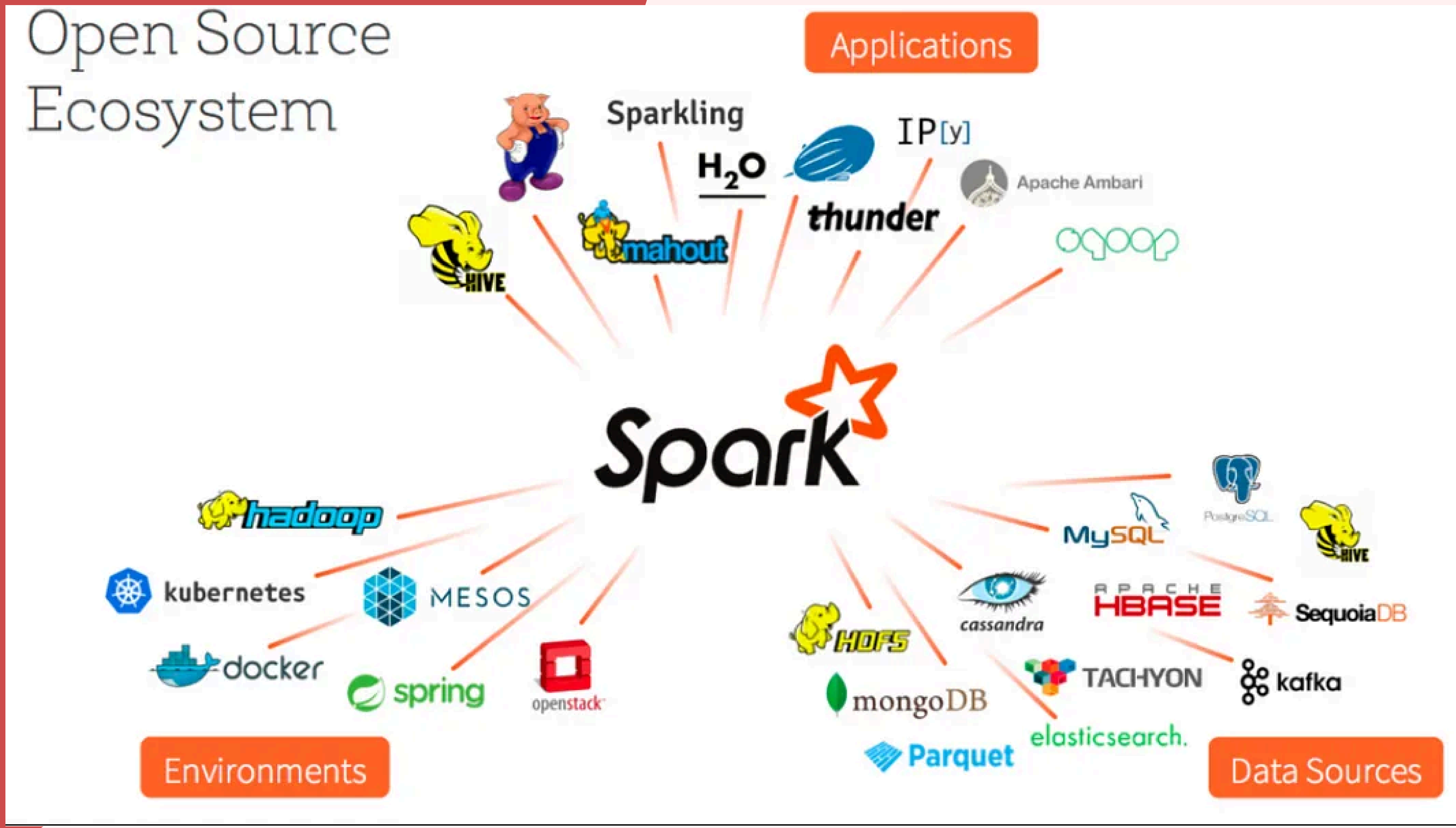
SPARK CORE & SPARK SQL

Sri Harshitha J-22011102030
Sneha M-22011102055
Yamini Krishna Kanuru-22011102039
Praneetha A-22011102011

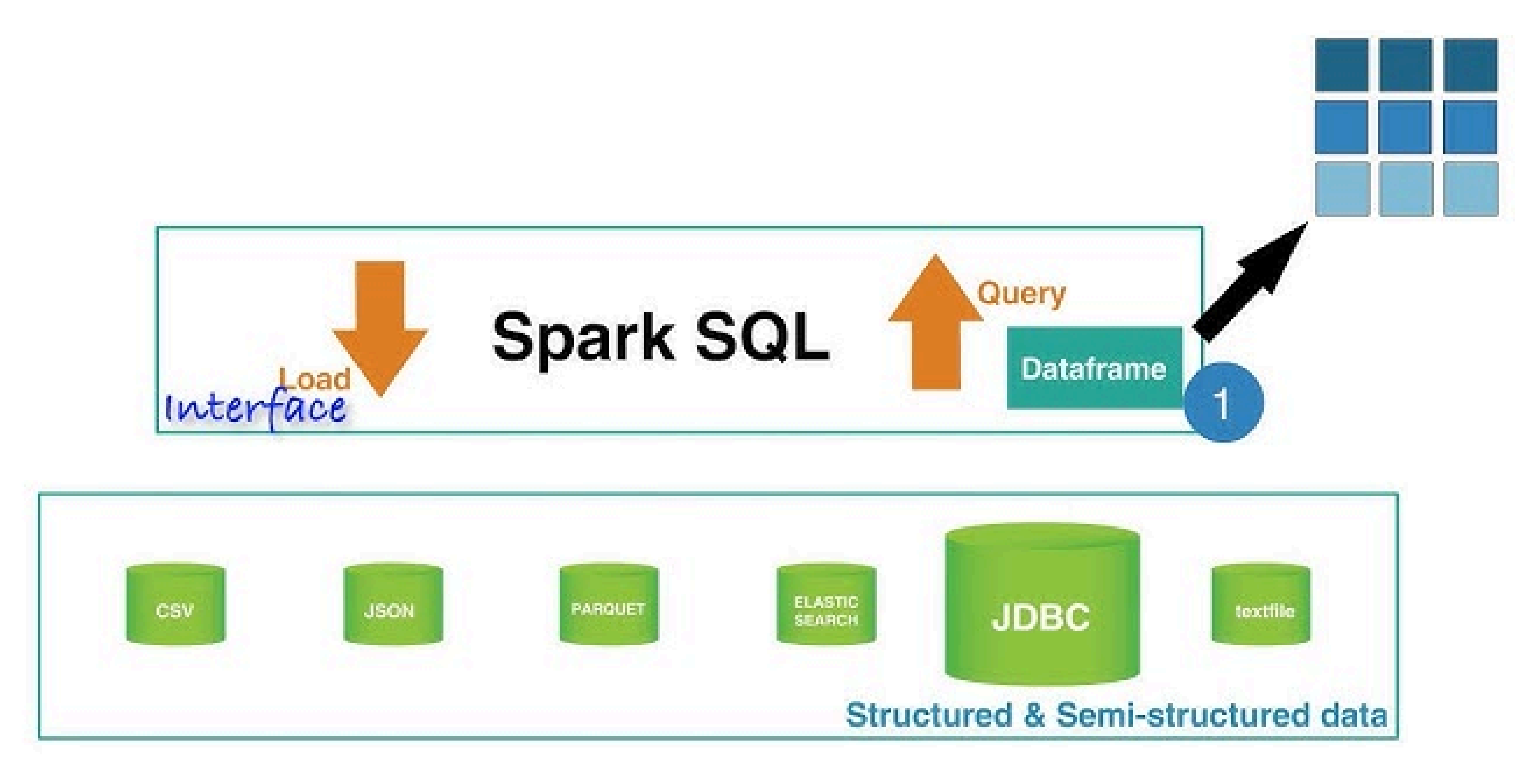
Introduction to Spark Core

- Apache Spark is an open-source distributed computing system that provides fast and general-purpose cluster computing.
- Spark Core is the foundation of Apache Spark, providing:
  - **Distributed task scheduling**
  - **Memory management**
  - **Fault recovery**
  - **Interactions with storage systems (HDFS, S3, etc.)**
- Originally, Spark's core abstraction was Resilient Distributed Datasets (RDDs), but DataFrames have largely replaced them due to their optimized execution and ease of use.



Introduction to Spark SQL

- Spark SQL is a module that allows querying structured data using SQL and the DataFrame API.
- Spark SQL exposes JDBC/ODBC server.
- Can also be connected using the spark shell.
- Can also be used with hive using hiveCtx.cacheTable(“tablename”).
- Provides SQL shell to directly create new tables or query from existing tables

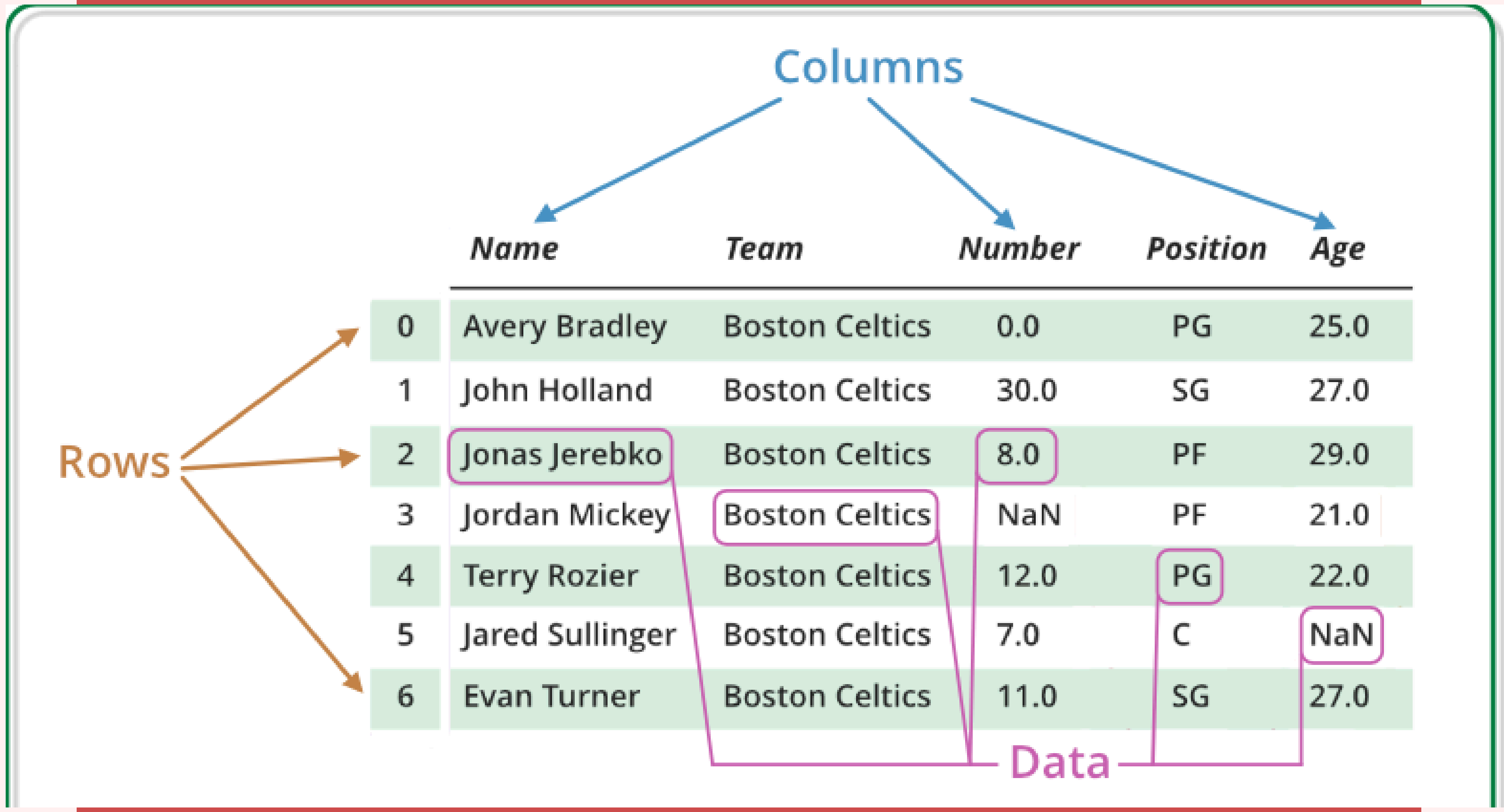


APACHE SPARK RDD VS. DATAFRAME

RDD		DATAFRAME
Structured or Unstructured	**DATA FORMAT**	Structured or Semi-structured
Any data source: Database or Text File	**API INTEGRATION**	Specific data source: JSON, AVRO, CSV
Supports OOP	**COMPILE-TIME**	Does not support OOP
Consistent Nature	**IMMUTABILITY**	Non-regeneratable domain object

DataFrames: A Higher-Level ... Abstraction

- DataFrames are distributed collections of structured data (like tables in relational databases).
- They provide an optimized API for large-scale batch and streaming data processing.
- More user-friendly than RDDs due to SQL-like query support.
- Schema enforcement ensures consistency in data processing.
- Supports automatic optimization via Spark's Catalyst Optimizer.
- Columnar storage format for better performance.
- Compatible with multiple data sources: JSON, CSV, Parquet, Hive, etc.
- Allows interoperability with Pandas DataFrames, BI tools (Tableau, Power BI), and machine learning workflows.
- Can seamlessly convert to RDDs when needed.



SparkSession – The Entry Point

- Unlike RDDs, DataFrames require a SparkSession to interact with Spark.
- SparkSession is the unified entry point for working with structured data in Spark.
- It manages the lifecycle of DataFrames, allowing you to read, process, and write data efficiently
- SparkSession replaces SparkContext, SQLContext, and HiveContext, making it the single access point for all operations.

- Example: Creating a SparkSession in PySpark:

python

```python
from pyspark.sql import SparkSession

# Creating a SparkSession
spark = SparkSession.builder.appName("MyApp").getOrCreate()
```

Creating and Manipulating DataFrames

01 Reading Data

- Load data from various sources (JSON, CSV, Parquet, Hive, etc.).
- Example: Reading a JSON file into a DataFrame:

```python
inputData = spark.read.json("data.json")
```

02 Running SQL Queries

- Create a temporary view to run SQL queries on DataFrames:

```python
inputData.createOrReplaceTempView("myView")

resultDF = spark.sql("SELECT name, age FROM myView WHERE age > 25")
resultDF.show()
```

DataFrame Operations •••

📌 DATAFRAMES PROVIDE VARIOUS TRANSFORMATION OPERATIONS TO MANIPULATE AND ANALYZE STRUCTURED DATA EFFICIENTLY.

01 Selecting Columns

- Use .select() to extract specific columns from a DataFrame.
- Example:

```python
resultDF.select("name").show()
```

02 Filtering Data

- Use .filter() to retrieve rows based on conditions.
- Example:

```python
resultDF.filter(resultDF["age"] > 25).show()
```

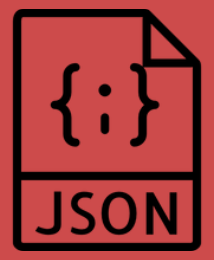
03 Grouping and Aggregation

- Use .groupby() and aggregate functions like .mean(), .sum(), etc.
- Example:

```python
resultDF.groupby("age").mean().show()
```

Working with Different Data Formats

```python
jsonDF = spark.read.json("data.json")
```




```python
parquetDF = spark.read.parquet("data.parquet")
```

```python
csvDF = spark.read.csv("data.csv", header=True, inferSchema=True)
```

Using DataFrames for Machine Learning

Spark MLlib is a scalable machine learning library that works efficiently with DataFrames.

- **FEATURE ENGINEERING EXAMPLE:**

```
from pyspark.ml.feature import VectorAssembler

# Combining multiple columns into a single feature vector
assembler = VectorAssembler(inputCols=["feature1", "feature2"], outputCol="features")
transformedDF = assembler.transform(resultDF)
```

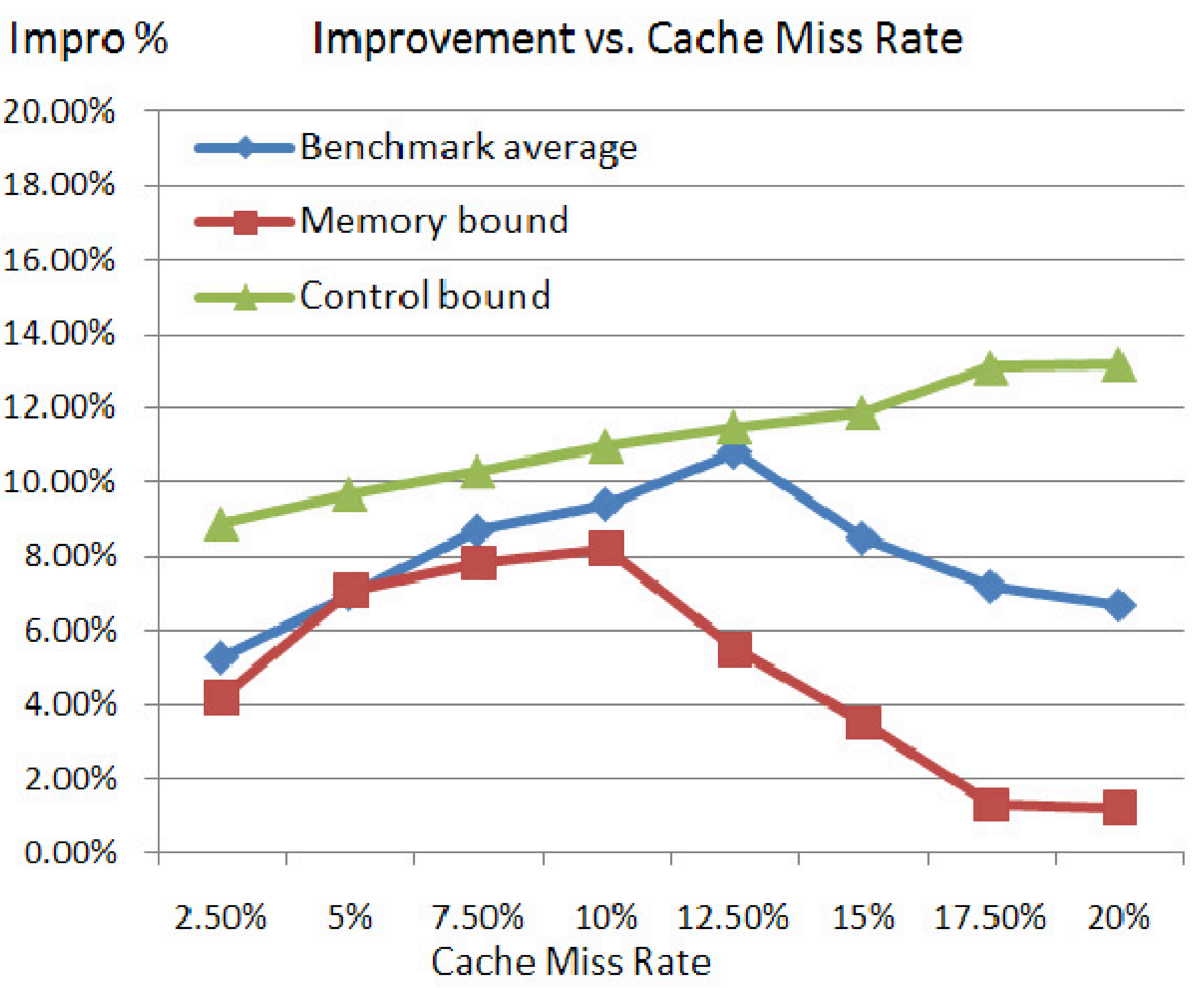
- **TRAINING A MACHINE LEARNING MODEL WITH DATAFRAMES**

```python
from pyspark.ml.regression import LinearRegression

# Create a Linear Regression model
lr = LinearRegression(featuresCol="features", labelCol="label")

# Train the model
model = lr.fit(transformedDF)

# Make predictions
predictions = model.transform(transformedDF)
predictions.show()
```



📌 Optimizing Spark SQL Queries

Why Optimization is Necessary?

Large-scale datasets can slow down query performance.

Optimizing queries helps reduce execution time and resource consumption.

- Partitioning:Distributes data across different nodes to improve parallel processing.
- Caching: Stores DataFrames in memory using .cache().
- Cache frequently accessed DataFrames for better performance.
- Broadcast Joins: Optimizes small dataset joins.
- Useful when joining a small dataset with a large one.
- Reduces shuffle operations.

THANK YOU


APACHE
Spark
BIG
DATA
ANALYTICS
TECHNOLOGIES
INFORMATION
STORAGE