- Ashutosh Bapat, Rushabh Lathia | 2016.05.17

FDWs – how to use and write
Tutorial @ PGCon 2016

postgres_fdw demo

Create Server options

- All libpq connection options except
  - User, password – user mapping options
  - client_encoding – set to local server's encoding
- Some relevant options
  - host/hostaddr: name/location of foreign server
  - port: port number
  - dbname: database name to connect to
  - sslmode and other SSL related options

Create server options

- Planner cost options
  - fdw_startup_cost: represents the cost of establishing connection, parsing and planning query
  - fdw_tuple_cost: represents the cost of transferring data per tuple
  - use_remote_estimate: use EXPLAIN to get the cost of executing query on the foreign server
- fetch_size: number of tuples to get in each fetch operation
- extensions: list of matching extensions available on the foreign server

Create user mapping options

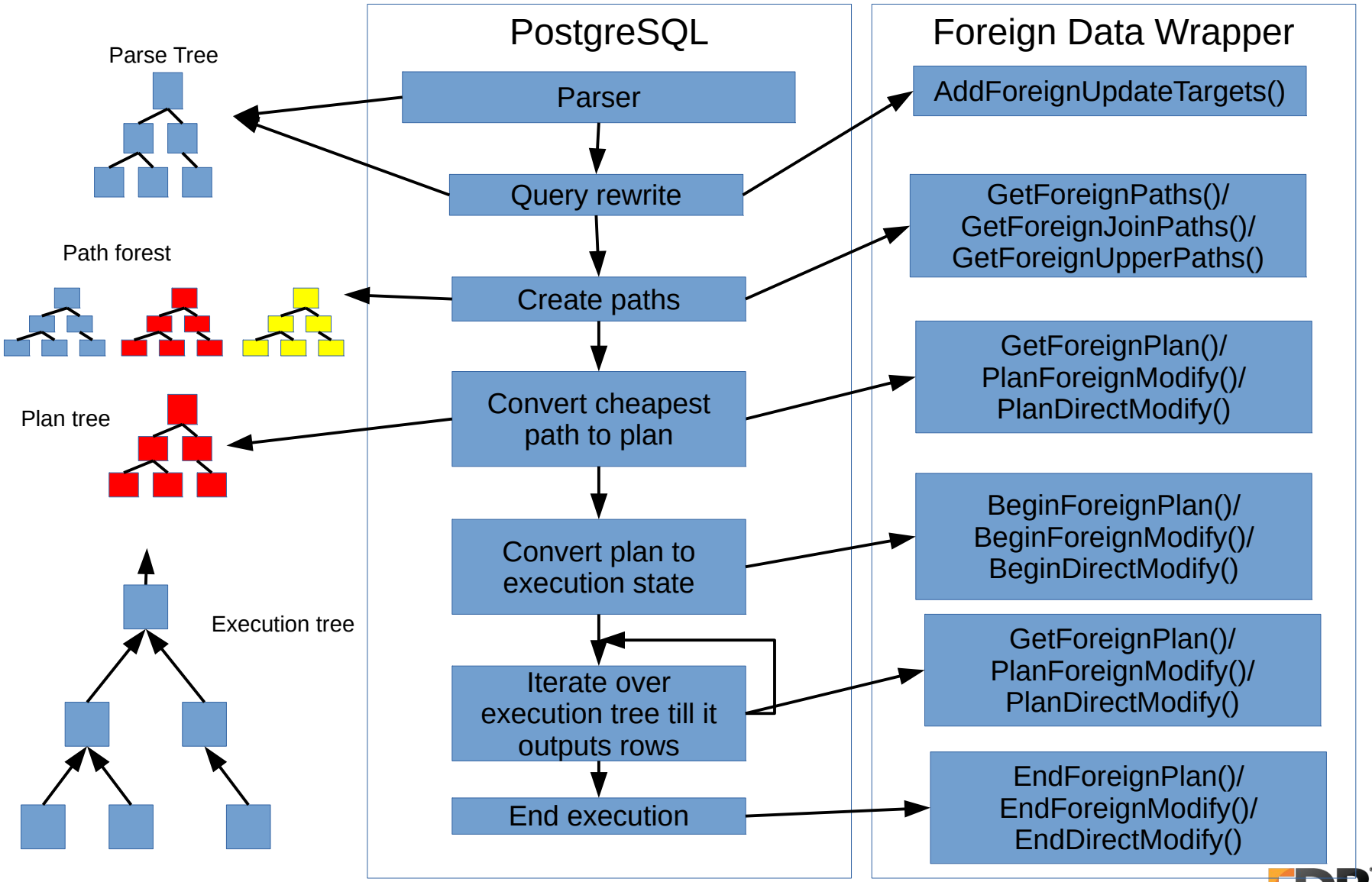
- user: foreign server user name to connect as
- password: password to be used for the user

Create foreign table options

- schema_name: schema on the foreign server in which foreign table is located
- table_name: name of the table on the foreign server
- column_name: name of the column on the foreign server
- use_remote_estimate, fetch_size: similar to the foreign server

Query processing in PostgreSQL

FDW and query execution



Node

- Basic block of any tree structure in PostgreSQL
- Broad types
  - Parse nodes – appear in parse trees
  - Expression nodes – appear everywhere to represent various expressions
  - Plan nodes – appear in plan tree
  - Execution state nodes – appear in execution tree, hold the current execution state of node

Path and path cost

- Each operation in a query can be realized in multiple ways
  - Joins: hash, merge, nested loop
- Each method is represented as a path
- Path
  - A light-weight plan
  - Estimated cost of path models execution time
  - Startup cost: cost expended before fetching any tuples
  - Total cost: startup cost + cost for fetching all tuples

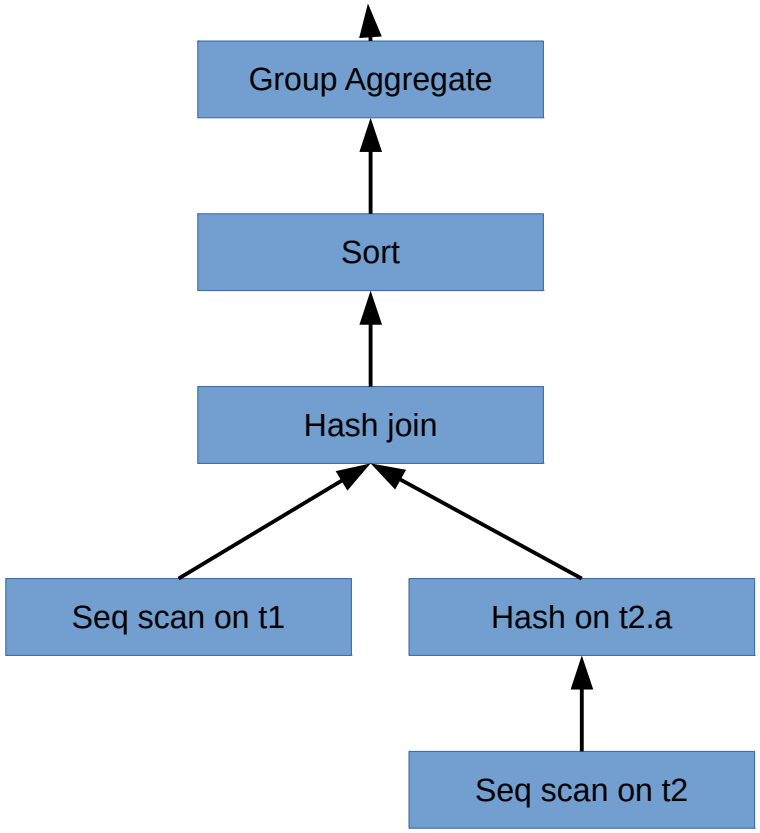
Relations: unit of query result

- Query: ordered set of (SQL) operations
- Relation: result of each operation
  - Result of scanning a table
  - Result of joins, grouping, limit etc.
- RelOptInfo
  - Represents results of various operations in query
  - Represents result of a node in plan/execution tree.
  - Holds all the paths for realizing that result
  - fdw_private member for FDW specific information
    - E.g. file_fdw stores path of file on the disk

Example

explain verbose select count(*), t1.a from t1, t2 where t1.a = t2.a group by t1.a, t1.b + t2.b order by t1.b + t2.b;

```
QUERY PLAN
------------------------------------------------------------------------------
----------
 GroupAggregate  (cost=45.13..46.66 rows=68
width=16)
   Output: count(*), t1.a, ((t1.b + t2.b))
   Group Key: ((t1.b + t2.b)), t1.a
   ->  Sort  (cost=45.13..45.30 rows=68 width=8)
         Output: t1.a, ((t1.b + t2.b))
         Sort Key: ((t1.b + t2.b)), t1.a
         ->  Hash Join  (cost=1.14..43.06 rows=68 width=8)
               Output: t1.a, (t1.b + t2.b)
               Hash Cond: (t1.a = t2.a)
               ->  Seq Scan on frgn_schema.t1
(cost=0.00..32.60 rows=2260 width=8)
                     Output: t1.a, t1.b
               ->  Hash  (cost=1.06..1.06 rows=6 width=8)
                     Output: t2.b, t2.a
                     ->  Seq Scan on frgn_schema.t2
(cost=0.00..1.06 rows=6 width=8)
                           Output: t2.b, t2.a
```



Writing a foreign data wrapper

blackhole_fdw – a great way to start

- Accepts everything and returns nothing
- Skeleton template for writing a new FDW
- Available at https://bitbucket.org/adunstan/blackhole_fdw/src
- Heavily annotated code
  - Author doesn't need to refer to documentation
  - Ready to use extension files

FDW handler and validator

- Handler function
  - Returns a structure of function pointers
  - Function pointers implement FDW APIs
- Validator function
  - Validates options given to CREATE/ALTER commands
  - Input: array of options with values, type of object (server, table, user mapping)
  - Returns nothing, should throw error on encountering an invalid option

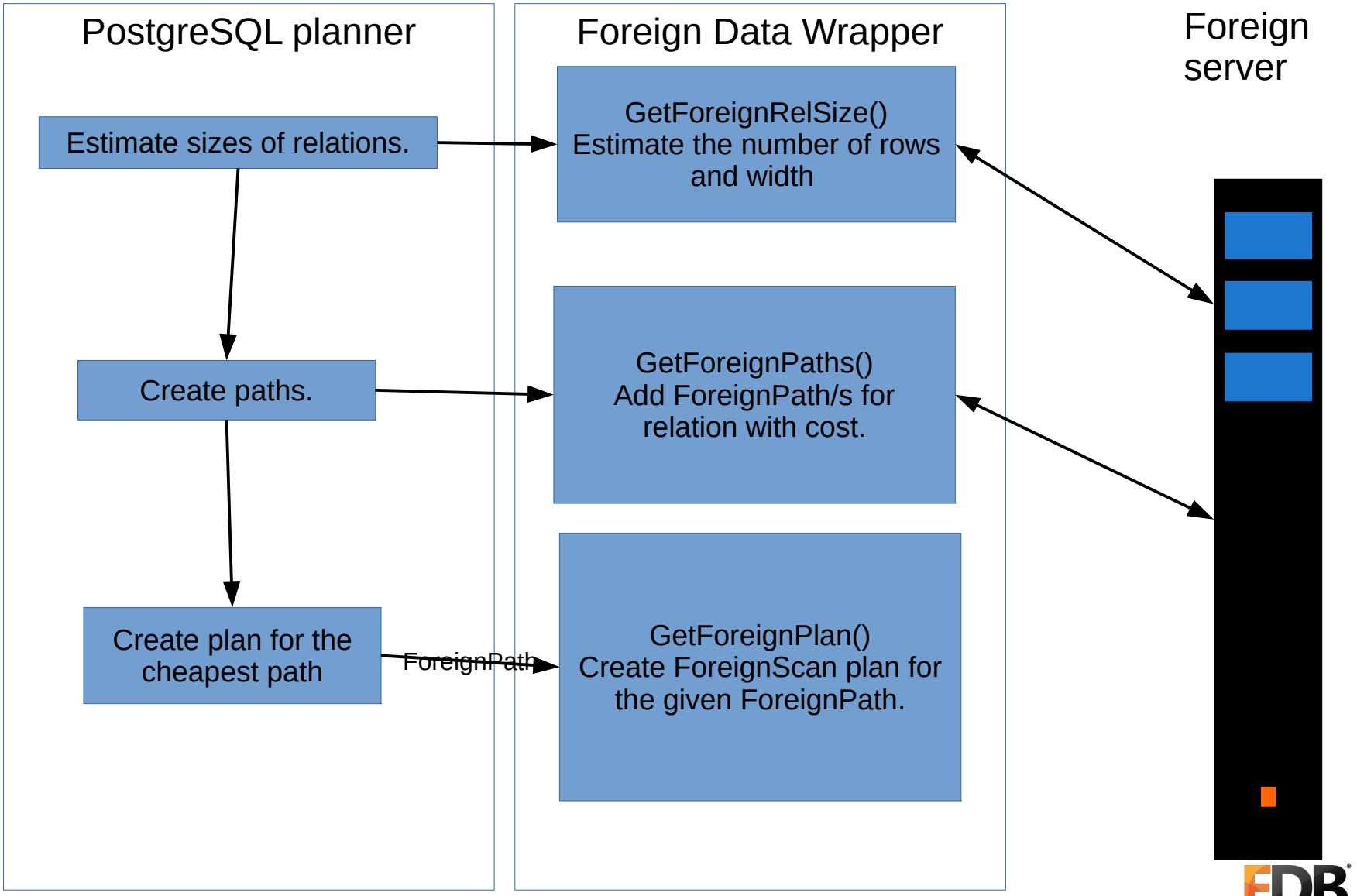
Pushing down operations

- FDWs aim at delegating or pushing down operations to the foreign server
- What can be pushed down (as of 9.6)
  - Expressions in SELECT clause
  - Conditions in WHERE, ON, HAVING clauses
  - Joins
  - Sorting
  - aggregates, grouping
  - Limit

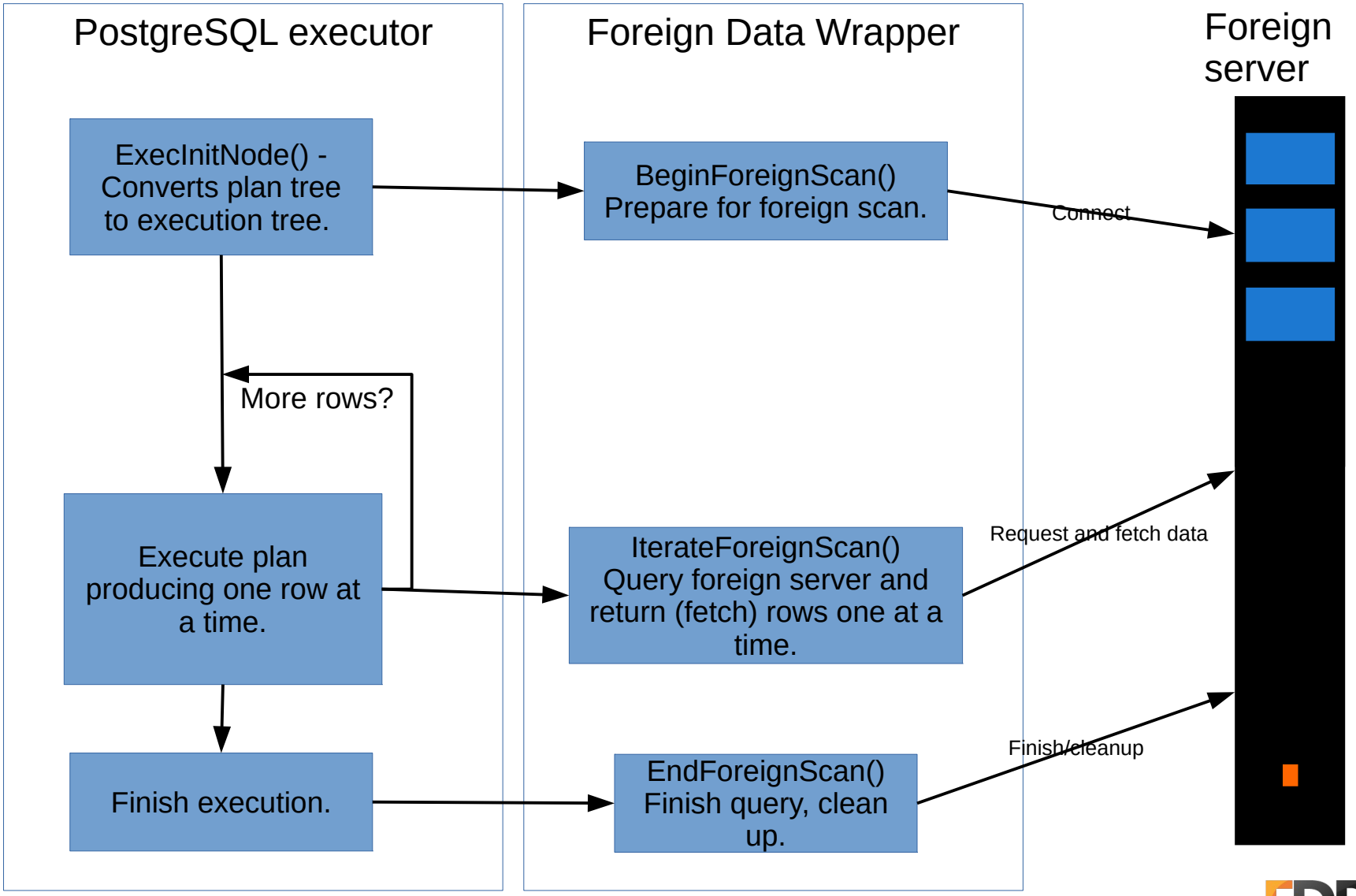
Push-down consideration

- Push-down safety
  - Can foreign server evaluate a construct?
    - Joins may not be evaluated by file_fdw
  - Evaluating construct on the foreign server should produce same result as local server
    - now(): unsafe
    - arithmetic, logical operations on integers: safe
- Pushdown efficiency
  - Is evaluation at foreign server going to improve performance?

Writing simple table scanner: planning



Writing simple table scanner: execution



Writing simple table scanner

- GetForeignPaths()
  - Calculate the cost of scanning the relation
  - Startup cost: cost for connecting to foreign server, querying etc.
  - Total cost: cost of fetching all the tuples from the foreign server
  - Create path using create_foreignscan_path()
  - Store the path using add_path()

Writing simple table scanner

- GetForeignPlan()
  - Inputs: previously created path, targetlist, restriction clauses etc.
  - Segregate the restriction clauses, target list entries into shippable, non-shippable items
  - Construct query/code to fetch the required data from the foreign server
  - Create ForeignScan node using make_foreignscan().

file_fdw executor using COPY protocol

- fileBeginForeignScan()
  - Calls BeginCopyFrom() with filename and foreign table options
  - Opens file, reads header if any
  - Sets up data type input functions
- fileEndForeignScan()
  - Calls EndCopyFrom()
  - Closes file
- fileRescanForeignScan()
  - EndCopyFrom(); BeginCopyFrom()

file_fdw: per row data conversion

- fileIterateForeignScan()
  - Calls NextCopyFrom()
  - Reads next record from file
  - Separates data column-wise using deliminator
  - For every column, converts input data to PostgreSQL data format using data type input functions
    - e.g. date_in() for text or date_receive() for binary

mongo_fdw in nutshell

```
CREATE FOREIGN TABLE warehouse(
        _id NAME,
        warehouse_id int,
         warehouse_name text,
        warehouse_created timestamptz))
SERVER mongo_server
        OPTIONS (database 'db', collection 'warehouse');

SELECT * FROM warehouse WHERE warehouse_id = 1;
```

Mongodb query: db.warehouse.find({"warehouse_id" : 1})

```
db.warehouse.find({"warehouse_id" : 1}).pretty()
{
    "_id" : ObjectId("53720b1904864dc1f5a571a0"),
    "warehouse_id" : 1,
    "warehouse_name" : "UPS",
    "warehouse_created" : ISODate("2014-12-12T07:12:10Z")
}
```

```
           _id           | warehouse_id | warehouse_name |     warehouse_created
-------------------------+--------------+----------------+---------------------------
53720b1904864dc1f5a571a0|            1 | UPS            | 12-DEC-14 12:12:10 +05:00
```

mongo_fdw: scanning a simple table

- MongoBeginForeignScan
  - Open connection to mongodb - MongoConnect()
  - Create mongo cursor – MongoCursorCreate()
- MongoRescanForeignScan
  - Close running cursor: MongoCursorDestroy()
  - Reopen it – MongoCursorCreate()
- MongoEndForeignScan
  - Close running cursor: MongoCursorDestroy().

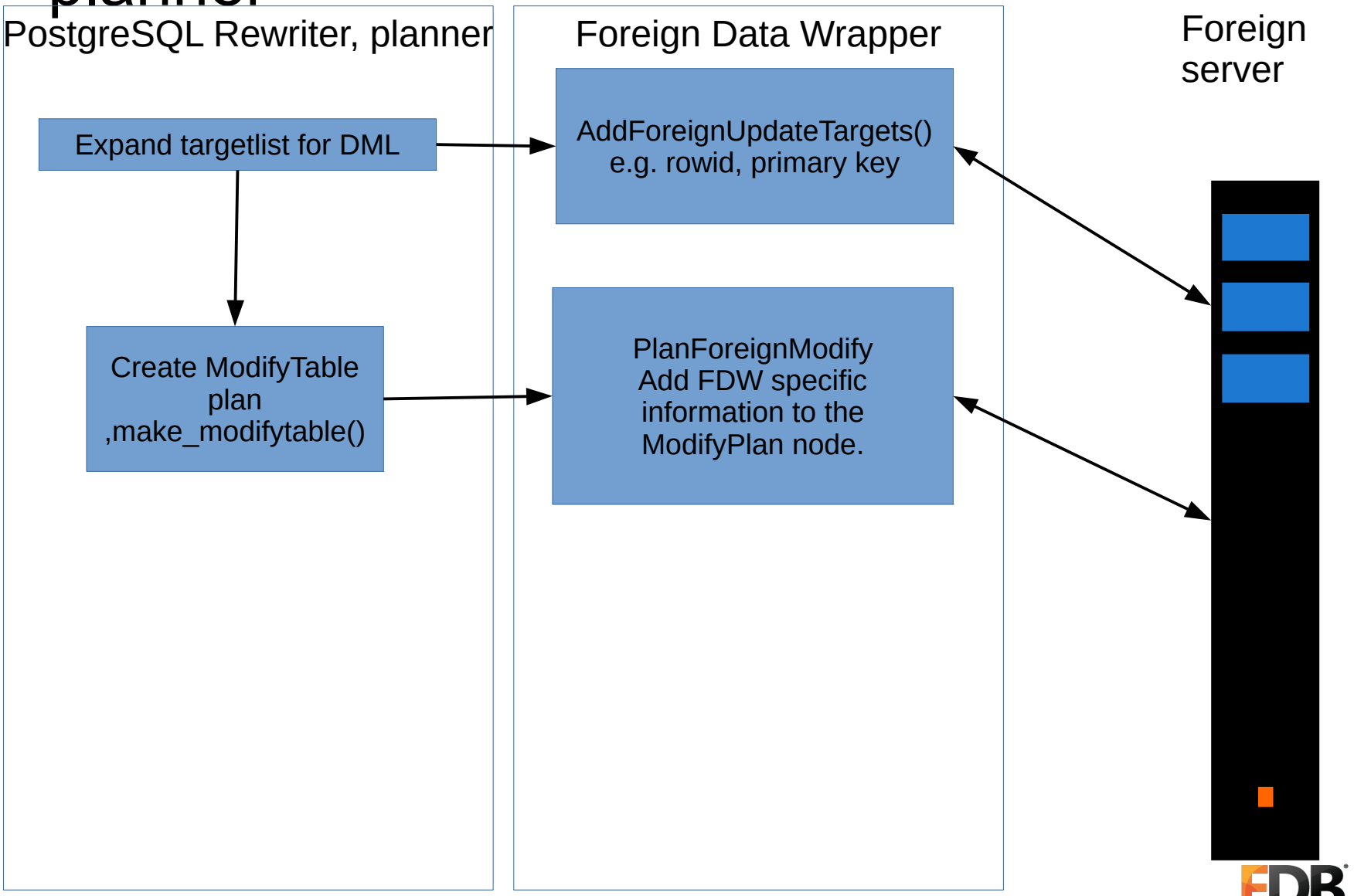
mongo_fdw: scanning a simple table

- MongoIterateForeignScan
  - Fetch next record – MongoCursorNext()
  - Fetch columns from record by iterating over the contents of record using MongoCursorBson(), BsonIterInit(), BsonIterNext().
  - Fetch column value using BsonIter<Type>
    - e.g. BsonIterInt(), BsonIterDouble(),
  - Convert to PostgreSQL using <Type>GetDatum() calls
    - e.g. Int32GetDatum(), Float4GetDatum()

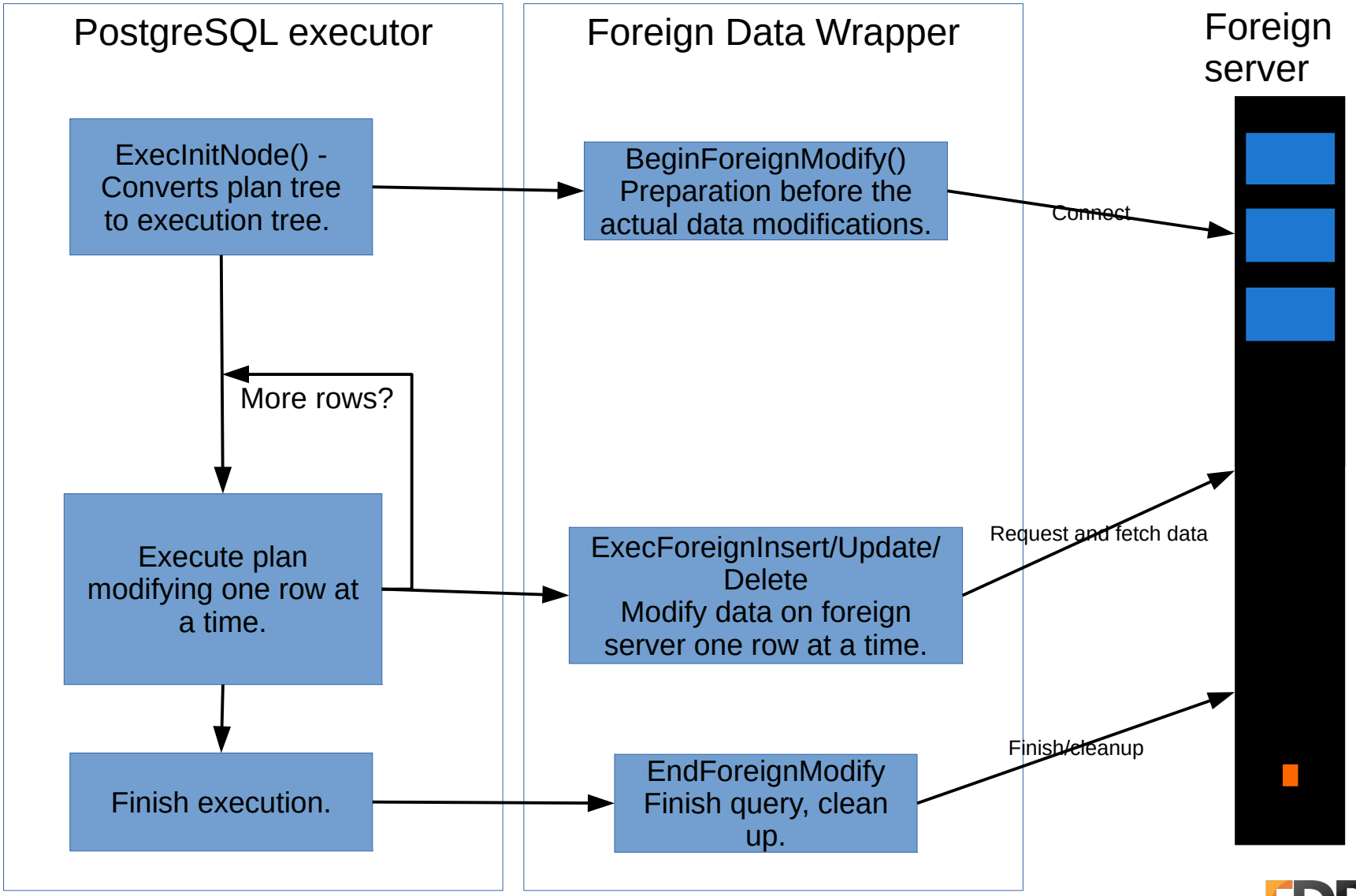
Join and post-join operation pushdown

- Use GetForeignJoinPaths() hook to add ForeignPaths for join between two foreign relations
  - Assess pushdown safety of join
- Use GetForeignUpperPaths() hook to add ForeignPaths for operations like grouping, aggregation, sort, limit etc.
- In GetForeignPlan() create a fdw_scan_tlist representing the result of join from the foreign server.

Modifying a foreign table: rewriter and planner



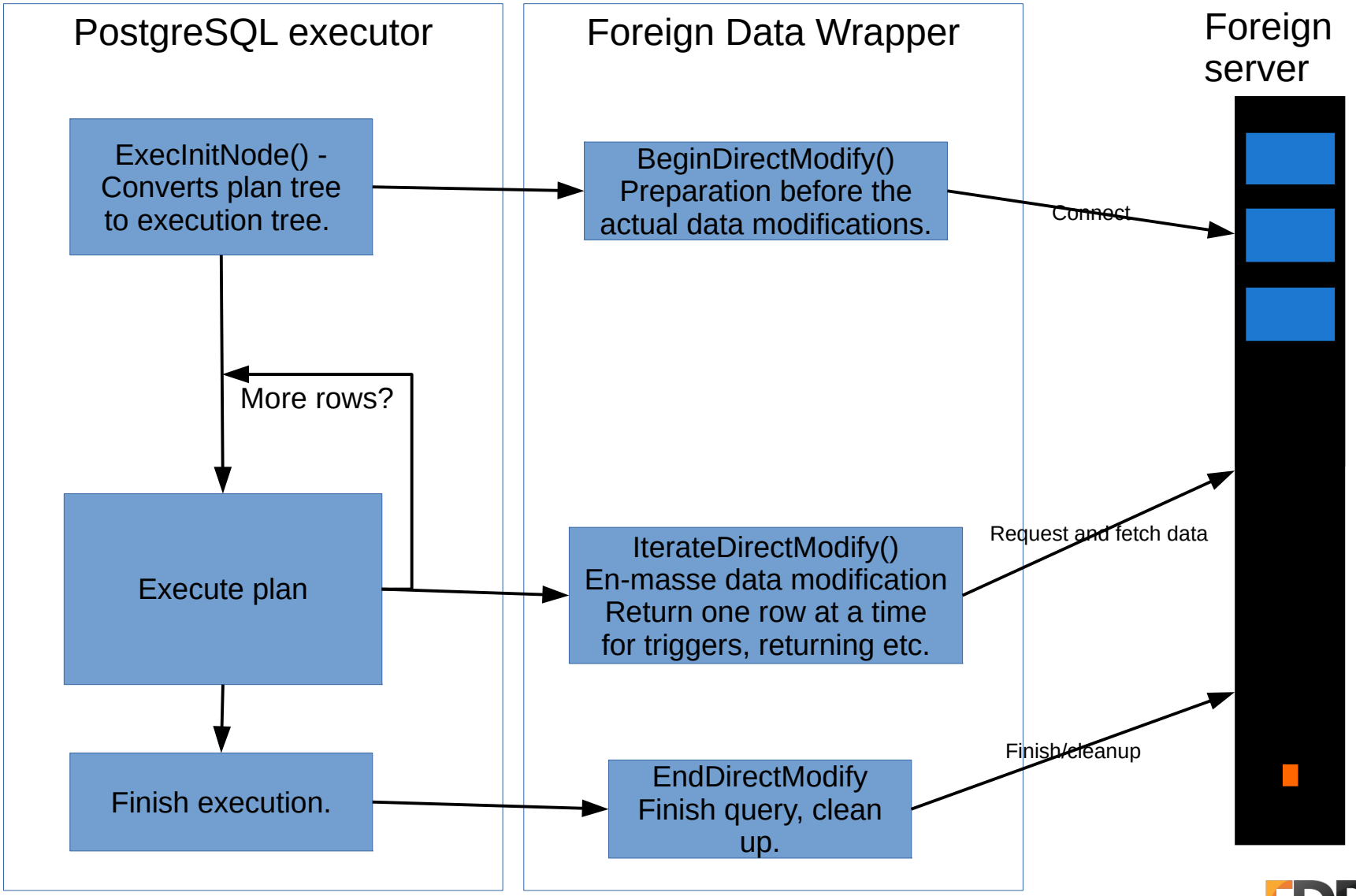
Modifying a foreign table: execution



Direct modification: planner

- PlanDirectModify()
  - Assess whether the DML is safe to be executed on the foreign server
  - Construct the query/code to execute the DML on the foreign server
  - Add ForeignScan plan as subplan to given ModifyTable plan

Direct Modification: execution



More APIs and further reading

- ExplainForeignScan, ExplainForeignModify
  - For adding FDW specific information in EXPLAIN output
- AnalyzeForeignTable
  - Scan foreign table to sample rows for collecting statistics
- ImportForeignSchema
  - Implementation hook for IMPORT FOREIGN SCHEMA command
- http://www.postgresql.org/docs/devel/static/fdwhandler.html

Multicorn

- Python based extension and FDW
- Makes it easy to write FDWs
- A “wrapper over wrapper”
- Good for quickies


THANK YOU
grazie
merci
spasiba
kam ouen
tak
gratzias
manana
mahalo
kitos
cheers
hvala
welalin
toda
gracias
grassie
thank you
danki
mahalo
danki
gracias
merci
thanks
na gode
mesi
dankon
modupe
talofa
miigwetch
thanks
takk
domo arrigato
danke
kitos
takk
dziekuje
grattitude
takk