Replication in Postgres

Fanavaran Anisa
Iran Linux House

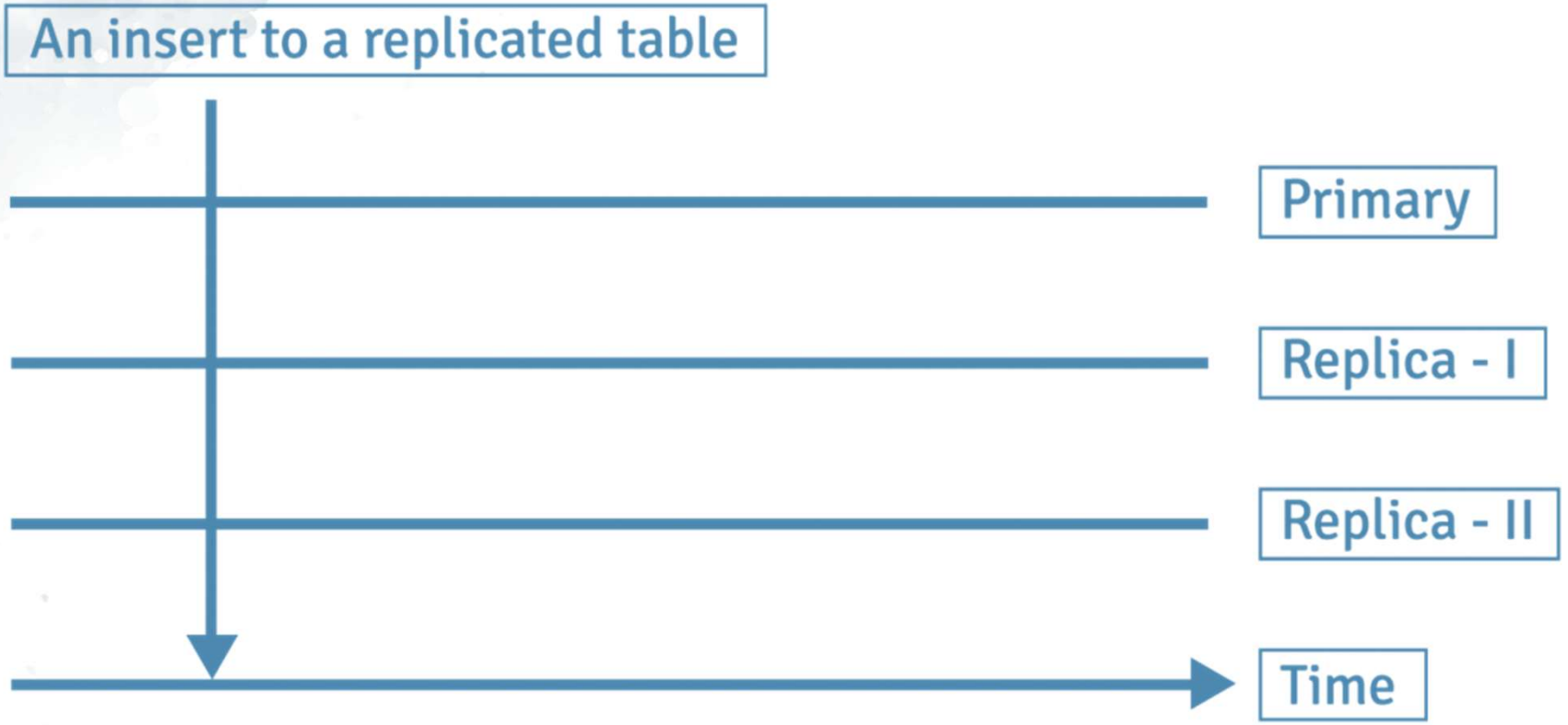
Linux & Open Source Training Center

www.anisa.co.ir

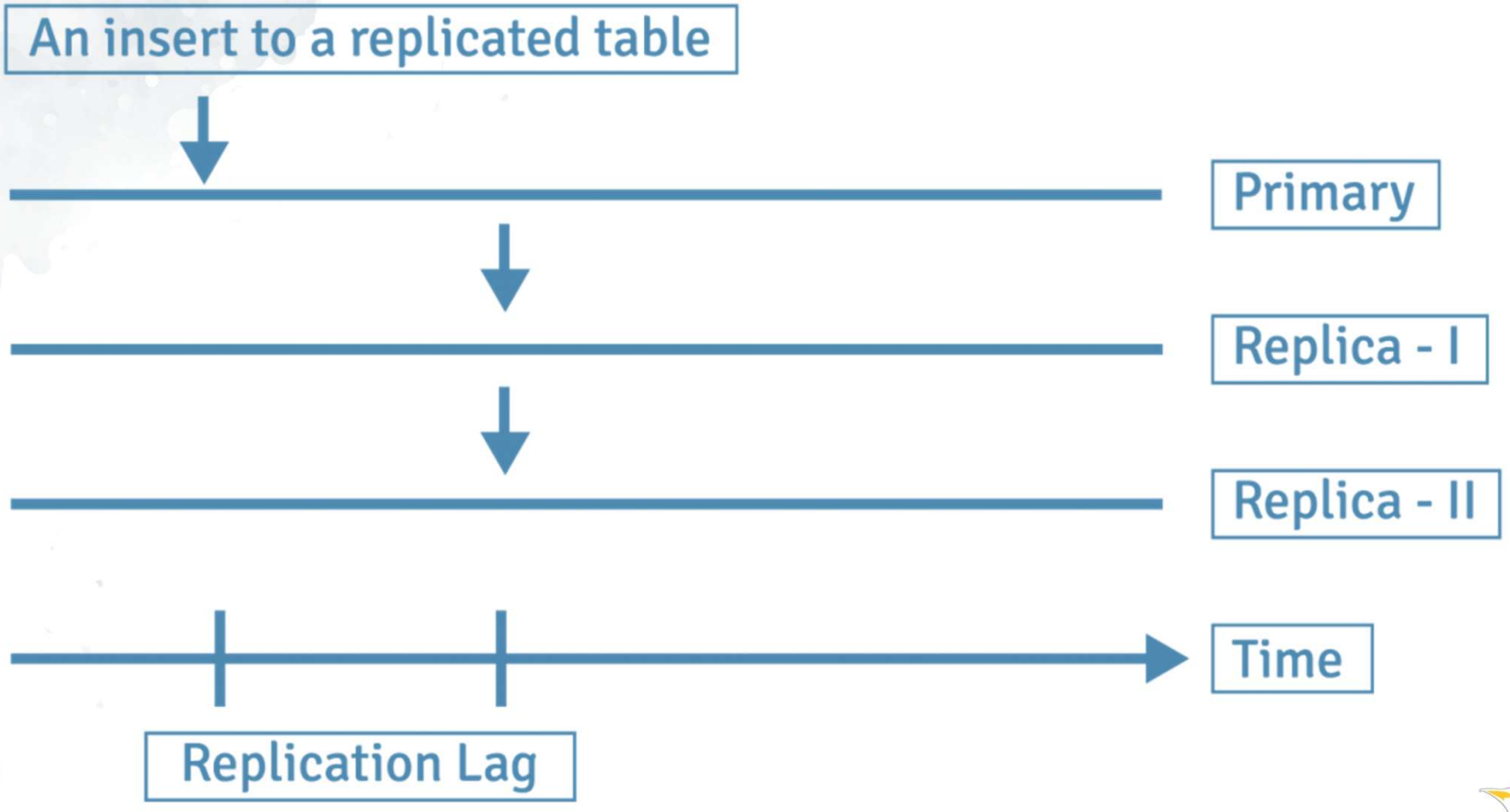
Replication Strategies

- Logical Replication
- Log-Shipping
- WAL Streaming

Synchronous Replication



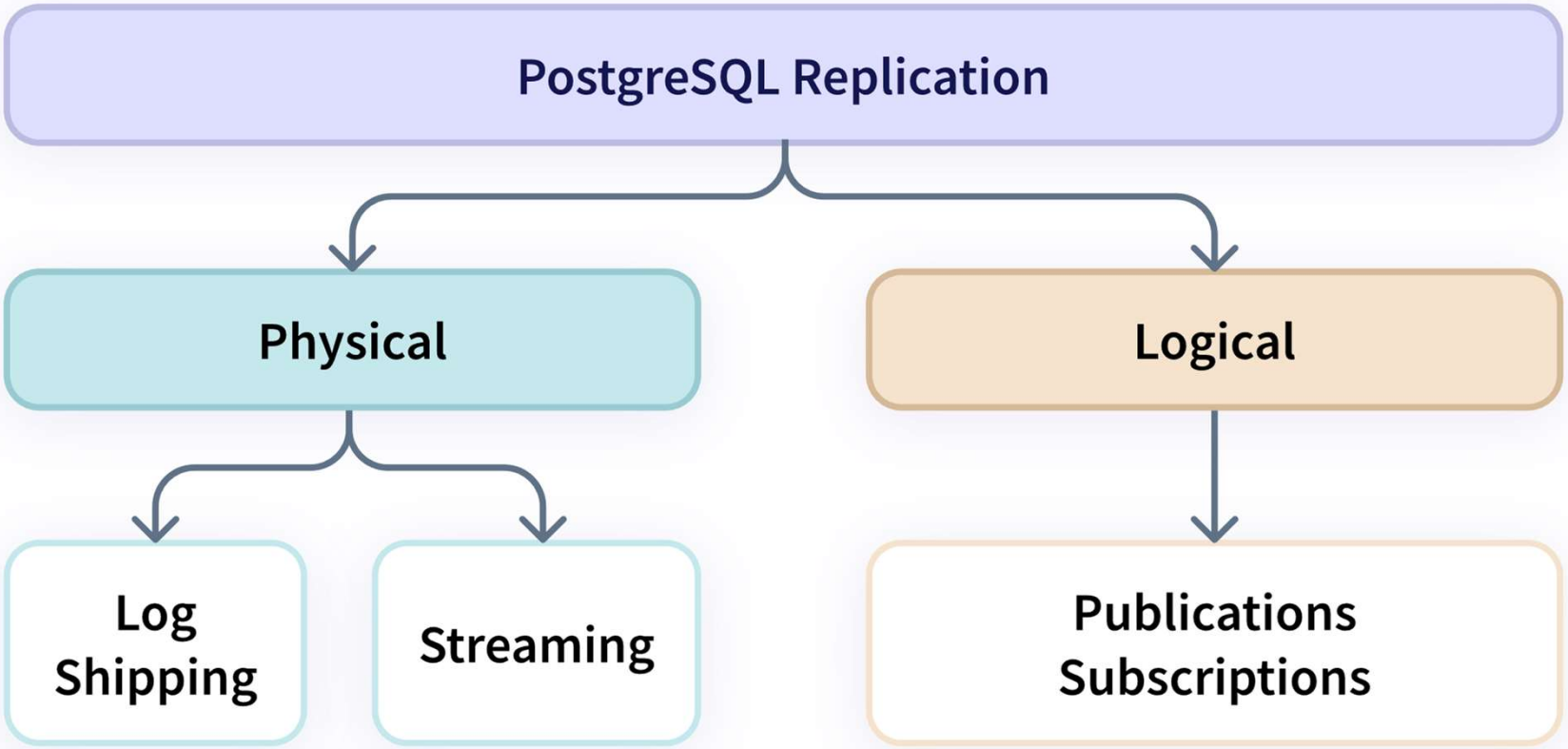
Asynchronous Replication



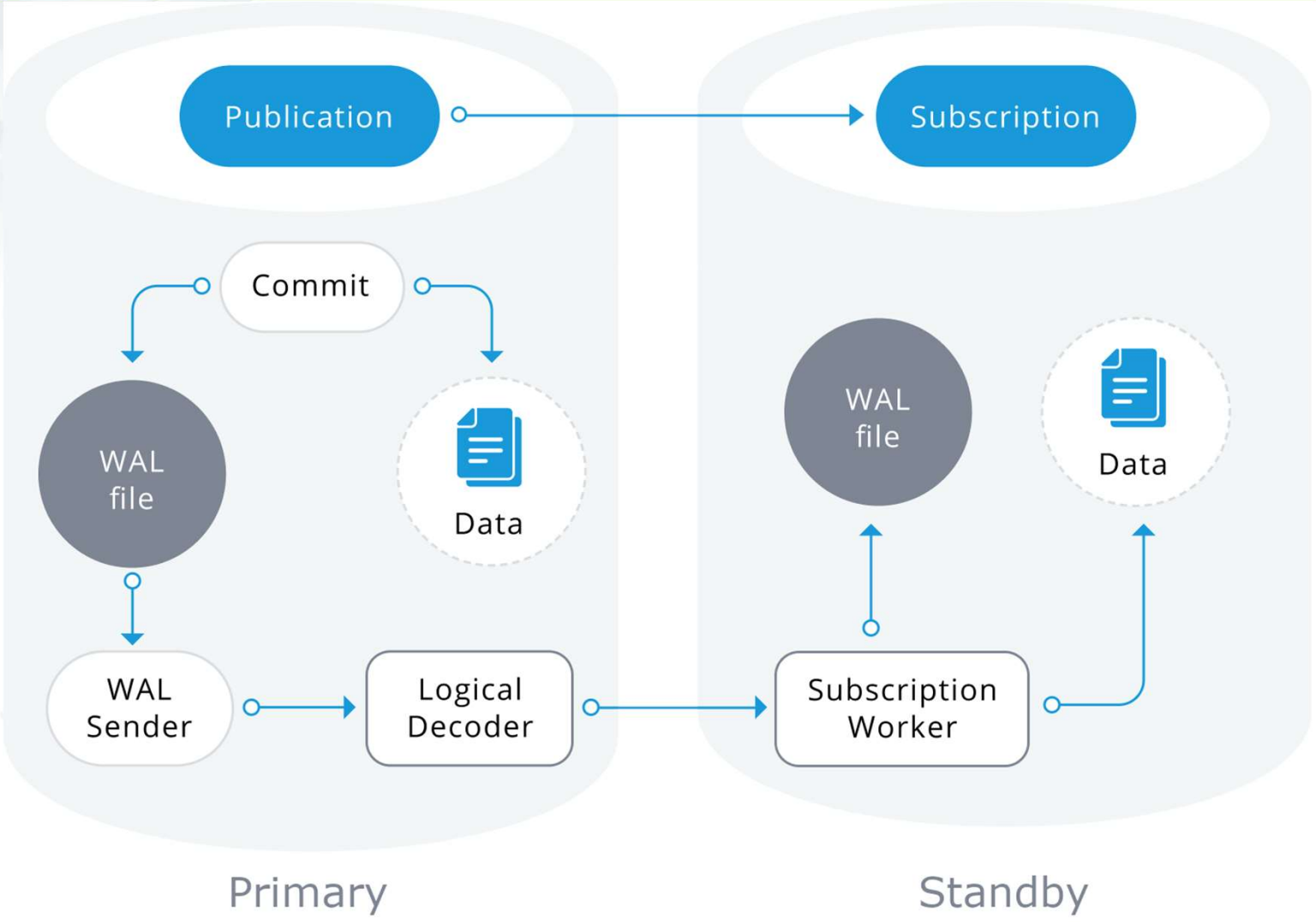
Multi-Master Replication (MMR)

In Multi-Master Replication (MMR), changes to table rows in more than one designated master database are replicated to their counterpart tables in every other master database. In this model conflict resolution schemes are often employed to avoid problems like duplicate primary keys

Postgres Replication Overview



Logical Replication



Logical Replication

- Logical replication uses a **publish** and **subscribe** model with one or more subscribers subscribing to one or more publications on a publisher node.
- **Subscribers** pull data from the **publications** they subscribe to and may subsequently re-publish data to allow cascading replication or more complex configurations.
- Logical replication of a table typically **starts with taking a snapshot of the data on the publisher database** and copying that to the subscriber.
- Once that is done, the changes on the publisher are sent to the subscriber as they occur in real-time.
- The subscriber applies the data in the same order as the publisher so that transactional consistency is guaranteed for publications within a single subscription.

Logical Replication

The typical use-cases for logical replication are:

- Sending incremental changes in a single database or a subset of a database to subscribers as they occur.
- Firing triggers for individual changes as they arrive on the subscriber.
- Consolidating multiple databases into a single one (for example for analytical purposes).
- Replicating between different major versions of PostgreSQL.
- Replicating between PostgreSQL instances on different platforms (for example Linux to Windows)
- Giving access to replicated data to different groups of users.
- Sharing a subset of the database between multiple databases..

Logical Replication -Publisher

- A published table must have a **replica identity** configured in order to be able to replicate UPDATE and DELETE operations, so that appropriate rows to update or delete can be identified on the subscriber side.
- By default, this is **the primary key**, if there is one. Another unique index (with certain additional requirements) can also be set to be the replica identity.
- If the table does not have any suitable key, then it can be set to **replica identity FULL**, which means the entire row becomes the key
- Publications are different from schemas and do not affect how the table is accessed. Each table can be added to multiple publications if needed. Publications may currently **only contain tables and all tables in schema**. Objects must be added explicitly, except when a publication is created for ALL TABLES.

Logical Replication -Publisher

```
test_pub=# CREATE PUBLICATION pub1 FOR TABLE t1;
CREATE PUBLICATION
test_pub=# CREATE PUBLICATION pub2 FOR TABLE t2 WITH (publish = 'truncate');
CREATE PUBLICATION
test_pub=# CREATE PUBLICATION pub3a FOR TABLE t3 WITH (publish = 'truncate');
CREATE PUBLICATION
test_pub=# CREATE PUBLICATION pub3b FOR TABLE t3 WHERE (e > 5);
CREATE PUBLICATION
```

Logical Replication - Row Filters

```
test_pub=# CREATE PUBLICATION p1 FOR TABLE t1 WHERE (a > 5 AND c = 'NSW');
CREATE PUBLICATION
test_pub=# CREATE PUBLICATION p2 FOR TABLE t1, t2 WHERE (e = 99);
CREATE PUBLICATION
test_pub=# CREATE PUBLICATION p3 FOR TABLE t2 WHERE (d = 10), t3 WHERE (g = 10);
CREATE PUBLICATION
```

Logical Replication - Row Filters

```
test_pub=# CREATE PUBLICATION p1 FOR TABLE t1 WHERE (a > 5 AND c = 'NSW');
CREATE PUBLICATION
test_pub=# CREATE PUBLICATION p2 FOR TABLE t1, t2 WHERE (e = 99);
CREATE PUBLICATION
test_pub=# CREATE PUBLICATION p3 FOR TABLE t2 WHERE (d = 10), t3 WHERE (g = 10);
CREATE PUBLICATION
```

Logical Replication - Row Filters

```
test_pub=# \dRp+
                                Publication p1
  Owner   | All tables | Inserts | Updates | Deletes | Truncates | Via root
----------+------------+---------+---------+---------+-----------+----------
 postgres | f          | t       | t       | t       | t         | f
Tables:
    "public.t1" WHERE ((a > 5) AND (c = 'NSW'::text))

                                Publication p2
  Owner   | All tables | Inserts | Updates | Deletes | Truncates | Via root
----------+------------+---------+---------+---------+-----------+----------
 postgres | f          | t       | t       | t       | t         | f
Tables:
    "public.t1"
    "public.t2" WHERE (e = 99)
```

Logical Replication - Column Filters

```
test_pub=# CREATE PUBLICATION p1 FOR TABLE t1 (id, b, a, d);
CREATE PUBLICATION
```

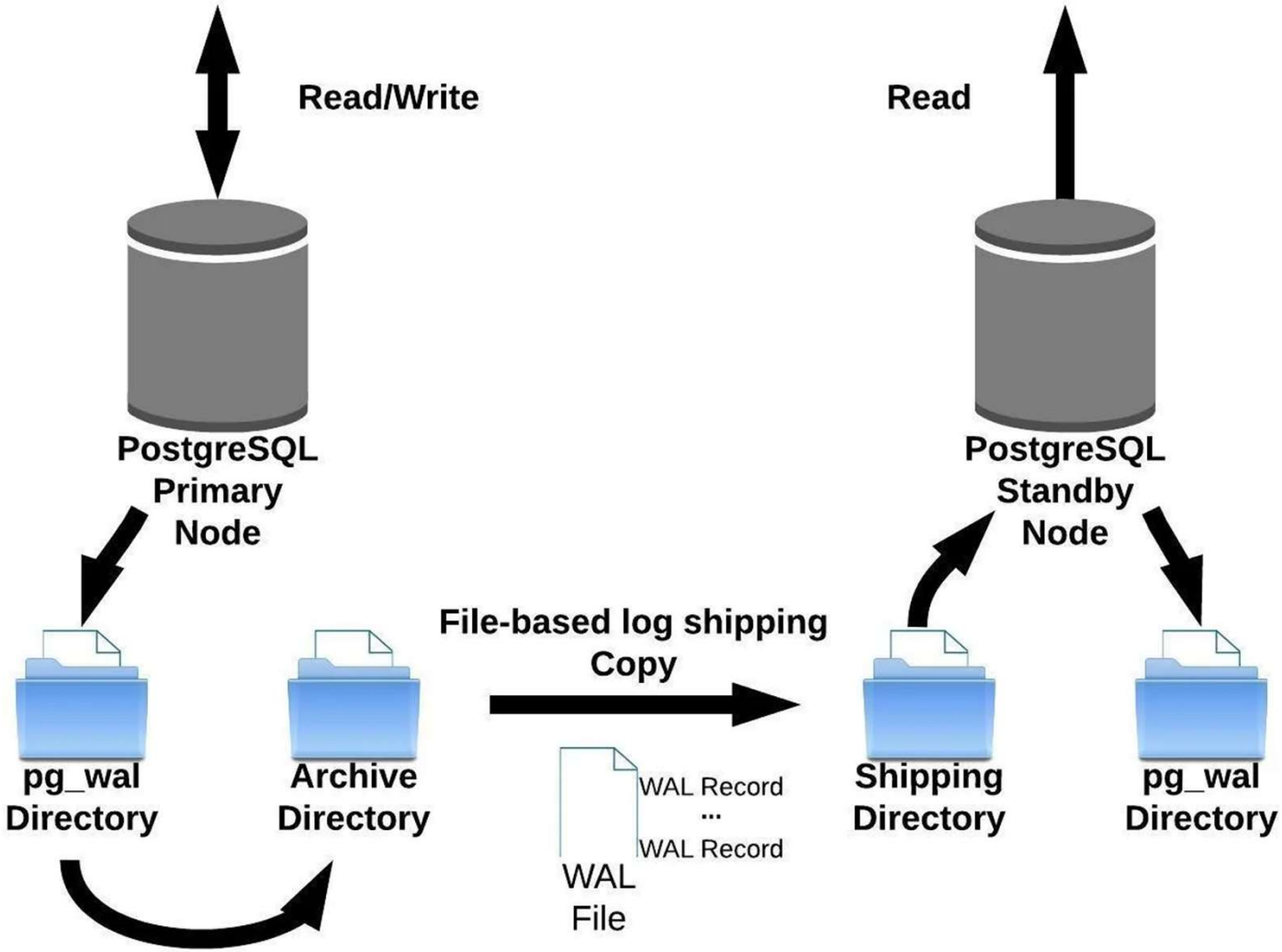
`psql` can be used to show the column lists (if defined) for each publication.

```
test_pub=# \dRp+
                                Publication p1
  Owner   | All tables | Inserts | Updates | Deletes | Truncates | Via root
----------+------------+---------+---------+---------+-----------+----------
 postgres | f          | t       | t       | t       | t         | f
Tables:
    "public.t1" (id, a, b, d)
```

Logical Replication - Subscriber

```
test_sub=# CREATE SUBSCRIPTION sub1
test_sub-# CONNECTION 'host=localhost dbname=test_pub application_name=sub1'
test_sub-# PUBLICATION pub1;
CREATE SUBSCRIPTION
test_sub=# CREATE SUBSCRIPTION sub2
test_sub-# CONNECTION 'host=localhost dbname=test_pub application_name=sub2'
test_sub-# PUBLICATION pub2;
CREATE SUBSCRIPTION
test_sub=# CREATE SUBSCRIPTION sub3
test_sub-# CONNECTION 'host=localhost dbname=test_pub application_name=sub3'
test_sub-# PUBLICATION pub3a, pub3b;
CREATE SUBSCRIPTION
```

File-Based Log Shipping



File-Based Log Shipping

- Continuous archiving can be used to create a high availability (HA) cluster.
- Directly moving WAL records from one database server to another is typically described **as log shipping**.
- This capability is widely referred to as **warm standby** or **log shipping**.
- PostgreSQL implements file-based log shipping by **transferring WAL records one file (WAL segment) at a time**.
- log shipping is **asynchronous**. As a result, there is a window for data loss should the primary server suffer a catastrophic failure; transactions not yet shipped will be lost.
- To have a more small data loss window, we can use **archive_timeout.** To limit how old unarchived data can be, you can set **archive_timeout** to force the server to switch to a new WAL segment file periodically

File-Based Log Shipping

- Recovery performance is sufficiently good that the standby will typically be only moments away from full availability once it has been activated.
- Restoring a server from an **archived base backup** and rollforward will take considerably longer, so that technique only offers a solution for disaster recovery, not high availability.
- A standby server can also be used for read-only queries, in which case it is called a **hot standby server**

File-Based Log Shipping - Planning

- It is usually wise to create the primary and standby servers so that **they are as similar as possible**, at least from the perspective of the database server
- Keep in mind that if **CREATE TABLESPACE** is executed on the primary, any new mount point needed for it must be created on the primary and all standby servers before the command is executed.
- Hardware need **not be exactly the same**, but experience shows that maintaining two identical systems is easier than maintaining two dissimilar ones over the lifetime of the application and system.
- In general, log shipping between servers running **different major PostgreSQL release levels is not possible**
- Store the WAL files in a shared folder so all the Replicas can access it.

File-Based Log Shipping – Primary Setting

- **Set up continuous archiving on the primary** to an archive directory accessible from the standby
- The archive **location should be accessible from the standby even when the primary is down**, i.e., it should reside on the standby server itself or another trusted server, not on the primary server.
- You can have **any number of standby servers**, but if you use streaming replication, make sure you set **max_wal_senders** high enough in the primary to allow them to be connected simultaneously
- Take **a base backup** to bootstrap the standby server.

```
archive_mode = on
archive_command = 'cp %p /archive/%f'
```

File-Based Log Shipping – Replica Setting

- Create a file **standby.signal** in the standby's cluster data directory.
- Set **restore_command** to a simple command to copy files from the WAL archive.
- If you plan to have multiple standby servers for high availability purposes, make sure that **recovery_target_timeline** is set to **latest** to make the standby server follow the timeline change that occurs at failover to another standby .
- **restore_command** should return immediately if the file does not exist; the server will retry the command again if necessary.
- To set up the standby server, **restore the base backup** taken from primary server.

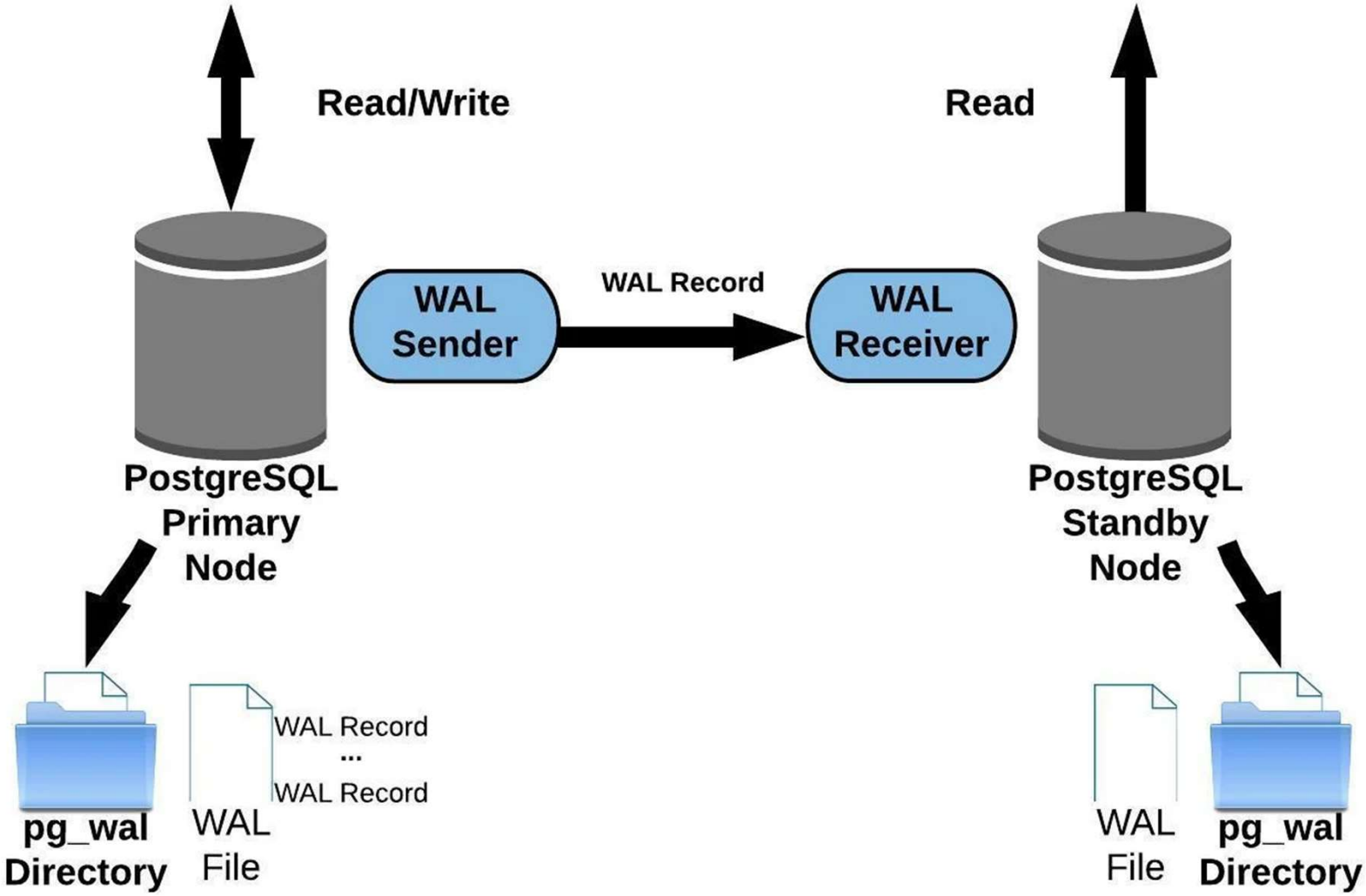
```
restore_command = ‘cp /archive/%f %p’
hot_standby = on
archive_cleanup_command = 'pg_archivecleanup /archive %r'
```

File-Based Log Shipping – archivecleanup

- use the **archive_cleanup_command** parameter to remove files that are no longer required by the standby server.
- The **pg_archivecleanup** utility is designed specifically to be used with **archive_cleanup_command** in typical **single-standby** configurations.
- Note however, that if you're using the archive for **backup purposes**, you need to retain files needed to recover from at least the latest base backup, even if they're no longer needed by the standby.

```
restore_command = ‘cp /archive/%f %p’
hot_standby = on
archive_cleanup_command = 'pg_archivecleanup /archive %r'
```

Streaming WAL Records



Streaming WAL Records

- The database servers **stream WAL records in chunks** to ensure that the data is always in sync.
- The standby servers **receive the WAL chunks by connecting with the master server**.
- advantage of streaming WAL records is that they don't wait for full capacity; these are streamed immediately. This helps in keeping the standby server up-to-date.
- **Streaming replication is asynchronous by default** however, it supports synchronous replication mode as well.
- If you use streaming replication **without file-based continuous archiving**, the server might recycle old WAL segments before the standby has received them. You can avoid this by setting **wal_keep_size** to a value large enough to ensure that WAL segments are not recycled too early

Streaming WAL Records – Primary Setting

- To use streaming replication, set up a file-based log-shipping standby server.
- On systems that support the keepalive socket option, setting **tcp_keepalives_idle**, **tcp_keepalives_interval** and **tcp_keepalives_count** helps the primary promptly notice a broken connection.
- https://postgresqlco.nf/doc/en/param/[parameter]
- Set the maximum number of concurrent connections from the standby servers (see **max_wal_senders** for details).

Streaming WAL Records – Authentication

- Standby servers must authenticate to the primary as an account that has the **REPLICATION** privilege or a **superuser**. It is recommended to create a dedicated user account with **REPLICATION** and **LOGIN** privileges for replication
- Client authentication for replication is controlled by a **pg_hba.conf** record specifying replication in the database field.

```
# Allow the user "foo" from host 192.168.1.100 to connect
to the primary
# as a replication standby if the user's password is
correctly supplied.
#
# TYPE DATABASE USER ADDRESS METHOD
host replication foo 192.168.1.100/32 md5
```

Streaming WAL Records – Primary Setting

- **pg_basebackup -U replication_user -D /backup -P -Xs -c fast**
- - if not , the replica not be synced (` FATAL:  database system identifier differs between the primary and standby`)

```
archive_mode = on
archive_command = ‘cp %p /archive/%f’
```

```
listen_addresses =‘*’
wal_level = replica
max_wal_senders = 10
wal_keep_size = 64
```

Streaming WAL Records – Primary Setting

- Postgres WAL replication is **asynchronous** by default however, it is possible to make streaming replication synchronous by setting the `**synchronous_standby_names**`
- **synchronous_standby_names** specifies the number and names of synchronous standbys that transaction commits made when **synchronous_commit** is set to **on**, **remote_apply** or **remote_write** will wait for responses from. Such transaction commits may never be completed if any one of synchronous standbys should crash.
- The best solution for high availability is to ensure you keep as many synchronous standbys as requested.

```
synchronous_standby_names = 'FIRST 2 (s1, s2, s3)’
synchronous_standby_names = 'ANY 2 (s1, s2, s3)'
```

Streaming WAL Records – Replica Setting

- Restore the **basebackup**
- create a `**standby.signal**` file in the data directory
- Start the replica server

```
hot_standby = on
primary_conninfo = 'host=postgres_primary port=5432
user=replication_user password=replica123'
recovery_target_timeline = 'latest’
```

Continuous Archiving in Standby

- When continuous WAL archiving is used in a standby, there are two different scenarios:
  - the WAL archive can be shared between the primary and the standby
  - the standby can have its own WAL archive. When the standby has its own WAL archive, set **archive_mode** to **always**, and the standby will call the archive command for every WAL segment it receives, whether it's by restoring from the archive or by streaming replication.
- The shared archive can be handled similarly, but the **archive_command** or **archive_library** must test if the file being archived exists already, and if the existing file has identical contents.

Continuous Archiving in Standby

- **archive_command** or **archive_library** must test if the file being archived exists already, and if the existing file has identical contents. This requires more care in the **archive_command or archive_library**, as it must be careful to not overwrite an existing file with different contents, but return success if the exactly same file is archived twice. And all that must be done free of race conditions, if two servers attempt to archive the same file at the same time.

- If **archive_mode** is set to on, the archiver is not enabled during recovery or standby mode. If the standby server is promoted, it will start archiving after the promotion, but will not archive any WAL or timeline history files that it did not generate itself

Replication Slots

- **Replication slots** provide an automated way to ensure that the primary does not remove WAL segments until they have been received by all standbys, and that the primary does not remove rows which could cause a recovery conflict even when the standby is disconnected.
- it is possible to prevent the removal of old WAL segments using **wal_keep_size**, or by storing the segments in an archive using **archive_command** or **archive_library**. However, these methods often result in retaining more WAL segments than required, **whereas replication slots retain only the number of segments known to be needed**.
- On the other hand, replication slots can retain so many WAL segments that they fill up the space allocated for **pg_wal**; **max_slot_wal_keep_size** limits the size of WAL files retained by replication slots.

Replication Slots

You can create a replication slot like this:

```
postgres=# SELECT * FROM pg_create_physical_replication_slot('node_a_slot');
  slot_name  | lsn
-------------+-----
 node_a_slot |

postgres=# SELECT slot_name, slot_type, active FROM pg_replication_slots;
  slot_name  | slot_type | active
-------------+-----------+--------
 node_a_slot | physical  | f
(1 row)
```

To configure the standby to use this slot, `primary_slot_name` should be configured on the standby. Here is a simple example:

```
primary_conninfo = 'host=192.168.1.50 port=5432 user=foo password=foopass'
primary_slot_name = 'node_a_slot'
```

Failover

- If the primary server fails then the standby server should begin **failover procedures** .
- PostgreSQL does not provide the system software required to identify a failure on the primary and notify the standby database server. Many such tools exist and are well integrated with the operating system facilities required for successful failover, such as IP address migration.
- **If the standby server fails** then no failover need take place. If the standby server can be restarted, even some time later, then the recovery process can also be restarted immediately, taking advantage of restartable recovery. If the standby server cannot be restarted, then a full new standby server instance should be created.

Failover

- **If the primary server fails** and the standby server becomes the new primary, and then the old primary restarts, you must have a mechanism for informing the old primary that it is no longer the primary. This is sometimes known as STONITH (**Shoot The Other Node In The Head**), which is necessary to avoid situations where both systems think they are the primary, which will lead to confusion and ultimately data loss.
- **Many failover systems use just two systems, the primary and the standby**, connected by some kind of heartbeat mechanism to continually verify the connectivity between the two and the viability of the primary. It is also possible to use a third system (called a **witness server**) to prevent some cases of inappropriate failover
- To trigger failover of a log-shipping standby server, run **pg_ctl promote** or call **pg_promote()**

Degenerate State

- **Once failover to the standby occurs, there is only a single server in operation**. This is known as a degenerate state. The former standby is now the primary, but the former primary is down and might stay down. To return to normal operation, a standby server must be recreated, either on the former primary system when it comes up, or on a third, possibly new, system.
- The **pg_rewind** utility can be used to speed up this process on large clusters.
- Once complete, the primary and standby can be considered to have switched roles. Some people choose to use a third server to provide backup for the new primary until the new standby server is recreated, though clearly this complicates the system configuration and operational processes.

Monitoring

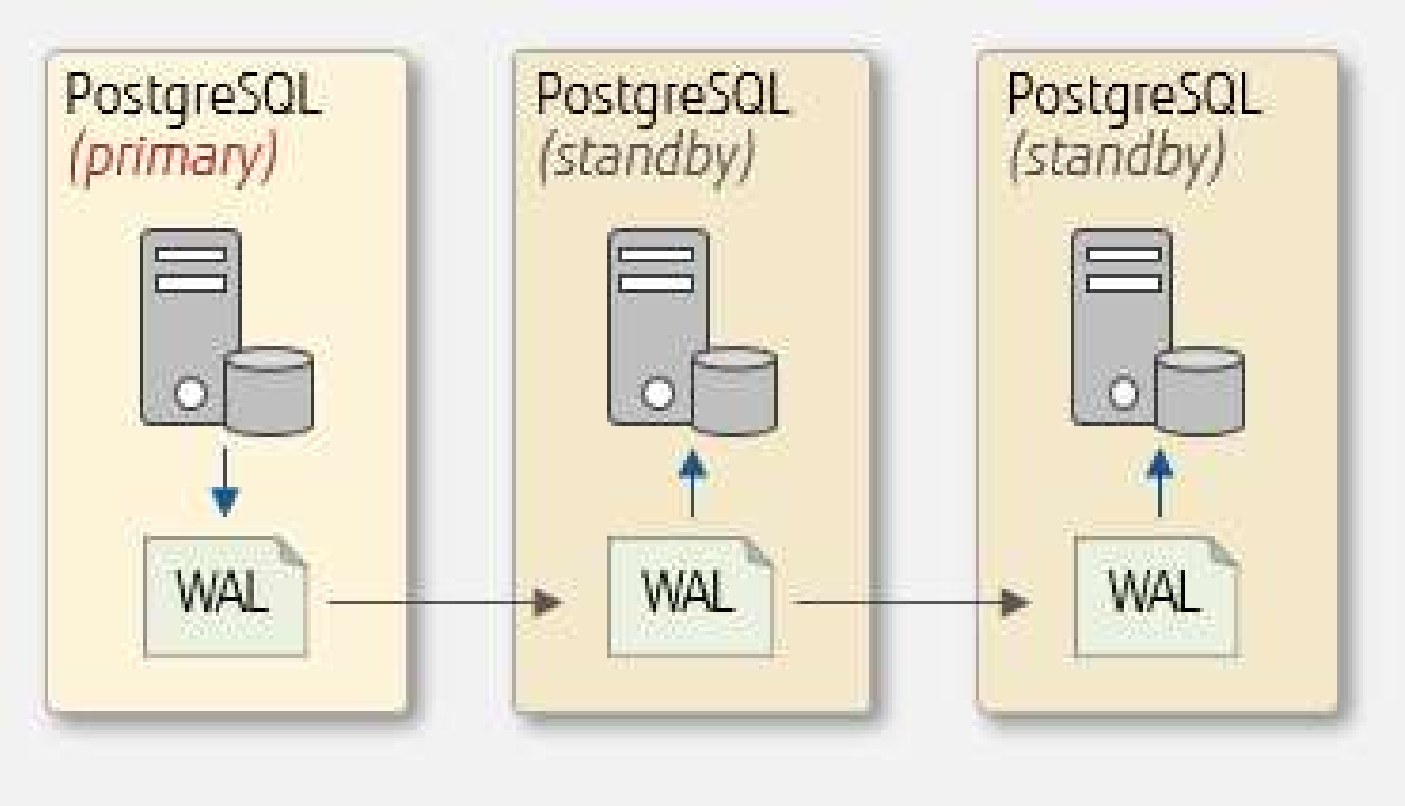
- An important health indicator of streaming replication is the amount of WAL records generated in the primary, but not yet applied in the standby.
- You can calculate this lag by comparing the current WAL write location on the primary with the last WAL location received by the standby. These locations can be retrieved using **pg_current_wal_lsn** on the primary and **pg_last_wal_receive_lsn** on the standby,

Monitoring

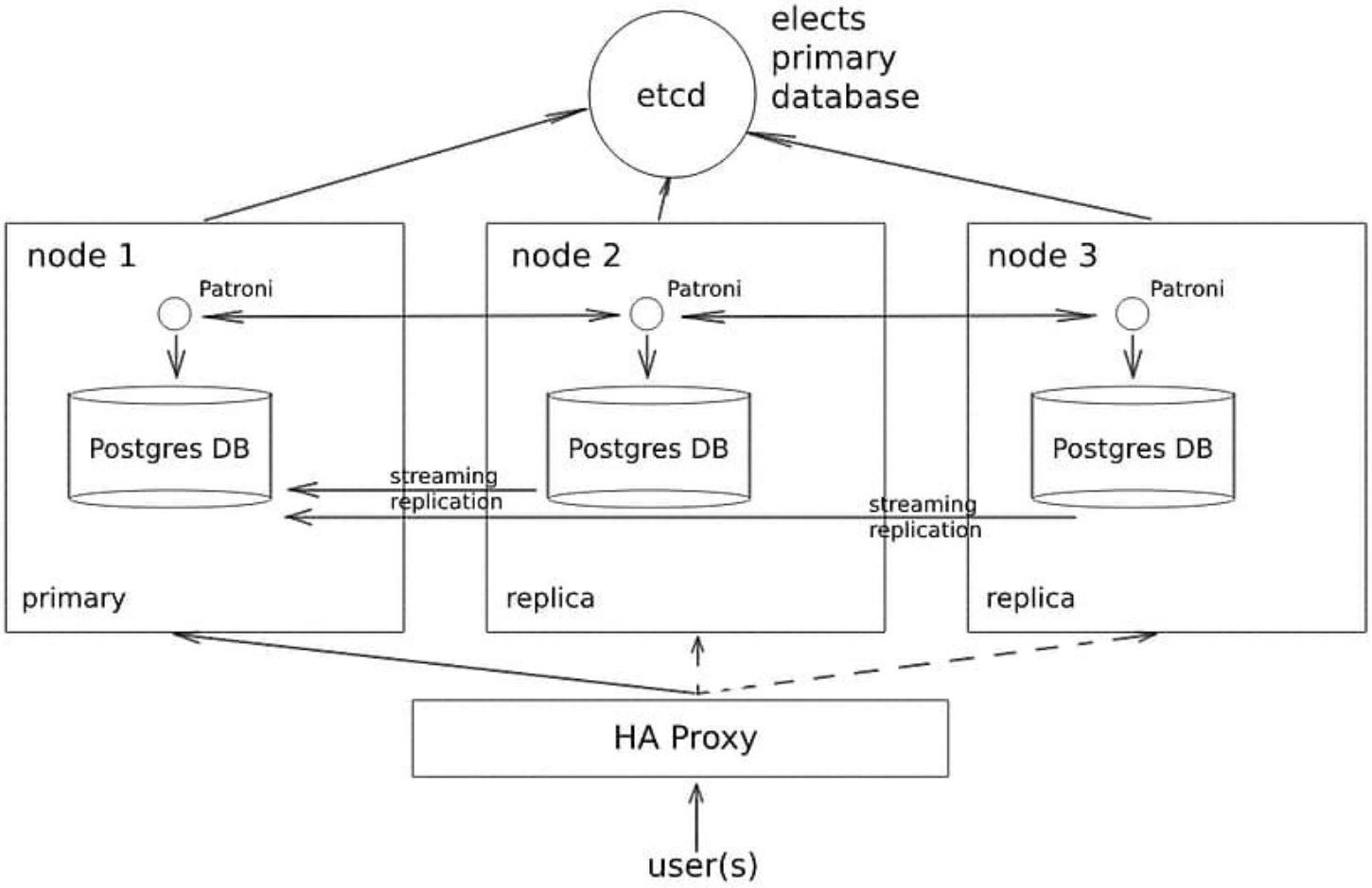
- You can retrieve a list of WAL sender processes via the **pg_stat_replication** view. Large differences between **pg_current_wal_lsn** and the view's **sent_lsn** field might indicate that the primary server is under heavy load, while differences between **sent_lsn** and **pg_last_wal_receive**_lsn on the standby might indicate network delay, or that the standby is under heavy load.

- On a hot standby, the status of the WAL receiver process can be retrieved via the **pg_stat_wal_receiver** view. A large difference between **pg_last_wal_replay_lsn** and the view's **flushed_lsn** indicates that WAL is being received faster than it can be replayed.

Cascading Replication



Failover – Third Party Tools : Patroni



Workshop Time!