PostgresPro

Ptrack 2.0: yet another block-level incremental backup engine

Alexey Kondratov

Postgres Professional

PGCon'20, May 27-28

Outline

- Motivation: incremental backups
- How Postgres works with data?
- Ptrack 1.0 recap
- Ptrack 2.0 overview
  - In-memory data structure and operations
  - Durability
- Limitations
- Public SQL API and configuration
- Benchmarks

Incremental backup

- Only **50%** out of our **10 GB** database has changed since the last backup.
- Copy only those **5 GB** during incremental backup instead of full **10 GB**.
- Spend twice as less disk space and time.
- Profit!

Incremental backup strategies

- **PAGE***: scan all WAL files in the archive from the moment of the previous full or incremental backup. Newly created backup contains only those pages that were mentioned in WAL records.
- **DELTA***: read all data files in PGDATA directory, compare LSNs and copy only those pages, that where changed since previous backup.

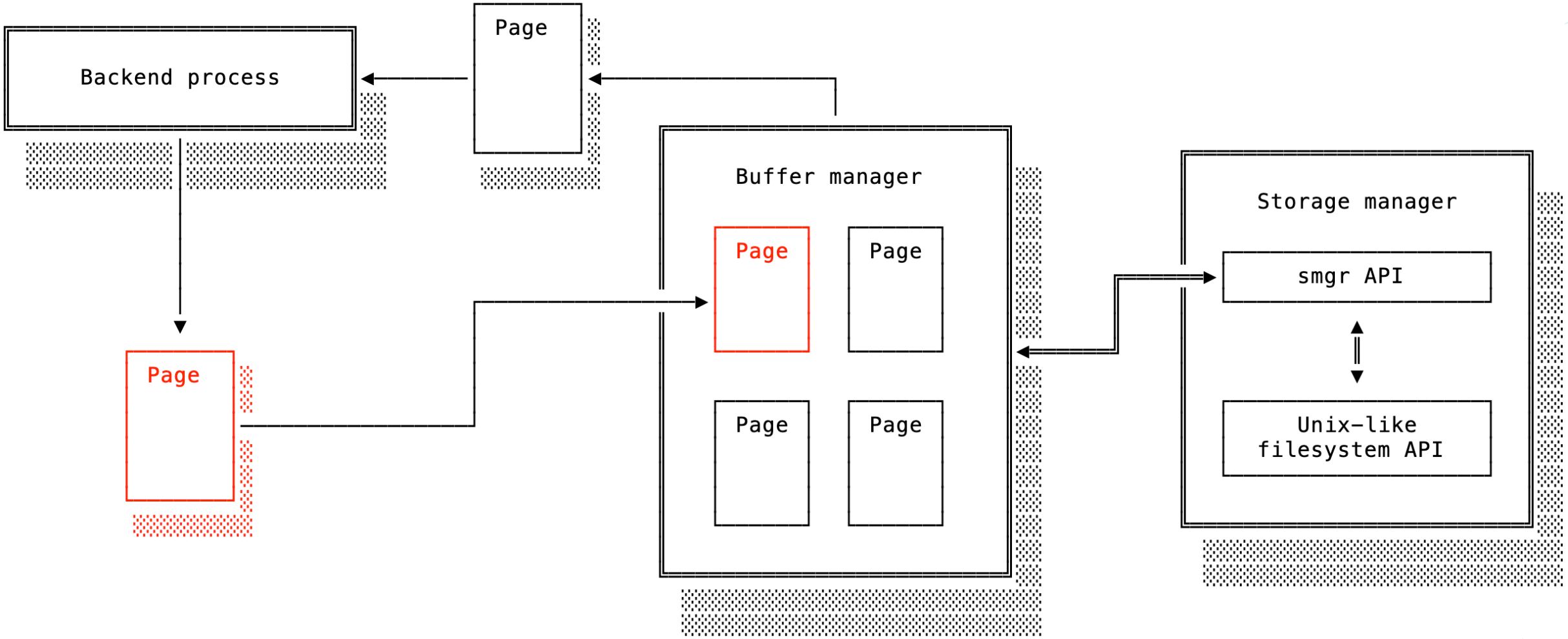
* pg_probackup terminology

Incremental backup strategies

- **PAGE***: scan all WAL files in the archive from the moment of the previous full or incremental backup. Newly created backup contains only those pages that were mentioned in WAL records.
- **DELTA***: read all data files in PGDATA directory, compare LSNs and copy only those pages, that where changed since previous backup.
- **PTRACK**: PostgreSQL tracks page changes on the fly, so we receive a ready to execute map of modified blocks.

* pg_probackup terminology

How Postgres works with data?



How Postgres works with data?

Code example: heapam.c > heap_insert()

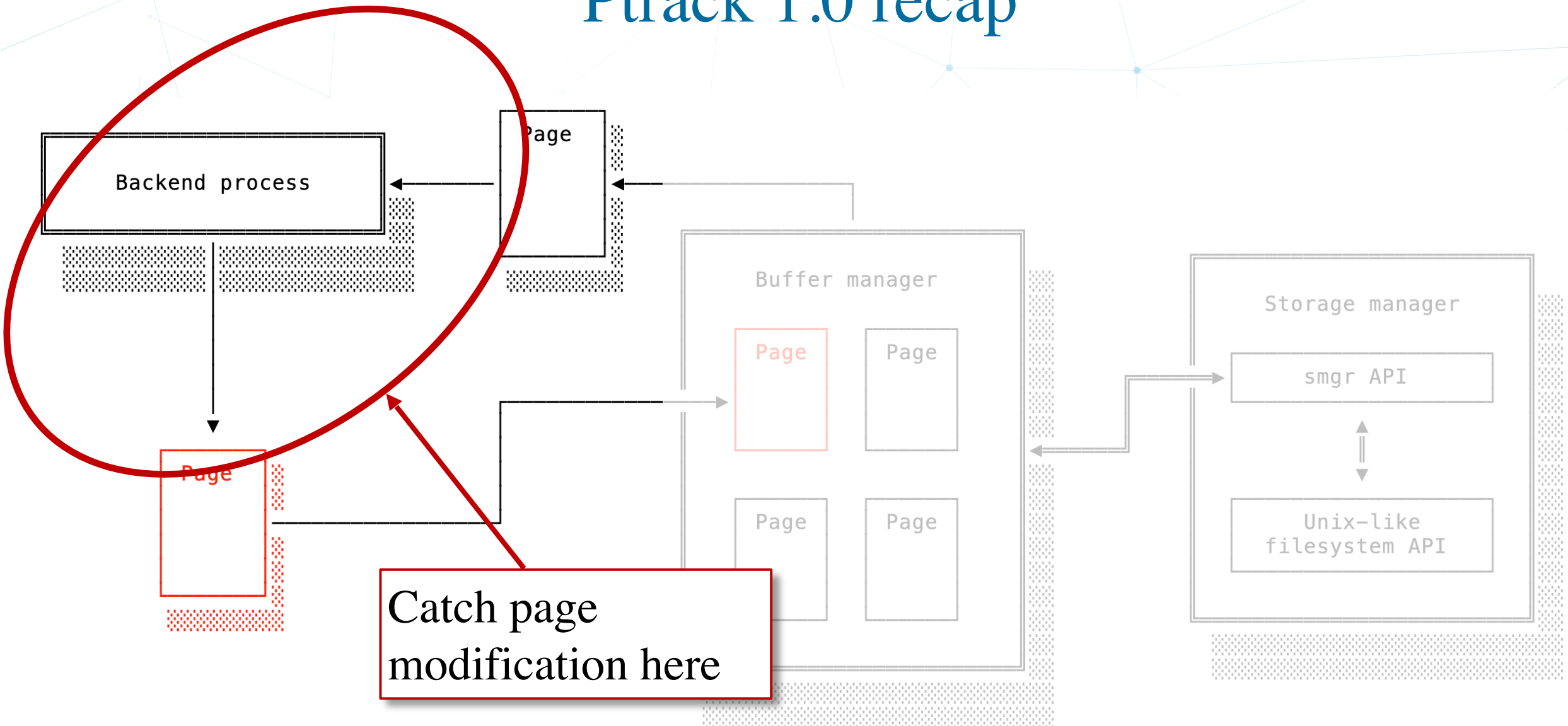
```
/*
 * Find buffer to insert this tuple into.  If the page is all visible,
 * this will also pin the requisite visibility map page.
 */
buffer = RelationGetBufferForTuple(relation, heaptup->t_len,
                                   InvalidBuffer, options, bistate,
                                   &vmbuffer, NULL);
...

/* NO EREPORT(ERROR) from here till changes are logged */
START_CRIT_SECTION();

RelationPutHeapTuple(relation, buffer, heaptup,
                     (options & HEAP_INSERT_SPECULATIVE) != 0);

...

MarkBufferDirty(buffer);

/* XLOG stuff */
if (!(options & HEAP_INSERT_SKIP_WAL) && RelationNeedsWAL(relation))
```

```
/* XLOG stuff */
if (!(options & HEAP_INSERT_SKIP_WAL) && RelationNeedsWAL(relation))
{
    xl_heap_insert xlrec;

    ...

    XLogBeginInsert();
    XLogRegisterData((char *) &xlrec, SizeOfHeapInsert);

    ...

    recptr = XLogInsert(RM_HEAP_ID, info);

    PageSetLSN(page, recptr);
}

END_CRIT_SECTION();

UnlockReleaseBuffer(buffer);
```

Ptrack 1.0 recap

- Use the same Buffer/Storage Manager machinery from PostgreSQL for Ptrack data pages.
- Add another relation fork ***_ptrack** in addition to ***_fsm / *_vm**.
- Track page modification in each place, when it is done.
- Read and reset Ptrack map after **pg_start_backup**().

Ptrack 1.0 recap



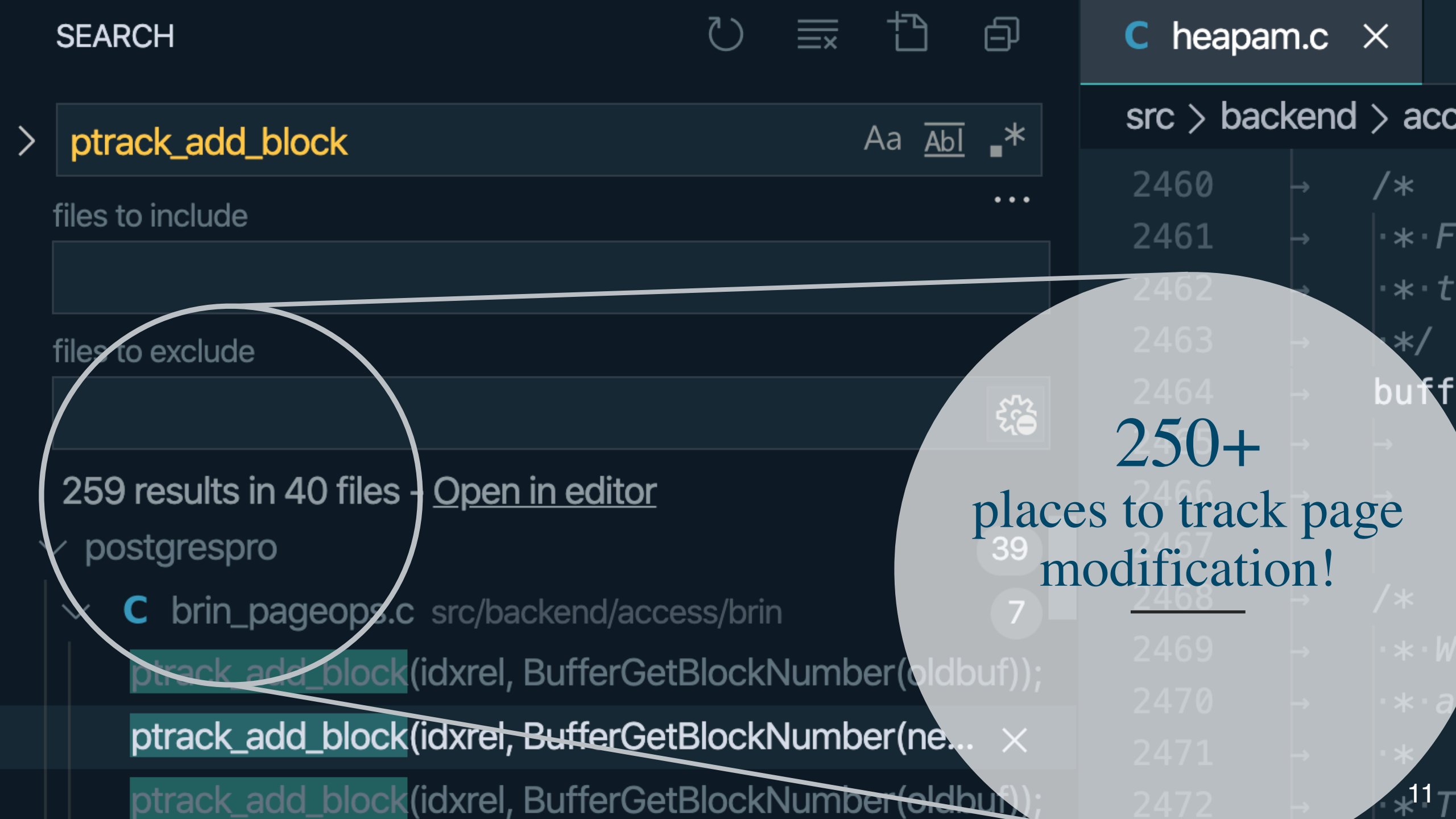
Ptrack 1.0 recap

Code example: heapam.c > heap_insert()

```
/*
 * Find buffer to insert this tuple into.  If the page is all visible,
 * this will also pin the requisite visibility map page.
 */
buffer = RelationGetBufferForTuple(relation, heaptup->t_len,
                                   InvalidBuffer, options, bistate,
                                   &vmbuffer, NULL);
...

/* NO EREPORT(ERROR) from here till changes are logged */
ptrack_add_block(relation, BufferGetBlockNumber(buffer));
START_CRIT_SECTION();

RelationPutHeapTuple(relation, buffer, heaptup,
                     (options & HEAP_INSERT_SPECULATIVE) !

...

MarkBufferDirty(buffer);
```

```
/* XLOG stuff */
if (!(options & HEAP_INSERT_SKIP_WAL) && RelationNeedsWAL(relation))
{
    xl_heap_insert xlrec;

    ...

    XLogBeginInsert();
    XLogRegisterData((char *) &xlrec, SizeOfHeapInsert);

    = XLogInsert(RM_HEAP_ID, info);

    LSN(page, recptr);

END_CRIT_SECTION();

UnlockReleaseBuffer(buffer);
```

We must track page modification before critical section


SEARCH
heapam.c
src > backend > acc
ptrack_add_block
Aa
Abl
files to include
files to exclude
259 results in 40 files - Open in editor
postgrespro
brin_pageops.c src/backend/access/brin
ptrack_add_block(idxrel, BufferGetBlockNumber(oldbuf));
ptrack_add_block(idxrel, BufferGetBlockNumber(ne...
ptrack_add_block(idxrel, BufferGetBlockNumber(oldbuf));
250+
places to track page modification!

SEARCH

heapam.c

src > backend > acc

ptrack_add_block

files to include

files to exclude

259 results in 40 files - Open in editor

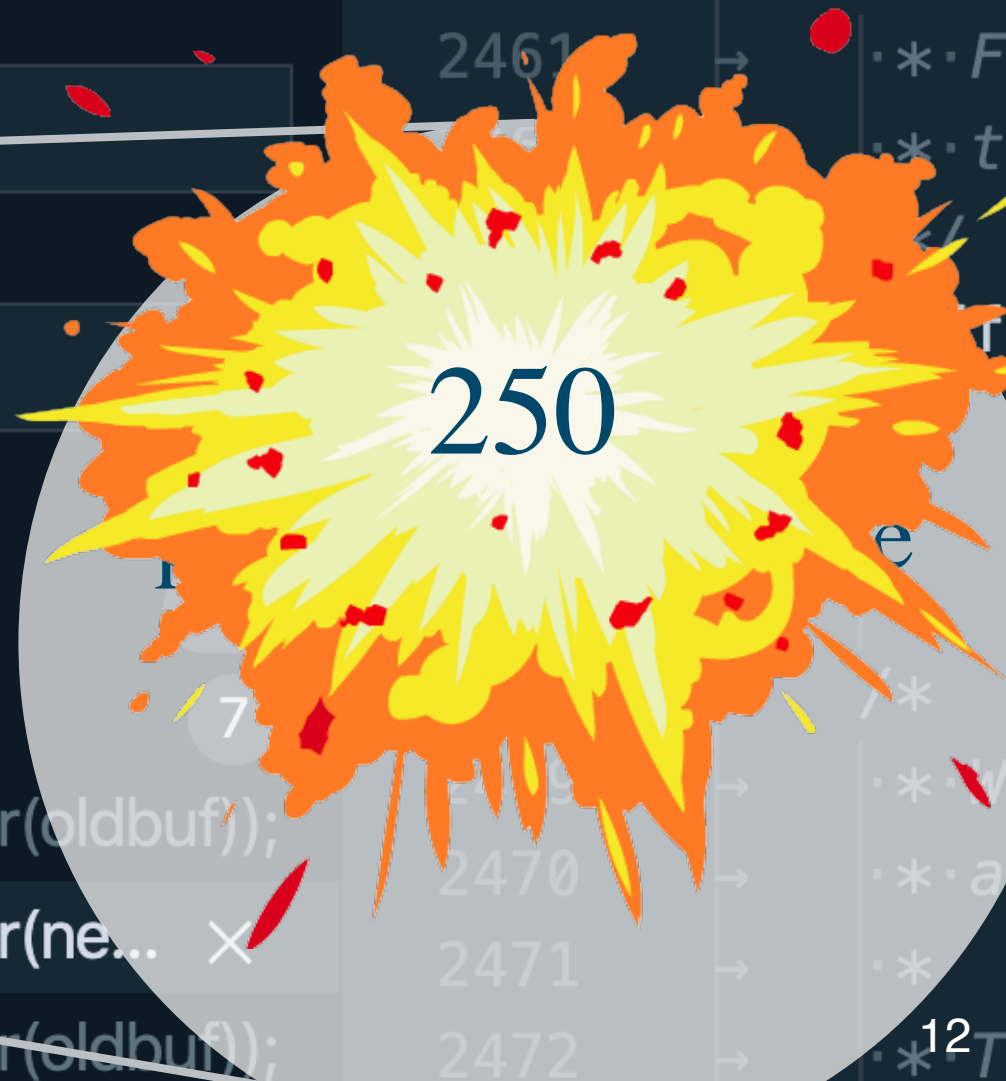
postgrespro

brin_pageops.c src/backend/access/brin

ptrack_add_block(idxrel, BufferGetBlockNumber(oldbuf));

ptrack_add_block(idxrel, BufferGetBlockNumber(ne...

ptrack_add_block(idxrel, BufferGetBlockNumber(oldbuf));

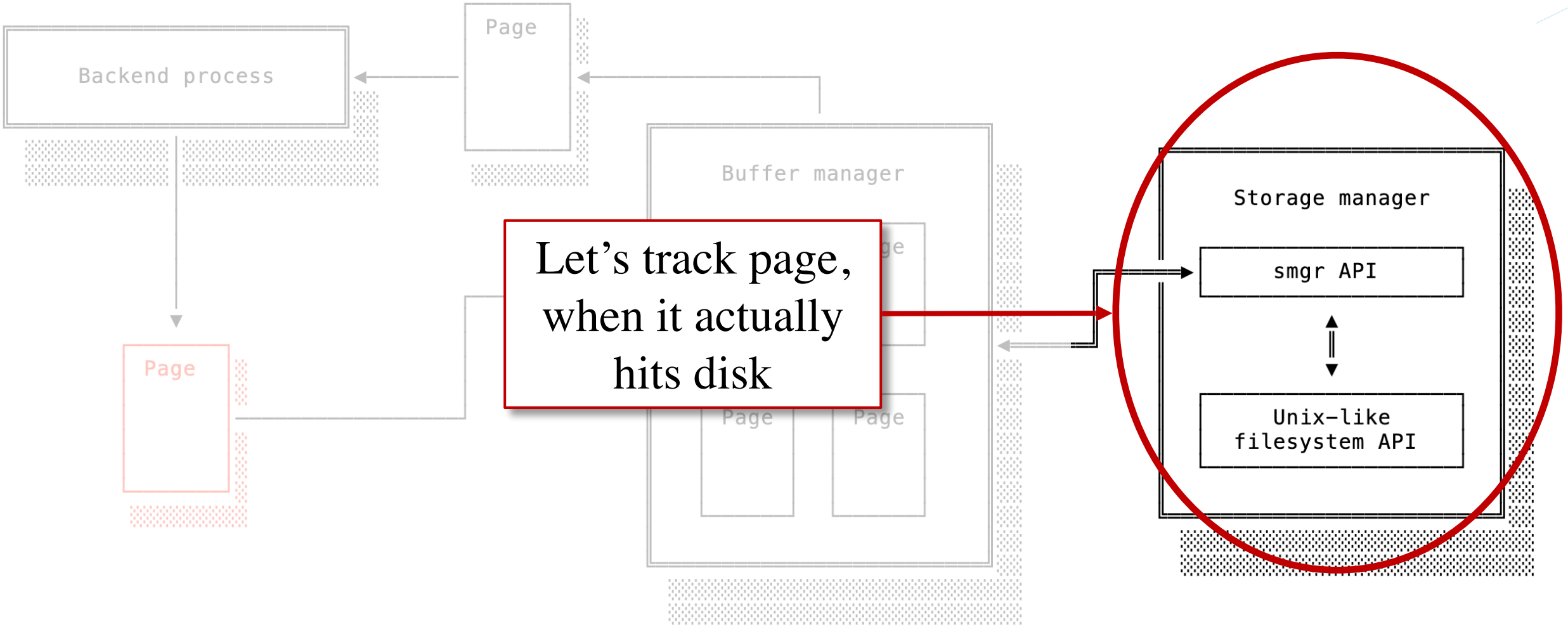


Ptrack 1.0 drawbacks

- Cannot mark blocks in a single place like **MarkBufferDirty**(), since it is called inside critical section.
- Too many places to put tracking routine call, too easy to miss some of them.
- Fused into PostgreSQL core.
- One extra file per relation.
- Additional workarounds to prevent data loss during Ptrack map reset.

Ptrack 2.0: can we do better?

Ptrack 2.0 overview



Ptrack 2.0 overview

- Postgres mostly modifies everything via Buffer manager, so we can catch these operations at the very bottom level, when the **affected pages are evicted back to disk**.
- Pages on replica and during redo process follow the same path, so there is no additional work to do.
- However, there are certain operations where Postgres simply copies the entire directory with all its content: **CREATE DATABASE**, **ALTER DATABASE … SET TABLESPACE**.

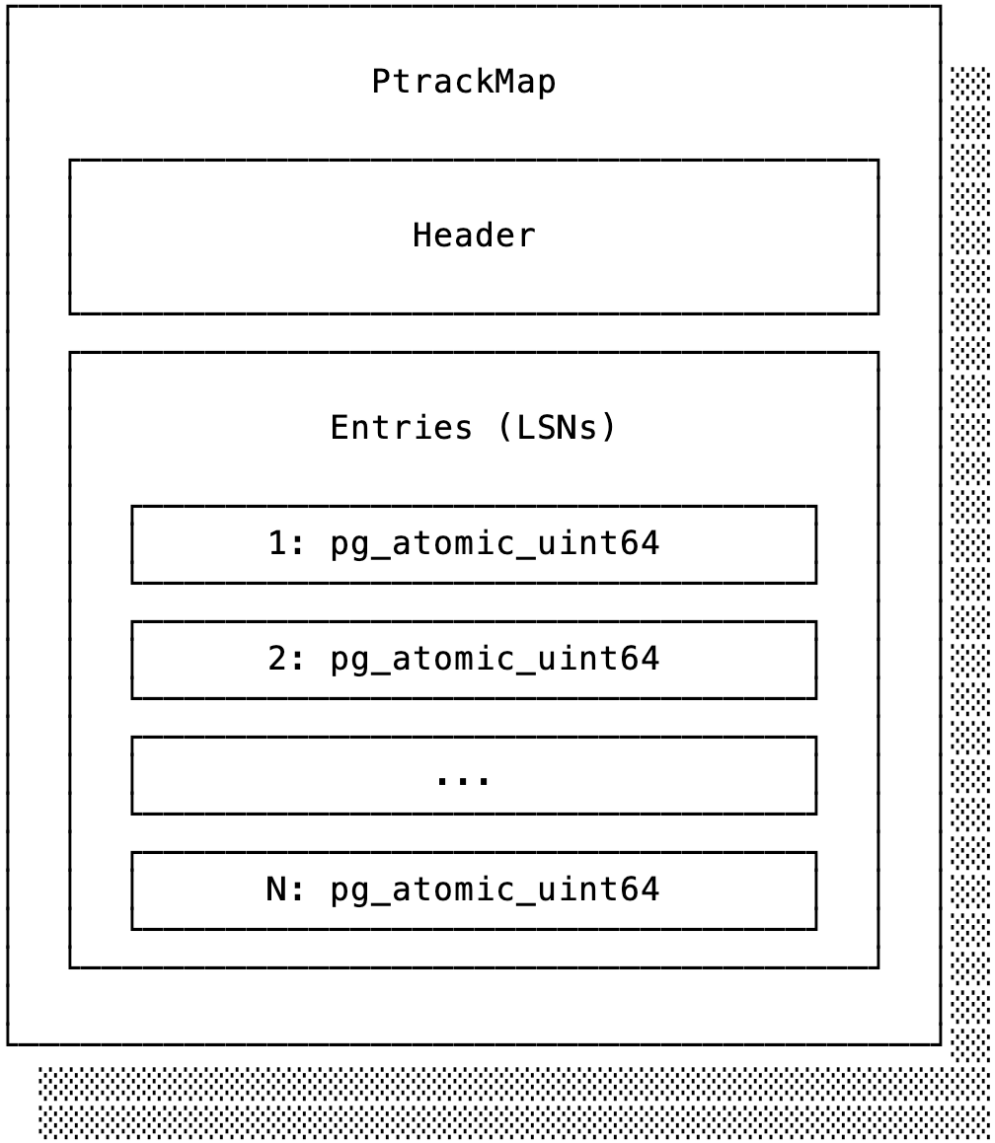
Ptrack 2.0 hooks

Ptrack core patch adds following hooks:

- smgrwrite() / mdwrite() hook
- smgrextend() / mdextend() hook
- copydir() hook
- Checkpoint (ProcessSyncRequests) hook

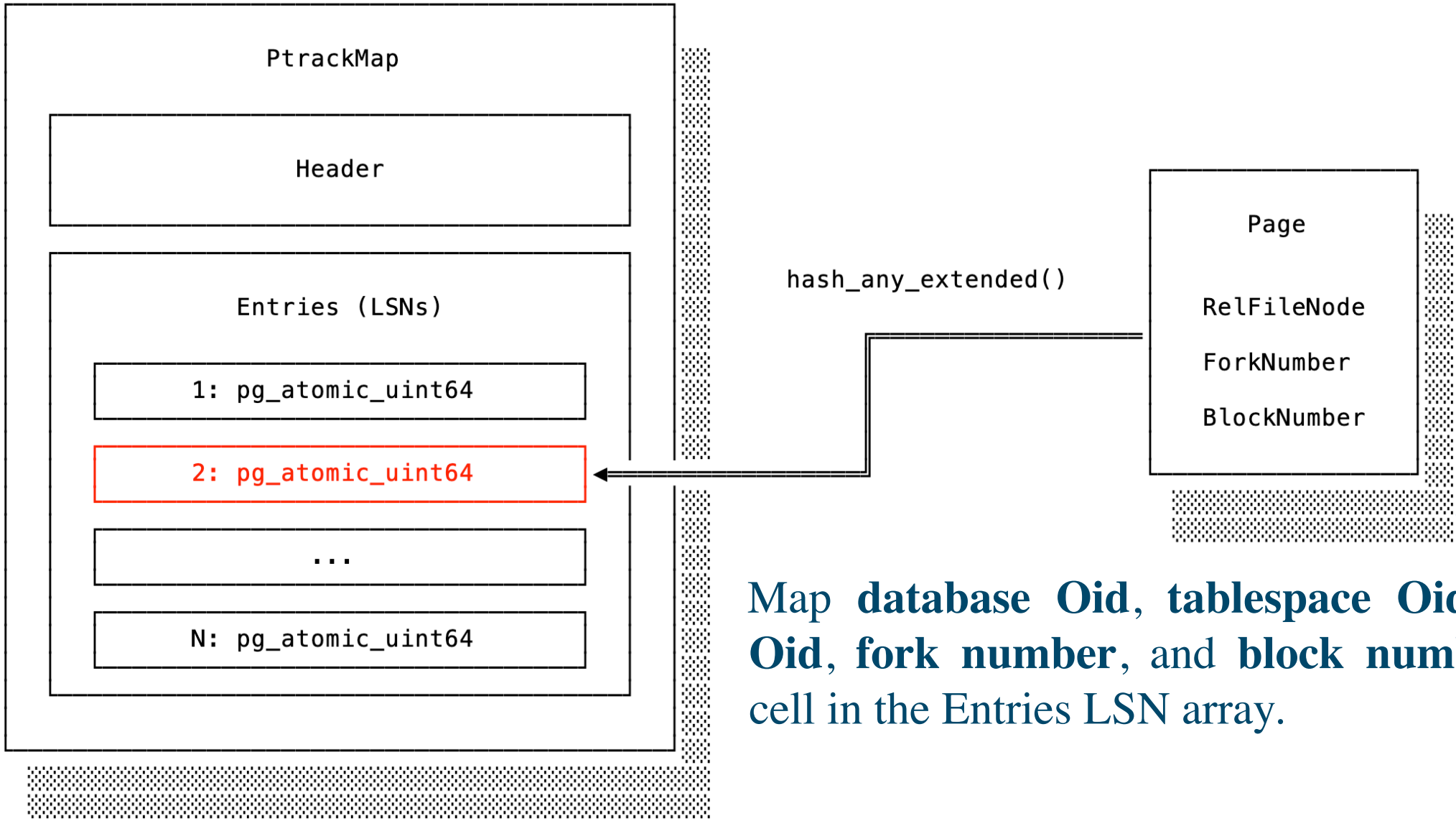
Only **four** places instead of **250** = **win**!

Ptrack 2.0 structure



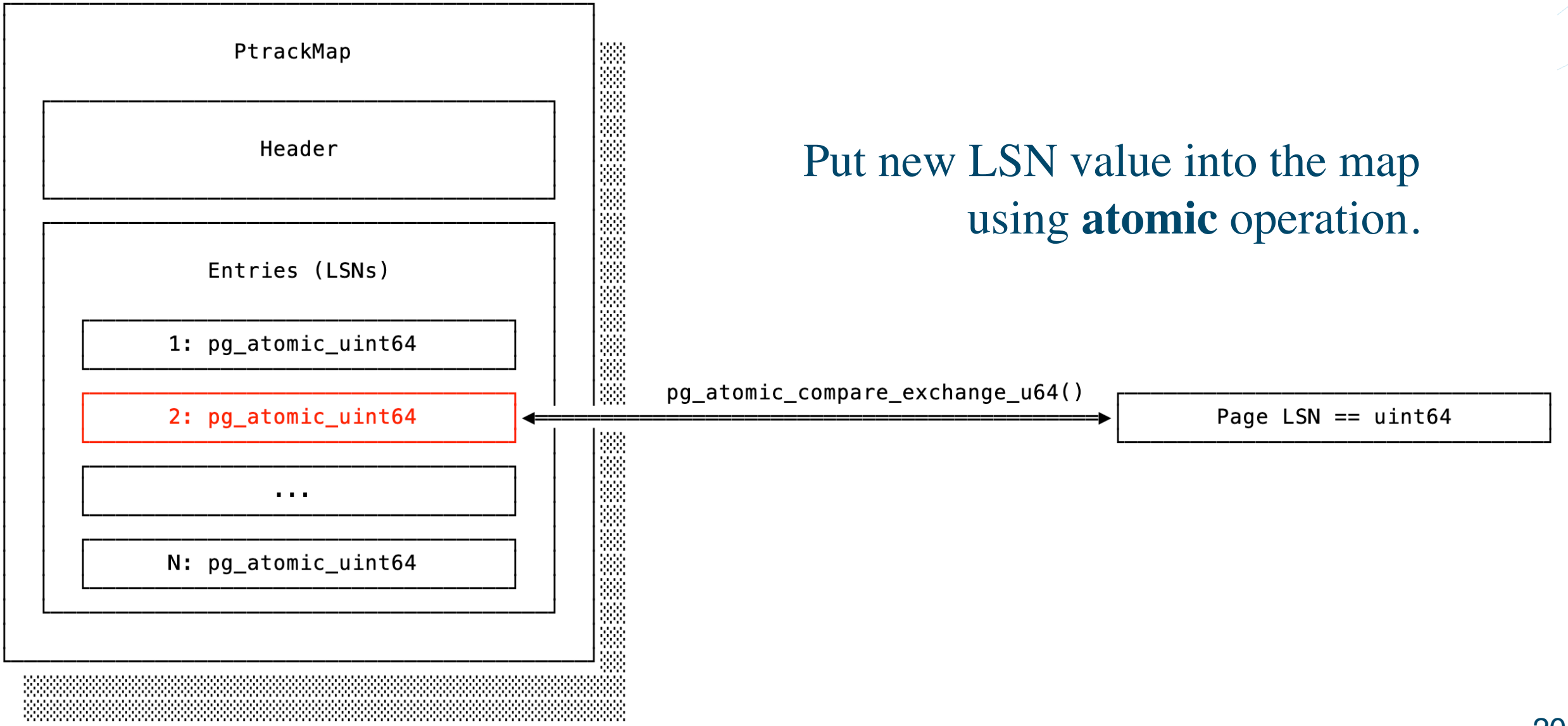
- Use a single cluster-wide **map of a fixed size** for modified page LSNs tracking.
- Load it in memory from the file using mmap().

Ptrack 2.0 structure



Map **database Oid**, **tablespace Oid**, **relation Oid**, **fork number**, and **block number** into a cell in the Entries LSN array.

Ptrack 2.0 operations



Put new LSN value into the map using **atomic** operation.

Ptrack 2.0 durability

Durably flush Ptrack map to disk during checkpoint:

1. Keep **ptrack.map** file since last checkpoint **intact**.
2. Read Ptrack map records **atomically** one by one into the local buffer.
3. Write buffer content into a transient file **ptrack.map.tmp**.
4. Calculate **CRC checksum** and write it at the end of file.
5. Durably replace **ptrack.map** with newly created **ptrack.map.tmp**.

Ptrack 2.0 limitations

- Due to the fixed size of Ptrack map there are may be false positives, but **never false negatives**. However, with **64 MB** of map you can track per-block changes in a **64 GB** database **without false positives**.
- You can only use Ptrack safely with **wal_level >= 'replica'**, since certain commands are designed not to write WAL at all if wal_level is minimal.
- Currently, you cannot resize Ptrack map in runtime, only **on postmaster start**.

Ptrack 2.0 public SQL API

- **ptrack_version**() — returns Ptrack version string.
- **ptrack_init_lsn**() — returns LSN of the Ptrack map initialization.
- **ptrack_get_pagemapset**('LSN') — returns a set of changed data files with bytea bitmaps of changed blocks since specified LSN.

Ptrack 2.0 configuration

- The only one configurable option is **ptrack.map_size** (in MB).
- To completely avoid false positives it is recommended to set **ptrack.map_size** to **1 / 1000** of expected PGDATA size.
- To disable Ptrack and clean up all remaining service files set **ptrack.map_size** to **0**.

Ptrack 2.0 usage

```
echo "shared_preload_libraries = 'ptrack'" >> postgres_data/postgresql.conf
echo "ptrack.map_size = 64" >> postgres_data/postgresql.conf
```

```
postgres=# CREATE EXTENSION ptrack;
```

```
postgres=# SELECT ptrack_get_pagemapset('0/186F4C8');
           ptrack_get_pagemapset
-----------------------------------------------
 (global/1262,"\\x010000000000000000000000")
 (global/2672,"\\x020000000000000000000000")
 (global/2671,"\\x020000000000000000000000")
(3 rows)
```

Ptrack 2.0 benchmarks

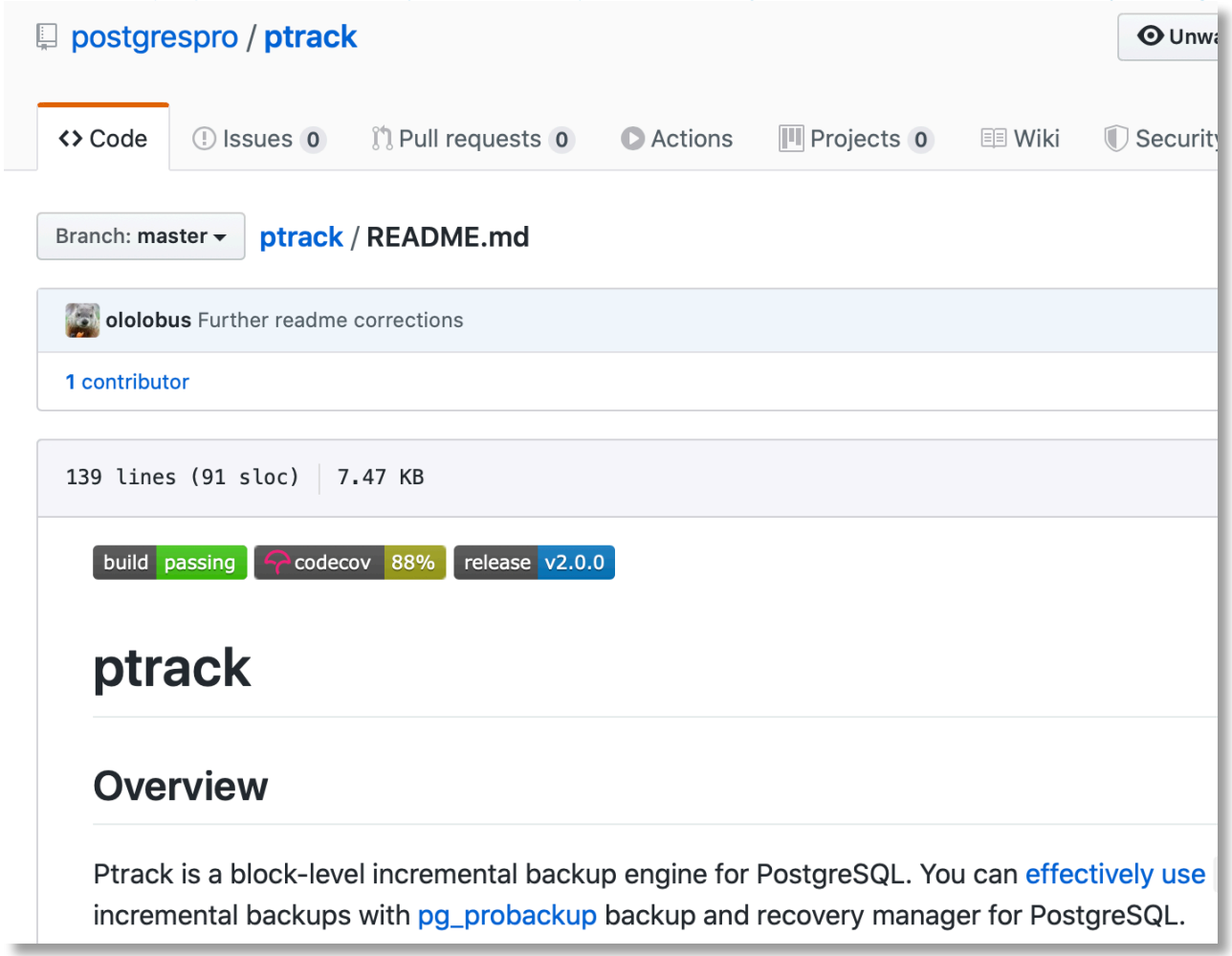
- **tmpfs** partition, **~1 GB** database (pgbench **scale = 133**), all defaults.
- **No pgbench_tellers** and **pgbench_branches updates** to lower lock contention.
- pgbench -s133 -c40 -j1 -n -P15 -T300 -f pgb.sql

ptrack.map_size, MB	REL_12_STABLE	32	64	256	512	1024
TPS	16900	16890	16855	16468	16490	16220

Open source

Ptrack and **pg_probackup** are available on GitHub:

- github.com/postgrespro/ptrack
- github.com/postgrespro/pg_probackup



Feedback

If you have any questions or comments:

- kondratov.aleksey@gmail.com
- github.com/ololobus
- twitter.com/ololobuss

Thank you!