



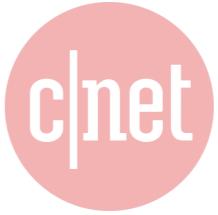
*for Software Engineers*

# **RED | VENTURES**

Majid Fatémian

[mfatemian@redventures.com](mailto:mfatemian@redventures.com)  
@majidfn

**RED | VENTURES**



healthline



Bankrate®

creditcards

the  
**POINTS**  
↗**GUY**

**M** MYMOVE™

all**connect**®

We help people  
discover and  
decide



healthline



Bankrate®

creditcards

the  
**POINTS**  
↗**GUY**

M MYMOVE™

allconnect®

# TECH & DATA



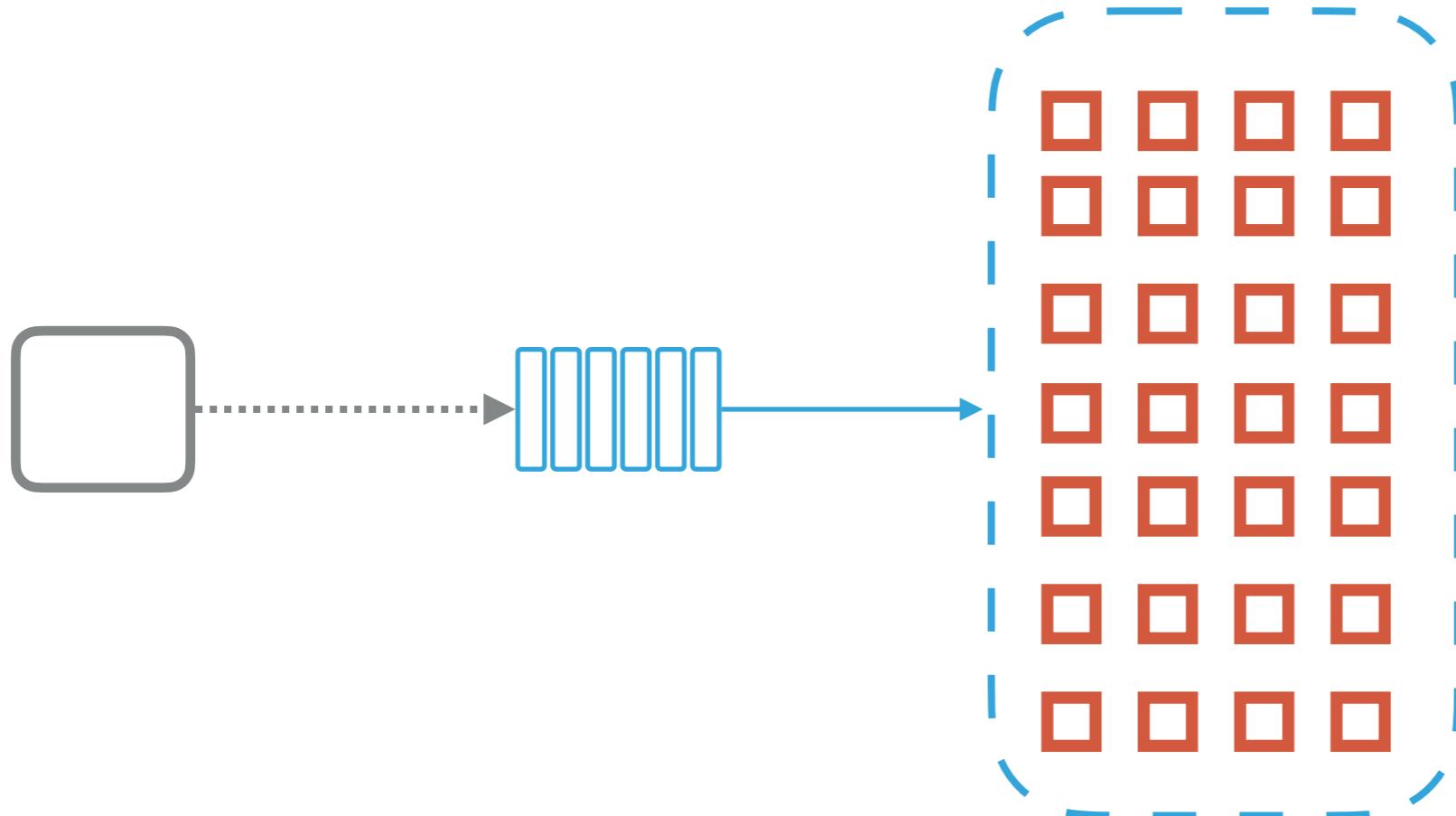
# **WHY THIS TALK?**

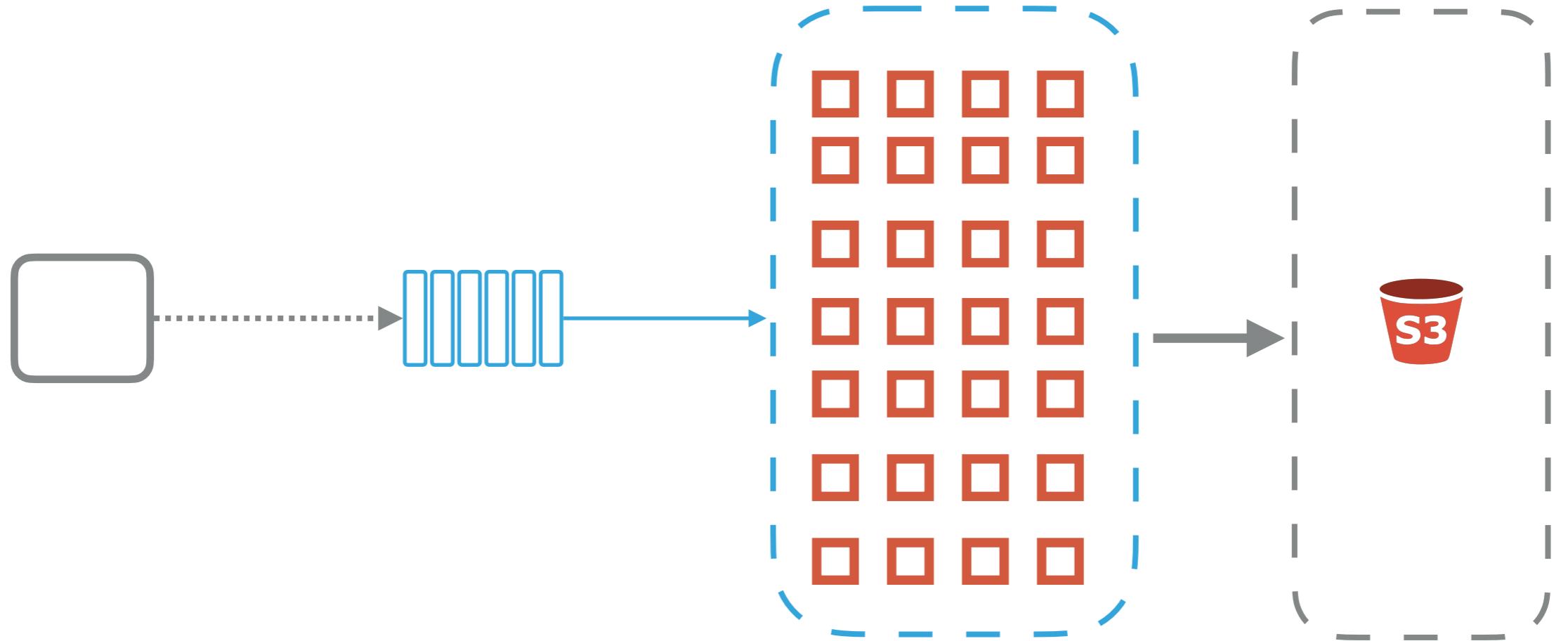
# **UNIFIED COMPUTING ENGINE**

---

*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 }
- JSON
  - { "userId": 1, "url": "page2", "latency": 24 }
  - { "userId": 4, "url": "page1", "latency": 40 }
  - { "userId": 4, "url": "page5", "latency": 53 }
- 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

---

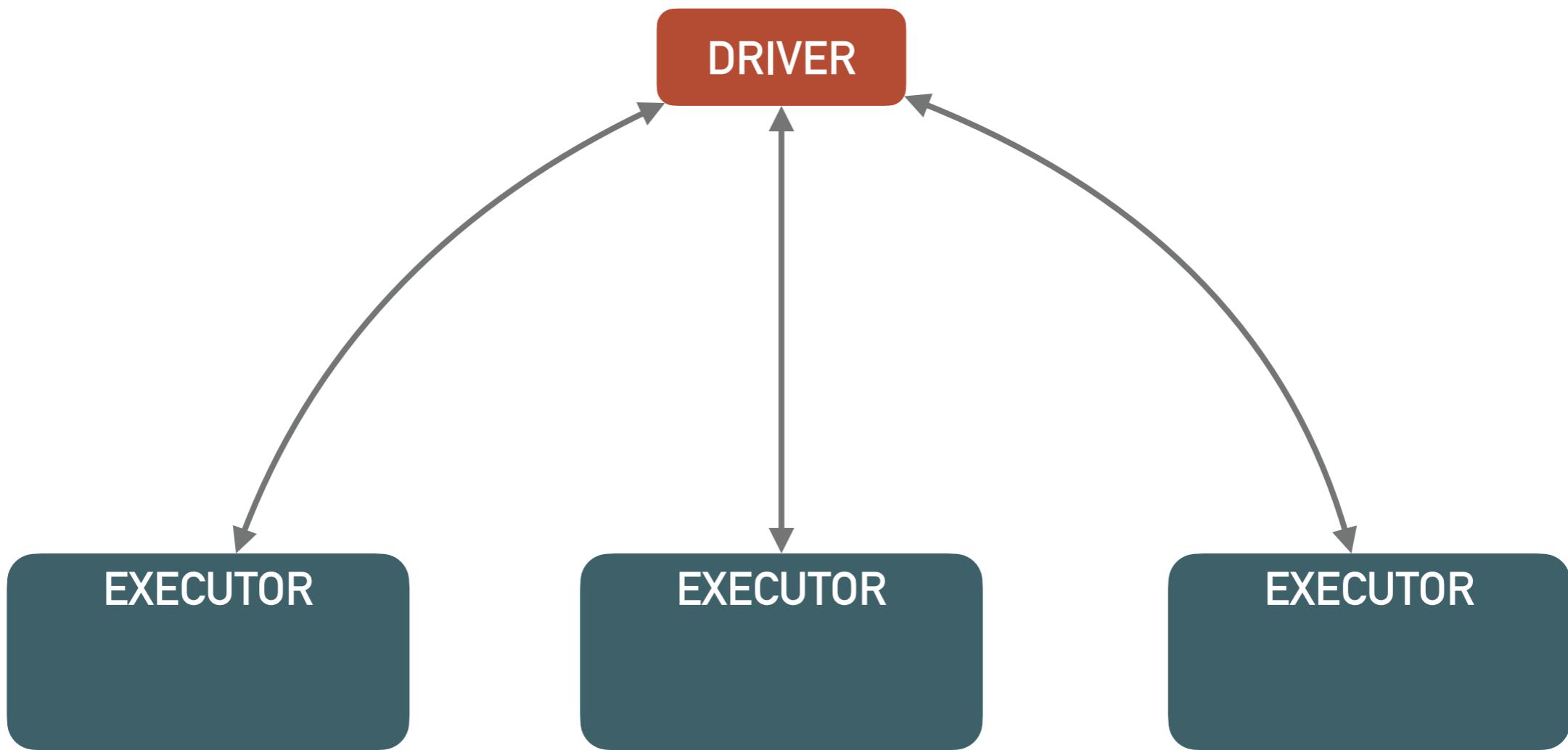
*That's all it's about*

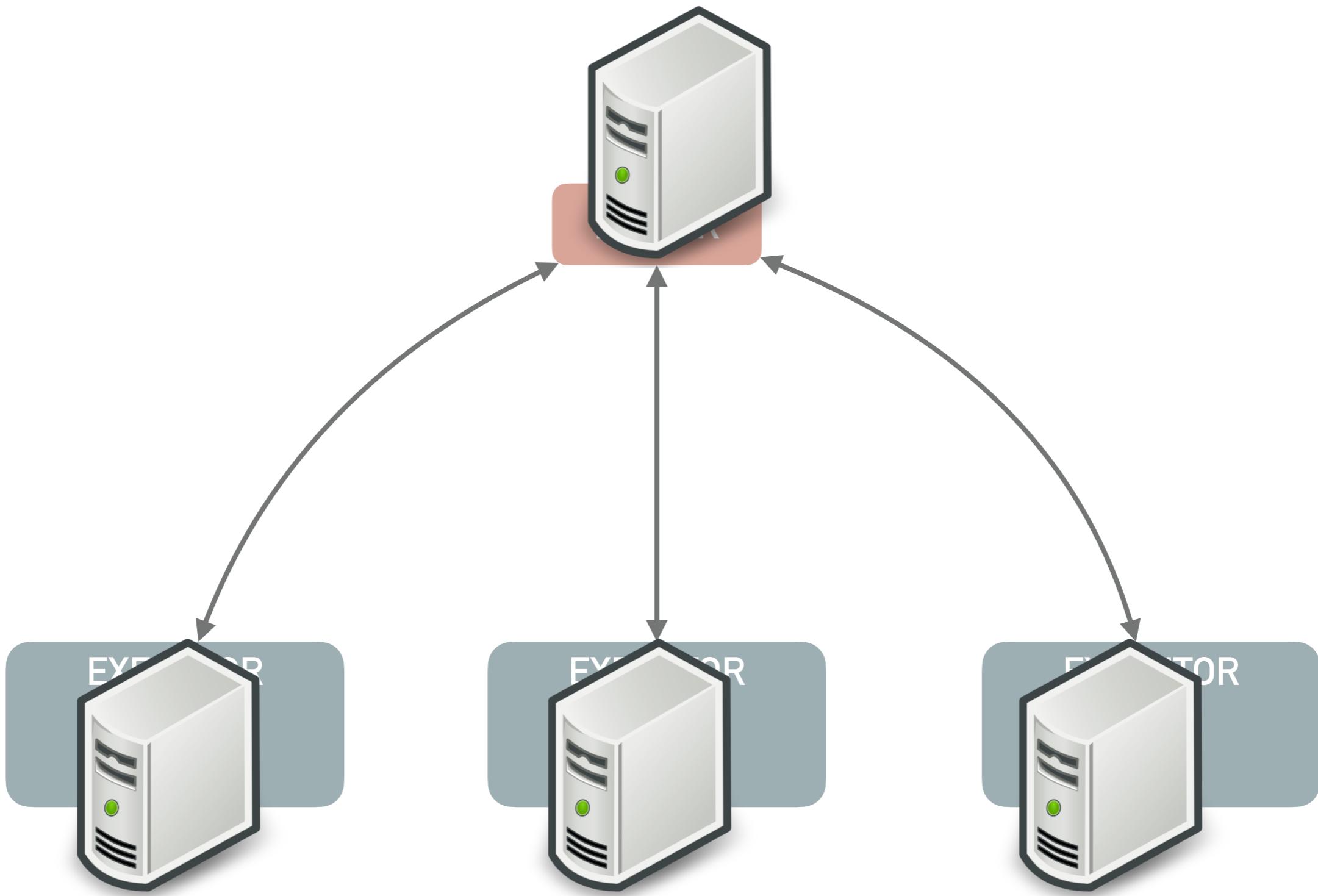
# DISTRIBUTED

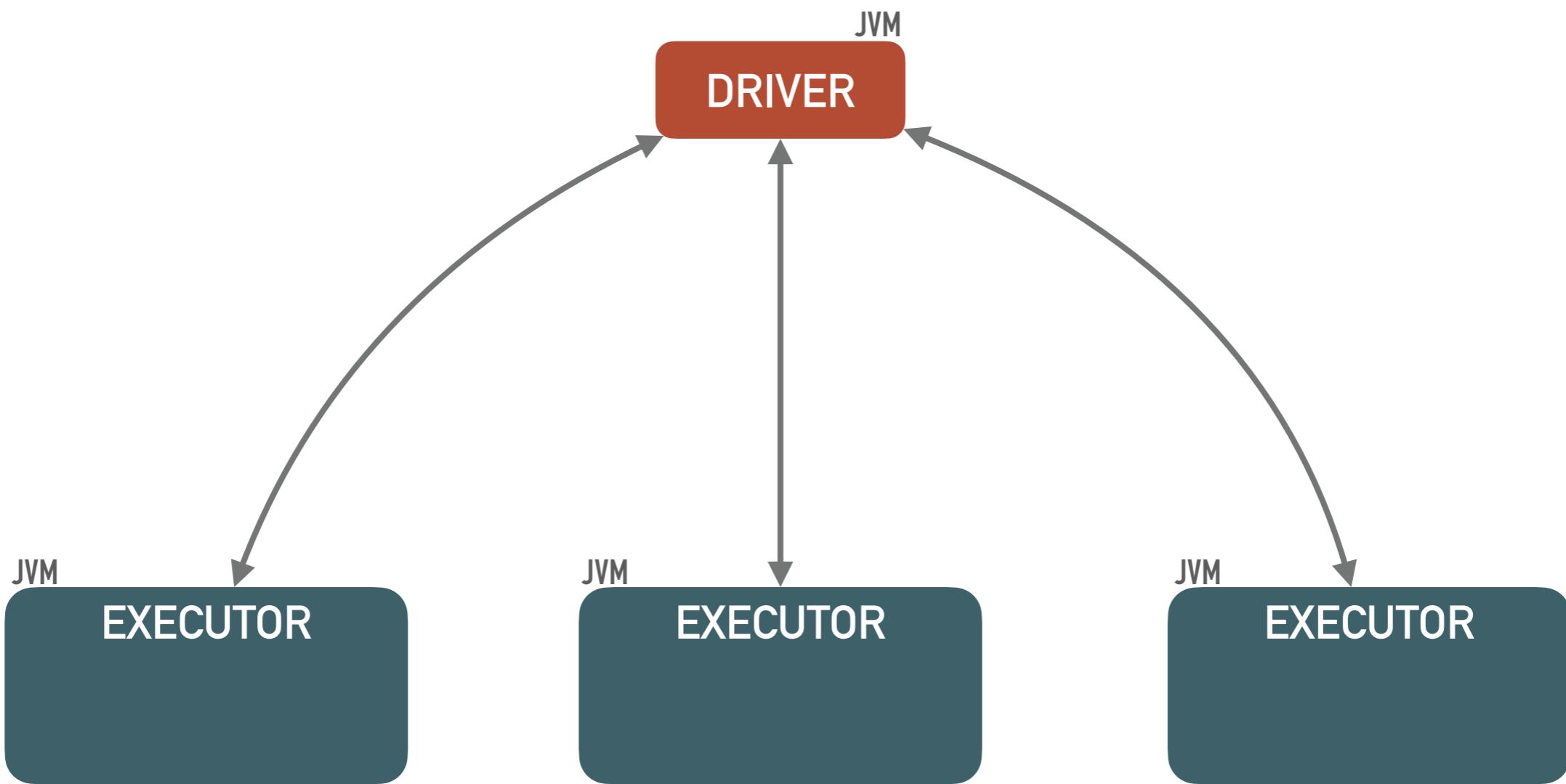
A close-up photograph of a large pile of M&M's candies. The candies are scattered across the frame in various colors: red, orange, yellow, green, blue, and brown. Each candy has a white 'M' printed on its surface. Overlaid on the center of the candies is the word "DISTRIBUTED" in a bold, white, sans-serif font.

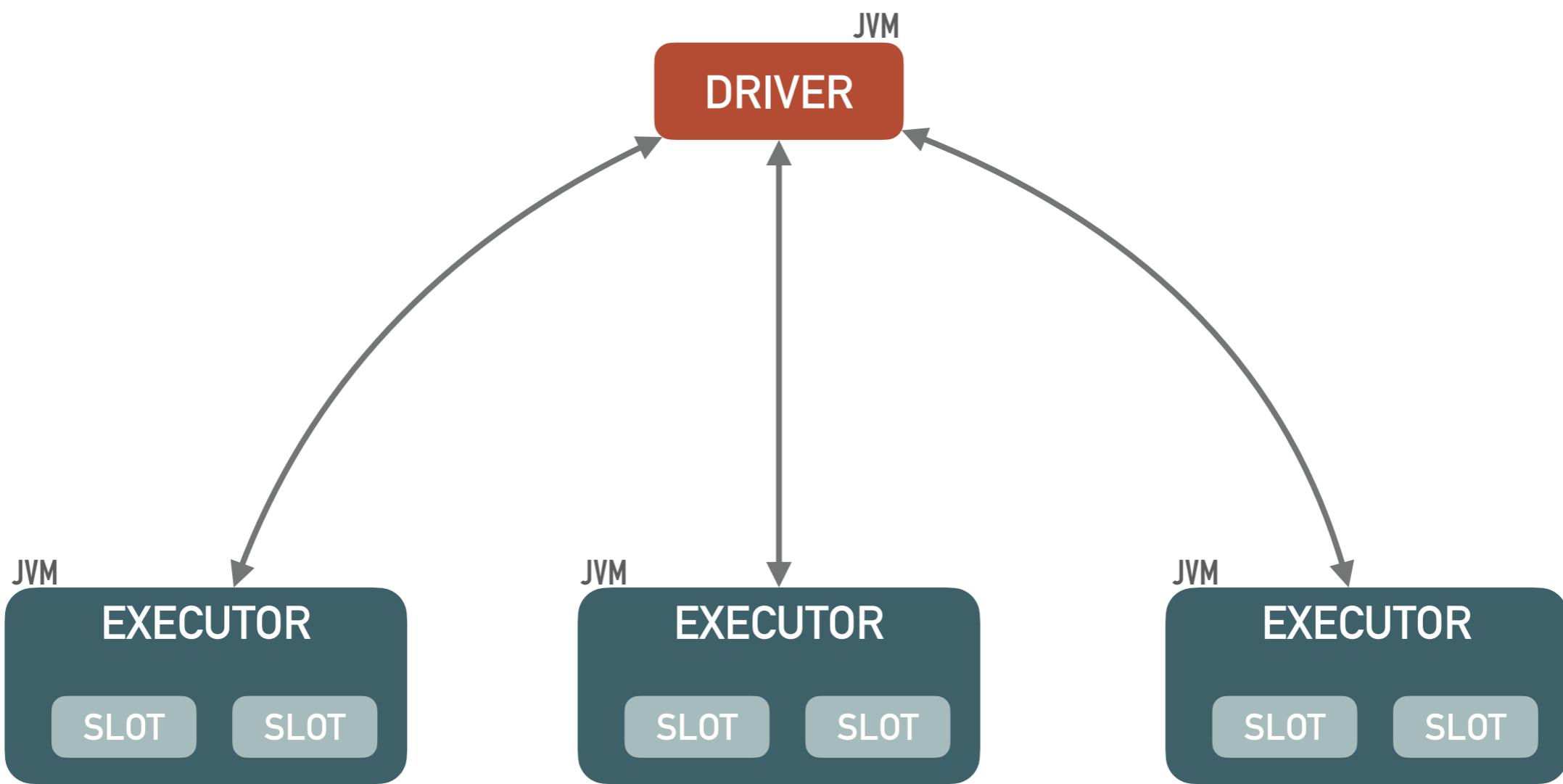
DISTRIBUTED

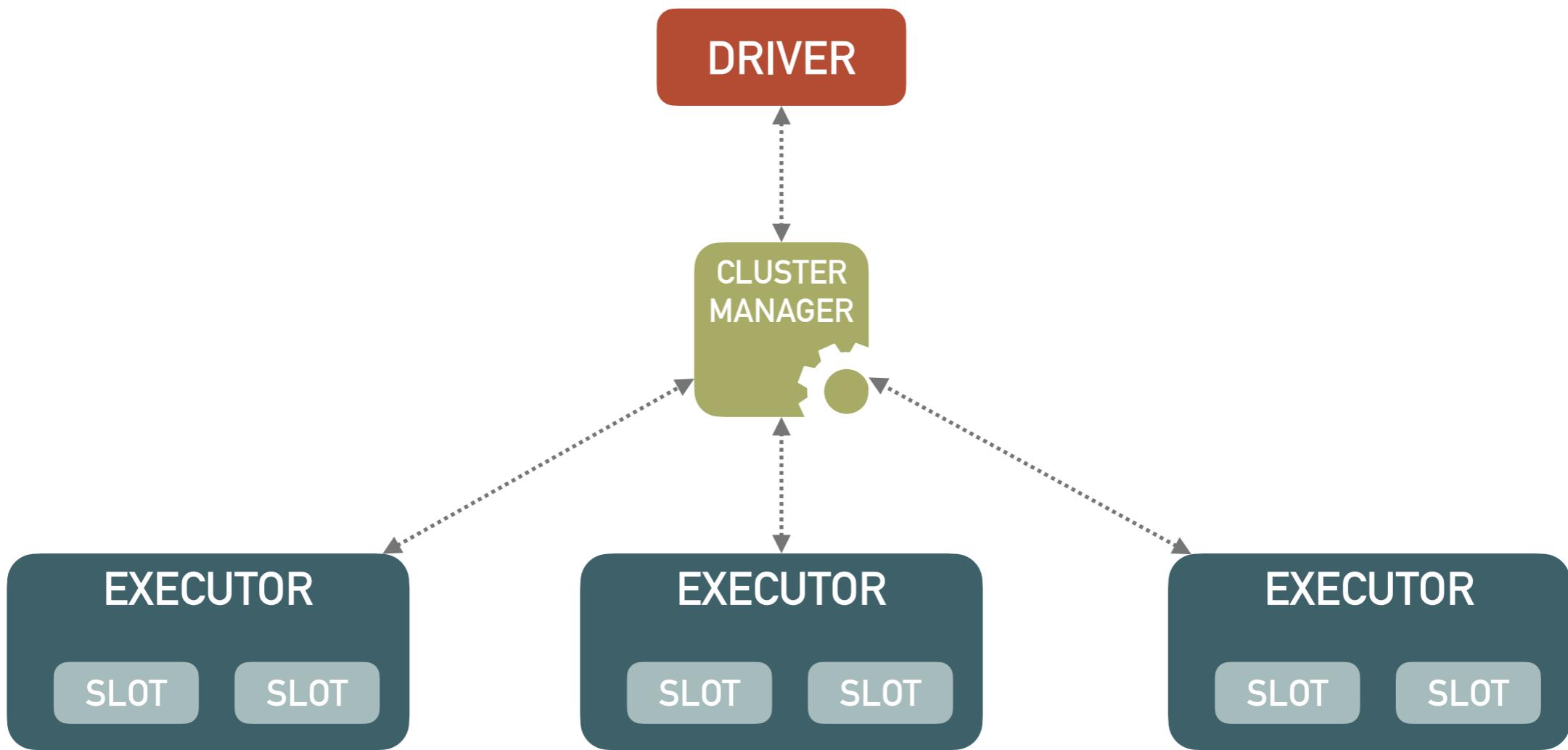
# SPARK CLUSTER

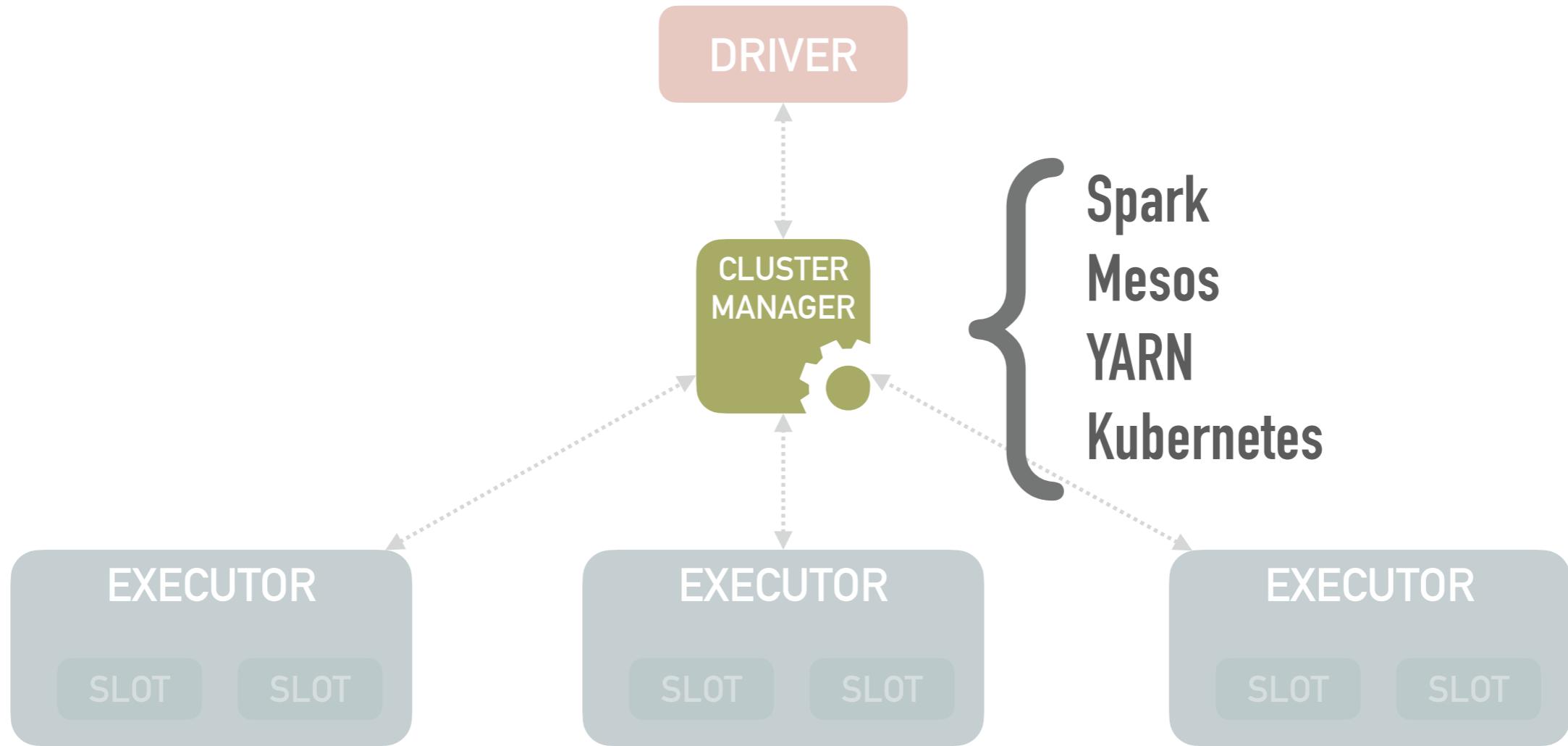


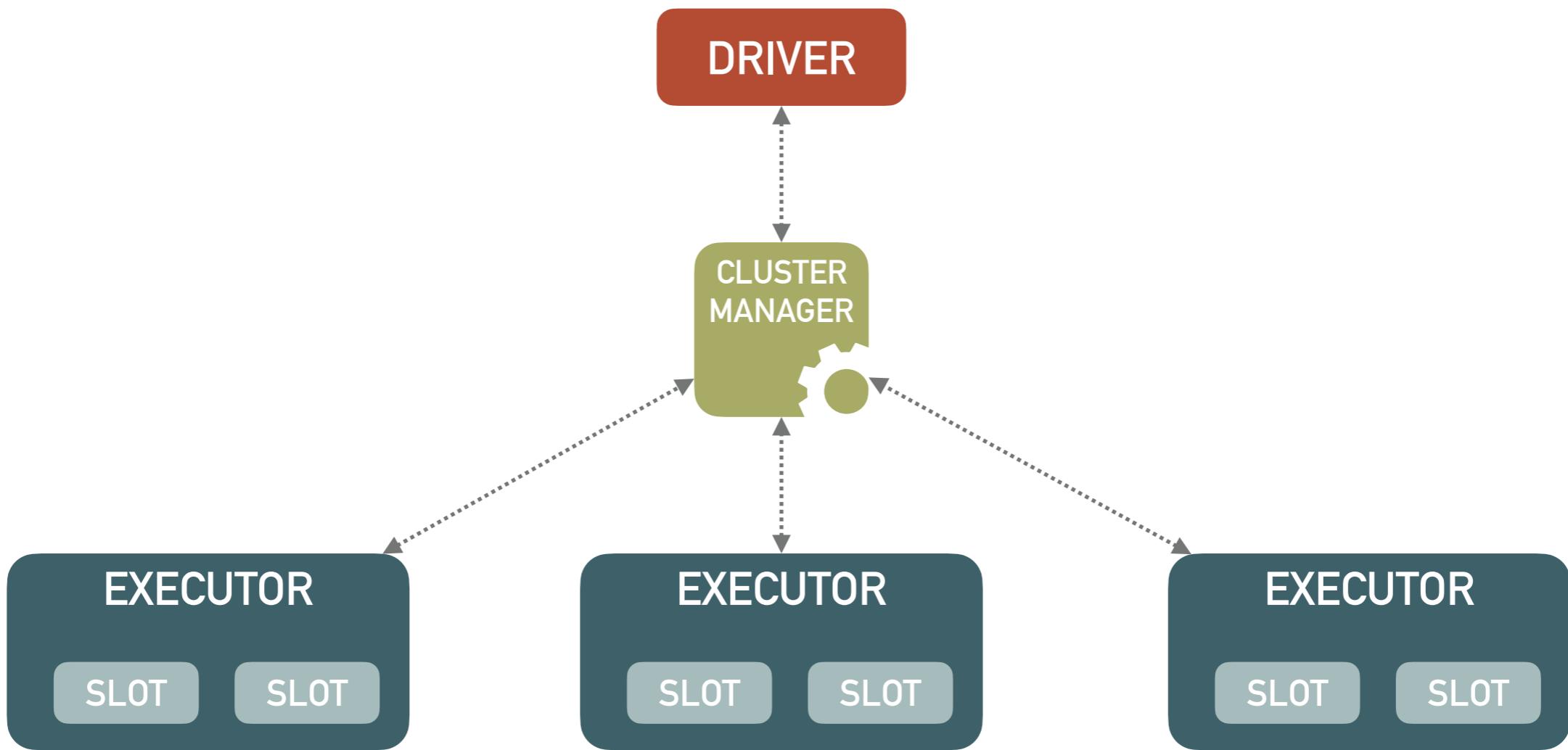






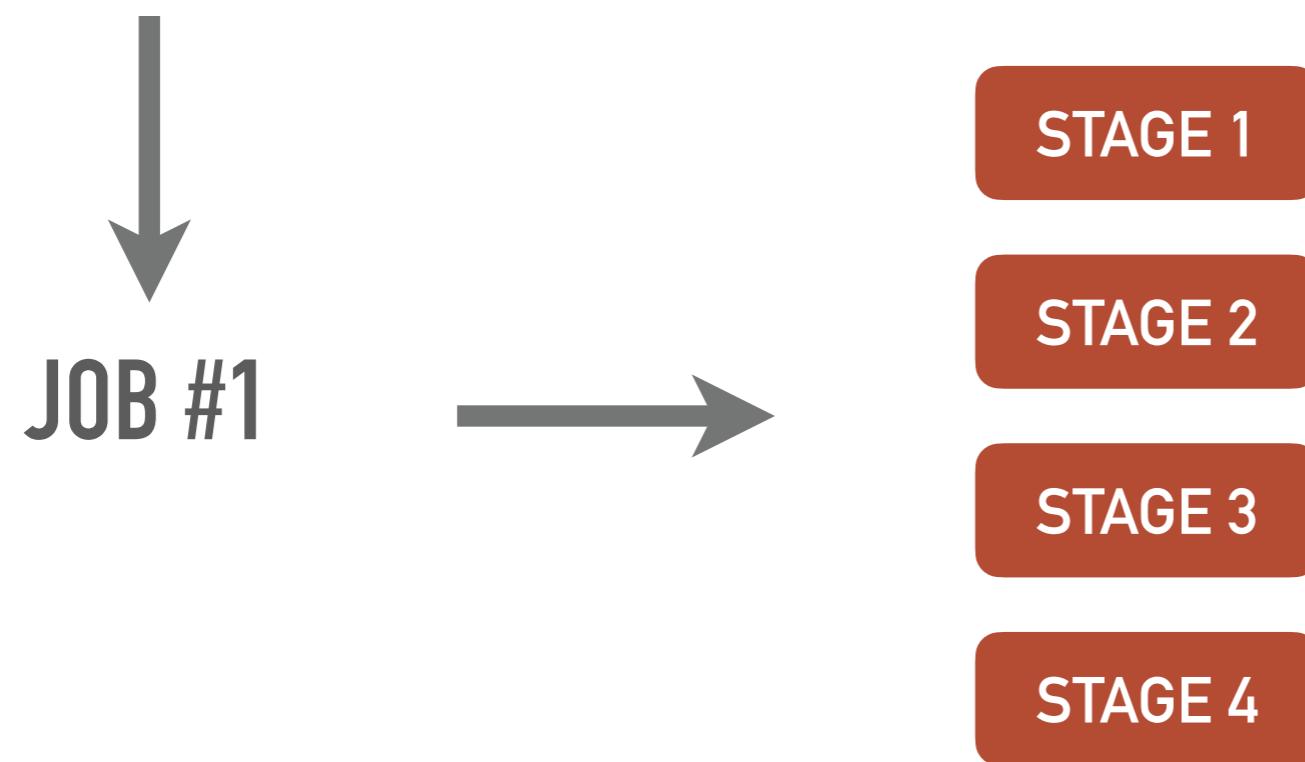


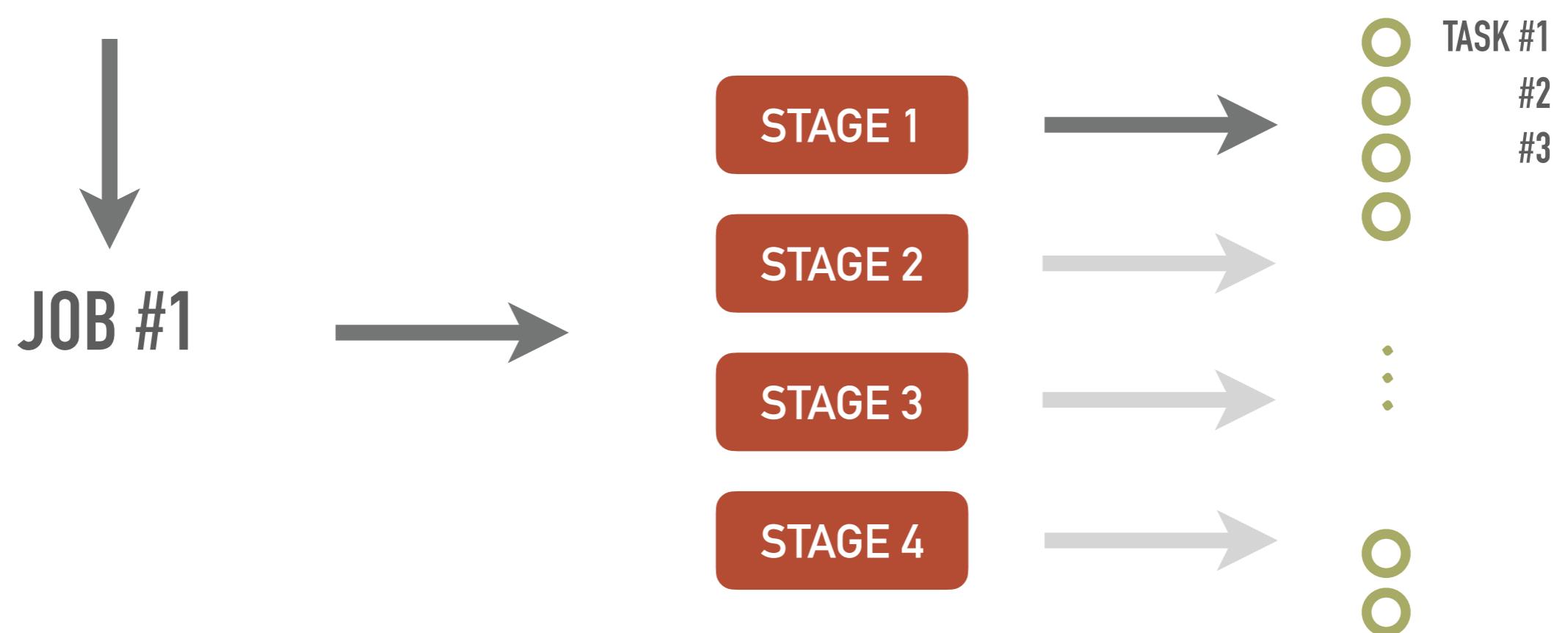


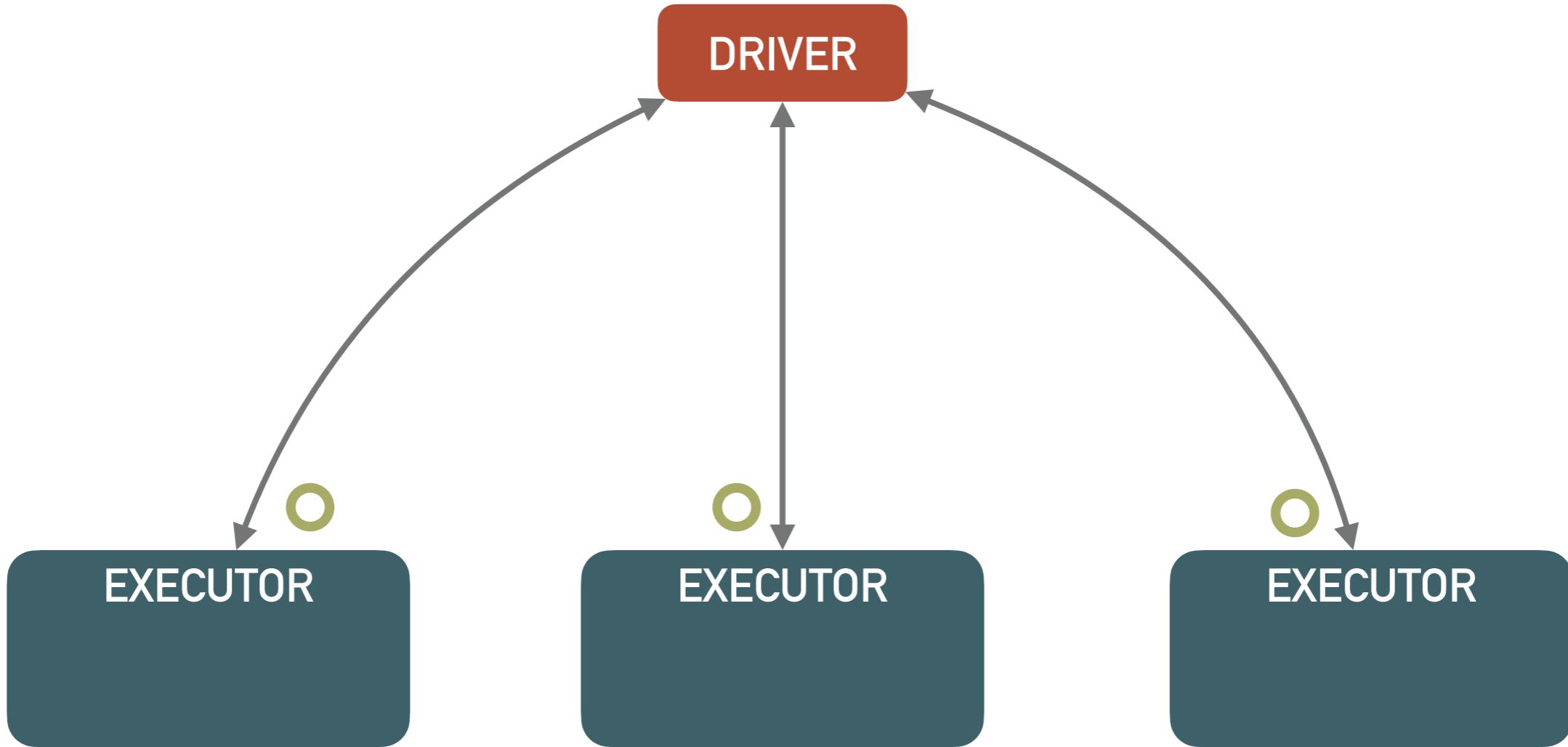


**JOB / STAGE / TASK**

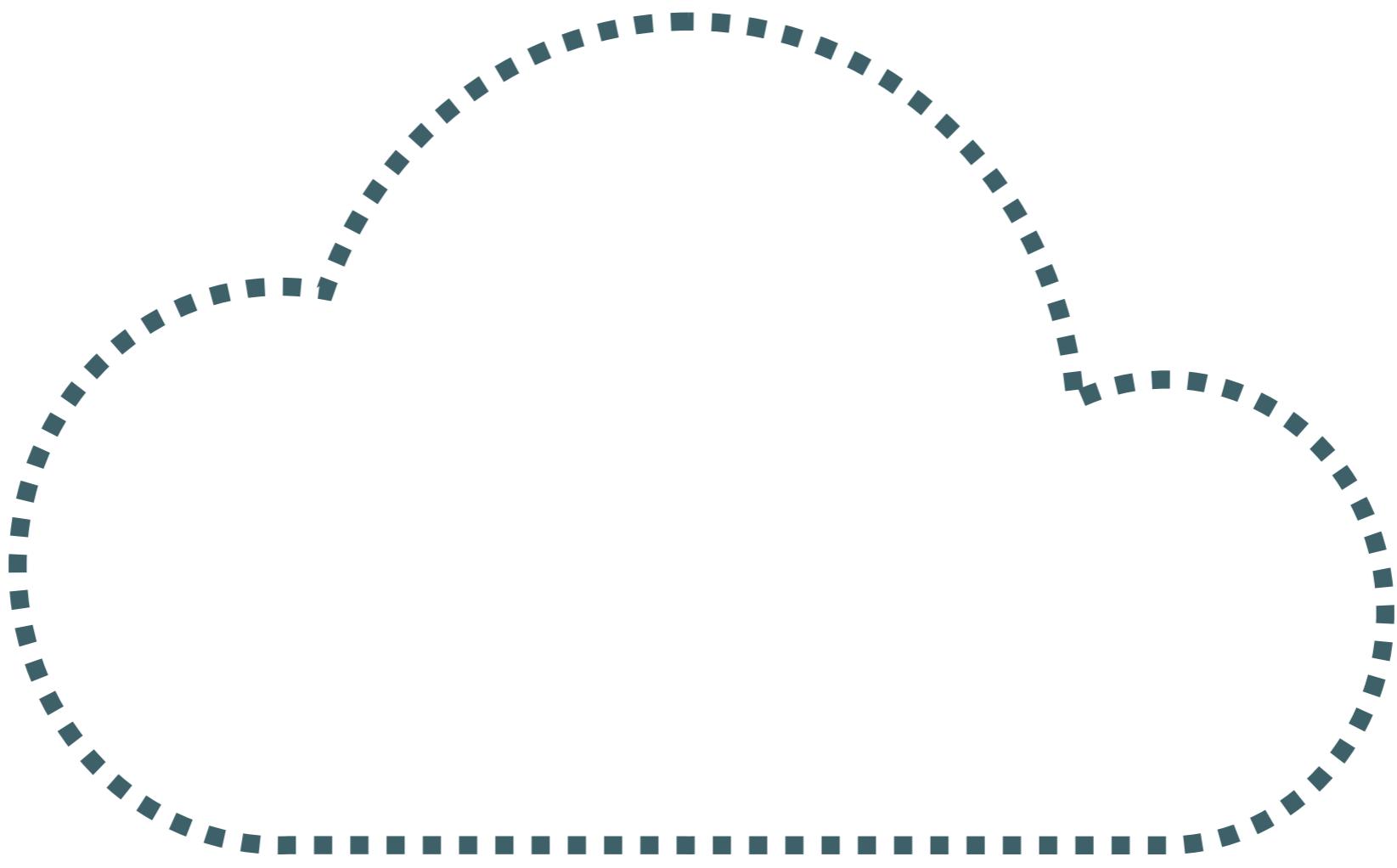
```
logs = spark.read.json("s3://logs-bucket")
logs.groupBy(logs.userId).avg(logs.latency)
```

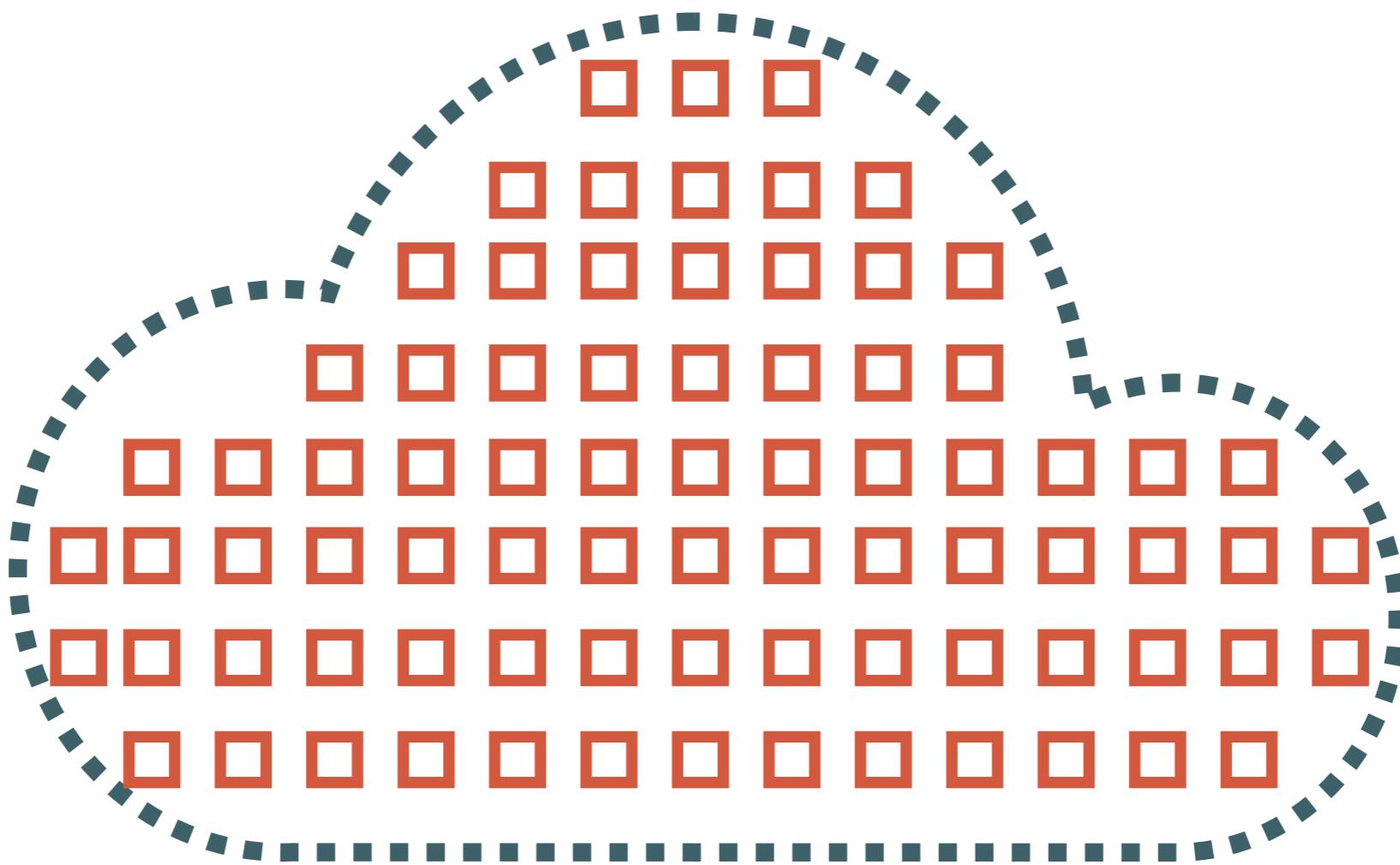




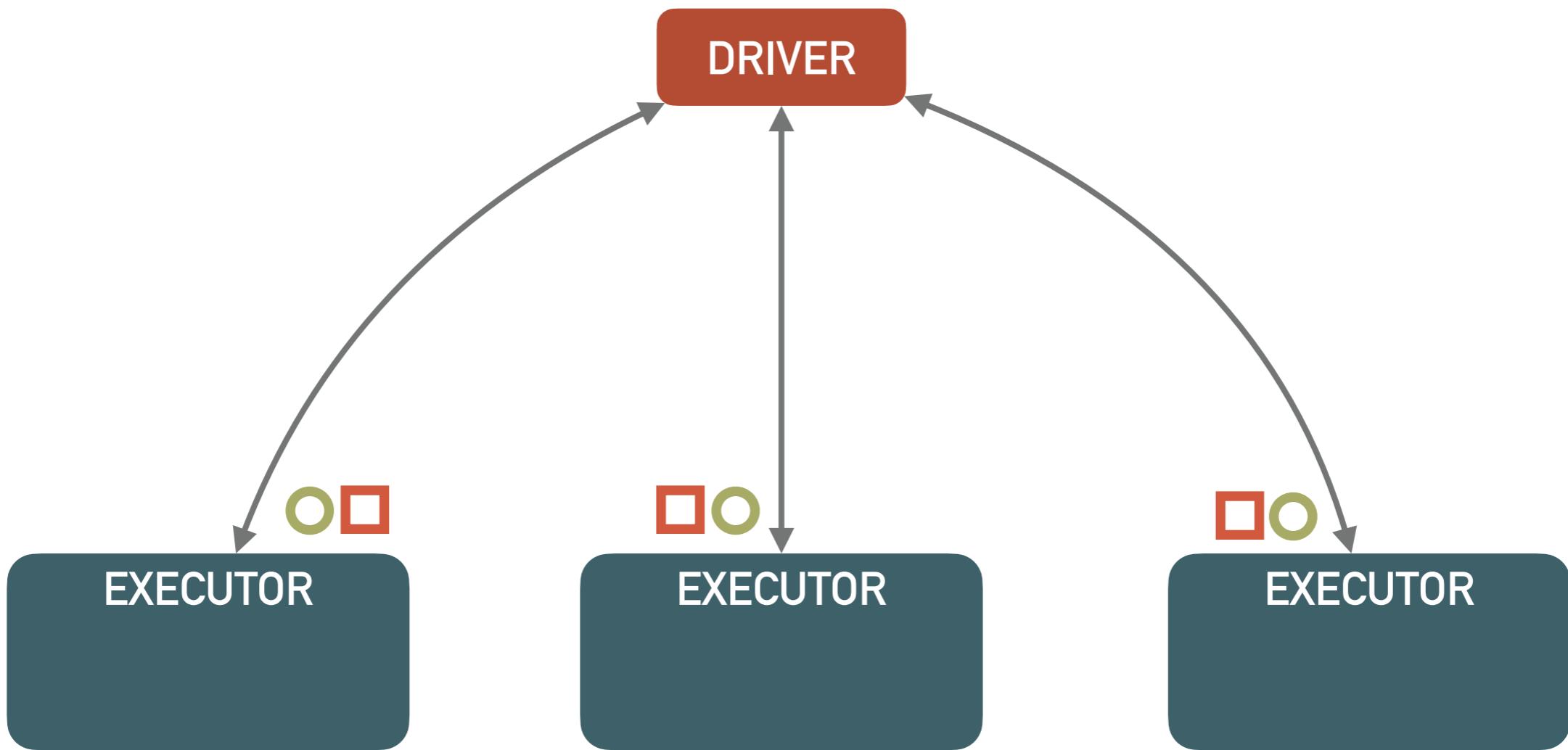


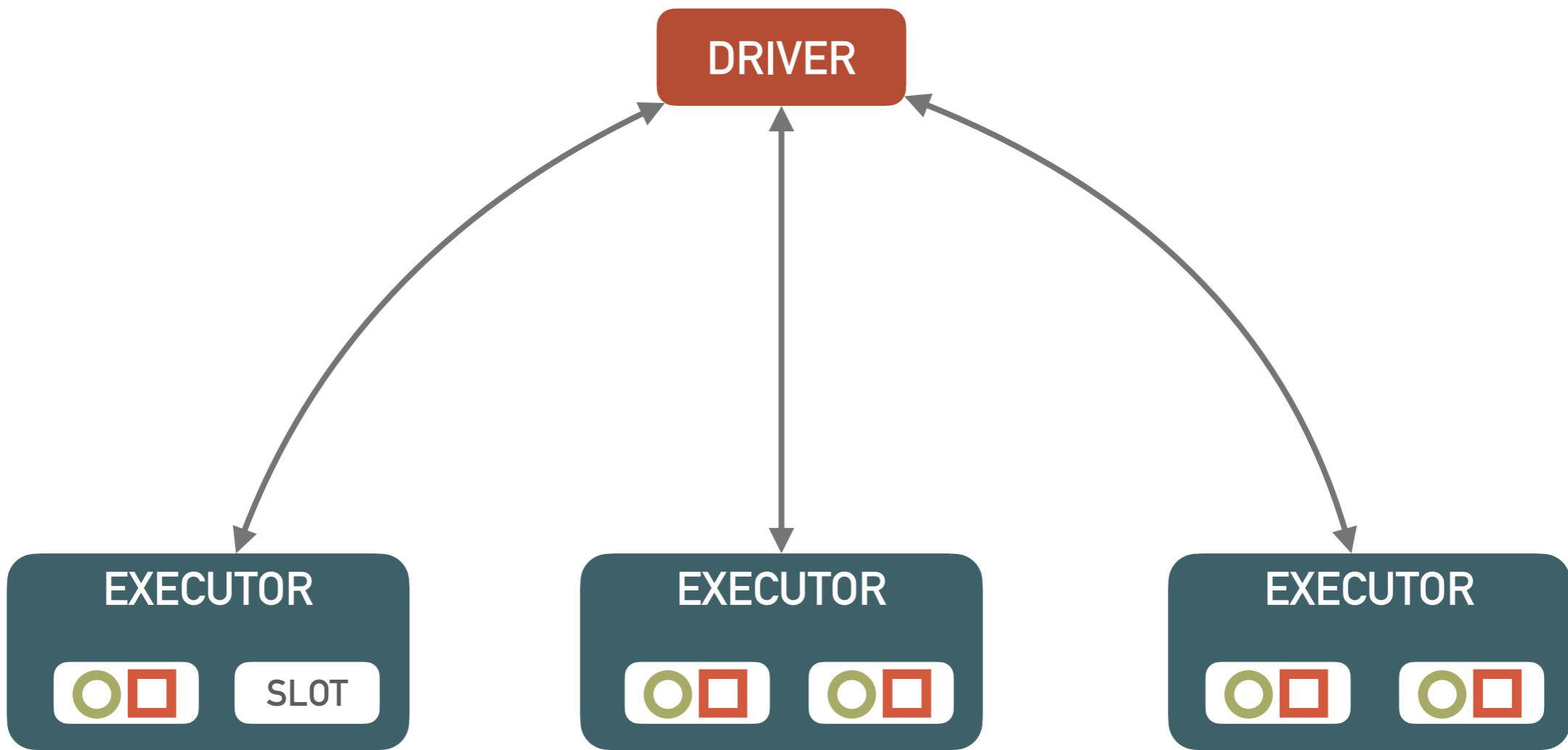
# PARTITION

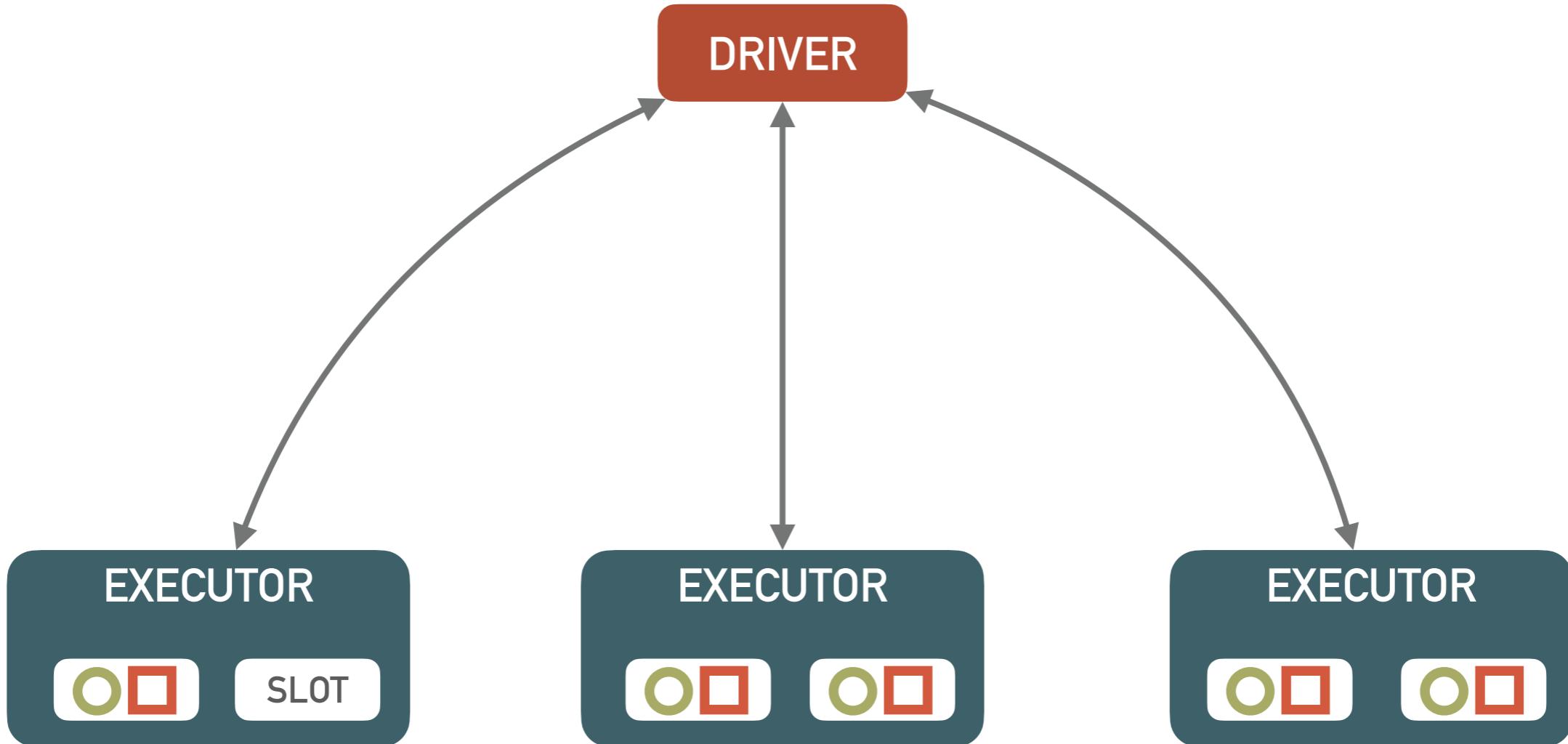












Task (what to do)

Partition (subset of data)

# **UNIFIED COMPUTING ENGINE**

---

*for large-scale distributed data processing*

**SQL**

**Streaming**

**Machine Learning**

**Graph**

Spark  
SQL

Spark  
Streaming

MLlib

GraphX

Spark Core Engine



Spark  
SQL

Spark  
Streaming

MLlib

GraphX

Spark Core Engine



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

# **UNIFIED COMPUTING ENGINE**

---

*for large-scale distributed data processing*

# 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

---

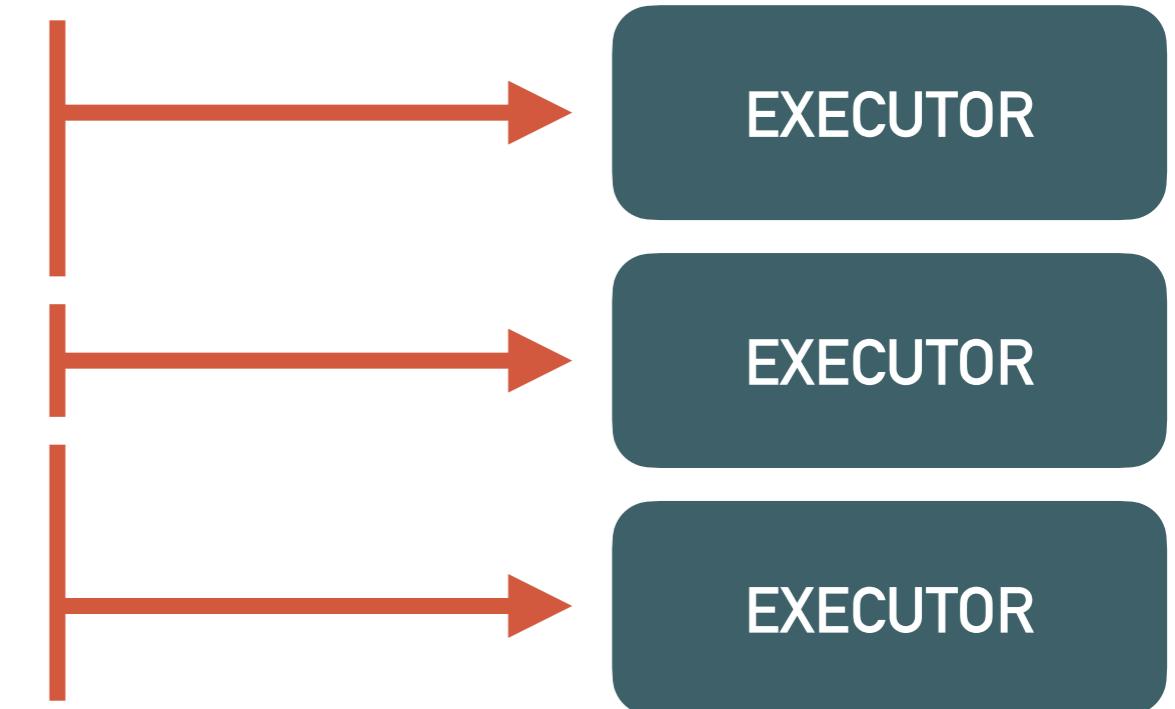
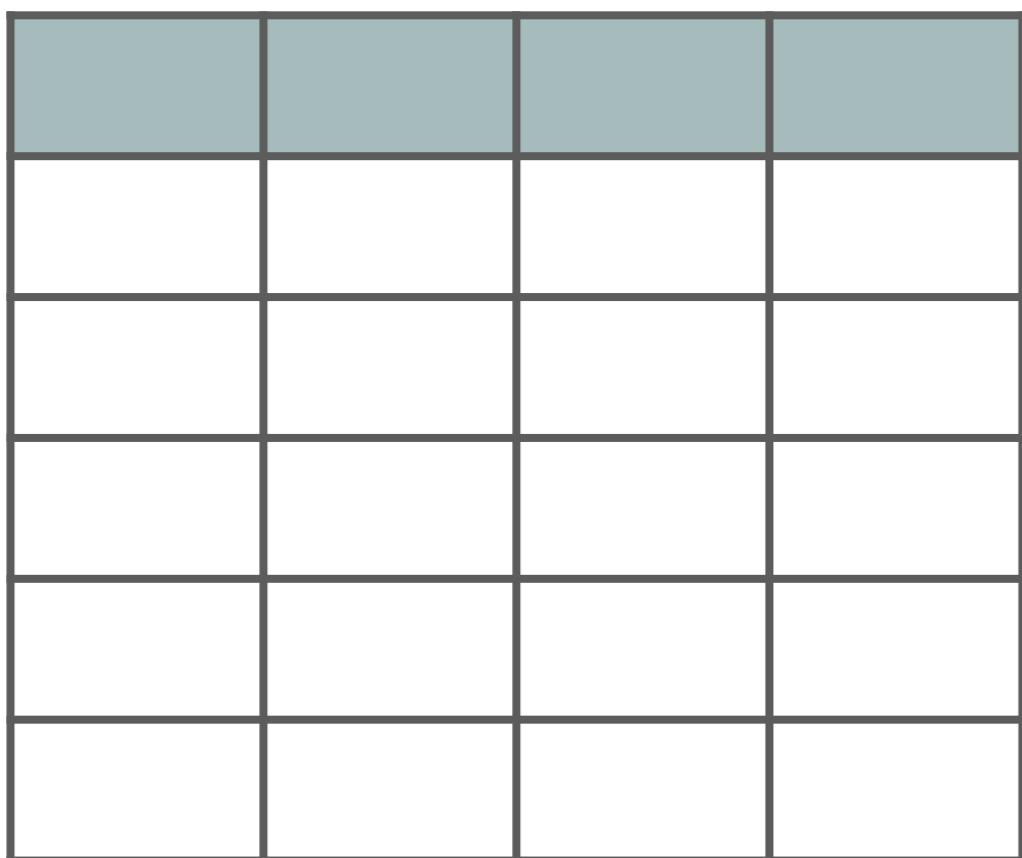
# DATAFRAME

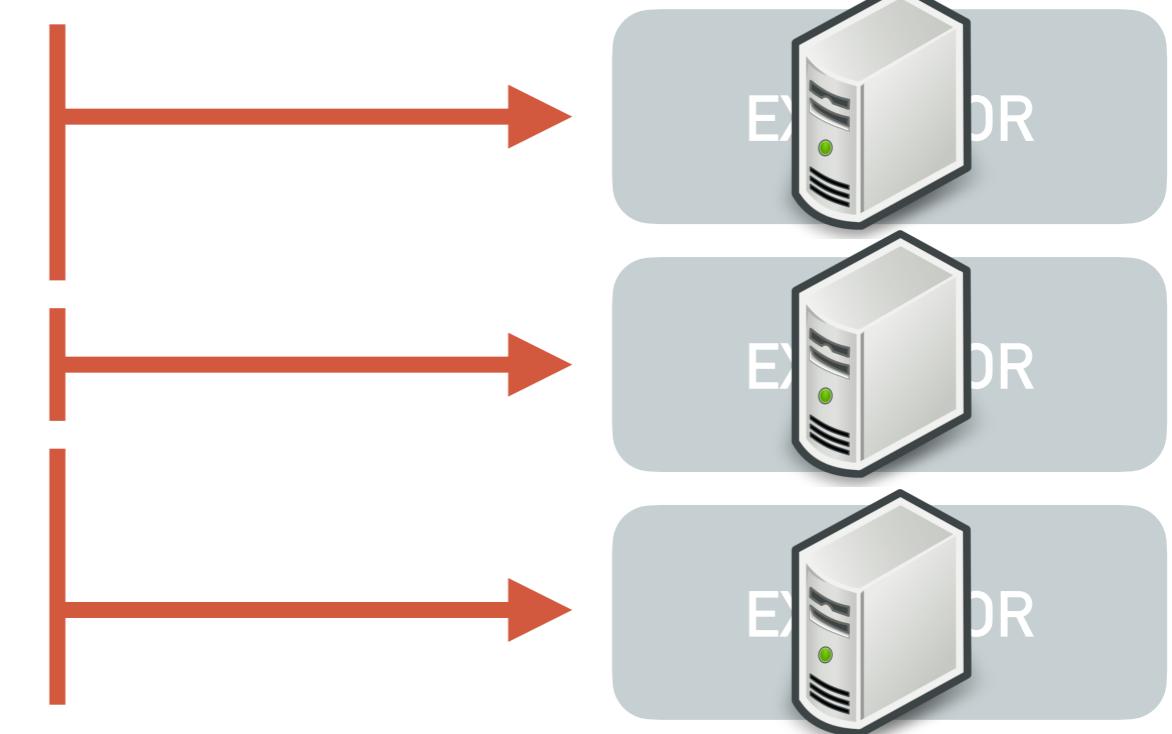
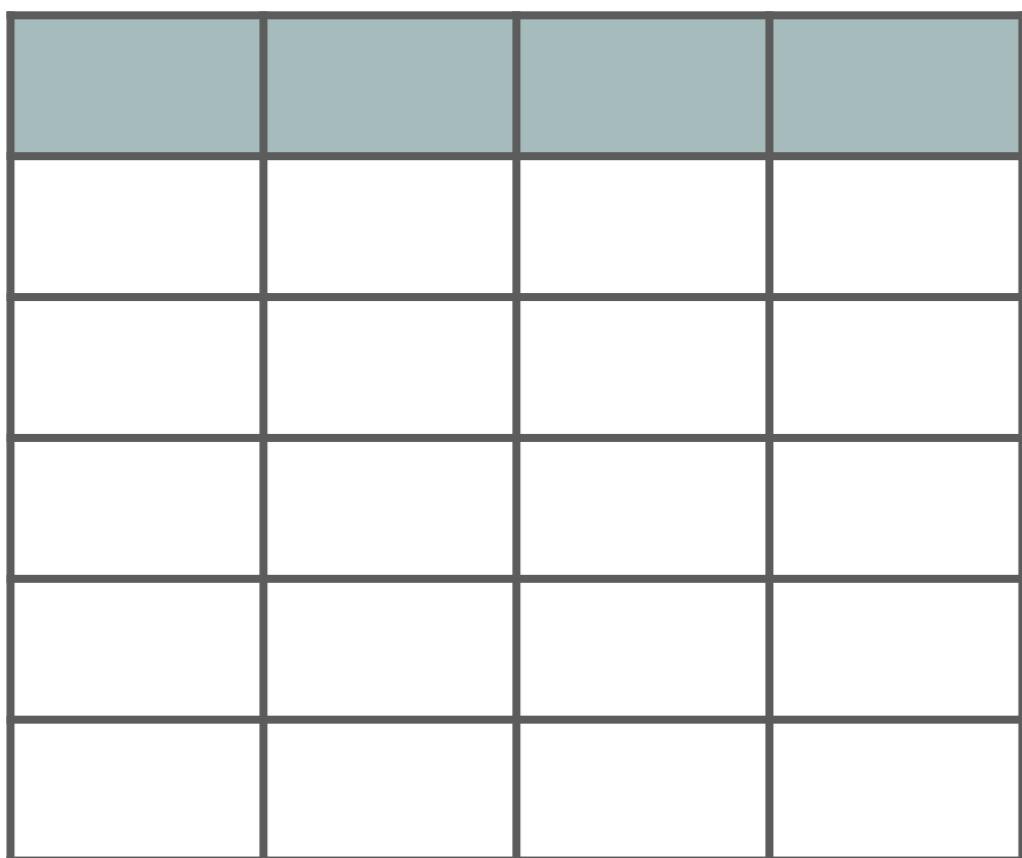
---

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

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

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



# DATAFRAME OPERATIONS

---



# DATAFRAME OPERATIONS

---



# DATAFRAME OPERATIONS

---



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

# DATAFRAME OPERATIONS

---



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

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


**logs**

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


logs

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


**logs**




**late\_logs**

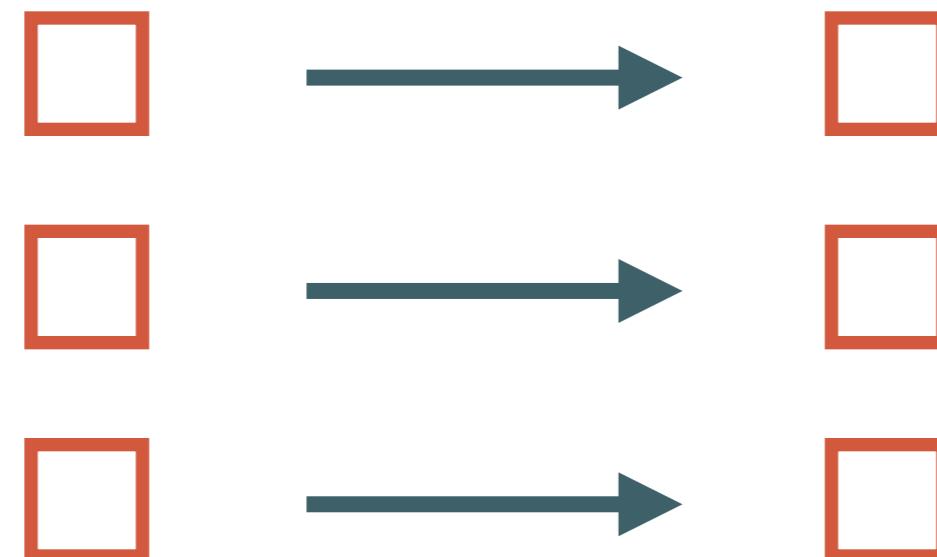
# NARROW TRANSFORMATIONS

---

One input partition contributes to one output

In-memory

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



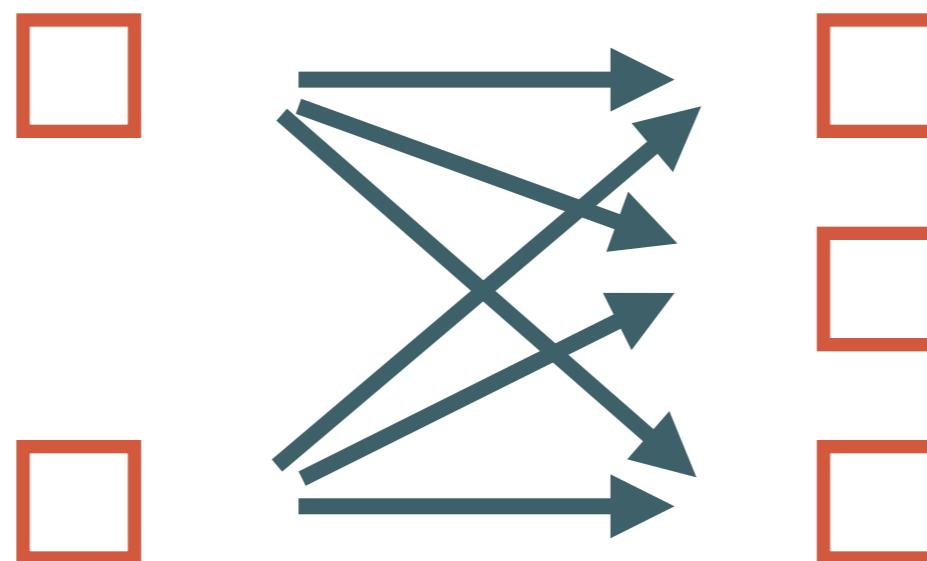
# WIDE TRANSFORMATIONS

---

One input Partition contributes to many outputs (shuffle)

Write results to disk

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



# DATAFRAME OPERATIONS

---



# DATAFRAME OPERATIONS

---

- **count**
- **collect**
- **take**
- **save**
- ...



# DATAFRAME OPERATIONS

---



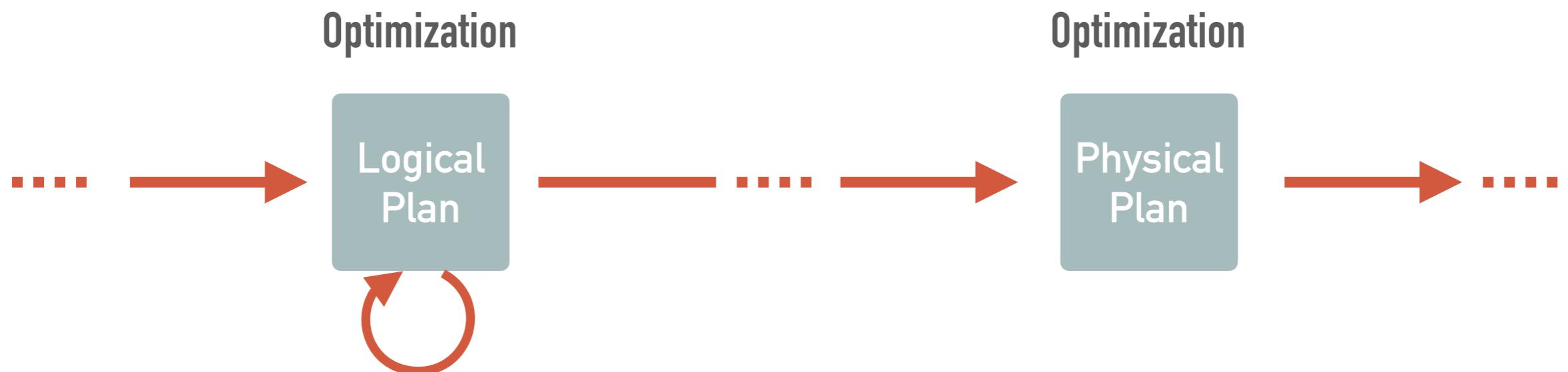
```
logs = spark.read.json("s3a://logs-bucket")
logs.groupBy(logs.userId)
    .avg(logs.latency)
```

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

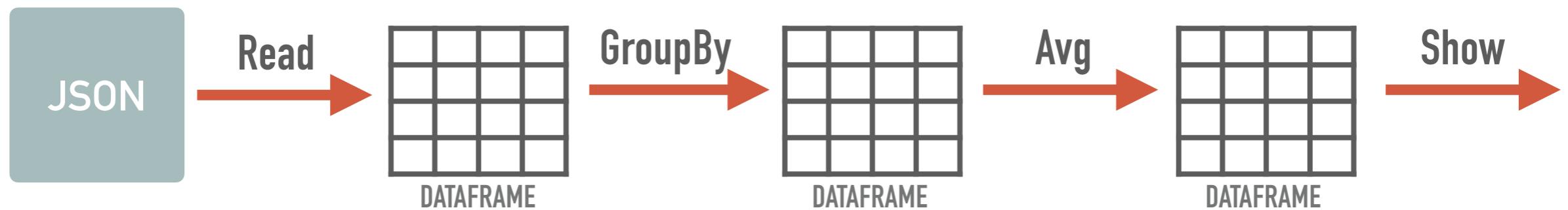
```
logs = spark.read.json("s3a://logs-bucket")  
logs.groupBy(logs.userId)  
    .avg(logs.latency)  
    .show()
```



```
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

# SPARK 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

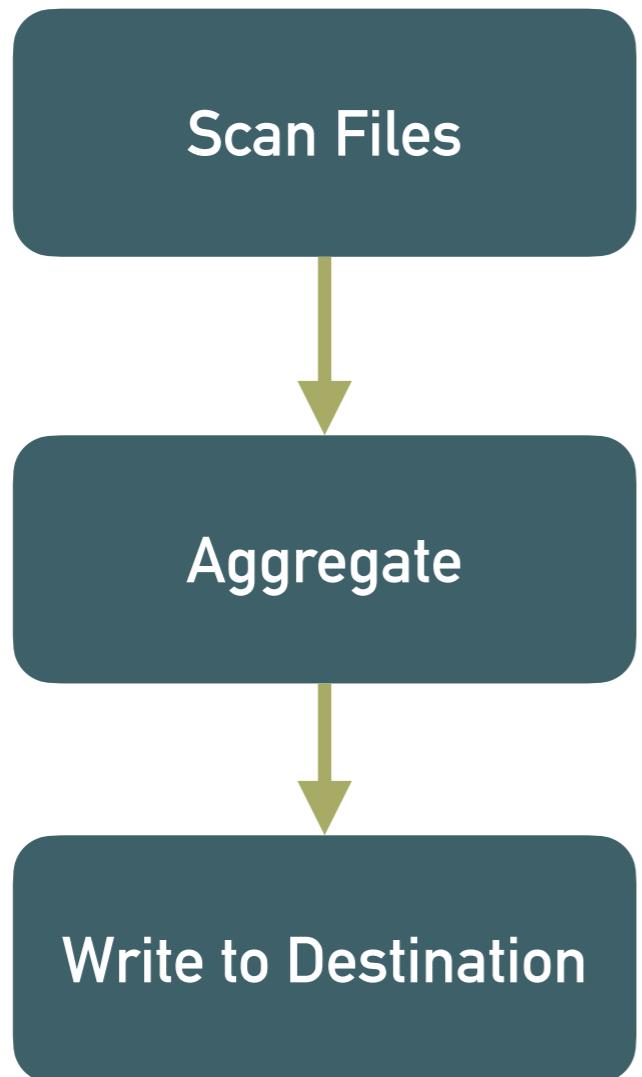
---

# PROBLEM

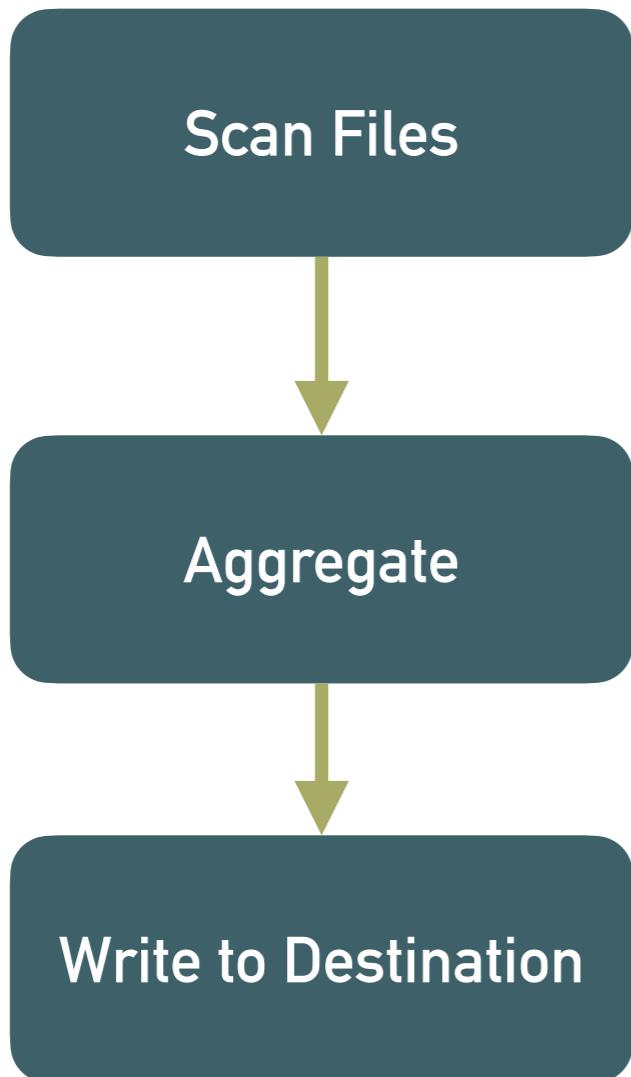
---

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

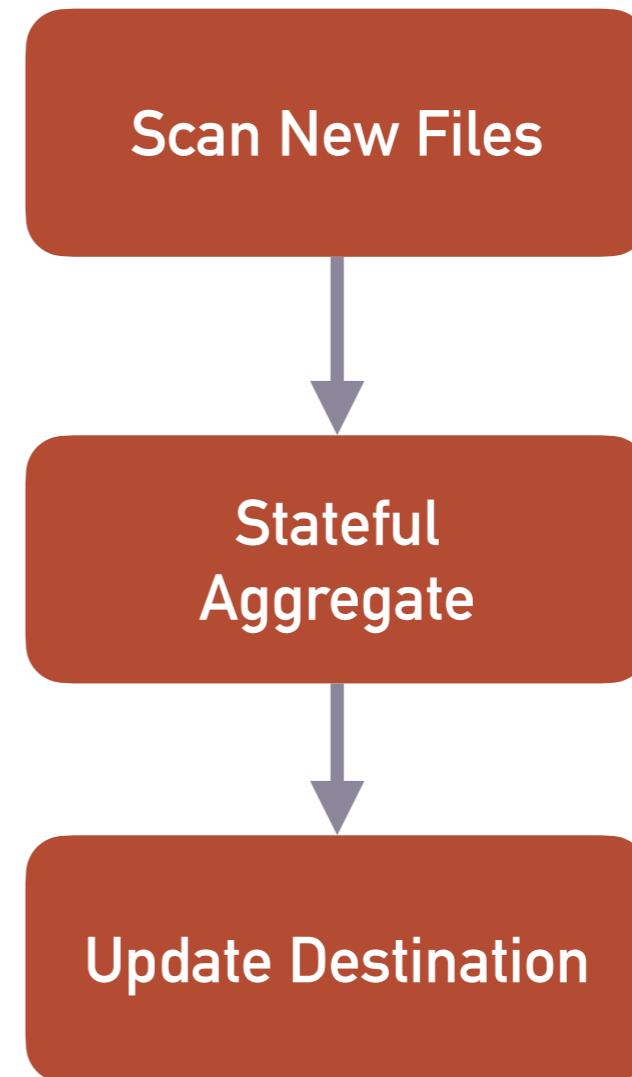
# Batch



# Batch



# Continuous



# 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 input sources
- Abstracting output sources

# **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)
- Inside the code (Local mode)
- Docker
- DataBricks + Community Edition

# DEMO

---

# QUESTIONS?

# WE ARE HIRING!

**RED | VENTURES**