Table Repacking, done right

Antonín Houska – Postgres developer, CYBERTEC
Álvaro Herrera – Postgres developer, EDB

Postgres Conference Europe 2025
Riga, Latvia
23-24 October 2025

Antonín Houska

- Antonín Houska, CYBERTEC
- ah@cybertec.at
- Postgres contributor since 2012
- Working as a Postgres developer for CYBERTEC since 2012

Álvaro Herrera

- Álvaro Herrera, EDB
- alvherre@kurilemu.de
- Postgres contributor since 2002
- Working as a Postgres developer for EDB since 2012

Talk structure

1. The problem: table bloat
2. The historical solution: VACUUM and friends
3. Third-party solutions
   - `pg_reorg`
   - `pg_repack`
   - `pg_squeeze`
4. Non-concurrent REPACK
5. REPACK CONCURRENTLY

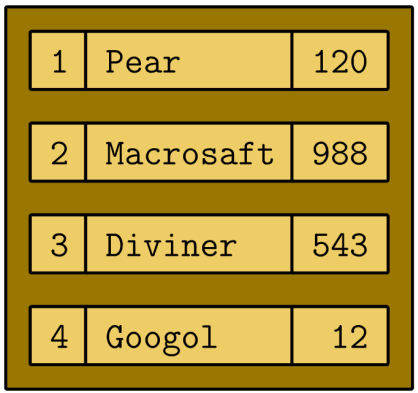
Table bloat

- Comes from *non-overwriting* MVCC implementation
- Non-overwriting: old versions of updated tuples are not immediately removable
- *vacuuming*[1] takes care of them afterwards

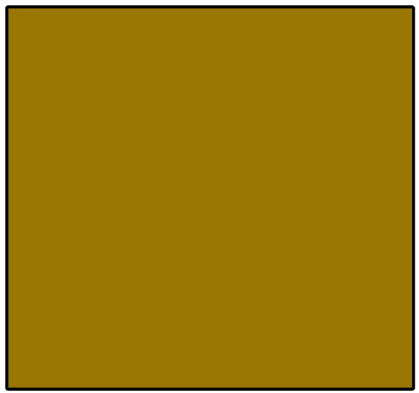
[1] And HOT-pruning.

Table bloat: other databases

table: companies

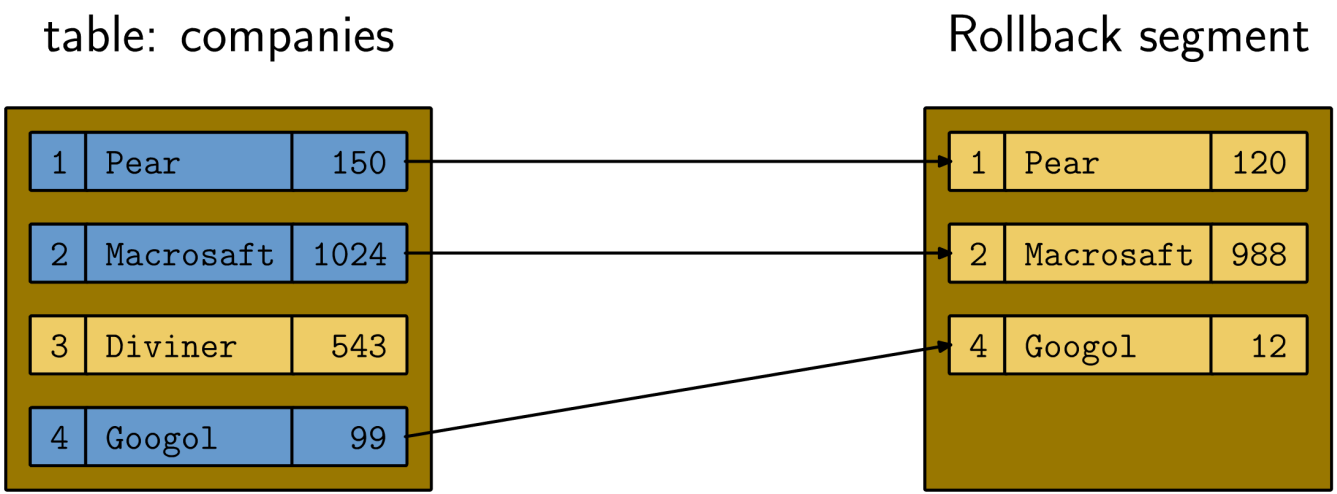


Rollback segment



An overwriting storage manager might use a “rollback segment”

Table bloat: other databases



As the table is updated, old tuples versions are moved to the rollback segment. The table doesn't need later cleanup.

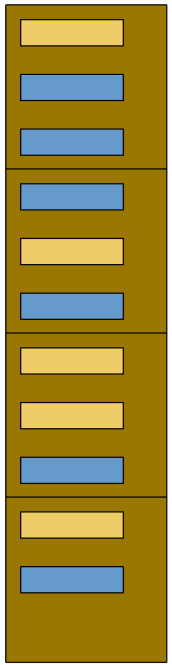
A rollback segment?

- requires handling of disk space for it
- and later cleanup
- notably: rollbacks are expensive
- and is more difficult to implement
- Postgres tried: see `zheap`
- Not yet achieved!

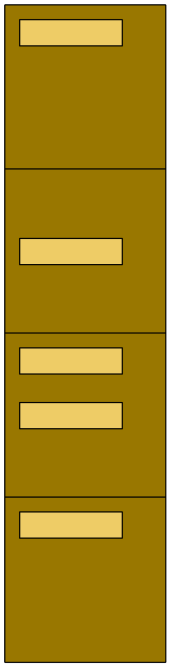
A rollback segment?

- requires handling of disk space for it
- and later cleanup
- notably: rollbacks are expensive
- and is more difficult to implement
- Postgres tried: see `zheap`
- Not yet achieved!

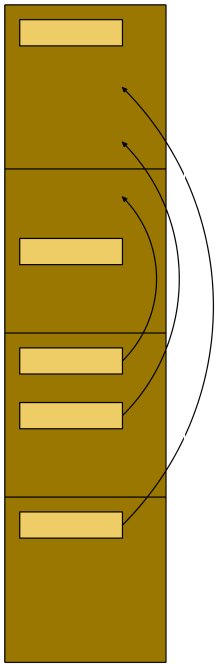
Berkeley: Ancient vacuuming technique



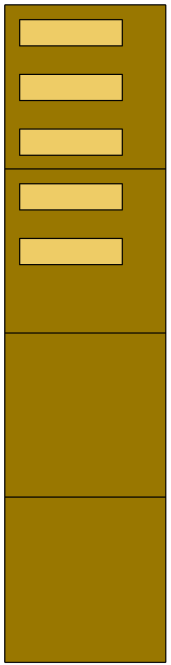
some dead tuples



dead tuples are removed



tuples moved to final locations



tuples in final locations

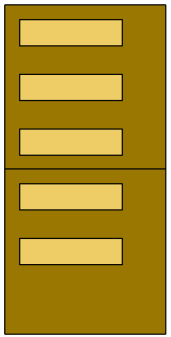


table can be truncated

“Lazy” vacuum?

- But this required “access exclusive” lock on the table
- Commit 4046e58c2478:
  Initial implementation of concurrent VACUUM.
  Tom Lane

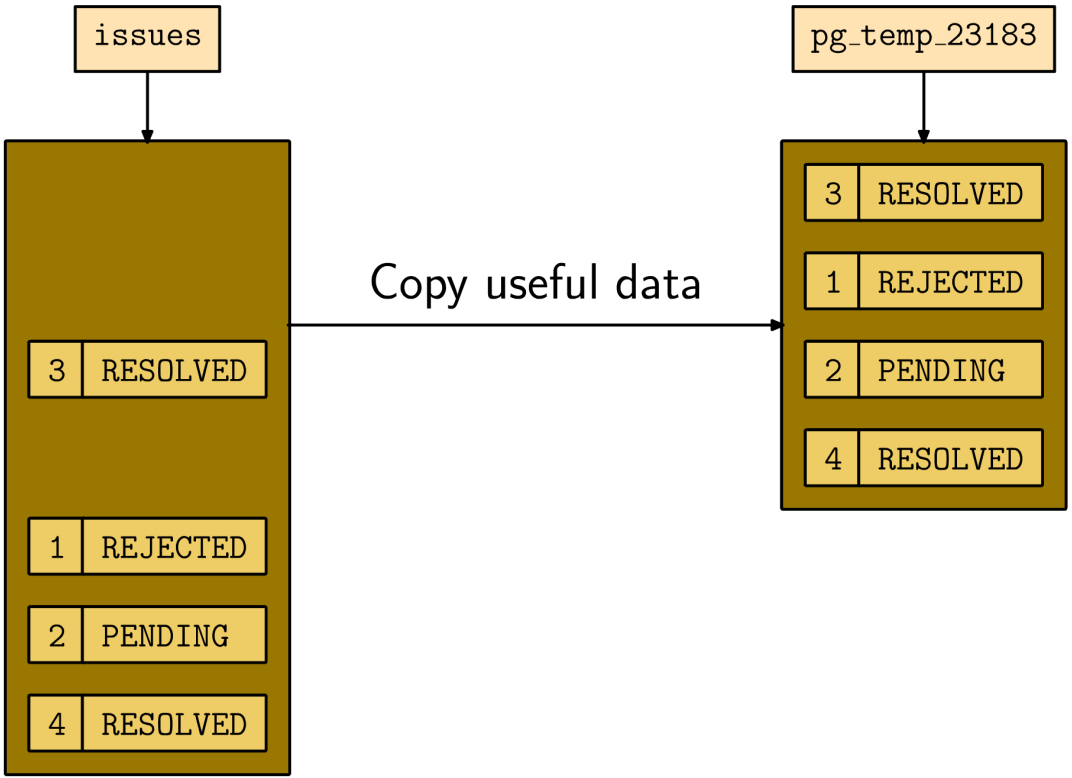
  Fri Jul 13 2001, Postgres 7.2
- Doesn’t require access exclusive lock anymore
- Operation can continue
- Disadvantage: surviving tuples cannot be moved across pages
- Old-style vacuum is renamed `VACUUM FULL`

A faster VACUUM FULL?

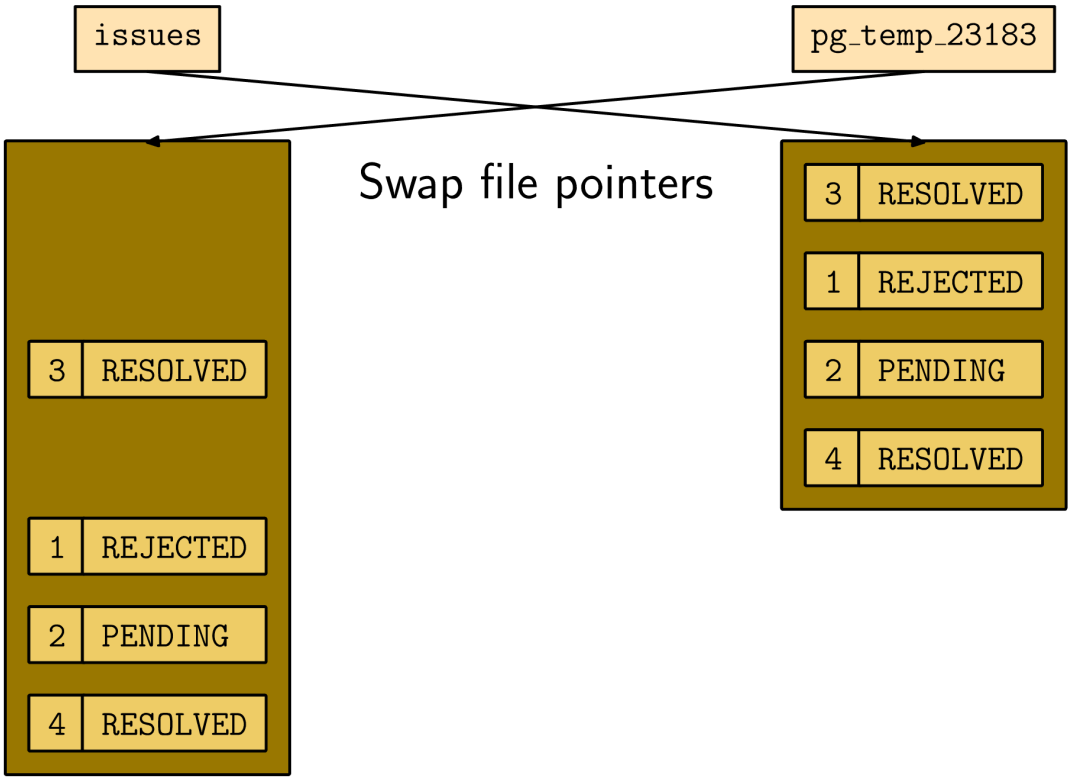
- Yes! *"Let's use `CLUSTER`," someone said*
- Commit 946cf229e89f:
  Support rewritten-based full vacuum as VACUUM FULL. Traditional VACUUM FULL was renamed to VACUUM FULL INPLACE.
  Itagaki Takahiro

  Wed Jan 6 2010, Postgres 9.0
- In 2010 (Postgres 9.0), VACUUM FULL was changed to use the CLUSTER code
- How does this work?

CLUSTER-based VACUUM FULL



CLUSTER-based VACUUM FULL



VACUUM FULL INPLACE removed

- Commit 0a469c87692d:
  Remove old-style VACUUM FULL (which was known for a little while as VACUUM FULL INPLACE), along with a boatload of subsidiary code and complexity. Per discussion, the use case for this method of vacuuming is no longer large enough to justify maintaining it; not to mention that we don't wish to invest the work that would be needed to make it play nicely with Hot Standby.
  Tom Lane

  Mon Feb 8 2010, Postgres 9.0

pg_reorg

- Created in 2008 by NTT
- https://ossc-db.github.io/pg_reorg/

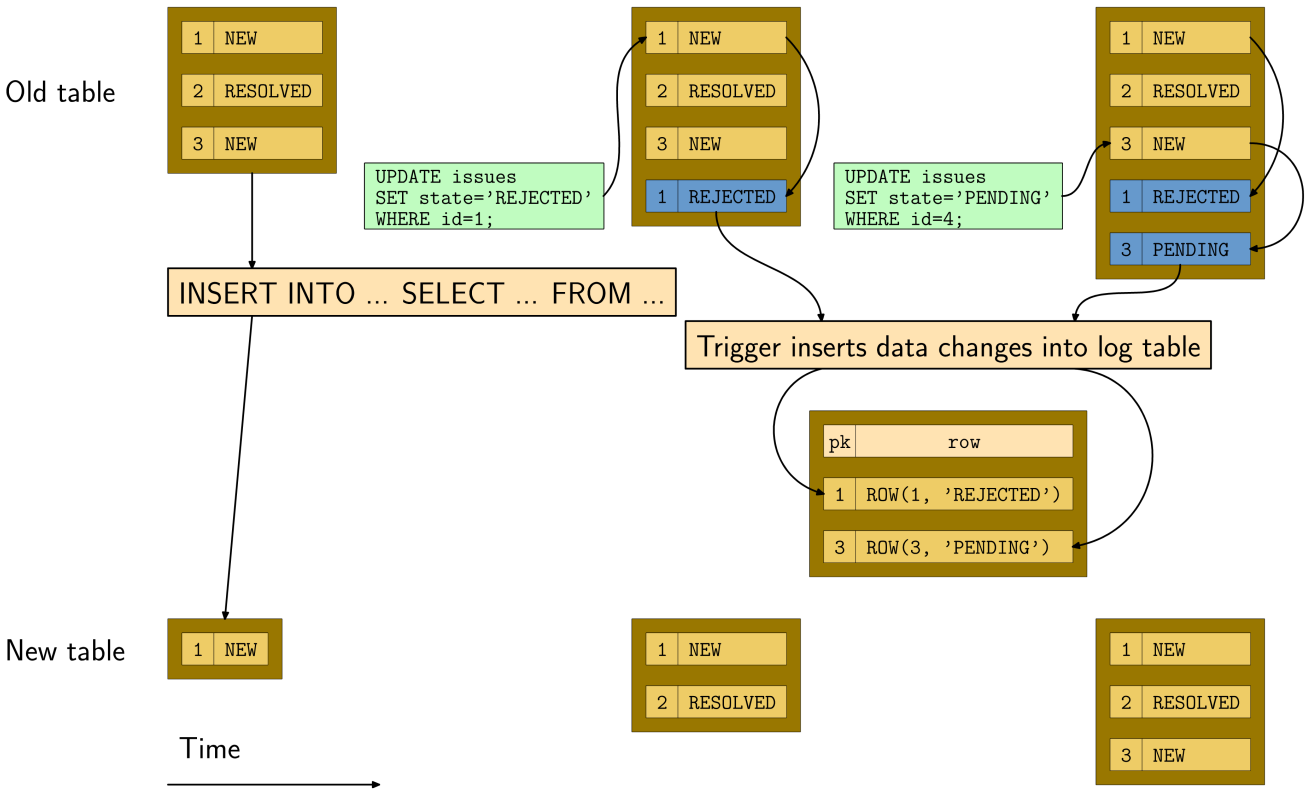
  "The module is developed to be a better alternative of CLUSTER and VACUUM FULL."
- Featured in Depesz's blog in 2011: Bloat Happens

  "All in all – it's a great tool, which does amazing job."
- Last release was 1.1.9 in 2013
- Pronounced dead in 2020

pg_repack

- Forked from pg_reorg in 2012
- Implemented in two parts:
  - A few server-side SQL and C functions
  - Workflow controlled by a client application

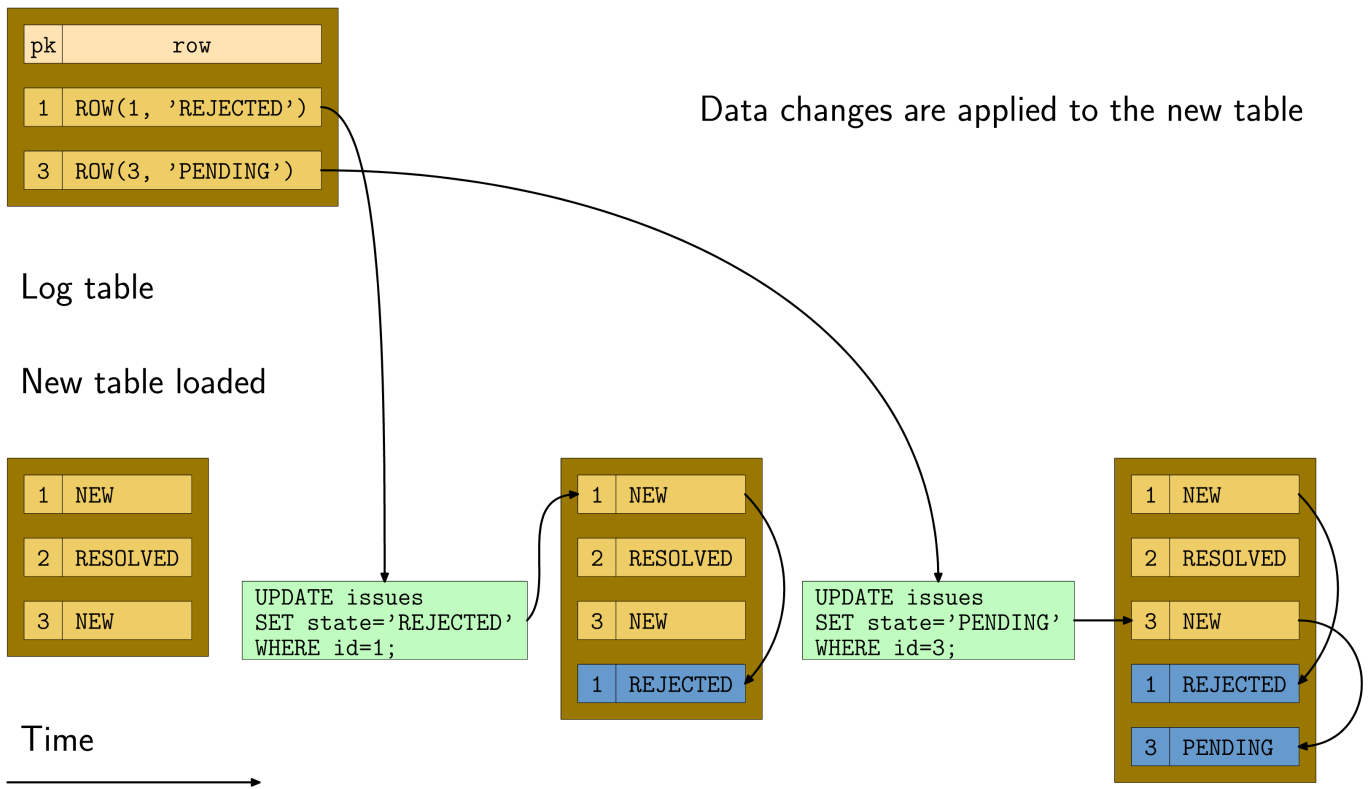
pg_repack



EDB POSTGRES AI

CYBERTEC

pg_repack


pk
row
1
ROW(1, 'REJECTED')
3
ROW(3, 'PENDING')
Data changes are applied to the new table
Log table
New table loaded
1
NEW
2
RESOLVED
3
NEW
UPDATE issues
SET state='REJECTED'
WHERE id=1;
1
NEW
2
RESOLVED
3
NEW
1
REJECTED
UPDATE issues
SET state='PENDING'
WHERE id=3;
1
NEW
2
RESOLVED
3
NEW
1
REJECTED
3
PENDING
Time

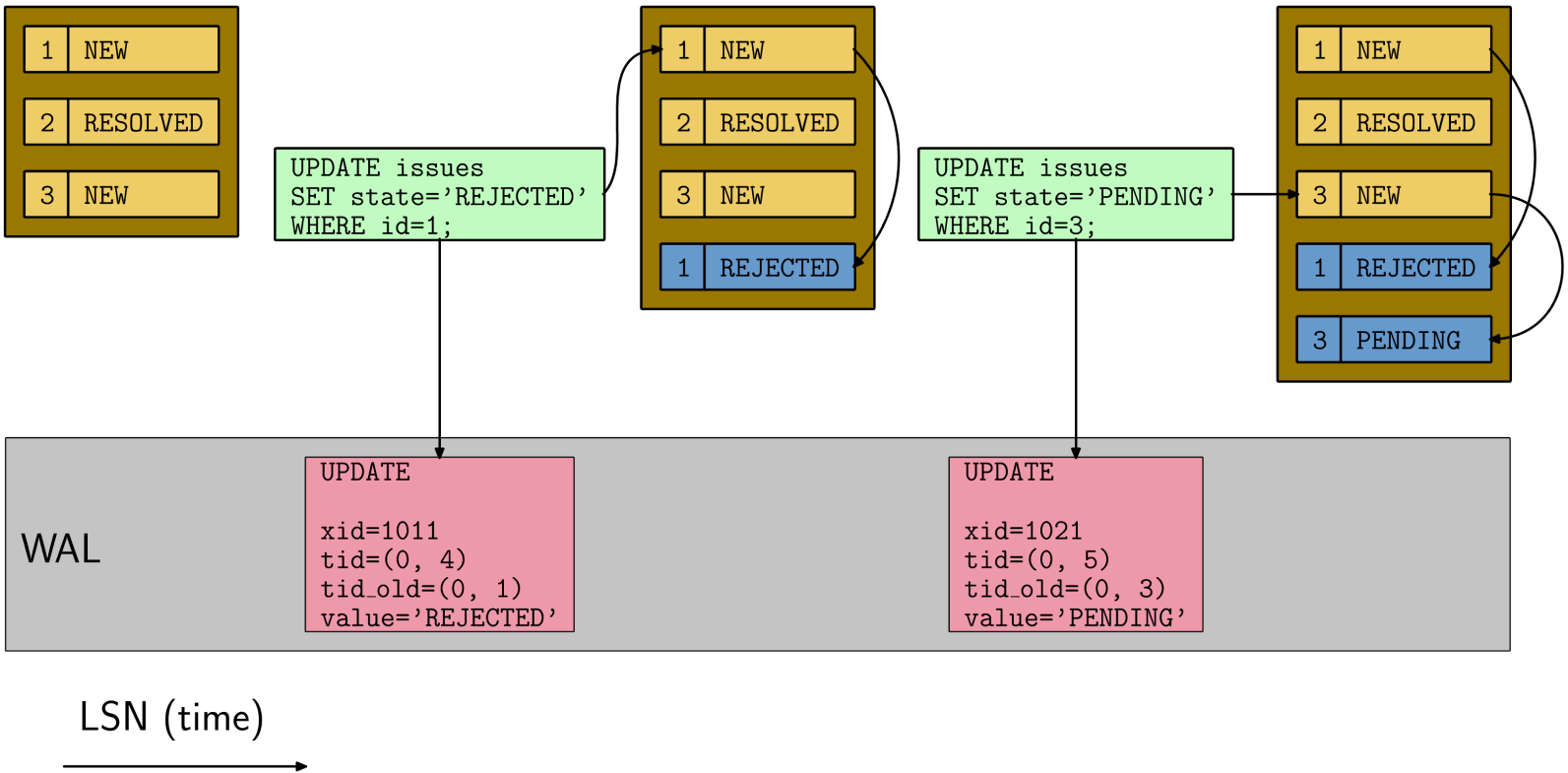
pg_repack's algorithm

- Like `VACUUM FULL`, it removes bloat by copying the useful data to a new table, swaps underlying files and drops the new table.
- Exclusive lock is held during the swap, but possibly a bit longer if the database is very busy.
- Data changes done by applications during the copy are captured by triggers and written to a "log table"; applied after the initial copying and index rebuild, right before the swap.
- Multiple backends can be launched to rebuild indexes

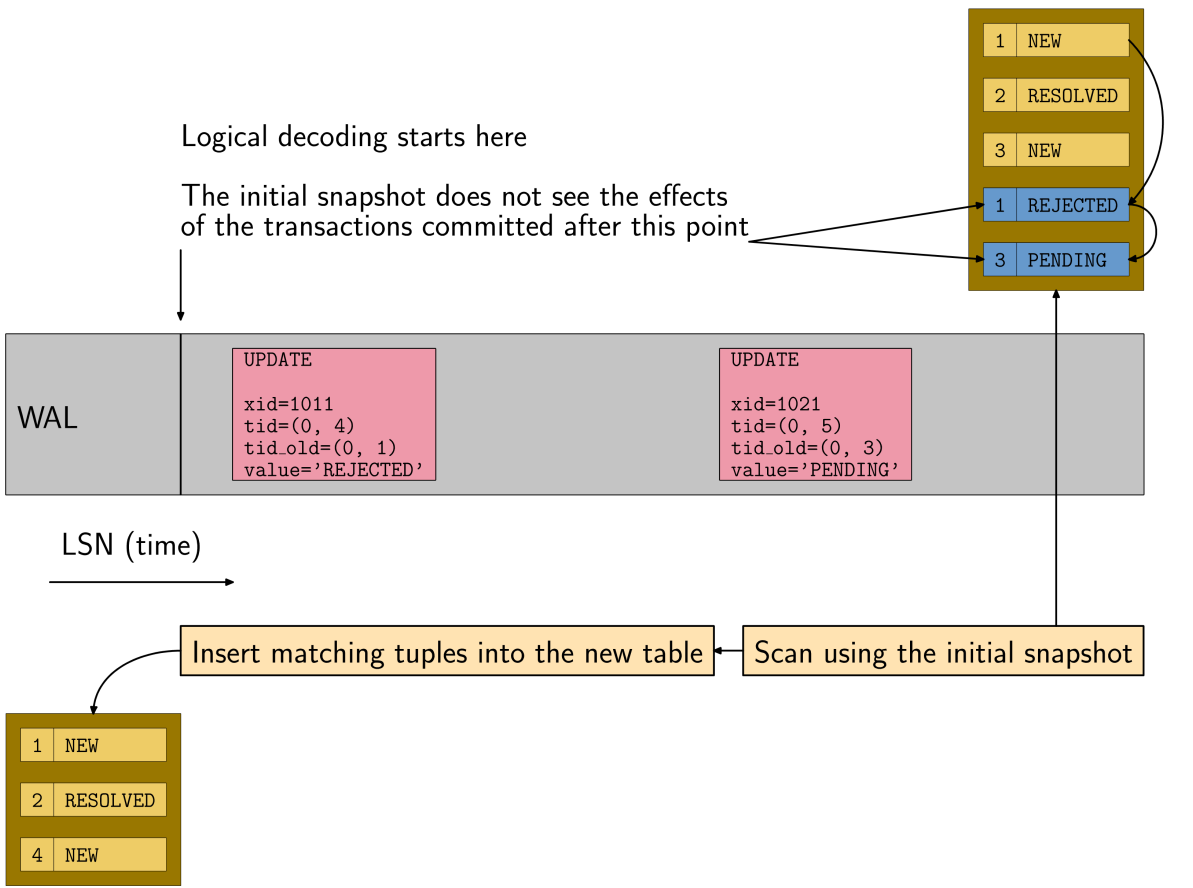
pg_squeeze

- Started in 2016 (PostgreSQL 9.5)
- Initial motivation: allow `pg_repack` to be scheduled without external tools
- Use of dual client/server implementation made this difficult
- Realization: better to reimplement everything with modern technology
  - background worker for scheduling (requires server-only code)
  - logical decoding (instead of triggers)
  - binary data rather than text
  - server API rather than SQL commands
- End result: a complete reimplementation

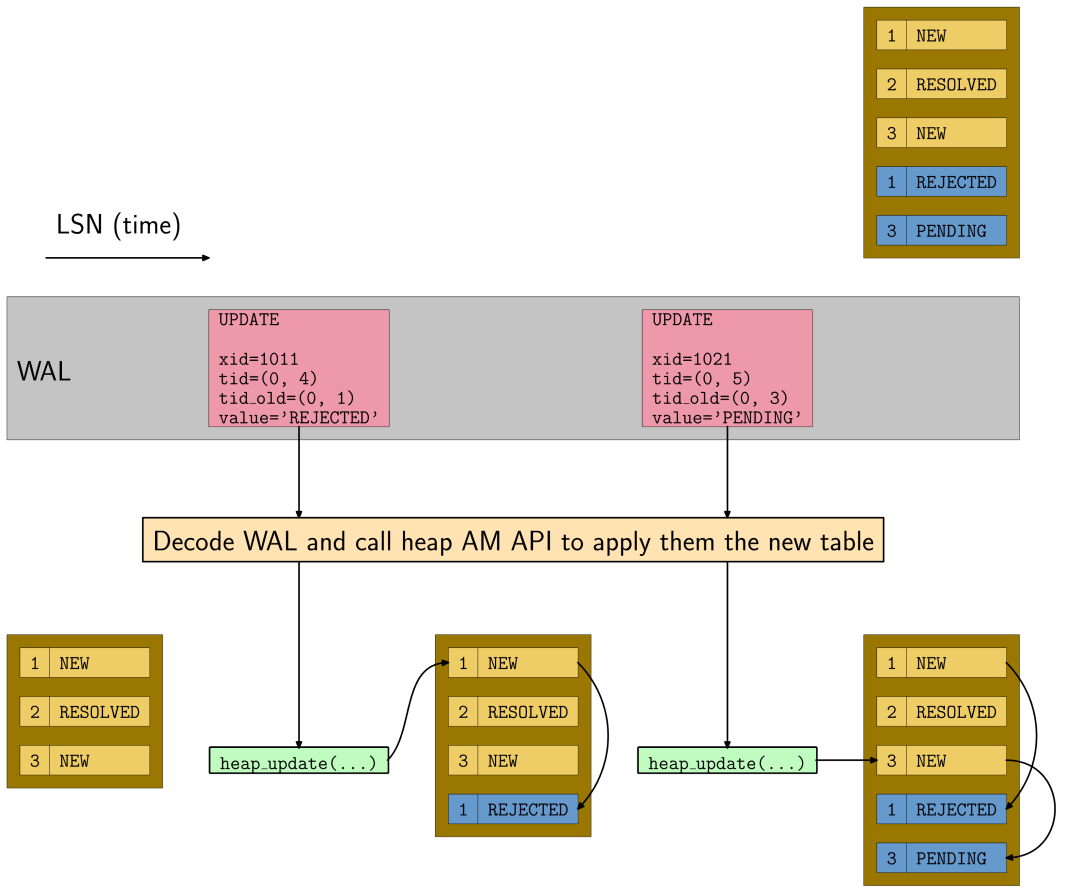
pg_squeeze


1 NEW
2 RESOLVED
3 NEW
UPDATE issues
SET state='REJECTED'
WHERE id=1;
1 NEW
2 RESOLVED
3 NEW
1 REJECTED
UPDATE issues
SET state='PENDING'
WHERE id=3;
1 NEW
2 RESOLVED
3 NEW
1 REJECTED
3 PENDING
WAL
UPDATE
xid=1011
tid=(0, 4)
tid_old=(0, 1)
value='REJECTED'
UPDATE
xid=1021
tid=(0, 5)
tid_old=(0, 3)
value='PENDING'
LSN (time)

pg_squeeze


1 NEW
2 RESOLVED
3 NEW
1 REJECTED
3 PENDING
Logical decoding starts here
The initial snapshot does not see the effects of the transactions committed after this point
WAL
UPDATE
xid=1011
tid=(0, 4)
tid_old=(0, 1)
value='REJECTED'
UPDATE
xid=1021
tid=(0, 5)
tid_old=(0, 3)
value='PENDING'
LSN (time)
Insert matching tuples into the new table
Scan using the initial snapshot
1 NEW
2 RESOLVED
4 NEW

pg_squeeze


1 NEW
2 RESOLVED
3 NEW
1 REJECTED
3 PENDING
LSN (time)
WAL
UPDATE
xid=1011
tid=(0, 4)
tid_old=(0, 1)
value='REJECTED'
UPDATE
xid=1021
tid=(0, 5)
tid_old=(0, 3)
value='PENDING'
Decode WAL and call heap AM API to apply them the new table
1 NEW
2 RESOLVED
3 NEW
heap_update(...)
1 NEW
2 RESOLVED
3 NEW
1 REJECTED
heap_update(...)
1 NEW
2 RESOLVED
3 NEW
1 REJECTED
3 PENDING

pg_squeeze

- cron-like scheduling. Squeeze table if the portion of bloat exceeded threshold specified by the DBA.
- Can move tables and indexes to different tablespaces.
- Cannot process *unlogged* tables.

The REPACK command

- REPACK subsumes CLUSTER and VACUUM FULL

Synopsis

```
REPACK [ ( option [, ...] ) ] [ table_and_columns [ USING INDEX [ index_name ] ] ]

where option can be one of:

    VERBOSE [ boolean ]
    ANALYZE [ boolean ]

and table_and_columns is:

    table_name [ ( column_name [, ...] ) ]
```

EDB POSTGRES AI

CYBERTEC

Single-table REPACK forms

- single-table `vacuum full`:

```
REPACK (ANALYZE) customers;
```

- single-table `cluster`:

```
REPACK (ANALYZE, VERBOSE) customers USING INDEX cust_pkey;
```

- single-table `cluster` using the stored index:

```
REPACK customers USING INDEX;
```

Full-database REPACK forms

- whole-database VACUUM FULL:

```
REPACK;
```

- whole-database CLUSTER:

```
REPACK USING INDEX;
```

Concurrent REPACK

```
REPACK (ANALYZE, CONCURRENTLY) customers USING INDEX tenant_idx;

REPACK (CONCURRENTLY) orders;

REPACK (CONCURRENTLY) USING INDEX;
```

- This behaves similar to pg_squeeze
- Differences of note:
  - Does not unlock the table before requesting the exclusive lock.
  - No scheduling – do we need that in core?

Concurrent REPACK (2)

Advantages over `pg_squeeze`:

- Should not restrict VACUUM of other tables (problem of "xmin horizon")
- MVCC safety (hopefully in PG 19)
- Fully integrated in core
  - Very easy to use

pg_repackdb

```
pg_repackdb repacks a PostgreSQL database.

Usage:
  pg_repackdb [OPTION]... [DBNAME]

Options:
  -a, --all                       repack all databases
      --concurrently              use concurrent mode
  -d, --dbname=DBNAME             database to repack
  -e, --echo                      show the commands being sent to the server
      --index[=INDEX]             repack following an index
  -j, --jobs=NUM                  use this many concurrent connections to repack
  -n, --schema=SCHEMA             repack tables in the specified schema(s) only
  -N, --exclude-schema=SCHEMA     do not repack tables in the specified schema(s)
  -q, --quiet                     don't write any messages
  -t, --table='TABLE[(columns)]'  repack specific table(s) only
  -v, --verbose                   write a lot of output
  -V, --version                   output version information, then exit
  -z, --analyze                   update optimizer statistics
  -?, --help                      show this help, then exit
```

Future work

- Allow to enable logical decoding on the fly
- Use a background worker for logical decoding
- Better control over repacking multiple tables
- Allow tables/indexes to move tablespace
- Migrate table to another table AM (zheap, OrioleDB, ...)
- Modify `ALTER TABLE` to rewrite tables using concurrent repack

For Those Listening

- First things first: do you like `REPACK`?
- Test the patch!
  https://commitfest.postgresql.org/patch/5117
- Leave feedback for talk & conference!

Thanks for listening!

Questions?

Antonín Houska, CYBERTEC
ah@cybertec.at
https://cybertec.at/

Álvaro Herrera, EDB
alvherre@kurilemu.de
https://enterprisedb.com/

Leave Feedback!