Innovative R&D by NTT

PostgreSQL Built-in Sharding:

Enabling Big Data Management with the Blue Elephant

E. Fujita, K. Horiguchi, M. Sawada, and A. Langote
NTT Open Source Software Center

Who Are We?

Kyotaro Horiguchi Amit Langote

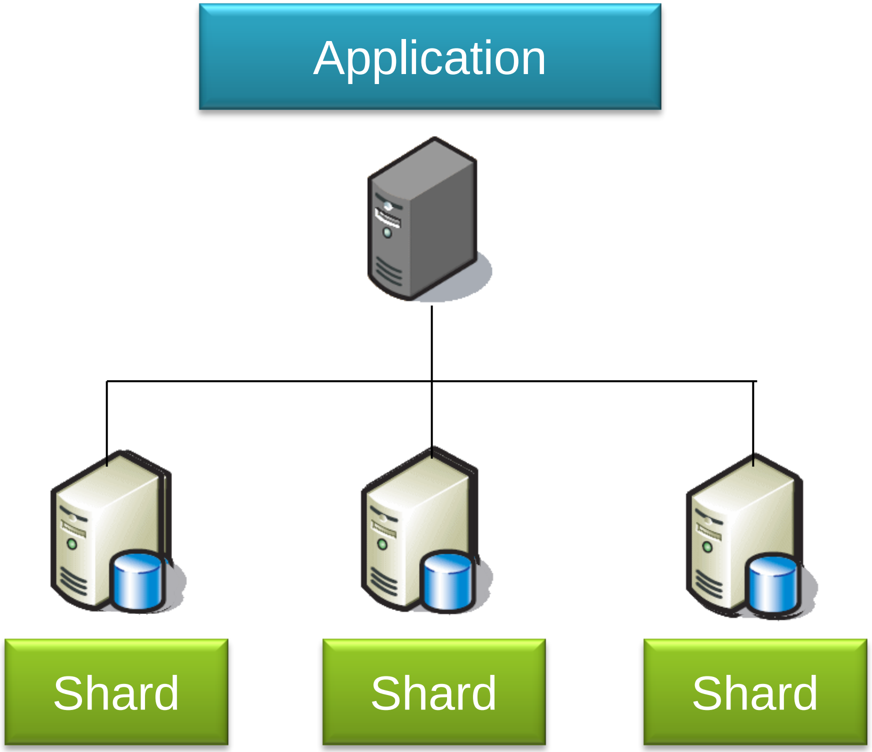
Etsuro Fujita Masahiko Sawada

Outline

- Database Sharding
- Built-in Sharding for PostgreSQL
- Core Features for Built-in Sharding
- Demonstrations
- Concluding Remarks

Database Sharding

- **A technique to scale out databases**
  - Spreads data across multiple shards
  - Allows high read/write scaling in environments that have very large data sets



Sharding at the Application Layer

- **Common practice to scale out PostgreSQL**
- **Problems:**
  - Need to write application logic to manage shards
    - Distributed transaction is cumbersome
    - Distributed join/aggregation is cumbersome

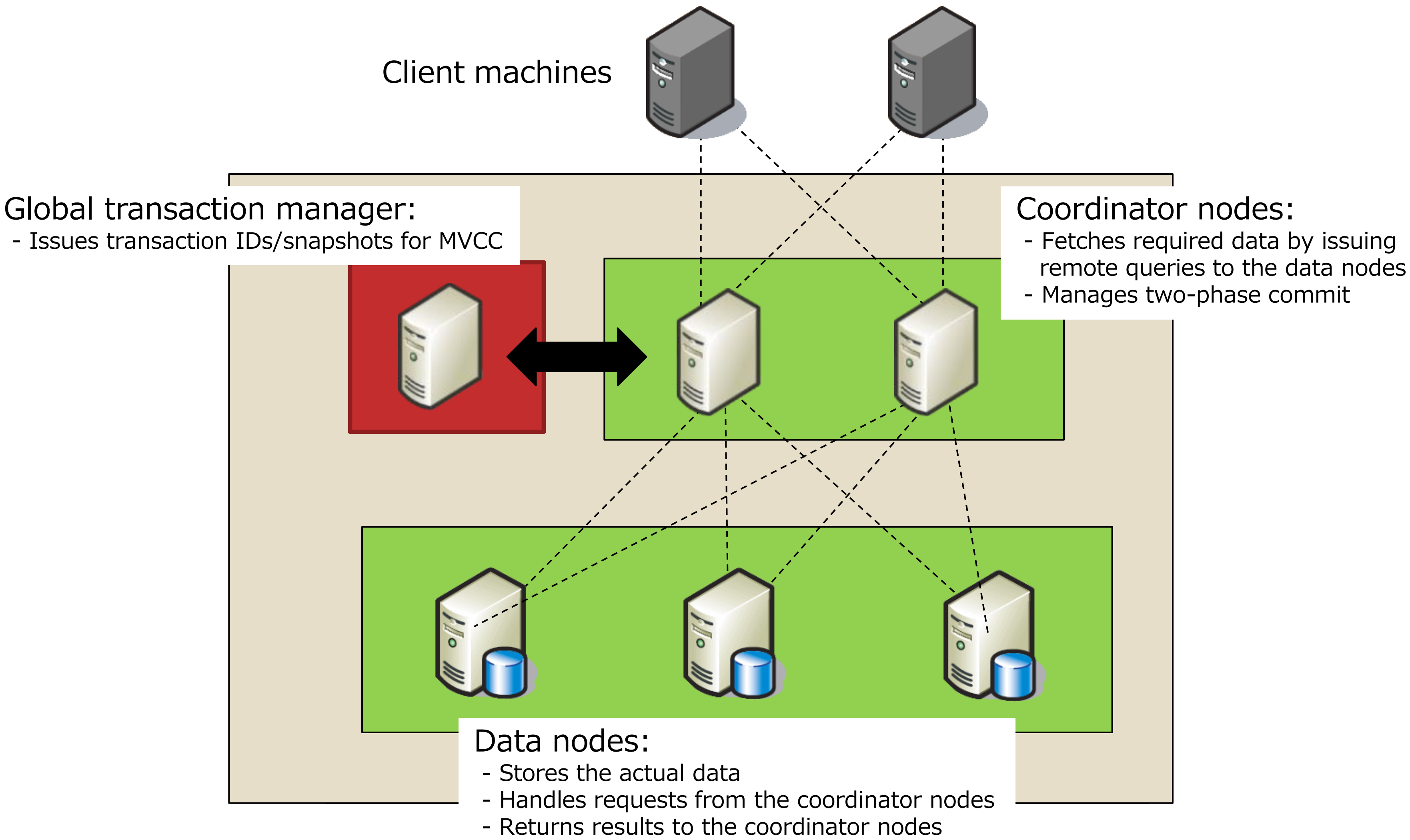
Sharding at the Database Layer

- **PostgreSQL external projects offer transparent sharding at the database layer:**
  - Postgres-XC
  - Postgres-XL
  - Postgres Pro
  - Citus DB

Posgres-XC: Overview

- Developed by NTT and EnterpriseDB (completed in 2014)
- Declarative table partitioning
- SQL-based remote database access
- Distributed transaction support
- Distributed join/aggregation support
- Cluster management
- Postgres-XL is a successor to Postgres-XC

Posgres-XC: Architechture



Client machines

Global transaction manager:
- Issues transaction IDs/snapshots for MVCC

Coordinator nodes:
- Fetches required data by issuing remote queries to the data nodes
- Manages two-phase commit

Data nodes:
- Stores the actual data
- Handles requests from the coordinator nodes
- Returns results to the coordinator nodes

Lessons Learned from Postges-XC

- **Good**
  - Provides cutting-edge technologies for sharding
- **Not good**
  - Difficult to maintain stable quality with limited resources
  - Difficult to date with the PostgreSQL source code with limited resources
- **What we believe is**
  - Built-in sharding for PostgreSQL is the right way to go

Outline

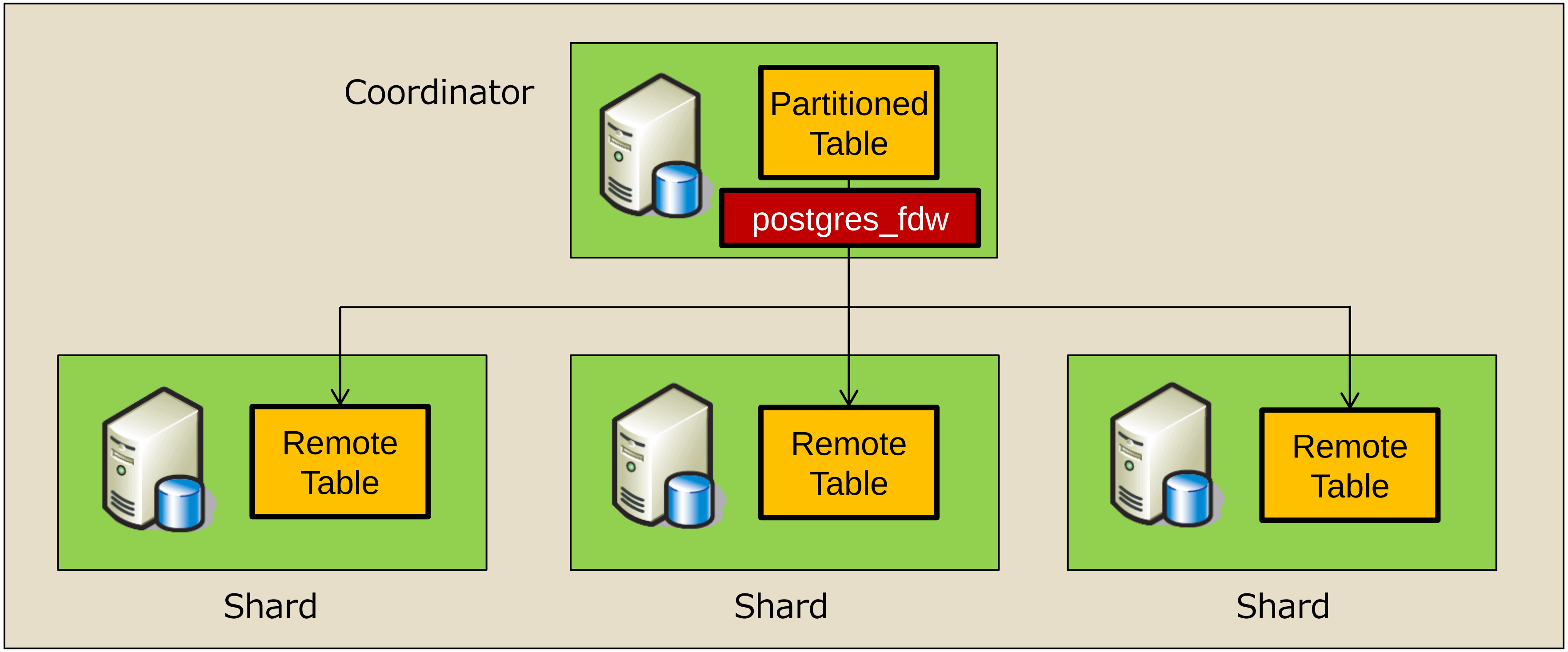
- Database Sharding
- **Built-in Sharding for PostgreSQL**
- Core Features for Built-in Sharding
- Demonstrations
- Concluding Remarks

Building blocks for built-in sharding

- **What we have achieved as of PostgreSQL 10**
  - Declarative table partitioning
    - Added in PostgreSQL 10
    - Spreads data into partitions across multiple shards
    - Don’t need cumbersome setting for that anymore
    - Opens the way to many kinds of optimizations
  - SQL-based remote database access
    - Provided by Foreign Data Wrapper (FDW)
    - Pushes database operations down to shards
      - Join pushdown in PostgreSQL 9.6
      - Aggregation pushdown in PostgreSQL 10
- **What we are planning to achieve**
  - Distributed transaction support
  - Smart query planning/execution

Built-in Sharding for PostgreSQL

- **Basic architecture**

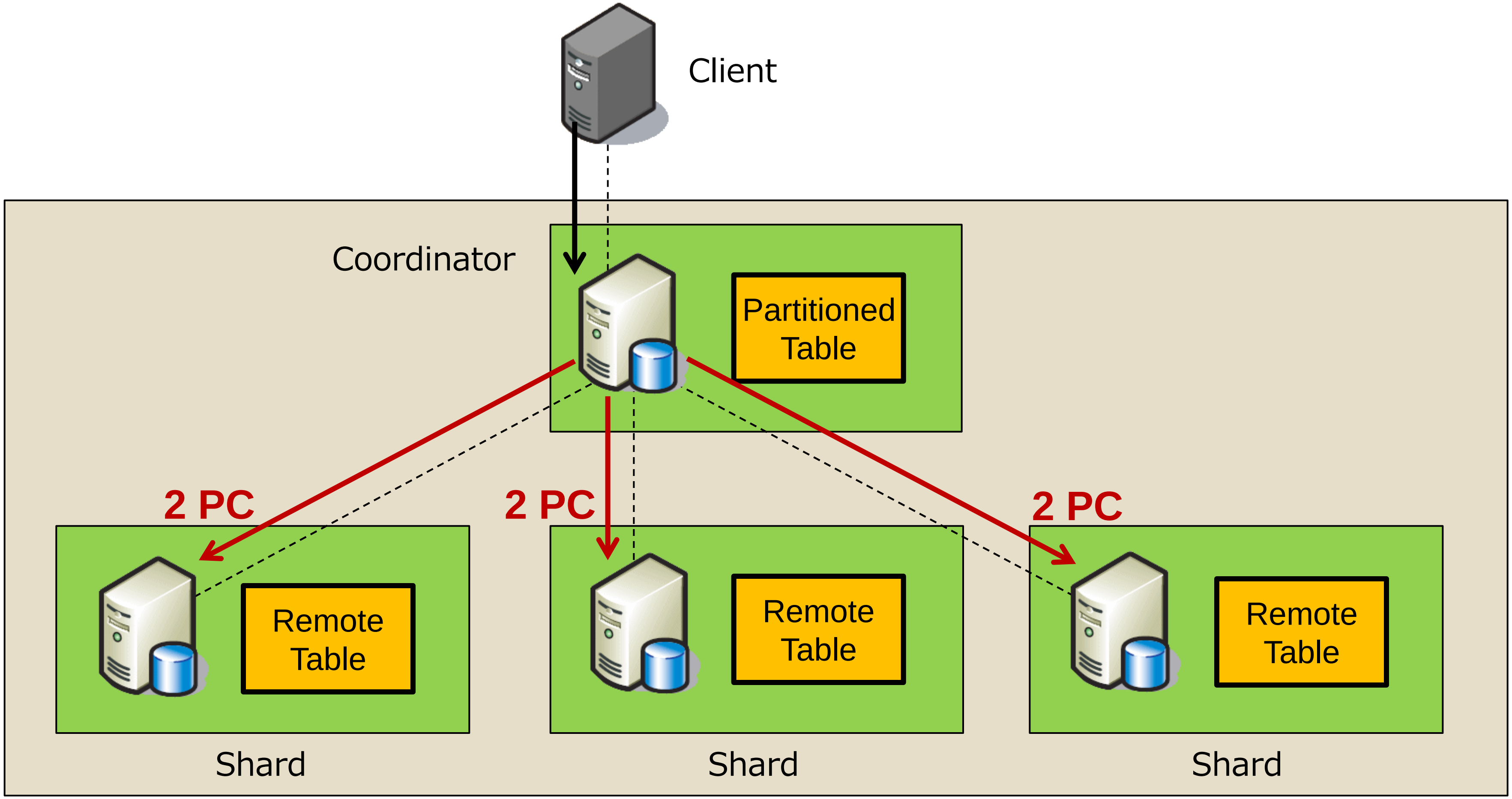


Towards OLTP/OLAP on built-in sharding

- **Missing pieces**
  - OLTP: Distributed transaction support
    - Extend transaction manager to support atomic commit/visibility
  - OLAP: Smart query planning/execution
    - Make planner more partitioning-aware
      - Distributed join/aggregation support
    - Integrate with logical replication
    - Improve executor to achieve parallelism on shards

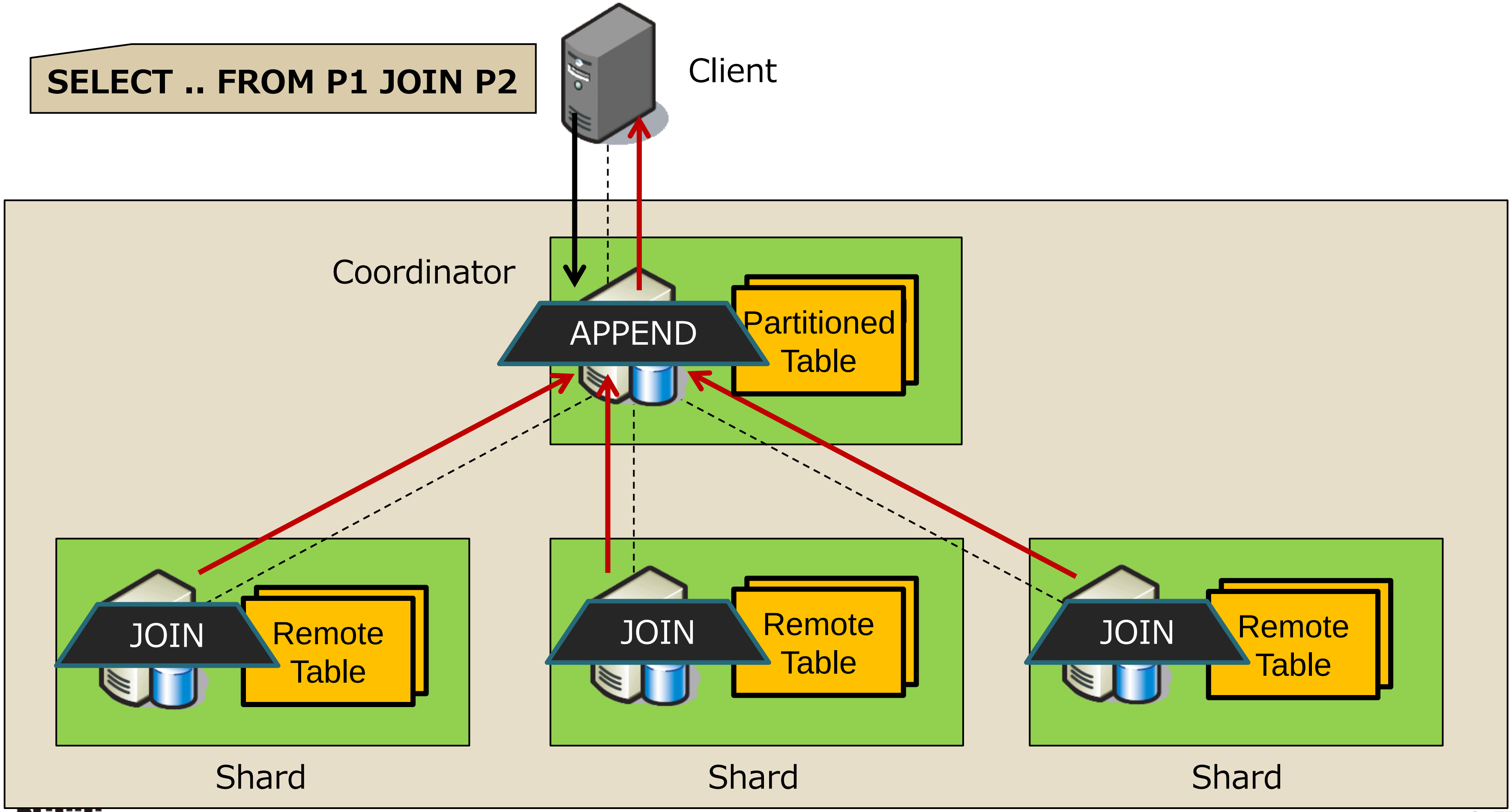
Extend Transaction Manager

- **Atomic commit**
  - Keeps transaction atomicity across shards



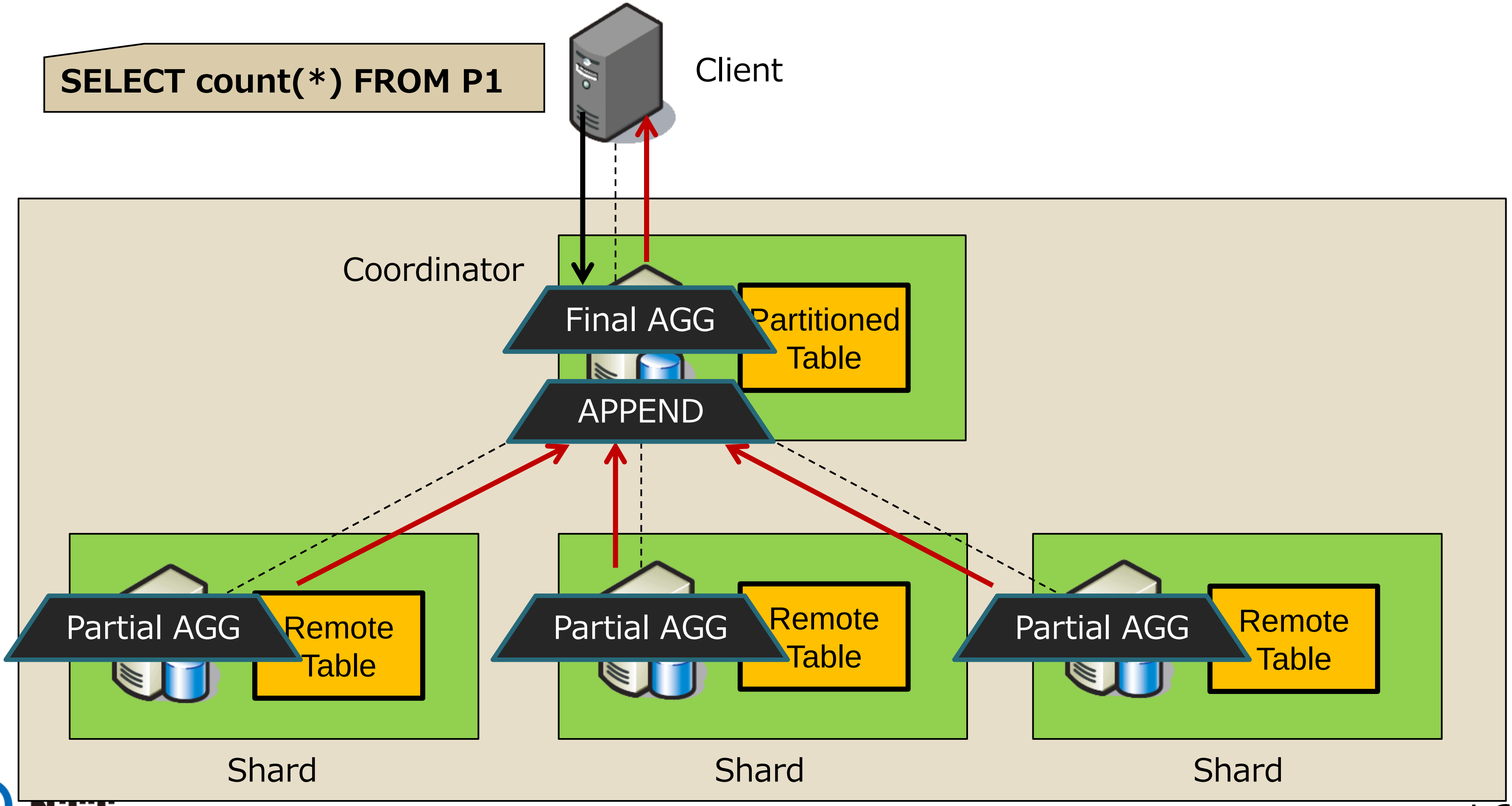
Make Planner More Partitioning-Aware

- **1. Partition-wise join**
  - Reduces cross-shard computation



Make Planner More Partitioning-Aware

- **2. Partition-wise aggregation**
  - Reduces cross-shard computation



Our Roadmap

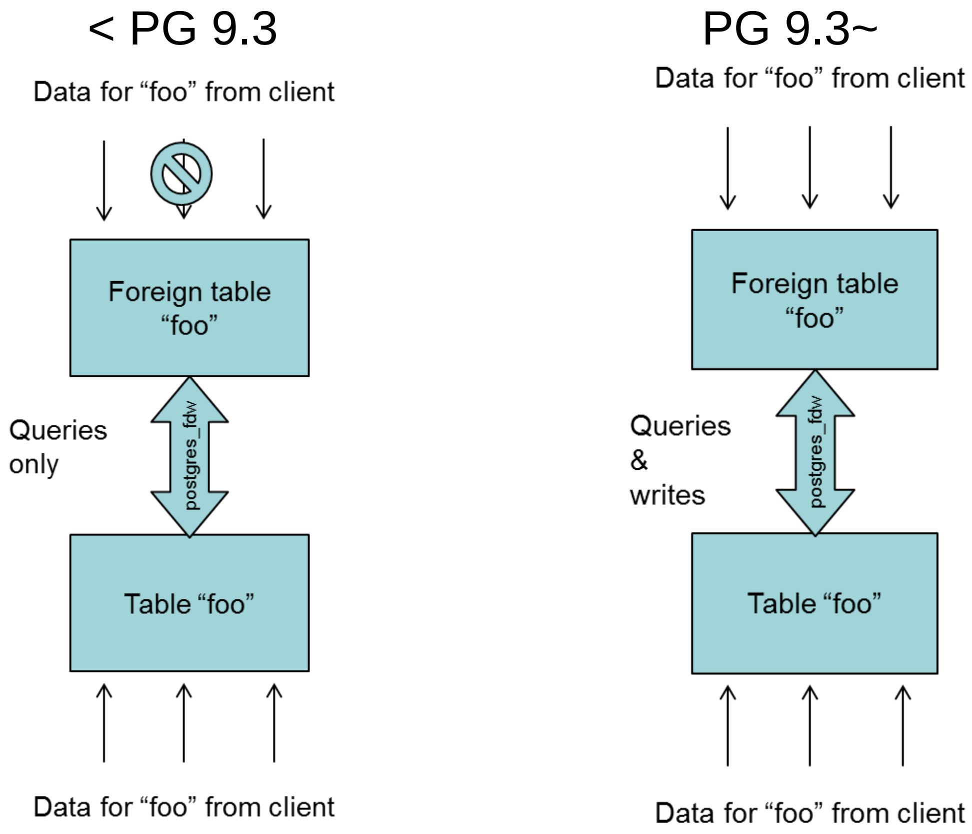
- **PostgreSQL 11**
  - (OLTP) Atomic commit
  - (OLAP) Partition-wise join/aggregation
- **PostgreSQL 12+**
  - (OLTP) Atomic visibility
  - (OLAP) Parallelism on shards

Outline

- Database Sharding
- Built-in Sharding for PostgreSQL
- **Core Features for Built-in Sharding**
- Demonstrations
- Concluding Remarks

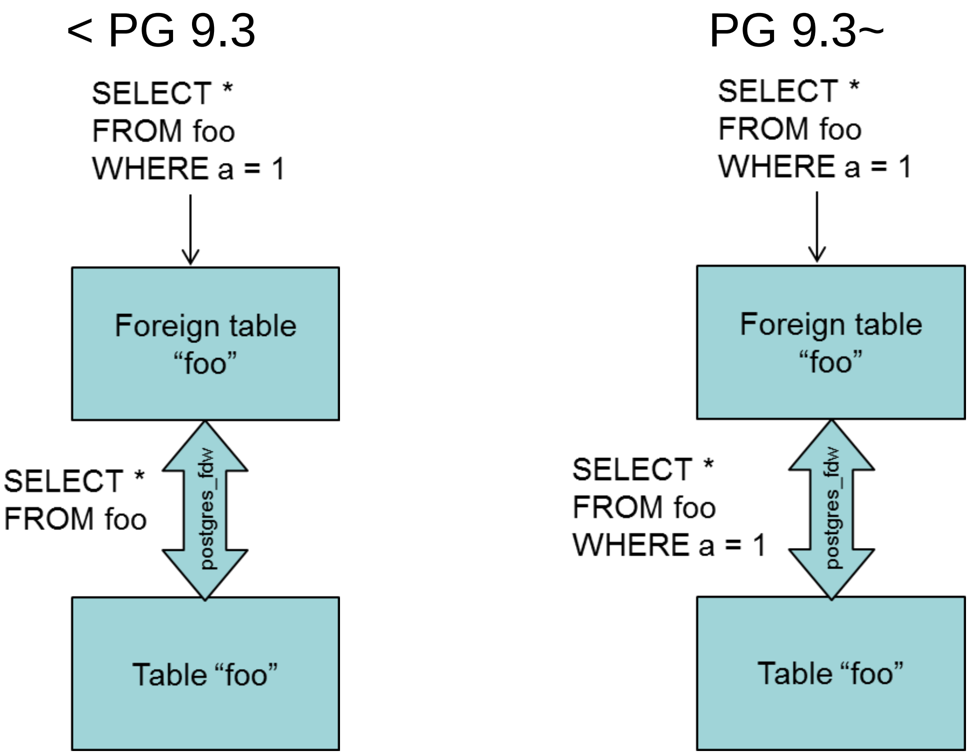
Progress up to PostgreSQL 10

- **PostgreSQL 9.3**
  - Writable foreign tables



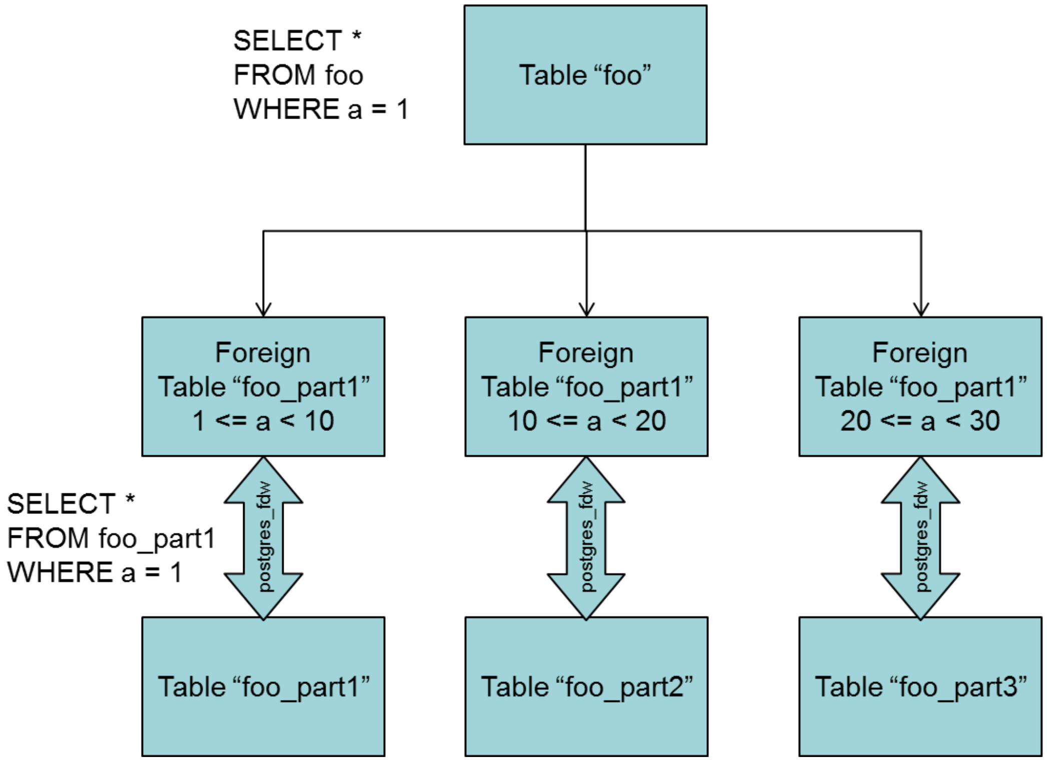
Progress up to PostgreSQL 10

- **PostgreSQL 9.3**
  - WHERE condition pushdown, etc.



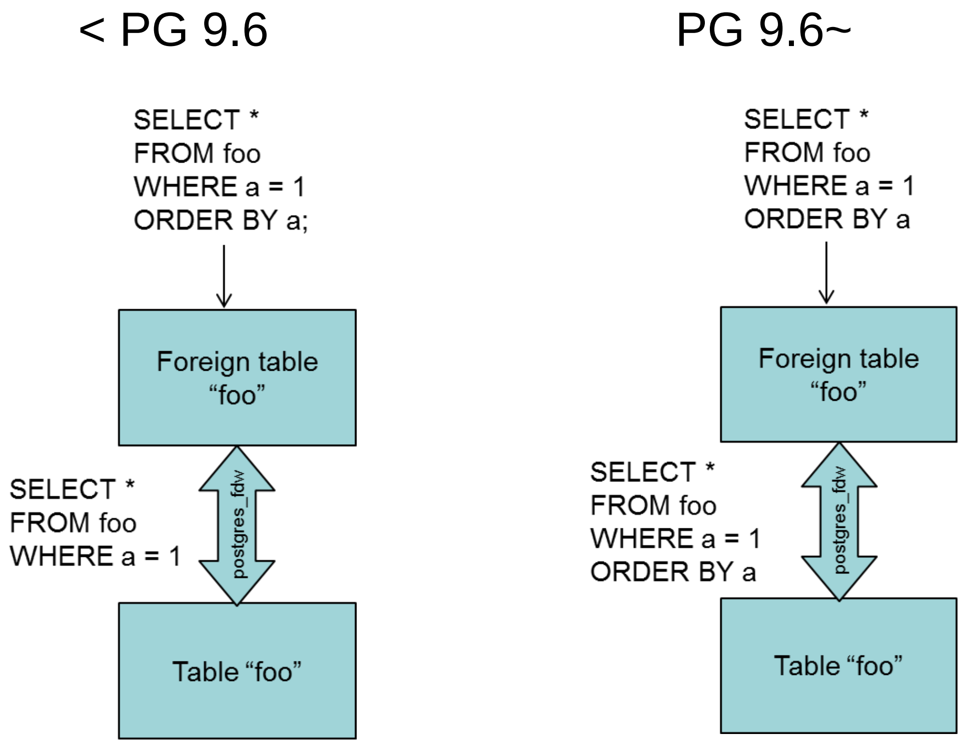
Progress up to PostgreSQL 10

- **PostgreSQL 9.5**
  - **Foreign table inheritance** as one of the first pieces of infrastructure to implement built-in sharding



Progress up to PostgreSQL 10

- **PostgreSQL 9.6**
  - **Sort pushdown** to get data in desired order from the remote server



< PG 9.6

SELECT *
FROM foo
WHERE a = 1
ORDER BY a;

Foreign table
“foo”

postgres_fdw

SELECT *
FROM foo
WHERE a = 1

Table “foo”

PG 9.6~

SELECT *
FROM foo
WHERE a = 1
ORDER BY a

Foreign table
“foo”

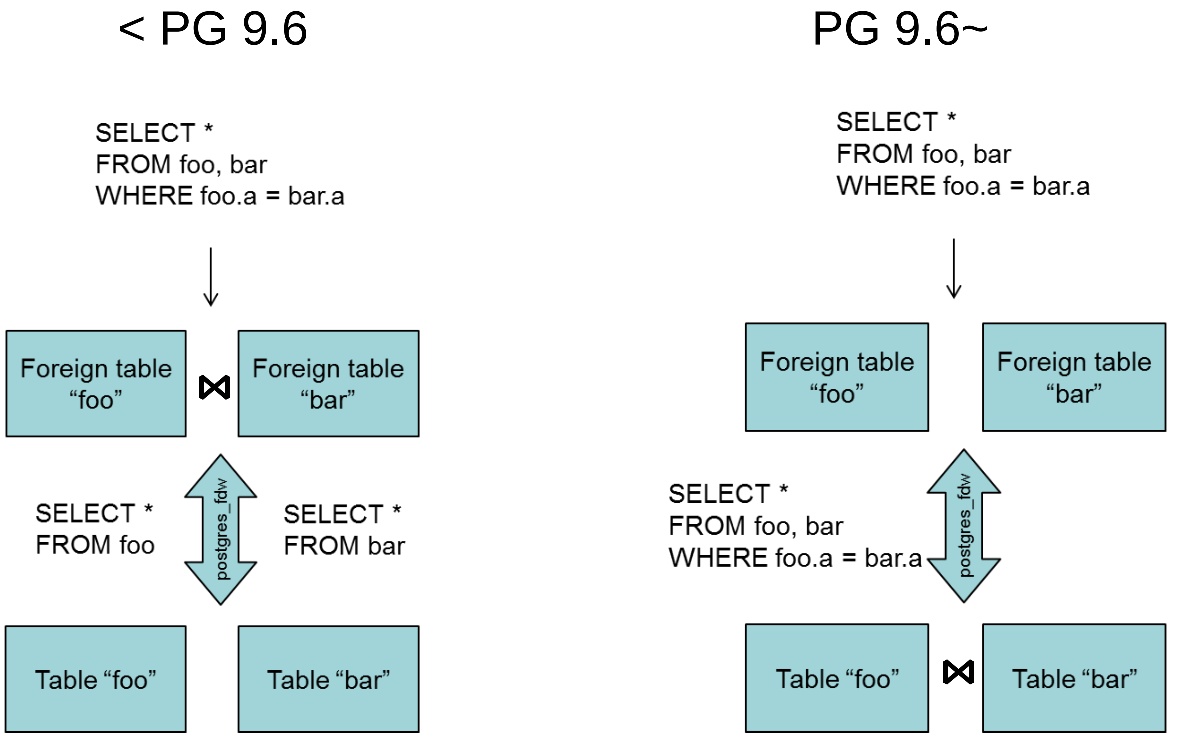
postgres_fdw

SELECT *
FROM foo
WHERE a = 1
ORDER BY a

Table “foo”

Progress up to PostgreSQL 10

- **PostgreSQL 9.6**
  - **Join pushdown** to remotely join tables known to be on the same remote server



< PG 9.6

SELECT *
FROM foo, bar
WHERE foo.a = bar.a

Foreign table “foo” ⋈ Foreign table “bar”

postgres_fdw

SELECT *
FROM foo

SELECT *
FROM bar

Table “foo”

Table “bar”

PG 9.6~

SELECT *
FROM foo, bar
WHERE foo.a = bar.a

Foreign table “foo”

Foreign table “bar”

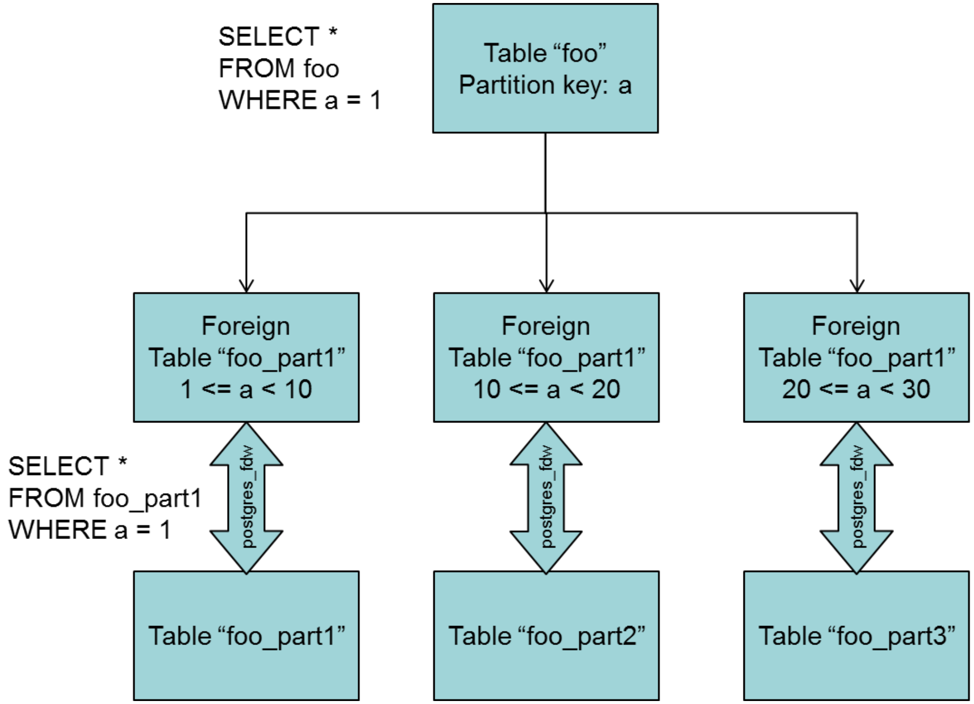
postgres_fdw

SELECT *
FROM foo, bar
WHERE foo.a = bar.a

Table “foo” ⋈ Table “bar”

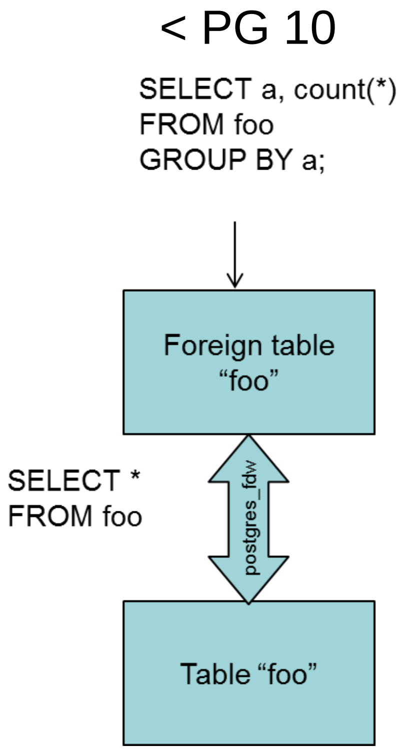
Progress up to PostgreSQL 10

- **PostgreSQL 10**
  - **Declarative table partitioning** where individual partitions can be foreign tables



Progress up to PostgreSQL 10

- **PostgreSQL 10**
  - **Aggregate pushdown** to perform grouped or non-grouped aggregates on the remote server



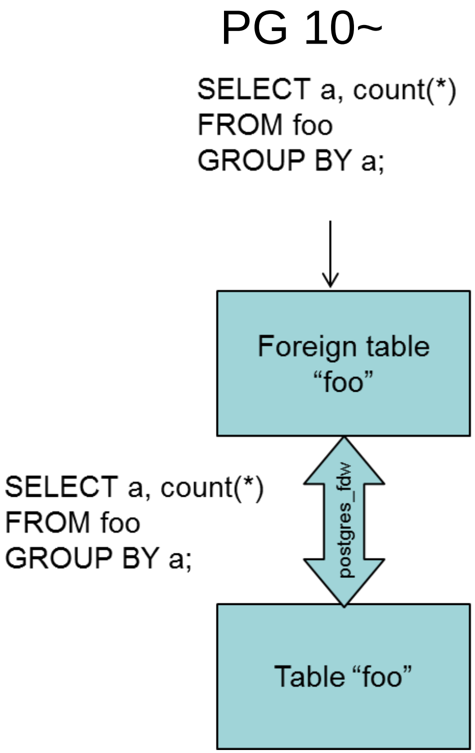
< PG 10

SELECT a, count(*)
FROM foo
GROUP BY a;

Foreign table "foo"

SELECT *
FROM foo

Table "foo"



PG 10~

SELECT a, count(*)
FROM foo
GROUP BY a;

Foreign table "foo"

SELECT a, count(*)
FROM foo
GROUP BY a;

Table "foo"

New in PostgreSQL 11

- **Basic features**
  - Hash partitioning
  - Tuple routing for foreign partitions
- **Distributed join/aggregation support**
  - Partition-wise join/aggregation
- **Distributed transaction support**
  - Atomic commit

Hash Partitioning

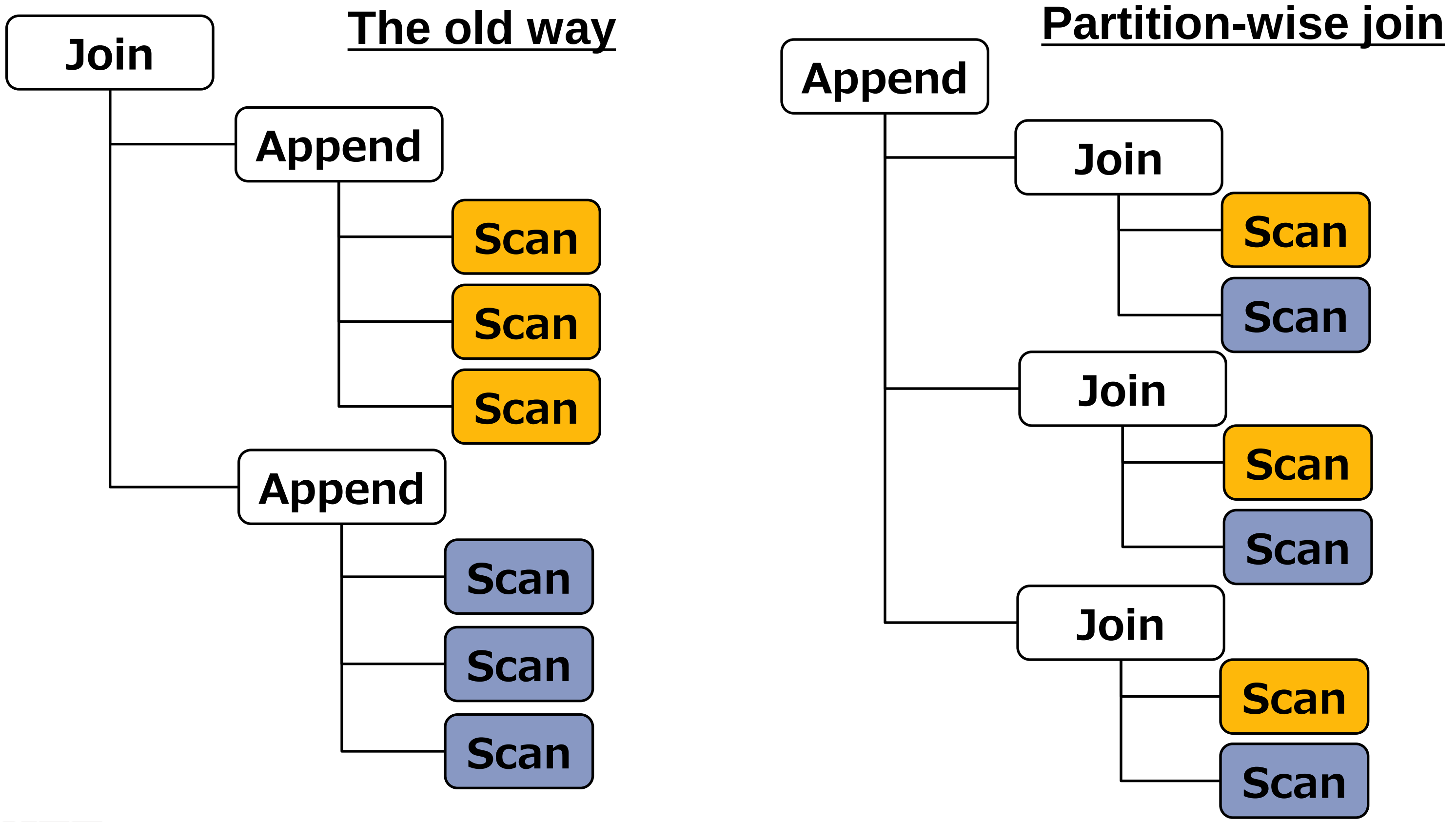
- Each partition is created by specifying a modulus and a remainder
- The data is uniformly distributed across all partitions

```
CREATE TABLE blogs (id int, title text, contents text)
        PARTITION BY hash (id);

CREATE TABLE blogs_1 PARTITION OF blogs
        FOR VALUES WITH (modulus 4, remainder 0);
CREATE TABLE blogs_2 PARTITION OF blogs
        FOR VALUES WITH (modulus 4, remainder 1);
CREATE TABLE blogs_3 PARTITION OF blogs
        FOR VALUES WITH (modulus 4, remainder 2);
CREATE TABLE blogs_4 PARTITION OF blogs
        FOR VALUES WITH (modulus 4, remainder 3);
```

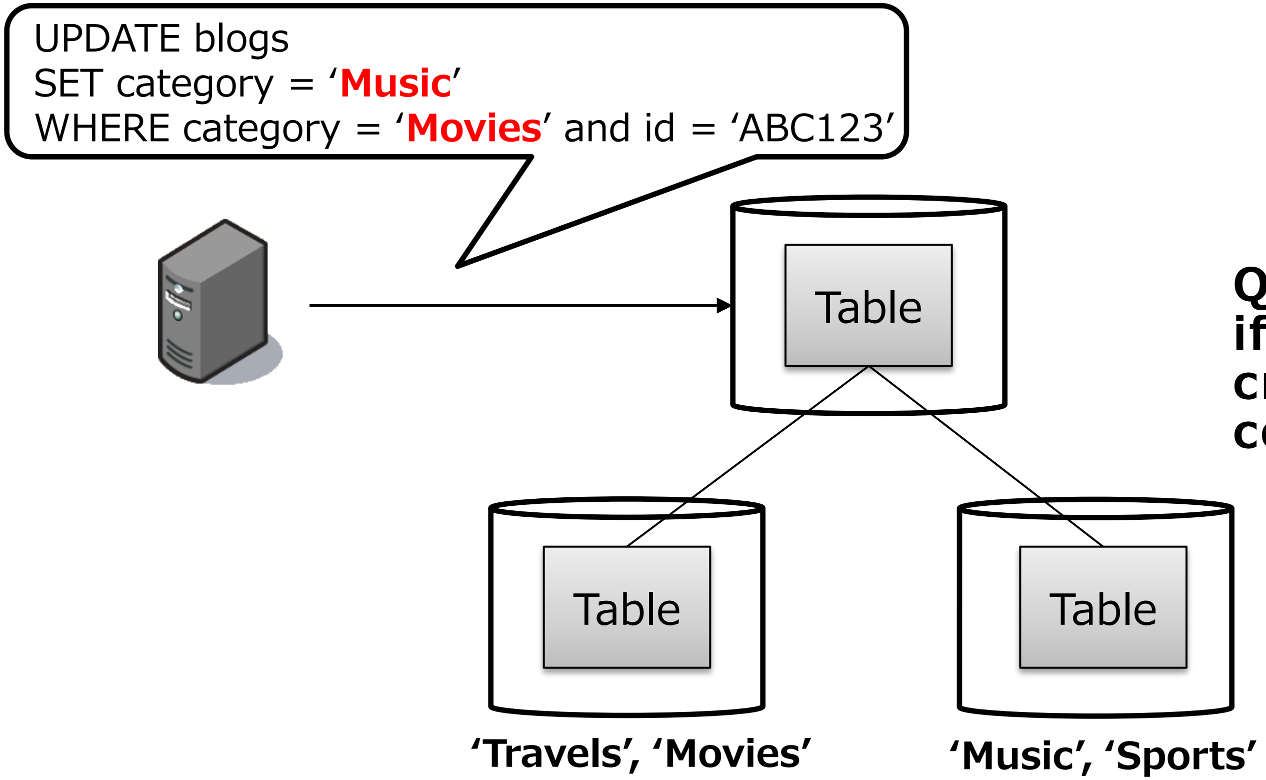
Partition-wise Join/Aggregation

- A join between partitioned tables to be performed by joining the matching partitions
- In built-in sharding, joins are executed on each shard servers



Atomic Commit

- Distributed transaction is either committed/aborted on ALL remote servers
- In the ongoing patch, we employ two-phase commit protocol to achieve atomic commit



```
UPDATE blogs
SET category = ‘Music’
WHERE category = ‘Movies’ and id = ‘ABC123’
```

Q. What happen if one shard crashes during commit?

Outline

- Database Sharding
- Built-in Sharding for PostgreSQL
- Core Features for Built-in Sharding
- **Demonstrations**
- Concluding Remarks

Demo: Schema

- Two tables to store user action data on a travel-related application
- Column highlighted in green is the partition/shard key

Table “flight_bookings”

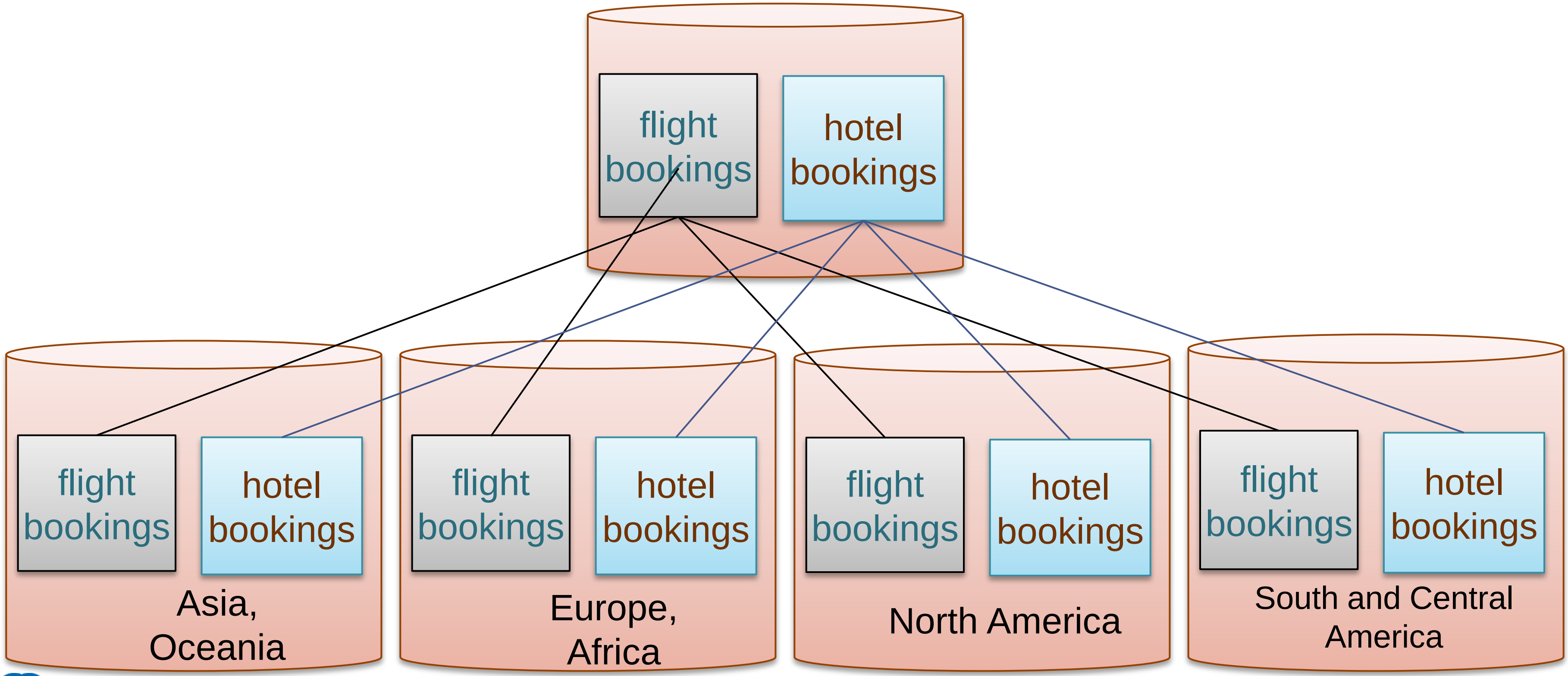
Column	Type
id	integer
user_id	integer
booked_at	timestamp without time zone
from_city	text
from_continent	text
to_city	text
to_continent	**text**

Table “hotel_bookings”

Column	Type
id	integer
user_id	integer
booked_at	timestamp without time zone
city_name	Text
continent	**text**
flight_id	integer REFERENCES flight_bookings (id)

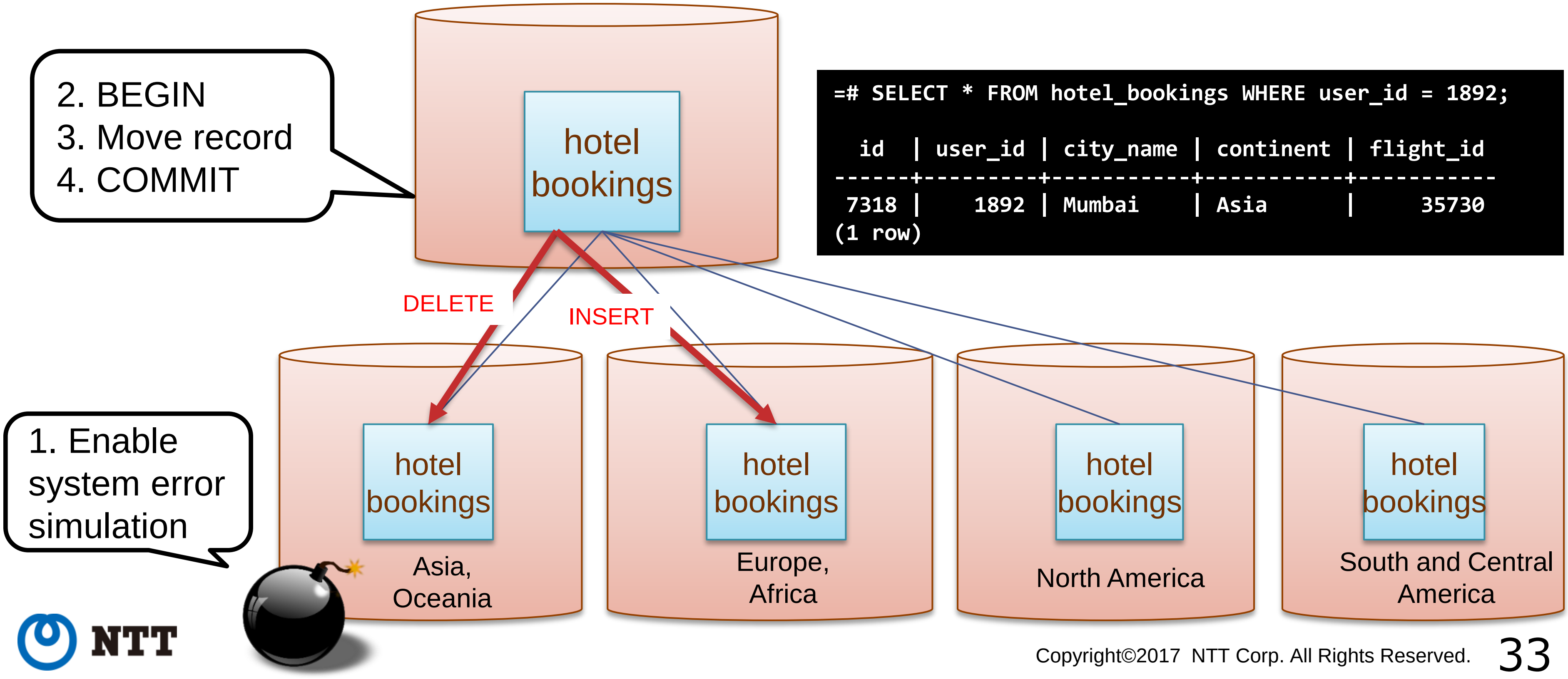
Demo: Data Layout

- 4 partitions of each table
- Since both tables are partitioned on the column containing same set of data in each table, corresponding tables on a given shard contain matching data in that column



Demo: Atomic Commit

- Transaction initiated by a user action to change the hotel booking for a given flight from ‘Mumbai’ to ‘Moscow’.
  - Causes the record to move from ‘Asia’ shard to ‘Europe’ shard
  - During commit phase, the ‘Asia’ shard fails
  - Whole transaction is aborted, so no data change occurs □



2. BEGIN
3. Move record
4. COMMIT

hotel bookings

```
=# SELECT * FROM hotel_bookings WHERE user_id = 1892;

  id  | user_id | city_name | continent | flight_id
------+---------+-----------+-----------+-----------
 7318 |    1892 | Mumbai    | Asia      |     35730
(1 row)
```

DELETE

INSERT

1. Enable system error simulation

hotel bookings

Asia, Oceania

hotel bookings

Europe, Africa

hotel bookings

North America

hotel bookings

South and Central America

Demo: OLAP Query

- Query to get per-continent count of flights that have a hotel booking associated with it

```
SELECT     F.to_continent, count(*)
FROM       flight_bookings F, hotel_bookings H
WHERE      F.to_continent = H.continent AND
           F.id = H.flight_id AND
           F.booked_at > ‘2017-10-01’
GROUP BY   F.to_continent;
```

Outline

- Database Sharding
- Built-in Sharding for PostgreSQL
- Core Features for Built-in Sharding
- Demonstrations
- **Concluding Remarks**

Concluding Remarks

- **Built-in sharding**
  - Towards OLTP/OLAP on built-in sharding
  - PostgreSQL 11
    - (OLTP) Atomic commit
    - (OLAP) Partition-wise join/aggregation
  - PostgreSQL 12+
    - (OLTP) Atomic visibility
    - (OLAP) Parallelism on shards
- **Remaining work**
  - Logical replication integration
  - Orchestration
  - Monitoring
  - High availability

References

- R. Haas: From FDWs to Sharding, PGCon 2015
- S. Riggs: Logical Replication, Sharding & Multimaster Clusters, PGConf.ASIA 2016
- M. Sawada: Built-in Sharding update and future, PGConf.Russia 2017
- A. Langote, E. Fujita, K. Horiguchi, and M. Sawada: Towards Built-in Sharding in Community PostgreSQL, PGCon 2017

Thank You

- **Any questions?**