Group 15

Hamza Karachiwala
Himanshu Pandey
Jayesh Yadav
Nidhi Makhijani

Shortcomings in relational databases

- Schema - Lacks flexibility
- Set based - Relationships need extracting
- Physical Storage - Hard to partition
- For large data volumes, shortcomings are magnified
- All comes down to performance

NoSQL models

- Graph models – Relationships – Neo4j
- Document models – Flexibility – MongoDB
- Key–Value models – Simplistic retrieval – Memcached
- Object Oriented Models – Inheritance/Polymorphism – Ontos

OrientDB and the Multi-model approach

- One-stop-shop
- Graph model for faster relationship extraction
- Document model for flexibility
- Aimed to solve all problems

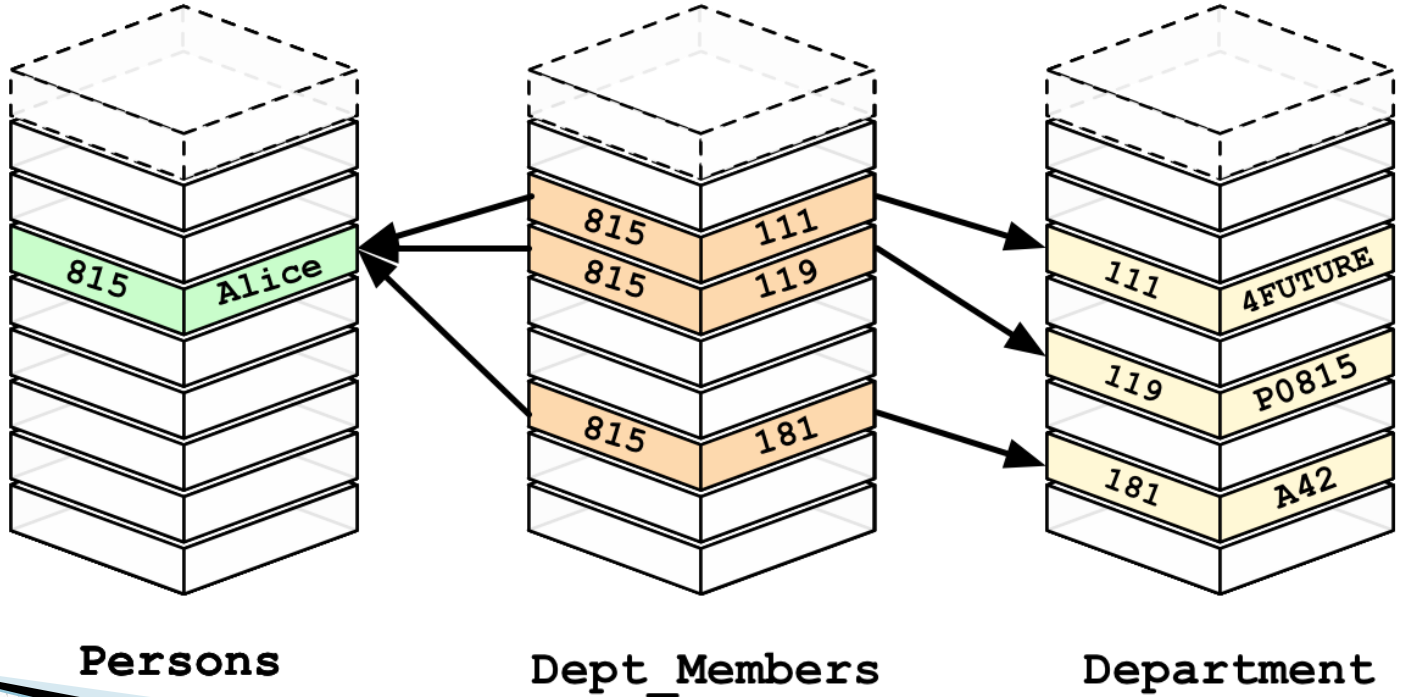
Features

- Relationships instead of Joins for speed
- Less need for multiple products
  - Multi-model
  - SQL
- Easier scalability
- Schemaless, schema full and schema-mixed
- ACID compliance
- HTTP REST for easy integration
- Object Oriented Concepts

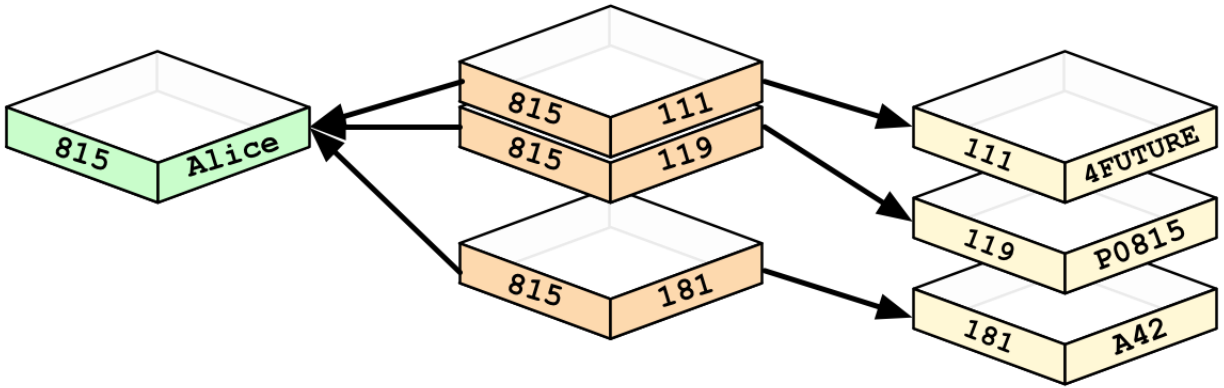
OrientDB Success Stories

- Investigation Unit to uncover hidden assets
- Process clustered IoT data in the cloud
- Provide business insights for targeted sales
- Fraud transaction detection
- Traffic modelling

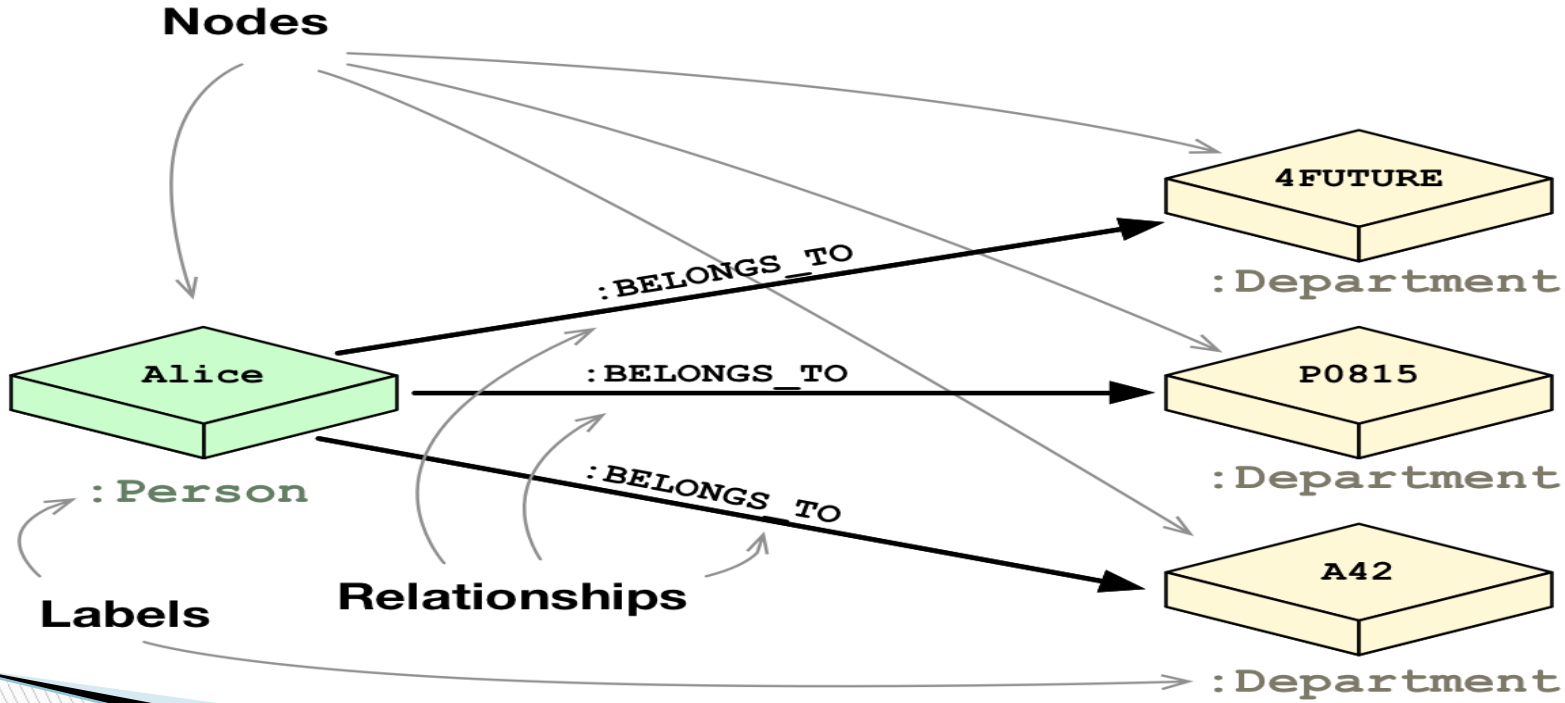
Why Graph Database?



Why Graph Database?



Why Graph Database?



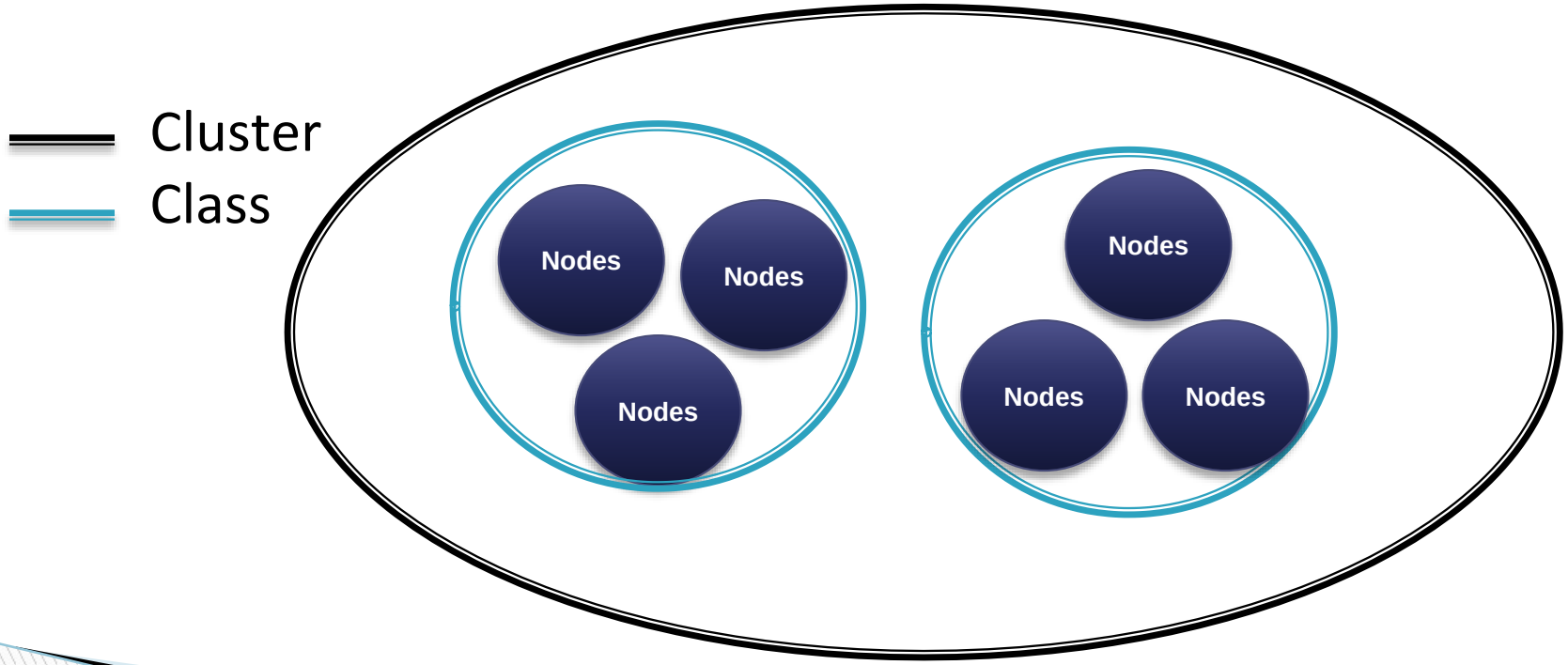
Graph Model Components

- Vertex
  - Unique identifier
  - Incoming Edges
  - Outgoing Edges
  - Basic unit modelled as a Class
- Edge
  - Unique identifier
  - Incoming Vertex (head)and Outgoing Vertex (tail)
  - Connect vertices
  - Regular edges vs lightweight edge

OrientDB Components

- Record – Smallest unit of database
  - Identified by RecordID (clusterID:clusterPosition)
- Classes – Like a table, schemaless, for grouping records
  - Inheritance and Polymorphism
  - Logical grouping
- Clusters – Physical grouping
  - Classes belong to a cluster
  - Parallelism
  - Efficient querying

Graph Model



DOCUMENT MODEL

Person

```
{
  @rid: #1:20,
  First: “John”,
  Last: “Doe”,
  children:  [
    #2:123, #2:124
  ]
}
```

Person

```
{
  @rid: #1:20,
  First: “Jane”,
  Last: “Doe”,
  Birthdate: 06/06/1986,
  children:  [ ],
  Pet: {
    name:”Tom”,
    type:”Cat”,
    color:”grey”
  }
}
```

- A form of data storage
- Commonly Schema-less
- Has a class type
- Can contain Links to other documents
- Can contain other documents
- Has a unique ID called record ID in format : #<clusterID>:<position>

Schema–Full and Mixed Mode Schema

Person
Birthdate : <Date>
First: <No Type>
Last: <No Type>
.
.
Additional fields can be inserted

Strict Mode: False

Person
First: <String>
Last: <String>
Birthdate:<Date>

If insert tries to use a field other than these, the query fails

Strict Mode: True

- All classes of documents can be set into Mixed-Mode Schema by setting Strict Mode: False.
- If strict mode is set to True, the class is Schema-Full.

RELATIONSHIPS IN DOCUMENT MODEL

Person

```
{
  @rid: #1:20,
  First: “John”,
  Last: “Doe”,
  children:  [
    #2:123, #2:124
  ]
}
```

Person

```
{
  @rid: #1:20,
  First: “Jane”,
  Last: “Doe”,
  Birthdate: 06/06/1986,
  children:  [ ],
  Pet: {
    name:”Tom”,
    type:”Cat”,
    color:”grey”
  }
}
```

Relationships in a document model can be of two types:

- Referenced
- Embedded

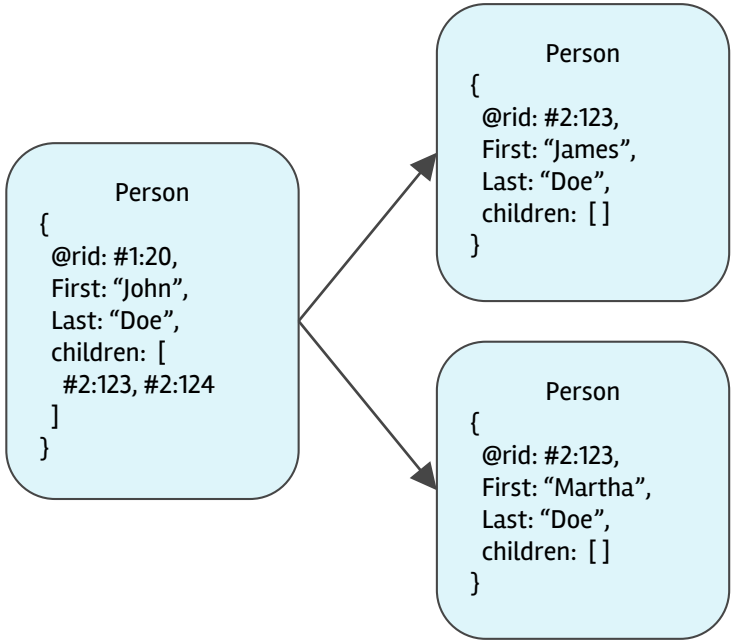
RELATIONSHIP : EMBEDDED

Person

```
{
  @rid: #1:20,
  First: “Jane”,
  Last: “Doe”,
  Birthdate: 06/06/1986,
  children:  [ ],
  Pet: {
    name:”Tom”,
    type:”Cat”,
    color:”grey”
  }
}
```

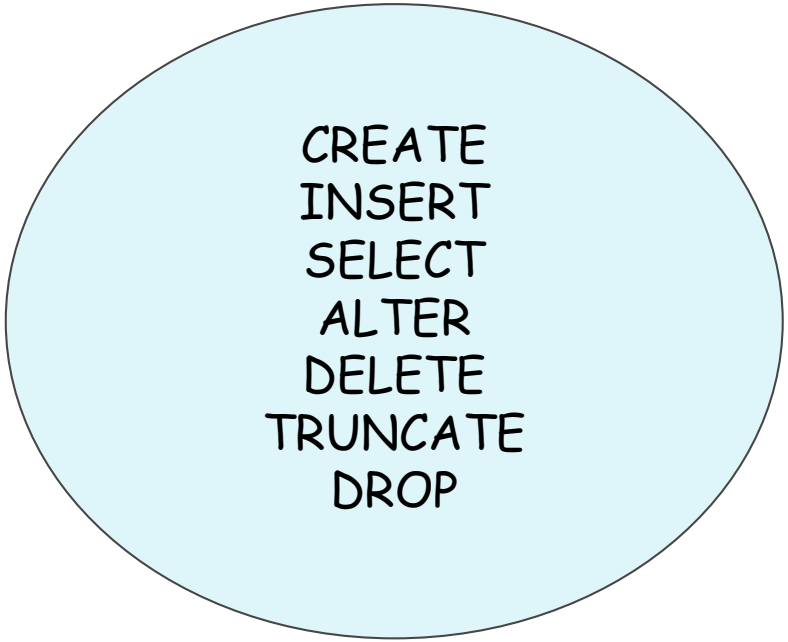
- Embedded document is dependent on parent for existence.
- Does not have unique ID.
- Uses embeddedlist, embeddedset, embeddedmap.

RELATIONSHIP : REFERENCED



- One-Many, Many-One and Many-Many relationships handled using containers like linklist, linkmap, linkset.
- Stores RID of linked records.
- Speeds up traversing.

EXTENDED SQL



- Easy to learn for existing developers.
- Additional extensions for traversing graph.

EXTENDED SQL – CREATE

```
create database <database-url> <user> <password>
<storage-type> [<db-type>]

create class <class-name>

create property <class>.<property> <type> [linktype]

create vertex

create edge from <vertex id using select> to <vertex
id using select>
```

EXTENDED SQL – ALTER

```
alter class - Ex. alter class Person STRICTMODE true

alter property <class>.<property> <attributeName>
<attributeValue>
```

EXTENDED SQL – SELECT

```
select [<projections>] from <target> [where
<conditions>] [group by <field>] [order by <fields>
[asc|desc]] [skip <numRecords>] [limit <MaxRecords>]
```

EXTENDED SQL – INSERT/UPDATE

```
insert into <target> [ (<fields>) values (<values>) |
     set <field>=<expression> ]
```

```
update <target> [ SET|REMOVE|INCREMENT|ADD
<field>=<value>[,]*] [where <conditions>]

[limit <MAX-RECORDS>]
```

EXTENDED SQL – DELETE

Delete works same as SQL. To delete all records of a class or cluster, we can use the Truncate Statement.

EXTENDED SQL – TRAVERSE

- TRAVERSE * FROM #1:12
- TRAVERSE * FROM #1:12 $depth <= 2
- SELECT FROM PERSON any() traverse(0,3) (firstname=”JOHN”)
- SELECT out(‘friends’).out(‘friends).out(‘friends’) FROM #1:!2
- SELECT DIJKSTRA($current,#1:12,’weight’) from V
- TRAVERSE friends from #1:12 while $depth <= 3 STRATEGY BREADTH_FIRST
- SELECT $path FROM (TRAVERSE any() FROM #1.12 while depth<=2)

EXTENDED SQL – TRAVERSE EXAMPLE

```
SELECT $path FROM ( TRAVERSE out() FROM #17:1419 WHILE $depth <= 10 STRATEGY BREADTH_FIRST)
```

$path
(#17:1419)
(#17:1419).out[0](#16:2262)
(#17:1419).out[1](#12:6767)
(#17:1419).out[0](#16:2262).out[0](#15:619)
(#17:1419).out[0](#16:2262).out[0](#15:619).out[1](#12:6780)
(#17:1419).out[0](#16:2262).out[0](#15:619).out[2](#12:6826)
(#17:1419).out[0](#16:2262).out[0](#15:619).out[3](#12:6998)
(#17:1419).out[0](#16:2262).out[0](#15:619).out[4](#12:7045)
(#17:1419).out[0](#16:2262).out[0](#15:619).out[5](#12:7092)
(#17:1419).out[0](#16:2262).out[0](#15:619).out[6](#12:7100)
(#17:1419).out[0](#16:2262).out[0](#15:619).out[7](#12:7112)
(#17:1419).out[0](#16:2262).out[0](#15:619).out[8](#12:7131)
(#17:1419).out[0](#16:2262).out[0](#15:619).out[9](#12:7137)
(#17:1419).out[0](#16:2262).out[0](#15:619).out[10](#12:7139)
(#17:1419).out[0](#16:2262).out[0](#15:619).out[11](#12:7184)
(#17:1419).out[0](#16:2262).out[0](#15:619).out[12](#12:7211)
(#17:1419).out[0](#16:2262).out[0](#15:619).out[13](#12:7236)
(#17:1419).out[0](#16:2262).out[0](#15:619).out[14](#12:7264)
(#17:1419).out[0](#16:2262).out[0](#15:619).out[15](#12:7265)
(#17:1419).out[0](#16:2262).out[0](#15:619).out[16](#12:7316)

SCRIPTS

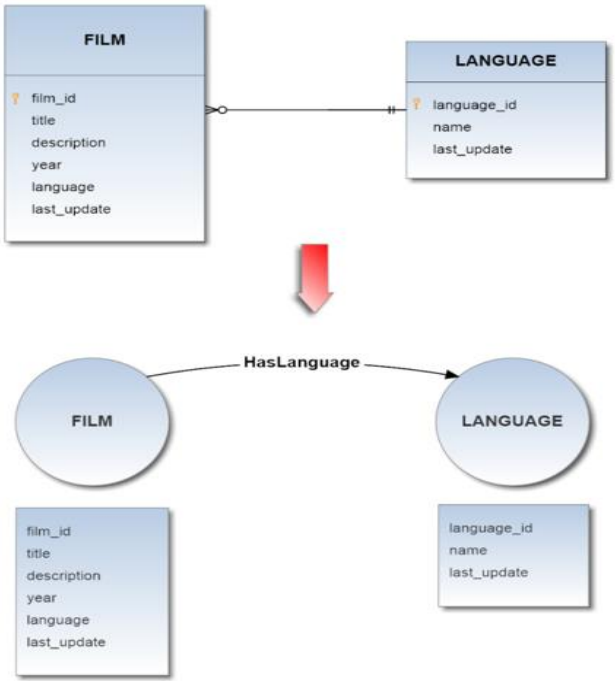
- OrientDB also provides a way to write Server Side Scripts.
- Currently SQL and Javascript are supported.
- More languages to be added in the future.

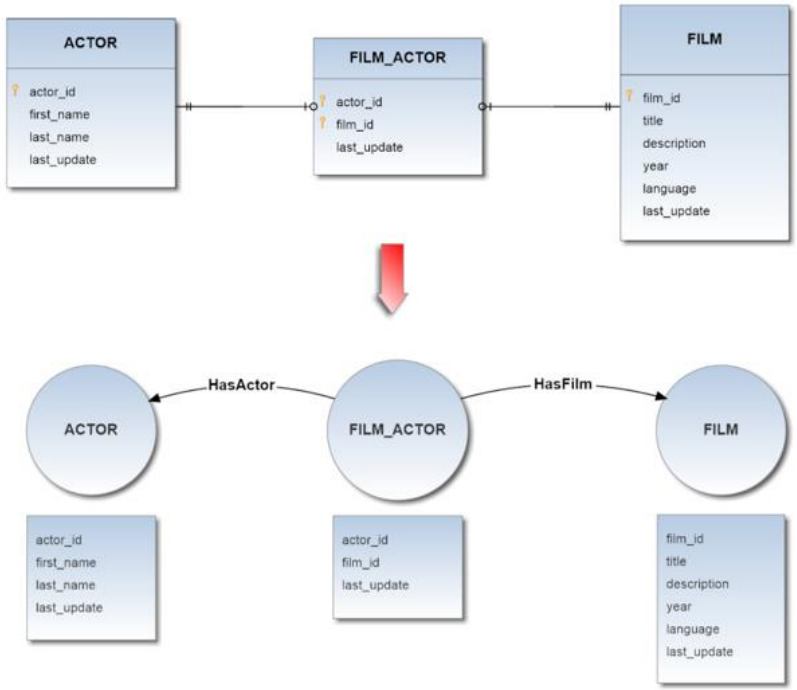
OrientDB Teleporter

ORIENTDB TELEPORTER

- Compatible with most of the RDBMS accessible with JDBC.
- As per OrientDB, Teleporter has been tested successfully with Oracle, SQLServer, MySQL, PostgreSQL and HyperSQL.
- The user can choose between two approaches to convert Relational Database to Graph Database,
  - Naive Strategy
  - Naive-Aggregate Strategy

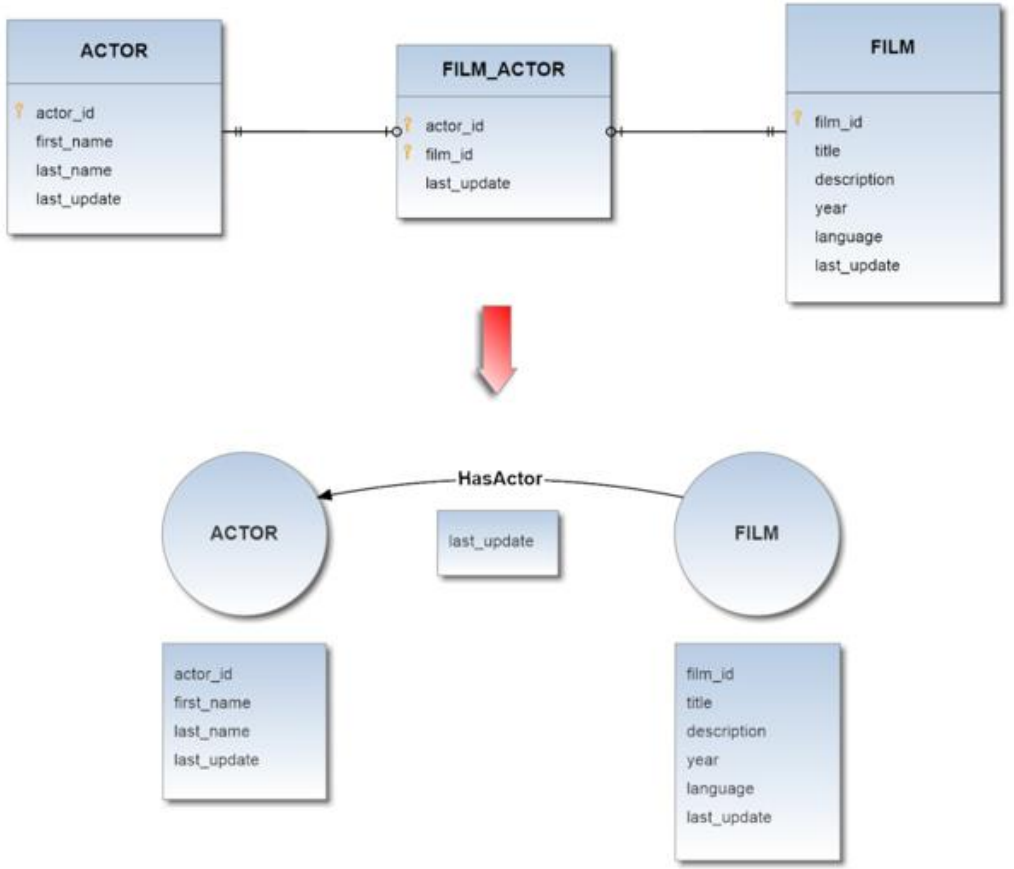
Naive Strategy





Source: www.orientdb.com

Naive–Aggregate Strategy



Source: www.orientdb.com

Driver API's

- Native Binary
- HTTP REST/JSON
- Java Wrapped

Native Binary

- Directly against the TCP/IP socket using the binary protocol
- Fastest way to interface a client application to an OrientDb server instance

Binary Protocol

- Intended to be read by a machine rather than humans
- Better performance as compared to text protocols such as HTTP or IRC
- Terse which translates into speed of transmission and interpretation

HTTP REST/JSON

- Talk with a OrientDB Server instance using the HTTP protocol and JSON
- Authentication and Security
- Keep–Alive for better performance.

HTTP Methods

- GET, to retrieve values from the database.
- POST, to insert values into the database.
- PUT, to change values into the database.
- DELETE, to delete values from the database.

Java Wrapped API

- OrientDB is written in Java
- This means that you can use its Java API's without needing to install any additional drivers or adapters
- Layer that links directly to the native Java driver.

Graph API

- If you work with graphs and want portable code across other Graph databases and OLAP systems.
- Easiest to switch to this when migrating from other Graph Databases, such as Neo4J or Titan.
- You can use OrientDB as a Graph Database, allowing you to work with Vertices and Edges.

Document API

- If your domain fits Document Database use case.
- Easiest to switch to this when migrating from other Document Databases, such as MongoDB and CouchDB.
- Handle records and documents.

Object API

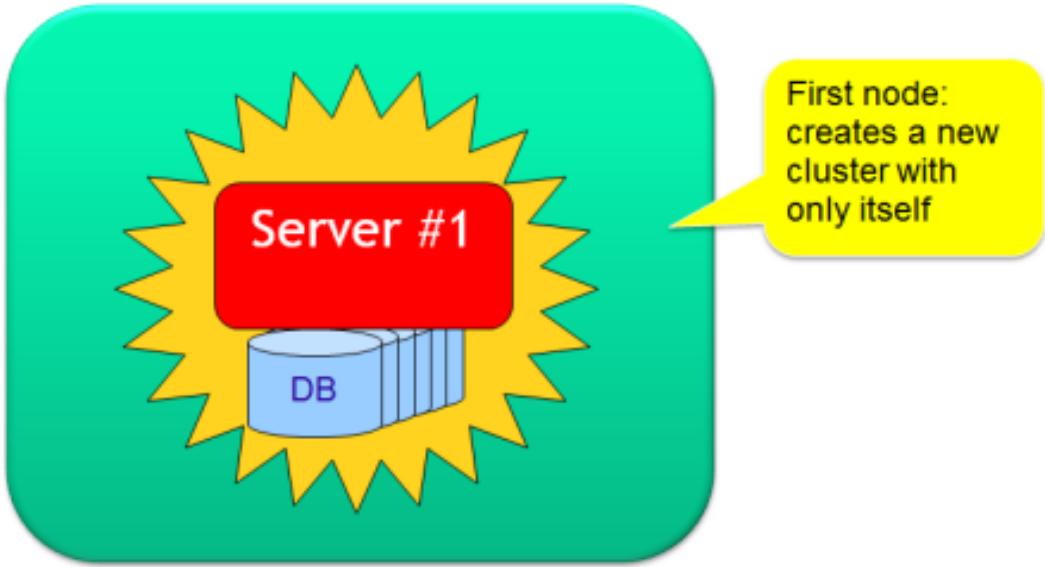
- Full Object Oriented abstraction that binds all database entities to POJO (Plain Old Java Objects).
- Easiest to switch to this when migrating from JPA applications.

Scaling

- Capability to support large volume of data
- OrientDB can be distributed across different servers and used in different ways to achieve the maximum of performance
- Multi master strategy over master – slave

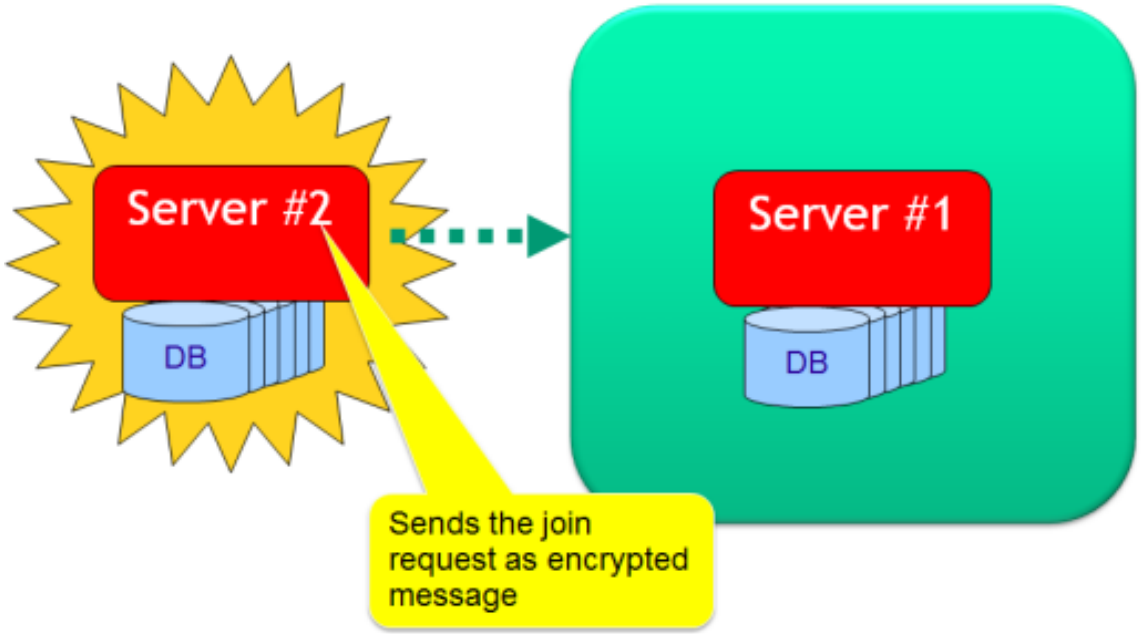
Distributed Architecture Lifecycle

- Discover if an existing cluster is available to join
- If available join the cluster otherwise
- Create a new cluster

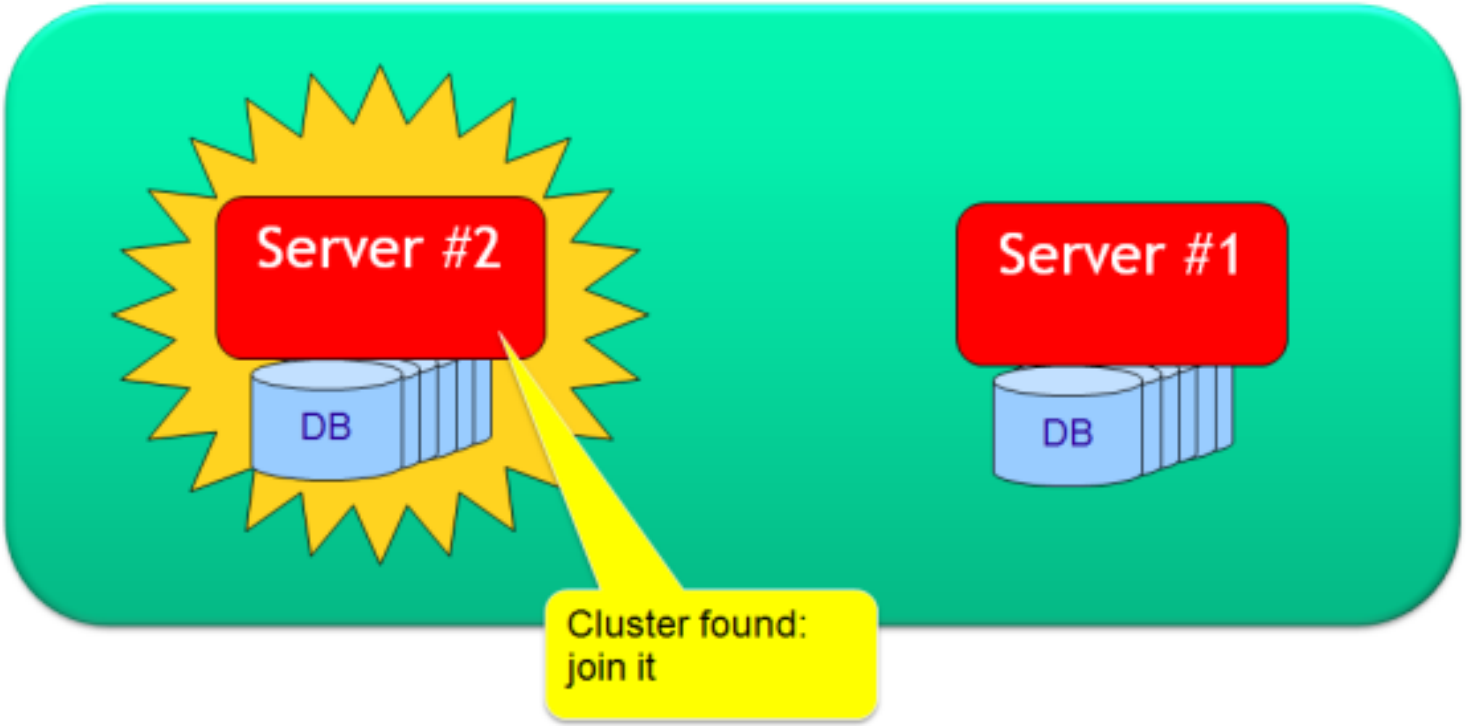


Distributed Architecture Lifecycle

- Join to an existing cluster
- Unique cluster name

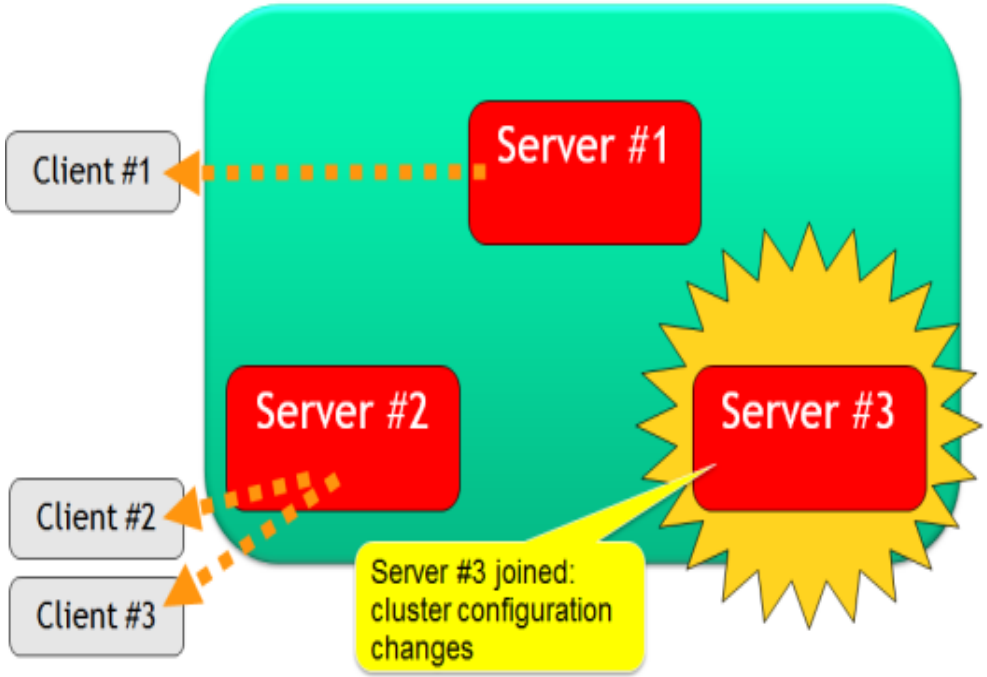


Distributed Architecture Lifecycle



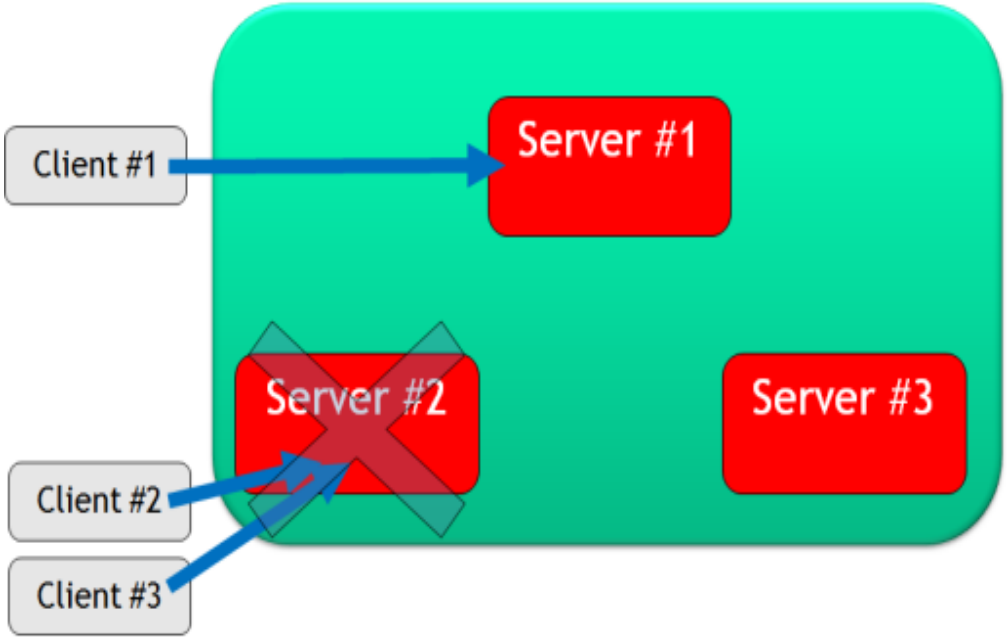
Distributed Architecture Lifecycle

- Configuration broadcasted for each join and release



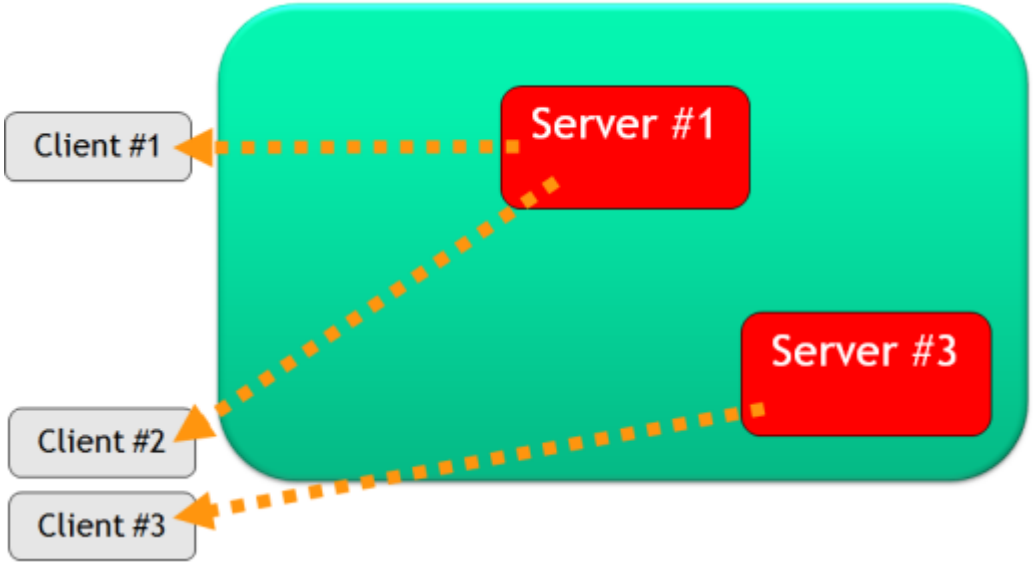
Distributed Architecture Lifecycle

- If node is unreachable treat as if node has left the cluster



Distributed Architecture Lifecycle

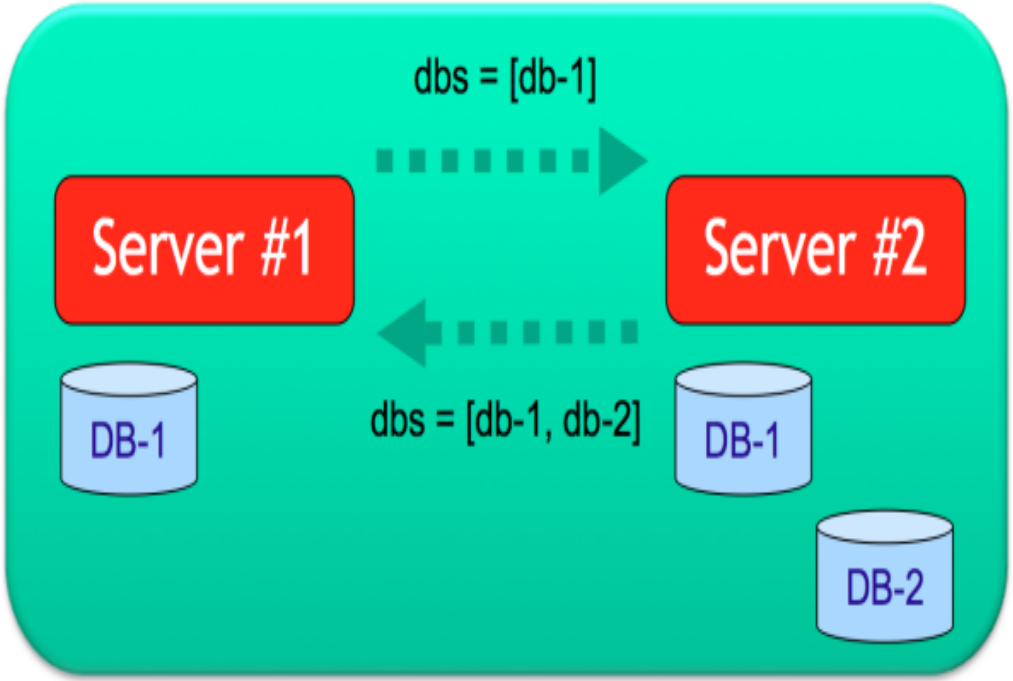
- If node is unreachable treat as if node has left the cluster



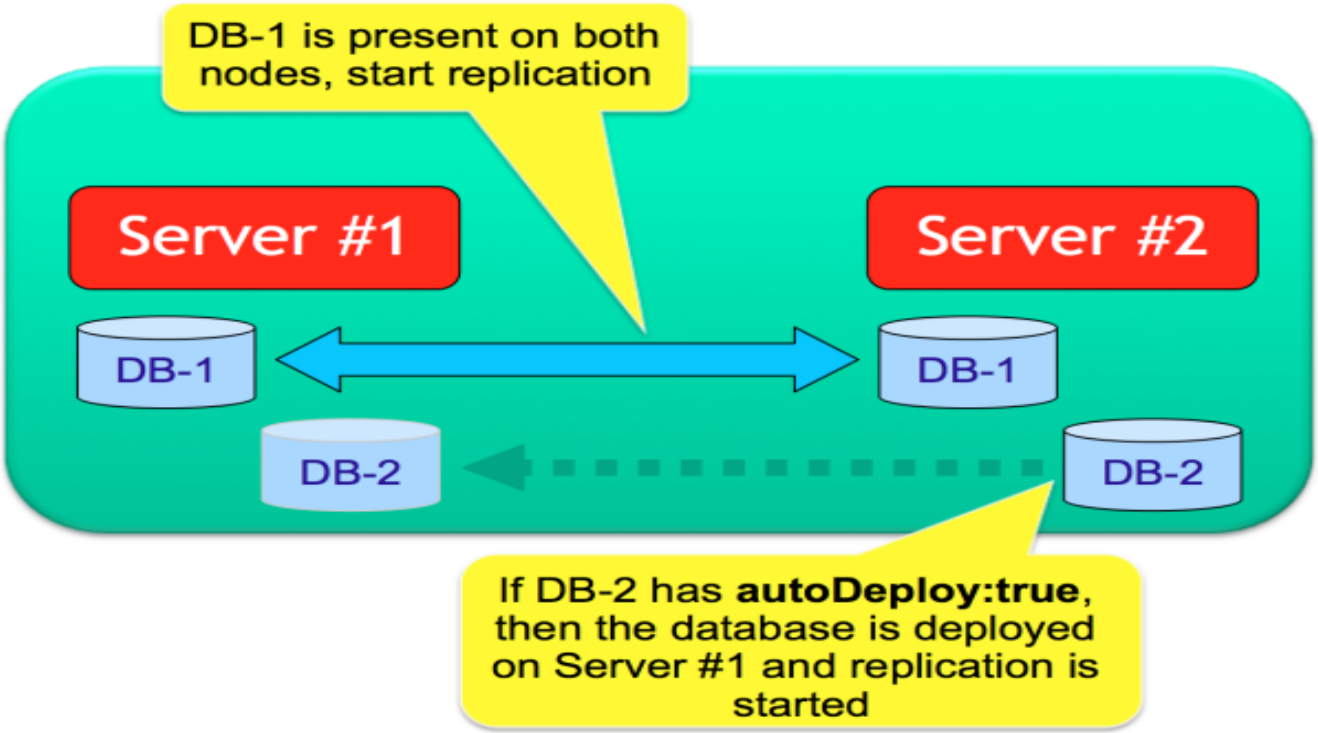
Distributed Architecture Replication

- List of databases

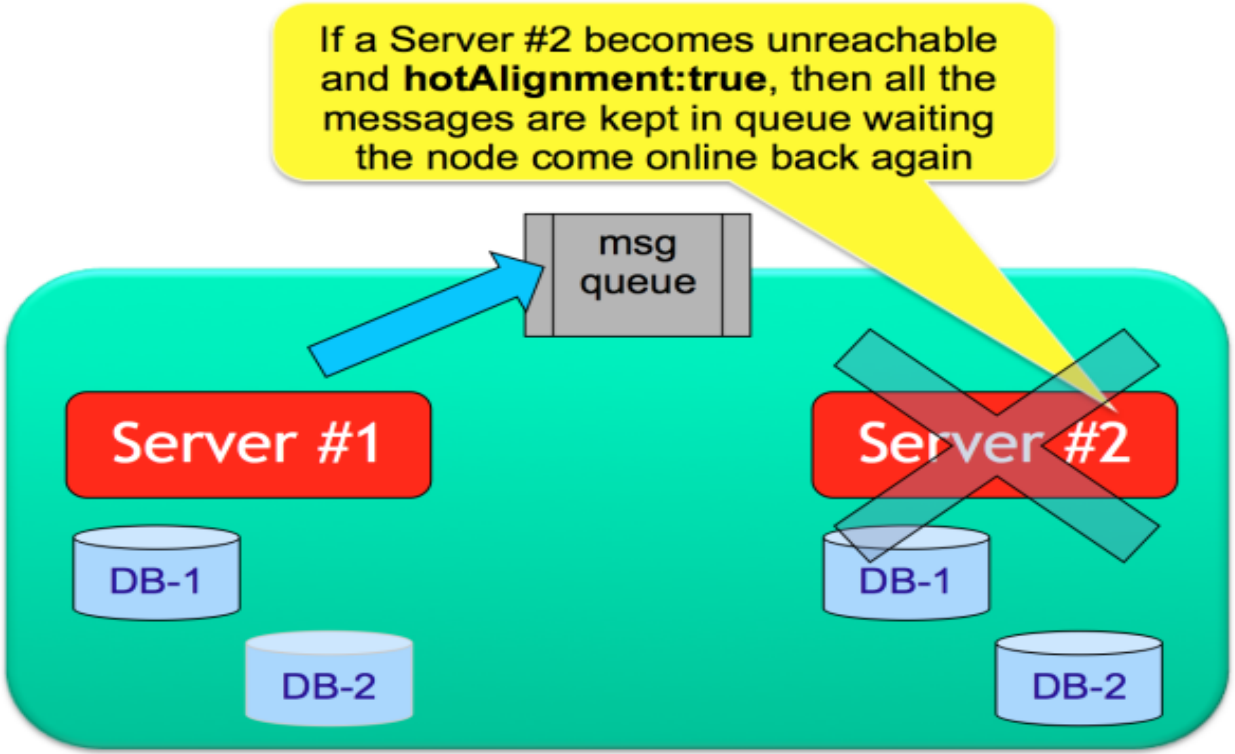
Shared between nodes



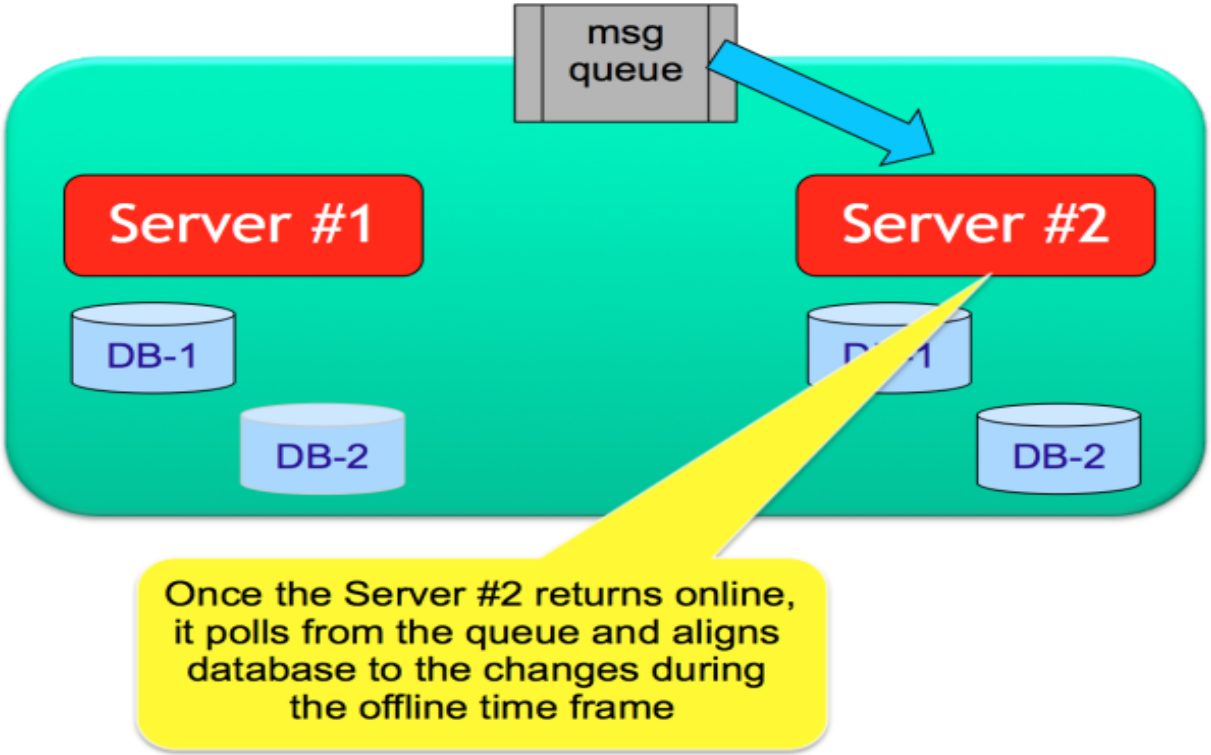
Distributed Architecture Replication



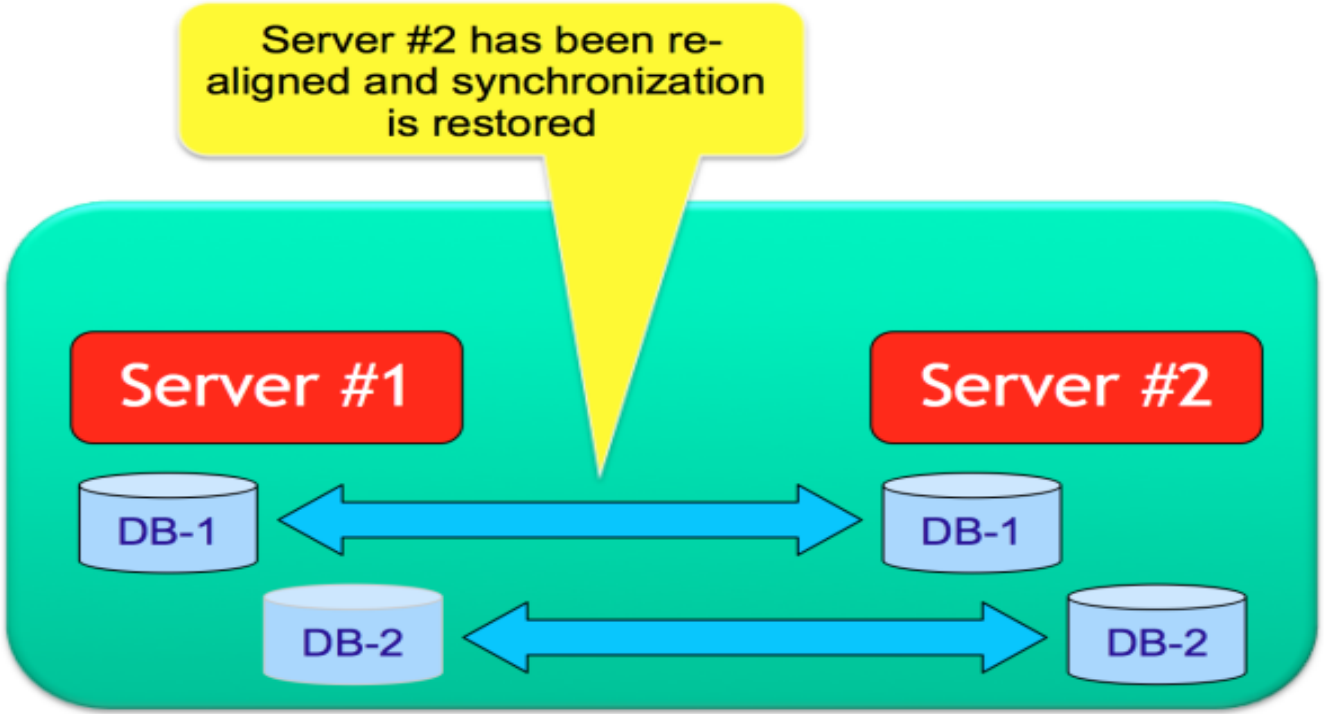
Distributed Architecture Replication



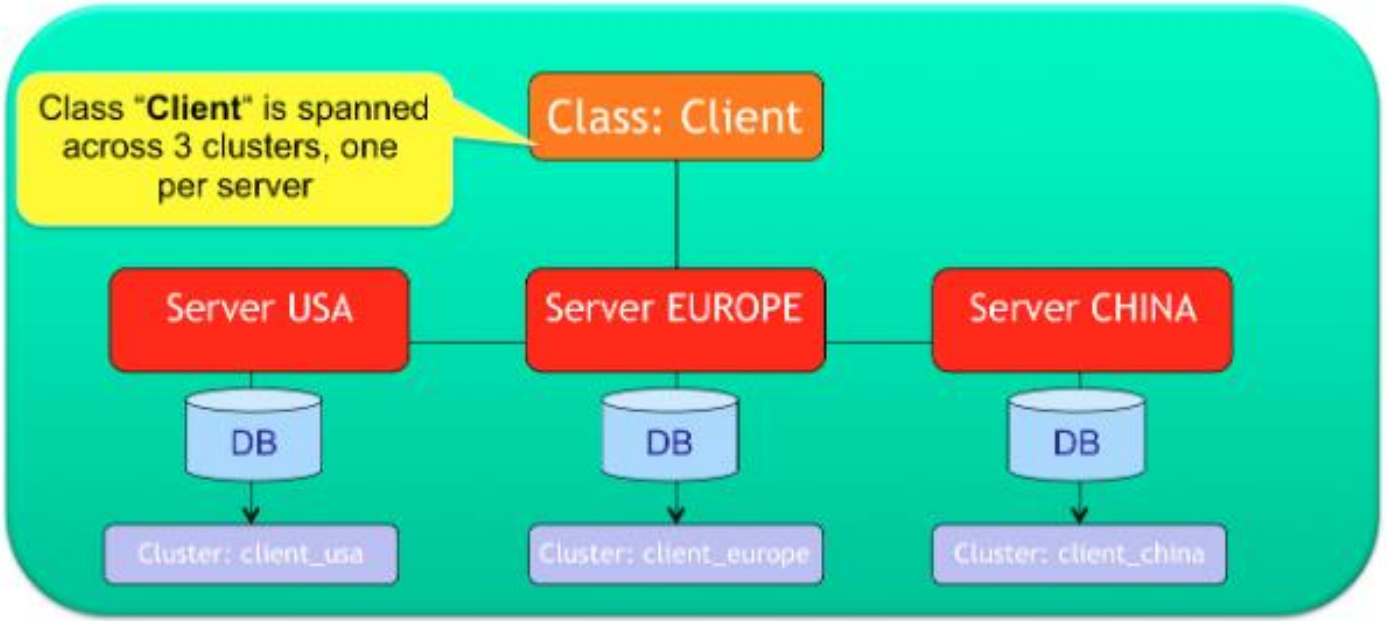
Distributed Architecture Replication



Distributed Architecture Replication

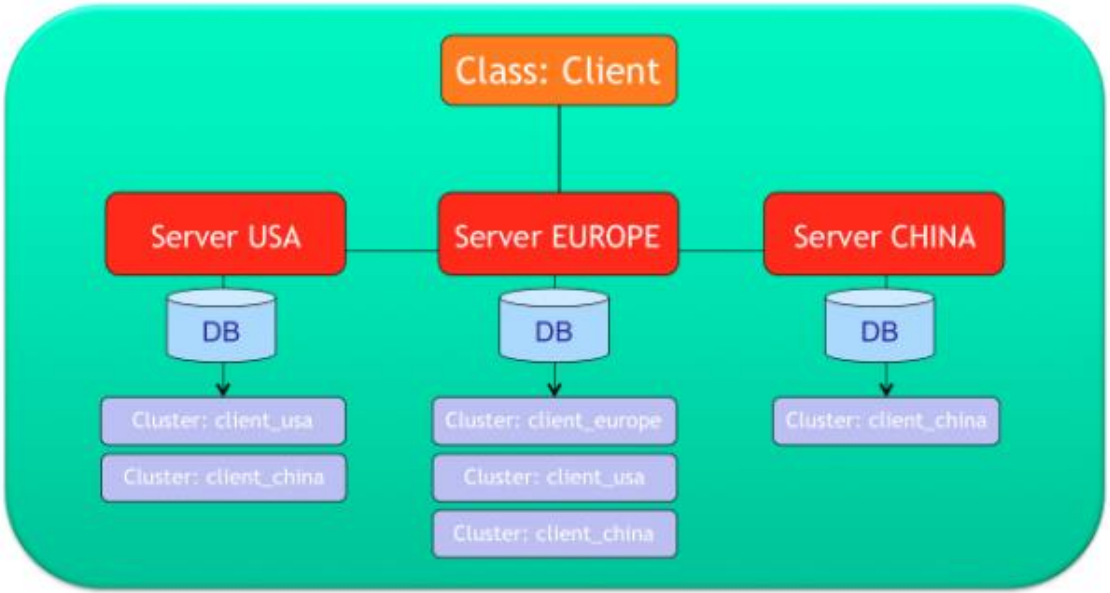


Distributed Architecture Sharding

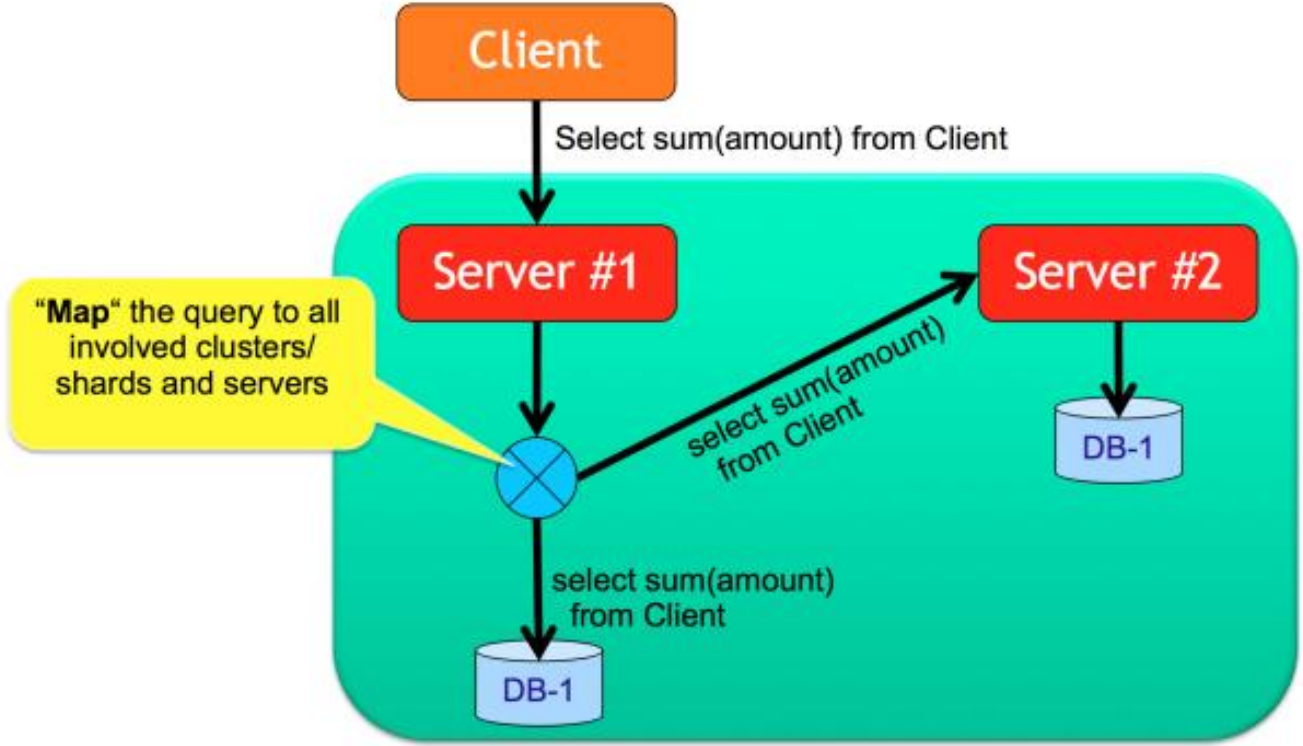


Distributed Architecture Sharding

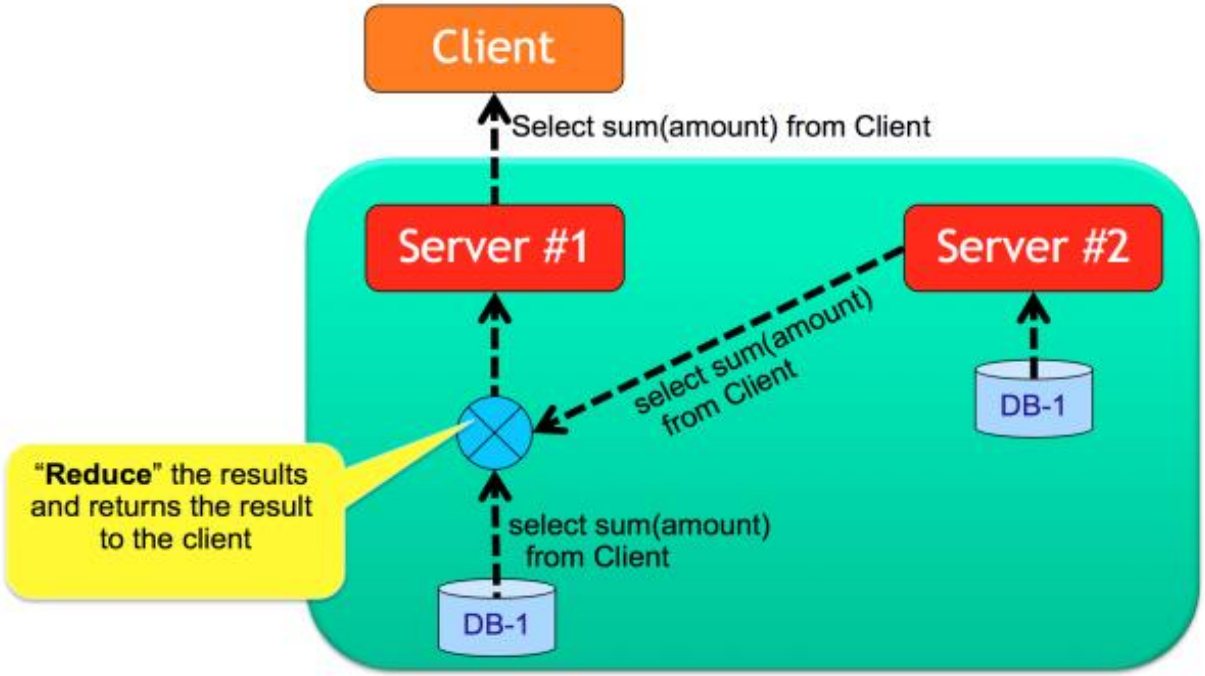
- Cluster Locality
- Multiple servers per cluster
- Create records
- Update & delete
- Read records



Distributed Architecture Sharding



Distributed Architecture Sharding



Concluding Remarks

- OrientDB is a multi-model solution
- Looks to cater across the breadth of industry
- Slowly gaining market share
- Some maintenance controversies
- Trails Neo4j and MongoDB in terms of popularity

THANK YOU!!