NTT

EnterpriseDB

Read/Write scalability and transaction management in Postgres-XC

CHAR(10), 3rd July, 2010

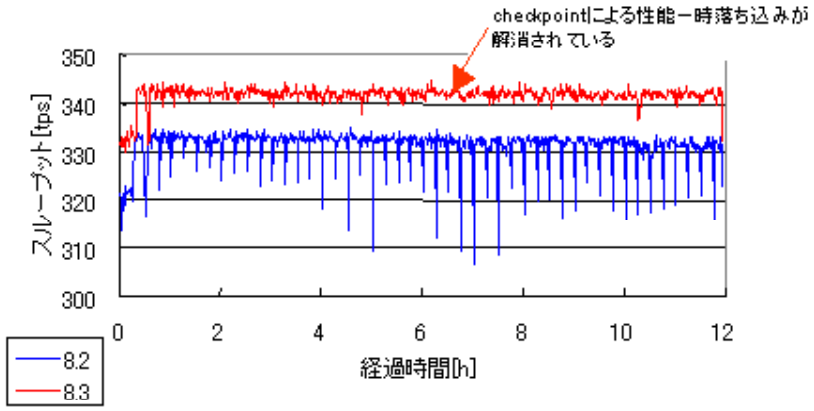
Koichi Suzuki

About NTT Open Source Activities

- What is NTT?
  - Japanese largest carrier
    - Including telephone, mobile, IT, etc. (NTT-Group)
- NTT's Open Source Activities
  - NTT Open Source Software Center
    - > 100 engienners
    - Group-wide OSS promotion/support
    - Database (almost PostgreSQL), Operating System, High-Availability Middleware (Heartbeat/Pacemaker)

NTT's contributions to PostgreSQL

- Performance Stability in 8.3
  - Smooth Checkpoint
- Streaming Replication in 9.0
- Others
  - Operation statistics collection
    - Pg_statsinfo
  - High speed data loader
    - Pg_bulkload
  - Archive log size compression
    - Pg_lesslog
- WIP
  - SQL/MED
  - Partitioning



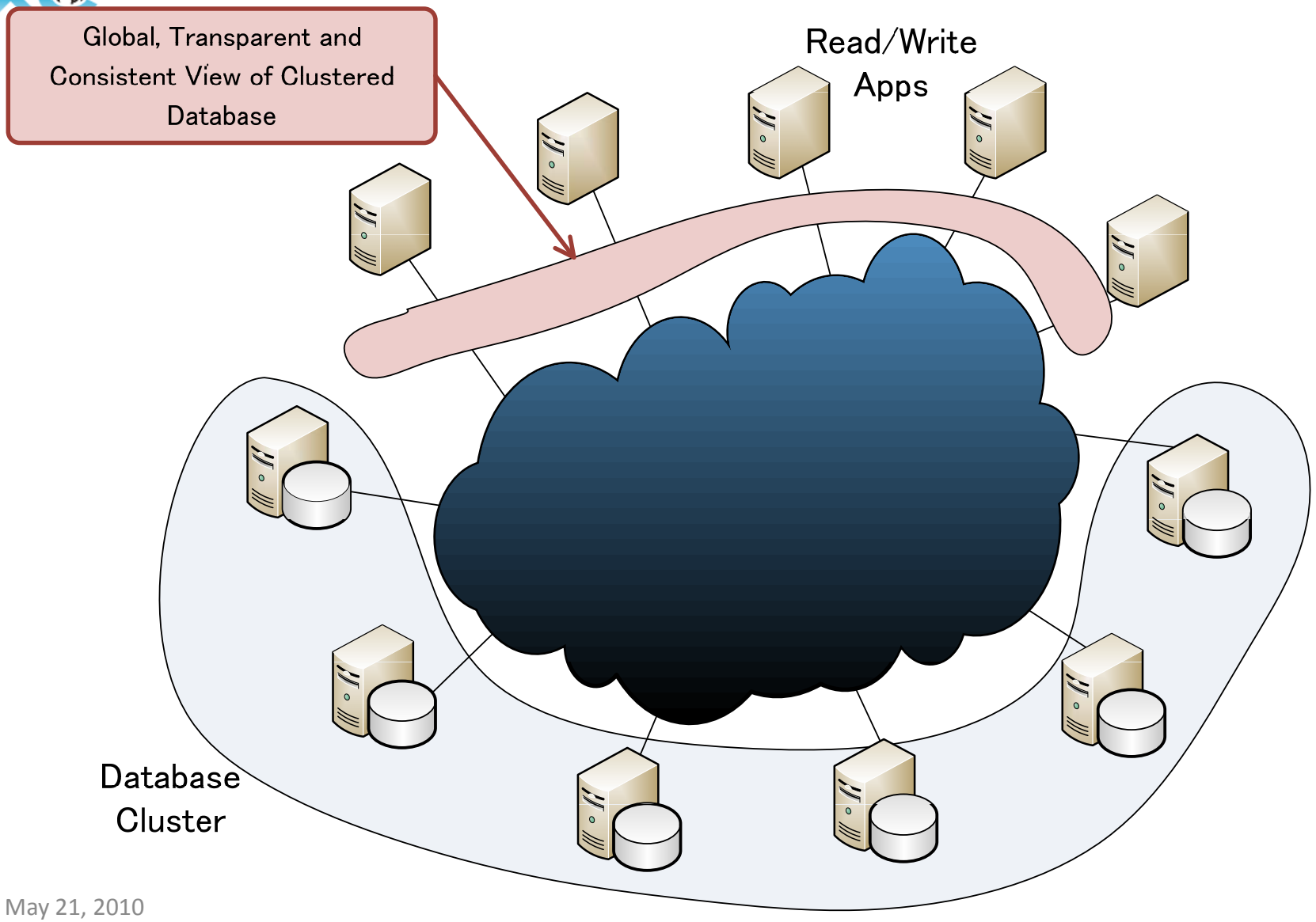
Today's Talk

- 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?

- Write-scalable PostgreSQL cluster
  - More than 3 1/4 performance scalability with five servers, compared with pure PostgreSQL (DBT-1)
- Synchronous multi-master configuration
  - Any update to any master is visible from other masters immediately.
  - Not just a “replication”
    - Distribution (partition) and replication combination of tables
- NTT and EnterpriseDB working together

Goal of Postgres-XC



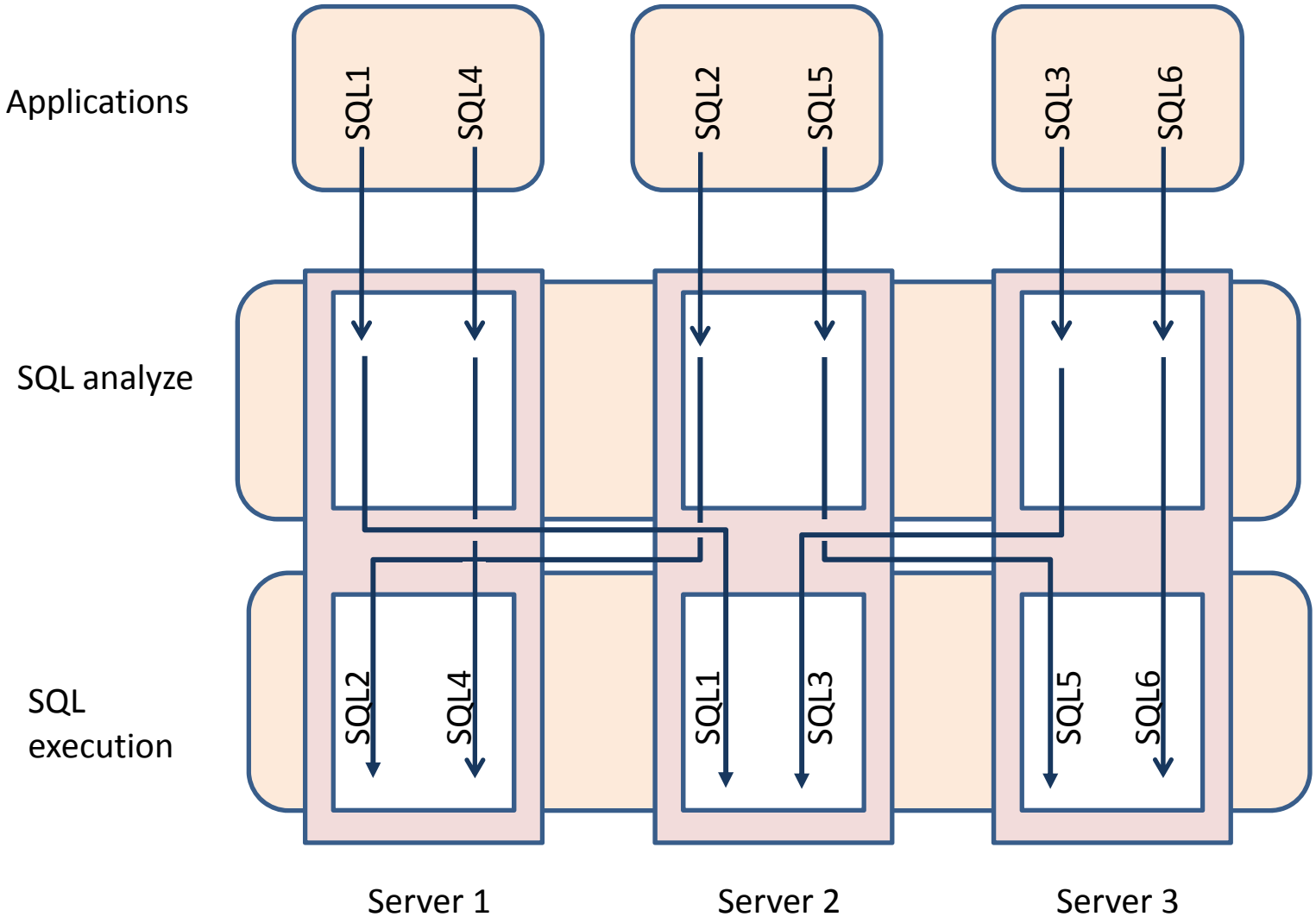
Current Status

- Version 0.91 – Now available
  - http://sourceforge.net/projects/postgres-xc/
  - Simple statement w/o cross-node operation
    - Sufficient to run DBT-1 and pgbench
  - Create Table
  - Copy
  - Replicated Table Update
- Version 0.92 – Coming Soon

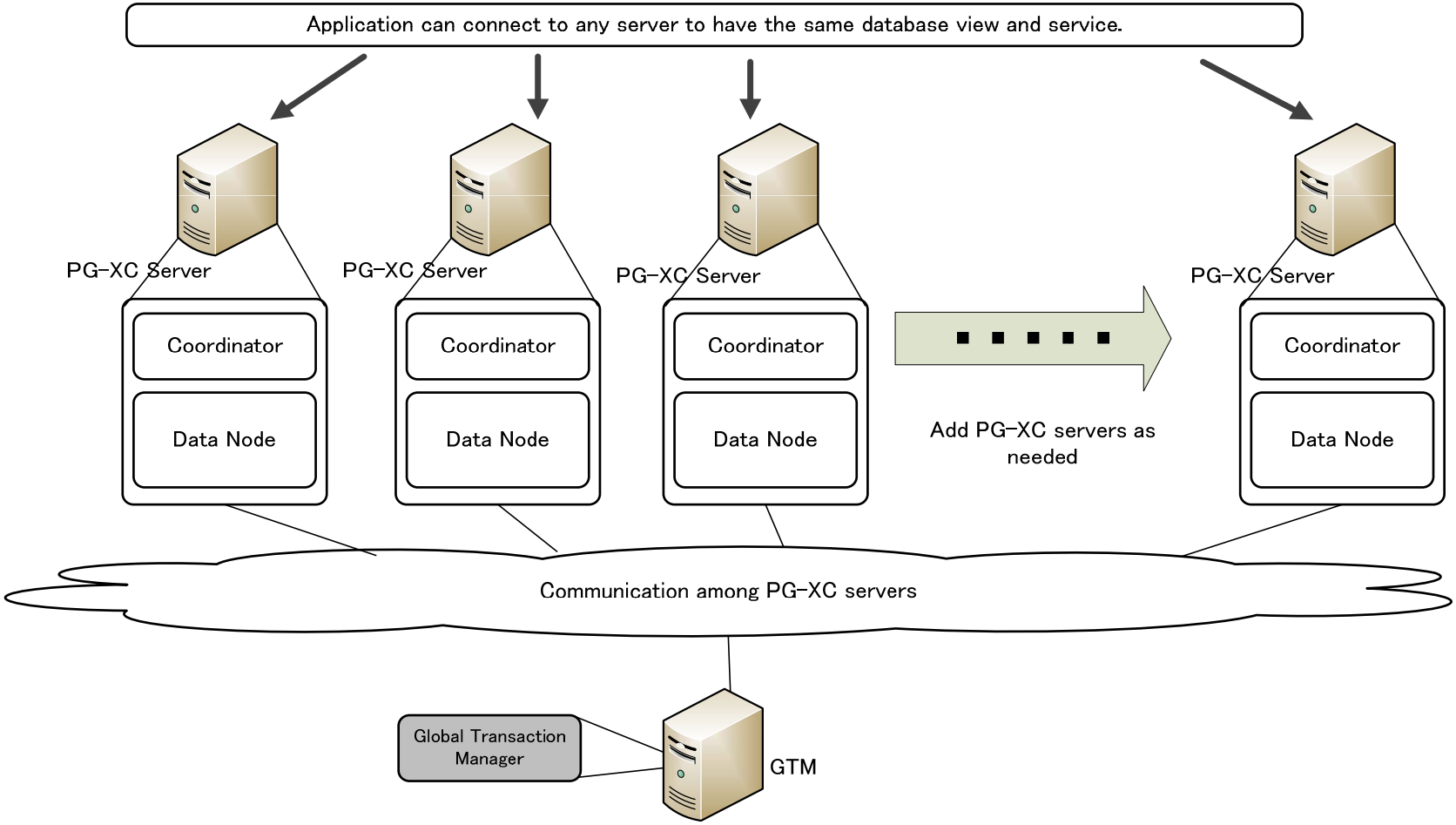
How to Achieve Read/Write Scalability

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

Parallel Transaction Execution



Postgres-XC Configuration Outline



Postgres-XC Components

- GTM (Global Transaction Manager)
  - Provide distributed MVCC feature to maintain transaction properties
    - Transaction ID
    - Snapshot
  - Provide other global data to statements
    - Sequence
    - Time/Sysdate (under plan)
- Coordinator
  - Parse statements and determine location of involved data
  - Transfer statements for each data node (if needed)
  - Application I/F
- Data Node
  - Store 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 in the statement basis (not WAL basis)
  - Distributed Table
    - Each tuple is assigned a Data Node to go
      - Based on a value of a column (distribution key)
        - » Hash
        - » Round-Robin
        - » Range (future)
        - » User-Defined (future)

How to Determine Distributed/Replicated?

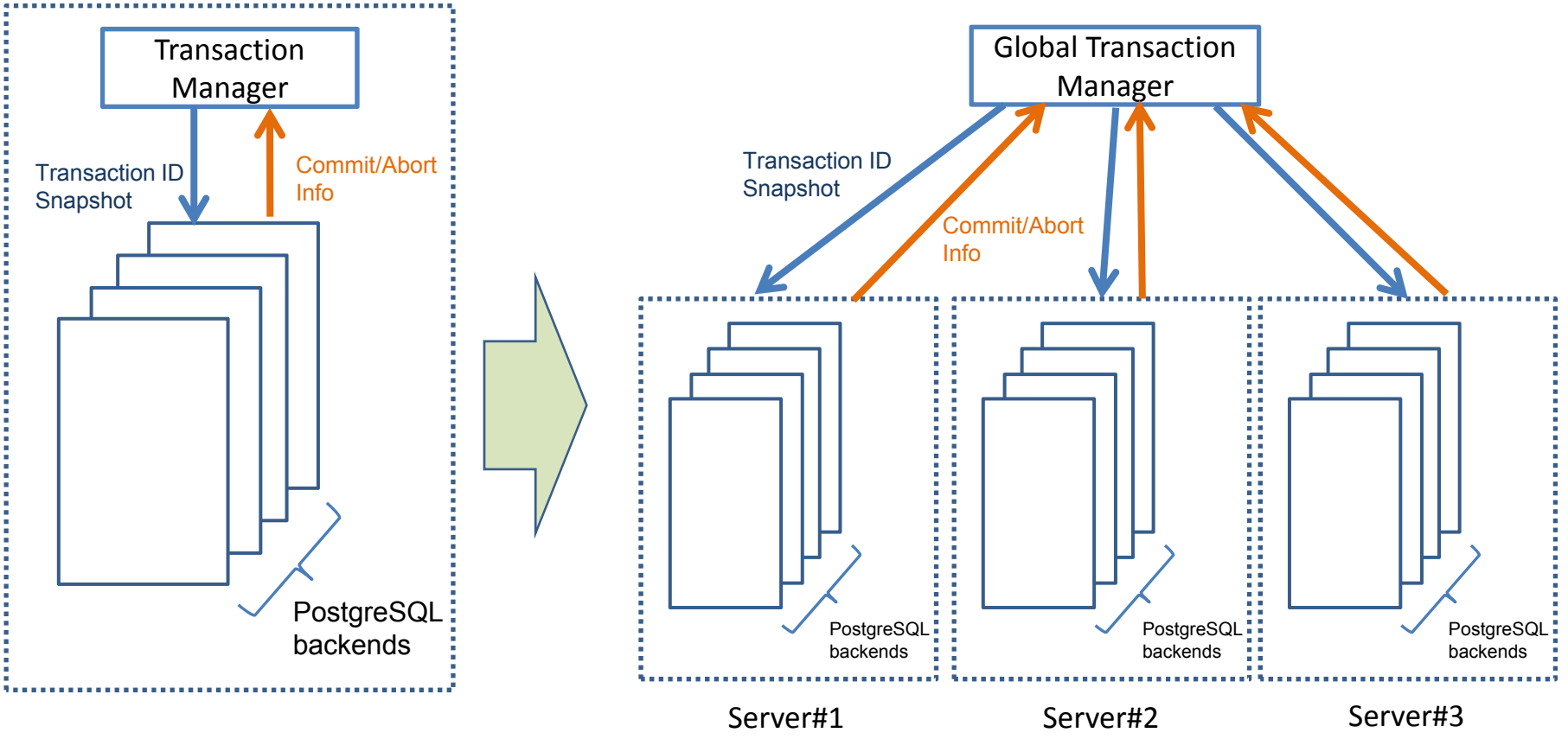
- Transaction tables may be partitioned so that each transaction can be executed in limited number of data nodes.
- Master tables may be replicated so that each transaction can read row values locally.

GTM – Global Transaction Manager

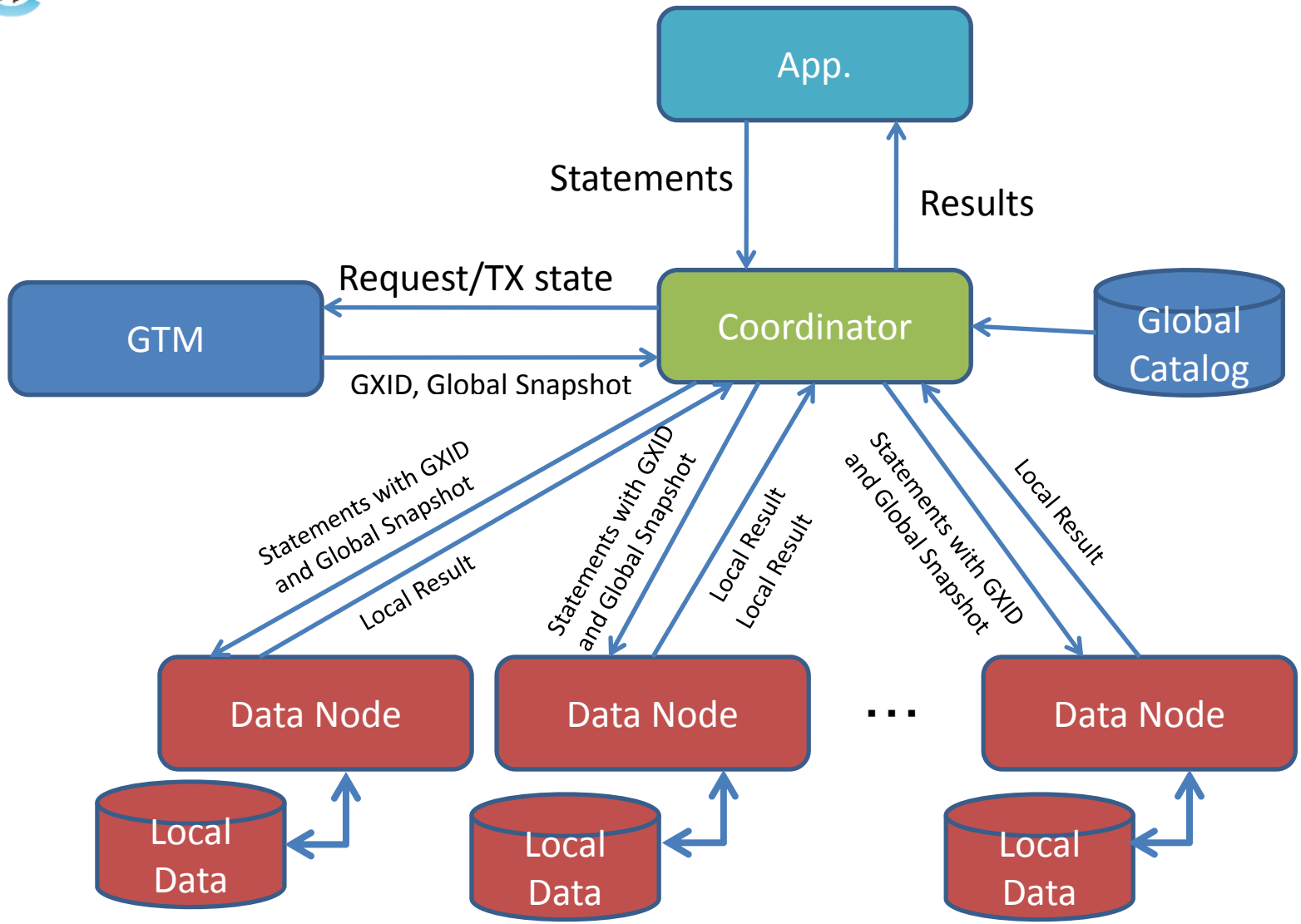
- GTM provides distributed MVCC capability to all the other nodes,
  - Extracted essential of transaction management feature of PostgreSQL
    - Unique Transaction ID (GXID, Global Transaction ID) assignment,
    - Gather transaction status from all the coordinators and maintain snapshot data,
    - Provide snapshot data to each transaction/statement (Global Snapshot).
  - Extract global value providing feature such as
    - Sequence
    - Time/sysdate (future)

Distributed MVCC - Outline

Extract PostgreSQL transaction management and apply globally



GTM and PG-XC Transaction Management



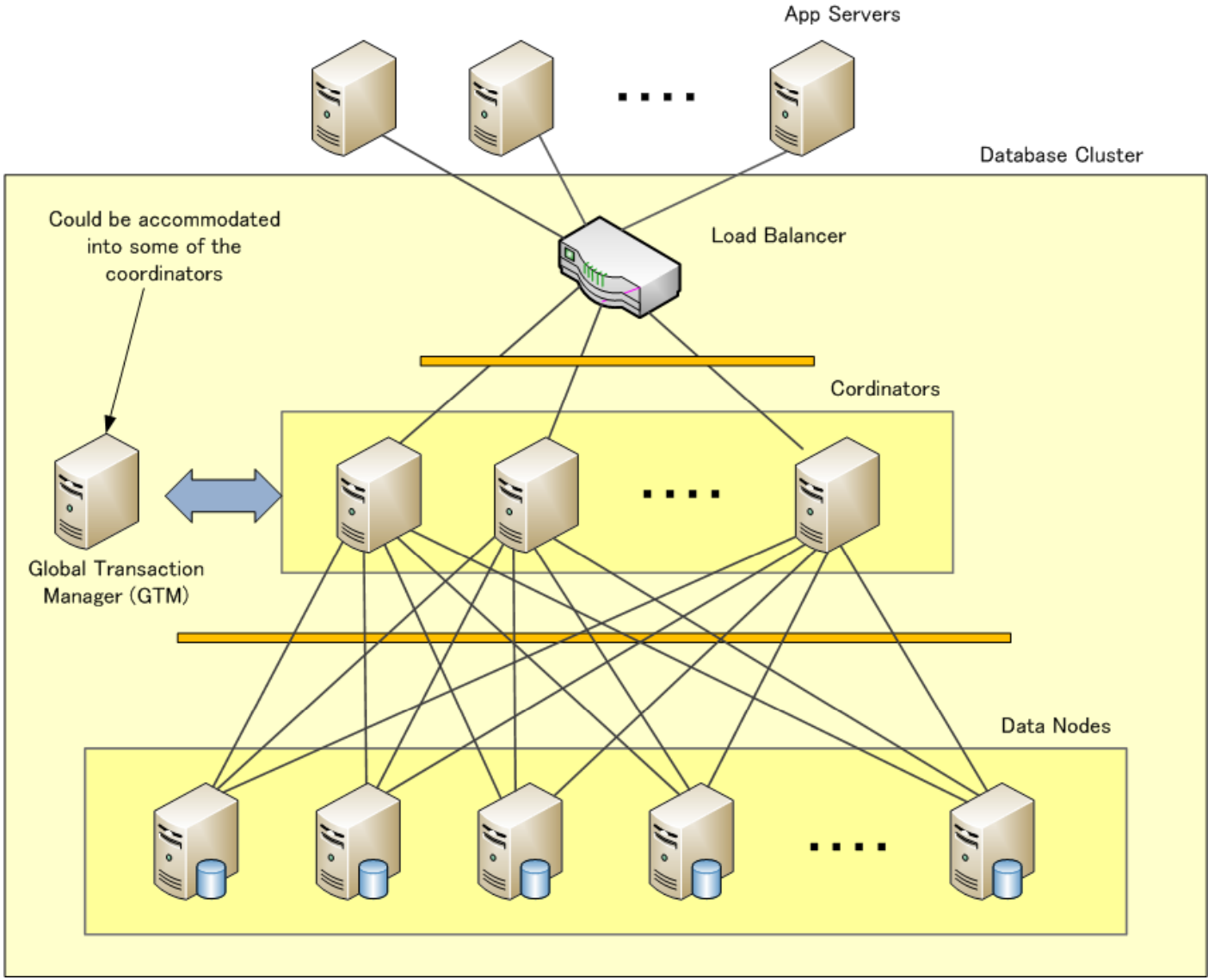
GXID and Snapshot

- 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 globally, same as standalone PostgreSQL.

Coordinator/Data Node Internals

Reference Architecture



Coordinator Overview

- Based on PostgreSQL 8.4.3
- 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

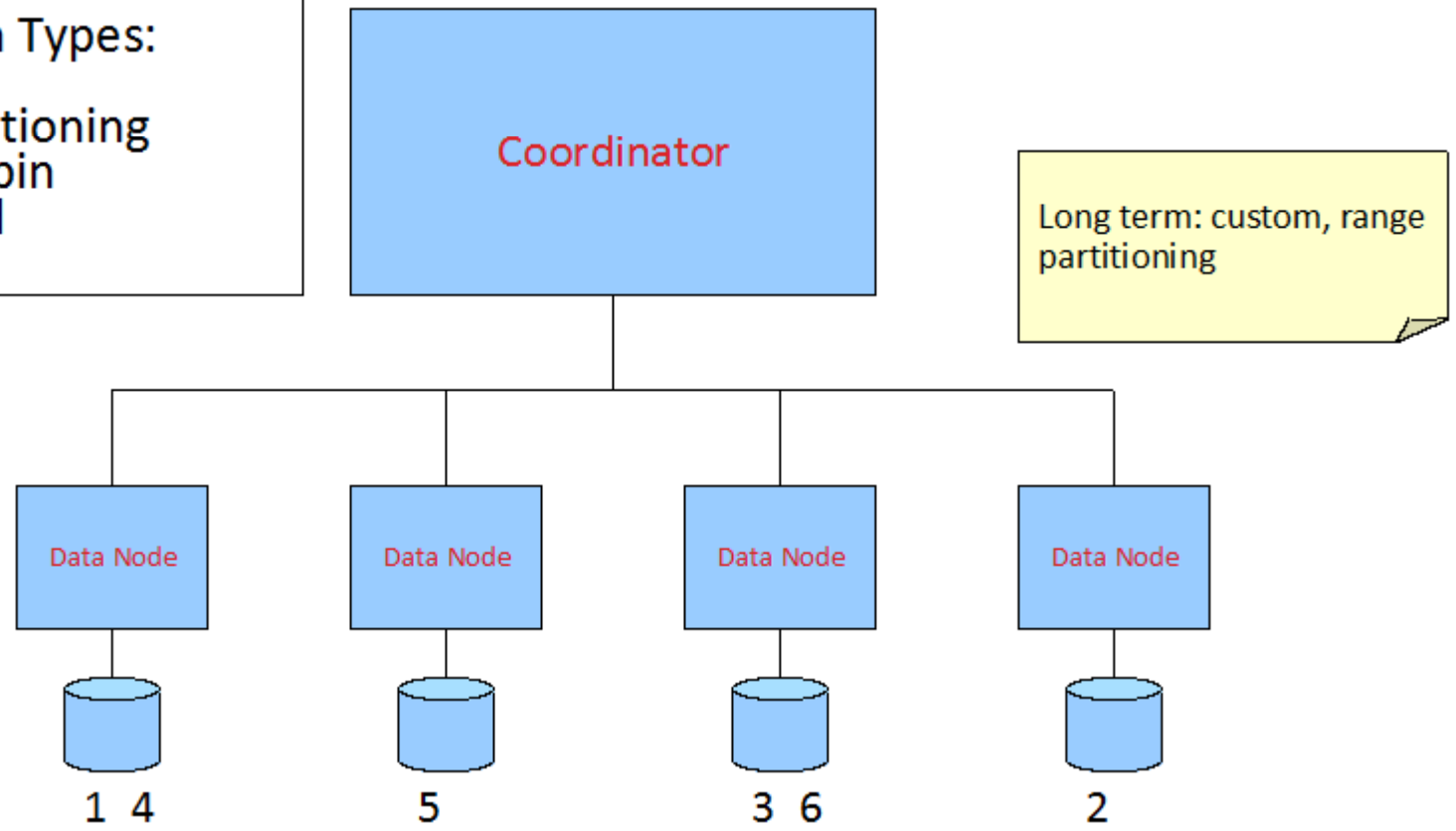
Data Node Overview

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

Data Distribution

Distribution Types:

- Hash Partitioning
- Round Robin
- Replicated



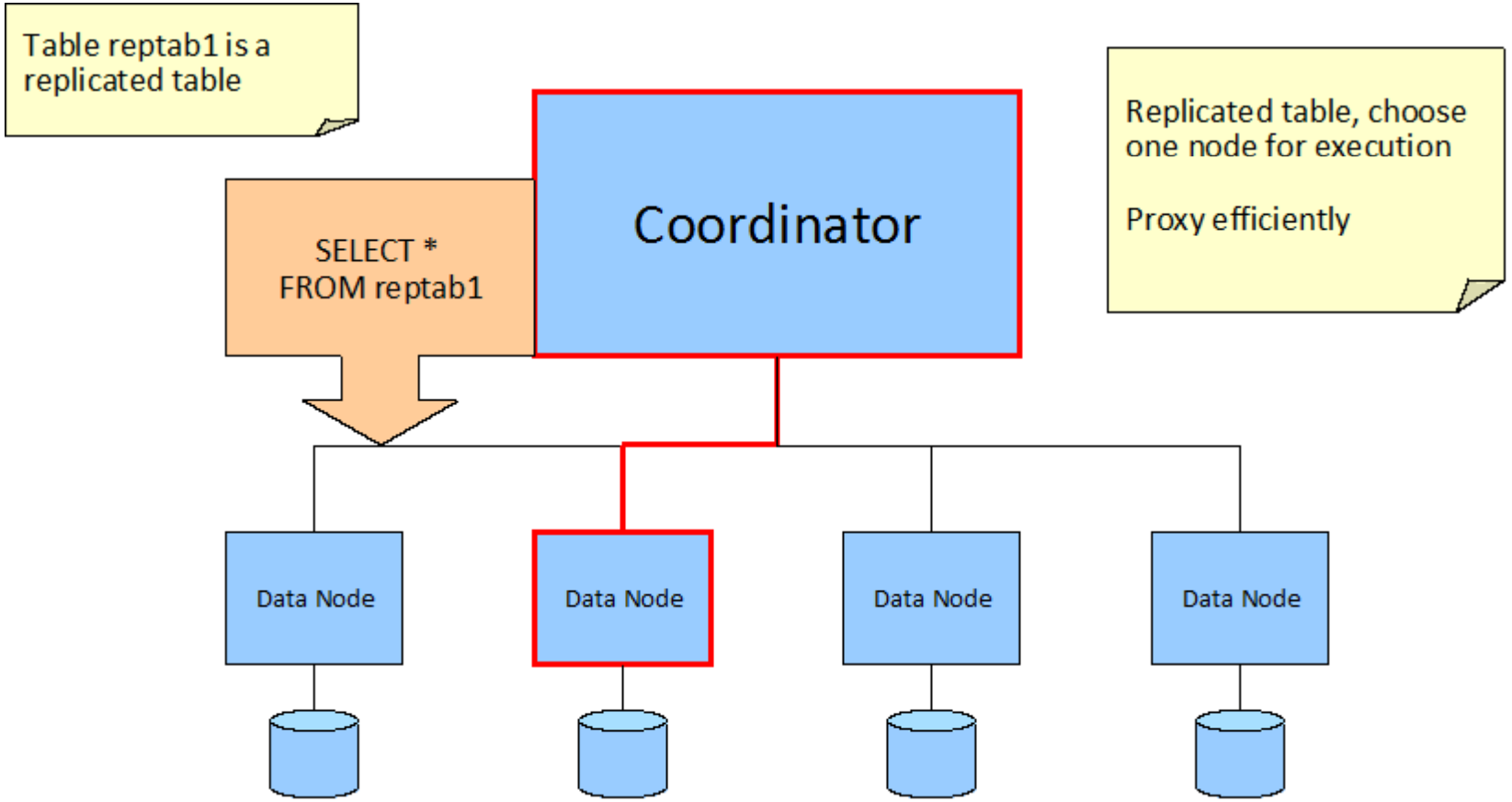
Connection Pooling

- 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

- Only basic statements currently handled
  - (no cross-node joins yet)
  - Single step execution only. No support if a statement has to be divided into multiple step of execution.
    - Will be improved in the next quarter.
- Use distribution information in Coordinator
- If more than one Data Node, send down statement to all simultaneously
- Recognize singleton statements
- Handle replicated tables
- Use two phase commit
  - (and use only when necessary)

Statement Handling - Execution



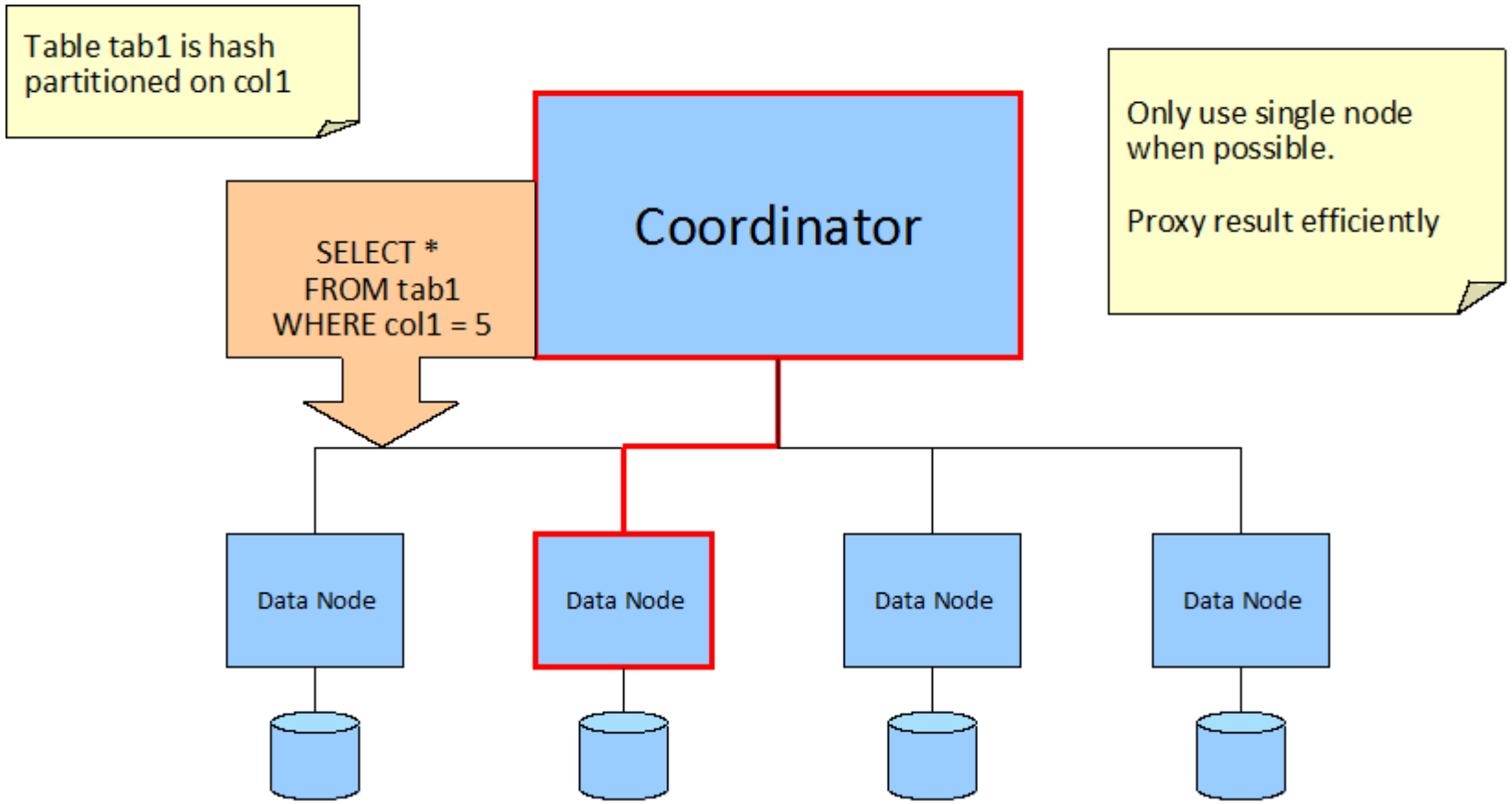
Queries with Replicated Tables

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

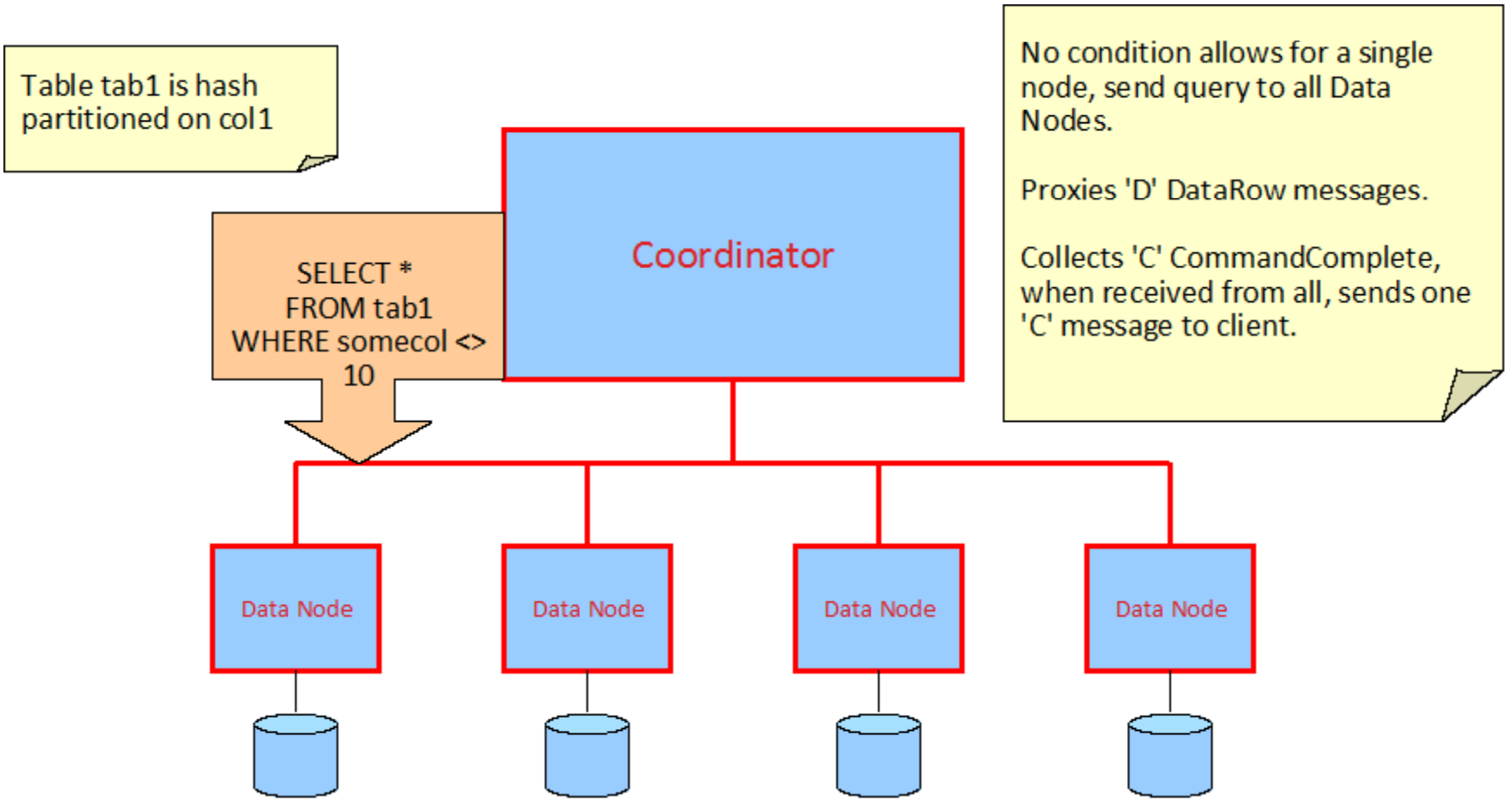
```
SELECT *
  FROM reptab1 r1 INNER JOIN reptab2 r2
    ON r1.col1 = r2.col2
```

- For write operations, use all nodes and two phase commit

Statement Handling - Execution



Statement Handling - Execution



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
  - If no Data Nodes have been written to, we do not send down any commit

*Stored functions could theoretically write to DB

Transaction Handling Considerations

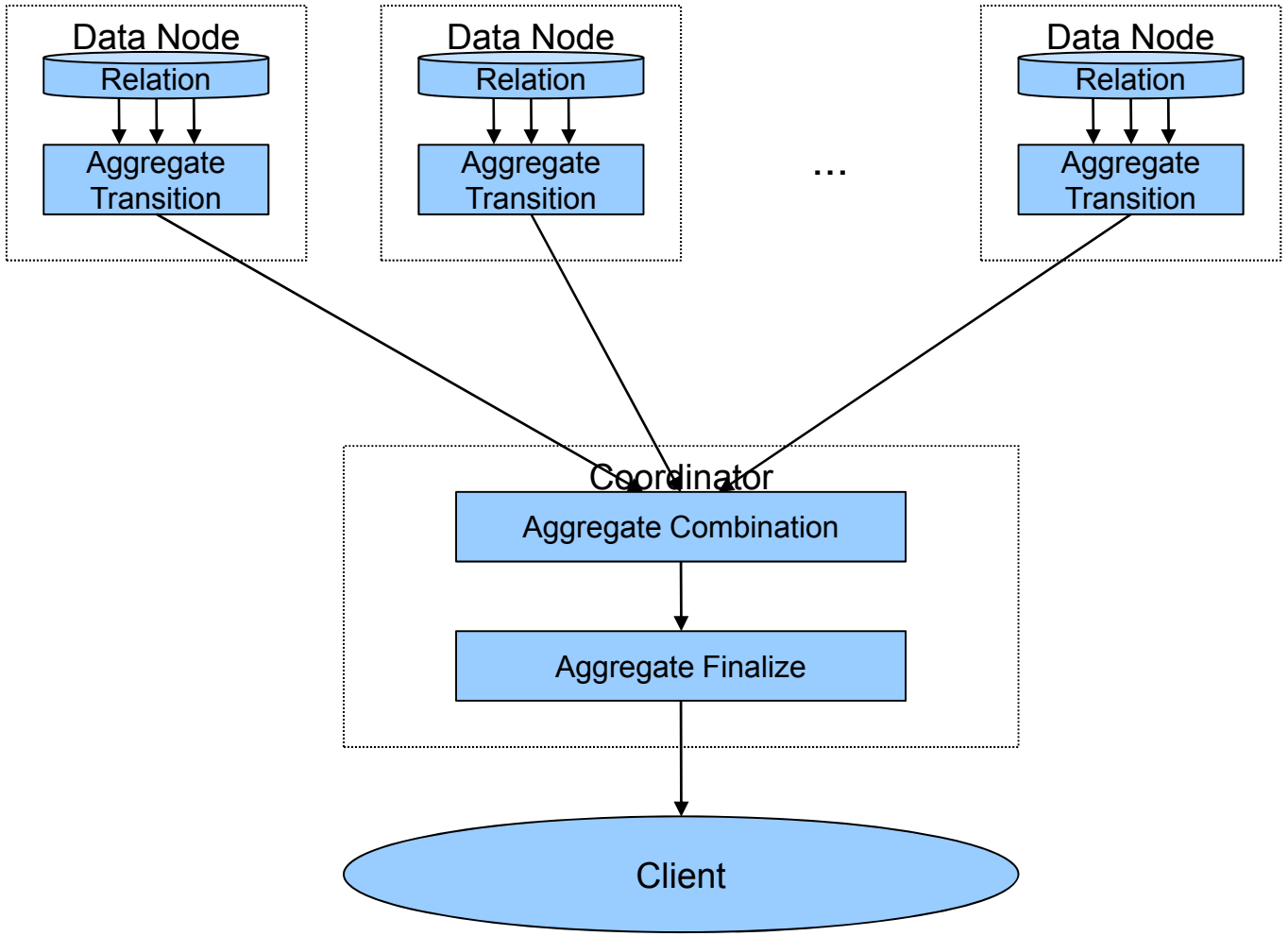
- Distributed transactions and two phase commit (2PC)
- Distributed Multi-Version Concurrency Control
  - Global Snapshots
  - Autovacuum
    - exclude XID in global snapshots
  - ANALZYE
  - Future optimization
  - CLOG
    - Careful when extending, not all transactions are on all nodes

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;
```

- Finalizer

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

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

Looking at Code

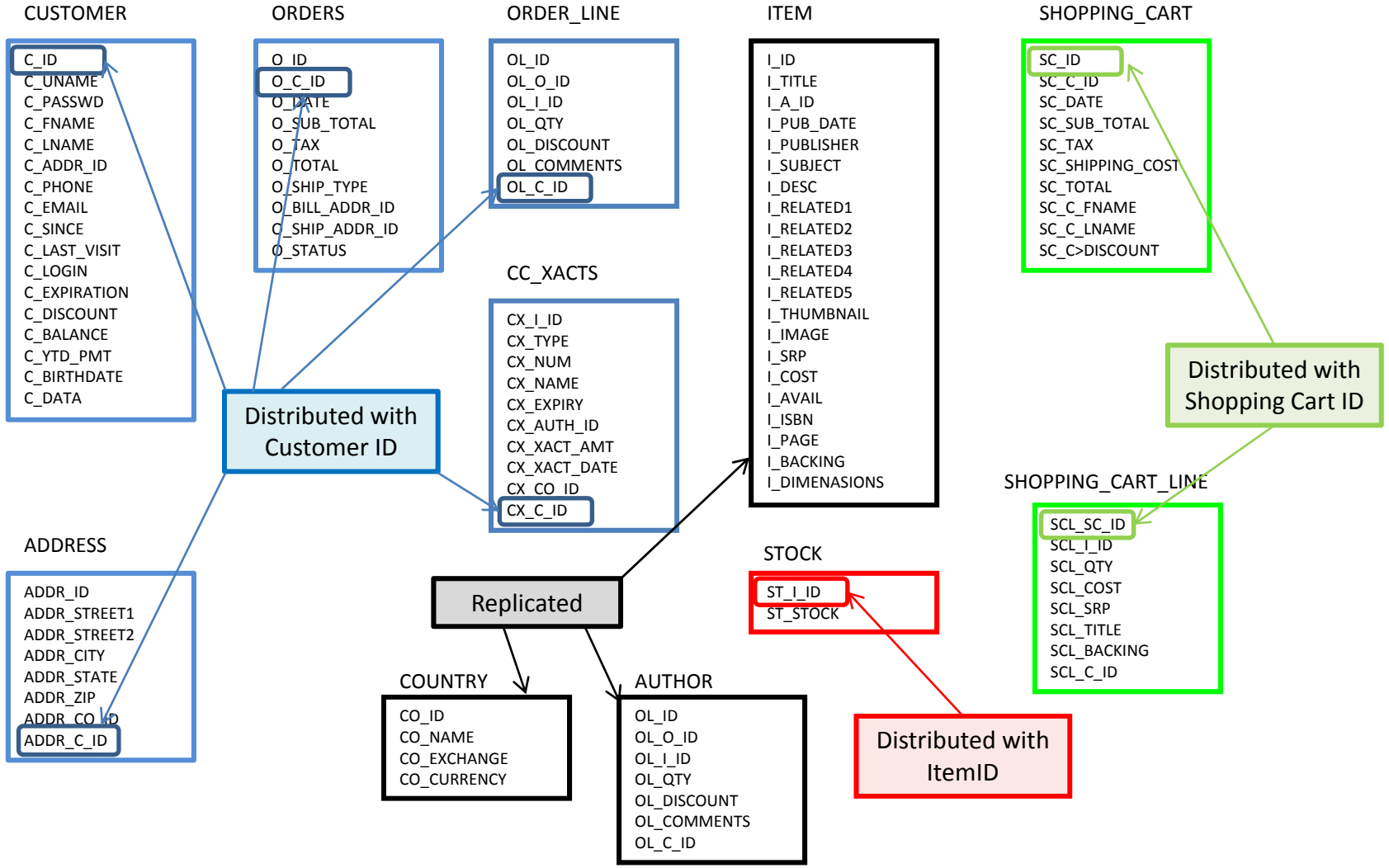
- Not (yet) overly invasive in PostgreSQL code
  - 8.4.2 → 8.4.3 merged cleanly
- Existing modules use `#ifdef PGXC` to identify Postgres-XC changes
- IS_PGXC_COORDINATOR and IS_PGXC_DATANODE easily identifies applicable code
- Advanced Coordinator logic in separate modules

Evaluation

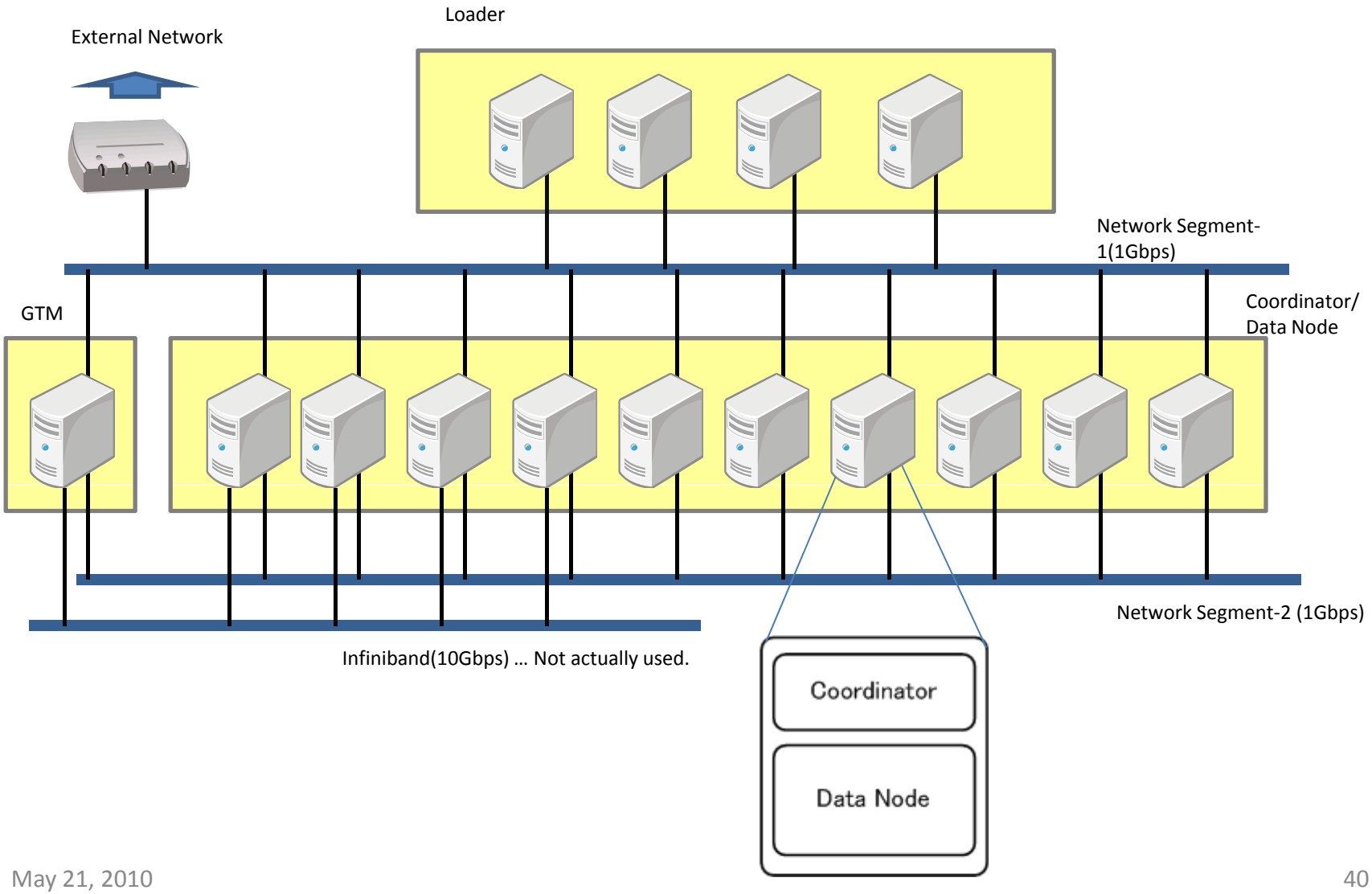
Postgres-XC Performance Benchmark

- 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



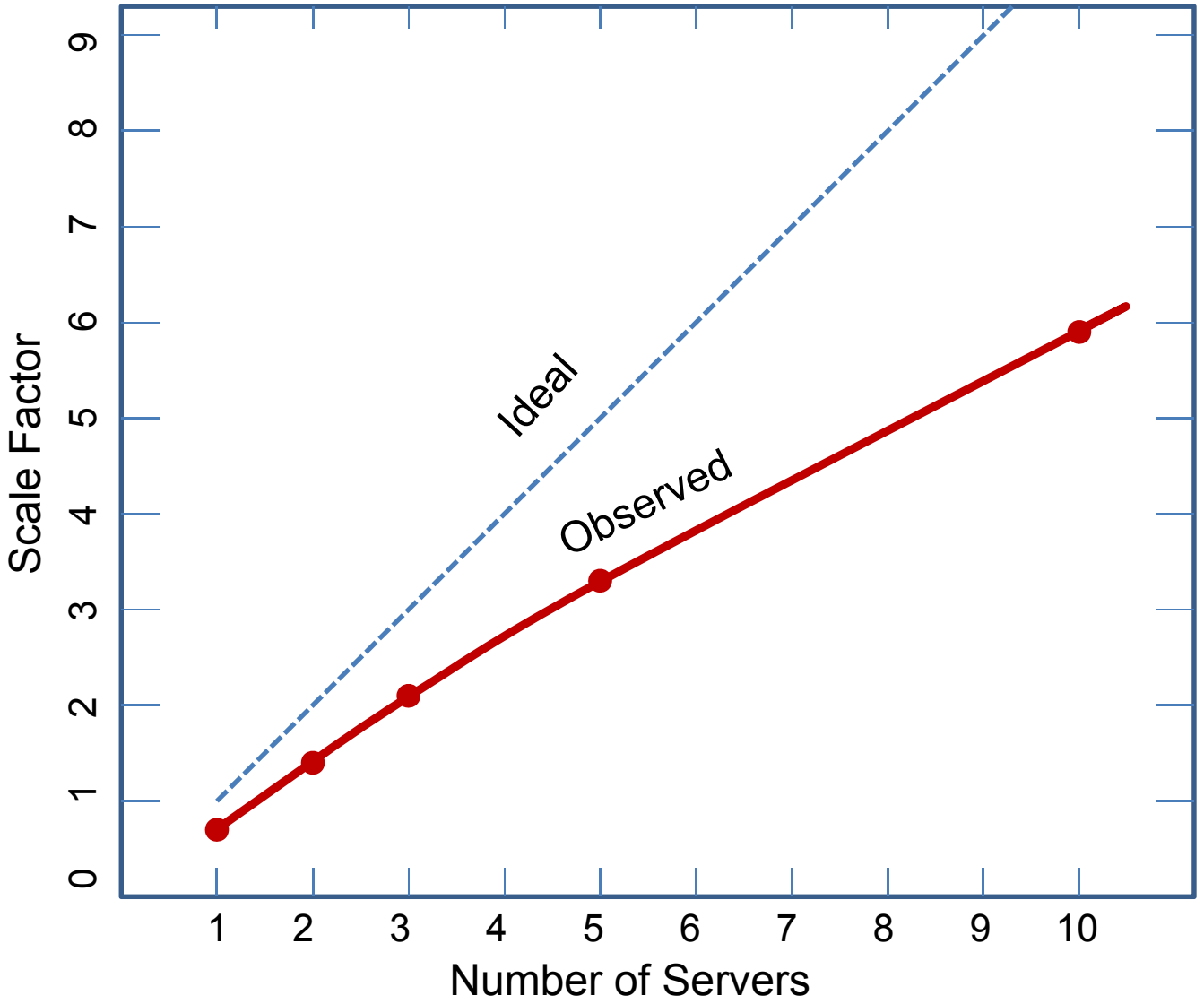
Evaluation Environment



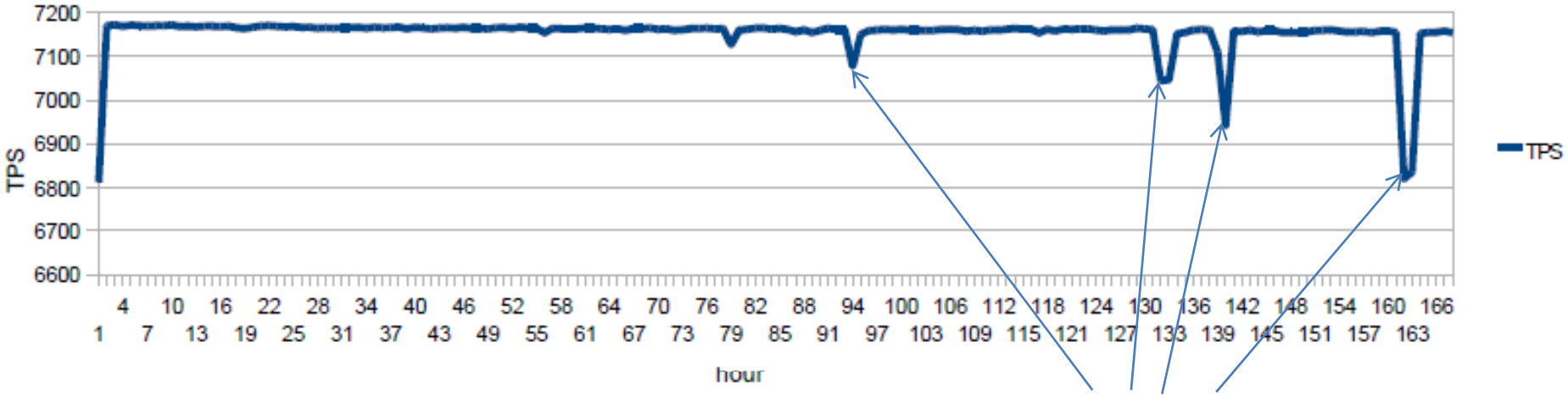
Server Spec

	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 14krpm x 2 ea

Scale Factor Summary



One Week Test



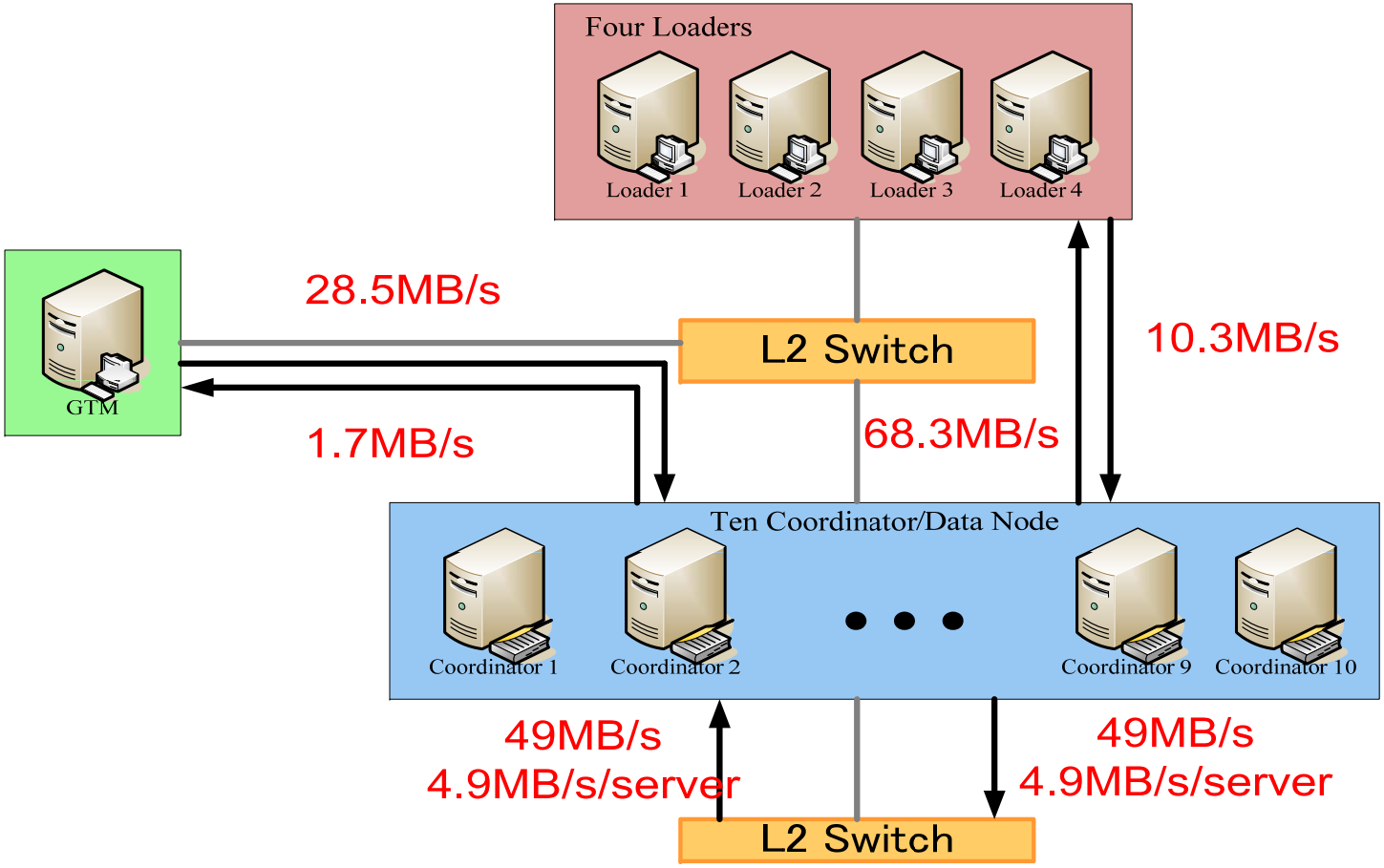
Vaccum Analyze may become long transactions to affect the throughput.

Reasonably stable in a long run (90% workload)

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

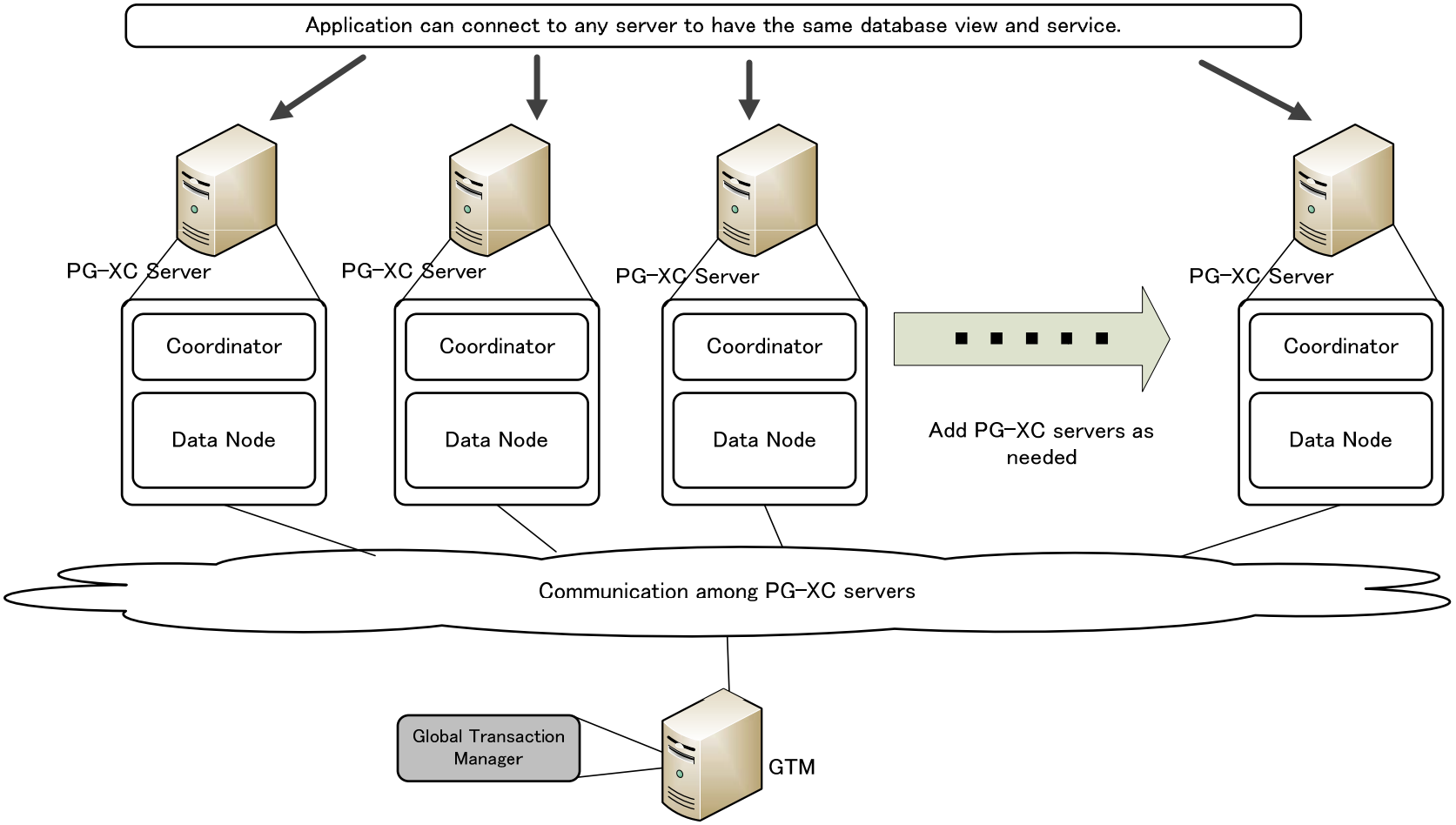
Network Workload



Evaluation Summary

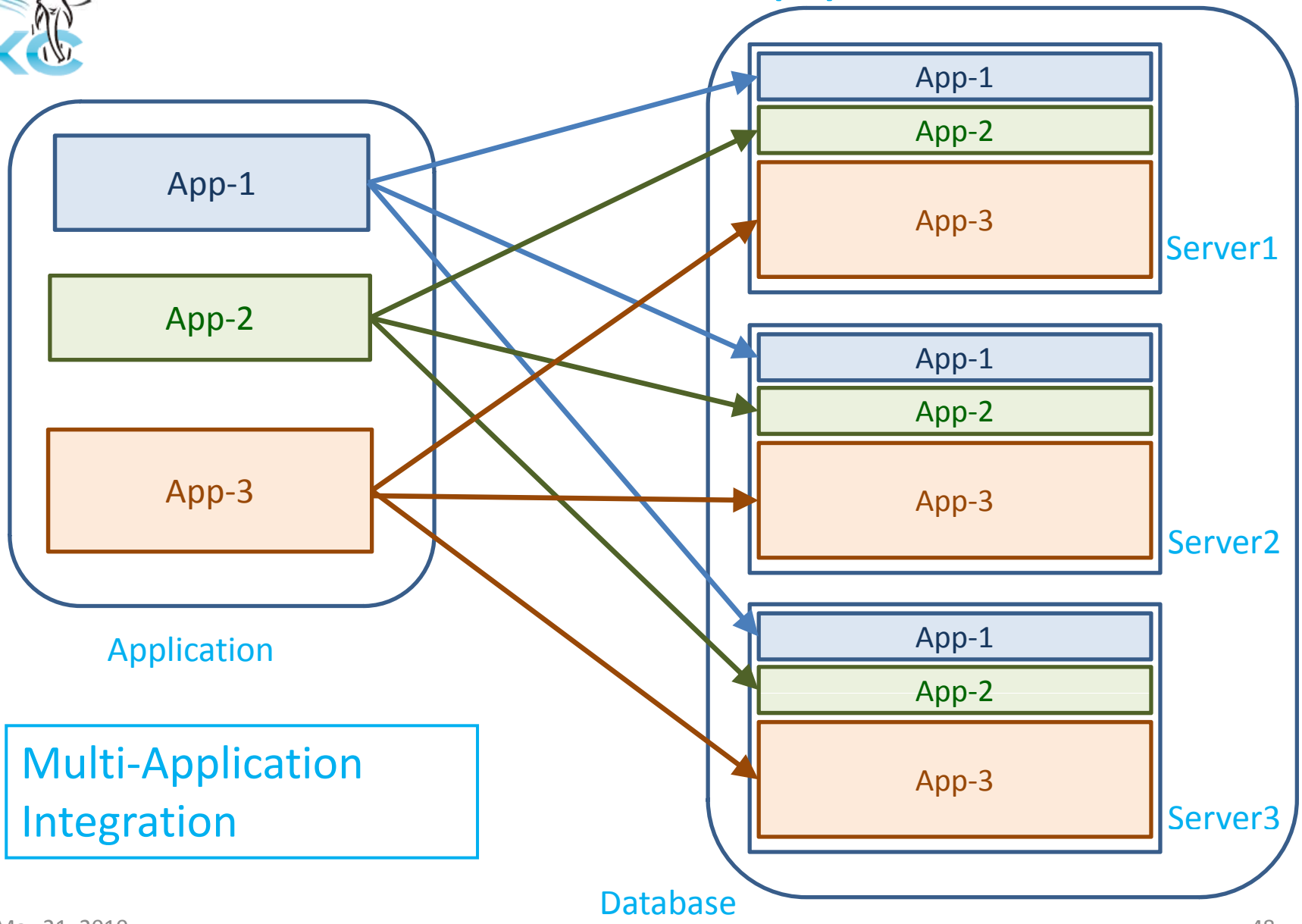
- PG-XC is reasonably scalable in both read/write.
- Need some tweak to stabilize the performance.
- Network workload is reasonable.
  - GTM Proxy works well.
  - More work is needed to accommodate more servers (thirty or more)
- Fundamentals are established
  - Will continue to extend statement support

Possible Use Case (1)



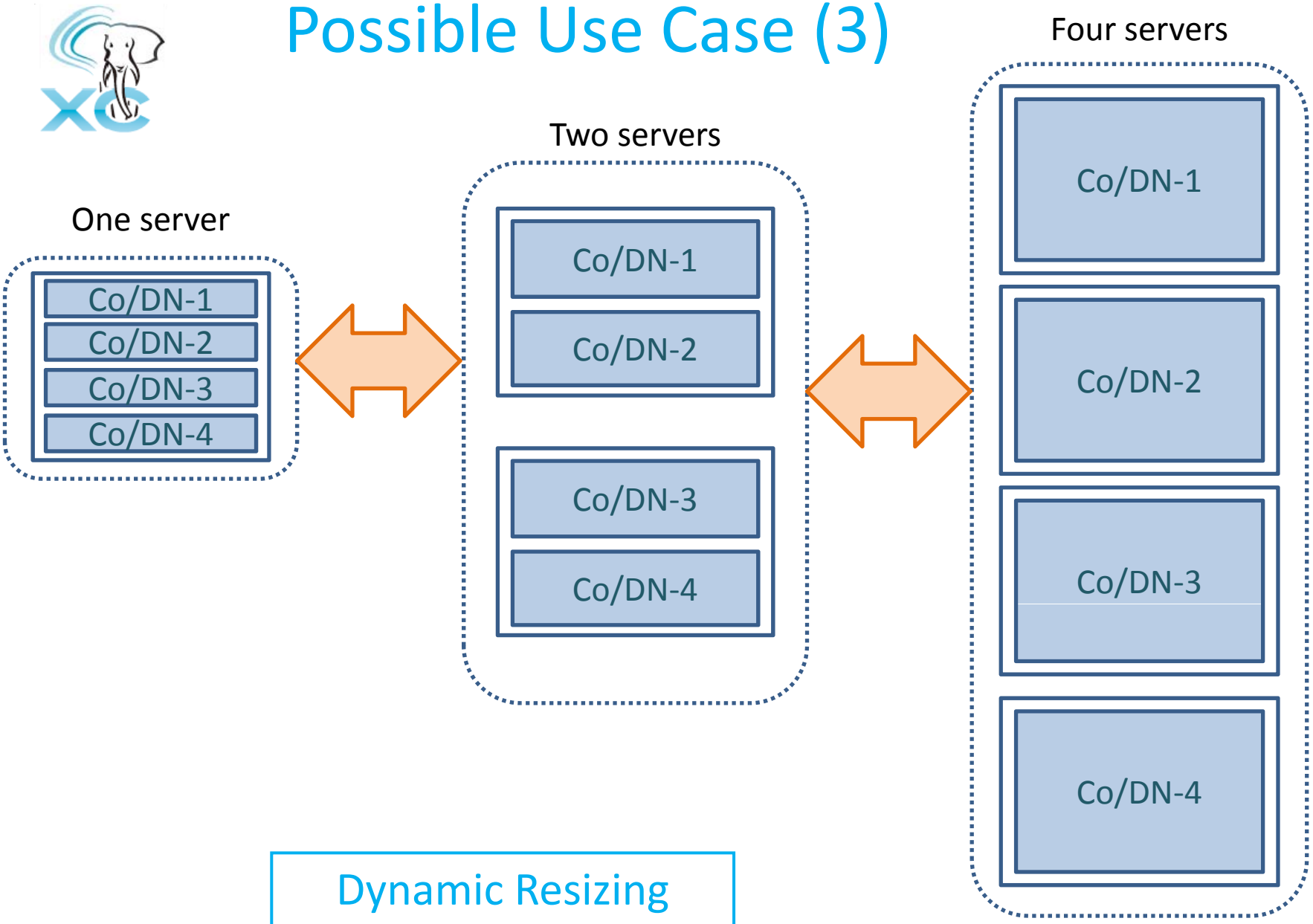
Large Scale Application

Possible Use Case (2)



Multi-Application Integration

Possible Use Case (3)



Dynamic Resizing (Cloud)

WIP

- WIP up to Version 0.92
  - ORDER BY/DISTINCT
  - Stored Functions (subset)
  - Subqueries (subset)
  - Views, Rules (subset)
  - Logical Backup/Restore
  - Aggregate Functions

Roadmap and Plan (1)

- Toward V.0.93 (Sept., 2010)
  - General Statement
    - SELECT statement
  - Cluster-Wide Installer
  - Cluster-Wide Operation Utilities
  - Regression Tests
  - Basic Cross-Node Operation (SELECT)
  - Forward Cursor w/o ORDER BY
  - Extended Query Protocol
    - JDBC
  - Global Timestamp
  - Drivers
    - ECPG, JDBC, PHP, etc.
  - Forward Cursor
  - 2PC from Apps
  - DDL Synchronization

Roadmap and Plan (2)

- Toward V.1.0 (Dec., 2010)
  - General DML Statement
    - UPDATE/INSERT/DELETE
    - PREPARE/EXECUTE etc.
  - Forward Cursor with ORDER BY
  - Session Parameters
  - Prepared Statement
  - General Functions
  - Session Parameters
- Beyond V.1.0
  - Physical Backup/Restore
    - PITR
  - Cross-node operation optimization
  - TEMP tables
  - Backward Cursor
  - Batch, Statement pushdown
  - Trigger
  - Global constraints
  - Tuple relocation
    - Distribute key update
  - HA solutions

Interesting Remarks with SQL/MED

- Postgres-XC vs SQL/MED
  - Tightly-coupled vs Autonomous
  - R/W vs Read-centric
  - Single application vs Independent applications

Nevertheless…

- Postgres-XC and SQL/MED shares cross-node operation
  - First SQL/MED effort is applicable to Postgres-XC as well.
    - Need to add global transaction feature.
    - Targeted to V.1.1.
  - Upcoming Postgres-XC effort can be brought to SQL/MED.
    - Postgres-XC targets only (a kind of) PostgreSQL database so our SQL/MED may need some more work to apply.

Developers Welcome

- We welcome people helping 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

Thank You Very Much