SPARK BASICS SIMPLY EXPLAINED

PART 1: HOW SPARK BOOSTS YOUR BIG DATA PROCESSING!

Data with
Nikk the Greek

APACHE Spark™

- ***What are the features of Spark?***
- ***How does it work lightning fast?***

Data with Nikk the Greek

The key features of APACHE Spark™

Simple. Fast. Scalable. Unified.

Key features

Batch/streaming data

Unify the processing of your data in batches and real-time streaming, using your preferred language: Python, SQL, Scala, Java or R.

SQL analytics

Execute fast, distributed ANSI SQL queries for dashboarding and ad-hoc reporting. Runs faster than most data warehouses.

Data science at scale

Perform Exploratory Data Analysis (EDA) on petabyte-scale data without having to resort to downsampling

Machine learning

Train machine learning algorithms on a laptop and use the same code to scale to fault-tolerant clusters of thousands of machines.

Reference: Apache Spark™ - Unified Engine for large-scale data analytics

Cloud providers support Spark

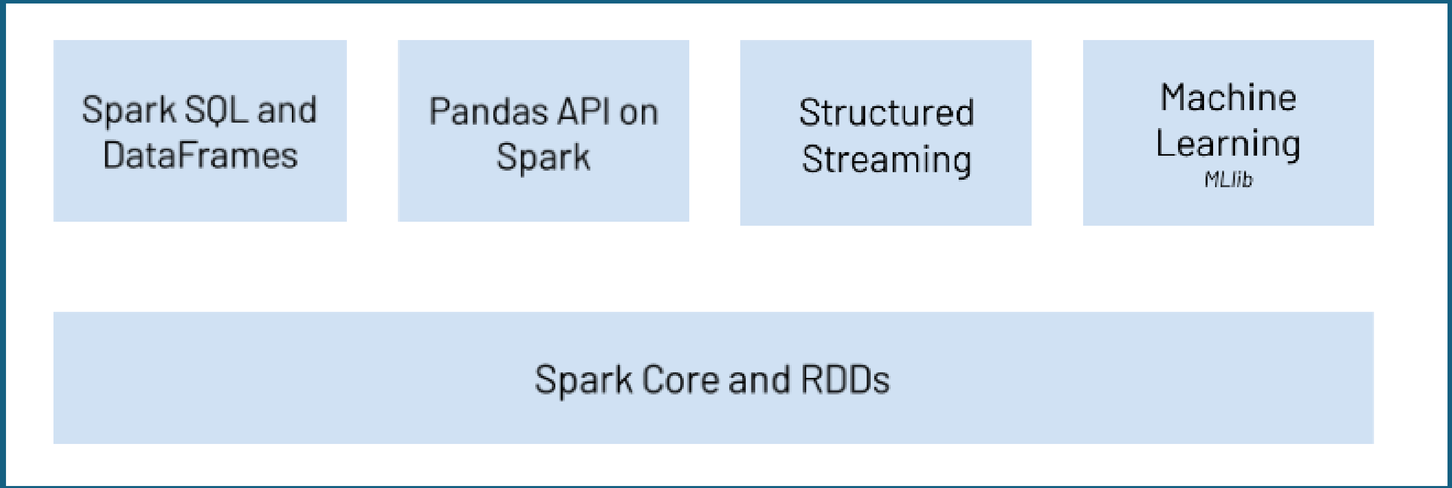
databricks

Microsoft Fabric

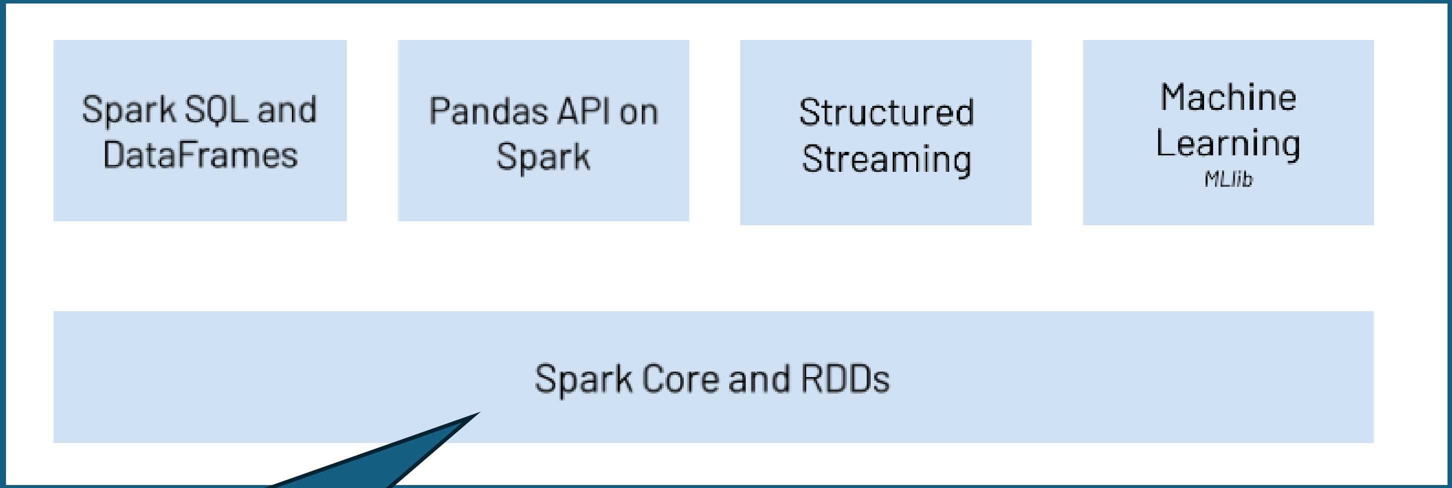
amazon EMR

Google Cloud Platform

Cloud Dataproc



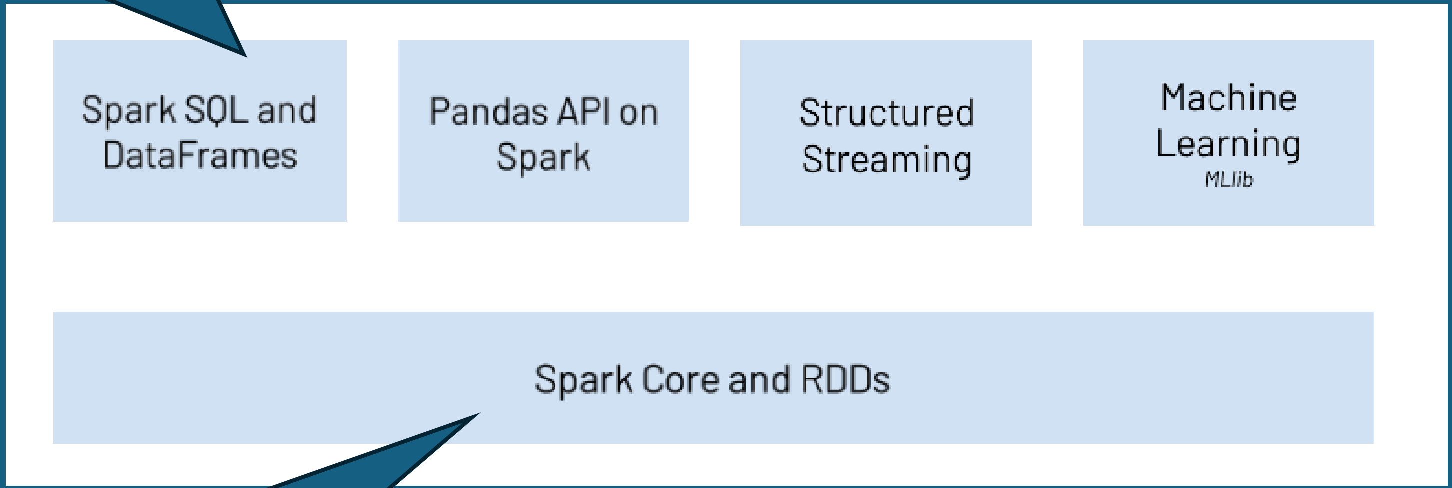
Reference: PySpark Overview — PySpark 3.5.0 documentation (apache.org)



- Underlying execution engine
- It provides RDDs (Resilient Distributed Datasets) and in-memory computing capabilities with fault tolerance
- DataFrames are built on top of RDDs with multiple optimizations

Reference: [*PySpark Overview — PySpark 3.5.0 documentation (apache.org)*]

- working with (semi-)structured data using DataFrames
- Leverage SQL warehouse capabilities with a hive metastore
- Uses lazy evaluation to leverage auto optimization using a Catalyst Optimizer

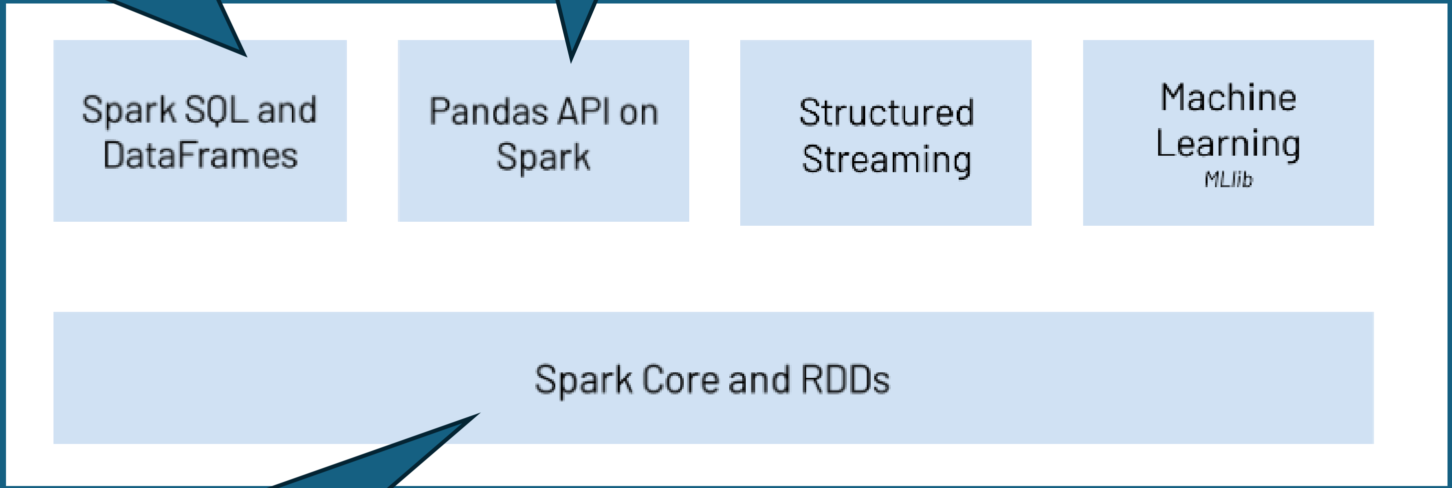


- Underlying execution engine
- It provides RDDs (Resilient Distributed Datasets) and in-memory computing capabilities with fault tolerance
- DataFrames are built on top of RDDs with multiple optimizations

Reference: PySpark Overview — PySpark 3.5.0 documentation (apache.org)

- working with (semi-)structured data using DataFrames
- Leverage SQL warehouse capabilities with a hive metastore
- Uses lazy evaluation to leverage auto optimization using a Catalyst Optimizer

- Use your Pandas code directly on Spark without any modification



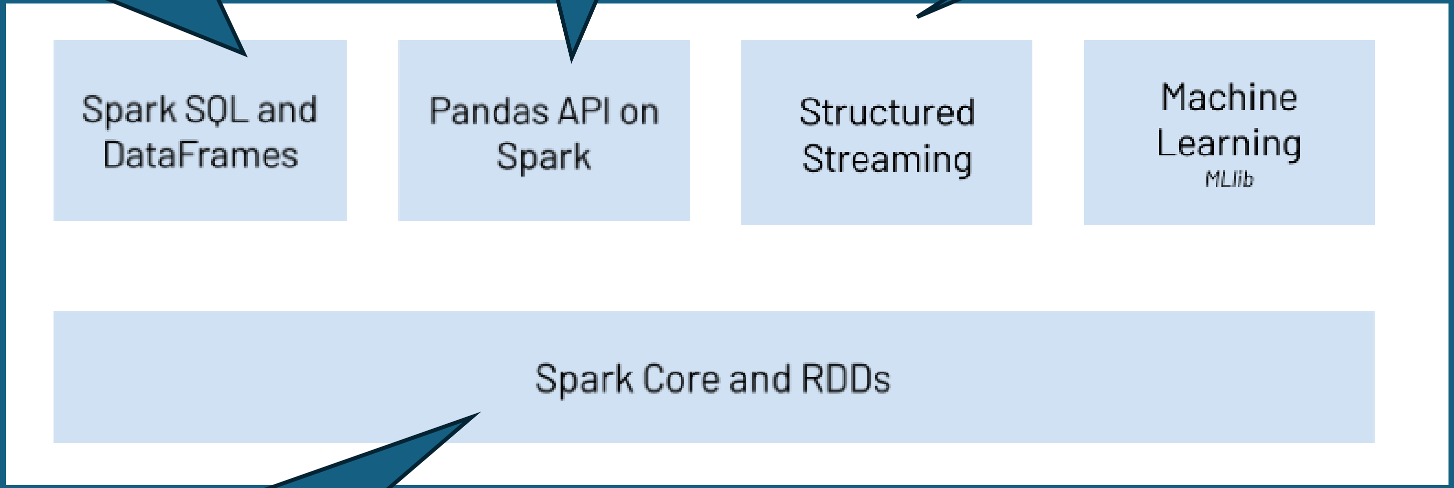
- Underlying execution engine
- It provides RDDs (Resilient Distributed Datasets) and in-memory computing capabilities with fault tolerance
- DataFrames are built on top of RDDs with multiple optimizations

Reference: PySpark Overview — PySpark 3.5.0 documentation (apache.org)

- working with (semi-)structured data using DataFrames
- Leverage SQL warehouse capabilities with a hive metastore
- Uses lazy evaluation to leverage auto optimization using a Catalyst Optimizer

- Use your Pandas code directly on Spark without any modification

- stream processing engine built on the Spark SQL engine.



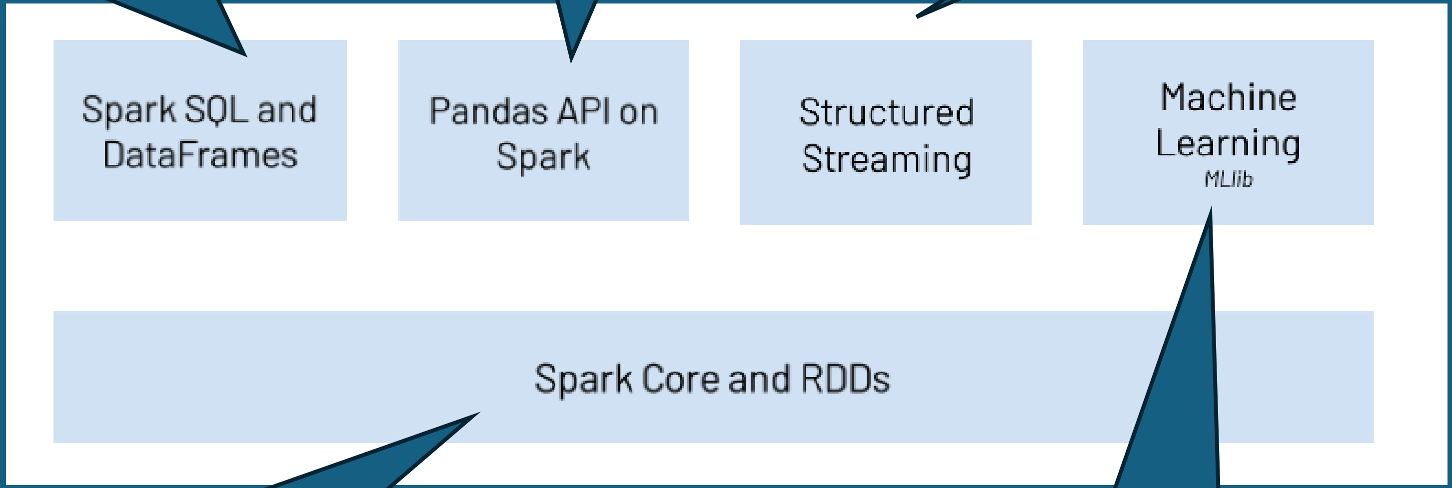
- Underlying execution engine
- It provides RDDs (Resilient Distributed Datasets) and in-memory computing capabilities with fault tolerance
- DataFrames are built on top of RDDs with multiple optimizations

Reference: PySpark Overview — PySpark 3.5.0 documentation (apache.org)

- working with (semi-)structured data using DataFrames
- Leverage SQL warehouse capabilities with a hive metastore
- Uses lazy evaluation to leverage auto optimization using a Catalyst Optimizer

- Use your Pandas code directly on Spark without any modification

- stream processing engine built on the Spark SQL engine.



- Underlying execution engine
- It provides RDDs (Resilient Distributed Datasets) and in-memory computing capabilities with fault tolerance
- DataFrames are built on top of RDDs with multiple optimizations

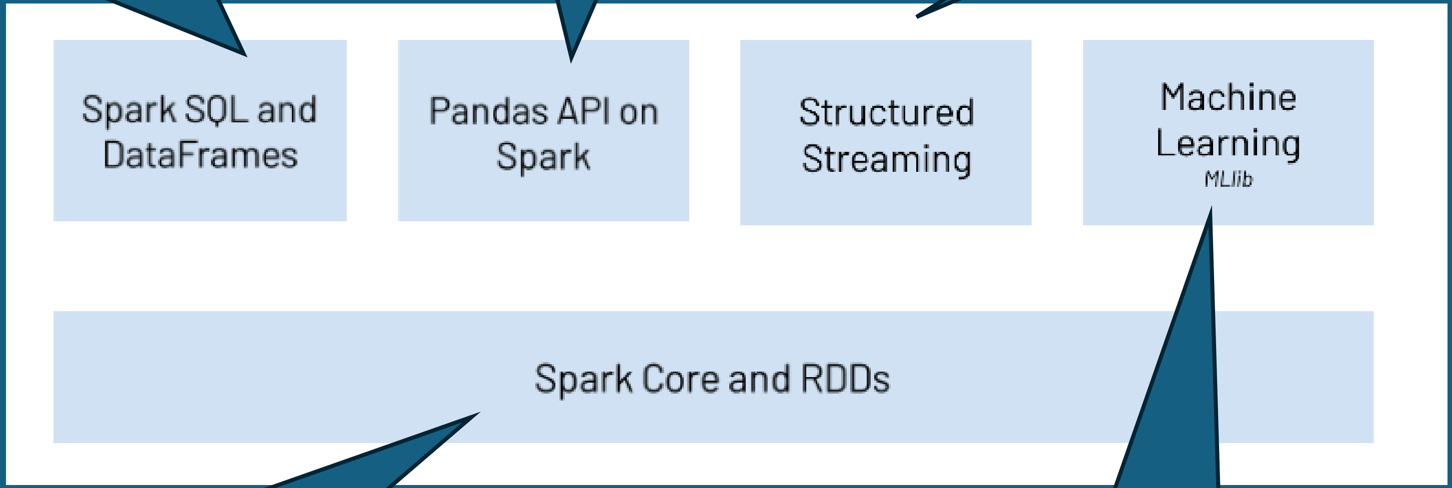
- A scalable machine learning library built on top of Spark

Reference: PySpark Overview — PySpark 3.5.0 documentation (apache.org)

- working with (semi-)structured data using DataFrames
- Leverage SQL warehouse capabilities with a hive metastore
- Uses lazy evaluation to leverage auto optimization using a Catalyst Optimizer

- Use your Pandas code directly on Spark without any modification

- stream processing engine built on the Spark SQL engine.



GraphX: Graph computation engine on Spark

- Underlying execution engine
- It provides RDDs (Resilient Distributed Datasets) and in-memory computing capabilities with fault tolerance
- DataFrames are built on top of RDDs with multiple optimizations

- A scalable machine learning library built on top of Spark

Reference: [PySpark Overview — PySpark 3.5.0 documentation (apache.org)]

Why is Apache Spark™ lightning fast?

*In short: The way Spark **distributes** and **optimizes** data processing in memory instead on storage like Hadoop*

In a 100TB sorting Benchmark Spark 3x faster with 10x less resources compared to Hadoop

Spark can be up to 100x faster than Hadoop

Reference: *Spark wins Daytona Gray Sort 100TB Benchmark | Apache Spark*

Count a small dataset

Load data

col1	col2
1	A
2	B
3	A
4	B
5	B
6	B
7	A
8	A
9	B
10	B
11	A
12	A

Filter col1 < 10

col1	col2
1	A
2	B
3	A
4	B
5	B
6	B
7	A
8	A
9	B

count

9

Count a big dataset – Python Out of Memory

Load data

col1	col2
1	A
2	B
3	A
4	B
5	B
6	B
7	A
8	A
9	B
10	B
…	…
99999999999	A

Filter col1 < 10

col1	col2
1	A
2	B
3	A
4	B
5	B
6	B
7	A
8	A
9	B

Python might break with out of Memory (OOM)

count

9

Spark parallizes by splitting data into partitions

Spark Partition

Load data

col1	col2
1	A
2	B
3	A

col1	col2
4	B
5	B
6	B

col1	col2
7	A
8	A
9	B

col1	col2
10	B
…	…
999…	A

Adjust partition sizes based on your ressources available

Filter col1 < 10

col1	col2
1	A
2	B
3	A

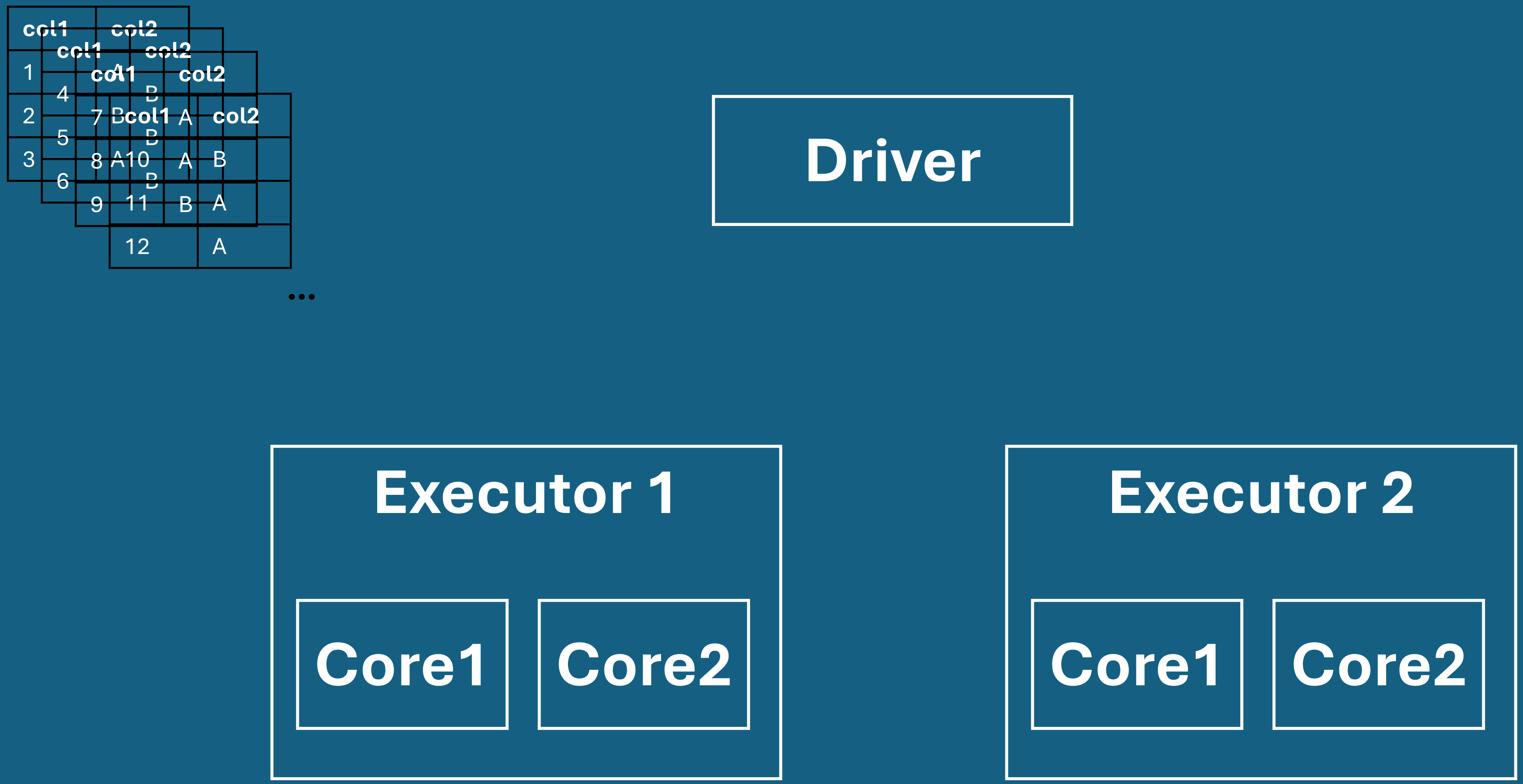
col1	col2
4	B
5	B
6	B

col1	col2
7	A
8	A
9	B

count

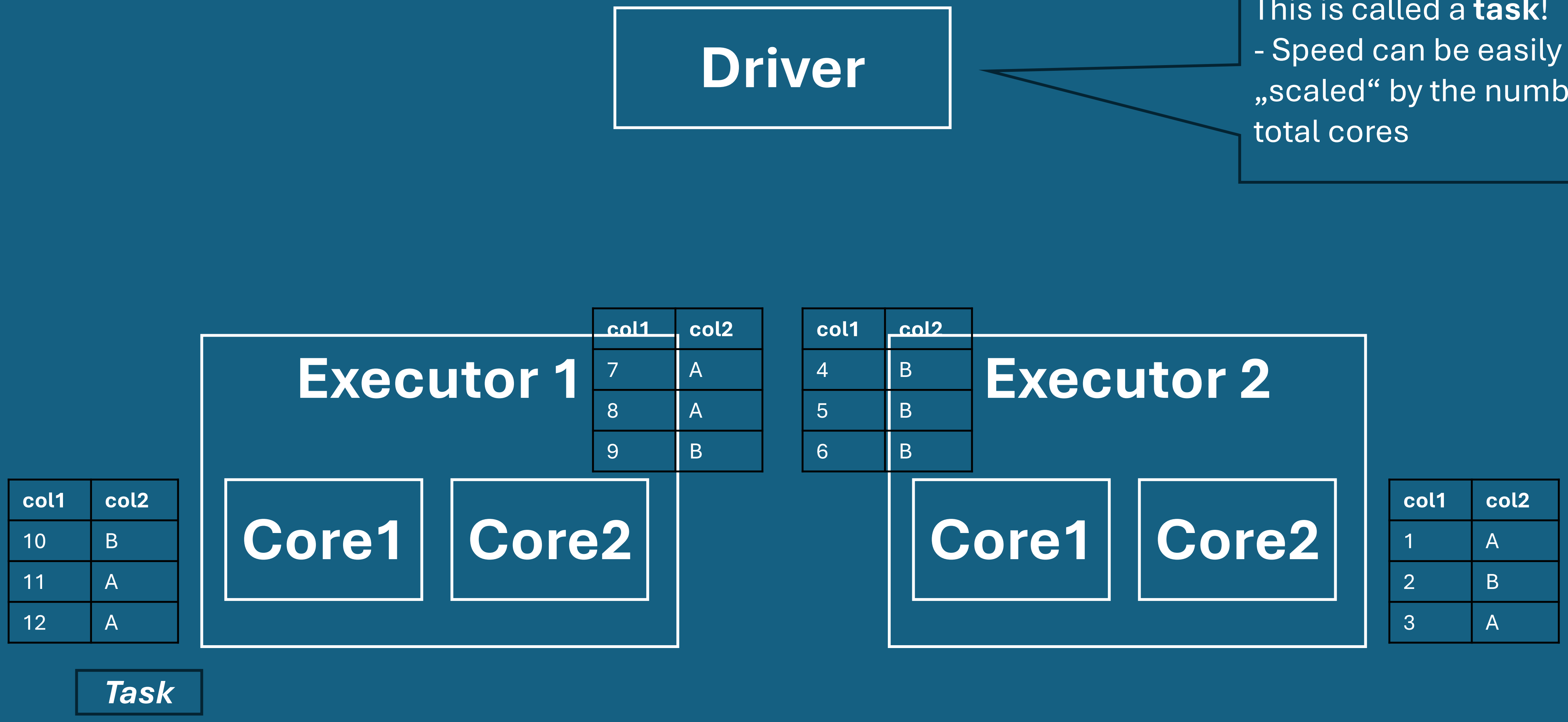
9

Spark Architecture



Spark Architecture

- One core executes one partition at the same time. This is called a **task**!
- Speed can be easily „scaled“ by the number of total cores



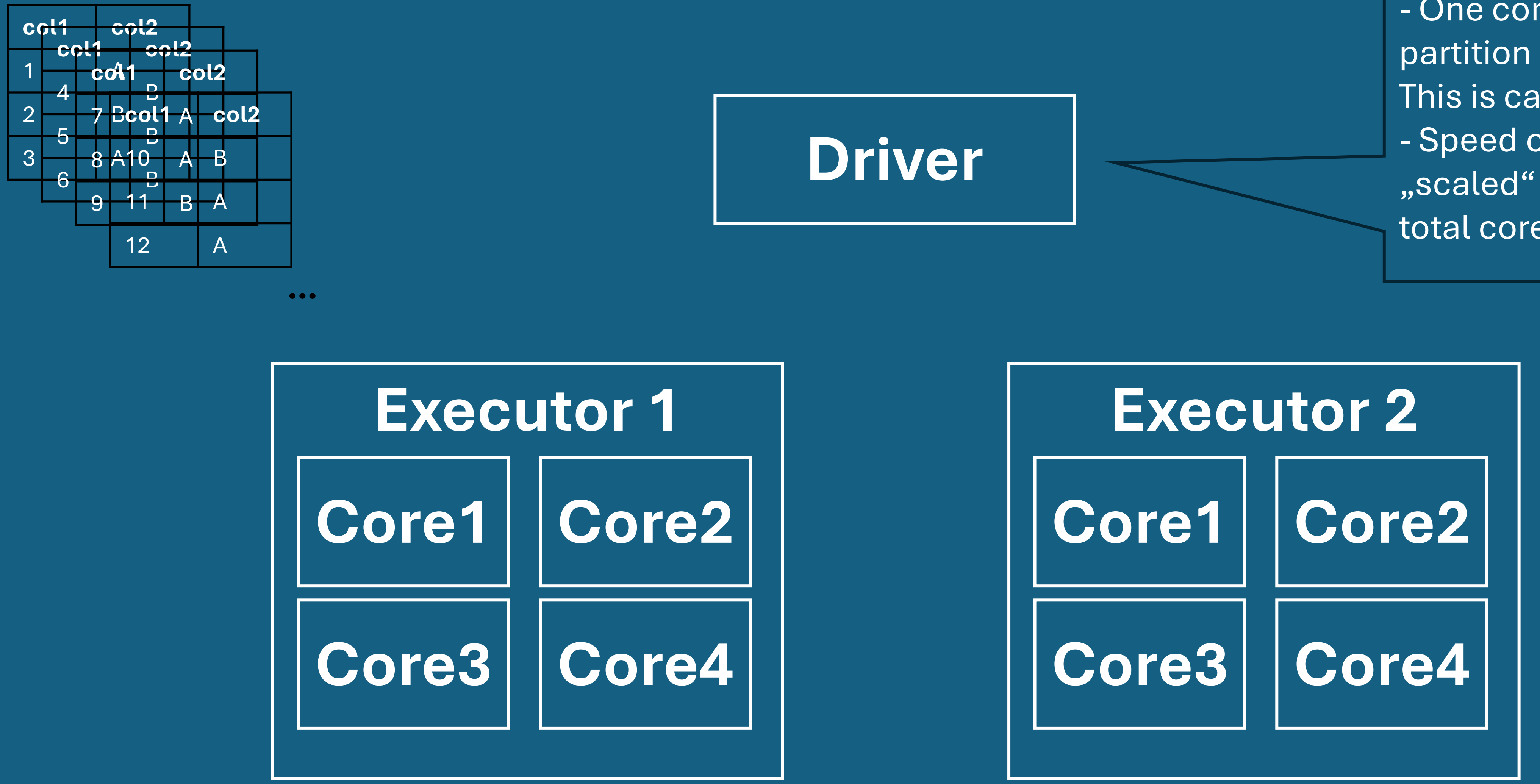
col1	col2
10	B
11	A
12	A

col1	col2
7	A
8	A
9	B

col1	col2
4	B
5	B
6	B

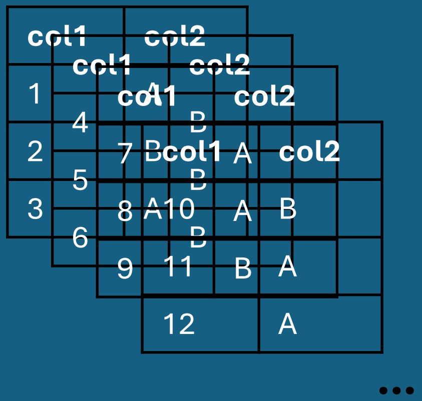
col1	col2
1	A
2	B
3	A

Spark Architecture



- One core executes one partition at the same time. This is called a **task**!
- Speed can be easily „scaled“ by the number of total cores

Spark Architecture



- One core executes one partition at the same time. This is called a **task**!
- Speed can be easily „scaled“ by the number of total cores

Driver

Executor 1

Core1 **Core2**

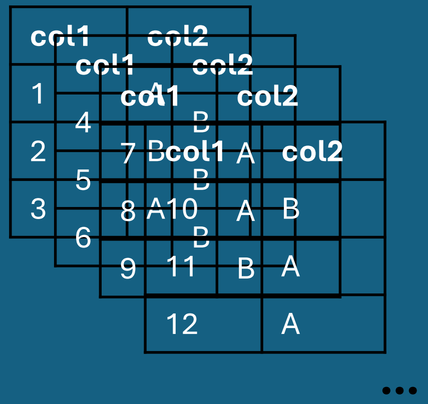
Executor 2

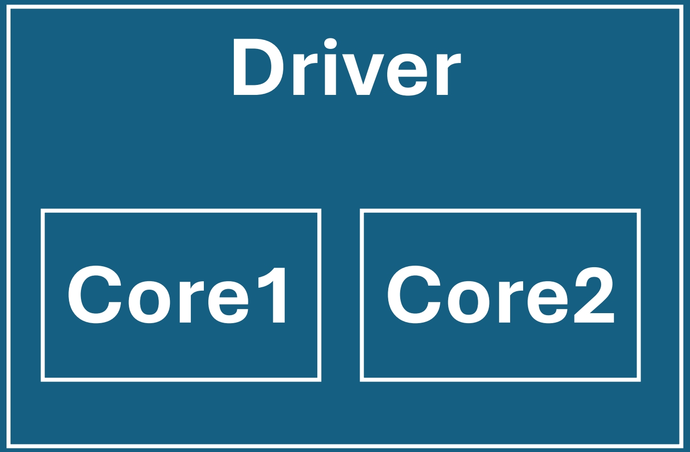
Core1 **Core2**

Executor 3

Core1 **Core2**

Single Node Cluster





Python vs. PySpark – Lazy Evaluation

```python
# Load parquet data from a defined path
pdf = pd.read_parquet(path)

# Filter data
pdf = pdf[pdf["id"] > 10000]

# Count
pdf.shape[0]
```

- Python loads all data in the first line into Memory
- After this the filtering and count is performedc
- For small datasets Python might be faster

```python
# Load parquet data from a defined path
sdf = spark.read.parquet(path)

# Filter data
sdf = sdf.filter(f.col("id") > 10000)

# Count
sdf.count()
```

Transformation

Action

- Spark is loading, filtering and counting data „lazily“ only with the count (called action)
- Based on all identfied steps the „Catalyst Optimizer“ finds the most efficient execution plan

Summary

Supports multiple languages like Python, SQL, Scala, Java

Spark is well supported by cloud providers as quasi standard for big data

A Pandas API on Spark is available

ML, streaming and batch processing is supported

Spark Architecture is build by a driver and one or multiple executors with cores to execute tasks

High scalability and flexibility due to parallisation based on partitions executed within tasks on the available cores

Lazy Evaluation until an action like count is executed for optimizing the query execution plan using the Catalyst Optimizer