zheap: less bloat, fewer writes, and just plain smaller

- Amit Kapila, Robert Haas | PGCon 2018

Contents

- Why zheap?
- Purpose of undo
- Zheap
- TPD (Extended transaction data)
- Undo
- Indexing in zheap
- Performance data
- Benefits and Drawbacks

Bloat: Motivation and Definition

- Original motivation: PostgreSQL tables tend to bloat, and when they do, it's hard to get rid of the bloat.
- Bloat occurs when the table and indexes grow even though the amount of real data being stored has not increased.
- Bloat is caused mainly by updates, because we must keep both the old and new updates for a period of time.
- Bloat can be a concern because of increased disk consumption, but typically a bigger concern is performance loss – if a table is twice as big as it “should be,” scanning it takes twice as long.

Bloat: Why a new storage format?

- All systems that use MVCC must deal with multiple row versions, but they store them in different places.
  - PostgreSQL and Firebird put all row versions in the table.
  - Oracle and MySQL put old row versions in the undo log.
  - SQL Server puts old row versions in tempdb.
- Leaving old row versions in the table makes cleanup harder – sometimes need to use CLUSTER or VACUUM FULL.
- Improving VACUUM helps contain bloat, but can't prevent it completely.

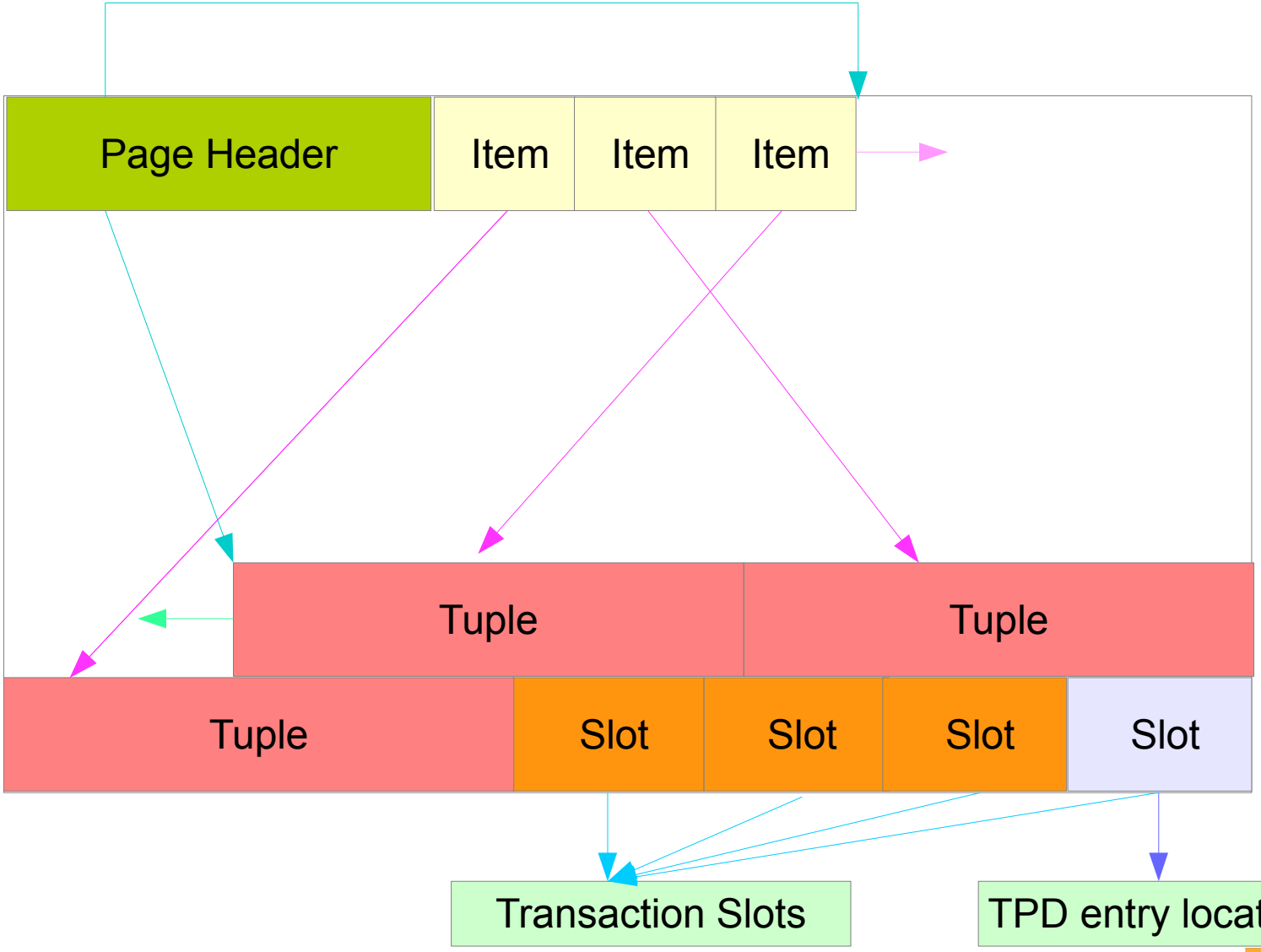
zheap: High-level goals

- Better bloat control
  - Perform updates “in place” to avoid creating bloat (when possible).
  - Reuse space right after COMMIT or ABORT – little or no need for VACUUM.
- Fewer writes
  - Eliminate hint-bits, freezing, and anything else that could dirty a page **other than a data modification**.
  - Allowing in-place updates when index column is updated by providing delete-marking in index. Indexes are not touched if the indexed columns are not changed. This will also help in containing the bloat.
- Smaller in size
  - Narrower tuple headers – most transactional information not stored within the page itself.
  - Eliminate most alignment padding.

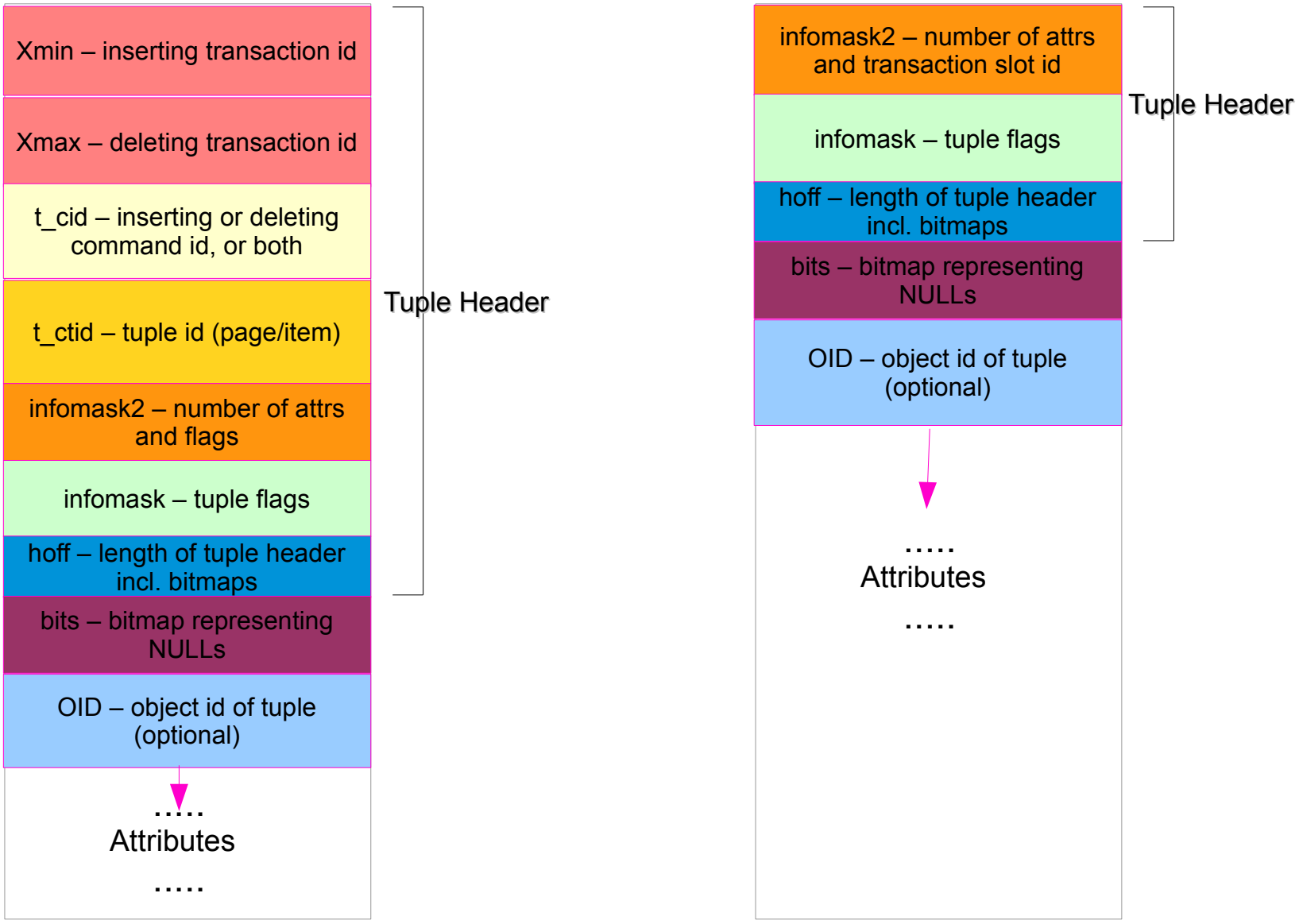
zheap: General Idea

- Each zheap page has fixed set of transaction slots containing transaction info (transaction id, epoch and the latest undo record pointer of that transaction).
- As of now, the number of slots are configurable and default value of same is four.
- Each transaction slot occupies 16 bytes.
- We allow the transaction slots to be reused after the transaction becomes too old to matter (older than oldest xid having undo), committed or rolled back. This allows us to operate without having too many slots.

zheap: Page Format



zheap: Tuple Format



Purpose of undo

- Undo is responsible for reversing the effects of aborted transactions.
- When a transaction performs an operation, it also writes it to the write-ahead log (REDO) and records the information needed to reverse it (UNDO). If the transaction aborts, UNDO is used to reverse all changes made by the transaction.
- It stores old versions of rows required for MVCC.
- Independent of avoiding bloat, having undo can provide systematic framework for cleaning.
  - For example, if a transaction creates a table and, while that transaction is still in progress, there is an operating system or PostgreSQL crash, the storage used by the table is permanently leaked. This could be fixed by undo.

zheap

- In this new heap, we need to emit an UNDO record for each of the Insert, Delete and Update operations.
- For an INSERT, we will insert the new write and emit UNDO which will remove it.
- For Delete, we will emit an UNDO which will put back the row.

zheap

- Update can be done in two ways:
  - An in-place update in which old row can be maintained in UNDO and new row in heap.
  - Cases where in-place update is not possible like
    - when the new row is bigger than old row and can't fit in same page, we need to perform Delete combined with Insert and emit an UNDO to reverse both the operations.
    - Index column is updated.
- First strategy is preferable as that doesn't bloat the heap, but I think we can't avoid to have non-in-place updates in some cases.

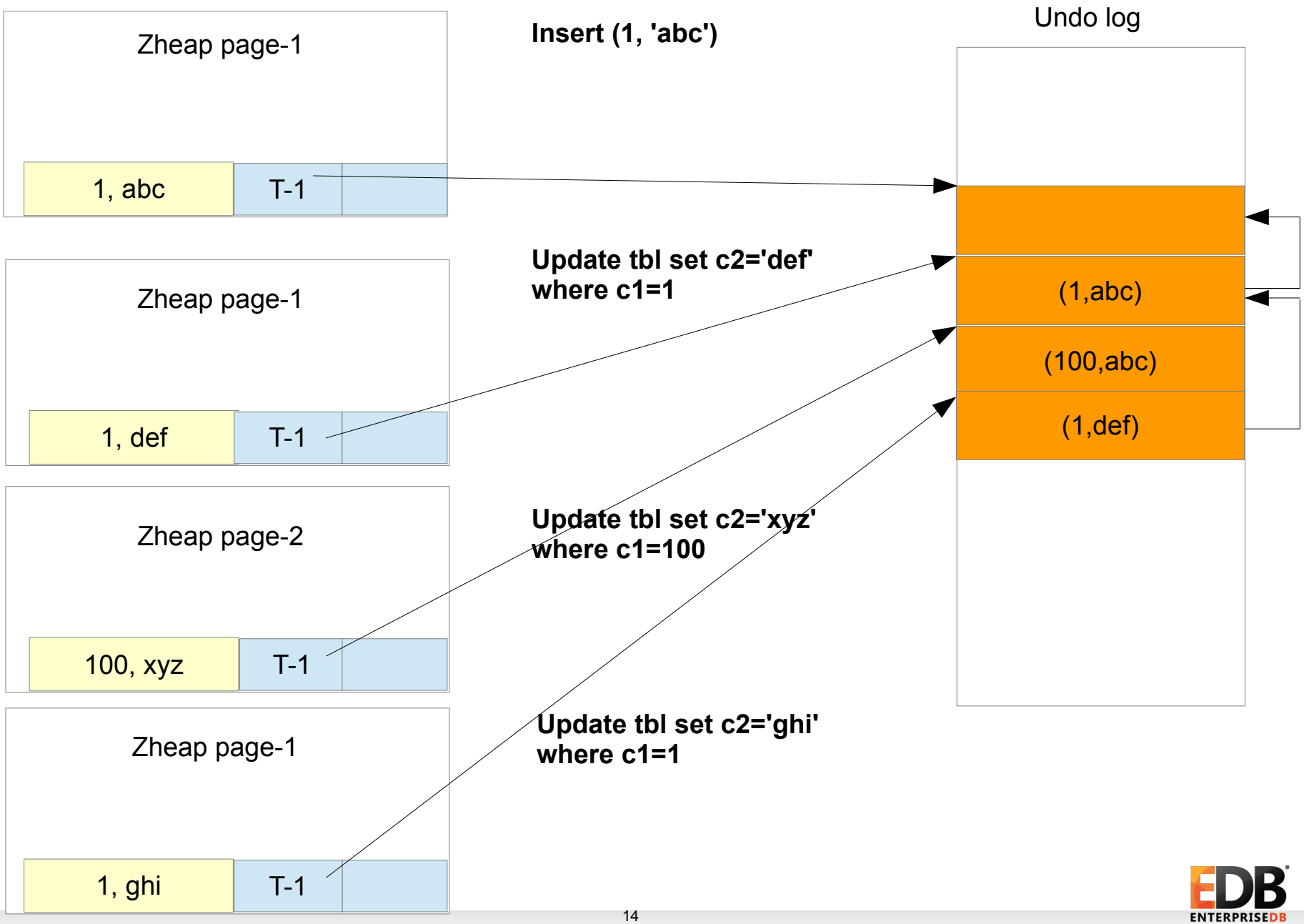
zheap: General Idea

- During scans, we need to make a copy of the tuple instead of just holding the pin on a page to avoid update of the tuple as we have in-place updates.
- Space can be reclaimed for
  - deletes
  - Non-in-place updates
  - updates that update to a smaller value.
- We can reuse the space when either the transaction that has performed the operation is committed and all-visible or if it is committed.
- We can immediately reclaim the space for inserts that are rolled-back.

Undo chains and visibility

- The undo chain is formed at page level for each transaction.
- Each undo record header contains the location of previous undo record of the transaction on the same page if any as shown in next slide.
- When the current tuple is not visible to the scan snapshot, we can traverse undo chain to find the version which is visible (if any).

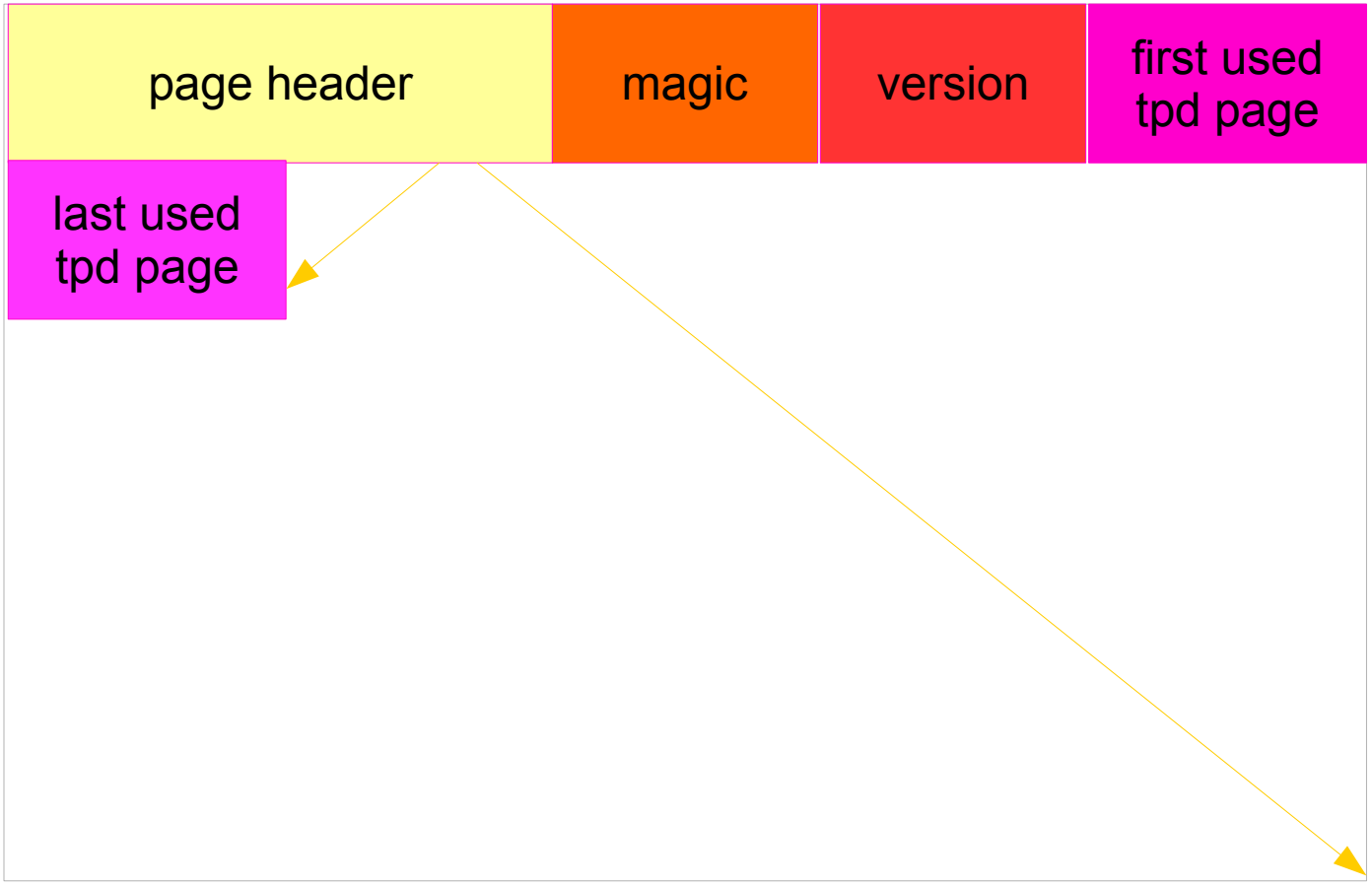
Undo chains


Zheap page-1
1, abc
T-1
Insert (1, 'abc')
Undo log
Zheap page-1
1, def
T-1
Update tbl set c2='def'
where c1=1
(1,abc)
Zheap page-2
100, xyz
T-1
Update tbl set c2='xyz'
where c1=100
(100,abc)
Zheap page-1
1, ghi
T-1
Update tbl set c2='ghi'
where c1=1
(1,def)

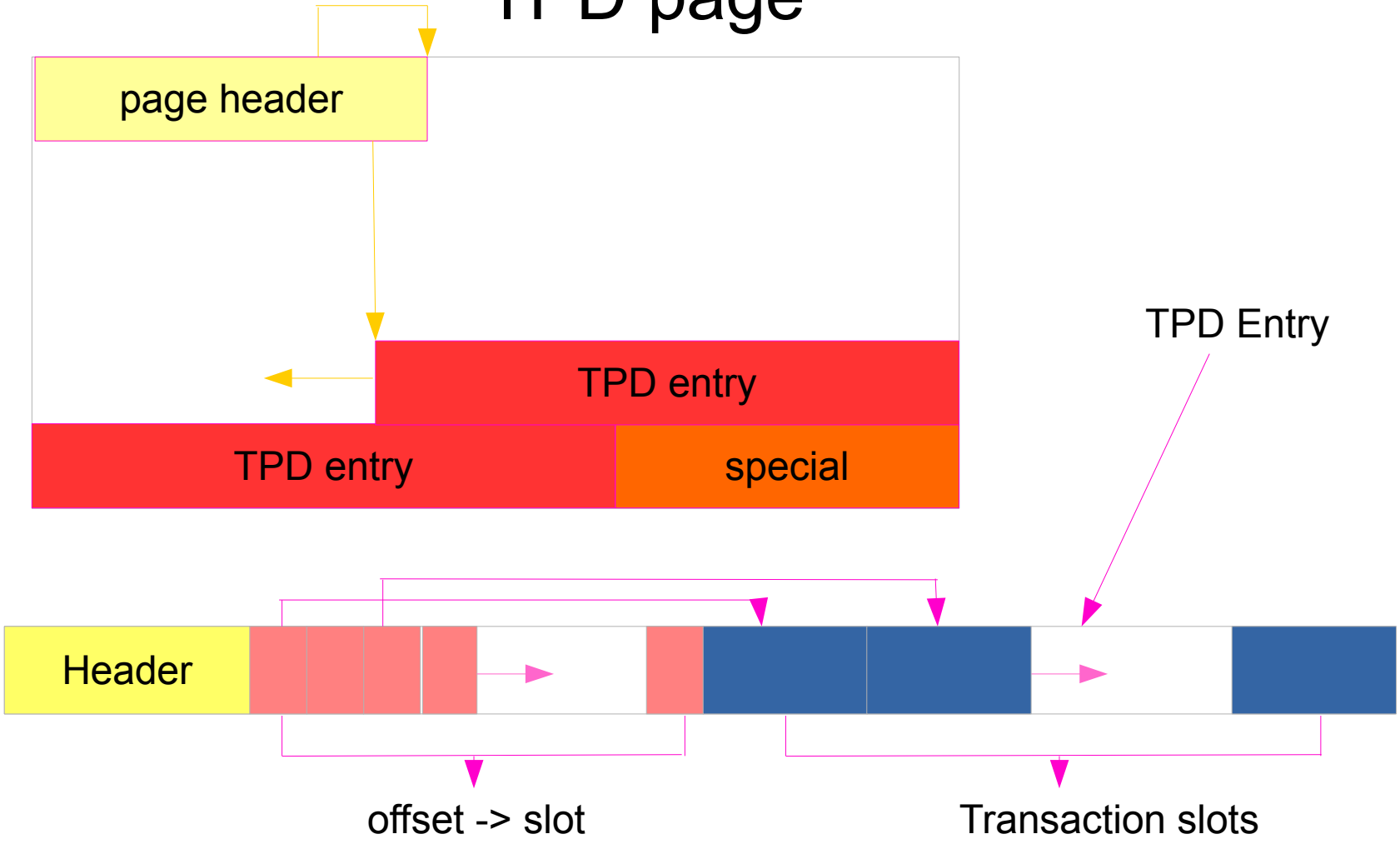
TPD

- TPD is nothing but temporary data page consisting of extended transaction slots from heap pages.
- Why we need TPD?
  - In the heap page we have fixed number of transaction slots which can lead to deadlock.
  - To support cases where a large number of transactions acquire SHARE or KEY SHARE locks on a single page.
- The TPD overflow pages will be stored in the zheap itself, interleaved with regular pages.
- We have a meta page in zheap from which all overflow pages are tracked.
- The idea of putting TPD in heap was of **Andres Freund**

Metapage



TPD page

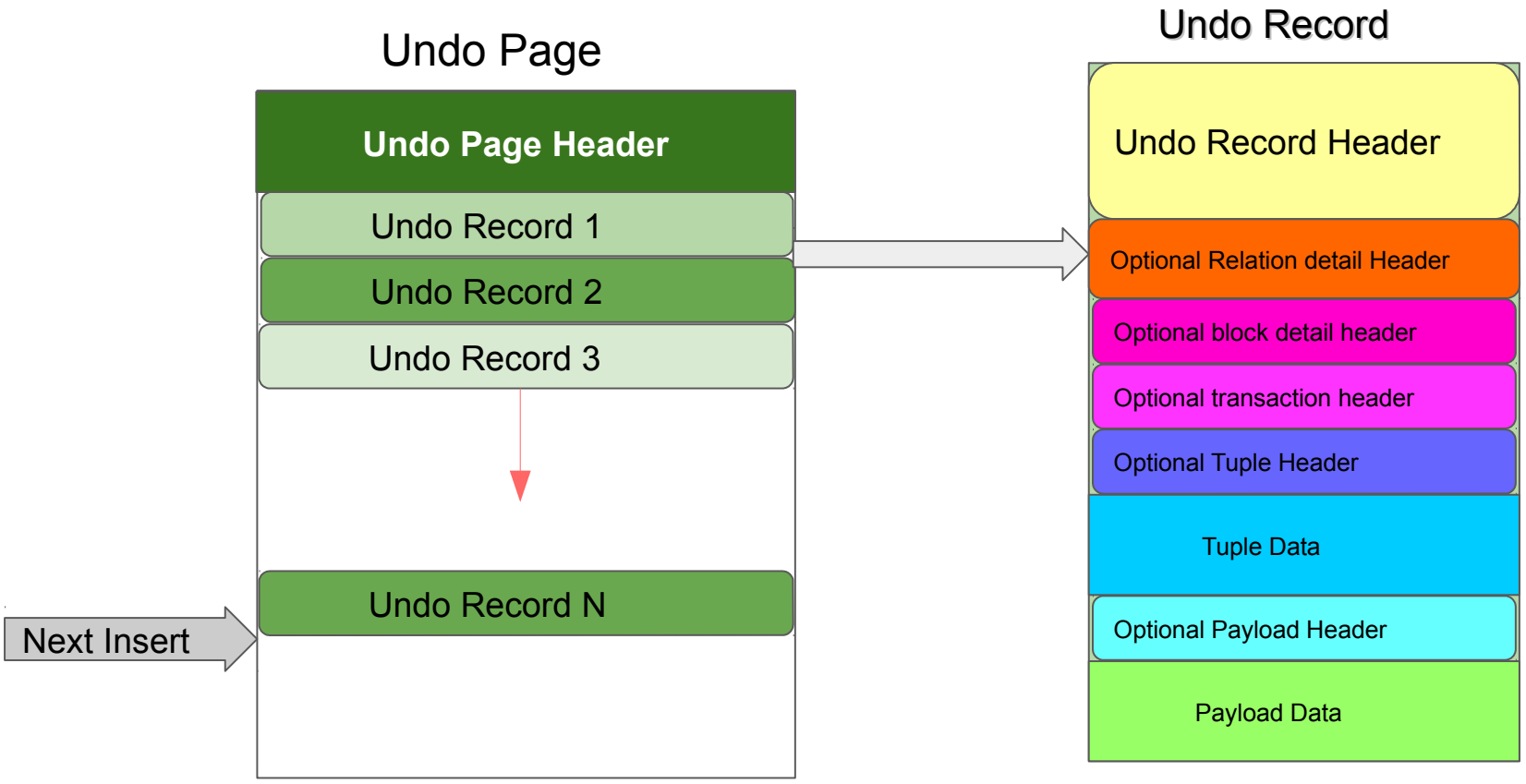


- TPD Entry acts like an extension of the transaction slot array in heap page.
- Tuple headers normally point to the transaction slot responsible for the last modification, but since there aren't enough bits available to do this in the case where a TPD is used, an offset -> slot mapping is stored in the TPD entry itself.

Alternatives for UNDO storage

- Single shared file
  - Insertion point will become point of contention.
  - Difficult to cleanup after transaction aborts.
- Single file per transaction
  - Can create many files.
  - Space wastage for short transactions especially in presence of long running queries during which we can't reclaim the space occupied by files.
- Single file per backend
  - No space wastage.
  - Cleanup can be performed efficiently by keeping the tail and head (insertion) pointer and keep the tail pointer moving and once it reaches insertion point, reclaim the space for file.

UNDO page format



WAL considerations for undo data

- One important consideration is that we don't need to have full page images for data in undo logs (except when data checksums are enabled) as the undo logs are always written serially, so there shouldn't be any torn page issue.
- Unlike heap,
  - we don't need to rely on the existing state of page to perform operation in the undo logs.
  - undo logs doesn't have any operations that move data, like heap page compaction/pruning.

Rollbacks

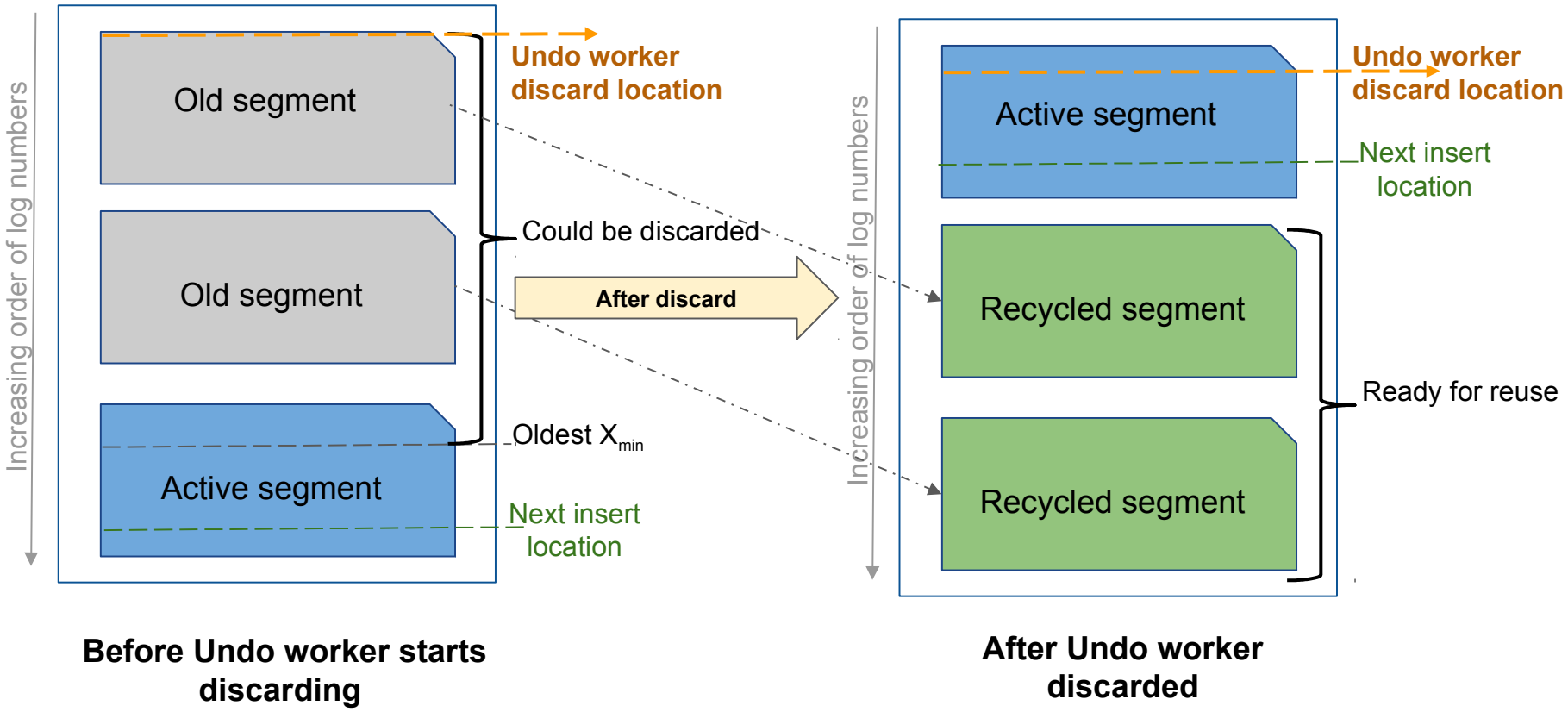
- We need to apply undo actions during Rollback, Rollback To Savepoint and on an error.
- On an error, we apply undo actions in a new transaction.
- We do try to combine and apply the undo actions of a page
  - to cut down the effort for locking-unlocking the page and
  - to reduce the amount of WAL
- If the size of undo for a particular transaction is greater than certain threshold (configurable), then we push the rollback request to background undo worker.

Undo retention

- UNDO data needs to be retained till the active transactions needs to see old versions
  - All transactions which are in-progress
  - For aborted transactions till the time UNDO actions have been performed
  - For committed transactions till the time they are all-visible

- We could reduce the time period for which UNDO needs to be retained in category 3 by implementing “snapshot too old”.
- We consider undo for a transaction to be discardable once its XID is smaller than oldestXmin.

Undolog processing



Undo discard mechanism performed by undo worker

Undolog processing

- The job of discarding the undo logs is performed by undo worker.
- It process all the undo logs.
- Identify the aborted transaction and perform their rollback.
- Process the requests of other backends to perform rollback.
- Discard unwanted undo segments from undo logs.
- Forget all the buffers corresponding to discarded undo to avoid I/O (required to flush those buffers).
- The undo worker mechanism can be extended to multiple undo workers to perform various jobs related to undo logs.
  - For example if there are many pending rollback requests, then we can spawn a new undo worker which can help in processing the requests.

Indexing & zheap

- Current version of zheap works without any changes to index access methods.
- We plan to continue supporting the use of unmodified index access methods with zheap.
- However, if indexes are modified to support “delete-marking”, we could do in-place updates even when indexed columns are modified.
- When performing an in-place update, mark the old index entry as possibly-deleted, and insert a new one. No change to indexes where the columns aren’t modified.
  - Instead of an insert into every index, we incur an insert into only those indexes where there’s a change, plus we delete-mark an entry for each insert.

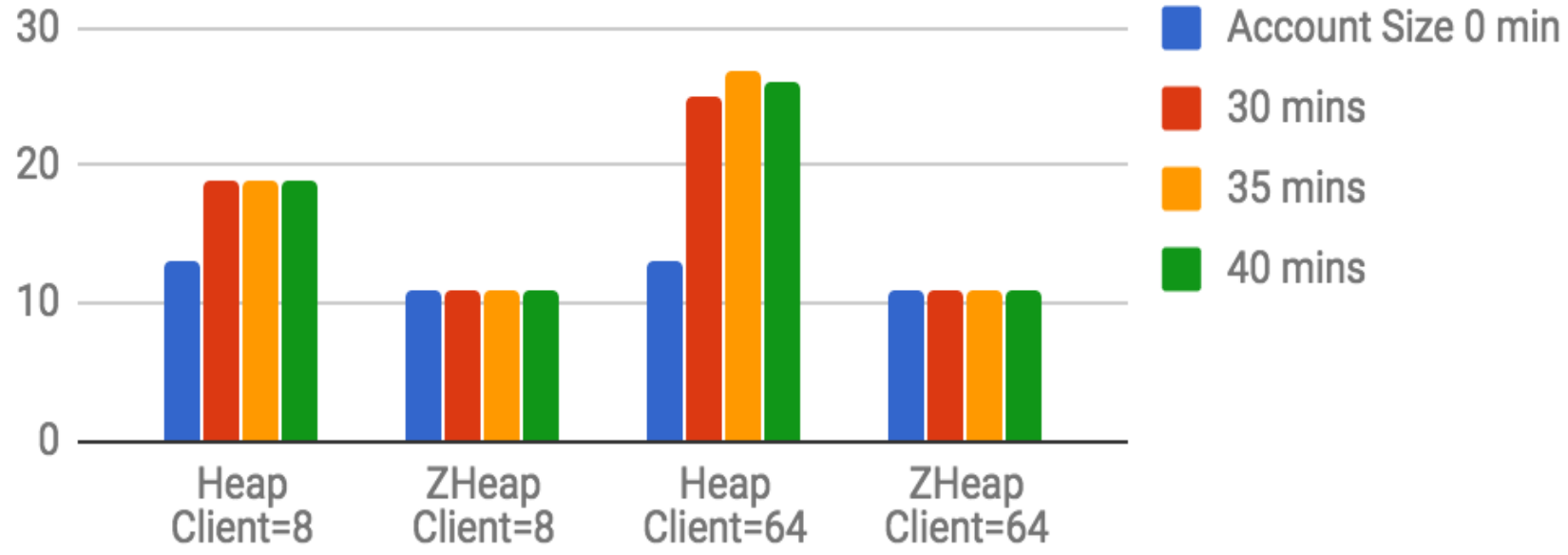
Eliminating VACUUM

- If all indexes on the table support delete-marking, maybe we don't need VACUUM any more.
- Remember, zheap pages don't need to be hinted, frozen, etc. If there are leftover tuples, we can remove them when we want to reuse the space, rather than doing it in advance.
- Delete-marked index tuples can be removed when the pages are scanned, or perhaps when they are evicted from shared buffers. Index pages that are never accessed again might be bloated, but that might not matter very much.
- If we don't VACUUM, we can't ever "lose" free space!
- Could still be an option for users wanting to clean up more aggressively.

Performance data (Test setup)

- Size and TPS comparison of heap and zheap
- We have used pgbench
  - to initialize the data (at scale factor 1000) and
  - then use the simple-update test (which comprises of one-update, one-select, one-insert) to perform updates.
- Machine details: x86_64 architecture, 2-sockets, 14-cores per socket, 2-threads per-core and has 64-GB RAM.
- Non-default parameters: shared_buffers=32GB, min_wal_size=15GB, max_wal_size=20GB, checkpoint_timeout=1200, maintenance_work_mem=1GB, checkpoint_completion_target=0.9, synchoronous_commit = off;

Accounts Size in GB (1 trans open for 30 mins)



- The Initial size of accounts table is 13GB in heap and 11GB in zheap.
- The size in heap grows to 19GB at 8-client count test and to 26GB at 64-client count test.
- The size in zheap remains at 11GB for both the client-counts at the end of test.
- All the undo generated during test gets discarded within a few seconds after the open transaction is ended.
- The TPS of zheap is ~40% more than heap in above tests at 8 client-count. In some other high-end machines, we have seen up to ~100% improvement for similar test.

Benefits

- Performing updates in-place wherever possible prevents bloat from being created.
- Old tuple versions are removed eagerly from the heap (as soon as the transaction ends).
- Most things that could cause a page to be rewritten multiple times are eliminated. Tuples no longer need to be frozen; instead, pages are implicitly frozen by the removal of associated UNDO.
- Because zheap is smaller on-disk, we get a small performance boost.
- In workloads where the heap bloats and zheap only bloats the undo, we get a massive performance boost.

Drawbacks

- Reading a page will be more expensive when there are active transactions operating on a page.
- Delete marking will have some overhead, but we will still win if there are many indexes on the table and only few of them got updated.
- Transaction abort can be lengthy.

Who?

- Amit Kapila (development lead)
- Dilip Kumar
- Kuntal Ghosh
- Mithun CY
- Ashutosh Sharma
- Rafia Sabih
- Beena Emerson
- Amit Khandekar
- Thomas Munro
- Neha Sharma

Thanks!