Getting Started with Apache Spark

Need for Apache Spark

- Two to three decades back, most organizations were only working with structured data sources.

- We used to extract data from them using extract, transform, and load, or ETL tools. Clean and transform the data, and then load it into a data warehouse.

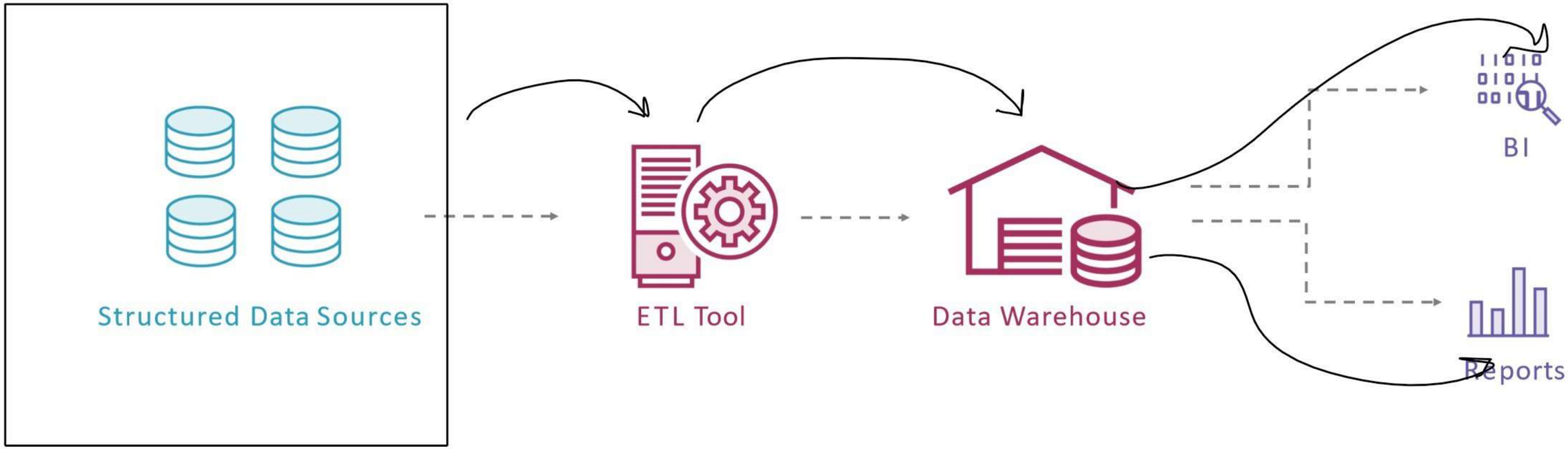
- So our data warehouse contains highly curated and structured data.

- From here, it was used for self- service BI and for reporting purposes.

- While this system worked well earlier, but the problem is that now structured data sources only account for 20% of organizations' data

structured / unstructured?

SAP HANA


Structured Data Sources
ETL Tool
Data Warehouse
BI
Reports

There is data available in a variety of file formats, like CSV, JSON, and log files. There is data from NoSQL databases.

There are audio and video files, and then there is streaming data as well. And this accounts for 80% of organizations' data.

So there has been an exponential growth, not just in types of data sources, but in data volumes as well.

To handle this, you also need to scale your infrastructure. While you may add more resources like CPU and RAM to your existing servers, it cannot be scaled up beyond a limit.

And finally, data pipelines have to quickly adapt to support other use cases, like machine learning and data science

5 V's of Big Data

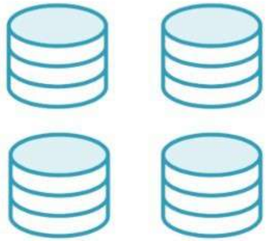
Volume → Scale of Data

Variety → Different forms of data

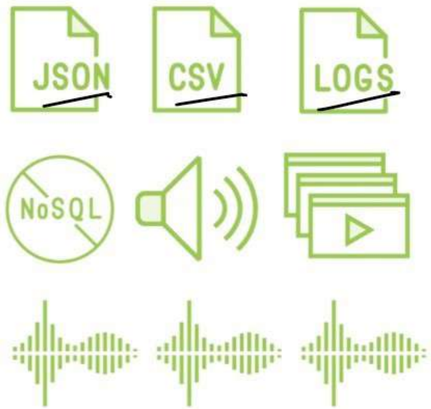
Velocity → Speed → Stock Market Load
→ Streaming Applications

Value → generating a value from data

Veracity → eliminate uncertainty in data
→ Validation scenario



Structured Data Sources



Unstructured &Streaming Data Sources

Exponential Data Growth

ETL Tool

Data Warehouse

? No SQL → mongo DB
JSON -
Parquet

Infrastructure Scalability

BI

Reports

Machine Learning

Data Science

Evolving Business Needs

History

Apache Hadoop ?

Earlier

ETL Tools / RDBMS

2004

MapReduce / Distributed File System

Paper published by Google for distributed data processing

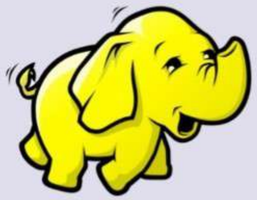
History

Earlier

ETL Tools / RDBMS

2006

Hadoop (MapReduce & HDFS)

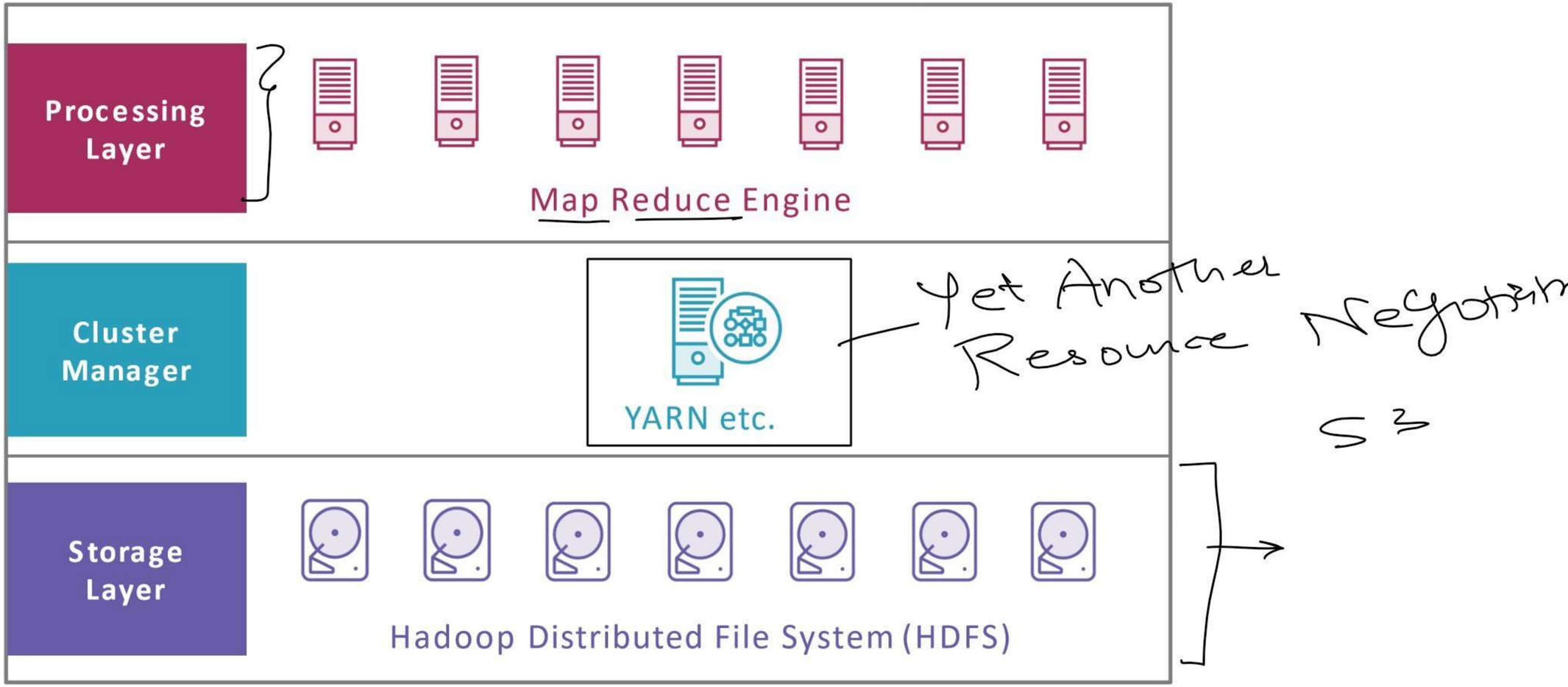


Framework for distributed storage & distributed data processing

2004

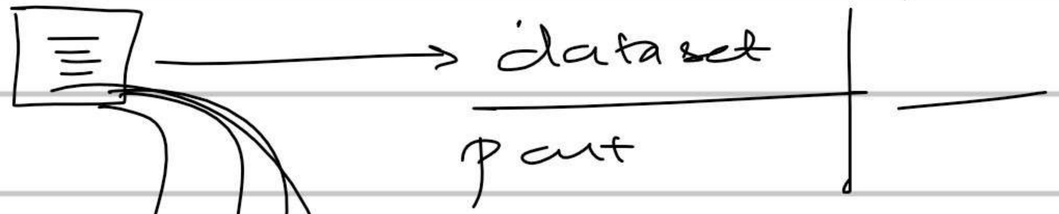
MapReduce / Distributed File System

Hadoop Components

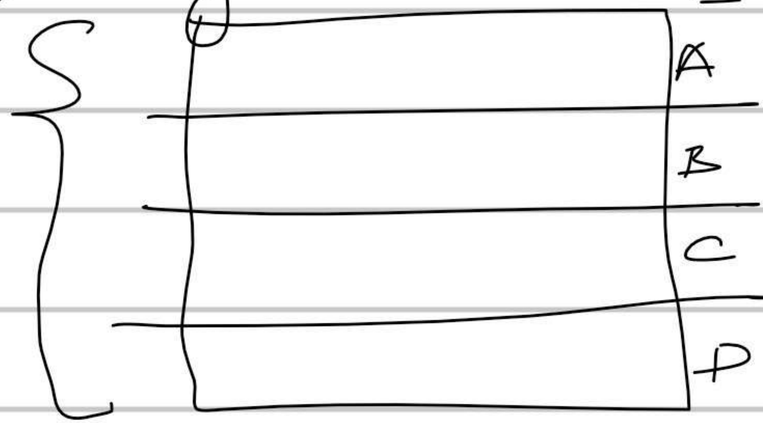


· Java / ——

Survey · Dataset



Sectors



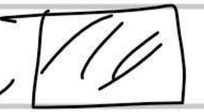
population

A – 19K

B → 20K

C → 21K

D → 25K



A → 5 mins

B → 5 mins

C · 5 mins

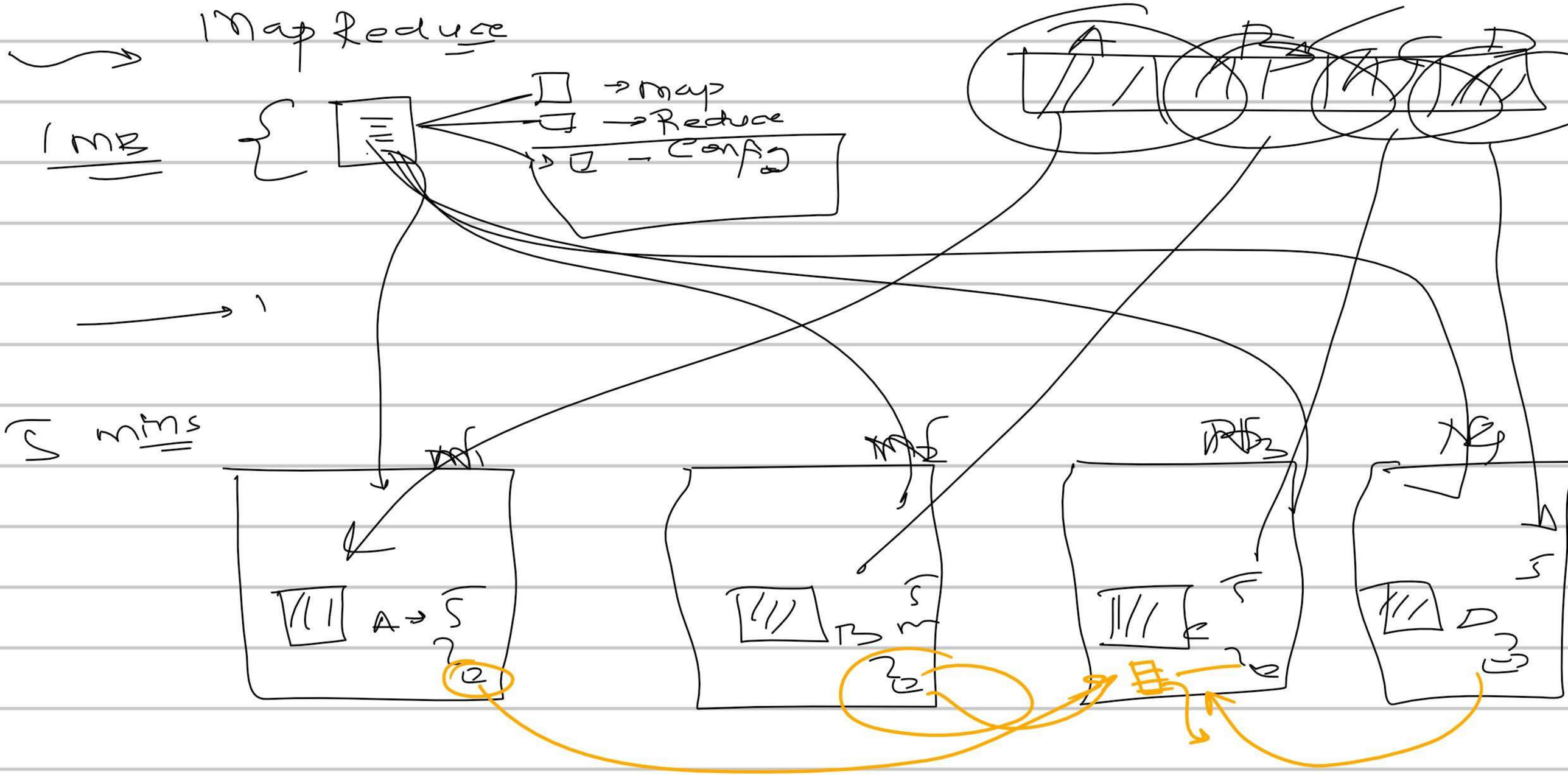
D · 5 mins

Count

→ 1 min

21 mins

Map Reduce



HDFS

An HDFS, or Hadoop Distributed File System, is a distributed storage. This layer has multiple disks, and the files are stored here in a distributed manner. What does that mean? This means if you have a large file, it is split into multiple blocks, and the blocks are placed onto different disks. You can keep adding more hardware to virtually store unlimited data. So HDFS provides a scalable, fault tolerant, and low- cost solution to store the files.

MR

MapReduce engine is used for data processing. It can process structured, as well as unstructured data stored in HDFS. And this processing is done in parallel using multiple machines in the Hadoop cluster.

And finally, there is a cluster manager, like YARN, in short fo Yet Another Resource Negotiator. So cluster manager is responsible for managing and allocating the resources required for processing the data.

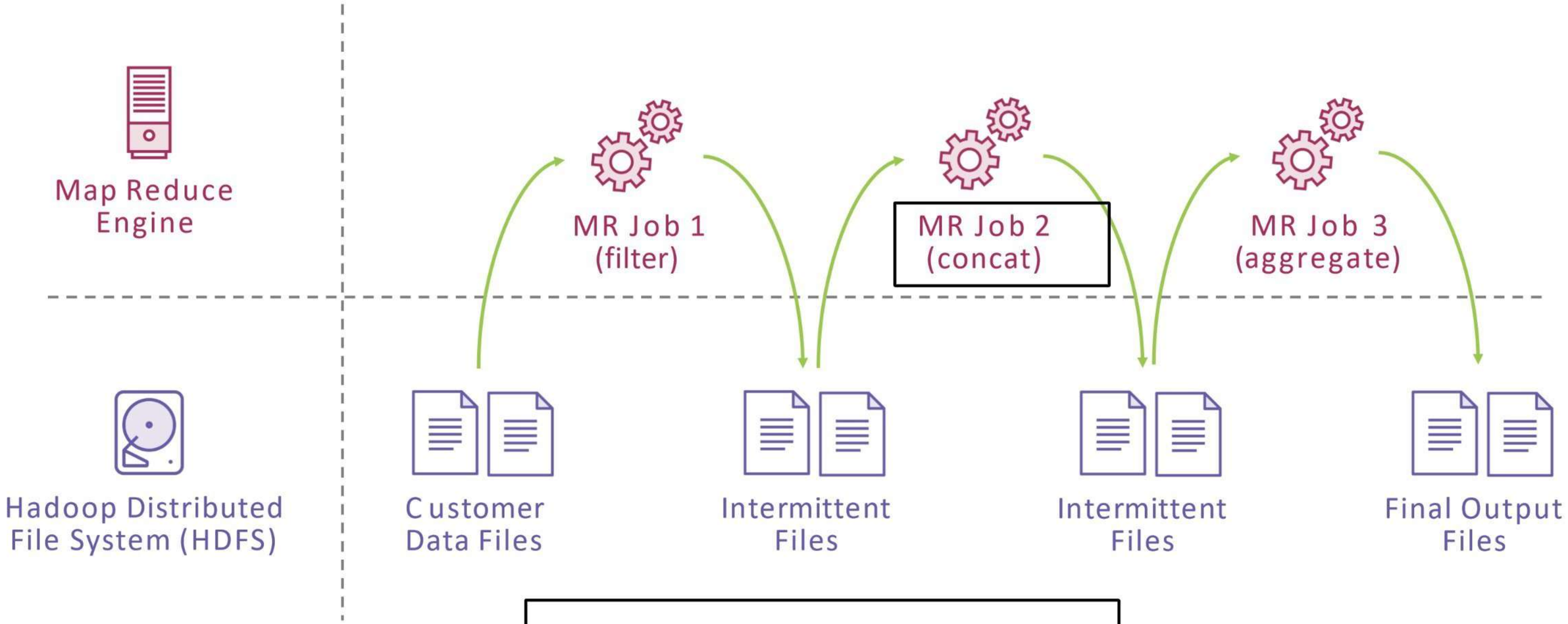
Let's say you have files stored in HDFS related to customers, and you want to filter customers who are new, combine their first and last names, and do some aggregates.

Now to process the data, you will write code that defines your MapReduce jobs.

So let's say the first job is to filter new customers. So MapReduce job will read the files and filter the data, and then write the intermittent output to HDFS.

Then for the next operation to combine first and last names, the second MapReduce job will read these intermittent files, apply the operation, and again write the partial output to disk.

How Hadoop Works?

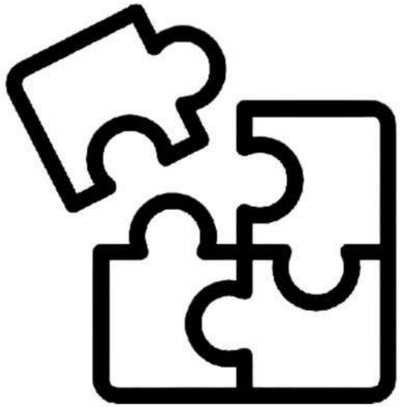


Disk-heavy Architecture

And for final operation, it will do the same and produce final output files. So if you carefully notice, each MapReduce job is reading and writing to disk.

This is what is known as disk- heavy architecture of Hadoop. So while Hadoop can process huge volumes of data, there are certain challenges as well.

Challenges with Hadoop



Infrastructure & environment complexity

Need to write a lot of code

Disk-heavy architecture: Multiple IO operations slows down processing

No built-in support for streaming, ML etc.

- New tools were built to handle them adding to complexity

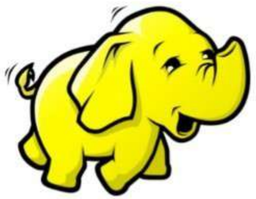
History

Earlier

ETL Tools / RDBMS

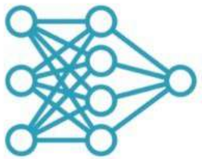
2006

Hadoop (MapReduce & HDFS)



2004

MapReduce / Distributed File System



2009

Apache Spark

In-memory engine for distributed data processing

Apache Spark

Extremely powerful analytics engine for large-scale distributed data processing, whether structured or unstructured

In-memory engine can run workloads up to 100x faster than Hadoop

Simplified code and multiple language support

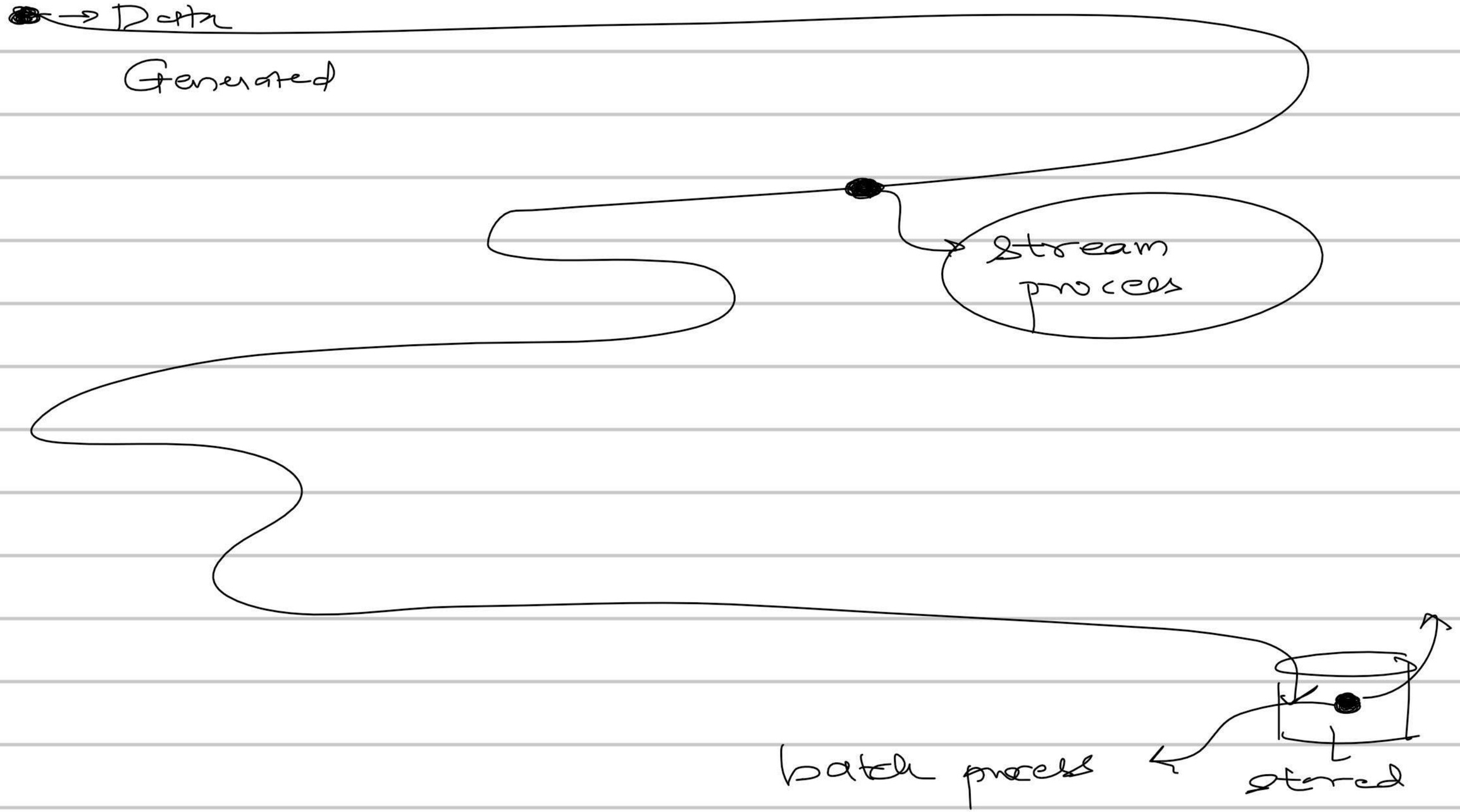
Scala

Dev → Scala
Java
Python
R

SQL

Unifies variety of use cases – batch processing, stream processing, machine learning & advanced analytics

APL



Apache Hadoop

→ open source

→ Cloudera

→ Hortonworks

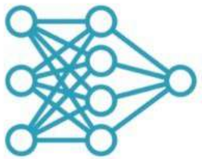
AWS, Azure, GCP

Apache Spark

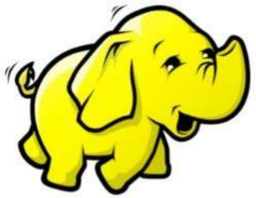
History

Earlier
ETL Tools / RDBMS

2004
MapReduce / Distributed File System



2006
Hadoop (MapReduce & HDFS)



2009
Apache Spark

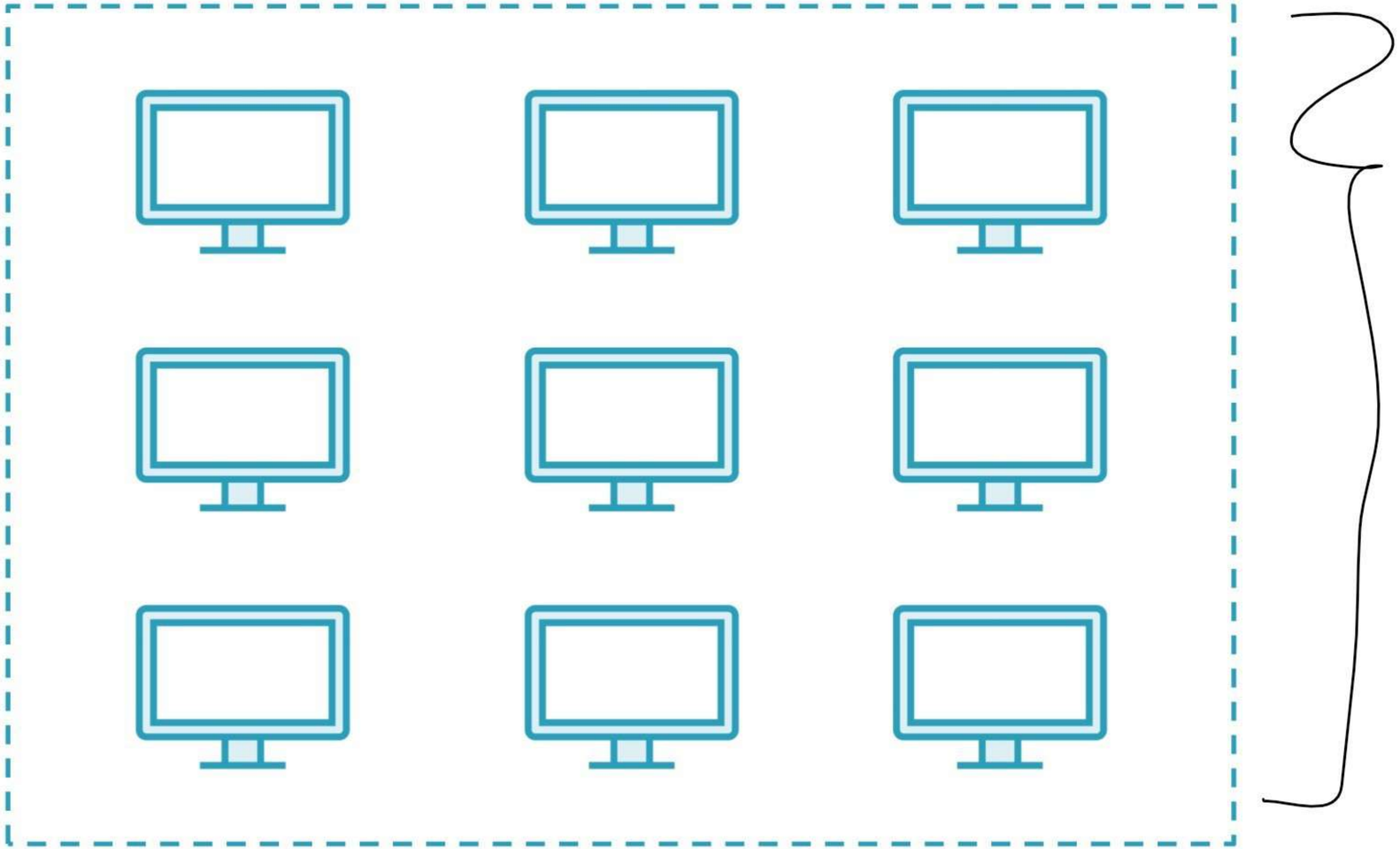
Now
Cloud-based Spark platforms

Databricks
Azure Synapse Analytics
Amazon EMR etc.

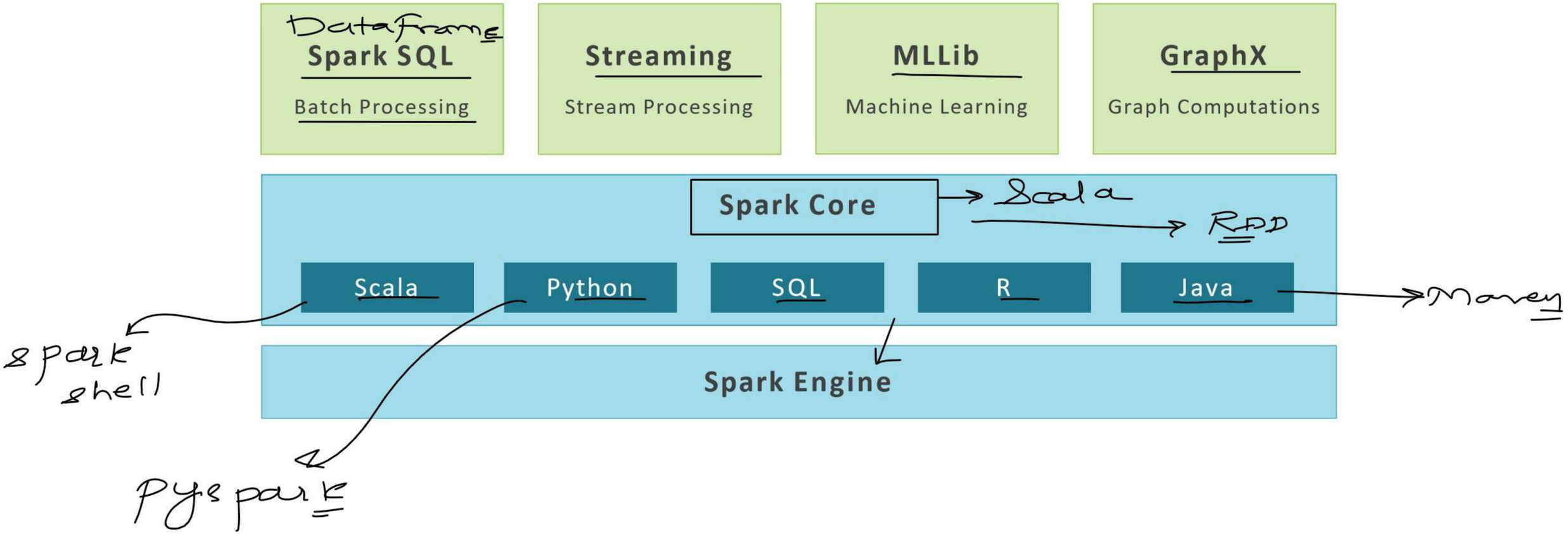
Understanding Spark Architecture & Ecosystem

Apache Spark is an extremely powerful, in-memory analytics engine built on cluster computing technology

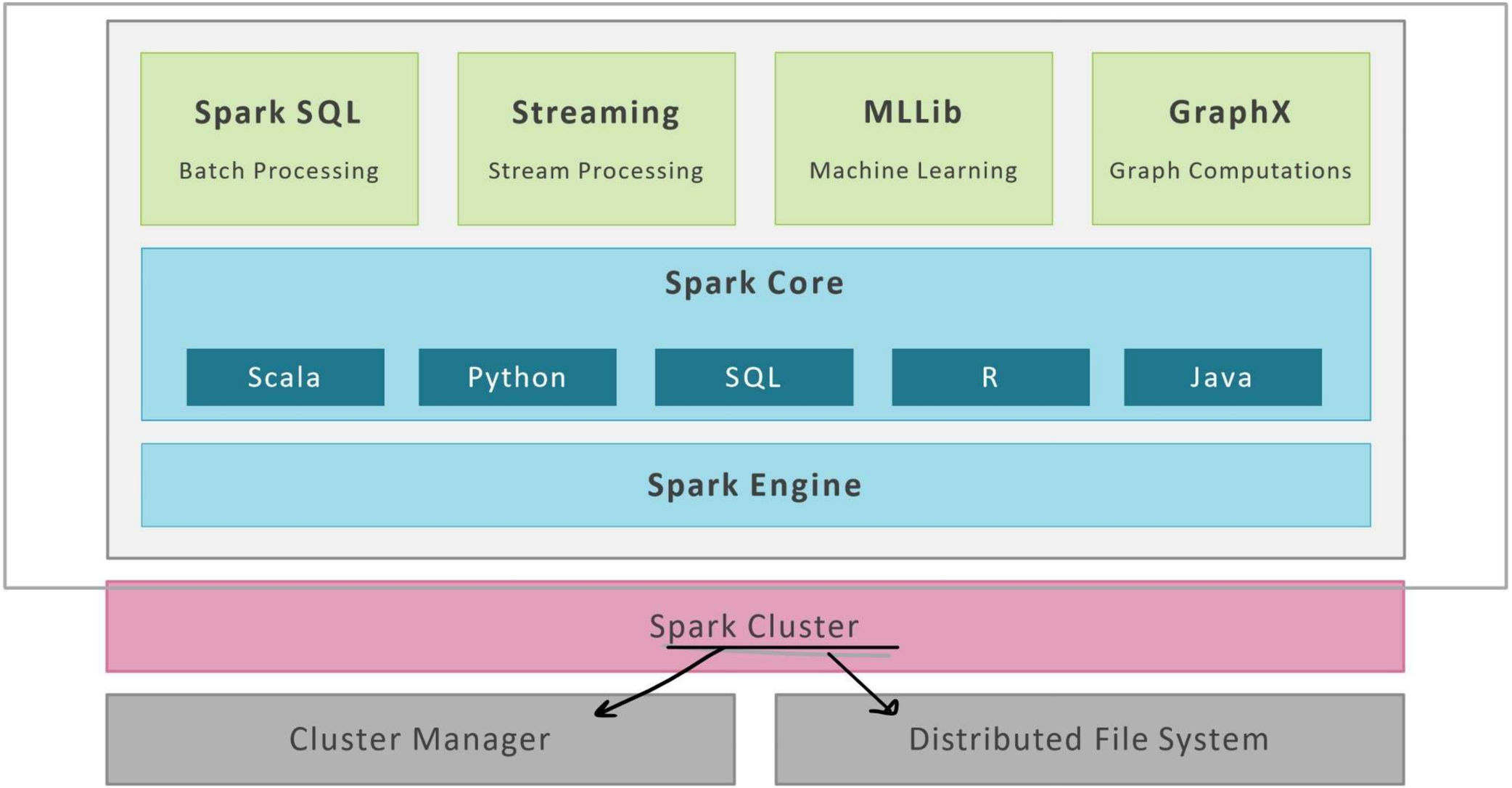
Spark Cluster



Spark Architecture



Spark Architecture



Spark Ecosystem

Cluster Managers

- YARN, Kubernetes, Mesos etc. , standalone

Distributed File System options

- HDFS, Azure Data Lake Store, Amazon S3 or Google Cloud Storage

Multiple language support

- Scala, Python, SQL, R & Java
- Open-source support for C#

Development options

- Console, IDEs (PyCharm, VS Code), Notebooks (Jupyter, Zeppelin) etc.

Available in Cloud Platforms

- Databricks, Cloudera, Azure Synapse Analytics, Azure HDInsight, Amazon EMR etc.

Open-source Connectors

Relational databases – SQL Server, Oracle

NoSQL – MongoDB, Cassandra, Azure Cosmos DB

Apache Hadoop, HBase, Hive

Cloud storage

MPP engines – AWS Redshift, Azure Dedicated SQL

Visualization tools – Power BI, Tableau

Streaming – Kafka, Azure Event Hubs, AWS Kinesis

...and much more

Check out more open-source projects…

https://spark.apache.org/third-party-projects.html

How Execution Happens in Spark?

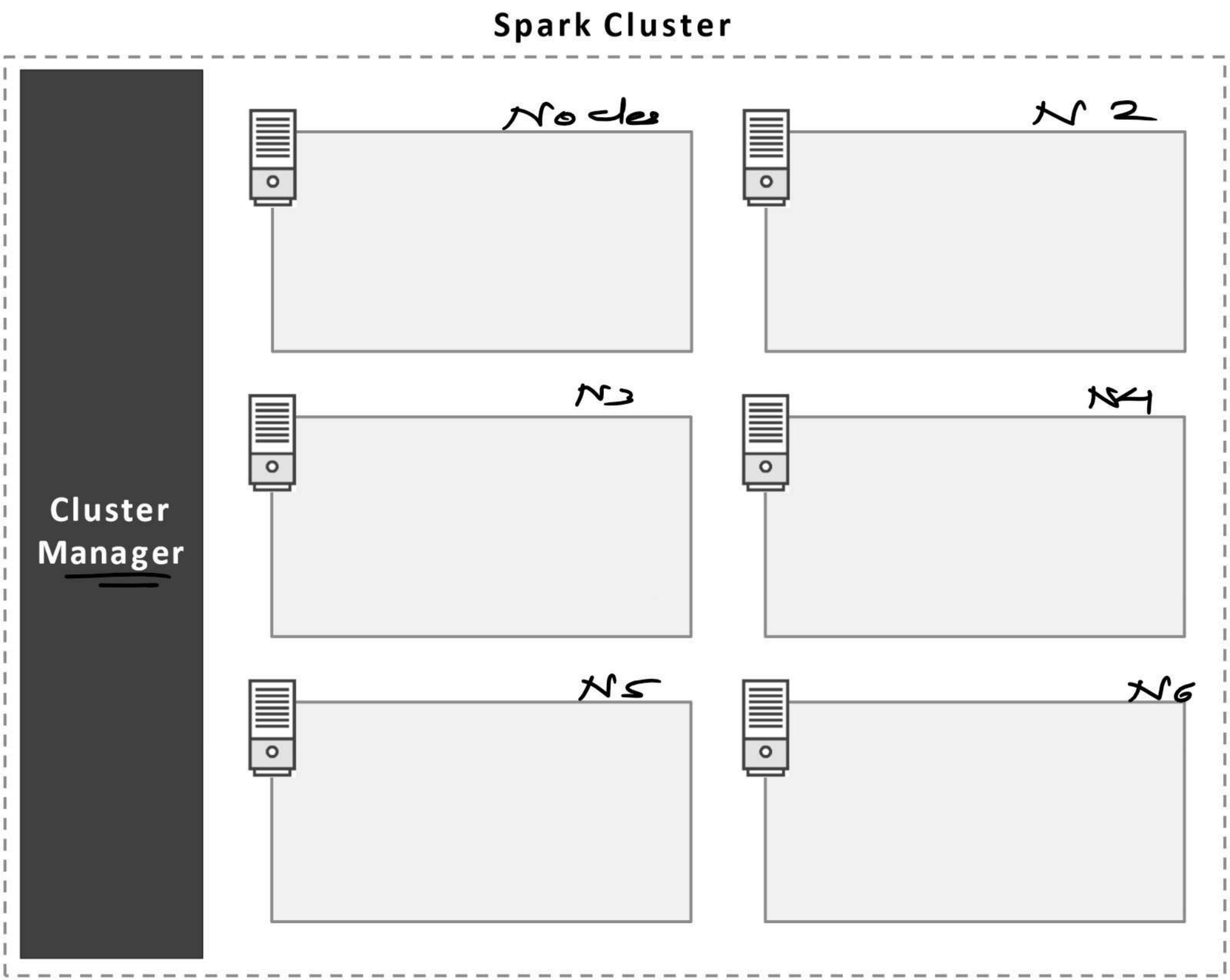
You can install Spark on a single machine, or you can install and run it together on multiple machines.

So a cluster is basically a group of machines or nodes. They may be virtual machines as well.

Then, it also has a cluster manager. The cluster manager allocates resources on the cluster to Spark applications.

Cluster is a group of machines / nodes

Cluster Manager allocates resources to Spark Applications on Cluster

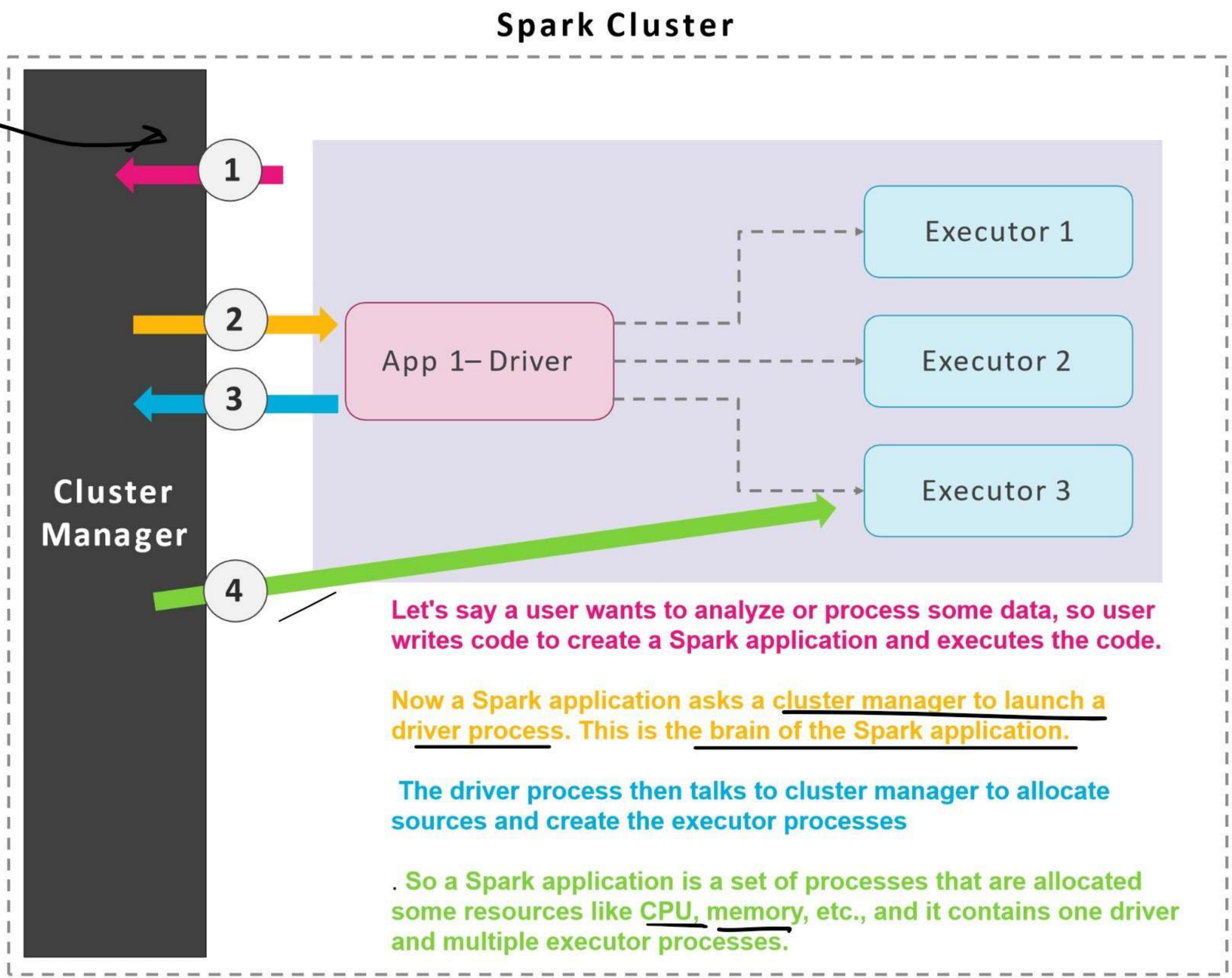


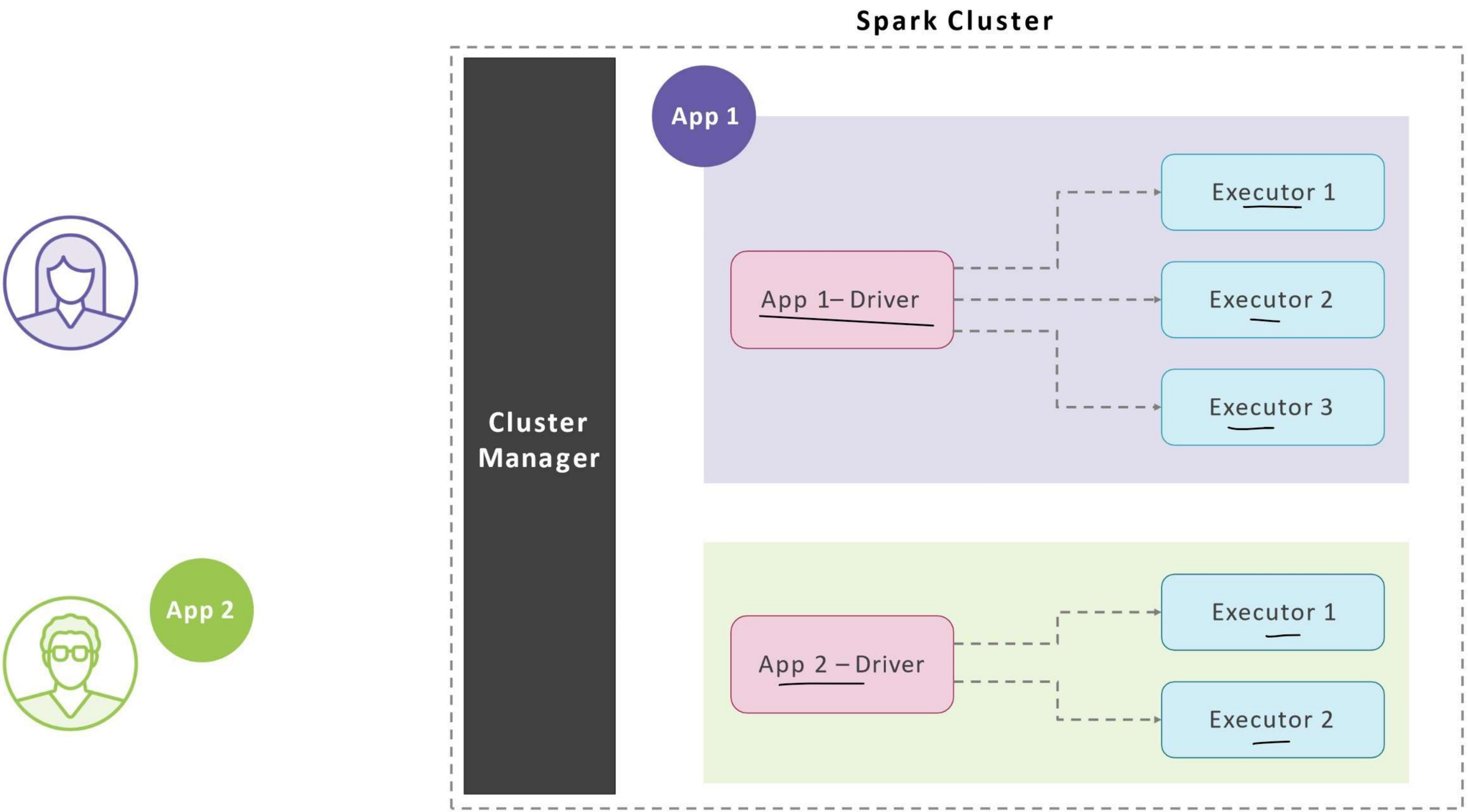
submit the spark code

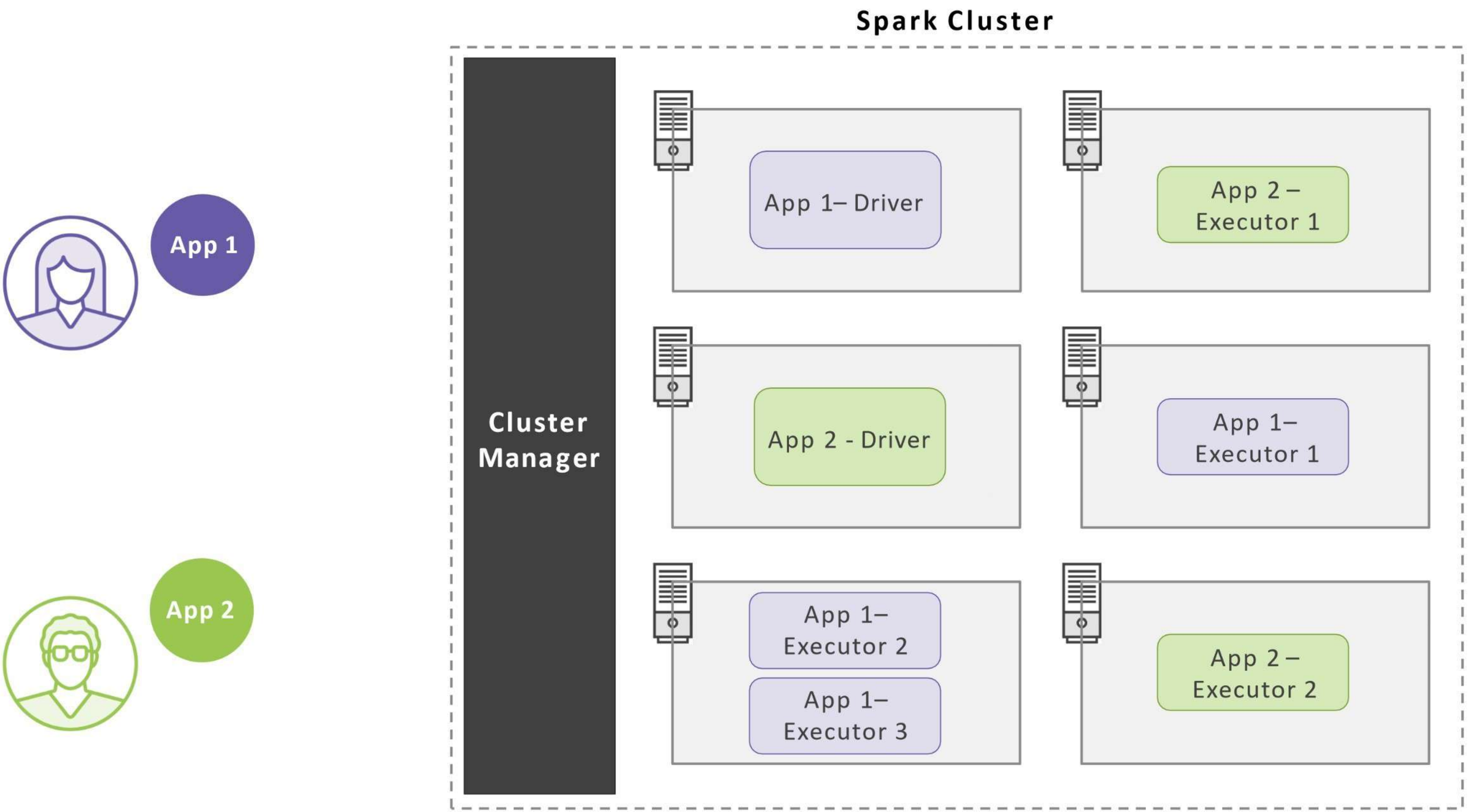
App 1

Spark Application is a set of processes

Has Driver & Executor processes




Spark Cluster
Cluster Manager
App 1
App 1– Driver
Executor 1
Executor 2
Executor 3
App 2
App 2 – Driver
Executor 1
Executor 2


Spark Cluster
App 1
App 2
Cluster Manager
App 1– Driver
App 2 – Executor 1
App 2 - Driver
App 1– Executor 1
App 1– Executor 2
App 1– Executor 3
App 2 – Executor 2

DRIVER

Driver takes input from user

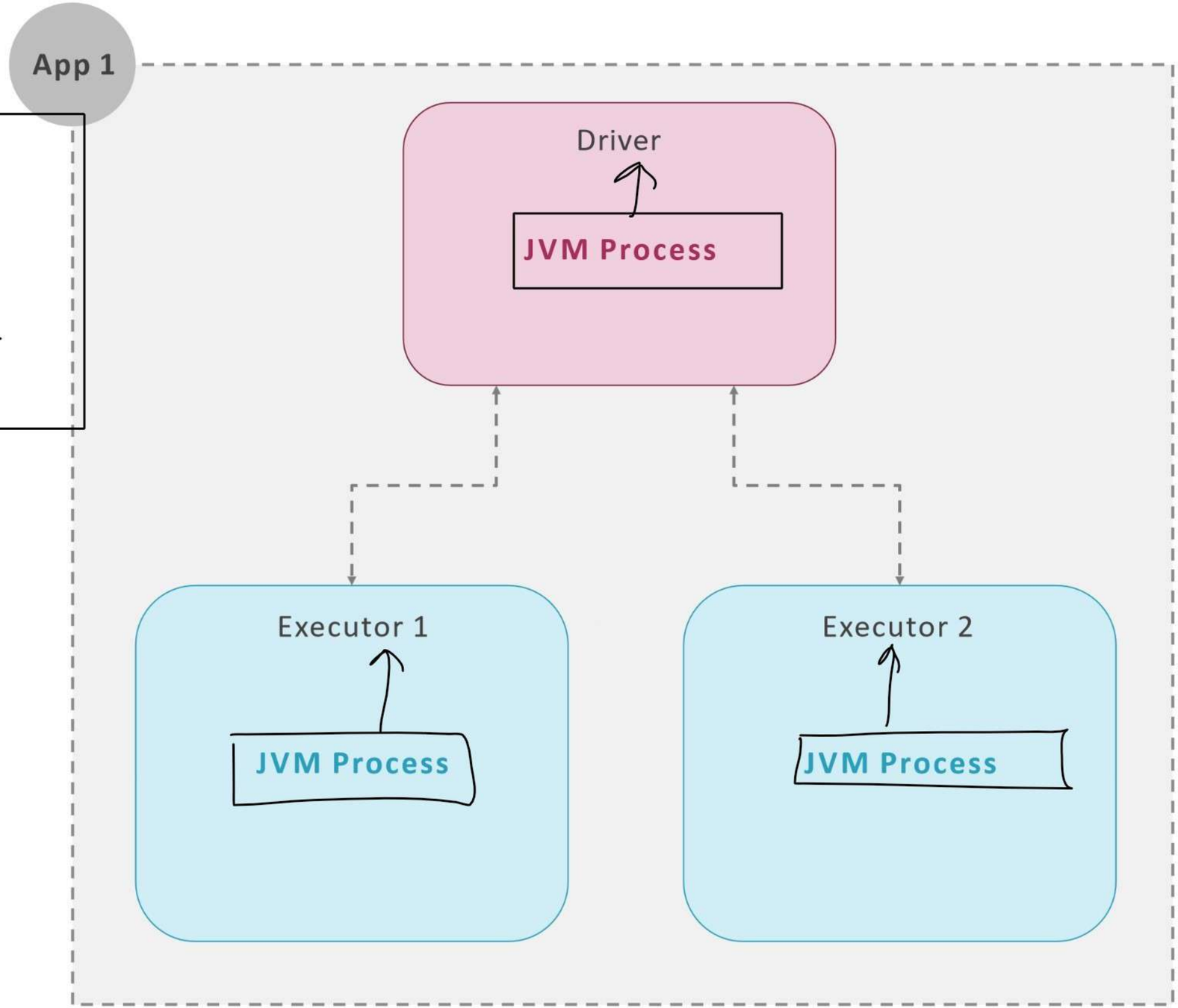
Determines how to execute a job

Analyzes job & distributes work to executors

EXECUTOR

Executors are responsible for executing the work (or code)

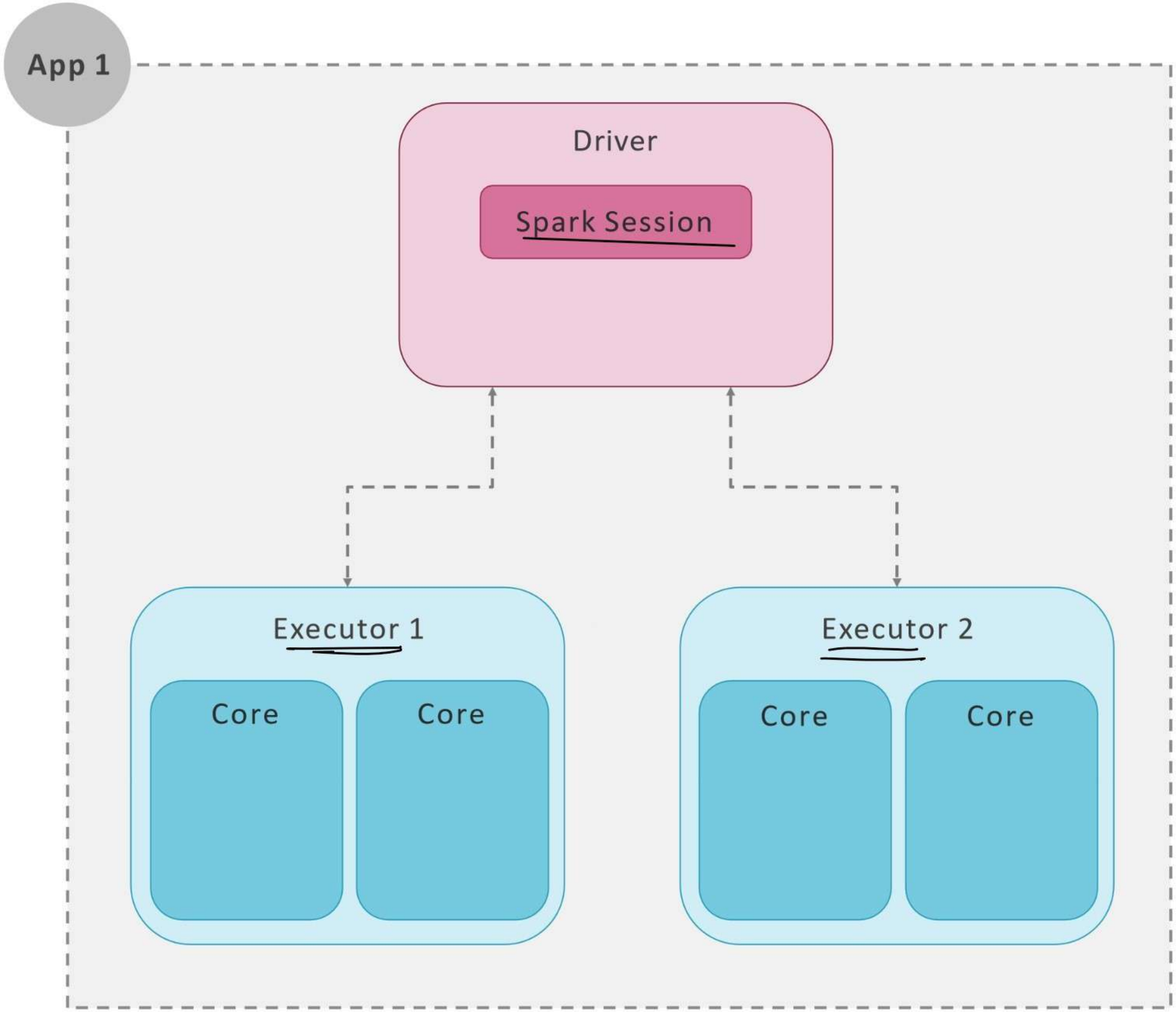
Returns result back to Driver



Spark Session is the entry point to all functionality of Spark

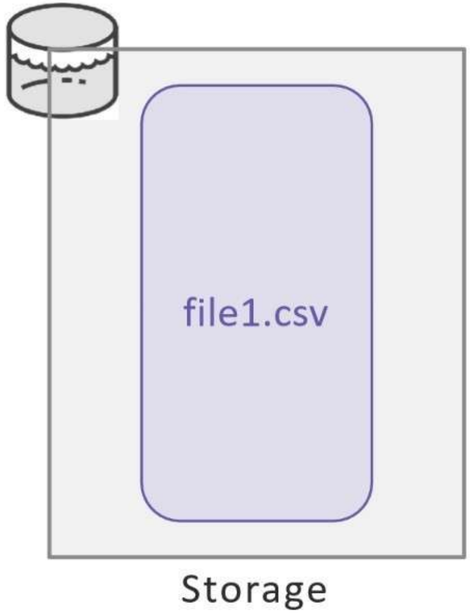
Use Spark Session to read file, create objects, run queries etc.

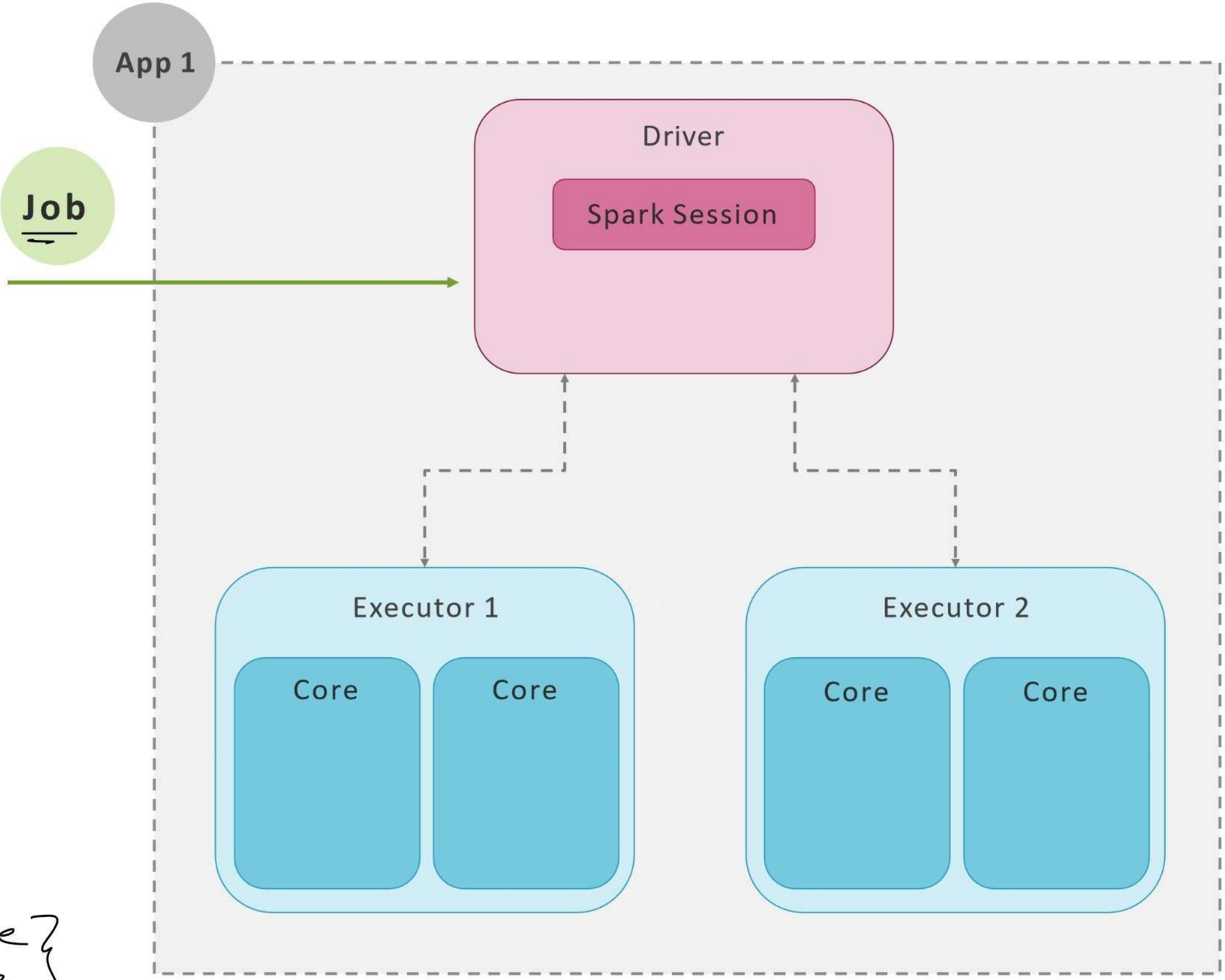
Executor Size
2 Cores & 14 GB RAM
(example)



1. Read file1.csv from Storage
2. Apply transformations
3. Write processed data to Storage

How Execution Happens?





2 core
14 GB

1. Read file1.csv from Storage
2. Apply transformations
3. Write processed data to Storage

App 1

Job

Driver

Spark Session

Logically split file1 in 4 parts

Executor 1

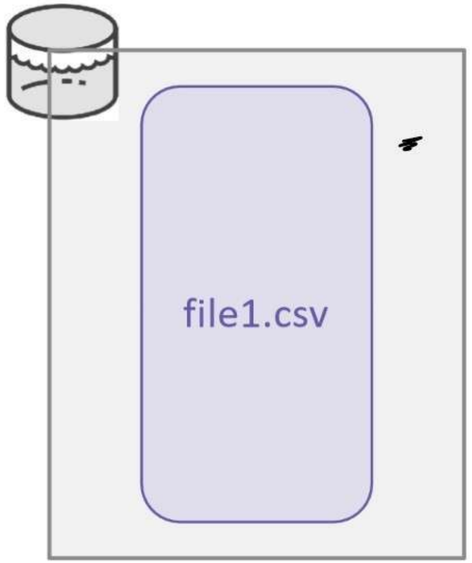
Core

Core

Executor 2

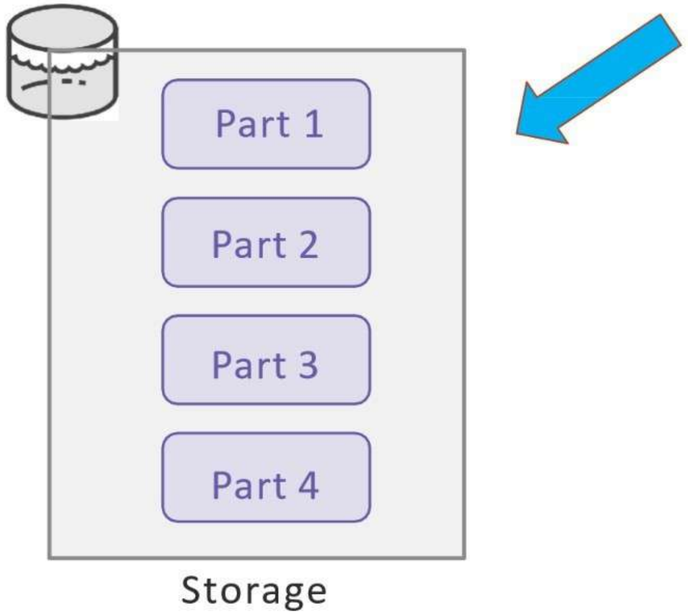
Core

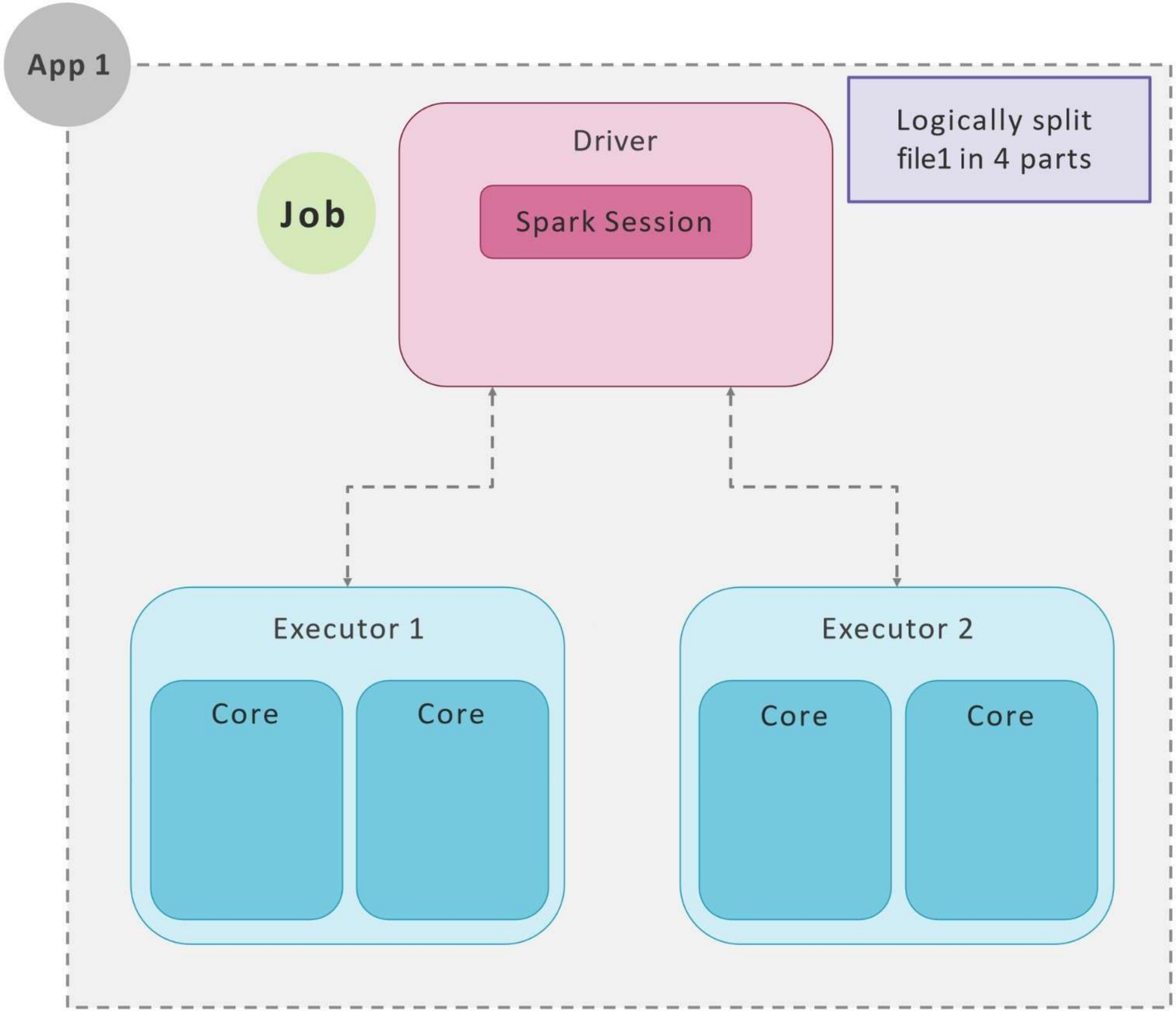
Core



Storage

1. Read file1.csv from Storage
2. Apply transformations
3. Write processed data to Storage

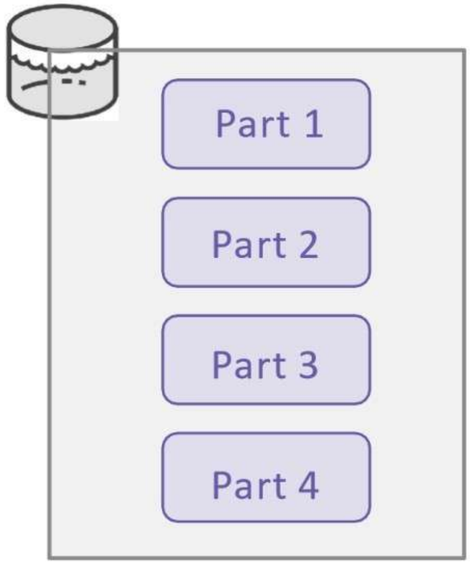


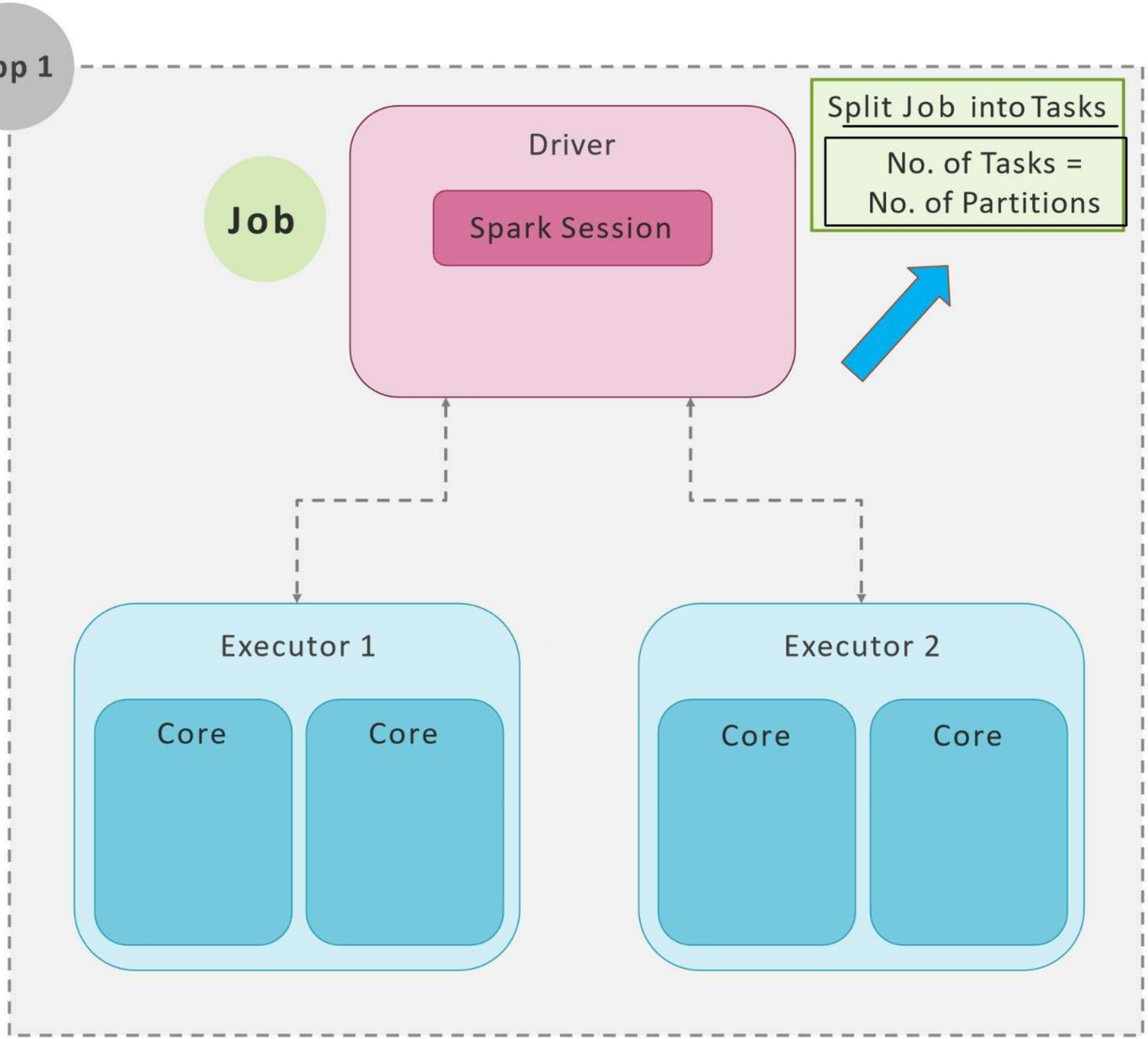


1 Job → 4 Task

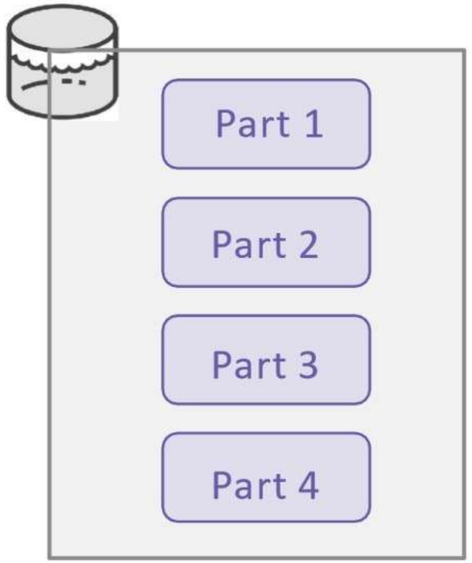
No. of partitions 4

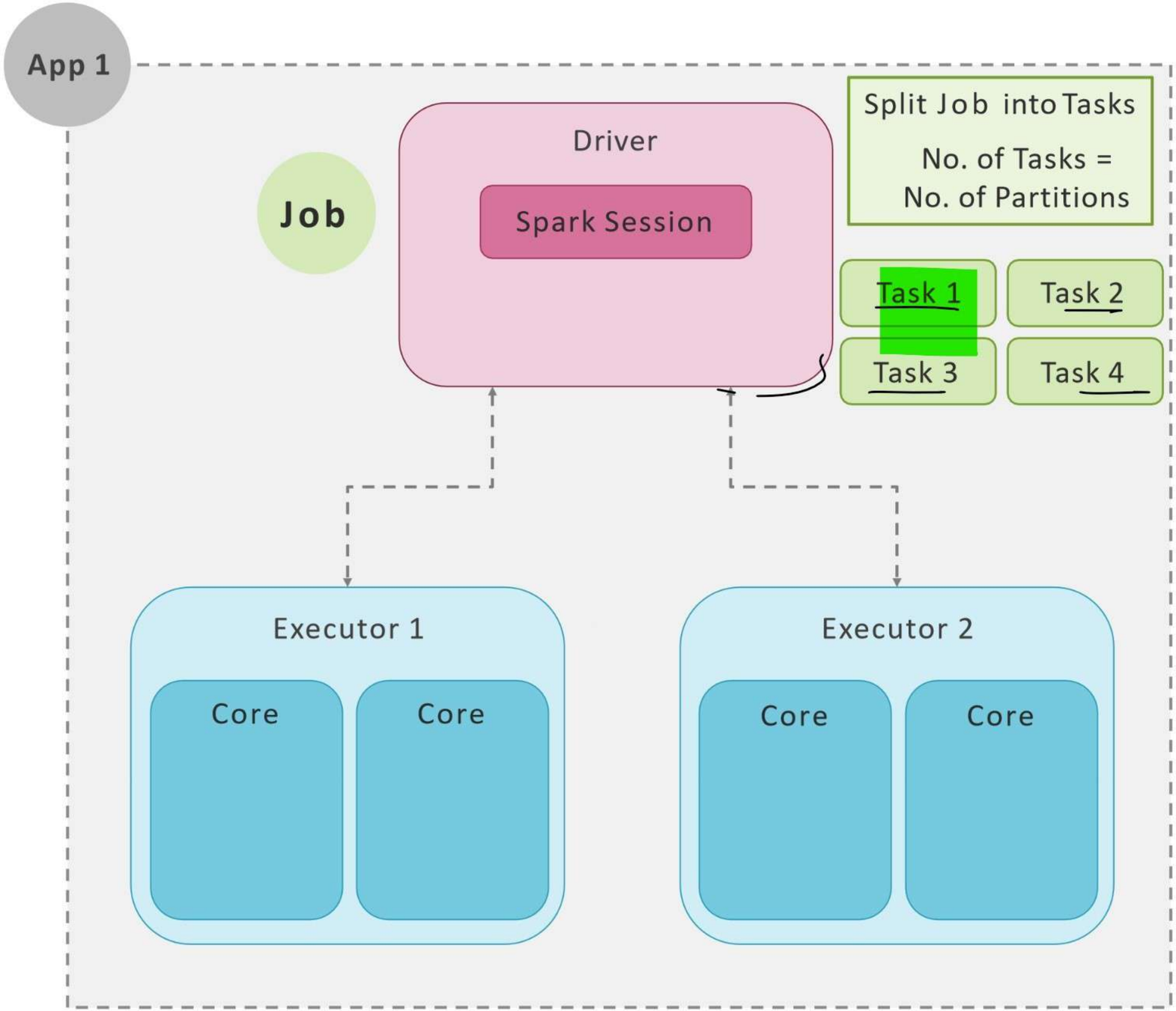
1. Read file1.csv from Storage
2. Apply transformations
3. Write processed data to Storage



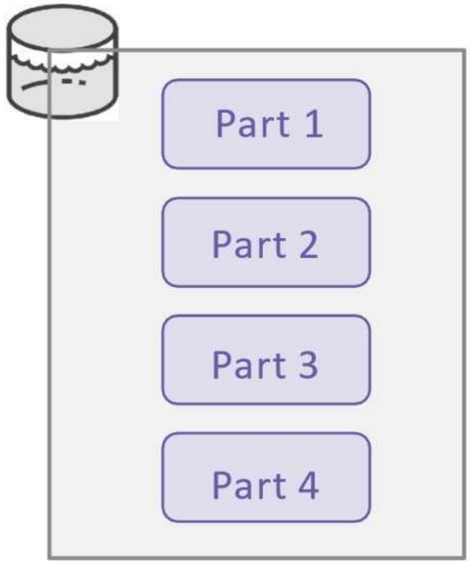


1. Read file1.csv from Storage
2. Apply transformations
3. Write processed data to Storage

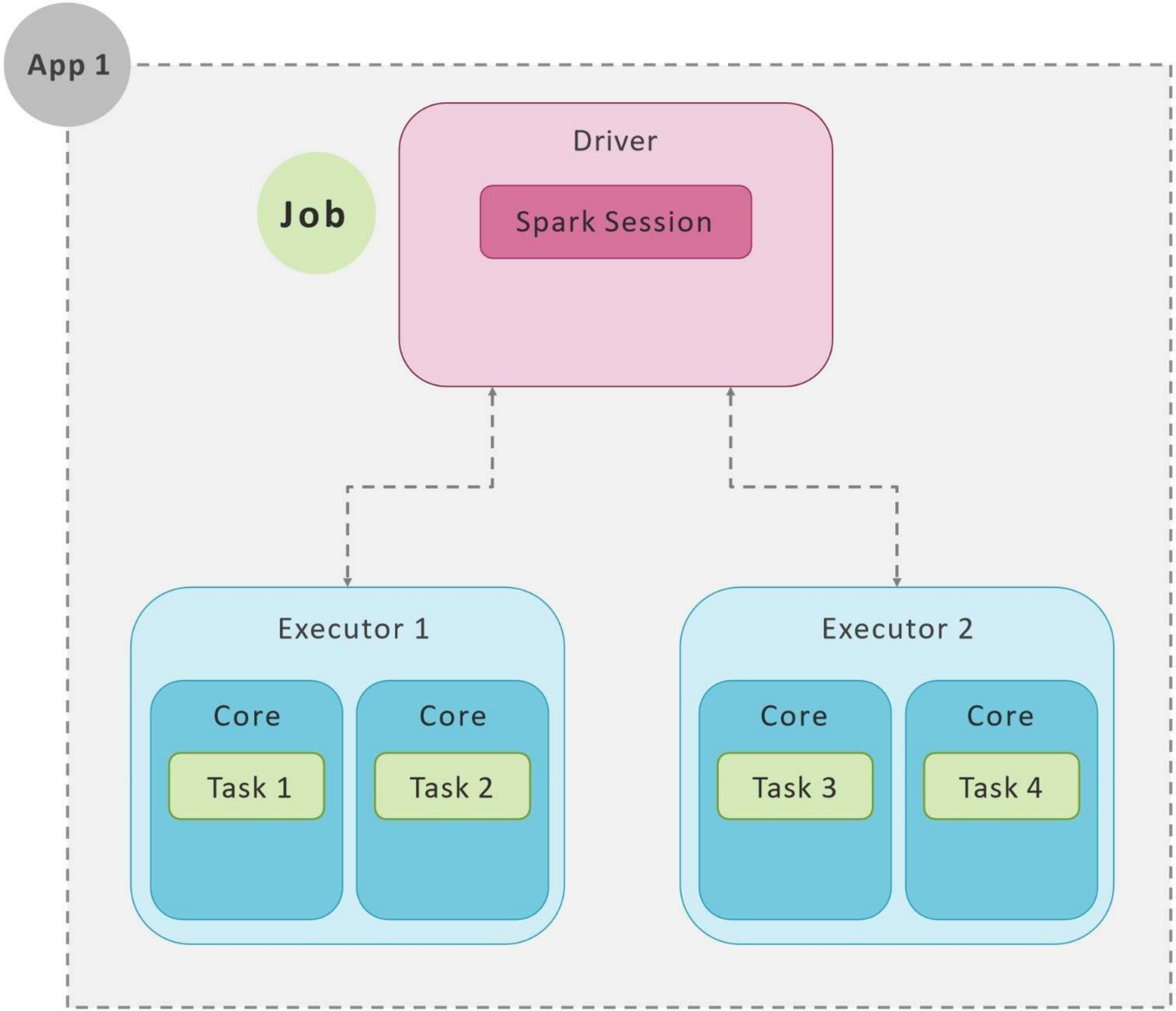


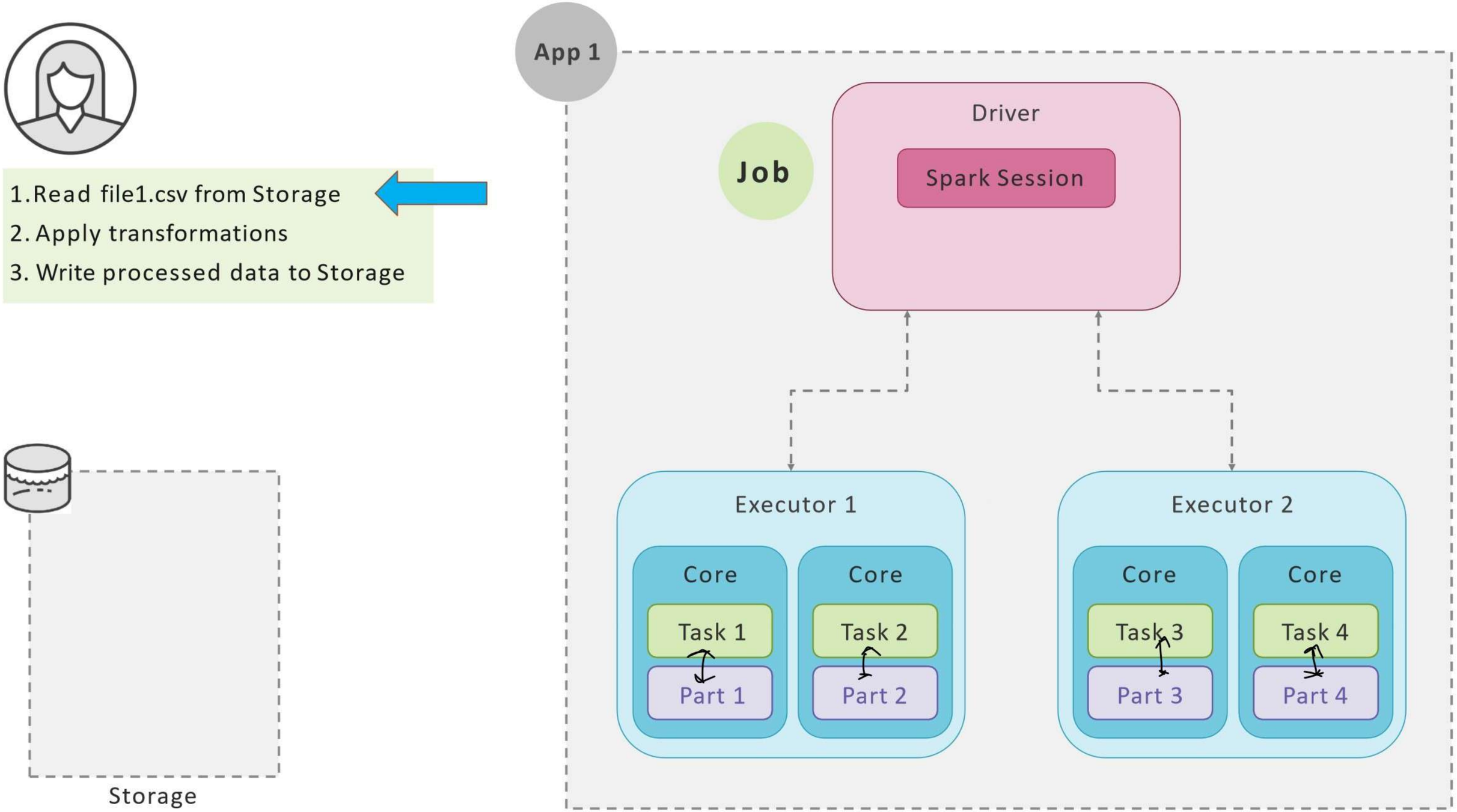


1. Read file1.csv from Storage
2. Apply transformations
3. Write processed data to Storage

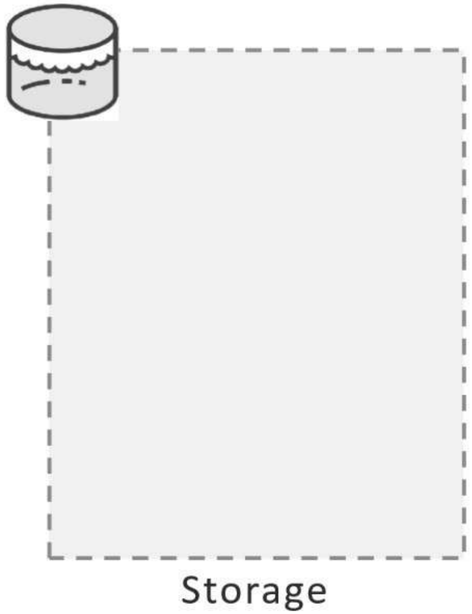


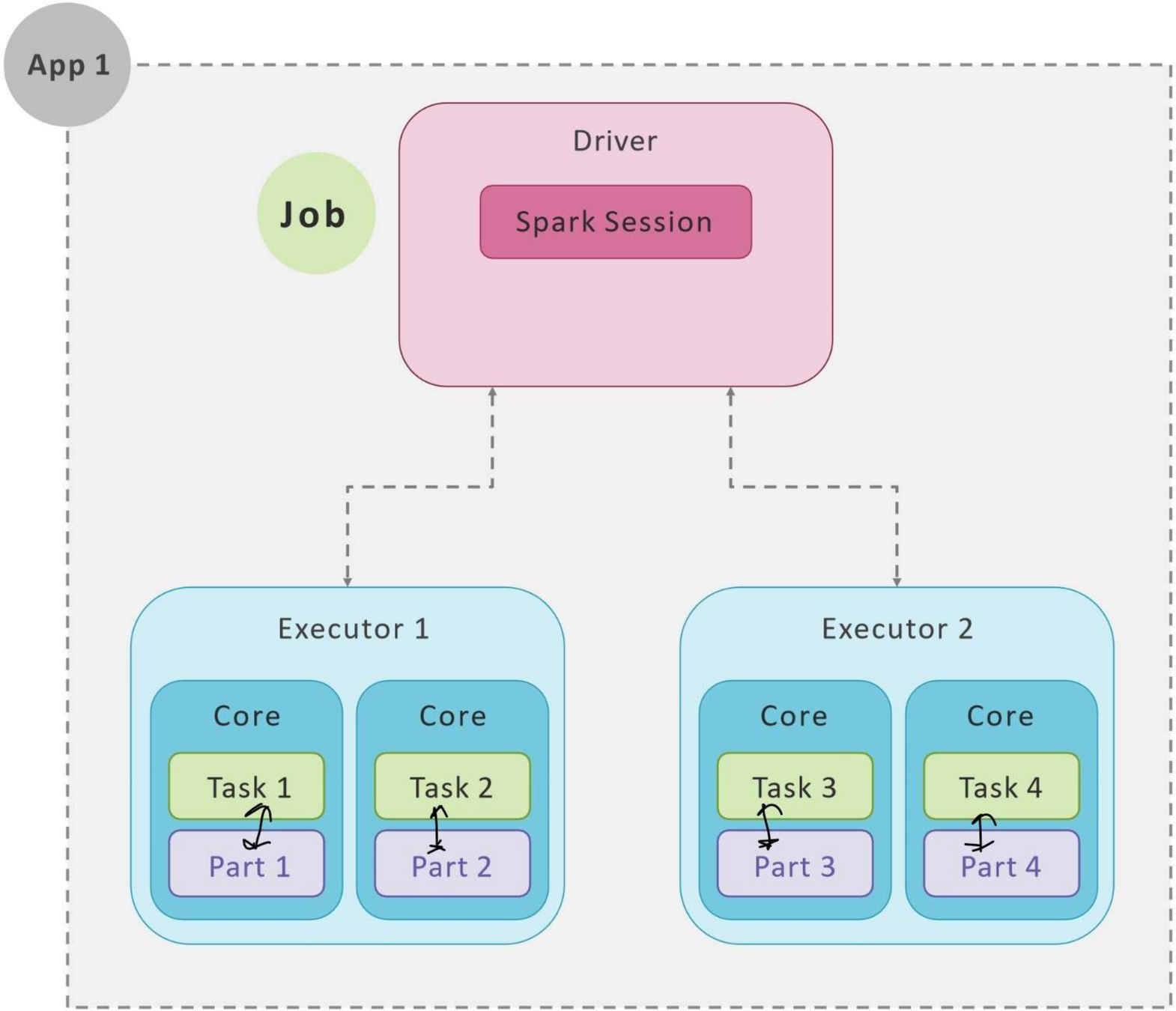
Storage

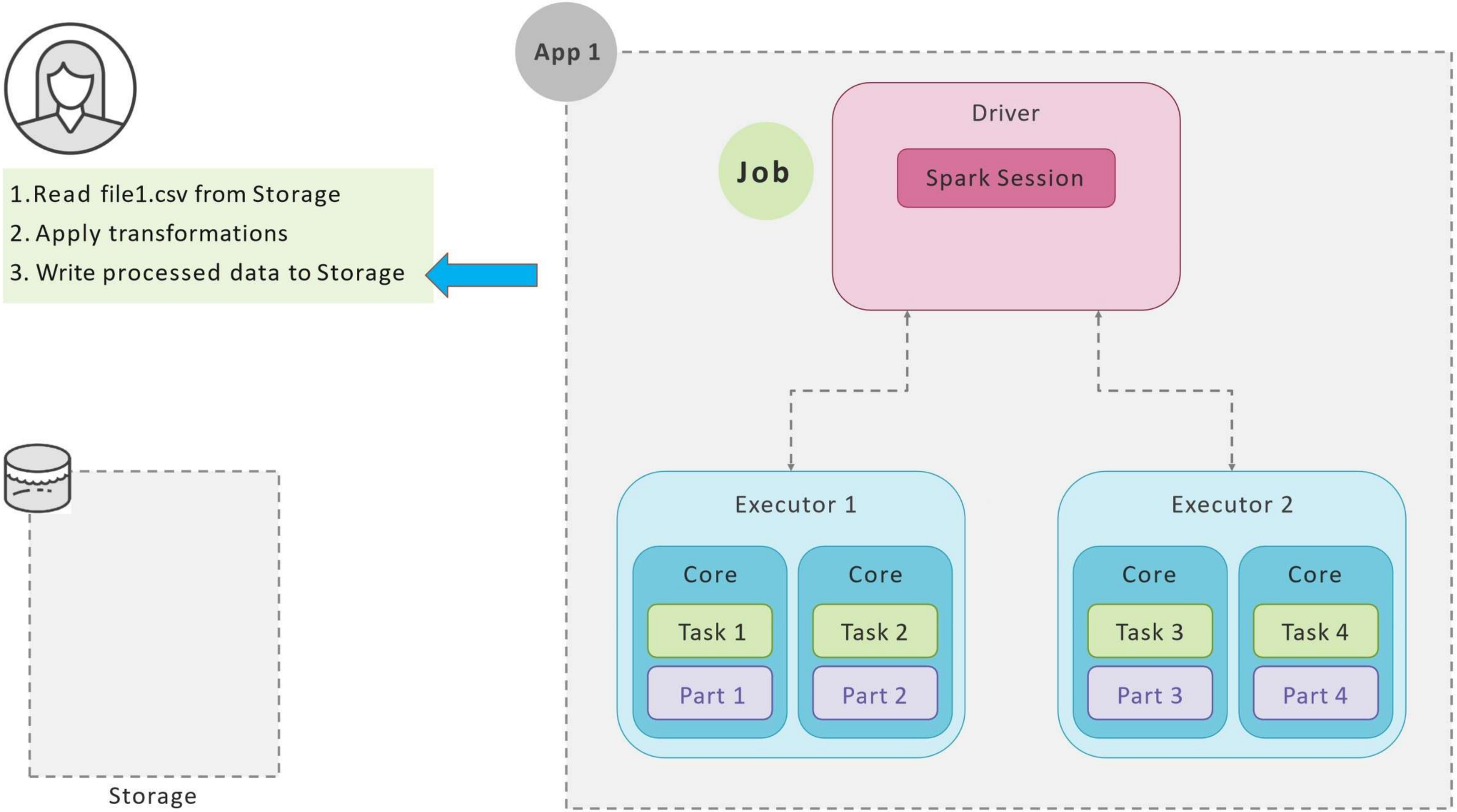


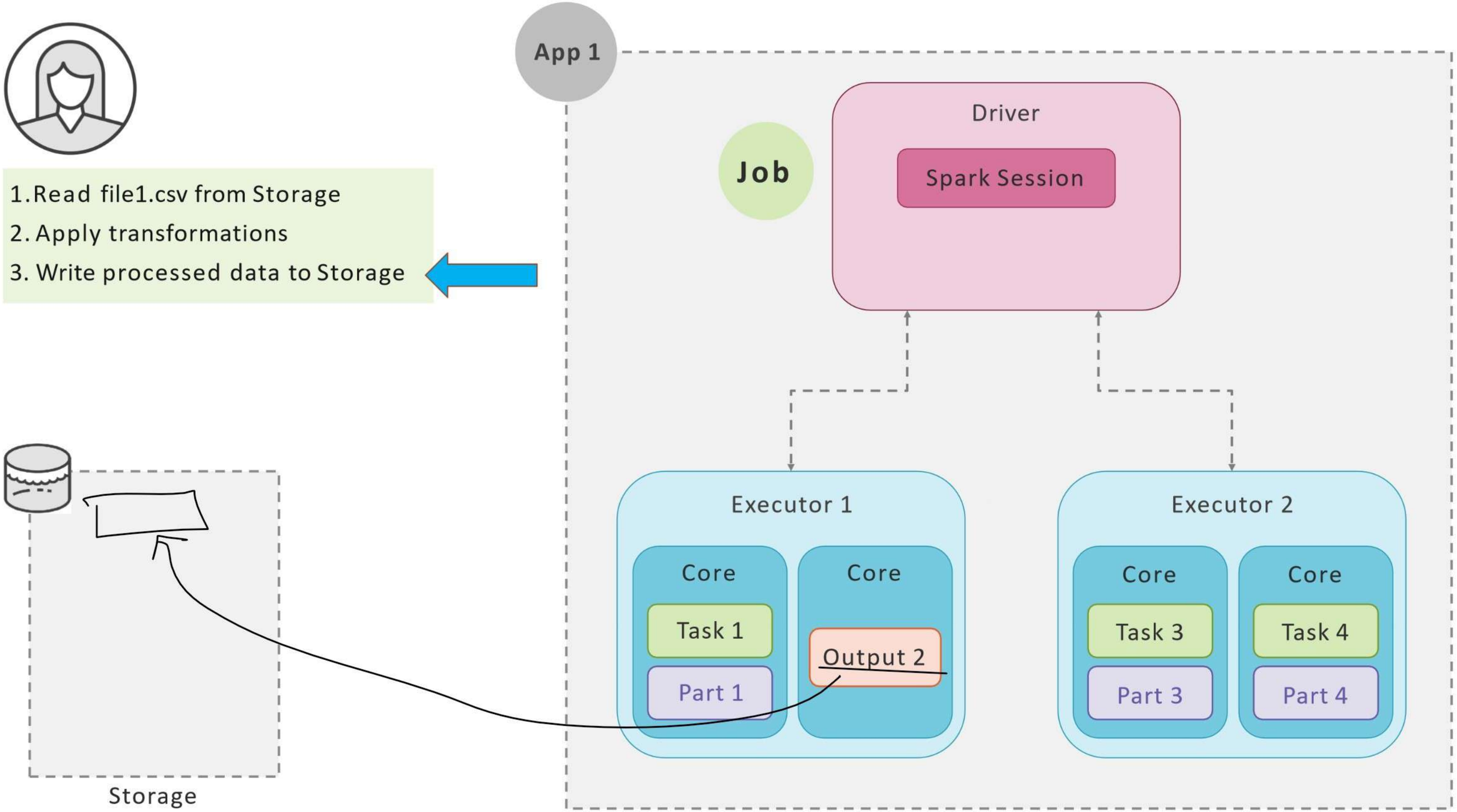

1. Read file1.csv from Storage
2. Apply transformations
3. Write processed data to Storage
Storage
App 1
Job
Driver
Spark Session
Executor 1
Core
Task 1
Part 1
Core
Task 2
Part 2
Executor 2
Core
Task 3
Part 3
Core
Task 4
Part 4

1. Read file1.csv from Storage
2. Apply transformations
3. Write processed data to Storage

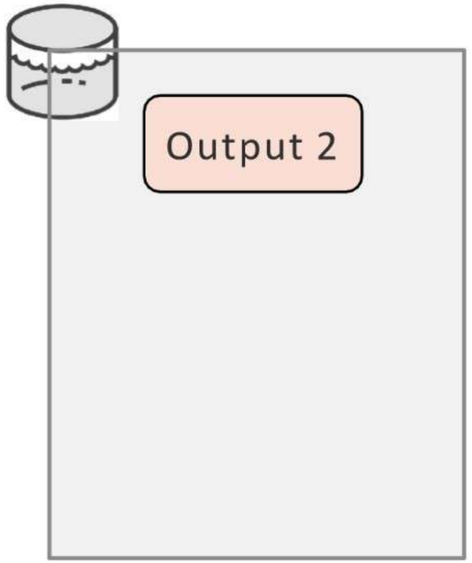


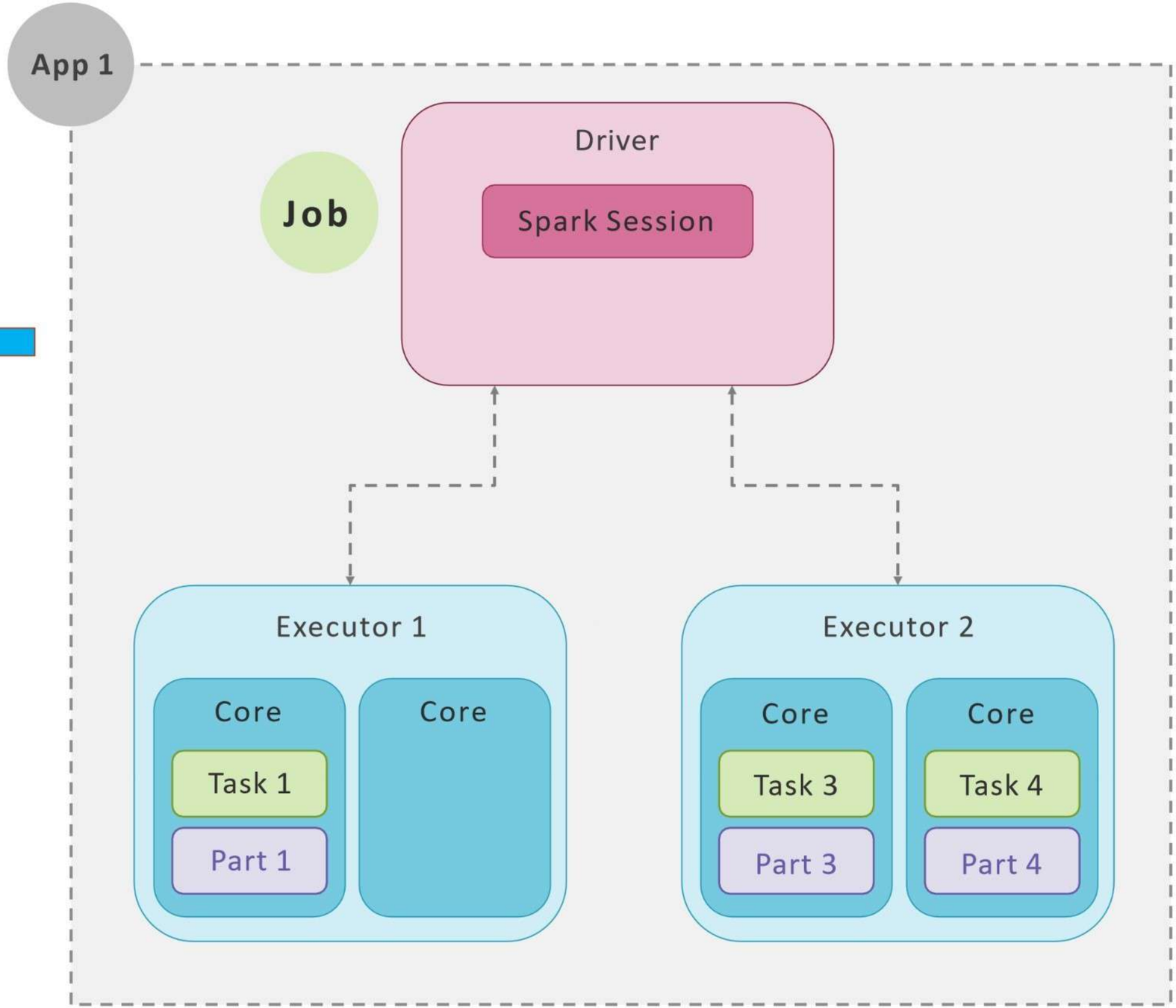



App 1
Job
Driver
Spark Session
1. Read file1.csv from Storage
2. Apply transformations
3. Write processed data to Storage
Executor 1
Core
Task 1
Part 1
Core
Task 2
Part 2
Executor 2
Core
Task 3
Part 3
Core
Task 4
Part 4
Storage

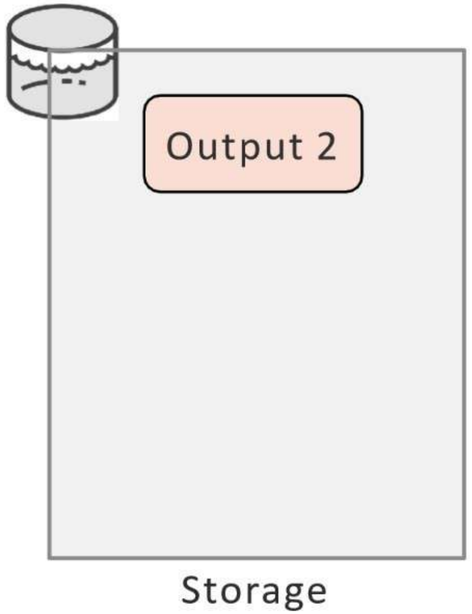

App 1
Job
Driver
Spark Session
1. Read file1.csv from Storage
2. Apply transformations
3. Write processed data to Storage
Executor 1
Core
Task 1
Part 1
Core
Output 2
Executor 2
Core
Task 3
Part 3
Core
Task 4
Part 4
Storage

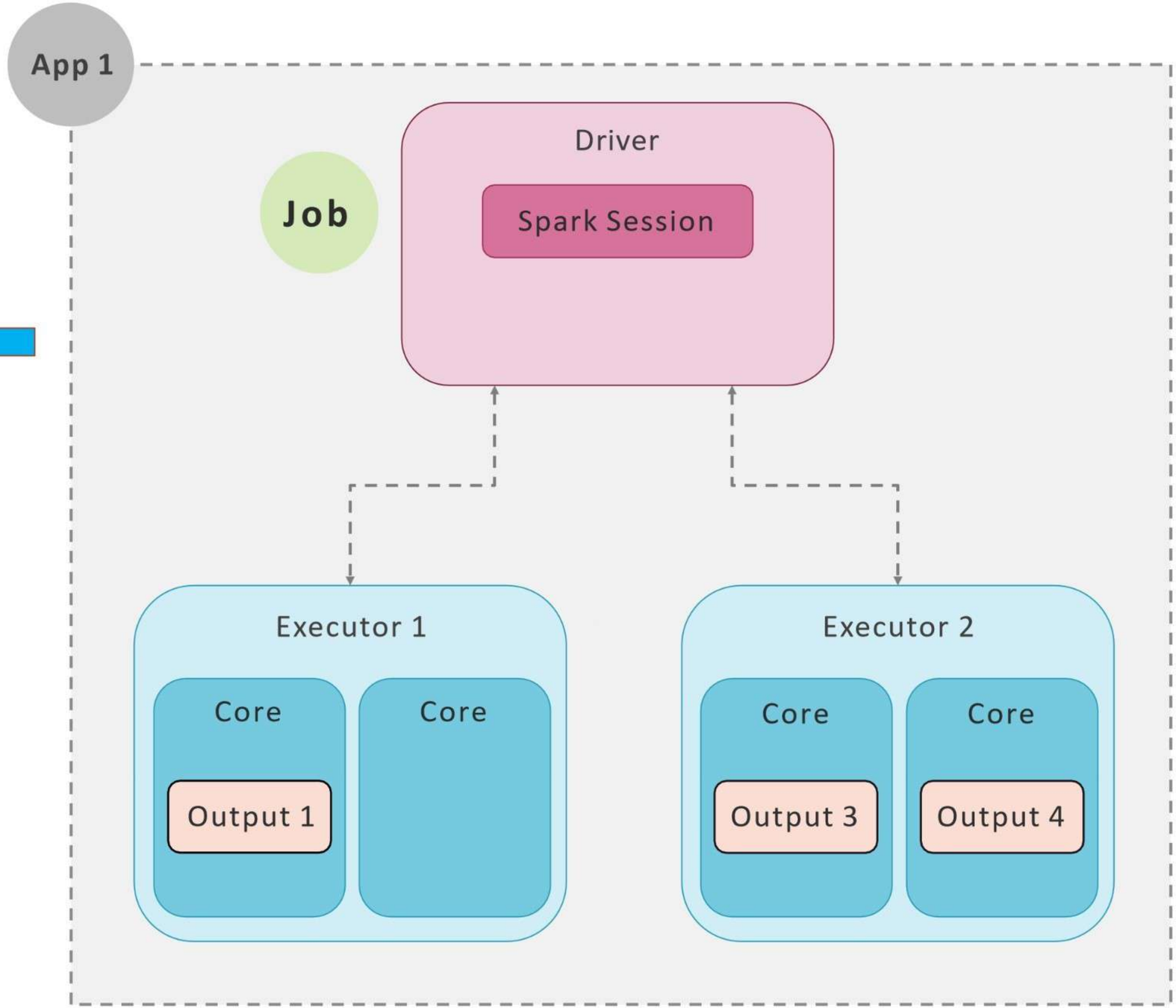
1. Read file1.csv from Storage
2. Apply transformations
3. Write processed data to Storage



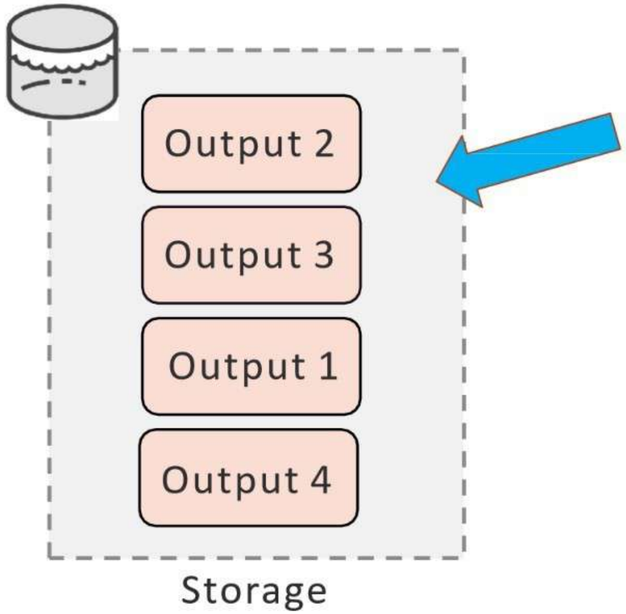


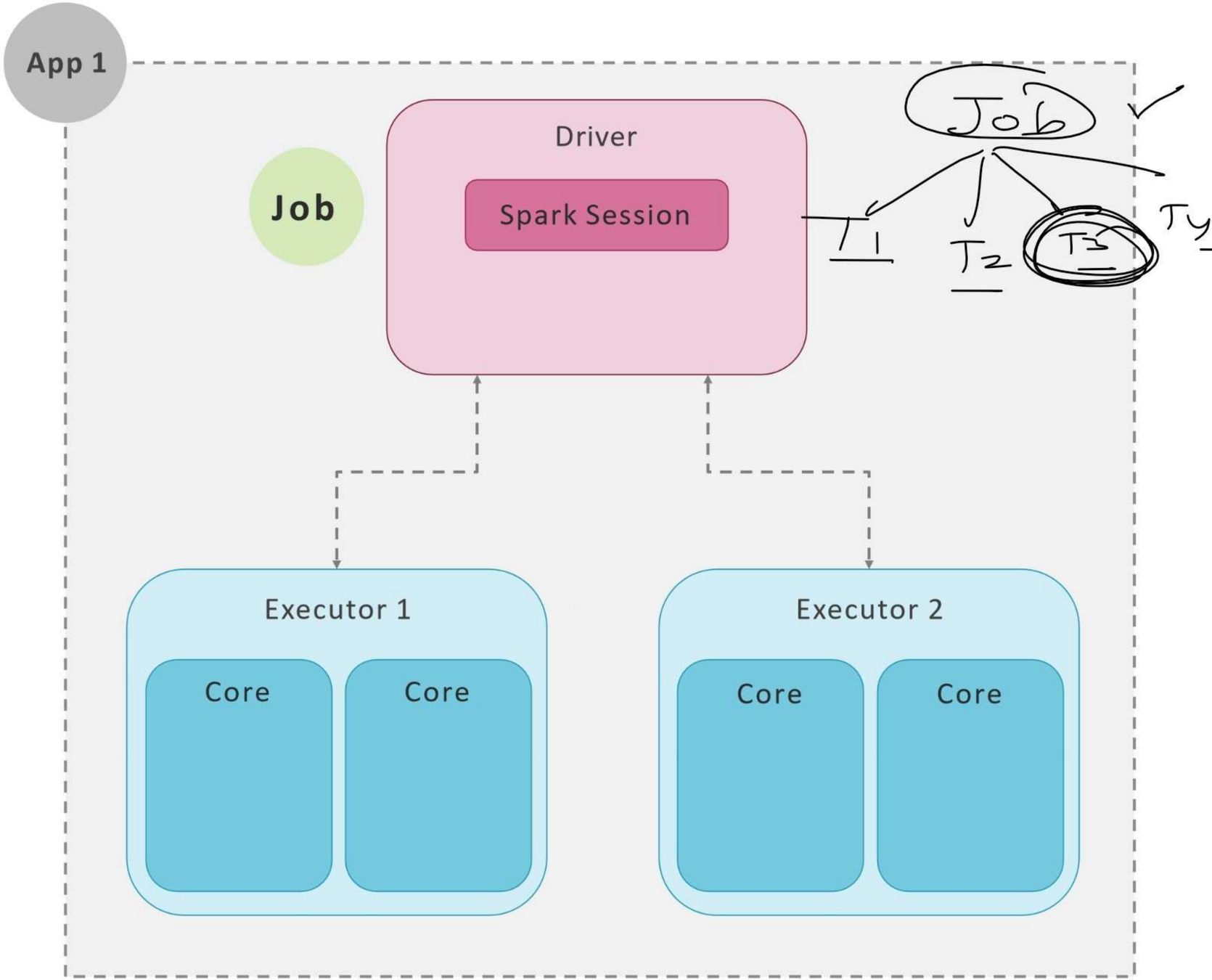
1. Read file1.csv from Storage
2. Apply transformations
3. Write processed data to Storage





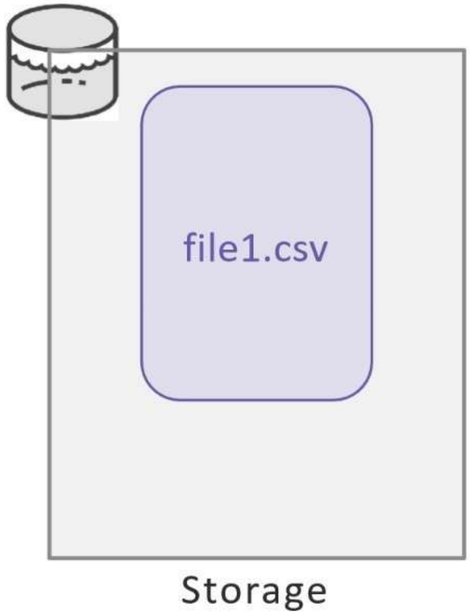
1. Read file1.csv from Storage
2. Apply transformations
3. Write processed data to Storage

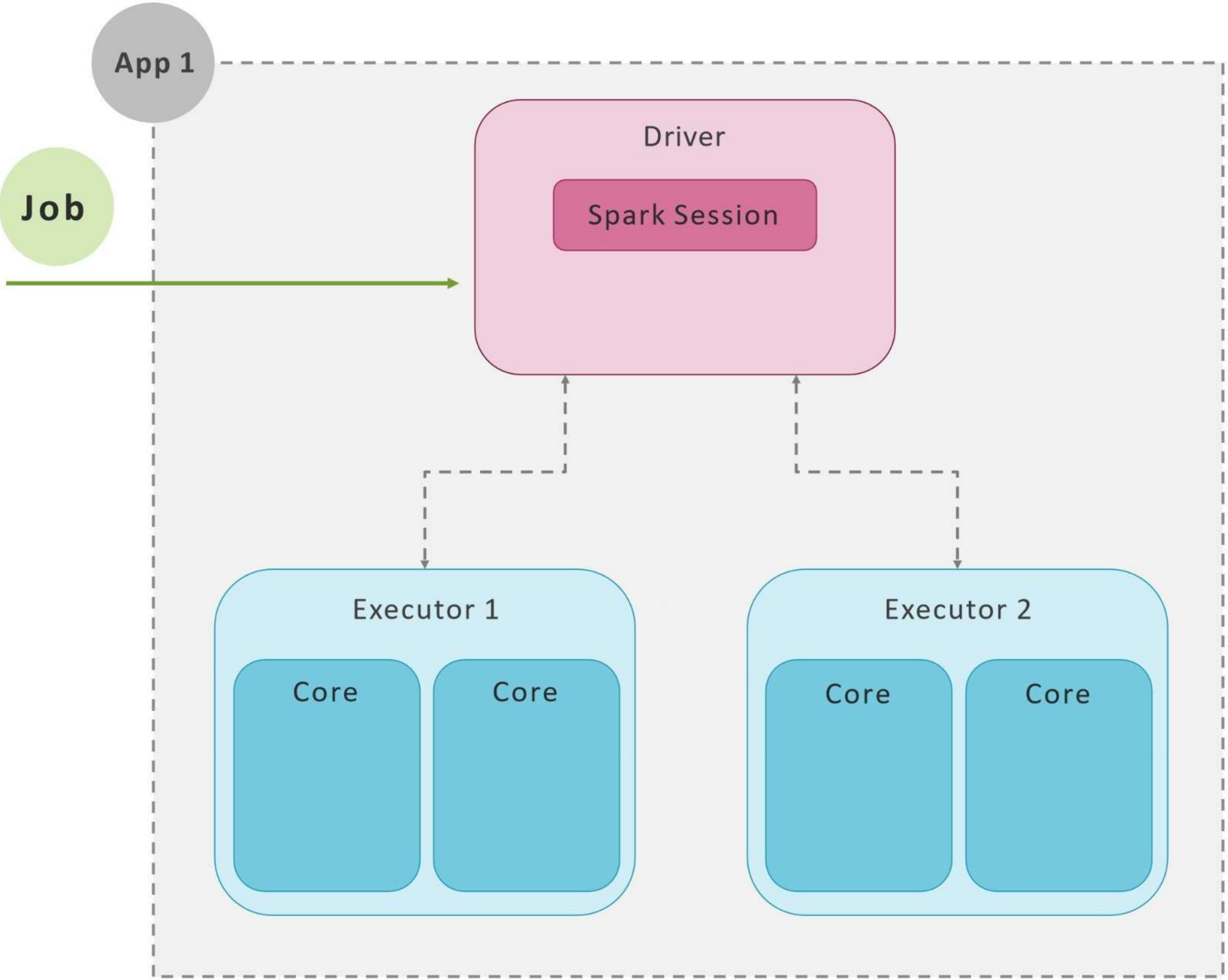




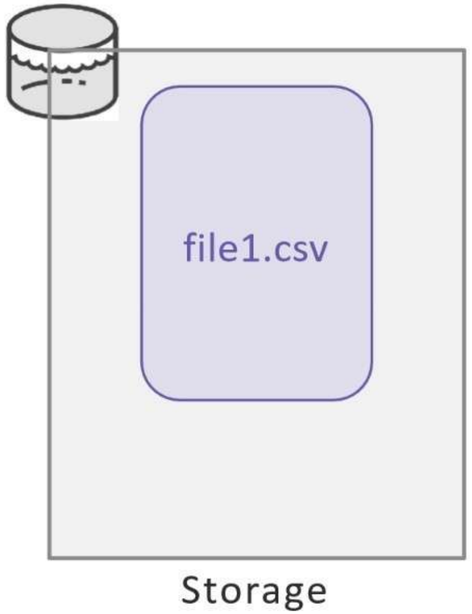
What if number of Tasks are lesser than available Cores?

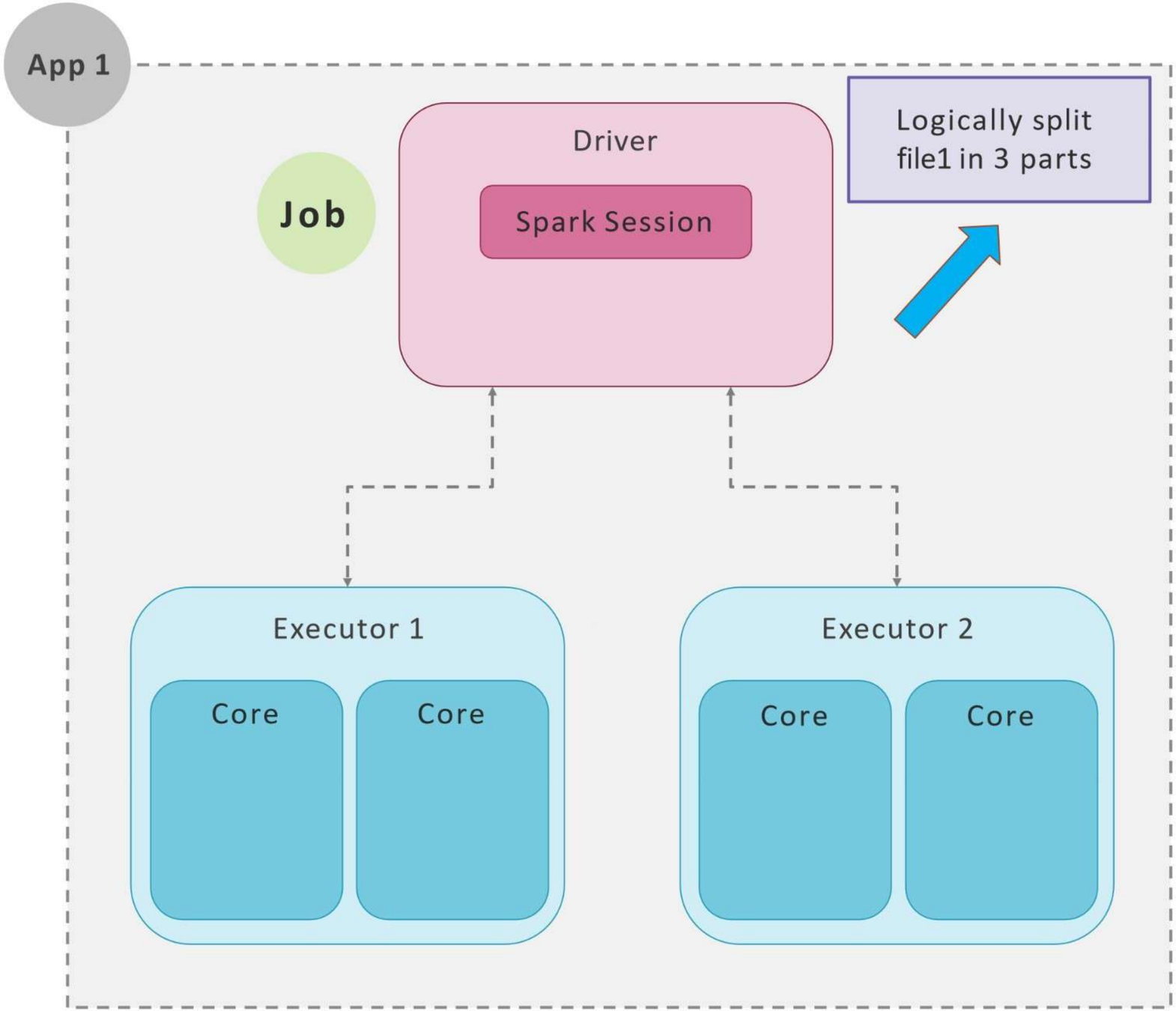
1. Read file1.csv from Storage
2. Apply transformations
3. Write processed data to Storage



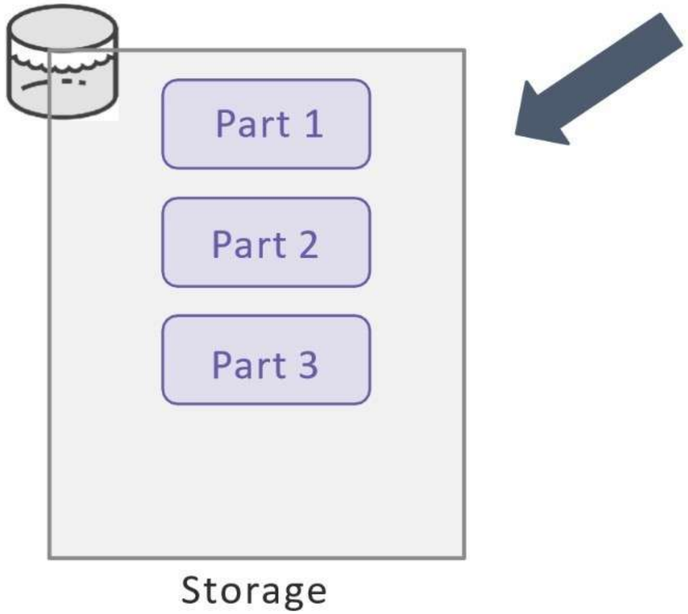


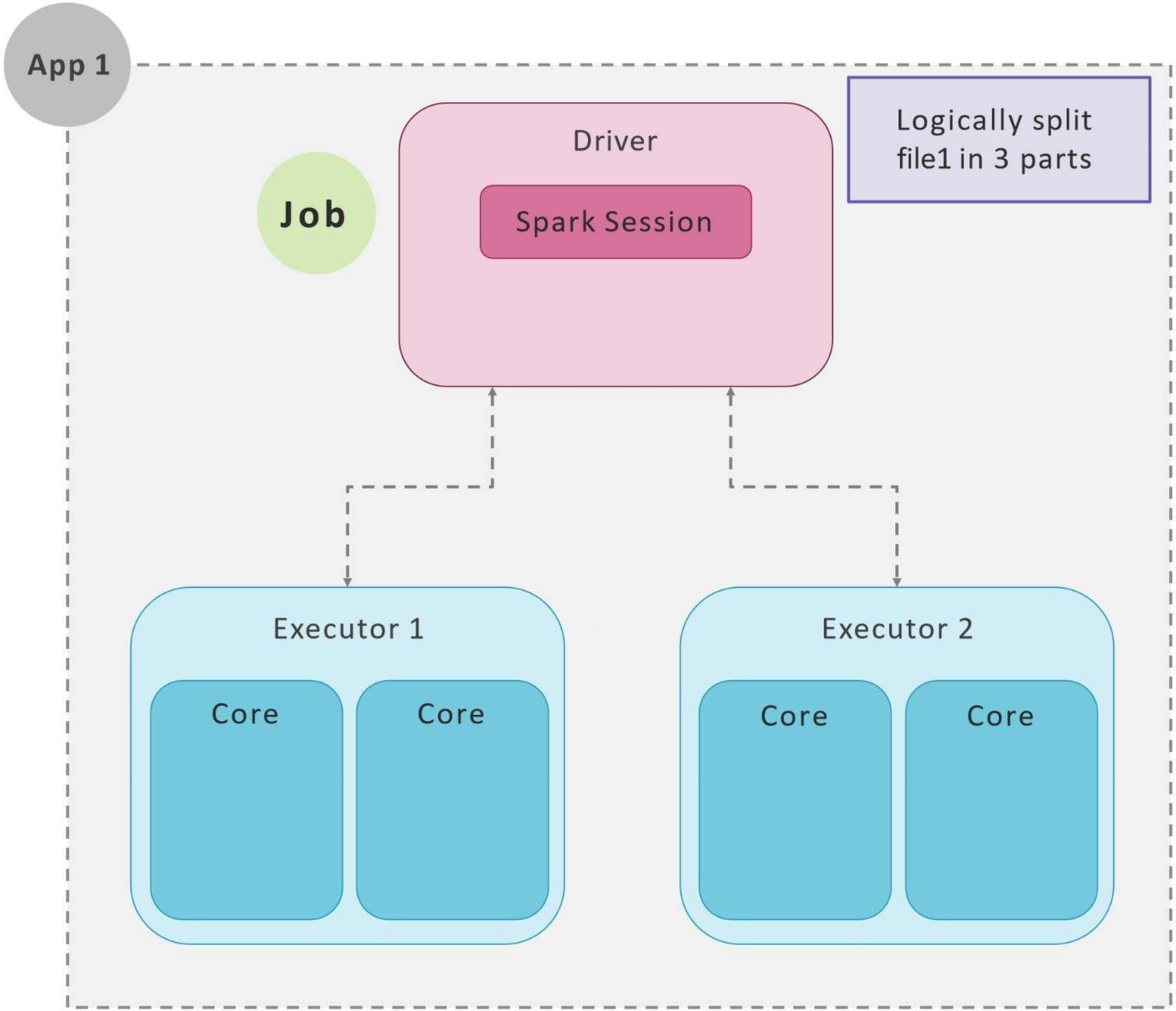
1. Read file1.csv from Storage
2. Apply transformations
3. Write processed data to Storage



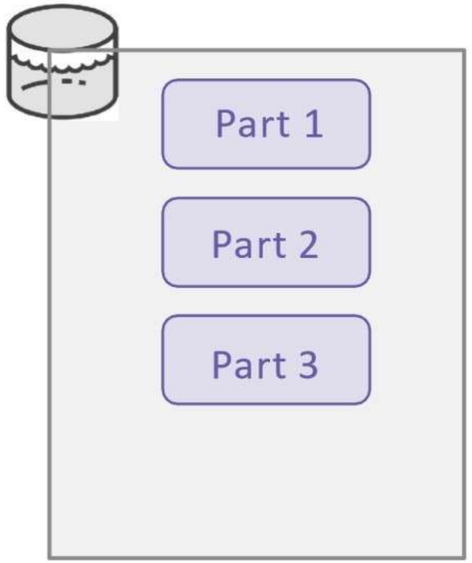


1. Read file1.csv from Storage
2. Apply transformations
3. Write processed data to Storage

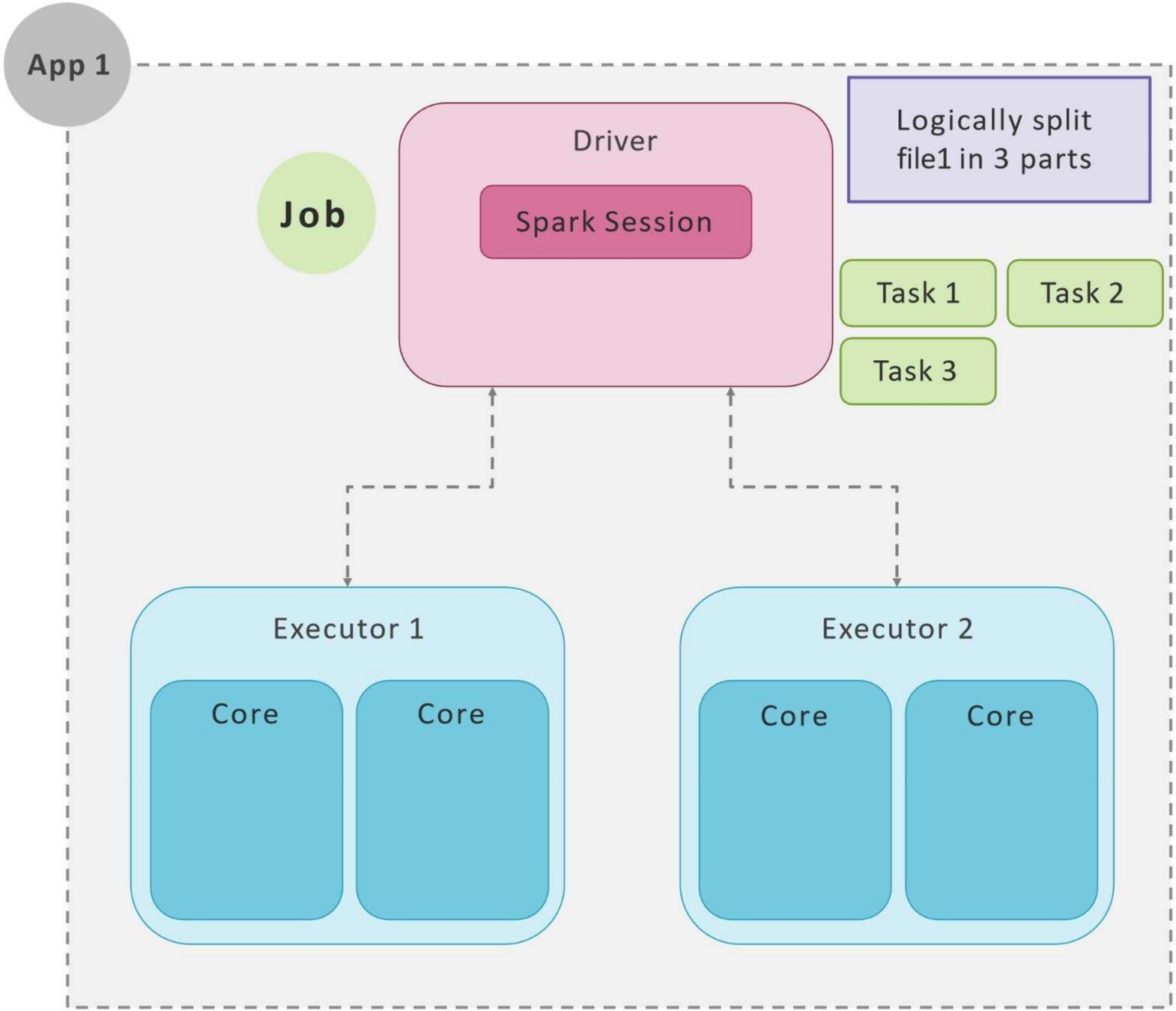




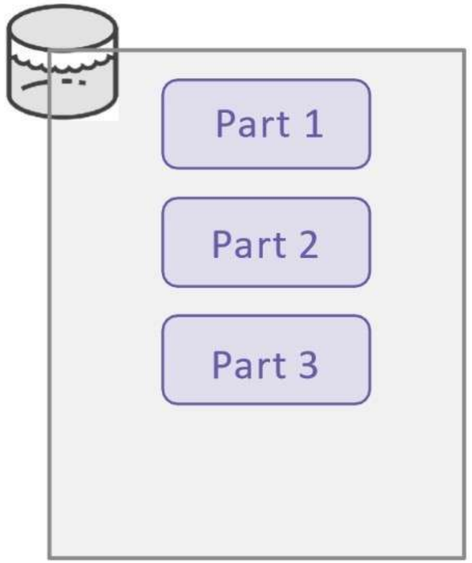
1. Read file1.csv from Storage
2. Apply transformations
3. Write processed data to Storage



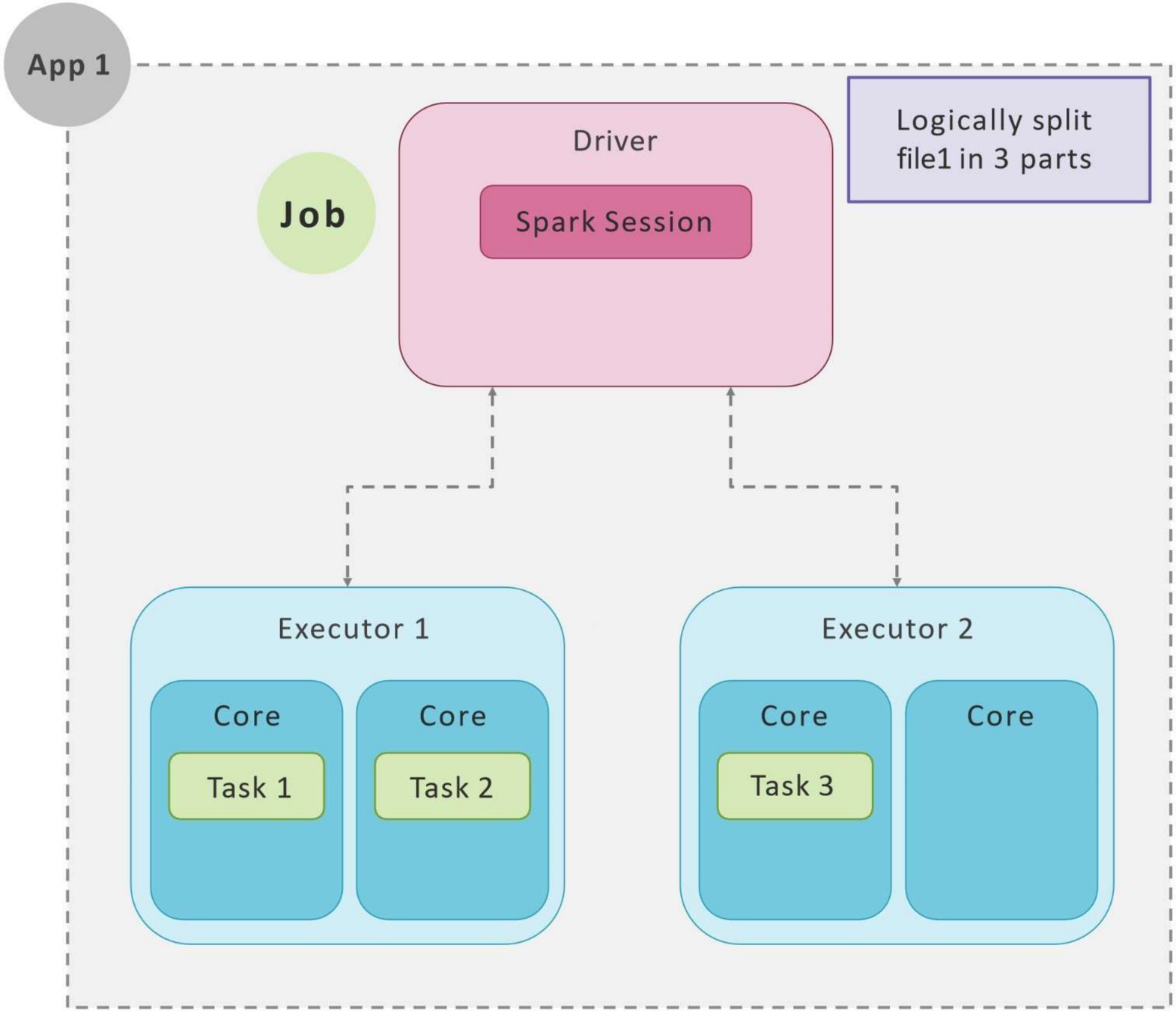
Storage



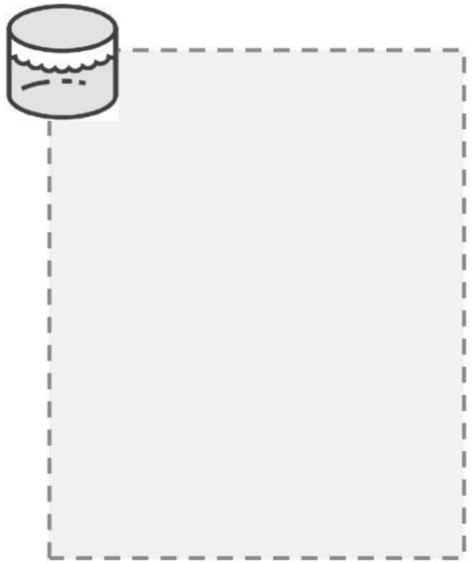
1. Read file1.csv from Storage
2. Apply transformations
3. Write processed data to Storage

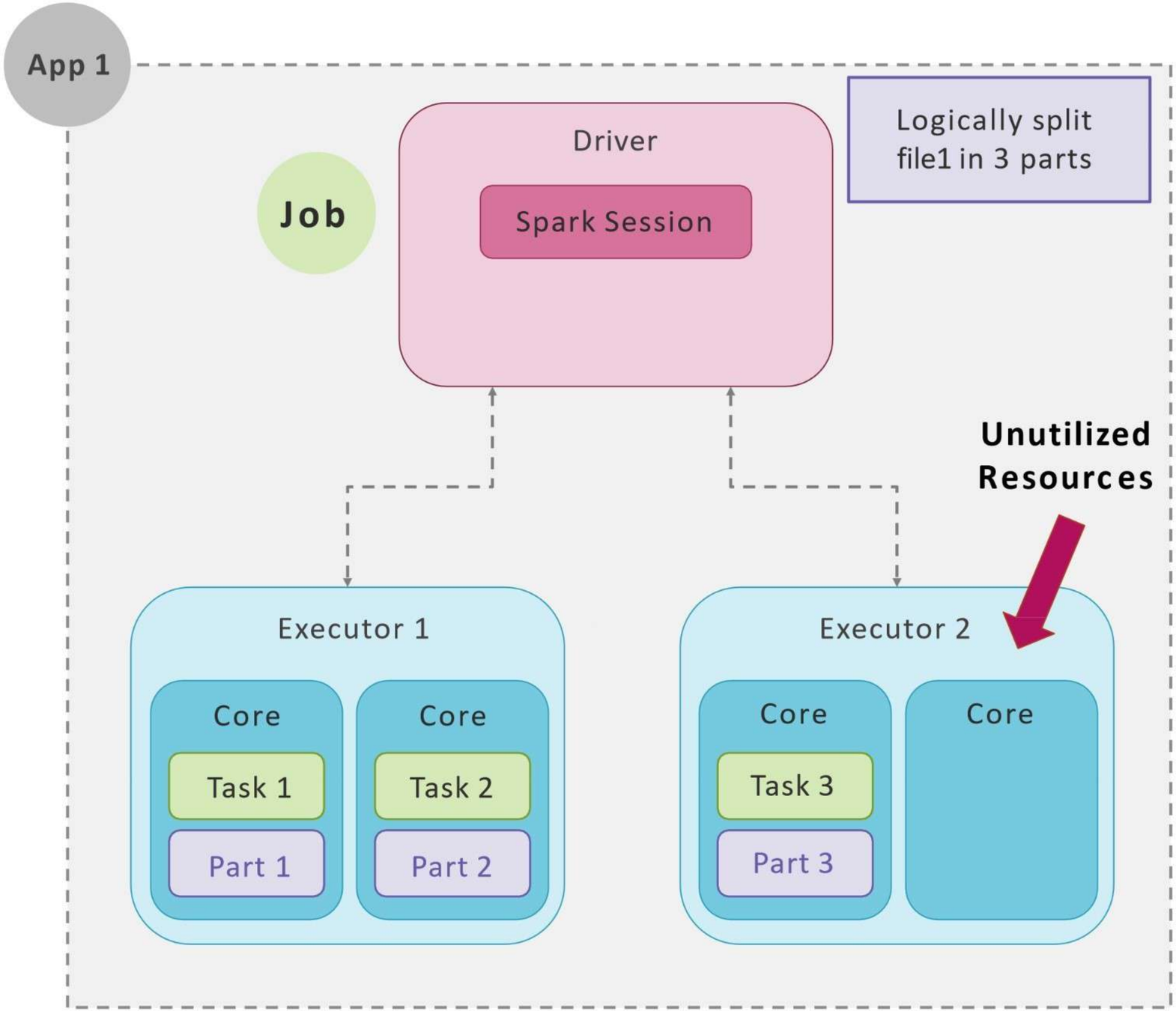


Storage



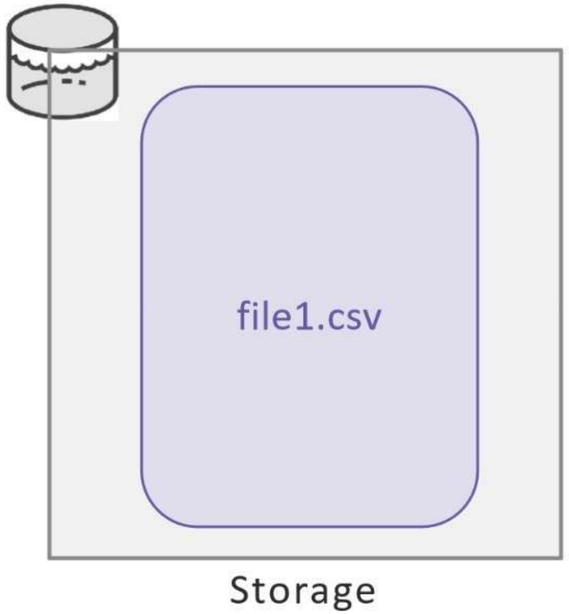
1. Read file1.csv from Storage
2. Apply transformations
3. Write processed data to Storage

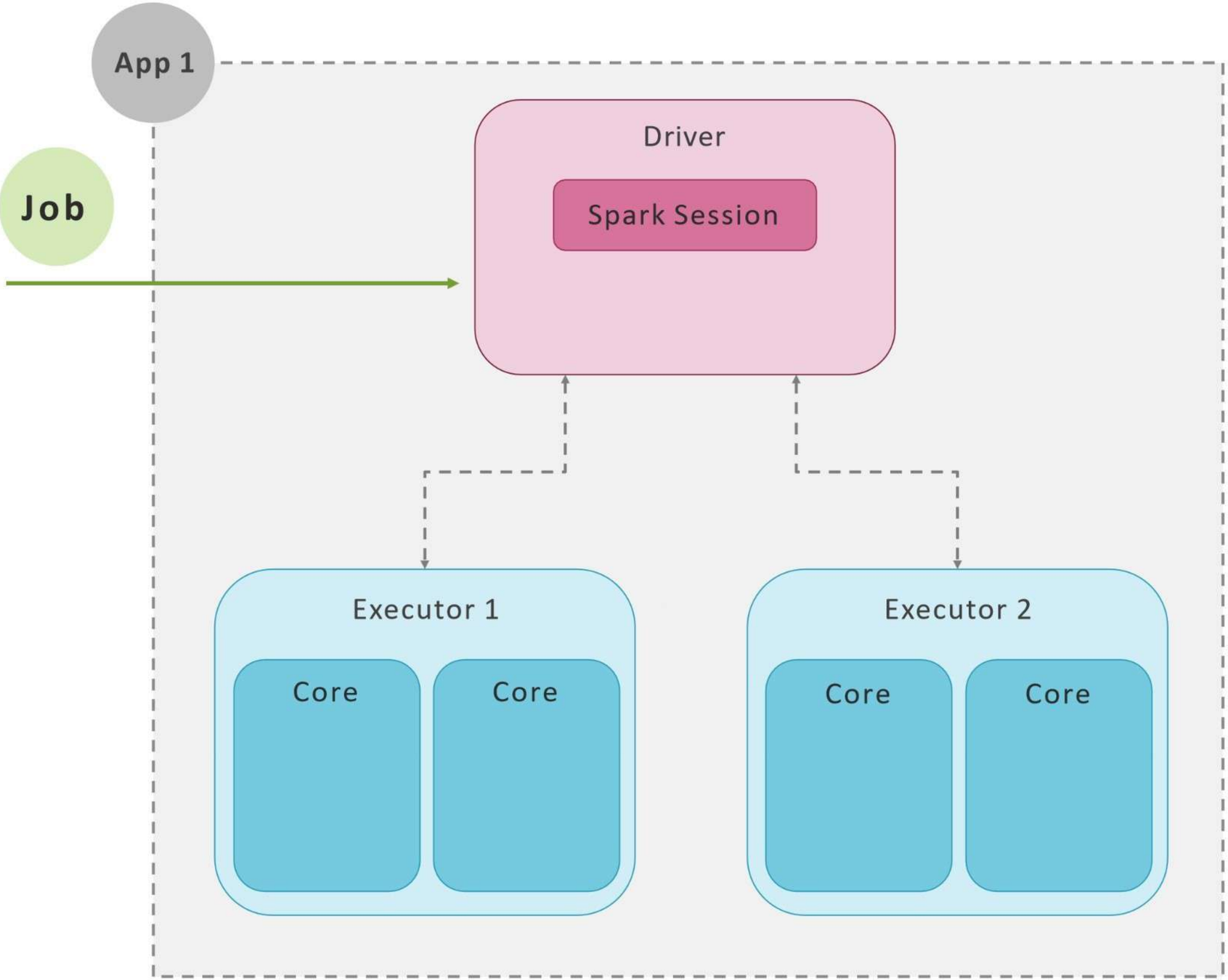




What if number of Tasks are more than available Cores?

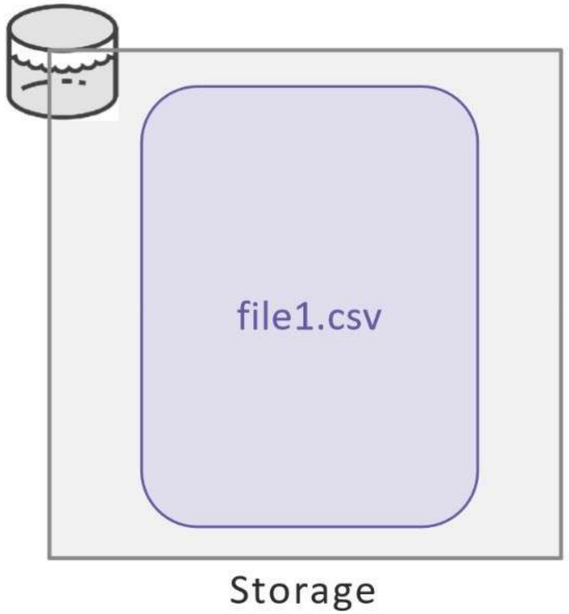
1. Read file1.csv from Storage
2. Apply transformations
3. Write processed data to Storage

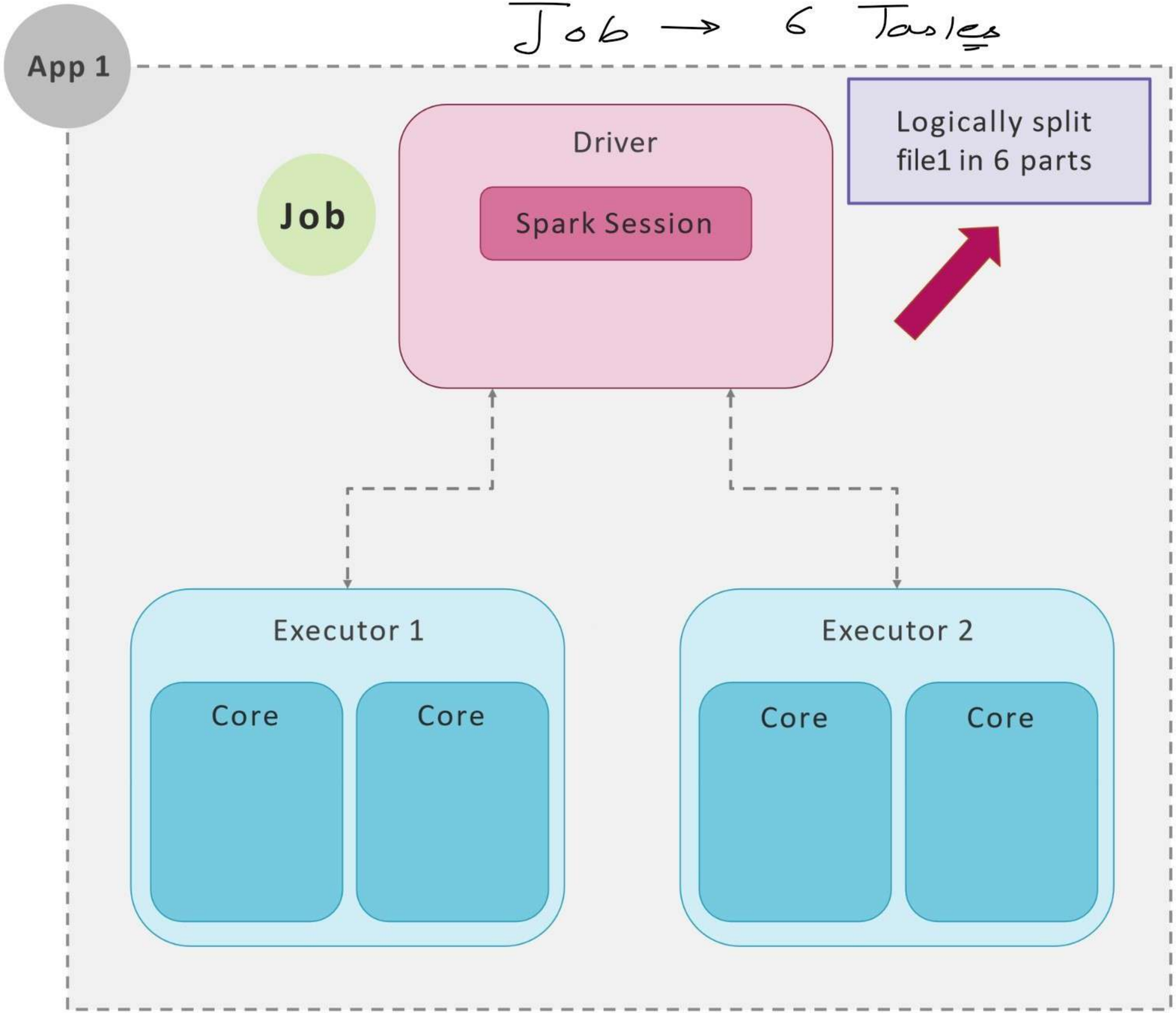




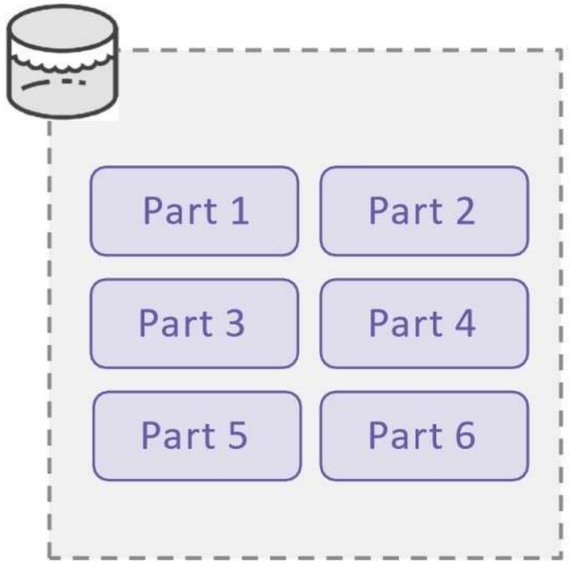
Job → 6 Tasks

1. Read file1.csv from Storage
2. Apply transformations
3. Write processed data to Storage

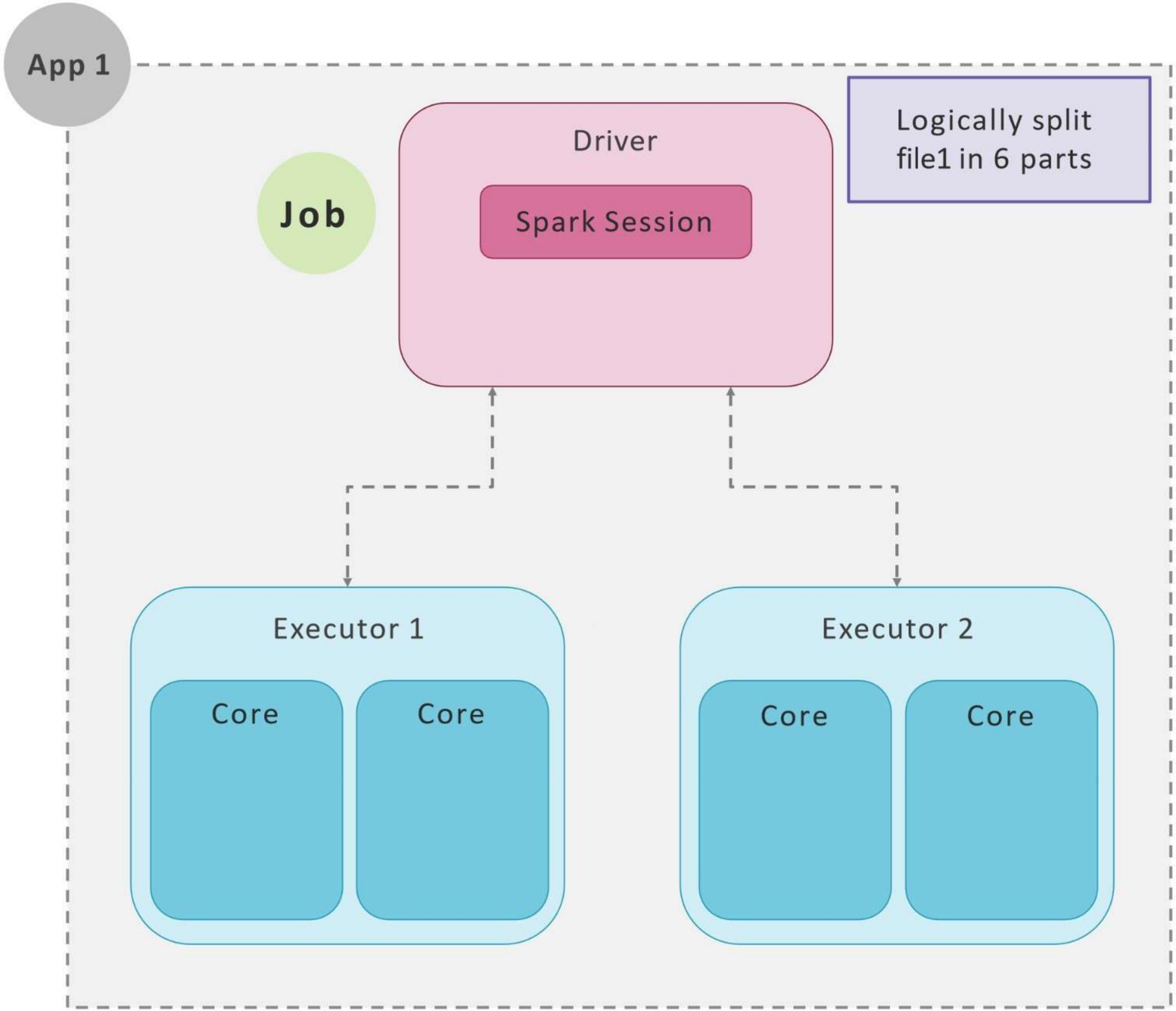




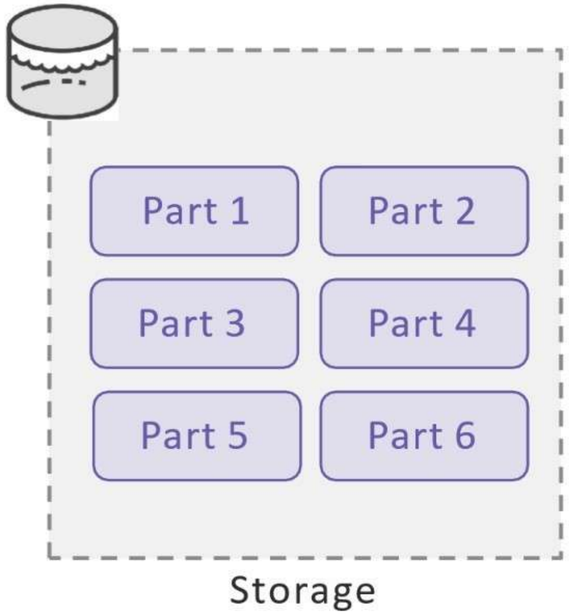
1. Read file1.csv from Storage
2. Apply transformations
3. Write processed data to Storage

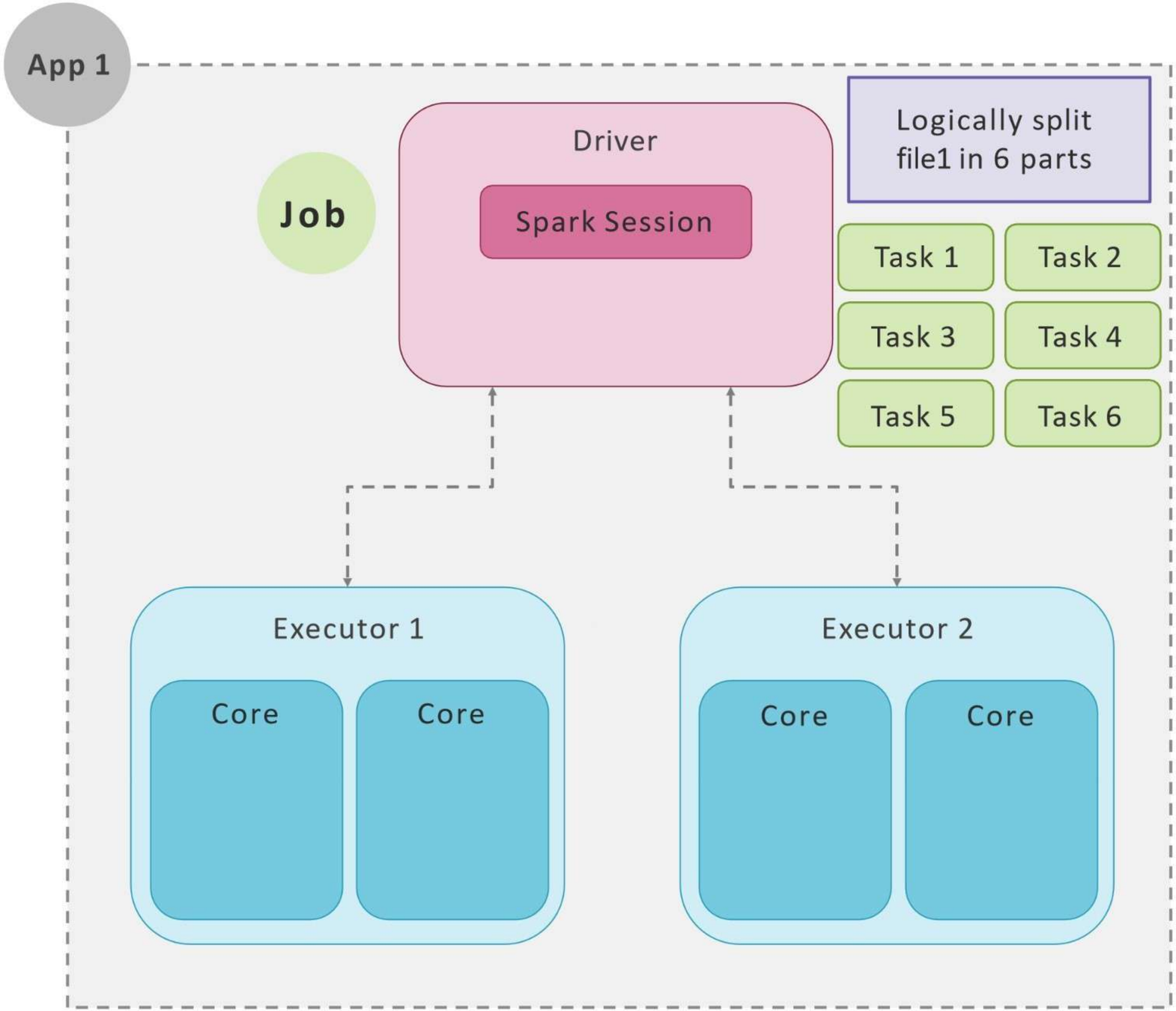


Storage

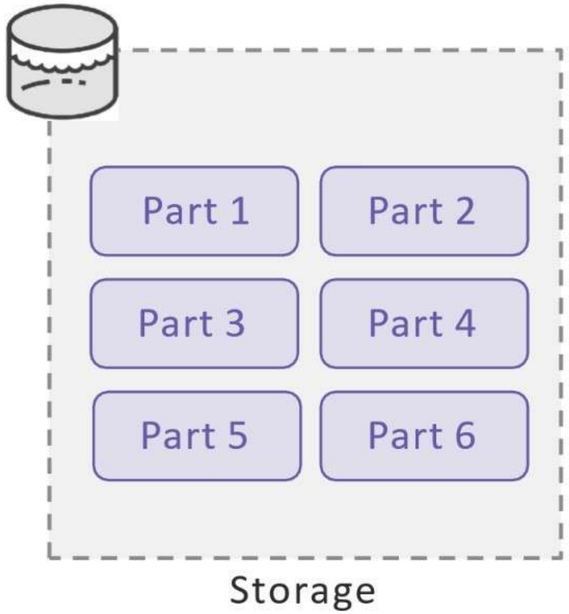


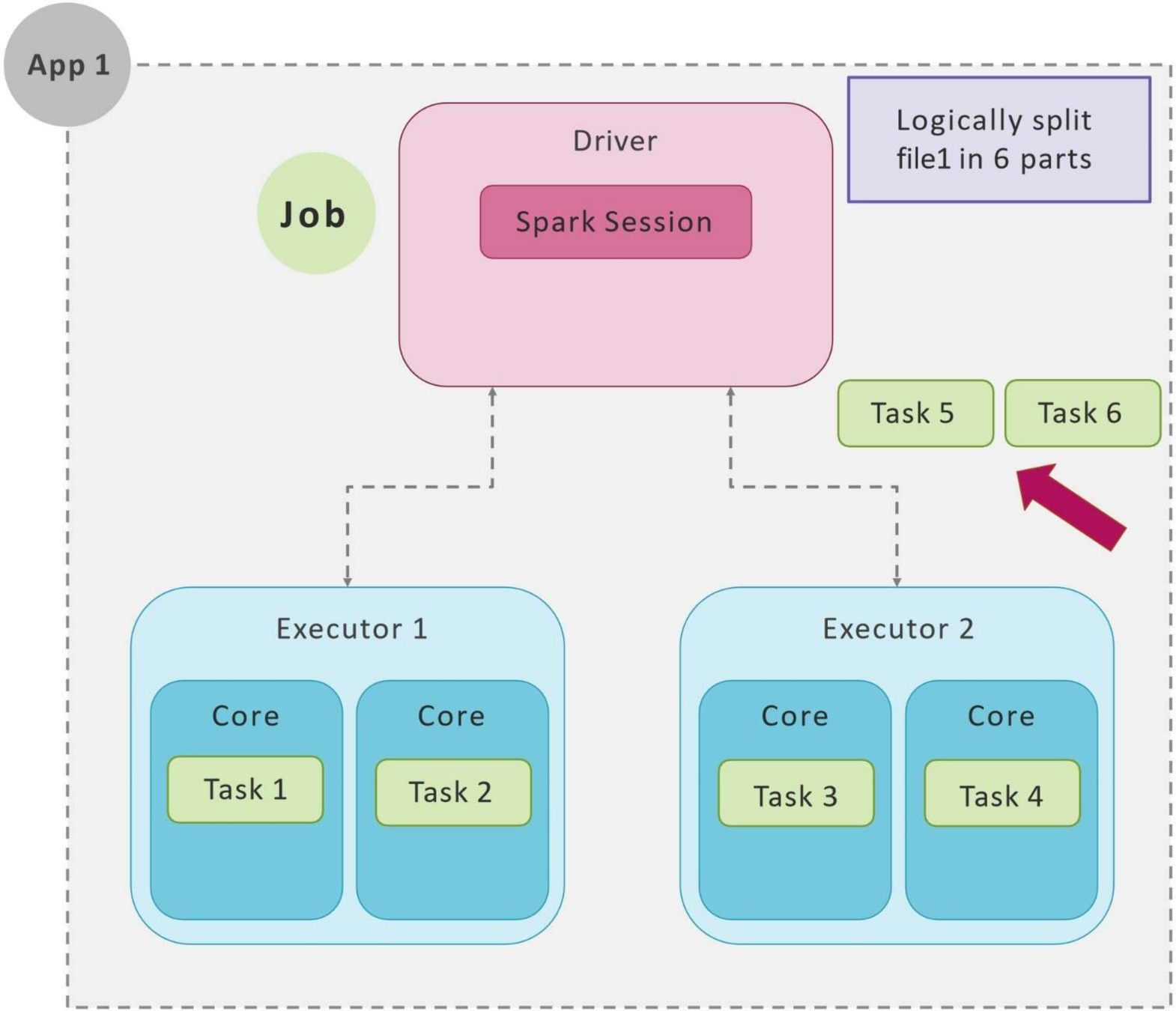
1. Read file1.csv from Storage
2. Apply transformations
3. Write processed data to Storage



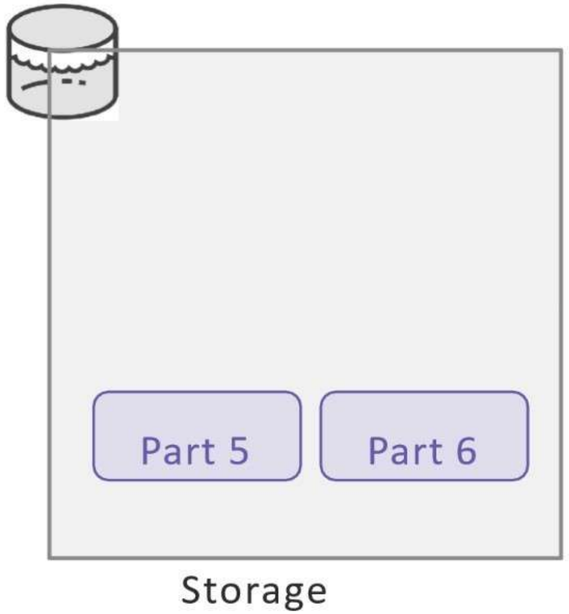


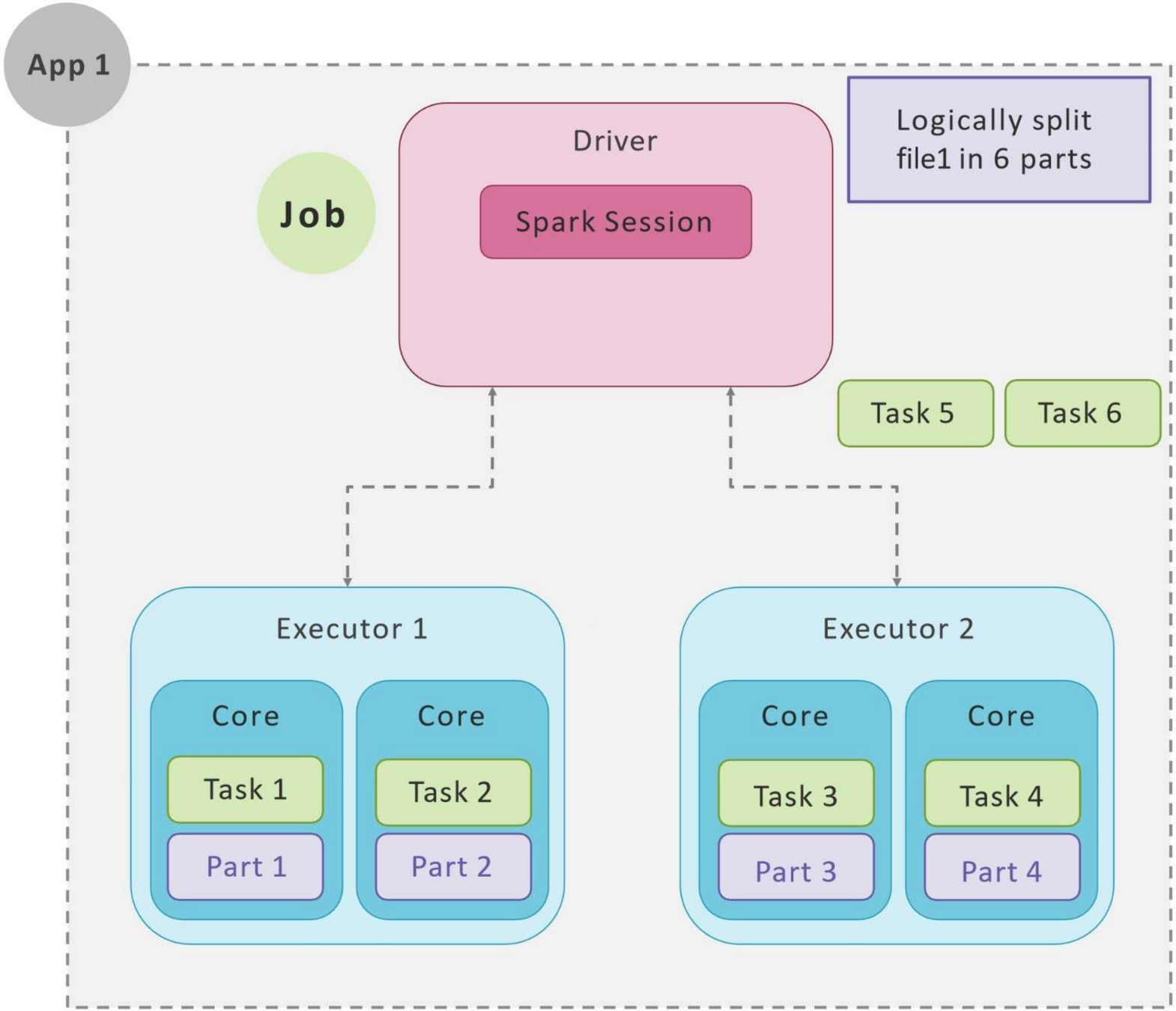
1. Read file1.csv from Storage
2. Apply transformations
3. Write processed data to Storage



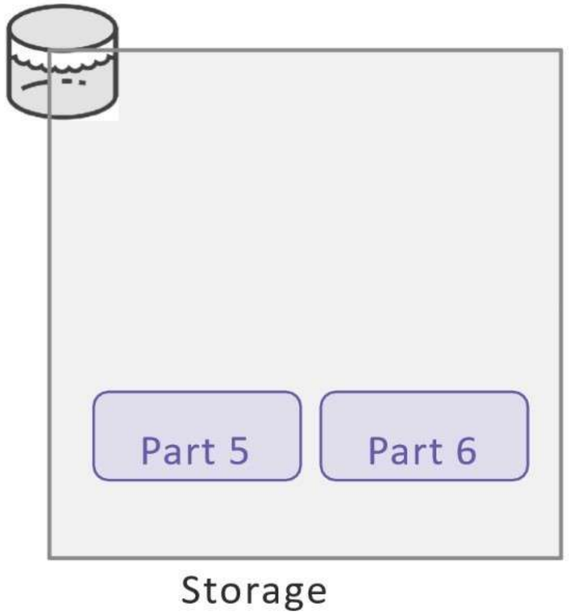


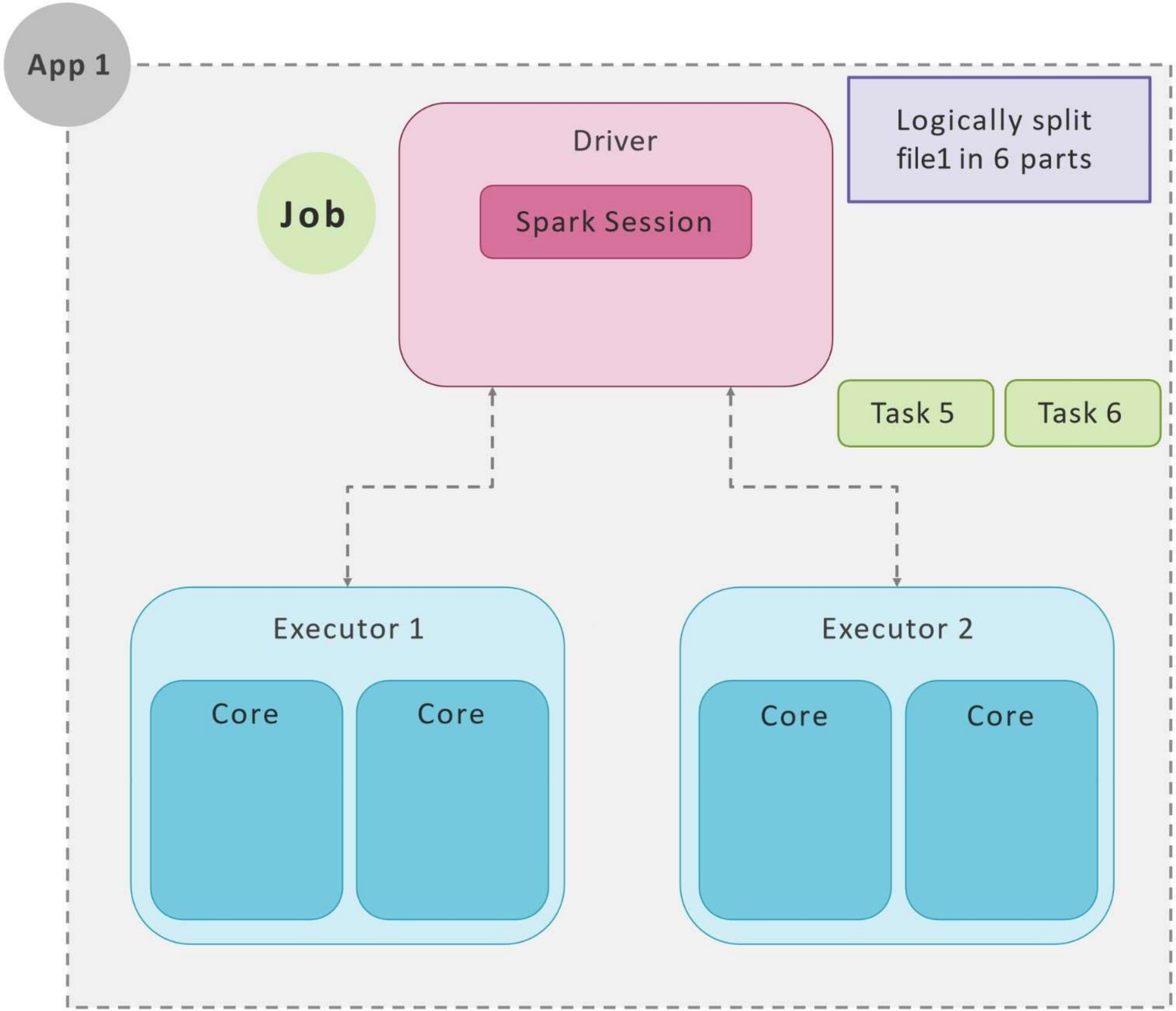
1. Read file1.csv from Storage
2. Apply transformations
3. Write processed data to Storage



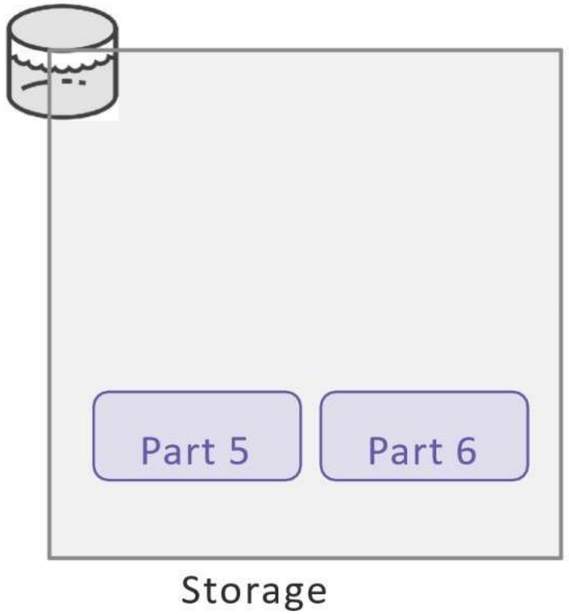


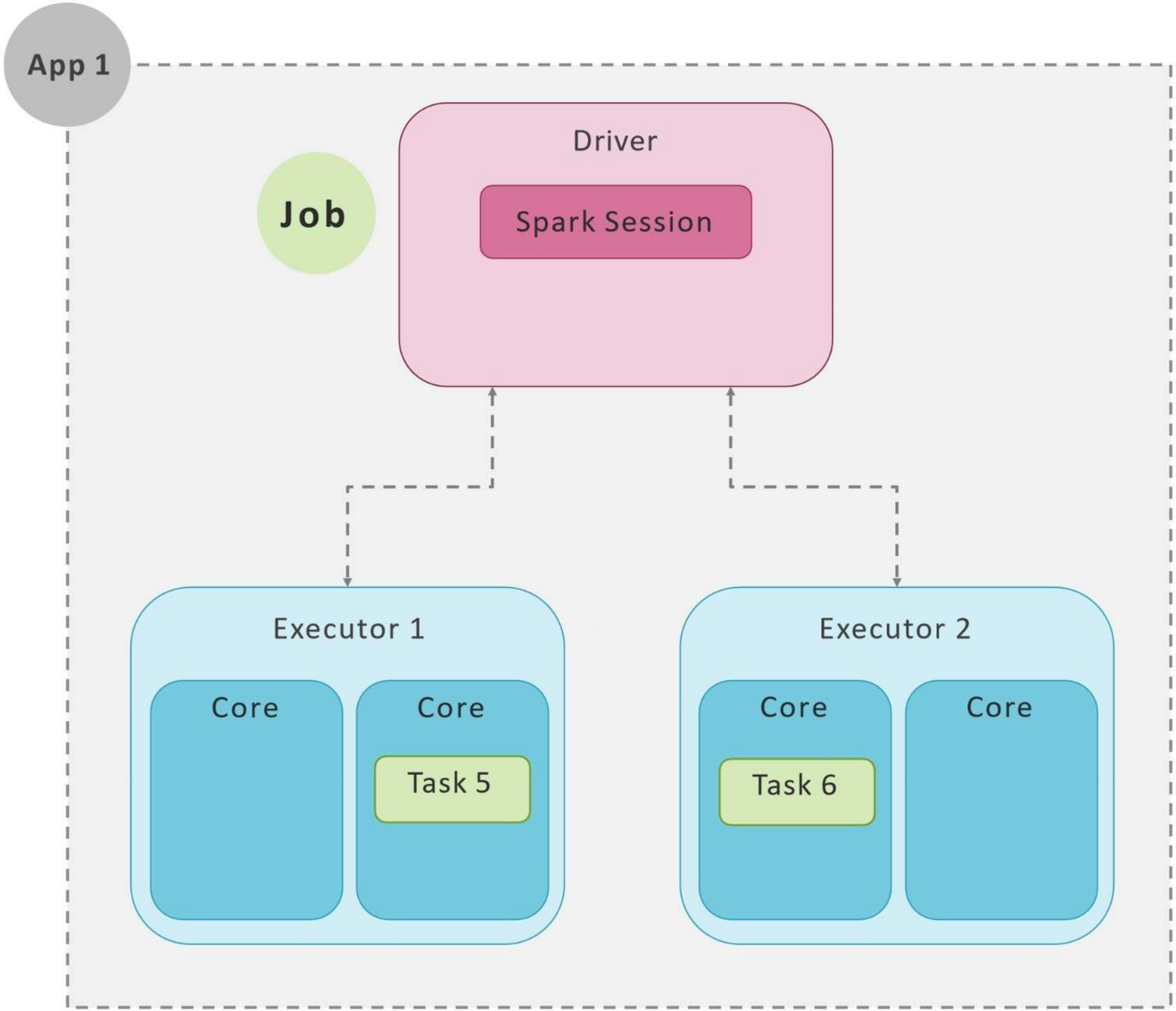
1. Read file1.csv from Storage
2. Apply transformations
3. Write processed data to Storage



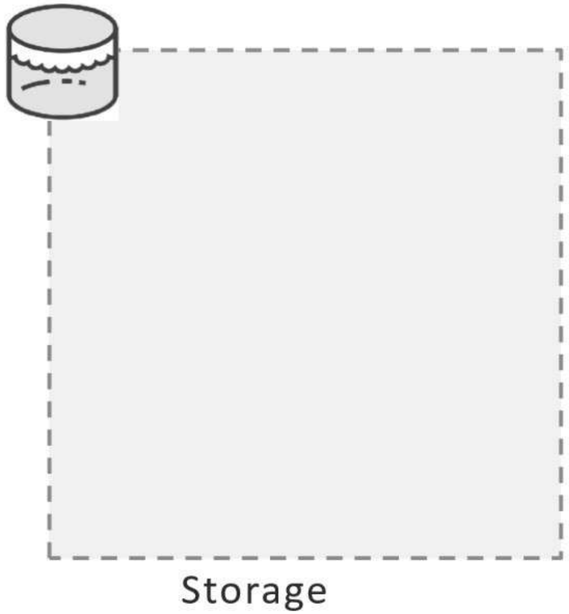


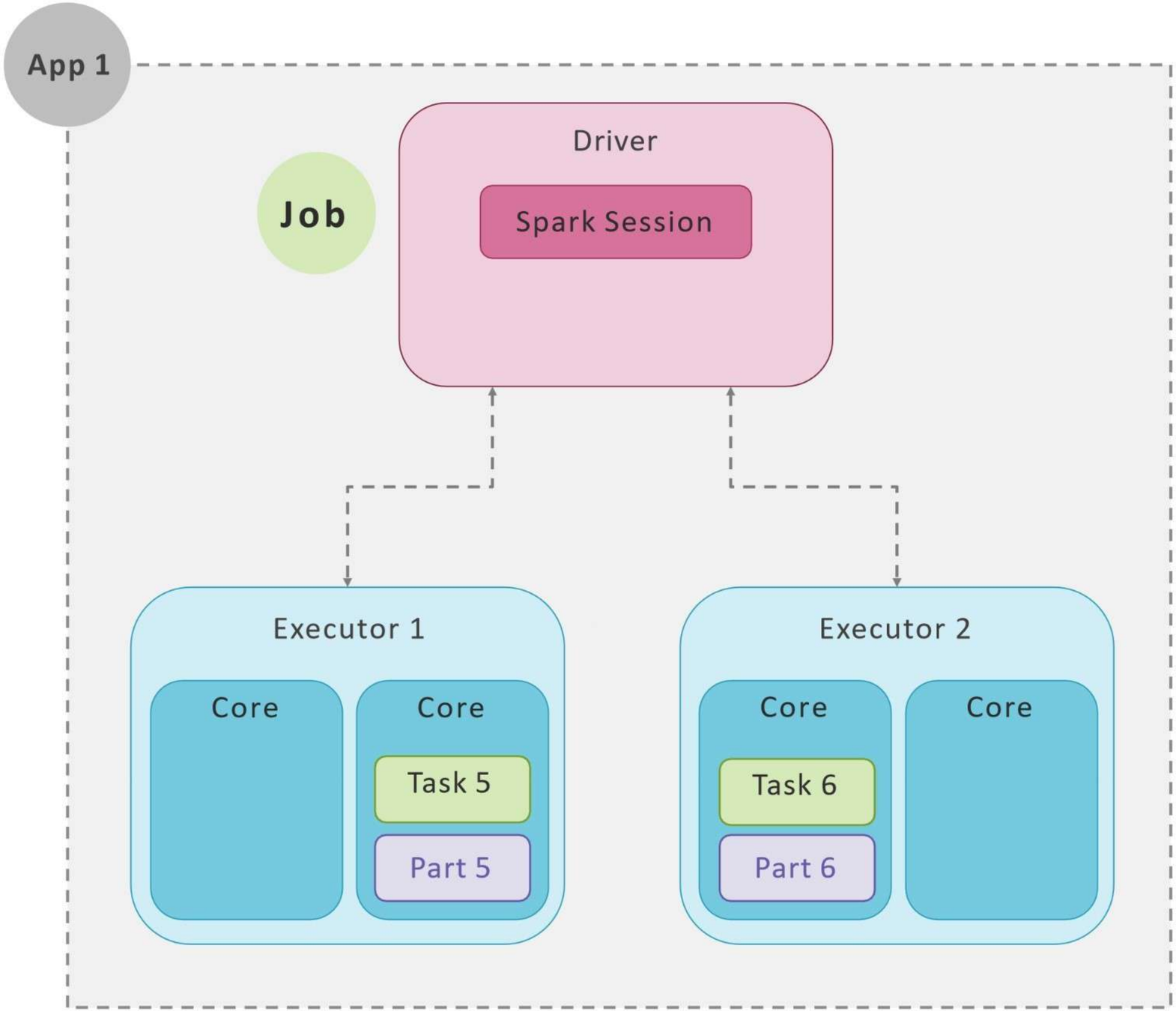
1. Read file1.csv from Storage
2. Apply transformations
3. Write processed data to Storage





1. Read file1.csv from Storage
2. Apply transformations
3. Write processed data to Storage





To improve Parallelization

- Increase number of Executors
- Add more cores to each executor

But provisioning too many resources results in under utilization (wastage)

Spark Application

Job

- Job is created when you need to execute code & take action (getting back results)

Partitions

- A partition is a chunk of data
- Driver decides how many partitions to be created
- Number of tasks = number of partitions
- Each task processes only one partition
- How many partitions to be created? How Spark decides that? Can we control? – we'll learn later!

Cores / Threads / Slots

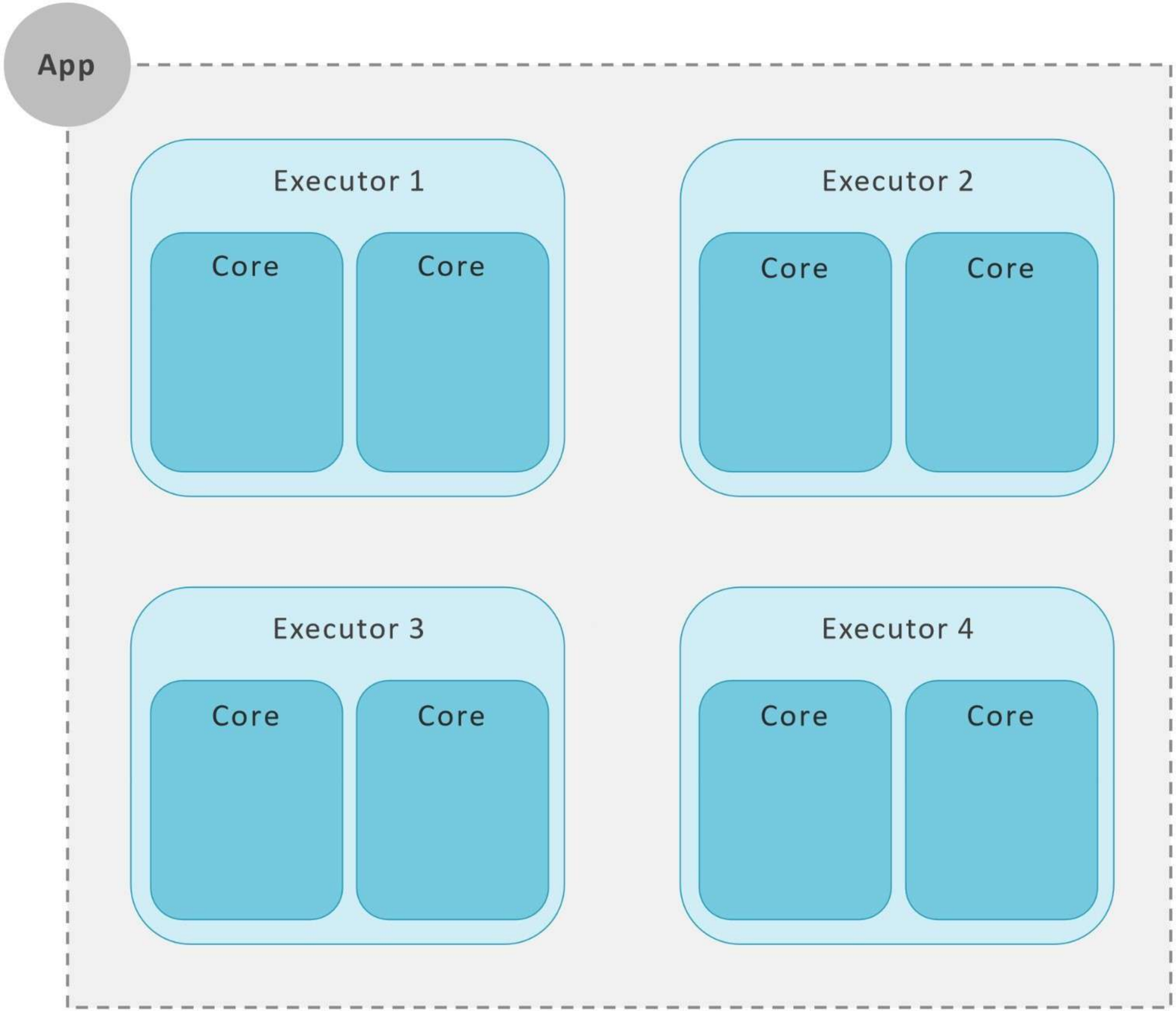
- Each core can execute only one task at a time
- Number of parallel tasks = Number of cores

Executor Size = 2 cores
14 GB RAM

Total Cores = 2 X 4
= 8 cores

Parallel Tasks = Total Cores

→ 8 tasks can execute in parallel



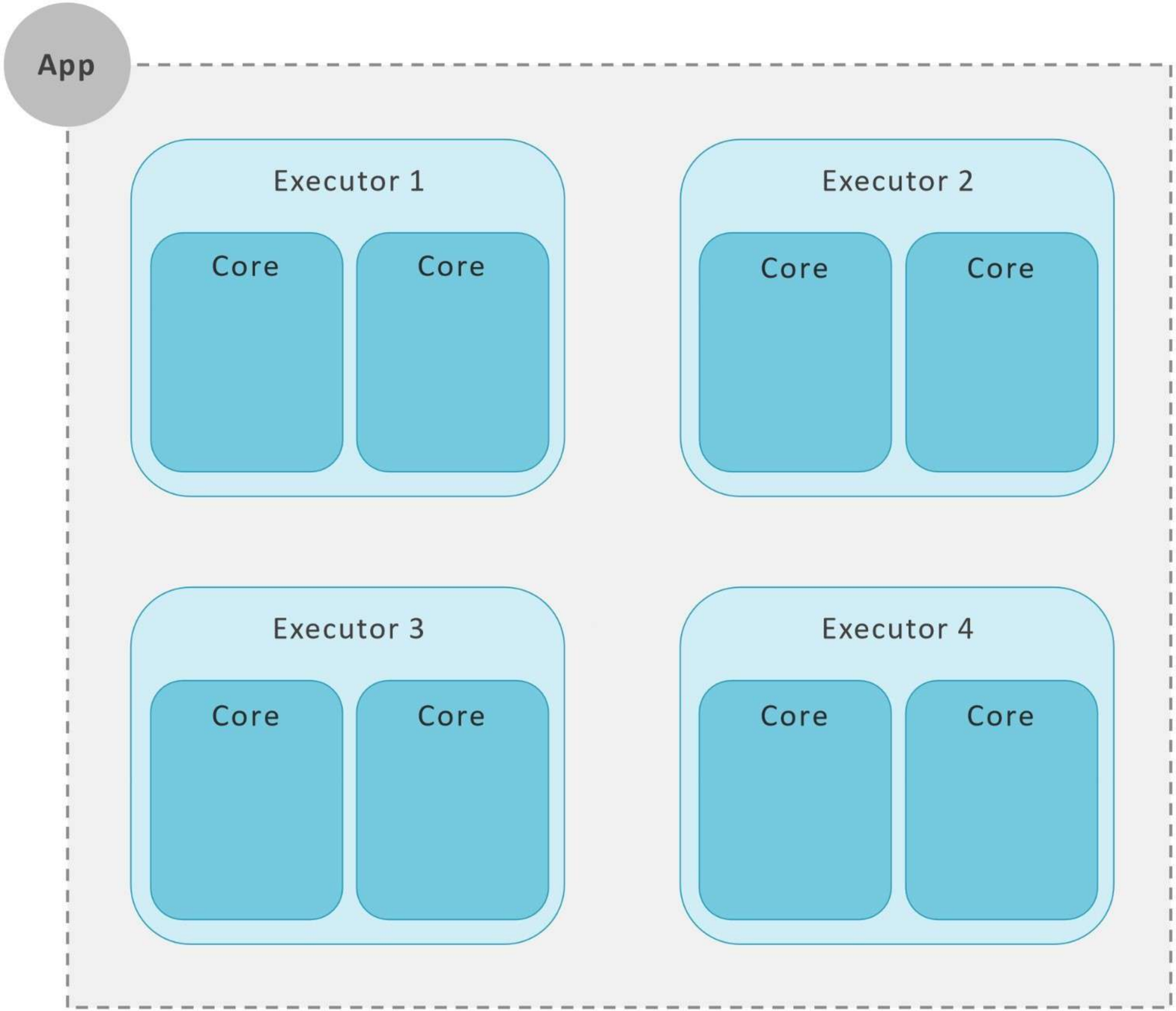
Partitions created by Driver = 16

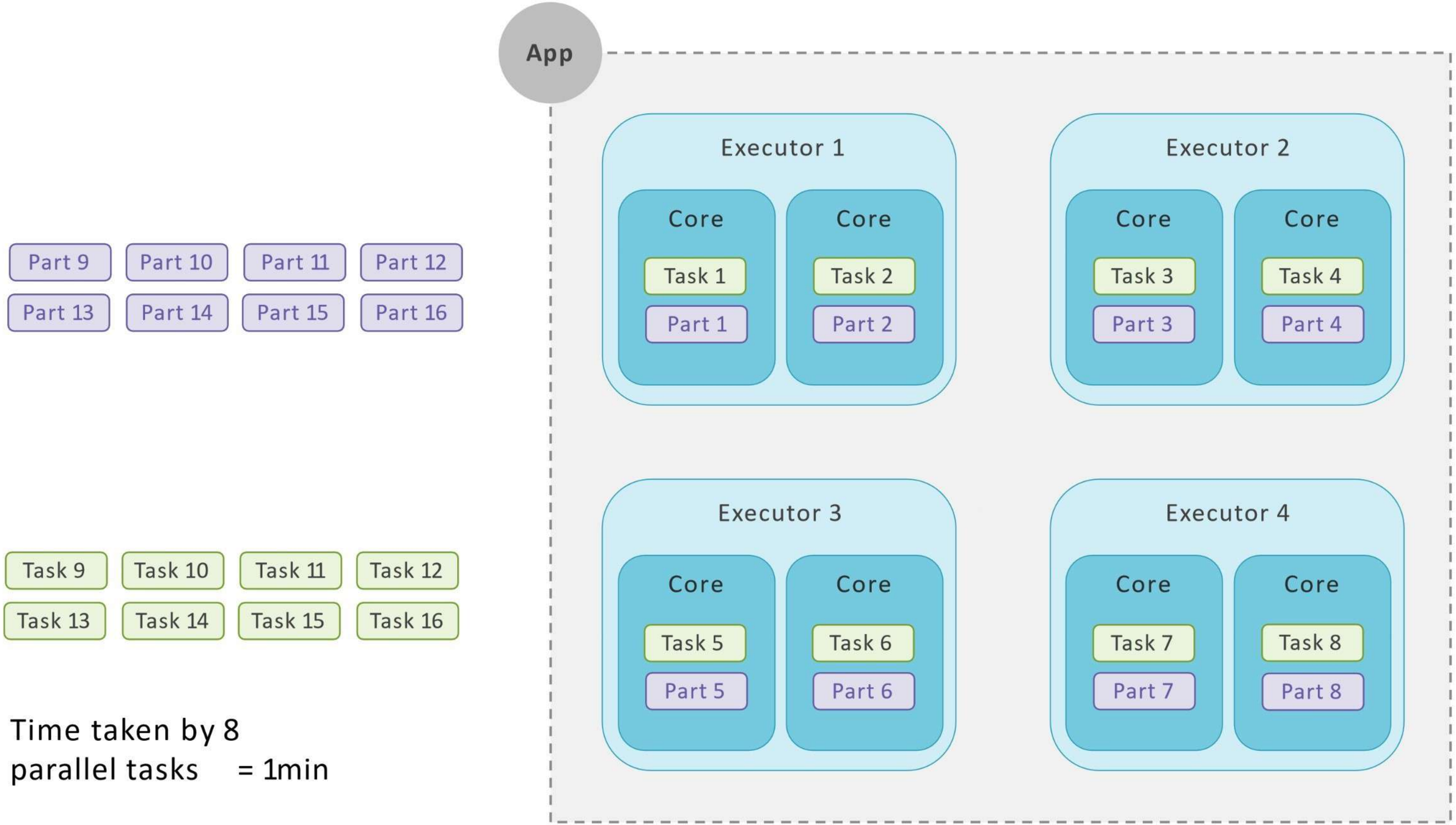
Part 1	Part 2	Part 3	Part 4
Part 5	Part 6	Part 7	Part 8
Part 9	Part 10	Part 11	Part 12
Part 13	Part 14	Part 15	Part 16

Total Tasks = Total Partitions

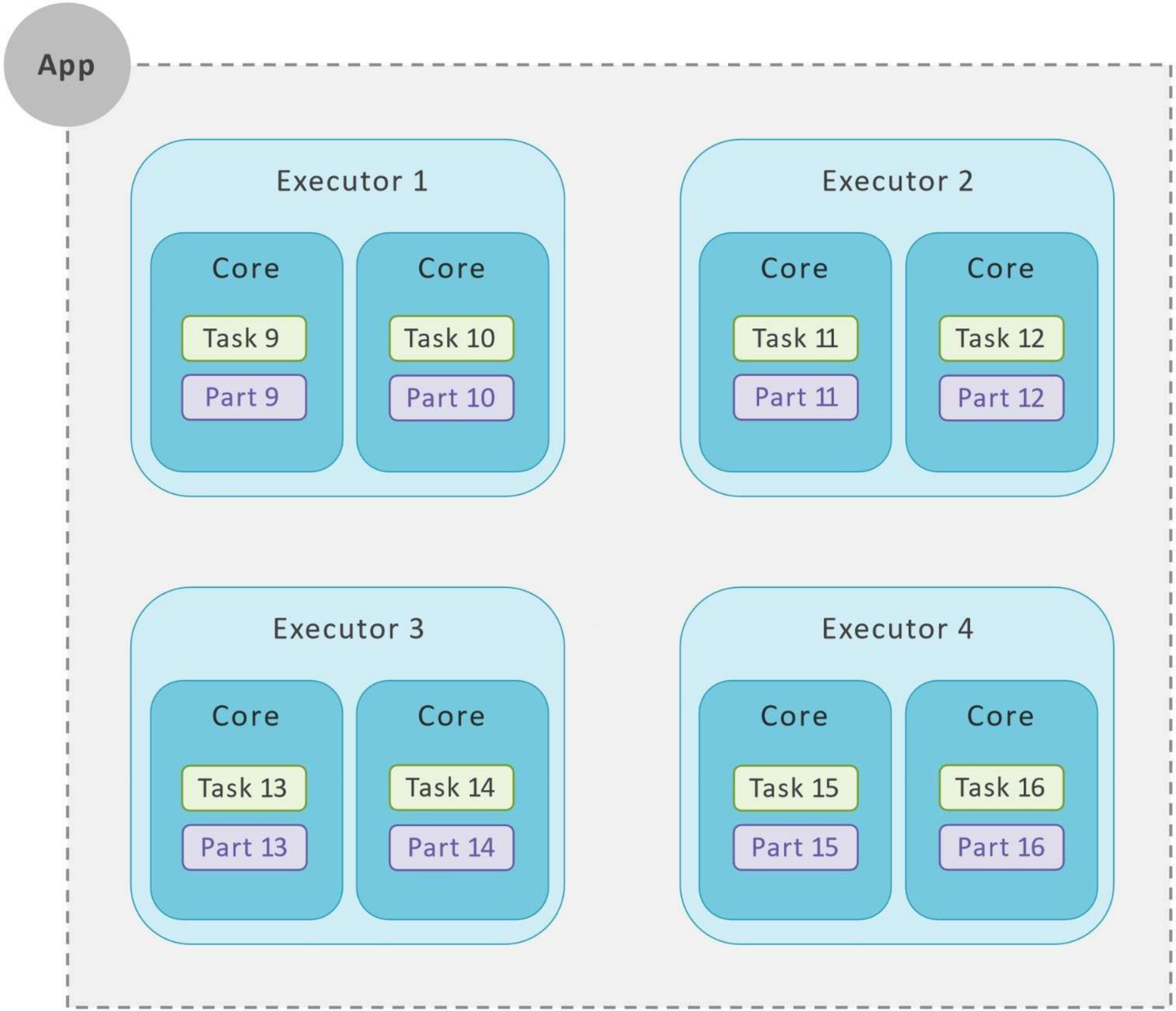
Task 1	Task 2	Task 3	Task 4
Task 5	Task 6	Task 7	Task 8
Task 9	Task 10	Task 11	Task 12
Task 13	Task 14	Task 15	Task 16

Time to complete task = 1min

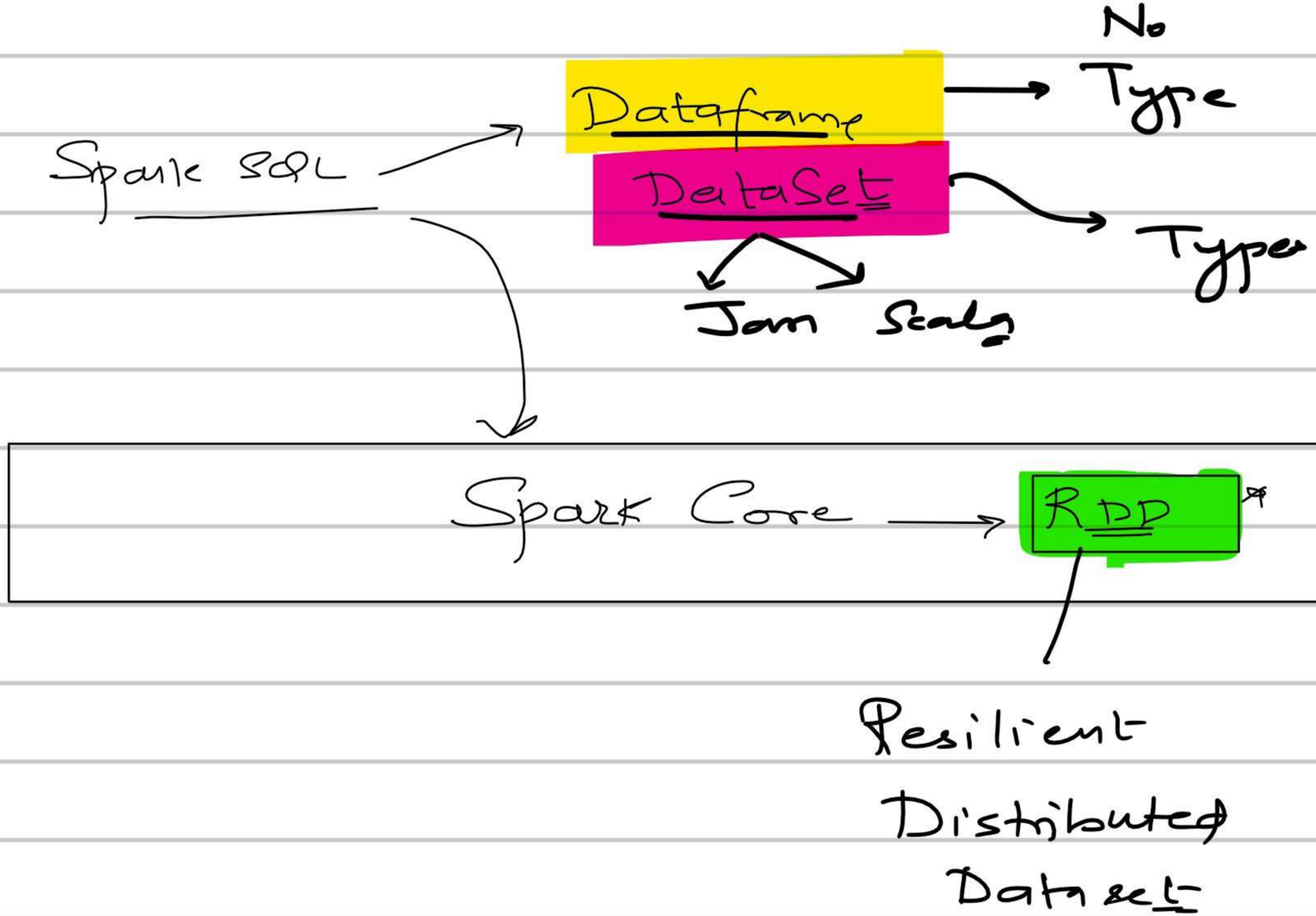



App
Executor 1
Core
Task 1
Part 1
Core
Task 2
Part 2
Executor 2
Core
Task 3
Part 3
Core
Task 4
Part 4
Executor 3
Core
Task 5
Part 5
Core
Task 6
Part 6
Executor 4
Core
Task 7
Part 7
Core
Task 8
Part 8
Part 9
Part 10
Part 11
Part 12
Part 13
Part 14
Part 15
Part 16
Task 9
Task 10
Task 11
Task 12
Task 13
Task 14
Task 15
Task 16
Time taken by 8
parallel tasks = 1min

Total time taken
by 16 tasks = 2 mins



Spark APIs – RDDs, DataFrames & Datasets


Spark SQL
Dataframe
No Type
DataSet
Type
Java
Scala
Spark Core
RDD
Resilient
Distributed
Dataset

Spark APIs

Resilient Distributed Datasets (RDDs)

DataFrames

Datasets

Resilient Distributed Datasets

Introduced with inception of Spark

Native data structure (Spark Core library)

RDD represents collection of data in memory

- Ex - When file is loaded in memory, it is called RDD

All processing in Spark happens on RDDs

Write code using low-level RDD APIs

- APIs load data in memory as RDDs & process them
- All languages supported – Scala, Java, R & Python

Spark does not apply any optimization to RDD code

Higher-Level Spark APIs

DataFrames

Introduced with Spark 1.3

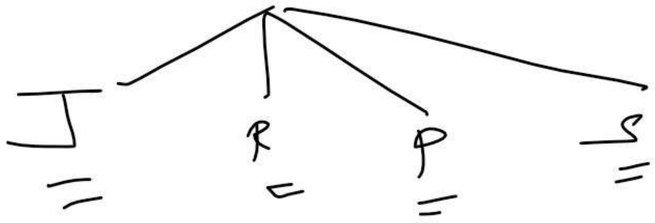
Based on RDDs

Collection of data, but in tabular format

No compile-time safety

Spark applies optimizations to code

Supported in all languages



Datasets

Introduced with Spark 1.6

Based on RDDs

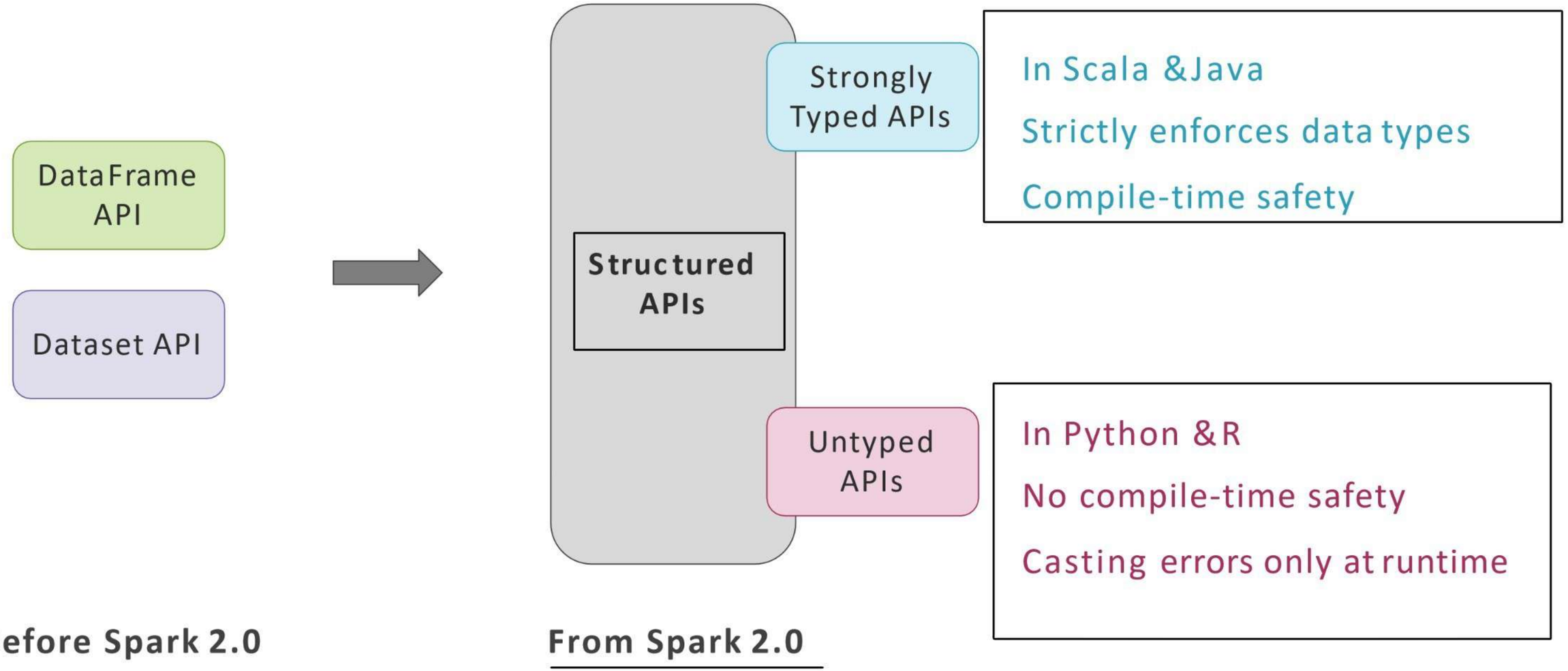
Combination of RDDs and DataFrames

Provides compile-time safety

Spark applies optimizations to code

Supported in Java &Scala

Higher-Level Spark APIs



Since we are going to use PySpark, we'll work with Untyped APIs (generally referred as DataFrames)

Summary

Background of Apache Spark

- Spark fits the bill for modern data processing needs
- Overcomes challenges faced by Hadoop

In-memory analytics engine that runs on a cluster

- Spark Core & Engine performs distributed processing
- Built-in libraries for various uses cases – batch processing, streaming, ML, graph computations
- Uses cluster manager & HDFS

Distributed execution architecture

- Spark application creates Driver & Executors
- Driver creates a Spark Session
- Cluster Manager handles allocation of resources
- Data is divided into Partitions
- Task is allocated to a core to process a data partition

Supports 3 APIs – RDDs, DataFrames & Datasets

Setting up Spark Environment

Understanding Spark Environments

Spark Development Options

Command line / Spark Shell

Notebooks

- Jupyter, Zeppelin
- Can run locally or in cloud platforms like Azure HDInsight & Google Dataproc

IDEs to build projects

- IntelliJ, PyCharm, VS Code
- Run code locally, or build & run in Spark cluster

Spark Submit command to run jobs

Cloud platform notebooks

- Offered by Databricks, Azure Synapse Analytics etc.

Cluster Managers

Manages & allocates resources to applications on the cluster

- Local
- Standalone
- YARN
- Mesos
- Kubernetes

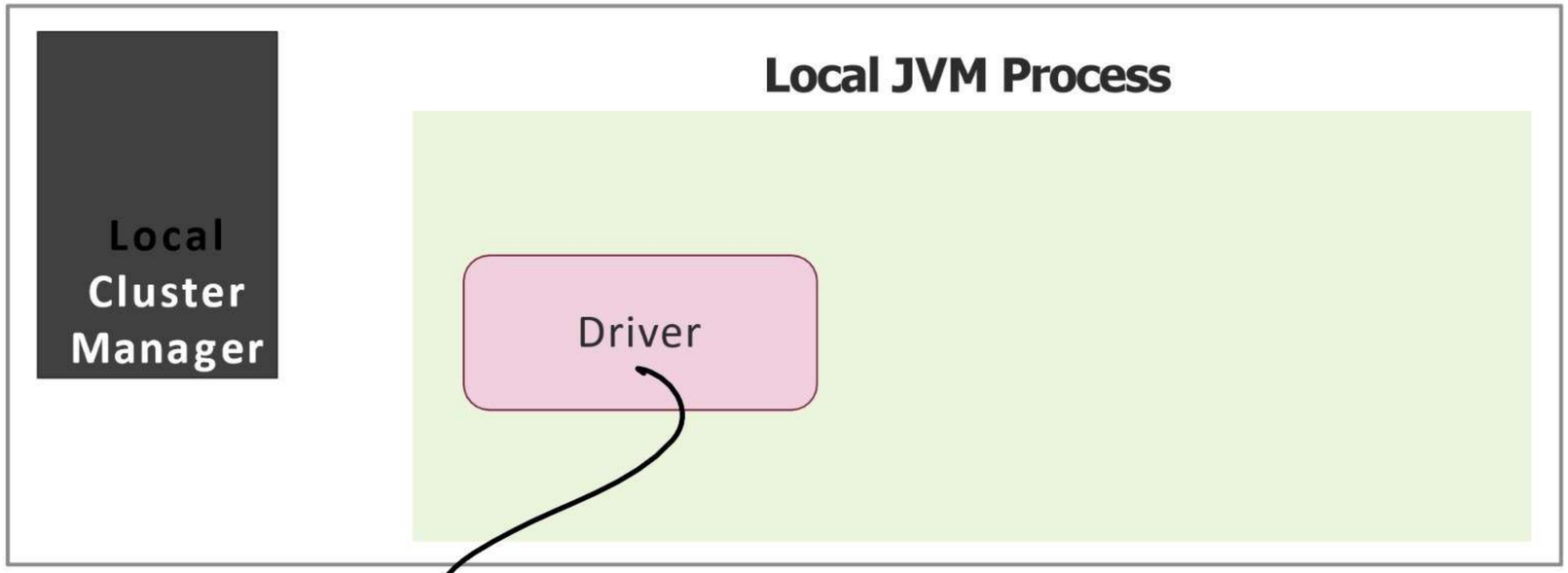
Application Execution Modes

Local Mode

Client Mode

Cluster Mode

Local Execution Mode



Driver = Executor

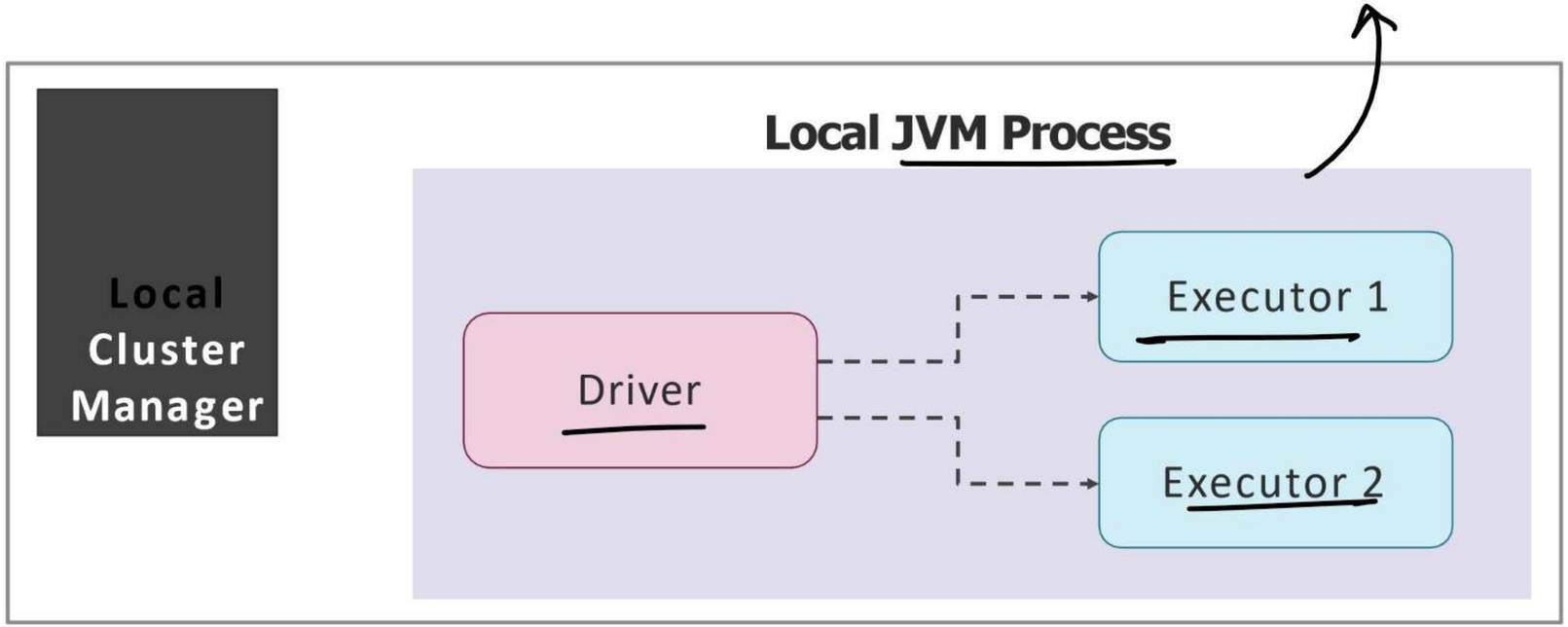
Single thread is used

Driver runs all the jobs

No executors are created

Local Execution Mode

Core



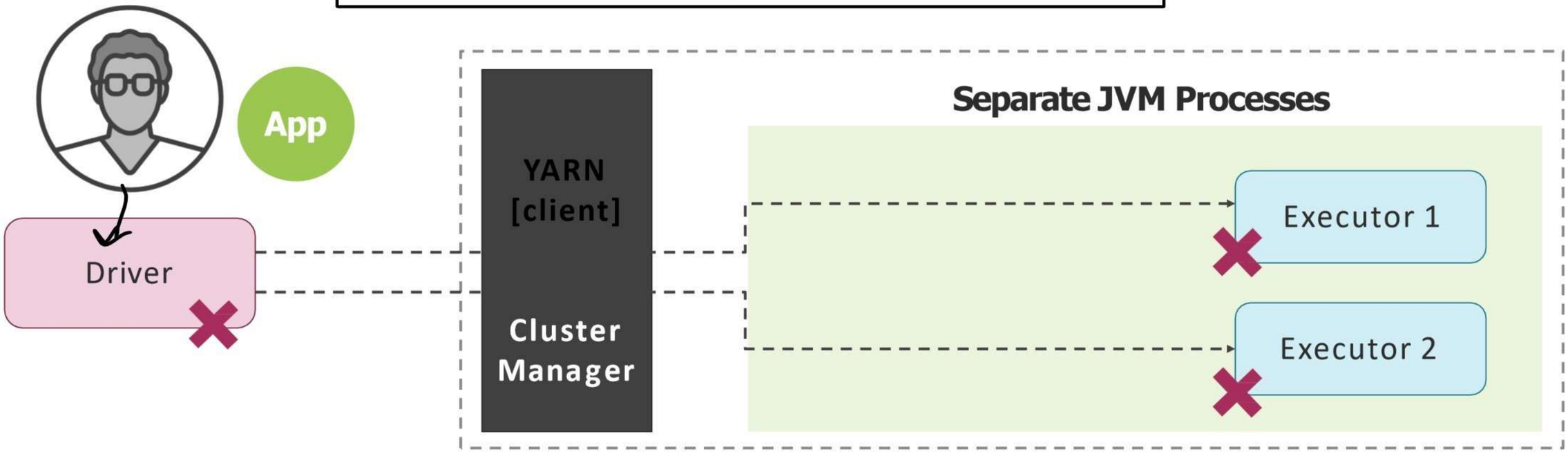
Three threads are created in same JVM

Executors run tasks

Simulation of production cluster environment

Execution Mode	How it works?	Cluster Managers	Development Tools	Purpose
Local	Driver & executors on same JVM	Local	Spark Shell Notebooks IDEs	Run locally for dev/testing

Client Execution Mode

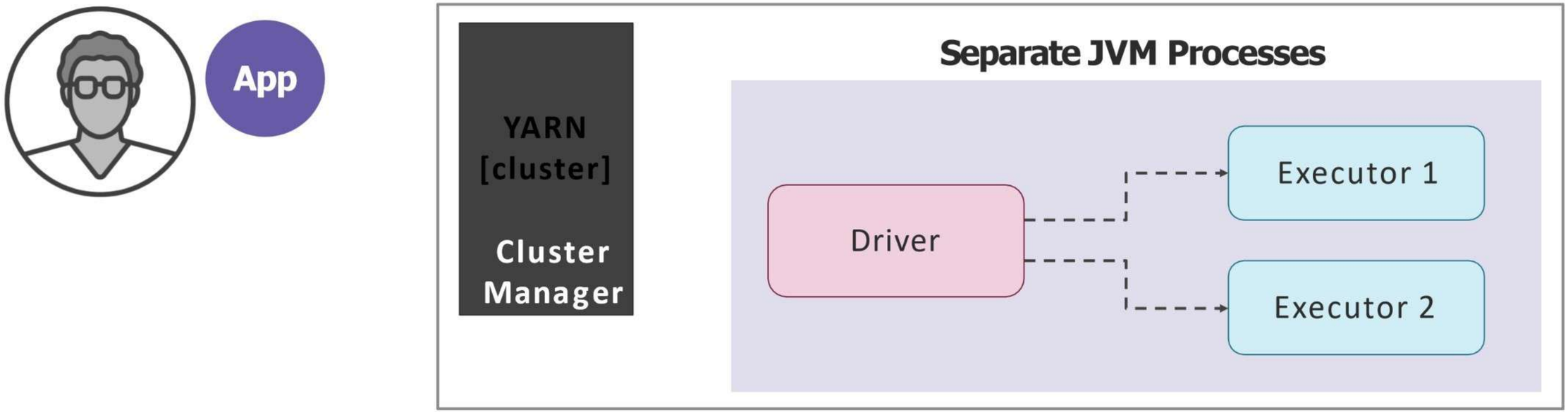


Driver runs on client machine

Each executor runs as a separate JVM process in cluster

Execution Mode	How it works?	Cluster Managers	Development Tools	Purpose
Local	Driver &executors on same JVM	Local	Spark Shell Notebooks IDEs	Run locally for dev/testing
Client	Driver on client, Executors on cluster	YARN [client] Standalone	Spark Shell Notebooks Spark-submit	Interactive querying, Data analysis

Cluster Execution Mode



Driver & executors run on cluster

Each process consumes resources on cluster

Used to run production workloads

Execution Mode	How it works?	Cluster Managers	Development Tools	Purpose
Local	Driver &executors on same JVM	Local	Spark Shell Notebooks IDEs	Run locally for dev/testing
Client	Driver on client, Executors on cluster	YARN [client] Standalone	Spark Shell Notebooks Spark-submit	Interactive querying, Data analysis
Cluster	Driver &executors in separate JVMs on cluster	YARN [cluster] Standalone Kubernetes Mesos	Spark-submit Cloud notebooks	Long running jobs in production

Let’s Install Spark

Spark 3+ Minimum Requirements

Windows, Linux, MacOS

Java v8 (8u92+) / v11 / v17

Scala 2.12 and above 2.13

Python 3.6 and above

Unzip utility like 7-Zip

Winutils.exe *(for Windows only)*

hadoop.dll

Option 4:
Running Jobs with Spark Submit

Anaconda Navigator

File Help

ANACONDA.NAVIGATOR

Connect

Home

Environments

Learning

Community

Documentation

Anaconda Blog

Search Environments

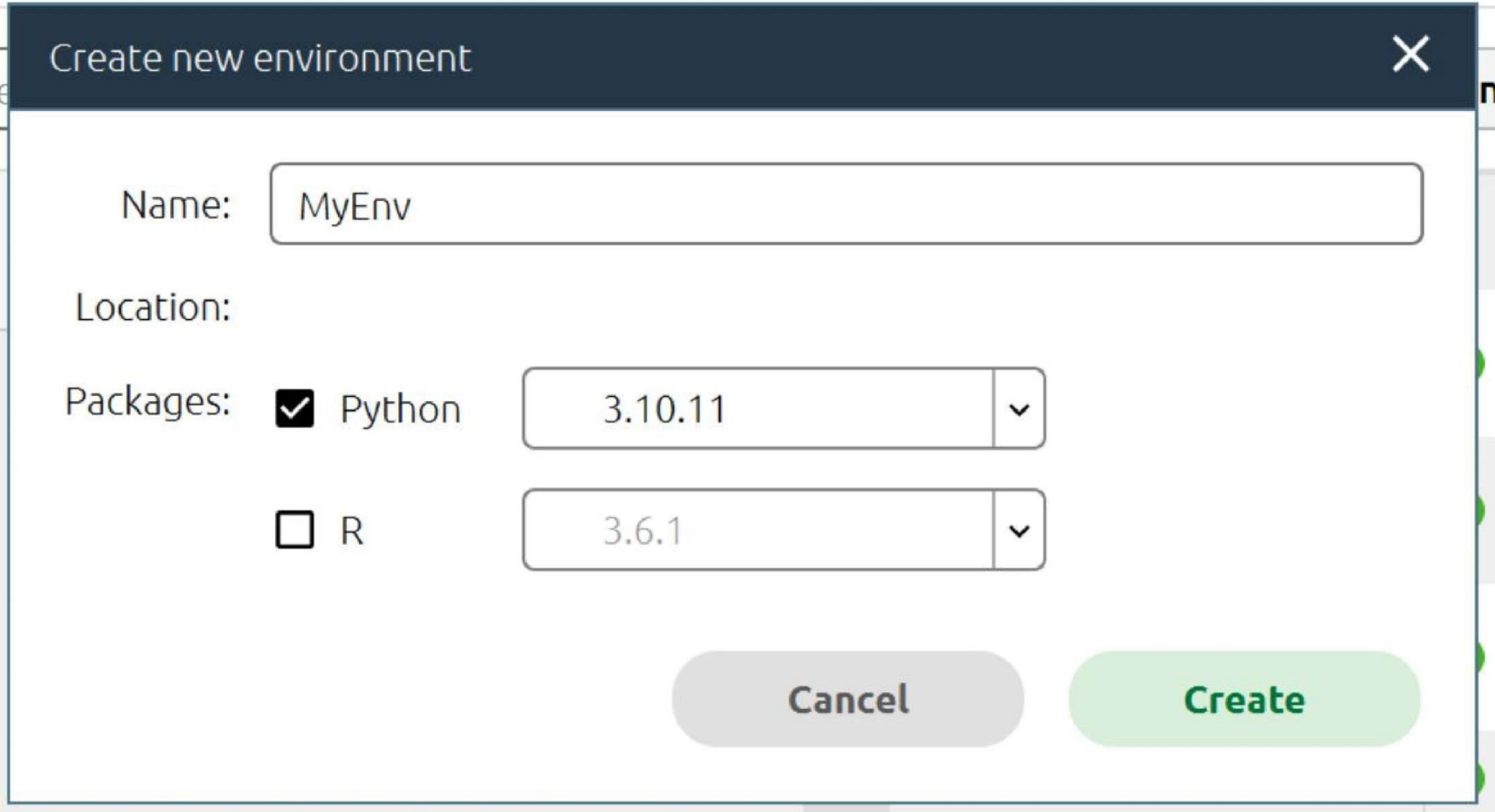
base (root)

Create Clone Import Backup Remove

Installed Channels Update index... Search Pac

Name	T	Description	Ver
alabaster		Configurable, python 2+3 compatible sphinx theme.	
anaconda-client		Anaconda.org command line client library	
anaconda-project		Tool for encapsulating, running, and reproducing data science projects	
anyio		High level compatibility layer for ...	
appdirs		A small python module for determining appropriate platform-specific dirs.	
argon2-cffi		The secure argon2 password hashing algorithm	

446 packages available


Create new environment
Name:
MyEnv
Location:
Packages:
Python
3.10.11
R
3.6.1
Cancel
Create

Anaconda Navigator

File Help

ANACONDA.NAVIGATOR

Connect

Home

Environments

Learning

Community

Documentation

Anaconda Blog

Search Environments

base (root)

MyEnv

Open Terminal

Open with Python

Open with IPython

Open with Jupyter Notebook

Create Clone Import Backup Remove

Installed Channels Update index... Search Pac

Name	T	Description	Ver
		High-quality data compressor	
		Certificates for use with other packages.	
		A portable foreign function interface library	
openssl		Openssl is an open-source implem…	
pip		Pypa recommended tool for installing python packages	
python		General purpose programming language	

15 packages available

ENG 06:38 AM

C:\windows\system32\cmd.e›

```
(MyEnv) C:\Users\VICTUS>conda install -c conda-forge findspark pyhive delta-spark jupyter
```

Spark-Submit

Utility to submit a Spark Application / Job to a cluster

Code can be in any language

- Scala, Python or Java

Can be submitted to any supported Cluster Manager

- Local, Standalone, YARN, Mesos, Kubernetes

Provide configuration and dependencies

Both Spark Shell & Spark-Submit are command line

- Spark Shell is typically used for interactive querying
- Spark-Submit is typically used to submit long running jobs

```
=========================================================
Steps to create multi-node cluster on local machine
=========================================================

A. In Command Prompt 1 - Spark Master

A.1. Create Spark master
      => spark-class org.apache.spark.deploy.master.Master --host <machine name or localhost>

      - Example => spark-class org.apache.spark.deploy.master.Master --host localhost

A.2. Copy Master URL and Spark UI URL
      - Typically master URL is at port 7077
      - Typically Spark UI is at port 4040
      - If any of these ports are in use, a different port might be used

-------------------------------------------------------------------------------------------------------
```

```
-----------------------------------------------------------------------------------------------------------

B. In Command Prompt 2 - Worker Node 1

B.1. Create worker node 1
      => spark-class org.apache.spark.deploy.worker.Worker <master URL> --host <machine name or localhost>

      - Example => spark-class org.apache.spark.deploy.worker.Worker spark://localhost:7077 --host localhost

-----------------------------------------------------------------------------------------------------------

C. In Command Prompt 3 - Worker Node 2

C.1. Create worker node 2
      => spark-class org.apache.spark.deploy.worker.Worker <master URL> --host <machine name or localhost>

      - Example => spark-class org.apache.spark.deploy.worker.Worker spark://localhost:7077 --host localhost

-----------------------------------------------------------------------------------------------------------
```

Summary

Can work with Spark in various ways

- Several development options and cluster managers

Understood application execution modes

- Local mode – Single JVM to run driver & executors
- Client mode – Driver on client & executors on cluster
- Cluster mode – Driver & executors run on cluster

Spark can be installed on local machine or on cluster

Development Options

- Command Line can be used for dev/testing
- Jupyter for interactive analysis or development
- PyCharm IDE for project development
- Spark-submit to run long running production jobs

Web UI can be used for monitoring

Working with RDDs – Resilient Distributed Datasets

Understanding RDDs

Resilient Distributed Dataset (RDD) is the native Data Structure of Spark

Resilient Distributed Datasets

Id, City, Amount
1,Seattle, 600
2, London, 300
3, Delhi, 700
4, Seattle, 400
5, Paris, 900
6, Delhi, 200
7, Seattle, 900

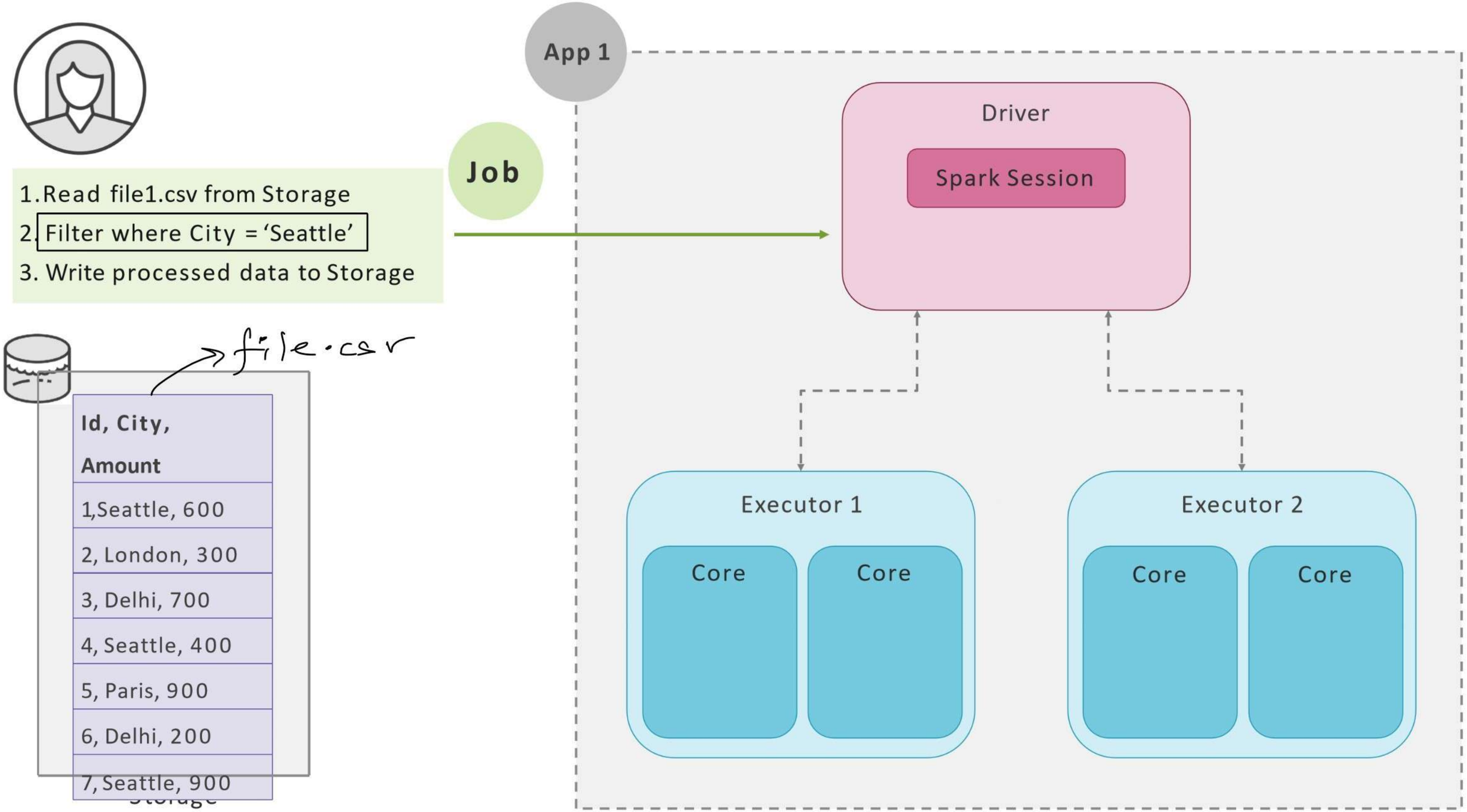
CSV File in Storage

Read File with RDD API

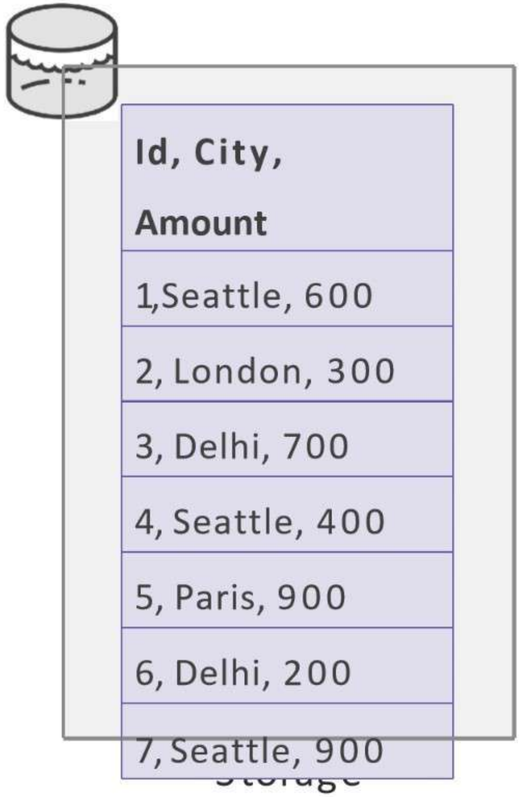
1,Seattle, 600
2, London, 300
3, Delhi, 700
4, Seattle, 400
5, Paris, 900
6, Delhi, 200
7, Seattle, 900

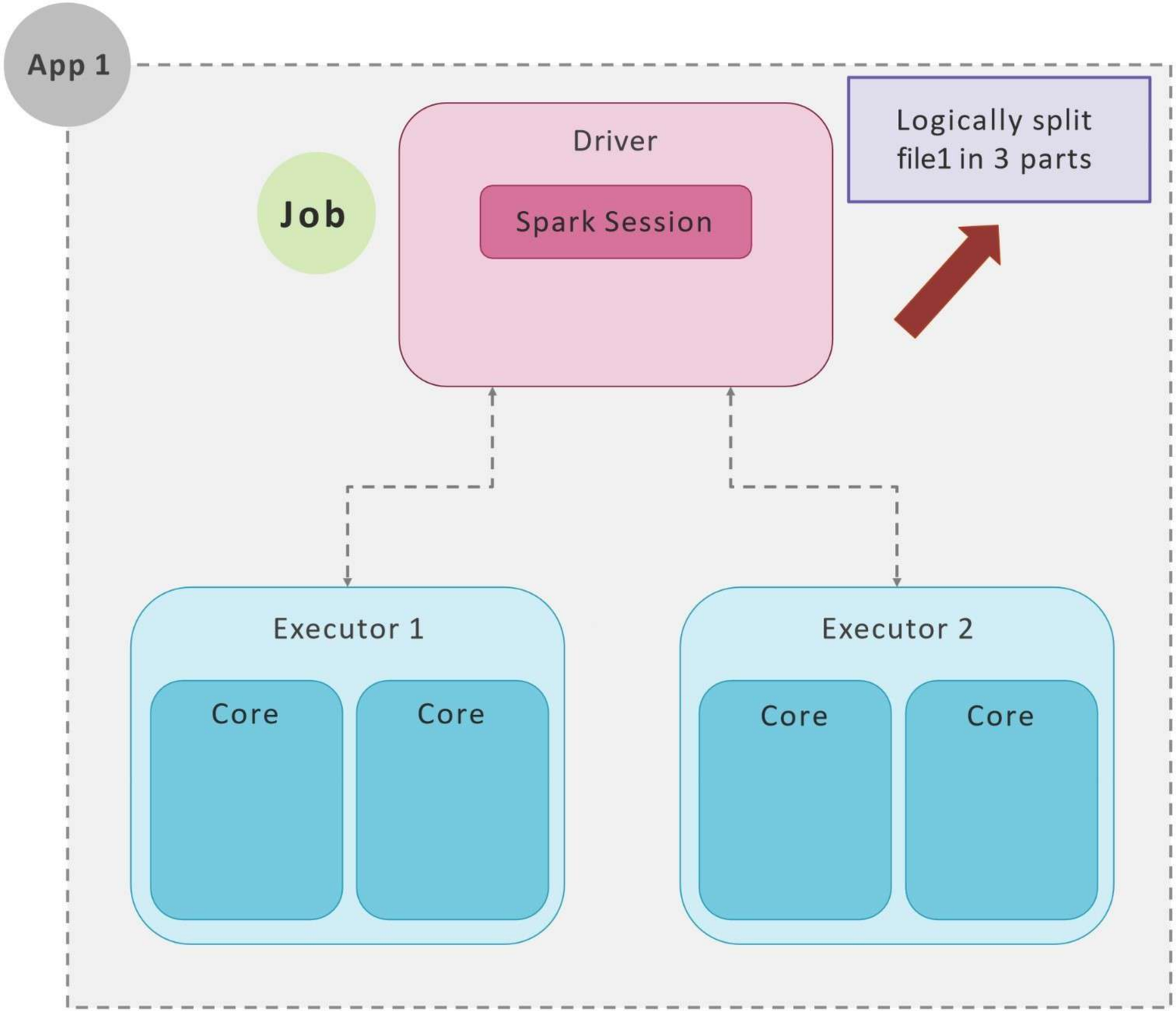
RDD in Spark

- In-memory objects
- Do not have schema
- Distributed collection of elements
- All processing in Spark happens on RDDs

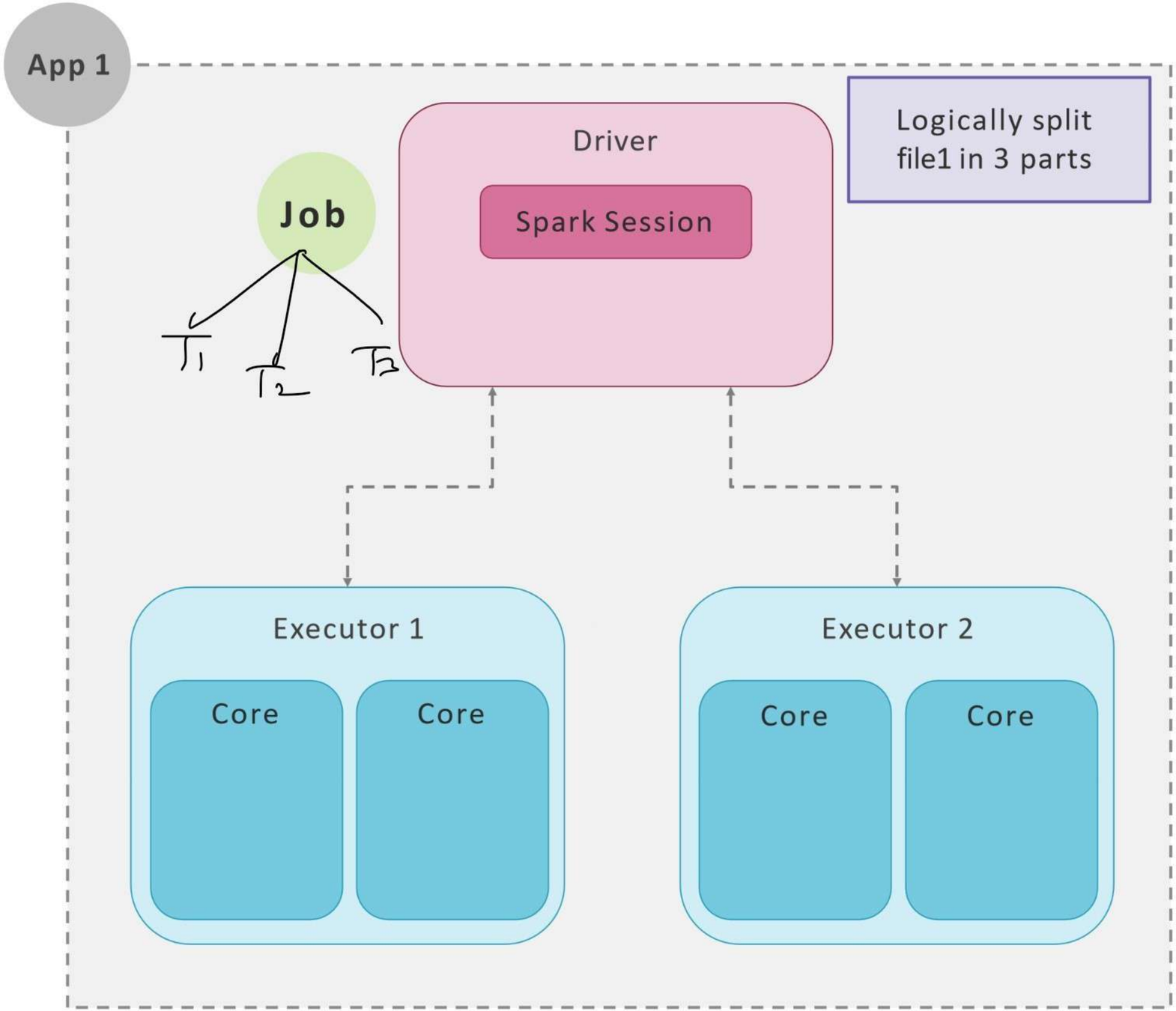

1.Read file1.csv from Storage
2. Filter where City = 'Seattle'
3. Write processed data to Storage
Job
App 1
Driver
Spark Session
Executor 1
Core
Core
Executor 2
Core
Core
file.csv
Id, City, Amount
1,Seattle, 600
2, London, 300
3, Delhi, 700
4, Seattle, 400
5, Paris, 900
6, Delhi, 200
7, Seattle, 900
Storage

1. Read file1.csv from Storage
2. Filter where City = 'Seattle'
3. Write processed data to Storage

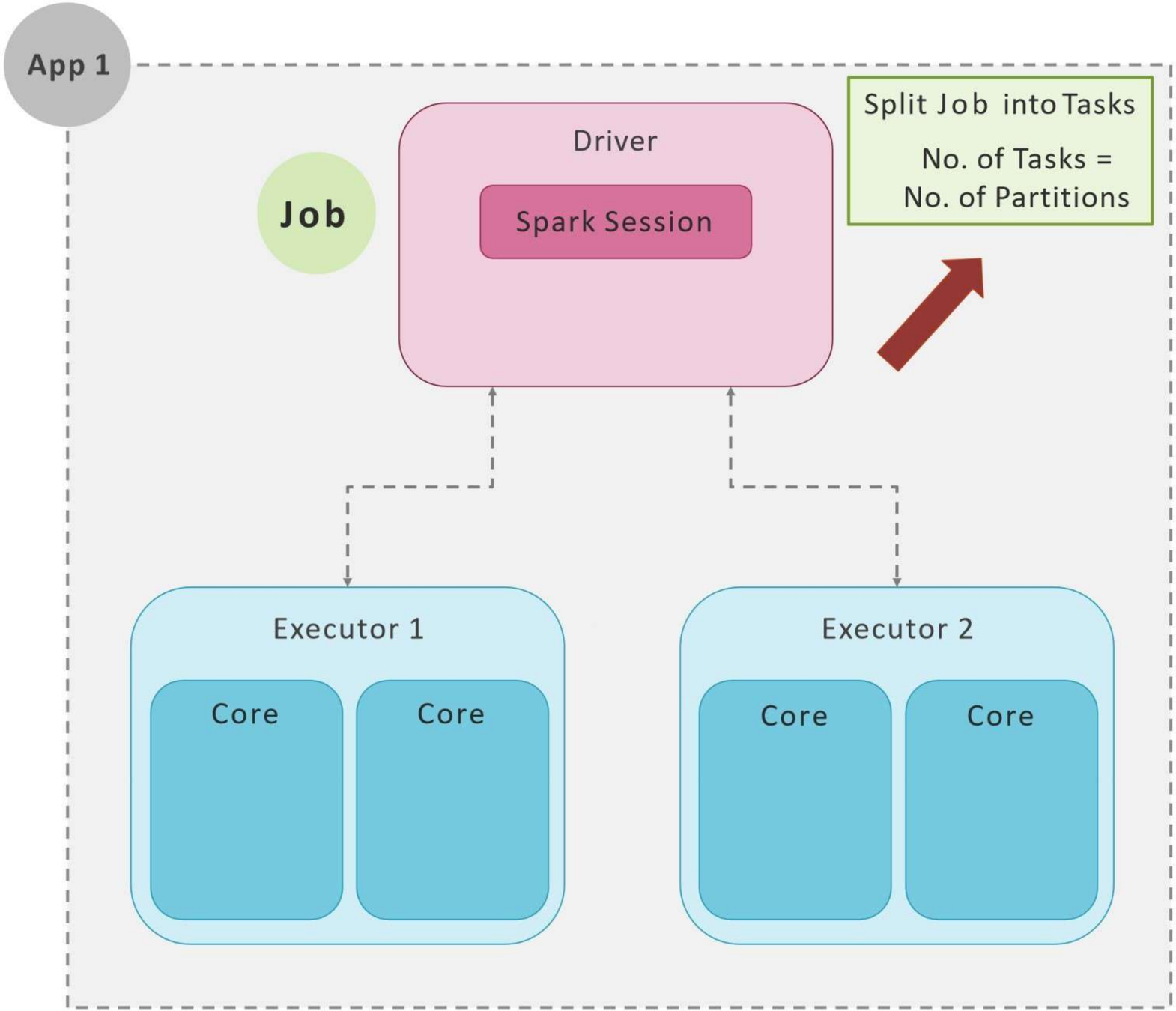




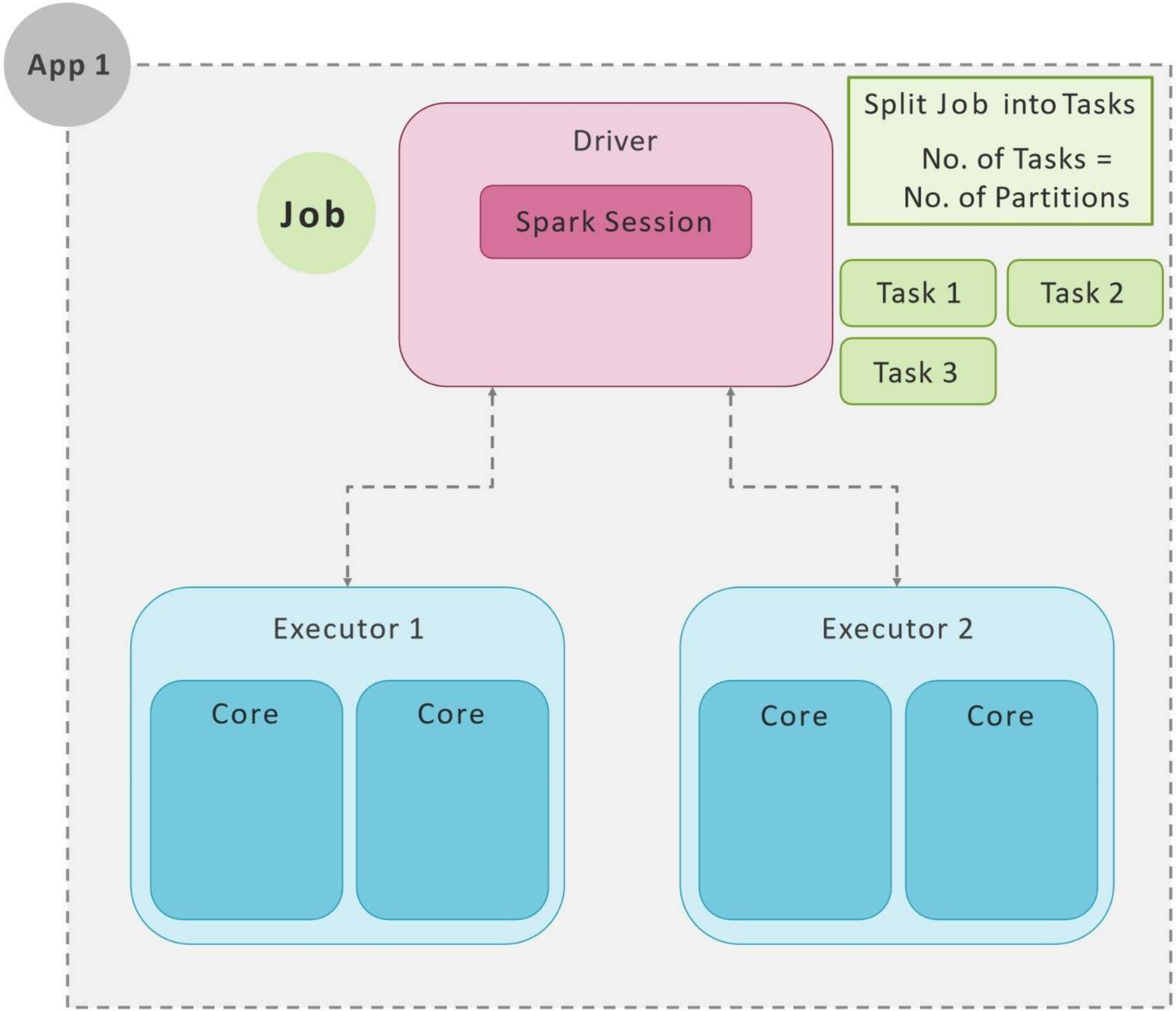
1. Read file1.csv from Storage
2. Filter where City = 'Seattle'
3. Write processed data to Storage

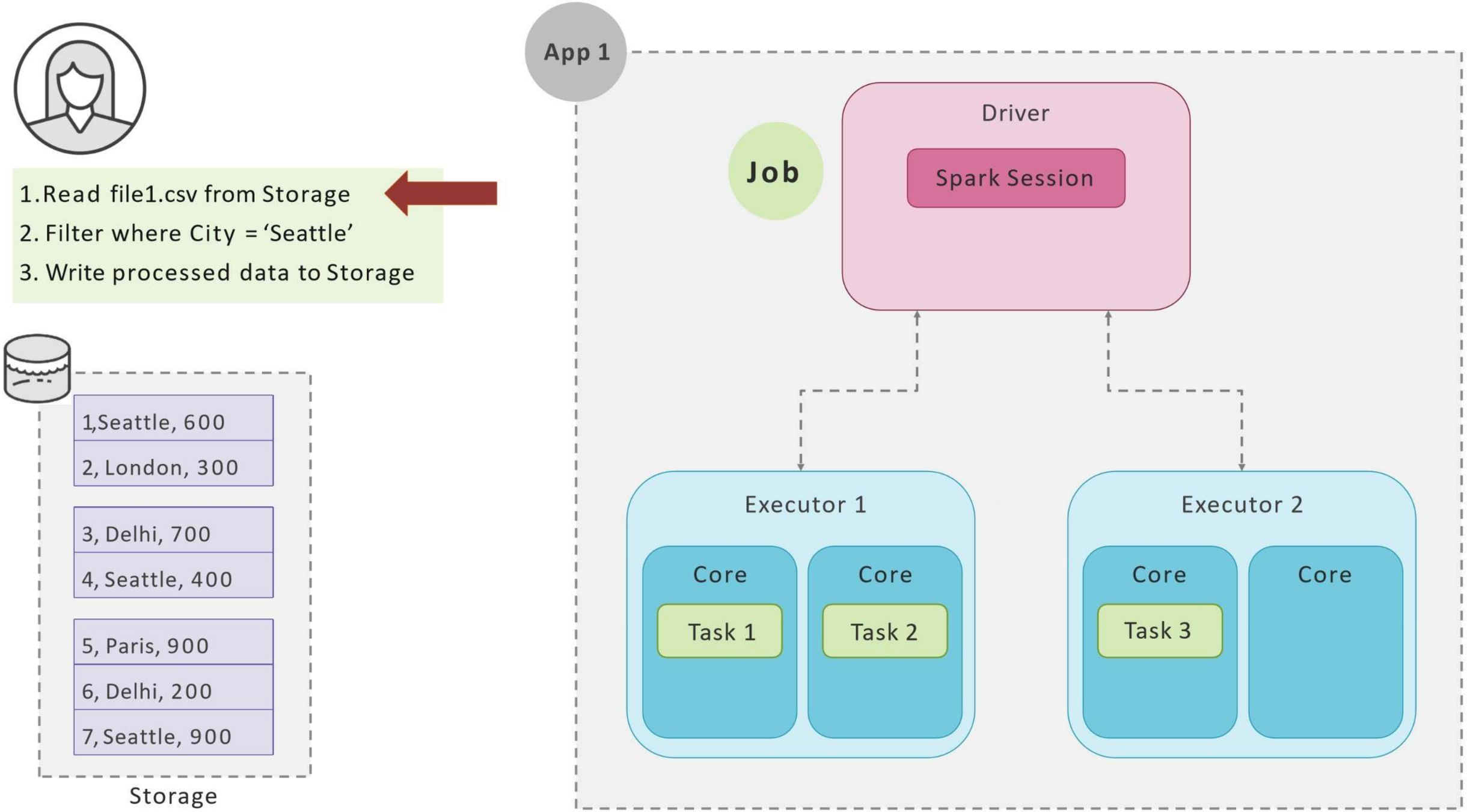


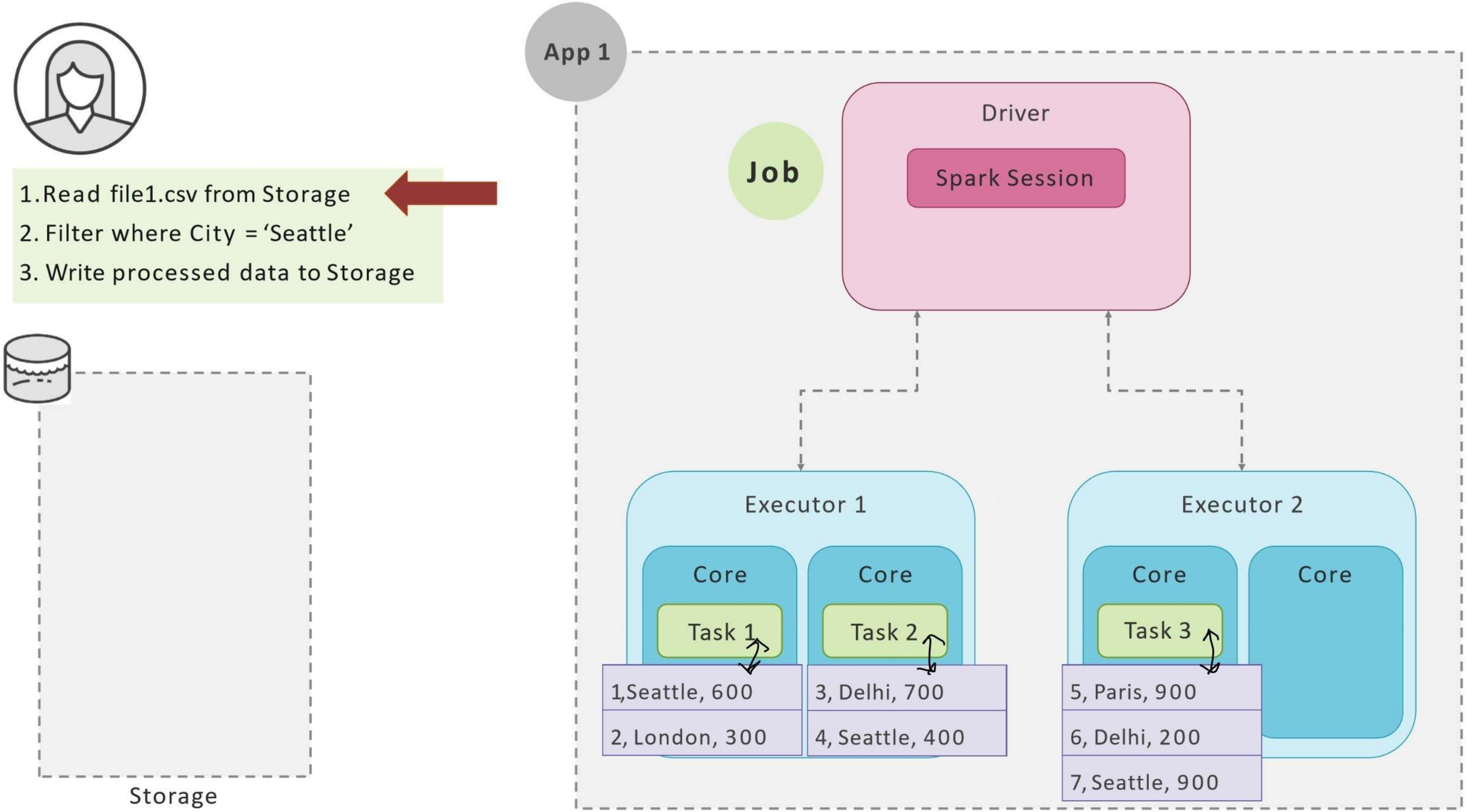
1. Read file1.csv from Storage
2. Filter where City = 'Seattle'
3. Write processed data to Storage

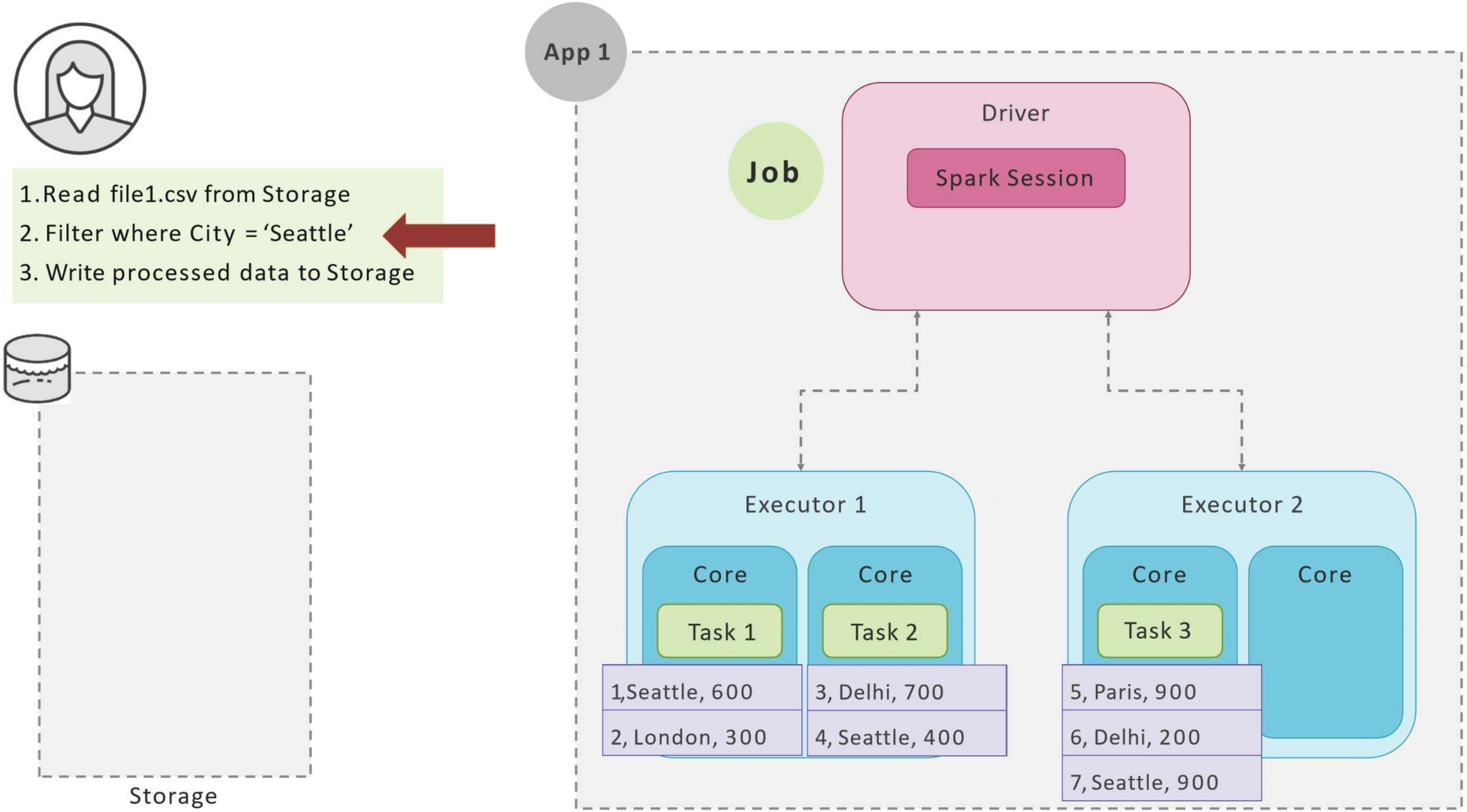


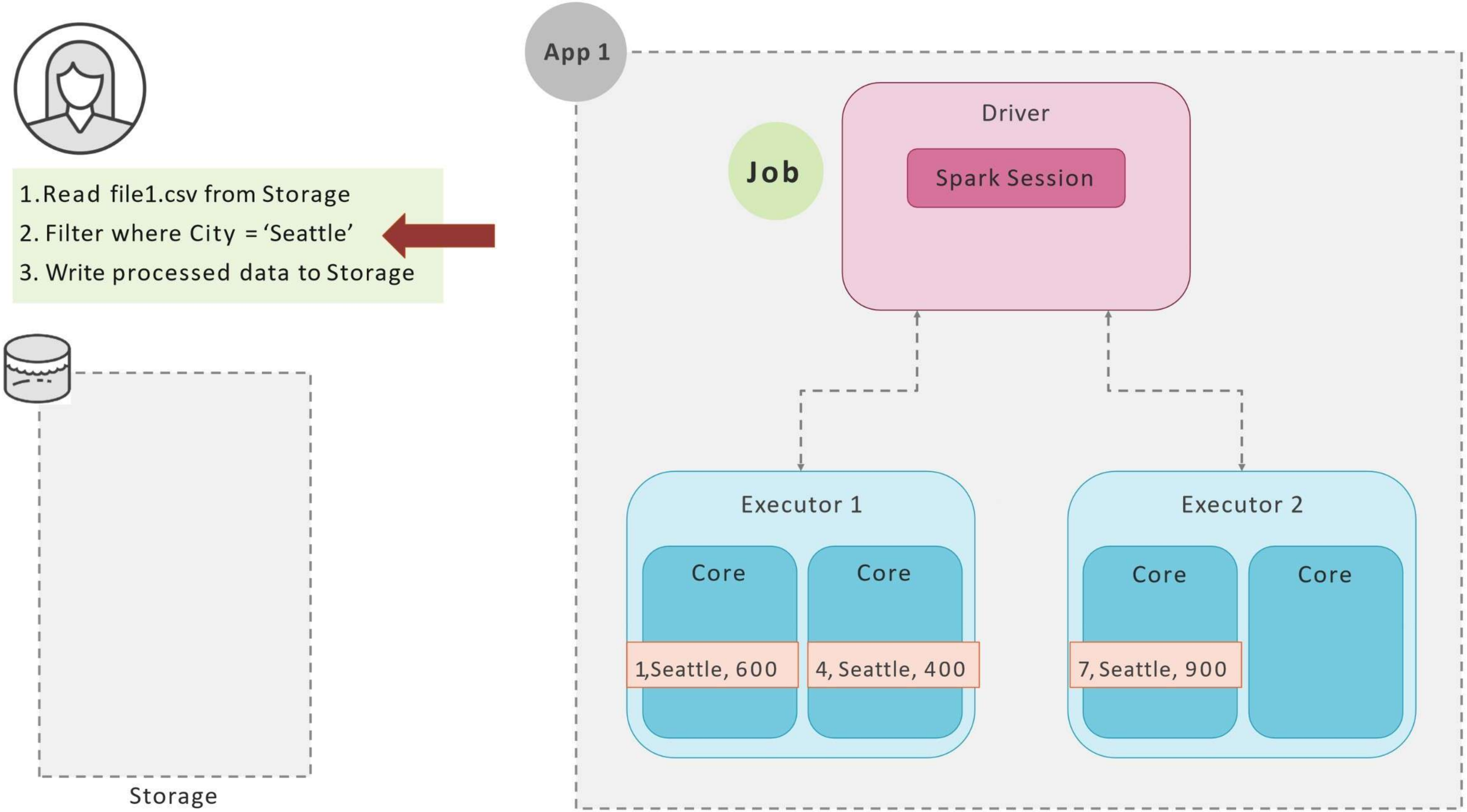
1. Read file1.csv from Storage
2. Filter where City = 'Seattle'
3. Write processed data to Storage

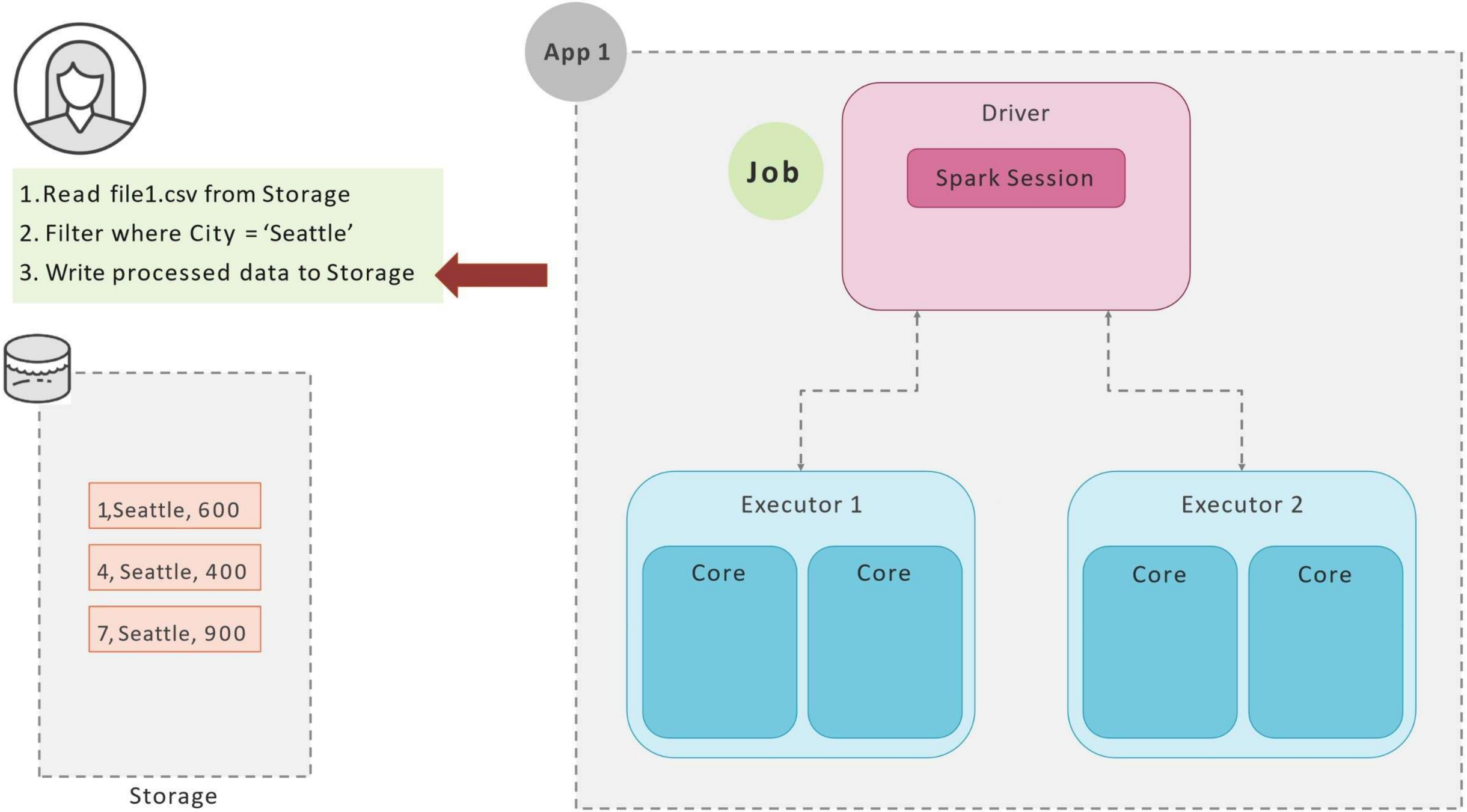



1.Read file1.csv from Storage
2. Filter where City = 'Seattle'
3. Write processed data to Storage
1,Seattle, 600
2, London, 300
3, Delhi, 700
4, Seattle, 400
5, Paris, 900
6, Delhi, 200
7, Seattle, 900
Storage
App 1
Job
Driver
Spark Session
Executor 1
Core
Task 1
Core
Task 2
Executor 2
Core
Task 3
Core


1.Read file1.csv from Storage
2. Filter where City = 'Seattle'
3. Write processed data to Storage
Storage
App 1
Job
Driver
Spark Session
Executor 1
Core
Core
Task 1
Task 2
1,Seattle, 600
2, London, 300
3, Delhi, 700
4, Seattle, 400
Executor 2
Core
Core
Task 3
5, Paris, 900
6, Delhi, 200
7, Seattle, 900


1.Read file1.csv from Storage
2. Filter where City = 'Seattle'
3. Write processed data to Storage
Storage
App 1
Job
Driver
Spark Session
Executor 1
Core
Task 1
Core
Task 2
1,Seattle, 600
2, London, 300
3, Delhi, 700
4, Seattle, 400
Executor 2
Core
Task 3
Core
5, Paris, 900
6, Delhi, 200
7, Seattle, 900


1.Read file1.csv from Storage
2. Filter where City = 'Seattle'
3. Write processed data to Storage
Storage
App 1
Job
Driver
Spark Session
Executor 1
Core
Core
1,Seattle, 600
4, Seattle, 400
Executor 2
Core
Core
7, Seattle, 900


App 1
Job
Driver
Spark Session
1.Read file1.csv from Storage
2. Filter where City = 'Seattle'
3. Write processed data to Storage
Executor 1
Core
Core
Executor 2
Core
Core
1,Seattle, 600
4, Seattle, 400
7, Seattle, 900
Storage

Features of RDDs

In-memory

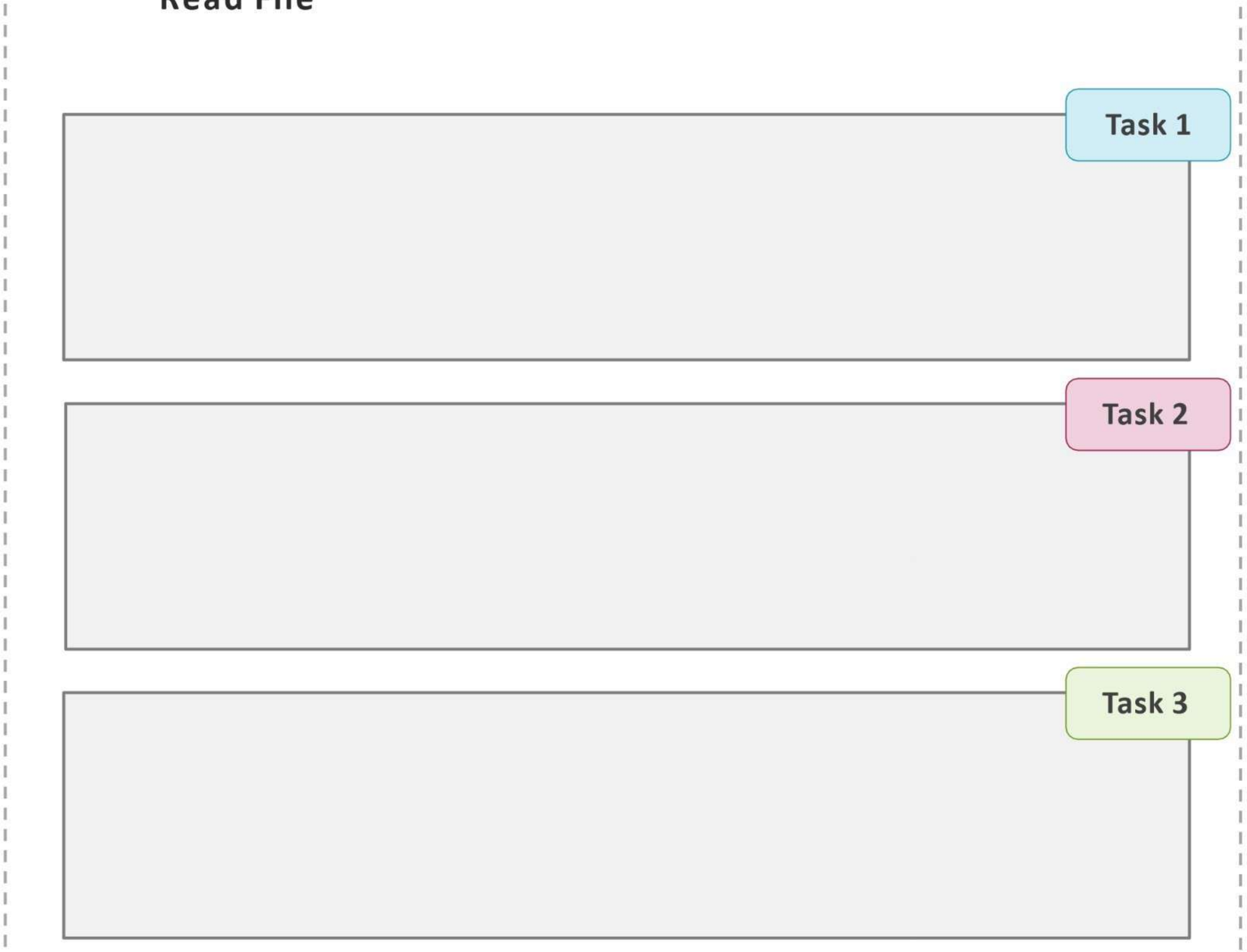
Resides in the memory of cluster

Partitioned

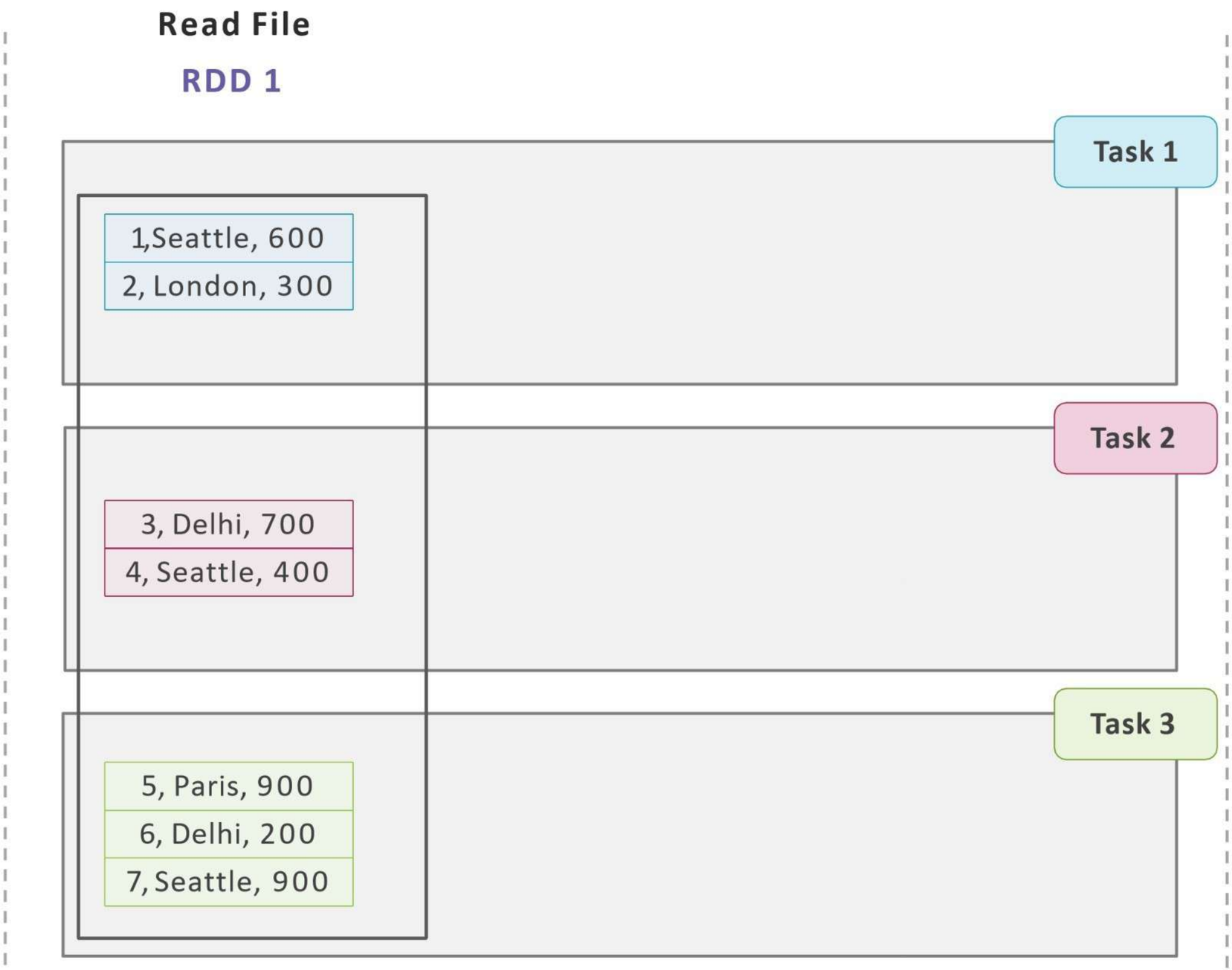
Split into partitions, processed by tasks

Read File

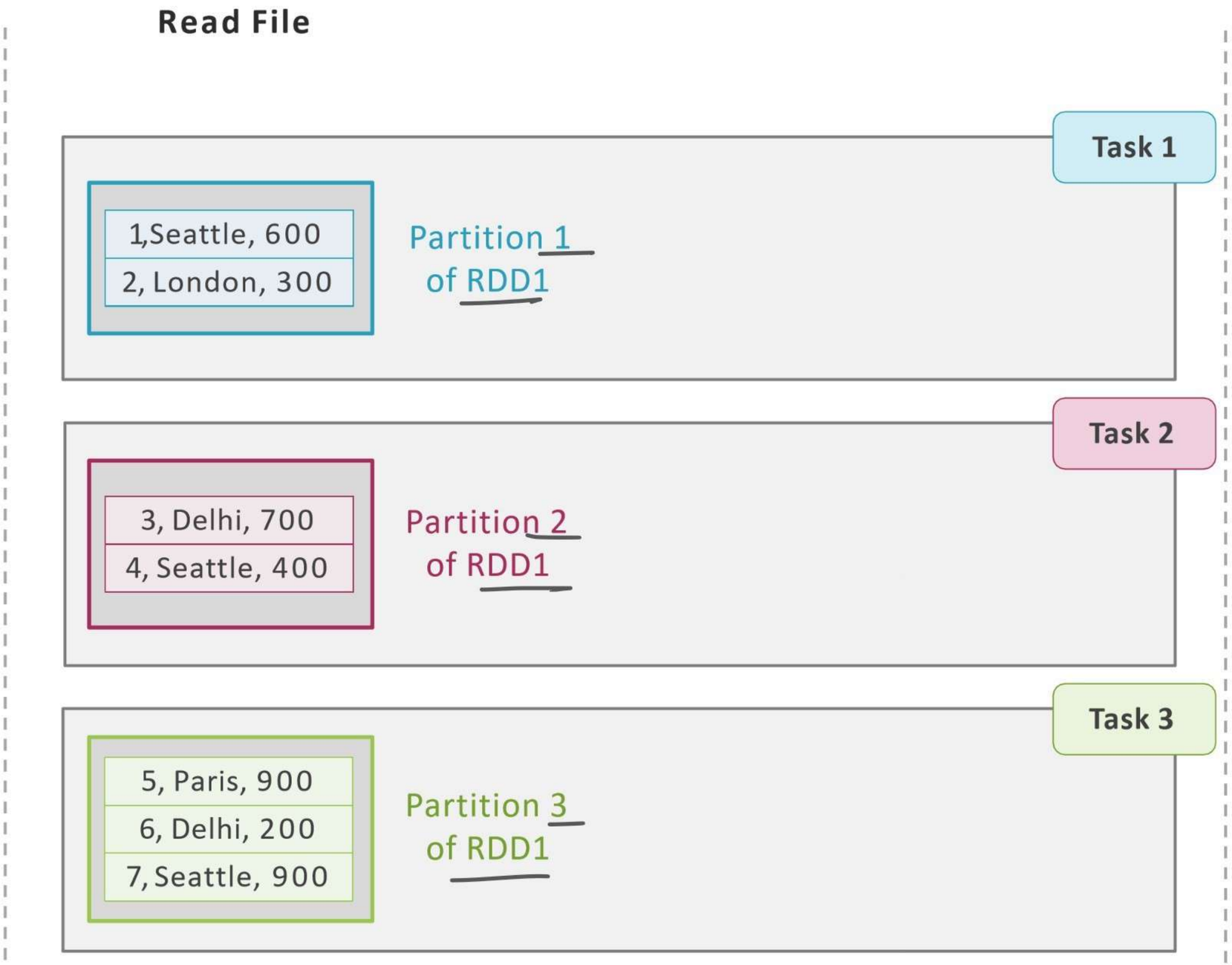
Id, City, Amount
1,Seattle, 600
2, London, 300
3, Delhi, 700
4, Seattle, 400
5, Paris, 900
6, Delhi, 200
7, Seattle, 900



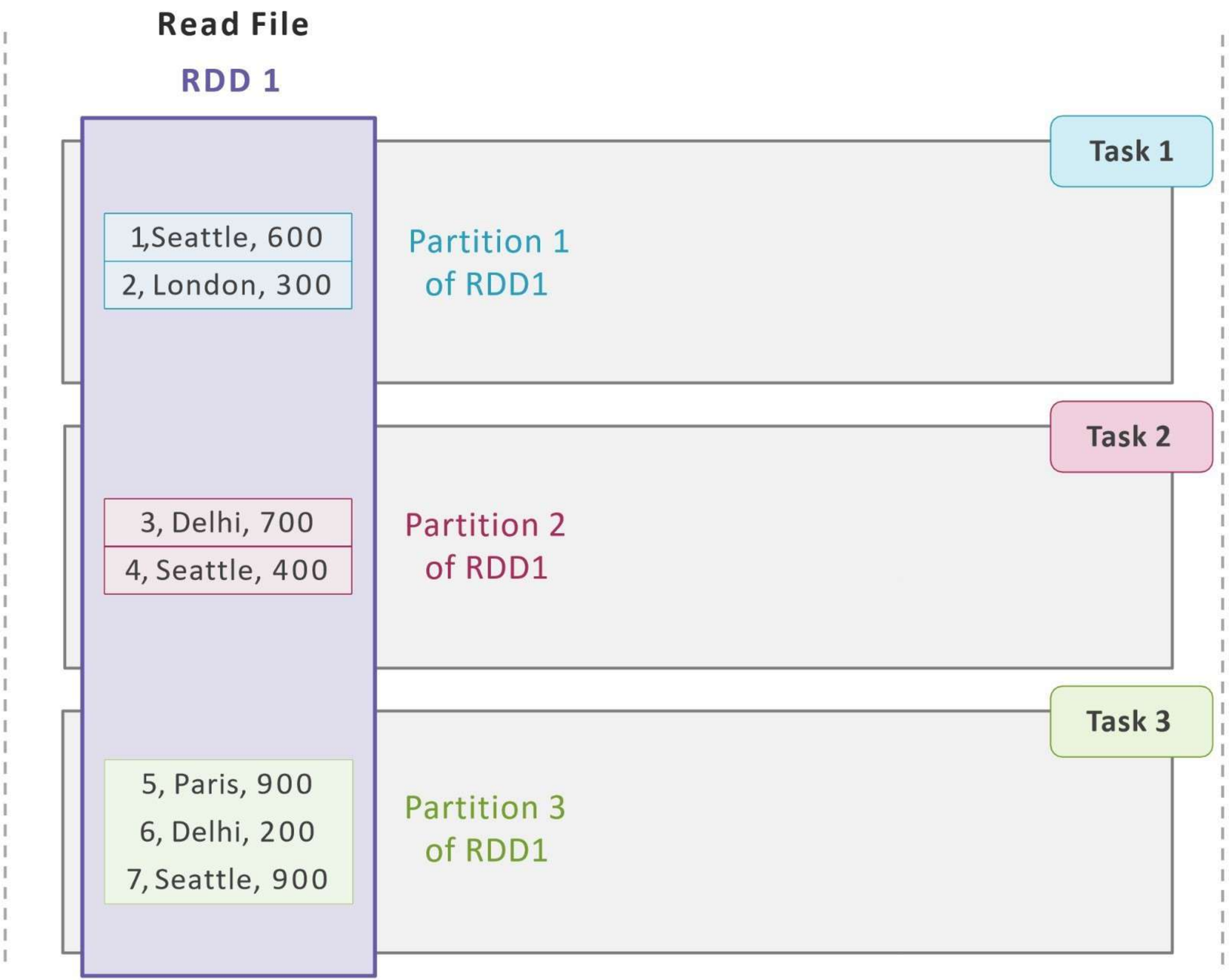
Id, City, Amount
1,Seattle, 600
2, London, 300
3, Delhi, 700
4, Seattle, 400
5, Paris, 900
6, Delhi, 200
7, Seattle, 900



Id, City, Amount
1,Seattle, 600
2, London, 300
3, Delhi, 700
4, Seattle, 400
5, Paris, 900
6, Delhi, 200
7, Seattle, 900



Id, City, Amount
1,Seattle, 600
2, London, 300
3, Delhi, 700
4, Seattle, 400
5, Paris, 900
6, Delhi, 200
7, Seattle, 900



Features of RDDs

In-memory

Resides in the memory of cluster

Partitioned

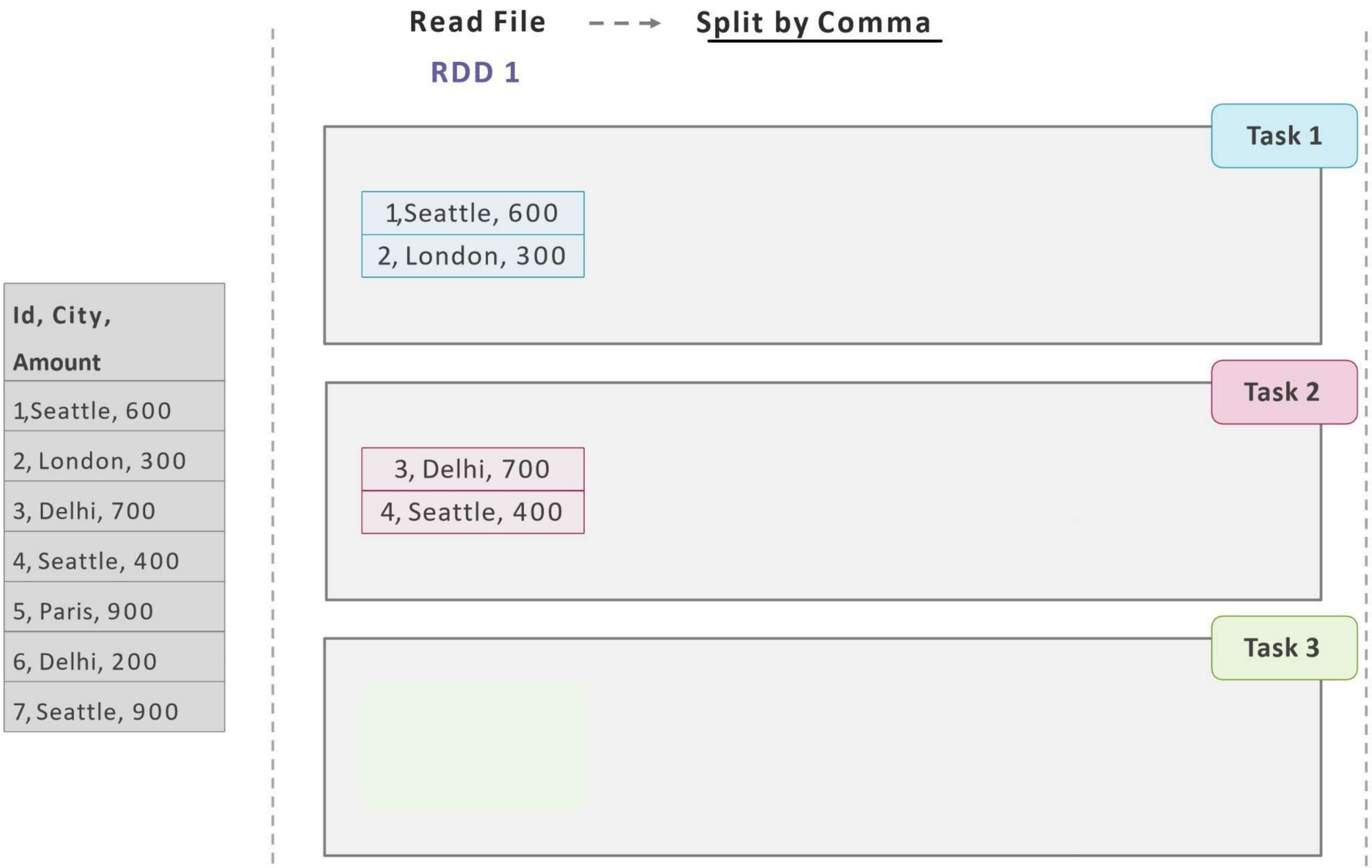
Split into partitions, processed by tasks

Read-only

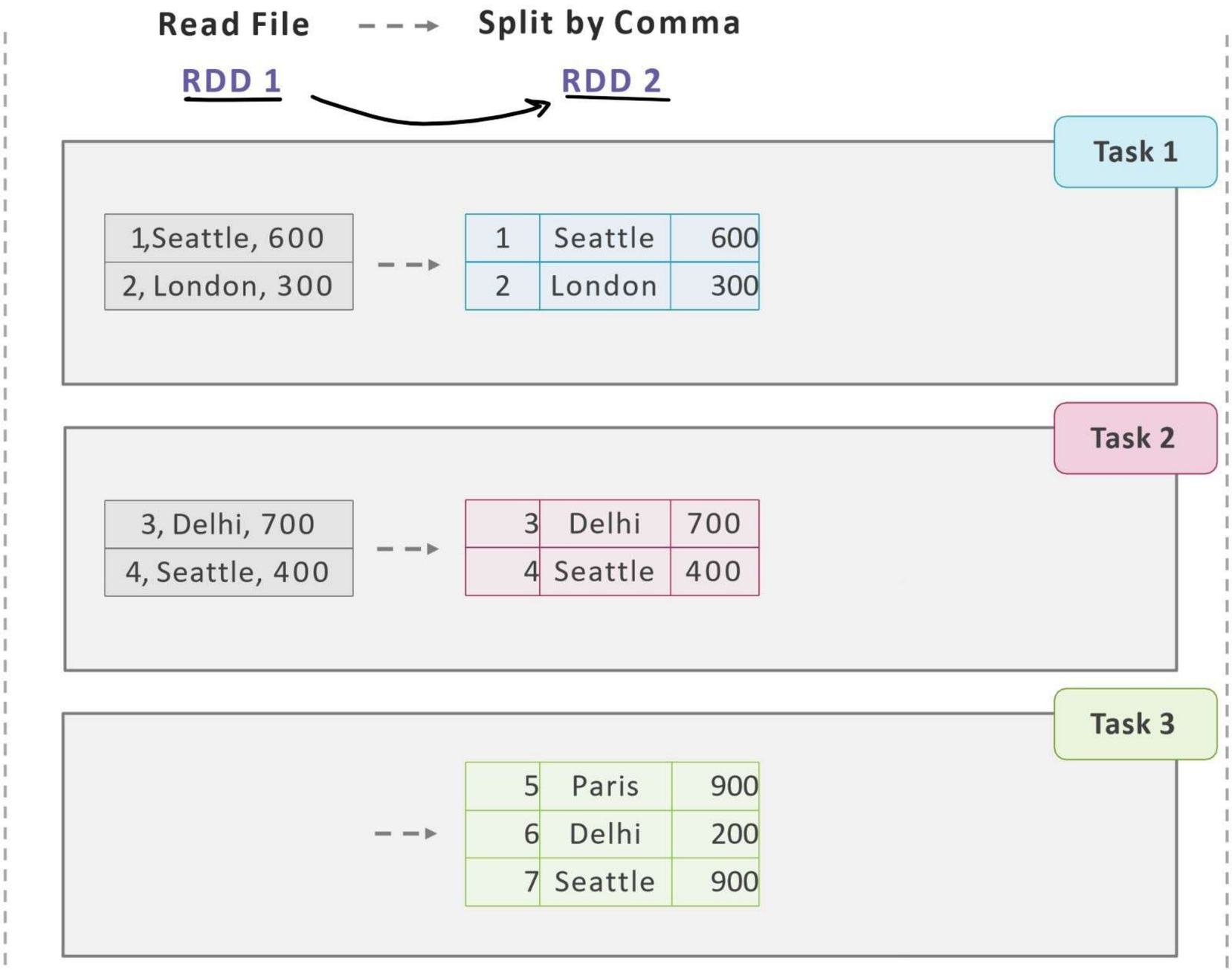
Transformed into another RDD or result

Transformation

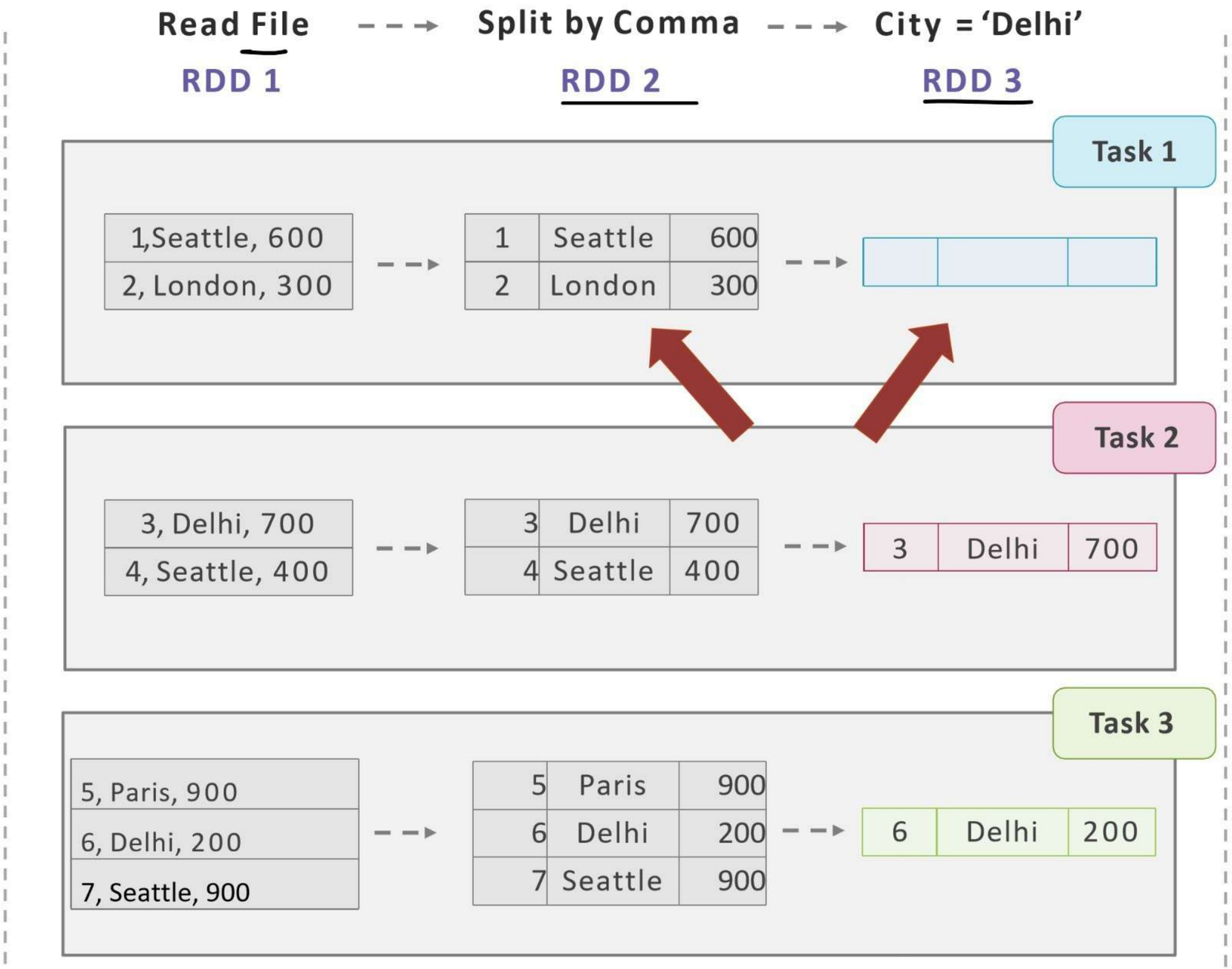
Action


Read File
Split by Comma
RDD 1
Id, City, Amount
1,Seattle, 600
2, London, 300
3, Delhi, 700
4, Seattle, 400
5, Paris, 900
6, Delhi, 200
7, Seattle, 900
Task 1
1,Seattle, 600
2, London, 300
Task 2
3, Delhi, 700
4, Seattle, 400
Task 3

Id, City, Amount
1,Seattle, 600
2, London, 300
3, Delhi, 700
4, Seattle, 400
5, Paris, 900
6, Delhi, 200
7, Seattle, 900



Id, City, Amount
1,Seattle, 600
2, London, 300
3, Delhi, 700
4, Seattle, 400
5, Paris, 900
6, Delhi, 200
7, Seattle, 900



Id, City, Amount
1,Seattle, 600
2, London, 300
3, Delhi, 700
4, Seattle, 400
5, Paris, 900
6, Delhi, 200
7, Seattle, 900

Read File → **Split by Comma** → **City = 'Delhi'**

RDD 1 **RDD 2** **RDD 3**

Task 1

1,Seattle, 600
2, London, 300

→

1	Seattle	600
2	London	300

→

Task 2

3, Delhi, 700
4, Seattle, 400

→

3	Delhi	700
4	Seattle	400

→

3	Delhi	700

Task 3

→

5	Paris	900
6	Delhi	200
7	Seattle	900

→

6	Delhi	200

3	Delhi	700
6	Delhi	200

Features of RDDs

In-memory

Resides in the memory of cluster

Partitioned

Split into partitions, processed by tasks

Read-only

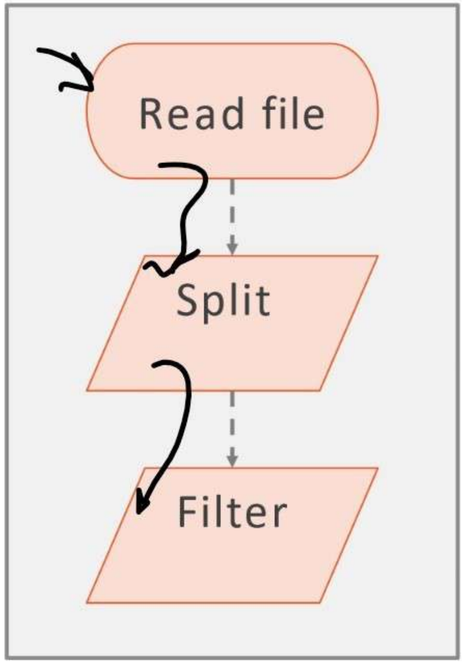
Transformed into another RDD or result

Resilient

Auto recover in case of a failure

Id, City, Amount
1,Seattle, 600
2, London, 300
3, Delhi, 700
4, Seattle, 400
5, Paris, 900
6, Delhi, 200
7, Seattle, 900

1. Read file1.csv from Storage → T
2. Split data by comma → T
3. Filter where City='Delhi' → T
4. Write processed data to storage → A



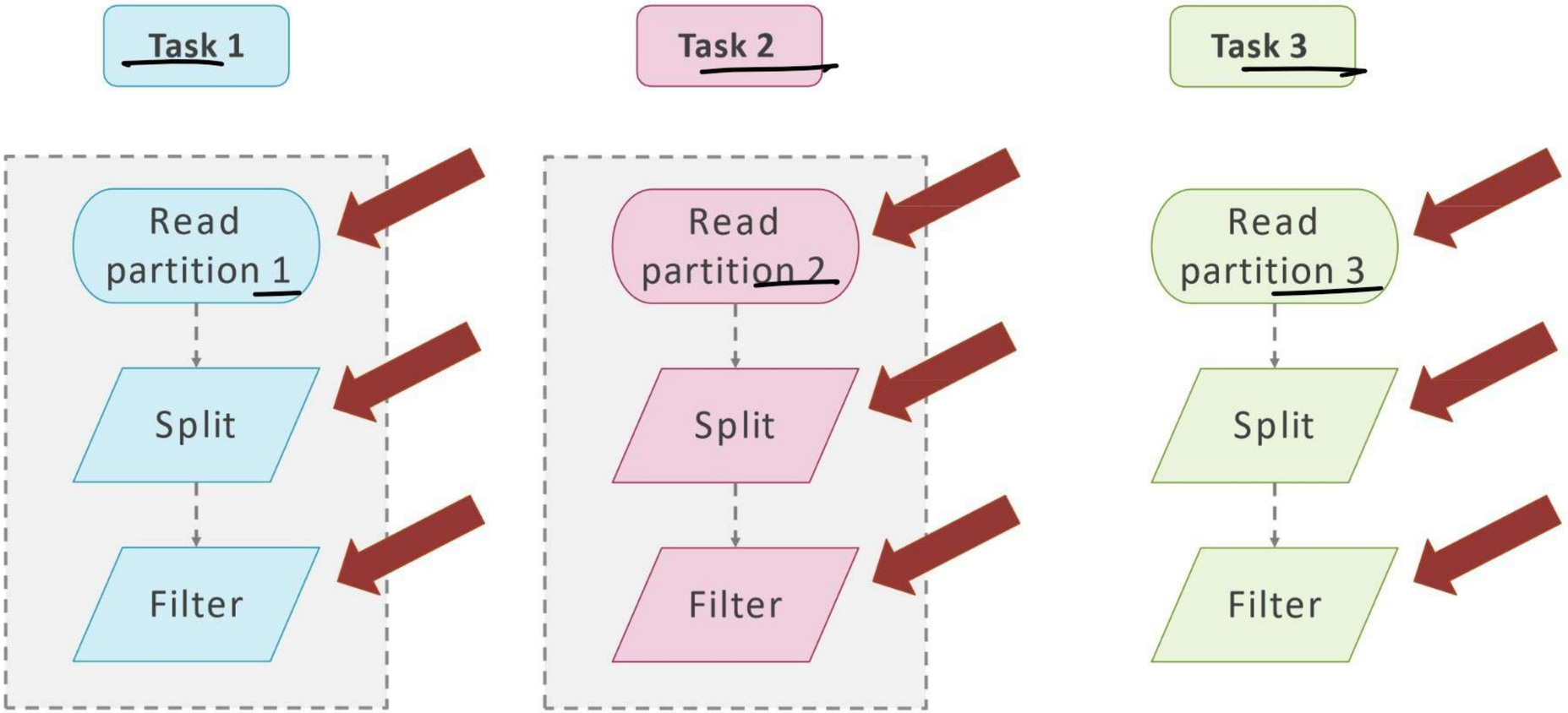
Directed Acyclic Graphs

Lineage Graph

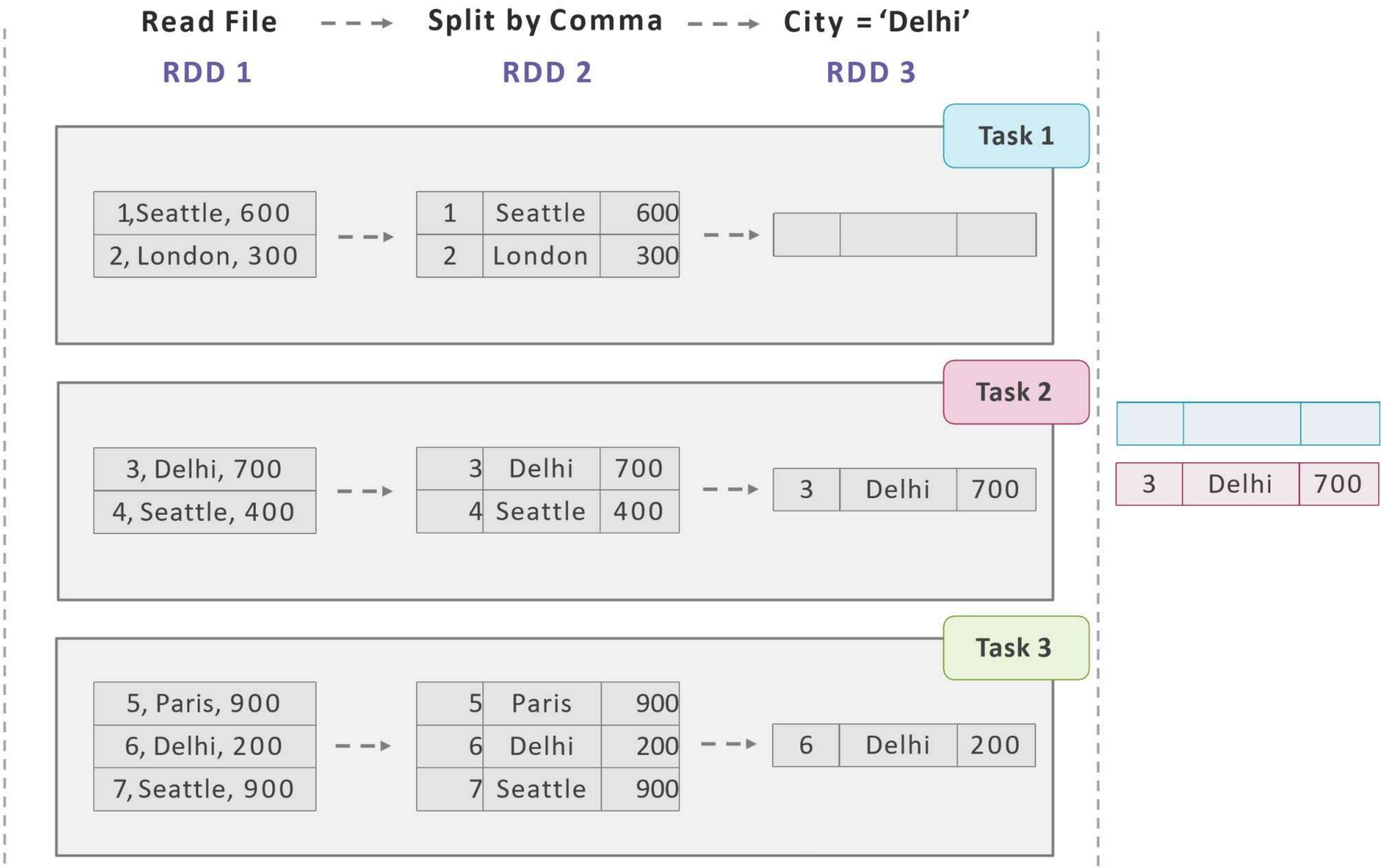
Defines steps that are used for creation of an RDD

1. Read file1.csv from Storage
2. Split data by comma
3. Filter where City='Delhi'
4. Write processed data to storage

Id, City, Amount
1,Seattle, 600
2, London, 300
3, Delhi, 700
4, Seattle, 400
5, Paris, 900
6, Delhi, 200
7, Seattle, 900



Id, City, Amount
1,Seattle, 600
2, London, 300
3, Delhi, 700
4, Seattle, 400
5, Paris, 900
6, Delhi, 200
7, Seattle, 900



Id, City, Amount
1,Seattle, 600
2, London, 300
3, Delhi, 700
4, Seattle, 400
5, Paris, 900
6, Delhi, 200
7, Seattle, 900

Read File → **Split by Comma** → **City = 'Delhi'**

RDD 1 **RDD 2** **RDD 3**

Task 1

1,Seattle, 600	→	1	Seattle	600	→			
2, London, 300		2	London	300				

Task 2

3, Delhi, 700	→	3	Delhi	700	→	3	Delhi	700
4, Seattle, 400		4	Seattle	400				

Task 3

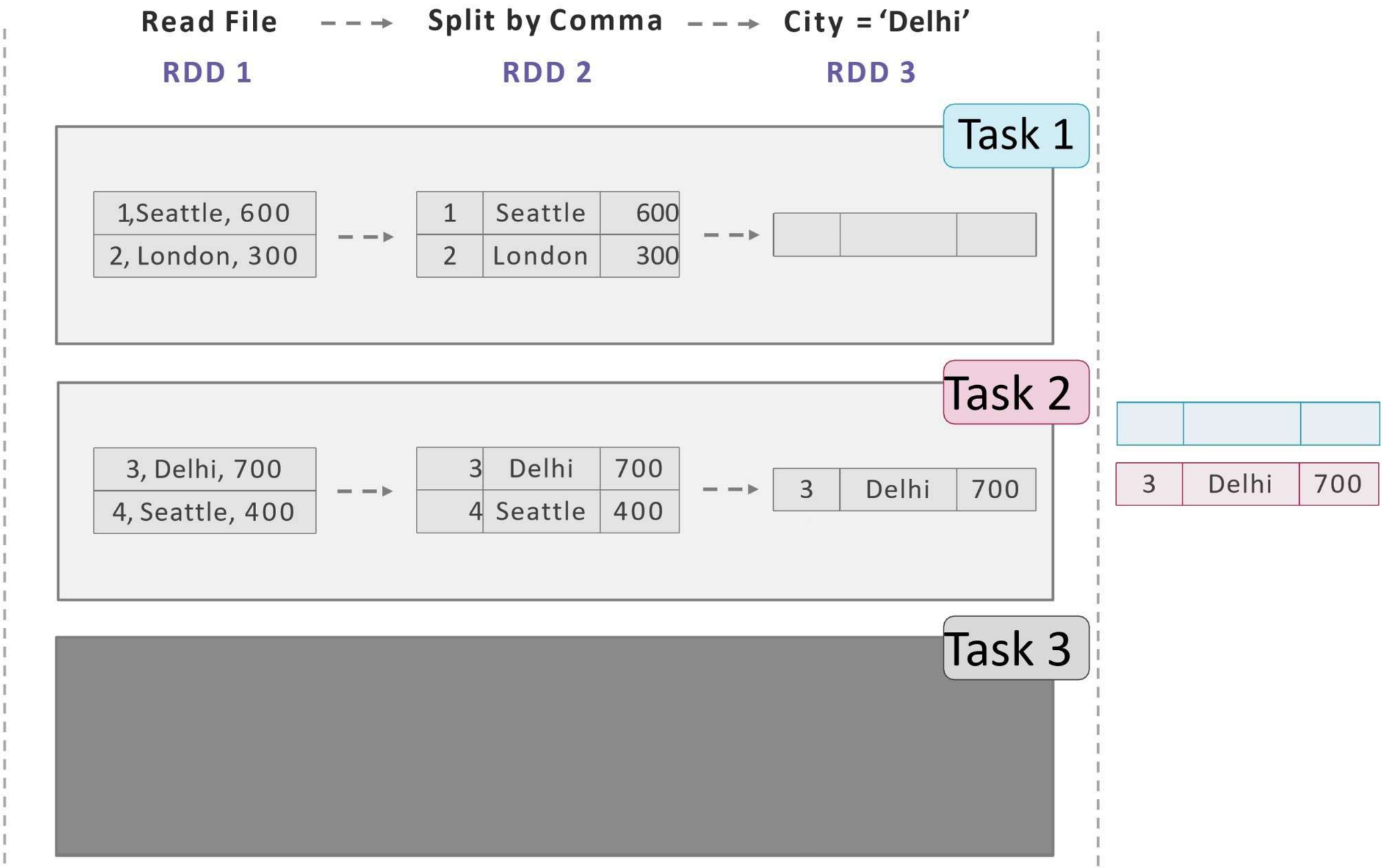
3	Delhi	700

Let say Task 1 and 2 runs fine, but Task 3 fails

Spark will re-execute failed Tasks using their Lineage Graph

Only if the failure is transient, &not permanent (like divide by zero)

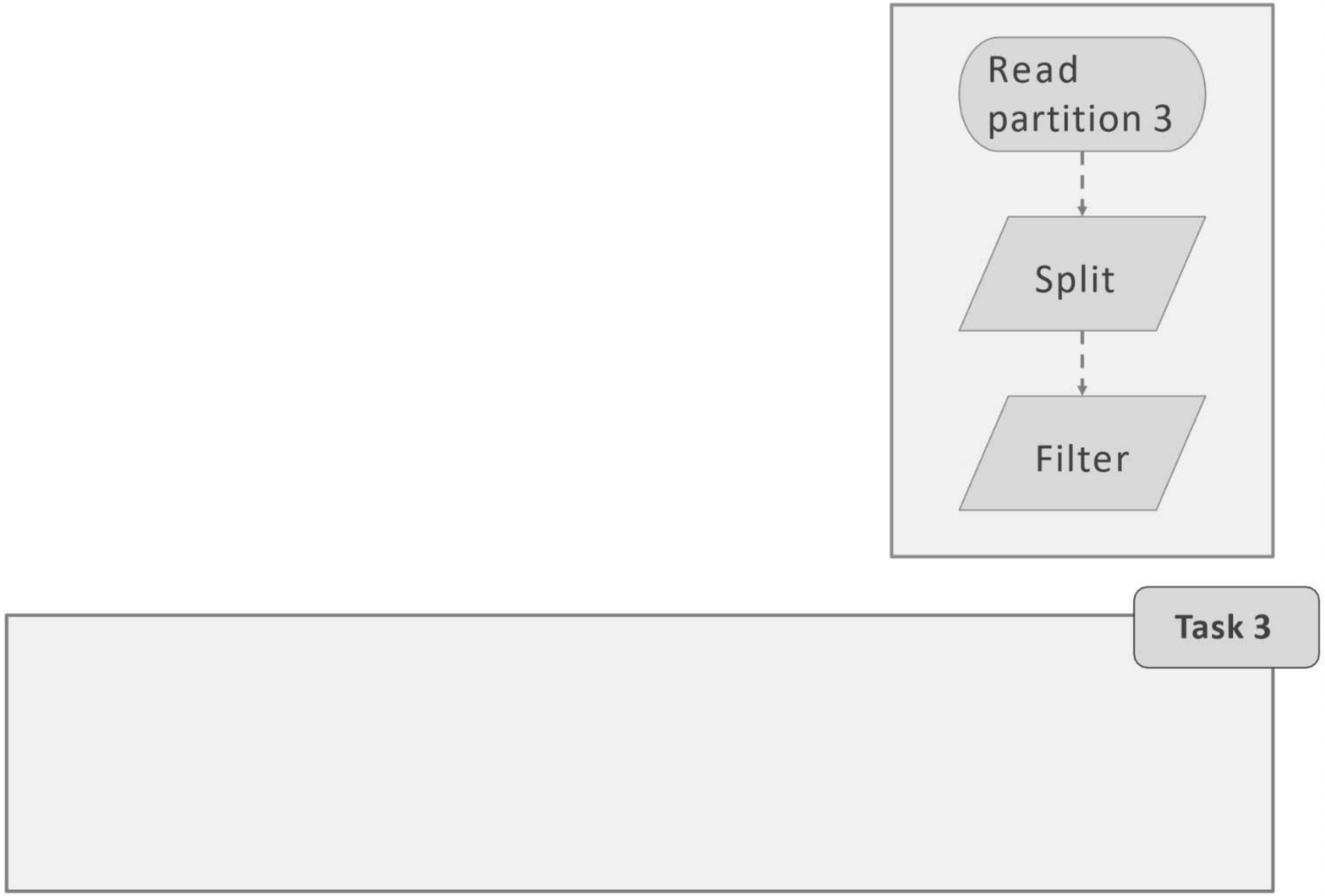
Id, City, Amount
1,Seattle, 600
2, London, 300
3, Delhi, 700
4, Seattle, 400
5, Paris, 900
6, Delhi, 200
7, Seattle, 900



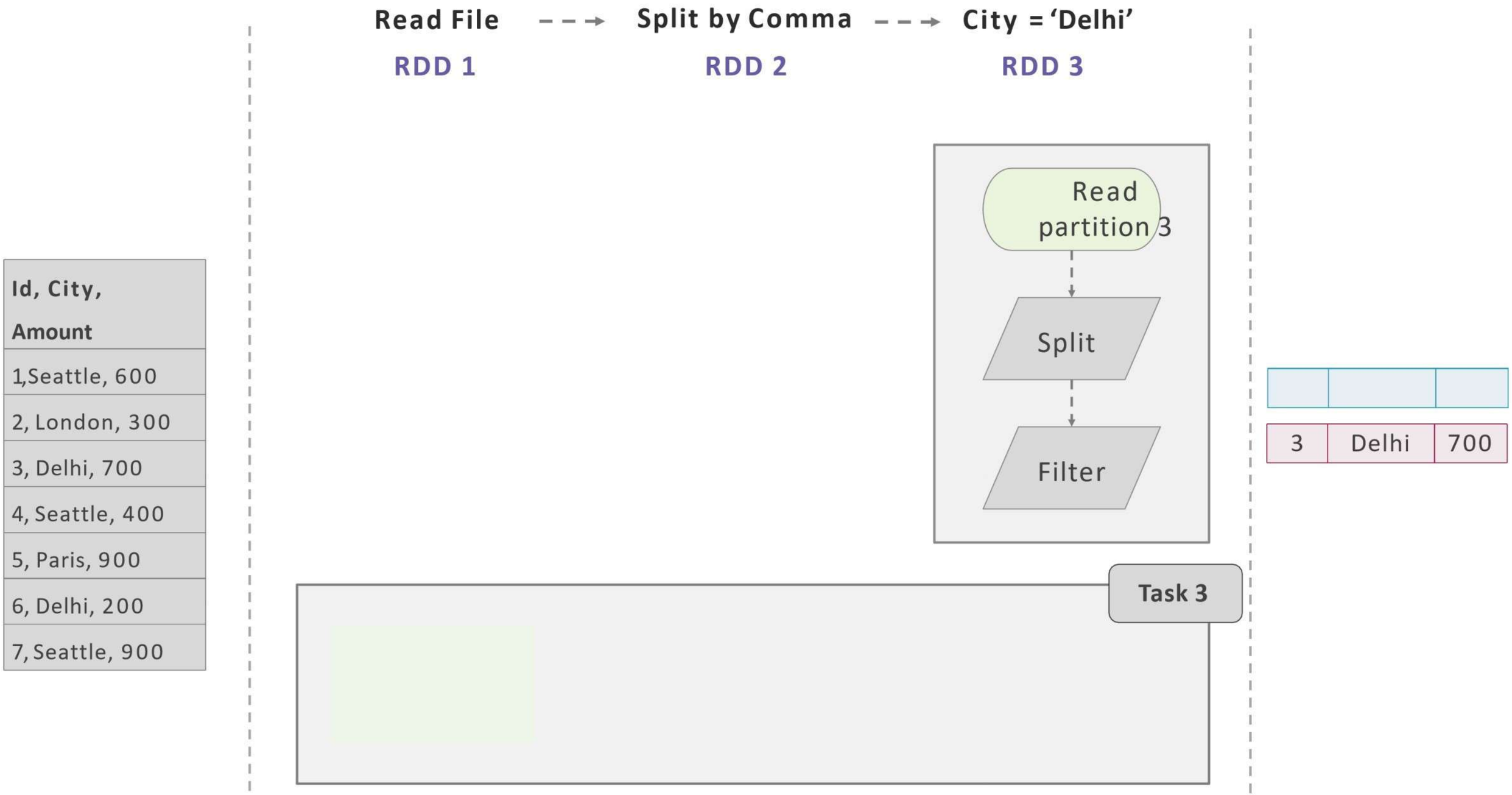
Read File → Split by Comma → City = 'Delhi'

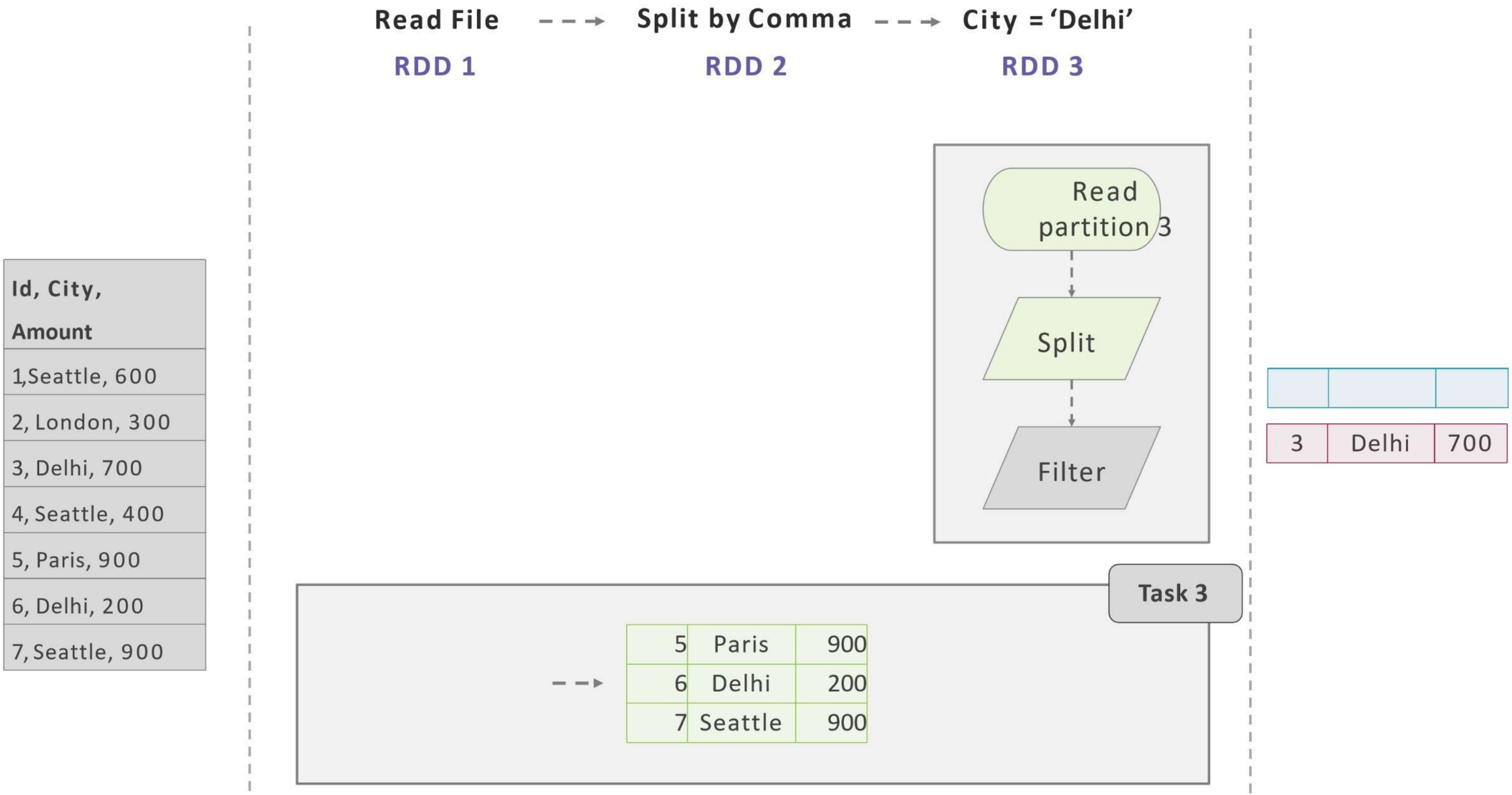
RDD 1 RDD 2 RDD 3

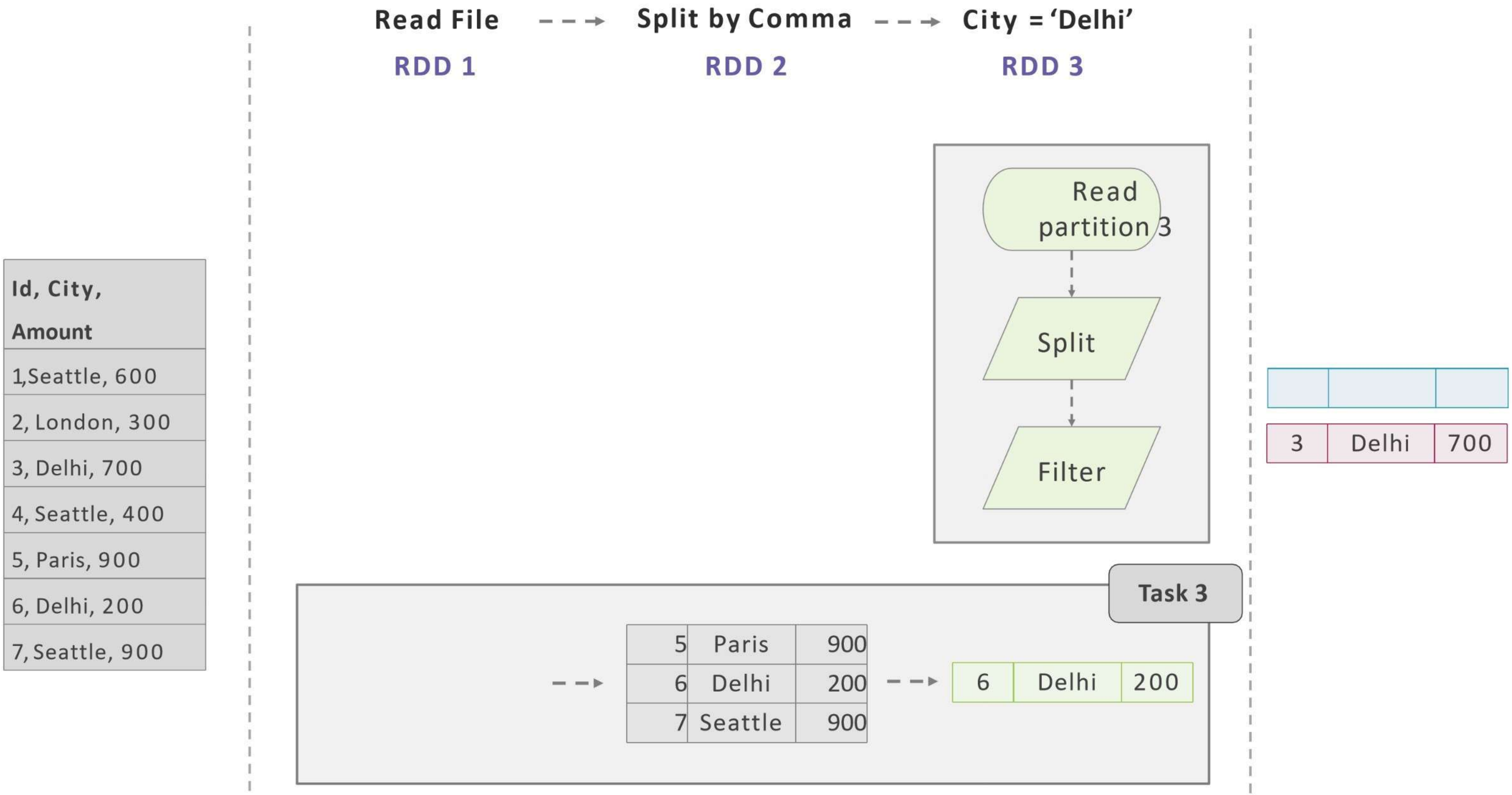
Id, City, Amount
1,Seattle, 600
2, London, 300
3, Delhi, 700
4, Seattle, 400
5, Paris, 900
6, Delhi, 200
7, Seattle, 900

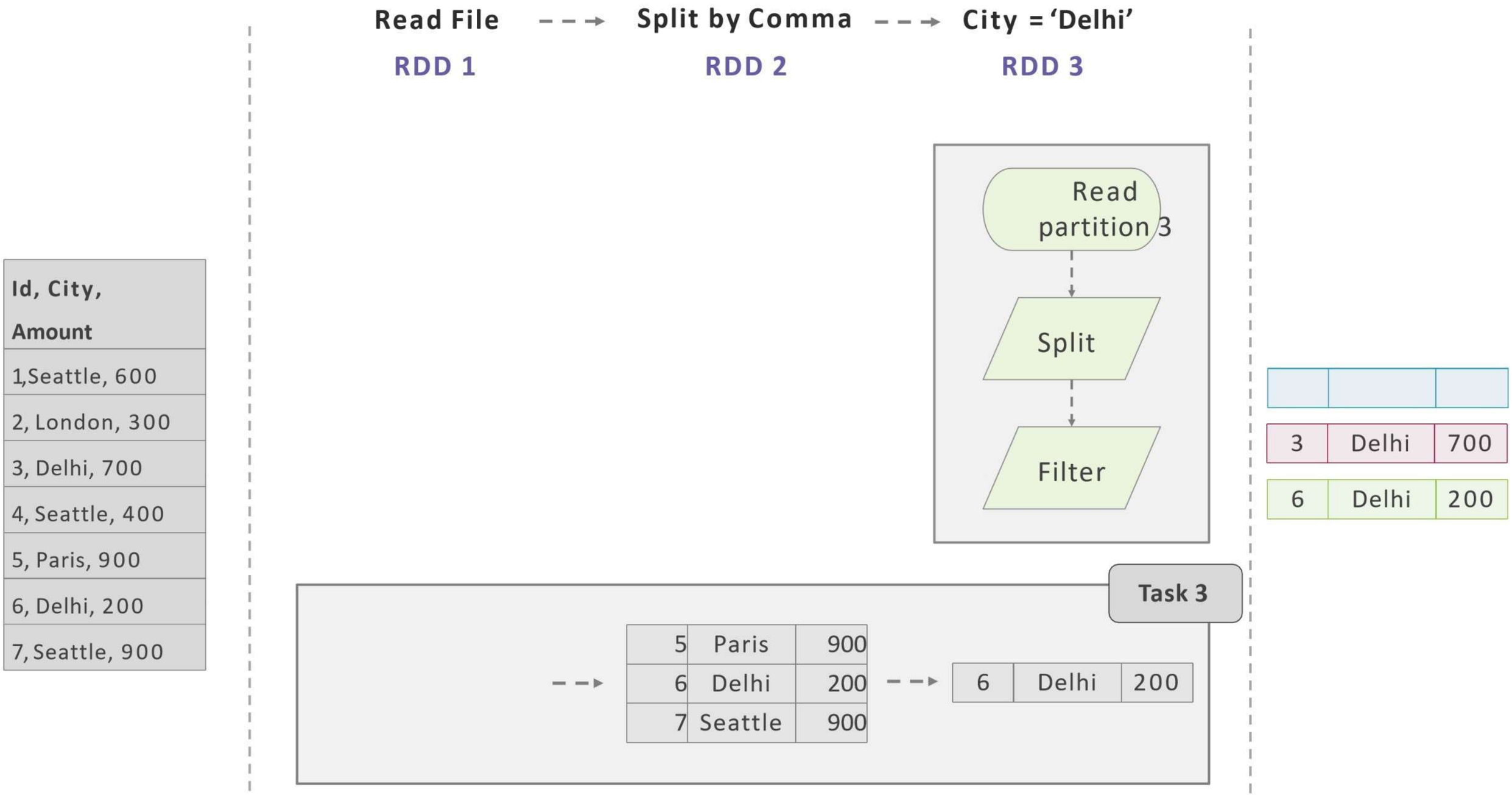


3	Delhi	700


Read File
Split by Comma
City = 'Delhi'
RDD 1
RDD 2
RDD 3
Id, City, Amount
1,Seattle, 600
2, London, 300
3, Delhi, 700
4, Seattle, 400
5, Paris, 900
6, Delhi, 200
7, Seattle, 900
Read partition 3
Split
Filter
Task 3
3
Delhi
700


Read File
Split by Comma
City = 'Delhi'
RDD 1
RDD 2
RDD 3
Id, City, Amount
1,Seattle, 600
2, London, 300
3, Delhi, 700
4, Seattle, 400
5, Paris, 900
6, Delhi, 200
7, Seattle, 900
Read partition 3
Split
Filter
Task 3
5 Paris 900
6 Delhi 200
7 Seattle 900
3 Delhi 700


Read File
RDD 1
Split by Comma
RDD 2
City = 'Delhi'
RDD 3
Id, City, Amount
1,Seattle, 600
2, London, 300
3, Delhi, 700
4, Seattle, 400
5, Paris, 900
6, Delhi, 200
7, Seattle, 900
Read partition 3
Split
Filter
Task 3
5 Paris 900
6 Delhi 200
7 Seattle 900
6 Delhi 200
3 Delhi 700


Read File
RDD 1
Split by Comma
RDD 2
City = 'Delhi'
RDD 3
Id, City, Amount
1,Seattle, 600
2, London, 300
3, Delhi, 700
4, Seattle, 400
5, Paris, 900
6, Delhi, 200
7, Seattle, 900
Read partition 3
Split
Filter
Task 3
5 Paris 900
6 Delhi 200
7 Seattle 900
6 Delhi 200
3 Delhi 700
6 Delhi 200

Features of RDDs

In-memory

Resides in the memory of cluster

Partitioned

Split into partitions, processed by tasks

Read-only

Transformed into another RDD or result

Resilient

Auto recover in case of a failure

Creating RDDs

Options to Create RDD

Parallelize a Collection

Read a File

From another RDD

```python
import findspark
findspark.init()
findspark.find()
```

```
'C:\\Installation\\spark-3.3.2-bin-hadoop3'
```

```python
from pyspark.sql import SparkSession
from pyspark.sql.types import *

spark = (
            SparkSession
                .builder
                .appName("RDDApp")
                .master("local[4]")
                .getOrCreate()
        )
```

```python
spark
```

Create RDD - Option 1: Parallelize

+ Code + Markdown

```
# Create a variable for SparkContext

sc = spark.sparkContext
```

```
# Create RDD using parallelize

numbersRdd = sc.parallelize( [1, 2, 3, 4, 5] )
```

[6]

```
# Check RDD partitions

numbersRdd.getNumPartitions()
```

```
4
```

[7]

```
# Get result from RDD

output = numbersRdd.collect()

print(output)
```

```
[1, 2, 3, 4, 5]
```

[8]

```
# Get any 2 records from RDD

numbersRdd.take(2)
```

```
[1, 2]
```

```python
# Get first record from RDD
# first() returns an element, while take(1) returns an array with one element

numbersRdd.first()
```

```
1
```

```python
# Create RDD with complex types

employeesRdd = sc.parallelize(
                                [
                                    [ 1, "Neha",  10000 ],
                                    [ 2, "Steve", 20000 ],
                                    [ 3, "Kari",  30000 ],
                                    [ 4, "Ivan",  40000 ],
                                    [ 5, "Mohit", 50000 ]
                                ]
                            )
```

```python
# Get first record from RDD

employeesRdd.first()
```

```
[1, 'Neha', 10000]
```

Create RDD - Option 2: Read a File

```
# Read TaxiZones.csv file and create RDD

taxiZonesRdd = sc.textFile("C:\SparkCourse\DataFiles\Raw\TaxiZones.csv")
```

12]

```
# Get 10 records

taxiZonesRdd.take(10)
```

13]

```
['1,EWR,Newark Airport,EWR',
 '2,Queens,Jamaica Bay,Boro Zone',
 '3,Bronx,Allerton/Pelham Gardens,Boro Zone',
 '4,Manhattan,Alphabet City,Yellow Zone',
 '5,Staten Island,Arden Heights,Boro Zone',
 '6,Staten Island,Arrochar/Fort Wadsworth,Boro Zone',
 '7,Queens,Astoria,Boro Zone',
 '8,Queens,Astoria Park,Boro Zone',
 '9,Queens,Auburndale,Boro Zone',
 '10,Queens,Baisley Park,Boro Zone']
```

```
# Check RDD partitions

taxiZonesRdd.getNumPartitions()
```

14]

2

```
# Read TaxiZones.csv file and create RDD (with 4 partitions)

taxiZonesRdd = sc.textFile("C:\SparkCourse\DataFiles\Raw\TaxiZones.csv", 4)

taxiZonesRdd.getNumPartitions()
```

15]

4

Create RDD - Option 3: Convert another RDD

```
# Apply map operation to split each record by comma, and get array of items

taxiZonesWithColsRdd = (
                        taxiZonesRdd

                            .map( lambda zone: zone.split(",") )
                    )

taxiZonesWithColsRdd.take(5)
```

6]

```
[['1', 'EWR', 'Newark Airport', 'EWR'],
 ['2', 'Queens', 'Jamaica Bay', 'Boro Zone'],
 ['3', 'Bronx', 'Allerton/Pelham Gardens', 'Boro Zone'],
 ['4', 'Manhattan', 'Alphabet City', 'Yellow Zone'],
 ['5', 'Staten Island', 'Arden Heights', 'Boro Zone']]
```

```python
# Apply filter operation
# Get data where Borough is 'Manhattan' and Zone begins with 'central'

filteredZonesRdd = (
                taxiZonesWithColsRdd

                    .filter( lambda zoneRow: zoneRow[1] == "Manhattan"

                                    and zoneRow[2].lower().startswith("central")
                                )
            )

filteredZonesRdd.take(5)
```

```
[['41', 'Manhattan', 'Central Harlem', 'Boro Zone'],
 ['42', 'Manhattan', 'Central Harlem North', 'Boro Zone'],
 ['43', 'Manhattan', 'Central Park', 'Yellow Zone']]
```

```python
# Get list of Zones with even LocationId

evenZoneIds = (
                    taxiZonesRdd

                        .filter( lambda zoneRow: int(zoneRow[0]) % 2 == 0 )
                )

evenZoneIds.take(5)
```

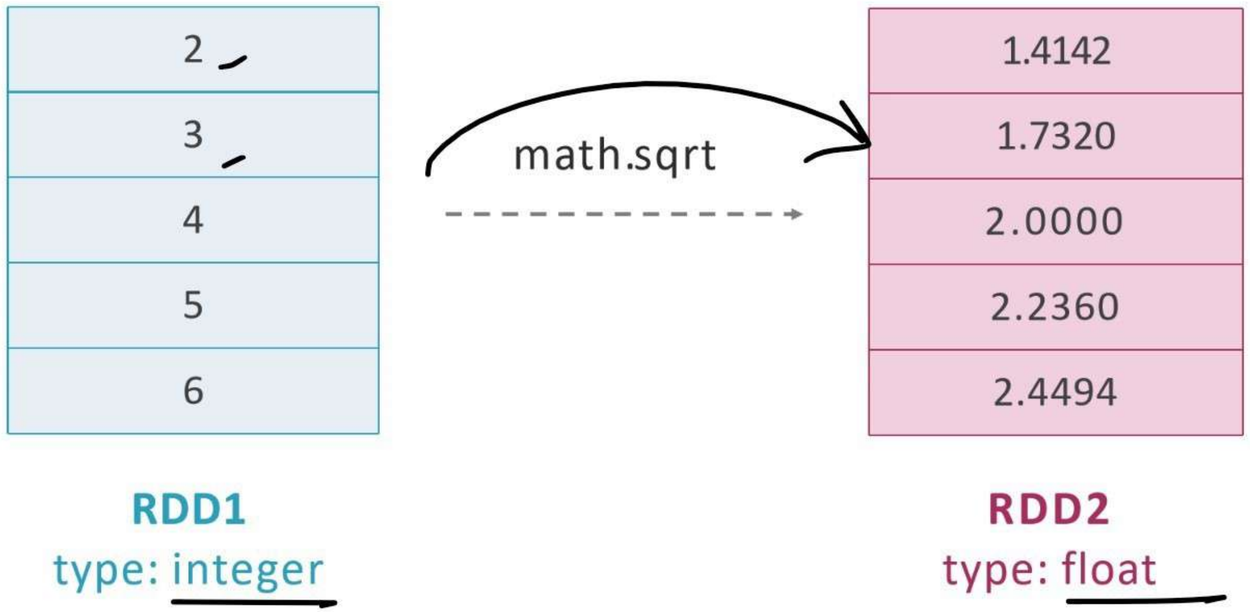
```
['2,Queens,Jamaica Bay,Boro Zone',
 '4,Manhattan,Alphabet City,Yellow Zone',
 '6,Staten Island,Arrochar/Fort Wadsworth,Boro Zone',
 '8,Queens,Astoria Park,Boro Zone',
 '20,Bronx,Belmont,Boro Zone']
```

Working with Pair RDDs

Spark → Scala

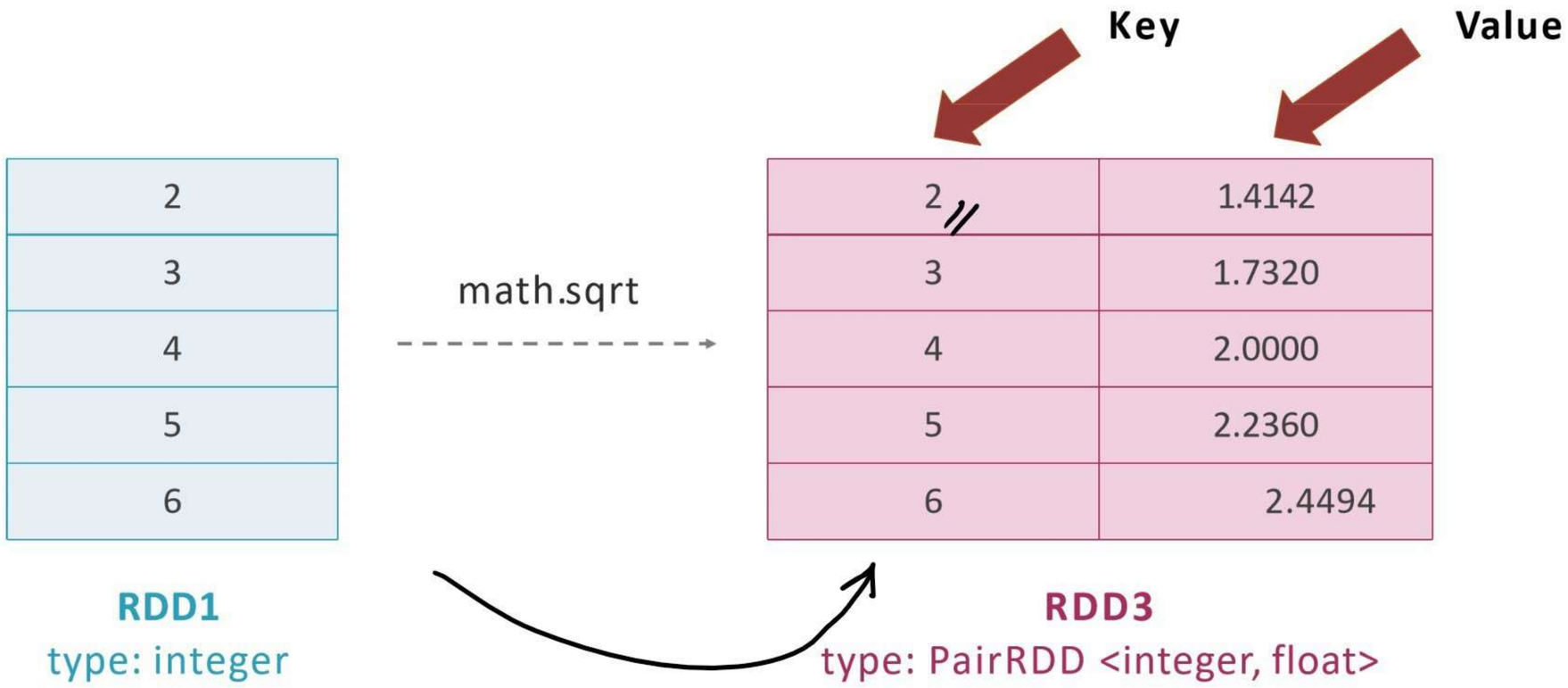
dev

Pair RDDs



Whose square root is it?

Pair RDDs


Key
Value
2
3
4
5
6
math.sqrt
2
1.4142
3
1.7320
4
2.0000
5
2.2360
6
2.4494
RDD1
type: integer
RDD3
type: PairRDD <integer, float>

Pair RDDs

Key
(CustomerID)

Value
(Customer Info)

1	['Mohit', '20000', 'Hyderabad']
2	['Ivan', '23000', 'Seattle']
3	['Kerri', '27000', 'London']
4	['Andrew', '18000', 'Paris']
5	['Neha', '32000', 'Delhi']

Customer Data – PairRDD <integer, array>

Pair RDDs

1	['Mohit', '20000', 'Hyderabad']
2	['Ivan', '23000', 'Seattle']
3	['Kerri', '27000', 'London']
4	['Andrew', '18000', 'Paris']
5	['Neha', '32000', 'Delhi']

RDDs with Key-Value pairs

- Two items are linked together
- Key is identifier. Value is corresponding data

Key need not be unique

Spark has special operations for Pair RDDs

- reducebyKey, sortByKey, countByKey etc.
- Some operations are costly

Pair RDDs

```
import math

numbersRDD = sc.parallelize( [2, 3, 4, 5, 6] )

# Create Pair RDD
numsWithSquareRootRdd = (
                    numbersRDD

                        .map(lambda num: (
                                                num,
                                                math.sqrt(num)
                                            )
                        )
                    )

numsWithSquareRootRdd.collect()
```

```
[(2, 1.4142135623730951),
 (3, 1.7320508075688772),
 (4, 2.0),
 (5, 2.23606797749979),
 (6, 2.449489742783178)]
```

Exercise 1

Use taxiZonesWithColsRdd and create a Pair RDD (LocationId as Key, Zone Details as Value)

```python
# Exercise 1 Answer

taxiZonesPairRdd = (
                    taxiZonesWithColsRdd

                        .map(lambda zoneRow: (
                                                zoneRow[0],
                                                zoneRow
                                              )
                             )
                   )

taxiZonesPairRdd.take(10)
```

reduce()

Calculate sum of Amount by City

Id, City, Amount
1,Seattle, 600
2, Seattle, 300
3, Delhi, 700
4, Seattle, 400
5, Delhi, 900
6, Delhi, 200
7, Seattle, 900

CSV File

Create

Pair RDD

Seattle	600
Seattle	300
Delhi	700
Seattle	400
Delhi	900
Delhi	200
Seattle	900

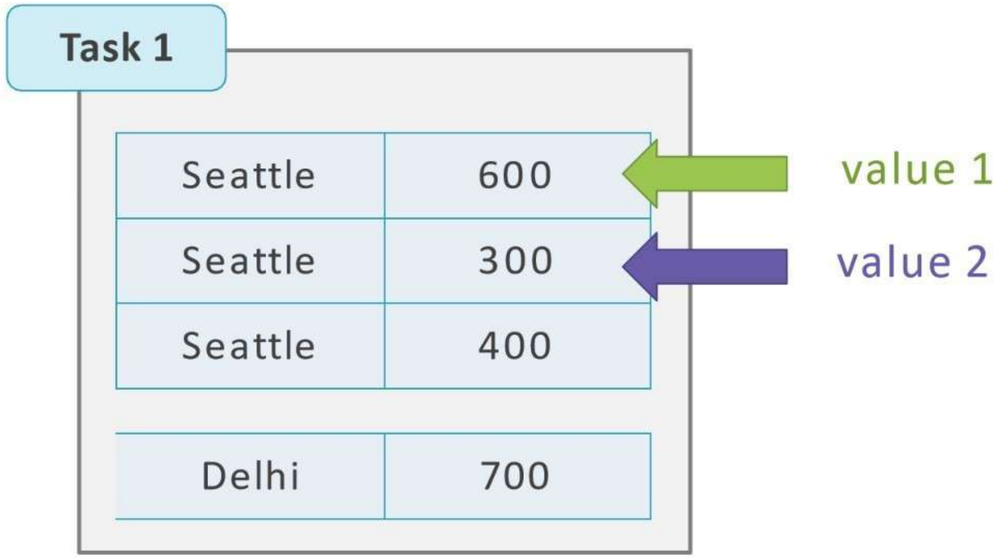
PairRDD <string, integer>

```
cityPairRDD
   .reduceByKey (lambda value1, value2: value1 + value2)
```

Calculate sum of Amount by City

Seattle	600
Seattle	300
Delhi	700
Seattle	400

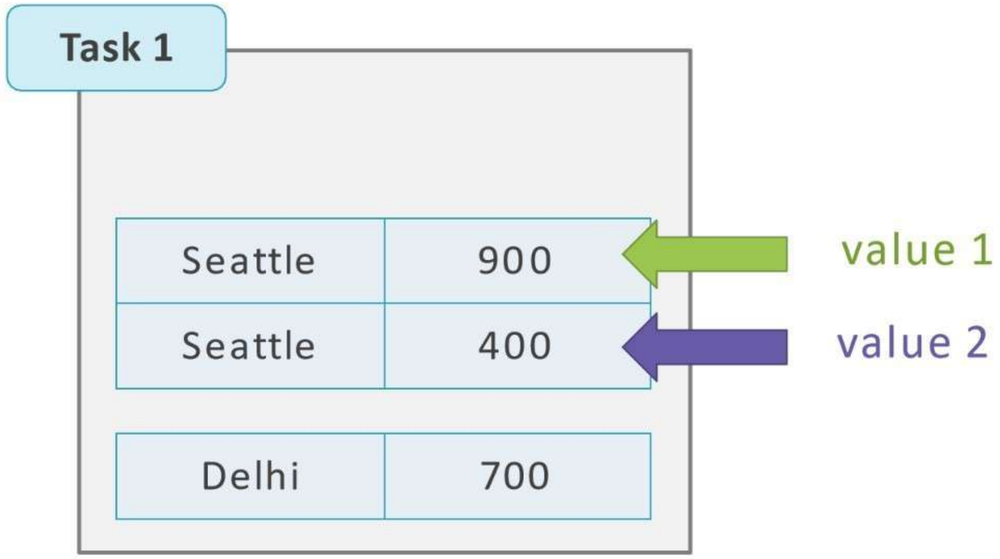
Delhi	900
Delhi	200
Seattle	900



Lambda Function

```
value1, value2: value1 + value2
```

Calculate sum of Amount by City



Delhi	900
Delhi	200
Seattle	900

Lambda Function

```
value1, value2: value1 + value2
```

Calculate sum of Amount by City

Task 1

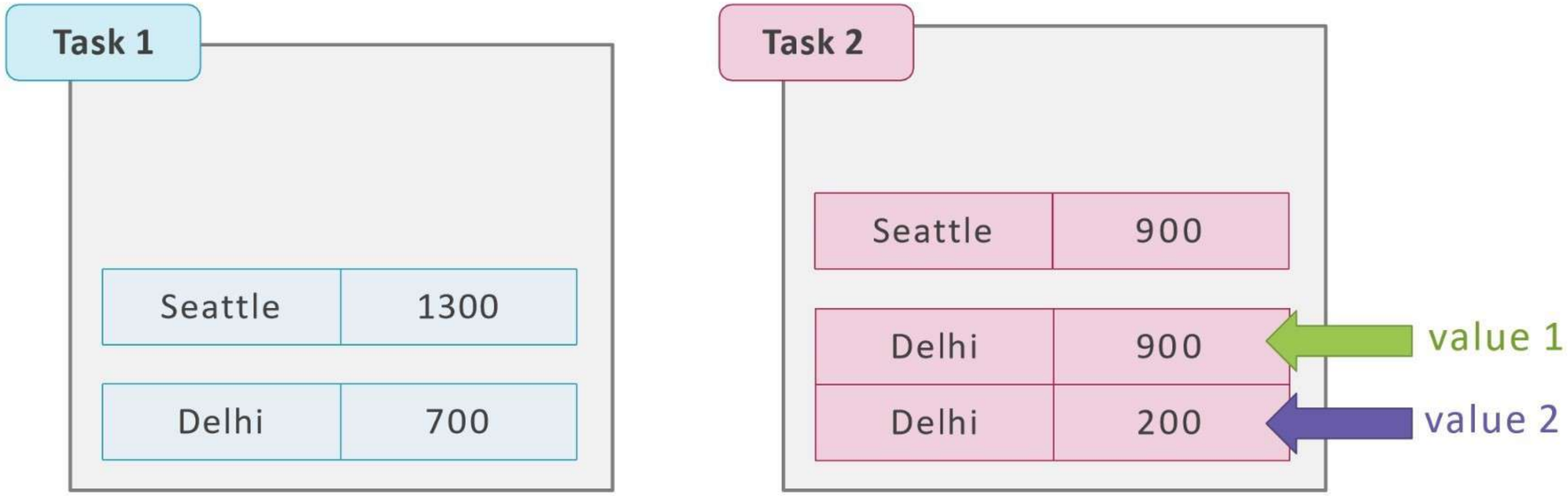
Seattle	1300
Delhi	700

Delhi	900
Delhi	200
Seattle	900

Lambda Function

```
value1, value2: value1 + value2
```

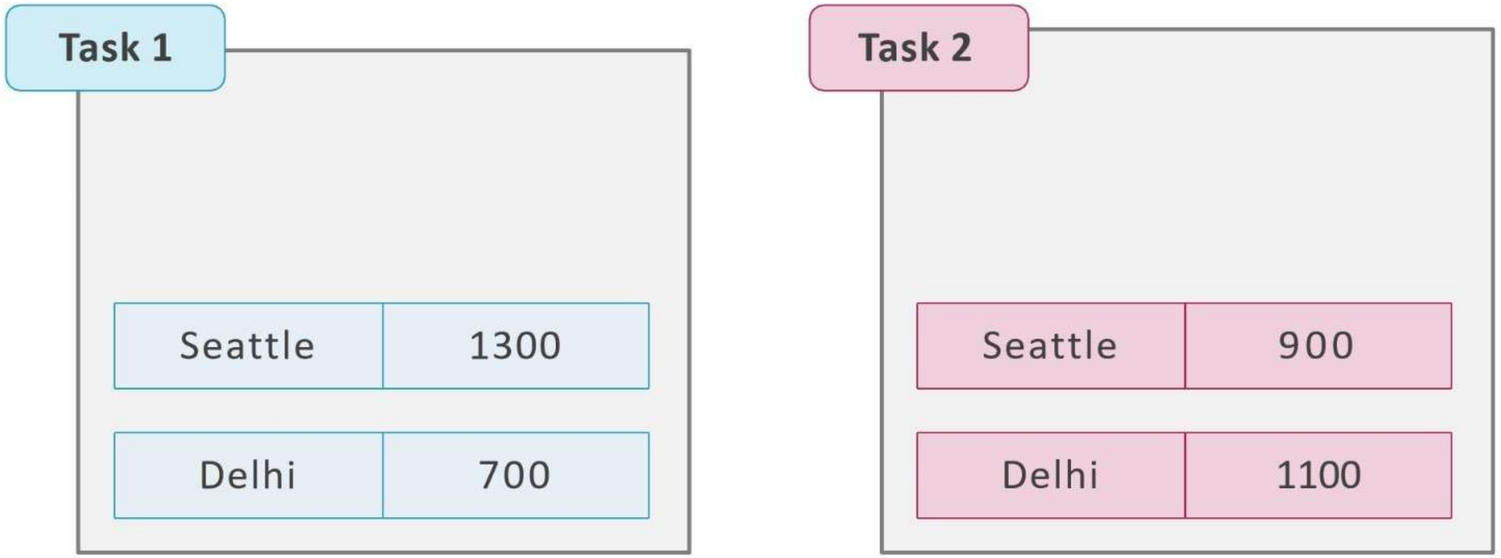
Calculate sum of Amount by City



Lambda Function

```
value1, value2: value1 + value2
```

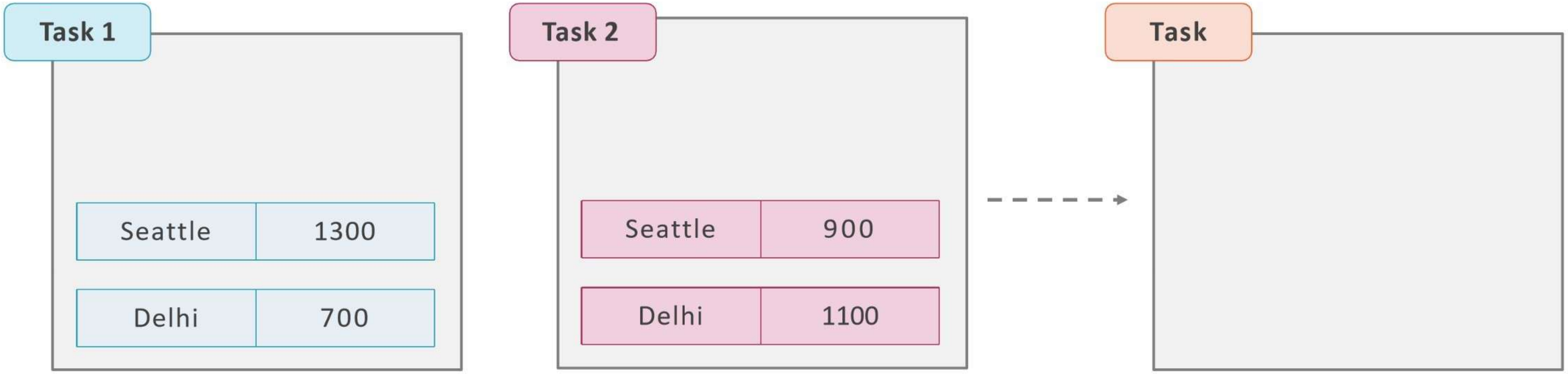
Calculate sum of Amount by City



Lambda Function

```
value1, value2: value1 + value2
```

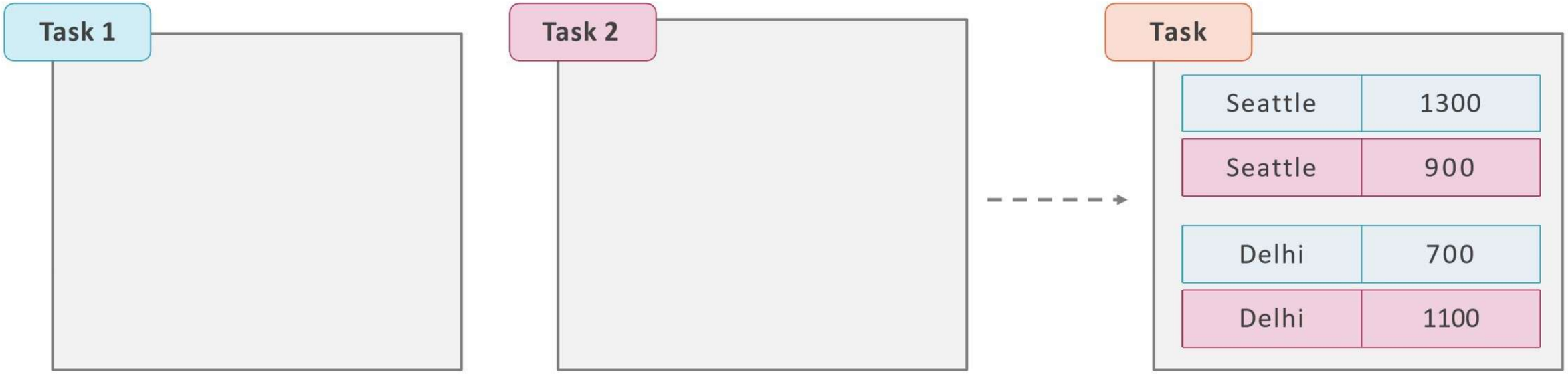
Calculate sum of Amount by City



Lambda Function

```
value1, value2: value1 + value2
```

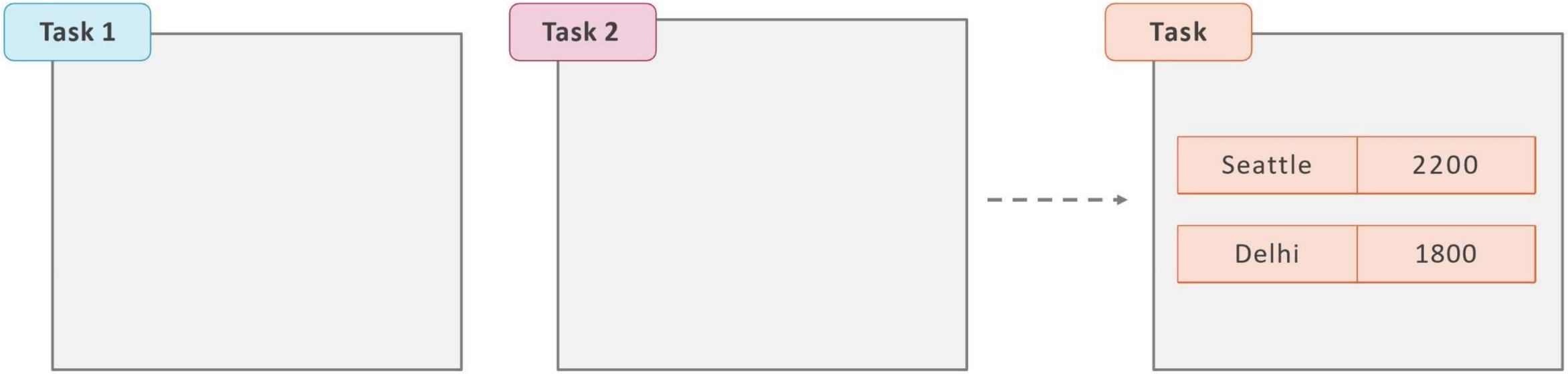
Calculate sum of Amount by City



Lambda Function

```
value1, value2: value1 + value2
```

Calculate sum of Amount by City



Lambda Function

```
value1, value2: value1 + value2
```

Calculate count of records for each Borough

+ Code + Markdown

```
# Create Pair RDD (LocationId as Key, Value as 1)

taxiZonesPairRdd = (
                    taxiZonesWithColsRdd

                        .map(lambda zoneRow: (
                                                zoneRow[1],    # Key   - Borough
                                                1              # Value - 1
                                              )
                             )
                   )

taxiZonesPairRdd.take(10)
```

20]

```
[('EWR', 1),
 ('Queens', 1),
 ('Bronx', 1),
 ('Manhattan', 1),
 ('Staten Island', 1),
 ('Staten Island', 1),
```

```python
# Calculate count of records for each Borough

boroughCountRdd = (
                    taxiZonesPairRdd

                        .reduceByKey( lambda value1, value2: value1 + value2 )
                    )

boroughCountRdd.collect()
```

```
[('Bronx', 43),
 ('Staten Island', 20),
 ('EWR', 1),
 ('Manhattan', 69),
 ('Brooklyn', 61),
 ('Unknown', 2),
 ('Queens', 69)]
```

Let’s explore more operations of Pair RDDs

```python
# Sort the data by Key

(
    taxiZonesPairRdd

        .sortByKey()

        .collect()
)
```

```
[('Bronx', 1),
 ('Bronx', 1),
 ('Bronx', 1),
 ('Bronx', 1),
 ('Bronx', 1),
 ('Bronx', 1),
 ('Bronx', 1),
 ('Bronx', 1),
 ('Bronx', 1),
 ('Bronx', 1),
```

```python
# Only return the keys

(
    taxiZonesPairRdd

        .keys()

        .take(10)
)
```

```
['EWR',
 'Queens',
 'Bronx',
 'Manhattan',
 'Staten Island',
 'Staten Island',
 'Queens',
 'Queens',
 'Queens',
 'Queens']
```

```
# Only return the values

(
    taxiZonesPairRdd

        .values()

        .take(10)
)
```

```
[1, 1, 1, 1, 1, 1, 1, 1, 1, 1]
```

```python
# Return distinct pairs

(
    taxiZonesPairRdd

        .distinct()

        .collect()
)
```

```
[('Bronx', 1),
 ('Staten Island', 1),
 ('Queens', 1),
 ('EWR', 1),
 ('Manhattan', 1),
 ('Brooklyn', 1),
```

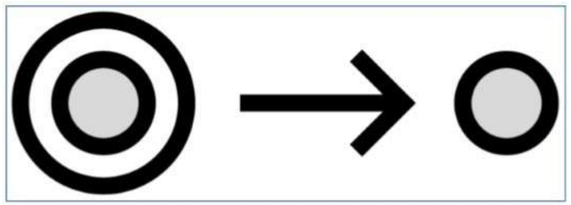
Applying Operations on RDDs

RDD Operations

Transformation

Action

Transformation Operation



Function that produces new RDD from existing RDDs

Transformation operations help in building a Lineage Graph

Examples:

- Read a file
- Convert sales amount from INR to USD
- Filter records with sales amount greater than 1000

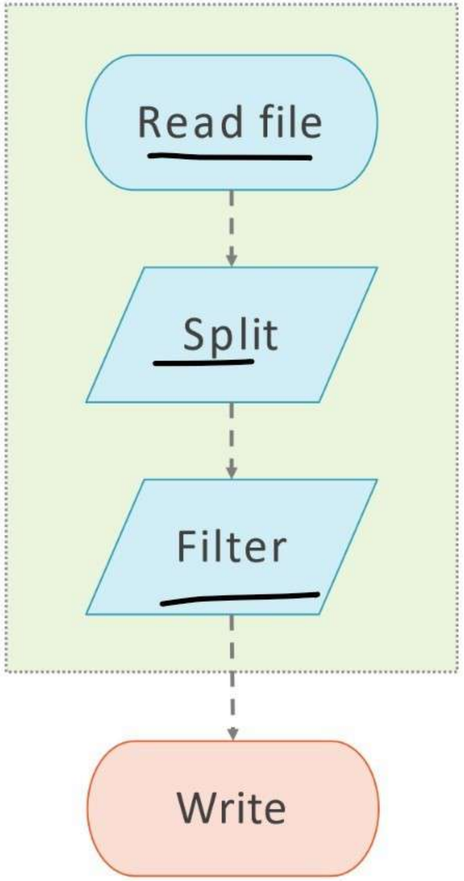
Transformations are Lazy operations, which are only executed when an Action operation is applied

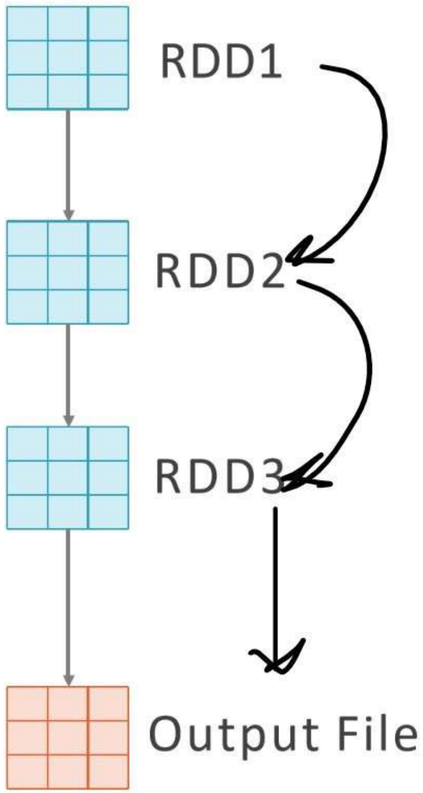
1. Read file1.csv from Storage → T
2. Split data by comma → T
3. Filter where City='Delhi' → T
4. Write processed data to storage → A

Id, City, Amount
1,Seattle, 600
2, London, 300
3, Delhi, 700
4, Seattle, 400
5, Paris, 900
6, Delhi, 200
7, Seattle, 900

Lineage Graph

Built using transformation operations

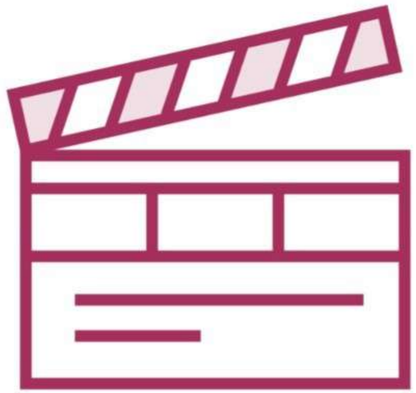




Action operation

collect()
take()
count()

Action Operation



Returns result of RDD computations

Triggers execution of transformations using Lineage Graph

Examples:

- Write the output to storage
- Print the output of operations
- Display the count

In [1]:
```
import findspark
findspark.init()
findspark.find()
```

Out[1]:
```
'E:\\Local d\\Zartab\\Apache Spark Installation\\spark-3.3.2-bin-hadoop3'
```

In [2]:
```
from pyspark.sql import SparkSession
from pyspark.sql.types import *

spark = (
            SparkSession
                .builder
                .appName("RDDOps")
                .master("local[4]")
                .getOrCreate()
        )
```

In [5]:
```
spark

sc=spark.sparkContext
```

In [6]:
```
# Create RDD using parallelize

numbersRDD= sc.parallelize([1,2,3,4,5,6])
```

In [7]:
```
spark
```

Out[7]:

SparkSession - in-memory

SparkContext

Spark UI

Version

`v3.3.2`

Master

`local[4]`

Spark 3.3.2 | Jobs | Stages | Storage | Environment | Executors | SQL / DataFrame

Spark Jobs (?)

User: VICTUS
Total Uptime: 16 s
Scheduling Mode: FIFO

▸ Event Timeline

```
In [8]: numbersRDD.collect()

Out[8]: [1, 2, 3, 4, 5, 6]
```

```
In [ ]:
```

Spark 3.3.2 | Jobs | Stages | Storage | Environment | Executors | SQL / DataFrame | **RDDOps** application UI

Spark Jobs (?)

User: VICTUS
Total Uptime: 1.4 min
Scheduling Mode: FIFO
Completed Jobs: 1

▸ Event Timeline

▾ Completed Jobs (1)

Page: 1 — 1 Pages. Jump to 1 . Show 100 items in a page. Go

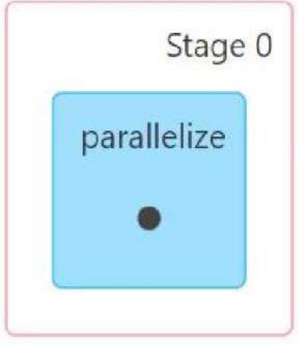
Job Id ▾	Description	Submitted	Duration	Stages: Succeeded/Total	Tasks (for all stages): Succeeded/Total
0	collect at C:\Users\VICTUS\AppData\Local\Temp\ipykernel_24236\2661653226.py:1 collect at C:\Users\VICTUS\AppData\Local\Temp\ipykernel_24236\2661653226.py:1	2023/05/18 06:58:17	0.6 s	1/1	4/4

Page: 1 — 1 Pages. Jump to 1 . Show 100 items in a page. Go

Details for Job 0

Status: SUCCEEDED
Submitted: 2023/05/18 06:58:17
Duration: 0.6 s
Completed Stages: 1

▸ Event Timeline
▾ DAG Visualization



▾ Completed Stages (1)

Page: 1

1 Pages. Jump to 1 . Show 100

Stage Id ▾	Description	Submitted	Duration	Tasks: Succeeded/Total	Input	Output

Using Narrow Transformations

Transformation Operations

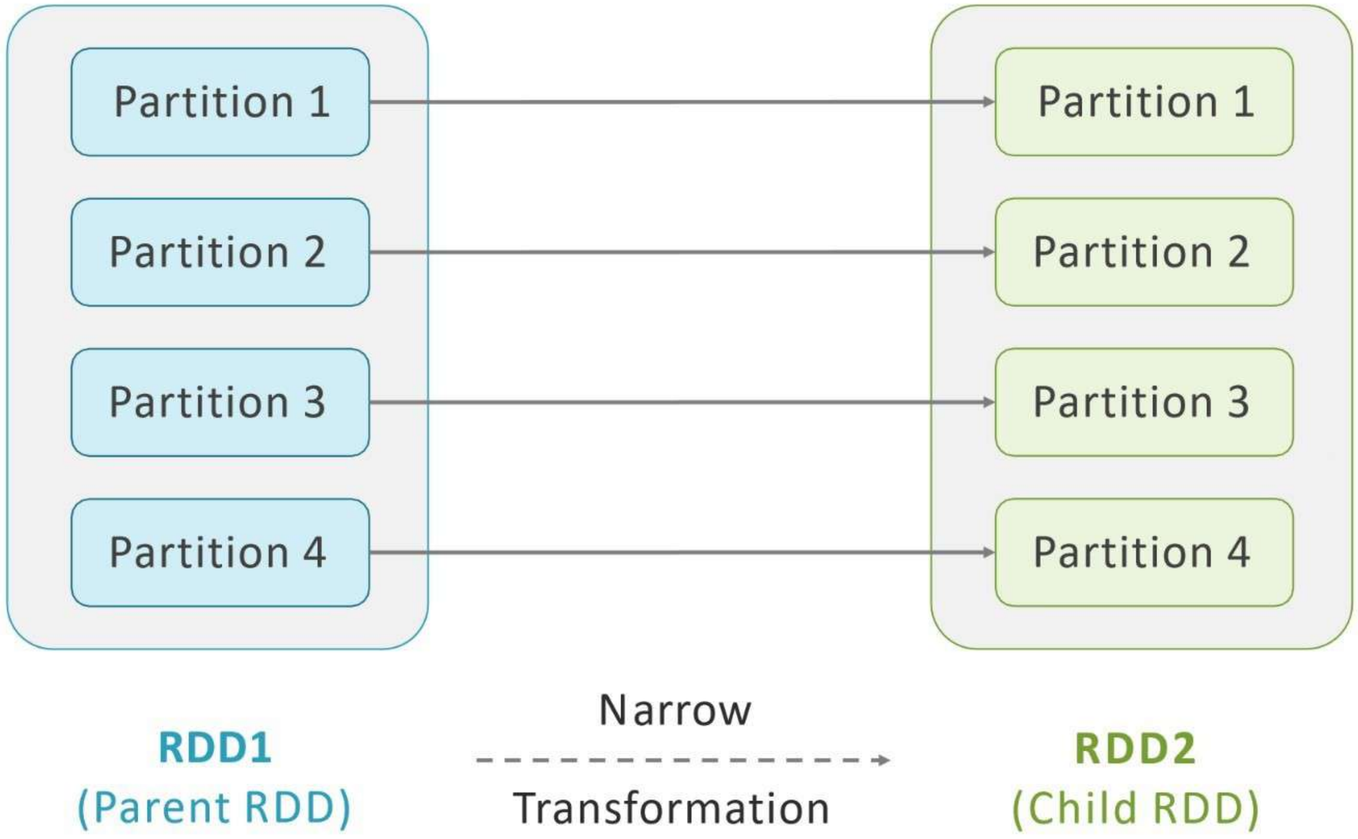
Narrow Dependency Transformation

Wide Dependency Transformation

A transformation where each input partition is used at-most once to produce output partitions

Narrow Transformations

Each input partition is used **at-most once** to produce output partitions

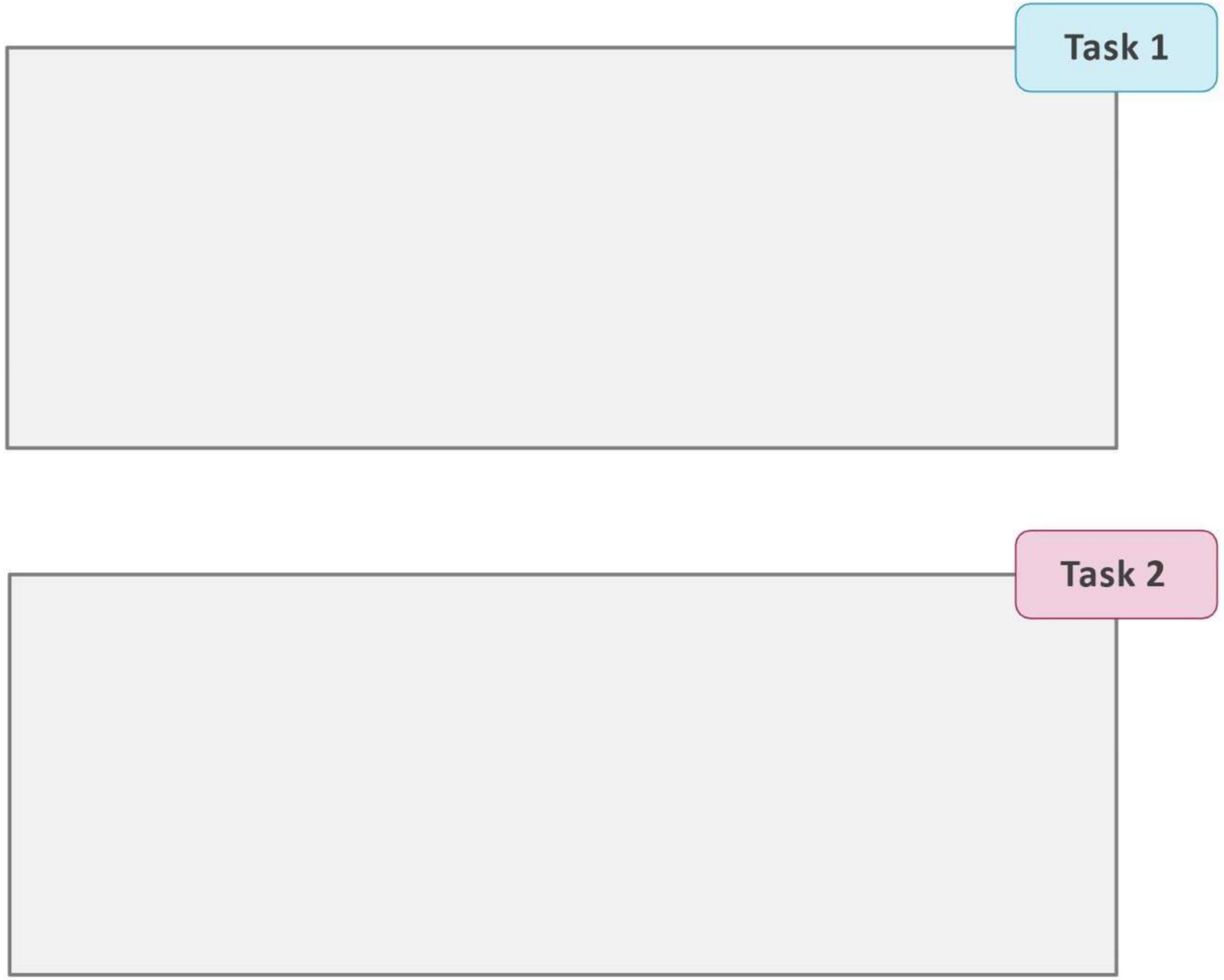


Filter & Map operations are examples of this

Id, City, Amount
1,Seattle, 600
2, London, 300
3, Delhi, 700
4, Seattle, 400
5, Paris, 900
6, Delhi, 200
7, Seattle, 900

Read file, split by comma

RDD 1



Id, City, Amount
1,Seattle, 600
2, London, 300
3, Delhi, 700
4, Seattle, 400
5, Paris, 900
6, Delhi, 200
7, Seattle, 900

Read file, split by comma

RDD 1

→

Filter City = 'Seattle'

RDD 2

Task 1

1	Seattle	600
2	London	300
3	Delhi	700

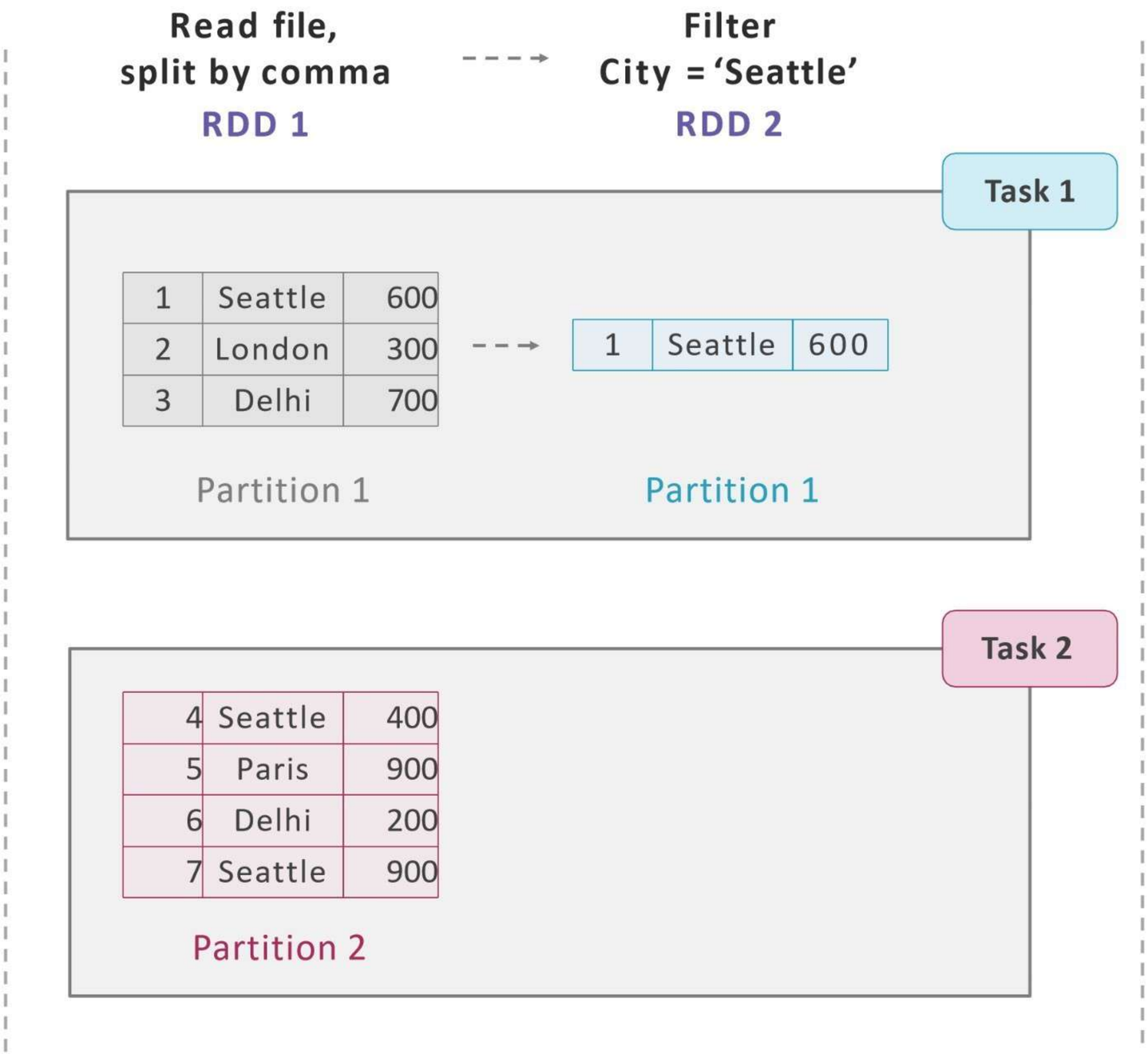
Partition 1

Task 2

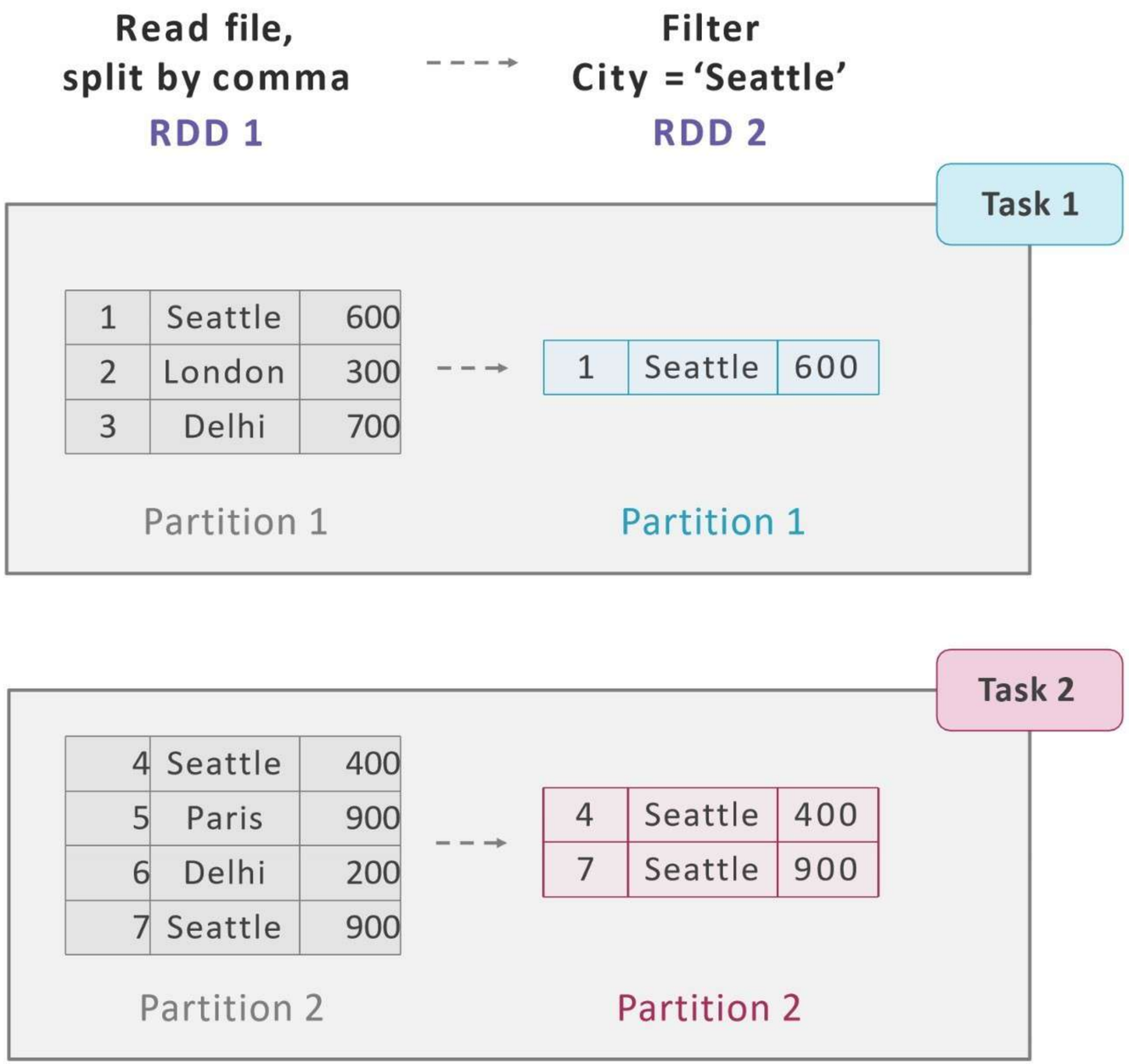
4	Seattle	400
5	Paris	900
6	Delhi	200
7	Seattle	900

Partition 2

Id, City, Amount
1,Seattle, 600
2, London, 300
3, Delhi, 700
4, Seattle, 400
5, Paris, 900
6, Delhi, 200
7, Seattle, 900



Id, City, Amount
1,Seattle, 600
2, London, 300
3, Delhi, 700
4, Seattle, 400
5, Paris, 900
6, Delhi, 200
7, Seattle, 900



Filter is a Narrow Transformation

Id, City, Amount
1,Seattle, 600
2, London, 300
3, Delhi, 700
4, Seattle, 400
5, Paris, 900
6, Delhi, 200
7, Seattle, 900

Read file, split by comma

RDD 1

→

Filter City = 'Seattle'

RDD 2

Task 1

1	Seattle	600
2	London	300
3	Delhi	700

Partition 1

→

1	Seattle	600

Partition 1

Task 2

4	Seattle	400
5	Paris	900
6	Delhi	200
7	Seattle	900

Partition 2

→

4	Seattle	400
7	Seattle	900

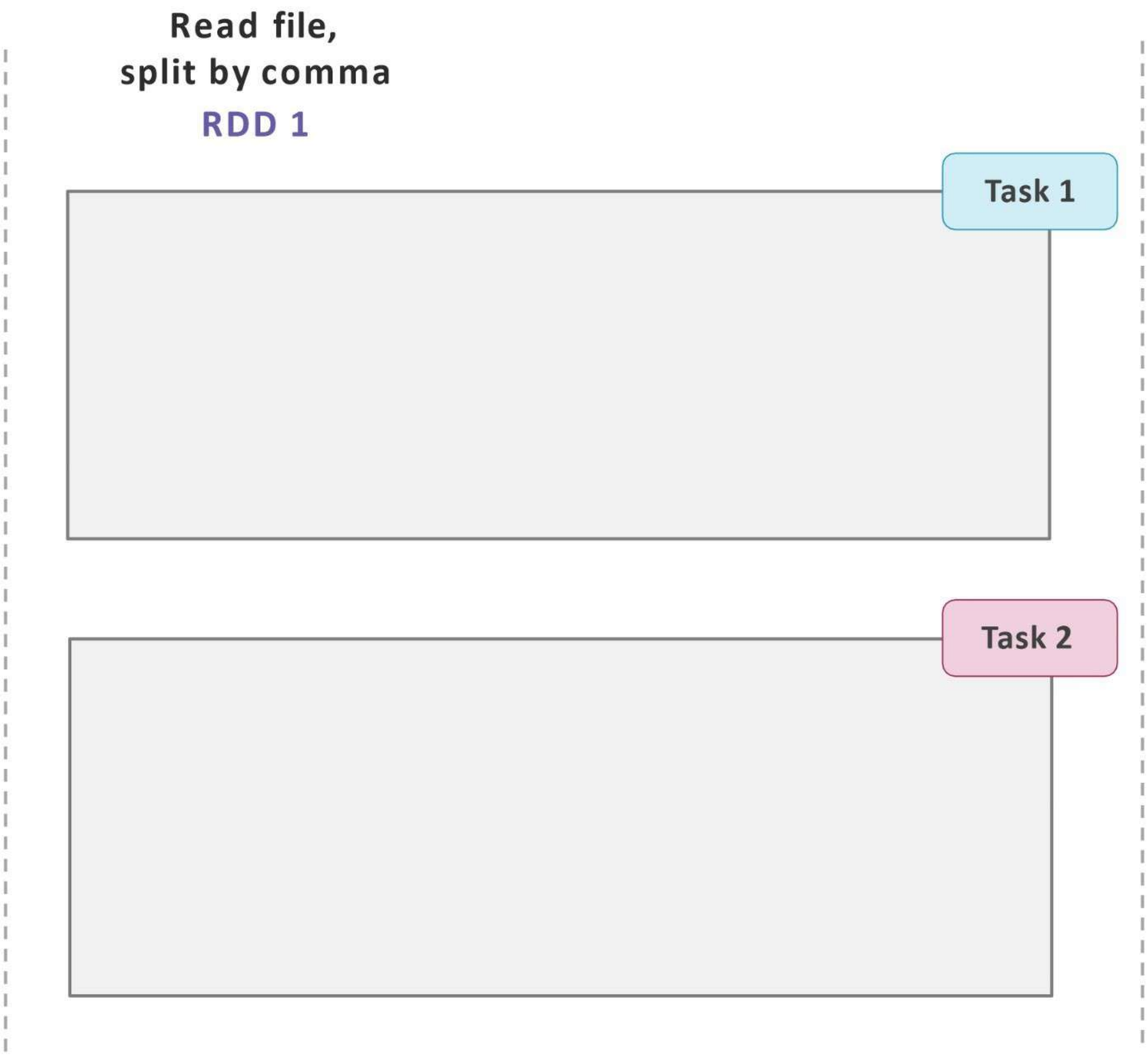
Partition 2

Filter is a Narrow Transformation

1	Seattle	600

4	Seattle	400
7	Seattle	900

Id, City, Amount
1,Seattle, 600
2, London, 300
3, Delhi, 700
4, Seattle, 400
5, Paris, 900
6, Delhi, 200
7, Seattle, 900



Id, City, Amount
1,Seattle, 600
2, London, 300
3, Delhi, 700
4, Seattle, 400
5, Paris, 900
6, Delhi, 200
7, Seattle, 900

Read file, split by comma

RDD 1

→

Use map() to derive annual amount

RDD 2

Task 1

1	Seattle	600
2	London	300
3	Delhi	700

Partition 1

Task 2

4	Seattle	400
5	Paris	900
6	Delhi	200
7	Seattle	900

Partition 2

Id, City, Amount
1,Seattle, 600
2, London, 300
3, Delhi, 700
4, Seattle, 400
5, Paris, 900
6, Delhi, 200
7, Seattle, 900

Read file, split by comma

RDD 1

→

Use map() to derive annual amount

RDD 2

Task 1

1	Seattle	600
2	London	300
3	Delhi	700

Partition 1

→

1	Seattle	600	7200
2	London	300	3600
3	Delhi	700	8400

Partition 1

Task 2

4	Seattle	400
5	Paris	900
6	Delhi	200
7	Seattle	900

Partition 2

→

4	Seattle	400	4800
5	Paris	900	10800
6	Delhi	200	2400
7	Seattle	900	10800

Partition 2

Map is a Narrow Transformation

Id, City, Amount
1,Seattle, 600
2, London, 300
3, Delhi, 700
4, Seattle, 400
5, Paris, 900
6, Delhi, 200
7, Seattle, 900

Read file, split by comma → **Use map() to derive annual amount**

RDD 1 → **RDD 2**

Task 1

1	Seattle	600
2	London	300
3	Delhi	700

Partition 1

1	Seattle	600	7200
2	London	300	3600
3	Delhi	700	8400

Partition 1

Task 2

4	Seattle	400
5	Paris	900
6	Delhi	200
7	Seattle	900

Partition 2

4	Seattle	400	4800
5	Paris	900	10800
6	Delhi	200	2400
7	Seattle	900	10800

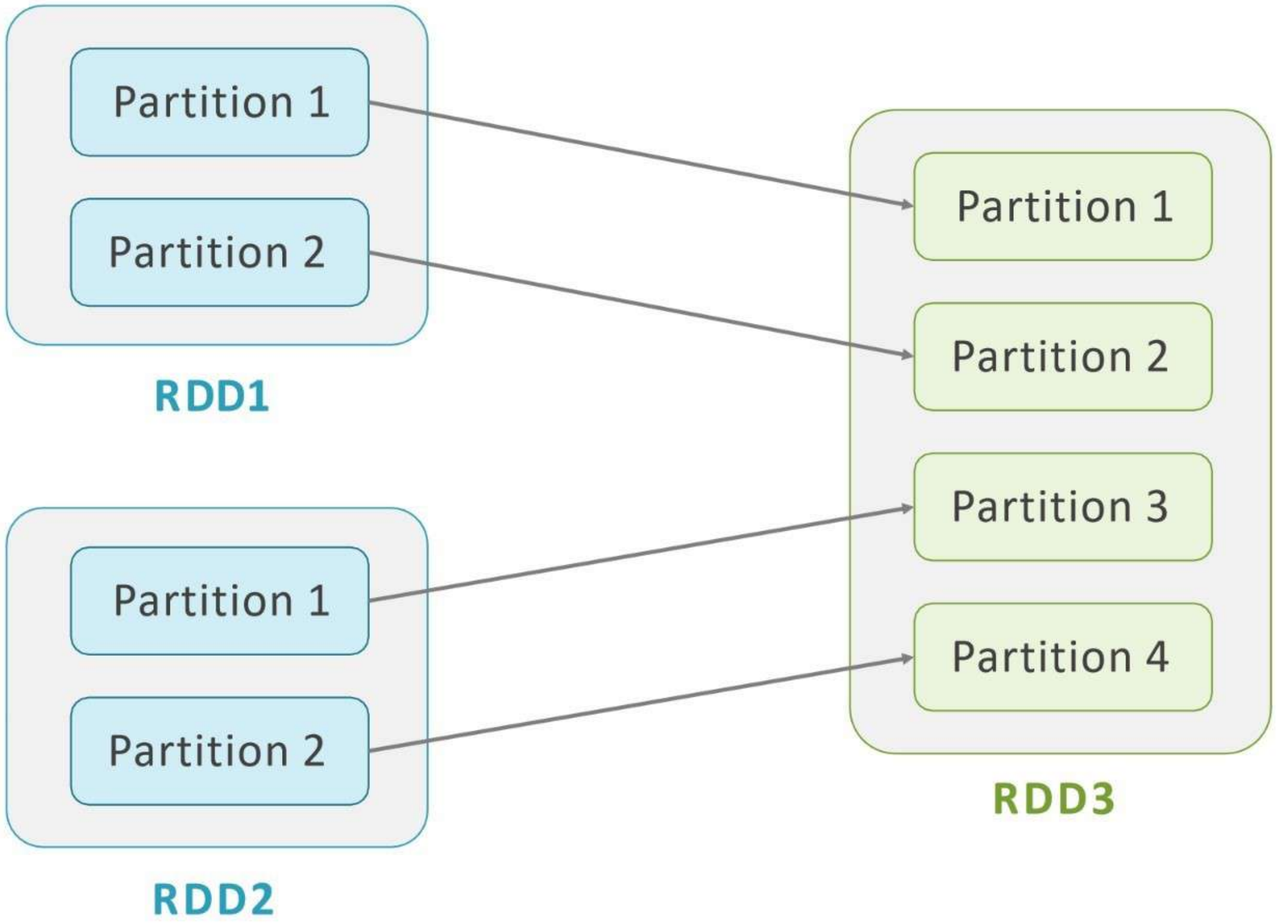
Partition 2

Map is a Narrow Transformation

1	Seattle	600	7200
2	London	300	3600
3	Delhi	700	8400

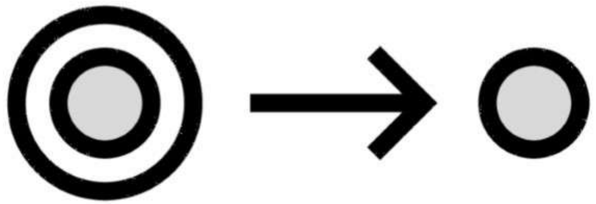
4	Seattle	400	4800
5	Paris	900	10800
6	Delhi	200	2400
7	Seattle	900	10800

Each input partition is used **at-most once** to produce output partitions



Union operation is an example of this

Narrow Transformation



Extremely fast

No data movement between partitions / No shuffling

Examples

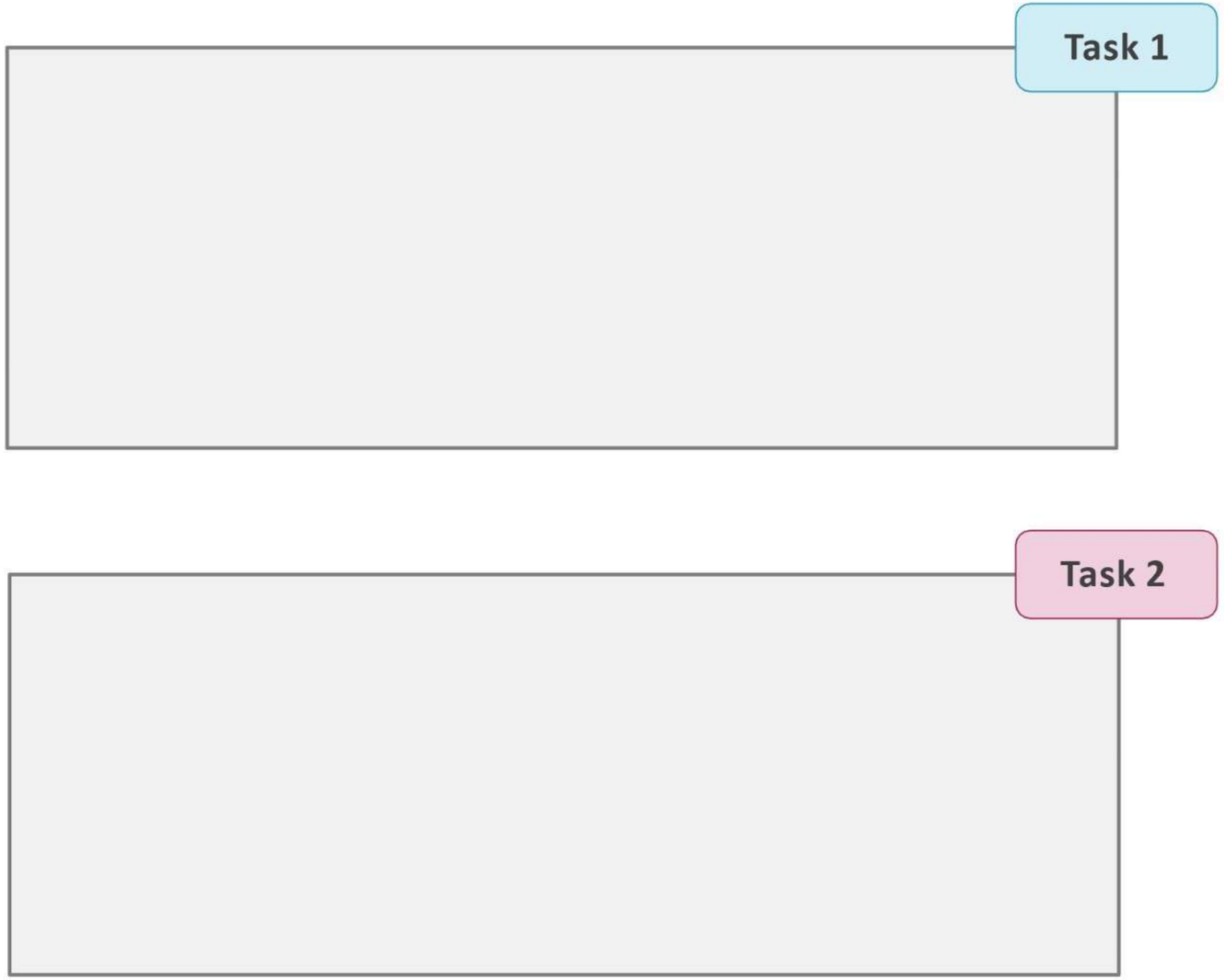
- Filter, Map, FlatMap, MapPartition, Sample, Union etc.

Wide Transformations and Data Shuffling

Id, City, Amount
1,Seattle, 600
2, London, 300
3, Delhi, 700
4, Seattle, 400
5, Paris, 900
6, Delhi, 200
7, Seattle, 900

Read file, split by comma

RDD 1



Id, City, Amount
1,Seattle, 600
2, London, 300
3, Delhi, 700
4, Seattle, 400
5, Paris, 900
6, Delhi, 200
7, Seattle, 900

Read file, split by comma

RDD 1

→

Group by City, calculate count

RDD 2

Task 1

1	Seattle	600
2	London	300
3	Delhi	700

Partition 1

Task 2

4	Seattle	400
5	Paris	900
6	Delhi	200
7	Seattle	900

Partition 2

Id, City, Amount
1,Seattle, 600
2, London, 300
3, Delhi, 700
4, Seattle, 400
5, Paris, 900
6, Delhi, 200
7, Seattle, 900

Read file, split by comma

RDD 1

→

Group by City, calculate count

RDD 2

Task 1

1	Seattle	600
2	London	300
3	Delhi	700

Partition 1

→

Seattle	1
London	1
Delhi	1

Partition 1

Task 2

4	Seattle	400
5	Paris	900
6	Delhi	200
7	Seattle	900

Partition 2

→

Seattle	2
Paris	1
Delhi	1

Partition 2

?

Id, City, Amount
1,Seattle, 600
2, London, 300
3, Delhi, 700
4, Seattle, 400
5, Paris, 900
6, Delhi, 200
7, Seattle, 900

Read file, split by comma

RDD 1

→

Group by City, calculate count

RDD 2

Task 1

1	Seattle	600
2	London	300
3	Delhi	700

Partition 1

→

Seattle	1
London	1
Delhi	1

Partition 1

Task 2

4	Seattle	400
5	Paris	900
6	Delhi	200
7	Seattle	900

Partition 2

→

Seattle	2
Paris	1
Delhi	1

Partition 2

Incorrect Output!

Data is not completely grouped

Seattle	1
London	1
Delhi	1

Seattle	2
Paris	1
Delhi	1

Transformation Operations

Narrow Dependency Transformation

Wide Dependency Transformation

Wide Transformation is a two-step process and requires shuffling of data

Id, City, Amount
1,Seattle, 600
2, London, 300
3, Delhi, 700
4, Seattle, 400
5, Paris, 900
6, Delhi, 200
7, Seattle, 900

Read file, get City &Amount

RDD 1

Seattle	600
London	300
Delhi	700

Partition 1

Seattle	400
Paris	900
Delhi	200
Seattle	900

Partition 2

Read file,
get City &Amount

RDD 1

Group by City, calculate sum of Amount

RDD 2

Step 1– Group: Shuffle / Exchange

Seattle	600
London	300
Delhi	700

Partition 1

Seattle	400
Paris	900
Delhi	200
Seattle	900

Partition 2

Shuffle Write

Read file,
get City &Amount
RDD 1

Group by City, calculate sum of Amount
RDD 2

Step 1– Group: Shuffle / Exchange

Seattle	600
London	300
Delhi	700

Partition 1

Seattle	400
Paris	900
Delhi	200
Seattle	900

Partition 2

Seattle	600
London	300
Delhi	700

Shuffle Block 1

Shuffle Write

Read file, get City &Amount

RDD 1

Group by City, calculate sum of Amount

RDD 2

Step 1– Group: Shuffle / Exchange

Seattle	600
London	300
Delhi	700

Partition 1

Seattle	400
Paris	900
Delhi	200
Seattle	900

Partition 2

Seattle	600
London	300
Delhi	700

Shuffle Block 1

Seattle	1300
Paris	900
Delhi	200

Shuffle Block 2

Shuffle Write → Shuffle Read

Read file, get City &Amount

RDD 1

Group by City, calculate sum of Amount

RDD 2

Step 1– Group: Shuffle / Exchange

Seattle	600
London	300
Delhi	700

Partition 1

Seattle	400
Paris	900
Delhi	200
Seattle	900

Partition 2

Seattle	600
London	300
Delhi	700

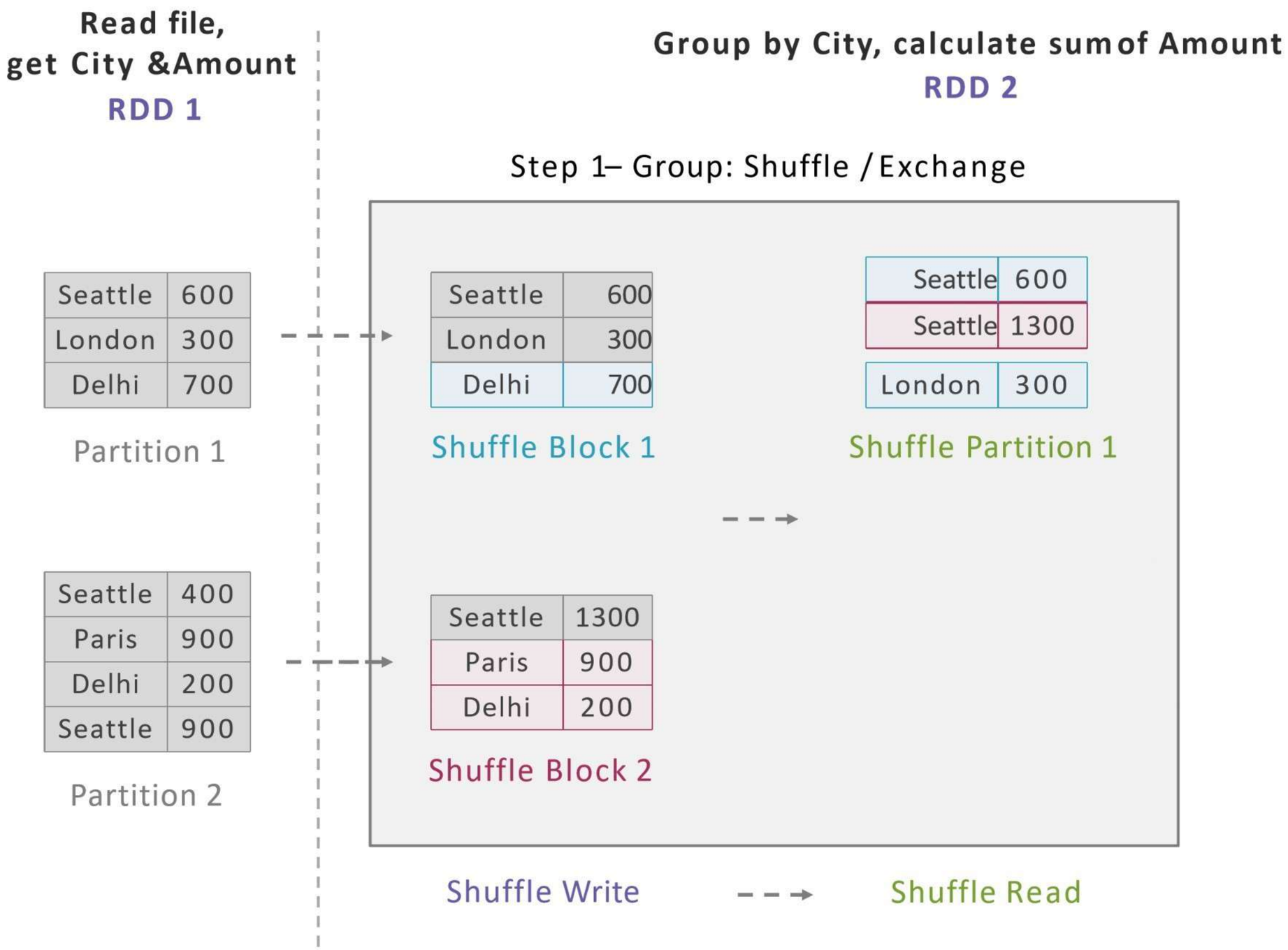
Shuffle Block 1

Seattle	1300
Paris	900
Delhi	200

Shuffle Block 2

Seattle	600
Seattle	1300

Shuffle Write → Shuffle Read


Read file,
get City &Amount
RDD 1
Group by City, calculate sum of Amount
RDD 2
Step 1– Group: Shuffle / Exchange
Seattle 600
London 300
Delhi 700
Partition 1
Seattle 400
Paris 900
Delhi 200
Seattle 900
Partition 2
Seattle 600
London 300
Delhi 700
Shuffle Block 1
Seattle 1300
Paris 900
Delhi 200
Shuffle Block 2
Seattle 600
Seattle 1300
London 300
Shuffle Partition 1
Shuffle Write
Shuffle Read

Read file, get City &Amount

RDD 1

Group by City, calculate sum of Amount

RDD 2

Step 1– Group: Shuffle / Exchange

Step 2 – Sum

Seattle	600
London	300
Delhi	700

Partition 1

Seattle	400
Paris	900
Delhi	200
Seattle	900

Partition 2

Seattle	600
London	300
Delhi	700

Shuffle Block 1

Seattle	1300
Paris	900
Delhi	200

Shuffle Block 2

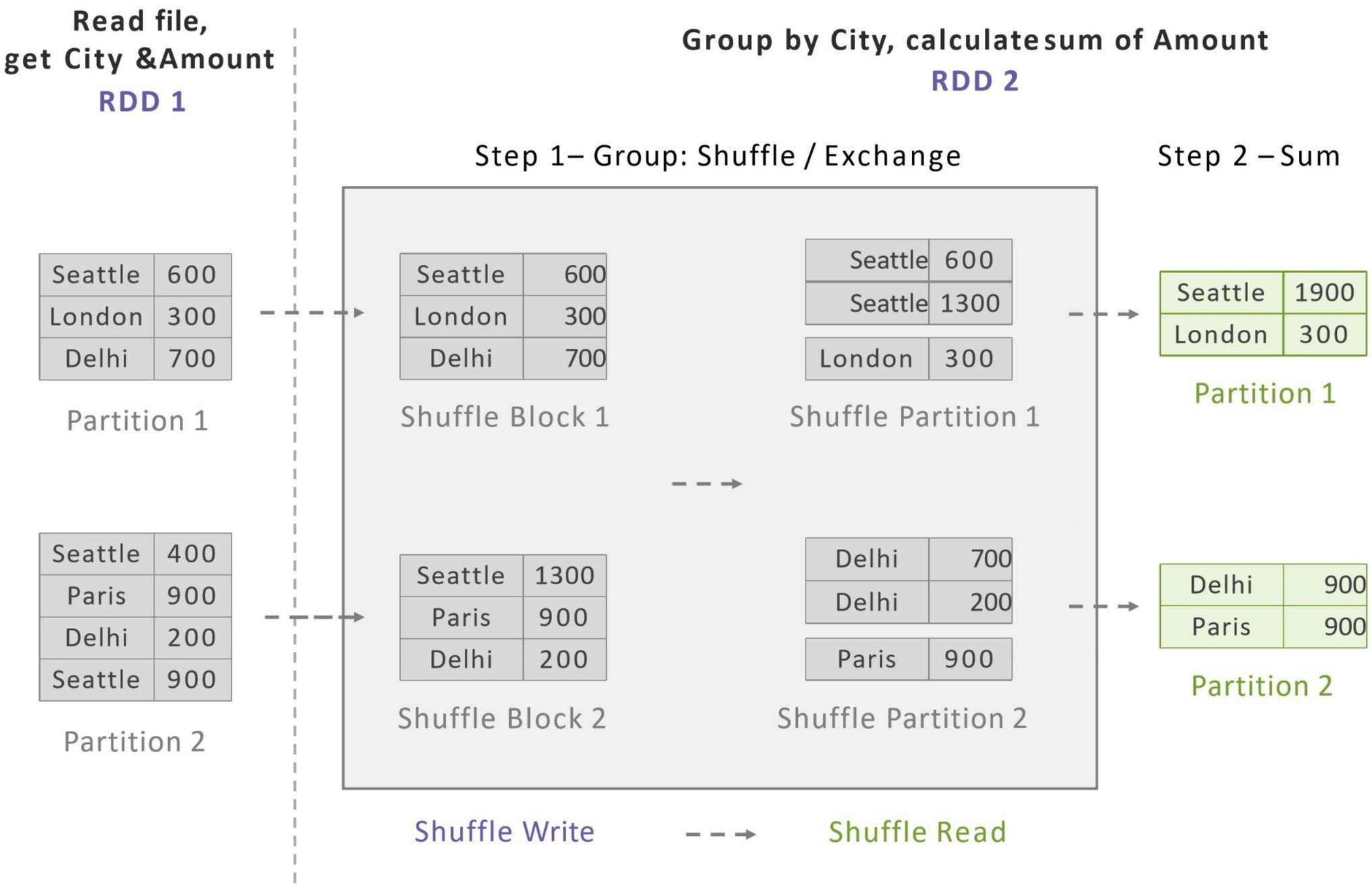
Seattle	600
Seattle	1300
London	300

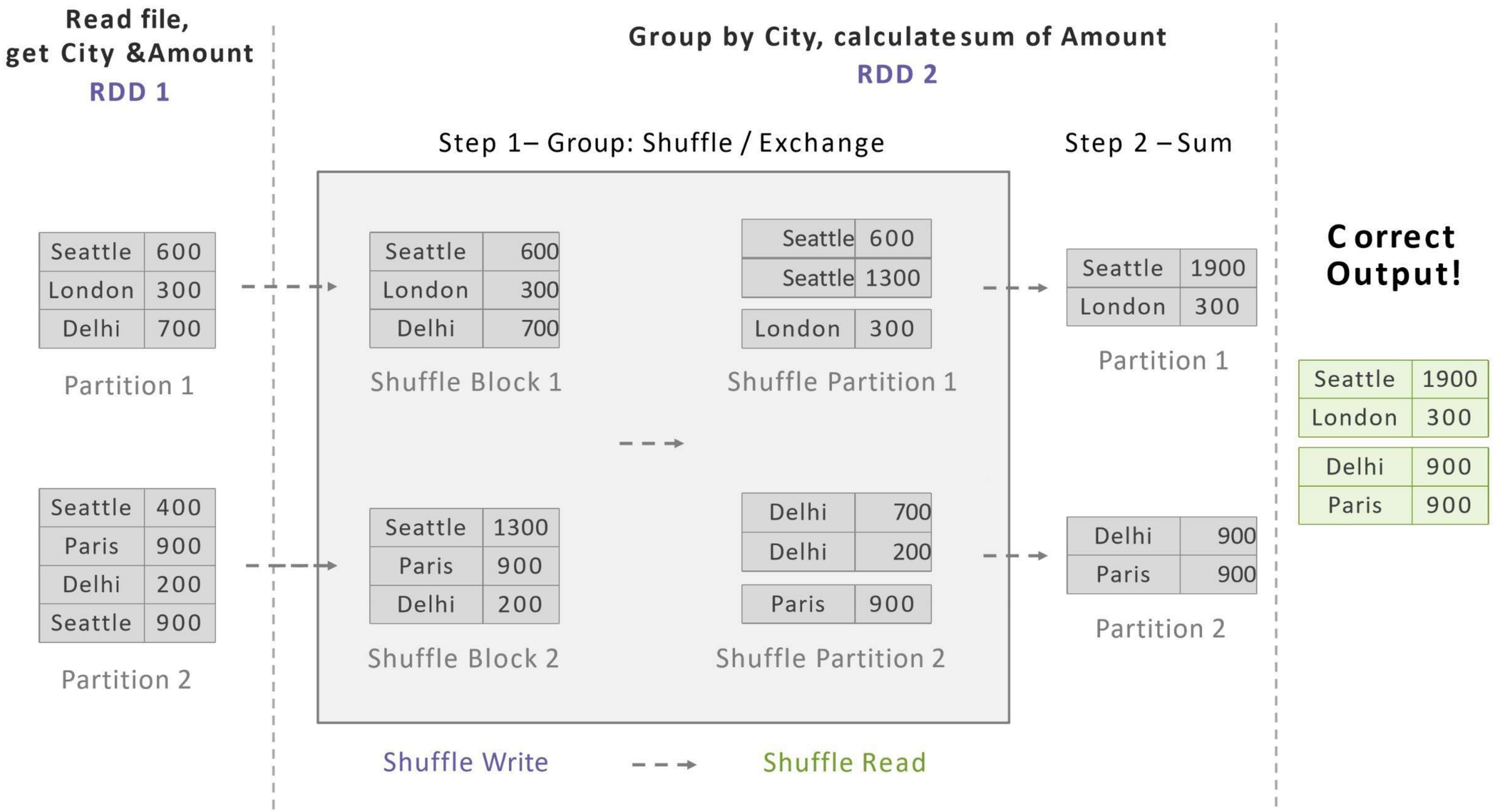
Shuffle Partition 1

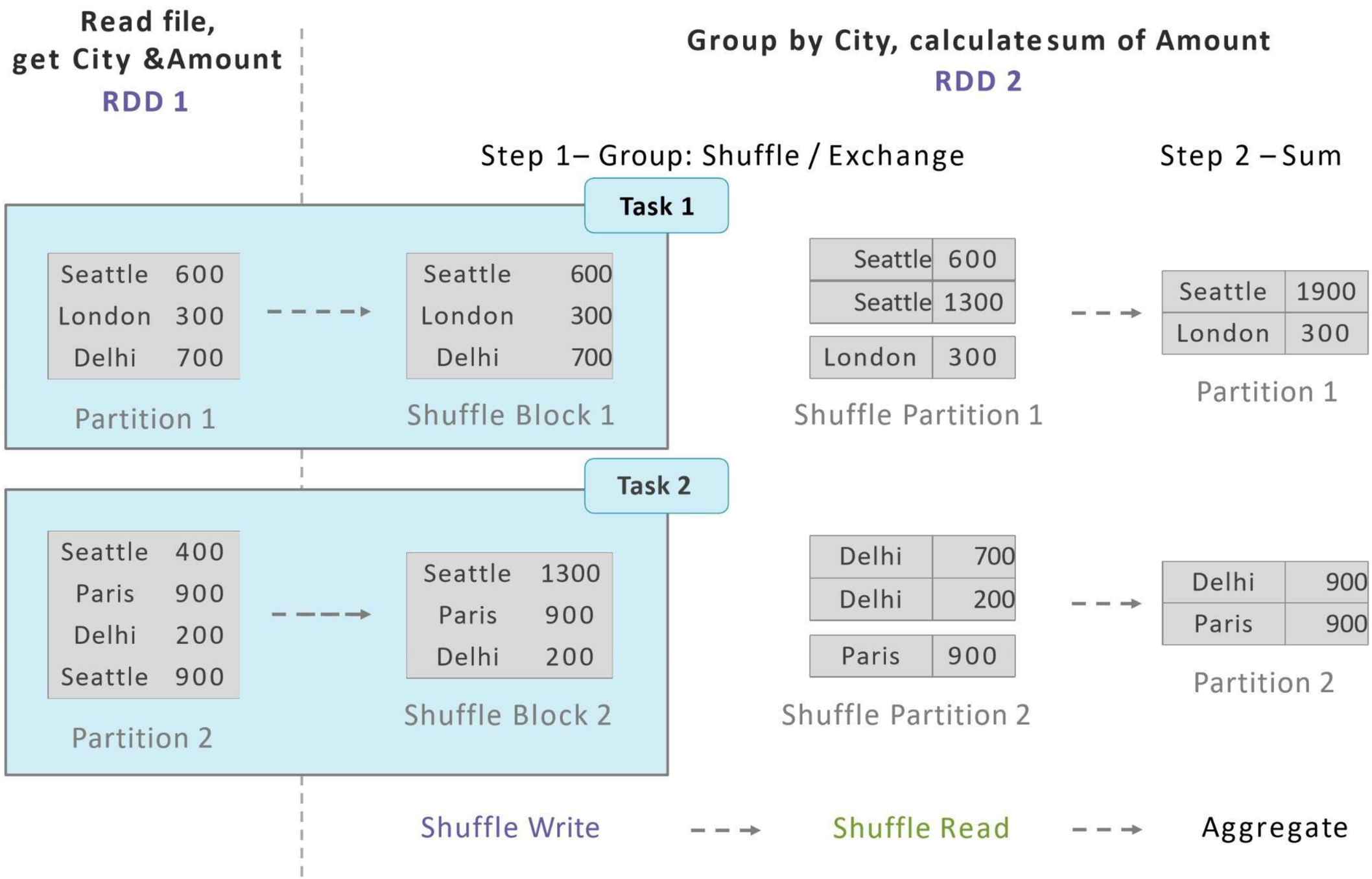
Delhi	700
Delhi	200
Paris	900

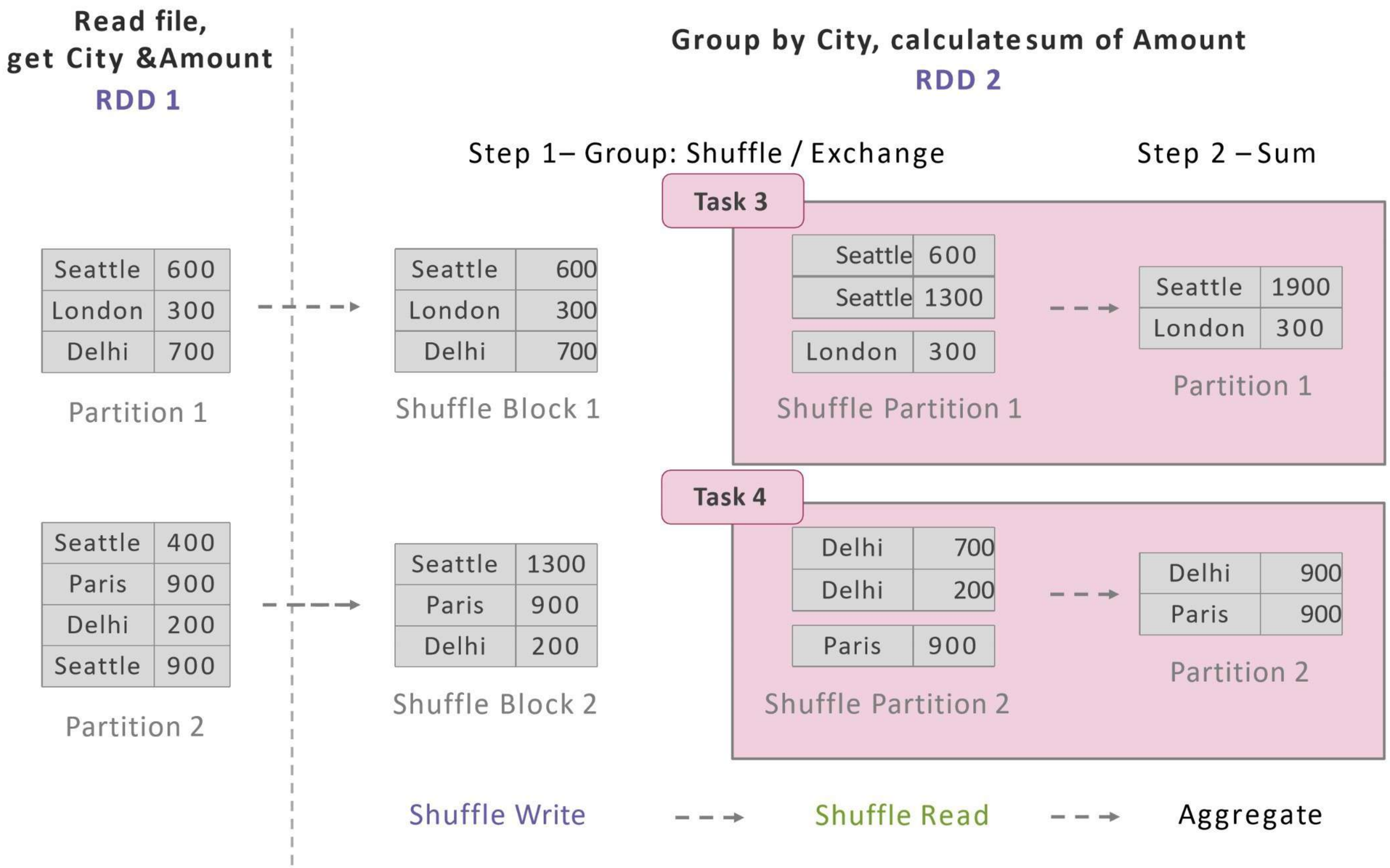
Shuffle Partition 2

Shuffle Write → Shuffle Read


Read file,
get City &Amount
RDD 1
Group by City, calculate sum of Amount
RDD 2
Step 1– Group: Shuffle / Exchange
Step 2 – Sum
Seattle 600
London 300
Delhi 700
Partition 1
Seattle 400
Paris 900
Delhi 200
Seattle 900
Partition 2
Seattle 600
London 300
Delhi 700
Shuffle Block 1
Seattle 1300
Paris 900
Delhi 200
Shuffle Block 2
Seattle 600
Seattle 1300
London 300
Shuffle Partition 1
Delhi 700
Delhi 200
Paris 900
Shuffle Partition 2
Seattle 1900
London 300
Partition 1
Delhi 900
Paris 900
Partition 2
Shuffle Write
Shuffle Read

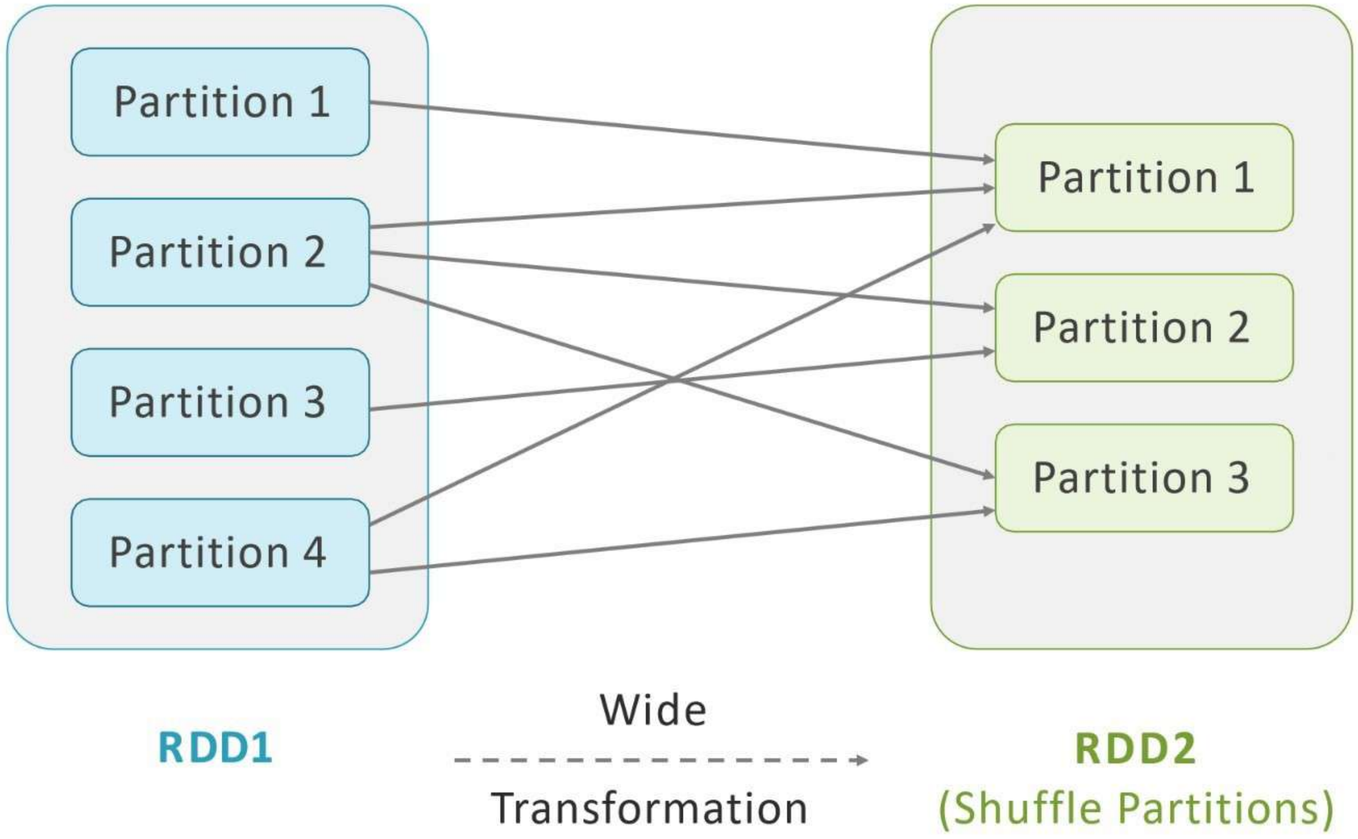

Read file,
get City &Amount
RDD 1
Group by City, calculate sum of Amount
RDD 2
Step 1– Group: Shuffle / Exchange
Step 2 – Sum
Seattle 600
London 300
Delhi 700
Partition 1
Seattle 400
Paris 900
Delhi 200
Seattle 900
Partition 2
Seattle 600
London 300
Delhi 700
Shuffle Block 1
Seattle 1300
Paris 900
Delhi 200
Shuffle Block 2
Seattle 600
Seattle 1300
London 300
Shuffle Partition 1
Delhi 700
Delhi 200
Paris 900
Shuffle Partition 2
Seattle 1900
London 300
Partition 1
Delhi 900
Paris 900
Partition 2
Correct Output!
Seattle 1900
London 300
Delhi 900
Paris 900
Shuffle Write
Shuffle Read


Read file,
get City &Amount
RDD 1
Group by City, calculate sum of Amount
RDD 2
Step 1– Group: Shuffle / Exchange
Step 2 – Sum
Task 1
Seattle 600
London 300
Delhi 700
Partition 1
Seattle 600
London 300
Delhi 700
Shuffle Block 1
Seattle 600
Seattle 1300
London 300
Shuffle Partition 1
Seattle 1900
London 300
Partition 1
Task 2
Seattle 400
Paris 900
Delhi 200
Seattle 900
Partition 2
Seattle 1300
Paris 900
Delhi 200
Shuffle Block 2
Delhi 700
Delhi 200
Paris 900
Shuffle Partition 2
Delhi 900
Paris 900
Partition 2
Shuffle Write
Shuffle Read
Aggregate


Read file,
get City &Amount
RDD 1
Group by City, calculate sum of Amount
RDD 2
Step 1– Group: Shuffle / Exchange
Step 2 – Sum
Task 3
Seattle 600
London 300
Delhi 700
Partition 1
Seattle 600
London 300
Delhi 700
Shuffle Block 1
Seattle 600
Seattle 1300
London 300
Shuffle Partition 1
Seattle 1900
London 300
Partition 1
Task 4
Seattle 400
Paris 900
Delhi 200
Seattle 900
Partition 2
Seattle 1300
Paris 900
Delhi 200
Shuffle Block 2
Delhi 700
Delhi 200
Paris 900
Shuffle Partition 2
Delhi 900
Paris 900
Partition 2
Shuffle Write
Shuffle Read
Aggregate

In Wide Transformation, one input partition might be used multiple times to produce output partitions

One input partition might be used multiple times to produce output partitions



ReduceByKey & Distinct operations are examples of this

Read file,
get City &Amount
RDD 1

Get distinct cities
RDD 2

Step 1– Group: Shuffle / Exchange

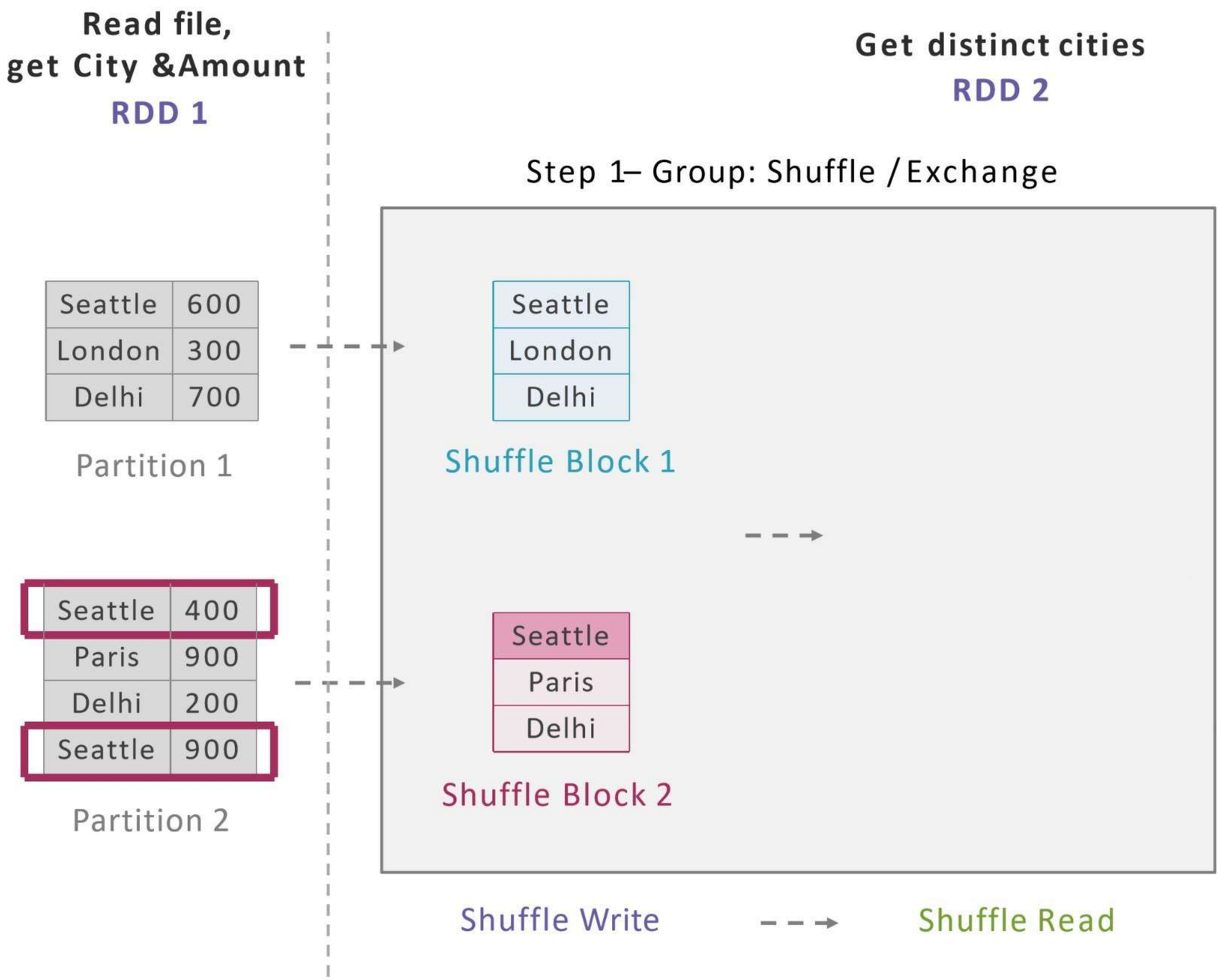
Seattle	600
London	300
Delhi	700

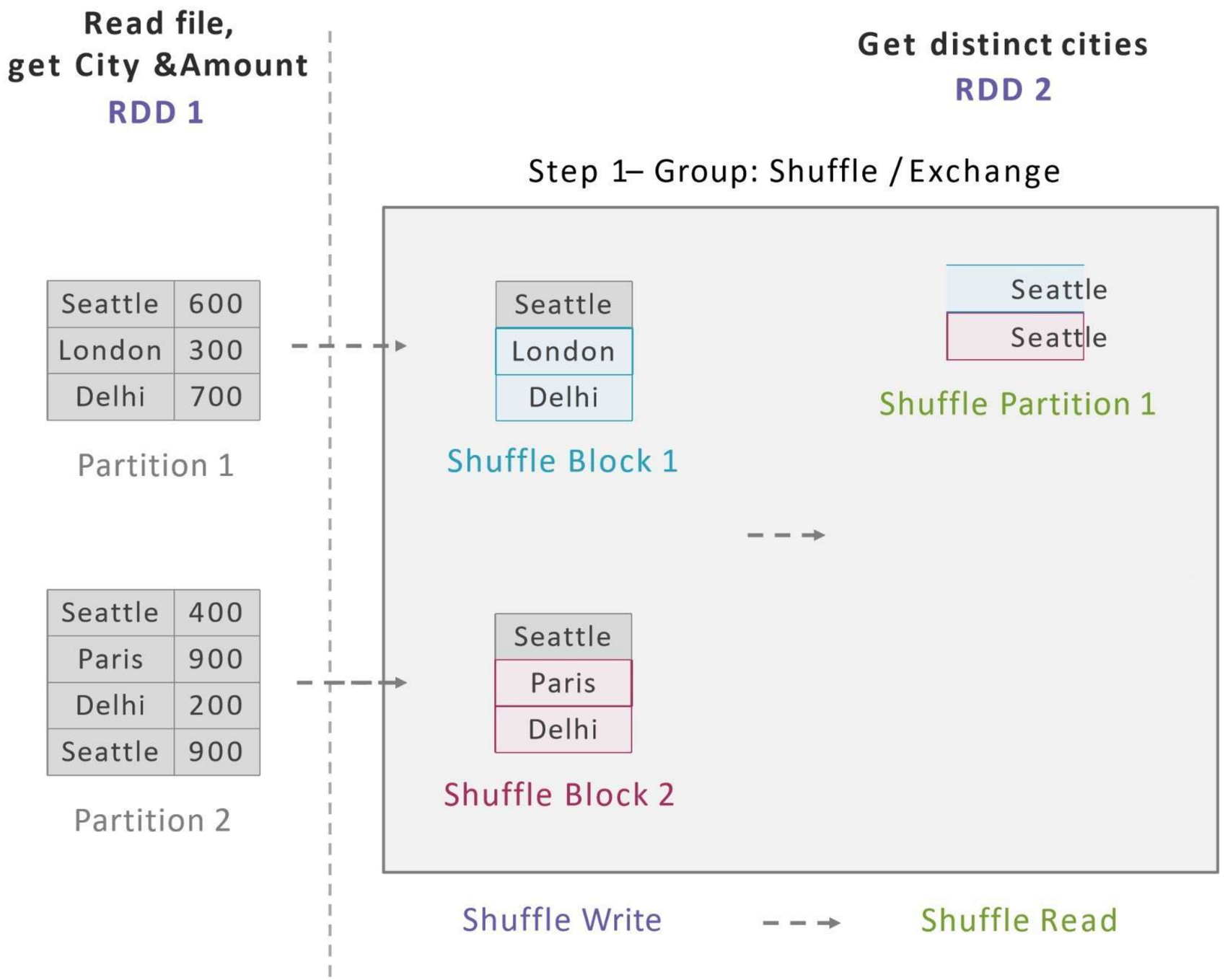
Partition 1

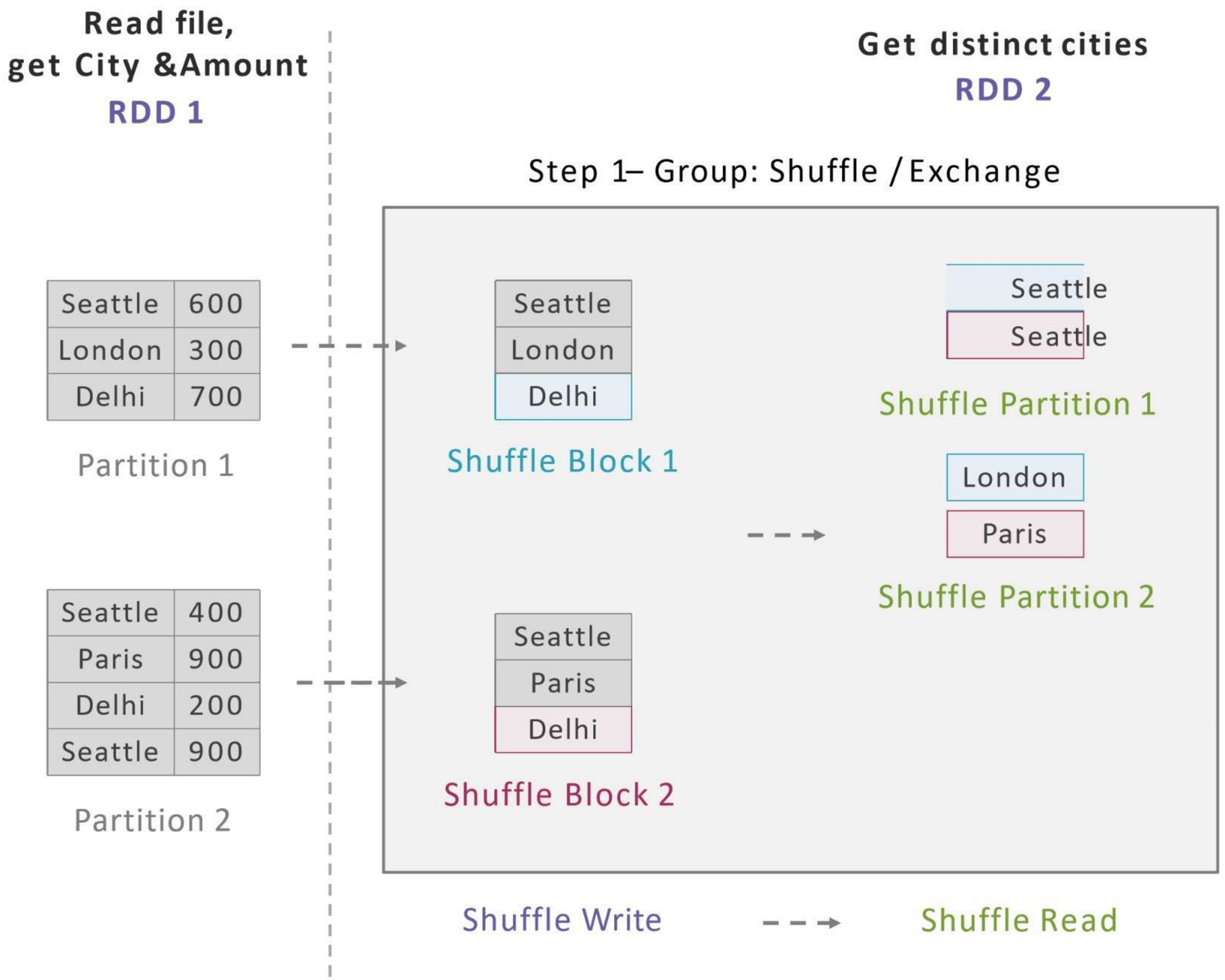
Seattle	400
Paris	900
Delhi	200
Seattle	900

Partition 2

Shuffle Write


Read file,
get City &Amount
RDD 1
Get distinct cities
RDD 2
Step 1– Group: Shuffle / Exchange
Seattle 600
London 300
Delhi 700
Partition 1
Seattle 400
Paris 900
Delhi 200
Seattle 900
Partition 2
Seattle
London
Delhi
Shuffle Block 1
Seattle
Paris
Delhi
Shuffle Block 2
Shuffle Write
Shuffle Read


Read file,
get City &Amount
RDD 1
Get distinct cities
RDD 2
Step 1– Group: Shuffle / Exchange
Seattle 600
London 300
Delhi 700
Partition 1
Seattle 400
Paris 900
Delhi 200
Seattle 900
Partition 2
Seattle
London
Delhi
Shuffle Block 1
Seattle
Paris
Delhi
Shuffle Block 2
Seattle
Seattle
Shuffle Partition 1
Shuffle Write
Shuffle Read


Read file,
get City &Amount
RDD 1
Get distinct cities
RDD 2
Step 1– Group: Shuffle / Exchange
Seattle 600
London 300
Delhi 700
Partition 1
Seattle 400
Paris 900
Delhi 200
Seattle 900
Partition 2
Seattle
London
Delhi
Shuffle Block 1
Seattle
Paris
Delhi
Shuffle Block 2
Seattle
Seattle
Shuffle Partition 1
London
Paris
Shuffle Partition 2
Shuffle Write
Shuffle Read

Read file, get City &Amount

RDD 1

Get distinct cities

RDD 2

Step 1– Group: Shuffle / Exchange

Step 2 – ~~Sum~~ Distinct

Seattle	600
London	300
Delhi	700

Partition 1

Seattle	400
Paris	900
Delhi	200
Seattle	900

Partition 2

Seattle
London
Delhi

Shuffle Block 1

Seattle
Paris
Delhi

Shuffle Block 2

Seattle
Seattle

Shuffle Partition 1

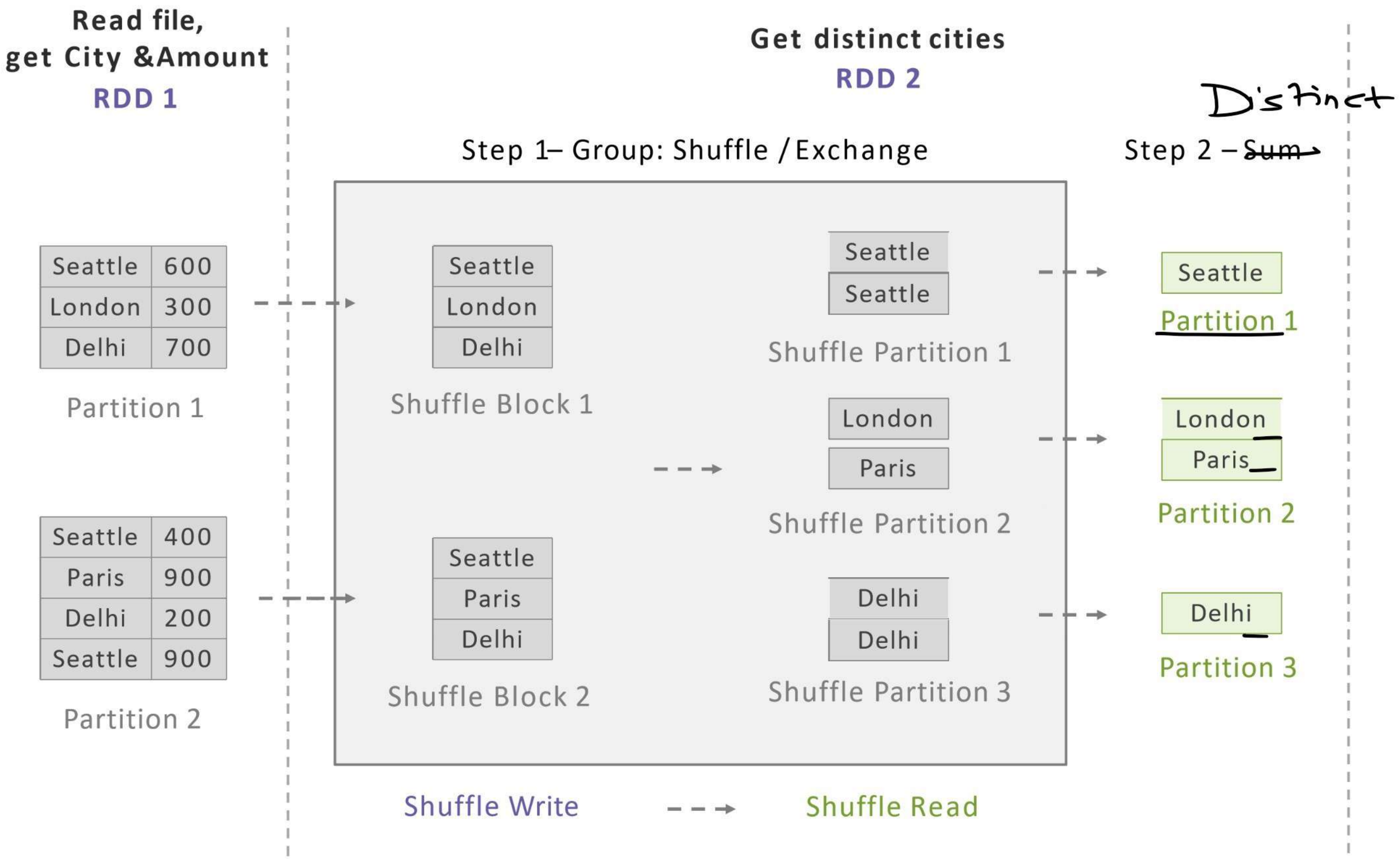
London
Paris

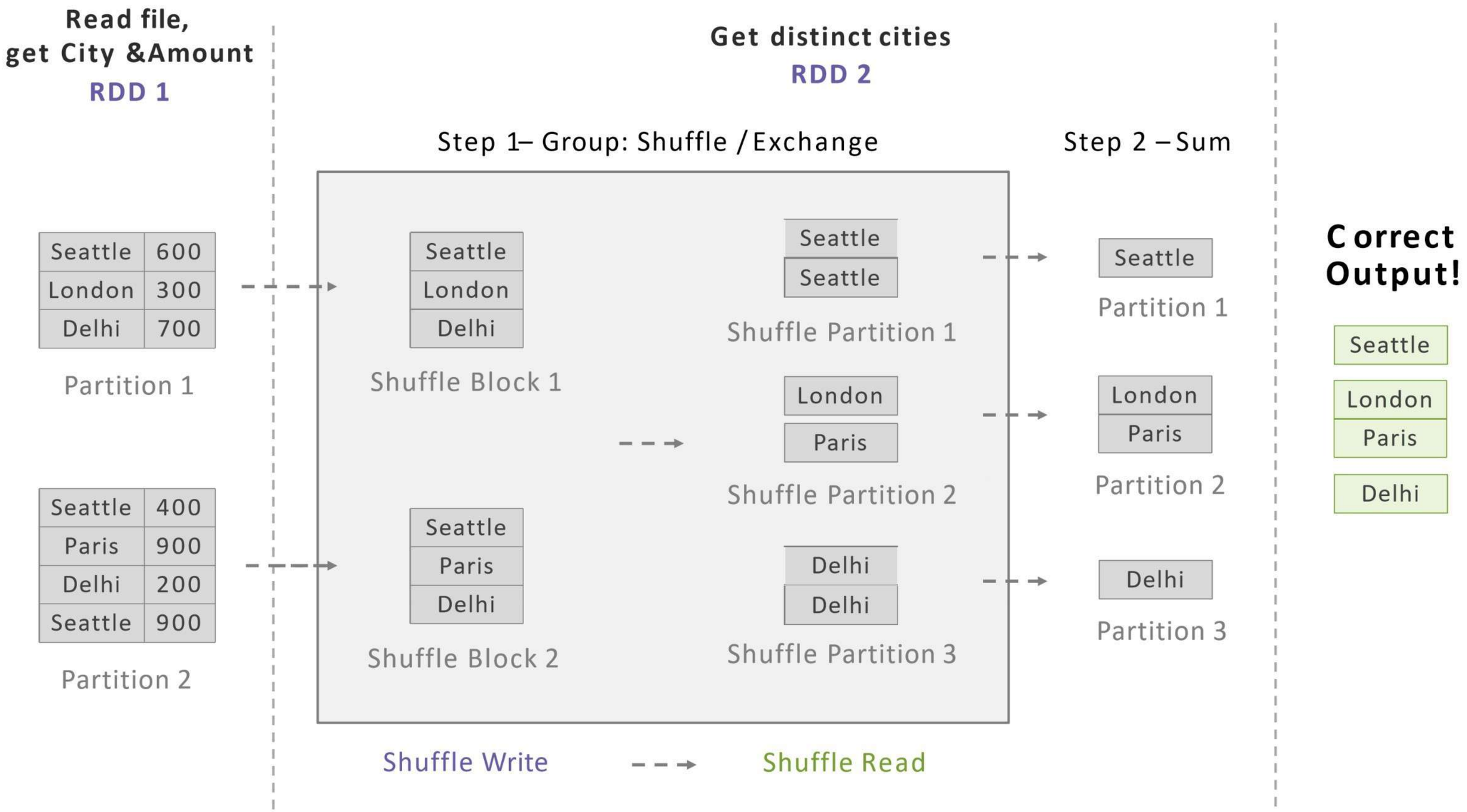
Shuffle Partition 2

Delhi
Delhi

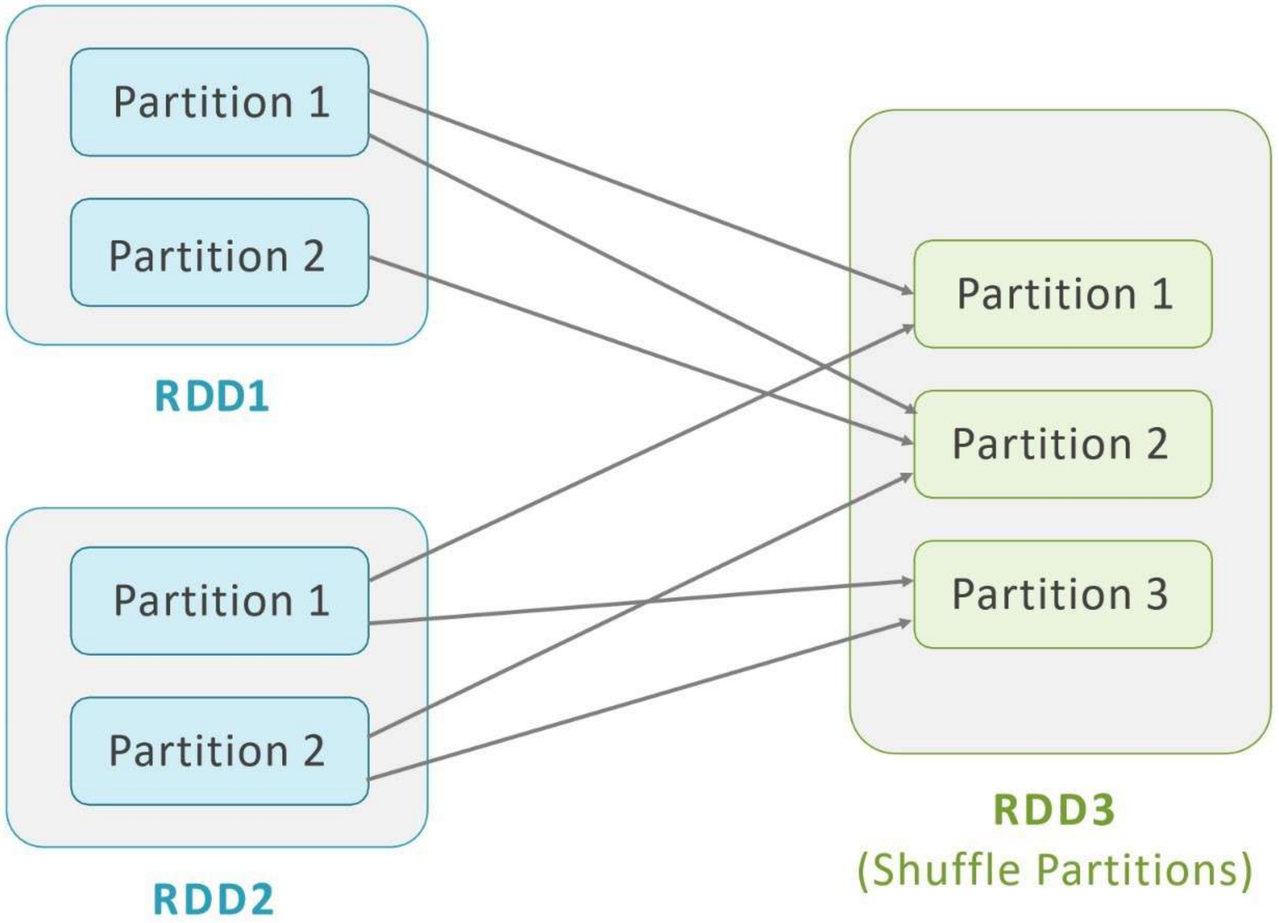
Shuffle Partition 3

Shuffle Write → Shuffle Read


Read file,
get City &Amount
RDD 1
Get distinct cities
RDD 2
Distinct
Step 1– Group: Shuffle / Exchange
Step 2 – ~~Sum~~
Seattle 600
London 300
Delhi 700
Partition 1
Seattle 400
Paris 900
Delhi 200
Seattle 900
Partition 2
Seattle
London
Delhi
Shuffle Block 1
Seattle
Paris
Delhi
Shuffle Block 2
Seattle
Seattle
Shuffle Partition 1
London
Paris
Shuffle Partition 2
Delhi
Delhi
Shuffle Partition 3
Seattle
Partition 1
London
Paris
Partition 2
Delhi
Partition 3
Shuffle Write
Shuffle Read

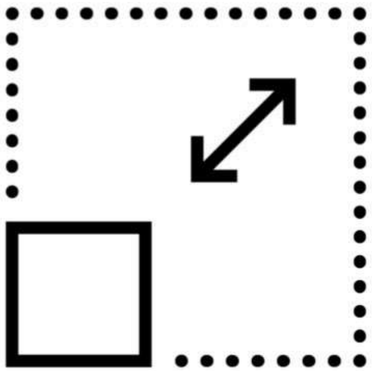

Read file,
get City &Amount
RDD 1
Get distinct cities
RDD 2
Step 1– Group: Shuffle / Exchange
Step 2 – Sum
Seattle 600
London 300
Delhi 700
Partition 1
Seattle 400
Paris 900
Delhi 200
Seattle 900
Partition 2
Seattle
London
Delhi
Shuffle Block 1
Seattle
Paris
Delhi
Shuffle Block 2
Seattle
Seattle
Shuffle Partition 1
London
Paris
Shuffle Partition 2
Delhi
Delhi
Shuffle Partition 3
Seattle
Partition 1
London
Paris
Partition 2
Delhi
Partition 3
Correct Output!
Seattle
London
Paris
Delhi
Shuffle Write
Shuffle Read

One input partition might be used multiple times to produce output partitions



Join operation is an example of this

Wide Transformation



Requires shuffling of data between partitions

Expensive operation

Shuffle Read can only start when all data is written out as Shuffle Blocks

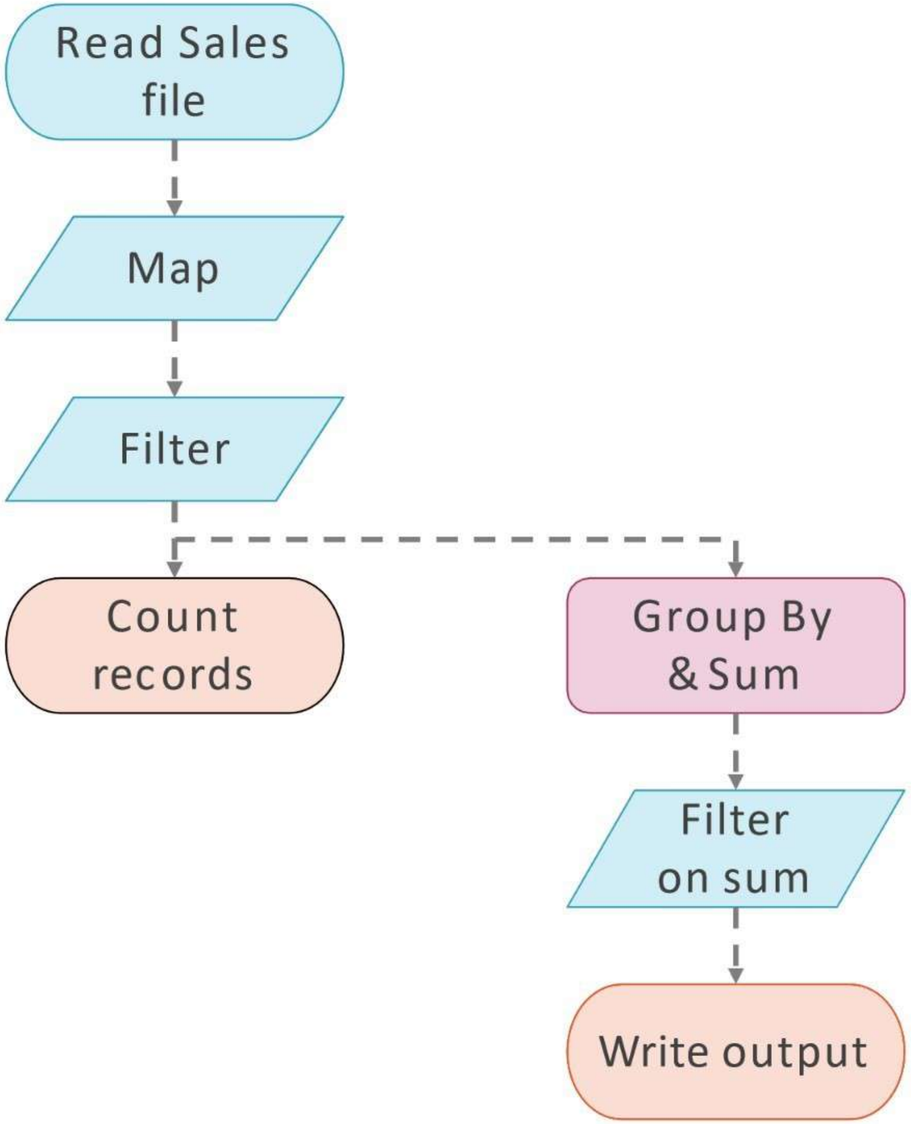
Number of shuffle partitions can be different than parent RDD partitions

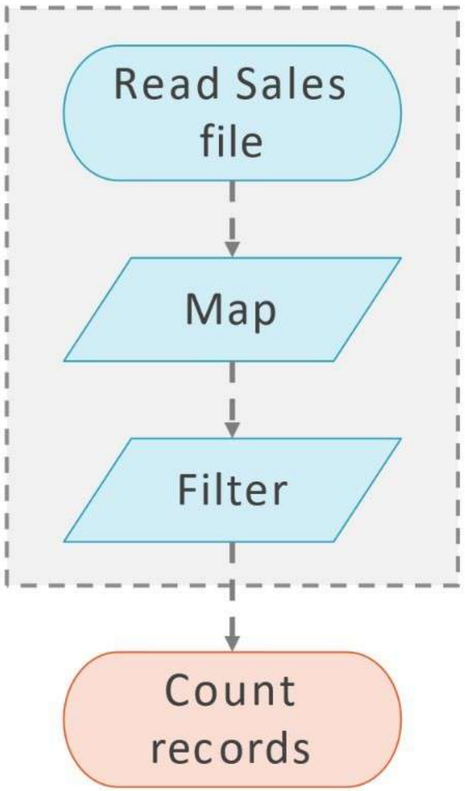
Examples

- reduceByKey, aggregateByKey, distinct, join, intersection etc.

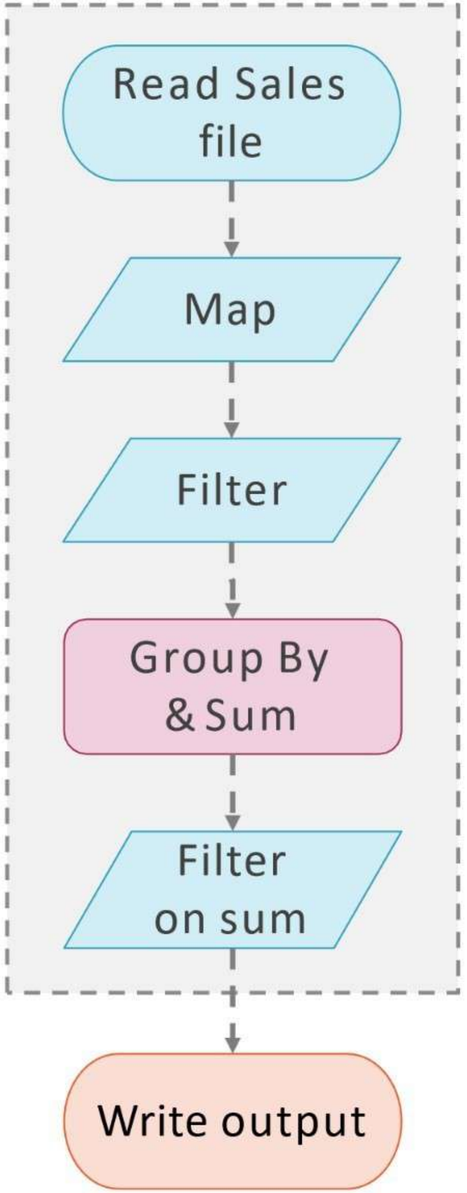
Spark Application Concepts: Jobs, Stages & Tasks

1. Read Sales csv file from Storage
2. Split Sales data by comma & define schema
3. Filter Sales where City = 'Delhi' → T
4. Count number of Sales → A
5. Group by Product, calculate sum of Amount
6. Filter grouped data where TotalAmount > '10000'
7. Write processed data to storage



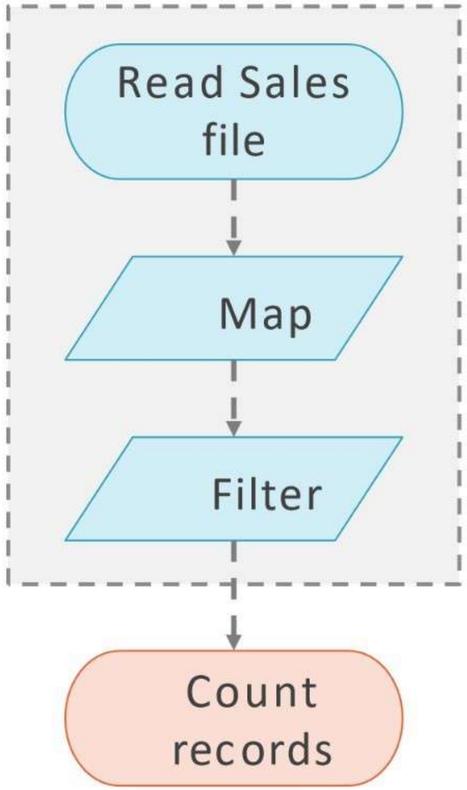


Job 1

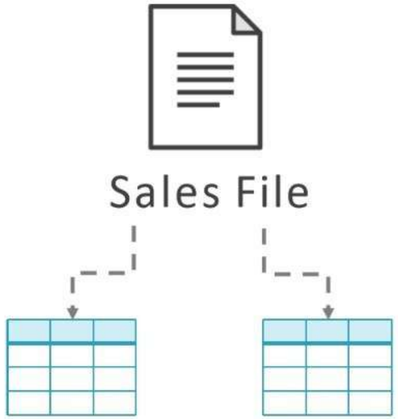


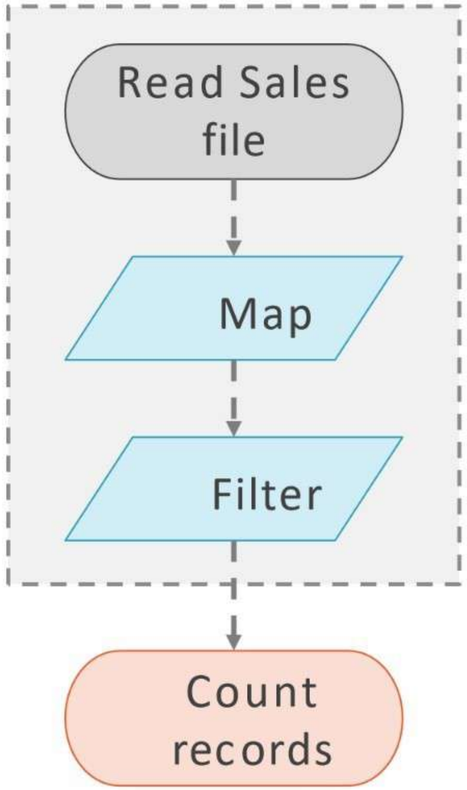
Job 2

No of Jobs
=
Action Operations

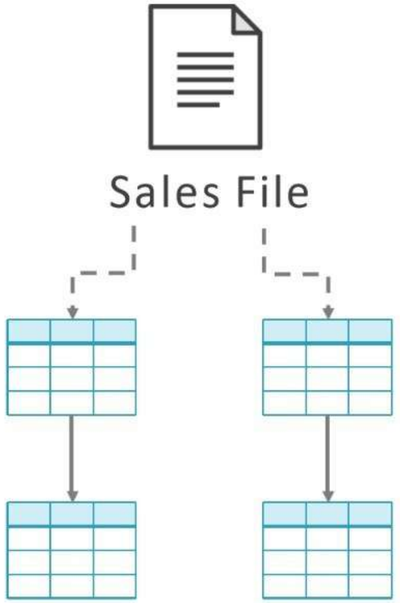


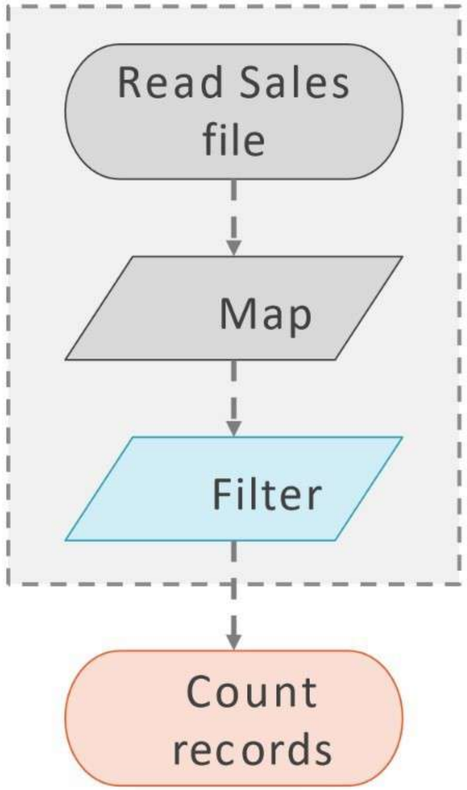
Job 1



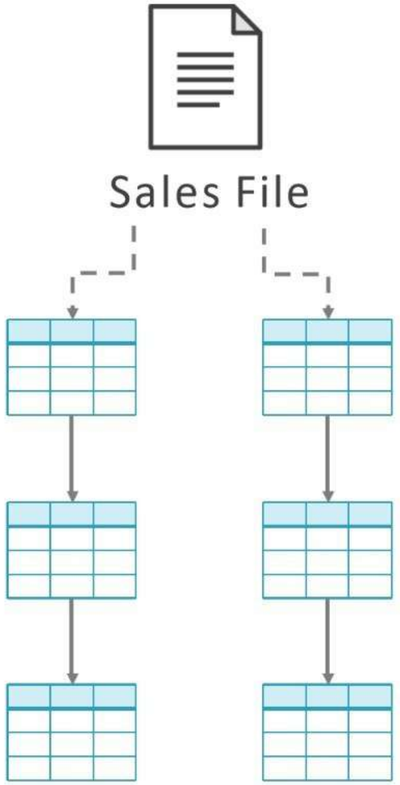


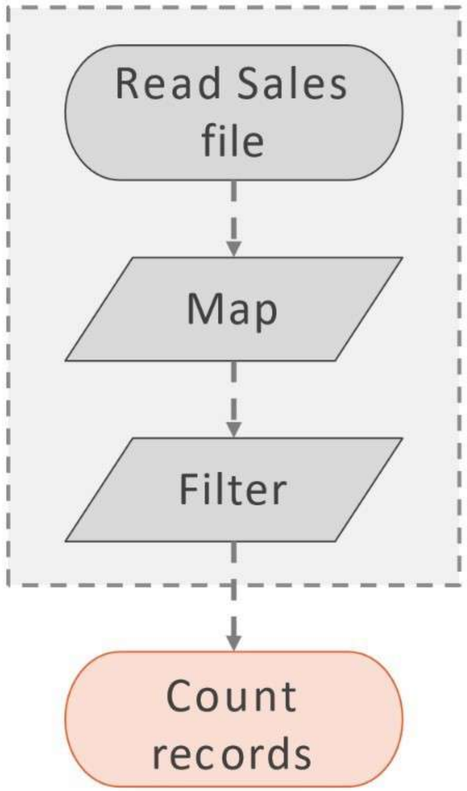
Job 1



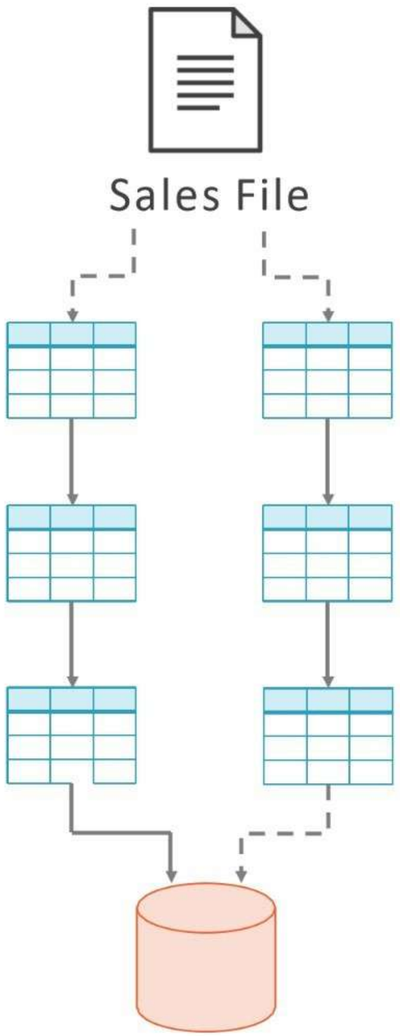


Job 1



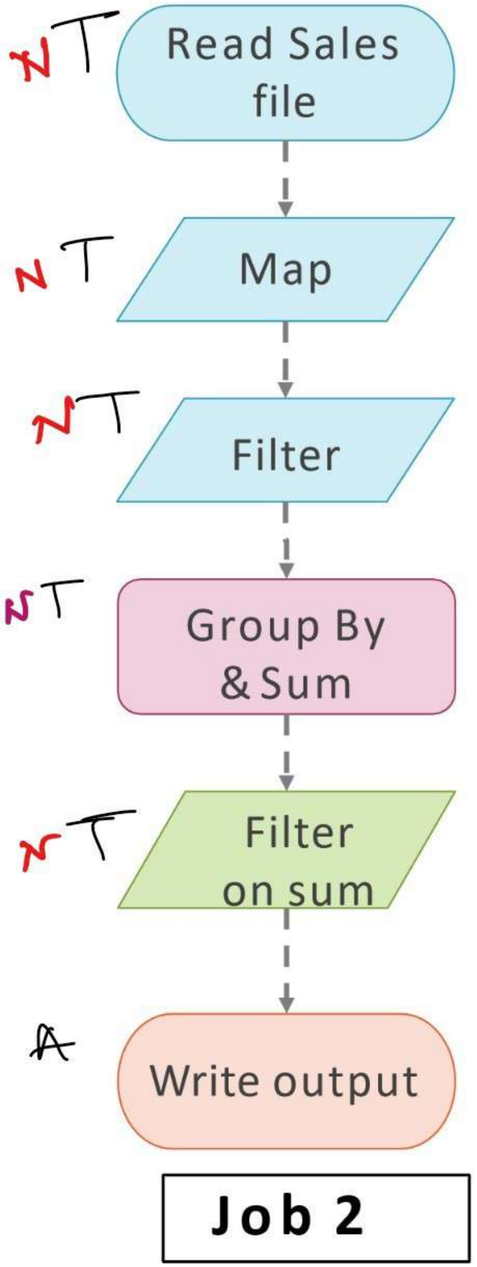


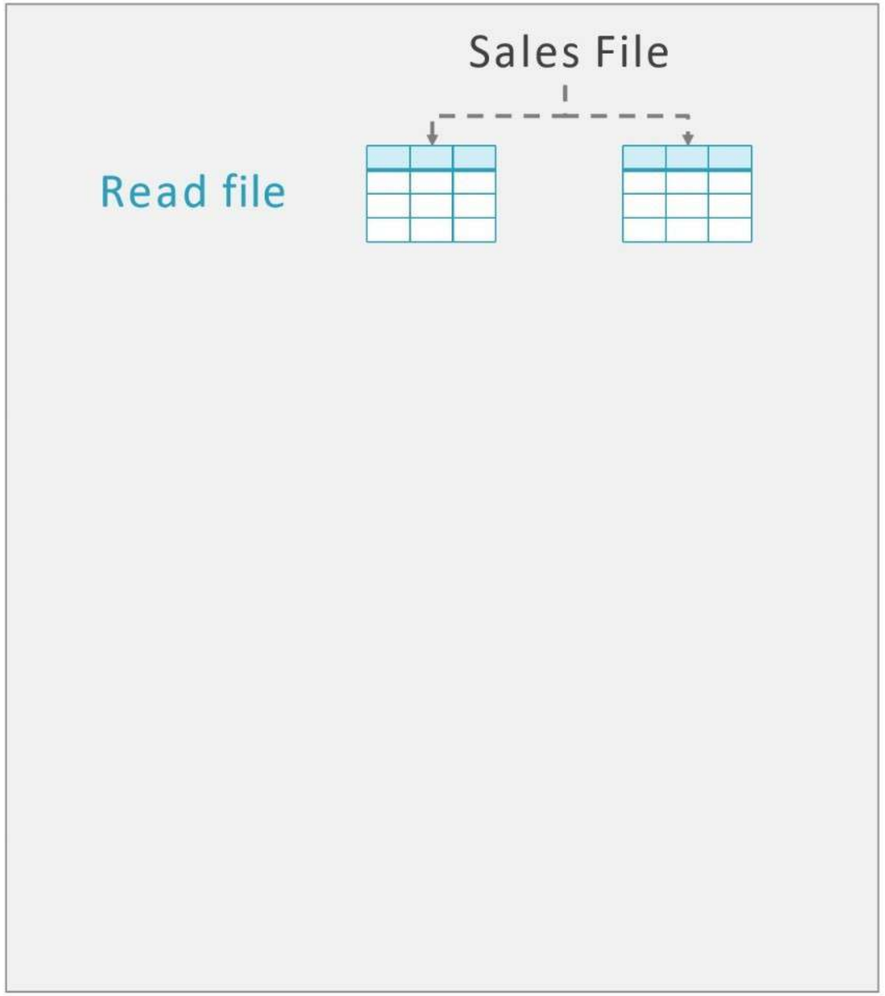
Job 1



Narrow Transformation

Number of output partitions are typically the same as input partitions

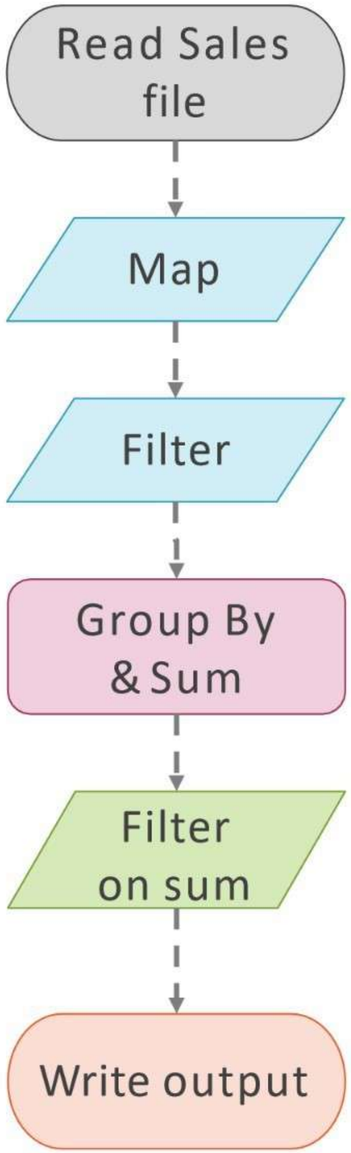




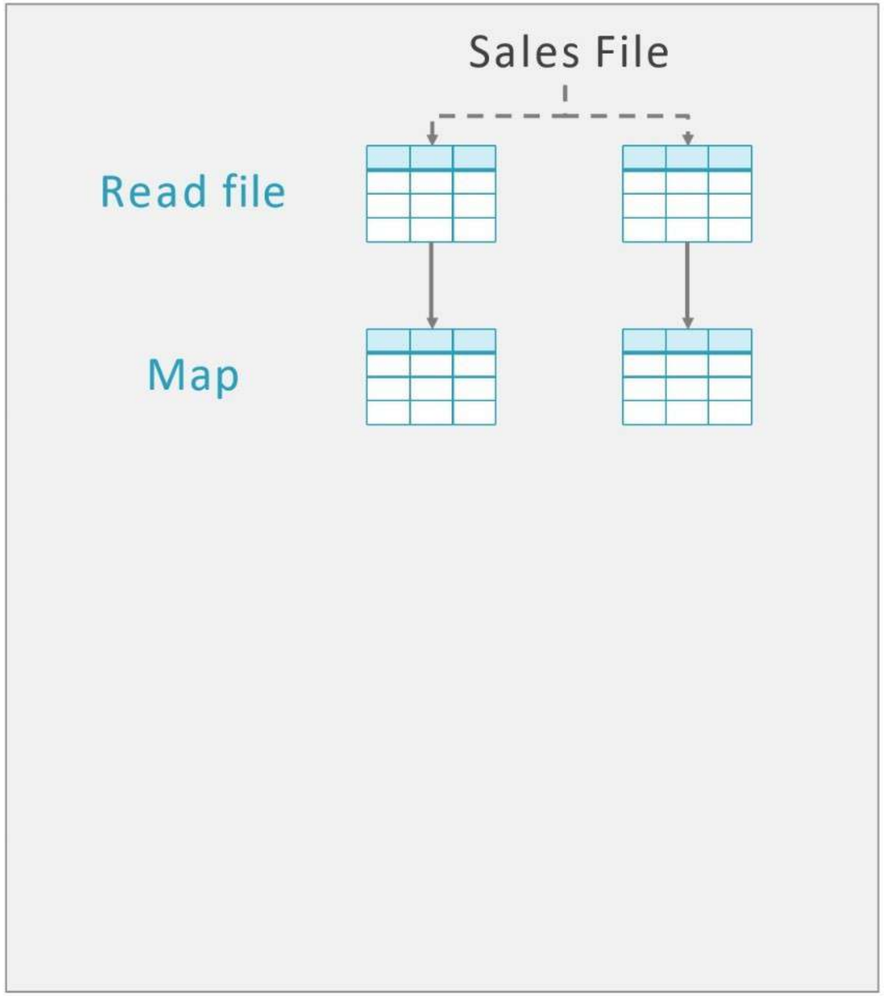
Stage 1

Stage 2

When Spark sees Wide Transformation, it splits the Job in Stages

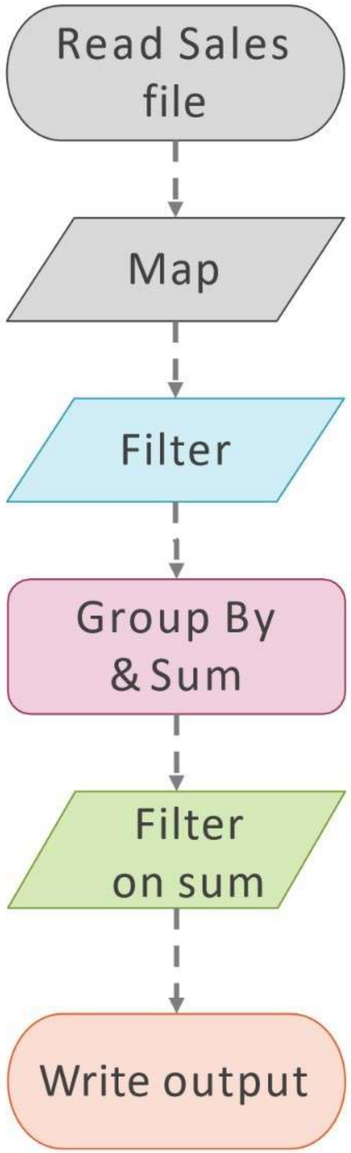


Job 2

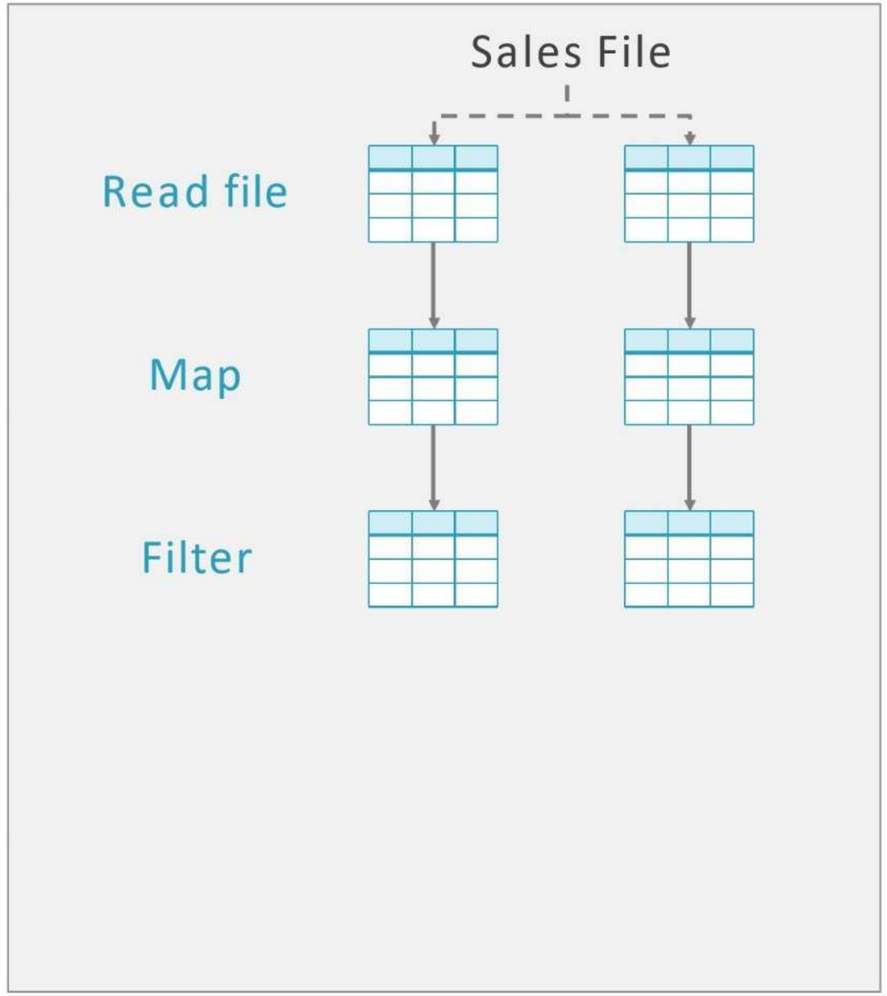


Stage 1

Stage 2

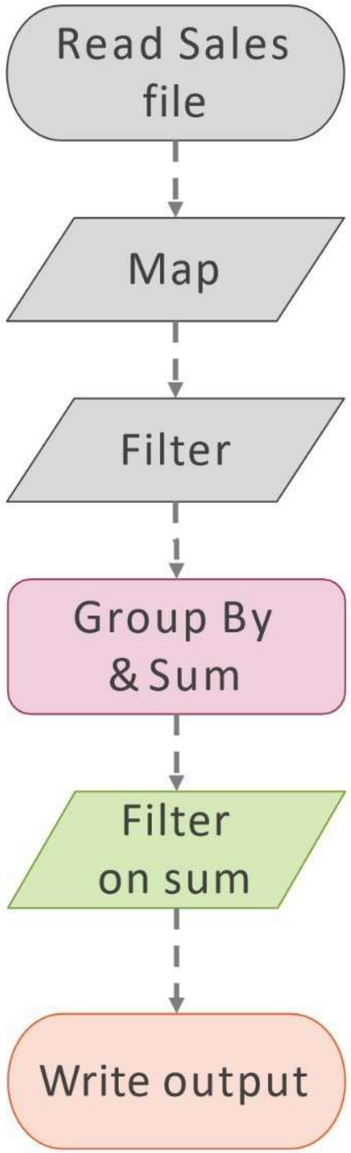


Job 2

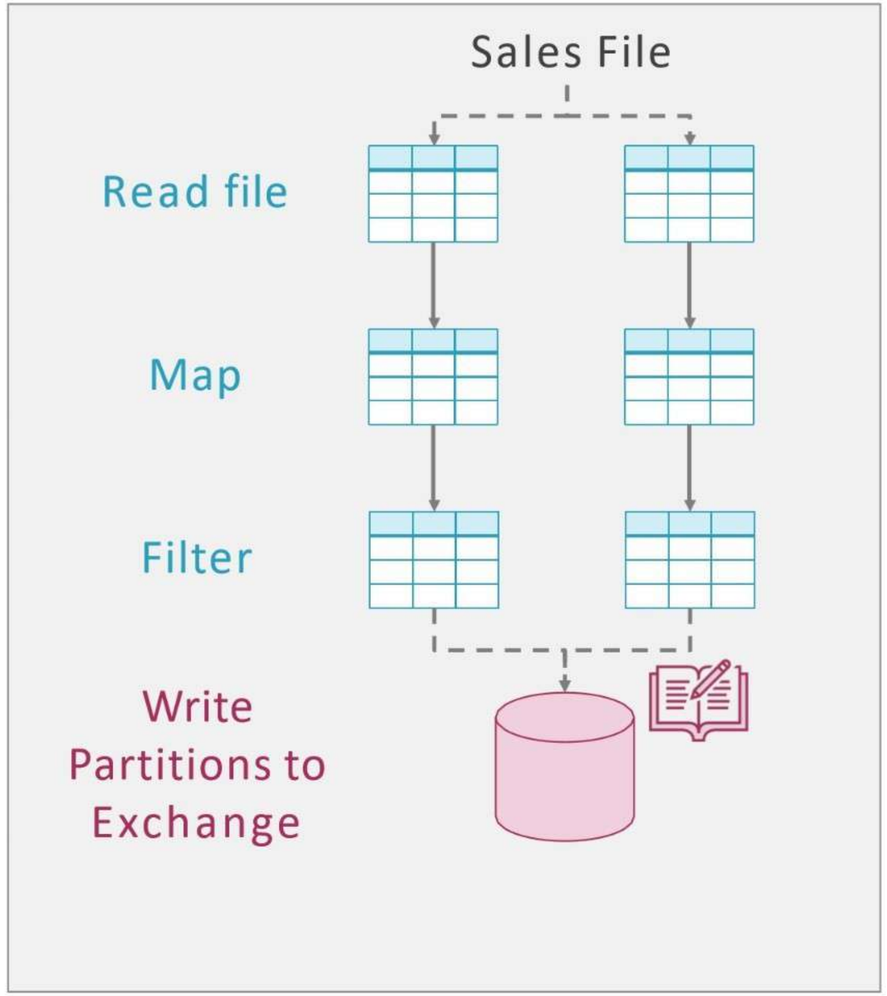


Stage 1

Stage 2

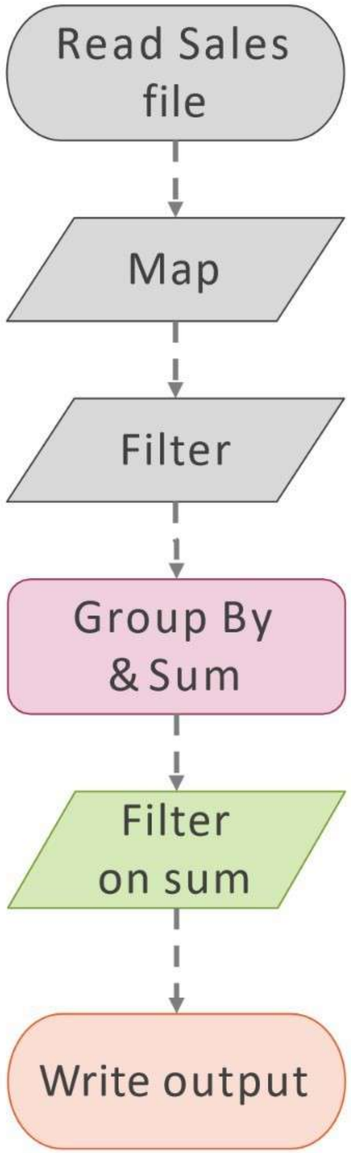


Job 2

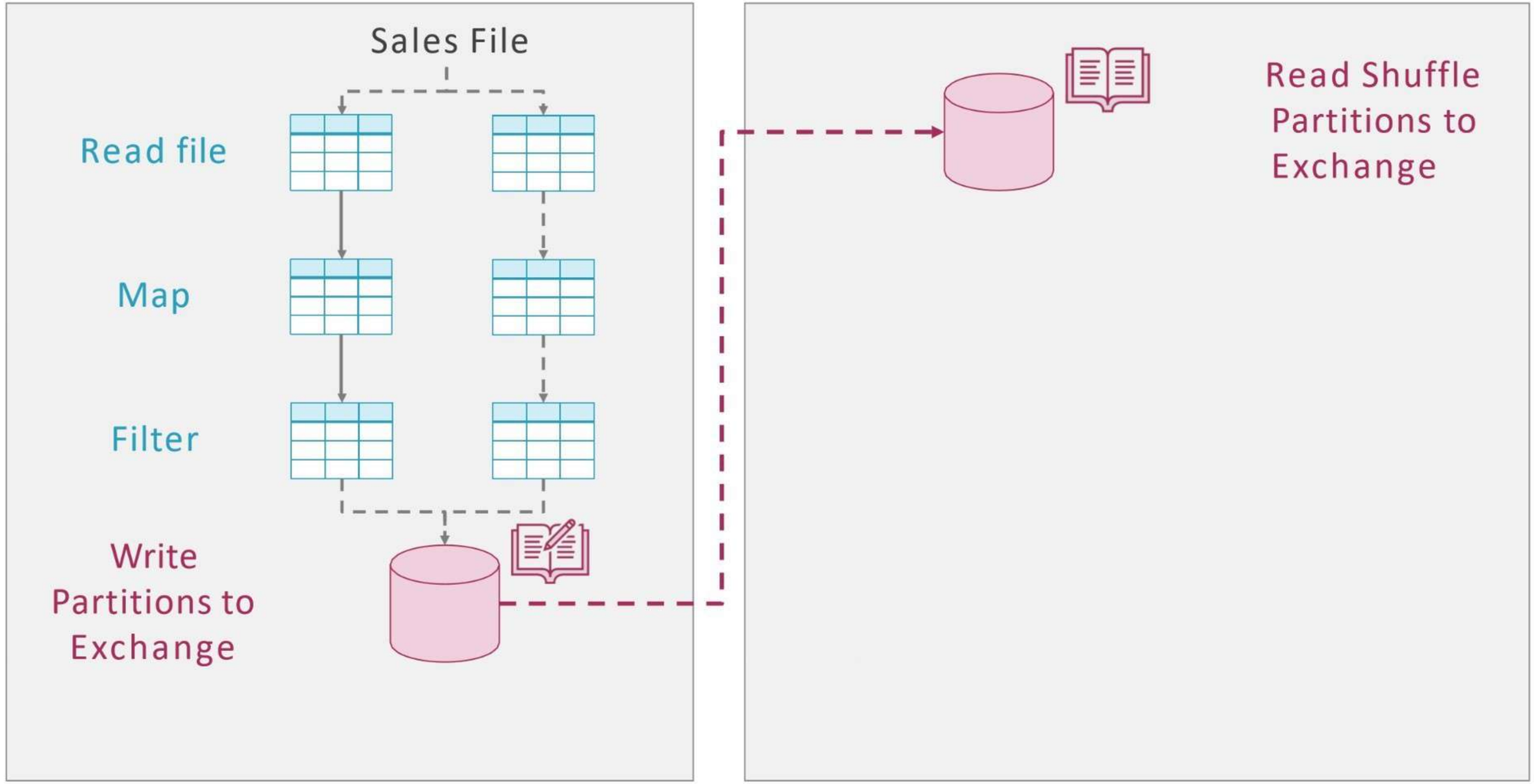


Stage 1

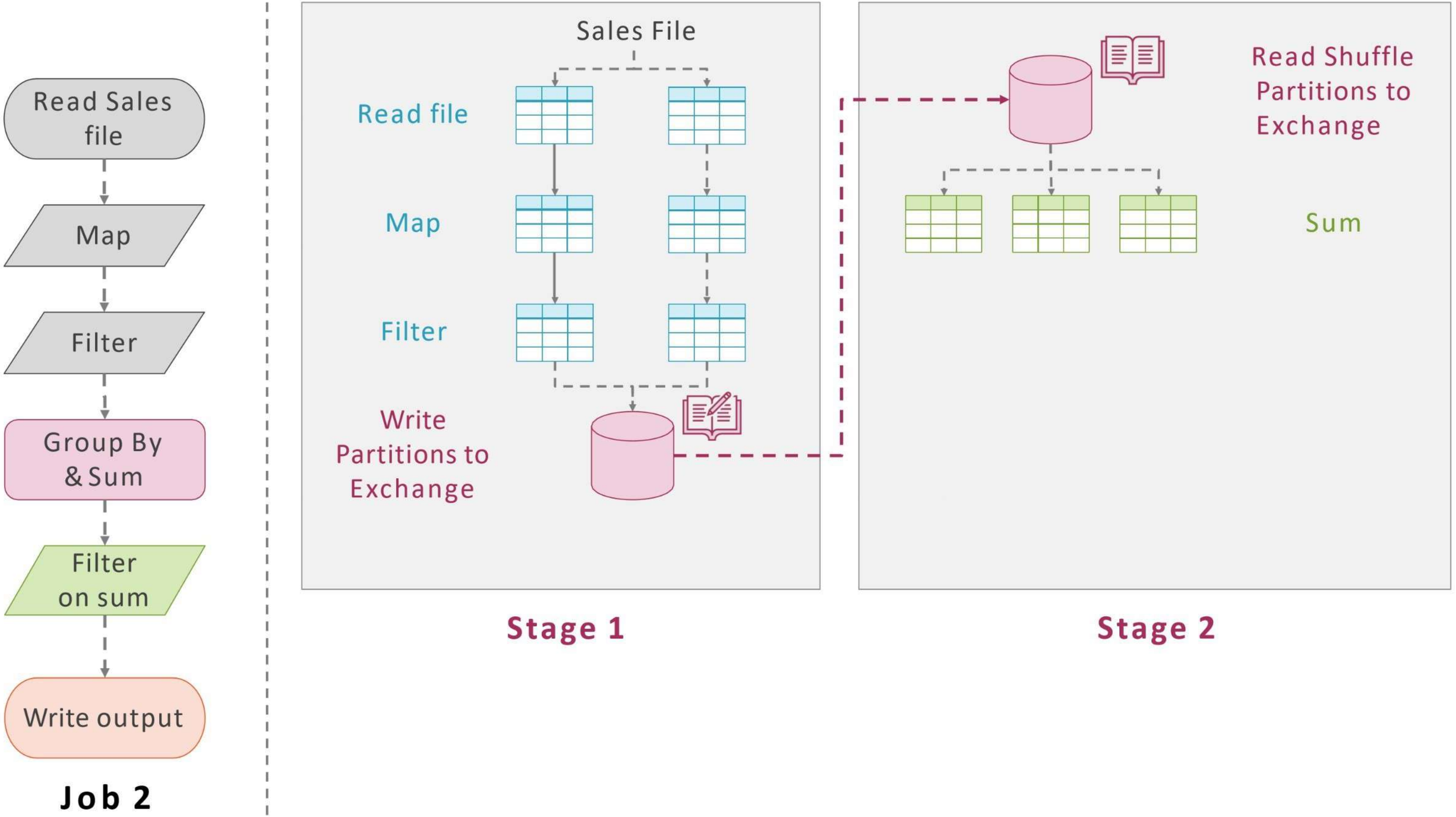
Stage 2

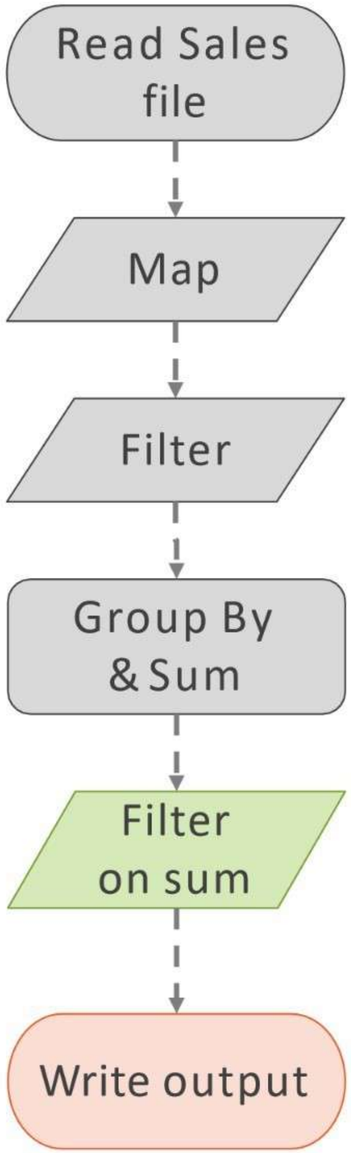


Job 2

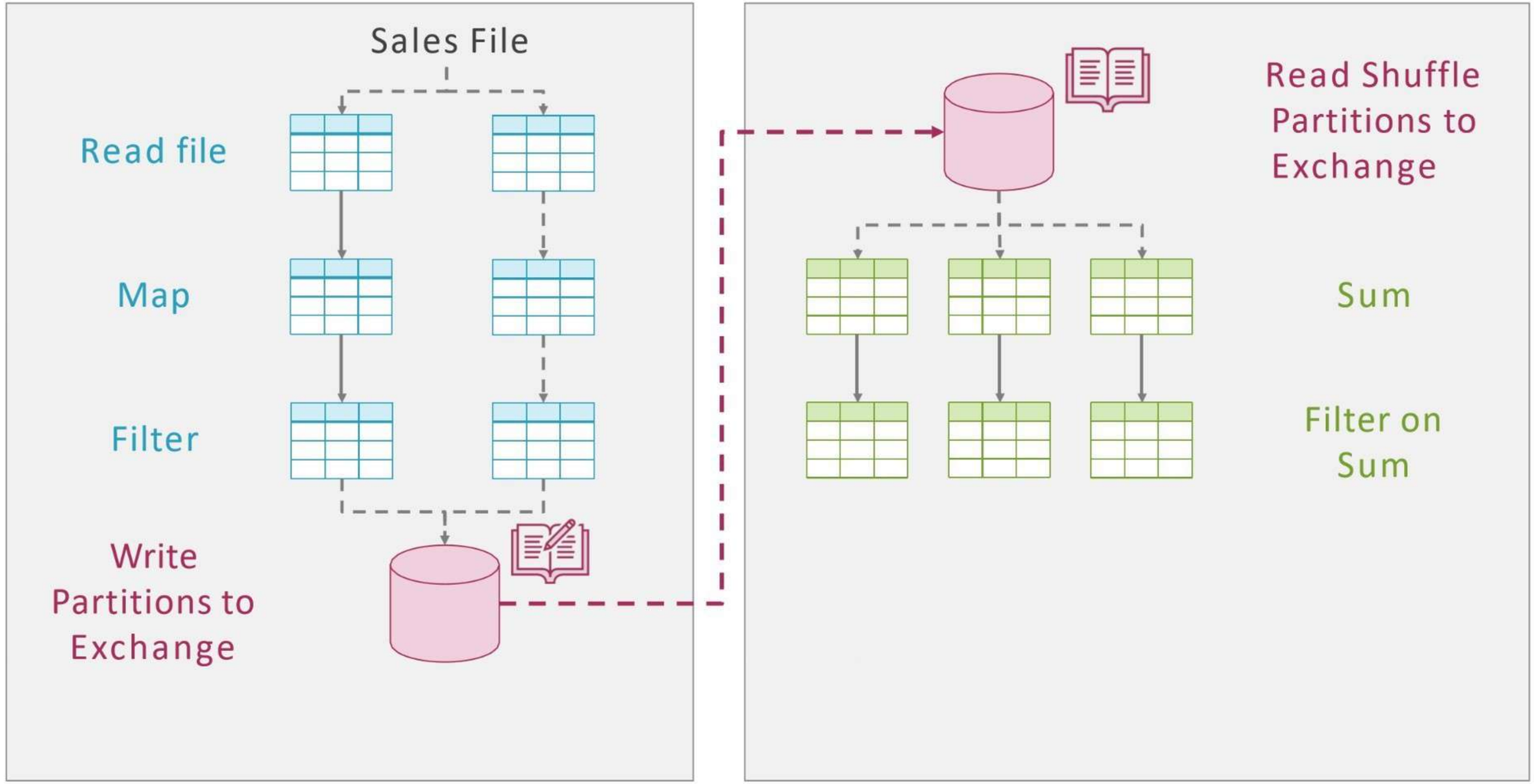


Stage 1 **Stage 2**

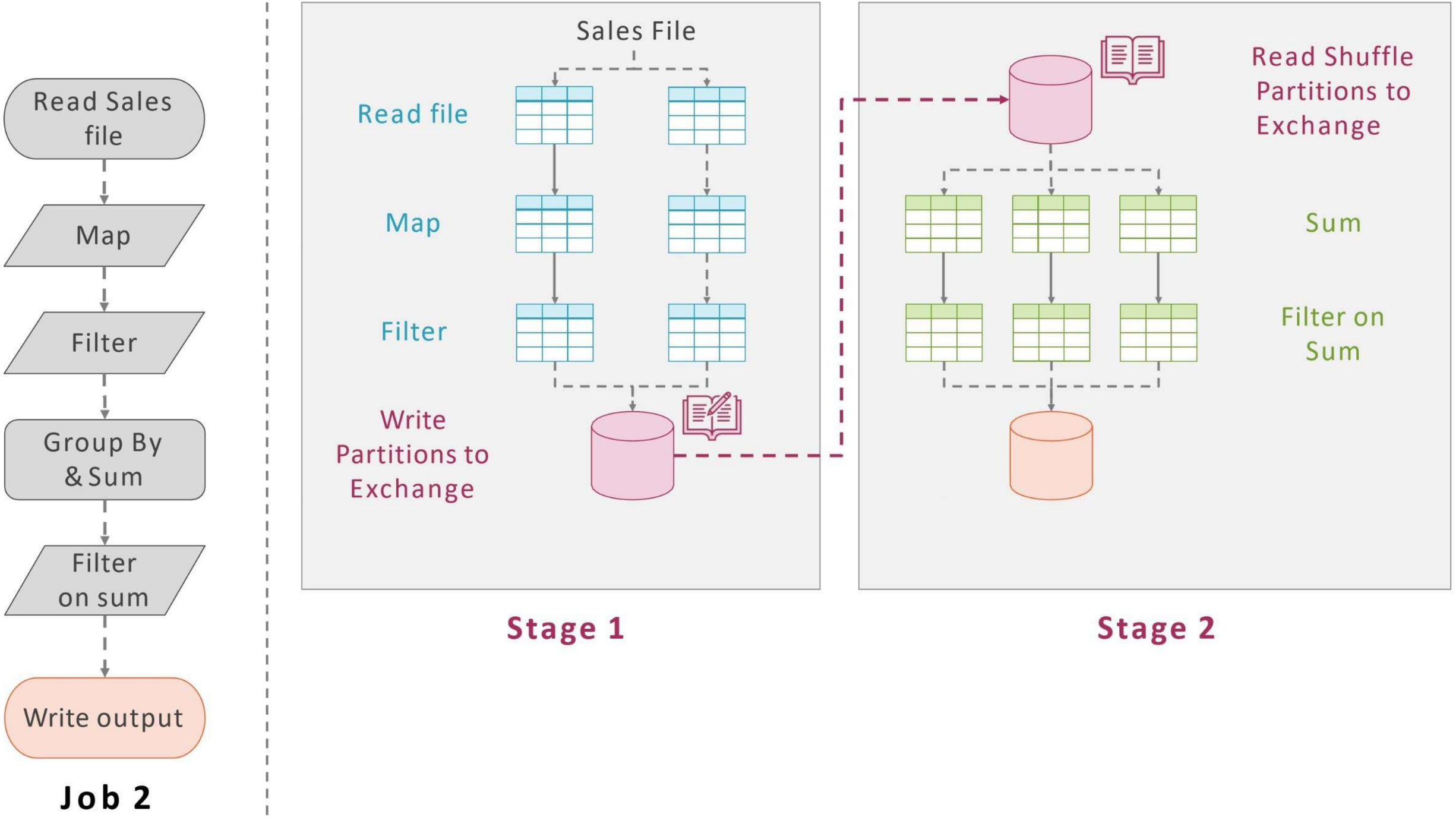

Read Sales file
Map
Filter
Group By & Sum
Filter on sum
Write output
Job 2
Sales File
Read file
Map
Filter
Write Partitions to Exchange
Stage 1
Read Shuffle Partitions to Exchange
Sum
Stage 2



Job 2



Stage 1 **Stage 2**


Read Sales file
Map
Filter
Group By & Sum
Filter on sum
Write output
Job 2
Sales File
Read file
Map
Filter
Write Partitions to Exchange
Stage 1
Read Shuffle Partitions to Exchange
Sum
Filter on Sum
Stage 2

Spark Execution Components

Spark Application is a set of resources

- Contains driver and executor processes

Multiple jobs can run in an Application

- Number of jobs = Action operations applied

Each job is divided into Stages

- Number of stages = Wide transformations + 1

Stages are typically executed in sequence

- When one stage finishes, then only next can start
- Exceptions – Join where 2 datasets can be read parallelly as separate stages

Each Stage has its own set of Tasks

- Number of tasks = Number of partitions
- Number of parallel tasks = Number of cores

File Edit View Insert Cell Kernel Widgets Help

Run Markdown

In [1]:
```
import findspark
findspark.init()
findspark.find()
```

Out[1]:
```
'E:\\Local d\\Zartab\\Apache Spark Installation\\spark-3.3.2-bin-hadoop3'
```

In [2]:
```
from pyspark.sql import SparkSession
from pyspark.sql.types import *

spark = (
            SparkSession
                .builder
                .appName("SparkConceptsApp")
                .master("local[4]")
                .getOrCreate()
        )

sc = spark.sparkContext

spark
```

Out[2]:

SparkSession - in-memory

SparkContext

Spark UI

Identify Entities

1. Narrow Transformations
2. Wide Transformations
3. Actions
4. Jobs
5. Stages
6. Tasks
7. Partitions

1. Read CSV File

Identify transformations, actions, jobs, stages, partitions, tasks

In [5]:
```
# Read File - TaxiZones.csv
taxiZonesRdd = sc.textFile("E:\SparkData\Raw\TaxiZones.csv", 4)

taxiZonesRdd.collect()
```

Out[5]:
```
['1,EWR,Newark Airport,EWR',
 '2,Queens,Jamaica Bay,Boro Zone',
 '3,Bronx,Allerton/Pelham Gardens,Boro Zone',
 '4,Manhattan,Alphabet City,Yellow Zone',
 '5,Staten Island,Arden Heights,Boro Zone',
 '6,Staten Island,Arrochar/Fort Wadsworth,Boro Zone',
 '7,Queens,Astoria,Boro Zone',
 '8,Queens,Astoria Park,Boro Zone',
 '9,Queens,Auburndale,Boro Zone',
 '10,Queens,Baisley Park,Boro Zone',
 '11,Brooklyn,Bath Beach,Boro Zone',
 '12,Manhattan,Battery Park,Yellow Zone',
 '13,Manhattan,Battery Park City,Yellow Zone',
 '14,Brooklyn,Bay Ridge,Boro Zone',
 '15,Queens,Bay Terrace/Fort Totten,Boro Zone'
```

1. Read CSV File

Identify transformations, actions, jobs, stages, partitions, tasks

In [5]:
```
# Read File - TaxiZones.csv
taxiZonesRdd = sc.textFile("E:\SparkData\Raw\TaxiZones.csv", 4)

taxiZonesRdd.collect()
```

Out[5]:
```
['1,EWR,Newark Airport,EWR',
 '2,Queens,Jamaica Bay,Boro Zone',
 '3,Bronx,Allerton/Pelham Gardens,Boro Zone',
 '4,Manhattan,Alphabet City,Yellow Zone',
 '5,Staten Island,Arden Heights,Boro Zone',
 '6,Staten Island,Arrochar/Fort Wadsworth,Boro Zone',
 '7,Queens,Astoria,Boro Zone',
 '8,Queens,Astoria Park,Boro Zone',
 '9,Queens,Auburndale,Boro Zone',
 '10,Queens,Baisley Park,Boro Zone',
 '11,Brooklyn,Bath Beach,Boro Zone',
 '12,Manhattan,Battery Park,Yellow Zone',
 '13,Manhattan,Battery Park City,Yellow Zone',
 '14,Brooklyn,Bay Ridge,Boro Zone',
 '15,Queens,Bay Terrace/Fort Totten,Boro Zone'
```

Transformation: 1
Action : 1
Jobs: 1
Stages: 1

Spark 3.3.2 | Jobs | Stages | Storage | Environment | Executors | SQL / DataFrame | **SparkConceptsApp** application UI

Spark Jobs (?)

User: VICTUS
Total Uptime: 4.5 min
Scheduling Mode: FIFO
Completed Jobs: 1

▸ Event Timeline

▾ Completed Jobs (1)

Page: 1 — 1 Pages. Jump to 1 . Show 100 items in a page. Go

Job Id ▾	Description	Submitted	Duration	Stages: Succeeded/Total	Tasks (for all stages): Succeeded/Total
0	collect at C:\Users\VICTUS\AppData\Local\Temp\ipykernel_13836\1901593258.py:4 collect at C:\Users\VICTUS\AppData\Local\Temp\ipykernel_13836\1901593258.py:4	2023/05/18 07:22:07	0.5 s	1/1	4/4

Page: 1 — 1 Pages. Jump to 1 . Show 100 items in a page. Go

2. Additional step: Split by comma

Identify transformations, actions, jobs, stages, partitions, tasks

In [6]:
```
# Read File - TaxiZones.csv
taxiZonesRdd = sc.textFile("E:\SparkData\Raw\TaxiZones.csv", 4)


# Map Operation - Split data by comma
taxiZonesWithColsRdd = taxiZonesRdd.map( lambda zone: zone.split(",") )


taxiZonesWithColsRdd.collect()
```

Out[6]:
```
[['1', 'EWR', 'Newark Airport', 'EWR'],
 ['2', 'Queens', 'Jamaica Bay', 'Boro Zone'],
 ['3', 'Bronx', 'Allerton/Pelham Gardens', 'Boro Zone'],
 ['4', 'Manhattan', 'Alphabet City', 'Yellow Zone'],
 ['5', 'Staten Island', 'Arden Heights', 'Boro Zone'],
 ['6', 'Staten Island', 'Arrochar/Fort Wadsworth', 'Boro Zone'],
 ['7', 'Queens', 'Astoria', 'Boro Zone'],
 ['8', 'Queens', 'Astoria Park', 'Boro Zone'],
 ['9', 'Queens', 'Auburndale', 'Boro Zone'],
 ['10', 'Queens', 'Baisley Park', 'Boro Zone'],
 ['11', 'Brooklyn', 'Bath Beach', 'Boro Zone'],
```

2. Additional step: Split by comma

Identify transformations, actions, jobs, stages, partitions, tasks

In [6]:
```
# Read File - TaxiZones.csv
taxiZonesRdd = sc.textFile("E:\SparkData\Raw\TaxiZones.csv", 4)


# Map Operation - Split data by comma
taxiZonesWithColsRdd = taxiZonesRdd.map( lambda zone: zone.split(",") )


taxiZonesWithColsRdd.collect()
```

Out[6]:
```
[['1', 'EWR', 'Newark Airport', 'EWR'],
 ['2', 'Queens', 'Jamaica Bay', 'Boro Zone'],
 ['3', 'Bronx', 'Allerton/Pelham Gardens', 'Boro Zone'],
 ['4', 'Manhattan', 'Alphabet City', 'Yellow Zone'],
 ['5', 'Staten Island', 'Arden Heights', 'Boro Zone'],
 ['6', 'Staten Island', 'Arrochar/Fort Wadsworth', 'Boro Zone'],
 ['7', 'Queens', 'Astoria', 'Boro Zone'],
 ['8', 'Queens', 'Astoria Park', 'Boro Zone'],
 ['9', 'Queens', 'Auburndale', 'Boro Zone'],
 ['10', 'Queens', 'Baisley Park', 'Boro Zone'],
 ['11', 'Brooklyn', 'Bath Beach', 'Boro Zone'],
```

Action : 1
Jobs: 1
Transformation: (2 Narrow)
Stages: 1

Spark 3.3.2 | Jobs | Stages | Storage | Environment | Executors | SQL / DataFrame | **SparkConceptsApp** application UI

Spark Jobs (?)

User: VICTUS
Total Uptime: 8.2 min
Scheduling Mode: FIFO
Completed Jobs: 2

▸ Event Timeline

▾ Completed Jobs (2)

Page: 1 | 1 Pages. Jump to 1 . Show 100 items in a page. Go

Job Id	Description	Submitted	Duration	Stages: Succeeded/Total	Tasks (for all stages): Succeeded/Total
1	collect at C:\Users\VICTUS\AppData\Local\Temp\ipykernel_13836\1379414820.py:9 collect at C:\Users\VICTUS\AppData\Local\Temp\ipykernel_13836\1379414820.py:9	2023/05/18 07:26:02	2 s	1/1	4/4
0	collect at C:\Users\VICTUS\AppData\Local\Temp\ipykernel_13836\1901593258.py:4 collect at C:\Users\VICTUS\AppData\Local\Temp\ipykernel_13836\1901593258.py:4	2023/05/18 07:22:07	0.5 s	1/1	4/4

In [7]:
```
# Check partitions after reading file & after map operation

print ("After reading file = " + str( taxiZonesRdd.getNumPartitions() ))

print ("After applying map = " + str( taxiZonesWithColsRdd.getNumPartitions() ))
```

```
After reading file = 4
After applying map = 4
```

3. Additional step: Create Pair RDD

Identify transformations, actions, jobs, stages, partitions, tasks

In [8]:
```
# Read File - TaxiZones.csv
taxiZonesRdd = sc.textFile("E:\SparkData\Raw\TaxiZones.csv", 4)


# Map Operation - Split data by comma
taxiZonesWithColsRdd = taxiZonesRdd.map( lambda zone: zone.split(",") )


# Map Operation - Create Pair RDD with <Borough, 1>
taxiZonesPairRdd = taxiZonesWithColsRdd.map( lambda zoneRow: (zoneRow[1], 1) )


taxiZonesPairRdd.count()
```

Out[8]: 265

3. Additional step: Create Pair RDD

Identify transformations, actions, jobs, stages, partitions, tasks

In [8]:
```
# Read File - TaxiZones.csv
taxiZonesRdd = sc.textFile("E:\SparkData\Raw\TaxiZones.csv", 4)


# Map Operation - Split data by comma
taxiZonesWithColsRdd = taxiZonesRdd.map( lambda zone: zone.split(",") )


# Map Operation - Create Pair RDD with <Borough, 1>
taxiZonesPairRdd = taxiZonesWithColsRdd.map( lambda zoneRow: (zoneRow[1], 1) )


taxiZonesPairRdd.count()
```

Out[8]: 265

Action : ?
Jobs: ?
Transformation: ?
Stages: ?

Spark 3.3.2 | Jobs | Stages | Storage | Environment | Executors | SQL / DataFrame | **SparkConceptsApp** application UI

Spark Jobs (?)

User: VICTUS
Total Uptime: 11 min
Scheduling Mode: FIFO
Completed Jobs: 3

▸ Event Timeline

▾ Completed Jobs (3)

Page: 1 | 1 Pages. Jump to 1 . Show 100 items in a page. Go

Job Id ▾	Description	Submitted	Duration	Stages: Succeeded/Total	Tasks (for all stages): Succeeded/Total
2	count at C:\Users\VICTUS\AppData\Local\Temp\ipykernel_13836\2789000522.py:13 count at C:\Users\VICTUS\AppData\Local\Temp\ipykernel_13836\2789000522.py:13	2023/05/18 07:30:02	2 s	1/1	4/4

4. Additional step: Find distinct records

Identify transformations, actions, jobs, stages, partitions, tasks

In [9]:
```
# Read File - TaxiZones.csv
taxiZonesRdd = sc.textFile("E:\SparkData\Raw\TaxiZones.csv", 4)


# Map Operation - Split data by comma
taxiZonesWithColsRdd = taxiZonesRdd.map( lambda zone: zone.split(",") )


# Map Operation - Create Pair RDD with <Borough, 1>
taxiZonesPairRdd = taxiZonesWithColsRdd.map( lambda zoneRow: (zoneRow[1], 1) )


# Distinct Operation - Find distinct items
distinctZonesRdd = taxiZonesPairRdd.distinct()


distinctZonesRdd.collect()
```

Out[9]:
```
[('Bronx', 1),
 ('Staten Island', 1),
```

4. Additional step: Find distinct records

Identify transformations, actions, jobs, stages, partitions, tasks

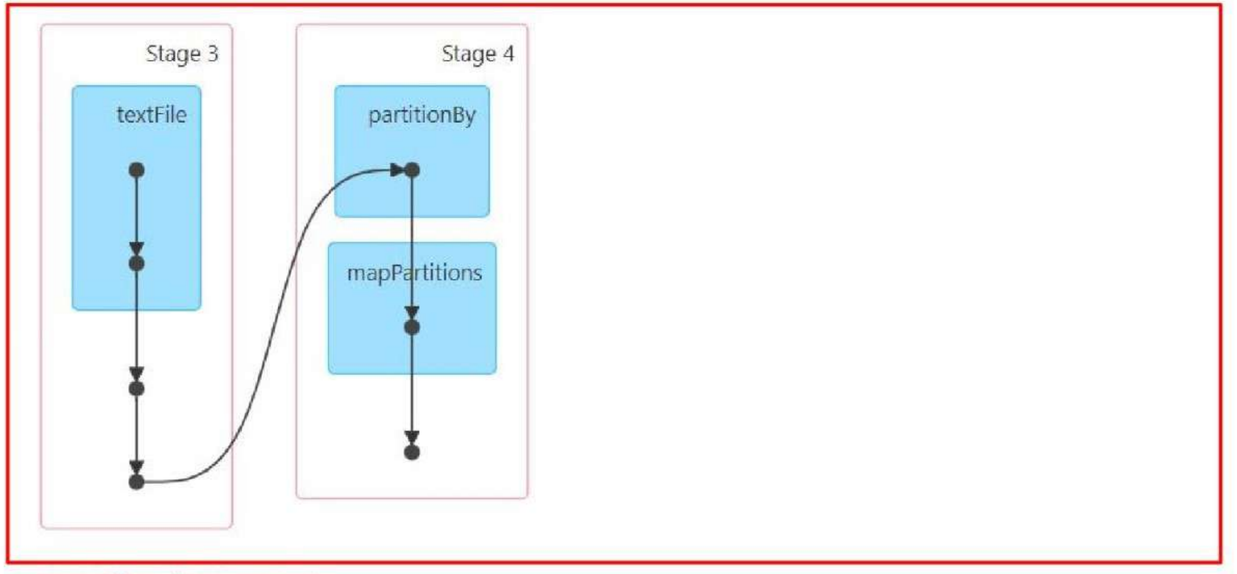
In [9]:
```
# Read File - TaxiZones.csv
taxiZonesRdd = sc.textFile("E:\SparkData\Raw\TaxiZones.csv", 4)


# Map Operation - Split data by comma
taxiZonesWithColsRdd = taxiZonesRdd.map( lambda zone: zone.split(",") )


# Map Operation - Create Pair RDD with <Borough, 1>
taxiZonesPairRdd = taxiZonesWithColsRdd.map( lambda zoneRow: (zoneRow[1], 1) )


# Distinct Operation - Find distinct items
distinctZonesRdd = taxiZonesPairRdd.distinct()


distinctZonesRdd.collect()
```

Out[9]:
```
[('Bronx', 1),
 ('Staten Island', 1),
```

Action : 2
Jobs: 2
Transformation:
3 narrow
1 wide
Stages: 2

Submitted: 2023/05/18 07:33:09
Duration: 4 s
Completed Stages: 2

▸ Event Timeline
▾ DAG Visualization



▾ Completed Stages (2)

Page: 1

1 Pages. Jump to 1 . Show 100 items in a page. Go

Stage Id ▾	Description		Submitted	Duration	Tasks: Succeeded/Total	Input	Output	Shuffle Read	Shuffle Write
4	collect at C:\Users\VICTUS\AppData\Local\Temp\ipykernel_13836\3401495829.py:17	+details	2023/05/18 07:33:11	2 s	4/4			1123.0 B	
3	distinct at C:\Users\VICTUS\AppData\Local\Temp\ipykernel_13836\3401495829.py:14	+details	2023/05/18 07:33:09	2 s	4/4	26.1 KiB			1123.0 B

5. Additional steps:

a) Group by Borough

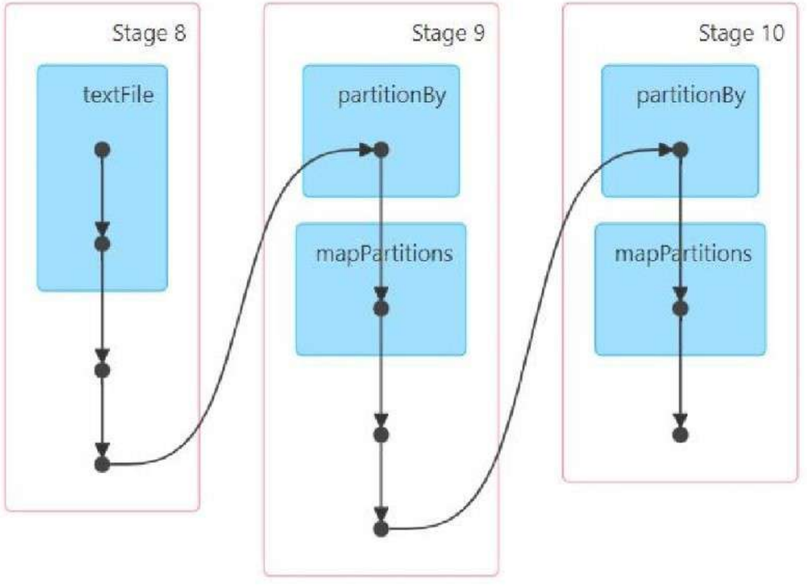
b.) Get rows with Borough count > 10

Identify transformations, actions, jobs, stages, partitions, tasks

In [11]:
```python
# Read File - TaxiZones.csv
taxiZonesRdd = sc.textFile("E:\SparkData\Raw\TaxiZones.csv", 4)


# Map Operation - Split data by comma
taxiZonesWithColsRdd = taxiZonesRdd.map( lambda zone: zone.split(",") )


# Map Operation - Create Pair RDD with <Borough, 1>
taxiZonesPairRdd = taxiZonesWithColsRdd.map( lambda zoneRow: (zoneRow[1], 1) )


# Distinct Operation - Find distinct items
distinctZonesRdd = taxiZonesPairRdd.distinct()


# ReduceByKey Operation - Group by Borough and calculate count
boroughCountRdd = distinctZonesRdd.reduceByKey( lambda value1, value2: value1 + value2 )


# Filter Operation - Get rows with Borough count > 10
filteredZonesRdd = boroughCountRdd.filter( lambda row: row[1] > 10 )


filteredZonesRdd.collect()
```

Completed Stages: 3

▸ Event Timeline
▾ DAG Visualization



▾ Completed Stages (3)

Page: 1

1 Pages. Jump to 1 . Show 100 items in a page. Go

Stage Id ▾	Description		Submitted	Duration	Tasks: Succeeded/Total	Input	Output	Shuffle Read	Shuffle Write
10	collect at C:\Users\VICTUS\AppData\Local\Temp\ipykernel_13836\3211385598.py:25	+details	2023/05/18 07:39:47	2 s	4/4			279.0 B	
9	reduceByKey at C:\Users\VICTUS\AppData\Local\Temp\ipykernel_13836\3211385598.py:18	+details	2023/05/18 07:39:45	2 s	4/4			1123.0 B	279.0 B
8	distinct at C:\Users\VICTUS\AppData\Local\Temp\ipykernel_13836\3211385598.py:14	+details	2023/05/18 07:39:43	2 s	4/4	26.1 KiB			1123.0 B

Run first and take methods

Identify transformations, actions, jobs, stages, partitions, tasks

In [12]:
```
# Read File - TaxiZones.csv
taxiZonesRdd = sc.textFile("E:\SparkData\Raw\TaxiZones.csv", 4)


print ( taxiZonesPairRdd.first() )

print ( taxiZonesPairRdd.take(2) )
```

```
('EWR', 1)
[('EWR', 1), ('Queens', 1)]
```

Spark 3.3.2 | Jobs | Stages | Storage | Environment | Executors | SQL / DataFrame | **SparkConceptsApp** application UI

Spark Jobs (?)

User: VICTUS
Total Uptime: 24 min
Scheduling Mode: FIFO
Completed Jobs: 10

▸ Event Timeline

▾ Completed Jobs (10)

Page: 1 | 1 Pages. Jump to 1 . Show 100 items in a page. Go

Job Id ▾	Description	Submitted	Duration	Stages: Succeeded/Total	Tasks (for all stages): Succeeded/Total
9	runJob at PythonRDD.scala:166 runJob at PythonRDD.scala:166	2023/05/18 07:43:53	0.5 s	1/1	1/1
8	runJob at PythonRDD.scala:166 runJob at PythonRDD.scala:166	2023/05/18 07:43:52	0.5 s	1/1	1/1

Summary

RDD is the native data structure of Spark

- In-memory, Partitioned, Read-only & Resilient

RDD can be created in many ways

- Parallelize a collection, read a file or convert an RDD

Pair RDD is an RDD with Key-Value pairs

Apply transformation and action operations on RDDs

- Transformation gives new RDD. Action gives result
- Transformations are lazy operations

Transformations are of two types

- Narrow – Each input partition is used at-most once
- Wide – An input partition may be used multiple times

Data shuffling is a two-step process

Application → Jobs → Stages → Tasks

DataFrame

Id,Name,Date,Amount
1,Neha,2023-01-01,372
2,Cathy,2023-01-12,293
3,Ivan,2023-01-03,938
4,Kerri,2023-01-17,220
5,Mohit,2023-01-02,494

CSV File in Storage

Read File with

DataFrame API

Id	Name	Date	Amount
1	Neha	2023-01-01	372
2	Cathy	2023-01-12	293
3	Ivan	2023-01-03	938
4	Kerri	2023-01-17	220
5	Mohit	2023-01-02	494

DataFrame in Spark

DataFrame

Id,Name,Date,Amount
1,Neha,2023-01-01,372
2,Cathy,2023-01-12,293
3,Ivan,2023-01-03,938
4,Kerri,2023-01-17,220
5,Mohit,2023-01-02,494

CSV File in Storage

Read File with

DataFrame API

Column

Id	Name	Date	Amount
1	Neha	2023-01-01	372
2	Cathy	2023-01-12	293
3	Ivan	2023-01-03	938
4	Kerri	2023-01-17	220
5	Mohit	2023-01-02	494

Row

DataFrame in Spark

RDD → low level

DataFrames
Dataset } → high level

DataFrames

Id	Name	Date	Amount
1	Neha	2023-01-01	372
2	Cathy	2023-01-12	293
3	Ivan	2023-01-03	938
4	Kerri	2023-01-17	220
5	Mohit	2023-01-02	494

Part of Spark SQL library

Built on top of RDDs
- In-memory, partitioned, read-only & resilient

Imposes a tabular structure on the data

Spark engine can apply optimizations

Less development effort → high level

Allows for structured data processing

Supports variety of Data Types
- Simple – String, Integer, Double, Boolean etc.
- Complex – Array, Timestamp, Binary etc.

Spark Optimization Engines

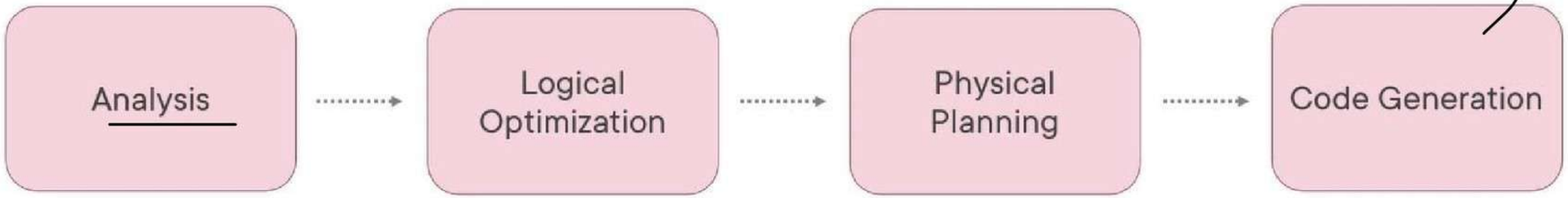
Catalyst Optimizer

Query Optimizer to produce optimized query plan

Tungsten Engine

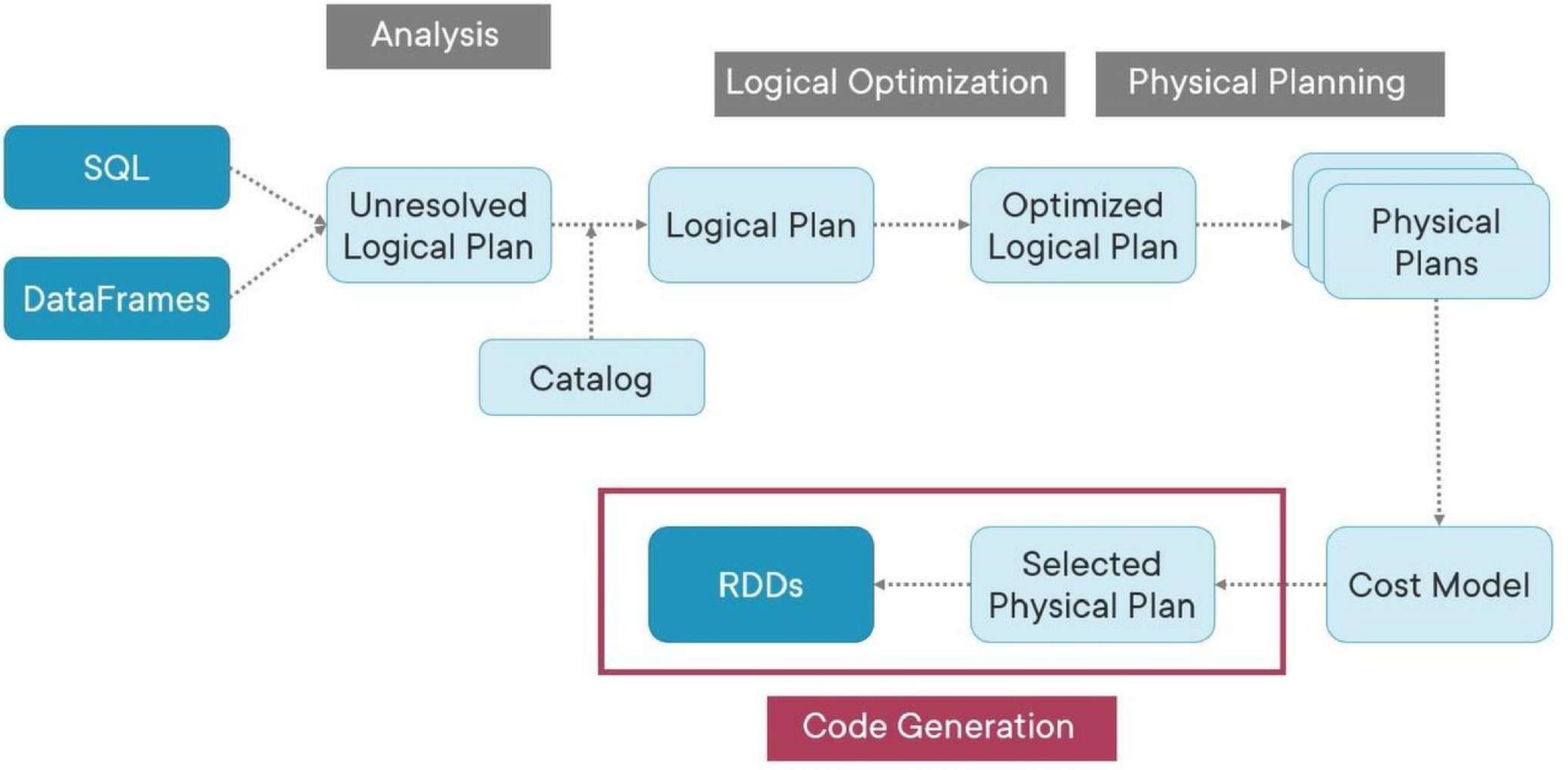
Improves efficiency of memory & CPU for Spark applications

Catalyst Optimization Phases



Tungsten Engine gets involved

Catalyst Optimization Process



Select * from employee where age > 30;

Analysis → confirm age & employee –

Logical Optimizer → age > 30 pushed near to data source

P

```
# Create DataFrame by reading CSV file
taxiRidesDF = spark.read.csv("Rides.csv")
```

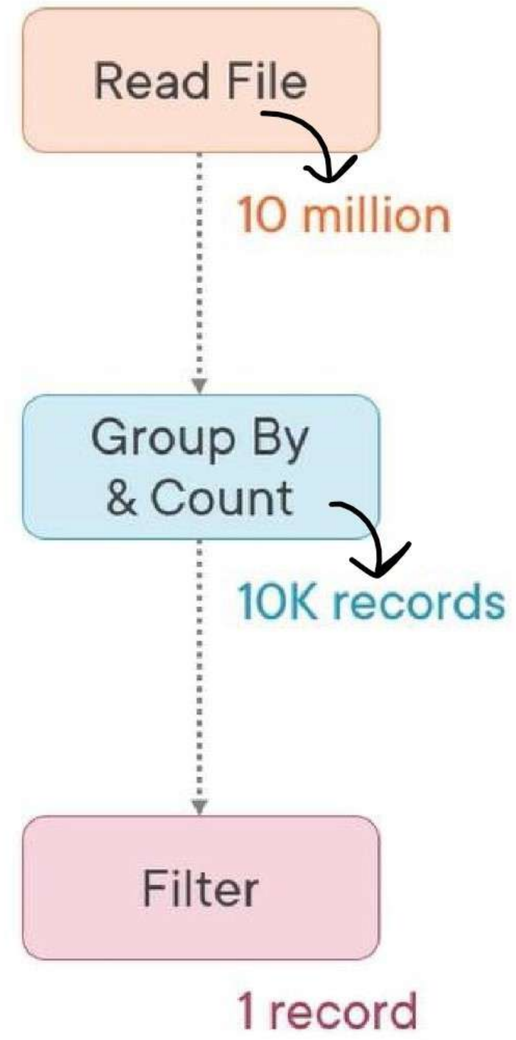
```
# Group data by Pickup Location
ridesByLocDF = taxiRidesDF
    .groupBy("PickupLocation")
    .count()
```

```
# Filter where Pickup Location is Airport
airportRidesDF = ridesByLocDF
    .where("PickupLocation == 'Airport'")
```

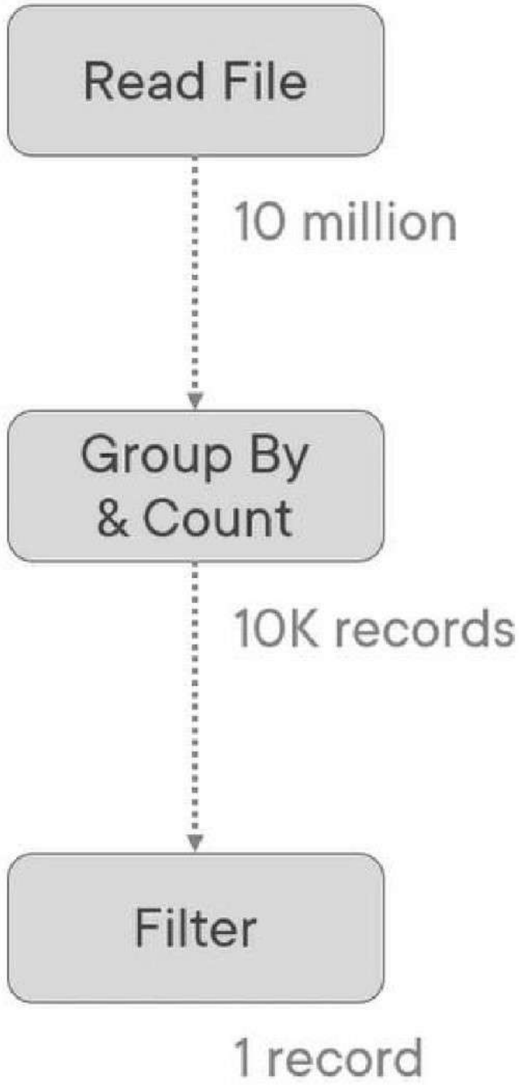
DataFrames

Dev → What?

How? → Spark



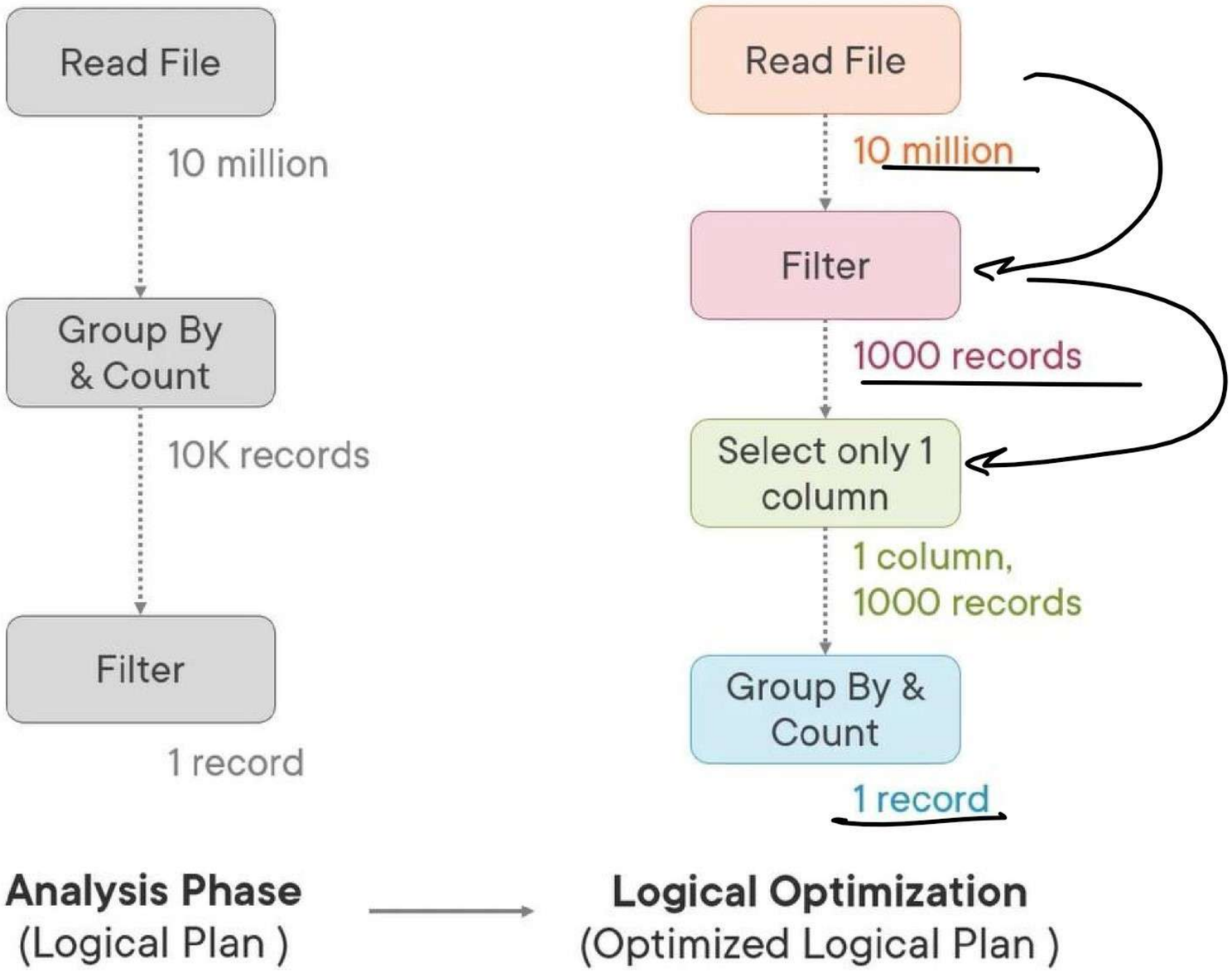
Analysis Phase
(Logical Plan)

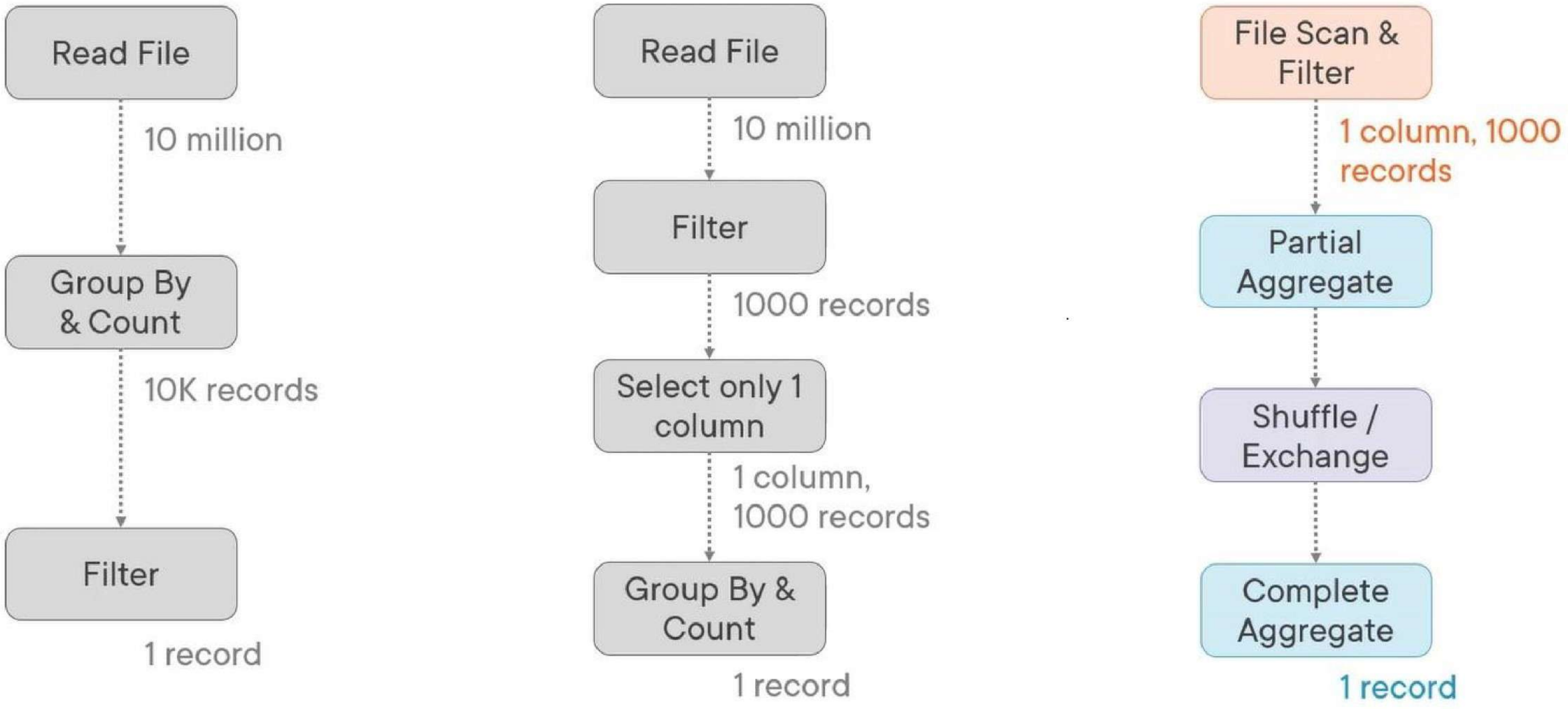


Analysis Phase
(Logical Plan)

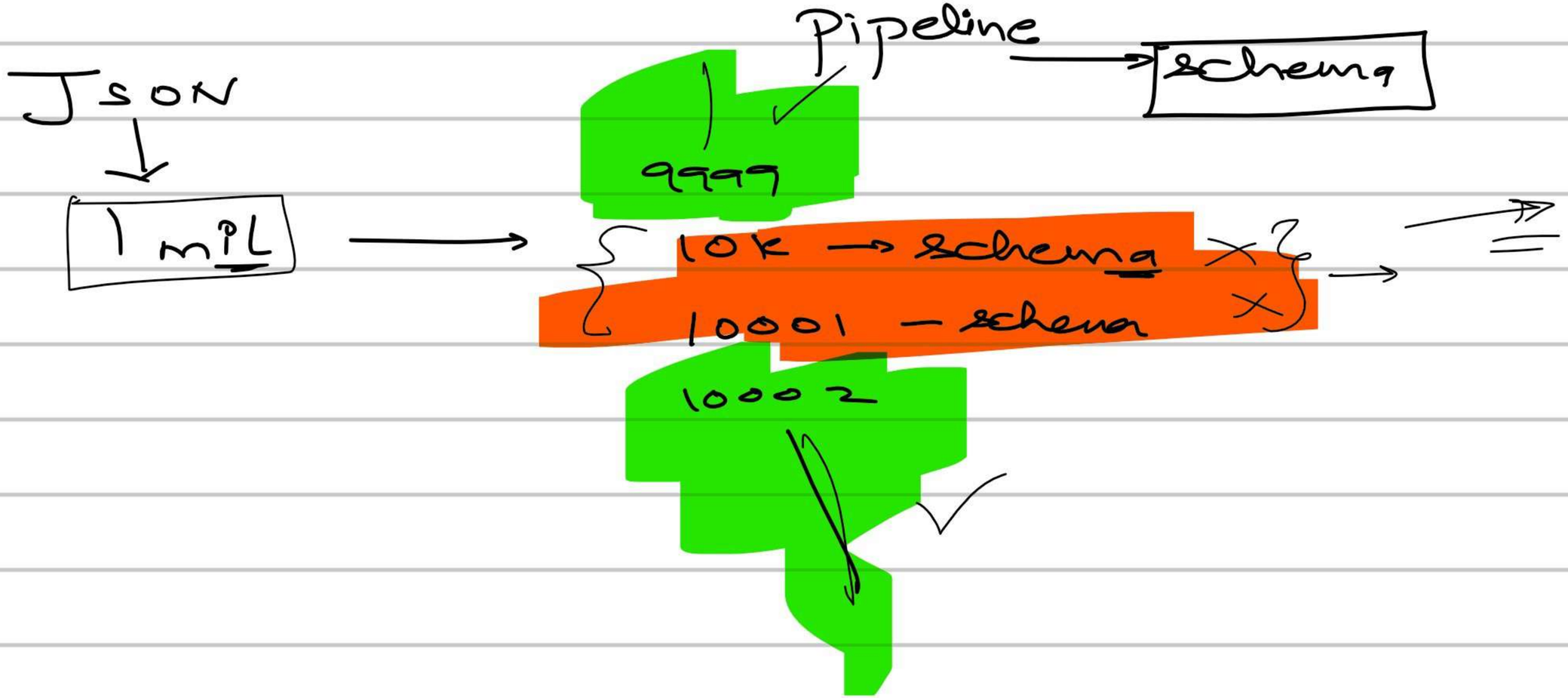
→

Logical Optimization
(Optimized Logical Plan)


Read File
10 million
Group By & Count
10K records
Filter
1 record
Read File
10 million
Filter
1000 records
Select only 1 column
1 column, 1000 records
Group By & Count
1 record
Analysis Phase
(Logical Plan)
Logical Optimization
(Optimized Logical Plan)



Analysis Phase (Logical Plan) → **Logical Optimization** (Optimized Logical Plan) → **Physical Planning** (Physical Plan)


Pipeline
schema
JSON
IMPL
9999
10K → schema
10001 – schema
10002

Performance Comparison

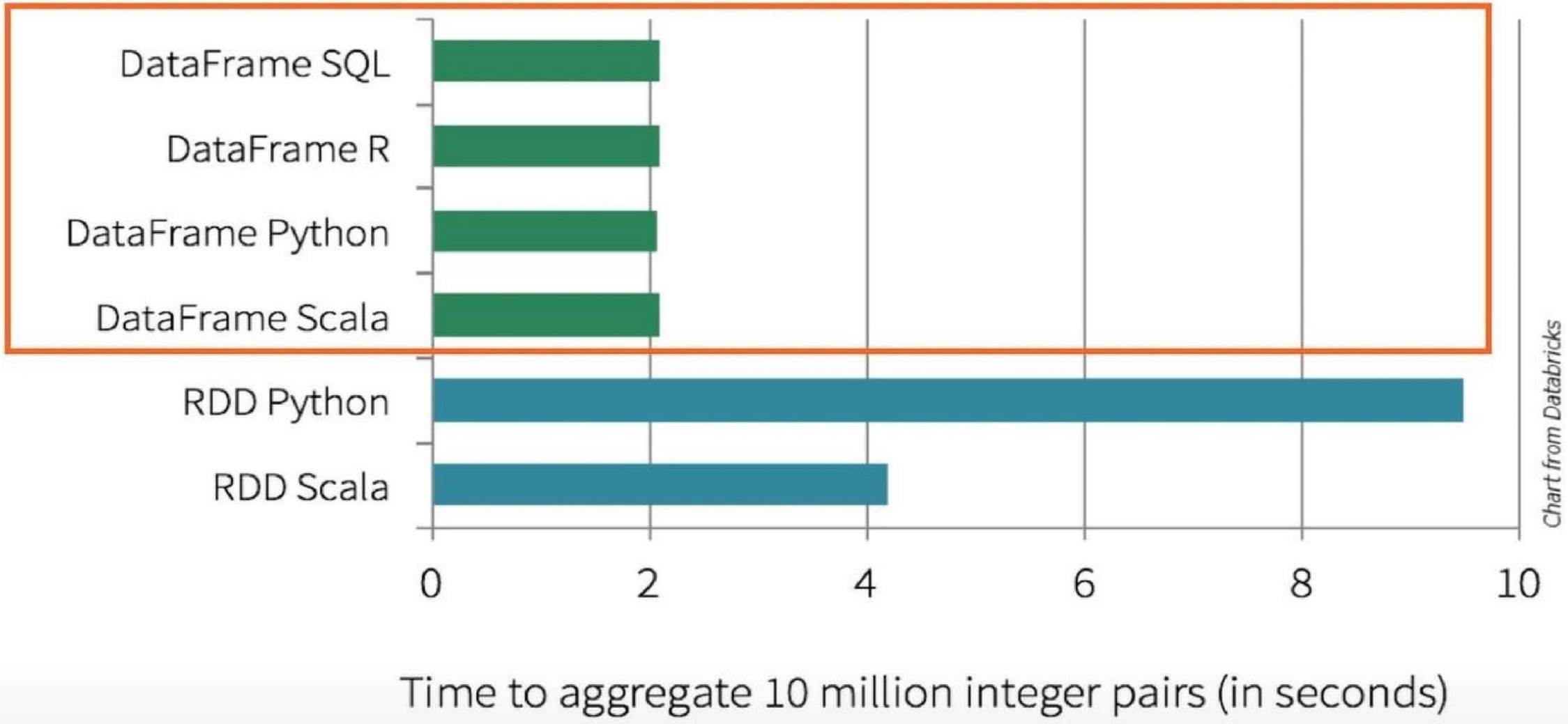


*Image Source: Databricks

Options to Create DataFrame

From RDD or Collection

Read from Data Source (file, database etc.)

From another DataFrame

In [1]:

Slide Type Slide

```
import findspark
findspark.init()
findspark.find()
```

Out[1]: 'E:\\Local d\\Zartab\\Apache Spark Installation\\spark-3.3.2-bin-hadoop3'

In [2]:

Slide Type Slide

```
from pyspark.sql import SparkSession
from pyspark.sql.types import *
from pyspark.sql.functions import *

spark = (
            SparkSession
                .builder

                .appName("TaxiOperationsDataFrameApp")
                .master("local[4]")

                .config("spark.dynamicAllocation.enabled", "false")
                .config("spark.sql.adaptive.enabled", "false")
                .getOrCreate()
        )

sc = spark.sparkContext

spark
```

Out[2]:

SparkSession - in-memory

SparkContext

Slide Type

Create DataFrame - Option 1.a: From RDD

In [4]:

Slide Type

```
# Create RDD

data = [
            [ 1, "Neha",  10000 ],
            [ 2, "Steve", 20000 ],
            [ 3, "Kari",  30000 ],
            [ 4, "Ivan",  40000 ],
            [ 5, "Mohit", 50000 ]
       ]

employeesRdd = sc.parallelize(data)
```

In [5]:

Slide Type

```
# Create DataFrame and show content

employeesDF = employeesRdd.toDF()

employeesDF.show()
```

```
+---+-----+-----+
| _1|   _2|   _3|
+---+-----+-----+
|  1| Neha|10000|
|  2|Steve|20000|
|  3| Kari|30000|
|  4| Ivan|40000|
|  5|Mohit|50000|
+---+-----+-----+
```

In [6]:

Slide Type

```
# Define column names for DataFrame

employeesDF = employeesDF.toDF("Id", "Name", "Salary")

employeesDF.show()
```

```
+---+-----+------+
| Id| Name|Salary|
+---+-----+------+
|  1| Neha| 10000|
|  2|Steve| 20000|
|  3| Kari| 30000|
|  4| Ivan| 40000|
|  5|Mohit| 50000|
+---+-----+------+
```

In [7]:

Slide Type

```
# Print DataFrame schema

employeesDF.printSchema()
```

```
root
 |-- Id: long (nullable = true)
 |-- Name: string (nullable = true)
 |-- Salary: long (nullable = true)
```

Create DataFrame - Option 1.b: From data collection

In [8]:

Slide Type

```
# Create DataFrame from collection

employeesDF = (
                    spark
                        .createDataFrame
                        (
                            data,                                   # Pass RDD or collection
                            "Id: long, Name: string, Salary: long"

                                                # Pass schema as array ["Id", "Name", "Salary"]
                        )
                )

employeesDF.show()
```

```
+---+-----+------+
| Id| Name|Salary|
```

Slide Type

Create DataFrame - Option 2: Read a File

In [10]:

Slide Type

```
# Read YellowTaxis csv file to create DataFrame

yellowTaxiDF = (
                  spark
                    .read
                    .csv("E:\SparkData\Raw\YellowTaxis_202210.csv")
                )
```

In [11]:

Slide Type

```
# Display DataFrame content

yellowTaxiDF.show()
```

In [11]:

Slide Type

```
# Display DataFrame content

yellowTaxiDF.show()
```

```
+--------+--------------------+--------------------+---------------+-------------+----------+---------
|     _c0|                 _c1|                 _c2|            _c3|          _c4|       _c5|
+--------+--------------------+--------------------+---------------+-------------+----------+---------
|VendorID|tpep_pickup_datetime|tpep_dropoff_date...|passenger_count|trip_distance|RatecodeID|store_and
|       1|2022-10-01T05:33:...|2022-10-01T05:48:...|            1.0|          1.7|       1.0|
|       2|2022-10-01T05:44:...|2022-10-01T05:49:...|            2.0|         0.72|       1.0|
|       2|2022-10-01T05:57:...|2022-10-01T06:07:...|            1.0|         1.74|       1.0|
|       1|2022-10-01T06:02:...|2022-10-01T06:08:...|            0.0|          1.3|       1.0|
|       1|2022-10-01T06:14:...|2022-10-01T06:20:...|            0.0|          1.0|       1.0|
|       1|2022-10-01T05:52:...|2022-10-01T06:22:...|            1.0|          6.8|       1.0|
|       2|2022-10-01T06:03:...|2022-10-01T06:14:...|            3.0|         1.88|       1.0|
|       1|2022-10-01T05:32:...|2022-10-01T06:20:...|            1.0|         12.2|       1.0|
|       2|2022-10-01T05:36:...|2022-10-01T05:54:...|            1.0|         7.79|       1.0|
|       2|2022-10-01T05:59:...|2022-10-01T06:13:...|            1.0|         4.72|       1.0|
|       1|2022-10-01T05:31:...|2022-10-01T05:50:...|            1.0|          8.8|       1.0|
|       1|2022-10-01T05:57:...|2022-10-01T06:29:...|            1.0|          8.6|       1.0|
|       1|2022-10-01T05:35:...|2022-10-01T06:05:...|            4.0|         12.0|       1.0|
```

In [13]:

```
# Take column names from file header row

yellowTaxiDF = (
                  spark
                    .read

                    .option("header", "true")

                    .csv("E:\SparkData\Raw\YellowTaxis_202210.csv")
                )

yellowTaxiDF.printSchema()
```

```
root
 |-- VendorID: string (nullable = true)
 |-- tpep_pickup_datetime: string (nullable = true)
 |-- tpep_dropoff_datetime: string (nullable = true)
 |-- passenger_count: string (nullable = true)
 |-- trip_distance: string (nullable = true)
 |-- RatecodeID: string (nullable = true)
```

In [14]:

Slide

```
# Use tab delimiter to read GreenTaxis file

greenTaxiDF = (
                  spark
                    .read
                    .option("header", "true")

                    .option("delimiter", "\t")

                    .csv("E:\SparkData\Raw\GreenTaxis_*.csv")
               )

greenTaxiDF.show()
```

```
+--------+--------------------+---------------------+---------------+-------------+-----
|VendorId|lpep_pickup_datetime|lpep_dropoff_datetime|passenger_count|trip_distance|Ratec
+--------+--------------------+---------------------+---------------+-------------+-----
|       2|2022-10-01T06:08:...| 2022-10-01T06:21:...|            1.0|         2.47|
|       2|2022-10-01T06:36:...| 2022-10-01T06:42:...|            1.0|         1.96|
```

Read JSON File

```
# Read PaymentTypes JSON file

paymentTypesDF = (
                        spark
                          .read
                          .json("E:\SparkData\Raw\PaymentTypes.json")
                    )

#paymentTypesDF = (
#                       spark
#                         .read

#                         .format("json")
#                         .load("C:\SparkCourse\DataFiles\Raw\PaymentTypes.json")
#                    )

paymentTypesDF.show()
```

```
+-----------+-------------+
```

```
#

paymentTypesDF.show()
```

```
+-----------+--------------+
|PaymentType|PaymentTypeID|
+-----------+--------------+
|Credit Card|             1|
|       Cash|             2|
|  No Charge|             3|
|    Dispute|             4|
|    Unknown|             5|
|Voided Trip|             6|
+-----------+--------------+
```

Schema Option 1 - No schema inference or definition

Check for jobs

In [16]:

```
# Read YellowTaxis csv file to create DataFrame

yellowTaxiDF = (
                  spark
                    .read
                    .option("header", "true")
                    .csv("E:\SparkData\Raw\YellowTaxis_202210.csv")
                )

yellowTaxiDF.printSchema()
```

```
root
 |-- VendorID: string (nullable = true)
 |-- tpep_pickup_datetime: string (nullable = true)
 |-- tpep_dropoff_datetime: string (nullable = true)
 |-- passenger_count: string (nullable = true)
```

Inferring Schema

Id	Name	Date	Amount
1	Neha	2023-01-01	372
2	Cathy	2023-01-12	293
3	Ivan	2023-01-03	938
4	Kerri	2023-01-17	220
5	Mohit	2023-01-02	494

Great for development or when incoming files are dynamic

Challenges

- Scanning big datasets will take lot of time
- Corrupt files will cause failures, difficult to debug

Recommended to manually define schema for DataFrames in production

Schema Option 2 - Infer schema

Check for jobs

```
# Read YellowTaxis csv file, and create DataFrame by inferring the sch

yellowTaxiDF = (
                  spark
                    .read
                    .option("header", "true")

                    .option("inferSchema", "true")

                    .csv("E:\SparkData\Raw\YellowTaxis_202210.csv")
                )

yellowTaxiDF.printSchema()
```

```
root
 |-- VendorID: integer (nullable = true)
 |-- tpep_pickup_datetime: timestamp (nullable = true)
```

Schema Option 3 - Define schema & apply

Check for jobs

Slide Type

```
# Create schema for Yellow Taxi Data

yellowTaxiSchema = (
                        StructType
                        ([
                            StructField("VendorId"                , IntegerType()   , True),
                            StructField("lpep_pickup_datetime"    , TimestampType() , True),
                            StructField("lpep_dropoff_datetime"   , TimestampType() , True),
                            StructField("passenger_count"         , DoubleType()    , True),
                            StructField("trip_distance"           , DoubleType()    , True),
                            StructField("RatecodeID"              , DoubleType()    , True),
                            StructField("store_and_fwd_flag"      , StringType()    , True),
                            StructField("PULocationID"            , IntegerType()   , True),
                            StructField("DOLocationID"            , IntegerType()   , True),
                            StructField("payment_type"            , IntegerType()   , True),
                            StructField("fare_amount"             , DoubleType()    , True),
```

```
            StructField("DOLocationID"            , IntegerType()   , True),
            StructField("payment_type"            , IntegerType()   , True),
            StructField("fare_amount"             , DoubleType()    , True),
            StructField("extra"                   , DoubleType()    , True),
            StructField("mta_tax"                 , DoubleType()    , True),
            StructField("tip_amount"              , DoubleType()    , True),
            StructField("tolls_amount"            , DoubleType()    , True),
            StructField("improvement_surcharge"   , DoubleType()    , True),
            StructField("total_amount"            , DoubleType()    , True),
            StructField("congestion_surcharge"    , DoubleType()    , True),
            StructField("airport_fee"             , DoubleType()    , True)
        ])
    )
```

In [19]:

```
# Read YellowTaxis csv file, and create DataFrame by applying the schema

yellowTaxiDF = (
                  spark
                    .read
                    .option("header", "true")

                    .schema(yellowTaxiSchema)

                    .csv("E:\SparkData\Raw\YellowTaxis_202210.csv")
                )

yellowTaxiDF.printSchema()
```

```
root
 |-- VendorId: integer (nullable = true)
 |-- lpep_pickup_datetime: timestamp (nullable = true)
 |-- lpep_dropoff_datetime: timestamp (nullable = true)
 |-- passenger_count: double (nullable = true)
 |-- trip_distance: double (nullable = true)
 |-- RatecodeID: double (nullable = true)
```

Define schema for JSON file

```
# Read JSON file

taxiBasesDF = (
                  spark
                    .read
                    .json("E:\SparkData\Raw\TaxiBases.json")
                )

taxiBasesDF.show()
```

```
---------------------------------------------------------------------------
isException                               Traceback (most recent call last)
n[20], line 9
1 # Read JSON file
3 taxiBasesDF = (
4                   spark
```

isException:

Spark 2.3, the queries from raw JSON/CSV files are disallowed when the
nced columns only include the internal corrupt record column
_corrupt_record by default). For example:

```
read.schema(schema).csv(file).filter($"_corrupt_record".isNotNull).count()
ark.read.schema(schema).csv(file).select("_corrupt_record").show().
```

d, you can cache or save the parsed results and then send the same query.

```
ample, val df = spark.read.schema(schema).csv(file).cache() and then
ter($"_corrupt_record".isNotNull).count().
```

Slide Type

```
# Read multiline JSON file

taxiBasesDF = (
                  spark
                    .read

                    .option("multiline", "true")

                    .json("E:\SparkData\Raw\TaxiBases.json")
               )

taxiBasesDF.show(truncate=False)
```

```
+-----------------------------------------------------------+----------+-----------------------------
|Address                                                    |Date      |Entity Name
+-----------------------------------------------------------+----------+-----------------------------
|{636, NEW YORK, 10001, NY, WEST   28 STREET}               |08/15/2019|VIER-NY,LLC
|{131, BRONX, 10468, NY, KINGSBRIDGE ROAD}                  |08/15/2019|VETERANS RADIO DISPATCHER CORP
|{115-54, ELMONT, 11003, NY, 238 STREET}                    |08/15/2019|ALPHA VAN LINE
|{866, BROOKLYN, 11208, NY, NEW LOTS AVENUE}                |08/15/2019|A.T.B. CAR AND LIMOUSINE SERVI
|{57-48, MASPETH, 11378, NY, MASPETH AVENUE}                |08/15/2019|KYOEI LIMOUSINE, INC.
|{31-00, LIC, 11101, NY, 47 AVENUE   SUITE # 4123A}         |08/15/2019|ENDOR CAR & DRIVER,LLC.
```

```python
# Check the complex structure of JSON data

taxiBasesDF.printSchema()
```

```
root
 |-- Address: struct (nullable = true)
 |    |-- Building: string (nullable = true)
 |    |-- City: string (nullable = true)
 |    |-- Postcode: long (nullable = true)
 |    |-- State: string (nullable = true)
 |    |-- Street: string (nullable = true)
 |-- Date: string (nullable = true)
 |-- Entity Name: string (nullable = true)
 |-- GeoLocation: struct (nullable = true)
 |    |-- Latitude: double (nullable = true)
 |    |-- Location: string (nullable = true)
 |    |-- Longitude: double (nullable = true)
 |-- License Number: string (nullable = true)
 |-- SHL Endorsed: string (nullable = true)
 |-- Telephone Number: long (nullable = true)
 |-- Time: string (nullable = true)
 |-- Type of Base: string (nullable = true)
```

Slide Type

```
# Define schema for JSON data

taxiBasesSchema = (
                    StructType
                    ([
                        StructField("License Number"         , StringType()    , True),
                        StructField("Entity Name"            , StringType()    , True),
                        StructField("Telephone Number"       , LongType()      , True),
                        StructField("SHL Endorsed"           , StringType()    , True),
                        StructField("Type of Base"           , StringType()    , True),

                        StructField("Address",
                                        StructType
                                        ([
                                            StructField("Building"   , StringType(),   True),
                                            StructField("Street"     , StringType(),   True),
                                            StructField("City"       , StringType(),   True),
                                            StructField("State"      , StringType(),   True),
                                            StructField("Postcode"   , StringType(),   True)
                                        ]),
                                    True
                                    )
```

```
                StructField("Postcode"    , StringType(),   True)
                ]),
            True
            ),

        StructField("GeoLocation",
                StructType
                ([
                    StructField("Latitude"   , StringType(),   True),
                    StructField("Longitude"  , StringType(),   True),
                    StructField("Location"   , StringType(),   True)
                ]),
            True
            )
    ])
)
```

Slide Type

```python
# Read JSON file by applying nested schema

taxiBasesDF = (
                  spark
                    .read
                    .option("multiline", "true")

                    .schema(taxiBasesSchema)

                    .json("E:\SparkData\Raw\TaxiBases.json")
               )

taxiBasesDF.show(truncate=False)
```

```
|B02844          |ENDOR CAR & DRIVER,LLC.                    |4154758459
```

Analyze Data

```
yellowTaxiAnalyzedDF = (
                            yellowTaxiDF.describe
                            (
                                "passenger_count",
                                "trip_distance"
                            )
                        )

yellowTaxiAnalyzedDF.show()
```

```
+-------+------------------+-----------------+
|summary|   passenger_count|    trip_distance|
+-------+------------------+-----------------+
|  count|           3542392|          3675412|
|   mean|1.3846934500755421|6.206976298167358|
| stddev|0.9302303297407405|640.8236808320215|
```

Data Quality Checks

Completeness

- Remove null or missing data
- Fill missing values with placeholder

Uniqueness

- Remove duplicate records

Timeliness

- Appropriate date range

Accuracy

- Remove inaccurate data

1. Accuracy Check: Filter inaccurate data

```
# Display the count before operation
print("Before operation = " + str( yellowTaxiDF.count()) )


yellowTaxiDF = (
                    yellowTaxiDF

                        .where("passenger_count > 0")

                        .filter(col("trip_distance") > 0.0)
                  )


# Display the count after operation
print("After operation = " + str( yellowTaxiDF.count()) )
```

```
Before operation = 3675412
After operation = 3422296
```

2.a. Completeness Check: Drop rows with nulls

In [27]:

```
# Display the count before operation
print("Before operation = " + str( yellowTaxiDF.count()) )


yellowTaxiDF = (
                    yellowTaxiDF
                           .na.drop('all')
                 )


# Display the count after operation
print("After operation = " + str( yellowTaxiDF.count()) )
```

```
Before operation = 3422296
After operation = 3422296
```

2.b. Completeness Check: Replace nulls with default values

]:

```
defaultValueMap = {'payment_type': 5, 'RateCodeID': 1}


yellowTaxiDF = (
                    yellowTaxiDF
                        .na.fill(defaultValueMap)
                )
```

3. Uniqueness Check: Drop duplicate rows

```
# Display the count before operation
print("Before operation = " + str( yellowTaxiDF.count()) )


yellowTaxiDF = (
                    yellowTaxiDF
                           .dropDuplicates()
                )


# Display the count after operation
print("After operation = " + str( yellowTaxiDF.count()) )
```

```
Before operation = 3422296
After operation = 3422295
```

Slide Type

4. Timeliness Check: Remove records outside the bound

Slide Type

```
# Display the count before operation
print("Before operation = " + str( yellowTaxiDF.count()) )


yellowTaxiDF = (

     yellowTaxiDF
        .where("lpep_pickup_datetime >= '2022-10-01' AND lpep_dropoff_datetime < '2022-11-01'")
)


# Display the count after operation
print("After operation = " + str( yellowTaxiDF.count()) )
```

```
Before operation = 3422295
After operation = 3393897
```

Chain all cleanup operation together

```
defaultValueMap = {'payment_type': 5, 'RateCodeID': 1}

# Read file
yellowTaxiDF = (
                  spark
                    .read
                    .option("header", "true")
                    .schema(yellowTaxiSchema)
                    .csv("E:\SparkData\Raw\YellowTaxis_202210.csv")
                )
```

```
                    .csv("E:\SparkData\Raw\YellowTaxis_202210.csv")
                )

# Cleanup data by applying data quality checks
yellowTaxiDF = (
                yellowTaxiDF

                    .where("passenger_count > 0")

                    .filter(col("trip_distance") > 0.0)

                    .na.drop('all')

                    .na.fill(defaultValueMap)

                    .dropDuplicates()

                    .where("lpep_pickup_datetime >= '2022-10-01' AND \"
                    lpep_dropoff_datetime < '2022-11-01'")
                )

# Display the count after operation
print("After operation = " + str( yellowTaxiDF.count()) )
```

```
yellowTaxiDF.printSchema()
```

```
root
 |-- VendorId: integer (nullable = true)
 |-- lpep_pickup_datetime: timestamp (nullable = true)
 |-- lpep_dropoff_datetime: timestamp (nullable = true)
 |-- passenger_count: double (nullable = true)
 |-- trip_distance: double (nullable = true)
 |-- RatecodeID: double (nullable = false)
 |-- store_and_fwd_flag: string (nullable = true)
 |-- PULocationID: integer (nullable = true)
 |-- DOLocationID: integer (nullable = true)
 |-- payment_type: integer (nullable = false)
 |-- fare_amount: double (nullable = true)
 |-- extra: double (nullable = true)
 |-- mta_tax: double (nullable = true)
 |-- tip_amount: double (nullable = true)
 |-- tolls_amount: double (nullable = true)
 |-- improvement_surcharge: double (nullable = true)
 |-- total_amount: double (nullable = true)
```

Transform Data

1. Select limited columns

```
yellowTaxiDF = (
                    yellowTaxiDF

                         # Select only limited columns
                         .select(
                                   "VendorID",

                                   col("passenger_count").cast(IntegerType()),

                                   column("trip_distance").alias("TripDistance"),
```

```
yellowTaxiDF = (
                    yellowTaxiDF

                         # Select only limited columns
                         .select(
                                   "VendorID",

                                   col("passenger_count").cast(IntegerType()),

                                   column("trip_distance").alias("TripDistance"),

                                   yellowTaxiDF.lpep_pickup_datetime,

                                   "lpep_dropoff_datetime",
                                   "PULocationID",
                                   "DOlocationID",
                                   "RatecodeID",
                                   "total_amount",
                                   "payment_type"
                              )

                         # Don't run, since airport_fee has not been selected above
                         # .drop("airport_fee")
                )

yellowTaxiDF.printSchema()
```

2. Rename columns

```
yellowTaxiDF = (
                    yellowTaxiDF

                         .withColumnRenamed("passenger_count", "PassengerCount")

                         .withColumnRenamed("lpep_pickup_datetime", "PickupTime")
                         .withColumnRenamed("lpep_dropoff_datetime", "DropTime")
                         .withColumnRenamed("PUlocationID", "PickupLocationId")
                         .withColumnRenamed("DOlocationID", "DropLocationId")
                         .withColumnRenamed("total_amount", "TotalAmount")
                         .withColumnRenamed("payment_type", "PaymentType")
                )

yellowTaxiDF.printSchema()
```

```
root
 |-- VendorID: integer (nullable = true)
```

3.a. Create derived columns - TripYear, TripMonth, TripDay

```
# Create derived columns for year, month and day
yellowTaxiDF = (
                  yellowTaxiDF

                        .withColumn("TripYear", year(col("PickupTime")))

                        .select(
                                    "*",

                                    expr("month(PickupTime) AS TripMonth"),

                                    dayofmonth(col("PickupTime")).alias("TripDay")
                                )
                )

yellowTaxiDF.printSchema()
```

```
root
 |-- VendorID: integer (nullable = true)
```

3.b. Create derived column - TripTimeInMinutes

Slide Type

```
# Option 1
yellowTaxiDF = (
                  yellowTaxiDF
                        .withColumn("TripTimeInMinutes",
                                        round(
                                                (unix_timestamp(col("DropTime"))
                                                     - unix_timestamp(col("PickupTime")))

                                                  / 60
                                            )
                                      )
               )

yellowTaxiDF.printSchema()
```

```
root
 |-- VendorID: integer (nullable = true)
 |-- PassengerCount: integer (nullable = true)
 |-- TripDistance: double (nullable = true)
```

Slide Type

```python
# Option 2

tripTimeInSecondsExpr = unix_timestamp(col("DropTime")) - unix_timestamp(col("PickupTime"))


tripTimeInMinutesExpr = round(tripTimeInSecondsExpr / 60)


yellowTaxiDF = (
                  yellowTaxiDF
                        .withColumn("TripTimeInMinutes", tripTimeInMinutesExpr)
                )

yellowTaxiDF.printSchema()
```

```
root
 |-- VendorID: integer (nullable = true)
 |-- PassengerCount: integer (nullable = true)
 |-- TripDistance: double (nullable = true)
 |-- PickupTime: timestamp (nullable = true)
 |-- DropTime: timestamp (nullable = true)
 |-- PickupLocationId: integer (nullable = true)
```

3.c. Create derived column - TripType

```
tripTypeColumn = (
                    when(
                            col("RatecodeID") == 6,
                              "SharedTrip"
                         )
                    .otherwise("SoloTrip")
                  )


yellowTaxiDF = (
                  yellowTaxiDF

                       .withColumn("TripType", tripTypeColumn)

                       .drop("RatecodeID")
                 )

yellowTaxiDF.printSchema()
```

Check Execution Plans

```
yellowTaxiDF.explain( mode = "extended" )

# Other modes - simple, codegen, cost, formatted
```

```
== Parsed Logical Plan ==
Project [VendorID#1218, PassengerCount#1371, TripDistance#1359, PickupTime#138
+- Project [VendorID#1218, PassengerCount#1371, TripDistance#1359, PickupTime#
   +- Project [VendorID#1218, PassengerCount#1371, TripDistance#1359, PickupTi
      +- Project [VendorID#1218, PassengerCount#1371, TripDistance#1359, Picku
         +- Project [VendorID#1218, PassengerCount#1371, TripDistance#1359, Pi
            +- Project [VendorID#1218, PassengerCount#1371, TripDistance#1359,
               +- Project [VendorID#1218, PassengerCount#1371, TripDistance#13
                  +- Project [VendorID#1218, PassengerCount#1371, TripDistance
                     +- Project [VendorID#1218, PassengerCount#1371, TripDista
                        +- Project [VendorID#1218, PassengerCount#1371, TripDi
```

Read JSON file using schema

Slide Type

```
taxiBasesSchema = (
                    StructType
                    ([
                        StructField("License Number"         , StringType()    , True),
                        StructField("Entity Name"            , StringType()    , True),
                        StructField("Telephone Number"       , LongType()      , True),
                        StructField("SHL Endorsed"           , StringType()    , True),
                        StructField("Type of Base"           , StringType()    , True),

                        StructField("Address",
                                        StructType
                                        ([
                                            StructField("Building"   , StringType(),   True),
                                            StructField("Street"     , StringType(),   True),
                                            StructField("City"       , StringType(),   True),
                                            StructField("State"      , StringType(),   True),
                                            StructField("Postcode"   , StringType(),   True)
                                        ]),
```

```
                                StructField("Building"   , StringType(),   True),
                                StructField("Street"     , StringType(),   True),
                                StructField("City"       , StringType(),   True),
                                StructField("State"      , StringType(),   True),
                                StructField("Postcode"   , StringType(),   True)
                            ]),
                        True
                    ),

            StructField("GeoLocation",
                            StructType
                            ([
                                StructField("Latitude"   , StringType(),   True),
                                StructField("Longitude"  , StringType(),   True),
                                StructField("Location"   , StringType(),   True)
                            ]),
                        True
                    )
        ])
    )
```

Slide Type

```
taxiBasesDF = (
                  spark
                    .read
                    .option("multiline", "true")

                    .schema(taxiBasesSchema)

                    .json("E:\SparkData\Raw\TaxiBases.json")
               )

taxiBasesDF.show(truncate=False)
```

```
+--------------+-------------------------------------+----------------+------------+-----------------
|License Number|Entity Name                          |Telephone Number|SHL Endorsed|Type of Base
+--------------+-------------------------------------+----------------+------------+-----------------
|B02865        |VIER-NY,LLC                          |6466657536      |No          |BLACK CAR BASE
|B02634        |VETERANS RADIO DISPATCHER CORP.      |7183647878      |No          |LIVERY BASE
|B80094        |ALPHA VAN LINE                       |5162850750      |No          |COMMUTER VAN AUTH
|B02677        |A.T.B. CAR AND LIMOUSINE SERVICE, INC.|7184854444      |No          |LIVERY BASE
|B02152        |KYOEI LIMOUSINE, INC.                |7183263258      |No          |LUXURY/LIMOUSINE
|B02844        |ENDOR CAR & DRIVER,LLC.              |4154758459      |No          |BLACK CAR BASE
```

Extract nested fields from JSON

Slide Type

```
taxiBasesFlatDF = (
                        taxiBasesDF
                            .select(
                                        col("License Number").alias("BaseLicenseNumber"),
                                        col("Entity Name").alias("EntityName"),

                                        col("Address.Building").alias("AddressBuilding"),

                                        col("Address.Street").alias("AddressStreet"),
                                        col("Address.City").alias("AddressCity"),
                                        col("Address.State").alias("AddressState"),
                                        col("Address.Postcode").alias("AddressPostCode"),

                                        col("GeoLocation.Latitude").alias("GeoLatitude"),
                                        col("GeoLocation.Longitude").alias("GeoLongitude")
                                    )
                    )

taxiBasesFlatDF.show()
```

Aggregate Data

Slide Type

```
yellowTaxiDFReport = (
                        yellowTaxiDF

                               .groupBy( "PickupLocationId", "DropLocationId" )

                               .agg(
                                       avg("TripTimeInMinutes").alias("AvgTripTime"),

                                       sum("TotalAmount").alias("SumAmount")
                                   )

                               .orderBy( col("PickupLocationId").desc() )
                     )

yellowTaxiDFReport.show()
```

```
+----------------+--------------+------------------+------------------+
|PickupLocationId|DropLocationId|       AvgTripTime|         SumAmount|
```

Exercises

1. Based on PickupTime, add a new column with day text = Monday to Sunday
2. Based on PickupTime, add a new column with month text = January to December
3. Based on PickupTime, add a new column with value being the last day of month

Check out full list of functions on following link

https://spark.apache.org/docs/latest/api/python/reference/pyspark.sql/functions.html

```
# Reduce number of DataFrame partitions to 4

yellowTaxiDF = yellowTaxiDF.coalesce(4)

yellowTaxiDF.rdd.getNumPartitions()
```

```
4
```

Slic

Save data in CSV format to storage

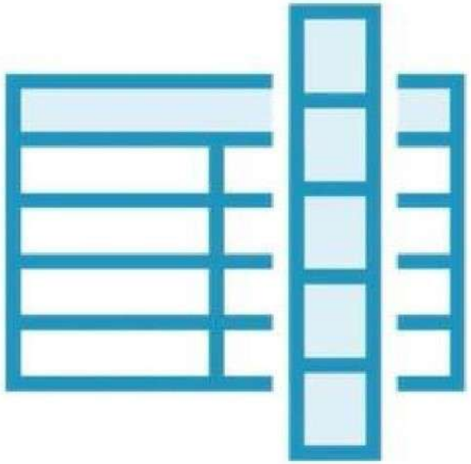
Slic

```
(
    yellowTaxiDF
            .write

            .option("header", "true")
            .option("dateFormat", "yyyy-MM-dd HH:mm:ss.S")

            .mode("overwrite")    # Options - Append, ErrorIfExists, Ignore, Overwrite

            .csv("E:\SparkData\Raw\YellowTaxisOutput.csv")
)
```

Slic

Append	-	Append DataFrame content to existing location, or create new one
ErrorIfExists	-	Throw error if files already exist
Ignore	-	If files already exist, don't do anything
Overwrite	-	Remove existing files and add new ones

Apache Parquet



- Columnar storage format
- Support complex nested data structures
- Stores schema in the file itself
- Supports efficient compression & encoding
- Binary files
- Querying is much faster than CSV/JSON

Save data in Parquet format to storage

```
(
    yellowTaxiDF
            .write

            .option("header", "true")
            .option("dateFormat", "yyyy-MM-dd HH:mm:ss.S")

            .mode("overwrite")

            .parquet("E:\SparkData\Raw\YellowTaxisOutput.parquet")
)
```

Save partitioned data in Parquet format to storage

```
(
    yellowTaxiDF
            .write

            .partitionBy("VendorId")

            .option("header", "true")
            .option("dateFormat", "yyyy-MM-dd HH:mm:ss.S")

            .mode("overwrite")

            .parquet("E:\SparkData\Raw\YellowTaxisPartitionedOutput.parquet")
)
```

Check performance between non-partitioned & partitioned datasets

1. Performance check: Run query on non-partitioned dataset

```
# Run query on non-partitioned dataset

outputDF = (
                spark
                    .read
                    .parquet("E:\SparkData\Raw\YellowTaxisOutput.parquet")

                    .where("VendorId = 1")

                    .groupBy("PickupLocationId")
                    .agg(avg("TotalAmount"))
            )

outputDF.show()
```

2. Performance check: Run query on partitioned dataset

Slide T

```
outputDF = (
                spark
                    .read
                    .parquet("E:\SparkData\Raw\YellowTaxisPartitionedOutput.parquet")

                    .where("VendorId = 1")

                    .groupBy("PickupLocationId")
                    .agg(avg("TotalAmount"))
            )

outputDF.show()
```

```
+----------------+------------------+
|PickupLocationId|  avg(TotalAmount)|
+----------------+------------------+
```

Options to Handle Corrupt Data

Text formats (CSV and JSON) have built-in support for handling corrupt records

Support 3 Parse Modes while reading data

- Permissive
- DropMalformed
- FailFast

PERMISSIVE : sets other fields to null when it meets a corrupted record, and puts the malformed string into a new field configured by columnNameOfCorruptRecord. When a schema is set by user, it sets null for extra fields.

DROPMALFORMED : ignores the whole corrupted records.

FAILFAST : throws an exception when it meets corrupted records.

In [1]:
```
import findspark
findspark.init()
findspark.find()
```

Out[1]:
```
'E:\\Local d\\Zartab\\Apache Spark Installation\\spark-3.3.2-bin-hadoop3'
```

In [2]:
```
from pyspark.sql import SparkSession
from pyspark.sql.types import *
from pyspark.sql.functions import *

spark = (
            SparkSession
                .builder
                .appName("ErrorHandlingApp")
                .master("local[4]")

                .config("spark.dynamicAllocation.enabled", "false")
                .config("spark.sql.adaptive.enabled", "false")
                .getOrCreate()
        )
```

E: > SparkData > Raw > {..} RateCodes.json > ...

```
1  {"RateCodeID":1,"RateCode":"Standard rate"}
2  {"RateCodeID":2,"RateCode":"JFK"}
3  {"RateCodeID":3,"RateCode":"Newark"}
4  {"RateCodeID":},"RateCode":"Westchester"}
5  {"RateCodeID":5,"RateCode":"Negotiated fare"}
6  {"RateCodeID":6,"RateCode":"Group ride"}
7
8
```

Read JSON file with default Parse Mode

```
rateCodesJsonDF_modes = (
                            spark
                              .read
                              .json("E:\SparkData\Raw\RateCodes.json")
                        )

rateCodesJsonDF_modes.show(truncate=False)
```

```
+---------------+----------+-----------------------------------------+
|RateCode       |RateCodeID|_corrupt_record                          |
+---------------+----------+-----------------------------------------+
|Standard rate  |1         |null                                     |
|JFK            |2         |null                                     |
|Newark         |3         |null                                     |
|null           |null      |{"RateCodeID":},"RateCode":"Westchester"}|
|Negotiated fare|5         |null                                     |
|Group ride     |6         |null                                     |
+---------------+----------+-----------------------------------------+
```

Read JSON file with Permissive mode

```
rateCodesJsonDF_modes = (
                                spark
                                  .read

                                  .option("mode", "PERMISSIVE")

                                  .json("E:\SparkData\Raw\RateCodes.json")
                            )

rateCodesJsonDF_modes.show(truncate=False)
```

```
+---------------+----------+-----------------------------------------+
|RateCode       |RateCodeID|_corrupt_record                          |
+---------------+----------+-----------------------------------------+
|Standard rate  |1         |null                                     |
|JFK            |2         |null                                     |
|Newark         |3         |null                                     |
|null           |null      |{"RateCodeID":},"RateCode":"Westchester"}|
|Negotiated fare|5         |null                                     |
|Group ride     |6         |null                                     |
+---------------+----------+-----------------------------------------+
```

```python
rateCodesJsonDF_modes = (
                            spark
                              .read

                              .option("mode", "PERMISSIVE")

                              .option("columnNameOfCorruptRecord", "CorruptData")

                              .json("E:\SparkData\Raw\RateCodes.json")
                          )

rateCodesJsonDF_modes.show(truncate=False)
```

```
+-----------------------------------------+---------------+----------+
|CorruptData                              |RateCode       |RateCodeID|
+-----------------------------------------+---------------+----------+
|null                                     |Standard rate  |1         |
|null                                     |JFK            |2         |
|null                                     |Newark         |3         |
|{"RateCodeID":},"RateCode":"Westchester"}|null           |null      |
```

```python
rateCodesJsonDF_modes = (
                            spark
                              .read

                              .option("mode", "PERMISSIVE")

                              .option("columnNameOfCorruptRecord", "RateCodeID")

                              .json("E:\SparkData\Raw\RateCodes.json")
                          )

rateCodesJsonDF_modes.show(truncate=False)
```

```
+---------------+----------------------------------------+
|RateCode       |RateCodeID                              |
+---------------+----------------------------------------+
|Standard rate  |null                                    |
|JFK            |null                                    |
|Newark         |null                                    |
|null           |{"RateCodeID":},"RateCode":"Westchester"}|
|Negotiated fare|null                                    |
```

Read JSON file with DropMalformed mode

```
rateCodesJsonDF_modes = (
                                spark
                                  .read

                                  .option("mode", "DROPMALFORMED")

                                  .json("E:\SparkData\Raw\RateCodes.json")
                              )

rateCodesJsonDF_modes.show(truncate=False)
```

```
+---------------+----------+
|RateCode       |RateCodeID|
+---------------+----------+
|Standard rate  |1         |
|JFK            |2         |
|Newark         |3         |
|Negotiated fare|5         |
```

Read JSON file with FailFast mode

```
rateCodesJsonDF_modes = (
                            spark
                              .read

                              .option("mode", "FAILFAST")

                              .json("E:\SparkData\Raw\RateCodes.json")
                          )

rateCodesJsonDF_modes.show(truncate=False)
```

```
10 rateCodesJsonDF_modes.show(truncate=False)

File E:\Local d\Zartab\Apache Spark Installation\spark-3.3.2-bin-hadoop3\python
ter.py:284, in DataFrameReader.json(self, path, schema, primitivesAsString, pre
mments, allowUnquotedFieldNames, allowSingleQuotes, allowNumericLeadingZero, al
AnyCharacter, mode, columnNameOfCorruptRecord, dateFormat, timestampFormat, mul
```

E: > SparkData > Raw > RateCodes.csv

```
RateCodeID,RateCode,IsApproved
1,Standard Rate,Yes
2,JFK,Yes
3,Newark,Yes
4,
Westchester,Yes
5,Negotiated fare,
6,GroupRide,Yes
```

Read CSV file with default Permissive mode

```
rateCodesCsvDF_modes = (
                            spark
                              .read
                              .option("header", "true")

                              .option("mode", "PERMISSIVE")

                              .csv("E:\SparkData\Raw\RateCodes.csv")
                          )

rateCodesCsvDF_modes.show(truncate=False)
```

```
+------------+----------------+-----------+
|RateCodeID  |RateCode        |IsApproved|
+------------+----------------+-----------+
|1           |Standard Rate   |Yes        |
|2           |JFK             |Yes        |
|3           |Newark          |Yes        |
|4           |null            |null       |
|Westchester|Yes              |null       |
```

Read CSV file with DropMalformed mode

```
rateCodesCsvDF_modes = (
                            spark
                              .read
                              .option("header", "true")

                              .option("mode", "DROPMALFORMED")

                              .csv("E:\SparkData\Raw\RateCodes.csv")
                         )

rateCodesCsvDF_modes.show(truncate=False)
```

```
+----------+---------------+----------+
|RateCodeID|RateCode       |IsApproved|
+----------+---------------+----------+
|1         |Standard Rate  |Yes       |
|2         |JFK            |Yes       |
|3         |Newark         |Yes       |
|5         |Negotiated fare|null      |
```

Read CSV file with FailFast mode

```
rateCodesCsvDF_modes = (
                            spark
                              .read
                              .option("header", "true")

                              .option("mode", "FAILFAST")

                              .csv("E:\SparkData\Raw\RateCodes.csv")
                          )

rateCodesCsvDF_modes.show(truncate=False)
```

```
        ... 1 more
Caused by: org.apache.spark.sql.catalyst.util.BadRecordException: java.lang.RuntimeE
rmed CSV record
        at org.apache.spark.sql.catalyst.csv.UnivocityParser.org$apache$spark$sql$ca
ocityParser$$convert(UnivocityParser.scala:330)
        at org.apache.spark.sql.catalyst.csv.UnivocityParser.$anonfun$parse$2(Univoc
a:275)
```

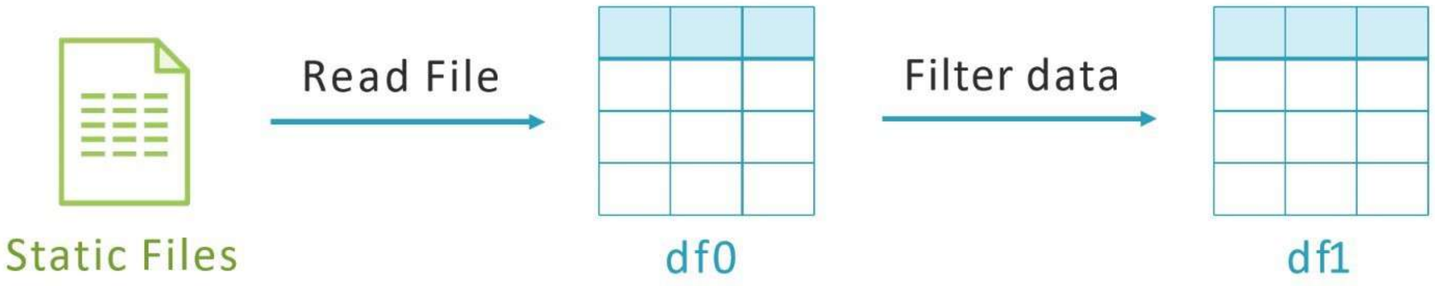
Working with Spark SQL, UDFs & Common DataFrame Operations

Overview

- Run SQL queries on DataFrames
- Work with Spark databases & tables
- Work with user-defined functions (UDFs)
- Perform operations on multiple datasets
- Perform window operations

Running SQL Queries on DataFrames

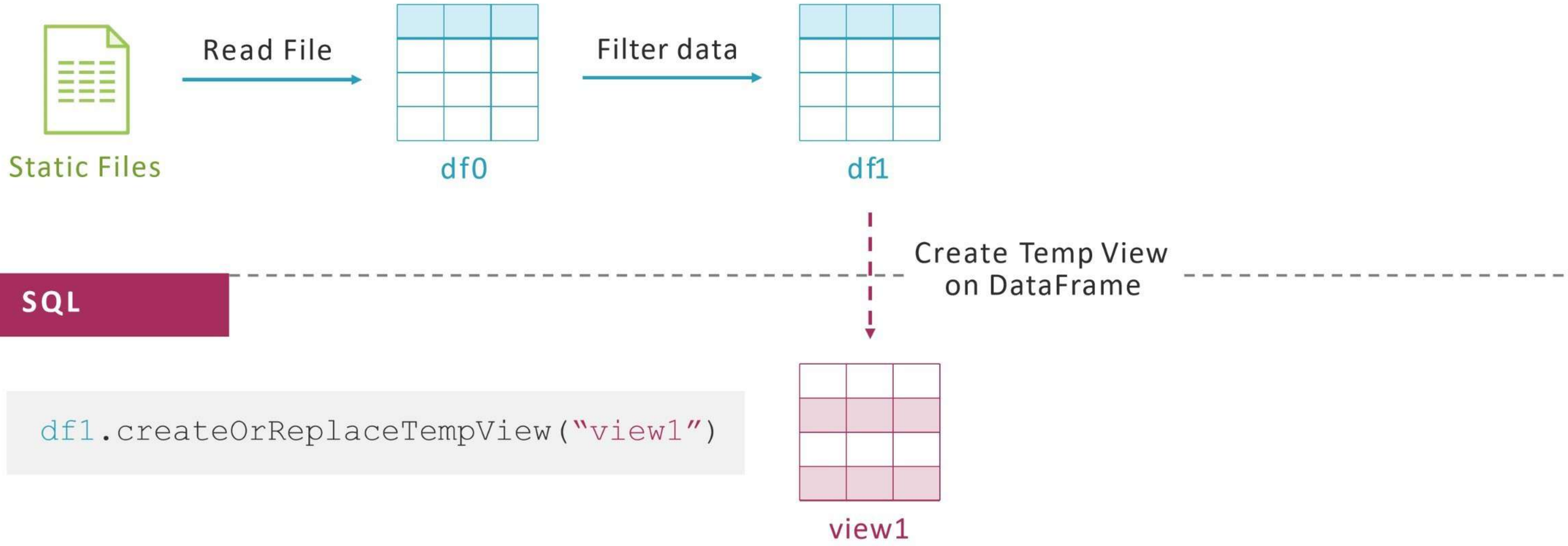
PySpark / Scala




```
df0 = spark
      .read
  .load (...)
```

```
df1 = df0
      .where (...)
```

PySpark / Scala



SQL

```
df1.createOrReplaceTempView("view1")
```

```
spark.sql( "SELECT * FROM view1" )
```

SQL Queries on DataFrames

Part of Spark SQL library

Create SQL View on top of DataFrame

- *Think of this like pointer to DataFrame*

Run SQL queries on the View

Same performance in SQL as using DataFrames in other languages

Use spark.sql in PySpark/Scala to run SQL queries

spark.sql returns output of query as a DataFrame

In [1]:
```
import findspark
findspark.init()
findspark.find()
```

Out[1]:
```
'E:\\Local d\\Zartab\\Apache Spark Installation\\spark-3.3.2-bin-hadoop3'
```

In [2]:
```
from pyspark.sql import SparkSession
from pyspark.sql.types import *
from pyspark.sql.functions import *

spark = (
            SparkSession
                .builder
                .appName("SparkSQLApp")
                .master("local[4]")

                .config("spark.dynamicAllocation.enabled", "false")
                .config("spark.sql.adaptive.enabled", "false")

                .getOrCreate()
        )

sc = spark.sparkContext

spark
```

Read Yellow Taxis data

```
# Create schema for Yellow Taxi Data

taxiSchema = (
                    StructType
                    ([
                        StructField("VendorId"                , IntegerType()   , True),
                        StructField("lpep_pickup_datetime"    , TimestampType() , True),
                        StructField("lpep_dropoff_datetime"   , TimestampType() , True),
                        StructField("passenger_count"         , DoubleType()    , True),
                        StructField("trip_distance"           , DoubleType()    , True),
                        StructField("RatecodeID"              , DoubleType()    , True),
                        StructField("store_and_fwd_flag"      , StringType()    , True),
                        StructField("PULocationID"            , IntegerType()   , True),
                        StructField("DOLocationID"            , IntegerType()   , True),
                        StructField("payment_type"            , IntegerType()   , True),
                        StructField("fare_amount"             , DoubleType()    , True),
                        StructField("extra"                   , DoubleType()    , True),
                        StructField("mta_tax"                 , DoubleType()    , True),
                        StructField("tip_amount"              , DoubleType()    , True),
                        StructField("tolls_amount"            , DoubleType()    , True),
                        StructField("improvement_surcharge"   , DoubleType()    , True),
                        StructField("total_amount"            , DoubleType()    , True),
                        StructField("congestion_surcharge"    , DoubleType()    , True),
                        StructField("airport_fee"             , DoubleType()    , True)
                    ])
                )
```

```python
# Read YellowTaxis CSV file

yellowTaxiDF = (
                  spark
                    .read
                    .option("header", "true")

                    .schema(taxiSchema)

                    .csv("E:\SparkData\Raw\YellowTaxis_202210.csv")
                )

yellowTaxiDF.printSchema()
```

```
root
 |-- VendorId: integer (nullable = true)
 |-- lpep_pickup_datetime: timestamp (nullable = true)
 |-- lpep_dropoff_datetime: timestamp (nullable = true)
 |-- passenger_count: double (nullable = true)
 |-- trip_distance: double (nullable = true)
 |-- RatecodeID: double (nullable = true)
 |-- store_and_fwd_flag: string (nullable = true)
 |-- PULocationID: integer (nullable = true)
 |-- DOLocationID: integer (nullable = true)
```

Create a SQL View based on Python DataFrame

```
yellowTaxiDF.createOrReplaceTempView("YellowTaxis")
```

Run SQL query on View

```
outputDF = spark.sql(
                        "SELECT * FROM YellowTaxis WHERE PULocationID = 171"
                    )

outputDF.show()
```

```
+--------+--------------------+---------------------+---------------+-------------+---
----------+-----------+----------+-----+-------+----------+-----------+----------
---+-----------+
|VendorId|lpep_pickup_datetime|lpep_dropoff_datetime|passenger_count|trip_distance|Rat
OLocationID|payment_type|fare_amount|extra|mta_tax|tip_amount|tolls_amount|improvement
rge|airport_fee|
+--------+--------------------+---------------------+---------------+-------------+---
```

Read Green Taxis data, and create Temp View

```
# Read GreenTaxis TSV file

greenTaxiDF = (
                    spark
                      .read
                      .option("header", "true")
                      .option("delimiter", "\t")
                      .csv("E:\SparkData\Raw\GreenTaxis_202210.csv")
                )

greenTaxiDF.createOrReplaceTempView("GreenTaxis")
```

Write SQL query to merge Yellow and Green Taxis data

```
spark.sql("""

SELECT 'Yellow'                    AS TaxiType

      , lpep_pickup_datetime       AS PickupTime
      , lpep_dropoff_datetime      AS DropTime
      , PULocationID               AS PickupLocationId
      , DOLocationID               AS DropLocationId
FROM YellowTaxis

UNION ALL

SELECT 'Green'                     AS TaxiType

      , lpep_pickup_datetime       AS PickupTime
      , lpep_dropoff_datetime      AS DropTime
      , PULocationID               AS PickupLocationId
      , DOLocationID               AS DropLocationId
FROM GreenTaxis

""").show()
```

Read Taxi Zones data, and create Global Temp View

```
taxiZonesSchema = "LocationID INT, Borough STRING, Zone STRING, ServiceZone STRING"

taxiZonesDF = (
                  spark
                    .read
                    .schema(taxiZonesSchema)
                    .csv("E:\SparkData\Raw\TaxiZones.csv")
               )


# Create global temp view
taxiZonesDF.createOrReplaceGlobalTempView("TaxiZones")


taxiZonesDF.show()
```

```
+----------+------------+--------------------+-----------+
|LocationID|     Borough|                Zone|ServiceZone|
+----------+------------+--------------------+-----------+
|         1|         EWR|      Newark Airport|        EWR|
|         2|      Queens|         Jamaica Bay|  Boro Zone|
|         3|       Bronx|Allerton/Pelham G...|  Boro Zone|
```

What's the difference between a Global view and a Temp view?

The difference between Global and Temp is how the lifetime of the view is tied to the application:

TempView

Creates or replaces a local temporary view with this **DataFrame**.
The lifetime of this temporary table is tied to the **SparkSession** that was used to create this DataFrame.

GlobalView

Creates or replaces a global temporary view using the given name.
The lifetime of this temporary view is tied to this **Spark application**.

Exercise

Figure out the number of rides grouped by Borough and type of Taxi

Create a report

Figure out number of rides, grouped by Borough and type of Taxi

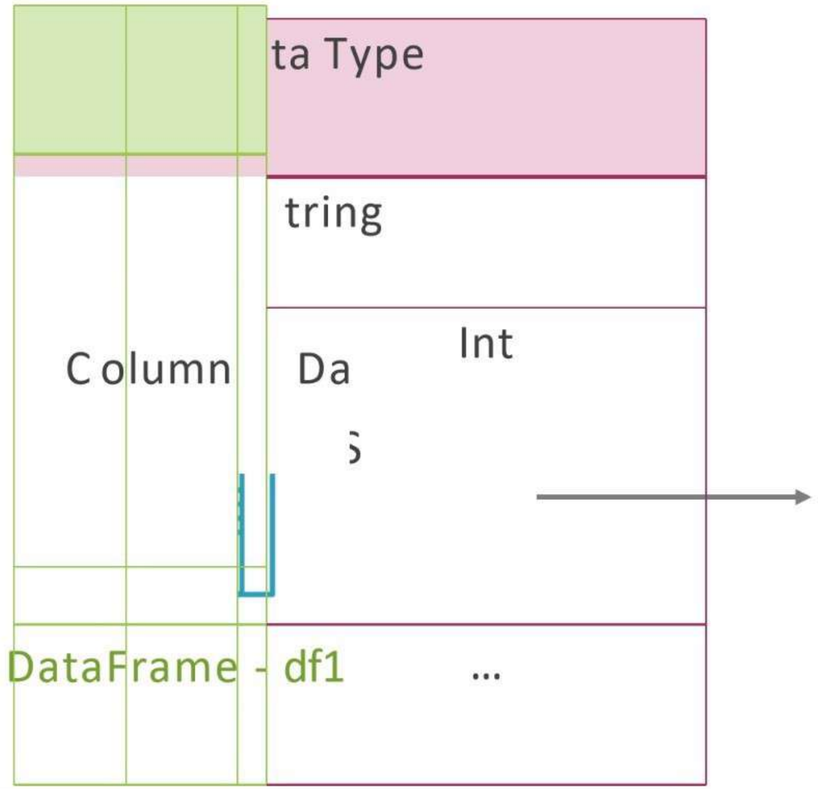
```
spark.sql("""

SELECT Borough, TaxiType, COUNT(*) AS TotalTrips

FROM global_temp.TaxiZones

LEFT JOIN
(

    SELECT 'Yellow' AS TaxiType, PULocationID FROM YellowTaxis

    UNION ALL

    SELECT 'Green' AS TaxiType, PULocationID FROM GreenTaxis

) AllTaxis

ON AllTaxis.PULocationID = TaxiZones.LocationID

GROUP BY Borough, TaxiType

ORDER BY Borough, TaxiType

""").show()
```

Borough	TaxiType	TotalTrips
Bronx	Green	1852
Bronx	Yellow	1395
Brooklyn	Green	11113
Brooklyn	Yellow	6475
EWR	Green	15
EWR	Yellow	362
Manhattan	null	2
Manhattan	Green	40545
Manhattan	Yellow	924413
Queens	null	1
Queens	Green	15377
Queens	Yellow	100804
Staten Island	null	6
Staten Island	Green	8
Staten Island	Yellow	97
Unknown	Green	412
Unknown	Yellow	15029

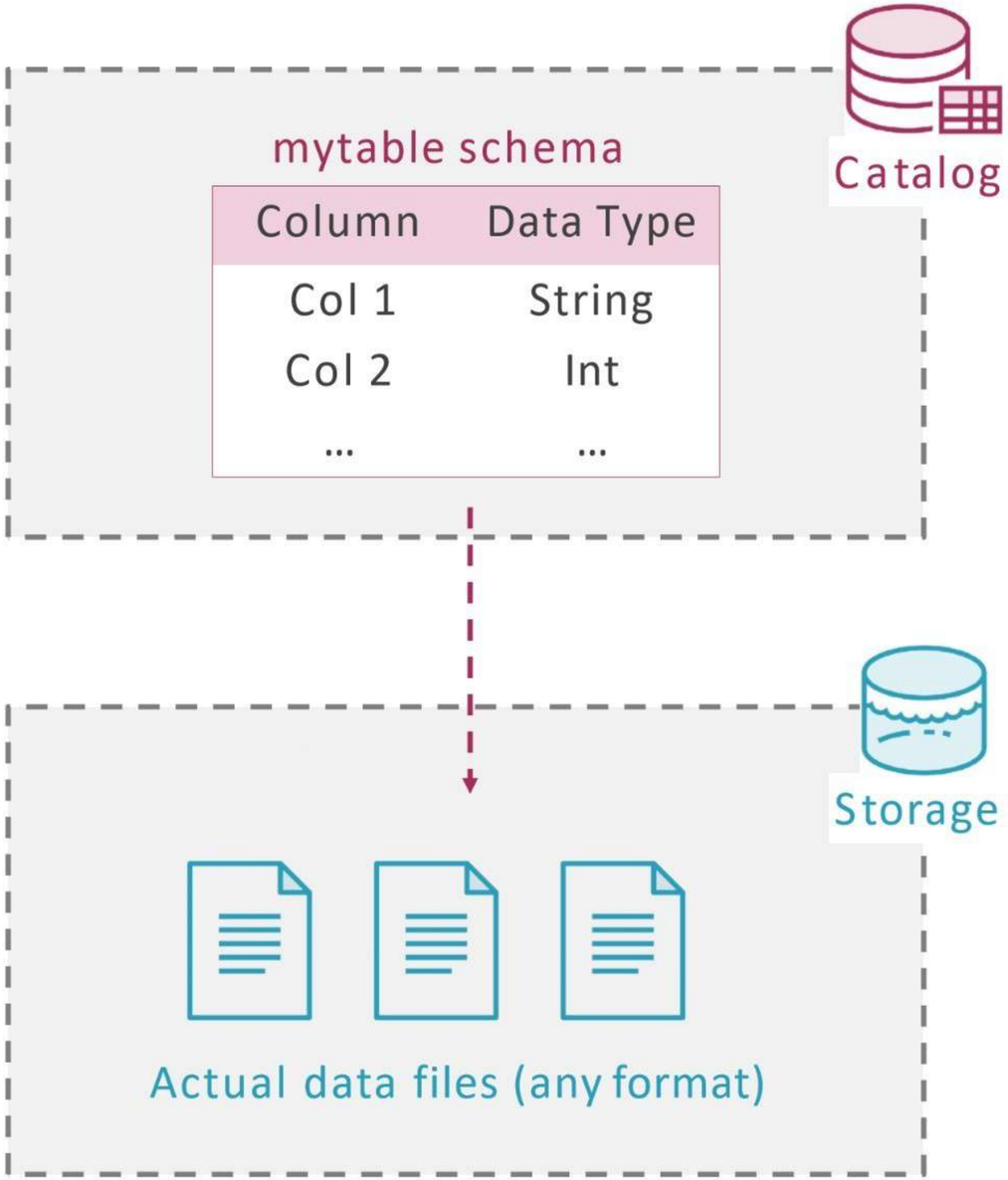
Working with Spark Tables

Writing Data from DataFrame

Directly to Storage	As Spark Table
Files stored in defined format in Storage	Files stored in defined format in Storage
No metadata registered	Metadata registered with a metastore
To query, read files from Storage	To query, either read files from Storage or directly reference the table




```
df1
    .write
    .saveAsTable ("mytable")
```

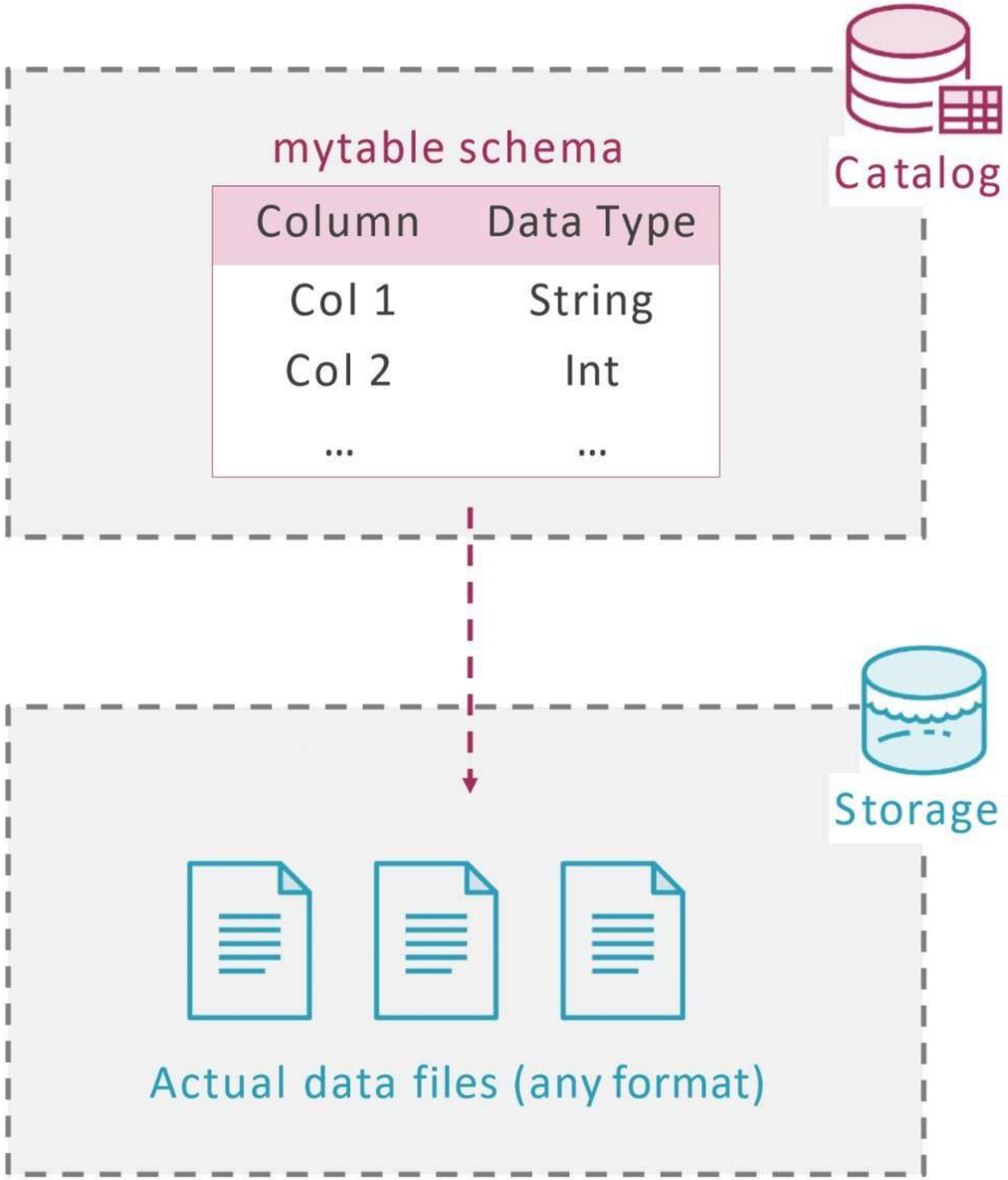


SQL

```
SELECT * FROM mytable
```

PySpark

```
df1 = spark
        .read
        .table("mytable")
```



Helps to manage datasets

No need to define/infer schema while reading

Catalog Types

In-memory Catalog

Only works in a session
Cleaned up when session ends

Persistent Catalog

Metadata is permanently stored
Can be used in any session
Spark has built-in **Hive** catalog

Types of Tables

Managed Table	Unmanaged / External Table
Schema and data is managed by Spark	Only schema is managed by Spark
Data is stored in default location	Data is stored in an external location
Dropping table deletes both schema &data	Dropping table deletes only schema, not data
Useful to persist staging data	Useful to persist processed data

```python
import findspark
findspark.init()
findspark.find()
```

```
'E:\\Local d\\Zartab\\Apache Spark Installation\\spark-3.3.2-bin-hadoop3'
```

```python
from pyspark.sql import SparkSession
from pyspark.sql.types import *
from pyspark.sql.functions import *

spark = (
            SparkSession
                .builder
                .appName("SparkTablesApp")
                .master("local[4]")

                .config("spark.dynamicAllocation.enabled", "false")
                .config("spark.sql.adaptive.enabled", "false")

                # Enable Hive support
                .enableHiveSupport()

                .getOrCreate()
        )

sc = spark.sparkContext

spark
```

```python
# Create schema for Yellow Taxi data
yellowTaxiSchema = (
                        StructType
                        ([
                            StructField("VendorId"                , IntegerType()   , True),
                            StructField("lpep_pickup_datetime"    , TimestampType() , True),
                            StructField("lpep_dropoff_datetime"   , TimestampType() , True),
                            StructField("passenger_count"         , DoubleType()    , True),
                            StructField("trip_distance"           , DoubleType()    , True),
                            StructField("RatecodeID"              , DoubleType()    , True),
                            StructField("store_and_fwd_flag"      , StringType()    , True),
                            StructField("PULocationID"            , IntegerType()   , True),
                            StructField("DOLocationID"            , IntegerType()   , True),
                            StructField("payment_type"            , IntegerType()   , True),
                            StructField("fare_amount"             , DoubleType()    , True),
                            StructField("extra"                   , DoubleType()    , True),
                            StructField("mta_tax"                 , DoubleType()    , True),
                            StructField("tip_amount"              , DoubleType()    , True),
                            StructField("tolls_amount"            , DoubleType()    , True),
                            StructField("improvement_surcharge"   , DoubleType()    , True),
                            StructField("total_amount"            , DoubleType()    , True),
                            StructField("congestion_surcharge"    , DoubleType()    , True),
                            StructField("airport_fee"             , DoubleType()    , True)
                        ])
                    )
```

```python
# Read YellowTaxis CSV file
yellowTaxiDF = (
                  spark
                    .read
                    .option("header", "true")
                    .schema(yellowTaxiSchema)
                    .csv("E:\SparkData\Raw\YellowTaxis_202210.csv")
                )

yellowTaxiDF.printSchema()
```

```
root
 |-- VendorId: integer (nullable = true)
 |-- lpep_pickup_datetime: timestamp (nullable = true)
 |-- lpep_dropoff_datetime: timestamp (nullable = true)
 |-- passenger_count: double (nullable = true)
 |-- trip_distance: double (nullable = true)
 |-- RatecodeID: double (nullable = true)
 |-- store_and_fwd_flag: string (nullable = true)
```

Create Database in Hive metastore

```
spark.sql("""

SHOW DATABASES

""").show()
```

```
+---------+
|namespace|
+---------+
|  default|
+---------+
```

```
spark.sql("""

CREATE DATABASE IF NOT EXISTS TaxisDB

""")
```

```
DataFrame[]
```

```
spark.sql("""

SHOW DATABASES

""").show()
```

```
+---------+
|namespace|
+---------+
|  default|
|  taxisdb|
+---------+
```

Save DataFrame as a Managed Spark Table in Hive

```
(
    yellowTaxiDF
            .write

            .mode("overwrite")

            .saveAsTable("TaxisDB.YellowTaxisManaged")
)
```

```
spark.sql("""

SHOW TABLES IN TaxisDB

""").show(50, truncate=False)
```

```
+---------+------------------+-----------+
|namespace|tableName         |isTemporary|
+---------+------------------+-----------+
|taxisdb  |yellowtaxismanaged|false      |
+---------+------------------+-----------+
```

Run queries on Managed Spark Table

```
spark.sql("""

SELECT *

FROM TaxisDB.YellowTaxisManaged

LIMIT 10

""").show()
```

```
+--------+--------------------+---------------------+---------------+-------------+------
-----------+------------+-----------+-----+-------+----------+------------+-------------
---+-----------+
|VendorId|lpep_pickup_datetime|lpep_dropoff_datetime|passenger_count|trip_distance|Rateco
OLocationID|payment_type|fare_amount|extra|mta_tax|tip_amount|tolls_amount|improvement_su
rge|airport_fee|
+--------+--------------------+---------------------+---------------+-------------+------
-----------+------------+-----------+-----+-------+----------+------------+-------------
---+-----------+
|       1| 2022-10-01 05:33:41|  2022-10-01 05:48:39|            1.0|          1.7|
107|           1|        9.5|  3.0|    0.5|      2.65|         0.0|                 0.3|
0.0|
```

```
outputDF = (
                spark
                    .read
                    .table("TaxisDB.YellowTaxisManaged")
            )

outputDF.limit(10).show()
```

```
+--------+--------------------+---------------------+---------------+---------
-----------+------------+-----------+-----+-------+----------+------------+---
---+-----------+
|VendorId|lpep_pickup_datetime|lpep_dropoff_datetime|passenger_count|trip_dist
OLocationID|payment_type|fare_amount|extra|mta_tax|tip_amount|tolls_amount|imp
rge|airport_fee|
+--------+--------------------+---------------------+---------------+---------
-----------+------------+-----------+-----+-------+----------+------------+---
---+-----------+
|       1| 2022-10-01 05:33:41|  2022-10-01 05:48:39|            1.0|
107|           1|        9.5|  3.0|    0.5|      2.65|         0.0|
0.0|
|       2| 2022-10-01 05:44:30|  2022-10-01 05:49:48|            2.0|
238|           2|        5.5|  0.5|    0.5|       0.0|         0.0|
0.0|
|       2| 2022-10-01 05:57:13|  2022-10-01 06:07:41|            1.0|
166|           1|        9.0|  0.5|    0.5|      2.06|         0.0|
```

```
spark.sql("""

DESCRIBE TABLE EXTENDED TaxisDB.YellowTaxisManaged

""").show(50, truncate=False)
```

```
+-----------------------------+-----------------------------------------------
-----------------------------------------------------------------------------
-----------------------+-------+
|col_name                     |data_type
|comment|
+-----------------------------+-----------------------------------------------
-----------------------------------------------------------------------------
-----------------------+-------+
|VendorId                     |int
|null   |
|lpep_pickup_datetime         |timestamp
|null   |
|lpep_dropoff_datetime        |timestamp
|null   |
|passenger_count              |double
|null   |
```

Save DataFrame as an Unmanaged / External Spark Table in Hive

Only change is to define the path

```
(
    yellowTaxiDF
            .write

            .mode("overwrite")

            .option("path", "E:\SparkData\Raw\Output\YellowTaxisOutput.parquet")

            #.option("format", "csv")             # Default is 'parquet'

            .saveAsTable("TaxisDB.YellowTaxis")
)
```

```
spark.sql("""

DESCRIBE TABLE EXTENDED TaxisDB.YellowTaxis

""").show(50, truncate=False)
```

```
+---------------------------+---------------------------------------------------------------+-------+
|col_name                   |data_type                                                      |comment|
+---------------------------+---------------------------------------------------------------+-------+
```

Drop External table and recreate using stored files

```
spark.sql("""
DROP TABLE TaxisDB.YellowTaxis
""")
```

```
DataFrame[]
```

```
spark.sql("""
CREATE TABLE TaxisDB.YellowTaxis
USING PARQUET
LOCATION "E://SparkData//Raw//Output//YellowTaxisOutput.parquet//"
""")
```

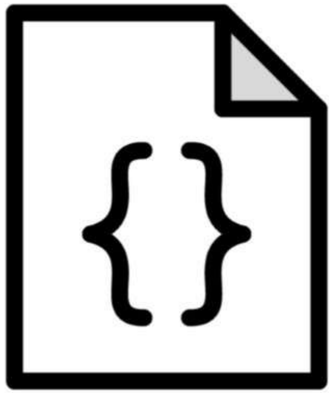
```
DataFrame[]
```

```
yellowTaxiDF = (
                  spark
                    .read
                    .table("TaxisDB.YellowTaxis")

                )

yellowTaxiDF.show()
```

```
+--------+--------------------+--------------------+---------------+-------------+----------+-----------------
-----------+------------+-----------+-----+-------+----------+------------+---------------------+------------+-
---+-----------+
|VendorId|lpep_pickup_datetime|lpep_dropoff_datetime|passenger_count|trip_distance|RatecodeID|store_and_fwd_fla
OLocationID|payment_type|fare_amount|extra|mta_tax|tip_amount|tolls_amount|improvement_surcharge|total_amount|c
rge|airport_fee|
+--------+--------------------+--------------------+---------------+-------------+----------+-----------------
-----------+------------+-----------+-----+-------+----------+------------+---------------------+------------+-
---+-----------+
|       1| 2022-10-01 05:33:41|  2022-10-01 05:48:39|            1.0|          1.7|       1.0|
107|           1|        9.5|  3.0|    0.5|      2.65|         0.0|                  0.3|       15.95|
0.0|
|       2| 2022-10-01 05:44:30|  2022-10-01 05:49:48|            2.0|         0.72|       1.0|
238|           2|        5.5|  0.5|    0.5|       0.0|         0.0|                  0.3|         9.3|
0.0|
```

Working with User Defined Functions (UDFs)

User Defined Functions



Extends functionality with custom Scala/Python methods

Invoked for each row in DataFrame

To register a function as Spark UDF:

- Register using udf() to use with DataFrames
- Register using spark.udf.register() to use in SQL

Challenges

- Optimizations can't be applied to UDFs
- Nulls can cause issues if not handled well

In [1]:
```python
import findspark
findspark.init()
findspark.find()
```

Out[1]:
```
'E:\\Local d\\Zartab\\Apache Spark Installation\\spark-3.3.2-bin-hadoop3'
```

In [2]:
```python
from pyspark.sql import SparkSession
from pyspark.sql.types import *
from pyspark.sql.functions import *

spark = (
            SparkSession
                .builder
                .appName("UdfApp")
                .master("local[4]")

                .config("spark.dynamicAllocation.enabled", "false")
                .config("spark.sql.adaptive.enabled", "false")

                .getOrCreate()
        )

sc = spark.sparkContext

spark
```

```python
# Read Cabs file
cabsDF = (
              spark
                .read
                .option("header", "true")
                .option("inferSchema", "true")

                .csv("E:\SparkData\Raw\Cabs.csv")
         )


# Create temp view
cabsDF.createOrReplaceTempView("Cabs")


cabsDF.show(truncate=False)
```

```
+---------+--------------------+------------------------+----------------+------+--------------------+-----------------+---------
|CabNumber|VehicleLicenseNumber|Name                    |LicenseType     |Active|PermitLicenseNumber|VehicleVinNumber |Wheelchai
+---------+--------------------+------------------------+----------------+------+--------------------+-----------------+---------
|T802127C |C19641              |ABCON INC.              |OWNER MUST DRIVE|YES   |null                |5TDBK3EH0DS268018|null
|T525963C |5362996             |ACCEPTABLE TAXI LLC     |NAMED DRIVER    |YES   |null                |JTDZN3EU1GJ044907|null
|T457353C |5122403             |ALLIS CAB CORP          |NAMED DRIVER    |YES   |null                |4T1BD1FK9FU150746|null
|T606373C |5386507             |BENE CAB CORP           |NAMED DRIVER    |YES   |AC992               |4T1BD1FK1EU137360|null
```

Create a function to convert case

```
def convertCase(str):

    result = ""

    nameWordsArray = str.split(",")

    for nameWord in nameWordsArray:
       result = (result
                    + nameWord[0:1].upper()              # Ex- for word 'MOHIT', returns=> 'M'
                    + nameWord[1:len(nameWord)].lower() # Ex- for word 'MOHIT', returns=> 'ohit, '
                    + ", "
                 )

    result = result[0:len(result) - 2]
                                        # Ex- for name 'Batra, Mohit, ' returns => 'Batra, Mohit'

    return result
```

Option 1: Register function as a User Defined Function (UDF)

This registration option is for using UDF in Python/Scala

```
convertCaseUdf = udf( lambda str: convertCase(str), StringType() )
```

Use UDF in DataFrame code

```
(
    cabsDF
        .select(
                    "Name",

                    convertCaseUdf(col("Name")).alias("Name_ConvertedCase")
                )

        # .withColumn ("Name_ConvertedCase", convertCaseUdf(col("Name")))

).show(truncate=False)
```

```
+-----------------------+-----------------------+
|Name                   |Name_ConvertedCase     |
+-----------------------+-----------------------+
|ABCON INC.             |Abcon inc.             |
|ACCEPTABLE TAXI LLC    |Acceptable taxi llc    |
```

Option 2: Register function as a User Defined Function (UDF)

This registration option is for using UDF in SQL

```
spark.udf.register("convertCaseSqlUdf", convertCase, StringType())
```

```
<function __main__.convertCase(str)>
```

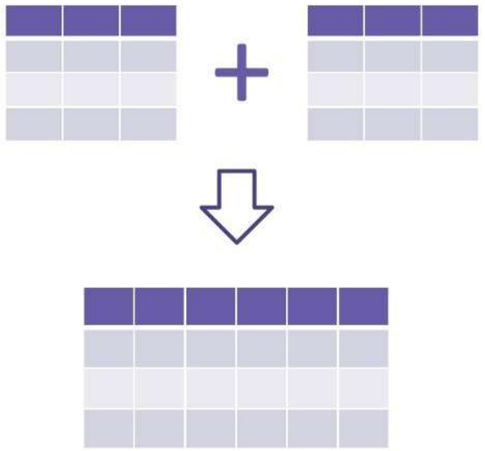
Use UDF in SQL query

```
spark.sql("""

    SELECT Name

         , convertCaseSqlUdf(Name) AS Name_ConvertedCase

    FROM Cabs

""").show(truncate=False)
```

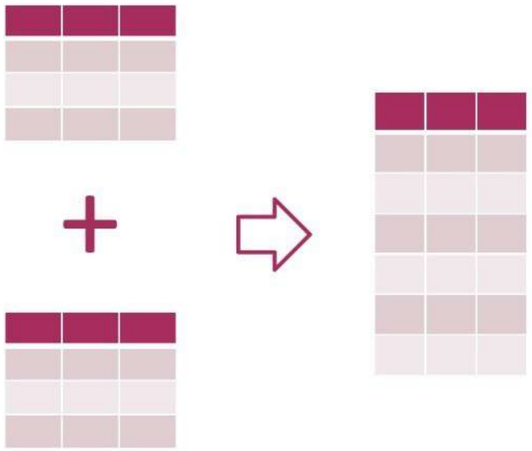
```
+-----------------------+-----------------------+
|Name                   |Name_ConvertedCase     |
+-----------------------+-----------------------+
|ABCON INC.             |Abcon inc.             |
```

Performing Operations on Multiple Datasets

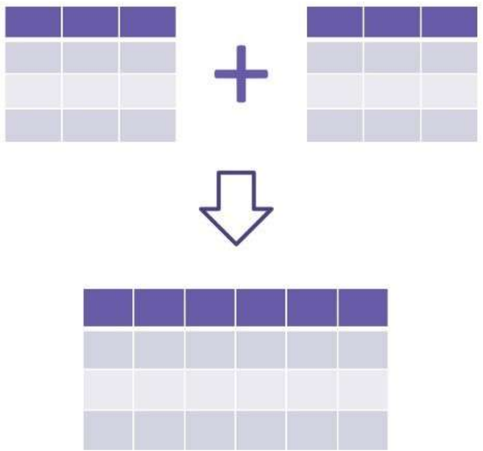
Operations on Multiple Datasets



Join Operations



Set Operations



Join Operations

Combines two datasets, based on conditions

Produces result with new set of columns

Write code in Python/Scala or use SQL queries

Spark supports

- Inner Join
- Left Outer Join
- Right Outer Join
- Full Outer Join
- Cross Join *(Python/Scala only)*
- Semi Join *(Python/Scala only)*
- Anti Join *(Python/Scala only)*

Spark 3 has several optimizations for Join operations

Inner Join

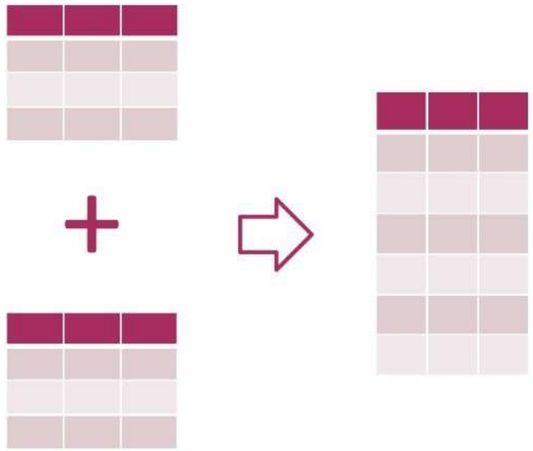
Dept Id	Dept Name
D1	Sales
D2	Marketing
D3	Tech

On

Dept Id

Dept Id	Emp Id	Emp Name
D1	E1	Anna
D1	E2	Steve
D3	E3	Neha
D5	E4	Ivan

Dept Id	Dept Name	Dept Id	Emp Id	Emp Name
D1	Sales	D1	E1	Anna
D1	Sales	D1	E2	Steve
D3	Tech	D3	E3	Neha



Set Operations

Applies to two datasets having same schema, without conditions

Output dataset has same schema as input datasets

Write code in Python/Scala or use SQL queries

Spark supports

- Union
- Union All
- Intersect
- Except / Minus

Union All

User Id	Name
U1	Anna
U2	Cristina
U3	Neha
U4	Steve

User Id	Name
U4	Steve
U5	Ivan
U6	Mohit

User Id	Name
U1	Anna
U2	Cristina
U3	Neha
U4	Steve
U4	Steve
U5	Ivan
U6	Mohit

Union

User Id	Name
U1	Anna
U2	Cristina
U3	Neha
U4	Steve

User Id	Name
U4	Steve
U5	Ivan
U6	Mohit

User Id	Name
U1	Anna
U2	Cristina
U3	Neha
U4	Steve
U5	Ivan
U6	Mohit

Intersect

User Id	Name
U1	Anna
U2	Cristina
U3	Neha
U4	Steve

User Id	Name
U4	Steve
U5	Ivan
U6	Mohit

User Id	Name
U4	Steve

Except / Minus

User Id	Name
U1	Anna
U2	Cristina
U3	Neha
U4	Steve

User Id	Name
U4	Steve
U5	Ivan
U6	Mohit

User Id	Name
U1	Anna
U2	Cristina
U3	Neha

```python
import findspark
findspark.init()
findspark.find()
```

```
'E:\\Local d\\Zartab\\Apache Spark Installation\\spark-3.3.2-bin-hadoop3'
```

```python
from pyspark.sql import SparkSession
from pyspark.sql.types import *
from pyspark.sql.functions import *

spark = (
            SparkSession
                .builder
                .appName("MultipleDatasetsApp")
                .master("local[4]")

                .config("spark.dynamicAllocation.enabled", "false")
                .config("spark.sql.adaptive.enabled", "false")

                .getOrCreate()
        )

sc = spark.sparkContext

spark
```

JOIN Operations

```python
# Create schema for Yellow Taxi data
yellowTaxiSchema = (
                        StructType
                        ([
                            StructField("VendorId"              , IntegerType()   , True),
                            StructField("PickupTime"            , TimestampType() , True),
                            StructField("DropTime"              , TimestampType() , True),
                            StructField("PassengerCount"        , DoubleType()    , True),
                            StructField("TripDistance"          , DoubleType()    , True),
                            StructField("RateCodeId"            , DoubleType()    , True),
                            StructField("StoreAndFwdFlag"       , StringType()    , True),
                            StructField("PickupLocationId"      , IntegerType()   , True),
                            StructField("DropLocationId"        , IntegerType()   , True),
                            StructField("PaymentType"           , IntegerType()   , True),
                            StructField("FareAmount"            , DoubleType()    , True),
                            StructField("Extra"                 , DoubleType()    , True),
                            StructField("MtaTax"                , DoubleType()    , True),
                            StructField("TipAmount"             , DoubleType()    , True),
                            StructField("TollsAmount"           , DoubleType()    , True),
                            StructField("ImprovementSurcharge"  , DoubleType()    , True),
                            StructField("TotalAmount"           , DoubleType()    , True),
                            StructField("CongestionSurcharge"   , DoubleType()    , True),
                            StructField("AirportFee"            , DoubleType()    , True)
                        ])
                    )
```

```python
# Read Yellow Taxis file
yellowTaxiDF = (
                  spark
                    .read
                    .option("header", "true")
                    .schema(yellowTaxiSchema)
                    .csv("E:\SparkData\Raw\YellowTaxis_202210.csv")
                )


# Create temp view
yellowTaxiDF.createOrReplaceTempView("YellowTaxis")


# Print the schema
yellowTaxiDF.printSchema()
```

```
root
 |-- VendorId: integer (nullable = true)
 |-- PickupTime: timestamp (nullable = true)
 |-- DropTime: timestamp (nullable = true)
 |-- PassengerCount: double (nullable = true)
 |-- TripDistance: double (nullable = true)
 |-- RateCodeId: double (nullable = true)
 |-- StoreAndFwdFlag: string (nullable = true)
```

```python
# Create schema for Taxi Zones data
taxiZonesSchema = "PickupLocationId INT, Borough STRING, Zone STRING, ServiceZone STRING"


# Read Taxi Zones file
taxiZonesDF = (
                  spark
                    .read
                    .schema(taxiZonesSchema)
                    .csv("E:\SparkData\Raw\TaxiZones.csv")
               )


# Create temp view
taxiZonesDF.createOrReplaceTempView("TaxiZones")


# Print the schema
taxiZonesDF.printSchema()
```

```
root
 |-- PickupLocationId: integer (nullable = true)
 |-- Borough: string (nullable = true)
 |-- Zone: string (nullable = true)
 |-- ServiceZone: string (nullable = true)
```

Display location information for each ride

Join DataFrames of Yellow Taxis and Taxi Zones

```
joinedDF = (
                yellowTaxiDF

                    .join
                    (
                        taxiZonesDF,

                        yellowTaxiDF.PickupLocationId == taxiZonesDF.PickupLocationId,

                                  # [condition1, condition2...]

                        "inner"   # left, leftouter, right, rightouter, full etc.
                    )
            )

joinedDF.printSchema()
```

```
root
 |-- VendorId: integer (nullable = true)
 |-- PickupTime: timestamp (nullable = true)
 |-- DropTime: timestamp (nullable = true)
```

```python
# Remove duplicate 'PickupLocationId' column

joinedDF = (
                yellowTaxiDF

                    .join
                    (
                        taxiZonesDF,

                        yellowTaxiDF.PickupLocationId == taxiZonesDF.PickupLocationId,

                                  # [condition1, condition2...]

                        "inner"   # left, leftouter, right, rightouter, full etc.
                    )

                    .drop(col("tz.PickupLocationId"))
            )

joinedDF.printSchema()
```

```
root
 |-- VendorId: integer (nullable = true)
 |-- PickupTime: timestamp (nullable = true)
 |-- DropTime: timestamp (nullable = true)
 |-- PassengerCount: double (nullable = true)
```

```python
# Join on same column name
# Only one column shows up in output

joinedDF = (
                yellowTaxiDF.alias("yt")

                    .join
                    (
                        taxiZonesDF.alias("tz"),

                        #col("yt.PickupLocationId") == col("tz.PickupLocationId")

                        ['PickupLocationId'],   # only one PickupLocationId column will be kept

                        "inner"
                    )
            )

joinedDF.printSchema()
```

```
root
 |-- PickupLocationId: integer (nullable = true)
 |-- VendorId: integer (nullable = true)
 |-- PickupTime: timestamp (nullable = true)
```

Exercise 1

Find all LocationIds in TaxiZones, from where no pickups have happened

```
# Exercise 1 Answer

(spark.sql("""

    SELECT DISTINCT tz.*

    FROM TaxiZones tz

        LEFT JOIN YellowTaxis yt ON yt.PickupLocationId = tz.PickupLocationId

        WHERE yt.PickupLocationId IS NULL

""")).show()
```

```
+----------------+-------------+--------------------+-----------+
|PickupLocationId|      Borough|                Zone|ServiceZone|
+----------------+-------------+--------------------+-----------+
|             214|Staten Island|South Beach/Donga...|  Boro Zone|
|              84|Staten Island|Eltingville/Annad...|  Boro Zone|
|              27|       Queens|Breezy Point/Fort...|  Boro Zone|
```

SET Operations

```
# Read Drivers file
driversDF = (
                  spark
                    .read
                    .option("header", "true")
                    .option("inferSchema", "true")
                    .csv("E:\SparkData\Raw\Drivers.csv")
            )

# Create temp view
driversDF.createOrReplaceTempView("Drivers")


driversDF.show()
```

```
+------------------+--------------------+--------------------+--------------+---------------+
|DriverLicenseNumber|                Name|                Type|ExpirationDate|LastDateUpdated|
+------------------+--------------------+--------------------+--------------+---------------+
|           5430898|   ABDEL-BAR,ESLAM,M|MEDALLION TAXI DR...|    04/12/2023|     04/22/2020|
|           5363749|ABDOUSAMADOV,ALIC...|MEDALLION TAXI DR...|    06/01/2020|     04/22/2020|
|           5534446|  ABDUHALIKOV,RUSTAM|MEDALLION TAXI DR...|    06/16/2020|     04/22/2020|
|           5935702|   ABDULLAEV,JONIBEK|MEDALLION TAXI DR...|    03/14/2022|     04/22/2020|
|           5255097|ABDULNABI,MASHHOUR,H|MEDALLION TAXI DR...|    03/16/2021|     04/22/2020|
```

```python
# Read Cabs file
cabsDF = (
              spark
                .read
                .option("header", "true")
                .option("inferSchema", "true")
                .csv("E:\SparkData\Raw\Cabs.csv")
         )

# Create temp view
cabsDF.createOrReplaceTempView("Cabs")


cabsDF.show()
```

```
+---------+-------------------+-------------------+----------------+------+-------------------+-----------------+-
|CabNumber|VehicleLicenseNumber|               Name|     LicenseType|Active|PermitLicenseNumber| VehicleVinNumber|W
+---------+-------------------+-------------------+----------------+------+-------------------+-----------------+-
| T802127C|              C19641|          ABCON INC.|OWNER MUST DRIVE|   YES|               null|5TDBK3EH0DS268018|
| T525963C|            5362996| ACCEPTABLE TAXI LLC|    NAMED DRIVER|   YES|               null|JTDZN3EU1GJ044907|
| T457353C|            5122403|      ALLIS CAB CORP|    NAMED DRIVER|   YES|               null|4T1BD1FK9FU150746|
| T606373C|            5386507|       BENE CAB CORP|    NAMED DRIVER|   YES|              AC992|4T1BD1FK1EU137360|
| T451536C|            5143151|   BOULOS TAXI CORP.|    NAMED DRIVER|   YES|               null|4T1BD1FK8FU142430|
| T500313C|            5268004|     CACERES,JAIME,P|OWNER MUST DRIVE|   YES|               null|4T1BF1FK8FU477889|
| T484321C|            5238526|CALCIUM ONE SERVI...|    NAMED DRIVER|   YES|               null|4T1BD1FK4FU176834|
```

```
1. Create list of all drivers
```

```
2. Create list of unique drivers
```

```
3. Create list of all registered drivers who are driving cabs
```

```
4. Create list of drivers driving cabs, but not registered
```

1. Create list of all drivers

```
(spark.sql("""

    (
        SELECT Name
        FROM Cabs
        WHERE LicenseType = 'OWNER MUST DRIVE'
    )

    UNION ALL

    (
        SELECT Name
        FROM Drivers
    )

""")).count()
```

```
157716
```

2. Create list of unique drivers

```
]: (spark.sql("""

    (
        SELECT Name
        FROM Cabs
        WHERE LicenseType = 'OWNER MUST DRIVE'
    )

    UNION

    (
        SELECT Name
        FROM Drivers
    )

""")).count()
```

```
]: 156566
```

3. Create list of all registered drivers who are driving cabs

```
(spark.sql("""

    (
        SELECT Name
        FROM Cabs
        WHERE LicenseType = 'OWNER MUST DRIVE'
    )

    INTERSECT

    (
        SELECT Name
        FROM Drivers
    )

""")).count()
```

```
1150
```

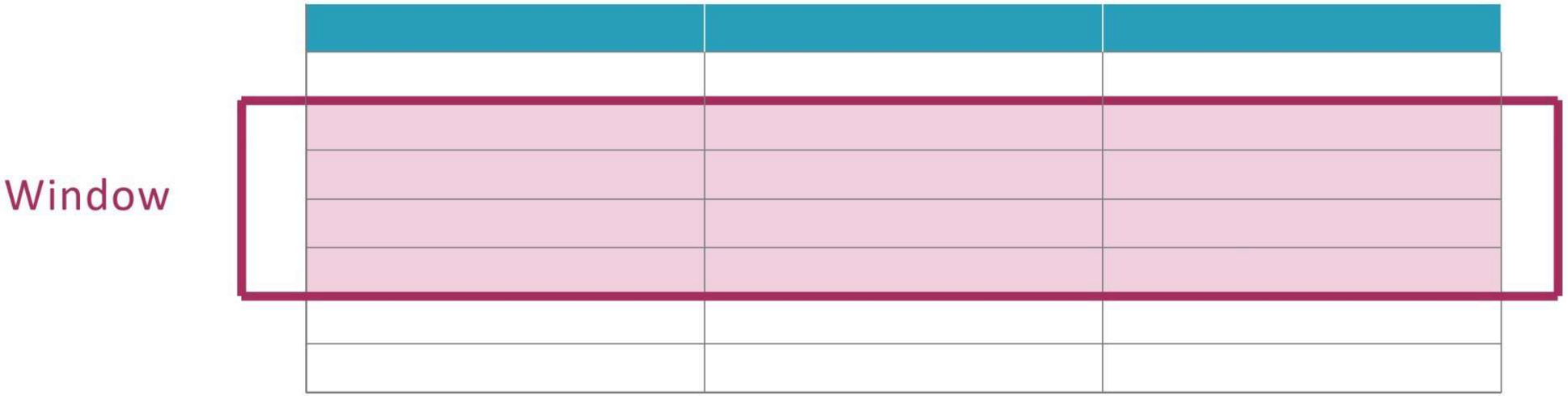
4. Create list of drivers driving cabs, but not registered

```
(spark.sql("""

    (
        SELECT Name
        FROM Cabs
        WHERE LicenseType = 'OWNER MUST DRIVE'
    )

    EXCEPT

    (
        SELECT Name
        FROM Drivers
    )

""")).count()
```

```
1940
```

Performing Window Operations

Window



Window is a subset of related rows in a DataFrame

Define how rows are related, say by time or location

Apply operations

- count, sum, average, min, max etc.

For each row in Window, operation is applied on all the rows of Window

Find share of each department in total expenses

Department	Expense
HR	10,000
Tech	20,000
Sales	30,000
Marketing	40,000

Window is set of rows from Departments table

Find share of each department in total expenses

Department	Expense
HR	10,000
Tech	20,000
Sales	30,000
Marketing	40,000

Expense % = Expense *100 / Total Expenses

Find share of each department in total expenses

Department	Expense	Total Expenses
HR	10,000	100,000
Tech	20,000	100,000
Sales	30,000	100,000
Marketing	40,000	100,000

100,000

Expense % = Expense *100 / Total Expenses

Find share of each department in total expenses

Department	Expense	Total Expenses	Expense %
HR	10,000	100,000	10
Tech	20,000	100,000	20
Sales	30,000	100,000	30
Marketing	40,000	100,000	40

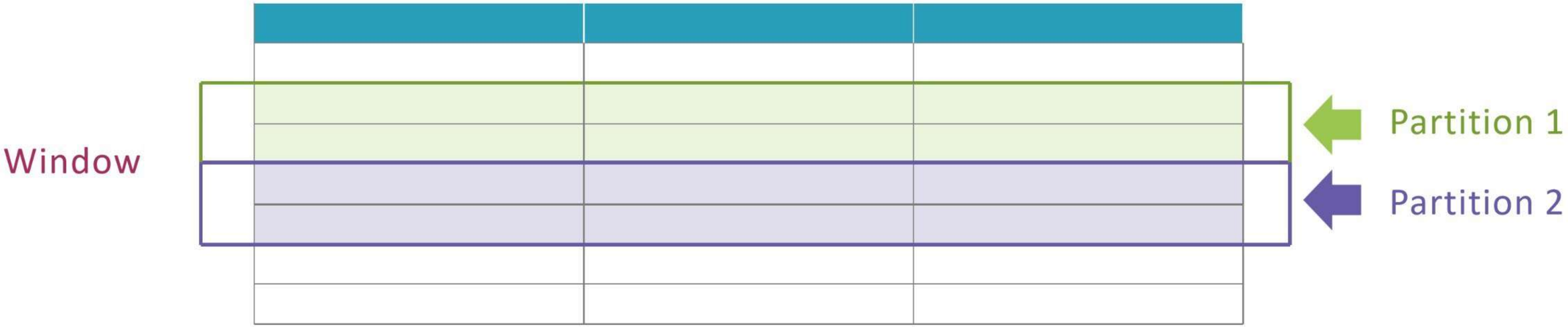
Expense % = Expense *100 / Total Expenses

Find share of each department in total expenses

Department	Expense	Total Expenses	Expense %
HR	10,000	100,000	10
Tech	20,000	100,000	20
Sales	30,000	100,000	30
Marketing	40,000	100,000	40

```
SELECT *
        ,Expense *100 /TotalExpenses          AS ExpensePercent
FROM (
          SELECT  Department, Expense
                  ,SUM (Expense)  OVER ()     AS TotalExpenses
          FROM Departments
        ) subquery
```

Window Partitions



Window can be further divided into Partitions

Operations are then applied on Partitions

- count, sum, average, min, max etc.

Find share of each employee in their department's payout

Department	Employee	Salary
Sales	Anna	8,000
Marketing	Steve	9,500
Tech	Cristina	9,000
Marketing	Neha	10,500
Tech	Kari	10,000
Sales	Ivan	10,000
Tech	Mohit	8,000

Window is set of rows from Employees table

Find share of each employee in their department's payout

Department	Employee	Salary
Sales	Anna	8,000
Marketing	Steve	9,500
Tech	Cristina	9,000
Marketing	Neha	10,500
Tech	Kari	10,000
Sales	Ivan	10,000
Tech	Mohit	8,000

Partition by Department

Find share of each employee in their department's payout

Department	Employee	Salary
Sales	Anna	8,000
Sales	Ivan	10,000
Marketing	Steve	9,500
Marketing	Neha	10,500
Tech	Cristina	9,000
Tech	Kari	10,000
Tech	Mohit	8,000

Find share of each employee in their department's payout

Department	Employee	Salary	Dept Payout
Sales	Anna	8,000	18,000
Sales	Ivan	10,000	18,000
Marketing	Steve	9,500	20,000
Marketing	Neha	10,500	20,000
Tech	Cristina	9,000	27,000
Tech	Kari	10,000	27,000
Tech	Mohit	8,000	27,000

Share % = Salary *100 / DeptPayout

Find share of each employee in their department's payout

Department	Employee	Salary	Dept Payout	Share %
Sales	Anna	8,000	18,000	44.4
Sales	Ivan	10,000	18,000	55.6
Marketing	Steve	9,500	20,000	47.5
Marketing	Neha	10,500	20,000	52.5
Tech	Cristina	9,000	27,000	33.3
Tech	Kari	10,000	27,000	37.0
Tech	Mohit	8,000	27,000	26.7

Share % = Salary *100 / DeptPayout

Find share of each employee in their department's payout

Department	Employee	Salary	Dept Payout	Share %
Sales	Anna	8,000	18,000	44.4
Sales	Ivan	10,000	18,000	55.6
Marketing	Steve	9,500	20,000	47.5
Marketing	Neha	10,500	20,000	52.5

```
SELECT *
        ,Salary *100 / DeptPayout                                   AS SharePercent
FROM (
          SELECT Department, Employee, Salary
                  ,SUM (Salary)      OVER ( PARTITION BY Department )   AS DeptPayout
          FROM Departments
        ) subquery
```

In [1]:
```
import findspark
findspark.init()
findspark.find()
```

In [2]:
```
from pyspark.sql import SparkSession
from pyspark.sql.types import *
from pyspark.sql.functions import *

spark = (
            SparkSession
                .builder
                .appName("WindowOperationsApp")
                .master("local[4]")

                .config("spark.dynamicAllocation.enabled", "false")
                .config("spark.sql.adaptive.enabled", "false")

                .getOrCreate()
        )

sc = spark.sparkContext

spark
```

```python
# Create schema for Yellow Taxi data
yellowTaxiSchema = (
                        StructType
                        ([
                            StructField("VendorId"                , IntegerType()   , True),
                            StructField("PickupTime"              , TimestampType() , True),
                            StructField("DropTime"                , TimestampType() , True),
                            StructField("PassengerCount"          , DoubleType()    , True),
                            StructField("TripDistance"            , DoubleType()    , True),
                            StructField("RateCodeId"              , DoubleType()    , True),
                            StructField("StoreAndFwdFlag"         , StringType()    , True),
                            StructField("PickupLocationId"        , IntegerType()   , True),
                            StructField("DropLocationId"          , IntegerType()   , True),
                            StructField("PaymentType"             , IntegerType()   , True),
                            StructField("FareAmount"              , DoubleType()    , True),
                            StructField("Extra"                   , DoubleType()    , True),
                            StructField("MtaTax"                  , DoubleType()    , True),
                            StructField("TipAmount"               , DoubleType()    , True),
                            StructField("TollsAmount"             , DoubleType()    , True),
                            StructField("ImprovementSurcharge"    , DoubleType()    , True),
                            StructField("TotalAmount"             , DoubleType()    , True),
                            StructField("CongestionSurcharge"     , DoubleType()    , True),
                            StructField("AirportFee"              , DoubleType()    , True)
                        ])
                    )
```

```python
# Read Yellow Taxis file
yellowTaxiDF = (
                    spark
                      .read
                      .option("header", "true")
                      .schema(yellowTaxiSchema)
                      .csv("E:\SparkData\Raw\YellowTaxis_202210.csv")
                )


# Create temp view
yellowTaxiDF.createOrReplaceTempView("YellowTaxis")
```

```python
# Create schema for Taxi Zones data
taxiZonesSchema = "PickupLocationId INT, Borough STRING, Zone STRING, ServiceZone STRING"


# Read Taxi Zones file
taxiZonesDF = (
                  spark
                    .read
                    .schema(taxiZonesSchema)
                    .csv("E:\SparkData\Raw\TaxiZones.csv")
              )


# Create temp view
taxiZonesDF.createOrReplaceTempView("TaxiZones")
```

Windows - Find share of each borough in terms of rides

1. Get rides for each borough

```
taxiRidesDF = (

  spark.sql("""
                SELECT tz.Borough
                     , COUNT(*)     AS RideCount

                FROM TaxiZones tz
                    INNER JOIN YellowTaxis yt ON yt.PickupLocationId = tz.PickupLocationId

                GROUP BY tz.Borough
   """)
)

taxiRidesDF.createOrReplaceTempView("TaxiRides")

taxiRidesDF.orderBy("Borough").show()
```

Borough	RideCount
Bronx	1395
Brooklyn	6475
EWR	362
Manhattan	924413
Queens	100804
Staten Island	97
Unknown	15029

2. Calculate total rides across all boroughs

a) Create Window over entire table

b) Add total rides (across all boroughs) against each row

```
taxiRidesWindowDF = (
    spark.sql("""
                    SELECT *
                         , SUM (RideCount)   OVER ()   AS TotalRideCount

                    FROM TaxiRides
    """)
)

taxiRidesWindowDF.orderBy("Borough").show()

taxiRidesWindowDF.createOrReplaceTempView("TaxiRidesWindow")
```

```
+-------------+---------+--------------+
|      Borough|RideCount|TotalRideCount|
+-------------+---------+--------------+
|        Bronx|     1395|       1048575|
|     Brooklyn|     6475|       1048575|
|          EWR|      362|       1048575|
```

3. Find share of each borough in terms of rides

Divide Borough's ride count with Total Ride Count (across all boroughs)

```
(spark.sql("""

            SELECT *
                  , ROUND( (RideCount * 100) / TotalRideCount, 2)   AS RidesSharePercent

            FROM TaxiRidesWindow
            ORDER BY Borough

""")).show()
```

```
+-------------+---------+---------------+-----------------+
|      Borough|RideCount|TotalRideCount|RidesSharePercent|
+-------------+---------+---------------+-----------------+
|        Bronx|     1395|        1048575|             0.13|
|     Brooklyn|     6475|        1048575|             0.62|
|          EWR|      362|        1048575|             0.03|
|    Manhattan|   924413|        1048575|            88.16|
|       Queens|   100804|        1048575|             9.61|
|Staten Island|       97|        1048575|             0.01|
|      Unknown|    15029|        1048575|             1.43|
```

Window Partitions - Find share of each zone in terms of rides, within their borough

1. Get rides for each zone

Zone is part of Borough

```
taxiRidesDF = (
    spark.sql("""
                    SELECT tz.Borough
                         , tz.Zone
                         , COUNT(*)     AS RideCount

                    FROM TaxiZones tz
                        INNER JOIN YellowTaxis yt ON yt.PickupLocationId = tz.PickupLocationId

                    GROUP BY tz.Borough
                           , tz.Zone
    """)
)

taxiRidesDF.orderBy("Borough", "Zone").show(truncate=False)

taxiRidesDF.createOrReplaceTempView("TaxiRides")
```

Borough	Zone	RideCount
Bronx	Allerton/Pelham Gardens	15
Bronx	Bedford Park	28
Bronx	Belmont	13
Bronx	Bronx Park	4
Bronx	Bronxdale	15
Bronx	City Island	1
Bronx	Claremont/Bathgate	18
Bronx	Co-Op City	63
Bronx	Country Club	2
Bronx	Crotona Park East	17
Bronx	East Concourse/Concourse Village	46
Bronx	East Tremont	37
Bronx	Eastchester	29
Bronx	Fordham South	16
Bronx	Highbridge	47
Bronx	Hunts Point	20
Bronx	Kingsbridge Heights	49
Bronx	Longwood	11

2. Calculate total rides across each borough

a) Create Window over entire table, and partition by Borough

b) Add total rides (across all zones in a borough) against each row

```
taxiRidesWindowDF = (
  spark.sql("""

            SELECT *
                 , SUM (RideCount)     OVER (PARTITION BY Borough)    AS TotalRideCountByBorough

            FROM TaxiRides
  """)
)

taxiRidesWindowDF.orderBy("Borough", "Zone").show(truncate=False)

taxiRidesWindowDF.createOrReplaceTempView("TaxiRidesWindow")
```

```
+-------+-------------------------------+---------+-----------------------+
|Borough|Zone                           |RideCount|TotalRideCountByBorough|
+-------+-------------------------------+---------+-----------------------+
|Bronx  |Allerton/Pelham Gardens        |15       |1395                   |
|Bronx  |Bedford Park                   |28       |1395                   |
```

3. Find share of each zone in terms of rides, within their borough

Divide Zone's ride count with Borough Ride Count

```
(spark.sql("""

    SELECT *
          , ROUND( (RideCount * 100) / TotalRideCountByBorough, 2)   AS RidesSharePercentInBorough

    FROM TaxiRidesWindow
    ORDER BY Borough, Zone

""")).show(truncate=False)
```

```
+-------+-------------------------------+---------+----------------------+-------------------------+
|Borough|Zone                           |RideCount|TotalRideCountByBorough|RidesSharePercentInBorough|
+-------+-------------------------------+---------+----------------------+-------------------------+
|Bronx  |Allerton/Pelham Gardens        |15       |1395                  |1.08                     |
|Bronx  |Bedford Park                   |28       |1395                  |2.01                     |
|Bronx  |Belmont                        |13       |1395                  |0.93                     |
|Bronx  |Bronx Park                     |4        |1395                  |0.29                     |
|Bronx  |Bronxdale                      |15       |1395                  |1.08                     |
|Bronx  |City Island                    |1        |1395                  |0.07                     |
|Bronx  |Claremont/Bathgate             |18       |1395                  |1.29                     |
|Bronx  |Co-Op City                     |63       |1395                  |4.52                     |
```

Summary

Create temp views on DataFrames & run SQL queries

Spark provides built-in catalogs (local & Hive)

- Data for Spark tables is stored in Storage as files
- Only metadata of dataset is stored in catalog

Write complex logic as UDF & invoke with DF/SQL code

Operations on multiple datasets

- Join operations – Inner, Outer & Cross joins
- Set operations – Union (All), Intersect, Except

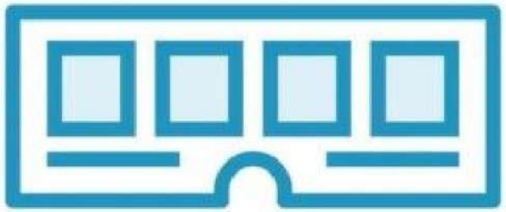
Use Windows to aggregate on subset of related rows

- Window can be divided into partitions
- Use Over & PartitionBy clauses to define window/partitions

Up Next:
Performing Optimizations in Spark

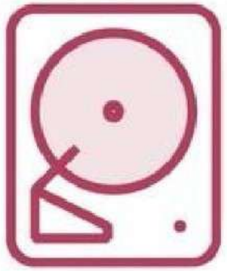
Working with Spark Partitions

Types of Partitioning



In-Memory Partitioning

- Chunks of data read in memory
- All partitions together constitute RDD/DataFrame



Disk Partitioning

- Writing output to disk by physically partitioning data based on columns

In-Memory Partition Settings

Settings for Reading Data

Wide transformations

Settings for Data Shuffling

1. For Reading Data

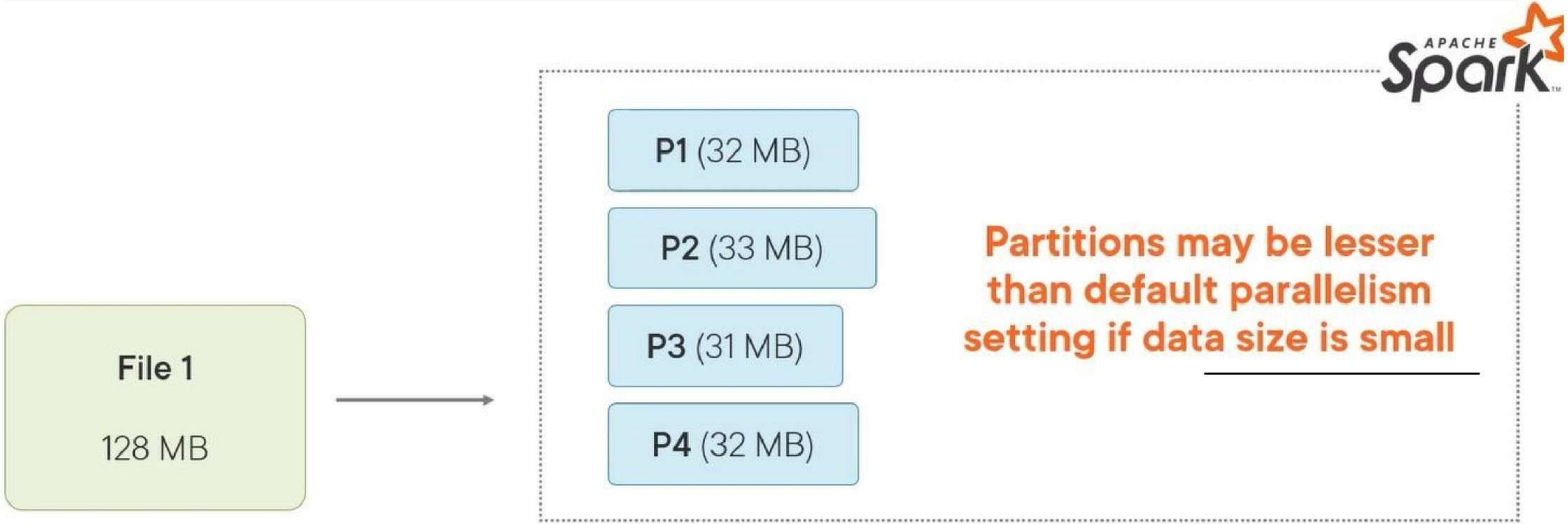
spark.default.parallelism = 4 → [default = no. of cores]

File 1

128 MB

1. For Reading Data

```
spark.default.parallelism = 4                    [default = no. of cores]
```

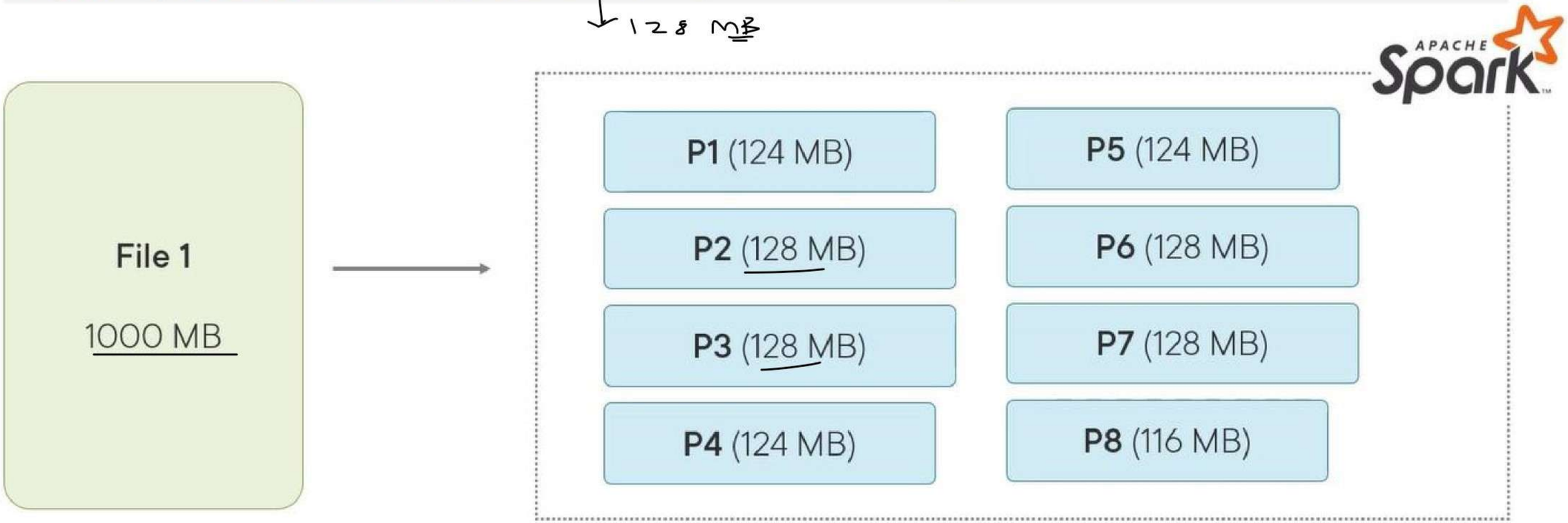


1. For Reading Data

spark.default.parallelism = 4 [default = no. of cores]

spark.sql.files.maxPartitionBytes = 128 MB [default = 128 MB]

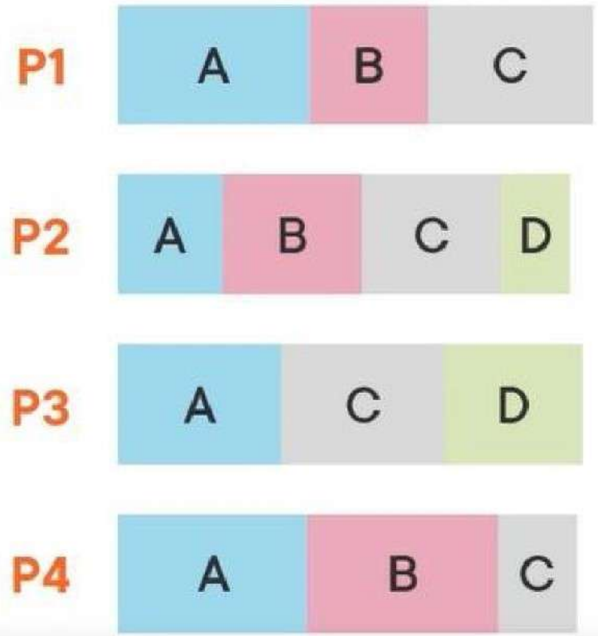
128 MB



2. For Shuffling Data

200

spark.sql.shuffle.partitions = 3 [default = 200]

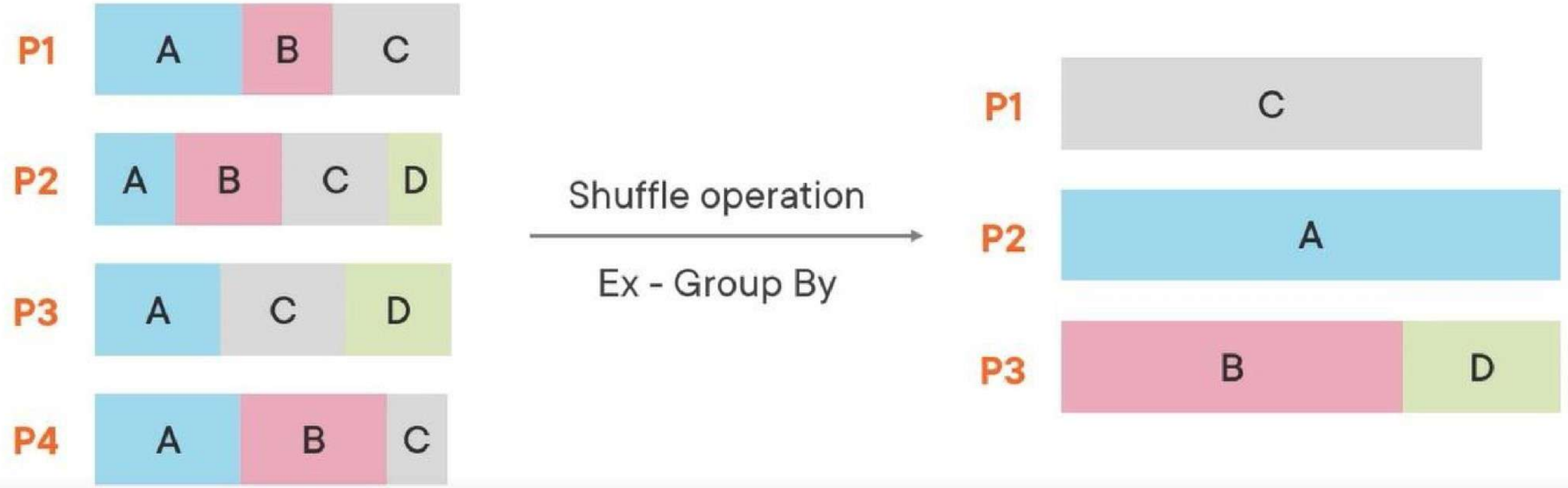


2. For Shuffling Data

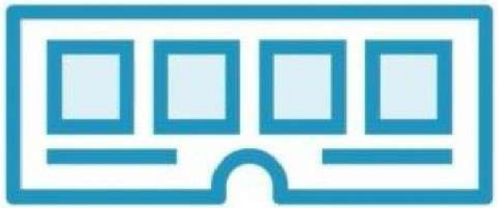
Stage → 4

Stage → 200

spark.sql.shuffle.partitions = 3 [default = 200]



Impact of In-Memory Partitions



Partitions are processed in parallel

Partitions and Cores determine parallelism of a Job

Having very few/big partitions:

- Each task may need to process lot of data
- Cluster resources may be under-utilized

Having lot of/small partitions:

- Too many tasks are created
- Reduces parallelism since tasks go in waiting state

In [1]:
```
import findspark
findspark.init()
findspark.find()
```

Out[1]:
```
'E:\\Local d\\Zartab\\Apache Spark Installation\\spark-3.3.2-bin-hadoop3'
```

```python
from pyspark.sql import SparkSession
from pyspark.sql.types import *
from pyspark.sql.functions import *

spark = (
            SparkSession
                .builder
                .appName("SparkPartitionsApp")

                # Local
                .master("local[4]")

                # Standalone/YARN

                #.config("spark.cores.max",            "6")

                #.config("spark.executor.memory",      "2g")
                #.config("spark.executor.cores",       "2")

                .config("spark.dynamicAllocation.enabled", "false")
                .config("spark.sql.adaptive.enabled", "false")

                .getOrCreate()
        )

sc = spark.sparkContext
```

Check default parallelism

```
sc.defaultParallelism
```

4

1. Partition settings while reading data

Check partitions for a small dataset

Spark finds optimal number of partitions

]:
```
# Read Taxi Zones data
taxiZonesDF = (
                  spark
                    .read
                    .option("inferSchema", "true")
                    .csv("E:\SparkData\Raw\TaxiZones.csv")
               )

# Check number of partitions
print("Partitions = "    + str( taxiZonesDF.rdd.getNumPartitions() ))

# Check number of records
print("Record Count = "  + str( taxiZonesDF.count() ))
```

```
Partitions = 1
Record Count = 265
```

Check partitions for a large dataset

```
# Read Yellow Taxis data
yellowTaxiDF = (
                  spark
                    .read
                    .option("header", "true")
                    .option("inferSchema", "true")
                    .csv("E:\SparkData\Raw\YellowTaxis_202210.csv")
                )

# Check number of partitions
print("Partitions = "    + str( yellowTaxiDF.rdd.getNumPartitions() ))

# Check number of records
print("Record Count = "  + str( yellowTaxiDF.count() ))
```

```
Partitions = 4
Record Count = 1048575
```

Change maximum partition size

```
spark.conf.set( "spark.sql.files.maxPartitionBytes", "64m" )
```

Check partitions for a large dataset

With smaller max partition size (64 MB)

```
# Read Yellow Taxis data
yellowTaxiDF = (
                  spark
                    .read
                    .option("header", "true")
                    .option("inferSchema", "true")
                    .csv("E:\SparkData\Raw\YellowTaxis_202210.csv")
                )

# Check default parallelism
print("Default Parallelism = "  + str( sc.defaultParallelism ))

# Check number of partitions
print("Partitions = "           + str( yellowTaxiDF.rdd.getNumPartitions() ))
```

```
Default Parallelism = 4
Partitions = 4
```

```python
# Read Yellow Taxis data
yellowTaxiDF = (
                  spark
                    .read
                    .option("header", "true")
                    .option("inferSchema", "true")
                    .csv("E:\SparkData\Raw\YellowTaxis_202210.csv")
                )

# Check default parallelism
print("Default Parallelism = "  + str( sc.defaultParallelism ))

# Check number of partitions
print("Partitions = "           + str( yellowTaxiDF.rdd.getNumPartitions() ))
```

```
Default Parallelism = 4
Partitions = 6
```

Create method to calculate DataFrame statistics

Finds data for each partition

Calculate count of records, and min & max values of a column across each partition

```
def getDataFrameStats(dataFrame, columnName):

    outputDF = (
                    dataFrame

                        # Get partition number for each record
                        .withColumn("Partition Number", spark_partition_id())


                        # Group by partition, and calculate stats for a column
                        .groupBy("Partition Number")
                        .agg(
                                count("*").alias("Record Count"),
                                min(columnName).alias("Min Column Value"),
                                max(columnName).alias("Max Column Value")
                            )

                        .orderBy("Partition Number")
                )

    return outputDF
```

Check stats for Yellow Taxis DataFrame

```
getDataFrameStats( yellowTaxiDF, "PULocationID" ).show()
```

```
+----------------+------------+----------------+----------------+
|Partition Number|Record Count|Min Column Value|Max Column Value|
+----------------+------------+----------------+----------------+
|               0|      271544|               1|             265|
|               1|      271149|               1|             265|
|               2|      271128|               1|             265|
|               3|      234754|               1|             265|
+----------------+------------+----------------+----------------+
```

2. Partition settings while shuffling data

Check default number of shuffle partitions

```
spark.conf.get( "spark.sql.shuffle.partitions" )
```

```
'200'
```

Apply a shuffle operation and check DataFrame stats

```
# Group the data
yellowTaxiGroupedDF = (
                            yellowTaxiDF
                                .groupBy("PULocationID")
                                .agg(sum("total_amount"))
                        )

# Check number of partitions
print("Partitions = "  + str( yellowTaxiGroupedDF.rdd.getNumPartitions() ))

# Get DataFrame stats
getDataFrameStats( yellowTaxiGroupedDF, "PULocationID" ).show()
```

```
Partitions = 200
```

Partition Number	Record Count	Min Column Value	Max Column Value
0	1	148	148
1	1	243	243
2	1	31	31
3	3	85	251
4	1	65	65
5	2	53	255
6	1	133	133
7	1	78	78
10	2	108	155
11	3	34	211
12	3	101	126
13	1	81	81
14	3	28	210
18	1	76	76
19	1	26	26
21	3	44	192
22	1	253	253
23	1	236	236
24	1	12	12
25	1	223	223

only showing top 20 rows

Change default number of shuffle partitions

```
spark.conf.set( "spark.sql.shuffle.partitions", 3 )
```

Apply a shuffle operation and check DataFrame stats

After changing default shuffle partitions

```
# Group the data
yellowTaxiGroupedDF = (
                            yellowTaxiDF
                                .groupBy("PULocationID")
                                .agg(sum("total_amount"))
                        )

# Check number of partitions
print("Partitions = "  + str( yellowTaxiGroupedDF.rdd.getNumPartitions() ))

# Get DataFrame stats
getDataFrameStats( yellowTaxiGroupedDF, "PULocationID" ).show()
```

```
Partitions = 3
+----------------+------------+----------------+----------------+
|Partition Number|Record Count|Min Column Value|Max Column Value|
+----------------+------------+----------------+----------------+
|               0|          85|               3|             263|
|               1|          82|               1|             265|
|               2|          85|              11|             264|
+----------------+------------+----------------+----------------+
```

Check stats for Yellow Taxis DataFrame

```
getDataFrameStats( yellowTaxiDF, "PULocationID" ).show()
```

```
+----------------+------------+----------------+----------------+
|Partition Number|Record Count|Min Column Value|Max Column Value|
+----------------+------------+----------------+----------------+
|               0|      271544|               1|             265|
|               1|      271149|               1|             265|
|               2|      271128|               1|             265|
|               3|      234754|               1|             265|
+----------------+------------+----------------+----------------+
```

Changing DataFrame Partitions

Methods to Change DataFrame Partitions

Repartition Method

Coalesce Method

Repartition Method

Typically used to increase number of partitions

Wide transformation – performs shuffling

- Avoid using it to decrease partitions

Partitioning Options

- Round Robin – Creates equal sized partitions
- Hash – Co-locates data based on columns
- Range – Sorts & co-locates data based on columns

Use Cases

- Reduce skewness (some partitions have much more data than others)
- Reduce size of partitions (when they are too big)
- Co-locate data based on certain columns

Coalesce Method

Used for decreasing number of partitions of DataFrame

- Cannot increase partitions

Narrow transformation – no shuffling

Can result in Out-of-Memory exceptions

- Eg: Existing DF = 100 partitions * 100 MB
- Coalesce(1) → New DF = 10,000 MB

Use Cases

- Partitions are very small
- Output is required in lesser number of files

Check stats for Yellow Taxis DataFrame

```
getDataFrameStats( yellowTaxiDF, "PULocationID" ).show()
```

```
+----------------+------------+----------------+----------------+
|Partition Number|Record Count|Min Column Value|Max Column Value|
+----------------+------------+----------------+----------------+
|               0|      271544|               1|             265|
|               1|      271149|               1|             265|
|               2|      271128|               1|             265|
|               3|      234754|               1|             265|
+----------------+------------+----------------+----------------+
```

1. Repartition DataFrame: Round-Robin partitioning

Create equal-sized partitions. Data is not co-located.

```
repartionedDF1 = yellowTaxiDF.repartition( 8 )


# Get DataFrame stats
getDataFrameStats( repartionedDF1, "PULocationID" ).show()
```

```
+----------------+------------+----------------+----------------+
|Partition Number|Record Count|Min Column Value|Max Column Value|
+----------------+------------+----------------+----------------+
|               0|      131071|               1|             265|
|               1|      131071|               1|             265|
|               2|      131072|               1|             265|
|               3|      131072|               1|             265|
|               4|      131072|               1|             265|
|               5|      131072|               1|             265|
|               6|      131073|               1|             265|
|               7|      131072|               1|             265|
+----------------+------------+----------------+----------------+
```

2. Repartition DataFrame: Hash partitioning

Co-locates the data. Partition sizes may not be the same.

4]:
```
# spark.sql.shuffle.partitions = 3


repartionedDF1 = yellowTaxiDF.repartition( "PULocationID" )


# Get DataFrame stats
getDataFrameStats( repartionedDF1, "PULocationID" ).show()
```

```
+----------------+------------+----------------+----------------+
|Partition Number|Record Count|Min Column Value|Max Column Value|
+----------------+------------+----------------+----------------+
|               0|      351490|               3|             263|
|               1|      407935|               1|             265|
|               2|      289150|              11|             264|
+----------------+------------+----------------+----------------+
```

2. Repartition DataFrame: Hash partitioning & define number of partitions

Co-locates the data. Partition sizes may not be the same.

```
repartionedDF1 = yellowTaxiDF.repartition( 8, "PULocationID" )


# Get DataFrame stats
getDataFrameStats( repartionedDF1, "PULocationID" ).show()
```

```
+----------------+------------+----------------+----------------+
|Partition Number|Record Count|Min Column Value|Max Column Value|
+----------------+------------+----------------+----------------+
|               0|      164216|              12|             265|
|               1|      160519|               6|             250|
|               2|       31660|               5|             259|
|               3|      123461|               1|             258|
|               4|       74579|              25|             254|
|               5|      204714|              15|             260|
|               6|      149760|               4|             264|
|               7|      139666|               8|             241|
+----------------+------------+----------------+----------------+
```

3. Repartition DataFrame: Range partitioning

Sort and co-locates the data. Partition sizes may not be the same.

```
repartionedDF1 = yellowTaxiDF.repartitionByRange( 8, "PULocationID" )


# Get DataFrame stats
getDataFrameStats( repartionedDF1, "PULocationID" ).show()
```

```
+----------------+------------+----------------+----------------+
|Partition Number|Record Count|Min Column Value|Max Column Value|
+----------------+------------+----------------+----------------+
|               0|      144718|               1|              79|
|               1|      161087|              80|             132|
|               2|      126488|             133|             142|
|               3|      125653|             143|             162|
|               4|      121298|             163|             186|
|               5|      126336|             188|             234|
|               6|      144991|             235|             239|
|               7|       98004|             240|             265|
+----------------+------------+----------------+----------------+
```

Coalesce DataFrame

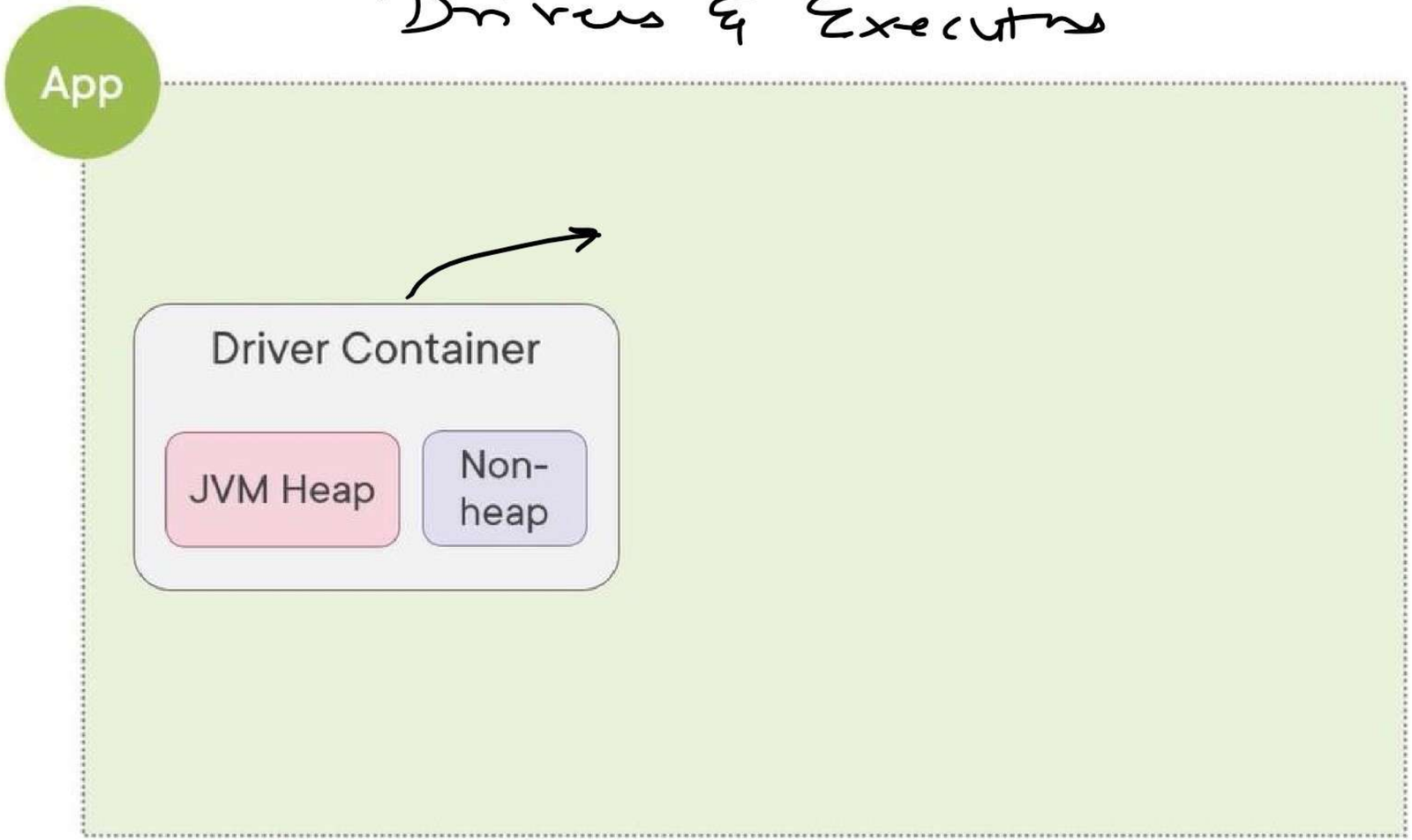
Reduces number of partitions

```
coalescedDF = yellowTaxiDF.coalesce( 2 )


# Get DataFrame stats
getDataFrameStats( coalescedDF, "PULocationID" ).show()
```

```
+----------------+------------+----------------+----------------+
|Partition Number|Record Count|Min Column Value|Max Column Value|
+----------------+------------+----------------+----------------+
|               0|      542693|               1|             265|
|               1|      505882|               1|             265|
+----------------+------------+----------------+----------------+
```

Memory Management

Drivers & Executors



JVM Heap memory is used for Spark activity

Overhead memory (non-heap memory) is used by non-JVM processes like VM overheads, buffers etc.

Driver Container

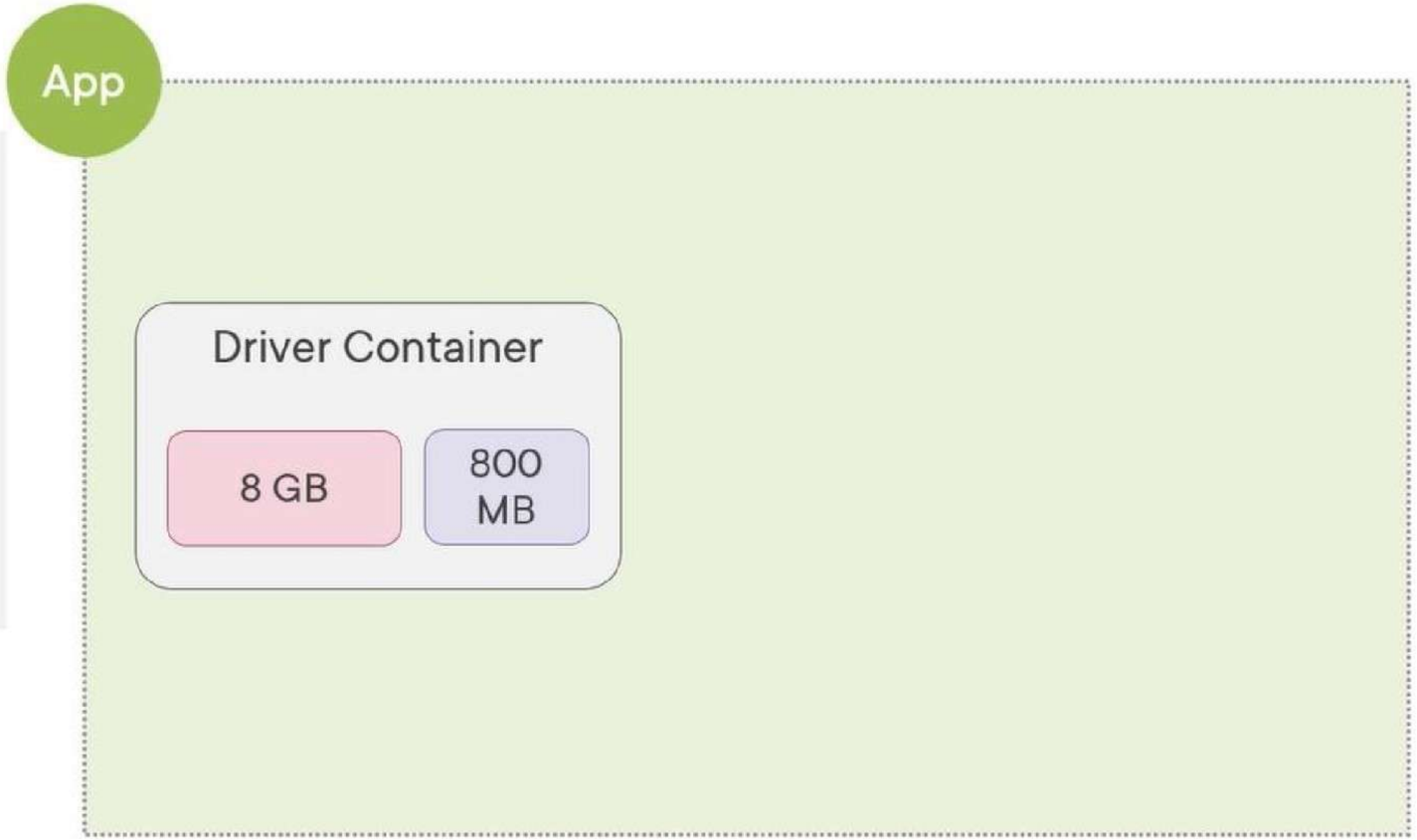
spark.driver.cores = 2

spark.driver.memory = 8 GB → RAM

spark.driver.memoryOverhead

= max (10% of memory or 384 MB)

= 800MB



Driver Container

spark.driver.cores = 2

spark.driver.memory = 8 GB

spark.driver.memoryOverhead

= max (10% of memory or 384 MB)

= 800MB

Executor Container

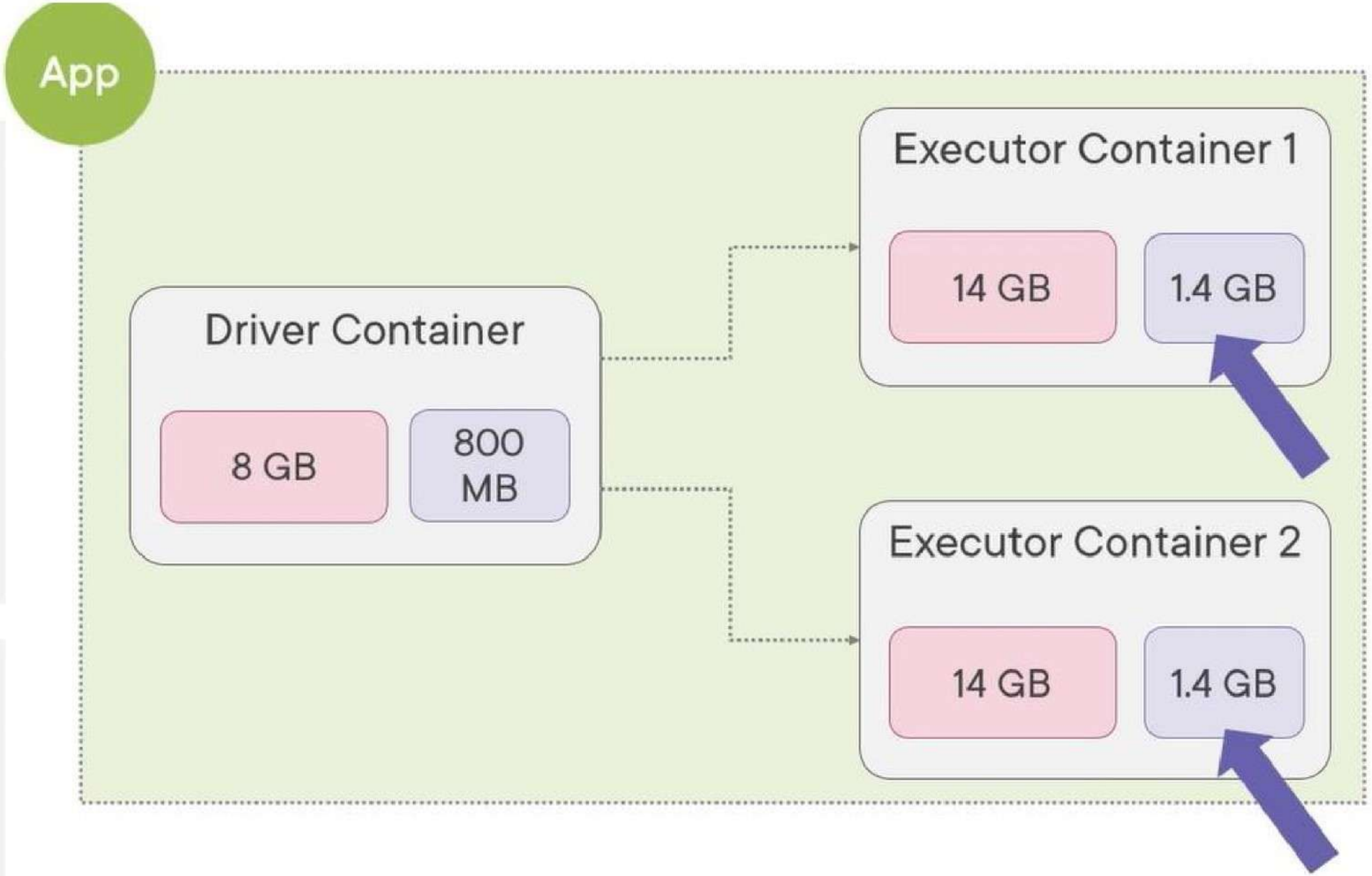
spark.executor.cores = 4

spark.executor.memory = 14 GB

spark.executor.memoryOverhead

= max (10% of memory or 384 MB)

= 1.4 GB



Executor JVM Heap

14,000 MB JVM Heap Memory

spark.executor.cores	= 4
spark.executor.memory	= 14 GB

Executor JVM Heap

300 MB

Reserved Memory = 300 MB

Reserved by Spark for internal purposes

Spark Memory
spark.memory.fraction

(14000 MB – 300 MB) * 60% = 8220 MB

For task execution, caching, shuffling, DataFrame operations etc.

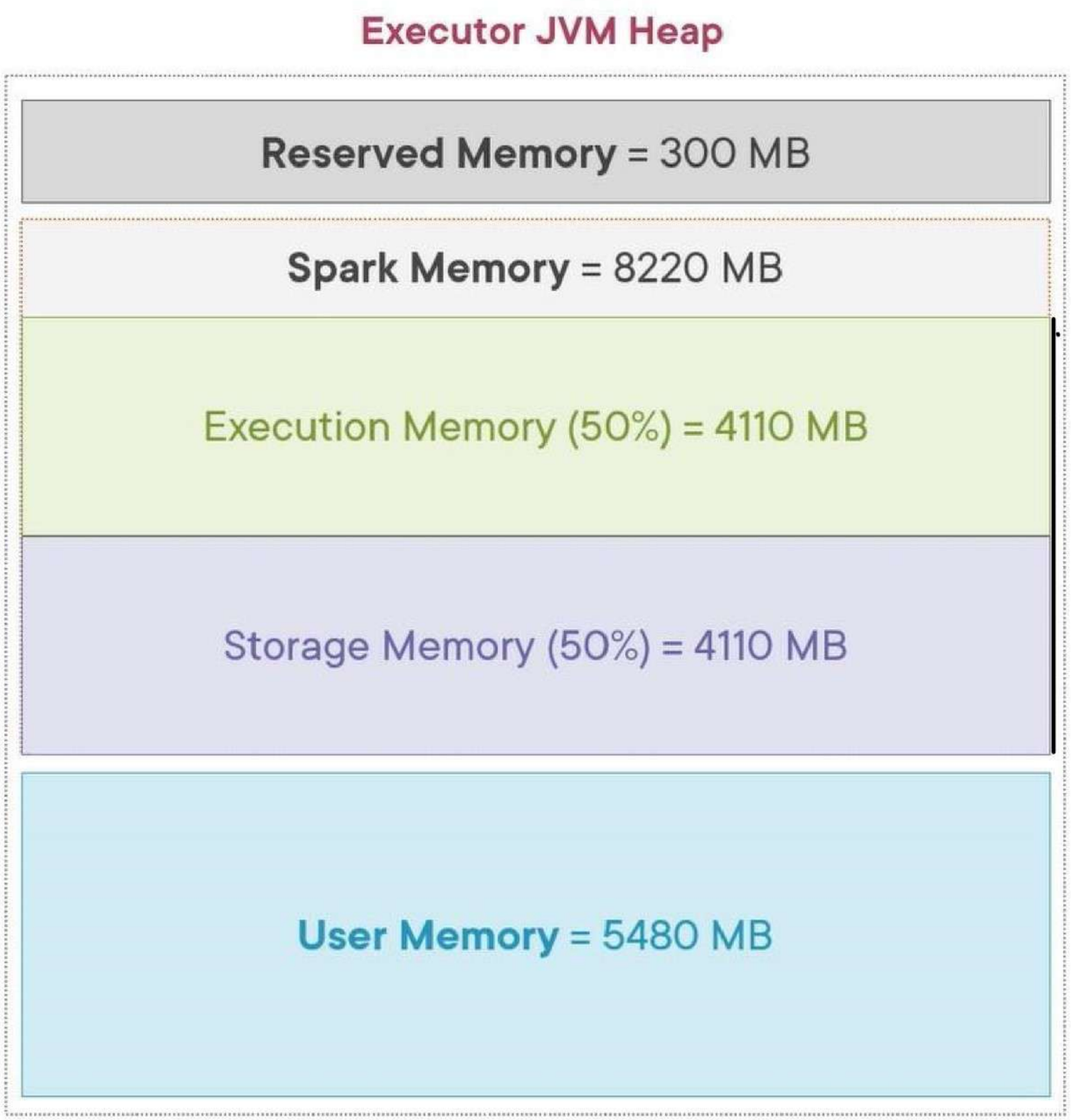
spark.executor.cores	= 4
spark.executor.memory	= 14 GB
spark.memory.fraction	= 0.6

60%

User Memory

(14000 MB – 300 MB) * 40% = 5480 MB

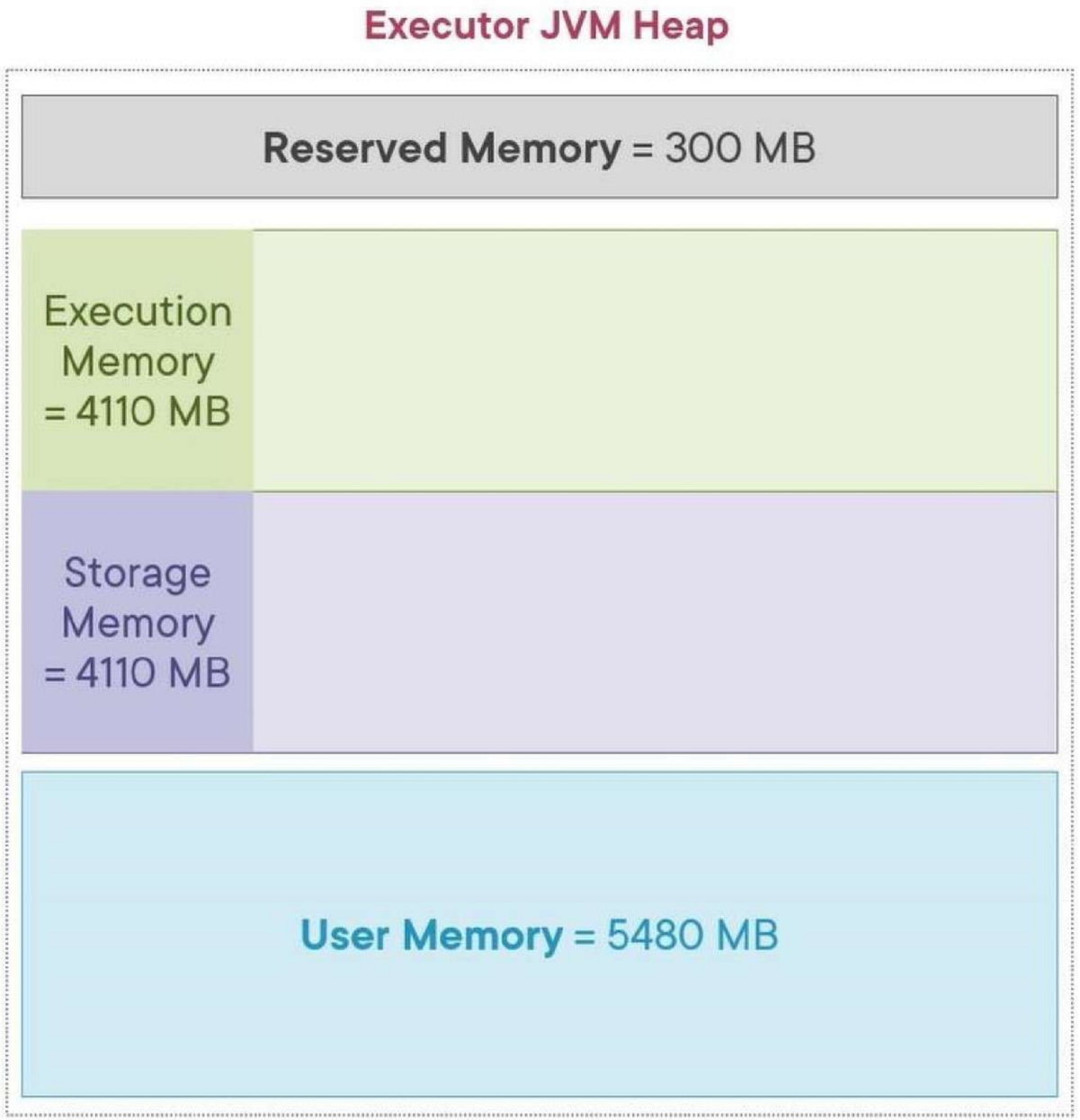
For storing data structures created by user, metadata, RDD operations, UDFs etc.



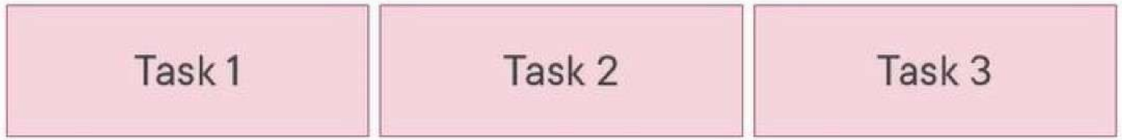
For DataFrame operations, task execution, shuffling etc.

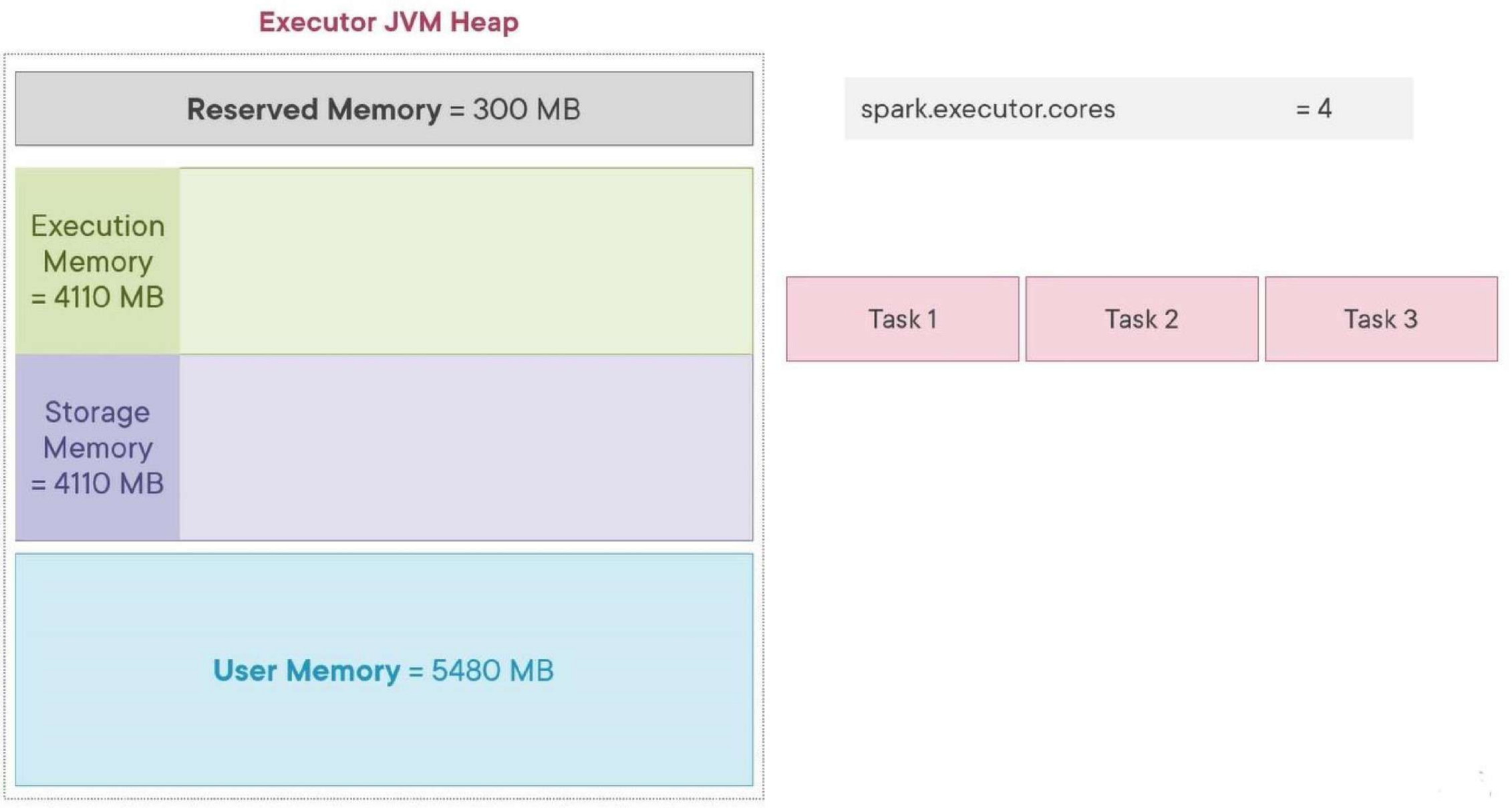
For storing cached data

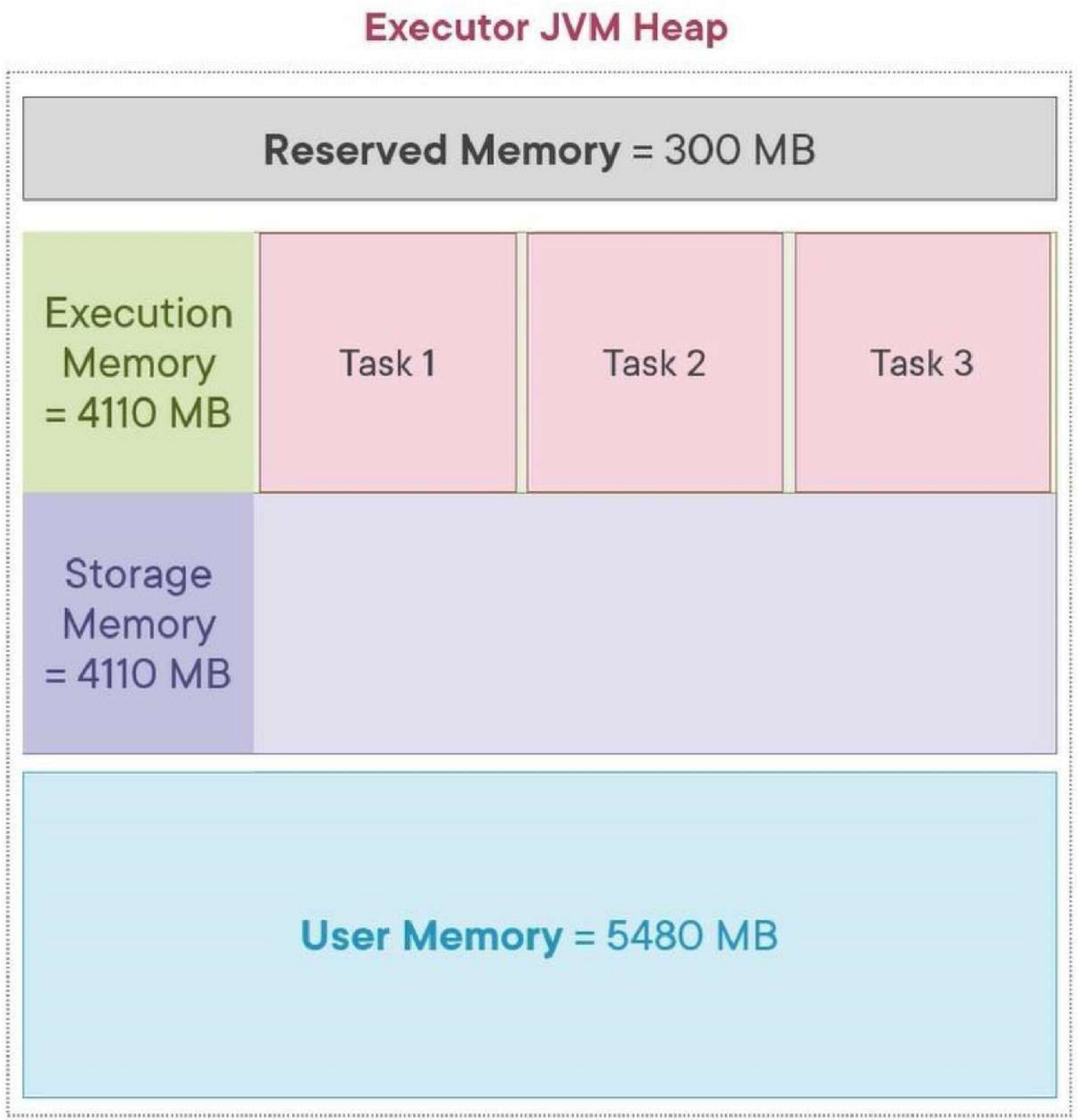
spark.executor.cores	= 4
spark.executor.memory	= 14 GB
spark.memory.fraction	= 0.6
spark.memory.storageFraction	= 0.5



spark.executor.cores = 4

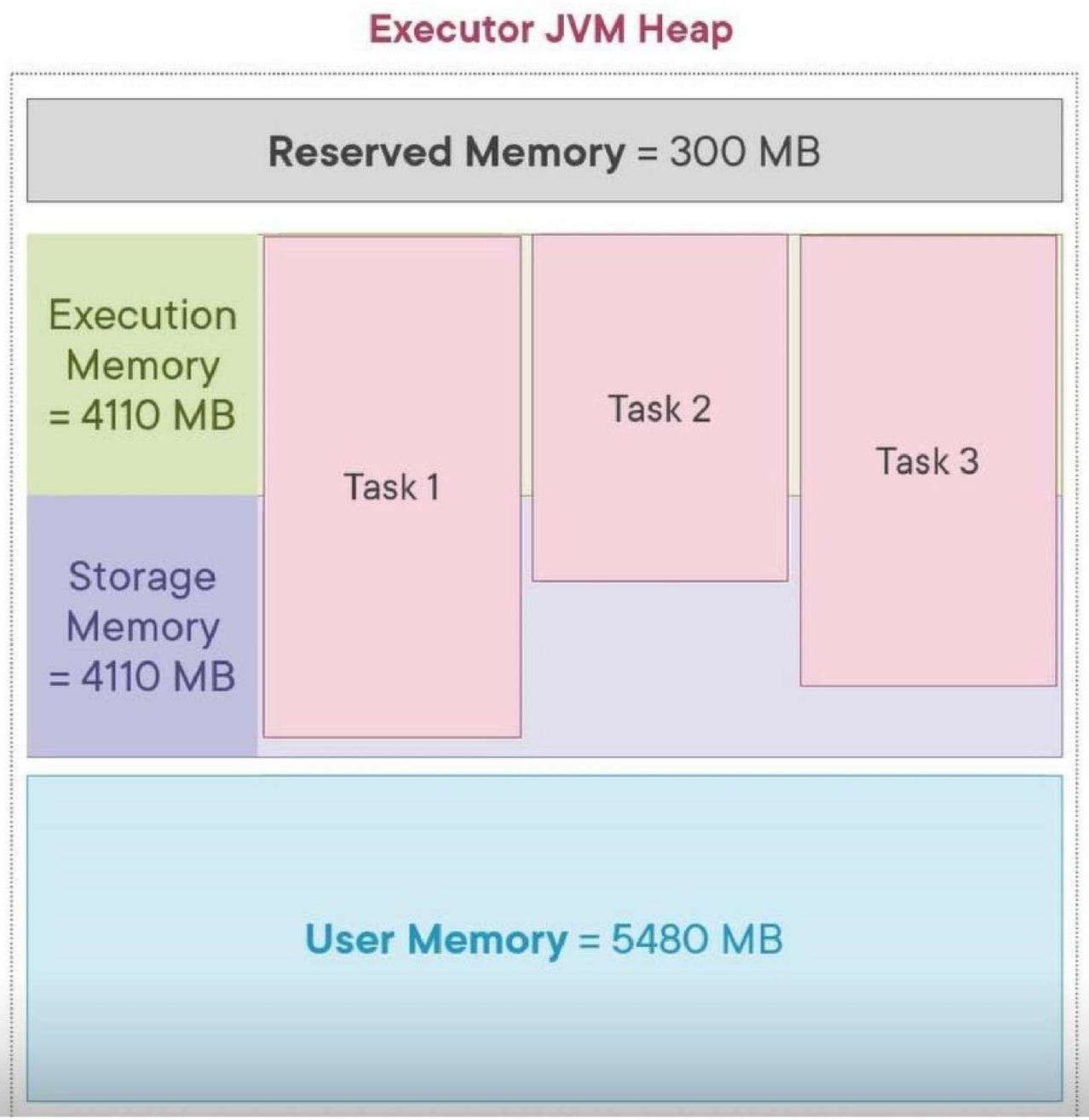



Executor JVM Heap
Reserved Memory = 300 MB
Execution Memory = 4110 MB
Storage Memory = 4110 MB
User Memory = 5480 MB
spark.executor.cores = 4
Task 1
Task 2
Task 3



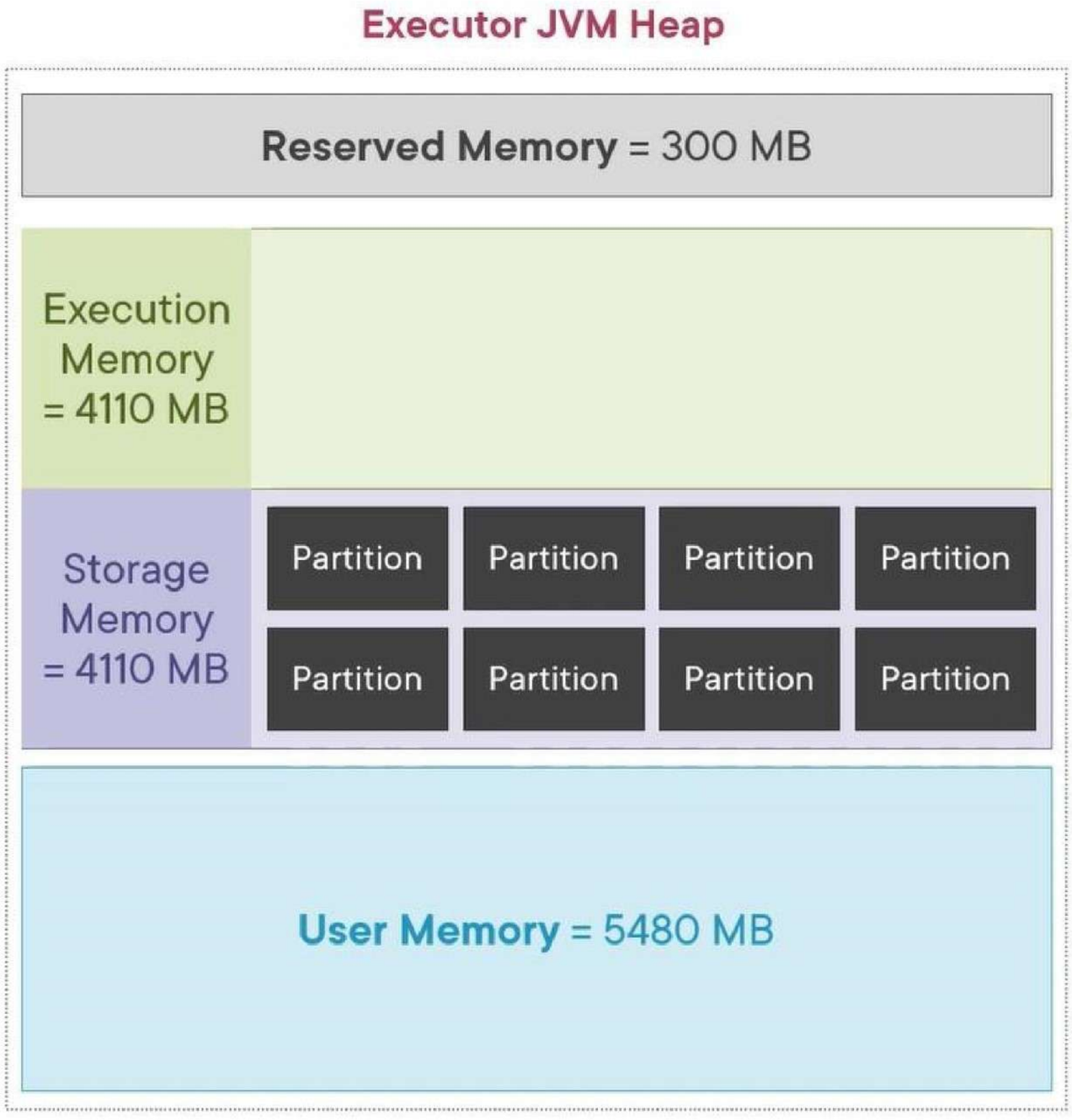
spark.executor.cores	= 4

Now let's say the value of spark.executor.cores is set to four. With each executor having four cores it can execute a maximum of four tasks in parallel. Now let's say you need to run three tasks. So all three of them will execute in parallel and share the same execution memory.

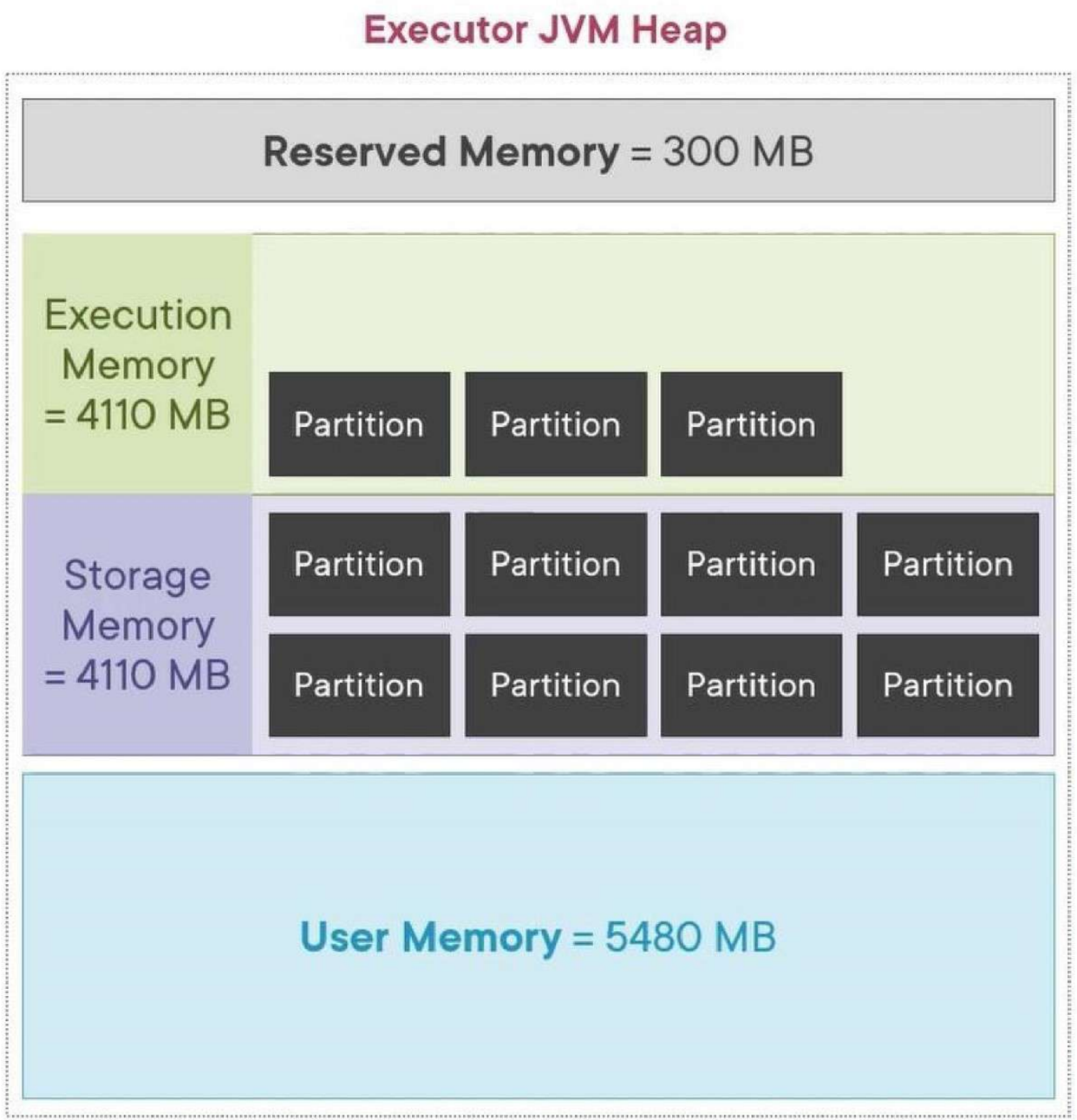


spark.executor.cores	= 4

In fact, if they need more memory, unified memory manager allows them to use storage memory as long as it is available. So this helps in optimal utilization



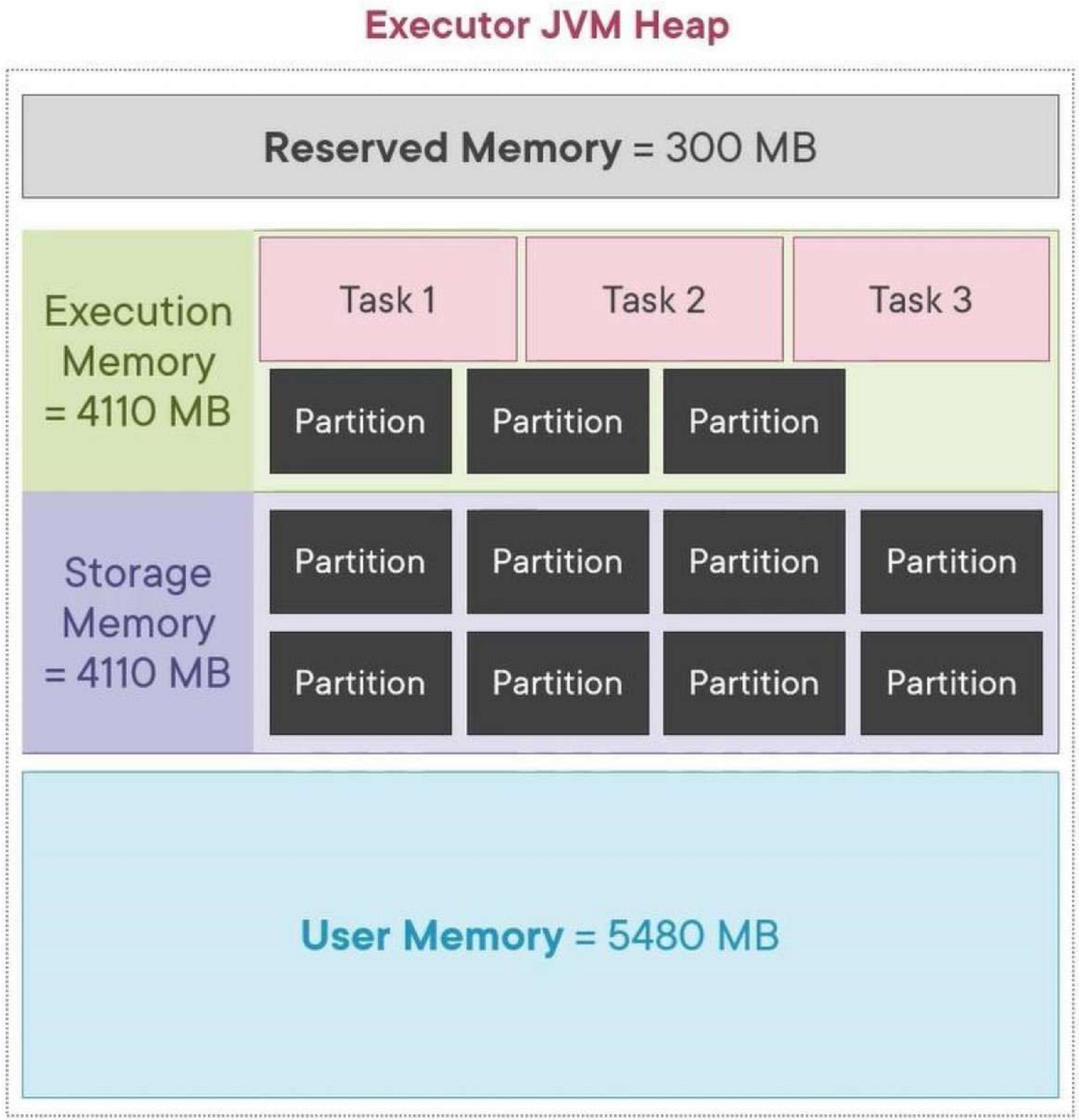
spark.executor.cores	= 4



spark.executor.cores	= 4

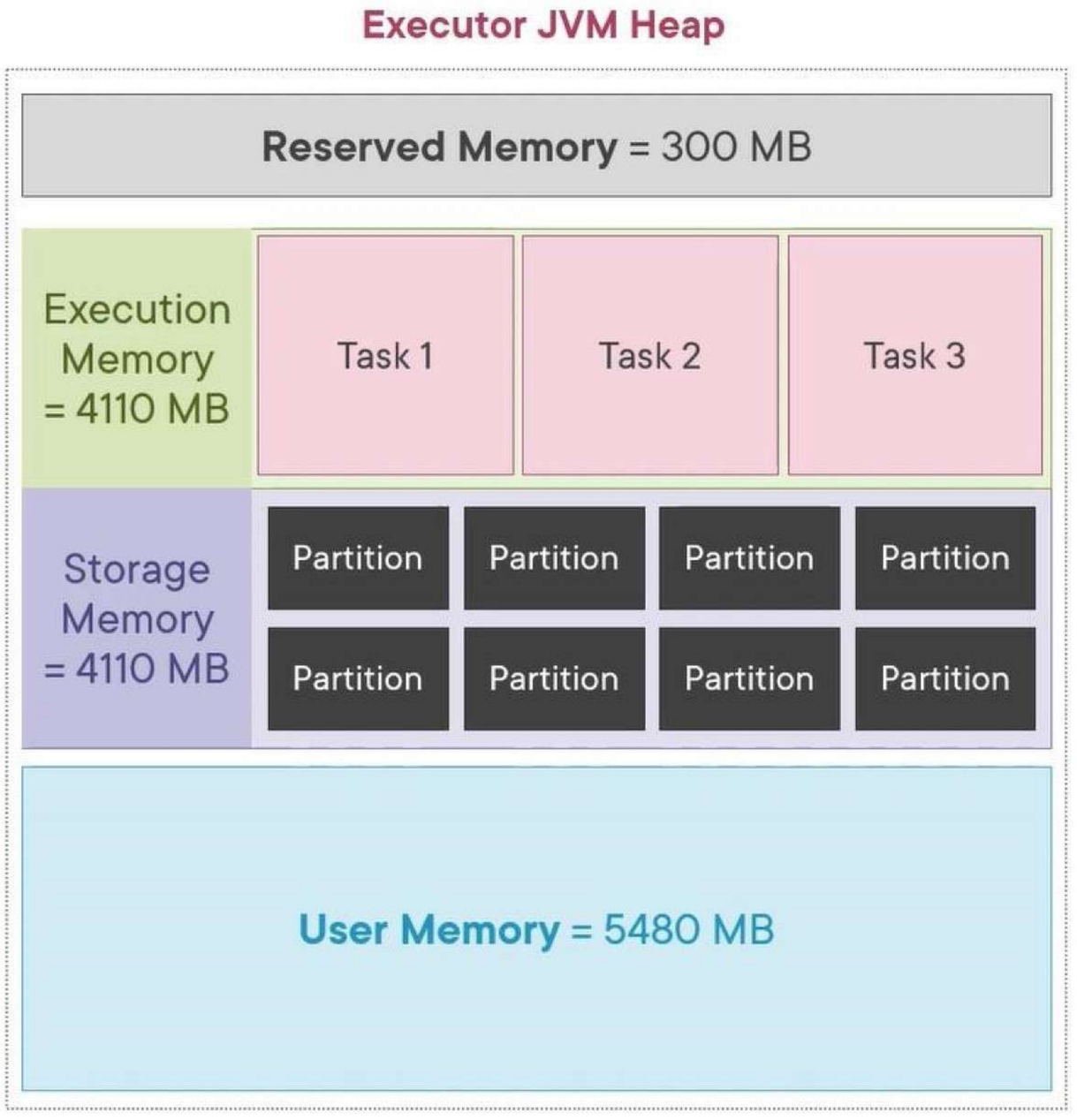
Now, in case the data being cached is more than storage memory, unified memory manager allows to cache these partitions in execution memory if it is available.

MEMORY
DISK



spark.executor.cores	= 4

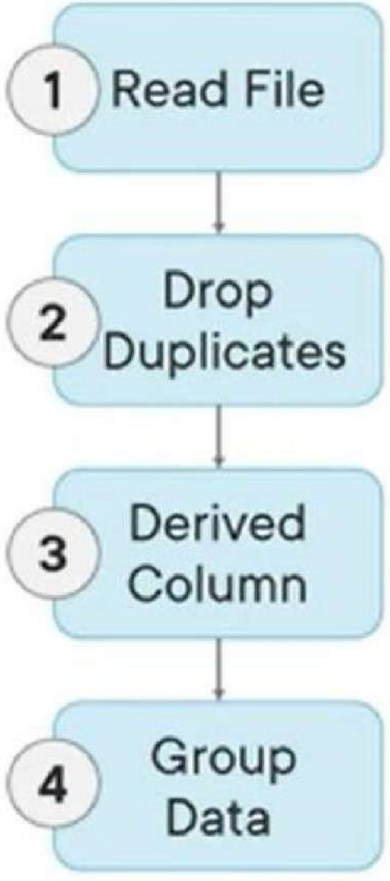
Now let's say you want to execute the tasks, so they will use execution memory, but if they need more, Spark will evict the partitions in execution memory and spill it over to disk, and then tasks can use the memory



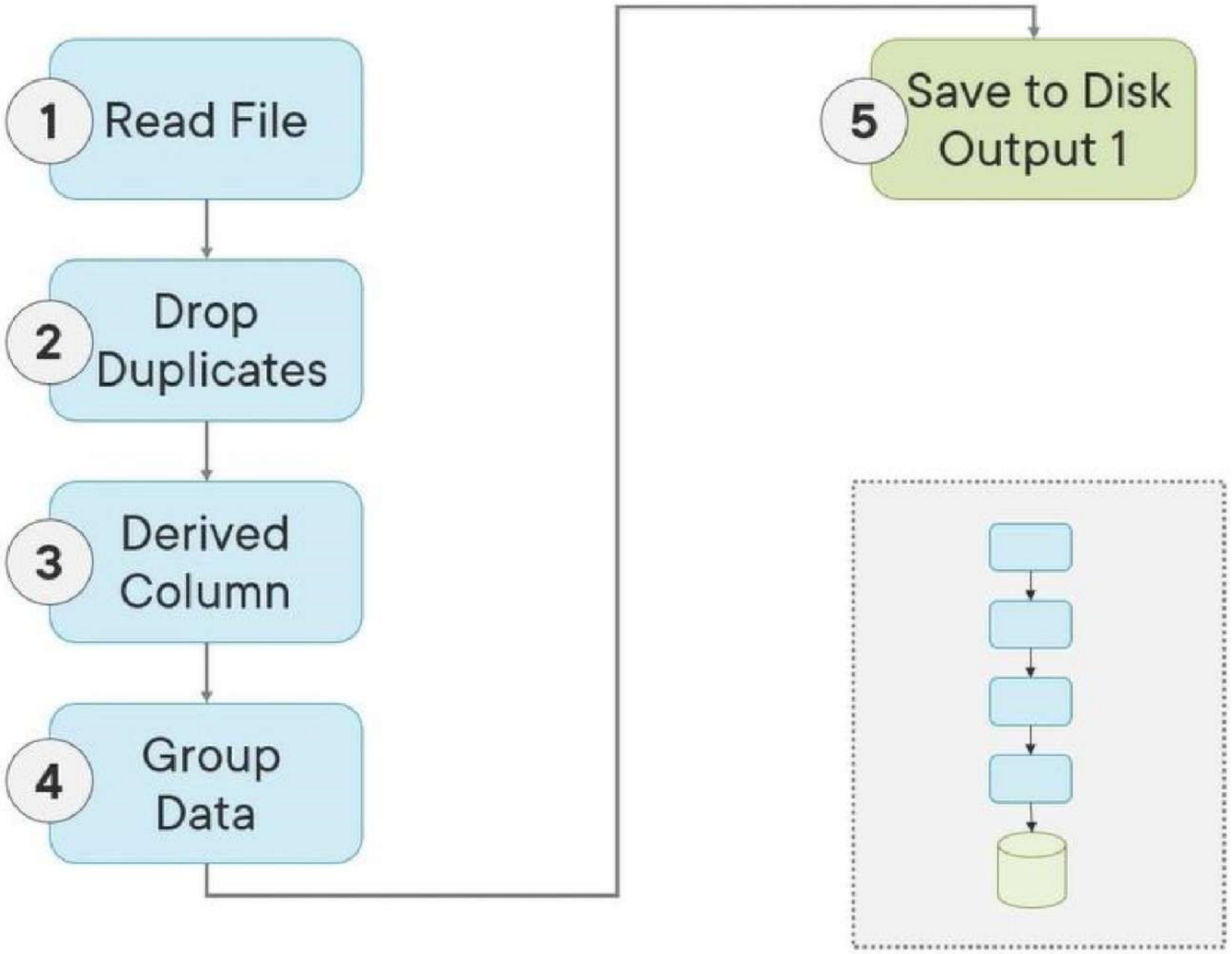
spark.executor.cores	= 4

If Tasks need more memory, it may result in Out-of-Memory exceptions

Persisting Data

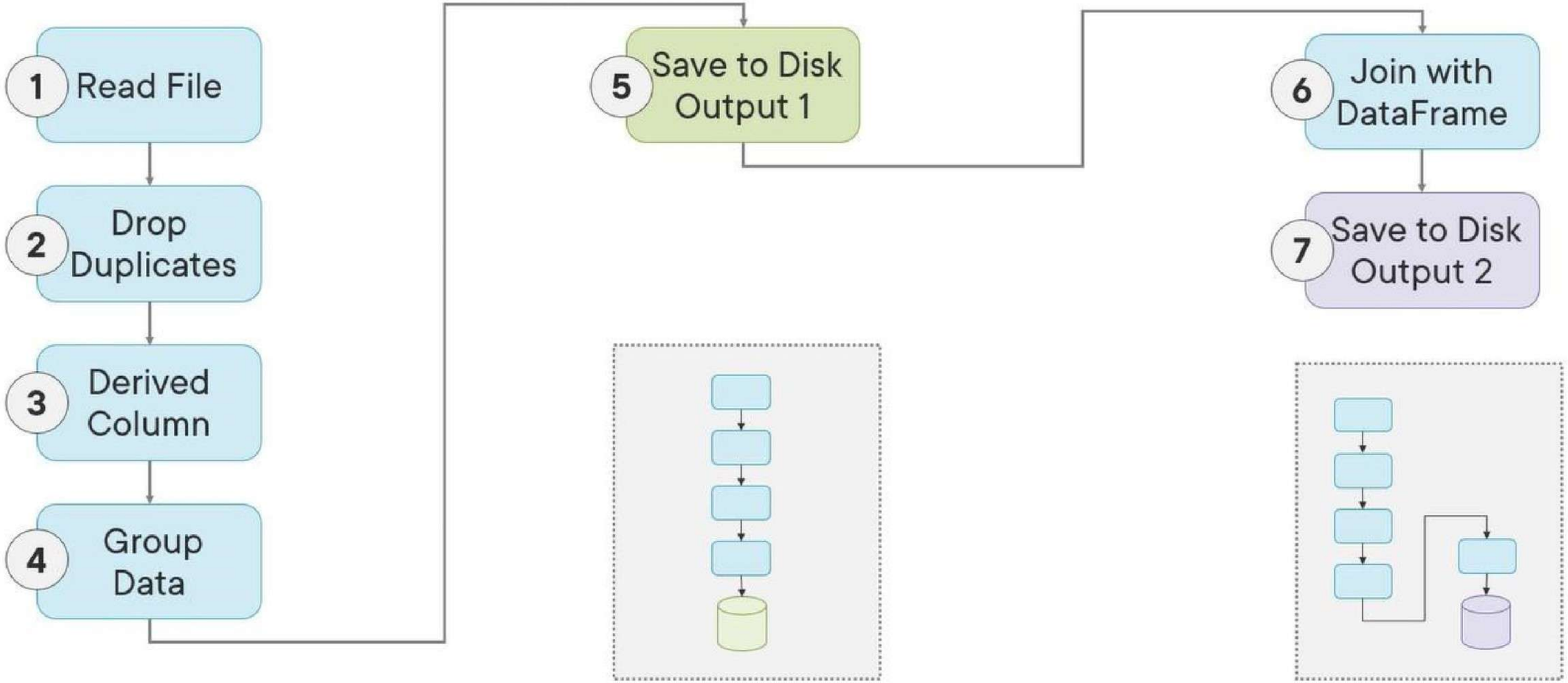


Transformations
Operations



Transformations Operations

Action 1
Executes all 4 transformations, & writes to disk



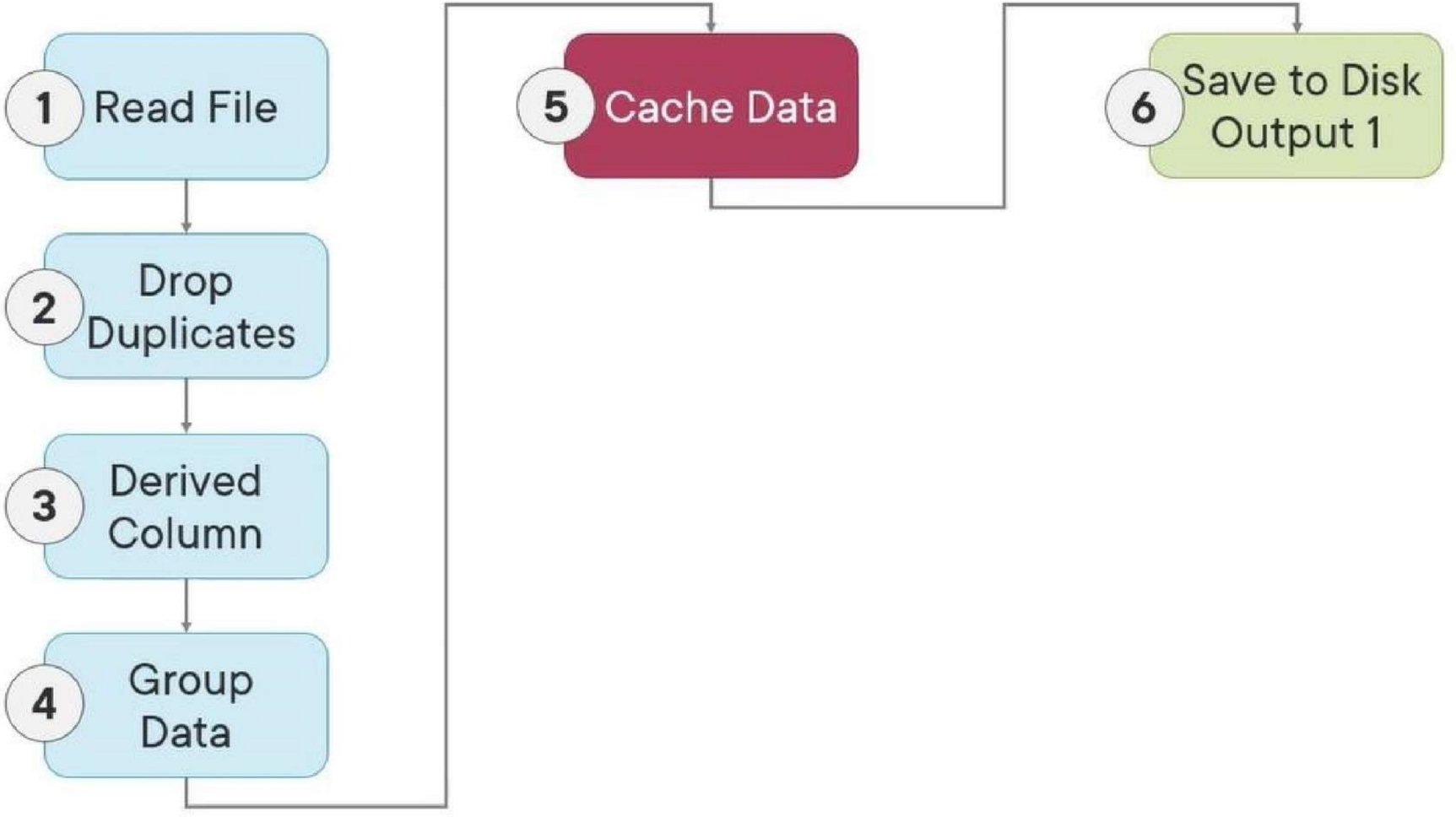
Transformations Operations

Action 1
Executes all 4 transformations, & writes to disk

Action 2
Executes all 5 transformations, & writes to disk

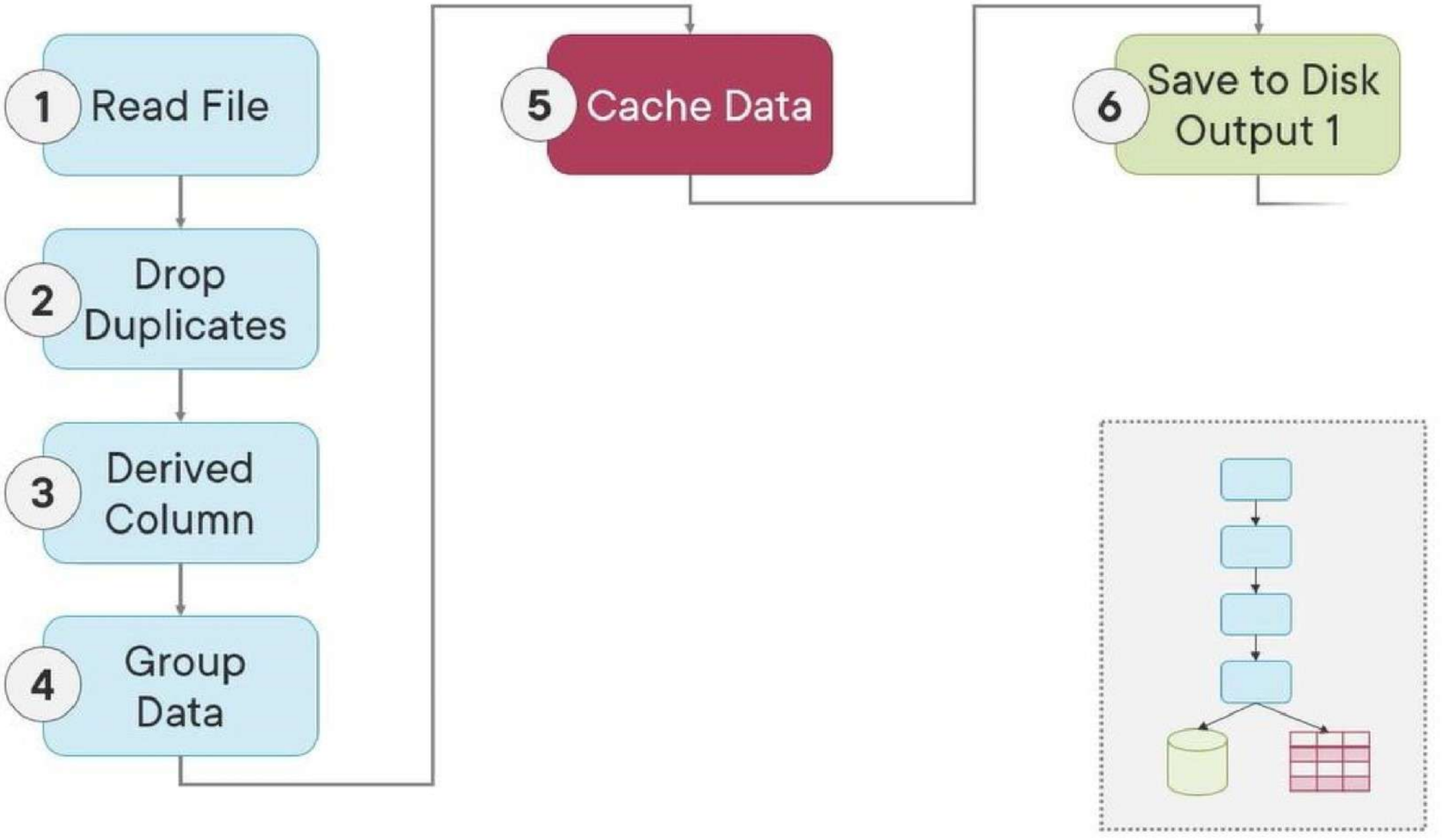
Persist data to avoid re-computation of complex transformations with every Action operation

Persisting data is a Lazy operation



Transformations Operations

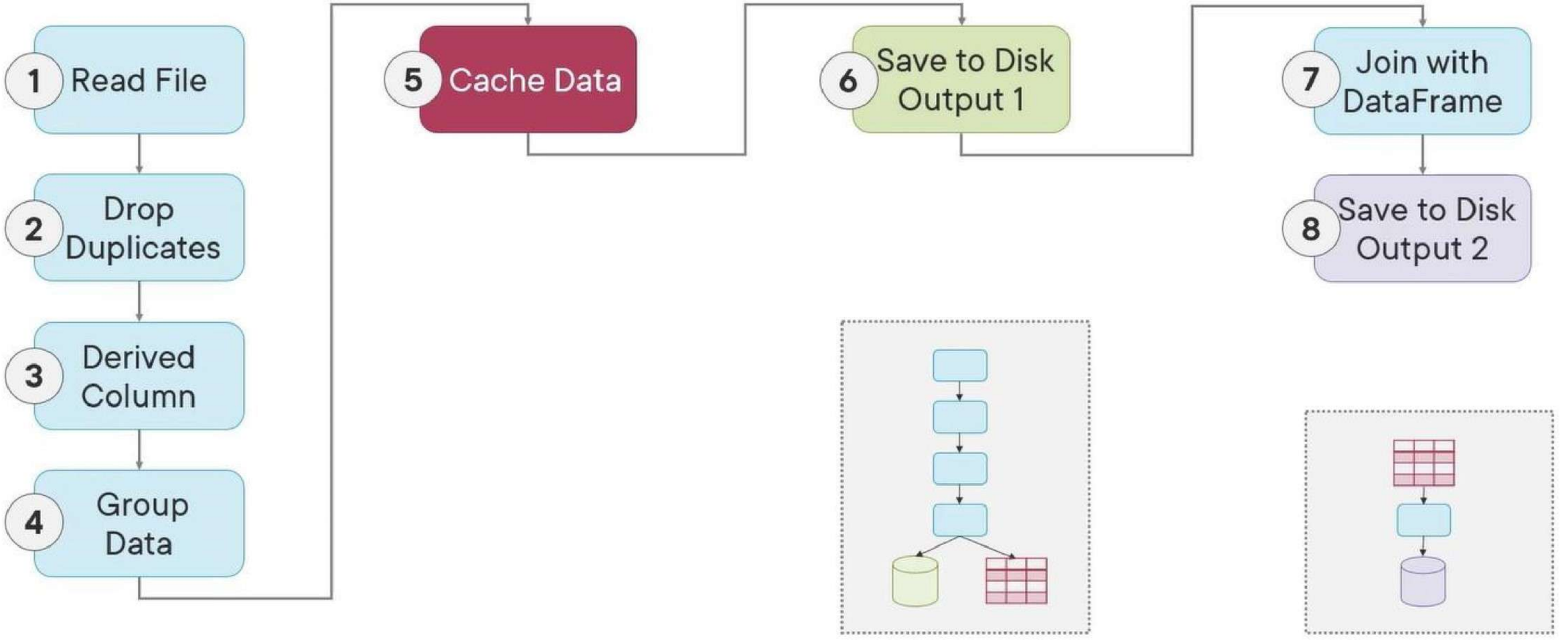
No data is cached
Lazy operation



Transformations Operations

No data is cached
Lazy operation

Action 1
Executes all 4 transformations, caches data, & writes to disk

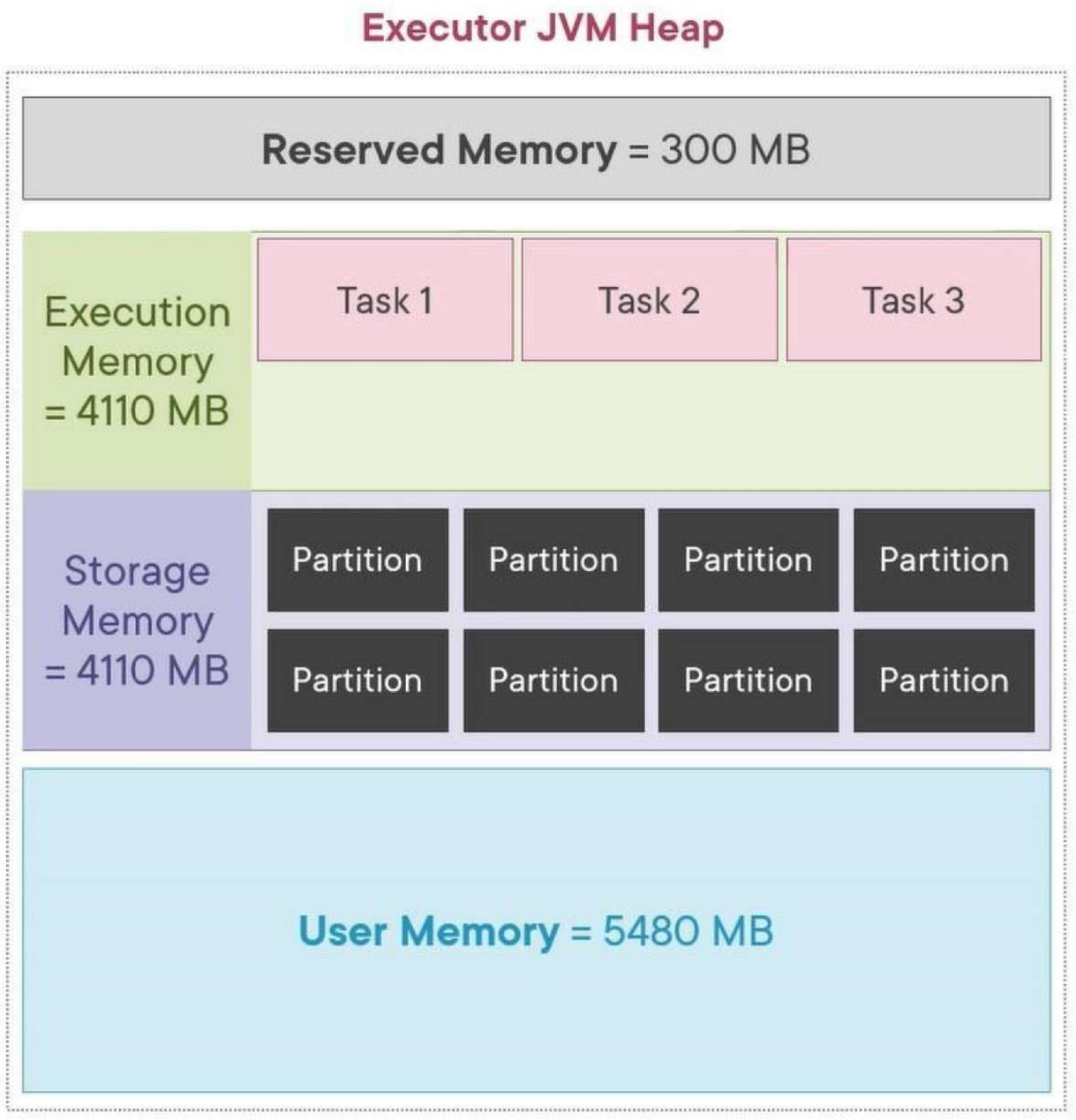


Transformations Operations

No data is cached
Lazy operation

Action 1
Executes all 4 transformations, caches data, & writes to disk

Action 2
Uses cached data applies join, & writes to disk



Partitions can be cached in Storage Memory

If Execution Memory is free, partitions can be cached there too

Partitions will be evicted from Execution Memory if tasks need them

Partitions can be spilled over to disk if Storage Memory does not have enough space

Cache can be manually evicted

Partitions are evicted from cache in LRU (Least-Recently-Used) fashion

Caching Methods

Two methods – cache() and persist()

Applying cache() and persist() without arguments

- On RDD: MEMORY_ONLY
- On DataFrame: MEMORY_AND_DISK

cache() has no arguments

persist() supports Storage Level as argument

- MEMORY_ONLY: Partitions that can fit in memory are cached; others are recomputed each time
- MEMORY_AND_DISK: Partitions that can fit in memory are cached; others are spilled to local disk of Worker
- DISK_ONLY: Only stored on disk; pulled when required
- *more...*

Recommended to unpersist data if not required

```python
import findspark
findspark.init()
findspark.find()
```

```
'E:\\Local d\\Zartab\\Apache Spark Installation\\spark-3.3.2-bin-hadoop3'
```

```python
from pyspark.sql import SparkSession
from pyspark.sql.types import *
from pyspark.sql.functions import *

spark = (
            SparkSession
                .builder
                .appName("PersistingDataApp")
                .master("local[4]")

                .config("spark.dynamicAllocation.enabled", "false")
                .config("spark.sql.adaptive.enabled", "false")

                .getOrCreate()
        )

sc = spark.sparkContext

spark
```

```python
# Read Yellow Taxis data
yellowTaxiDF = (
                  spark
                    .read
                    .option("header", "true")
                    .option("inferSchema", "true")
                    .csv("E:\SparkData\Raw\YellowTaxis_202210.csv")
                )

# Aggregate the data
yellowTaxiGroupedDF = (
                            yellowTaxiDF
                                .dropDuplicates()

                                .groupBy("PULocationID")
                                .agg(sum("total_amount"))
                        )
```

Save data without caching

```
(
    yellowTaxiGroupedDF
            .write

            .option("header", "true")
            .option("dateFormat", "yyyy-MM-dd HH:mm:ss.S")

            .mode("overwrite")

            .csv("E:\SparkData\Raw\Output\CacheTest_WithoutEnabling.csv")
)
```

Spark 3.3.2 | Jobs | Stages | Storage | Environment | Executors | SQL / DataFrame | PersistingDataApp application U

Spark Jobs (?)

User: VICTUS
Total Uptime: 18 min
Scheduling Mode: FIFO
Completed Jobs: 3

▸ Event Timeline

▾ Completed Jobs (3)

Page: 1 | 1 Pages. Jump to 1 . Show 100 items in a page. Go

Job Id ▾	Description	Submitted	Duration	Stages: Succeeded/Total	Tasks (for all stages): Succeeded/Total
2	csv at NativeMethodAccessorImpl.java:0 csv at NativeMethodAccessorImpl.java:0	2023/05/20 00:35:20	31 s	3/3	404/404
1	csv at NativeMethodAccessorImpl.java:0 csv at NativeMethodAccessorImpl.java:0	2023/05/20 00:35:09	4 s	1/1	4/4

Apply persist operation on DataFrame

This will not cache data because it's a lazy operation

```
import pyspark

yellowTaxiGroupedDF.persist( pyspark.StorageLevel.MEMORY_AND_DISK )
```

```
DataFrame[PULocationID: int, sum(total_amount): double]
```

Storage

Save data with caching enabled

This will save data to disk and cache the data

```
(
    yellowTaxiGroupedDF
            .write

            .option("header", "true")
            .option("dateFormat", "yyyy-MM-dd HH:mm:ss.S")

            .mode("overwrite")

            .csv("E:\SparkData\Raw\Output\CacheTest_EnabledFirstTime.csv")
)
```

Spark 3.3.2 | Jobs | Stages | Storage | Environment | Executors | SQL / DataFrame | **PersistingDataApp** application UI

Spark Jobs (?)

User: VICTUS
Total Uptime: 22 min
Scheduling Mode: FIFO
Completed Jobs: 4

▸ Event Timeline

▾ Completed Jobs (4)

Page: 1 | 1 Pages. Jump to 1 . Show 100 items in a page. Go

Job Id ▾	Description	Submitted	Duration	Stages: Succeeded/Total	Tasks (for all stages): Succeeded/Total
3	csv at NativeMethodAccessorImpl.java:0 csv at NativeMethodAccessorImpl.java:0	2023/05/20 00:39:57	22 s	3/3	404/404
2	csv at NativeMethodAccessorImpl.java:0 csv at NativeMethodAccessorImpl.java:0	2023/05/20 00:35:20	31 s	3/3	404/404
1	csv at NativeMethodAccessorImpl.java:0 csv at NativeMethodAccessorImpl.java:0	2023/05/20 00:35:09	4 s	1/1	4/4

Spark 3.3.2 Jobs Stages Storage Environment Executors SQL / DataFrame PersistingDataApp application UI

Storage

RDDs

ID	RDD Name	Storage Level	Cached Partitions	Fraction Cached	Size in Memory	Size on Disk
29	*(3) HashAggregate(keys=[PULocationID#24], functions=[sum(total_amount#33)], output=[PULocationID#24, sum(total_amount)#76]) +- Exchange hashpartitioning(PULocationID#24, 200), ENSURE_REQUIREMENTS, [plan_id=106] +- *(2) HashAggregate(keys=[PULocationID#24], functions=[partial_sum(total_amount#33)], output=[PULocationID#24, sum#82]) +- *(2) HashAggregate(keys=[DOLocationID#25, improvement_surcharge#32, tpep_dropoff_datetime#19, PULocationID#24, trip_distance#21, tolls_amount#31, RatecodeID#22, VendorID#17, tip_amount#30, payment_type#26, fare_amount#27, passenger_count#20, store_and_fwd_flag#23, extra#28, airport_fee#35, congestion_surcharge#34, total_amount#33, tpep_pickup_datetime#18, mta_tax#29], functions=[], output=[PULocationID#24, total_amount#33]) +- Exchange hashpartitioning(DOLocationID#25, improvement_surcharge#32, tpep_dropoff_datetime#19, PULocationID#24, trip_distance#21, tolls_amount#31, RatecodeID#22, VendorID#17, tip_amount#30, payment_type#26, fare_amount#27, passenger_co...	Disk Memory Serialized 1x Replicated	200	100%	84.5 KiB	0.0 B

Save data to disk by using persisted data

This will take cached data and write to disk

```
(
    yellowTaxiGroupedDF
            .write

            .option("header", "true")
            .option("dateFormat", "yyyy-MM-dd HH:mm:ss.S")

            .mode("overwrite")

            .csv("E:\SparkData\Raw\Output\CacheTest_EnabledAndCached.csv")
)
```

Spark 3.3.2 | Jobs | Stages | Storage | Environment | Executors | SQL / DataFrame | **PersistingDataApp** application U

Spark Jobs (?)

User: VICTUS
Total Uptime: 23 min
Scheduling Mode: FIFO
Completed Jobs: 5

▸ Event Timeline

▾ Completed Jobs (5)

Page: 1

1 Pages. Jump to 1 . Show 100 items in a page. Go

Job Id ▾	Description	Submitted	Duration	Stages: Succeeded/Total	Tasks (for all stages): Succeeded/Total
4	csv at NativeMethodAccessorImpl.java:0 csv at NativeMethodAccessorImpl.java:0	2023/05/20 00:41:34	2 s	1/1 (2 skipped)	200/200 (204 skipped)
3	csv at NativeMethodAccessorImpl.java:0 csv at NativeMethodAccessorImpl.java:0	2023/05/20 00:39:57	22 s	3/3	404/404
2	csv at NativeMethodAccessorImpl.java:0 csv at NativeMethodAccessorImpl.java:0	2023/05/20 00:35:20	31 s	3/3	404/404
1	csv at NativeMethodAccessorImpl.java:0 csv at NativeMethodAccessorImpl.java:0	2023/05/20 00:35:09	4 s	1/1	4/4

Unpersist data

This will remove cached partitions

```
yellowTaxiGroupedDF.unpersist()
```

```
DataFrame[PULocationID: int, sum(total_amount): double]
```

Storage

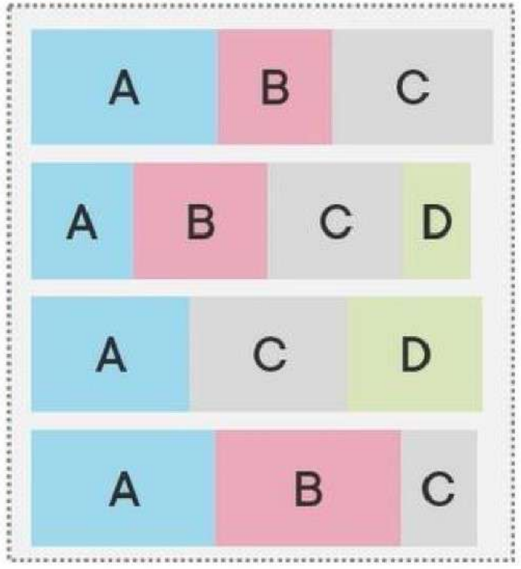
Spark Join Strategies and Broadcast Joins

Spark Join Strategy determines how to move, shuffle, sort, group & merge data across executors during join operation

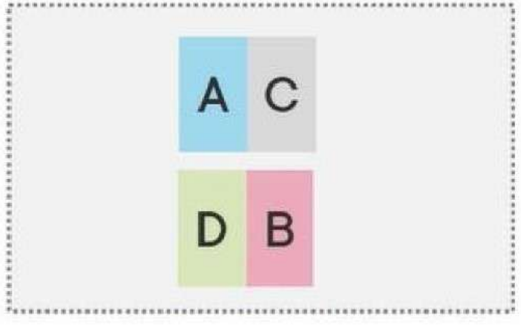
Join Strategies

- Broadcast Hash Join
- Shuffle Sort Merge Join
- Shuffle Hash Join
- Cartesian Join
- Broadcast Nested Loop Join

Shuffle Sort Merge Join

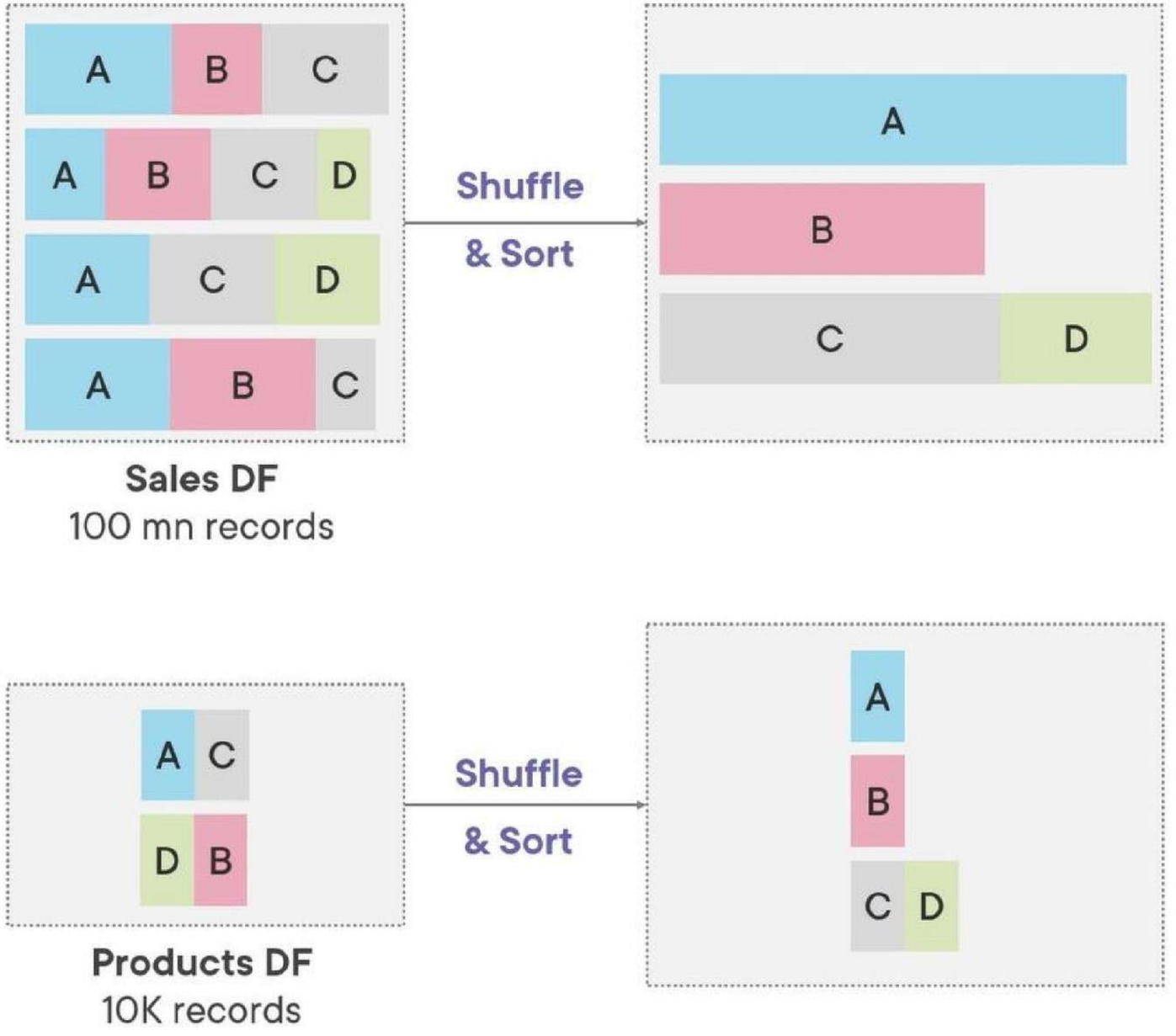


Sales DF
100 mn records

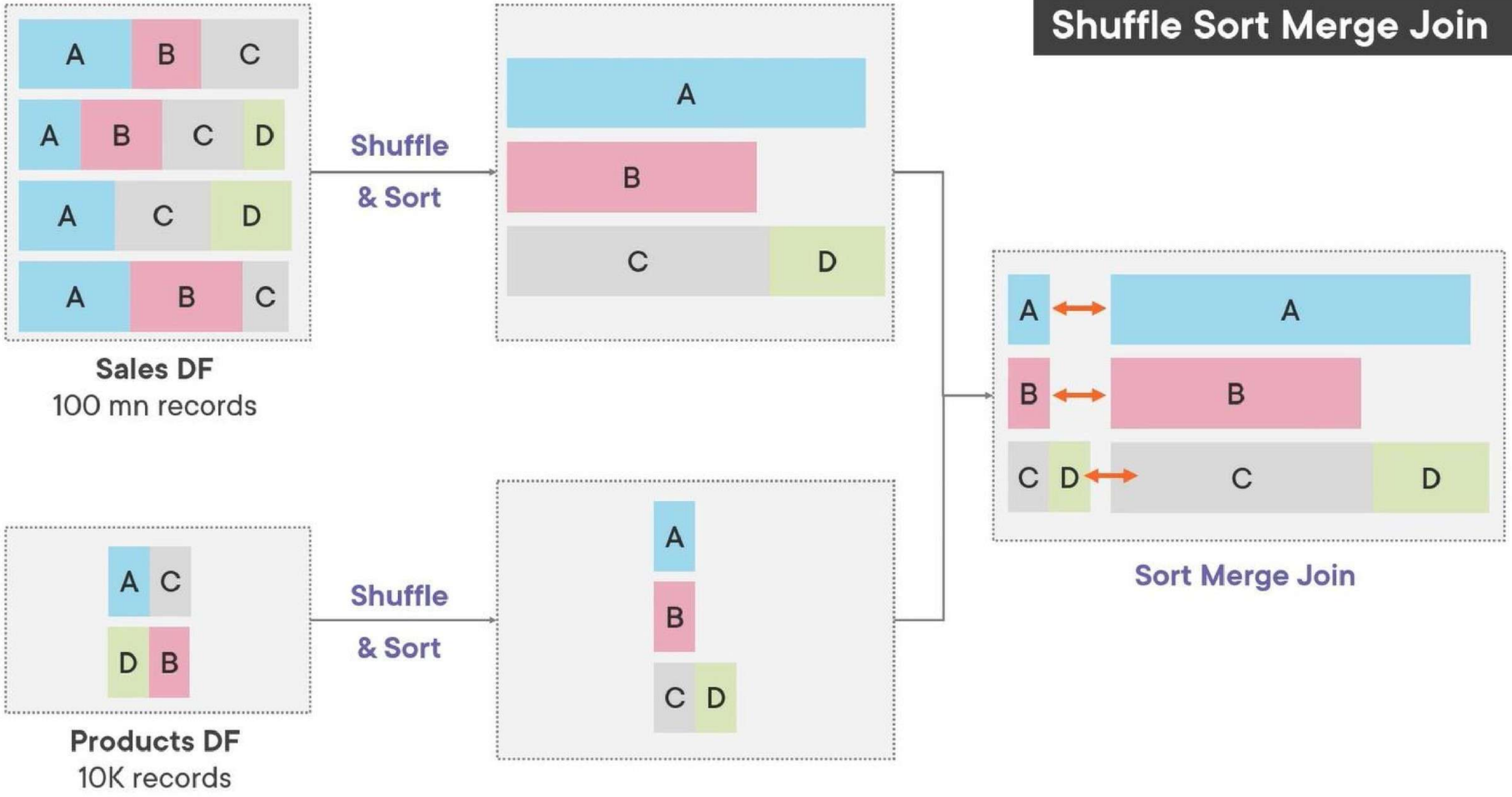


Products DF
10K records

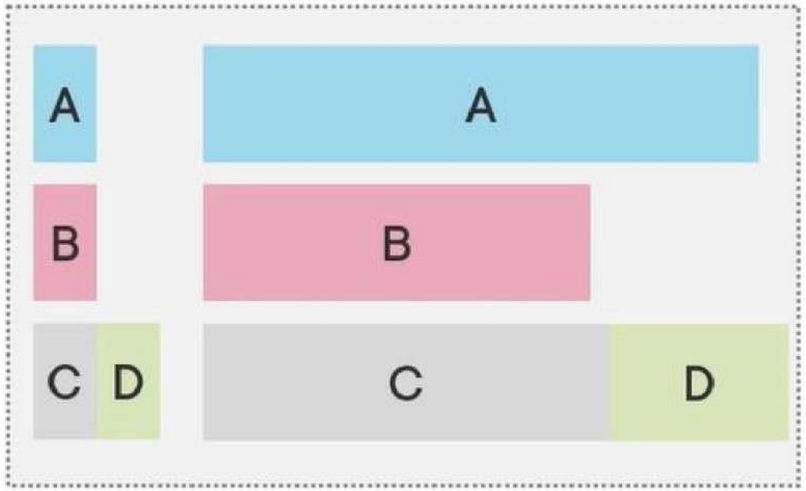
Shuffle Sort Merge Join



Shuffle Sort Merge Join



Shuffle Sort Merge Join



Involves shuffling & sorting of data for both datasets

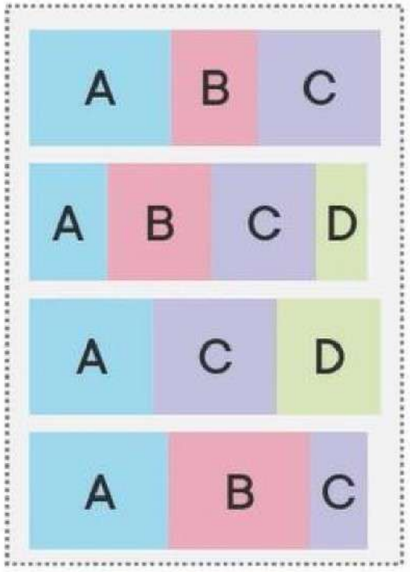
Great for joining two large datasets

But even if one dataset is small, shuffling still happens

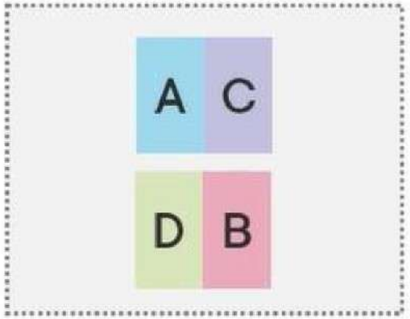
Expensive join strategy

If one dataset in a join operation is small, avoid shuffling using Broadcast Hash Join

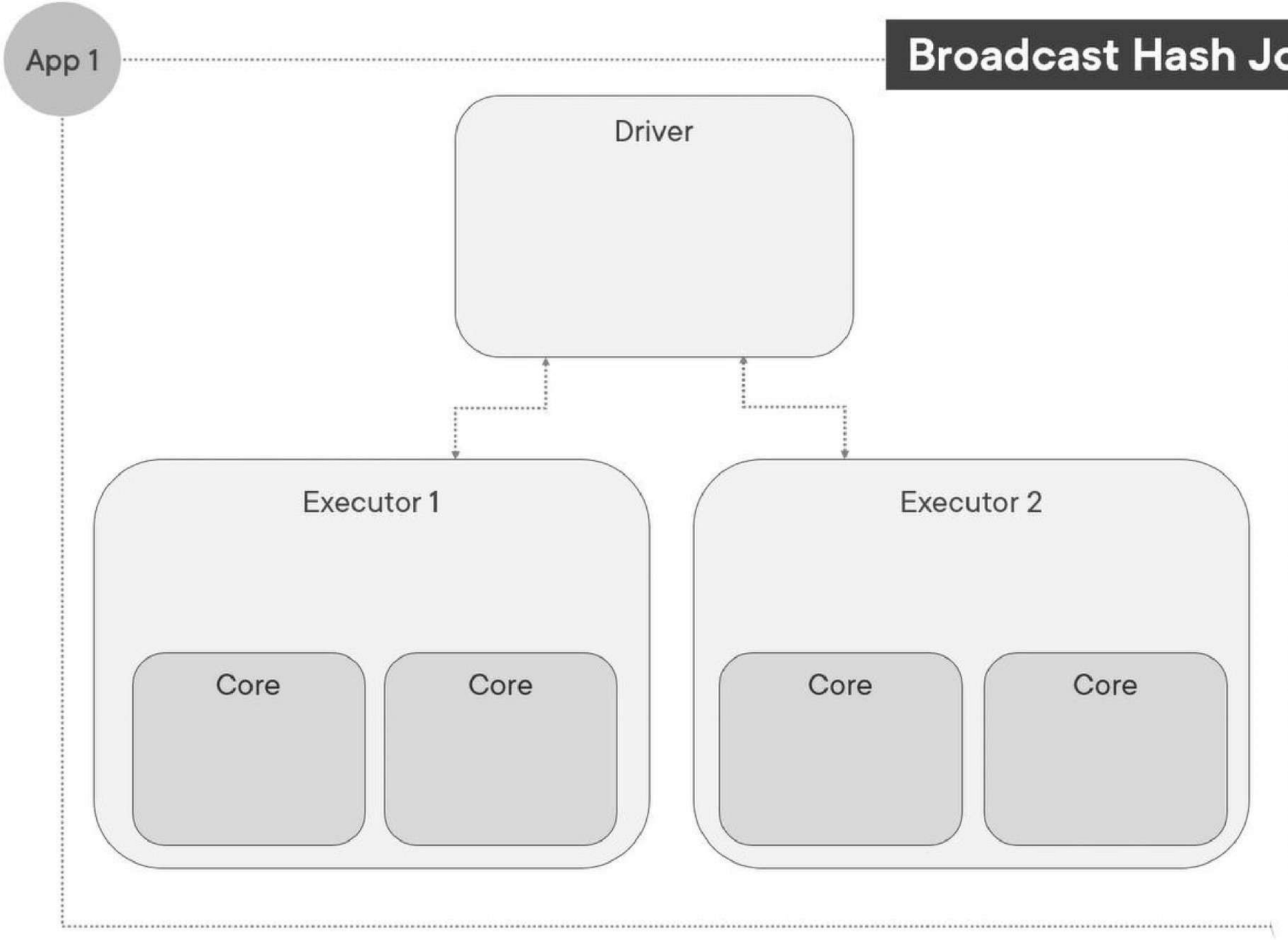
Broadcast Hash Join

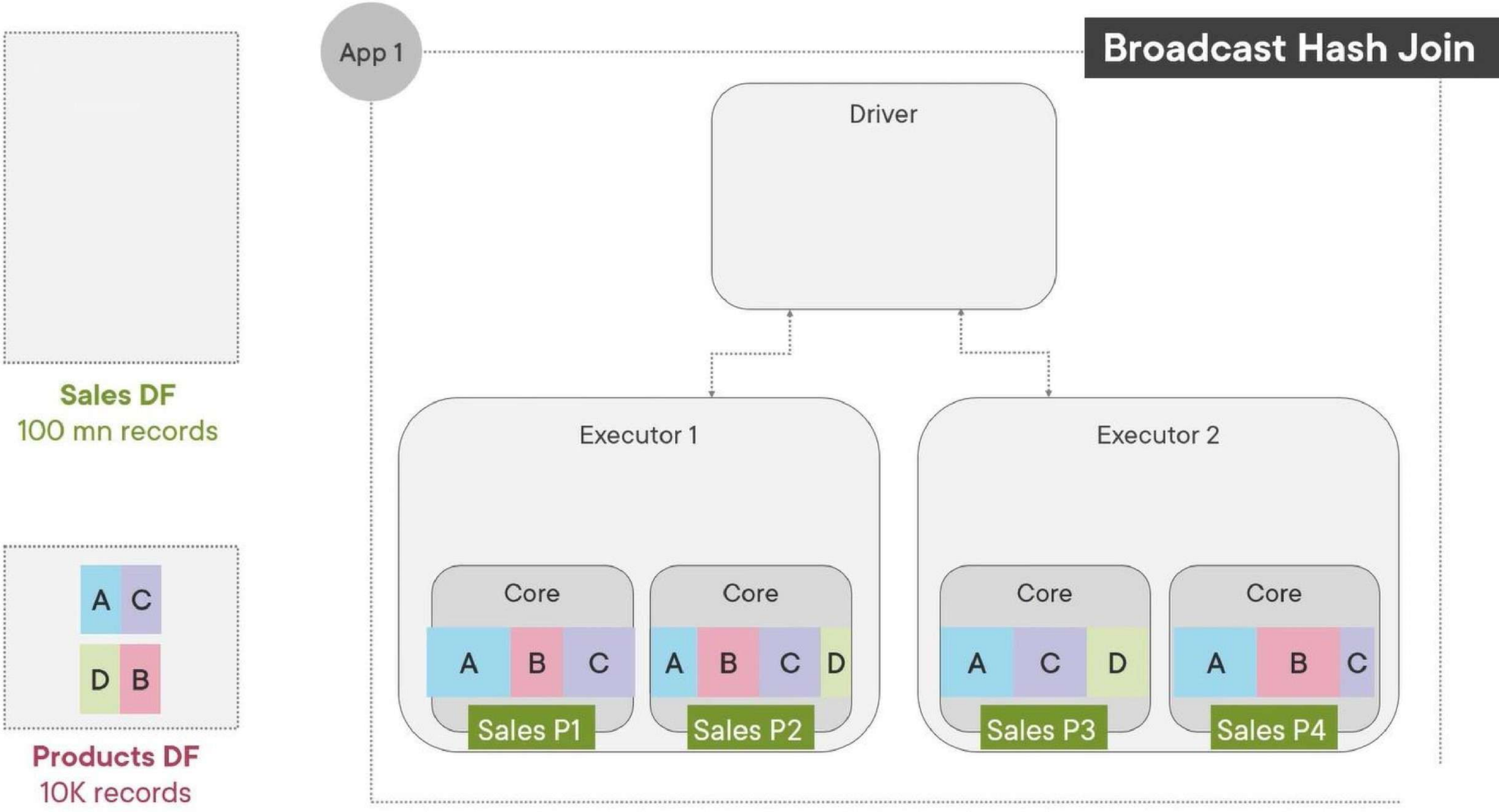


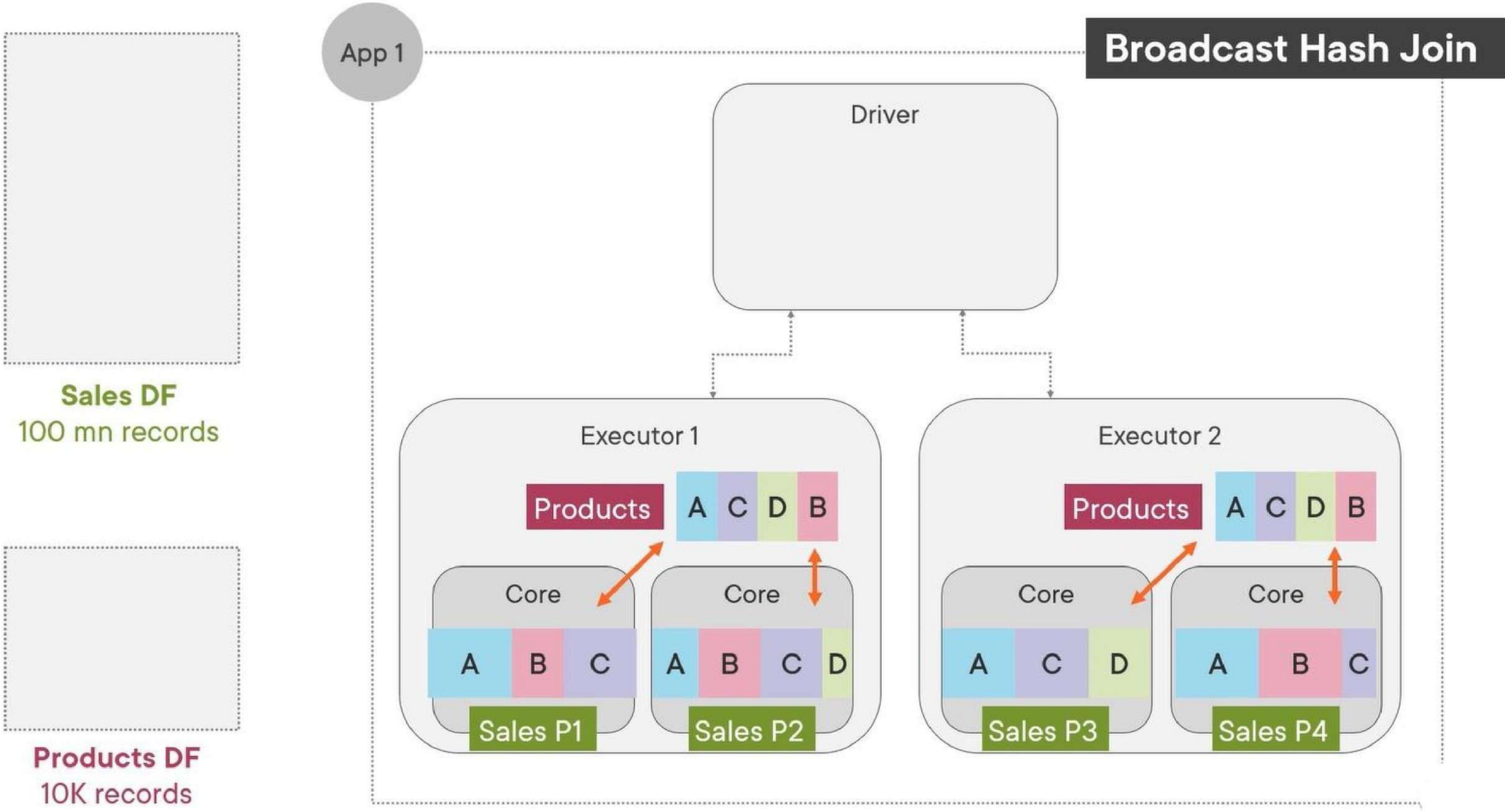
Sales DF
100 mn records



Products DF
10K records




Broadcast Hash Join
App 1
Driver
Sales DF
100 mn records
Executor 1
Executor 2
Core
A B C
Sales P1
Core
A B C D
Sales P2
Core
A C D
Sales P3
Core
A B C
Sales P4
A C
D B
Products DF
10K records


Broadcast Hash Join
App 1
Driver
Sales DF
100 mn records
Products DF
10K records
Executor 1
Products
A C D B
Core
A B C
Sales P1
Core
A B C D
Sales P2
Executor 2
Products
A C D B
Core
A C D
Sales P3
Core
A B C
Sales P4

Broadcast Hash Join

Useful when one of datasets is small

Copy of smaller dataset is sent to each executor once

No data shuffling required

Non-expensive & fastest join operation

Spark can perform auto broadcast or can be forced

Change Auto Broadcast setting

In real-life scenario, only disable if there is a specific use case, otherwise keep it enabled

```
# Check default value

spark.conf.get( "spark.sql.autoBroadcastJoinThreshold" )
```

```
'10485760b'
```

```
# Disable auto broadcast by setting value to -1

spark.conf.set( "spark.sql.autoBroadcastJoinThreshold", "-1" )
```

```python
# Create schema for Yellow Taxi Data

yellowTaxiSchema = (
                        StructType
                        ([
                            StructField("VendorId"                , IntegerType()   , True),
                            StructField("PickupTime"              , TimestampType() , True),
                            StructField("DropTime"                , TimestampType() , True),
                            StructField("PassengerCount"          , DoubleType()    , True),
                            StructField("TripDistance"            , DoubleType()    , True),
                            StructField("RateCodeId"              , DoubleType()    , True),
                            StructField("StoreAndFwdFlag"         , StringType()    , True),
                            StructField("PickupLocationId"        , IntegerType()   , True),
                            StructField("DropLocationId"          , IntegerType()   , True),
                            StructField("PaymentType"             , IntegerType()   , True),
                            StructField("FareAmount"              , DoubleType()    , True),
                            StructField("Extra"                   , DoubleType()    , True),
                            StructField("MtaTax"                  , DoubleType()    , True),
                            StructField("TipAmount"               , DoubleType()    , True),
                            StructField("TollsAmount"             , DoubleType()    , True),
                            StructField("ImprovementSurcharge"    , DoubleType()    , True),
                            StructField("TotalAmount"             , DoubleType()    , True),
                            StructField("CongestionSurcharge"     , DoubleType()    , True),
                            StructField("AirportFee"              , DoubleType()    , True)
                        ])
                    )
```

```
yellowTaxiDF = (
                    spark
                      .read
                      .option("header", "true")
                      .schema(yellowTaxiSchema)
                      .csv("E:\SparkData\Raw\YellowTaxis_202210.csv")
                  )

yellowTaxiDF.count()
```

```
1048575
```

```
taxiZonesSchema = "PickupLocationId INT, Borough STRING, Zone STRING, ServiceZone STRING"

taxiZonesDF = (
                  spark
                    .read
                    .schema(taxiZonesSchema)
                    .csv("E:\SparkData\Raw\TaxiZones.csv")
               )

taxiZonesDF.count()
```

```
265
```

Join large and small datasets

Auto-broadcast is already disabled

```
]: joinedDF = (
                yellowTaxiDF
                    .join
                    (
                        taxiZonesDF,

                        yellowTaxiDF.PickupLocationId == taxiZonesDF.PickupLocationId,

                        "inner"
                    )
            )

joinedDF.show()
```

Spark 3.3.2 | Jobs | Stages | Storage | Environment | Executors | SQL / DataFrame | SparkJoinsApp application U

Spark Jobs (?)

User: VICTUS
Total Uptime: 3.1 min
Scheduling Mode: FIFO
Completed Jobs: 3

▸ Event Timeline

▾ Completed Jobs (3)

Page: 1

1 Pages. Jump to 1 . Show 100 items in a page. Go

Job Id ▾	Description	Submitted	Duration	Stages: Succeeded/Total	Tasks (for all stages): Succeeded/Total
2	showString at NativeMethodAccessorImpl.java:0 showString at NativeMethodAccessorImpl.java:0	2023/05/20 00:59:25	6 s	3/3	6/6
1	count at NativeMethodAccessorImpl.java:0 count at NativeMethodAccessorImpl.java:0	2023/05/20 00:57:48	76 ms	2/2	2/2
0	count at NativeMethodAccessorImpl.java:0 count at NativeMethodAccessorImpl.java:0	2023/05/20 00:57:46	1 s	2/2	5/5

Spark 3.3.2 Jobs Stages Storage Environment Executors SQL / DataFrame SparkJoinsApp application UI

SQL / DataFrame

Completed Queries: 3

Completed Queries (3)

Page: 1 1 Pages. Jump to 1 . Show 100 items in a page. Go

ID	Description	Submitted	Duration	Job IDs
2	showString at NativeMethodAccessorImpl.java:0 +details	2023/05/20 00:59:25	6 s	[2]
1	count at NativeMethodAccessorImpl.java:0 +details	2023/05/20 00:57:48	0.3 s	[1]
0	count at NativeMethodAccessorImpl.java:0 +details	2023/05/20 00:57:45	3 s	[0]

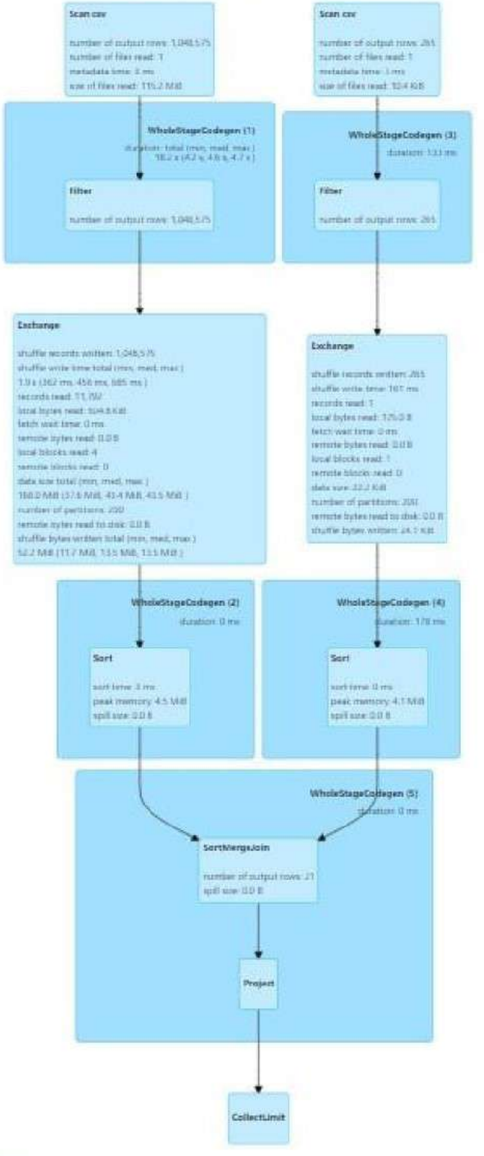
Page: 1 1 Pages. Jump to 1 . Show 100 items in a page. Go

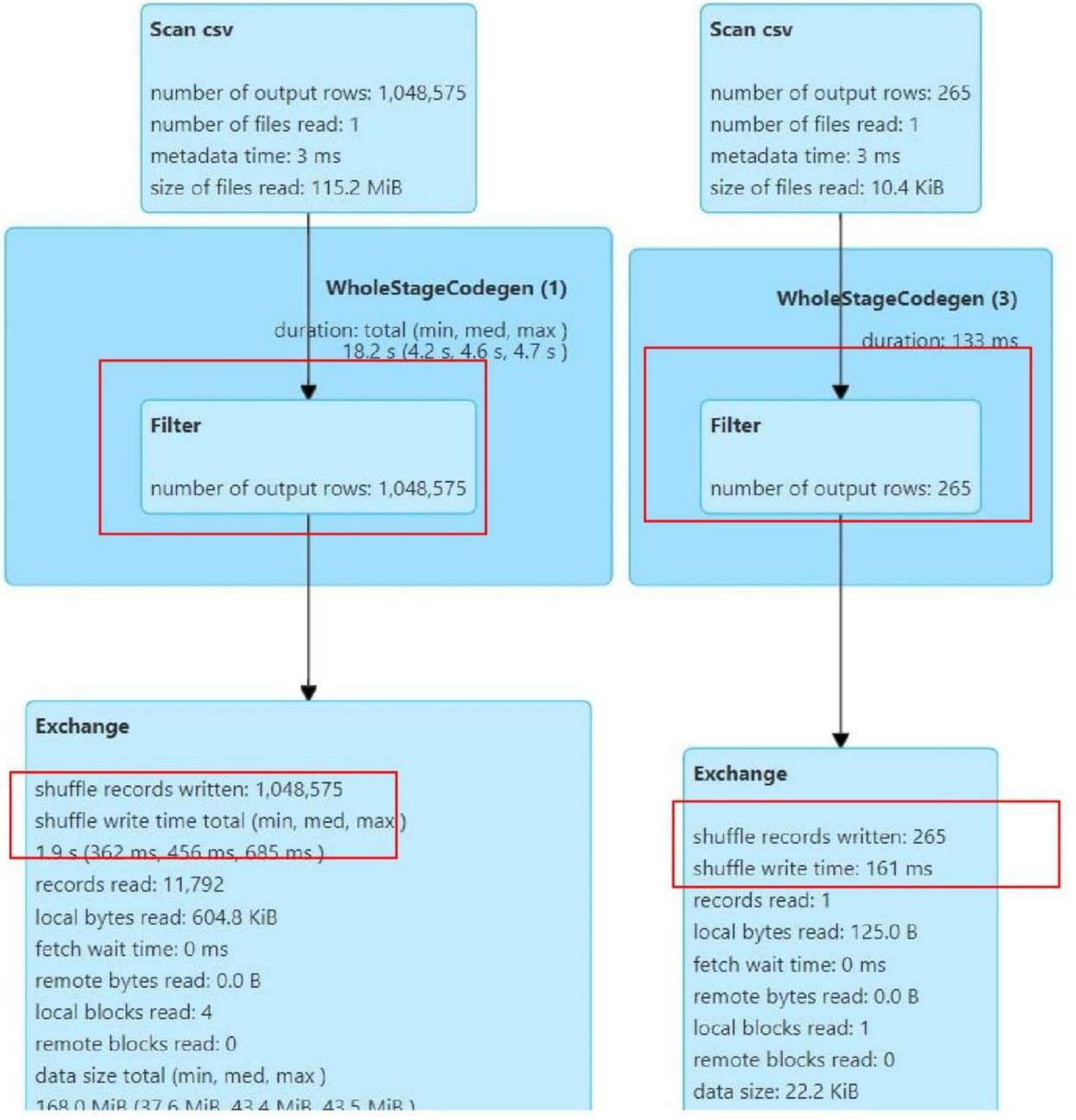
Spark 3.3.2 Jobs Stages Storage Environment Executors SQL / DataFrame

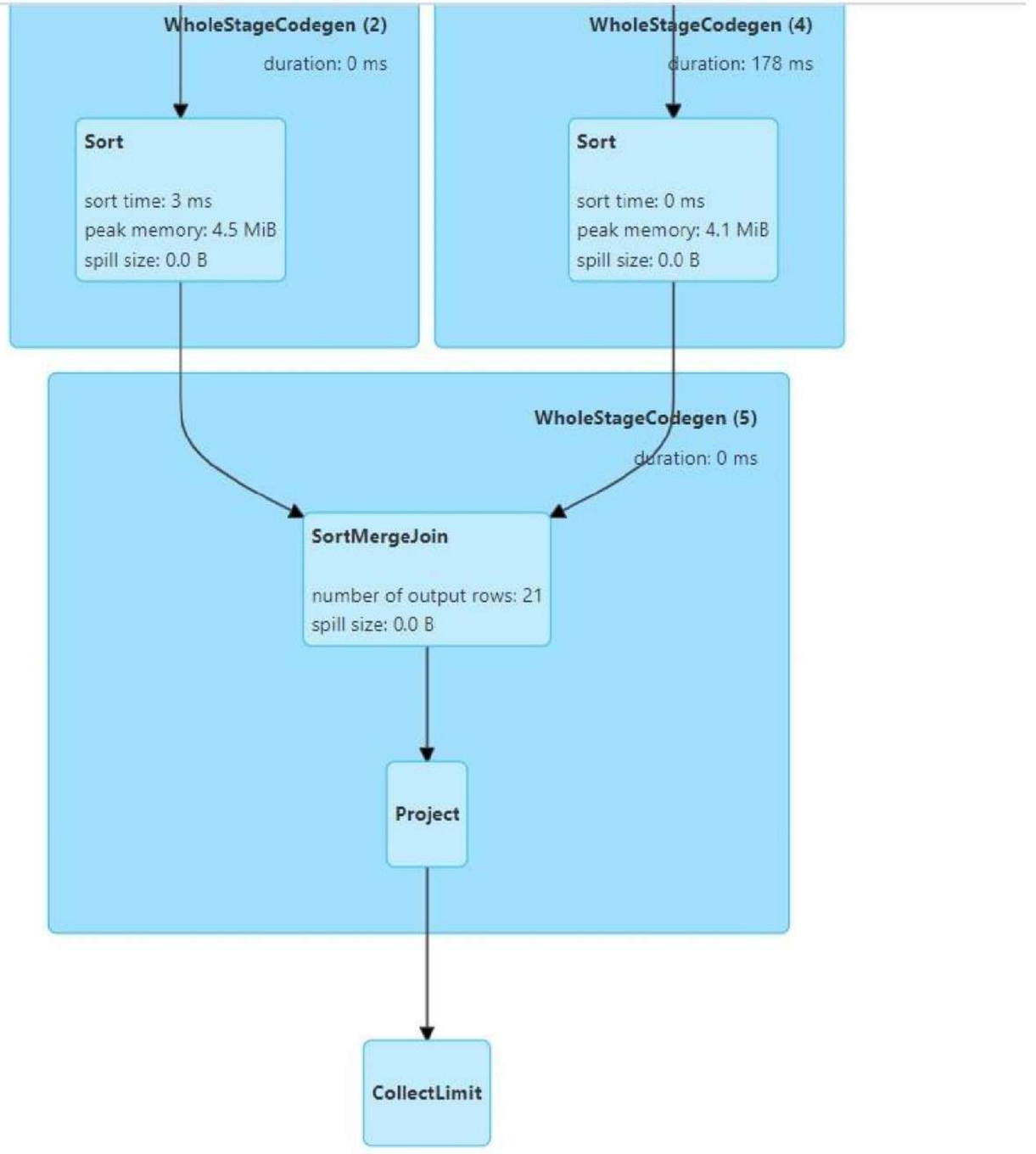
Details for Query 2

Submitted Time: 2023/05/20 08:59:25
Duration: 6 s
Succeeded Jobs: 2

☐ Show the Stage ID and Task ID that corresponds to the max metric




Scan csv
number of output rows: 1,048,575
number of files read: 1
metadata time: 3 ms
size of files read: 115.2 MiB
WholeStageCodegen (1)
duration: total (min, med, max)
18.2 s (4.2 s, 4.6 s, 4.7 s)
Filter
number of output rows: 1,048,575
Exchange
shuffle records written: 1,048,575
shuffle write time total (min, med, max)
1.9 s (362 ms, 456 ms, 685 ms)
records read: 11,792
local bytes read: 604.8 KiB
fetch wait time: 0 ms
remote bytes read: 0.0 B
local blocks read: 4
remote blocks read: 0
data size total (min, med, max)
168.0 MiB (37.6 MiB, 43.4 MiB, 43.5 MiB)
Scan csv
number of output rows: 265
number of files read: 1
metadata time: 3 ms
size of files read: 10.4 KiB
WholeStageCodegen (3)
duration: 133 ms
Filter
number of output rows: 265
Exchange
shuffle records written: 265
shuffle write time: 161 ms
records read: 1
local bytes read: 125.0 B
fetch wait time: 0 ms
remote bytes read: 0.0 B
local blocks read: 1
remote blocks read: 0
data size: 22.2 KiB


WholeStageCodegen (2)
duration: 0 ms
Sort
sort time: 3 ms
peak memory: 4.5 MiB
spill size: 0.0 B
WholeStageCodegen (4)
duration: 178 ms
Sort
sort time: 0 ms
peak memory: 4.1 MiB
spill size: 0.0 B
WholeStageCodegen (5)
duration: 0 ms
SortMergeJoin
number of output rows: 21
spill size: 0.0 B
Project
CollectLimit

Don’t run this

Option 1 - Broadcast Hash Join: Set join threshold

```
]: # Will not run. Will manually perform broadcast

spark.conf.set("spark.sql.autoBroadcastJoinThreshold", "10485760b")
```

Option 2 - Broadcast Hash Join: Manually broadcast small dataset

In case below, Taxi Zones is smaller dataset, so manually broadcast it.

Perform broadcast manually if "smaller" dataset size is bigger than auto threshold size.

```
# Manually broadcast Taxi Zones

joinedDF = (
                yellowTaxiDF
                    .join
                    (
                        broadcast(taxiZonesDF),

                        yellowTaxiDF.PickupLocationId == taxiZonesDF.PickupLocationId,
                        "inner"
                    )
            )

joinedDF.show()
```

```
+--------+-------------------+-------------------+--------------+------------+----------+---------------+--
---------+-----------+----------+-----+-----+---------+-----------+--------------------+-----------+-----
----+----------------+---------+--------------------+-----------+
|VendorId|         PickupTime|          DropTime|PassengerCount|TripDistance|RateCodeId|StoreAndFwdFlag|Pi
```

Spark 3.3.2 | Jobs | Stages | Storage | Environment | Executors | SQL / DataFrame | SparkJoinsApp application UI

Spark Jobs (?)

User: VICTUS
Total Uptime: 6.5 min
Scheduling Mode: FIFO
Completed Jobs: 5

▸ Event Timeline

▾ Completed Jobs (5)

Page: 1

1 Pages. Jump to 1 . Show 100 items in a page. Go

Job Id (Job Group) ▾	Description	Submitted	Duration	Stages: Succeeded/Total	Tasks (for all stages): Succeeded/Total
4	showString at NativeMethodAccessorImpl.java:0 showString at NativeMethodAccessorImpl.java:0	2023/05/20 01:02:57	0.2 s	1/1	1/1
3 (3cab88f5-5c10-4b45-b07d-2448238fd197)	broadcast exchange (runId 3cab88f5-5c10-4b45-b07d-2448238fd197) $anonfun$withThreadLocalCaptured$1 at FutureTask.java:264	2023/05/20 01:02:56	31 ms	1/1	1/1
2	showString at NativeMethodAccessorImpl.java:0 showString at NativeMethodAccessorImpl.java:0	2023/05/20 00:59:25	6 s	3/3	6/6
1	count at NativeMethodAccessorImpl.java:0 count at NativeMethodAccessorImpl.java:0	2023/05/20 00:57:48	76 ms	2/2	2/2
0	count at NativeMethodAccessorImpl.java:0 count at NativeMethodAccessorImpl.java:0	2023/05/20 00:57:46	1 s	2/2	5/5

SQL / DataFrame

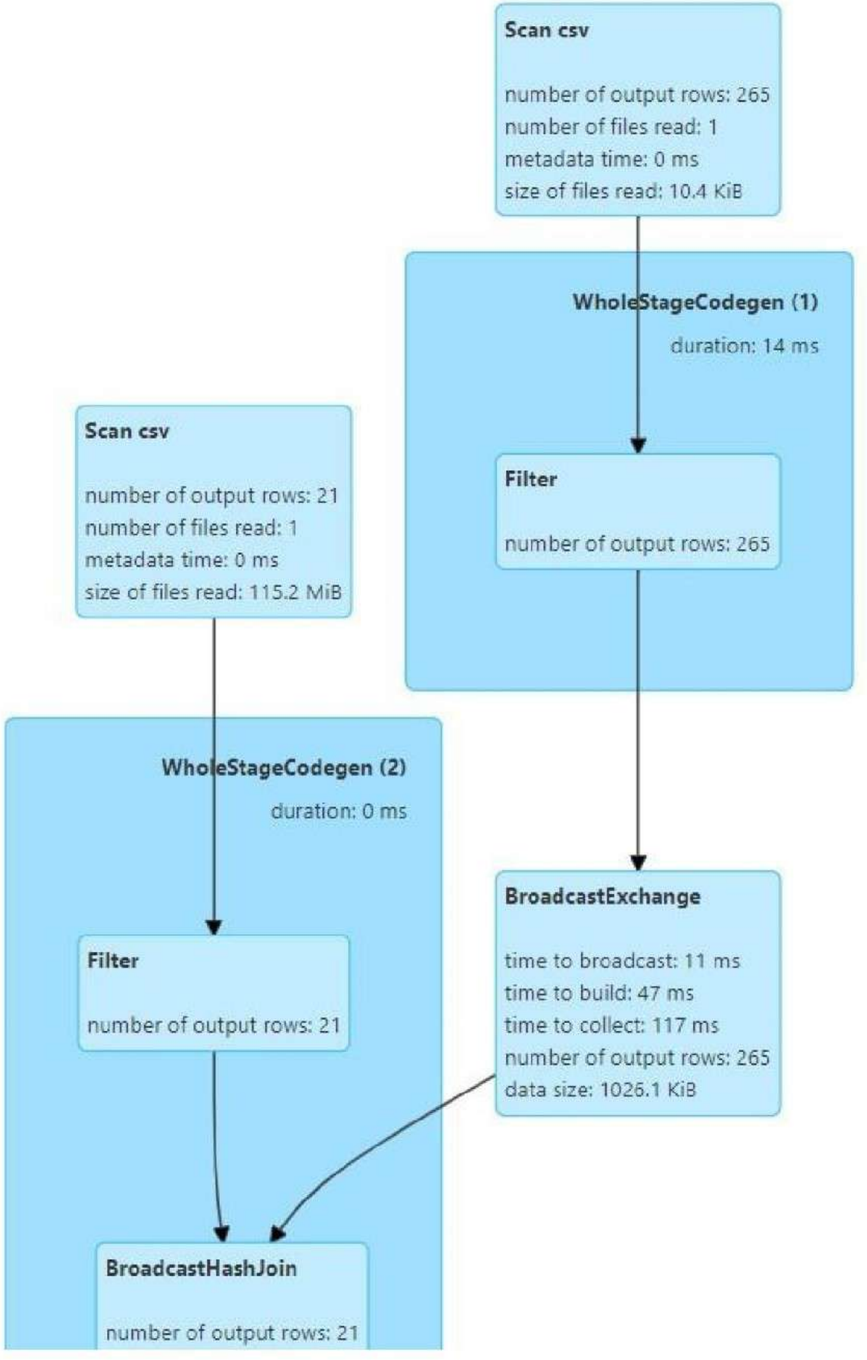
Completed Queries: 4

Completed Queries (4)

Page: 1 1 Pages. Jump to 1 . Show 100 items in a page. Go

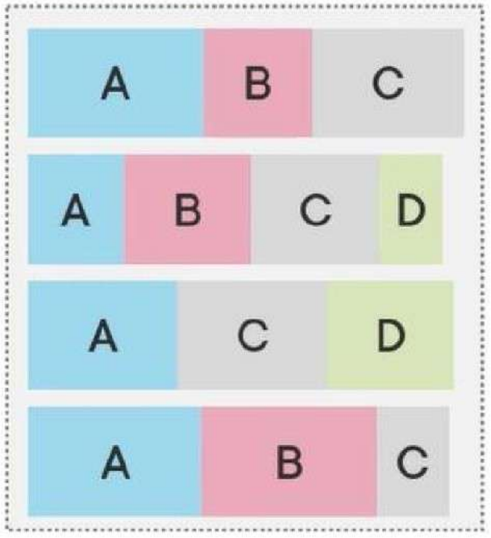
ID	Description	Submitted	Duration	Job IDs
3	showString at NativeMethodAccessorImpl.java:0 +details	2023/05/20 01:02:56	0.9 s	[3][4]
2	showString at NativeMethodAccessorImpl.java:0 +details	2023/05/20 00:59:25	6 s	[2]
1	count at NativeMethodAccessorImpl.java:0 +details	2023/05/20 00:57:48	0.3 s	[1]
0	count at NativeMethodAccessorImpl.java:0 +details	2023/05/20 00:57:45	3 s	[0]

Page: 1 1 Pages. Jump to 1 . Show 100 items in a page. Go

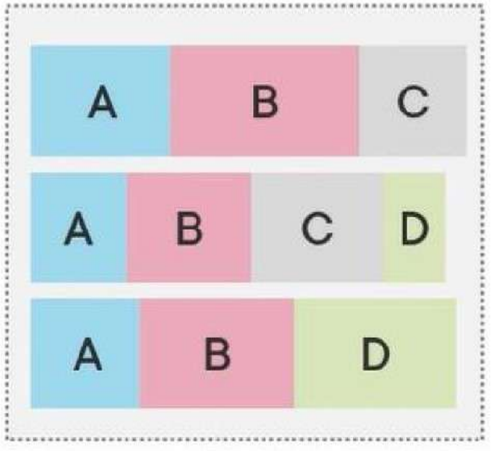

Scan csv
number of output rows: 265
number of files read: 1
metadata time: 0 ms
size of files read: 10.4 KiB
WholeStageCodegen (1)
duration: 14 ms
Filter
number of output rows: 265
Scan csv
number of output rows: 21
number of files read: 1
metadata time: 0 ms
size of files read: 115.2 MiB
WholeStageCodegen (2)
duration: 0 ms
Filter
number of output rows: 21
BroadcastExchange
time to broadcast: 11 ms
time to build: 47 ms
time to collect: 117 ms
number of output rows: 265
data size: 1026.1 KiB
BroadcastHashJoin
number of output rows: 21

Optimizing Shuffle Sort Join with Bucketing

Shuffle Sort Merge Join



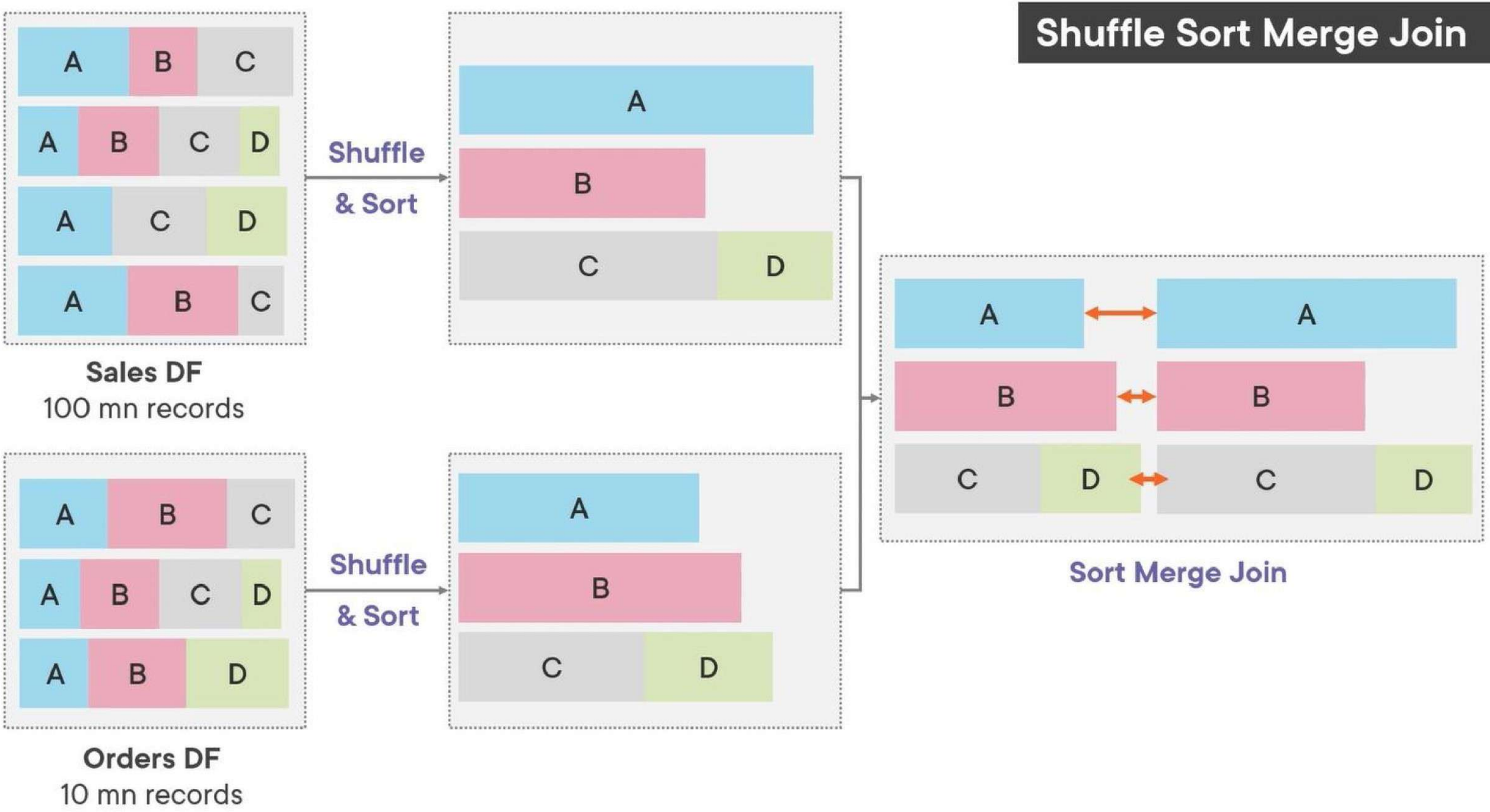
Sales DF
100 mn records



Orders DF
10 mn records

Shuffle Sort Merge Join

Shuffle Sort Merge Join



Bucketing is an optimization technique to avoid shuffles during joins

What is Bucketing?

Id	City	Amount
1	Seattle	600
2	London	300
3	Seattle	700
4	Delhi	400
5	Paris	900
6	Seattle	900
7	Delhi	200

DataFrame

Hash function

What is Bucketing?

Id	City	Amount
1	Seattle	600
2	London	300
3	Seattle	700
4	Delhi	400
5	Paris	900
6	Seattle	900
7	Delhi	200

DataFrame

Hash function

1	Seattle	600
3	Seattle	700

2	London	300

What is Bucketing?

Id	City	Amount
1	Seattle	600
2	London	300
3	Seattle	700
4	Delhi	400
5	Paris	900
6	Seattle	900
7	Delhi	200

DataFrame

Hash function

1	Seattle	600
3	Seattle	700
6	Seattle	900

2	London	300
5	Paris	900

4	Delhi	400
7	Delhi	200

Bucketed data is written to disk

Bucketing

Pre-shuffle large datasets & write it to disk in buckets

Use bucketed data in subsequent queries

Use when dataset is frequently used in joins, aggregations or window operations

Conditions for joining bucketed datasets

- Bucketed datasets must be stored as tables
- Both datasets must be bucketed on join columns
- Number of buckets must be the same

Optimize shuffle sort merge join performance with Bucketing

Create two views on a large DataFrame

Note using same DataFrame for joining is for demo purpose only.
In real-life scenario, you would be (likely) joining two different datasets.

```
yellowTaxiDF.createOrReplaceTempView("YellowTaxis1_Unbucketed")

yellowTaxiDF.createOrReplaceTempView("YellowTaxis2_Unbucketed")
```

Join two unbucketed datasets

This should perform Shuffle Sort Merge Join

```
spark.sql("""

SELECT *

FROM YellowTaxis1_Unbucketed b1

    JOIN YellowTaxis2_Unbucketed b2 ON b1.PickupLocationId = b2.DropLocationId

""").show()
```

Write datasets as tables with bucketing

1. Bucketing should be on columns that would be used in joins
2. Both datasets should be stored with same number of buckets

```
(
    yellowTaxiDF
            .write

            .bucketBy(4, "PickupLocationId")

            #.sortBy("PickupLocationId")      Optionally sort the data

            .option("header", "true")
            .option("dateFormat", "yyyy-MM-dd HH:mm:ss.S")
            .mode("overwrite")
            .format("csv")

            .saveAsTable("YellowTaxis_PickupBucket")
)
```

```
(
    yellowTaxiDF
            .write

            .bucketBy(4, "DropLocationId")

            #.sortBy("DropLocationId")        Optionally sort the data

            .option("header", "true")
            .option("dateFormat", "yyyy-MM-dd HH:mm:ss.S")
            .mode("overwrite")
            .format("csv")

            .saveAsTable("YellowTaxis_DropBucket")
)
```

Join two bucketed tables

1. Bucketing is on columns being used in joins
2. Both tables have same number of buckets

```
spark.sql("""

SELECT *

FROM YellowTaxis_PickupBucket b1

    JOIN YellowTaxis_DropBucket b2 ON b1.PickupLocationId = b2.DropLocationId

""").show()
```

Dynamic Resource Allocation

Resource Scheduling for a Job

Scheduling Across Applications

Scheduling Within an Application

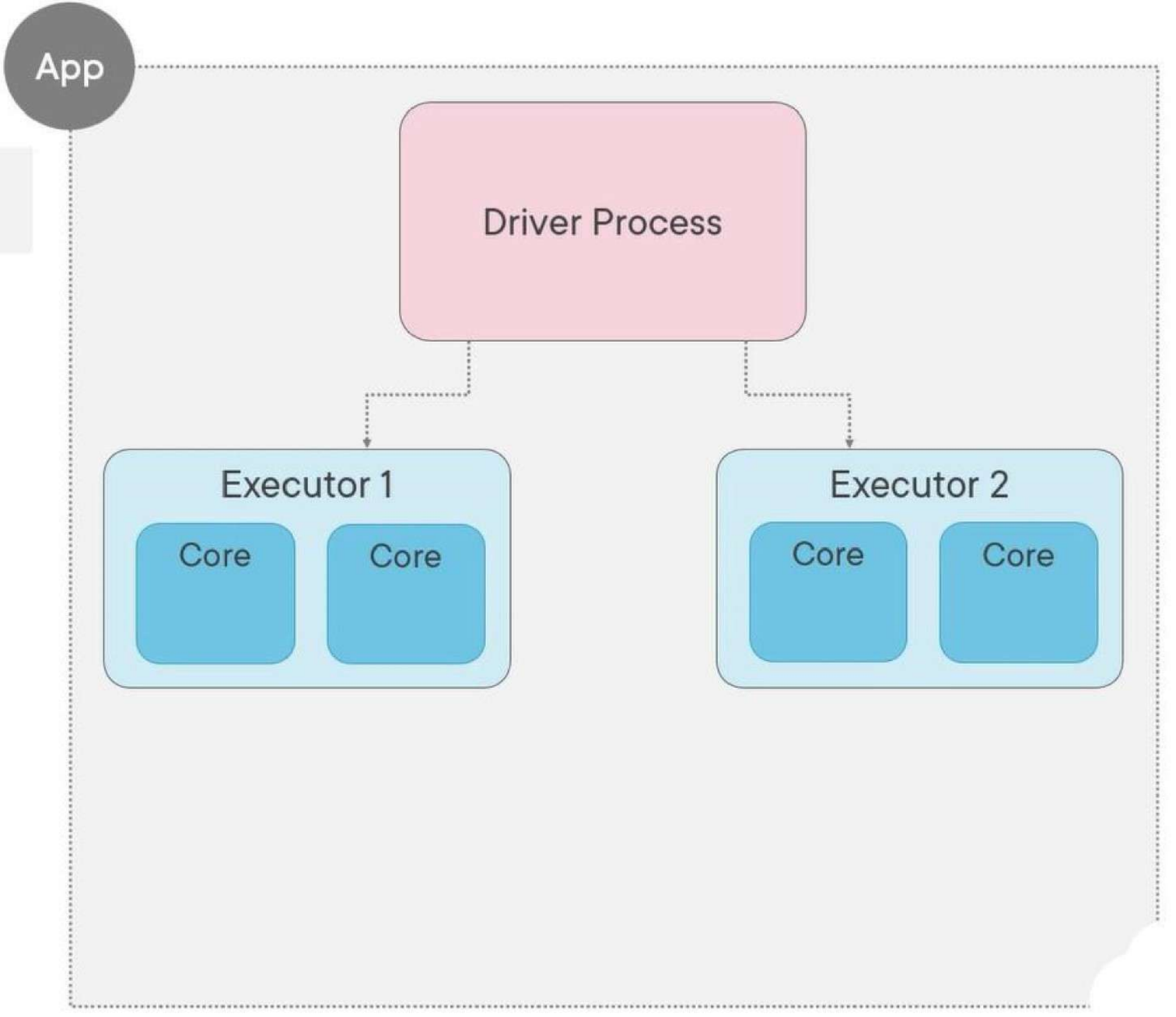
Resource Scheduling Across Applications

Static Resource Allocation

Dynamic Resource Allocation

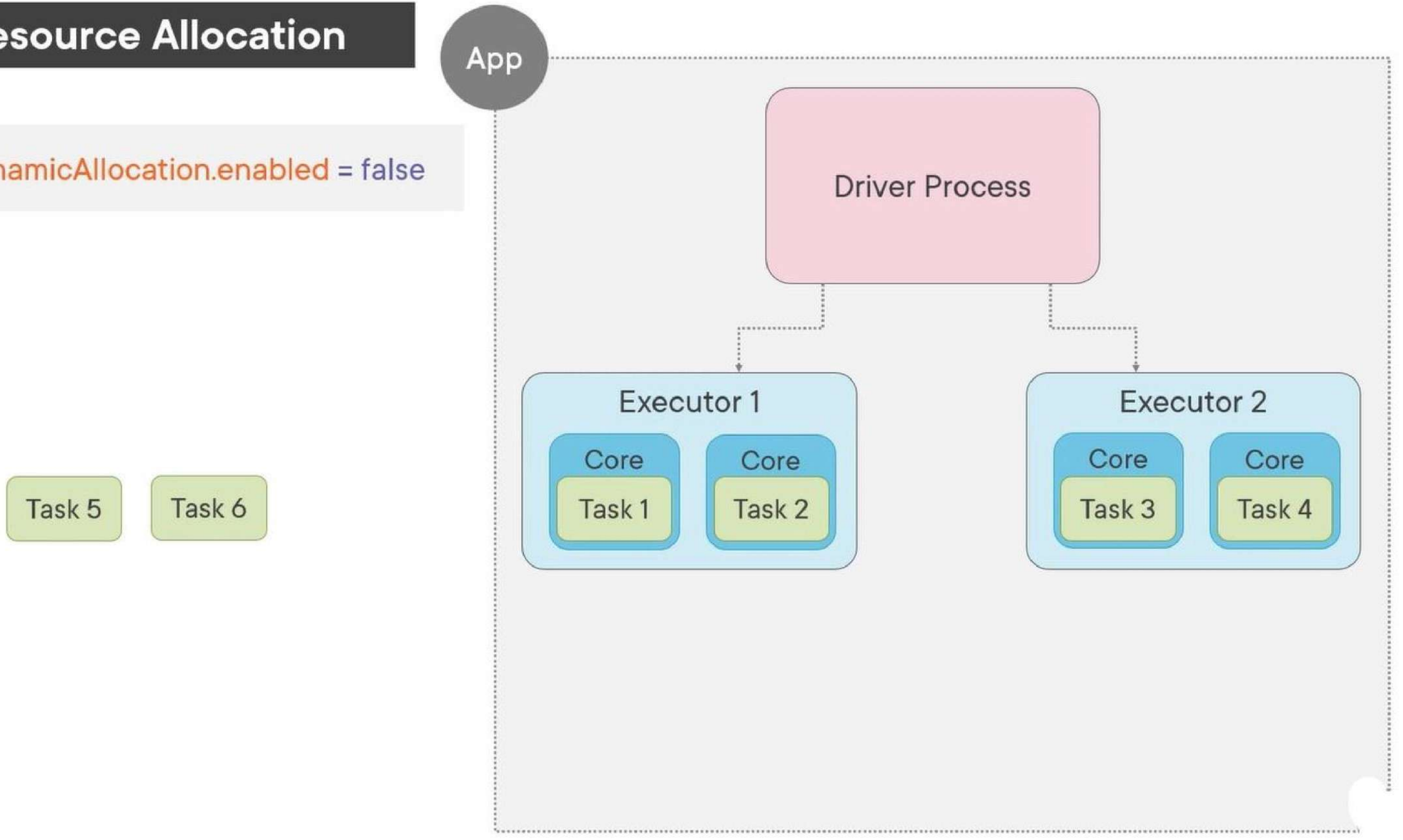
Static Resource Allocation

spark.dynamicAllocation.enabled = false



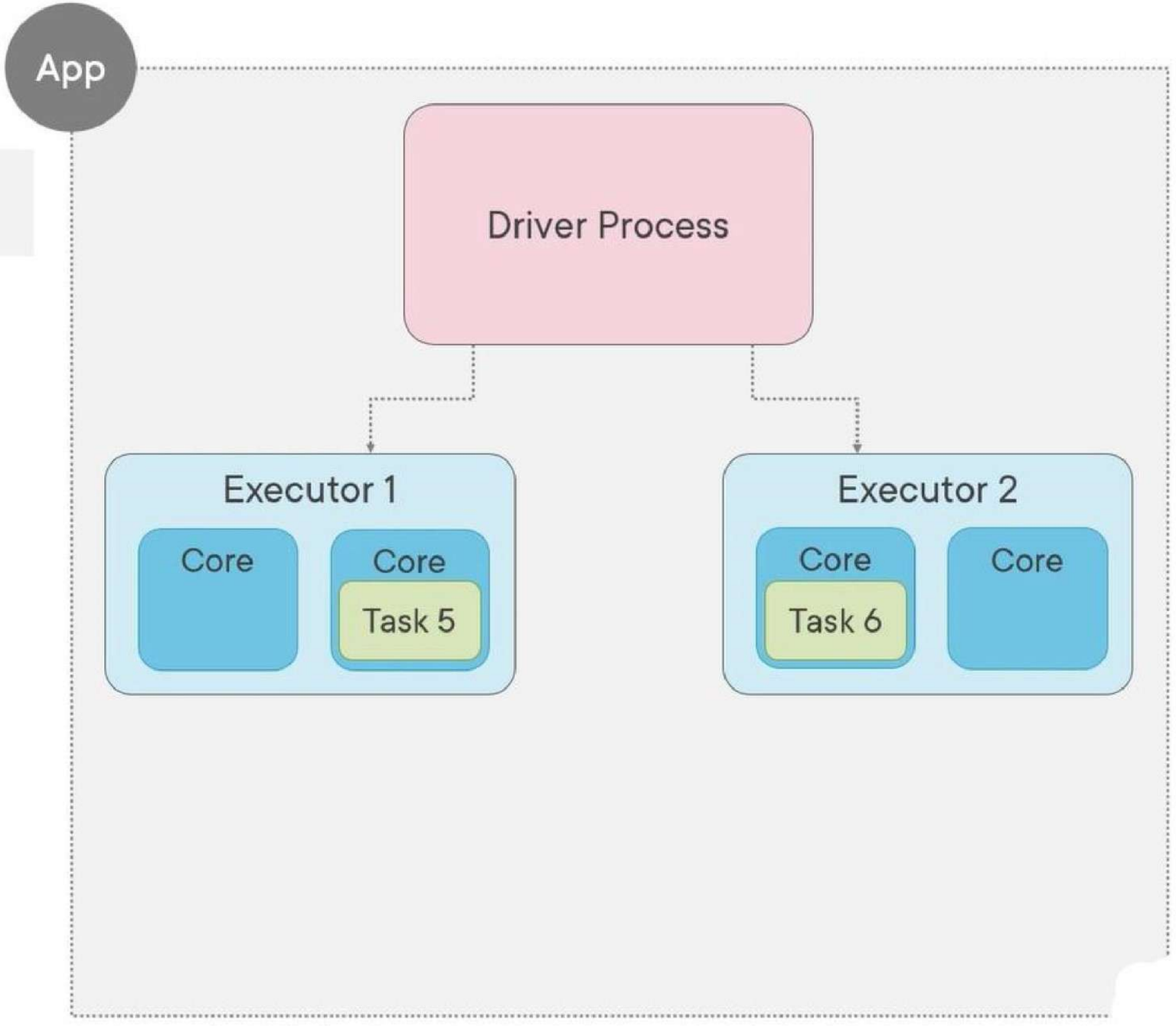
Static Resource Allocation

spark.dynamicAllocation.enabled = false



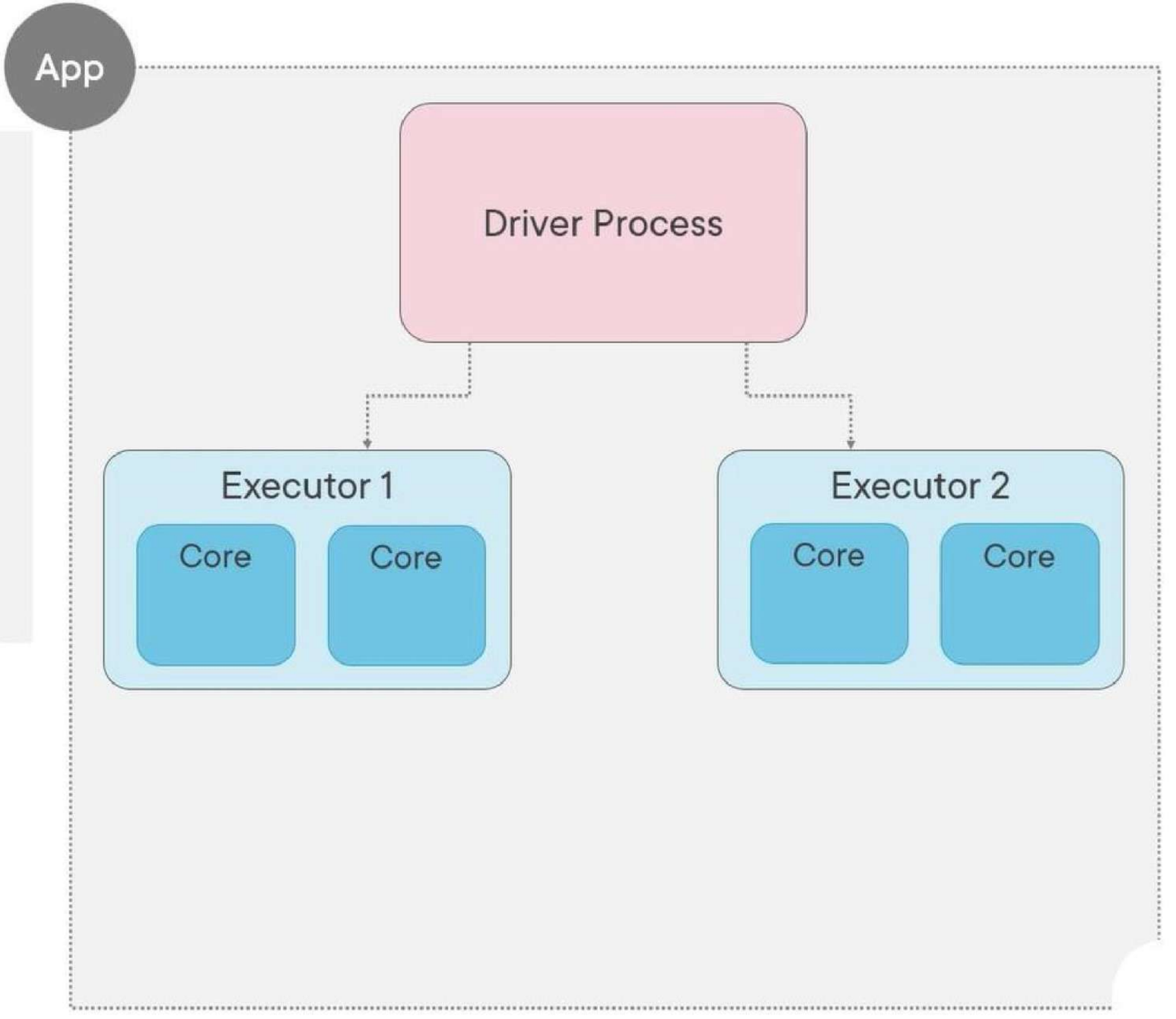
Static Resource Allocation

```
spark.dynamicAllocation.enabled = false
```



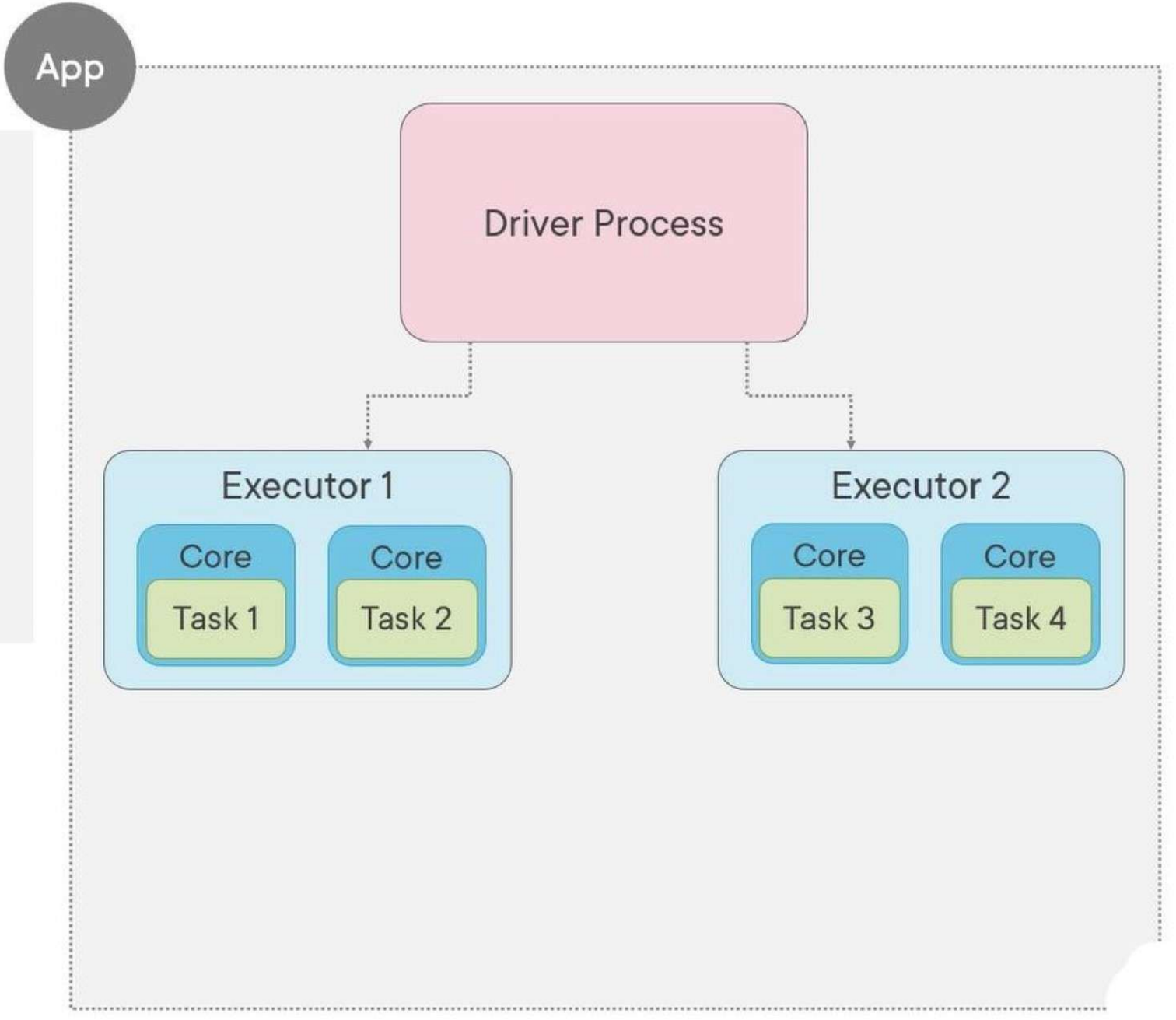
Dynamic Resource Allocation

```
spark.dynamicAllocation :
    .enabled = true
    .shuffleTracking. enabled = true

    .minExecutors = 0
    .maxExecutors = 5
```



Dynamic Resource Allocation

```
spark.dynamicAllocation :
    .enabled = true
    .shuffleTracking. enabled = true

    .minExecutors = 0
    .maxExecutors = 5

    .schedulerBacklogTimeout = 1s
```



Task 5

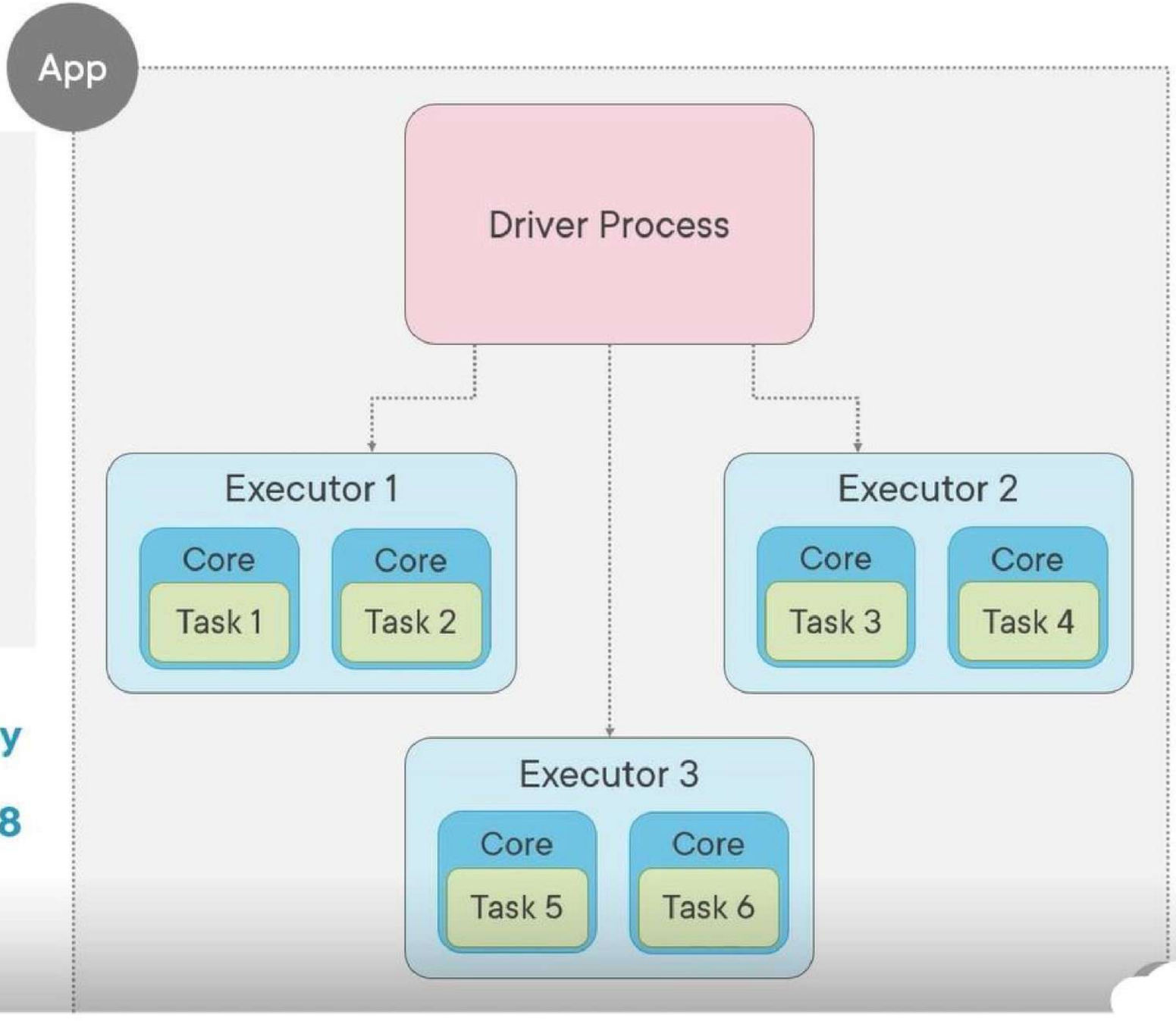
Task 6

Dynamic Resource Allocation

```
spark.dynamicAllocation :
    .enabled = true
    .shuffleTracking. enabled = true

    .minExecutors = 0
    .maxExecutors = 5

    .schedulerBacklogTimeout = 1s
```

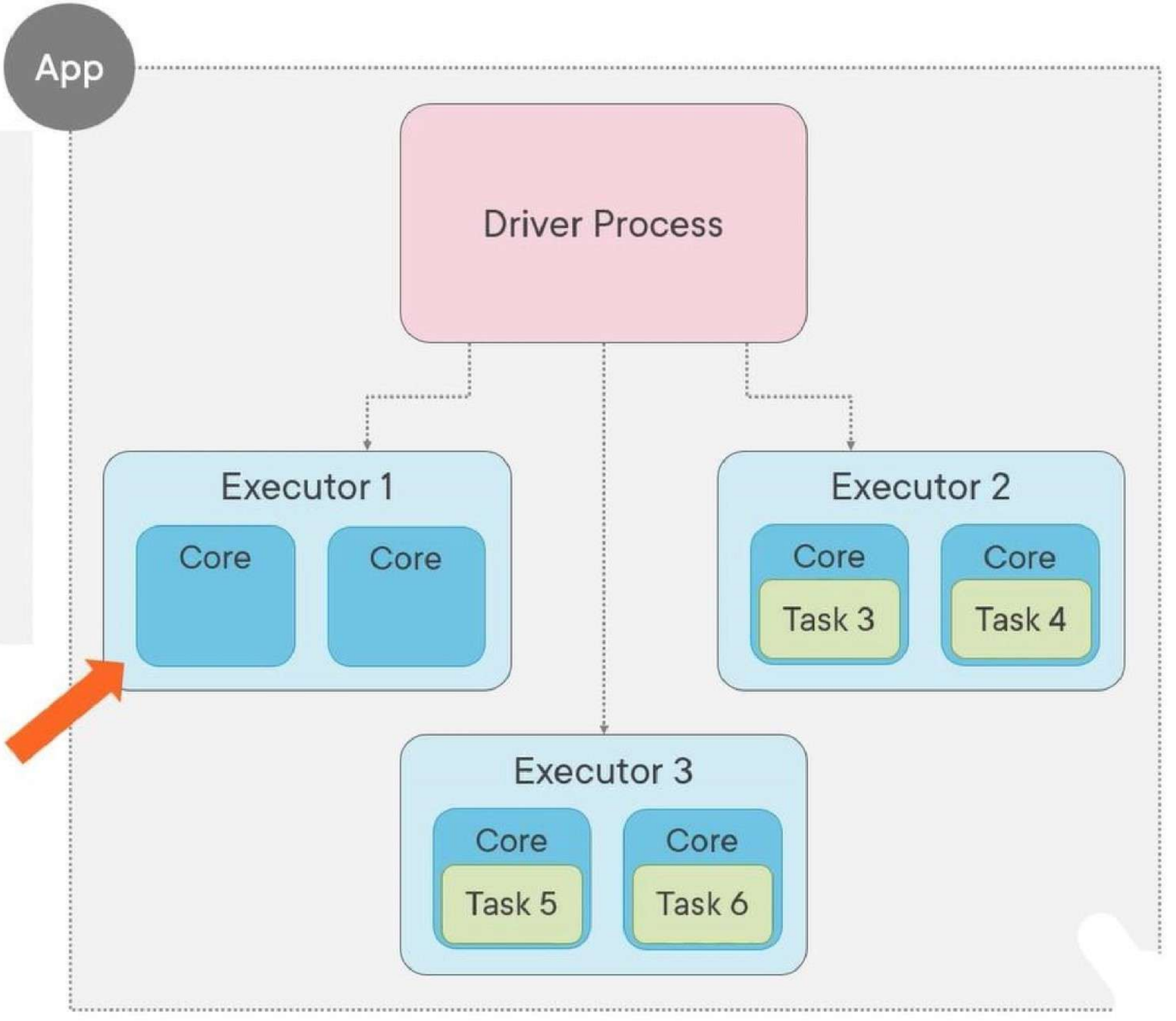
Executors will increase exponentially

Starting with 1 executor – then 2, 4, 8



Dynamic Resource Allocation

```
spark.dynamicAllocation :
    .enabled = true
    .shuffleTracking. enabled = true

    .minExecutors = 0
    .maxExecutors = 5

    .schedulerBacklogTimeout = 1s
    .executorIdleTimeout = 60s
```

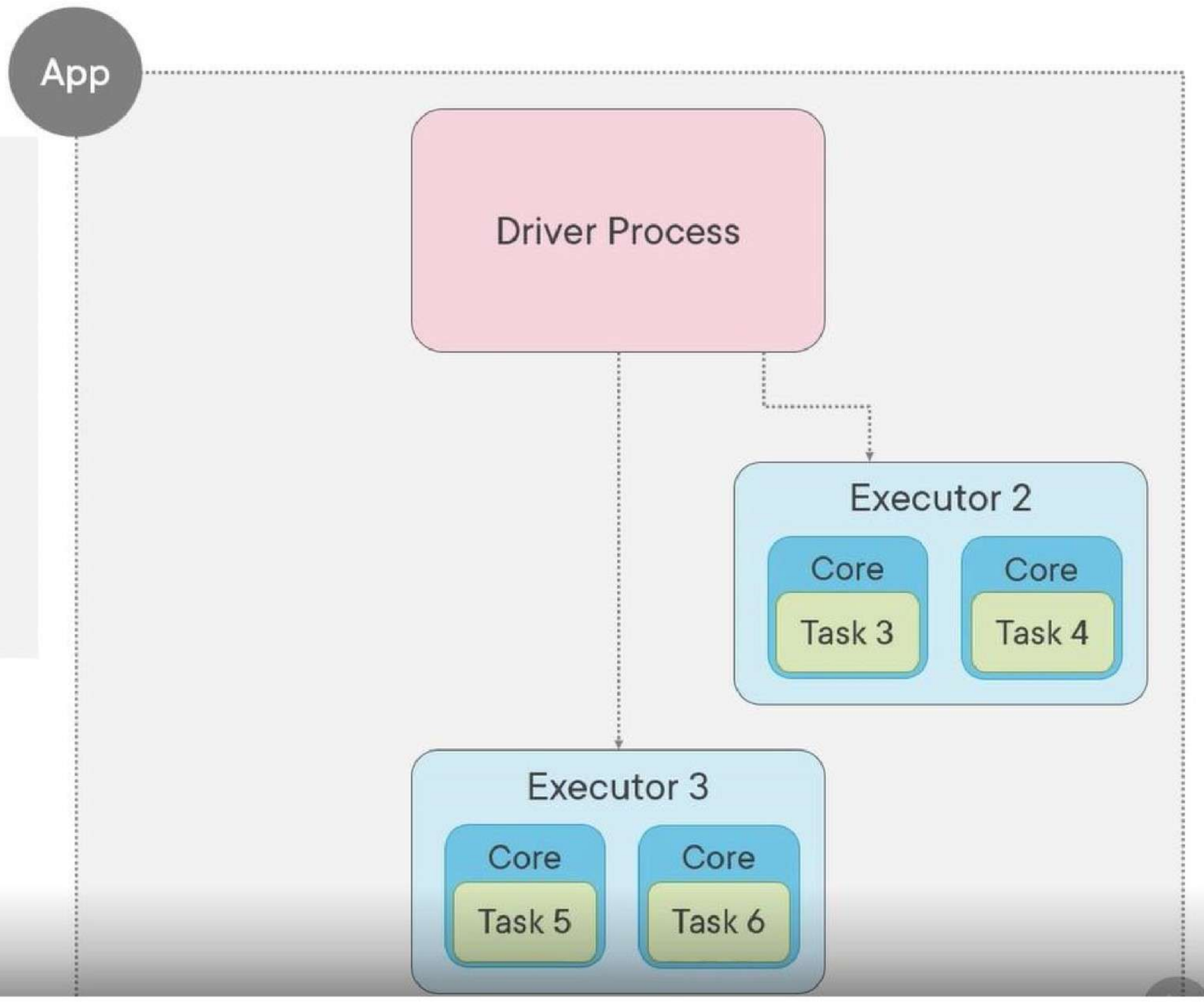


Dynamic Resource Allocation

```
spark.dynamicAllocation :
    .enabled = true
    .shuffleTracking. enabled = true

    .minExecutors = 0
    .maxExecutors = 5

    .schedulerBacklogTimeout = 1s
    .executorIdleTimeout = 60s
```

Any cached data on executor will also be removed

No shuffle data is lost



Request more executors
to complete pending tasks!

Remove executors
when they are idle!

Use Cases

Running ad-hoc interactive applications

Long running ETL jobs to process multiple entities

Jobs with large shuffle operations

Start the multinode cluster

```
=========================================================
Steps to create multi-node cluster on local machine
=========================================================

A. In Command Prompt 1 - Spark Master

A.1. Create Spark master
      => spark-class org.apache.spark.deploy.master.Master --host <machine name or localhost>

      - Example => spark-class org.apache.spark.deploy.master.Master --host localhost

A.2. Copy Master URL and Spark UI URL
      - Typically master URL is at port 7077
      - Typically Spark UI is at port 4040
      - If any of these ports are in use, a different port might be used

-------------------------------------------------------------------------------------------------

B. In Command Prompt 2 - Worker Node 1

B.1. Create worker node 1
      => spark-class org.apache.spark.deploy.worker.Worker <master URL> --host <machine name or localhost>

      - Example => spark-class org.apache.spark.deploy.worker.Worker spark://localhost:7077 --host localhost

-------------------------------------------------------------------------------------------------

C. In Command Prompt 3 - Worker Node 2

C.1. Create worker node 2
      => spark-class org.apache.spark.deploy.worker.Worker <master URL> --host <machine name or localhost>

      - Example => spark-class org.apache.spark.deploy.worker.Worker spark://localhost:7077 --host localhost
```

Go to Spark UI for Master

```
Command Prompt - spark-cla   Command Prompt - spark-clas   Command Prompt - spark-clas

Microsoft Windows [Version 10.0.22621.1702]
(c) Microsoft Corporation. All rights reserved.

C:\Users\VICTUS>spark-class org.apache.spark.deploy.master.Master --host localhost
Using Spark's default log4j profile: org/apache/spark/log4j2-defaults.properties
23/05/20 01:24:46 INFO Master: Started daemon with process name: 27128@LAPTOP-2ERRB9V0
23/05/20 01:24:46 INFO SecurityManager: Changing view acls to: VICTUS
23/05/20 01:24:46 INFO SecurityManager: Changing modify acls to: VICTUS
23/05/20 01:24:46 INFO SecurityManager: Changing view acls groups to:
23/05/20 01:24:46 INFO SecurityManager: Changing modify acls groups to:
23/05/20 01:24:46 INFO SecurityManager: SecurityManager: authentication disabled; ui acls disabled; users  with view permissions: Set
(VICTUS); groups with view permissions: Set(); users  with modify permissions: Set(VICTUS); groups with modify permissions: Set()
23/05/20 01:24:47 INFO Utils: Successfully started service 'sparkMaster' on port 7077.
23/05/20 01:24:47 INFO Master: Starting Spark master at spark://localhost:7077
23/05/20 01:24:47 INFO Master: Running Spark version 3.3.2
23/05/20 01:24:47 INFO Utils: Successfully started service 'MasterUI' on port 8080.
23/05/20 01:24:47 INFO MasterWebUI: Bound MasterWebUI to 0.0.0.0, and started at http://LAPTOP-2ERRB9V0:8080
23/05/20 01:24:47 INFO Master: I have been elected leader! New state: ALIVE
23/05/20 01:25:14 INFO Master: Registering worker localhost:63833 with 16 cores, 62.7 GiB RAM
23/05/20 01:25:18 INFO Master: Registering worker localhost:63891 with 16 cores, 62.7 GiB RAM
```

Spark Master at spark://localhost:7077

URL: spark://localhost:7077
Alive Workers: 2
Cores in use: 32 Total, 0 Used
Memory in use: 125.3 GiB Total, 0.0 B Used
Resources in use:
Applications: 0 Running, 0 Completed
Drivers: 0 Running, 0 Completed
Status: ALIVE

Workers (2)

Worker Id	Address	State	Cores	Memory	Resources
worker-20230520012513-localhost-63833	localhost:63833	ALIVE	16 (0 Used)	62.7 GiB (0.0 B Used)	
worker-20230520012518-localhost-63891	localhost:63891	ALIVE	16 (0 Used)	62.7 GiB (0.0 B Used)	

Running Applications (0)

Application ID	Name	Cores	Memory per Executor	Resources Per Executor	Submitted Time	User	State	Duration

Completed Applications (0)

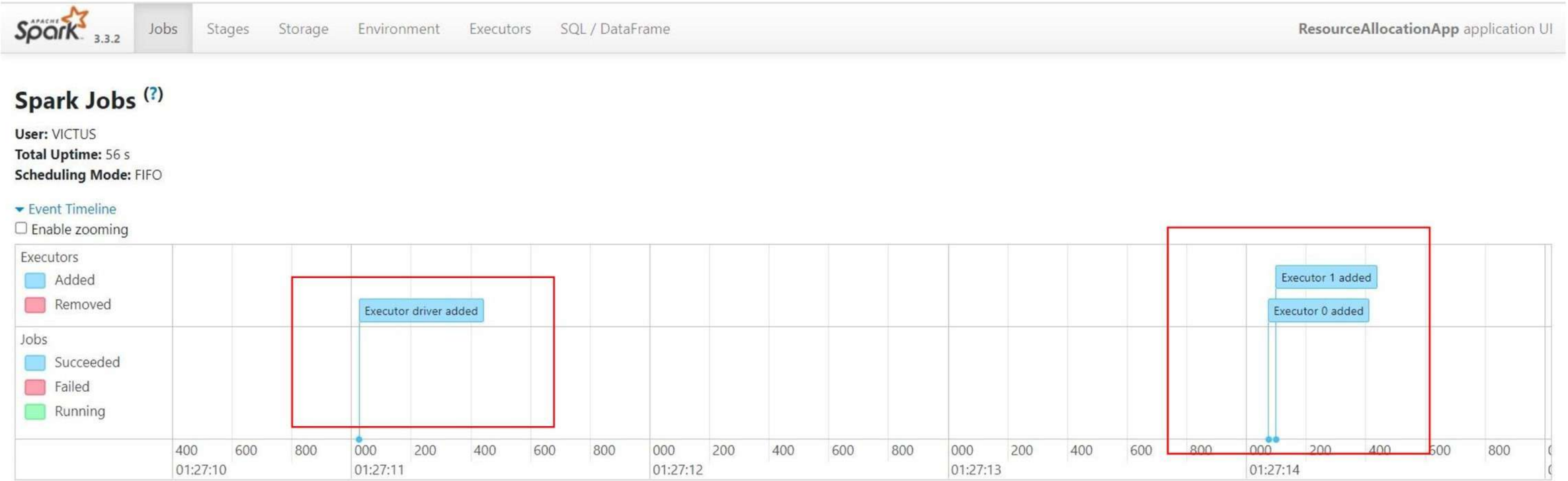
Application ID	Name	Cores	Memory per Executor	Resources Per Executor	Submitted Time	User	State	Duration

Connect notebook to Cluster

```
In [1]: import findspark
        findspark.init()
        findspark.find()

Out[1]: 'E:\\Local d\\Zartab\\Apache Spark Installation\\spark-3.3.2-bin-hadoop3'
```

```python
from pyspark.sql import SparkSession
from pyspark.sql.types import *
from pyspark.sql.functions import *

spark = (
            SparkSession
                .builder
                .appName("ResourceAllocationApp")

                # Connect to multi-node cluster
                .master("spark://localhost:7077")


                # Disable Dynamic Resource Allocation
                .config("spark.dynamicAllocation.enabled", "false")


                # Define configuration for application
                .config("spark.cores.max",          "4")
                .config("spark.executor.memory",    "2g")
                .config("spark.executor.cores",     "2")

                .config("spark.sql.adaptive.enabled", "false")

                .getOrCreate()
        )
```

Check the Master Spark UI



Restart the Application

Restart the Application

In [ ]:

In [1]:
```
import findspark
findspark.init()
findspark.find()
```

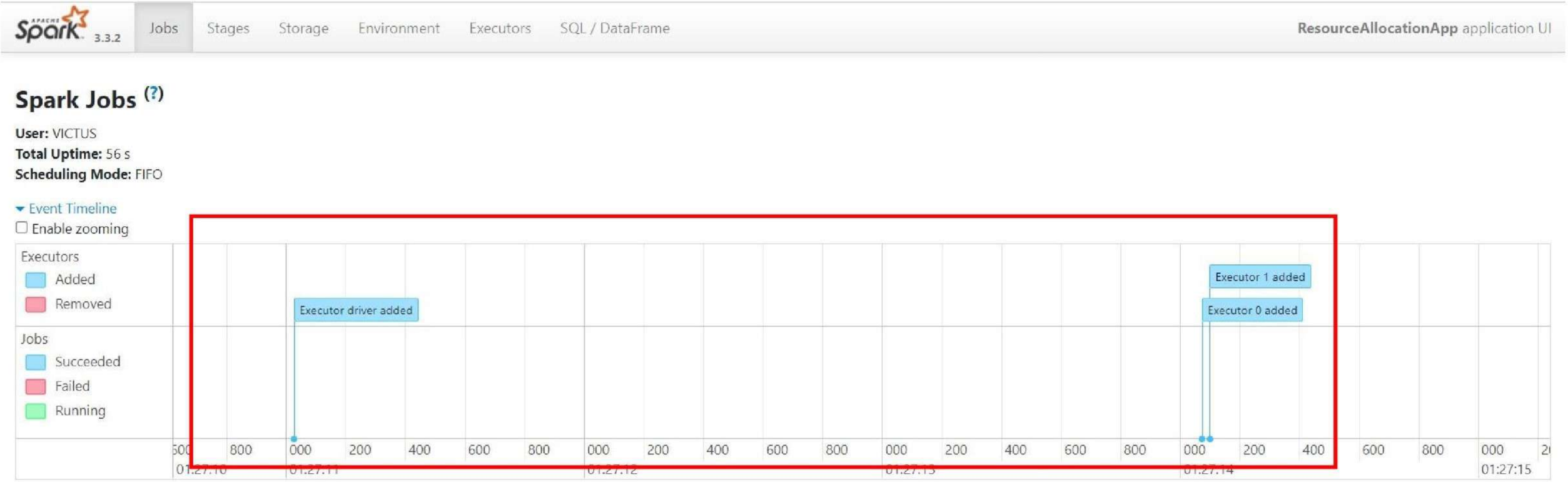
Out[1]: 'C:\\Installation\\spark-3.3.2-bin-hadoop3'

Enable Dynamic Allocation

```python
from pyspark.sql import SparkSession
from pyspark.sql.types import *
from pyspark.sql.functions import *

spark = (
            SparkSession
                .builder
                .appName("ResourceAllocationApp")


                # Connect to multi-node cluster
                .master("spark://localhost:7077")

                .config("spark.sql.adaptive.enabled", "false")


                # Enable dynamic allocation
                .config("spark.dynamicAllocation.enabled",                "true")
                .config("spark.dynamicAllocation.shuffleTracking.enabled", "true")
```

```
# Define configuration for application
.config("spark.executor.memory",    "2g")
.config("spark.executor.cores",     "2")

.config("spark.executor.instances", "2")


# Define minimum and maximum executors
.config("spark.dynamicAllocation.minExecutors", "0")
.config("spark.dynamicAllocation.maxExecutors", "5")

# Set scheduler backlog timeout
.config("spark.dynamicAllocation.schedulerBacklogTimeout",   "1s")

# Set executor idle timeout
.config("spark.dynamicAllocation.executorIdleTimeout",       "10s")

# Set cached idle timeout
.config("spark.dynamicAllocation.cachedExecutorIdleTimeout", "10s")


.getOrCreate()
)

spark
```

Check the UI


Spark 3.3.2
Jobs
Stages
Storage
Environment
Executors
SQL / DataFrame
ResourceAllocationApp application UI
Spark Jobs (?)
User: VICTUS
Total Uptime: 56 s
Scheduling Mode: FIFO
Event Timeline
Enable zooming
Executors
Added
Removed
Jobs
Succeeded
Failed
Running
Executor driver added
Executor 1 added
Executor 0 added
01:27:15

Wait for 10 seconds

```
yellowTaxiDF = (
                  spark
                    .read
                    .option("header", "true")
                    .option("inferSchema", "true")
                    .csv("E:\SparkData\Raw\YellowTaxis_202210.csv")
                )

yellowTaxiGroupedDF = (
                            yellowTaxiDF
                                .dropDuplicates()

                                .groupBy("PULocationID")
                                .agg(sum("total_amount"))
                        )

yellowTaxiGroupedDF.show()
```

```
+------------+------------------+
|PULocationID| sum(total_amount)|
+------------+------------------+
|         148|234499.04999999987|
|         243| 2877.810000000001|
```

Spark 3.3.2 | Jobs | Stages | Storage | Environment | Executors | SQL / DataFrame | ResourceAllocationApp applica

Spark Jobs (?)

User: VICTUS
Total Uptime: 1.6 min
Scheduling Mode: FIFO
Active Jobs: 1
Completed Jobs: 2

Event Timeline

Enable zooming

Active Jobs (1)

Page: 1

1 Pages. Jump to 1 . Show 100 items in a page.

Resource Allocation Using Fair Scheduling

Resource Scheduling for a Job

Scheduling Across Applications

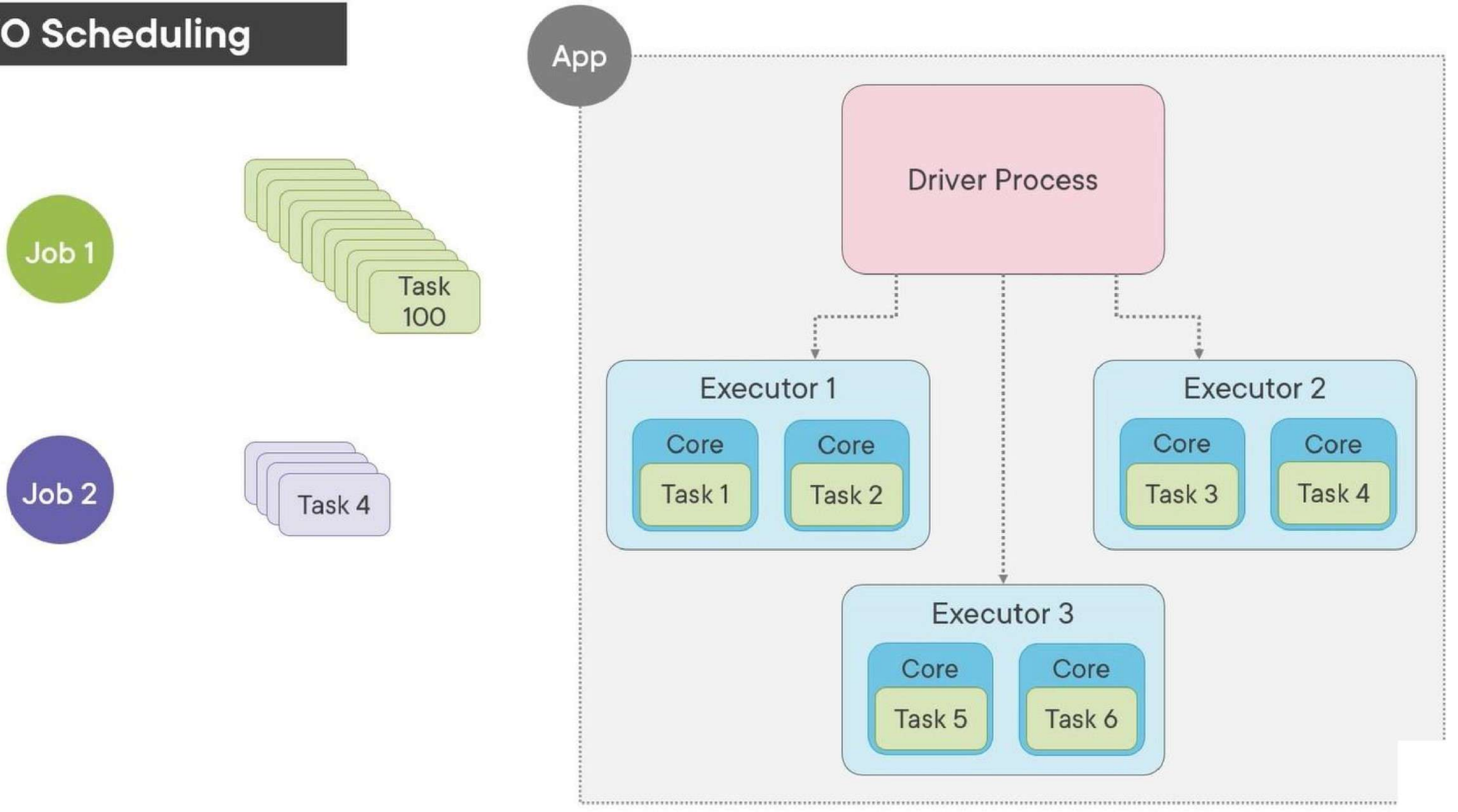
Scheduling Within an Application

Resource Scheduling Within Application

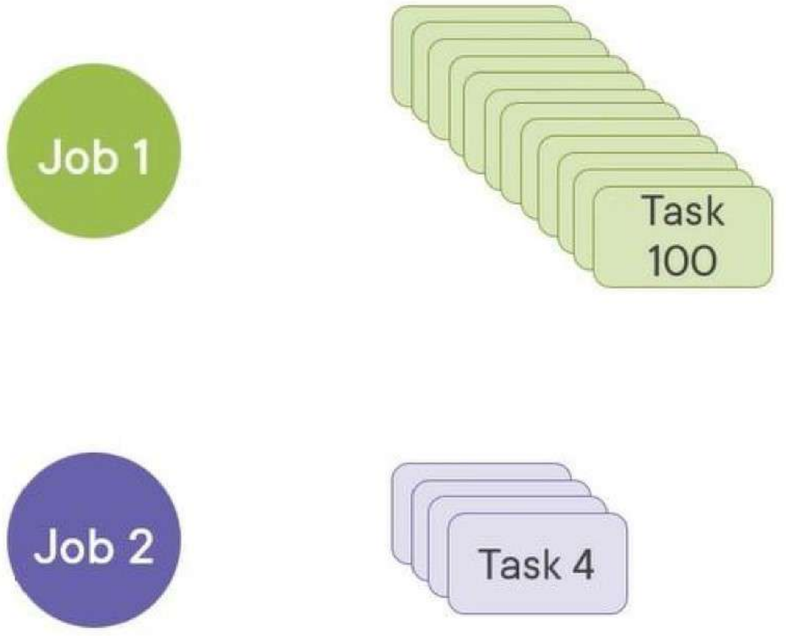
FIFO Scheduling

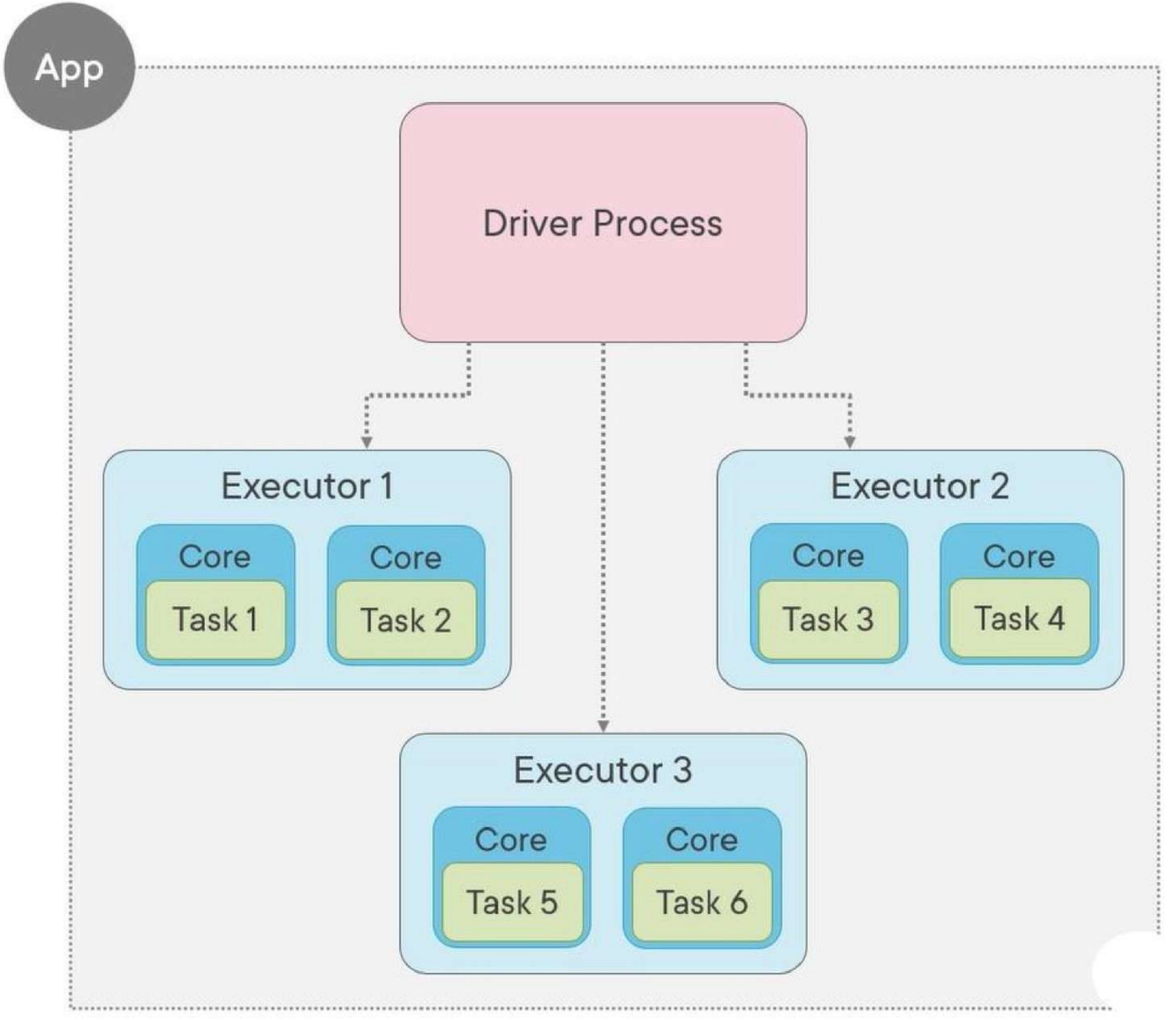
FAIR Scheduling

FIFO Scheduling

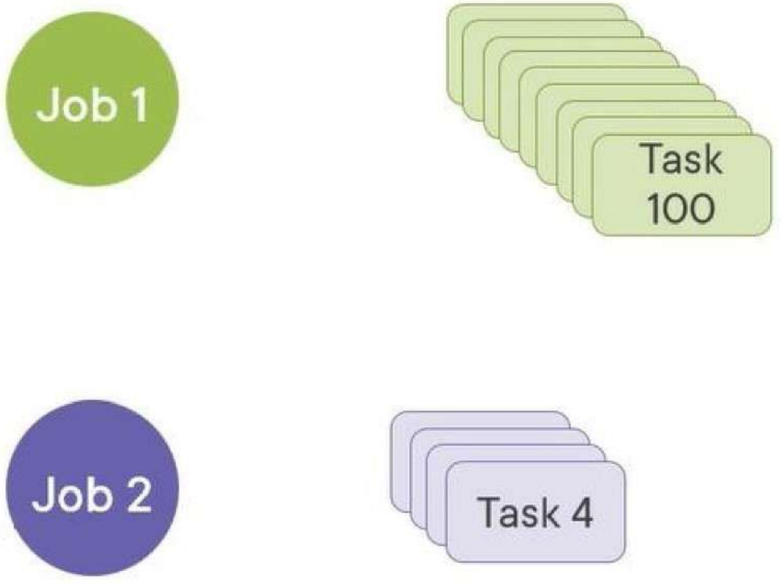


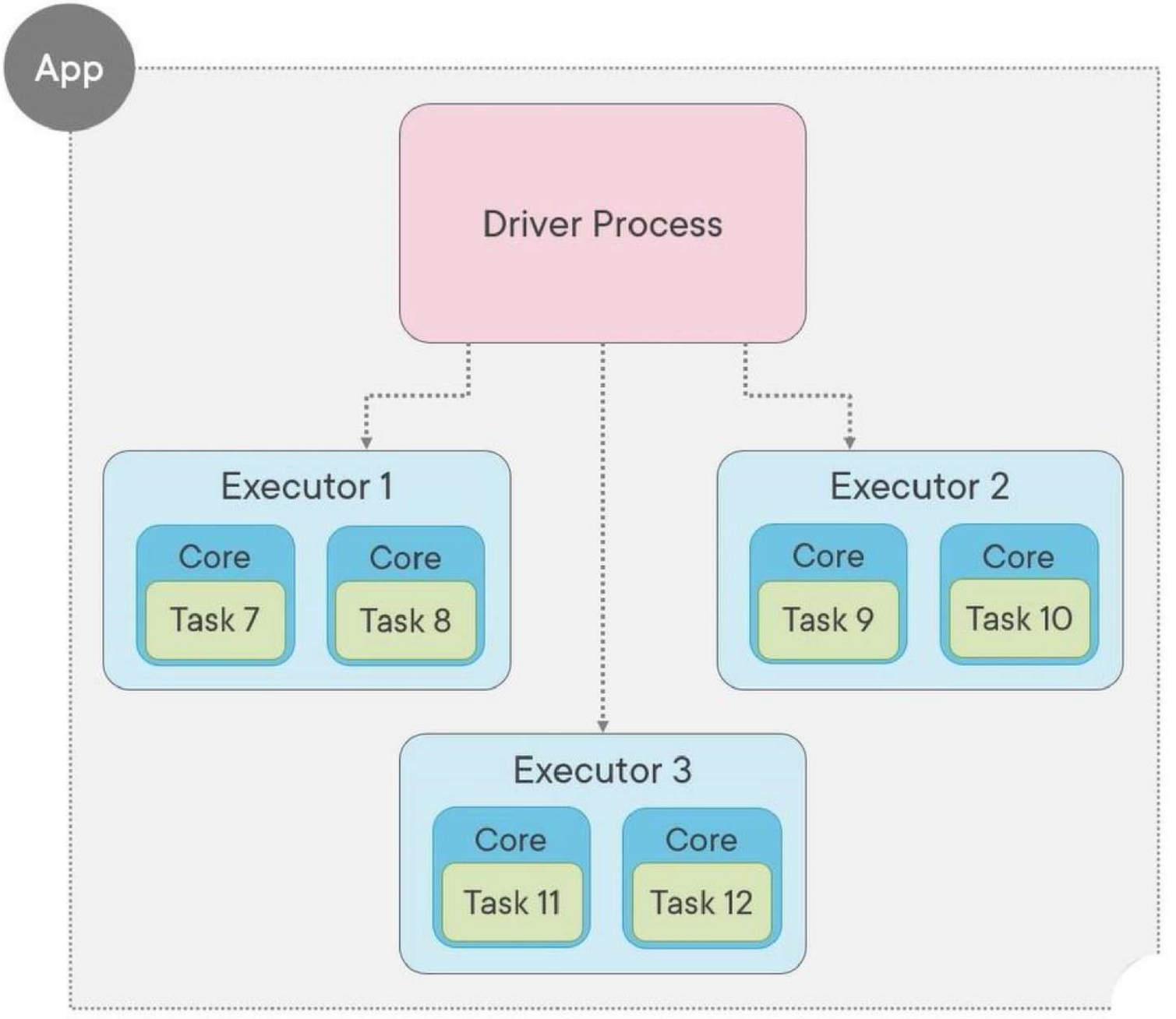
FIFO Scheduling





FIFO Scheduling



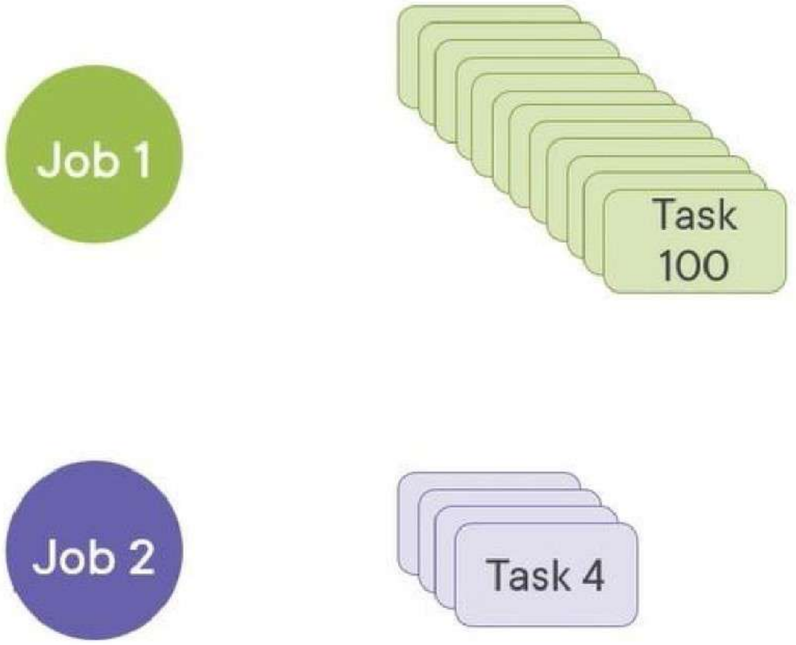


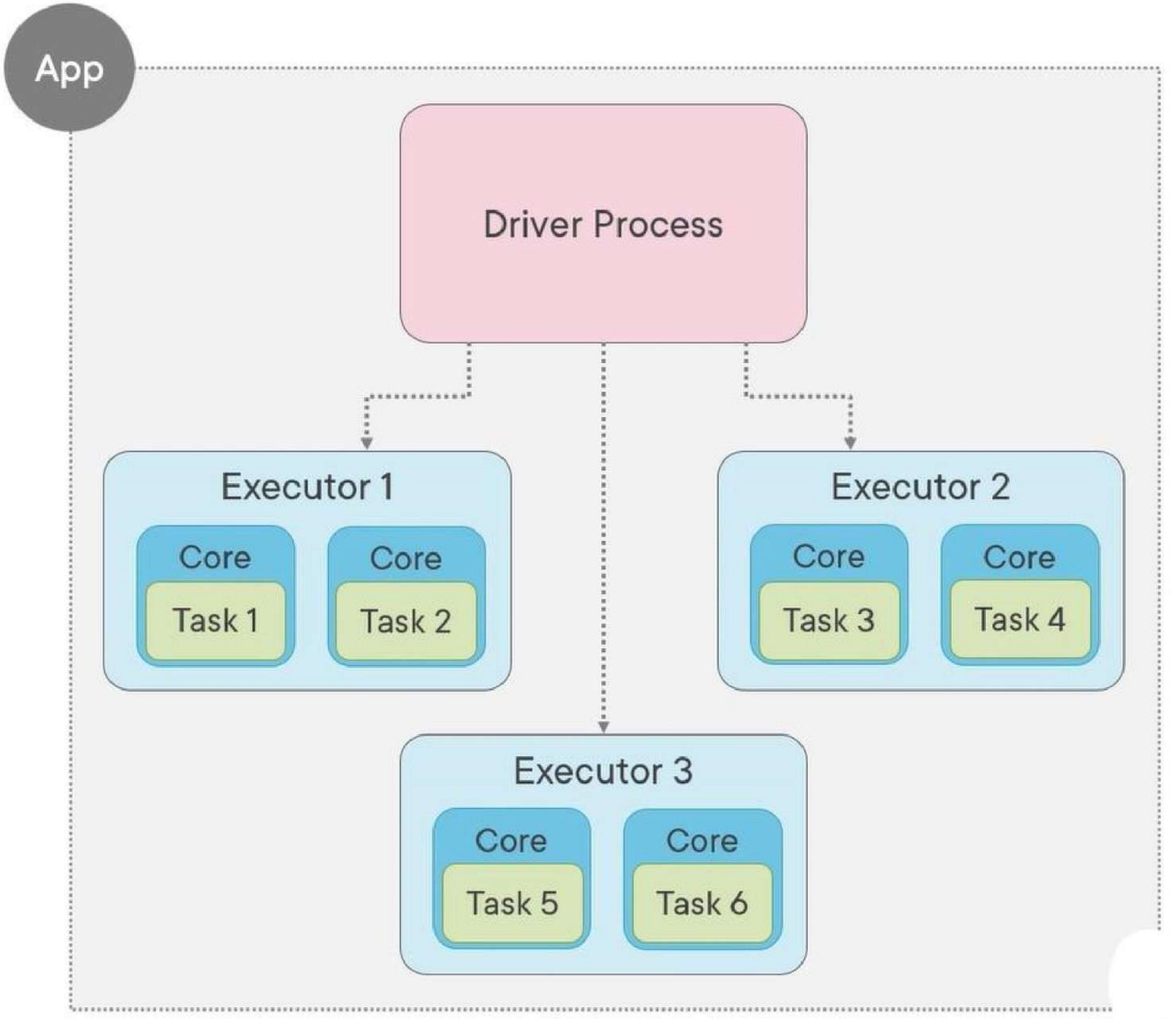
Default scheduling

First job gets priority on resources

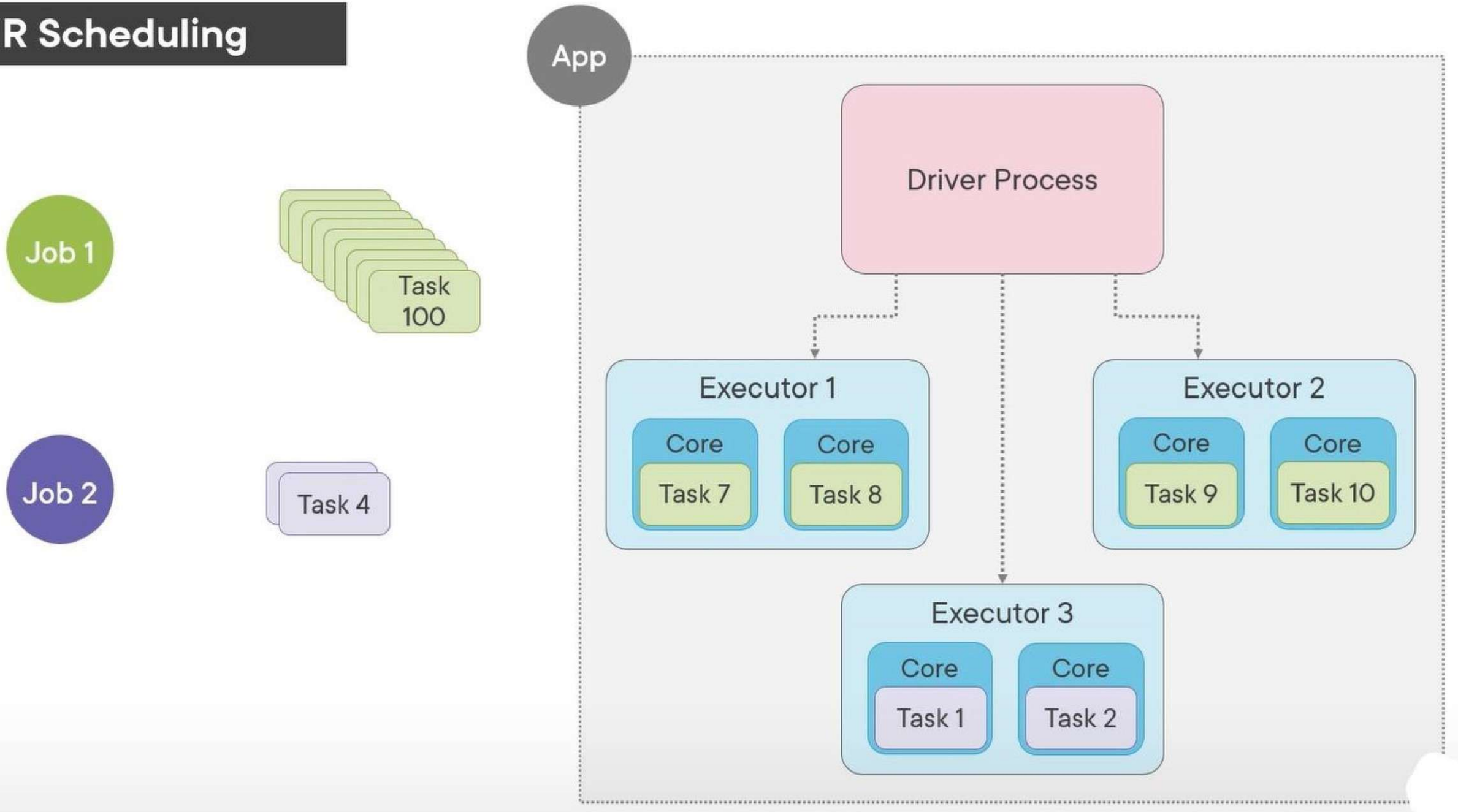
Other jobs in queue are delayed

FAIR Scheduling



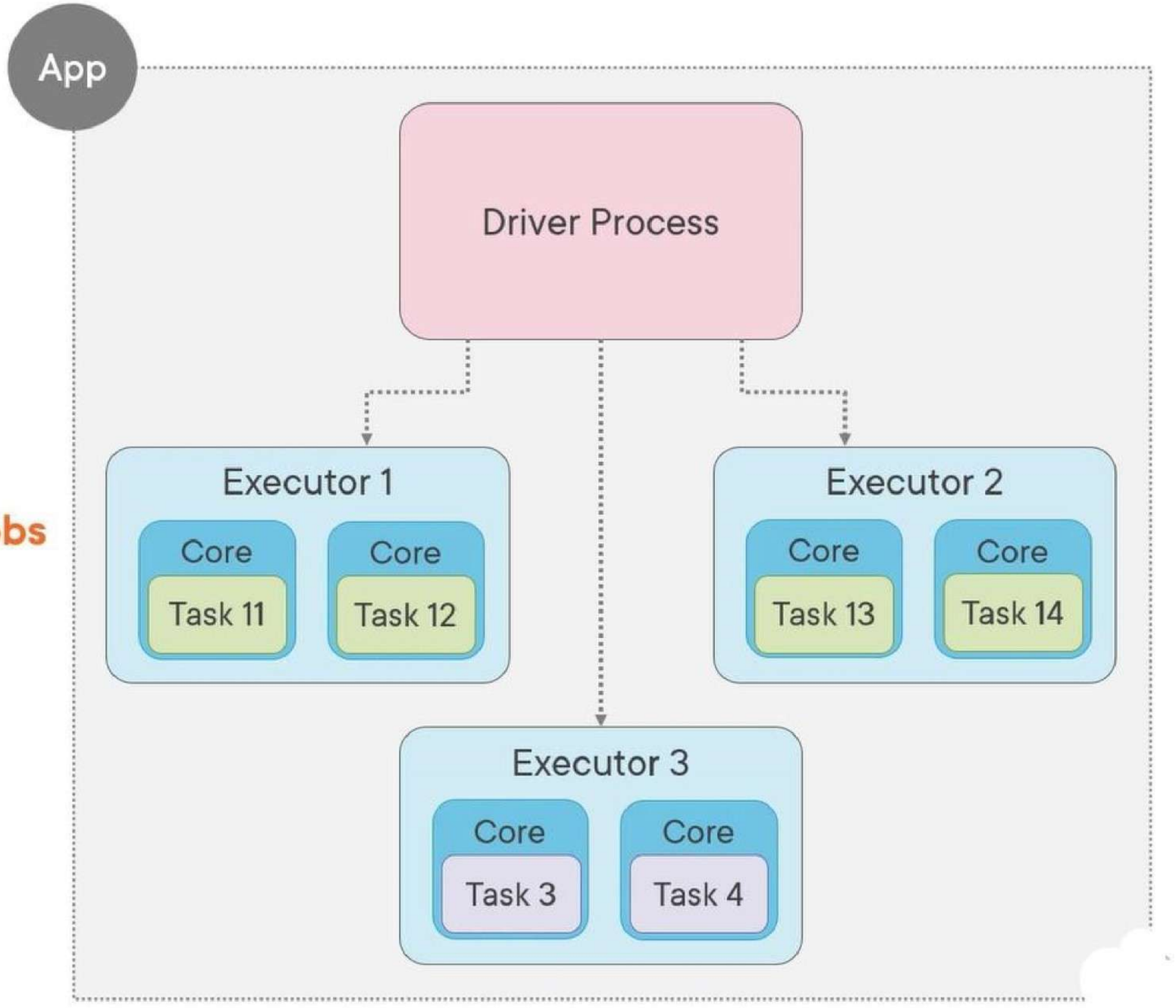


FAIR Scheduling

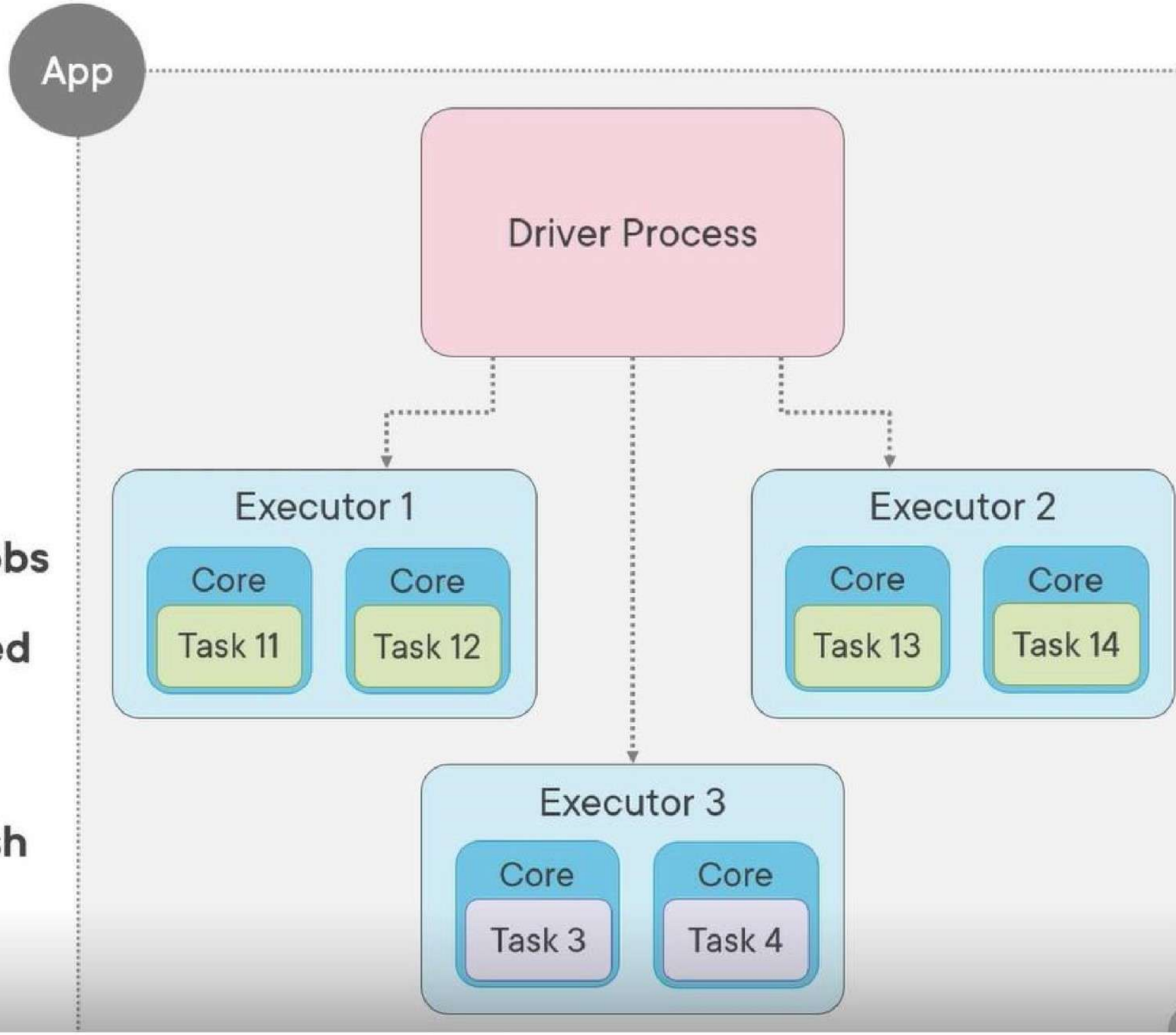


FAIR Scheduling

Fair sharing of resources between jobs



FAIR Scheduling



Fair sharing of resources between jobs

Tasks from different jobs are assigned in a round-robin fashion

Small jobs can finish faster, without waiting for long-running jobs to finish

To configure, set:

- spark.scheduler.mode = FAIR