

# TOSHIBA

FOSDEM PGDAY 2020



## High-performance SQL cluster engine.

Taiga Katayama

Jan 31, 2020

# Self-introduction

- Name: Taiga Katayama
- Live in Tokyo.
- Engineer at TOSHIBA CORPORATION, Open Source technology department.
  - Working on the database technology for 10 years.
    - Development of RDBMS and DB integration software using PostgreSQL and SQLite.
  - Since 2016 in relation to PostgreSQL.
    - Mainly developing FDW.



# Agenda

- Overview of PGSpider
- Features
  - Cascade connection
  - Multi-Tenant
  - Parallel processing
  - Pushdown
  - Keep-Alive
- Internal mechanism
  - How to realize features
- Performance measurement
- Summary

# Agenda

- Overview of PGSpider
- Features
  - Cascade connection
  - Multi-Tenant
  - Parallel processing
  - Pushdown
  - Keep-Alive
- Internal mechanism
  - How to realize features
- Performance measurement
- Summary

# Which Distributed SQL Engines is Better?

**In IIoT Era, Distributed SQL Engine is a Important Technology !!**

## **The Industrial Internet of Thing (IIoT) Era is coming.**

- IIoT data enable to accurate various growth with AI (such as manufacturing, power plant).

## **Challenges:**

### **How to consolidate data distributed in various databases?**

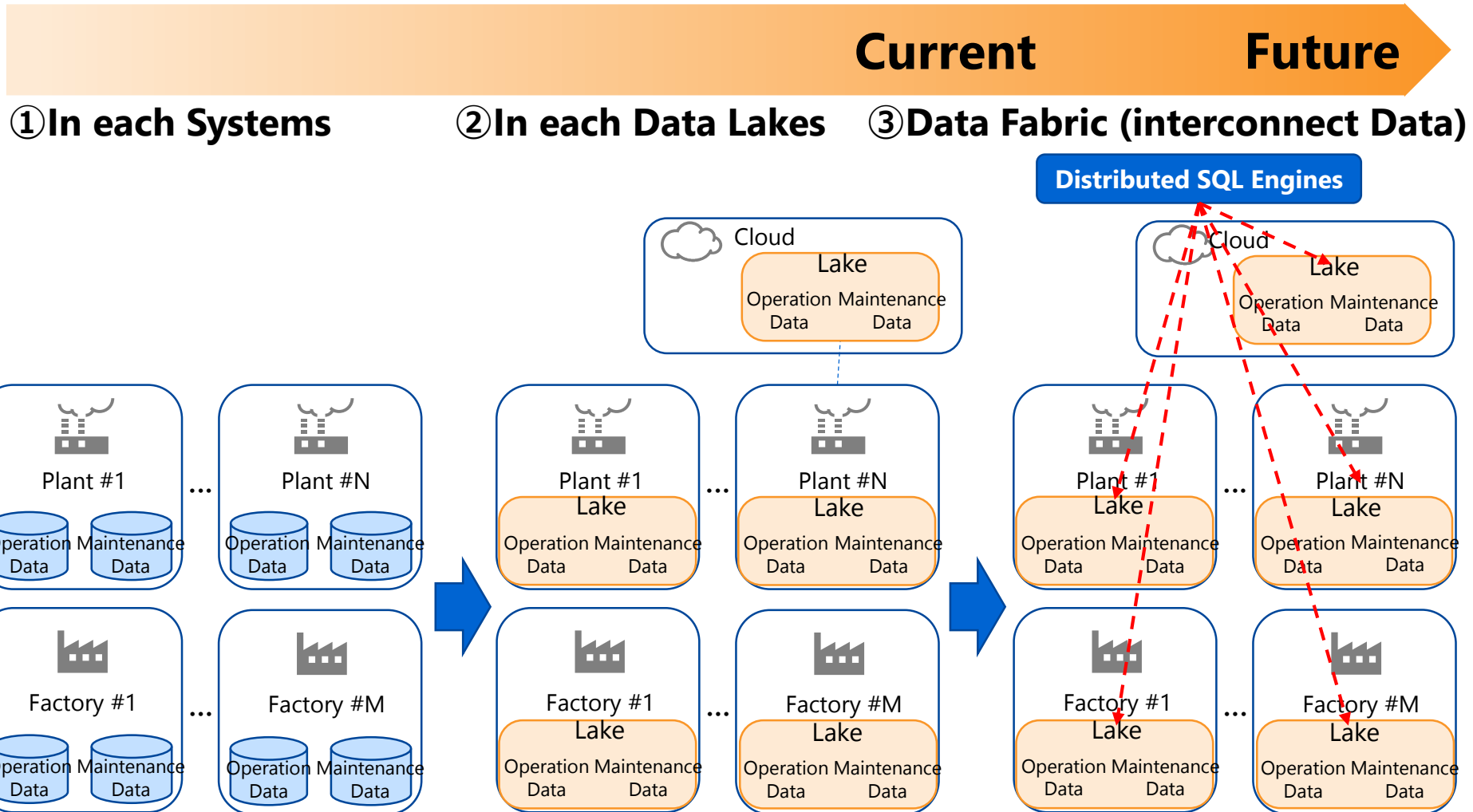
- In IIoT Era, Data sources/Databases are NOT in one database.

## **Solution: Distributed SQL Engine**

- It is one of core technologies in Our Cyber Physical Systems Strategy.
  - reference:  
[https://www.toshiba.co.jp/about/ir/en/pr/pdf/tpr20181122e\\_2.pdf](https://www.toshiba.co.jp/about/ir/en/pr/pdf/tpr20181122e_2.pdf)
- We are going to show you Performance Benchmark of them.

# Where are Data Sources?

Data Source/Database is distributed in various places.



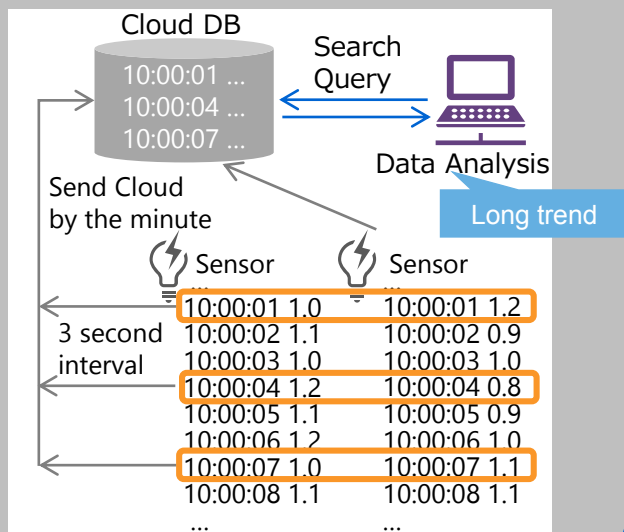
# What is PGSpider?



## Data retrieval system for distributed big data

**Before**

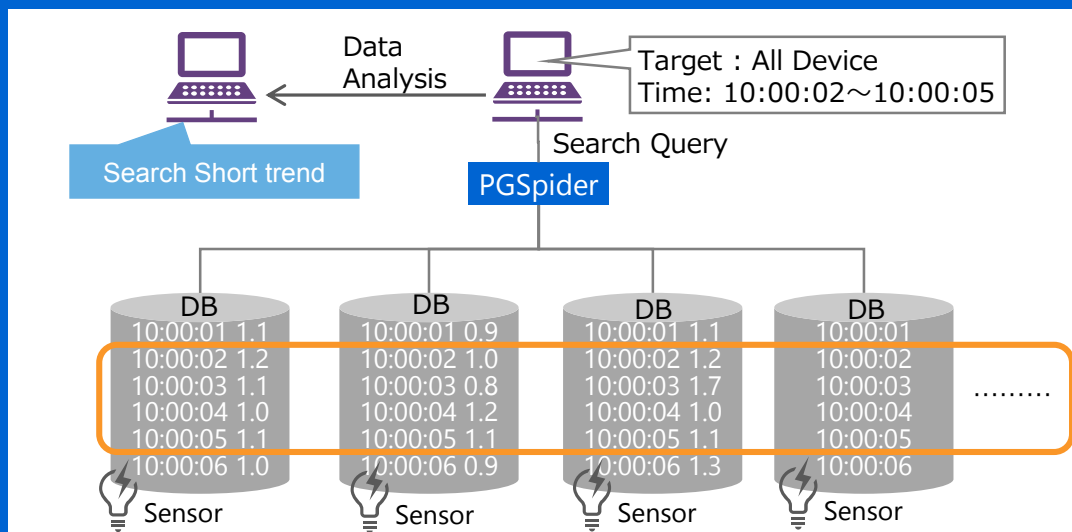
- Retrieval from Cloud
- Consider sampling interval
- Wait upload schedule



Existing System

**After**

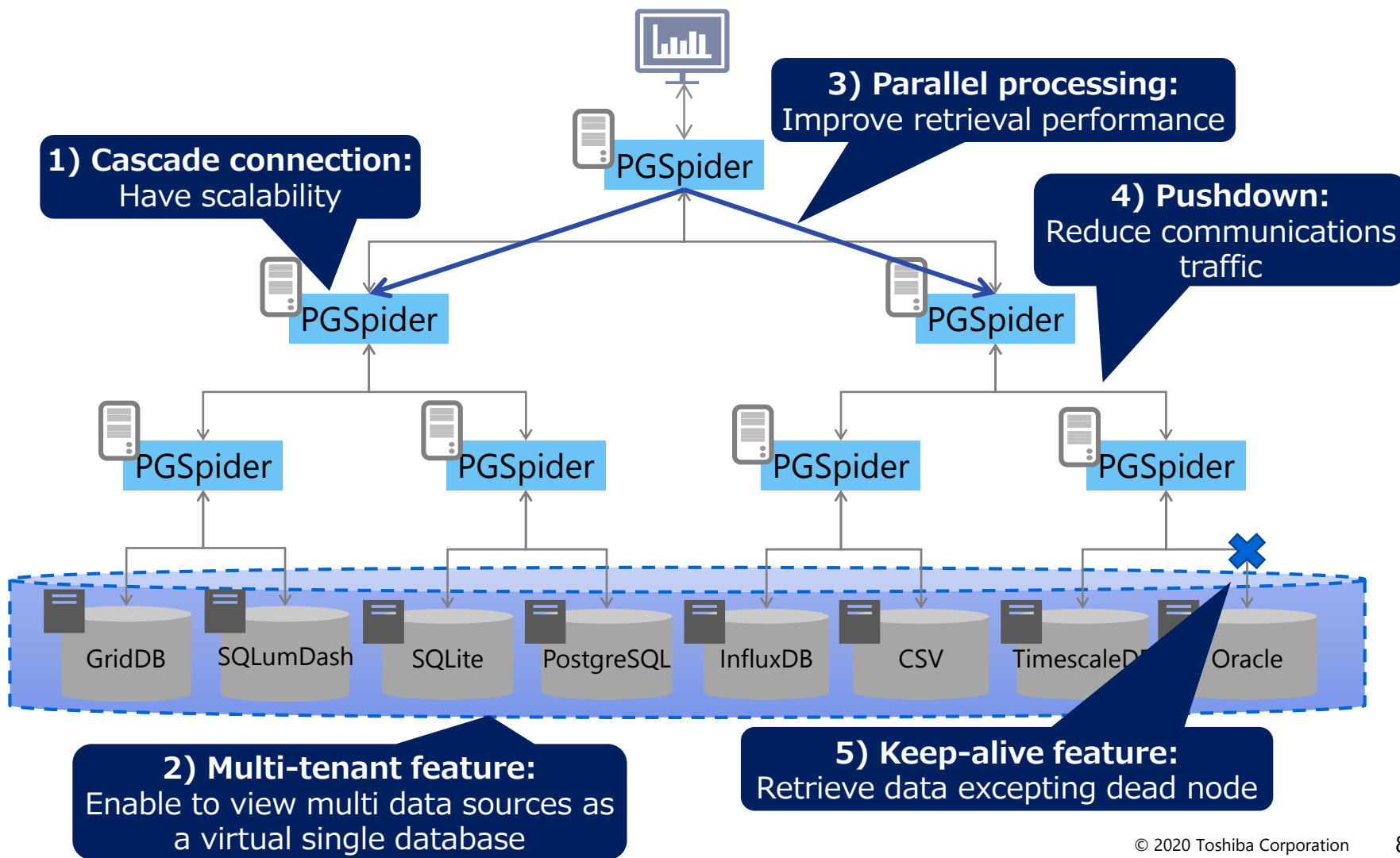
- Retrieval directly from edge as device side
- Retrieval raw data, Not consider sampling interval
- Cross-searching, No time lag
- Same I/F (SQL), Heterogeneous multi data source



System using PGSpider

# PGSpider's Features: Overview

Retrieve the distributed data source vertically.





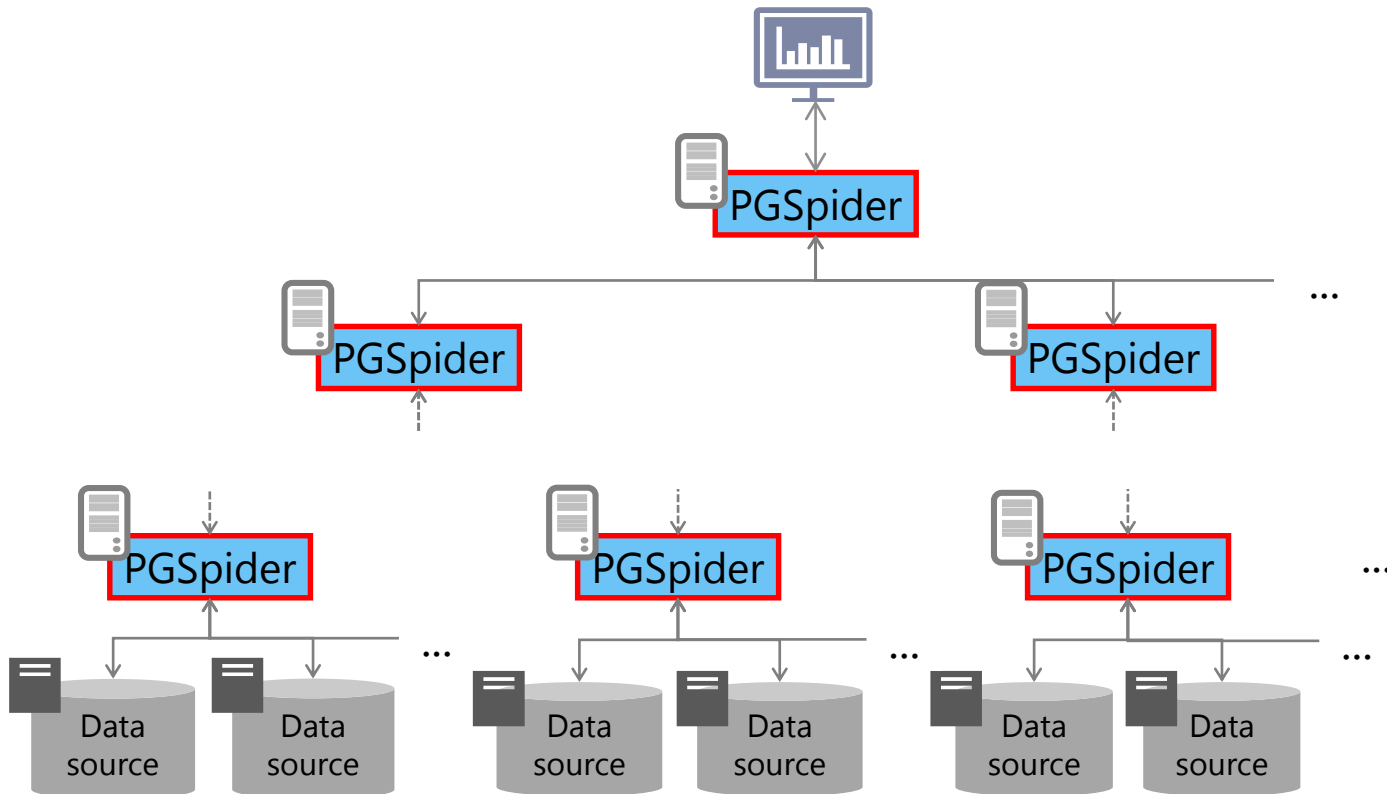
# Agenda

- Overview of PGSpider
- Features
  - Cascade connection
  - Multi-Tenant
  - Parallel processing
  - Pushdown
  - Keep-Alive
- Internal mechanism
  - How to realize features
- Performance measurement
- Summary

## Features: Cascade connection

### PGSpider can connect to PGSpider.

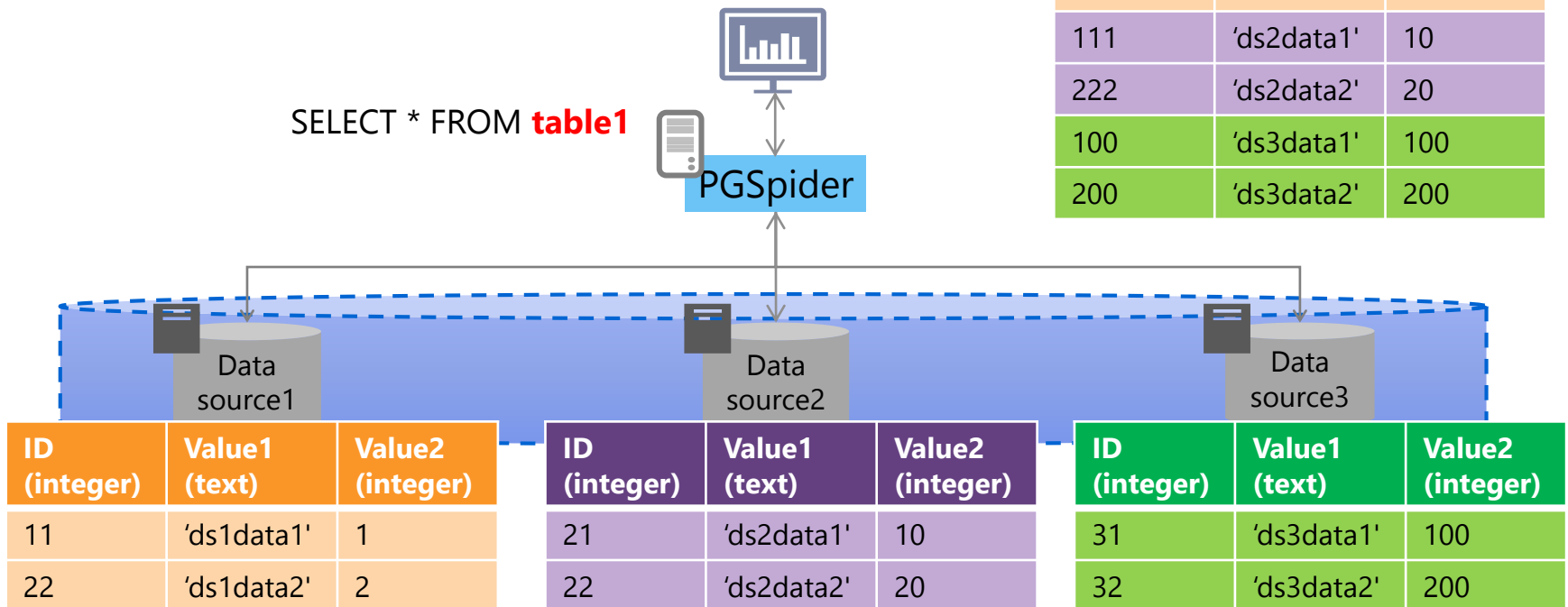
- Make it possible to access a number of data sources.



# Features: Multi-Tenant

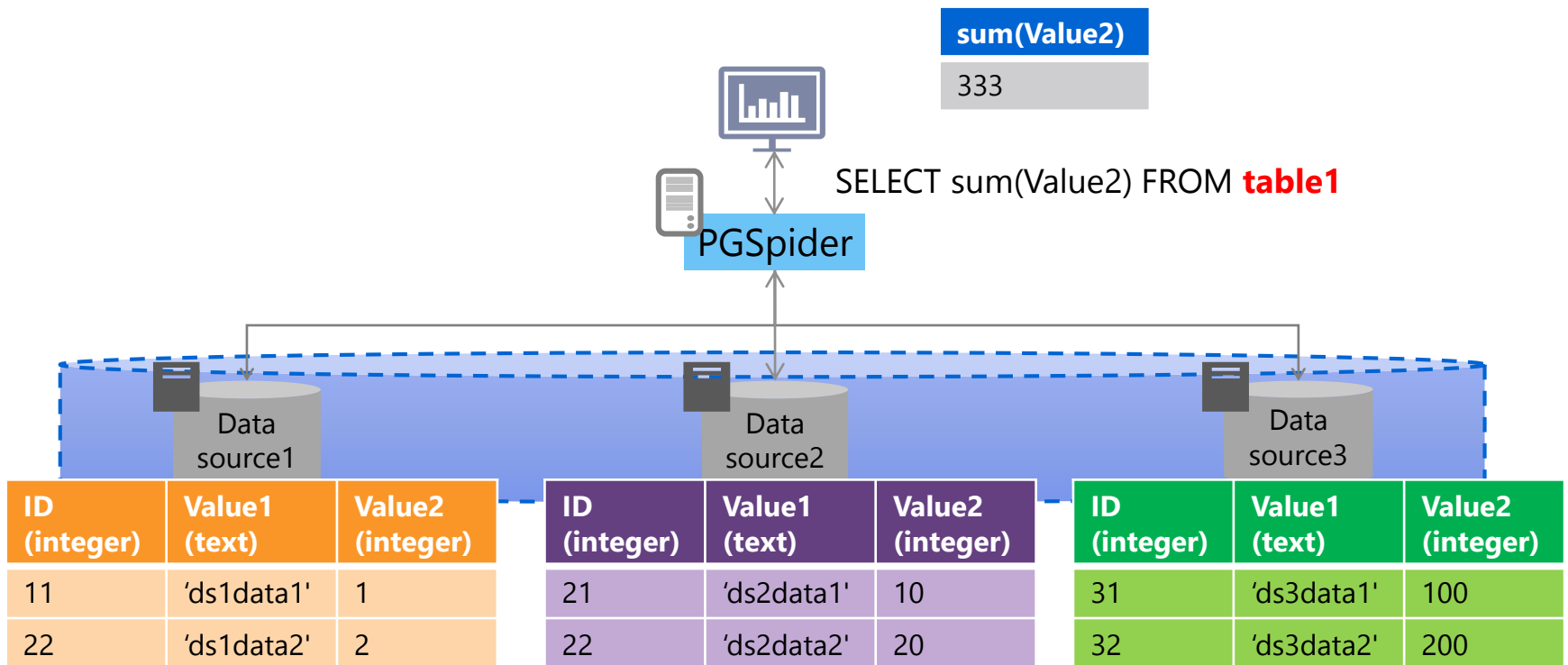
## User can get records in multi tables by one SQL easily.

- There are tables with similar schema in each data source.
- PGSpider can view them as a single virtual table.
  - User can see the table collected from tables in each data source by UNION ALL.



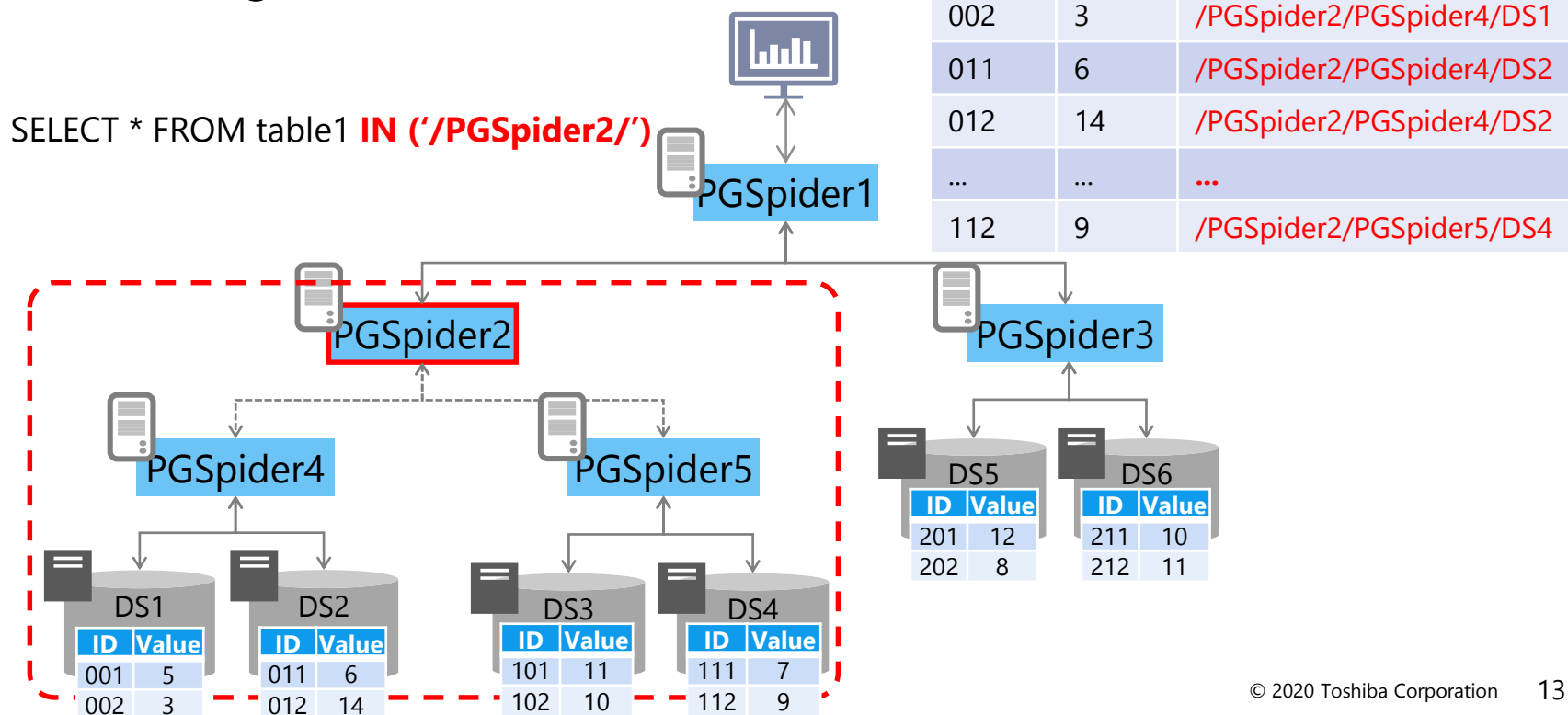
# Features: Multi-Tenant

- Example of aggregate function on Multi-Tenant.
  - table1 is expanded to ds1.table1 + ds2.table1 + ds3.table1 logically.



# Features: Filtering nodes on Multi-Tenant

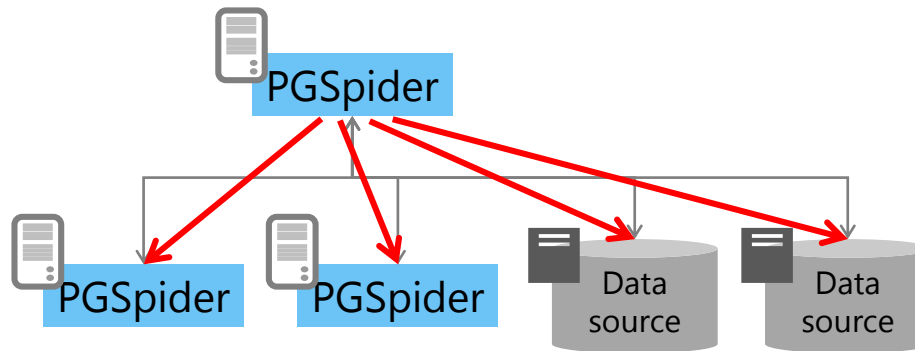
- User might want to filter nodes for multi tenant table.
  - PGSpider supports new IN-clause to specify the target node.
  - Node location is expressed by URL.
- User might want to know where the result row is stored.
  - PGSpider returns a result set having Nodename column.



## Features: Parallel processing

**Executes queries and fetches results from child nodes in parallel.**

- Improve retrieval performance.



## Retrieving records with "Value" greater than 10.

SELECT ...  
**WHERE**  
**Value>10**

## PGSpider

SELECT ...  
**WHERE**  
**Value > 10**

# PGSpider

SELECT ...  
**WHERE**  
Value>10

# PGSpider

SELECT ...  
**WHERE**  
Value>10

# PGSpider

SELECT ...  
WHERE  
Value>10

SELECT ...  
**WHERE**  
**Value>10**

SELECT ...  
**WHERE**  
**Value>10**

SELECT ...  
WHERE  
Value > 10

SELECT ...  
WHERE  
Value>10

SELECT ...  
WHERE  
Value>10

SELECT ...  
WHERE  
Value>10

SELECT ...  
WHERE  
Value>10

ID	Value
001	5
002	3

ID	Value
011	6
012	14

ID	Value
101	11
102	10

ID	Value
111	7
112	9

ID	Value
201	12
202	8

ID	Value
211	10
212	11

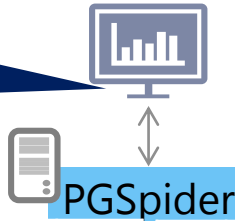
ID	Value
301	9
302	8

ID	Value
311	10
312	8

- WHERE condition
- ORDER BY
- Aggregate function
- GROUP BY

## Reduce communications traffic.

Retrieving records with "Value" greater than 10.



ID	Value
012	14
101	11
202	12
212	11

Supporting expression

- WHERE condition
- ORDER BY
- Aggregate function
- GROUP BY

ID	Value	ID	Value
012	14	202	12
101	12	212	11

ID	Value	ID	Value
012	14	101	11

ID	Value	ID	Value
202	12	212	11

The diagram illustrates the PGSpider component, represented by a blue box with the text "PGSpider". Above this box is a server icon (a rectangle with horizontal lines and a small circle) and a downward-pointing arrow, indicating a connection from a database to PGSpider. Below the PGSpider box is a table with two columns, "ID" and "Value". The first row of the table contains the values "012" and "14".

The diagram illustrates the PGSpider tool's interaction with a database. At the top, a document icon with three horizontal lines and a small circle below it represents a configuration or data source. A vertical arrow points from this icon down to a light blue rectangular box labeled "PGSpider". Below the "PGSpider" box is a database table. The table has two columns, each with a header row and a data row. The first column's header is "ID" and its data is "101". The second column's header is "Value" and its data is "11". A vertical arrow points from the "PGSpider" box down to the "Value" header of the second column, indicating a specific data point or operation.

ID	Value	ID	Value
101	11		

ID	Value	ID	Value
202	12	212	11

ID	Value	ID	Value
311	10	312	8

ID	Value
001	5
002	3

ID	Value
101	11
102	10
111	7
112	9

ID	Value
201	12
202	8
211	10
212	11

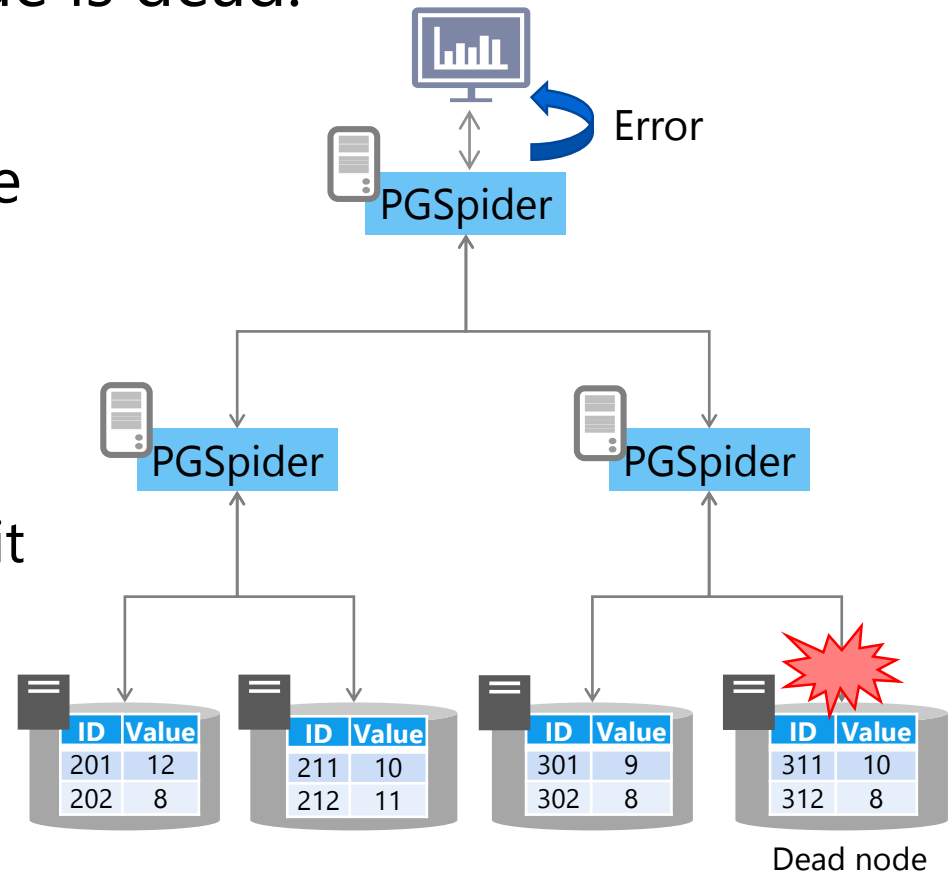
ID	Value
301	9
302	8
311	10
312	8



## Features: Keep-Alive

- There are issues if a node is dead.

- It might have to wait timeout to get response from dead node.
- Because an error is returned to application, it cannot get a result.
- Dead node might revive. it should check child node state in every query.

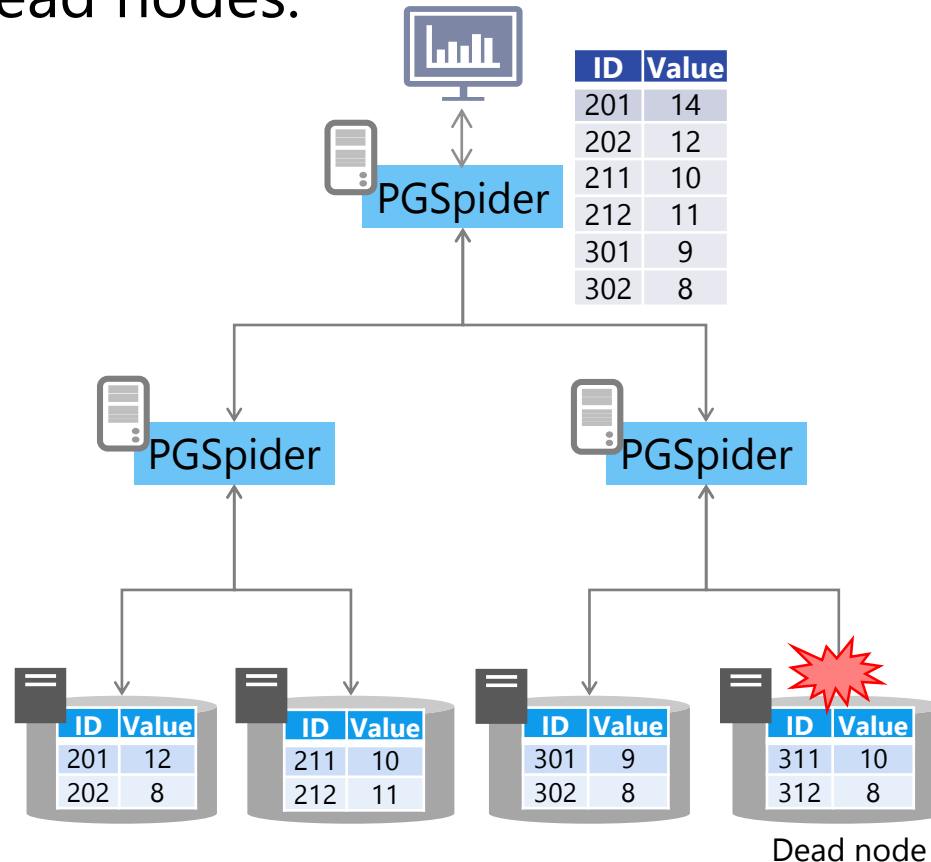


**Retrieves data from only valid child nodes.**

## Features: Keep-Alive

### Monitors child node state in the background.

- No need to wait timeout even if child node is dead.
- Ignore dead nodes.



# Agenda

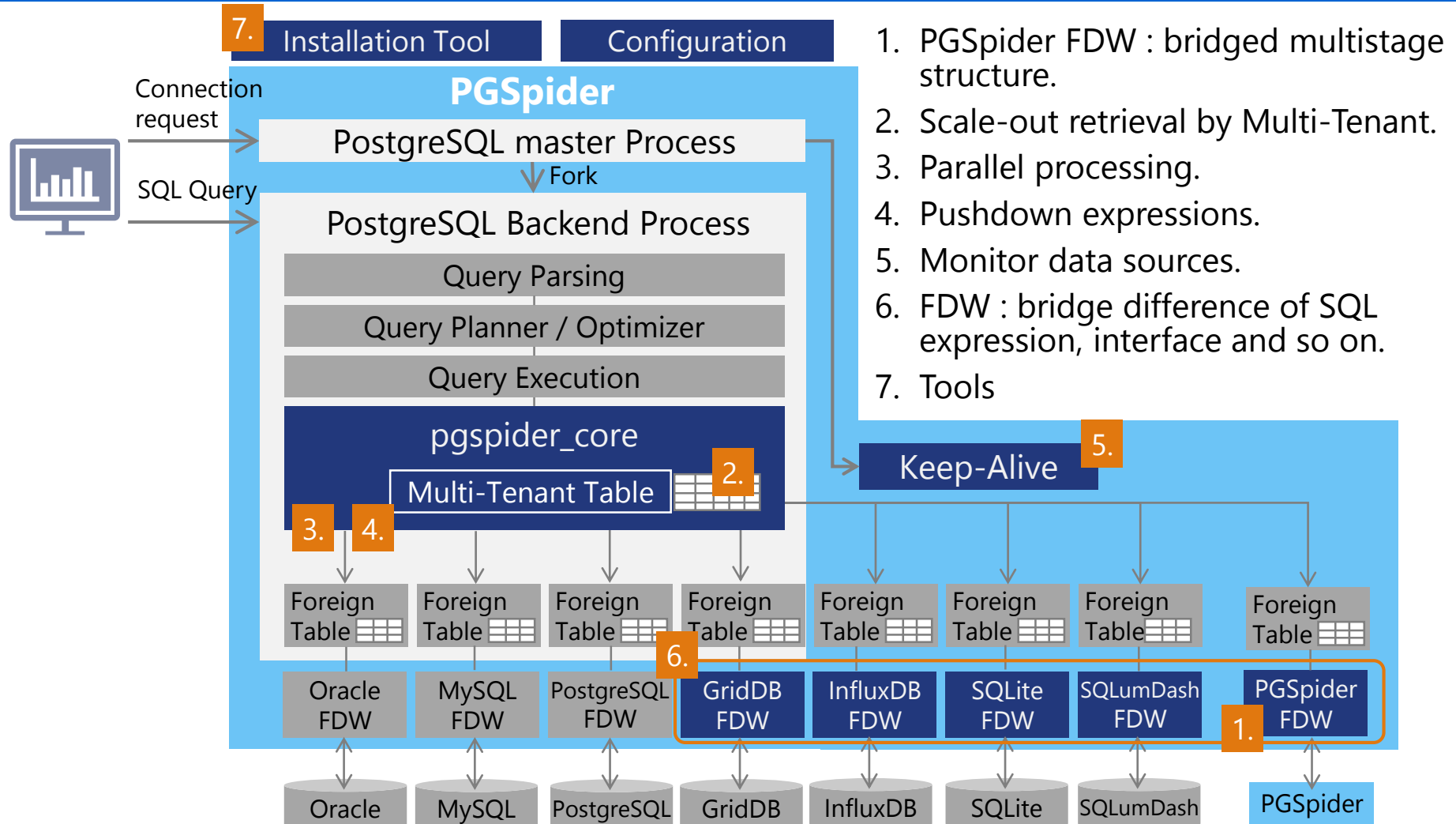
- Overview of PGSpider
- Features
  - Cascade connection
  - Multi-Tenant
  - Parallel processing
  - Pushdown
  - Keep-Alive
- Internal mechanism
  - How to realize features
- Performance measurement
- Summary

# Internal architecture

PGSpider

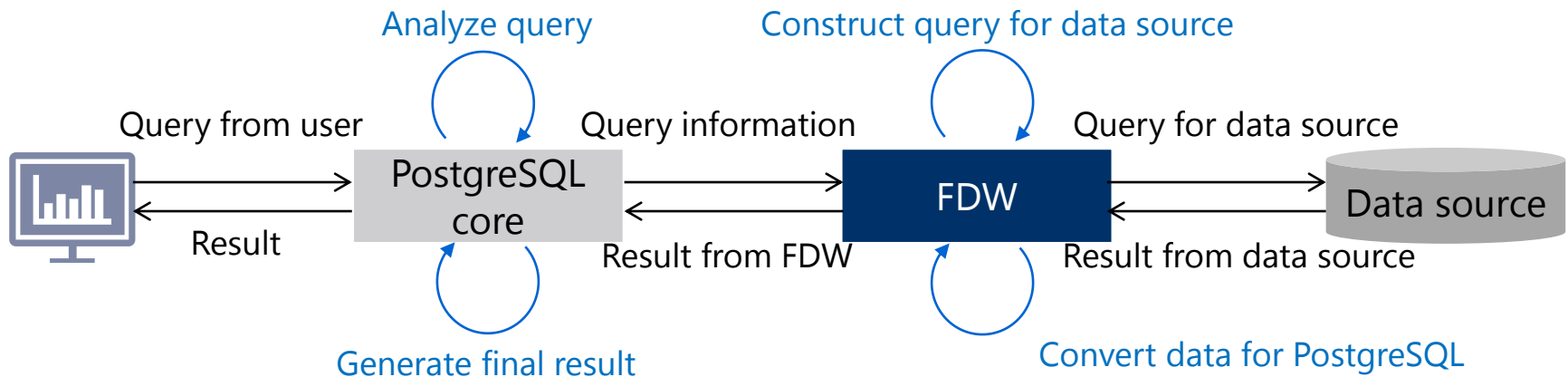
Module developed  
by TOSHIBA  
PostgreSQL  
Original Module

## Develop PGSpider based on PostgreSQL.



# What is FDW?

- Foreign Data Wrapper (FDW)
  - This feature enables user to access the other database from PostgreSQL
  - User can access various data sources by installing Data source FDW.

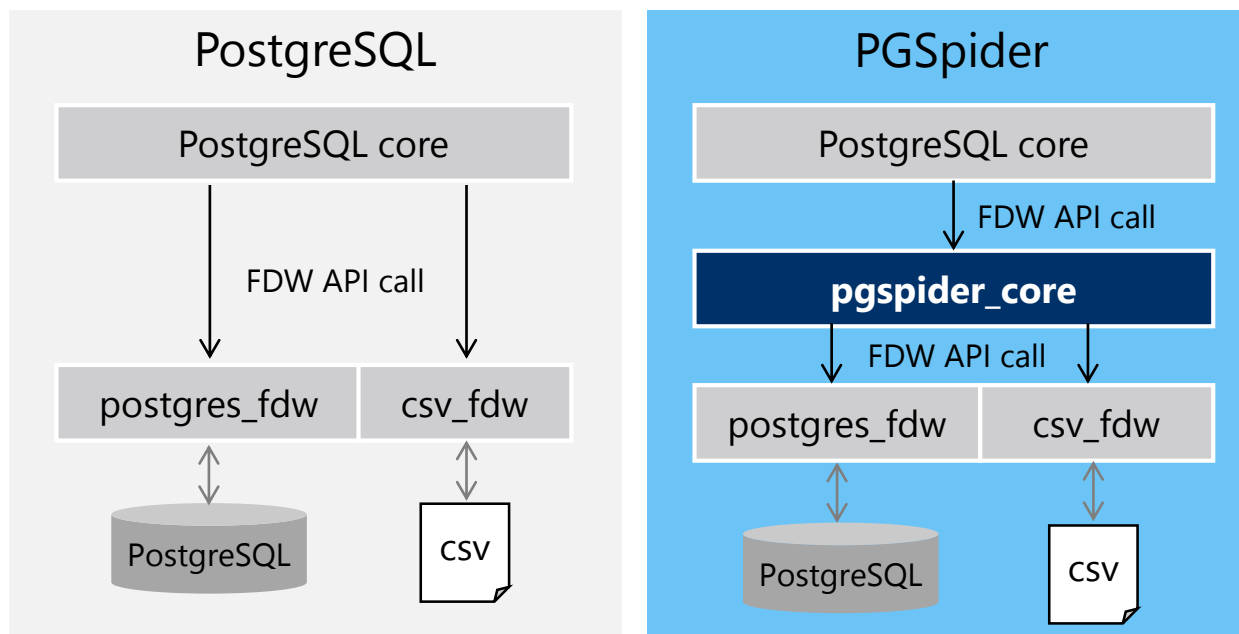


**PGSpider uses PostgreSQL's FDW.**

# What is pgspider\_core?

## pgspider\_core bridges PostgreSQL core and data source FDWs.

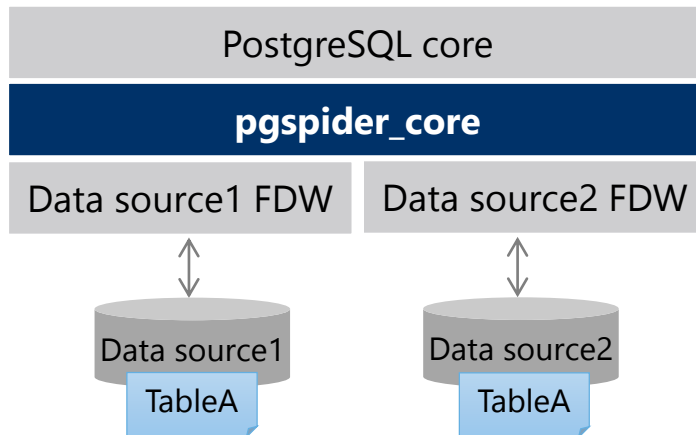
- Retrieve data from multi data sources using FDW of data source.
- Seen from PostgreSQL core, pgspider\_core is also one of FDW.



# Multi-Tenant and schema construction

pgspider\_core needs to expand Multi-Tenant table to data source tables.

- Assuming that
  - Foreign table for pgspider\_core is created.
  - Foreign tables for each data source are also created.
- Searches data source tables by name having [Multi-Tenant table name] + "\_\_" + [data source name].



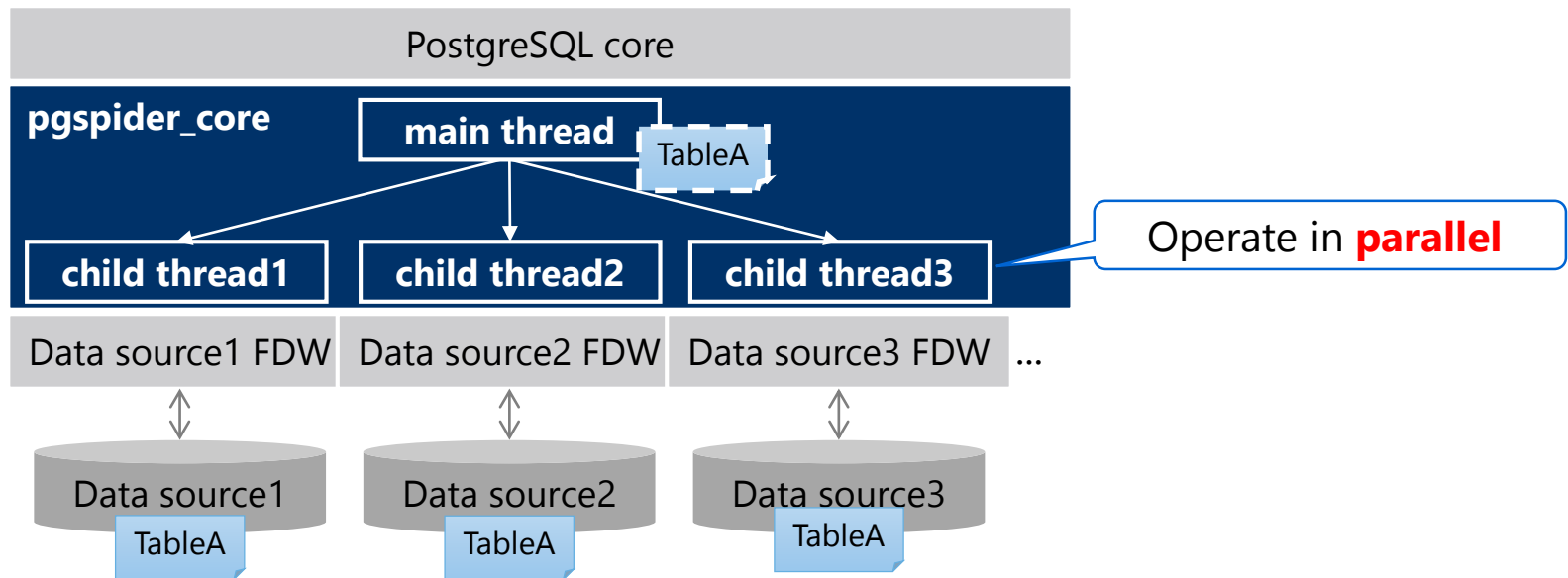
```
pgspider=# \det
          List of foreign tables
 Schema |      Table      |  Server
-----+-----+-----
 public | TableA          | pgspider_core
 public | TableA__ds1     | ds1
 public | TableA__ds2     | ds2
```

**Get data source FDWs based on the naming rule.**

# Parallel processing

## pgspider\_core creates threads for each child tables.

- Expand multi-tenant table to child tables.
- Create new threads for each child tables to access corresponding data source.

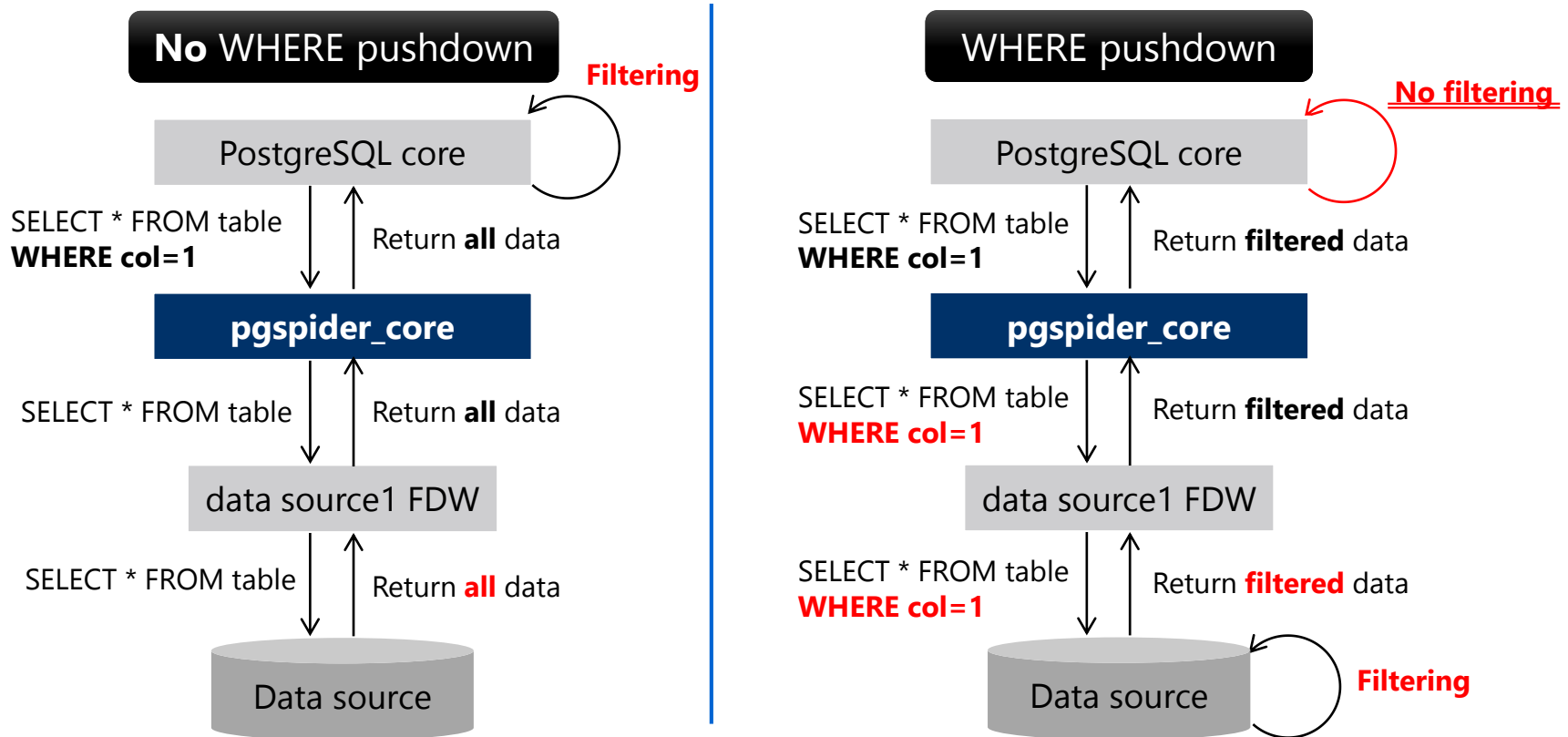




# Pushdown

PGSpider uses PostgreSQL's pushdown feature.

- Execute WHERE-clause and aggregate function in child node.

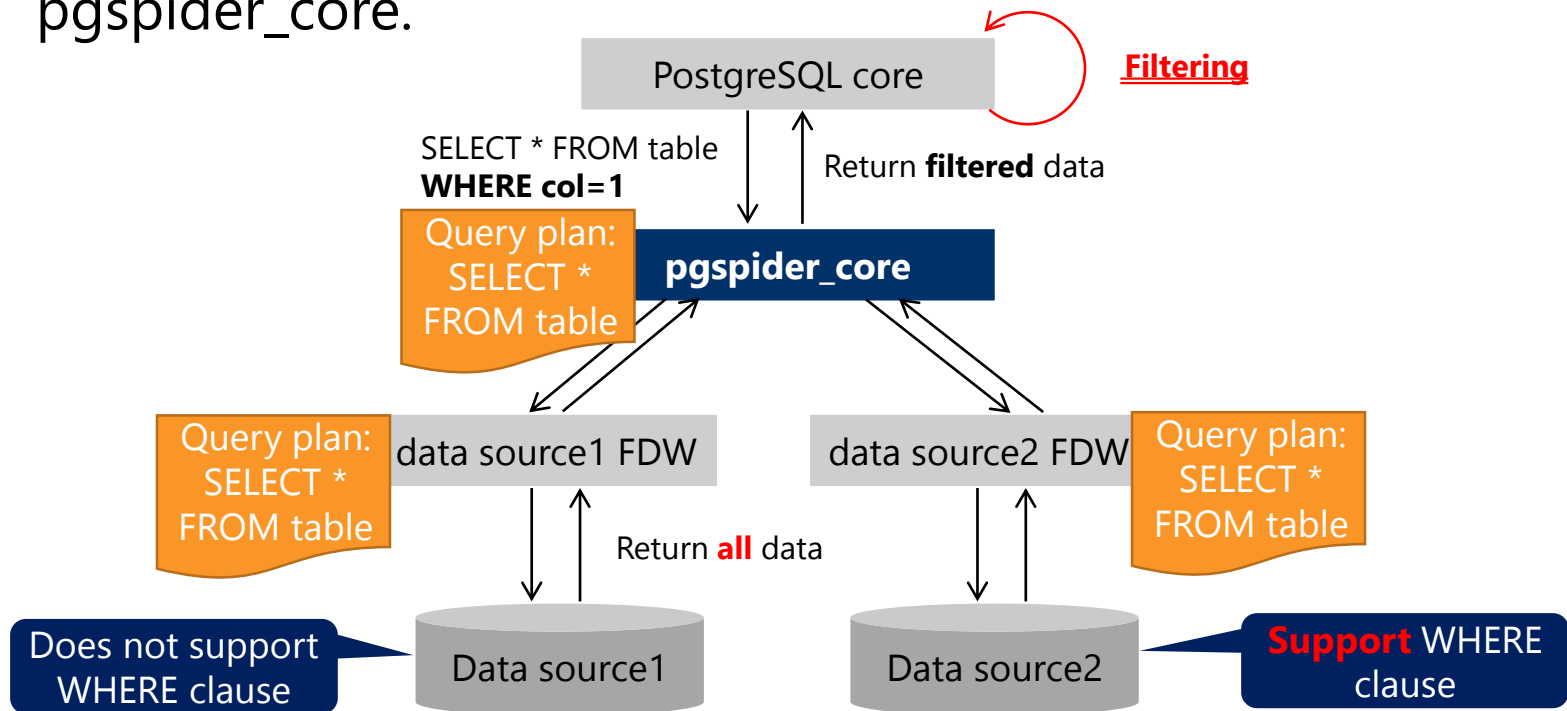


**PostgreSQL core behaves on the assumption that returned data is already filtered.**

# Pushdown's issue and solution

Some data sources don't support to pushdown.

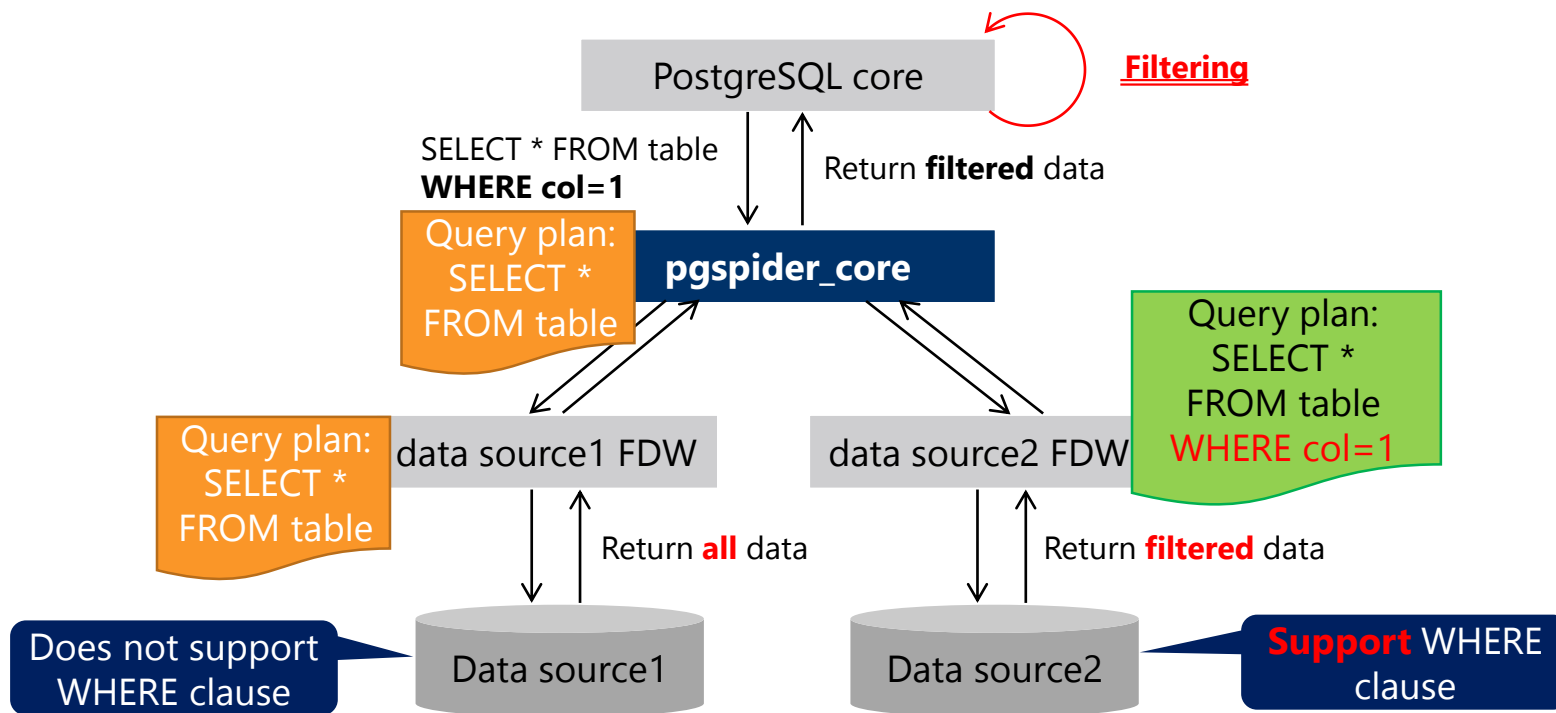
- PGSpider cannot pushdown unless all data sources support it because data source FDW uses same query plan as that of pgspider\_core.



**Create different query plans of data source FDW and pgspider\_core.**

# Solution of WHERE condition pushdown

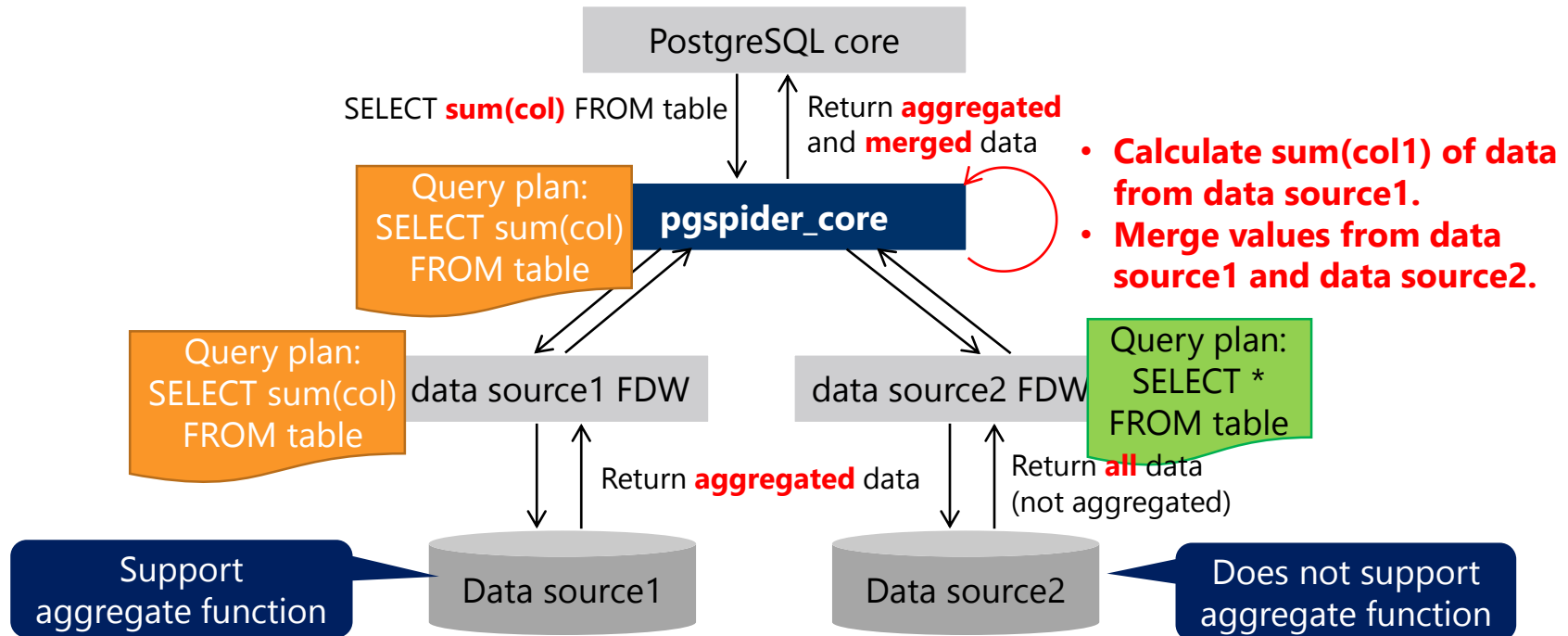
PostgreSQL core do filtering even if pgspider\_core pushdowns.



# Aggregate pushdown

Fall into similar situation as WHERE condition pushdown.

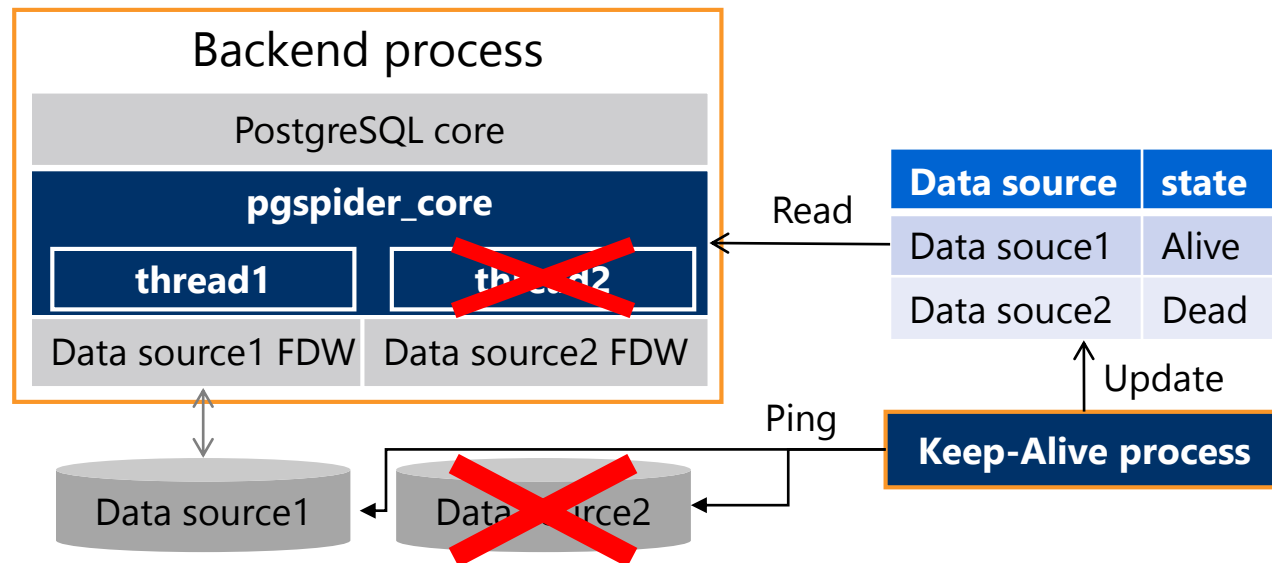
- PostgreSQL core expects that pgspider\_core returns **single aggregated** value.



**pgspider\_core do aggregation and merging.**

# Keep-Alive

- Alive checking process is implemented as `BackGroundWorkerProcess`.
  - It manages node state on PostgreSQL shared memory which is shared by backend processes.
- The process checks child node state independently from backend processes.
  - Ping to all child nodes and update state.
- `pgspider_core` on the backend process does not create thread for dead node.



# Agenda

- Overview of PGSpider
- Features
  - Cascade connection
  - Multi-Tenant
  - Parallel processing
  - Pushdown
  - Keep-Alive
- Internal mechanism
  - How to realize features
- Performance measurement
- Summary

# Overview of performance measurement

- Compare the performance with competitive software.
  - PGSpider vs Presto vs Apache Drill vs Dremio.
  - Data sources
    - PostgreSQL as relational database.
    - TimescaleDB as timeseries database.
- Condition
  - Use default parameter for all software.
  - Measure the 1st query execution time.
    - Cache is reset on all nodes.
- Machines
  - AWS: t3.xlarge (4vCPU), 6 instances
  - OS : CentOS7
  - Memory: 16GB
- Scenario
  - We executed SQLs on 3 constitution.

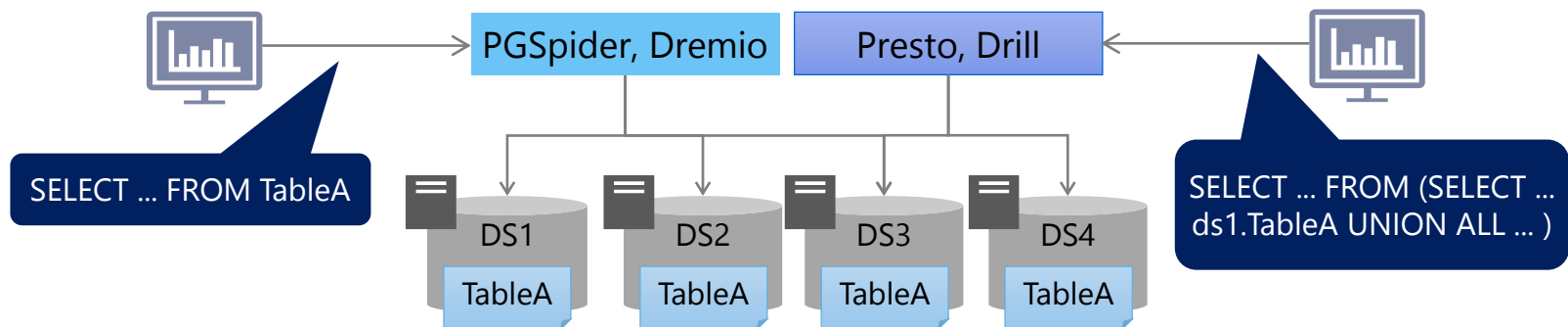
# Scenarios

- Scenario1: Search for big tables stored in timeseries databases.
  - LIMIT (Hit 1, 1000, 10000 records)
  - ORDER BY with LIMIT (Hit 1 record)
  - WHERE (Hit 1 record)
  - Aggregate
  - Aggregate with GROUP BY
  - Aggregate with GROUP BY and HAVING
  - Aggregate with GROUP BY and ORDER BY
- Scenario2: Join small tables stored in relational databases.
  - JOIN
  - JOIN and aggregate
  - JOIN and aggregate with WHERE
- Scenario3: Join “one master table on relational database” with “large tables on timeseries databases”.
  - JOIN
  - JOIN and aggregate with GROUP BY
  - JOIN with subquery





# Appendix

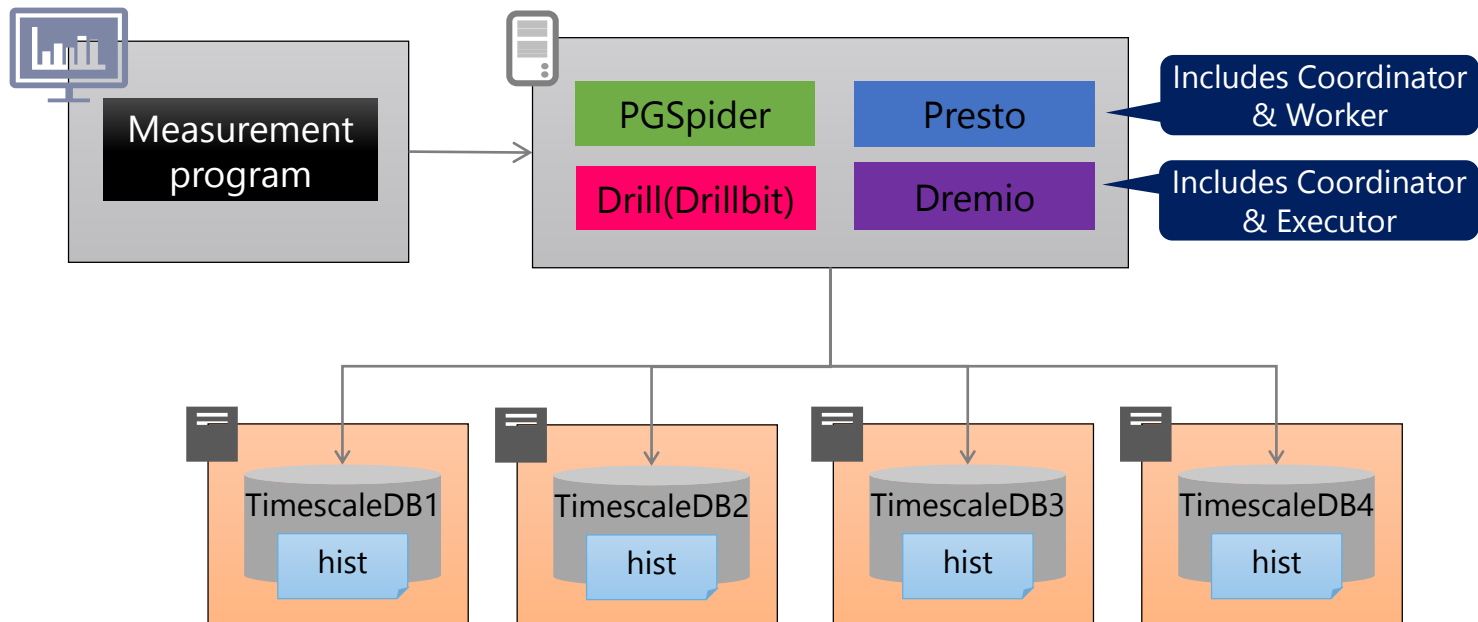
- Presto, Drill, Dremio don't have Multi-Tenant feature.
- PGSpider:
  - `SELECT sum(val) FROM TableA`
    - "table" is expanded automatically.
- Dremio: Use Virtual Dataset feature (Like view)
  - **CREATE VDS** TableA(col1 ...) as `SELECT * FROM ds1.TableA UNION ALL ... SELECT * FROM ds4.TableA`
  - `SELECT sum(val) FROM TableA`
- Presto, Drill: Expand the table by using **UNION ALL** manually
  - `SELECT sum(val) FROM (SELECT * FROM ds1.TableA UNION ALL ... SELECT * FROM ds4.TableA)`



# Scenario1: Node organization

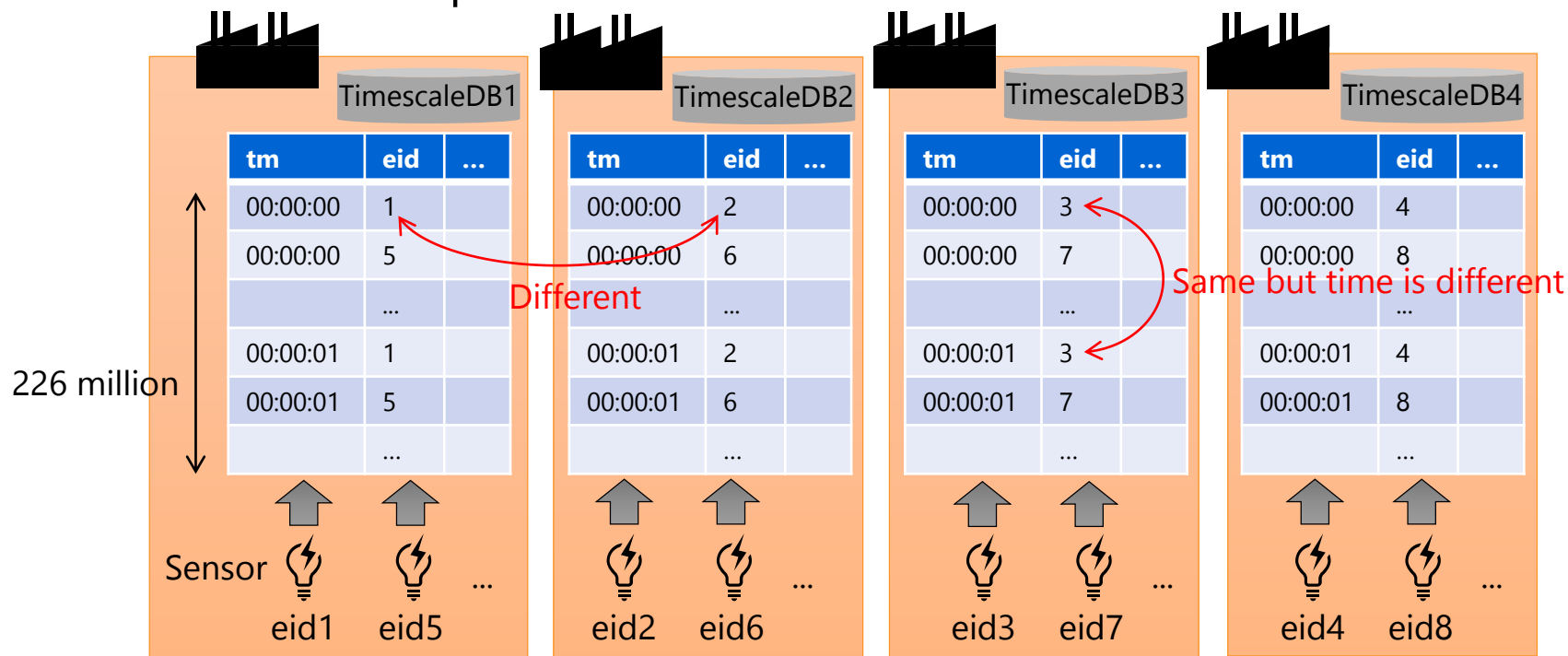
  AWS instance

- There are one cluster-engine.
- There are max 4 data sources (TimescaleDB).
  - 4 variations: Change the number of data sources 1, 2, 3, 4.



# Scenario 1: Evaluation data

- Each TimescaleDB has "hist" table storing sensor data.
  - 226 million records per node.
  - 900 million records in total.
- Each record has "eid" indicating sensor ID.
  - It is unique at each timestamp.
  - Don't overlap ID between data sources.

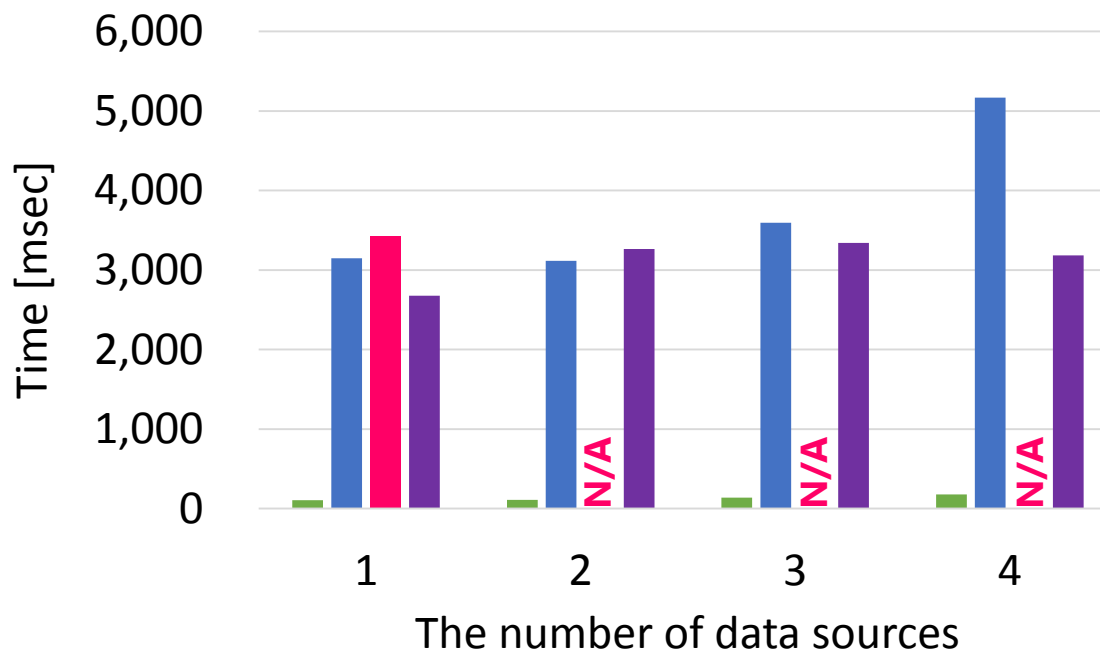


## Scenario1: WHERE (Hit 1 record)

PGSpider Presto Drill Dremio

SELECT \* FROM hist WHERE v=999 AND eid = 'sensor' AND ts = timestamp '2019-01-01 11:04:37'

- Drill becomes error if there are 2 or more nodes (Server crash).

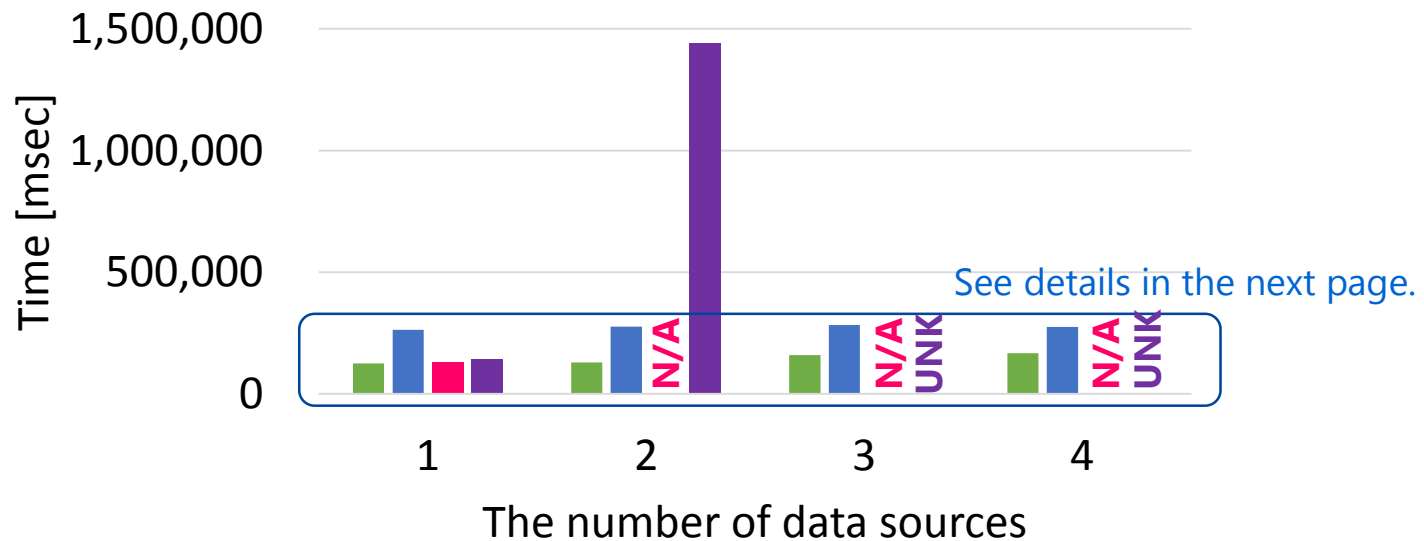


**All software are fast because of WHERE pushdown.  
Especially PGSpider is less overhead.**

# Scenario1: Aggregate

SELECT sum(v) FROM hist

- Drill becomes error if there are 2 or more nodes.
- Dremio is very slow if there are 2 or more nodes.
  - Did not measure Dremio of case 3 and 4 nodes.

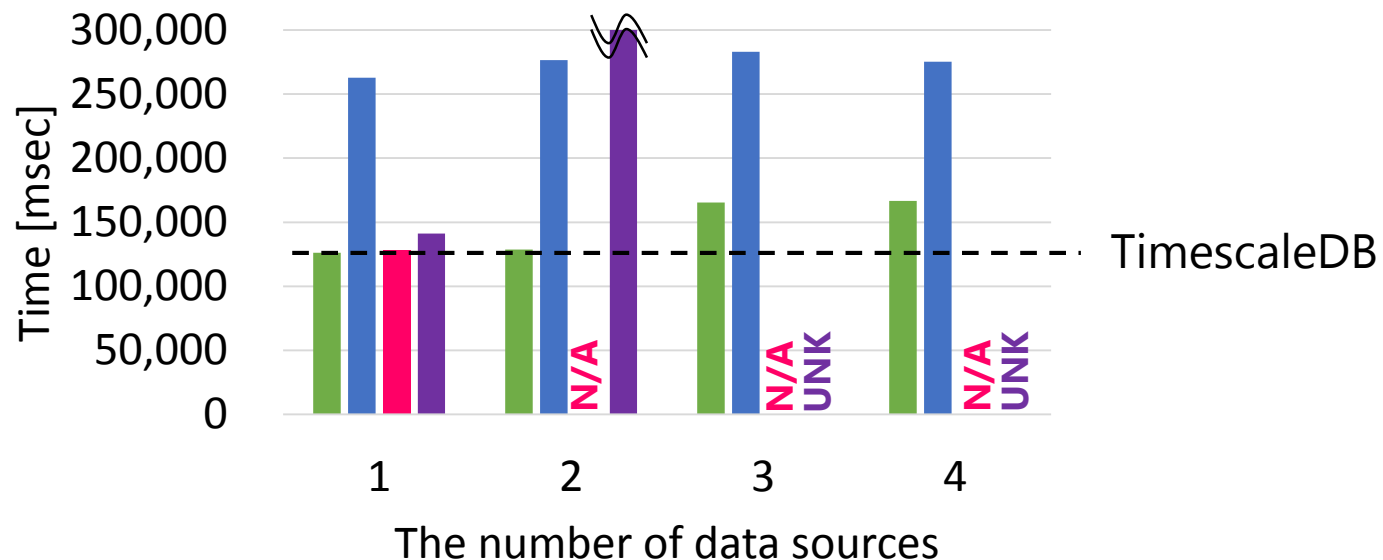


# Scenario1: Aggregate

PGSpider Presto Drill Dremio

SELECT sum(v) FROM hist

- TimescaleDB's performance (direct execution) is about 130,000 msec.
- PGSpider is faster because it pushdowns aggregate function.
- Presto does not pushdown.
  - There are a lot of data transfer and high CPU/memory usage.

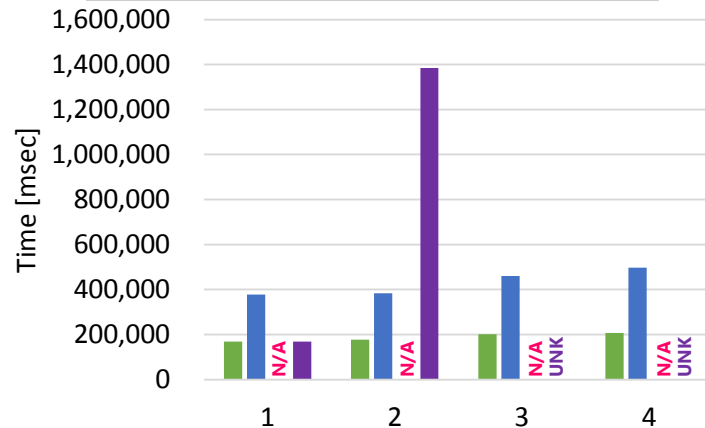


**PGSpider is better in terms of resource usage.**

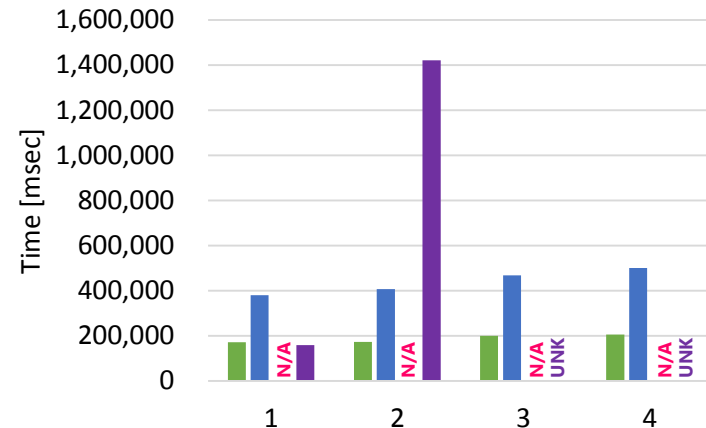
# Scenario1

- Trend is same.

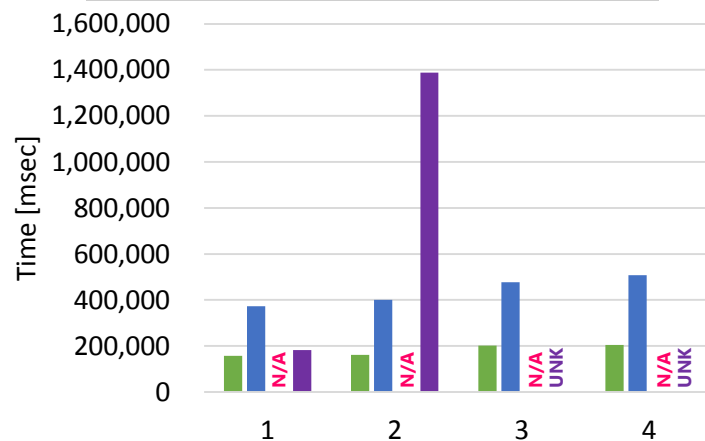
SELECT eid, avg(v) FROM hist GROUP  
BY eid



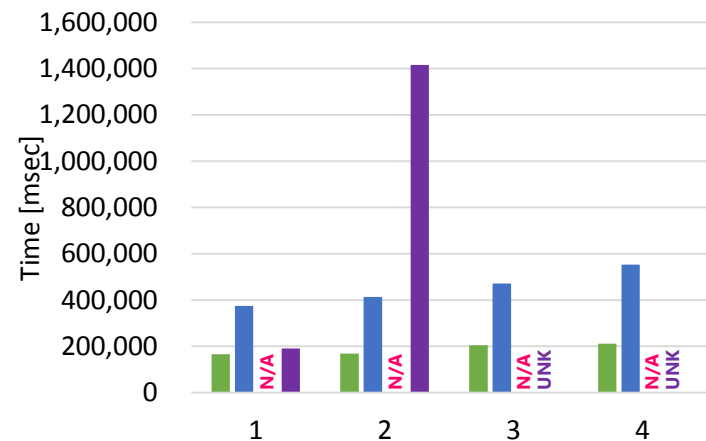
SELECT eid, avg(v) FROM hist GROUP  
BY eid HAVING avg(v) > 280



SELECT eid, avg(v) FROM hist GROUP  
BY eid ORDER BY eid



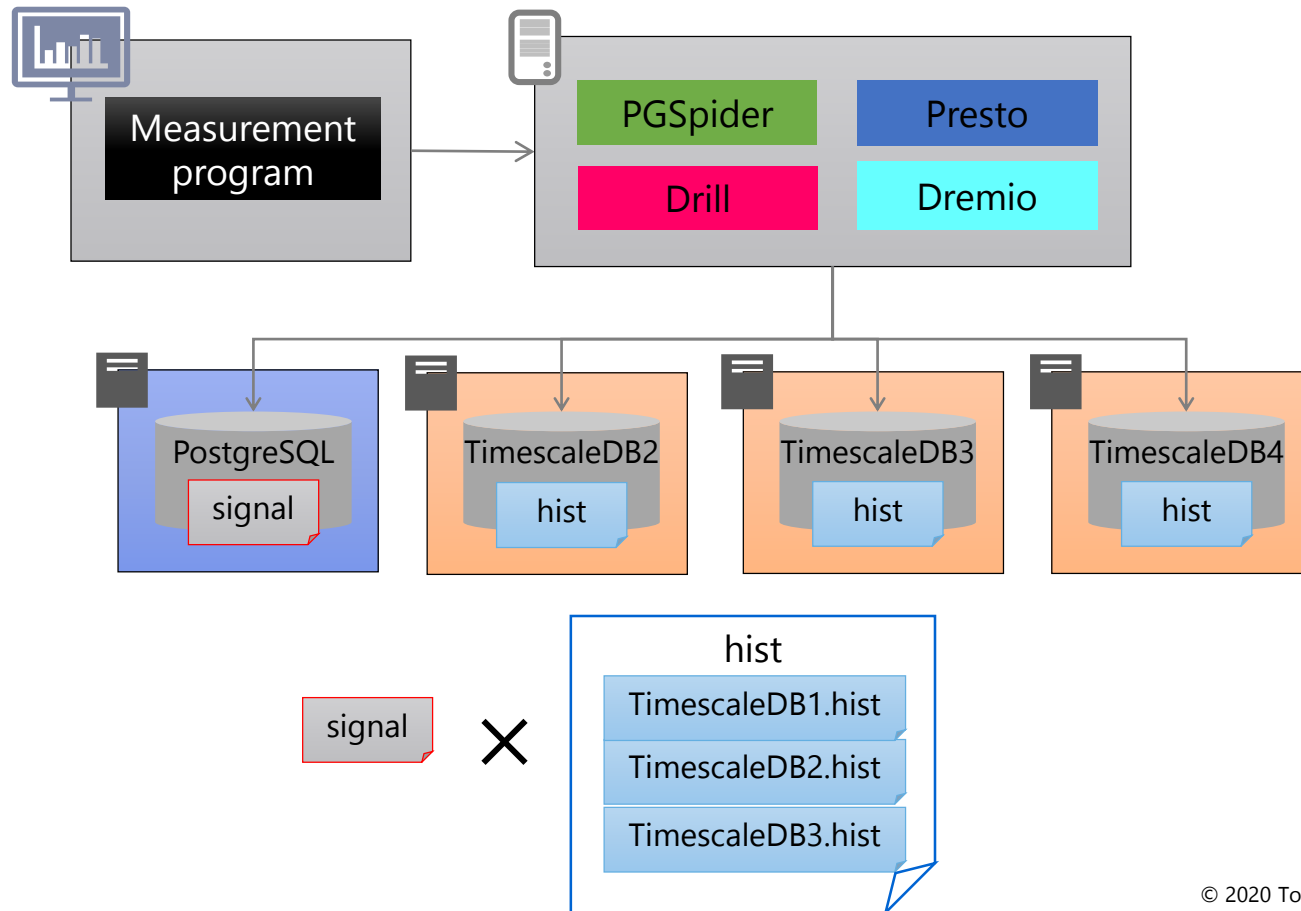
SELECT eid, variance(v) FROM hist  
GROUP BY eid ORDER BY eid



# Scenario3: Node organization

   AWS instance

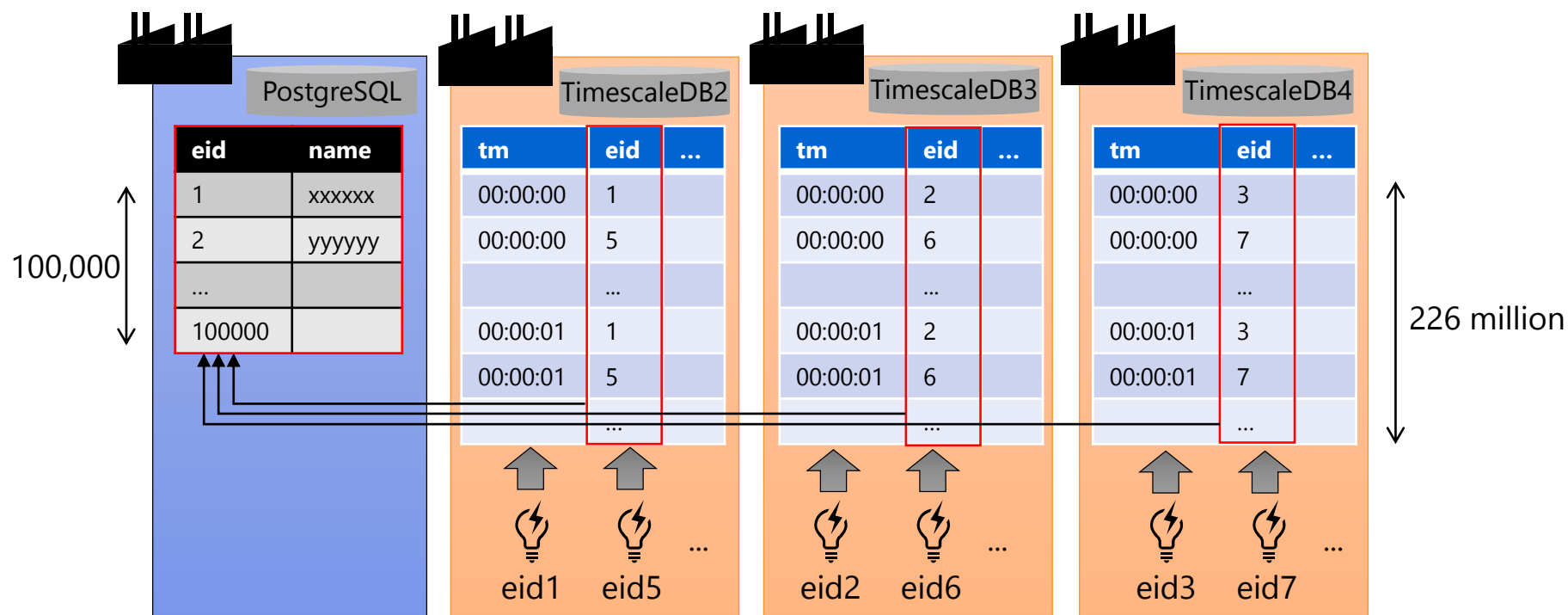
- There are one relational database (PostgreSQL).
- There are max 3 timeseries database (TimescaleDB).
  - 3 variations: Change the number of timeseries database 1, 2, 3.









## Scenario3: Evaluation data

- Each TimescaleDB has "hist" table storing sensor data.
  - Same as Scenario1.
  - 226 million records per node. About 700 million records in total.
- PostgreSQL has "signal" table.
  - Store sensor names corresponding to "eid".
  - There are 100,000 records.



## Scenario3: large data INNER JOIN

 PGSpider  Presto  Drill  Dremio

```
SELECT name, avg(v) FROM data INNER JOIN signal ON  
data.eid = signal.eid GROUP BY name
```

- Drill becomes error.
- Dremio is very slow if there are 2 or more nodes.
  - Did not measure Dremio of case 3 and 4 nodes.
- PGSpider is also very slow, no response for more than 1 hour.

## Scenario3: large data INNER JOIN

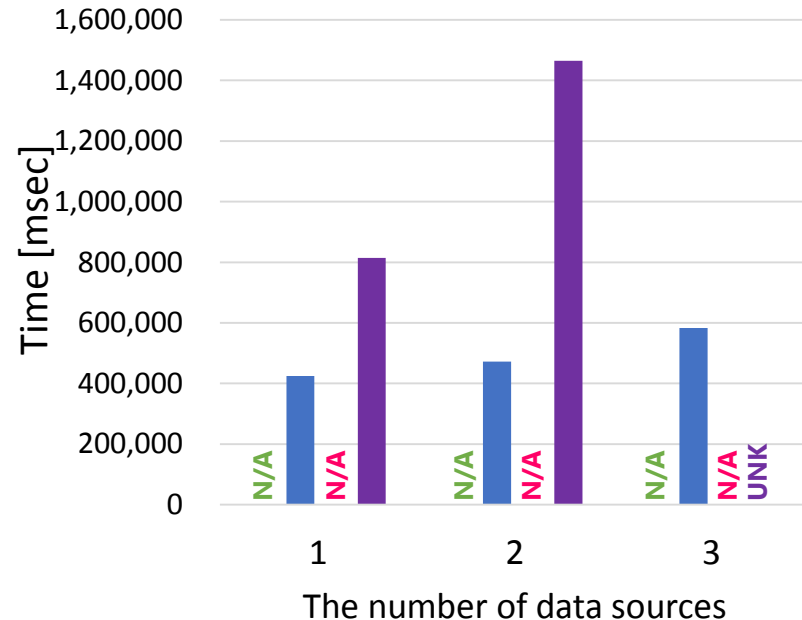
PGSpider Presto Drill Dremio

SELECT name, avg(v) FROM data INNER JOIN signal ON  
data.eid = signal.eid GROUP BY name

- Drill becomes error.
- Dremio is very slow if there are 2 or more nodes.
  - Did not measure Dremio of case 3 and 4 nodes.
- PGSpider is also very slow, no response for more than 1 hour.

Reason why PGSPider is slow:

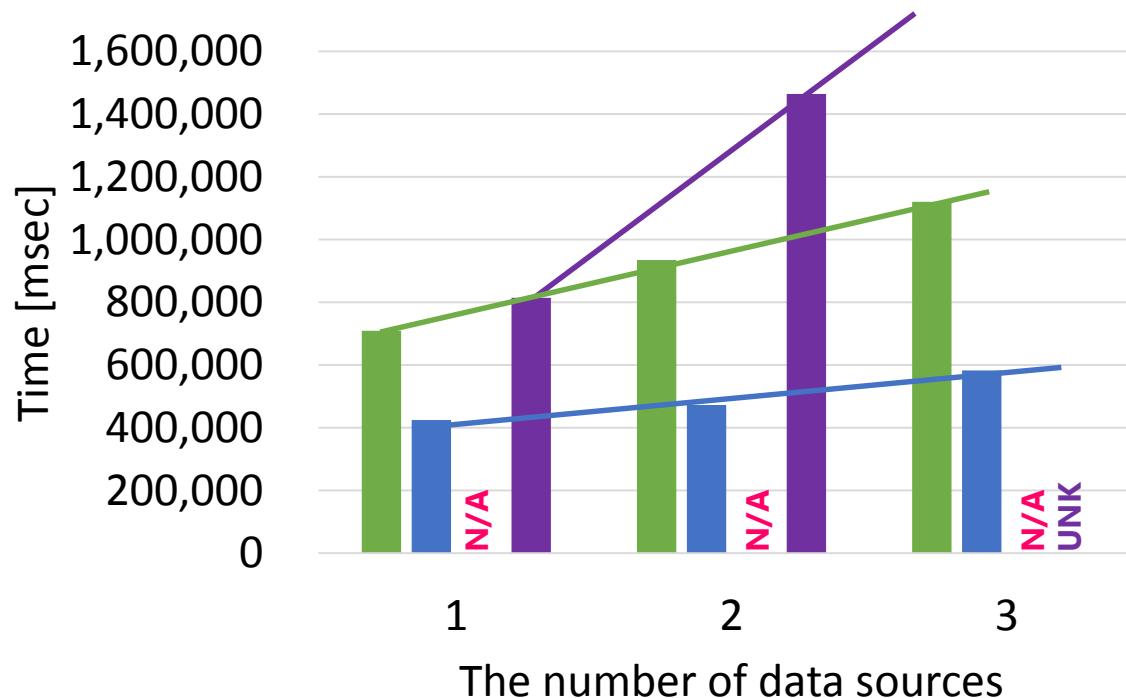
- Use merge join algorithm.
  - Retrieve all record.
- > Presto also does.
- **Sort records.**
- > **This takes a long time.**



## Scenario3: large data INNER JOIN

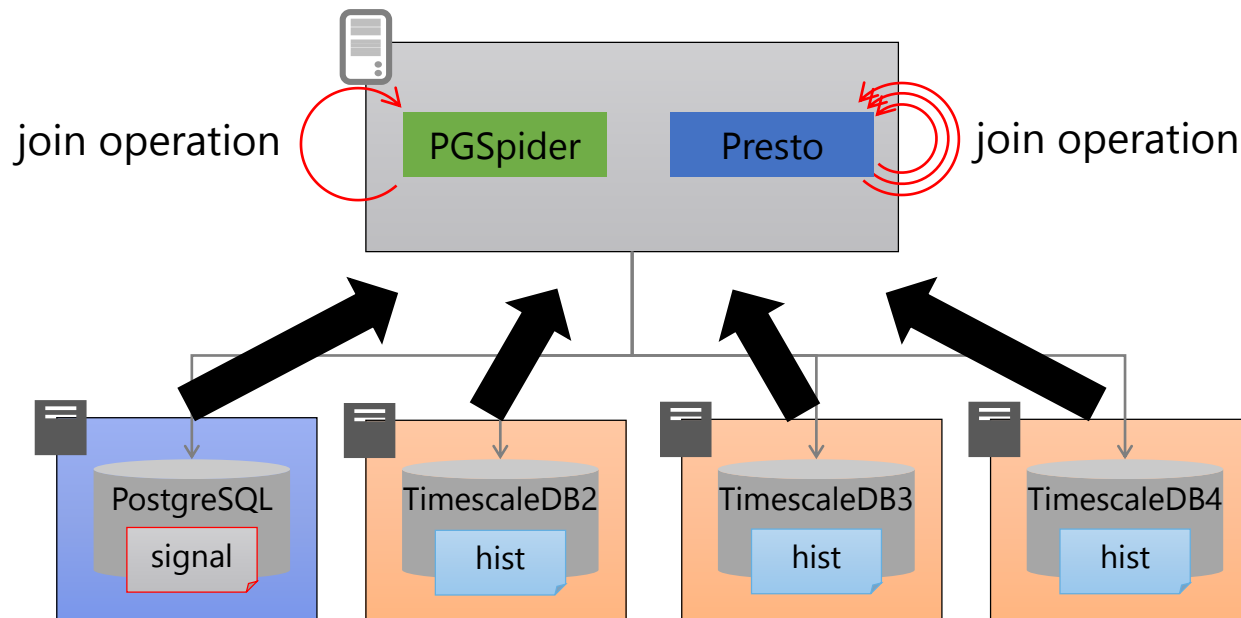
PGSpider Presto Drill Dremio

- Configured to use hash join on PGSpider.
  - "SET enable\_mergejoin TO off"
- Presto is affected less by data size than PGSpider.
  - Gradient of line is smaller.



## Scenario3: Analysis

- This scenario requires both software to retrieve all data.
- Presto might be able to do join operation in parallel.
- PGSpider can do in parallel **only** data retrieval.



# Result summary

	PGSpider	Presto	Drill	Dremio
<b>Scenario1: Search for big tables stored in timeseries databases.</b>				
LIMIT (Hit 1, 1000, 10000 records)	😊	😐	N/A	😐
ORDER BY with LIMIT (Hit 1 record)	😐	😊	N/A	N/A
WHERE (Hit 1 record)	😊	😐	N/A	😐
Aggregate	😊	😐	N/A	UNK
Aggregate with GROUP BY	😊	😐	N/A	UNK
Aggregate with GROUP BY and HAVING	😊	😐	N/A	UNK
Aggregate with GROUP BY and ORDER BY	😊	😐	N/A	UNK
<b>Scenario2: Join small tables stored in relational databases.</b>				
JOIN	😊	😐	😐	😐
JOIN and Aggregate	😊	😐	😐	😐
JOIN and aggregate with WHERE	😊	😐	😐	😐
<b>Scenario3: JOIN large tables.</b>				
JOIN with WHERE	😊	😊	N/A	N/A
JOIN and aggregate with GROUP BY	UNK(*1)	😊	N/A	N/A
JOIN with subquery	UNK(*1)	N/A	N/A	UNK

\*1 😐 if use hash join.

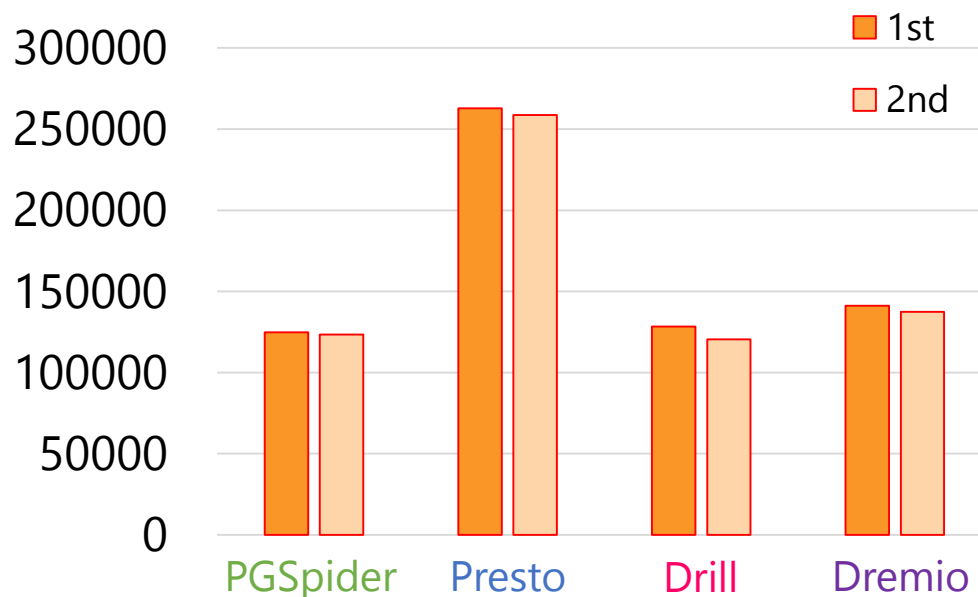
## Complement: Impact of cache

We have measured a time of 1st query execution so far.

- Systems (OS and software) have no cache.

Result of 2nd time query execution.

- Scenario2: SELECT sum(v) FROM hist
- 1 data source.



**Cache has little effect.**

# Conclusion

- We proposed PGSpider for IIoT system.
  - Developed by modifying PostgreSQL.
- PGSpider can retrieve data from a lot of nodes.
  - Easy to access tables by one SQL with Multi-Tenant feature.
  - Retrieve data in parallel.
  - Improve performance by controlling pushdown.
  - Ignore dead node by checking node existence.
- Measured time compared with competitive software.
  - PGSpider is less overhead.
  - Needs to improve data joining operation for large tables.





Thank you!

Questions?

PGSPider source code:

<https://github.com/pgspider/pgspider>

Email:

[taiga.katayama@toshiba.co.jp](mailto:taiga.katayama@toshiba.co.jp)

**TOSHIBA**