Data Engineering

- Data Processing
- Batching
- Spark

Data Engineering Course
Lucas Rosa
2024

Data Processing
Introduction

Lucas Porto Rosa
Brazilian, 36 years
Principal Data Engineer at HBDC (Metys & Hugo Boss)
LinkedIn: https://www.linkedin.com/in/lucprosa/

Personal

Moved to Portugal in Oct 2021 with wife and dog
Gaúcho, gremista
Crafter beer, travel, guitar player, etc.

Work Experience

> 10 years of experience working with data as DBA, Business Intelligence analyst and Data Engineer

Education

System Development Analysis
MBA Data Science
Tech Certifications

Data Processing
Introduction

Data Engineer experience?
Spark?
SQL?
Python?

Data Processing
Introduction

Development Environment

Google Colab - https://colab.research.google.com/
GitHub - https://github.com/lucprosa/dataeng-basic-course/
Dataproc - https://cloud.google.com/

Data Processing

Agenda

	15 -**Data Processing** Tutor: Lucas Rosa Horário: 19h - 23h	16 -**Data Processing** Tutor: Lucas Rosa Horário: 9h - 18h	17
	22 -**Data Processing** Tutor: Lucas Rosa Horário: 19h - 23h	23 -**Real-Time Data (Streaming)** Tutor: Lucas Rosa Horário: 9h - 18h	24
	29	30 -**Real-Time Data (Streaming)** Tutor: Lucas Rosa Horário: 9h - 18h	

- Day 15
  - Data Processing / Batching Introduction
  - Spark Introduction, components
  - Hands-On (Google Colab + Dataproc)
  - Spark architecture
  - Data Solutions / Alternatives to Spark
- Day 16
  - Spark common Issues
  - Hands-On (Google Colab + Dataproc)
  - Concepts about ETL/Medallion Architecture
  - Technical challenge
- Day 17
  - Tech challenges (continuation)
  - Doubts/Question

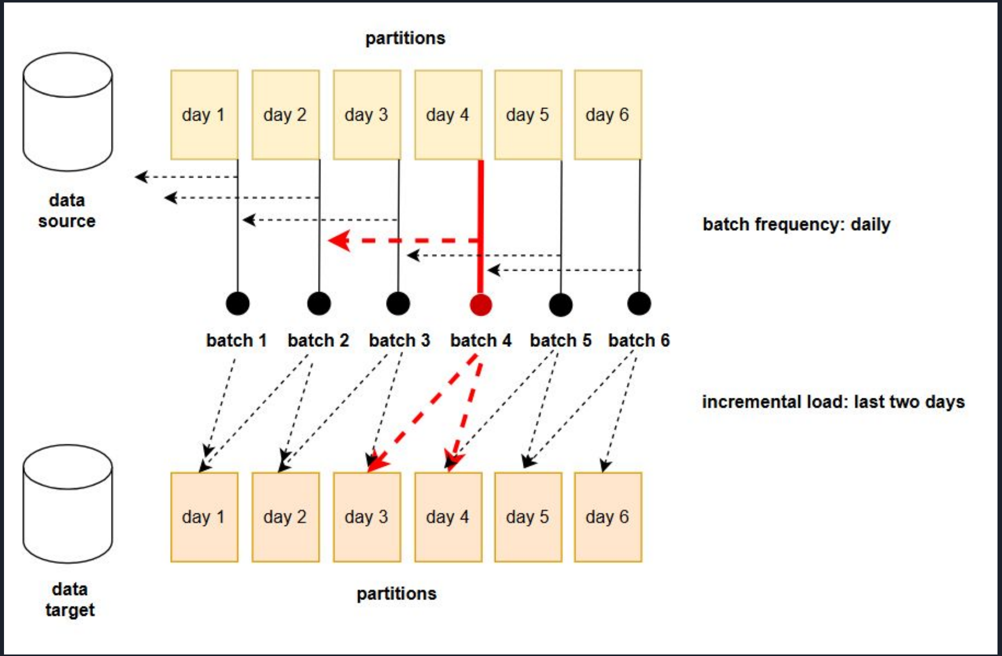
Data Processing

Index

Data Processing
Batching

Data Processing
Batching

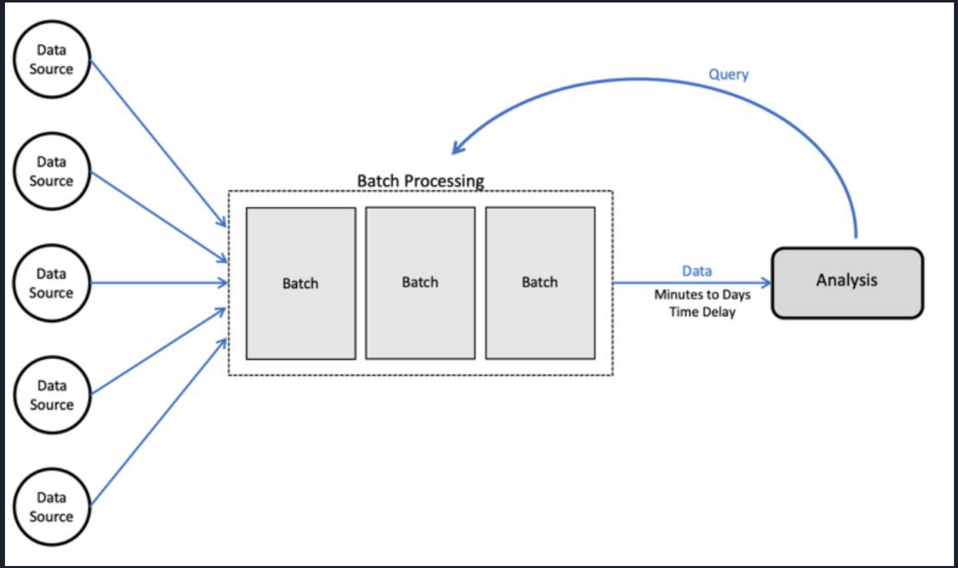
- Batch jobs
- Data is collected, stored and processes in batches
- Jobs are scheduled / batch frequency
- Full and incremental loads



Data Processing

Batching

- Process data in batches / chunks
- **Data Volume**: Large amount of data
- **Data Latency:** High latency (hourly, daily, weekly, monthly)
- **Cost**: Low cost (comparison to streaming)



Data Processing

Batching

USE CASES

Data Integration, data consolidation
ETL/ELT jobs
Data Quality
Data Archiving
Backups
Data Mining

BENEFITS

Data Analytics
Dashboarding
Reports
Machine Learning
Monitoring KPIs
Business decision-making
Security alerts
Data Quality checks
Data transformations & enrichments

Apache Spark

Data Processing

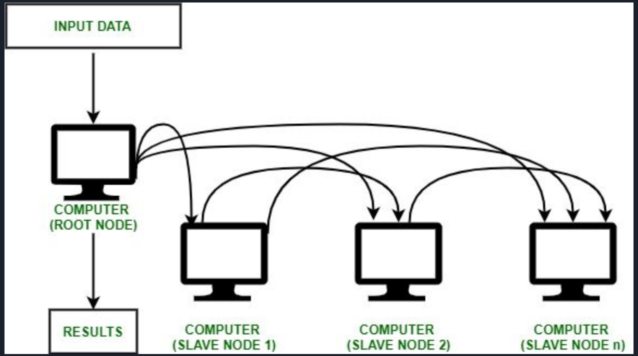
Apache Spark

- Apache Spark
- https://spark.apache.org/

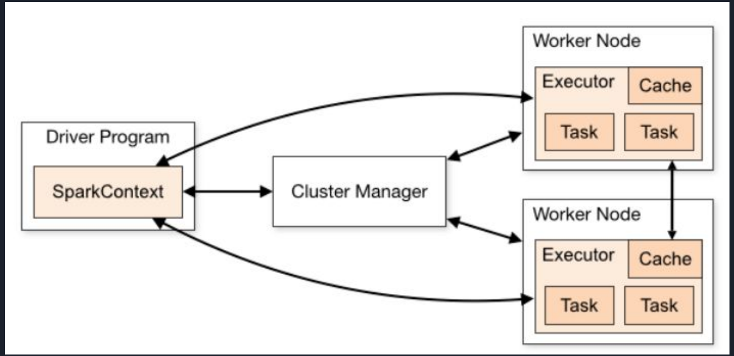
Data Processing
Apache Spark

- Unified engine for large-scale data analytics
- Distributed and parallel computing, in-memory, fault tolerant, etc
- Open Source (https://www.apache.org/)
- Developed in 2009 (UC Berkeley) to replace MapReduce computing paradigm
- Spark cluster components (driver/master node, workers, executors)
- RDDs (resilient distributed dataset)

Distributed/Parallel Processing



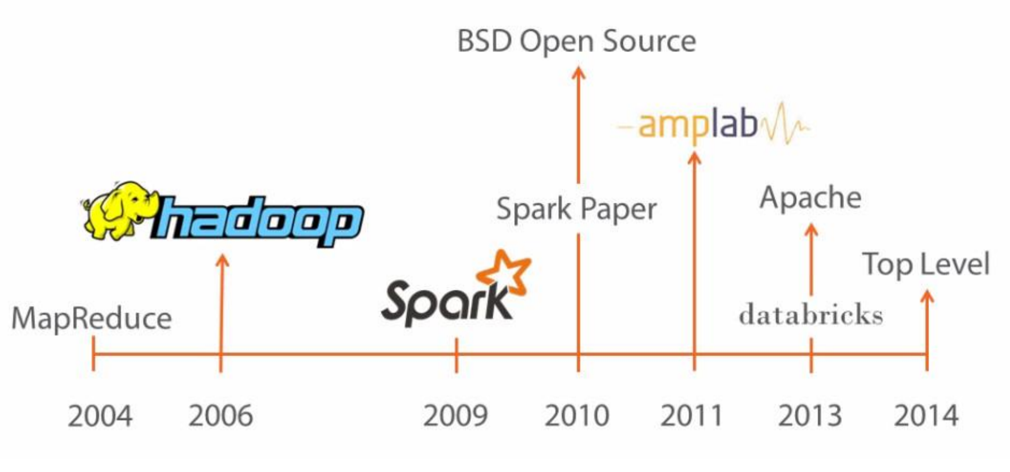
Spark architecture



Data Processing

Apache Spark

- Big Data problems
  - How to store?
  - How to process?

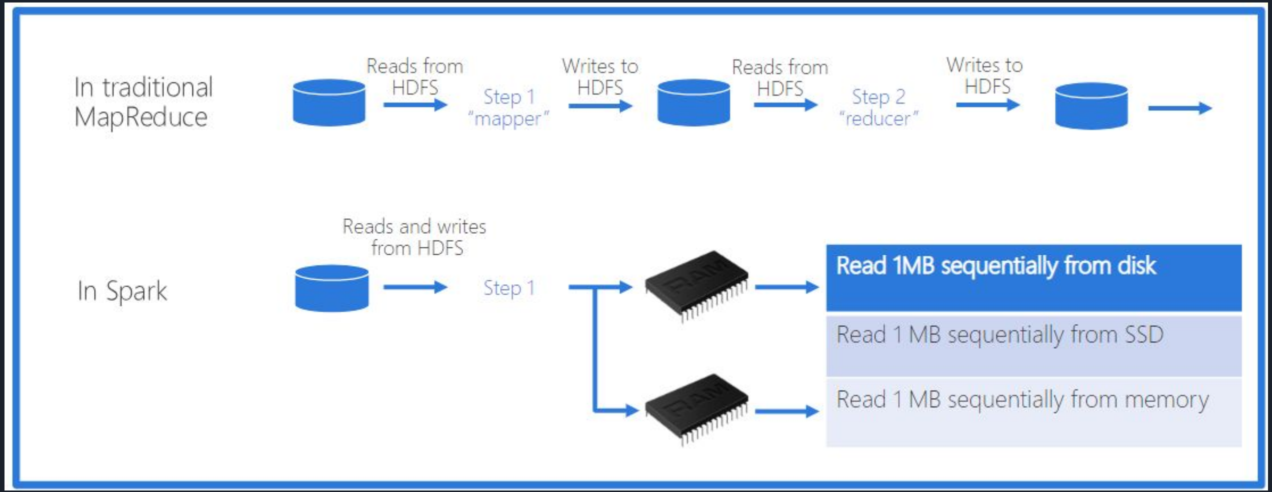


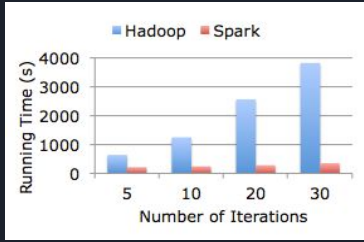
- 2002 - Hadoop - Apache Nutch
- 2003 - Google - GFS (Google File System)
- 2004 - Google - MapReduce
  - Big data processing model
  - Java focused
  - Map, Shuffle, Partition, Reduce
  - Read/write intensive
- 2004 - Hadoop - Apache Nutch
  - GFS + MapReduce
- 2006 - Hadoop - Yahoo + Apache Nutch
  - HDFS + MapReduce
- 2008 - Apache Hadoop
- 2009 - Spark - Research at UC Berkeley AmpLabs
- 2010 - Spark - First paper
- 2013 - Spark - Apache Software Foundation
- 2014 - Spark 1.0

Data Processing

Apache Spark

- MapReduce vs Spark





https://amplab.cs.berkeley.edu/projects/spark-lightning-fast-cluster-computing/

Data Processing

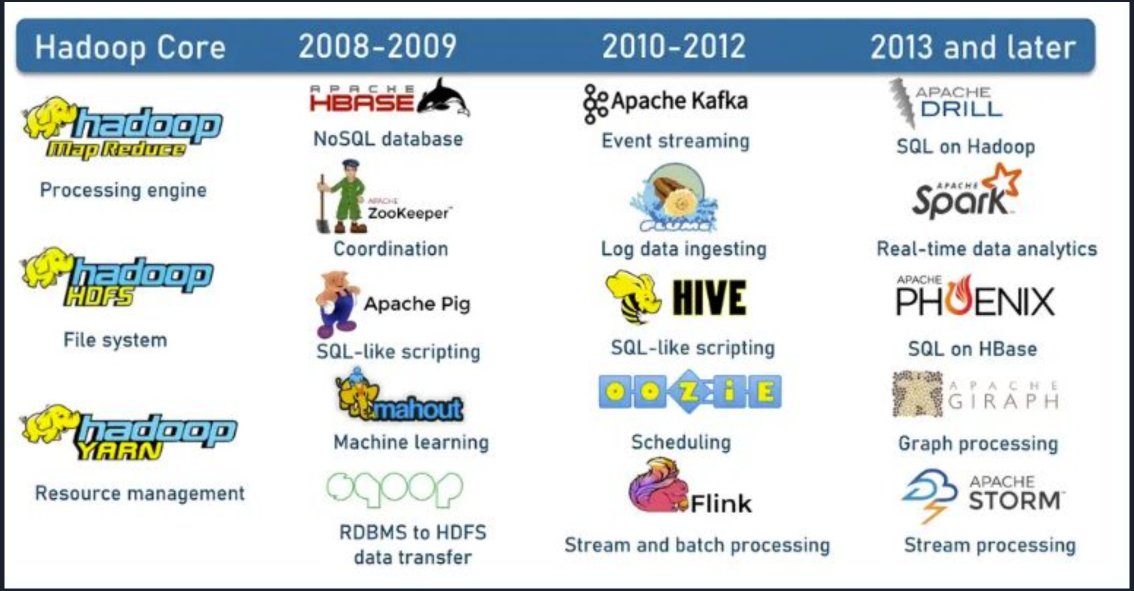
Apache Spark

- MapReduce vs Spark

Criteria	Hadoop	Spark
Real-time Data Processing	Primarily for batch processing, not optimized for real-time tasks.	Well-suited for real-time or near-real-time processing due to in-memory speed.
Accessing Data Randomly in Memory	Reads and writes data to/from disk, less efficient for random memory access.	Designed for in-memory processing, allowing efficient random data access.
Iterative and Interactive Operations	Writes intermediate results to disk, and can be slow for iterative or interactive tasks.	Optimized for iterative and interactive operations, keeping data in memory.

Data Processing
Apache Spark

- Hadoop Ecosystem timeline



Data Processing

Apache Spark

Spark - How/where to use it?

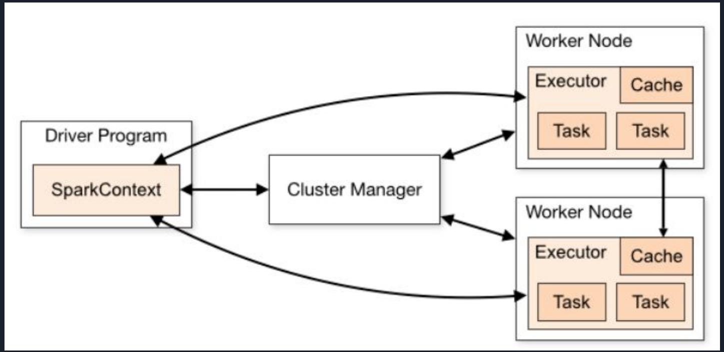
Local machine, standalone, K8	SaaS / PaaS	Hadoop cluster on Cloud	Hadoop cluster on Prem
APACHE Spark, docker, kubernetes	databricks, snowflake (Snowpark), Amazon Athena, Azure Synapse Analytics, Google Big Query	amazon EMR, Cloud Dataproc, HDInsight	cloudera, HORTONWORKS, hadoop

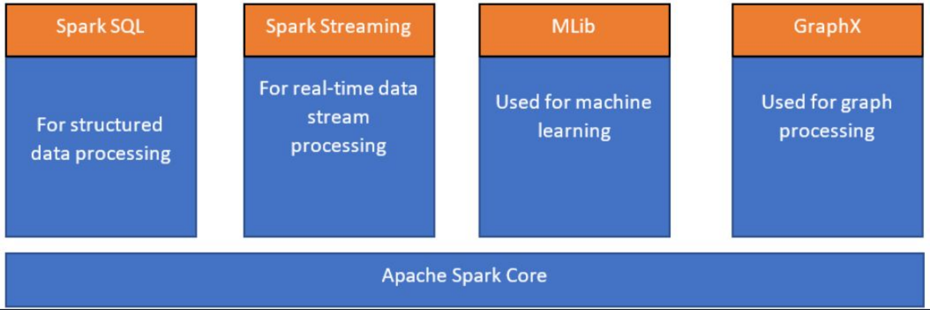
Apache Spark Architecture

Data Processing

Apache Spark

- APIs
- Components
- Architecture





Reference: https://spark.apache.org/docs/latest/cluster-overview.html

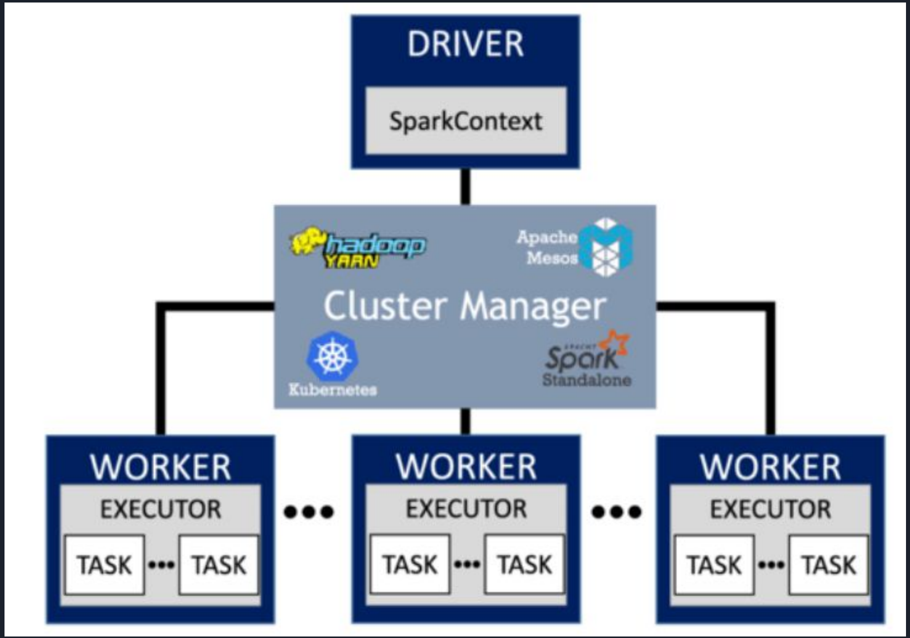
Data Processing

Apache Spark

- **Spark APIs (languages)**
  - Java, Scala, Python, R
- **Spark APIs**
  - **SQL API**
  - Dataset API (only Scala and Java)
  - **Dataframe API**
  - Pandas API
  - MLib for machine learning
  - GraphX for graph processing
  - **Structure Streaming** for stream processing
  - Spark Connect API (> 3.4)
- **Spark CLI**
  - spark-shell
  - spark-submit
  - **pyspark**
  - sparkR
  - spark-sql

Data Processing
Apache Spark

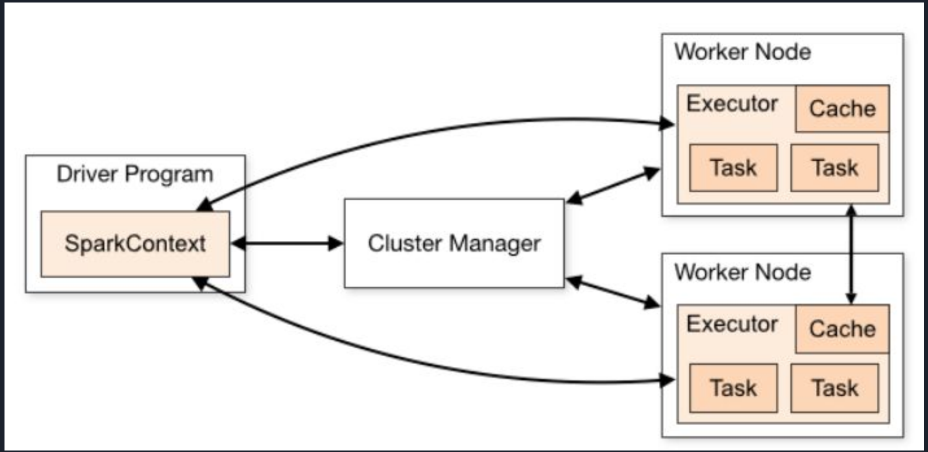
- Spark Components
  - Driver
  - Workers
  - Cluster Manager
    - YARN
    - MESOS
    - Kubernetes



Data Processing

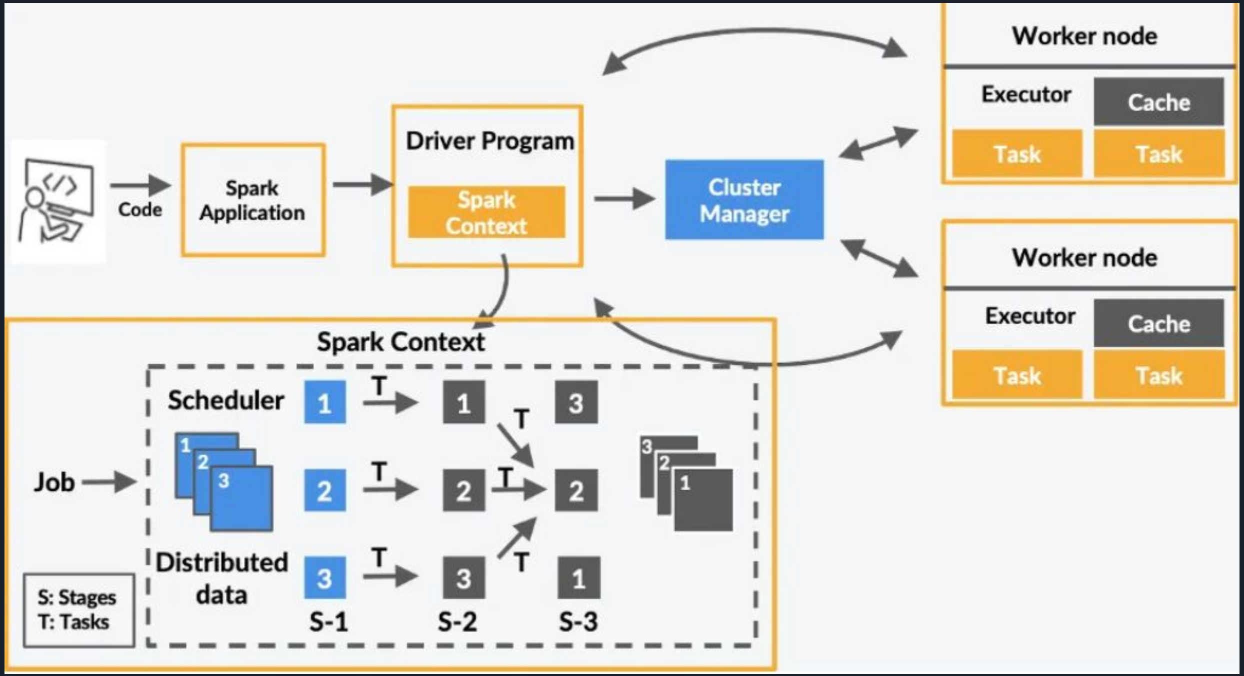
Apache Spark

- **Workers** - Machines in the cluster
- **Driver** - Central control for Spark application (main method)
- **Cluster Manager** - Launches executors and allocate and manage resources
- **Executors** - Processes running tasks on workers (can take one partition at the time)
- **Cores** - CPU cores allocated to executors
- **Cache** - Memory or disk caching in workers
- **Tasks** - Spark commands sent by Driver to Executors
- **Stages** - Contains a quantity of tasks
- **Partitions** - Logical chunk of data in a large distributed dataset (128MB)



Data Processing

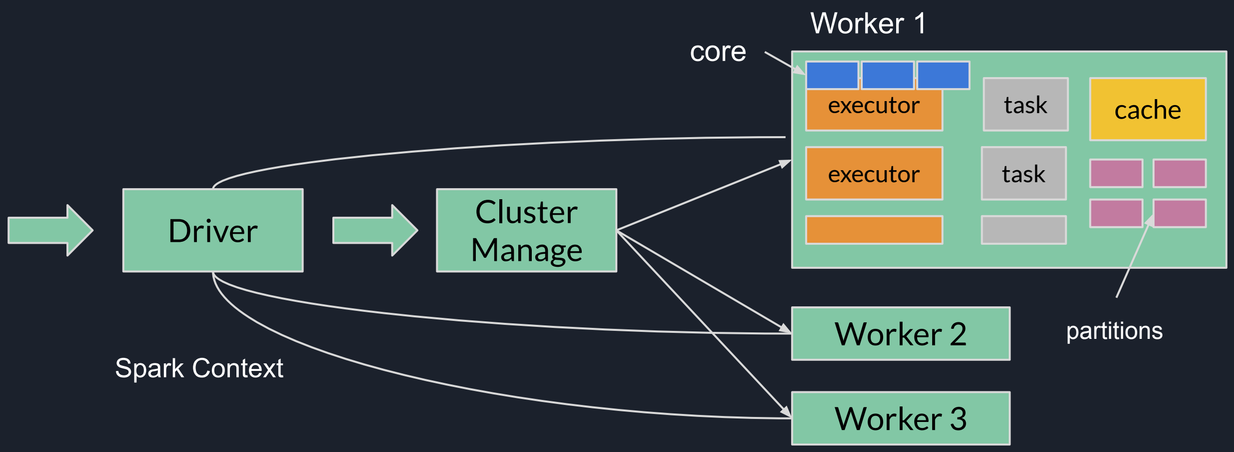
Apache Spark



Data Processing

Apache Spark

CODE:
line 1 - read csv 1 from path 1 -> df1
line 2 - read csv 2 from path 2 -> df2
line 3 - add new column to df1 -> df1
line 4 - add new column to df2 -> df2
line 5 - join df1 with df2 -> df3
line 6 - aggregate data -> df4
line 7 - write df4 to the lake



- *code is splitted in tasks and spread among the workers by the driver*
- *RDDs will be splitted among the partitions in the workers*
- *executors run one task per core*
- *single task will operate on a single partition*

- *parallelism*
- *distributed workload*
- *fault-tolerance (operation lineage)*

To evaluate Spark parallel tasks and performance

- *How many workers does the cluster have?*
- *How much Memory and CPU each worker have?*
- *Define memory allocation and CPU for each executor and driver*
- *Define shuffle partitions*

Data Processing

Apache Spark

- **RDD**
  Resilient distributed dataset (Fault-tolerant collection of elements that can be operated on in parallel)
  Ex: read dat from parquet (1GB) → rdd (contains many partitions distributed across the cluster)

- **TRANSFORMATIONS**
  Operations that return another RDD/dataset as output
  Ex: join, groupBy, filter

- **ACTIONS**
  Operations that return a value to the driver after running computation on the RDD/dataset
  Ex: write, count, show, collect

- **DAG**
  Logical execution plan for a job. Sequence of operations.
  Ex: T1 -> T2 -> T3 -> T4 -> A1

Data Processing
Apache Spark

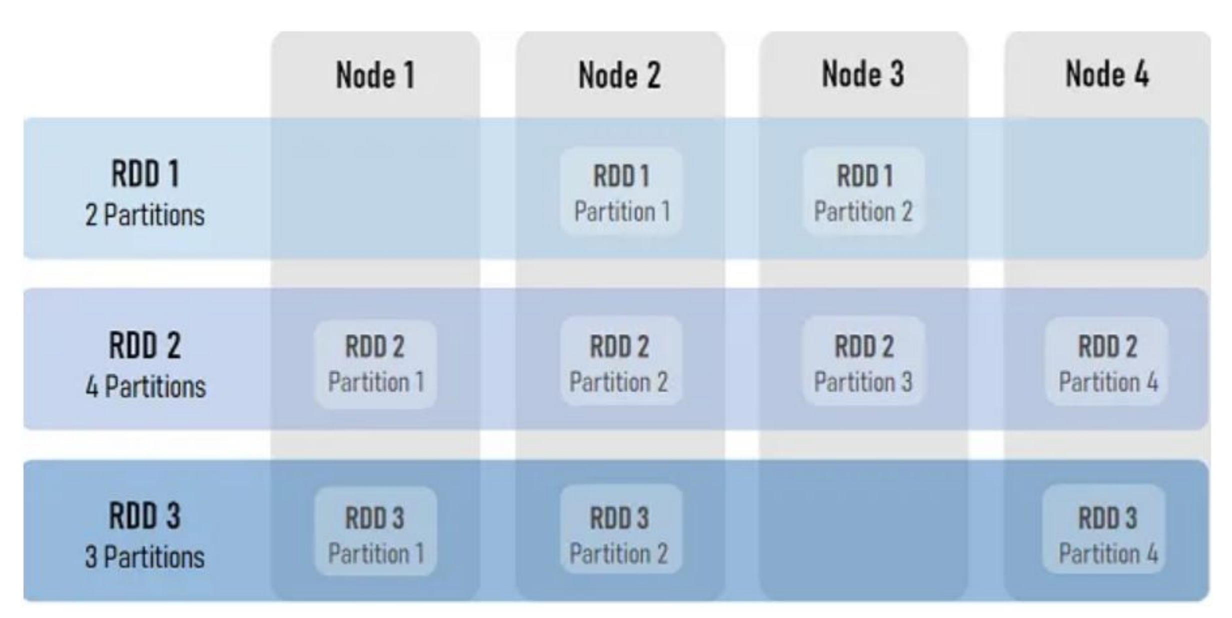
- **DATASETS**
  Distributed collection of data (strong typing, ability to use powerful lambda functions)
  RDD’s benefits + Spark SQL’s optimized execution engine

- **DATAFRAMES**
  Dataset organized into named columns. Equivalent to a table in a relational database
  RDD’s benefits + Spark SQL’s optimized execution engine

Data Processing

Apache Spark

RDDs x Partitions x Workers(Nodes)



Spark Hands-On

Data Processing
Apache Spark

HANDS-ON

https://github.com/lucprosa/dataeng-basic-course/tree/main

spark/examples/00-setup.ipynb	Installation, Spark Session, Spark Context
spark/examples/01-rdds.ipynb	RDDs
spark/examples/02-dataframes.ipynb	DataFrame
spark/examples/03-sql.ipynb	SQL, Temp Views, Spark Catalog
spark/examples/04-joins.ipynb	Joins
spark/examples/05-aggregations.ipynb	Aggregations
spark/examples/06-write_partitioning.ipynb	Writing operation, Write Mode, partitionBy
spark/examples/07-udf.ipynb	User-defined functions
spark/examples/09-windows-function.ipynb	Windows Function
spark/examples/10-misc_performance.ipynb	Cache, Persist, broadcast join, repartition/coalesce, explain

Data Processing

Apache Spark

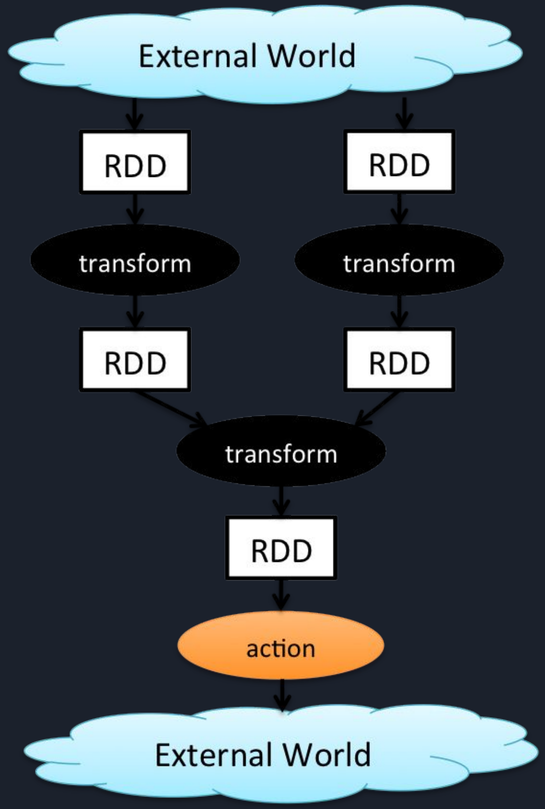
- Lazy Evaluation - Action vs Transformation
- DAG - Directed Acyclic Graph
- Wide & Narrow Transformations
- Data Shuffling

Transformations

- Wide (multiple partitions) - groupBy, join, distinct…
- Narrow (single partition) - map, filter, union…

Actions

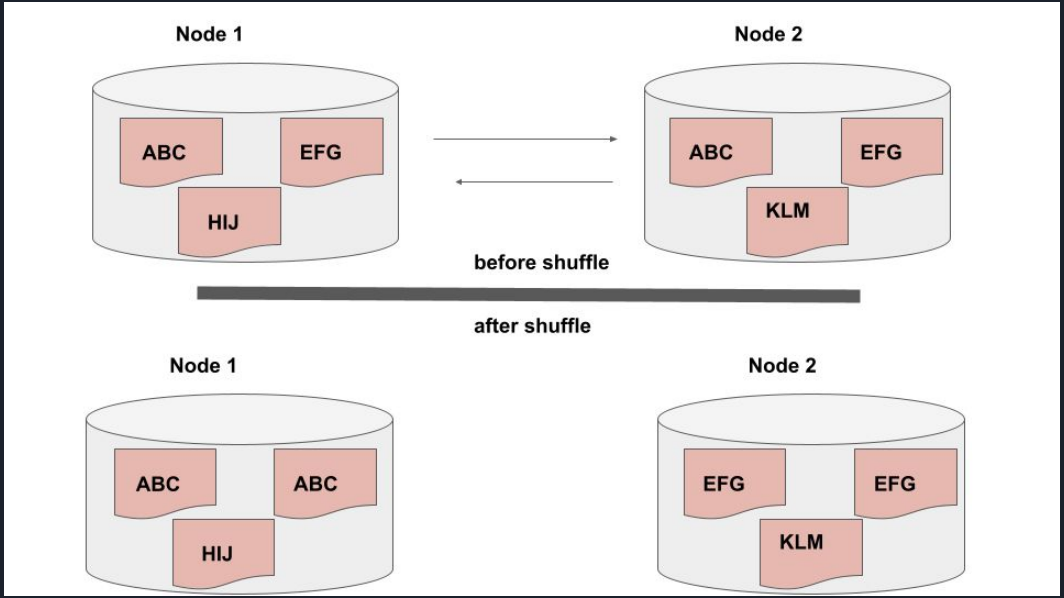
- count, collect, top, take, write/save…



Data Processing

Apache Spark

- Data Shuffling

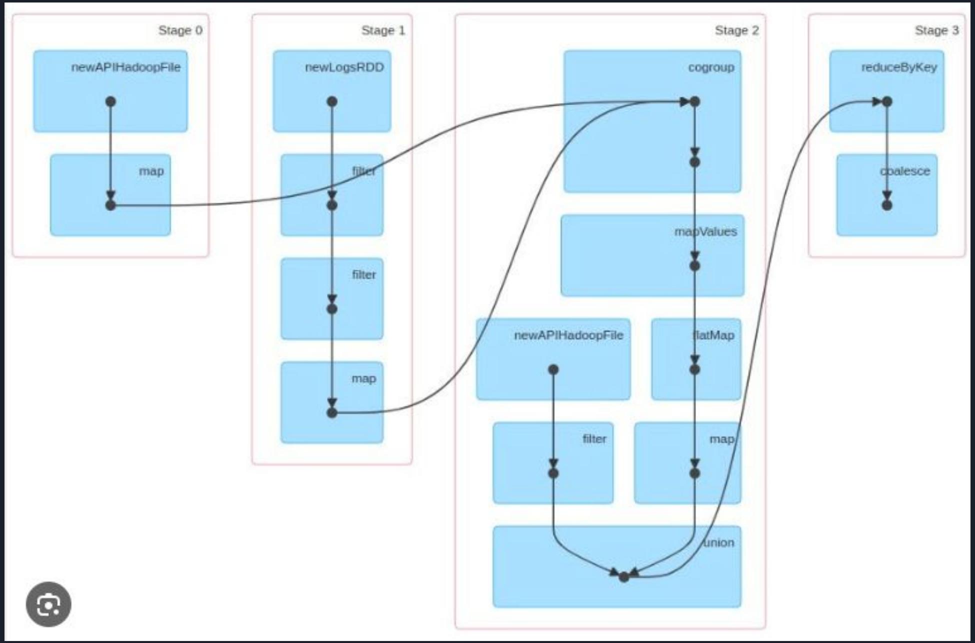


- ABC and EFG in different / same nodes
- Wide transformations
- Partitioning
- Cluster configuration
- Data sizing
- spark.sql.shuffle.partitions
  - 200 (default)

Data Processing

Apache Spark

- Spark DAG Example



- Stages
- Tasks

Data Processing

Apache Spark

Spark has many properties

- spark.executor.memory
- spark.executor.cores
- spark.executor.memoryOverhead
- spark.driver.memory
- spark.driver.cores
- spark.sql.shuffle.partitions
- …

To get/set properties: *spark.conf.get("property")* & *spark.conf.set("property", value)*

Spark Docs: https://spark.apache.org/docs/3.5.1/configuration.html

break.

Apache Spark Common Issues

Data Processing

Apache Spark

- Common issues
  - Having big files / few partitions - Less executors will be used, rest of executors will be idle (less parallelism = less performance)
  - Having too many small files - Requires more network communication for small files and increase data shuffling across the workers
  - Wrong partition logic in tables - Avoid columns with high cardinality, choose columns that can be used in filters and aggregations
  - Skewed data - Data distribution is not correct on partitions (adjust partition logic / salting technique)
  - Data Shuffle errors - Joins/group by can cause wrong data distribution across partition / adjust shuffle partitions size
  - OutOfMemory - JVM error, driver or executor run out of memory
  - Performing big transformations that requires data shuffling, using not optimal configuration or processing big amount of data with small resources
  - UDFs performance

Data Processing

Apache Spark

HANDS-ON

https://github.com/lucprosa/dataeng-basic-course/tree/main

spark/misc/read_from_api.ipynb	Reading from API
spark/misc/etl_program.ipynb	ETL program template
spark/misc/word_count.ipynb	Example of using RDDs

Data Processing

Lakehouse, Medallion Architecture

- **Lakehouse** = Data Warehouse + Data Lake
  - *Query Engines, data governance, ACID transactions, all type of data, etc*
- **Query Engines** - Query data from databases and data lakes, provides many features like ACID transactions, time-travel, better read/write performance, etc (delta.io, Iceberg, Hudi)
- **Medallion Architecture** - Design pattern to organize data in the data lake
  - How to organize data in a data lake?
  - Transient, Staging, Bronze, Silver (enriched), Gold (curated)

Reference:
Lakehouse - https://www.databricks.com/glossary/data-lakehouse
Delta Lake - https://delta.io/
Medallion - https://www.databricks.com/glossary/medallion-architecture

Data Processing
ETL flow

- ETL / Spark jobs
  - Ingestion -> Ingest from external data sources and write into bronze/raw layer
  - Cleansing -> Read from bronze, apply transformations and write into silver layer
  - Enrich -> Read from silver, apply business logic/aggregations and write into gold layer

Data Processing

ETL flow

1. import libraries

```
from pyspark.sql import DataFrame
```

2. Read the data sources

```
# reading from a table
df = spark.table("sales_db.sales")

# reading from a parquet file
df_1 = spark.read.parquet("/mnt/sales_db/sales")

# reading from a csv file
df_2 = spark.read.format("csv").load("/mnt/sales_db/sales")
```

3. Apply transformations

```
# rdd 1
df = df.filter("col1 == 'a'")

# rdd 2
df = df.limit(100)

# rdd3
df = df.join(df_1, "col1").select("col1", "col2", "col3")

# rdd4
output = df.union(df_2)
```

4. Write into target

```
output.write.format("delta").saveAsTable("sales_db.new_table")
```

Data Processing
Data Orchestrator

How to run / orchestrate Spark jobs?

- Apache Airflow - https://airflow.apache.org/
- Prefect - https://www.prefect.io/
- Azkaban - https://azkaban.github.io/
- Azure Data Factory
- Databricks Workflow
- Cron Linux

Data Processing
Apache Spark

SPARK CHALLENGES

https://github.com/lucprosa/dataeng-basic-course/tree/main/spark/challenges

Data Engineering

- Data Processing
- Batching
- Spark

Data Engineering Course
Lucas Rosa
2024