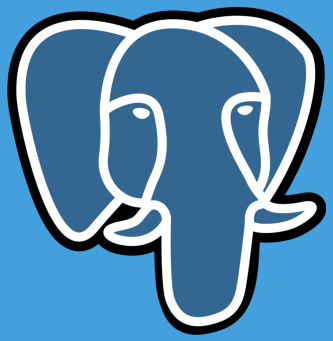
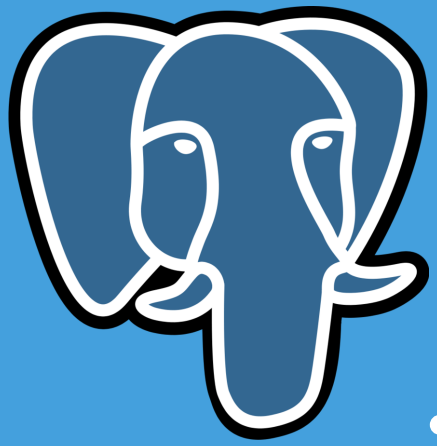


Evolution of Fault Tolerance in PostgreSQL

Gulcin Yildirim 156398, MSc

IAF0530 @ Tallinn University of Technology

18 April 2016 Tallinn

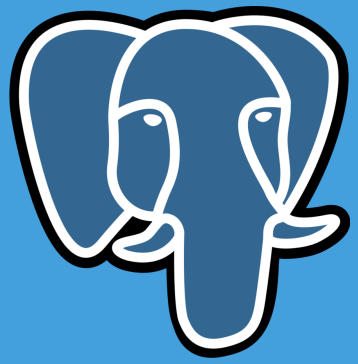


Agenda

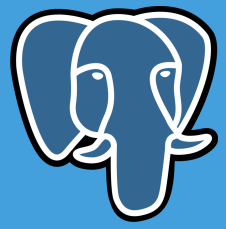
- Overview of PostgreSQL Database
- PostgreSQL Fault Tolerance: WAL
- What is Replication?
- Replication Methods for PostgreSQL
- Physical Replication
- Streaming Replication and WAL
- Managing Timeline Issues: pg_rewind
- Trigger-based Replication: Londiste, Slony
- Logical Decoding : BDR and pglogical

PostgreSQL in a nutshell (9.5)

- Advanced **open source** db system
- SQL standards compliance up to SQL:2011
- Supports different data models: relational, document (JSON and XML), and key/value (hstore extension)
- Highly extensible
- Fully ACID-compliant (atomicity, consistency, isolation, durability)
- Allows physical and logical replication
- Built-in physical and logical backup solution
- Synchronous and asynchronous transactions
- PITR (Point-in-time Recovery)
- MVCC (Multiversion concurrency control)



PostgreSQL is robust!



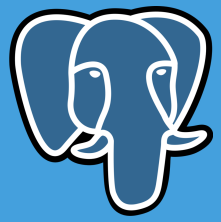
All actions on the database are performed within **transactions**, protected by a **transaction log** that will perform **automatic crash recovery** in case of software failure.

Databases may be optionally created with **data block checksums** to help diagnose **hardware faults**. Multiple **backup mechanisms** exist, with full and detailed **PITR**, in case of the need for detailed recovery. A variety of diagnostic tools are available.

Database replication is supported **natively**.

Synchronous Replication can provide greater than **"5 Nines" (99.999 percent)** availability and data protection, if properly configured and managed.

WAL Write-ahead Log

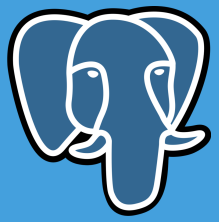


The WAL consists of a series of binary files written to the pg_xlog subdirectory of the PostgreSQL data directory.

Each change made to the database is recorded first in WAL, hence the name "**write-ahead**" log, as a synonym of "**transaction log**". When a transaction commits, the default—and safe—behaviour is to force the WAL records to disk.

Should PostgreSQL crash, the WAL will be replayed, which returns the database to the point of the last committed transaction, and thus ensures the durability of any database changes.

Transaction? Commit?

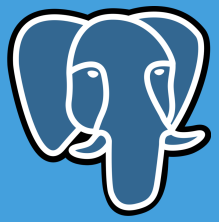


Transactions are a fundamental concept of all database systems. The essential point of a transaction is that it bundles multiple steps into a single, all-or-nothing operation.

The intermediate states between the steps are not visible to other concurrent transactions, and if some failure occurs that prevents the transaction from completing, then none of the steps affect the database at all. (PostgreSQL does not support dirty-reads.)

Database changes themselves aren't written to disk at transaction commit. Those changes are written to disk sometime later by the background writer on a well-tuned server. (WAL)

Checkpoint

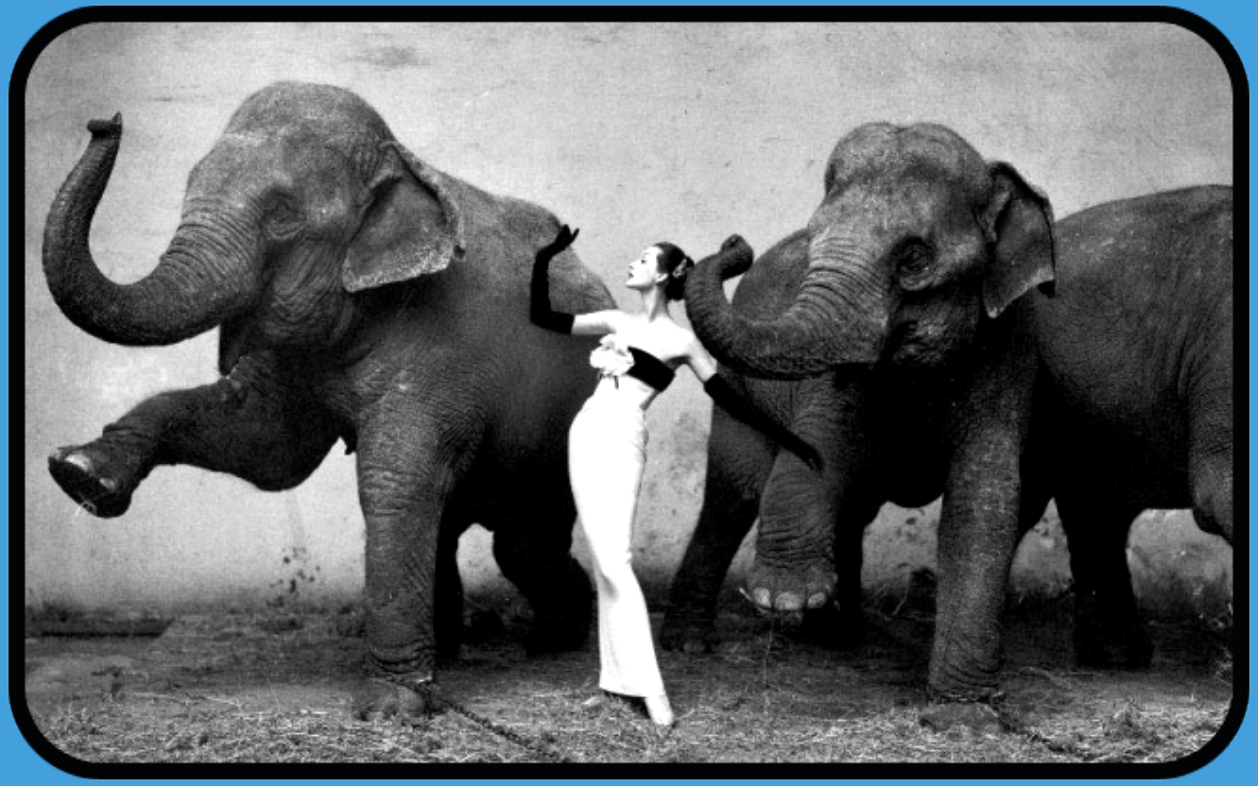


Crash recovery replays the WAL, but from what point does it start to recover?

Recovery starts from points in the WAL known as **checkpoints**. The duration of crash recovery depends on the number of changes in the transaction log since the last checkpoint. A checkpoint is a known safe starting point for recovery, since it guarantees that all the previous changes to the database have already been written to disk.

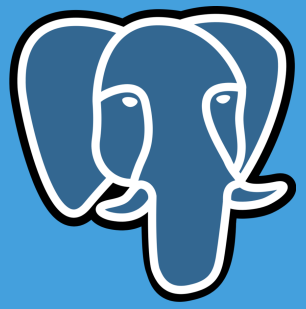
A checkpoint can be either **immediate** or **scheduled**. Immediate checkpoints are triggered by some action of a superuser, such as the CHECKPOINT command or other; scheduled checkpoints are decided automatically by PostgreSQL.

PostgreSQL Replication



Database replication is the term we use to describe the technology used to maintain a **copy** of a set of data on a **remote** system.

Postgres Replication History



- PostgreSQL 7.x (~2000)
  - Replication should not be part of core Postgres
  - Londiste - Slony (trigger based logical replication)
- PostgreSQL 8.0 (2005)
  - Point-In-Time Recovery (WAL)
- PostgreSQL 9.0 (2010)
  - Streaming Replication (physical)
- PostgreSQL 9.4 (2014)
  - Logical Decoding (changeset extraction)

Physical Replication

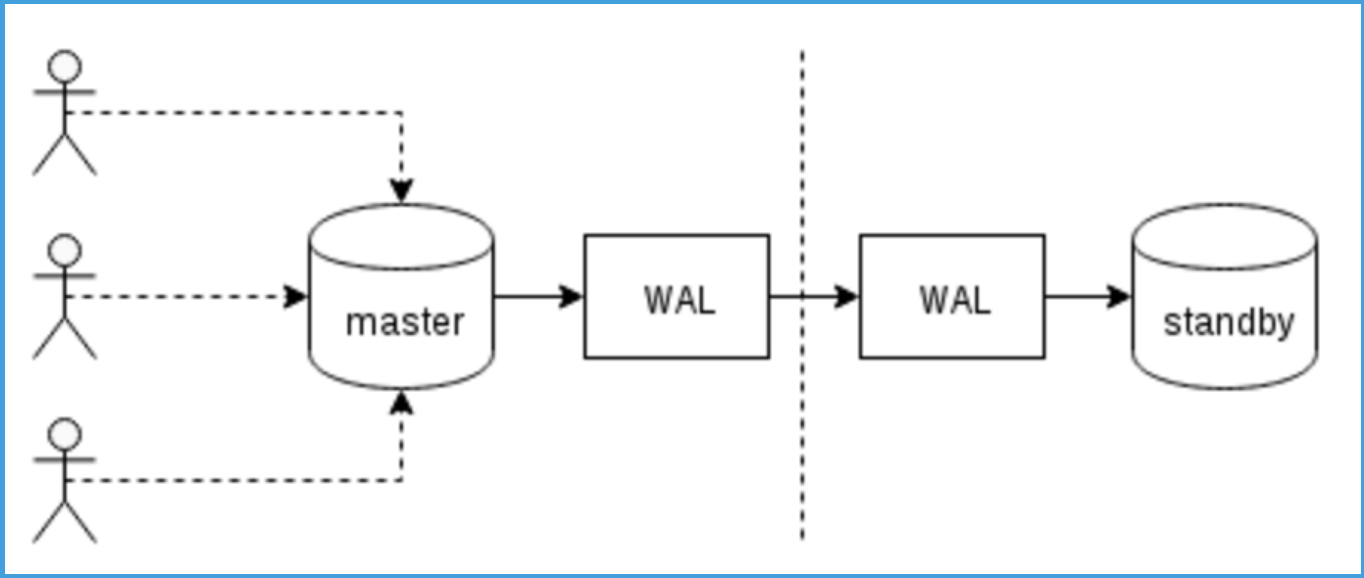
Physical Replication

The existing replication is more properly known as Physical Streaming Replication since we are streaming a series of physical changes from one node to another. That means that when we insert a row into a table we generate change records for the insert plus all of the index entries.

When we VACUUM a table we also generate change records.

Also, Physical Streaming Replication records all changes at the byte/block level, making it very hard to do anything other than just replay everything.

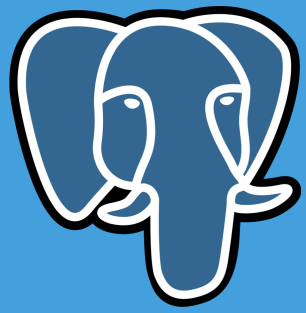
Physical Replication



WAL over network from master to standby

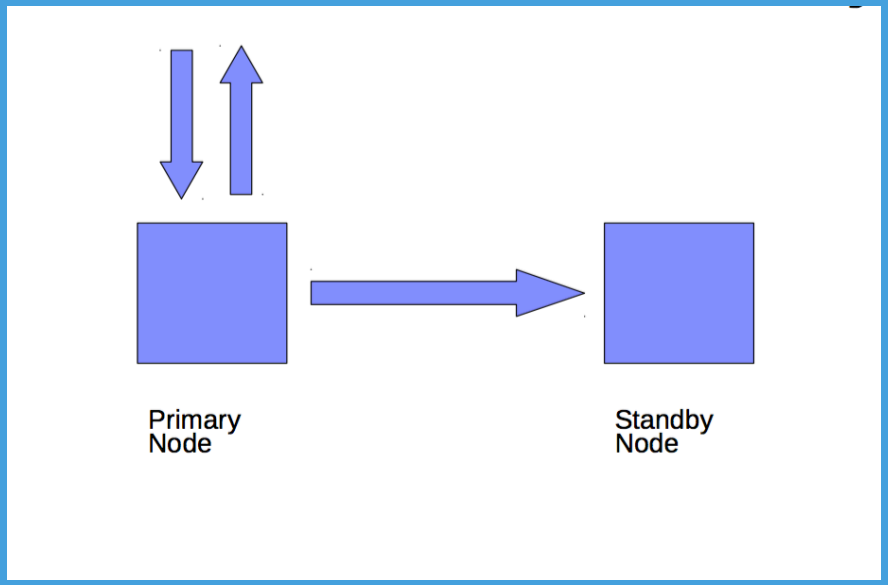
- Sending files: scp, rsync, ftp
- Streaming changes: using internal protocol (sender and receiver processes)

Standby Modes



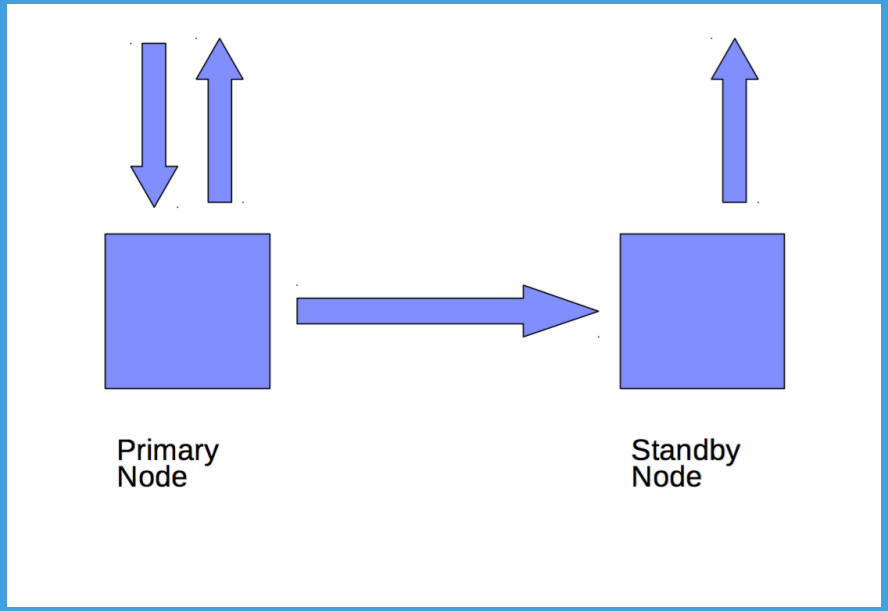
- Warm Standby
  - Can be activated immediately, but cannot perform useful work until activated
- Hot Standby
  - Node is already active
  - Read-only queries only
- Multi-Master
  - All nodes can perform read/write work

Warm Standby



Warm Standby

Hot Standby



Hot Standby

WAL and Replication

WAL Level	Suitable For
• minimal ---------->	• crash recovery
• replica ---------->	• physical replication • file-based archiving
• logical ---------->	• logical replication

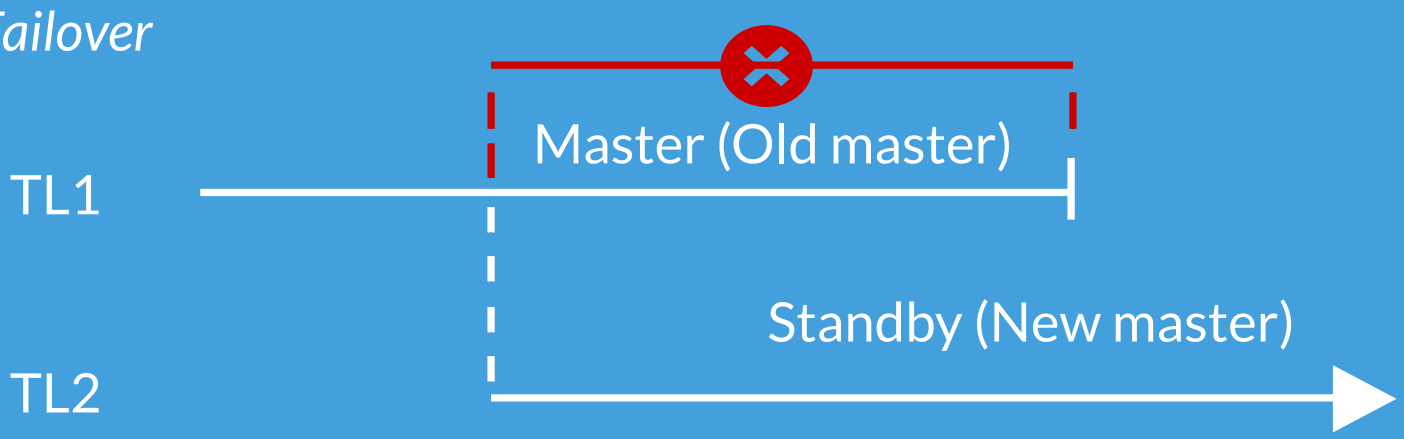
Failover and Switchover

In single-master replication, if the master dies, one of the standbys must take its place (***promotion***). Otherwise, we will not be able to accept new write transactions. Thus, the term designations, master and standby, are just roles that any node can take at some point. To move the master role to another node, we perform a procedure named **Switchover**.

If the master dies and does not recover, then the more severe role change is known as a **Failover**. In many ways, these can be similar, but it helps to use different terms for each event.

Timelines

Failover



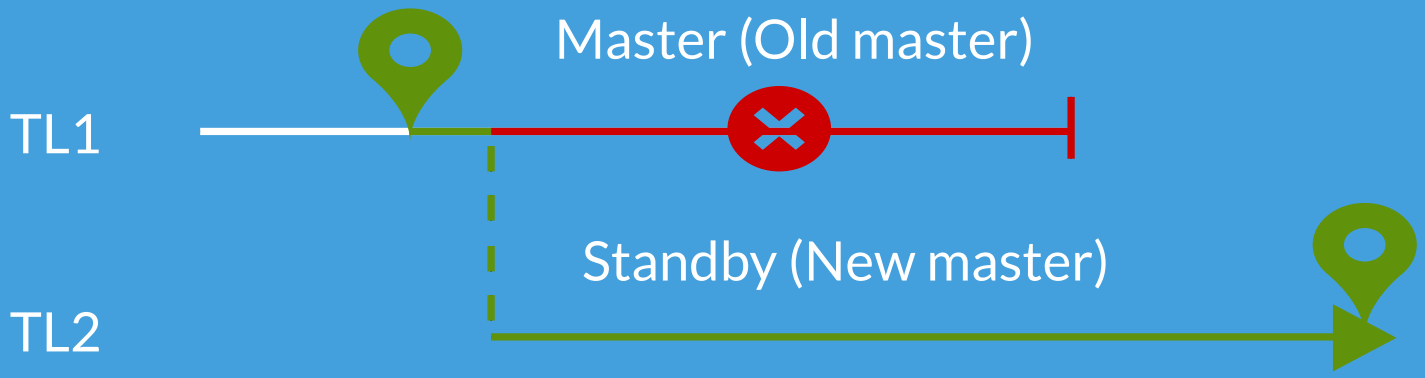
- There are outstanding changes in the old master
- Timeline increase represents new history of changes
- Changes from the old timeline can't be replayed on the servers that switched to new timeline
- The old master can't follow the new master

Timelines

Switchover

- There are no outstanding changes in the old master
- Timeline increase represents new history of changes
- The old master can become standby for the new master

pg_rewind (9.5)



- Outstanding changes are removed using data from the new master
- The old master can follow the new master

Synchronous commit

By default, PostgreSQL implements **asynchronous replication**, where data is streamed out whenever convenient for the server. As we've seen this can mean data loss in case of **failover**. It's possible to ask Postgres to require one (or more) standbys to acknowledge replication of the data prior to commit, this is called **synchronous replication** (*synchronous commit*).

With **synchronous replication**, the replication delay directly affects the elapsed time of transactions on the master. With **asynchronous replication**, the master may continue at full speed.

Synchronous replication guarantees that data is written to at least two nodes before the user or application is told that a transaction has committed.

Synchronous commit

The user can select the commit mode of each transaction, so that it is possible to have both synchronous and asynchronous commit transactions running concurrently.

This allows flexible trade-offs between performance and certainty of transaction durability.

Logical Replication

Logical Replication

Unlike **physical replication** which captures changes to the raw data on disk, the **logical replication** captures the logical changes to the individual records in database and replicates those.

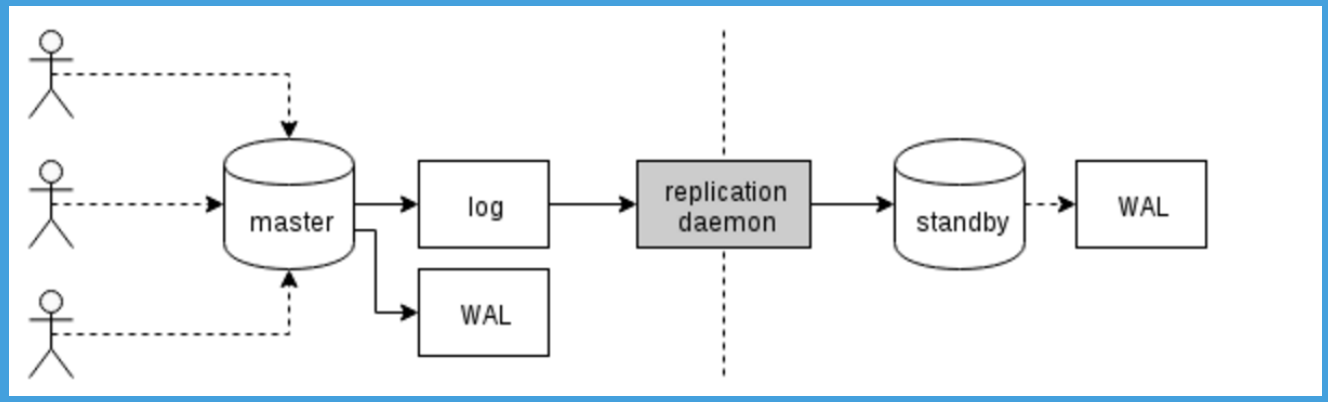
This allows for more complex replication topology than master and standby and also allows for partial replication of the database (*selective replication*) .

The logical records work across major releases, so we can use this to **upgrade** from one release to another.

There are two basic approaches to logical replication, the **trigger-based** and the **changeset extraction** (called **logical decoding** in PostgreSQL).

Trigger-based Replication

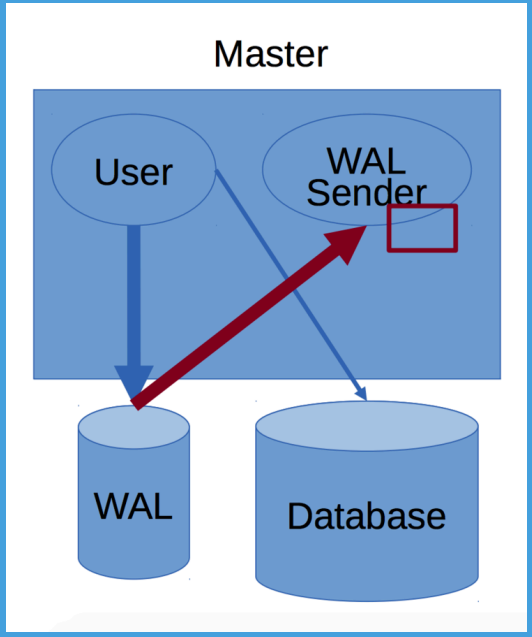
Triggered-based Replication



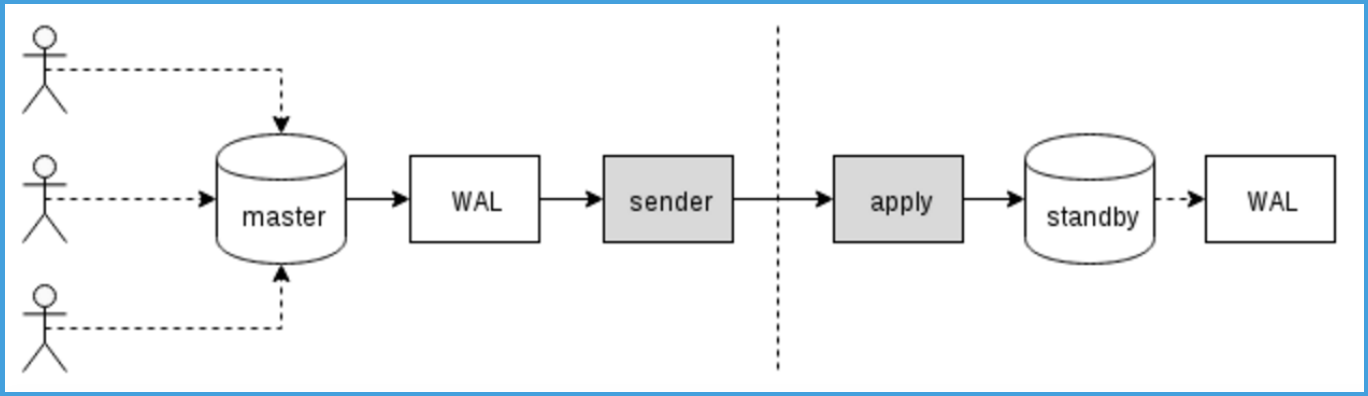
- Slony (~2004), Londiste (~2007)
- Predates the physical replication (as a result of the "no replication in core philosophy")
- Uses triggers to capture the changes to individual table
  - Increases the amount of work needed to be done for each write
- Use table(s) as queue
  - Duplicates all writes

Logical Decoding a.k.a. Changeset Extraction

- Extracts information from Write-Ahead Log into logical changes (INSERT/UPDATE/DELETE)
- Per row and commit ordered
- No write amplification
- C API for output plugin
- No DDL
- SQL Interface
- Streaming Interface



Logical Streaming Replication



- Build on top of logical decoding
- Uses same transport mechanism as streaming replication (sender & apply)
  - Allows for synchronous commit
- Currently available as extensions: **BDR** and **pglogical**
- Better performant than trigger-based replications

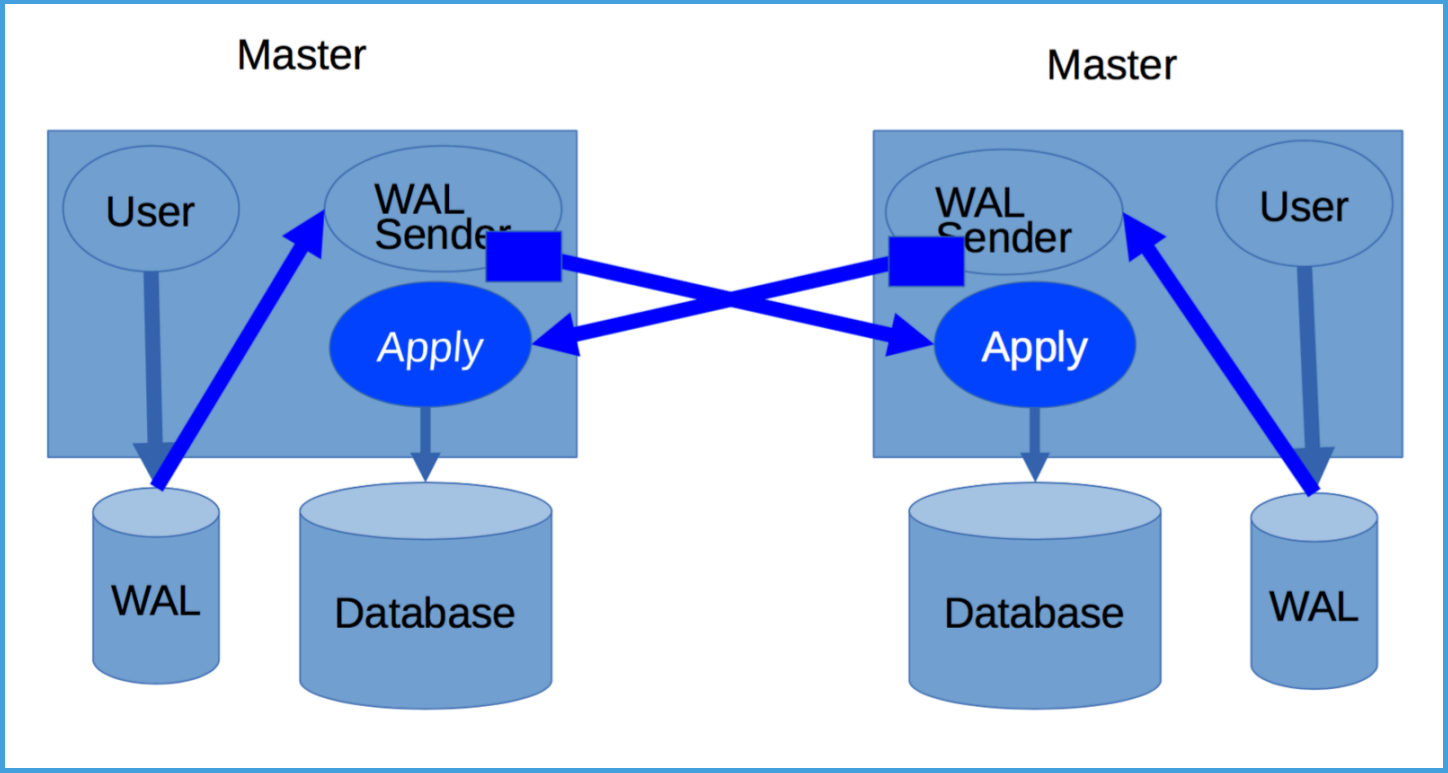
pglogical

- **Publish** / **Subscribe** model
- Multiple upstream (publisher) servers into a single subscriber
- Replicates transactions in commit order
- Selective Replication
- Online Upgrade
- Data Transport
  - Data integration
  - Streaming changes to analytical database
  - Master configuration data management
  - ...
- Optionally synchronous (mix)

BDR Bi-directional Replication

- The project is used for feeding logical replication development in PostgreSQL
- Multi-master
- Asynchronous
- Optimistic conflict detection (after commit)
- Does not prevent concurrent writes
- Conflict resolution:
-
  - Happens automatically
  - Last update wins by default
  - Custom resolution triggers
- Eventually consistent (cluster)

BDR Bi-Directional Replication



Bi-directional Replication

Thank you!

References

1. PostgreSQL 9 High Availability Cookbook
2. PostgreSQL Replication
3. PostgreSQL Documentation - High Availability, Load Balancing and Replication
4. BDR Documentation
5. PostgreSQL 9 Administration Cookbook - Second Edition
6. Why Logical Replication?
7. pglogical
8. Performance limits of logical replication solutions
9. Streaming replication slots in PostgreSQL 9.4
10. Failover slots for PostgreSQL
11. Continuous Archiving and Point-in-Time Recovery (PITR)
12. pg_rewind Nordic PGDay presentation by Heikki Linnakangas