Solving PostgreSQL wicked problems

Alexander Korotkov

Oriole DB Inc.

2021

PostgreSQL has two sides

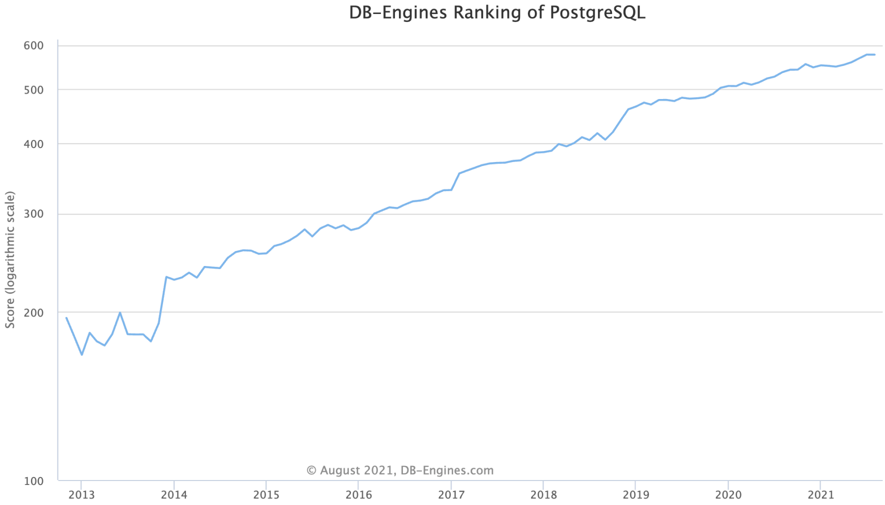
The bright side of PostgreSQL

PostgreSQL – one of the most popular DBMS'es[1]

Rank			DBMS	Score		
Jan 2021	Dec 2020	Jan 2020		Jan 2021	Dec 2020	Jan 2020
1.	1.	1.	Oracle	1322.93	-2.66	-23.75
2.	2.	2.	MySQL	1252.06	-3.40	-22.60
3.	3.	3.	Microsoft SQL Server	1031.23	-6.85	-67.31
4.	4.	4.	PostgreSQL	552.23	+4.65	+45.03
5.	5.	5.	MongoDB	457.22	-0.51	+30.26
6.	6.	6.	IBM Db2	157.17	-3.26	-11.53
7.	7.	↑ 8.	Redis	155.01	+1.38	+6.26

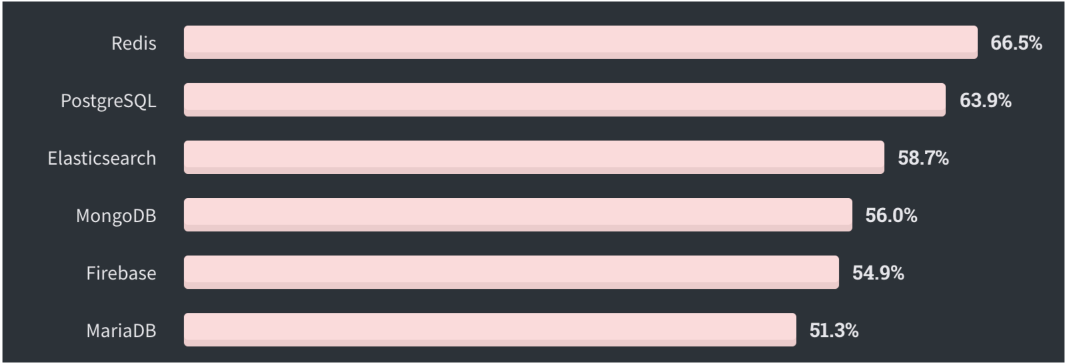
[1] According to db-engines.com

PostgreSQL – strong trend[2]



[2] https://db-engines.com/en/ranking_trend/system/PostgreSQL

PostgreSQL – most loved RDBMS[3]



[3] According to Stackoverflow 2020 survey

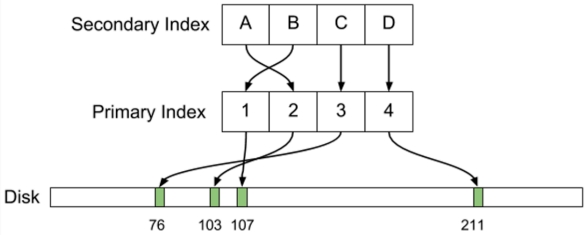
The dark side of PostgreSQL

Criticism of PostgreSQL (1/2)

Why Uber Engineering Switched from Postgres to MySQL

Evan Klitzke

July 26, 2016



https://eng.uber.com/postgres-to-mysql-migration/

Criticism of PostgreSQL (2/2)

10 Things I Hate About PostgreSQL

Rick Branson Follow

Apr 4 · 10 min read

Over the last few years, the software development community's love affair with the popular open-source relational database has reached a bit of a fever pitch. This Hacker News thread covering a piece titled "PostgreSQL is the worlds' best database", busting at the seams with fawning sycophants lavishing unconditional praise, is a perfect example of this phenomenon.

https://medium.com/@rbranson/10-things-i-hate-about-postgresql-20dbab8c2791

10 wicked problems of PostgreSQL

Problem name	Known for	Work started	Resolution
1. Wraparound	20 years	15 years ago	Still WIP
2. Failover Will Probably Lose Data	20 years	16 years ago	Still WIP
3. Inefficient Replication That Spreads Corruption	10 years	8 years ago	Still WIP
4. MVCC Garbage Frequently Painful	20 years	19 years ago	Abandoned
5. Process-Per-Connection = Pain at Scale	20 years	3 years ago	Abandoned
6. Primary Key Index is a Space Hog	13 years	—	Not started
7. Major Version Upgrades Can Require Downtime	21 years	16 years ago	Still WIP
8. Somewhat Cumbersome Replication Setup	10 years	9 years ago	Still WIP
9. Ridiculous No-Planner-Hints Dogma	20 years	11 years ago	Extension
10. No Block Compression	12 years	11 years ago	Still WIP

*** Scalability on modern hardware**

The exciting moment

- PostgreSQL community have proven to be brilliant on solving non-design issues, providing fantastic product to the market.

The exciting moment

- PostgreSQL community have proven to be brilliant on solving non-design issues, providing fantastic product to the market.
- As a result, PostgreSQL has had a **strong upwards trend** for many years.

The exciting moment

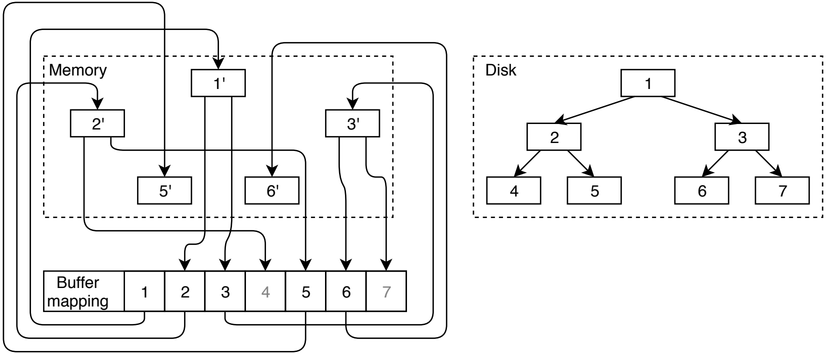
- PostgreSQL community have proven to be brilliant on solving non-design issues, providing fantastic product to the market.
- As a result, PostgreSQL has had a **strong upwards trend** for many years.
- At the same time, the PostgreSQL community appears to be dysfunctional in solving design issues, attracting severe criticism. Nevertheless, critics **not yet** break the upwards trend.

The exciting moment

- PostgreSQL community have proven to be brilliant on solving non-design issues, providing fantastic product to the market.
- As a result, PostgreSQL has had a **strong upwards trend** for many years.
- At the same time, the PostgreSQL community appears to be dysfunctional in solving design issues, attracting severe criticism. Nevertheless, critics **not yet** break the upwards trend.
- It appears to be a **unique moment** for PostgreSQL redesign!

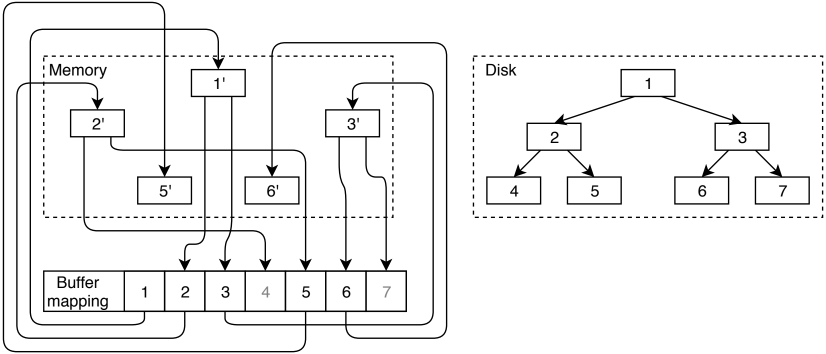
How could we solve the PostgreSQL wicked problems?

Traditional buffer management



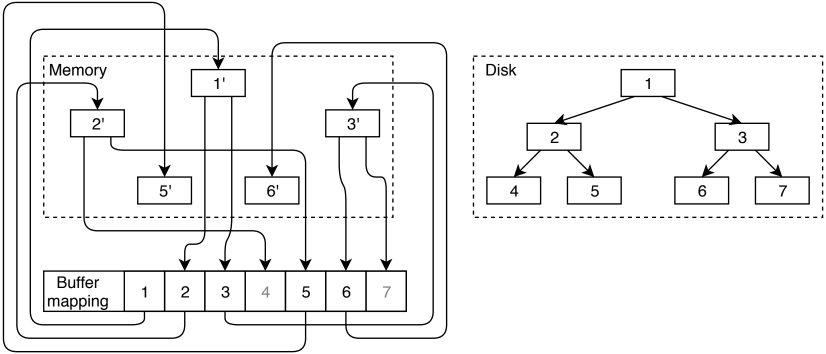
- Each page access requires lookup into buffer mapping data structure.

Traditional buffer management



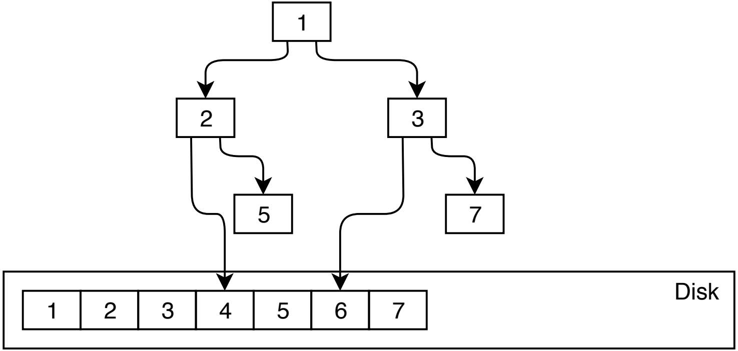
- Each page access requires lookup into buffer mapping data structure.
- Each B-tree key lookup takes multiple buffer mapping lookups.

Traditional buffer management



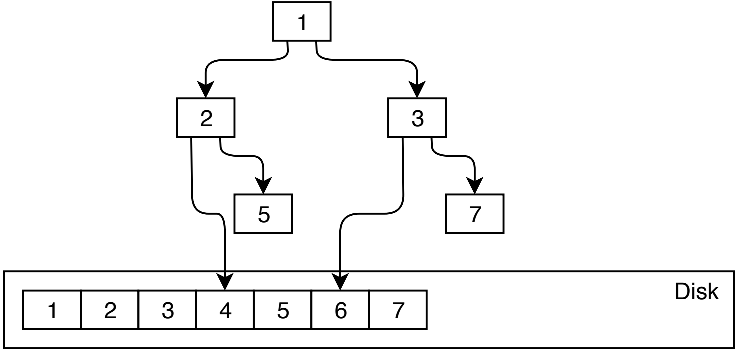
- Each page access requires lookup into buffer mapping data structure.
- Each B-tree key lookup takes multiple buffer mapping lookups.
- Accessing cached data doesn't scale on modern hardware.

Solution: Dual pointers



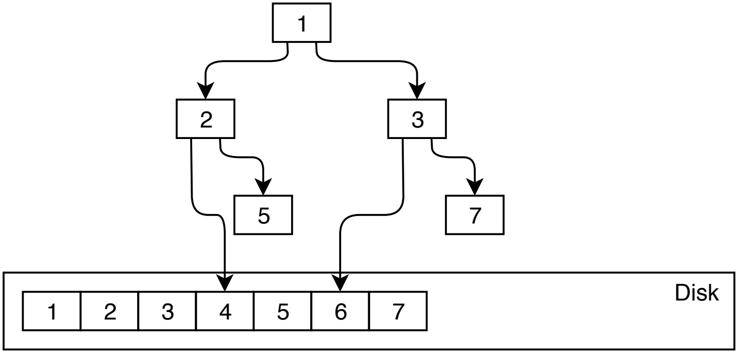
- In-memory page refers either in-memory or on-disk page.

Solution: Dual pointers



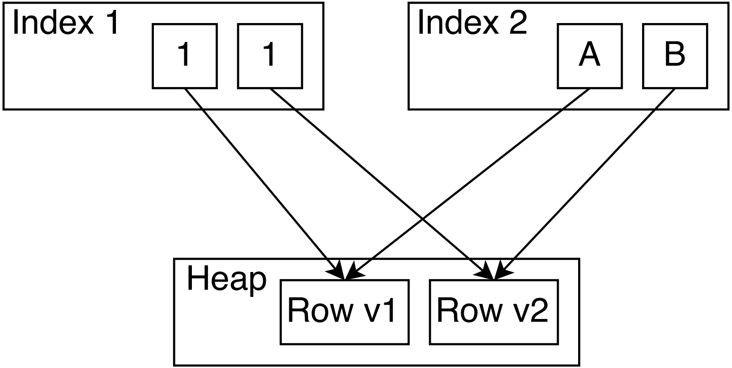
- In-memory page refers either in-memory or on-disk page.
- Accessing cached data without buffer mapping lookups.

Solution: Dual pointers



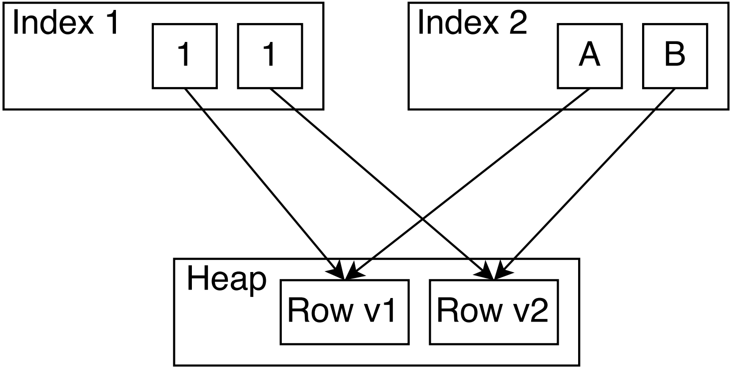
- In-memory page refers either in-memory or on-disk page.
- Accessing cached data without buffer mapping lookups.
- Good scalability!

PostgreSQL MVCC = bloat + write-amplification



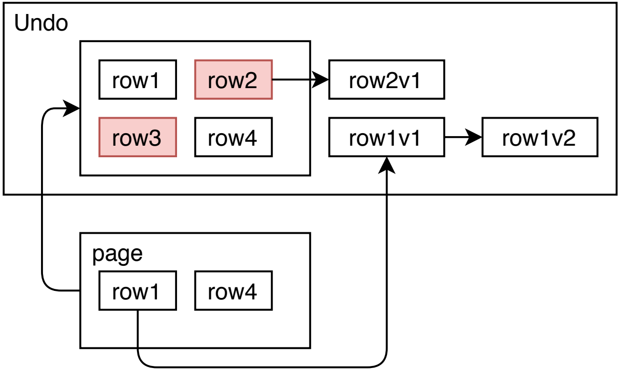
- New and old row versions shares the same heap.

PostgreSQL MVCC = bloat + write-amplification



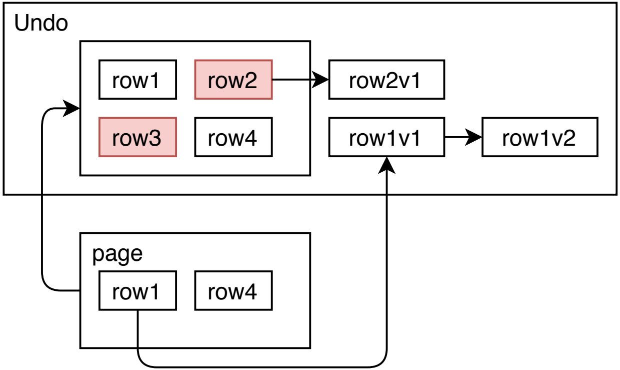
- New and old row versions shares the same heap.
- Non-HOT updates cause index bloat.

Solution: undo log for both pages and rows



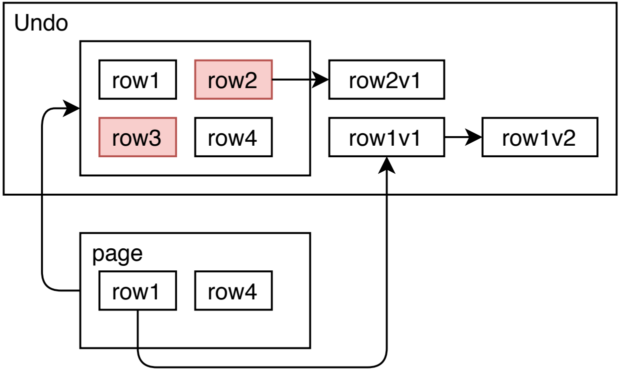
- Old row versions form chains in undo log.

Solution: undo log for both pages and rows



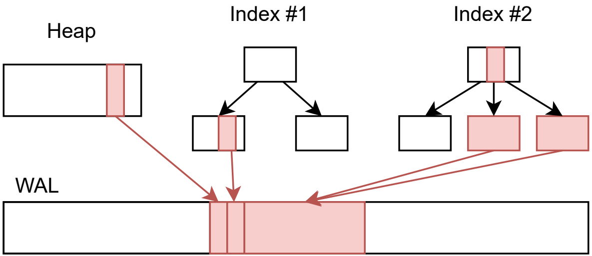
- Old row versions form chains in undo log.
- Page-level chains evict deleted rows from primary storage.

Solution: undo log for both pages and rows



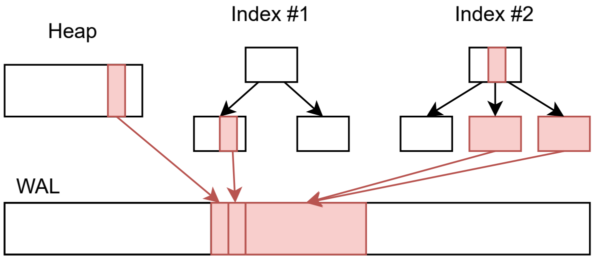
- Old row versions form chains in undo log.
- Page-level chains evict deleted rows from primary storage.
- Update only indexes with changed values.

Block-level WAL



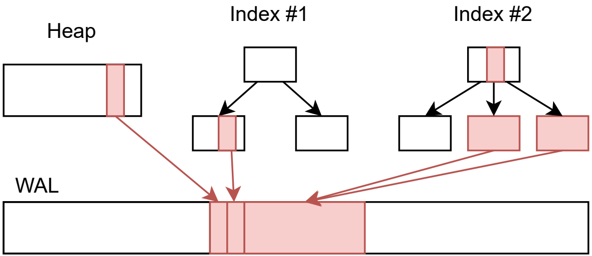
- Huge WAL traffic.

Block-level WAL



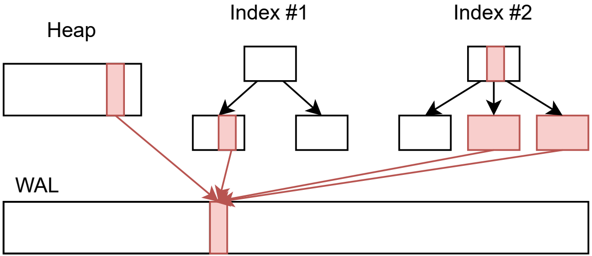
- Huge WAL traffic.
- Problems with parallel apply.

Block-level WAL



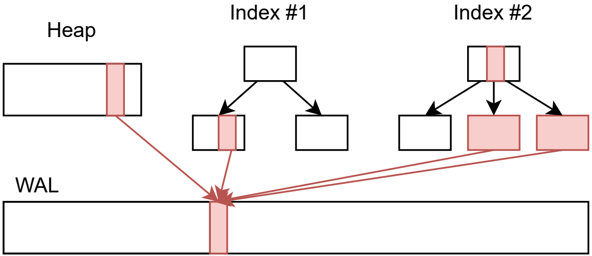
- Huge WAL traffic.
- Problems with parallel apply.
- Not suitable for multi-master replication.

Solution: row-level WAL



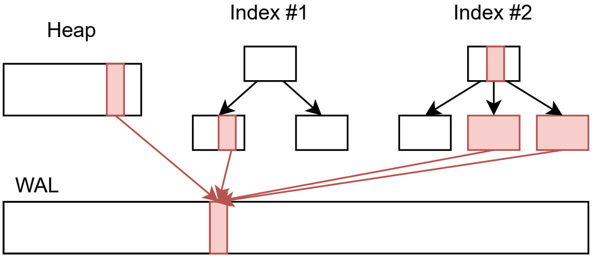
- Very compact.

Solution: row-level WAL



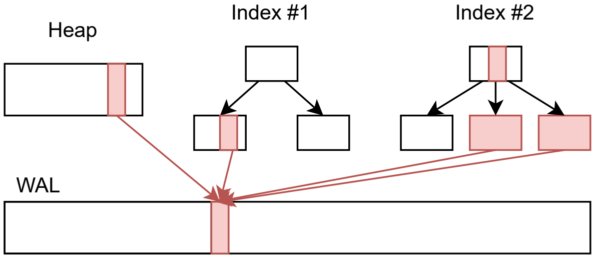
- Very compact.
- Apply can be parallelized.

Solution: row-level WAL



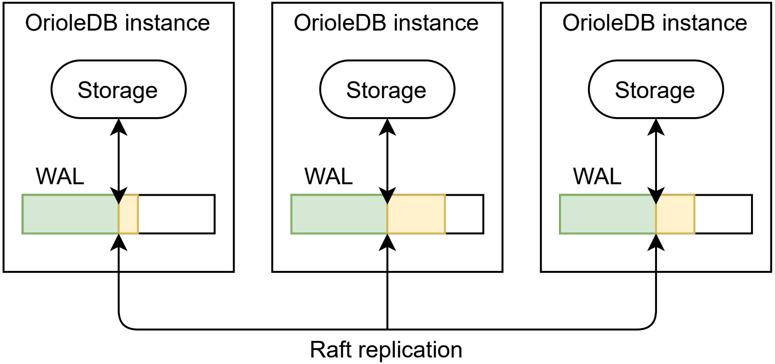
- Very compact.
- Apply can be parallelized.
- Suitable for multimaster (row-level conflicts, not block-level).

Solution: row-level WAL

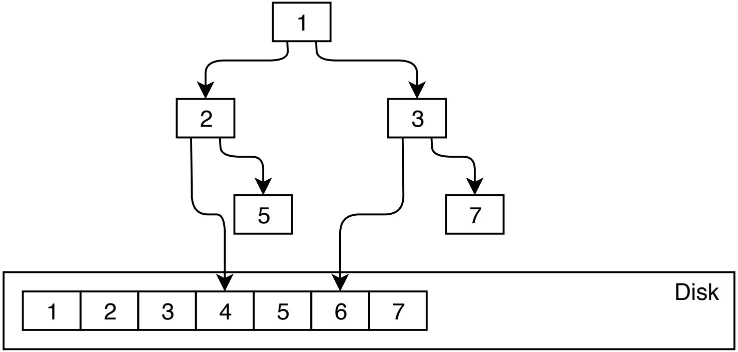


- Very compact.
- Apply can be parallelized.
- Suitable for multimaster (row-level conflicts, not block-level).
- Recovery needs structurally consistent checkpoints.

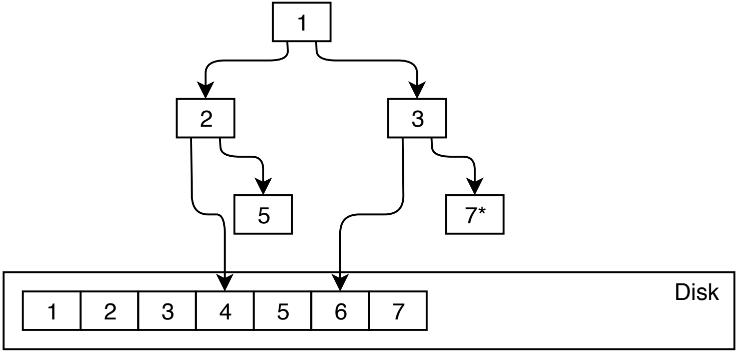
Row-level WAL based multimaster



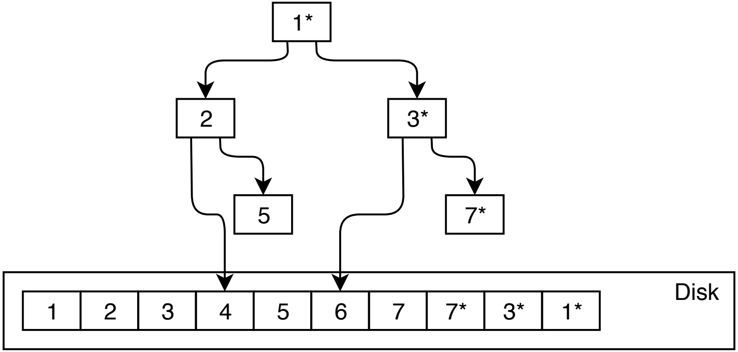
Copy-on-write checkpoints (1/4)



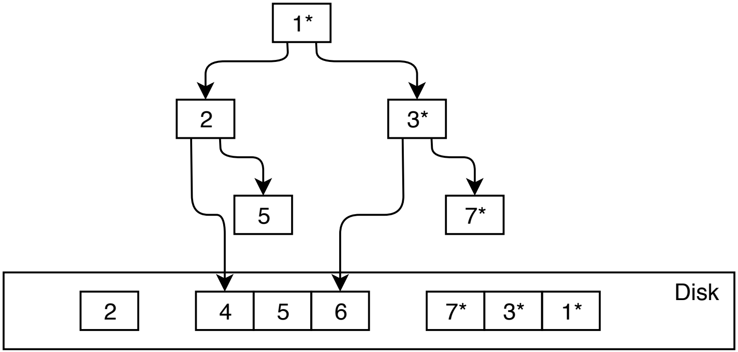
Copy-on-write checkpoints (2/4)



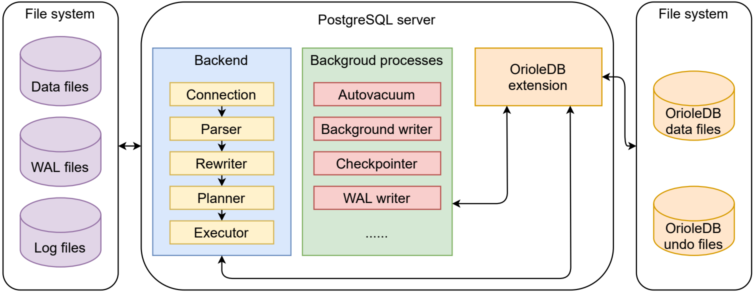
Copy-on-write checkpoints (3/4)



Copy-on-write checkpoints (4/4)

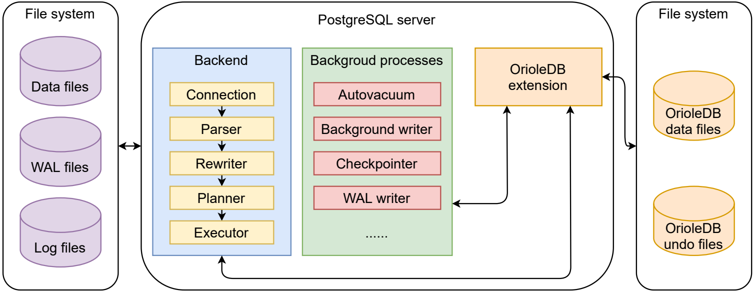


What do we need from PostgreSQL extendability?



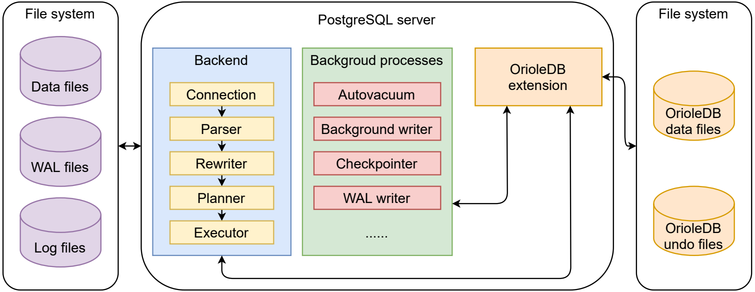
- Extended table AM.

What do we need from PostgreSQL extendability?



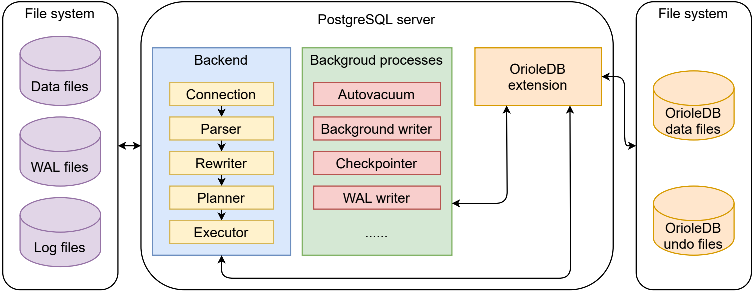
- Extended table AM.
- Custom toast handlers.

What do we need from PostgreSQL extendability?



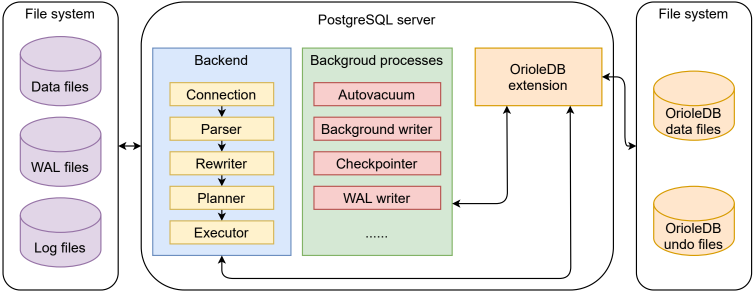
- Extended table AM.
- Custom toast handlers.
- Custom row identifiers.

What do we need from PostgreSQL extendability?



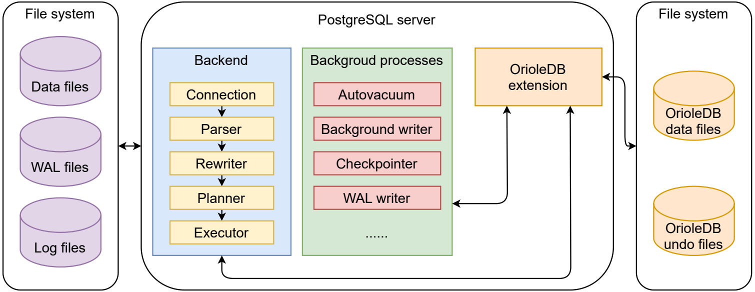
- Extended table AM.
- Custom toast handlers.
- Custom row identifiers.
- Custom error cleanup.

What do we need from PostgreSQL extendability?



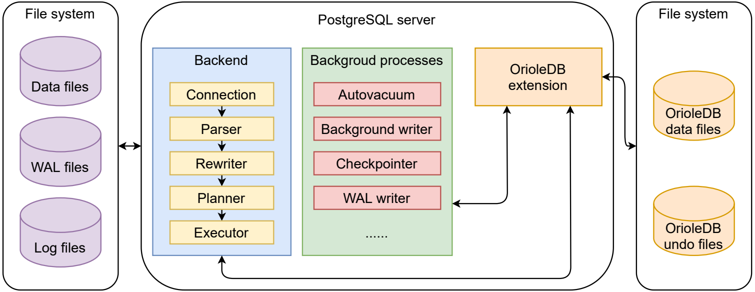
- Extended table AM.
- Custom toast handlers.
- Custom row identifiers.
- Custom error cleanup.
- Recovery & checkpointer hooks.

What do we need from PostgreSQL extendability?



- Extended table AM.
- Custom toast handlers.
- Custom row identifiers.
- Custom error cleanup.
- Recovery & checkpointer hooks.
- Snapshot hooks.

What do we need from PostgreSQL extendability?



- Extended table AM.
- Custom toast handlers.
- Custom row identifiers.
- Custom error cleanup.
- Recovery & checkpointer hooks.
- Snapshot hooks.
- Some other miscellaneous hooks total 1K lines patch to PostgreSQL Core

OrioleDB = PostgreSQL redesign

PostgreSQL	Oriole data base
Block-level WAL	Row-level WAL
Buffer mapping	Direct page links
Buffer locking	Lock-less access
Bloat-prone MVCC	Undo log
Cumbersome block-level WAL replication	Raft-based multimaster replication of row-level WAL

OrioleDB's answer to 10 wicked problems of PostgreSQL

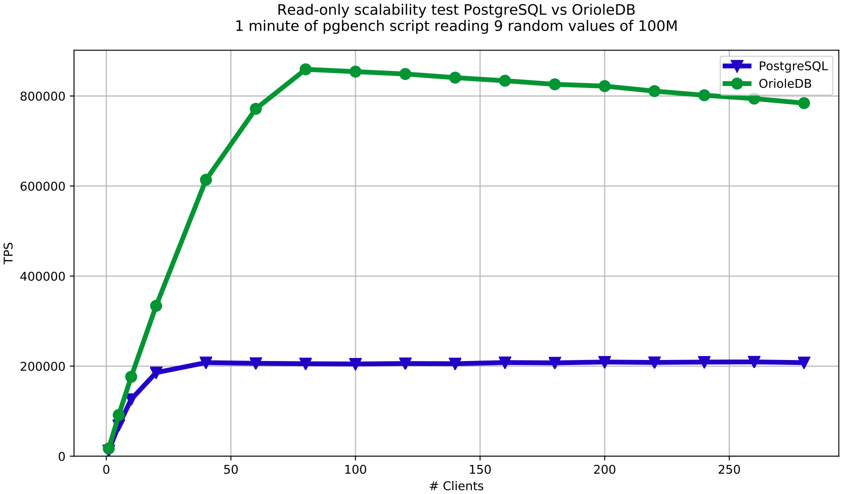
Problem name	Solution
1. Wraparound	**Native 64-bit transaction ids**
2. Failover Will Probably Lose Data	**Multimaster replication**
3. Inefficient Replication That Spreads Corruption	**Row-level replication**
4. MVCC Garbage Frequently Painful	**Non-persistent undo log**
5. Process-Per-Connection = Pain at Scale	**Migration to multithread model**
6. Primary Key Index is a Space Hog	**Index-organized tables**
7. Major Version Upgrades Can Require Downtime	**Multimaster + per-node upgrade**
8. Somewhat Cumbersome Replication Setup	**Simple setup of raft-based multimaster**
9. Ridiculous No-Planner-Hints Dogma	**In-core planner hints**
10. No Block Compression	**Block-level compression**

*** Scalability on modern hardware**

Let’s do some benchmarks! [4]

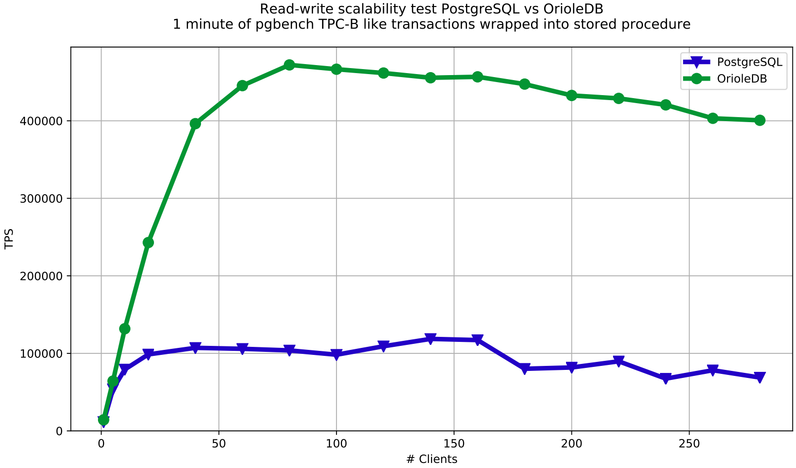
[4] https://gist.github.com/akorotkov/f5e98ba5805c42ee18bf945b30cc3d67

OrioleDB benchmark: read-only scalability



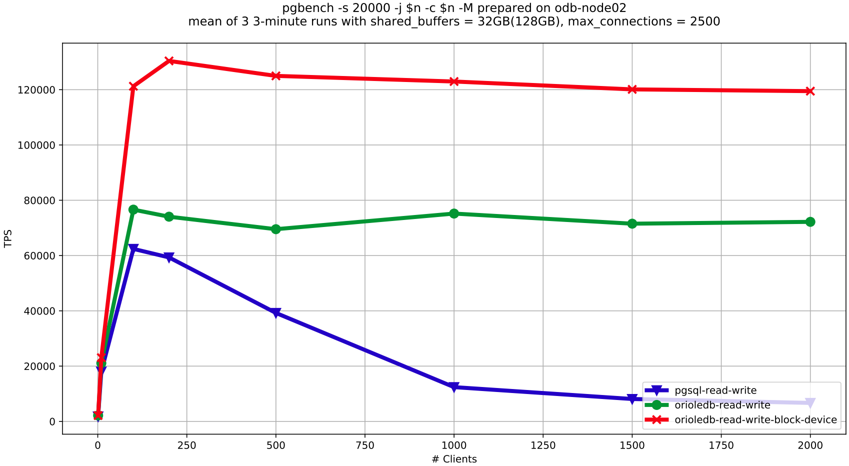
OrioleDB: 4X higher TPS!

OrioleDB benchmark: read-write scalability in-memory case



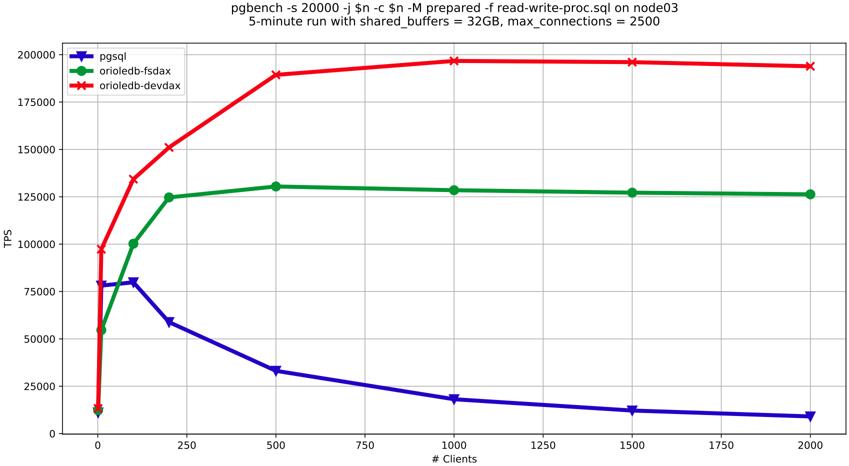
OrioleDB: 3.5X higher TPS!

OrioleDB benchmark: read-write scalability external storage case



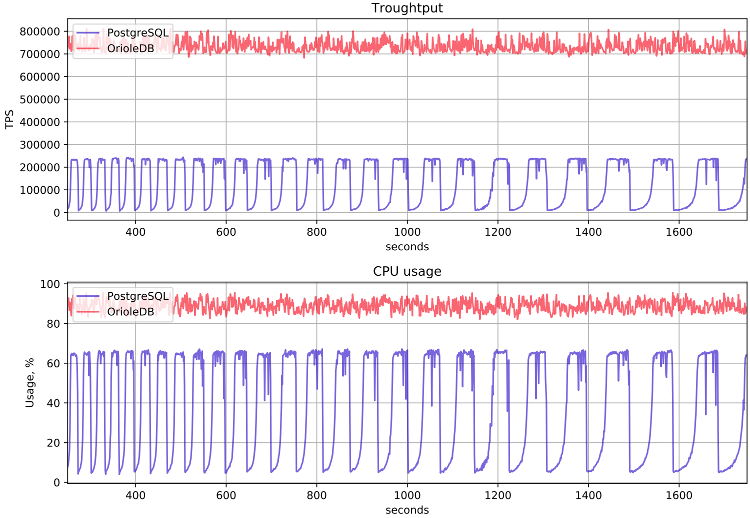
OrioleDB: up to 50X higher TPS!

OrioleDB benchmark: read-write scalability
Intel Optane persistent memory



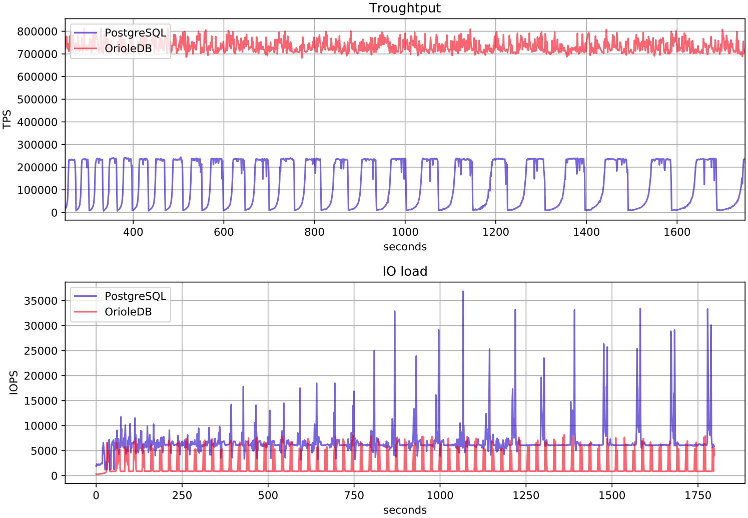
OrioleDB: up to 50X higher TPS!

OrioleDB benchmark: write-amplification & bloat test: CPU



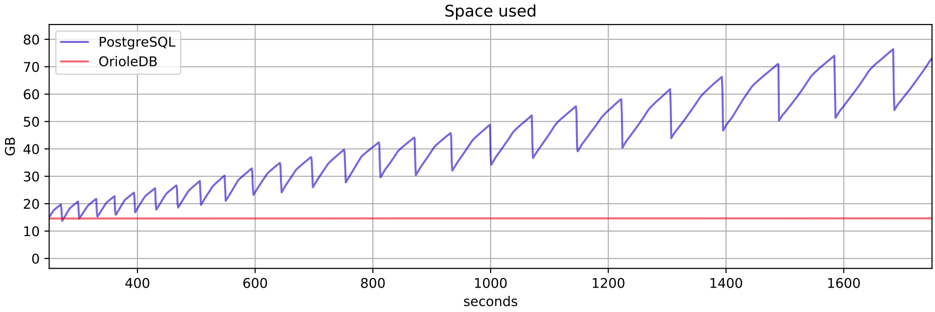
OrioleDB: 5X higher TPS! 2.3X less CPU/TPS!

OrioleDB benchmark: write-amplification & bloat test: IO



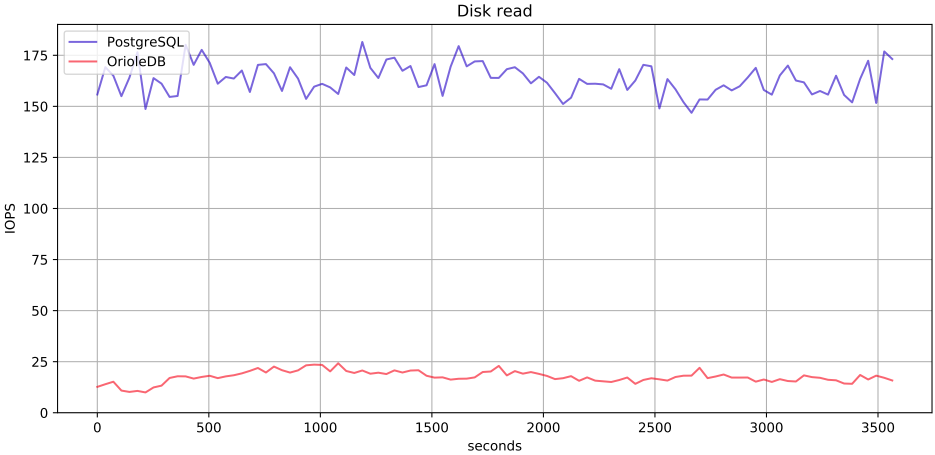
OrioleDB: 5X higher TPS! 22X less IO/TPS!

OrioleDB benchmark: write-amplification & bloat test: space



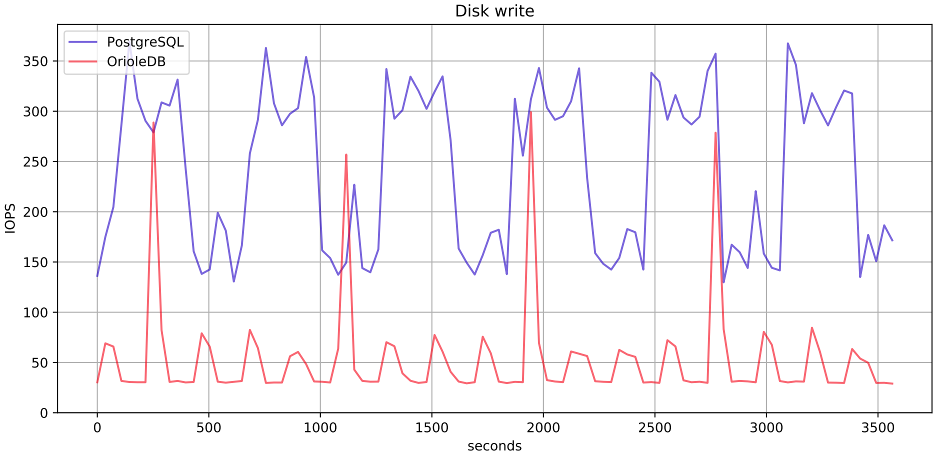
OrioleDB: no bloat!

OrioleDB benchmark: taxi workload (1/3): read



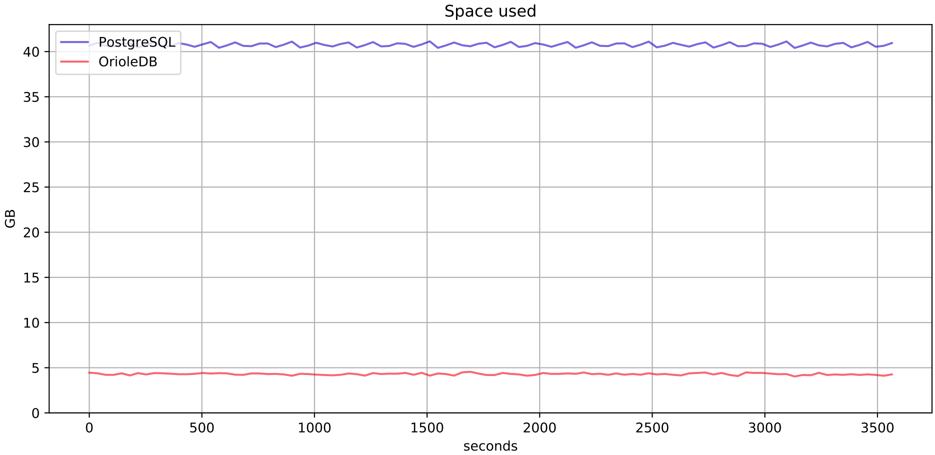
OrioleDB: 9X less read IOPS!

OrioleDB benchmark: taxi workload (2/3): write



OrioleDB: 4.5X less write IOPS!

OrioleDB benchmark: taxi workload (3/3): space



OrioleDB: 8X less space usage!

OrioleDB = Solution of wicked PostgreSQL problems + extraordinary performance

Roadmap

- Basic engine features ✔
- Table AM interface implementation ✔
- Data compression ✔
- Undo log ✔
- TOAST support ✔
- Parallel row-level replication ✔
- Partial and expression indexes ✔

Initial release

- GiST/GIN analogues

OrioleDB status

- Release is scheduled for December 1st 2021;
- `https://github.com/orioledb/orioledb`;
- If you need more explanation, don't hesitate to make pull requests.