Investigating Apache Spark for Physics Analysis

Luca Canali
CERN IT, Spark and Analytics Service
May 2022

Speaker

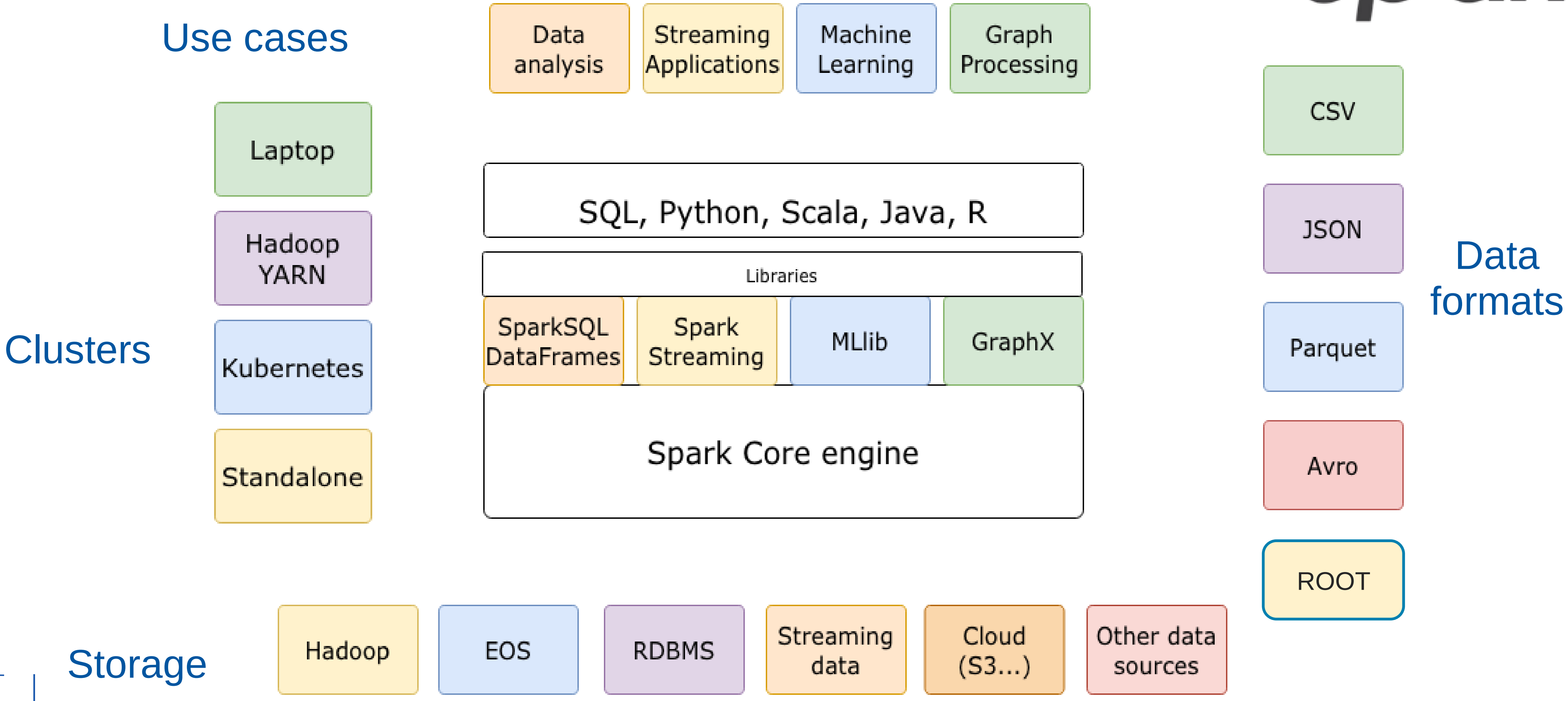
- Short intro: Luca Canali
  - Data engineer at CERN IT
  - Working with Spark and Hadoop/Big Data services in IT-DB/IT-DA since its start in 2016.
    - Previously participated in the CMS Big Data Project https://cms-big-data.github.io/
    - Contributed (minor features/patches) to Apache Spark
  - Oracle DBA at CERN, since 2005. Working with IT and ATLAS DBA teams.

Motivations and Scope

- Context: Spark service at CERN
- Recent work on Spark Arrow UDF + work on implementing example analyses using PySpark on Jupyter notebooks
  - Blog: Can High Energy Physics Analysis Profit from Apache Spark APIs? https://db-blog.web.cern.ch/node/186

- I will mix an intro to Spark with notebooks examples
- Final thoughts on what I believe works OK with this approach and what needs improvements
- Not a goal: compete with state-of-the art software for analysis

Apache Spark Ecosystem



Apache Spark Adoption

- Who is using Spark, how, and why?
- Databricks
  - They sell a cloud-based analytics platform, centered around Spark
    - Development in the Spark ecosystem (Data Lakehouse, MLFlow)
    - They also have custom Spark improvements
  - Top contributors and drivers of the open-source development
- Cloud vendors
  - All offer user-facing “Big Data” platforms, typically including Spark
- Many tech giants use it internally: Facebook, Apple, MS, Baidu, Netflix, etc
  - Some contribute back to Spark development (Apache Spark PMCs)

Spark @ CERN

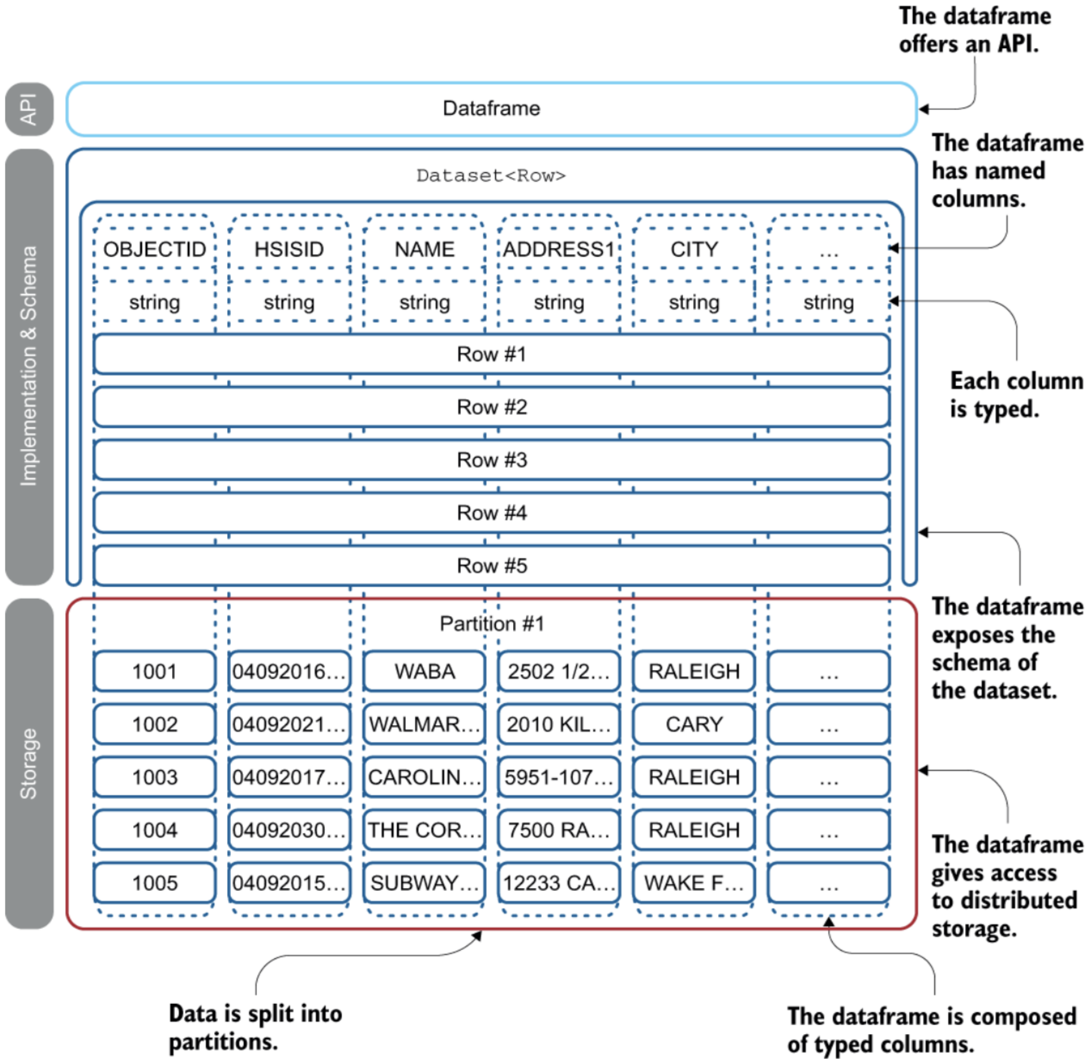
- Key component of the Hadoop platforms.
  - IT monitoring, IT security
  - Experiments computing data
  - Physics: RDataFrame, CMS Spark, CMS Muon POG
- We also provide Spark on Kubernetes clusters
- User-access
  - Notebooks, often via SWAN. Some use of Spark on R
  - Batch jobs: Python, Scala, Java
- NXCals platform
  - Critical logging system for the accelerator complex
  - Platform based on Hadoop, API based on Spark

Demo 1

- Can Apache Spark run basic physics analysis?
- Let’s start with a “Hello World!” example
- Dimuon mass spectrum analysis at

https://github.com/LucaCanali/Miscellaneous/tree/master/Spark_Physics

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

Data Formats: ROOT, Parquet

- Data format is key for performance
  - ROOT can be ingested by Spark using Laurelin by A. Melo
  - Uproot and Laurelin can also be used to convert from ROOT to Parquet

- Spark is optimized for Apache Parquet (and ORC too)
  - Columnar format, with encoding, compression, schema
  - Spark has a custom vectorized Parquet reader, for performance
  - Filter pushdown
    - Filters can be resolved at the Parquet level
    - Statistics of min/max and other metadata
    - Recently also bloom filters

DataFrame API Basics

- Selections and projection are easily scalable
- Filter operations naturally fit with the DataFrame API

```
# Apply filters to the input data
# - select only events with 2 muons
# - select only events where the 2 muons have opposite charge

df_muons = df_muons.filter("nMuon == 2").filter("Muon_charge[0] != Muon_charge[1]")
```

- Expressions and formulas

```
df_with_dimuonmass = df_muons.selectExpr("""
     sqrt(2 * Muon_pt[0] * Muon_pt[1] *
             ( cosh(Muon_eta[0] - Muon_eta[1]) - cos(Muon_phi[0] - Muon_phi[1]) )
        ) as Dimuon_mass""")
```

Histograms with the DataFrame API

- Operation with data aggregation/shuffle
- Implemented using the *width_bucket* function
  - Works on many SQL engines

```
histogram_data = (
    df_with_dimuonmass
        .selectExpr(f"width_bucket(Dimuon_mass, {min_val}, {max_val}, {num_bins}) as bucket")
        .groupBy("bucket")
        .count()
        .orderBy("bucket")
    )

# convert bucket number to the corresponding dimoun mass value
histogram_data = histogram_data.selectExpr(f"round({min_val} + (bucket - 1/2) * {step},2) as value", "count as N_events")
```

Actions and Transformations

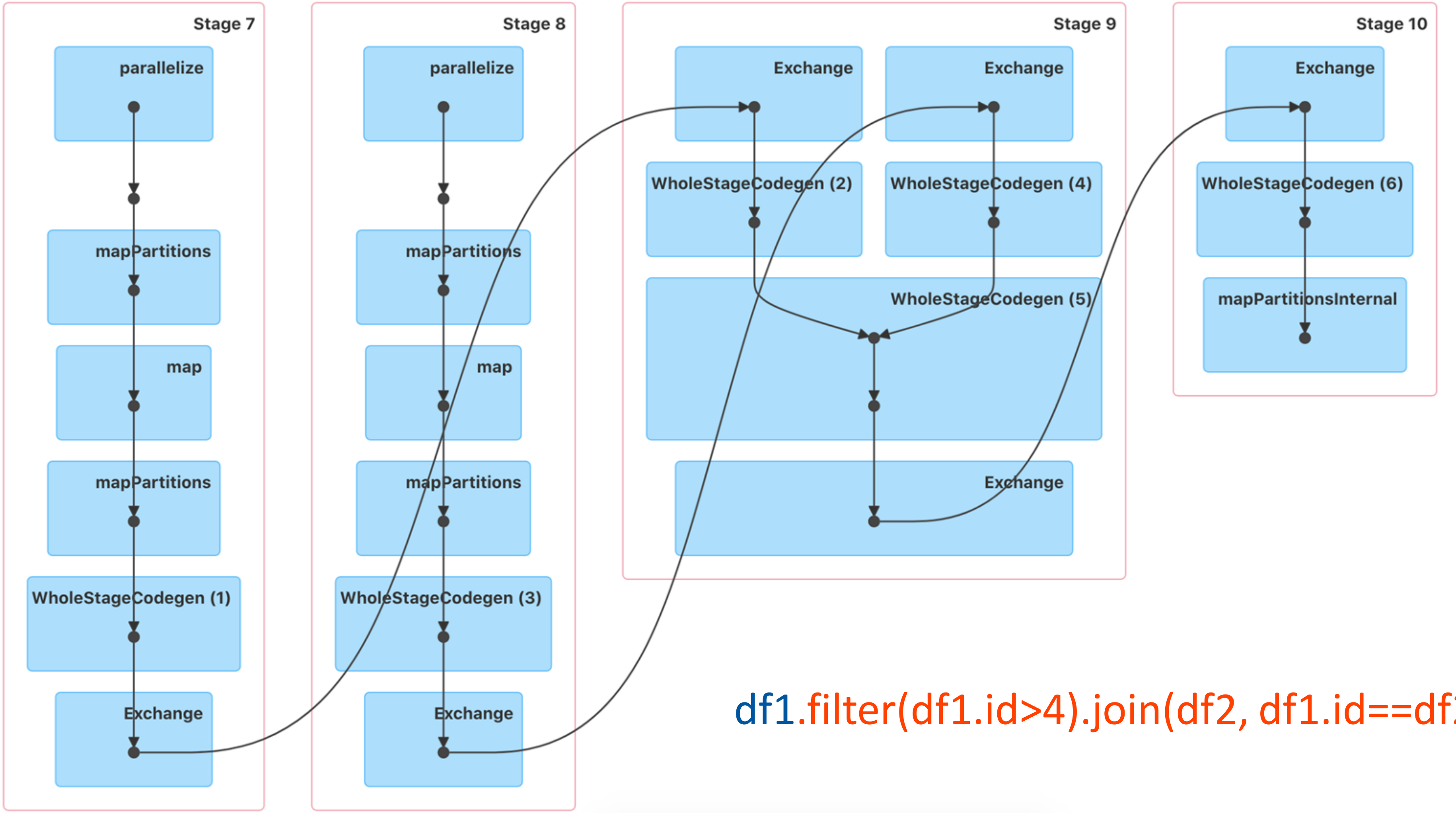
- Two types of operations on DataFrames:
  - **Transformations**: transform a DF in another one:
    - filter, select, …
  - **Actions**: trigger computation and return value
    - collect, toPandas, …

- **Lazy evaluation** and **immutability**:
  - Spark parses and optimizes only when an action is requested
    - You can express DataFrame transformations using many steps, for readability
  - Fault tolerance
    - the transformations can be replayed on the original DF (or on some of its partitions)

Spark Actions and DAG

- Invoking an action creates a job which is then divided in stages and executed by tasks.
  - Spark defines the computation using graphs (DAG).
  - Operations are grouped in stages.
  - Uses map-shuffle-reduce operations.

Execution DAG From the WebUI

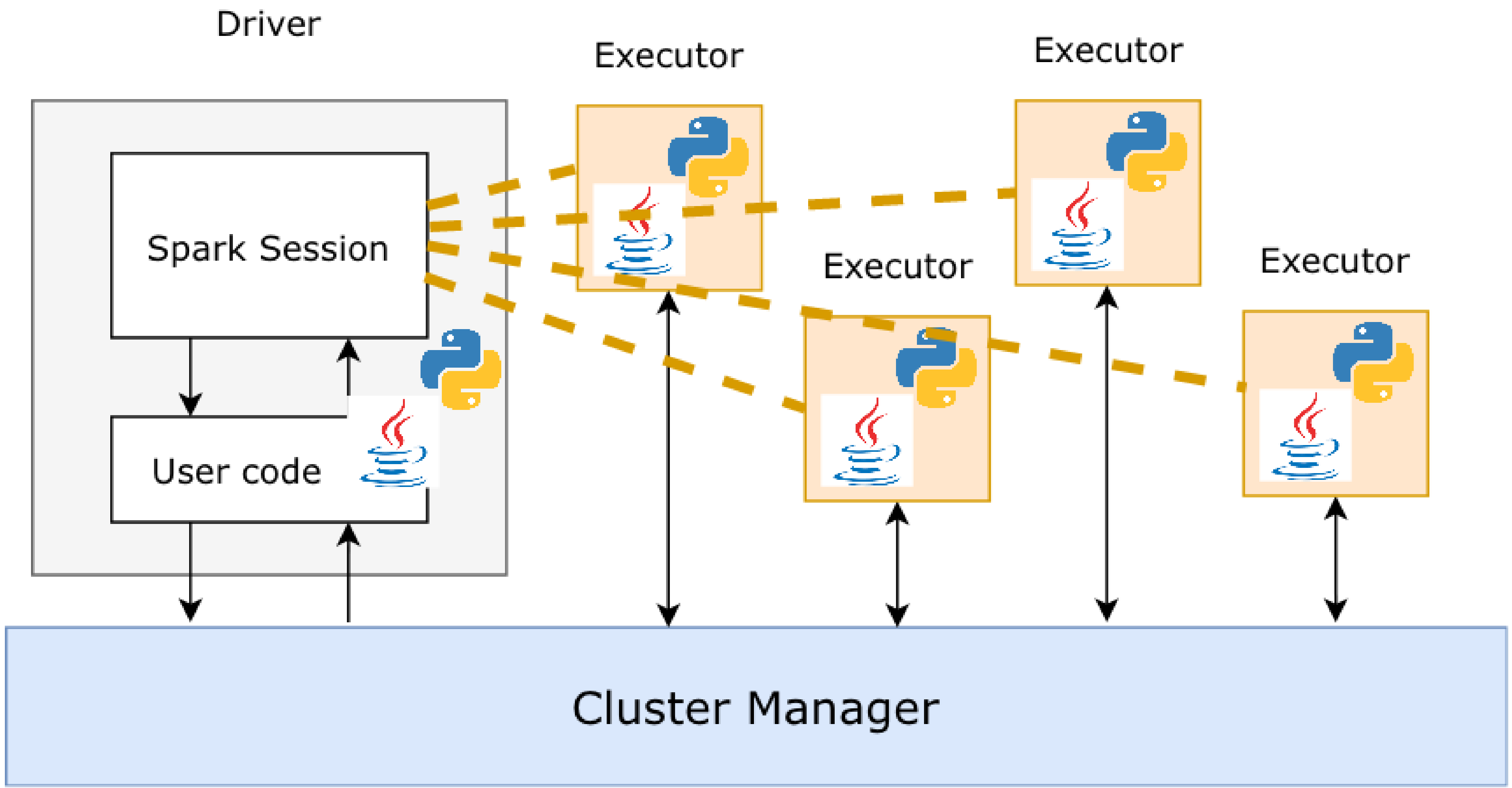


df1.filter(df1.id>4).join(df2, df1.id==df2.id).count()

Spark Tasks and Executors

- Tasks are the units of parallelization and are run concurrently on the available executors.
- Executors are the scalable engines that run tasks
  - From local mode (laptop) to 1000s of executors
  - Executors are JVM instances
    - They have associated CPU cores, memory
  - Executors as containers
    - execution engines / clusters: K8S, YARN, Stand alone Spark cluster

How Spark Runs Jobs at Scale



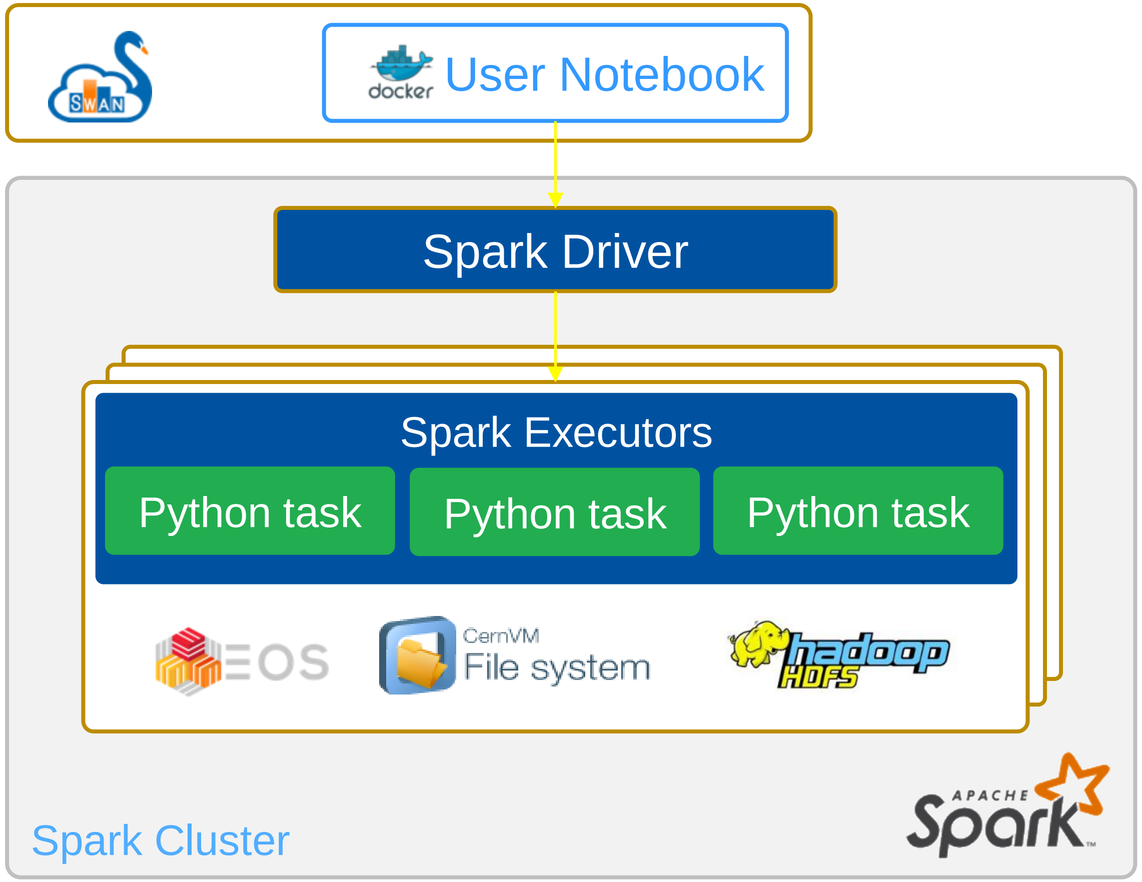
Apache Spark Clusters at CERN

- Spark running on clusters:
  - YARN/Hadoop -> established
  - Spark on Kubernetes -> growing adoption

Accelerator logging (part of LHC infrastructure)	Hadoop - YARN - 30 nodes (Cores - 1200, Mem - 13 TB, Storage – 7.5 PB)
General Purpose	Hadoop - YARN, 47 nodes (Cores – 2.0k, Mem – 25 TB, Storage – 16 PB)
Cloud containers	Kubernetes on Openstack VMs, Cores - 270, Mem – 2 TB Storage: remote HDFS or custom storage (CERN EOS, for physics data, S3 on Ceph also available). Note: GPU resources available.

SWAN Integration with Apache Spark

- SWAN service: https://swan.web.cern.ch/swan/
  - Notebooks for web-based analysis
- Integrated with CERN Spark Clusters
  - Reduces configuration complexity for users
- CERN software environment
  - Software from CVMFS
- Graphical Jupyter extensions developed
  - Spark Connector
  - Spark Monitor
- Access to Spark Clusters
  - NXCals: – Dedicated cluster for accelerator logging
  - Analytix: – General purpose YARN cluster
  - Cloud Containers: – General purpose Kubernetes cluster
  - Storage access: HDFS, EOS, S3



Demo 2

- Dimuon mass spectrum analysis on a cluster:

https://github.com/LucaCanali/Miscellaneous/tree/master/Spark_Physics

Spark can Handle Complex Schemas

- Example of complex schema for HEP

```
schema = "event LONG, HLT struct<flag1:boolean, flag2:boolean>, muons
ARRAY<STRUCT<pt:FLOAT, eta:FLOAT, phi:FLOAT, mass:FLOAT>>"

df.printSchema()

 |-- event: long (nullable = true)
 |-- HLT: struct (nullable = true)
 |    |-- flag1: boolean (nullable = true)
 |    |-- flag2: boolean (nullable = true)
 |-- muons: array (nullable = true)
 |    |-- element: struct (containsNull = true)
 |    |    |-- pt: float (nullable = true)
 |    |    |-- eta: float (nullable = true)
 |    |    |-- phi: float (nullable = true)
 |    |    |-- mass: float (nullable = true)
```

Handling Arrays

- Nested data is hard to handle
  - Does not fit naturally to DataFrame and SQL operations
- Available solutions in Spark
  - Array functions
    - Several available, example: array_min, array_sort, array_zip, …
  - "Explode" function
    - Transforms array values into DataFrame rows
  - Higher order functions
    - Process map/filter/aggregate on arrays elements (see next slide)
  - UDF: User-defined functions
    - General solution.
    - Spark UDF can be in Python or Scala

Spark Higher Order Functions

- Push filters, maps and reduce into arrays with higher order functions

```
# Array processing with Spark higher order functions in SQL
# Filter values > 38 from an array of numbers (temperature readings)

spark.sql("""
SELECT id, val, filter(val, t -> t > 38) as high
FROM temp_data""").show()

+---+----------------------------+--------+
|id |temp_celsius                |high    |
+---+----------------------------+--------+
|1  |[35, 36, 32, 30, 40, 42, 38]|[40, 42]|
|2  |[31, 32, 34, 55, 56]        |[55, 56]|
+---+----------------------------+--------+
```

Python UDF (User Defined Functions)

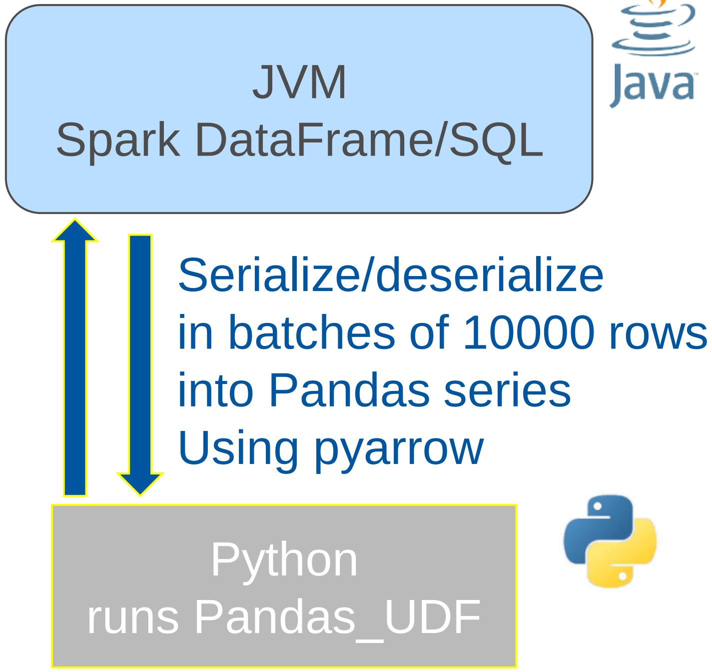
- Faster serialization (data movement Python - JVM)
- Send Pandas series to Python UDF for “bulk processing”

```
import pandas as pd
from pyspark.sql.functions import pandas_udf

@pandas_udf("long")
def multiply_func(a: pd.Series, b: pd.Series) -> pd.Series:
    return a * b

spark.udf.register("multiply_func", multiply_func)

sql("select multiply_func(1,1)").show()
sql("select multiply_func(id,2) from range(10)").show()
sql("select multiply_func(id,2) from range(10000)").collect()
```



Arrow UDF – Spark Improvement

- Improvement to UDF for Spark 3.3.0
  - Bypasses conversion to Pandas
  - Awkward array can be used instead of Pandas
  - Improved performance for complex data with arrays
    - Details at: https://github.com/LucaCanali/Miscellaneous/blob/master/Spark_Notes/Spark_MapInArrow.md
  - Originally, this started by needs of Coffea team
  - Finally implemented as mapInArrow
    - Somehow a compromise, as the original arrow_udf idea was rejected
    - Good that Spark PMC at Databricks picked this up anyways
  - See SPARK-37227 and the original PR #34505

Demo 3

- HEP analysis benchmark notebooks

https://github.com/LucaCanali/Miscellaneous/tree/master/Spark_Physics

- Thanks to: https://iris-hep.org/projects/adl-benchmarks-index.html

Demo 4

- Outreach-style analysis
- Using Spark and Parquet: more familiar tools to data scientists outside HEP

- See example Higgs boson analysis at:

https://github.com/LucaCanali/Miscellaneous/tree/master/Spark_Physics

Previous Work (ML for HEP)

- Machine learning pipeline
  - Spark used for HEP data preparation at scale
  - DL distributed training on cloud resources
    - with TensorFlow on GPU + also tested with Spark on CPU
  - Comput Softw Big Sci 4, 8 (2020) https://rdcu.be/b4Wk9

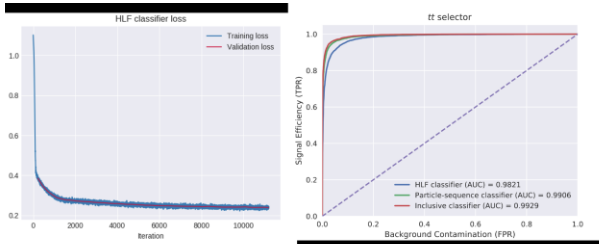
Data and models from Research

Input: labeled data and DL models

Feature engineering at scale

Hyperparameter optimization (Random/Grid search)

Distributed model training



Output: particle selector model

Previous Work (Data Reduction)

- CMS Bigdata project
- Data reduction at scale with Spark, up to 1 PB
    - This focused on scaling out a simple computation/filter and running massive I/O in parallel https://doi.org/10.1051/epjconf/201921406030

Runtime performance in minutes for different input sizes
Config: 100 Spark executors, 8 cores per Spark executor, 7 GB per Spark executor.
CPU: running on YARN.
Storage: reading from EOS using Hadoop-XRootD connector

Lessons Learned and Wrap-up

What Spark Can Offer to HEP

- Spark DataFrame API:
  - Provide powerful abstractions and rich language(s)
    - Both for data preparation and analysis
  - Mature and an industry reference
  - Can handle complex schemas

- Run DataFrame locally and at scale using distributed computing
  - Runs on clusters and cloud (YARN/Hadoop, Kubernetes)
  - HPC and batch: can run Spark stand-alone (requires extra integration work)

- Integration with a large ecosystem
  - Can use for many file formats: Parquet, csv, ROOT, …
  - Storage systems: HDFS, S3, EOS, ...
  - External systems: databases, elastic search, streaming, etc

To Improve: Performance Gap

- Apache Spark Performance
  - Identified several areas of improvement for Apache Spark (3.2)

- Python UDF performance
  - Sending data to Python workers has been improved but still slow

- Spark functions
  - Higher order functions performance need improvements
  - More array functions and functions for Lorentz vector processing?

- Spark engine
  - Apache Spark does not (yet) have vectorized execution
  - State-of-the-art HEP tools have large parts running native code vs. Spark is currently mostly Scala and Java running on the JVM (JIT compiled).

Open Questions: Data Formats

- ROOT data format ingestion is not optimized for Spark
    - It’s a hard job with limited resources
    - Kudos to Andrew for Laurelin library + uproot team
- Apache Parquet and ORC
    - Optimized reader for Spark, supported by a large community
- The flatter the data the better
    - Data in nanoAOD format already much easier to process with Spark than deeply nested AOD

Plans and Future Work

- Gather feedback
- Develop further examples and benchmarks
  - Trailing on work done with Coffea/awkward array and with ROOT/RDataFrame
- Piggyback on Apache Spark improvements
  - Spark is still improving quite fast
- Work with community (HEP and Spark)
  - We noticed some interest by Apache Spark committers on understanding HEP use cases
  - Occasionally work together, as in the case of arrow UDF, also currently open SPARK-34265 and SPARK-38098
  - Some physicists also interested in trying out Spark ?

References + Acknowledgments

- Machine Learning Pipelines with Modern Big Data Tools for High Energy Physics, Matteo Migliorini, Riccardo Castellotti, Luca Canali, Marco Zanetti, *Comput Softw Big Sci* **4,** 8 (2020).
- Using Big Data Technologies for HEP Analysis, M. Cremonesi *et al.*, EPJ Web of Conferences 214, 06030 (2019)
- Evolution of the Hadoop Platform and Ecosystem for High Energy Physics, Z. Baranowski *et al.*, EPJ Web of Conferences 214, 04058 (2019)
- Big Data Tools and Cloud Services for High Energy Physics Analysis in TOTEM Experiment, V. Avati *et al.*, 2018, Proceeding of: 2018 IEEE/ACM International Conference on Utility and Cloud Computing Companion (UCC Companion)
- CMS Analysis and Data Reduction with Apache Spark, O. Gutsche et al. 2018 J. Phys.: Conf. Ser.1085 042030
- Evaluating Query Languages and Systems for High-Energy Physics Data, Dan Graur, Ingo Müller, Mason Proffitt, Ghislain Fourny, Gordon T. Watts, Gustavo Alonso, Proc. VLDB Endow., Vol 15, Issue 2 (2021).

- Get started with Spark at CERN: https://hadoop-user-guide.web.cern.ch/

- Thanks:
  - to the teams in the IT Hadoop, Spark and streaming service, and in the SWAN service, in particular to Riccardo Castellotti.
  - to the members and contributors to the (now finished) openlab project on data analysis with CMS
  - to Jim Pivarski, Andrew Melo, Lindsey Grey for discussion and their work on Spark arrow integration improvements and Spark-ROOT format integration
  - Lukas Heinrich, Gordon Watts, Ghislain Fourny, Ingo Müller, for discussions.
  - Ruslan Dautkhanov and Hyukjin Kwon from Databricks for their work and support on adding mapInArrow
  - Matteo Migliorini and Marco Zanetti for the work on ML pipelines.