21 September 2021

High Availability & Replication:

Replication Performance,
and <u>why</u> it matters

EDB

Evolution of Replication

• Trigger-based Replication	Slony, Londiste	2004
• Physical File-based Replication	PG8.2	2006
• **Physical Streaming Replication**	PG9.0	2010
• **Logical Streaming Replication**	PG10	2017

Why?

- Why does it matter how fast replication is?
- Why does it matter how fast persistence is?
- Robustness and High Availability features are slow

synchronous_commit

EDB

synchronous_commit = on

- Means all WAL will be fsynced to disk before COMMIT
- Ensures that any COMMIT is **Durable**

synchronous_commit = off

- Means report COMMIT to user **before** changes have gone to disk
- For short transactions, can be ~10x faster, with slower disk technology
- COMMITs may be lost if the server crashes

synchronous_commit = remote_apply

- Means wait for COMMIT to be applied to remote server(s) (Assuming synchronous_standby_names is set) **Synchronous Replication**
- Can be much slower as we wait for round trip and apply
- Most sessions spend a long time waiting for reply, so causes a drop in throughput as well as loss of latency
- Use more sessions when your applications require Sync Replication
- Restricts from transactions on the origin node from going too fast for the standby nodes

synchronous_commit = remote_write

- Means wait for WAL for COMMIT to be written to remote server(s)
  Synchronous Replication, with a performance boost

- Slightly faster than remote_apply, since need not wait for apply

- Also gives more consistent replication lag since effects that slow down apply do not affect the response to origin client
- However, still affected by origin tasks generating too much WAL, such as CREATE INDEX, ALTER TABLE, VACUUM, VACUUM FULL, CLUSTER etc..

synchronous_commit = local

- Means don't wait for WAL for COMMIT to be sent to remote server(s)
  Asynchronous Replication

- Faster on local node, but can allow the origin node to process transactions faster than they can be applied, causing a backlog to develop
- **Typical cause of replication lag**

Example

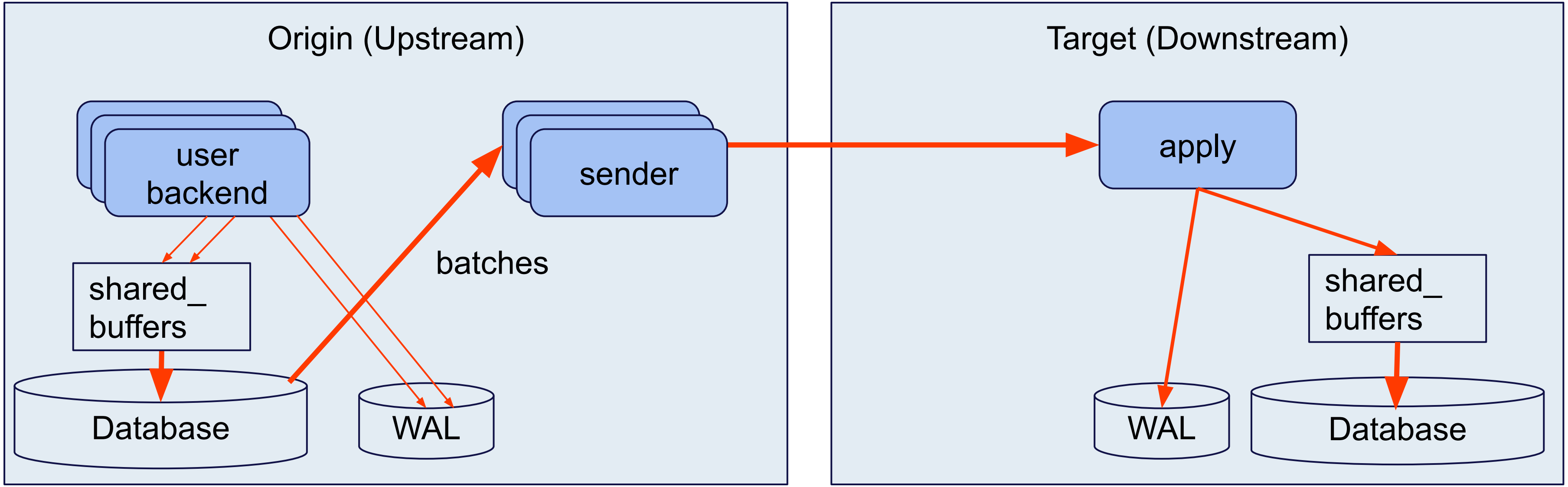
- Server capable of 100k TPS
- Replication capable of 25k TPS
- Application exceeds 25k TPS then backlog begins to develop
- If App runs at 75k TPS for 1 hours, then a backlog of 50k xacts will develop, which would take 2 hours to clear on a quiet server, longer in other cases

- At failover, much data could be lost if this is not managed
- **Replication performance is very important for your data!**

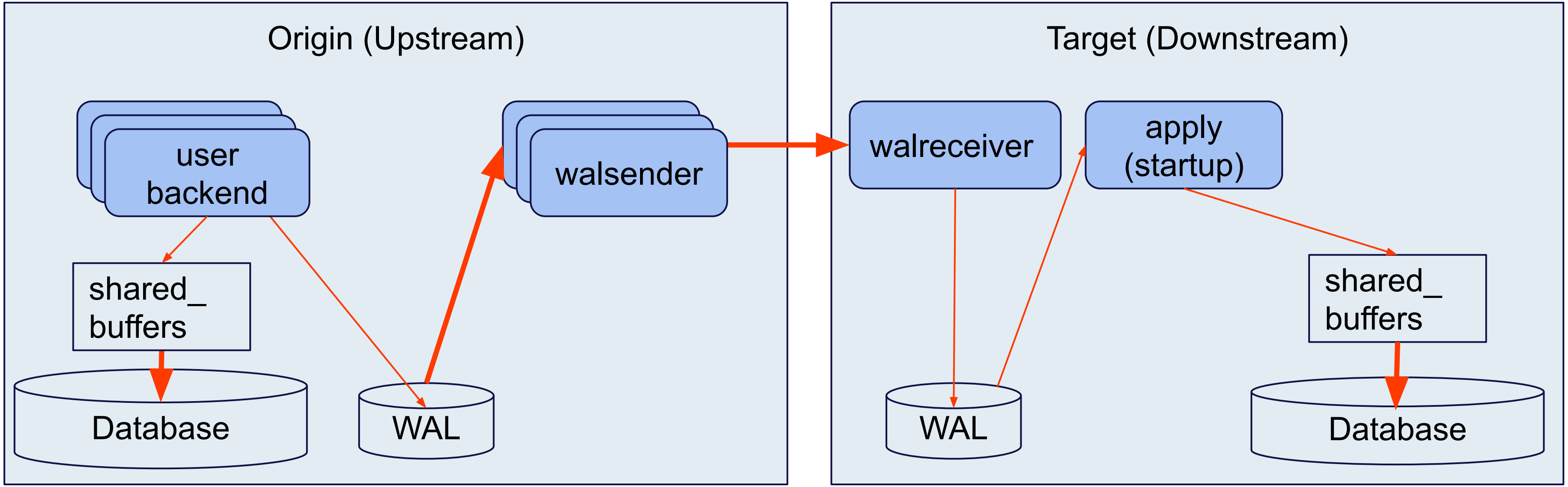
Replication Architectures & Performance

EDB

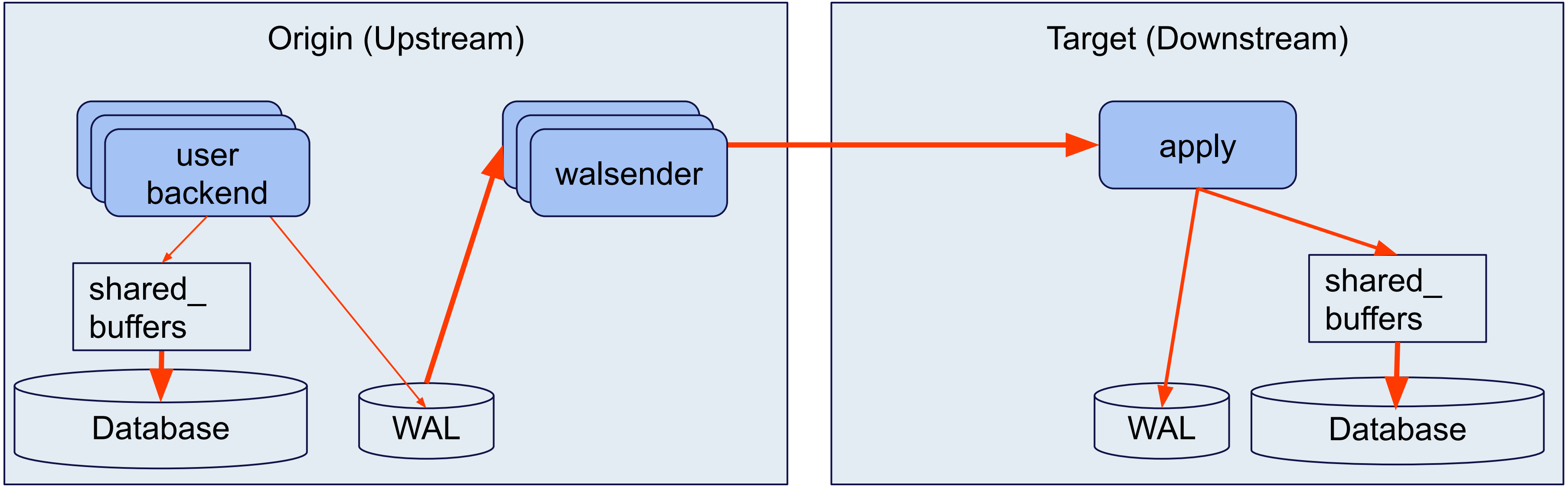
PostgreSQL Trigger-based Replication



PostgreSQL Physical Streaming Replication [PG9]



PostgreSQL Logical Streaming Replication [PG10]



Replication Architecture Comparisons

- WAL Overhead on Origin
- Network bandwidth used
- Number/architecture of Processes performing work
- Type of Apply (direct block/row search)

VLDB Issues

Physical: Row Hints and Index Queries

- Row hints are not passed across as WAL records **in all cases**
- Indexes cannot trust that the item killed hint has been replicated, so index searches are less than optimal in a table with UPDATEs
- Can be a noticeable performance issue

Cache and I/O Effects

- To maintain good performance for replication, working set of database must remain in cache
- Any cache shortfall will become I/O
- Architecture should allow I/O avoidance
  - Physical via Full Page Writes
  - Logical via Parallel Apply

- => Cache on replicas should match cache on origin

Type of Apply

- Physical
  - Direct block access
  - Prefetch coming in PG15?
- Logical
  - Rows searched using Primary Keys
  - Index entries also need to be re-applied

  - Btree access is O(logN), while Hash indexes can be O(k)
  - Hash indexes become more important for Logical Replication

Basebackup & Catchup

- Basebackup needed as first stage of replication setup
- Catchup starts when base backup ends

- P = processing rate of Origin node
- R = processing rate of Replication
- T = time to take Basebackup

- Catchup time = $\dfrac{T * P}{R - P}$

Major Release Upgrades without downtime

- Only possible with Logical Replication since significantly fewer dependencies to specific release formats and behavior

- Can VLDBs that rely on Physical Replication be upgraded sensibly?

Conclusions

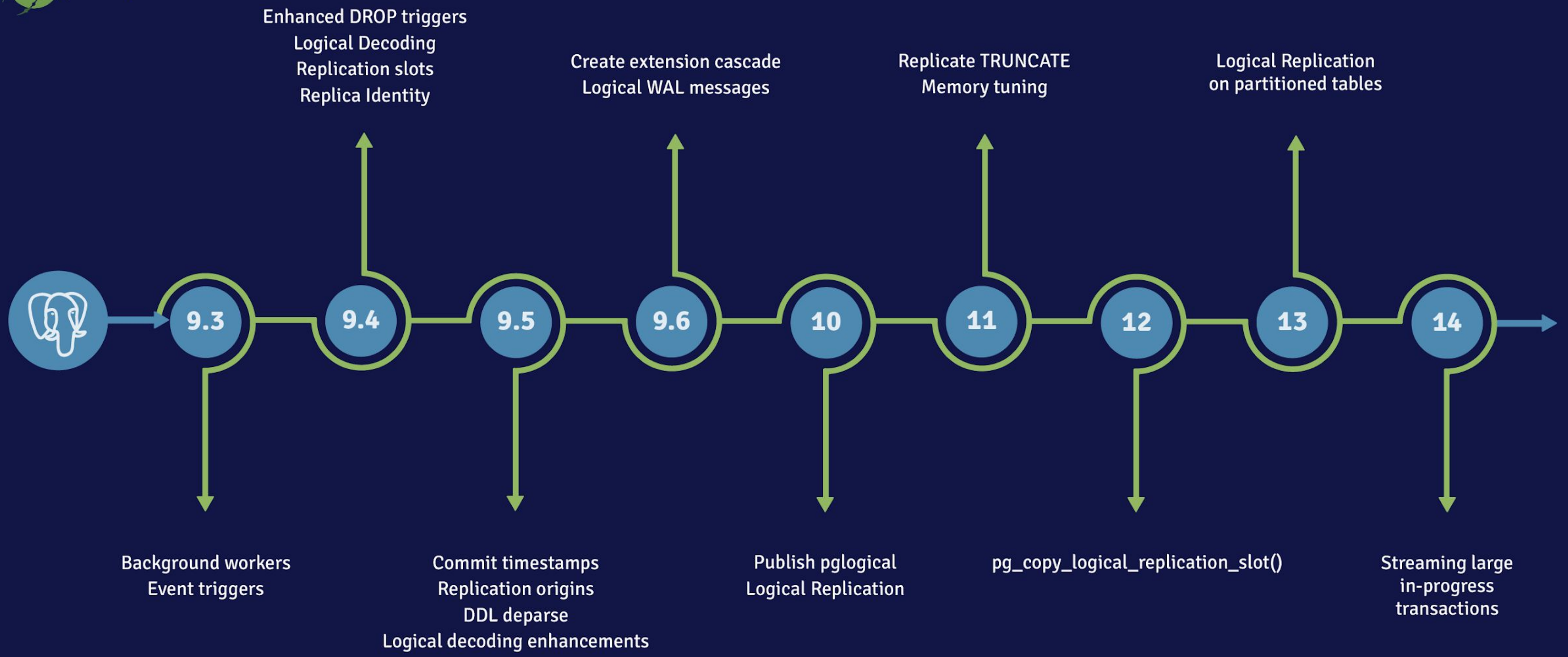
Maturity

• Trigger-based Replication	Slony, Londiste	2004
• Physical File-based Replication	PG8.2	2006
• Physical Streaming Replication	PG9.0	2010
• Logical Streaming Replication	PG10	2017
• BDR3.6		2019
• BDR3.7		2021

EDB

Postgres BDR

BDR project contributions to PostgreSQL



Thanks!

Simon.Riggs@enterprisedb.com

EDB