Apache Calcite: A Foundational Framework for Optimized Query Processing Over Heterogeneous Data Sources

Edmon Begoli, Jesús Camacho-Rodríguez, Julian Hyde, Michael J. Mior, Daniel Lemire

2018 SIGMOD, Houston, Texas, USA

Outline

Background and History

Architecture

Adapter Design

Optimizer and Planner

Adoption

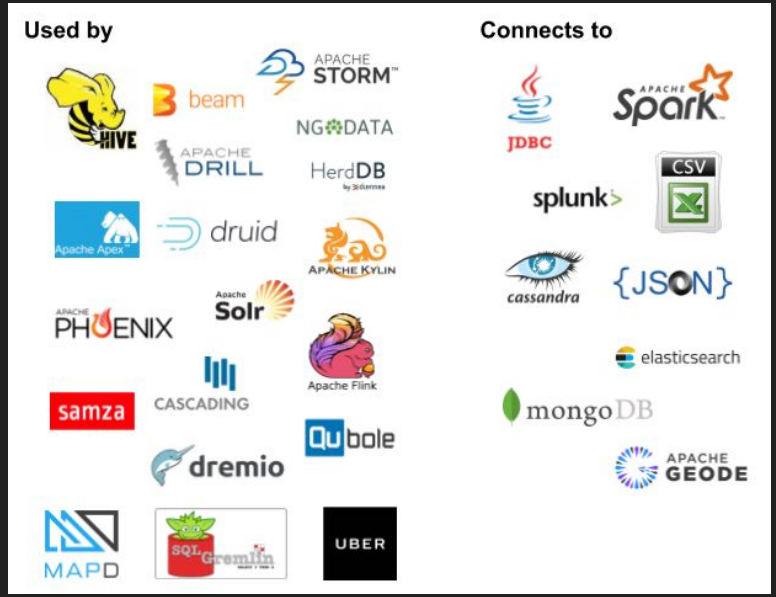
Uses in Research and Scholastic Potential

Roadmap and Future Work

What is Calcite?

Apache Calcite is an extensible framework for building data management systems.

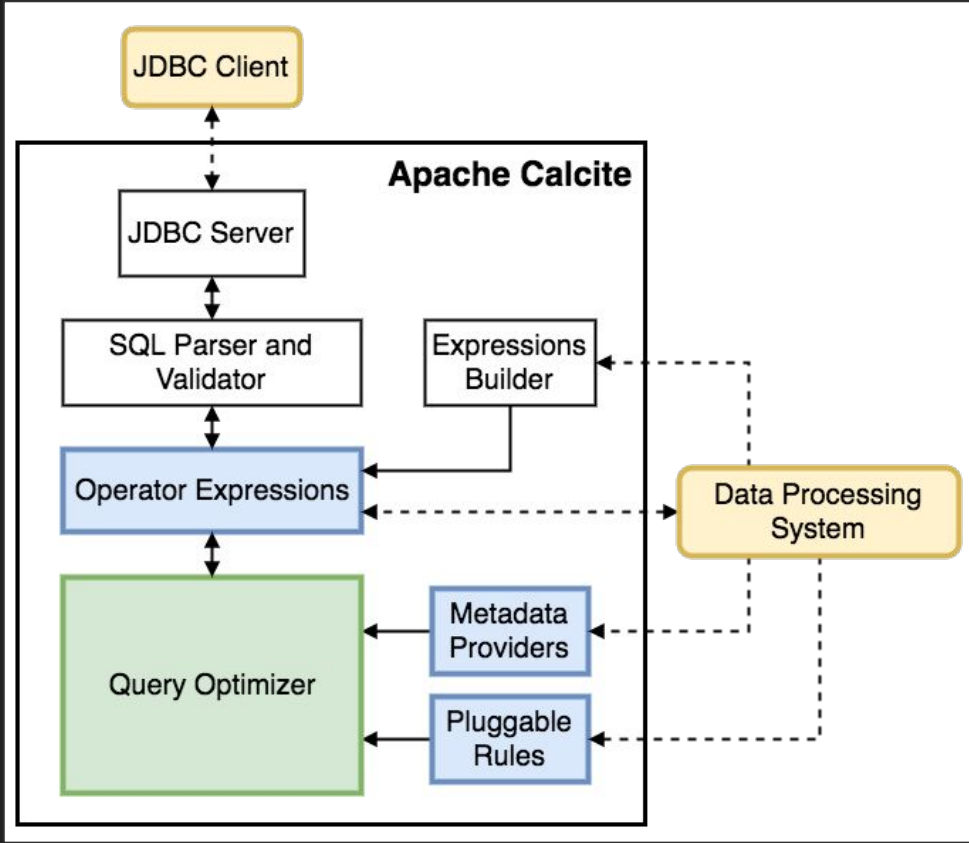
It is an open source project governed by the Apache Software Foundation, is written in **Java**, and is used by dozens of projects and companies, and several research projects.



Origins and Design Principles

Origins	2004 – LucidEra and SQLstream were each building SQL systems; 2012 – Pare down code base, enter Apache as incubator project
Problem	Building a high-quality database requires ~ 20 person years (effort) and 5 years (elapsed)
Solution	Create an open source framework that a community can contribute to, and use to build their own DBMSs
Design principles	Flexible → Relational algebra Extensible/composable → Volcano-style planner Easy to contribute to → Java, FP style
Alternatives	PostgreSQL, Apache Spark, AsterixDB

Architecture



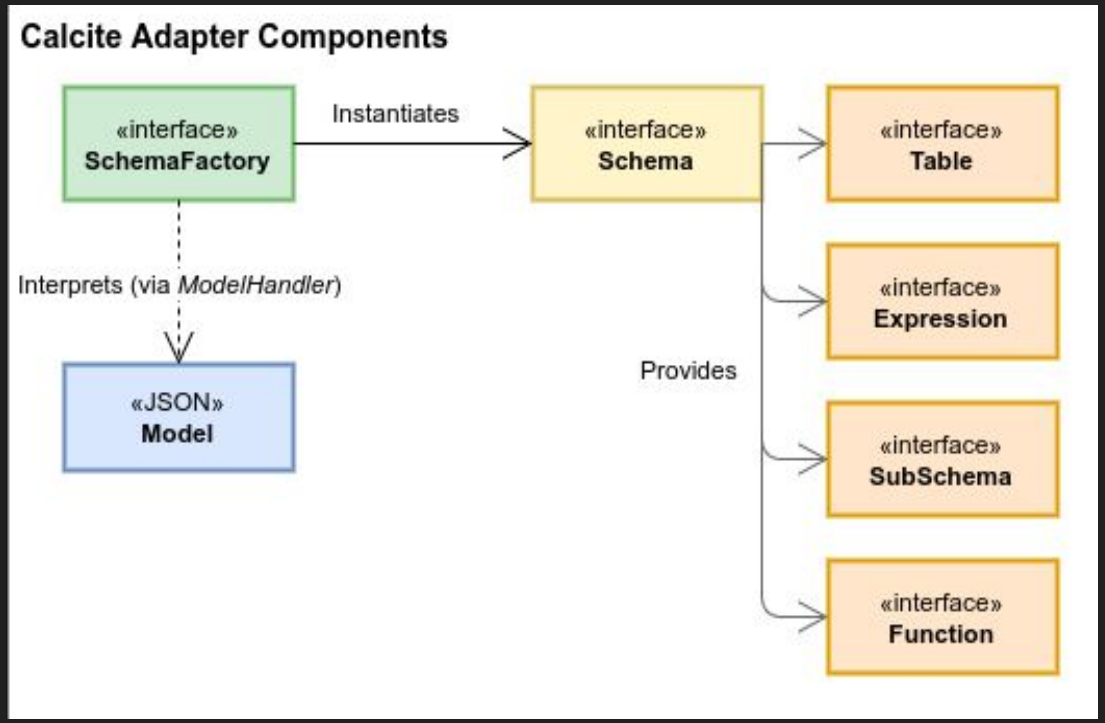
Core – Operator expressions (relational algebra) and planner (based on Volcano/Cascades)

External – Data storage, algorithms and catalog

Optional – SQL parser, JDBC & ODBC drivers

Extensible – Planner rewrite rules, statistics, cost model, algebra, UDFs

Adapter Design



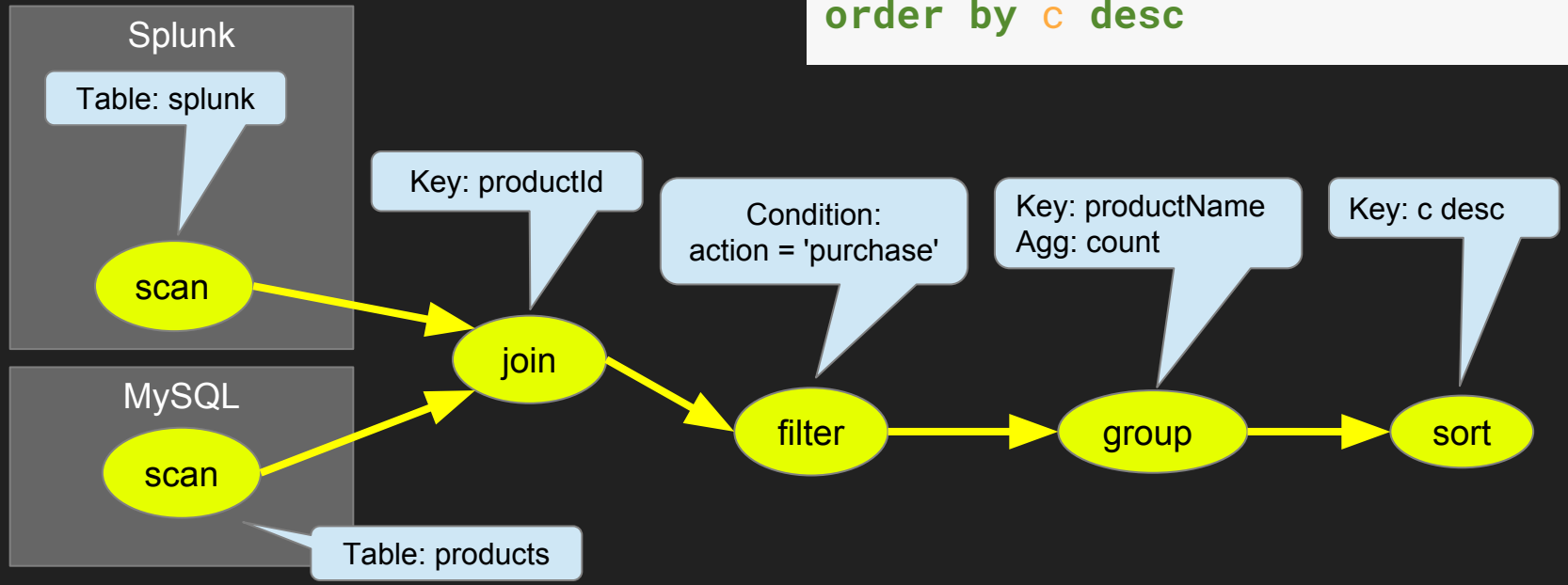
A pattern that defines how Calcite incorporates diverse data sources for general access.

Model – specification of the physical properties of the data source.

Schema – definition of the data (format and layouts) found in the model.

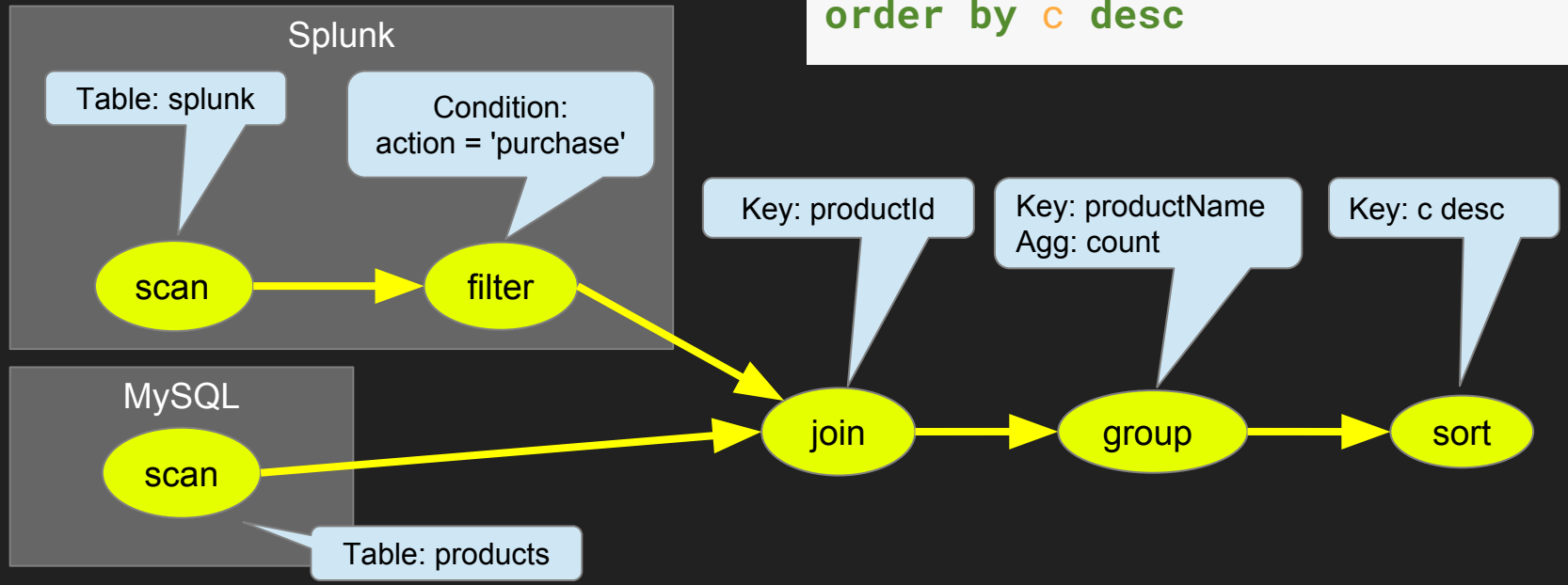
Represent query as relational algebra

```
select p.productName, count(*) as c
from splunk.splunk as s
    join mysql.products as p
    on s.productId = p.productId
where s.action = 'purchase'
group by p.productName
order by c desc
```

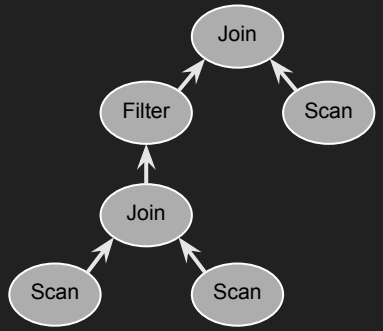


Optimize query by applying transformation rules

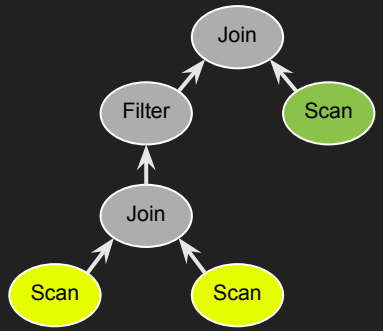
```
select p.productName, count(*) as c
from splunk.splunk as s
    join mysql.products as p
    on s.productId = p.productId
where s.action = 'purchase'
group by p.productName
order by c desc
```



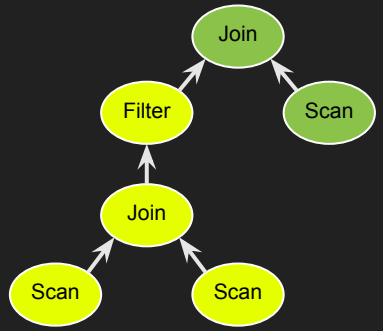
Conventions



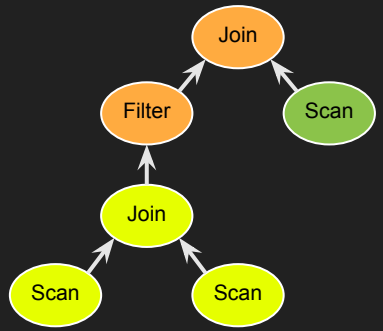
1. Plans start as logical nodes.



2. Assign each Scan its table's native convention.



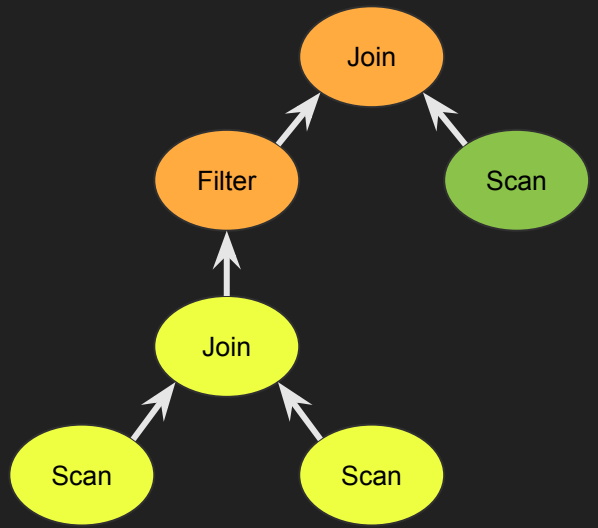
3. Fire rules to propagate conventions to other nodes.



4. The best plan may use an engine not tied to any native format.

To implement, generate a program that calls out to query1 and query2.

Conventions & adapters



Convention provides a uniform representation for hybrid queries

Like ordering and distribution, convention is a **physical property** of nodes

Adapter =
schema factory (lists tables)
+ convention
+ rules to convert nodes to convention

Streaming SQL

Stream ~= append-only table

Streaming queries return deltas

Stream-table duality: Orders is used as both stream and table

Our contributions:

- ➢ Popularize streaming SQL
- ➢ SQL parser / validator / rules
- ➢ Reference implementation & TCK

```
select stream *
from Orders as o
where units > (
  select avg(units)
  from Orders as h
  where h.productId = o.productId
  and h.rowtime >
    o.rowtime - interval '1' year)
```

“Show me real-time orders whose size is larger than the average for that product over the preceding year”

Uses and Adoption

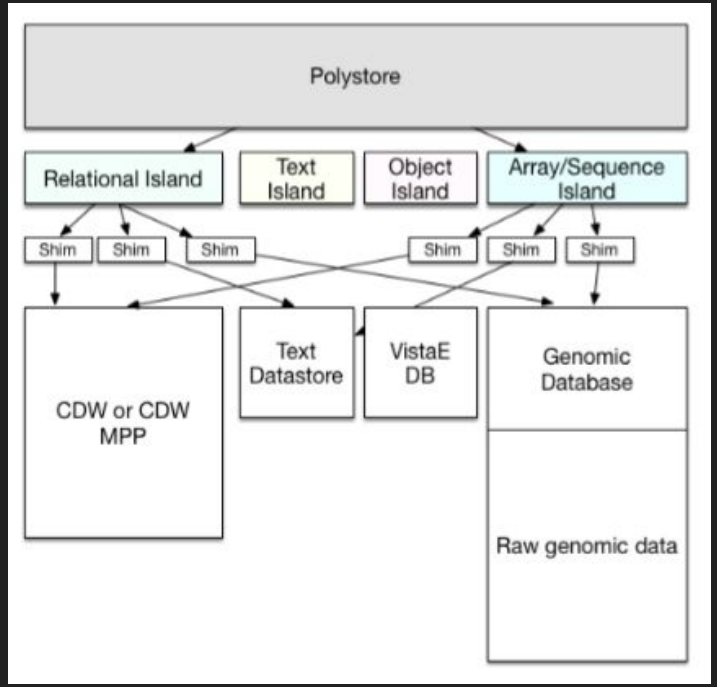
System	Query Language	JDBC Driver	SQL Parser and Validator	Relational Algebra	Execution Engine
Apache Drill	SQL + extensions	✓	✓	✓	Native
Apache Hive	SQL + extensions			✓	Apache Tez, Apache Spark
Apache Solr	SQL	✓	✓	✓	Native, Enumerable, Apache Lucene
Apache Phoenix	SQL	✓	✓	✓	Apache HBase
Apache Kylin	SQL	✓	✓		Enumerable, Apache HBase
Apache Apex	Streaming SQL	✓	✓	✓	Native
Apache Flink	Streaming SQL	✓	✓	✓	Native
Apache Samza	Streaming SQL	✓	✓	✓	Native
Apache Storm	Streaming SQL	✓	✓	✓	Native
MapD [32]	SQL		✓	✓	Native
Lingual [30]	SQL		✓	✓	Cascading
Qubole Quark [42]	SQL	✓	✓	✓	Apache Hive, Presto

Used by

Connects to

Uses in Research

- Polystore research – use as lightweight heterogeneous data processing platform
- Optimization and query profiling – general performance, and optimizer research
- Reasoning over Streams, Graphs – under consideration
- Open-source, production grade learning and research platform



Future Work and Roadmap

- Support its use as a **standalone engine** – DDL, materialized views, indexes and constraints.
- Improvements to the **design and extensibility of the planner** (modularity, pluggability)
- Incorporation of new **parametric approaches into the design of the optimizer.**
- Support for an extended set of SQL commands, functions, and utilities, including full compliance with **OpenGIS** (spatial).
- New adapters for non-relational data sources such as **array databases.**
- Improvements to **performance profiling and instrumentation.**

Thank you! Questions?

@ApacheCalcite

https://calcite.apache.org

https://arxiv.org/abs/1802.10233

Extra slides

Calcite framework

Relational algebra

RelNode (operator)
- TableScan
- Filter
- Project
- Union
- Aggregate
- …

RelDataType (type)
RexNode (expression)
RelTrait (physical property)
- RelConvention (calling-convention)
- RelCollation (sortedness)
- RelDistribution (partitioning)

RelBuilder

SQL parser

SqlNode
SqlParser
SqlValidator

Metadata

Schema
Table
Function
- TableFunction
- TableMacro

Lattice

JDBC driver

Transformation rules

RelOptRule
- FilterMergeRule
- AggregateUnionTransposeRule
- 100+ more

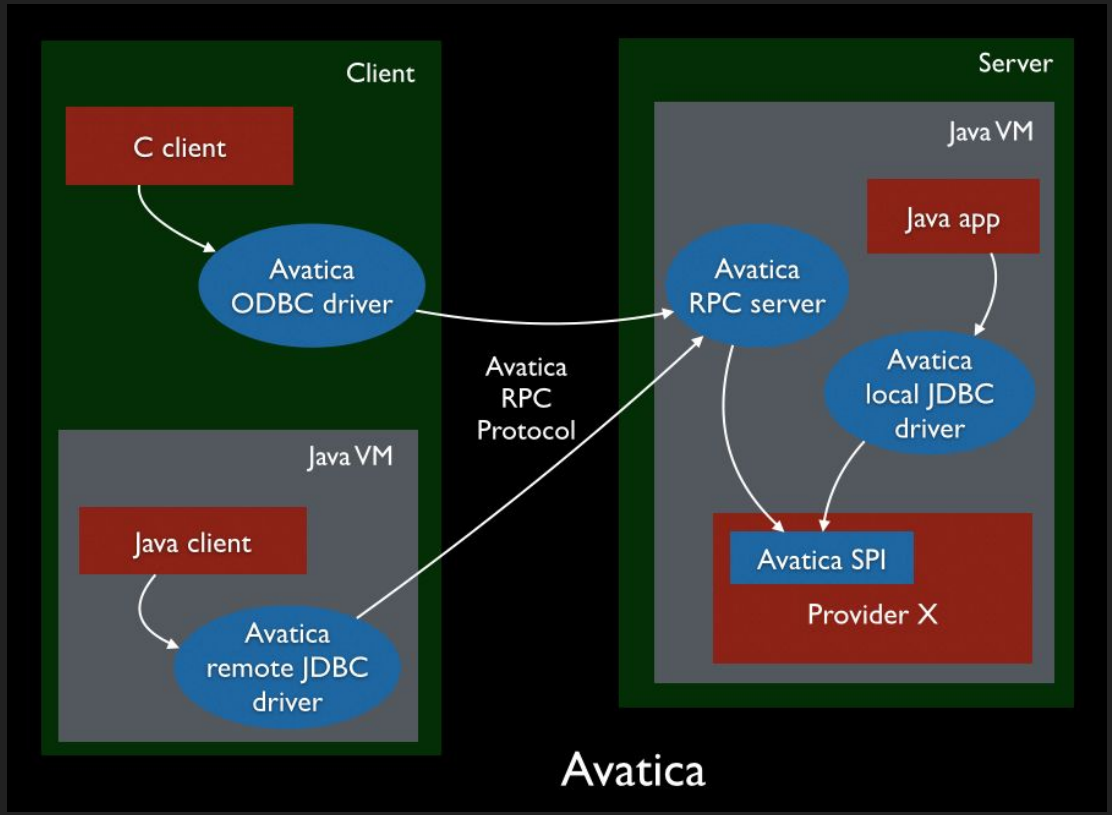
Global transformations
- Unification (materialized view)
- Column trimming
- De-correlation

Cost, statistics

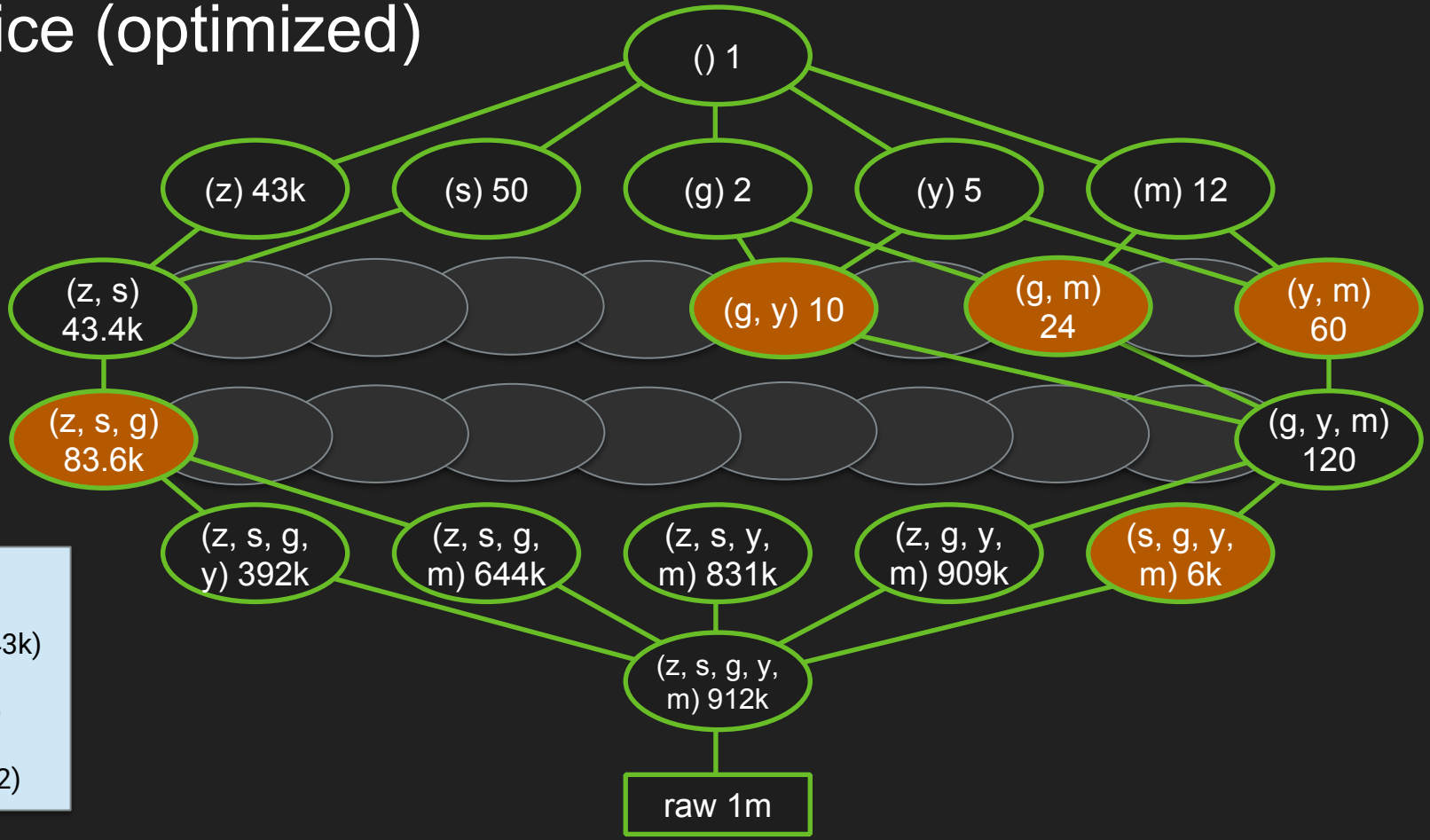
RelOptCost
RelOptCostFactory
RelMetadataProvider
- RelMdColumnUniquensss
- RelMdDistinctRowCount
- RelMdSelectivity

Avatica

- Database connectivity stack
- Self-contained sub-project of Calcite
- Fast, open, stable
- Protobuf or JSON over HTTP
- Powers Phoenix Query Server



Lattice (optimized)



Key

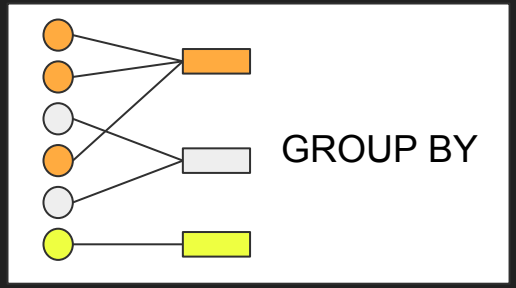
z zipcode (43k)
s state (50)
g gender (2)
y year (5)
m month (12)

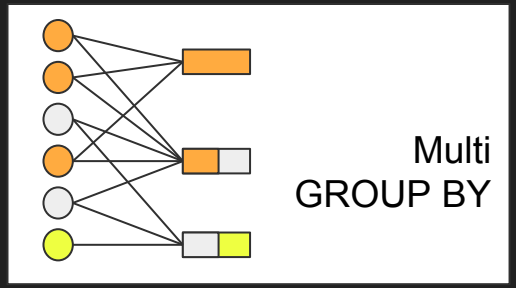
Aggregation and windows on streams

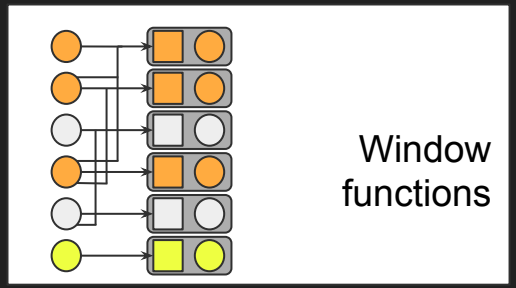
GROUP BY aggregates multiple rows into sub-totals

- In regular GROUP BY each row contributes to exactly one sub-total
- In multi-GROUP BY (e.g. HOP, GROUPING SETS) a row can contribute to more than one sub-total

Window functions (OVER) leave the number of rows unchanged, but compute extra expressions for each row (based on neighboring rows)

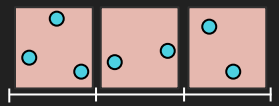






Tumbling, hopping & session windows in SQL

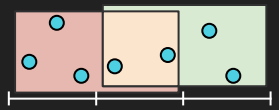
Tumbling window



```
select stream … from Orders
group by floor(rowtime to hour)
```

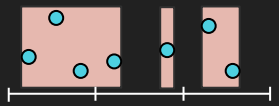
```
select stream … from Orders
group by tumble(rowtime, interval '1' hour)
```

Hopping window



```
select stream … from Orders
group by hop(rowtime, interval '1' hour,
    interval '2' hour)
```

Session window



```
select stream … from Orders
group by session(rowtime, interval '1' hour)
```

Controlling when data is emitted

Early emission is the defining characteristic of a streaming query.

The **emit** clause is a SQL extension inspired by Apache Beam's "trigger" notion. (Still experimental… and evolving.)

A relational (non-streaming) query is just a query with the most conservative possible emission strategy.

```
select stream productId,
  count(*) as c
from Orders
group by productId,
  floor(rowtime to hour)
emit at watermark,
  early interval '2' minute,
  late limit 1;
```

```
select *
from Orders
emit when complete;
```