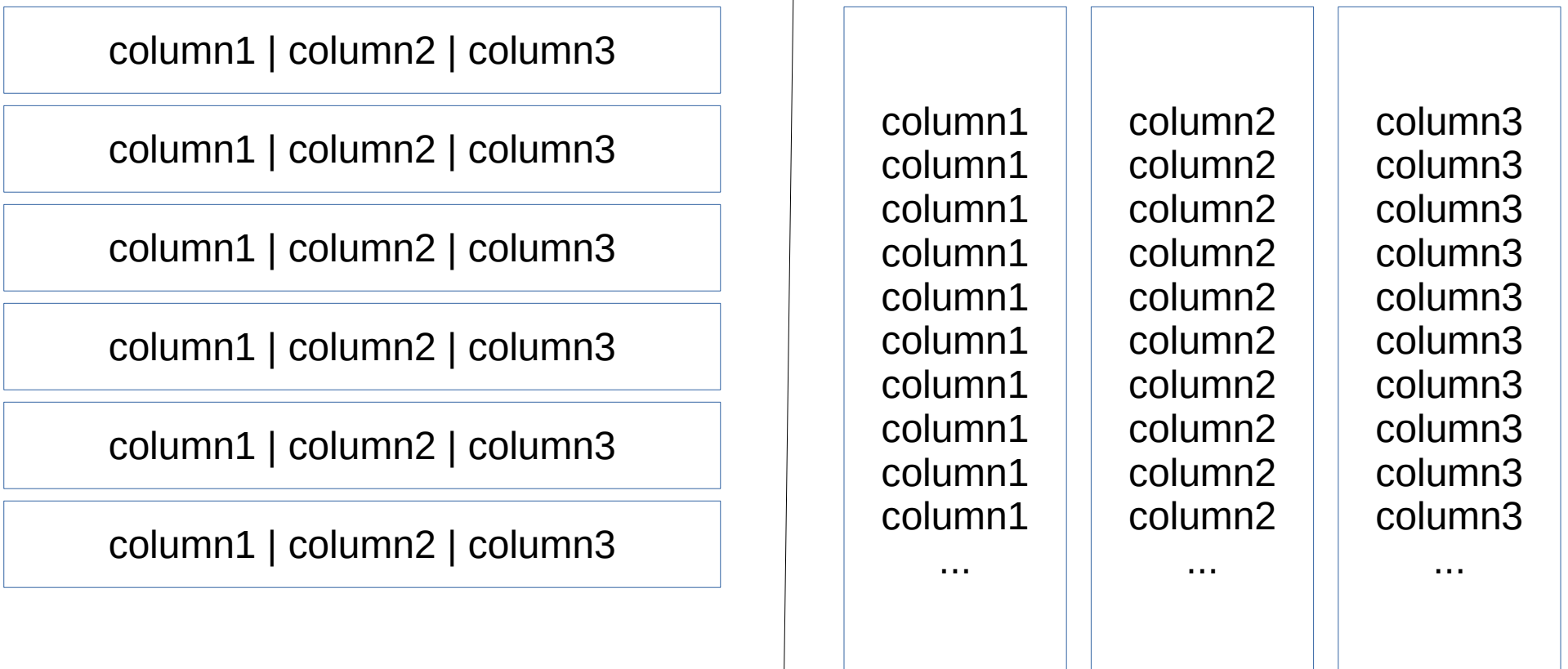
ZedStore – Column store for PostgreSQL

Heikki Linnakangas, Pivotal

PGConf.eu 2019, Milan


column1 | column2 | column3
column1 | column2 | column3
column1 | column2 | column3
column1 | column2 | column3
column1 | column2 | column3
column1 | column2 | column3
column1
column1
column1
column1
column1
column1
column1
column1
column1
...
column2
column2
column2
column2
column2
column2
column2
column2
column2
...
column3
column3
column3
column3
column3
column3
column3
column3
column3
...

Why column store?

- Queries that scan only one or two columns out of a wide table
  - Only need to access the needed columns
  - Analytics queries, aggregation
- Compression
  - Similar data closely together → compresses well

Column store disadvantages

- Stitching rows back together is slow
  - Single-row fetching is expensive
  - Bad for OLTP

Column store tricks & variants

- Hybrid column / row store
  - Store chunks of rows, column-oriented within chunk
- Store each column sorted
  - Implicit index on every column

Column Stores in PostgreSQL ecosystem

- cstore_fdw
- AOCO tables in Greenplum
- vops
- Roll your own, using a view on a join

Project background

- Work started in Spring 2019
- Around the same time that the Table AM API was committed to PostgreSQL v12

– Hey, let’s build Yet Another Column Store!

– Great idea!

Pivotal’s reasons

- Greenplum v6 is a fork of PostgreSQL 9.4
- Greenplum v7 will be based on PostreSQL X.X
- Greenplum has a column store implementation, called AOCO tables

Greenplum AOCO

- Append-Optimized Column-Oriented
  - Formerly Append-Only Column-Oriented
- Consists of multiple relfiles, one for each column
- Support for updates bolted on using an auxiliary heap table to store visibility information
- Support for indexes also bolted on and awkward

Greenplum AOCO

- Has served well, but shows its age
- Requires effort to maintain as we merge with PostgreSQL
- We want to replace it with Zedstore

The Team

Group of Pivotal’s experienced PostgreSQL / Greenplum hackers:

- Alexandra Wang
- Ashwin Agrawal
- Heikki Linnakangas
- Melanie Plageman
- Taylor Vesely

“Zedstore” ?

“
BTW, can I express a small measure of disappointment that the name for the thing under discussion on this thread chose to be called "zedstore"? That seems to invite confusion with "zheap", especially in parts of the world where the last letter of the alphabet is pronounced "zed," where people are
going to say zed-heap and zed-store. Brr.

Robert Haas wrote on 2019-04-09 on pgsql-hackers

Requirements

Zedstore design requirements

- Community project
  - PostgreSQL licence
  - Built-in or extension
- General purpose
  - All functionality must work
  - All indexes, all datatypes, MVCC, etc.
- Column-oriented

Zedstore performance goals

- Must be faster than heap for *some* queries
- Good-enough performance
  - vs Greenplum AOCO
  - vs other implementations
- Compressed
  - With good-enough compression ratio

More design goals

- No VACUUM, no freezing
- Built-in TOASTing of large datums
- Fast ADD/DROP COLUMN
- Support later evolution
  - different compression schemes
  - vectorization

Zedstore design

- Plugs into the Table AM API
- Relies on normal PostgreSQL infrastructure
  - 8k blocks
  - Buffer manager
  - WAL-logging

Status

- Most functionality works
  - make check-world mostly passes
- Performance is getting there
  - Not great vs competitors
  - Faster than heap for the right query

Zedstore internal structure

B-trees for the win!

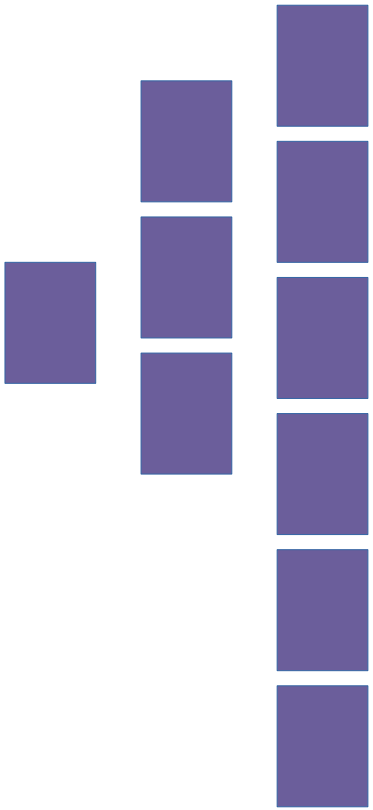
Zedstore concepts

- TID
  - 64-bit integer
  - Uniquely identifies a row
  - Every row version is assigned a new TID (no in-place UPDATE yet)
  - Usable space is only 48 bits, so that zedstore TID fits in PostgreSQL ItemPointer

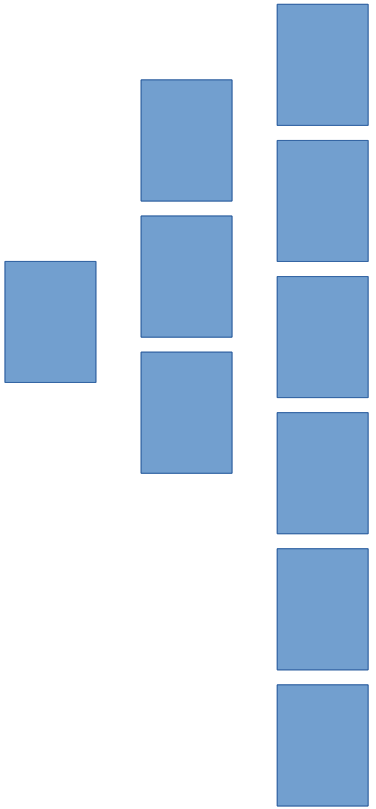
Forest of B-trees

- Metadata page
- TID tree
  - B-tree containing visibility information
  - Keyed by TID
- Attribute trees, one for each column
  - B-trees containing column data
  - Also keyed by TID!
- UNDO log for visibility information

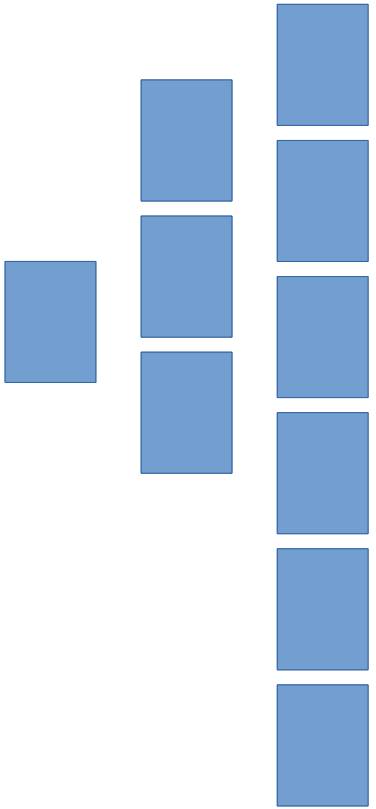
TID tree



Attribute tree
for column 1



Attribute tree
for column 2



Metapage

Free
Space
Map

UNDO
log

zsTOAST
pages

Most pages are B-tree pages

```
select count(*), pg_zs_page_type('lineitem', g)
from generate_series(0, pg_table_size('lineitem') / 8192 - 1) g
group by 2;

 count | pg_zs_page_type
-------+-----------------
     1 | META
 41291 | BTREE
     5 | UNDO
   130 | FREE
(4 rows)
```

Most pages are attribute leaf pages

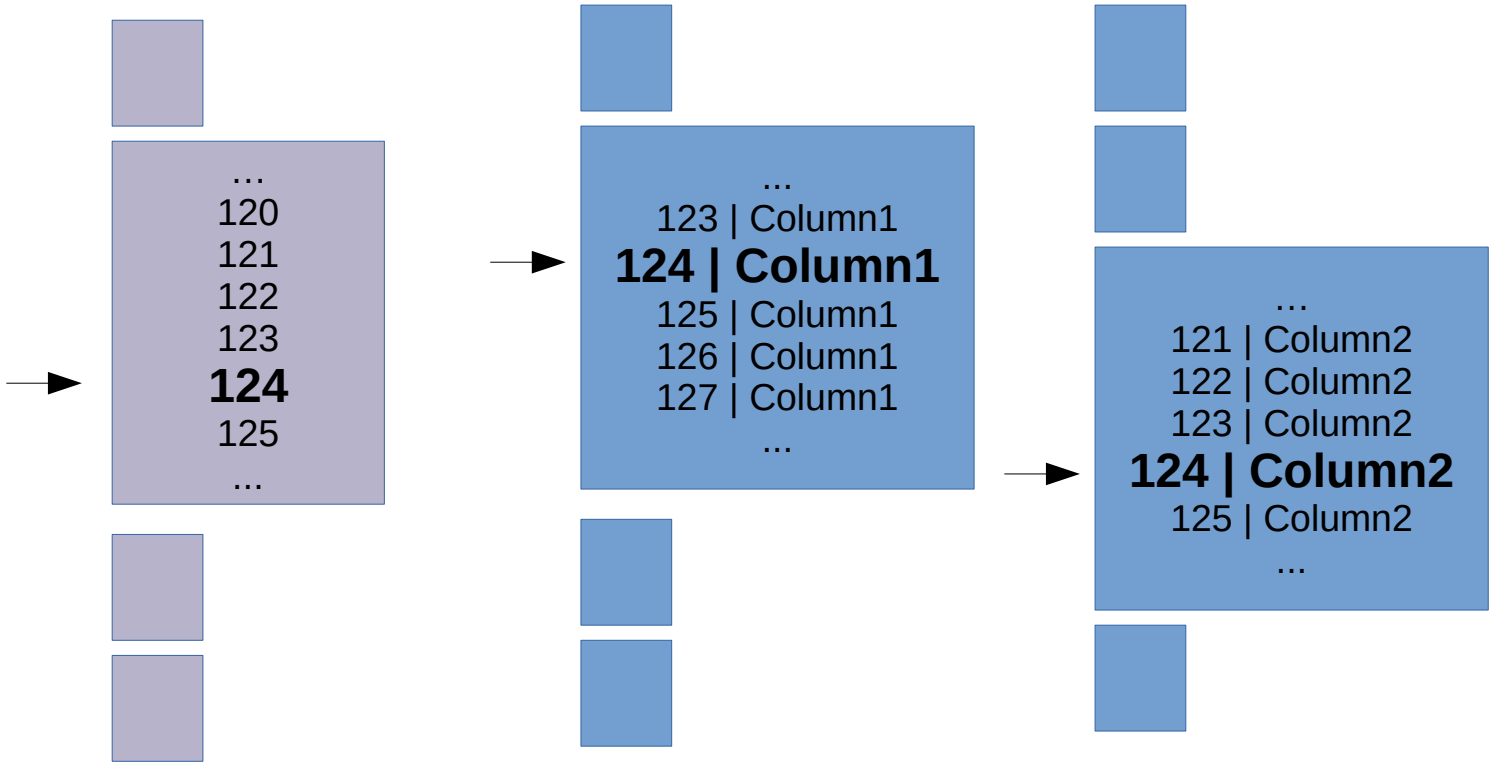
```
select attno=0 as is_tidtree_page, level, count(*)
from pg_zs_btree_pages('lineitem')
group by 1, 2 order by 1, 2;
```

```
 is_tidtree_page | level | count
-----------------+-------+-------
 f               |     0 | 40571
 f               |     1 |    95
 f               |     2 |    14
 t               |     0 |   608
 t               |     1 |     2
 t               |     2 |     1
(6 rows)
```

Scanning the trees

- When querying, the TID tree and all the needed attribute trees are scanned in lock-step
- On insert, a new item is added to each tree

Sequential Scan



TID tree page format

- TID and UNDO pointer for every row
  - In a compact representation
  - Loosely based on “Efficient Columnar Storage in B-trees” by Goetz Graefe
- Detailed visibility information stored in UNDO log
  - UNDO log can be trimmed when transactions age

TID tree page, logical content

TID | UNDO pointer

119 | ALL_VISIBLE

120 | 7599293

121 | DEAD

123 | 7599293

124 | 8000222

TID tree page physical content

TID | UNDO p

119 | ALL_VIS

120 | 7599293

121 | DEAD

123 | 7599293

124 | 8000222

Delta encoded, Simple-8b, Typically 1-10 bits per TID

TIDs | UNDO pointers | UNDO indexes

119, 1, 1, 2, 1 |
7599293, 8000222 |
V, 0, D, 0, 1

2 bits per TID for UNDO indexes

Attribute trees

- TID + column data
- Divided into “chunks”, containing data for 1-60 rows
- LZ4 compressed

Attribute page format

TID | column value

119 | 2001-01-01

120 | 1999-12-24

121 | 2000-05-01

123 | 2019-10-17

124 | 2018-10-23

Attribute chunk physical layout

TID | column value

119 | 2001-01-01

120 | 1999-12-24

121 | 2000-05-01

123 | 2019-10-17

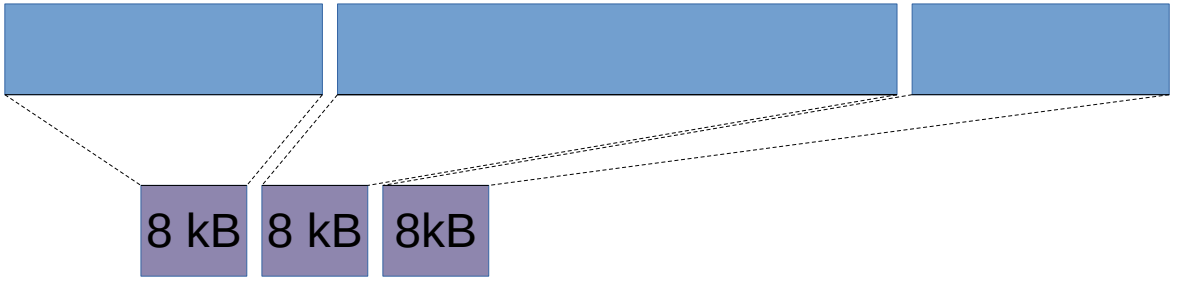
124 | 2018-10-23

TIDs | column values

119,1,1,2,1 | 2001-01-01, 1999-12-24, 2000-05-01, 2019-10-17, 2018-10-23

Compression

- LZ4
- Compress into 8k output blocks.
  - not all libraries can do that



Compression

- Future work:
  - Dictionary
  - Delta encoding
  - Different algorithms
  - Different “profiles” for different datatypes
- Compress larger input blocks, split compressed blocks across pages

Zedstore storage design

- Pages can be split and moved around freely
  - TID is a logical row identifier
  - Makes online VACUUM FULL possible
- Pages are self-identifying
  - Helps with disaster recovery and forensics

Demo

~$ ls -lh lineitem.sql

-rw-r--r-- 1 heikki heikki **946M** Aug 29 2016 lineitem.sql

```
~$ less lineitem.sql
...
CREATE TABLE lineitem (
    l_orderkey integer NOT NULL,
    l_partkey integer NOT NULL,
    l_suppkey integer NOT NULL,
    l_linenumber integer NOT NULL,
    l_quantity double precision NOT NULL,
    l_extendedprice double precision NOT NULL,
    l_discount double precision NOT NULL,
    l_tax double precision NOT NULL,
    l_returnflag character(1) NOT NULL,
    l_linestatus character(1) NOT NULL,
    l_shipdate date NOT NULL,
    l_commitdate date NOT NULL,
    l_receiptdate date NOT NULL,
    l_shipinstruct text NOT NULL,
    l_shipmode text NOT NULL,
    l_comment text NOT NULL
);

...
```

~$ less lineitem.sql

...

COPY lineitem (l_orderkey, l_partkey, l_suppkey, l_linenumber, l_quantity, l_extendedprice, l_discount, l_tax, l_returnflag, l_linestatus, l_shipdate, l_commitdate, l_receiptdate, l_shipinstruct, l_shipmode, l_comment) FROM stdin;

1 155190 7706 1 17 21168.2299999999996 0.04000000000000000008 0.02000000000000000004 N O 1996-03-13 1996-02-12

1996-03-22 DELIVER IN PERSON TRUCK blithely regular ideas caj

1 67310 7311 2 36 45983.1600000000035 0.0899999999999999967 0.0599999999999999978 N O 1996-04-12 1996-02-28

1996-04-20 TAKE BACK RETURN MAIL slyly bold pinto beans detect s

1 63700 3701 3 8 13309.6000000000004 0.100000000000000006 0.02000000000000000004 N O 1996-01-29 1996-03-05

1996-01-31 TAKE BACK RETURN REG AIR deposits wake furiously dogged,

1 2132 4633 4 28 28955.6399999999994 0.0899999999999999967 0.0599999999999999978 N O 1996-04-21 1996-03-30

1996-05-16 NONE AIR even ideas haggle. even, bold reque

1 24027 1534 5 24 22824.4799999999996 0.100000000000000006 0.04000000000000000008 N O 1996-03-30 1996-03-14

1996-04-01 NONE FOB carefully final gr

...

```
postgres=# set
default_table_access_method = zedstore;
SET

postgres=# \i ~/lineitem.sql
CREATE TABLE
COPY 6001215
Time: 19082.391 ms (00:19.082)


postgres=# \d+
                                List of
relations
 Schema |   Name   | Type  | Owner  |
Persistence |  Size  | Description
--------+----------+-------+--------
+-------------+--------+-------------
 public | lineitem | table | heikki |
permanent   | 324 MB |
(1 row)
```

```
postgres=# set
default_table_access_method = heap;
SET

postgres=# \i ~/lineitem.sql
CREATE TABLE
COPY 6001215
Time: 19527.532 ms (00:19.528)


postgres=# \d+
                                List of
relations
 Schema |   Name   | Type  | Owner  |
Persistence |  Size  | Description
--------+----------+-------+--------
+-------------+--------+-------------
 public | lineitem | table | heikki |
permanent   | 817 MB |
(1 row)
```

```
postgres=# select sum(l_quantity) from
lineitem_zedstore;
    sum
-----------
 153078795
(1 row)

Time: 375.363 ms
postgres=# select sum(l_quantity) from
lineitem_zedstore;
    sum
-----------
 153078795
(1 row)

Time: 371.240 ms
postgres=# select sum(l_quantity) from
lineitem_zedstore;
    sum
-----------
 153078795
(1 row)

Time: 373.869 ms
```

```
postgres=# select sum(l_quantity) from
lineitem_heap;
    sum
-----------
 153078795
(1 row)

Time: 585.413 ms
postgres=# select sum(l_quantity) from
lineitem_heap;
    sum
-----------
 153078795
(1 row)

Time: 587.493 ms
postgres=# select sum(l_quantity) from
lineitem_heap;
    Sum
-----------
 153078795
(1 row)

Time: 565.927 ms
```

Aya Iwata / Fujitsu:

Zedstore: Compression ratio

FUJITSU

- pgbench -i -s 1000

	PG v10	PG v12 + Zedstore	ratio
pgbench_accounts	1,281 MB	264 MB	-80%
pgbench_branches	8,192 bytes	56 kB	+600%
pgbench_tellers	72 kB	64 kB	-11%

- SELECT AVG() on 1 million rows: 4,679.0 ms ✉ 379.7 ms
- Loading pgbench_accounts data: 50.3 s ✉ 26.2 s

https://www.postgresql.org/message-id/101f8490-a7bc-230e-cb38-730e26ca81bd%40catalyst.net.nz

“storagetest” performance suite

testname	heap time	heap size	heap wal	ZS time	ZS size	ZS wal	time ratio	size ratio	wal ratio
onecol, insert-select	0.261109	18153472	32094264	0.082401	2719744	4768224	0.32	0.15	0.15
onecol, COPY	0.104713	18153472	6281568	0.070589	2523136	3463944	0.67	0.14	0.55
onecol, SELECT, seqscan	0.093415	18161664	0	0.073447	2523136	0	0.79	0.14	
onecol, SELECT, bitmap scan	0.146578	29433856	0	0.146383	13795328	0	1	0.47	
onecol, deleted half	0.138483	29433856	14041192	0.562652	18776064	86793056	4.06	0.64	6.18
onecol, vacuumed	0.090547	29433856	1959760	0.296329	18776064	38206984	3.27	0.64	19.5
nullcol, insert-select	0.309857	29433856	30088360	0.15649	18776064	3327656	0.51	0.64	0.11
nullcol, COPY	0.092809	29433856	5277592	0.077313	18776064	2020440	0.83	0.64	0.38
pgbench, FOR SHARE	0.26859	8192	2125888	1.074371	49152	7005376	4	6	3.3
pgbench, UPDATE	0.506583	1351680	2877488	3.871771	983040	79851576	7.64	0.73	27.75
inlinecompress, insert-select	1.060063	5095424	6419016	0.304543	1105920	1915216	0.29	0.22	0.3
inlinecompress, COPY	1.961442	5095424	4233264	1.124002	1490944	2700904	0.57	0.29	0.64
inlinecompress, SELECT, seqsc	2.112152	5103616	0	1.240088	1490944	0	0.59	0.29	
inlinecompress, SELECT, bitma	4.259661	9912320	0	2.32936	6299648	0	0.55	0.64	
inlinecompress, deleted half	0.03532	9912320	1386808	0.095112	6389760	20135192	2.69	0.64	14.52
inlinecompress, vacuumed	0.030352	9912320	287208	0.093844	6389760	4101664	3.09	0.64	14.28
toastcol, insert-select	2.376949	6316032	6188576	2.368413	8224768	5908056	1	1.3	0.95
toastcol, COPY	4.160188	6316032	6188448	4.169799	8282112	7893192	1	1.31	1.28
toastcol, SELECT, seqscan	4.159859	6332416	0	4.128762	8282112	56	0.99	1.31	
toastcol, SELECT, bitmap scan	4.378755	6332416	0	4.321935	8282112	0	0.99	1.31	
toastcol, deleted half	0.001798	6332416	97544	0.002902	8282112	1581200	1.61	1.31	16.21
toastcol, vacuumed	0.013666	6332416	101584	0.011794	8282112	73104	0.86	1.31	0.72

TODO

Zedstore TODOs

- Performance
- Different compression schemes
- Free space management
- TID reuse

Zedstore TODOs

- In-place UPDATEs
- Faster ALTER TABLE DROP COLUMN

PostgreSQL TODO

- Pass Column projection down to table AM
  - A column store **really** doesn’t want to fetch columns unnecessarily

PostgreSQL TODO

- Table AM API improvements
  - Visibility map, for index-only scans
  - “block” sampling and scanning
  - Cost estimation
  - Toasting
- Wider TIDs

PostgreSQL TODOs

- UNDO logging

Longer-term future work

- Executor tricks
  - Vectorization
  - Filtering before fetching all columns

Thanks!

- Development happens on github:

  https://github.com/greenplum-db/postgres/tree/zedstore

- Test your workload, tell us how it works!