Getting started

(Developer's edition)

RED | VENTURES

Majid Fatemian

mfatemian@redventures.com
@majidfn

aws CERTIFIED - SAA

RED | VENTURES

TECH & DATA-DRIVEN

CUSTOMER ACQUISITION

SALES & MARKETING

DATA


PRODUCT RECOMMENDATION
MACHINE LEARNING
DATA SCIENCE
USER EXPERIENCE
REALTIME ANALYSIS
DATA WAREHOUSE

WHY THIS TALK?

UNIFIED COMPUTING ENGINE

for large-scale distributed data processing

EXAMPLE

PROBLEM

- 10K log files
- JSON
- S3
- Average latency for each user

PROBLEM

- 10K log files

```
{"userId": 1, "url": "page1", "latency": 30}
{"userId": 1, "url": "page2", "latency": 24}
{"userId": 4, "url": "page1", "latency": 40}
{"userId": 4, "url": "page5", "latency": 53}
```

- JSON
- S3

- Average Latency for each user

SOLUTION – IMPERATIVE / PROCEDURAL

```
// Import AWS SDK.
// Define Log Struct.

users = map[user][sum][count]

for file in s3.ListFiles("s3://logs-bucket") {
    content = s3.ReadFile(file)
    for line in content {
        log = Unmarshal(line)
        users[log.UserId].sum += log.latency
        users[log.UserId].count++
    }
}

for user in users {
    print user, sum / count
}
```

SOLUTION – APACHE SPARK (DECLARATIVE)

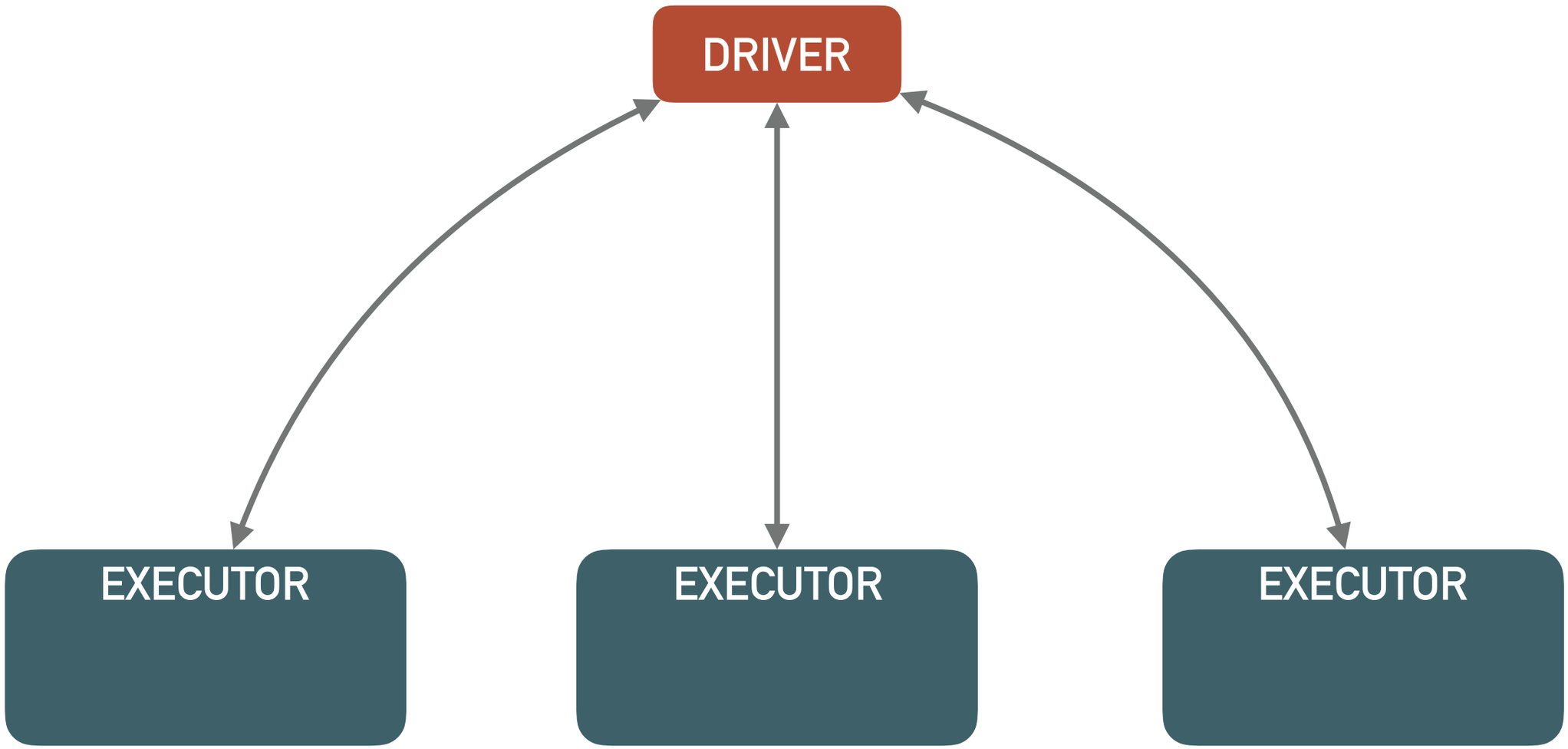
```
logs = spark.read.json("s3a://logs-bucket")
logs.groupBy(logs.userId).avg(logs.latency)
```

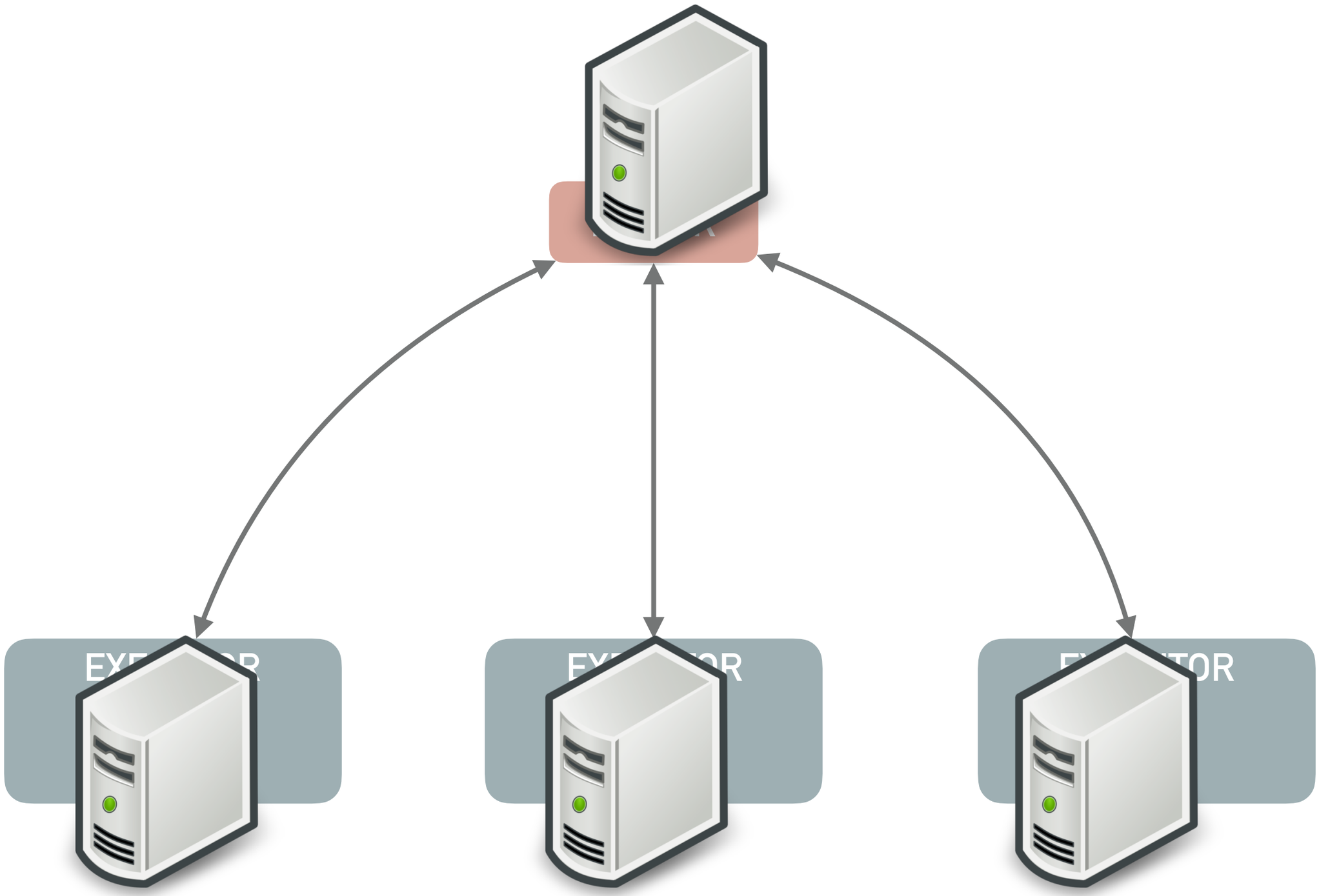
ABSTRACTION

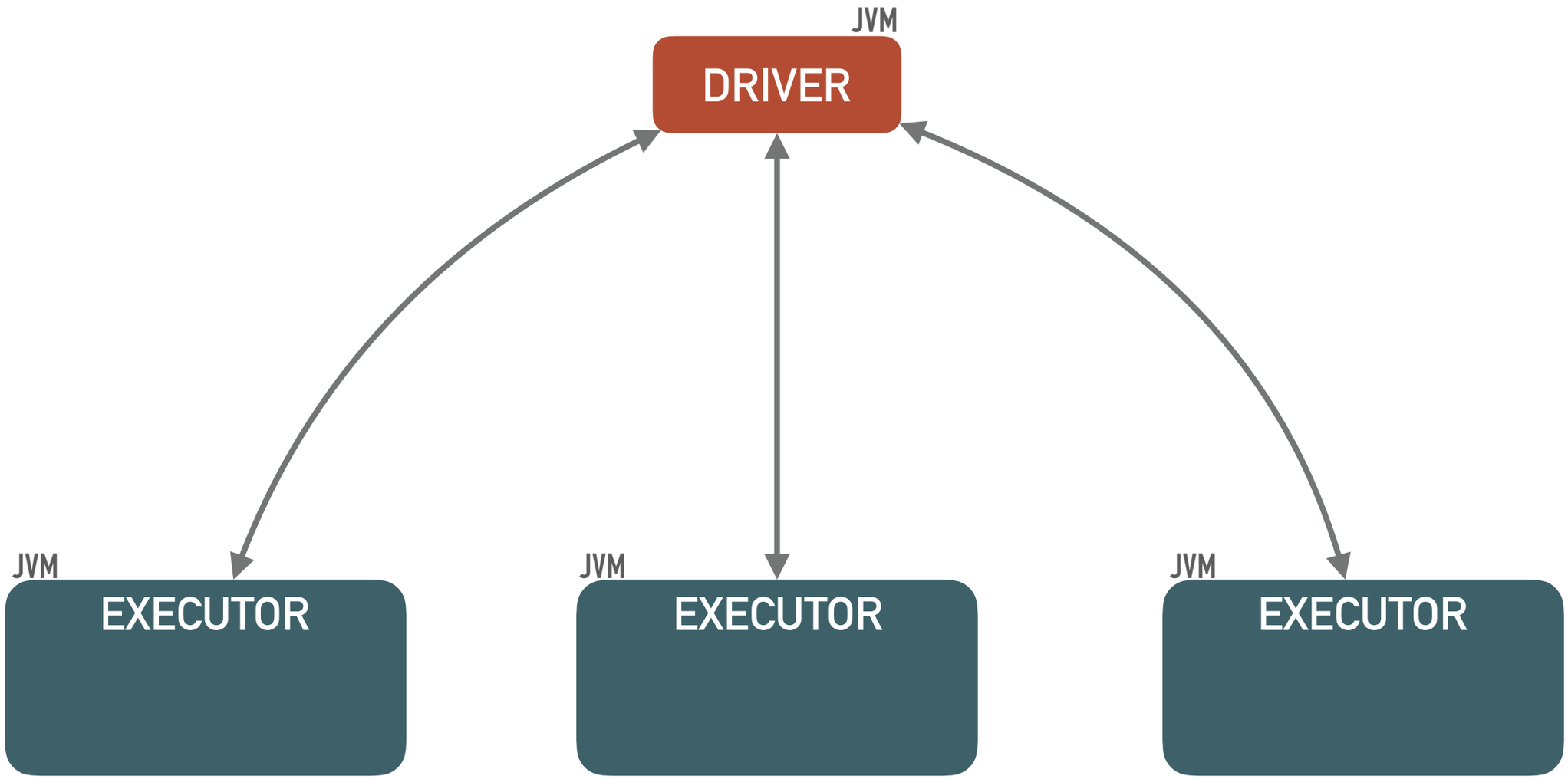
That's all it's about

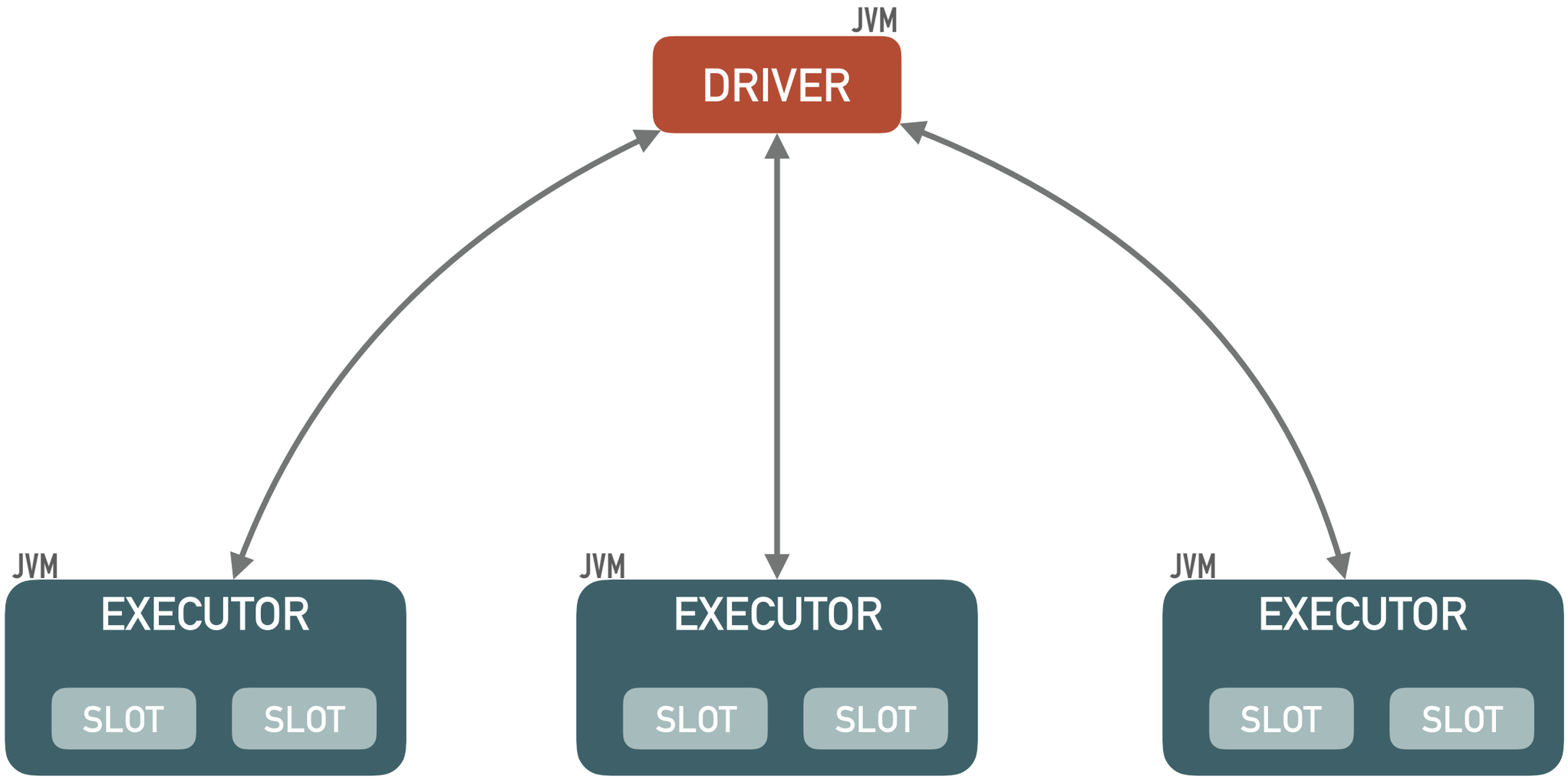
DISTRIBUTED

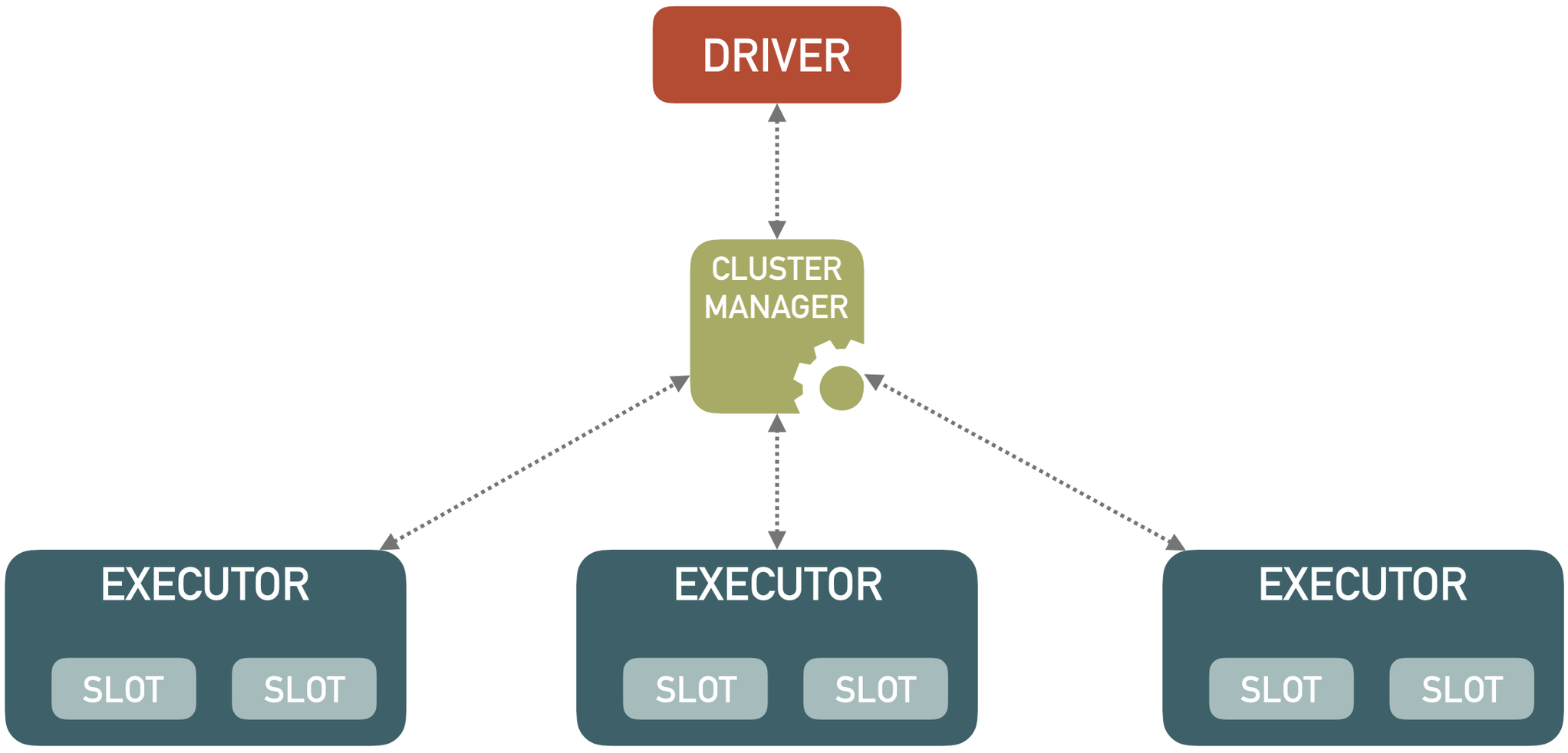
SPARK CLUSTER

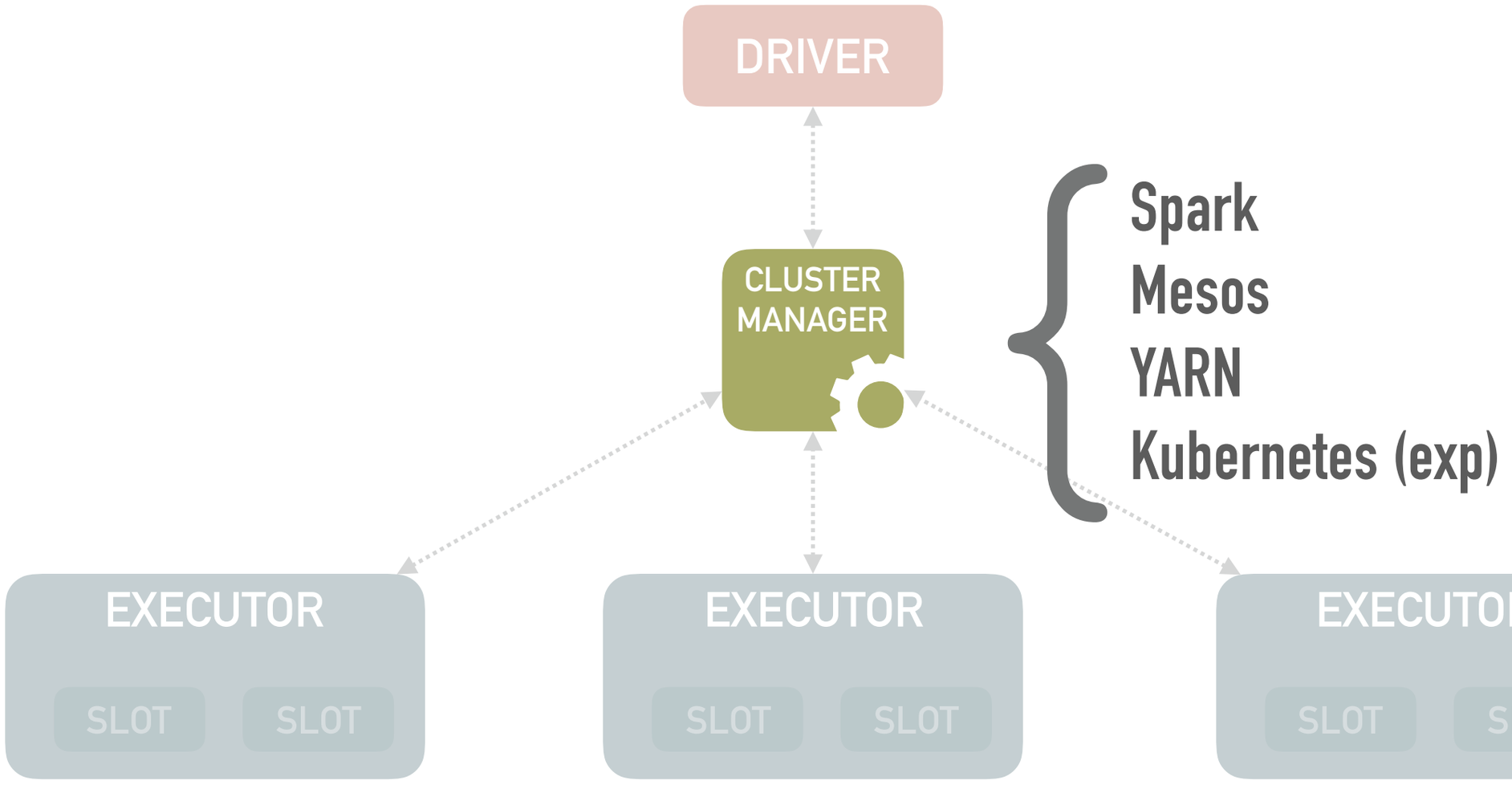

DRIVER
EXECUTOR
EXECUTOR
EXECUTOR

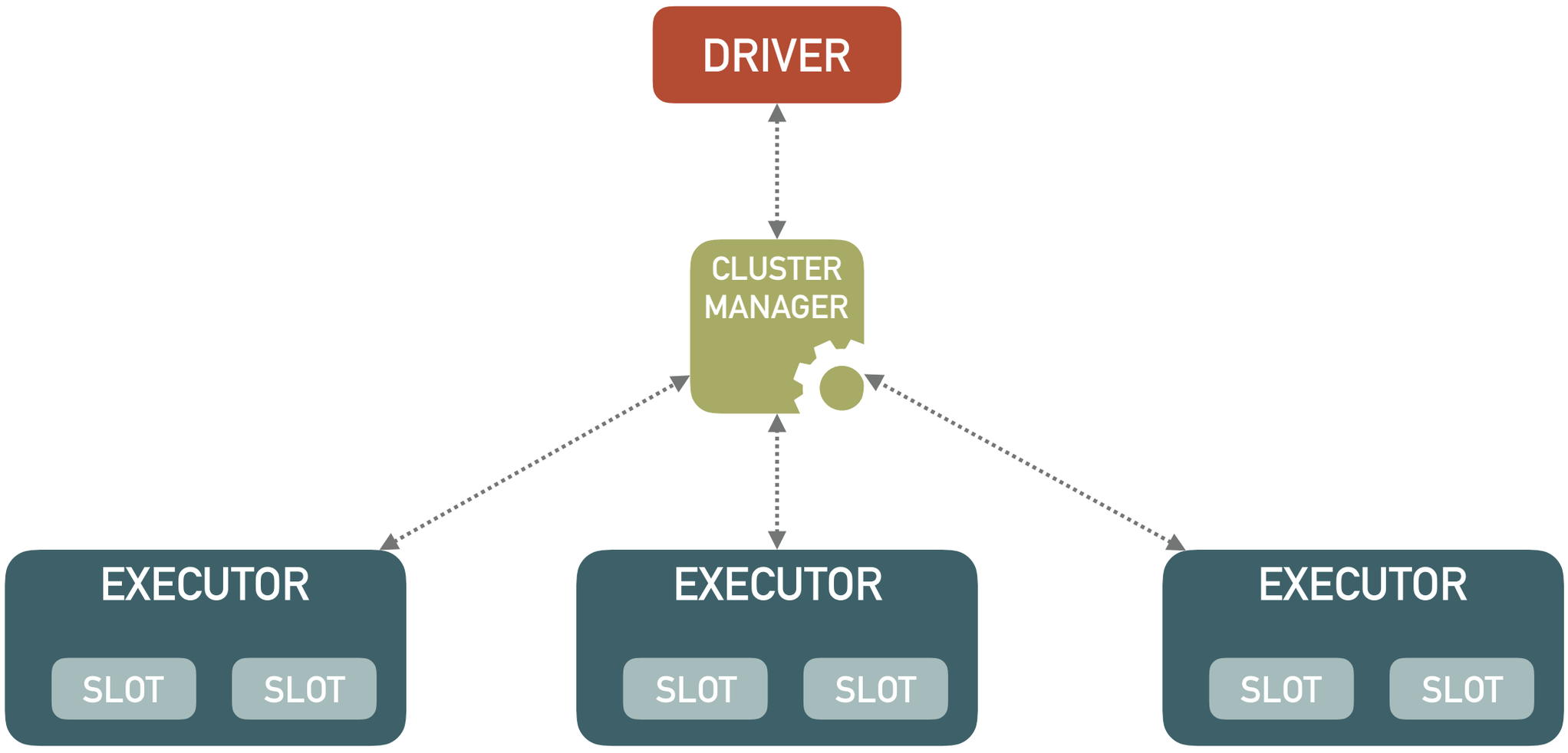



JVM
DRIVER
JVM
EXECUTOR
JVM
EXECUTOR
JVM
EXECUTOR


JVM
DRIVER
JVM
EXECUTOR
SLOT
SLOT
JVM
EXECUTOR
SLOT
SLOT
JVM
EXECUTOR
SLOT
SLOT

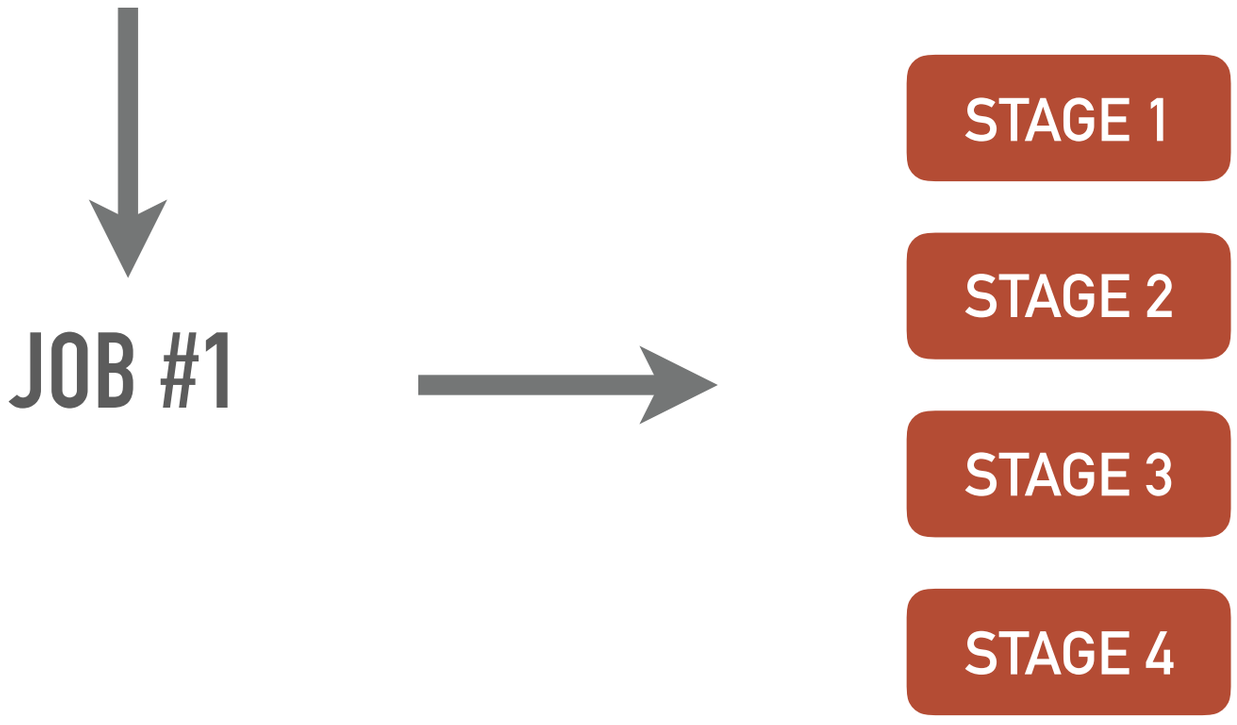

DRIVER
CLUSTER MANAGER
EXECUTOR
SLOT
SLOT
EXECUTOR
SLOT
SLOT
EXECUTOR
SLOT
SLOT

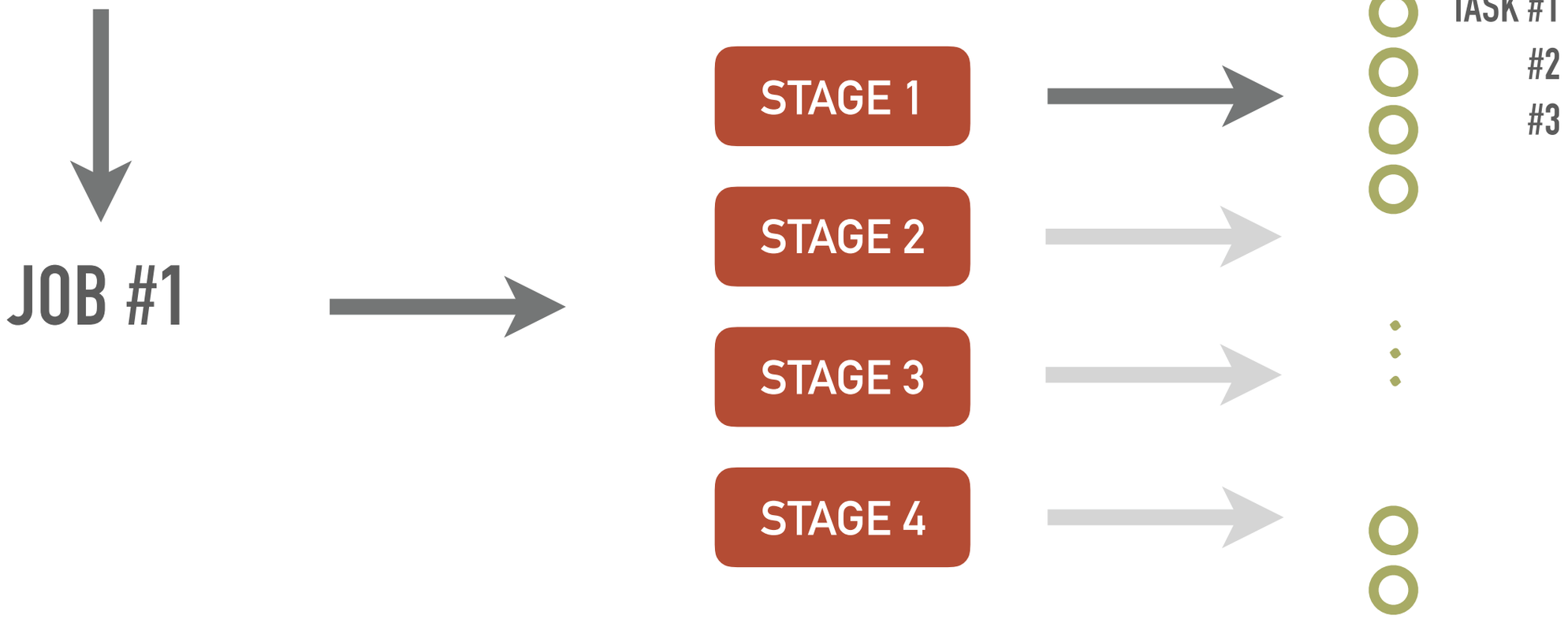

DRIVER
CLUSTER MANAGER
Spark
Mesos
YARN
Kubernetes (exp)
EXECUTOR
SLOT
SLOT
EXECUTOR
SLOT
SLOT
EXECUTOR
SLOT
SLOT

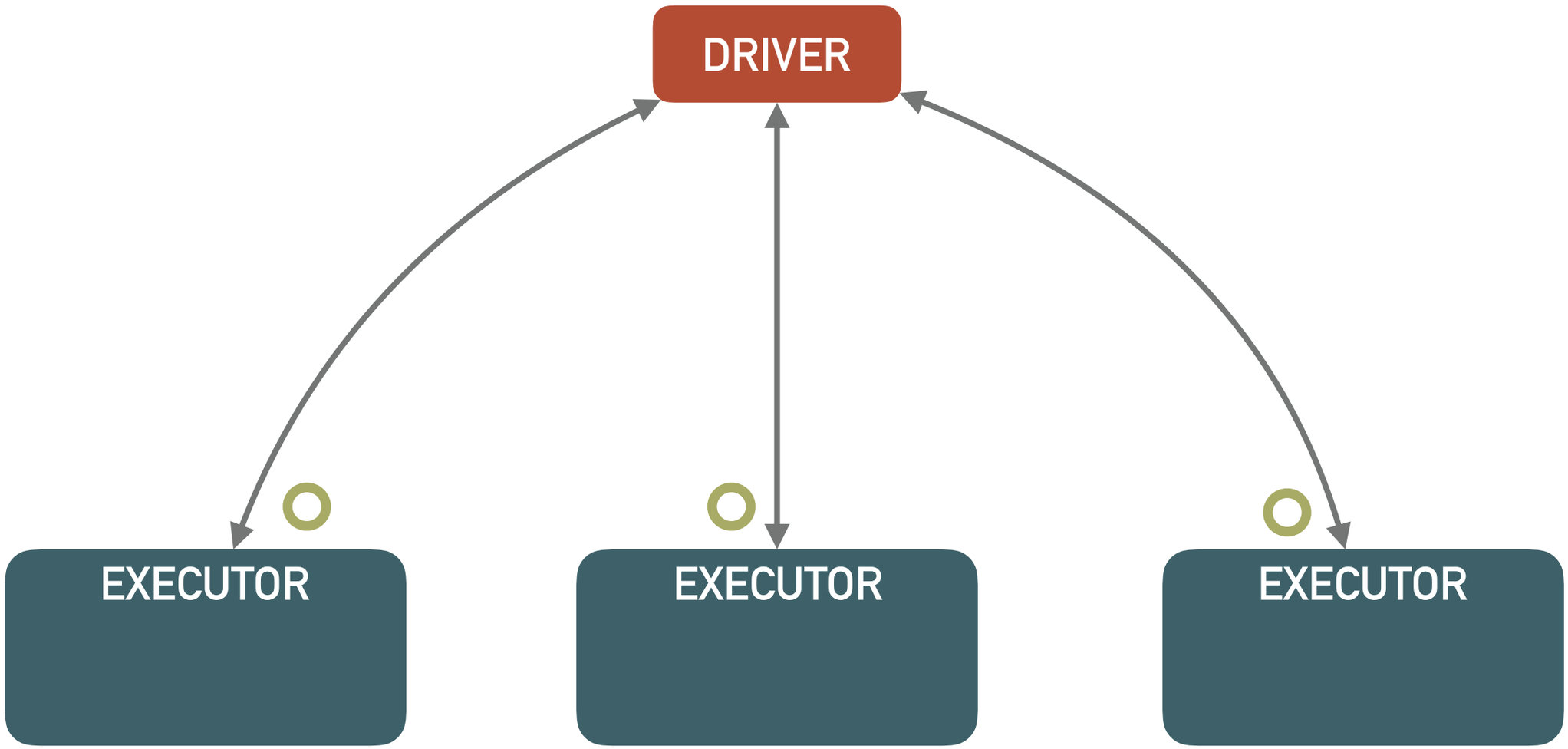

DRIVER
CLUSTER MANAGER
EXECUTOR
SLOT
SLOT
EXECUTOR
SLOT
SLOT
EXECUTOR
SLOT
SLOT

JOB / STAGE / TASK

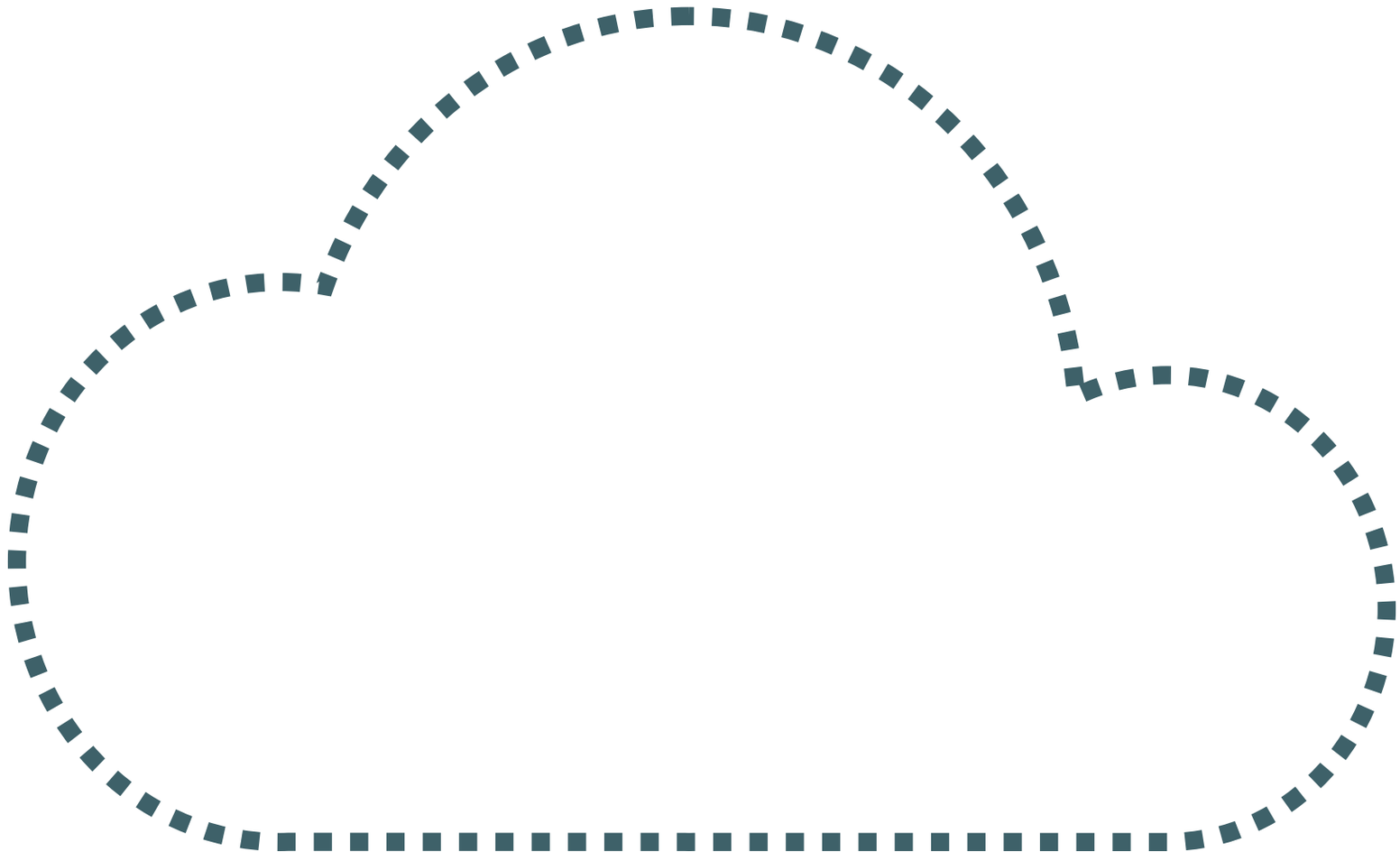
```
logs = spark.read.json("s3://logs-bucket")
logs.groupBy(logs.userId).avg(logs.latency)
```

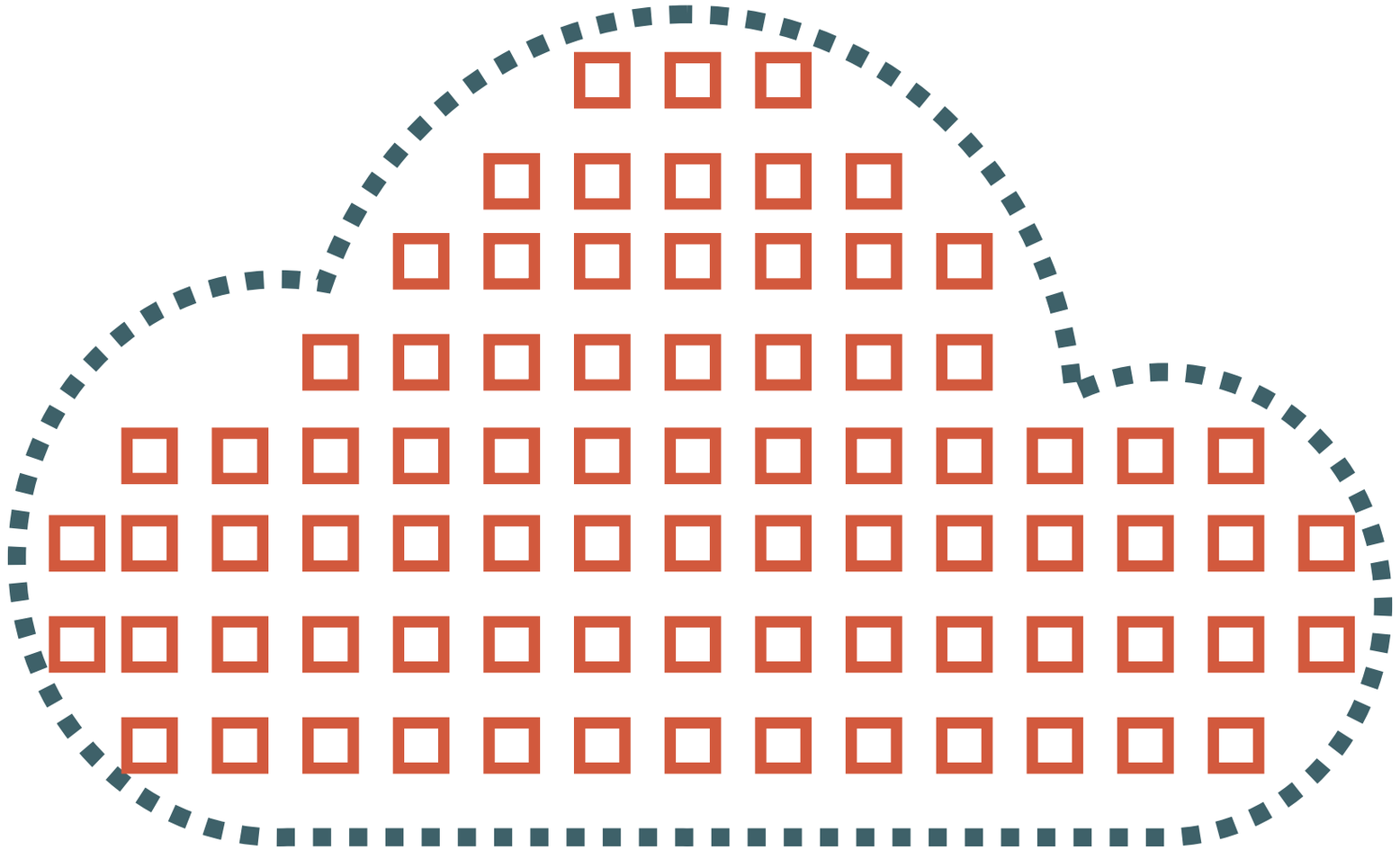

JOB #1
STAGE 1
STAGE 2
STAGE 3
STAGE 4

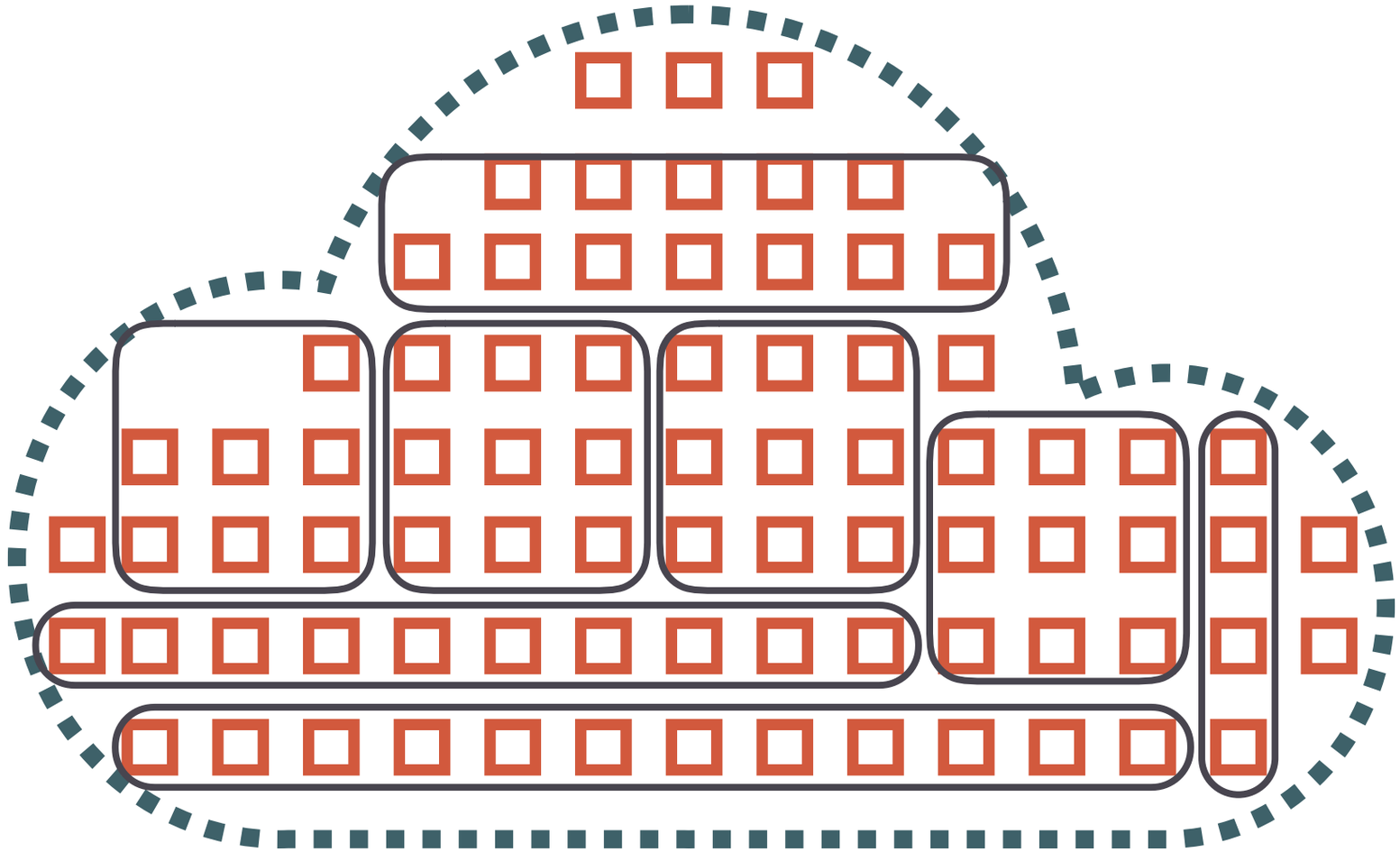

JOB #1
STAGE 1
STAGE 2
STAGE 3
STAGE 4
TASK #1
#2
#3

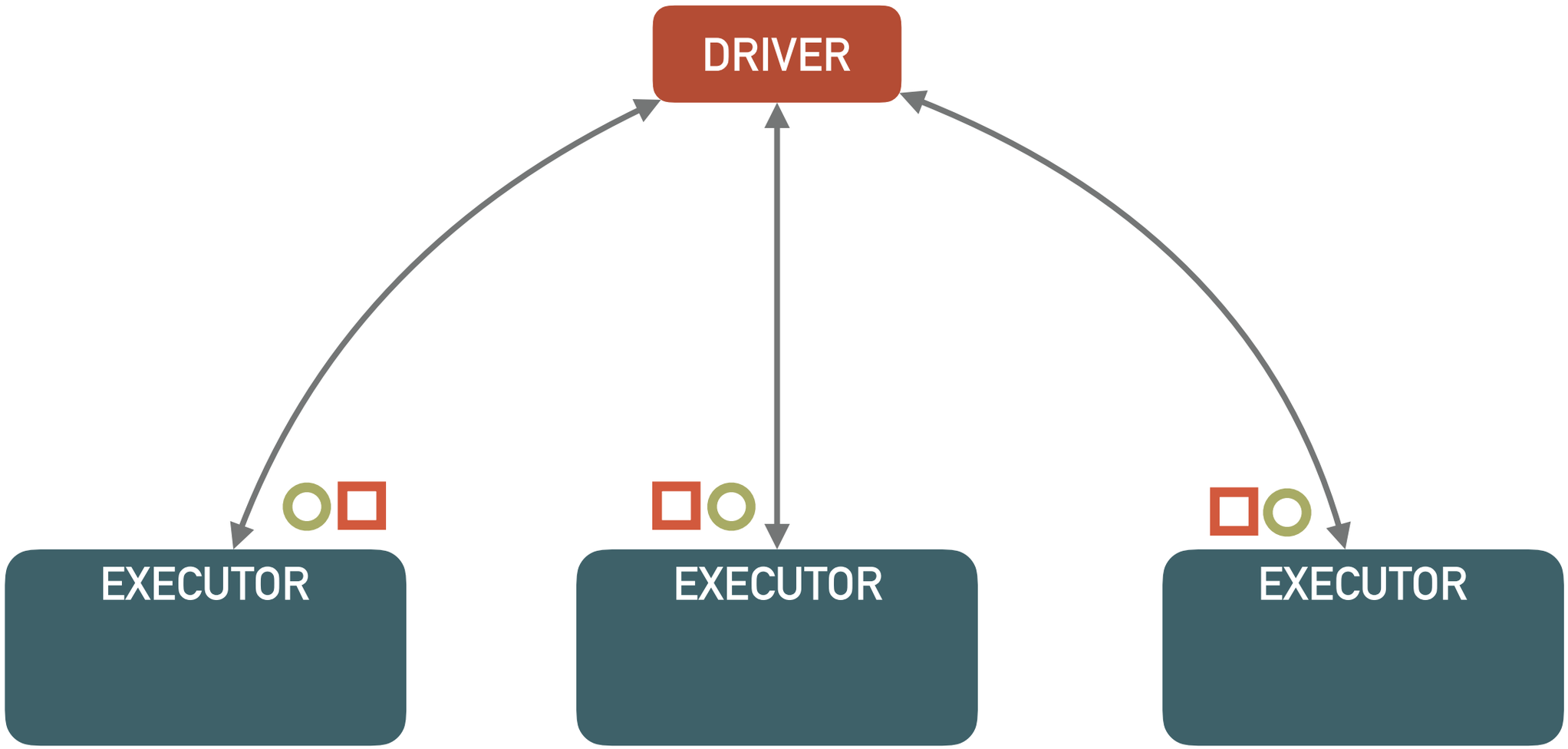

DRIVER
EXECUTOR
EXECUTOR
EXECUTOR

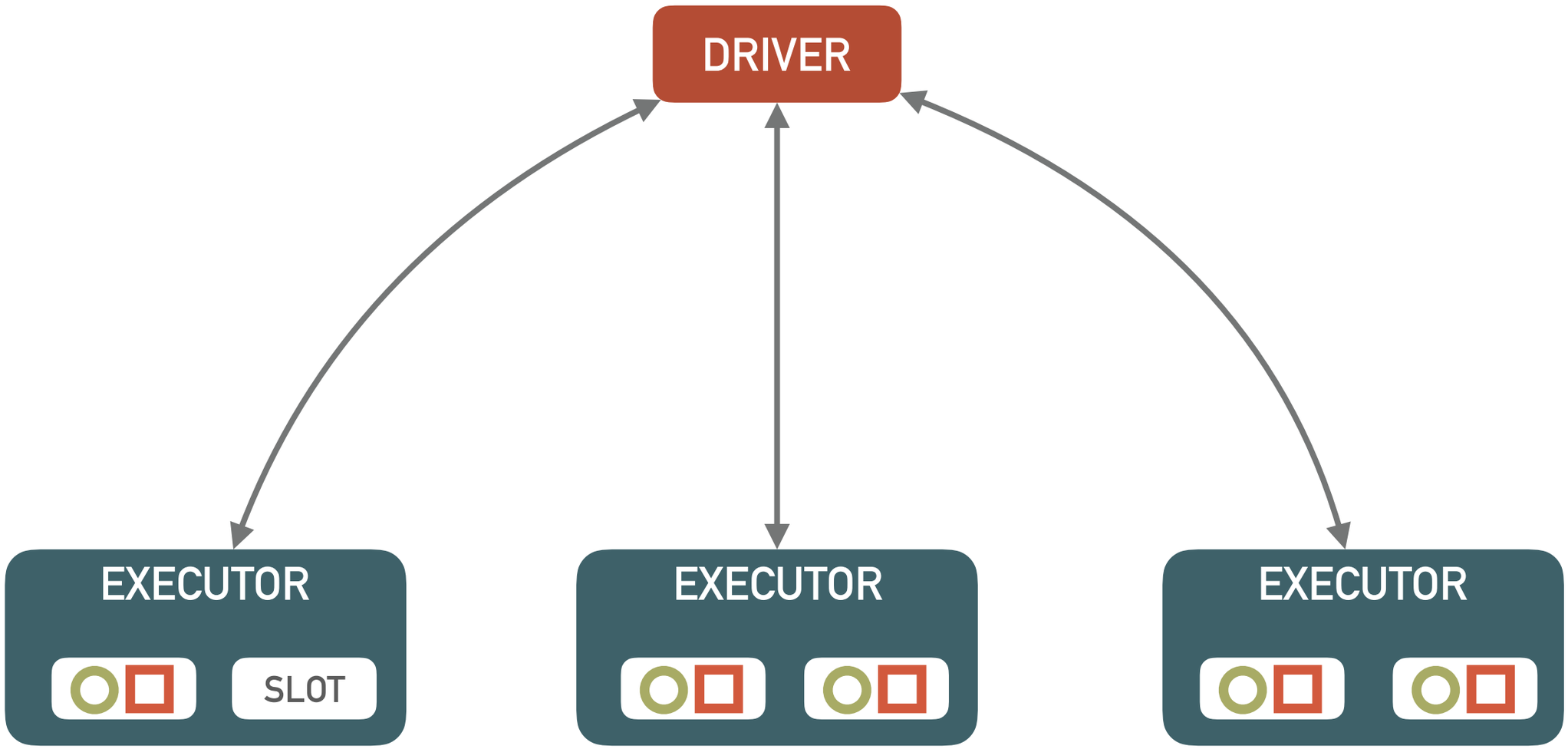
PARTITION








DRIVER
EXECUTOR
EXECUTOR
EXECUTOR


DRIVER
EXECUTOR
SLOT
EXECUTOR
EXECUTOR

UNIFIED

SQL

Streaming

Machine Learning

Graph

Spark SQL

Spark Streaming

MLlib

GraphX

Spark Core Engine

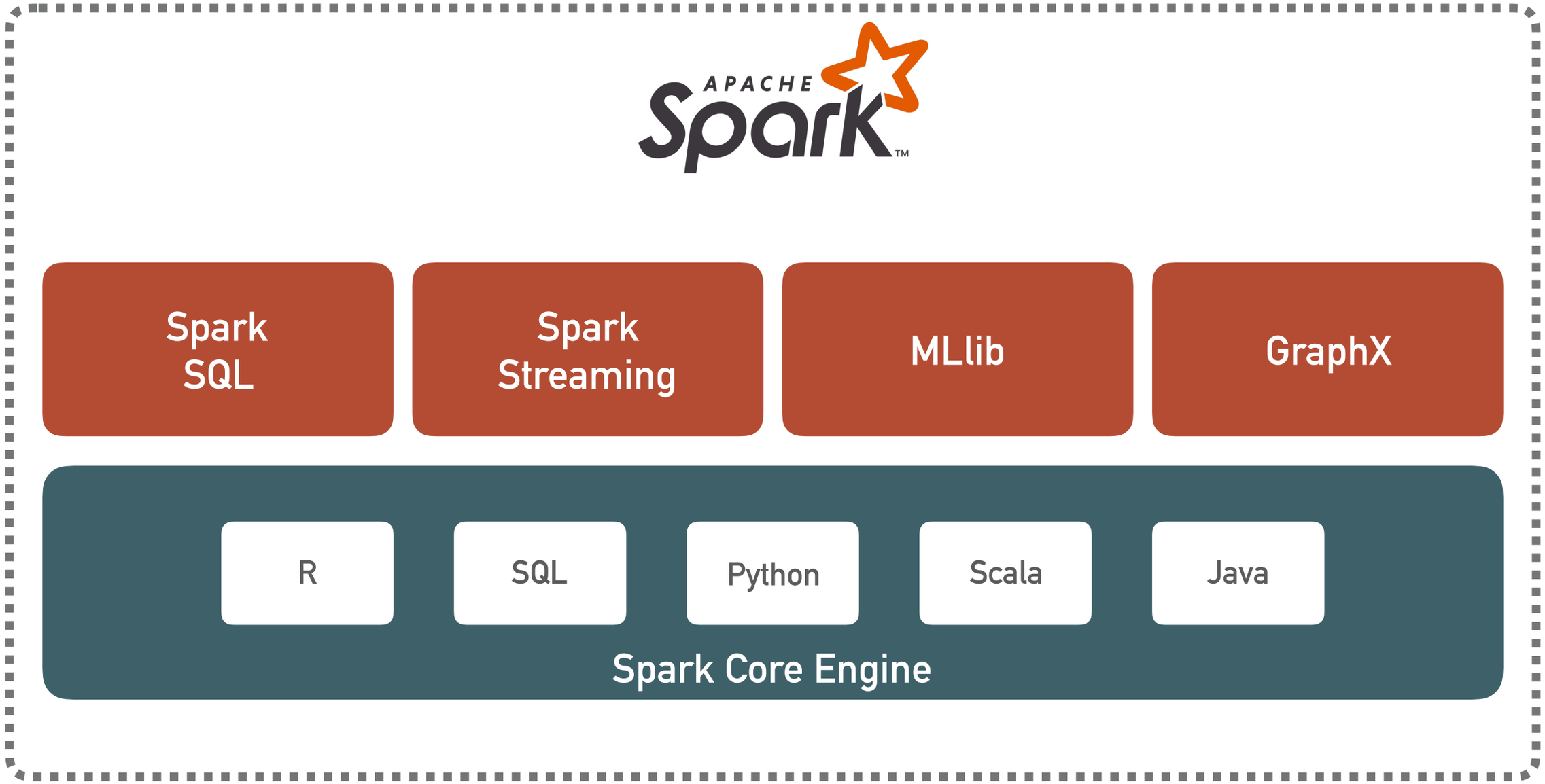
Spark SQL

Spark Streaming

MLlib

GraphX

Spark Core Engine


APACHE
Spark
Spark SQL
Spark Streaming
MLlib
GraphX
R
SQL
Python
Scala
Java
Spark Core Engine

UNIFIED

- Supports wide range of tasks
- Interactive and iterative ad-hoc queries
- Advanced ML
- Graph processing
- Streaming
- Consistent set of APIs across languages

COMPUTING

Engine

COMPUTING

- Limits its scope to computing / analytics
- No storage
- Apache Hadoop: storage + computation
  (HDFS) (MapReduce)

DATA SOURCES

DATA SOURCES – JSON FROM S3

```
spark
      .format("json")
      .read("s3a://logs-bucket")
```

DATA SOURCES – JSON FROM FILE SYSTEM

```
spark
      .format("json")
      .read("/path/logs-folder")
```

DATA SOURCES – CSV FROM S3

```
spark
      .format("csv")
      .read("s3a://logs-bucket-csvs")
```

DATA SOURCES – SQL DB – JDBC

```
spark
      .read
      .jdbc(jdbcUrl, "users", conProps)
```

DATA SOURCES – CASSANDRA

```
spark
    .format("org.apache.spark.sql.cassandra")
    .options(Map( "table" -> "X", "keyspace" -> "Y"))
    .read()
```

OTHER CONNECTORS

- MongoDB
- Kafka
- ElasticSearch
- Neo4j
- Redis
- SnowFlake
- Network socket

OTHER FILE FORMATS

- Avro
- Parquet
- Zip files
- LZO compressed

DATA SINKS

DATA SINKS – JSON TO S3

```
spark
    .format("json")
    .write("s3a://logs-result")
```

DATA SINKS – SQL DB – JDBC

```
spark
    .write
    .jdbc(jdbcUrl, "users", conProps)
```

API

Structured Streaming | Spark ML | GraphFrames

DataFrame | SQL | DataSet APIs

RDD APIs

Spark Core Engine

Spark Core Engine

RDD APIs

Spark Core Engine

RDD

Resilient Distributed Dataset

RDD

- Main abstraction in Spark
- Collections of distributed items
- Parallelism
- Fault tolerance

DataFrame APIs

RDD APIs

Spark Core Engine

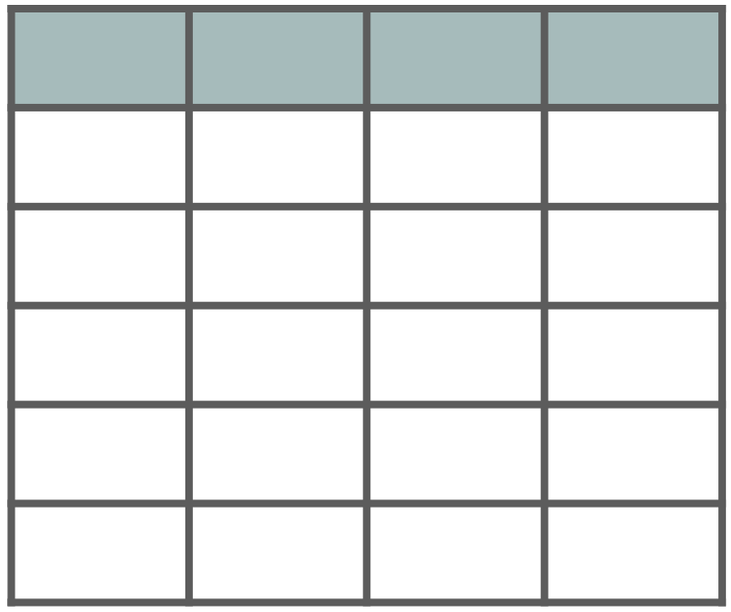
DATAFRAME

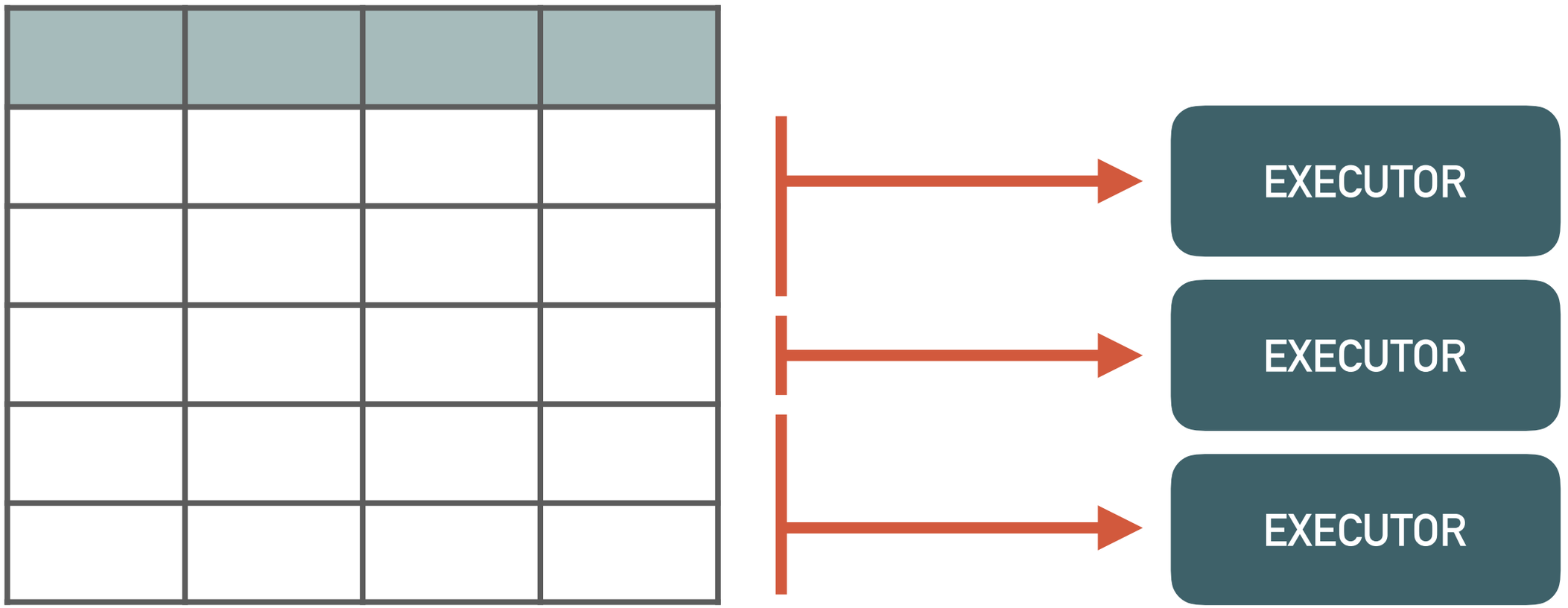
DATAFRAME

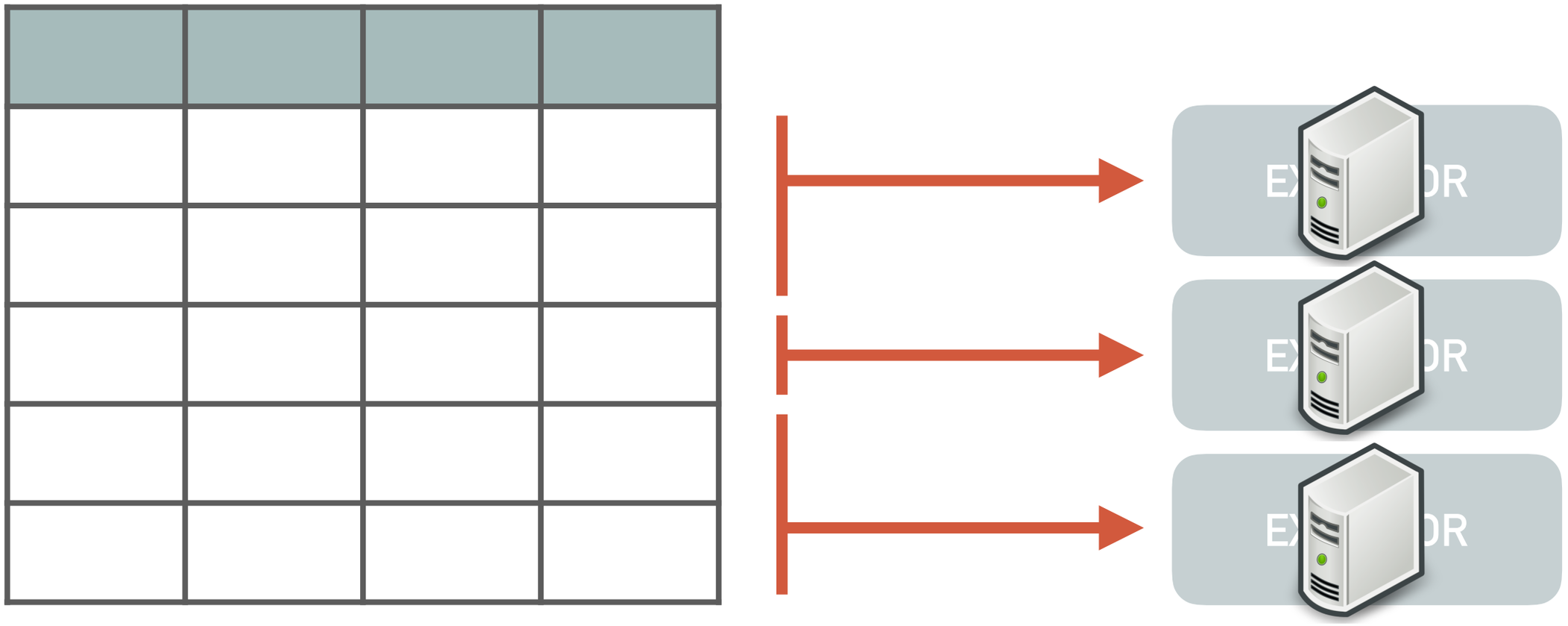
- Structured API
- Table: rows & columns
- Spreadsheet, but distributed

```
logs = spark.read.json("s3a://logs-bucket")
```

```
logs = spark.read.json("s3a://logs-bucket")
```




EXECUTOR
EXECUTOR
EXECUTOR


EX OR
EX OR
EX OR

TRANSFORMATIONS

TRANSFORMATIONS

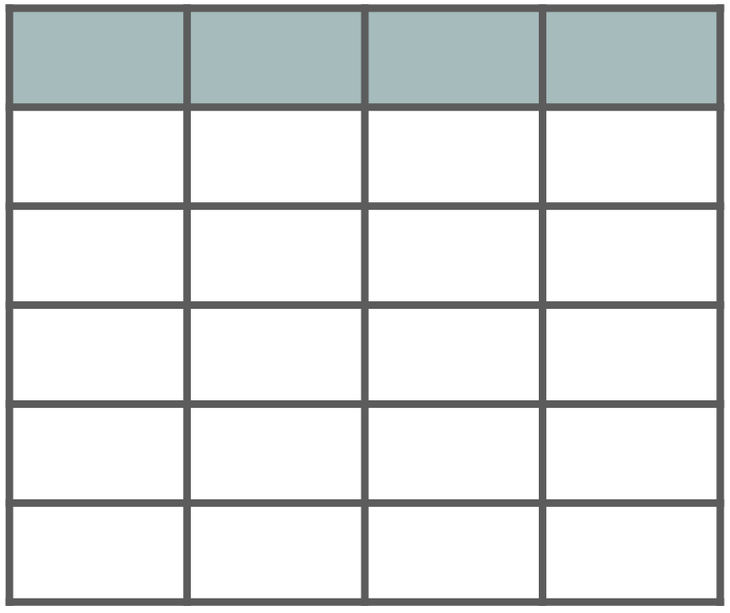
- filter
- map
- join
- groupByKey
- sortByKey
- ...

TRANSFORMATIONS

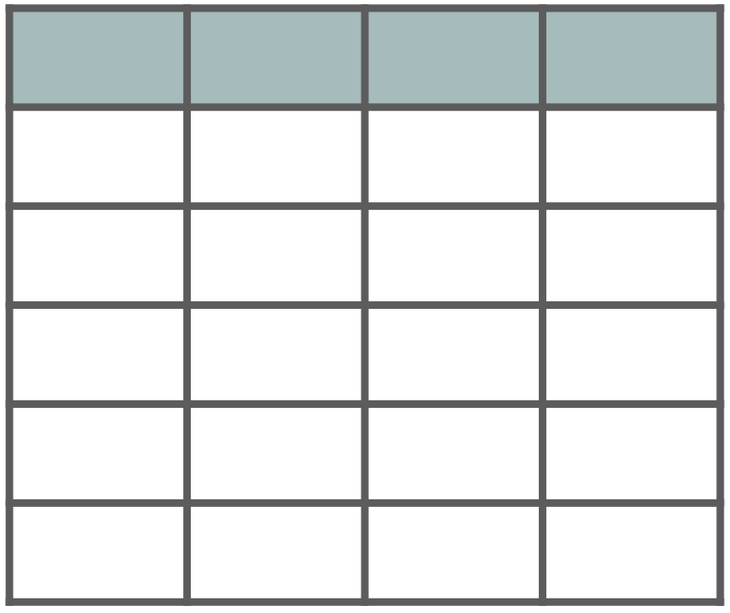
- Data is immutable in Spark

```
logs = spark.read.json("s3a://logs-bucket")
```

```
logs = spark.read.json("s3a://logs-bucket")
```



```
logs = spark.read.json("s3a://logs-bucket")
late_logs = logs.where(“latency > 30”)
```



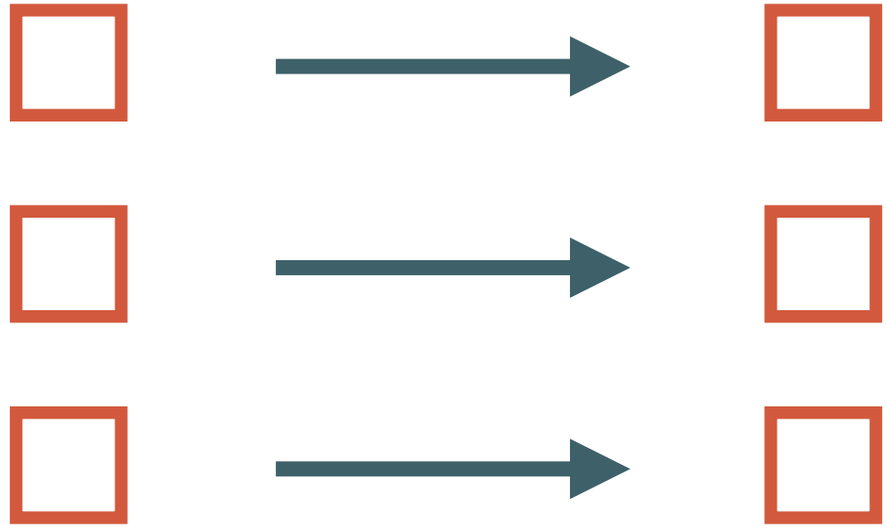
```
late_logs = logs.where("latency > 30")
```

NARROW TRANSFORMATIONS

One input partition contributes to one output

In-memory

Like (usually): **`where, filter, map`**

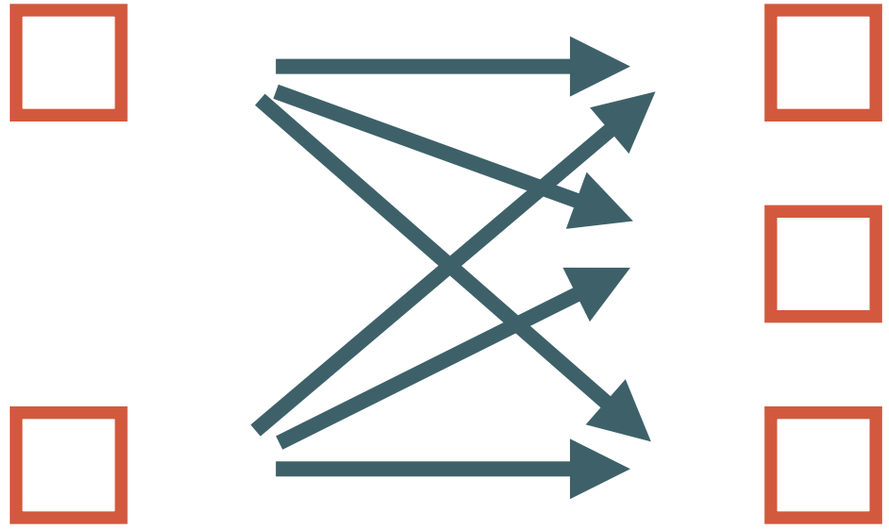


WIDE TRANSFORMATIONS

One input Partition contributes to many outputs (shuffle)

Write results to disk

Like (usually): **`groupByKey, join, aggregation`**



ACTIONS

ACTIONS

- count
- collect
- take
- saveAsTextFile
- …

LAZY

Evaluation

```
logs = spark.read.json("s3a://logs-bucket")

logs.groupBy(logs.userId)
    .avg(logs.latency)
```

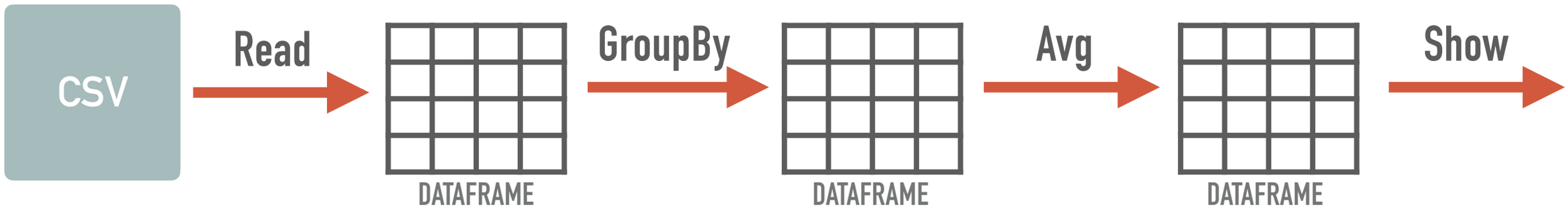
```
logs = spark.read.json("s3a://logs-bucket")

logs.groupBy(logs.userId)
     .avg(logs.latency)
     .show()
```

```
logs = spark.read.json("s3a://logs-bucket")
logs.groupBy(logs.userId)
      .avg(logs.latency)
      .show()
```

```
logs = spark.read.json("s3a://logs-bucket")
logs.groupBy(logs.userId)
    .avg(logs.latency)
    .show()
```

```
logs = spark.read.json("s3a://logs-bucket")
logs.groupBy(logs.userId)
      .avg(logs.latency)
      .show()
```



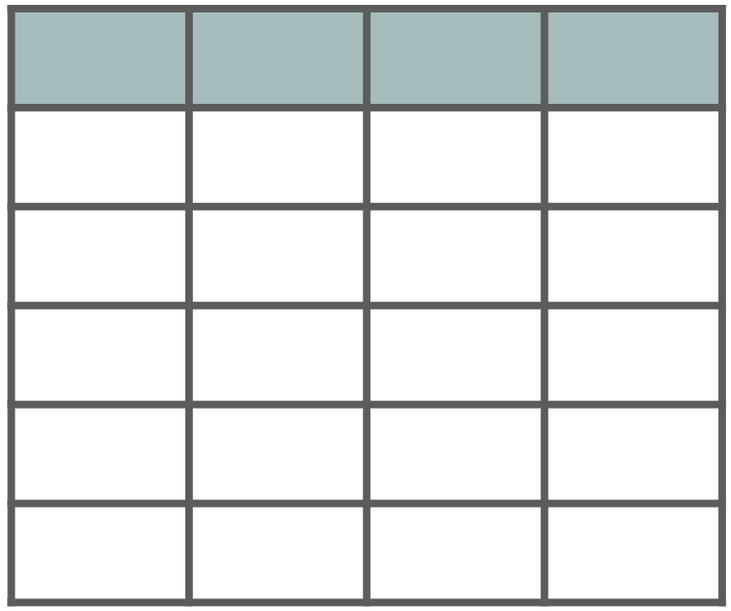
DataFrame | SQL APIs

RDD APIs

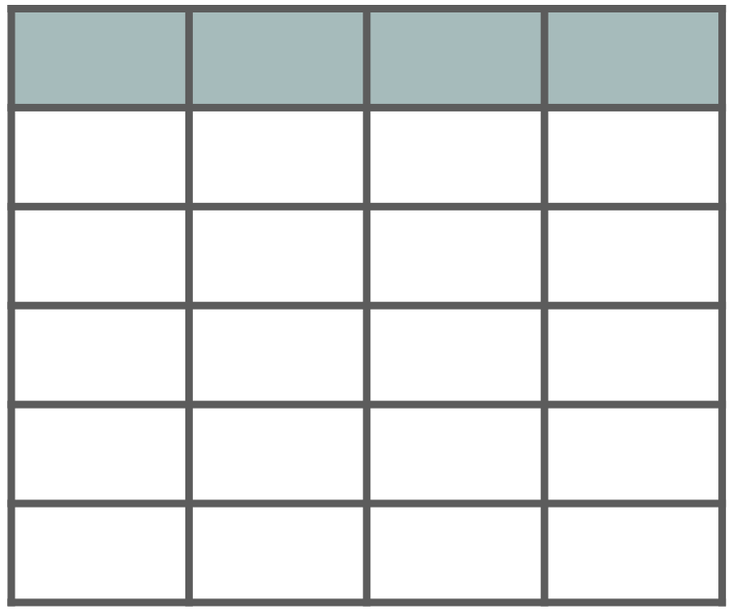
Spark Core Engine

SQL

```
logs = spark.read.json("...")
```



```
logs = spark.read.json("...")
logs.createOrReplaceTempView("logs")
```



```
logs = spark.read.json("...")
logs.createOrReplaceTempView("logs")
sqlDF = spark.sql("""
    SELECT userId, AVG(latency)
    FROM logs
    GROUP BY userId
""")
```

```
logs = spark.read.json("...")
logs.createOrReplaceTempView("logs")
sqlDF = spark.sql("""
    SELECT userId, AVG(latency)
    FROM logs
    GROUP BY userId
""")

sqlDF.show()
```

Structured Streaming | Spark ML | GraphFrames

DataFrame | SQL | DataSet APIs

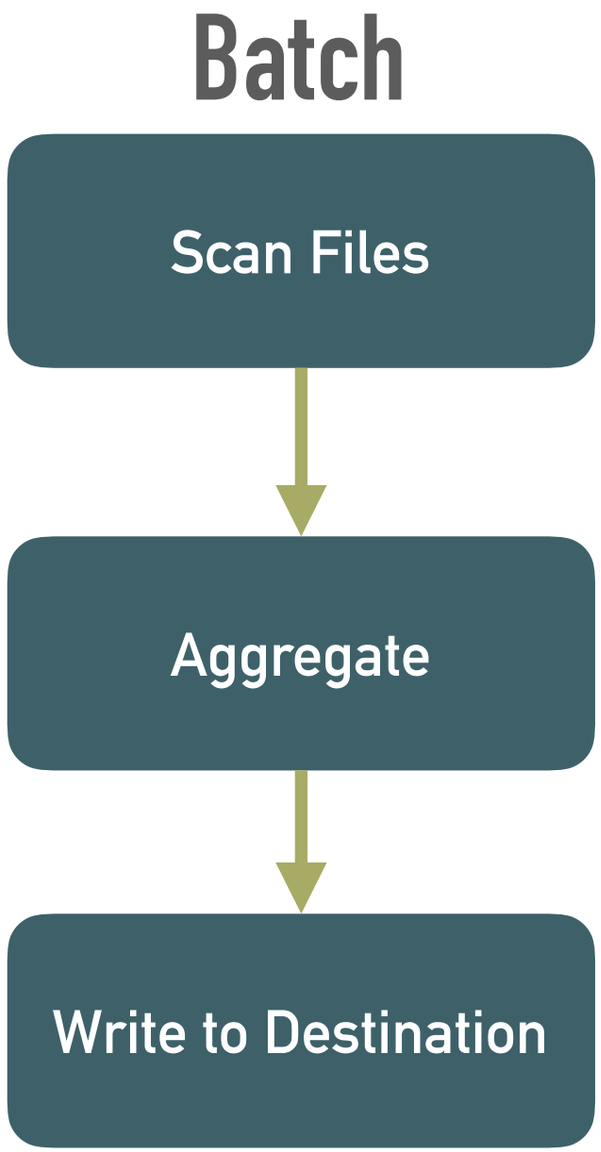
RDD APIs

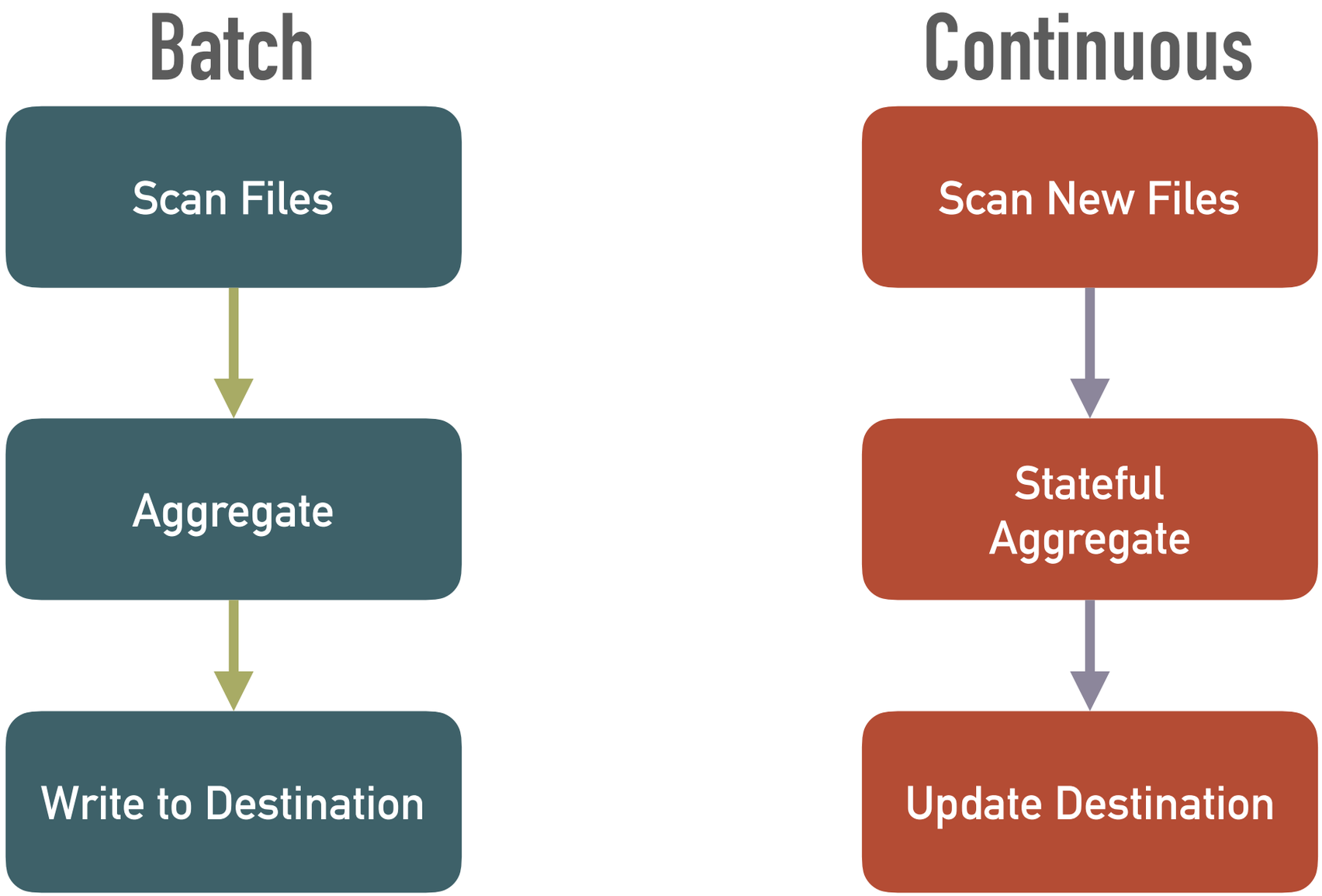
Spark Core Engine

STRUCTURED STREAMING

PROBLEM

- 10K log files **& growing**
- JSON
- S3
- Average latency for each user


Batch
Scan Files
Aggregate
Write to Destination


Batch
Scan Files
Aggregate
Write to Destination
Continuous
Scan New Files
Stateful Aggregate
Update Destination

STRUCTURED STREAMING

- Fault tolerance
- Checkpointing
- Continuous DataFrame

BATCH AGGREGATION

- Re-Calculate / Scan the whole dataset

```
logs = spark.read.json("s3a://logs-bucket")
logs.groupBy(logs.userId)
    .avg(logs.latency)
```

CONTINUOUS AGGREGATION

```
logs = spark.readStream.json("s3a://logs-bucket")
logs.groupBy(logs.userId)
    .avg(logs.latency)
```

CONTINUOUS AGGREGATION

```
logs = spark.readStream.json("s3a://logs-bucket")
logs.groupBy(logs.userId)
    .avg(logs.latency)
```

ALL ABOUT ABSTRACTION

- Abstracting cluster management
- Abstracting state management
- Abstracting workload distribution
- Abstracting data sources
- Abstracting data sinks

UNIFIED COMPUTING ENGINE

for large-scale distributed data processing

WHEN TO USE APACHE SPARK?

- Data could be partitioned
- Processing could be parallelized
- Data set is *big(!) enough*

HOW TO RUN SPARK

HOW TO RUN SPARK

- AWS Elastic Map Reduce (EMR)
- DataBricks + Community Edition
- Inside the code (Local mode)
- Docker

QUESTIONS?

EXTRA

GROWTH OF DATA

- Moore's law slowed down
- Multiple cores, parallel processing, distributed computing
- Growth of data
- Storage is cheap, processing is not

HISTORY OF SPARK

- 2009 by Matei Zaharia
- 2013 donated to Apache foundation
- 2014 top level Apache project
- 85% written in Scala

AWS S3 AUTH

➤ 1

```
sc._jsc.hadoopConfiguration().set("fs.s3n.awsAccessKeyId", ACCESS_KEY)
sc._jsc.hadoopConfiguration().set("fs.s3n.awsSecretAccessKey", SECRET_KEY)
sc.textFile("s3a://%s/.../..." % MOUNT_NAME)
```

➤ 2

```
sc.textFile("s3a://%s:%s@%s/.../..." % ACCESS_KEY, ENCODED_SECRET_KEY, BUCKET_NAME)
```

DATA SOURCES – SQL DB – JDBC

```
val jdbcUrl = s"jdbc:mysql://${h}:${p}/${db}"
conProps.put("user", s"${u}")
conProps.put("password", s"${pwd}")
spark
     .read
  .jdbc(jdbcUrl, "users", conProps)
```