THE DATA SCIENCE LAB
Spark Data Frames

COM 490

Agenda 2025 – module 3b

19.02	Introduction to Data Science with Python
26.02	(Bigger) Data Science with Python
05.03	Introduction to Big Data Technologies
12.03	Big Data Wrangling with Hadoop
19.03	Introduction to Spark
26.03	Advanced Big Data Queries
02.04	Spark Data Frames
09.04	Advanced Spark
16.04	Introduction to Stream Processing
30.04	Stream Processing with Kafka
07.05	Advanced Stream Processing
14.06	Final Project Q&A
21.05	Final Project Due
28.05	Oral Sessions

Week 5 Recap

- What is Spark?
- RDDs
  - Immutable
  - Resilient
  - Lineage
- Operations on RDDs
  - Transformations
  - Actions
  - Lazy execution
- Exercises to get started using the Guttenberg corpus

Spark & RDDs – Questions?

Today’s Agenda

- Introduction to Data Frames
- DataFrames and PySpark under the hood
- Exercises week 7
  - Guttenberg corpus

Introduction to Spark DataFrames

RDD Revisited

- Resilient → lineage
- Distributed → partitions
- Unstructured → key-row pairs
- Type safe
  - Scala’s compiler optimization
- Use of lambda functions
- Fine grained control – tell spark how to transform a data
  - low level – more responsibility to the programmer:
    - decide transformations and actions
    - which part of the data
    - in what order

Spark DataFrames

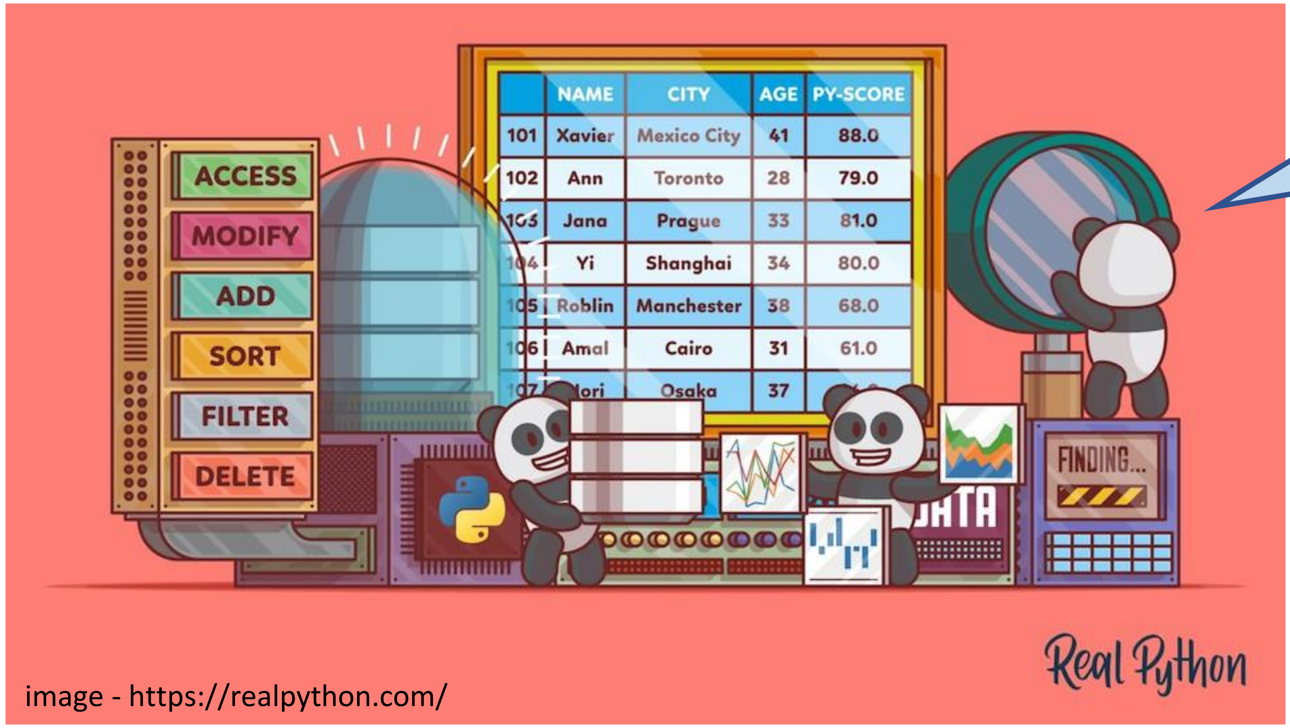
- Distributed Collections of Data
  - Organized into rows of named columns
  - Very much like relational database Tables
  - Optimized for relational-type of queries on tables (logical plan optimization)

Structured

Col 1	Col 2	Col 3
1	a	10:00
2	b	11:00
3	c	12:00
4	d	13:00
5	e	14:00

Origin of Data Frames

Spark Data Frame API Inspired by R and Python's Pandas



	NAME	CITY	AGE	PY-SCORE
101	Xavier	Mexico City	41	88.0
102	Ann	Toronto	28	79.0
103	Jana	Prague	33	81.0
104	Yi	Shanghai	34	80.0
105	Robin	Manchester	38	68.0
106	Amal	Cairo	31	61.0
107	Nori	Osaka	37	[illegible]

image - https://realpython.com/

want to join the Big Data party!

What are Spark Data Frames

- Inspired by R and Python Pandas

SOURCE

- Local File Systems
- Distributed File Systems (HDFS)
- Cloud Storage (S3)
- External data bases
- Spark RDD

DATA FORMAT – out of the box

- TEXT
- JSON
- CSV
- Parquet
- ORC
- Hive Table

+ **Other with plugins** (Avro, ElasticSearch, Cassandra, ...)

- Parallelism & query optimizer, unlike R and Python

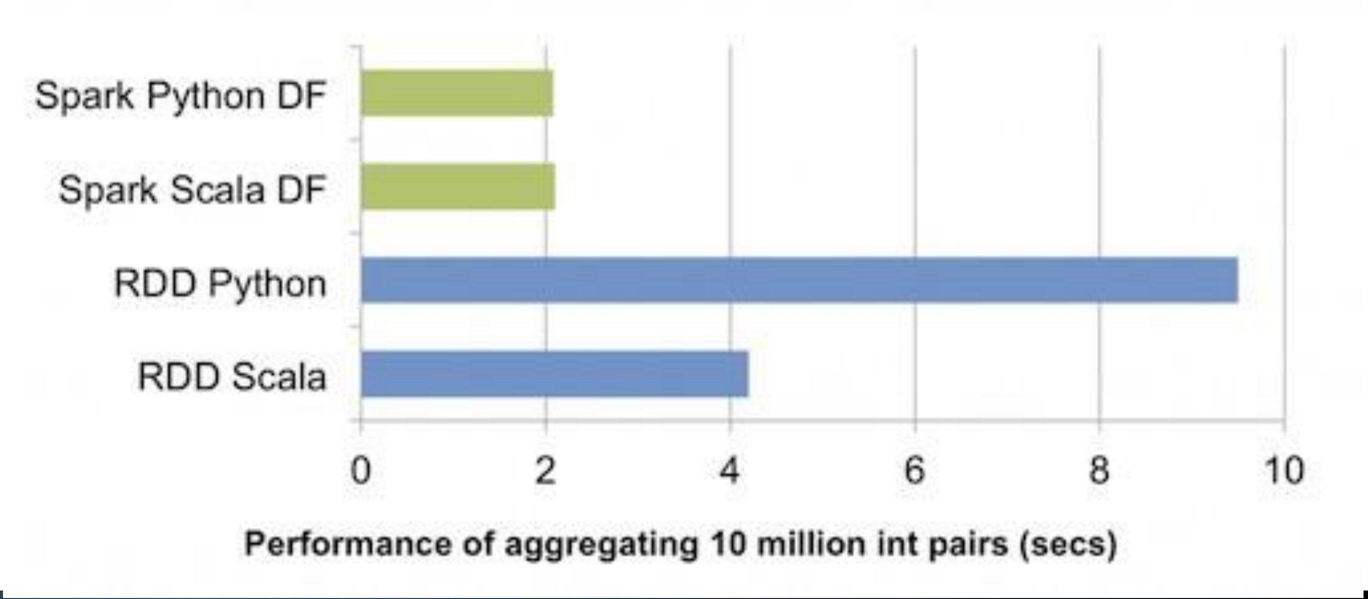
Why Spark Data Frames – RDD vs DataFrame

Resilient Distributed Dataset (RDD)	**Data Frame**
Structured & unstructured data	Structured data (table, named columns)
Schema must be declared manually	Auto discovery of file schema
Lambda functions (map, reduce)	Declarative, almost as SQL queries
Lower level language	Higher level language
No built-in other than generic compiler optimization. Must be done manually	Execution optimization
Type safety at compile time	Type safety at run-time (e.g. trying to access a non-existing column)

Spark DataFrame performance

- DataFrame's data is managed off-JVM more optimal
  - No need for Java/Scala (de)serialization when accessing object
  - Avoid garbage collection
- Aggregation (group by) is harder and not as efficient with RDD.

In comparison, exploration analysis is quick and easier on large DataFrame



databrkick blog 2015
(*) to be taken with a grain of salt

Which one should I use? RDD vs DataFrame

- Use RDD for operations that require:
  - low level functionalities
  - control on unstructured data

- Use DataFrame for:
  - high level (SQL like) operations
  - on structured data

DataFrame under the hood

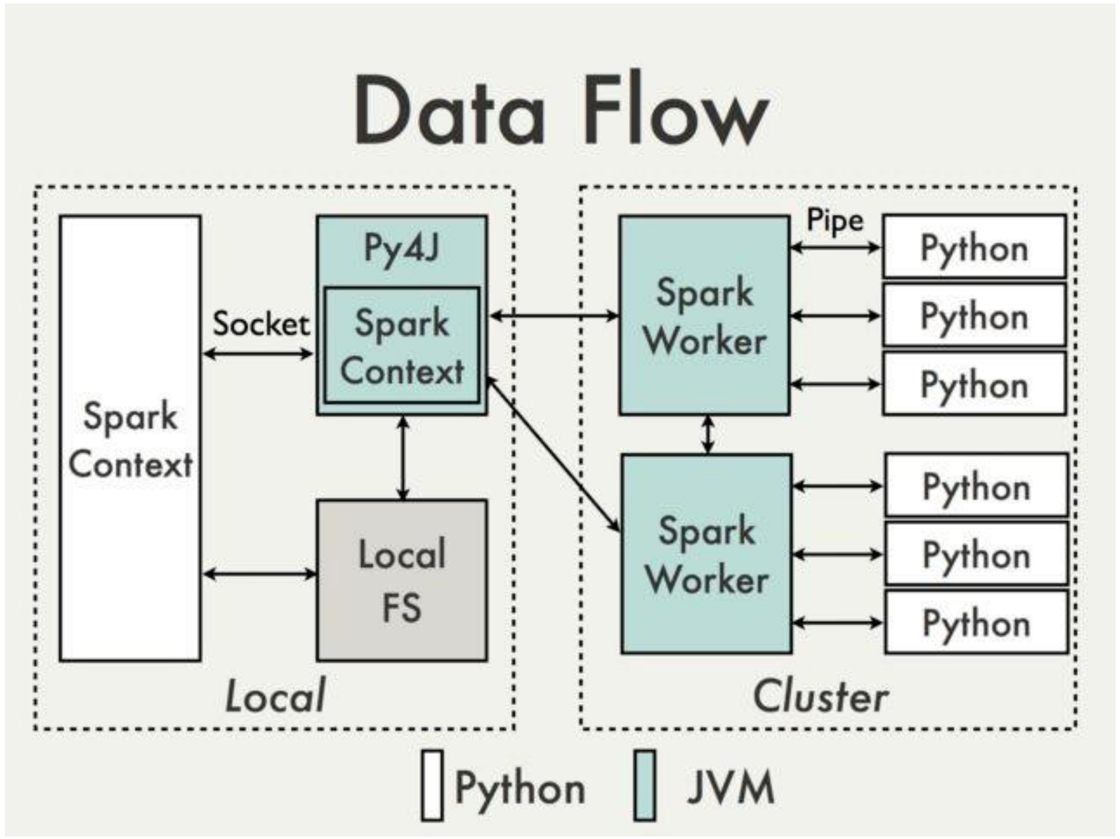
PySpark

- PySpark - Python front end API for Spark

- Interface RDDs with Python
- Py4J – python library to dynamically access JVM objects
- Compatible with
  - PySparkSQL – SQL query library for DataFrame
  - MLib – Machine learning library
  - GraphFrames – Graph processing based on DataFrames (Graphx is on RDDs)

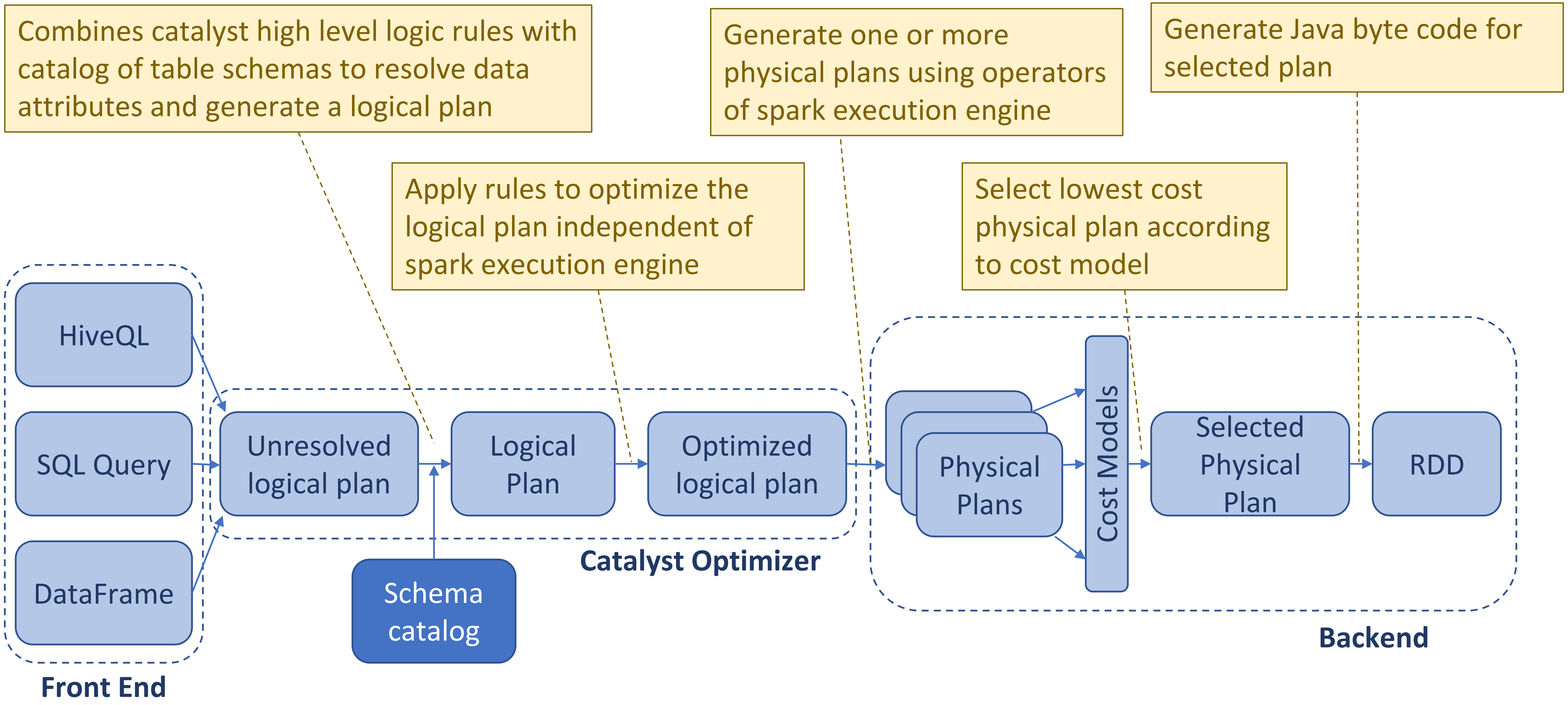
PySpark

- Spark workers pull data from source into JVM
- Data is actually processed into python subprocesses
- (de)serialization and streaming at every step



PySpark internal

Catalyst Optimizer


Combines catalyst high level logic rules with catalog of table schemas to resolve data attributes and generate a logical plan
Generate one or more physical plans using operators of spark execution engine
Generate Java byte code for selected plan
Apply rules to optimize the logical plan independent of spark execution engine
Select lowest cost physical plan according to cost model
HiveQL
SQL Query
DataFrame
Front End
Unresolved logical plan
Logical Plan
Optimized logical plan
Catalyst Optimizer
Schema catalog
Physical Plans
Cost Models
Selected Physical Plan
RDD
Backend

A parting word on Spark DataSets

- DataSet = extensions of DataFrame with convenience of RDD.
  - Strong type safety
  - RDD with Spark SQL optimized execution engine
  - Operate on serialized data (no deserialization overhead)
- Available only on Scala and Java
  - Since 1.6 DataFrame on Scala and Java are alias for DataSet[row]

Useful References

- Spark docs http://spark.apache.org/docs/latest
- DataFrames and code generation https://medium.com/virtuslab/spark-sql-under-the-hood-part-i-26077f85ebf0
- Python Spark DataFrames starter documentation https://spark.apache.org/docs/latest/api/python/getting_started/quickstart_df.html
- Spark MLlib guide https://spark.apache.org/docs/latest/ml-guide.html

Start your engines

https://com490-2024.epfl.ch

https://dslabgit.datascience.ch/course/2025/module-3b

(Fork and git clone)