NTT

EnterpriseDB®

POSTGRES XC

Write-scalable, synchronous multi-master PostgreSQL cluster with shared nothing approach

Koichi Suzuki

Mason Sharp

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? (1)

- 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

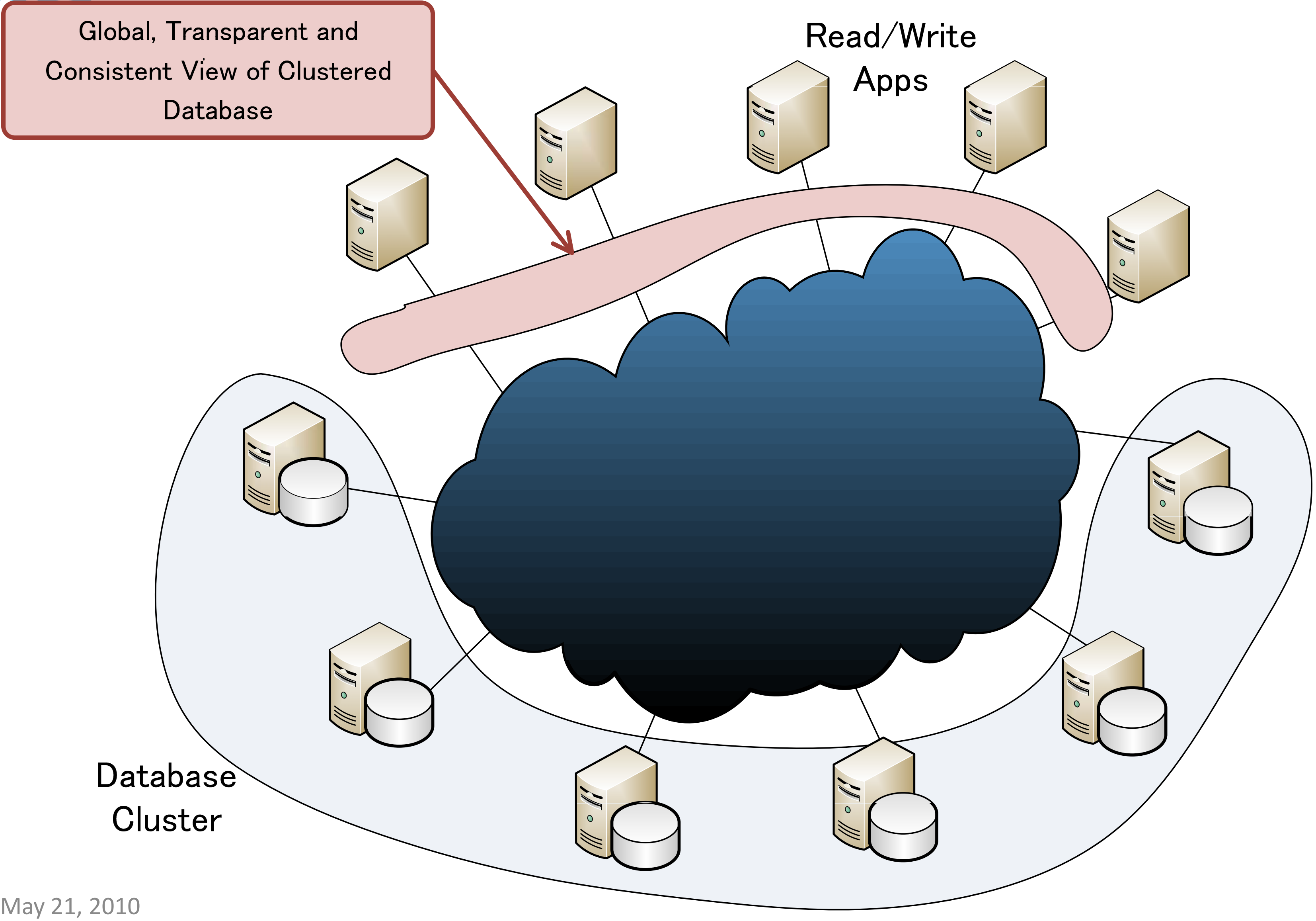
What is Postgres-XC? (2)

- Table location transparent
  - Tables can be replicated or distributed (partitioned)
    - Best utilize parallelism among involved servers.
  - 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-1/2 etc.)
  - Transactions can be executed in parallel in multiple data nodes.
- Complicated data warehouse (DBT-3 etc.)
  - Statement can be divided into several pieces executed in parallel in multiple data nodes.
    - (Statement handling not available yet.)

Goal of Postgres-XC



Global, Transparent and Consistent View of Clustered Database

Why Write-Scalability?

- Many application could be write-traffic bottleneck such as –
  - Access log in BLOG/SNS
  - Mission critical systems like internet shopping site, telephone customer billing, call information and securities trade
- Now application has to deal with such write bottleneck using multi-database.
  - Not distribution-transparent.
  - No cluster-specific application codes.
- As applications grow
  - It is desirable to make database distribution transparent for write operations too.

- But it is complicated …

Why Shared-Nothing?

- Most Cost-Efficient
  - Configurable only with commodity hardware
  - No need for dedicated disk system
  - No need for dedicated interface
- Flexibility to deploy
  - Can apply very simple to complicated cluster configuration

Current Status and Plan

- Version 0.9 is available
  - http://sourceforge.net/projects/postgres-xc/
  - Simple statement w/o cross-node operation
    - Sufficient to run DBT-1 and pgbench
  - Create Table
- Version 0.91 – Now available (Wednesday)
  - Copy
  - Aggregate Functions
  - Replicated Table Update

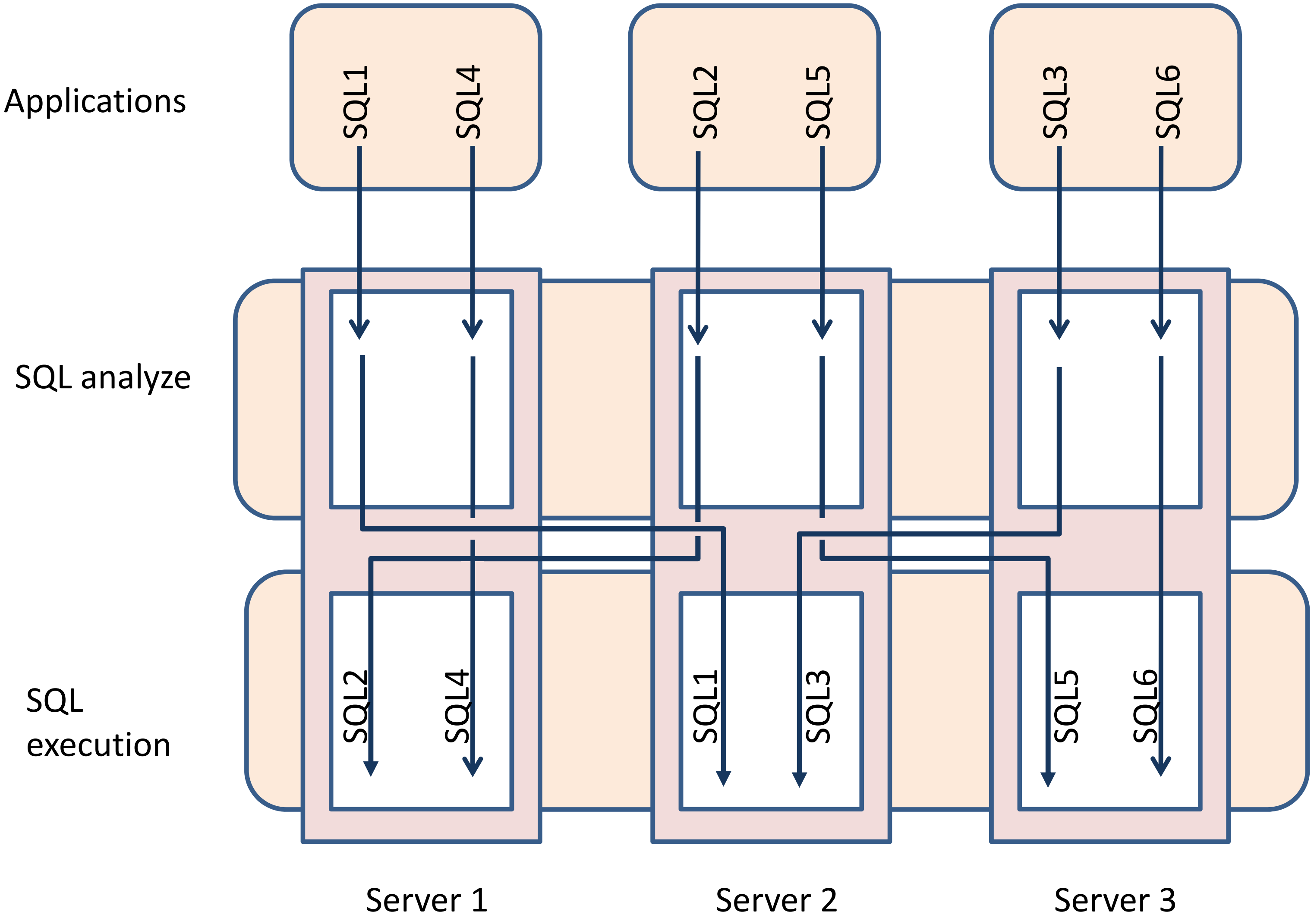
Current Status and Plan (cond.)

- Version 1.0 - July, 2010
  - Order By, Distinct
  - Subqueries
  - Views/Rules
  - DDLs
- Version 1.1 – Sept, 2010 (Still planning)
  - Installer
  - Operation Utilities
  - Dump/Restore (logical)
  - Cross-node operation (basic)
  - Extended Query Protocol (JDBC)
  - Global Timestamp
  - Simple Cursor (ECPG, JDBC, PHP, etc.)

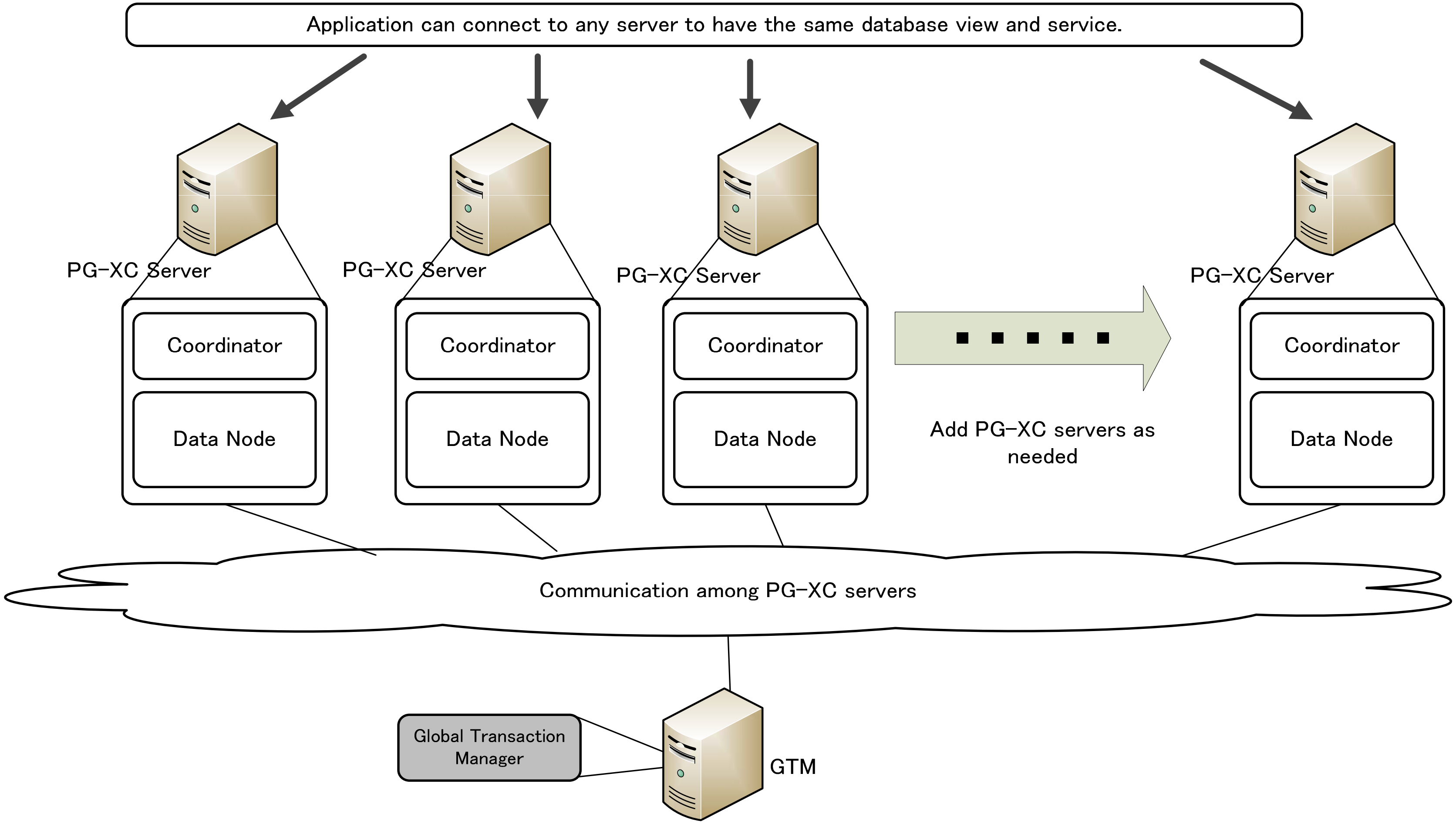
How to Achieve Read/Write Scalability

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

Parallel Transaction Execution



Postgres-XC Configuration Outline



Application can connect to any server to have the same database view and service.

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 (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)

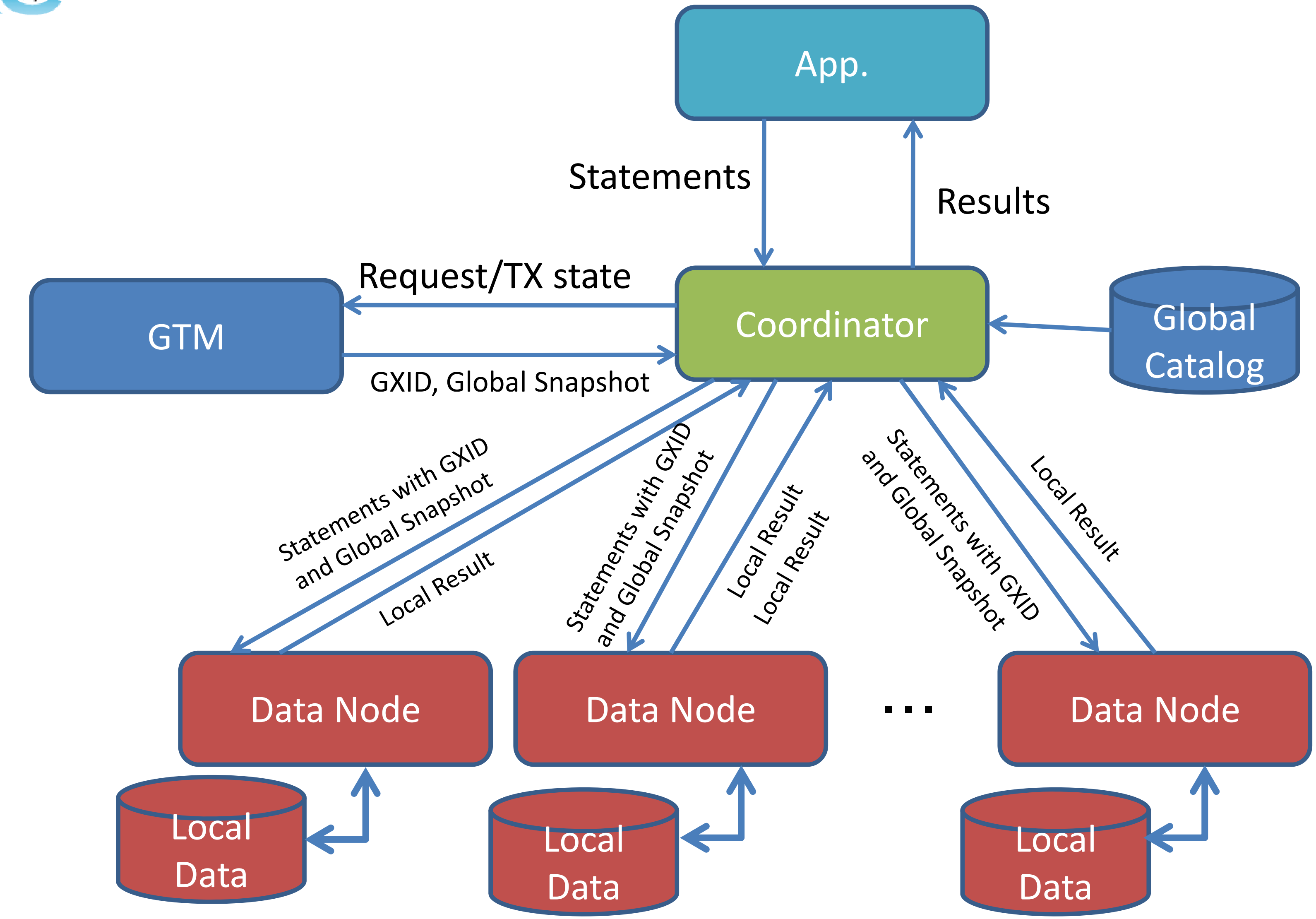
How to Determine Distributed/Replicated?

- 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

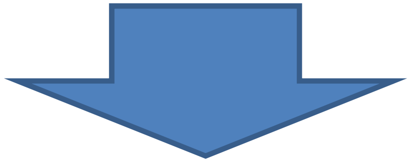
- GTM is the key of Postgres-XC transaction management
  - Extracted essential of transaction management feature of PostgreSQL
    - Unique Transaction ID (GXID, Global Transaction ID) assignment,
    - Gather transaction status from all the coordinators and maintain snapshot data,
    - Provide snapshot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

GTM and PG-XC Transaction Management



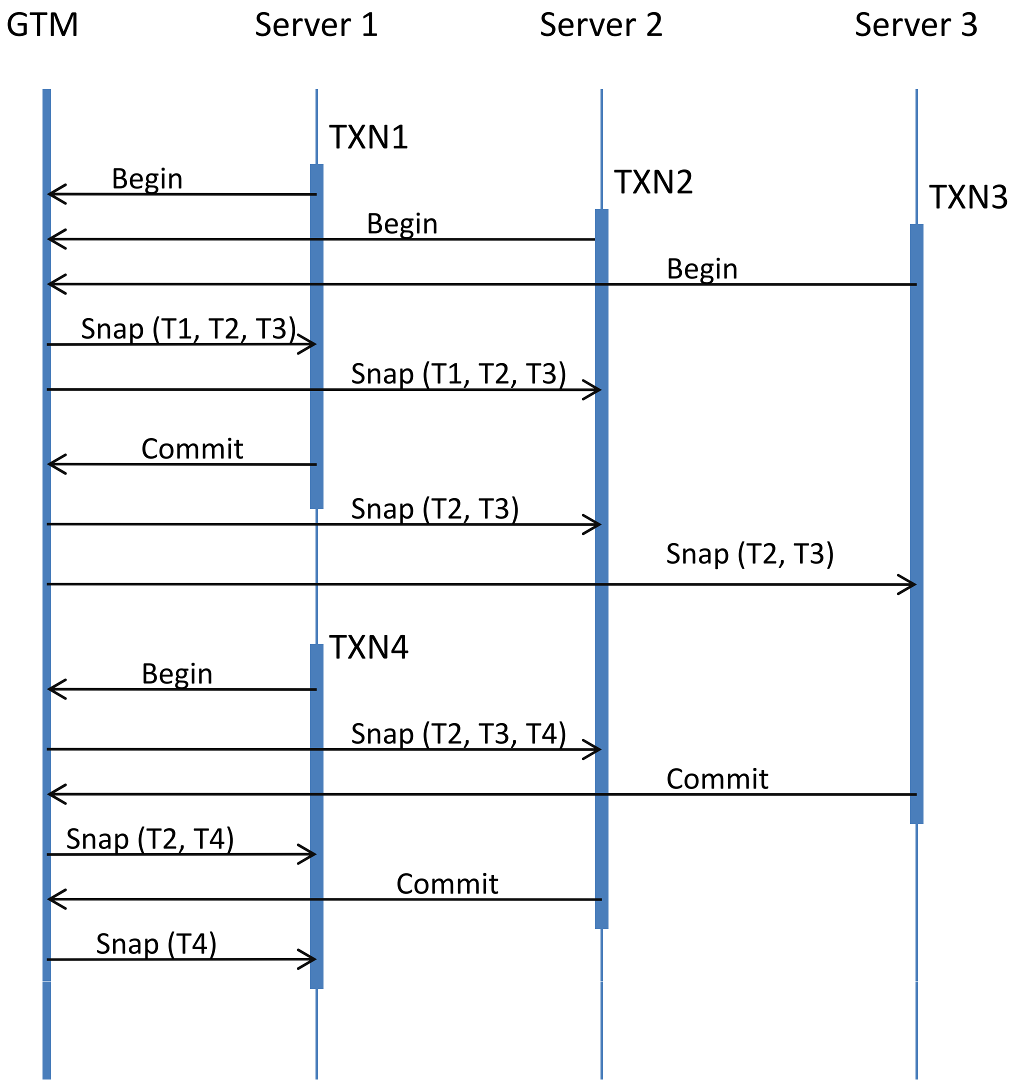
GXID and Snapshot

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



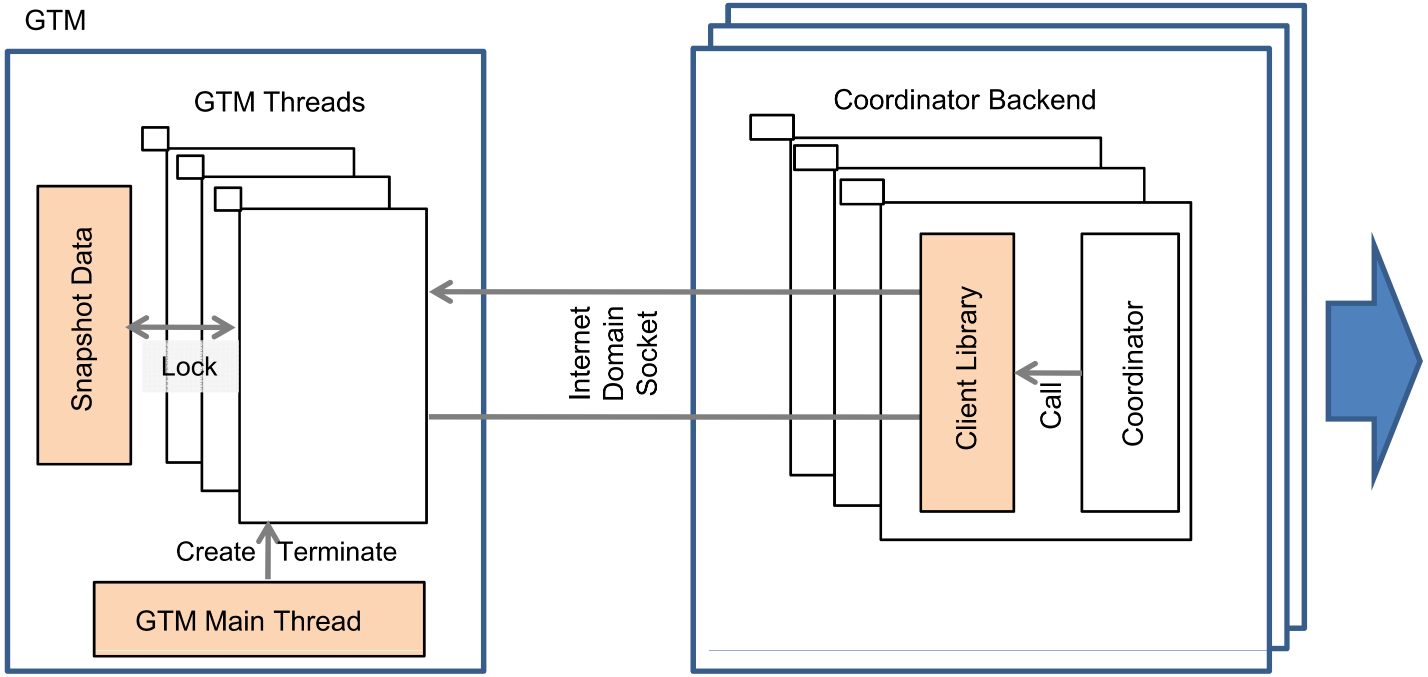
- Data node can handle transactions from different coordinators without consistency problem.
- Visibility is maintained globally, same as standalone PostgreSQL.

Outline of PG-XC Transaction Management



Can GTM be a Performance Bottleneck?

- Depending on implementation
  - Current Implementation

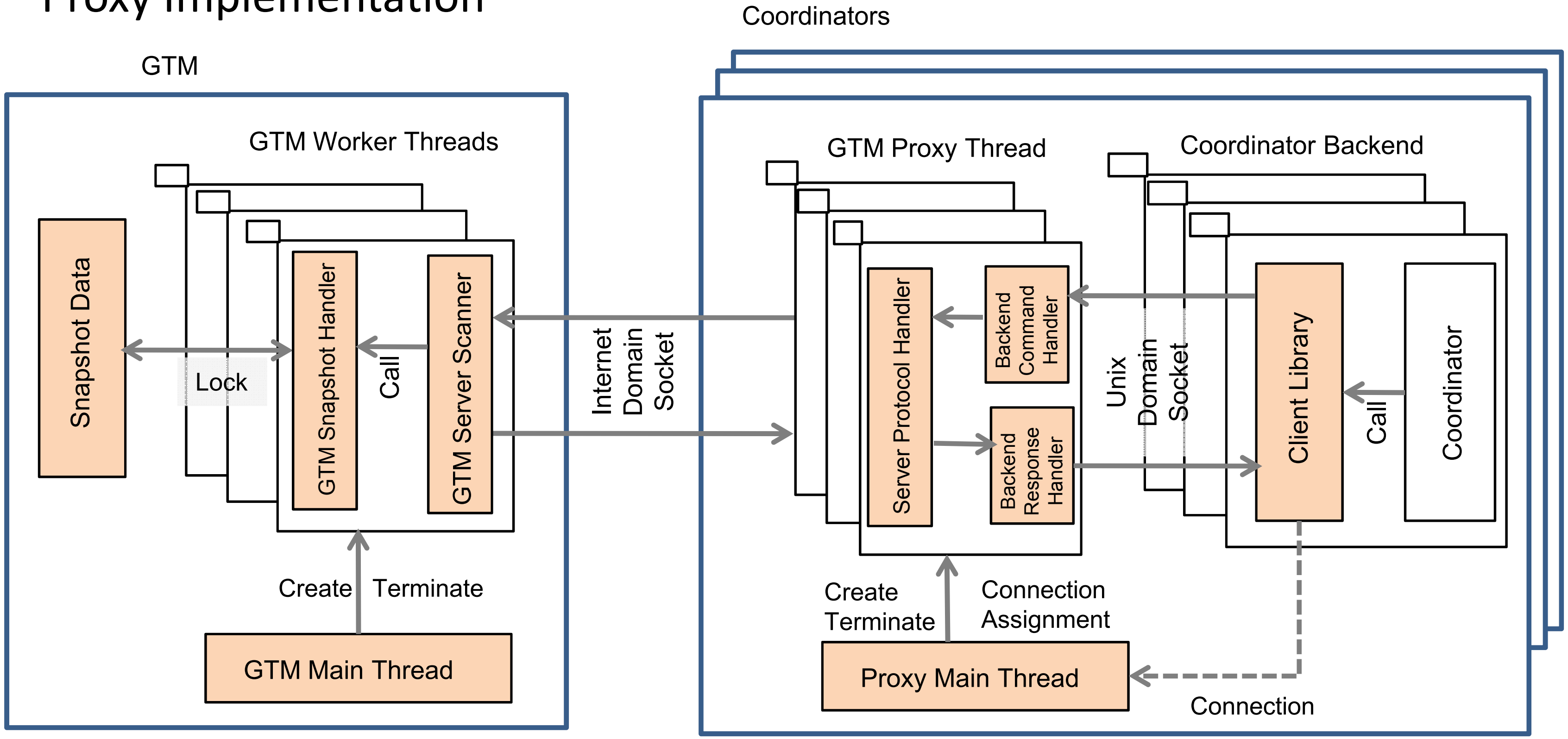


Applicable up to five PG-XC servers (DBT-1)

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

Can GTM be a Performance Bottleneck?

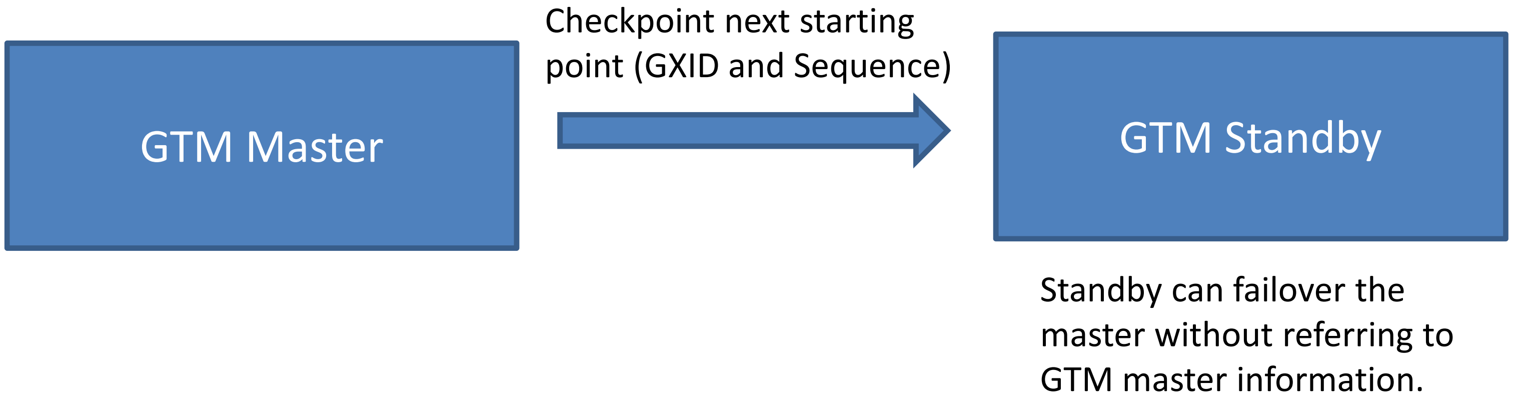
- Proxy Implementation



- Very good potential
  - Request/Response grouping
  - Single representative snapshot applied to multiple transactions
- Maybe applicable for more than ten PG-2 servers

Can GTM be a SPOF?

- Simple to implement GTM standby



GTM Master

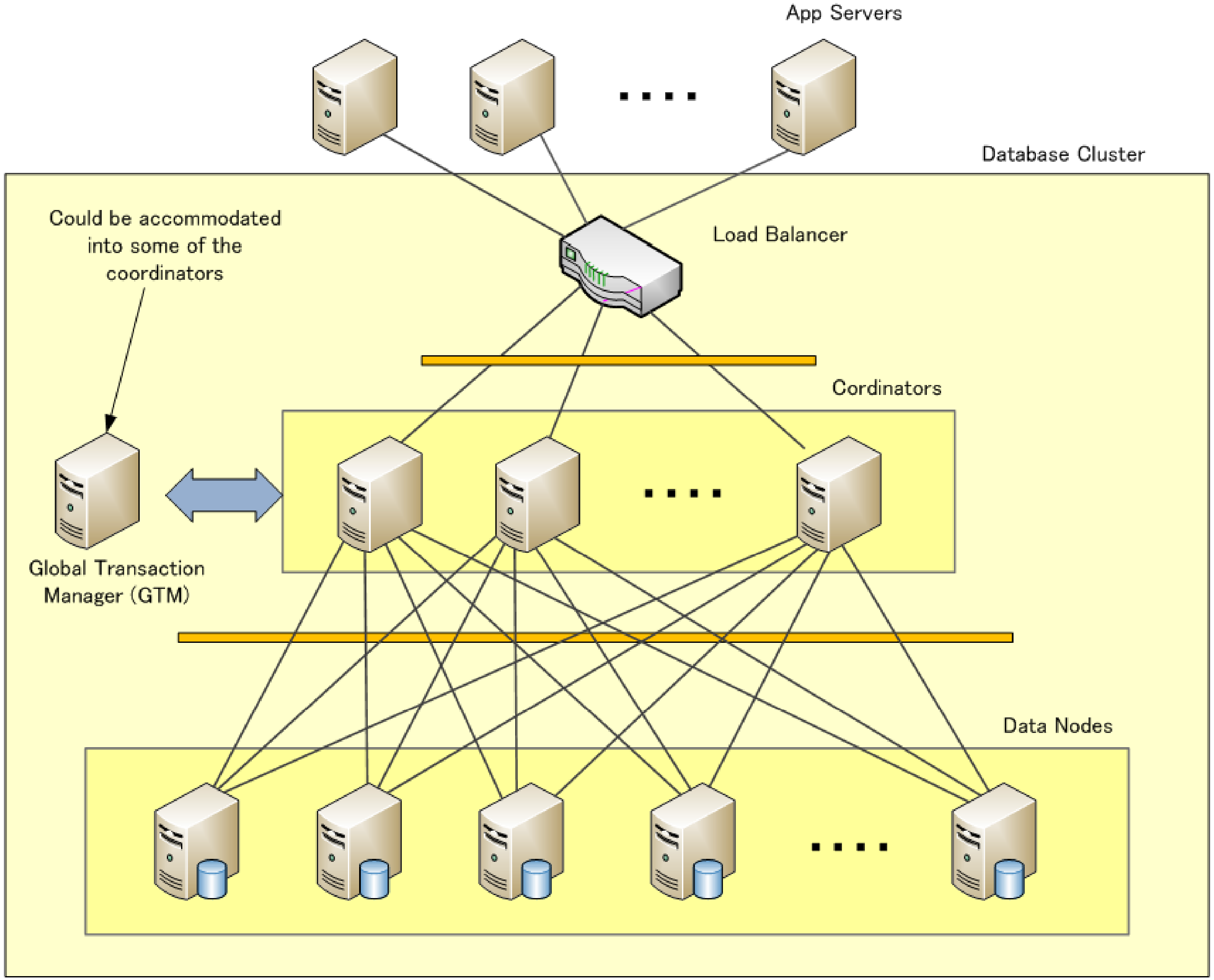
Checkpoint next starting point (GXID and Sequence)

GTM Standby

Standby can failover the master without referring to GTM master information.

Coordinator/Data Node Internals

Reference Architecture


App Servers
. . . .
Database Cluster
Could be accommodated into some of the coordinators
Load Balancer
Cordinators
. . . .
Global Transaction Manager (GTM)
Data Nodes
. . . .

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

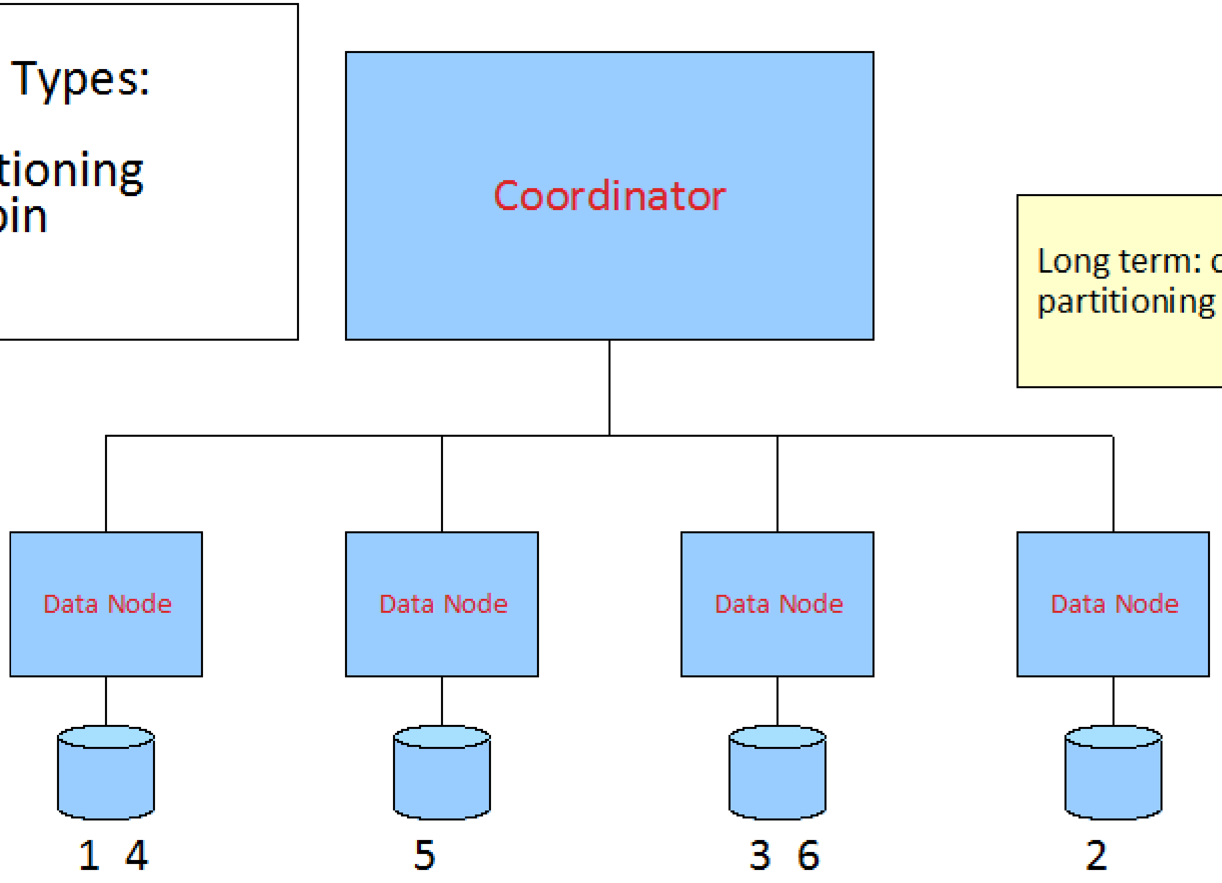
Postgres-XC Request Handling

- 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



Long term: custom, range partitioning

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)
- 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

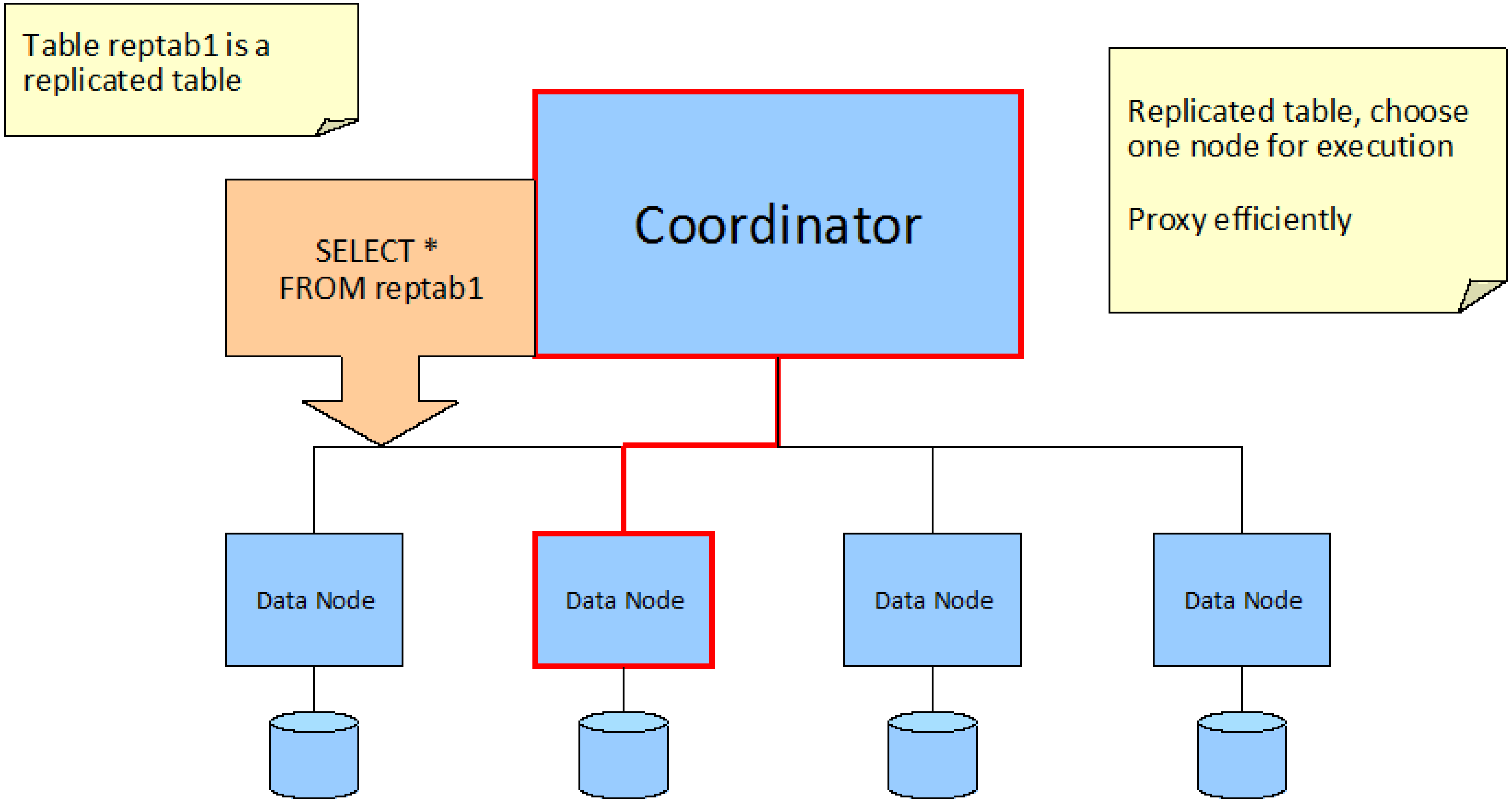


Table reptab1 is a replicated table

SELECT *
FROM reptab1

Replicated table, choose one node for execution

Proxy efficiently

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

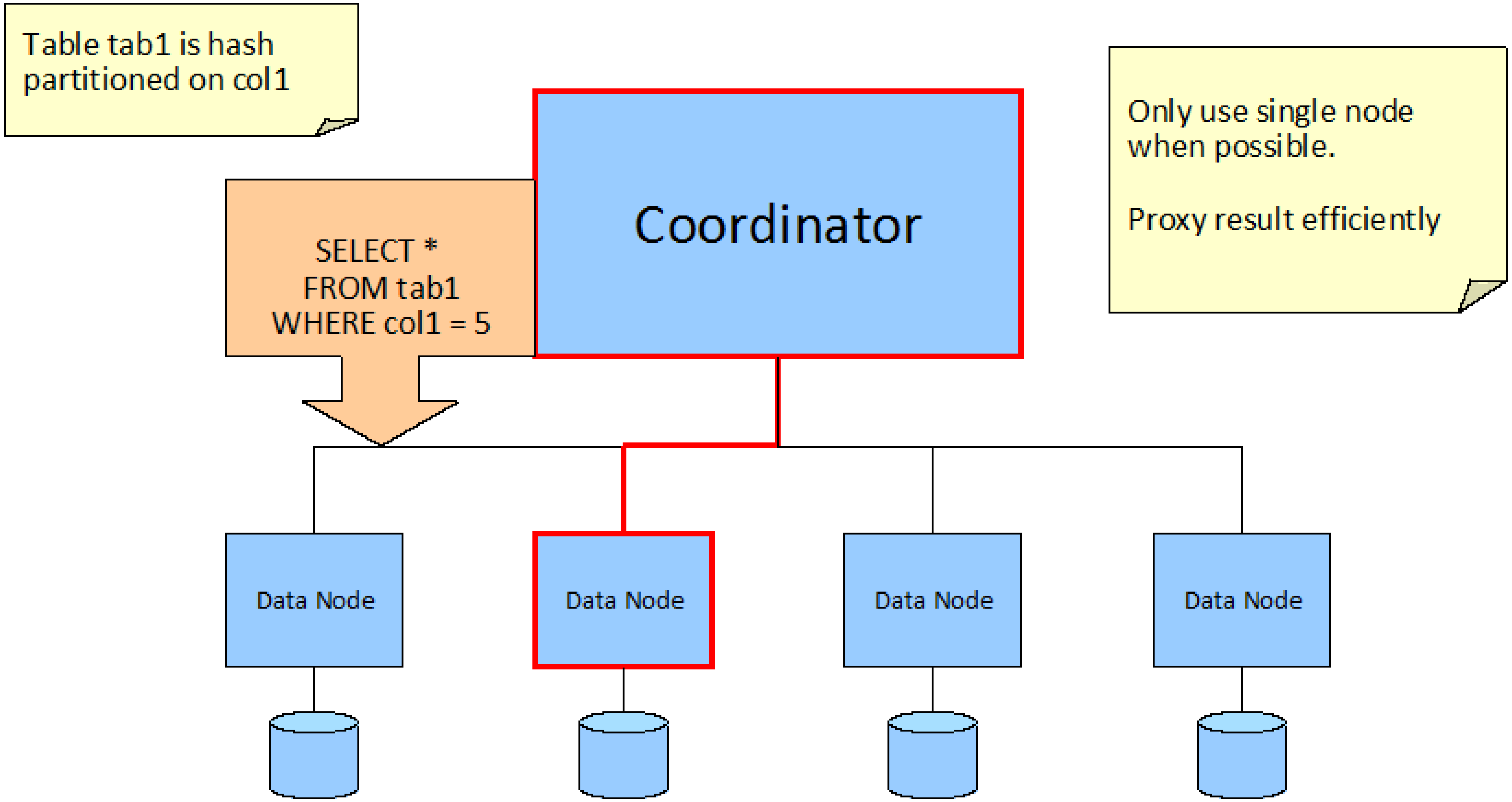


Table tab1 is hash partitioned on col1

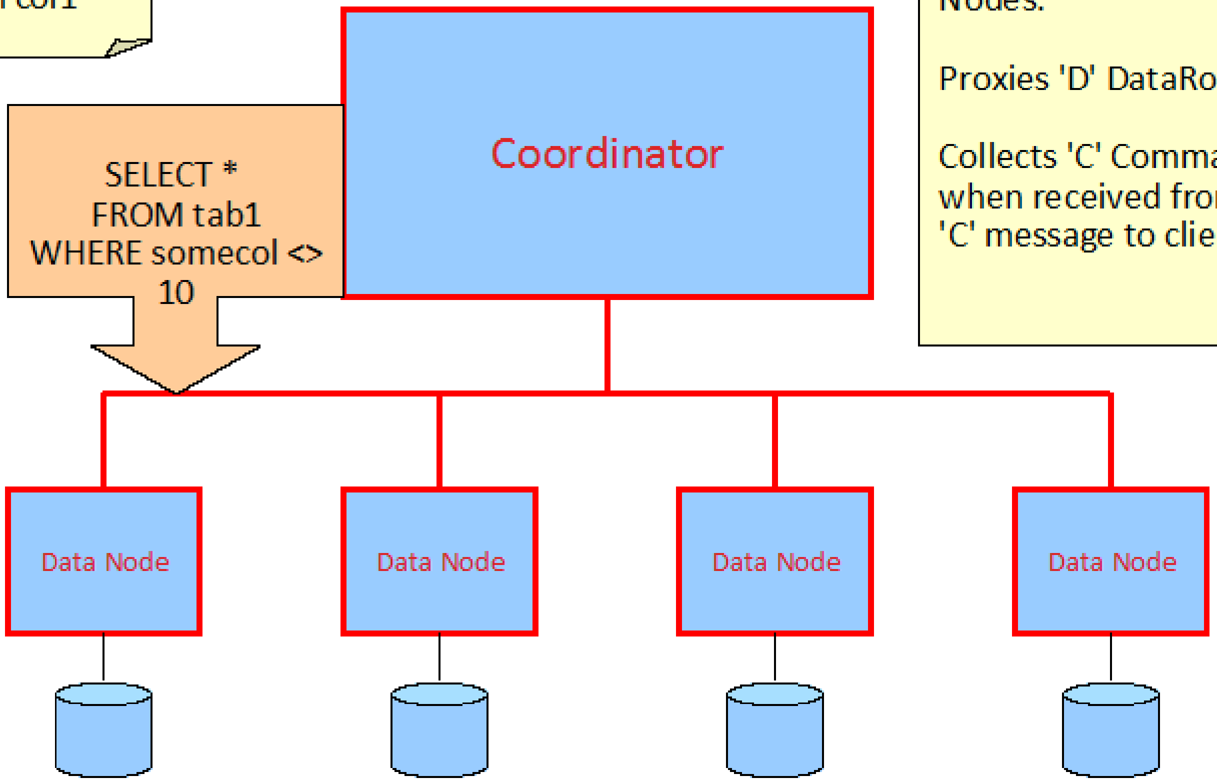
SELECT *
FROM tab1
WHERE col1 = 5

Only use single node when possible.

Proxy result efficiently

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
  - 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

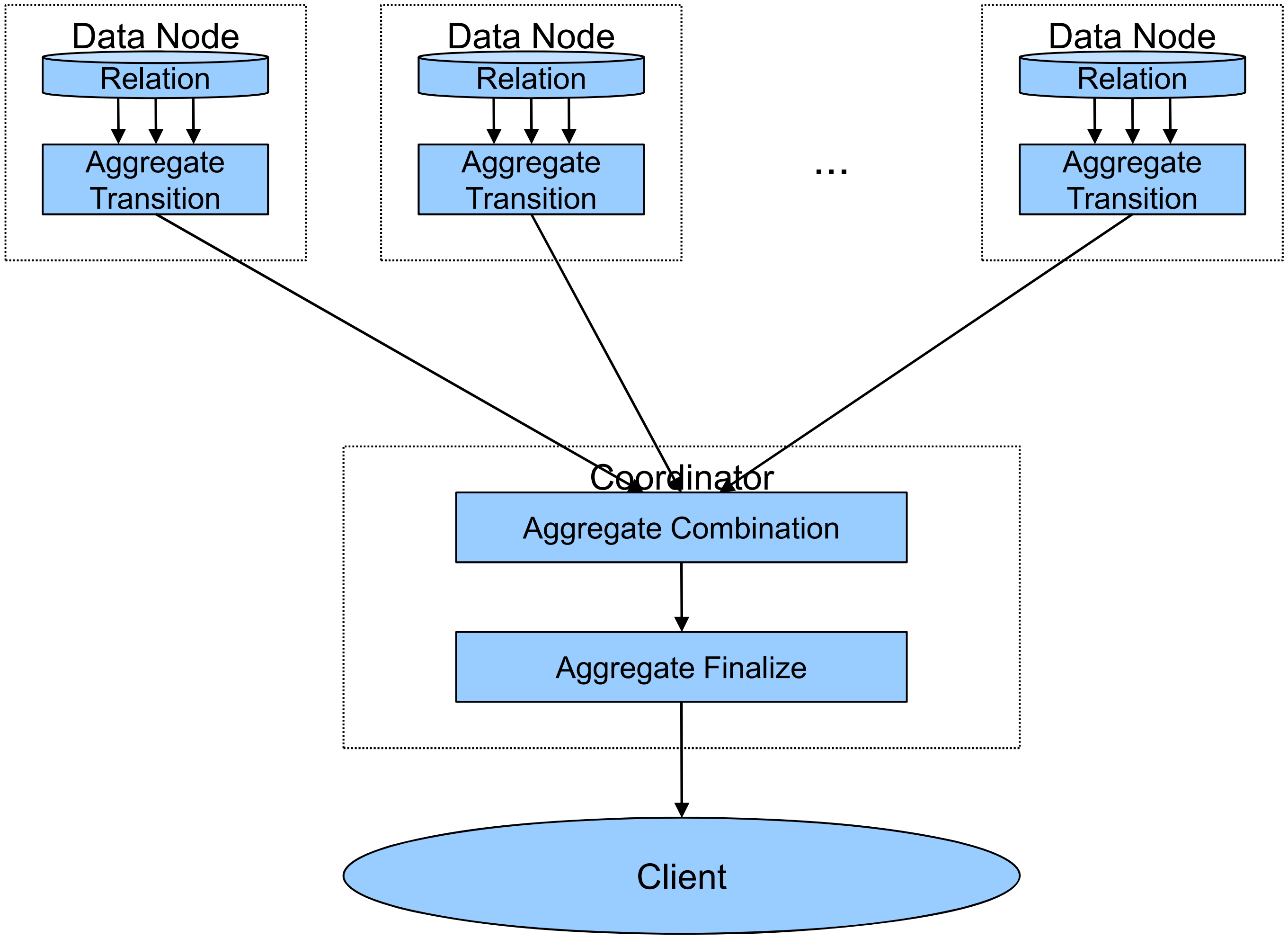
- Distributed transactions and two phase commit (2PC)
- Distributed Multi-Version Concurrency Control
  - Global Snapshots
  - Autovacuum
    - exclude XID in global snapshots
  - ANALZYE
  - Future optimization
  - CLOG
    - Careful when extending, not all transactions are on all nodes

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;
```

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

- Finalizer

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

Looking at Code

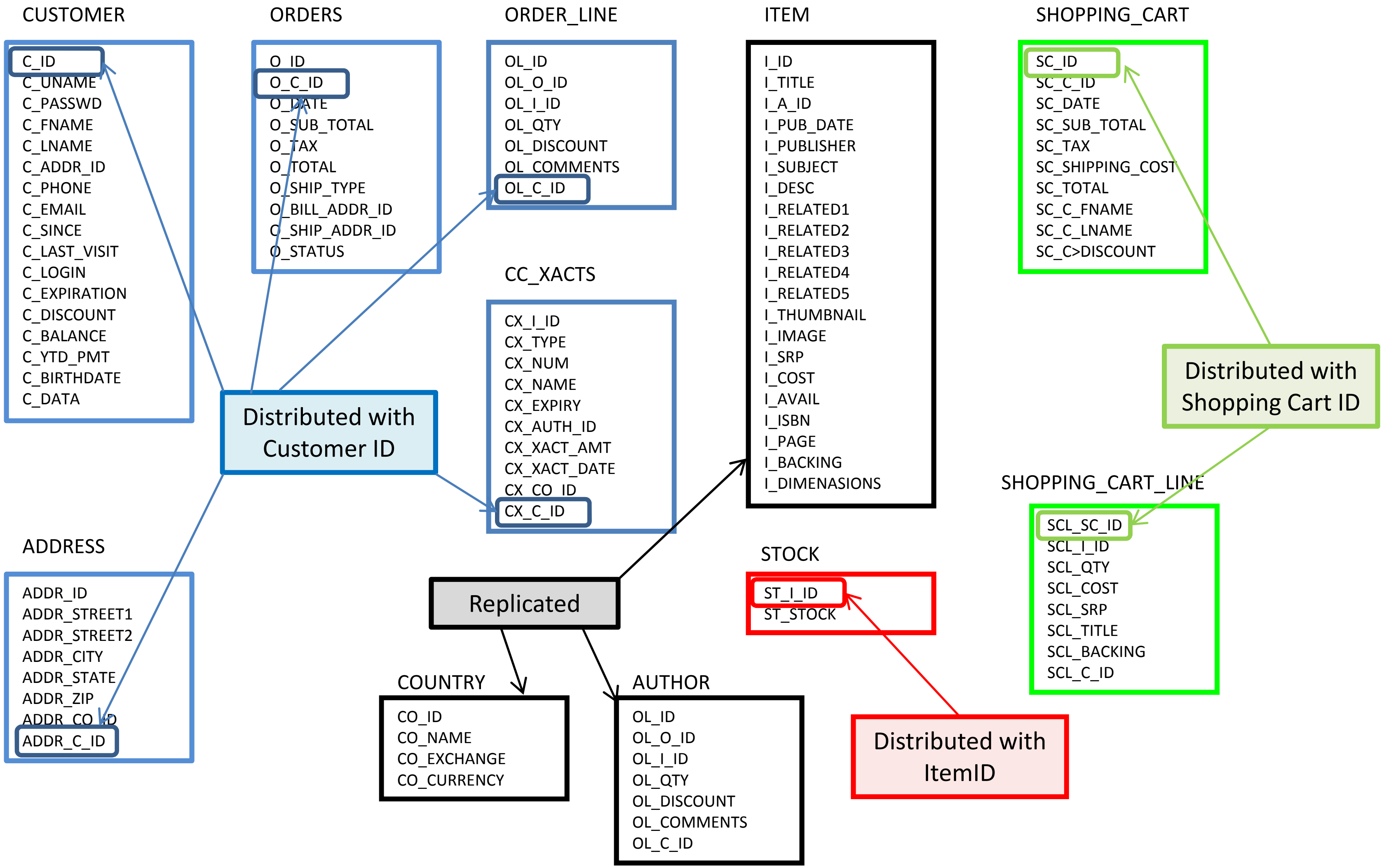
- Not (yet) overly invasive in PostgreSQL code
  - 8.4.2 → 8.4.3 merged cleanly
- Existing modules use `#ifdef PGXC` to identify Postgres-XC changes
- IS_PGXC_COORDINATOR and IS_PGXC_DATANODE easily identifies applicable code
- Advanced Coordinator logic in separate modules

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



CUSTOMER

C_ID
C_UNAME
C_PASSWD
C_FNAME
C_LNAME
C_ADDR_ID
C_PHONE
C_EMAIL
C_SINCE
C_LAST_VISIT
C_LOGIN
C_EXPIRATION
C_DISCOUNT
C_BALANCE
C_YTD_PMT
C_BIRTHDATE
C_DATA

ADDRESS

ADDR_ID
ADDR_STREET1
ADDR_STREET2
ADDR_CITY
ADDR_STATE
ADDR_ZIP
ADDR_CO_ID
ADDR_C_ID

ORDERS

O_ID
O_C_ID
O_DATE
O_SUB_TOTAL
O_TAX
O_TOTAL
O_SHIP_TYPE
O_BILL_ADDR_ID
O_SHIP_ADDR_ID
O_STATUS

Distributed with Customer ID

ORDER_LINE

OL_ID
OL_O_ID
OL_I_ID
OL_QTY
OL_DISCOUNT
OL_COMMENTS
OL_C_ID

CC_XACTS

CX_I_ID
CX_TYPE
CX_NUM
CX_NAME
CX_EXPIRY
CX_AUTH_ID
CX_XACT_AMT
CX_XACT_DATE
CX_CO_ID
CX_C_ID

Replicated

COUNTRY

CO_ID
CO_NAME
CO_EXCHANGE
CO_CURRENCY

AUTHOR

OL_ID
OL_O_ID
OL_I_ID
OL_QTY
OL_DISCOUNT
OL_COMMENTS
OL_C_ID

ITEM

I_ID
I_TITLE
I_A_ID
I_PUB_DATE
I_PUBLISHER
I_SUBJECT
I_DESC
I_RELATED1
I_RELATED2
I_RELATED3
I_RELATED4
I_RELATED5
I_THUMBNAIL
I_IMAGE
I_SRP
I_COST
I_AVAIL
I_ISBN
I_PAGE
I_BACKING
I_DIMENASIONS

STOCK

ST_I_ID
ST_STOCK

Distributed with ItemID

SHOPPING_CART

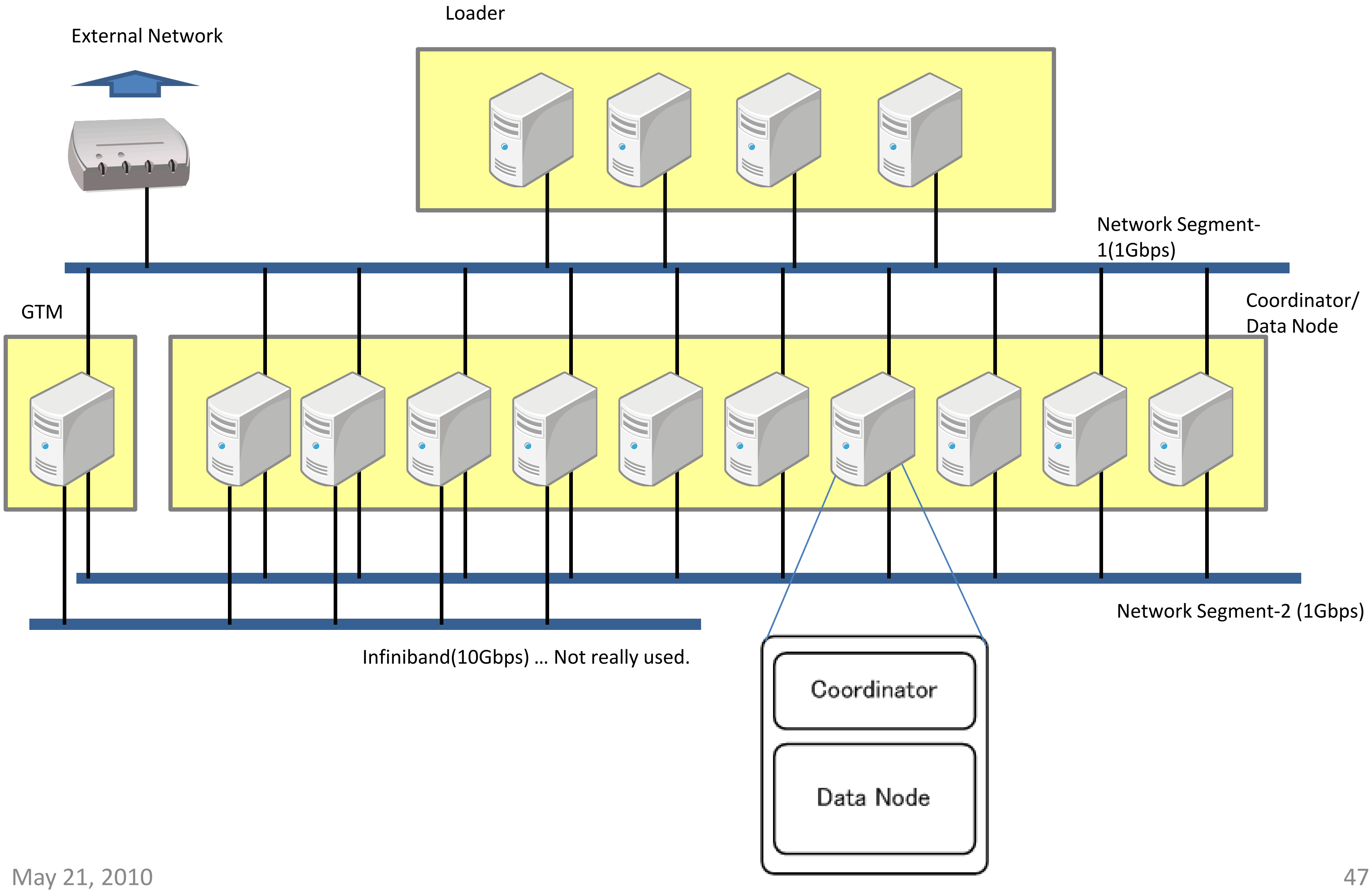
SC_ID
SC_C_ID
SC_DATE
SC_SUB_TOTAL
SC_TAX
SC_SHIPPING_COST
SC_TOTAL
SC_C_FNAME
SC_C_LNAME
SC_C>DISCOUNT

Distributed with Shopping Cart ID

SHOPPING_CART_LINE

SCL_SC_ID
SCL_I_ID
SCL_QTY
SCL_COST
SCL_SRP
SCL_TITLE
SCL_BACKING
SCL_C_ID

Evaluation Environment



Server Spec

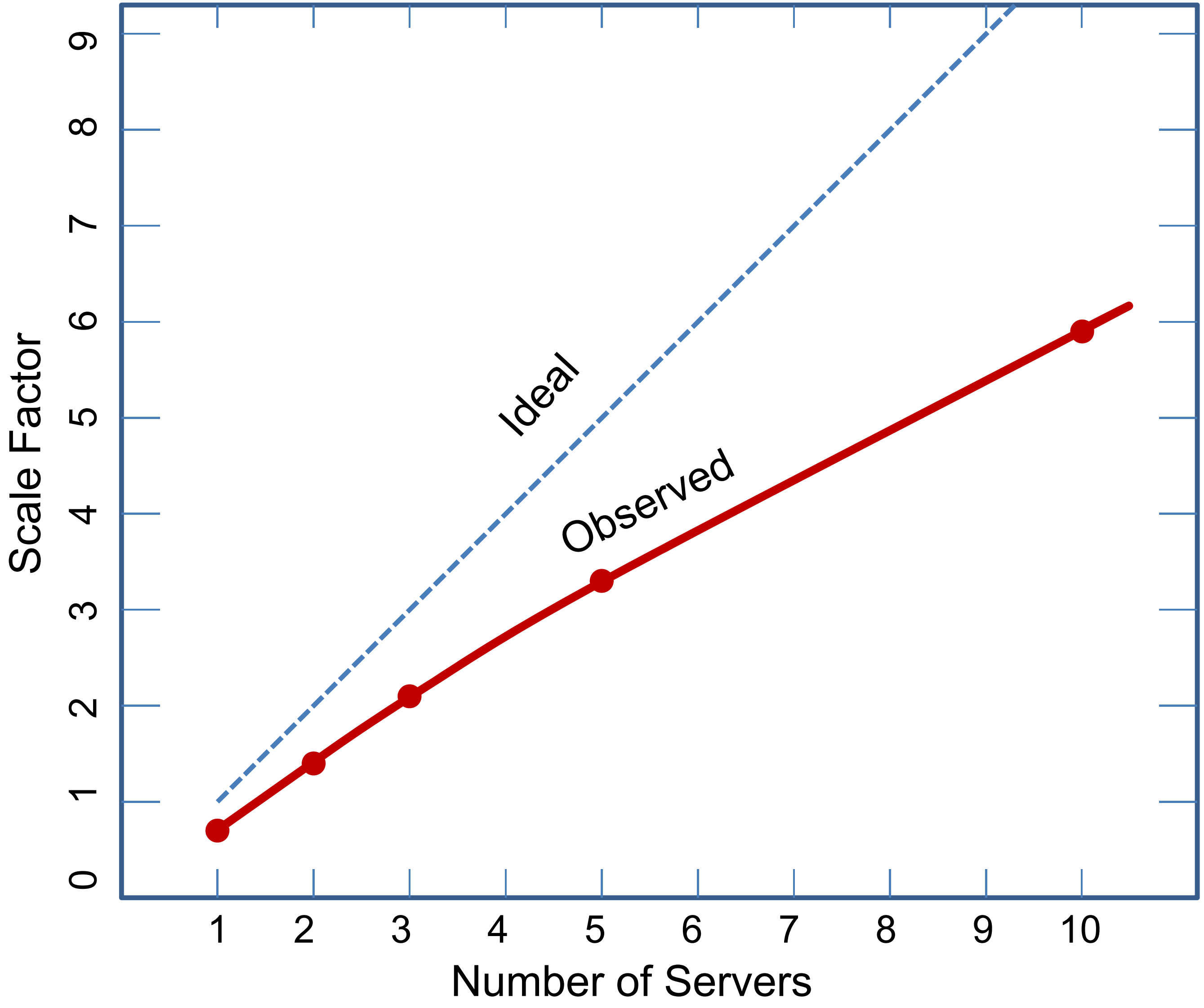
	Coordinator/Data Node	GTM/Loader
Make	HP Proliant DL360 G6	HP Proliant DL360 G5
CPU	Intel® Xeon® E5504 2.00GHz x 4	Intel® Xeon® X5460 3.16GHz x 4
Cache	4MB	6MB
MEM	12GB	6GB
HDD	146GB SAS 15krpm x 4 ea	146GP SAS 14krpm x 2 ea

Evaluation Summary

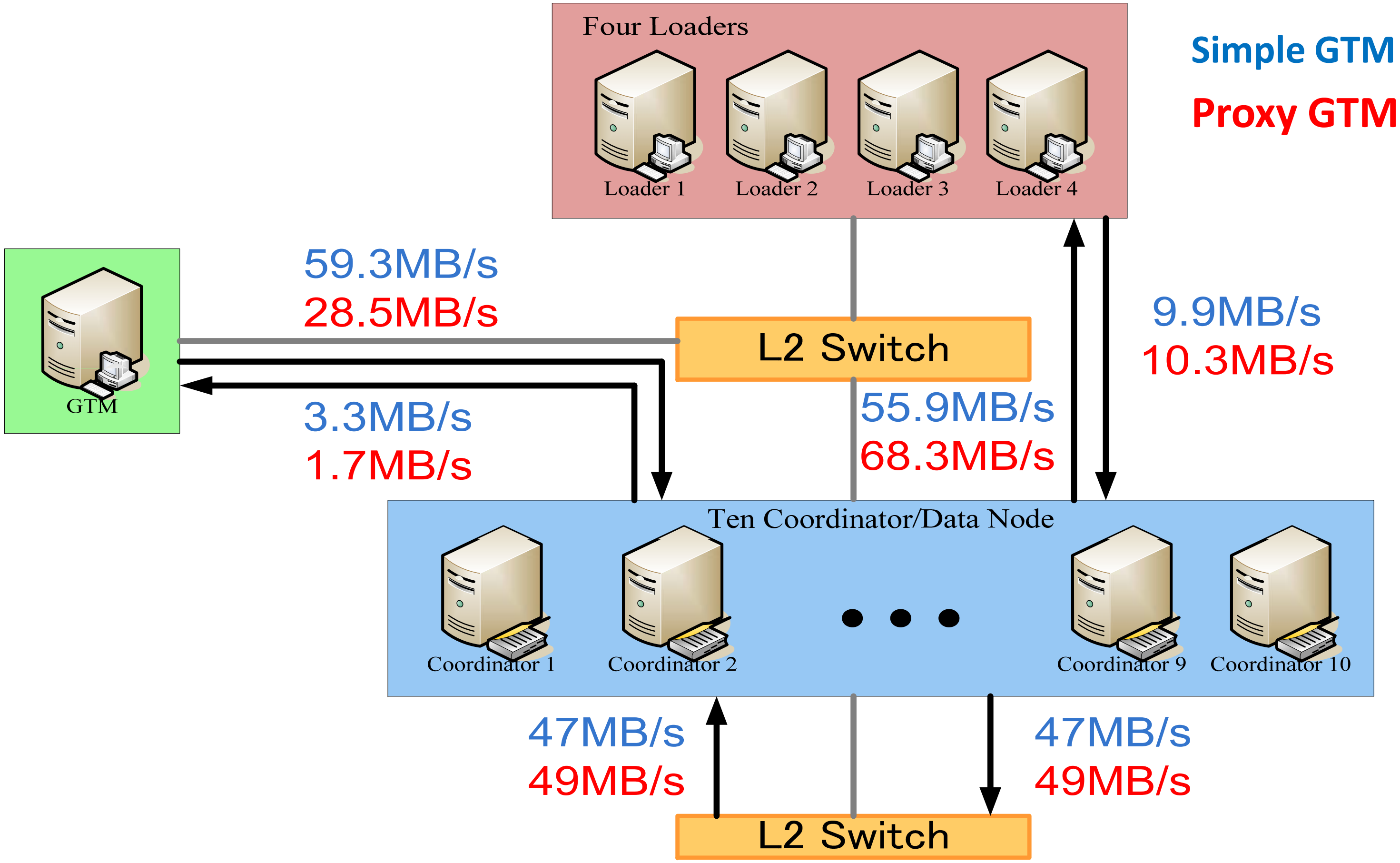
Full Load Throughput

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



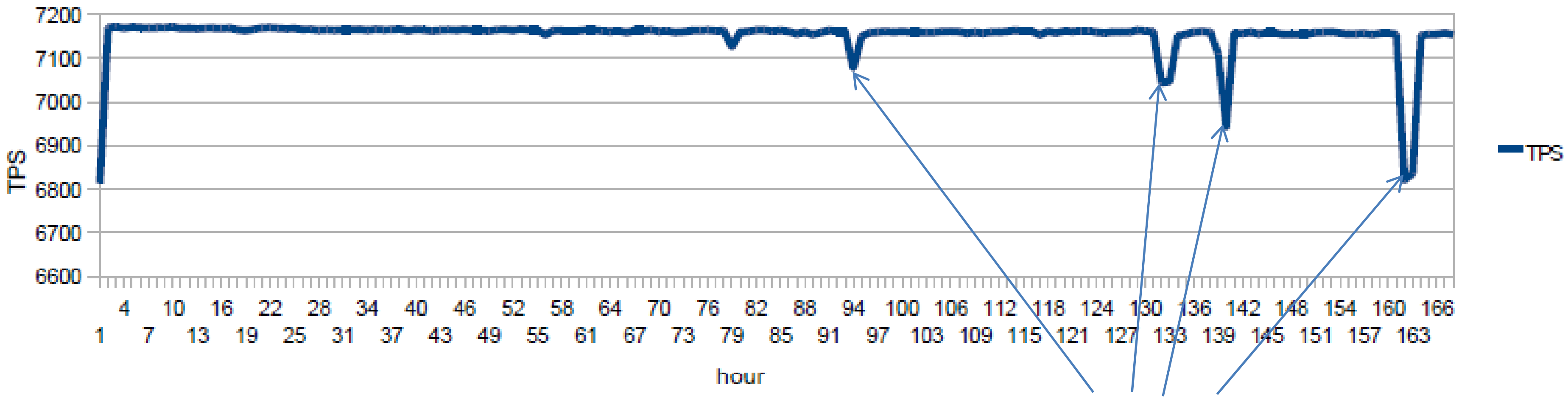
Network Workload



Simple GTM

Proxy GTM

One Week Test



Vaccum Analyze may become long transactions to affect the throughput.

Reasonably stable in a long run (90% workload)

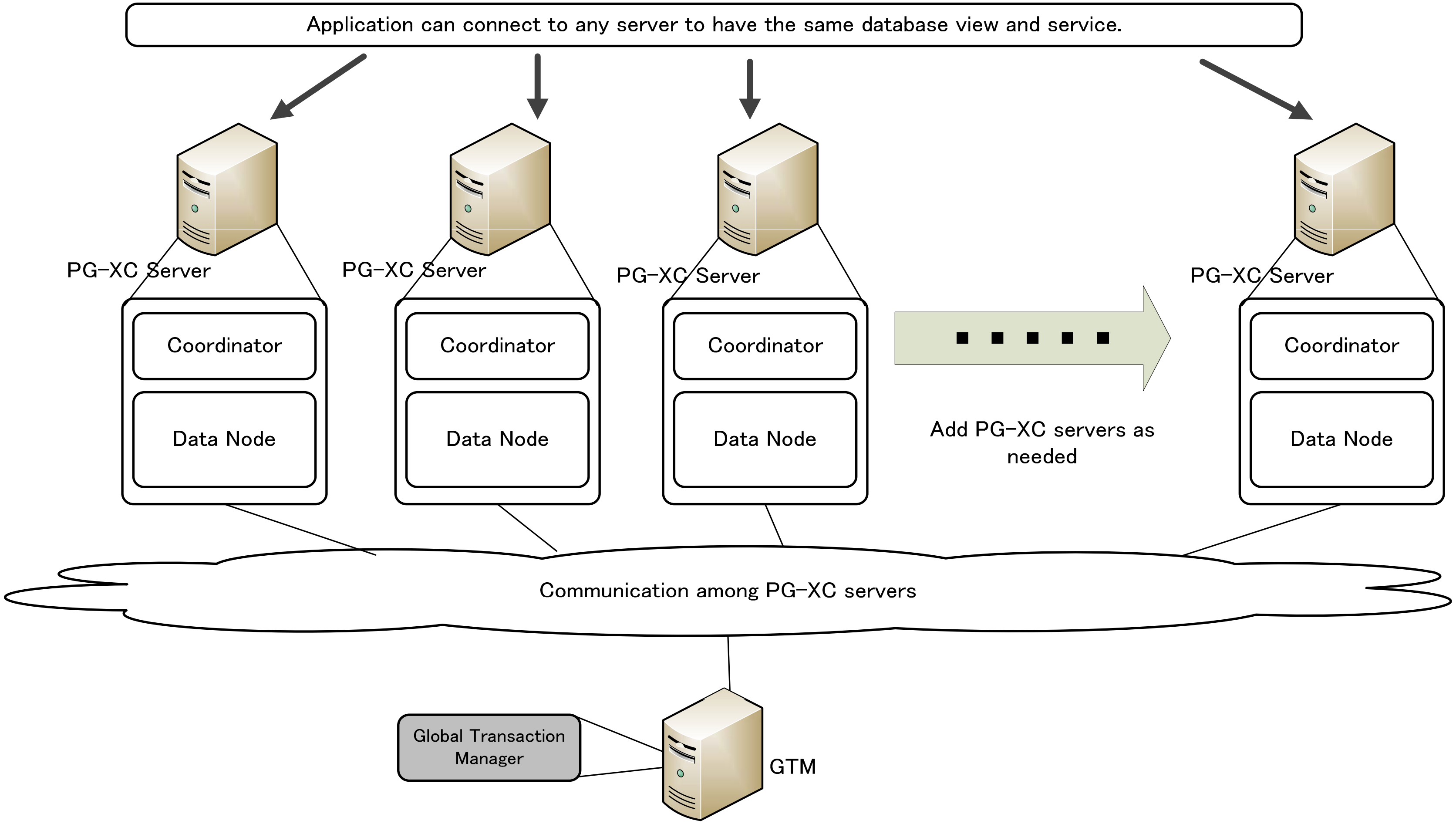
Avoiding Long Transactions

- 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

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

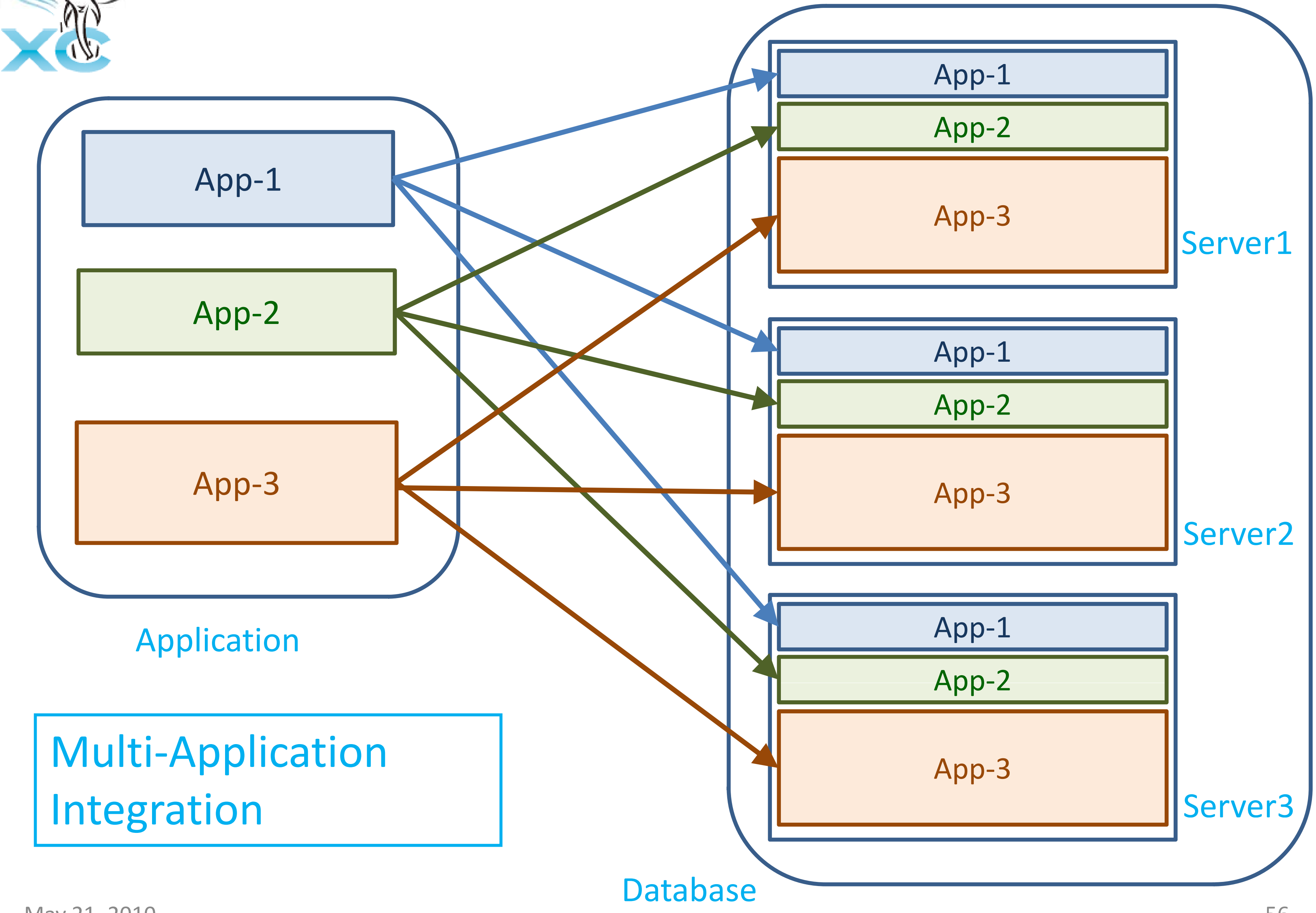
Possible Use Case (1)



Application can connect to any server to have the same database view and service.
PG-XC Server
Coordinator
Data Node
PG-XC Server
Coordinator
Data Node
PG-XC Server
Coordinator
Data Node
Add PG-XC servers as needed
PG-XC Server
Coordinator
Data Node
Communication among PG-XC servers
Global Transaction Manager
GTM

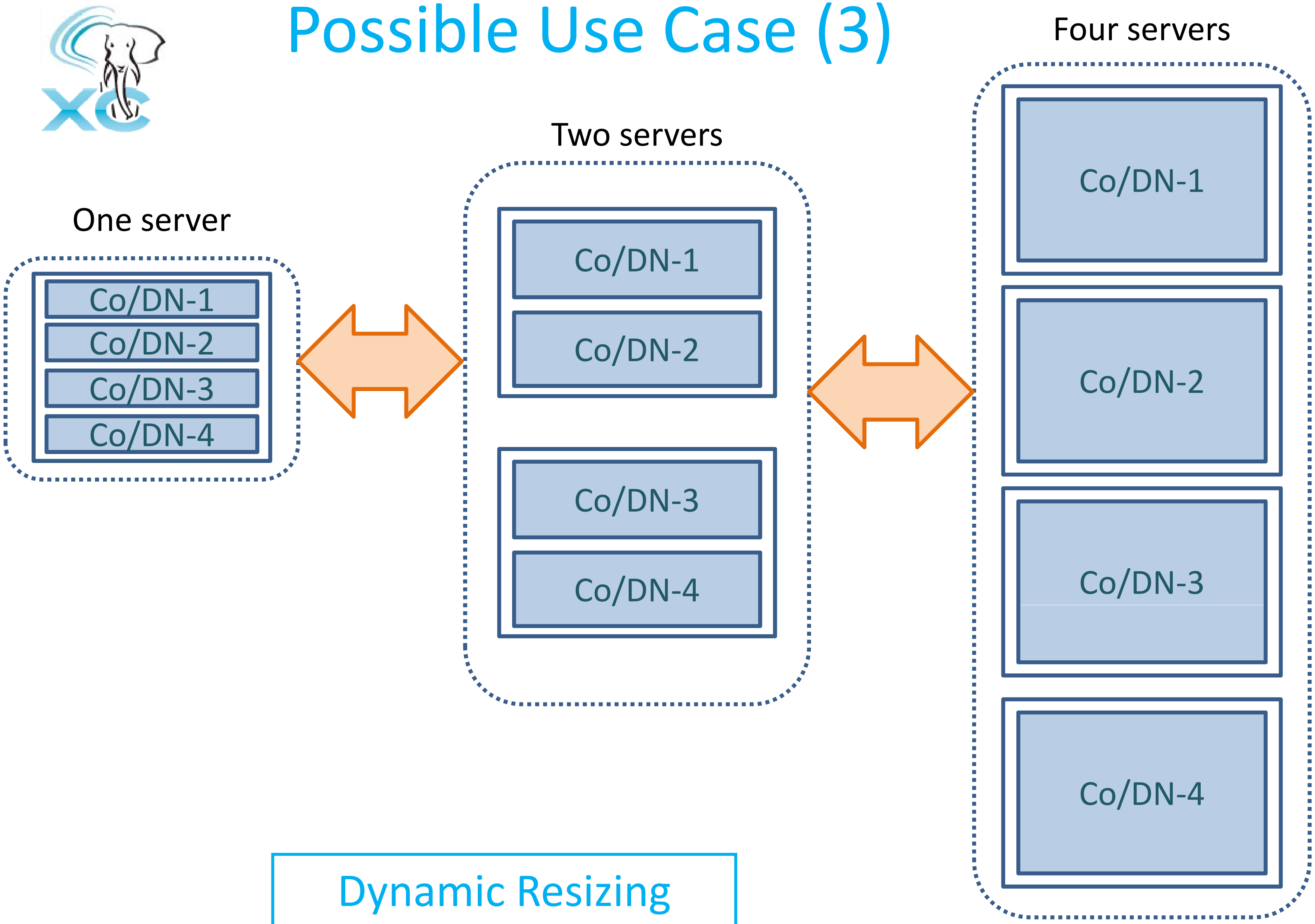
Large Scale Application

Possible Use Case (2)



Multi-Application Integration

Possible Use Case (3)



Dynamic Resizing
(Cloud)

WIP

- WIP up to V.1.0
  - ORDER BY/DISTINCT
  - Stored Functions
  - Subqueries
  - Views, Rules

Roadmap and Plan (1)

- Toward V.1.1 (Sept., 2010)
  - Cluster-Wide Installer
  - Cluster-Wide Operation Utilities
  - Regression Tests
  - Logical Backup/Restore
  - Basic Cross-Node Operation
  - TEMP Table
  - Extended Query Protocol
    - JDBC
  - Global Timestamp
  - Drivers
    - ECPG, JDBC, PHP, etc.
  - Forward Cursor (w/o ORDER BY)

Roadmap and Plan (2)

- Beyond V.1.1
  - Physical Backup/Restore
    - PITR
  - Cross-node operation optimization
    - Tuple transfer Infrastructure from node to node
  - More variety of statements
    - INSRET … FROM SELECT
  - Prepared Statement
  - General Aggregate Functions
  - General Functions
  - Savepoint
  - Session Parameters
  - 2PC from Apps
  - Forward Cursor with ORDER BY
  - Backward Cursor
  - Batch, Statement pushdown
  - Catalog Synchronize with DDL
  - Trigger
  - Global constraints
  - Tuple relocation
    - Distribute key update

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

POSTGRES XC

Thank You Very Much