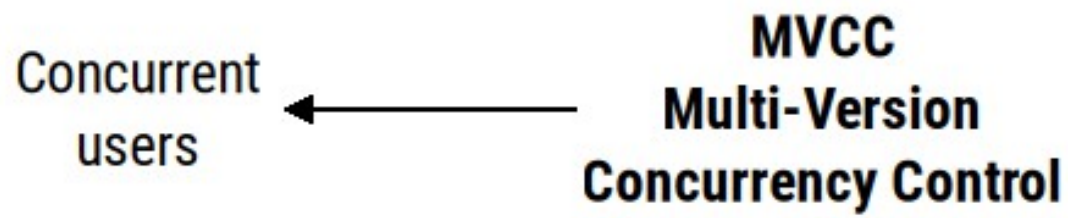
We have Multiple Concurrent Versions of this Title Trying to Understand MVCC

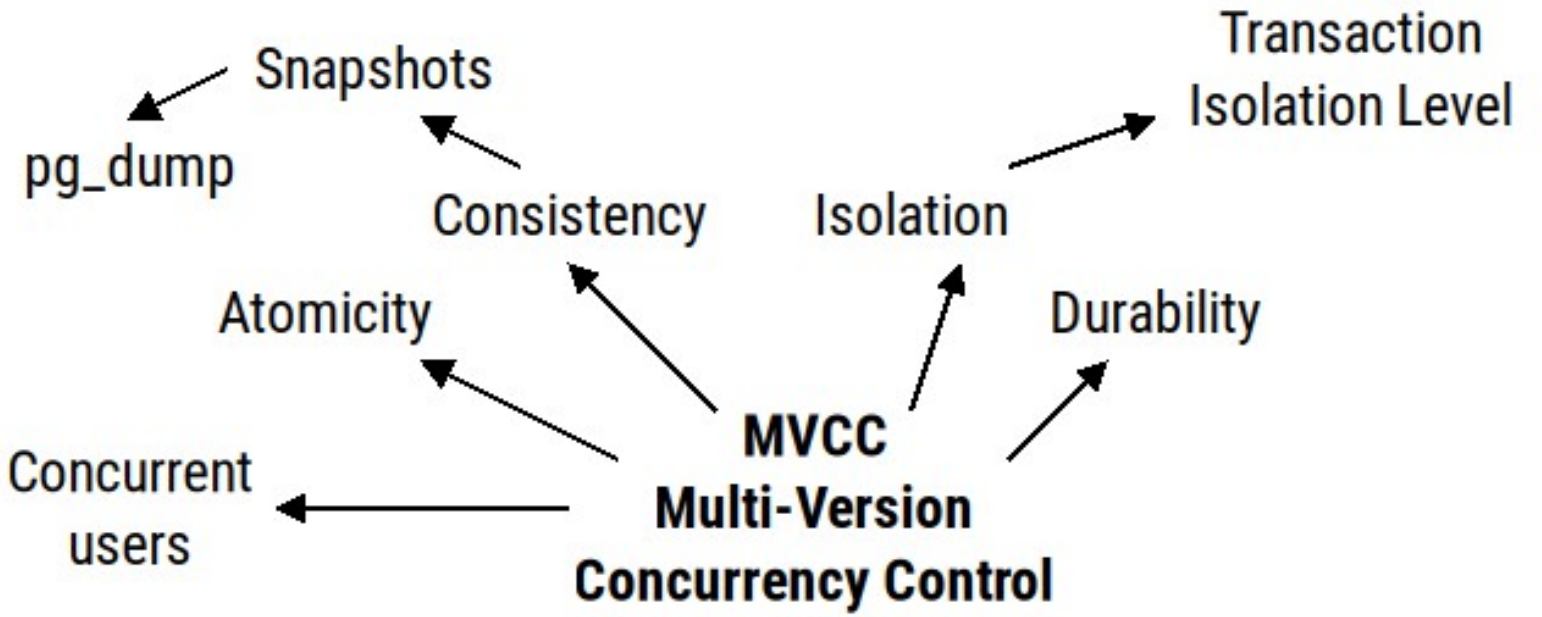
Boriss Mejías
Holistic System Software Engineer
Air Guitar Player

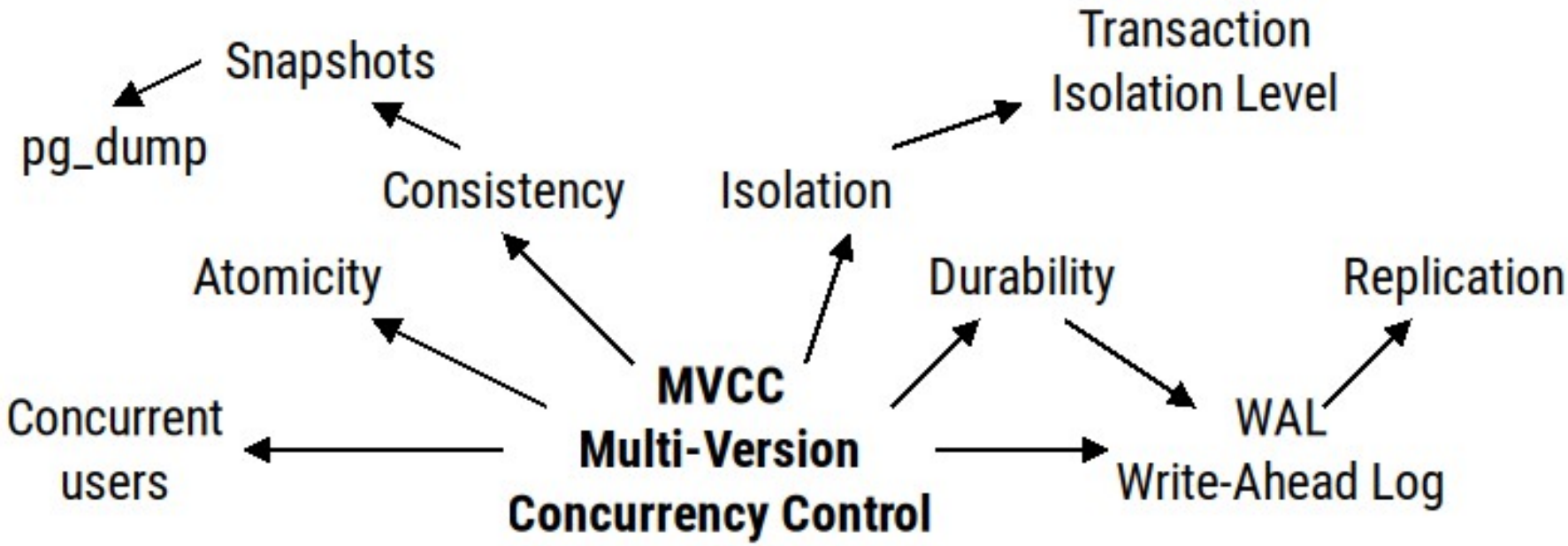
Akshat Jaimini
Software Engineer @DeepSource
Metalhead

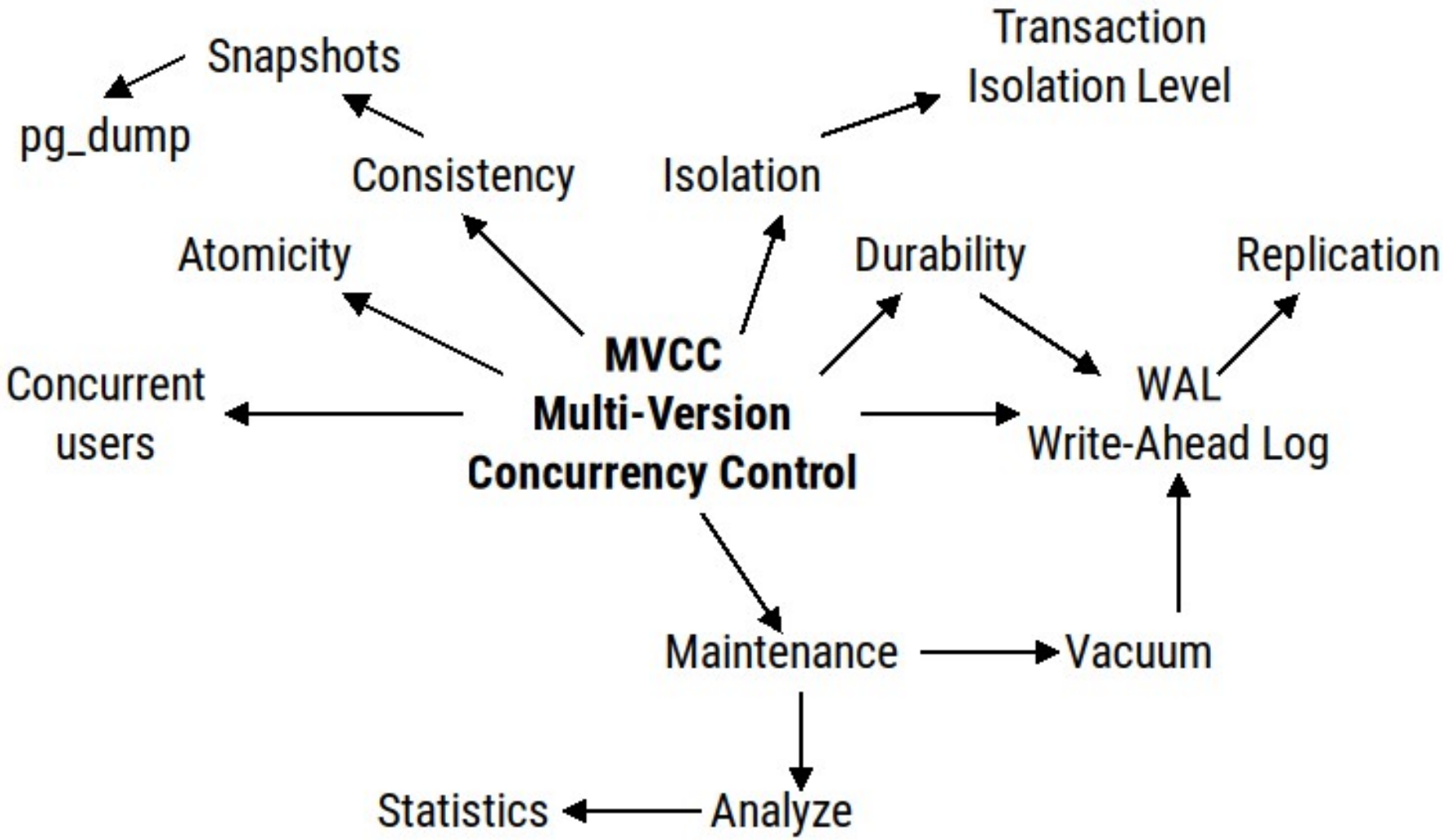
PgConf Europe – Riga - October 24, 2025

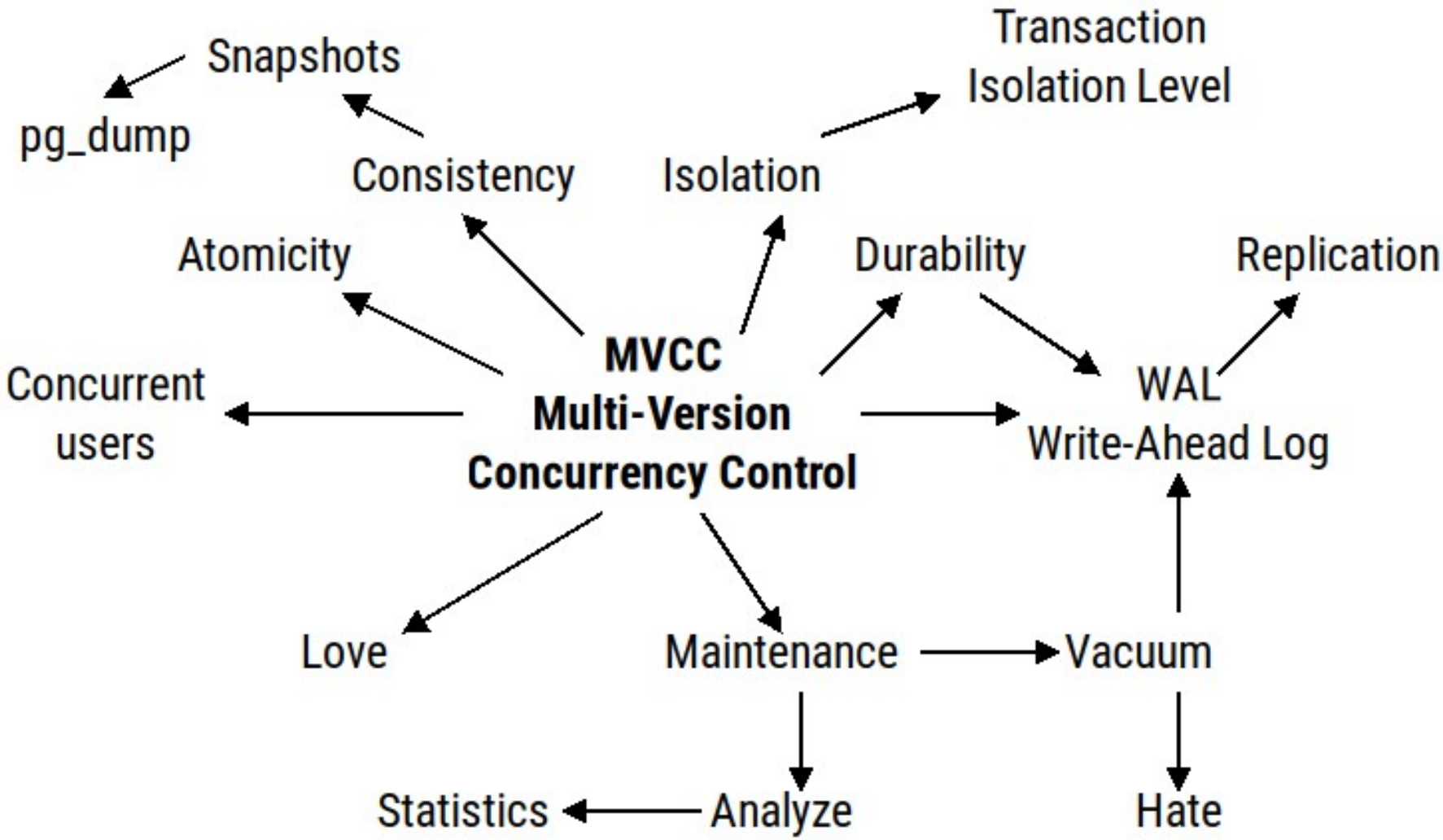
MVCC
Multi-Version
Concurrency Control











A cold but beautiful
autumn day in Riga
with fresh air and crispy grass

No user (client backend)
was connected to the database
but we observed
a large replication lag (several GB)

autovacuum was running
cleaning **dead rows**
as the gardeners in the park
were cleaning the dead leaves

Multi-Version Concurrency Control

Each row can have multiple versions
Only one version is visible to each observer
Time consistent view of the whole database
Snapshot

Everything runs in a transaction

```
INSERT INTO key_value
VALUES (1867, 'Mikhail Eisenstein');
```

Everything runs in a transaction

```
BEGIN;
INSERT INTO key_value
VALUES (1867, 'Mikhail Eisenstein');
COMMIT;
```

Concurrency

```
BEGIN;



SELECT value
  FROM key_value
 WHERE key = 1867;


COMMIT;
```

```
BEGIN;
UPDATE key_value
SET value = 'Mihails Eizenšteins'
WHERE key = 1867;


COMMIT;
```

Time

MVCC Basic Concepts

Each transaction has an identifier
`txid_current()`

Each row has visibility information
`xmin` and `xmax`

Each transaction has a status
`txid_status(txid)`

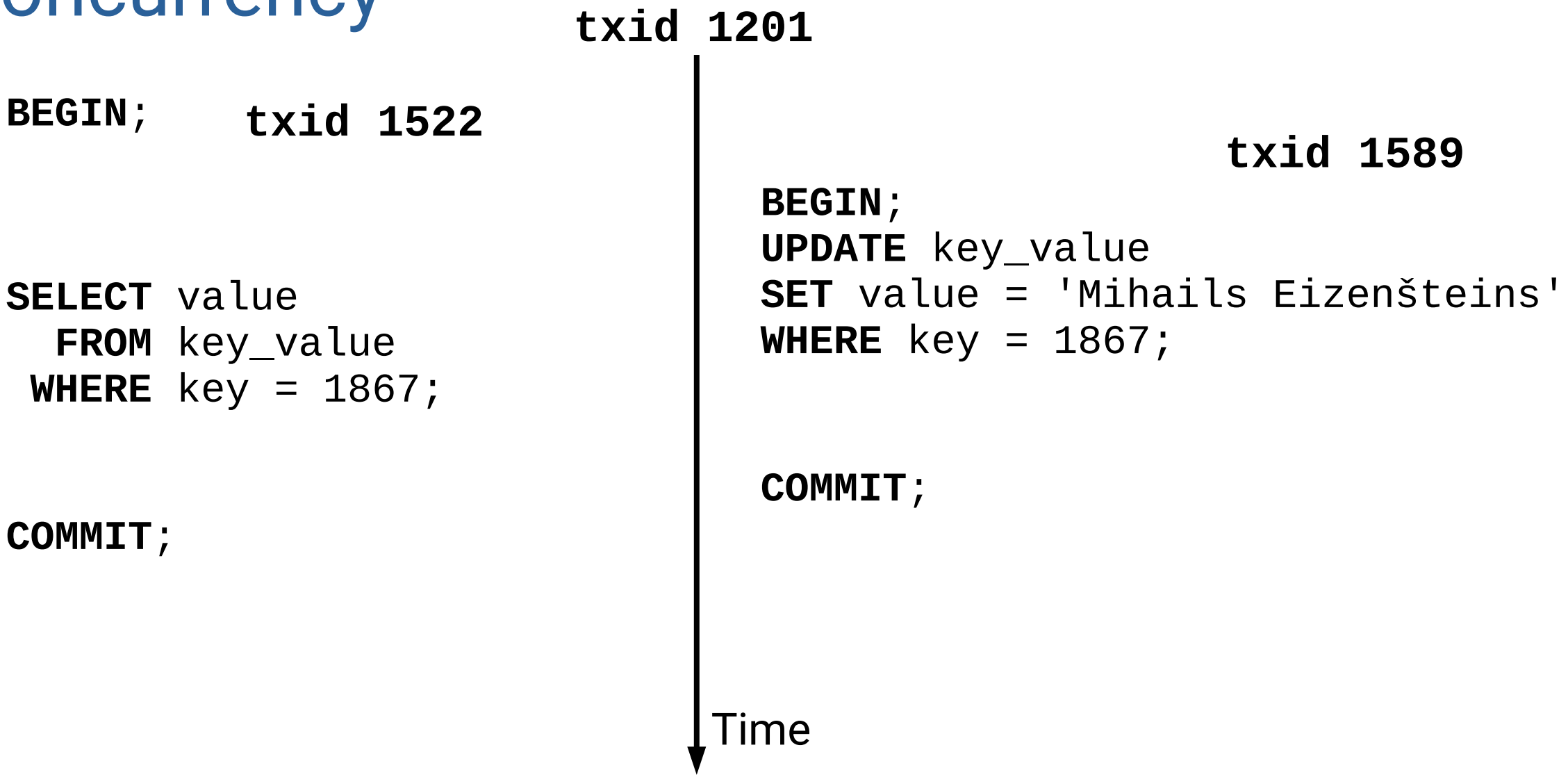
SQL write statements

`INSERT` sets **`xmin`**

`DELETE` sets **`xmax`**

`UPDATE` is delete and insert
sets **`xmax`**
creates a new version with **`xmin`**

Concurrency



Multi-Versions of the Row

```
xmin xmax    key   value
1201 0       1867  'Mikhail Eisenstein'

UPDATE →
xmin xmax    key   value
1201 1589    1867  'Mikhail Eisenstein'
1589 0       1867  'Mihails Eizenšteins'
```

Commit state of txid 1589 will be
“in progress”, “committed” or “aborted”

Dead rows

Dead rows

DELETE creates a dead row

UPDATE: *delete* and insert → dead row

ROLLBACK of writes → dead row

Too many dead rows → bloated table

Isolation Levels

Transaction Isolation Level

READ COMMITTED
REPEATABLE READ
SERIALIZABLE
READ UNCOMMITTED

Transaction Isolation Level

READ COMMITTED
REPEATABLE READ
SERIALIZABLE
~~READ UNCOMMITTED~~

Concurrency

```
BEGIN;


SELECT value
  FROM key_value
 WHERE key = 1867;



SELECT value
  FROM key_value
 WHERE key = 1867;

COMMIT;
```

```
BEGIN;
UPDATE key_value
SET value = 'Mihails Eizenšteins'
WHERE key = 1867;


COMMIT;
```

Time

Vacuum

VACUUM

VACUUM <table_name>;
Removes dead rows
Unless they are still potentially visible

Vacuum

```
BEGIN;
```

```
SELECT value
  FROM key_value
 WHERE key = 1867;
```

```
SELECT value
  FROM key_value
 WHERE key = 1867;
```

```
BEGIN;
UPDATE key_value
SET value = 'Mihails Eizenšteins'
WHERE key = 1867;

COMMIT;
```

Vacuum

```
BEGIN;
```

```
SELECT value
  FROM key_value
 WHERE key = 1867;
```

```
SELECT value
  FROM key_value
 WHERE key = 1867;
```

```
BEGIN;
UPDATE key_value
SET value = 'Mihails Eizenšteins'
WHERE key = 1867;

COMMIT;

VACUUM;
```

Time

Table pg_catalog.pg_stat_all_tables

```
-[ RECORD 1 ]-------+-----------
relid               | 24652
schemaname          | riga
relname             | key_value
n_tup_ins           | 11284004
n_tup_upd           | 6712
n_tup_del           | 425152
n_tup_hot_upd       | 713
n_live_tup          | 10858852
n_dead_tup          | 5666
n_mod_since_analyze | 6210
n_ins_since_vacuum  | 4851
```

VACUUM riga.key_value;

```
-[ RECORD 1 ]-------+-----------
relid               | 24652
schemaname          | riga
relname             | key_value
n_tup_ins           | 11284004
n_tup_upd           | 6712
n_tup_del           | 425152
n_tup_hot_upd       | 713
n_live_tup          | 10858852
n_dead_tup          | 0
n_mod_since_analyze | 6210
n_ins_since_vacuum  | 0
```

VACUUM ANALYZE riga.key_value;

```
-[ RECORD 1 ]-------+-----------
relid               | 24652
schemaname          | riga
relname             | key_value
n_tup_ins           | 11284004
n_tup_upd           | 6712
n_tup_del           | 425152
n_tup_hot_upd       | 713
n_live_tup          | 10858852
n_dead_tup          | 0
n_mod_since_analyze | 0
n_ins_since_vacuum  | 0
```

VACUUM

VACUUM <table_name>;
Removes dead rows
Unless they are still potentially visible

Hunt ‘idle in transaction’ sessions!!
And long running queries!!

Use pg_catalog.pg_stat_activity

```
SELECT pid
FROM pg_stat_activity
WHERE state = 'idle in transaction';

SELECT pid
     , query
     , wait_event_type
     , wait_event
     , now() - query_start
FROM pg_stat_activity
WHERE state='active'
AND now()-query_start > interval '15 minutes';
```

Use pg_catalog.pg_stat_activity

```
SELECT pg_terminate_backend(pid)
FROM pg_stat_activity
WHERE state = 'idle in transaction';

SELECT pg_cancel_backend(pid)
     , query
     , wait_event_type
     , wait_event
     , now() - query_start
FROM pg_stat_activity
WHERE state='active'
AND now()-query_start > interval '15 minutes';
```

VACUUM effects

VACUUM locks against DDL
INSERT, DELETE, UPDATE can still run
It does not reduce size of tables

VACUUM effects

VACUUM locks against DDL
INSERT, DELETE, UPDATE can still run
It does not reduce size of tables

Divertimento or delude

VACUUM effects

VACUUM locks against DDL
INSERT, DELETE, UPDATE can still run
It does not reduce size of tables

Divertimento or delude

VACUUM FULL

VACUUM effects

VACUUM locks against DDL
INSERT, DELETE, UPDATE can still run
It does not reduce size of tables

Divertimento or delude

VACUUM FULL
Locks Everything

VACUUM effects

VACUUM locks against DDL
INSERT, DELETE, UPDATE can still run
It does not reduce size of tables

Divertimento or delude

VACUUM FULL
Locks Everything
It creates a new table

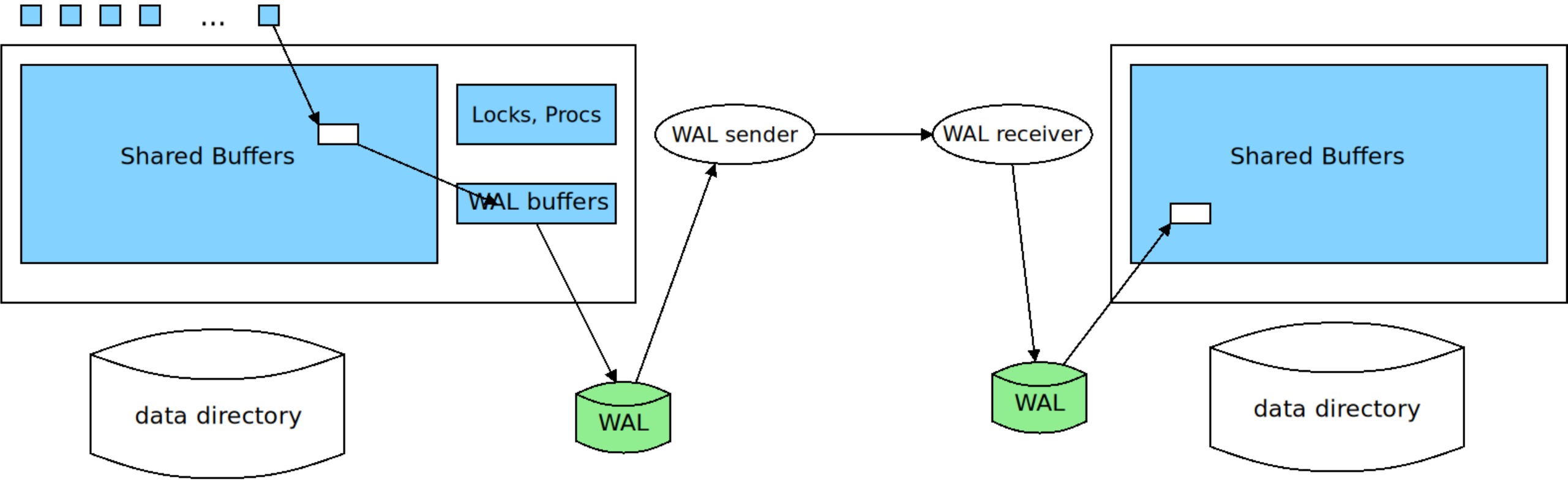
VACUUM effects

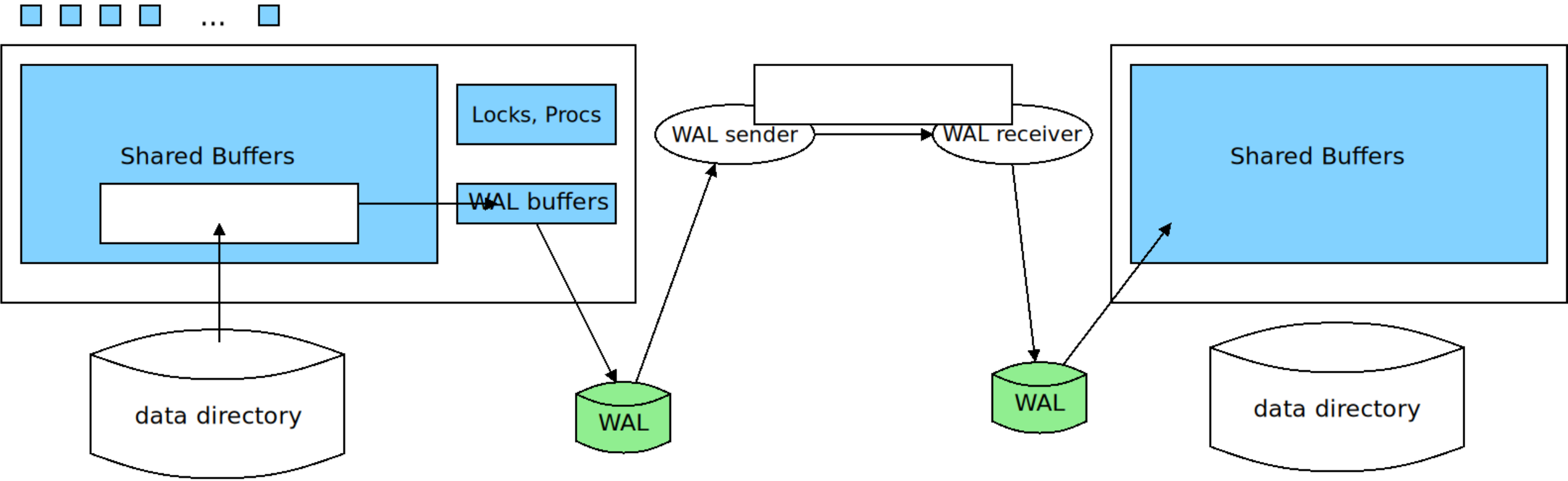
VACUUM locks against DDL
INSERT, DELETE, UPDATE can still run
It does not reduce size of tables

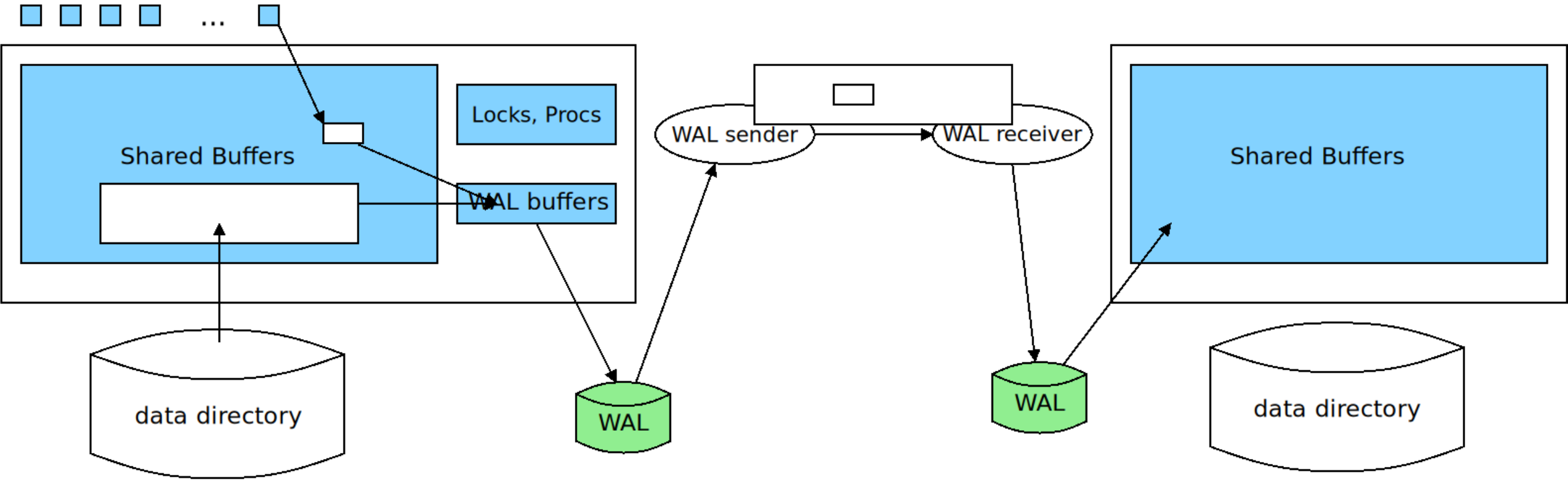
Divertimento or delude

VACUUM FULL
Locks Everything
It creates a new table
Think of a butterfly

Replication


...
Shared Buffers
Locks, Procs
WAL buffers
WAL sender
WAL receiver
Shared Buffers
data directory
WAL
WAL
data directory


...
Shared Buffers
Locks, Procs
WAL buffers
WAL sender
WAL receiver
Shared Buffers
data directory
WAL
WAL
data directory


...
Shared Buffers
Locks, Procs
WAL buffers
WAL sender
WAL receiver
Shared Buffers
data directory
WAL
WAL
data directory

autovacuum

Autovacuum

It runs **VACUUM** and **ANALYZE**
Runs all the time
No scheduling, just nap times
It cancels itself to avoid blocking user’s actions

Table pg_catalog.pg_stat_all_tables

```
-[ RECORD 1 ]-------+-----------
relid               | 24652
schemaname          | riga
relname             | key_value
last_vacuum         | 2025-10-17 14:18:24
last_autovacuum     | 2025-10-23 06:23:16
last_analyze        | 2025-03-13 12:58:42
last_autoanalyze    | 2025-10-23 06:23:16
vacuum_count        | 7
autovacuum_count    | 6128
analyze_count       | 10676
autoanalyze_count   | 10
```

Check the progress

Table pg_catalog.pg_stat_progress_vacuum

```
-[ RECORD 1 ]------+-----------
pid                | 27666
datid              | 18613
datname            | riga
relid              | 24652
phase              | performing final cleanup
heap_blks_total    | 1
heap_blks_scanned  | 1
heap_blks_vacuumed | 1
index_vacuum_count | 0
max_dead_tuples    | 291
num_dead_tuples    | 0
```

Table pg_catalog.pg_stat_progress_analyze

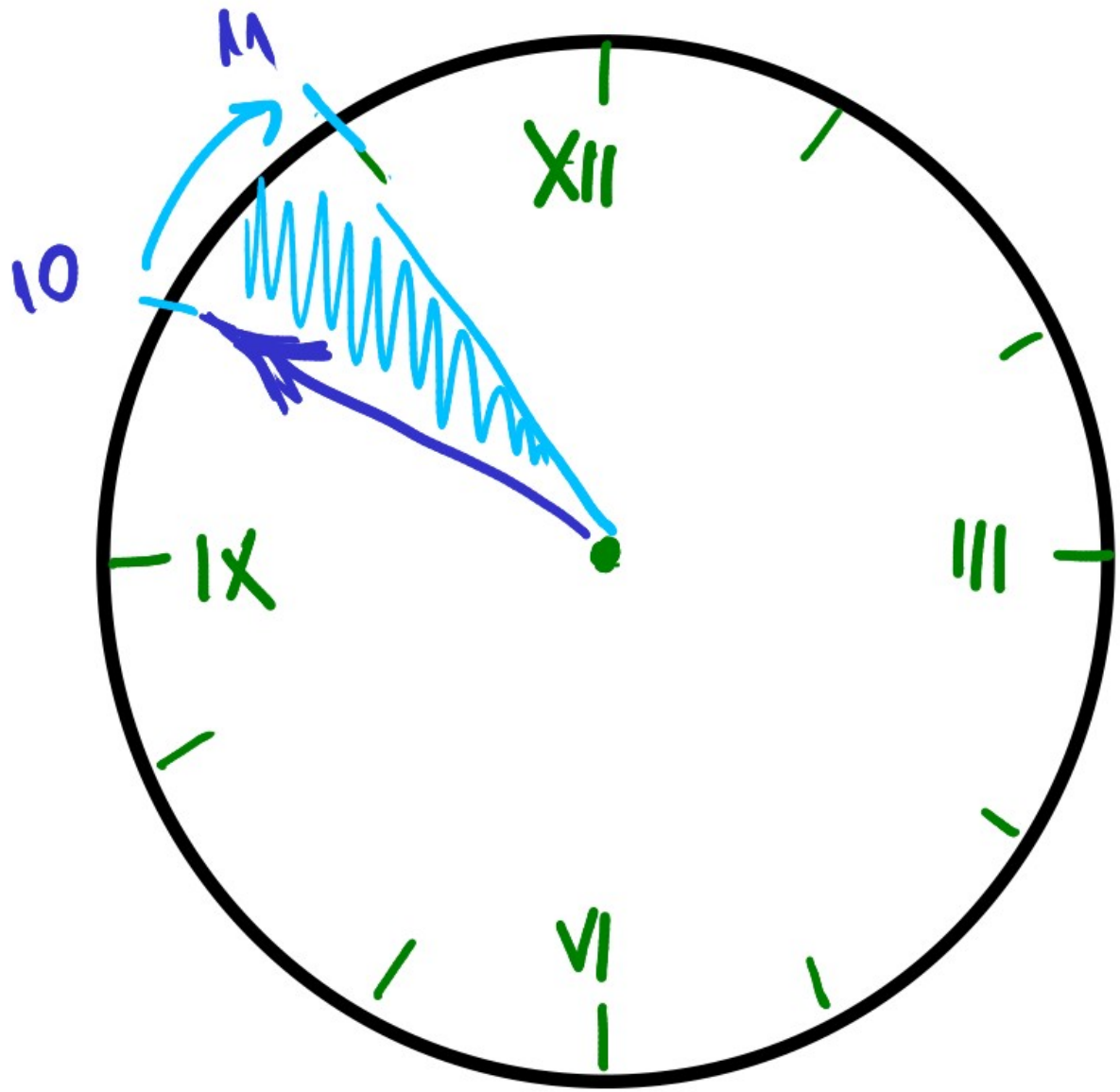
```
-[ RECORD 1 ]----------+-----------
pid                    | 3012146
datid                  | 18613
datname                | riga
relid                  | 66209
phase                  | acquiring sample rows
sample_blks_total      | 30000
sample_blks_scanned    | 3454
ext_stats_total        | 0
ext_stats_computed     | 0
child_tables_totals    | 0
child_tables_done      | 0
current_child_table_relid | 0
```

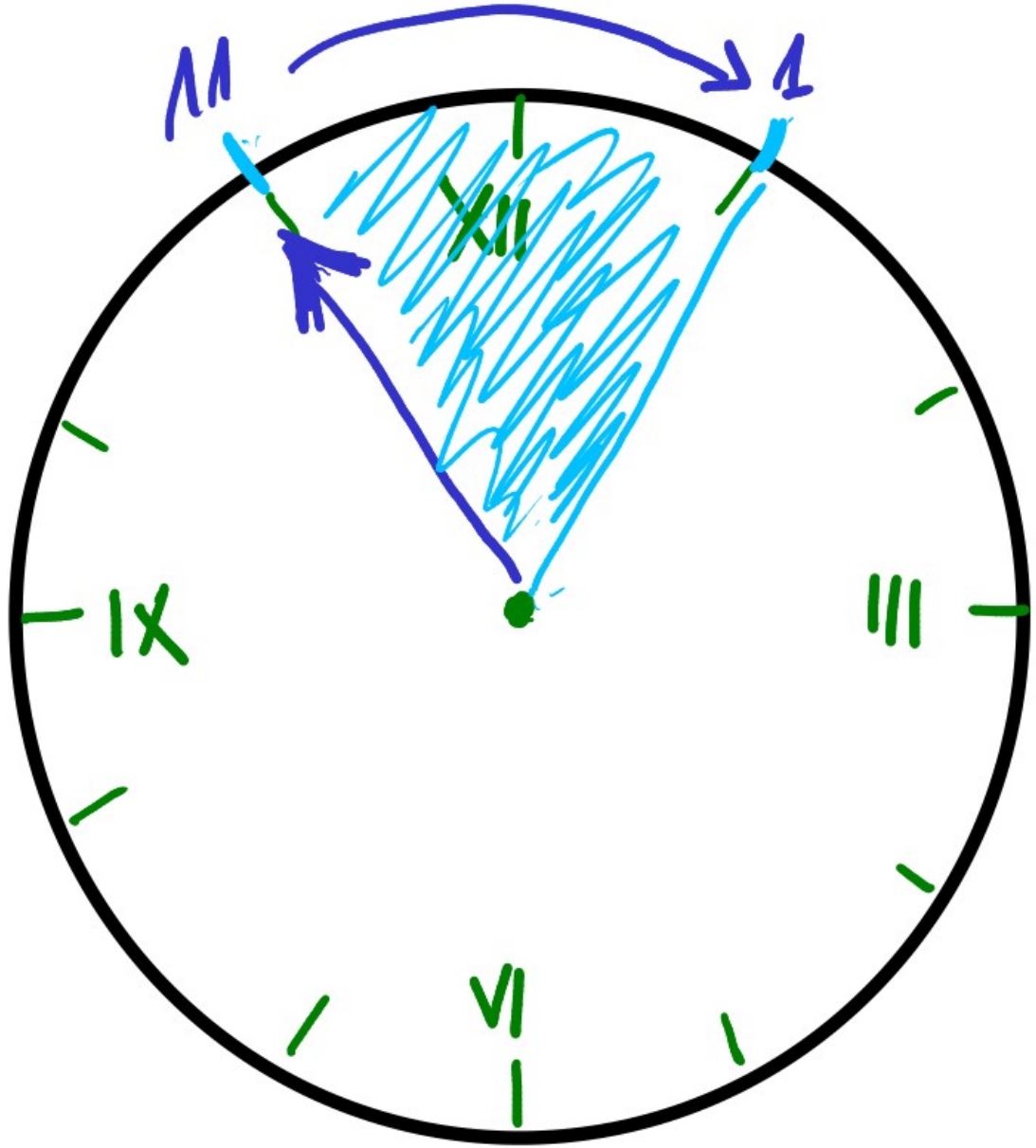
At the Pub

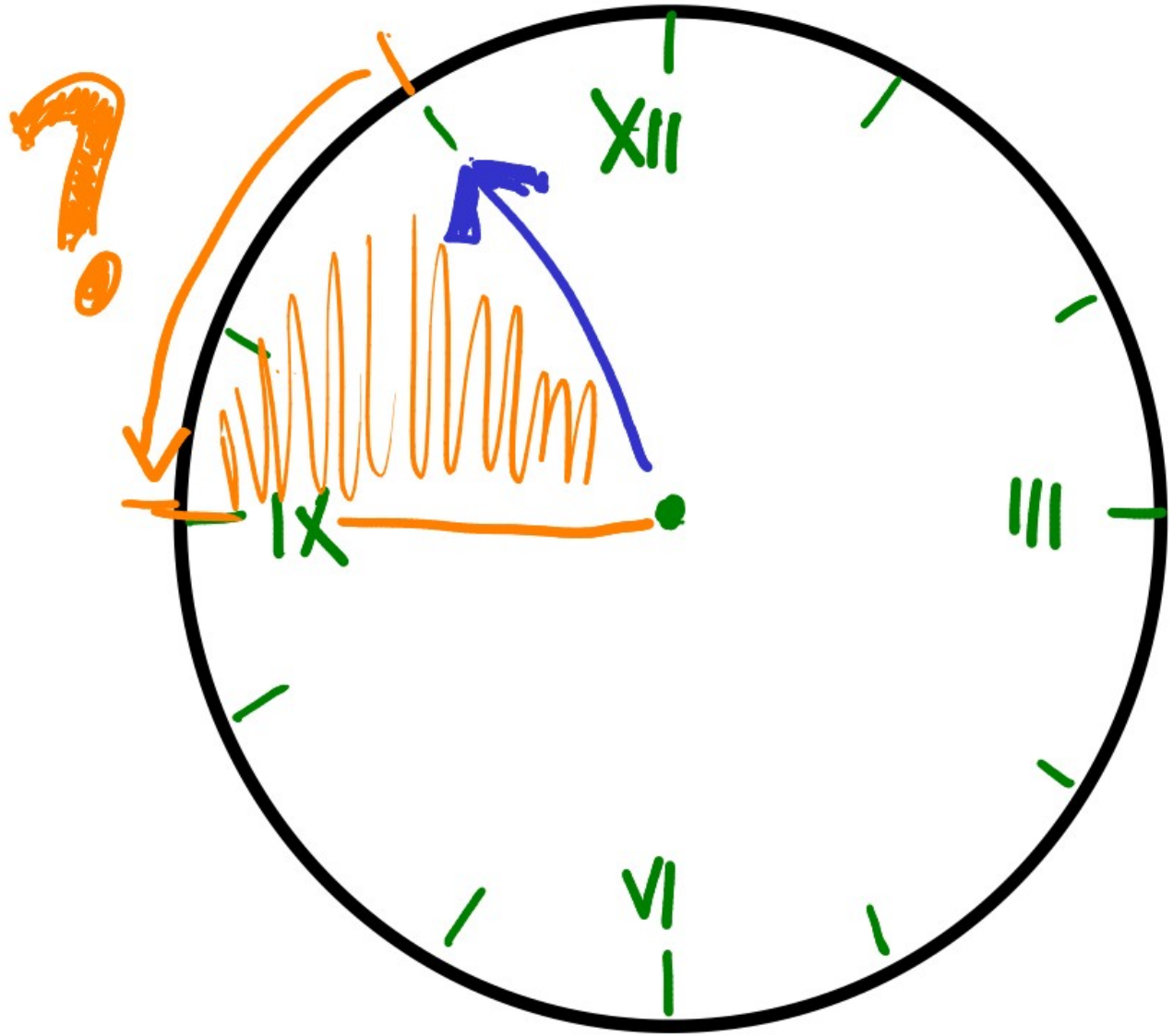
Freezing

Transactions are 4 byte integers

Counter wraps every 4 billion xids


11
10
XII
IX
III
VI


11
1
XII
IX
III
VI


XII
III
VI
IX

Freezing

Transactions are 4 byte integers

Counter wraps every 4 billion xids

Visibility of 2 billion xids to the past

Very old rows get frozen

Table pg_catalog.pg_database

```
datname        | name
datdba         | oid
encoding       | integer
datcollate     | name
datctype       | name
datistemplate  | boolean
datallowconn   | boolean
datconnlimit   | integer
datlastsysoid  | oid
datfrozenxid   | xid
datminmxid     | xid
dattablespace  | oid
datacl         | aclitem[]
```

Table pg_catalog.pg_database

```
datname       | name
datdba        | oid
encoding      | integer
datcollate    | name
datctype      | name
datistemplate | boolean
datallowconn  | boolean
datconnlimit  | integer
datlastsysoid | oid
datfrozenxid  | xid
datminmxid    | xid
dattablespace | oid
datacl        | aclitem[]
```

Table pg_catalog.pg_database

```
SELECT datname
     , datfrozenxid
     , age(datfrozenxid)
FROM pg_database;
```

Closing Words

Multi-Version Concurrency Control and VACUUM

Multi-Version Concurrency Control is crucial for ACID

Dead rows are created → Maintenance

VACUUM removes dead rows

Autovacuum does it for you

Plenty of data exposed at the catalog

Thank You

🐘 @tchorix@mastodon.world
🦋 @tchorix.bsky.social
💼 linkedin.com/in/boriss-mejías-4637401
🐙 github.com/bmejias

Ⓜ @medium.com/@destrex271
🦋 @kyllex5.bsky.social
💼 linkedin.com/in/akshat-jaimini-05a610203/
🐙 github.com/destrex271