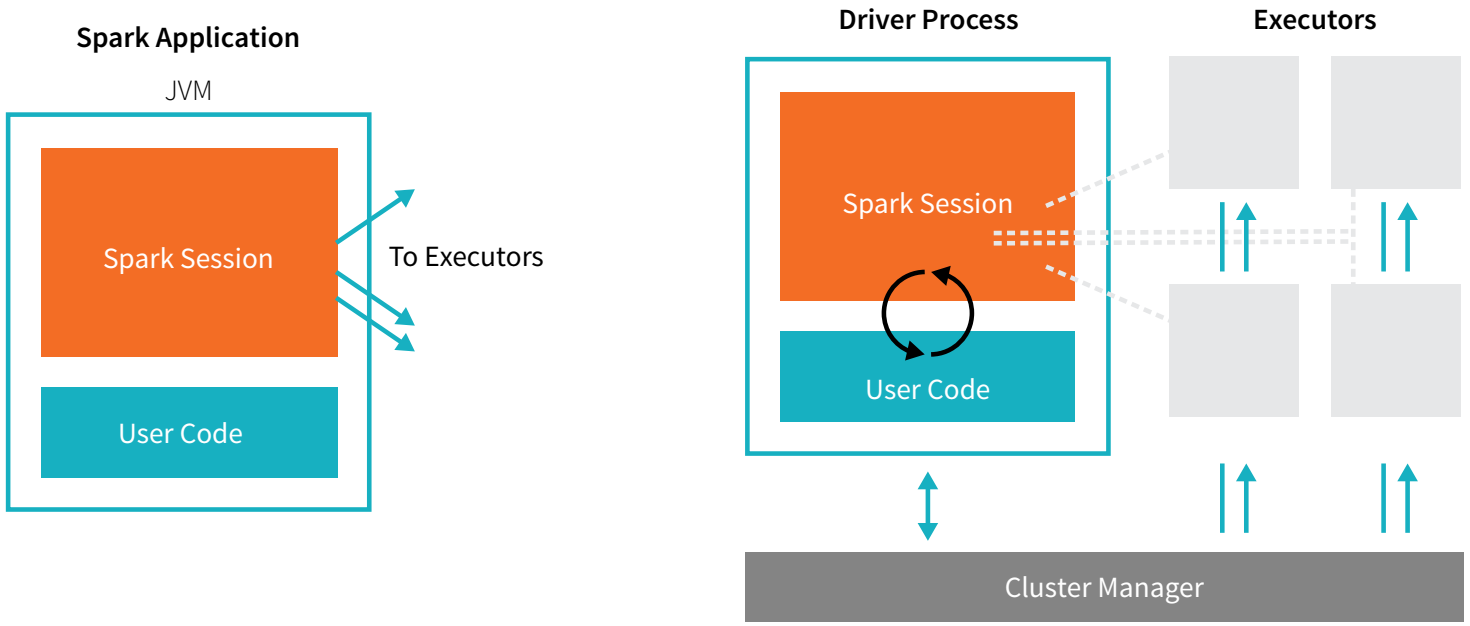
Spark SQL Tutorial

Some theory ☺

What is an RDD?

- Dependencies
- Partitions (with optional locality info)
- Compute function: Partition => Iterator [T]

Brief recap: Spark Execution



dbTrento

Why structured collections ?

Many datasets have a fixed structure

- JSON
- CSV
- DATABASES

RDDs : Imperative API

MAP and REDUCE operations specify how but not what

Single RDD operations are not optimizable

exampleRDD.map(lambda x: x *x);

The order of the operations matters

RDD1.join(RDD2).filter(…);

DataFrames and Datasets

Two different notions of structured collections:

- DataFrames
- Datasets

Distributed table-like collections

- well defined rows and columns

Immutable and lazy evaluated plans

- Conceptually similar to RDDs
- More complex and optimizable

Schemas + Rows and Columns

Definition of:

- Column names
- Types

Spark is effectively a programming language of its own

Internal used of **Catalyst:**

- Maintenance of its own type information
- Planning & preprocessing

Seems overkill BUT it opens a lot of possibilities for optimizations

Structured APIs of different languages (Java, Scala, Python, R) are using Spark types

Columns and Rows

Column

- Simply type: Integer or String
- Complex type: array, map
- Can contain *null values*
- Transformations

Row

- Record of data
- Type Row
- Different sources: SQL, RDDs, other data sources or manually created

Spark Types

Data type	*Value type in Scala*	*API to access or create a data type*
ByteType	Byte	ByteType
ShortType	Short	ShortType
IntegerType	Type	IntegerType
LongType	Long	LongType
FloatType	Float	FloatType
DoubleType	Double	DoubleType
DecimalType	java.math.BigDecimal	DecimalType
StringType	String	StringType
BinaryType	Array [Byte]	BinaryType
BooleanType	Boolean	BooleanType
TimestampType	java.sql.Timestamp	TimestampType
DateType	java.sql.Date	DateType
ArrayType	scala.collection.Seq	ArrayType(elementType, [containsNull]) Note: The default value of containsNull is true.
MapType	scala.collection.Map	MapType (keyType, valueType, [valuecontainsNull]) Note: The default value of valuecontainsNull is true.
StructType	org.apache.spark.sql.Row	StructType (fields) Note: fields is a Seq of StructFields. Also, two fields with the same name are not allowed.
Struct Field	The value type in Scala of the data type of this field (For example, Int for a StructField with the data type IntegerType)	StructField(name, dataType, [nullable]) Note: The default value of nullable is true.

DataFrame vs Datasets

Two more APIs

- "untyped" DataFrames
- "typed" Dataset

What does it mean "untyped"?

- There is a type
- It's maintained by Spark
- check those types at *runtime*

Datasets with types conform at *compile time*

Datasets are available on JVM based languages: Scala and Java

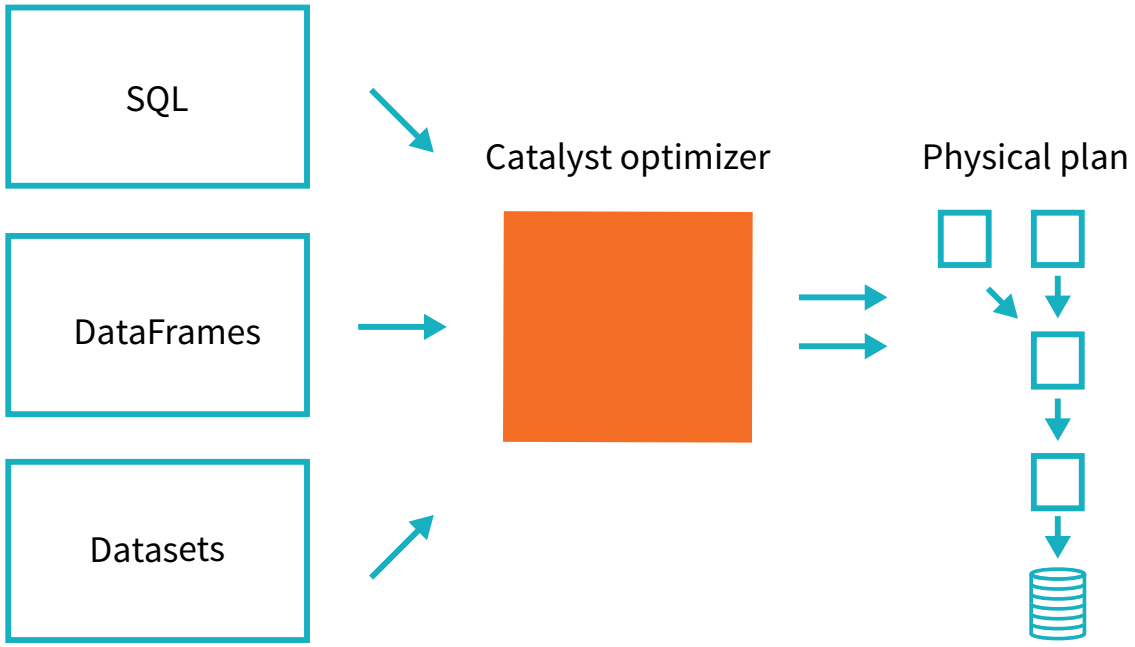
DataFrames are simply Datasets of Types ***Row***

Overview of Structured API Execution

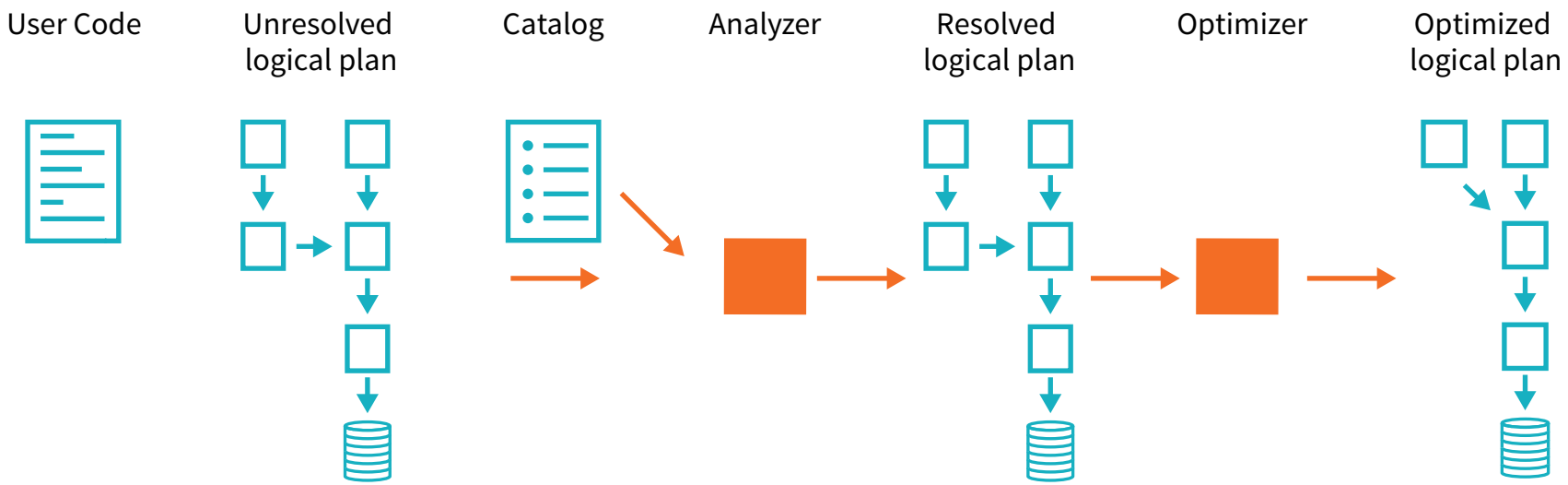
Steps

1. Write DataFrame/Dataset/SQL Code
2. If valid code, Spark converts this to a *Logical Plan*
3. Spark transforms this *Logical plan* to a *Physical Plan*, checking for optimization along the way
4. Spark then executes this *Physical Plan* (RDD manipulations) on the cluster

Overview of structured API Execution (1)

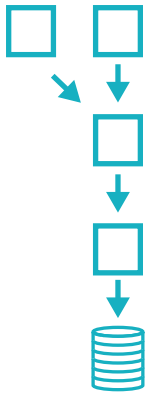


Overview of structured API Execution (2)



Overview of structured API Execution (3)

Optimized logical plan

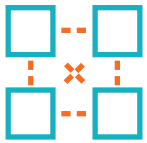


Physical plans

Cost model

Best physical plan

Executed on cluster



Basic structured operations

DataFrame consists of a series of records

- Row type
- Columns

Schemas define the **name** and the type of each **columns**

Dataframe are partitioned on different machines

Example 1

```
# in Python
myRange = spark.range(1000).toDF("number")
```

Transformation

```
divisBy2 = myRange.where("number % 2 = 0")
```

Action

```
divisBy2.count()
```

End-to-End Example

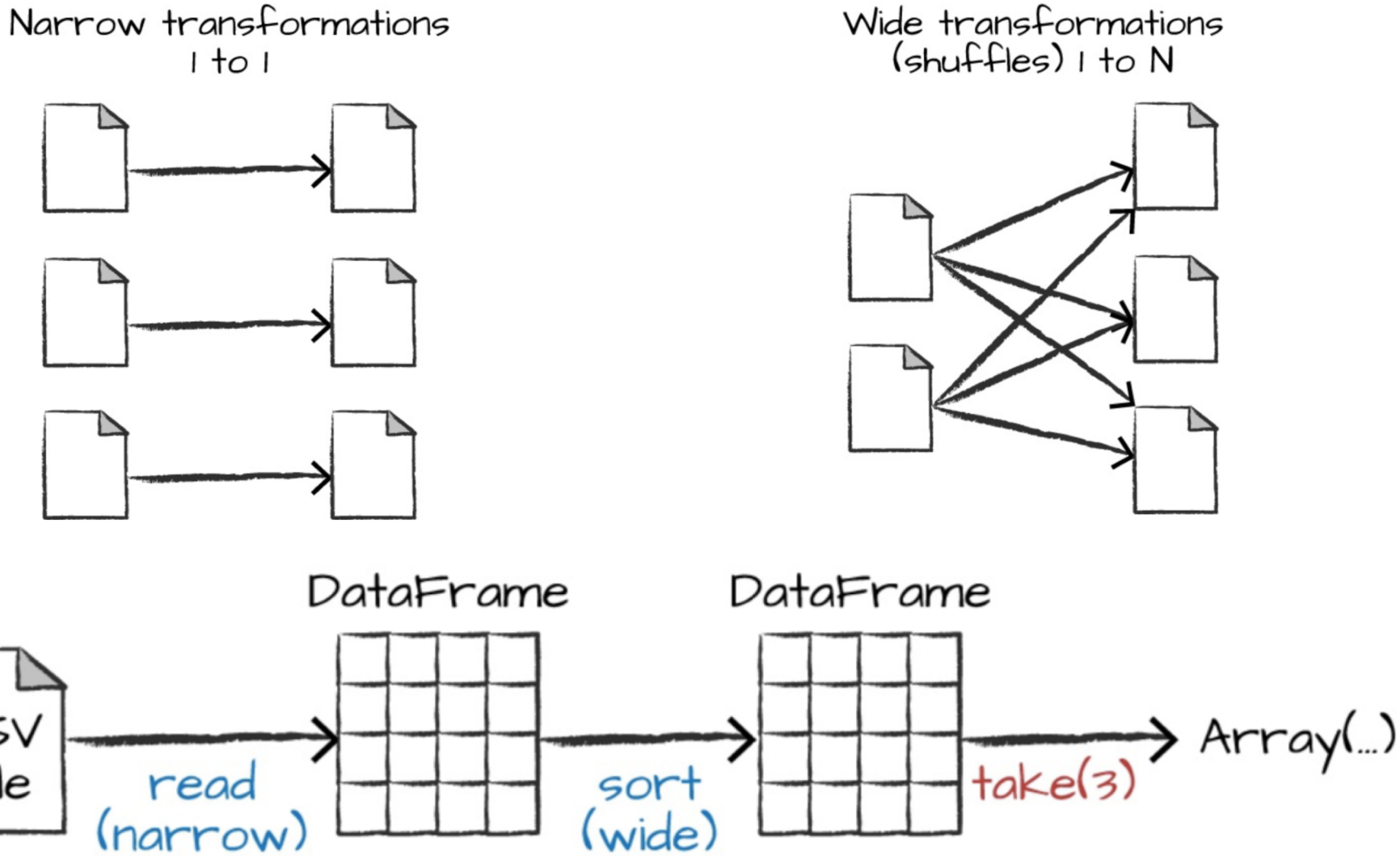
```
$ head /data/flight-data/csv/2015-summary.csv

DEST_COUNTRY_NAME,ORIGIN_COUNTRY_NAME,count
United States,Romania,15
United States,Croatia,1
United States,Ireland,344
```

End-To-End Example


```
flightData2015.take(3)
```

```
Array([United States,Romania,15], [United States,Croatia...
```

Wide and Narrow Transformations



Basic structured operations: SCHEMA

```
# in Python
flightData2015 = spark\
  .read\
  .option("inferSchema", "true")\
  .option("header", "true")\
  .csv("/data/flight-data/csv/2015-summary.csv")
```

RETURNS

```
StructType(List(StructField(DEST_COUNTRY_NAME,StringType,true),
StructField(ORIGIN_COUNTRY_NAME,StringType,true),
StructField(count,LongType,true)))
```

Basic structured operations: SCHEMA

```
from pyspark.sql.types import StructField, StructType, StringType, LongType

myManualSchema = StructType([
  StructField("DEST_COUNTRY_NAME", StringType(), True),
  StructField("ORIGIN_COUNTRY_NAME", StringType(), True),
  StructField("count", LongType(), False, metadata={"hello":"world"})
])
df = spark.read.format("json").schema(myManualSchema)\
  .load("/data/flight-data/json/2015-summary.json")
```

Columns

```
from pyspark.sql.functions import col, column
col("someColumnName")
column("someColumnName")
```

```
df.col("count")
```

Expressions

An expression is a set of transformations

```
from pyspark.sql.functions import col, expr
```

Columns as expressions

expr("someCol - 5") SAME as col("someCol") - 5 SAME as expr("someCol") - 5

Not so easy at the beginning but **REMEMBER**:

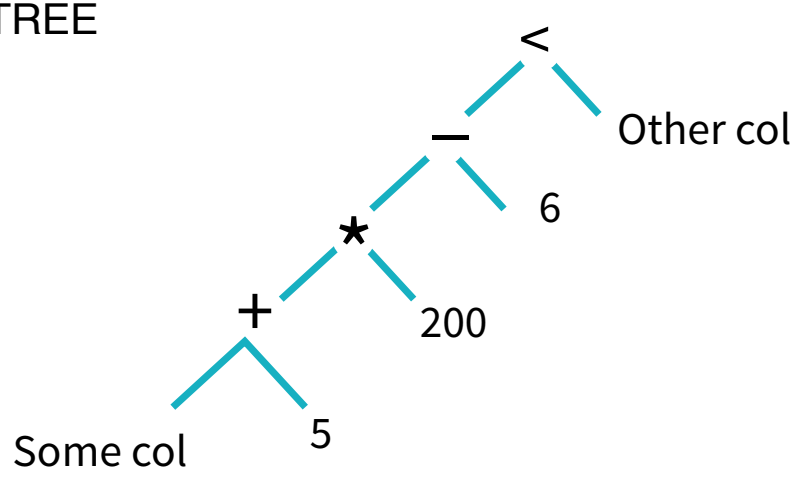
1. Columns are just expressions

2. Columns and transformations of those column compile to the same logical plan as parsed expressions.

Expression example

```
(((col("someCol") + 5) * 200) - 6) < col("otherCol")
```

LOGICAL TREE



CODE:

```
from pyspark.sql.functions import expr

expr("(((someCol + 5) * 200) - 6) < otherCol")
```

Dataframe Operations

Retrieve columns

```
spark.read.format("json")
.load("/mnt/defg/flight-data/json/2015-summary.json")
.columns
```

Create rows

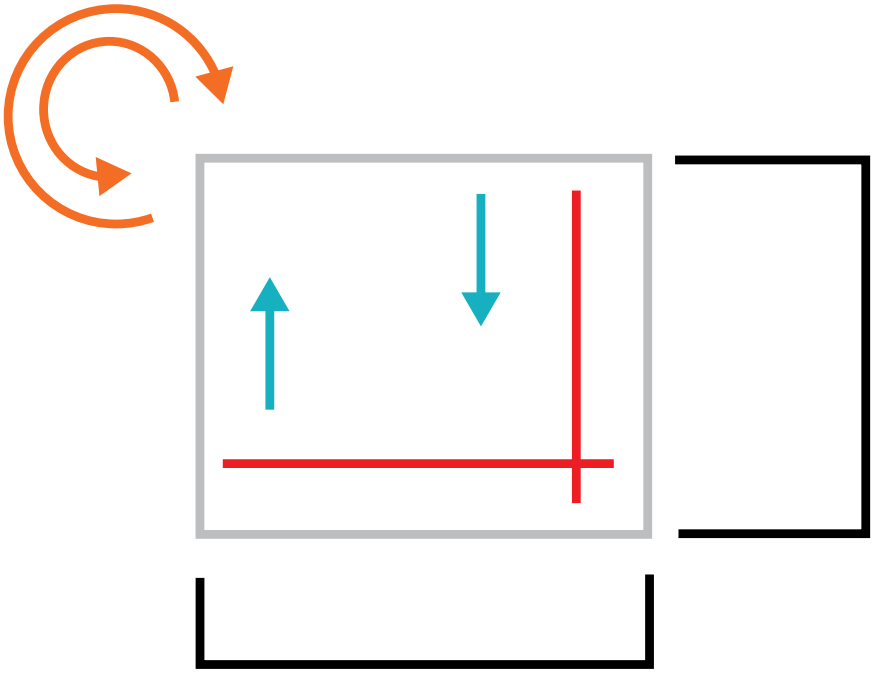
```
from pyspark.sql import Row
myRow = Row("Hello", None, 1, False)
```

Accessing rows

```
%scala
myRow(0) // type Any
myRow(0).asInstanceOf[String] // String
myRow.getString(0) // String
myRow.getInt(2) // String
```

```
# in Python
myRow[0]
myRow[2]
```

DataFrame Transformations



- Remove columns or rows
- Row into a column
  or column into a row
- Add rows or columns
- Sort data by values in rows

Create DataFrames From File

```
df = spark.read.format("json")\
   .load("/mnt/defg/flight-data/json/2015-summary.json")

df.createOrReplaceTempView("dfTable")
```

Create Dataframe by hand

```
from pyspark.sql import Row
from pyspark.sql.types import StructField, StructType, StringType, LongType
myManualSchema = StructType([
  StructField("some", StringType(), True),
  StructField("col", StringType(), True),
  StructField("names", LongType(), False)
])
myRow = Row("Hello", None, 1)
myDf = spark.createDataFrame([myRow], myManualSchema)
myDf.show()
```

```
+-----+----+-----+
| some| col|names|
+-----+----+-----+
|Hello|null|    1|
+-----+----+-----+
```

Select and SelectExpr (1)

```sql
-- in SQL
SELECT * FROM dataFrameTable
SELECT columnName FROM dataFrameTable
SELECT columnName * 10, otherColumn, someOtherCol as c
       FROM dataFrameTable
```

```python
# in Python
sqlWay = spark.sql("""
SELECT DEST_COUNTRY_NAME, count(1)
FROM flight_data_2015
GROUP BY DEST_COUNTRY_NAME
""")

dataFrameWay = flightData2015\
  .groupBy("DEST_COUNTRY_NAME")\
  .count()
```

Select and SelectExpr (1)

```
spark.sql("SELECT max(count) from flight_data_2015").take(1)
```

```
# in Python
from pyspark.sql.functions import max

flightData2015.select(max("count")).take(1)
```

Select and SelectExpr (2)

```python
# in Python
df.select("DEST_COUNTRY_NAME").show(2)
```

```sql
-- in SQL
SELECT DEST_COUNTRY_NAME FROM dfTable LIMIT 2
```

```
+-----------------+
|DEST_COUNTRY_NAME|
+-----------------+
|    United States|
|    United States|
+-----------------+
```

```python
# in Python
maxSql = spark.sql("""
SELECT DEST_COUNTRY_NAME, sum(count) as destination_total
FROM flight_data_2015
GROUP BY DEST_COUNTRY_NAME
ORDER BY sum(count) DESC
LIMIT 5
""")
```

```
+-----------------+-----------------+
|DEST_COUNTRY_NAME|destination_total|
+-----------------+-----------------+
|    United States|           411352|
|           Canada|             8399|
|           Mexico|             7140|
|   United Kingdom|             2025|
|            Japan|             1548|
+-----------------+-----------------+
```

```python
# in Python
from pyspark.sql.functions import desc

flightData2015\
  .groupBy("DEST_COUNTRY_NAME")\
  .sum("count")\
  .withColumnRenamed("sum(count)", "destination_total")\
  .sort(desc("destination_total"))\
  .limit(5)\
  .show()
```

dbTrento

Datasets

We have to give the types information

- The types are virtual
- Same APIs of the DataFrame
- JavaBeans or StructType

Encoders

- Optimize memory space
- In Java you need to specify the types!

DataFrame == Dataset<Row>

Dataset Example

```
public static class Person implements Serializable {
  private String name;
  private int age;

  public String getName() {    return name;  }
  public void setName(String name) {    this.name = name;  }
  public int getAge() {    return age;  }
  public void setAge(int age) {   this.age = age;  }
}

// Encoders are created for Java beans
Encoder<Person> personEncoder = Encoders.bean(Person.class);
Dataset<Person> javaBeanDS = spark.createDataset( personList, personEncoder);
Dataset<Person> people = spark.read().parquet("...").as(Encoders.bean(Person.class));
```

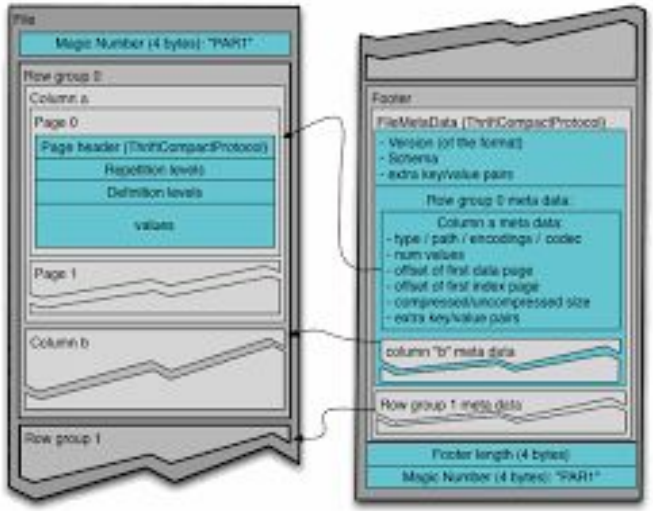
SparkSession

Provides methods and attribute for SQL

- **read** – create un DataFrame from a supported source
- **sql(STRING query)** – query data throught SQL
- **catalog** – contains information about database and table
- **createDataFrame()** – create a DataFrame from an rdd given a defined schema

Parquet

- Column storage
- Each .parquet file is a directory
- Supported by Spark
- Optimized for Big Data



dbTrento

Structure

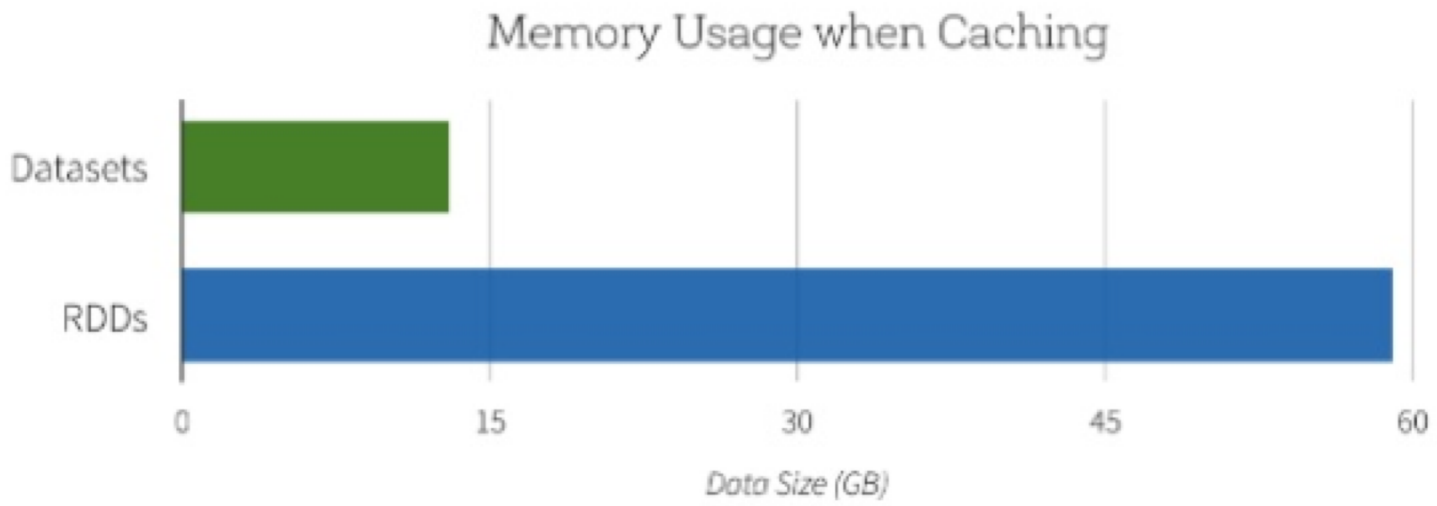
Why?

A STRUCTURE limits what we can express

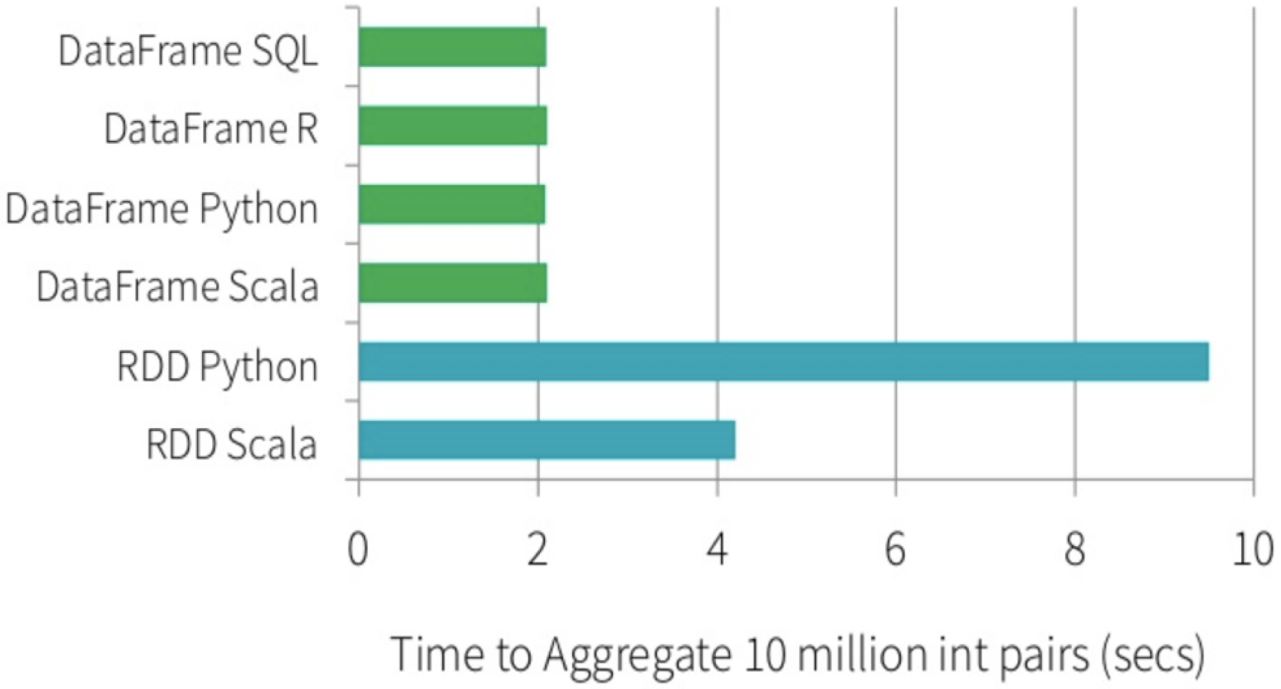
It can accommodate the vast majority of computations

- **Limiting the space of what can be expressed enables optimizations!**

Performance (1)



Performance (2)



Code Repository

https://github.com/forons/BigDataExamples

Spark tutorials links

Basic video tutorial:

https://www.youtube.com/playlist?list=PLTPXxbhUt-YWSgAUhrnkyphnh0oKIT8-j

Official tutorial:

https://spark.apache.org/docs/latest/rdd-programming-guide.html
https://spark.apache.org/docs/latest/sql-programming-guide.html

Helpful links:

https://github.com/databricks/learning-spark

http://spark.apache.org/examples.html

Contacts

For any problem, send a mail to

daniele.foroni@unitn.it