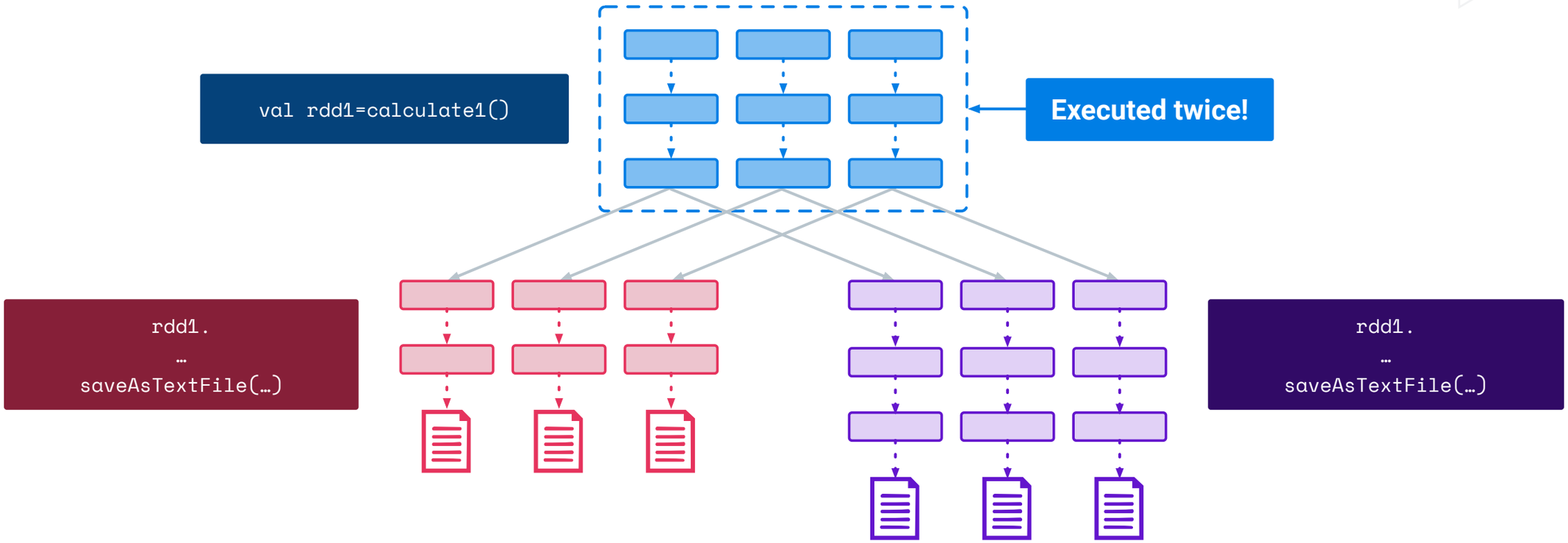

YOU WANT:
NOT:


FOO
V1
FOO
V2
FOO
V3
BAR
V4
SALT
FOO_1
V1
FOO_2
V2
FOO_1
V3
BAR_1
V4
EXEC
FOO_1
OUT1
FOO_2
OUT2
BAR_1
OUT3
MERGE
FOO
OUT
BAR
OUT
Key
Value

# Caching


val rdd1=calculate1()
rdd1.
…
saveAsTextFile(…)
rdd1.
…
saveAsTextFile(…)

# Caching


val rdd1=calculate1()
Executed twice!
rdd1.
…
saveAsTextFile(…)
rdd1.
…
saveAsTextFile(…)



# Caching

- Branch in execution plan is a candidate for caching
- Spark UI shows how much memory an RDD is taking
- You cannot control priority - it's LRU
- Whole partition must fit in memory!



# Caching

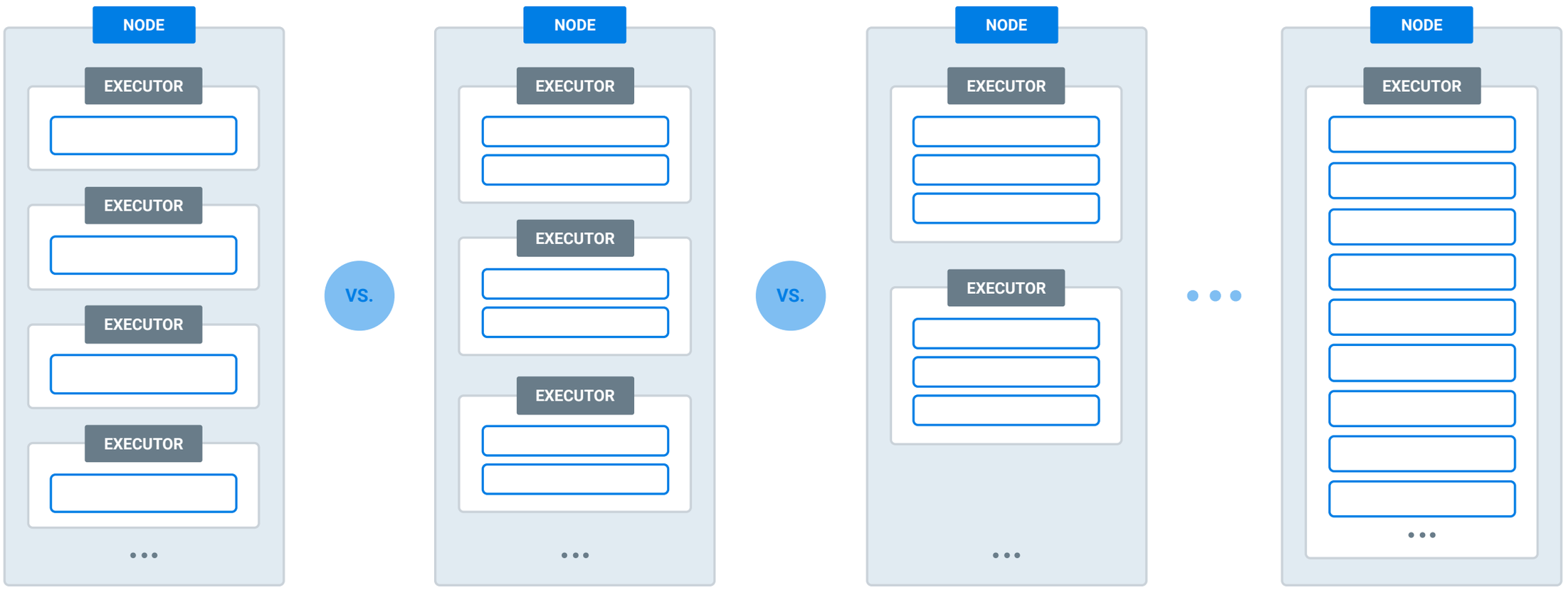
- Don't cache to disk if computation is cheap
- Caching with RF - only when recreation is extremely costly
- Checkpointing vs caching
- Shuffle data is automatically persisted

# Sizing executors


Executor - JVM process able to run one or more tasks

Example config:

```
—executor-cores 3 —executor-memory 10g
```

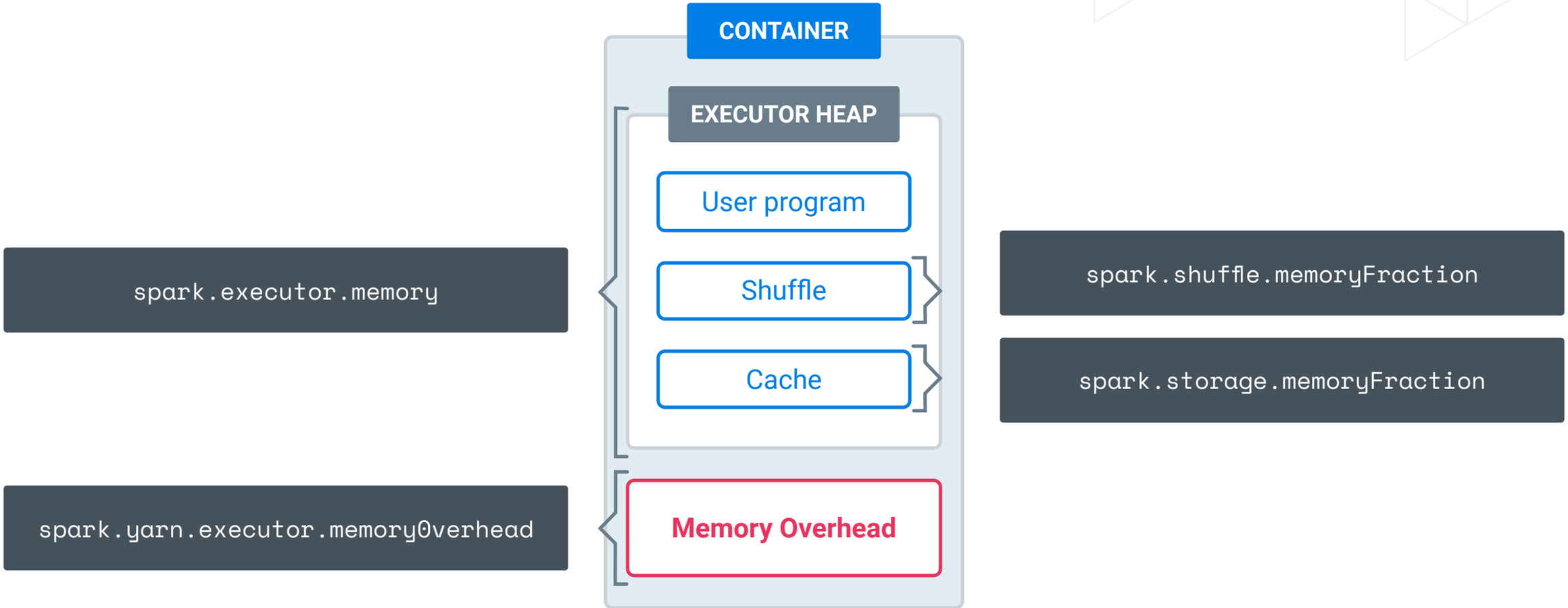




# Sizing executors

- Spark can benefit from running multiple tasks in the same JVM
- Many cores leads to GC problems
- Large executors might be not granular enough
- 1-4 CPUs should be good for start for ETL
- Play with large executors if you heavily rely on caching or broadcast variables

# Sizing executors

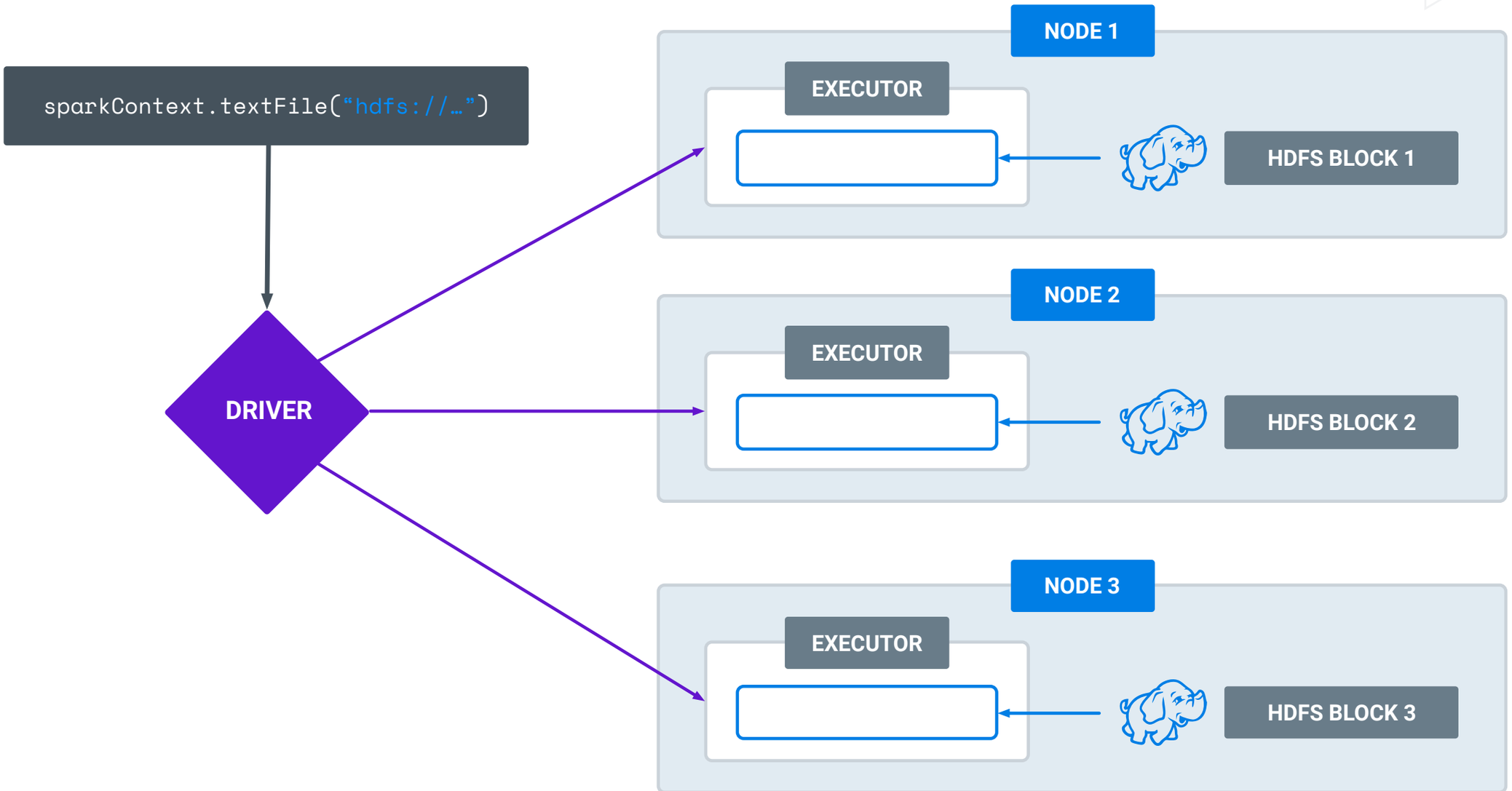




# Sizing executors

- Keep in mind memory overhead
- Keep some resources for OS
- Determine memory consumption - cache an RDD
- Consider dynamic resource allocation?

# Locality

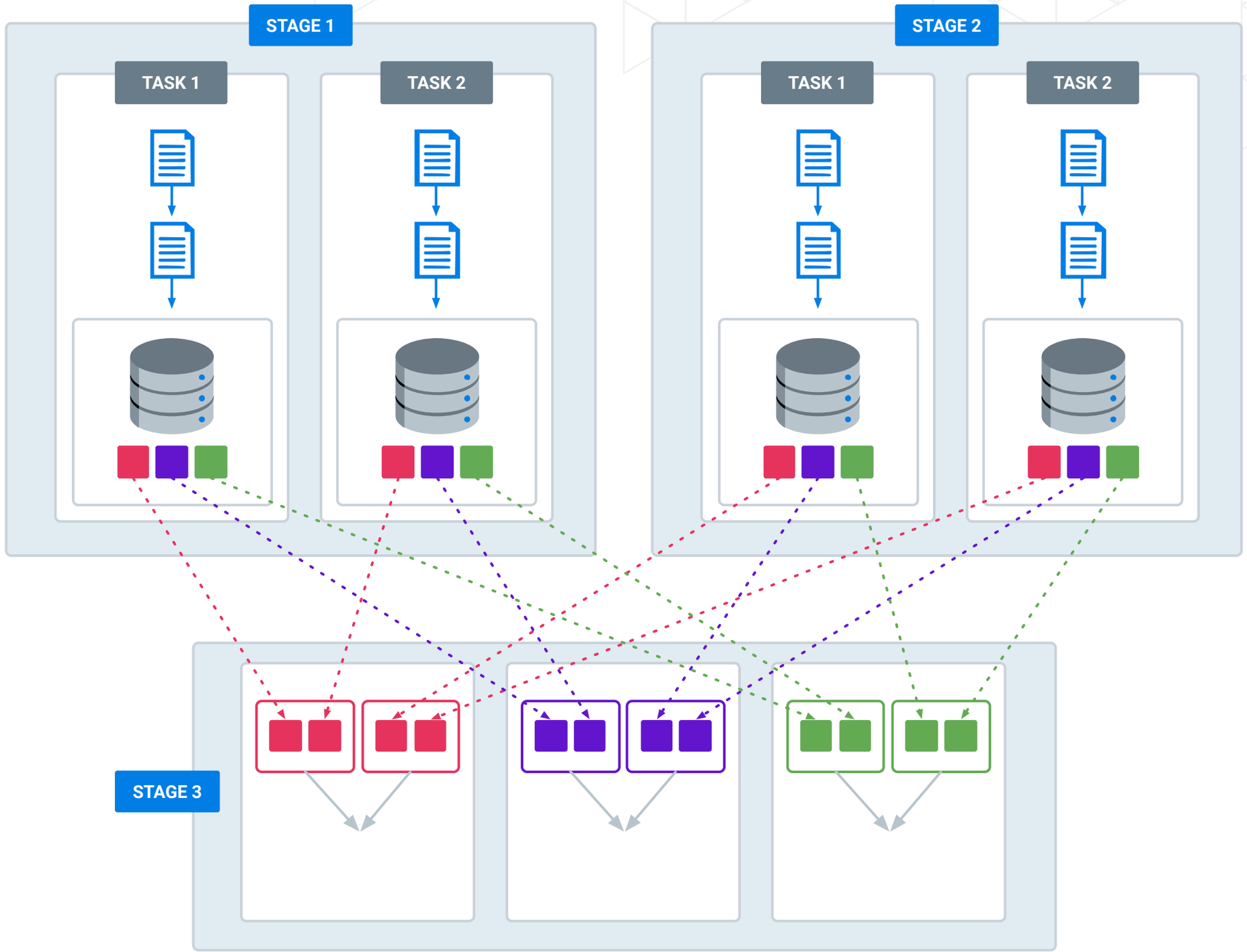

sparkContext.textFile("hdfs://…")
DRIVER
NODE 1
EXECUTOR
HDFS BLOCK 1
NODE 2
EXECUTOR
HDFS BLOCK 2
NODE 3
EXECUTOR
HDFS BLOCK 3

# Locality

- Increase number of executors
- For small jobs better to leave as is
- spark.locality.wait parameter

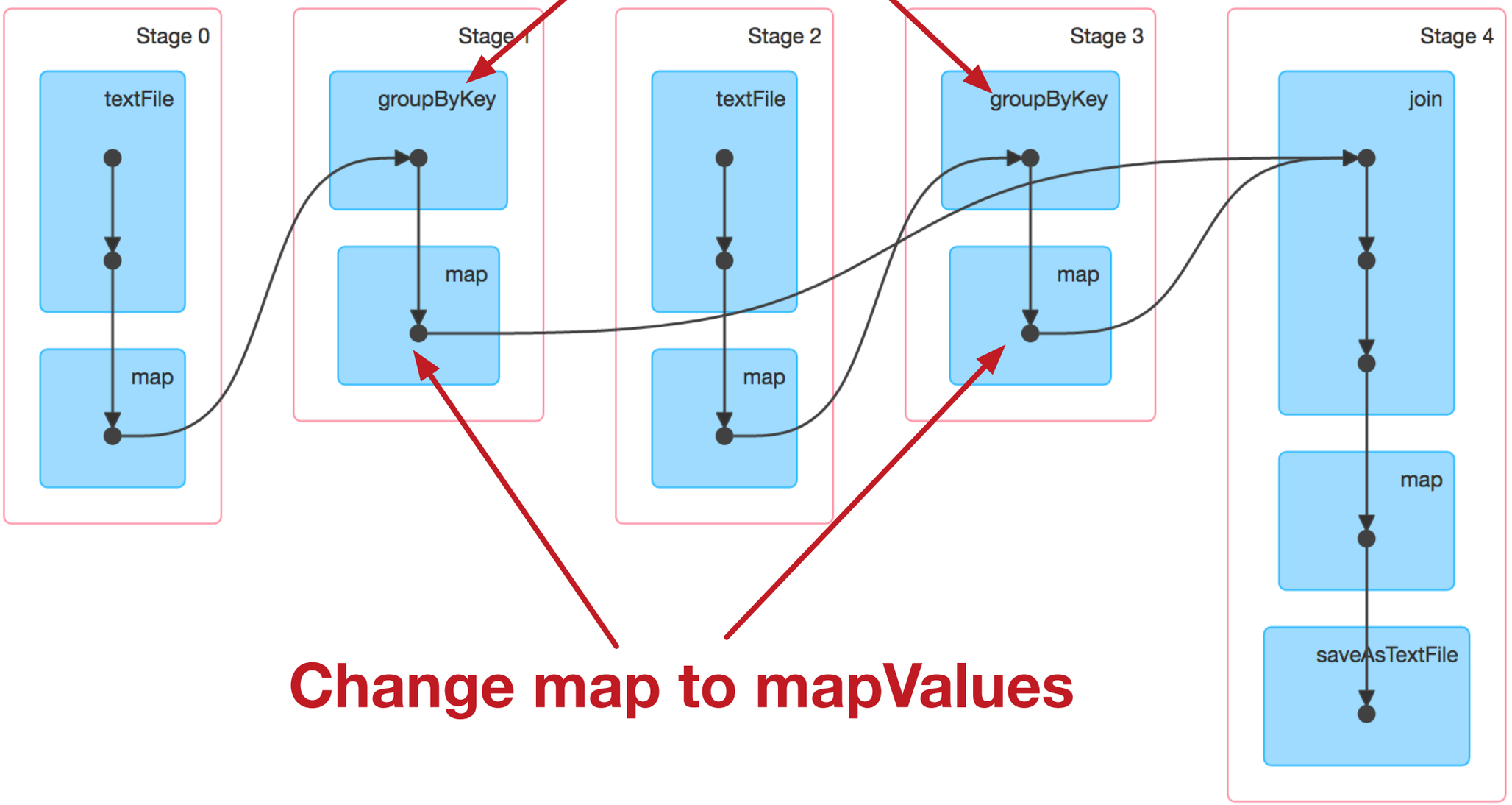
| Index ▲ | ID | Attempt | Status | Locality Level |
|---|---|---|---|---|
| 0 | 289 | 0 | SUCCESS | RACK_LOCAL |
| 1 | 3635 | 0 | SUCCESS | RACK_LOCAL |
| 2 | 3054 | 0 | SUCCESS | RACK_LOCAL |
| 3 | 1503 | 0 | SUCCESS | NODE_LOCAL |

# Join zoom-in



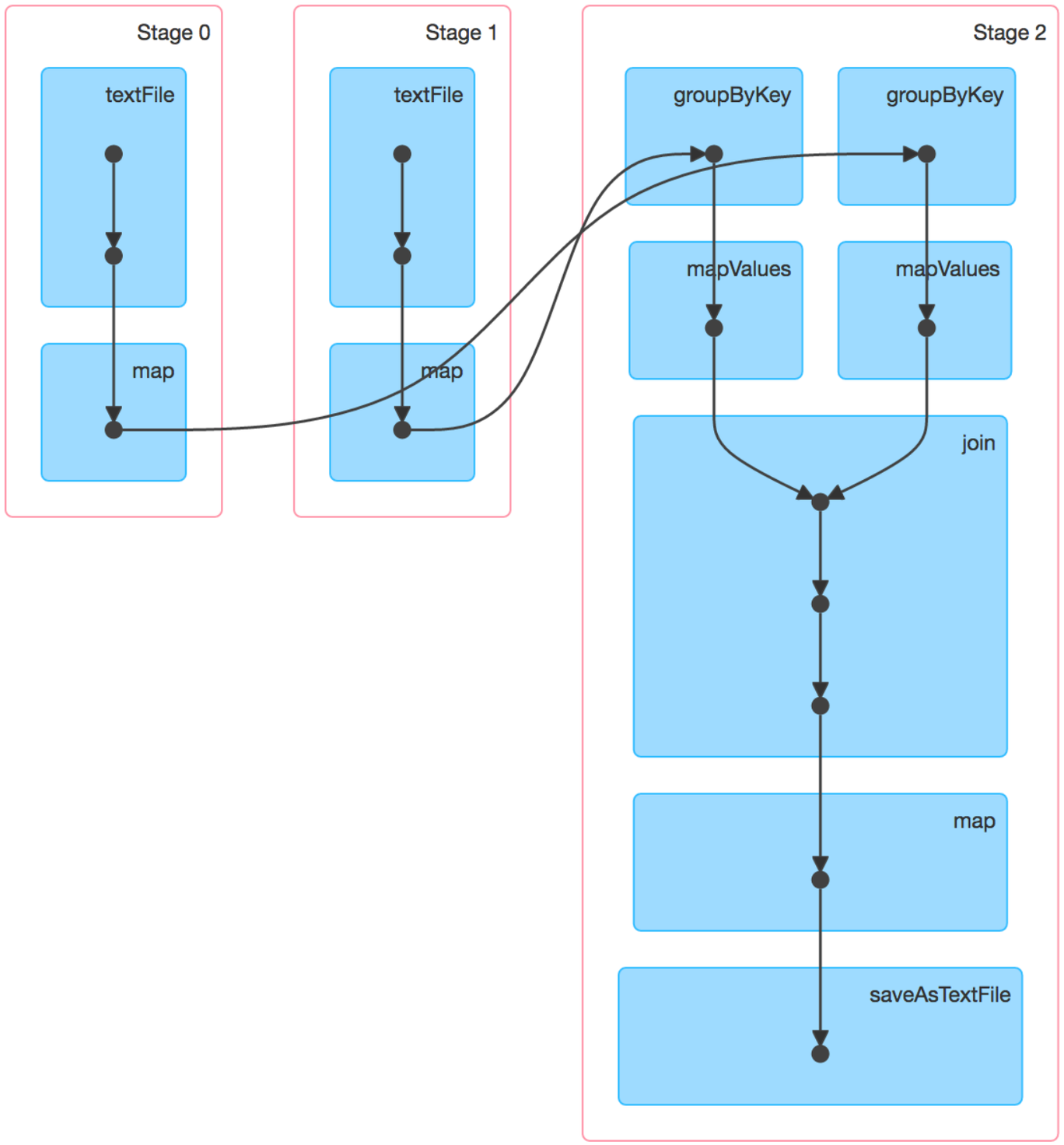
# Optimize shuffle

**Use the same number of partition**

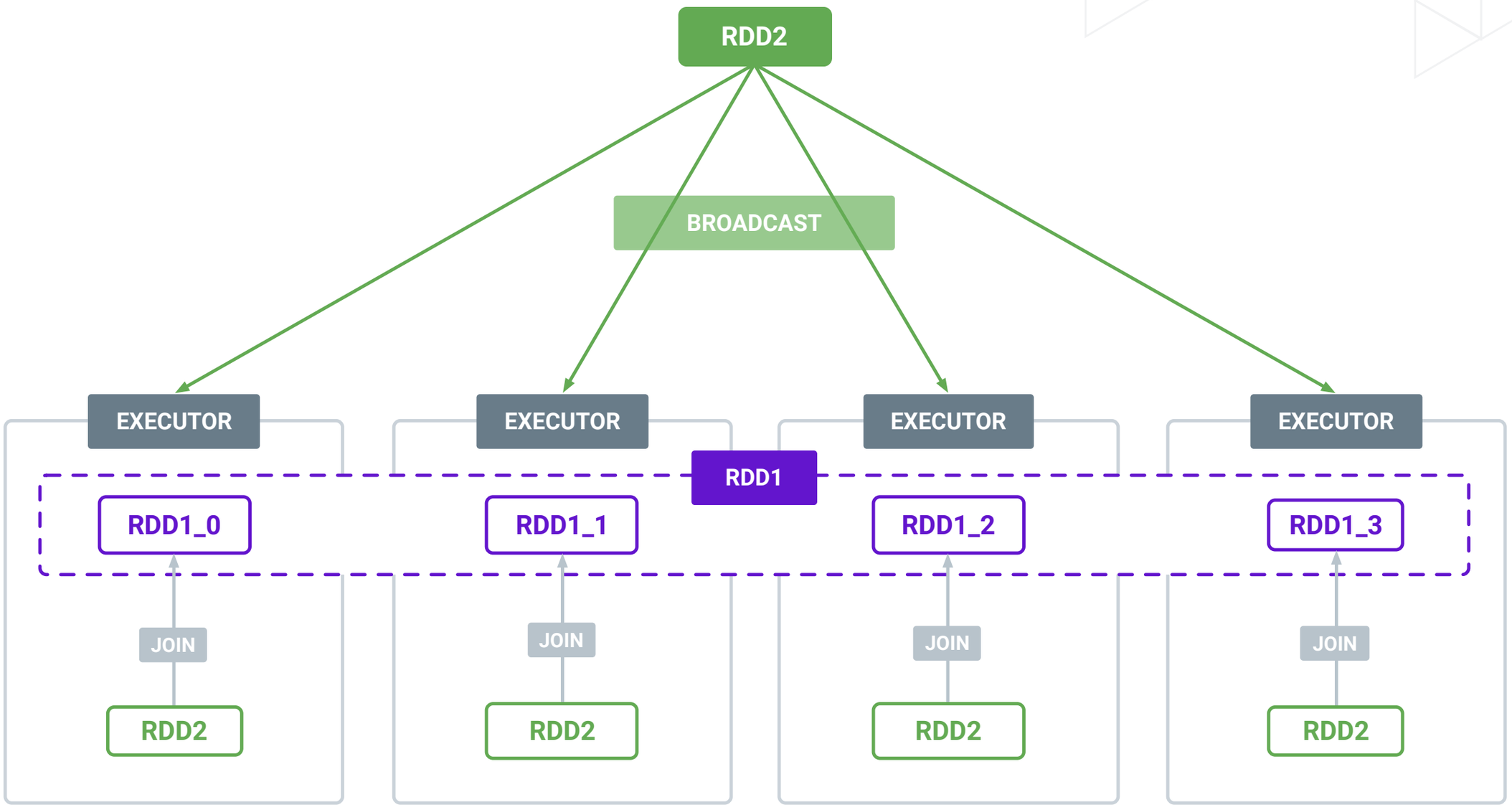



**Change map to mapValues**

# Optimize shuffle



# Broadcast variables





# Optimize shuffle - recap

- Control number of partitions
- Use mapValues instead of map if you can
- Broadcast variables
- Filter before shuffle
- Avoid groupByKey, use reduceByKey

# General notes

- Tests vs no tests? - Test your code!
- You are probably not the only user of the cluster
- What are you optimizing for?
- Share the knowledge
- Spark actually works :)

# Q & A

# Thank you and get in touch!

marcin@tantusdata.com