
# Building query compilers using Apache Calcite

Stamatis Zampetakis • February 21, 2022

# About me

Stamatis Zampetakis @szampetak
Staff Software Engineer @ Cloudera, Hive query optimizer team
PMC member of Apache Calcite; Apache Hive committer
PhD in Data Management, INRIA & Paris-Sud University



# Outline

1. About Cloudera
2. The Apache Software Foundation (ASF)
3. Motivation
4. Calcite overview
5. Coding module I: Main components
6. Hybrid planning
7. Coding module II: Simple operators/rules
8. Coding module III: Multiple datasources
9. Coding module IV: Advanced operators/rules

CLOUDERA

THE ENTERPRISE DATA CLOUD COMPANY

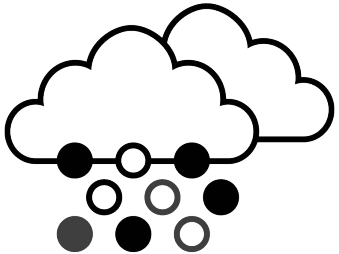
We believe that data can make what is impossible today, possible tomorrow

We empower people to transform complex data into clear and actionable insights

We deliver cloud-native services to manage and secure the data lifecycle in any cloud or data center

CLOUDERA

THE ENTERPRISE DATA CLOUD COMPANY

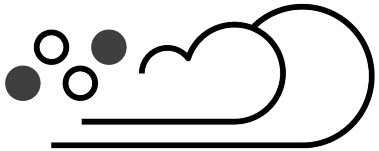


Any Cloud


Data Lifecycle

CLOUDERA SDX

Secure & Governed



Open

# HOW DO CUSTOMERS USE CLOUDERA?

Every business use case is a data lifecycle use case

| | BANKING | TECHNOLOGY | TELCO | LIFE SCIENCES | PUBLIC SECTOR |
|---|---|---|---|---|---|
| USE CASES | • **Fraud detection**<br>• Anti-money laundering<br>• Spend analytics | • Customer analytics<br>• Threat detection<br>• **Predictive support** | • Churn analysis<br>• Customer care<br>• **Network optimization** | • **Patient care (IoT)**<br>• Genomics research<br>• Regulatory compliance | • Cyber<br>• Fraud detection<br>• **Regulatory Compliance** |
| KEY CUSTOMERS | • Barclays<br>• Citi<br>• **Santander UK** | • Cisco<br>• Intel<br>• **Reef Technology** | • Globe Telecom<br>• Deutsche Telecom<br>• Robi Axiata | • **GlaxoSmithKline**<br>• Clearsense<br>• Cerner | • **AFP**<br>• **Services Australia**<br>• **ATO**<br>• **Department of Health** |

# Apache Software Foundation

What is it:

- Non-profit public charity organization
- Incorporated in the United States of America
- Formed in 1999

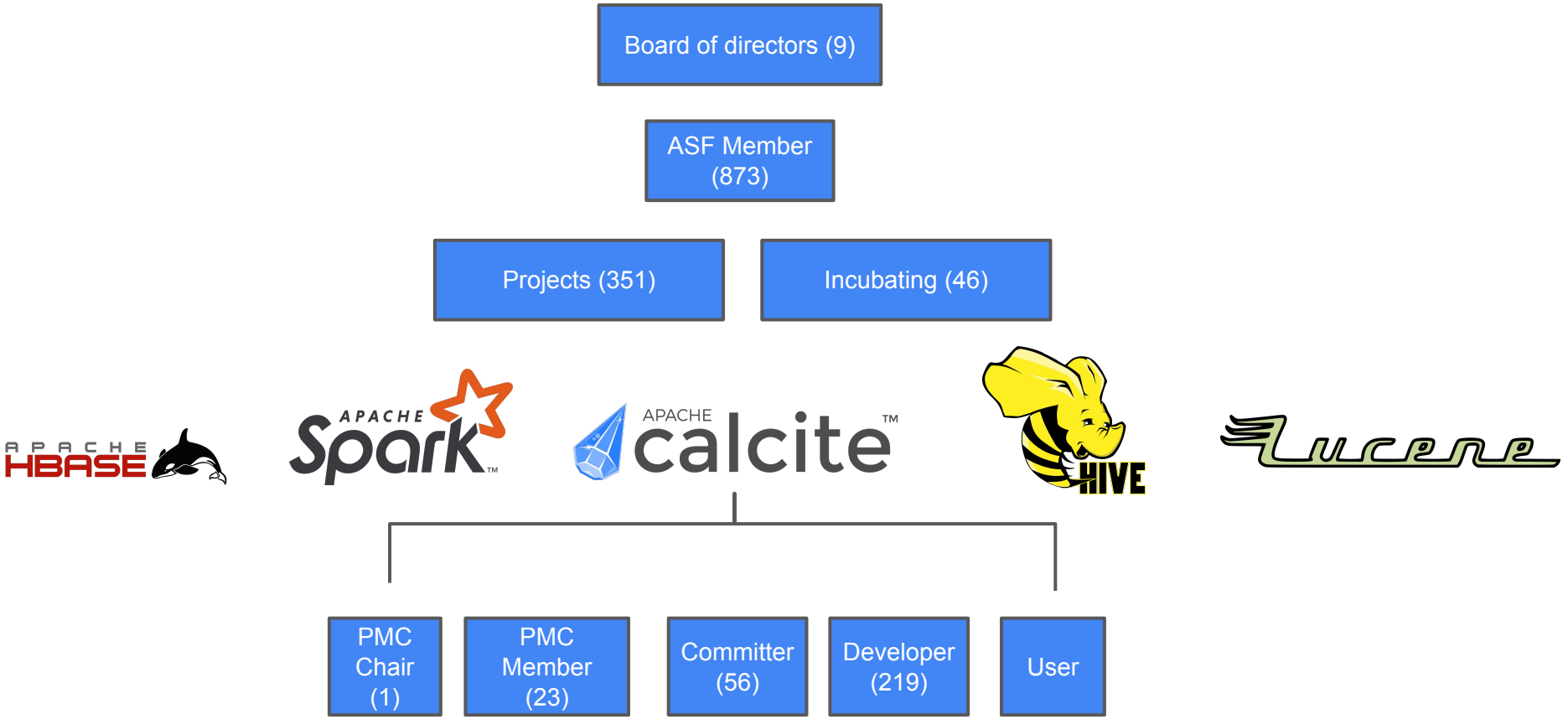
Why it was formed:

- Provide a foundation for open, collaborative software development projects
- Create an independent legal entity to which companies and individuals can donate resources
- Protect individual volunteers from legal suits
- Protect the 'Apache' brand from being abused by other organizations

# Meritocracy

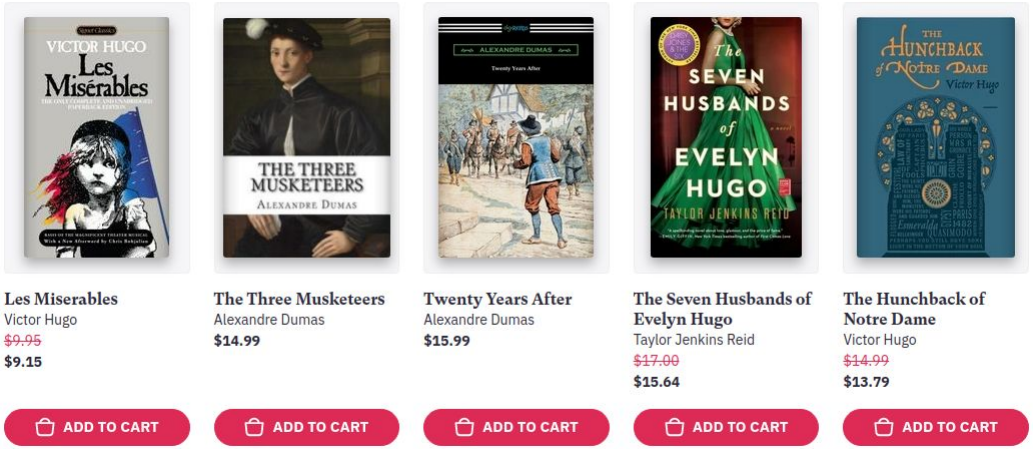
- Started by a diverse group of people with common interests
- Support and maintain the HTTPD web server written by the NCSA
- As the software involved more people were attracted and started to help out
- People who contributed a lot “earned” the merit to be part of the community
- Process scaled very well since newcomers were seen as volunteers and not as contenders
- Commitment to a task and collaboration (especially under disagreements) were (and still are) very important for the Apache group

# The Foundation Structure



# Motivation

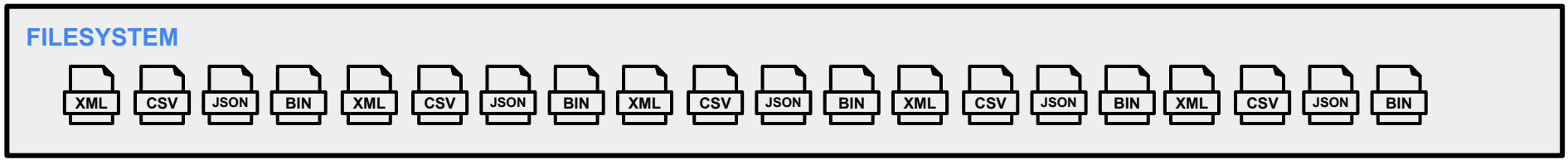
# Motivation: Data views




1. Retrieve books and authors
2. Display image, title, price of the book along with firstname & lastname of the author
3. Sort the books based on their id (price or something else)
4. Show results in groups of five

# What, where, how data are stored?

# What, where, how data are stored?

# What, where, how data are stored?

| AUTHOR | |
|---|---|
| id | int |
| fname | string |
| lname | string |
| birth | date |

author

0..1

| BOOK | |
|---|---|
| id | int |
| title | string |
| price | decimal |
| year | int |
















**FILESYSTEM**

# What, where, how data are stored?


**360+ DBMS**

**FILESYSTEM**

# Apache Lucene

- ★ Open-source search engine written in Java
- ★ Apache project since 2005
- ★ Powerful indexing & search features
- ★ Spell checking, hit highlighting
- ★ Advanced analysis/tokenization capabilities
- ★ ACID transactions
- ★ Ultra compact memory/disk format

# How to query the data?

1. Retrieve books and authors
2. Display image, title, price of the book along with firstname & lastname of the author
3. Sort the books based on their id (price or something else)
4. Show results in groups of five

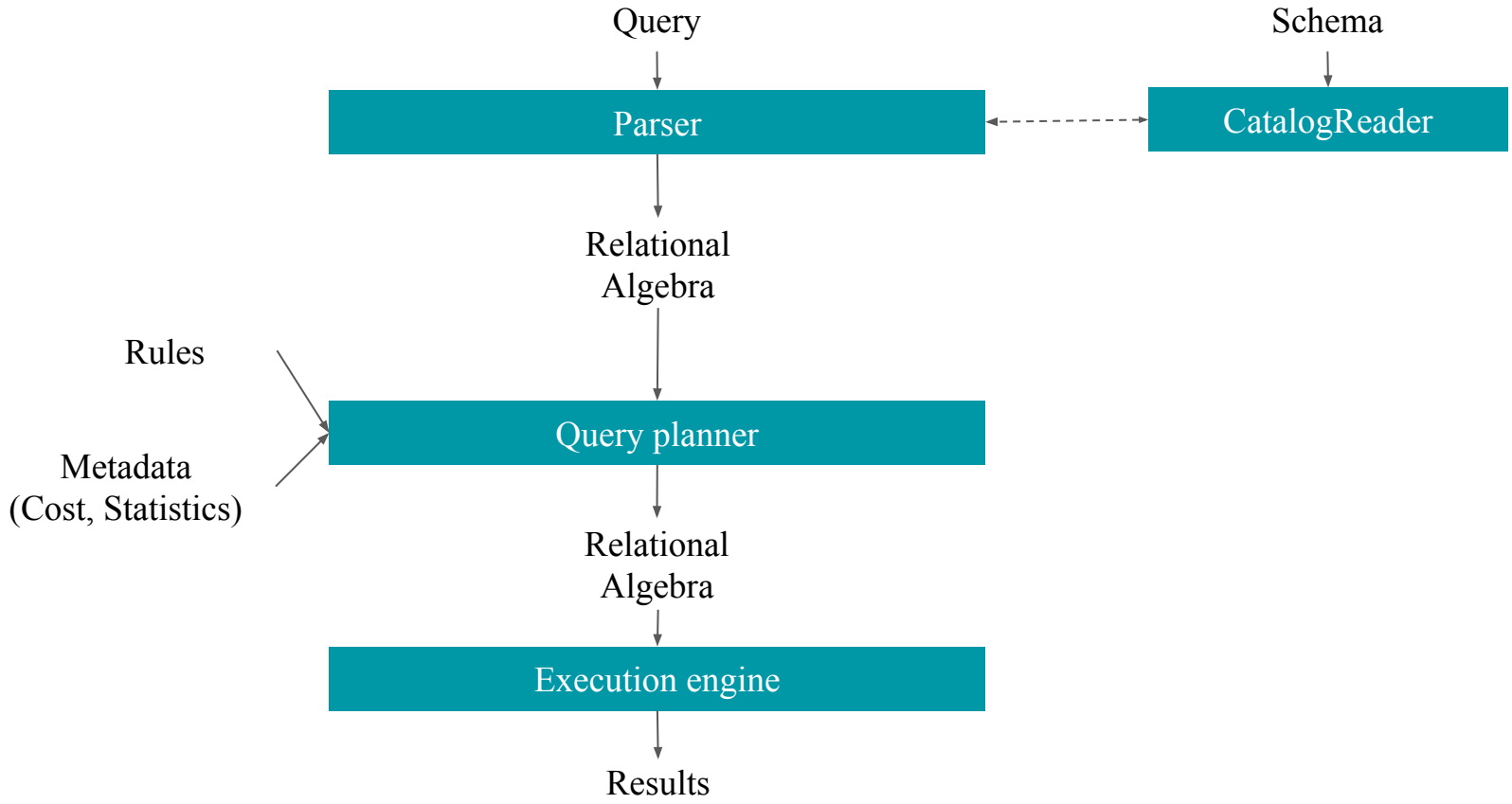
```
SELECT b.id, b.title, b.year, a.fname, a.lname
FROM Book b
LEFT OUTER JOIN Author a ON b.author=a.id
ORDER BY b.id
LIMIT 5
```

# Calcite overview

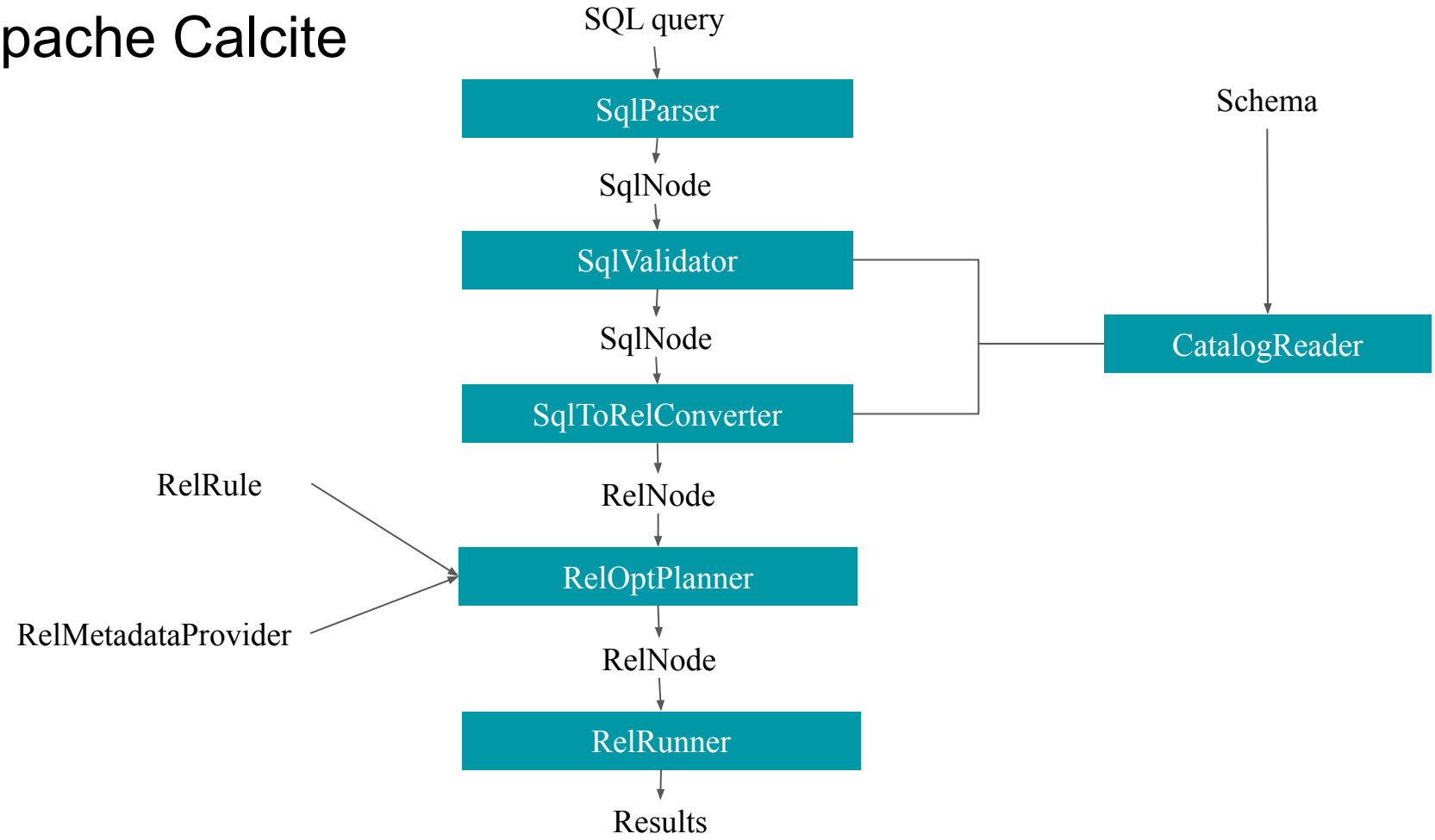
# Apache Calcite

- ★ Open source query processing framework written in Java
- ★ Apache project since October 2015
- ★ Industry-standard SQL parser and validator
- ★ Customizable optimizers: Volcano, Heuristic
- ★ Hundreds of pluggable transformation rules
- ★ Logical and physical algebraic operators
- ★ Adapters for SQL & NoSQL engines
- ★ Used by many data management systems: Hive, Spark, Flink, Solr, Phoenix, Drill

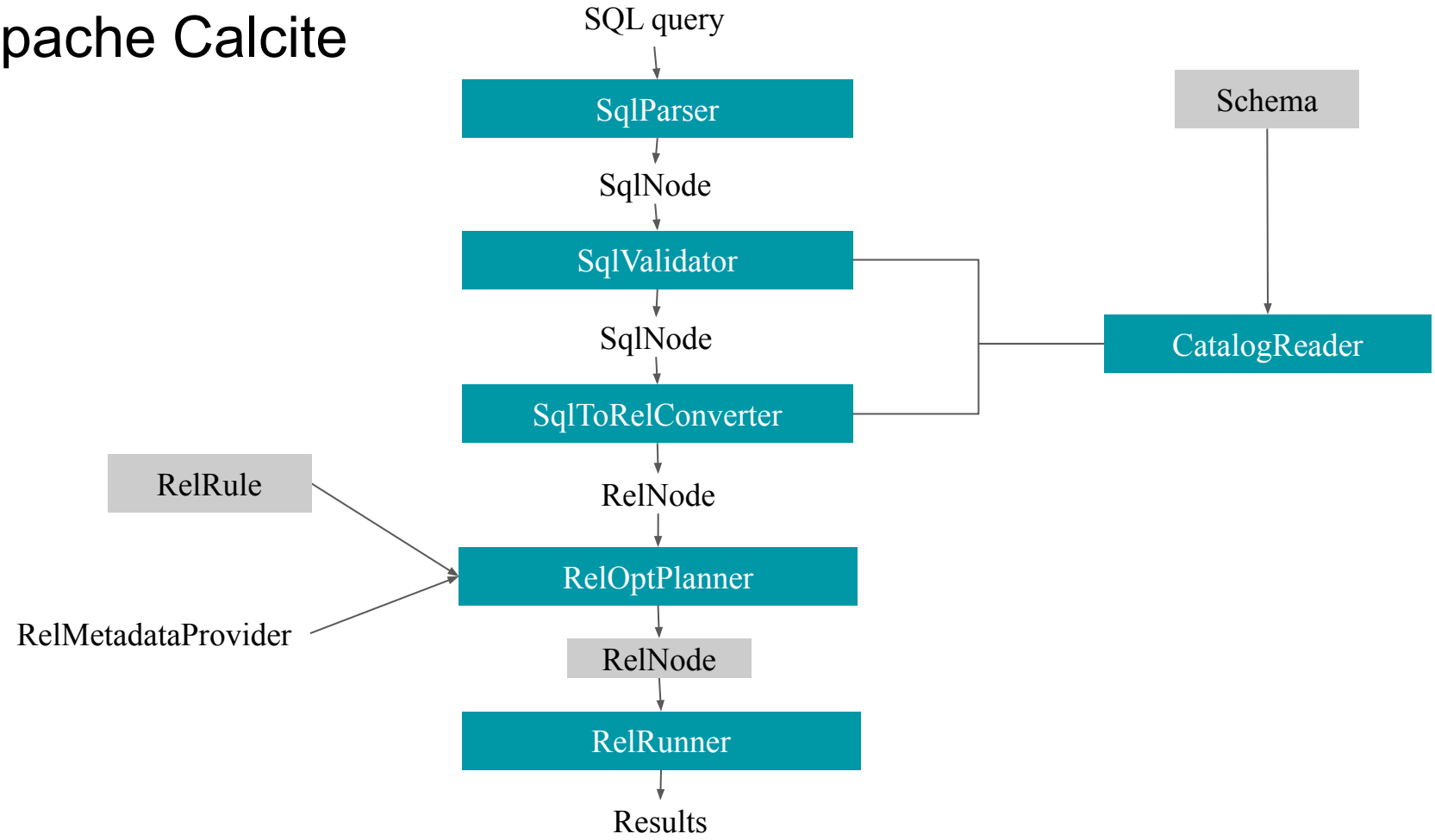
# Query processor architecture



# Apache Calcite



# Apache Calcite



# Coding module I: Main components

# Setup Environment

**Requirements:** Git, JDK version ≥ 1.8

```
git --version
java -version
```

**Steps**

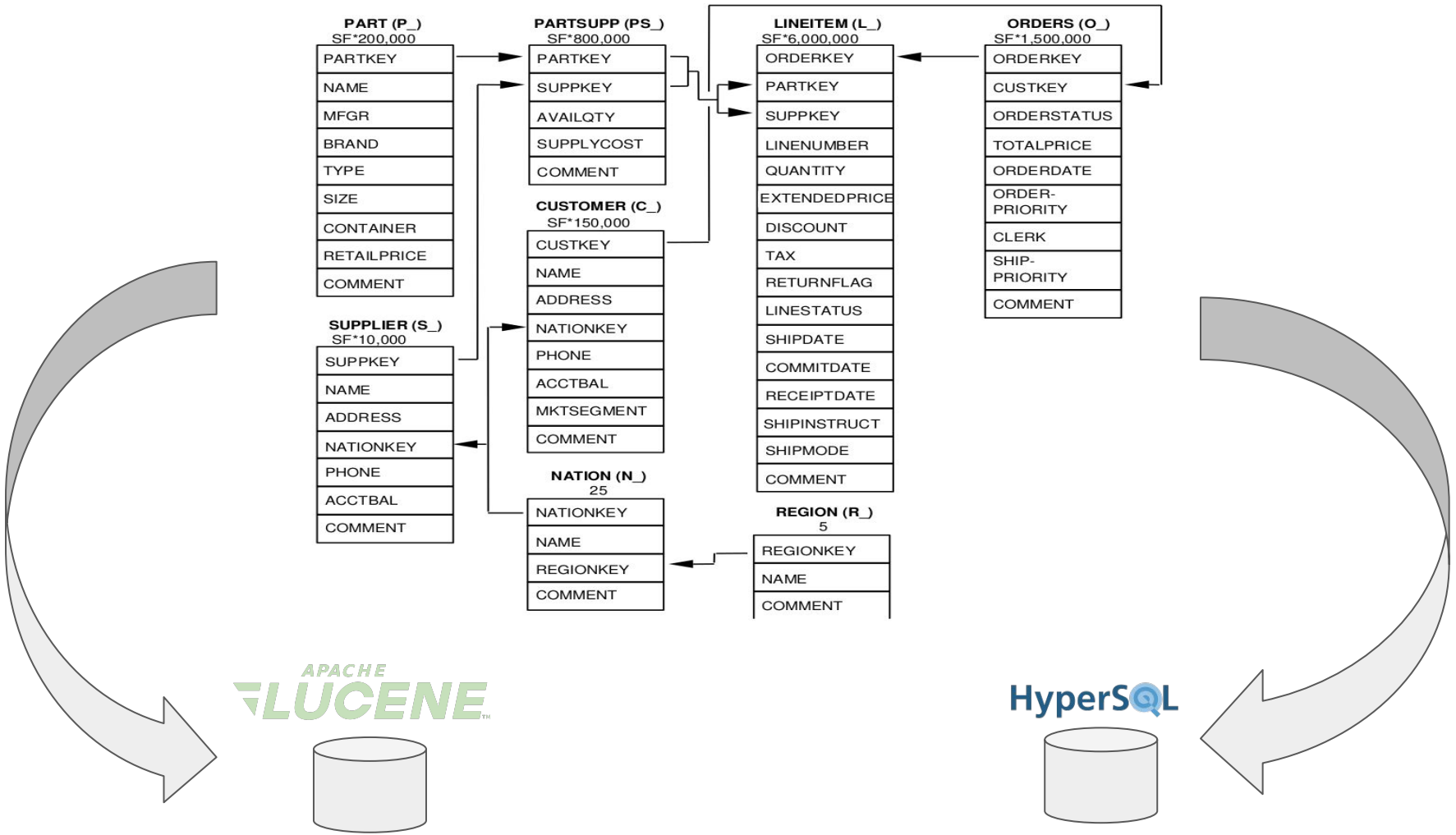
1. Clone GitHub repository

```
git clone
https://github.com/zabetak/cy-calcite-tutorial.git
```

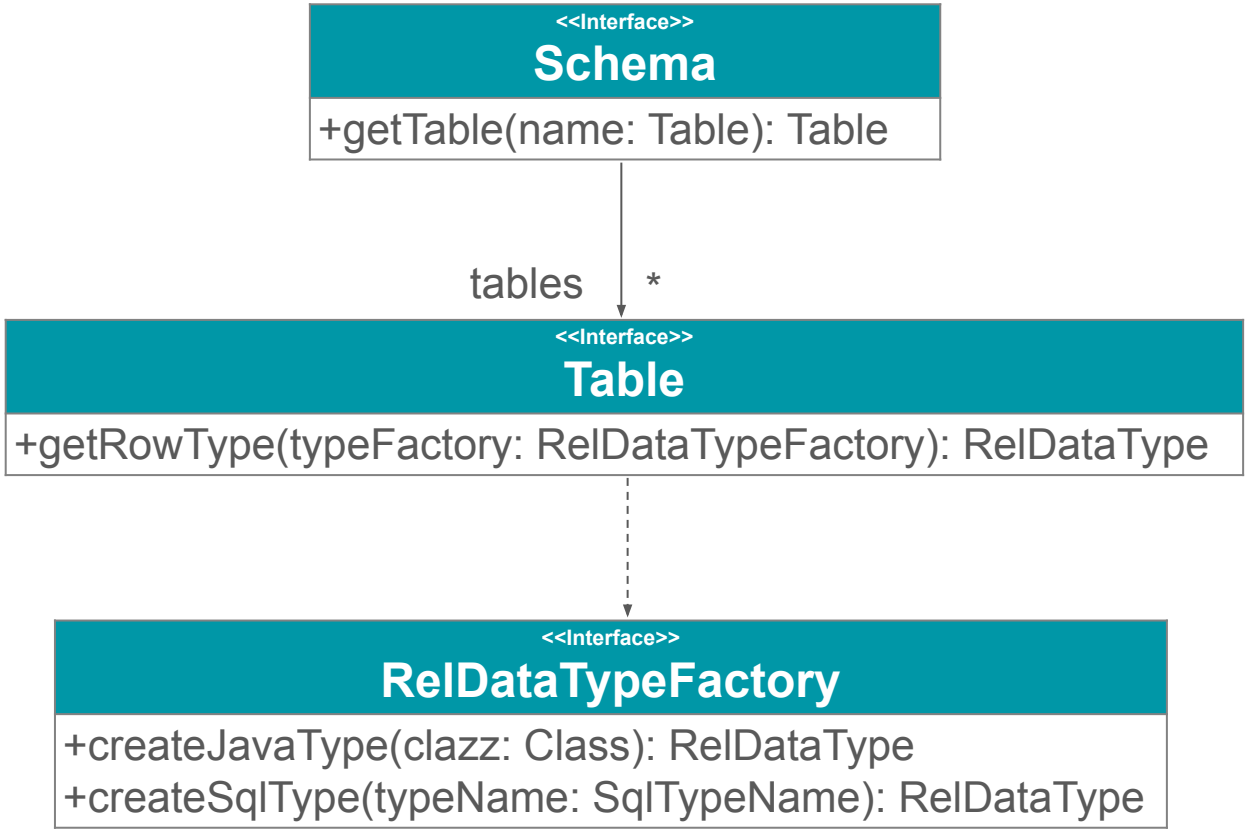
2. Compile the project and download dependencies

```
cd cy-calcite-tutorial
./mvnw package -DskipTests
```

3. Load to IDE (preferred IntelliJ)
    a. Click Open
    b. Navigate to `cy-calcite-tutorial` directory
    c. Select `pom.xml` file
    d. Choose “Open as Project”


PART (P_)
SF*200,000
PARTKEY
NAME
MFGR
BRAND
TYPE
SIZE
CONTAINER
RETAILPRICE
COMMENT
PARTSUPP (PS_)
SF*800,000
PARTKEY
SUPPKEY
AVAILQTY
SUPPLYCOST
COMMENT
LINEITEM (L_)
SF*6,000,000
ORDERKEY
PARTKEY
SUPPKEY
LINENUMBER
QUANTITY
EXTENDEDPRICE
DISCOUNT
TAX
RETURNFLAG
LINESTATUS
SHIPDATE
COMMITDATE
RECEIPTDATE
SHIPINSTRUCT
SHIPMODE
COMMENT
ORDERS (O_)
SF*1,500,000
ORDERKEY
CUSTKEY
ORDERSTATUS
TOTALPRICE
ORDERDATE
ORDER-PRIORITY
CLERK
SHIP-PRIORITY
COMMENT
CUSTOMER (C_)
SF*150,000
CUSTKEY
NAME
ADDRESS
NATIONKEY
PHONE
ACCTBAL
MKTSEGMENT
COMMENT
SUPPLIER (S_)
SF*10,000
SUPPKEY
NAME
ADDRESS
NATIONKEY
PHONE
ACCTBAL
COMMENT
NATION (N_)
25
NATIONKEY
NAME
REGIONKEY
COMMENT
REGION (R_)
5
REGIONKEY
NAME
COMMENT
APACHE
LUCENE
HyperSQL

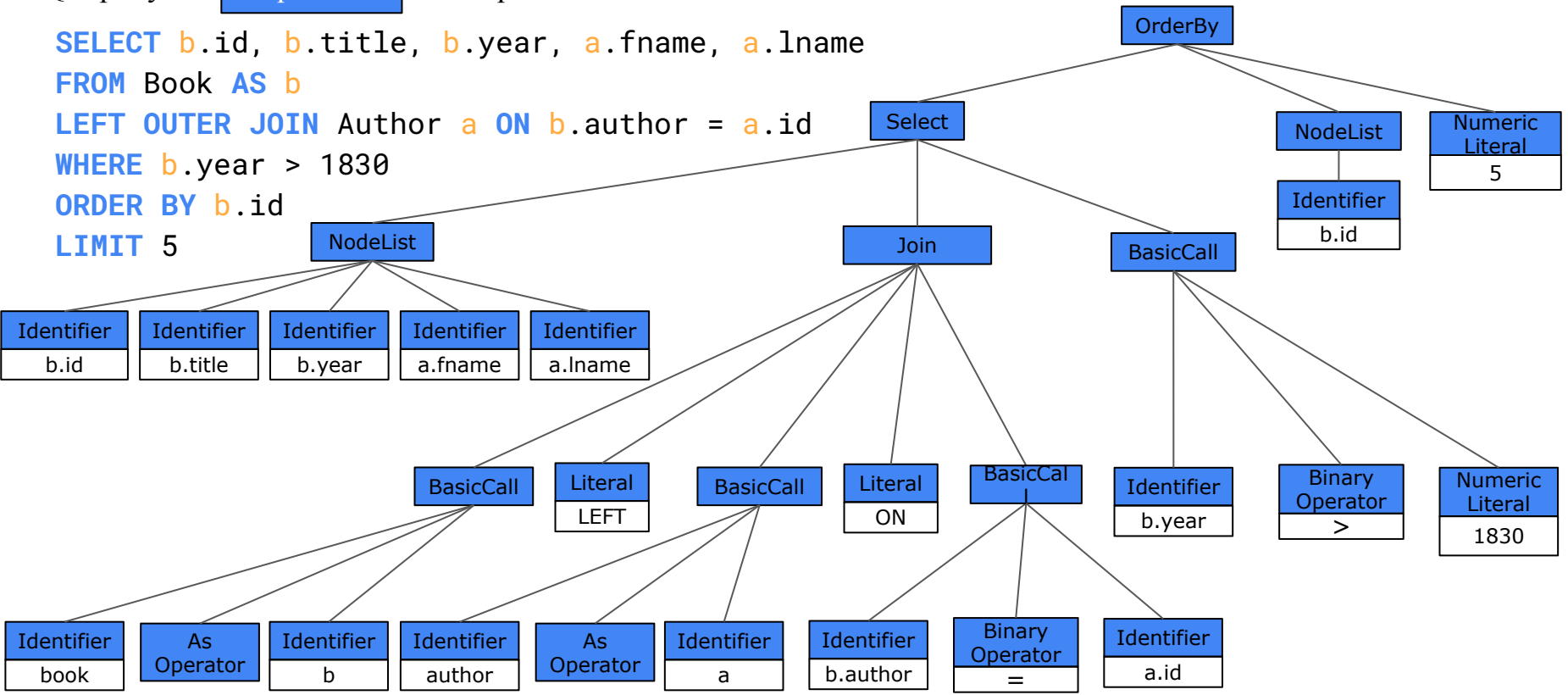
# Setup schema & type factory



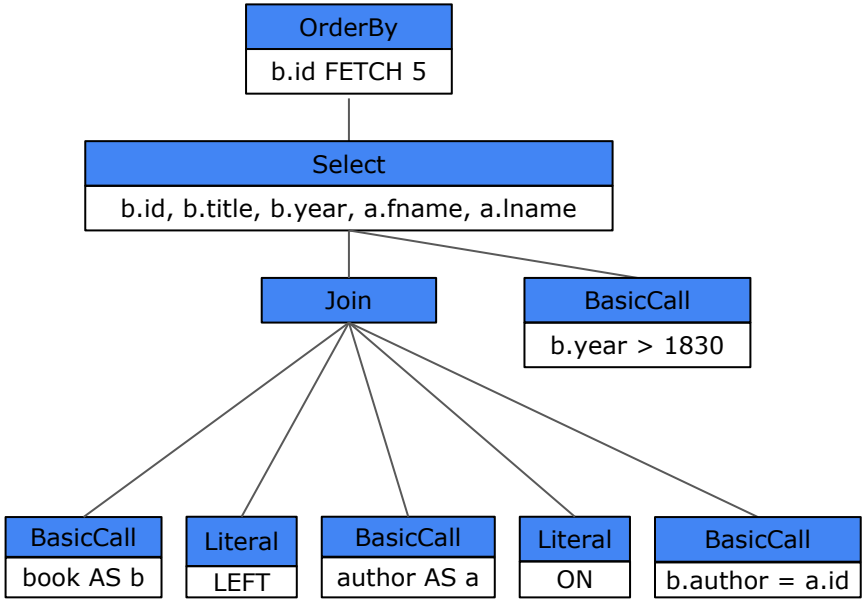
# Query to Abstract Syntax Tree (AST)

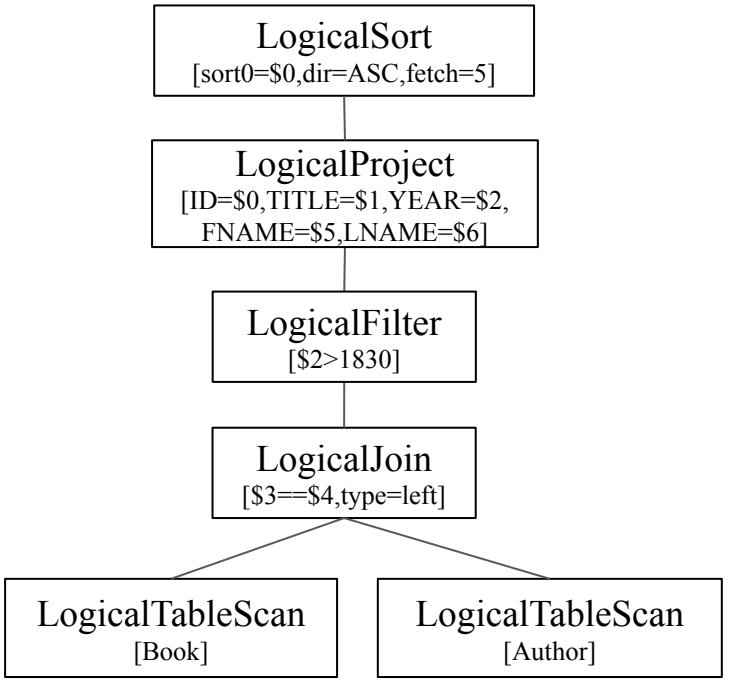
SQL query → SqlParser → SqlNode

```
SELECT b.id, b.title, b.year, a.fname, a.lname
FROM Book AS b
LEFT OUTER JOIN Author a ON b.author = a.id
WHERE b.year > 1830
ORDER BY b.id
LIMIT 5
```

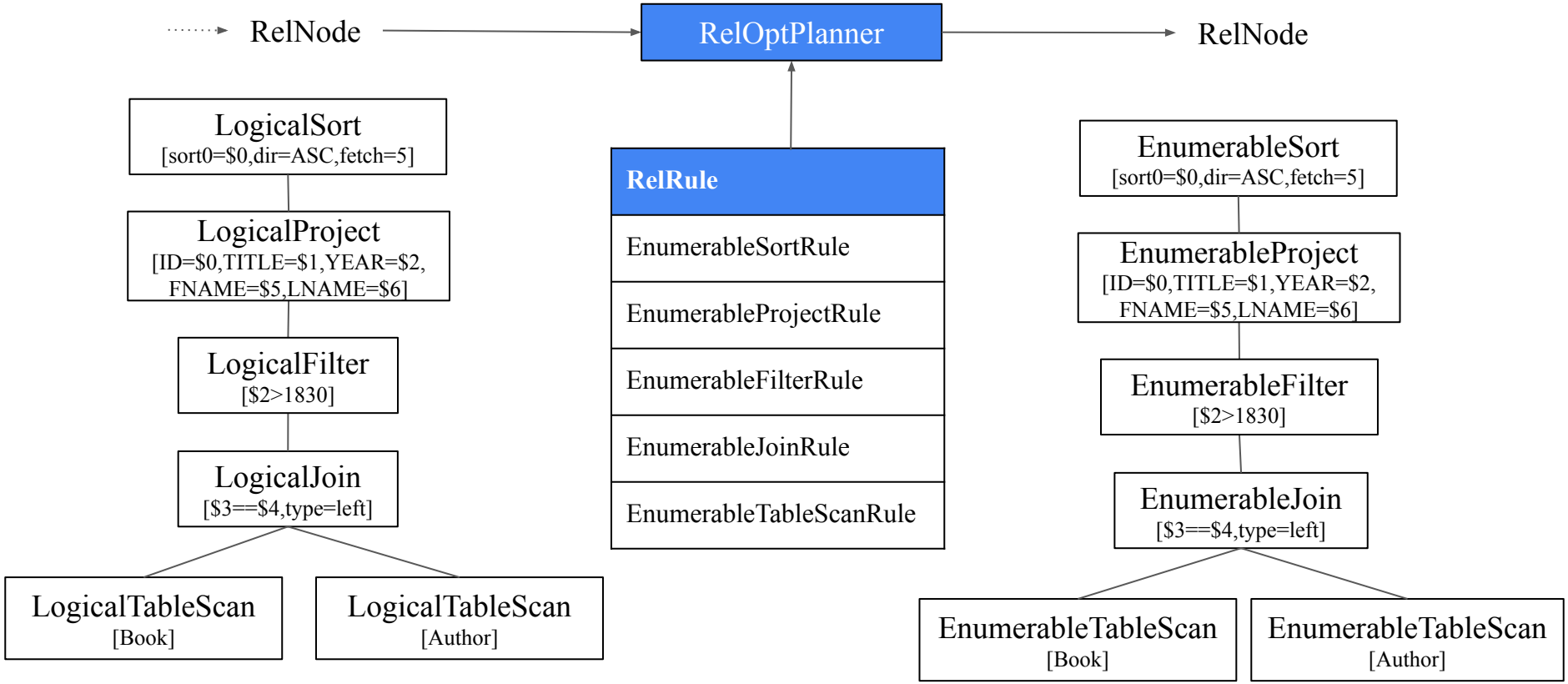


# AST to logical plan

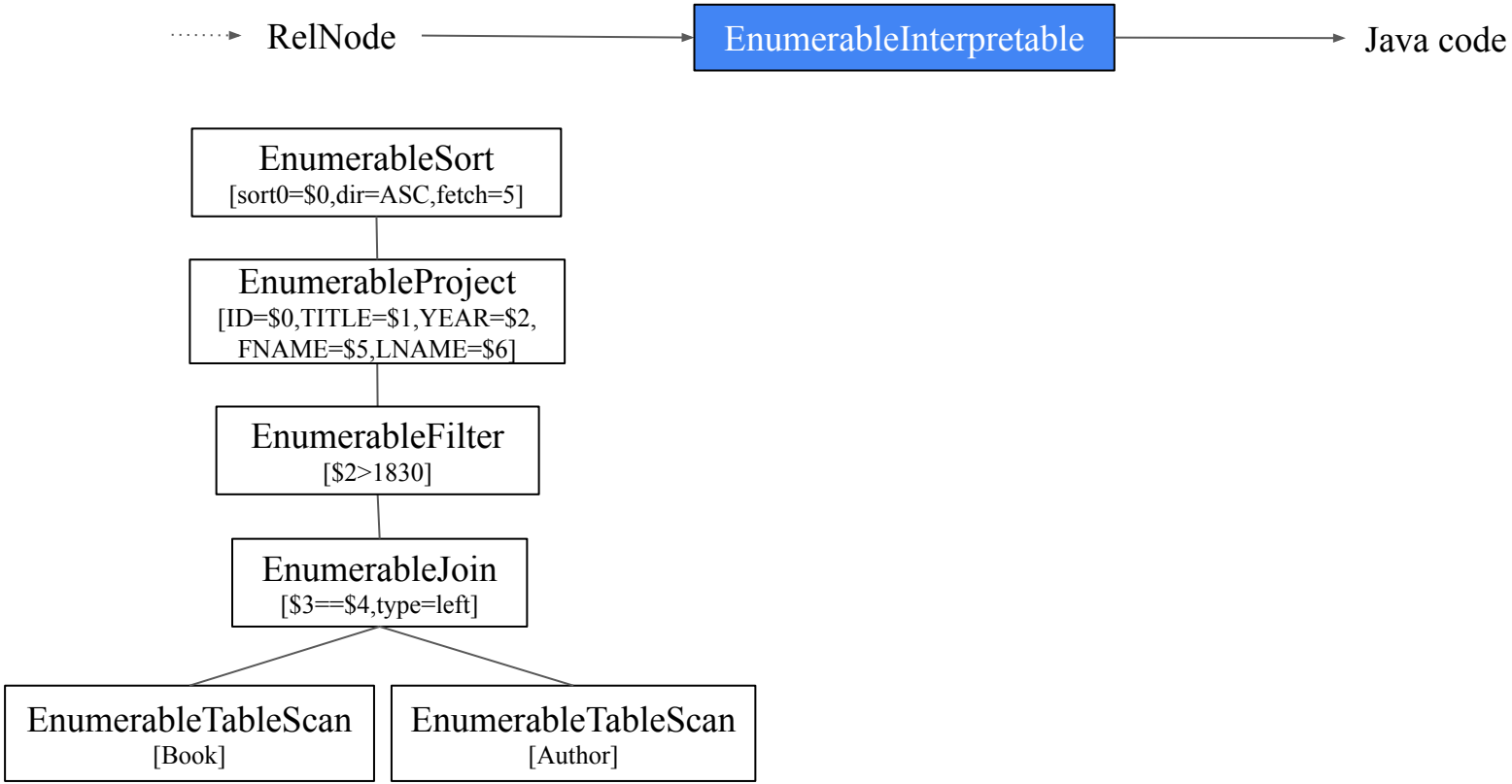



# Logical to physical plan



# Physical to Executable plan



# Exercise I: Execute more SQL queries

Include GROUP BY and other types of clauses:

```
SELECT o.o_custkey, COUNT(*)
FROM orders AS o
GROUP BY o.o_custkey
```

# Exercise I: Execute more SQL queries

Include GROUP BY and other types of clauses:

```
SELECT o.o_custkey, COUNT(*)
FROM orders AS o
GROUP BY o.o_custkey
```

- Missing rule to convert `LogicalAggregate` to `EnumerableAggregate`
- Add `EnumerableRules.ENUMERABLE_AGGREGATE_RULE` to the planner

# Exercise II: Improve performance by applying more optimization rules

Push filter below the join:

```
SELECT c.c_name, o.o_orderkey, o.o_orderdate
FROM customer AS c
INNER JOIN orders AS o ON c.c_custkey = o.o_custkey
WHERE c.c_custkey < 3
ORDER BY c.c_name, o.o_orderkey
```

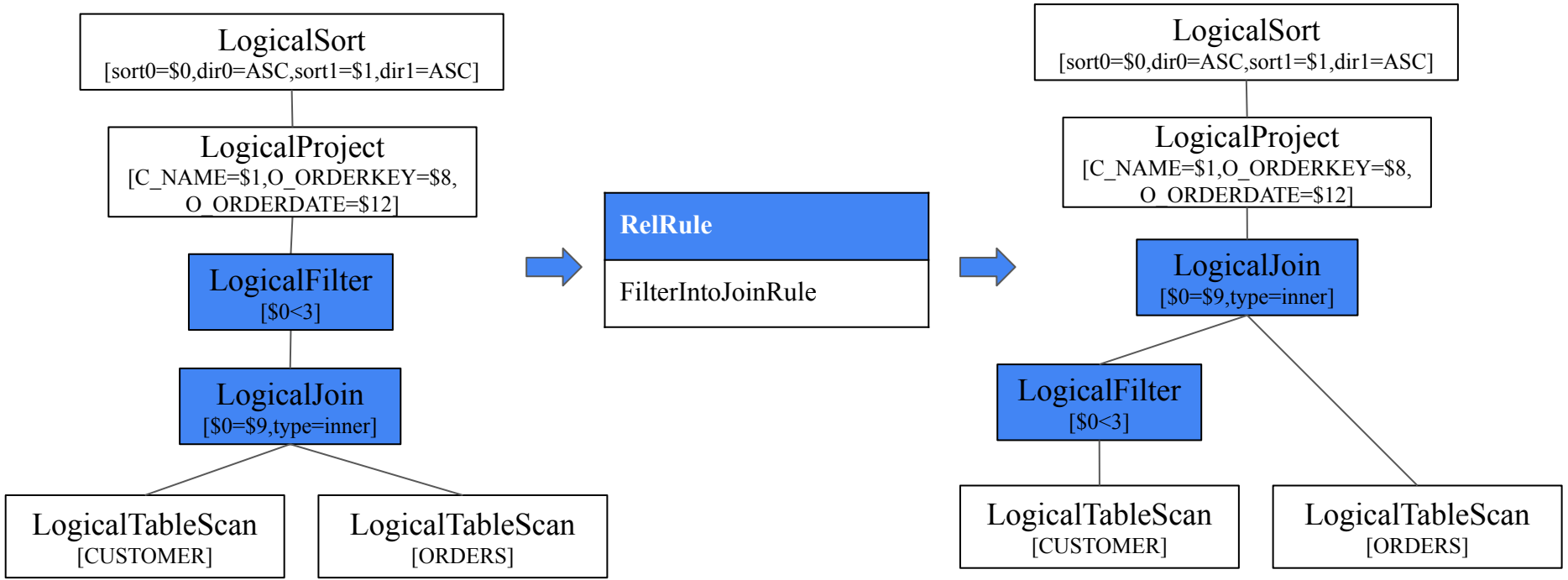
# Exercise II: Improve performance by applying more optimization rules

Push filter below the join:

```
SELECT c.c_name, o.o_orderkey, o.o_orderdate
FROM customer AS c
INNER JOIN orders AS o ON c.c_custkey = o.o_custkey
WHERE c.c_custkey < 3
ORDER BY c.c_name, o.o_orderkey
```

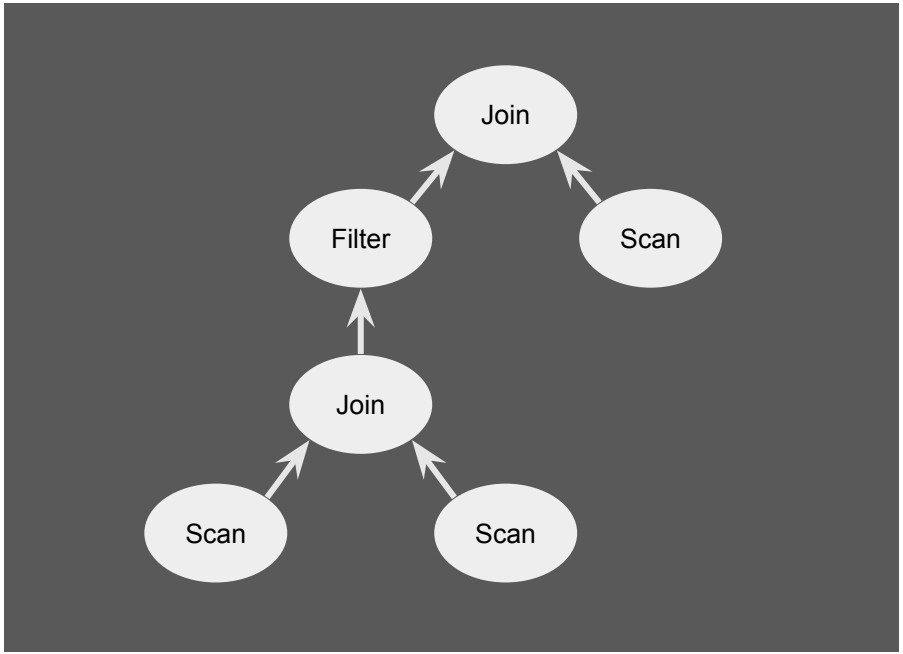
1. Add rule `CoreRules.FILTER_INTO_JOIN` to the planner
2. Compare plans before and after (or logical and physical)
3. Check cost estimates by using `SqlExplainLevel.ALL_ATTRIBUTES`

# Exercise II: Improve performance by applying more optimization rules



# Hybrid planning

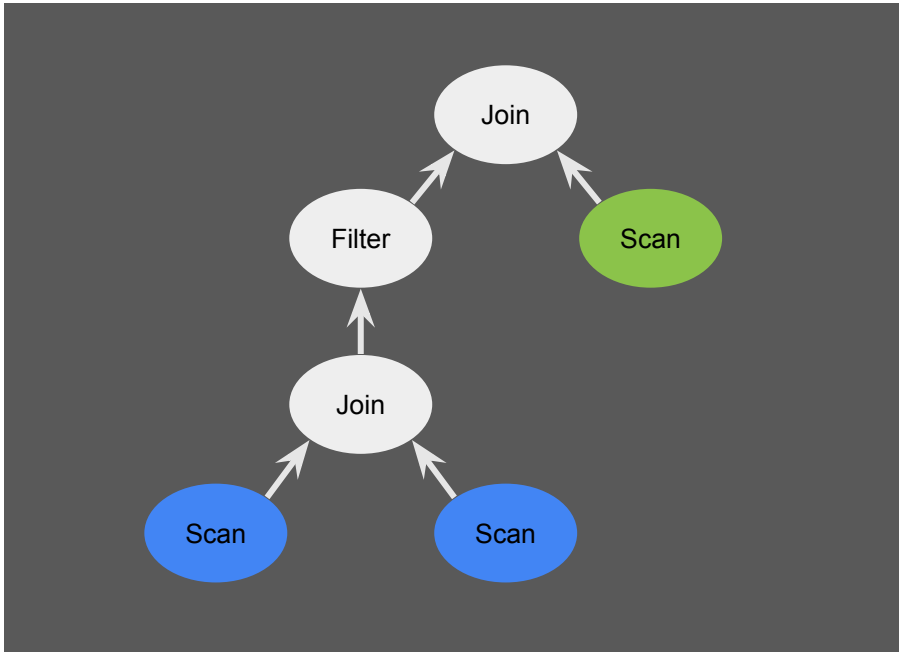
# Calling convention




Initially all nodes belong to “logical” calling convention.

Logical calling convention cannot be implemented, so has infinite cost

# Calling convention

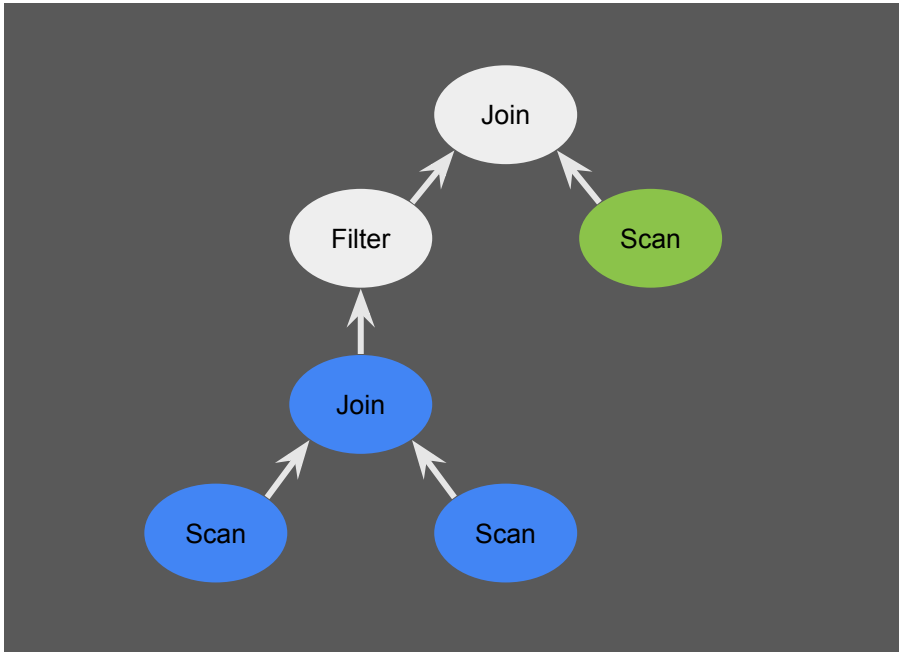



Tables can’t be moved so there is only one choice of calling convention for each table.

Examples:

- Enumerable
- Lucene
- JDBC
- Druid
- Drill
- HBase

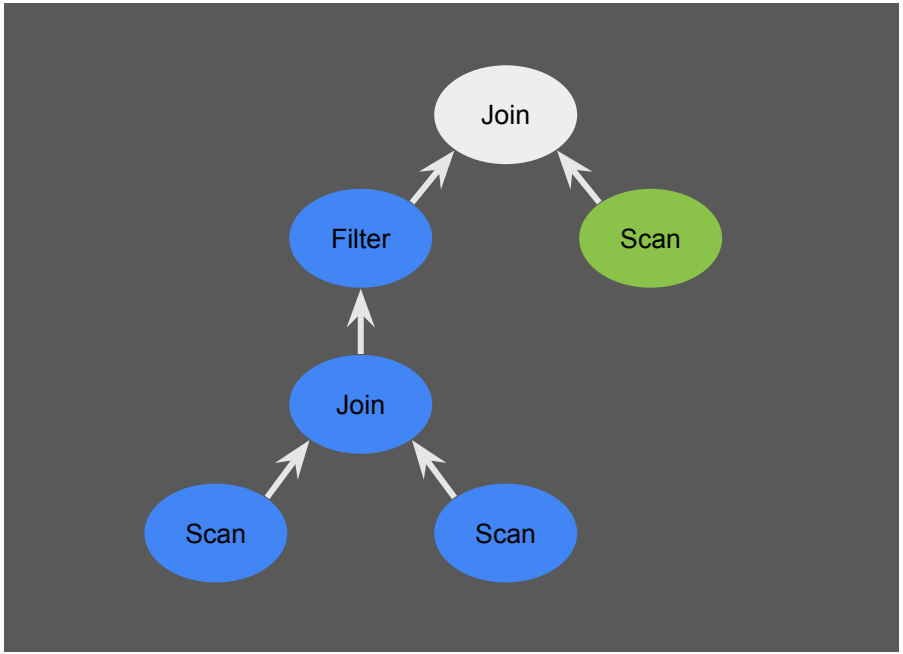
# Calling convention




Rules fire to convert nodes to particular calling conventions.

The calling convention propagates through the tree.

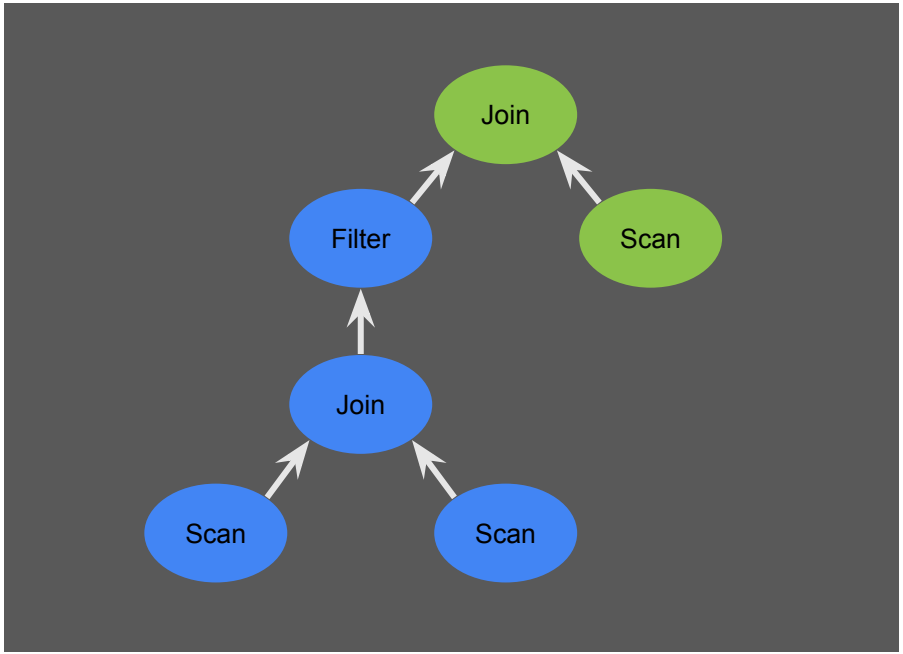
# Calling convention




Rules fire to convert nodes to particular calling conventions.

The calling convention propagates through the tree.

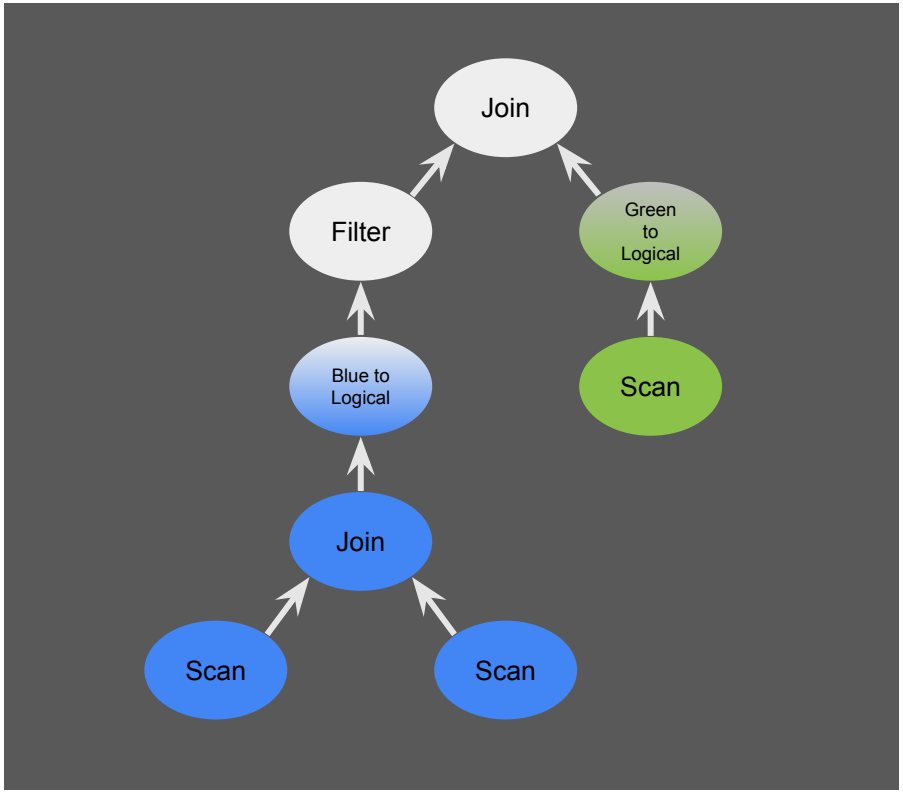
# Calling convention




Rules fire to convert nodes to particular calling conventions.

The calling convention propagates through the tree.

# Converters

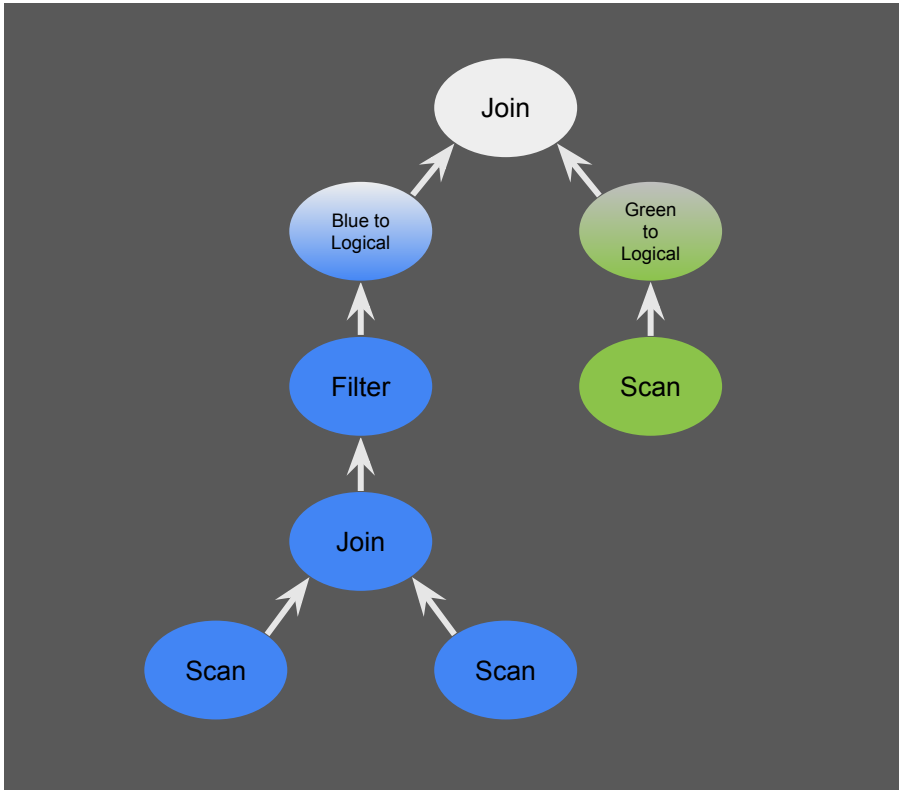



To keep things honest, we need to insert a **converter** at each point where the convention changes.

```
BlueFilterRule:

LogicalFilter(BlueToLogical(Blue b))
  →
BlueToLogical(BlueFilter(b))
```

# Converters

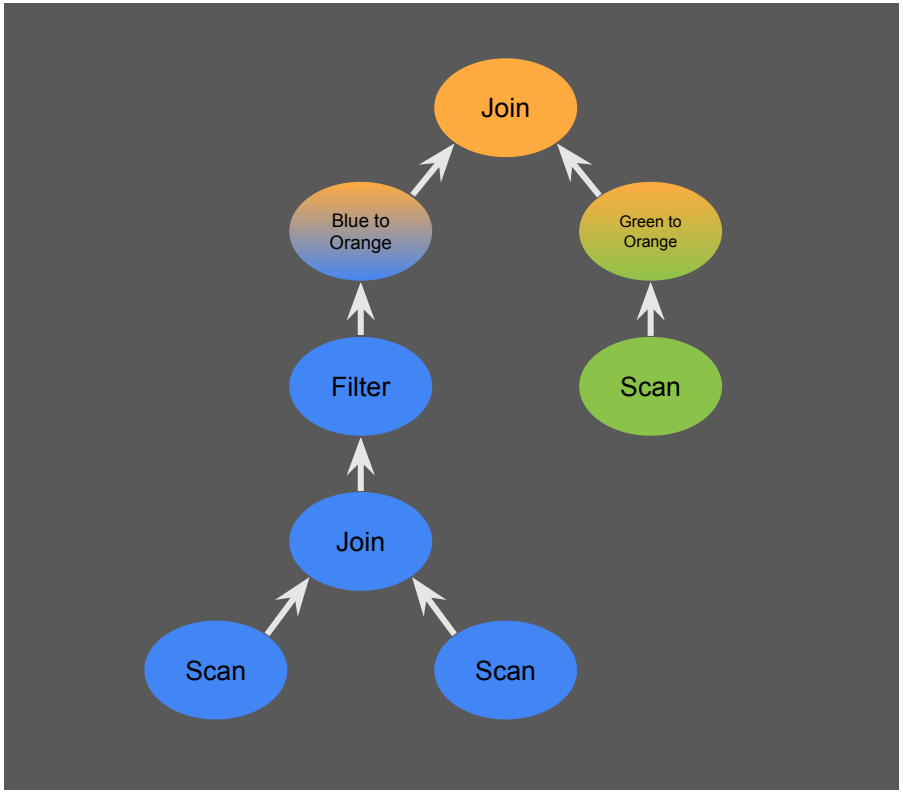



To keep things honest, we need to insert a **converter** at each point where the convention changes.

```
BlueFilterRule:

LogicalFilter(BlueToLogical(Blue b))
  →
BlueToLogical(BlueFilter(b))
```

# Generating programs to implement hybrid plans




Hybrid plans are glued together using an **engine** - a convention that does not have a storage format. (Example engines: Drill, Spark, Presto.)

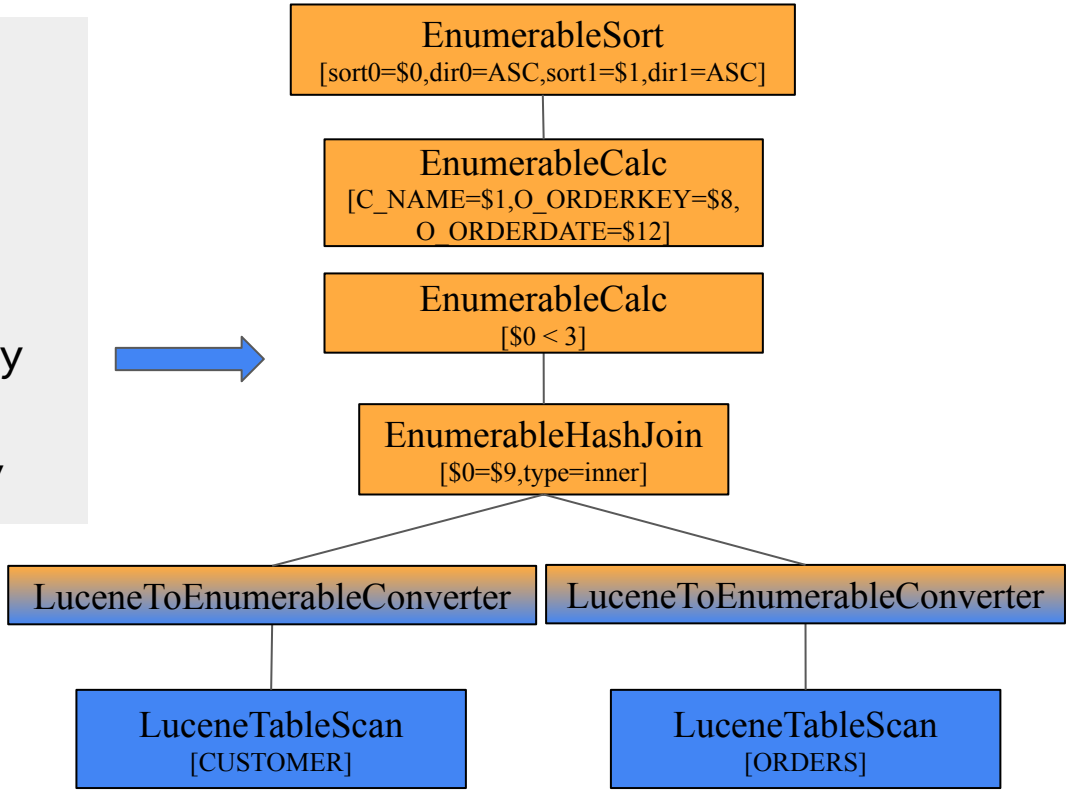
To implement, we generate a program that calls out to query1 and query2.

The "Blue-to-Orange" converter is typically a function in the Orange language that embeds a Blue query. Similarly "Green-to-Orange".

# Coding module II: Custom operators/rules

# What we want to achieve?

```
SELECT c.c_name,
 o.o_orderkey,
 o.o_orderdate
FROM lucene.customer AS c
INNER JOIN lucene.orders AS o
    ON c.c_custkey = o.o_custkey
WHERE c.c_custkey < 3
ORDER BY c.c_name, o.o_orderkey
```



# What do we need?

Calling conventions:

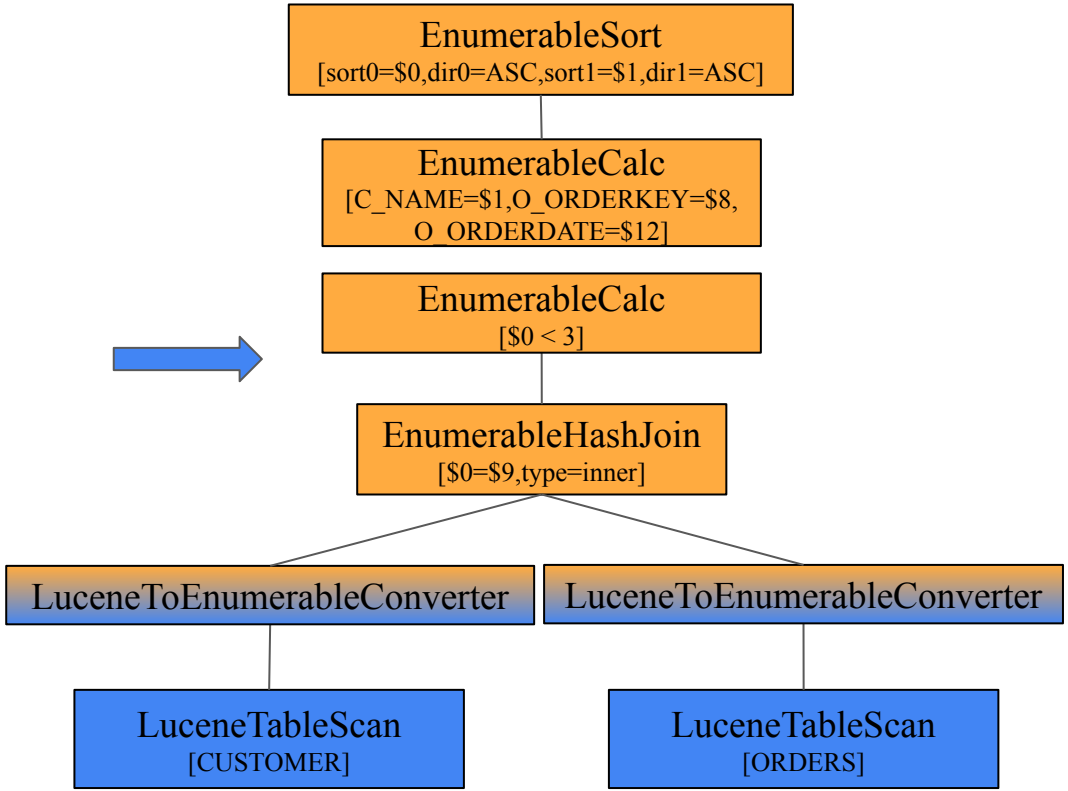
1. **Enumerable**
2. **Lucene**

Custom operators:

1. `LuceneTableScan`
2. `LuceneToEnumerableConverter`

Custom conversion rules:

1. `LogicalTableScan` → `LuceneTableScan`
2. `LuceneANY` → `LuceneToEnumerableConverter`



# What do we need?

Calling conventions:

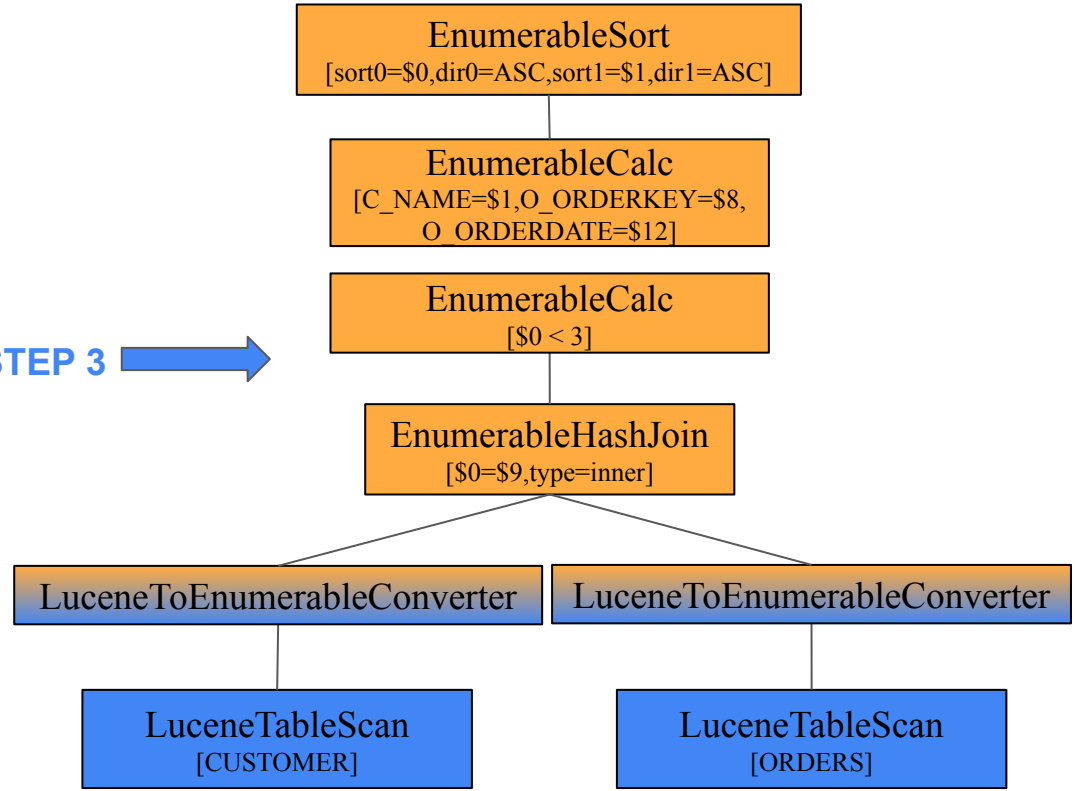
1. **Enumerable**
2. **Lucene**

Custom operators:

1. `LuceneTableScan` **STEP 1**
2. `LuceneToEnumerableConverter` **STEP 3**

Custom conversion rules:

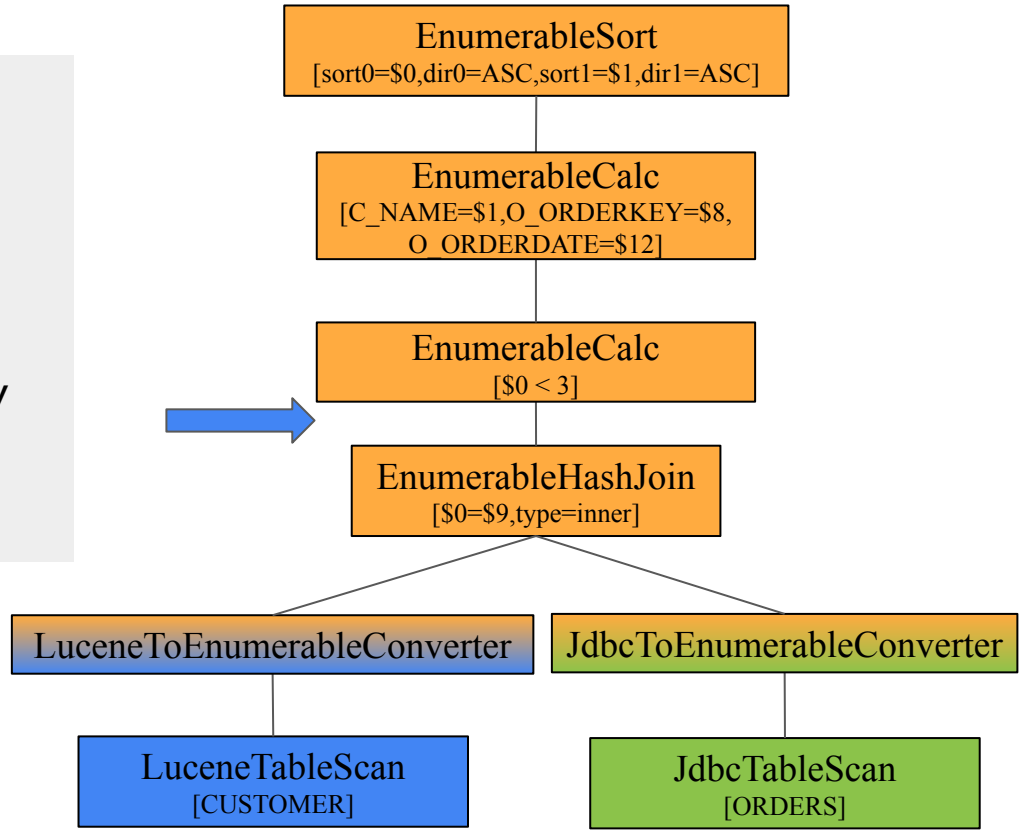
1. `LogicalTableScan` → `LuceneTableScan` **STEP 2**
2. `LuceneANY` → `LuceneToEnumerableConverter` **STEP 4**



# Coding module III: Multiple data sources

# What we want to achieve?

```
SELECT c.c_name,
 o.o_orderkey,
 o.o_orderdate
FROM lucene.customer AS c
INNER JOIN hyper.orders AS o
    ON c.c_custkey = o.o_custkey
WHERE c.c_custkey < 3
ORDER BY c.c_name, o.o_orderkey
```



# What do we need?

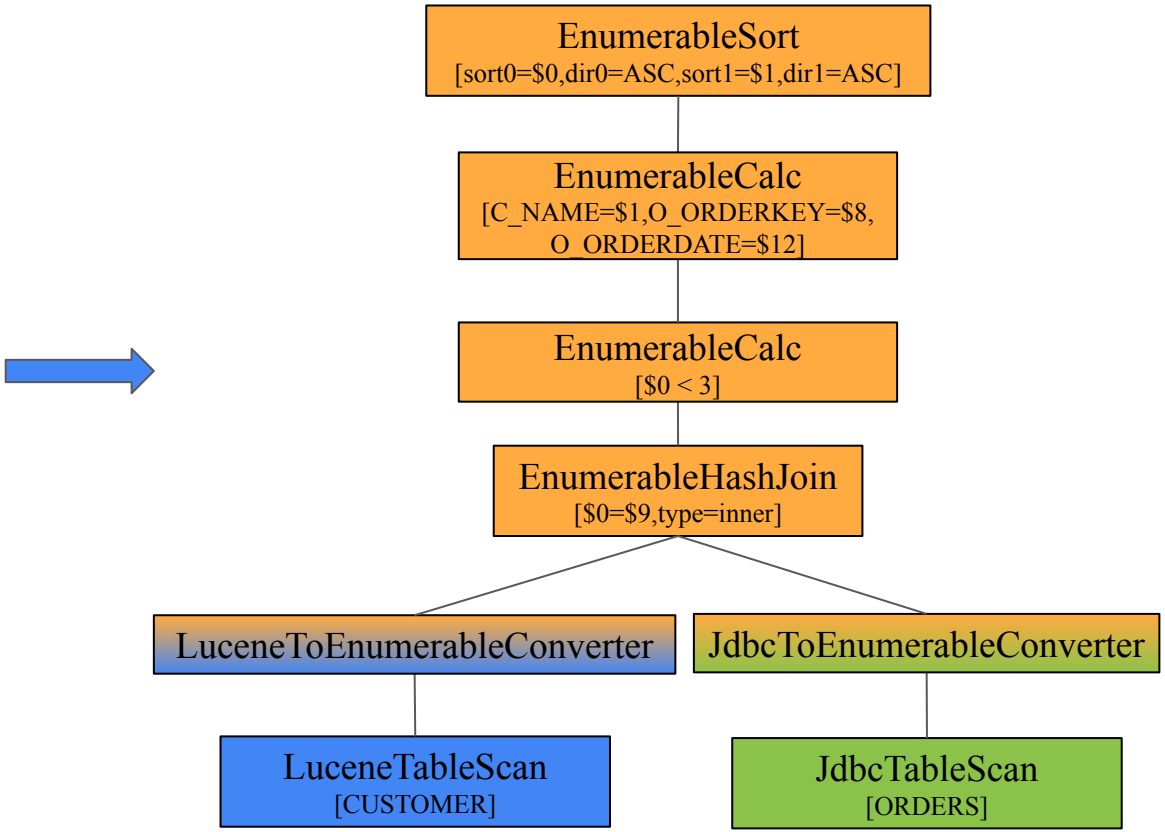
Schemata:

1. Lucene
2. JDBCSchema

Calling conventions:

1. **Enumerable**
2. **Lucene**
3. **JDBC**

What rules?
What operators?



# What do we need?

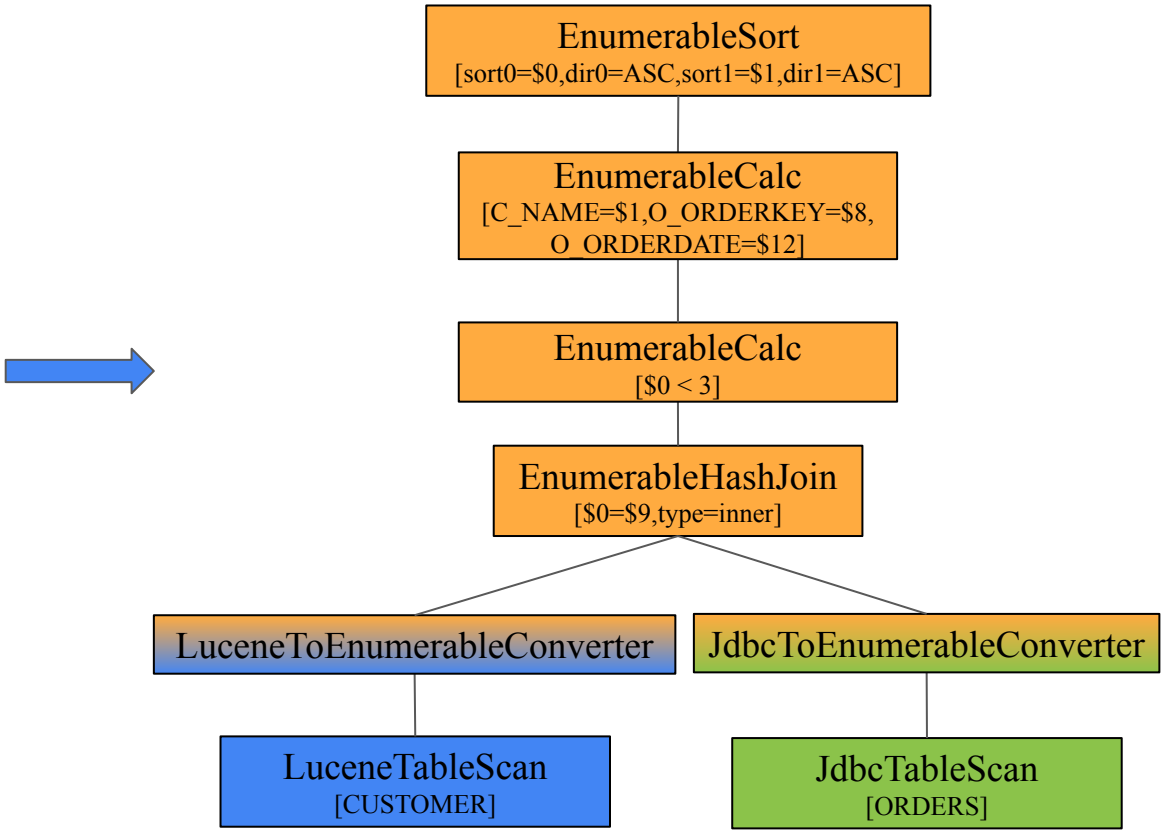
Schemata:

1. Lucene
2. **JDBCSchema**

Calling conventions:

1. **Enumerable**
2. **Lucene**
3. **JDBC**

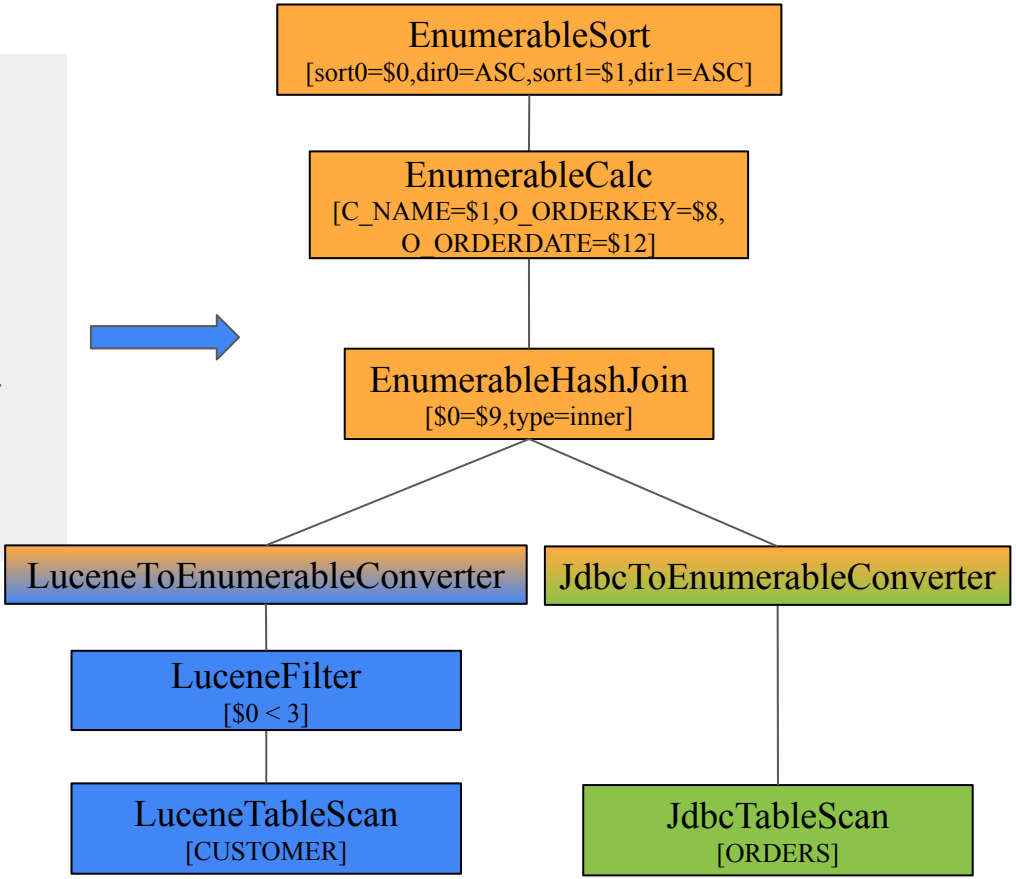
What rules?
What operators?



# Coding module IV: Advanced operator/rules

# What we want to achieve?

```
SELECT c.c_name,
 o.o_orderkey,
 o.o_orderdate
FROM lucene.customer AS c
INNER JOIN hyper.orders AS o
    ON c.c_custkey = o.o_custkey
WHERE c.c_custkey < 3
ORDER BY c.c_name, o.o_orderkey
```



# What do we need?

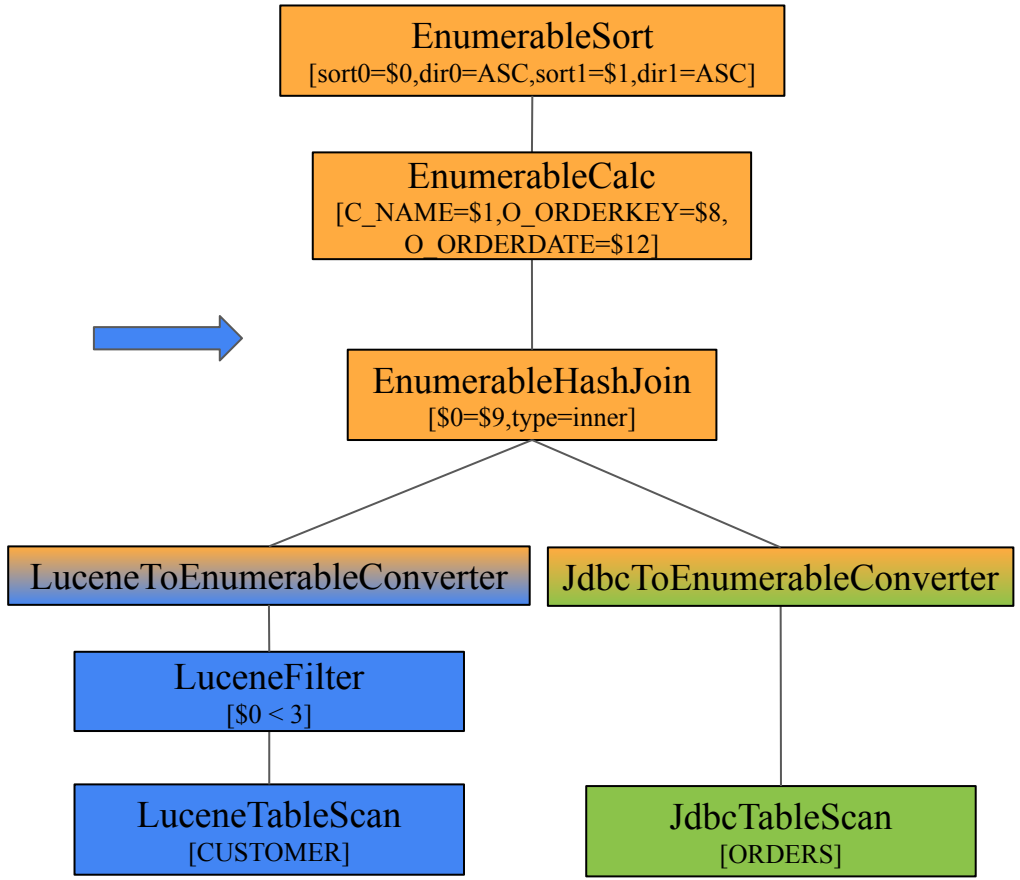
Calling conventions:

1. **Enumerable**
2. **Lucene**
3. **JDBC**

Custom operators: `LuceneFilter`

Custom conversion rules:

`LogicalFilter` → `LuceneFilter`




Stamatis Zampetakis
@szampetak

https://calcite.apache.org

# Thank you!

# Resources

- Calcite project: https://calcite.apache.org
- Calcite-tutorial:
  - Video: https://www.youtube.com/watch?v=mel0W12f_nw
  - Code: https://github.com/zabetak/calcite-tutorial
- What is in a Lucene index? https://youtu.be/T5RmMNDR5XI