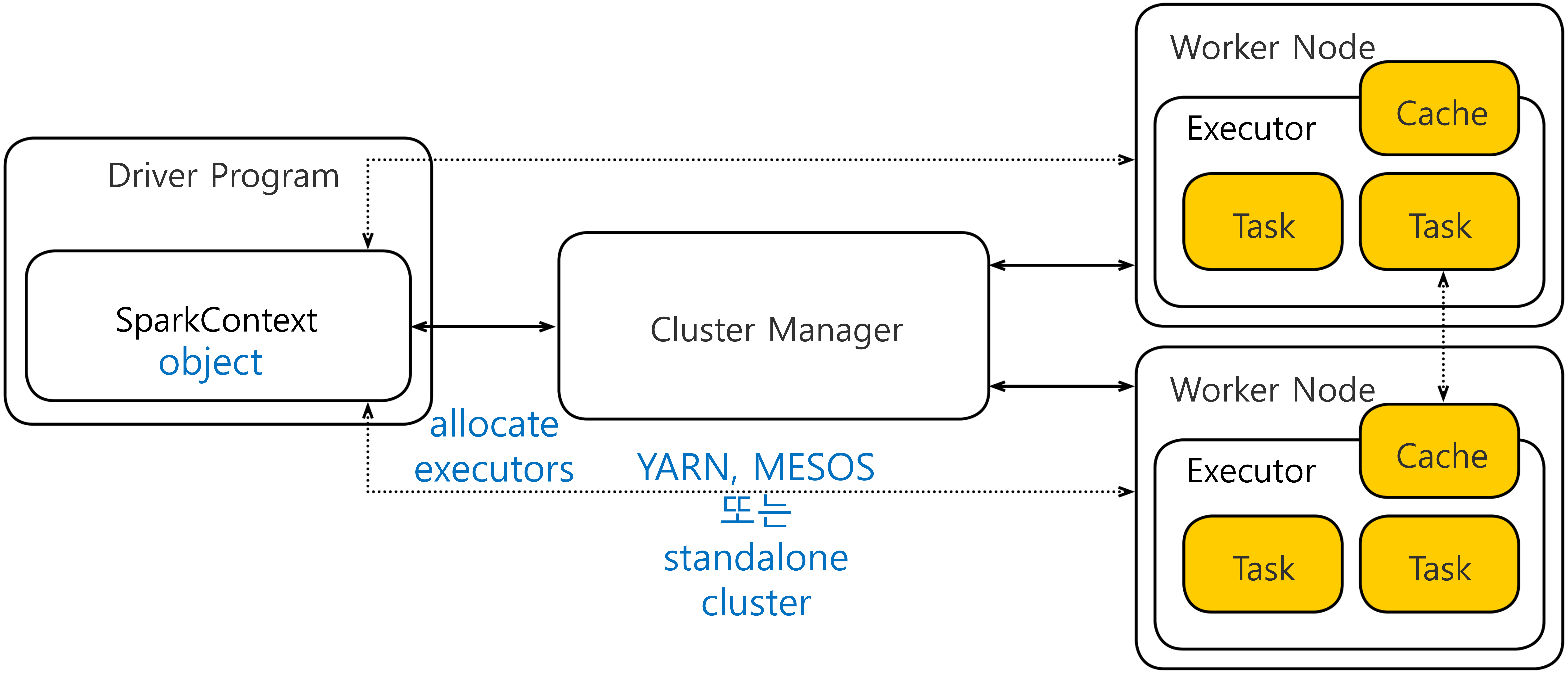
Yandex

# Execution & scheduling

# SparkContext

- Tells your application how to access a cluster
- Coordinates processes on the cluster to run your application



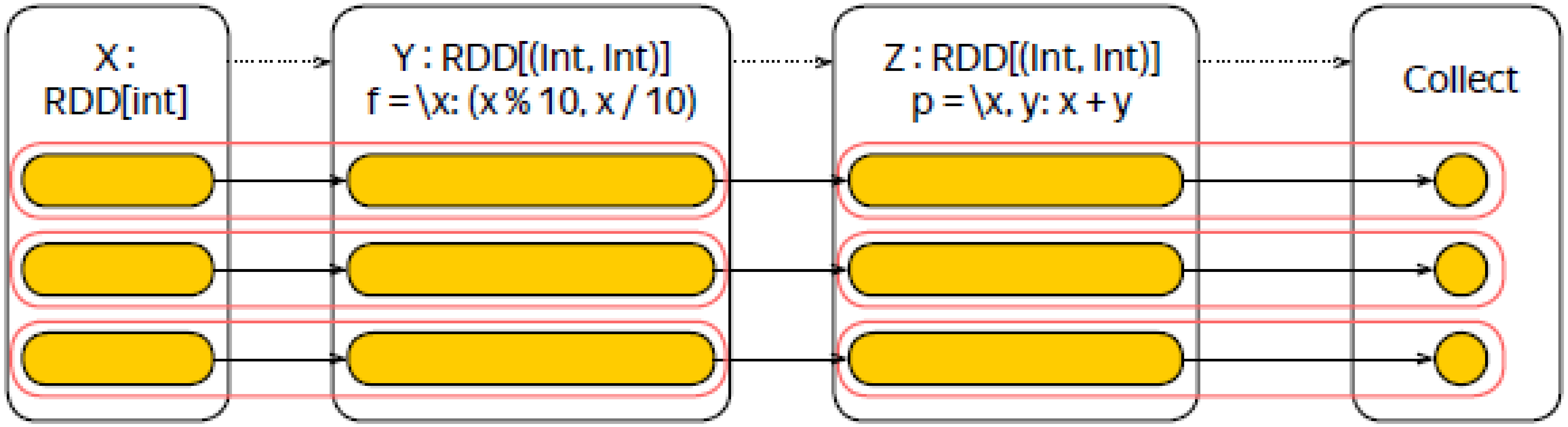
# Jobs, stages, tasks

- Task is a unit of work to be done
  - Tasks are created by a job scheduler for every job stage
- Job is spawned in response to a Spark action
  - Job is divided in smaller sets of tasks called stages

# Jobs, stages, tasks (example)

```
›› Z = X
   .map(lambda x: (x % 10, x / 10))
   .reduceByKey(lambda x, y: x + y)
   .collect()
```

1. Invoking an action… collect()
2. …spawns the job…
3. … that gets divided into the stages by the job scheduler…
4. …and tasks are created for every job stage.



# Jobs, stages, tasks

Stage는 파이프라인을 위한 구조

- Job stage is a pipelined computation spanning between materialization boundaries
  - not immediately executable RDD Level
- Task is a job stage bound to particular partitions
  - immediately executable Partition Level

›› Materialization happens when reading, shuffling or passing data to an action Materializatoin == building

- narrow dependencies allow pipelining
- wide dependencies forbid it

# SparkContext의 역할

- Tracks liveness of the executors
  - required to provide fault-tolerance

- Schedules multiple concurrent jobs
  - to control the resource allocation within the application

- Performs dynamic resource allocation
  - to control the resource allocation between different applications

# Summary

›› The SparkContext is the core of your application

›› The driver communicates directly with the executors

›› Execution goes as follows:
Action ➞ Job ➞ Job Stages ➞ Tasks

›› Transformations with narrow dependencies allow pipelining

# Caching & persistence

Intermediate Data

# Quick reminder

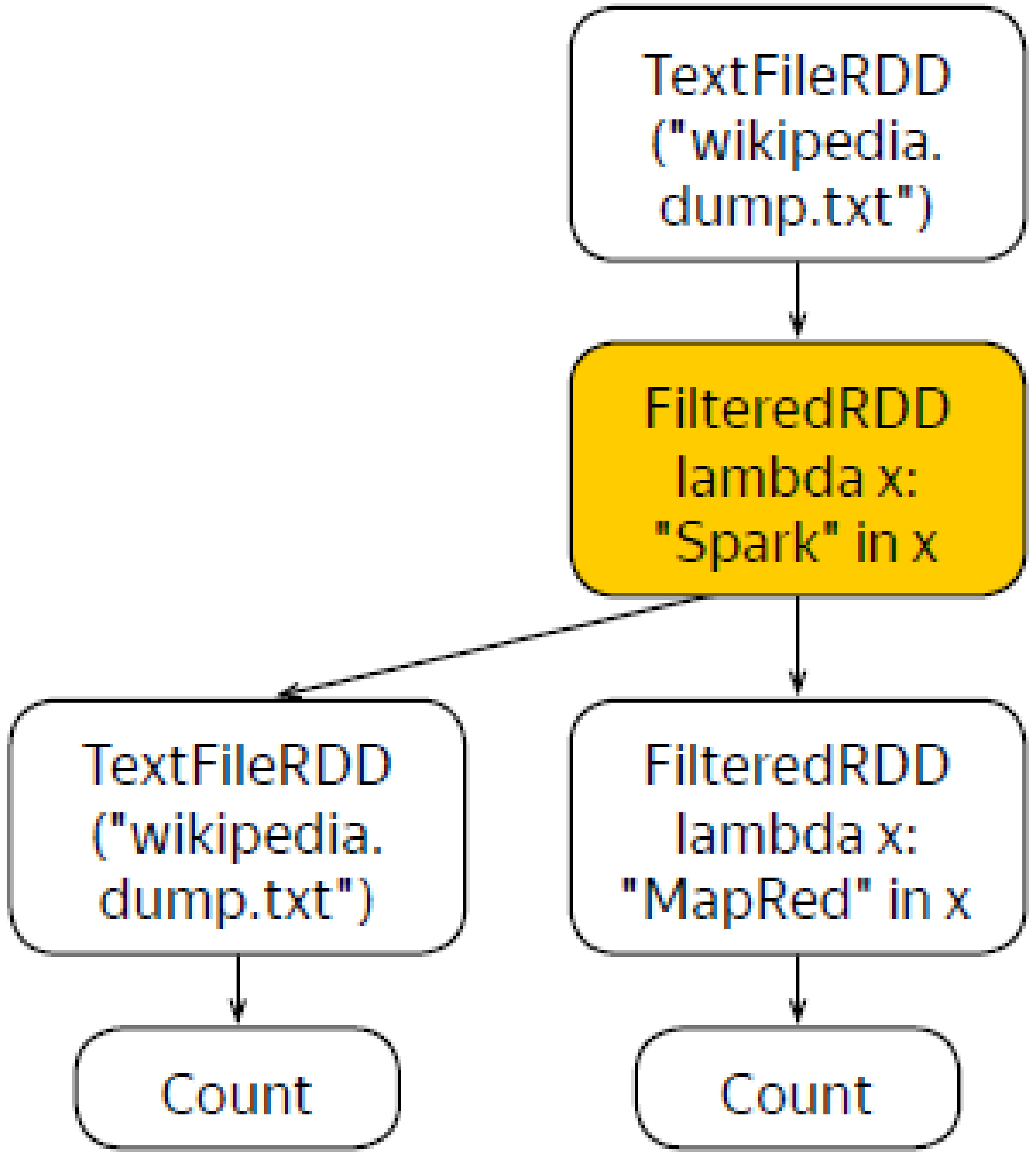
›› RDDs are partitioned

›› Execution is build around the partitions

›Block is a unit of input and output in Spark

# Motivating example

```
sc = SparkContext(…)
wiki = sc.textFile("wikipedia.dump.txt")

spark_articles = wiki.filter(
lambda x: "Spark" in x)

spark_articles.cache()

hadoop_articles = spark_articles.filter(
lambda x: "Hadoop" in x)
mapreduce_articles = spark_articles.filter(
lambda x: "MapRed" in x)
print(hadoop_articles.count())
print(mapreduce_articles.count())
```



TextFileRDD ("wikipedia. dump.txt")

FilteredRDD lambda x: "Spark" in x

TextFileRDD ("wikipedia. dump.txt")

FilteredRDD lambda x: "MapRed" in x

Count

Count

메모리에 RDD를 cache로 저장하면 불필요한 작업을 줄일 수 있다.
(같은 작업 반복 X )

# Controlling persistence level

›› rdd.persist(storageLevel)

››sets RDD's storage to persist across operations after it is computed for the first time

››storageLevel is a set of flags controlling the persistence, typical values are

DISK_ONLY

– save the data to the disk,

MEMORY_ONLY

– keep the data in the memory

MEMORY_AND_DISK

– keep the data in the memory; when out of memory – save it to the disk

DISK_ONLY_2, MEMORY_ONLY_2, MEMORY_AND_DISK_2

– same as about, but make two replicas ← improves failure recovery times!

›› rdd.cache() = rdd.persist(MEMORY_ONLY)

cache함수는 메모리에 저장한다는 함수의 shortcut이다.

# Best practices

일반적인 데이터 persist 방법

- For interactive sessions
  - cache preprocessed data

- For batch computations
  - cache dictionaries
  - cache other datasets that are accessed multiple times

- For iterative computations
  - cache static data

- And do benchmarks!

# Summary

- Performance may be improved by persisting data across operations
  - in interactive sessions, iterative computations and hot datasets

- You can control the persistence of a dataset
  - whether to store in the memory or on the disk
  - how many replicas to create

# Broadcast variables

Shared Data

# Broadcast variable

›› Broadcast variable is a read-only variable that is efficiently shared among tasks

›› Distribution is done by a torrent-like protocol (extremely fast!)

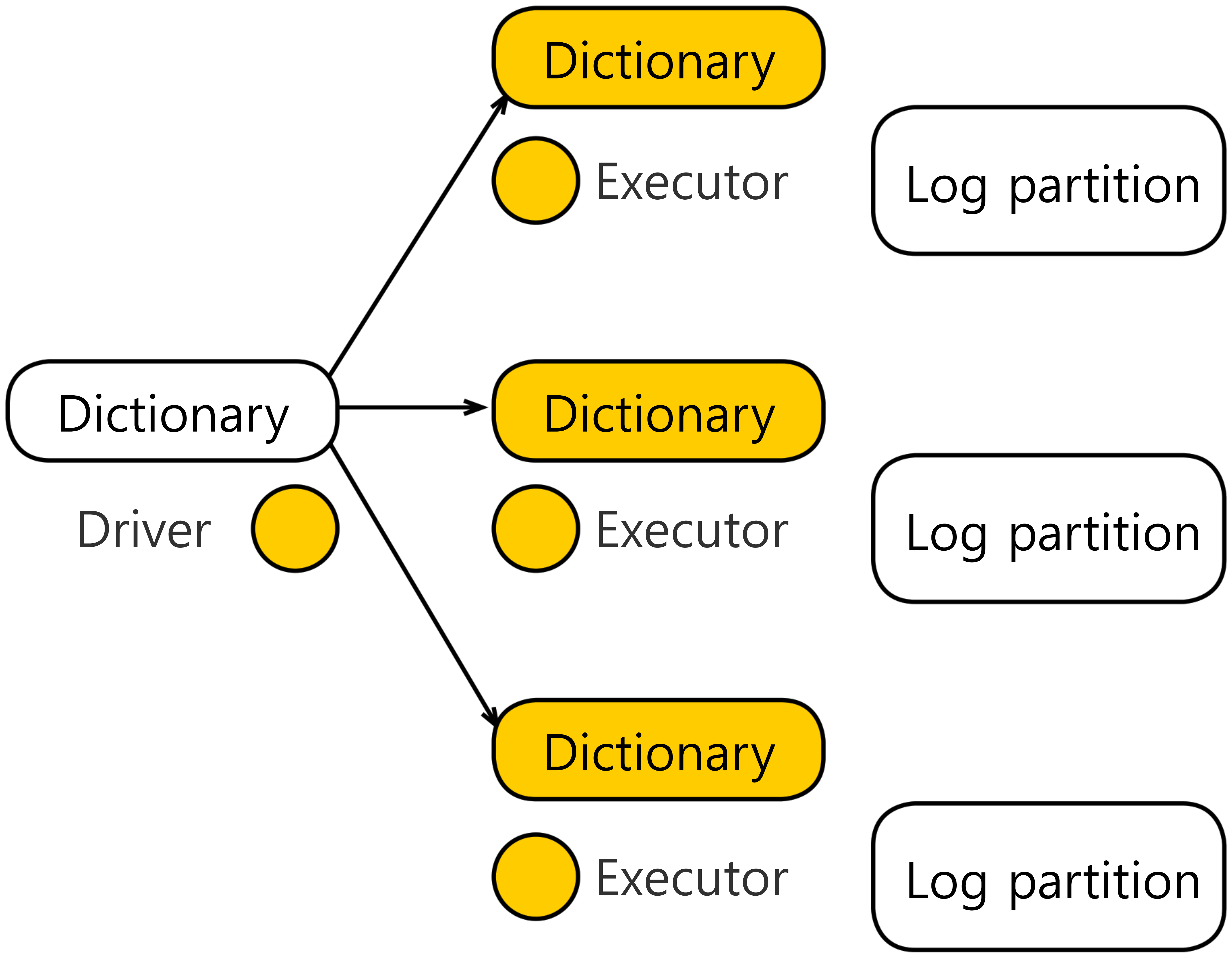
›› Distributed efficiently compared to captured variables

일반 variable을 closure에 넣으면 1 to many protocol
( 한 곳에서 많은 executor로 전달해야 함 )

Broadcast variable은 many to many protocol (토렌트와 같은 방식)

# Motivating example

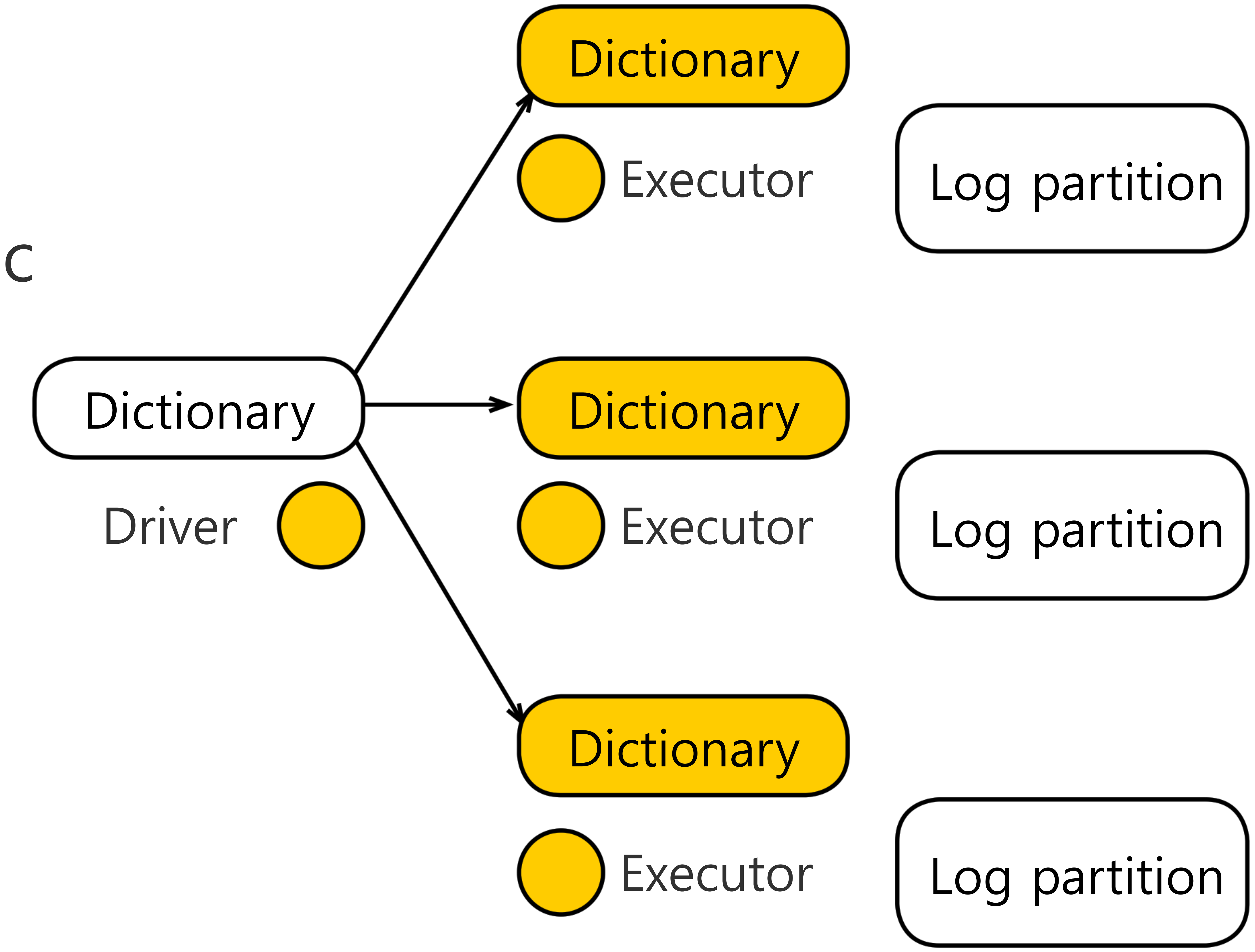
- Input:
  1TB partitioned log, 1GB IP dictionary
- Task:
  resolve IP addresses
- Idea:
  distribute the dictionary
  query it locally



# Motivating example

Serial distribution via the closure
(from the driver to every executor)
~1000 (tasks) * 1GB = 1TB of traffic

1GB 데이터를 driver가 모두 전달



Parallel distribution via
the broadcast variable
(torrent-like)
~1-2 GB of traffic

Faster!

# Motivating example 2

```
sc = SparkContext(conf=...)

# compute the dictionary
my_dict_rdd = sc.textFile(...).map(...).filter(...)
my_dict_data = my_dict_rdd.collect()

# distributed the dictionary via the broadcast variable
broadcast_var = sc.broadcast(my_dict_data)

# use the broadcast variable within the task
my_data_rdd = sc.textFile(...).filter(
lambda x: x in broadcast_var.value)
```

# Summary

›› Broadcast variables are read-only shared variables with effective sharing mechanism

단, memory에 맞는 양의 데이터를 활용가능하다.

›› Useful to share dictionaries, models

# Accumulator variables

Useful for the control flow, monitoring, profiling & debugging

# Accumulator variable

› Accumulator variable is a read-write variable that is shared among tasks

›› Writes are restricted to increments! synchronization 문제를 피하기 위해

- ››i. e.: var += delta
- ››addition may be replaced by any associate, commutative operation

›› Reads are allowed only by the driver program! task에서는 읽을 수 없다.

# Guarantees on the updates

›› In actions updates are applied exactly once

action에서는 accumulator에 한 번만 적용된다.

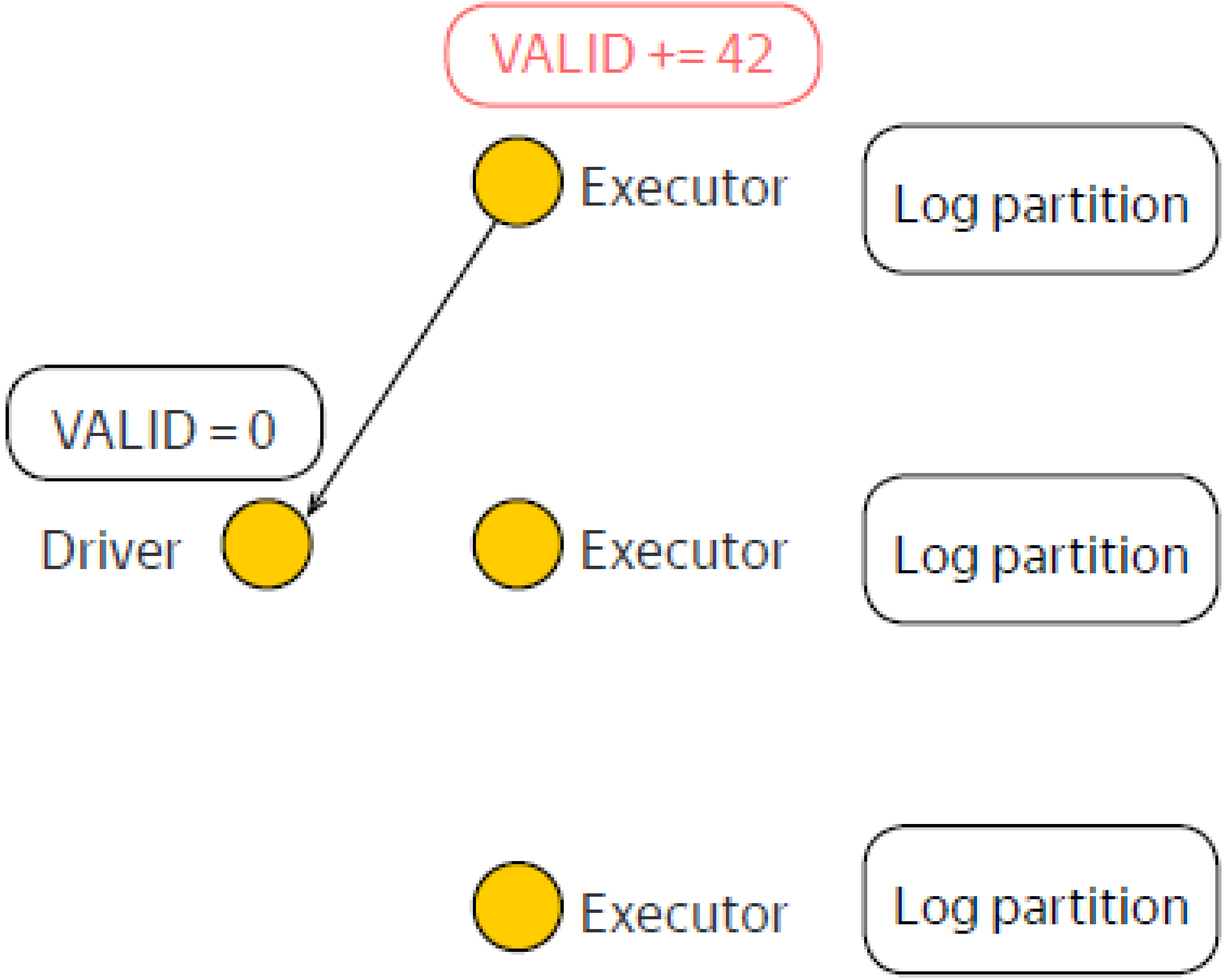
›› In transformations there are no guarantees as the transformation code may be re-executed

transformation은 재실행될 수 있기 때문에 보장할 수 없다.

# Example (similar to the previous video)

- Input:
  1TB partitioned log

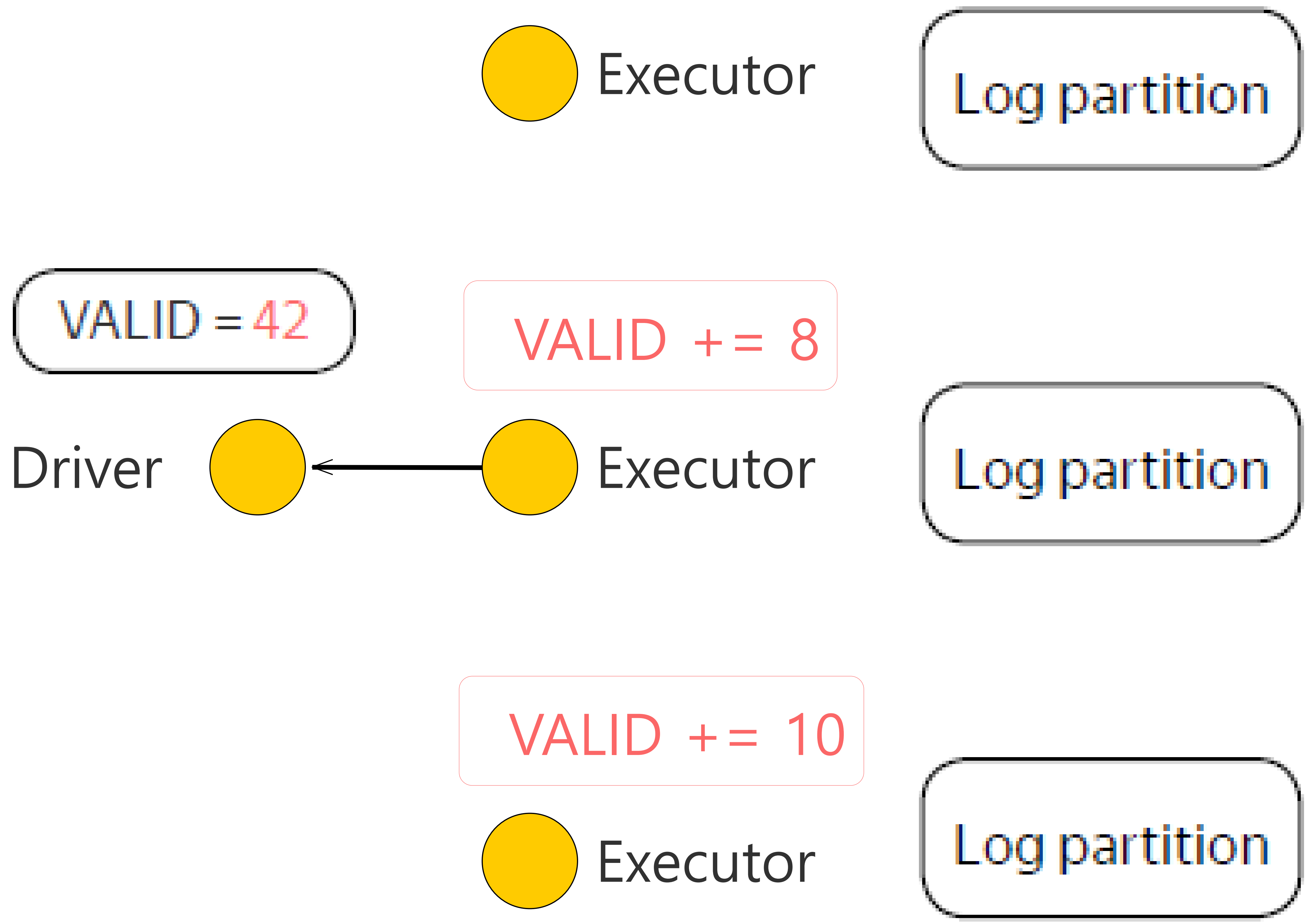
- Task:
  resolve IP addresses
  AND
  collect metrics:
  # of valid records



# Example (similar to the previous video)

- Input:
  1TB partitioned log

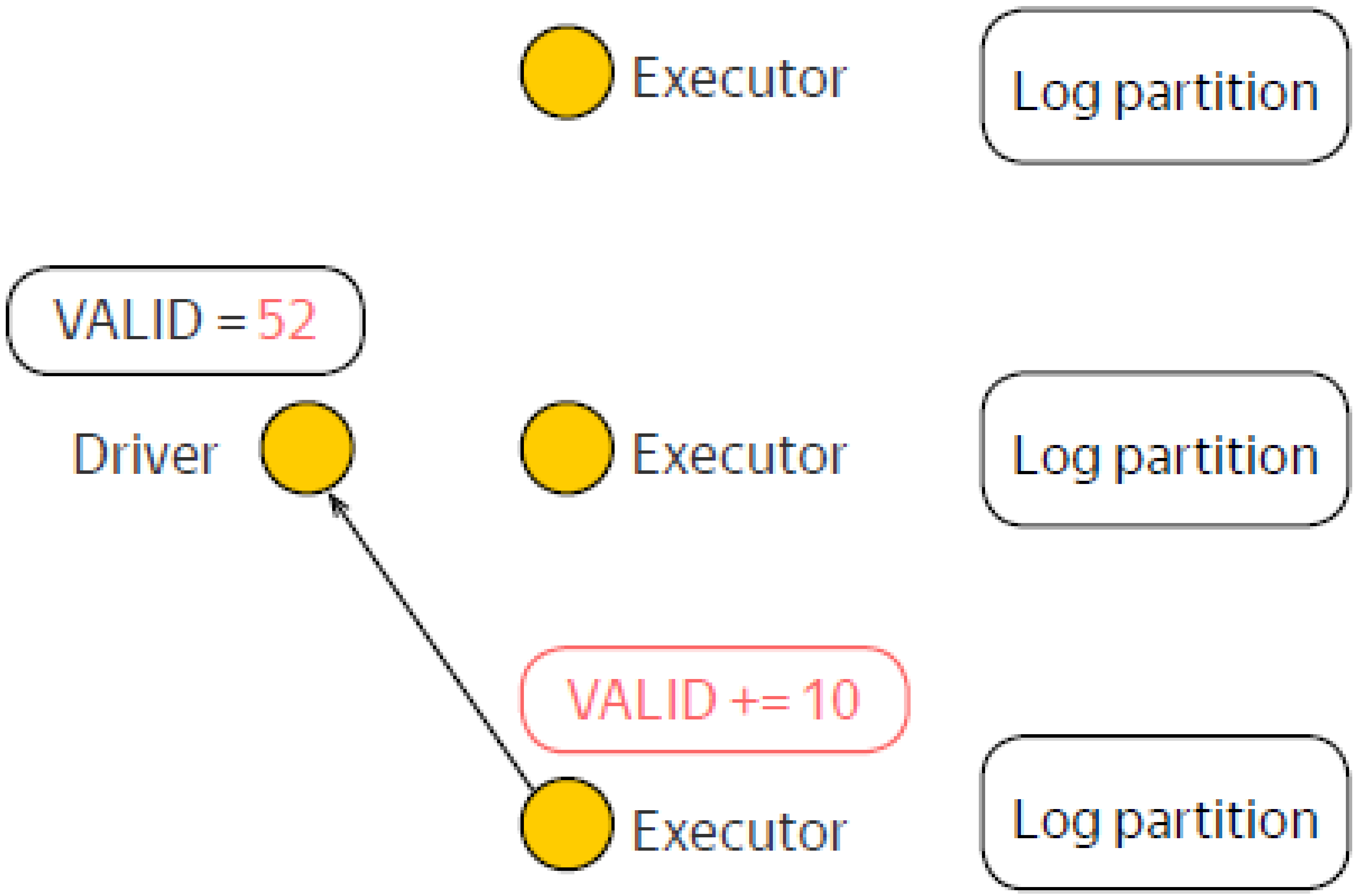
- Task:
  resolve IP addresses
  AND
  collect metrics:
  # of valid records



# Example (similar to the previous video)

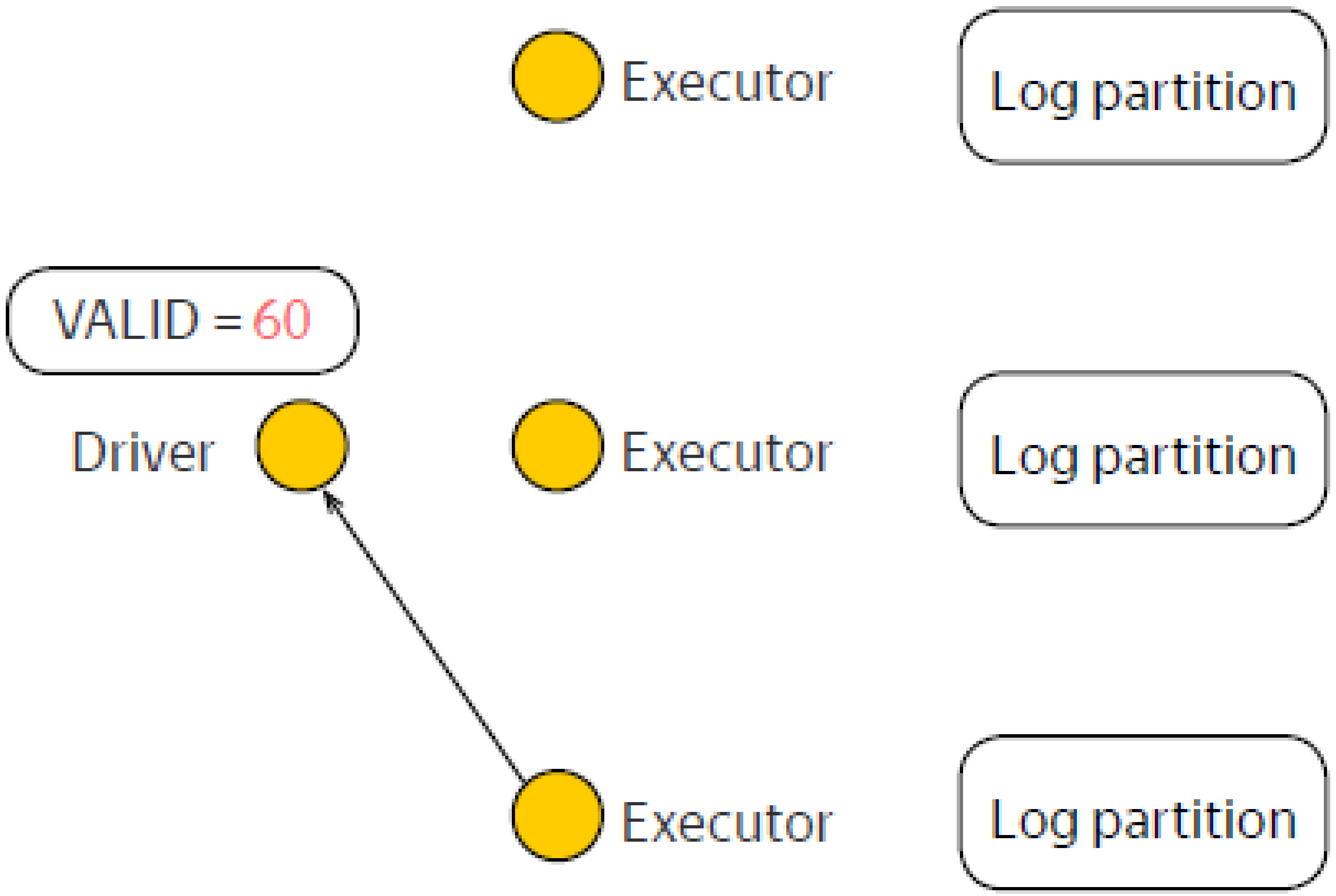
- Input:
  1TB partitioned log

- Task:
  resolve IP addresses
  AND
  collect metrics:
  # of valid records



# Example (similar to the previous video)

- Input:
  1TB partitioned log
- Task:
  resolve IP addresses
  AND
  collect metrics:
  # of valid records



# Use cases

- Performance counters
  - # of processed records, total elapsed time, total error and so on and so forth
- Simple control flow
  - conditionals: stop on reaching a threshold for corrupted records
  - loops: decide whether to run the next iteration of an algorithm or not
- Monitoring
  - export values to the monitoring system
- Profiling & debugging

# Summary

›› Accumulators are read-write shared variables with restricted updates

  - ››increments only
  - ››can use custom associative, commutative operation for the updates
  - ››can read the total value only in the driver

›› Useful for the control flow, monitoring, profiling & debugging

# Getting started with Spark & Python

# Installing Spark locally

- Navigate to http://spark.apache.org/docs/latest/#downloading and follow the instructions

- At the time of making this video
  - download .tar.gz
  - extract it
  - run ./bin/pyspark from the extracted directory

- If you have IPython installed, you can run PYSPARK_DRIVER_PYTHON=ipython pyspark

## ./bin/pyspark

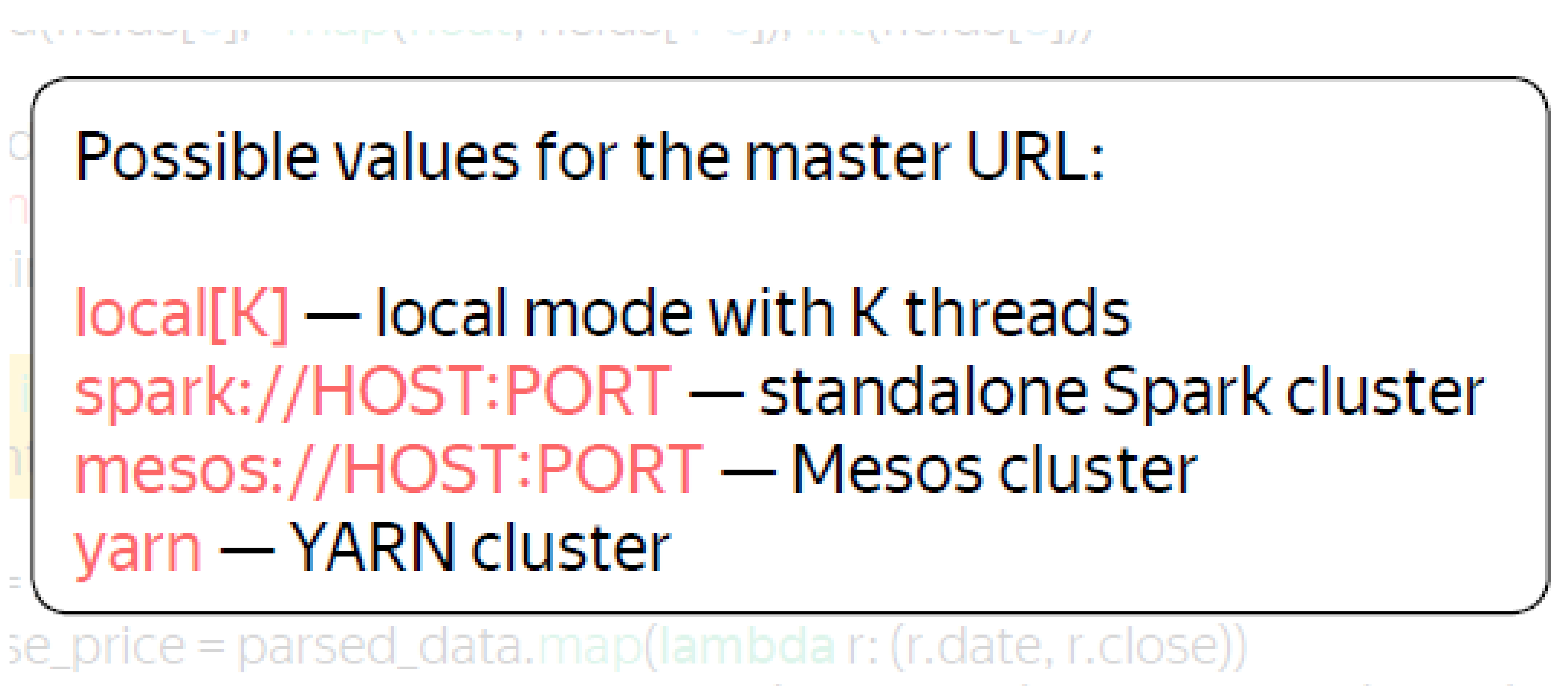
```
Python 2.7.10 (default, Feb  6 2017, 23:53:20)
[GCC 4.2.1 Compatible Apple LLVM 8.0.0 (clang-800.0.34)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
Using Spark's default log4j profile: org/apache/spark/log4j-defaults.properties
Setting default log level to "WARN".
To adjust logging level use sc.setLogLevel(newLevel). For SparkR, use setLogLevel(newLevel).
Welcome to
      ____              __
     / __/__  ___ _____/ /__
    _\ \/ _ \/ _ `/ __/  '_/
   /__ / .__/\_,_/_/ /_/\_\   version 2.2.0
      /_/

Using Python version 2.7.10 (default, Feb  6 2017 23:53:20)
SparkSession available as 'spark'.
>>> sc
<SparkContext master=local[*] appName=PySparkShell>
>>>
```

spark shell에서는 sc가 자동으로 생성된다.

## SparkContext 생성 방법

```
from pyspark import SparkConf, SparkContext
sc = SparkContext(conf=SparkConf().setAppName("MyApp").setMaster("local"))
```

## Master 설정 방법 ( Cluster Mode )

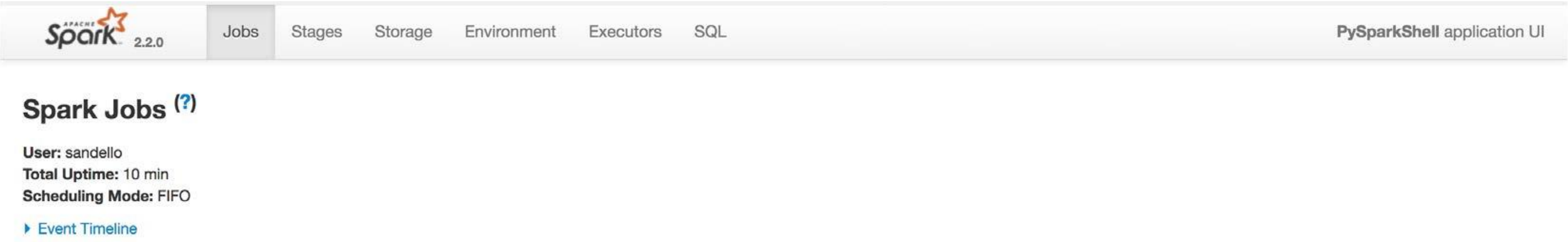


Possible values for the master URL:

local[K] — local mode with K threads
spark://HOST:PORT — standalone Spark cluster
mesos://HOST:PORT — Mesos cluster
yarn — YARN cluster

```
se_price = parsed_data.map(lambda r: (r.date, r.close))
```

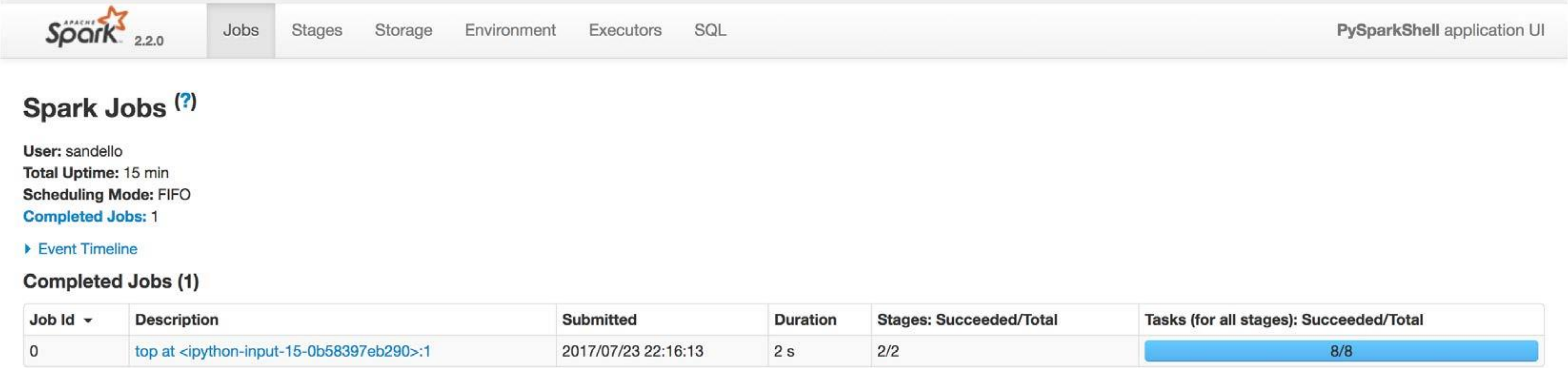
# Spark UI



Spark Jobs (?)

**User:** sandello
**Total Uptime:** 10 min
**Scheduling Mode:** FIFO

▸ Event Timeline



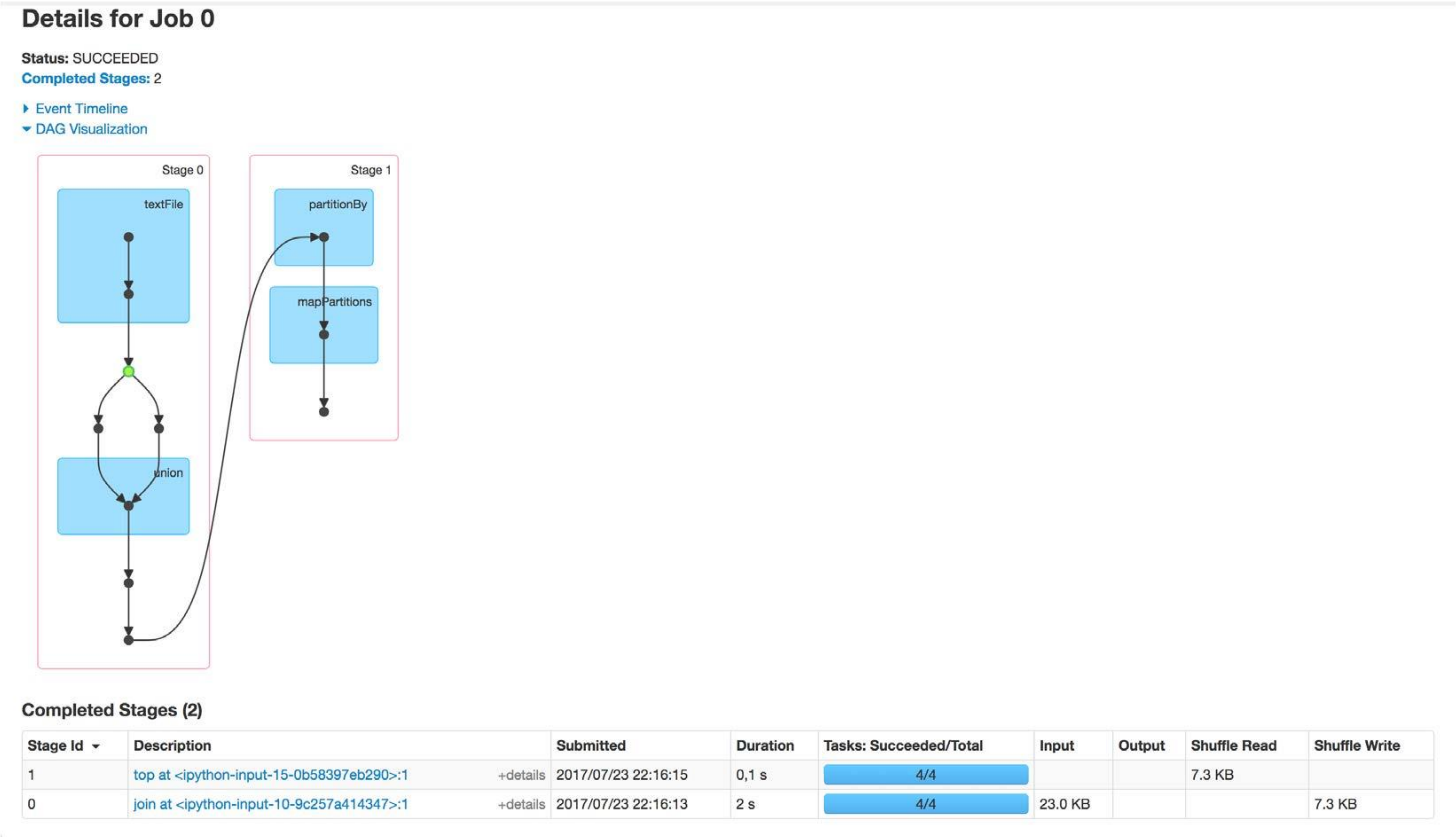
## Spark Jobs (?)

**User:** sandello
**Total Uptime:** 15 min
**Scheduling Mode:** FIFO
**Completed Jobs:** 1

▸ Event Timeline

### Completed Jobs (1)

| Job Id ▾ | Description | Submitted | Duration | Stages: Succeeded/Total | Tasks (for all stages): Succeeded/Total |
|---|---|---|---|---|---|
| 0 | top at <ipython-input-15-0b58397eb290>:1 | 2017/07/23 22:16:13 | 2 s | 2/2 | 8/8 |



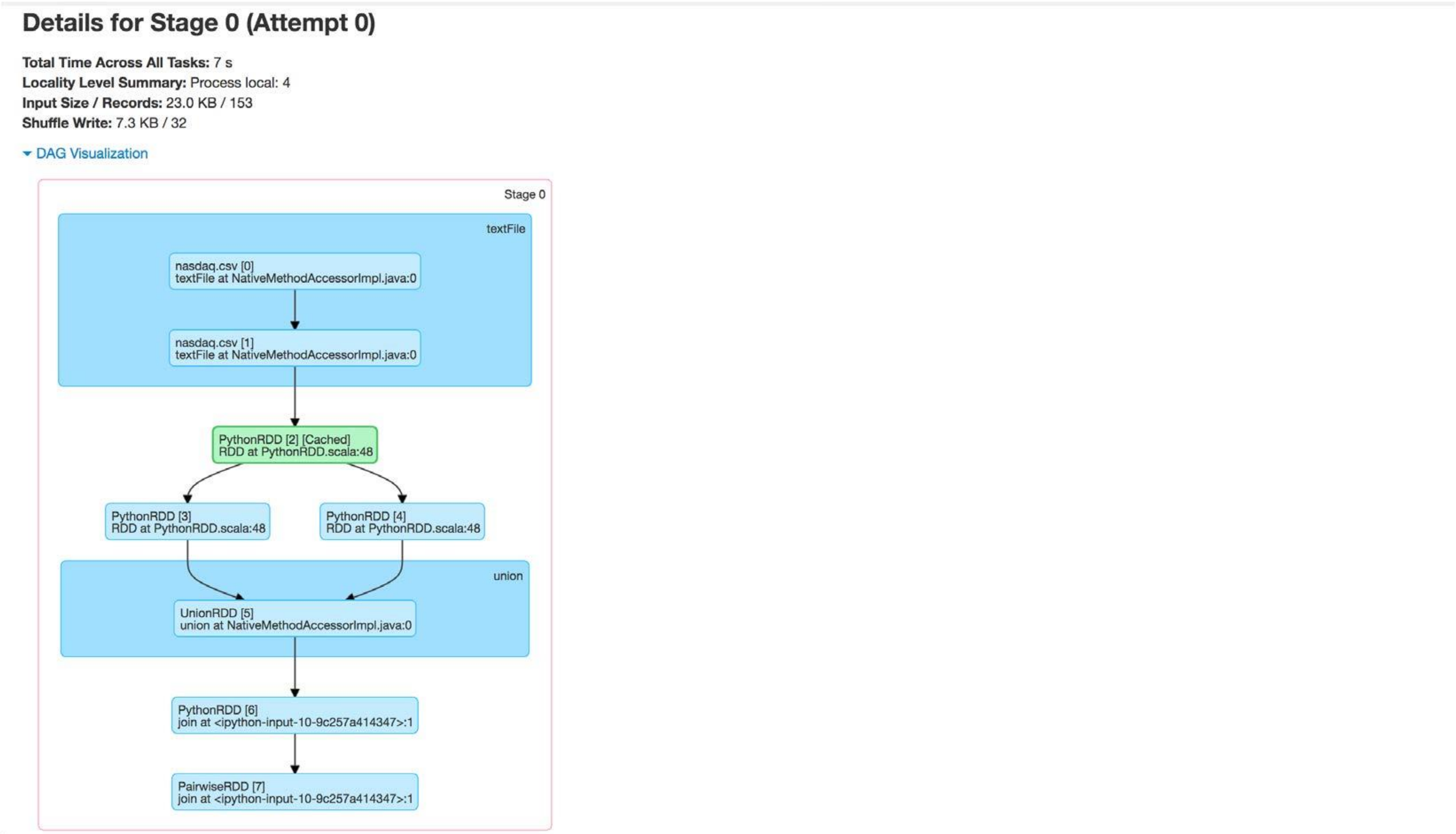
## Details for Job 0

**Status:** SUCCEEDED
**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 | top at <ipython-input-15-0b58397eb290>:1 +details | 2017/07/23 22:16:15 | 0,1 s | 4/4 | | | 7.3 KB | |
| 0 | join at <ipython-input-10-9c257a414347>:1 +details | 2017/07/23 22:16:13 | 2 s | 4/4 | 23.0 KB | | | 7.3 KB |

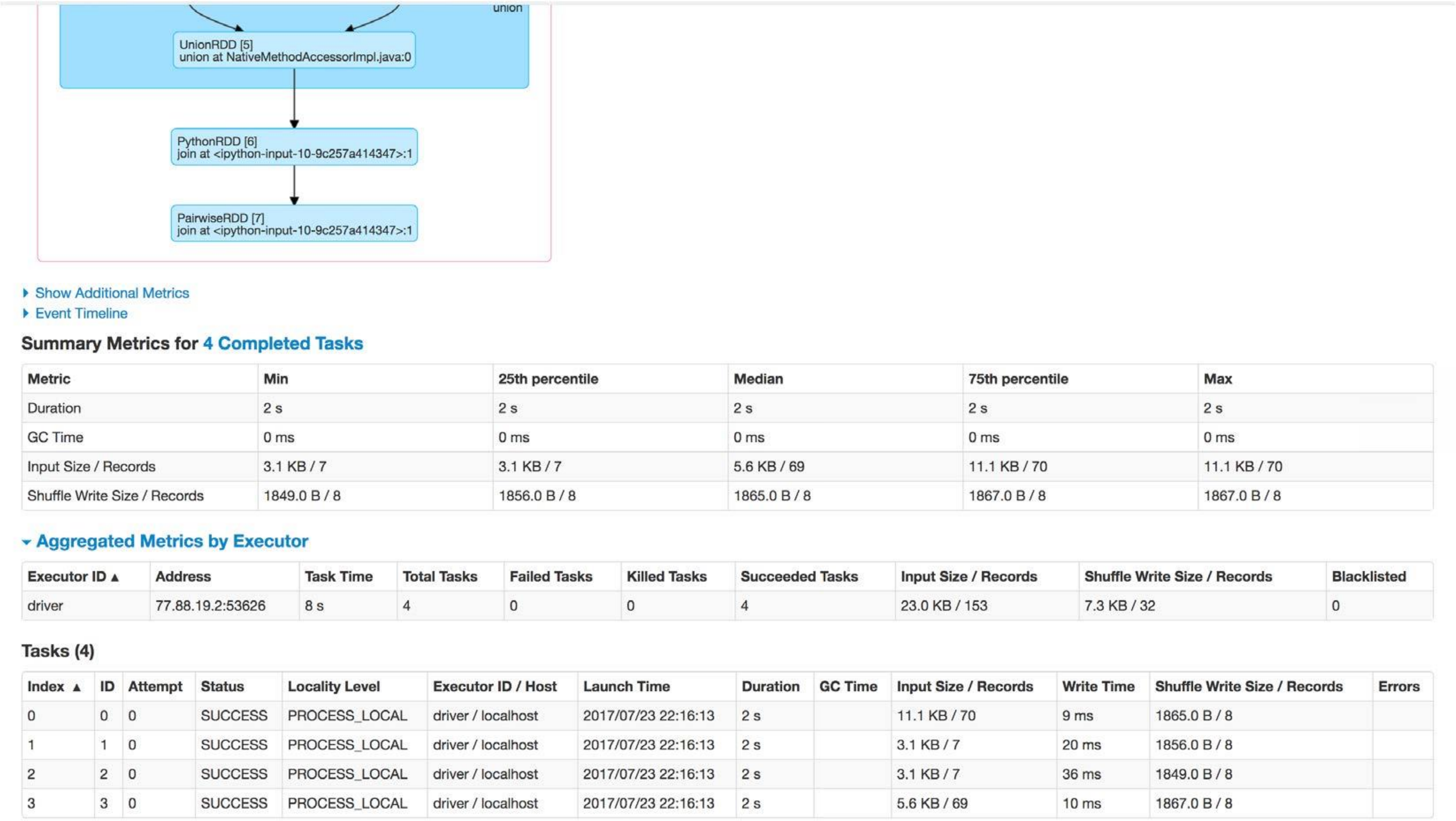


# Details for Stage 0 (Attempt 0)

**Total Time Across All Tasks:** 7 s
**Locality Level Summary:** Process local: 4
**Input Size / Records:** 23.0 KB / 153
**Shuffle Write:** 7.3 KB / 32

▾ DAG Visualization

Stage 0
textFile
nasdaq.csv [0]
textFile at NativeMethodAccessorImpl.java:0
nasdaq.csv [1]
textFile at NativeMethodAccessorImpl.java:0
PythonRDD [2] [Cached]
RDD at PythonRDD.scala:48
PythonRDD [3]
RDD at PythonRDD.scala:48
PythonRDD [4]
RDD at PythonRDD.scala:48
union
UnionRDD [5]
union at NativeMethodAccessorImpl.java:0
PythonRDD [6]
join at <ipython-input-10-9c257a414347>:1
PairwiseRDD [7]
join at <ipython-input-10-9c257a414347>:1


union
UnionRDD [5]
union at NativeMethodAccessorImpl.java:0
PythonRDD [6]
join at <ipython-input-10-9c257a414347>:1
PairwiseRDD [7]
join at <ipython-input-10-9c257a414347>:1

▸ Show Additional Metrics
▸ Event Timeline

**Summary Metrics for 4 Completed Tasks**

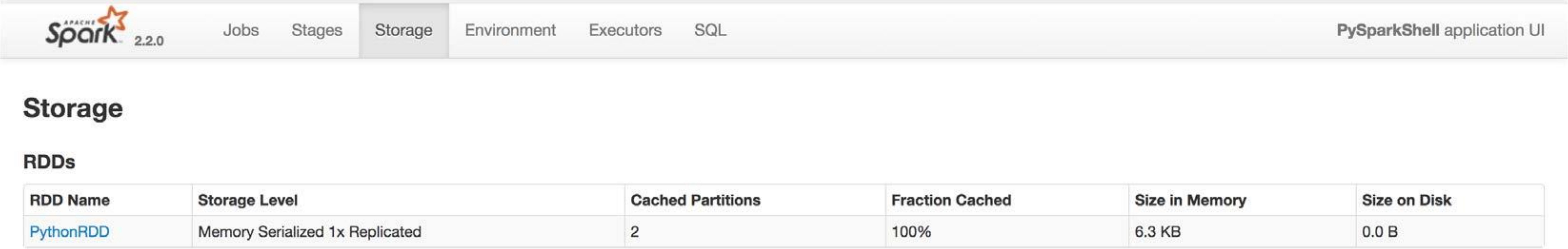
| Metric | Min | 25th percentile | Median | 75th percentile | Max |
|---|---|---|---|---|---|
| Duration | 2 s | 2 s | 2 s | 2 s | 2 s |
| GC Time | 0 ms | 0 ms | 0 ms | 0 ms | 0 ms |
| Input Size / Records | 3.1 KB / 7 | 3.1 KB / 7 | 5.6 KB / 69 | 11.1 KB / 70 | 11.1 KB / 70 |
| Shuffle Write Size / Records | 1849.0 B / 8 | 1856.0 B / 8 | 1865.0 B / 8 | 1867.0 B / 8 | 1867.0 B / 8 |

▾ **Aggregated Metrics by Executor**

| Executor ID ▲ | Address | Task Time | Total Tasks | Failed Tasks | Killed Tasks | Succeeded Tasks | Input Size / Records | Shuffle Write Size / Records | Blacklisted |
|---|---|---|---|---|---|---|---|---|---|
| driver | 77.88.19.2:53626 | 8 s | 4 | 0 | 0 | 4 | 23.0 KB / 153 | 7.3 KB / 32 | 0 |

**Tasks (4)**

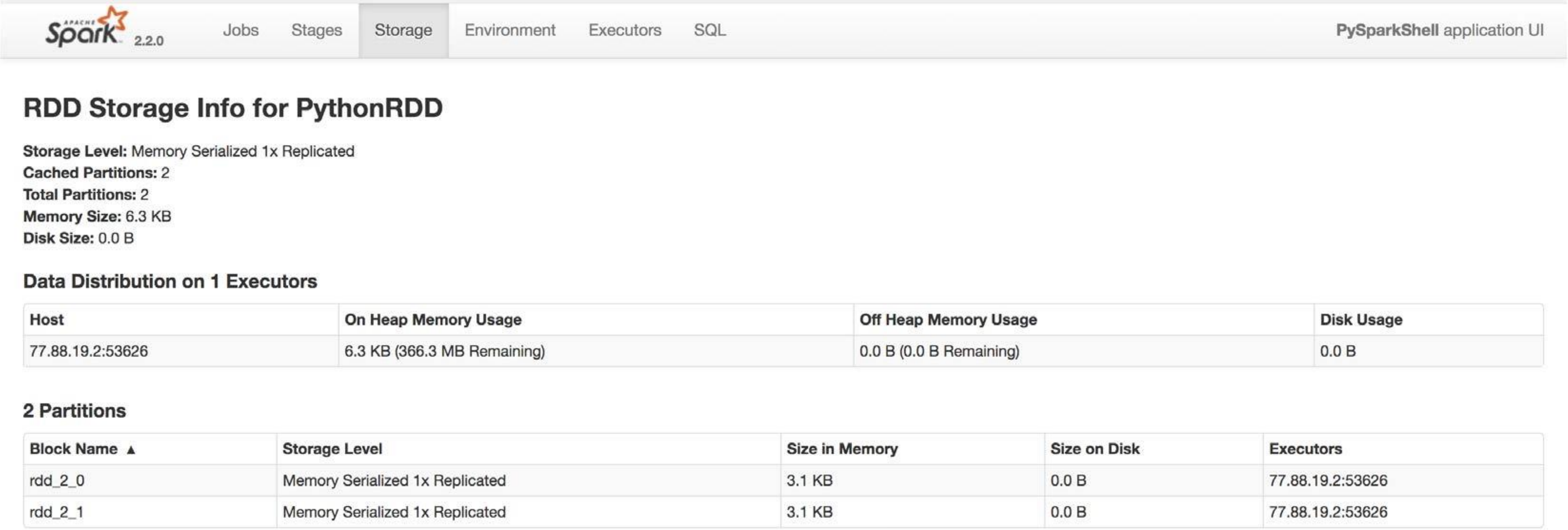
| Index ▲ | ID | Attempt | Status | Locality Level | Executor ID / Host | Launch Time | Duration | GC Time | Input Size / Records | Write Time | Shuffle Write Size / Records | Errors |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | SUCCESS | PROCESS_LOCAL | driver / localhost | 2017/07/23 22:16:13 | 2 s |  | 11.1 KB / 70 | 9 ms | 1865.0 B / 8 |  |
| 1 | 1 | 0 | SUCCESS | PROCESS_LOCAL | driver / localhost | 2017/07/23 22:16:13 | 2 s |  | 3.1 KB / 7 | 20 ms | 1856.0 B / 8 |  |
| 2 | 2 | 0 | SUCCESS | PROCESS_LOCAL | driver / localhost | 2017/07/23 22:16:13 | 2 s |  | 3.1 KB / 7 | 36 ms | 1849.0 B / 8 |  |
| 3 | 3 | 0 | SUCCESS | PROCESS_LOCAL | driver / localhost | 2017/07/23 22:16:13 | 2 s |  | 5.6 KB / 69 | 10 ms | 1867.0 B / 8 |  |



# Storage

## RDDs

| RDD Name | Storage Level | Cached Partitions | Fraction Cached | Size in Memory | Size on Disk |
|---|---|---|---|---|---|
| PythonRDD | Memory Serialized 1x Replicated | 2 | 100% | 6.3 KB | 0.0 B |



## RDD Storage Info for PythonRDD

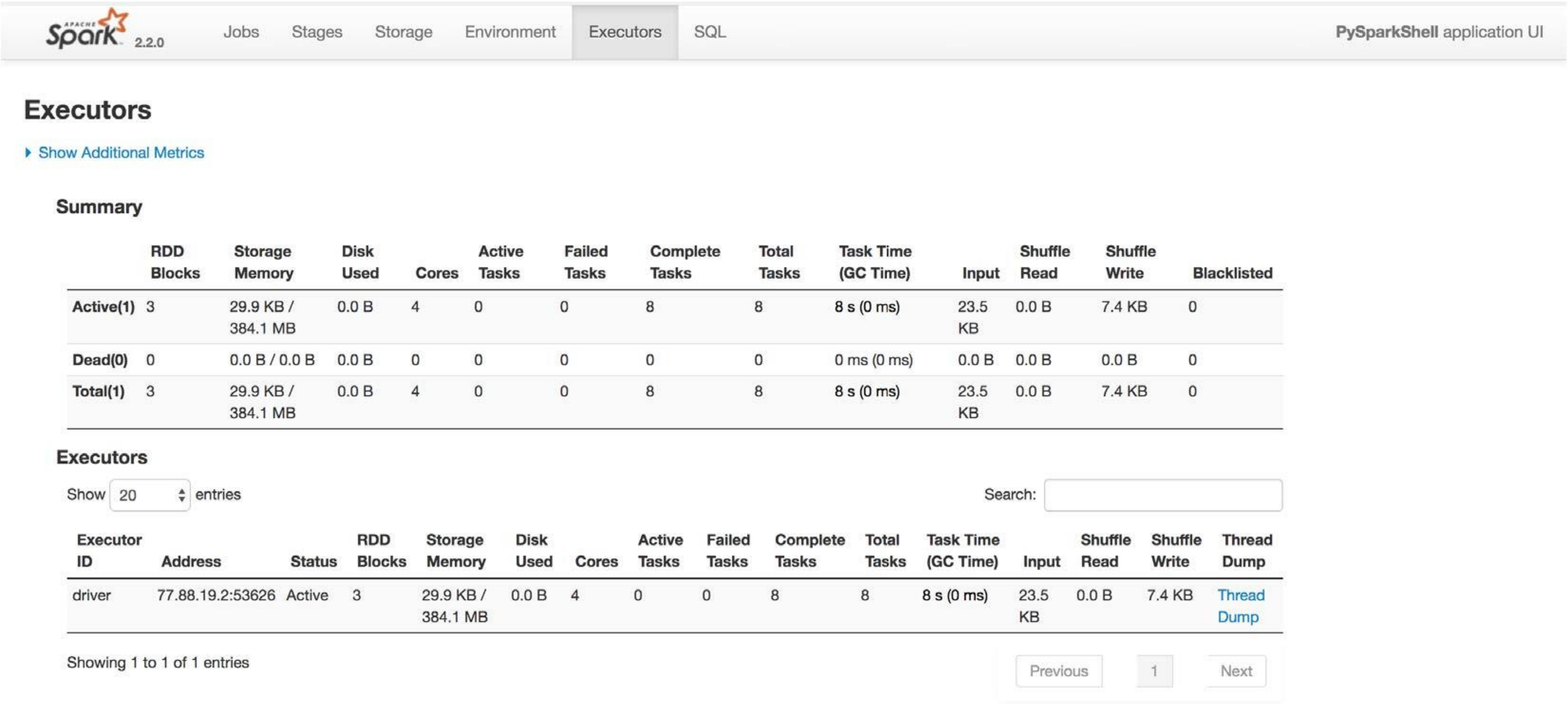
**Storage Level:** Memory Serialized 1x Replicated
**Cached Partitions:** 2
**Total Partitions:** 2
**Memory Size:** 6.3 KB
**Disk Size:** 0.0 B

### Data Distribution on 1 Executors

| Host | On Heap Memory Usage | Off Heap Memory Usage | Disk Usage |
|---|---|---|---|
| 77.88.19.2:53626 | 6.3 KB (366.3 MB Remaining) | 0.0 B (0.0 B Remaining) | 0.0 B |

### 2 Partitions

| Block Name ▲ | Storage Level | Size in Memory | Size on Disk | Executors |
|---|---|---|---|---|
| rdd_2_0 | Memory Serialized 1x Replicated | 3.1 KB | 0.0 B | 77.88.19.2:53626 |
| rdd_2_1 | Memory Serialized 1x Replicated | 3.1 KB | 0.0 B | 77.88.19.2:53626 |



# Executors

▸ Show Additional Metrics

## Summary

| | RDD Blocks | Storage Memory | Disk Used | Cores | Active Tasks | Failed Tasks | Complete Tasks | Total Tasks | Task Time (GC Time) | Input | Shuffle Read | Shuffle Write | Blacklisted |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **Active(1)** | 3 | 29.9 KB / 384.1 MB | 0.0 B | 4 | 0 | 0 | 8 | 8 | 8 s (0 ms) | 23.5 KB | 0.0 B | 7.4 KB | 0 |
| **Dead(0)** | 0 | 0.0 B / 0.0 B | 0.0 B | 0 | 0 | 0 | 0 | 0 | 0 ms (0 ms) | 0.0 B | 0.0 B | 0.0 B | 0 |
| **Total(1)** | 3 | 29.9 KB / 384.1 MB | 0.0 B | 4 | 0 | 0 | 8 | 8 | 8 s (0 ms) | 23.5 KB | 0.0 B | 7.4 KB | 0 |

## Executors

Show 20 entries

Search:

| Executor ID | Address | Status | RDD Blocks | Storage Memory | Disk Used | Cores | Active Tasks | Failed Tasks | Complete Tasks | Total Tasks | Task Time (GC Time) | Input | Shuffle Read | Shuffle Write | Thread Dump |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| driver | 77.88.19.2:53626 | Active | 3 | 29.9 KB / 384.1 MB | 0.0 B | 4 | 0 | 0 | 8 | 8 | 8 s (0 ms) | 23.5 KB | 0.0 B | 7.4 KB | Thread Dump |

Showing 1 to 1 of 1 entries

# Summary

›› You have learned how to open and use Spark UI

›› In the next course of the specialization you will learn how to use the interface to optimize your application performance

**Big**DATAteam