High Availability in Postgres-XC
The symmetric PostgreSQL cluster

Koichi Suzuki
Postgres-XC Development Group

PostgreSQL Conference Europe, 2012
October 24th, 2012
Prague, Czech Republic

NTT DATA

Outline of the Talk

- Postgres-XC overview
  - What it is
  - Architecture and scalability
- SPOF analysis
- Failure characteristics
  - Comparison with a commercial one
- Failure handling and HA
- Current status and future schedule

Self Introduction

- Postgres-XC leader and core architect, as well as a core developer
  - Whole architecture design
  - Global transaction management and data distribution as a key for write-scalability
- Work for NTT DATA Intellilink
  - Subsidiary of NTT DATA corporation dedicated for system platform
  - Member of NTT group company
- Resources
  - koichi.clarinet@gmail.com (facebook, linkedin)
  - @koichiclarinet (twitter)

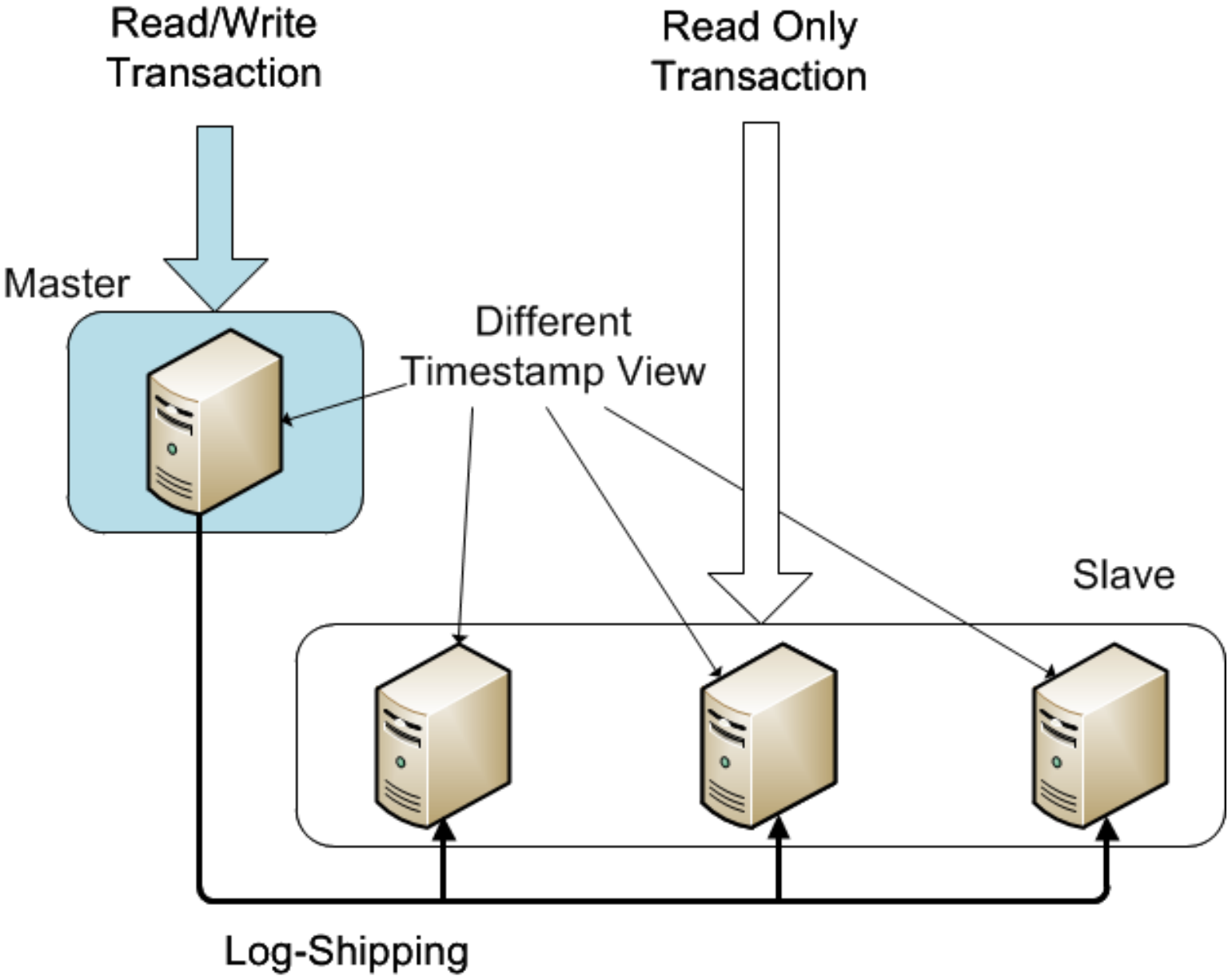
Postgres-XC Overview

NTT DATA

What is Postgres-XC

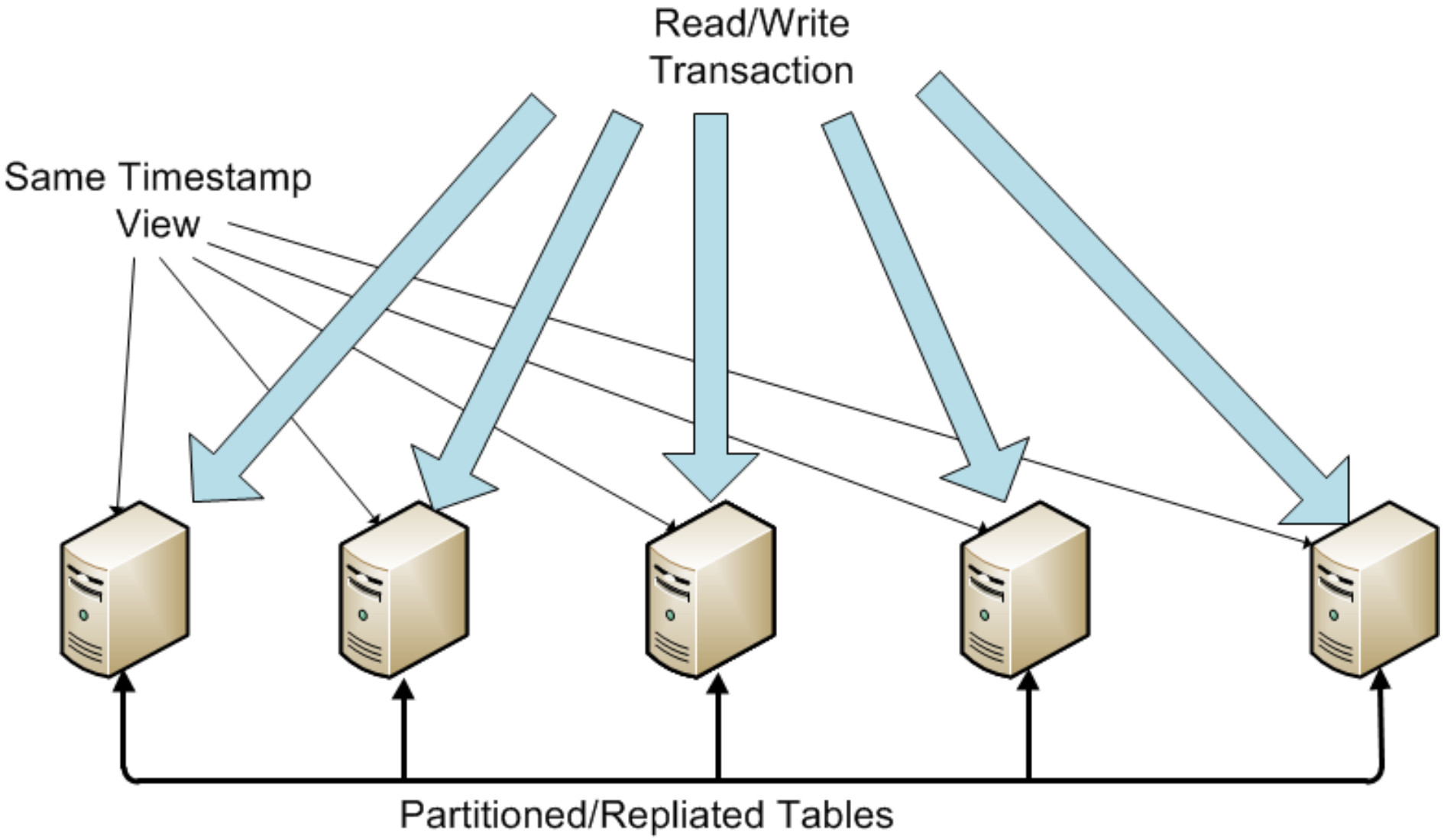
- Symmetric PostgerSQL cluster
  - No master/slave replication
  - No read-only clusters
  - Every node can issue both read/write
  - Every node provides single consistent database view
  - Transparent transaction management
- Not just a replication
  - Each table can be replicated/distributed by sharding
  - Parallel transaction/query execution
    - So both read/write scalability

Master/Slave with Log Shipping



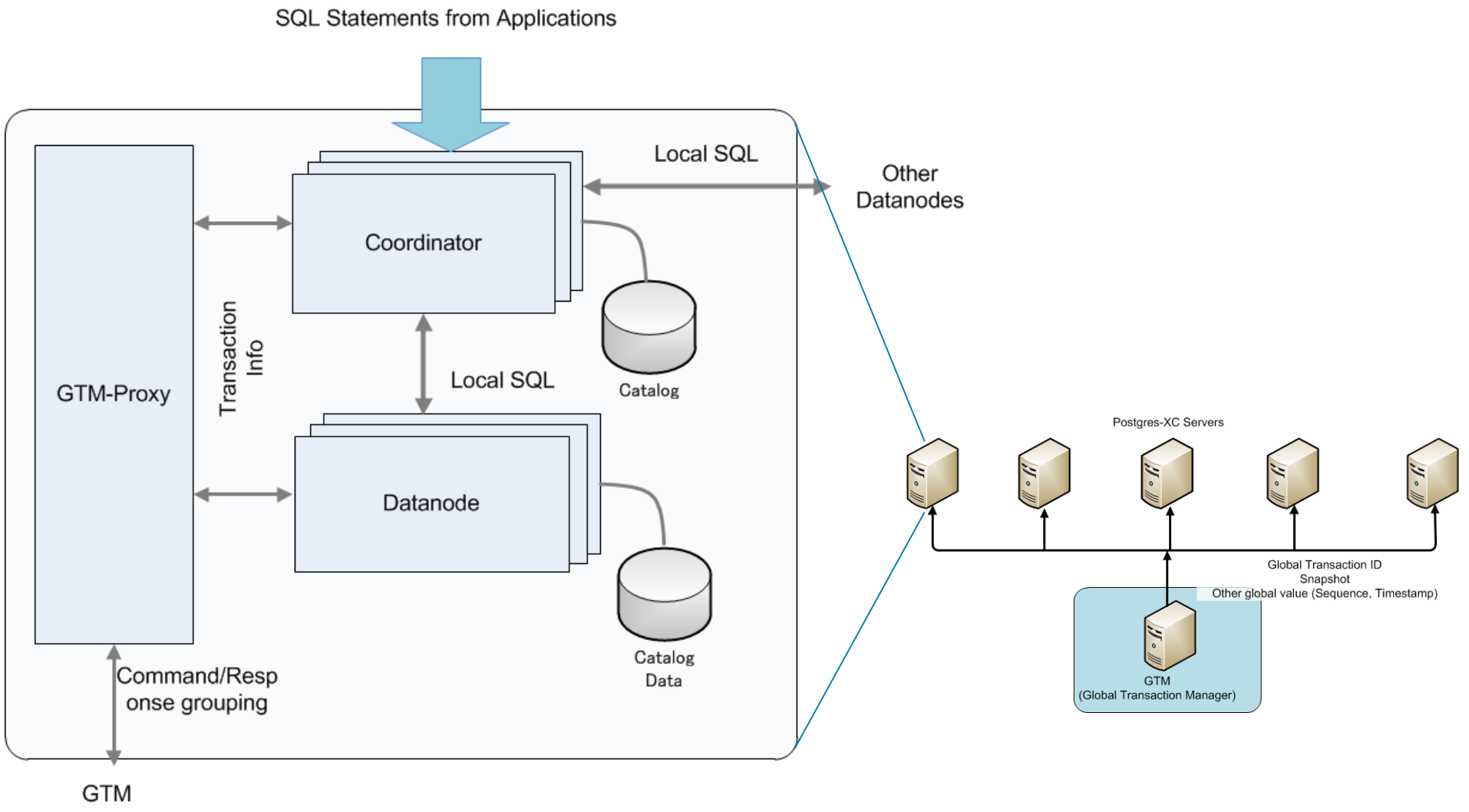
October 24th, 2012

Postgres-XC Symmetric Cluster



October 24th, 2012

Architecture and Configuration



October 24th, 2012

Outline of Components

- GTM (Global Transaction Manager)
  - Distributed MVCC
    - Provide global transaction ID (GXID) to all the transactions
    - Provide global snapshot to all the transactions
  - Sequence
- GTM_Proxy
  - Group communications to GTM and reduce amount of GTM network workload
- Coordinator
  - Handles incoming SQL statements
    - Parse, plan, conduct execution in datanodes and the coordinator.
    - Integrate local results from each datanode involved.
- Datanode
  - Actual data storage
    - Almost vanilla PostgreSQL

Share the binary

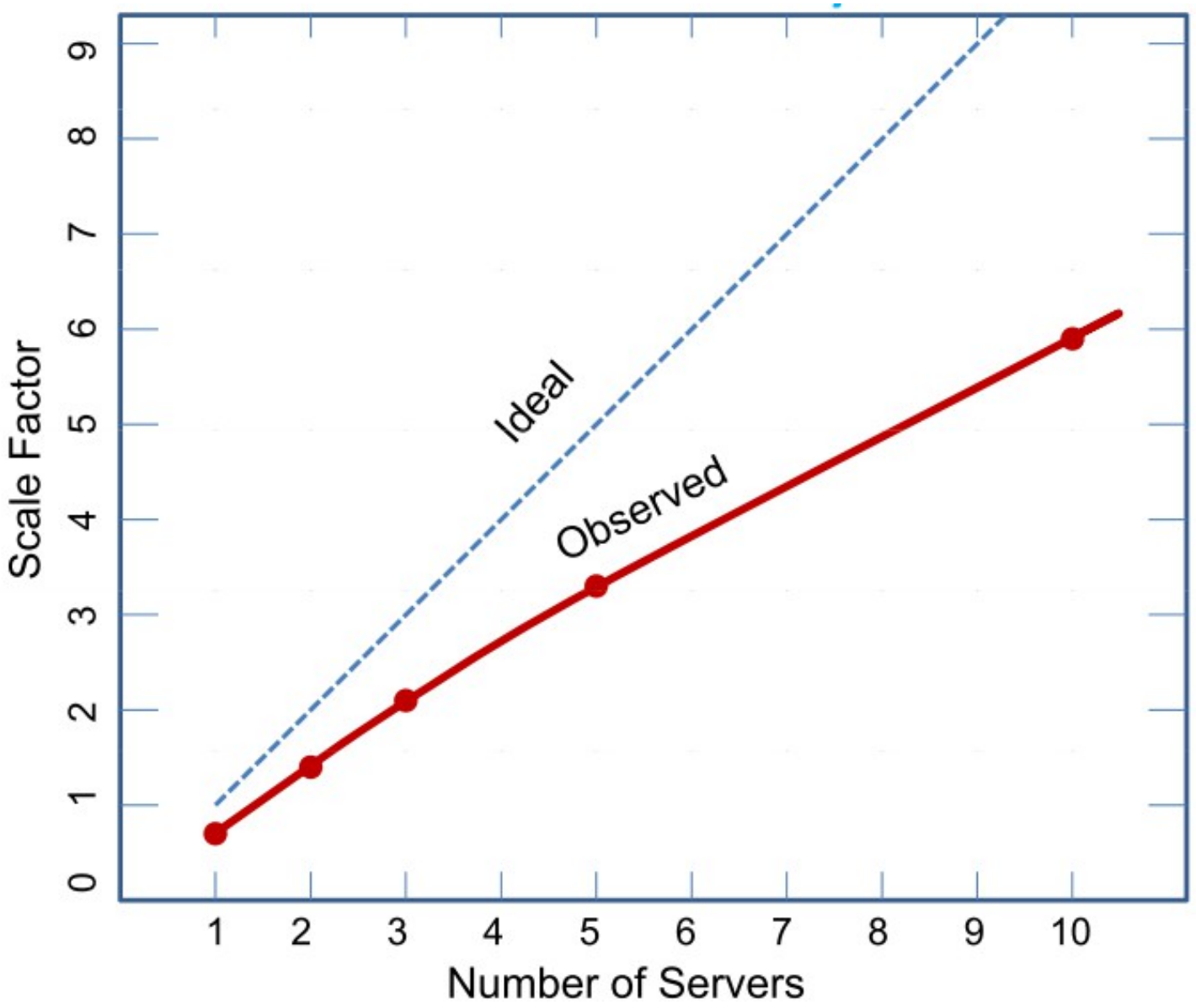
Flexible Configuration of Comonents

- Each coordinator/datanode can be configured in any servers, same or different, as log as
  - Each component does not share the following set of resources
    - Listening IP addresses
    - Listening port
    - Work Directories
- For simplicity and better workload balance, the following is advised:
  - Have separate GTM server
  - Each of others should have
    - One GTM proxy (for network workload improvement)
    - One Coordinator
      - Some transactions may benefit from data located at local datanode (preferred node)
    - One Datanode
      - Automatic workload balance between coordinator and datanode

Flexible Configuration of Comonents

- Each coordinator/datanode can be configured in any servers, same or different, as log as
  - Each component does not share the following set of resources
    - Listening IP addresses
    - Listening port
    - Work Directories
- For simplicity and better workload balance, the following is advised:
  - Have separate GTM server
  - Each of others should have
    - One GTM proxy (for network workload improvement)
    - One Coordinator
      - Some transactions may benefit from data located at local datanode (preferred node)
    - One Datanode
      - Automatic workload balance between coordinator and datanode

Scalability



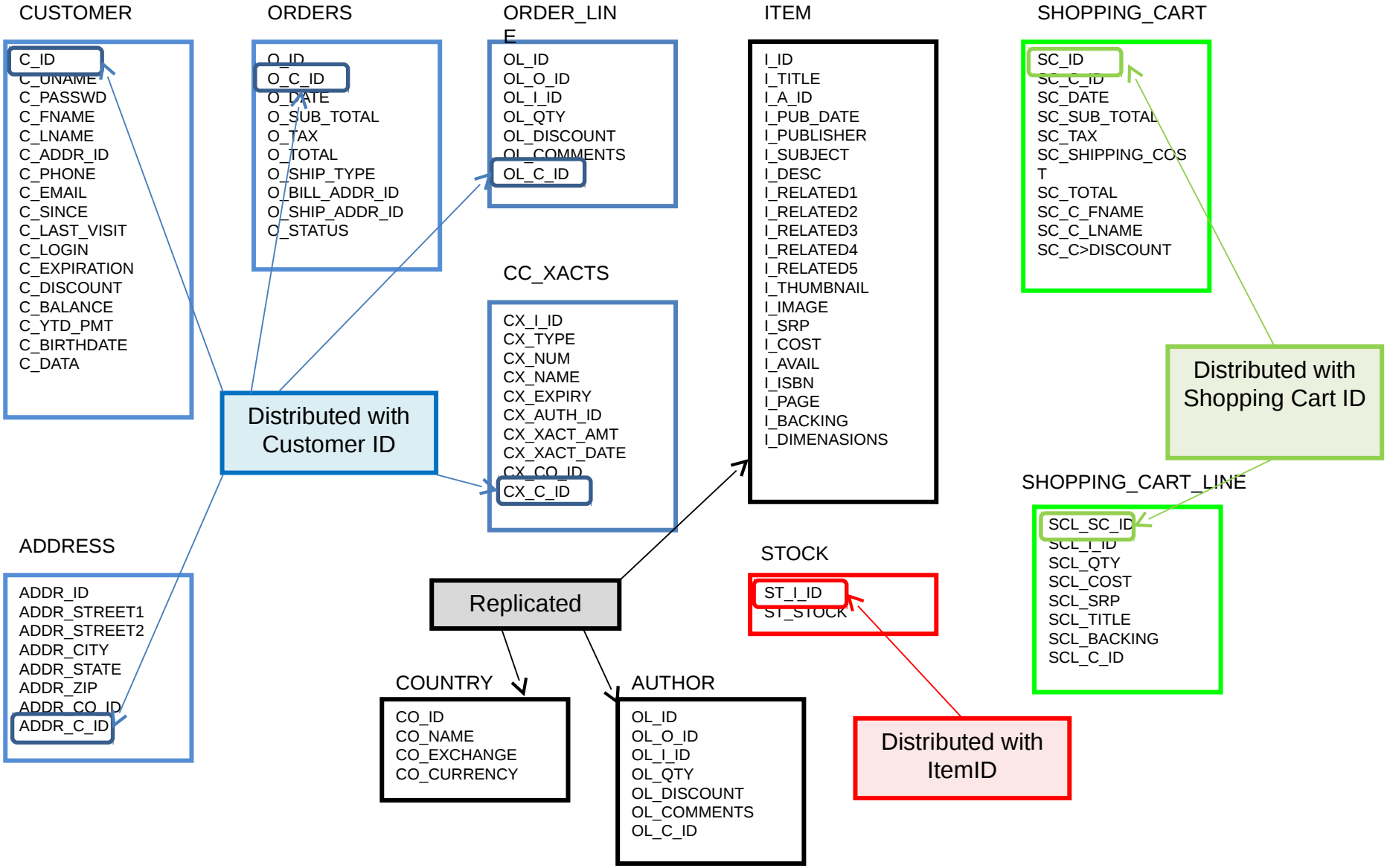
DBT-1 (Rev)

October 24th, 2012

Combining sharding and replication

NTT DATA

DBT-1 example



October 24th, 2012

How XC Scales both Read/Write

- Transaction Tables → Sharding
  - Only one write
  - Parallel writes in datanodes
- Master Tables → Replication
  - Relatively static: Not significant many-writes overhead
  - Local join with transaction tables → Most join operation can be done locally in datanodes

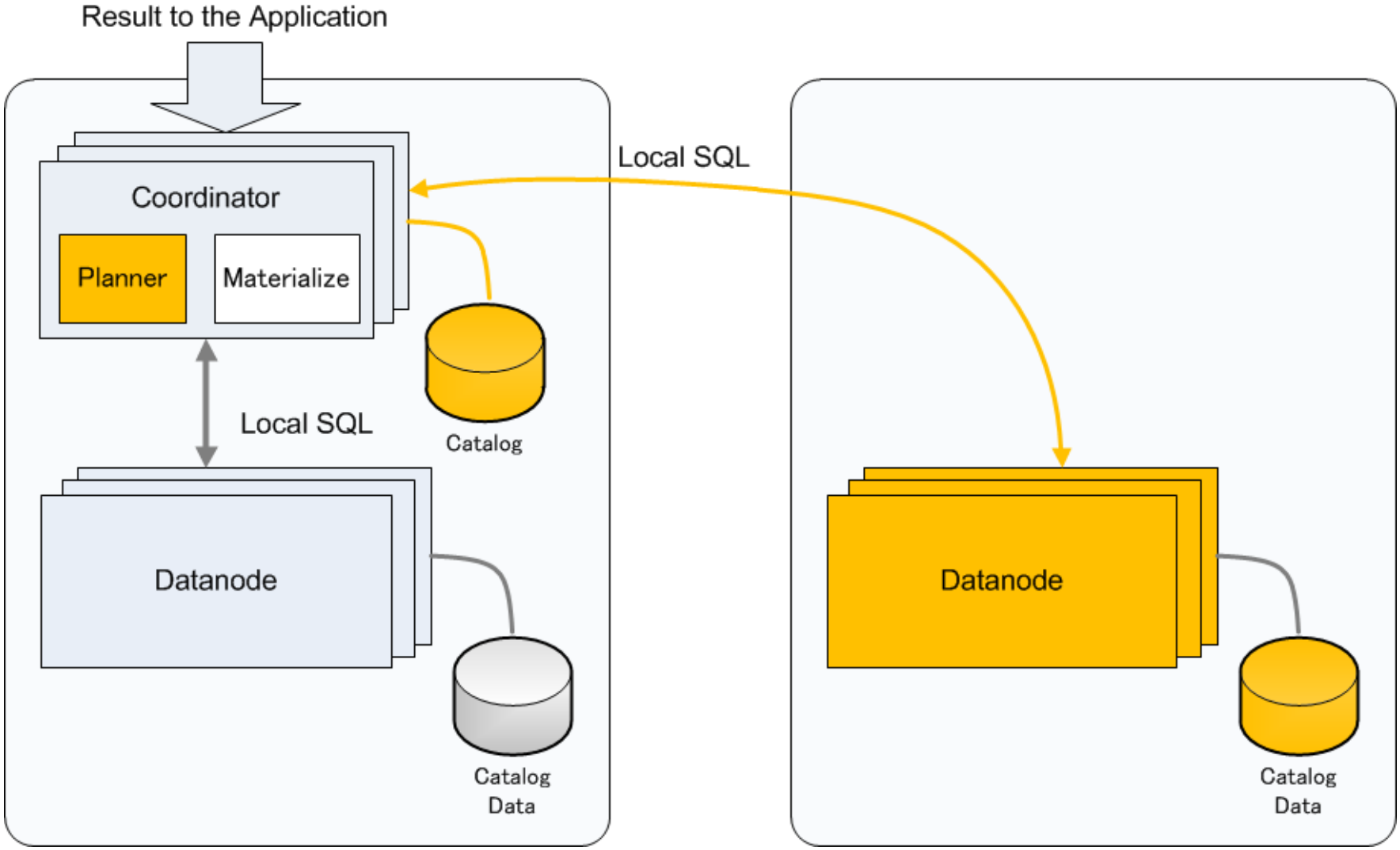
Postgres-XC API

- Binary compatible with PostgreSQL
  - Limited support for ODBC
  - JDBC may have a couple of restrictions
- Compatible statements to PostgreSQL
  - Slight difference
    - CREATE TABLE, etc.
    - Constraints can be enforced only locally in each datanodes.
  - Extra
    - Coordinator/datanode membership management, etc.
      - CREATE/ALTER/DROP NODE, EXECUTE DIRECT...
    - Extension in aggregates
      - Combiner functions
    - Maintain consistency in point-in-time-recovery
      - CREATE BARRIER
- No load balancing so far
- You should notice
  - OID is local to each node

October 24th, 2012

Join Example (1)

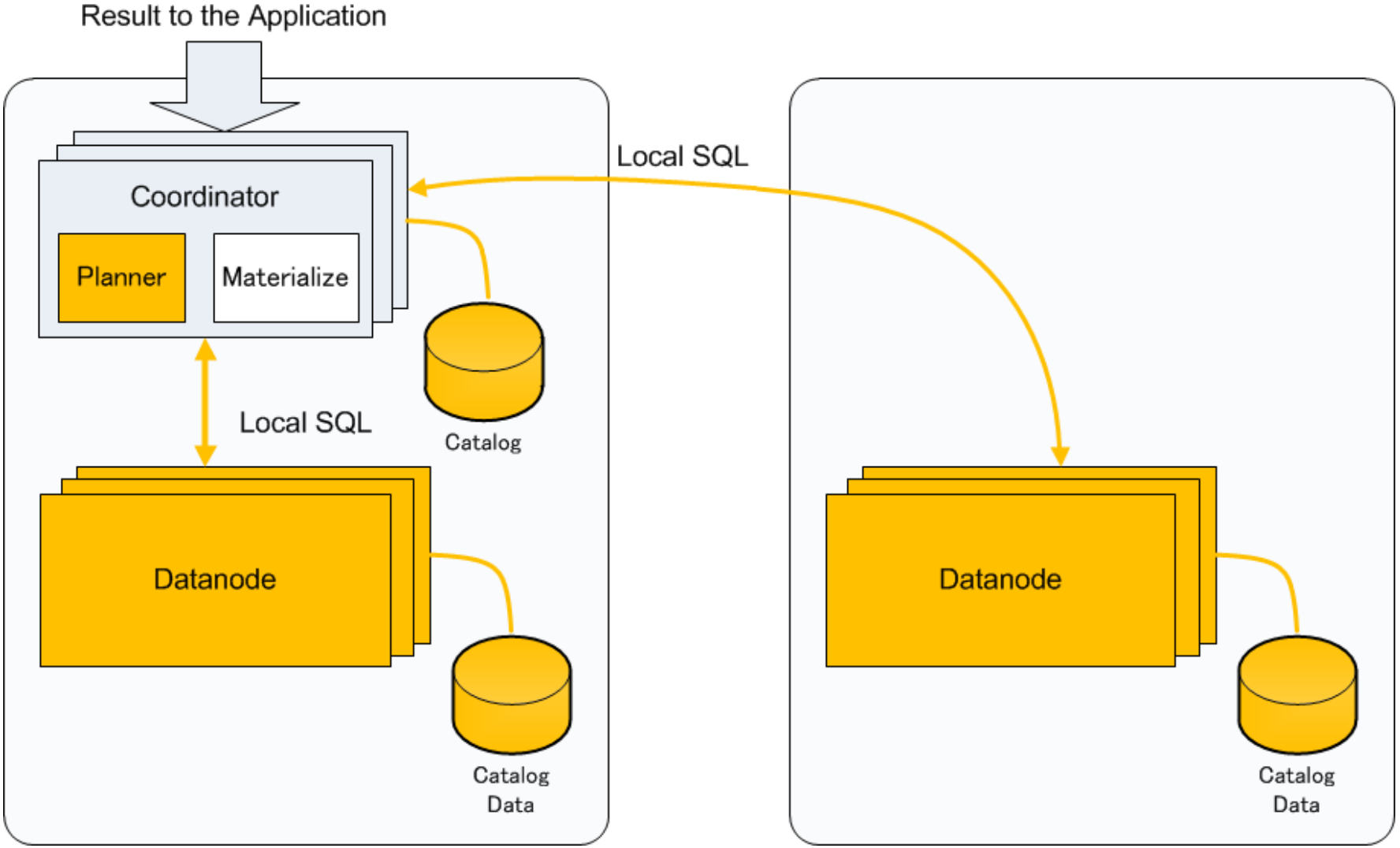
- Replicated Table and Partitioned Table
  - Can determine which datanode to go from WHERE clause



October 24th, 2012

Join Example (2)

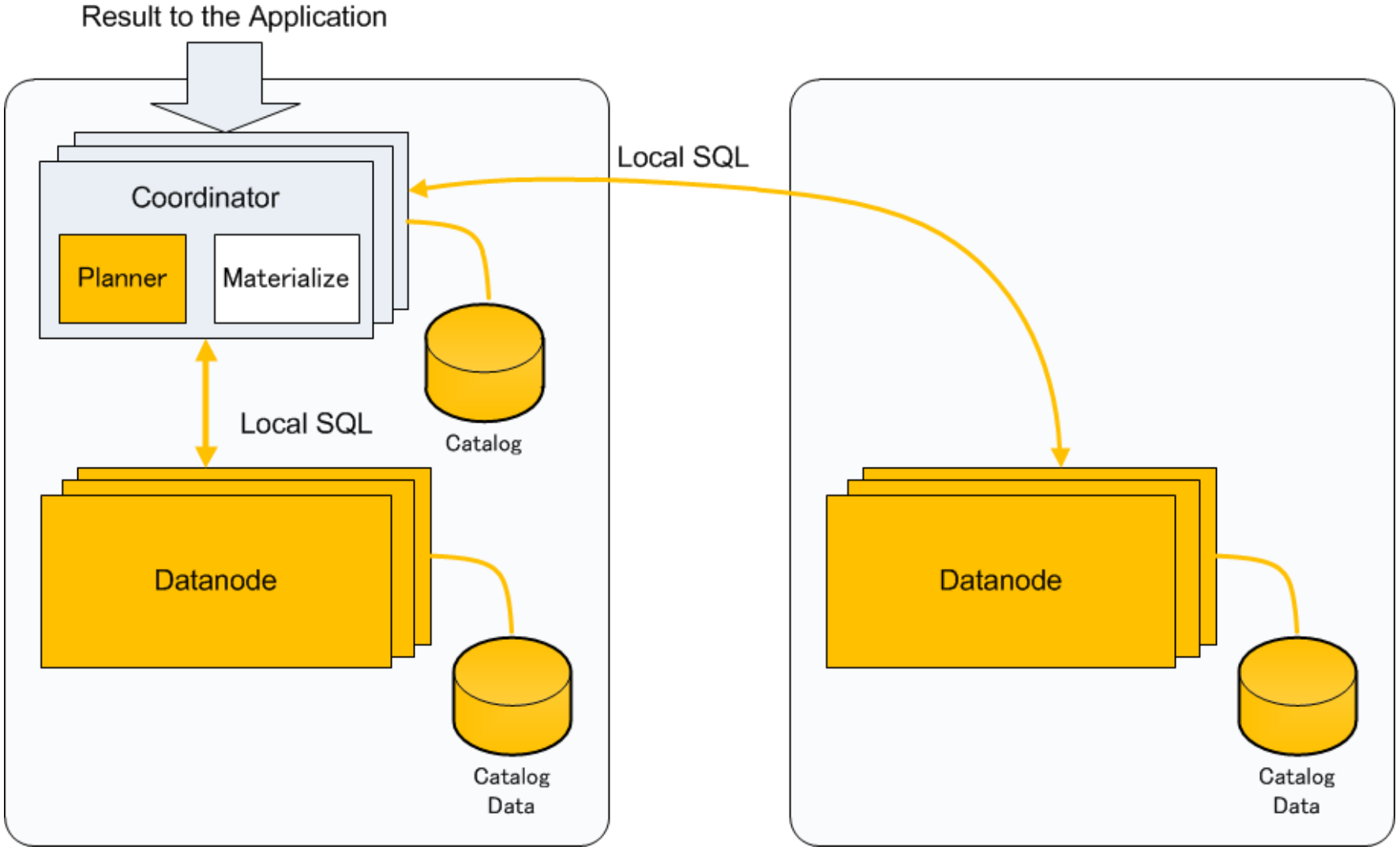
- Replicated Table and Partitioned Table
  - Cannot determine which datanode to go



October 24th, 2012

Join Example (3)

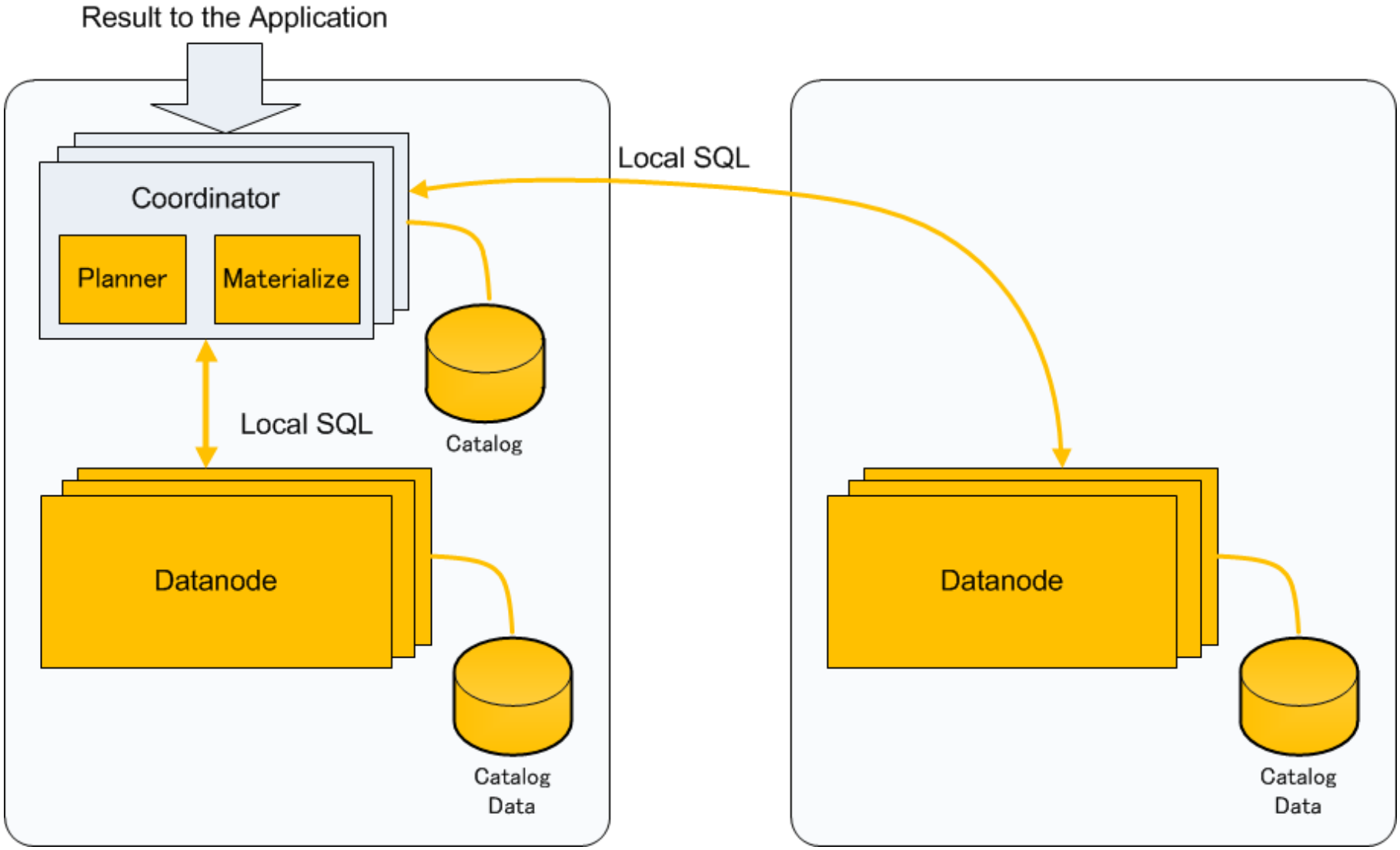
- Partitioned Table and Partitioned Table
  - Both Join columns are distribution (partitioning) column



October 24th, 2012

Join Example (4)

- Partitioned Table and Partitioned Table
  - One of Join columns are not distribution (partitioning) column



SPOF Analysis

NTT DATA

Postgres-XC Single Point of Failure

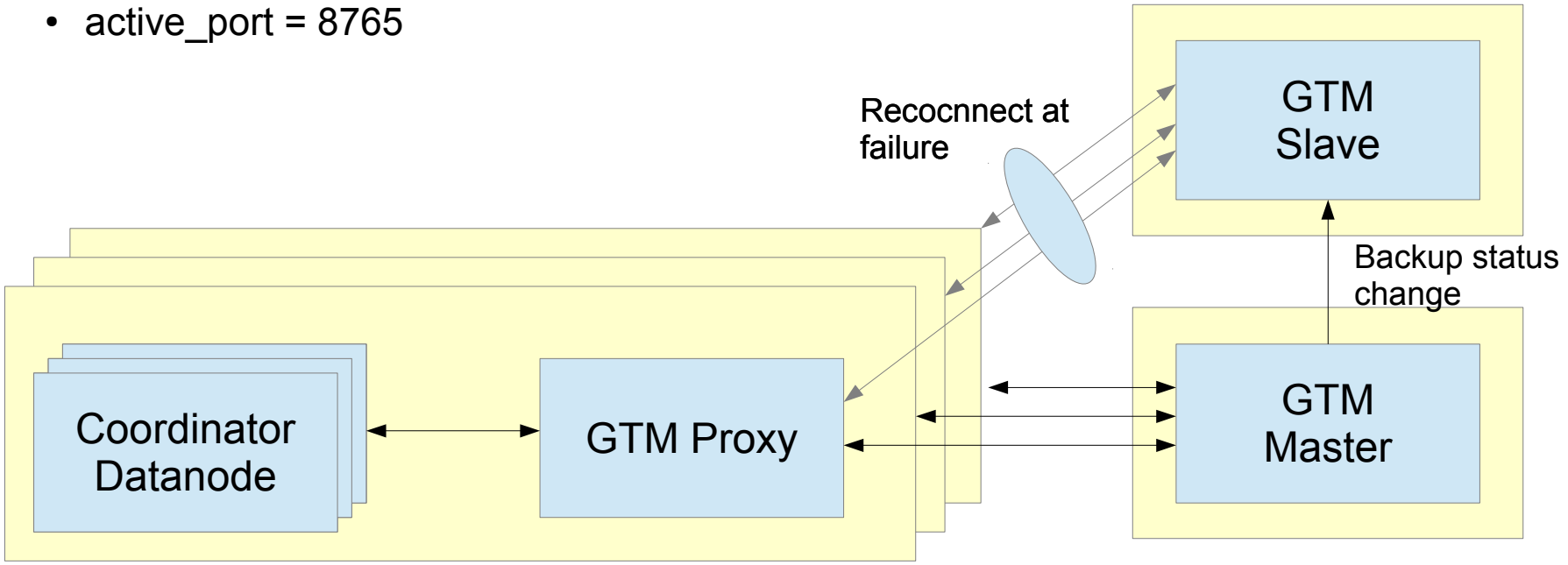
- GTM
  - Obviously SPOF
- GTM-Proxy
  - No persistent data hold
  - Just restart when fail
- Coordinator
  - Every coordinator is essentially a copy
  - When fails, other coordinators work
- Datanode
  - SPOF for sharded table

Backup for SPOF component

- GTM
  - Specific backup for GTM (GTM Standby)
    - Most information are kept on-memory
      - Open TXNs
        - Only the next GXID is needed to restart whole cluster, kept on disk.
    - Copies every internal status change to the backup
      - Similar to the log shipping in PostgreSQL
    - Can promote to the master
      - GTM-Proxy help this failover
- Datanode
  - Need backup
  - Can use PostgreSQL's means
    - Log Shipping
    - Shared disk
- Coordinator
  - Not critical but may want to have backups
  - Can use similar means as Datanodes.

GTM Slave (GTM Standby)

- Same binary to GTM
  - Backs up everything on the fly.
  - Can promote to the master (gtm_ctl promote)
  - Configure using gtm.conf
    - startup = ACT|STANDBY
    - active_host = 'active_gtm_host'
    - active_port = 8765



Datanodes

- Almost all the techniques for PostgreSQL backup/failover are available
  - Streaming replication
  - Shared disk re-mount
- Subject to coordinators
  - Coordinators should reconfigure failed datanode at failover
  - Coordinators should clean connections to failed datanode before reconfiguration
- GTM
  - Reconnect to (new) local GTM proxy

Coordinators

- Only catalog is stored
  - Very stable and static
  - All the coordinators are essentially the same copy
- Datanode HA technique can be applied
  - Streaming replication
  - Shared disk remount
- One more option at a failure
  - No failover
  - Remaining coordinators will take care of TXNs
  - Failed coordinator can be restored offline
    - Backup/restore
    - Copy catalogue from a remaining coordinator

Failure Characteristics

PostgreSQL

- Promote one of the slaves
- Application connect to promoted PostgreSQL
- Everything stops and then restarts

Postgres-XC

- Promote the slave of the failed component
- Reconfigure with new master
- Whole cluster continues to run, only affected TXNs fail

- Just one component failure may not lead to whole cluster fail.
- Can configure appropriate slaves for each component to continue XC operation.

Impact of each component failure (1)

- GTM
  - Failover to GTM slave
  - No TXN loss
- GTM-Proxy
  - Restart GTM-Proxy
  - Reconnect to other GTM-Proxy
  - Involved TXN may fail
- Coordinator
  - Involved TXN fails
  - Failover
  - or Use remaining coordinators

Impact of each component failure (2)

- Datanode
  - Failover to the slave
    - Streaming replication
    - Shared disk remount
      - Much less resources
      - Shared disk could be a SPOF

XC vs. O*R*

Feature	Postgres-XC	O____ R__
Background Databsae	PostgreSQL	O_____
Architecture	Shared Nothing	Shared Everything
Number of Servers	Experience: 10 Maybe 20 or more	?? (Most deployments are two server configuration)
Hardware Requirement	None	Shared Disk
Read Scale	Yes	Yes
Write Scale	Yes	Depends (Application level partitioning)
Storage Failure	Limited impact Component failover Cluster keeps running	Whole cluster fails Cluster-wide failover
Server Failure	Affected components needs failover Others keep running	Remaining servers continues service

Failure Handling and HA

NTT DATA

General

- XC is not a simple replication
  - When a component fails, other components is still alive and can continue to provide database service.
  - Remaining components may need reconfiguration to accommodate new master of the failed component.
- This section shows what to do to prepare slaves and failover each component
- Useful shell scripts will be found in `pgxc_ctl` directory in `https://github.com/koichi-szk/PGXC-Tools`

Coordinator Handling Options

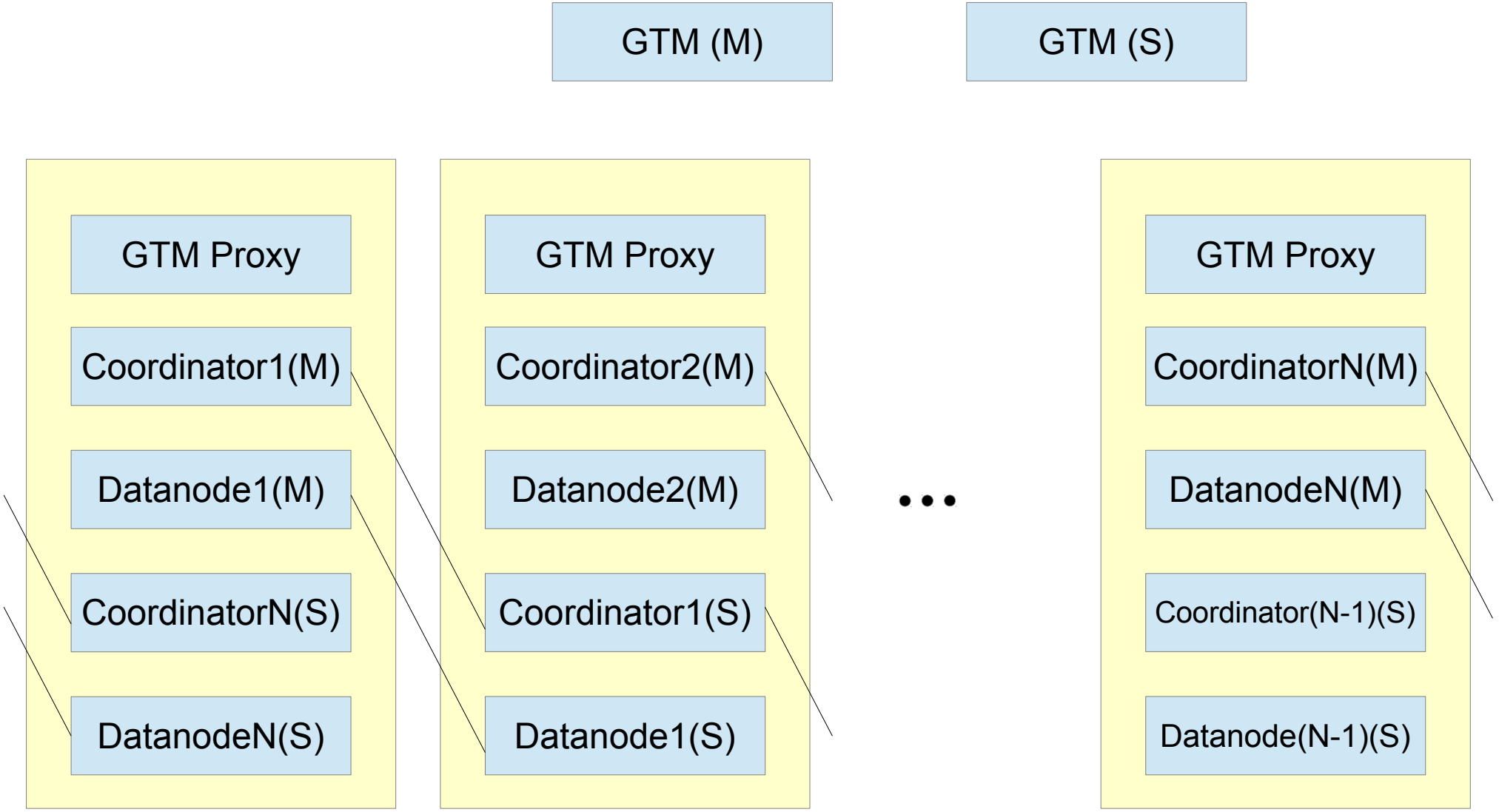
Option1. Use coordinator slave

- Additional resource for slaves

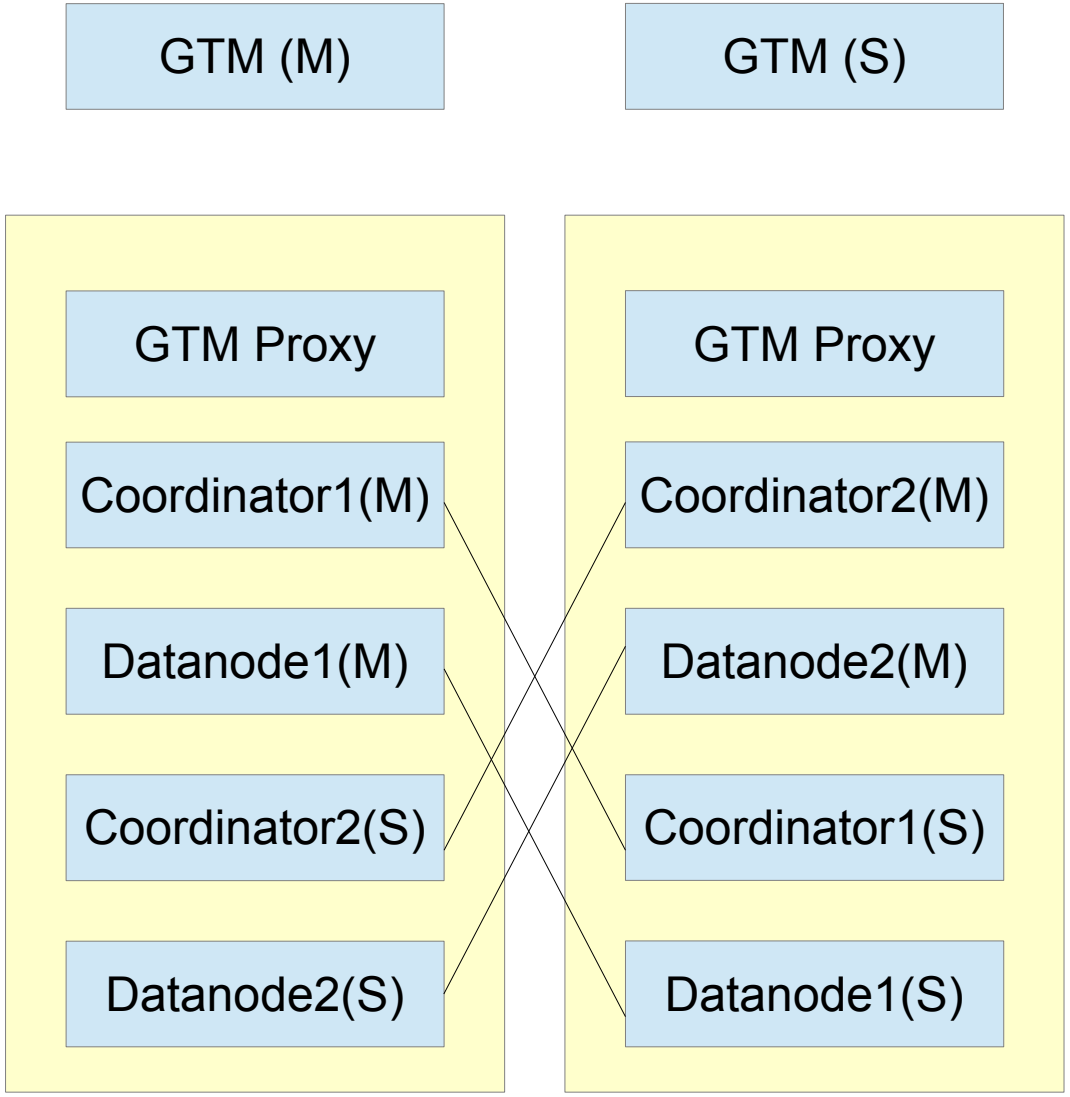
Option2. No backup

- Use remaining coordinator at failure
- Restore offline with backups

Circular Configuration



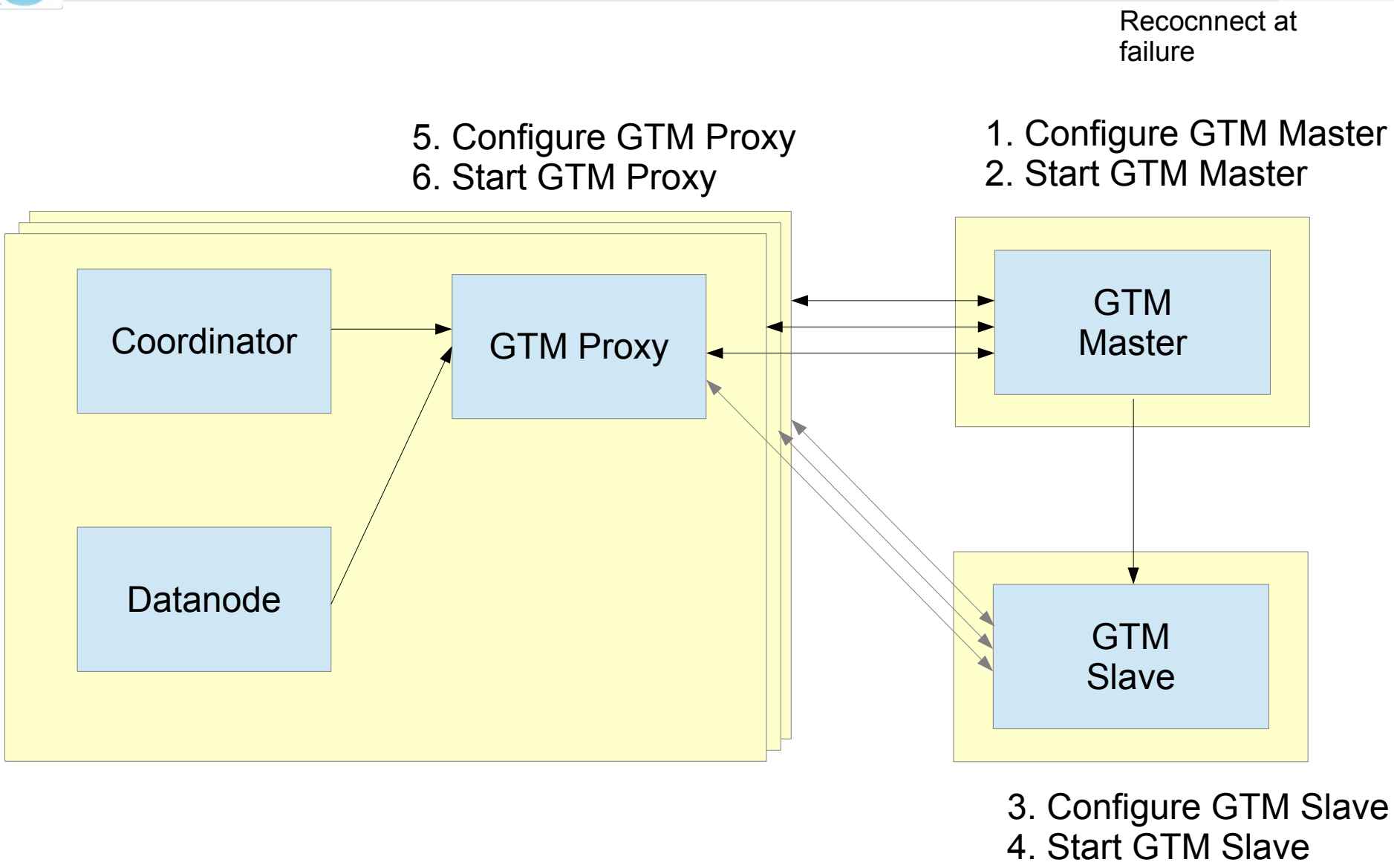
Pair Configuration



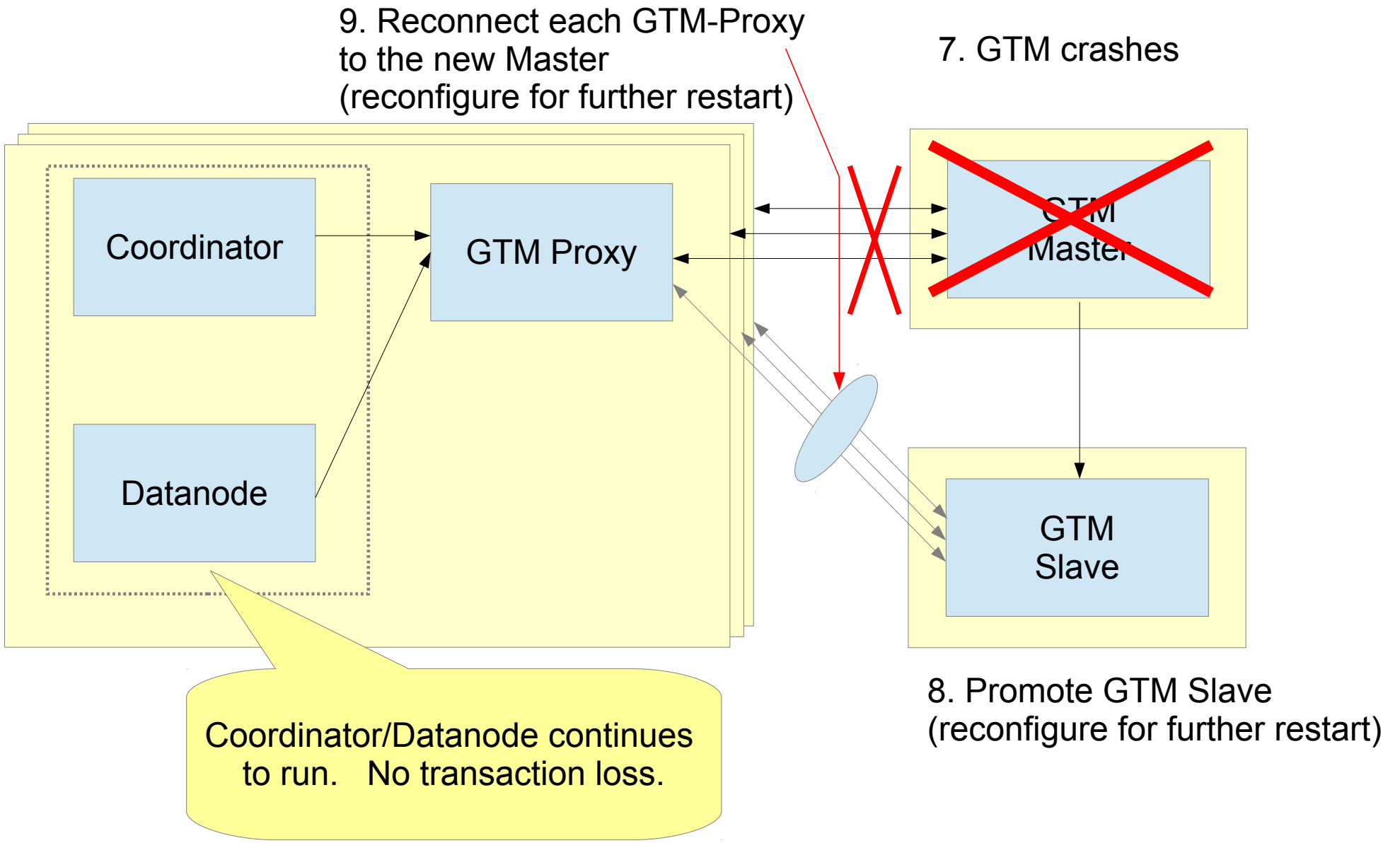
• • •

Easier integration with Pacemaker

Failure handling outline – GTM(1)



Failure handling outline – GTM(2)



October 24th, 2012

GTM failure handling steps (1)

1. Configure GTM master

```
$ ssh gtm_master_host initgtm -Z gtm -D gtm_master_dir
$ ssh gtm_master_host cat >> gtm_master_dir/gtm.conf << EOF
listen_addresses = 'listen_addresses'
port = gtmMasterPort
nodename = 'gtmName'
startup = ACT
EOF
$
```

2. Start GTM master

```
$ ssh gtm_master_host gtm_ctl start -Z gtm -D gtm_master_dir
$
```

GTM failure handling steps (2)

3. Configure GTM slave

```
$ ssh gtm_slave_host initgtm -Z gtm -D gtm_slave_dir
$ ssh gtm_slave_host cat >> gtm_slave_dir/gtm.conf << EOF
listen_addresses = 'listen_addresses'
port = gtmMasterPort
nodename = 'gtmName'
startup = STANDBY
active_host = 'gtmMasterServer'
active_port = gtmMasterPort
EOF
$
```

4. Start GTM slave

```
$ ssh gtm_slave_host gtm_ctl start -Z gtm -D gtm_slave_dir
$
```

GTM failure handling steps (3)

5. Configure GTM proxy

```
$ ssh $ii initgtm -Z gtm -D gtm_proxy_dir
$ ssh $ii cat >> gtm_proxy_dir/gtm_proxy.conf << EOF
listen_addresses = 'listen_addresses'
port = gtmProxyPort
nodename = 'gtm_proxy_Name'
startup = STANDBY
active_host = 'gtmMasterServer'
active_port = gtmMasterPort
gtm_host = gtmMasterServer
gtm_port = gtmMasterPort
gtm_connect_retry_interval = 1
EOF
$
```

Do for all the gtm proxies you're configuring

6. Start GTM proxy

```
$ ssh gtm_slave_host gtm_ctl start -Z gtm -D gtm_slave_dir
$
```

Do for all the gtm proxies you're configuring

GTM failure handling steps (4)

8. Promote GTM slave to master

```
$ ssh gtm_slave_host gtm_ctl promote -Z gtm -D gtm_slave_dir
$
```

Reconfigure for further restart

```
$ ssh gtm_slave_host cat >> gtm_slave_dir/gtm.conf << EOF
startup = STANDBY
EOF
$
```

GTM failure handling steps (5)

9. Reconnect each GTM-Proxy to the new Master

```
$ ssh gtmProxyServer gtm_ctl reconnect -Z gtm_proxy \
  -D gtmProxyDir -o "-s gtmMasterServer -t gtmMasterPort"
$
```

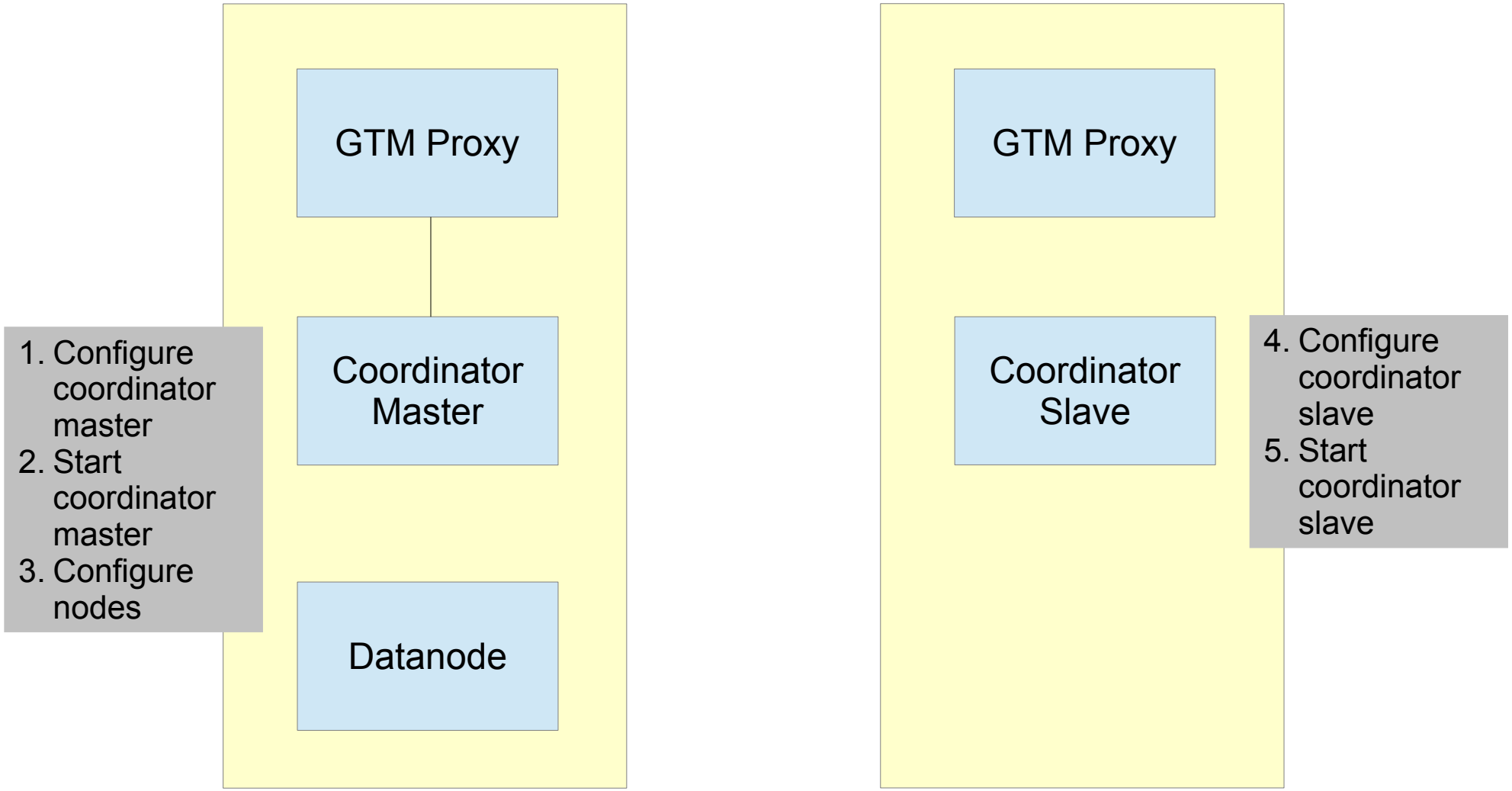
Do for all the gtm proxies you're configuring

Reconfigure for further restart

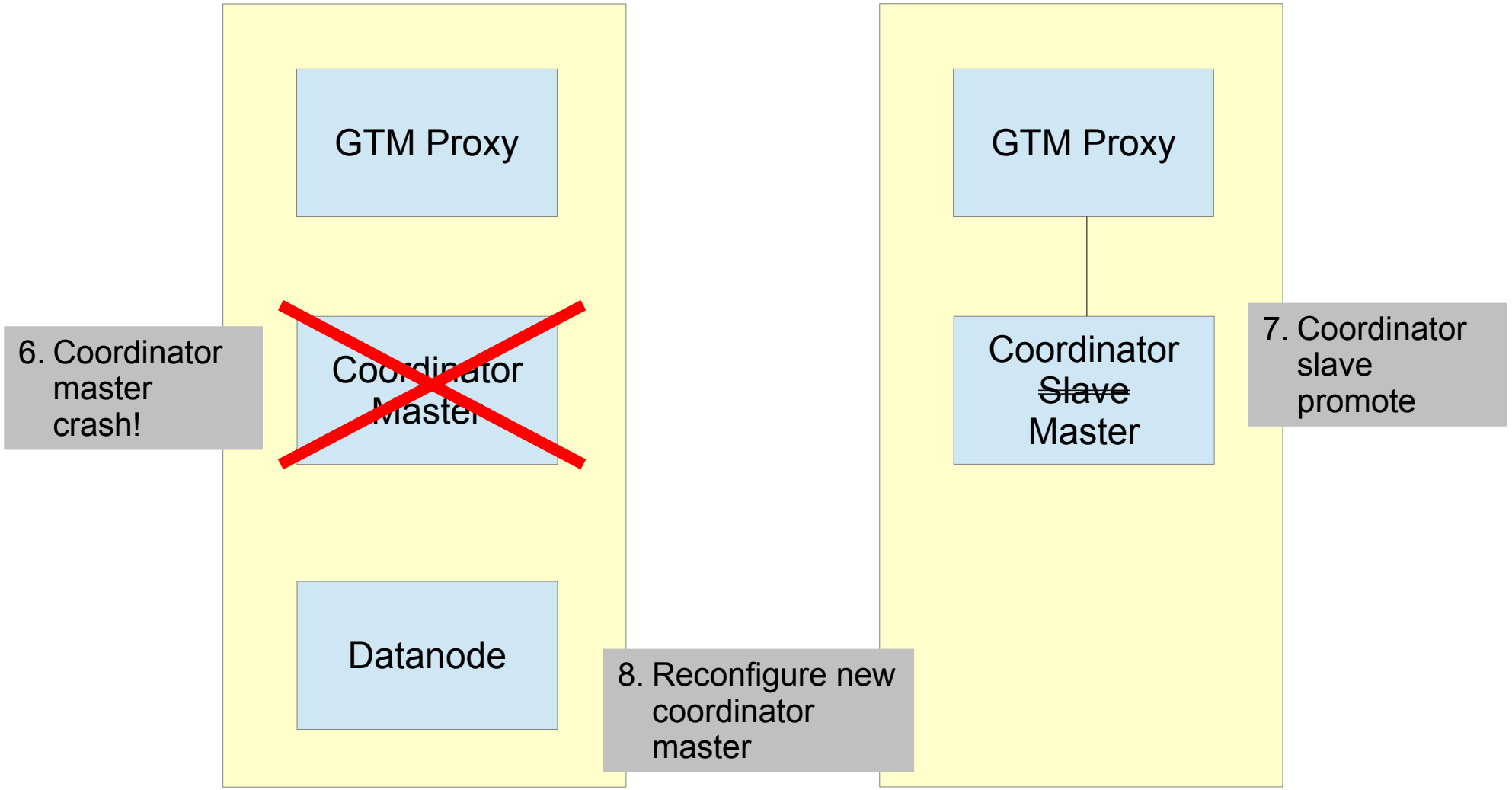
```
$ ssh gtmProxyServer cat >> gtmProxyDir/gtm_proxy.conf << EOF
gtm_host = 'gtmMasterServer'
gtm_port = gtmMasterPort
EOF
$
```

Do for all the gtm proxies you're configuring

Failure Handling Outline - Coordinator(1)



Failure Handling Outline - Coordinator(2)



Coordinator failure handling steps (1)

1. Configure coordinator masters

```
$ ssh coordMasterServer initdb --nodename coordName -D coordMasterDir
$ ssh coordMasterServer “cat >> coordMasterDir/postgresql.conf” << EOF
listen_addresses = 'listen_addresses'
port = coordPort
pooler_port = poolerPort
gtm_host = 'GTMProxyHost'
gtm_port = GTMProxyPort
wal_level = hot_standby
archive_mode = on
archive_command = 'rsync %p CoordSlaveServer:coordArchLogDir/%f'
max_wal_senders = coordMaxWalSender
EOF
$ ssh coordMasterServer cat >> coordMasterDir/pg_hba.conf << EOF
host replication pgxcOwner CIDR-addresses trust
EOF
$
```

Do for all the coordinators you're configuring

Coordinator failure handling steps (2)

2. Start coordinator masters

```
$ ssh coordMasterServer pg_ctl start -Z coordinator -D coordMasterDir
```

Do for all the coordinators you're configuring

3. Configure nodes

```
$ psql -p coordPort -h coordMasterServer postgres
CREATE NODE …
CREATE NODE …
ALER NODE …
\q
$
```

All other coordinators/datanodes

Myself

Do for all the coordinators you're configuring

Coordinator failure handling steps (3)

4. Configure coordinator slaves

```
$ ssh coordSlaveServer pg_basebackup -p coordPort -h coordMasterServer \
  -D coordSlaveDir -x
$ ssh coordSlaveServer “cat >> coordSlaveDir/recovery.conf”  <EOF
standby_mode = on
primary_conninfo = 'host = coordMasterServer port = coordPort user = pgxcOwner
application_name = 'coordName'
restore_command = 'cp coordArchLogDir/%f %p'
archive_cleanup_command = 'pg_archivecleanup coordArchLogDir %r'
EOF
$ ssh coordSlaveServer "cat >> coordSlaveDir/postgresql.conf" <<EOF
hot_standby = on
port = coordPort
EOF
```

Do for all the coordinators you're configuring

5. Start coordinator slaves

```
$ ssh coordSlaveServer pg_ctl start -Z coordinator -D coordSlaveDir
$ ssh coordMasterServer “cat >> coordMasterDir/postgresql.conf”  <EOF
synchronous_commit = on
synchronous_standby_names = 'coordName'
EOF
$ ssh coordSlaveServer "cat >> coordSlaveDir/postgresql.conf" <<EOF
hot_standby = on
port = coordPort
EOF
$ ssh coordMasterServer pg_ctl reload -Z coordinator -D coordMasterDir
$
```

Do for all the coordinators you're configuring

Coordinator failure handling steps (4)

7. Coordinator slave promote

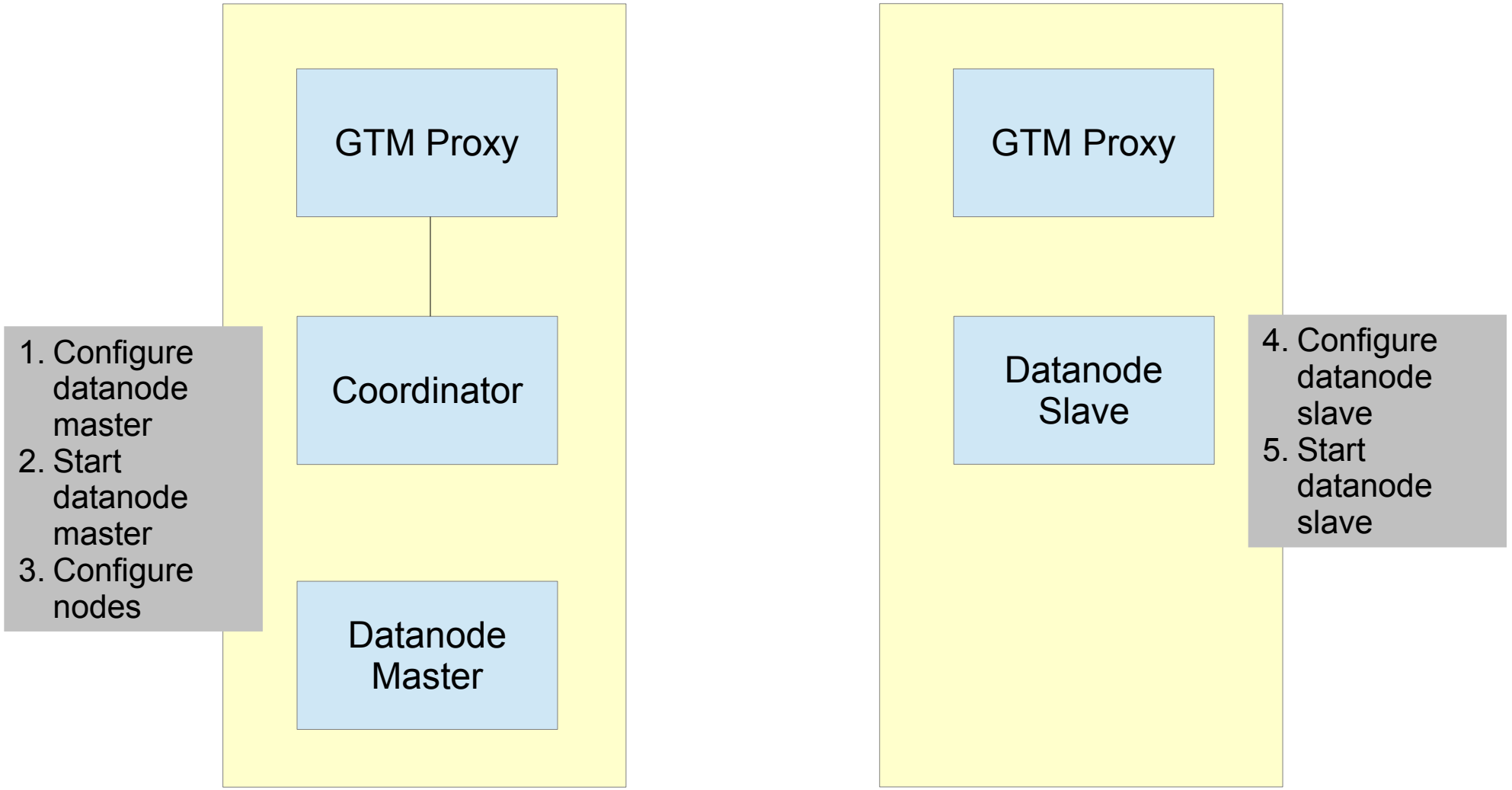
```
$ ssh coordSlaveServer pg_ctl promote -Z coordinator -D coordSlaveDir
$ ssh coordSlaveServer "cat >> coordSlaveDir/postgresql.conf" <<EOF
gtm_host = 'targetGTMhost'
gtm_port = targetGTMport
EOF
$ ssh coordSlaveServer pg_ctl restart -Z coordinator -D coordSlaveDir
$
```

8. Reconfigure new coordinator master

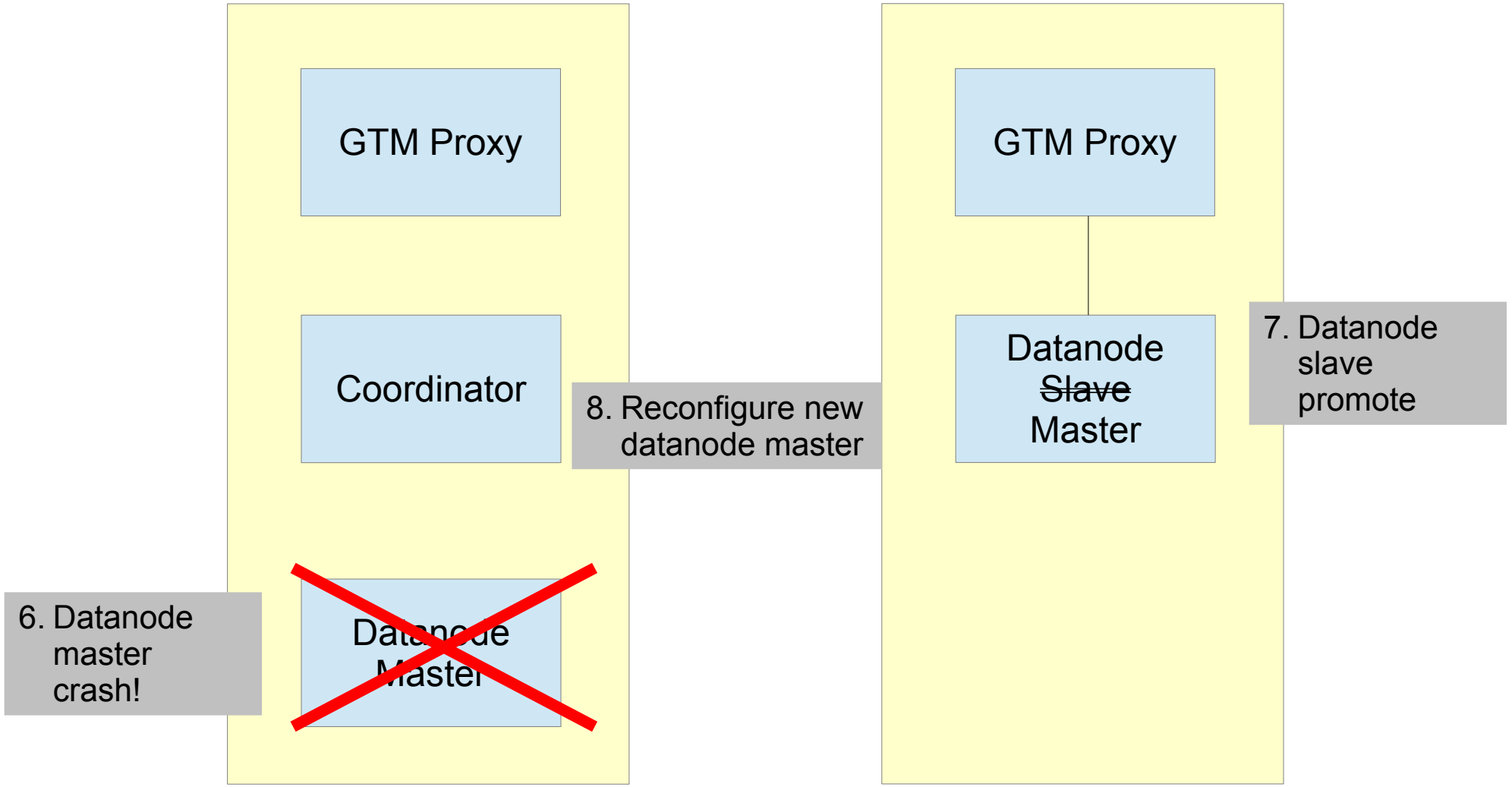
```
$ cat > cmdfile <<EOF
ALTER NODE coordName WITH (HOST='coordSlaveServer', PORT=coordPort);
SELECT pgxc_pool_reload();
\q
EOF
$ psql -p coordPort -h coordMasterServer dbname pgxcOwner -f cmdfile
$
```

Do for all coordinators

Failure Handling Outline - Datanode(1)



Failure Handling Outline - Coordinator(2)



Datanode failure handling steps (1)

1. Configure Datanode masters

```
$ ssh datanodeMasterServer initdb --nodename datanodeName -D datanodeMasterDir
$ ssh datanodeMasterServer "cat >> datanodeMasterDir/postgresql.conf" << EOF
listen_addresses = 'listen_addresses'
port = datanodePort
pooler_port = poolerPort
gtm_host = 'GTMProxyHost'
gtm_port = GTMProxyPort
wal_level = hot_standby
archive_mode = on
archive_command = 'rsync %p datanodeSlaveServer:datanodeArchLogDir/%f'
max_wal_senders = datanodeMaxWalSender
EOF
$ ssh datanodeMasterServer cat >> datanodeMasterDir/pg_hba.conf << EOF
host replication pgxcOwner CIDR-addresses trust
EOF
$
```

Do for all the coordinators you're configuring

Datanode failure handling steps (2)

2. Start datanode masters

```
$ ssh datanodeMasterServer pg_ctl start -Z datanode -D datanodeMasterDir
```

Do for all the datanodes you're configuring

3. Configure nodes – Has already been done!

Datanode failure handling steps (3)

4. Configure datanode slaves

```
$ ssh datanodeSlaveServer pg_basebackup -p datanodePort -h datanodeMasterServer \
  -D datanodeSlaveDir
$ ssh datanodeSlaveServer “cat >> datanodeSlaveDir/recovery.conf”  <EOF
standby_mode = on
primary_conninfo = 'host = datanodeMasterServer port = datanodePort user = pgxcOwner
application_name = 'datanodeName'
restore_command = 'cp datanodeArchLogDir/%f %p'
archive_cleanup_command = 'pg_archivecleanup datanodeArchLogDir %r'
EOF
$ ssh datanodeSlaveServer "cat >> datanodeSlaveDir/postgresql.conf" <<EOF
hot_standby = on
port = datanodePort
EOF
```

Do for all the datanodes you're configuring

5. Start datanode slaves

```
$ ssh datanodeSlaveServer pg_ctl start -Z datanode -D datanodeSlaveDir
$ ssh datanodeMasterServer “cat >> datanodeMasterDir/postgresql.conf”  <EOF
synchronous_commit = on
synchronous_standby_names = 'coordName'
EOF
$ ssh datanodeSlaveServer "cat >> datanodeSlaveDir/postgresql.conf" <<EOF
hot_standby = on
port = datnodePort
EOF
$ ssh datanodeMasterServer pg_ctl reload -Z datanode -D datanodeMasterDir
$
```

Do for all the datanodes you're configuring

Datanode failure handling steps (4)

7. Datanode slave promote

```
$ ssh datanodeSlaveServer pg_ctl promote -Z datanode -D datanodeSlaveDir
$ ssh datanodeSlaveServer "cat >> datanodeSlaveDir/postgresql.conf" <<EOF
gtm_host = 'targetGTMhost'
gtm_port = targetGTMport
EOF
$ ssh datanodeSlaveServer pg_ctl restart -Z datanode -D datanodeSlaveDir
$
```

8. Reconfigure new datanode master

```
$ cat > cmdfile <<EOF
ALTER NODE datanodeName WITH (HOST='datanodeSlaveServer', PORT=datanodePort);
SELECT pgxc_pool_reload();
\q
EOF
$ psql -p coordPort -h coordMasterServer dbname pgxcOwner -f cmdfile
$
```

Do for all the coordinators

When a Server is Gone?

- Both Coordinator and Datanode fail
  - They have to be failed-over
  - Combination of the cases above
  - Don't need to fail-over GTM-Proxy
    - Use it running at the target server.

Complicated Scripts ...

- Tools will be available soon from separate team
  - RA for Pacemaker
  - Installation/configuration
  - Backup/Restore
  - Cluster-wide stats info
- Now writing pgxc_ctl scripts
  - Various useful bash scripts for various scenes
    - Deployment/configuration/start/stop/failover
    - Configuration
    - Reconfigure at fault handling
    - Log (hopefully)
    - Cluster operation commands
    - Export current status for backup (hopefully)
  - Only works at linux/bash
  - Useful to write scripts for other HA middleware and operating system

Coming soon too!!

Postgres-XC Core Status

- Version 1.0.1: Sept., 2012
  - Based upon PostgreSQL 9.1
  - Major SQL statements
  - GTM standby (slave)
  - Coordinator/Datanode streaming replication

October 24th, 2012

Postgres-XC Core Development Underway

- Trigger
- Returning
- WHERE CURRENT OF
- Online node management
  - Node addition/removal
- Table redistribution
  - Foreground
  - Concurrent
- Utilities
  - GTM stats/housekeeping
- Planner improvement
- Error handling improvement
  - Features out of transaction block in PG (could not handle using 2PC)
  - Error handling from multiple datanodes in single operation

October 24th, 2012

Postgres-XC release cycle

- Annual major release
  - Merge PostgreSQL major release
  - Additional Feature
  - Next release will be around April, 2013
- Quarterly minor release
  - Bug
  - Security
  - Catch up PostgreSQL minor releases

Join the Community

- As a tester - bug report
- As a packager
  - Debian, RPM, pkg...
- As a coder
  - Why not writing new stuff? => Feature requests in SF tracker
  - Bug correction and stabilization? => Bug tracker in SF
- As a documentation reviewer
- Anything I am forgetting here...

Resources

- Useful pages
  - http://postgres-xc.sourceforge.net/
  - https://sourceforge.net/projects/postgres-xc/
  - http://postgresxc.wikia.com/wiki/Postgres-XC_Wiki
- Mailing lists: please subscribe
  - developers
  - committers
  - bugs
  - announce
  - general

Another Postgres-XC efforts

- Installer
- Operation stats collection
- Backup/Restore
- RAs for Pacemaker/Heartbeat

  - Being done by separate development team
  - Sorry, now close effort
  - Will be made open soon

Useful Materials

- Postgres-XC
  - http://postgres-xc.sourceforge.net/ (project web site)
  - http://sourceforge.net/projects/postgres-xc/ (development site)
  - http://postgresxc.wikia.com/wiki/Postgres-XC_Wiki (Wiki site)
  - https://github.com/koichi-szk/PGXC-Tools (pgxc_ctl, gtm_util tools)
- PostgreSQL resource agents for Pacemaker/Heartbeat
  - sf-ex : mount filesystem exclusively
    - https://github.com/ClusterLabs/resource-agents/blob/master/heartbeat/sfex
  - postgres – streaming replication
    - https://github.com/ClusterLabs/resource-agents/blob/master/heartbeat/pgsql

Thank you very much!!

NTT

EnterpriseDB®
The Enterprise PostgreSQL Company

NTT DATA