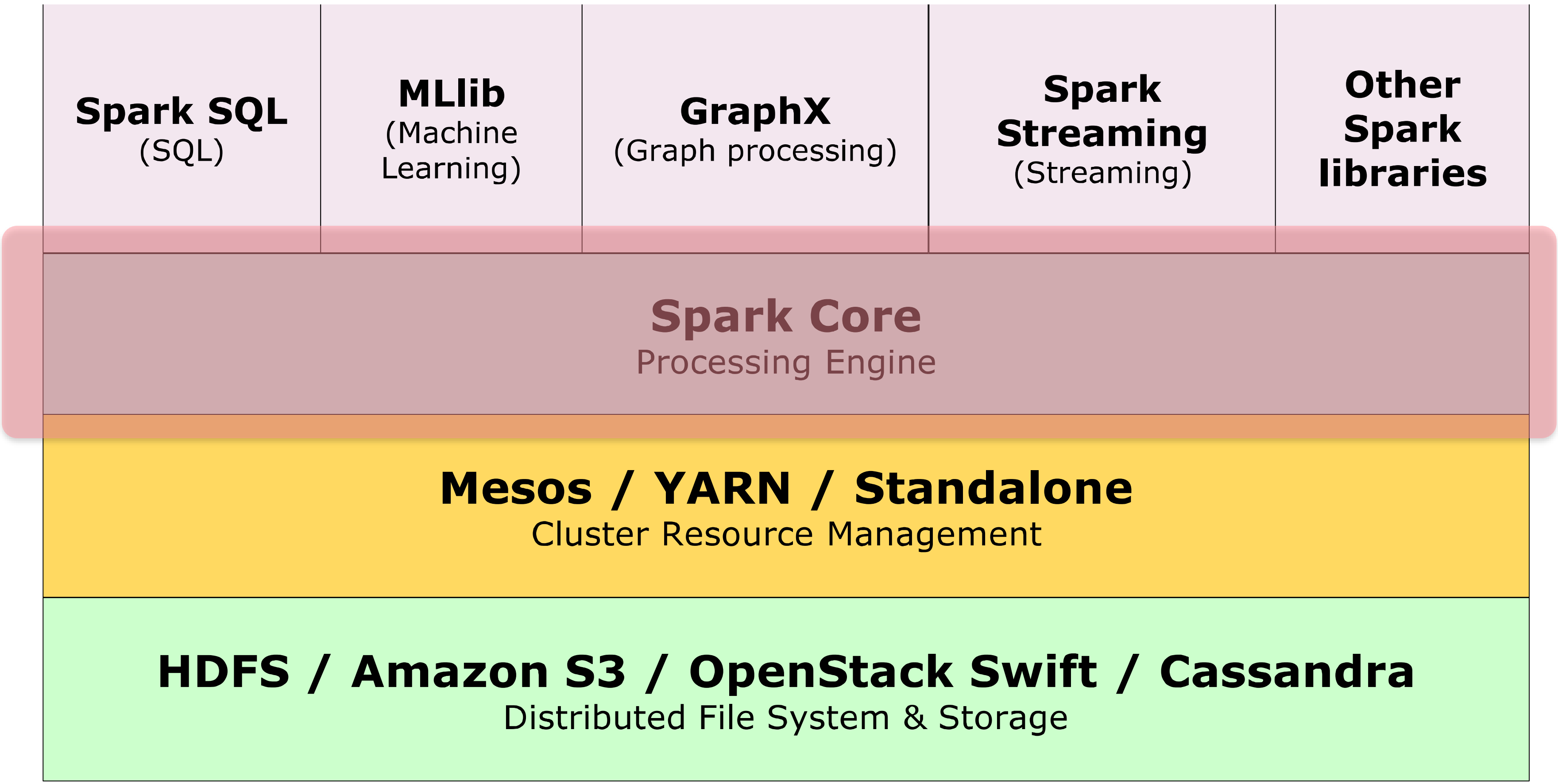
Programming for Data Science at Scale

Distributed Query Processing

THE UNIVERSITY
of EDINBURGH

Amir Shaikhha, Fall 2024

Recap: Spark Software Stack



Recap: Programming Models

- Spark vs. Hadoop MapReduce
  - More flexible programming model
  - General execution graphs
  - In-memory storage

RDD Example

- Let’s count UK students who have debt & financial dependents

```
case class Demographic(id: Int, age: Int, ...)
case class Finances(id: Int, hasDebt: Boolean, ...)

// Pair RDD (id, demographics)
val demographics = sc.textFile(...)...

// Pair RDD (id, finances)
val finances = sc.textFile(...)...
```

RDD Example

- Possibility 1

```
demographics.join(finances)
  .filter({ p =>
    p._2._1.country == "UK" &&
    p._2._2.hasFinancialDependents &&
    p._2._2.hasDebt
  }).count
```

- Steps
  1. Inner join
  2. Filter to only consider people in UK
  3. Filter to only consider people with debt & finanical depedents

RDD Example

- Possibility 2

```
val filtered = finances.filter({p =>
    p._2.hasFinancialDependents &&
    p._2.hasDebt })
demographics.filter( p => p._2.country == "UK")
   .join(filtered)
   .count
```

- Steps
  1. Filter to only consider people with debt & finanical depedents
  2. Filter to only consider people in UK
  3. Inner join on smaller datasets

RDD Example

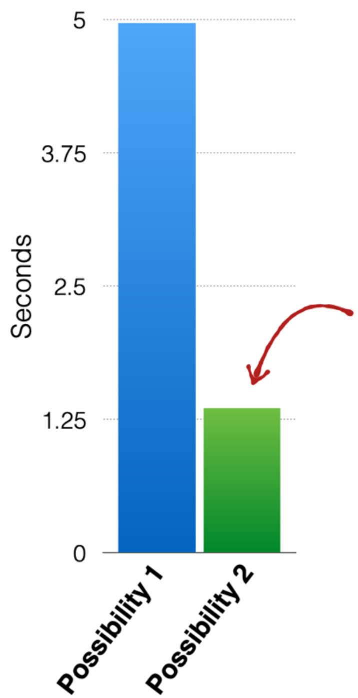
- Possibility 3

```
val cart = demographics.cartesian(finances)
cart.filter(p => p._1._1 == p._2._1)
  .filter({ p =>
    p._1._2.country == "UK" &&
    p._2._2.hasFinancialDependents &&
    p._2._2.hasDebt
  }).count
```

- Steps
  1. Cartesian product on both datasets
  2. Filter to only consider the pairs with the same id
  3. Filter to only consider people in UK
  4. Filter to only consider pople with debt & finanical depedents

RDD Example

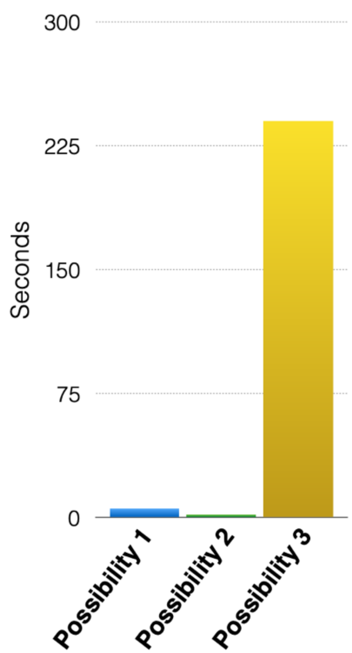
- The end result is the same for all three of these possibilities
- However, the execution time is vastly different



Filtering data first is 3.6x faster.

RDD Example

- The end result is the same for all three of these possibilities
- However, the execution time is vastly different



Cartesian product is 177x slower!

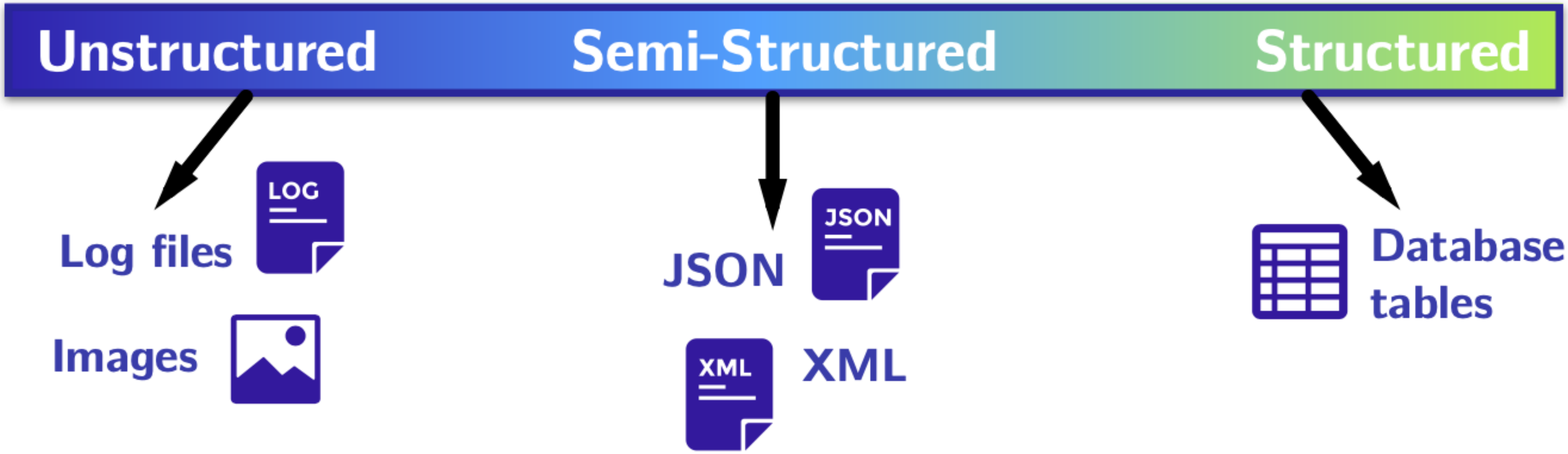
RDD Example

- So far, it was the *responsibility of the programmer* to think carefully about how Spark jobs might actually be executed cluster to get *good performance*
- Could Spark automatically rewrite the code in possibility 3 to possibility 2?

Given more structural information, Spark can do many optimizations.

Structured vs. Unstructured Data

- Data falls on spectrum from unstructured to structured.



Structured Data vs RDDs

- Spark RDDs don’t know anything about the **schema** of data
- Spark only knows that the RDD is parameterized with arbitrary types (e.g., `Person`, `Account`, `Demographic`)
- However, it doesn’t know anything about the structure of these types

Structured Data Example

- Assume a dataset of Account objects

```
case class Account(name: String, balance: Double, risk: Boolean)
```

- What Spark RDDs see:

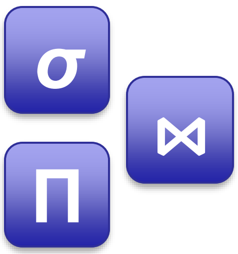
- What DBMSes see:

name: String	balance: Double	risk: Boolean
name: String	balance: Double	risk: Boolean
name: String	balance: Double	risk: Boolean
name: String	balance: Double	risk: Boolean

Structured vs Unstructured Computation

- The same can be said about **computation**.
- Spark:
  - **Functional transformations** on data.
  - Passing function literals to higher-order functions (e.g., `map`, `flatMap`, and `filter`)

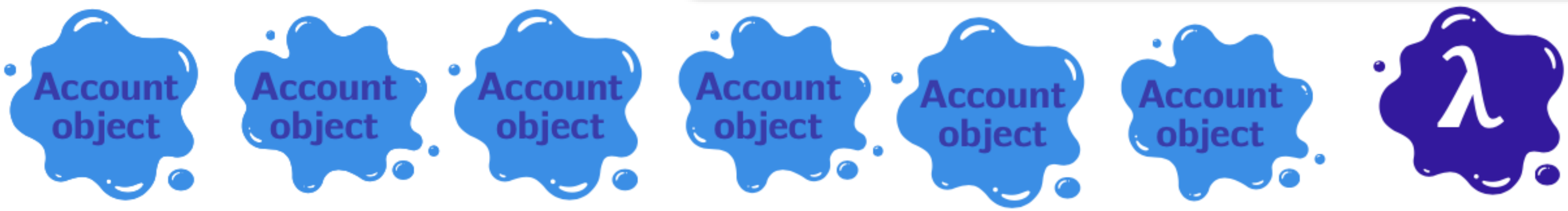
- DBMSes:
  - **Delarative transformations** on data
  - Specialized/structured, pre-defined operations



Structured vs. Unstructured

- Spark RDDs:

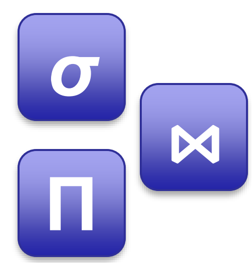
Not so much structure.
Difficult to Optimize!



- DBMSes:

Lots of structure.
Lots of optimization opportunities

name: String	balance: Double	risk: Boolean
name: String	balance: Double	risk: Boolean
name: String	balance: Double	risk: Boolean
name: String	balance: Double	risk: Boolean

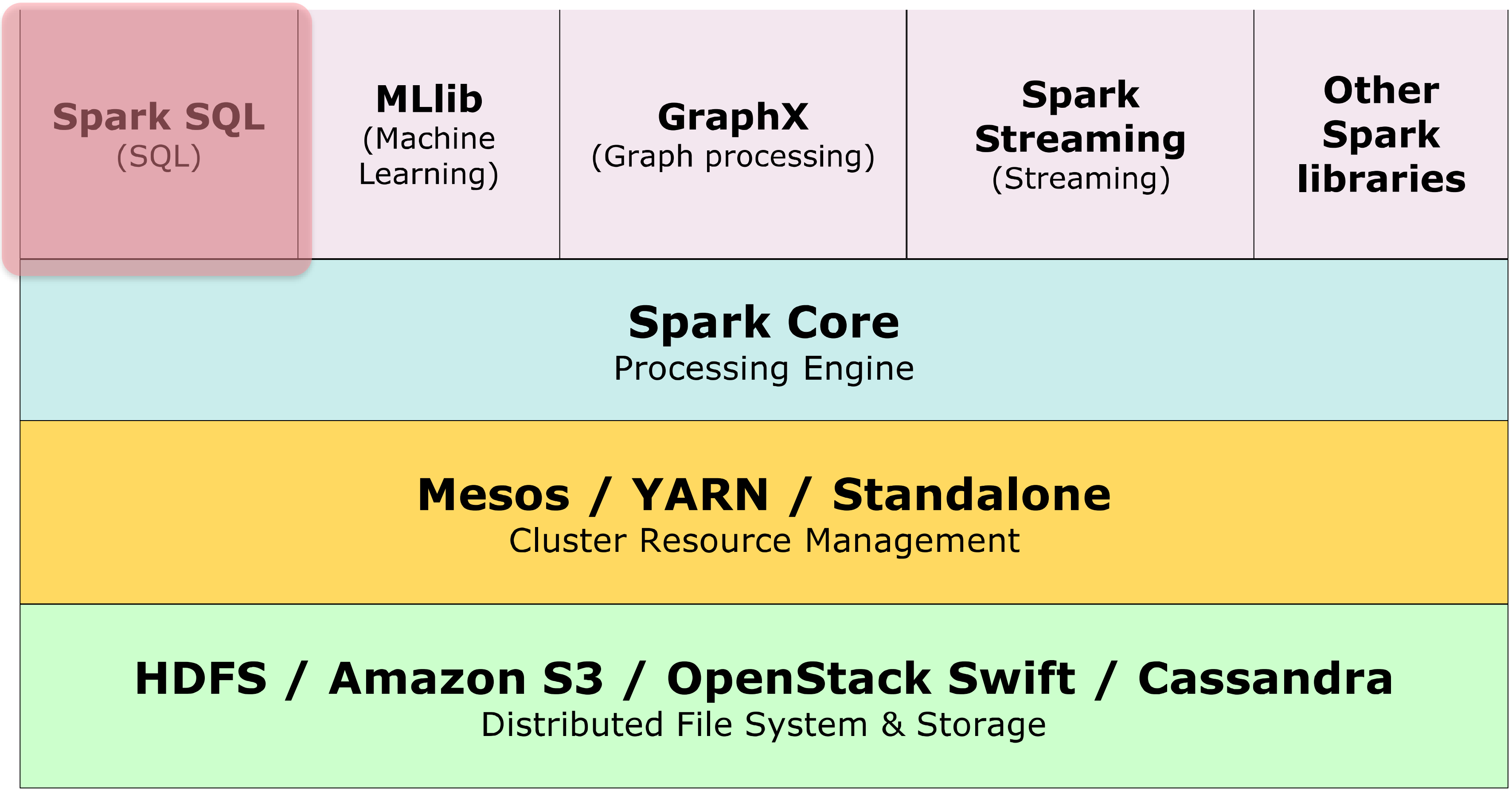


Optimizations + Spark?

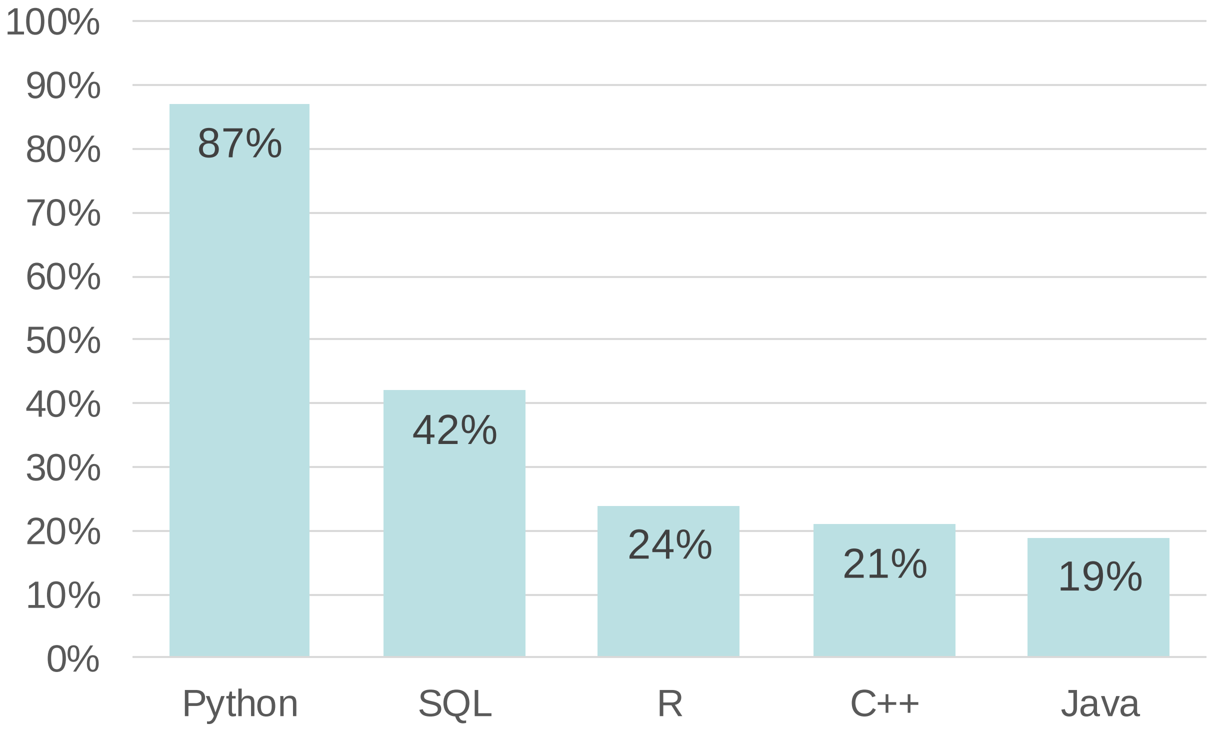
- How can Spark automatically do these optimizations?

Spark SQL

Spark Software Stack



Relational Queries (SQL)



[Kaggle Survey 2020]

Relational Queries (SQL)

- Everything about SQL is structured
- SQL = Structured Query Language
  - Fixed set of data types: Int, Long, String, etc.
  - Fixed set of operations: select, where, group by, join, etc.
- Relational databases exploit these structures to get performance speedups

Relational Queries (SQL)

- Data organized into one or more tables
- Table = Relation
  - Column=Attribute
  - Row=Record=Tuple
- Tables represent a collection of objects of a certain type

SQL for Spark

- It's hard to connect big data processing pipelines to a relational database
- It would be nice to
  - Seamlessly intermix SQL queries with Scala
  - Get all the DB optimizations on Spark jobs

Spark SQL delivers both!

Spark SQL Goals

1. Support relational processing on both Spark RDDs and on external data sources with a friendly API
2. High performance, by using techniques from the DB community
3. Support new data sources such as semi-structured data and external DBs.

Spark SQL APIs

- DataFrames
- SQL literal syntax
- Datasets

DataFrame

- Core abstraction of Spark SQL
  - Equivalent to a table in a relational DB
- DataFrame = RDD + schema
- DataFrames are untyped!
  - Scala compiler doesn't check the types in their schema
  - Transformations are untyped.

Creating DataFrames

- From RDDs
  - Inferring schema
  - Explicitly specifying schema
- Reading a data source from file

Creating DataFrames (cont.)

- From RDDs
  - Inferring schema

```
val rowRDD = ...
// DataFrame by inferring schema
val peopleDF = spark.createDataFrame(rowRDD)
```

  - Explicitly specifying schema

```
val rowRDD = ...
// DataFrame by explicitly specifying schema
val peopleDF = spark.createDataFrame(rowRDD, schema)
```

SQL literal syntax

- Progammers can use SQL syntax to operate on DataFrames

```
// DataFrame by explicitly specifying schema
val peopleDF = spark.createDataFrame(rowRDD, schema)


// SQL literals are passed to sql method
spark.sql("SELECT * FROM people WHERE age > 27")
```

How to connect `people` and `peopleDF`?

SQL literal syntax (cont.)

- Progammers can use SQL syntax to operate on DataFrames

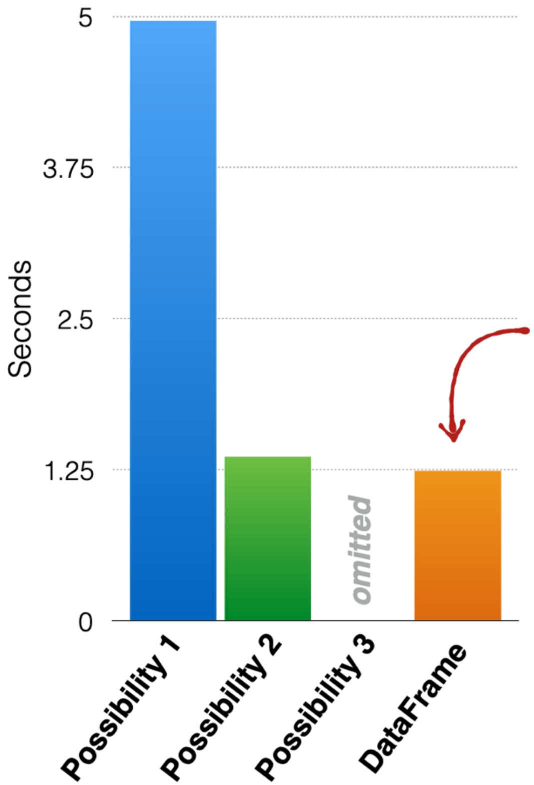
```
// DataFrame by explicitly specifying schema
val peopleDF = spark.createDataFrame(rowRDD, schema)
// Register the DataFrame as a SQL temporary view
peopleDF.createOrRepalceTempView("people")
// SQL literals are passed to sql method
spark.sql("SELECT * FROM people WHERE age > 27")
```

DataFrame API

- A relational API over Spark RDDs
  - `select`
  - `where`
  - `limit`
  - `orderBy`
  - `groupBy`
  - `join`
- Can be automatically aggressively optimized

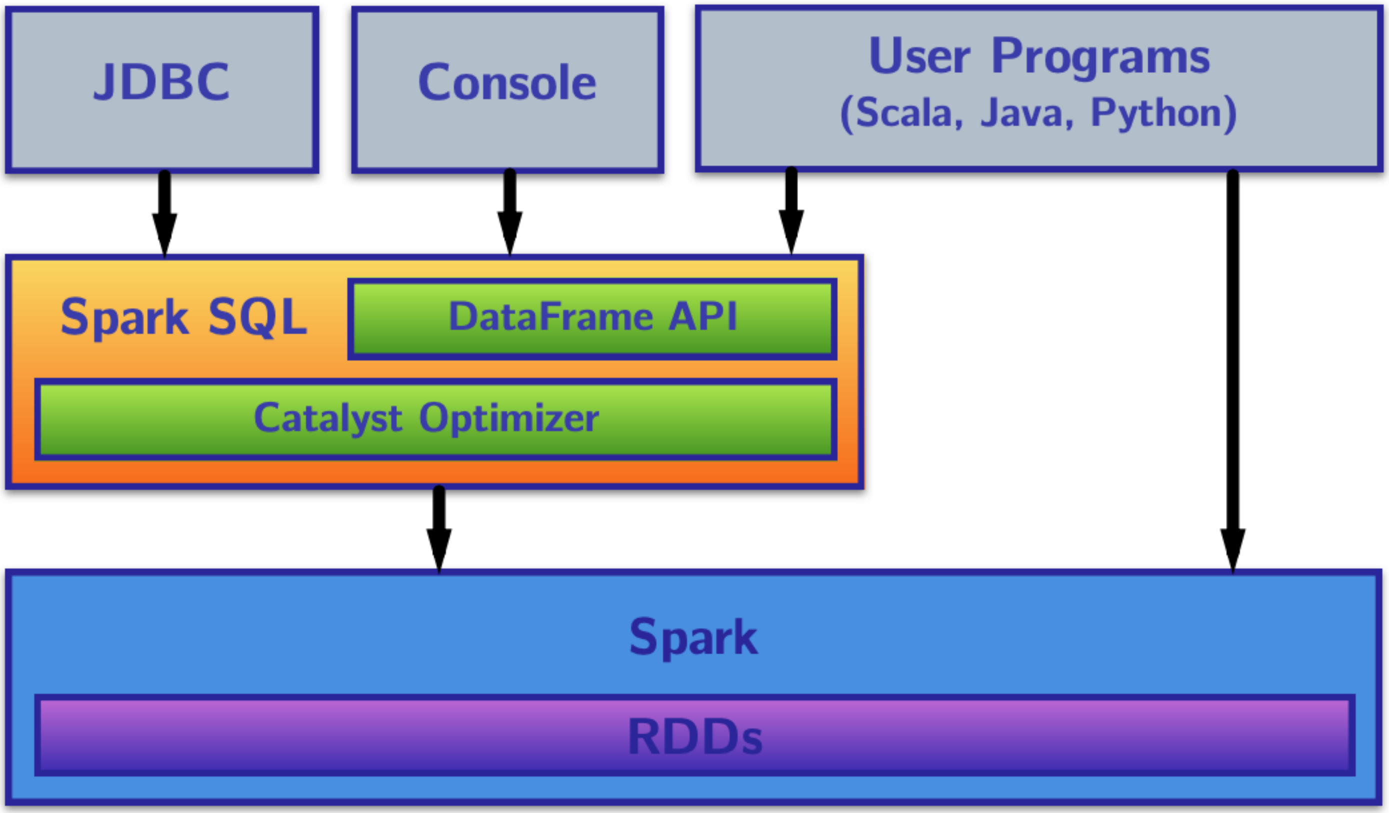
DataFrame Example

```
demographicsDF.join(financesDF,
    demographicsDF("ID") === financesDF("ID"), "inner")
  .filter($"hasDebt" && $"hasFinancialDependents")
  .filter($"country" === "UK")
  .count
```



4x faster than almost the same program written using RDDs

Spark SQL Architecture



Catalyst

- Spark SQL’s query optimizer
- Assumptions
  - Has full knowledge of all data types
  - Knows the exact schema of our data
  - Has detailed knowledge of computations
- Optimizations
  - Reordering operations
  - Reduce the amount of data read
  - Pruning unneeded partitioning

Limitations of DataFrame

- Untyped
  - Runtime exceptions even if the code compiles
  - Would be great to catch such errors at compilation time
- Limited data types
  - Semi-structured/structured data
  - Otherwise, use RDDs

Dataset

- Typed variant of DataFrame!

```
type DataFrame = Dataset[Row]
```

- In the middle between DataFrames and RDDs
  - DataFrame operations
  - More typed operations
  - Higher-order functions like map, flatMap, filter

Limitations of Dataset

- Catalyst cannot optimize higher-order functional operations
  - Similar to RDDs
- Limited data types
  - Semi-structure/structured data
  - Otherwise, use RDDs

Dataset / DataFrame / RDD

- Use datasets when
  - Structured/semi-structured data
  - Type-safety
  - Functional APIs
  - Good performance, but not the best
- Use DataFrames when
  - Structured/semi-structured data
  - Best possible performance, automatically optimized
- Use RDDs when
  - Unstructured/complex data
  - Fine-tune and manage low-level datails of RDD computations

Resources

- Compulsory reading:
  - Spark SQL [SIGMOD’15]
    - Spark SQL: Relational data processing in Spark
- Recommended reading
  - Apache PIG [VLDB’09]
  - Shark [SIGMOD’13]
  - DyradLINQ [OSDI’08]

QUESTIONS?