SQL for NoSQL and how Apache Calcite can help

FOSDEM 2017

Christian Tzolov

Engineer at Pivotal

BigData, Hadoop, Spring Cloud Dataflow

Apache Committer, PMC member

Apache {Crunch, Geode, HAWQ, ...}

blog.tzolov.net

twitter.com/christzolov

nl.linkedin.com/in/tzolov

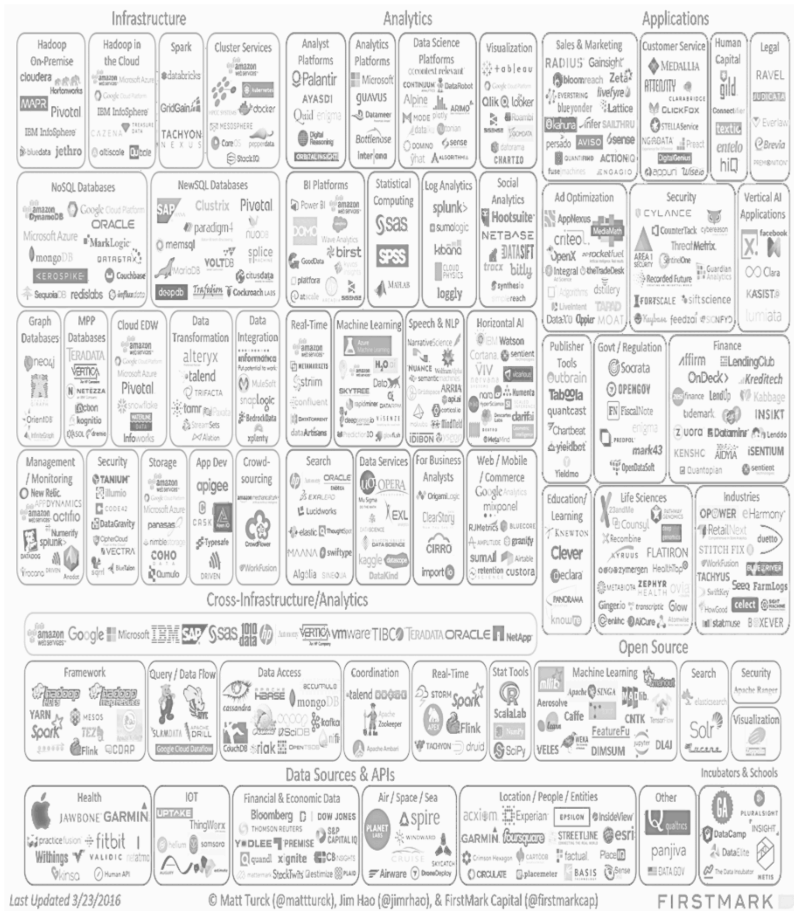
Disclaimer *This talk expresses my personal opinions. It is not read or approved by Pivotal and does not necessarily reflect the views and opinions of Pivotal nor does it constitute any official communication of Pivotal. Pivotal does not support any of the code shared here.*

"It will be interesting to see what happens if an established ***NoSQL*** *database decides to implement a reasonably standard* ***SQL****;*

The only predictable outcome for such an eventuality is plenty of argument."

2012, Martin Fowler, P.J.Sadalage, NoSQL Distilled

Data Big Bang



Why?

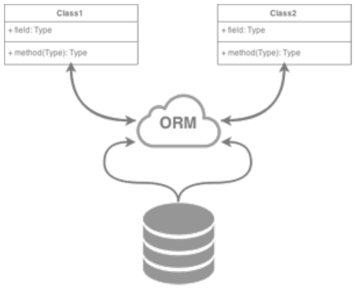
NoSQL Driving Forces

- Rise of Internet Web, Mobile, IoT – Data Volume, Velocity, Variety challenges

  ACID & 2PC clash with Distributed architectures. CAP, PAXOS instead..

- Row-based Relational Model. Object-Relational Impedance Mismatch

  More convenient data models: Datastores, Key/Value, Graph, Columnar, Full-text Search, Schema-on-Load…

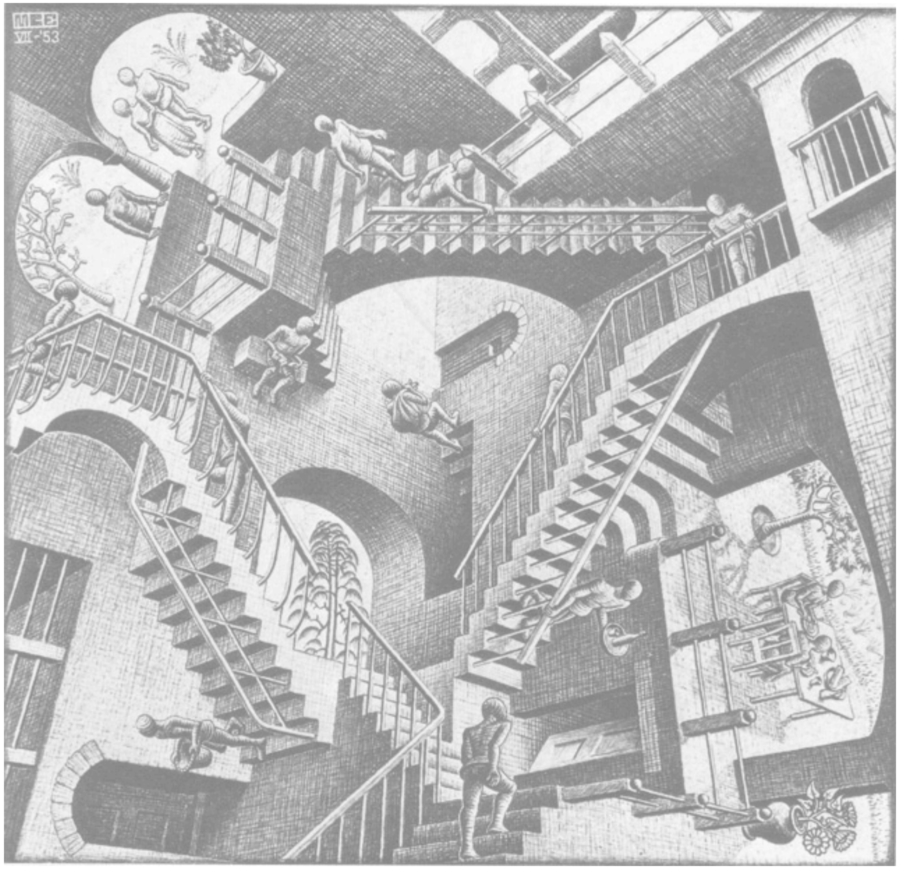


- Infrastructure Automation and Elasticity (Cloud Computing)

  Eliminate operational complexity and cost. Shift from Integration to application databases …

Data Big Bang Implications

- Over 150 commercial NoSQL and BigData systems.
- Organizations will have to mix data storage technologies!
- How to integrate such multitude of data systems?



“Standard” Data Process/Query Language?

- Functional - Unified Programming Model
  - Apache {Beam, Spark, Flink, Apex, Crunch}, Cascading
  - Converging around Apache Beam

```
pcollection.apply(Read.from(”in.txt"))
    .apply(FlatMapElements.via((String word) ->
            asList(word.split("[^a-zA-Z']+")))
    .apply(Filter.by((String word)->!word.isEmpty()))
    .apply(Count.<String>perElement())
```

Batch & Streaming, OLTP

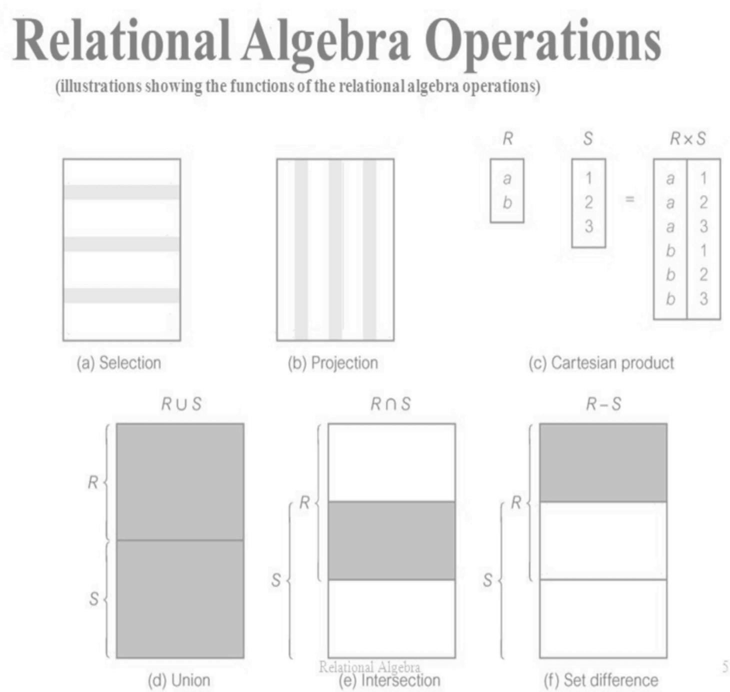
- Declarative - SQL
  - Adopted by many NoSQL Vendors
  - Most Hadoop tasks: Hive and SQL-on-Hadoop
  - Spark SQL - most used production component for 2016
  - Google F1

```
SELECT b."totalPrice", c."firstName”
FROM "BookOrder" as b
INNER JOIN "Customer" as c
    ON b."customerNumber" = c."customerNumber”
WHERE  b."totalPrice" > 0;
```

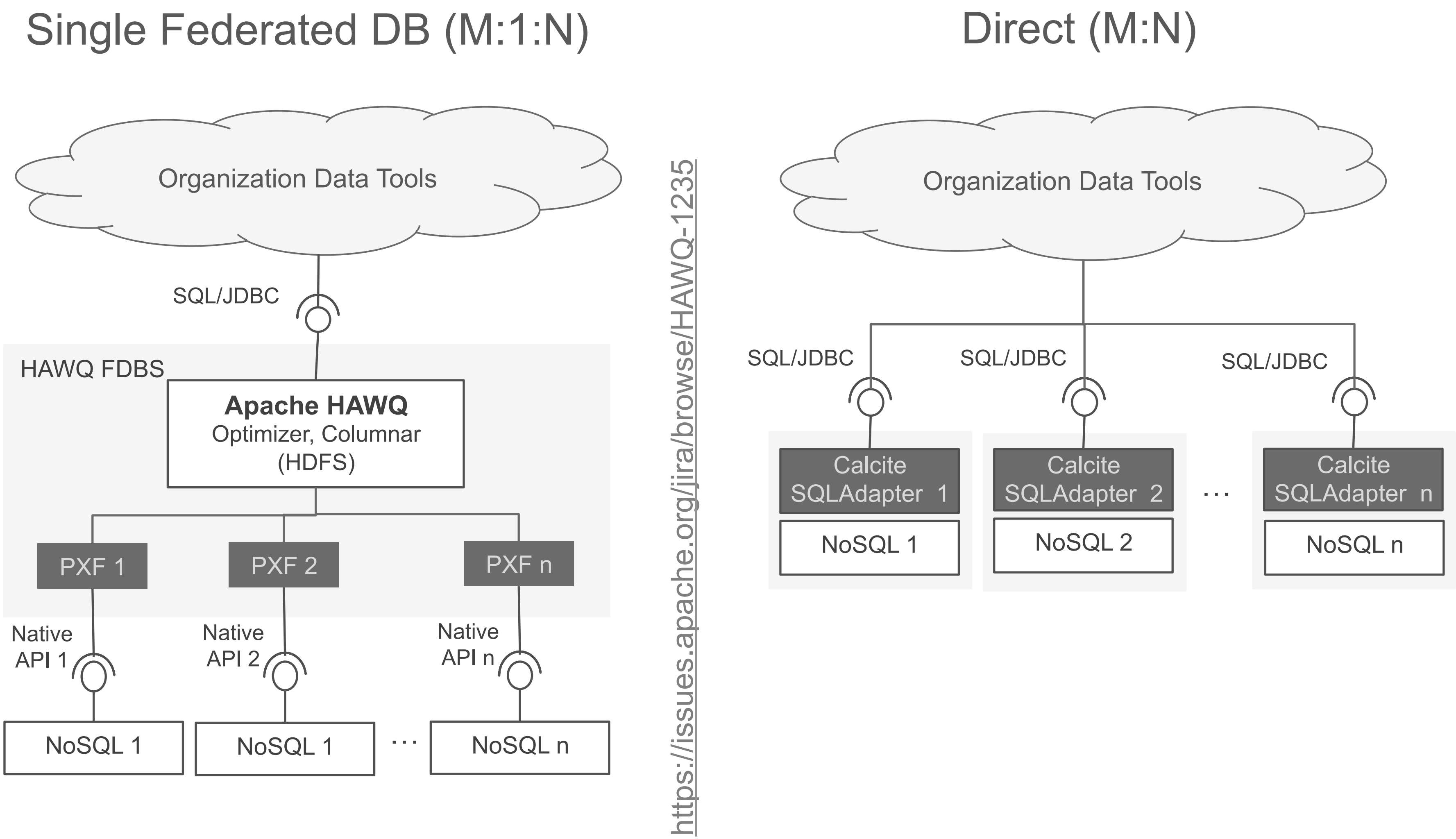
OLAP, EDW, Exploration

SQL for NoSQL?

- Extended Relational Algebra - already present in most NoSql data system
- Relational Expression Optimization – Desirable but hard to implement

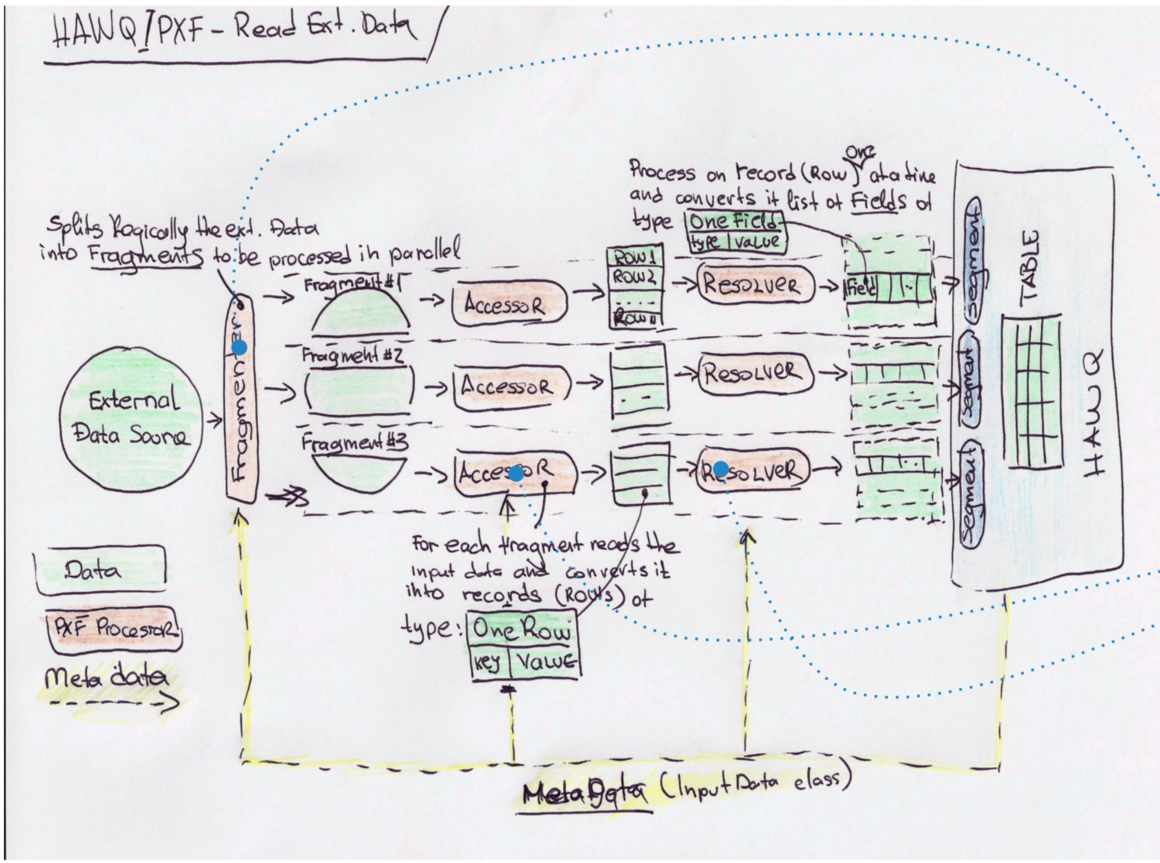


Organization Data - Integrated View



Single Federated Database

Federated External Tables with Apache HAWQ - MPP, Shared-Noting, SQL-on-Hadoop



CREATE EXTERNAL TABLE MyNoSQL
(
  customer_id TEXT,
  first_name TEXT,
  last_name TEXT,
  gender TEXT
)
LOCATION ('pxf://MyNoSQL-URL>?
- FRAGMENTER=MyFragmenter&
- ACCESSOR=MyAccessor&
- RESOLVER=MyResolver&')

FORMAT
'custom'(formatter='pxfwritable_import');

Apache Calcite?

Java framework that allows SQL interface and advanced query optimization, for virtually any data system

- Query Parser, Validator and Optimizer(s)
- JDBC drivers - local and remote
- Agnostic to data storage and processing

Calcite Application

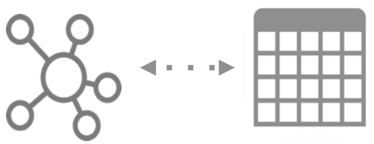
- Apache Apex
- Apache Drill
- Apache Flink
- Apache Hive
- Apache Kylin
- Apache Phoenix
- Apache Samza
- Apache Storm
- Cascading
- Qubole Quark
- SQL-Gremlin

…

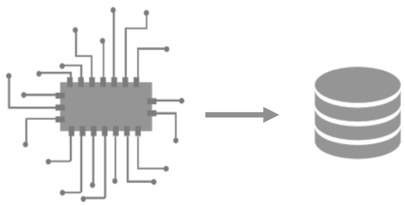
- **Apache Geode**

SQL Adapter Design Choices

SQL completeness vs. NoSql design integrity

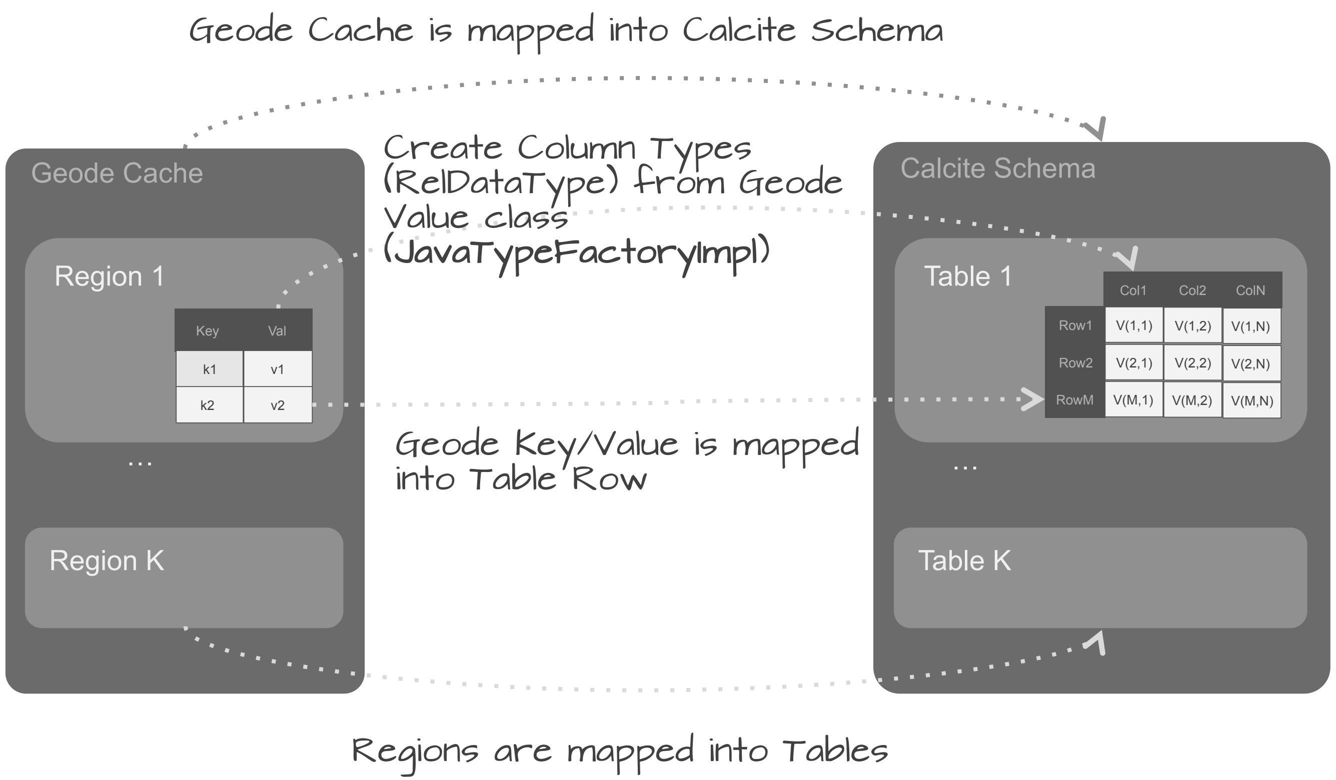


- Data Type Conversion
  - *Catalog* – namespaces accessed in queries
  - *Schema* - collection of schemas and tables
  - *Table* - single data set, collection of rows
  - *RelDataType* – SQL fields types in a Table

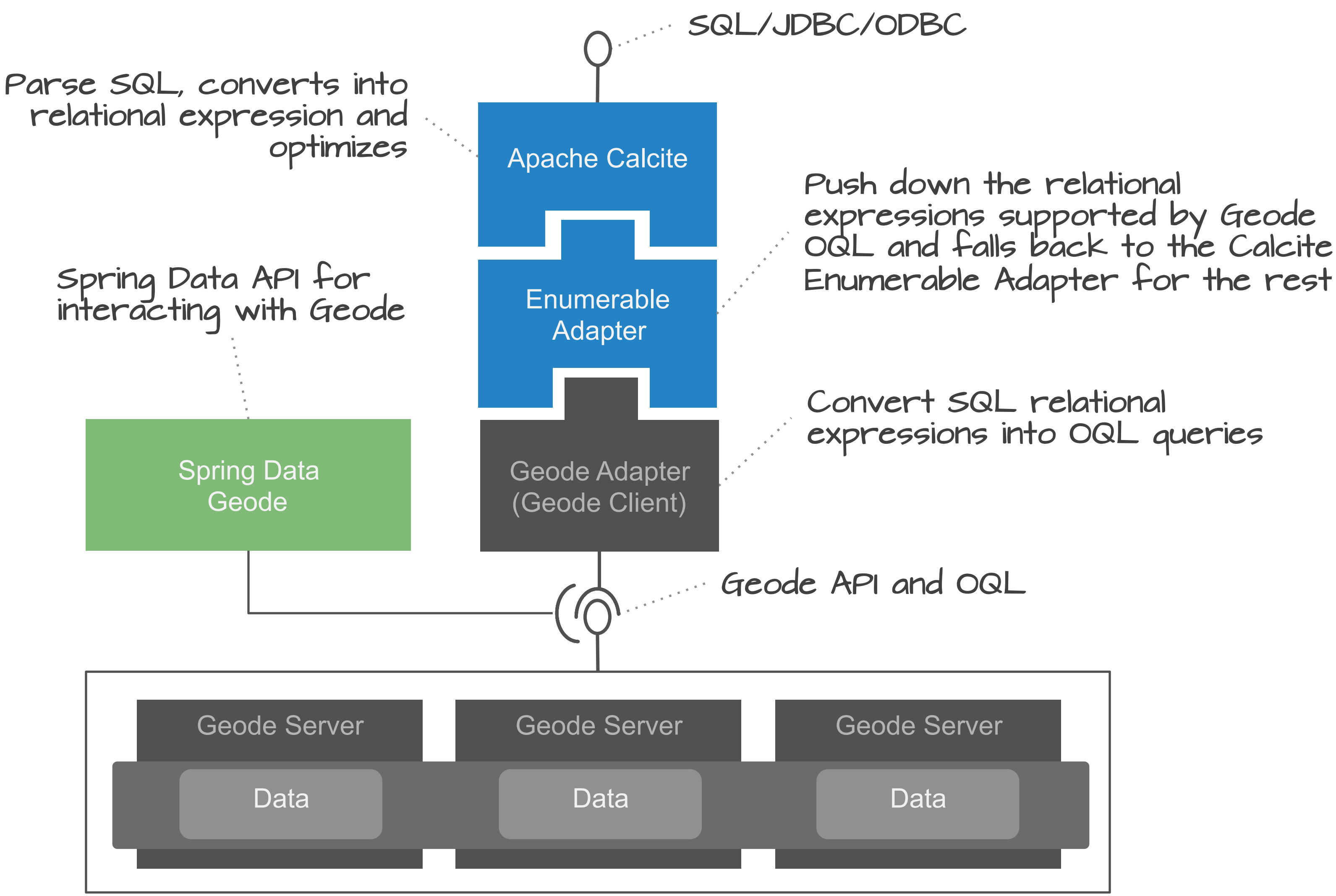


- Move Computation to Data
  - (simple) Predicate Pushdown: *Scan*, *Filter*, *Projection*
  - (complex) Custom Relational Rules and Operations: *Sort, Join, GroupBy* ...

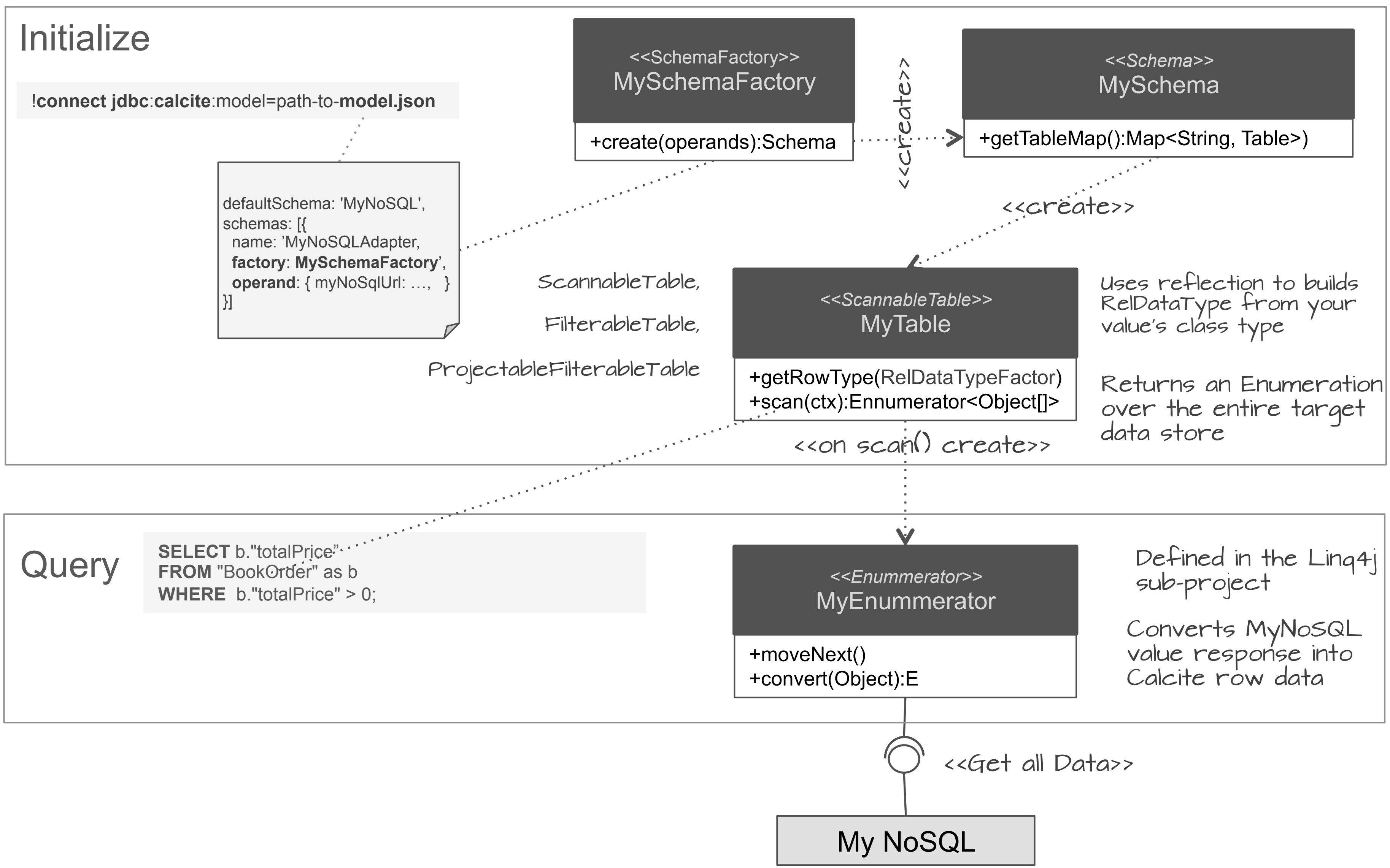
Geode to Calcite Data Types Mapping



Geode Adapter - Overview



Simple SQL Adapter



Non-Relational Tables (Simple)

Scanned without intermediate relational expression.

- **ScannableTable** - can be scanned

```
Enumerable<Object[]> scan(DataContext root);
```

- **FilterableTable** - can be scanned, applying supplied filter expressions

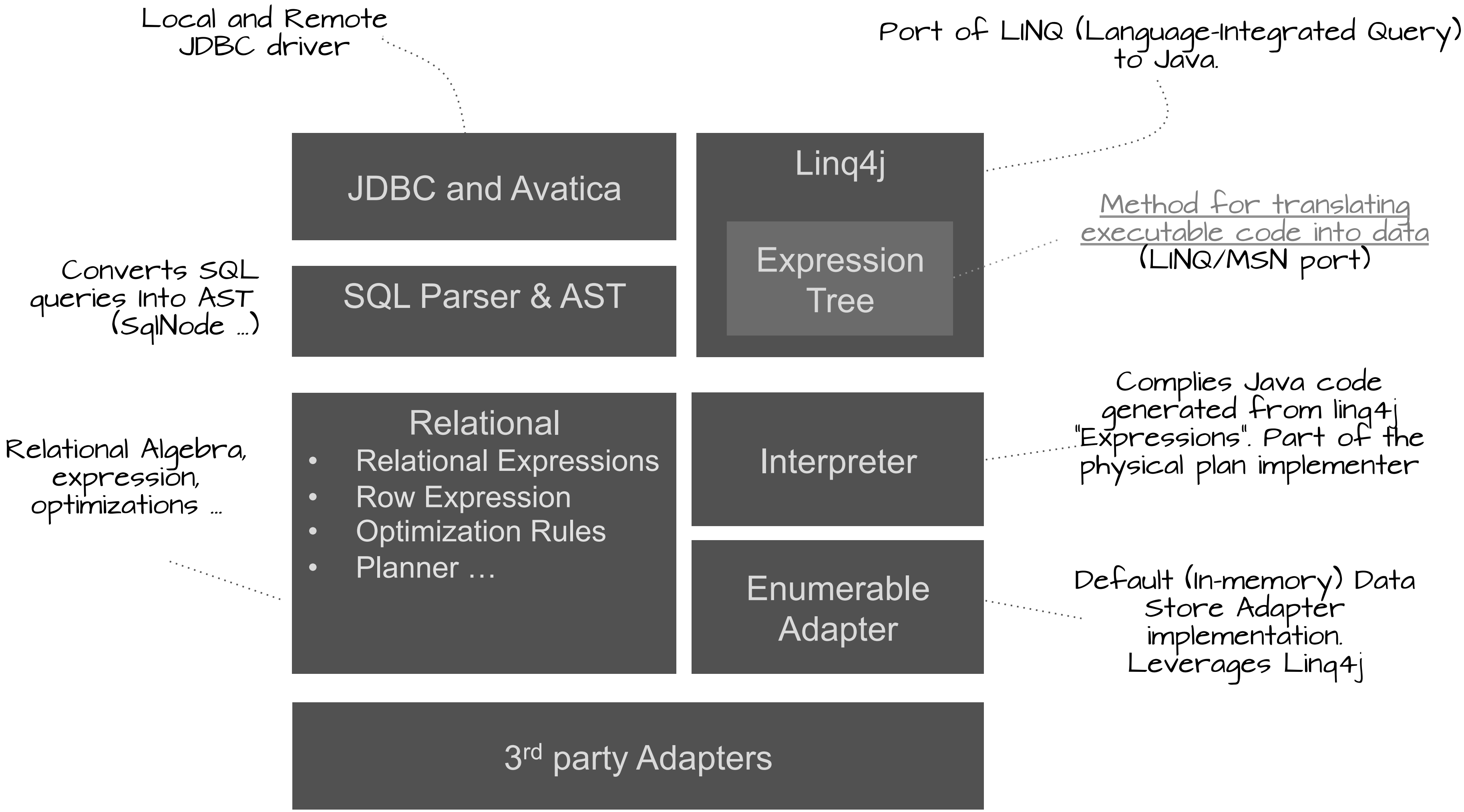
```
Enumerable<Object[]> scan(DataContext root, List<RexNode> filters);
```

- **ProjectableFilterableTable** - can be scanned, applying supplied filter expressions and projecting a given list of columns

```
Enumerable<Object[]> scan(DataContext root, List<RexNode> filters, int[] projects);
```

Calcite Ecosystem

Several “semi-independent” projects.



Calcite SQL Query Execution Flow

1. On new SQL query JDBC delegates to **Prepare** to prepare the query execution

2. Parse SQL, convert to rel. expressions. **Validate** and **Optimize** them

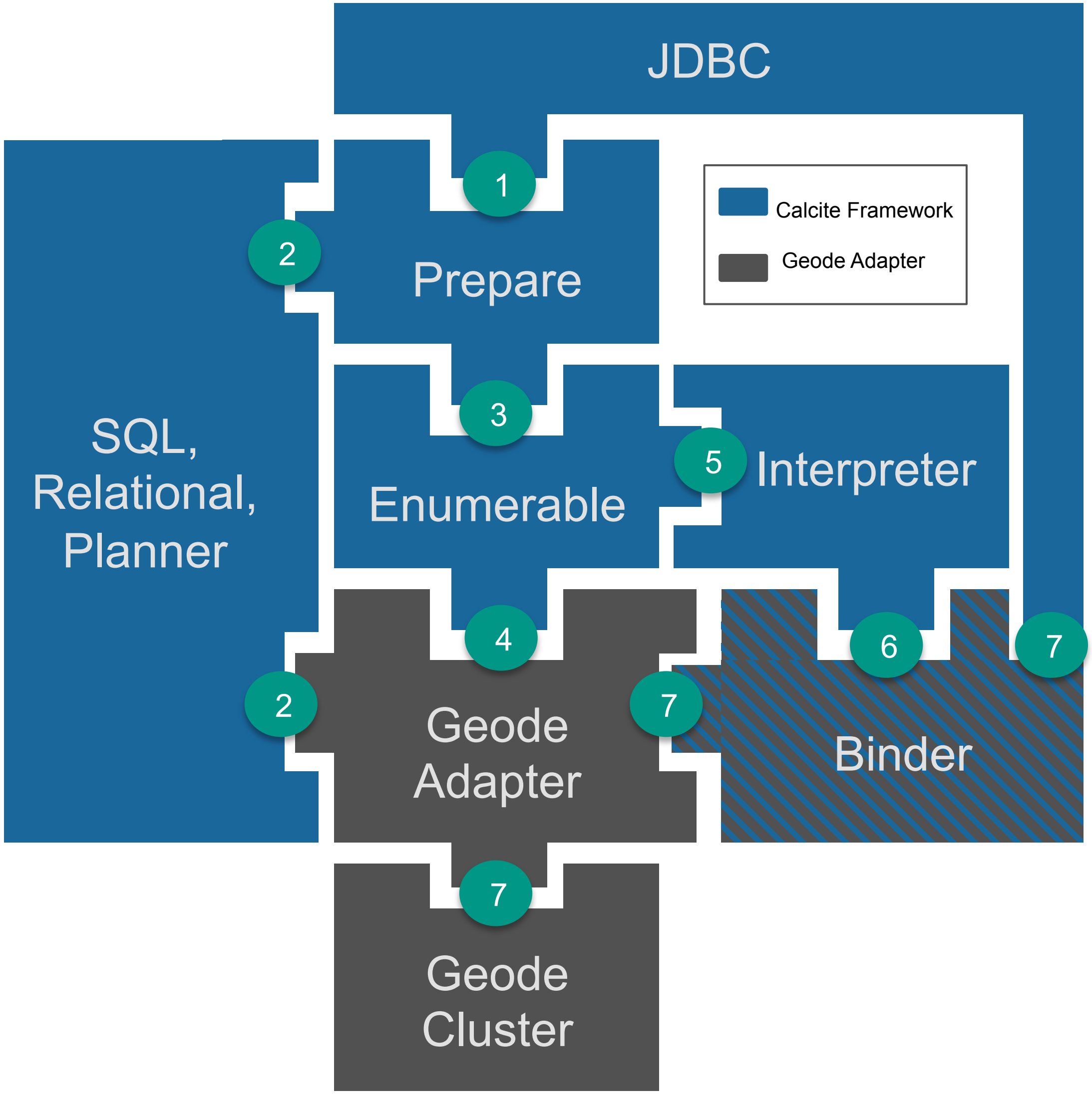
3. Start building a physical plan from the relation expressions

4. Implement the Geode relations and encode them as **Expression tree**

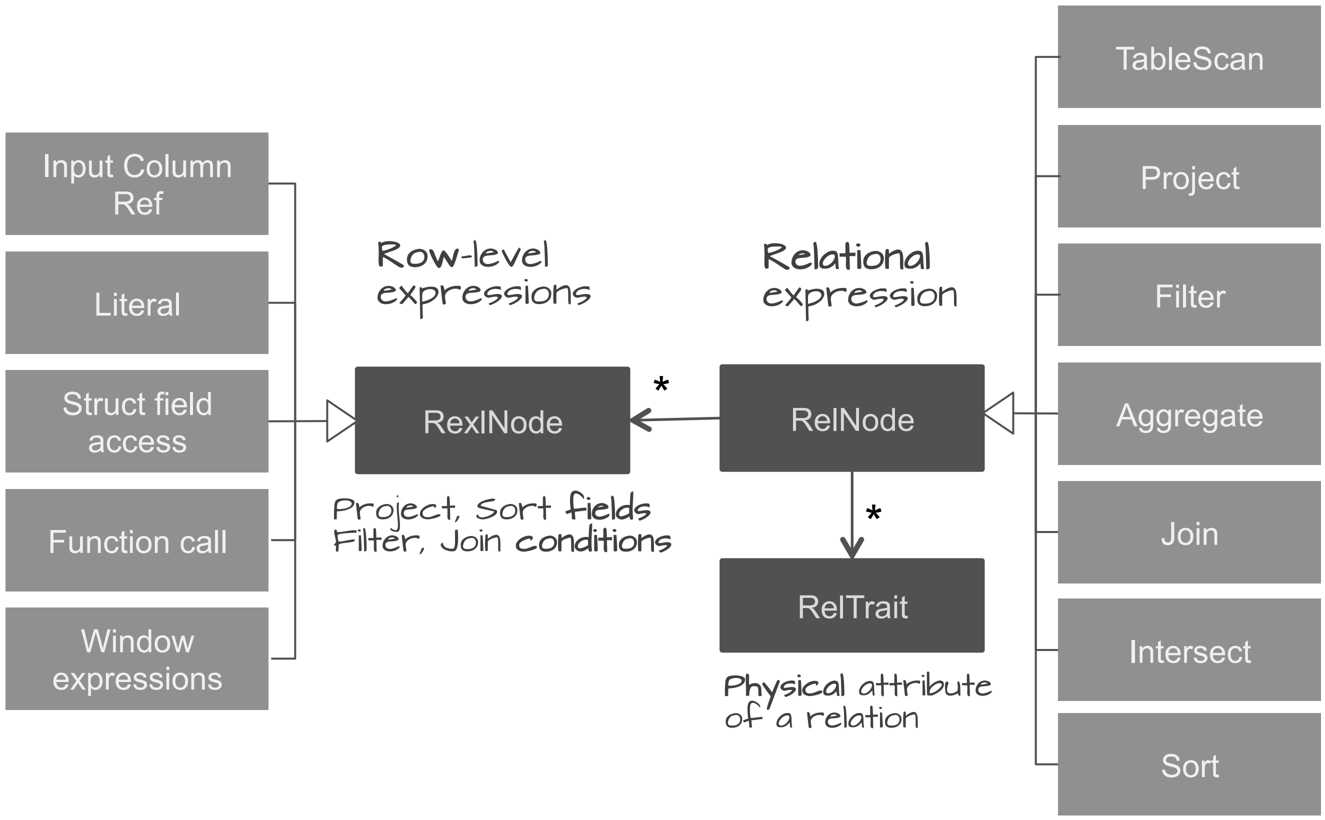
5. Pass the Expression tree to the Interpreter to generate Java code

6. Generate and Compile a Binder instance that on 'bind()' call runs Geodes' query method

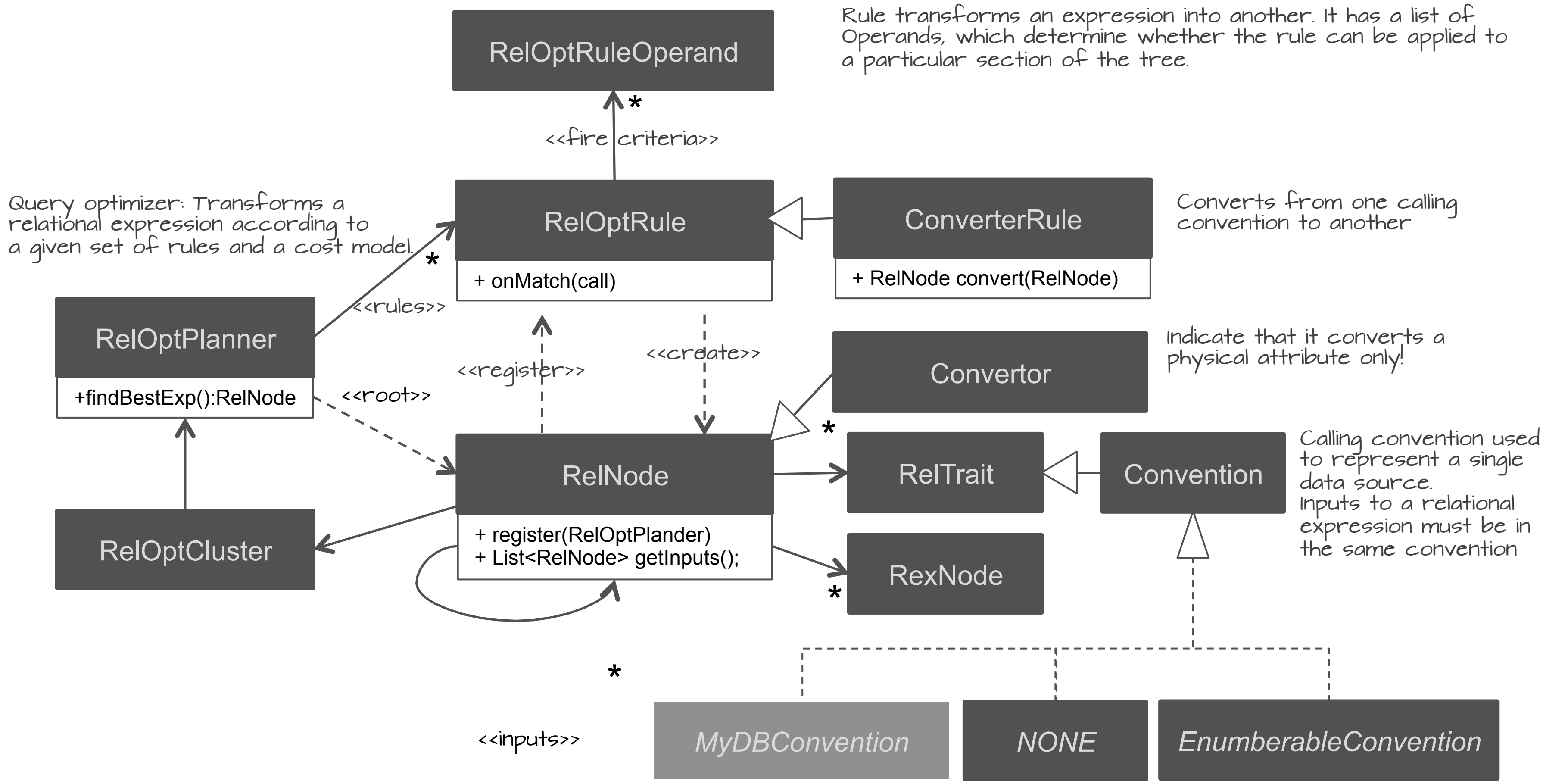
7. JDBC uses the newly compiled Binder to perform the query on the Geode Cluster



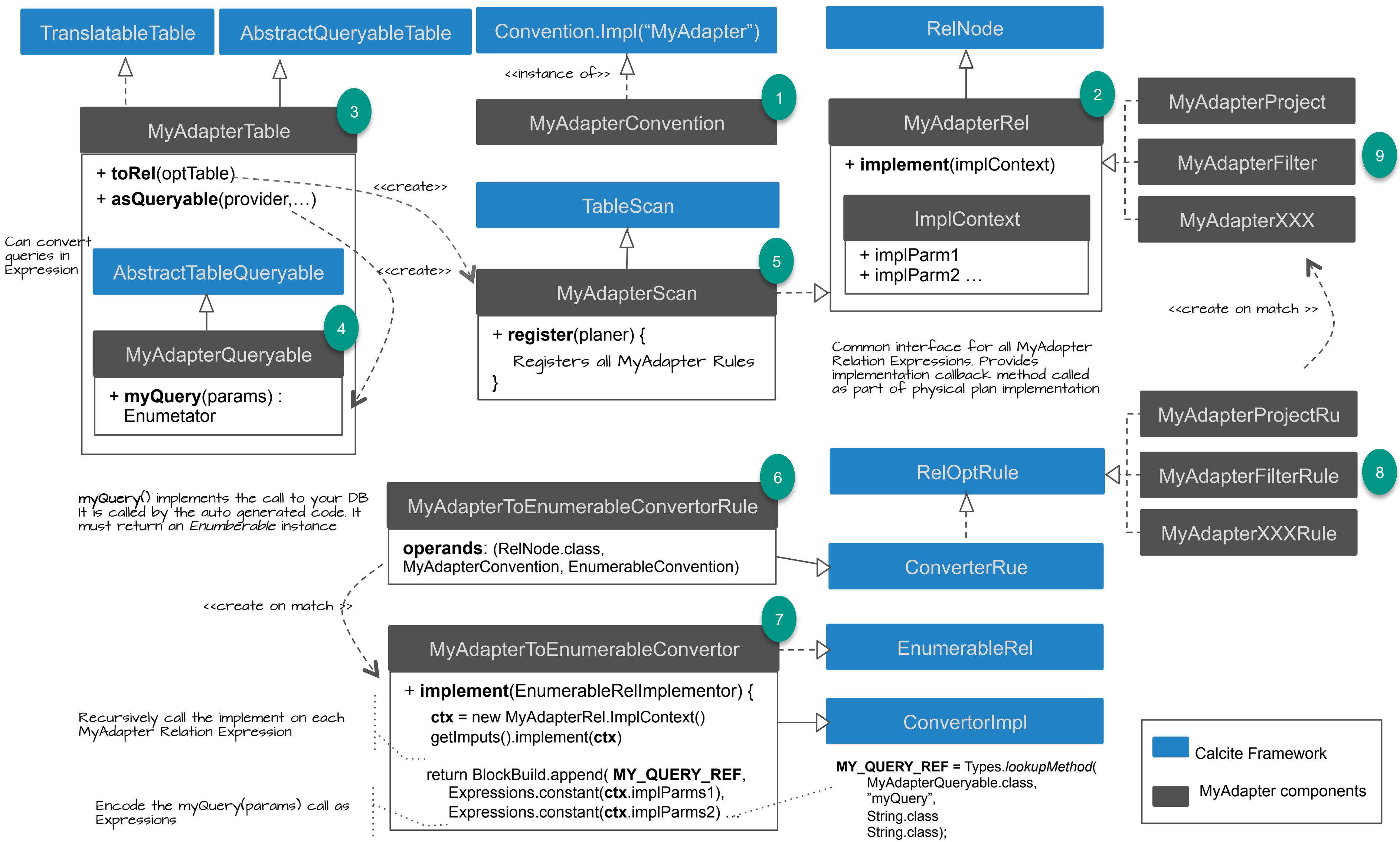
Calcite Relational Expressions



Calcite Relational Expressions

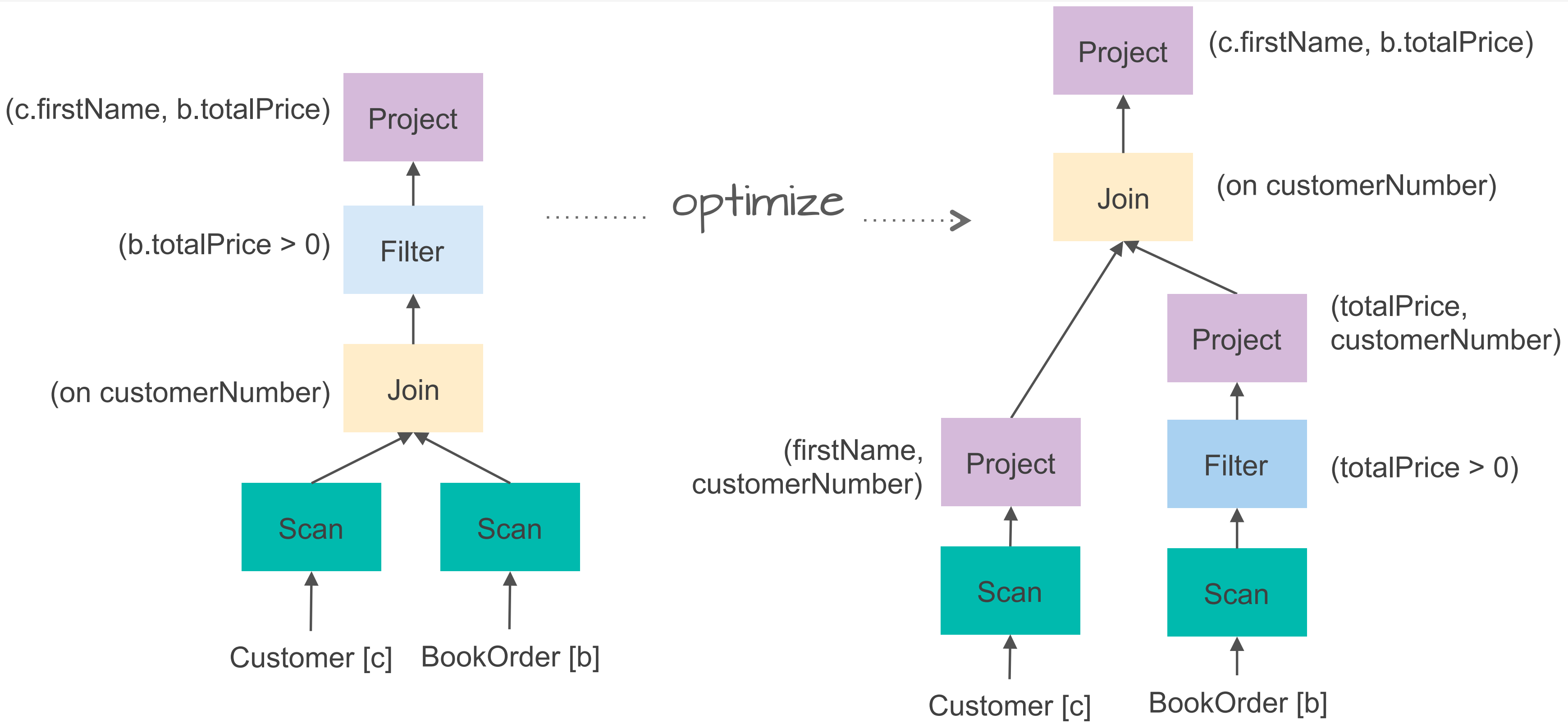


Calcite Adapter Implementation Patterns



Relational Algebra

```
SELECT b."totalPrice", c."firstName”
FROM "BookOrder" as b
INNER JOIN "Customer" as c ON b."customerNumber" = c."customerNumber”
WHERE  b."totalPrice" > 0;
```



Calcite with Geode - Without Implementation

SELECT b."totalPrice", c."firstName”
FROM "BookOrder" as b
INNER JOIN "Customer" as c **ON** b."customerNumber" = c."customerNumber”
WHERE b."totalPrice" > 0;

```
'LogicalProject(totalPrice=[$6], firstName=[$8])
  LogicalFilter(condition=[>($6, 0)])
    LogicalJoin(condition=[=($5, $7)], joinType=[inner])
      LogicalTableScan(table=[[TEST, BookOrder]])
      LogicalTableScan(table=[[TEST, Customer]])
'
```

Calcite with Geode – Scannable Table (Simple)

SELECT b."totalPrice", c."firstName”
FROM "BookOrder" as b
INNER JOIN "Customer" as c **ON** b."customerNumber" = c."customerNumber”
WHERE b."totalPrice" > 0;

```
'EnumerableCalc(expr#0..3=[{inputs}], totalPrice=[$t1], firstName=[$t3])
  EnumerableJoin(condition=[=($0, $2)], joinType=[inner])
    EnumerableCalc(expr#0..6=[{inputs}], expr#7=[0], expr#8=[>($t6, $t7)],
      EnumerableInterpreter
        BindableTableScan(table=[[TEST, BookOrder]])
    EnumerableCalc(expr#0..4=[{inputs}], proj#0..1=[{exprs}])
      EnumerableInterpreter
        BindableTableScan(table=[[TEST, Customer]])
```

Calcite with Geode – Relational (Complex)

SELECT b."totalPrice", c."firstName”
FROM "BookOrder" as b
INNER JOIN "Customer" as c **ON** b."customerNumber" = c."customerNumber”
WHERE b."totalPrice" > 0;

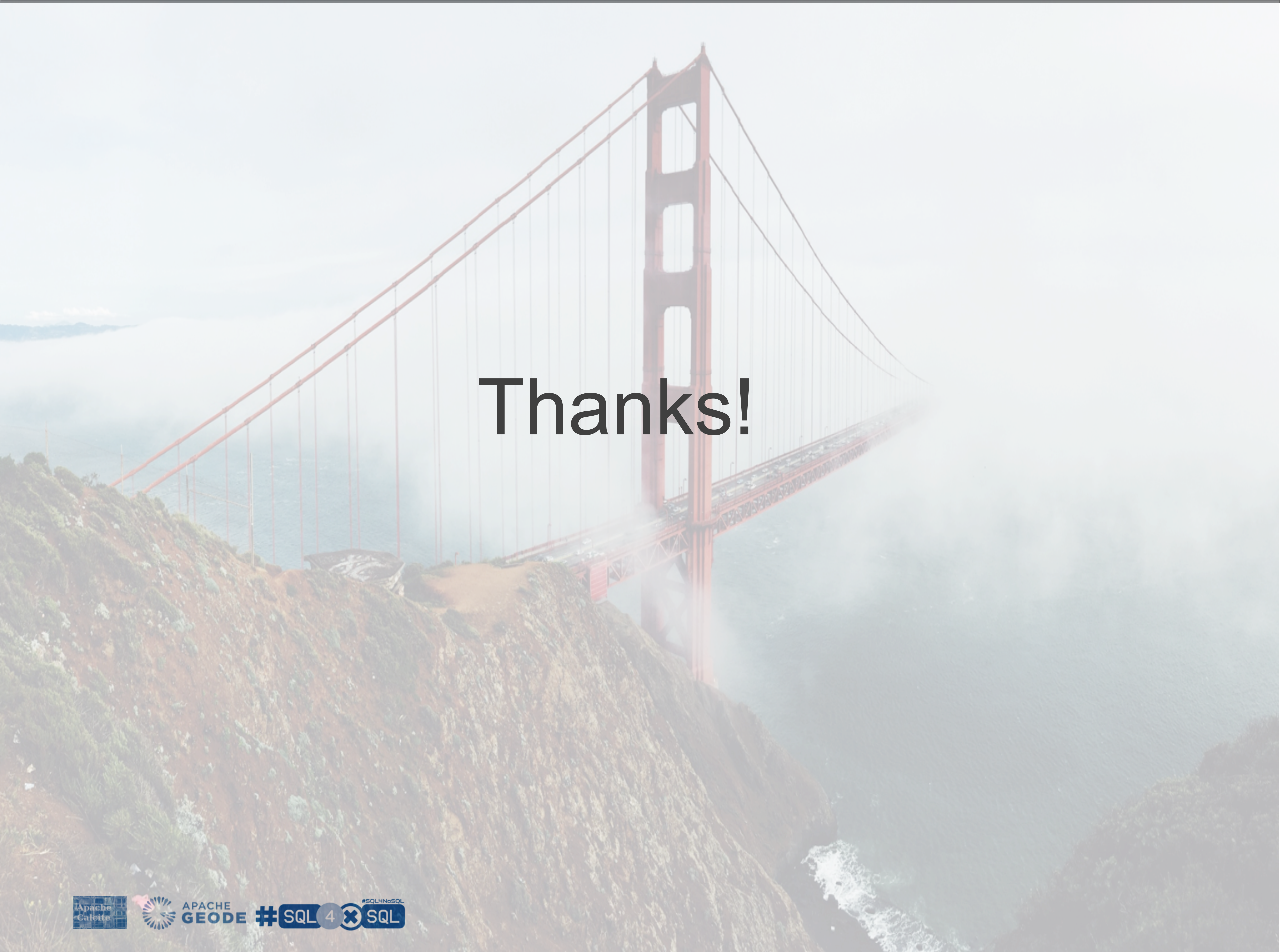
```
'PLAN'
'EnumerableCalc(expr#0..3=[{inputs}], totalPrice=[$t0], firstName=[$t3])
  EnumerableJoin(condition=[=($1, $2)], joinType=[inner])
    GeodeToEnumerableConverterRel
      GeodeProjectRel(totalPrice=[$3], customerNumber=[$6])
        GeodeFilterRel(condition=[>($3, 0)])
          GeodeTableScanRel(table=[[TEST, BookOrder]])
    GeodeToEnumerableConverterRel
      GeodeProjectRel(customerNumber=[$0], firstName=[$1])
        GeodeTableScanRel(table=[[TEST, Customer]])
'
```

Calcite JDBC Connection

```java
public static void main(String[] args) throws Exception {

    Properties info = new Properties();
    info.put("model",
            "inline:"
                    + "{\n"
                    + "  version: '1.0',\n"
                    + "  schemas: [\n"
                    + "     {\n"
                    + "       type: 'custom',\n"
                    + "       name: 'TEST',\n"
                    + "       factory: 'org.apache.calcite.adapter.geode.rel.GeodeSchemaFactory',\n"
                    + "       operand: {\n"
                    + "         locatorHost: 'localhost', \n"
                    + "         locatorPort: '10334', \n"
                    + "         regions: 'BookMaster,Customer,InventoryItem,BookOrder', \n"
                    + "         pdxSerializablePackagePath: 'net.tzolov.geode.bookstore.domain.*' \n"
                    + "       }\n"
                    + "     }\n"
                    + "   ]\n"
                    + "}");

    Class.forName("org.apache.calcite.jdbc.Driver");

    Connection connection = DriverManager.getConnection("jdbc:calcite:", info);
    Statement statement = connection.createStatement();
    ResultSet resultSet = statement.executeQuery(
            "SELECT b.\"totalPrice\", c.\"firstName\" " +
            "FROM \"TEST\".\"BookOrder\" as b " +
            "INNER JOIN \"TEST\".\"Customer\" as c ON b.\"customerNumber\" = c.\"customerNumber\" " +
            "WHERE  b.\"totalPrice\" > 0");


    final StringBuilder buf = new StringBuilder();
    while (resultSet.next()) {
        ResultSetMetaData metaData = resultSet.getMetaData();
        for (int i = 1; i <= metaData.getColumnCount(); i++)
            buf.append(i > 1 ? "; " : "").append(metaData.getColumnLabel(i)).append("=").append(resultSet.getObject(i));
        System.out.println(buf.toString());
        buf.setLength(0);
    }
    resultSet.close();
    statement.close();
    connection.close();
}
```

References

- Big Data is Four Different Problems, 2016, M.Stonebraker: https://www.youtube.com/watch?v=S79-buNhdhI
- Turning Database Inside-Out, 2015 (M. Kleppmann) https://www.confluent.io/blog/turning-the-database-inside-out-with-apache-samza
- NoSQL Distilled, 2012 (Pramod J. Sadalage and M.Fowler) https://martinfowler.com/books/nosql.html
- Architecture of a Database System, 2007 (J.M. Hellerstein, M. Stonebraker, J. Hamilton)http://db.cs.berkeley.edu/papers/fntdb07-architecture.pdf
- ORCA: A Modular Query Optimizer Architecture for Big Data: http://15721.courses.cs.cmu.edu/spring2016/papers/p337-soliman.pdf
- Apache Geode Project (2016) : http://geode.apache.org
- Geode Object Query Language (OQL) : http://bit.ly/2eKywgp
- Apache Calcite Project (2016) : https://calcite.apache.org
- Apache Geode Adapter for Apache Calcite: https://github.com/tzolov/calcite
- Relational Algebra Operations: https://www.coursera.org/learn/data-manipulation/lecture/4JKs1/relational-algebra-operators-union-difference-selection



Thanks!

Apache Geode?

"… in-memory, distributed database with strong consistency built to support low latency transactional applications at extreme scale"

Why Apache Geode?

China Railway

5,700 train stations
4.5 million tickets per day
20 million daily users
1.4 billion page views per day
40,000 visits per second

Indian Railways

7,000 stations
72,000 miles of track
23 million passengers daily
120,000 concurrent users
10,000 transactions per minute

https://pivotal.io/big-data/case-study/distributed-in-memory-data-management-solution
https://pivotal.io/big-data/case-study/scaling-online-sales-for-the-largest-railway-in-the-world-china-railway-corporation

Geode Features

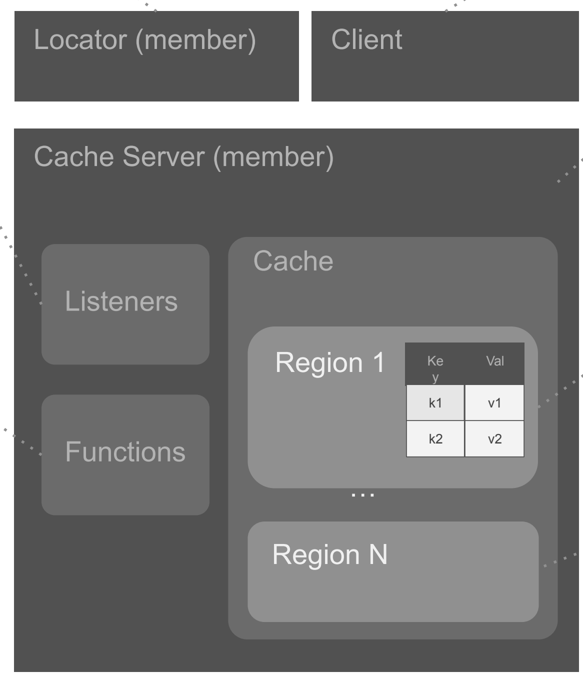
- In-Memory Data Storage
  - Over 100TB Memory
  - JVM Heap + Off Heap
- Any Data Format
  - Key-Value/Object Store
- ACID and JTA Compliant Transactions
- HA and Linear Scalability
- Strong Consistency
- Streaming and Event Processing
  - Listeners
  - Distributed Functions
  - Continuous OQL Queries
- Multi-site / Inter-cluster
- Full Text Search (Lucene indexes)
- Embedded and Standalone
- Top Level Apache Project

Apache Geode Concepts

Locator - tracks system members and provides membership information

Client -read and modify the content of the distributed system

Listener - event handler. Registers for one or more events and notified when they occur



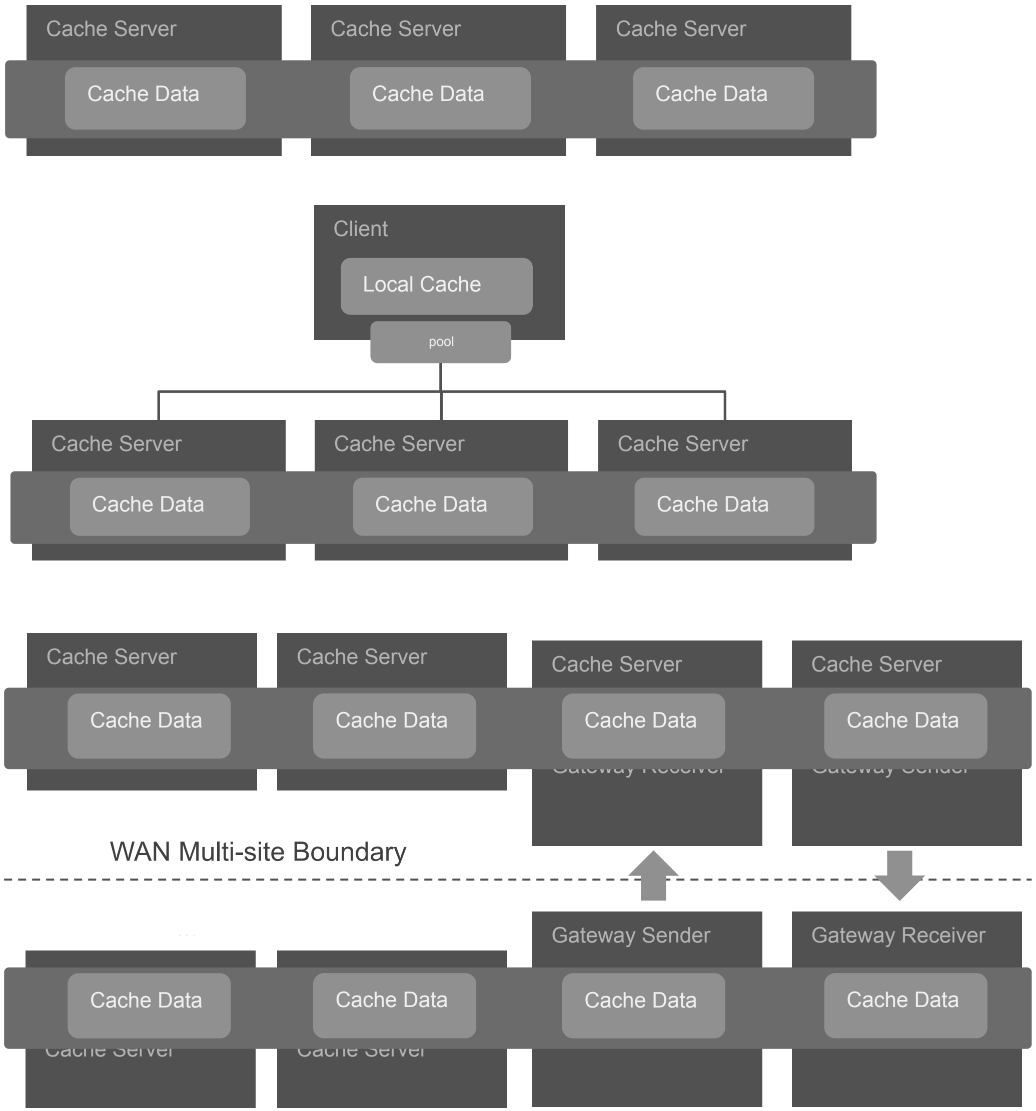
CacheServer - process connected to the distributed system with created **Cache**

Functions - distributed, concurrent data processing

Region - consistent, **distributed Map** (key-value), Partitioned or Replicated

Cache - In-memory collection of **Regions**

Geode Topology



Peer-to-Peer

Client-Server

Multi-Site