Write-Scalable Shared-Nothing PostgreSQL Cluster

Today's Talk

- What is Postgres-XC?
  - Concept and Ultimate Goal
- How to achieve read/write scalability
- Postgres-XC component
  - Global Transaction Manager
  - Coordinator
  - Data Node
- Current Status and Evaluation
- Possible Applications
- Issues and Roadmap

What is Postgres-XC **NOT**?

- Not multi-master replication solution
- Not a read-balancing solution
- No native HA (yet)

What is Postgres-XC? (1)

- Write-scalable PostgreSQL cluster
  - More than 3.4 performance scalability with five servers, compared with pure PostgreSQL (DBT-1)
- Global multi-coordinator configuration
  - Any update to any master is visible from other masters immediately.

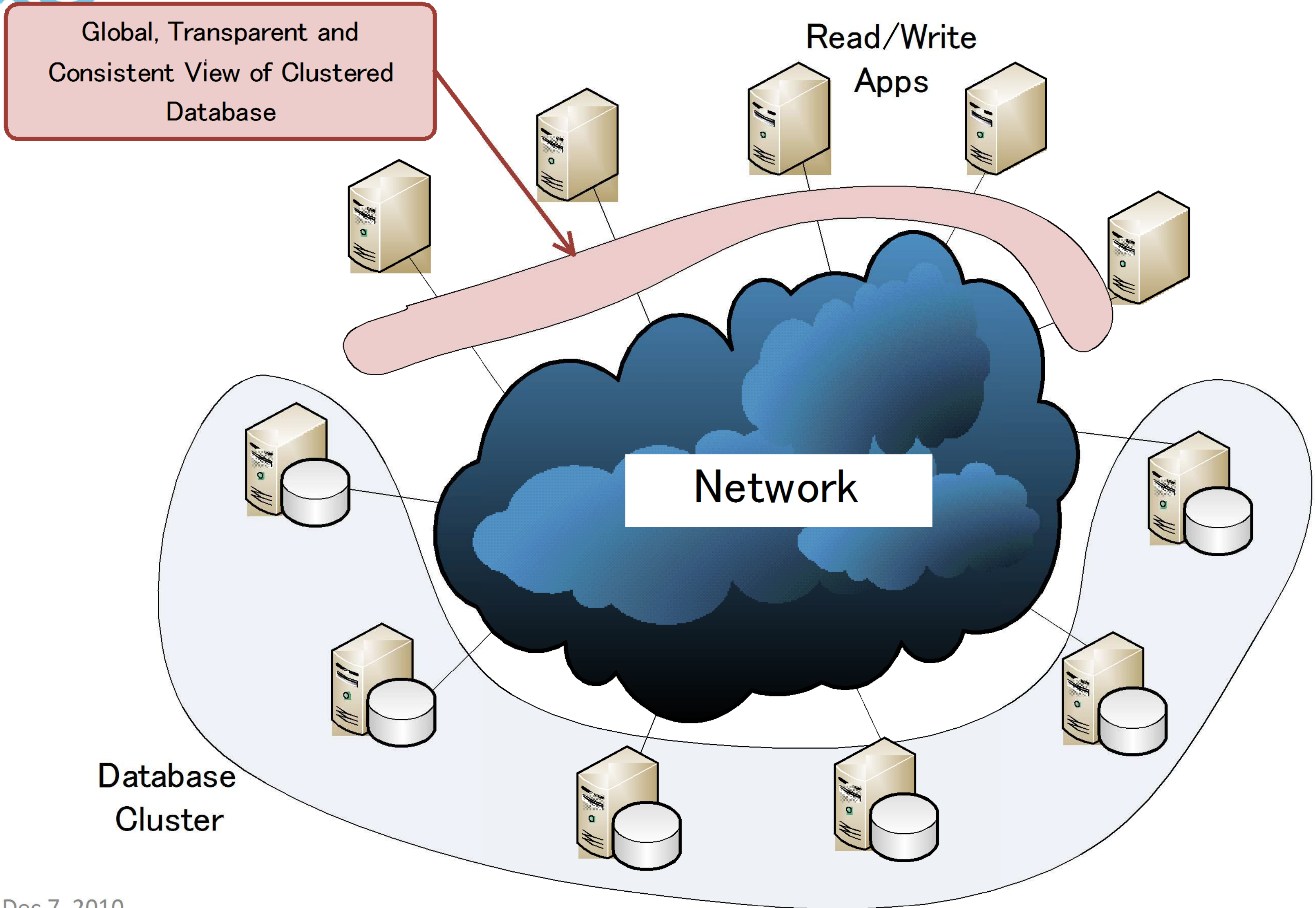
What is Postgres-XC? (2)

- Table location transparent
  - Tables can be replicated or distributed (partitioned or round robin)
  - Can continue to use the same applications.
  - No change in transaction handling.
- Based upon PostgreSQL
- Same API to Apps as PostgreSQL

Postgres-XC Applications

- Short transaction applications (DBT-1, DBT-2 etc.)
  - Transactions can be executed in parallel in multiple data nodes.
- Data warehouse (DBT-3 etc.)
  - Statement can be divided into several pieces executed in parallel in multiple data nodes.
    - (Complex statement handling still very primitive)

Goal of Postgres-XC



Why Write-Scalability?

- Other solutions can achieve some scalability with replicated read-only slaves, but does not help with writes
- Many applications could be write-traffic bottlenecked
  - Blogs, Social Networks
  - Mission critical systems like internet shopping site, telephone customer billing, call information and securities trading
- Application has to deal with such write bottleneck using multiple databases via sharding
  - Not distribution-transparent
  - Possible consistency issues
- As applications grow
  - It is desirable to make database distribution transparent for write operations too.

Why Shared-Nothing?

- Most Cost-Efficient
- Flexible to deploy
  - Can apply very simple to complicated cluster configuration

How to Achieve Read/Write Scalability

- Parallelism
  - Transactions run in parallel in database cluster
  - A statement can run in parallel in database cluster (future)
- Maintain Transaction Control
  - Transaction Timestamp (Transaction ID)
  - MVCC visibility
- Provide Global Values
  - Sequence
  - Timestamp

Development History

0.9.3
Coordinator Synchronization for DDL
Basic Cursors
Basic Cross-node Joins
Global Timestamps
JDBC/Extended Query Protocol support
2PC from Applications
Configuration Utility

0.9.2
pg_backup & pg_restore
Mutli-node ORDER BY & DISTINCT
Cold Coordinator Synchronization

0.9.1
COPY FROM/TO
Synch Update Replicated
Aggregate Functions

0.9 Basic Version
Single-step Statements
Distributed Transactions

2010 March　May　July　Oct

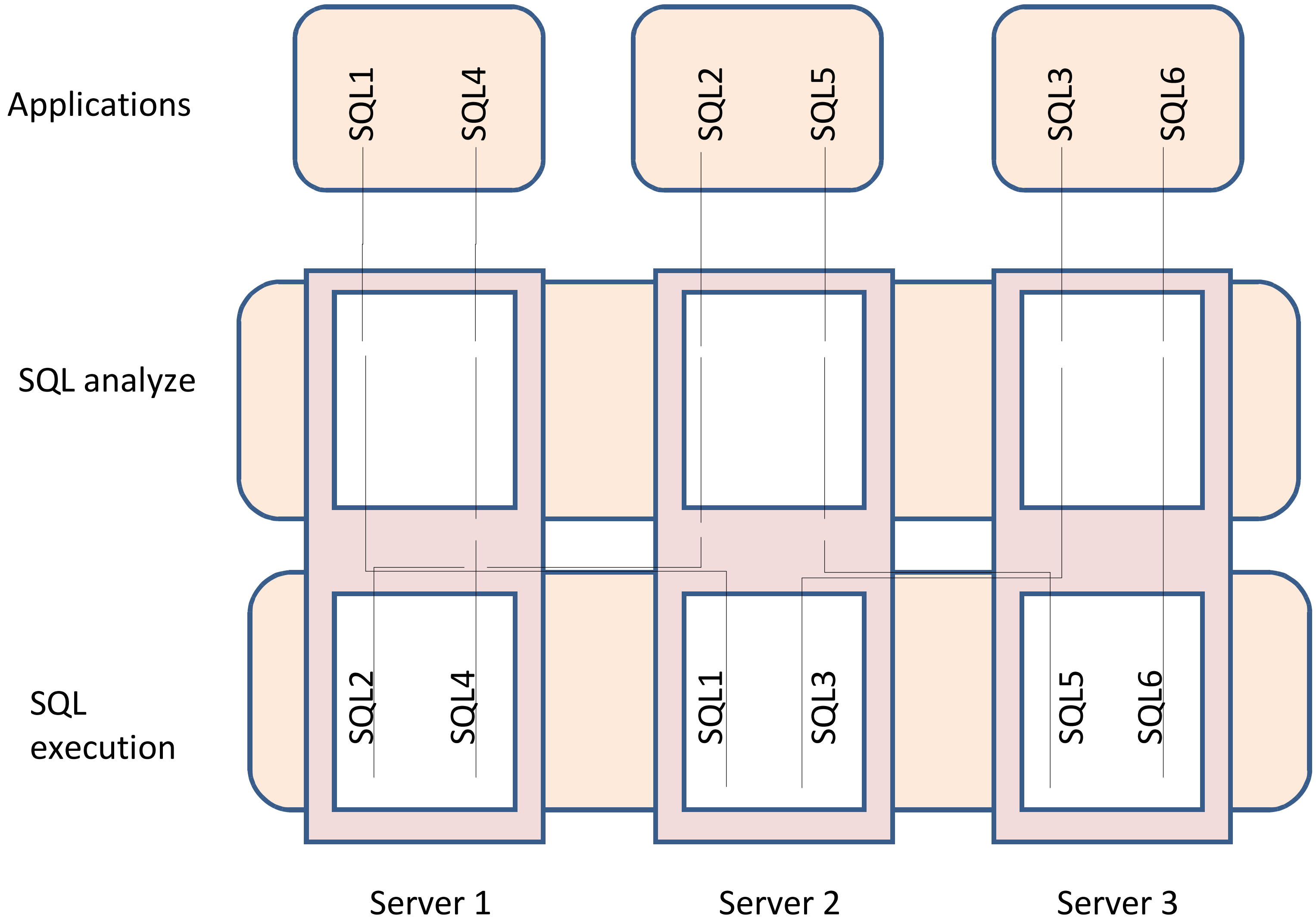
Current Status and Plan

- Version 0.9.3 is available now
  - http://sourceforge.net/projects/postgres-xc
- January 2011
  - UPDATE/DELETE WHERE CURRENT OF
  - Single-step Prepared Statements
  - Join push down for cross node joins + only select needed columns
  - ANALYZE & snapshots
  - INSERT SELECT
  - COPY SELECT
  - CLEAN CONNECTION for the pooler

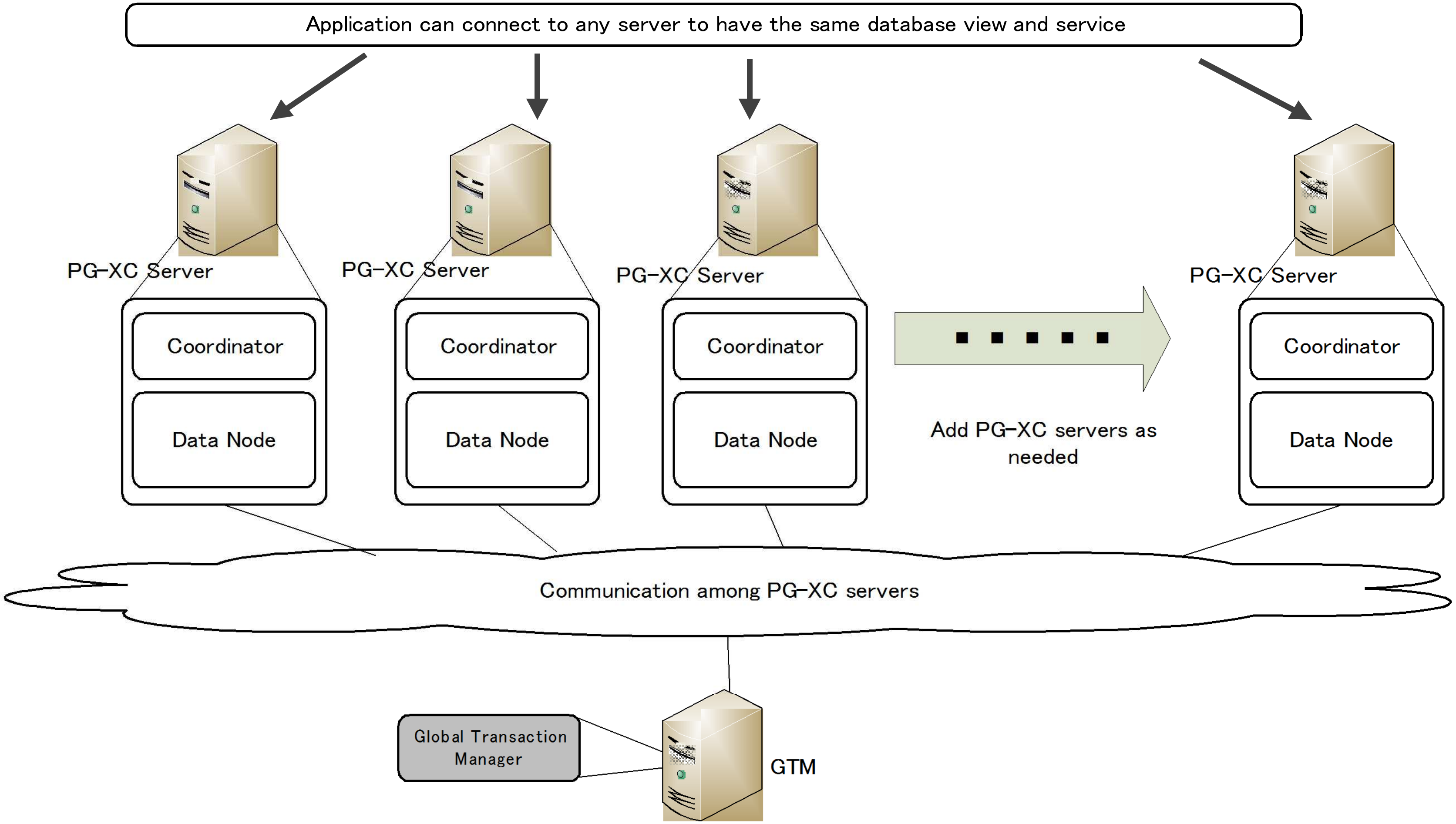
Roadmap and Plan

- Beyond
  - Point in Time Recovery
  - Cross-node optimization
    - Tuple transfer Infrastructure from node to node
  - More variety of SQL statements
  - Multi-step Prepared Statement
  - Expanded cursor support
  - General Stored Functions
  - Savepoint
  - Session Parameters & Pooling
  - High Availability
  - Pooler improvements
  - Trigger
  - Global constraints
  - Tuple relocation
    - Distribution key update
  - Performance improvements
  - Regression Tests (to be continued)

Concurrent Transaction Execution



Postgres-XC Configuration



Postgres-XC Components

- GTM (Global Transaction Manager)
  - Provide global transaction information to each transaction
    - Transaction ID
    - Snapshot
  - Provide other global data to statements
    - Sequence
    - Time/Sysdate
- Coordinator
  - Parse statements and determine location of involved data
  - Transfer statements for each data node (if needed)
  - Application Interface
- Data Node
  - Stores actual data
  - Execute statements from Coordinators

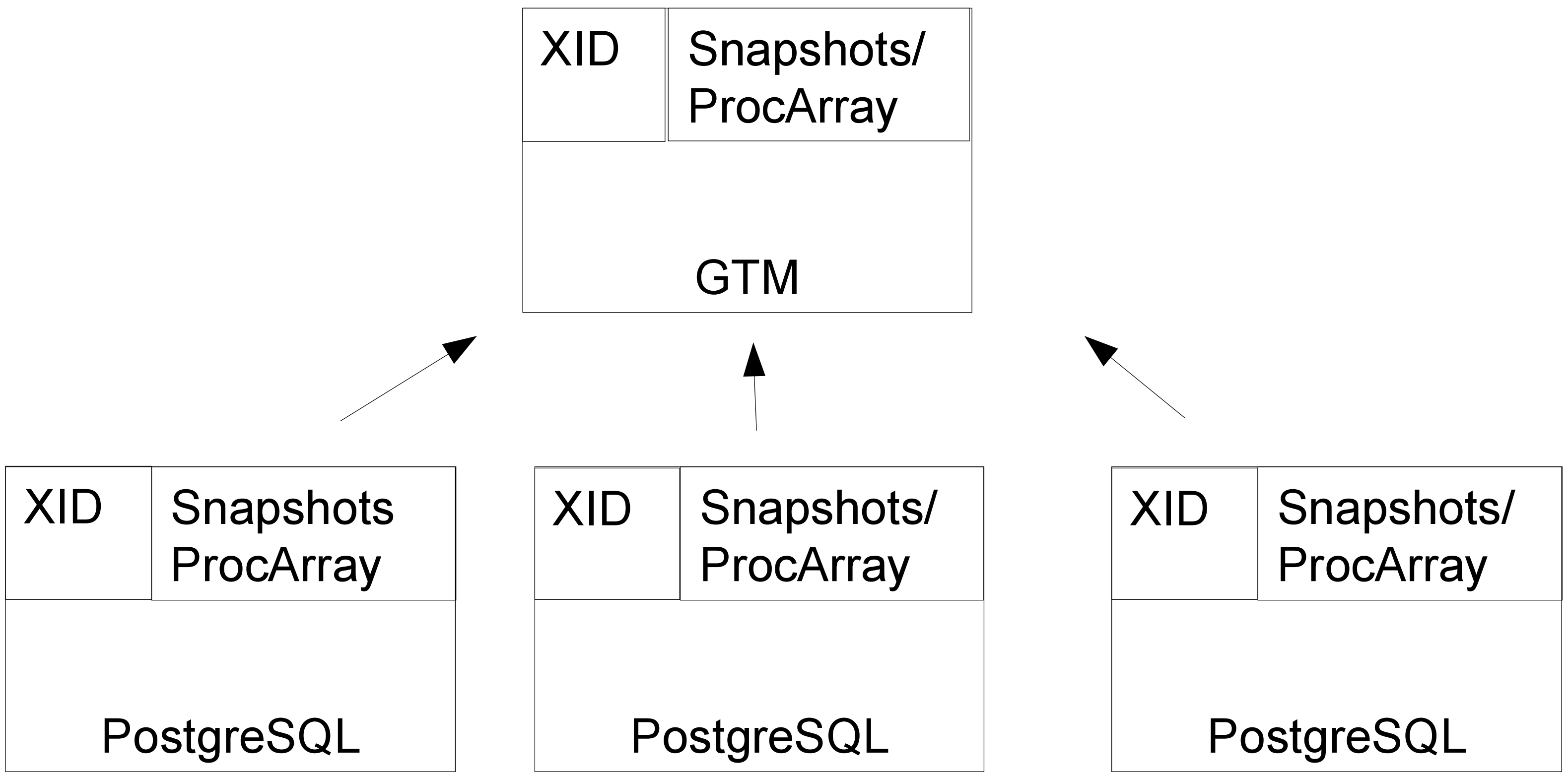
Tables in Postgres-XC

- Tables are replicated or distributed
  - Replicated Table
    - Each Data Node stores whole replicated table
    - Replication is maintained synchronously per statement basis (not WAL basis)
    - Typically static data
  - Distributed Table
    - Each tuple is assigned a Data Node
      - Based on a value of a column (distribution key)
        - » Hash
        - » Round-Robin
        - » Range (future)
        - » User-Defined (future)

How to Determine Distributed/Replicated?

- Transaction tables may be partitioned so that each transaction can be executed in limited number of data nodes.
- Static reference tables may be replicated so that each transaction can read row values locally.

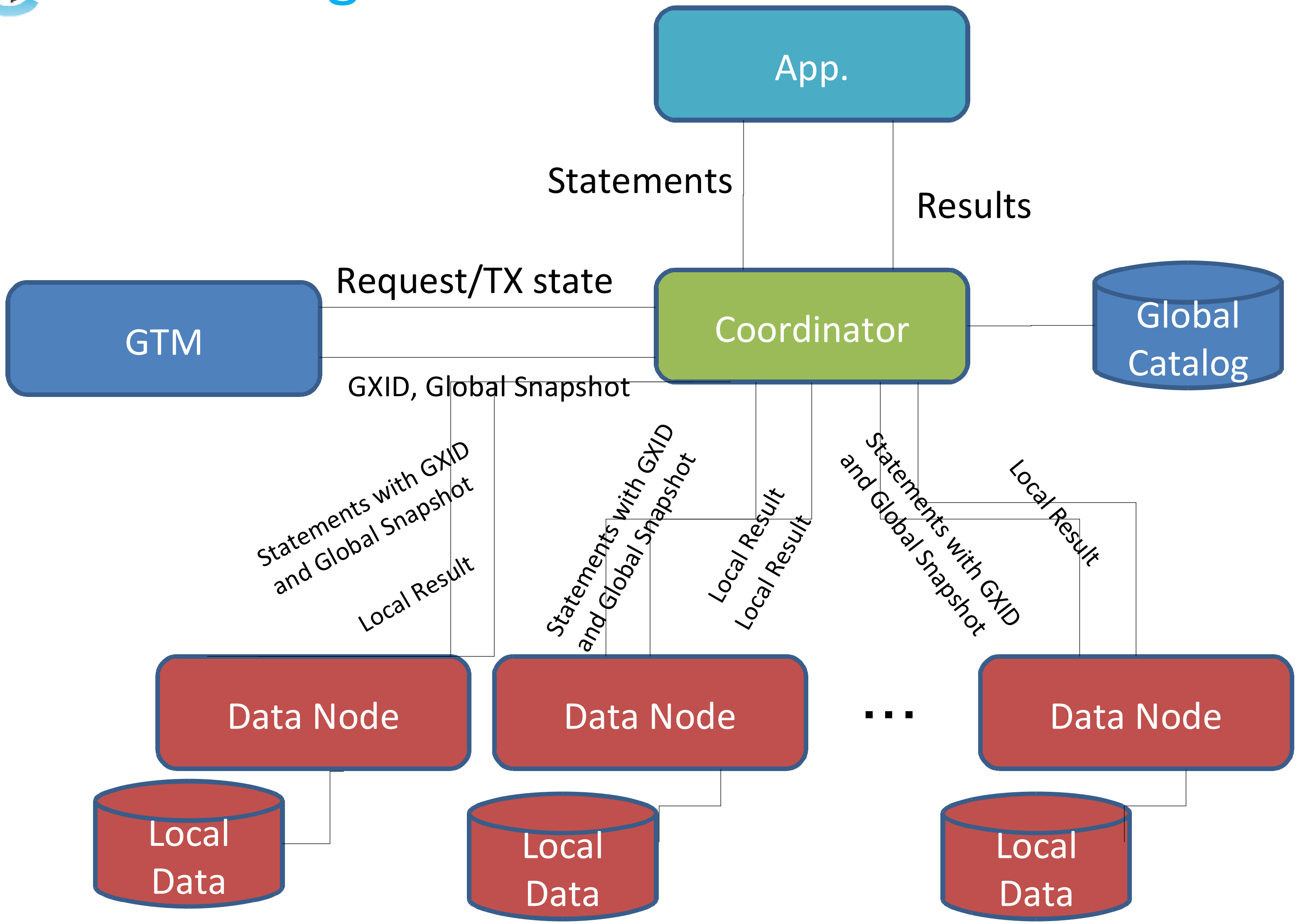
GTM – Global Transaction Manager



GTM – Global Transaction Manager

- GTM is the key of Postgres-XC transaction management
  - Based on extracted transaction management from PostgreSQL
    - Unique Transaction ID (GXID, Global Transaction ID) assignment,
    - Gather transaction status from all the coordinators and maintain snapshot data,
    - Distributed MVCC (Multi-version Concurrency Control) to provide a global snapshot for each statement
  - Extract global value providing feature such as
    - Sequence
    - Time/sysdate (future)

GTM and PG-XC Transaction Management

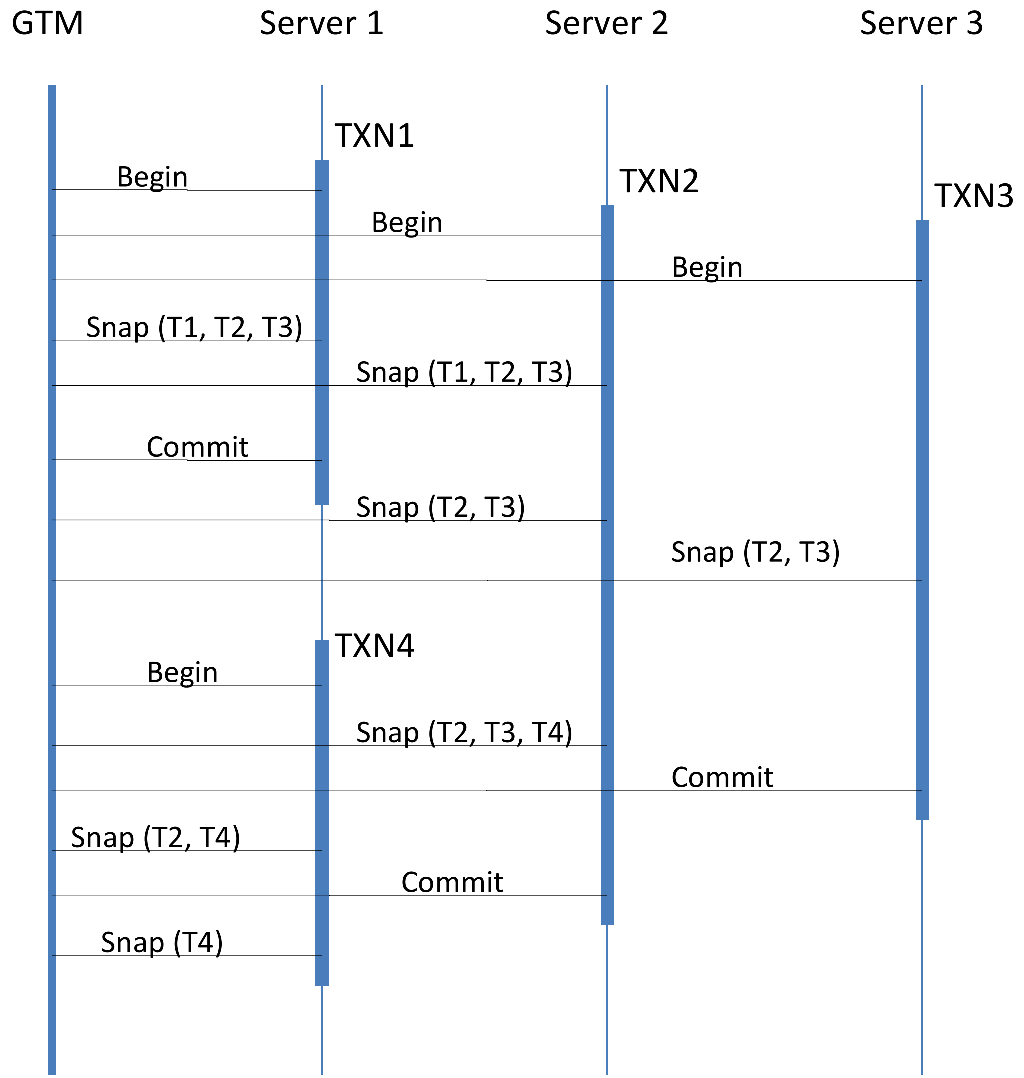


GXID and Snapshot

- GXID
  - Unique Transaction ID in the system
- Global Snapshot
  - Includes snapshot information of transactions in other coordinators.

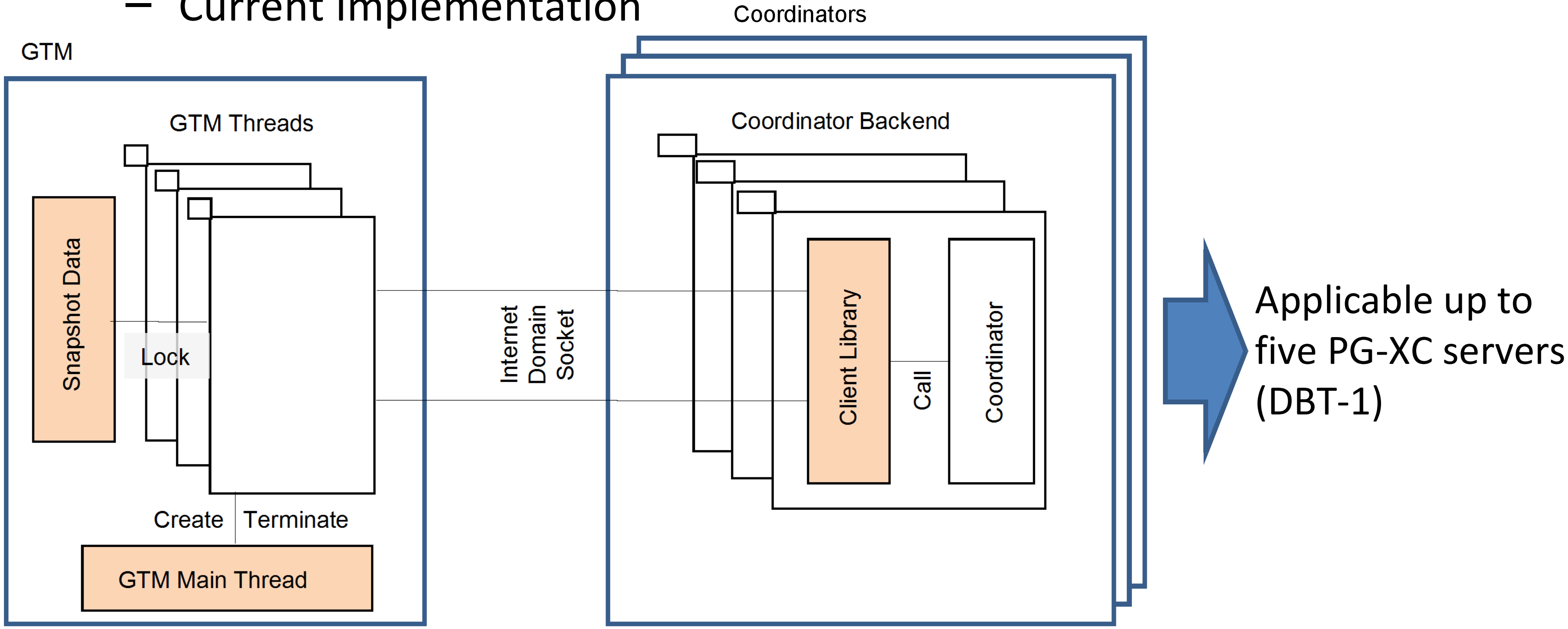
- Data node can handle transactions from different coordinators without consistency problem.
- Visibility is maintained as standalone PostgreSQL.

Outline of PG-XC Transaction Management



Can GTM be a Performance Bottleneck?

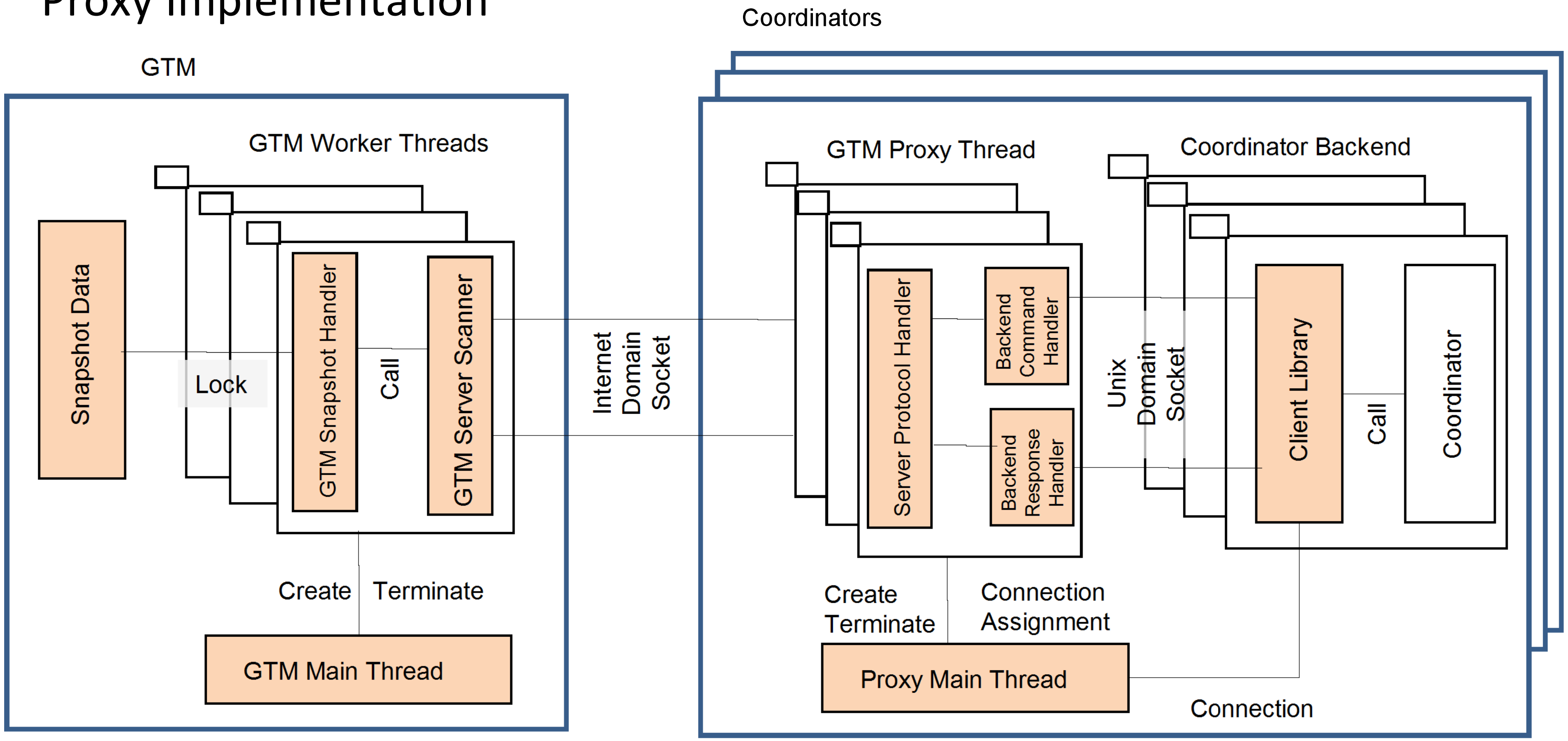
- Depending on implementation
  - Current Implementation



  - Large snapshot size and number
  - Too many interaction between GTM and Coordinators

Can GTM be a Performance Bottleneck?

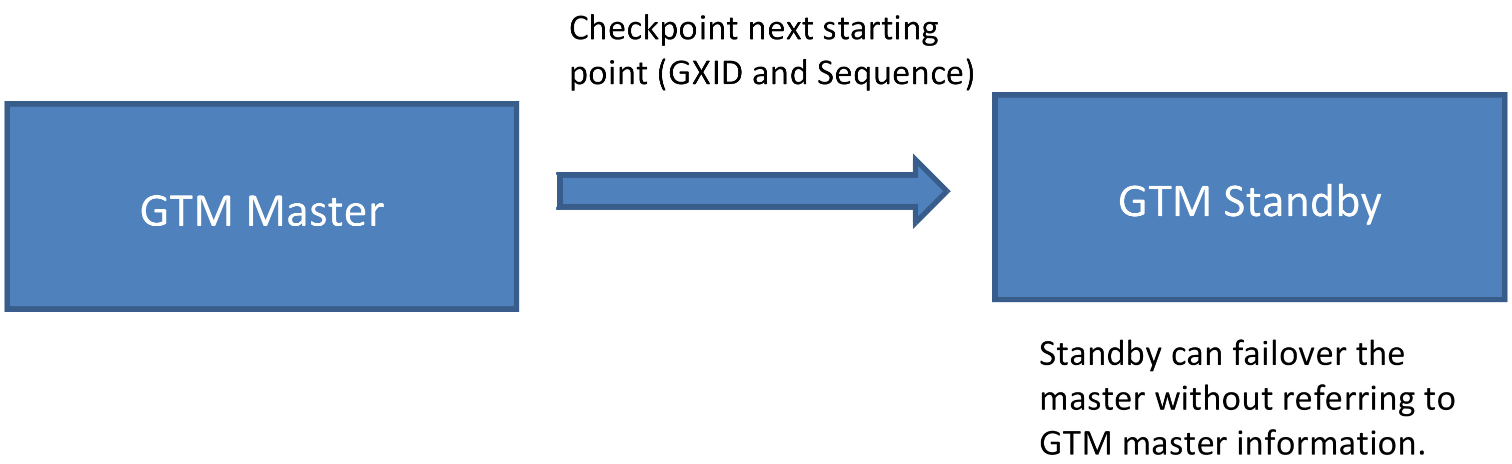
- Proxy Implementation



- Very good potential
  - Request/Response grouping
  - Single representative snapshot applied to multiple transactions
- Maybe applicable for more than ten PG-XC servers

Can GTM be a SPOF?

- Implement GTM standby

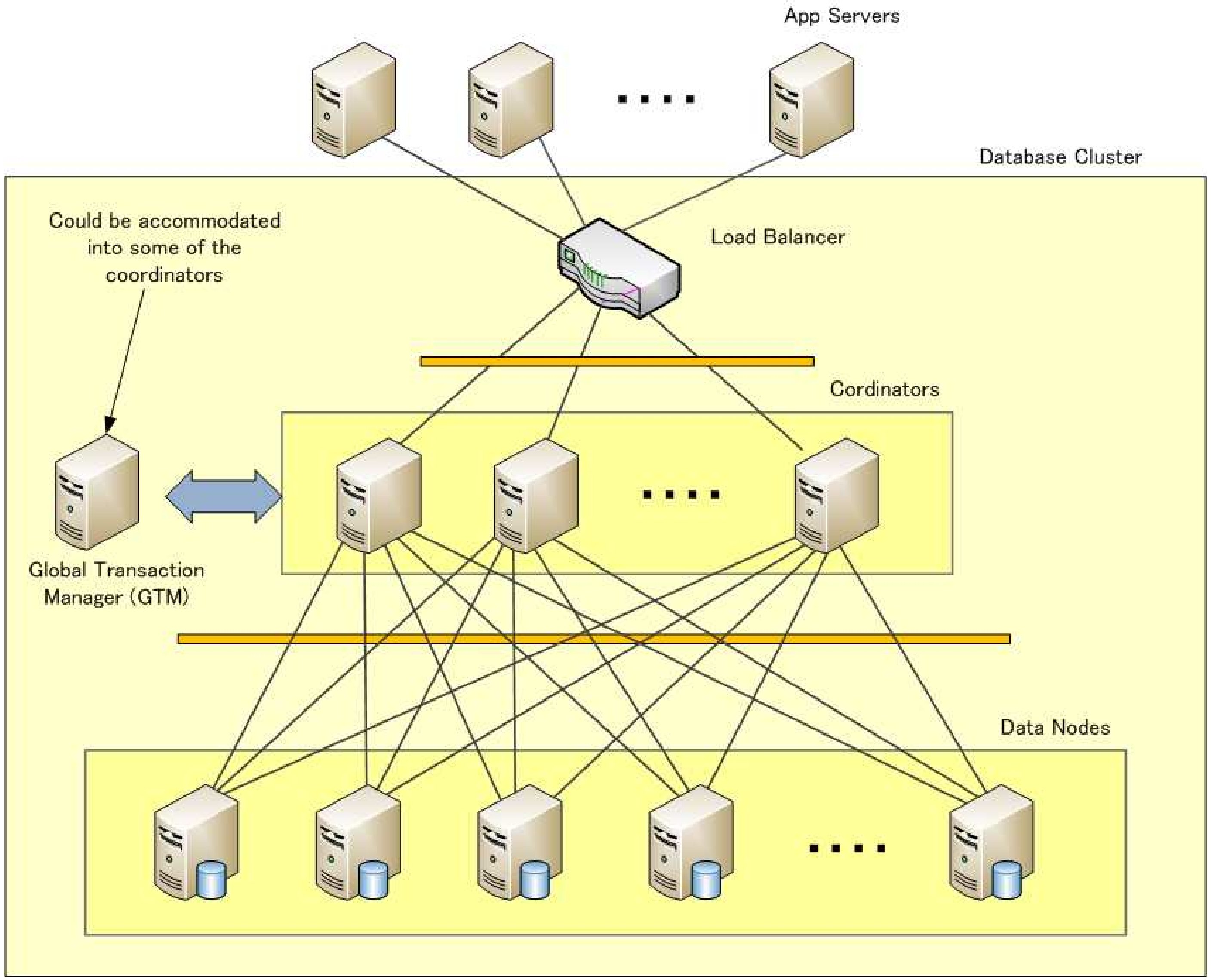


Coordinator & Data Node Internals

Looking at Code

- Not (yet) overly invasive in PostgreSQL code
  - 8.4.2 → 8.4.3 merged cleanly
- Existing modules use `#ifdef PGXC` to identify Postgres-XC changes
- IS_PGXC_COORDINATOR and IS_PGXC_DATANODE easily identifies applicable code
- Advanced Coordinator logic & GTM in separate modules

Reference Architecture



Coordinator Overview

- Based on PostgreSQL 8.4.3 (9.0 soon)
- Accepts connections from clients
- Parses requests
- Examines requests, reroutes to Data Nodes
- Interacts with Global Transaction Manager
- Uses pooler for Data Node connections
- Sends down XIDs and snapshots to Data Nodes
- Uses two phase commit if necessary

Data Node Overview

- Based on PostgreSQL 8.4.3 (9.0 soon)
- Where user created data is actually stored
- Coordinators (not clients) connects to Data Nodes
- Accepts XID and snapshots from Coordinator
- Special autovacuum/analyze handling
- The rest is fairly similar to vanilla PostgreSQL

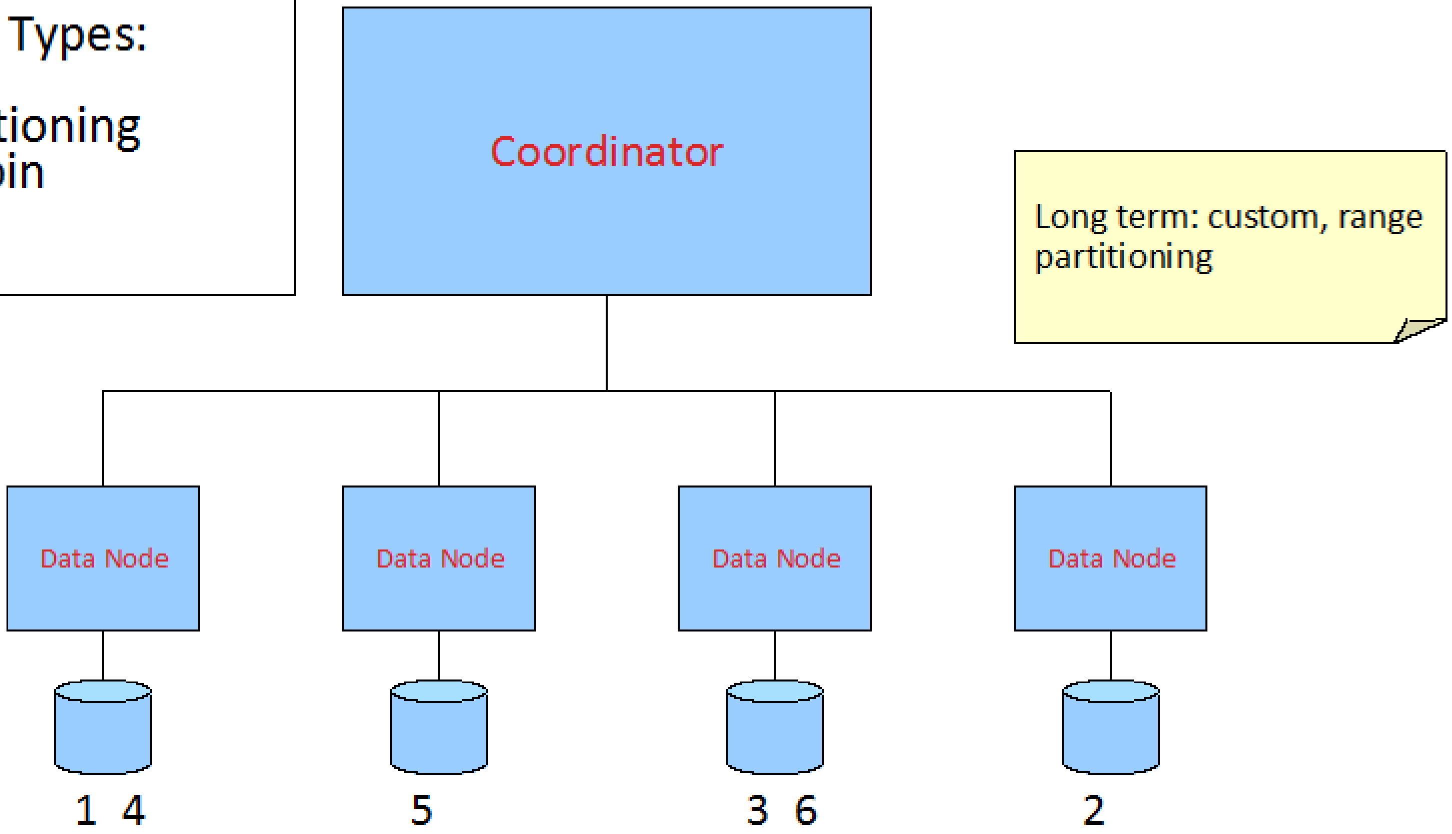
Postgres-XC Request Handling

- Data Distribution
- Pooler
- Statements
  - Only involve nodes as needed
  - Proxy efficiently
  - If multiple nodes, issue query simultaneously
  - Global MVCC
- Transactions

Data Distribution

Distribution Types:

- Hash Partitioning
- Round Robin
- Replicated



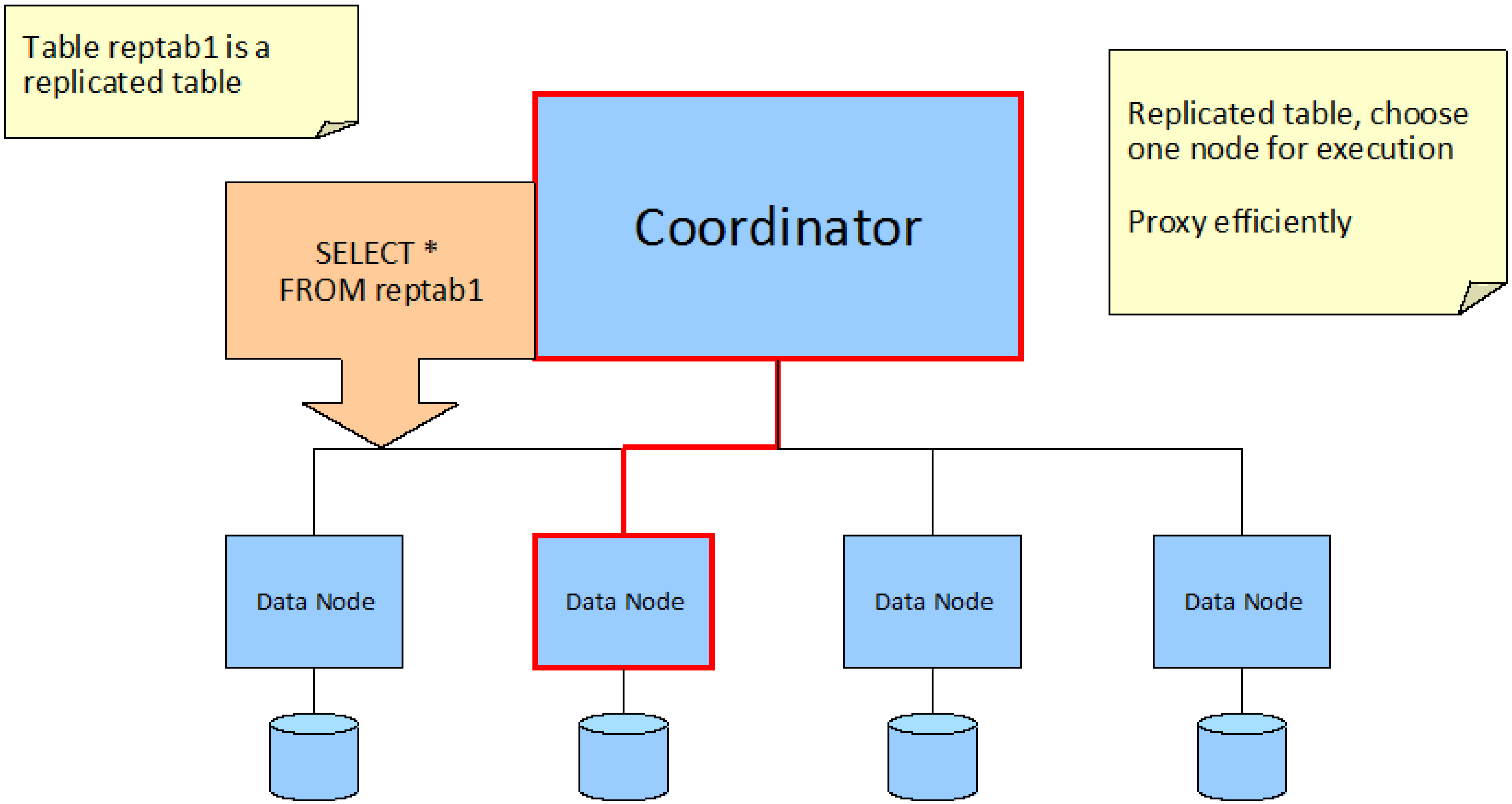
Connection Pooling

- The Coordinator forks off a pooler process for managing connections to the Data Nodes
- Coordinator obtains connections from pooler process as needed
  - Not every transaction needs all Data Nodes
- At commit time, Coordinator returns connections to the pool
- As we add clients and multiple Coordinators, we want to prevent an explosion of required connections at the data node level by pooling instead

Statement Handling

- Large coverage of SQL statements handled
  - (cross-node joins inefficient)
- Use distribution information in Coordinator
- If more than one Data Node, send down statement to all simultaneously
- Recognize singleton statements
- Recognize single-step statements
- Handle replicated tables
- Use two phase commit
  - (and use only when necessary)

Statement Handling - Execution



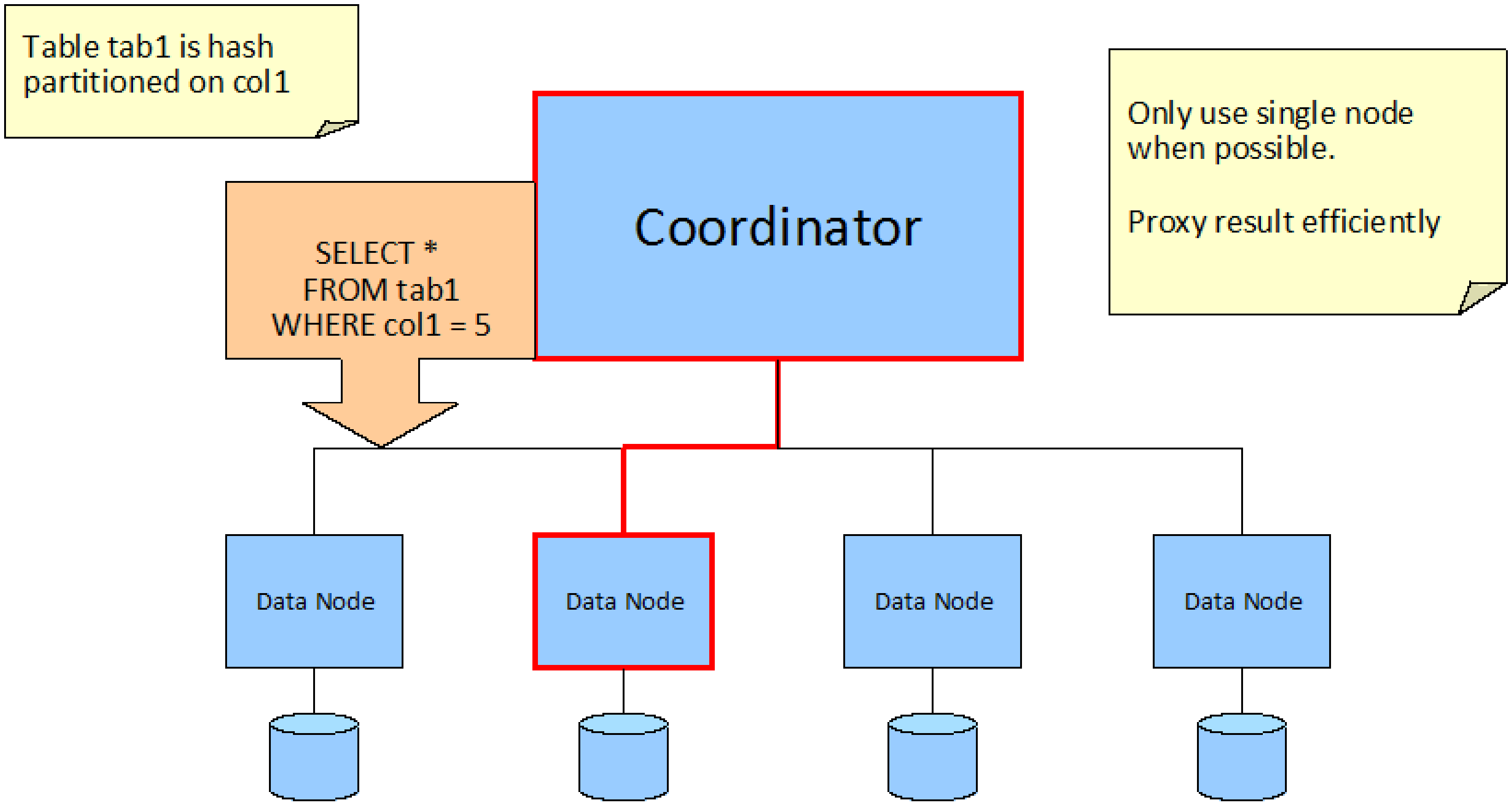
Queries with Replicated Tables

- Choose a node via round robin to execute on
- Recognize queries with joins between replicated tables

```
SELECT *
  FROM reptab1 r1 INNER JOIN reptab2 r2
    ON r1.col1 = r2.col2
```

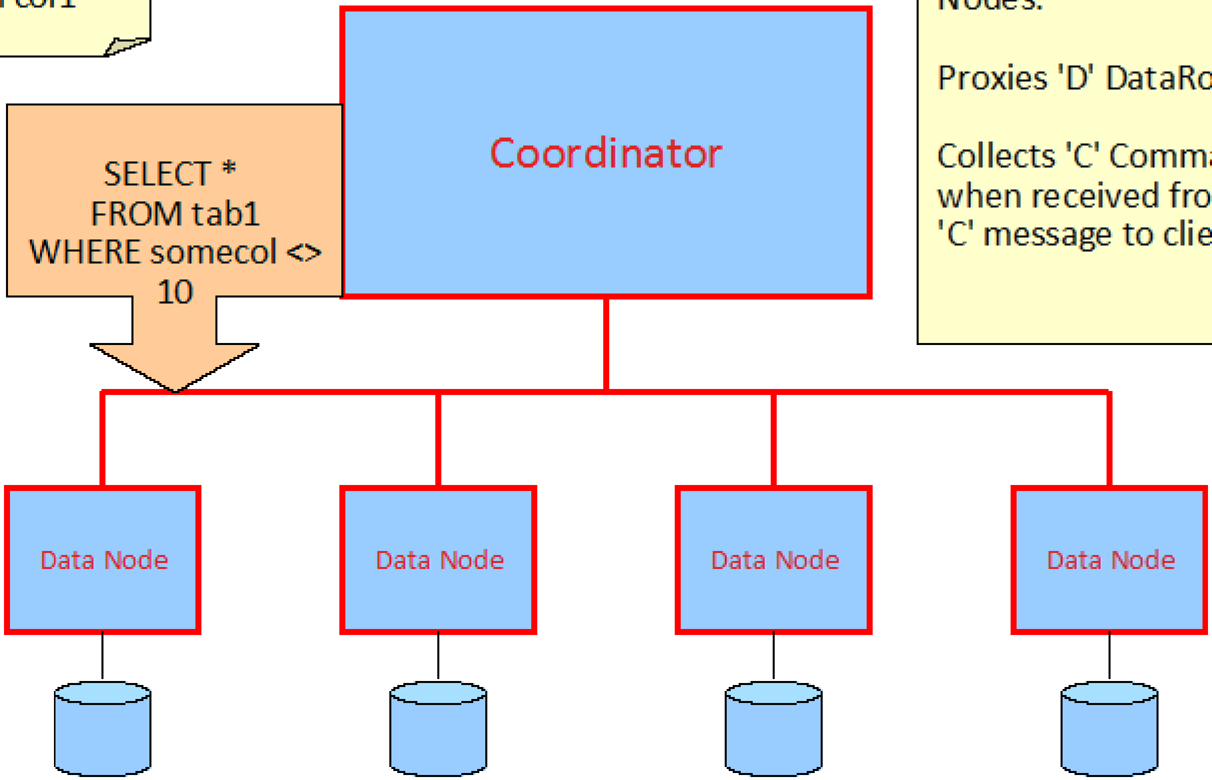
- For write operations
  - All nodes
  - Two phase commit
  - Write on single “primary” data node first to avoid deadlocks

Statement Handling - Execution



Statement Handling - Execution

Table tab1 is hash partitioned on col1



No condition allows for a single node, send query to all Data Nodes.

Proxies 'D' DataRow messages.

Collects 'C' CommandComplete, when received from all, sends one 'C' message to client.

Queries with Partitioned Tables

- Check WHERE clause to see if we can execute on one node
- Recognize queries with joins with replicated tables

```
SELECT *
  FROM tab1 t INNER JOIN reptab1 r
    ON t.col2 = r.col3
 WHERE t.col1 = 1234
```

- Recognize queries with joins on respective partitioned columns

```
SELECT *
  FROM tab1 t1 INNER JOIN tab2 t2
    ON t1.col1 = t2.col1
 WHERE t.col1 = 1234
```

Visibility and Data Node Handling

- When the first statement of a transaction needs to execute, a global XID is obtained from GTM
- Each time a new Data Node connection joins a transaction, the Coordinator sends down a GXID to the Data Node
- Each statement execution requires a new snapshot being obtained from GTM
- Before sending down a SQL statement, the Coordinator first passes down a snapshot to the Data Nodes

Transactions and Data Node Handling

- The Coordinator tracks read and write activity*
- At commit time
  - If we have only written to one Data Node, we simply issue commit to the node
  - If we have written to more than one Data Node, we use two phase commit

*Stored functions could theoretically write to DB

Transaction Handling Considerations

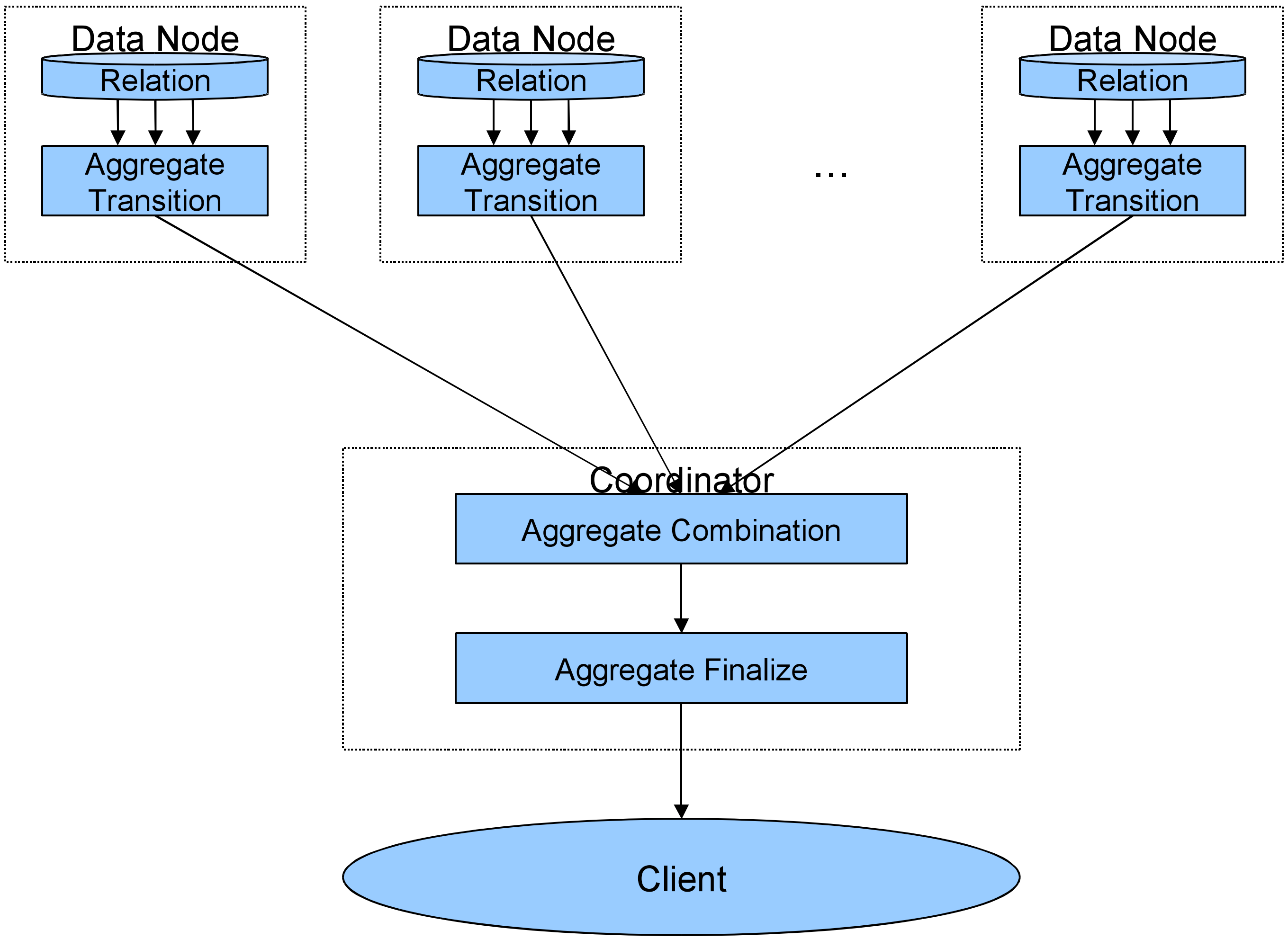
- Distributed transactions and two phase commit (2PC)
- Distributed Multi-Version Concurrency Control
  - Global Snapshots
  - Autovacuum
    - exclude XID in global snapshots
  - ANALZYE
  - Future optimization
  - CLOG
    - Careful when extending, not all transactions are on all nodes

Aggregate Handling

- Traditional PostgreSQL in Two Phases:
  - Transition Function
  - Finalizer Function
- Postgres-XC uses Three Phases:
  - Transition Function
  - Collector Function
  - Finalizer Function

Aggregate Handling

Postgres-XC Aggregate Flow



Aggregate Handling - AVG

- AVG (Average) needs to sum all elements and divide by the count
- Transition

```
arg1[0]+=arg2;
arg1[1]++;
return arg1;
```

- Combiner (only in Postgres-XC)

```
arg1[0]+=arg2[0];
arg1[1]+=arg2[1];
return arg1;
```

Get the sum of the sums and the sum of the counts from the Data Nodes

- Finalizer

```
return arg1[0]/arg1[1];
```

UPDATE / DELETE WHERE CURRENT OF cursor

- Partitioned Tables
  - Fetch one row at a time, track source data node
  - Pass UPDATE/DELETE WHERE CURRENT OF down to the appropriate node
- Replicated Tables
  - SELECT FOR UPDATE required
  - Fetch from primary data node, along with CTID
  - When WHERE CURRENT OF, fetch uniquely identifying info for tuple, issue UPDATE/DELETE

INSERT SELECT

- Execute SELECT
- Send rows down to Data Nodes via COPY (FROM STDIN)
  - Take into account if destination table is partitioned or replicated

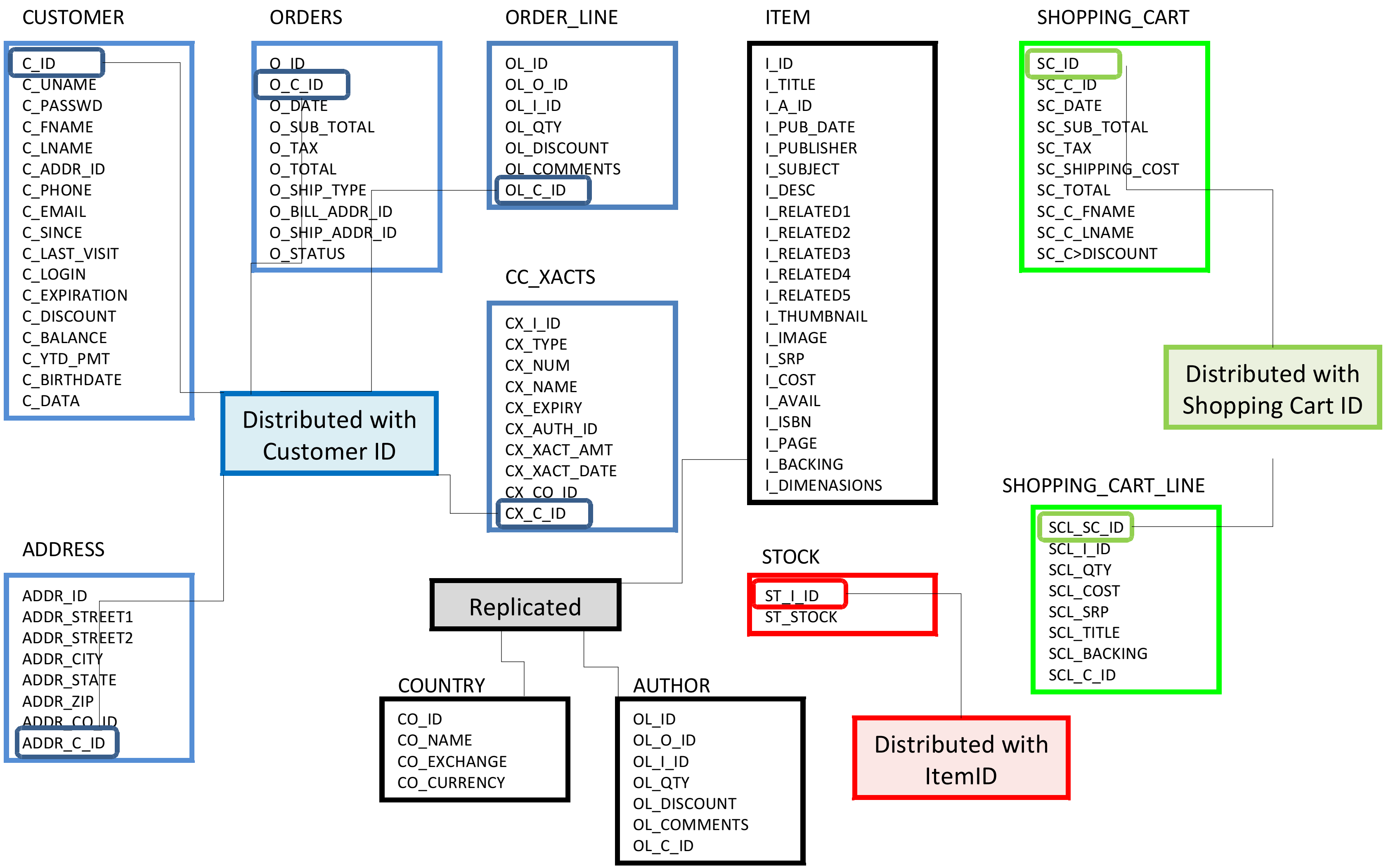
- Can be improved
  - Data Node to Data Node communication
  - Avoid extra conversions

Evaluation

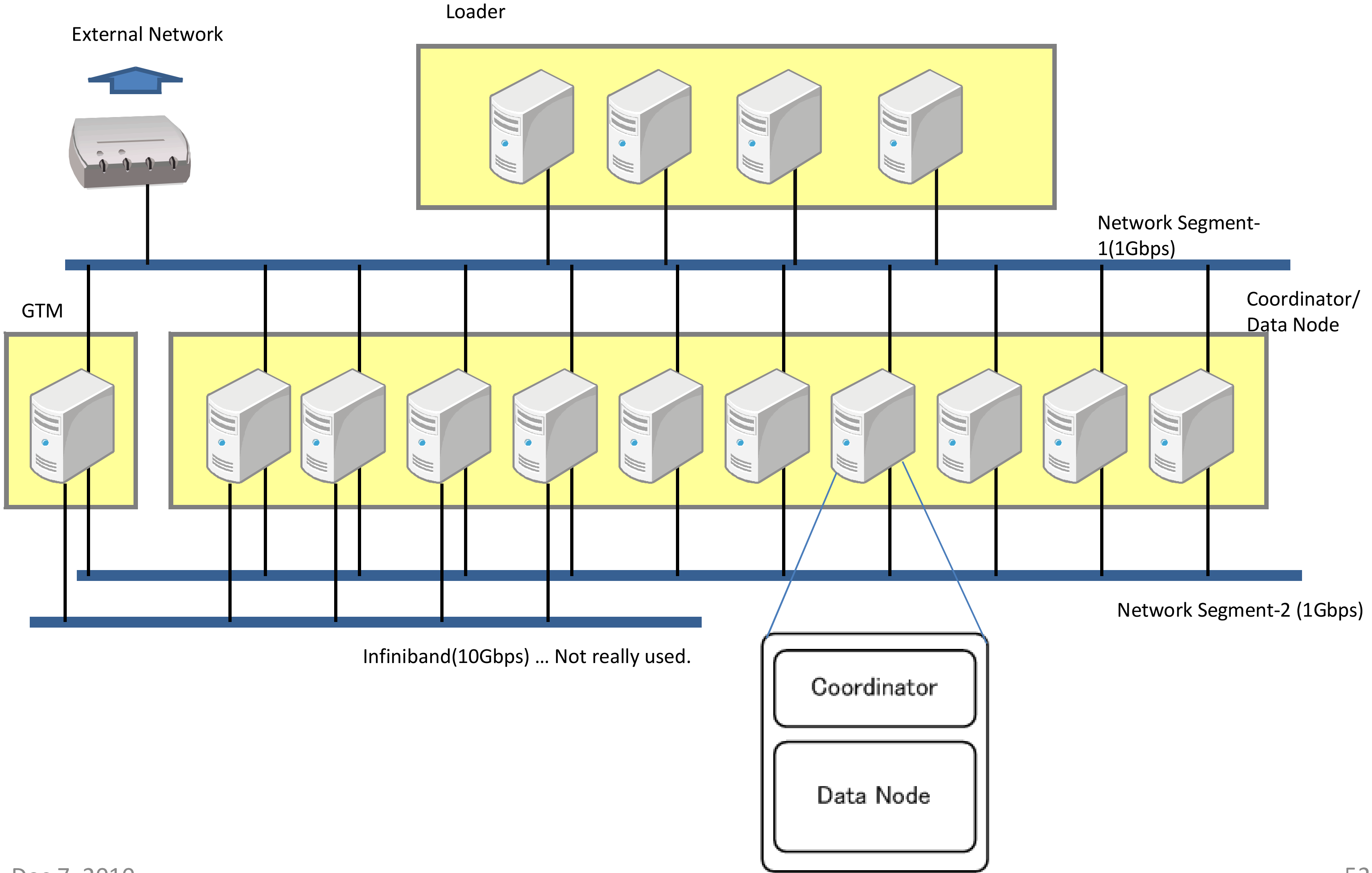
Postgres-XC Performance Benchmark

- Based on DBT-1
  - Typical Web-based benchmark
  - We had good experience on this
- Changes from the original
  - Changed ODBC to libpq
    - Put much more workload
  - Added distribution keys
    - Can be automatically generated in the future
  - One table divided into two
    - According to the latest TPC-W specification
    - Matches Postgres-XC characteristics

DBT-1-based Table Structure



Evaluation Environment



Server Spec

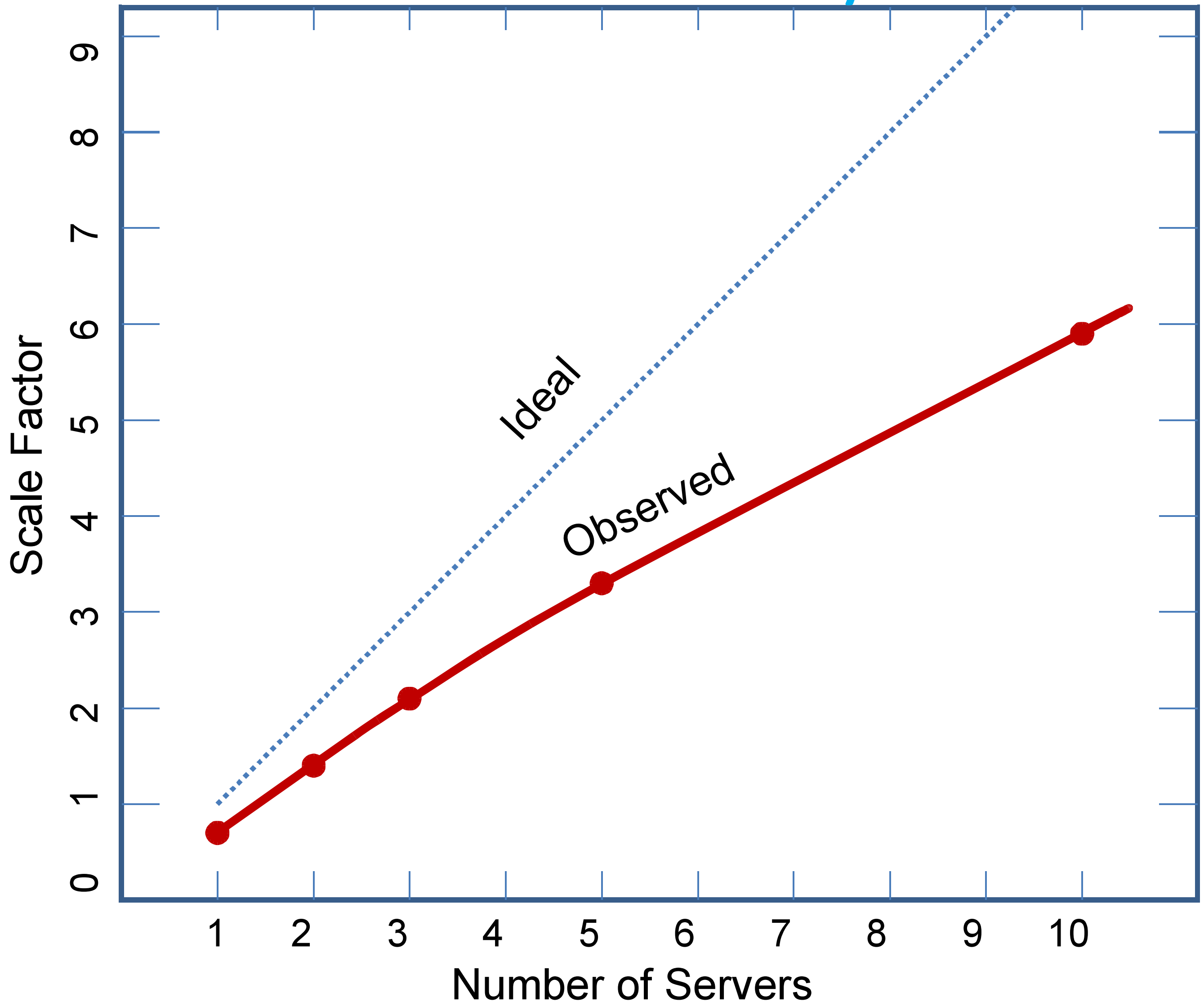
	Coordinator/Data Node	GTM/Loader
Make	HP Proliant DL360 G6	HP Proliant DL360 G5
CPU	Intel® Xeon® E5504 2.00GHz x 4	Intel® Xeon® X5460 3.16GHz x 4
Cache	4MB	6MB
MEM	12GB	6GB
HDD	146GB SAS 15krpm x 4 ea	146GP SAS 15krpm x 2 ea

Evaluation Summary

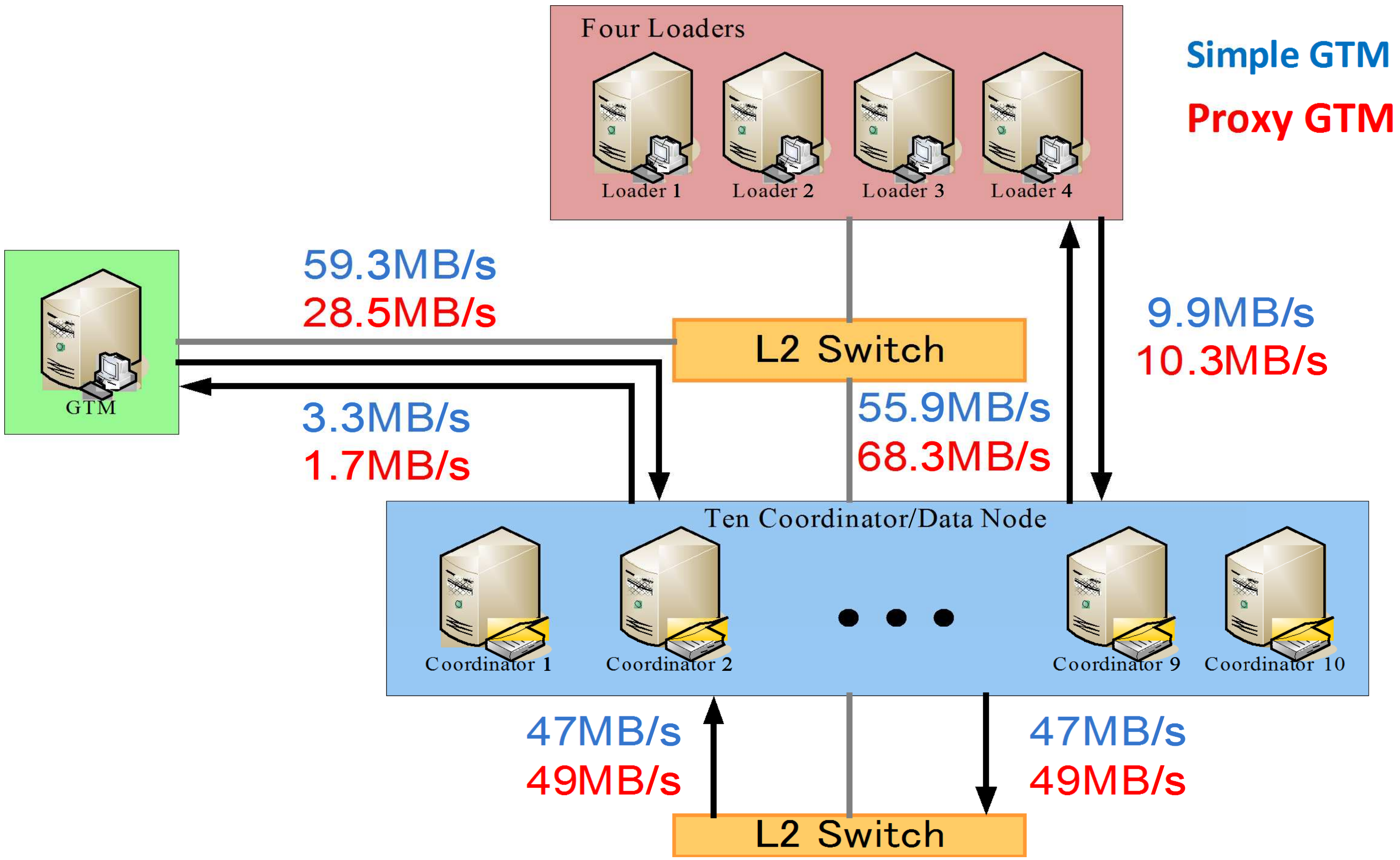
Full Load Throughput

Database	Num. of Servers	Throughput (TPS)	Scale Factor
PostgreSQL	1	2,617	1.0
Postgres-XC	1	1,869	0.71
Postgres-XC	2	3,646	1.39
Postgres-XC	3	5,379	2.06
Postgres-XC	5	8,473	3.24
Postgres-XC	10	15,380	5.88

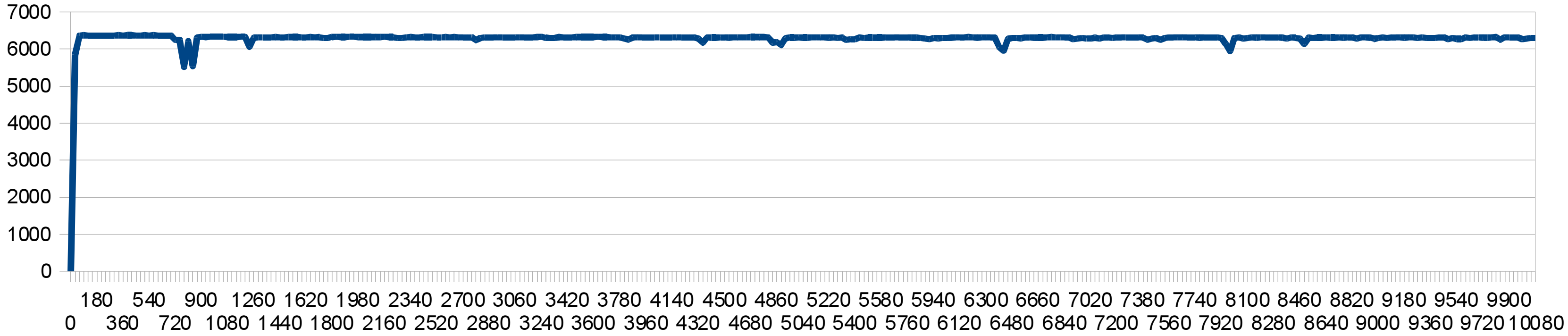
Scale Factor Summary



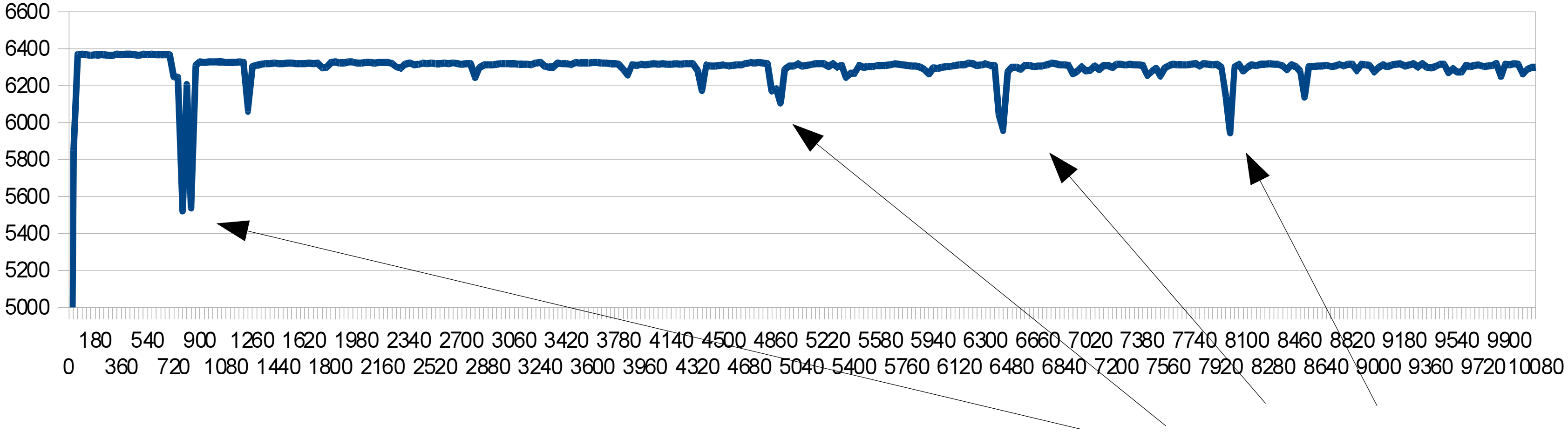
Network Workload



One Week Test



Zoom:



Reasonably stable in a long run (90% workload)

Vaccum Analyze may become long transactions to affect the throughput.

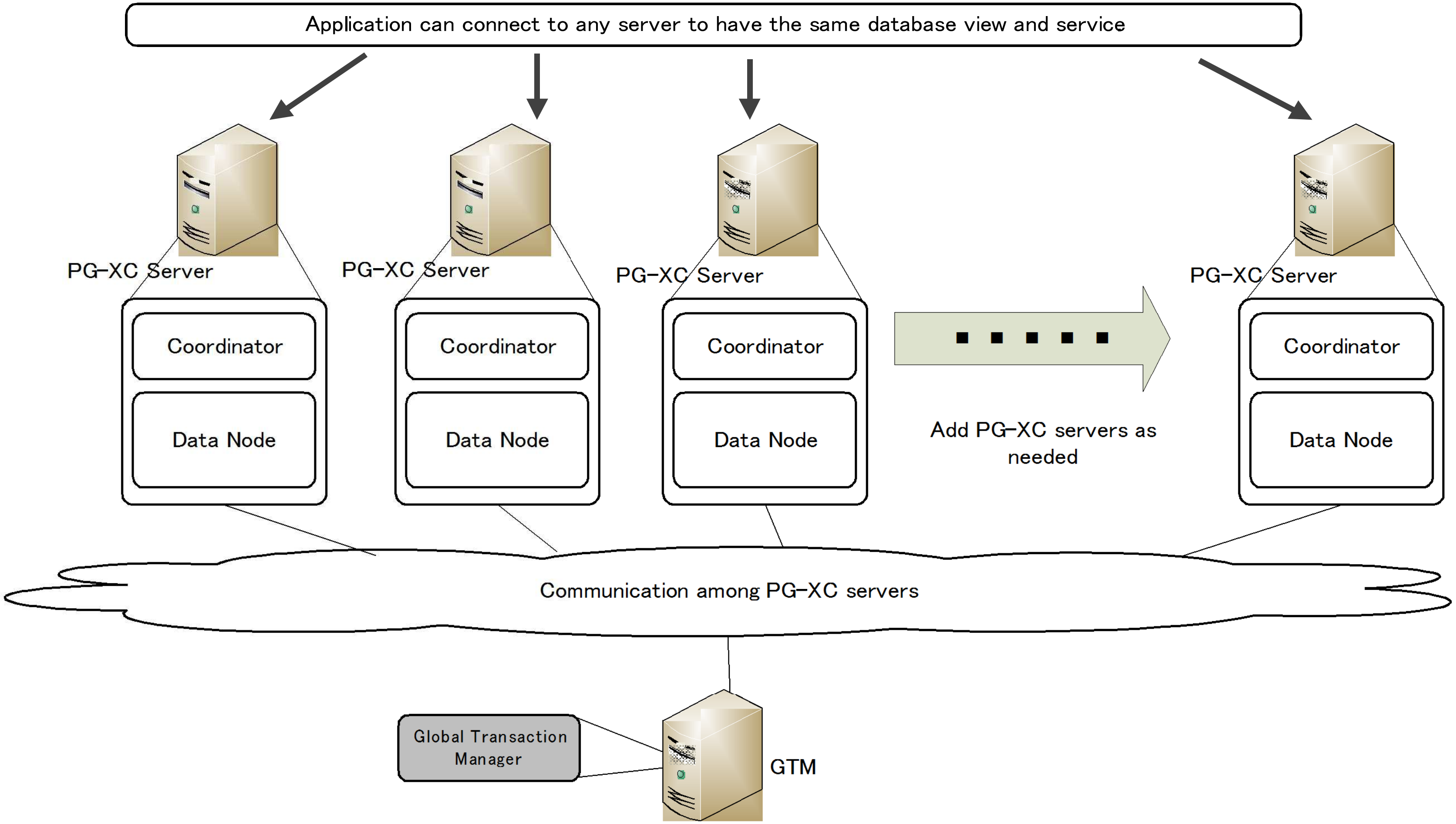
Avoiding Long Transactions

- Vacuum
  - Needs GXID
  - Vacuum's GXID need not to appear in local or global snapshot
- Vacuum Analyze
  - Needs GXID
  - GXID should appear in local snapshot
  - GXID need not appear in global snapshot (January 2011)

Evaluation Summary

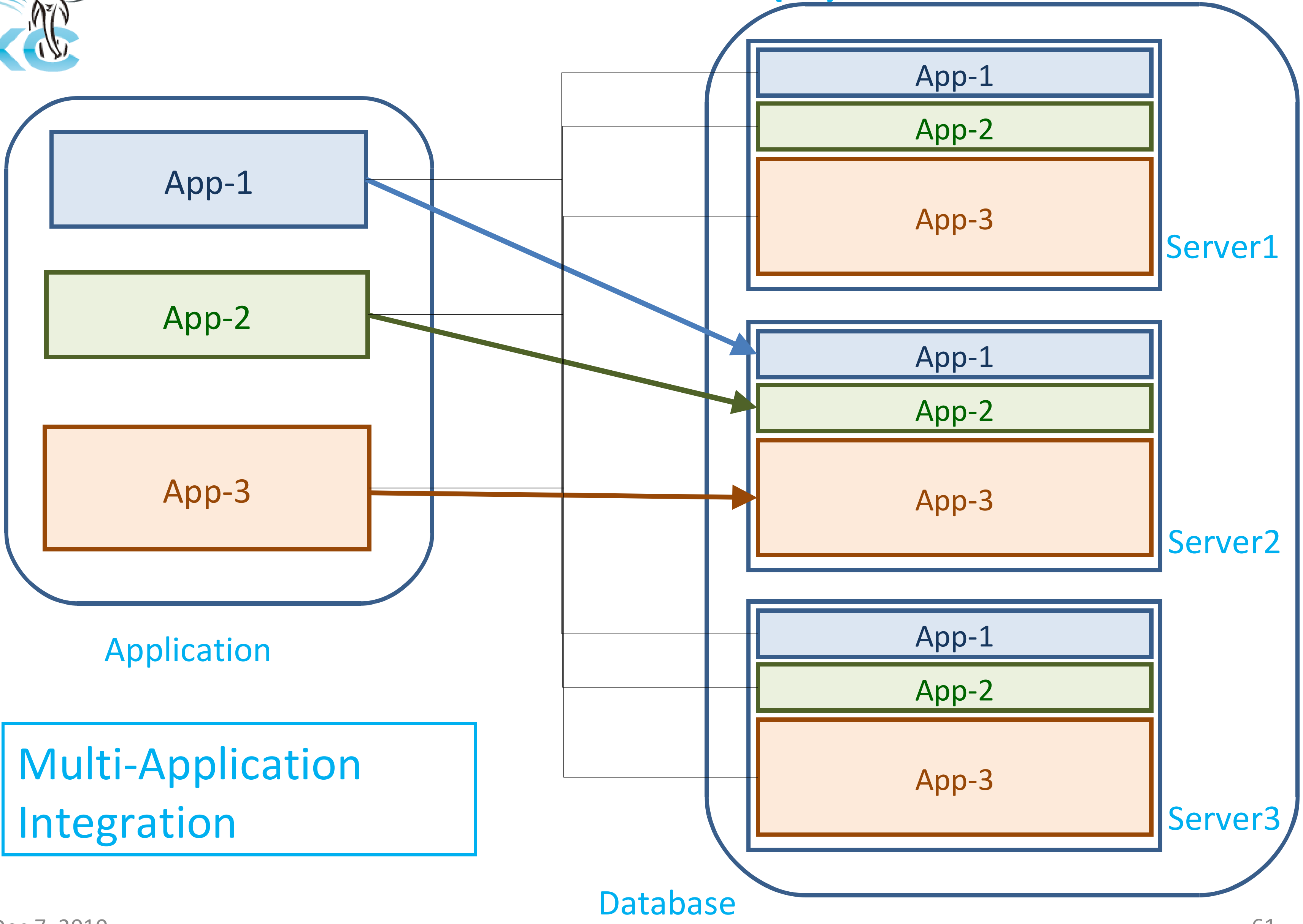
- PG-XC is reasonably scalable in both read/write.
- Need some tweaking to stabilize the performance.
- Network workload is reasonable.
  - GTM Proxy works well
  - More work is needed to accommodate more servers (thirty or more)
- Fundamentals are established
  - Will continue to extend statement support

Possible Use Case (1)



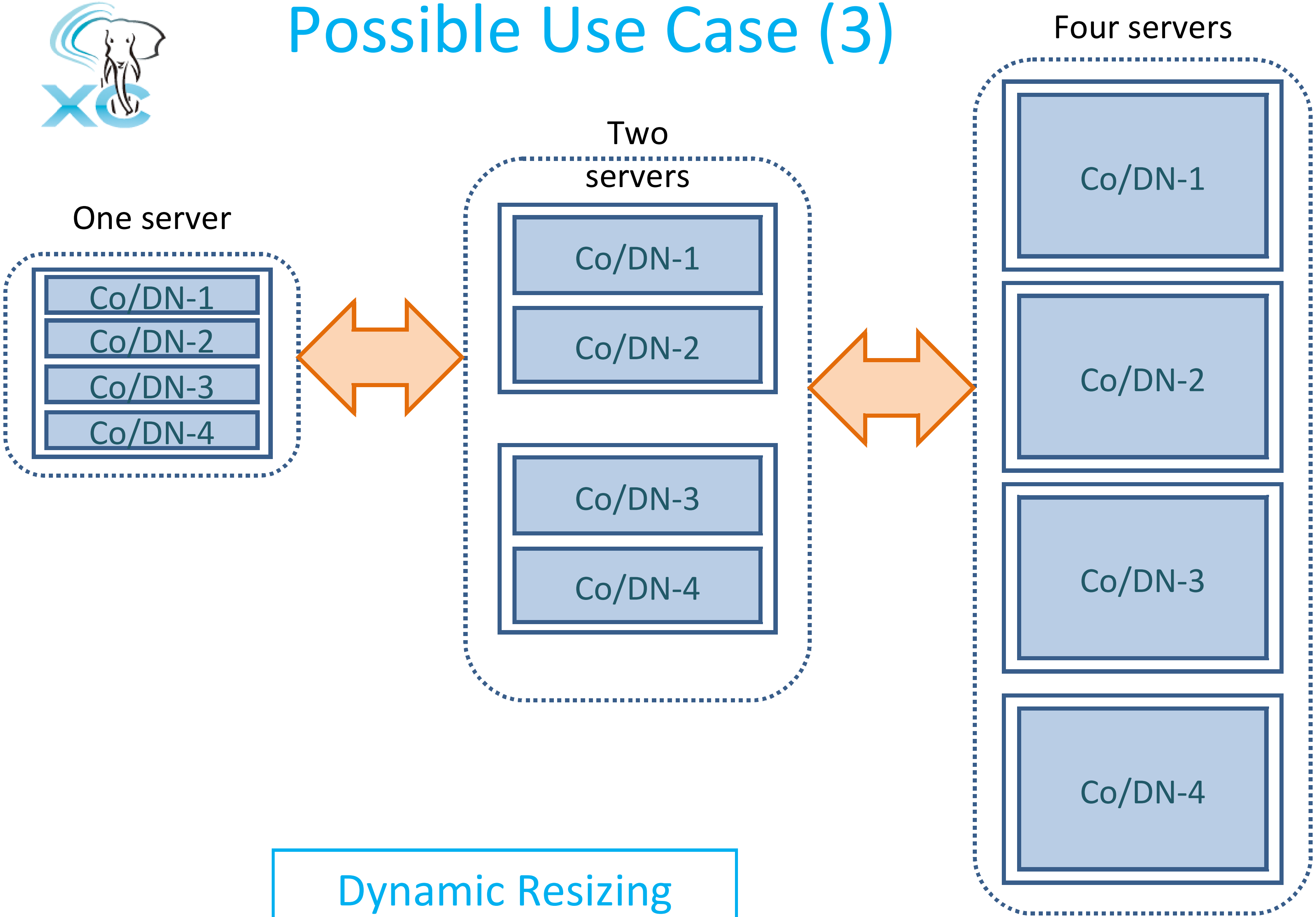
Large Scale Application

Possible Use Case (2)



Multi-Application
Integration

Possible Use Case (3)



Dynamic Resizing
(Cloud)

Developers Welcome

- We welcome people to help the project
  - Each issue in WIP and the roadmap is composed of small manageable pieces.
  - If you are interested in the project, please contact us.
- Project Home Page

  http://postgres-xc.sourceforge.net/
- Contact

  koichi.szk@gmail.com

  mason.sharp@gmail.com

POSTGRES XC

Thank You Very Much