

CAN WE SKIP RECOVERY? THE ARCHITECTURE OF ON-DEMAND WAL REPLAY.

Srinath Reddy Sadipiralla
Postgres Hacker @EDB
PGConf.EU 2026

©EDB 2026 — ALL RIGHTS RESERVED.

Feedback



EDB POSTGRES AI

©EDB 2026 — ALL RIGHTS RESERVED.

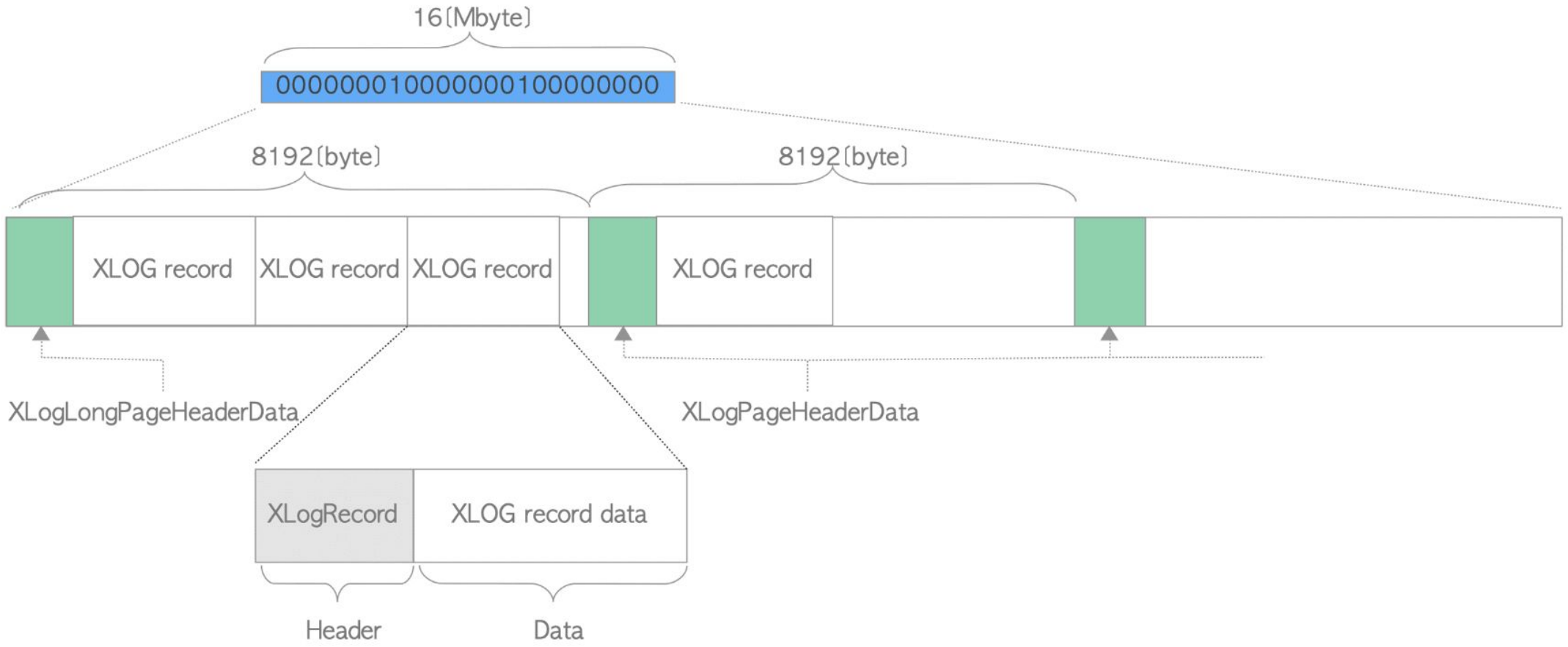
Why I started this project?



EDB POSTGRES AI

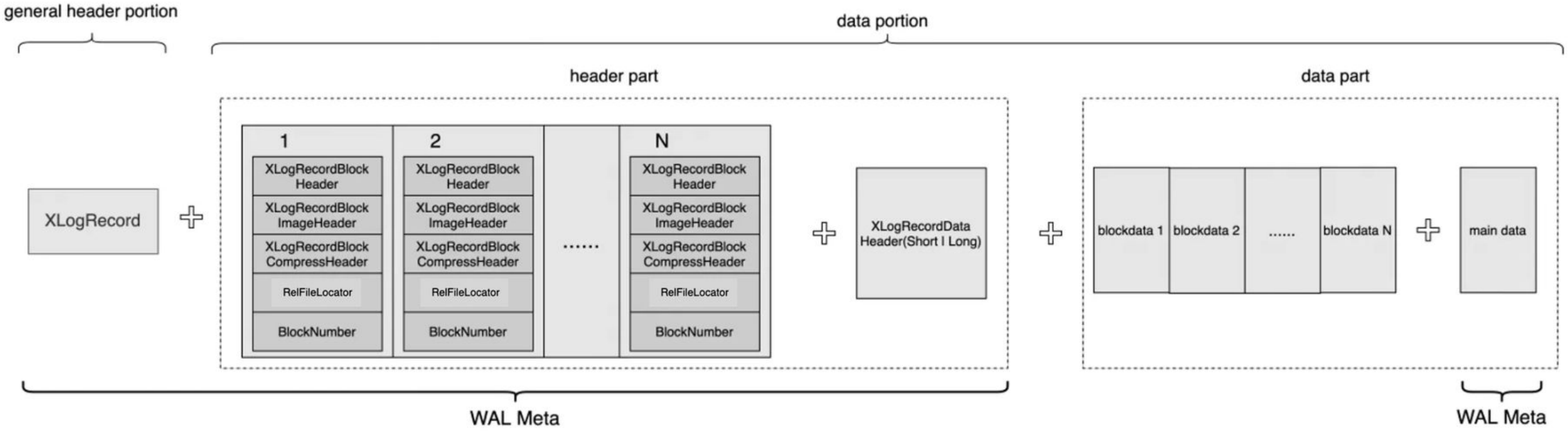
©EDB 2026 — ALL RIGHTS RESERVED.

Internals of WAL (1/2)



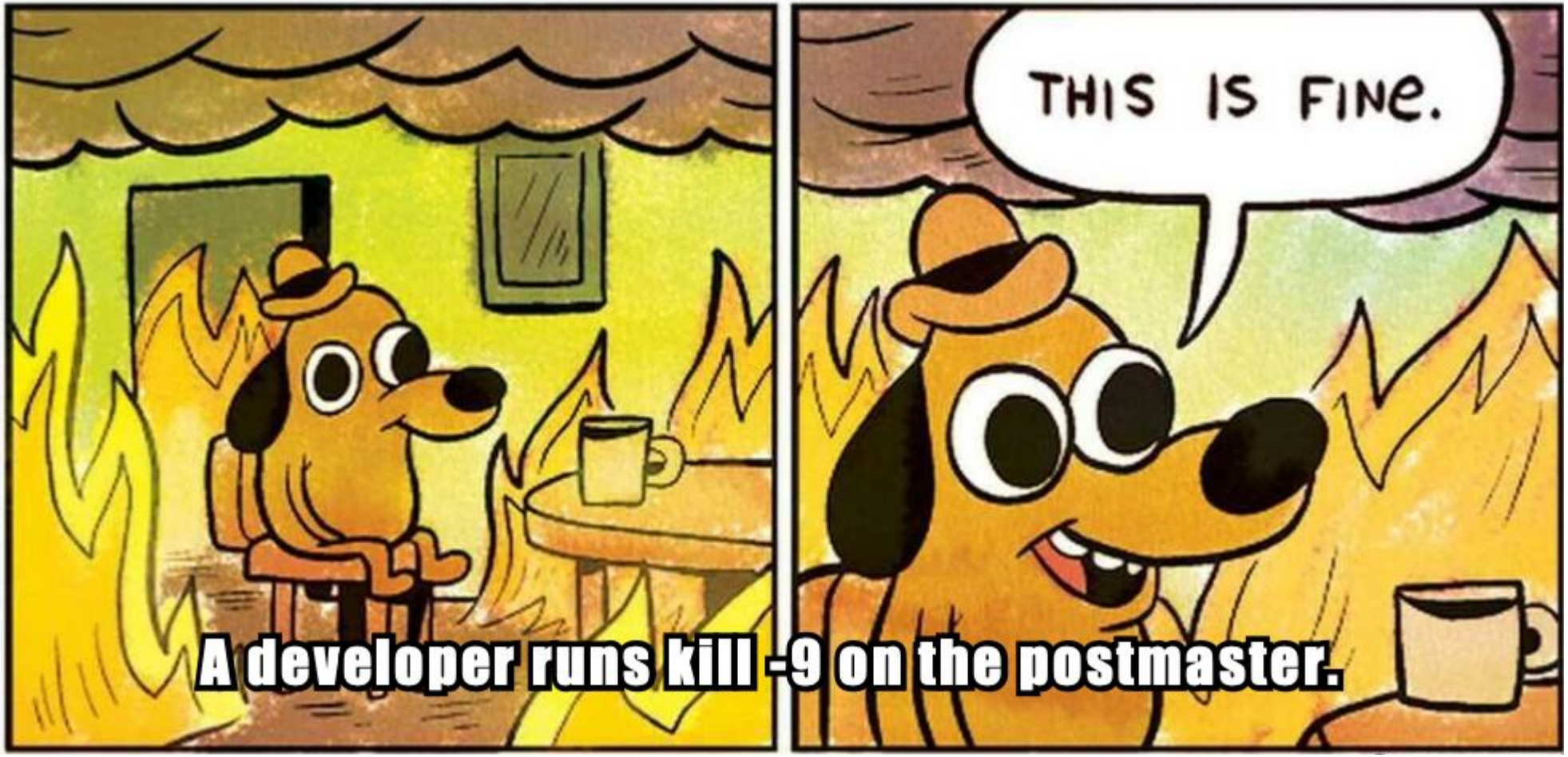
©EDB 2026 — ALL RIGHTS RESERVED.

Internals of WAL (2/2)



©EDB 2026 — ALL RIGHTS RESERVED.

Crash Recovery in Postgres (1/3)

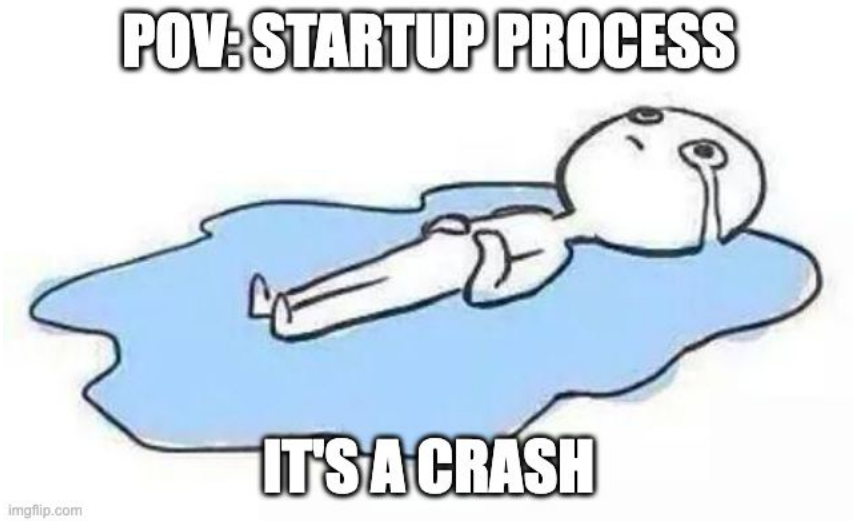


EDB POSTGRES AI

©EDB 2026 — ALL RIGHTS RESERVED.

Crash Recovery in Postgres (2/3)

- When Postgres server starts postmaster which then starts a child process called startup process.
- Startup process reads the **PGDATA/base/pg_control** file and checks the state of the server.
- If **checkPoint.redo != CheckPointLoc** or **state != DB_SHUTDOWNED**



©EDB 2026 — ALL RIGHTS RESERVED.

Crash Recovery in Postgres (3/3)



©EDB 2026 — ALL RIGHTS RESERVED.

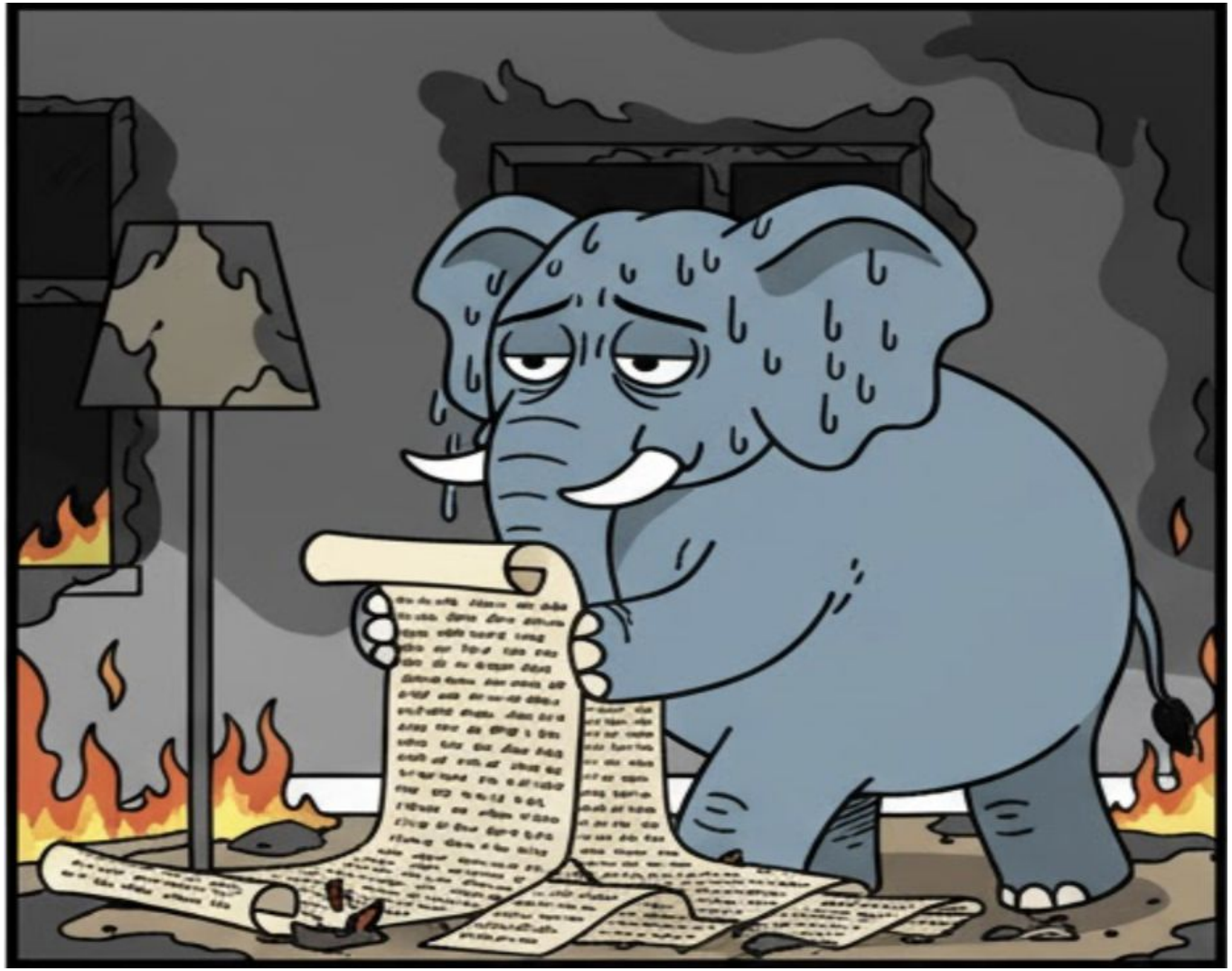
The Problem (1/3)



EDB POSTGRES AI

©EDB 2026 — ALL RIGHTS RESERVED.

The Problem (2/3)



©EDB 2026 — ALL RIGHTS RESERVED.

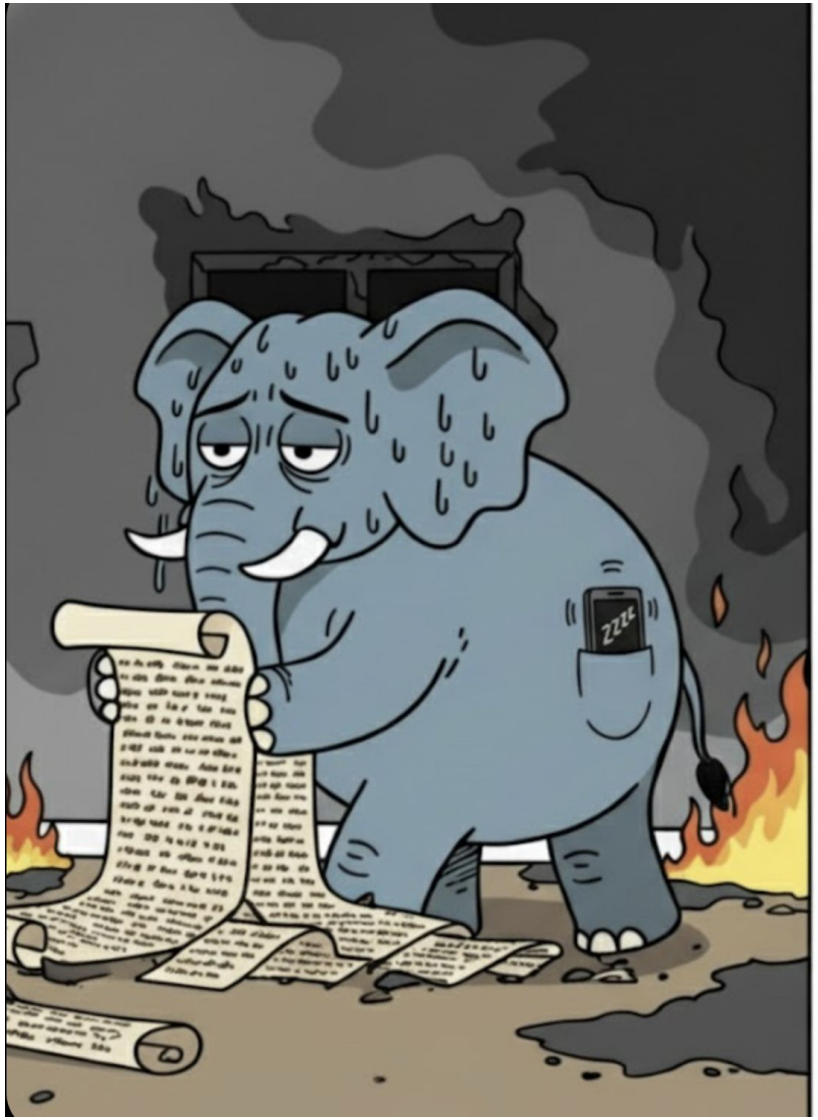
The Problem (3/3)

- Reducing checkpoint interval helps; dirty pages gets flushed, old WALs are removed frequently.
- Less WALs to replay during crash recovery.
- At the expense of making backends executing *queries slow forever until the server is up*;

  needs to read pages from disk more frequently.

©EDB 2026 — ALL RIGHTS RESERVED.

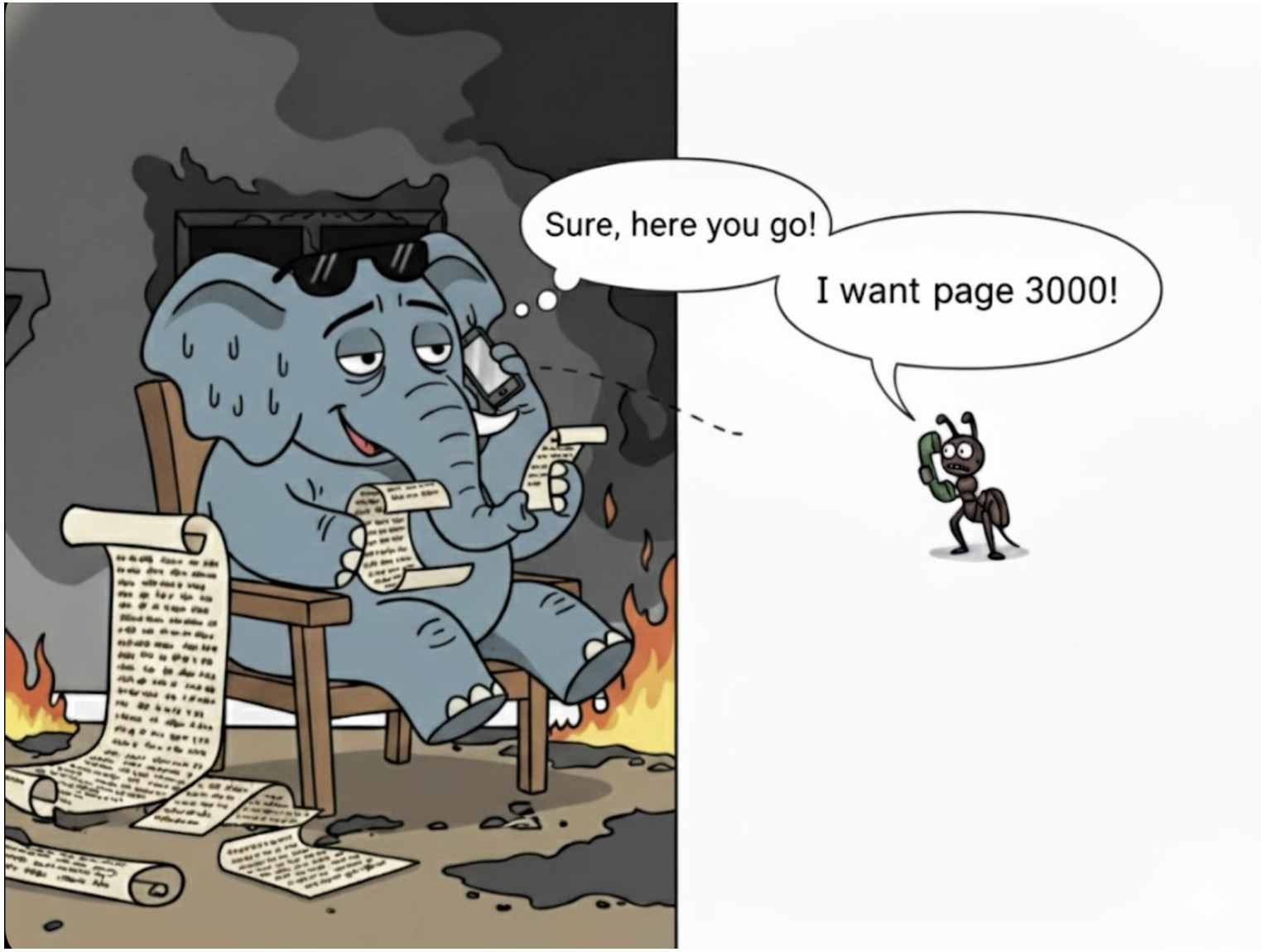
Is it even a problem?





©EDB 2026 — ALL RIGHTS RESERVED.

Solution



©EDB 2026 — ALL RIGHTS RESERVED.

Design



EDB POSTGRES AI

Skipping the WAL?

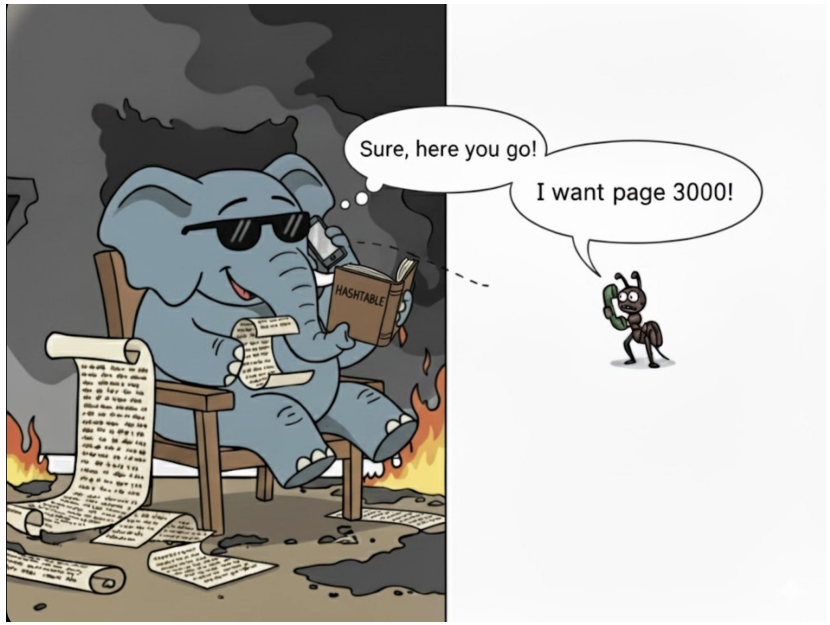
- To make crash recovery fast w/ less server downtime we skip replaying WALs in startup process

  except **RM_XACT_ID** WAL records: keep the transaction status & snapshots consistent.

- Do the replay of WALs when a client asks for a page.

©EDB 2026 — ALL RIGHTS RESERVED.

Isn’t it slow to read, filter the WALs per page request?



- I use a hashtable whose key is page’s Buffertag and value is a list of LSNs of the WAL records which affected this page.
- This hashtable is prepared when the startup process is reading WALs from disk.

©EDB 2026 — ALL RIGHTS RESERVED.

checkpointer and bgwriter processes

- These help plain crash recovery.
- In fast crash recovery, no replay, no dirty pages, no WAL recycle.
- But bgwriter can help after the server is open for connections.

©EDB 2026 — ALL RIGHTS RESERVED.

What if no one reads a page for a long time?

- WAL accumulation and slow queries.
- Fixed these with a new process: does on-demand replay of all pages and dies.
- Helps as a signal to start the checkpointer process.

©EDB 2026 — ALL RIGHTS RESERVED.

What if 2 processes try to read same page at a time?

- On-demand replay is done in between **StartBufferIO** and **TerminateBufferIO**

  when someone is reading a page from disk.

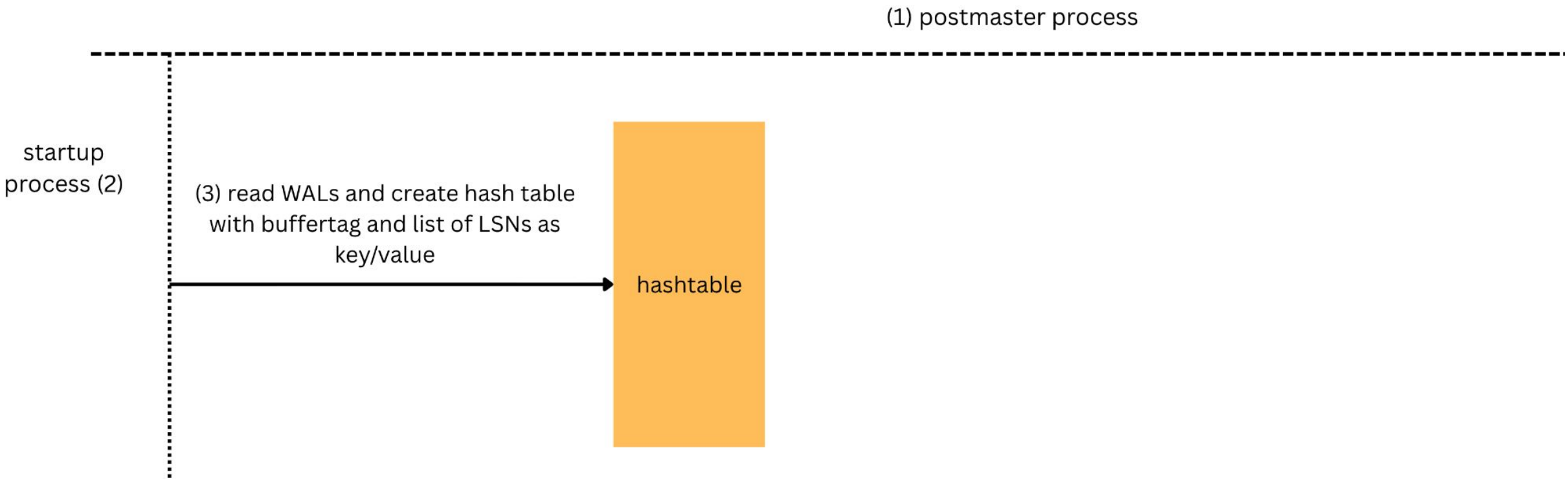
- The buffer state is set as **BM_IO_IN_PROGRESS** until TerminateBufferIO is called.

- If anyone tries to read the same page they will be stuck in **WaitIO** loop until

  BM_IO_IN_PROGRESS is cleared by TerminateBufferIO.

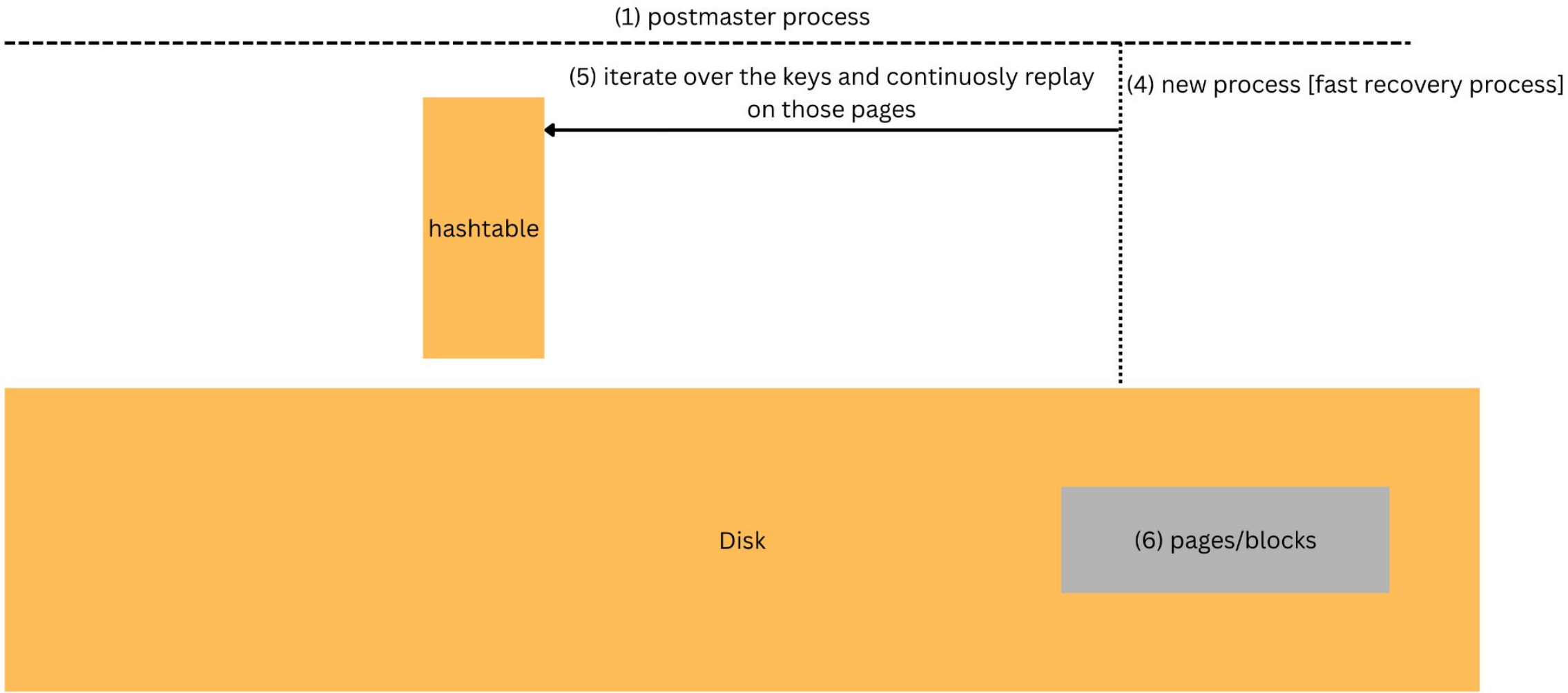
©EDB 2026 — ALL RIGHTS RESERVED.

Summary (1/4)



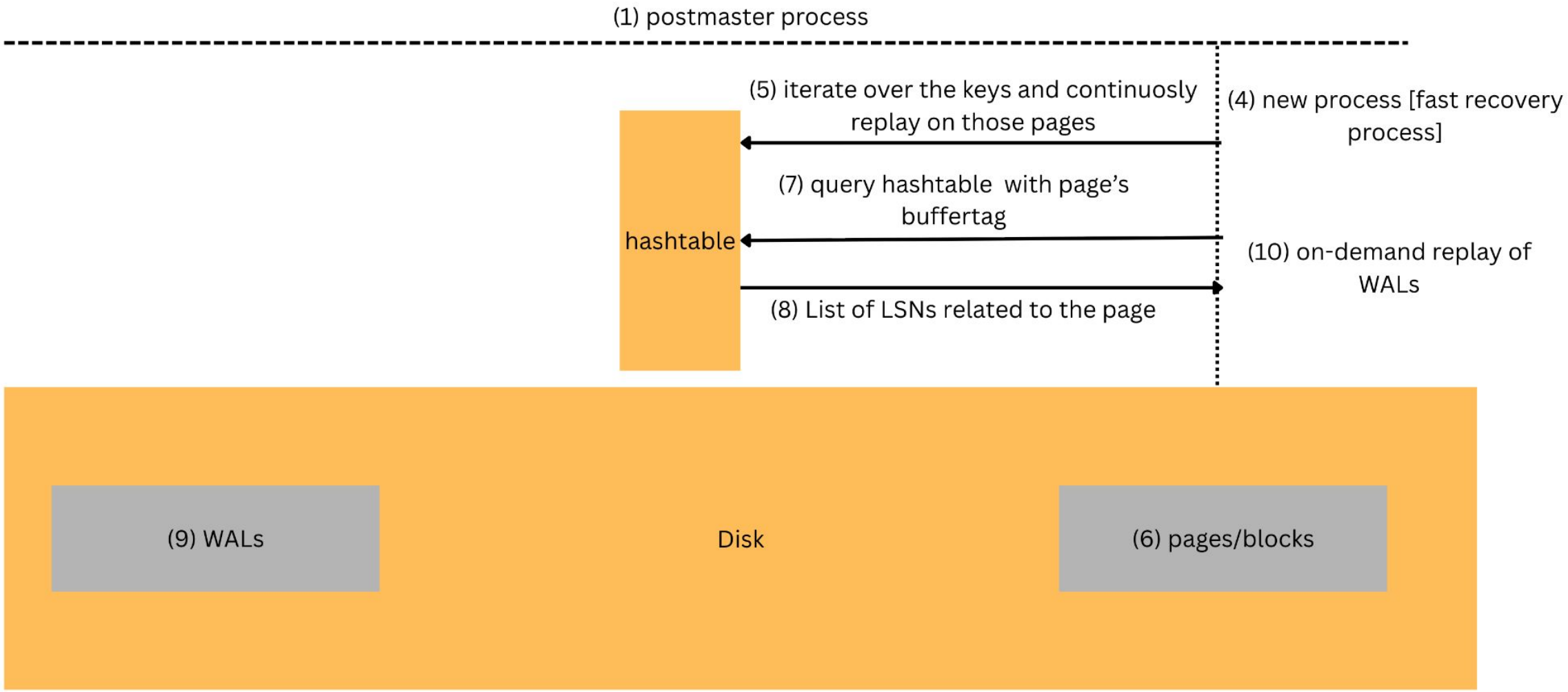
©EDB 2026 — ALL RIGHTS RESERVED.

Summary (2/4)



EDB POSTGRES AI
©EDB 2026 — ALL RIGHTS RESERVED.

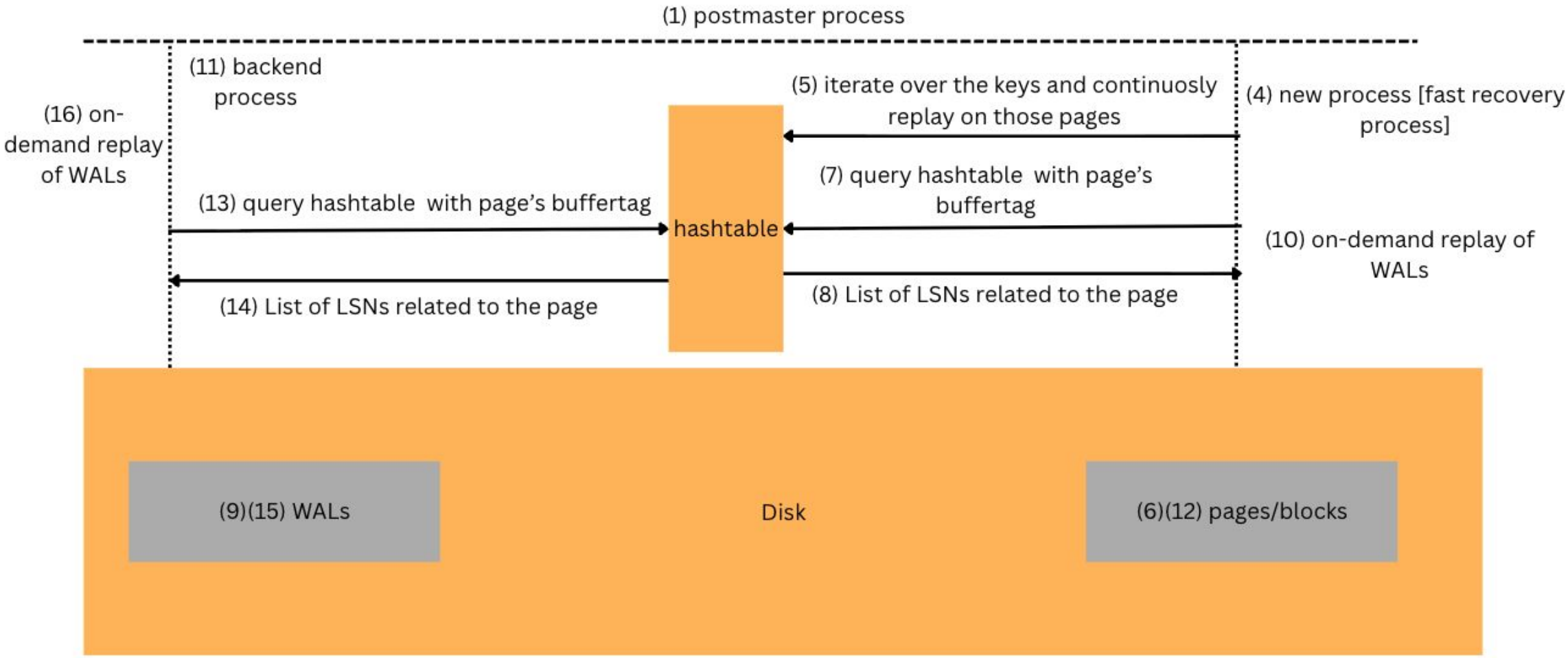
Summary (3/4)



EDB POSTGRES AI

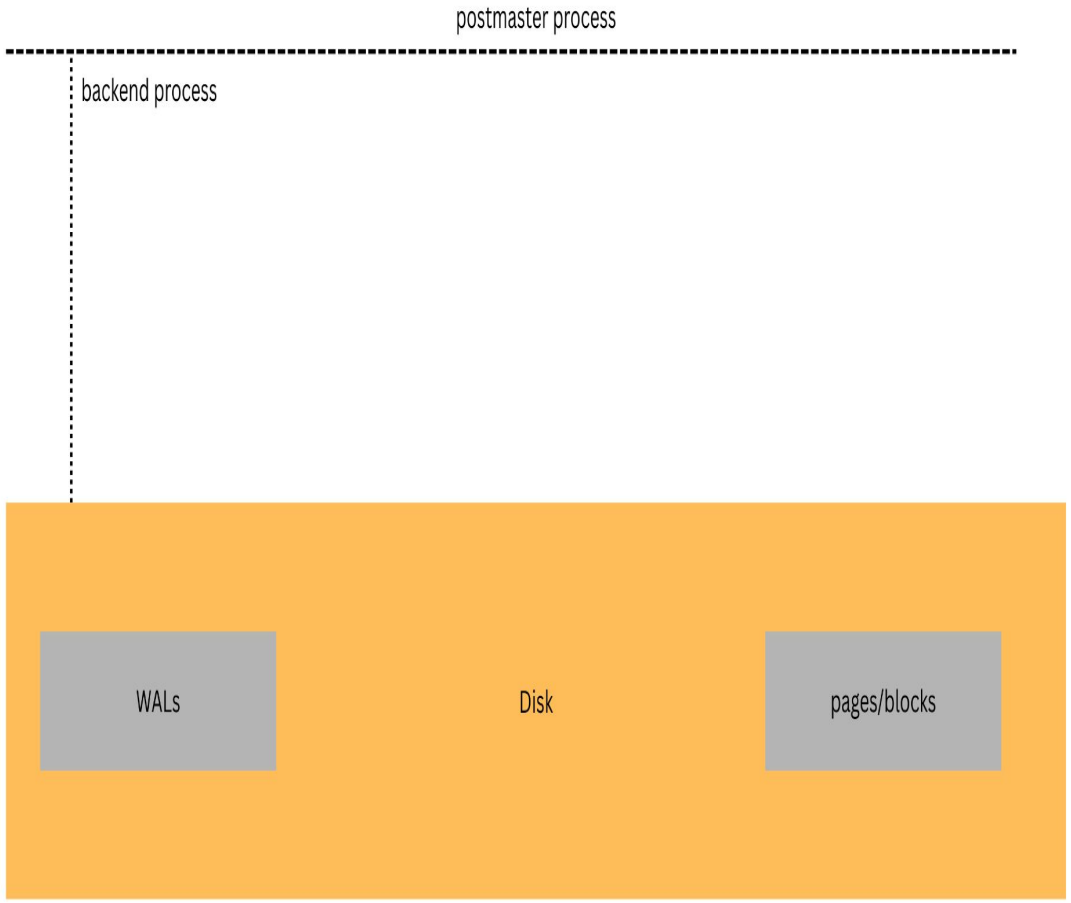
©EDB 2026 — ALL RIGHTS RESERVED.

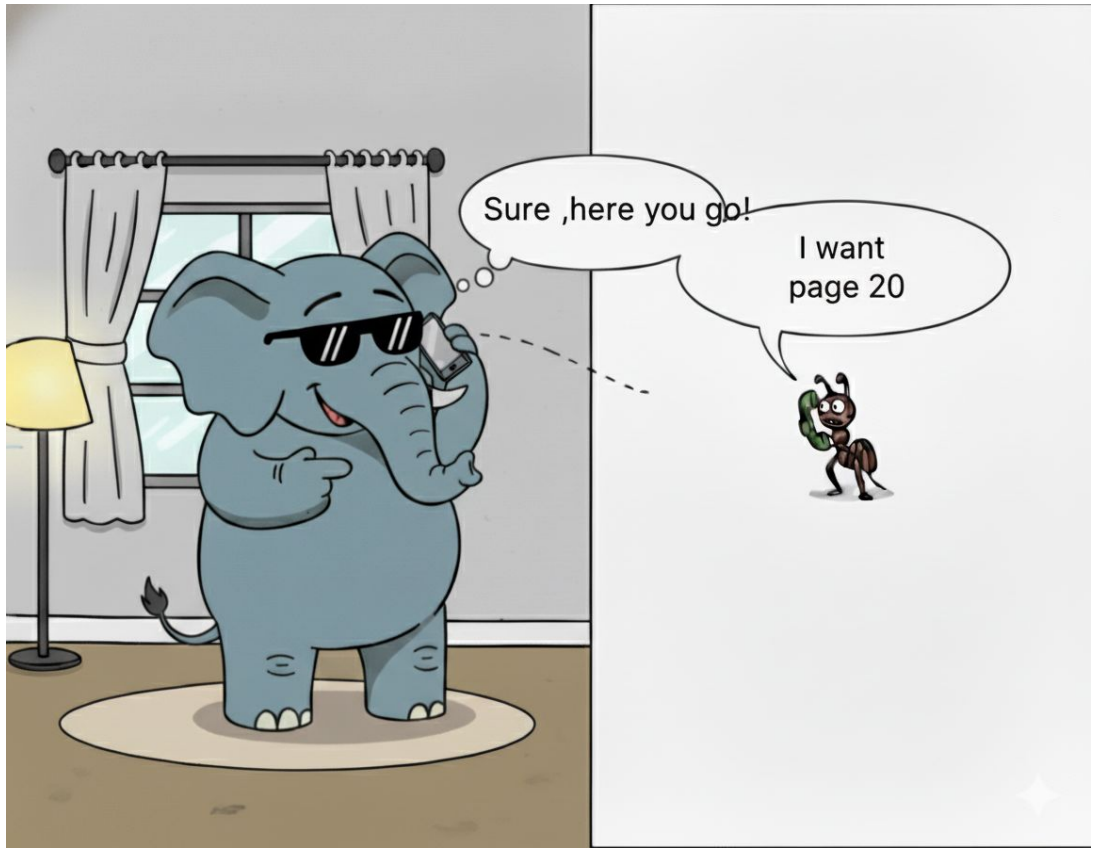
Summary (4/4)



©EDB 2026 — ALL RIGHTS RESERVED.

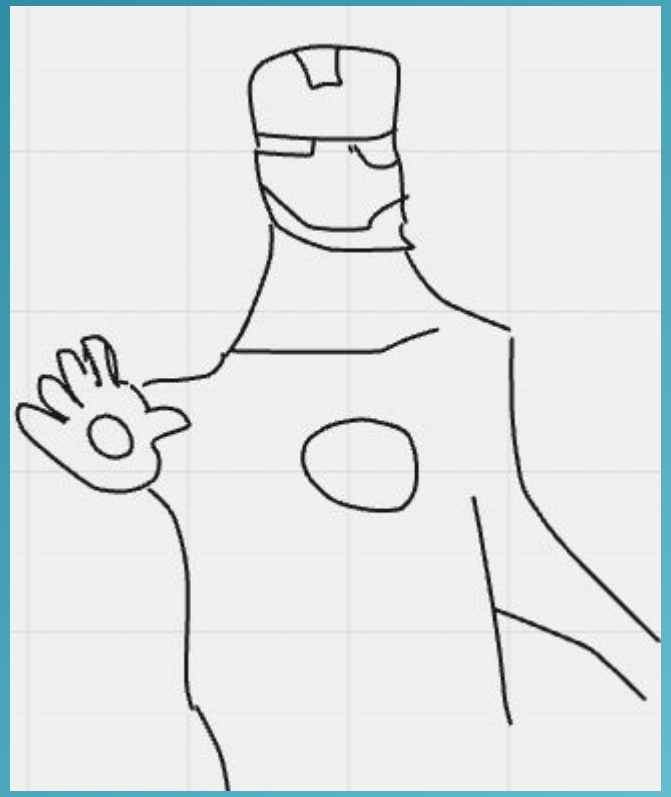
After all pages got recovered





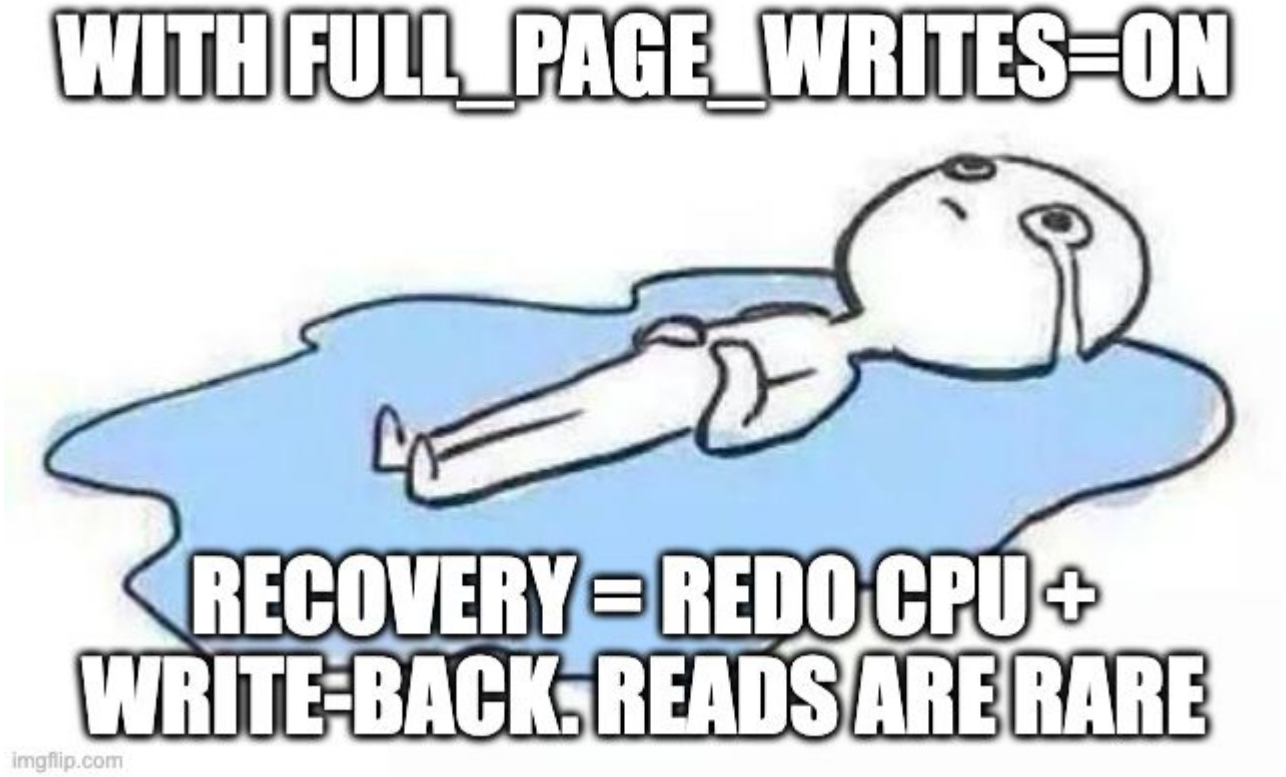
©EDB 2026 — ALL RIGHTS RESERVED.

Reality Check(s)



EDB POSTGRES AI

recovery = random page reads; skip them = ⚡ recovery.



©EDB 2026 — ALL RIGHTS RESERVED.

Never give up💪 trying, like startup process

- I totally “forgot” about checkpointer’s other job i.e doing **fsyncs**.
- During TRUNCATE replay, startup tells checkpointer to forget any pending fsync.
- But me being me, didn’t even allow checkpointer to start :(
- Fix: now checkpointer runs; it just isn’t allowed to checkpoint while pages are pending.

©EDB 2026 — ALL RIGHTS RESERVED.

ghost👻 pages in LSNHashTable

- Create a 3,000-page table, drop it, crash.
- **LSNHashTable** still listed ghost pages; fast crash recovery worker tried to read them; file was gone; boom worker crashed.
- Fix: forget LSNHashTable entries when the drop is replayed, same for truncation, for DROP DATABASE.

©EDB 2026 — ALL RIGHTS RESERVED.

The cost of blindly🙈 skipping all WALs

- Post recovery 40k XACTs, the server failed with FATAL at transaction 32768.
- The CLOG page for it had been created by a **CLOG_ZEROPAGE** record,

  I deferred everything except **RM_XACT_ID** records.
- Whole design now rests on this rule: defer a record if it has a block reference;

  everything else is eager: CLOG, multixact, smgr, xact, database operations.
- The gate became one line: **XLogRecHasAnyBlockRefs**.

©EDB 2026 — ALL RIGHTS RESERVED.

AIO broke me

- AIO moved **StartBufferIO**-**TerminateBufferIO** block into critical section.
- My first reaction was to replay **after** the page became **valid**.
- That gave me 6 different crashes, all to prove 1, we can't replay safely after **BM_VALID**.
- The fix was always in the home: **StartReadBuffersImpl**, before the I/O is issued.

©EDB 2026 — ALL RIGHTS RESERVED.

Small but Important

- **Eager** records that touch an indexed page (**TRUNCATE** reading the boundary VM/FSM page).
- **VM** pages are indexed like any page and apply their own part of each record; no special code.
- **FSM** is skipped during on-demand replay: not WAL-logged in normal operation, corrected by the next insert or VACUUM.

©EDB 2026 — ALL RIGHTS RESERVED.

Small but Important

- I use DSA for the **LSNHashTable** to fix unknown upfront number of page entries problem.
- Per-page splitting: a record's other blocks report **BLK_DONE** (redo must handle that anyway); blocks it initializes get a scratch local buffer. Replays never nest.
- **pg_basebackup** fails fast instead of copying stale pages, checkpoints are not allowed during on-demand replay.
- **Crash recovery only**: archive recovery, standby and single-user mode log a line and replay normally.

©EDB 2026 — ALL RIGHTS RESERVED.

Patch



©EDB 2026 — ALL RIGHTS RESERVED.

Numbers



EDB POSTGRES AI

Low Downtime

Scenario	Stock Recovery	Fast Recovery	Speedup
A. Warm Cache (2GB WAL)	15.1 s	**2.7 s**	5.5x
B. Cold Cache (2.7GB WAL)	96.1 s	**43.0 s**	2.2x
C. Heavy CPU (10GB WAL)	199.4 s	**106.4 s**	1.9x
D. FPW Off (1.8GB WAL)	197.2 s	**40.0 s**	4.9x

EDB POSTGRES AI

©EDB 2026 — ALL RIGHTS RESERVED.

How???

Phase from B	Stock Recovery	Fast Recovery
Redo / Scan	65.7 s	42.2 s
End-of-Recovery Checkpoint	30.4 s	0.8 s
Total Time Offline	96.1 s	43.0 s

©EDB 2026 — ALL RIGHTS RESERVED.

The I/O bottleneck - Let’s Off FPW + 1.8GB of WAL

Metric	Stock Recovery	Fast Recovery	Impact
Time to Accept Connections	**197.2 s**	**40.0 s**	**4.9x Faster**
Total Redo / Scan Time	165.7 s	39.9 s	-
Time wasted purely on I/O wait	150.2 s	28.2 s	**Random reads eliminated**
End-of-Recovery Checkpoint	31.6 s	0 s	Bypassed
First Seq Scan After Ready	1.5 s	3.9 s	Minor delay

©EDB 2026 — ALL RIGHTS RESERVED.

The Trade-Off - 1st reads pays the price

Metric(10GB WAL, 55 Million records)	**Stock Recovery**	**Fast Recovery**	**Impact**
Time to Accept Connections	**199.4 s**	**106.4 s**	**Downtime ends 90s early**
End-of-Recovery Checkpoint	26.1 s	0 s	Bypassed
First Seq Scan (Hot Table)	10.4 s	72.6 s	**Deferred Cost**
Background Drain Time	0 s	506.8 s	**Traded to Background**

©EDB 2026 — ALL RIGHTS RESERVED.

What’s next? - A Request to break it



©EDB 2026 — ALL RIGHTS RESERVED.

What’s next? - TODO List

- Parallelizing fast crash recovery worker.
- LSNHashTable may hit the RAM limit, maybe spilling helps?
- Thanks to Alvaro Herrera for doing an IRL patch reviewing, so cooooooooooooooooooooooool.
- Many bugs I am sure people on the mailing list will be happy to share 😁.

©EDB 2026 — ALL RIGHTS RESERVED.

Contributors (Good stuff - it’s on them , Bad stuff - it’s on me :)

```
commit 7c5fe3dcd18fca579ee1ade5188e5003bda875bd (HEAD -> master)
Author: Srinath Reddy Sadipiralla <srinath2133@gmail.com>
Date:   Thu Oct 15 20:46:34 2026 +0530

    Acknowledge talk rehearsal participants and slide feedback

    This commit formally recognizes the team members who helped
    refine the presentation. A special thanks to Floor for
    organizing the rehearsal, and to everyone who took time out
    of their schedules to attend and provide valuable feedback
    to improve the slides.

    Author: Srinath Reddy Sadipiralla
    Reviewed-by: Boriss Mejias
    Reviewed-by: Floor Drees
    Reviewed-by: Jonathan Gonzalez
    Reviewed-by: Peter Eisentraut
    Reviewed-by: Suraj Kharage
    Reviewed-by: Xavier Fischer
```


©EDB 2026 — ALL RIGHTS RESERVED.

Feedback



EDB POSTGRES AI

©EDB 2026 — ALL RIGHTS RESERVED.

Q&A



EDB POSTGRES AI

Thank You :)



Have more questions on this talk?/anything Postgres?/how to fight Skynet/Ultron? - meet me at the EDB booth.

To know which “Earth” I am from and more info

©EDB 2026 — ALL RIGHTS RESERVED.