PostgreSQL-Consulting

data egret

Let's Pull the Plug on the Autovacuum!?

Alexey Lesovsky

alexey.lesovsky@dataegret.com

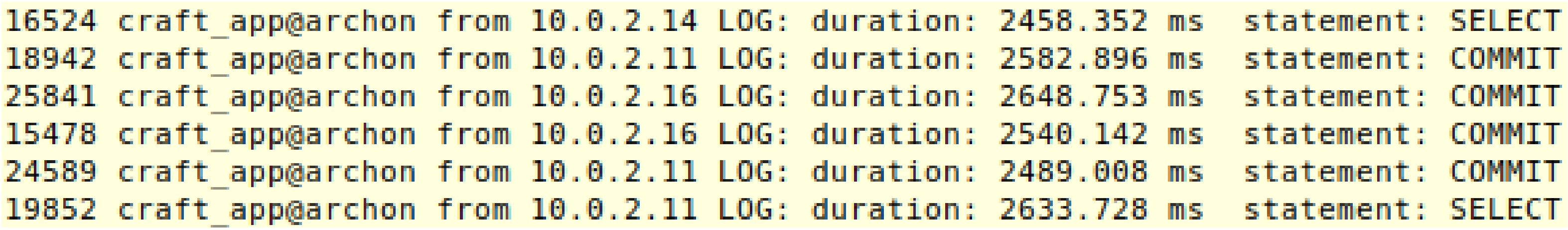
01 **Vacuum is the source of all problems!**

02 **Or is it a solution?**

01

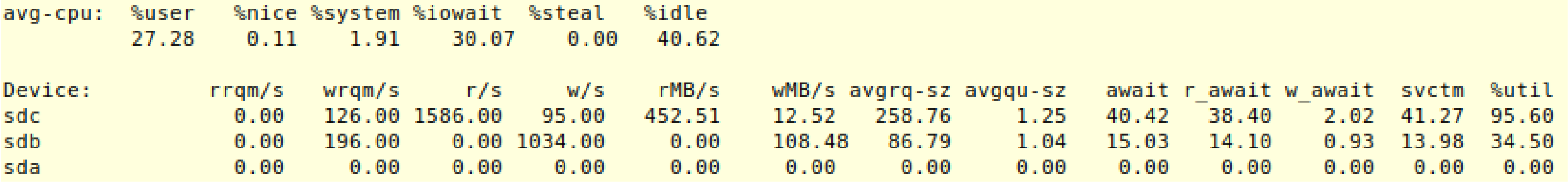
Vacuum
is the source
of all problems

01 Guys, what's wrong with the database?



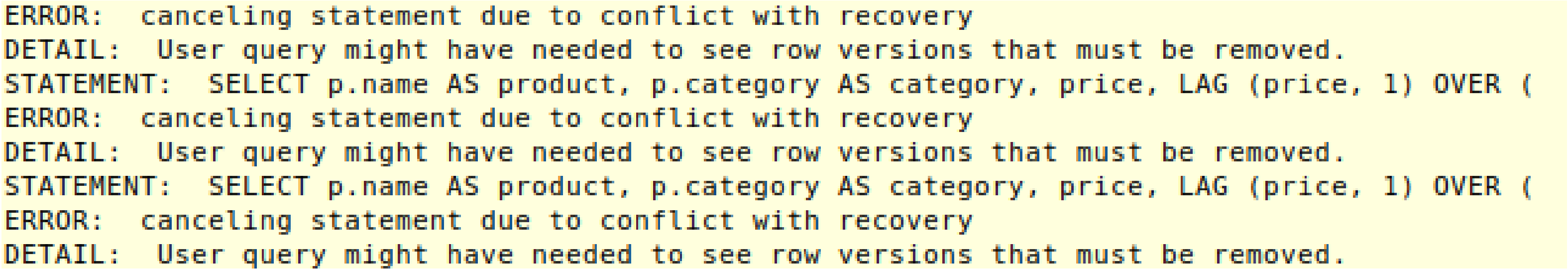
```
16524 craft_app@archon from 10.0.2.14 LOG: duration: 2458.352 ms  statement: SELECT
18942 craft_app@archon from 10.0.2.11 LOG: duration: 2582.896 ms  statement: COMMIT
25841 craft_app@archon from 10.0.2.16 LOG: duration: 2648.753 ms  statement: COMMIT
15478 craft_app@archon from 10.0.2.16 LOG: duration: 2540.142 ms  statement: COMMIT
24589 craft_app@archon from 10.0.2.11 LOG: duration: 2489.008 ms  statement: COMMIT
19852 craft_app@archon from 10.0.2.11 LOG: duration: 2633.728 ms  statement: SELECT
```

01 Guys, what's wrong with the database?



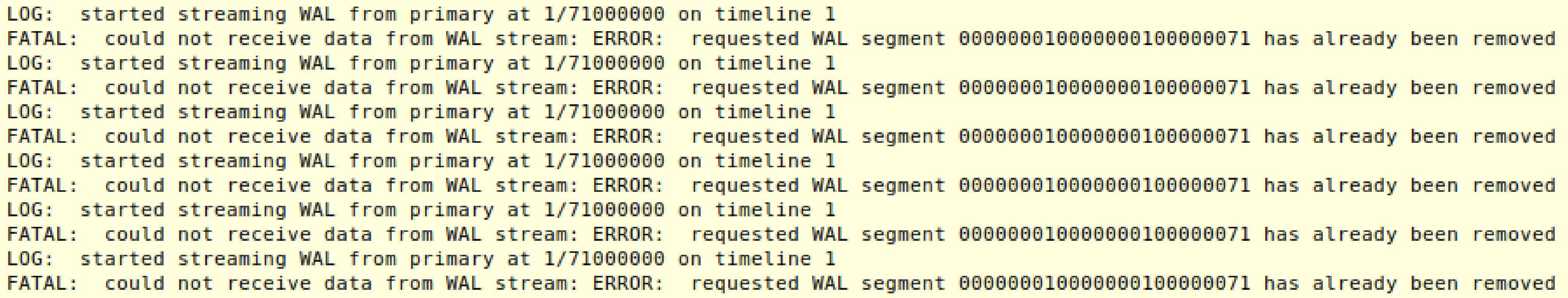
```
avg-cpu:  %user   %nice %system %iowait  %steal   %idle
          27.28    0.11    1.91   30.07    0.00   40.62

Device:         rrqm/s   wrqm/s     r/s     w/s    rMB/s    wMB/s avgrq-sz avgqu-sz   await r_await w_await  svctm  %util
sdc               0.00   126.00 1586.00   95.00   452.51    12.52   258.76     1.25   40.42   38.40    2.02  41.27  95.60
sdb               0.00   196.00    0.00 1034.00     0.00   108.48    86.79     1.04   15.03   14.10    0.93  13.98  34.50
sda               0.00     0.00    0.00    0.00     0.00     0.00     0.00     0.00    0.00    0.00    0.00   0.00   0.00
```

01 And queries are cancelled...



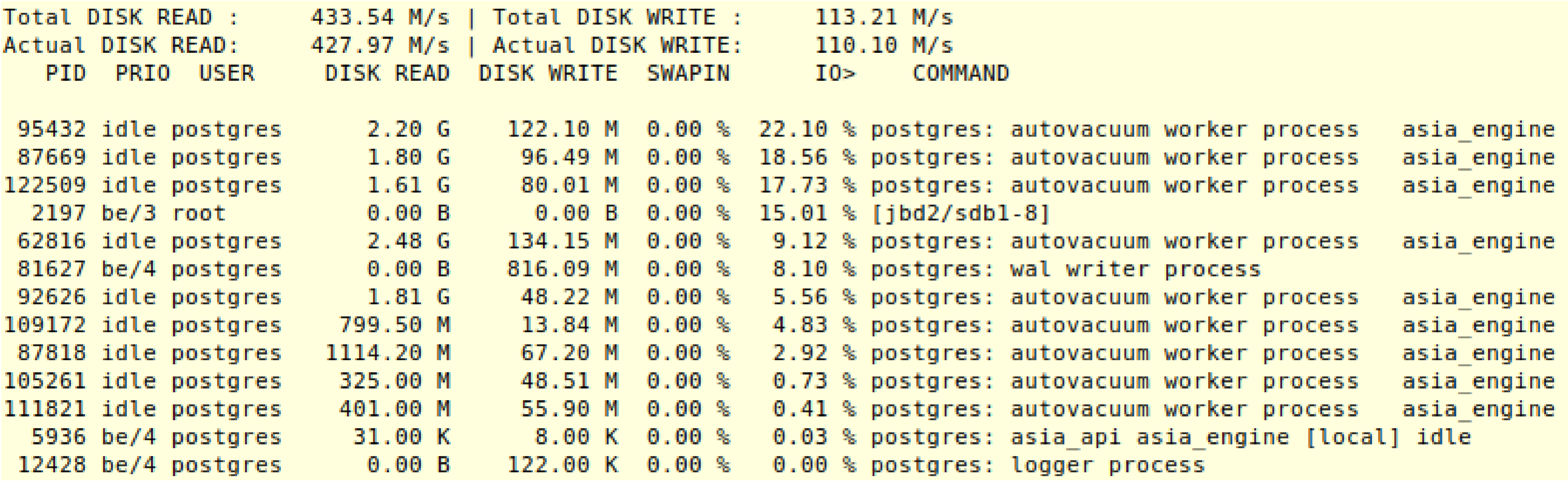
```
ERROR:  canceling statement due to conflict with recovery
DETAIL:  User query might have needed to see row versions that must be removed.
STATEMENT:  SELECT p.name AS product, p.category AS category, price, LAG (price, 1) OVER (
ERROR:  canceling statement due to conflict with recovery
DETAIL:  User query might have needed to see row versions that must be removed.
STATEMENT:  SELECT p.name AS product, p.category AS category, price, LAG (price, 1) OVER (
ERROR:  canceling statement due to conflict with recovery
DETAIL:  User query might have needed to see row versions that must be removed.
```

01 Where is our standby??



```
LOG:  started streaming WAL from primary at 1/71000000 on timeline 1
FATAL:  could not receive data from WAL stream: ERROR:  requested WAL segment 000000010000000100000071 has already been removed
LOG:  started streaming WAL from primary at 1/71000000 on timeline 1
FATAL:  could not receive data from WAL stream: ERROR:  requested WAL segment 000000010000000100000071 has already been removed
LOG:  started streaming WAL from primary at 1/71000000 on timeline 1
FATAL:  could not receive data from WAL stream: ERROR:  requested WAL segment 000000010000000100000071 has already been removed
LOG:  started streaming WAL from primary at 1/71000000 on timeline 1
FATAL:  could not receive data from WAL stream: ERROR:  requested WAL segment 000000010000000100000071 has already been removed
LOG:  started streaming WAL from primary at 1/71000000 on timeline 1
FATAL:  could not receive data from WAL stream: ERROR:  requested WAL segment 000000010000000100000071 has already been removed
LOG:  started streaming WAL from primary at 1/71000000 on timeline 1
FATAL:  could not receive data from WAL stream: ERROR:  requested WAL segment 000000010000000100000071 has already been removed
```

01 It seems vacuums kill everything



```
Total DISK READ :     433.54 M/s | Total DISK WRITE :     113.21 M/s
Actual DISK READ:     427.97 M/s | Actual DISK WRITE:     110.10 M/s
   PID  PRIO  USER     DISK READ  DISK WRITE  SWAPIN      IO>    COMMAND

 95432 idle postgres      2.20 G    122.10 M  0.00 %  22.10 % postgres: autovacuum worker process   asia_engine
 87669 idle postgres      1.80 G     96.49 M  0.00 %  18.56 % postgres: autovacuum worker process   asia_engine
122509 idle postgres      1.61 G     80.01 M  0.00 %  17.73 % postgres: autovacuum worker process   asia_engine
  2197 be/3 root          0.00 B      0.00 B  0.00 %  15.01 % [jbd2/sdb1-8]
 62816 idle postgres      2.48 G    134.15 M  0.00 %   9.12 % postgres: autovacuum worker process   asia_engine
 81627 be/4 postgres      0.00 B    816.09 M  0.00 %   8.10 % postgres: wal writer process
 92626 idle postgres      1.81 G     48.22 M  0.00 %   5.56 % postgres: autovacuum worker process   asia_engine
109172 idle postgres    799.50 M     13.84 M  0.00 %   4.83 % postgres: autovacuum worker process   asia_engine
 87818 idle postgres   1114.20 M     67.20 M  0.00 %   2.92 % postgres: autovacuum worker process   asia_engine
105261 idle postgres    325.00 M     48.51 M  0.00 %   0.73 % postgres: autovacuum worker process   asia_engine
111821 idle postgres    401.00 M     55.90 M  0.00 %   0.41 % postgres: autovacuum worker process   asia_engine
  5936 be/4 postgres     31.00 K      8.00 K  0.00 %   0.03 % postgres: asia_api asia_engine [local] idle
 12428 be/4 postgres      0.00 B    122.00 K  0.00 %   0.00 % postgres: logger process
```

So, what to do?

01 Solution!?

autovacuum = off

01 Everything looks ok now, but...

- Forget about query planner statistics.

01 Everything looks ok now, but...

- Forget about query planner statistics.
- Tables and indexes bloat.

01 Everything looks ok now, but...

- Forget about query planner statistics.
- Tables and indexes bloat.
- Innefficient usage of shared buffers.

01 Everything looks ok now, but...

- Forget about query planner statistics.
- Tables and indexes bloat.
- Innefficient usage of shared buffers.
- Performance slowdown.

01 Hmm, everything is not so well

Test case – https://goo.gl/Tql87l

- Before: 3565.5 tps, 0.839 ms, 3% of shared_buffers.
- After: 172.8 tps, 17.373 ms, 21% of shared_buffers.

02

Vacuum?
That sounds familiar

02 Why does postgres need a vacuum?

Does everyone know about MVCC?

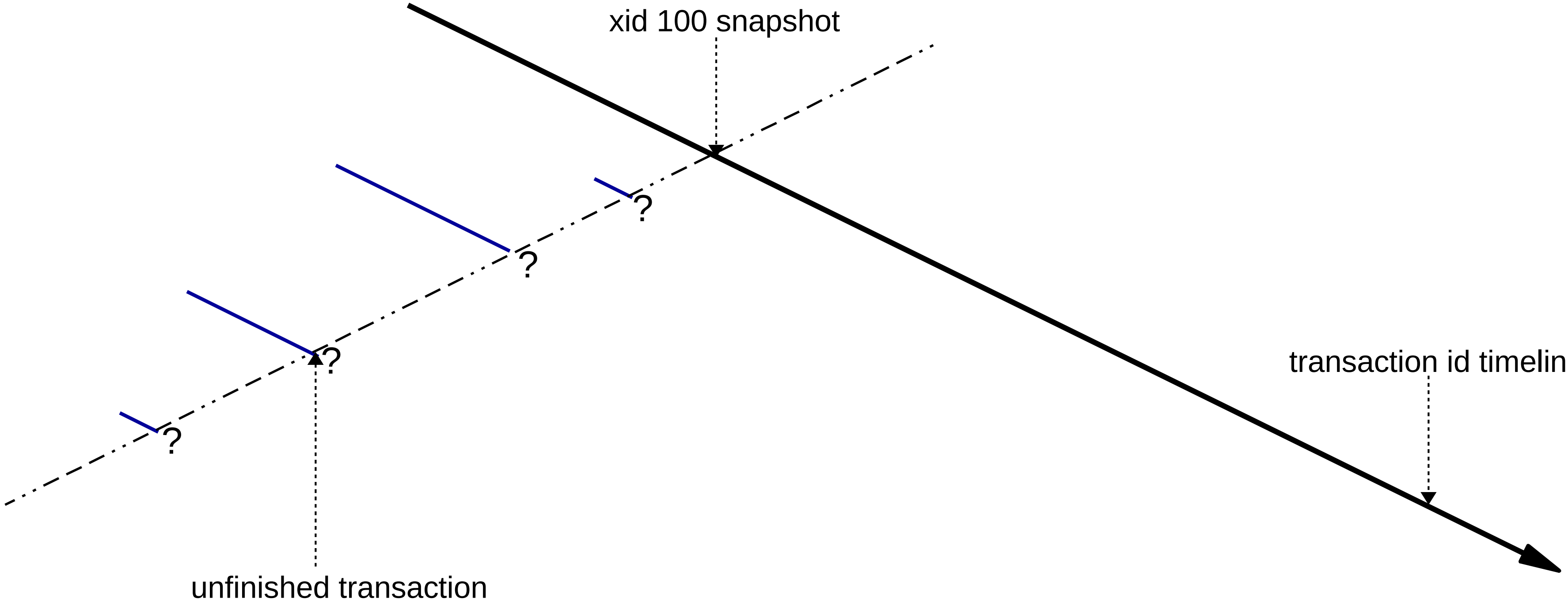
02 Why does postgres need a vacuum?

MVCC – Multi-Version Concurrency Control:

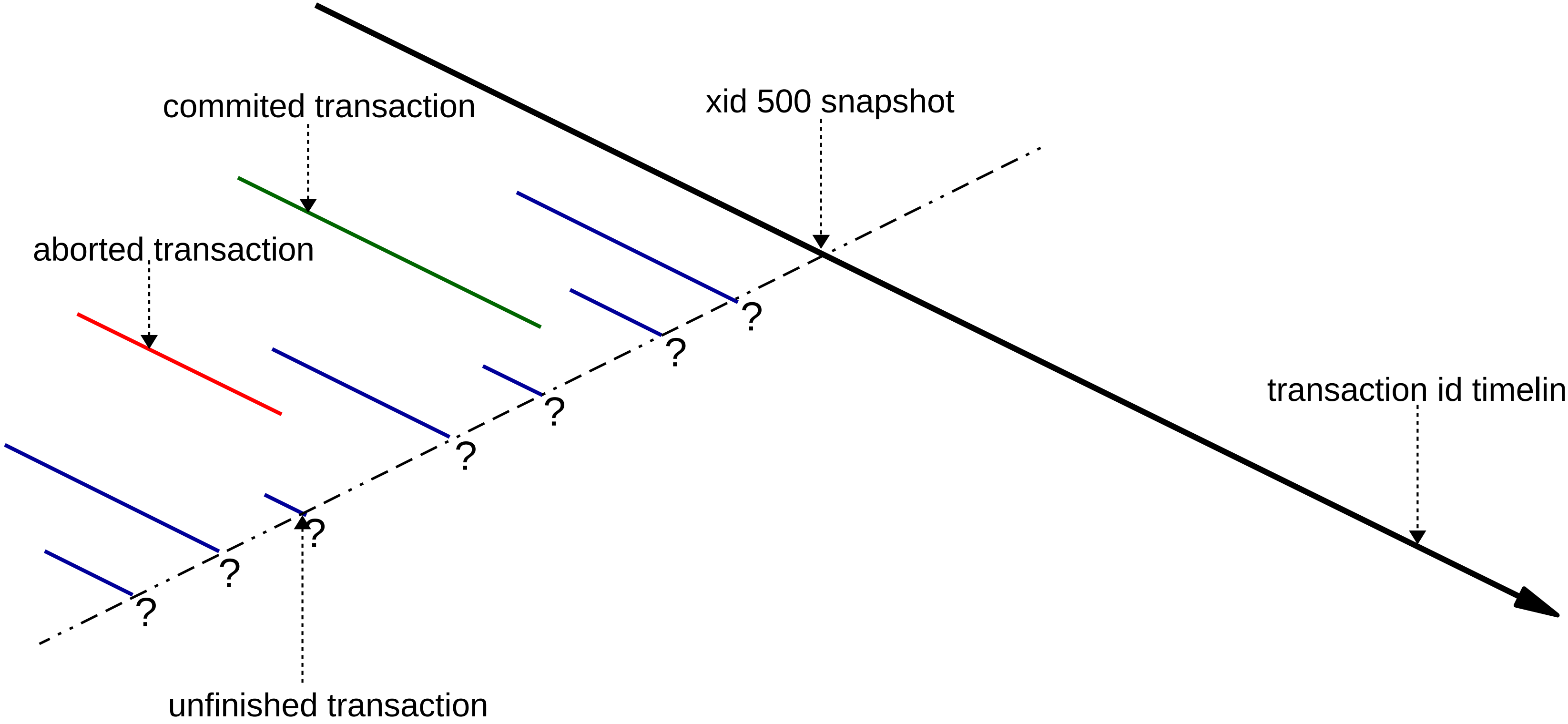
- Good performance with concurrent access.
- On intensive read and write access.
- Readers don't block readers; Writers don't block writers*.

* *of course there are always exceptions.*

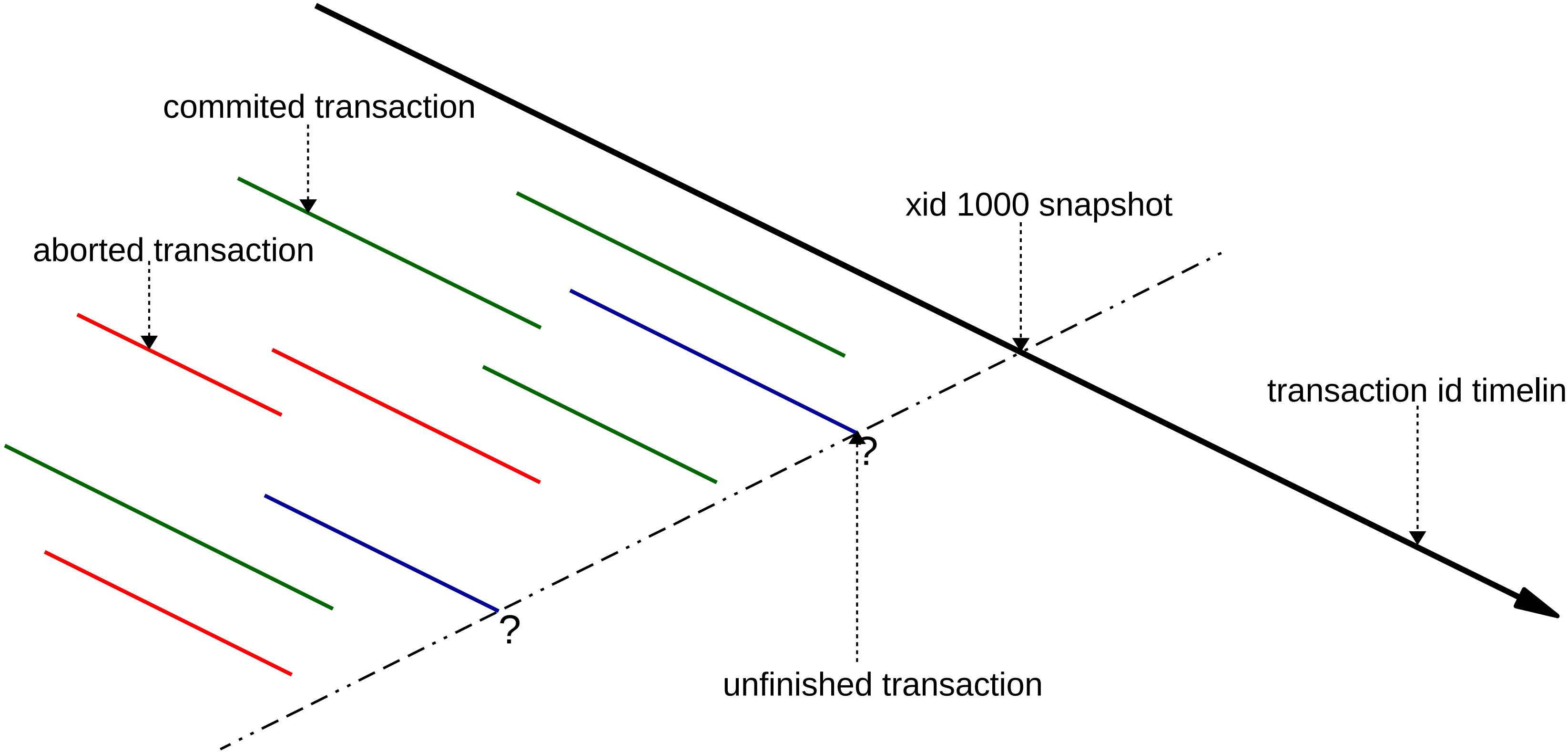
02 MVCC


xid 100 snapshot
?
?
?
?
transaction id timeline
unfinished transaction

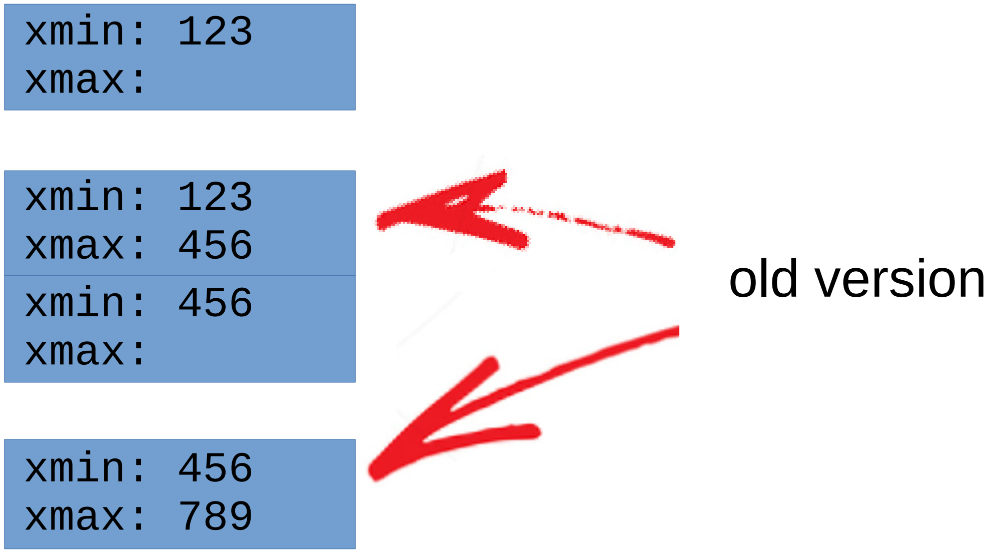
02 MVCC


commited transaction
xid 500 snapshot
aborted transaction
?
?
?
?
?
?
?
transaction id timeline
unfinished transaction

02 MVCC


commited transaction
aborted transaction
xid 1000 snapshot
transaction id timeline
?
?
unfinished transaction

02 MVCC



INSERT row by xact 123

```
xmin: 123
xmax:
```

UPDATE row by xact 456

```
xmin: 123
xmax: 456
```

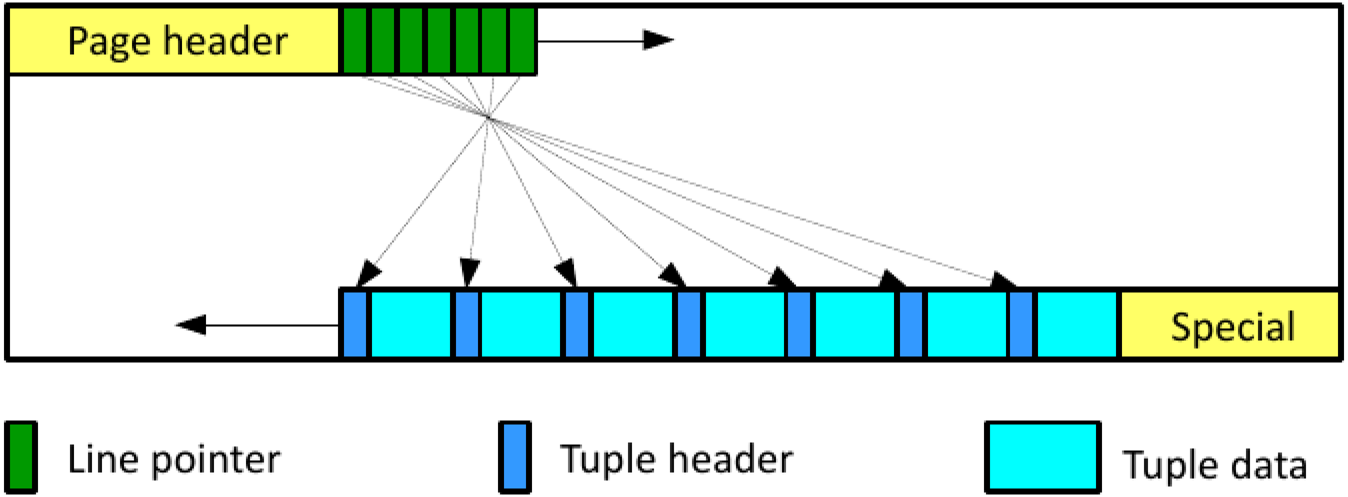
```
xmin: 456
xmax:
```

DELETE row by xact 789

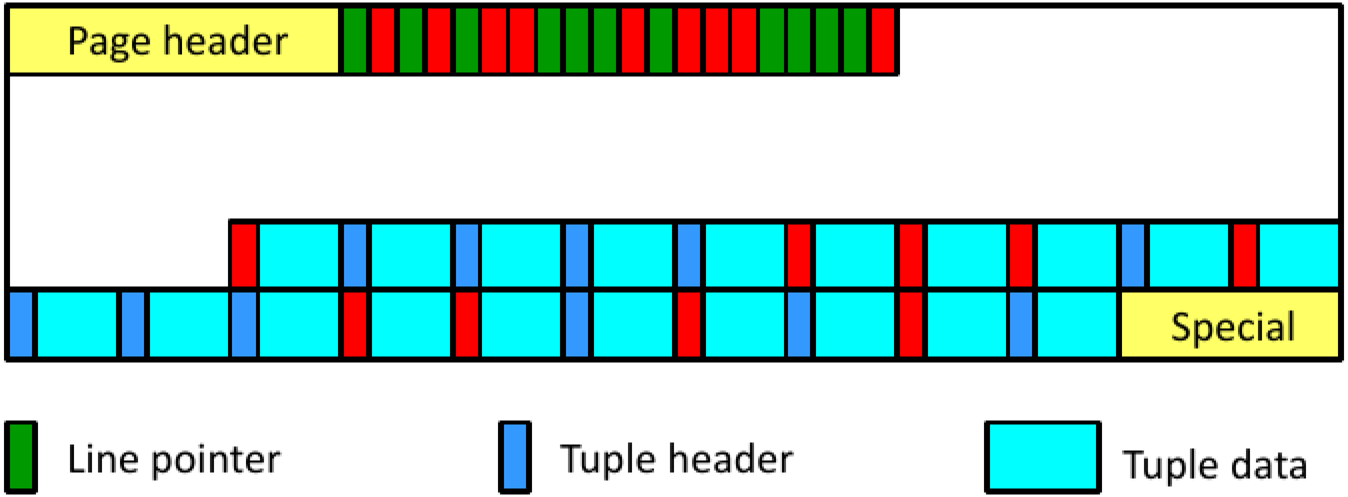
```
xmin: 456
xmax: 789
```

old version

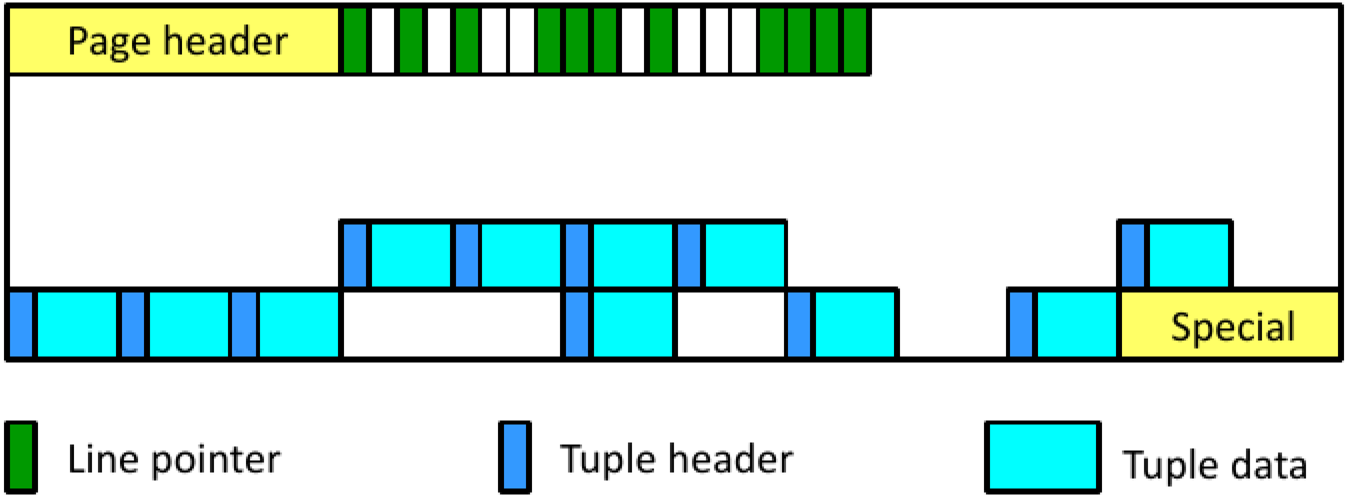
02 What about data pages?



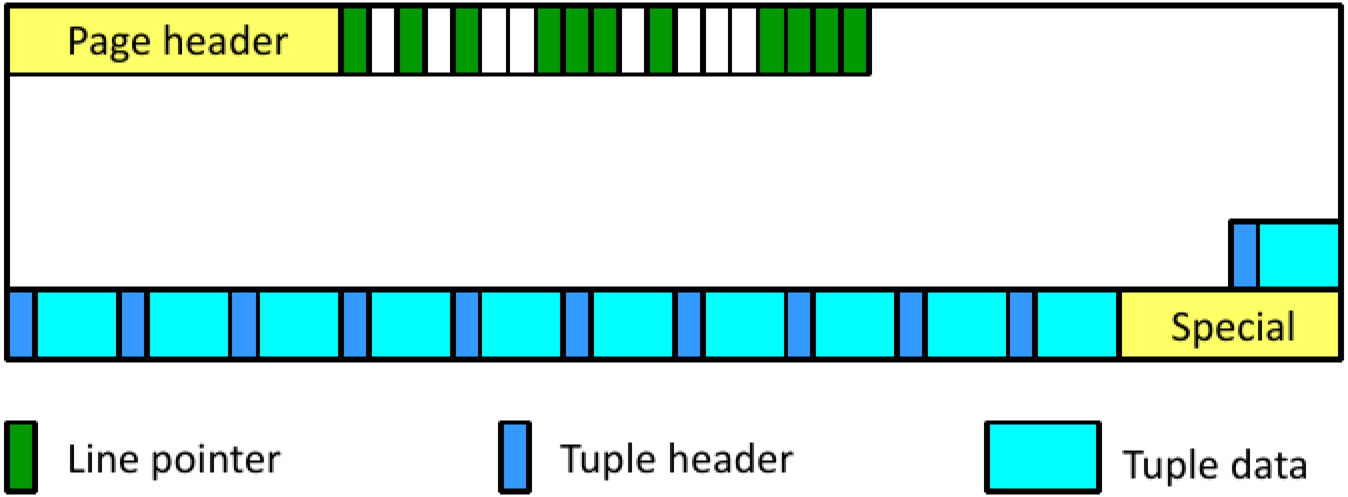
02 What about data pages?



02 What about data pages?



02 What about data pages?



02 Why do you need the vacuum

To keep size of tables in optimal.

To minimize *bloat*.

To keep actual data in shared buffers.

To keep performance.

02 Autovacuum key points

Autovacuum is the **background** task/process:

- Enabled by default, but limited by the number of concurrent workers.
- Starts with an interval.
- Cleans tables/indexes and gathers statistics for query planner.

02 Autovacuum key points

Processes databases/tables/indexes by list:

- Databases with risk of *wraparound*.
- Databases which aren't processed for a while.
- Tables with many dead tuples.

02 Autovacuum key points

Avoid using default settings on modern hardware.

Vacuum becomes better from version to version.

03

Right,
Let's tune the autovacuum

03 How to begin?

1. there can be multiple workers at a time:

- *autovacuum_max_workers*
- *autovacuum_naptime*

03 How to begin?

2. launch of the vacuum depends on the number of «*dead*» tuples:

- *autovacuum_vacuum_threshold*
- *autovacuum_vacuum_scale_factor*

03 How to begin?

2. launch of the vacuum depends on the number of «*dead*» tuples:

- *autovacuum_vacuum_threshold*
- *autovacuum_vacuum_scale_factor*

n_dead_tup > (reltuples * scale_factor) + threshold

03 How can we use it?

Scale factor vs. *Threshold*

03 How to begin?

3. (auto)vacuum is mostly cost-based:

- *vacuum_cost_limit* – divides between all **active** workers.
- *vacuum_cost_delay*
- *vacuum_cost_page_hit*
- *vacuum_cost_page_miss*
- *vacuum_cost_page_dirty*

03 Different storage?

- HDD – yes, they are still used.

03 Different storage?

- HDD – yes, they are still used.
- SSD – even their performance sometimes is not sufficient.

03 Different storage?

- HDD – yes, they are still used.
- SSD – even their performance sometimes is not sufficient.
- NVMe – what are you doing here??

03 Is there a rule of thumb?

Tune *delay* and *limit*.

03 Example for SSD

```
vacuum_cost_delay = 0
vacuum_cost_page_hit = 0
vacuum_cost_page_miss = 5
vacuum_cost_page_dirty = 5
vacuum_cost_limit = 200
--
autovacuum_max_workers = 10
autovacuum_naptime = 1s
autovacuum_vacuum_threshold = 50
autovacuum_analyze_threshold = 50
autovacuum_vacuum_scale_factor = 0.05
autovacuum_analyze_scale_factor = 0.05
autovacuum_vacuum_cost_delay = 5ms
autovacuum_vacuum_cost_limit = -1
```

03 Application's approach

Avoid *long* or *idle transactions*.

Use *statement_timeout*, *idle_in_transaction_session_timeout*.

Use cron tasks when timeouts are not flexible.

Avoid long or idle transactions.

03 Application's approach

Avoid too *long queries*, or *too long* or *idle transactions*.

Use *statement_timeout, idle_in_transaction_session_timeout.*

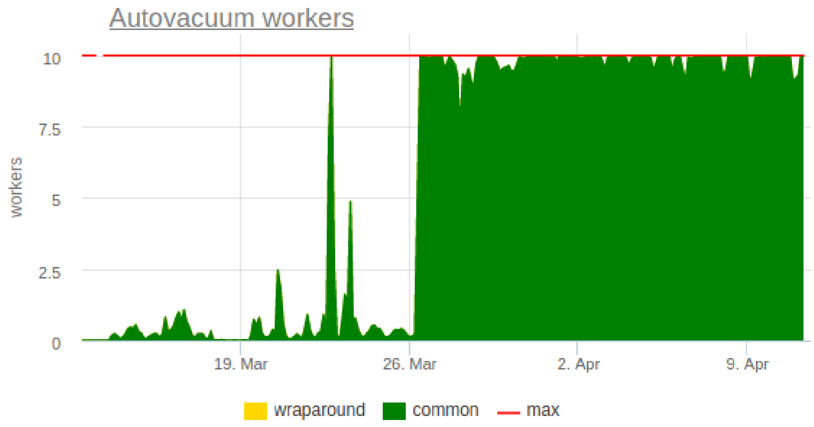
Use cron tasks when timeouts are not flexible.

03 Application's approach

Avoid regular '*LOCK TABLE*'.

03 Application's approach

```
20470 [LOCK TABLE waiting] LOG:  process 20470 still waiting for ExclusiveLock on relation 2627 of database 9939 after 1000.049 ms
20470 [LOCK TABLE waiting] DETAIL:  Process holding the lock: 20950. Wait queue: 20470.
20470 [LOCK TABLE waiting] STATEMENT:  LOCK TABLE user_ips in EXCLUSIVE MODE
20950 [] ERROR:  canceling autovacuum task
20950 [] CONTEXT:  automatic vacuum of table "synapse03.public.user_ips"
20470 [LOCK TABLE waiting] LOG:  process 20470 acquired ExclusiveLock on relation 2627 of database 9939 after 1000.125 ms
20470 [LOCK TABLE waiting] STATEMENT:  LOCK TABLE user_ips in EXCLUSIVE MODE
20470 [LOCK TABLE] LOG:  duration: 1000.195 ms  statement: LOCK TABLE user_ips in EXCLUSIVE MODE
```



03 Alternative medicine

pgcompacttable – «slow», lightweight, safe.

pg_repack – fast, simple, stable, but sometimes «unsafe».

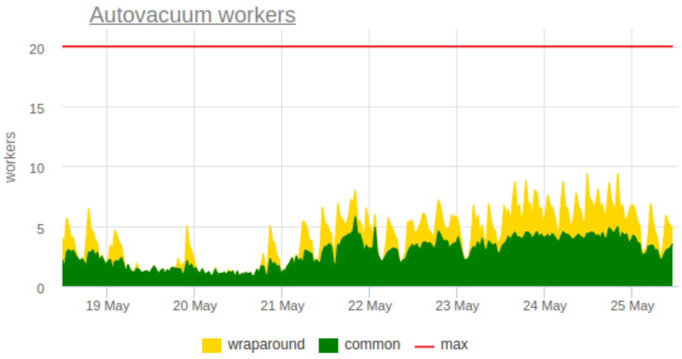
A few words about monitoring…

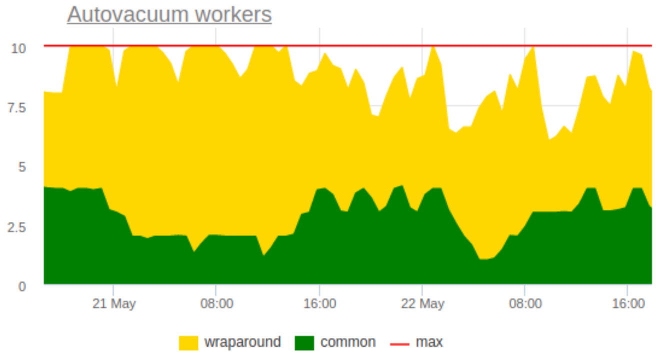
03 How to monitor vacuum

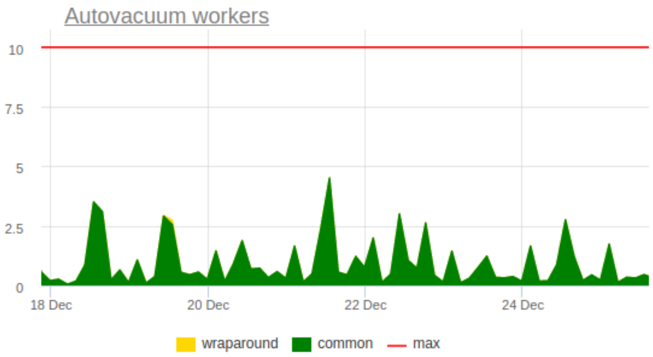
pg_stat_activity – must be in every monitoring.

- Number and type of workers.
- Workers' age.

03 How to monitor vacuum







03 How to monitor vacuum

pg_stat_progress_vacuum – when it's important to check the details.

03 How to monitor vacuum

```
table pg_stat_progress_vacuum;

-[ RECORD 1 ]------+------------------
pid                | 143080
datid              | 120319
datname            | maindb
relid              | 184299
phase              | vacuuming heap
heap_blks_total    | 22520662
heap_blks_scanned  | 22520662
heap_blks_vacuumed | 3544974
index_vacuum_count | 1
max_dead_tuples    | 178956970
num_dead_tuples    | 1531080
```

```
http://bit.do/vacuum_activity

-[ RECORD 1 ]------+------------------
pid                | 143080
duration           | 00:17:00.434357
waiting            | f
mode               | wraparound
database           | maindb
table              | data.executions
phase              | vacuuming heap
table_size         | 172 GB
total_size         | 299 GB
scanned            | 172 GB
vacuumed           | 27 GB
scanned_pct        | 100.0
vacuumed_pct       | 15.7
index_vacuum_count | 1
dead_pct           | 0.9
```

03 And finally

- Vacuum is a good guy.
- Vacuum is NOT tricky.
- **Don't disable autovacuum.**

03 And finally

- Vacuum is a good guy.
- Vacuum is NOT tricky.
- **Don't disable autovacuum.**
- **Avoid idle transactions.**

Thanks!