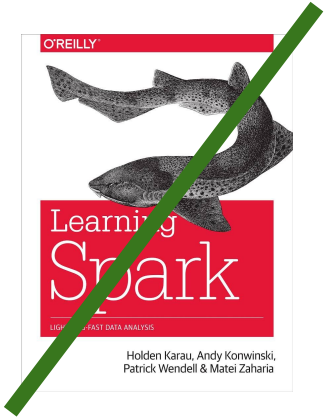
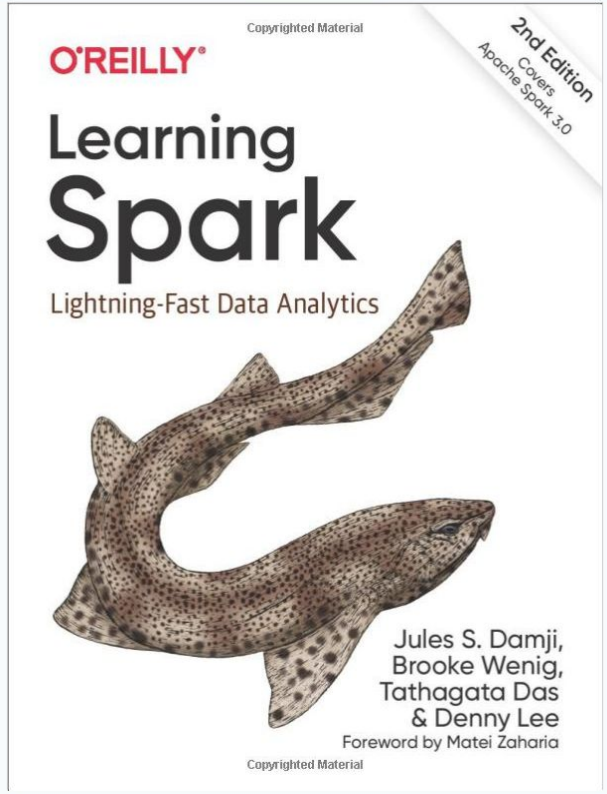
ADALTAS

2. Introduction to

Petra KAFERLE DEVISSCHERE

Before we start...





Modules

ADALTAS

Part 1: Introduction, RDDs

- Quick introduction to Apache Spark
- Spark internals
- What are RDDs and how to use them?
- Demo: Unstructured data analysis with RDDs

Part 2: Spark SQL and DataFrames

- Using Spark SQL API to analyze structured datasets
- Why SQL?
- Lab: Structured data analysis with DataFrames

What is Apache Spark?

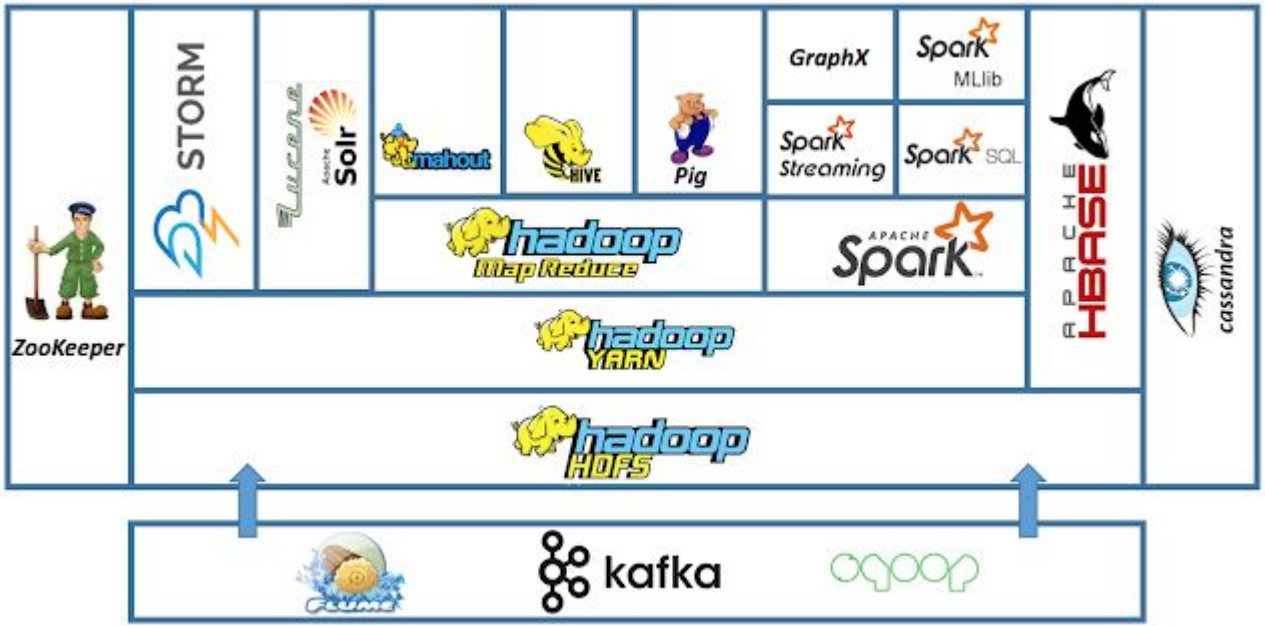
ADALTAS

- Fast (in-memory), distributed (parallel), general-purpose **cluster computing system** - spark.apache.org

- **Open Source** project (Apache Software Foundation)

- Strongly tied to the **Hadoop** ecosystem

Hadoop Ecosystem



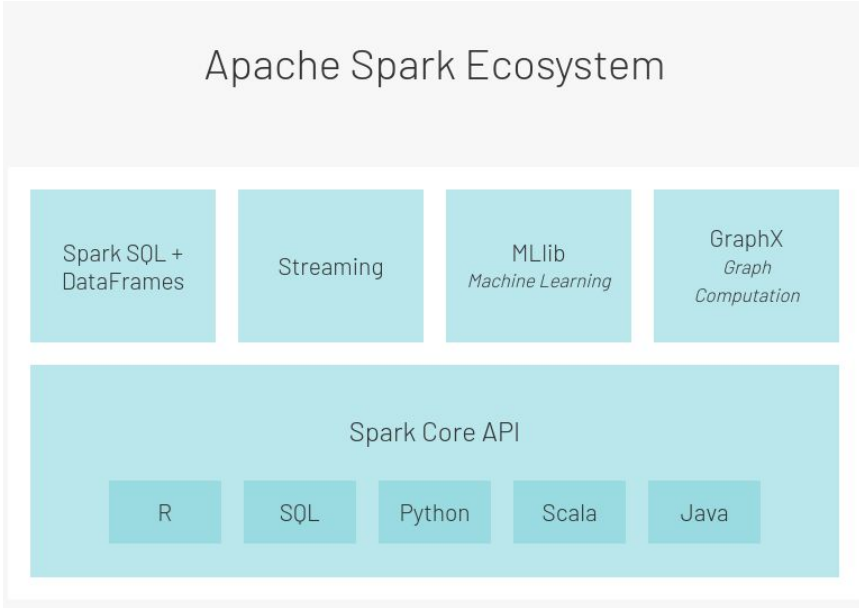
What is Apache Spark?

ADALTAS

- Written in **Scala** → runs in the JVM (Java Virtual Machine)
- Pick your language: **Scala**, **Python**, **R, SQL**, **Java** (not in the notebook environment)
- Sparks transforms your code into **tasks** to run on the **cluster nodes**

Spark Ecosystem

ADALTAS



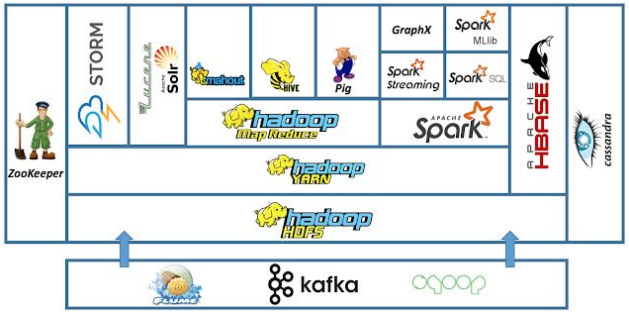
Use cases

ADALTAS

- Analyze / transform / apply ML models on:
  - Very **large datasets** (Extract, Transform and Load)
  - **Streaming** data (in near-real-time)
  - **Graphs** (network analysis)
- of structured (tables), semi-structured (JSON) or unstructured (text) data

Spark Internals

- Spark connects to **resource managers:**
  - Hadoop **YARN**
  - Apache Mesos
  - Kubernetes
  - Spark standalone



- that **distribute resources** (RAM, CPU) to applications running on a cluster

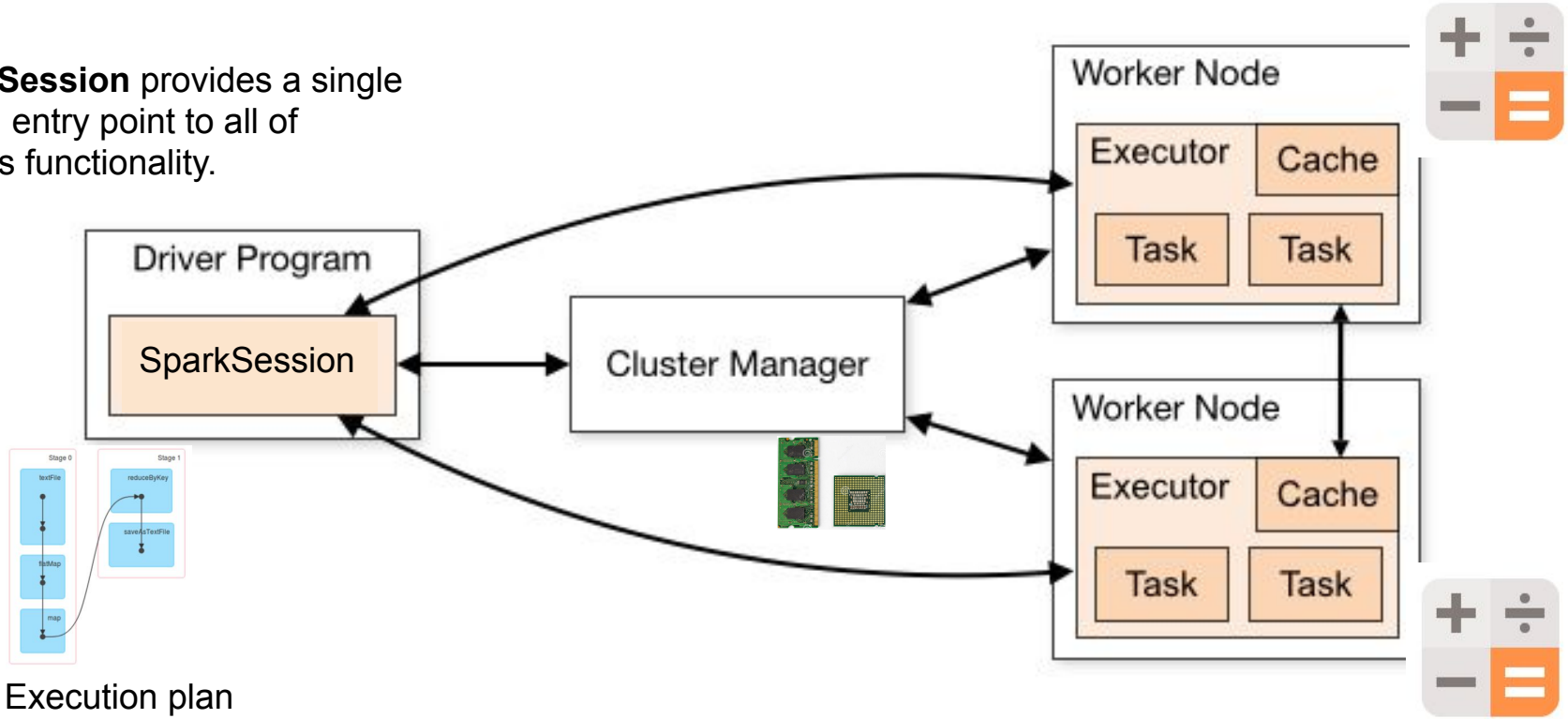
Spark Internals

ADALTAS

- You: Write the **code** and **submit** it
- Spark:
  1. Asks for resources to **create driver + executors**
  2. **Driver** transforms the **code** into **tasks**
  3. **Driver** sends **tasks** to **executors**
  4. **Executors** sends **results** to **driver**

Spark Internals

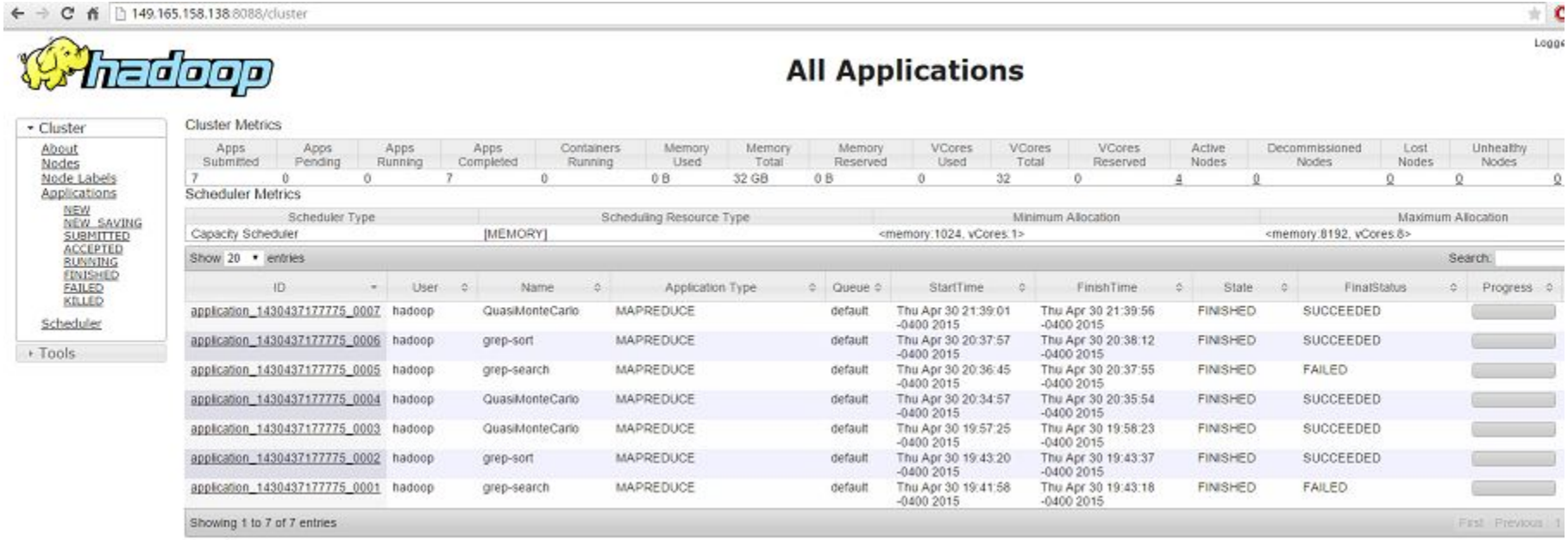
SparkSession provides a single unified entry point to all of Spark's functionality.



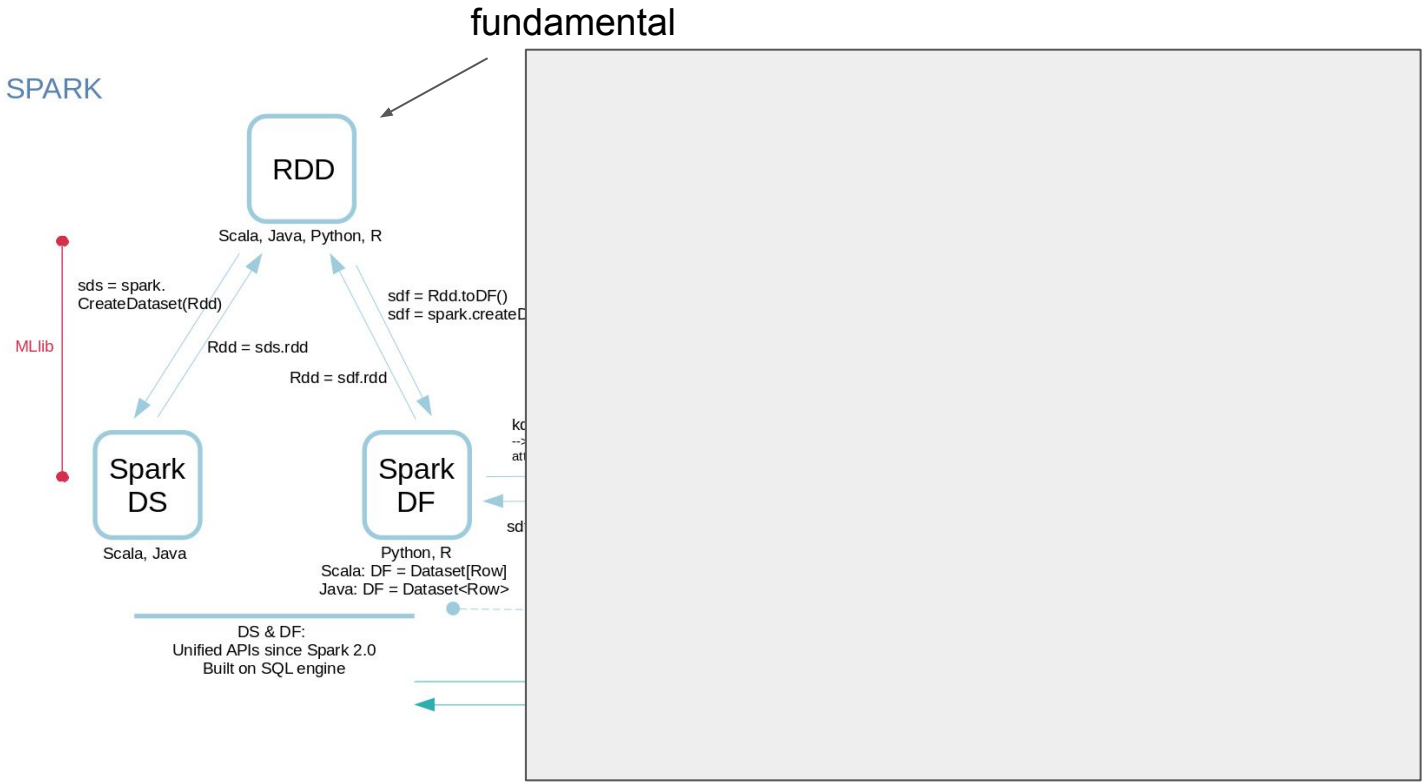
Execution plan

YARN (cluster manager)

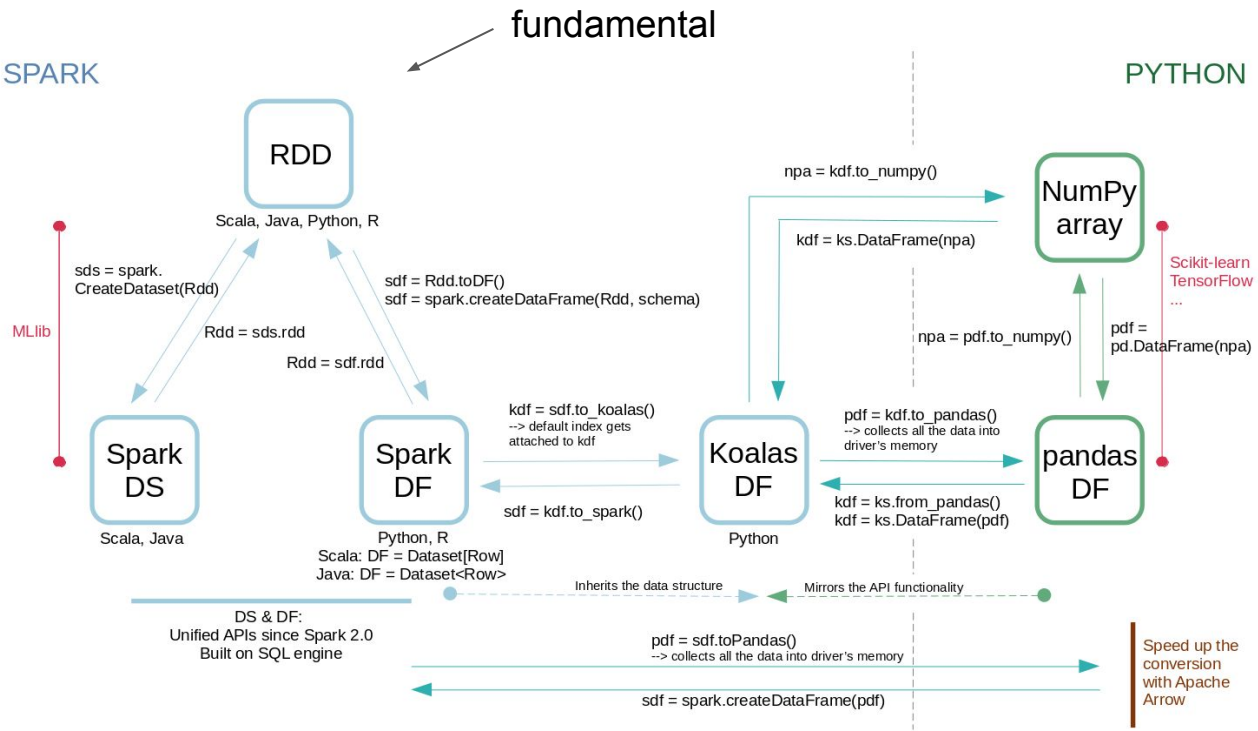
ADALTAS



Spark Data Structures



Spark Data Structures



Spark Operations

ADALTAS

2 types of **operations:**

- **Transformations:** transform a Spark DataFrame/RDD into a new DataFrame/RDD without altering the original data
- **Actions:** get the result

Transformations	Actions
orderBy()	show()
groupBy()	take()
filter()	count()
select()	collect()
join()	save()

Lazy evaluation: transformations triggered when action is called.

RDDs: Resilient Distributed Datasets

ADALTAS

- A **fault-tolerant collection** of elements partitioned **across the nodes** of the cluster (parallelism)
- An element can be: string, array, dictionary, etc.
- An RDD is **immutable**
- Transformations: lambda expressions on **key-value pairs**
- An RDD can be **persisted** in memory for reuse (avoid recomputing)

RDDs: Resilient Distributed Datasets

ADALTAS

- Mostly load data from **HDFS** (or Hadoop-like file system)
- RDDs are **partitioned:**
  - **1 task** runs on **1 partition**
  - Default = 1 partition per CPU core

Spark RDD API

ADALTAS

Chain transformations and use the result with an action :

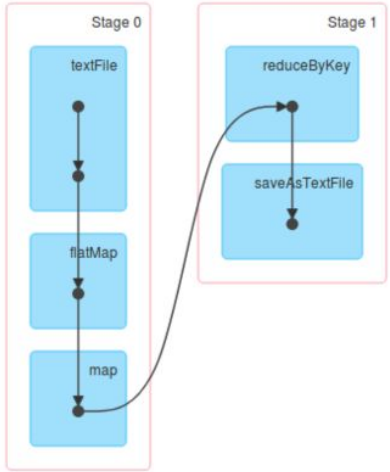
```
rdd1 = sc.wholeTextFiles('hdfs://text/file/path') \
      .map(lambda x: x.split(',')) \    ← transformation
      .flatMap(...) \                   ← transformation
      .groupByKey(...)                  ← transformation

rdd1.take(10)    ← action
```

Spark RDD API

ADALTAS

When an **action** is run:

- Spark builds a **Directed Acyclic Graph** (DAG) of **stages**
- **1 stage = X tasks** (1 by RDD partition)
- Tasks are sent to **executors**



Spark + RDD: full recap

ADALTAS

1. Spark **creates driver + executors**
2. Spark transforms your **code** into **stages** (DAG)
3. Each **executor** gets **partitions** of the RDD
4. For each **stage**, the **driver** sends a **task** to each **executor** to run on each **partition**

RDDs: Pros and Cons

ADALTAS

- For user:
  - complicated to express complex ideas
  - difficult to understand the code
- For Spark: lambda functions are opaque (no optimization)
+ Developers: low level control of execution

RDDs vs DataFrames: code

ADALTAS

```python
# In Python
# Create an RDD of tuples (name, age)
dataRDD = sc.parallelize([("Brooke", 20), ("Denny", 31), ("Jules", 30),
  ("TD", 35), ("Brooke", 25)])
# Use map and reduceByKey transformations with their lambda
# expressions to aggregate and then compute average

agesRDD = (dataRDD
  .map(lambda x: (x[0], (x[1], 1)))
  .reduceByKey(lambda x, y: (x[0] + y[0], x[1] + y[1]))
  .map(lambda x: (x[0], x[1][0]/x[1][1])))
```

How to do it?

```python
# Create a DataFrame
data_df = spark.createDataFrame([("Brooke", 20), ("Denny", 31), ("Jules", 30),
  ("TD", 35), ("Brooke", 25)], ["name", "age"])
# Group the same names together, aggregate their ages, and compute an average
avg_df = data_df.groupBy("name").agg(avg("age"))
# Show the results of the final execution
avg_df.show()
```

```
+------+--------+
|  name|avg(age)|
+------+--------+
|Brooke|    22.5|
| Jules|    30.0|
|    TD|    35.0|
| Denny|    31.0|
+------+--------+
```

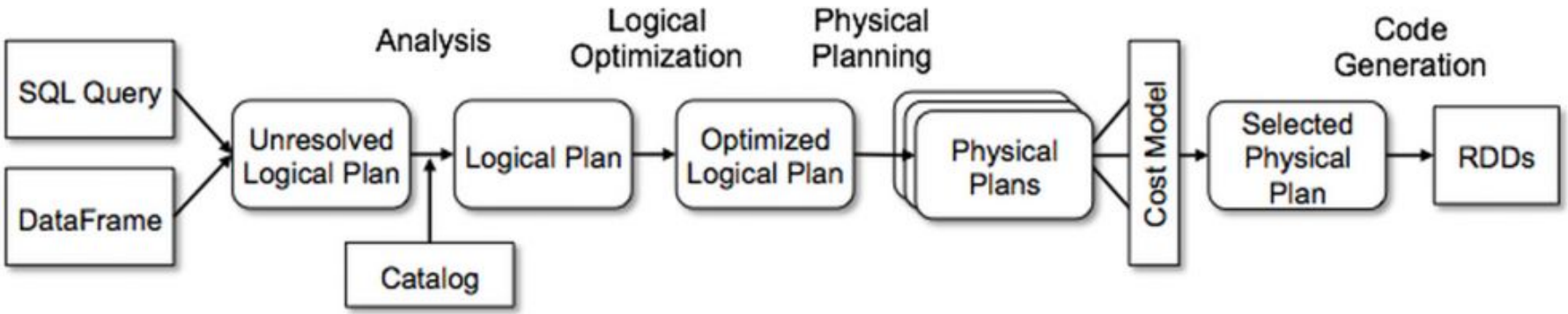
What to do?

Catalyst Optimizer

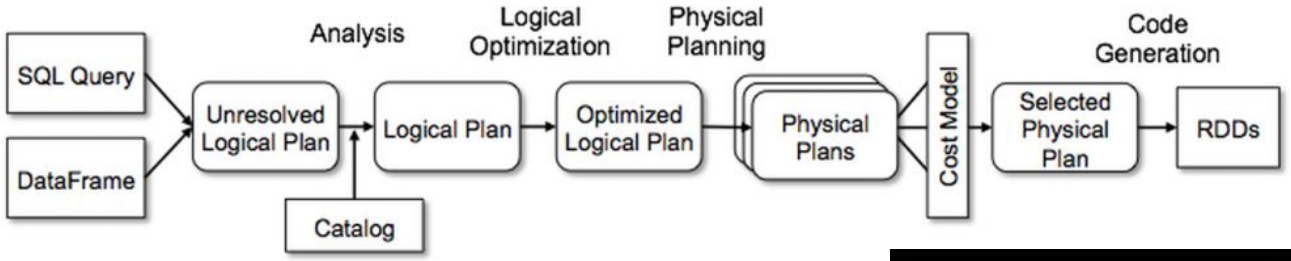
ADALTAS

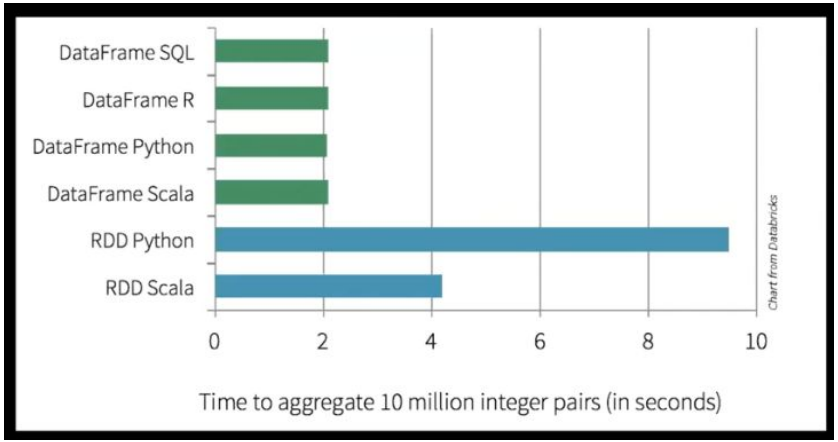
```
someRdd
        .reduceByKey(lambda x, y: ...)
        .filter(lambda x: ...)
```

```
someDF
        .groupBy("...")
        .filter(cond)
```



Catalyst Optimizer





DataFrames

ADALTAS

- Structured dataset:
  - In-memory, distributed tables
  - Named and typed columns: schema
  - Collection of `Row`s
- Sources available: structured files, Hive tables, RDBMS (MySQL, PostgreSQL, …), RDDs
- High-level APIs

DataFrames

ADALTAS

Querying DataFrames:

- By chaining functions
- By writing standard SQL strings

Why SQL?

ADALTAS

- Around since the 70s
- Huge enterprise usage:
  - Lots of users
  - Lots of projects
- But: cannot be used for ML or graph analyses