2ndQuadrant®
PostgreSQL

Towards full ACID Distributed Transactions Support with Foreign Data Wrapper

Nikhil Sontakke

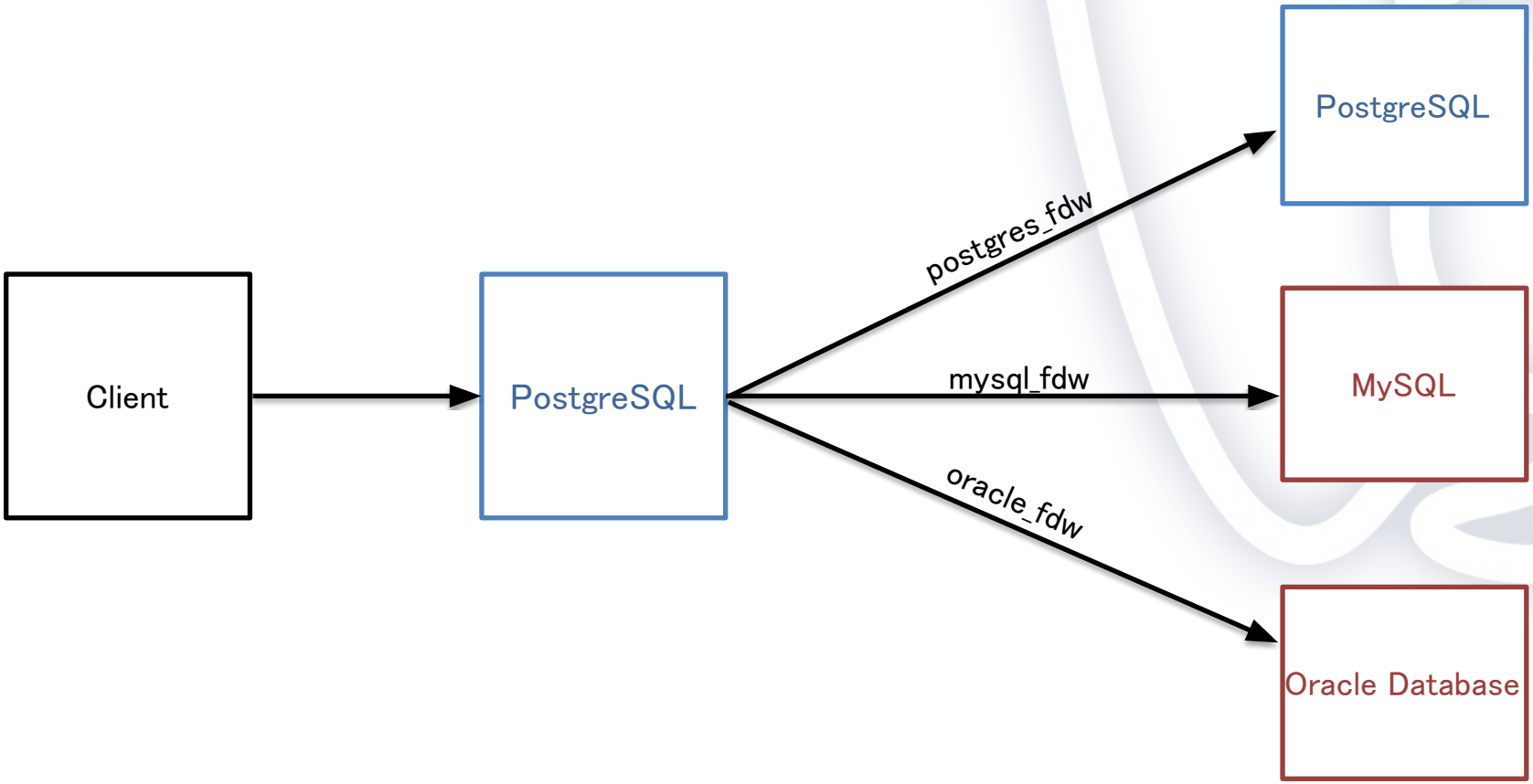
WhoAmI?

- Working on PostgreSQL code since 2006
- Part of a few startups
  - OyeStyle
  - StormDB (acquired by Translattice, Inc.)
  - SecureDB
- Co-Organizer of PGConf India :-)
- PostgreSQL Consultant, 2ndQuadrant

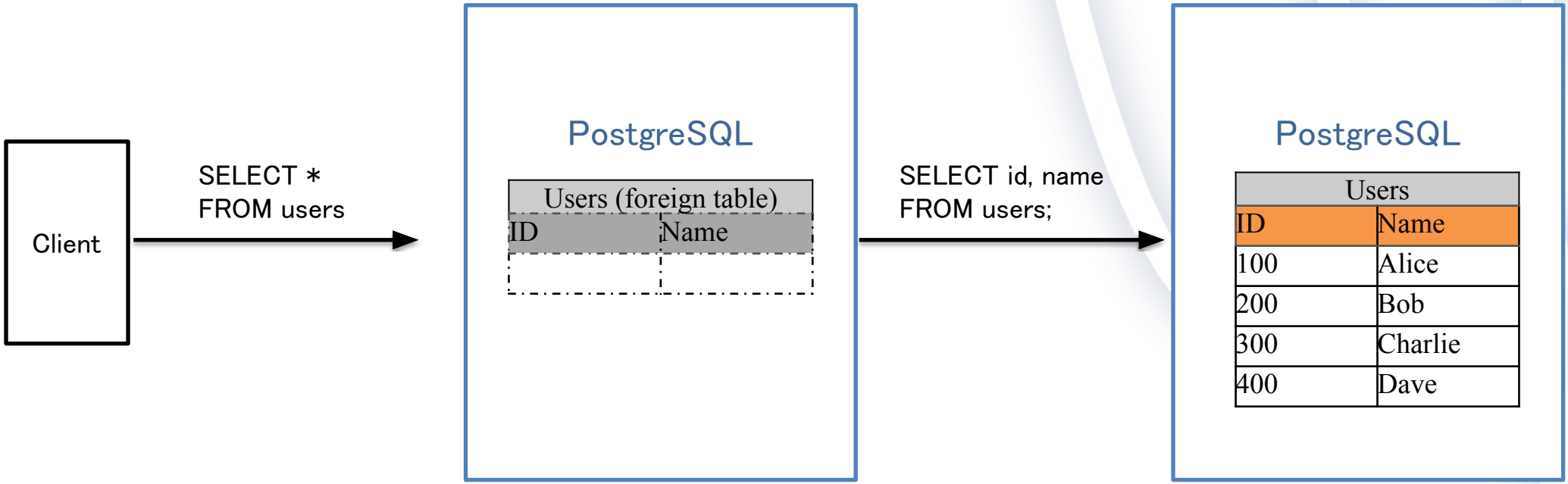
What is Foreign Data Wrapper?

- Implementation of the SQL/MED
- Access data that resides outside PostgreSQL using regular SQL queries
- Foreign tables
- Pluggable architecture

Foreign Data Wrapper



Foreign Data Wrapper in details



Foreign Data Wrapper with Reads

- Most of the operations can be pushed down to the remote server
  - Joins
  - Aggregations
  - Sorts etc

Foreign Data Wrapper with Writes

- Writable foreign tables
  - PostgreSQL version 9.3 and later

Transactions: Anomalies

The phenomena due to concurrent transactions which are prohibited at various levels are:

- `Dirty Read`

A transaction reads data written by a concurrent uncommitted transaction.

- `Non-repeatable Read`

A transaction re-reads data it has previously read and finds that data has been modified by another transaction (that committed since the initial read).

- `Phantom Read`

A transaction re-executes a query returning a set of rows that satisfy a search condition and finds that the set of rows satisfying the condition has changed due to another recently-committed transaction.

- `Serialization anomaly`

The result of successfully committing a group of transactions is inconsistent with all possible orderings of running those transactions one at a time.

Consistency and Isolation: Transactions

- PostgreSQL uses MVCC (Multi-Version Concurrency Control) with SI (Snapshot Isolation)
- Snapshot is a view of the database at a given point in time. PostgreSQL selects the appropriate version of an item by applying **visibility check rules**.

  e.g: 10:20:10,14,15

Transaction Isolation Levels

Isolation Level	Nonrepeatable Read	Phantom Read	Serialization Anomaly
Read uncommitted	Possible	Possible	Possible
Read committed	Possible	Possible	Possible
Repeatable read	Not possible	Allowed, but not in PG	Possible
Serializable	Not possible	Not possible	Not possible

FDW and Transactions

- FDW plugin is responsible for transaction management on the remote nodes (foreign transaction)
  - begin, commit, rollback and savepoint
- Foreign transaction termination uses XactCallback, which is called BEFORE committing the local transaction

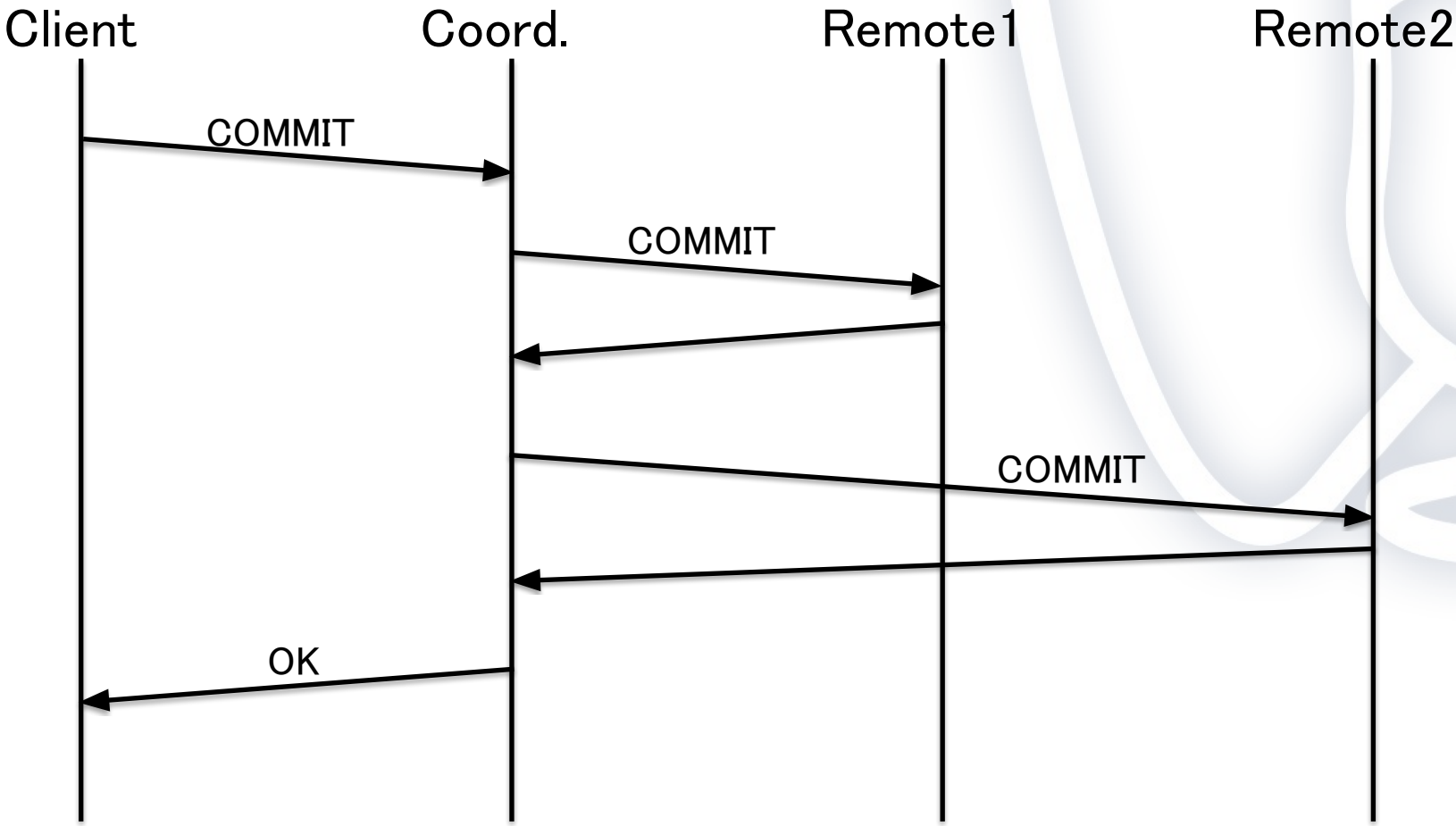
postgres_fdw's Transaction Management

- Open a foreign transaction when FDW accesses the remote table for the **first** time within the local transaction
- Foreign transaction uses SERIALIZABLE when the local transaction has SERIALIZABLE level. Otherwise uses REPEATABLE READ
- This ensures that if a query performs multiple tables scans on the remote server, it will get snapshot-consistent results for all the scans

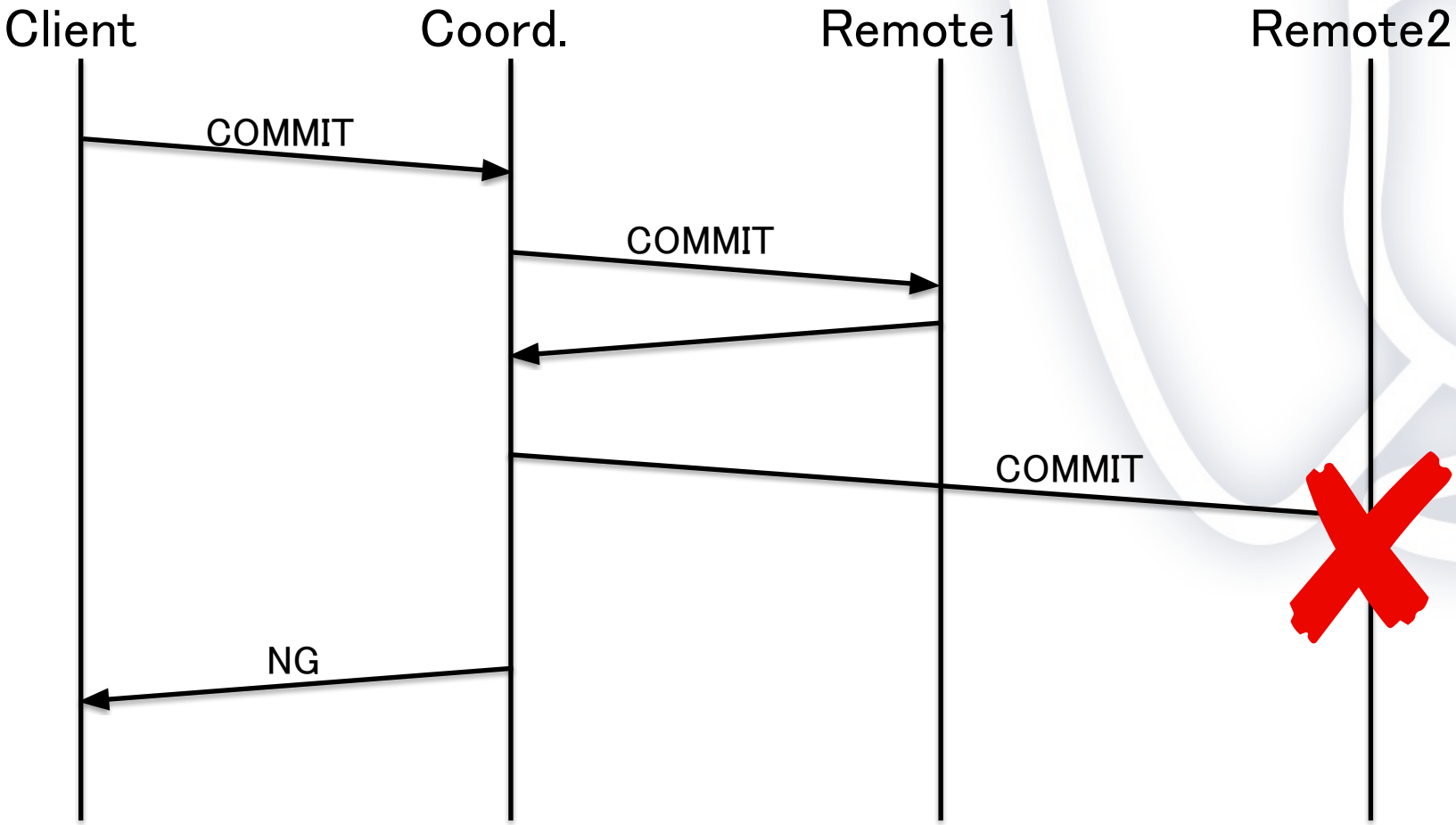
Atomic Commit Problem

- Foreign transactions are committed one by one before the local transaction commits
- If a remote server crashes during the commit, some transactions are committed whereas others are not (It's the same when the local node crashes)
- There is no guarantee that all servers (including local) commit or rollback

FDW Transaction Commit Sequence



FDW Transaction Commit Sequence



Solution: Two-Phase Commit

- A type of atomic commitment protocol
- Fail-stop model
- Consists of two phases: Prepare and Commit

1. The coordinator sends PREPARE to all participants

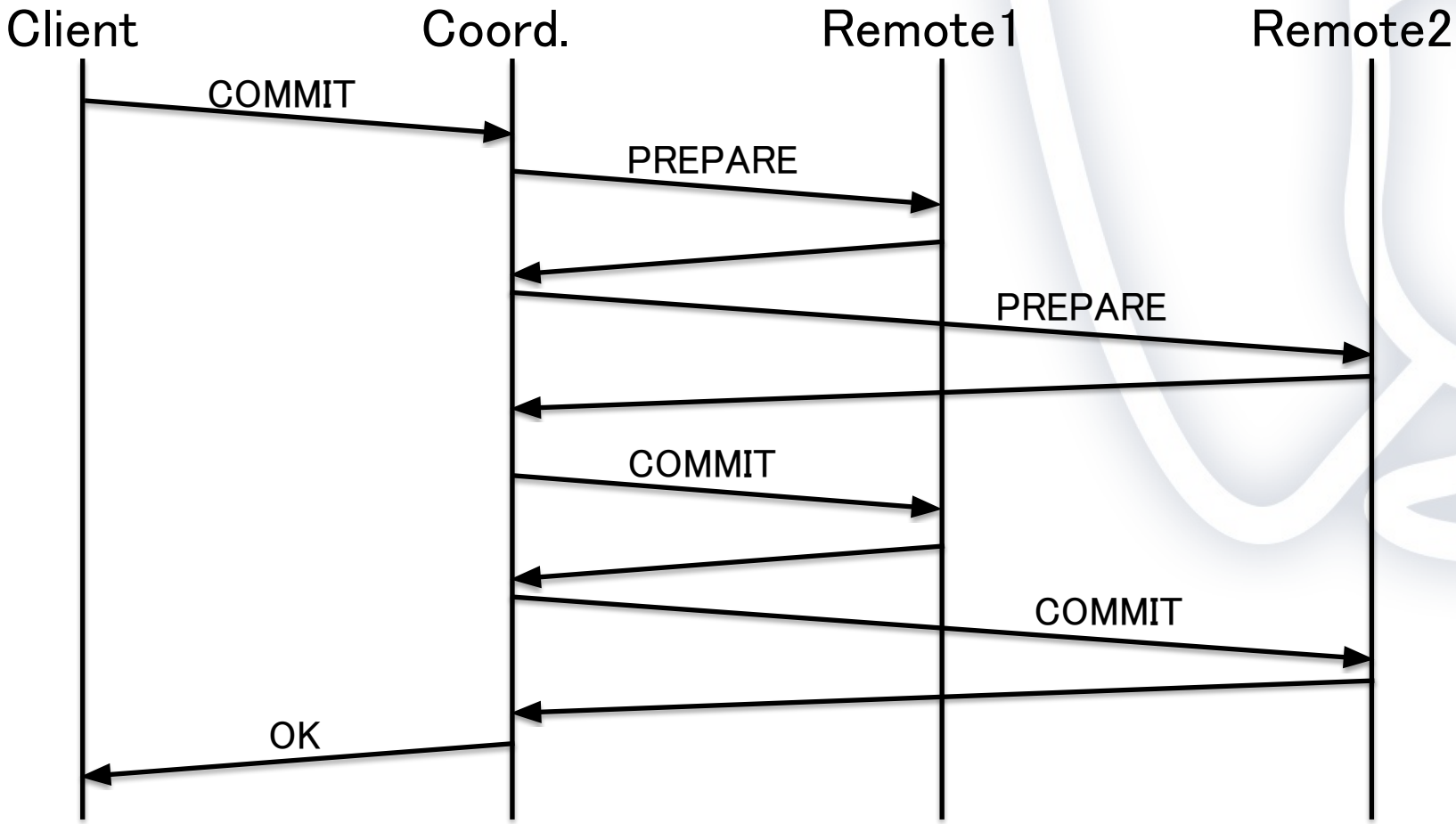
2. The coordinator sends COMMIT to all participants if and only if ALL participants sent OK in the PREPARE phase

3. Otherwise the coordinator sends ROLLBACK to all the participants

2PC for FDW

- Work in Progress (yet to make it into PG code)
- First proposal in 2015. Patches submitted till late 2018
- The core manages remote transactions
- Introduces new FDW API for transaction management
- Transaction involving foreign transactions implements commit via following steps:
  1. Prepare all foreign transactions
  2. Commit locally
  3. Commit all foreign transactions

2PC for FDW


Client
Coord.
Remote1
Remote2
COMMIT
PREPARE
PREPARE
COMMIT
COMMIT
OK

2PC for FDW

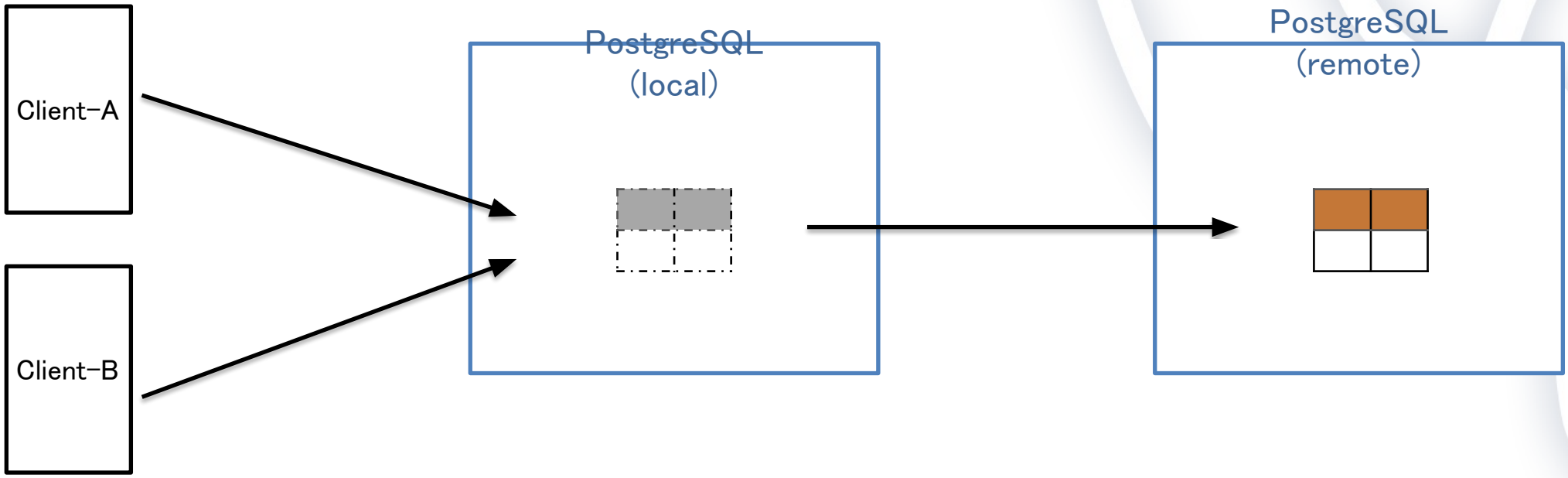
- The core persists information about foreign transactions to the disk via WAL records
- **Foreign Transaction Resolver**:
  When the coordinator/participant crashes during the prepare phase, the foreign transaction information is recovered and is rollbacked after restart
- When the coordinator/participant crashes during commit phase, if coordinator has committed locally, then foreign transactions that are in prepared state are committed after restart. Aborted otherwise.
- Solves the Atomic Commit problem

Atomic Visibility: Inconsistent Read

- One of the most important goals of FDW is that if the client uses PostgreSQL server with foreign servers then it needs to function the same way a single PostgreSQL server would do
- Unfortunately, current FDW doesn't work even if a transaction involves only one remote server

Inconsistent Read: Example 1

- Two clients concurrently read/write from/to remote table
- The regular table on the remote node has 100 rows
- Use READ COMMITTED isolation level



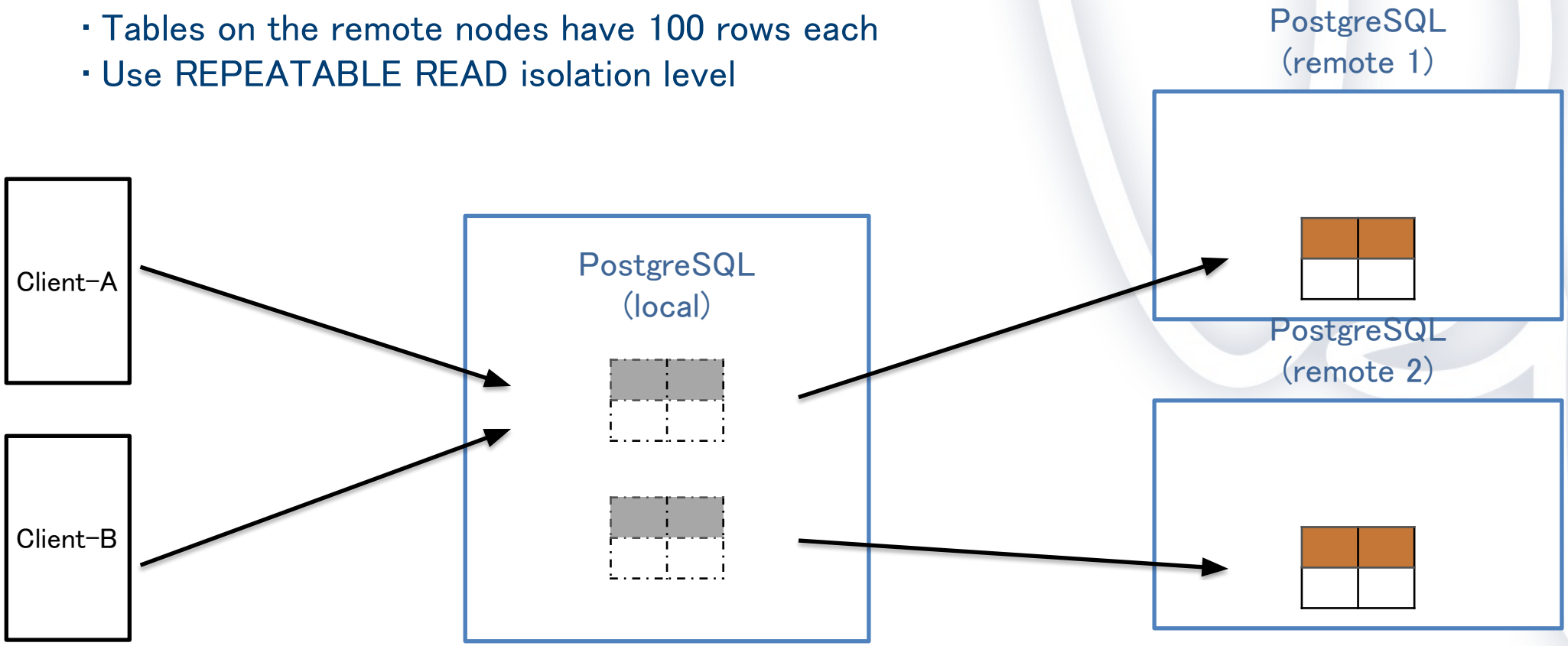
Inconsistent Read: Example 1

- The second read by Client A(#6) should return 90 if we were to use a single PostgreSQL server
- However since foreign transaction is opened with REPEATABLE READ, two reads (#2 and #6) return the same results

#	Local Client A	Local Client B
1	BEGIN /* remote REPEATABLE READ txn */	
2	SELECT count(*) FROM remote_tbl; count ———- 100	
3		BEGIN
4		DELETE FROM remote_tbl WHERE id < 10;
5		COMMIT
6	SELECT count(*) FROM remote_tbl; count ———- 100	
7	COMMIT	

Inconsistent Read: Example 2

- Two clients and two foreign servers
- Tables on the remote nodes have 100 rows each
- Use REPEATABLE READ isolation level



Inconsistent Read: Example 2

- The second read (#6) by client-A should return 100 if we were to use PostgreSQL server
- However because a transaction on the remote node is started when the local server touches it for the first time; the transaction on the remote node 2 is initiated at #6, therefore it returns 90.

#	Local Client A	Local Client B
1	BEGIN TRANSACTION ISOLATION LEVEL REPEATABLE READ;	
2	SELECT count(*) FROM tbl_on_remote1; count ———- 100	
3		BEGIN TRANSACTION ISOLATION LEVEL REPEATABLE READ;
4		DELETE FROM tbl_on_remote2 WHERE id < 10;
5		COMMIT
6	SELECT count(*) FROM tbl_on_remote2; /* remote REPEATABLE READ txn initiated */ count ———- 90	
7	COMMIT	

Atomic Visibility Solution: Global Snapshots

- Provide global consistent snapshots
- Global snapshots could be regular PostgreSQL snapshots (xmin, xmax and in-progress xids), or timestamp or (commit sequence number) CSN etc.

Global Transaction Manager: Examples

- Postgres-XL has a dedicated global transaction manager node (GTM node)
  - All coordinators have to access GTM to get a global consistent snapshot
- Google percolator has similar concept: Timestamp Oracle
  - Which can produce timestamps in a strictly increasing order
  - Timestamps coming from the timestamp oracle are used as the time when read/write operations happen
- But.. SPOF issues with GTMs

Clock-SI

- “Clock-SI: Snapshot Isolation for Partitioned Data Stores Using Loosely Synchronized Clocks” Jiaquing et al.
- Use physical time as CSN
- A transaction with snapshot from the future must wait until it becomes present in local time

Summary

- Foreign Data Wrapper is a powerful feature to access distributed data from across heterogenous data stores
- The biggest missing piece is transaction management:
  - Atomic commit and consistent reads
- Several ideas proposed and are WIP in the community

Acknowledgements

- Masahiko Sawada
- Ashutosh Bapat
- Postgresql hackers mailing list

2ndQuadrant
Mission Critical Databases

Website https://www.2ndquadrant.com/
Blog https://blog.2ndquadrant.com/
Email info@2ndQuadrant.com