VACUUM Strategy: Autovacuum, FSM, Visibility Map

Selena Deckelmann
End Point Corporation
@selenamarie

Hi!

- Postgres consultant for End Point
- User Group Liaison for postgresql.org
- Conference Organizer


teh vacuum!
itz got me!
ICANHASCHEEZBURGER.COM

What we'll cover

- VACUUM basics: MVCC, FULL
- The old Free Space Map (8.0->8.3)
- The new Free Space Map (8.4-> the FUTURE)
- B.A.: Before Autovacuum
- A.A.: After Autovacuum
- Autovacuum best practices

Why VACUUM sucks

- Lots of I/O
- Long running VACUUM queries that annoy sysadmins, clients, DBAs
- Different than other databases (manual)

Why do we have VACUUM anyway?

- Normal maintenance for dead rows
- Extreme maintenance for table bloat
- Preventing transaction wrap-around

VACUUMing for normal maintenance

Let's start with: MVCC

Multi Version Concurrency Control

PgCon 2009

VACUUMing for normal maintenance

MVCC - Pessimistic rollback behavior

Old versions of rows stored in the same relation space

PgCon 2009

A very simple example

- SIMPLE. ABSTRACTED.
- I am not explaining HOT today.
- See Pavan's presentation from PgCon 2008: http://www.pgcon.org/2008/schedule/events/105.en.html

How MVCC works

Table

"Cats are not very cute."

Example 1

Table

ID #2 - SELECT

"Cats are not very cute."

How MVCC works

Table

ID #2 - SELECT

"Cats are not very cute."

"Cats are adorable."

ID #3 - UPDATE

There's a new row version!

How MVCC works

Once Transaction ID #3 is committed, the original row is no longer visible to **future** transactions, but still exists and is visible to ID #2.

How MVCC works

How MVCC works

Once Transaction ID #2 ends, then the row associated with ID #1 is no longer visible to any future transactions.

Example 2

Table

~~"Cats are not very cute."~~

"Cats are adorable."

"Cats should rule the world."

ID #6 - UPDATE

Example 2

Table

~~"Cats are not very cute."~~

"Cats are adorable."

~~"Cats should rule the world."~~

ID #6 - UPDATE

ID #6 - ROLLBACK

Vacuum?

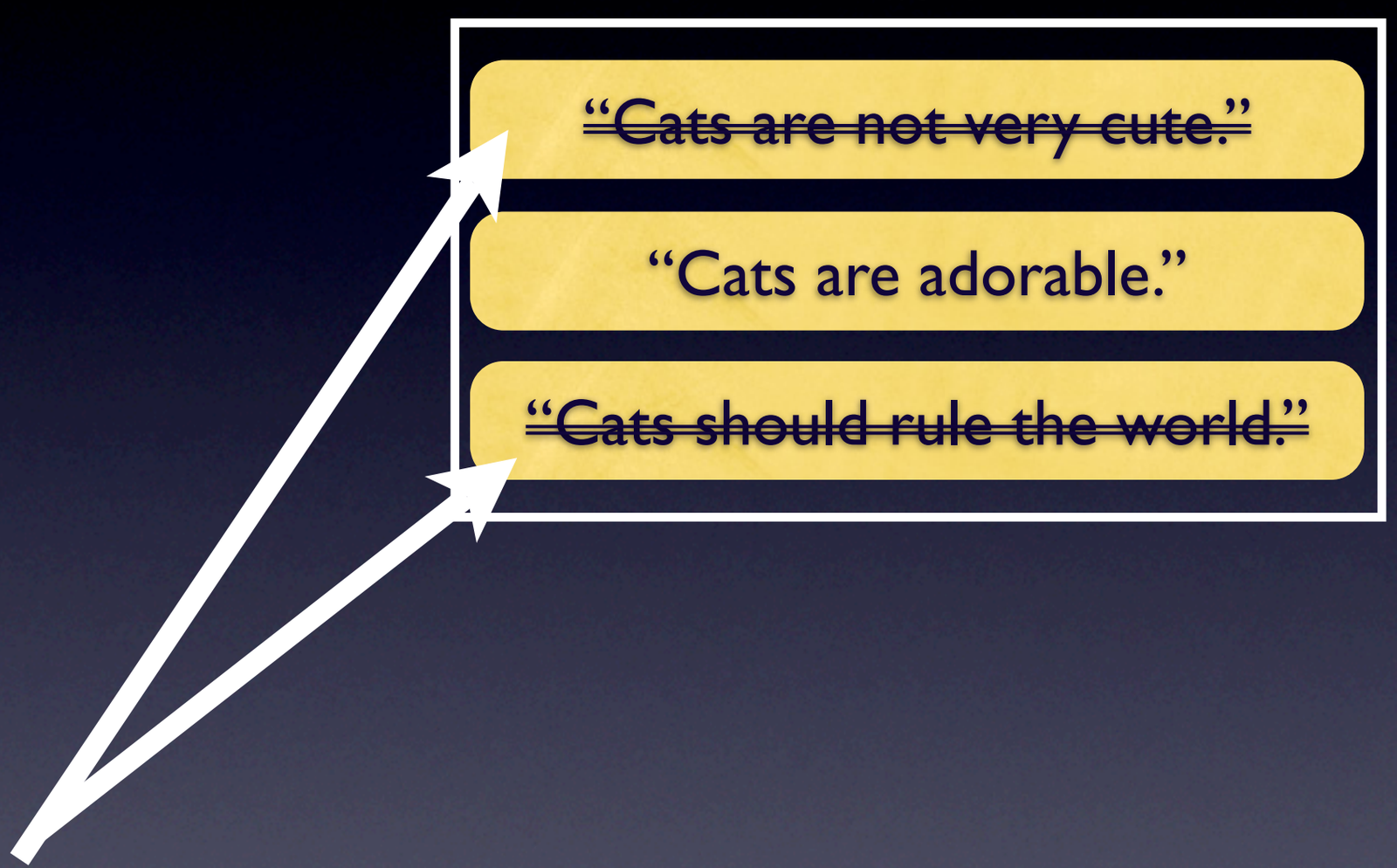
Table

~~"Cats are not very cute."~~

"Cats are adorable."

~~"Cats should rule the world."~~

Vacuum!



VACUUM cleans up these rows

Table

"Cats are adorable."

PgCon 2009

BLOAT

PgCon 2009

BLOAT

- A measure of how space "dead tuples" take up in tables and indexes
- Bloat slows down scans and will eventually cause basic table operations to be slow.

BLOAT Avoidance!

- VACUUM regularly
- Keep Free Space Map big enough (pre 8.4)
- Monitor bloat with check_postgres:
  http://bucardo.org/check_postgres

  check_postgres_bloat --port=5432
  --warning='100 M' --critical='200 M'

BLOAT

```
SELECT
  schemaname, tablename, reltuples::bigint, r[illegible], ot[illegible],
  ROUND(CASE WHEN otta=0 THEN 0.0 ELSE sml.relpages/otta::numeric END,1) AS tbloat,
  CASE WHEN relpages < otta THEN 0 ELSE relpages::bigint - otta END AS wastedpages,
  CASE WHEN relpages < otta THEN 0 ELSE bs*(sml.relpages-otta)::bigint END AS wastedbytes,
  CASE WHEN relpages < otta THEN '0 bytes'::text ELSE (bs*(relpages-otta))::bigint || ' bytes' END AS wastedsize,
  iname, ituples::bigint, ipages::bigint, iotta,
  ROUND(CASE WHEN iotta=0 OR ipages=0 THEN 0.0 ELSE ipages/iotta::numeric END,1) AS ibloat,
  CASE WHEN ipages < iotta THEN 0 ELSE ipages::bigint - iotta END AS wastedipages,
  CASE WHEN ipages < iotta THEN 0 ELSE bs*(ipages-iotta) END AS wastedibytes,
  CASE WHEN ipages < iotta THEN '0 bytes' ELSE (bs*(ipages-iotta))::bigint || ' bytes' END AS wastedisize
FROM (
  SELECT
    schemaname, tablename, cc.reltuples, cc.relpages, bs,
    CEIL((cc.reltuples*((datahdr+ma-
      (CASE WHEN datahdr%ma=0 THEN ma ELSE datahdr%ma END))+nullhdr2+4))/(bs-20::float)) AS otta,
    COALESCE(c2.relname,'?') AS iname, COALESCE(c2.reltuples,0) AS ituples, COALESCE(c2.relpages,0) AS ipages,
    COALESCE(CEIL((c2.reltuples*(datahdr-12))/(bs-20::float)),0) AS iotta
  FROM (
    SELECT
      ma,bs,schemaname,tablename,
      (datawidth+(hdr+ma-(case when hdr%ma=0 THEN ma ELSE hdr%ma END)))::numeric AS datahdr,
      (maxfracsum*(nullhdr+ma-(case when nullhdr%ma=0 THEN ma ELSE nullhdr%ma END))) AS nullhdr2
    FROM (
      SELECT
        schemaname, tablename, hdr, ma, bs,
        SUM((1-null_frac)*avg_width) AS datawidth,
        MAX(null_frac) AS maxfracsum,
        hdr+(
          SELECT 1+count(*)/8
          FROM pg_stats s2
          WHERE null_frac<>0 AND s2.schemaname = s.schemaname AND s2.tablename = s.tablename
        ) AS nullhdr
      FROM pg_stats s, (
        SELECT
          (SELECT current_setting('block_size')::numeric) AS bs,
          CASE WHEN substring(v,12,3) IN ('8.0','8.1','8.2') THEN 27 ELSE 23 END AS hdr,
          CASE WHEN v ~ 'mingw32' THEN 8 ELSE 4 END AS ma
        FROM (SELECT version() AS v) AS foo
      ) AS constants
      GROUP BY 1,2,3,4,5
    ) AS foo
  ) AS rs
  JOIN pg_class cc ON cc.relname = rs.tablename
  JOIN pg_namespace nn ON cc.relnamespace = nn.oid AND nn.nspname = rs.schemaname AND nn.nspname <> 'information_schema'
  LEFT JOIN pg_index i ON indrelid = cc.oid
  LEFT JOIN pg_class c2 ON c2.oid = i.indexrelid
) AS sml
WHERE tablename = 'addr'
ORDER BY wastedbytes DESC LIMIT 1;
```

PgCon 2009

Fixing bloat

- VACUUM FULL
- CLUSTER
- TRUNCATE
- Or most extreme: DROP/CREATE

Transaction wraparound

- Transaction wrap around avoidance!
- This is a counter.
- Transaction ID **will eventually** wrap
- Before we had autovacuum?
  Database shut down awaiting a VACUUM.

VACUUM vs VACUUM FULL

VACUUM updates the Free Space Map, and marks space to be reused

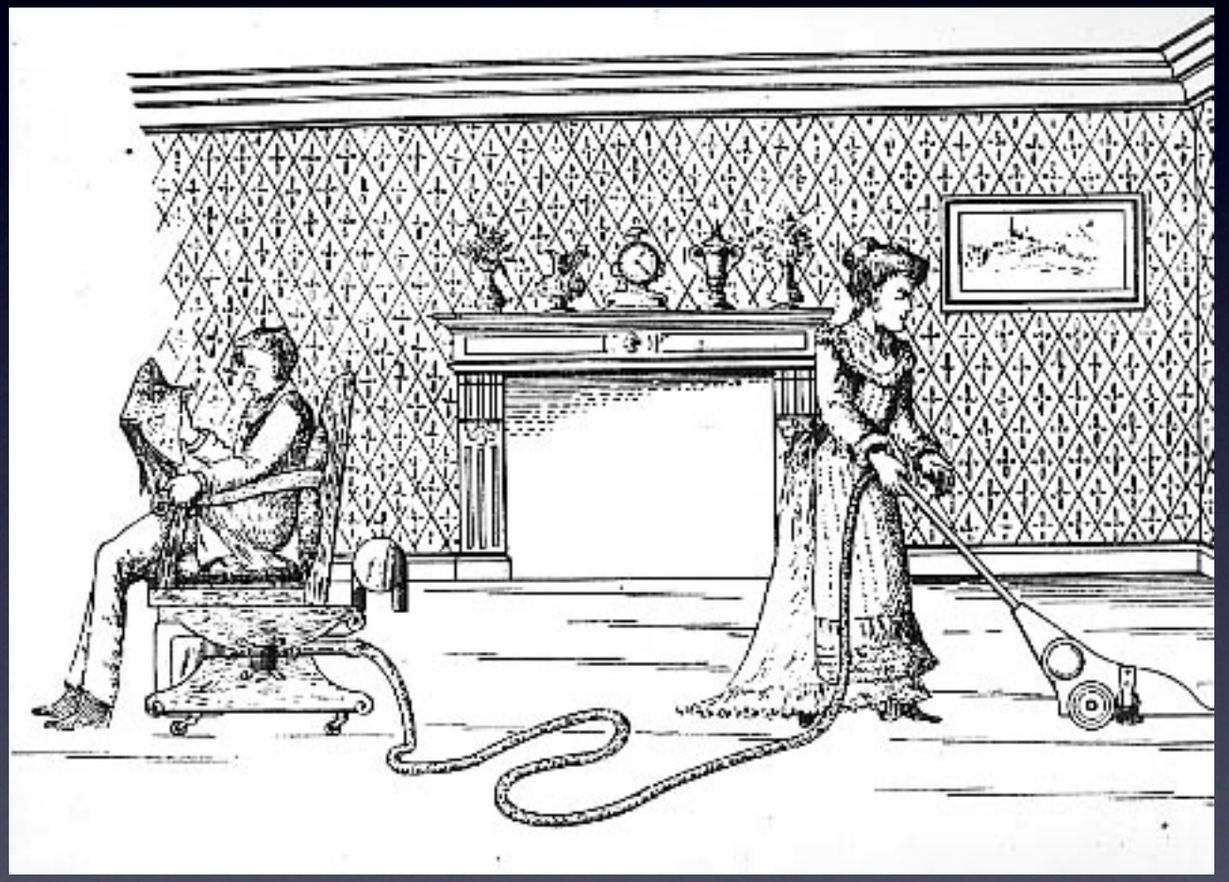
VACUUM FULL compacts and frees space back up to the filesystem

Regular VACUUM FULL is not recommended :)

VACUUM: pre 8.0

- 6.x-7.1: VACUUM FULL every time :(
- 7.2: Lazy VACUUM (Thanks, Tom!)

VACUUM strategy: 8.0



PgCon 2009

VACUUM strategy: 8.0

- Manual VACUUMing
- CRON
- Not fun.

Managing VACUUM

```
vacuum_cost_limit = 200          # 1-10000 credits
vacuum_cost_page_hit = 1         # 0-10000 credits
vacuum_cost_page_miss = 10       # 0-10000 credits
vacuum_cost_page_dirty = 20      # 0-10000 credits
vacuum_cost_delay = 0            # 0-1000 milliseconds
vacuum_freeze_min_age = 100000000
```

Managing VACUUM

```
vacuum_cost_limit = 200         # 1-10000 credits

vacuum_cost_page_hit = 1        # 0-10000 credits
vacuum_cost_page_miss = 10      # 0-10000 credits
vacuum_cost_page_dirty = 20     # 0-10000 credits
```

page_hit - “estimated cost for vacuuming a buffer found in the shared buffer cache. It represents the cost to lock the buffer pool, lookup the shared hash table and scan the content of the page. The default value is one.”

Managing VACUUM

```
vacuum_cost_limit = 200         # 1-10000 credits

vacuum_cost_page_hit = 1        # 0-10000 credits
vacuum_cost_page_miss = 10      # 0-10000 credits
vacuum_cost_page_dirty = 20     # 0-10000 credits
```

page_miss: “estimated cost for vacuuming a buffer that has to be read from disk. This represents the effort to lock the buffer pool, lookup the shared hash table, read the block in from the disk and scan its content.”

Managing VACUUM

```
vacuum_cost_limit = 200         # 1-10000 credits

vacuum_cost_page_hit = 1        # 0-10000 credits
vacuum_cost_page_miss = 10      # 0-10000 credits
vacuum_cost_page_dirty = 20     # 0-10000 credits
```

page_dirty: “estimated cost charged when vacuum modifies a block that was previously clean. Represents the extra I/O required to flush the dirty block out to disk again”

Managing VACUUM

```
vacuum_cost_delay = 0                   # 0-1000 milliseconds
```

cost_delay: number of milliseconds for VACUUM to sleep after exceeding the vacuum_cost_limit

The idea is to reduce the impact of I/O during VACUUM by spreading it out.

If you must: Start small, measure the changes!

Managing VACUUM

```
vacuum_freeze_min_age = 100000000
```

"The maximum time that a table can go unvacuumed is two billion transactions minus the vacuum_freeze_min_age that was used when it was last vacuumed."

8.1: Autovacuum

- Turned off by default
- Low lock priority - won't block DDL

Managing autovacuum

```
autovacuum = on
log_autovacuum_min_duration = -1

autovacuum_vacuum_scale_factor = 0.2
autovacuum_analyze_scale_factor = 0.1
autovacuum_vacuum_threshold = 50
autovacuum_analyze_threshold = 50
autovacuum_freeze_max_age = 200000000
autovacuum_vacuum_cost_delay = 20
autovacuum_vacuum_cost_limit = -1

autovacuum_max_workers = 3
autovacuum_naptime = 1min
```

Managing autovacuum

```
autovacuum = on
log_autovacuum_min_duration = -1
```

-1: don't log
0: log all
N: Log any that take longer than N seconds

EXAMPLE:

```
LOG:  automatic vacuum of table "public.mytable":
        index scans: 1
        pages: 0 removed, 5795 remain
        tuples: 179 removed, 37323 remain
        system usage: CPU 0.01s/0.02u sec elapsed 10.00 sec
```

Managing autovacuum

```
autovacuum_vacuum_scale_factor = 0.2
autovacuum_analyze_scale_factor = 0.1
```

How much of a table can change before VACUUM or ANALYZE are run.

YMMV but:

May have to lower vacuum_scale_factor: 0.1, 0.05
May have to lower analyze_scale_factor: 0.1

Managing autovacuum

vacuum threshold =

vacuum base threshold

+ vacuum scale factor * number of tuples

number of tuples == pg_class.reltuples

Managing autovacuum (before 8.4)

```
postgres@planetbeta:5432=# \d pg_autovacuum
    Table "pg_catalog.pg_autovacuum"
      Column       |  Type   | Modifiers
-------------------+---------+-----------
 vacrelid          | oid     | not null
 enabled           | boolean | not null
 vac_base_thresh   | integer | not null
 vac_scale_factor  | real    | not null
 anl_base_thresh   | integer | not null
 anl_scale_factor  | real    | not null
 vac_cost_delay    | integer | not null
 vac_cost_limit    | integer | not null
 freeze_min_age    | integer | not null
 freeze_max_age    | integer | not null
Indexes:
    "pg_autovacuum_vacrelid_index" UNIQUE, btree (vacrelid)
```

Managing autovacuum (before 8.4)

Caveats:

pg_autovacuum table not backed up - have to grab them explicitly

Slony

Managing autovacuum (8.4)

Set with Storage Parameters.

http://www.postgresql.org/docs/8.4/static/sql-createtable.html#SQL-CREATETABLE-STORAGE-PARAMETERS

Example:

```
CREATE TABLE test ( id int )
    WITH (autovacuum_enabled=TRUE);
```

Managing autovacuum (8.4)

Can also change parameters after table is created with ALTER TABLE commands.

Example:

ALTER TABLE test SET (autovacuum_enabled = FALSE);

```
test=# select relname,reloptions from pg_class where relname
= 'test';
 relname |          reloptions
---------+------------------------------
 test    | {autovacuum_enabled=false}
(1 row)
```

Managing autovacuum (8.4)

All options also include a separate configuration control for toast, specified by prefixing the setting with 'toast.'.

```
autovacuum_enabled
autovacuum_vacuum_scale_factor
autovacuum_analyze_scale_factor
autovacuum_vacuum_threshold
autovacuum_analyze_threshold
autovacuum_freeze_max_age
autovacuum_vacuum_cost_delay
autovacuum_vacuum_cost_limit
autovacuum_freeze_min_age
autovacuum_freeze_table_age
autovacuum_max_workers
autovacuum_naptime
```

Old Free Space Map (before 8.4)

- A shared memory block
- Requires a stop/start database to adjust parameters
- Rebuilt every time you run VACUUM
- Lost on crash or PITR

Old Free Space Map

Parameters to adjust:

max_fsm_pages

max_fsm_relations

Based on output from VACUUM VERBOSE

Example VACUUM VERBOSE

```
INFO:  free space map contains 964 pages
in 608 relations

DETAIL:  A total of 10208 page slots are
in use (including overhead).
10208 page slots are required to track
all free space.

Current limits are:  204800 page slots,
1000 relations, using 1265 kB.
```

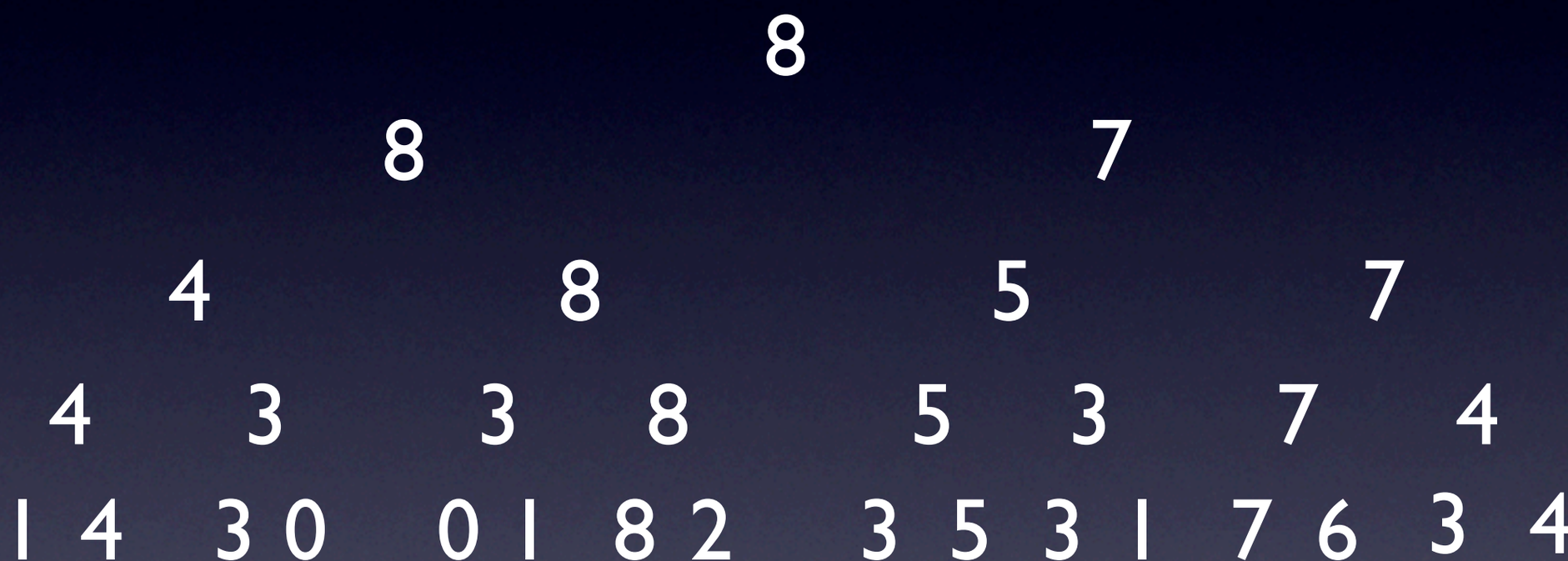
8.4

- Free Space Map and Visibility Map
- Heikki Linnakangas, lead developer
- Heikki's FOSDEM presentation: http://wiki.postgresql.org/wiki/Image:FSM_and_Visibility_Map.pdf

New Free Space Map

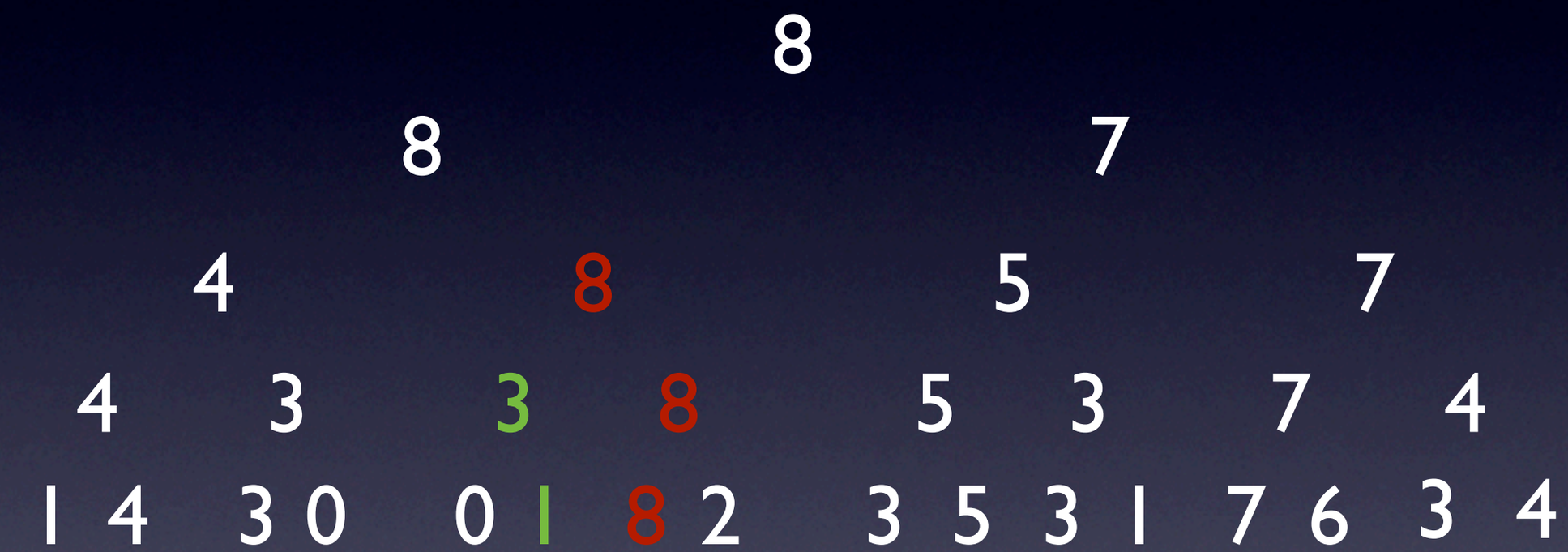
- Binary tree structure
- Stored on disk in normal 8k pages
- NO MORE CONFIGURATION

Free Space Map



PgCon 2009

Searching



Start at bottom.
Climb up for node >= 8
Climb down path to find the page with 8 blocks free.

Visibility Map

- A bitmap of heap pages
- 1 means "all tuples on page are visible to all transactions"
- Set during a VACUUM
- Cleared during INSERT, UPDATE, DELETE
- Failed to clear? Not a big deal.

Partial VACUUM in 8.4!

- Visibility Map lets VACUUM skip pages already marked as “visible”

Example

```
test=# CREATE TABLE test (id int4);
CREATE TABLE
test=# INSERT INTO test SELECT
generate_series(1,100000);
INSERT 0 100000
test=# delete from test where id < 50000;
DELETE 49999
```

Version 8.3 VACUUM

```
test=# VACUUM VERBOSE test;
INFO:  vacuuming "public.test"
INFO:  "test": removed 49999 row versions in
197 pages
INFO:  "test": found 49999 removable, 50001
nonremovable row versions in 393 pages
DETAIL:  0 dead row versions cannot be
removed yet.
There were 0 unused item pointers.
198 pages contain useful free space.
0 pages are entirely empty.
CPU 0.00s/0.00u sec elapsed 0.00 sec.
```

Version 8.3 VACUUM

```
test=# VACUUM VERBOSE test;
INFO:  vacuuming "public.test"
INFO:  "test": found 0 removable, 50001
nonremovable row versions in 393 pages
DETAIL:  0 dead row versions cannot be
removed yet.
There were 49999 unused item pointers.
198 pages contain useful free space.
0 pages are entirely empty.
CPU 0.00s/0.00u sec elapsed 0.00 sec.
VACUUM
```

Version 8.4 VACUUM

```
test=# VACUUM VERBOSE test;
INFO:  vacuuming "public.test"
INFO:  "test": found 0 removable, 8141 nonremovable
row versions in 228 out of 393 pages
DETAIL:  0 dead row versions cannot be removed yet.
There were 49999 unused item pointers.
0 pages are entirely empty.
CPU 0.00s/0.00u sec elapsed 0.00 sec.
VACUUM
```

```
test=# VACUUM VERBOSE test;
INFO:  vacuuming "public.test"
INFO:  "test": found 0 removable, 0 nonremovable row
versions in 31 out of 393 pages
DETAIL:  0 dead row versions cannot be removed yet.
There were 7905 unused item pointers.
0 pages are entirely empty (ADD AT THE END).
CPU 0.00s/0.00u sec elapsed 0.00 sec.
VACUUM
```

FSM & VM: file forks

```
lulu-2:11563 postgres$ ls -al | head
total 10408
drwx------  208 postgres  daemon    7072 Apr 30 22:37 .
drwx------    6 postgres  daemon     204 May  5 10:51 ..
-rw-------    1 postgres  daemon    8192 Apr 30 22:37 112
-rw-------    1 postgres  daemon    8192 Apr 30 22:37 113
-rw-------    1 postgres  daemon   57344 Apr 30 22:37 11447
-rw-------    1 postgres  daemon   24576 Apr 30 22:37 11447_fsm
-rw-------    1 postgres  daemon    8192 Apr 30 22:37 11447_vm
```

- Article about Free Space Map and Visibility Map: http://tr.im/hKnE

Questions?

Thanks!

- Tom for making lazy VACUUM!
- Heikki Linnakangas for rewriting the Free Space Map and his FOSDEM talk.
- Magnus Hagander for help with the MVCC slides.

VACUUM Strategy: Autovacuum, FSM, Visibility Map

Selena Deckelmann
End Point Corporation
selena@endpoint.com
@selenamarie