

# 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*<sup>1</sup> takes care of them afterwards

---

<sup>1</sup> And HOT-pruning.



# Table bloat: other databases

Rollback segment

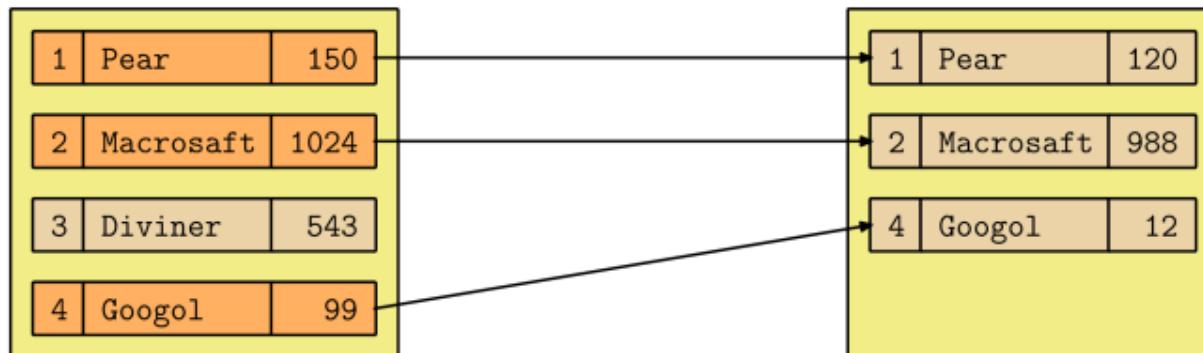
1	Pear	120
2	Macrosaft	988
3	Diviner	543
4	Googol	12



An overwriting storage manager might use a “rollback segment”

# Table bloat: other databases

Rollback segment



As the table is updated, the old 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!

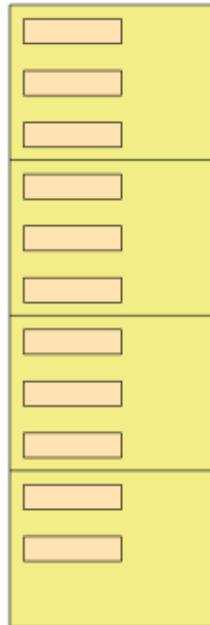


# A rollback segment?

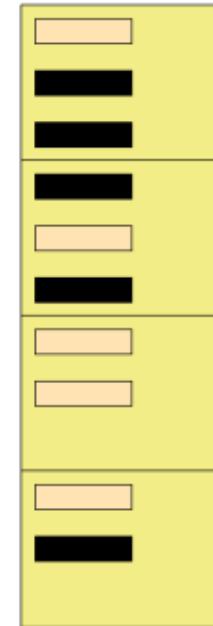
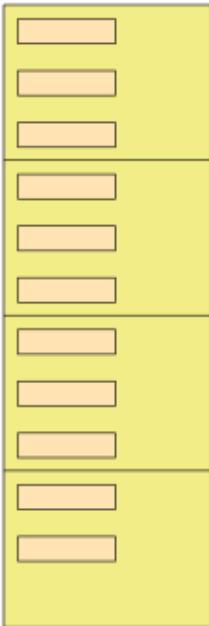
- requires handling of disk space for it
- and later cleanup
- notably: rollbacks are expensive
- and is more difficult to implement
- Postgres tried: see zheap
- Not yet achieved!



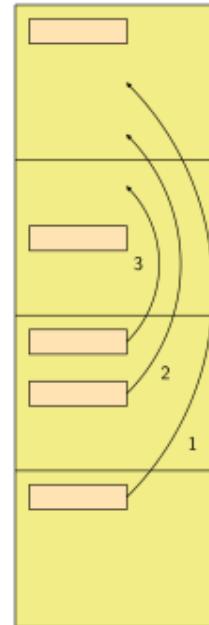
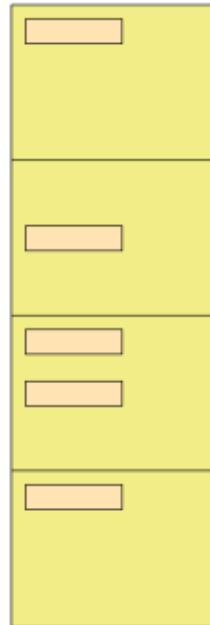
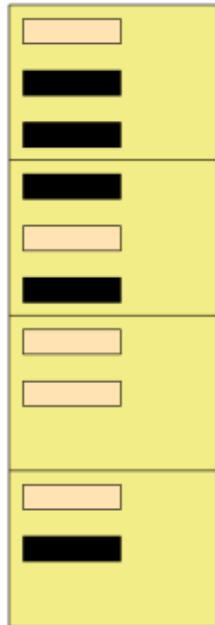
# How 7.1 and earlier handled vacuuming



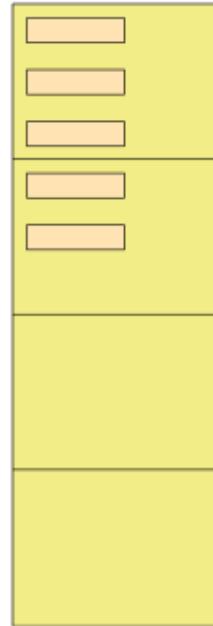
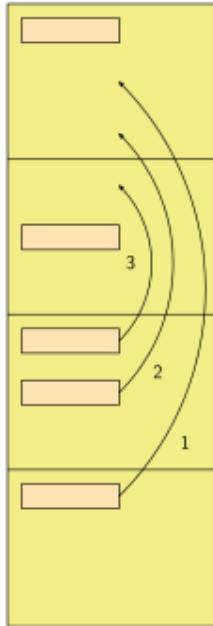
# Dead tuples



# VACUUM phase 1: Dead tuples removed



# VACUUM phase 2: Compact table

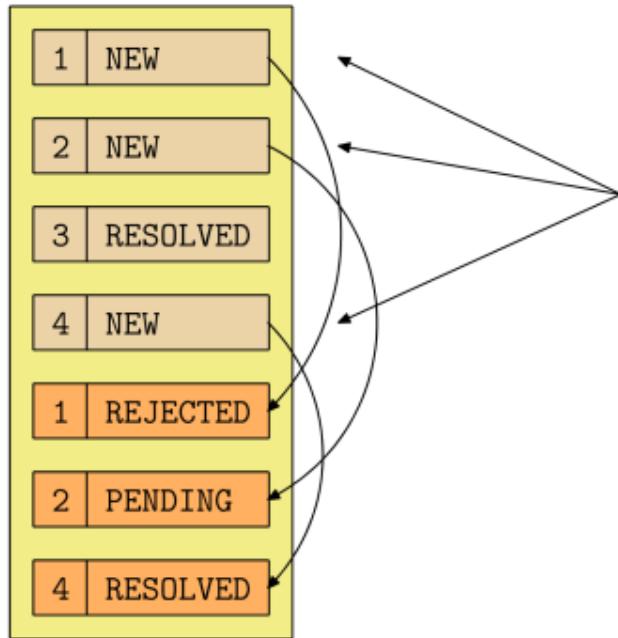


# “Lazy” vacuum?

- 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



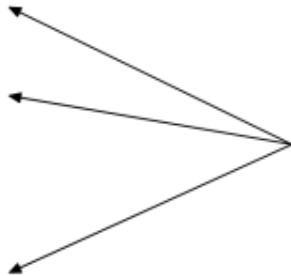
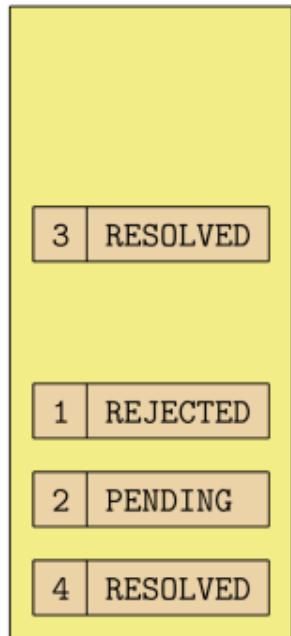
# The problem: table bloat



The old tuple versions will be removed by VACUUM



# The problem: table bloat



Postgres does not always manage  
to reuse the free space

This technique is called “lazy vacuum”

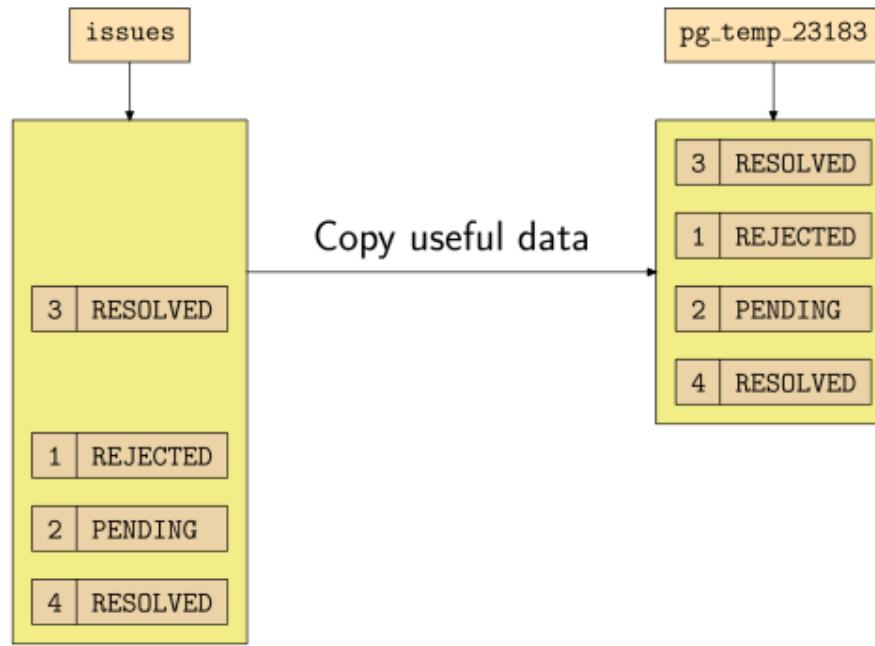


# A faster VACUUM FULL?

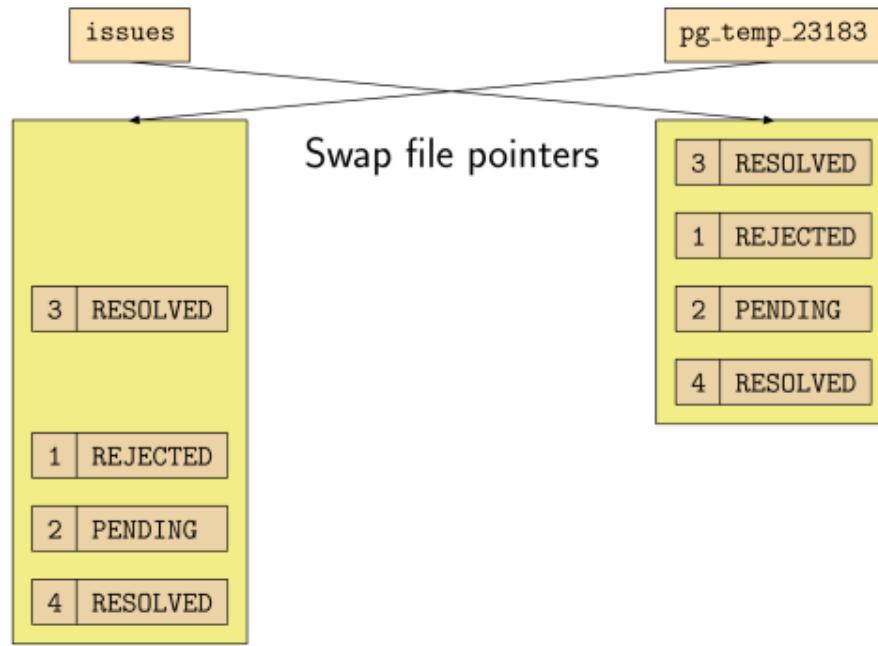
- Yes! “*Let’s use CLUSTER,*” someone said
- Commit 946cf229e89f: ↗  
Support rewritten-based full vacuum as VACUUM FULL. Traditional VACUUM FULL was renamed to VACUUM FULL INPLACE.  
Itagaki Takahiro  
Wed Jan 6 2010, Postgres 9.0
- In 2010 (Postgres 9.0), VACUUM FULL was changed to use the CLUSTER code
- How does this work?



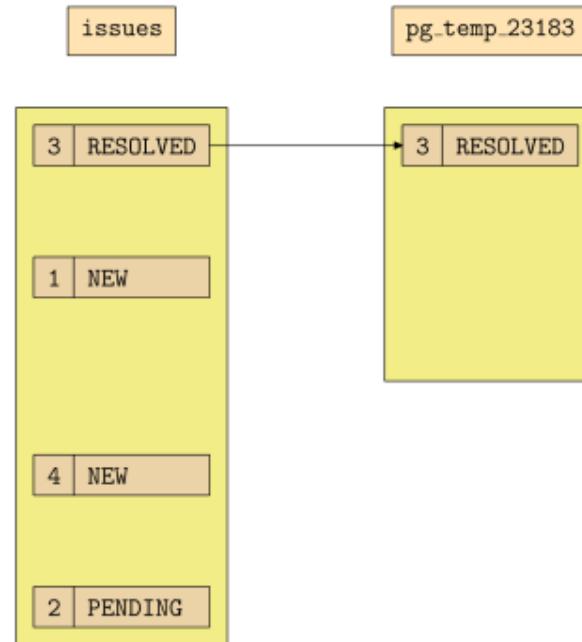
# CLUSTER-based VACUUM FULL



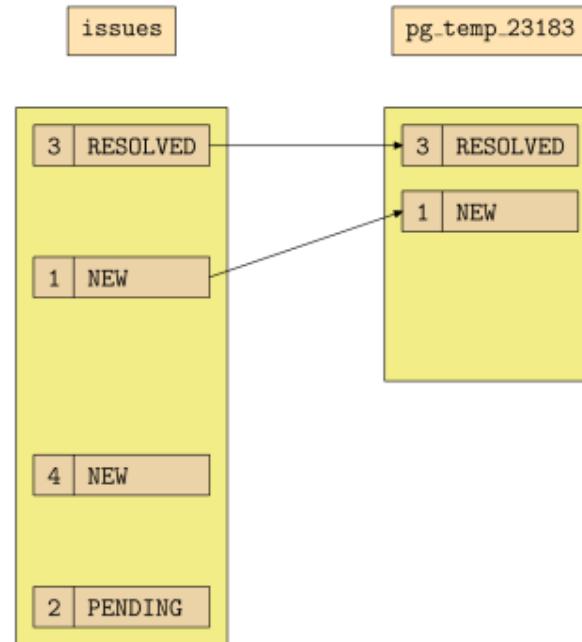
# CLUSTER-based VACUUM FULL



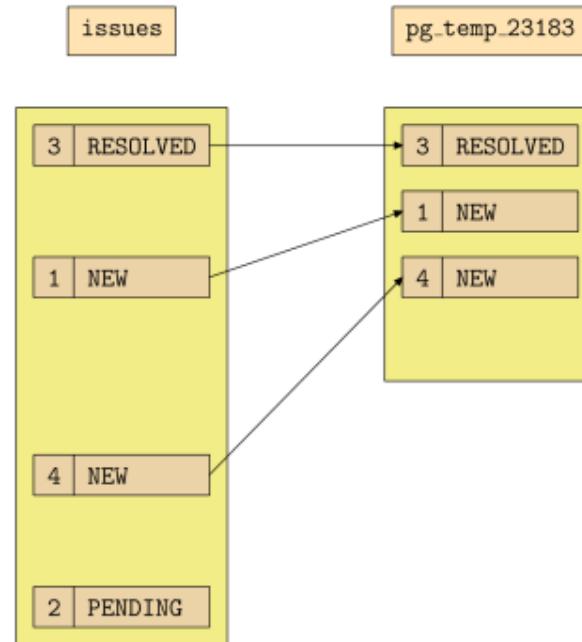
# Exclusive lock is needed



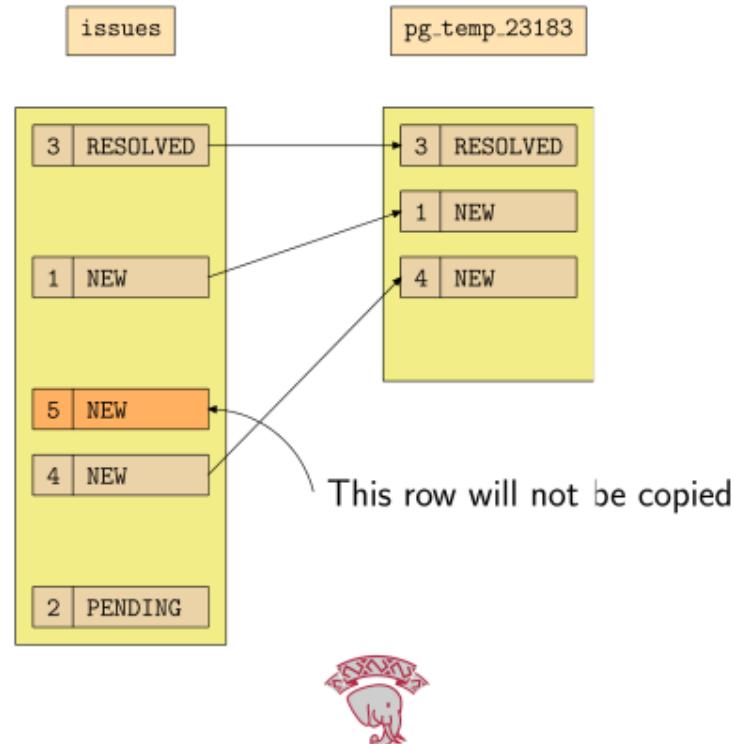
# Exclusive lock is needed



# Exclusive lock is needed



# Exclusive lock is needed



# 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

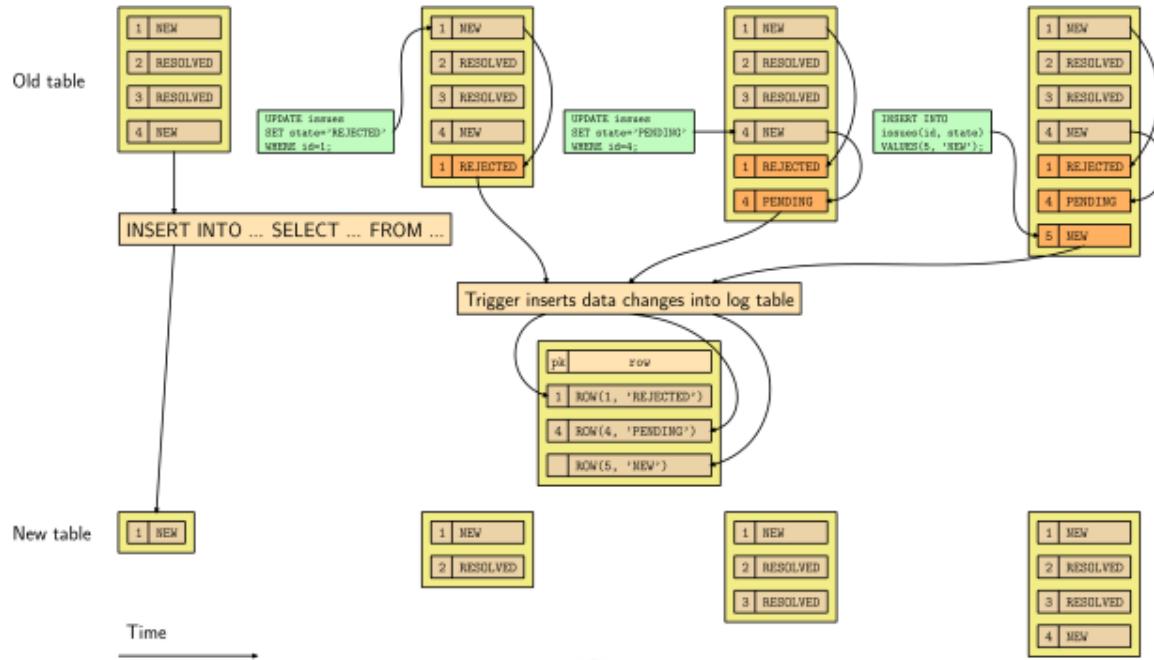
- Created in 2008 by NTT
- [https://ossc-db.github.io/pg\\_reorg/](https://ossc-db.github.io/pg_reorg/)  
*"The module is developed to be a better alternative of CLUSTER and VACUUM FULL."*
- Featured in Depesz's blog in 2011: Bloat Happens  
*"All in all – it's a great tool, which does amazing job."*
- Last release was 1.1.9 in 2013
- Pronounced dead in 2020



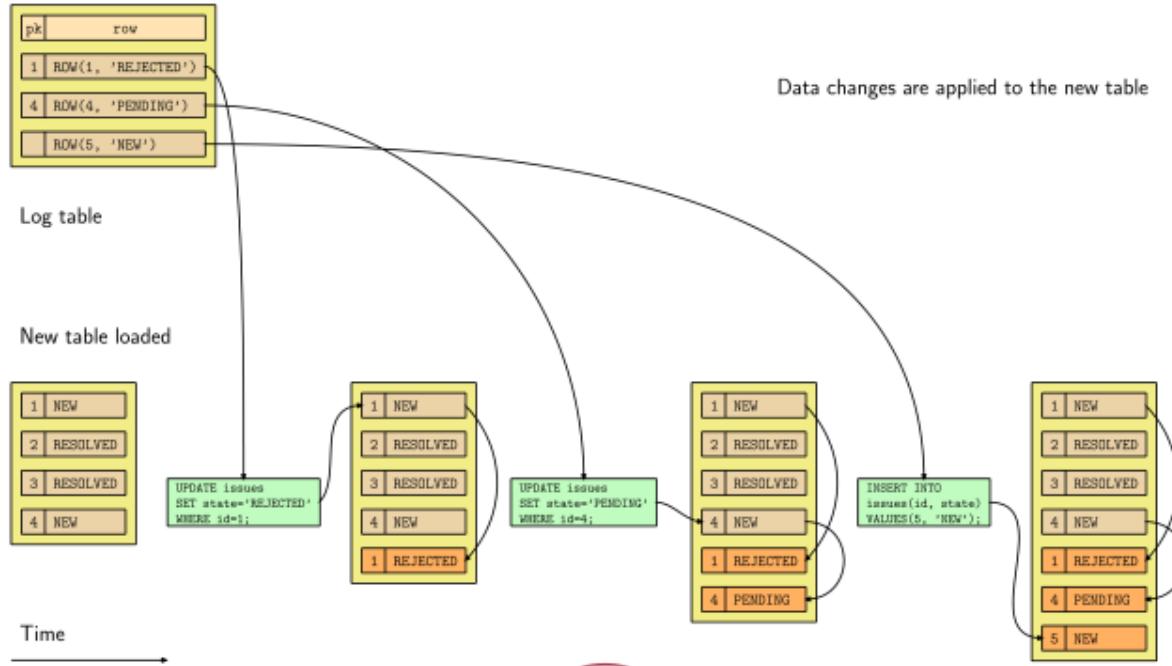
# pg\_repack

- 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



# pg\_repack



## 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

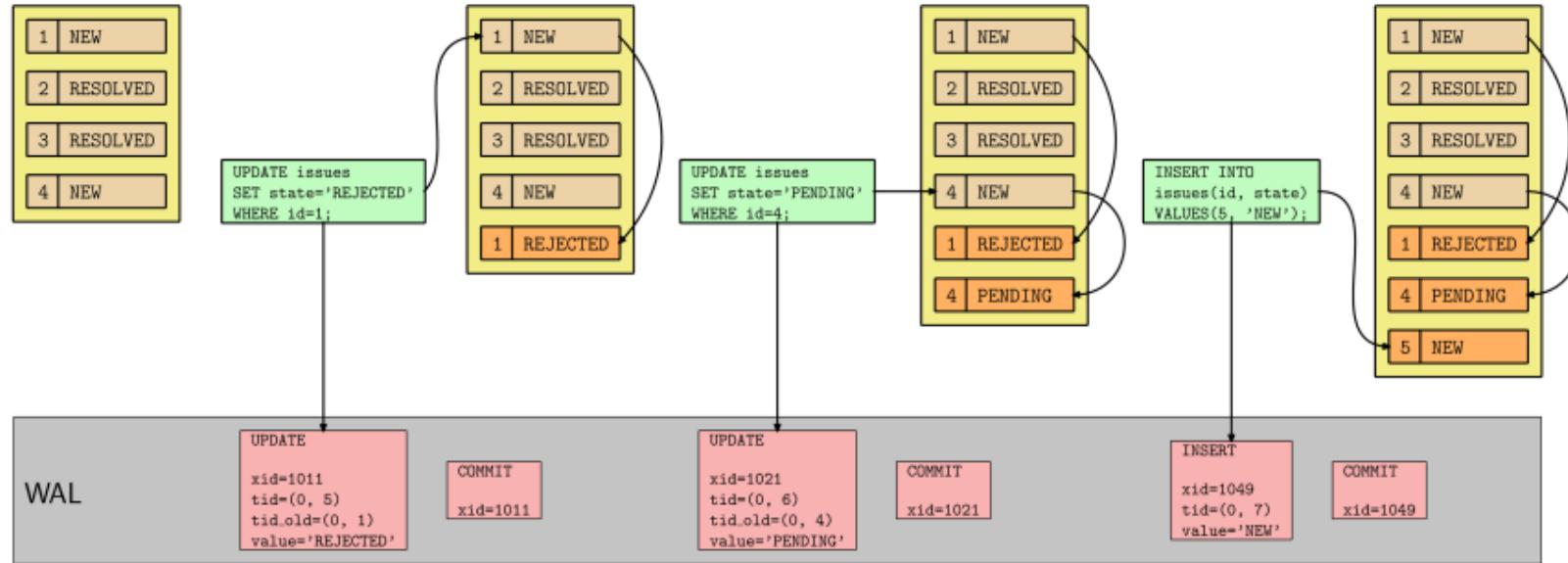


## pg\_squeeze

- 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 workers (instead of a client application)
  - logical decoding (instead of triggers)
- End result: a complete reimplementation



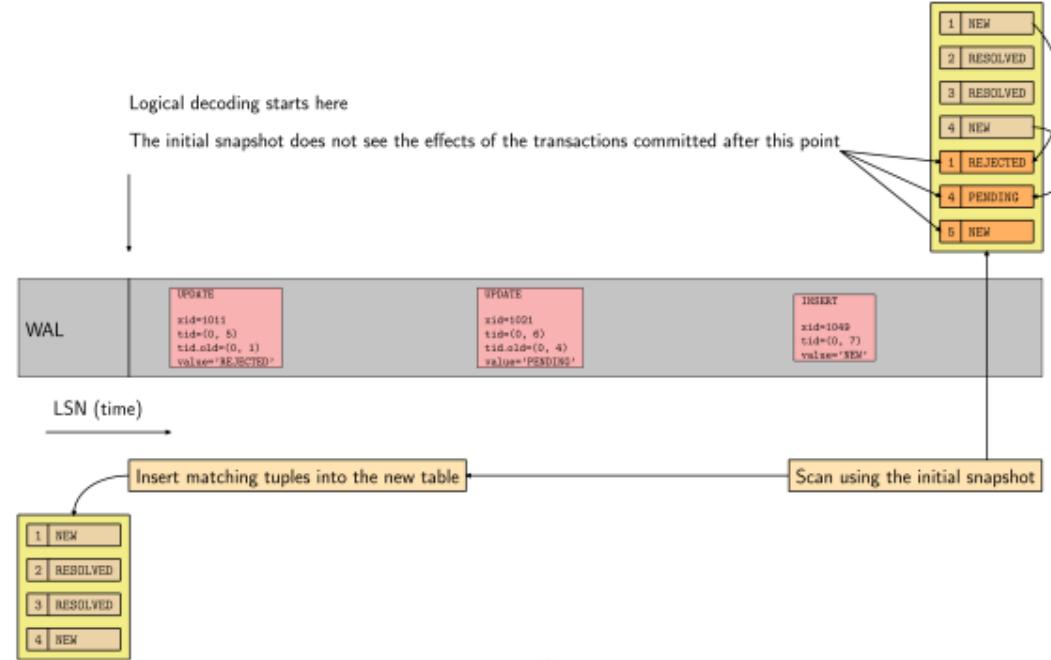
# pg\_squeeze



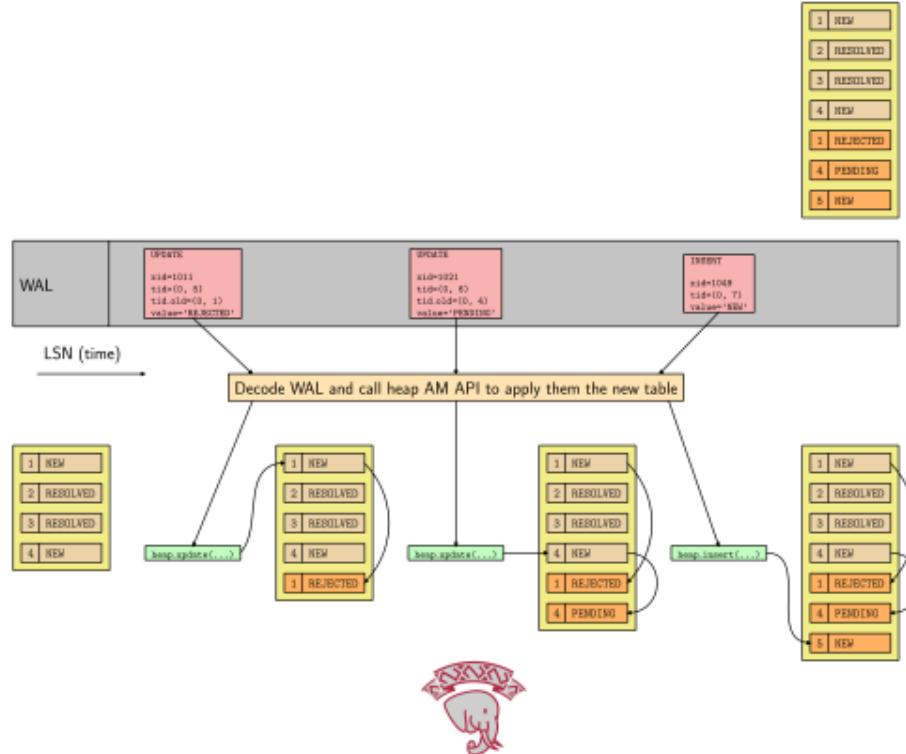
I SN (time)



# pg\_squeeze



# pg\_squeeze



## pg\_squeeze

- Use logical decoding (introduced in PostgreSQL 9.4) rather than triggers.
- Use server API rather than SQL commands to manipulate tables, indexes and data.
- Work with binary data rather than text.
- cron-like scheduling. Squeeze table if the portion of bloat exceeded threshold specified by the DBA.

# 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 [, ...] ) ]
```



# 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 require wal\_level=logical
- MVCC safety (hopefully in PG 19)
- Should not restrict VACUUM of other tables (problem of “xmin horizon”)

# pg\_repackdb

`pg_repackdb` repacks a PostgreSQL database.

Usage:

```
pg_repackdb [OPTION]... [DBNAME]
```

Options:

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

Thanks!

## Questions?

Antonín Houska, CYBERTEC  
[ah@cybertec.at](mailto:ah@cybertec.at)

Álvaro Herrera, EDB  
[alvherre@kurilemu.de](mailto:alvherre@kurilemu.de)



Leave Feedback!

