# Introduction to Apache Spark APIs for Data Processing

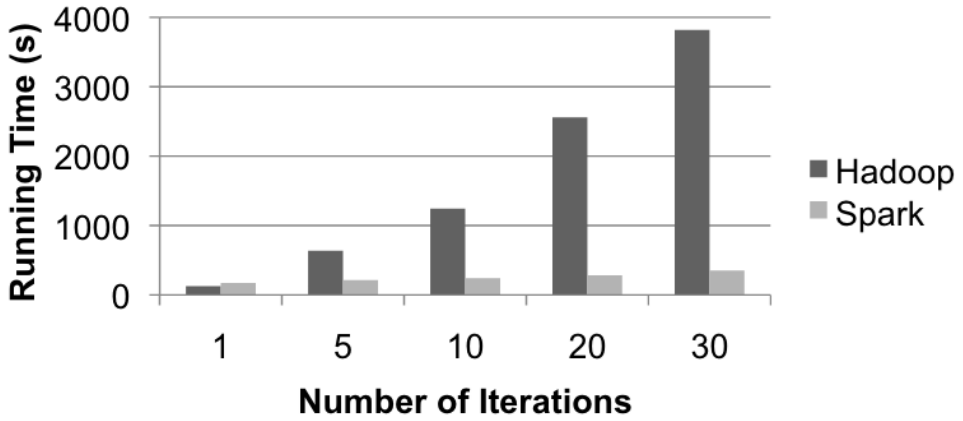
Luca Canali
CERN IT, Data Analytics and Spark Service

# Open Source Big Data - The Beginning

- 2004: “MapReduce: Simplified Data Processing on Large Clusters” J. Dean and S. Ghemawat (Google)
- 2006: Apache Hadoop as an open source implementation of the MapReduce programming model
  - Hadoop MapReduce
  - Hadoop YARN
  - Hadoop Distributed File System (HDFS)

# Spark to the Rescue

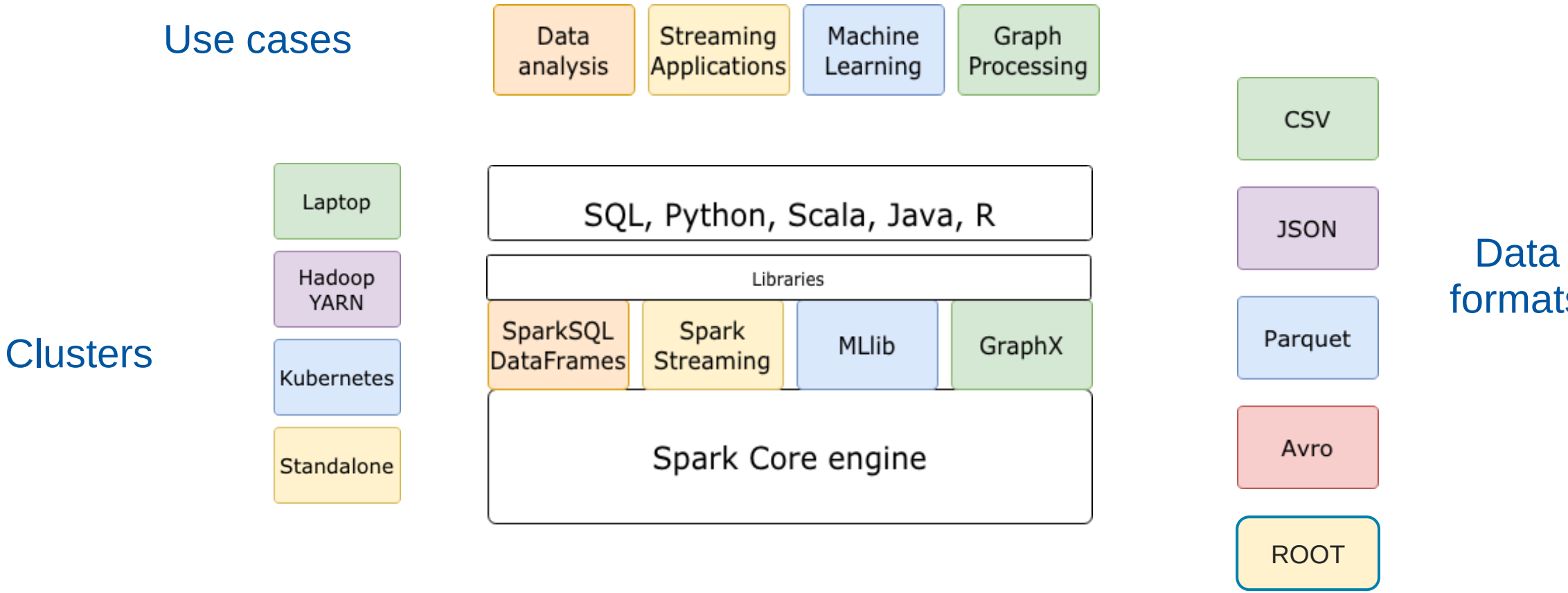
- 2009: “Spark: Cluster Computing with Working Sets”, M. Zaharia et al.
  - MR is slow, and is hard to program!
  - Spark introduces Resilient Distributed Dataset (RDD) abstraction
- 2014: Introduction of SparkML and GraphX
- 2015: Introduction of Spark SQL and DataFrame APIs
- 2016-today: large adoption of Apache Spark in the industry (Databricks, Apple, Netflix...) and active development
- 2022 - June: latest release Spark 3.3.0



# What is Apache Spark?

- A unified analytics engine for large-scale data processing with expressive development APIs
  - Enables processing of large data sets
  - Allows for sophisticated analytics, real-time streaming, and machine learning

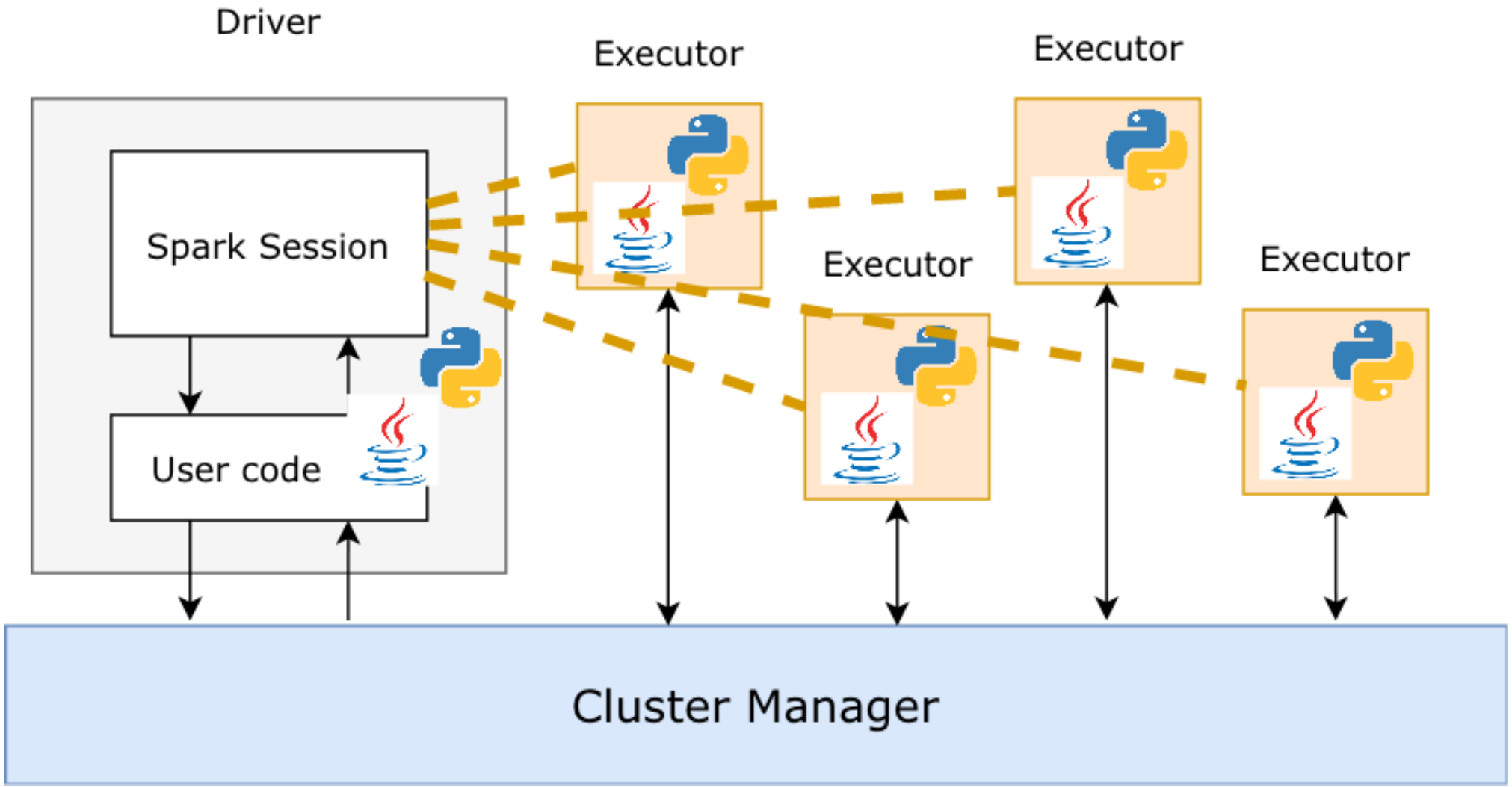
# What is Apache Spark?



# How does it work?

- Computations are distributed across several nodes
- Optimized for running at scale
  - Fault tolerance

# Spark Architecture



# Spark Architecture - SparkSession

- One-to-one correspondence between a SparkSession and Spark Application
- SparkSession
  - is the entry point for user-defined data processing
- SparkSession
  - is available as variable spark when you start Scala console (spark-shell) or Python console (pyspark)

# Spark Architecture

- **Driver**
  - SparkSession is created and resides here
  - Distributes and schedules work across the executors
  - Manages executors lifecycle
  - When using REPL (command line) is the rntry point for Spark Shell (Scala) PySpark (Python)

# Spark Architecture – Executor(s)

- Responsible for carrying out the work assigned by the driver, at scale
- Reading data from Storage (HDFS or external sources)
- Storing the data in cache in memory or on HDDs
- Performing all data processing
- Writing data to Storage (HDFS or external sinks)

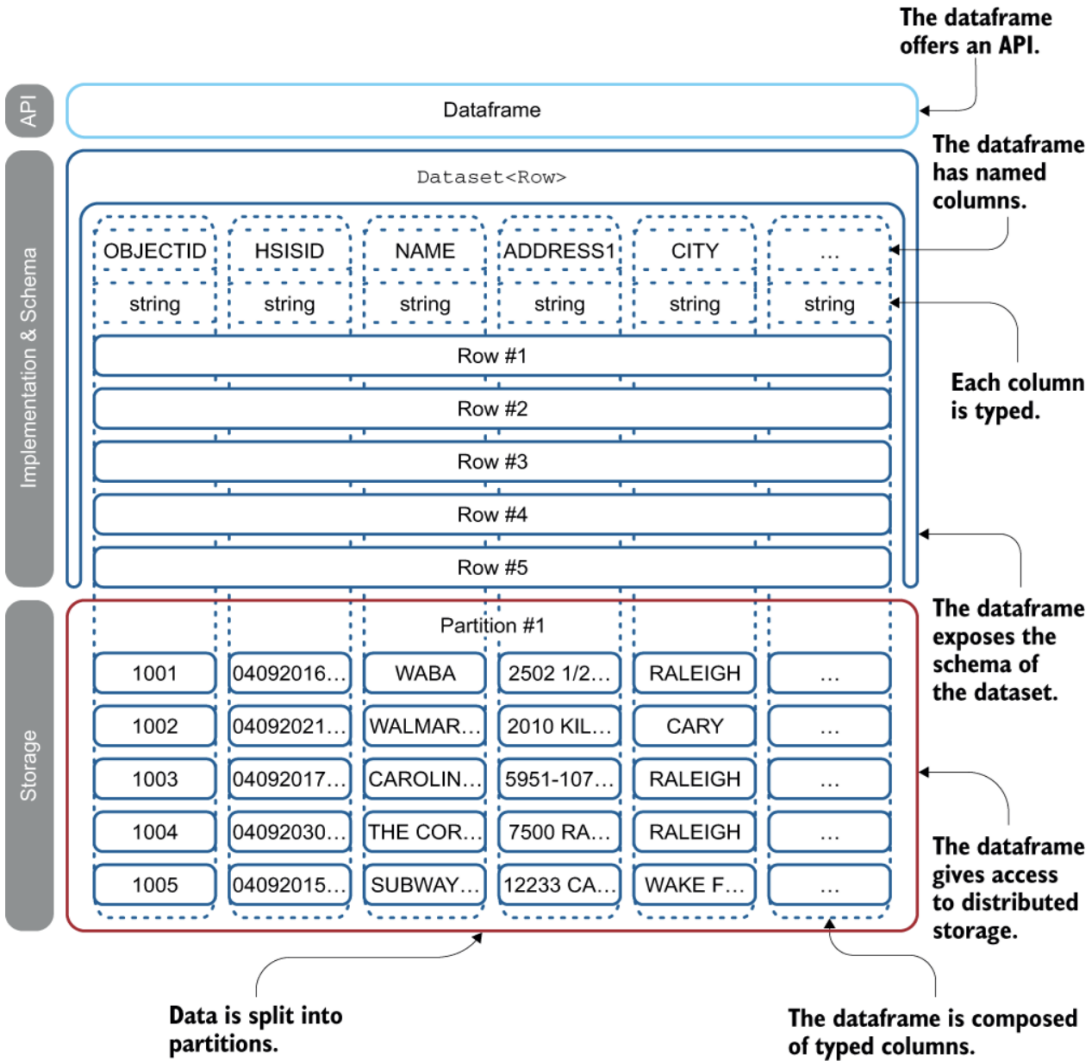
# Cluster Manager

- The main cluster managers are:
  - YARN: cluster manager of the Hadoop project
  - Kubernetes: Linux containers orchestrator for cloud developments
  - Standalone: use this to manually setup a cluster
- Deploy modes:
  - **client** mode: the driver is external from the cluster (i.e. on your desktop or on a dedicated host)
  - **cluster** mode: the application is running entirely in the cluster (useful for batch use cases)

# Spark DataFrames (DF)

- DataFrames are the higher-level data structure and API in Spark
  - Implemented using an immutable distributed table of records with rows, columns and a schema
- Analogous to:
  - a Table in a DB (but: no indexes, primary keys, constraints, etc…)
  - a DataFrame in Python / R
- Important:
  - DataFrames are divided in partitions, distributed across multiple executors

# Main Data Abstraction: Spark DataFrames




- DataFrame is a table-like abstraction
  - similar to Pandas DF
- Handles data with a schema
- DFs are partitioned and immutable
  - enables parallel execution
  - and fault tolerance at scale

Figure from "Spark in action", Jean-Georges Perrin, Manning, 2020

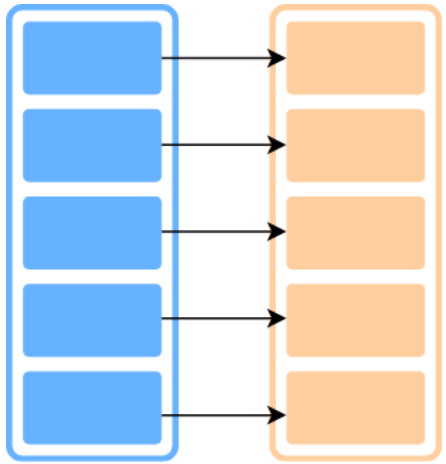
# Actions and Transformations

- Two types of operations on DFs:
  - **Transformations**: transform a DF in another one:
    - filter, select, orderBy, ...
    - lazy evaluation: transformations do not trigger computation
  - **Actions**: trigger computation and return value
    - show, count, collect, write, …

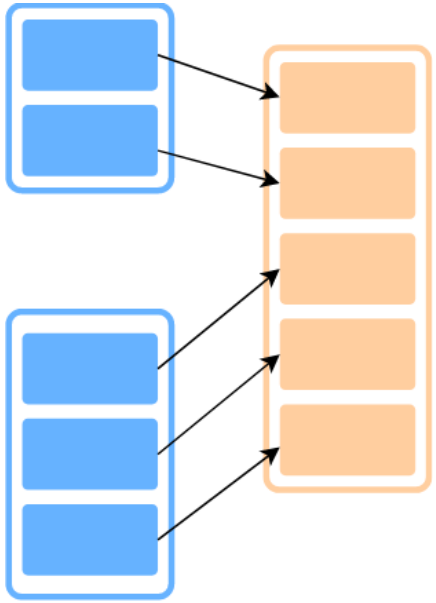
# Narrow and Wide Transformations

- Narrow transformations
  - are more performant, because they will be executed in one pass in memory thanks to lazy evaluation
- Wide transformations
  - result in data exchange between nodes, in a process called shuffle
- Shuffle optimization key for distributed data operations

# Narrow and Wide Transformations

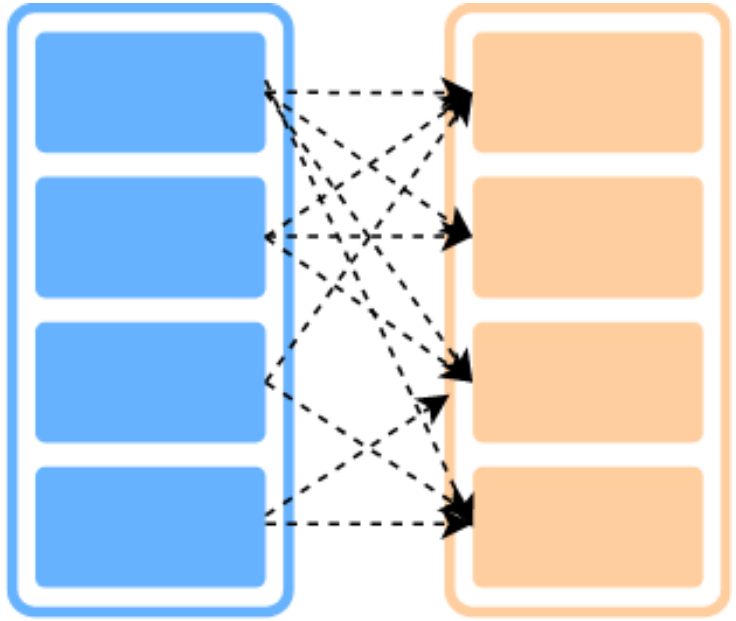


- filter/ select



- union

- groupby



# Actions

- Actions instruct Spark to compute a result from a series of transformations
- Types of actions:
  - actions for viewing data in the console
  - collecting data to native objects, in respective languages
  - writing data to storage systems (HDFS, S3, EOS etc)

# Actions and Transformations

- **Lazy evaluation** and **immutability**:
  - Optimize query when more information is available
  - Fault tolerance: the transformations can be replayed on the original DF

# Example of Actions and Transformations

```
from pyspark.sql import Row

df2=spark.createDataFrame([Row(id=x) for x in
range(10)])

df1=spark.createDataFrame([Row(id=x) for x in
range(10)])

df1.filter(df1.id>4).join(df2,
df1.id==df2.id).count()
```

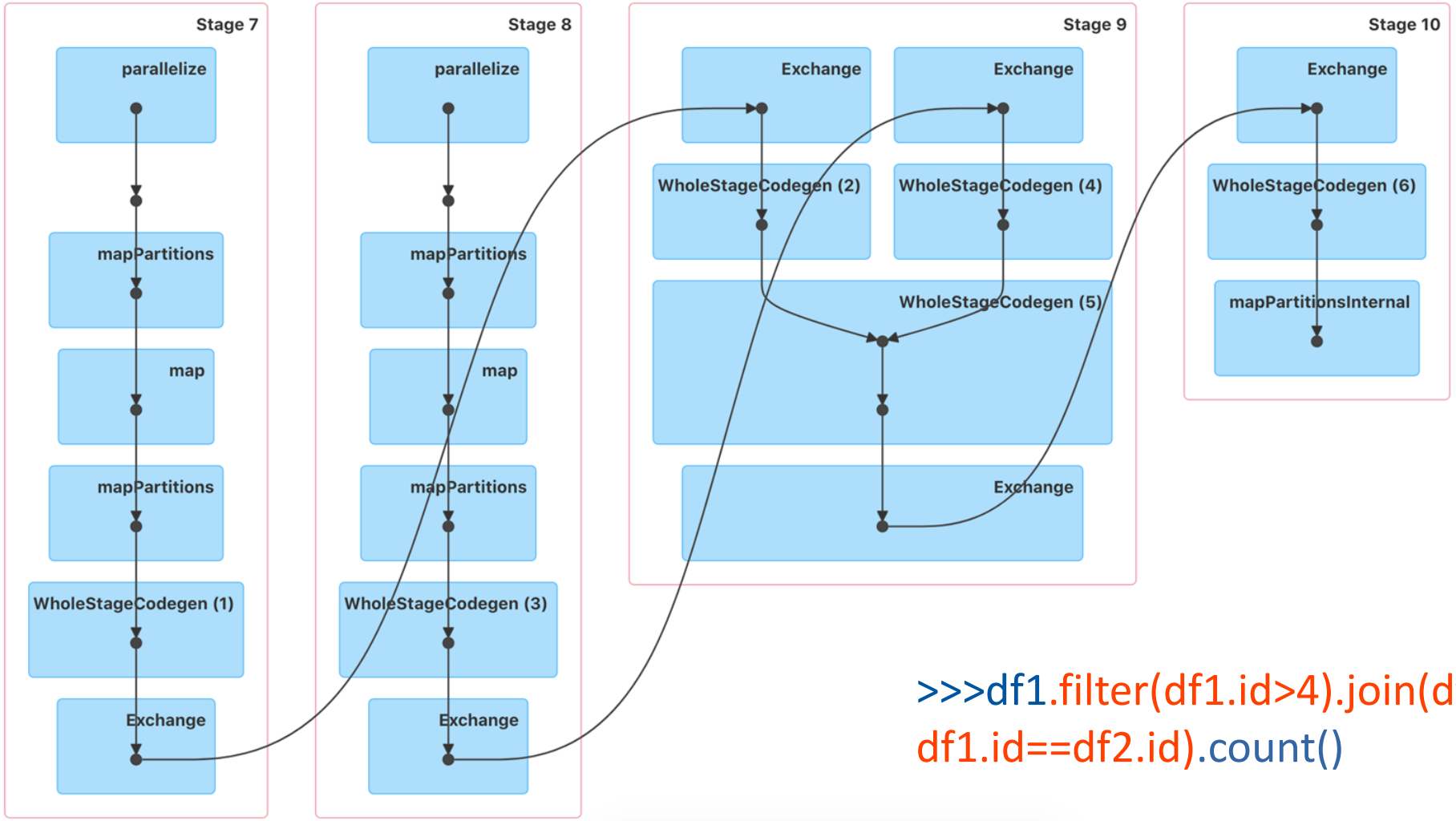
TRANSFORMATION

ACTION

# What about the Execution?

- Invoking an action creates a job, which is then divided in stages and tasks.
- Spark triggers the creation of graph of computations (DAG) and its division into stages and tasks.
- Tasks are the units of parallelization and are run concurrently in the executors.

# What about the execution?




```
>>>df1.filter(df1.id>4).join(df2,
df1.id==df2.id).count()
```

# Web ui

▾ Active 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 |
|---|---|---|---|---|---|
| 7 | count at <console>:26<br>count at <console>:26 (kill) | 2019/08/10 17:50:13 | 17 s | 0/2 | 0/5 (4 running) |

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

▾ Completed Jobs (7)

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 |
|---|---|---|---|---|---|
| 6 | show at <console>:26<br>show at <console>:26 | 2019/08/10 17:49:30 | 0.4 s | 1/1 | 1/1 |
| 5 | show at <console>:28<br>show at <console>:28 | 2019/08/10 17:48:32 | 0.8 s | 3/3 | 9/9 |
| 4 | show at <console>:28<br>show at <console>:28 | 2019/08/10 17:47:40 | 2 s | 3/3 | 9/9 |

## Executors

▾Show Additional Metrics

- Select All
- On Heap Memory
- Off Heap Memory

### Summary

| | RDD Blocks | Storage Memory | Disk Used | Cores | Active Tasks | Failed Tasks | Complete Tasks | Total Tasks | Task Time (GC Time) | Input | Shuffle Read | Shuffle Write | Blacklisted |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Active(3) | 0 | 5.9 KiB / 1.1 GiB | 0.0 B | 2 | 0 | 0 | 5 | 5 | 4 s (0.2 s) | 0.0 B | 0.0 B | 0.0 B | 0 |
| Total(3) | 0 | 5.9 KiB / 1.1 GiB | 0.0 B | 2 | 0 | 0 | 5 | 5 | 4 s (0.2 s) | 0.0 B | 0.0 B | 0.0 B | 0 |
| Dead(0) | 0 | 0.0 B / 0.0 B | 0.0 B | 0 | 0 | 0 | 0 | 0 | 0.0 ms (0.0 ms) | 0.0 B | 0.0 B | 0.0 B | 0 |

### Executors

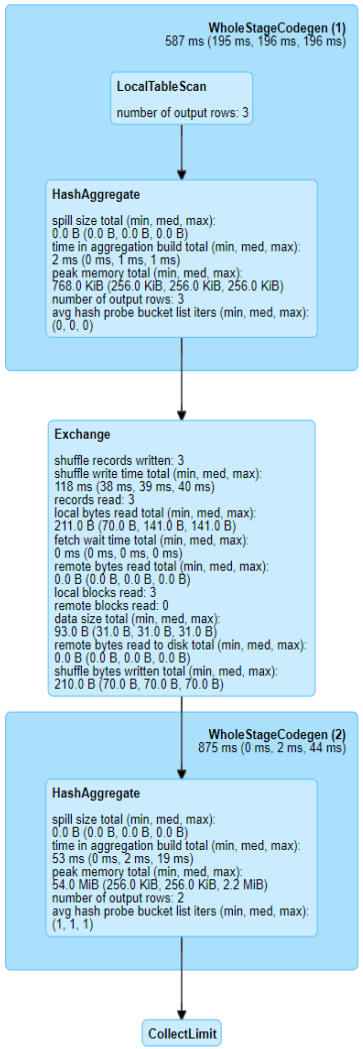
Show 20 entries — Search:

| Executor ID | Address | Status | RDD Blocks | Storage Memory | Disk Used | Cores | Active Tasks | Failed Tasks | Complete Tasks | Total Tasks | Task Time (GC Time) | Input | Shuffle Read | Shuffle Write | Logs | Thread Dump |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 10.12.221.27:55834 | Active | 0 | 2 KiB / 366.3 MiB | 0.0 B | 1 | 0 | 0 | 3 | 3 | 2 s (0.1 s) | 0.0 B | 0.0 B | 0.0 B | stdout stderr | Thread Dump |
| 0 | 10.12.221.27:55835 | Active | 0 | 2 KiB / 366.3 MiB | 0.0 B | 1 | 0 | 0 | 2 | 2 | 2 s (94.0 ms) | 0.0 B | 0.0 B | 0.0 B | stdout stderr | Thread Dump |
| driver | 10.12.221.27:55827 | Active | 0 | 2 KiB / 366.3 MiB | 0.0 B | 0 | 0 | 0 | 0 | 0 | 0.0 ms (0.0 ms) | 0.0 B | 0.0 B | 0.0 B | | Thread Dump |

Showing 1 to 3 of 3 entries — Previous 1 Next

## Details for Query 2

**Submitted Time:** 2019/11/20 09:31:38
**Duration:** 1 s
**Succeeded Jobs:** 1 2 3 4 5




▸ Details

# DataFrame API

# Schema

- Schema is metadata: column names, and data types of a DataFrame
- Can be inferred on read, but it's better to specify it when using large JSON/CSV
  - performance
  - correctness
- Some formats store data with its schema and are very useful for data analytics.
  - Apache Parquet, ORC

```
Id, Name, Surname
1,Albert,Einstein
2,Isaac,Newton
.....

myschema="Id int, Name string,
Surname string"

spark.read.csv("scientists_names.
csv" , schema=myschema)
```

# Columns

- Columns in Spark DFs are similar to columns in spreadsheet, databases or pandas DataFrames
- Select
  - df.c
  - df["c"]
- Manipulate
  - df.withColumn("a*3",expr("a*3"))
  - df.withColumn("isEven",expr("a%2"))
- Remove
  - df.drop(a)

**Remember! Neither of these modify the existing Dataframe, they all return a new one**

# Basic DataFrame Operations

- Projection

```
df = spark.range(10)
df2 = df.select("id")
```

- Selection

```
df3 = df.select("id").filter("id > 5")
```

- Aggregations

```
from pyspark.sql.functions import sum
df4 = df.agg(sum("id"))
```

# Caching DataFrames

- The cache method tells the Spark engine that it must store the DF locally for later reuse
- Cache is evaluated lazily, which means it is run only when the first action is run
- Use unpersist to remove data from cache

# Key Learning Points

- Apache Spark
  - Is a library and framework for data processing at scale
  - Large adoption in industry and many integrations with storage and compute systems
- Spark DataFrame
  - It's a scalable and fault-tolerant abstraction for processing data with schema
  - It has a rich and powerful API for processing large data sets