PERCONA LIVE 2023

The Many Flavours of Replication

Robert Bernier
Consultant

Wednesday, May 24th

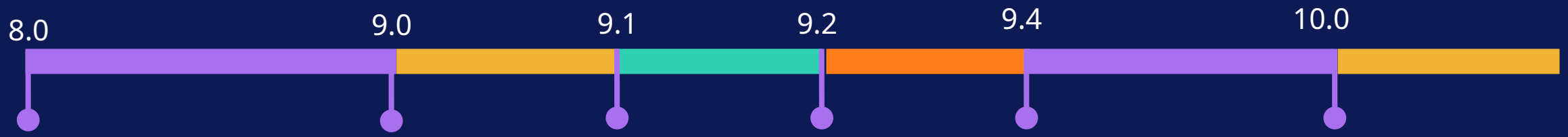
09:30 AM MST

PERCONA

The Many Flavours Of Replication

A Walk-thru of the various methods of PostgreSQL Replication

Timeline



|---------- Log Shipping ---------->|---------- Streaming ---------->|---------- Logical ---------->

Warm Standby (PITR)

Hot Standby

Without Slots

Cascading

With Slots

|---------- Asynchronous ---------->|---------- Synchronous Replication ---------->

Sending Servers

max_wal_senders
max_replication_slots
wal_keep_segments
wal_sender_timeout
track_commit_timestamp

Master Server

synchronous_standby_names
vacuum_defer_cleanup_age

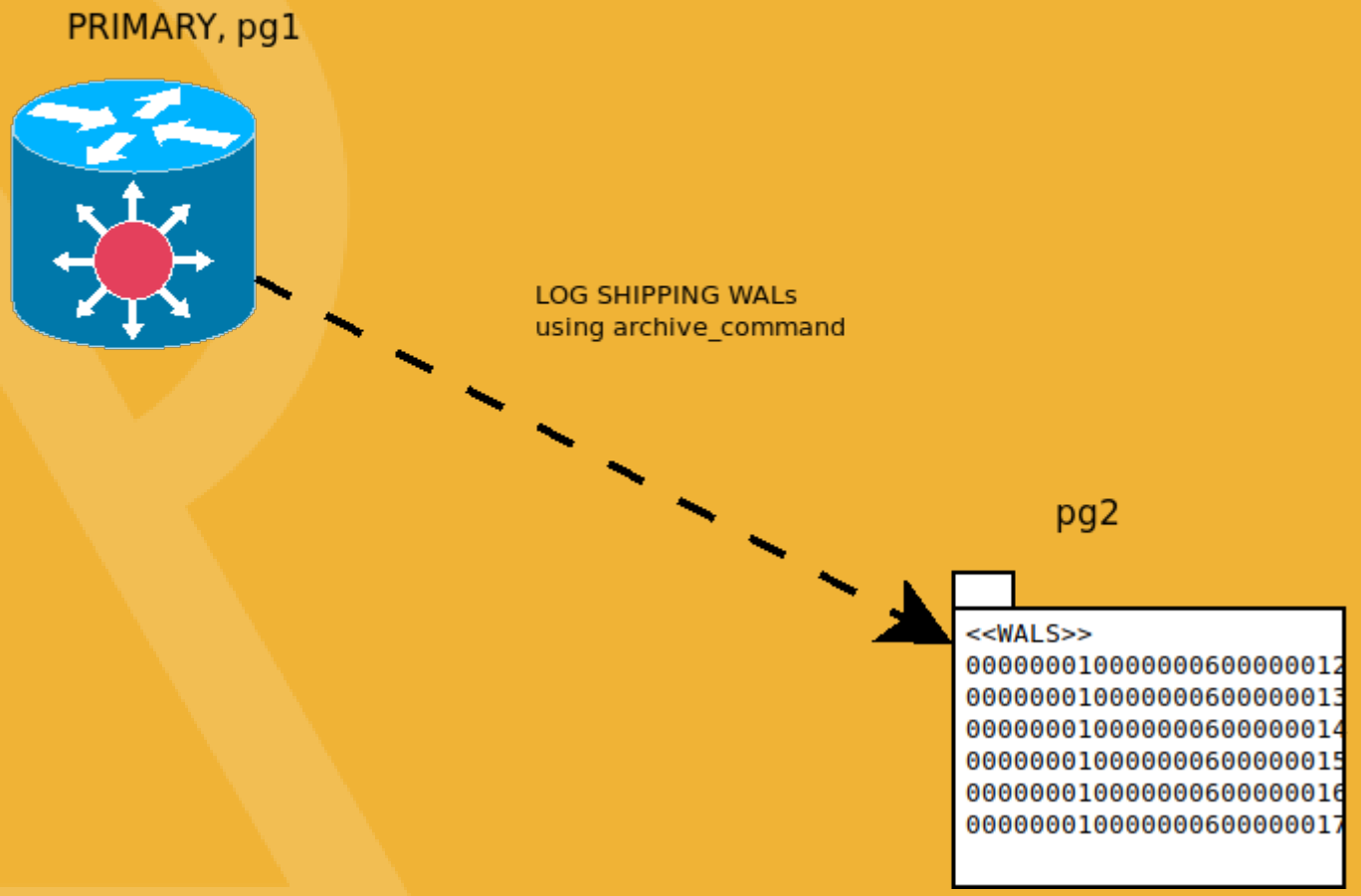
Standby Servers

primary_conninfo
primary_slot_name
hot_standby
promote_trigger_file
max_standby_archive_delay
max_standby_streaming_delay
wal_receiver_status_interval
host_standby_feedback
wal_receiver_timeout
wal_retrieve_retry_interval
recovery_min_apply_delay

Subscribers

max_logical_replication_workers
max_sync_workers_per_subscription

WAL LOG ARCHIVING



Log Shipping

Generate public key for postgres on PRIMARY and copy to REPLICA

```
ssh postgres@pg1
```

```
-bash-4.2$ ssh-keygen
Generating public/private rsa key pair.
Enter file in which to save the key (/var/lib/pgsql/.ssh/id_rsa):
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /var/lib/pgsql/.ssh/id_rsa.
Your public key has been saved in /var/lib/pgsql/.ssh/id_rsa.pub.
The key fingerprint is:
SHA256:7Ik+QQzwmZMrtW5HTtZAAEymAV5arTY8B0z5tZC4JmI postgres@pg1
The key's randomart image is:
...
```

```
# copy public key to hosts pg2 and pg3
ssh-copy-id postgres@pg2
ssh-copy-id postgres@pg3
```

Log Shipping Cont'd

```
# Host pg2: Create WAL directory on REPLICA
ssh postgres@pg2
mkdir -p $HOME/WAL
exit
```

```
# Host pg1: setup WAL Log shipping
# as root, sudo as postgres
ssh root@pg1
su - postgres
```

```
-- update system as superuser postgres
alter system set archive_mode = on;
alter system set archive_command = 'scp %p pg2:WAL/%f';
alter system set wal_keep_size = 100;
alter system set wal_log_hints = 'on';
```

```
# as root, restart postgres service
systemctl restart postgresql@14-main
```

Log Shipping Cont'd

```
-- Host pg1: Generate WALS
-- Login pg1, as postgres superuser and perform the following
dropdb database if exists db01;
create database db01;
\c db01
select *,'hello world'::text as comments
    into table t1
    from (select * from generate_series(1,1e6))t;
-- Flush data files to disk
checkpoint;
-- Force switch to a new write-ahead log file
select pg_walfile_name(pg_switch_wal());
```

```
# Host pg1: remote LOGIN host pg2
su - postgres
ssh postgres@pg2 ls -l WAL
```

Backups

About Basebackups

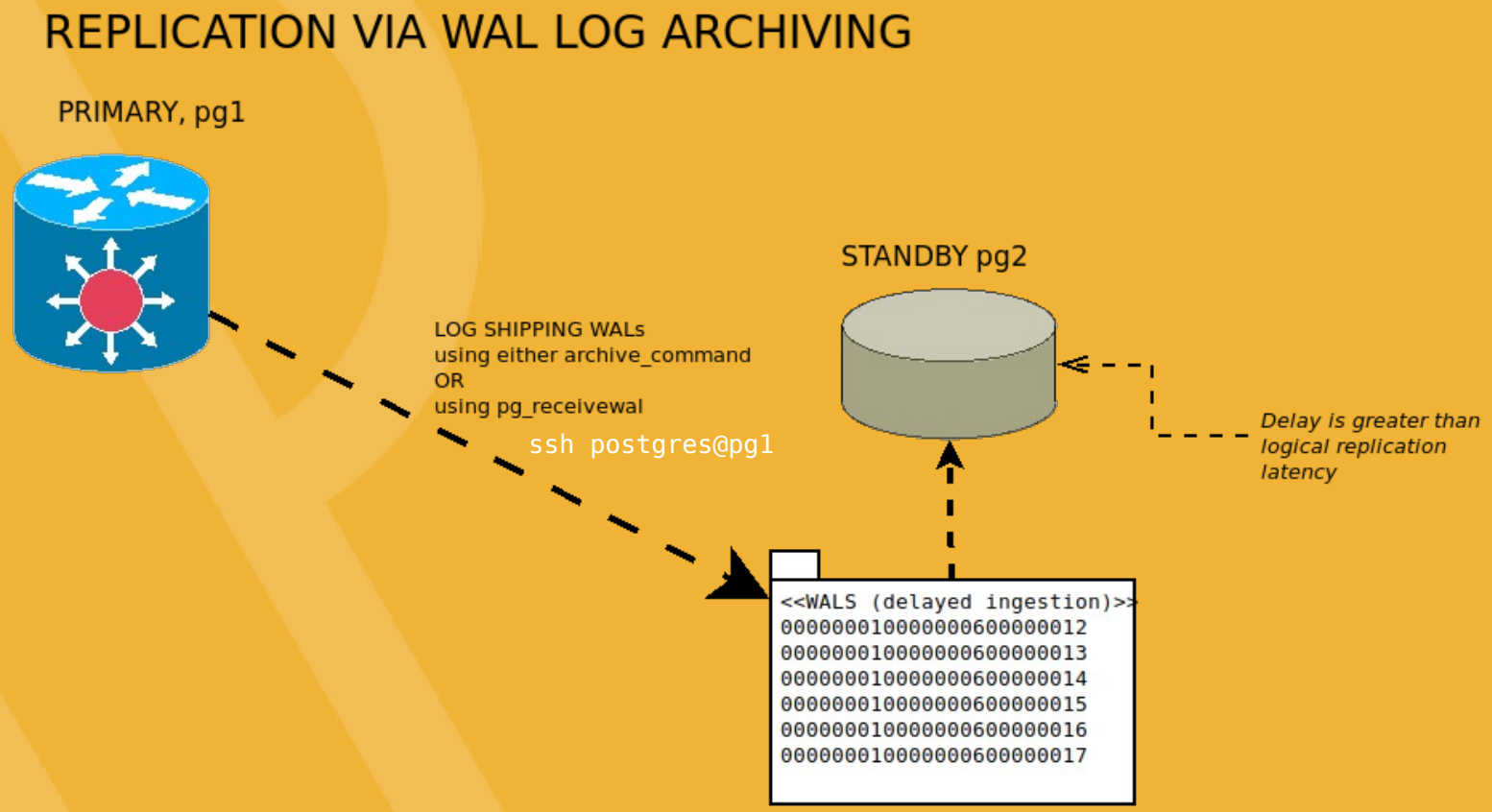
```
pg_basebackup --help
Usage:
  pg_basebackup [OPTION]...
Options controlling the output:
  -D, --pgdata=DIRECTORY receive base backup into directory
  -F, --format=p|t       output format (plain (default), tar)
  -r, --max-rate=RATE    maximum transfer rate to transfer data directory
                         (in kB/s, or use suffix "k" or "M")
  -R, --write-recovery-conf
                         write configuration for replication
  -T, --tablespace-mapping=OLDDIR=NEWDIR
                         relocate tablespace in OLDDIR to NEWDIR
      --waldir=WALDIR    location for the write-ahead log directory
  -X, --wal-method=none|fetch|stream
                         include required WAL files with specified method
  -z, --gzip             compress tar output
  -Z, --compress=0-9     compress tar output with given compression level

General options:
  -c, --checkpoint=fast|spread
                         set fast or spread checkpointing
  -C, --create-slot      create replication slot
  -l, --label=LABEL      set backup label
  -n, --no-clean         do not clean up after errors
  -N, --no-sync          do not wait for changes to be written safely to disk
  -P, --progress         show progress information
  -S, --slot=SLOTNAME    replication slot to use
  -v, --verbose          output verbose messages
  -V, --version          output version information, then exit
      --no-slot          prevent creation of temporary replication slot
      --no-verify-checksums
                         do not verify checksums
  -?, --help             show this help, then exit
```

Backups Cont’d

```
Connection options:
  -d, --dbname=CONNSTR   connection string
  -h, --host=HOSTNAME    database server host or socket directory
  -p, --port=PORT        database server port number
  -s, --status-interval=INTERVAL
                         time between status packets sent to server (in seconds)
  -U, --username=NAME    connect as specified database user
  -w, --no-password      never prompt for password
  -W, --password         force password prompt (should happen automatically)
```

Log Shipping Replication



Replication Via Log Shipping

```
# Confirm remote connectivity access
# on pg1: check for listening service
[root@pg1 ~]# netstat -tlnp
Active Internet connections (only servers)
Proto Recv-Q Send-Q Local Address           Foreign Address         State       PID/Program name
tcp        0      0 0.0.0.0:22              0.0.0.0:*               LISTEN      297/sshd
tcp        0      0 0.0.0.0:5432            0.0.0.0:*               LISTEN      1629/postmaster
tcp6       0      0 :::22                   :::*                    LISTEN      297/sshd
tcp6       0      0 :::5432                 :::*                    LISTEN      1629/postmaster
```

```
# on pg2: attempt test login
[root@pg2 ~] psql 'host=pg1 user=postgres password=postgres' -c "select 1 as ping"
 ping
------
    1
```

```
# Check Server Status On Host pg2

Method 1:
systemctl status postgresql-14       # CENTOS
systemctl status postgresql@14-main  # Ubuntu

Method 2:
ps aux | grep postgres
```

Replication Via Log Shipping Cont'd

```
# Delete any pre-existing data cluster
systemctl stop postgresql@14-main
rm -rf /var/lib/postgreql/14/data/*
```

```
# Perform BaseBackup

su - postgres
export PGDATA=/var/lib/postgresql/14/main
cd $HOME
/usr/bin/pg_basebackup -d 'host=pg1 user=postgres password=postgres port=5432' \
                       --wal-method=stream \
                       -c fast \
                       -l basebackup \
                       -D /var/lib/postgresql/14/main \
                       -P -v
```

```
# Configure REPLICA

echo "
hot_standby = 'on'
recovery_target_timeline = 'latest'

restore_command='cp /var/lib/postgresql/WAL/%f "%p"'
archive_cleanup_command = 'pg_archivecleanup /var/lib/postgresql/WAL %r'
" >> /var/lib/postgresql/14/main/postgresql.auto.conf

touch /var/lib/postgresql/14/main/standby.signal
```

Replication Via Log Shipping Cont'd

```
# As root: REPLICA service start
systemctl start postgresql@14-main
-----------------------------------------------------------------
netstat -tlnp
Active Internet connections (only servers)
Proto Recv-Q Send-Q Local Address           Foreign Address         State       PID/Program name
tcp        0      0 0.0.0.0:22              0.0.0.0:*               LISTEN      297/sshd
tcp        0      0 0.0.0.0:5432            0.0.0.0:*               LISTEN      867/postmaster
tcp6       0      0 :::22                   :::*                    LISTEN      297/sshd
tcp6       0      0 :::5432                 :::*                    LISTEN      867/postmaster
```

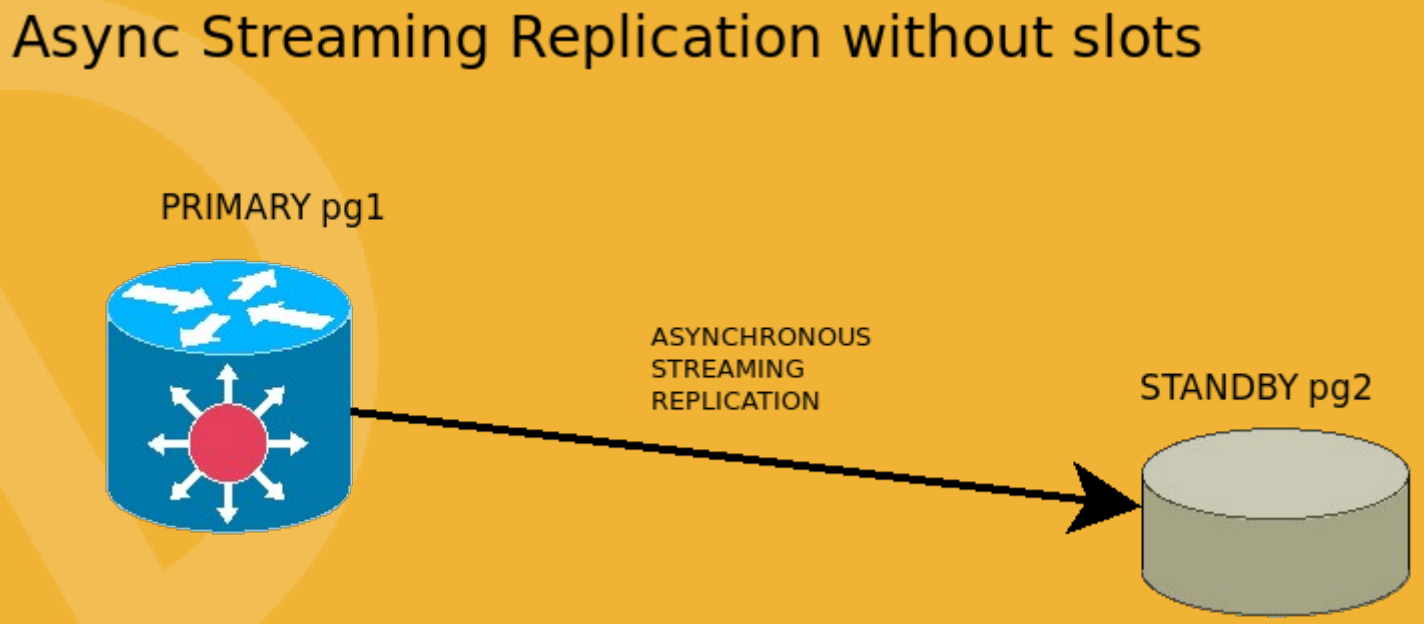
```
# Create table and populate records on host pg1

SQL="select * into table t2 from generate_series(1,1e6)"
psql 'host=pg1 dbname=db01 user=postgres password=postgres'<<<$SQL
psql 'host=pg1 dbname=db01 user=postgres password=postgres' -c 'checkpoint;select pg_switch_wal()'
```

```
# Confirm replication on host pg2

[root@pg2 ~] psql 'host=pg2 dbname=db01 user=postgres password=postgres' -c '\dt+'
                    List of relations
 Schema | Name | Type  |  Owner   | Size  | Description
--------+------+-------+----------+-------+-------------
 public | t1   | table | postgres | 50 MB |
 public | t2   | table | postgres | 35 MB |
```

Streaming Replication Without Slots



Replication Via Streaming

Without Slots

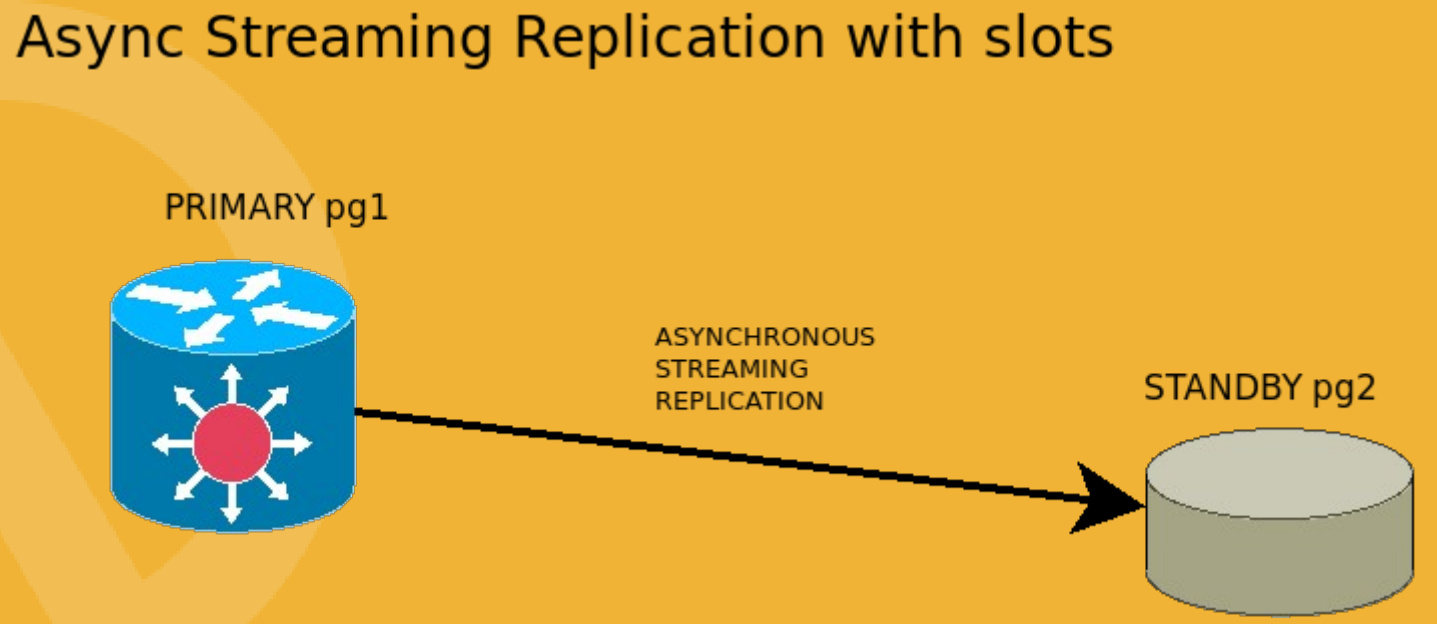
```
# Execute on PRIMARY, host pg1
-- Add a replicating ROLE
create role replicant with login replication password 'mypassword';
--
-- enable streaming replication
alter system set wal_level = 'replica';
```

```
# as root; reload the service on PRIMARY pg1
systemctl reload postgresql@14-main
```

```
-- Execute on REPLICA, host pg2
-- point REPLICA to PRIMARY
alter system set primary_conninfo = 'host=pg1 user=replicant password=mypassword';
```

```
# as root; restart the service on REPLICA pg2
systemctl restart postgresql@14-main
```

Streaming Replication With Slots



Replication Via Streaming

With Slots

METHOD 1: update existing configuration

```
# Execute on PRIMARY, host pg1
select pg_create_physical_replication_slot('pg2');
```

```
# Execute on REPLICA, host pg2

-- point REPLICA to PRIMARY using slot
alter system set primary_slot_name = 'pg2'
-----------------------------------------------------
# as root; restart the service on REPLICA pg2
systemctl restart postgresql@14-main
```

```
# Execute on PRIMARY, host pg1
# confirm replication slot is active

postgres=# select slot_name, slot_type, active, active_pid from pg_replication_slots;
 slot_name | slot_type | active | active_pid
-----------+-----------+--------+------------
 pg2       | physical  | t      |       2313
```

Replication Via Streaming

With Slots

METHOD 2: generate new basebackup

As root: execute the following on REPLICA pg2

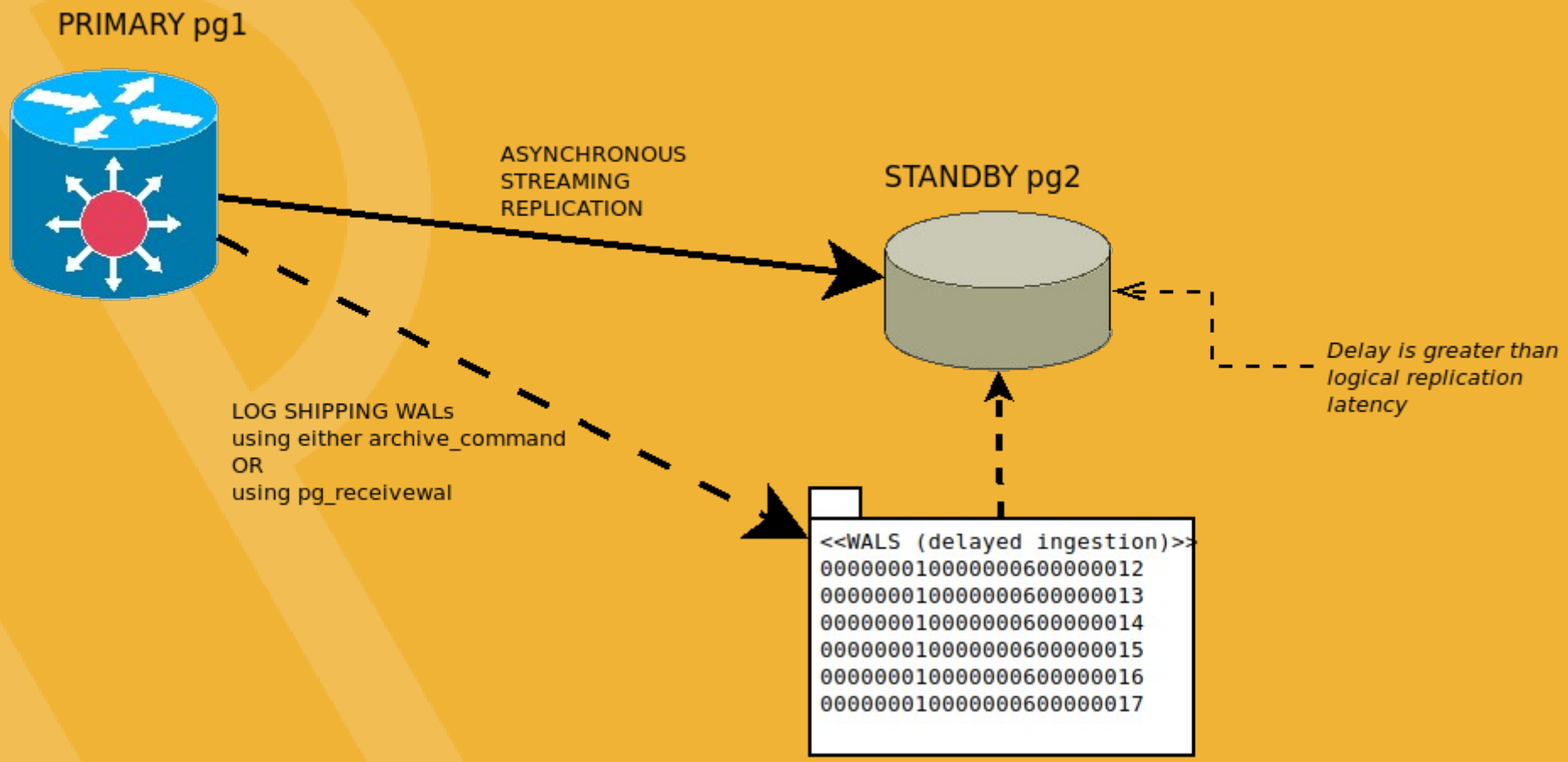
- Stop the service
- Delete the data cluster
- Create a new basebackup using slots, it's assumed no physical slot has already been created on PRIMARY

```
export PGDATA=/var/lib/pgsql/14/data       # CENTOS
export PGDATA=/var/lib/postgresql/14/main  # Ubuntu
```

```
# PGDATA environment variable assumes CENTOS

systemctl stop postgresql@14-main
su - postgres
rm -rf $PGDATA
/usr/bin/pg_basebackup -d 'host=pg1 user=postgres password=postgres port=5432' \
                       --wal-method=stream \
                       -l basebackup \
                       -D $PGDATA \
                       -R -P -v \
                       --create-slot \
                       --slot=pg2
touch $PGDATA/standby.signal
```

Async Streaming Replication with slots and Log Shipping Hybrid



PostgreSQL Replication

Variations

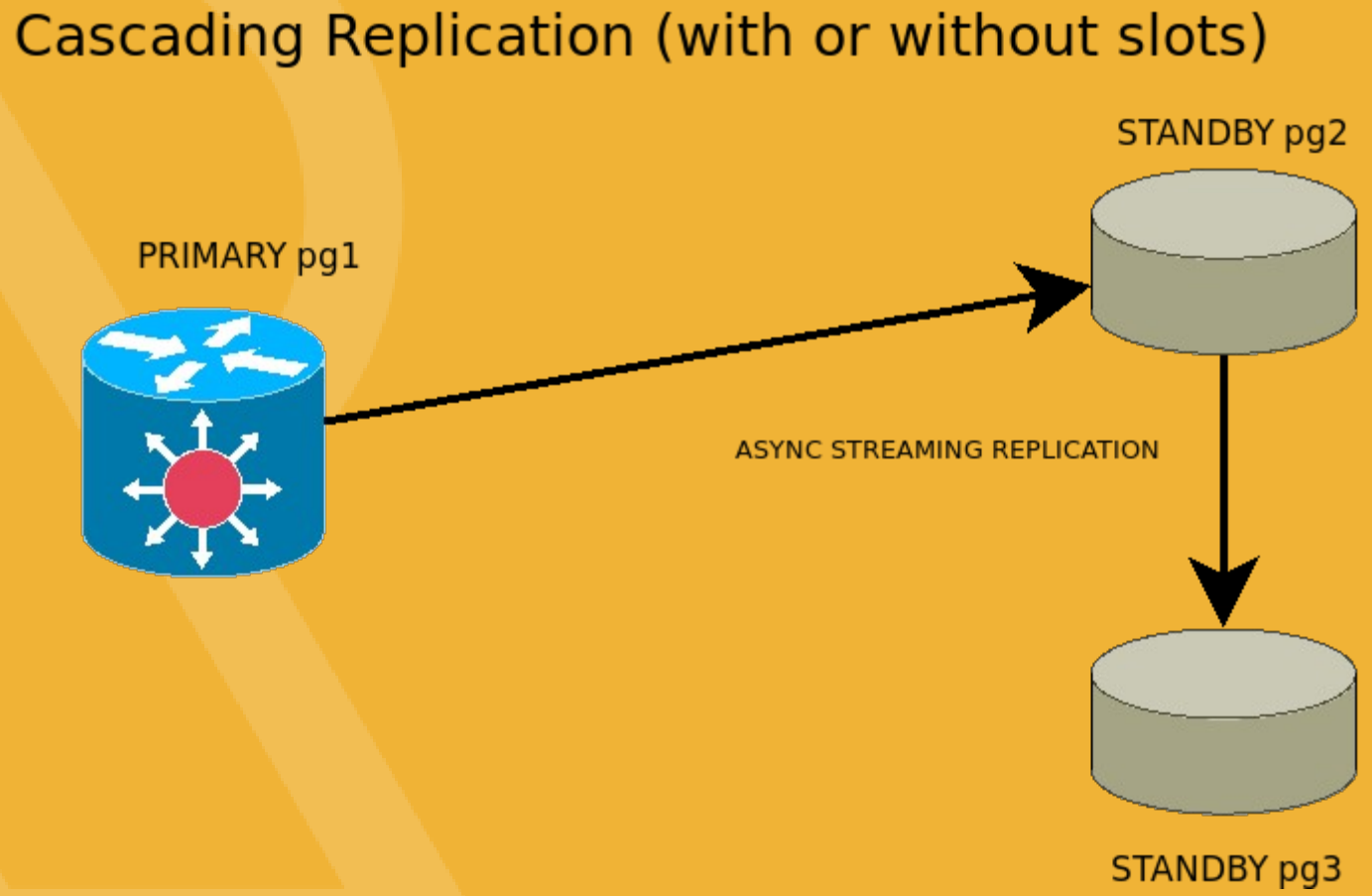
```
# Standby Mode: (requires a server restart)

– Warm Standby (No Read)
alter system set hot_standby=’off’;

– Hot Standby (DEFAULT: Read-Only)
alter system set hot_standby=’on’;
```

Variations Cont'd

Cascading Replication



Cascading Replication

Example

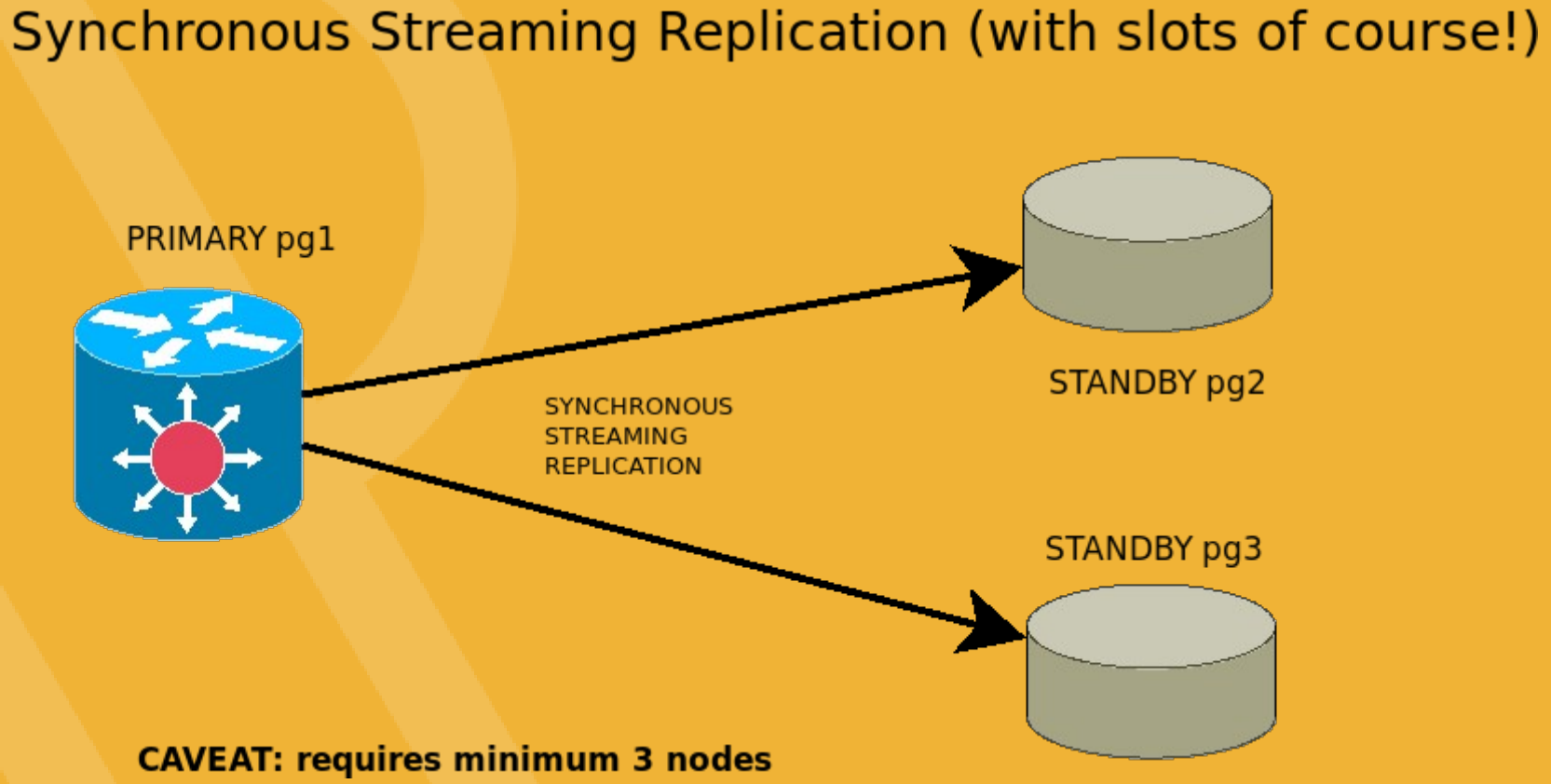
```
Perform BaseBackup

systemctl stop postgresql@14-main
su - postgres
export PGDATA=/var/lib/postgresql/14/main
rm -rf $PGDATA
# host is changed from pg1 to pg2
/usr/bin/pg_basebackup -d 'host=pg2 user=postgres password=postgres port=5432' \
                       --wal-method=stream \
                       -l basebackup \
                       -D /var/lib/postgresql/14/main \
                       -R -P -v \
                       --create-slot \
                       --slot=pg3
```

```
Configure CASCADED REPLICA, valid for pg ver 12+

touch $PGDATA/standby.signal
```

Synchronous Replication



Async vs Sync Replication

```
- Asynchronous replication is non-blocking and returns as soon as the transaction is committed on the PRIMARY.
- Synchronous replication commits a transaction only when the operation completes on the slave(s).

------------------------------------------------------------------------------
#synchronous_standby_names = '' # SYNC ORDER
                                #   num_sync is the number of synchronous standbys
                                #   that transactions need to wait for replies from
                                #
                                # '*' OR 'all'
                                # [FIRST] num_sync ( standby_name [, ...] )
                                # ANY num_sync ( standby_name [, ...] )
                                # standby_name [, ...]
                                #


------------------------------------------------------------------------------
#synchronous_commit = on        # synchronization level;
                                #   off           # synchronous replication is ignored
                                #
                                #   local         # guaranteed data flush only on the primary node
                                #
                                #   remote_write  # commit waits for confirmation from standby
                                #                      writing the record (not yet flushed)
                                #
                                #   remote_apply  # standby replies when the commit record is replayed
                                #
                                #   on            # waits until data is flushed
                                #                      to the transaction log on all hosts

------------------------------------------------------------------------------
cluster_name (string)           # Sets a name that identifies this database cluster.
```

Synchronous Replication

Example

```
HOST=PG2

# TIP: set cluster_name=pg2
/usr/bin/pg_basebackup -d 'host=pg1 user=postgres password=postgres port=5432 application_name=pg2' \
                       --wal-method=stream \
                       -l basebackup \
                       -D /var/lib/postgresql/14/main \
                       -R -P -v \
                       --create-slot \
                       --slot=pg2

echo "cluster_name = 'pg2'" >> /var/lib/postgresql/14/main/postgresql.auto.conf

touch $PGDATA/standby.signal
```

```
HOST=PG3

# TIP: set cluster_name=pg3
/usr/bin/pg_basebackup -d 'host=pg1 user=postgres password=postgres port=5432 application_name=pg3' \
                       --wal-method=stream \
                       -l basebackup \
                       -D /var/lib/postgresql/14/main \
                       -R -P -v \
                       --create-slot \
                       --slot=pg3
echo "cluster_name = 'pg3'" >> /var/lib/postgresql/14/main/postgresql.auto.conf

touch $PGDATA/standby.signal
```

Synchronous Replication

Example Cont'd

```
HOST=PG1

# default = on, requires restart
show synchronous_commit;

# requires reload
alter system set synchronous_standby_names='pg2,pg3';
select pg_reload_conf();


# validate
select application_name,state,sync_state from pg_stat_replication order by 1;
 application_name |   state   | sync_state
------------------+-----------+------------
 pg2              | streaming | sync
 pg3              | streaming | potential
```

```
Example Alternative Replication Wait Modes

alter system set synchronous_standby_names='FIRST 1 (pg3,pg2)';
alter system set synchronous_standby_names='FIRST 2 (pg3,pg2)';
alter system set synchronous_standby_names='ANY 2 (pg3,pg2)';
```

Synchronous Replication Caveat

```
Basebackup behaviour: log shipping versus streaming (with and without slots)

Monitor:
      PRIMARY:
            select * from pg_stat_replication;
            select * from pg_replication_slots;
            select * from pg_get_replication_slots();
      REPLICA: postgres logs
            cat <postgres log> | grep -E 'ERROR|FATAL'

Slot administration:
      CLI:
            pg_receivewal
      FUNCTIONS:
            pg_drop_replication_slot
            pg_copy_physical_replication_slot
            pg_create_physical_replication_slot
            pg_replication_slot_advance
```

Logical Replication

About

A method of replicating data objects and their changes, based upon their replication identity (usually a primary key). While streaming replication uses exact block addresses, logical replication is byte-by-byte.

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.

Capabilities

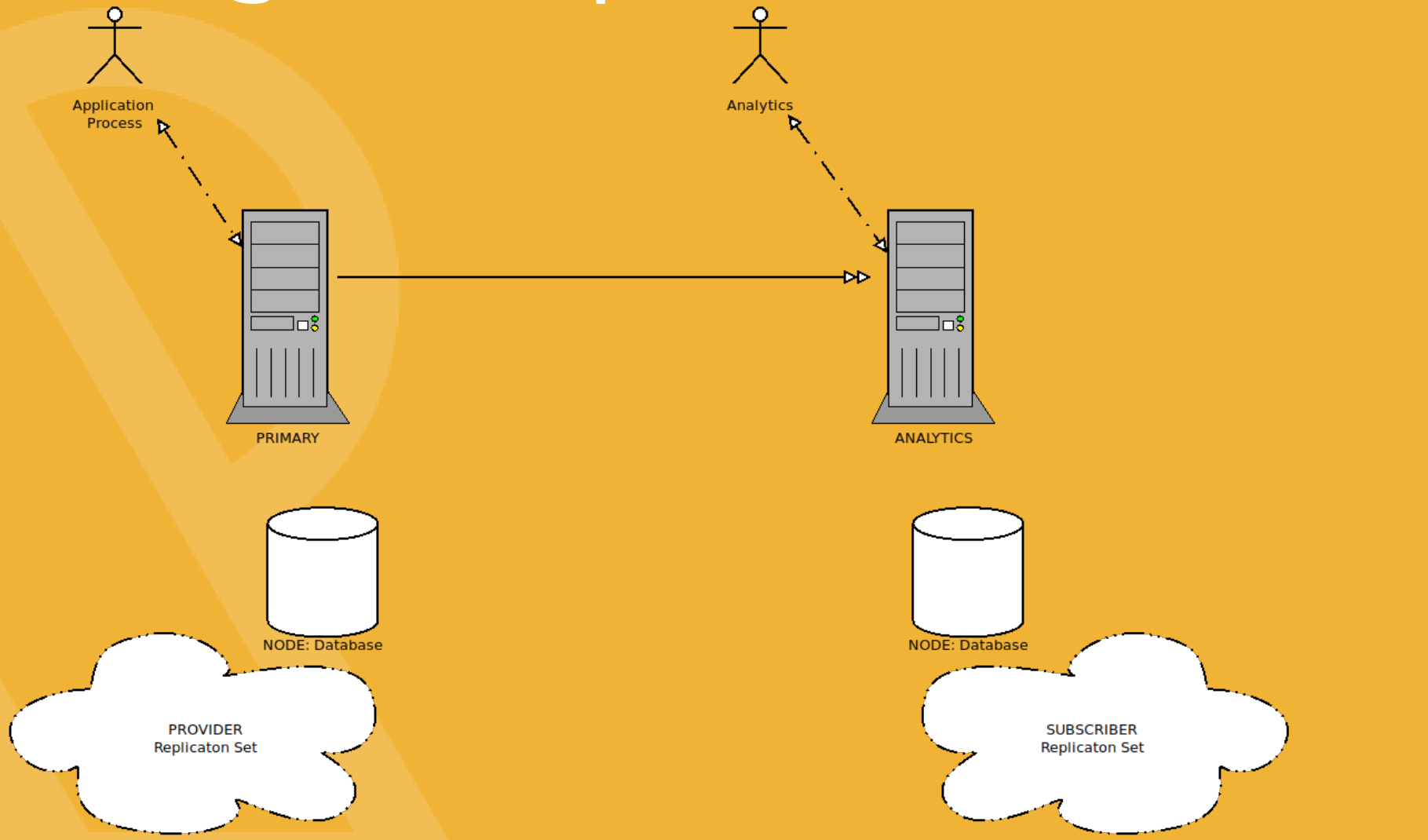
- UPGRADE: Upgrade PostgreSQL from 9.4 to 9.5, without downtime
- SCALE OUT: Copy all or a selection of database tables to other nodes in a cluster
- AGGREGATE: Accumulate changes from sharded database servers into a Data Warehouse
- INTEGRATE: Feed database changes in real-time to other systems
- PROTECT: Provide backup or high availability for clusters, replacing earlier technologies

Method

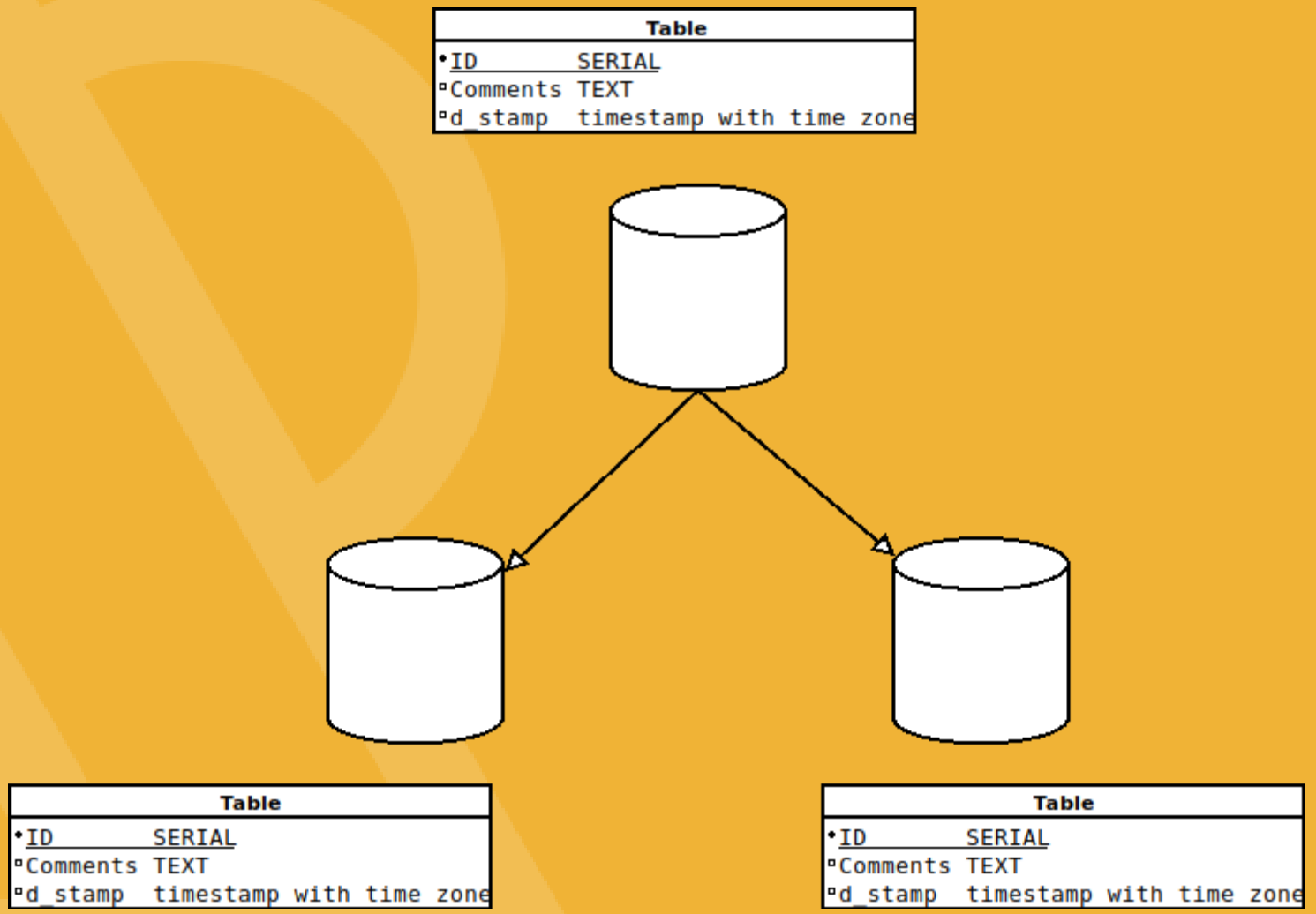
Logically replicating a table starts with snapshot of the data from the publisher database and copying that to the subscriber database.

Changes on the publisher are sent to the subscriber real-time and is applied in the same order.

Logical Replication



Logical Replication



Logical Replication

```
Method
- create the database(s)
- create two schemas
- create and populate table(s)
- create the logical node(s)
- create replication set(s)
- subsribe to provider(s)
- peform DML operations
```

```
CREATE PUBLICATION name
    [ FOR TABLE [ ONLY ] table_name [ * ] [, ...]
      | FOR ALL TABLES ]
    [ WITH ( publication_parameter [= value] [, ... ] ) ]


CREATE SUBSCRIPTION subscription_name
    CONNECTION 'conninfo'
    PUBLICATION publication_name [, ...]
    [ WITH ( subscription_parameter [= value] [, ... ] ) ]
```

Logical Replication

Example 1

```
-- host:pg1

drop database if exists db01;
create database db01;
\c db01
alter role postgres with password 'postgres';
create schema a;
create table a.t1(i serial primary key,comments text);
create publication provider1 FOR TABLE a.t1;
```

```
-- host:pg3

create database db01;
alter role postgres with password 'postgres';
\c db01
create schema a;
create table a.t1(i serial primary key,comments text);
create subscription mysub
         connection 'host=pg1 port=5432 user=postgres dbname=db01 password=postgres'
        publication provider1
               with (enabled = true);
```

Logical Replication

Example 2

```
export PGUSER=postgres PGPASSWORD=postgres
```

```
createdb -h pg1 db02
createdb -h pg3 db02
/usr/bin/pgbench -h pg1 -i db02
psql 'host=pg1 dbname=db02' -c 'alter table pgbench_history add primary key(tid,bid,aid,delta,mtime);'
pg_dump -s -h pg1 db02 | psql -h pg3 db02
```

```
pg1: create PUBLICATION, execute on host pg1, database db02

set search_path=public;
create publication publication1 for all tables;
```

```
pg3: create SUBSCRIPTION, execute on host pg3, database db02

create subscription subscript_set1
    connection 'host=pg1 dbname=db02 user=postgres password=postgres'
    publication publication1
        with (copy_data = true, create_slot = true, enabled = true, slot_name = myslot1);
```

```
Execute benchmarking on host pg1, database db02

/usr/bin/pgbench -h pg1 -c 4 -j 2 -T 100 -b tpcb-like db02
```

Logical Replication

Caveat

- DDLs not supported
- No Replication Queue Flush (Failover is problematic)
- No Cascaded Replication
- One unique index/constraint/pk per table
- Permissions (remote access by subscriber)
- Primary key must exist
- Sequences
- Triggers
- Truncate command is not propagated
- Unlogged/temporary tables not supported

Thank You!

Robert.bernier@percona.com