Zedstore

Columnar storage for PostgreSQL

Alexandra Wang, Soumyadeep Chakraborty
VMware Greenplum

Agenda

- Goals
- Design internals with demo
- Performance
- Open areas of work
- Get involved!

Goals

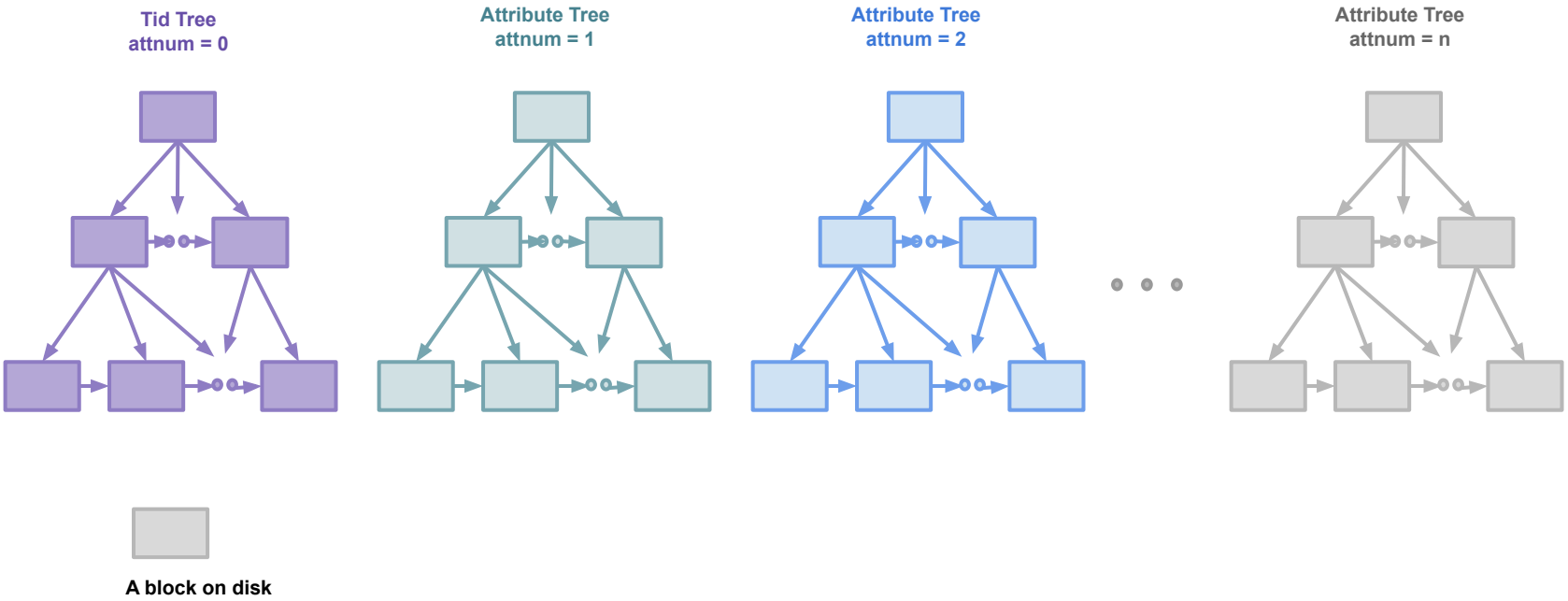
- A column store that every Postgres user can use
- Fully leverage the on-disk adjacency of column-wise storage
  - Efficient and extensive compression
  - Queries on subsets of columns should be fast
- Optimized for OLAP workloads and bulk data ingestion
- Reasonable OLTP performance and feature parity with heap
- Fully MVCC, crash-safe and supports replication

Design

Design

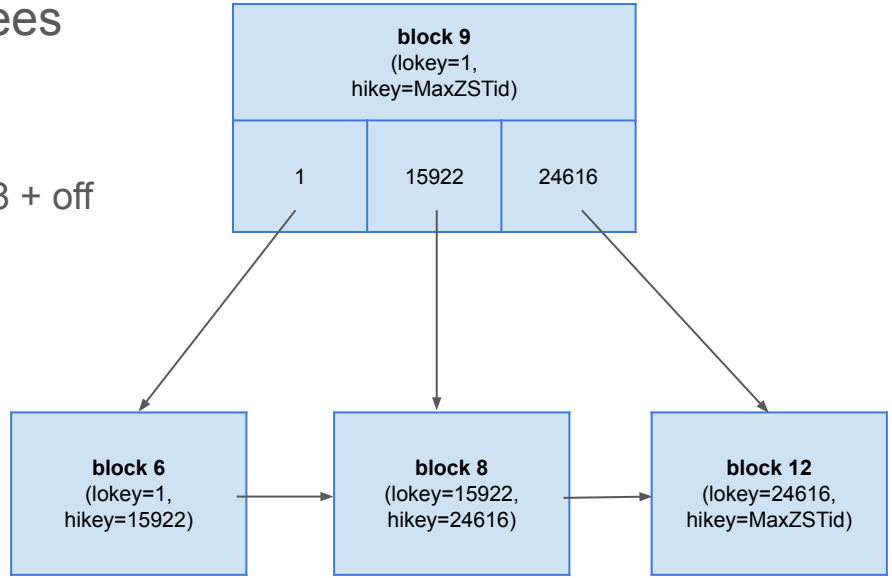
- Leverages the Table AM API
- Leverages PostgreSQL infrastructure
  - 8KB (BLCKSZ) fixed block size
  - Buffer manager
  - WAL logging - with custom WAL records
- Each column is a B-Tree

Forest of B-Trees



ZSTid & B-Tree pages

- 64-bit tuple identifier, key of the B-Trees
- 1-1 mapping with ItemPointer:
  - ZSTidFromItemPointer(blk, off) = blk * 128 + off
  - Only 48 bits is used
- Purely logical
  - Does not tie tuple to physical location
  - A tuple's tid never changes



Demo

```
postgres=# create table foo(i int, j text) using zedstore; -- create a zedstore table
CREATE TABLE
postgres=# insert into foo select i, repeat('a', i/100) from generate_series(1, 30000)i;
INSERT 0 30000
postgres=# set default_table_access_method = 'zedstore'; -- we can also set this guc for the entire session
SET
postgres=# select ctid, ctid::zstid as zstid, i from foo where ctid::zstid::int8 % 128 = 1 limit 2; -- ctid to zstid mapping
 ctid  | zstid |  i
-------+-------+-----
 (0,1) | 1     |   1
 (1,1) | 129   | 129
(2 rows)

postgres=# select * from pg_zs_btree_pages('foo') where attno = 2; -- btree pages for column j
 blkno |  nextblk   | attno | level | lokey |      hikey      | nitems | ncompressed | totalsz | uncompressedsz | freespace
-------+------------+-------+-------+-------+-----------------+--------+-------------+---------+----------------+-----------
     6 |          8 |     2 |     0 |     1 |           15922 |      1 |           1 |    8136 |        1306398 |         0
     8 |         12 |     2 |     0 | 15922 |           24616 |      1 |           1 |    8136 |        1827390 |         0
     9 | 4294967295 |     2 |     1 |     1 | 281474976645120 |      3 |             |         |                |      8088
    12 | 4294967295 |     2 |     0 | 24616 | 281474976645120 |      1 |           1 |    6553 |        1510944 |      1583
(4 rows)
```

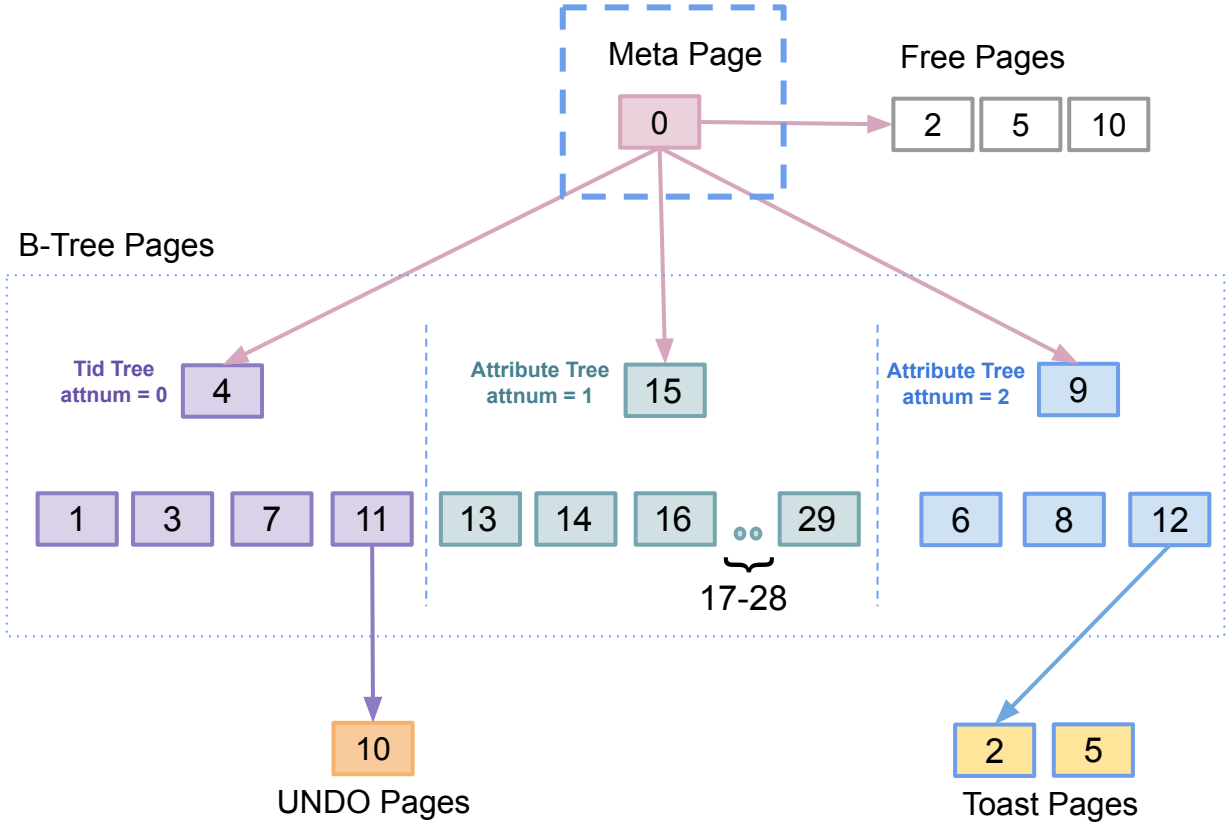
Demo

```
postgres=# select * from pg_zs_page_type('foo', 0); -- page type of block 0
 pg_zs_page_type
-----------------
 META
(1 row)

postgres=# select count(*), pg_zs_page_type('foo', blkno) as page_type
postgres-# from generate_series(0, pg_table_size('foo') / 8192 - 1) blkno
postgres-# group by page_type; -- counts for each page type
 count | page_type
-------+-----------
     1 | META
    26 | BTREE
     3 | FREE
(3 rows)

postgres=# insert into foo values (1000000, repeat('x', 1000000)) ; -- insert an oversized datum
INSERT 0 1
postgres=# select count(*), pg_zs_page_type('foo', blkno) as page_type
postgres-# from generate_series(0, pg_table_size('foo') / 8192 - 1) blkno
postgres-# group by page_type; -- counts for each page type
 count | page_type
-------+-----------
     1 | META
    26 | BTREE
     1 | UNDO
     2 | TOAST
(4 rows)
```

Page types

Blocks layout for table *foo*

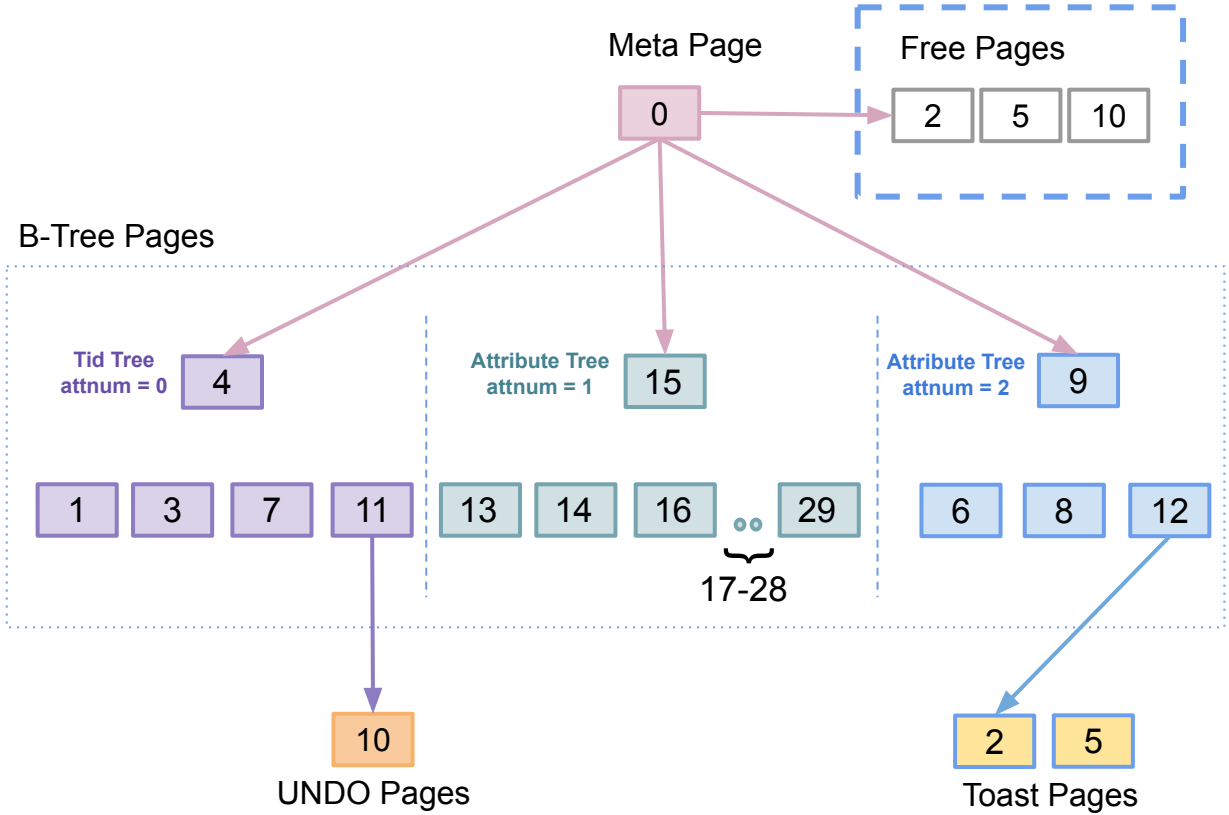
0	1	2	3
4	5	6	7
8	9	10	11
12	13	14	15
16	17	18	19
20	21	22	23
24	25	26	27
28	29		



Page types

Blocks layout for table *foo*

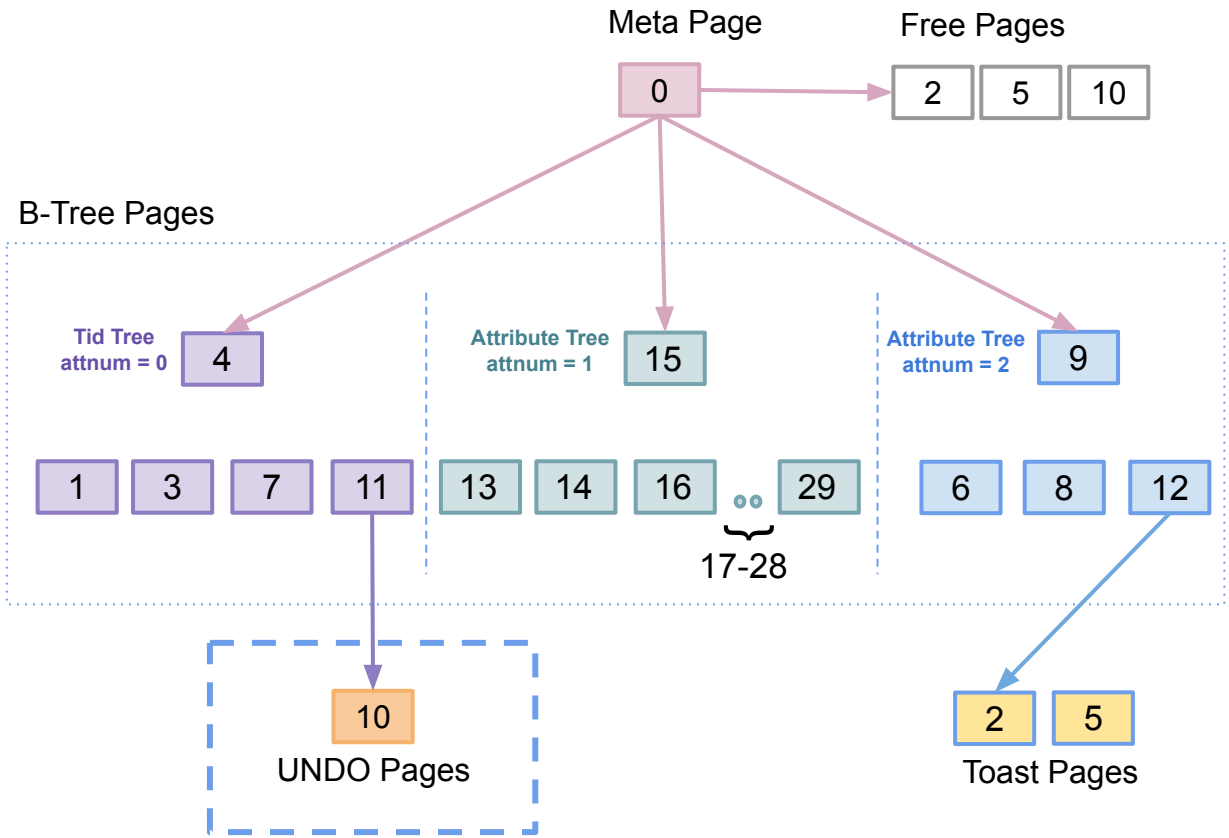
0	1	2	3
4	5	6	7
8	9	10	11
12	13	14	15
16	17	18	19
20	21	22	23
24	25	26	27
28	29		



Page types

Blocks layout for table *foo*

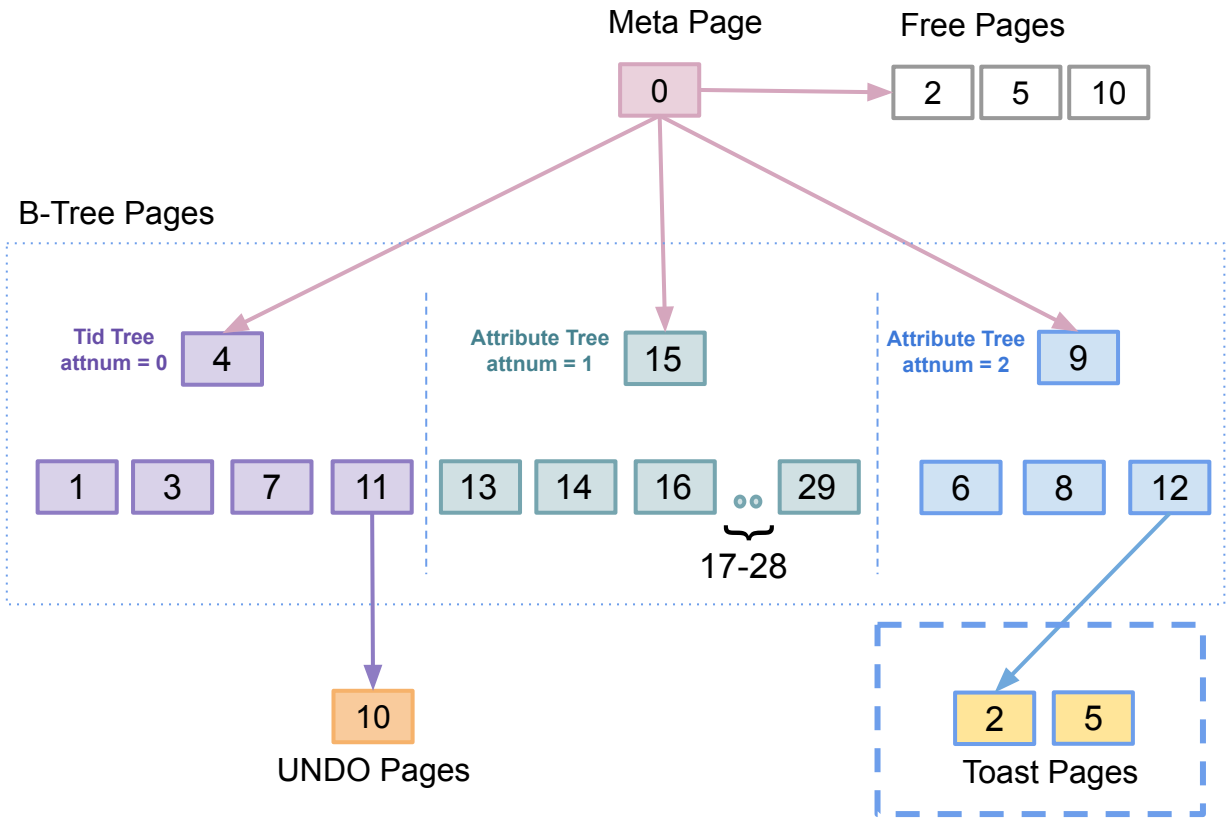
0	1	2	3
4	5	6	7
8	9	10	11
12	13	14	15
16	17	18	19
20	21	22	23
24	25	26	27
28	29		



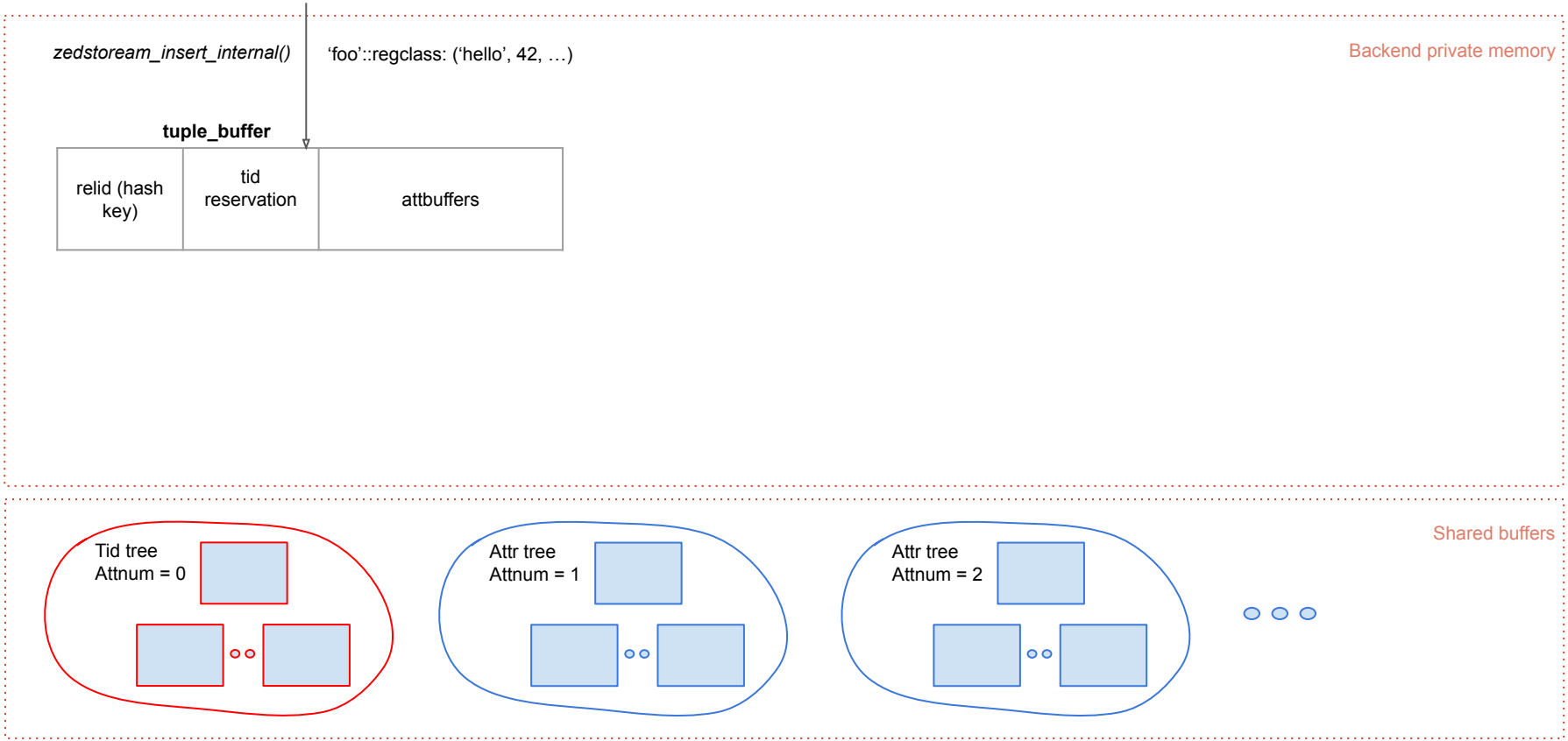
Page types

Blocks layout for table *foo*

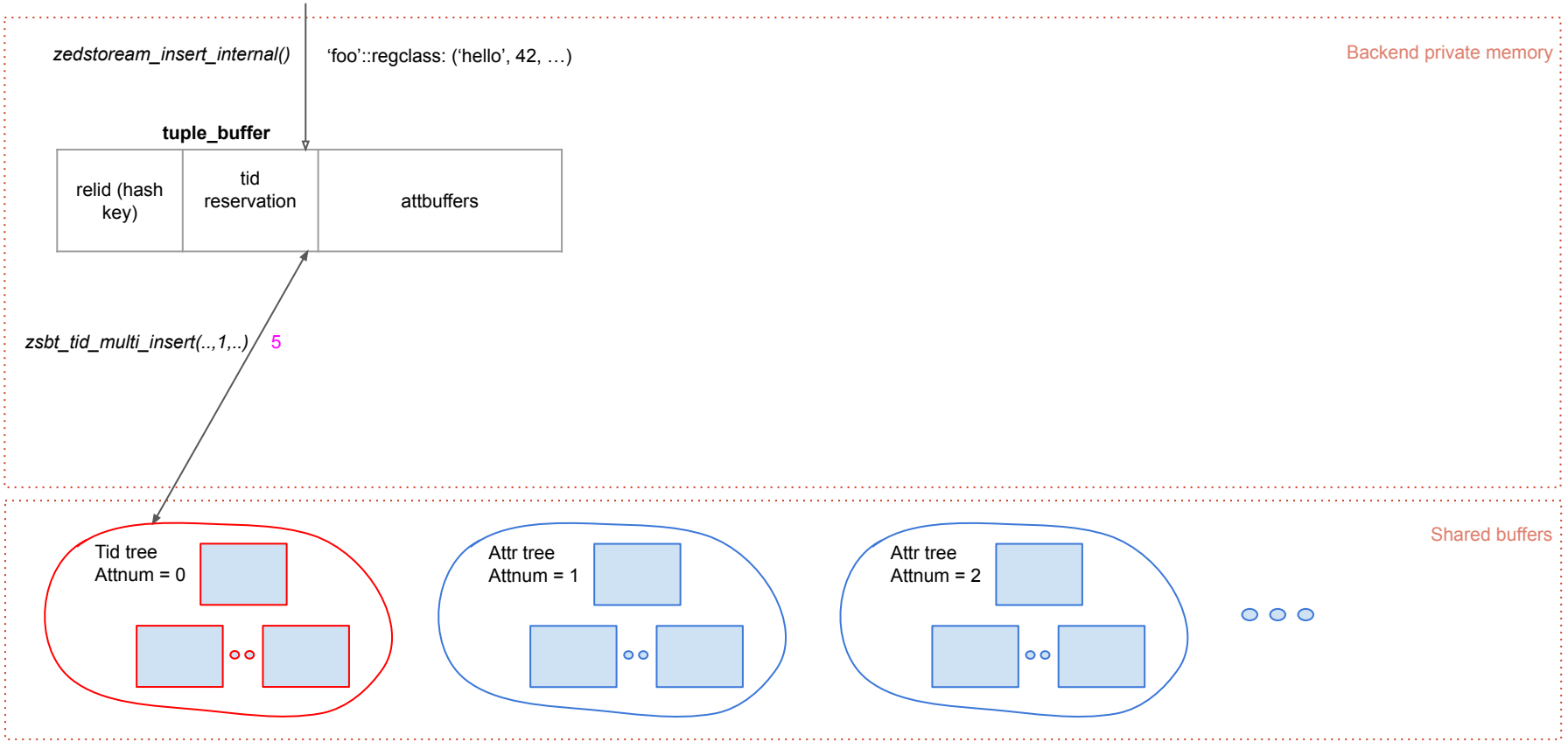
0	1	2	3
4	5	6	7
8	9	10	11
12	13	14	15
16	17	18	19
20	21	22	23
24	25	26	27
28	29		



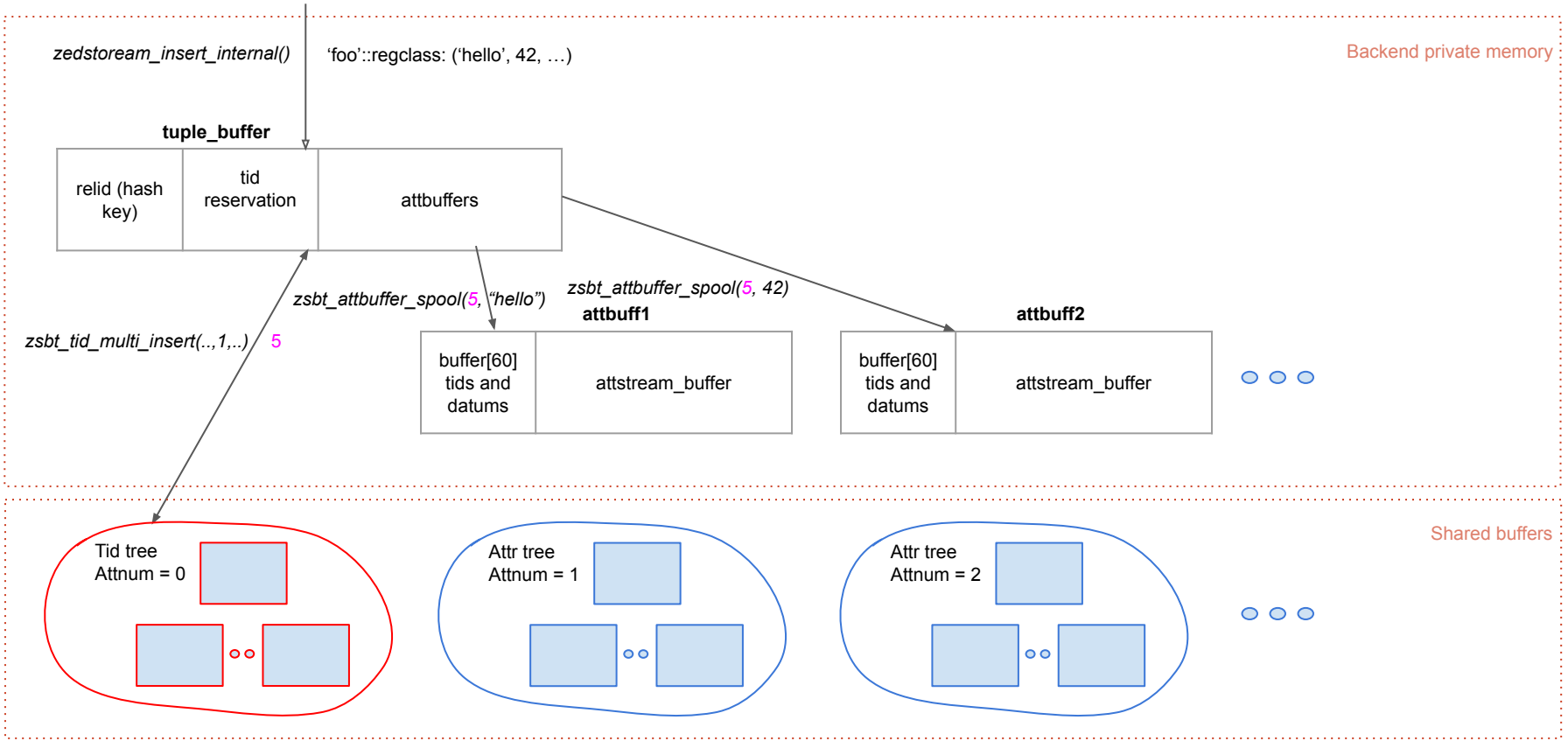
INSERT INTO foo VALUES(‘hello’, 42, …)



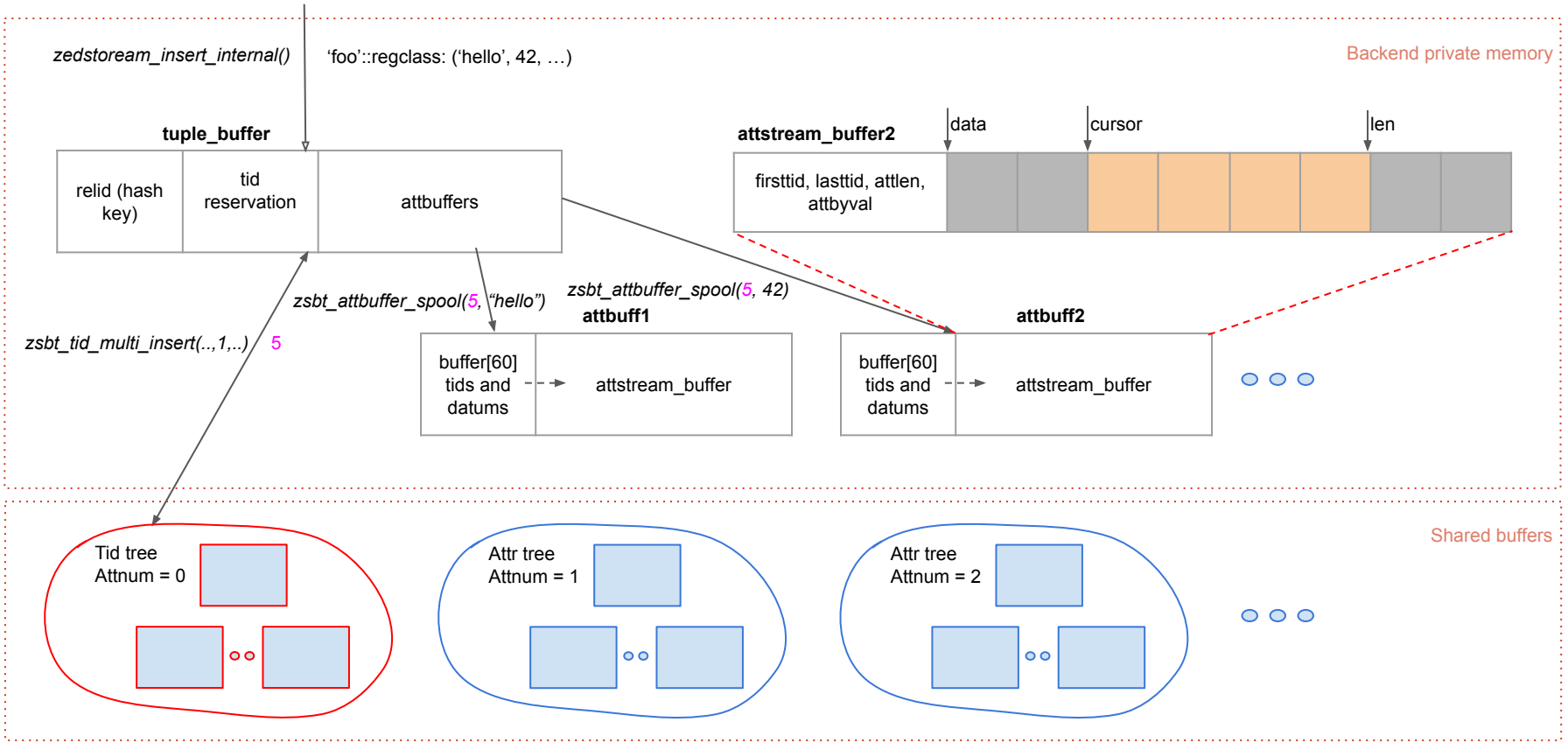
INSERT INTO foo VALUES('hello', 42, …)



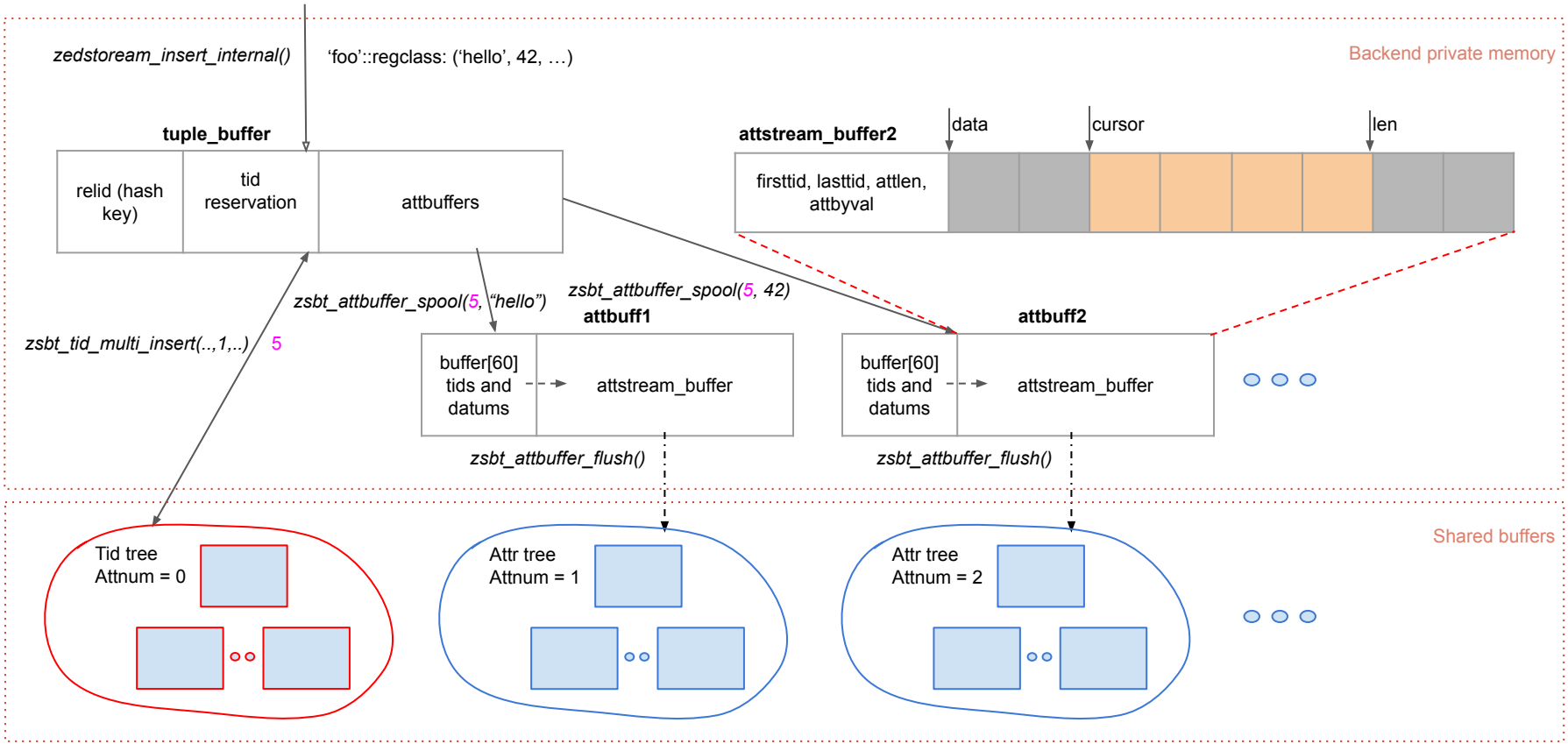
INSERT INTO foo VALUES('hello', 42, …)



INSERT INTO foo VALUES('hello', 42, …)



INSERT INTO foo VALUES('hello', 42, …)



SELECT i, j FROM foo

zedstoream_beginscan(...)

zedstoream_beginscan_with_column_projection(...)

zedstoream_getnextslot(...)

zsbt_tid_begin_scan(...)

zsbt_attr_begin_scan(...)

zsbt_tid_scan_next(...)

For all attributes in col list:

zsbt_attr_fetch(...)

/ fill in visibility info */*

return slot

SELECT i, j FROM foo

zedstoream_beginscan(...)

zedstoream_beginscan_with_column_projection(...)

zedstoream_getnextslot(...)

zsbt_tid_begin_scan(...)

zsbt_attr_begin_scan(...)

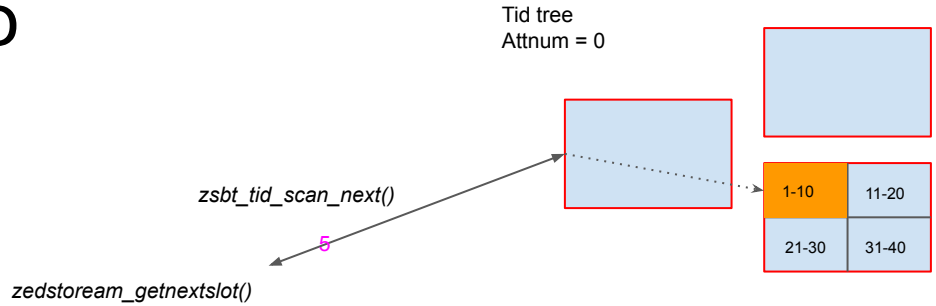
zsbt_tid_scan_next(...)

For all attributes in col list:

zsbt_attr_fetch(...)

/ fill in visibility info */*

return slot



SELECT i, j FROM foo

zedstoream_beginscan(...)

zedstoream_beginscan_with_column_projection(...)

zedstoream_getnextslot(...)

zsbt_tid_begin_scan(...)

zsbt_attr_begin_scan(...)

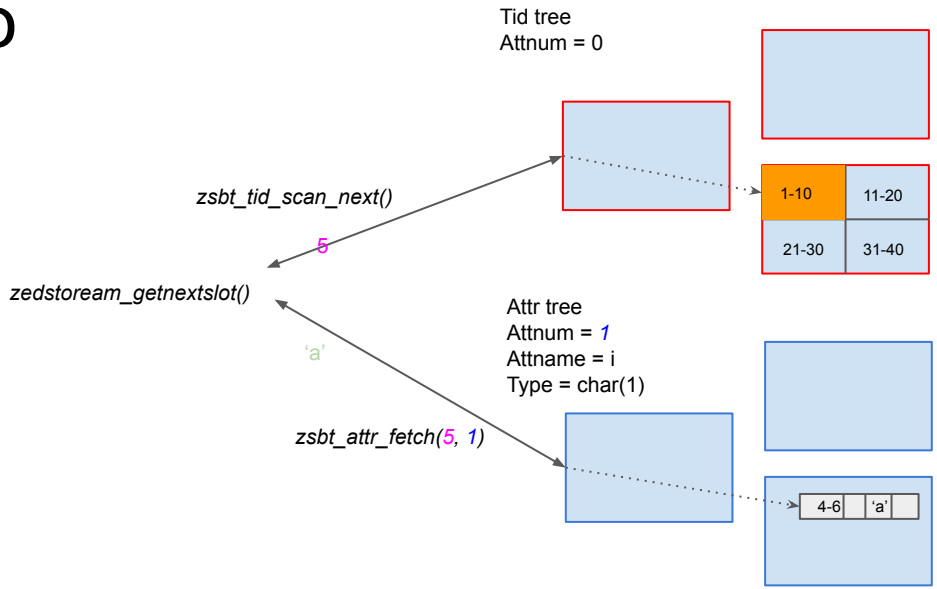
zsbt_tid_scan_next(...)

For all attributes in col list:

zsbt_attr_fetch(...)

/ fill in visibility info */*

return slot



SELECT i, j FROM foo

zedstoream_beginscan(...)

zedstoream_beginscan_with_column_projection(...)

zedstoream_getnextslot(...)

zsbt_tid_begin_scan(...)

zsbt_attr_begin_scan(...)

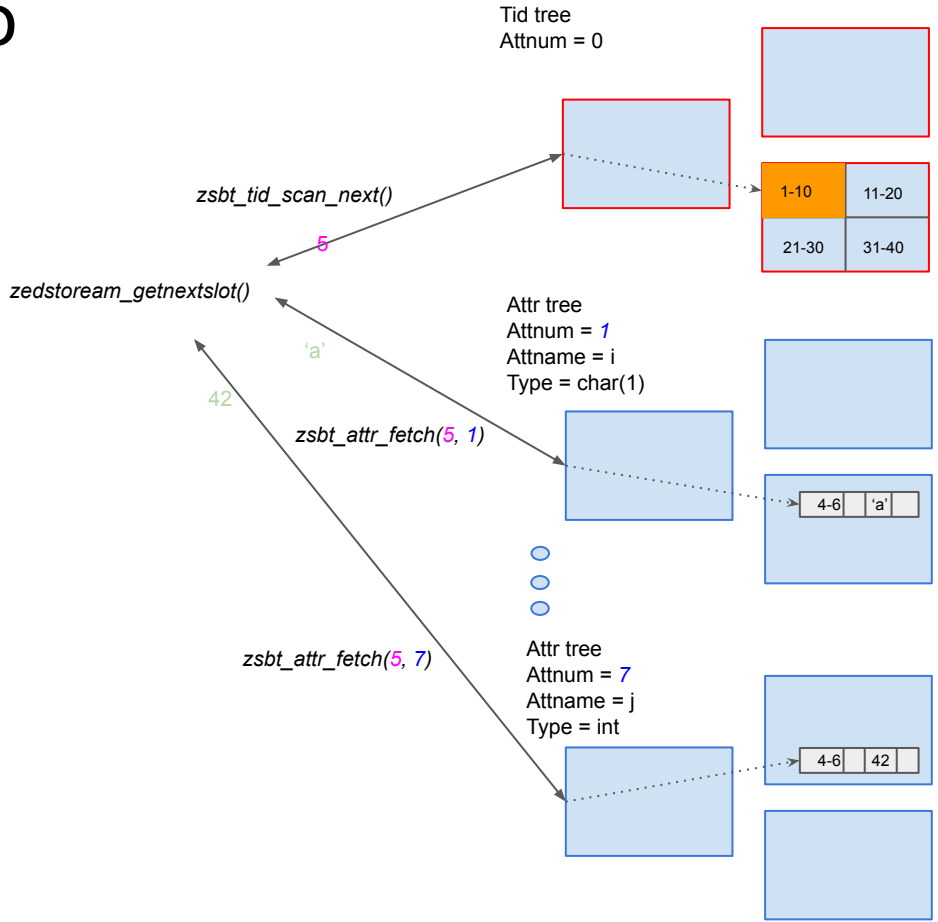
zsbt_tid_scan_next(...)

For all attributes in col list:

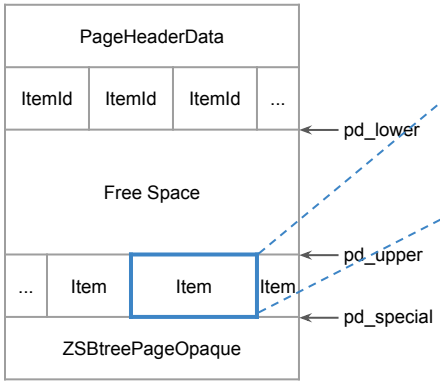
zsbt_attr_fetch(...)

/ fill in visibility info */*

return slot



TID tree leaf page layout



Tid leaf page

ZSTidArrayItem

Header - t_firsttid, t_lasttid - Sizes of other sections - size of item itself	**TIDs:** - 64 bit simple-8b codeword - Delta encoded - Typically 1-10 bits per TID	**UNDO slots** - 0-2 unique UNDO pointers to UNDO pages	**UNDO slotwords** - 64 bit ints packed with each TID's UNDO pointer - 2 bits per TID

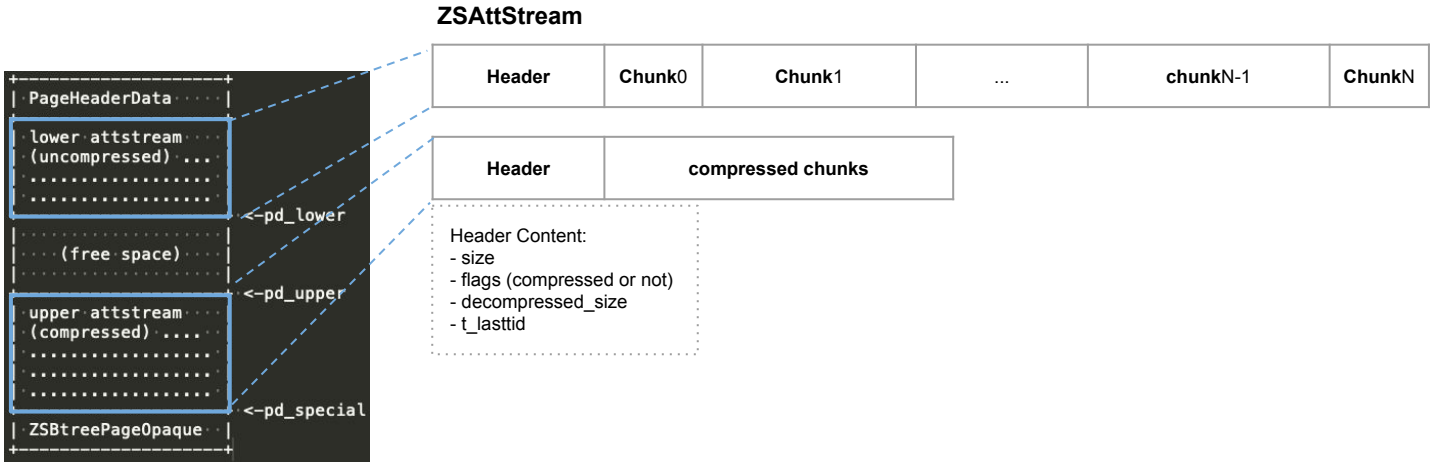
Logical content:

TID | UNDO pointer
119 | ALL_VISIBLE
120 | 7599293
121 | DEAD
123 | 7599293
124 | 8000222
125 | 8000222

Physical content:

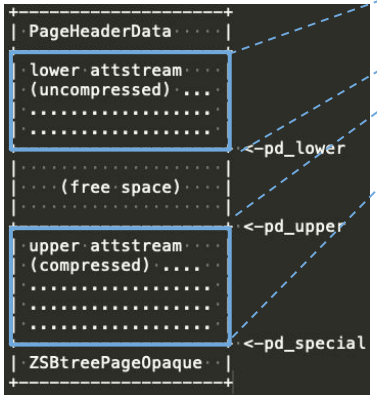
TIDs | UNDO pointers | UNDO indexes
119, 1, 1, 2, 1, 1 | 7599293, 8000222 | V, 0, D, 0, 1, 1

Attribute tree leaf page layout



Attr leaf page

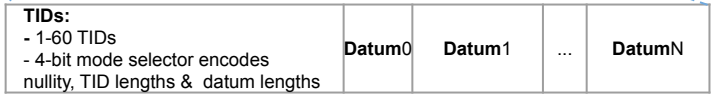
Attribute tree leaf page layout



Attr leaf page

ZSAttStream

Header Content:
- size
- flags (compressed or not)
- decompressed_size
- t_lasttid



Logical content:

TID | column values
31 | 31
32 | 32
33 | 33
34 | 34
...
60 | 60

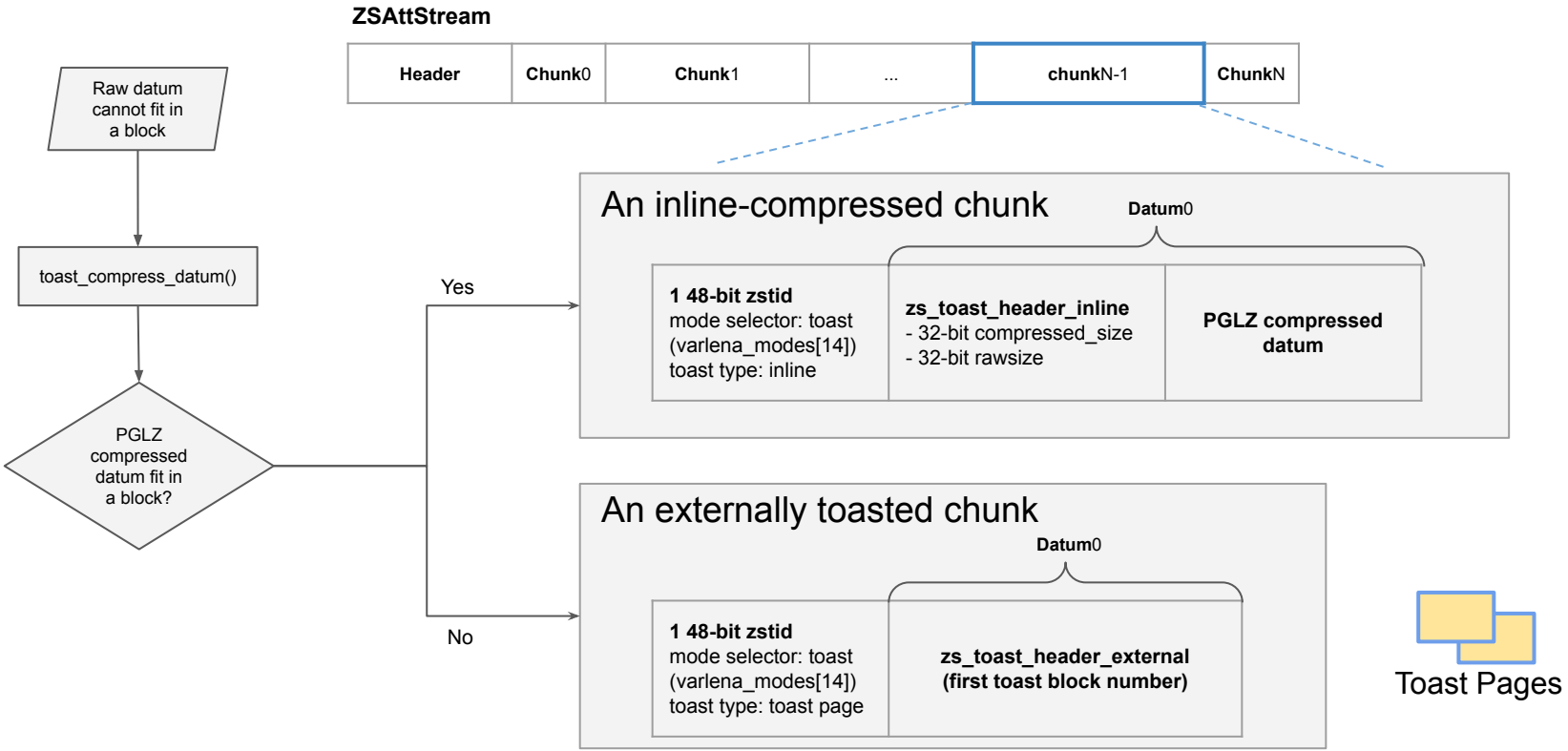
Physical content:

TIDs | column values
31, 1, 1, 1,..., 1 | 31, 32, 33, 34, … 60

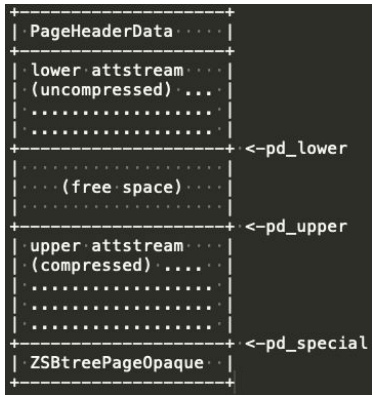
Demo

```
postgres=# truncate foo;  insert into foo select i, repeat('x', 100000) from generate_series(1, 100) i;
TRUNCATE TABLE
INSERT 0 100
postgres=# select * from pg_zs_btree_pages('foo') where level = 0; -- show all leaf b-tree pages
 blkno |  nextblk   | attno | level | lokey |      hikey      | nitems | ncompressed | totalsz | uncompressedsz | freespace
-------+------------+-------+-------+-------+-----------------+--------+-------------+---------+----------------+-----------
     1 | 4294967295 |     0 |     0 |     1 | 281474976645120 |      3 |           0 |     264 |            264 |      7860
     3 | 4294967295 |     1 |     0 |     1 | 281474976645120 |      1 |           0 |     456 |            456 |      7680
     4 | 4294967295 |     2 |     0 |     1 | 281474976645120 |      1 |           1 |     526 |         116400 |      7610
(3 rows)

postgres=# select chunkno, upperstream, compressed, chunk_start, chunk_len, firsttid, lasttid, itemcount, substring(chunk, 1, 1
0) as chunk_truncated
postgres-# from pg_zs_dump_attstreams('foo', 3); -- show all chunks for block 3
 chunkno | upperstream | compressed | chunk_start | chunk_len | firsttid | lasttid | itemcount |    chunk_truncated
---------+-------------+------------+-------------+-----------+----------+---------+-----------+------------------------
       0 | f           | f          |           0 |       128 | 1        | 30      |        30 | \xffffff3f000000100100
       1 | f           | f          |         128 |       128 | 31       | 60      |        30 | \xffffff3f000000101f00
       2 | f           | f          |         256 |       128 | 61       | 90      |        30 | \xffffff3f000000103d00
       3 | f           | f          |         384 |        48 | 91       | 100     |        10 | \x21841042082100905b00
(4 rows)

postgres=# \x
Expanded display is on.
postgres=# with chunk0 as
postgres-# (select attbyval, attlen, prevtid, firsttid, lasttid, chunk from pg_zs_dump_attstreams('foo'::regclass, 3) where chu
nkno=0)
postgres-# select (pg_zs_decode_chunk(attbyval, attlen, prevtid, lasttid, chunk)).* from chunk0; -- decode chunk0 for block 3
-[ RECORD 1 ]+-------------------------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------------------------------------------------
-------------------------------------------------------
num_elements | 30
tids         | {1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,26,27,28,29,30}
datums       | {"\\x01000000","\\x02000000","\\x03000000","\\x04000000","\\x05000000","\\x06000000","\\x07000000","\\x08000000"
,"\\x09000000","\\x0a000000","\\x0b000000","\\x0c000000","\\x0d000000","\\x0e000000","\\x0f000000","\\x10000000","\\x11000000",
"\\x12000000","\\x13000000","\\x14000000","\\x15000000","\\x16000000","\\x17000000","\\x18000000","\\x19000000","\\x1a000000","
\\x1b000000","\\x1c000000","\\x1d000000","\\x1e000000"}
isnulls      | {f,f,f,f,f,f,f,f,f,f,f,f,f,f,f,f,f,f,f,f,f,f,f,f,f,f,f,f,f,f}
```

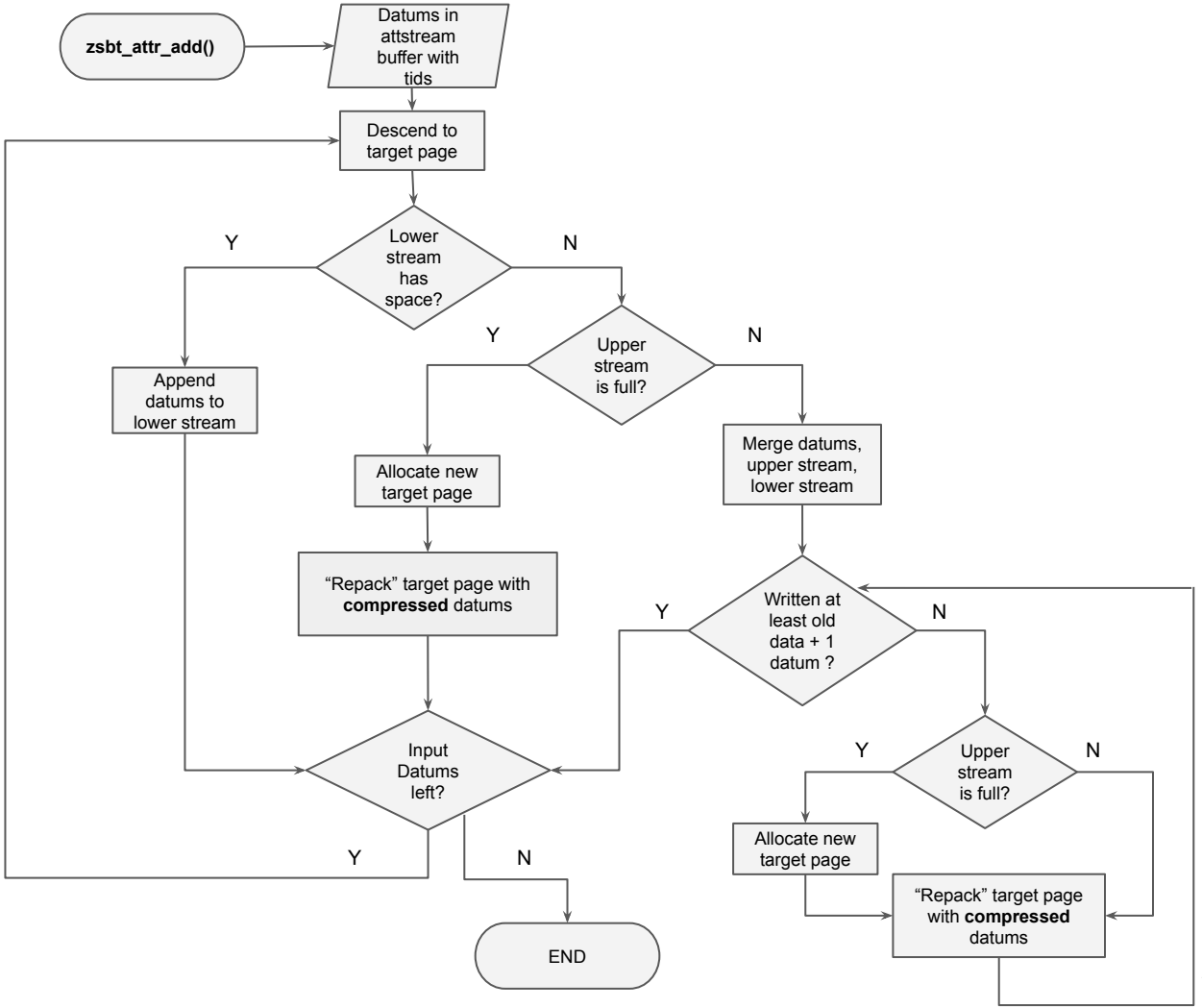
Attribute tree leaf w/ oversized datum



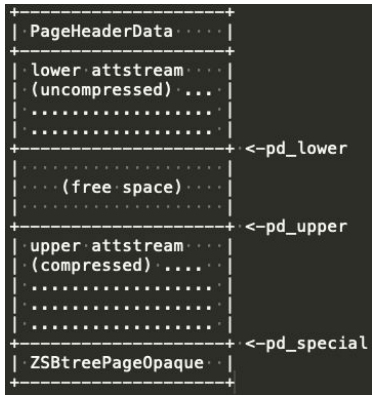
Att leaf packing



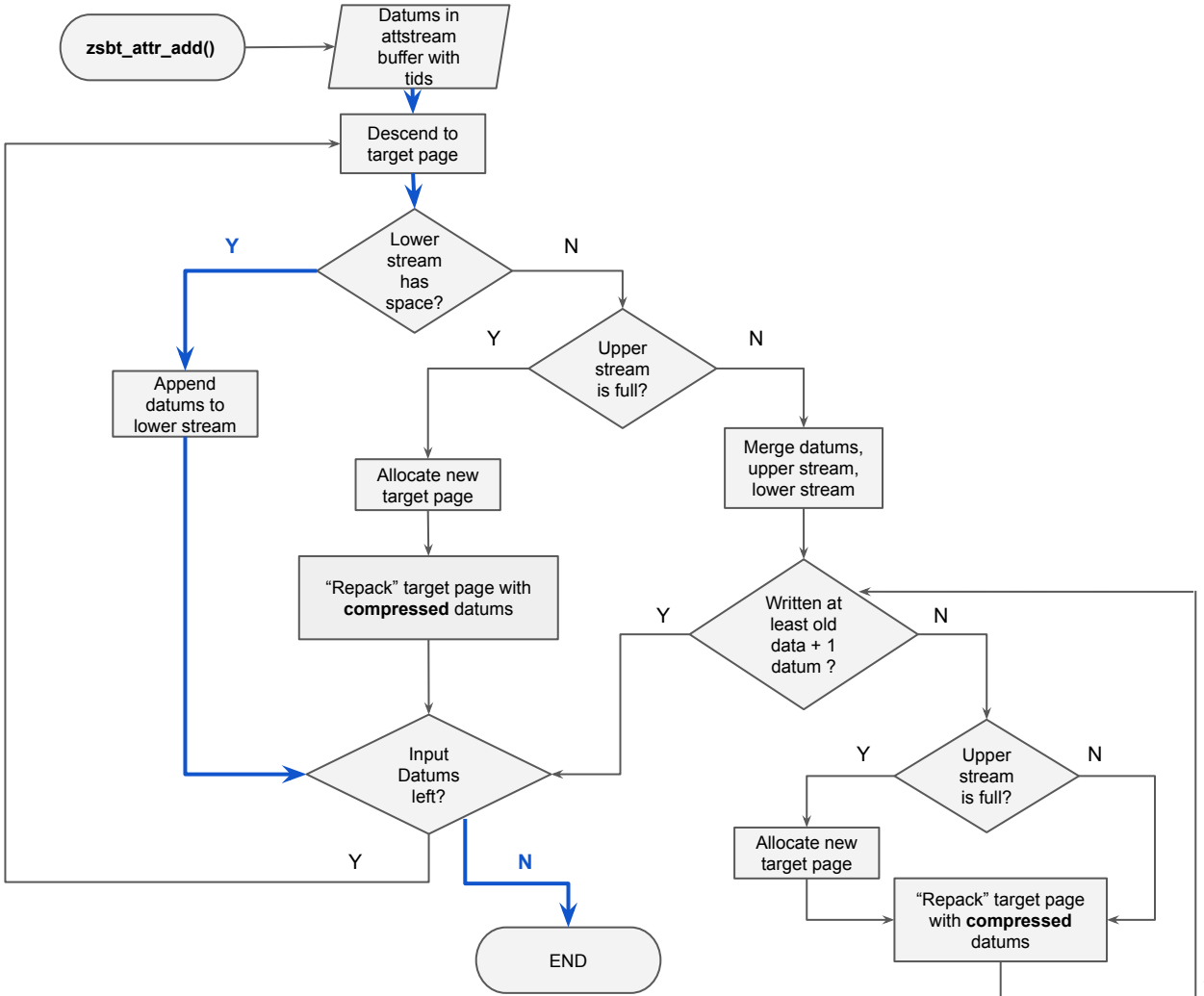
Attr leaf



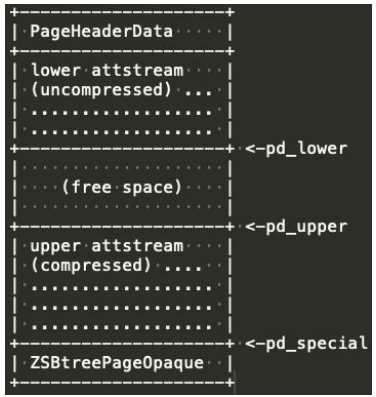
Att leaf packing



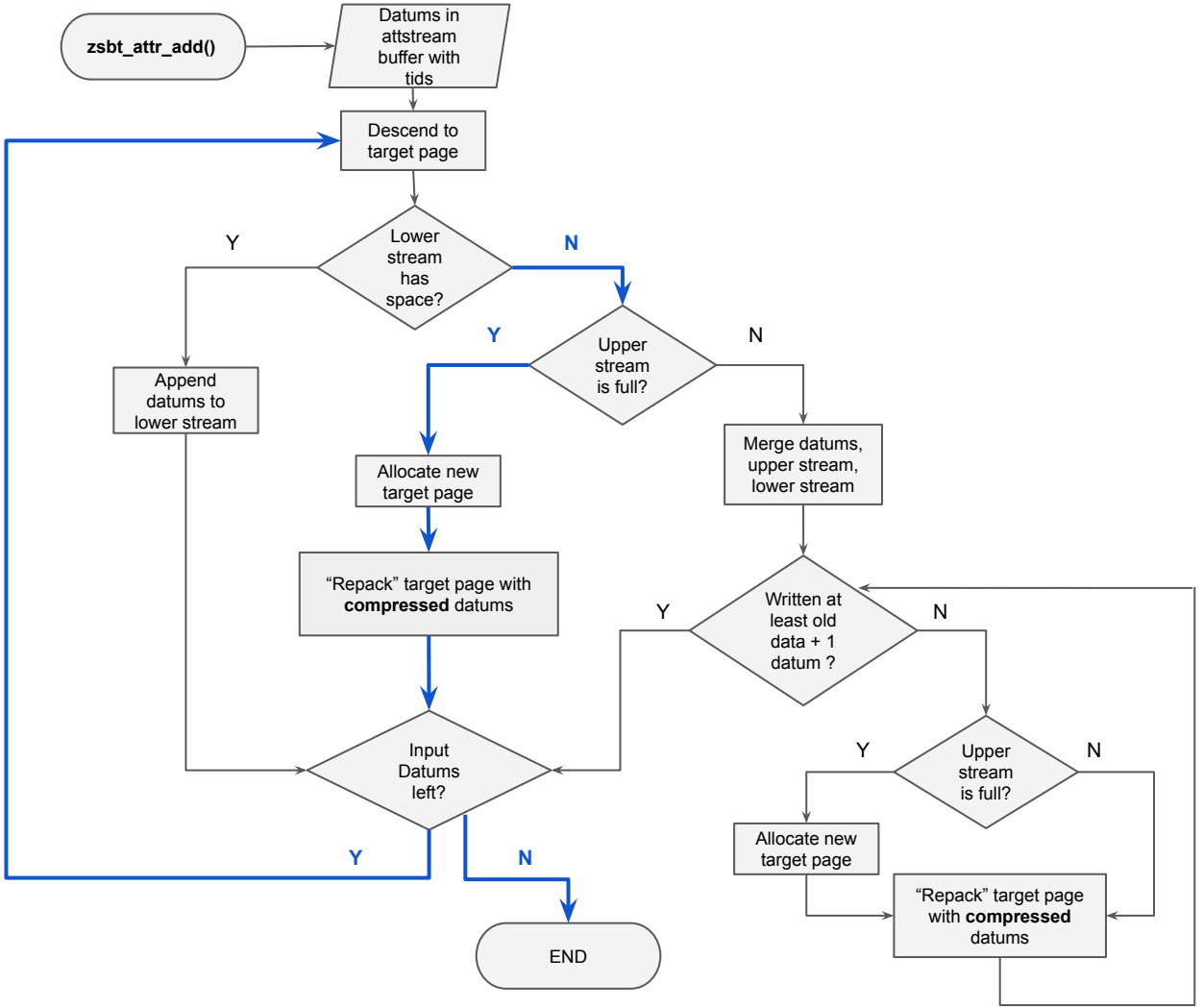
Attr leaf



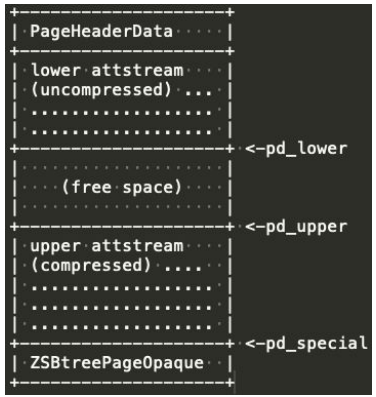
Att leaf packing



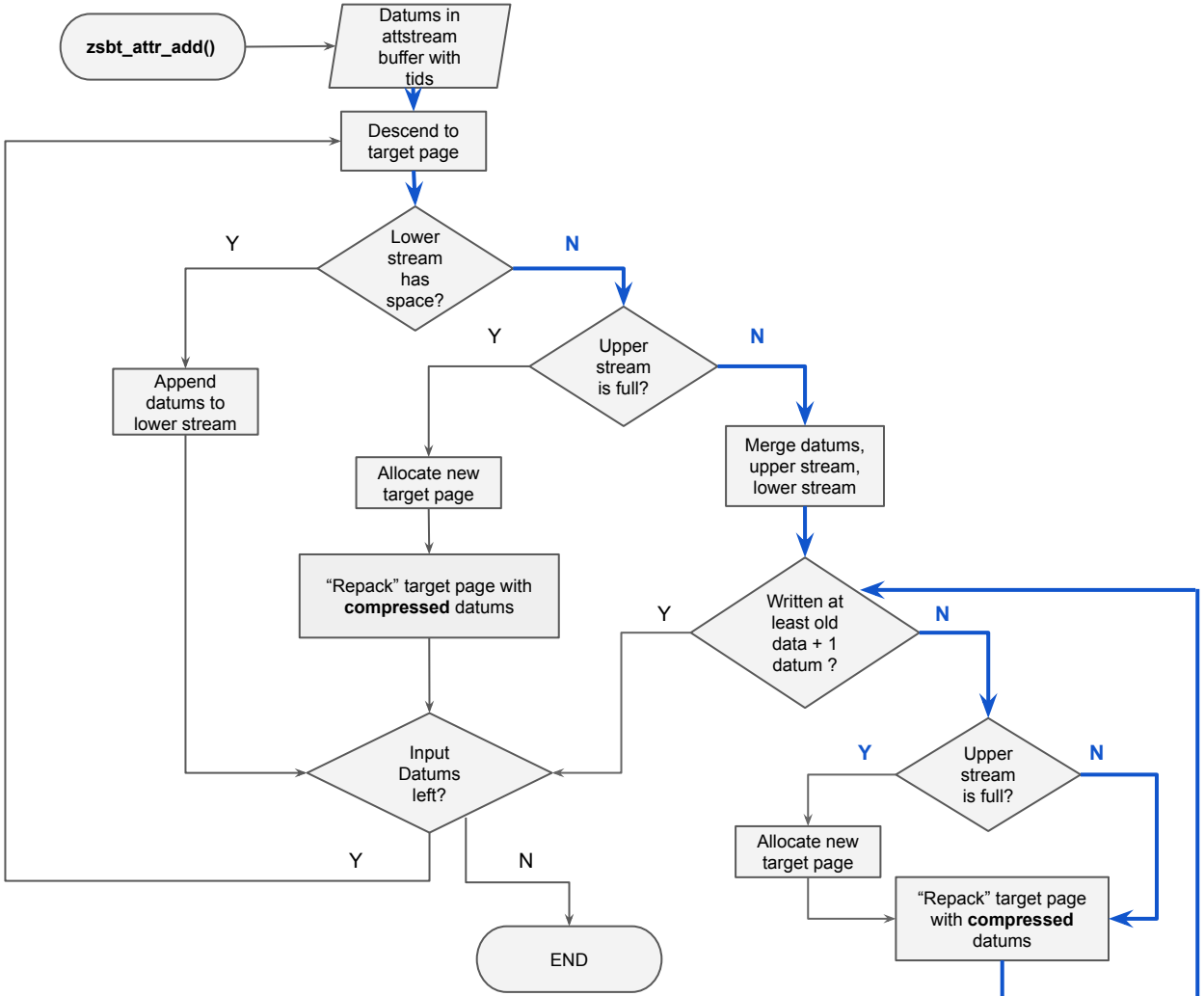
Attr leaf



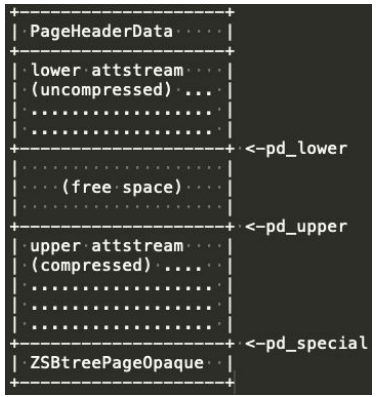
Att leaf packing



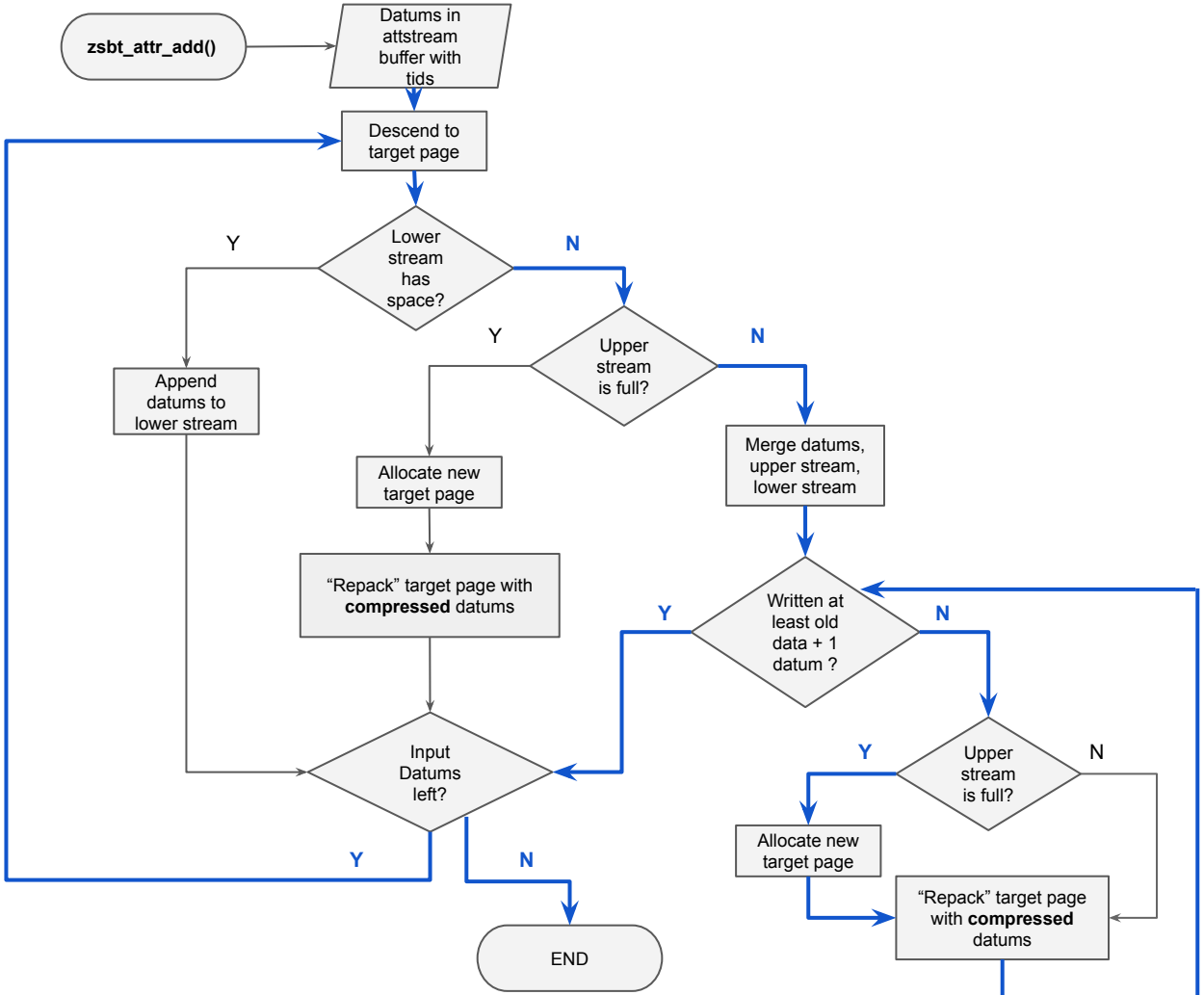
Attr leaf



Att leaf packing



Attr leaf



Performance

On-disk footprint and COPY performance

Table	Heap	Zed (N=1, P=1)	Zed (N=1, P=16)	Zed (N=10, P=16) (new default)	Zed (N=100, P=16)	Zed (N=1000, P=16)
catalog_sales	15.0G	8.0G	15.0G	9.1G	7.7G	7.5G
catalog_returns	1.2G	0.8G	1.5G	0.9G	0.7G	0.7G
store_returns	1.9G	1.2G	2.1G	1.2G	1.1G	1.1G
store_sales	21.0G	10.0G	17.0G	11G	10.1G	10.1G
COPY runtime (P=16)	8min	----------	100min	30min	10min	7min

★ Issue: Bloat from inefficient page splits due to out-of-order tid inserts from parallel COPYs
★ Fix: reserve tids in batches of N for multi-insert (COPYs)
★ N = MULTI_INSERT_TID_RESERVATION_FACTOR
★ P = # concurrent COPY commands loading data into the same table.
★ COPY runtime applies to loading the data for all of the tables mentioned here.

Single column projection scale test

```
CREATE TABLE tpcds.store_sales (
    ss_sold_date_sk integer,
    ss_sold_time_sk integer,
    ss_item_sk int NOT NULL,
    ss_customer_sk integer,
    ss_cdemo_sk integer,
    ss_hdemo_sk integer,
    ss_addr_sk integer,
    ss_store_sk integer,
    ss_promo_sk integer,
    ss_ticket_number bigint NOT NULL,
    ss_quantity integer,
    ss_wholesale_cost numeric(7,2),
    ss_list_price numeric(7,2),
    ss_sales_price numeric(7,2),
    ss_ext_discount_amt numeric(7,2),
    ss_ext_sales_price numeric(7,2),
    ss_ext_wholesale_cost numeric(7,2),
    ss_ext_list_price numeric(7,2),
    ss_ext_tax numeric(7,2),
    ss_coupon_amt numeric(7,2),
    ss_net_paid numeric(7,2),
    ss_net_paid_inc_tax numeric(7,2),
    ss_net_profit numeric(7,2)
)
```

Experiment Parameters:

- **--with-lz4**; Opt level: -O2
- Table: **store_sales**; **TPC-DS scale: 270 (~102GB raw data)**;
- Disk: **NVMe SSD & Rotational hard disk**
- Data loading method: **serial & concurrent COPY**
- GUCs set in database:
  shared_buffers: 10GB; max_wal_size: 1GB; checkpoint_flush_after: 1MB;
  max_parallel_workers: 8; max_parallel_maintenance_workers: 8;
  maintenance_work_mem: 4GB; log_statement: all; effective_cache_size: 32GB;
  track_io_timing: on # this is important, it shows explain analyze I/O timings

EXPLAIN (ANALYZE, BUFFERS, TIMING, VERBOSE)
SELECT ss_sold_date_sk FROM store_sales;
Measurement: I/O read time reported by the “I/O Timings” field

For cold results, before every explain analyze run, we **restart the database** to flush the buffers and **clear the OS page cache**.

* Note: EXPLAIN ANALYZE and track_io_timing introduces significant timing overhead

Results w/o optimization (cold) - NVMe SSD

Query: SELECT ss_sold_date_sk FROM store_sales;

COPY parallelism	AM	Table size	Buffers read	Bytes read	Read speed	I/O time	Total time
1	Heap	112G	14727504	112G	185 MB/s	62s	129s
	Zedstore	61G	180779	1.4G	72 MB/s	20s	75s
16	Heap	112G	14727504	112G	185 MB/s	59s	127s
	Zedstore	59G	181220	1.4G	72 MB/s	20s	85s

Results w/o optimization (cold) - HDD

Query: SELECT ss_sold_date_sk FROM store_sales;

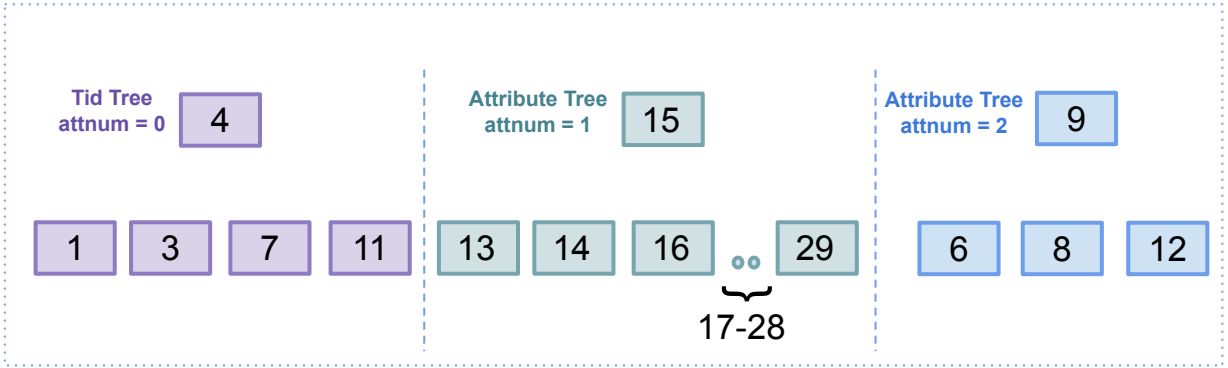
COPY parallelism	AM	Table size	Buffers read	Bytes read	Read speed	I/O time	Total time
16	Heap	112G	14727504	112G	185 MB/s	115s	212s
	Zedstore	59G	181220	1.4G	2.3MB/s	634s	730s

B-Tree page randomness

Blocks layout for table *foo*

0	1	2	3
4	5	6	7
8	9	10	11
12	13	14	15
16	17	18	19
20	21	22	23
24	25	26	27
28	29		

B-Tree Pages

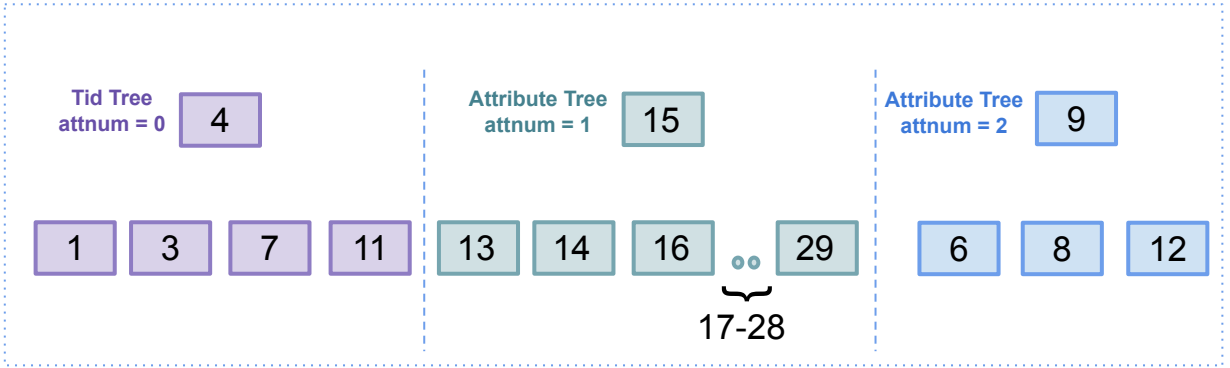


B-Tree page randomness

Blocks layout for table *foo*

0	1	2	3
4	5	6	7
8	9	10	11
12	13	14	15
16	17	18	19
20	21	22	23
24	25	26	27
28	29		

B-Tree Pages



Attribute and Tid tree pages are not contiguous enough on disk

1. HDD suffers more from random reads than NVMe SSD
2. Concurrent loads magnifies attribute page randomness

Optimization to reduce page randomness

- Attribute-level free page maps (FPMs)
- Reloption: ***zedstore_rel_extension_factor*** (default = 1)
  - Extends zedstore relations by *zedstore_rel_extension_factor* # of **consecutive** blocks.
  - Prepend the extra blocks to the attribute-level FPM

* More details: see hacker's thread: Rel Ext factor

Results w/ optimization (cold) - NVMe SSD

Query: SELECT ss_sold_date_sk FROM store_sales;

COPY parallelism	AM	Table size	Rel Ext factor	I/O time	Total time
16	Heap	112G	NA	59s	127s
	Zedstore	59G	1	20s	85s
	Zedstore	61G	4096	21s	87s

Results w/ optimization (cold) - HDD

Query: SELECT ss_sold_date_sk FROM store_sales;

COPY parallelism	AM	Table size	Bytes read	Rel Ext factor	Read speed	I/O time	Total time
16	Heap	112G	112G	NA	185 MB/s	115s	212s
	Zedstore	59G	1.4G	1	2.3 MB/s	634s	730s
	Zedstore	61G	1.4G	4096	573 MB/s	2.5s	82s

Results w/o optimization (warm) - NVMe SSD

Query: SELECT ss_sold_date_sk FROM store_sales;

AM	Table size	First run		Second run	
		I/O time	Total time	I/O time	Total time
Heap	112G	59s	127s	59s	127s
Zedstore	59G	20s	85s	0s	85s

Second run: w/o optimization & w/o clearing cache
shared_buffers = 10G
COPY parallelism = 16

SELECT * (cold)

Query: SELECT * FROM store_sales;

AM	Table size	Disk	Rel Ext factor	I/O time	Total time
Heap	112G	NVMe SSD	NA	64s	127s
Zedstore	59G	NVMe SSD	1	449s	757s
Zedstore	61G	NVMe SSD	4096	487s	812s
Heap	112G	HDD	NA	130s	214s
Zedstore	59G	HDD	1	2401s	2813s
Zedstore	59G	HDD	4096	354s	716s

COPY parallelism = 16

Storageperf test suite - NVMe SSD

testname	heap time (s)	heap size (bytes)	heap wal (bytes)	ZS time (s)	ZS size (bytes)	ZS wal (bytes)	time ratio (zs/h)	size ratio (zs/h)	wal ratio (zs/h)
onecol, insert-select	0.489284	18153472	32094320	0.166281	2793472	4767728	0.34	0.15	0.15
onecol, COPY	0.17744	18153472	6281568	0.122072	2457600	3185832	0.69	0.14	0.51
onecol, SELECT, seqscan	0.088754	18161664	0	0.170485	2457600	0	1.92	0.14	
onecol, SELECT, seqscan, parallel seqscan disabled	0.186381	18161664	0	0.159142	2457600	0	0.85	0.14	
onecol, SELECT, bitmap scan	0.319402	24018944	0	0.323106	8314880	0	1.01	0.35	
onecol, deleted half	0.26686	24018944	14041192	1.258317	13508608	84885592	4.72	0.56	6.05
onecol, vacuumed	0.111973	24018944	1521984	0.55551	13508608	37797928	4.96	0.56	24.83
nullcol, insert-select	0.599832	24018944	30088384	0.313375	13508608	3327152	0.52	0.56	0.11
nullcol, COPY	0.158409	24018944	5277560	0.105692	13508608	1742328	0.67	0.56	0.33
lockperf, pgbench, FOR SHARE	2.995451	8192	2841952	2.927252	114688	9022728	0.98	14	3.17
lockperf, pgbench, UPDATE	4.374822	212992	3139960	10.056802	917504	72857392	2.3	4.31	23.2
inlinecompress, insert-select	2.029034	5095424	6419032	0.172824	1105920	1915240	0.09	0.22	0.3
inlinecompress, COPY	3.282228	5095424	4233320	1.520007	1105920	1798432	0.46	0.22	0.42
inlinecompress, SELECT, seqscan	3.387322	5103616	0	3.320613	1105920	0	0.98	0.22	
inlinecompress, SELECT, bitmap scan	6.750271	5496832	0	6.531338	1499136	0	0.97	0.27	
inlinecompress, deleted half	0.029051	5496832	1386792	0.156748	1982464	8265224	5.4	0.36	5.96
inlinecompress, vacuumed	0.020428	5496832	262520	0.092524	1982464	4224352	4.53	0.36	16.09
toastcol, insert-select	4.430296	6316032	6188624	4.474798	8224768	5908024	1.01	1.3	0.95
toastcol, COPY	7.149559	6316032	6188424	7.194714	8224768	5899112	1.01	1.3	0.95
toastcol, SELECT, seqscan	6.607809	6332416	0	6.602179	8224768	0	1	1.3	
toastcol, SELECT, bitmap scan	6.580433	6332416	56	6.575141	8224768	56	1	1.3	1
toastcol, deleted half	0.004883	6332416	97544	0.004553	8224768	83984	0.93	1.3	0.86
toastcol, vacuumed	0.012939	6332416	101584	0.014681	8224768	70304	1.13	1.3	0.69

Open areas

Open areas - performance

- Making index (only) scans more efficient
- BRIN improvements
- Eliminate TOAST bloat
  - For UPDATEs/DELETEs - zedstore-toast pages leaked!
- Avoid full table rewrites for ALTER TABLE column ops
  - ADD/DROP COL
  - SET DATA TYPE
- Making updates/deletes faster
  - table_tuple_fetch_row_version() fetches full row
  - In-place updates
- Improve parallel seq scan
- Faster attstream decoding

Open areas

- Column families / row store
- Tid reuse
- Replace UNDO with upstream UNDO
- Make meta-page overflow to support wider tables
- Reuse free space on partially full pages
- Compression:
  - Different compression algorithms
  - Dictionary based tuple level compression
  - Roaring bitmaps
- Make planner aware!

Related hackers threads

- Extracting only the columns needed for a query in planner.
- APIs to pass down projection list.
  - beginscan_with_column_projection()
- Statistic patch
  - Better estimation of number of pages for queries that select a subset of columns
  - Using these statistics in the planner to get better plans.
- UNDO framework

Tools to play with Zedstore

- Github repo
- Inspect functions: src/backend/access/zedstore/zedstore_inspect.c
- Ansible playbook for TPC-DS loading
- To run storageperf:
  - `cd src/test/storageperf && PATH=<path_to_bin>:$PATH psql postgres -f driver.sql`
- To run the regress tests:

```
EXTRA_REGRESS_OPTS="--ignore-plans-and-tuple-order-diff" PGOPTIONS="-c
default_table_access_method=zedstore" make installcheck
```

How you can get involved!

- Join the zedstore discussion on Hackers:
- The #zedstore channel on Greenplum slack. Sign up!

Thank you to everyone involved in the thread!

- Ajin Cherian
- Alexandra Wang
- Alvaro Herrera
- Amit Kapila
- Andreas Karlsson
- Andres Freund
- Ashutosh Sharma
- Ashwin Agrawal
- David Kimura
- Heikki Linnakangas
- Jesse Zhang
- Justin Pryzby
- Konstantin Knizhnik
- Magnus Hagander
- Mark Kirkwood
- Melanie Plageman
- Pengzhou Tang
- Peter Geoghegan
- Rafia Sabih
- Robert Eckhardt
- Robert Haas
- Soumyadeep Chakraborty
- Stephen Frost
- Taylor Vesely
- Tom Lane
- Tomas Vondra
- Tsunakawa Takayuki

Q&A