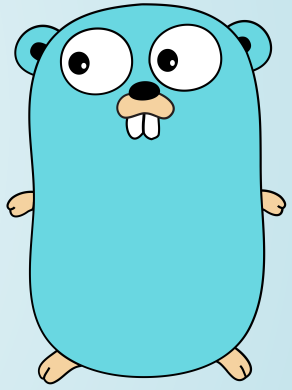
Postgres logical replication & PeerDB

Helsinki Gophers meetup

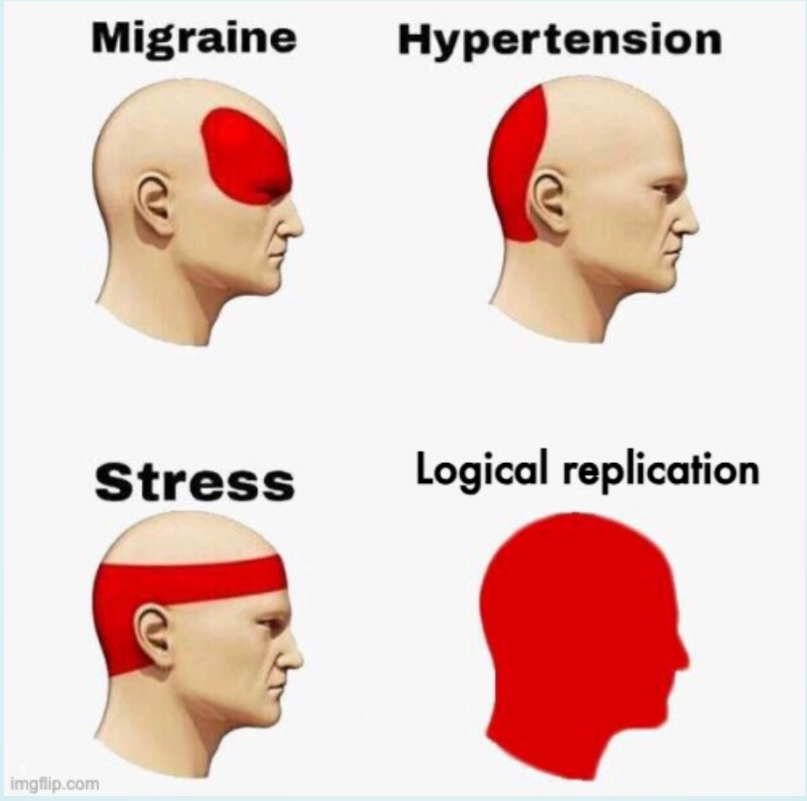
18 Mar 2025

Nikolay Kuznetsov

@nikolayk812



Types of Headaches



No-traffic AWS RDS with active logical replication slot loses 2GB of disk space every 3 hours

About me

Senior software engineer

Pre-owned project at Zalando Helsinki

C → Java → Kotlin → Go

Author of pgx-outbox library

PostgreSQL

Reliable and feature-rich

Open-source, strong community

Very permissive license

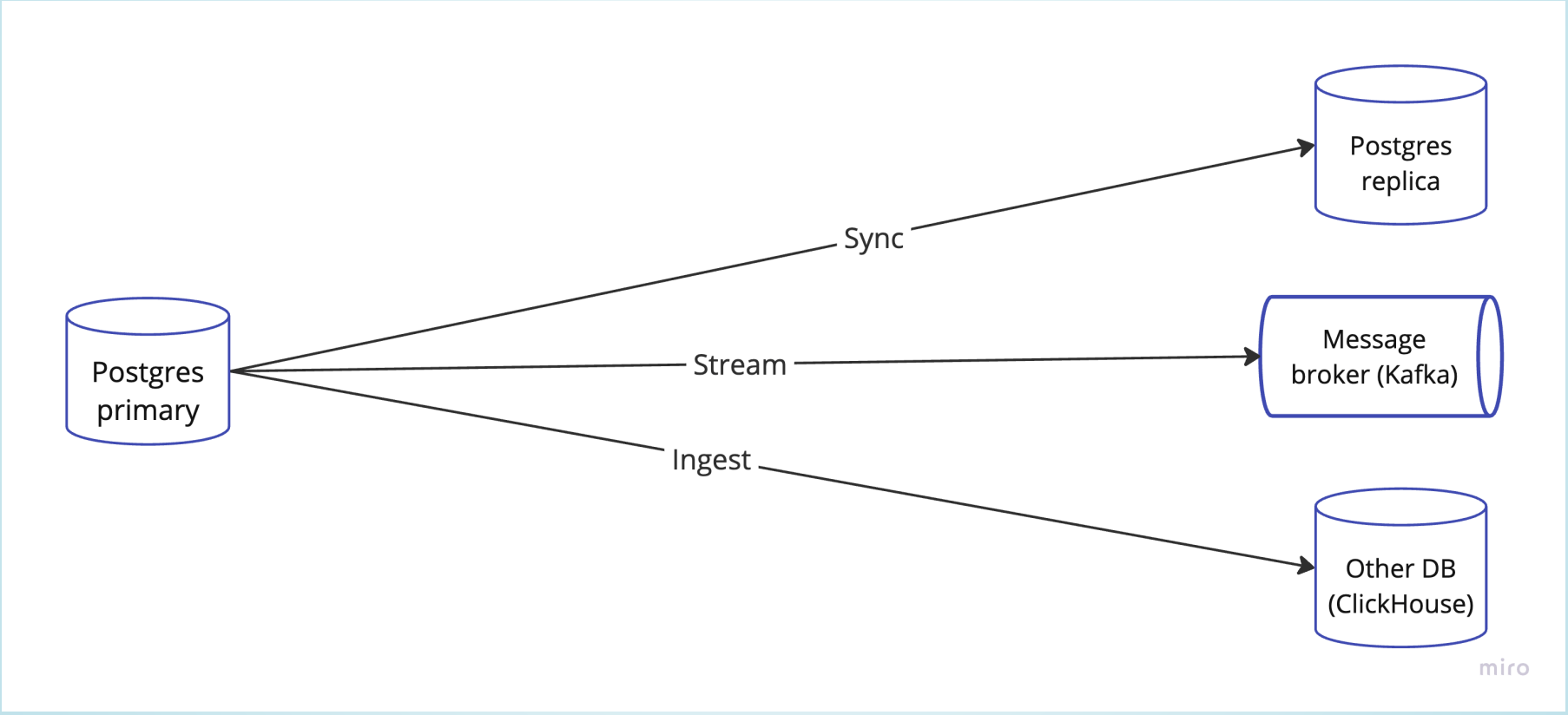
Why replicating data?

Syncing to a read replica

Streaming to a messaging system

Ingesting to another database

Replication scenarios



Replicating how?

Batch export

Physical replication (WAL)

Logical replication (WAL)

Batch export

pg_dump and *pg_restore*

COPY TO / FROM

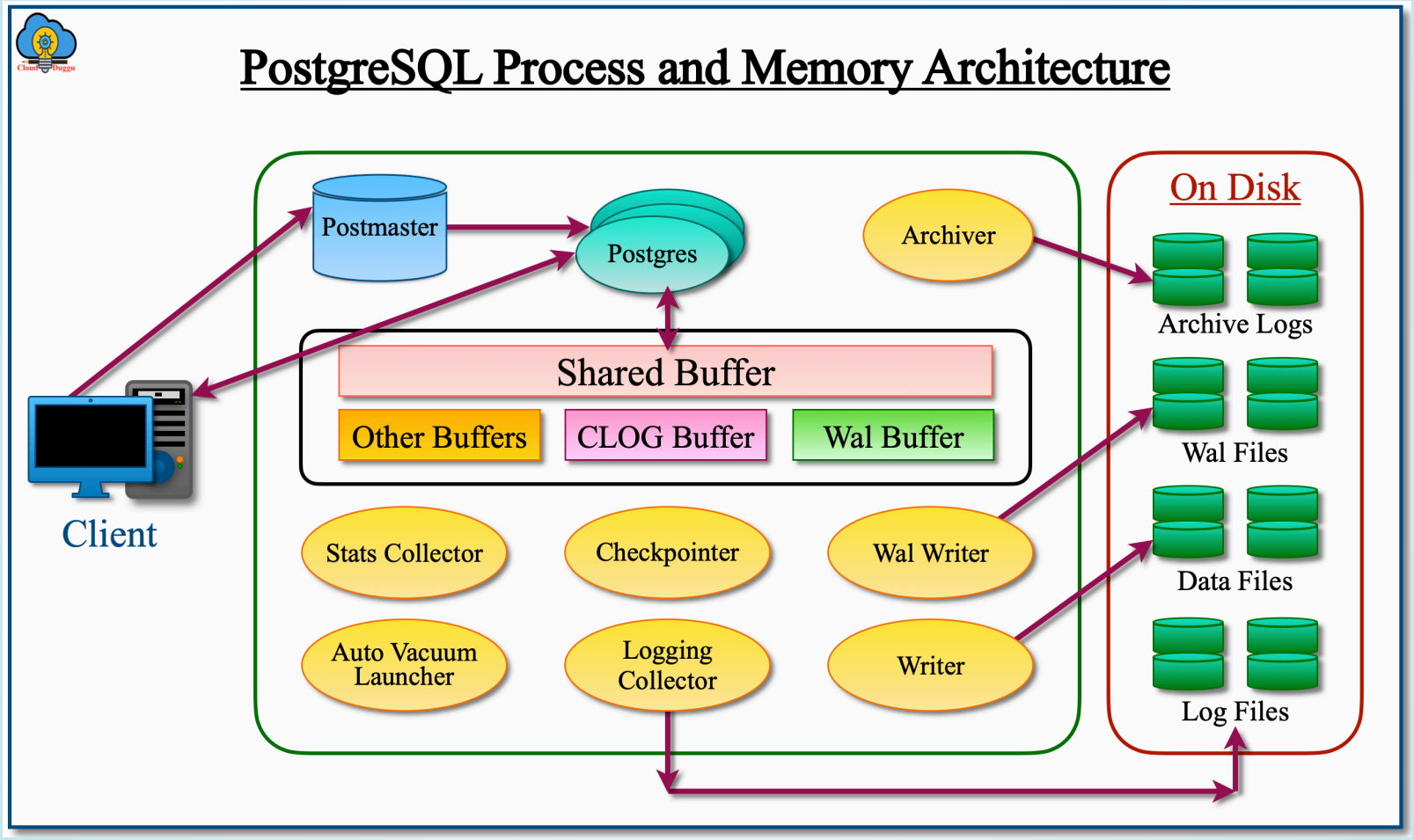
not real-time, high load, slow

Write-Ahead Log

TX1	START TRANSACTION	0/AC90FD
TX1	INSERT INTO t1 VALUES ...	0/AC9102
TX2	START TRANSACTION	0/AC9110
TX2	INSERT INTO t1 VALUES ...	0/AC912F
TX2	UPDATE t1 SET ...	0/AC9134
TX3	START TRANSACTION	0/AC9140
TX3	INSERT INTO t1 VALUES ...	0/AC914A
TX1	UPDATE t1 SET ...	0/AC9153
TX2	DELETE FROM t1 WHERE ...	0/AC9155
TX3	UPDATE t1 SET ...	0/AC915D
TX2	ROLLBACK	0/AC9160
TX3	DELETE FROM t1 WHERE ...	0/AC9168
TX3	COMMIT	0/AC916A
TX1	DELETE FROM t1 WHERE ...	0/AC9170
TX1	COMMIT	0/AC9179

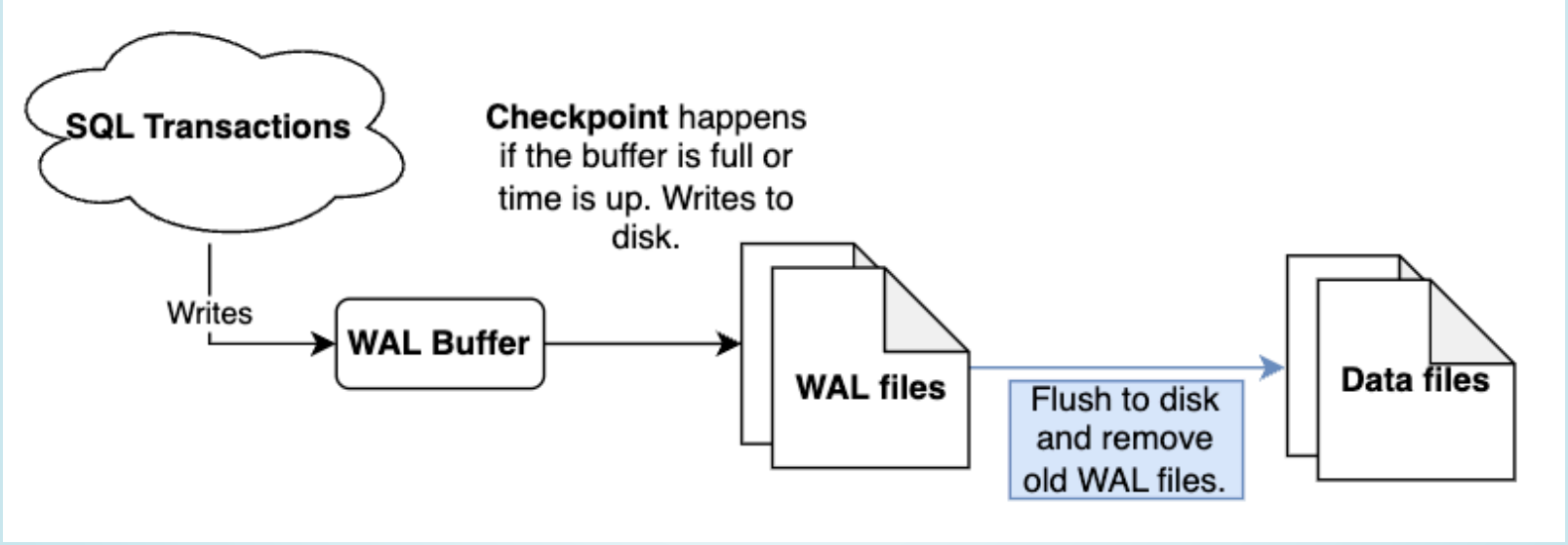
https://www.dolthub.com/blog/2024-03-08-postgres-logical-replication/

PostgreSQL Process and Memory Architecture



https://www.cloudduggu.com/postgresql/architecture/

WAL



https://www.artie.com/blogs/postgres-write-ahead-logs

Physical replication

Fast, but requires compatibility

Exact copy of the primary

Schema (DDL) + data changes (DML)

Write-Ahead Log **shipping**

Logical replication

Table-level, not full database

Write-Ahead Log **decoding**

Captures **row** changes (DML):

INSERT, *UPDATE*, *DELETE*, TRUNCATE

More flexible, but more complex

pgx driver

most popular Go driver

by Jack Christensen @jackc

11.5K stars on GitHub

PGX Top to Bottom

jackc/pglogrepl

decodes logical replication messages

includes a basic example

350 stars, 170 users

not actively maintained

Publication

```sql
CREATE PUBLICATION pub1 FOR TABLE t1, t2;
CREATE PUBLICATION pub2 FOR ALL TABLES;
-- WITH (publish = 'insert, update', 'delete', 'truncate');
SELECT * FROM pg_publication;
```

```go
query := "CREATE PUBLICATION pub1 FOR TABLE t1"

result := conn.Exec(ctx, query) // low level *pgconn.PgConn
defer result.Close()

_, err := result.ReadAll()
```

Replication slot

```
SELECT pg_create_logical_replication_slot('slot', 'pgoutput');
-- temporary slot is automatically dropped when connection is closed
SELECT pg_create_logical_replication_slot('tmp_slot', 'pgoutput', true);

SELECT * FROM pg_replication_slots;
```

```
pglogrepl.CreateReplicationSlot(ctx, conn, "tmp_slot", "pgoutput",
   pglogrepl.CreateReplicationSlotOptions{Temporary: true})
```

Logical decoding plugins

pgoutput - built-in, default

test_decoding - built-in, only for testing

wal2json - 3rd party, produces JSON

decoderbufs, *pglogical*

Plugin arguments

```
pluginArguments := []string{
    "proto_version '2'", // versions 3 and 4 are not supported
    fmt.Sprintf("publication_names '%s'", "pub1"),
    "messages 'false'",  // pg_logical_emit_message() is not used
    "streaming 'false'", // receive only committed transactions
}
```

Start replication

```
START_REPLICATION SLOT tmp_slot LOGICAL 0/16B4F20
   ("proto_version" '2',
   "publication_names" 'pub1',
   "messages" 'false',
   "streaming" 'false');
```

```
sysIdent, err := pglogrepl.IdentifySystem(ctx, conn)

pglogrepl.StartReplication(ctx, conn, "tmp_slot", sysIdent.XLogPos,
   pglogrepl.StartReplicationOptions{PluginArgs: pluginArguments})
```

Message loop

```
for {
    var rawMsg pgproto3.BackendMessage
    rawMsg, err := conn.ReceiveMessage(ctx)

    switch msg.Data[0] {
    // XLog is historical name for WAL
    case pglogrepl.XLogDataByteID: // 'w'
        err = handleXLogData(msg.Data[1:]) // actual data

    case pglogrepl.PrimaryKeepaliveMessageByteID: // 'k'
        err = handlePrimaryKeepalive(msg.Data[1:])
    }
}
```

Transactions again

TX1	START TRANSACTION	0/AC90FD
TX1	INSERT INTO t1 VALUES ...	0/AC9102
TX2	START TRANSACTION	0/AC9110
TX2	INSERT INTO t1 VALUES ...	0/AC912F
TX2	UPDATE t1 SET ...	0/AC9134
TX3	START TRANSACTION	0/AC9140
TX3	INSERT INTO t1 VALUES ...	0/AC914A
TX1	UPDATE t1 SET ...	0/AC9153
TX2	DELETE FROM t1 WHERE ...	0/AC9155
TX3	UPDATE t1 SET ...	0/AC915D
TX2	ROLLBACK	0/AC9160
TX3	DELETE FROM t1 WHERE ...	0/AC9168
TX3	COMMIT	0/AC916A
TX1	DELETE FROM t1 WHERE ...	0/AC9170
TX1	COMMIT	0/AC9179

https://www.dolthub.com/blog/2024-03-08-postgres-logical-replication/

Replication data

0/AC916A	RELATION t1 columns ...	
0/AC9140	BEGIN	TX3
0/AC914A	INSERT t1 tuple [...]	TX3
0/AC914F	INSERT t1 tuple [...]	TX3
0/AC915D	UPDATE t1 old [...] new [...]	TX3
0/AC9160	UPDATE t1 old [...] new [...]	TX3
0/AC9164	UPDATE t1 old [...] new [...]	TX3
0/AC9168	DELETE t1 tuple [...]	TX3
0/AC916A	COMMIT	TX3
0/AC9179	KEEP ALIVE	
0/AC9179	BEGIN	TX1
0/AC9102	INSERT t1 tuple [...]	TX1
0/AC9153	UPDATE t1 old [...] new [...]	TX1
0/AC9170	DELETE t1 tuple [...]	TX1
0/AC9176	DELETE t1 tuple [...]	TX1
0/AC9179	COMMIT	TX1

https://www.dolthub.com/blog/2024-03-08-postgres-logical-replication/

Handle data

```
func handleXLogData(data []byte) error {
    xld, err := pglogrepl.ParseXLogData(data)

    logicalMsg, err := pglogrepl.ParseV2(xld.WALData, false)

    switch msg := logicalMsg.(type) {
    case *pglogrepl.RelationMessageV2:
        // remember table structure: column names and types
        relations[msg.RelationID] = msg

    case *pglogrepl.InsertMessageV2:
        var raw map[string]interface{}
        raw, err = handleInsert(msg)
    }
```

Handle inserts

```
func handleInsert(msg *pglogrepl.InsertMessageV2)
                (raw map[string]interface{}, err error) {

    for columnIdx, col := range msg.Tuple.Columns {
        column, err := getRelationColumn(msg.RelationID, columnIdx)

        switch col.DataType {
        case 'n': // null
            raw[column.Name] = nil
        case 't': // text
            val, err := decodeTextColumn(col.Data, column.DataType)
            raw[column.Name] = val
        }
    }
}
```

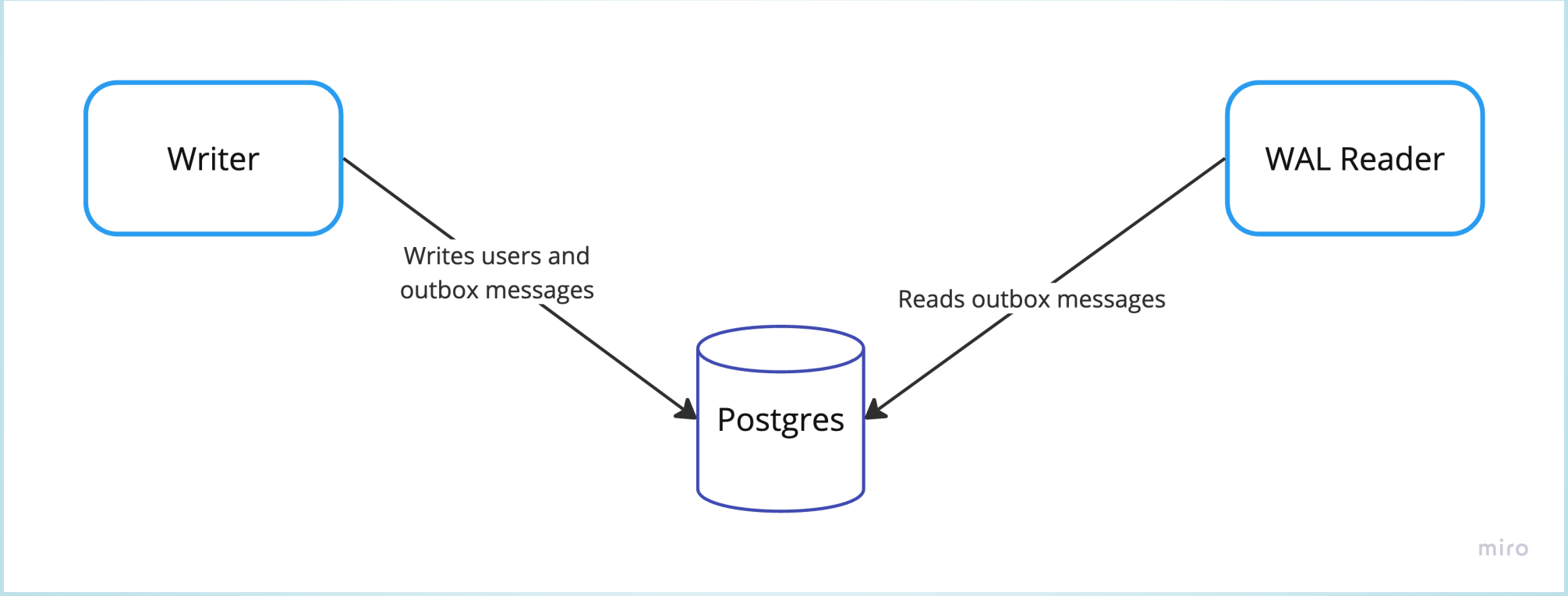
pgx-outbox WAL

pglogrepl wrapper to read only INSERTs

```
func NewReader(connStr, table, publication, slot string) (*Reader, erro

// type RawMessage map[string]interface{}
func (r *Reader) Start(ctx) (<-chan RawMessage, <-chan error)
```

Demo


Writer
WAL Reader
Writes users and
outbox messages
Reads outbox messages
Postgres
miro

PeerDB

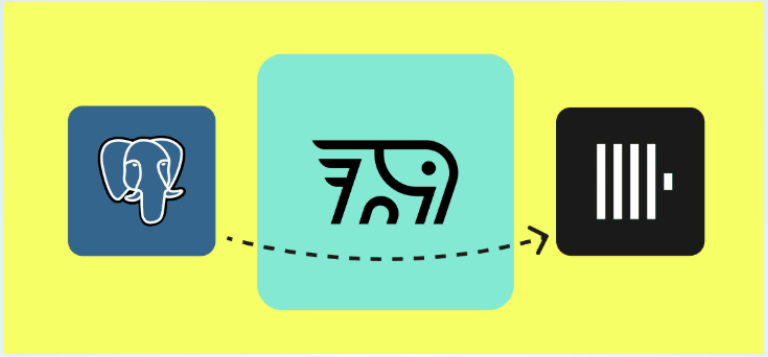
efficient data streaming from Postgres

written in Go (and Rust)

uses *Temporal* orchestration engine

open-source, Elastic License 2.0 (ELv2)

PeerDB



Y-combinator 2023

acquired by ClickHouse

integrated to ClickPipes

Takeaways

Logical replication powerful but challenging:

- acknowledging WAL positions (LSNs)

- large transactions, replication lag

- initial data load, deletion handling

Thank you!

jackc/pglogrepl

nikolayk812/pgx-outbox/wal

PeerDB-io/peerdb