Real Federation Database System leveraging PostgreSQL FDW

Yotaro Nakayama

Technology Research & Innovation
Nihon Unisys, Ltd.

Motivation for the Federation Database

Motivation

- **Data Integration Solution view point**
  - Data Integration and Information Integration are hot topics in these days.
  - Requirement of information integration has increased in recent year.
- **Technical view point**
  - Study of Distributed Database has long history and the background of related technology of Distributed Database has changed and improved. And now a days, the data moves from local storage to cloud. So, It may be worth rethinking the technology.

Topics

1. **Introduction - Federation Database System as Virtual Data Integration Platform**
2. **Implementation of Federation Database with PostgreSQL FDW**
   - Foreign Data Wrapper Enhancement
   - Federated Query Optimization
3. **Use Case of Federation Database**

   Example of Use Case and expansion of FDW
4. **Result and Conclusion**

Topics

1. **Introduction - Federation Database System as Virtual Data Integration Platform**
2. **Implementation of Federation Database with PostgreSQL FDW**
   - Foreign Data Wrapper Enhancement
   - Federated Query Optimization
3. **Use Case of Federation Database**

   Example of Use Case and expansion of FDW
4. **Result and Conclusion**

PostgreSQL FDW as Virtual Data Integration Platform

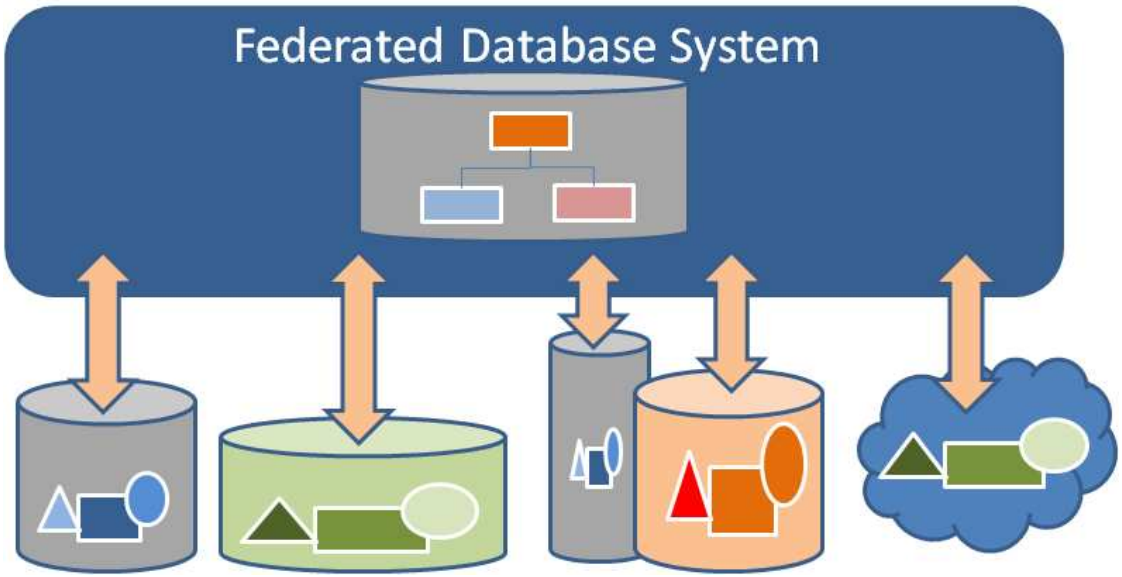
- **Transparent access to heterogeneous DBMS**
- **Optimized Database schema design according to optimized business process**

Functionality

Access heterogeneous external data source as local data
Location, Vendor and Platform Transparency
Without change of existing system
Real-time Data Reference and Data Update

Effectiveness

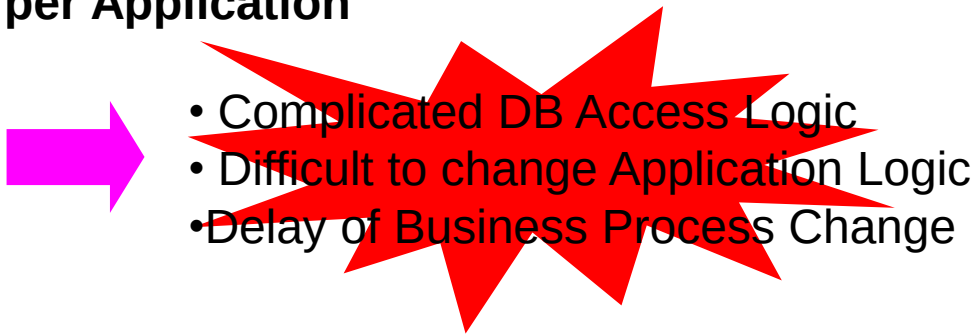
Optimization of Business Process across multiple business systems
Integrated Database access
Avoid redundant data management as “Single Version of the Truth ”



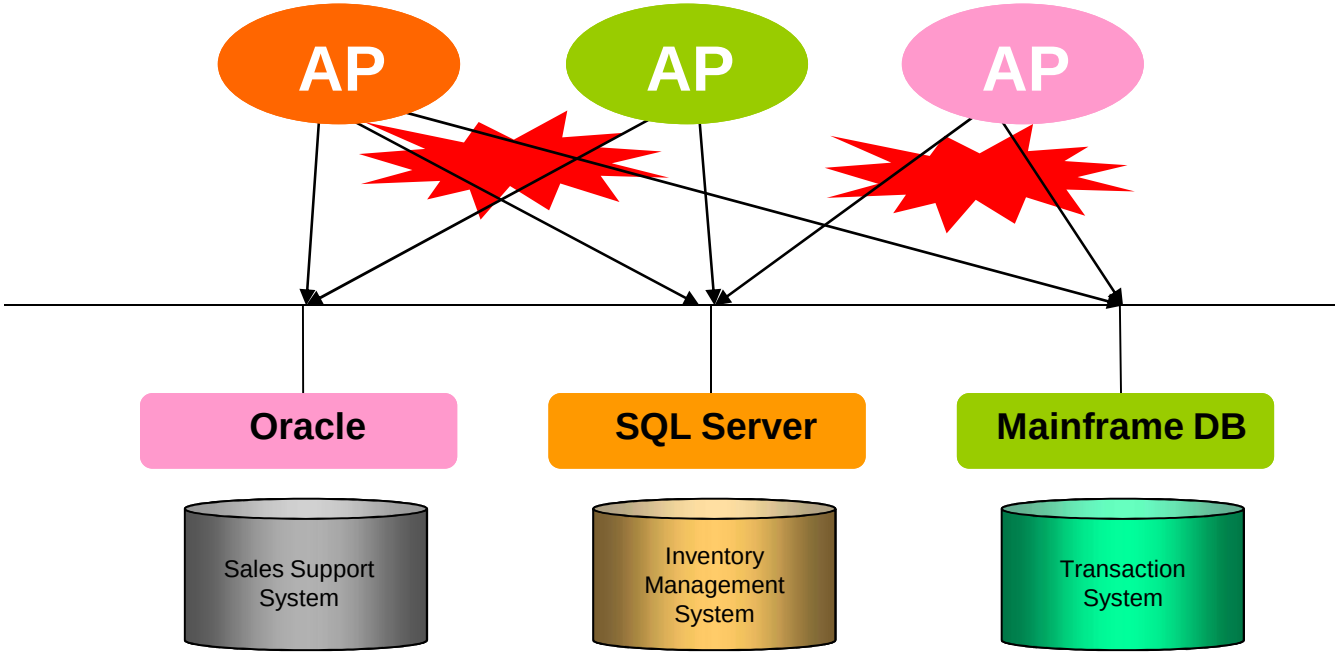
Problem of Heterogeneous Data Integration

Accessing to Business System per Application

- Maintaining address per DBMS
- Vendor Local SQL
- Character Code Exchange
- Semantics of Data Model



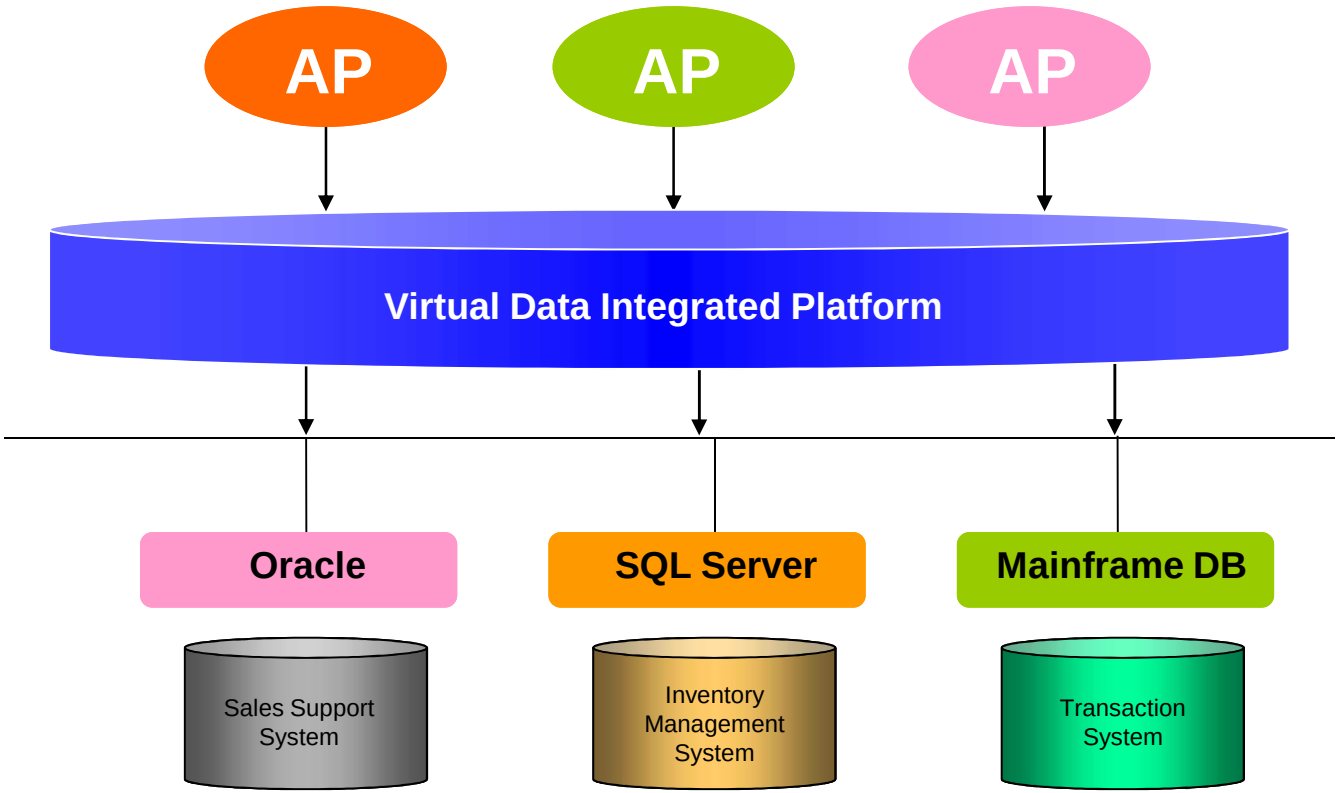
- Complicated DB Access Logic
- Difficult to change Application Logic
- Delay of Business Process Change



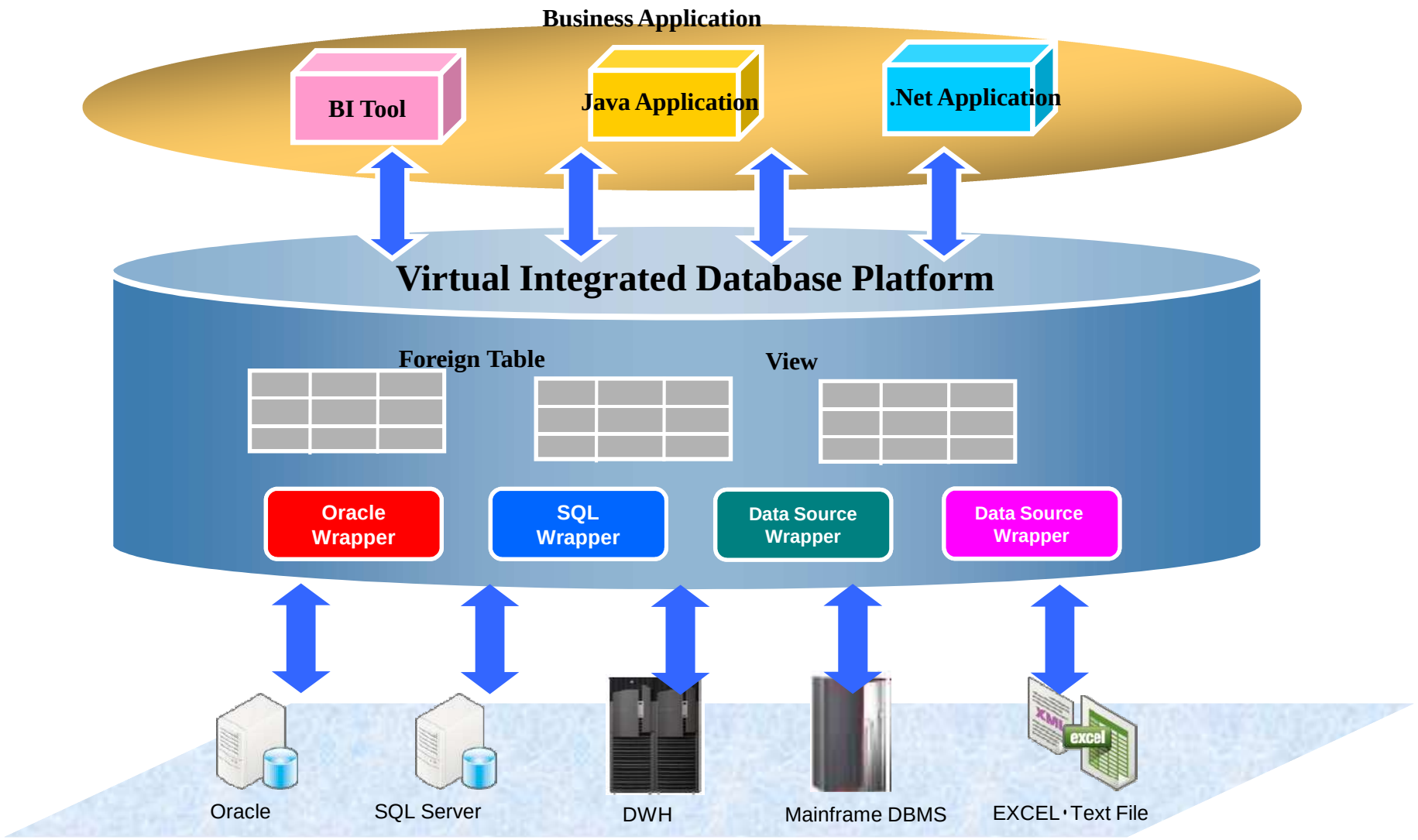
Advantage of Federation Database

- Without change of existing System and Database
- Transparent Access to Distributed Databases through virtual schema
- Transparency to Vendor, Platform and Location

- Improve Data Access Maintainability
- Improve Change and Maintainability to Business Logic



Federated Database System Platform

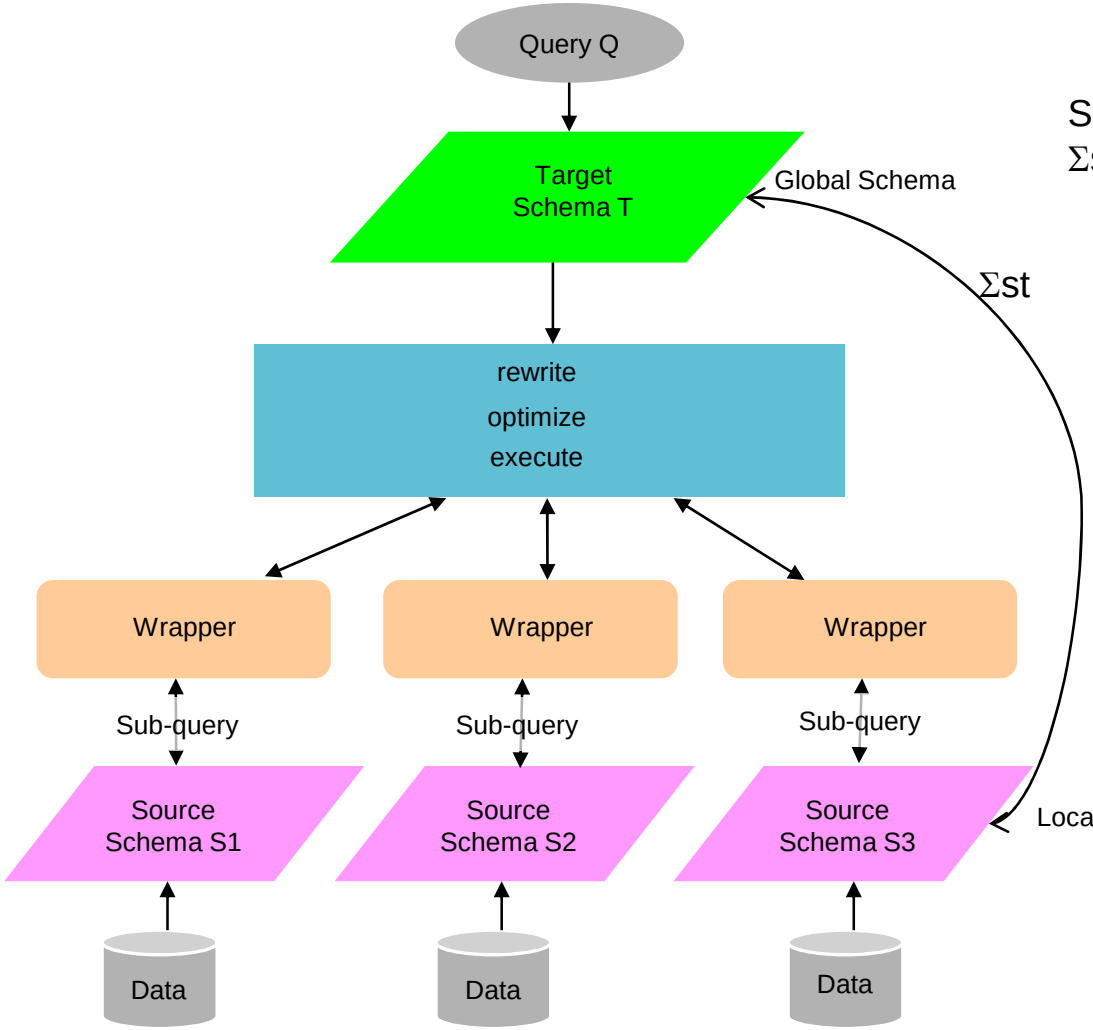


Topics

1. Introduction - Federation Database System as Virtual Data Integration Platform
2. **Implementation of Federation Database with PostgreSQL FDW**
   - **Foreign Data Wrapper Enhancement**
   - Federated Query Optimization
3. Use Case of Federation Database

   Example of Use Case and expansion of FDW
4. Result and Conclusion

Schema mapping in Data Integration

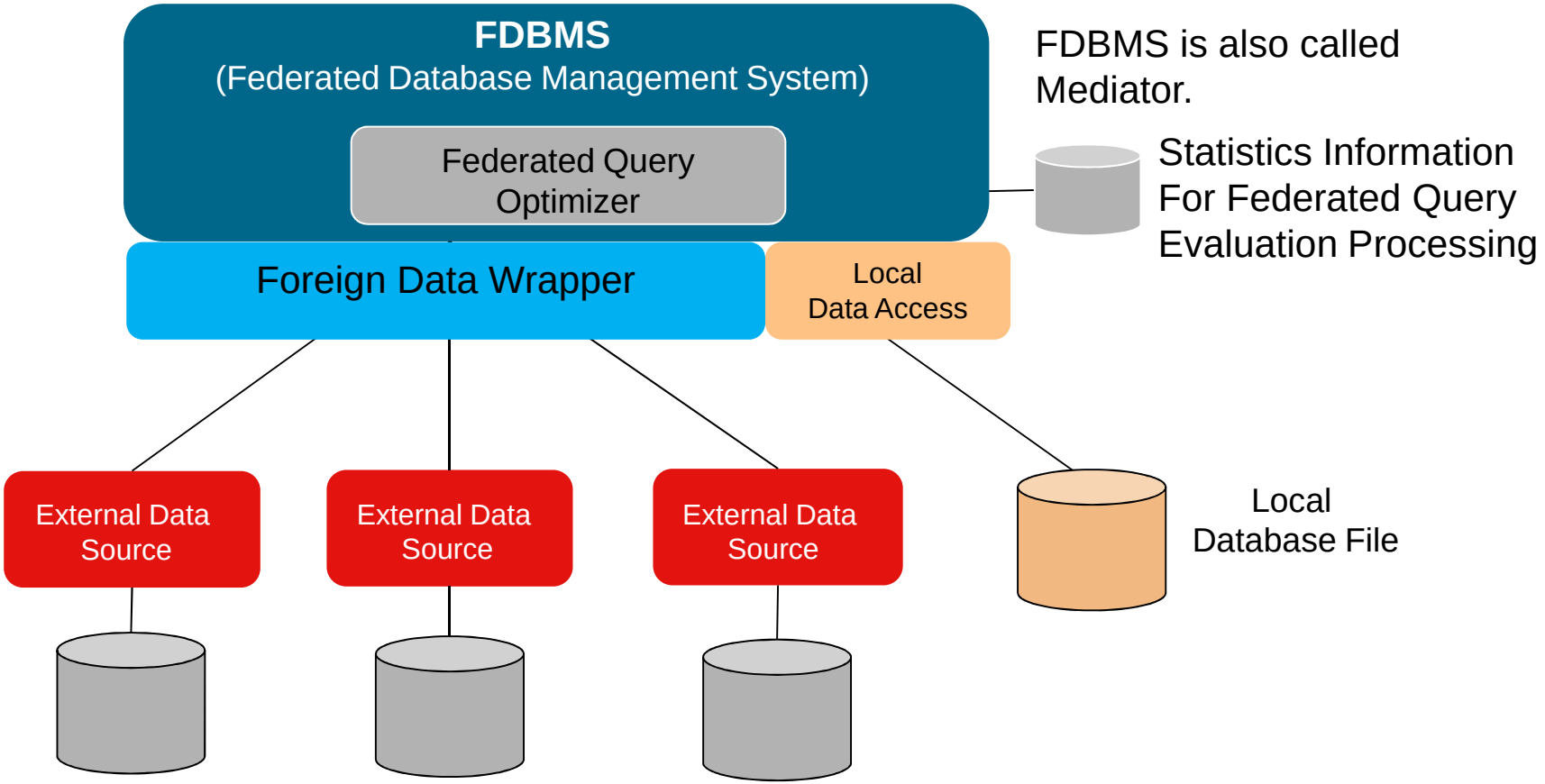


Schema Mapping M = (S, T, Σ)
Σst : set of dependencies from source to target.

Types of Schema Mapping are as follows:
- Global as View
- Local as View
- Global and Local as View

Federated Database is understood as GAV.

Review of Federation Database Architecture



Implementation of Federation Database with PostgreSQL FDW (1)

- **What should be considered for Federation Database System with PostgreSQL FDW?**
  - SQL generation and data type mapping are needed to be considered. Because, current FDW is only assumed for the connection between PostgreSQL.

- **Various Heterogeneity**
  - Location and/or Physical Address
  - Connection protocol
  - Connection granularity and object
  - Data Model
  - Query Language
  - Data Type, Data value – character code, precision
- **Distributed Query**
  - Distributed Query Optimization and Processing

Implementation of Federation Database with PostgreSQL FDW (2)

Two aspects of approaches to the FDB implementation

- **Data Access Layer**
  - Data Access Protocol
  - SQL generation for External Data Source

- **Federation Aware Query Optimization**
  - Adequate Pushdown Condition and/or Strategy for Query shipment
  - Adequate Join order and Join method
  - Statistics information for External Data Source

Data Access Layer

- **Data Access Protocol**
- **SQL generation for External Data Source**

Heterogeneity of External Data Source

- SQL dialect
- Access method or protocol
  - ODBC, JDBC, Native Driver, Web Service
- Database Object Mapping: Database, Schema, Table, Column, Type, Domain, User, Role, UDF, Connection
  - Oracle connects per instance, SQL Server and PostgreSQL connects per database.
  - Data Type Mapping
- Character Set, Character Code conversion
  - Kanji Character: EUC, SJIS, Unicode
- Data format, locale
  - date, time format, time zone

Prototyping of Foreign Data Wrapper Enhancement

- **Based on PostgreSQL FDW**
  - We tried to adopt the Patch around Nov. 2010 Version.
- **Enable access to the Heterogeneous Relational Database**
  - Oracle 11g Linux (CentOS 5.5)
  - Microsoft SQL Server 2008 (Windows2008 Server)
  - PostgreSQL FDW for ODBC
- **Developing Wrapper for Oracle, SQL Server and PostgreSQL FDW for ODBC.**
- **Use UnixODBC for ODBC driver**
- **Enhanced Deparse for dialect of SQL Syntax**
  - Translation from parse tree to SQL
  - Differences of SQL between Oracle and SQL Server

Review of PostgreSQL SQL/MED feature

- **SQL/MED Standard**
  - Information technology — Database languages — SQL — Part 9: Management of External Data (SQL/MED) ISO/IEC 9075-9
    - External Data Source such as Multimedia Data, Relational Database

- **FDW Definition Syntax**
  - Foreign Data Wrapper Definition

    CREATE FOREIGN DATA WRAPPER *hoodb_wrapper*
    HANDLER *hoodb_fdw_handler*
    VALIDATOR *hoodb_fdw_validator*;
  - Foreign Server Definition

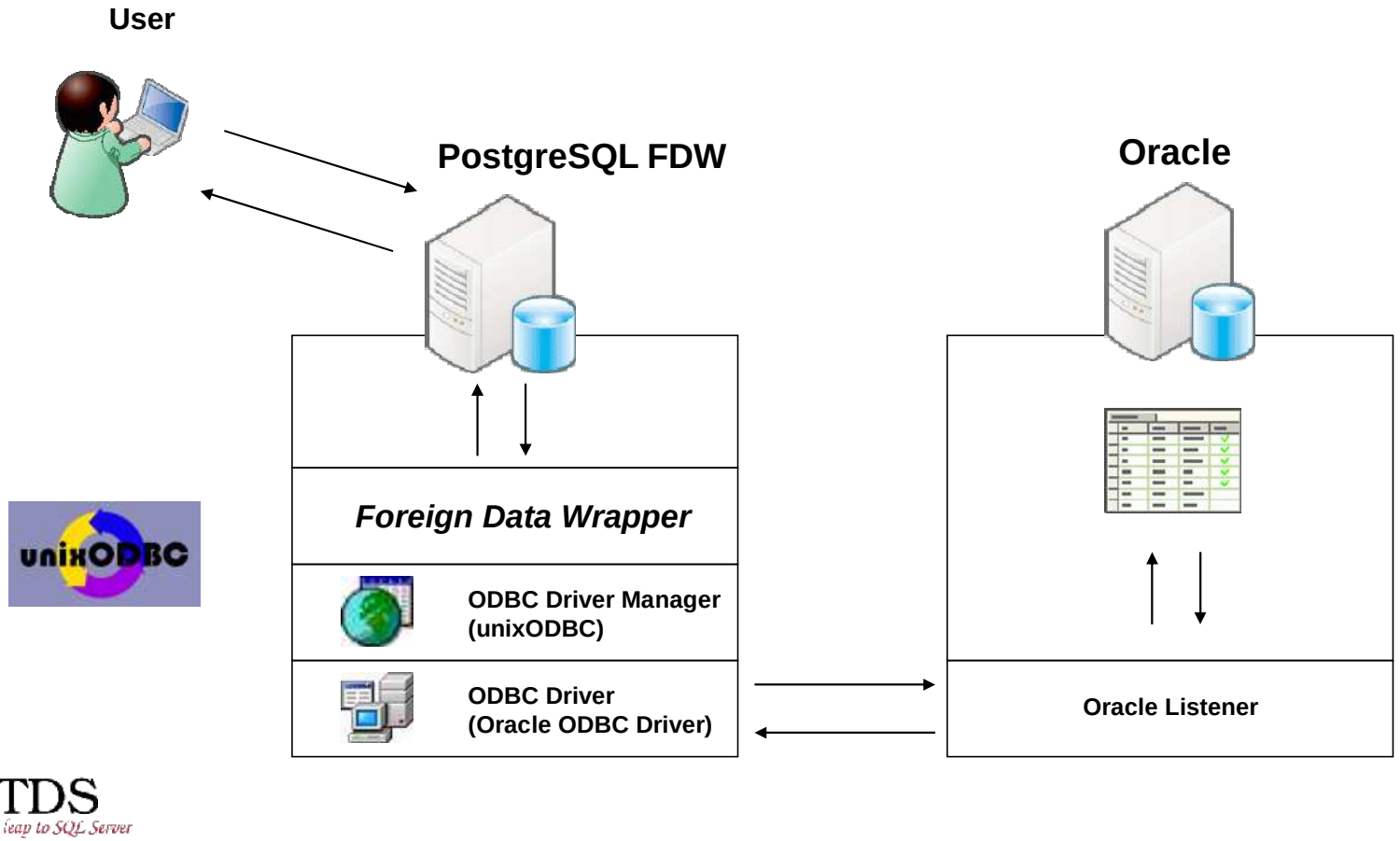
    CREATE SERVER *hoo_sv1*
    FOREIGN DATA WRAPPER *hoodb_wrapper*
    OPTIONS (host 'localhost', dbname 'fdb1');
  - Foreign Table Definition

    CREATE FOREIGN TABLE *hogeTable (c1 int, c2 char(50))* [*INHERTIS (parent)*] SERVER *hoo_sv1* OPTIONS (...);
  - User Definition

    CREATE USER MAPPING FOR postgres SERVER fdb1
    OPTIONS (user ’scott’, password ’tiger’);

FDW using UnixODBC Adopter for ODBC Connection

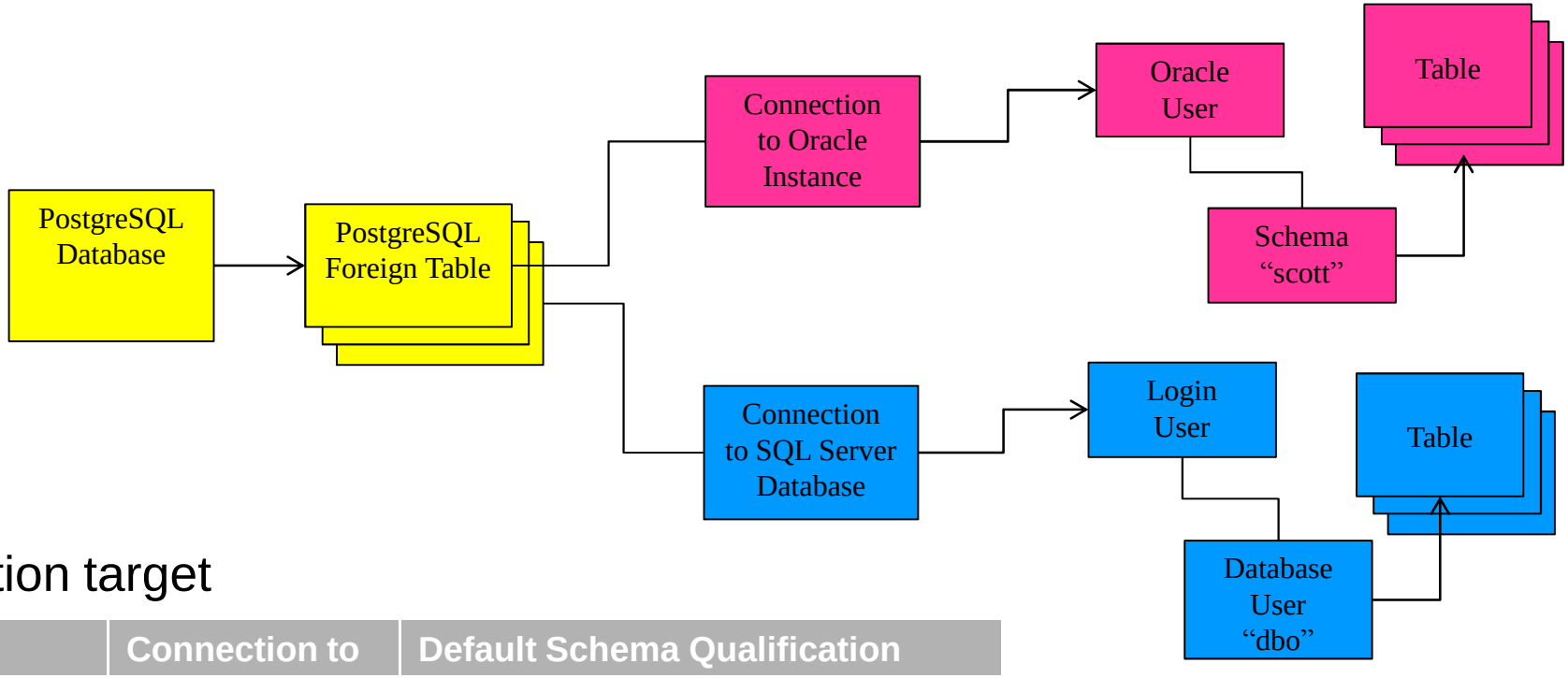
- **User accesses Oracle as if it is a local table of PostgreSQL**



- For SQL Server, FreeTDS is used to connect external SQL Server.

Connection to Various Database

- **Which does PG connect to Database or Instance?**



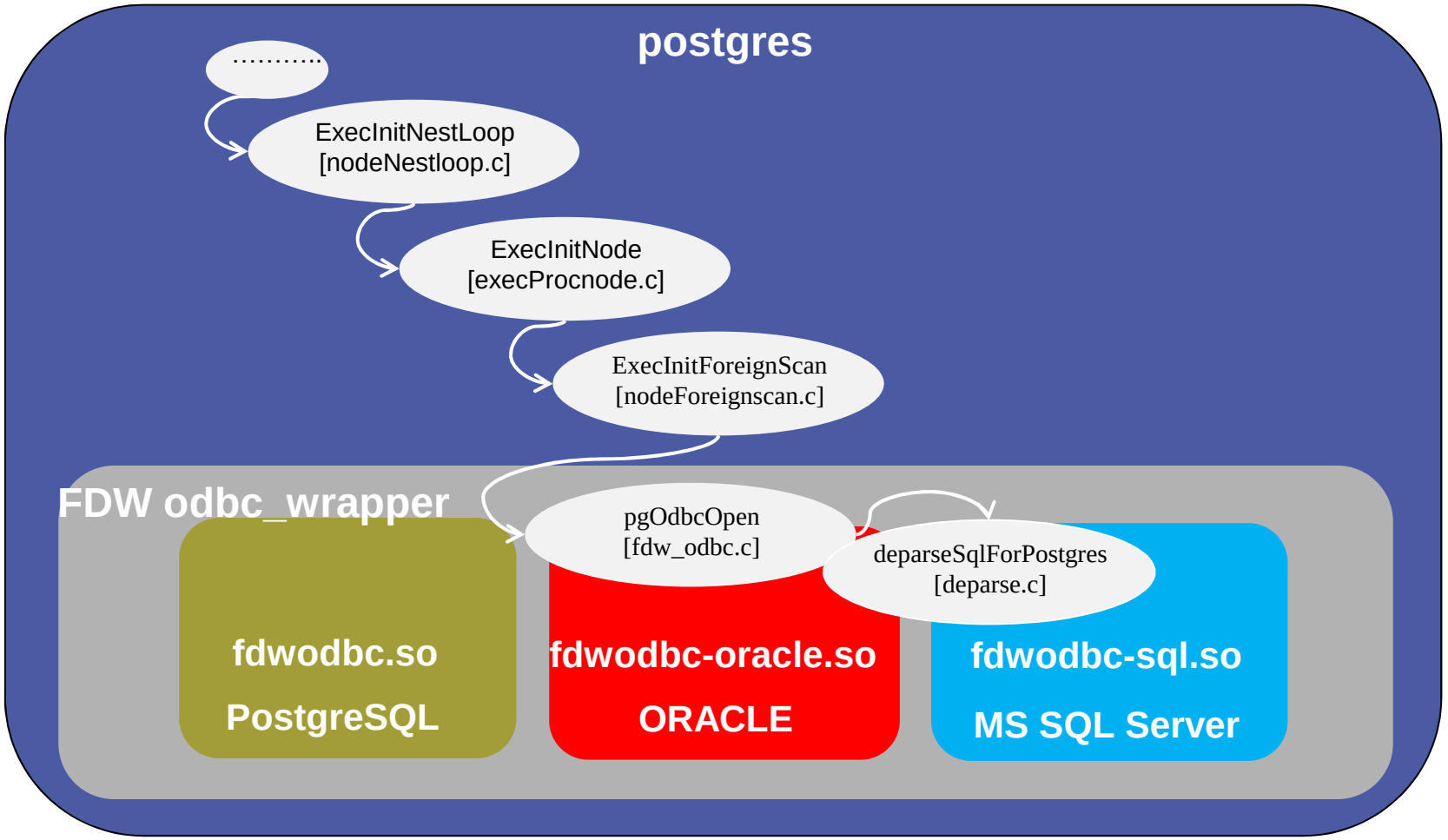
Connection target

Product	Connection to	Default Schema Qualification
PostgreSQL	Database	'Public'
Oracle	Instance	Login user as default schema
MS SQL Server	Database	'dbo' Note - SQL Server login for user has two types: Windows Authentication Mode SQL Server Authentication Mode

SQL Server Database granularity is smaller than concept of Oracle instance.

FDW library for each External Data Source

- **Foreign Data Wrappers for Heterogeneous RDBs**



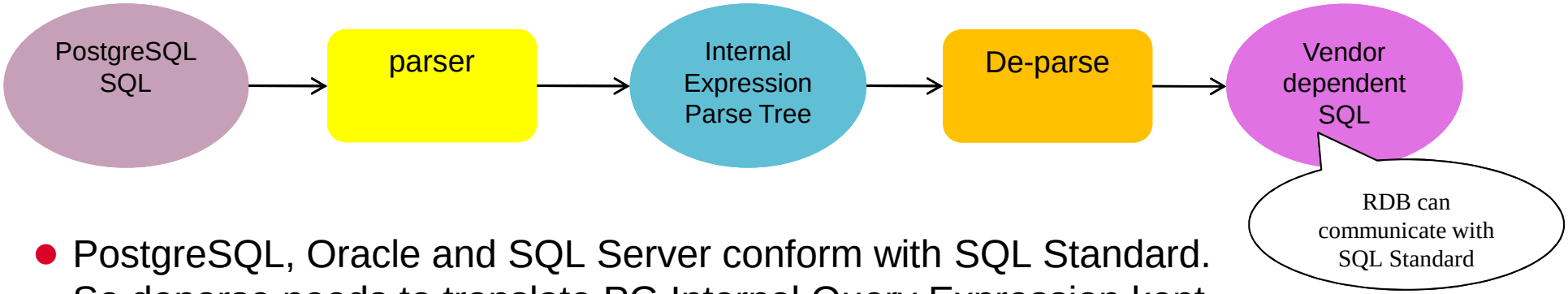
Registration of FDW routines

DDL for ODBC FDW

```
CREATE OR REPLACE FUNCTION fdw_odbc_handler () RETURNS fdw_handler
AS '/usr/local/pgsql-fdw/lib/fdwodbc-oracle', 'pg_odbc_oracle_fdw_handler'
LANGUAGE C STRICT;

CREATE FOREIGN DATA WRAPPER oracle_odbc_wrapper
        HANDLER fdw_odbc_handler
    VALIDATOR postgresql_fdw_validator;

-- create FOREIGN SERVER for remote database 1
CREATE SERVER depvmoracle
        FOREIGN DATA WRAPPER oracle_odbc_wrapper
    OPTIONS (host '10.16.163.184', dbname 'orcl');

CREATE USER MAPPING FOR PUBLIC SERVER depvmoracle
    OPTIONS (user 'dep', password 'dep2010');

CREATE FOREIGN TABLE Product(
 ProductID double precision,
 NAME nchar(50),
 PRODUCTNUMBER nchar(25),
 COLOR nchar(15),
 LISTPRICE numeric(19,4)
) SERVER depvmoracle
OPTIONS(nspname 'dep',relname'product');
```

Consideration of SQL Transparency

- SQL translation is two folds.
  - Parse and De-parse



- PostgreSQL, Oracle and SQL Server conform with SQL Standard. So deparse needs to translate PG Internal Query Expression kept in parse-tree to SQL Standard.
  - Oracle understands Standard SQL Syntax.
    select a.a, b.b from a LEFT OUTER JOIN b ON a.b = b.b;
  - Oracle's unique outer join syntax:
    select a.a, b.b from a, b where a.b = b.b(+);
- It is convenient to utilize PostgreSQL's internal query expression directly or little bit modification.

Example of Deparse Consideration (1)

Some PostgreSQL Internal Query Expression does not communicate with other RDB.

- **Relation name for FROM phrase**
  - PostgreSQL adds name space and default name space is ‘public’.
  - Name space is something like Schema. So, Name space is needed to be mapped to different RDB.
- **WHERE phrase**
  - SQL with WHERE Clause - In condition clause, CAST is added.
  - Oracle, SQL Server does not understand the PostgreSQL CAST operation.
  - Example: SQL Statement
    - Executed SQL: SELECT name FROM person WHERE name = 'aaa';
    - Created SQL : SELECT name FROM person WHERE (name = 'aaa'::text);
  - Generate appropriate SQL
    - SELECT name FROM person WHERE (name = ‘aaa’);
- **LIKE operation**
  - SQL with LIKE Operation - LIKE is converted to ~~
  - Example: SQL Statement
    - Executed SQL: SELECT name FROM person WHERE name LIKE ‘aaa%’;
    - Created SQL: SELECT name FROM person WHERE (name ~~ 'aaa%'::text);
  - Generate appropriate SQL
    - SELECT name FROM person WHERE (name LIKE ‘aaa%’);

Example of Deparse Consideration (2)

- **IN operation**
  - SQL with IN Operation – IN clause is converted into ANY clause
  - Example SQL statement:
    - Executed SQL: select * from person_oracle where id in (1, 2, 3, 4);
    - Created SQL: SELECT person_oracle.id FROM oraclefdw.person_oracle person_oracle WHERE (id = ANY ('{1,2,3,4}'));
    - Generate appropriate SQL: SELECT person_oracle.id FROM oraclefdw.person_oracle person_oracle WHERE (id = ANY (1,2,3,4));
- **Data Type Conversion**
  - ODBC data type as Type Mapping Mediator
    - time stamp with time zone
      - UnixODBC does not support timestamp with time zone
  - Different Data Type has impact on result value.
    - Result data value type unmatch: Data type required by PostgreSQL and Data type returned by Oracle is different.

    e.g. Select count(id) / 3 from person;
    - Oracle returns the float value. But PostgreSQL requires integer and this causes unexpected value error.

Topics

1. Introduction - Federation Database System as Virtual Data Integration Platform
2. **Implementation of Federation Database with PostgreSQL FDW**
   - Foreign Data Wrapper Enhancement
   - **Federated Query Optimization**
3. Use Case of Federation Database

   Example of Use Case and expansion of FDW
4. Result and Conclusion

Federated Query Optimization

- **Federated Query Optimization is Distributed Resource Aware.**
  - Strategy for Query shipment and adequate Pushdown Condition
  - Adequate Join Order for Foreign Table
  - Statistics information of External Data Source

- **Statistics Information for Federated Query Optimization needs more efforts.**

  Statistics information of External Data Source:
  - Database Profile
    - Number of rows
    - Cardinality
    - Record Size
    - Selectivity

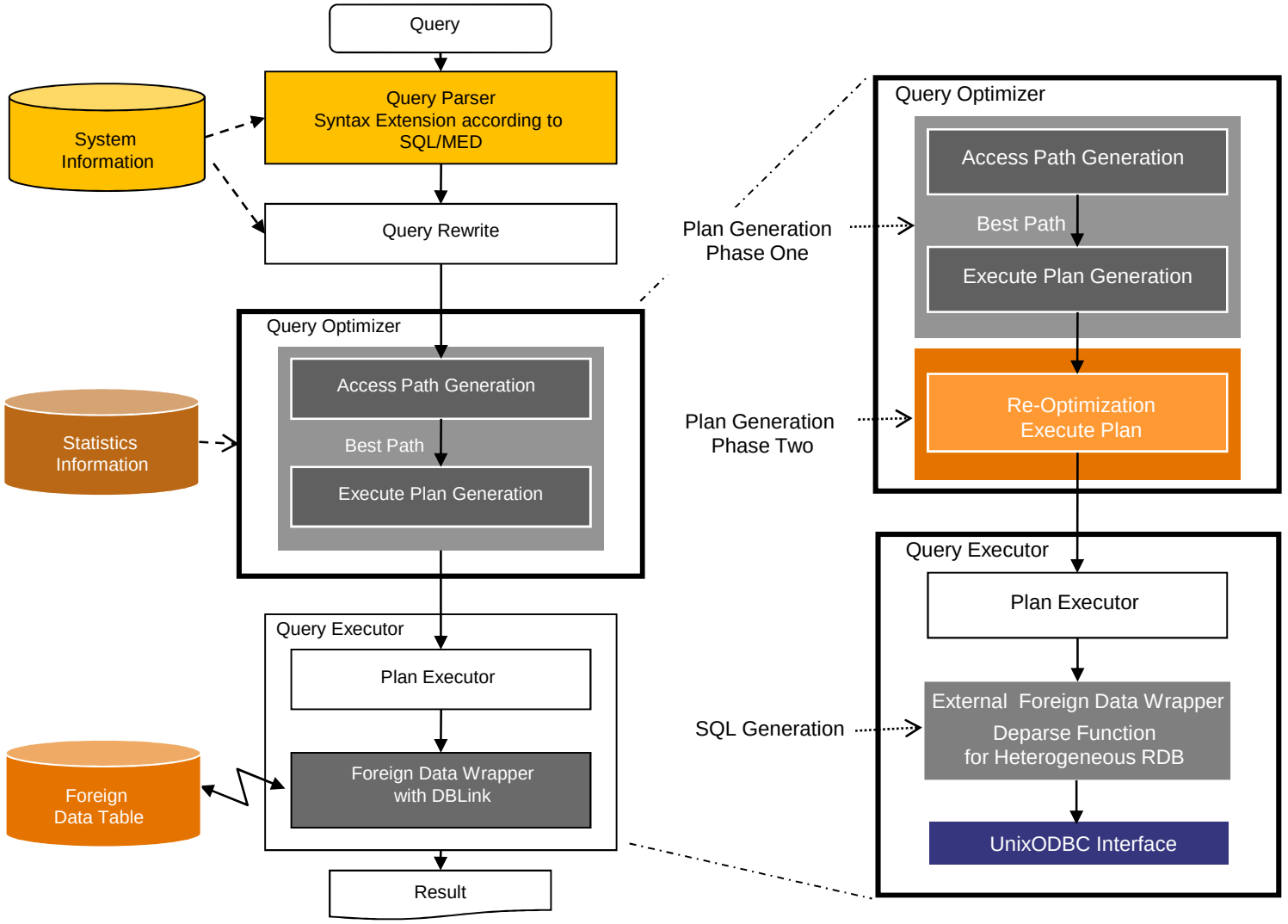
Optimization for Federated Query Processing

- **General Optimization policy of Distributed Query Processing**
  - Response Time Model
    - Pushing down query condition and/or join that reduce selectivity to outside external servers.
    - By calculation, make proper use of cost of join algorithms
  - Resource Cost Model
    - Optimization strategy that considering of ratio of external data source and internal data source
    - Precede the cost and determine optimization

- **Two phase re-optimization method for Response Time Model**
  - Our implementation is based on the response time model
  - Two phase re-optimize method
    - PostgreSQL adopts Heuristic, Dynamic Programming and Cost Base optimization strategy.
      We adopted the way that re-optimize generated plan tree. This approach reduces optimization cost itself. And it reduces changes to existing PostgreSQL optimizer mechanism.

Two phase re-optimize method

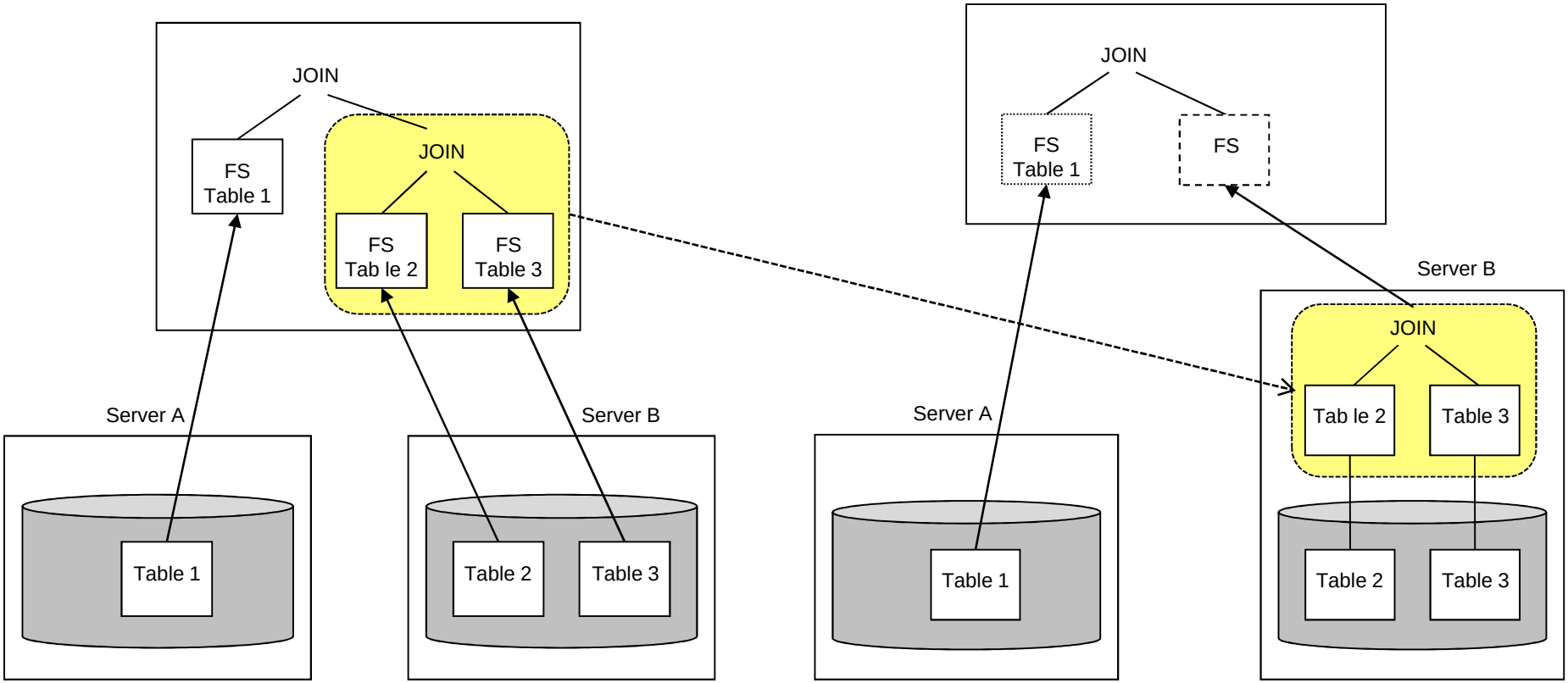
- Reduction of optimization cost itself.
- Reduction of changes of existing PostgreSQL optimizer mechanism.



Optimization Strategy for Response Time Model

A join of two foreign tables where on the same server is pushed down onto the server.

Note - More strictly, we need to know the performance of the RDB on Server B.
If the server B is poor resource to execute the join, the optimizer needs to decide to perform the join task on the local server.



Enhancement of Federated Query Optimization (1)

- **JOIN Optimization**
  - JOIN of two foreign tables in the same server
  - JOIN Cost Sensitive Optimization for Foreign Table
    - Exclusion of Nest Loop Join for Foreign Table

- **Optimize SELECT clause – column sensitive select list**
  - select column1, column2 from foreign_table;

    This case, only column1 and column2 is used for shipped query.
  - As mentioned in PostgreSQL SQL/MED development, the case that a column is only used in order by clause.

- **Condition and Function Pushdown**
  - WHERE clause Pushdown (FDW version default)
  - Functions Pushdown
    - select func1(column1) from foreign_table;
    - select * from foreign_table where func1(column1) > 0;
    - select count(column1) from foreign_table group by func1(column2);

Enhancement of Federated Query Optimization (2)

- **Comparison of SQL for Pushdown**
  - CREATE FOREGIN TABLE ft1
    (col1 int, col2 text, col3 date, col4 timestamp)

Original SQL	[1] select col1 from ft1 [2] select *function*(col2) from ft1 where *function*(col2) > 10 [3] select *func*(col1), *func*(col1), col2, col3, col4 from ft1 [4] select col3 from ft1 where *immutable_function(*col1) = 1 [5] select col3 from ft1 where *non_immutable_function*(col1) = 1 [6] select count(col1) from ft1
Shipped SQL from FDW	[1] select col1, col2, col3, col4 from ft1 [2] select col1, col2, col3, col4 from ft1 where *function*(col2) > 10 [3] select col1, col2, col3, col4 from ft1 [4] select col1, col2, col3, col4 from ft1 where *immutable_function*(col1) = 1 [5] select col1, col2, col3, col4 from ft1 [6] select col1, col2, col3, col4 from ft1
Shipped SQL from Federation Extended FDW	[1] select col1 from ft1 [2] select *function*(col2) from ft1 where *function*(col2) > 10 [3] select *func*(col1), *func*(col1), col2, col3, col4 from ft1 [4] select col3 from ft1 where *immutable_function*(col1) = 1 [5] select col3 from ft1 where *non_immutable_function*(col1) = 1 [6] select count(col1) from ft1

Enhancement of Federated Query Optimization (3)

- **Aggregate Function Pushdown in the same server**
  - COUNT, AVG, SUM, MIN, MAX
  - Example:
    - select sum(column1), avg(column2), count(column3) from foreign_table;
  - Constraint at this implementation:
    COUNT(*) is the exception for Pushdown.
    - select count(*) from foreign_table;

- **GROUP BY Pushdown**
  - Example:
    - select column1 from foreign_table group by column1;
    - select count(column1) from foreign_table group by column1;
  - Constraint at this implementation:
    COUNT(*) is the exception for Pushdown.
    - select count(*) from foreign_table group by column1;

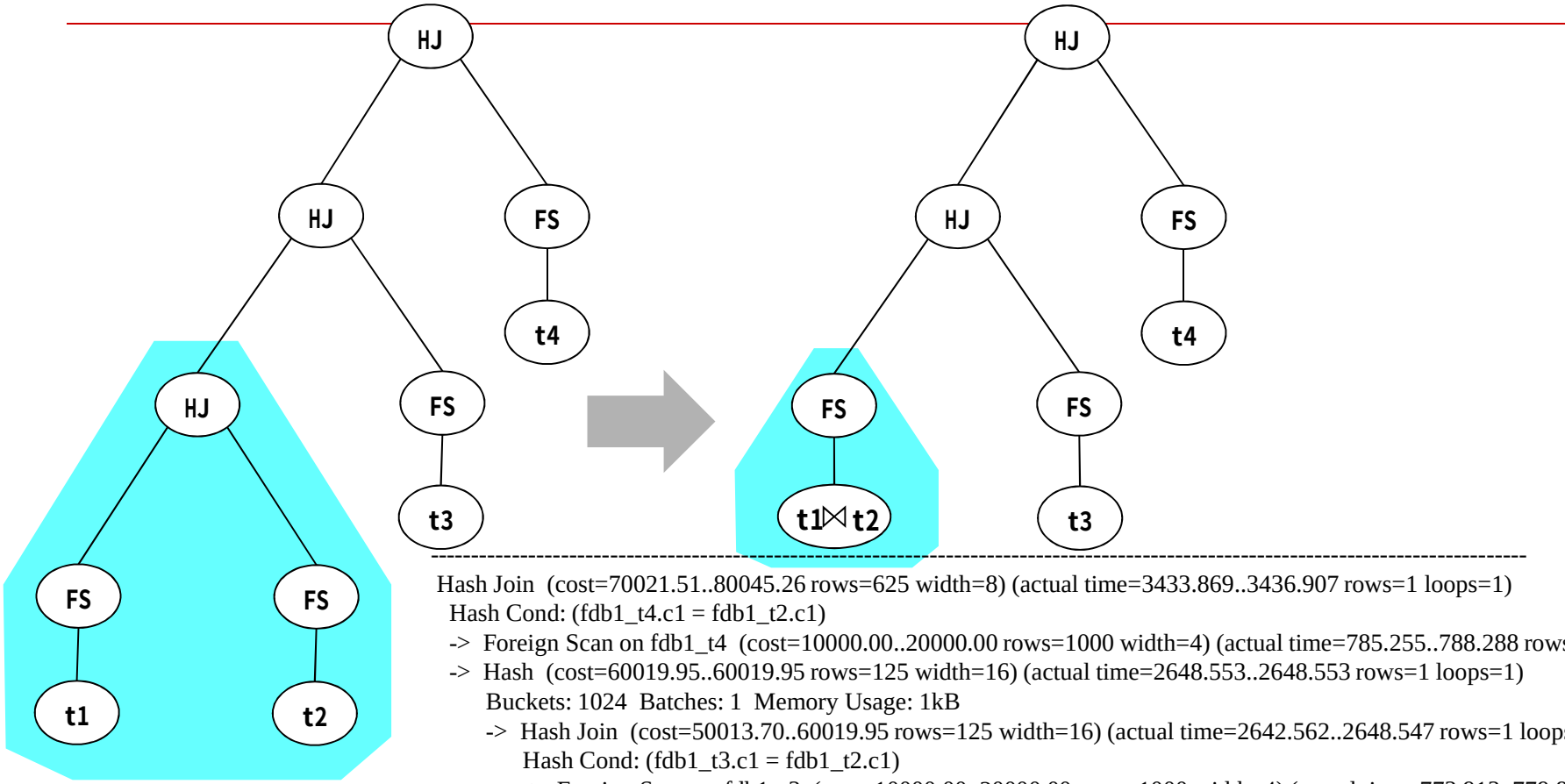
Optimization of Join Plan – Case 1 Left Depth Tree

Contraction of Plan tree

Join of Tables on same Foreign Node:
This case generates Left Depth Three.

```
fdbs=# explain analyze select fdb1_t1.c1, fdb1_t2.c2 from
  (((fdb1_t1 right outer join fdb1_t2 on fdb1_t1.c1=fdb1_t2.c1)
      right outer join fdb1_t3 on fdb1_t2.c1=fdb1_t3.c1)
         right outer join fdb1_t4 on fdb1_t3.c1=fdb1_t4.c1)    where fdb1_t2.c3=1201;
                                        QUERY PLAN

-----------------------------------------------------------------------------------------------------------------------------
 Hash Join  (cost=70021.51..80045.26 rows=625 width=8) (actual time=3433.869..3436.907 rows=1 loops=1)
   Hash Cond: (fdb1_t4.c1 = fdb1_t2.c1)
   ->  Foreign Scan on fdb1_t4  (cost=10000.00..20000.00 rows=1000 width=4) (actual time=785.255..788.288 rows=3 loops=1)
   ->  Hash  (cost=60019.95..60019.95 rows=125 width=16) (actual time=2648.553..2648.553 rows=1 loops=1)
         Buckets: 1024  Batches: 1  Memory Usage: 1kB
         ->  Hash Join  (cost=50013.70..60019.95 rows=125 width=16) (actual time=2642.562..2648.547 rows=1 loops=1)
               Hash Cond: (fdb1_t3.c1 = fdb1_t2.c1)
               ->  Foreign Scan on fdb1_t3  (cost=10000.00..20000.00 rows=1000 width=4) (actual time=773.913..779.892 rows=3 loops=1)
               ->  Hash  (cost=20000.00..20000.00 rows=1000 width=12) (actual time=1868.618..1868.618 rows=1 loops=1)
                     Buckets: 1024  Batches: 1  Memory Usage: 1kB
                     ->  Foreign Scan  (cost=10000.00..20000.00 rows=1000 width=12) (actual time=1866.665..1868.604 rows=1 loops=1)
                           ->  Foreign Scan on fdb1_t2  (cost=10000.00..20000.00 rows=1000 width=8) (never executed)
                                 Filter: (c3 = 1201)
                           ->  Foreign Scan on fdb1_t1  (cost=10000.00..20000.00 rows=1000 width=4) (never executed)
 Total runtime: 3469.116 ms
(15 rows)

fdbs=#
```

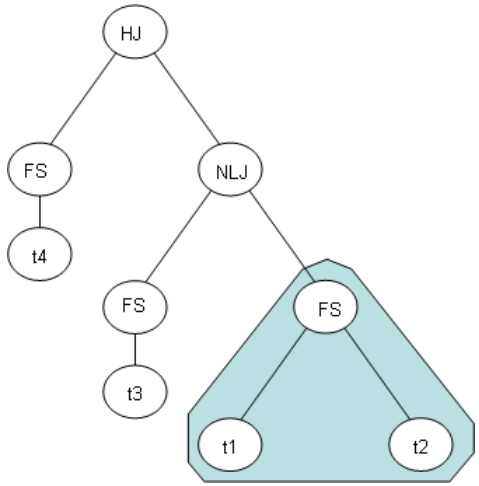
Contraction of Plan tree




```
Hash Join  (cost=70021.51..80045.26 rows=625 width=8) (actual time=3433.869..3436.907 rows=1 loops=1)
  Hash Cond: (fdb1_t4.c1 = fdb1_t2.c1)
  ->  Foreign Scan on fdb1_t4  (cost=10000.00..20000.00 rows=1000 width=4) (actual time=785.255..788.288 rows=3 loops=1)
  ->  Hash  (cost=60019.95..60019.95 rows=125 width=16) (actual time=2648.553..2648.553 rows=1 loops=1)
        Buckets: 1024  Batches: 1  Memory Usage: 1kB
        ->  Hash Join  (cost=50013.70..60019.95 rows=125 width=16) (actual time=2642.562..2648.547 rows=1 loops=1)
              Hash Cond: (fdb1_t3.c1 = fdb1_t2.c1)
              ->  Foreign Scan on fdb1_t3  (cost=10000.00..20000.00 rows=1000 width=4) (actual time=773.913..779.892 rows=3 loops=1)
              ->  Hash  (cost=20000.00..20000.00 rows=1000 width=12) (actual time=1868.618..1868.618 rows=1 loops=1)
                    Buckets: 1024  Batches: 1  Memory Usage: 1kB
                    ->  Foreign Scan  (cost=10000.00..20000.00 rows=1000 width=12) (actual time=1866.665..1868.604 rows=1 loops=1)
                          ->  Foreign Scan on fdb1_t2  (cost=10000.00..20000.00 rows=1000 width=8) (never executed)
                                Filter: (c3 = 1201)
                          ->  Foreign Scan on fdb1_t1  (cost=10000.00..20000.00 rows=1000 width=4) (never executed)
Total runtime: 3469.116 ms
(15 rows)
```

Optimization of Join Plan – Case 2 Right Depth Tree

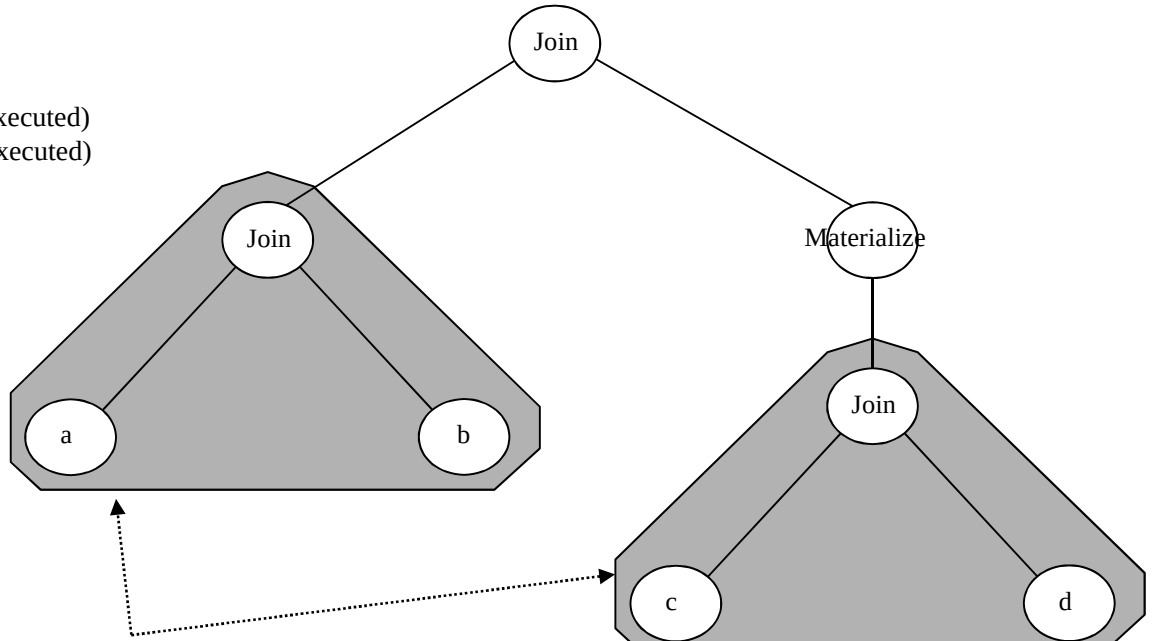
This case generates right depth tree.




```
yndb1=# explain analyze select fdb1_t1.c1, fdb1_t2.c2
from (((fdb1_t1 inner join fdb1_t2 on fdb1_t1.c1=fdb1_t2.c1)
    inner join fdb1_t3 on fdb1_t2.c1=fdb1_t3.c1)
          inner join fdb1_t4 on fdb1_t3.c1=fdb1_t4.c1)
where fdb1_t2.c3=1201 ;
                                    QUERY PLAN
-------------------------------------------------------------------------------------------
-------------------------------------------
 Hash Join  (cost=50012.50..80487.20 rows=625 width=8) (actual time=1235.267..1238.669 rows=1 loops=1)
   Hash Cond: (fdb1_t1.c1 = fdb1_t4.c1)
   ->  Nested Loop  (cost=30000.00..60452.51 rows=125 width=16) (actual time=1017.518..1020.916 rows=1 loops=1)
         Join Filter: (fdb1_t1.c1 = fdb1_t3.c1)
         ->  Foreign Scan  (cost=10000.00..20000.00 rows=1000 width=12) (actual time=810.709..811.589 rows=1 loops=1)
               ->  Foreign Scan on fdb1_t1  (cost=10000.00..20000.00 rows=1000 width=4) (never executed)
               ->  Foreign Scan on fdb1_t2  (cost=10000.00..20000.00 rows=1000 width=8) (never executed)
                     Filter: (c3 = 1201)
         ->  Materialize  (cost=10000.00..20005.00 rows=1000 width=4) (actual time=206.788..209.297 rows=3 loops=1)
               ->  Foreign Scan on fdb1_t3  (cost=10000.00..20000.00 rows=1000 width=4) (actual time=206.751..209.251 rows=3 loops=1)
   ->  Hash  (cost=20000.00..20000.00 rows=1000 width=4) (actual time=217.664..217.664 rows=3 loops=1)
         Buckets: 1024  Batches: 1  Memory Usage: 1kB
         ->  Foreign Scan on fdb1_t4  (cost=10000.00..20000.00 rows=1000 width=4) (actualtime=216.757..217.649 rows=3 loops=1)
 Total runtime: 1313.361 ms
(14 rows)
```

Optimization of Join Plan – Case 3 Bushy Tree

```
select * from
 ( select a.id from person_test a, person_test b where a.name = b.name ) e,
 ( select d.id from person_test c, person_test d where c.name = d.name ) f
 where e.id = f.id and e.id = 3;

Nested Loop
  ->  Foreign Scan
      ->  Foreign Scan on person_test b (never executed)
      ->  Foreign Scan on person_test a (never executed)
          Filter: (id = 3)
  ->  Materialize
      ->  Foreign Scan
          ->  Foreign Scan on person_test c (never executed)
          ->  Foreign Scan on person_test d (never executed)
              Filter: (id = 3)
 Total runtime: 46.800 ms
(11 rows)
```



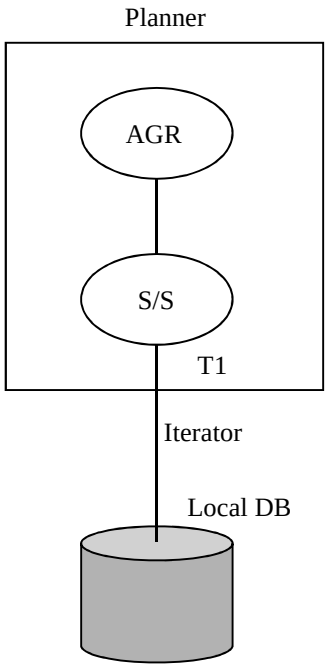
Well-written Join Plans are pushed down for each external server.

Optimization of Aggregate Function Execution Plan (1)

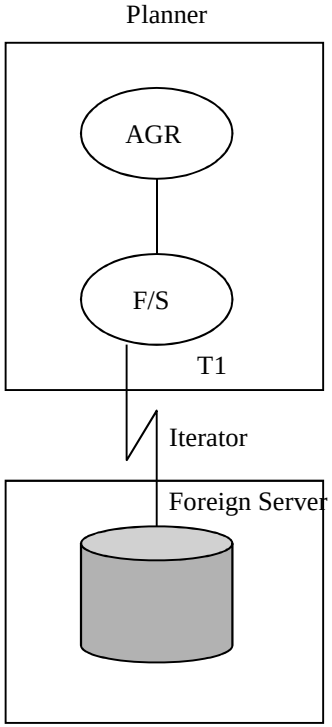
Consider Optimization on Aggregate Function: AVG,COUNT,MAX,MIN,SUM

Example: Select AVG(c1) from t1 where c2 > 10000 ;

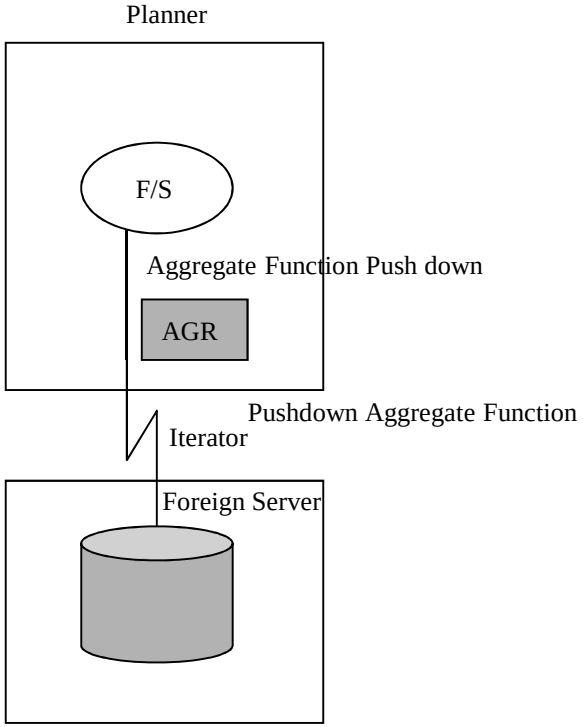
Local Execute Plan for
Aggregate Function on Local Table



Execute Plan for
Aggregate Function on Foreign Table



Federated Database Version
Execute Plan for
Aggregate Function on Foreign Table



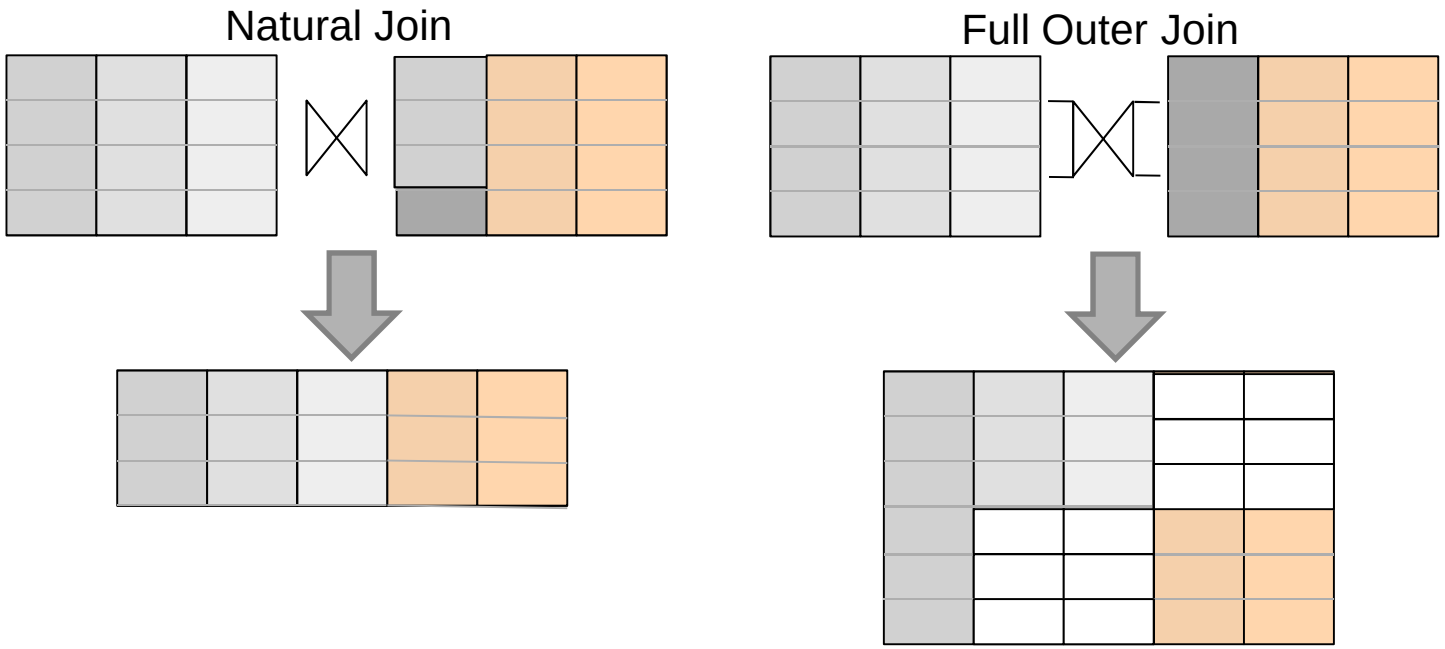
Optimization of Aggregate Function Execution Plan (2)

select avg(a.aid + b.bid + t.bid) from pgbench_accounts a, pgbench_branches b, pgbench_tellers t ;

```
adbs=# explain analyze select avg( a.aid + b.bid + t.bid )  from pgbench_accounts a, pgbench_branches b, pgbench_tellers t ;
                                  QUERY PLAN
-------------------------------------------------------------------------------------------
 Aggregate  (cost=15072505.00..15072505.02 rows=1 width=12) (actual time=32464.872..32464.
873 rows=1 loops=1)
   ->  Nested Loop  (cost=30000.00..12572505.00 rows=1000000000 width=12) (actual time=17986.549..31545.068 rows=1000000 loops=1)
         ->  Foreign Scan  (cost=10000.00..20000.00 rows=1000 width=8) (actual time=17780.458..29354.627 rows=100000 loops=1)
               ->  Foreign Scan on pgbench_accounts a  (cost=10000.00..20000.00 rows=1000width=4) (never executed)
               ->  Foreign Scan on pgbench_branches b  (cost=10000.00..20000.00 rows=1000width=4) (never executed)
         ->  Materialize  (cost=10000.00..20005.00 rows=1000 width=4) (actual time=0.003..0.008 rows=10 loops=100000)
               ->  Foreign Scan on pgbench_tellers t  (cost=10000.00..20000.00 rows=1000 width=4) (actual time=206.072..206.826 rows=10 loops=1)
Total runtime: 33246.390 ms
(8 rows)

adbs=#
```

OUTER JOIN Optimization (1)

- **Shipping of join of foreign tables on the same RDB**
  - This strategy has advantage if the result of joined relation is smaller than that of the local PostgreSQL.
  - Outer join case, the result relation size may be larger than inner join case. In this case, data transfer cost may be higher than nojoin query shipment strategy.



OUTER JOIN Optimization (2)

- **Tentative implementation for OUTER JOIN sensitive optimization**

INNER JOIN CASE

```
fdbs=# explain analyze select * from fdb1_t1 t1 join fdb1_t2 t2 on t1.c2 = t2.c2 ;
                                                    QUERY PLAN
-------------------------------------------------------------------------------------------------------------------
 Foreign Scan  (cost=10000.00..20000.00 rows=1000 width=24) (actual time=879.346..880.566rows=3 loops=1)
   ->  Foreign Scan on fdb1_t1 t1  (cost=10000.00..20000.00 rows=1000 width=12) (never executed)
   ->  Foreign Scan on fdb1_t2 t2  (cost=10000.00..20000.00 rows=1000 width=12) (never executed)
 Total runtime: 884.291 ms
(4 rows)
```

FULL OUTER JOIN CASE

```
fdbs=# explain analyze select * from fdb1_t1 t1 full outer join fdb1_t2 t2 on t1.c2 = t2.c2 ;
                                                                QUERY PLAN
-------------------------------------------------------------------------------------------------------------------------------------------
 Merge Full Join  (cost=40099.66..40179.66 rows=5000 width=24) (actual time=546.655..546.701 rows=10 loops=1)
   Merge Cond: (t1.c2 = t2.c2)
   ->  Sort  (cost=20049.83..20052.33 rows=1000 width=12) (actual time=263.716..263.730 rows=10 loops=1)
         Sort Key: t1.c2
         Sort Method:  quicksort  Memory: 17kB
         ->  Foreign Scan on fdb1_t1 t1  (cost=10000.00..20000.00 rows=1000 width=12) (actual time=259.268..263.684 rows=10 loops=1)
   ->  Sort  (cost=20049.83..20052.33 rows=1000 width=12) (actual time=282.920..282.924 rows=3 loops=1)
         Sort Key: t2.c2
         Sort Method:  quicksort  Memory: 17kB
         ->  Foreign Scan on fdb1_t2 t2  (cost=10000.00..20000.00 rows=1000 width=12) (actual time=281.522..282.893 rows=3 loops=1)
 Total runtime: 552.998 ms
(11 rows)
```

Remaining Issues (1)

- **Data translation from Heterogeneous RDBs**
  - Translation of data type and precision
- **Generalization for Join pushdown**
  - Current our tentative implementation works for one JOIN of two foreign tables of leaf nodes.
    - JOINs of two foreign tables in the same server
- **Constraint Mapping Specification**
  - Constraints in SQL
    - NOT NULL
    - PRIMARY KEY, UNIQUE
    - REFERENCES
  - INSERT/UPDATE/DELETE is not supported now. So these are not critical concerns.

Remaining Issues (2)

- **Statistics Information for External Data Source performance cost**
  - External Database Profile
    - Cardinality: the number of distinct value in Relation
    - Size of attribute, seize of record
    - Selectivity for each attribute
      - the number of distinct values appearing in Relation, max and min
    - How to correct these information from external servers?
  - INDEX information
    - CREATE FOREIGN INDEX is needed or not.
    - CREATE FOREIGN INDEX is not SQL Standard in SQL/MED. But some Federation Database Products support CREATE FOREIGN INDEX.
    - This information may be considered to be utilized in the optimizer for foreign data access.
  - Performance information of External DBMS
    - External DB Server performance and cost information
      - DB Server Performance Ration
        - DBMS characteristic
        - Hardware Resource characteristic

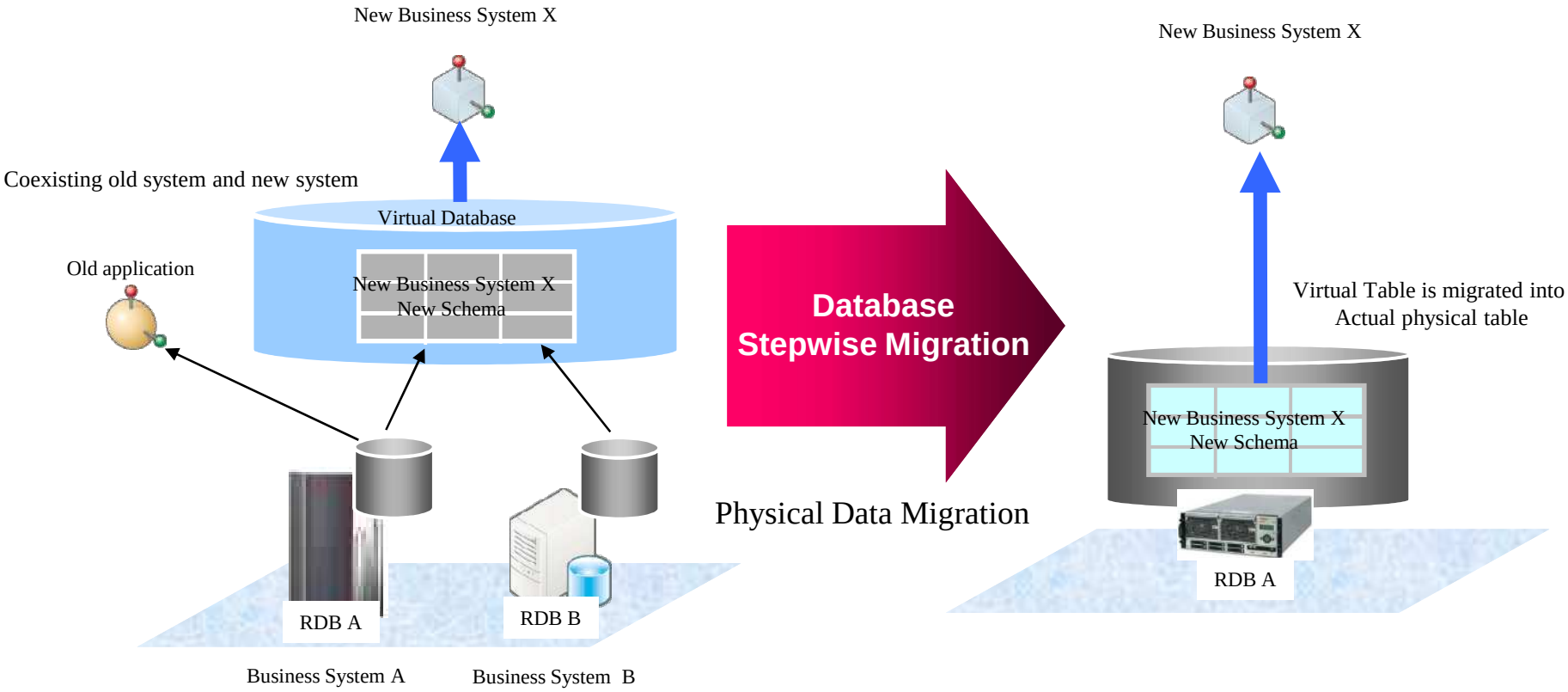
Topics

1. Introduction - Federation Database System as Virtual Data Integration Platform
2. Implementation of Federation Database with PostgreSQL FDW
   - Foreign Data Wrapper Enhancement
   - Federated Query Optimization
3. **Use Case of Federation Database**

   Example of Use Case and expansion of FDW
4. Result and Conclusion

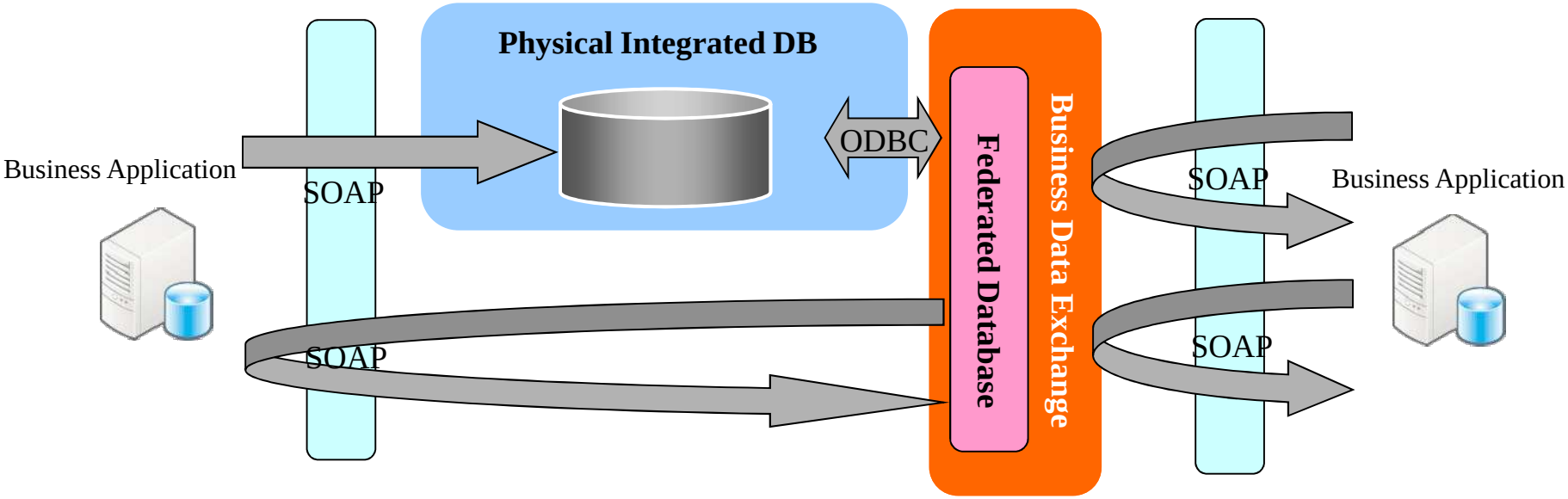
Use Case of Federation Database (1)

- **Data Migration Solution**
  - Prototyping is possible on new system
  - Testing application without actual data migration



Use Case of Federation Database (2)

- **Real time Data Access Service for Physical Data Integration**



- **Another Use Case**
  - Data Ware House Extension
  - Cloud Data Integration

Result and Conclusion

- **PostgreSQL FDW for Federation Database System**
  - Federation Database is good solution for virtual Information Integration. PostgreSQL FDW is valuable to realize Federation Database.
- **Federated Query Optimization challenges**
  - Enhancement of PostgreSQL optimization mechanism for Federated Query Optimization
- **Future Challenges**
  - Improve Optimization Plan for Foreign Table
  - Data Update
  - Extend Foreign Data Wrapper for Various Data Source
    - XML Data
    - Web Service

- **Questions?**