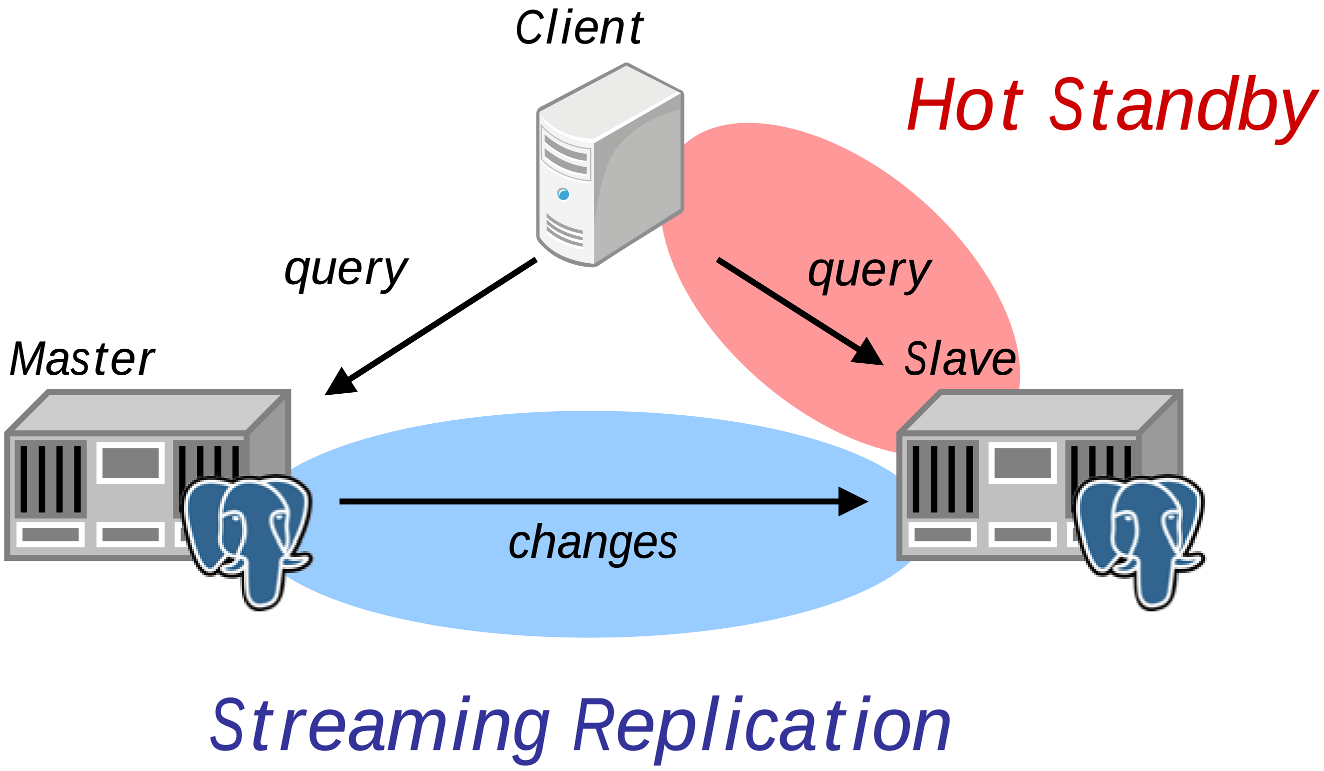
Streaming Replication

&

Hot Standby

v8.5～



Client

query

query

Hot Standby

Master

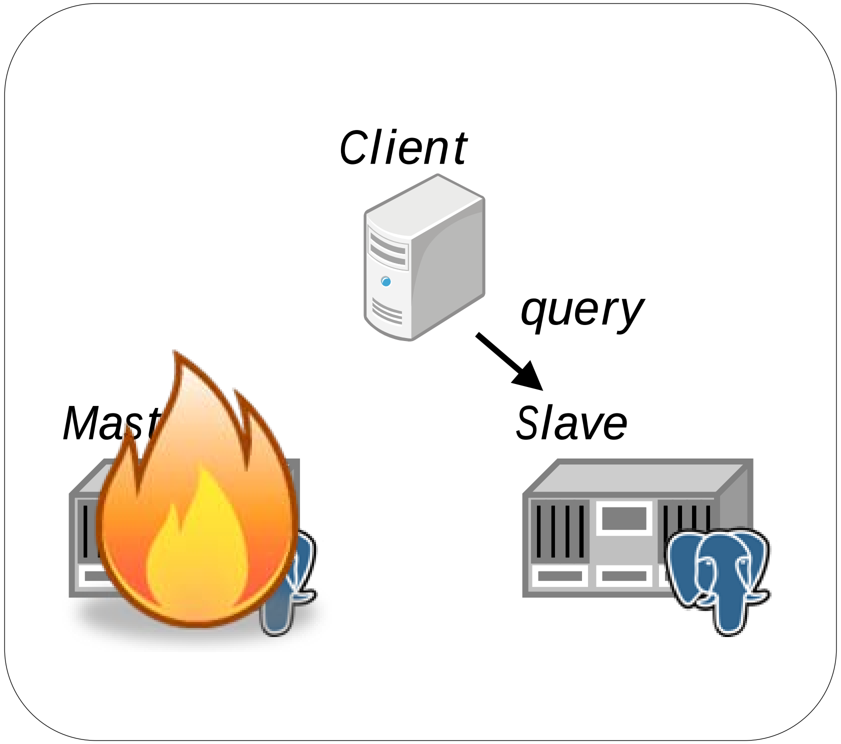
Slave

changes

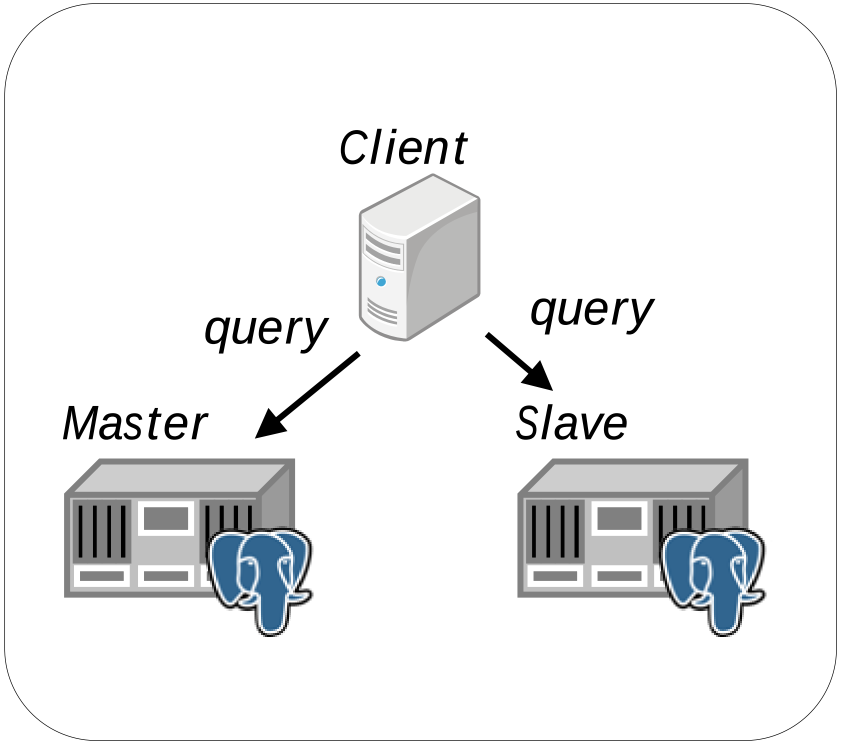
Streaming Replication

Why Streaming Replication & Hot Standby?

High Availability



Load Balancing



Schedule

1. Talk: Streaming Replication
2. Talk: Hot Standby
3. Demo

Streaming Replication

Masao Fujii
NTT OSS Center

Fujii Masao

- *Database engineer at NTT OSS Center*
- *Support and consulting*
- *Implementing Streaming Replication*

History

Historical policy

- *Avoid putting replication into core Postgres*
- *No "one size fits all" replication solution*

Replication War!?

rubyrep

PL/Proxy

DBmirror

Postgres-2

warm-standby

Cybercluster

Slony-I

PGCluster-II

syncreplicator

Mammoth

Postgres-R

Londiste

pgpool

Sequoia

Bucardo

twin

PyReplica

pgpool-II

RepDB

GridSQL

PGCluster

PostgresForest

No default choice

- *Too complex to use for simple cases*
- *vs. other dbms*

Proposal of built-in replication

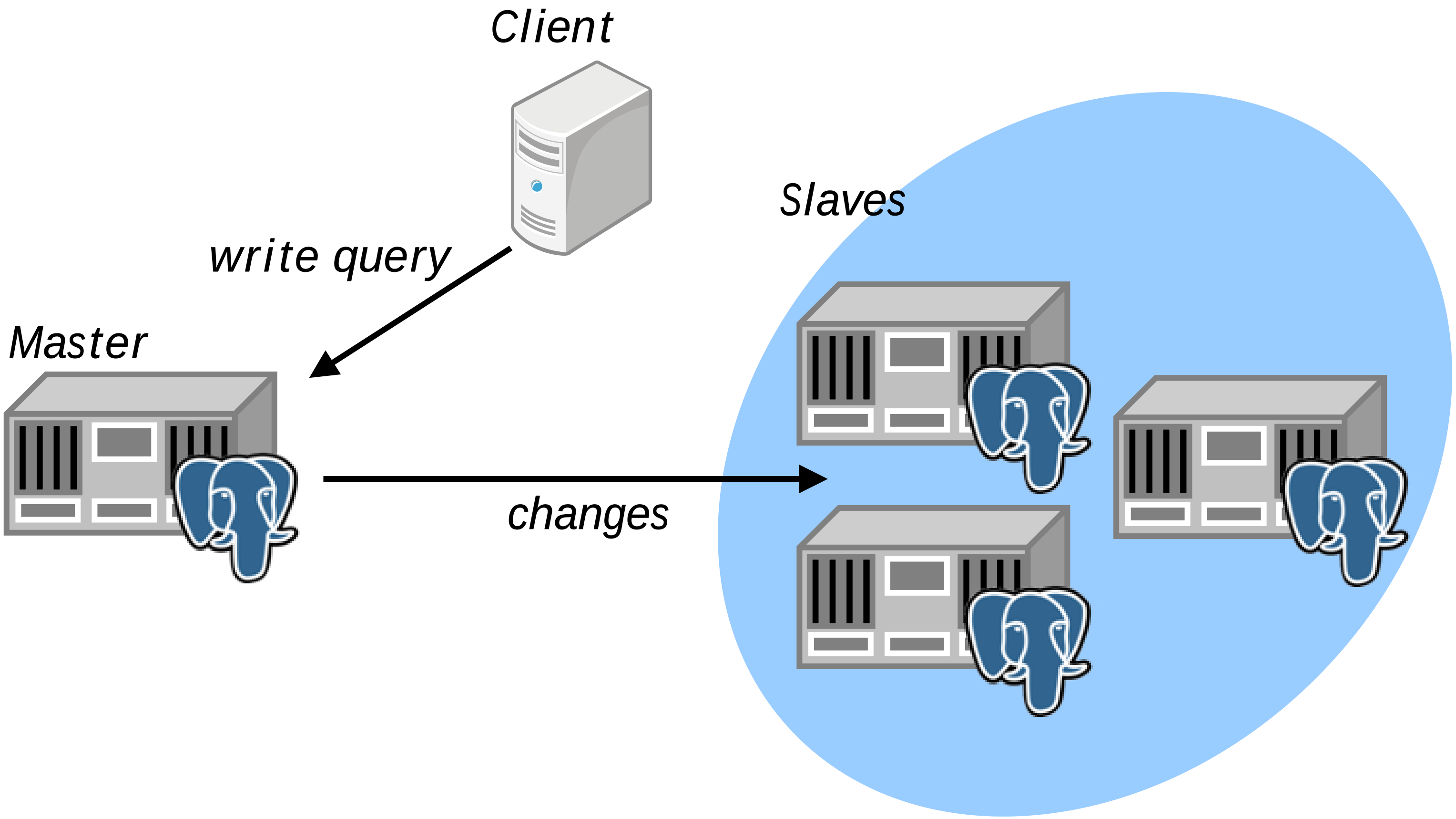
- *by NTT OSSC @ PGCon 2008 Ottawa*

Core team statement

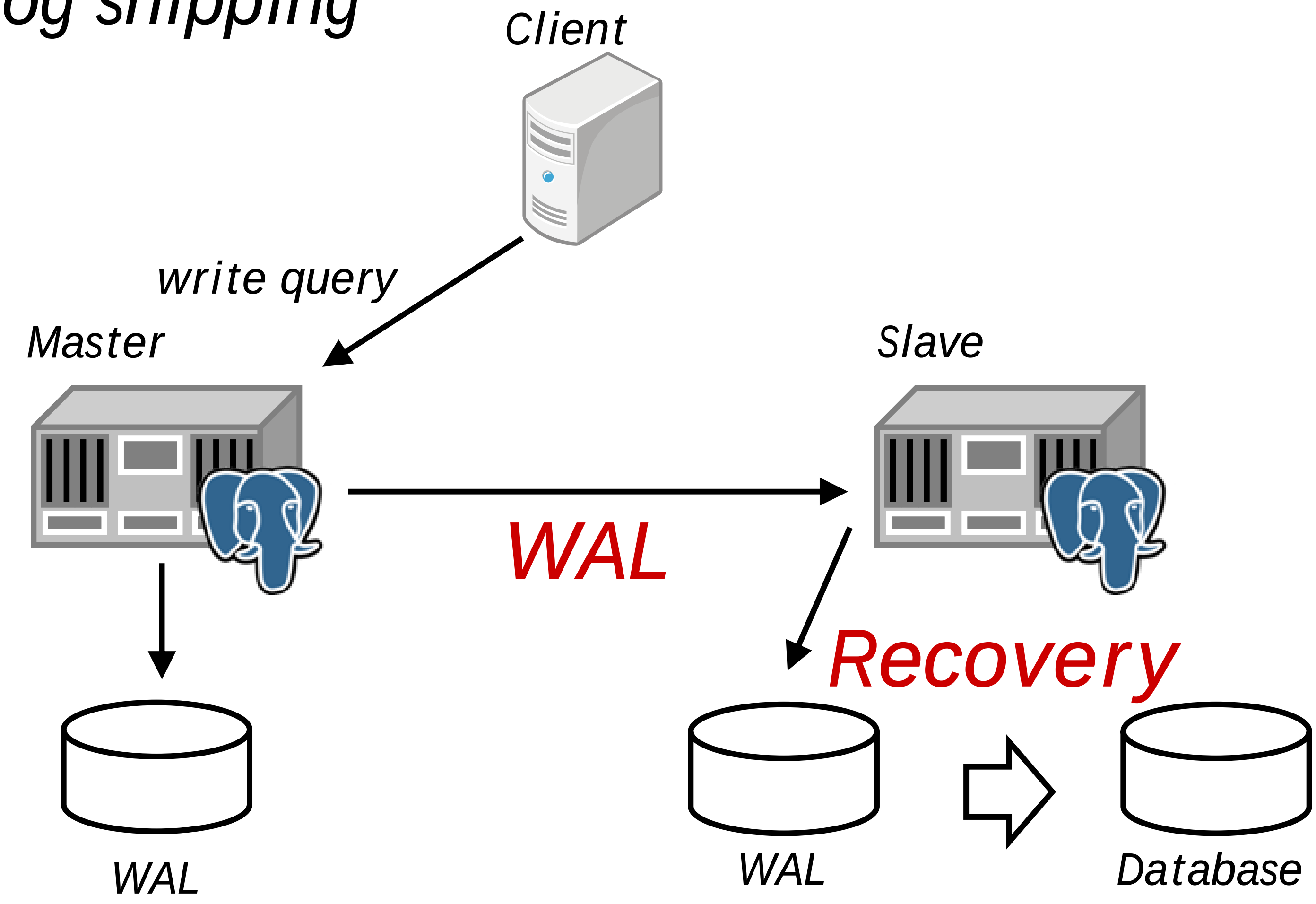
- *It is time to include a **simple**, reliable basic replication feature in the **core** system*
- *NOT replace the existing projects*

Features

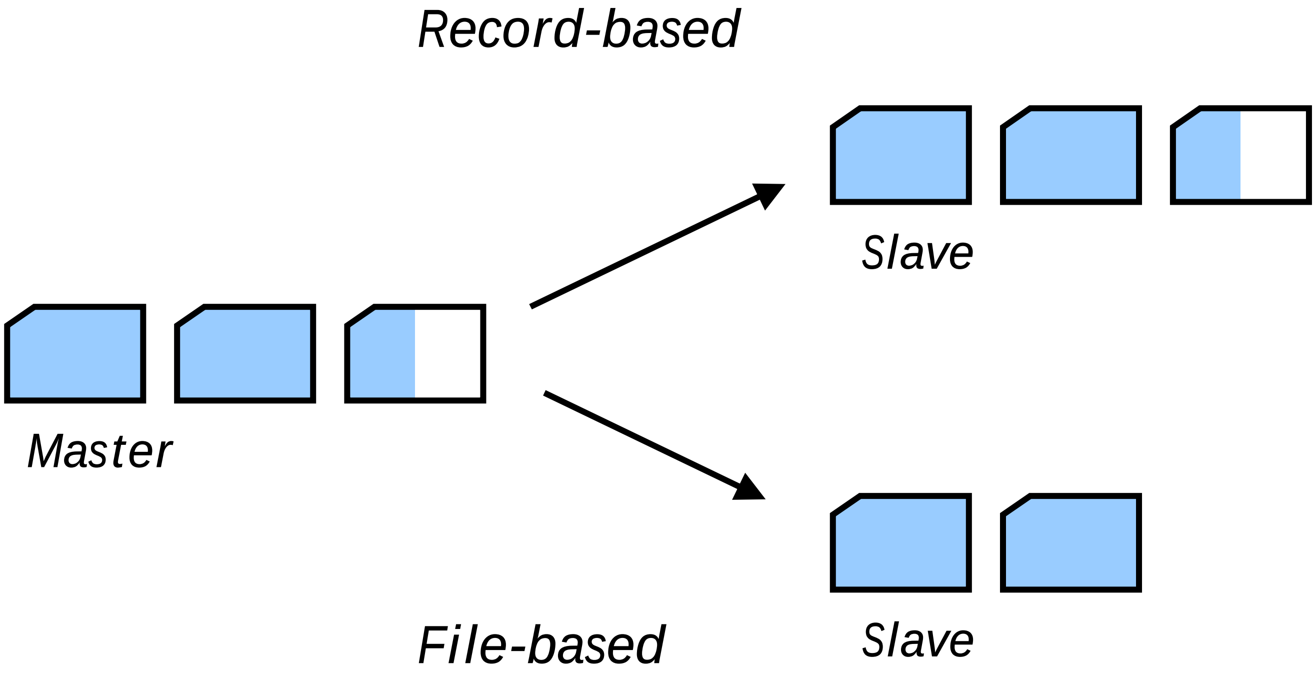
Master - Slaves



Log shipping



Record-based log shipping


Record-based

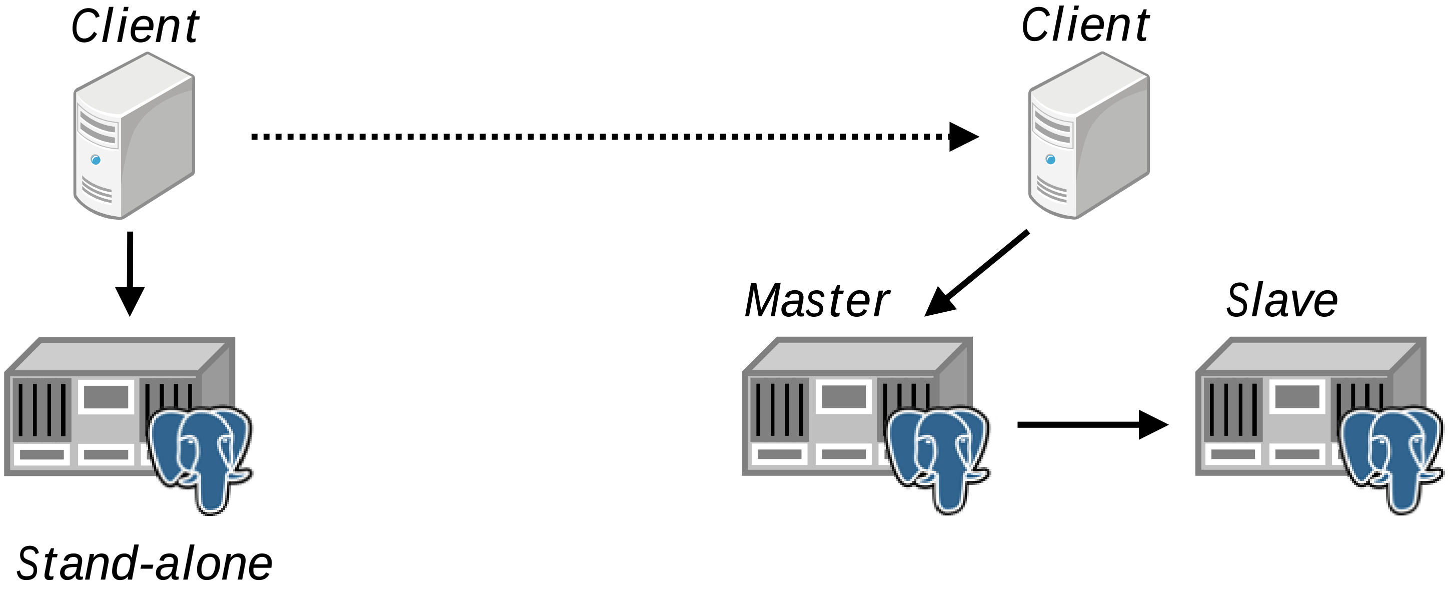
Slave

Master

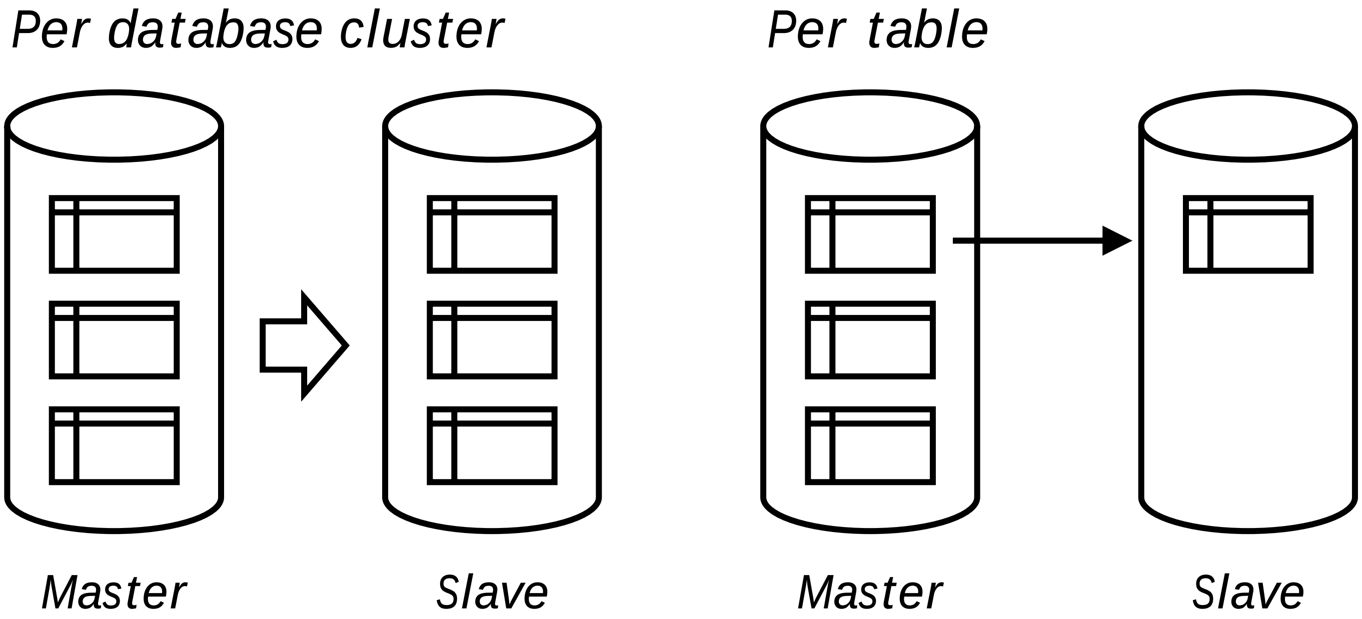
File-based

Slave

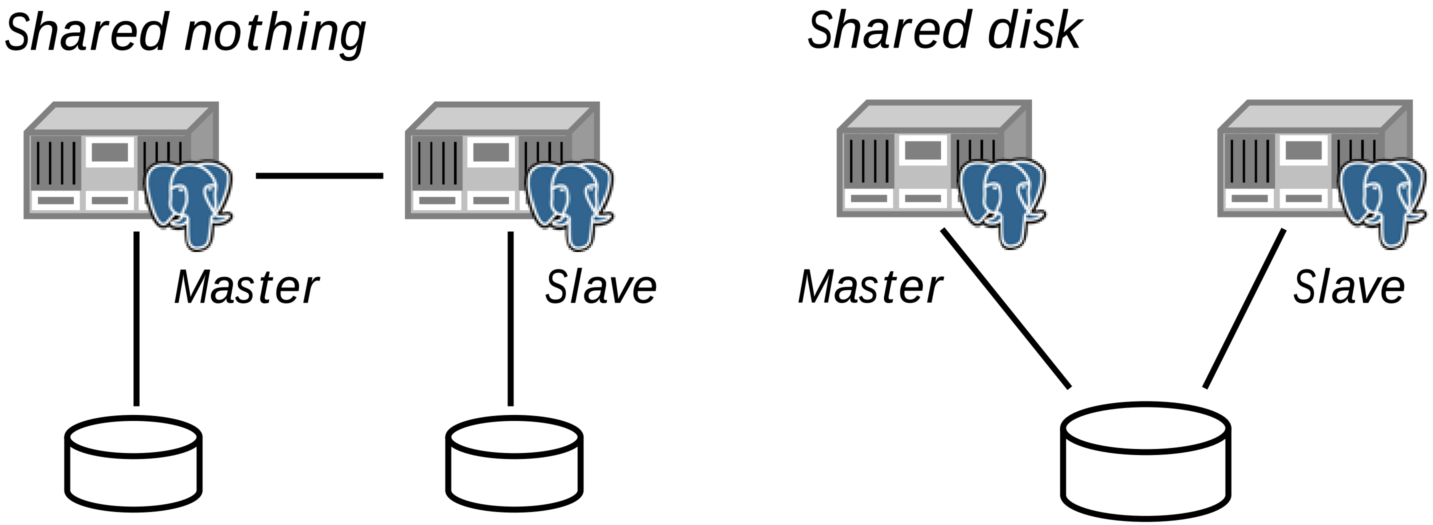
No migration required



Per database cluster granularity



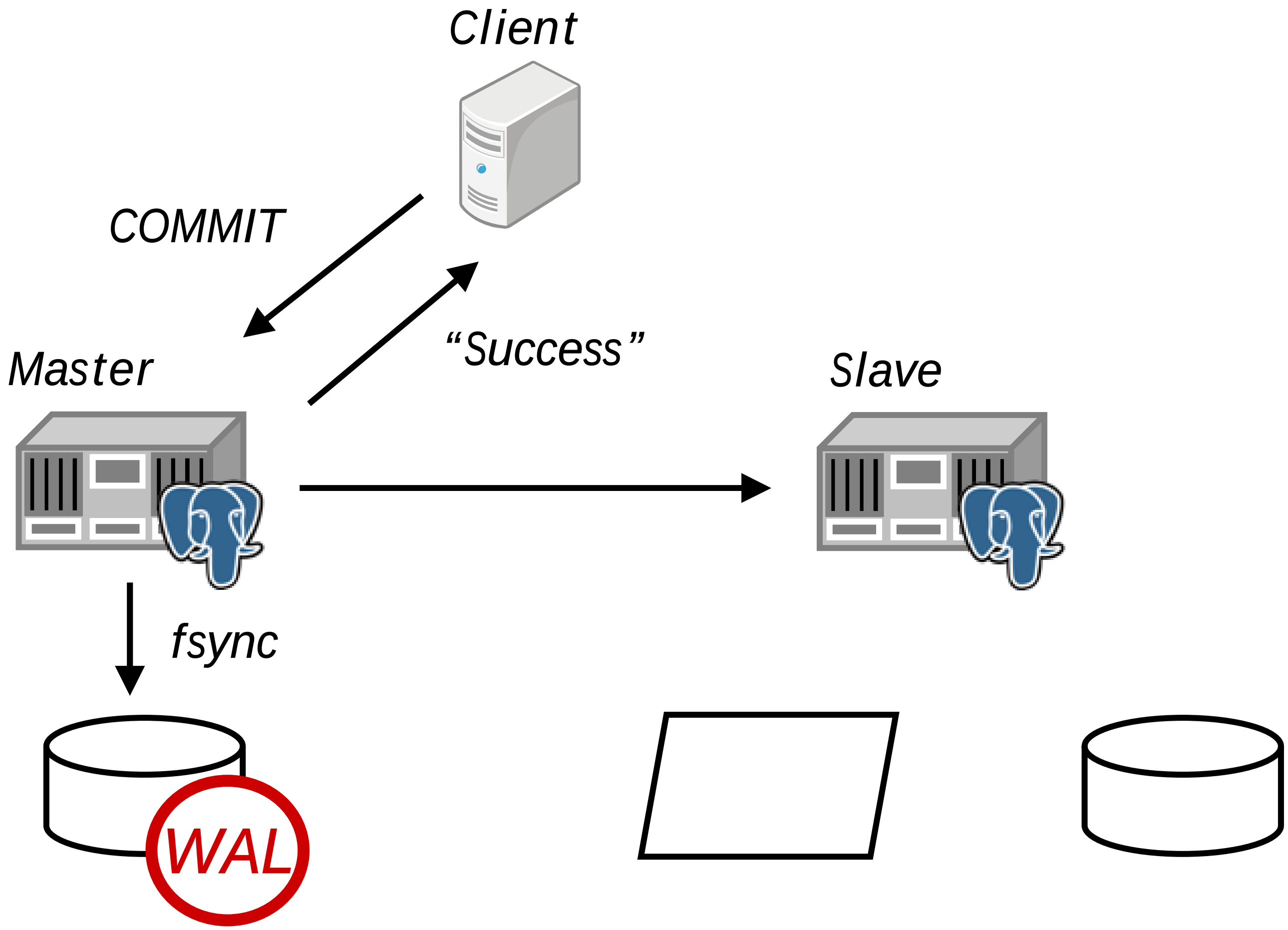
Shared nothing



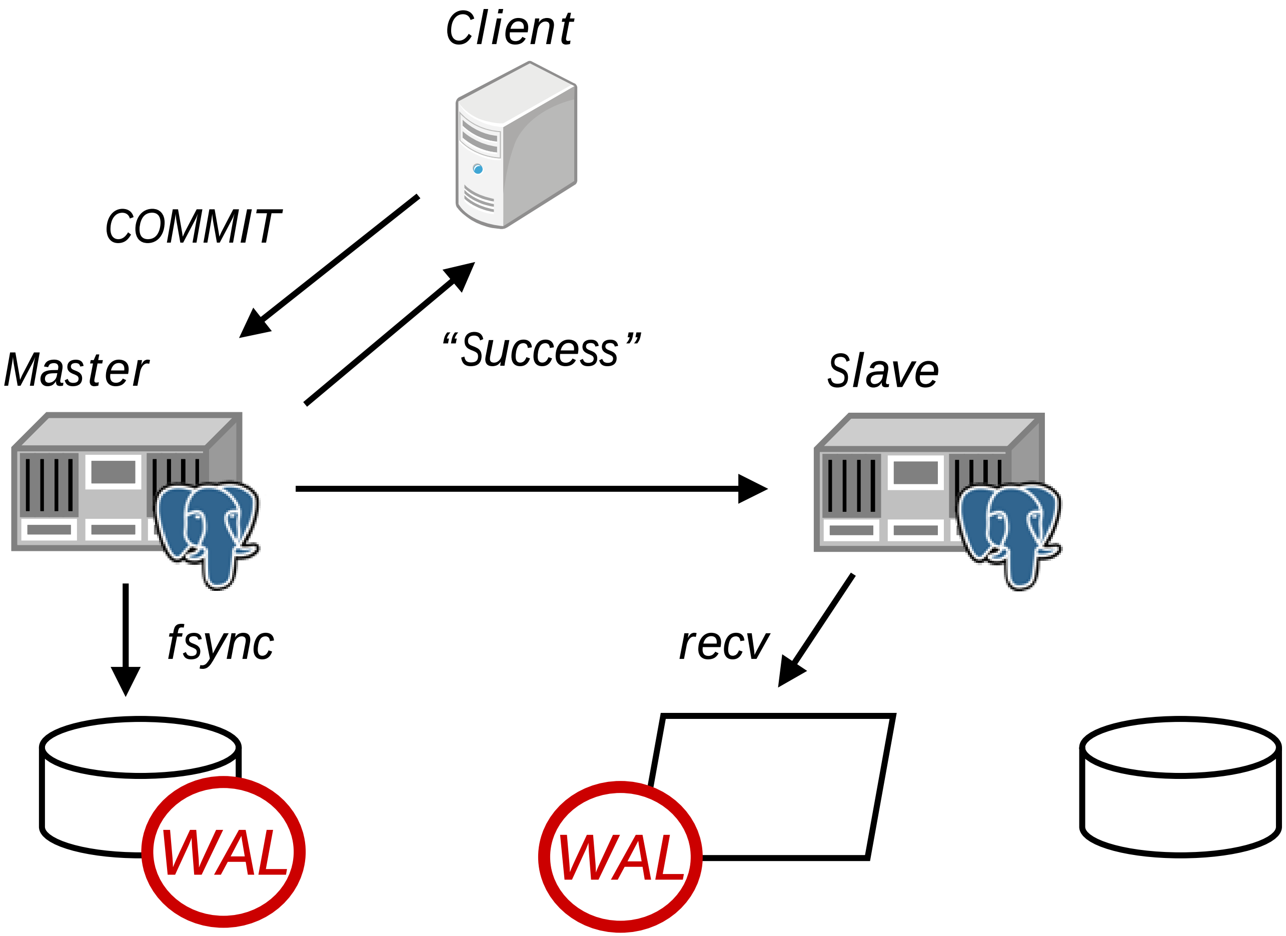
Synchronization modes

- *async*
- *recv*
- *fsync*
- *apply*

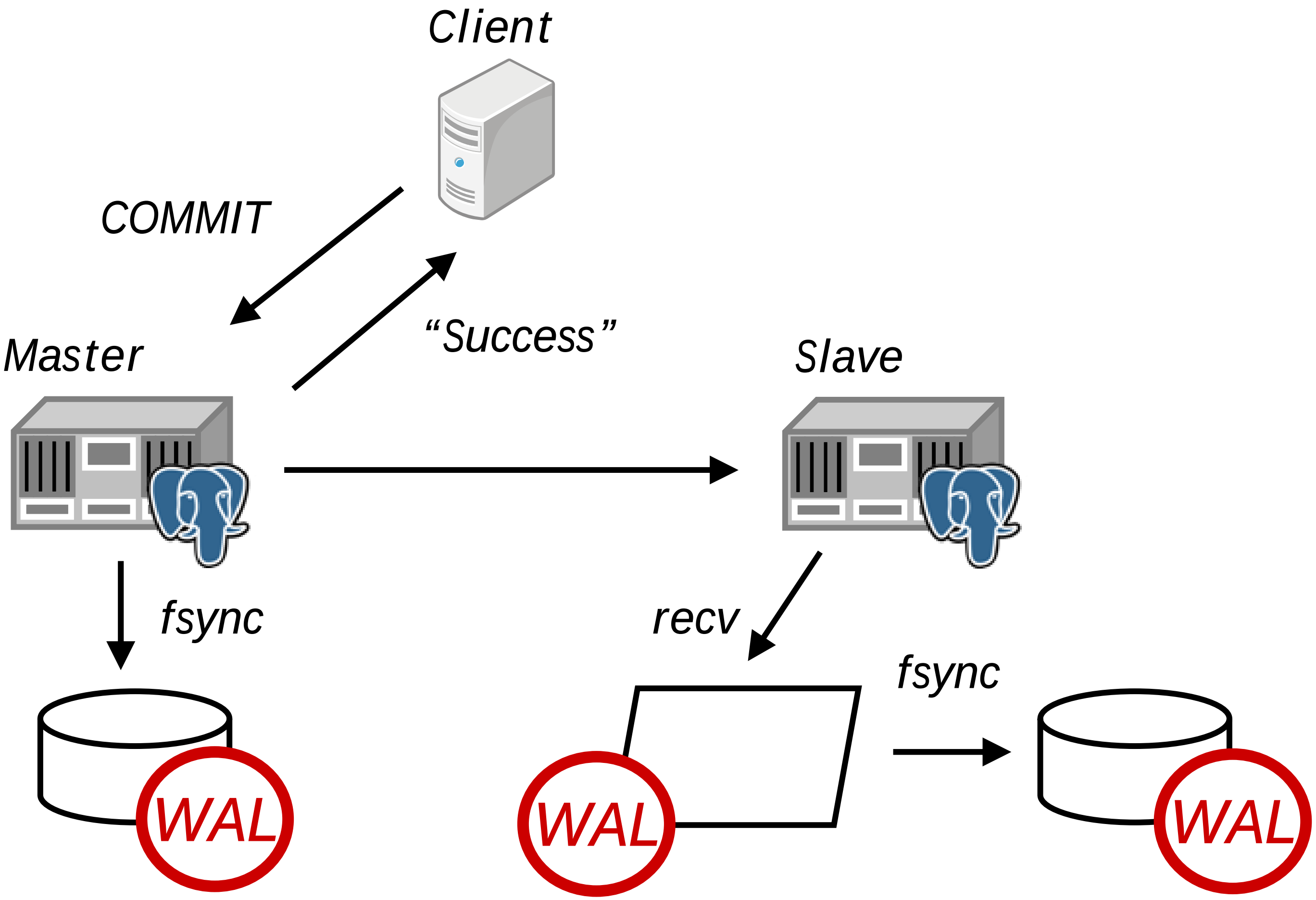
async mode



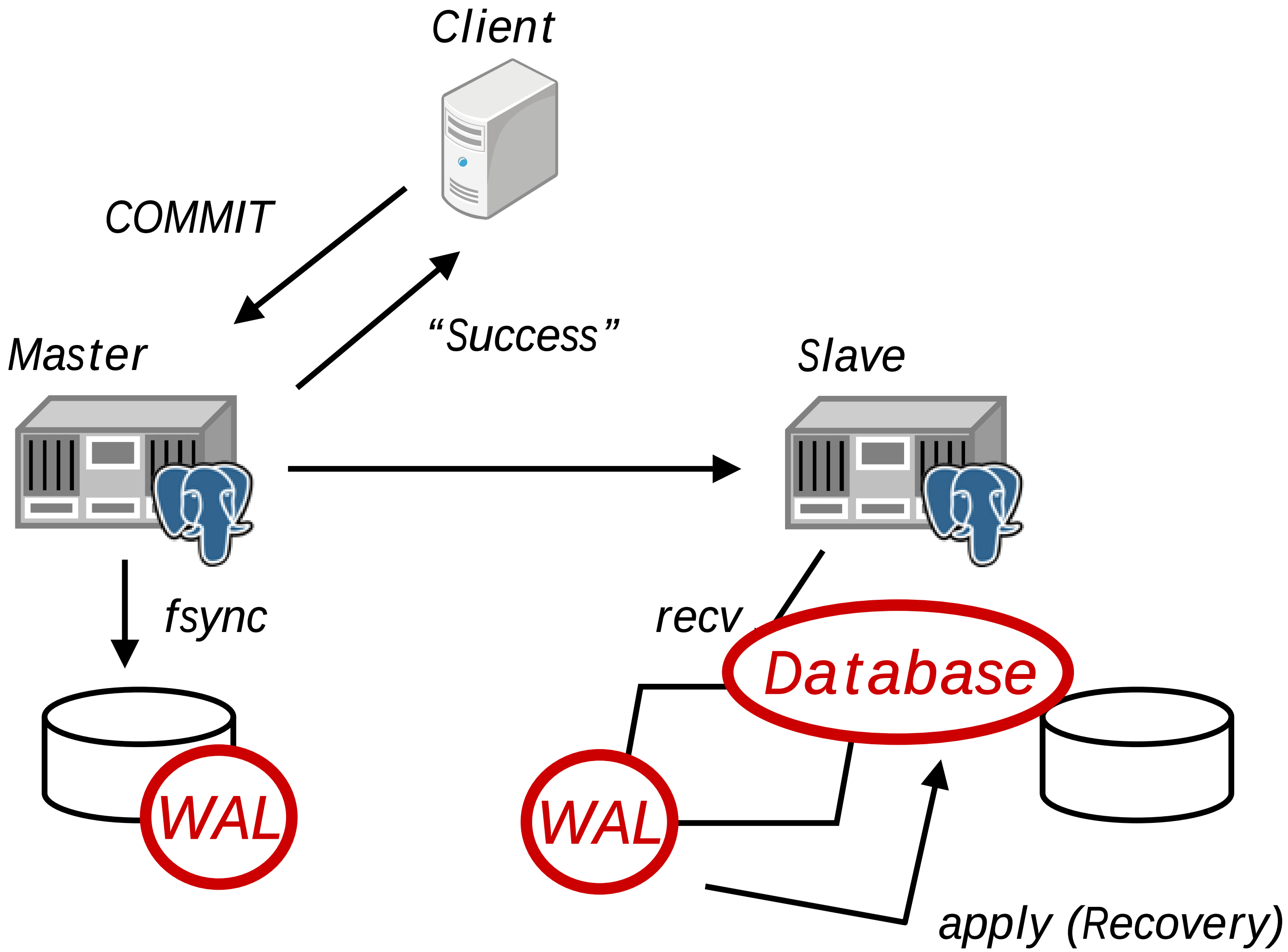
recv mode



fsync mode


Client
COMMIT
"Success"
Master
Slave
fsync
recv
fsync
WAL
WAL
WAL

apply mode


Client
COMMIT
"Success"
Master
Slave
fsync
recv
Database
WAL
WAL
apply (Recovery)

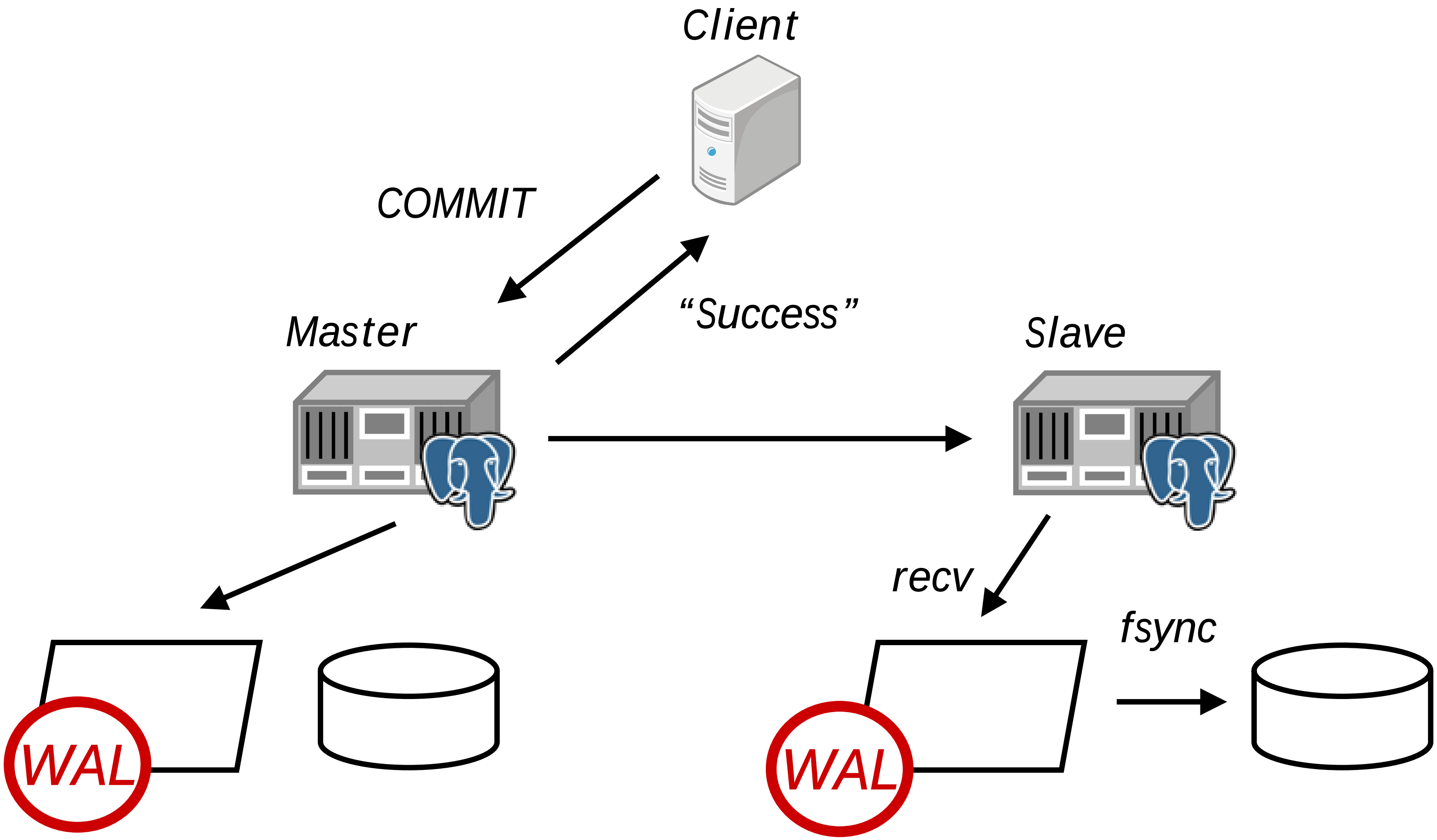
Synchronization mode

modes	Master	Slave		
	fsync	recv	fsync	apply
async	✓			
recv	✓	✓		
fsync	✓	✓	✓	
apply	✓	✓		✓

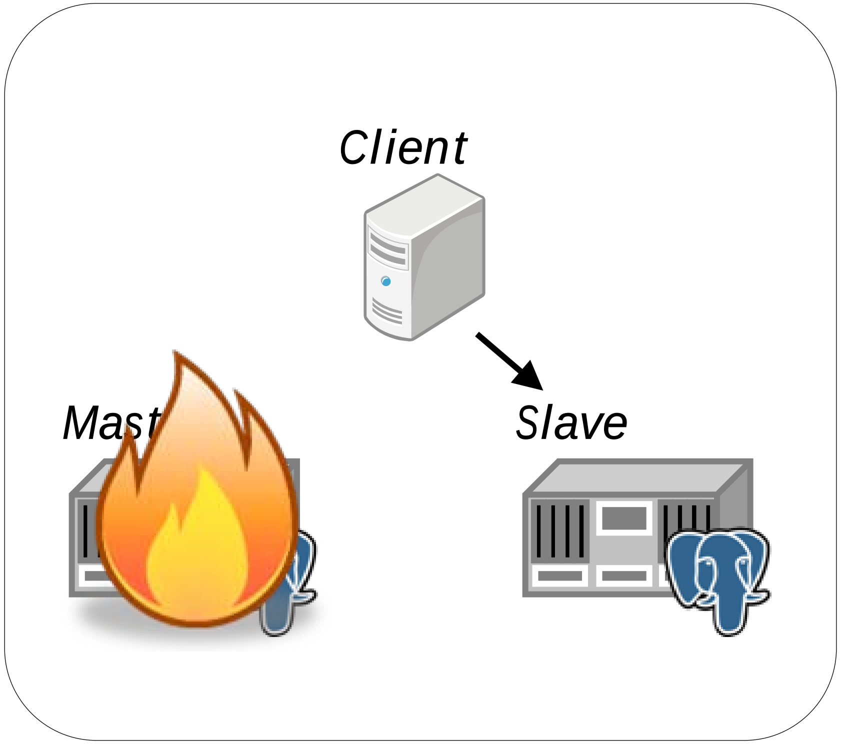
↑ fast

↓ durable

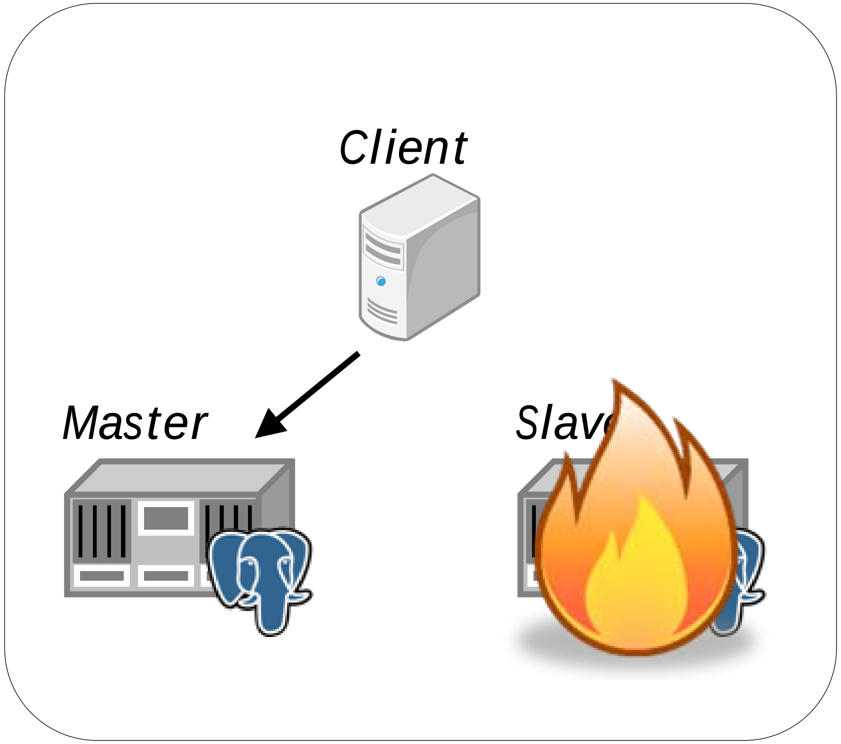
My favorite mode



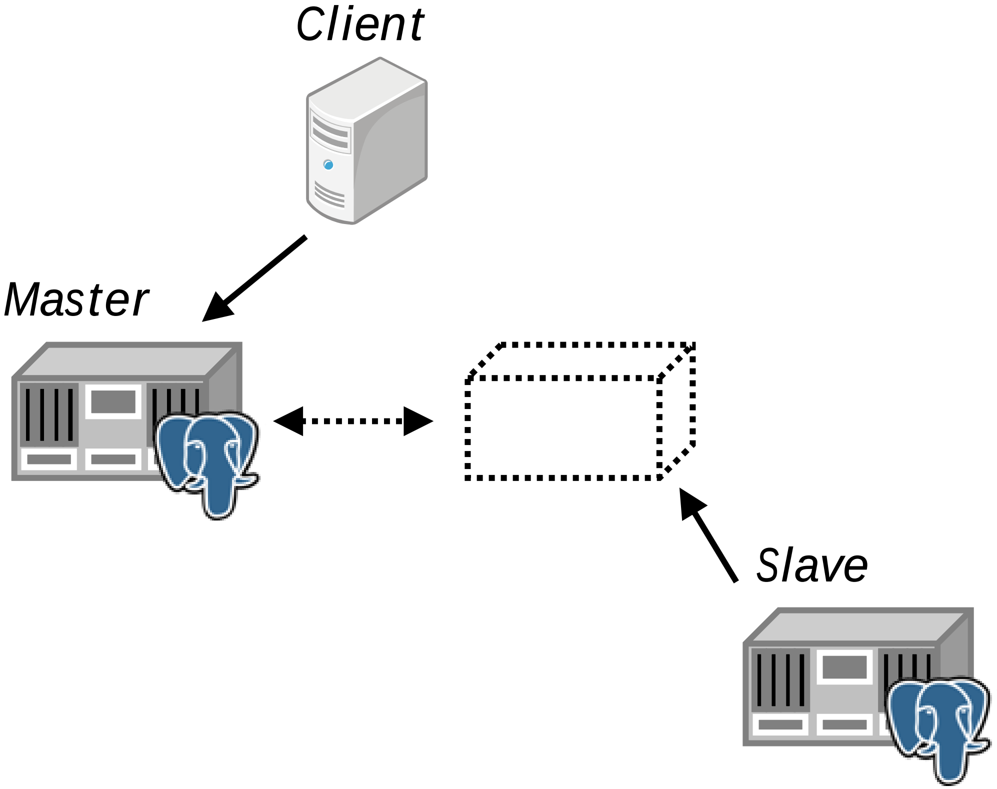
Fail Over



Split



Online Re-sync



Built-in

- *Easy to install and use*
- *Highly active community*

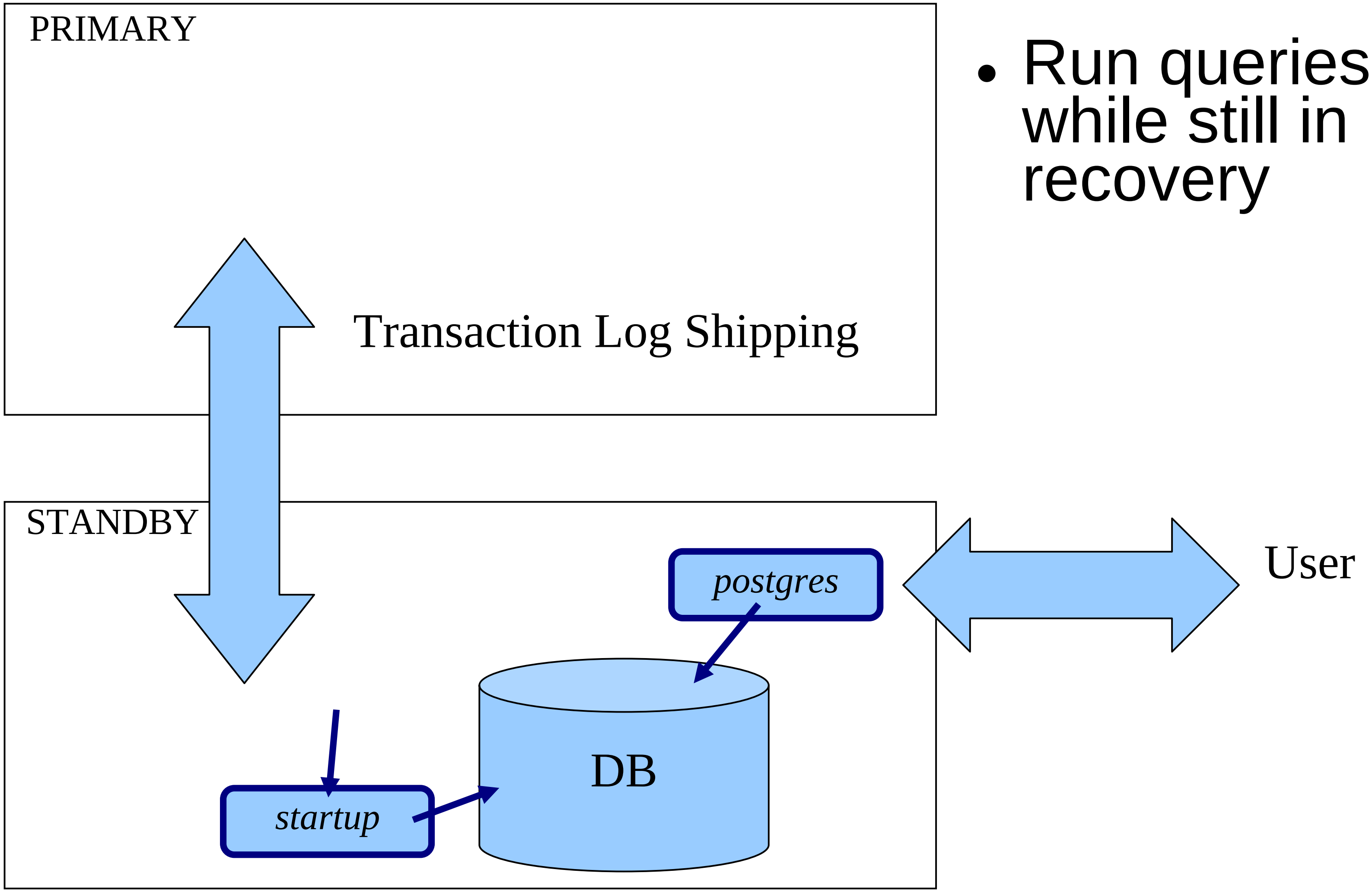
2ndQuadrant
Professional PostgreSQL

Hot Standby

Simon Riggs
2nd Quadrant

simon@2ndQuadrant.com

Hot Standby



- Run queries while still in recovery

Hot Standby Overview

- *Allows users to connect in read-only mode*
  - *Allowed: SELECT, SET, LOAD, COMMIT/ROLLBACK*
  - *Disallowed: INSERT, UPDATE, DELETE, CREATE, 2PC,*
  - *SELECT ... FOR SHARE/UPDATE, nextval(), LOCK*
  - *No admin commands:*
    ANALYZE, VACUUM, REINDEX, GRANT
- *Simple configuration*
  - *recovery_connections = on # default on*
- *Performance Overhead*
  - *Master: <0.1% overhead from additional WAL*
  - *Standby: 2% CPU overhead*
- *Queries continue running when exit recovery*

Hot Standby Query Conflicts

- *Master: Connections can interfere and deadlock*
- *Standby: Queries can conflict with recovery*
  - *Recovery always wins*
- *Causes of conflicts*
  - *Cleanup records (HOT, VACUUM)*
  - *Btree cleanup records are a problem!*
  - *DROP DATABASE, DROP TABLESPACE*
- *Conflict resolution*
  - *Wait, then Cancel – set with max_standby_delay*

How does it work?

- *Read-only transactions forced*
- *Snapshot data emulated on standby*
  - *Minimal information inferred from WAL*
- *Locks held only for AccessExclusiveLocks*
- *Cache invalidations*
- *Careful analysis of conflicts*

Project Deliverables

- *Virtual Transactions (8.3) (Florian/Tom)*
- *Atomic Subtransactions (8.4) (Simon)*
- *Database consistent state (8.4) (Simon/Heikki)*
- *Bgwriter active during recovery (8.4) (Simon/Heikki)*
- *Removal of DB/Auth Flat File (8.5) (Tom)*
- *Main Hot Standby patch (8.5) (Simon/Heikki)*
- *Removal of Non-Transactional Cache Inval (Tom!)*
- *Advanced PITR functions (8.5) (Simon)*

Project Overview

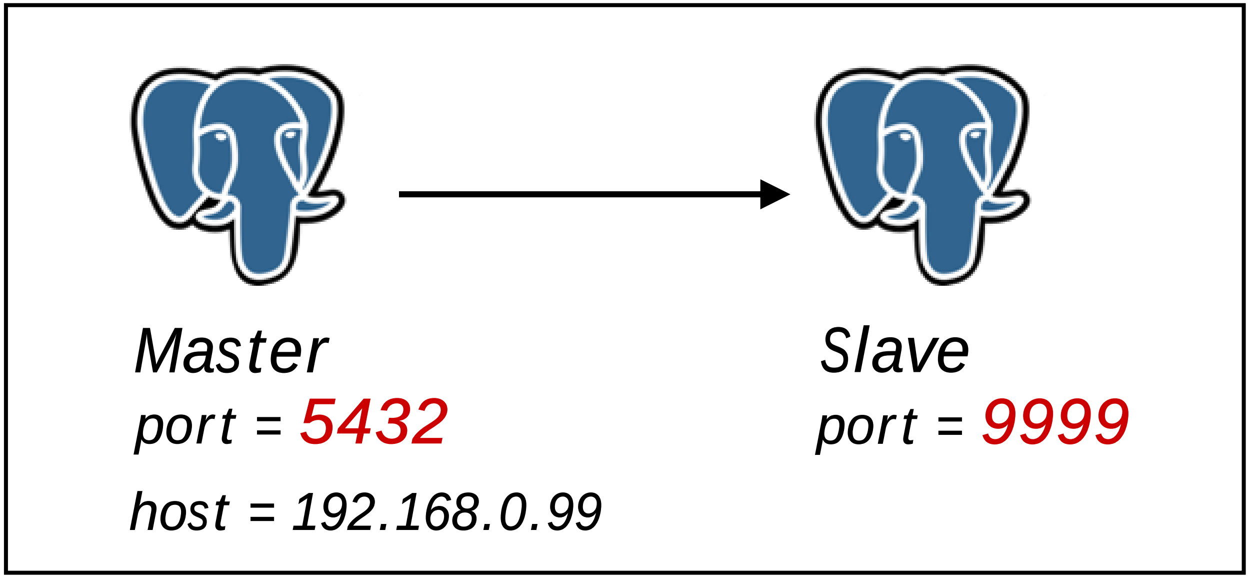
- *Touches ~80 files, >10,000 lines*
- *Effort*
  - *Analysis & Dev ~7 man months from Simon*
  - *Testing by 5 staff in 2ndQuadrant, led by Gianni Ciolli*
  - *Lengthy review by Heikki Linnakangas*
- *Changes*
  - *Around 50% of bugs found by code inspection*
  - *> 50 changes and enhancements as a result of refactoring, review and discussion*

Demo

Scenario

- *Configuration*
- *Checking of basic features*
- *Failover*

Configuration



Master
port = 5432
host = 192.168.0.99

Slave
port = 9999

$HOME

```
$HOME
├── master            -- $PGDATA
├── archive_master    -- archival area
├── slave             -- $PGDATA
└── archive_slave     -- archival area
```

Configuration

1. Create the initial database cluster in the master as usual

```
$ initdb –D master -–locale=C  --encoding=UTF8
```

2. Enable XLOG archiving

```
$ mkdir archive_master
$ emacs master/postgresql.conf
archive_mode         = on
archive_command   = ‘cp %p ../archive_master/%f’
```

Configuration

3. Set the maximum number of concurrent connections from the slaves

```
$ emacs master/postgresql.conf
max_wal_senders = 5
```

4. Set up connections and authentication

```
$ emacs master/postgresql.conf
listen_addresses = ‘192.168.0.99’

$ emacs master/pg_hba.conf
host  replication  postgres  192.168.0.99/32  trust
```

Configuration

5. Start postgres on the master

```
$ pg_ctl –D master start
```

6. Make a base backup, load it onto the slave

```
$ psql –p5432 –c”SELECT pg_start_backup(‘demo’, true)”
$ cp –r master slave
$ psql –p5432 –c”SELECT pg_stop_backup()”
```

Configuration

7. Change the slave’s configuration

```
$ rm slave/postmaster.pid
$ mkdir archive_slave
$ emacs slave/postgresql.conf
port                = 9999
archive_command     = ‘cp %p ../archive_slave/%f’
```

Configuration

8. Create a recovery.conf in the slave

```
$ emacs slave/recovery.conf
standby_mode          = ‘on’
primary_conninfo      = ‘host=192.168.0.99 port=5432
      user=postgres’
trigger_file          = ‘../trigger’
```

9. Start postgres on the slave

```
$ pg_ctl –D slave start
```

Checking of basic features

- Session1 on master

```
$ psql –p5432
=# CREATE TABLE demo (i int);
=# INSERT INTO demo VALUES (generate_series(1,100));
     //write queries can be executed on master
=# SELECT count(*) FROM demo;
     //read queries also can be executed on master
```

- Session1 on slave

```
$ psql –p9999
=# SELECT count(*) FROM demo;
     //read queries can be executed on slave
=# INSERT INTO demo VALUES (9999);
     //error occurs: write queries cannot be executed on
     slave
```

Correct handling of snapshots

- Session1 on slave

```
=# BEGIN;
=# SET TRANSACTION ISOLATION LEVEL SERIALIZABLE;
        //cannot see the transaction which starts after this
=# SELECT count(*) FROM demo;
```

- Session1 on master

```
=# INSERT INTO demo VALUES(generate_series(1, 100));
```

- Session1 on slave

```
=# SELECT count(*) FROM demo;
        //result=100, cannot see the recent insertion on master
        because of serializable isolation level
```

- Session 2 on slave

```
=# SELECT count(*) FROM demo;
        //result=200, the recent insertion is visible
```

Lock propagation

- Session1 on master

```
=# BEGIN;
=# LOCK TABLE demo;
=# SELECT pg_switch_xlog();
        //required to ship the WAL of “LOCK TABLE” to slave
```

- Session1 on slave

```
=# SELECT count(*) FROM demo;
        //sleep until “LOCK TABLE” is committed on master
```

- Sessionn2 on slave

```
=# SELECT current_query, waiting FROM pg_stat_activity;
        //shows query waiting
```

- Session1 on master

```
#= COMMIT;
        //the waiting query gets up
```

Failover

- *Let’s see the query is still running when failover completes*

- *Session1 on slave*

```
=# SELECT pg_sleep(20);
```

- *Kill the master’s postmaster*

```
$ pg_ctl –D master –mi stop
```

- *Bring the slave up*

```
$ touch trigger
$ psql –p9999
#= SELECT current_query FROM pg_stat_activity;
        //can see pg_sleep is still running
#= INSERT INTO demo VALUES(9999);
        //write queries can be executed because slave becomes
        master
```

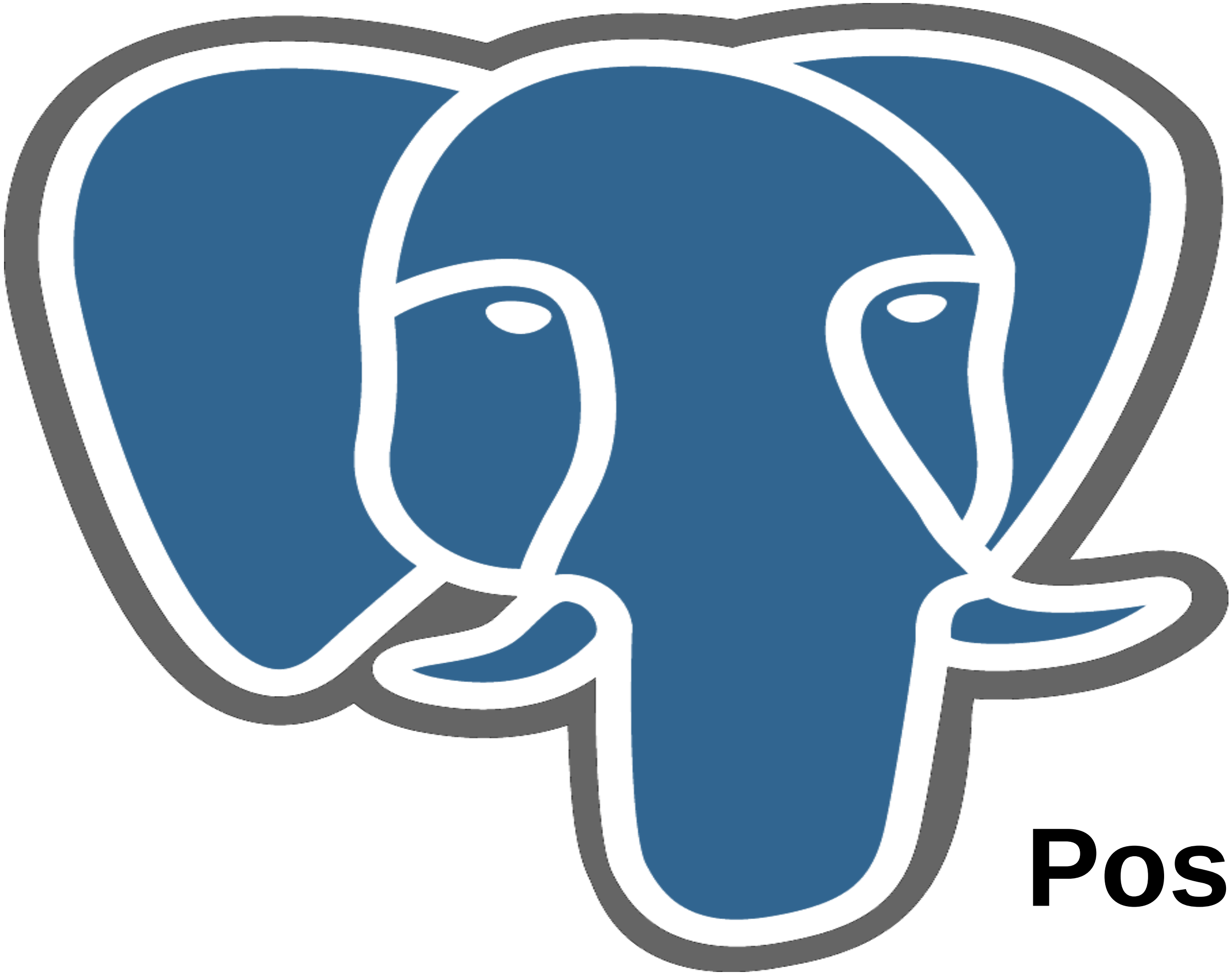
Ending

Road to v8.5

- *Needs your help*

2ndQuadrant
Professional PostgreSQL

NTT
Open Source
Software Center



Postgres