SparkSQL

- Relational Data Processing in Spark

Presented by – Siddharth Kanojiya
Student id - 301303930

Collaborators

- Databricks Inc, MIT CSAIL, AMPLab UC Berkeley

Released in May 2014

Outline

- **What is Spark?** ⟵
- What was the motivation behind SparkSQL?
- What were the goals of SparkSQL?
- How did Spark SQL achieve those goals?
  - DataFrame API
  - Catalyst

Teaser

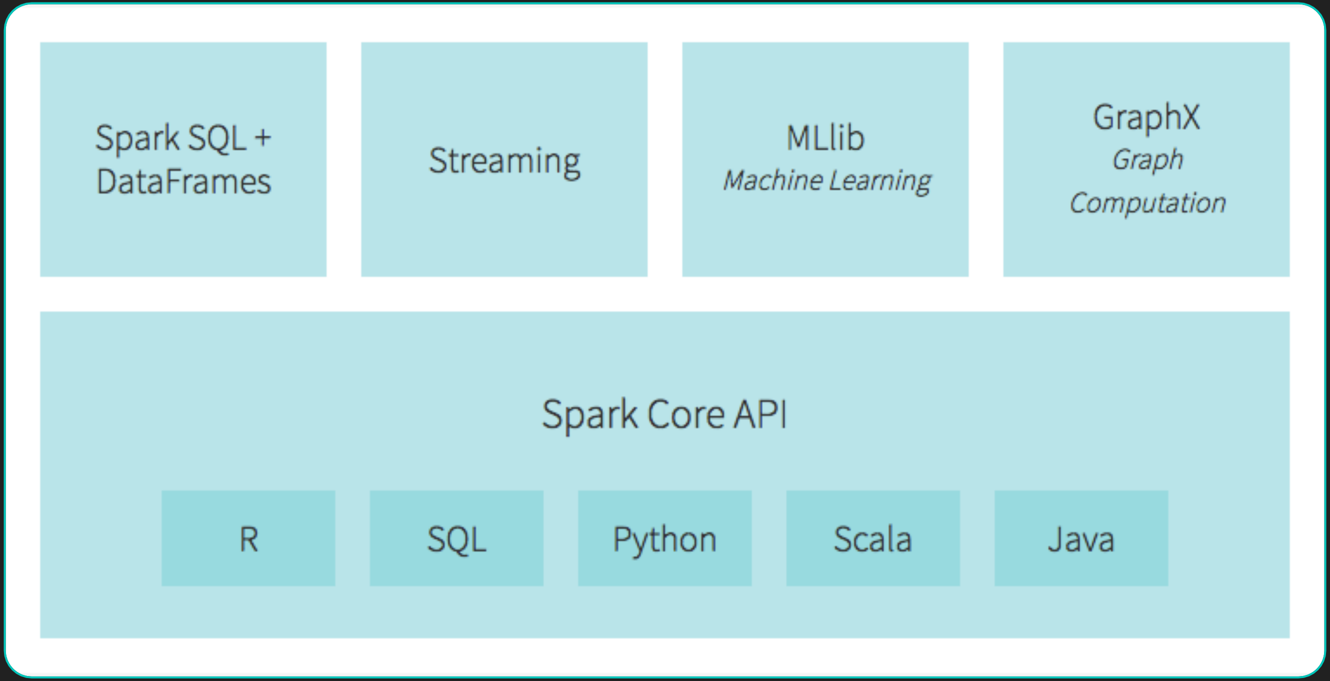
I want to transfer 1 Exabyte data each from Mumbai, Seoul and Singapore data centers to Sydney. My transfer link supports 10Gb/s.

- 1 TB = 1000 GB | 1 PB = 1000 TB | 1 ExaB = 1000 PB

How do I do it?

What is APACHE Spark™ ?

- It is a big data processing tool originally developed at UC Berkeley in 2009
- Works on the principle of distributing workloads on a cluster
- Famous applications – Machine learning, Graph processing and Streaming



Outline

- What is Spark?
- **What was the motivation behind SparkSQL?**
- What were the goals of SparkSQL?
- How did Spark SQL achieve those goals?
  - DataFrame API
  - Catalyst

Motivation behind SparkSQL

- Big data processing include:
  - ETL operations such as Join, Filter, etc. *Pref: Relational queries*
  - Complex algorithms such as Machine learning. *Pref: Procedural queries*
- Problem :
  - Forced to chose one paradigm or the other
- Solution:
  - **SparkSQL** intermixes both relational and procedural models

Outline

- What is Spark?
- What was the motivation behind SparkSQL?
- **What were the goals of SparkSQL?** ⟵
- How did Spark SQL achieve those goals?
  - DataFrame API
  - Catalyst

Goals of SparkSQL

- Support relational query within Spark programs and external data sources
- Provide high performance using established DBMS techniques
- Easily support new data sources, including semi-structured data and external databases
- Enable extension with advanced analytics such as Machine Learning, Graphs

Outline

- What was the motivation behind SparkSQL?
- What were the goals of SparkSQL?
- **How did Spark SQL achieve those goals?**
  - **DataFrame API**
  - Catalyst

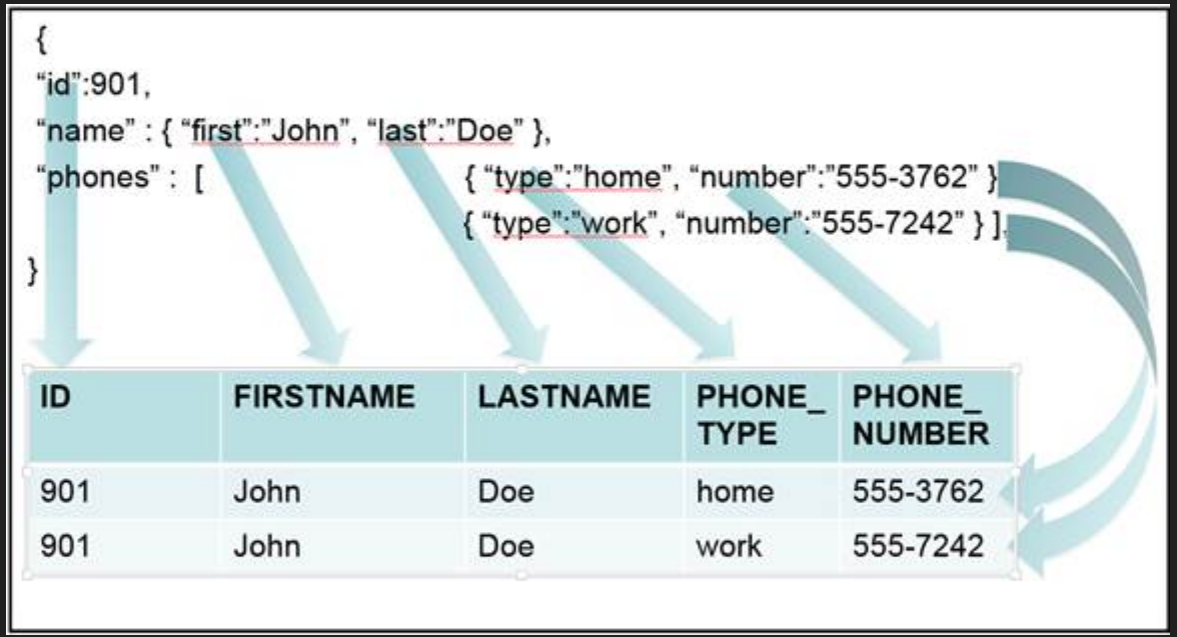
DataFrame API

What are DataFrames?

- Relational representation of data sources (for e.g. JSON)
- Schema based – Explicit or Implicit
- Can be partitioned
- Lazily evaluated
- How to create one?
  - df = spark.read.json(’/Downloads/contacts.json’,

    schema=contacts_schema,

    header=True)



```
{
"id":901,
"name" : { "first":"John", "last":"Doe" },
"phones" : [                    { "type":"home", "number":"555-3762" }
                                { "type":"work", "number":"555-7242" } ]
}
```

ID	FIRSTNAME	LASTNAME	PHONE_ TYPE	PHONE_ NUMBER
901	John	Doe	home	555-3762
901	John	Doe	work	555-7242

Ways to query DataFrame

<table>
<tr><th colspan="2"><em>Q) Find number of female employees in each department</em></th></tr>
<tr><td><strong>DSL (Domain Specific Language)
Or Procedural</strong></td><td><strong>SQL (Structured Query Language)
Or Relational</strong></td></tr>
<tr><td>employees
.join(dept, employees("deptId") === dept("id"))
.where(employees("gender") === "female")
.groupBy(dept("id"), dept("name"))
.agg(count("name"))</td><td>“SELECT count(d.name) FROM employees e
JOIN dept d
ON e.deptid = d.id
WHERE e.gender = ‘female’
GROUP BY d.id, d.name”</td></tr>
</table>

Which one do you prefer?

User Defined Functions (UDF)

- Define your own computations on DataFrames

- For e.g.

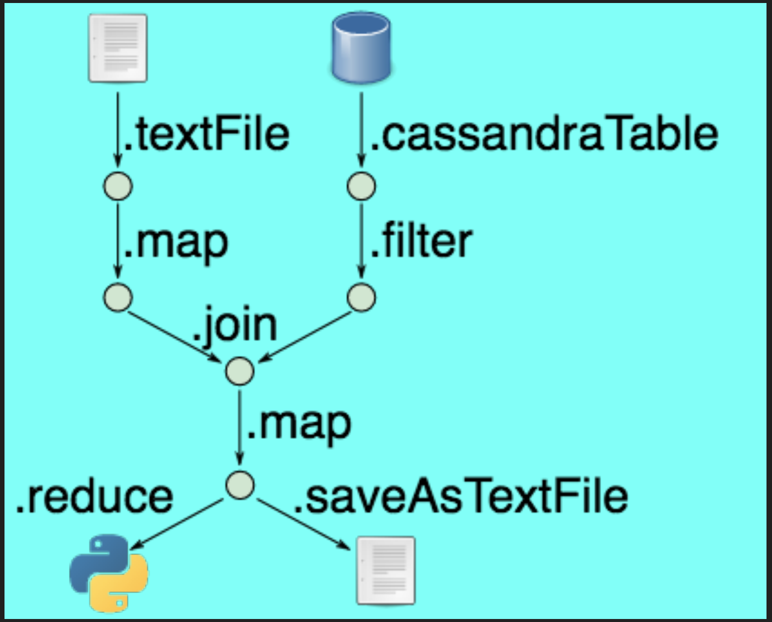
```
val model: LogisticRegressionModel = ...

ctx.udf.register("predict",
(x: Float, y: Float) => model.predict(Vector(x, y)))

ctx.sql("SELECT predict(age, weight) FROM users")
```

Lazy evaluation

- Generates a logical plan(a Directed Acyclic Graph) for set of operations on a dataframe
- No operation is actually performed until an "output" operation is requested for e.g. show(), to_csv()
- Demo

Why is it needed?

- Identify parallel and sequential tasks



Outline

- What was the motivation behind SparkSQL?
- What were the goals of SparkSQL?
- **How did Spark SQL achieve those goals?**
  - DataFrame API
  - **Catalyst** ⟵

Catalyst – a Query Optimizer

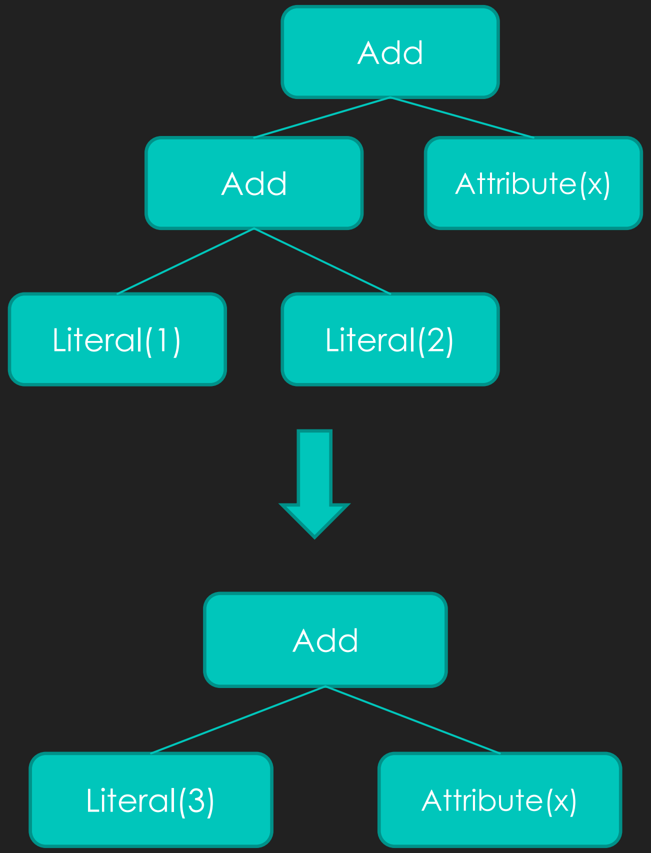
Why was Catalyst introduced?

1. Make it easy to add new optimization techniques to tackle Big Data problems such as semi-structured data
2. Let external developers write customized optimization rules specific to their data source

Catalyst : Trees and Rules

- Trees
  - Basic data type for Catalyst
  - For e.g. Add(Attribute(x), Add(Literal(1), Literal(2))
- Rules
  - For e.g. tree.transform {

```
Case Add(Literal(c1), Literal(c2)) => Literal(c1+c2)
}
```



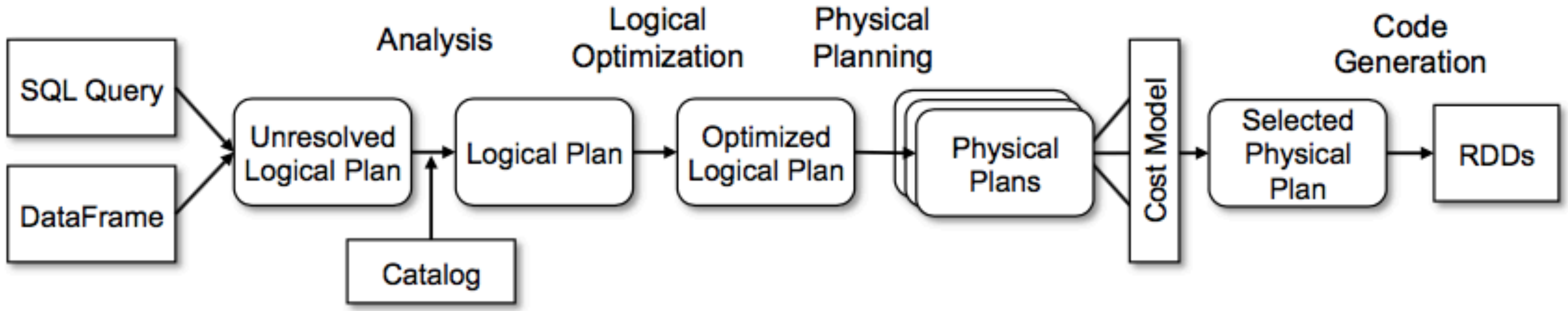
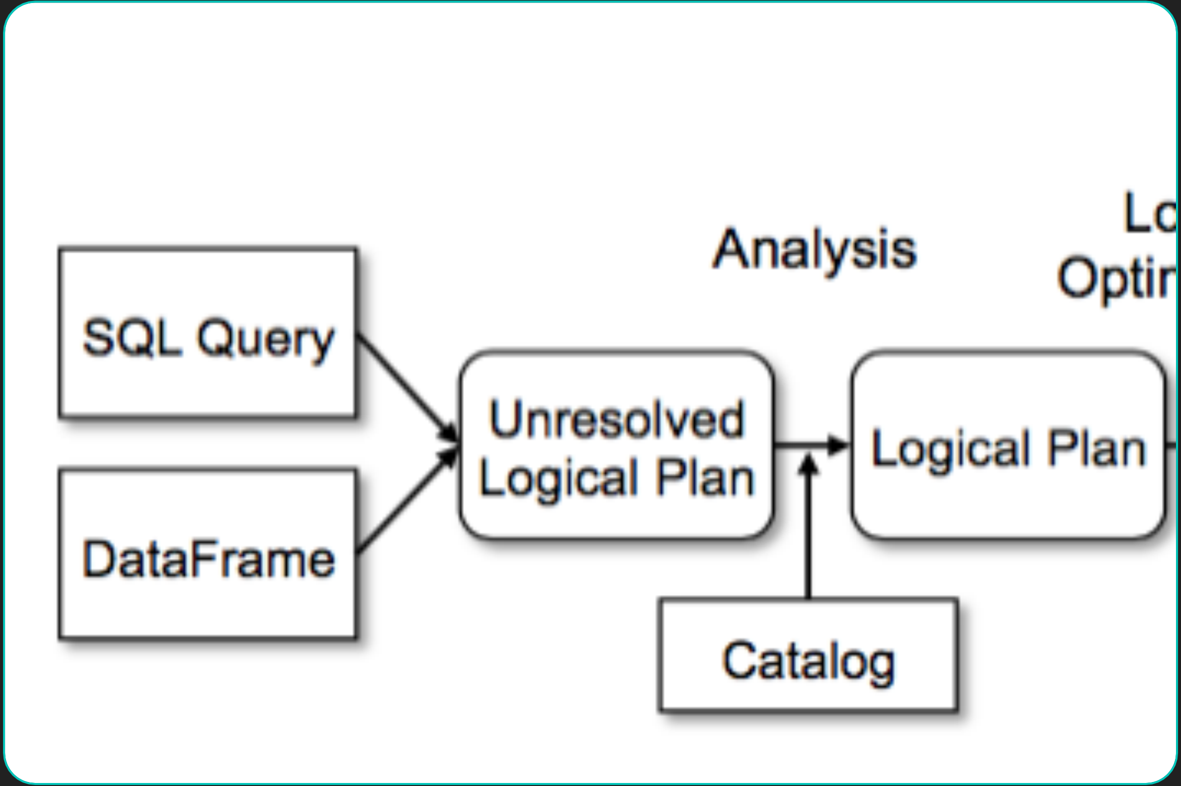


Figure 3: Phases of query planning in Spark SQL. Rounded rectangles represent Catalyst trees.

Catalyst: Optimization phases

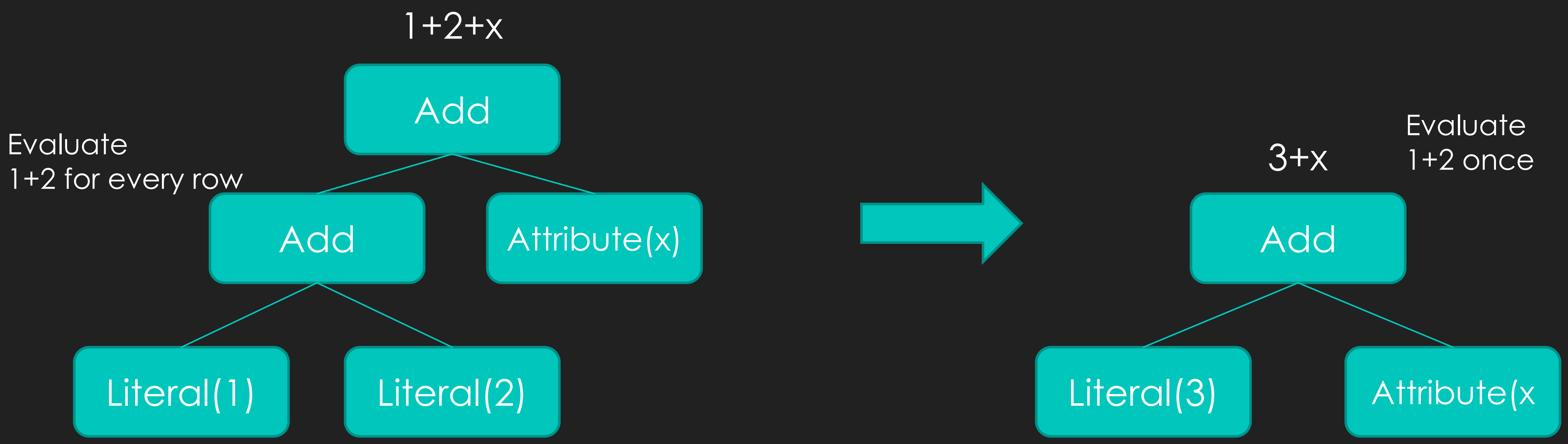
Analysis phase

- Checks for table and column names
- Verifies the data type of column and the operation requested
- Apply cast to subexpressions. For e.g. 1 + col. 1 is casted differently when col is numeric/date

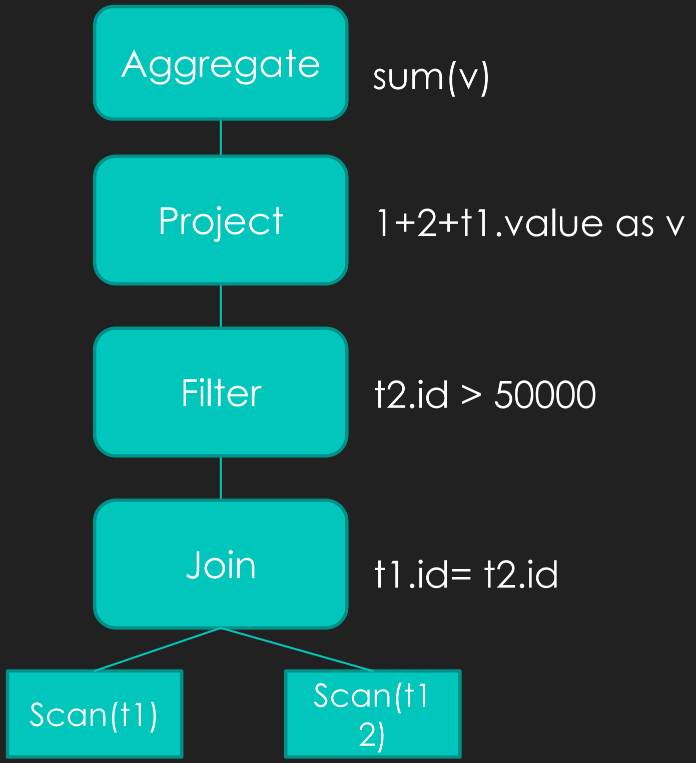


Logical Optimization phase

Constant folding



Logical Optimization phase(contd)



Aggregate sum(v)

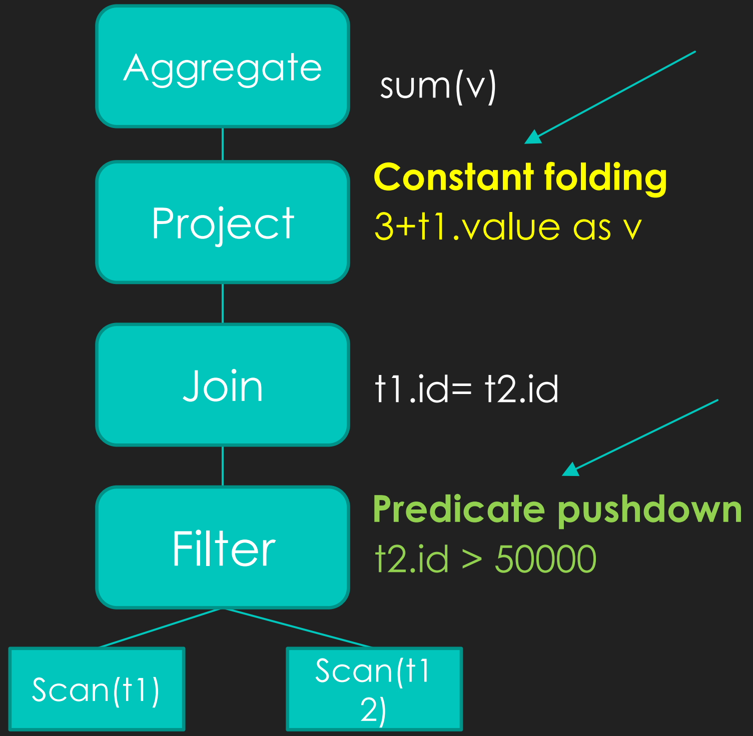
Project 1+2+t1.value as v

Filter t2.id > 50000

Join t1.id= t2.id

Scan(t1) Scan(t1 2)

User entered query



Aggregate sum(v)

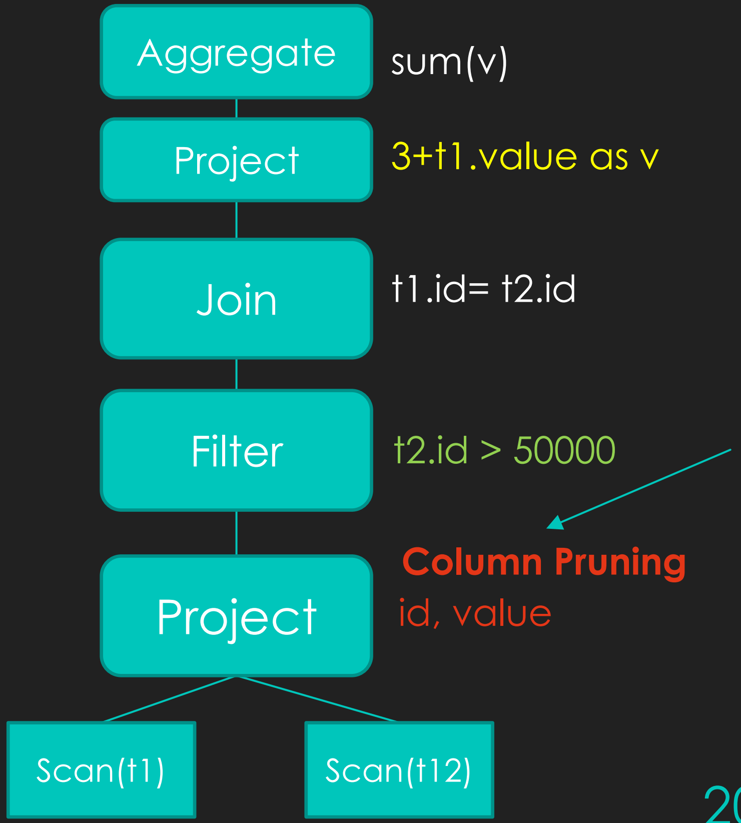
Project **Constant folding** 3+t1.value as v

Join t1.id= t2.id

Filter **Predicate pushdown** t2.id > 50000

Scan(t1) Scan(t1 2)

Optimized version



Aggregate sum(v)

Project 3+t1.value as v

Join t1.id= t2.id

Filter t2.id > 50000

Project **Column Pruning** id, value

Scan(t1) Scan(t12)

More optimized version

Query Planning: Physical Plans

- Generate multiple physical plans
- Least costly plan will be chosen
- Demo

Teaser Revisited

I want to transfer 1 Exabyte data each from Mumbai, Seoul and Singapore data centers to Sydney. My transfer link supports 10Gb/s.

- 1 TB = 1000 GB | 1 PB = 1000 TB | 1 ExaB = 1000 PB

- How do I do it?

- Video
- Solution: Catalyst (Physical Planner) with Amazon Snowball API and Amazon snowmobile.
- Further reading - https://databricks.com/blog/2017/04/01/next-generation-physical-planning-in-apache-spark.html

Thank you

Any Questions?

Questions?

- Why did Shark discontinue?
  - Can query on external sources, but sources inside a Spark program
  - Optimizer was only MapReduce-compatible. Difficult to extend new data types for Machine learning