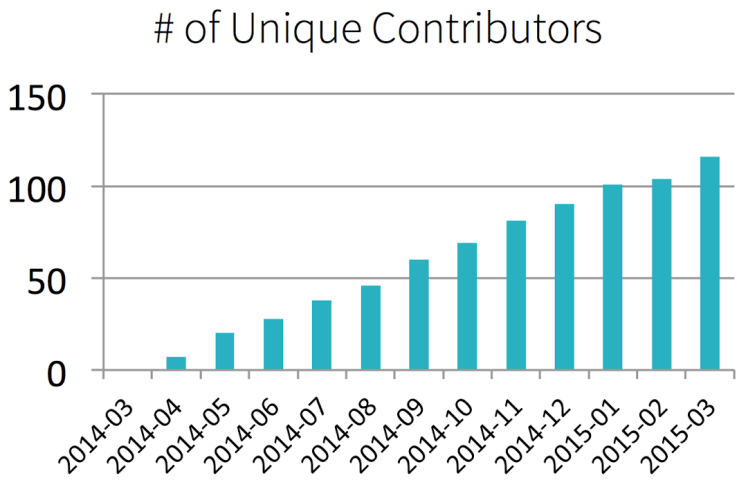
Structured Data Analysis with Spark SQL

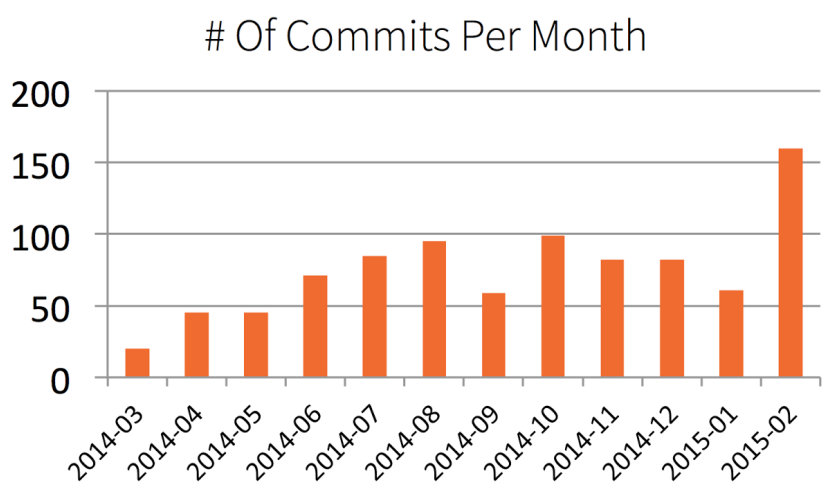
Spark Summit China 2015

Cheng Lian

Spark SQL

Part of the core distribution since Spark 1.0 (April 2014)

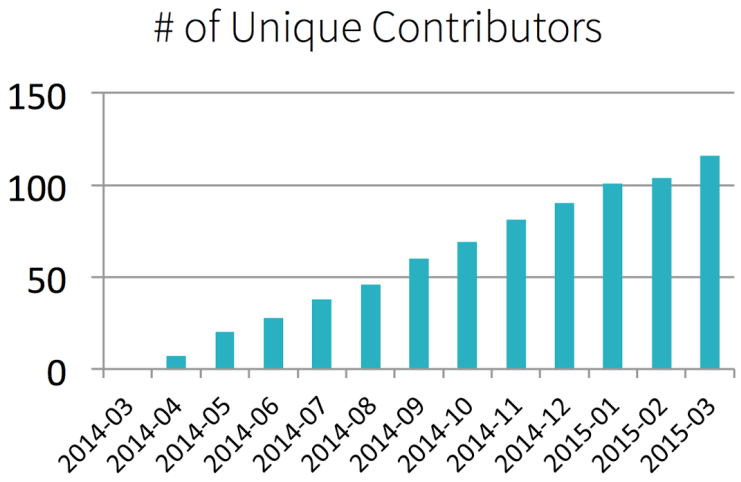


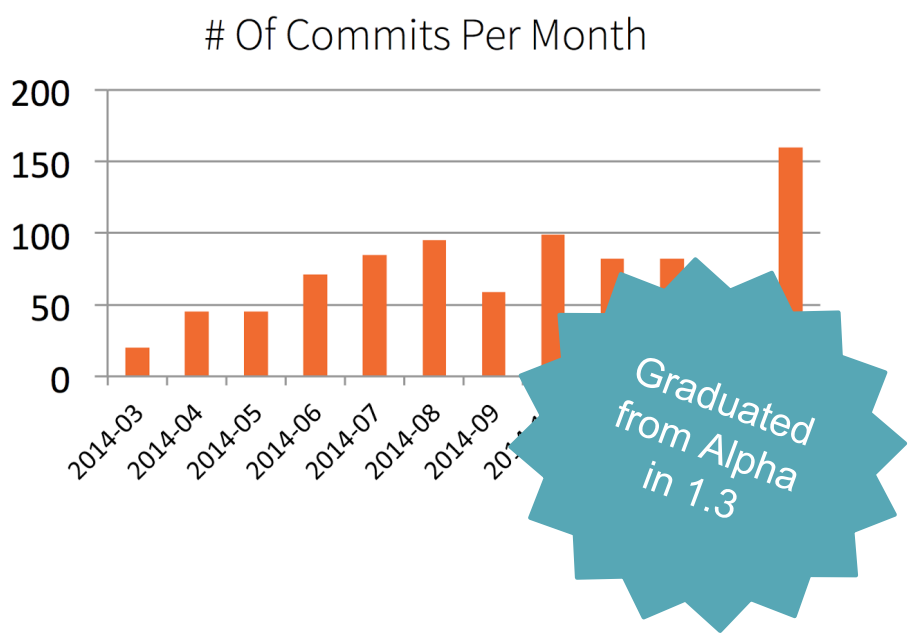


databricks™

Spark SQL

Part of the core distribution since Spark 1.0 (April 2014)





databricks

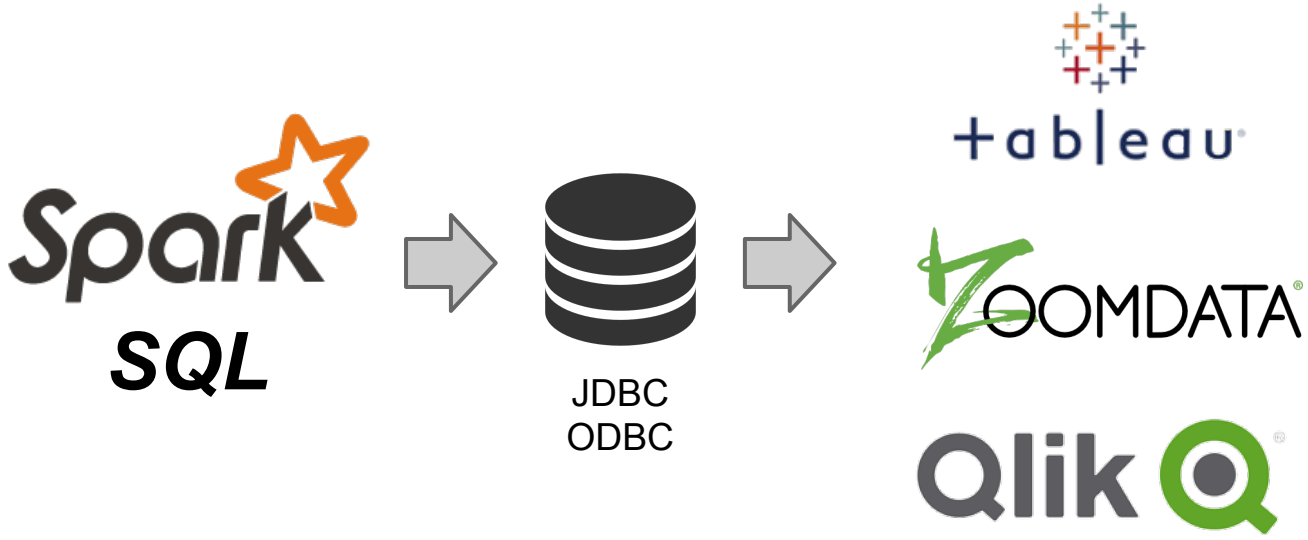
Spark SQL

Runs SQL / HiveQL queries, optionally alongside or replacing existing Hive deployments


```
SELECT COUNT(*)
FROM hiveTable
WHERE hive_udf(data)
```

Spark SQL

Connect existing BI tools to Spark through JDBC/ODBC



Spark SQL

Bindings in Python, Scala, and Java

databricks™

But... Hey

is not about SQL...

is about more than SQL

Official definition

is a Spark module for

Structured Data processing

SQL: The Whole Story

Creating and Running Spark Programs Faster:

- Write less code
- Read less data
- Let the optimizer do the hard work

DataFrame

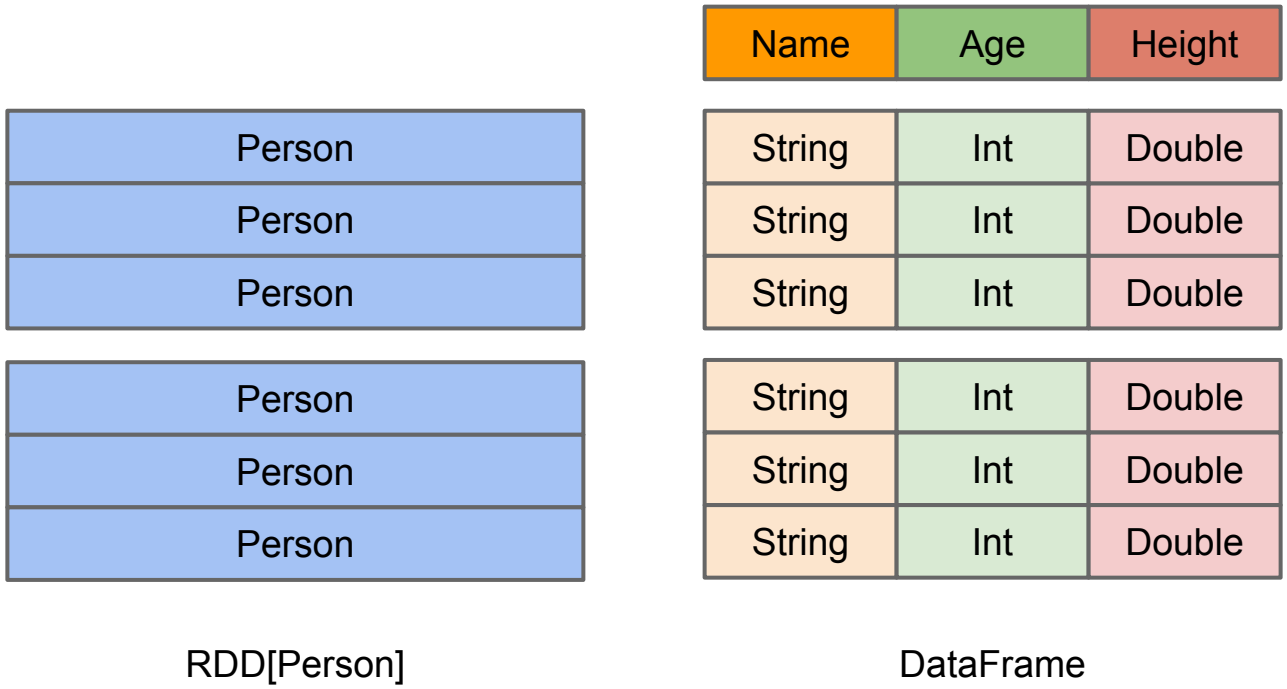
- A distributed collection of rows organized into named columns
  - Evolved from SchemaRDD (cf. Spark < 1.3)
- An abstraction for selecting, filtering, aggregating, and plotting structured data
- Inspired by R and Python Pandas
  - Single machine small data processing experiences applied to distributed big data

DataFrame

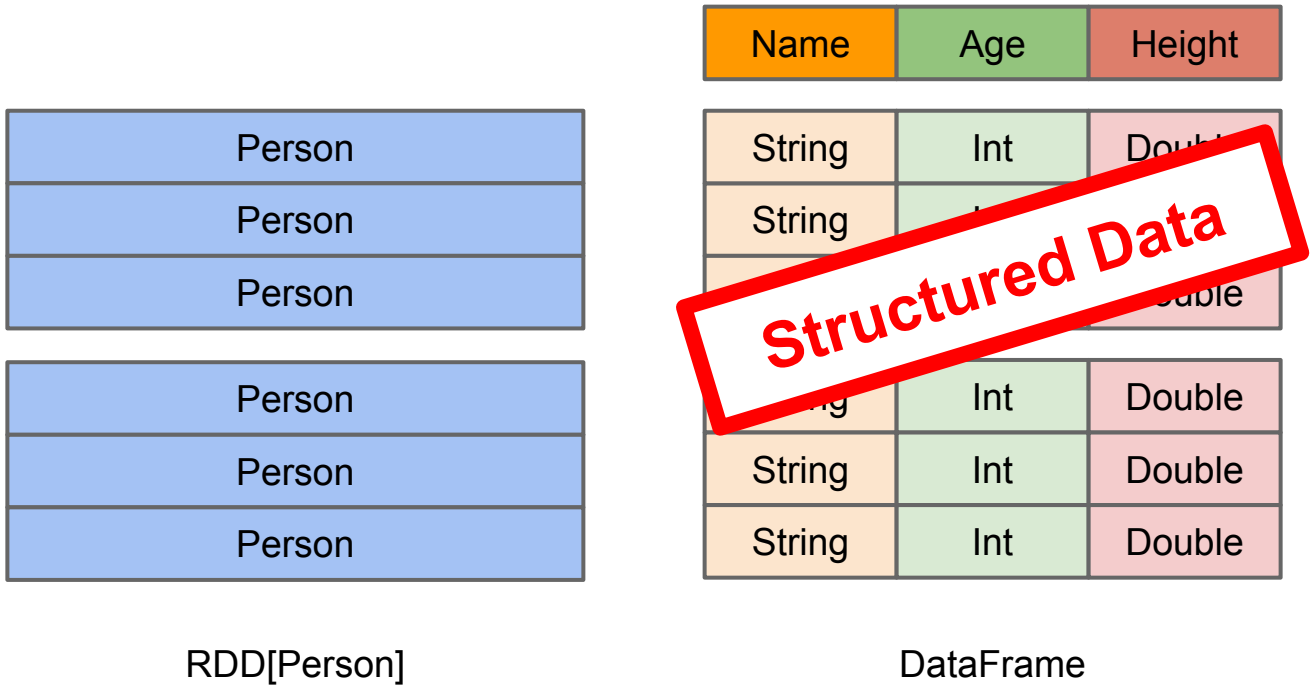
Common operations can be expressed concisely as calls to the DataFrame API

- Selecting required columns
- Joining different data sources
- Aggregation (count, sum, average, etc)
- Filtering

DataFrame v.s. RDD[T]



DataFrame v.s. RDD[T]



RDD[Person]

DataFrame

External Data Sources API

- An extensible way to integrate a variety of external data sources into Spark SQL
- Can read and write DataFrames using a variety of formats and storage systems

External Data Sources API

DataFrame

Write Less Code

```
private IntWritable one = new IntWritable(1);
private IntWritable output = new IntWritable();
protected void map(LongWritable key, Text value, Context context) {
  String[] fields = value.split("\t");
  output.set(Integer.parseInt(fields[1]));
  context.write(one, output);
}
```

```
private IntWritable one = new IntWritable(1);
private DoubleWritable average = new DoubleWritable();
protected void reduce(IntWritable key, Iterable<IntWritable> values, Context context) {
  int sum = 0;
  int count = 0;

  for(IntWritable value : values) {
    sum += value.get();
    count++;
  }

  average.set(sum / (double) count);
  context.write(key, average);
}
```


```
sc.textFile("hdfs://...")\
  .map(lambda x: (x[0], [x[1], 1]))\
  .reduceByKey(
    lambda x, y: [x[0] + y[0], x[1] + y[1]])\
  .map(lambda x: [x[0], x[1][0] / x[1][1]])\
  .collect()
```

Spark Core

```
sqlContext.table("people")\
          .groupBy("name")\
          .agg("name", avg("age"))\
          .collect()
```

Spark SQL

databricks™

Write Less Code for Clarity!

```
# ???
sqlContext.table("people")\
          .groupBy("name")\
          .agg("name", avg("age"))\
          .collect()
```

Write Less Code for Clarity!

```
# Compute an average
sqlContext.table("people")\
          .groupBy("name")\
          .agg("name", avg("age"))\
          .collect()
```

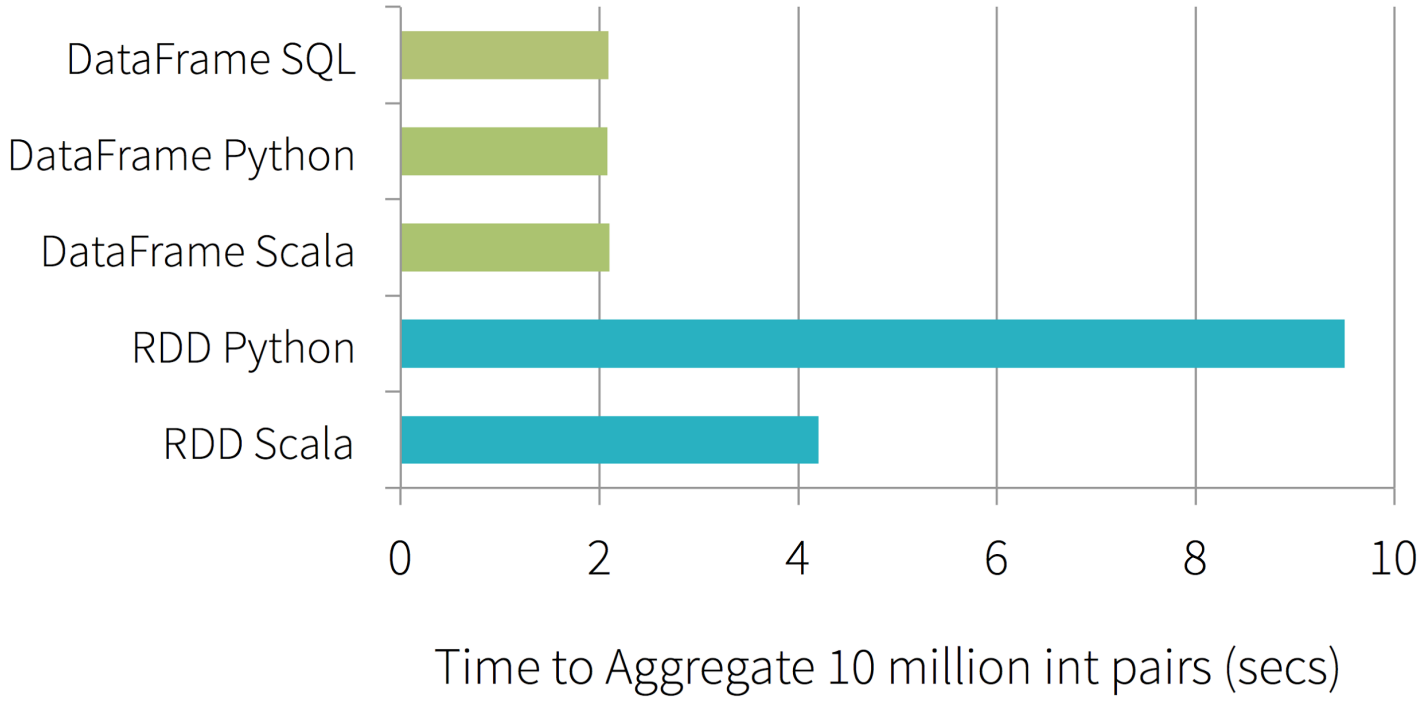
Write Less Code for Clarity!

```
# Compute an average
sqlContext.table("people")\
          .groupBy("name")\
          .agg("name", avg("age"))\
          .collect()
```

Full API Docs:

- Python
- Scala
- Java

Write Less Code and Run Faster!

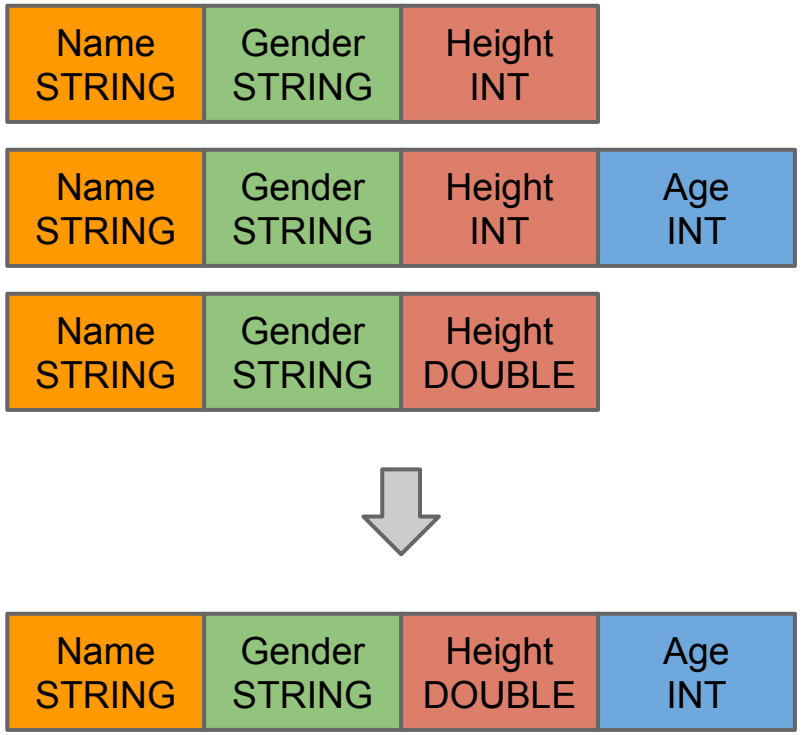


Eliminate Boilerplate Code

Schema inference

- Big data tends to be dirty
- Infer schema from semi-structured data (i.e. JSON)
- Merge different but compatible versions of schema (i.e. JSON, Parquet)

```
{“Name”: “Alice”, “Gender”: “F”, “Height”: 160}
{“Name”: “Bob”,   “Gender”: “M”, “Height”: 175, “Age”: 20}
{“Name”: “Cavin”, “Gender”: “M”, “Height”: 180.3}
```



Name STRING	Gender STRING	Height DOUBLE	Age INT
Alice	F	160	null
Bob	M	175	20
Cavin	M	180.3	null

databricks™

Eliminate Boilerplate Code

Automatic partition discovery

- Discover Hive style partitioned table directory layout
- Infer partition column types and values from partition directory paths

The fastest way to process big data is to

The fastest way to process big data is to

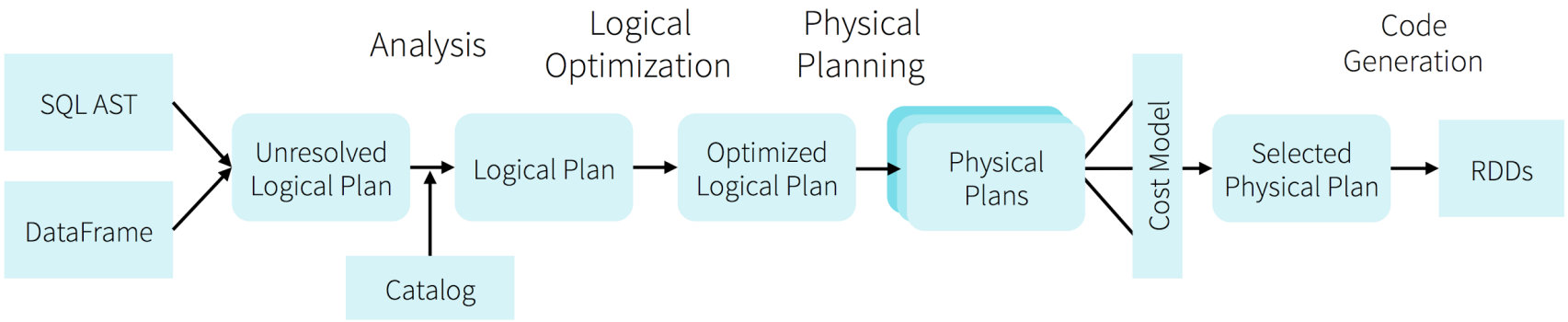
IGNORE it

Read Less Data

Spark SQL can help you:

- Converting to more efficient data formats
- Using columnar formats (Parquet)
- Leveraging Hive style partitioning (i.e. **`/year=2015/month=04/...`**)
- Skipping data using min/max statistics
- Pushing predicates into intelligent storage systems (i.e. Parquet on HDFS, JDBC)

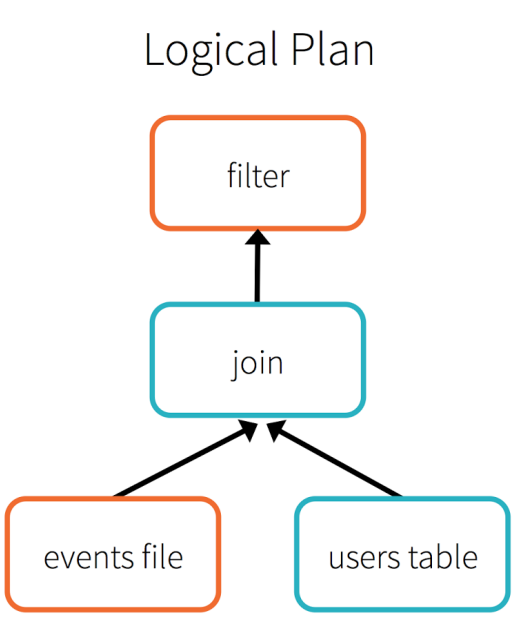
Plan Optimization and Execution

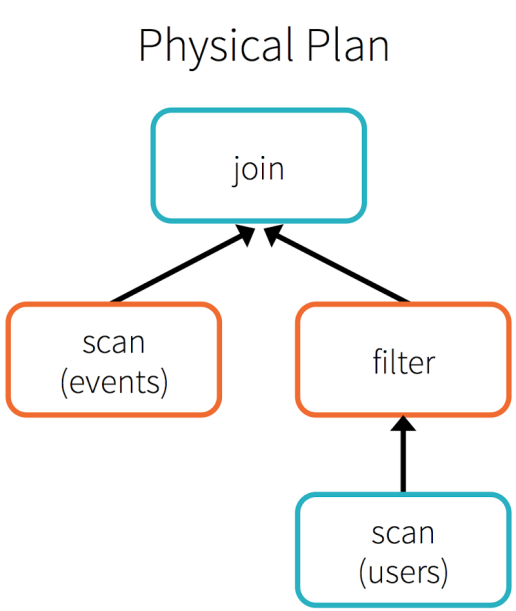


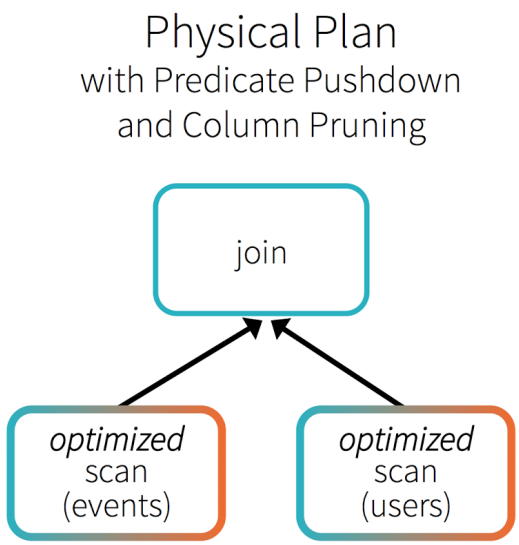
DataFrames and SQL share the same optimization/execution pipeline

```
def add_demographics(events):
  u = sqlContext.table("users")                    # Load partitioned Hive Table
  events\
    .join(u, events.user_id == u.user_id)\         # Join on user_id
    .withColumn("city", zipToCity(df.zip))         # Run UDF to add city column

events = add_demographics(sqlCtx.load("/data/events", "parquet"))
training_data = events\
  .where(events.city == "New York")\
  .select(events.timestamp).collect()
```



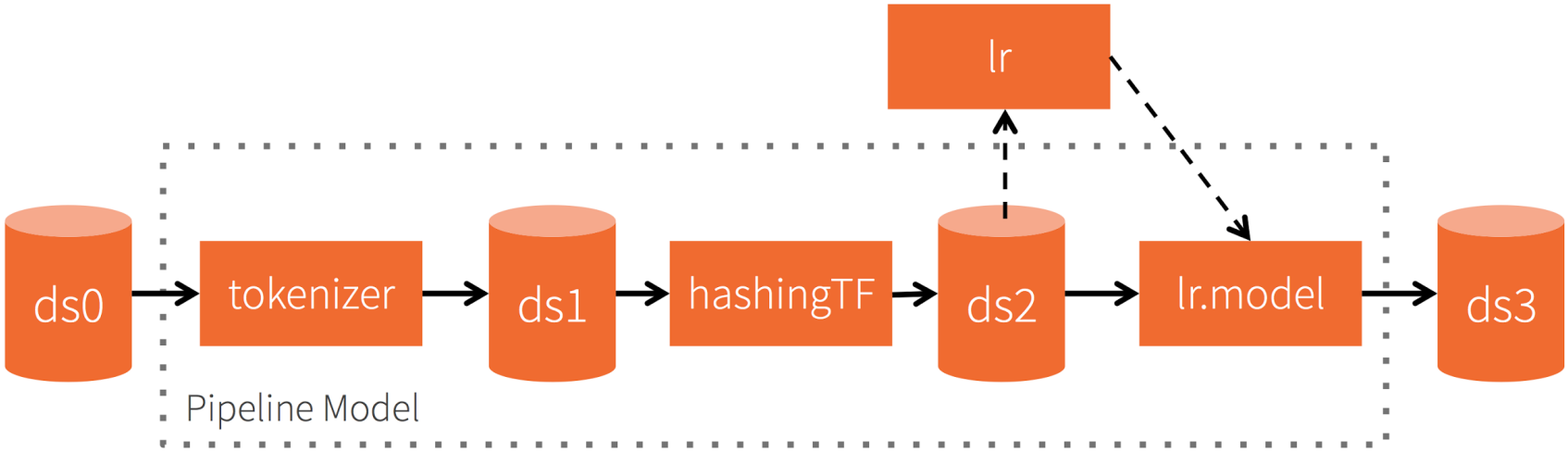




databricks™

Machine Learning Pipelines

```
tokenizer = Tokenizer(inputCol="text", outputCol="words")
hashingTF = HashingTF(inputCol="words", outputCol="features")
lr = LogisticRegression(maxIter=10, regParam=0.01)
pipeline = Pipeline(stages=[tokenizer, hashingTF, lr])
df = sqlContext.load("/path/to/data")
model = pipeline.fit(df)
```



databricks™

DataFrame as the New RDD

Compared to RDD, DataFrame

- Leverages schema information of the data for well targeted optimizations
- Provides more user friendly and intuitive APIs instead of FP style transformations
- Is well integrated with external data sources
- Is becoming efficient data sharing abstraction between different Spark modules

Create and run Spark programs faster:

- Write less code
- Read less data
- Let the optimizer do the hard work

Thanks!

Q & A