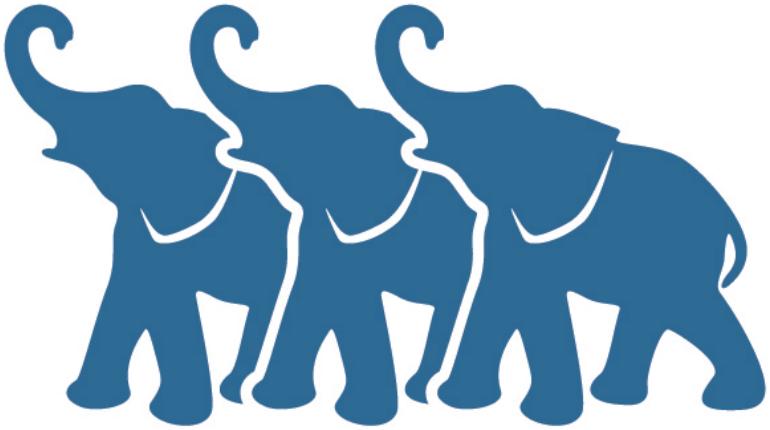


Scaling PostgreSQL on Postgres-XL

February, 2015

Mason Sharp

PGConf.ru

Agenda

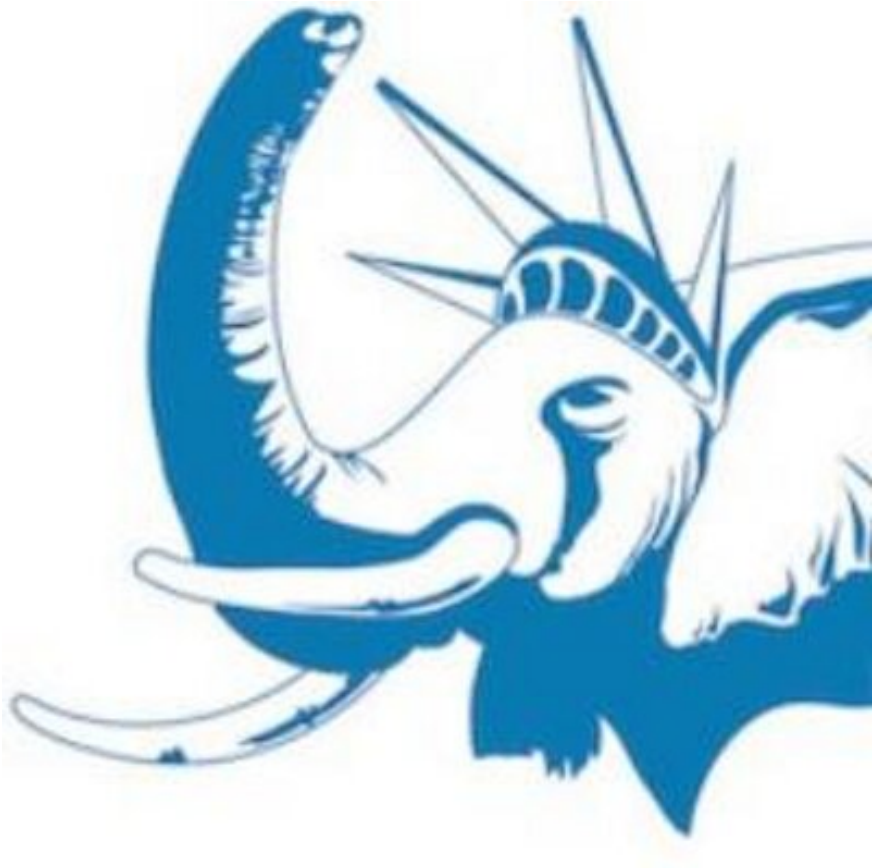
- Intro
- Postgres-XL Background
- Architecture Overview
- Distributing Tables
- Configuring a Local Cluster
- Differences to PostgreSQL
- Community

whoami

- Co-organizer of NYC PUG
- I like database clusters
  - GridSQL
  - Postgres-XC
  - Postgres-XL
- Work:
  - TransLattice
  - StormDB
  - EnterpriseDB

PGConf.US

- March 25–27, 2015
- New York City
- 44 Talks
- 4 Tracks!



Postgres-XL

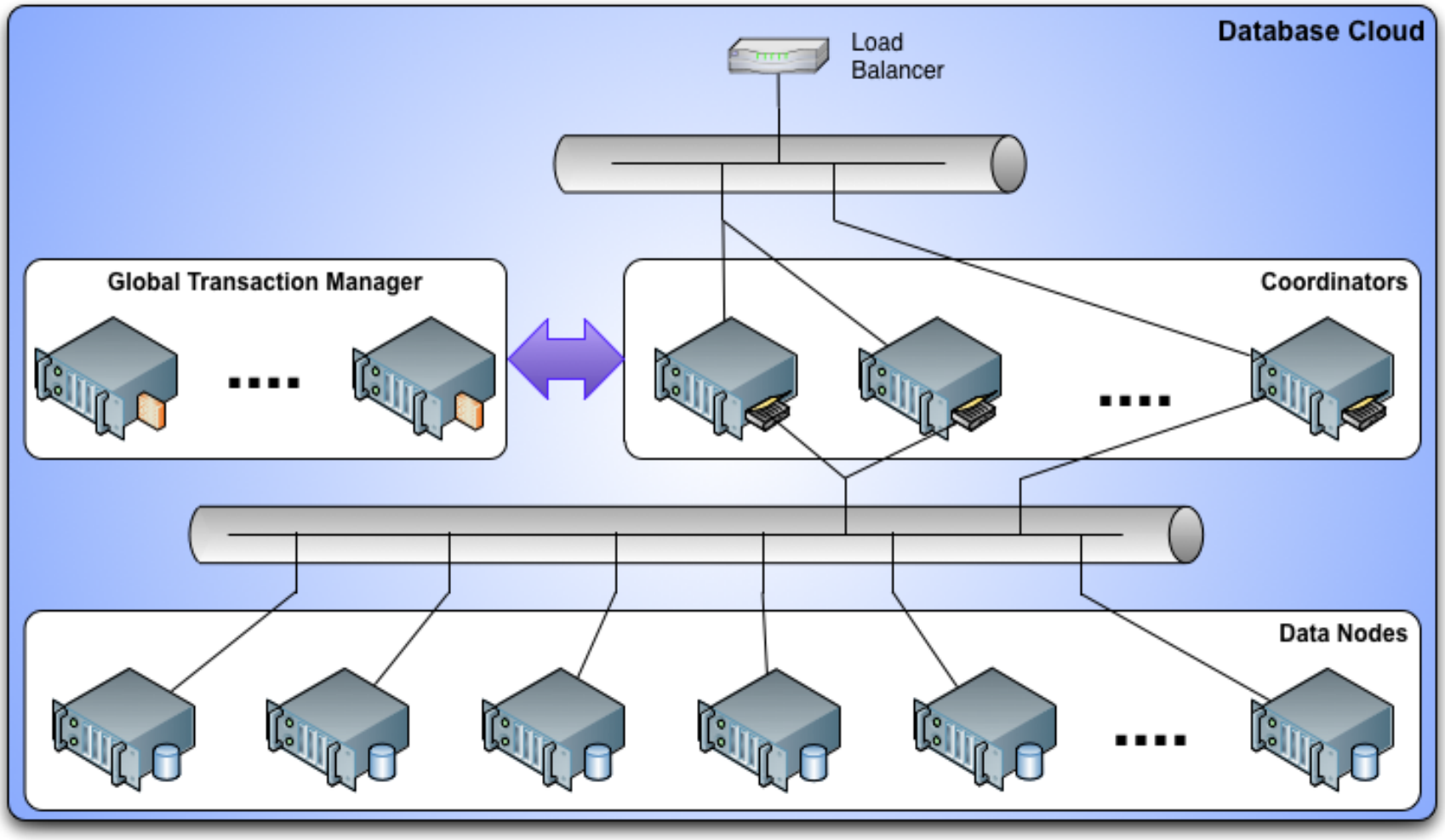
- Open Source
- Scale-out RDBMS
  - MPP for OLAP
  - OLTP write and read scalability
  - Custer-wide ACID properties
  - Multi-tenant security
  - Can also act as distributed key-value store

Postgres-XL

- Hybrid Transactional and Analytical Processing (HTAP)
- Can replace multiple systems

Postgres-XL

- Rich SQL support
  - Correlated Subqueries


Database Cloud
Load
Balancer
Global Transaction Manager
Coordinators
Data Nodes

Postgres-XL Features

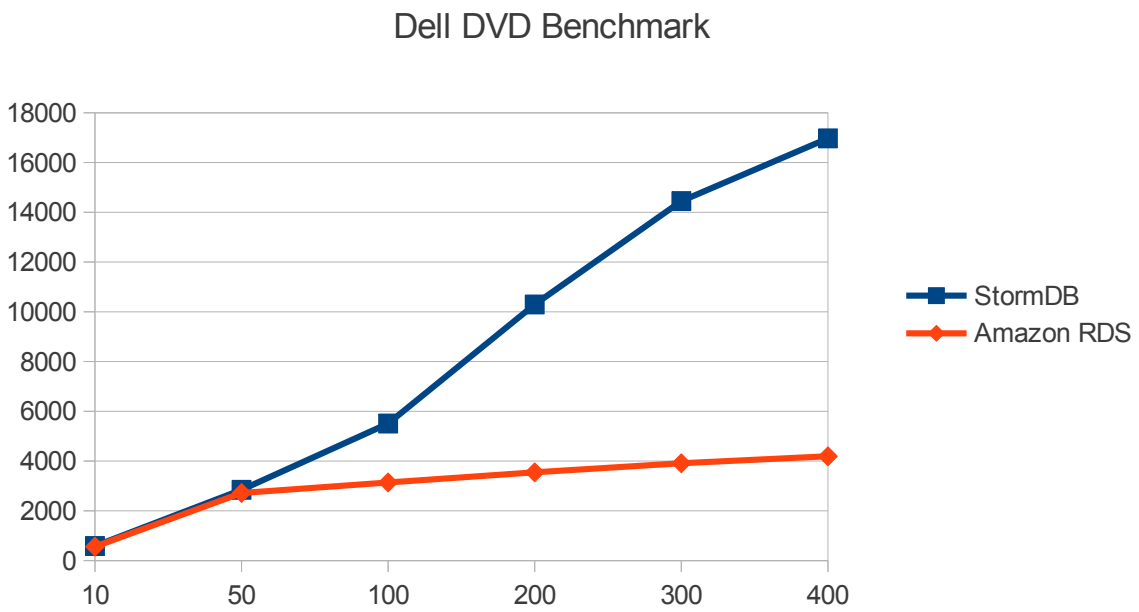
- Large subset of PostgreSQL SQL
- Distributed Multi-version Concurrency Control
- Contrib-friendly
- Full Text Search
- JSON and XML Support
- Distributed Key-Value Store

Postgres-XL Missing Features

- Built-in High Availability
  - Use external like Corosync/Pacemaker
  - Area for future improvement
- “Easy” to re-shard / add nodes
  - Some pieces there
  - Tables locked during redistribution
  - Can however “pre-shard” multiple nodes on the same server or VM
- Some FK and UNIQUE constraints
- Recursive CTEs (WITH)

Performance

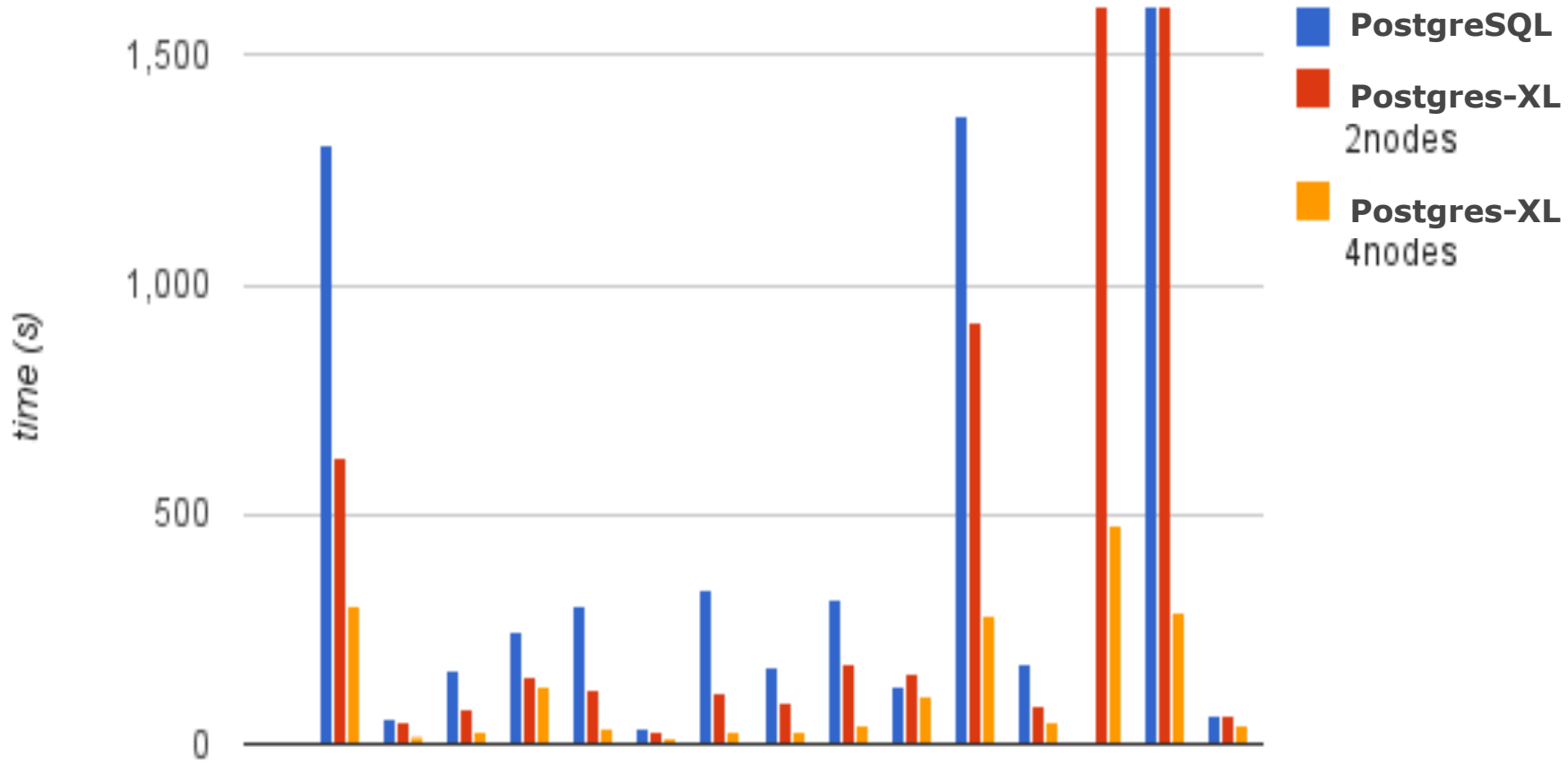
Transactional work loads



OLTP Characteristics

- Coordinator Layer adds about 30% overhead to DBT-1 (TPC-W) workload
  - Single node (4 cores) gets 70% performance compared to vanilla PostgreSQL
- If linear scaling 10 nodes should achieve 7x performance compared to PostgreSQL
- Actual result: 6 – 6.4 x
- More nodes, more open transactions, means larger snapshots and more visibility checking

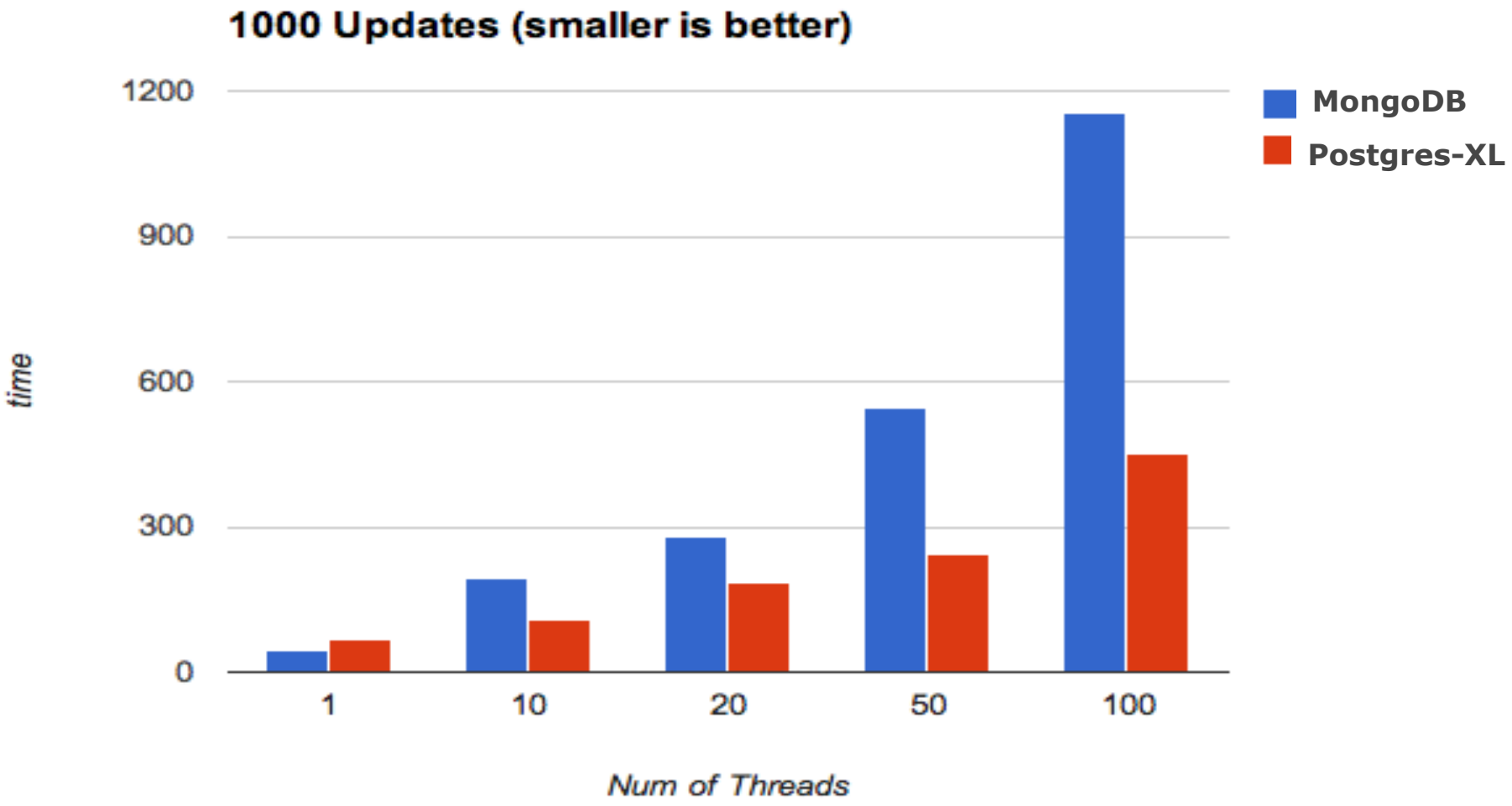
MPP Performance – DBT-3 (TPC-H)



OLAP Characteristics

- Depends on schema
- Star schema?
  - Distribute fact tables across nodes
  - Replicate dimensions
- Linear/near-linear query performance for straight-forward queries
- Other schemes different results
  - Sometimes “better than linear” due to more caching

Key-value Store Comparison

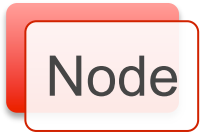


Multiversion Concurrency Control (MVCC)

MVCC

- Readers do not block writers
- Writers do not block readers
- Transaction Ids (XIDs)
  - Every transaction gets an ID
- Snapshots contain a list of running XIDs

MVCC – Regular PostgreSQL



Example:

T1 Begin...

T2 Begin; INSERT...; Commit;

T3 Begin...

T4 Begin; SELECT

- T4's snapshot contains T1 and T3
  - T2 already committed
  - T4 can see T2's commits, but not T1's nor T3's

MVCC – Multiple Node Issues

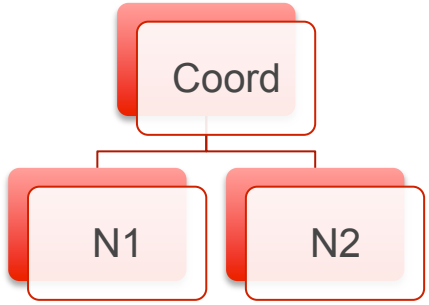
Example:

T1 Begin... **N1,N2**

T2 Begin; INSERT...; Commit;

T3 Begin... **N2,N1**

T4 Begin; SELECT



- Node 1: T2 Commit, T4 SELECT
- Node 2: T4 SELECT, T2 Commit
- T4's SELECT statement returns inconsistent data
  - Includes data from Node1, but not Node2.

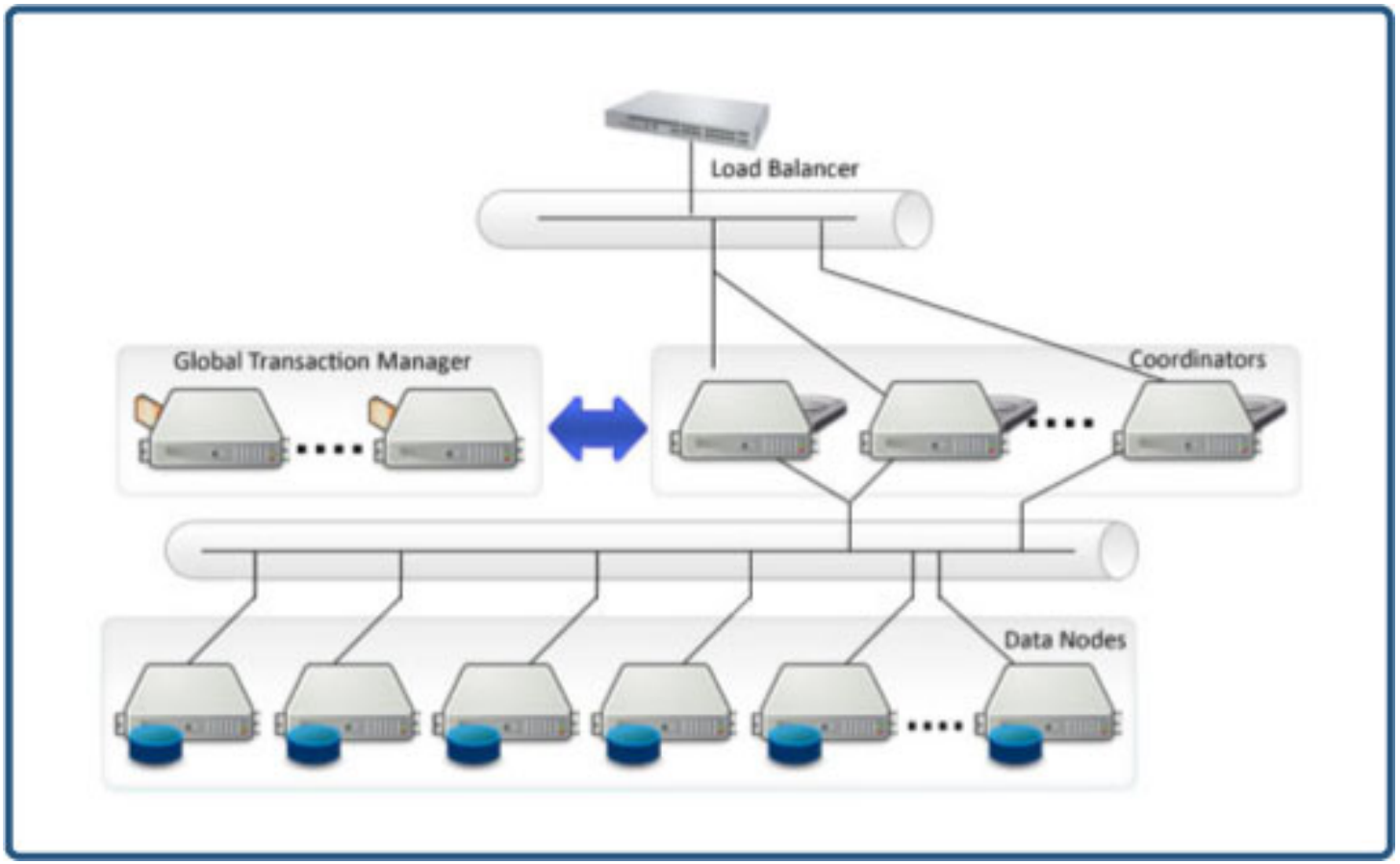
MVCC – Multiple Node Issues

Postgres-XL Solution:

- Make XIDs and Snapshots cluster-wide

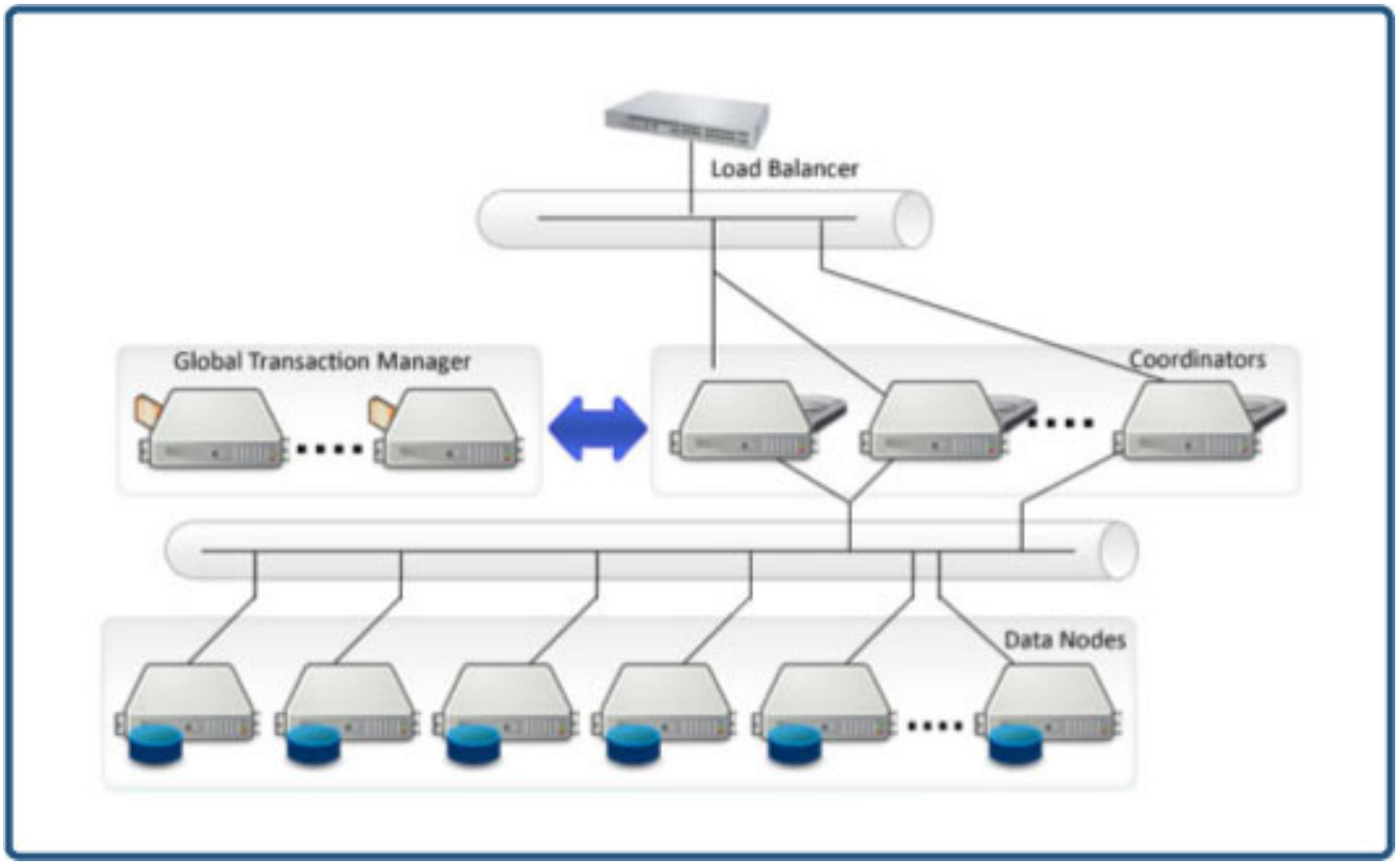
Postgres-XL Technology

- Users connect to any Coordinator



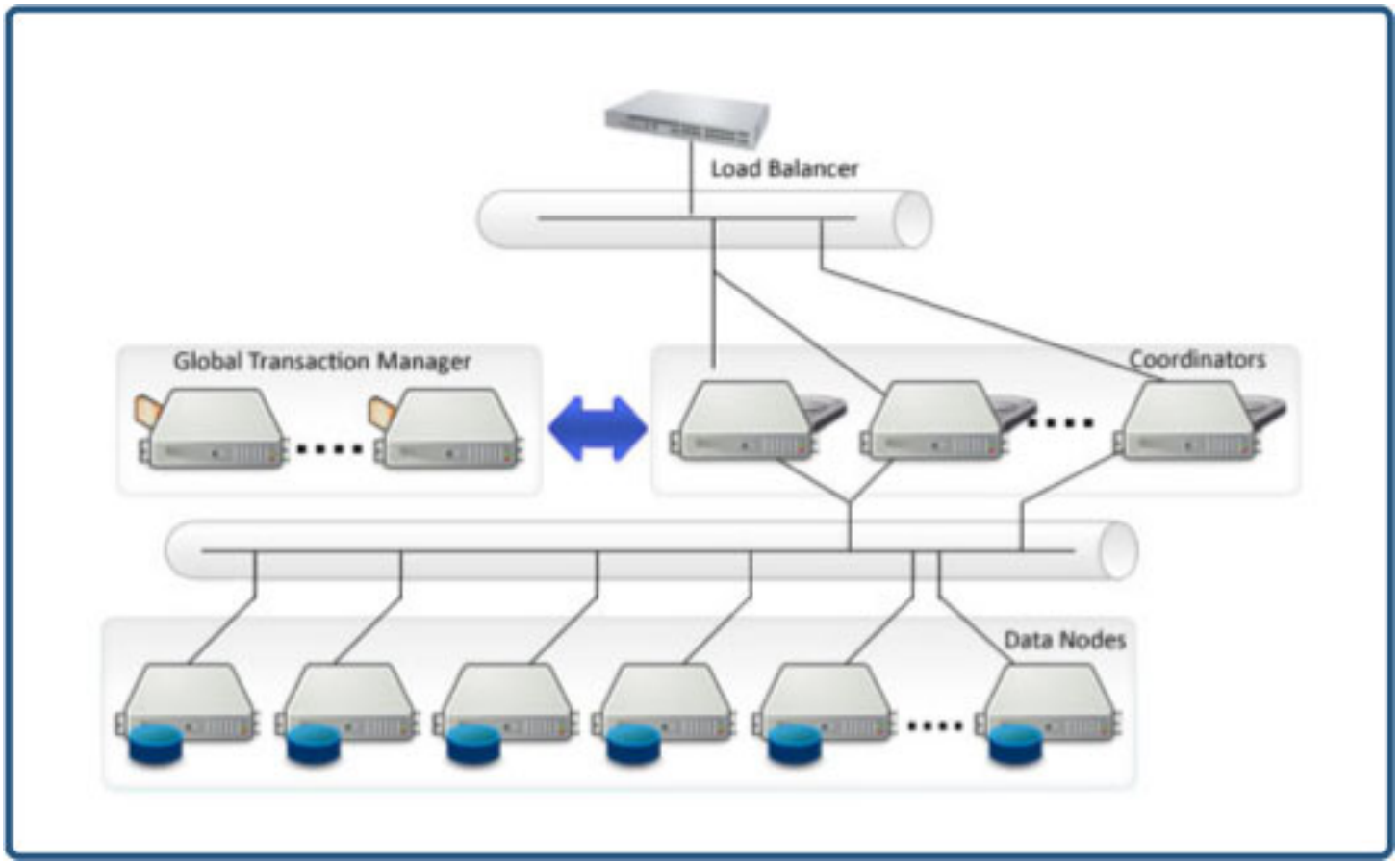
Postgres-XL Technology

- Data is automatically spread across cluster



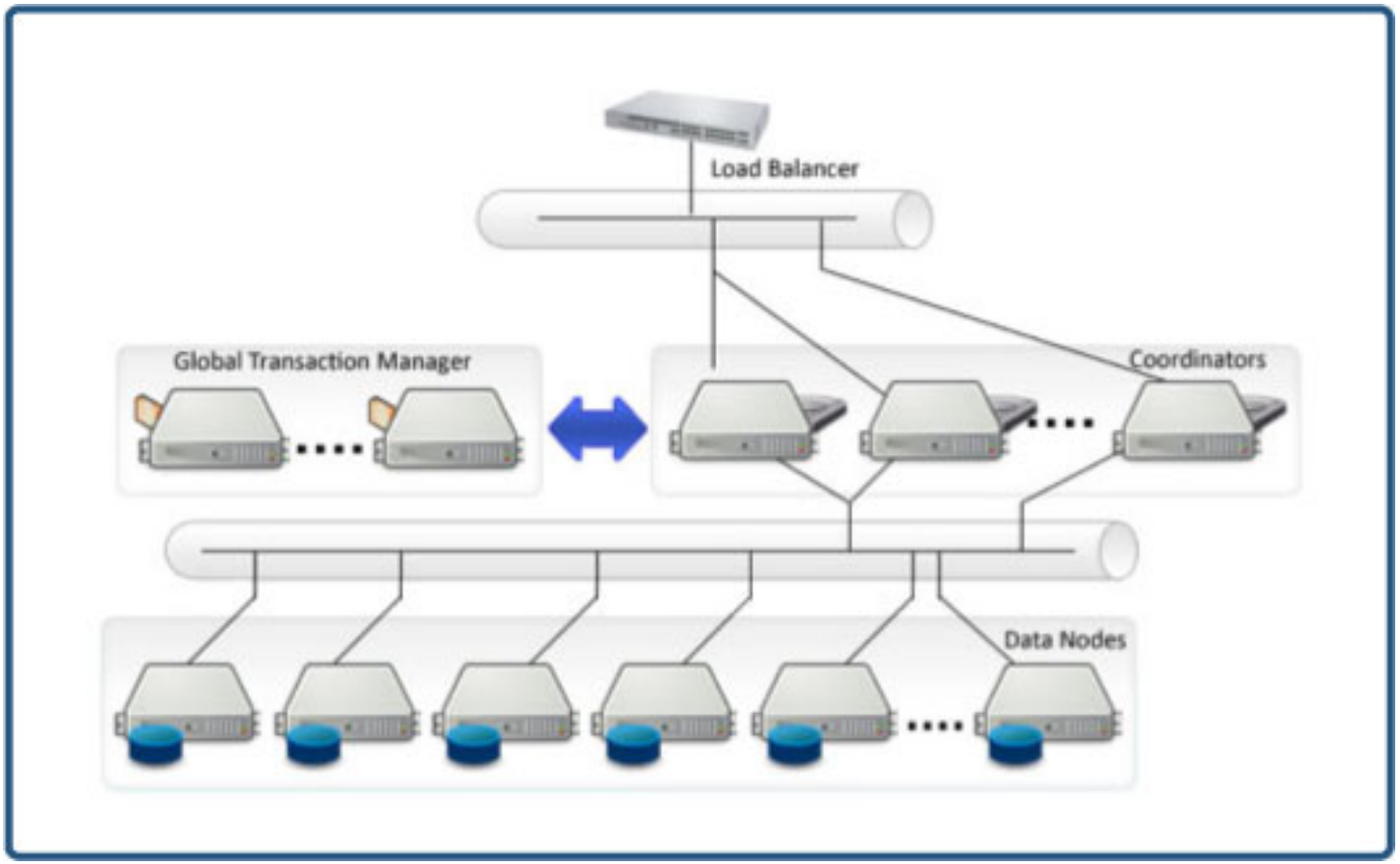
Postgres-XL Technology

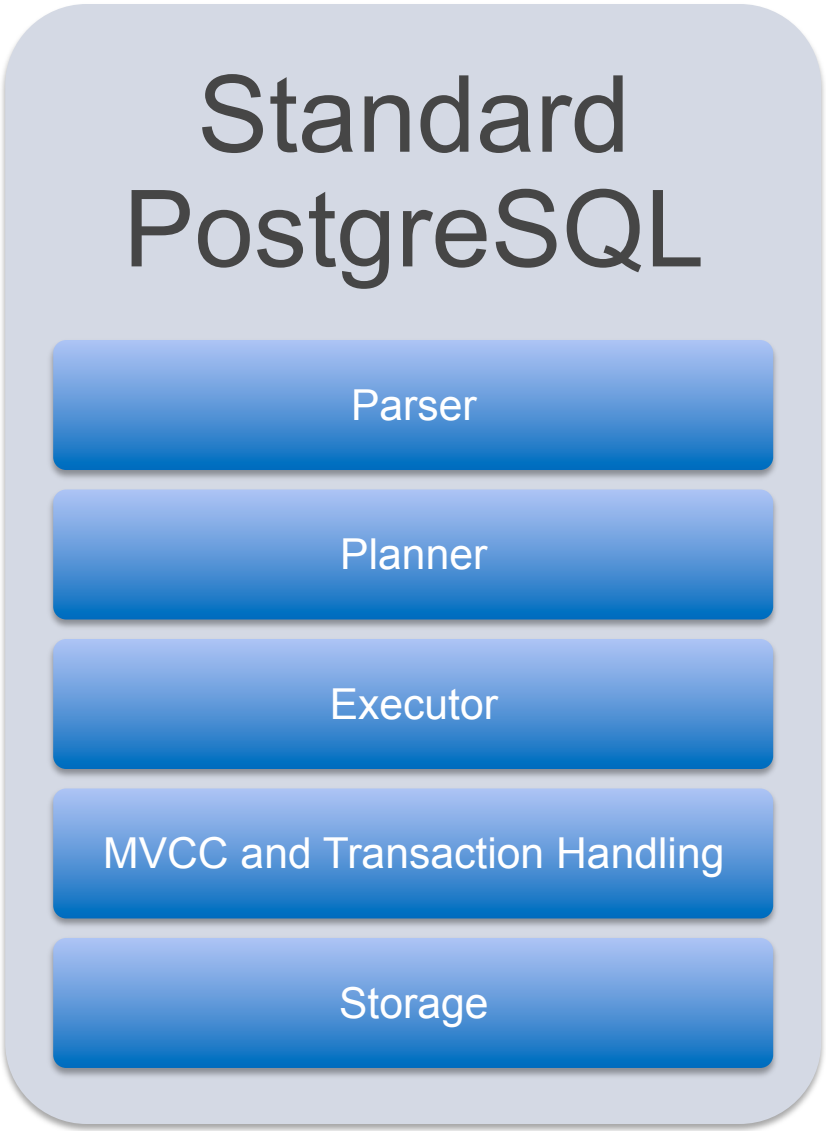
- Queries return back data from all nodes in parallel

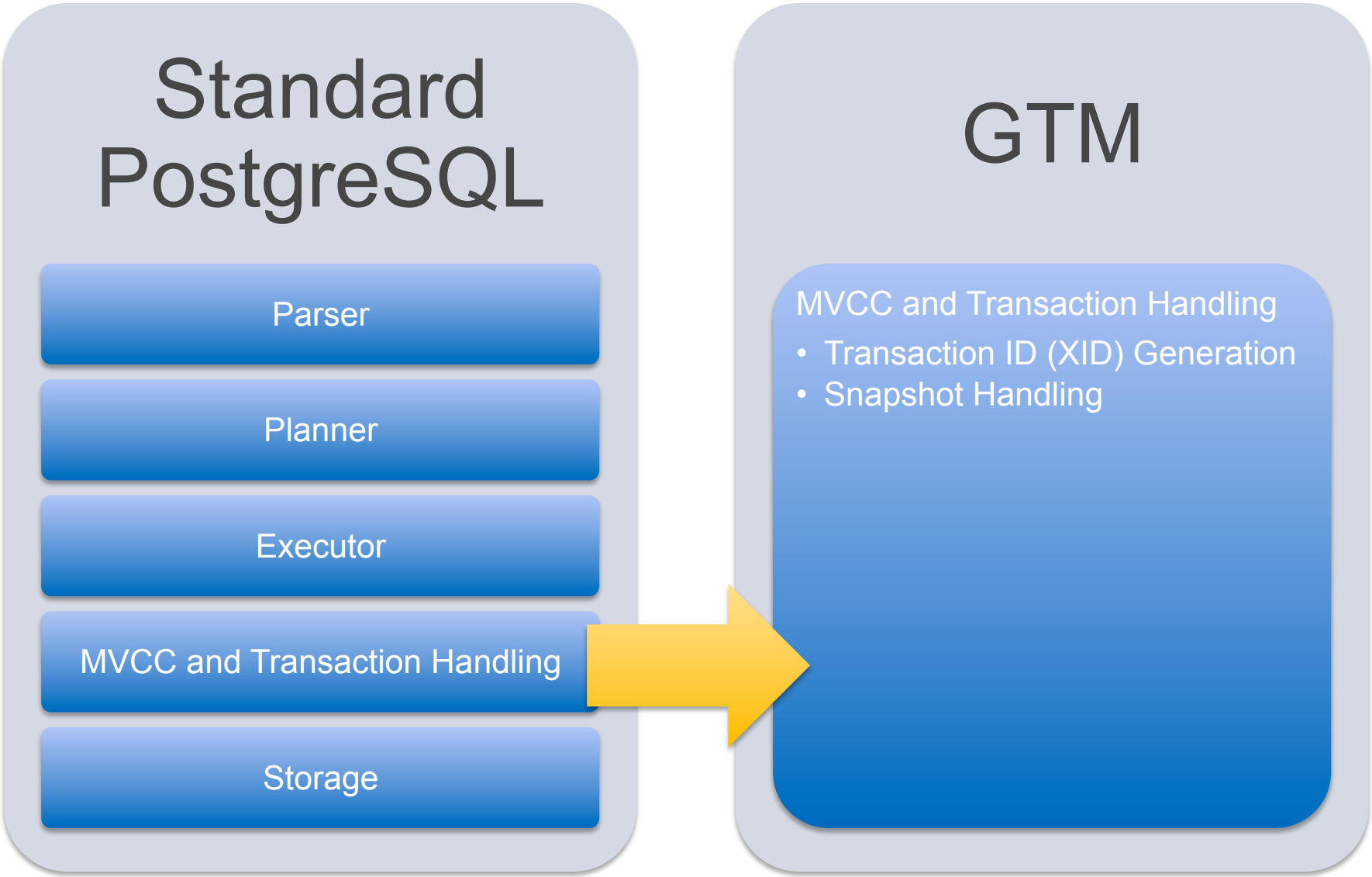


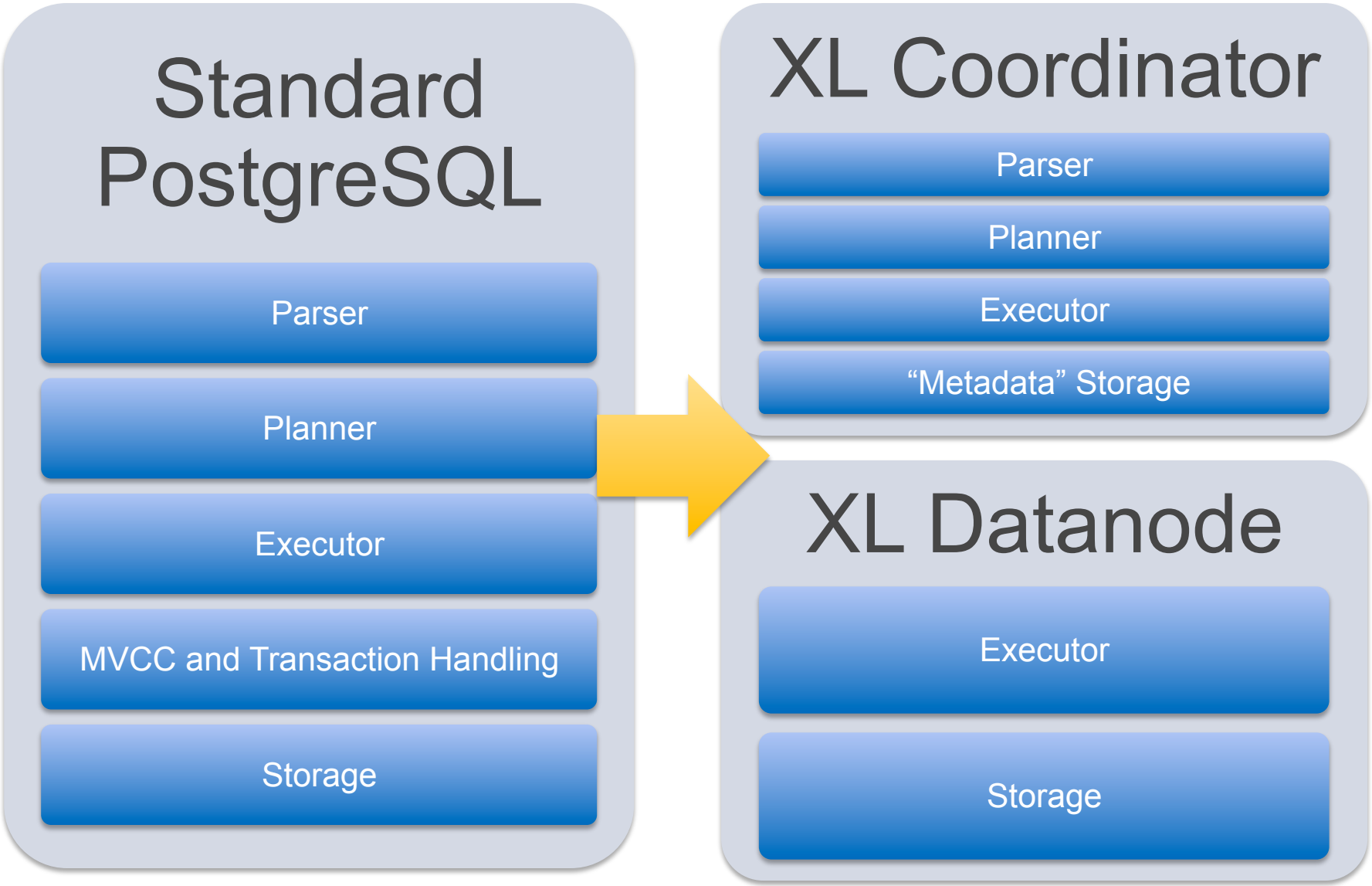
Postgres-XL Technology

- GTM maintains a consistent view of the data




Standard PostgreSQL
Parser
Planner
Executor
MVCC and Transaction Handling
Storage


Standard PostgreSQL
Parser
Planner
Executor
MVCC and Transaction Handling
Storage
GTM
MVCC and Transaction Handling
• Transaction ID (XID) Generation
• Snapshot Handling

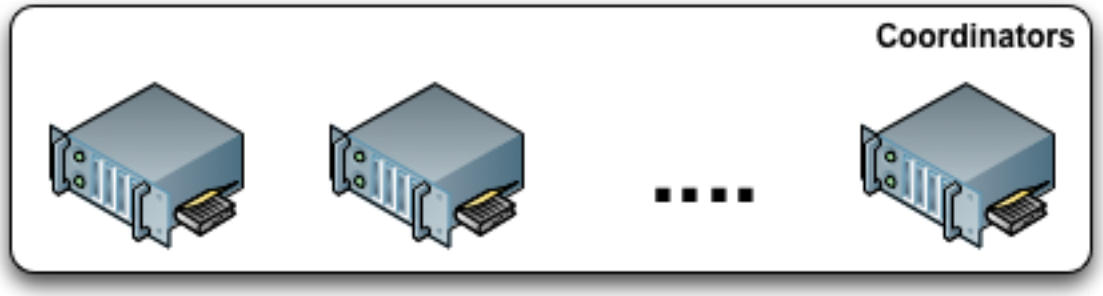

Standard PostgreSQL
Parser
Planner
Executor
MVCC and Transaction Handling
Storage
XL Coordinator
Parser
Planner
Executor
“Metadata” Storage
XL Datanode
Executor
Storage

Postgres-XL Architecture

- Based on the Postgres-XC project
- Coordinators
  - Connection point for applications
  - Parsing and planning of queries
- Data Nodes
  - Actual data storage
  - Local execution of queries
- Global Transaction Manager (GTM)
  - Provides a consistent view to transactions
  - GTM Proxy to increase performance

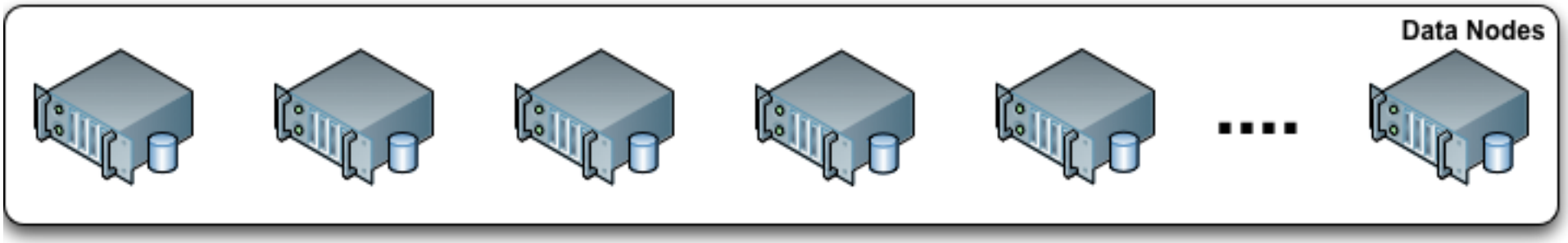
Coordinators

- Handles network connectivity to the client
- Parse and plan statements
- Perform final query processing of intermediate result sets
- Manages two-phase commit
- Stores global catalog information



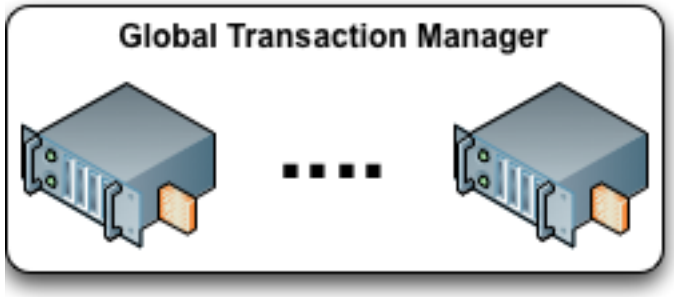
Data Nodes

- Stores tables and indexes
- Only coordinators connects to data nodes
- Executes queries from the coordinators
- Communicates with peer data nodes for distributed joins



Global Transaction Manager (GTM)

- Handles necessary MVCC tasks
  - Transaction IDs
  - Snapshots
- Manages cluster wide values
  - Timestamps
  - Sequences
- GTM Standby can be configured



Global Transaction Manager (GTM)

- Can this be a single point of failure?

Global Transaction Manager (GTM)

- Can this be a single point of failure?

Yes.

Global Transaction Manager (GTM)

- Can this be a single point of failure?

Yes.

Solution: GTM Standby

Global Transaction Manager (GTM)

- Can GTM be a bottleneck?

Global Transaction Manager (GTM)

- Can GTM be a bottleneck?

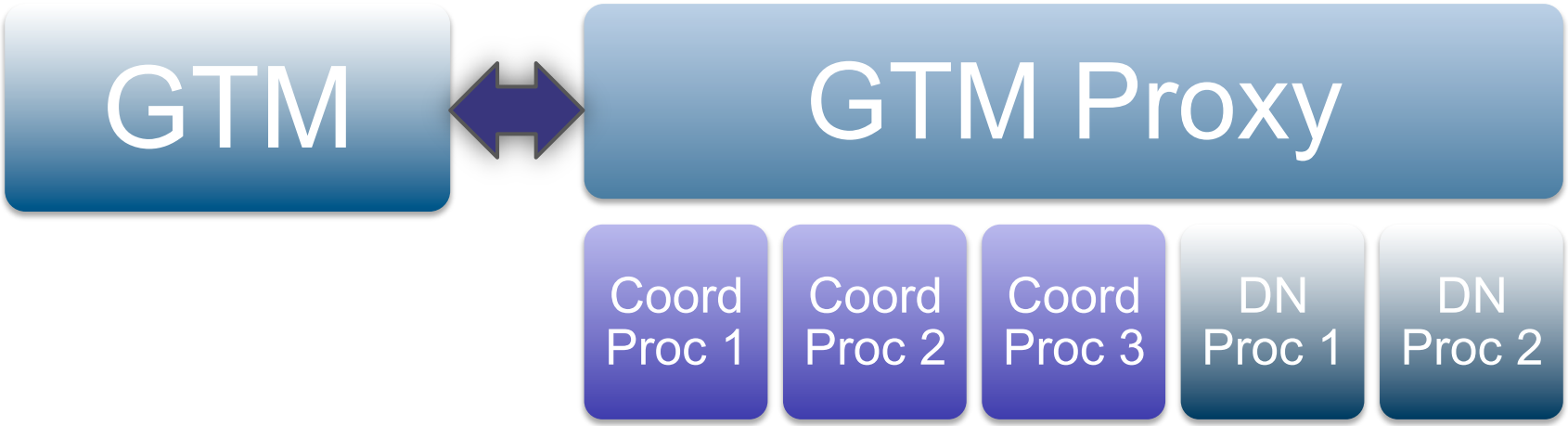
Yes.

Global Transaction Manager (GTM)

- Can GTM be a bottleneck?

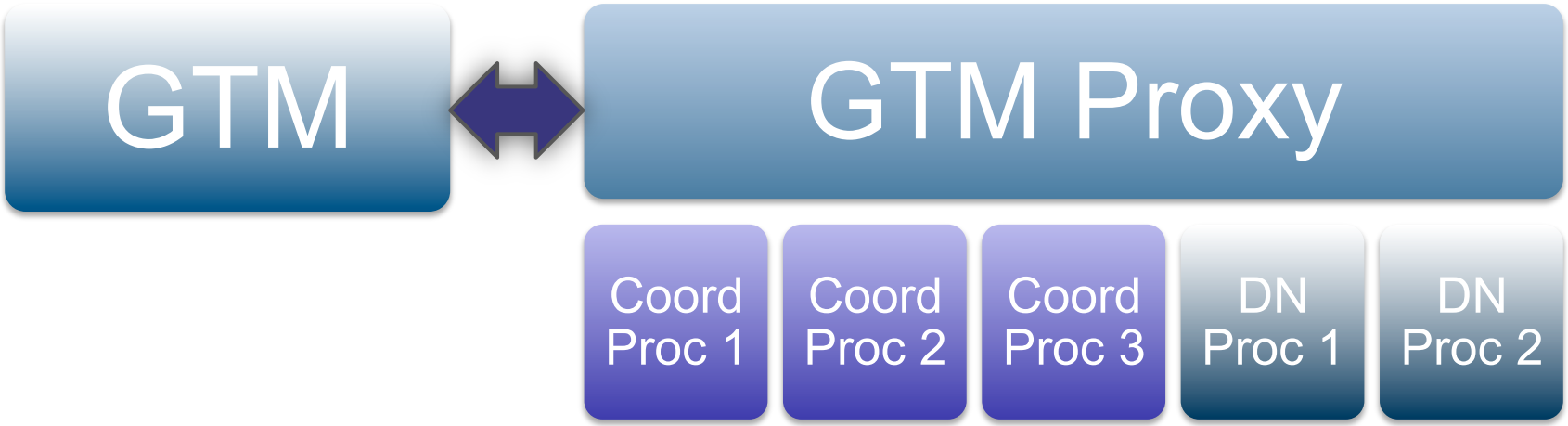
Yes.

Solution: GTM Proxy



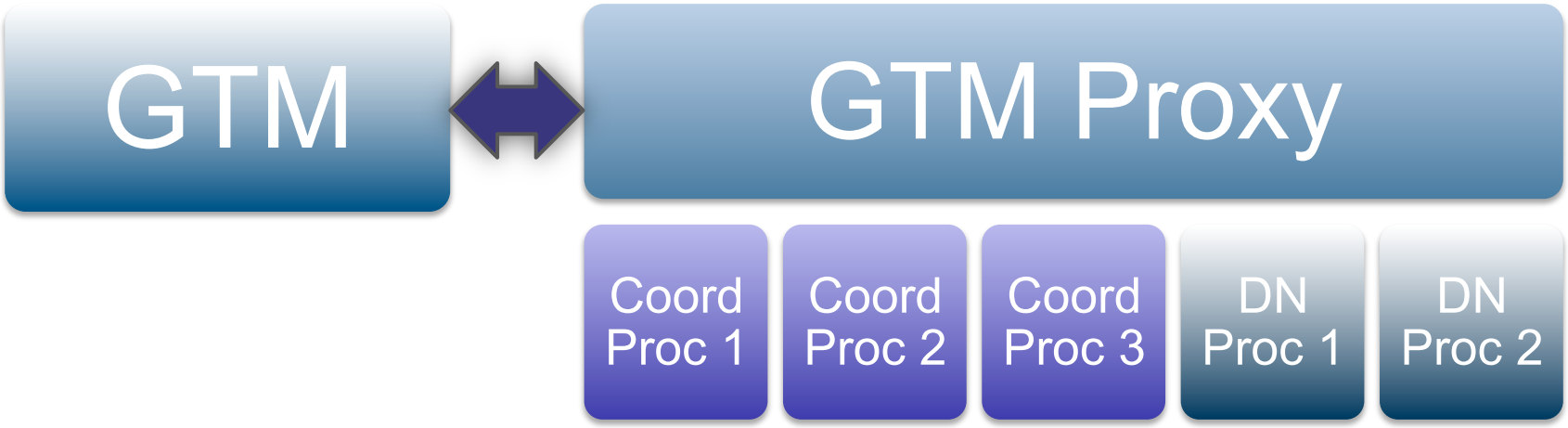
GTM Proxy

- Runs alongside Coordinator or Datanode
- Backends use it instead of GTM
- Groups requests together
- Obtain range of transaction ids (XIDs)
- Obtain snapshot



GTM Proxy

- Example: 10 process request a transaction id
- They each connect to local GTM Proxy
- GTM Proxy sends **one** request to GTM for 10 XIDs
- GTM locks procarray, allocates 10 XIDs
- GTM returns range
- GTM Proxy demuxes to processes

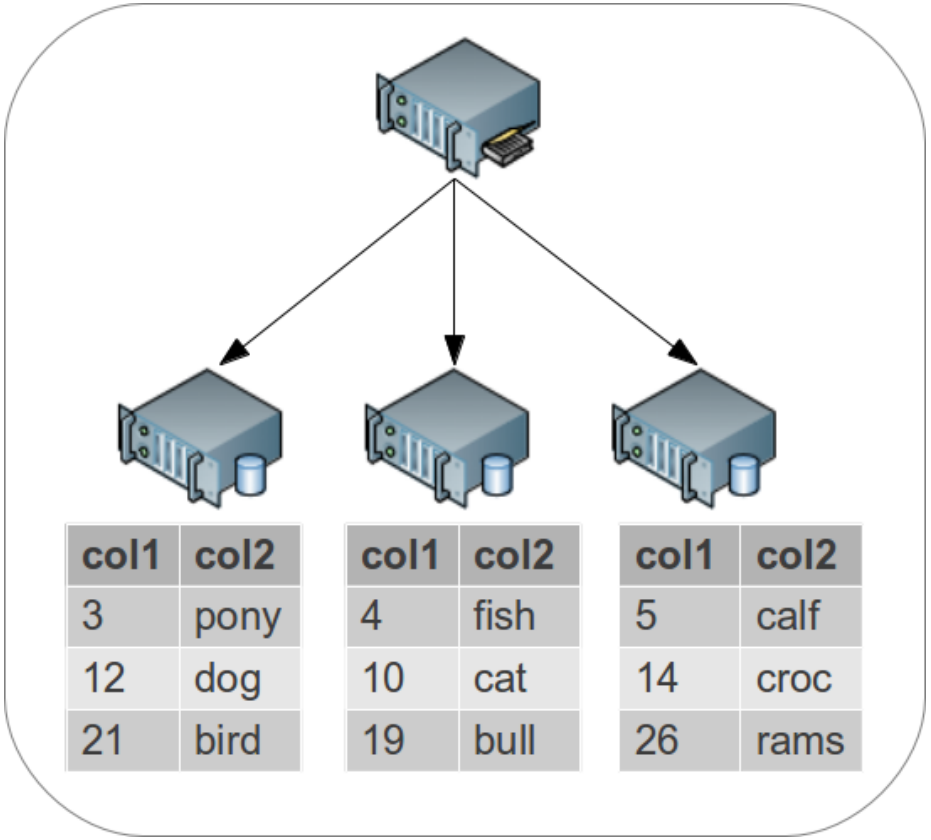


Data Distribution – Replicated Tables

- Good for read only and read mainly tables
- Sometimes good for dimension tables in data warehousing
- If coordinator and datanode are colocated, local read occurs
- Bad for write-heavy tables
- Each row is replicated to all data nodes
- Statement based replication
- “Primary” avoids deadlocks

Data Distribution – Distributed Tables

- Good for write-heavy tables
- Good for fact tables in data warehousing
- Each row is stored on one data node
- Available strategies
  - Hash
  - Round Robin
  - Modulo



Availability

- No Single Point of Failure
  - Global Transaction Manager Standby
  - Coordinators are load balanced
  - Streaming replication for data nodes
- But, currently manual…

DDL

Defining Tables

CREATE TABLE my_table (…)

DISTRIBUTE BY

HASH(col) | MODULO(col) | ROUNDROBIN | REPLICATION

[TO NODE (nodename[,nodename…])]

Defining Tables

```
CREATE TABLE state_code (
  state_code  CHAR(2),
  state_name VARCHAR,
  :
) DISTRIBUTE BY REPLICATION
```

An exact replica on each node

Defining Tables

```
CREATE TABLE invoice (
  invoice_id SERIAL,
  cust_id INT,
  :
) DISTRIBUTE BY HASH(invoice_id)
```

Distributed evenly amongst sharded nodes

Defining Tables

```
CREATE TABLE invoice (
  invoice_id SERIAL,
  cust_id INT ….
) DISTRIBUTE BY HASH(invoice_id);

CREATE TABLE lineitem (
  lineitem_id SERIAL,
  invoice_id INT …
) DISTRIBUTE BY HASH(invoice_id);
```

Joins on invoice_id can be pushed down to the datanodes

test2=# create table t1 (col1 int, col2 int) **distribute by hash(col1);**

test2=# create table t2 (col3 int, col4 int) **distribute by hash(col3);**

explain select * from t1 inner join t2 on **col1 = col3**;

Join push-down. Looks much like regular PostgreSQL

Remote Subquery Scan on all (**datanode_1,datanode_2**)

-> Hash Join

Hash Cond:

-> Seq Scan on t1

-> Hash

-> Seq Scan on t2

```
test2=# explain select * from  t1 inner join t2
on col2 = col3;

Remote Subquery Scan on all
(datanode_1,datanode_2)
   ->  Hash Join
         Hash Cond:
         ->  Remote Subquery Scan on all
(datanode_1,datanode_2)
               Distribute results by H: col2
               ->  Seq Scan on t1
         ->  Hash
               ->  Seq Scan on t2
```

Will read t1.col2 once and put in shared queue for consumption for other datanodes for joining. Datanode-datanode communication and parallelism

By the way

- PostgreSQL does not have query parallelism – yet

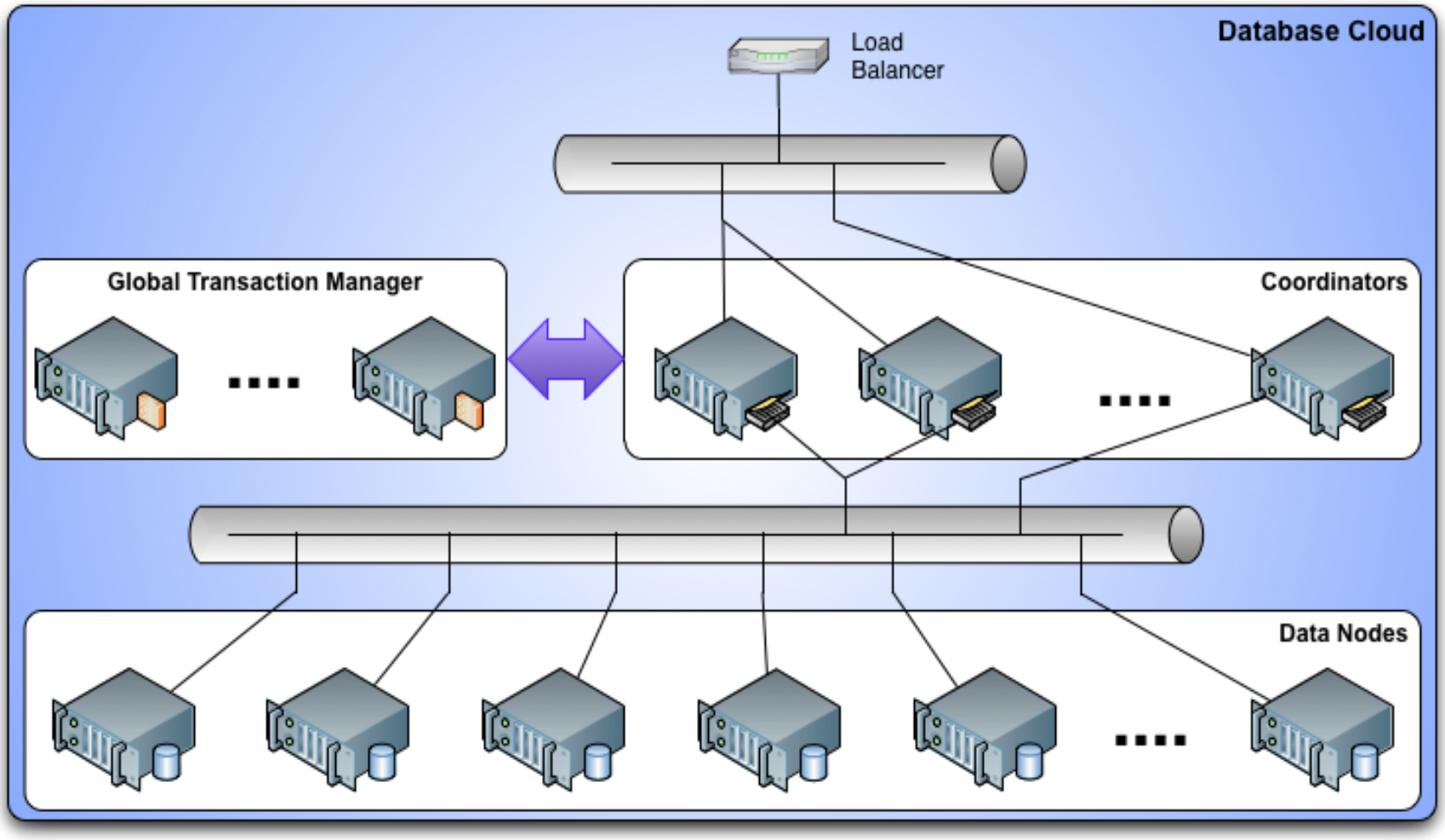
By the way

- PostgreSQL does not have query parallelism – yet
- Example: 16 core box, 1 user, 1 query
  - Just 1 core will be used
  - **15 (or so) idle cores**

By the way

- PostgreSQL does not have query parallelism – yet
- Example: 16 core box, 1 user, 1 query
  - Just 1 core will be used
  - 15 (or so) idle cores

- You can use Postgres-XL even on a single server and configure multiple datanodes to achieve better performance thanks to parallelism


Database Cloud
Load
Balancer
Global Transaction Manager
Coordinators
Data Nodes

Configuration


168 FEET 7 INCHES
VERRÜCKT
168 FEET 7 INCHES

Use pgxc_ctl!

(demo in a few minutes)

(please get from git HEAD for latest bug fixes)

Otherwise, PostgreSQL-like steps apply:

initgtm

initdb

postgresql.conf

- Very similar to regular PostgreSQL
- But there are extra parameters
  - max_pool_size
  - min_pool_size
  - pool_maintenance _timeout
  - remote_query_cost
  - network_byte_cost
  - sequence_range
  - pooler_port
  - gtm_host, gtm_port
  - shared_queues
  - shared_queue_size

Install

- Download RPMs
  - http://sourceforge.net/projects/postgres-xl/files/Releases/Version_9.2rc/

Or

- configure; make; make install

Setup Environment

- .bashrc
- ssh used, so ad installation bin dir to PATH

Avoid password with pgxc_ctl

- ssh-keygen –t rsa (in ~/.ssh)
- For local test, no need to copy key, already there
- cat ~/.ssh/id_rsa.pub >> ~/.ssh/ authorized_keys

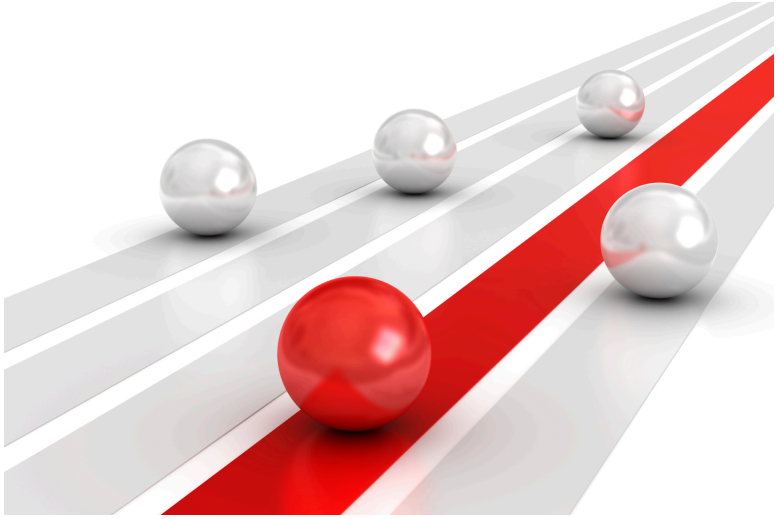
pgxc_ctl

- Initialize cluster
- Start/stop
- Deploy to node
- Add coordinator
- Add data node / slave
- Add GTM slave

pgxc_ctl.conf

- Let’s create a local test cluster!
  - One GTM
  - One Coordinator
  - Two Datanodes

  Make sure all are using different ports
    - including pooler_port



pgxc_ctl.conf

pgxcOwner=pgxl

pgxcUser=$pgxcOwner

pgxcInstallDir=/usr/postgres-xl-9.2

pgxc_ctl.conf

gtmMasterDir=/data/gtm_master

gtmMasterServer=localhost

gtmSlave=n

gtmProxy=n

pgxc_ctl.conf

```
coordMasterDir=/data/coord_master
coordNames=(coord1)
coordPorts=(5432)
poolerPorts=(20010)
coordMasterServers=(localhost)
coordMasterDirs=($coordMasterDir/coord1)
```

pgxc_ctl.conf

coordMaxWALSenders=($coordMaxWALsender)

coordSlave=n

coordSpecificExtraConfig=(none)

coordSpecificExtraPgHba=(none)

pgxc_ctl.conf

datanodeNames=(dn1 dn2)

datanodeMasterDir=/data/dn_master

datanodePorts=(5433 5434)

datanodePoolerPorts=(20011 20012)

datanodeMasterServers=(localhost localhost)

datanodeMasterDirs=($datanodeMasterDir/dn1 $datanodeMasterDir/dn2)

pgxc_ctl.conf

datanodeMaxWALSenders=($datanodeMaxWalSender $datanodeMaxWalSender)

datanodeSlave=n

primaryDatanode=dn1

pgxc_ctl

pgxc_ctl init all

Now have a working cluster

PostgreSQL Differences

pg_catalog

- pgxc_node
  - Coordinator and Datanode definition
  - Currently not GTM
- pgxc_class
  - Table replication or distribution info

Additional Commands

- PAUSE CLUSTER / UNPAUSE CLUSTER
  - Wait for current transactions to complete
  - Prevent new commands until UNPAUSE
- EXECUTE DIRECT

  ON (nodename) ‘command’
- CLEAN CONNECTION
  - Cleans connection pool
- CREATE NODE / ALTER NODE

Noteworthy

- SELECT pgxc_pool_reload()
- pgxc_clean for 2PC

Looking at Postgres-XL Code

- XC

#ifdef PGXC

- XL

#ifdef XCP

#ifndef XCP

Community

Website and Code

- postgres-xl.org
- sourceforge.net/projects/postgres-xl
- Will add mirror for github.com
- Docs
  - files.postgres-xl.org/documentation/index.html

Philosophy

- Stability and bug fixes over features
- Performance-focused
- Open and inclusive
  - Welcome input for roadmap, priorities and design
  - Welcome others to become core members and contributors

Roadmap

- New release
- Improve Cluster Management
- Catch up to PostgreSQL Community -> 9.3, 9.4
- Make XL a more robust analytical platform
  - Distributed Foreign Data Wrappers
- GTM optional
- Native High Availability

Thank You!

msharp@maputodata.com
@mason_db