SparkSQL

Structured big-data processing in Spark

Structured Data Processing

- Spark RDD is good for general-purpose processing
- For (semi-)structured data, you need to provide your own parser and logic
- Due to the popularity of (semi-) structured data processing, SparkSQL was added to facilitate this task

Shark (Spark on Hive)

- A small side project that aimed to running RDD jobs on Hive data using HiveQL
- Still limited to the data model of Hive
- Tied to the Hadoop world

SparkSQL

- Redesigned to consider Spark query processing model
- Supports all the popular relational operators
- Can be intermixed with RDD operations
- Uses the Dataframe API which is an enhancement to the RDD API

Dataframe = RDD + schema

Dataframes

- SparkSQL's counterpart to relations or tables in RDMBS
- Consists of rows and columns
- A dataframe is **NOT** in 1NF
  - Why?
- Can be created from various data sources
  - CSV file
  - JSON file
  - MySQL database
  - Hive
  - Parquet column-format files

Dataframe Vs RDD

Dataframe

- Lazy execution
- Spark is aware of the data model
- Spark is aware of the query logic
- Can optimize the query

RDD

- Lazy execution
- The data model is hidden from Spark
- The transformations and actions are black boxes
- Cannot optimize the query

Built-in operations in SprkSQL

- Filter (Selection)
- Select (Projection)
- Join
- GroupBy (Aggregation)
- Load/Store in various formats
- Cache
- Conversion between RDD (back and forth)

SparkSQL Examples

Project Setup

```
# In dependencies pom.xml
<!--
https://mvnrepository.com/artifact/org.apache.spark
/spark-sql -->
<dependency>
    <groupId>org.apache.spark</groupId>
    <artifactId>spark-sql_2.12</artifactId>
    <version>3.1.1</version>
</dependency>
```

Code Setup

```
val conf = new SparkConf()
val sparkSession = SparkSession.builder().config(conf)
  .appName("test")
  .master("local")
  .getOrCreate()


//val sparkContext = sparkSession.sparkContext
// Read CSV file
val log_file = sparkSession.read
  .option("delimiter", "\t")
  .option("header", "true")
  .option("inferSchema", "true")
  .csv("nasa_19950801.tsv")
```

Inspect Relations

log_file.show()

```
+--------------------+-------+---------+------+--------------------+--------+------+-------+---------+
|                host|logname|     time|method|                 url|response| bytes|referer|useragent|
+--------------------+-------+---------+------+--------------------+--------+------+-------+---------+
|pppa006.compuserv...|      -|807256800|   GET|/images/launch-lo...|     200|  1713|   null|     null|
|  vcc7.langara.bc.ca|      -|807256804|   GET|/shuttle/missions...|     200|  8677|   null|     null|
|pppa006.compuserv...|      -|807256806|   GET|/history/apollo/i...|     200|  1173|   null|     null|
|thing1.cchem.berk...|      -|807256870|   GET|/shuttle/missions...|     200|  4705|   null|     null|
|       202.236.34.35|      -|807256881|   GET|     /whats-new.html|     200| 18936|   null|     null|
|bettong.client.uq...|      -|807256884|   GET|/history/skylab/s...|     200|  1687|   null|     null|
|       202.236.34.35|      -|807256884|   GET|/images/whatsnew.gif|     200|   651|   null|     null|
|       202.236.34.35|      -|807256885|   GET|/images/KSC-logos...|     200|  1204|   null|     null|
|bettong.client.uq...|      -|807256900|   GET|/history/skylab/s...|     304|     0|   null|     null|
```

log_file.printSchema()

```
root
 |-- host: string (nullable = true)
 |-- logname: string (nullable = true)
 |-- time: integer (nullable = true)
 |-- method: string (nullable = true)
 |-- url: string (nullable = true)
 |-- response: integer (nullable = true)
 |-- bytes: integer (nullable = true)
 |-- referer: string (nullable = true)
 |-- useragent: string (nullable = true)
```

Filter Example

```
// Select and count OK lines
val ok_lines = log_file.filter("response=200")
val ok_count = ok_lines.count()
println(s"Number of ok lines is ${ok_count}")
```

```
Number of ok lines is 27972
```

SQL Example

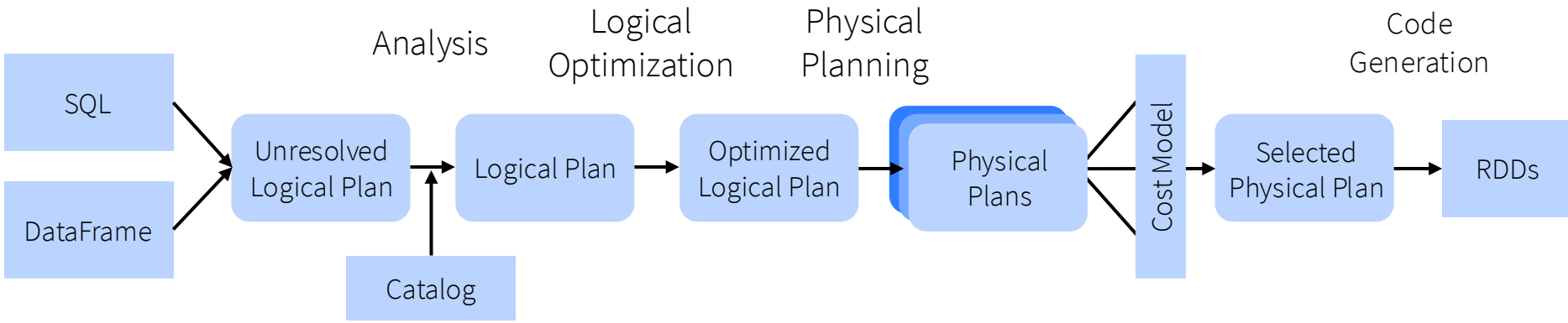
```
log_file.createOrReplaceTempVie
w("log_lines")
log_file.sqlContext.sql("""
  SELECT response, sum(bytes)
  FROM log_lines
  GROUP BY response
  """)
  .show()
// Cleanup
sparkSession.stop()
```

```
+--------+----------+
|response|sum(bytes)|
+--------+----------+
|     404|         0|
|     200| 481974462|
|     304|         0|
|     302|     26005|
+--------+----------+
```

SparkSQL Features

- Catalyst query optimizer
- Code generation
- Integration with high-level libraries

SparkSQL Query Plan



DataFrames and SQL share the same optimization/execution pipeline

Credits: M. Armbrust

Parsing

- Builds an initial unresolved logical plan
- Constructed from either a SQL query or Dataframe code
- Might be logically incorrect, e.g., points to wrong relation names, attribute names, or uses incorrect data types

Analysis

- Verifies that the plan is logically correct, i.e., can be executed
- Recursively applies several rules to resolve parts of the query
  - Resolve a relation
  - Resolve an attribute type
  - Propagate the name aliases and coerce data types throughout the plan

Logical Optimizations

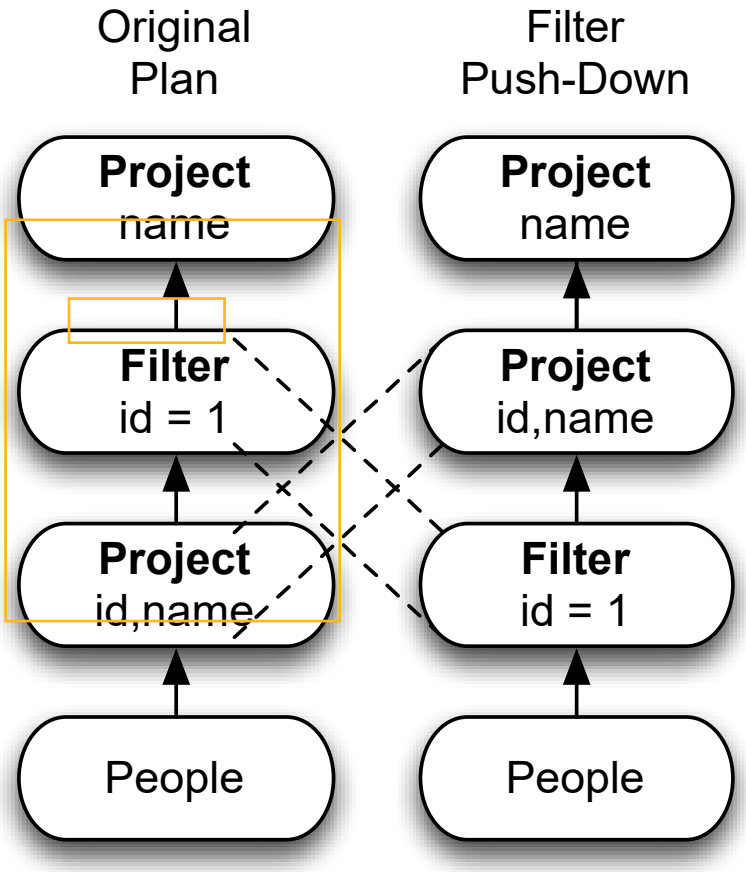
- Rule-based optimization
- Predicate push down for filters
- Projection pruning
- Simplification of wildcard and regular expression matching for strings

Catalyst Query Optimizer

- Extensible query optimizer
- Can be extended with user-provided rules
- Transforms part of a tree to another tree

Catalyst Query Optimizer

- Extensible rule-based optimizer
- Users can define their own rules



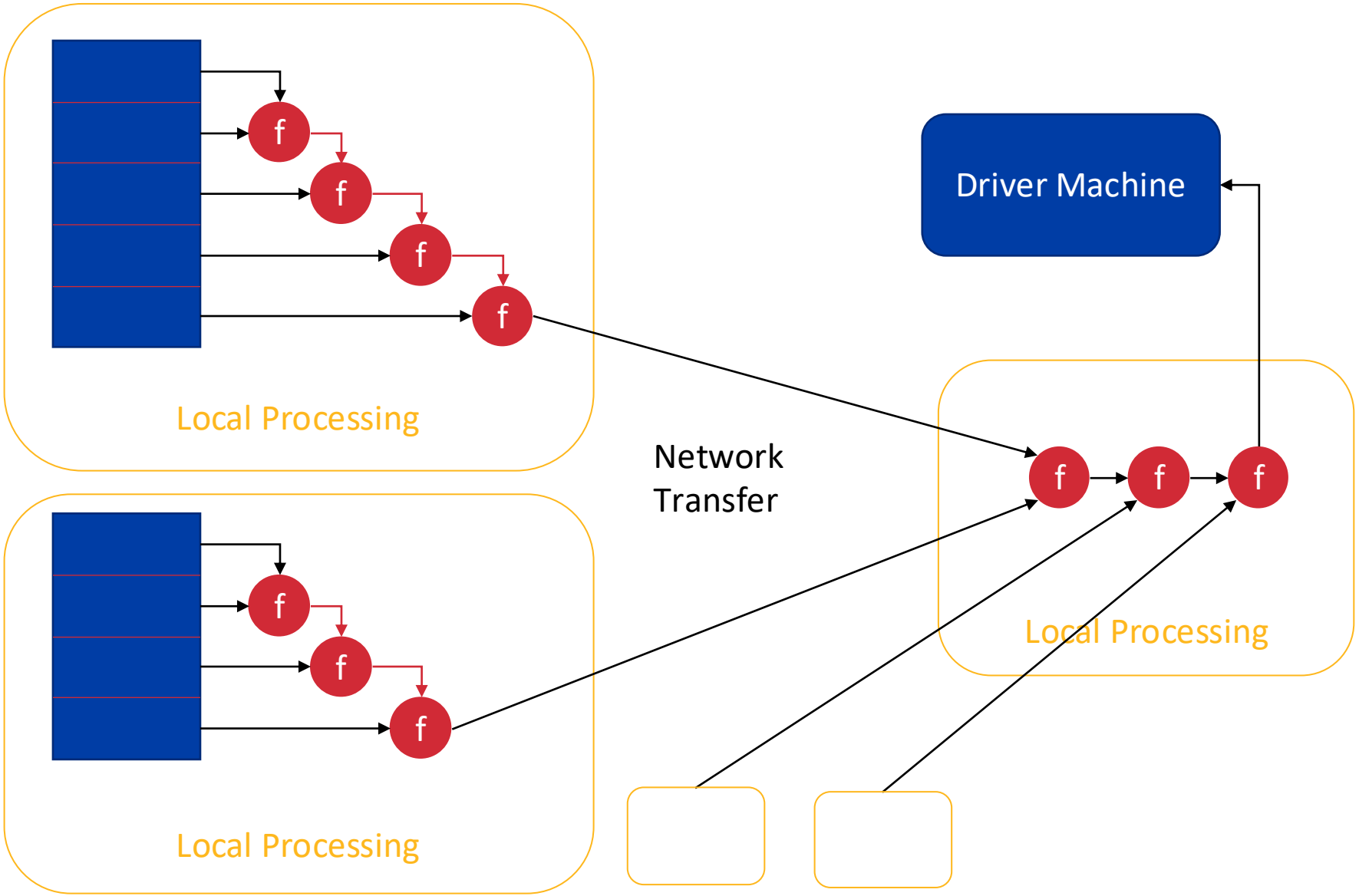
Physical Optimization

- Generates multiple physical query plans
- Applies a cost model to choose one of them
- Example, use broadcast join when one dataset is known to be very small

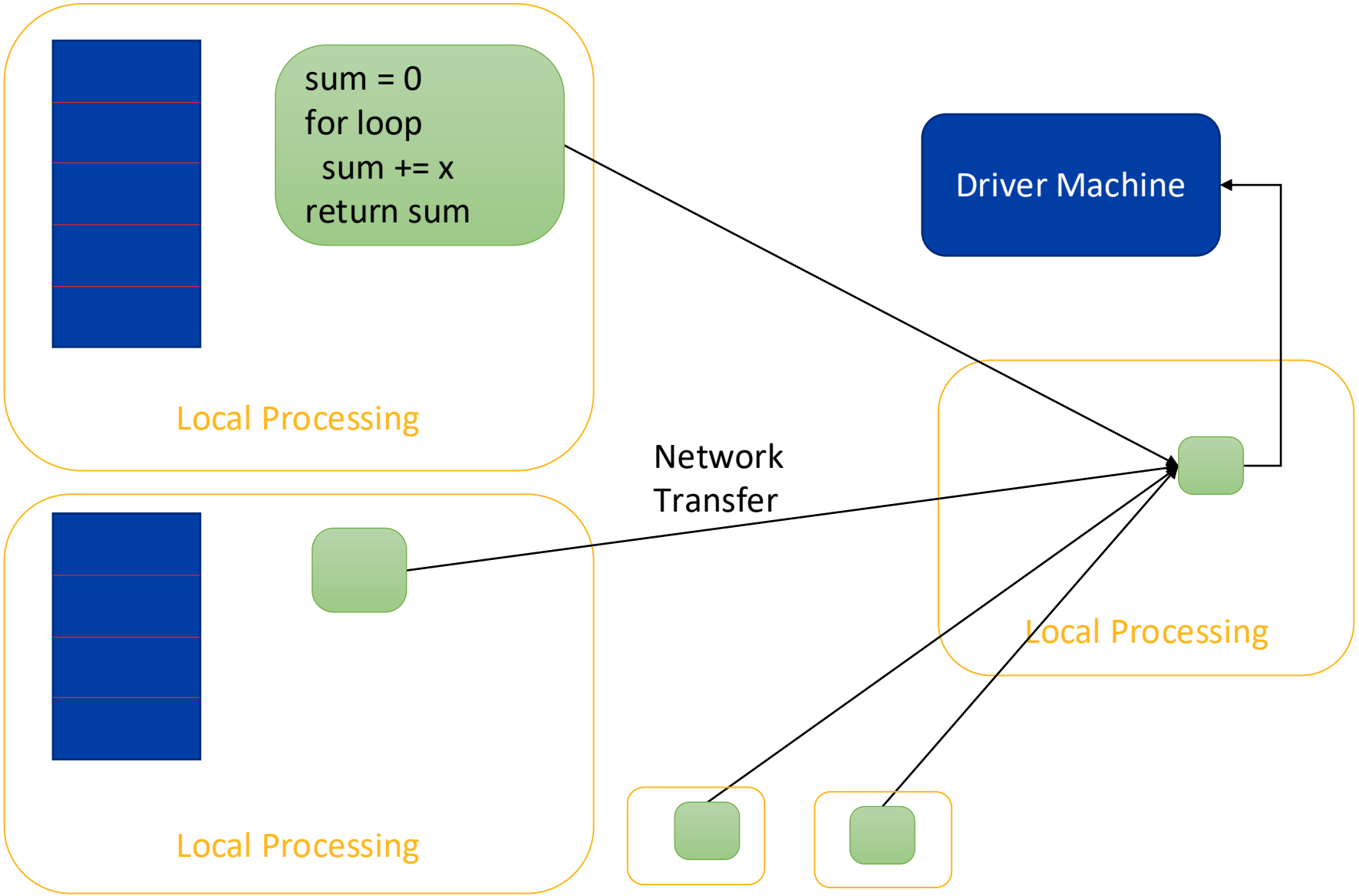
Code Generation

- Shift from black-box UDF to **Expressions**
- Example
  - `# Filter`
  - `Dataset<Row> ok_lines = log_file.filter("response=200");`
  - `# Grouped aggregation`
  - `Dataset<Row> bytesPerCode = log_file.sqlContext().sql("SELECT response, sum(bytes) from log_lines GROUP BY response");`
- SparkSQL understands the logic of user queries and rewrites them in a more concise way

Example: Aggregation (SparkRDD)



Example: Aggregation (SparkSQL)



Integration

- SparkSQL is integrated with other high-level interfaces such as MLlib, PySpark, and SparkR
- SparkSQL is also integrated with the RDD interface and they can be mixed in one program

Further Reading

- Documentation
  - http://spark.apache.org/docs/latest/sql-programming-guide.html
- SparkSQL paper
  - M. Armbrust *et al*. "Spark SQL: Relational Data Processing in Spark." SIGMOD 2015