PostgreSQL Replication 2.0

NTT OSS Center
Masahiko Sawada

PGConf.ASIA 2017

Who am I

- **Masahiko Sawada**
  - @sawada_masahiko

- **NTT Open Source Software Center**

- **PostgreSQL contributor**
- **PostgreSQL technical support**
- **Maintenance of PostgreSQL related tools**

- **Will talk tomorrow again**
  - PostgreSQL Built-in Sharding
    — Enabling big data management with the blue elephant —

POSTGRESQL REPLICATION IS AWESOME!

Index

- What is Database Replication?
- The History of PostgreSQL Replication
- Logical Replication has came
- Summary

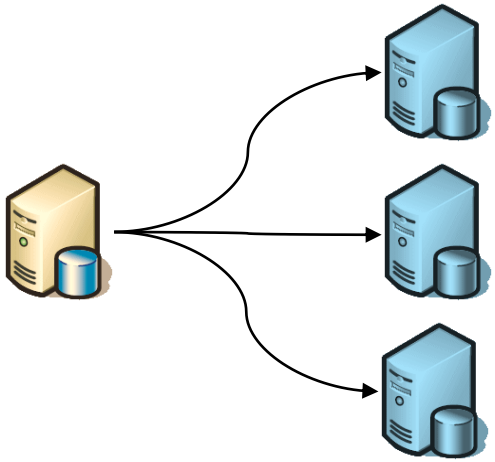
WHAT IS DATABASE REPLICATION?

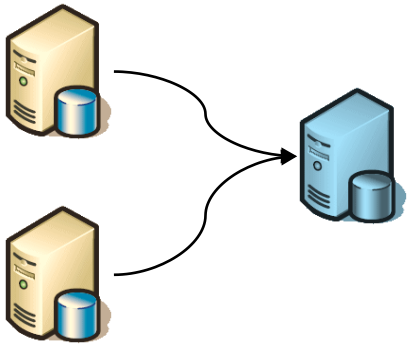
What is Database Replication?

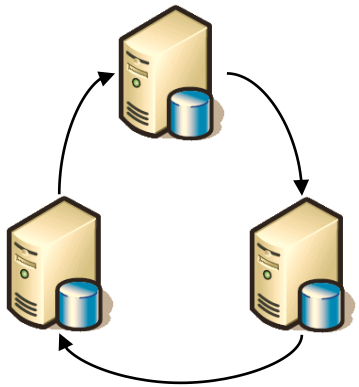
- **Keeping a copy of the data on multiple machines**
  - Continue working even if some of its parts have failed
  - Keep data geographically close to your users
  - Scale out the number of machines that can serve read queries

- **Master and Standby**
  - Primary and Slave, Leader and Follower

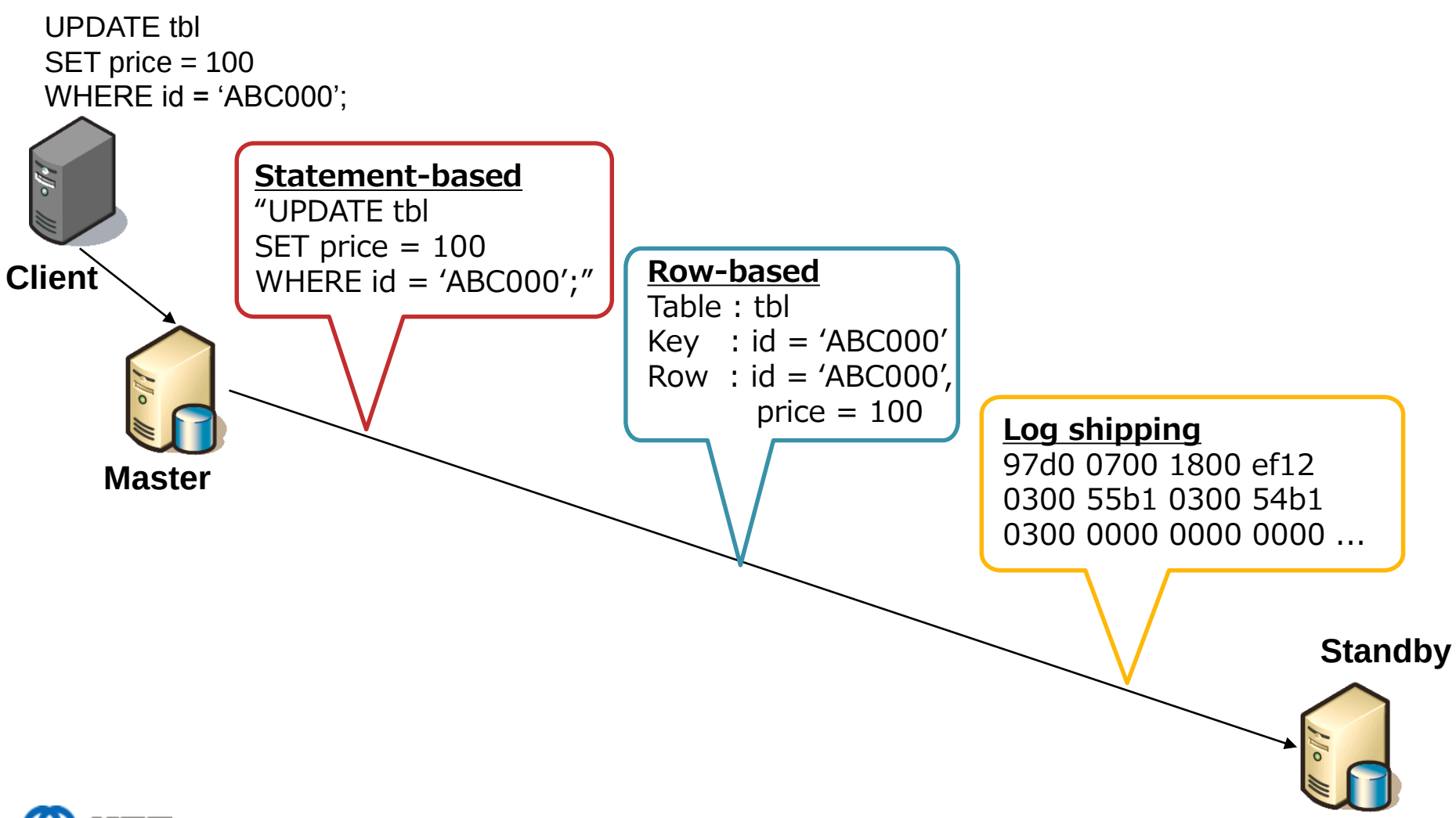
- **Replication Topology**





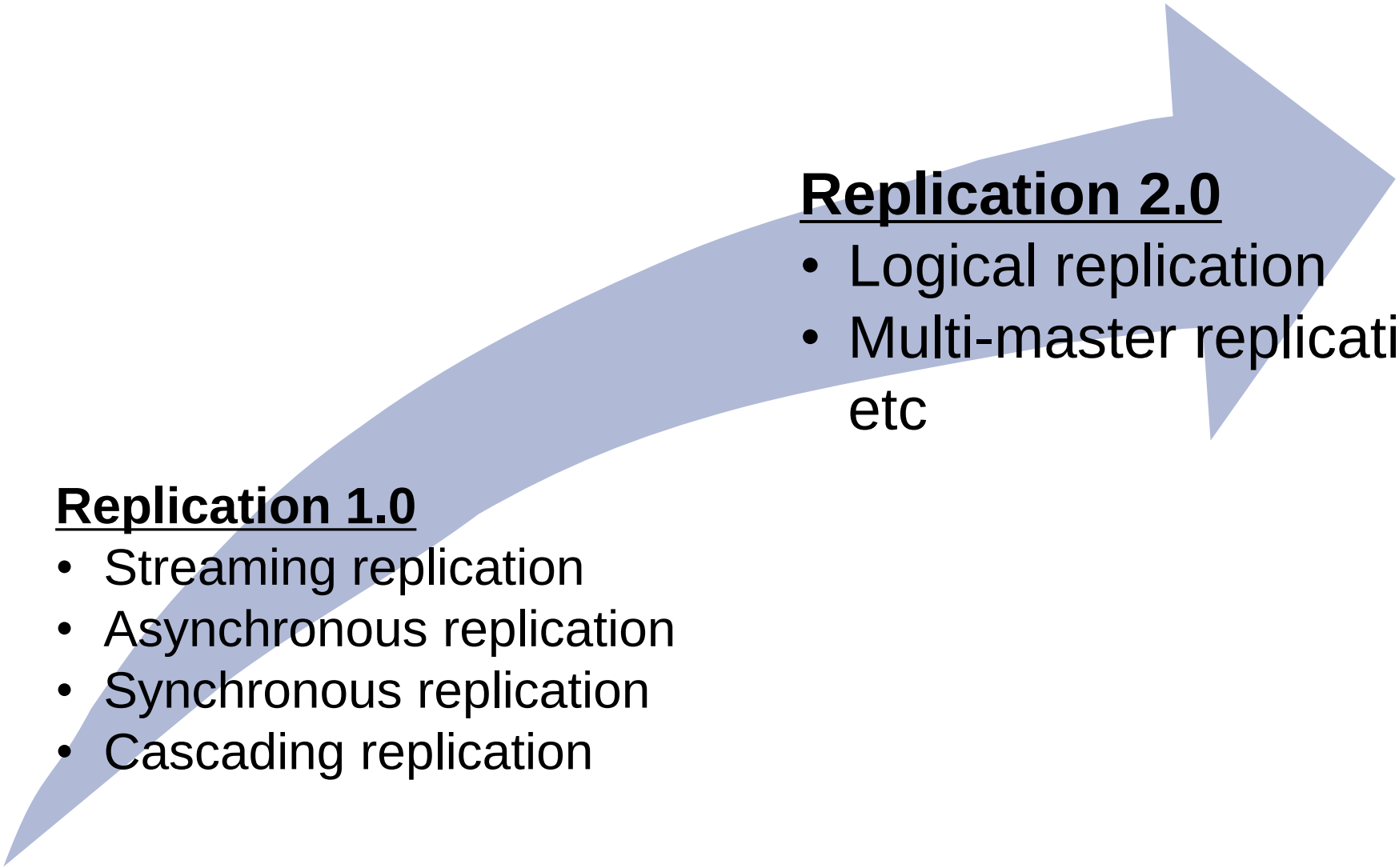


What to Replicate



THE HISTORY OF POSTGRESQL REPLICATION

PostgreSQL Replication is Awesome



Replication 2.0

- Logical replication
- Multi-master replication

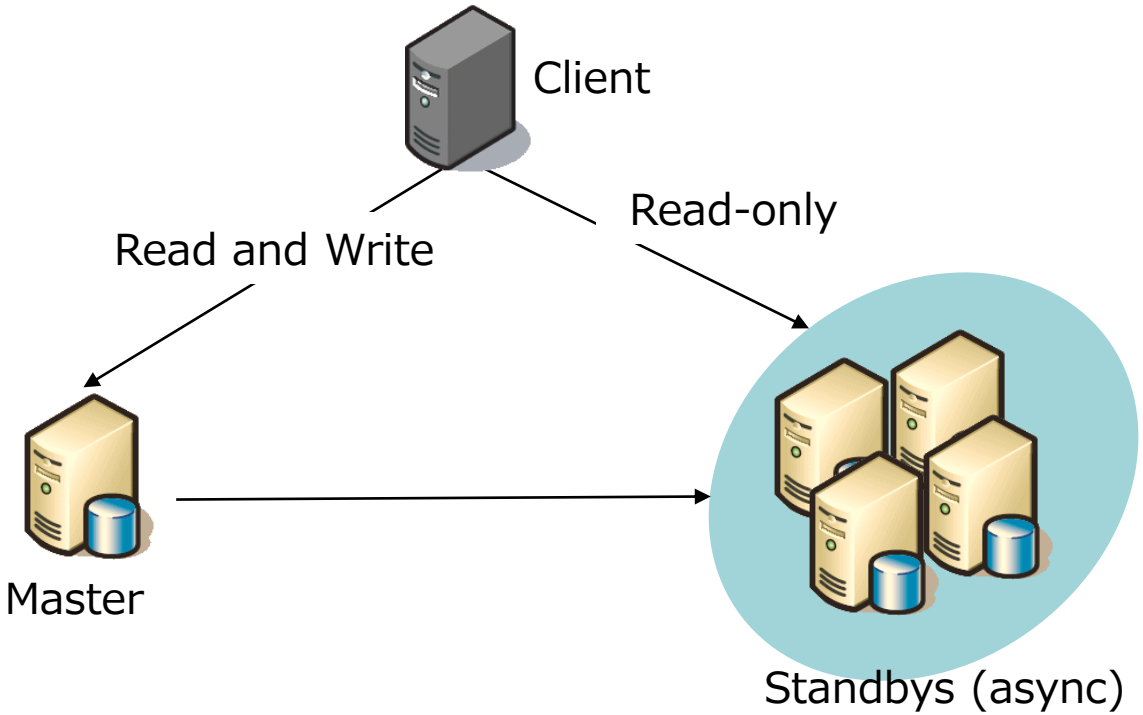
etc

Replication 1.0

- Streaming replication
- Asynchronous replication
- Synchronous replication
- Cascading replication

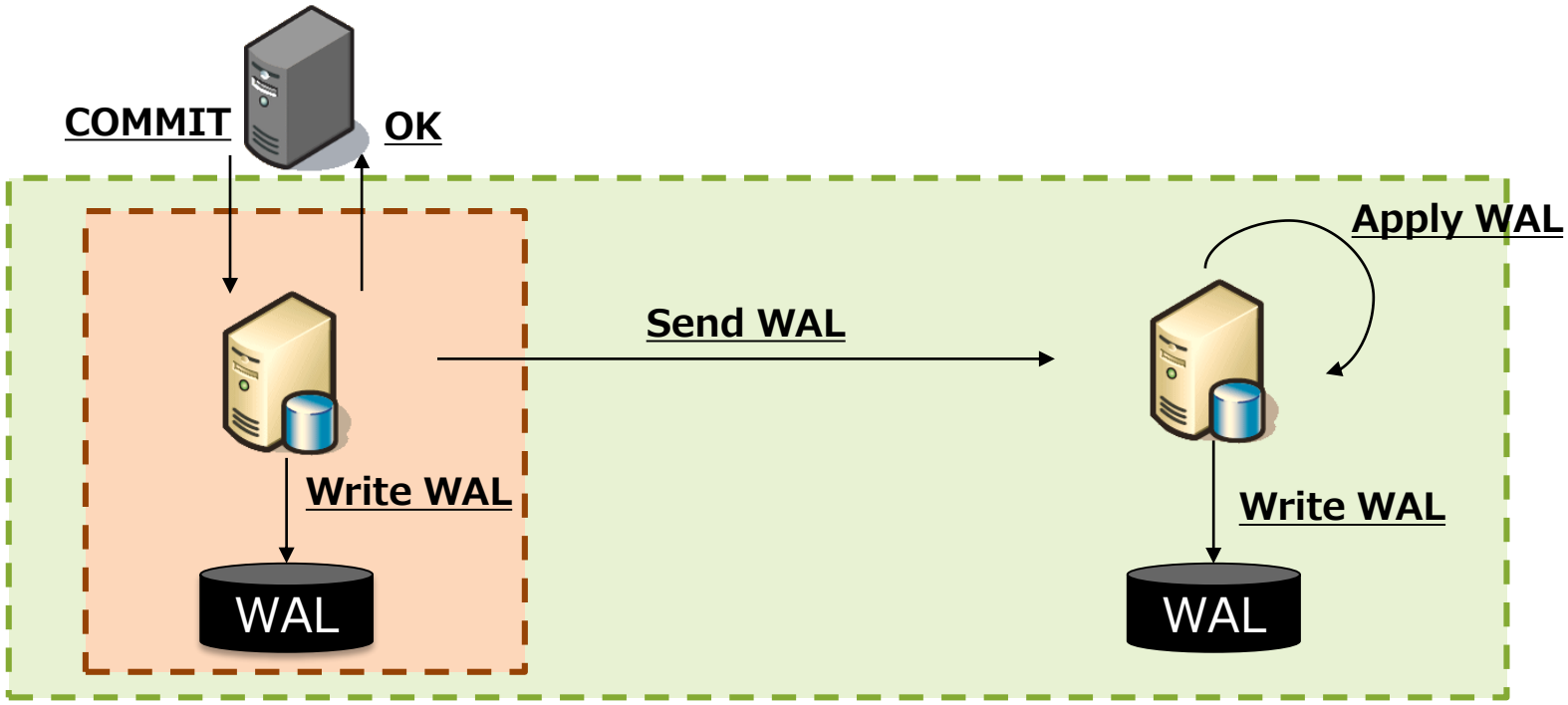
Backing to 2008..

- Streaming (physical) replication has been introduced
- Log (Write-Ahead Log) shipping
- Build an exactly same database cluster
- Single-master, multi-slaves



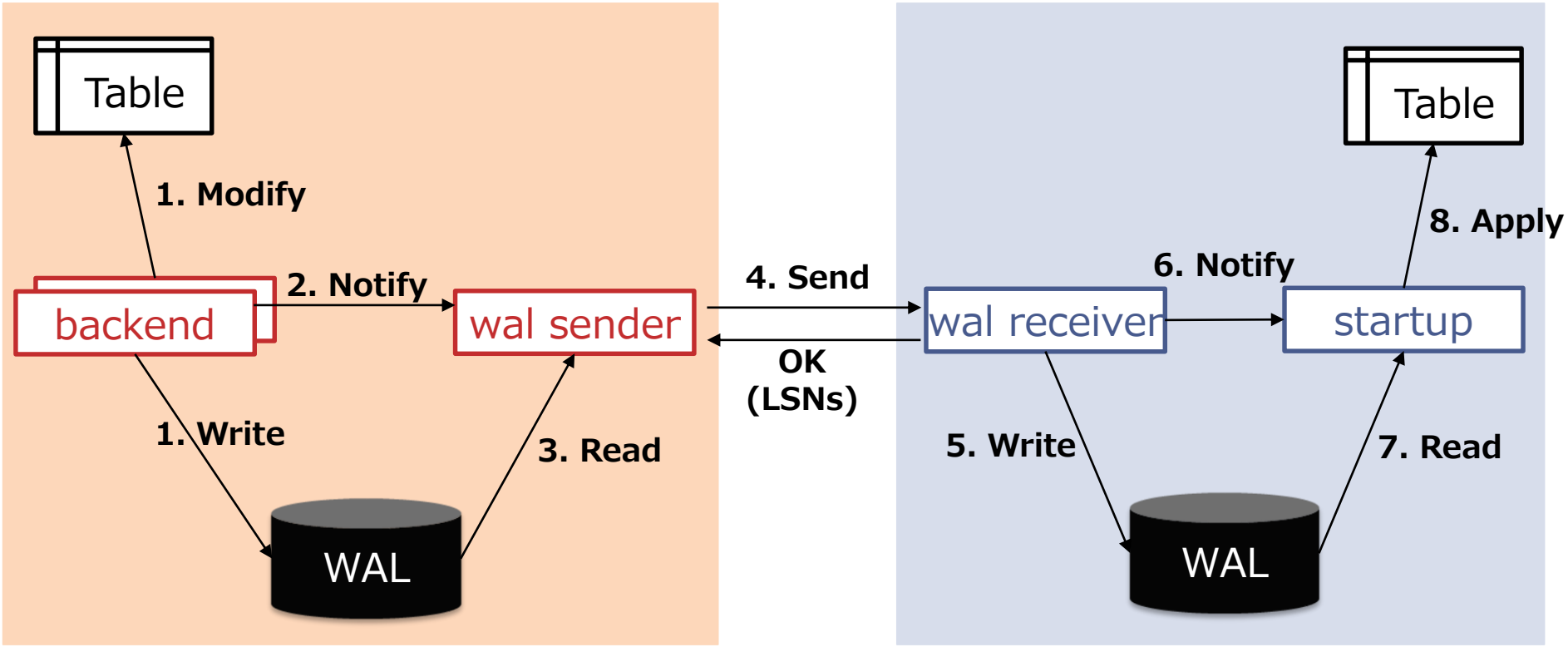
Basic Architecture of Streaming Replication

- Master server sends WAL, standby server receives it
- Standby server continues to receive and apply the received WAL
- Standby server can promote to a new master server



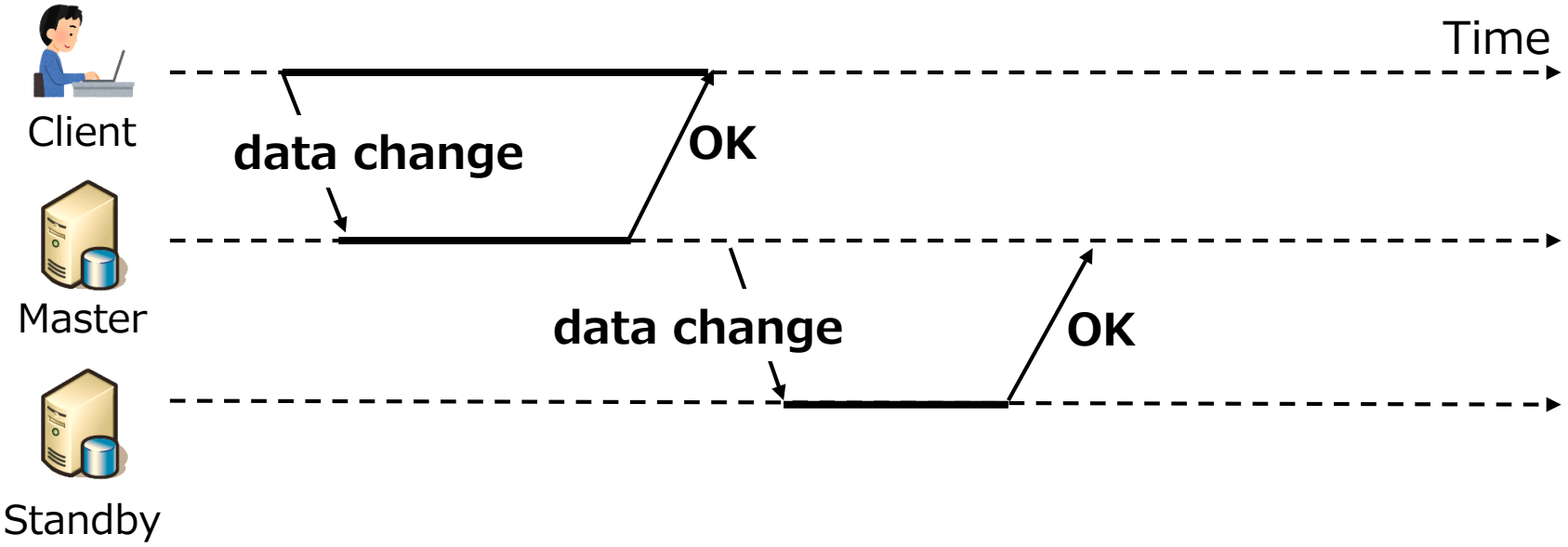
Basic Architecture in detail

- Wal sender process sends WAL to wal receiver process
- The standby server is doing the archive recovery
- In asynchronous replication, backend returns OK to client after step2



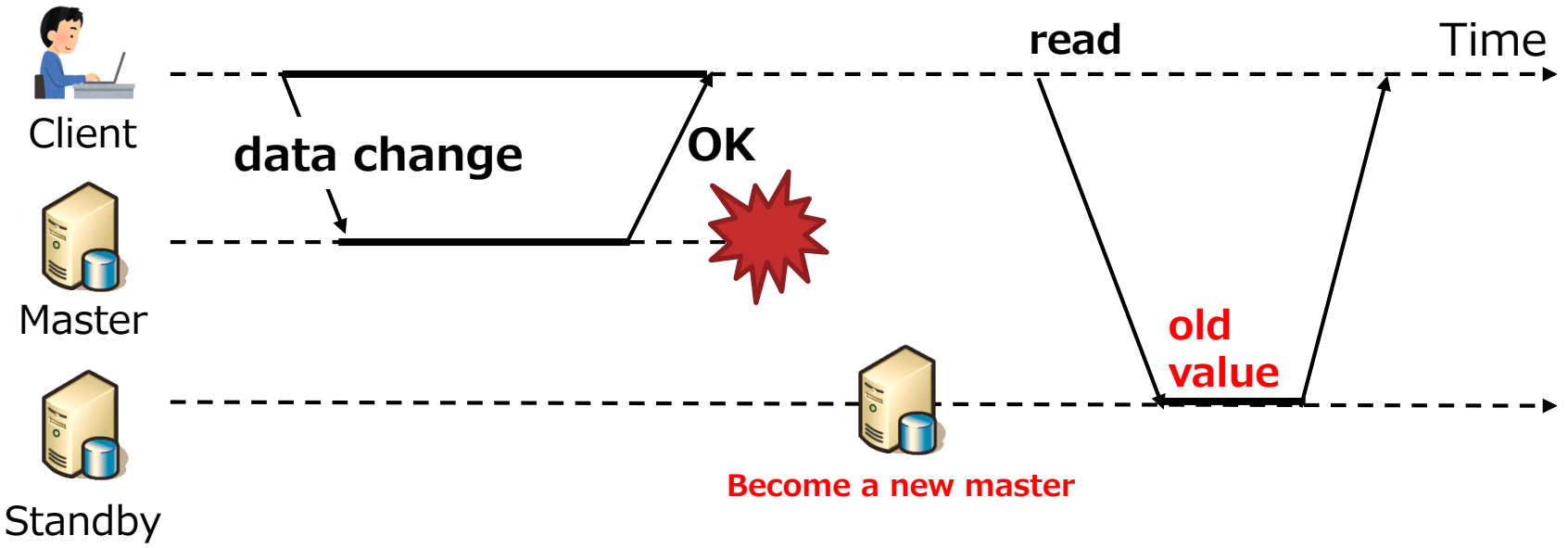
Asynchronous replication

- **Send WAL to standby server asynchronously**
- **Low overhead**
- **Commit does NOT wait to be replicated to the standby server**
  - Committed data could get loss on standby server



Asynchronous replication

- **Send WAL to standby server asynchronously**
- **Low overhead**
- **Commit does NOT wait to be replicated to the standby server**
  - Committed data could get loss on standby server



Read Replica (hot standby)

- “hot_standby = on” on standby server
- Enable to issuing READ SQL to a standby server
- For read balancing
- Note that the result on standby servers might be old

Handling Query Conflicts

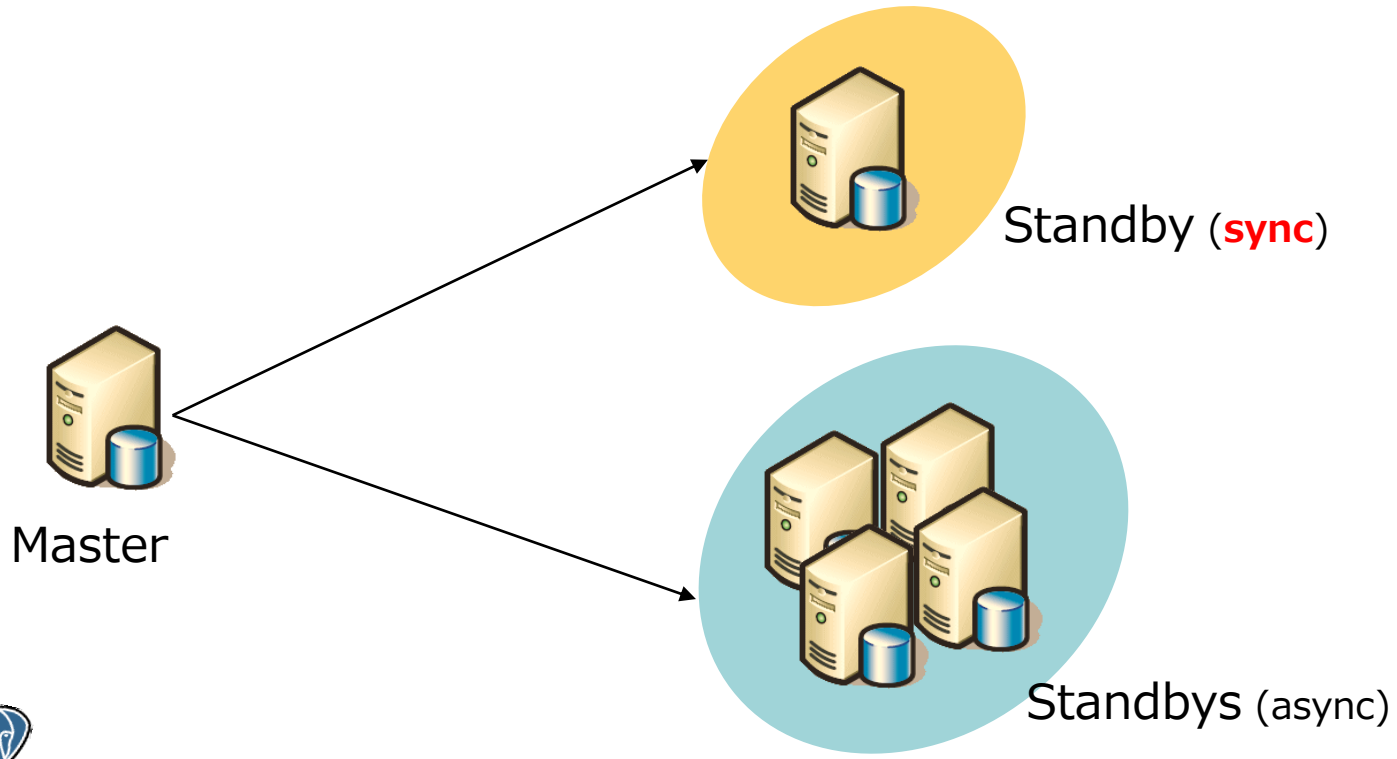
- **Conflict between queries on standby server and streaming replication**
  - DROP TABLE and SELECT
  - Vacuum cleanup and SELECT
  - Access Exclusive Lock and SELECT
  - etc

- **GUC Parameters on master server side**
  - vacuum_defer_cleanup_age

- **GUC parameters on standby server side**
  - hot_standby_feedback
  - max_standby_archive_delay
  - max_standby_streaming_delay

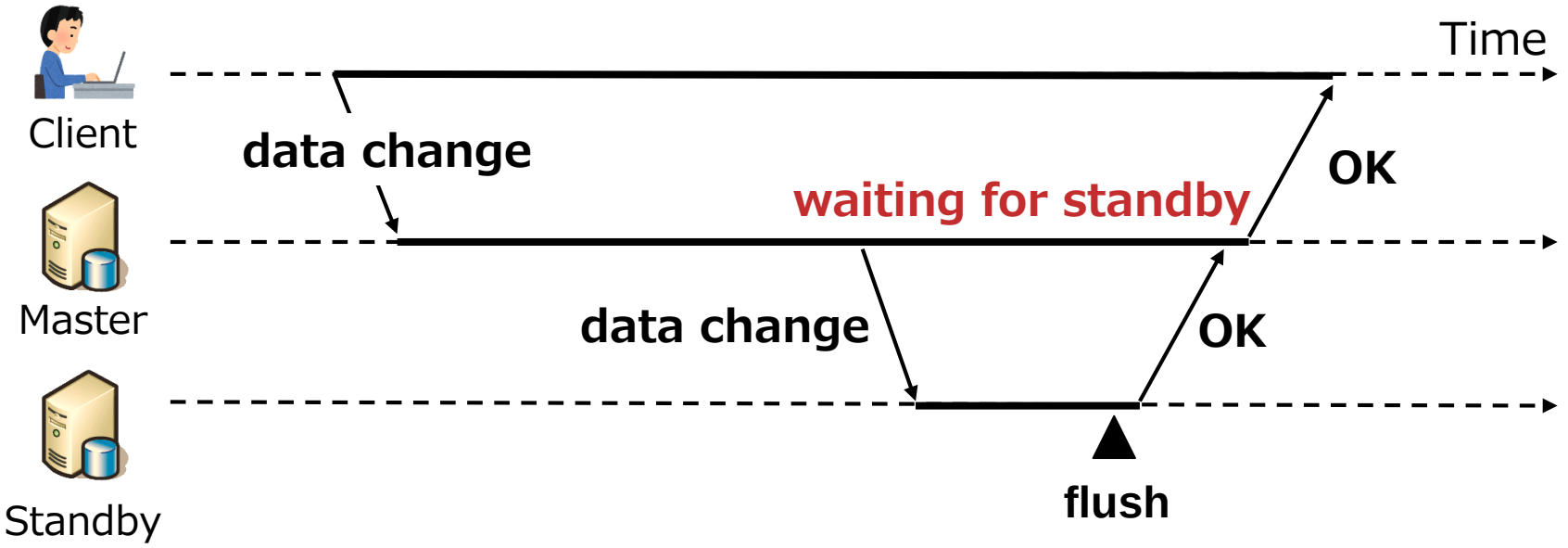
Synchronous Replication

- Only one standby can be “synchronous” standby
- Others are asynchronous standby
- synchronous_standby_names = ‘server1, server2’



Synchronous Replication

- **Commit waits for data to be replicated to the standby server**
- **When transaction commit, it guaranteed that the data is written on both the master server and the standby server**

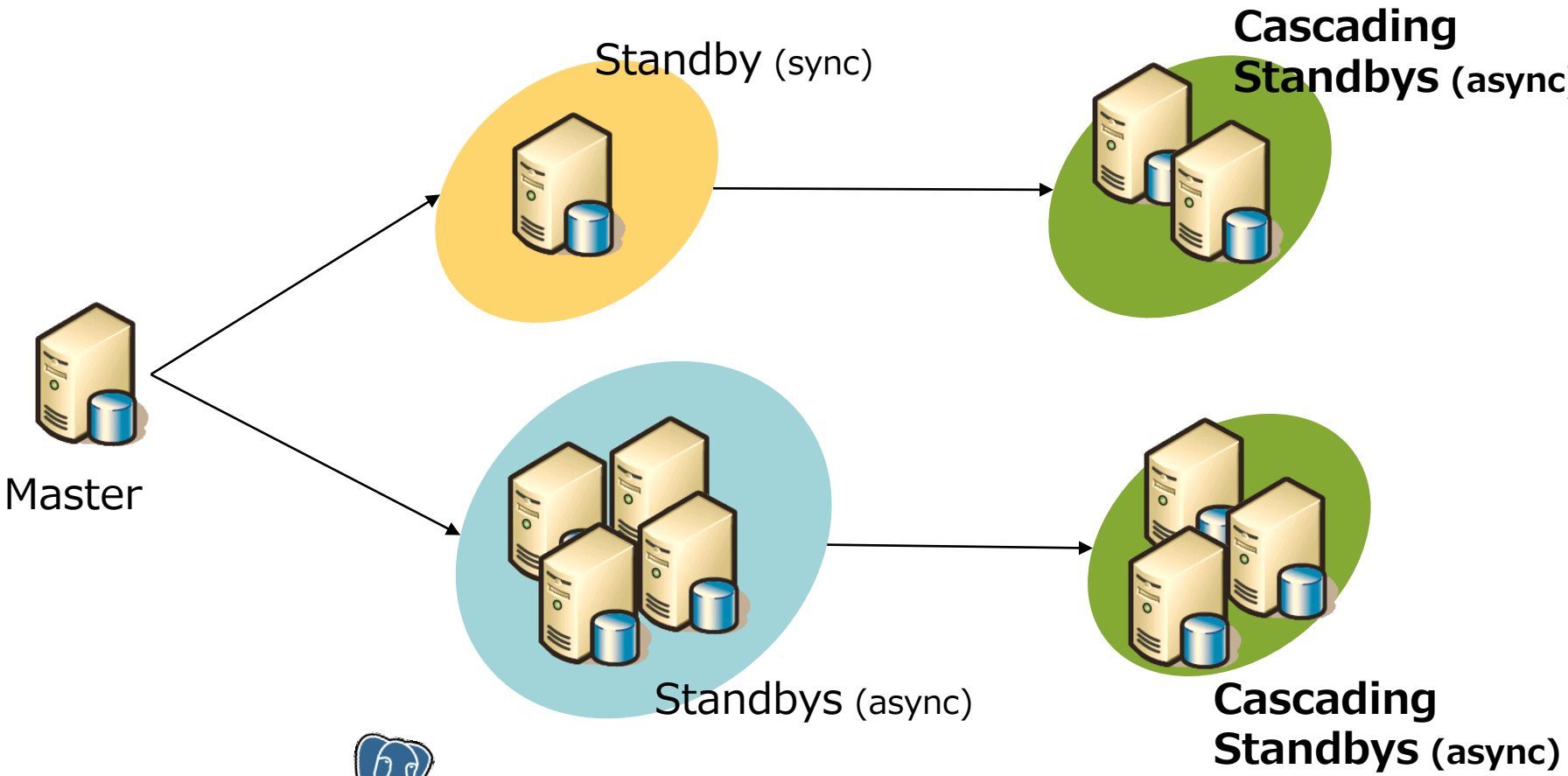


pg_basebackup

- **Taking a whole database cluster**
- **Easy to set up standby server**
- **pg_basebackup uses replication connection**
  - connect to wal sender
  - The master server needs to allow replication connection
- **Consume max_wal_senders**

Cascading Replication

- **A standby server has standby servers**



An old problem of replication

- **Required WAL might be archived on the master server while disconnection**
  - “FATAL: could not receive data from WAL stream ERROR: requested WAL segment 000000010000000000000007 has already been removed”

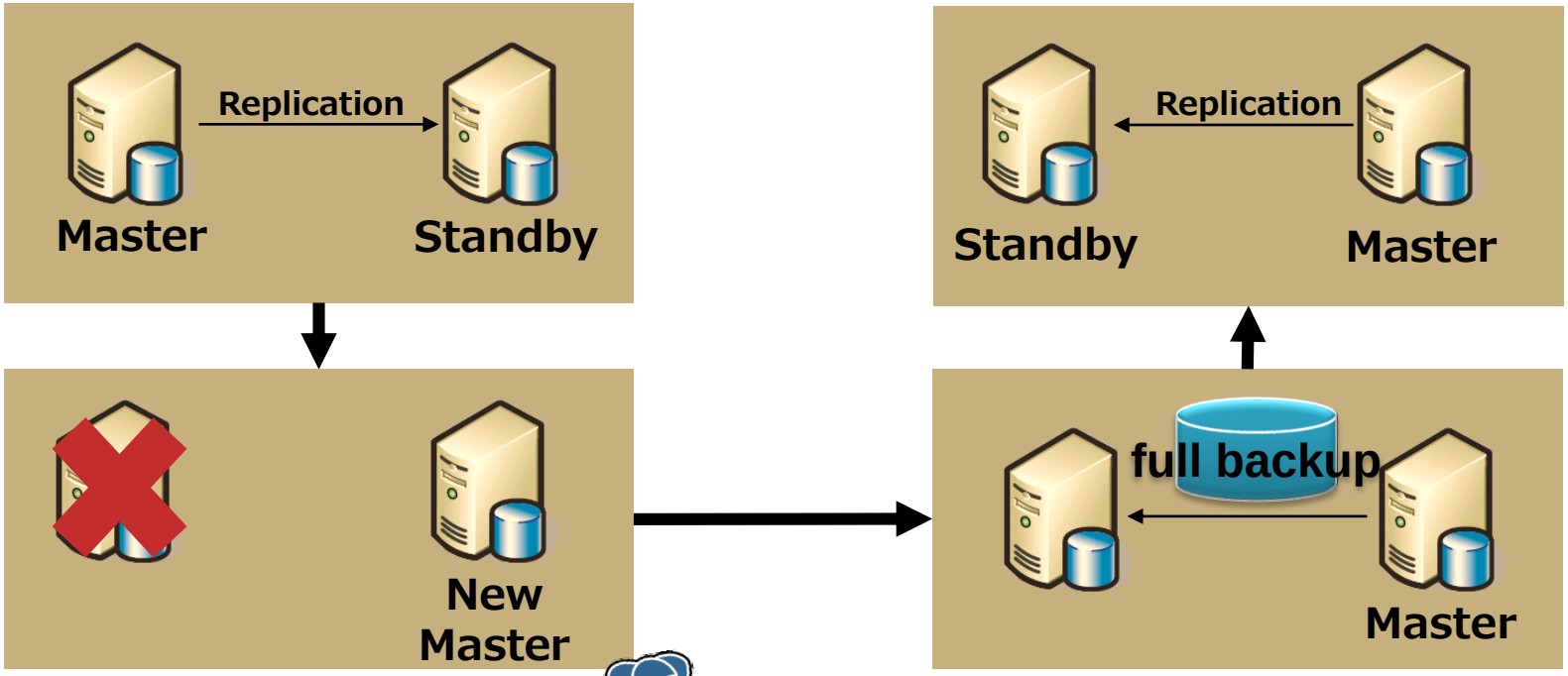
- **Solutions**
  - restore_command = ‘scp hostname:/path/to/%f %p’
    - What happen if archived WAL also has been removed?
  - wal_keep_segments
    - What happen if many WAL are generated much more than estimate?

Replication Slots

- **Ensure that the master doesn't remove WAL segments until they have been received**
- **Using streaming replication on a replication slot**
  - primary_slot_name in recovery.conf
- **Note that dangling replication slots could be cause of disk full**

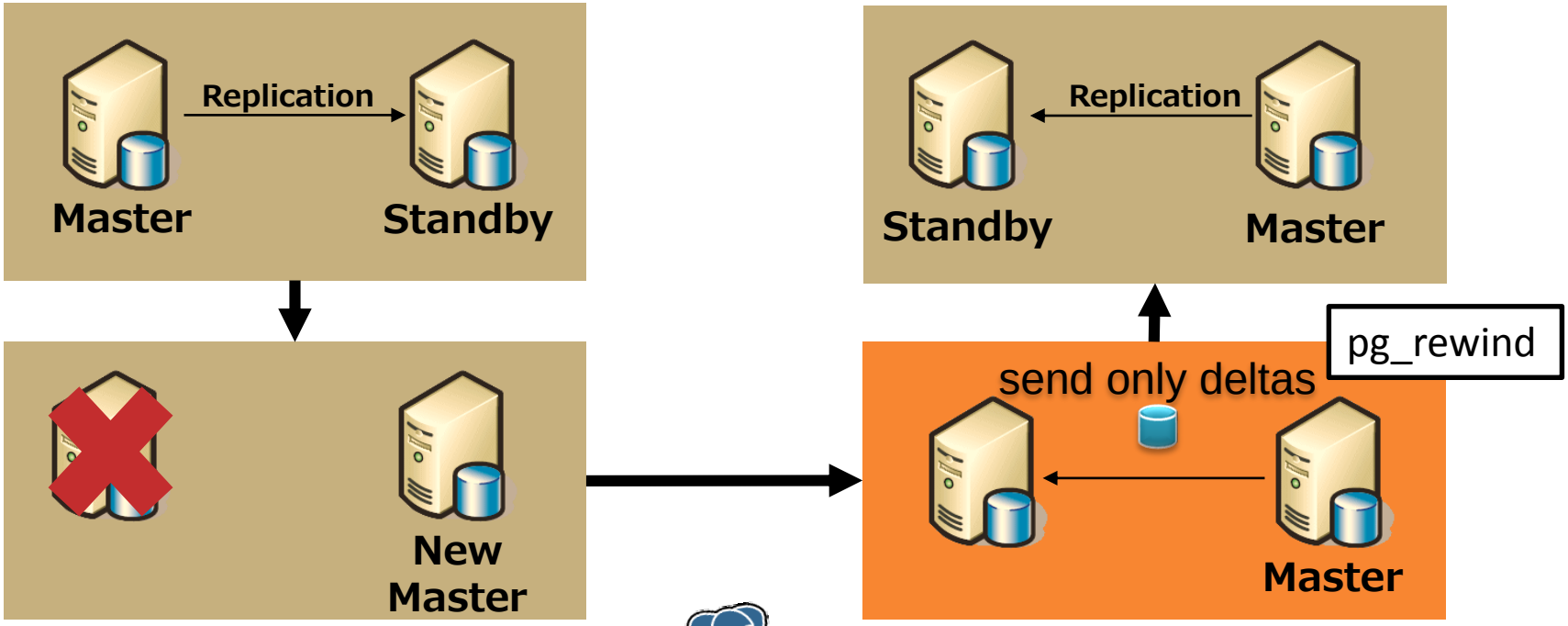
Bringing old master online after fail-over

- Needed full base backup after fail-over to bring old master server back online
- Take a long time if database is very large



pg_rewind

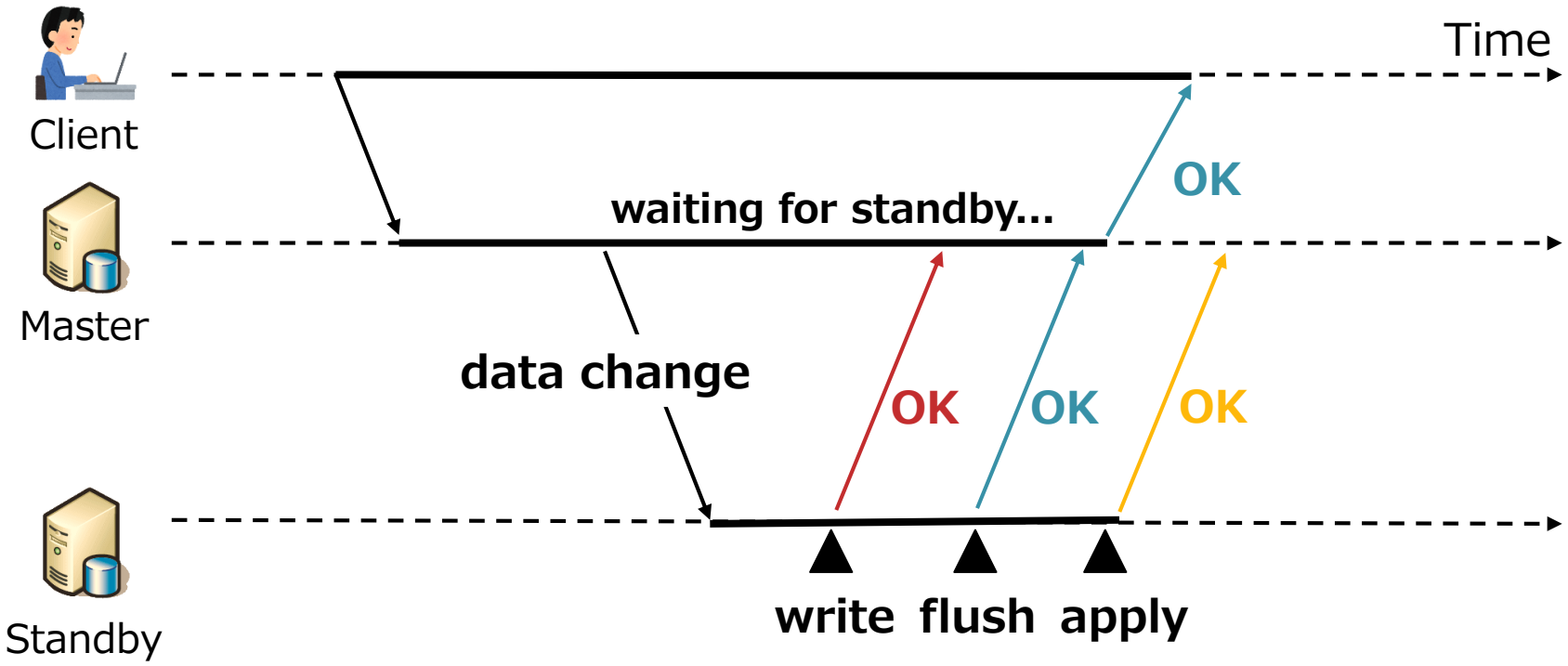
- Synchronize data with another data directory that was forked from it
- Don’t need a full base backup, send only changed blocks
- Also don’t need to read through unchanged blocks



Reliability Control

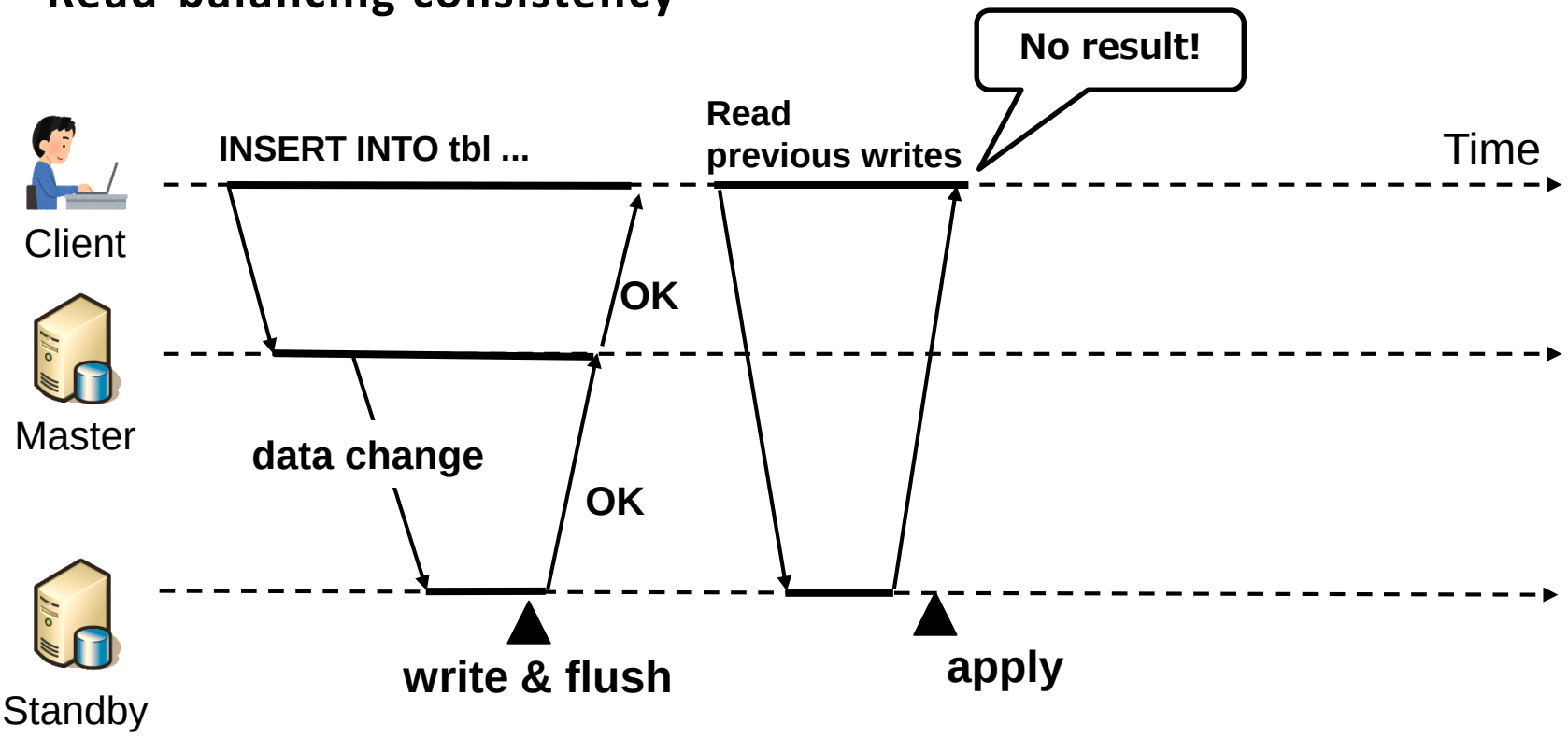
synchronous_commit =

[off | local | remote_write | on | remote_apply]



Reading Your Own Writes

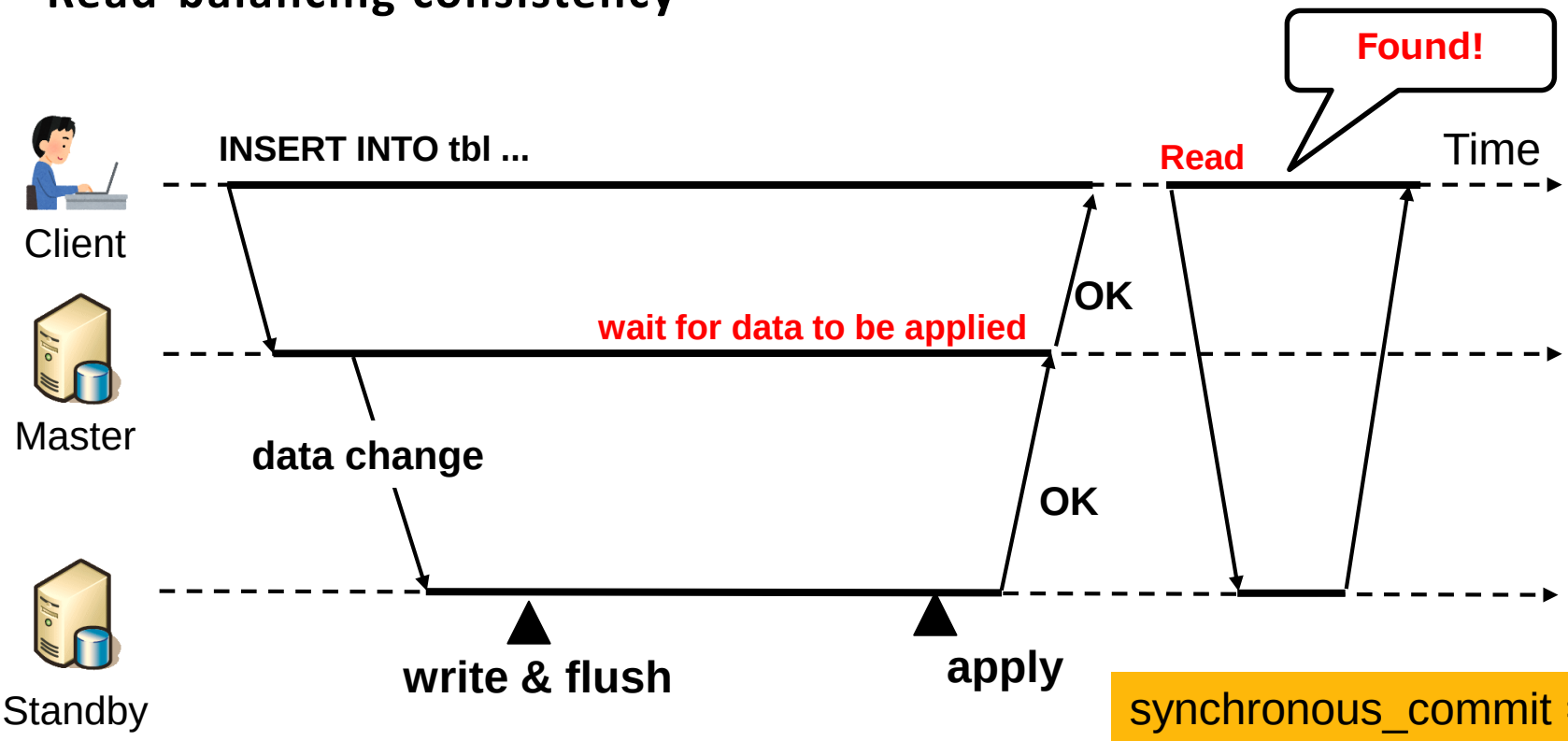
- **“remote_apply”**
  - it’s guaranteed that a session committing a transaction on a master node will be visible for session on the standby once it has been committed
- **Read-balancing consistency**



synchronous_commit = on

Reading Your Own Writes

- **“remote_apply”**
  - it’s guaranteed that a session committing a transaction on a master node will be visible for session on the standby once it has been committed
- **Read-balancing consistency**



synchronous_commit = remote_apply

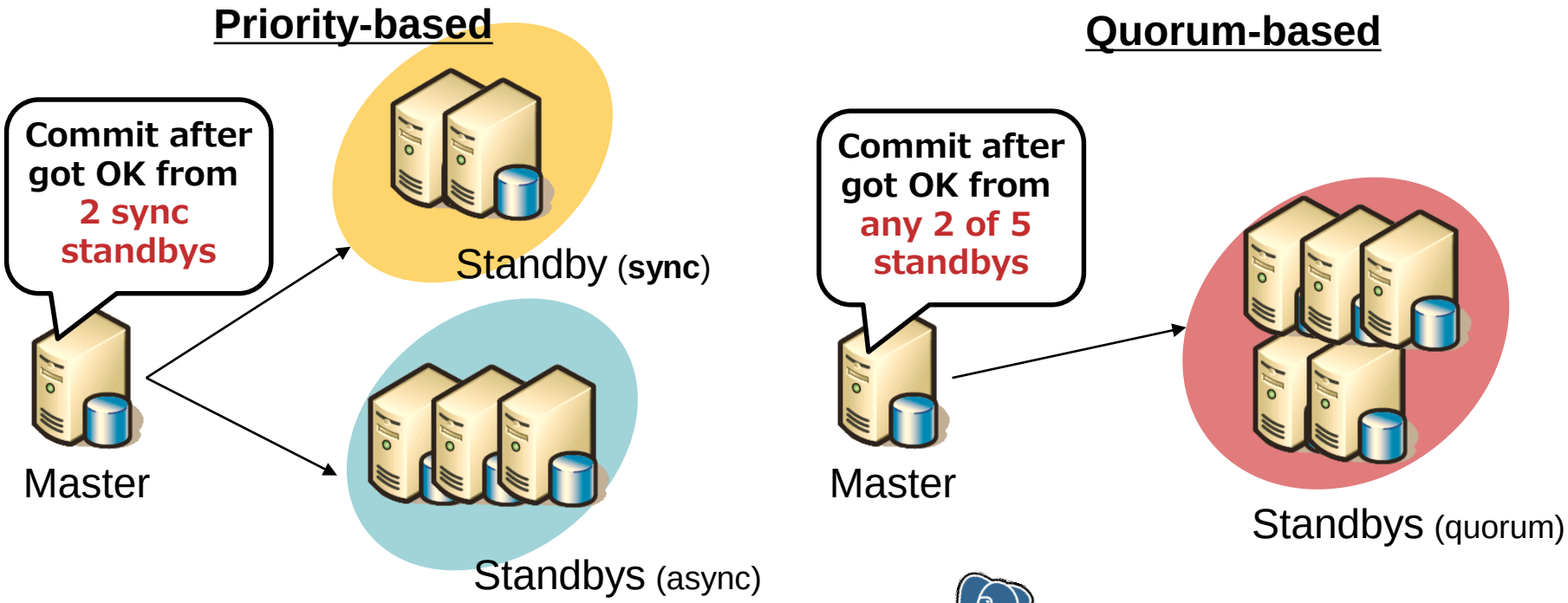
Monitoring

- Measuring replication lags
- write_lag, flush_lag, apply_lag

```
SELECT application_name, write_lag, flush_lag, replay_lag
FROM pg_stat_replication ;

 application_name |    write_lag    |    flush_lag    |   replay_lag
------------------+-----------------+-----------------+-----------------
 node1            | 00:00:00.022447 | 00:00:00.029091 | 00:00:00.68424
 node2            | 00:00:00.003227 | 00:00:00.004059 | 00:00:20.148379
 node3            | 00:00:00.020398 | 00:00:00.023971 | 00:00:00.614862
(3 rows)
```

Multiple Synchronous Replication

- **Enable to have more than one synchronous standbys**
- **Two methods**
  - priority-based
  - quorum-based



NTT 9.0 9.1 9.2 9.3 9.4 9.5 9.6 10

Summary for Streaming Replication

- **Write-Ahead Log shipping**
- **wal sender process and wal receiver process**
- **Single master, multiple standbys**
- **Asynchronous replication**
- **Synchronous replication**
  - priority-based and quorum-based
- **Cascading replication**
- **Replication lag**

LOGICAL REPLICATION HAS COME!

Logical Replication

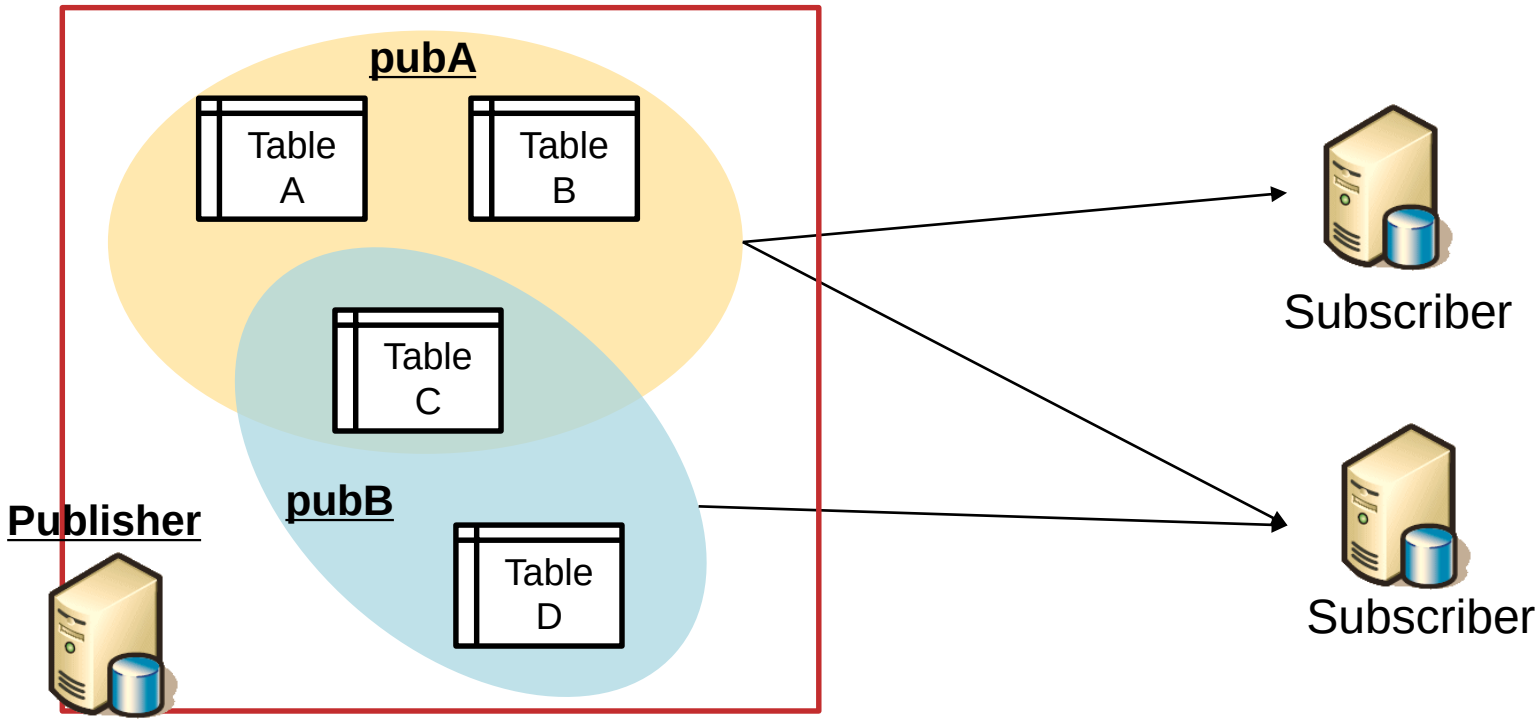
- Row based
- Replicate a subset of a database
- Receive changes from multiple servers
- Replicate to a different major versions of PostgreSQL
- Initial data copy
- Publication / Subscription model

Use cases

- Partial replication (sending a subset of a database)
- Consolidating multiple database into a single one
- On-line major version upgrading
- Multi-master replication

Publication / Subscription

- Publication is a set of changes generated from a table or group of tables
- Subscription defines set of publications to which it wants to subscribe



Initial Setup

```
-- On Publisher
CREATE TABLE tbl (k int primary key, v int);
CREATE PUBLICATION tbl_pub FOR TABLE tbl;
INSERT INTO tbl VALUES (1), (2), (3);
```

```
-- On Subscriber
CREATE TABLE tbl (k int primary key, v int);
CREATE SUBSCRIPTION tbl_sub CONNECTION '...' PUBLICATION tbl_pub;
SELECT * FROM tbl;
 c
---
 1
 2
 3
(3 rows)
```

Logical Decoding

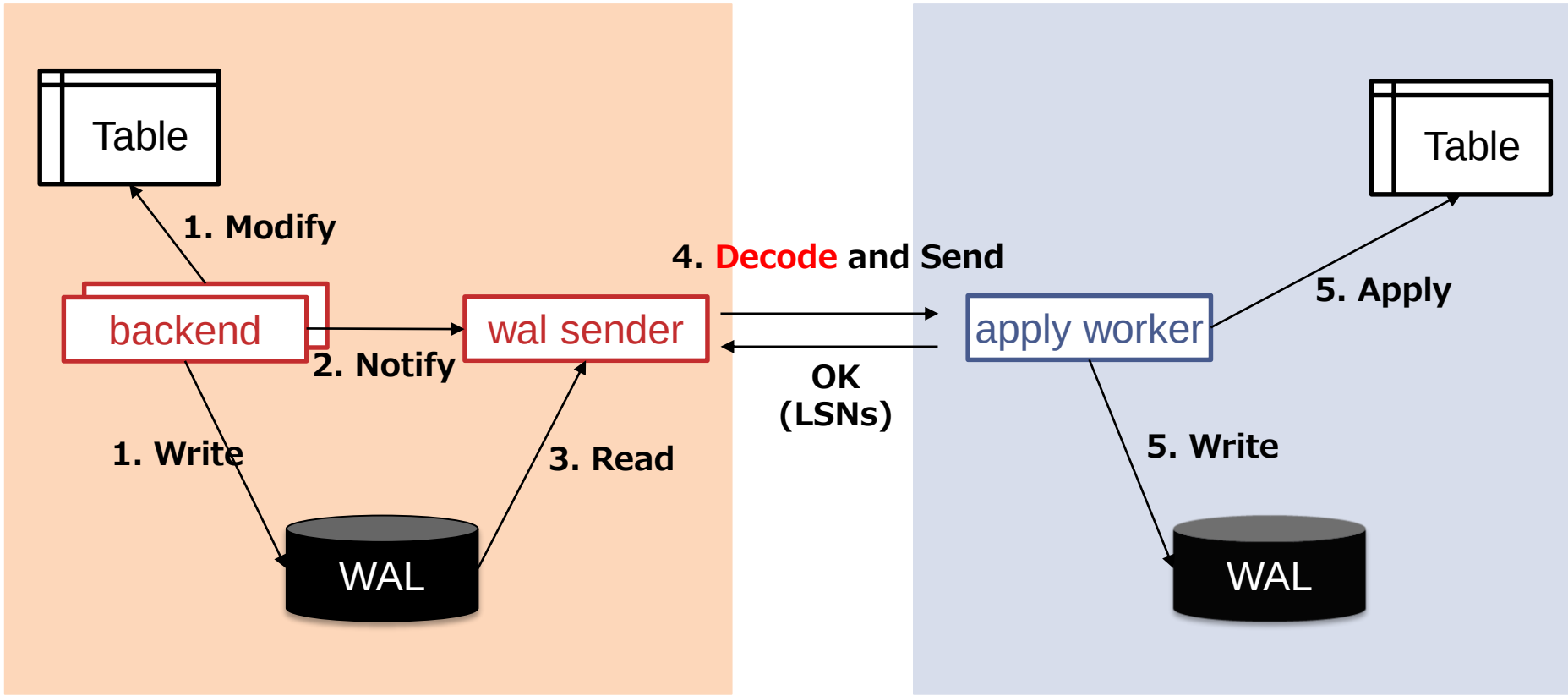
- An infrastructure feature of Logical Replication
- Logical Replication sends decoded WAL data (ROW-based)

```
BEGIN;
CREATE TABLE tbl (c int primary key);
INSERT INTO tbl VALUES (1), (2), (3);
COMMIT;
```

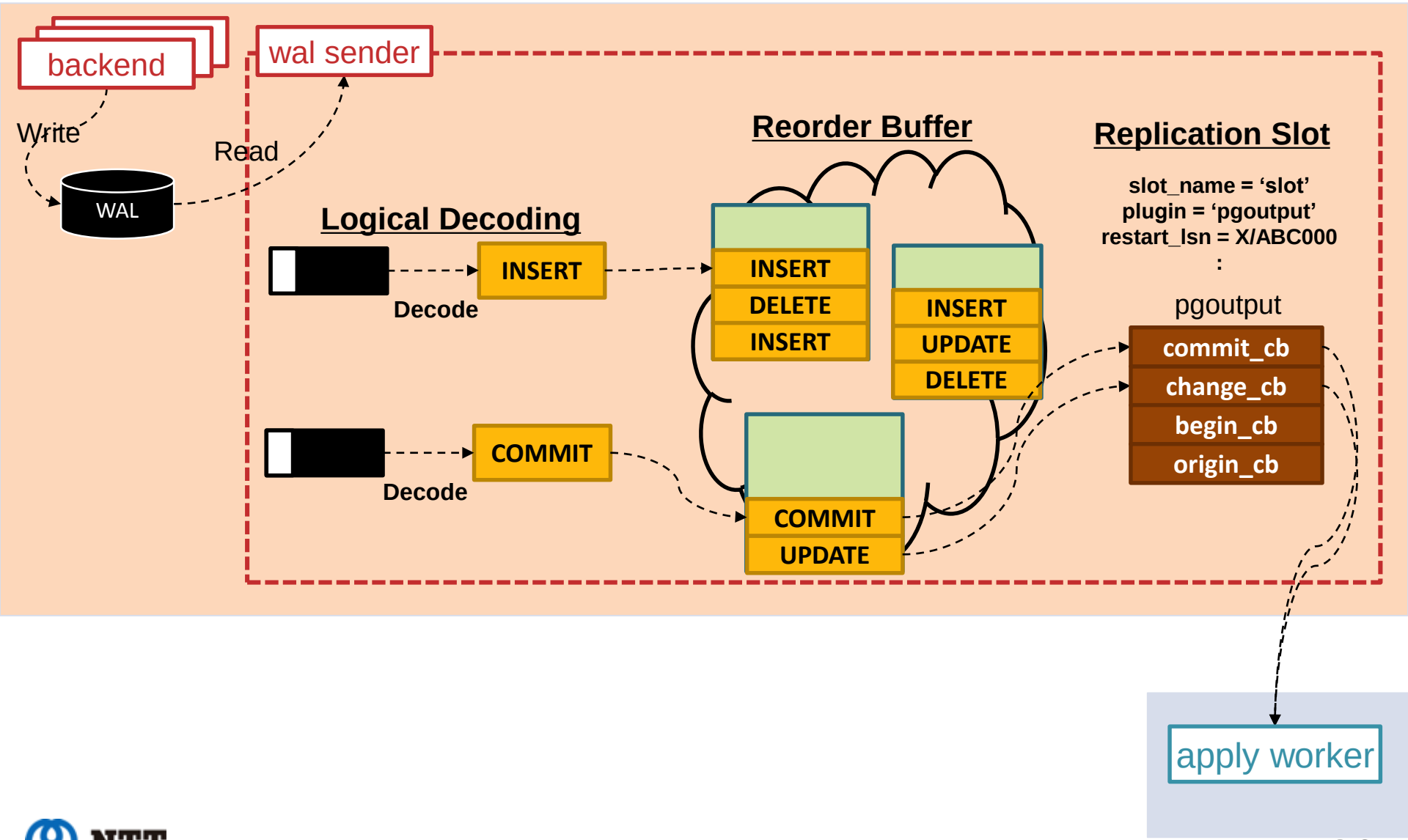
```
SELECT  lsn, data FROM pg_logical_slot_get_changes('slot',
pg_current_wal_lsn(), 10);

    lsn     |                  data
------------+----------------------------------------
 1/331723E8 | BEGIN 242422
 1/3317A778 | table public.tbl: INSERT: c[integer]:1
 1/3317A848 | table public.tbl: INSERT: c[integer]:2
 1/3317A8C8 | table public.tbl: INSERT: c[integer]:3
 1/3317ABC0 | COMMIT 242422
(5 rows)
```

Basic Architecture in detail

- After read WAL, wal sender process decodes it using a plugin (pgoutput)
- What to send by wal sender is, row-level changes

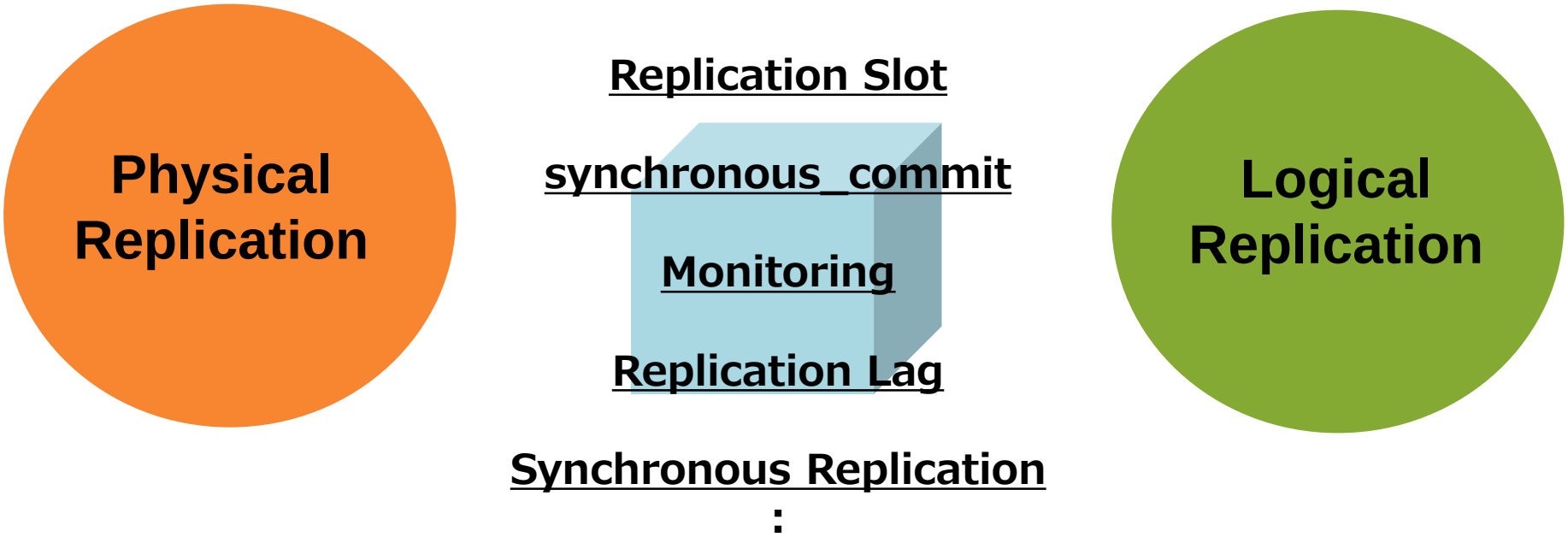


Basic Architecture in MORE detail



Streaming Replication and Logical Replication

- Logical Replication has been developed since 9.4
- Many common components



Common components

- **wal sender process**
- **Replication slot**
  - keeping WAL segments, logical decoding plugin
- **pg_stat_replication**
  - Same monitoring interface

Many New Components Are Required

- Decoded WAL receiver (apply worker)
- Management of apply workers (logical replication launcher)
- Logical replication protocol
- Snapshot builder
- Replication origin
- Reorder buffer
- Initial table synchronization
- Relation mapping

etc

Known Restrictions

- **Utility commands are not supported**
  - DDLs, two-phase commit entries etc
- **Not “streaming” logical replication**
  - Decoded WAL is replicated when commit
- **When using synchronous replication**
  - can only use server-level (not subscription-level)
  - wait for unsubscribed server
- **“UPDATE OF” trigger**
- **Concurrency restriction**
  - Concurrent CREATE SUBSCRIPTION
    - Work around
  - Concurrent ALTER SUBSCRIPTION REFRESH PUBLICATION

For Further Enhancement

- **Ease restrictions**
  - DDLs and utility command replication
  - Streaming logical replication
- **Online major version upgrading tool**
- **Confliction monitoring/management**
- **Built-in automatic fail-over**

SUMMARY

PostgreSQL Replication is AWESOME

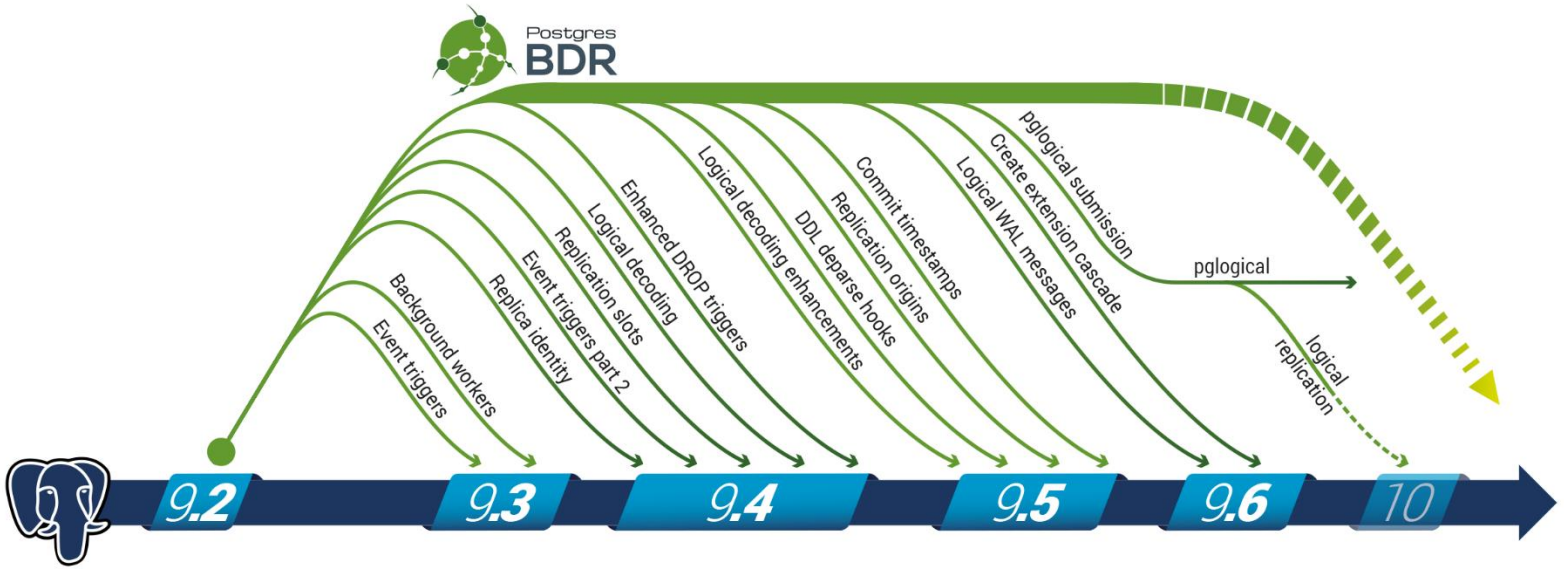
- **Physical Replication is 10 years old**
  - “Sea change” feature
  - High functionality
  - Matured
  - Being continued to be evolved

- **Logical Replication is 0 year old**
  - Also “Sea change” feature
  - Has enormous potentialities
  - Some restrictions
  - Common components with streaming replication

THANK YOU !!

Masahiko Sawada
mail: sawada_masahiko_f7@lab.ntt.co.jp

- **Has been implementing since PostgreSQL 9.2**
- **Incremental development**



https://blog.2ndquadrant.com/bdr-history-and-future/

Taking backups on standby server

- archive_mode = [on | always | off]
- Enable to use on-line backup using pg_start_backup() and pg_stop_backup()