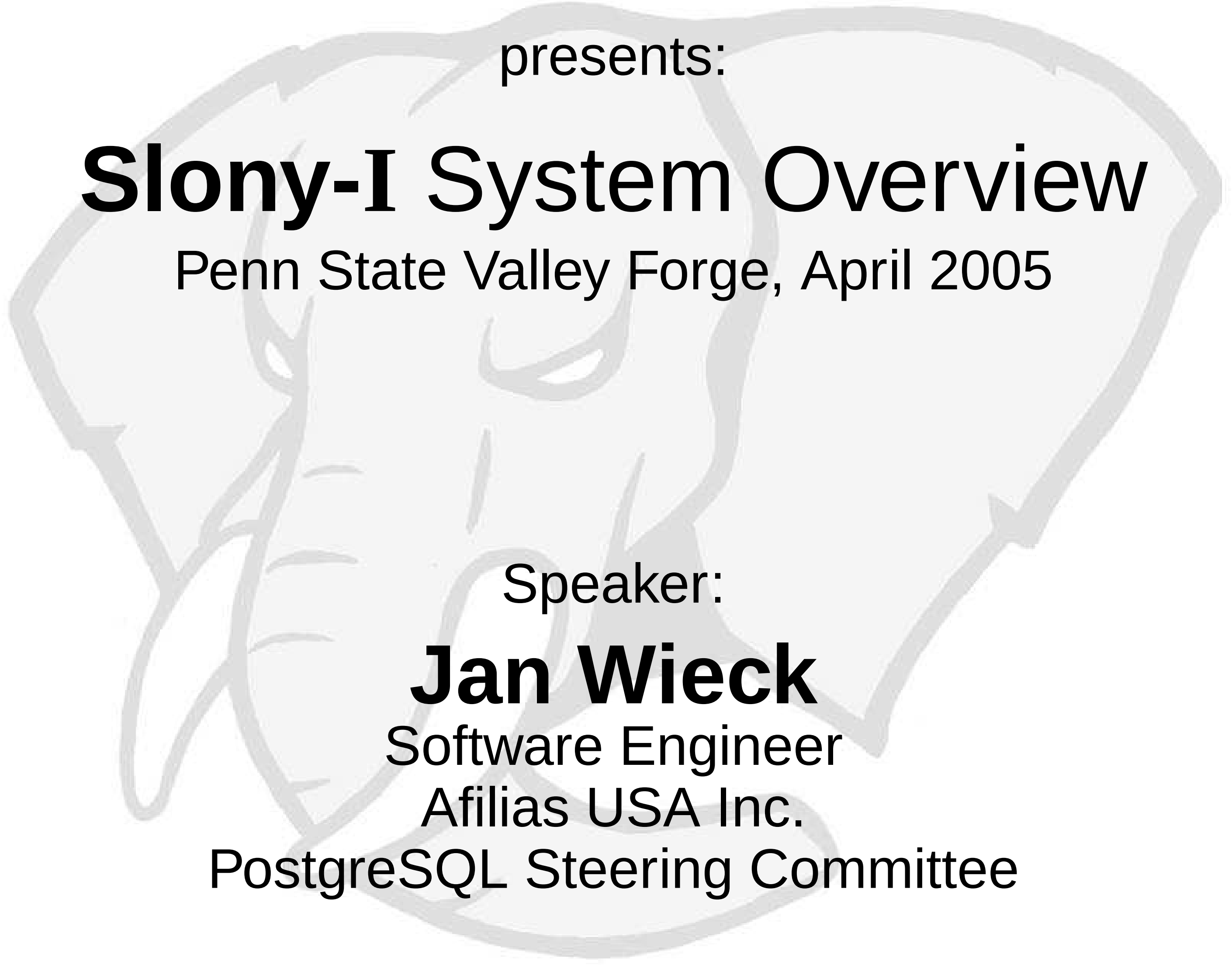
Afilias GLOBAL REGISTRY SERVICES



presents:

Slony-I System Overview

Penn State Valley Forge, April 2005

Speaker:

Jan Wieck

Software Engineer
Afilias USA Inc.
PostgreSQL Steering Committee

Agenda

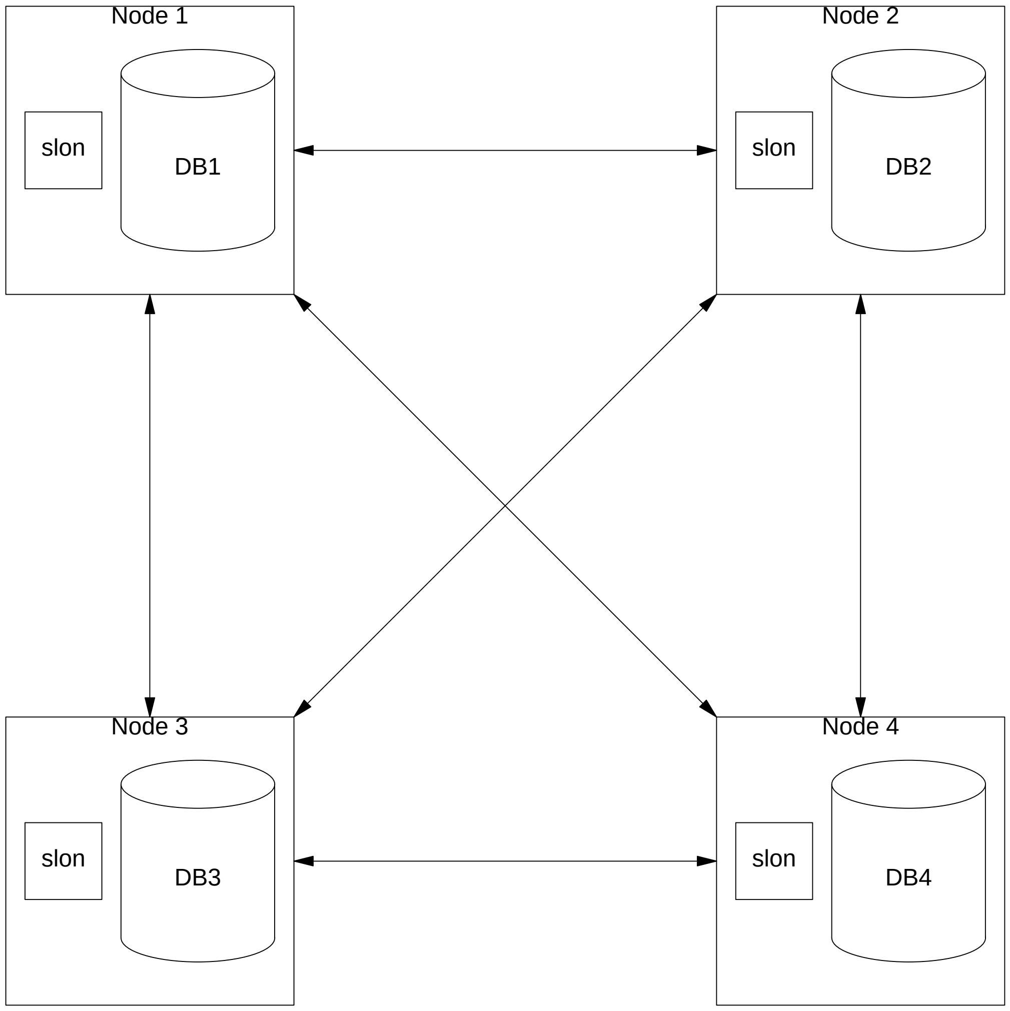
- The node network - building a Slony-I cluster
  What is a node
  A simple node network
  Listening for events
  Event flow
- Subscribing to sets
  Creating the initial copy of the database
  Cascaded subscriptions
- Replicating data
  Splitting the update stream into chunks
- Advanced features
  Provider change
  Switchover
  Failover
- Slony-I and Pgpool
  Doing 6 hours downtime in 30 seconds
- Database schema changes
- About an integrated replication system
- The future of Slony-I

A simple node network

Tables sl_node and sl_path

A node is a combination of a database and one slon process that considers it the "local database"

In a simple configuration, all nodes have a "path" to each other. So the table sl_path will have an entry telling every slon how to connect to every other nodes database. These connections are only established when needed.



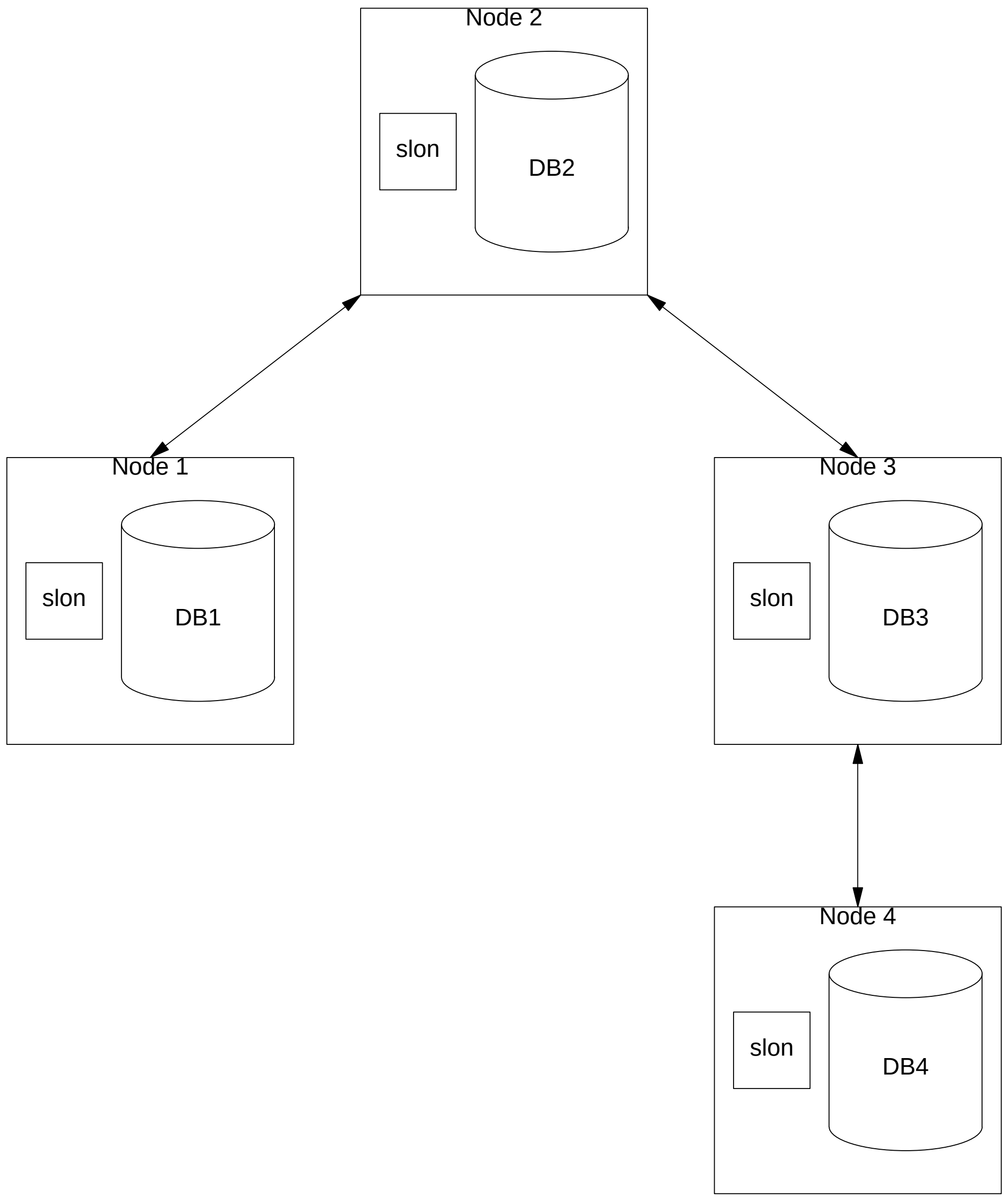
Listening for events

While replicating data is the final goal, it is not the first thing to get working. To understand the advanced features of the Slony-I replication system it is important not to associate any particular node with a fixed "master" or "slave" role.

Table sl_listen

The entries in sl_listen control the logical flow of events.

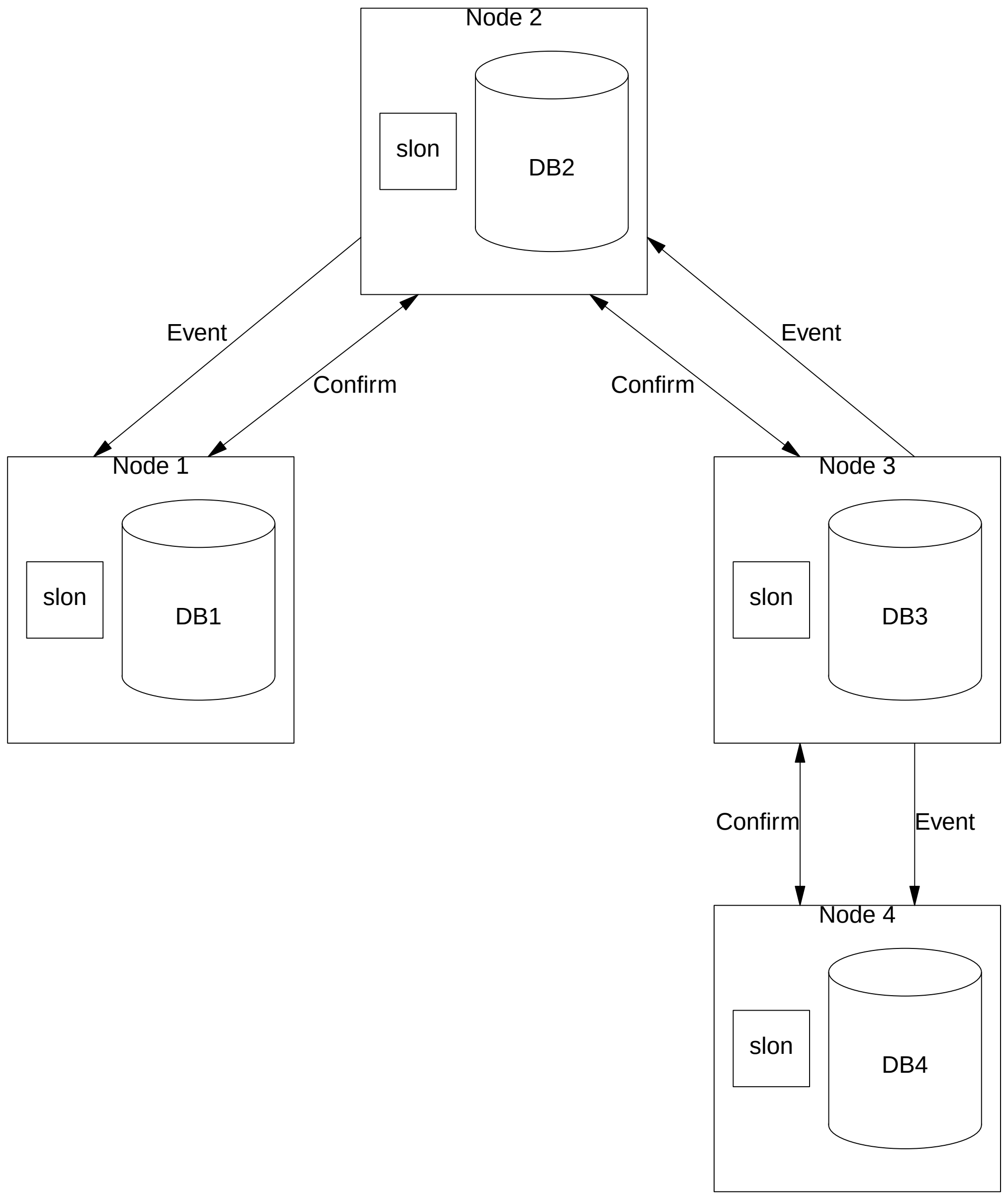
sl_origin	sl_receiver	sl_provider
1	2	1
1	3	2
1	4	3
2	1	2
2	3	2
2	4	3
3	1	2
3	2	3
3	4	3
4	1	2
4	2	3
4	3	4



Event flow

Tables sl_event and sl_confirm

- Event happens on Node 3
- Nodes 2 and 4 get notification, read the event, process and confirm it within one transaction on the local database
- Node 1 gets notification, reads event on 2, processes and confirms.
- When the event processing transactions on nodes 1, 2 and 4 commit, the remote listen threads get notified and propagate the confirmation.
- Periodically the cleanup thread checks for events that are confirmed by all other known nodes and removes them (including the replication data that belongs to them).



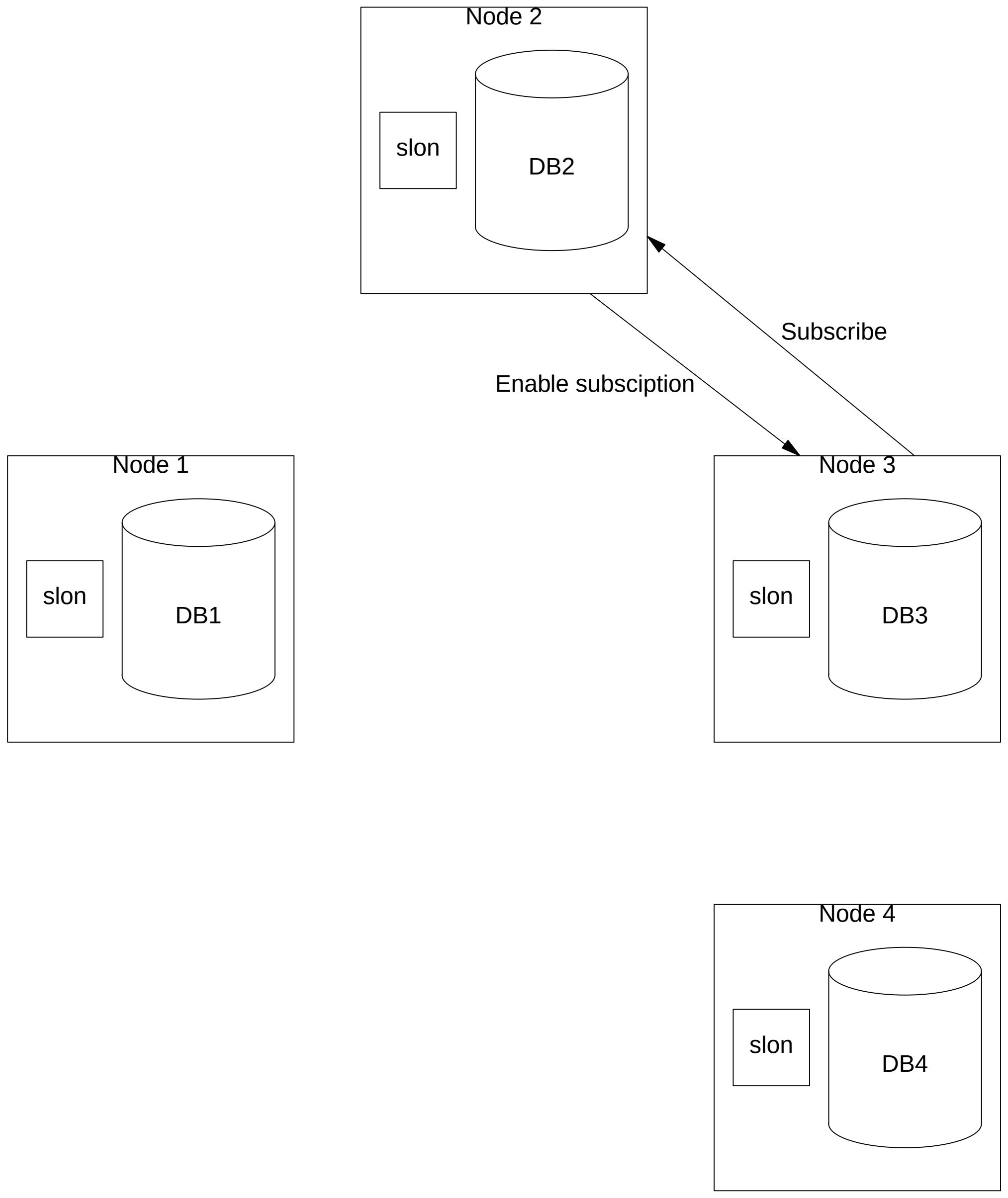
Sets and subscribing

Database objects (tables and sequences) are organized in sets. To start replicating data, Slony-I needs to copy an initial snapshot of the set from the provider node to the subscriber.

The following assumes that Node 2 is the origin of a set.

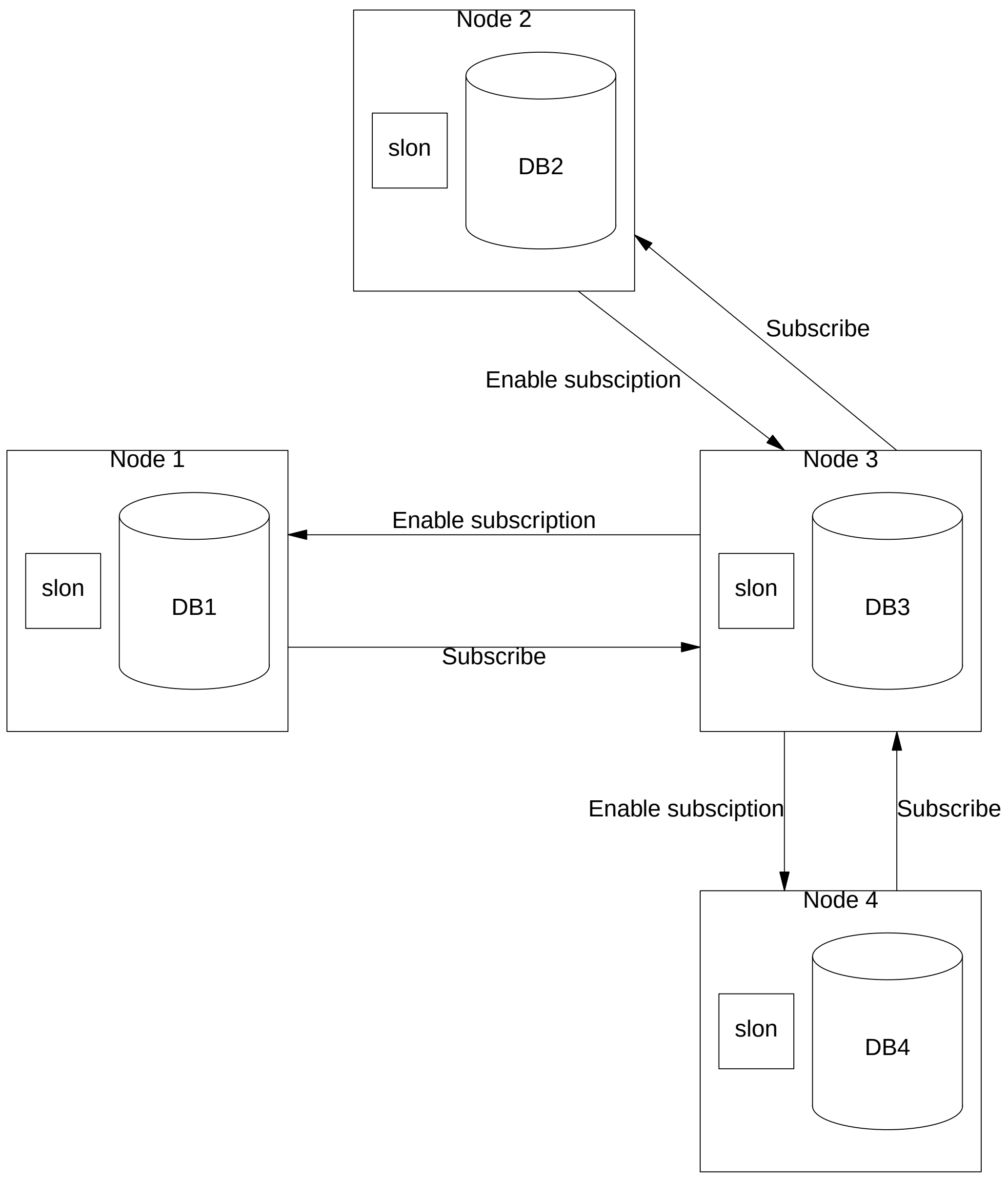
- The SUBSCRIBE_SET event is generated on Node 3. As usual the event propagates to all nodes.
- When receiving the event, Node 2 generates the ENABLE_SUBSCRIPTION event in return.
- When Node 3 processes the enable event, it copies over the sl_table and sl_sequence entries for the set, disables triggers and rules defined for the tables, adds a protective trigger that denies user application updates, copies over each of the tables data and remembers the exact transaction status of the point when the set got copied.

That the activation of the subscription starts from the origin is important. All other forwarding subscribers must know about it.



Cascaded subscription

To keep the initial set copy IO off the origin, which is usually the main DB server, every other subscriber can be instructed to copy the data from an existing subscriber, acting as data provider for the new node. This requires that the provider has a forwarding subscription that is active.



Replicating data

A SYNC event is basically like every other event. The originating node records the transaction state information of the serializable transaction that creates the event and the event is propagated through the node network.

After subscribing to a set and finishing the first SYNC event (the first one is a little different because the COPY happened somewhere in between two SYNC events), the subscribed sets on the replica are always replicated up to a specific SYNC event of the set origin.

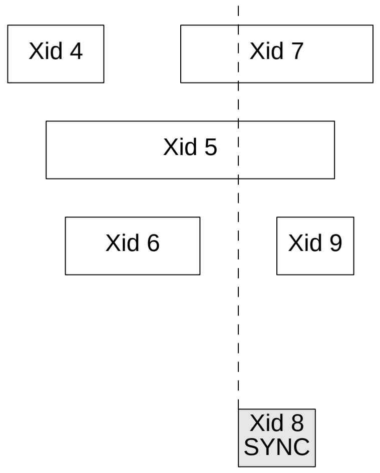
When a SYNC event arrives at a node that has set(s) subscribed that origin on the same node as the SYNC originated, it will select the delta between the current local set status, and the new events transaction information. This data is transformed into INSERT, UPDATE and DELETE statements that get executed against the local database. In addition, if the set is subscribed in forwarding mode, the selected log data is stored locally as well, so that cascaded subscribers can select it from this node as soon as they receive the event.

Replicating in small units

A serializable snapshot

The MVCC information for a serializable transaction snapshot tells which transactions logically precede or follow the current transaction. Using this information it is possible to define logical cuts into the continuous stream of overlapping transactions.

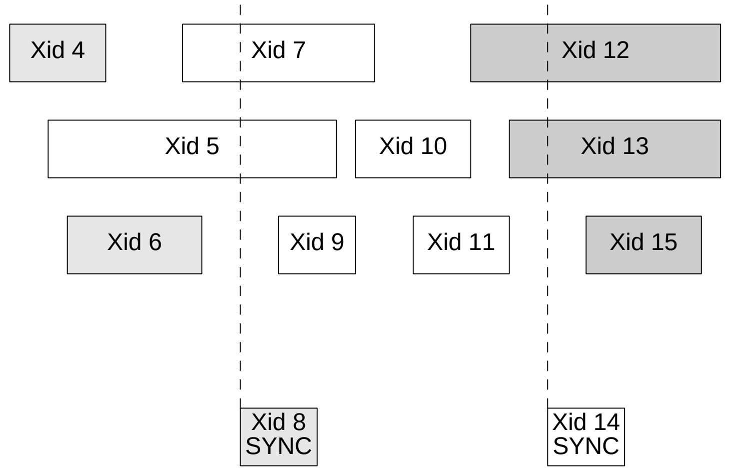
In the example on the right, the SYNC event generated in transaction 8 would remember (from the SerializableSnapshotData) that transactions 5 and 7 where still in progress when transaction 8 began.



Replicating in small units

One Chunk of replication data

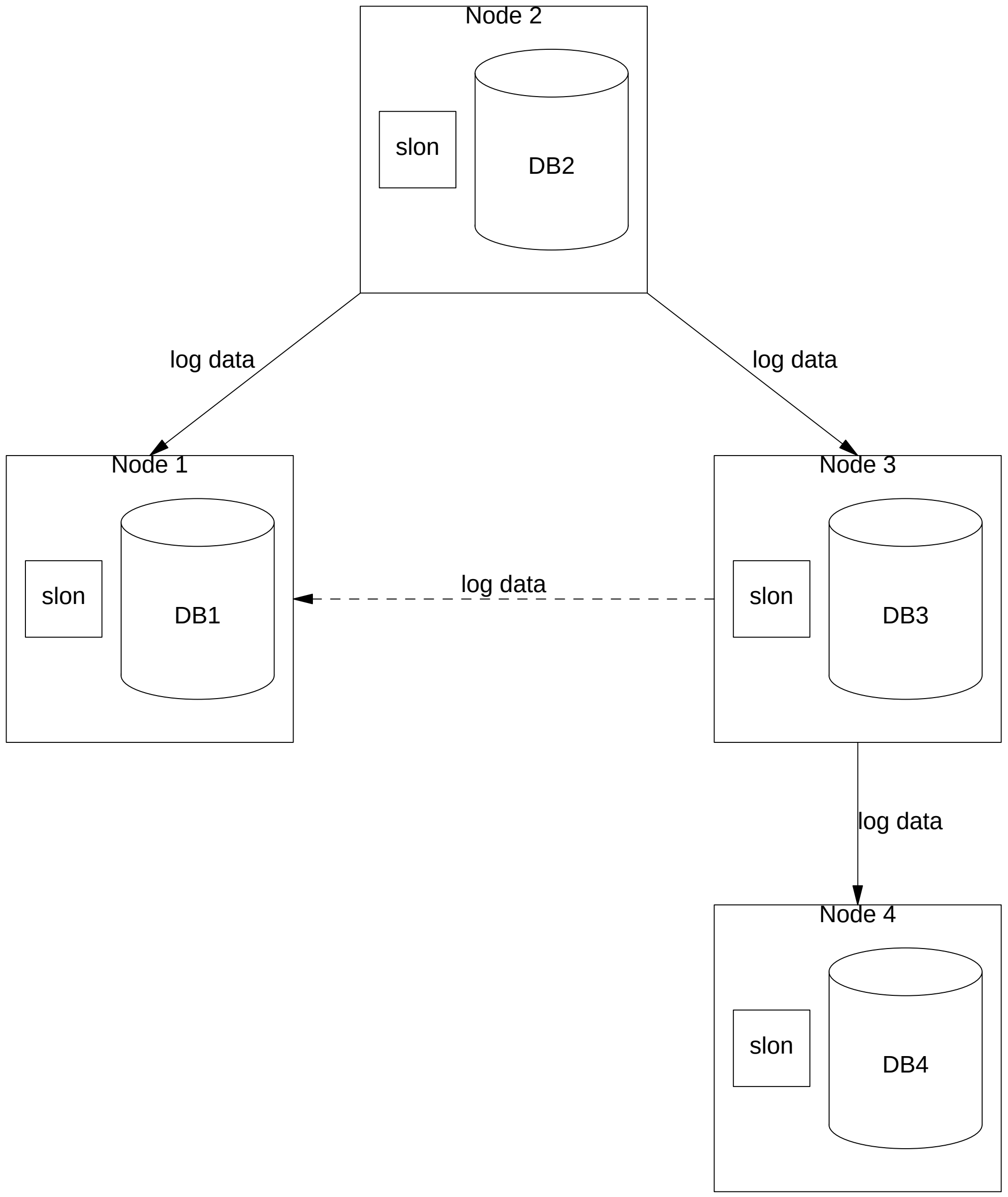
Transactions that have not been committed when one SYNC event is created belong to the following SYNC event, even if their numeric transaction ID is smaller than the one of the SYNC event itself.ID is smaller than the one of the SYNC event itself.



Provider change

Because in Slony-I the logical segmentation of the replication information is done on the origin by creating the SYNC events (every subscriber always replicates up to the transaction status of one SYNC and commits the changes to the local DB), and because all nodes that do log forwarding keep those log rows until the corresponding SYNC events have been confirmed by every subscribed node, it is easy to change the data provider.

Assuming Node 1 is subscribed and currently selecting the log data from Node 3, a "subscribe set" command will simply update the data provider information in the sl_subscribe configuration table. Without the need to rebuild the data from scratch, Node 1 becomes a replica that reads the log information directly from the origin Node 2.

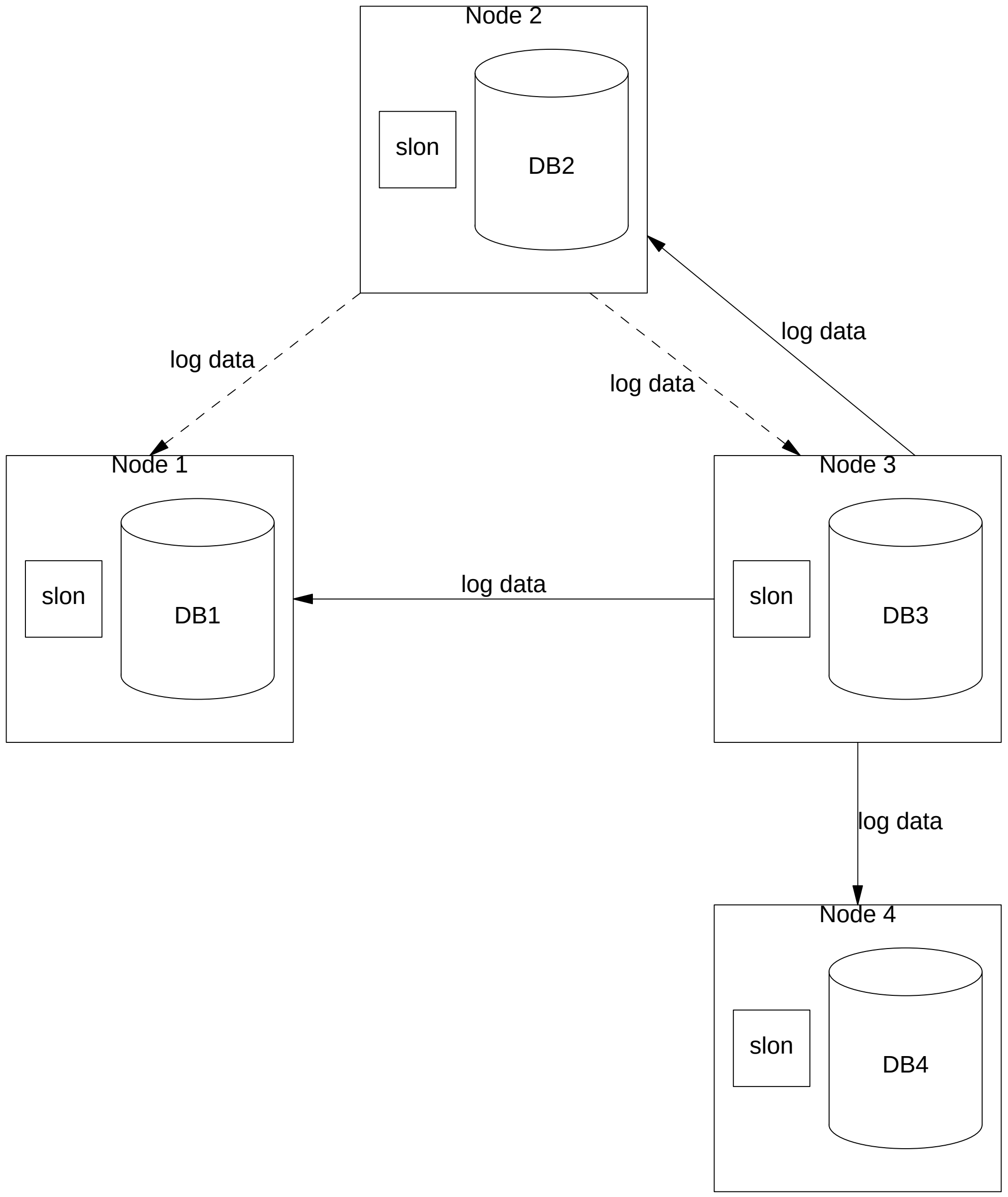


Switchover

Slony-I has a feature for controlled transfer of the origin of a set. The procedure to do so is to lock the set logically. This causes all updates to the tables contained in the set to be denied on the current origin. Then a MOVE_SET event is issued which transfers the origin. The stored procedure that generates the MOVE_SET event also generates a SYNC event before, and since all events are processed by the other nodes in order it is guaranteed that at the moment the nodes consider the new node as origin, they are all replicated to that status.

On the old origin (Node 2), the MOVE_SET event causes that it becomes a subscriber. Which means that is it at the very moment the new origin (Node 3) takes over and allows for updates by the client application, it is a fully synchronized replica.

In the sample configuration on the right a maintenance shutdown of the main DB server would be possible after stopping the application, doing the "lock set", "move set", "subscribe" (instructing Node 1 to replicate against Node 3) commands, and then restarting the application now issuing updates against Node 3. This entire reconfiguration can be done within seconds.

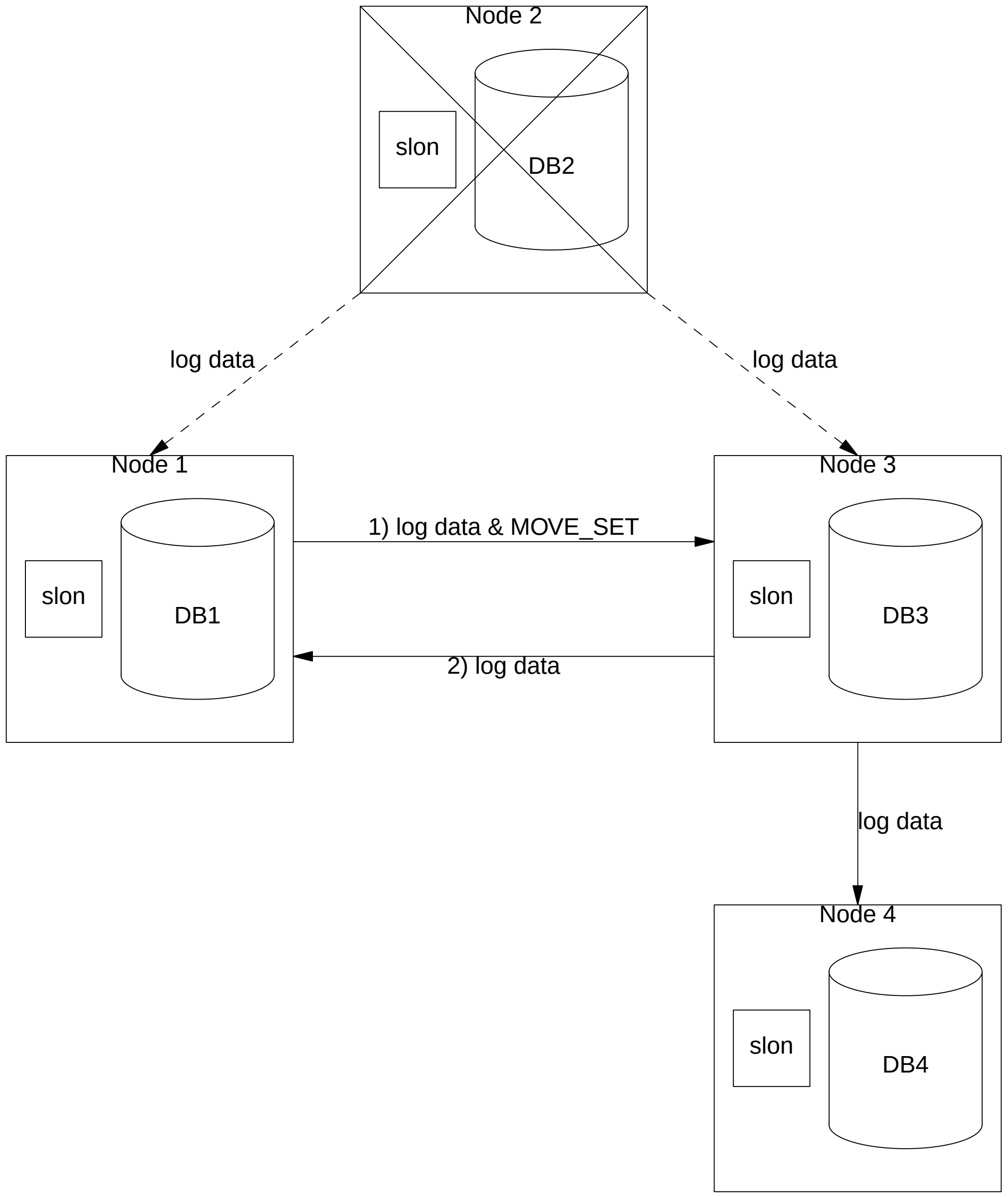


Failover

The failover is a combination of provider changes and a synthetic MOVE_SET.

Assuming that Node 3 is the designated backup server for Node 2, the situation would be very simple if Node 3 at the time Node 2 fails is the most advanced subscriber (no other node has replicated more data than Node 3).

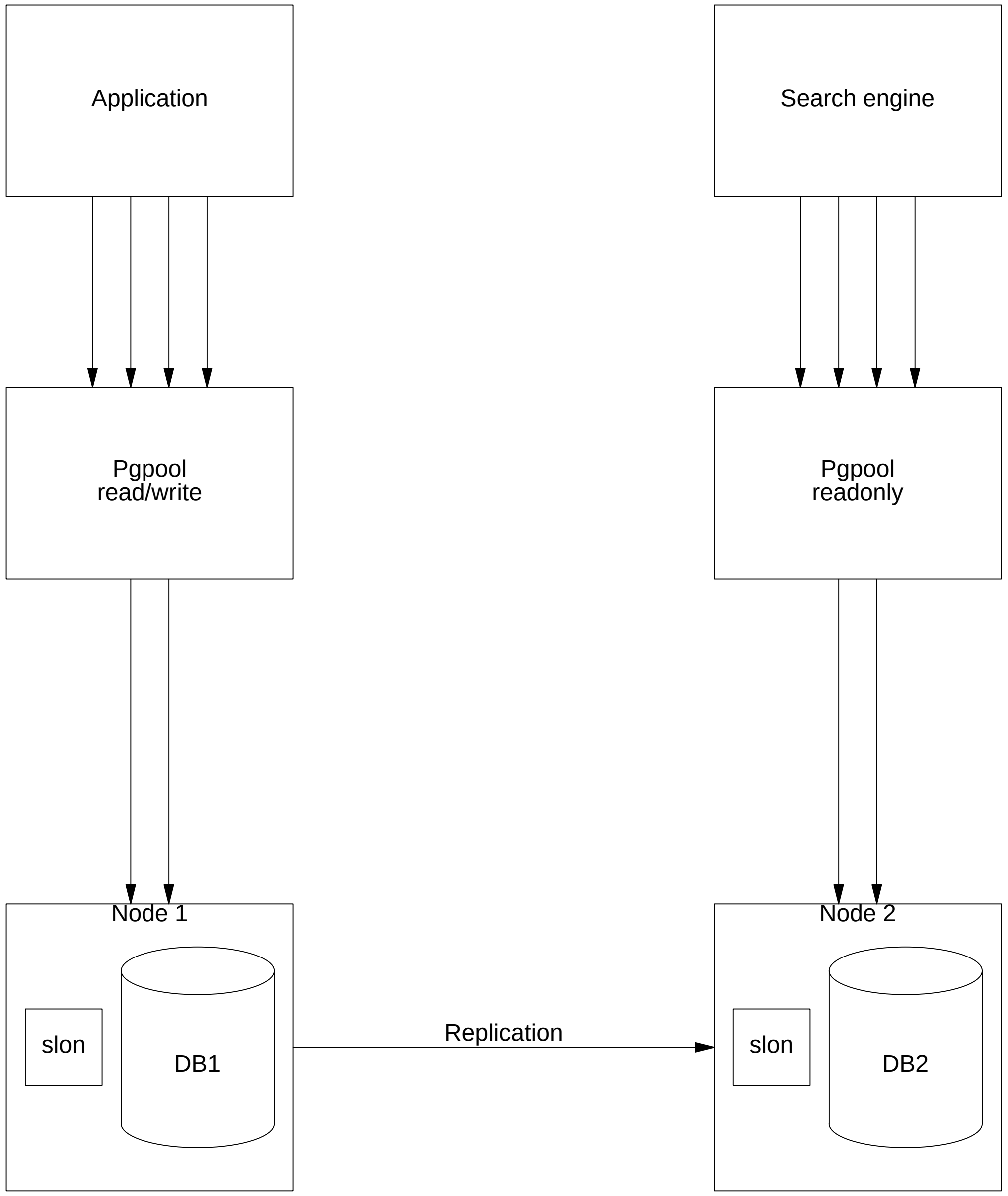
If that is not the case, the failover procedure is to stop all nodes receiving events from Node 2, determine which is the most advanced replica, change the designated backup server to use that as provider. The a synthetic MOVE_SET event, injected at the most advanced replica will cause the data to become available for update on the backup server as soon as it has caught up to the last known status of the failed server.



Slony-I and pgpool

In addition to the administrative advantages of Pgpool alone, the combination of Pgpool together with the Slony-I replication system offers unprecedented database availability for PostgreSQL.

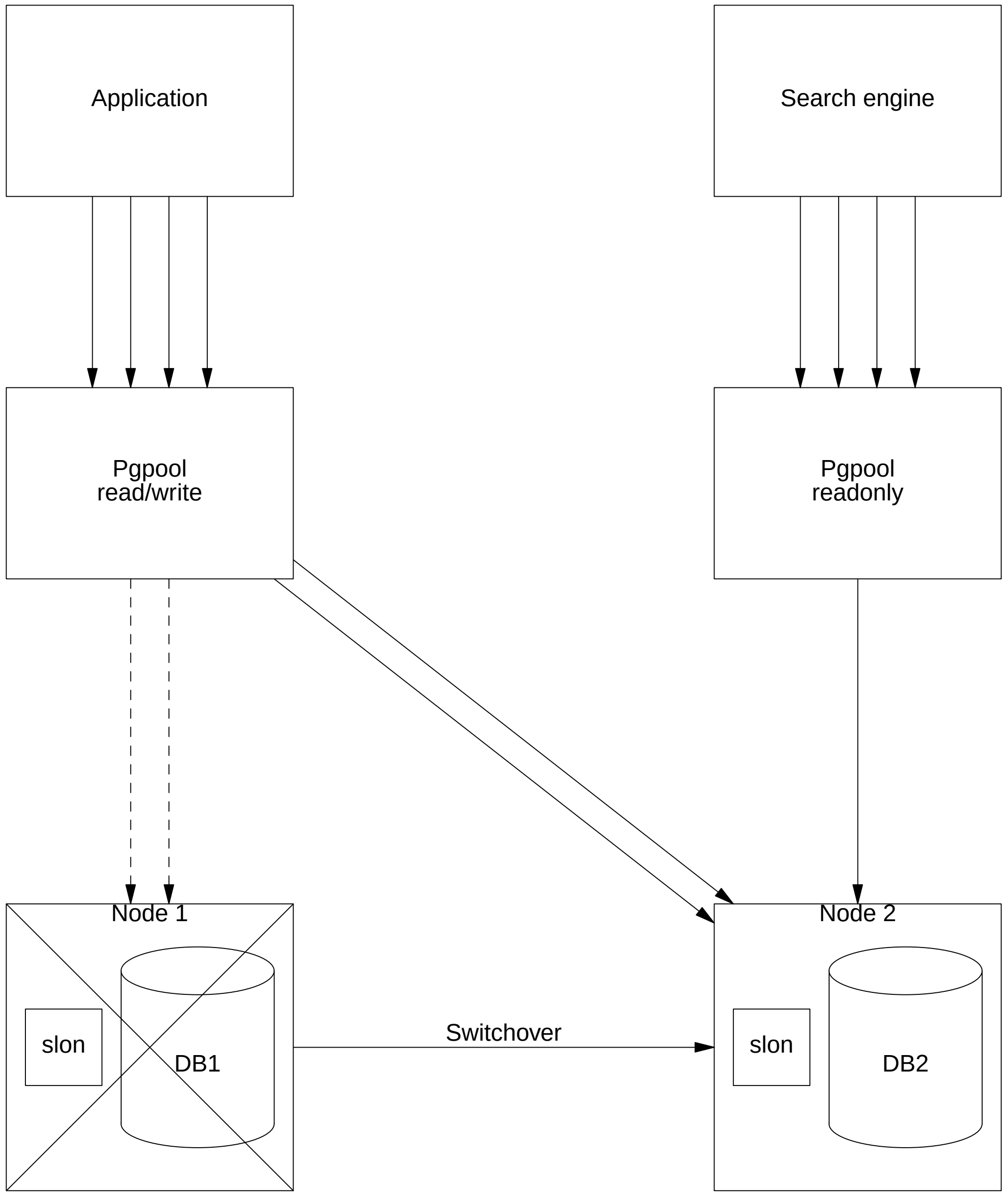
Database servers need frequent maintenance. For this, it is sometimes required to shut down the database system or even the entire server. In 24x7 installations this is known as "scheduled downtime".



Doing 6 hours of downtime in 30 seconds

To shutdown Node 1 for a scheduled maintenance shutdown, the Pgpool configuration is changed and the Slony-I system performs a switchover.

After the switchover, Node 1 is a fully synchronized subscriber. That means, it can safely be shut down and/or the maintenance tasks performed offline. When the Slony-I replication engine is restarted later, it will catch up. The Pgpool configuration changes get reverted and the data origing transferred back to Node 1 and everything is back as it was.



DB Schema changes and why automatic DDL replication is bad

Database schema changes like for example adding tables or adding columns to existing tables, require in a Slony-I replicated environment that the same operations are performed on the original and all replicas at the same logical point in time, from a transactional point of view. Otherwise the replication log data could contain data for a column that does not yet exist in the replica, or a later performed setting of a new column to default values on the replica could overwrite already replicated information.

To avoid these conflicts and allow schema modifications to be performed even in a currently updated database, Slony-I supports the execution of SQL scripts through the replication event system. This guarantees that all nodes execute the script at the same logical point in time within the transaction and event flow.

In the current version of Slony-I direct executed DDL on a replica could even lead to serious corruptions due to the way this version disables constraints, rules and user defined triggers for the target tables.

Another aspect of replicating database Schema changes is the fact that sometimes it is desirable to have different schemas on purpose. With the way PostgreSQL’s system catalog works, this would be very difficult if a replication system attempts to

About the demand for an integrated replication solution

Many users have asked for a replication solution that is integrated into the PostgreSQL database backend. Many developers have claimed that it is difficult if not impossible to keep a replication system up to date with the rapid progress made by the PostgreSQL main project.

Slony-I was designed originally to work with PostgreSQL versions 7.3 and 7.4. Version 1.0.0 of Slony-I was released in July 2004. Only 3 days after the announcement of the BETA cycle for PostgreSQL 8.0, the Slony-I team presented version 1.0.2 that is capable of replicating between 7.3, 7.4 and 8.0-BETA versions.

The future of Slony-I

- Backup and incremental backup

  A Slony-I utility will create database dumps and a replica will write SQL scripts that contain all queries the replication engine needs to perform since that dump.

- Sync pipelining

  The Slony-I engine will be allowed to open more than one cursor to increase the speed of the catch up.

- Slony-I and Pgpool even better

- Complete handover to Open Source Community