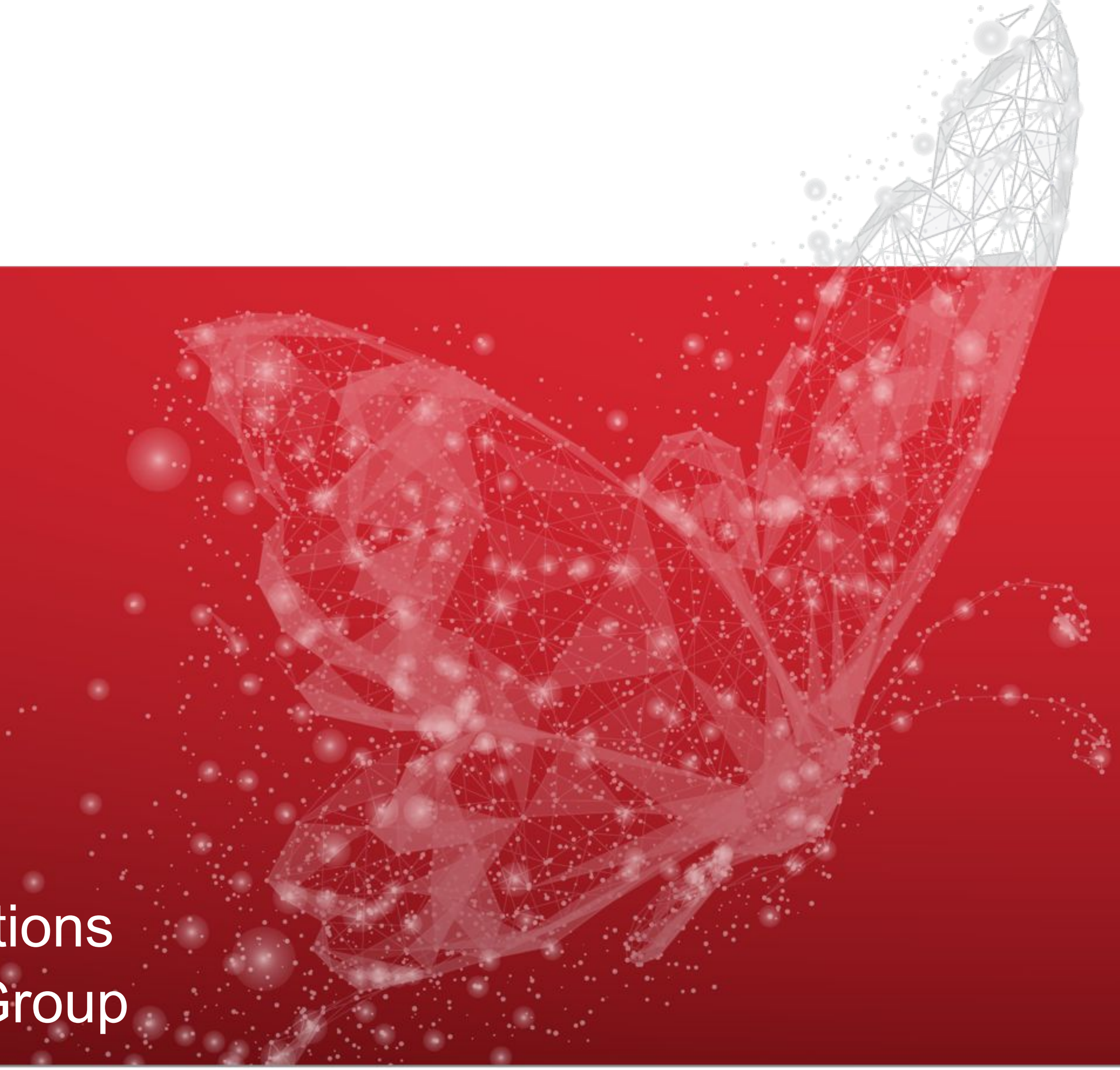
Getting Started With Apache Ignite SQL

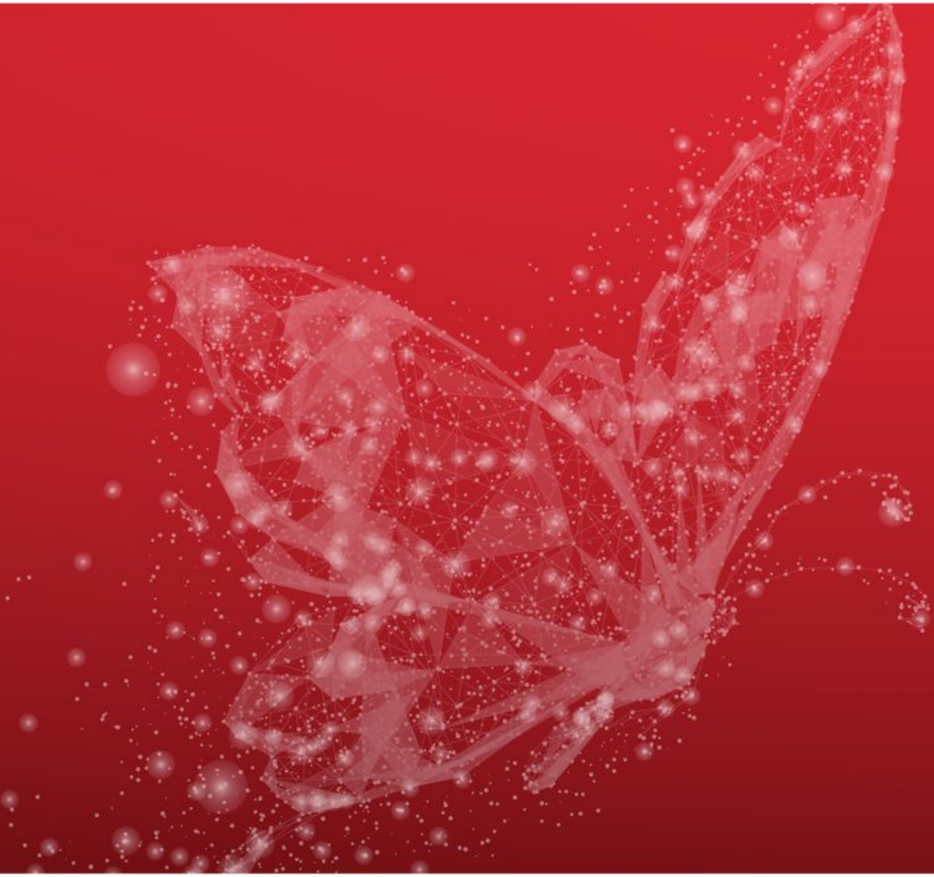


Denis Magda, GridGain Developer Relations
Igor Seliverstov, GridGain Architecture Group

Topics

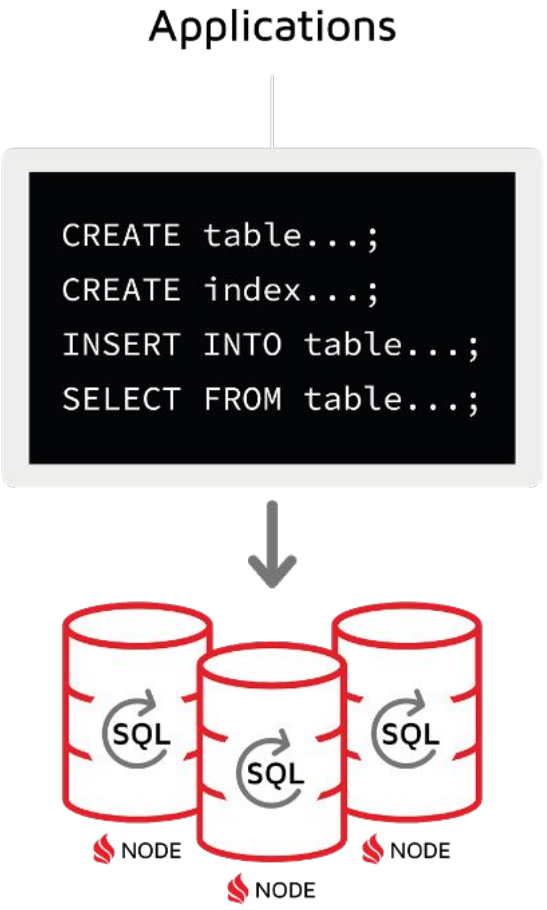
- Ignite SQL Basics: DML, DDL, connectivity, configuration
- Affinity Co-Location and Distributed JOINs
- Beyond Memory Capacity: Disk Tier Usage and Memory Quotas
- Ignite SQL Evolution With Apache Calcite

Ignite SQL Basics



Ignite SQL = ANSI SQL at Scale

- ANSI-99 DML and DDL syntax
  - SELECT, UPDATE, CREATE…
- Distributed joins, grouping, sorting
- Schema changes in runtime
  - ALTER TABLE, CREATE/DROP INDEX
- Works with in-memory and *disk-only* records
  - *If Ignite Persistence is used as a disk tier*

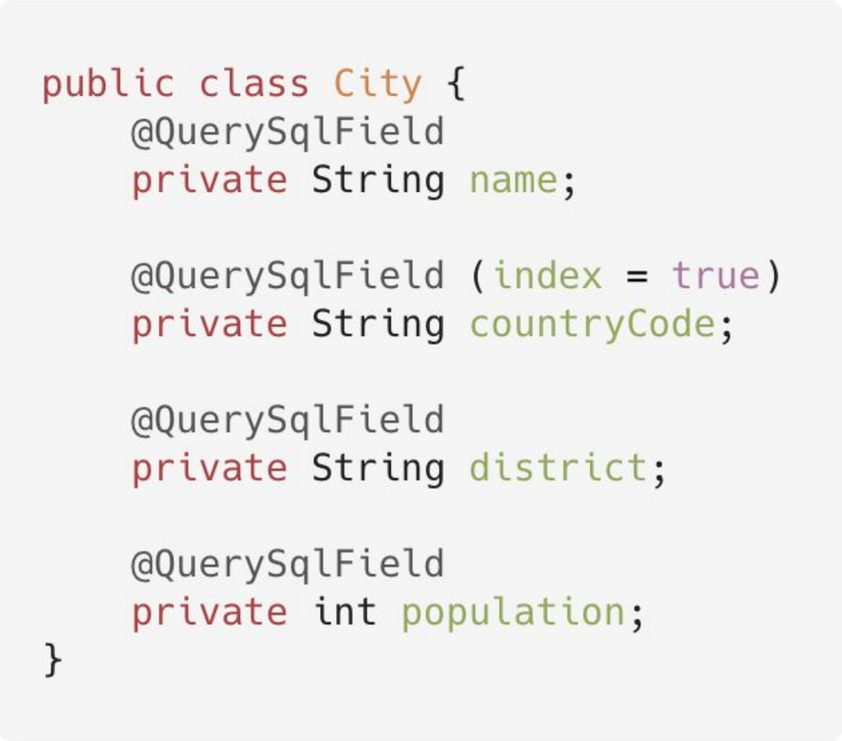


```
CREATE table...;
CREATE index...;
INSERT INTO table...;
SELECT FROM table...;
```

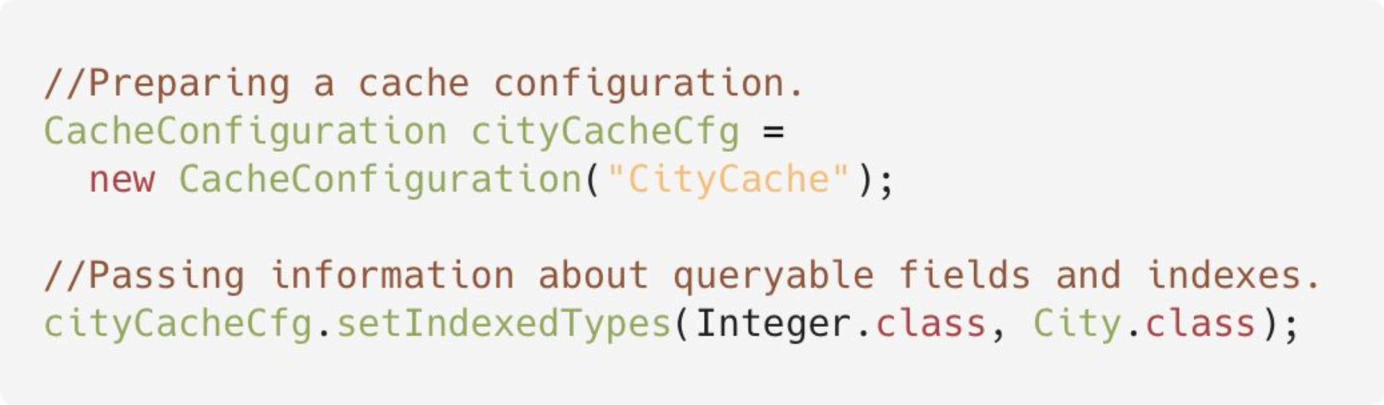
Connectivity Options

- Thick Client APIs
  - Java, C#/.NET, C++
- JDBC and ODBC drivers
- Thin Client APIs
  - Multi-language support

Configuration Option #1: Programmatically With Annotations



```java
public class City {
    @QuerySqlField
    private String name;

    @QuerySqlField (index = true)
    private String countryCode;

    @QuerySqlField
    private String district;

    @QuerySqlField
    private int population;
}
```

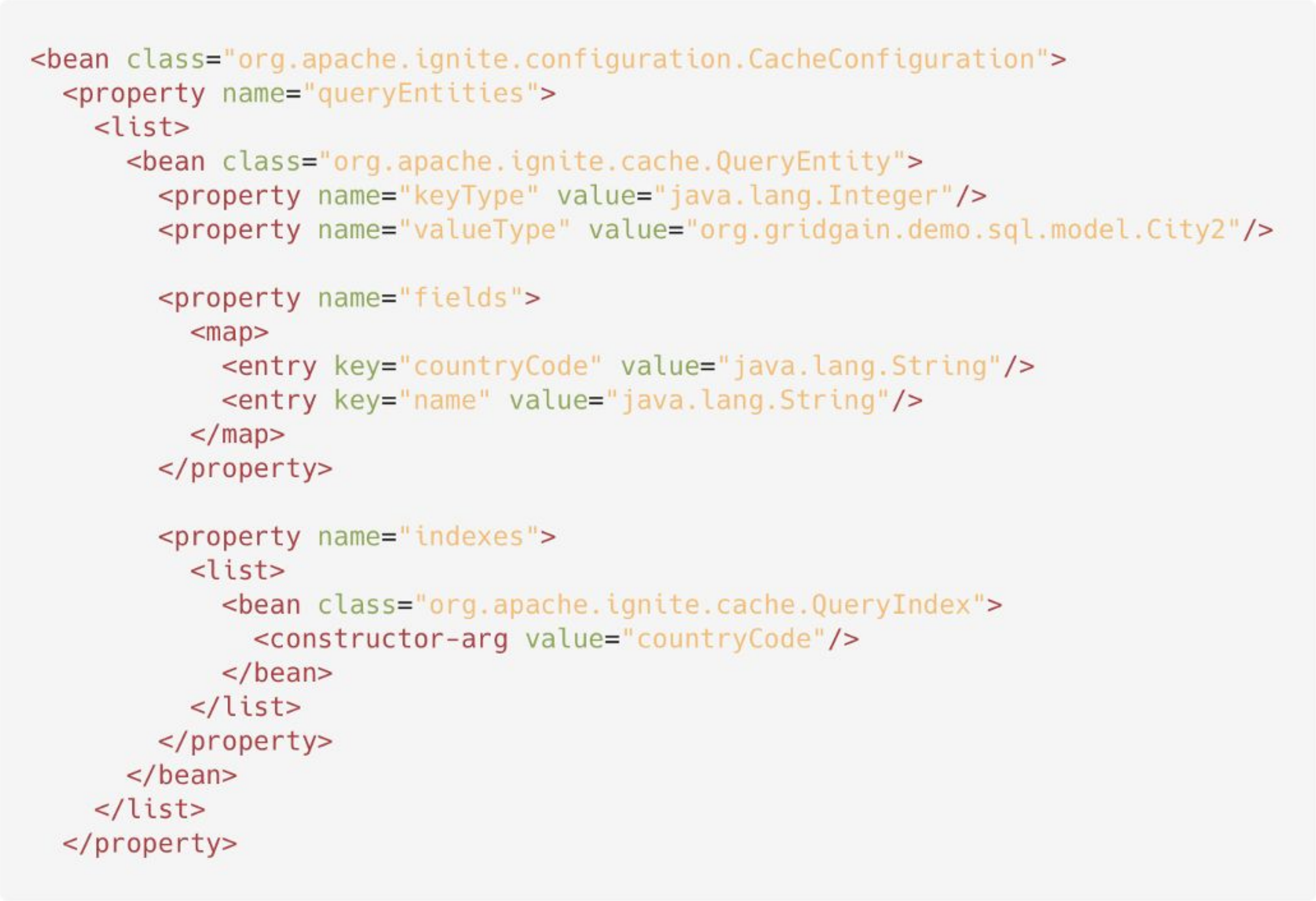


```java
//Preparing a cache configuration.
CacheConfiguration cityCacheCfg =
  new CacheConfiguration("CityCache");

//Passing information about queryable fields and indexes.
cityCacheCfg.setIndexedTypes(Integer.class, City.class);
```

Usage Scenario:

- Spring-style development by annotating POJOs
- DDL can be used to apply changes in runtime.

Configuration Option #2:
Spring XML With Query Entities

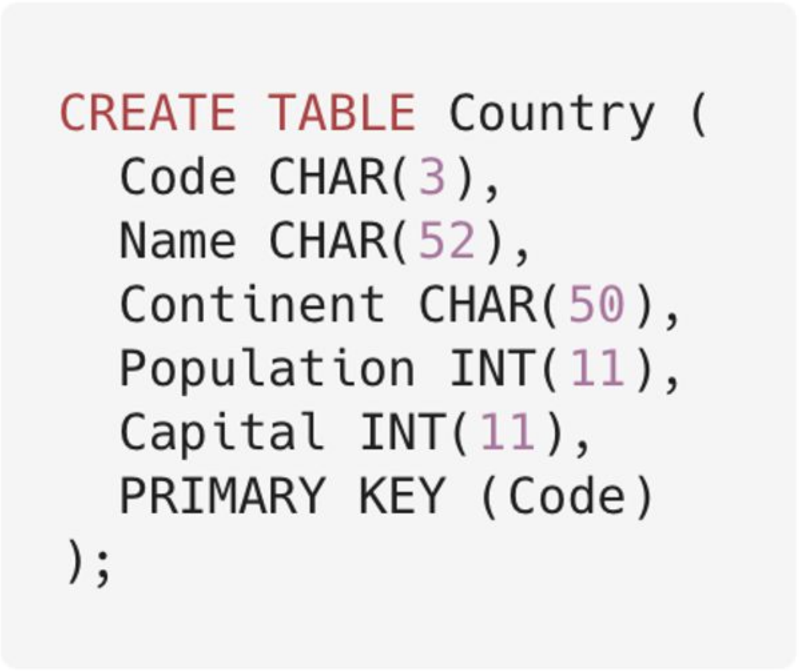


```
<bean class="org.apache.ignite.configuration.CacheConfiguration">
  <property name="queryEntities">
    <list>
      <bean class="org.apache.ignite.cache.QueryEntity">
        <property name="keyType" value="java.lang.Integer"/>
        <property name="valueType" value="org.gridgain.demo.sql.model.City2"/>

        <property name="fields">
          <map>
            <entry key="countryCode" value="java.lang.String"/>
            <entry key="name" value="java.lang.String"/>
          </map>
        </property>

        <property name="indexes">
          <list>
            <bean class="org.apache.ignite.cache.QueryIndex">
              <constructor-arg value="countryCode"/>
            </bean>
          </list>
        </property>
      </bean>
    </list>
  </property>
```

Usage Scenario:

- Ignite as a cache that writes-through changes to an external database.
- DDL can be used to apply changes in runtime.

Configuration Option #3:
In Pure SQL With DDL



```
CREATE TABLE Country (
  Code CHAR(3),
  Name CHAR(52),
  Continent CHAR(50),
  Population INT(11),
  Capital INT(11),
  PRIMARY KEY (Code)
);
```

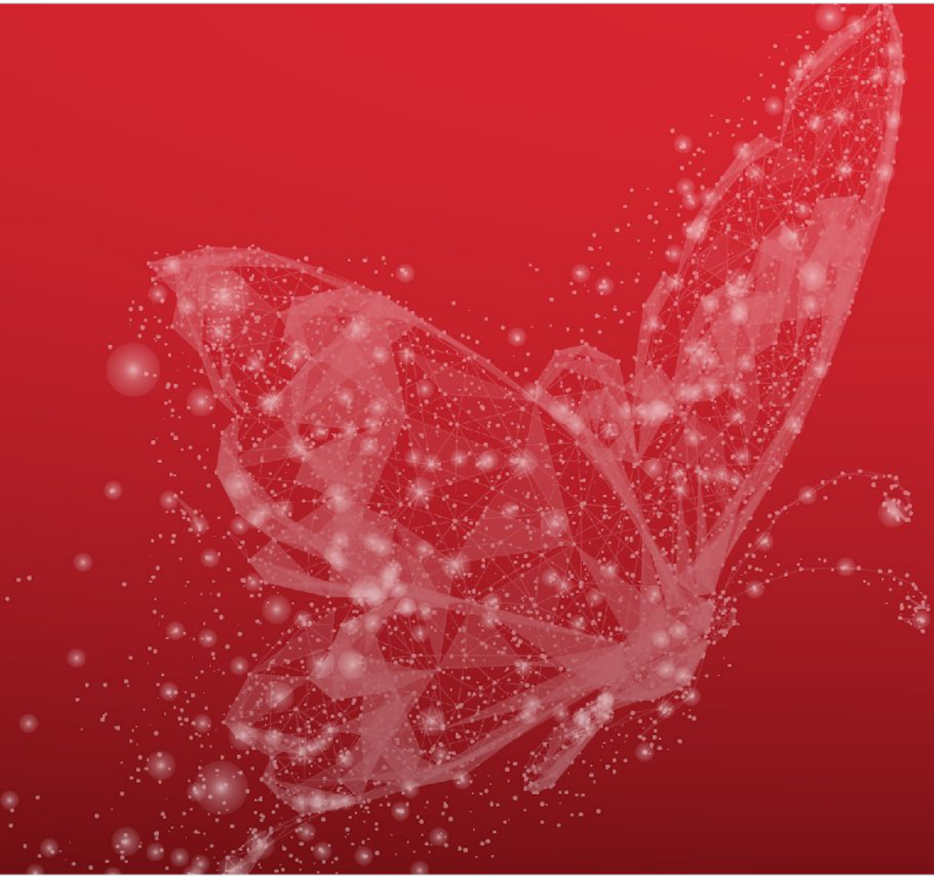
Usage Scenario:

- SQL-driven applications
- Green-field applications using Ignite as a database with its native persistence

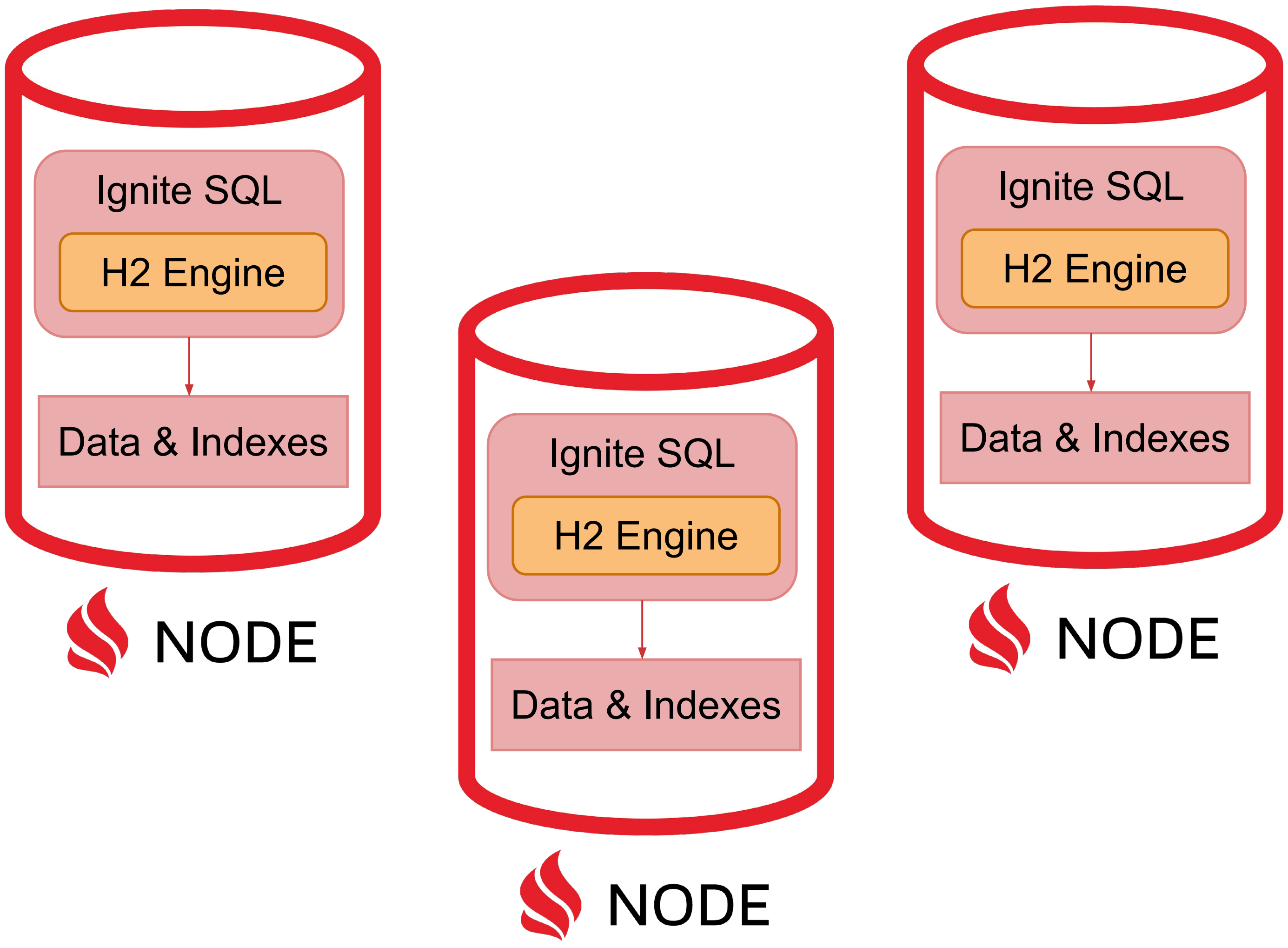
Demo Time

Cluster Startup and Database Creation

Affinity Co-Location and Distributed JOINs



Ignite SQL Engine Internals



GridGain

Query Execution Phases

```
SELECT AVG(population) FROM City
```

=

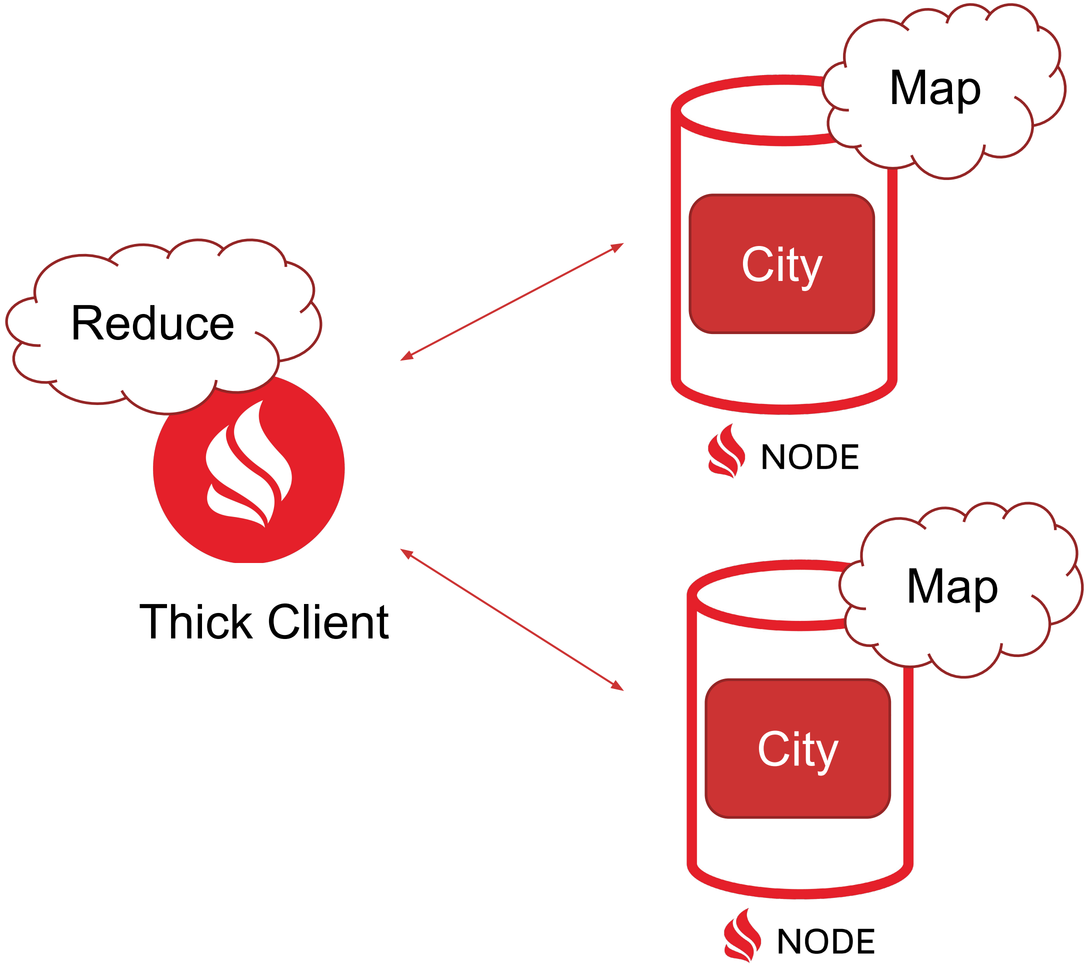
Map

```
SELECT SUM(population) as sum0,
COUNT(population) as count0
FROM City
```

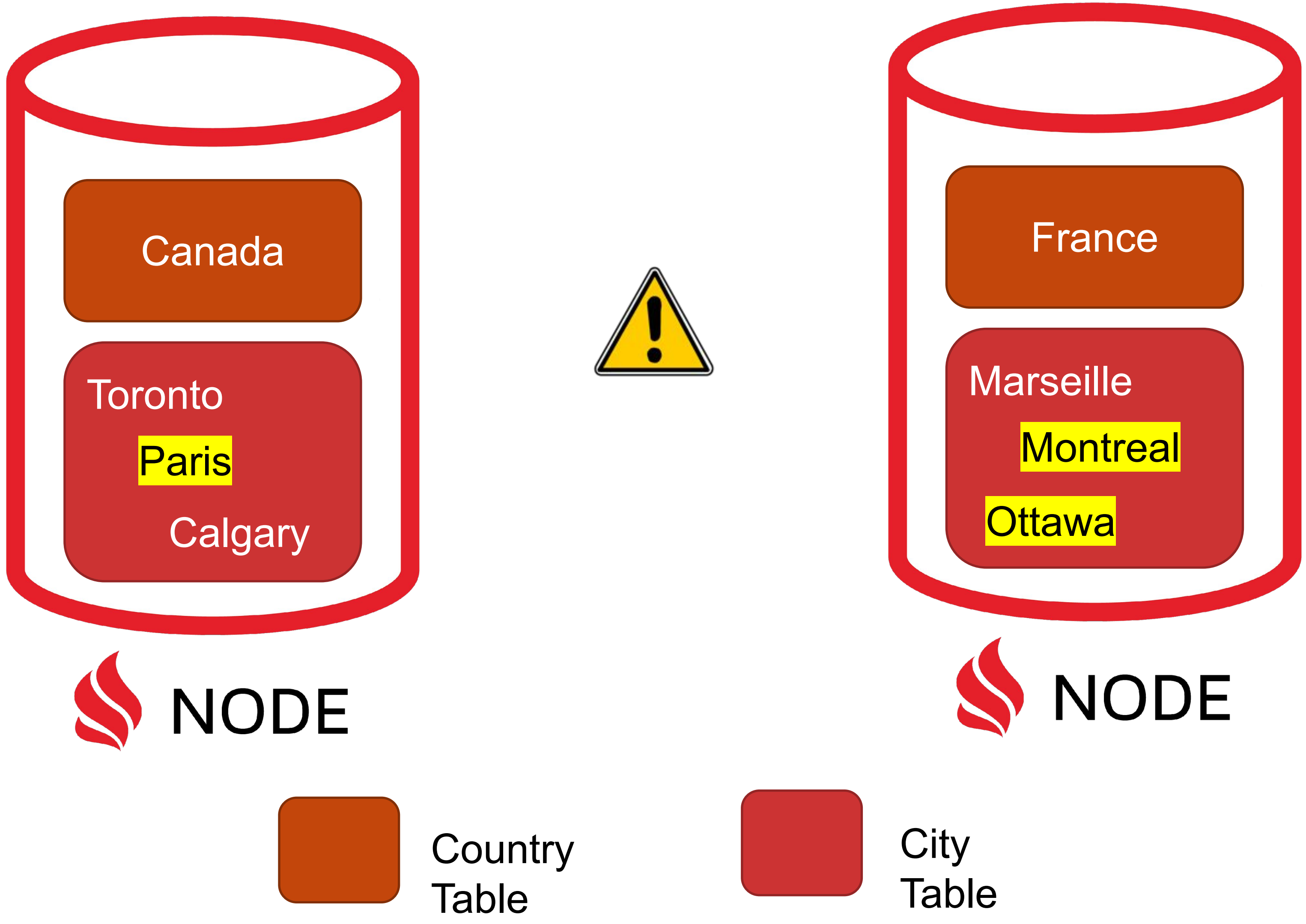
+

Reduce

```
SELECT SUM(sum0)/SUM(count0)
FROM resultTable
```



Default Data Distribution

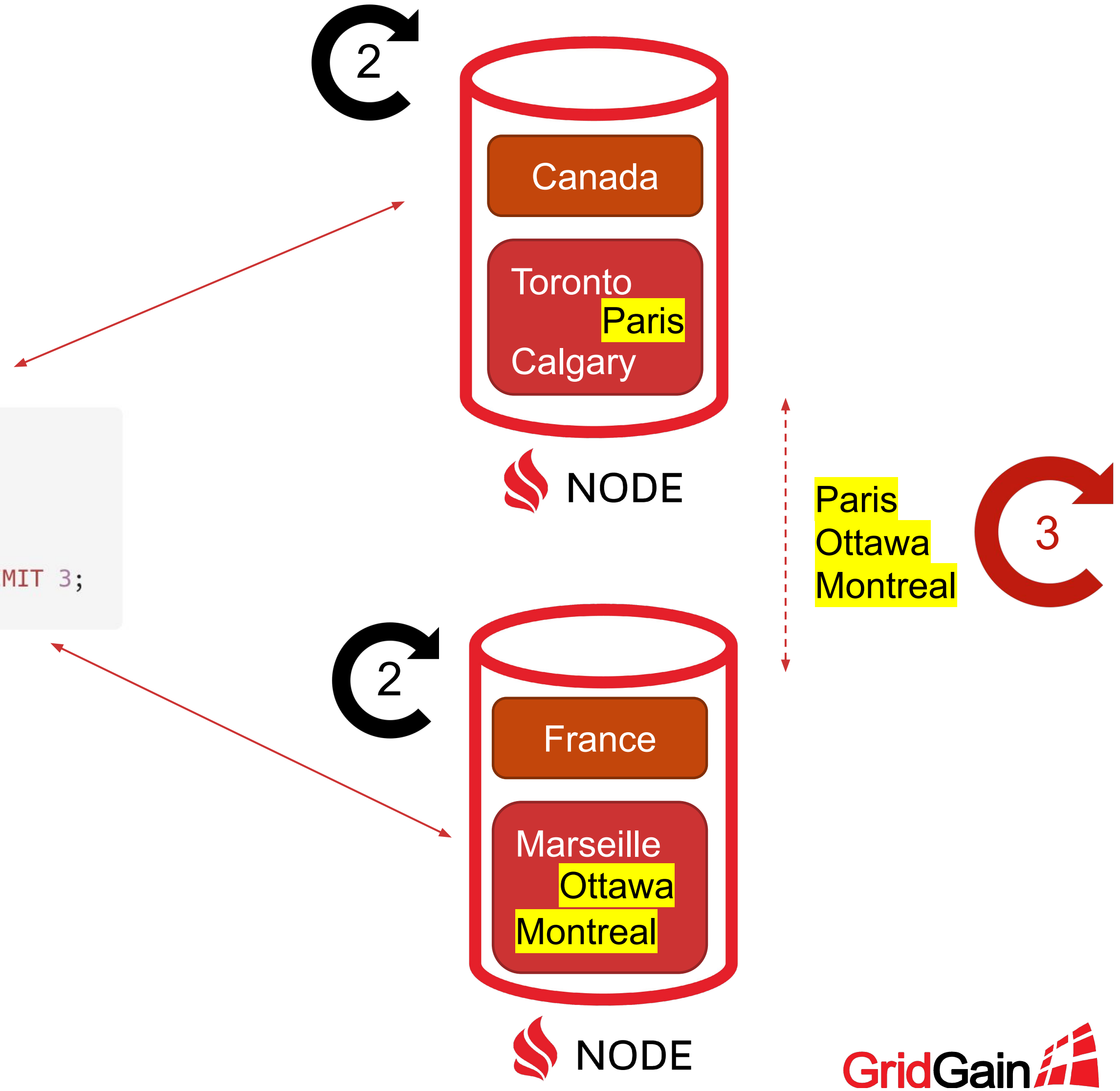


SQL JOIN With Data Shuffling

Thick Client

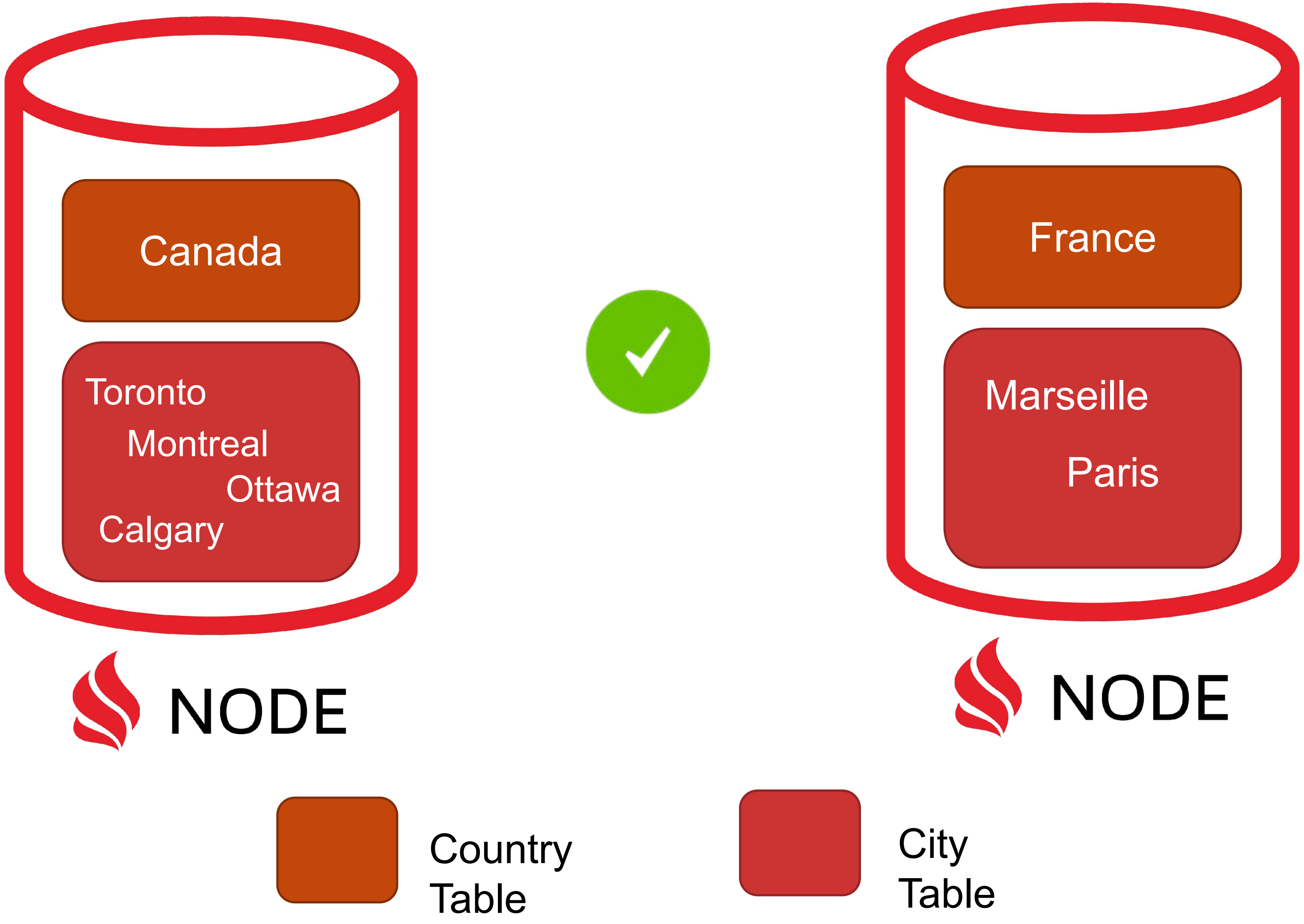
1 & 4

```
SELECT country.name, city.name,
MAX(city.population) as max_pop FROM country
JOIN city ON city.countrycode = country.code
WHERE country.code IN ('CAN','FRA')
GROUP BY country.name, city.name ORDER BY max_pop DESC LIMIT 3;
```



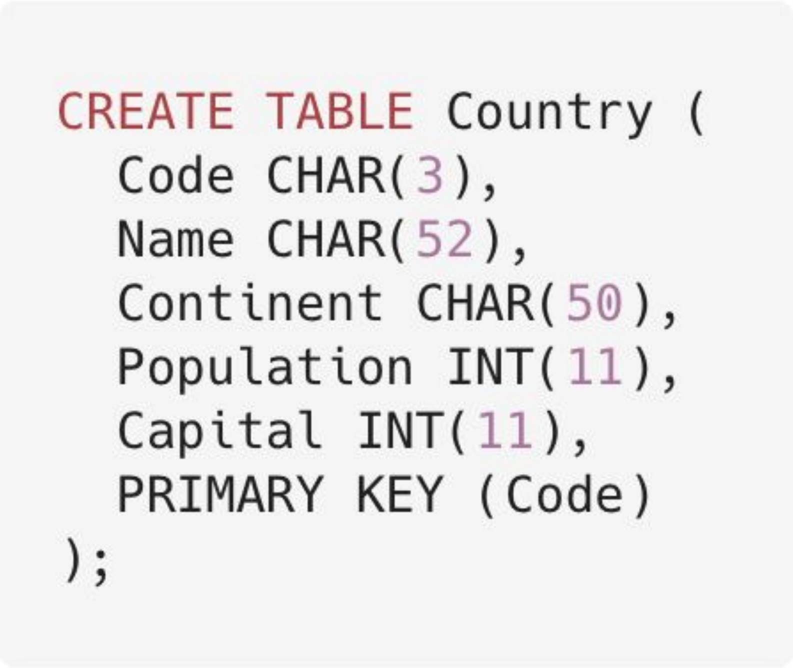
1. Initiating Execution
2. Execution on Servers (map phase)
3. Data Shuffling
4. Reduce Phase

Co-Located Distribution (aka. Affinity Co-Location)

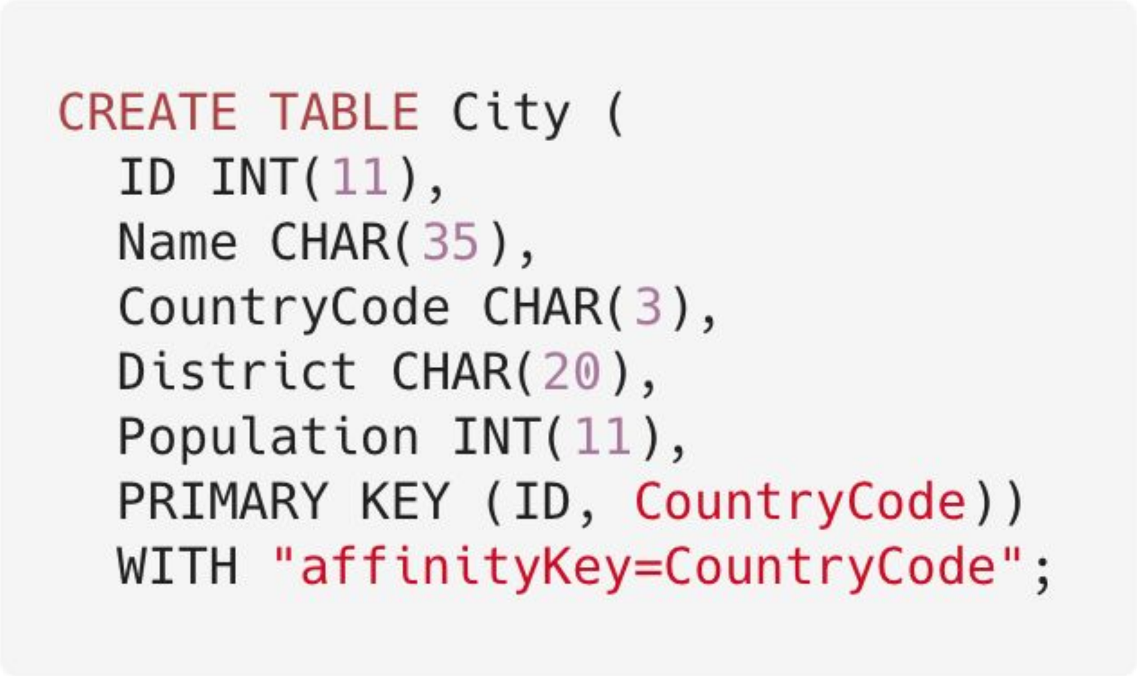


GridGain

All You Need is to Configure Affinity Key



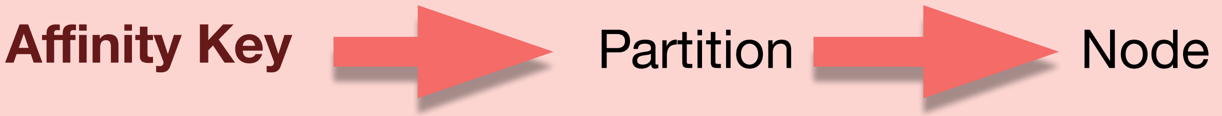
```sql
CREATE TABLE Country (
  Code CHAR(3),
  Name CHAR(52),
  Continent CHAR(50),
  Population INT(11),
  Capital INT(11),
  PRIMARY KEY (Code)
);
```

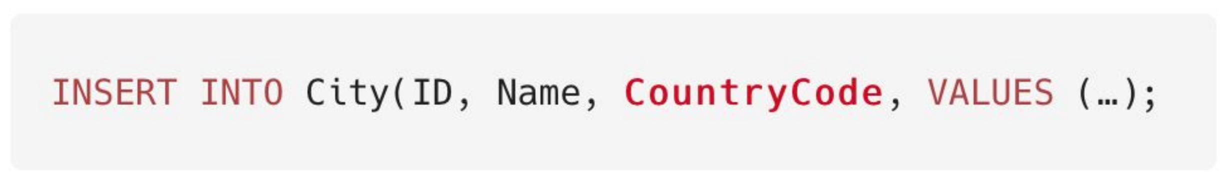


```sql
CREATE TABLE City (
  ID INT(11),
  Name CHAR(35),
  CountryCode CHAR(3),
  District CHAR(20),
  Population INT(11),
  PRIMARY KEY (ID, CountryCode))
  WITH "affinityKey=CountryCode";
```

Affinity Key to Node Mapping Process

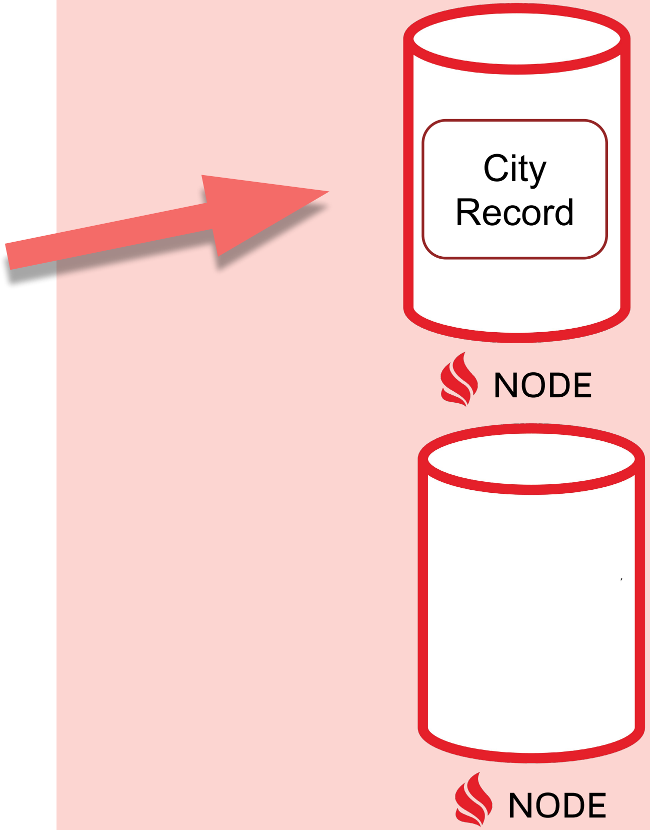
Application Process



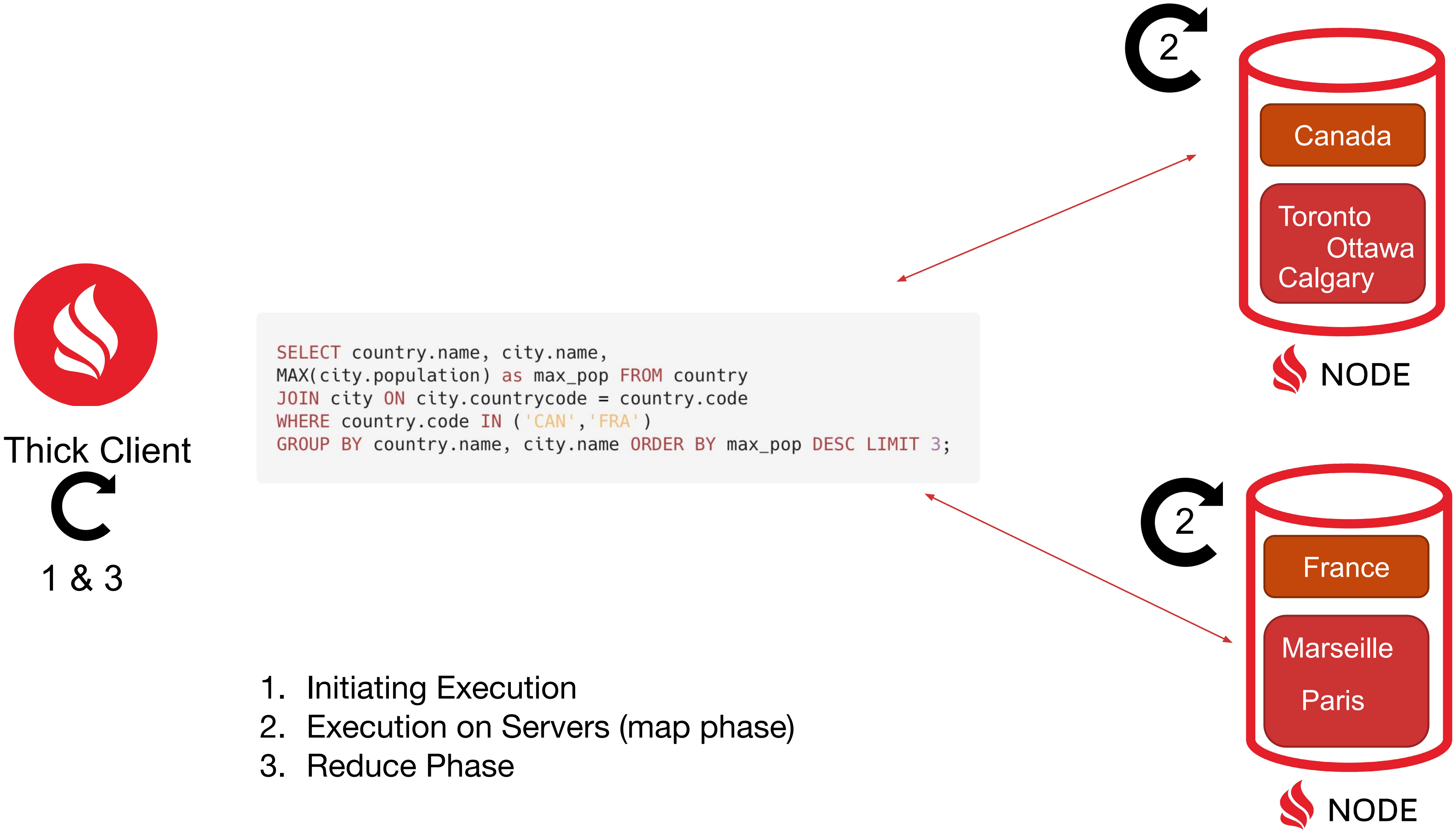


```
INSERT INTO City(ID, Name, CountryCode, VALUES (…);
```

Network Call



High-Performance SQL JOIN



```
SELECT country.name, city.name,
MAX(city.population) as max_pop FROM country
JOIN city ON city.countrycode = country.code
WHERE country.code IN ('CAN','FRA')
GROUP BY country.name, city.name ORDER BY max_pop DESC LIMIT 3;
```

1. Initiating Execution
2. Execution on Servers (map phase)
3. Reduce Phase

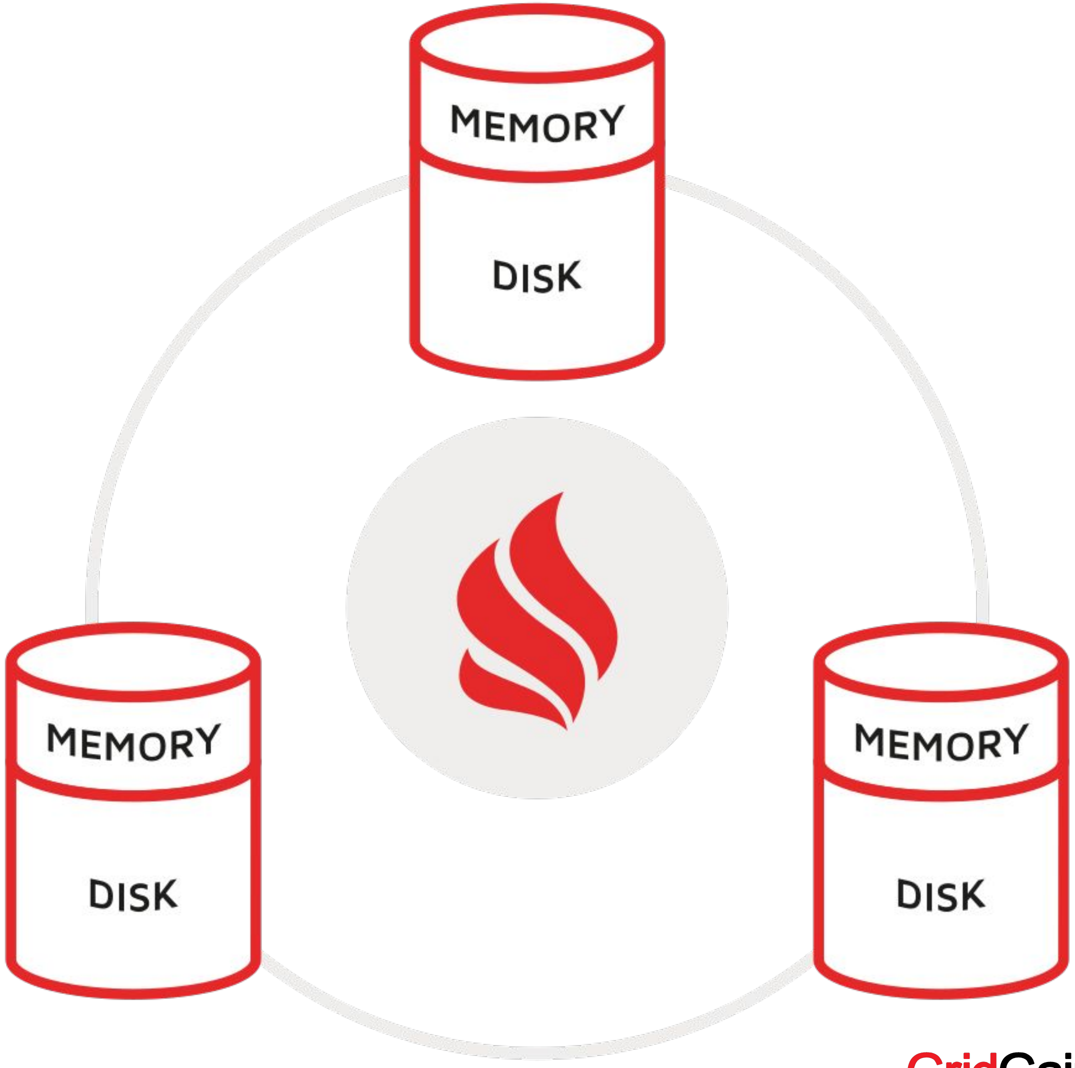
Demo Time

Queries With JOINs

Beyond Memory Capacity: Disk-Tier and Memory Quotas

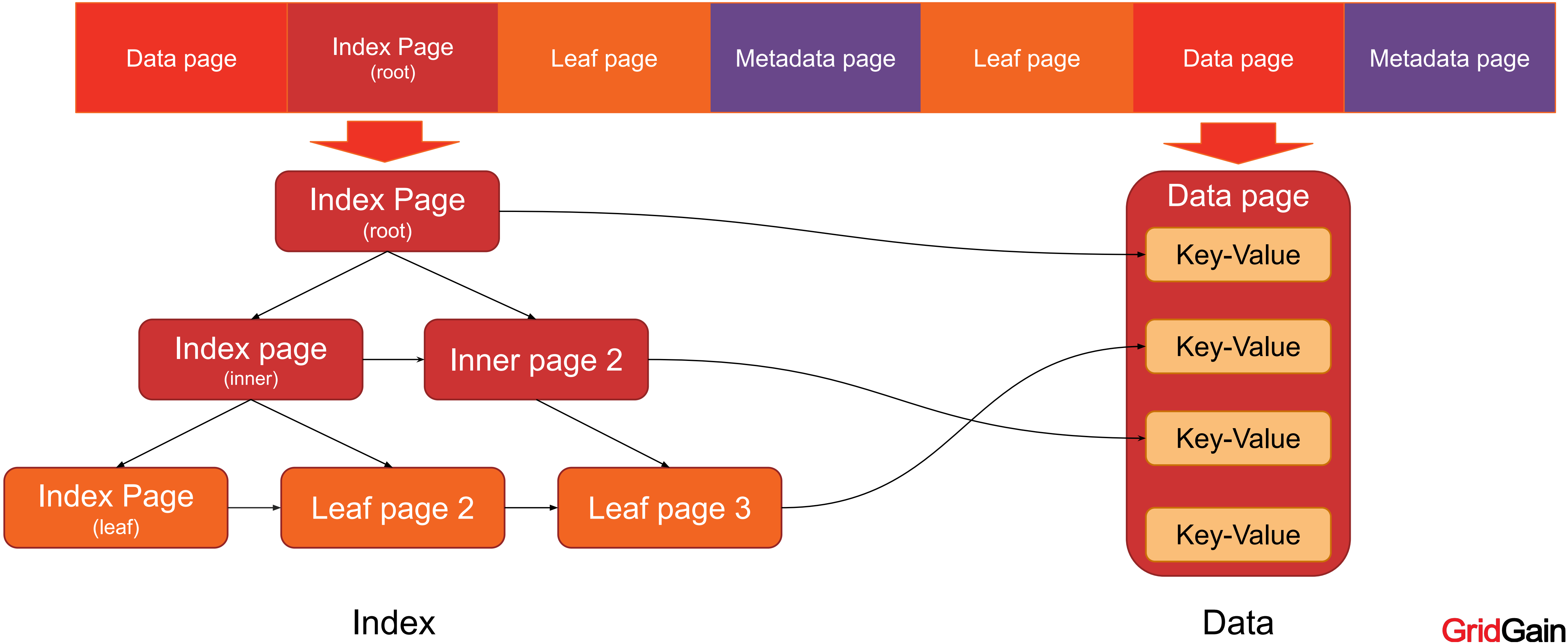
Multi-Tier Storage architecture

1. **In-Memory** - General in-memory caching, high-performance computing
2. **In-Memory + Native Persistence** - Ignite as an in-memory database
3. **In-Memory + External Database** - Acceleration of services and APIs with write-through and write-behind capability



Multi-Tier Storage Architecture

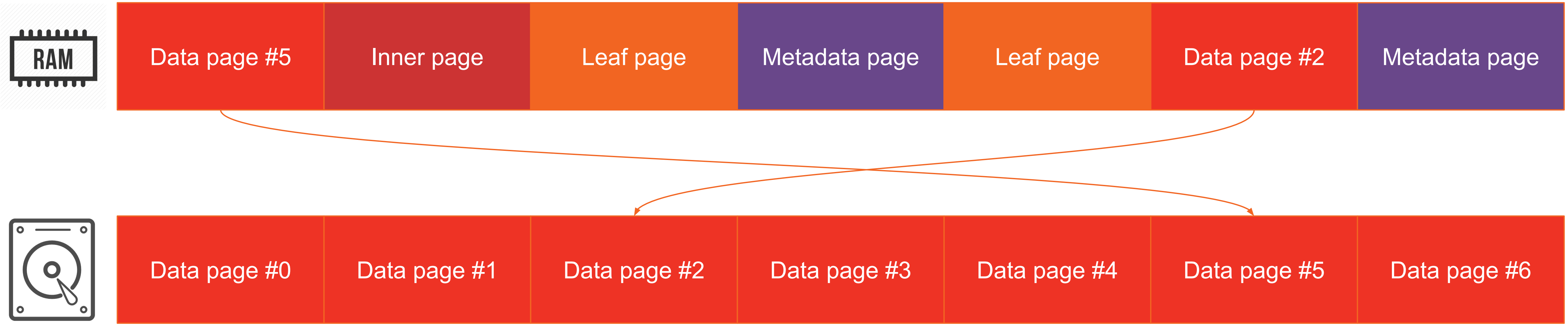
Memory segment



GridGain

Multi-Tier Storage Architecture

Memory segment



Data page #5 | Inner page | Leaf page | Metadata page | Leaf page | Data page #2 | Metadata page

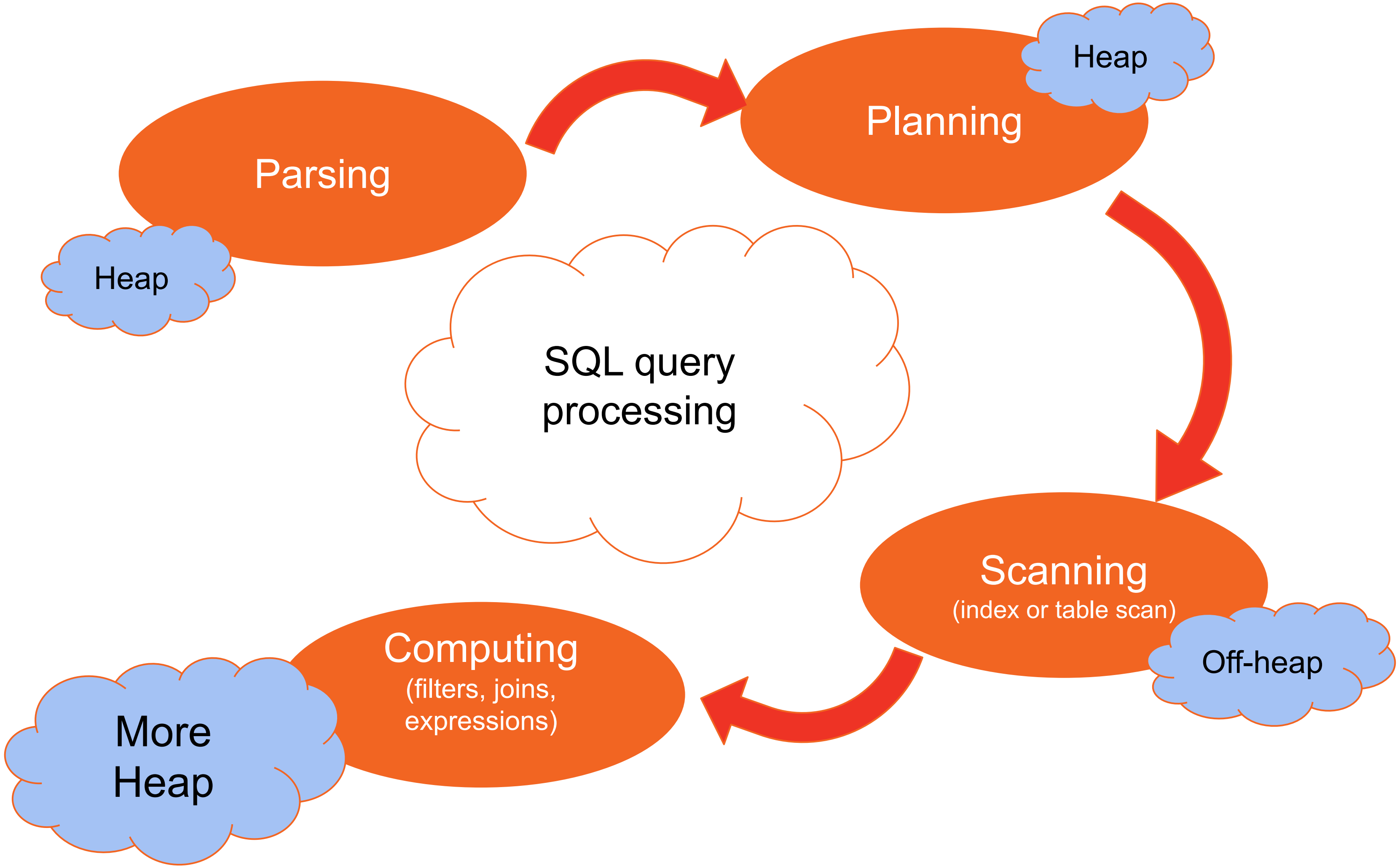
Data page #0 | Data page #1 | Data page #2 | Data page #3 | Data page #4 | Data page #5 | Data page #6

Partition file with Data

PageId → **Pages map** → **Pointer in a memory segment** (read/write ops)

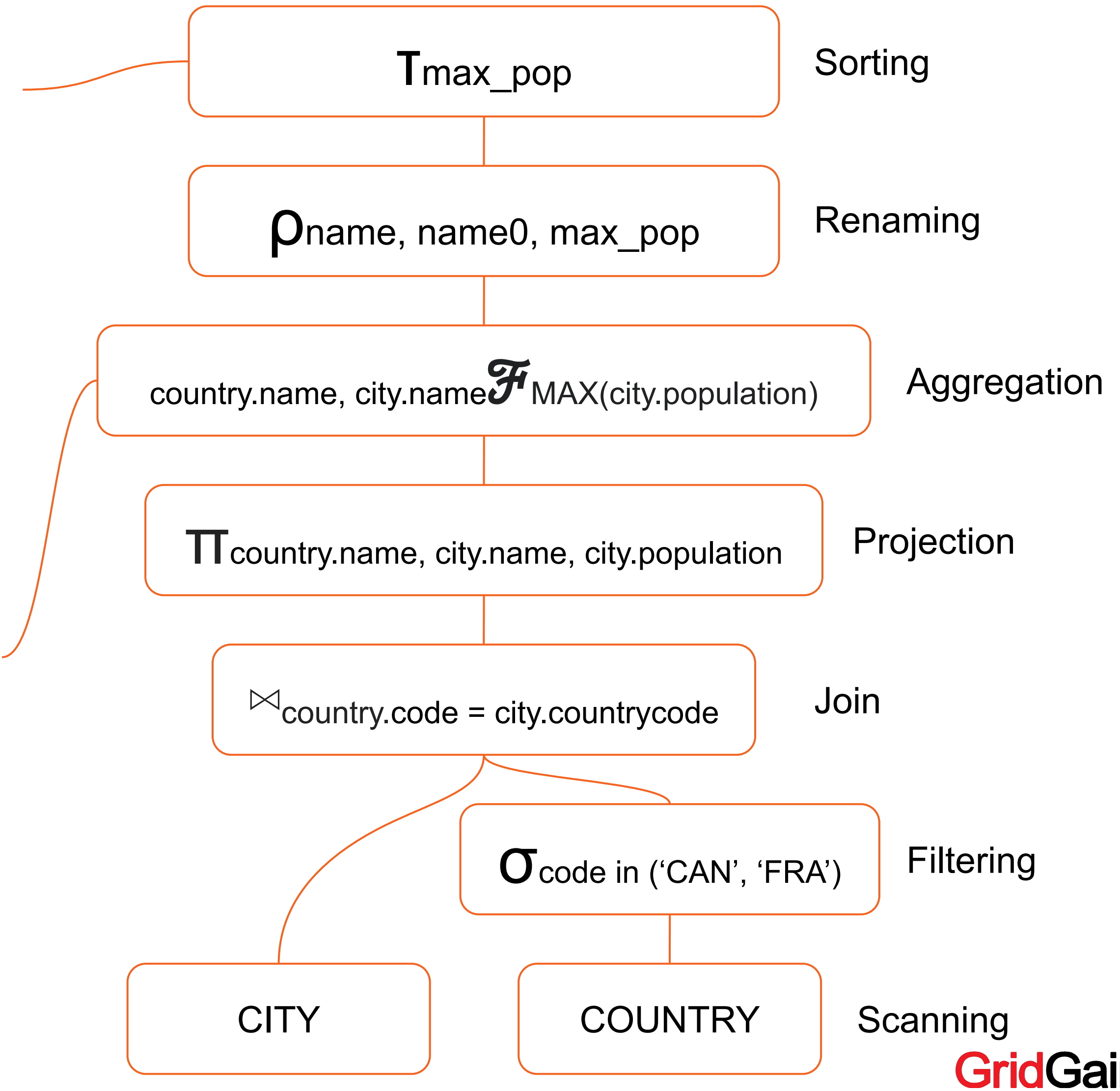
PageId → **Position in a file** (load page/checkpoint)

Java off-heap vs Java heap



Java off-heap vs Java heap

```sql
SELECT country.name, city.name,
MAX(city.population) as max_pop FROM country
JOIN city ON city.countrycode = country.code
WHERE country.code IN ('CAN','FRA')
GROUP BY country.name, city.name ORDER BY max_pop DESC LIMIT 3;
```

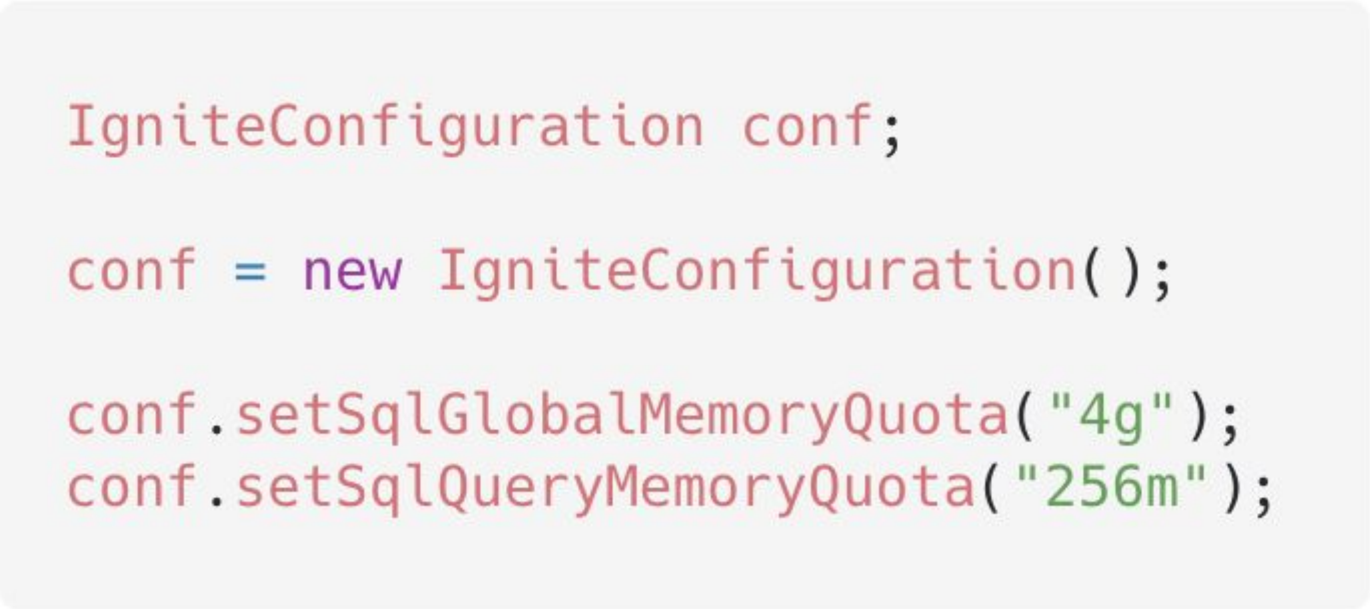


Here we need full set in heap

Here we need full set in heap too

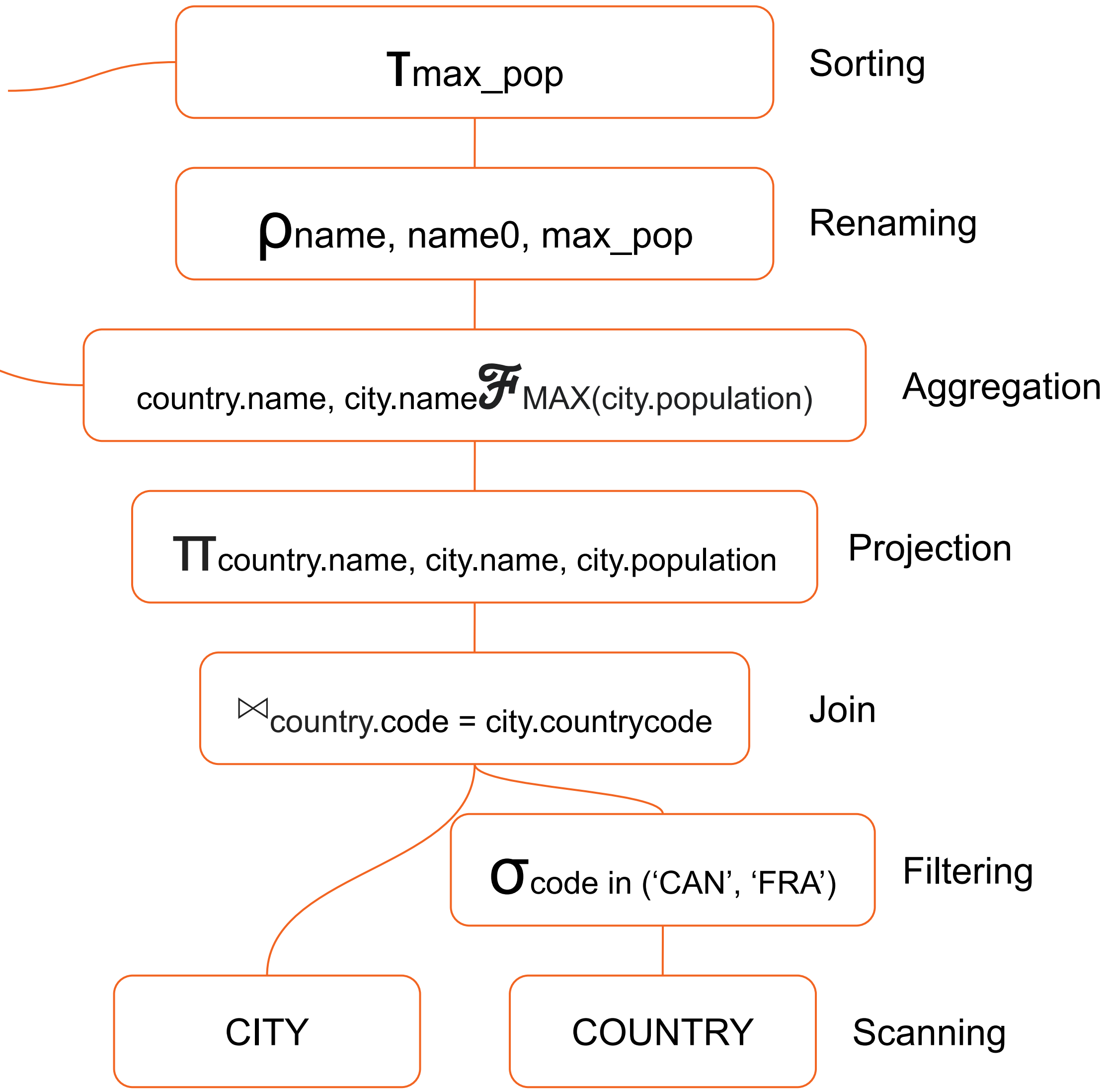
Query memory quotas

How to configure:



```
IgniteConfiguration conf;

conf = new IgniteConfiguration();

conf.setSqlGlobalMemoryQuota("4g");
conf.setSqlQueryMemoryQuota("256m");
```

Interim results offloading



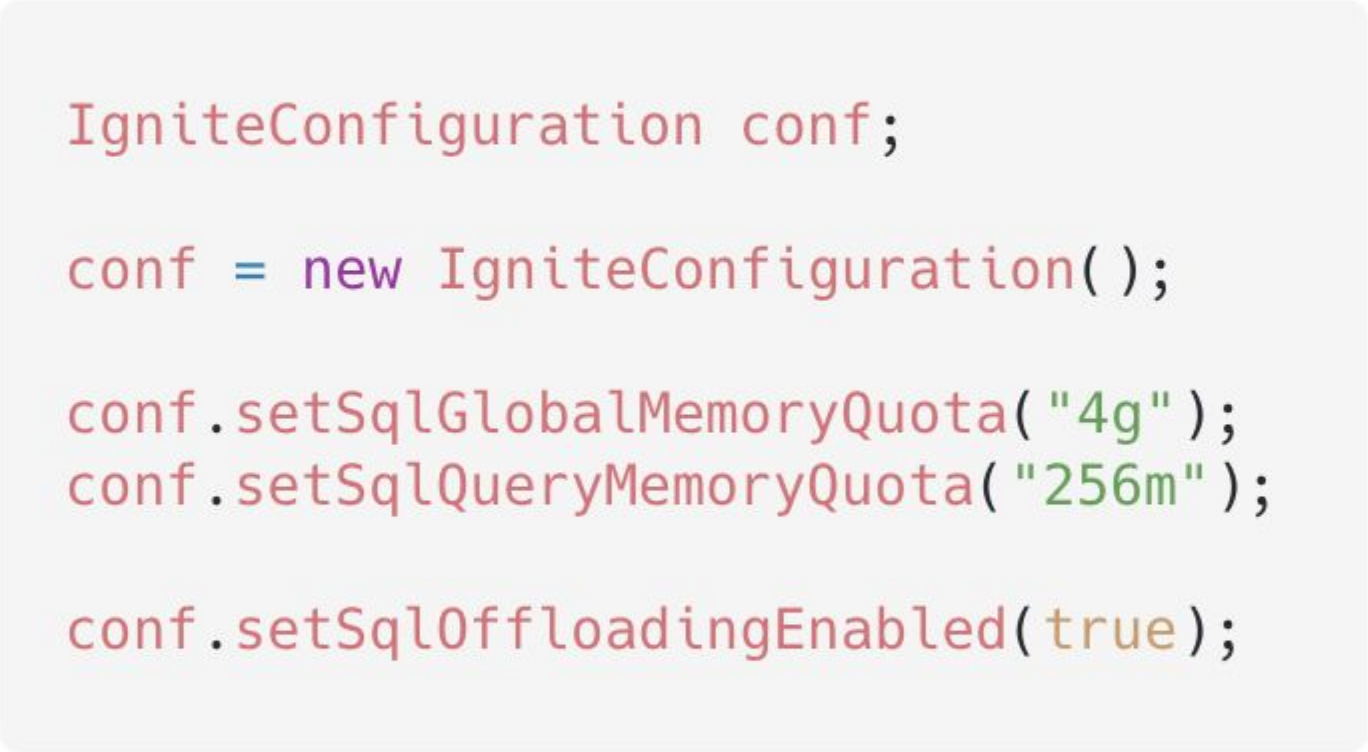
Why don't you flush result sets to disk?

And it

GridGain

Intermediate results offloading

How to configure:



```
IgniteConfiguration conf;

conf = new IgniteConfiguration();

conf.setSqlGlobalMemoryQuota("4g");
conf.setSqlQueryMemoryQuota("256m");

conf.setSqlOffloadingEnabled(true);
```

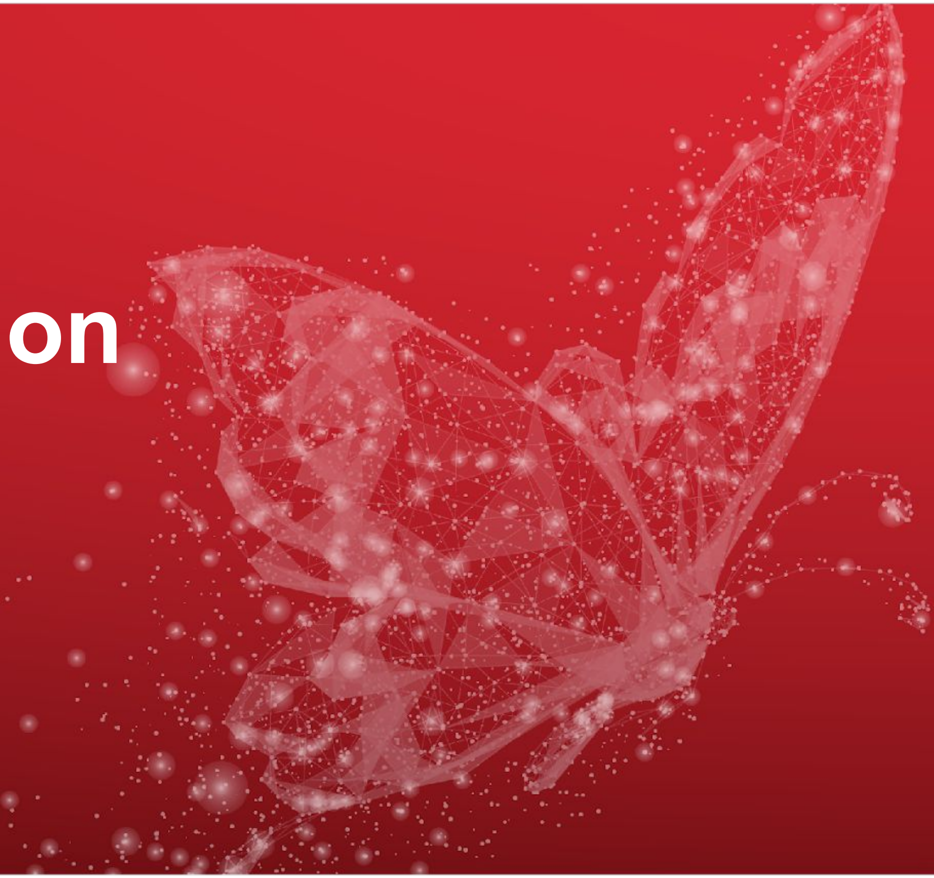
When you need quotas/offloading enabled

- Sorting (ORDER BY)
- Grouping (DISTINCT, GROUP BY)
- Complex subqueries

Demo Time

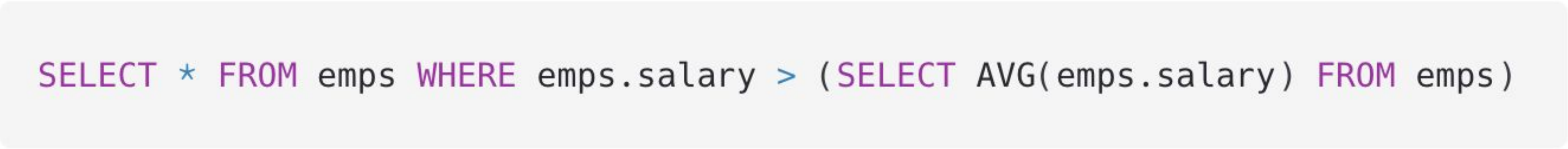
Running SQL Over Disk-Only Records

Apache Ignite SQL Evolution With Apache Calcite



Why do we need it?

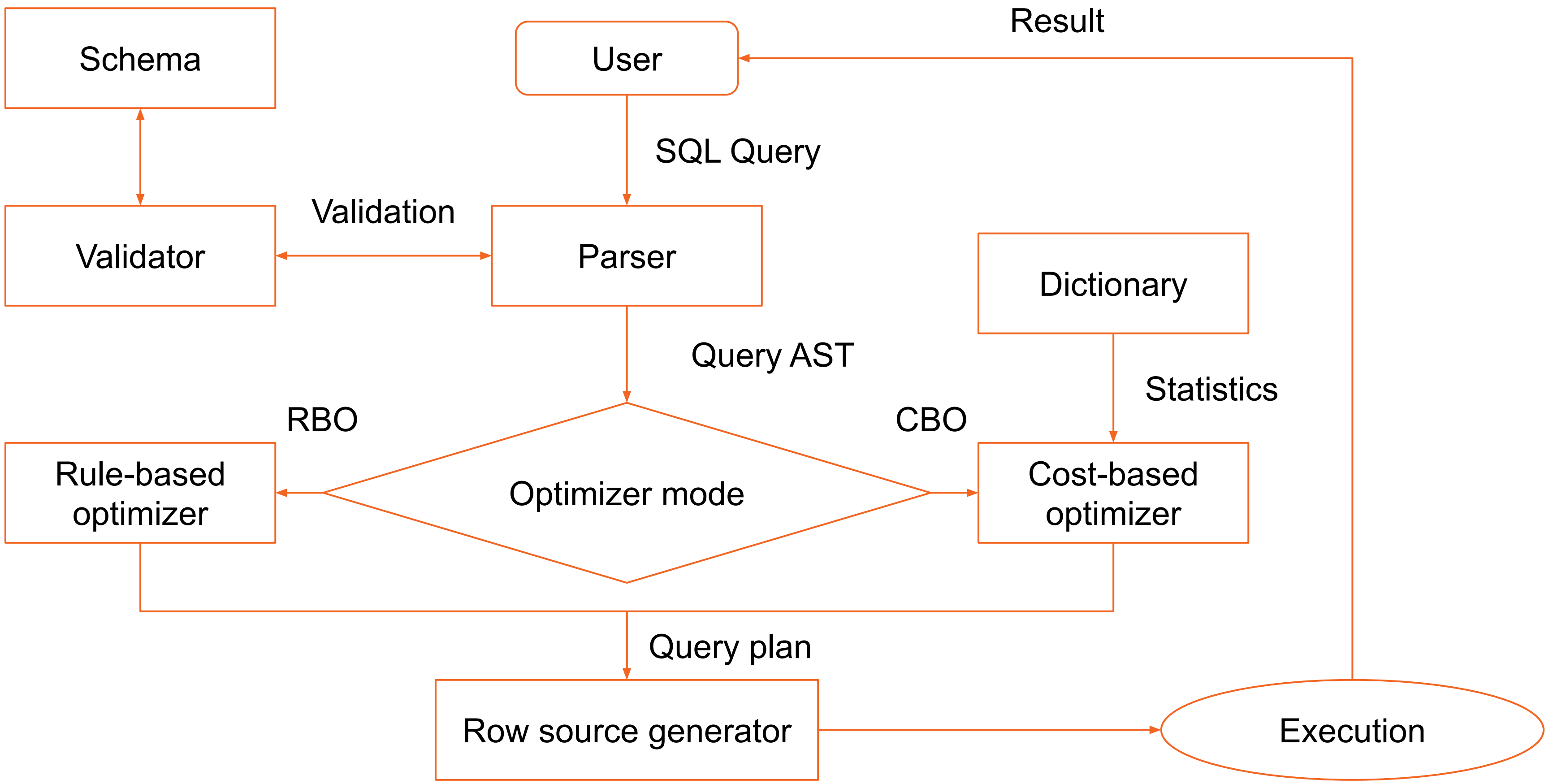
Here we need Map-Reduce phase too



```
SELECT * FROM emps WHERE emps.salary > (SELECT AVG(emps.salary) FROM emps)
```

Here we need Map-Reduce phase

Typical execution flow

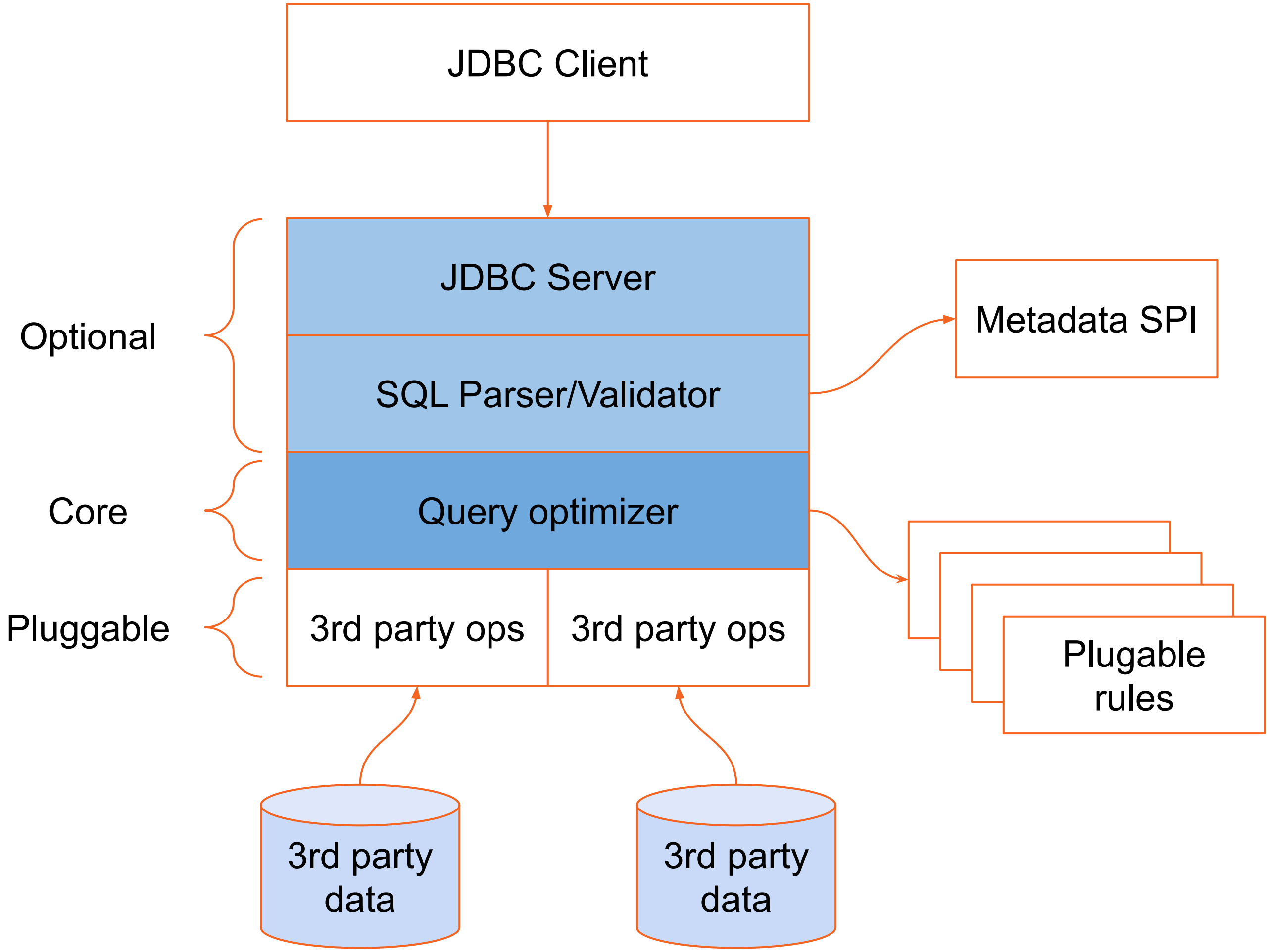


Apache Calcite

APACHE calcite™

Need to implement:

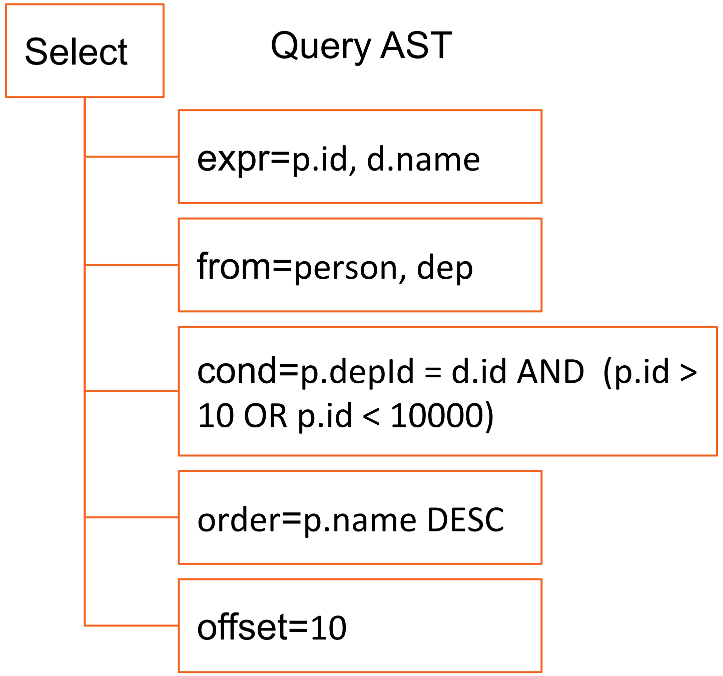
- Splitter
- Runtime
- Indexes support
- DML support
- DDL support



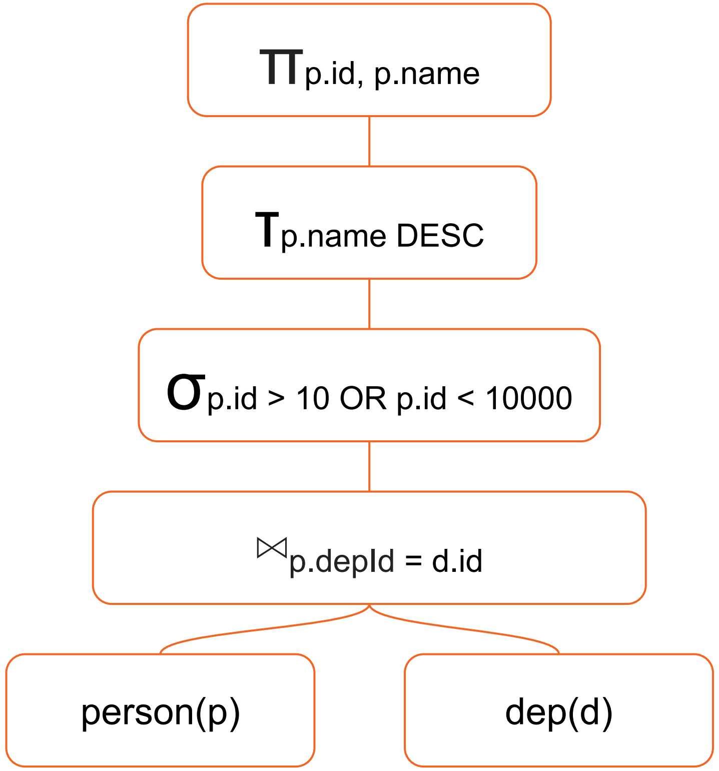
GridGain

Query Parser and Transformer

```
SELECT p.id, d.name
FROM person p, dep d
WHERE p.depId = d.id AND  (p.id > 10 OR p.id < 10000)
ORDER BY p.name DESC
```

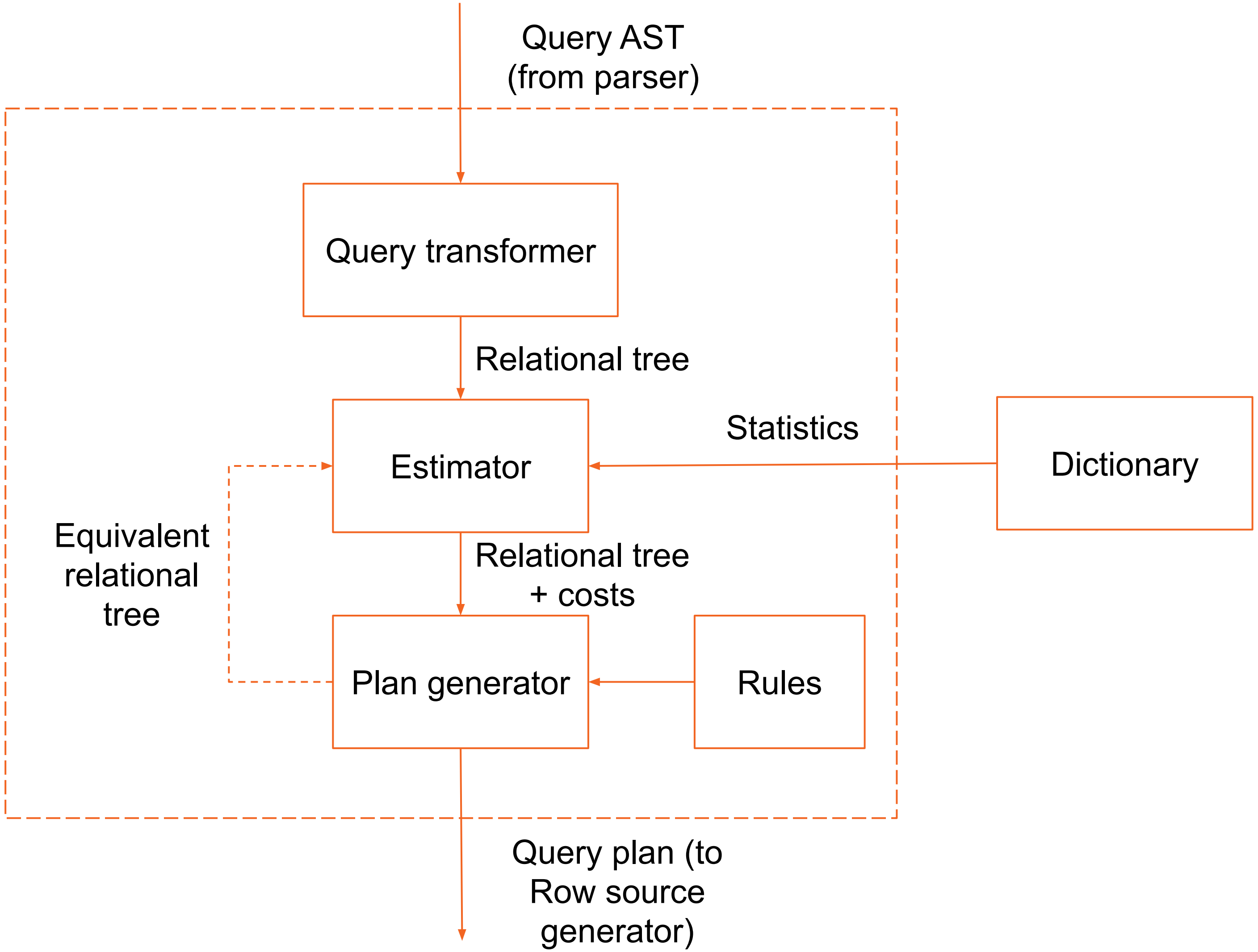


Relational operators tree
(query plan)

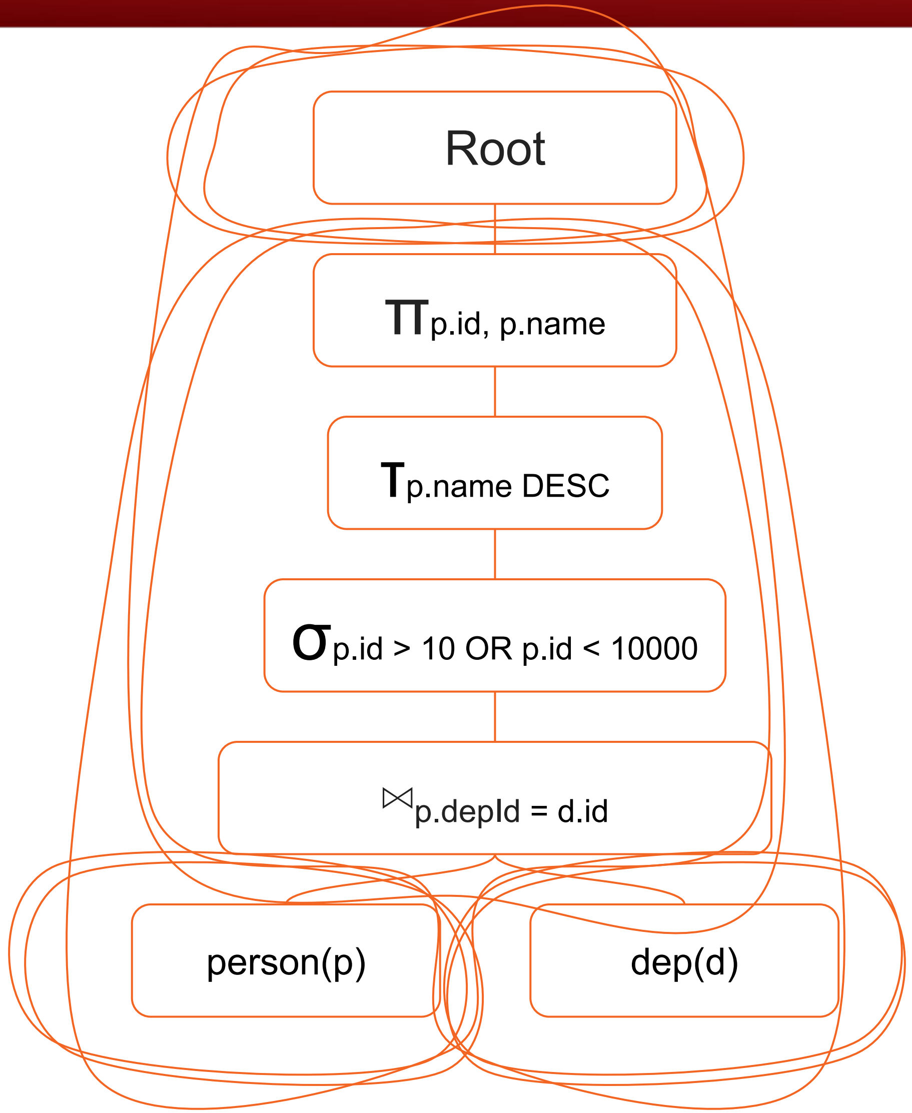


$\pi_{p.id,\ p.name}$
$\tau_{p.name\ DESC}$
$\sigma_{p.id\ >\ 10\ OR\ p.id\ <\ 10000}$
$\bowtie_{p.depId\ =\ d.id}$
person(p)
dep(d)

Cost-Based Optimizer



Cost-Based Splitter



Root

$\Pi_{p.id,\ p.name}$

$\tau_{p.name\ DESC}$

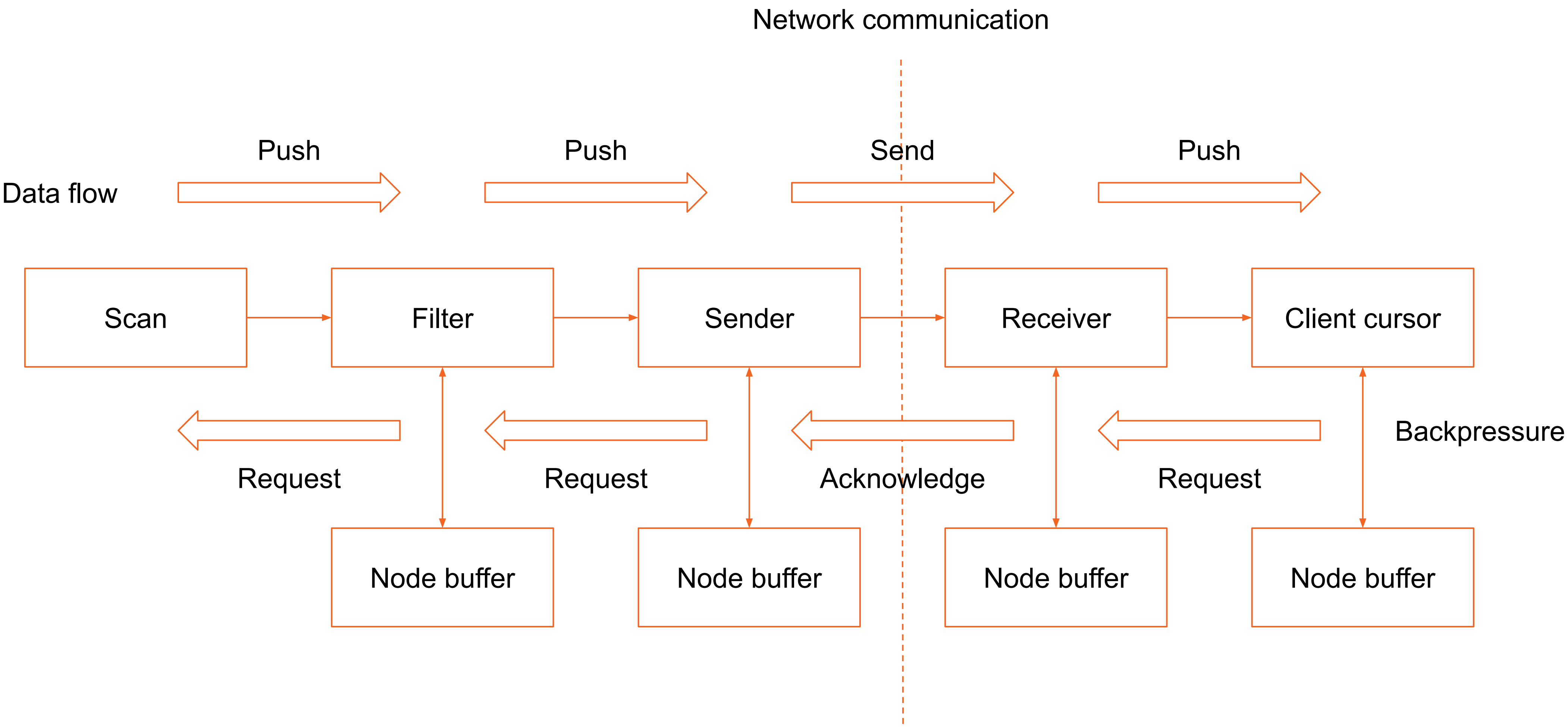
$\sigma_{p.id\ >\ 10\ OR\ p.id\ <\ 10000}$

$\bowtie_{p.depId\ =\ d.id}$

person(p)

dep(d)

Reactive Execution Flow



GridGain

Demo Time

Calcite Prototype Demo With Sub-Queries

Learn More

- Apache Ignite SQL
  - https://apacheignite-sql.readme.io/docs
- Memory Quotas (available in GridGain Community Edition):
  - https://www.gridgain.com/docs/latest/developers-guide/memory-configuration/memory-quotas
- Demos shown in this webinar
  - https://github.com/GridGain-Demos/ignite-sql-intro-samples
- New Apache Calcite-based engine
  - https://cwiki.apache.org/confluence/display/IGNITE/IEP-37%3A+New+query+execution+engine