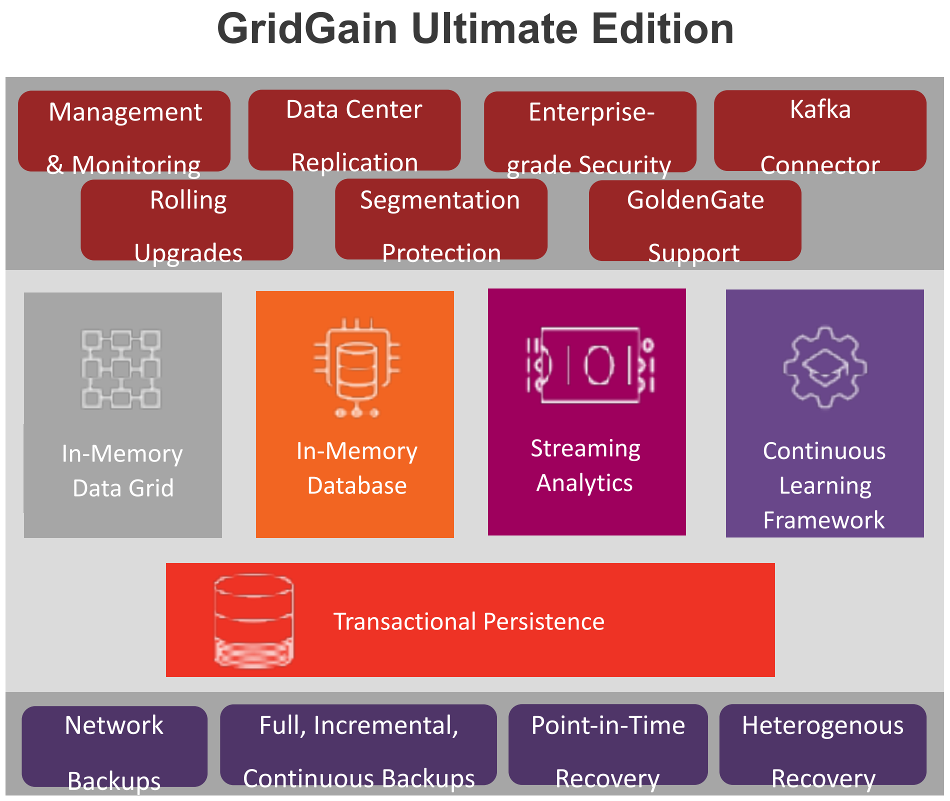
How to Add Speed and Scale to SQL Support New Data Needs and Keep Your RDBMS

Valentin Kulichenko

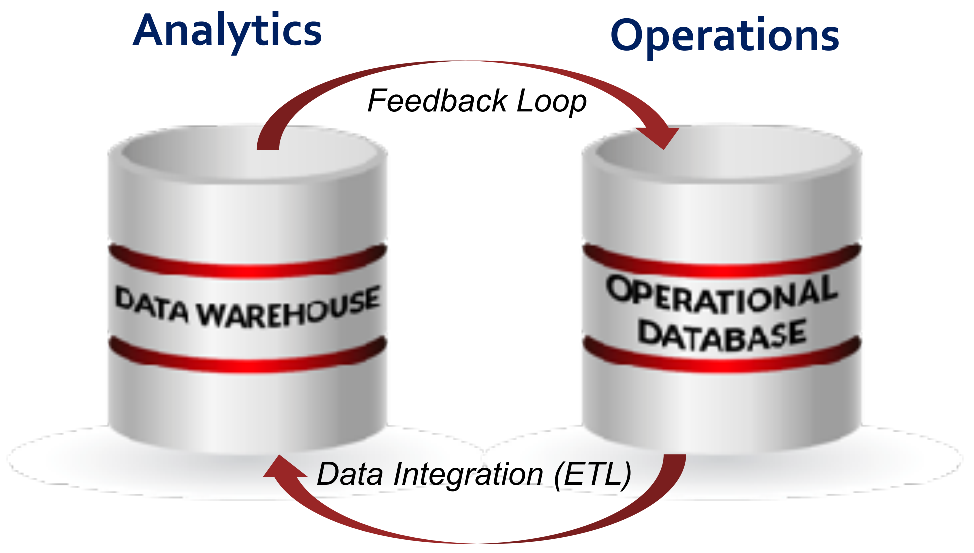
GridGain In-Memory Computing Platform

- Built on Apache Ignite
  - Comprehensive platform that supports all projects
  - No rip and replace
  - In-memory **speed**, petabyte **scale**
  - Enables HTAP, streaming analytics and continuous learning
- What GridGain adds
  - Production-ready releases
  - Enterprise-grade integration, security, deployment and management
  - Global support and services
  - Proven for mission critical apps

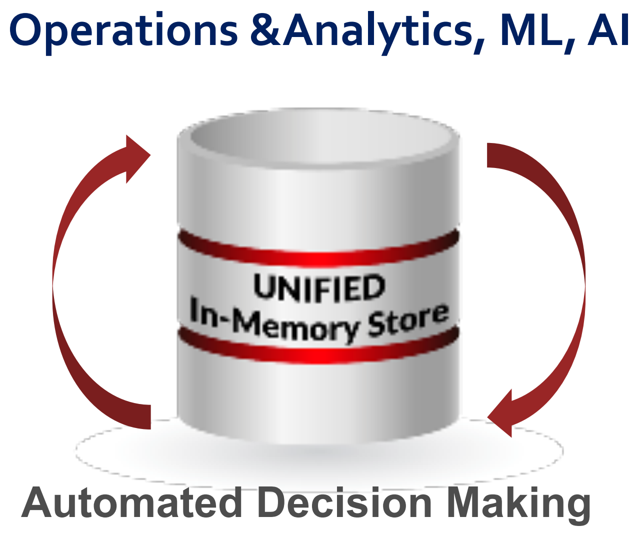


Traditional ETL vs GridGain HTAP Architecture

Traditional ETL Architecture



Unified HTAP Architecture

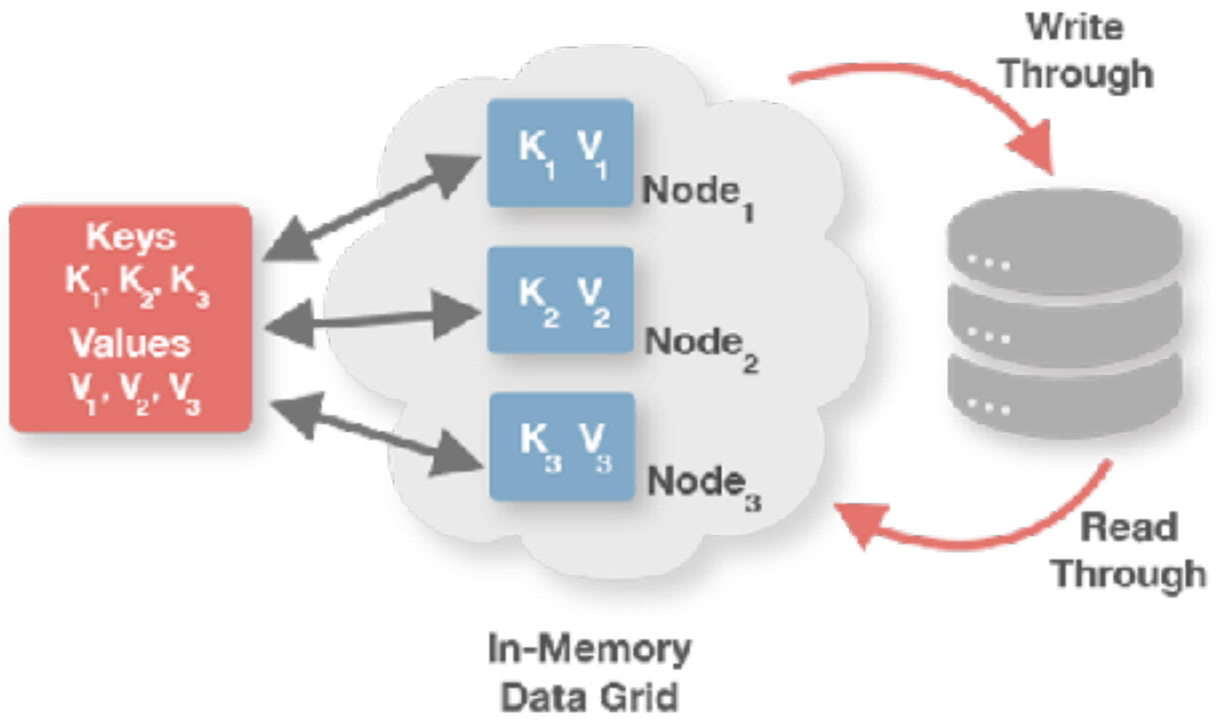


Memory-Centric Architecture Advantages

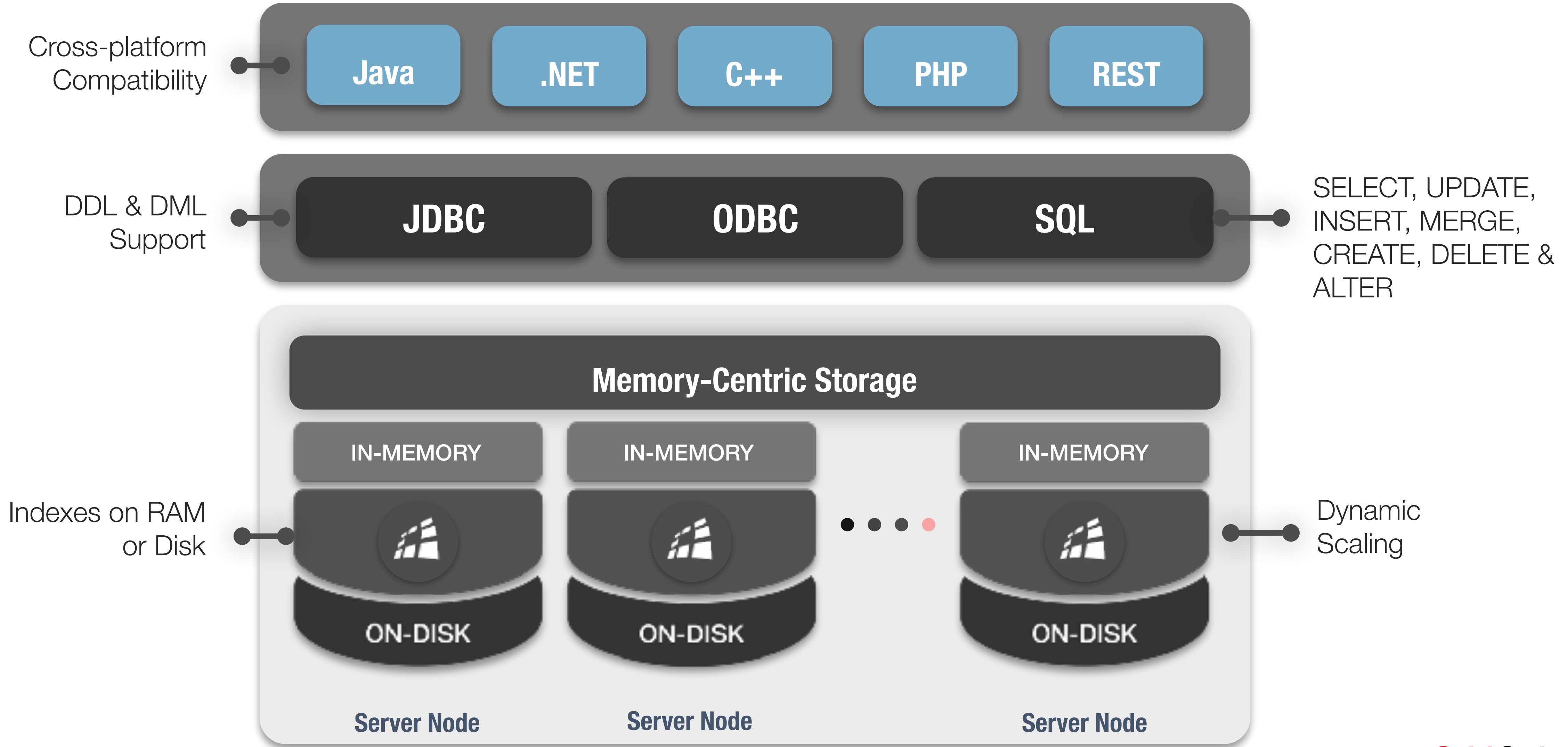
Mode	Description	Major Advantage
In-Memory	100% data in the In-Memory Store (only)	Maximum performance possible (data is never written to disk)
In-Memory + 3rd Party DB	Data in the In-Memory Data Store as a caching layer (aka. in-memory data grid) 3rd Party DB (RDBMS, NoSQL, etc) used for persistence	Horizontal scalability Faster reads and writes
In-Memory + Persistent Store	The whole data set is stored both in memory and on disk	Survives cluster failures
100% on Disk + In-Memory Cache	100% of data is in GridGain Persistent Store and a subset is in memory	Unlimited data scale beyond RAM capacity

Turbocharging Database Systems

- Database Caching Use Case
- Slide Ignite in between Database System and applications
- No 'rip and replace' Performance Boost
- Keep data both in memory and Database System
- Scale to 1000s of nodes
- Automatic Read-Through and Write-Through
- Key-Value Operations Only
- ANSI-99 SQL
- Over in-memory data sets



Distributed SQL



Connectivity

- JDBC
- ODBC
- REST
- Java, .NET and C++ APIs

```
// Register JDBC driver.
Class.forName("org.apache.ignite.IgniteJdbcThinDriver");

// Open the JDBC connection.
Connection conn = DriverManager.getConnection("jdbc:ignite:thin://192.168.0.50");
```

```
./sqlline.sh --color=true --verbose=true -u jdbc:ignite:thin://127.0.0.1/
```

Data Definition Language

- CREATE/DROP TABLE
- CREATE/DROP INDEX
- ALTER TABLE
- Changes Durability
    - Ignite Native Persistence

https://apacheignite-sql.readme.io/docs/ddl

```
CREATE TABLE `city` (
  `ID` INT(11),
  `Name` CHAR(35),
  `CountryCode` CHAR(3),
  `District` CHAR(20),
  `Population` INT(11),
  PRIMARY KEY (`ID`, `CountryCode`)
) WITH "template=partitioned, backups=1, affinityKey=CountryCode";
```

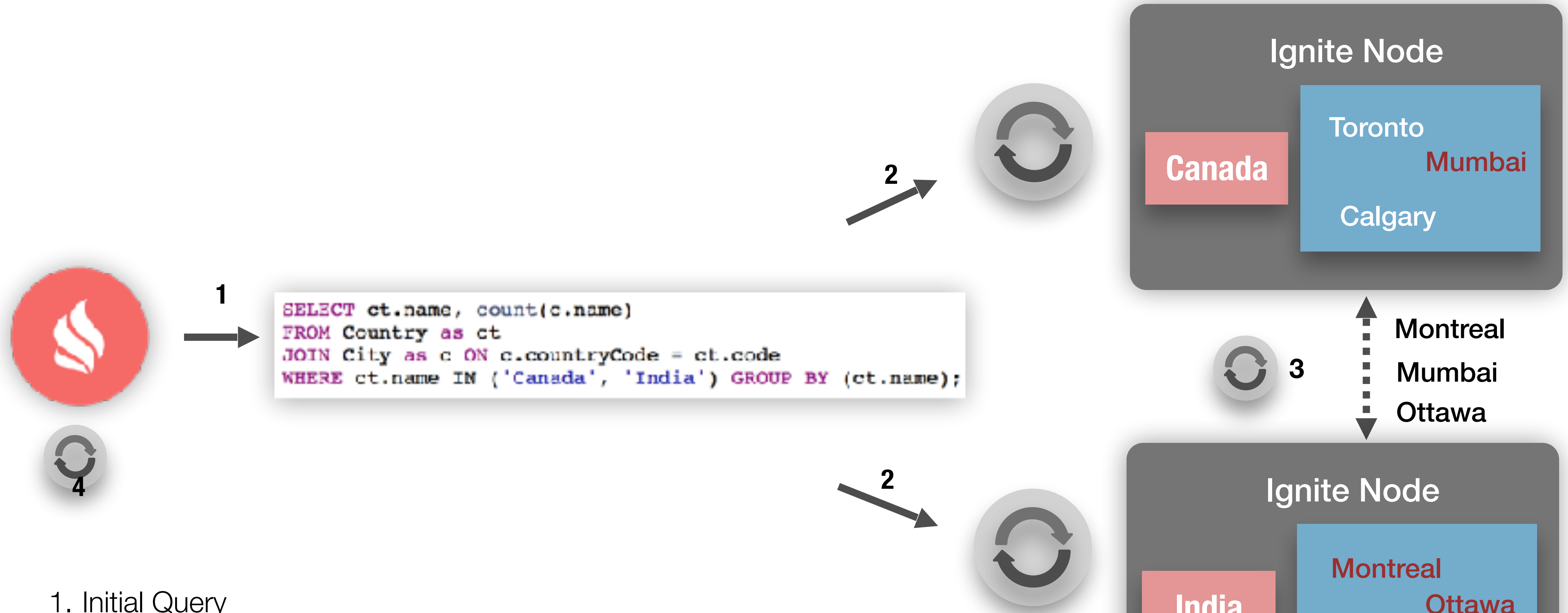
Data Manipulation Language

- ANSI-99 specification
- Fault-tolerant and consistent
- INSERT, UPDATE, DELETE
- SELECT
    - JOINs
    - Subqueries

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

https://apacheignite-sql.readme.io/docs/dml

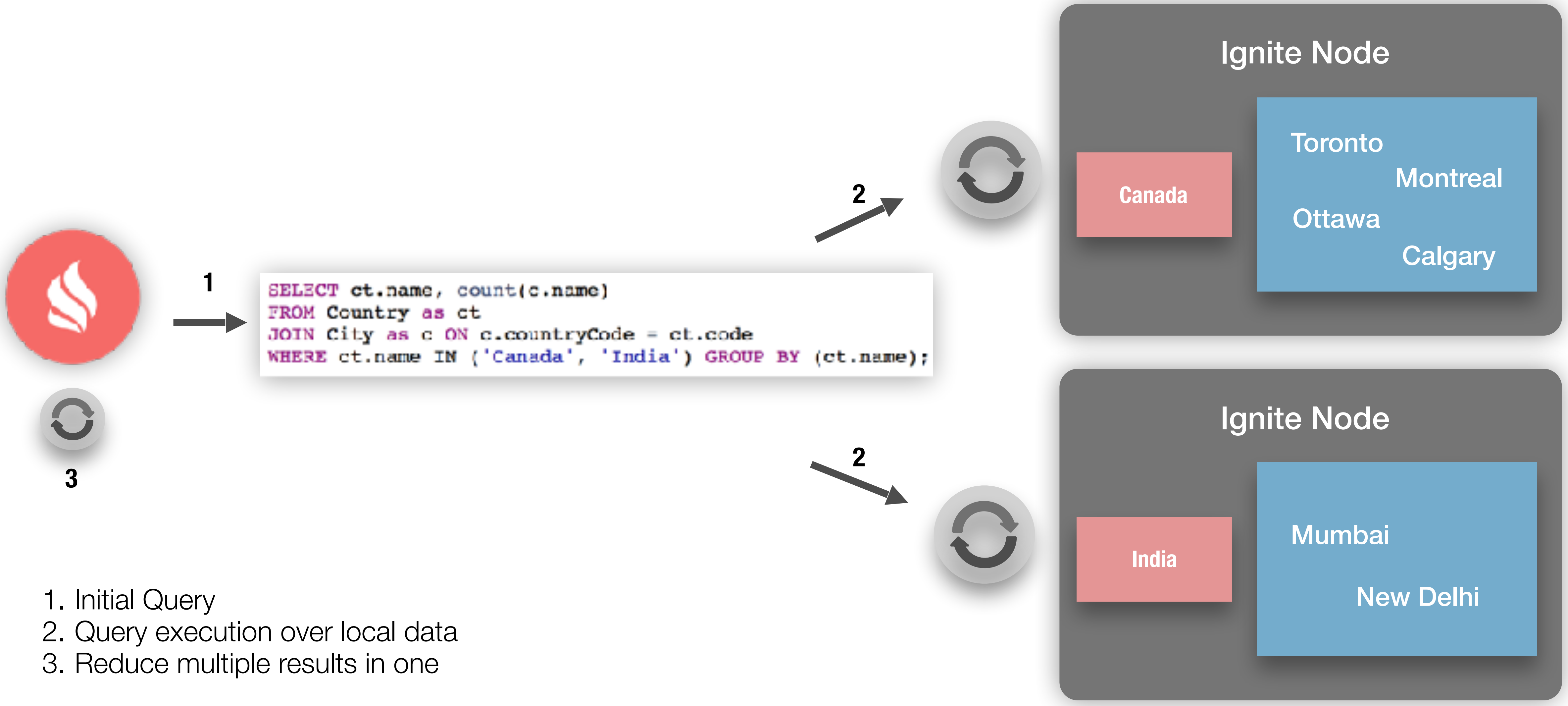
Non-Collocated Joins



```
SELECT ct.name, count(c.name)
FROM Country as ct
JOIN City as c ON c.countryCode = ct.code
WHERE ct.name IN ('Canada', 'India') GROUP BY (ct.name);
```

1. Initial Query
2. Query execution (local + remote data)
3. Potential data movement
4. Reduce multiple results in one

Collocated Joins



```
SELECT ct.name, count(c.name)
FROM Country as ct
JOIN City as c ON c.countryCode = ct.code
WHERE ct.name IN ('Canada', 'India') GROUP BY (ct.name);
```

1. Initial Query
2. Query execution over local data
3. Reduce multiple results in one

Demo

Questions?