Getting the most out of pg_stat_io

About me

Daniel Westermann

Principal Consultant

Technology Leader Open Infrastructure

+41 79 927 2446

daniel.westermann[at]dbi-services.com

https://www.linkedin.com/in/daniel-westermann/

@danielwestermann@mastodon.social

All pictures in the slides from: https://unsplash.com/

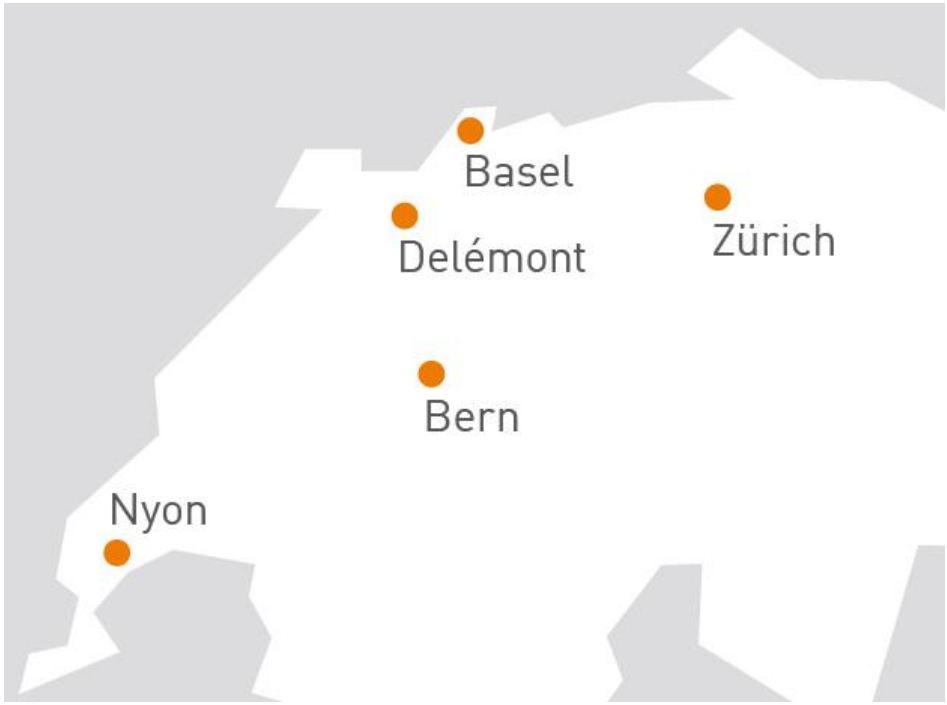
Who we are

The Company

- Founded in 2010
- More than 100 employees
- Specialized in the Middleware Infrastructure
  - The invisible part of IT
- Customers in Switzerland and all over Europe

Our Offer

- Consulting
- Service Level Agreements (SLA)
- Trainings
- License Management



swiss safety center ISO 20000 certified system

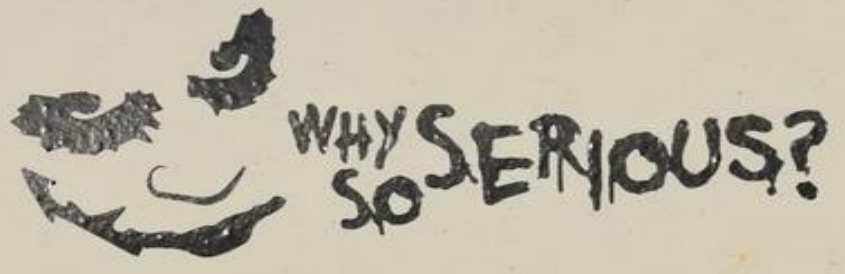
swiss safety center ISO 27001 certified system


Who does not know
pg_stat_statements?

Don't

You're not in the wrong talk

The reason I'm asking is…



pg_stat_statements
Why always 0 for four columns?

There are four columns which are empty(0) by default

```
postgres=# select version();
                                                  version
-------------------------------------------------------------------------------------------------------
 PostgreSQL 16.0 on x86_64-pc-linux-gnu, compiled by gcc (Debian 12.2.0-14) 12.2.0, 64-bit
(1 row)

postgres=# \o | egrep "blk_write_time|blk_read_time"
postgres=# \d pg_stat_statements
 blk_read_time          | double precision |           |          |
 blk_write_time         | double precision |           |          |
 temp_blk_read_time     | double precision |           |          |
 temp_blk_write_time    | double precision |           |          |
```

pg_stat_statements
Why always 0 for four columns?

There are four columns which are empty(0) by default

```
postgres=# select count(*)
             from pg_stat_statements
            where blk_read_time > 0
               or blk_write_time > 0
               or temp_blk_read_time > 0
               or temp_blk_write_time > 0;
 count
-------
     0
(1 row)
```

- You might easily verify this with the statement above
  - (if you did not change the default configuration)

pg_stat_statements
Why always 0 for four columns?

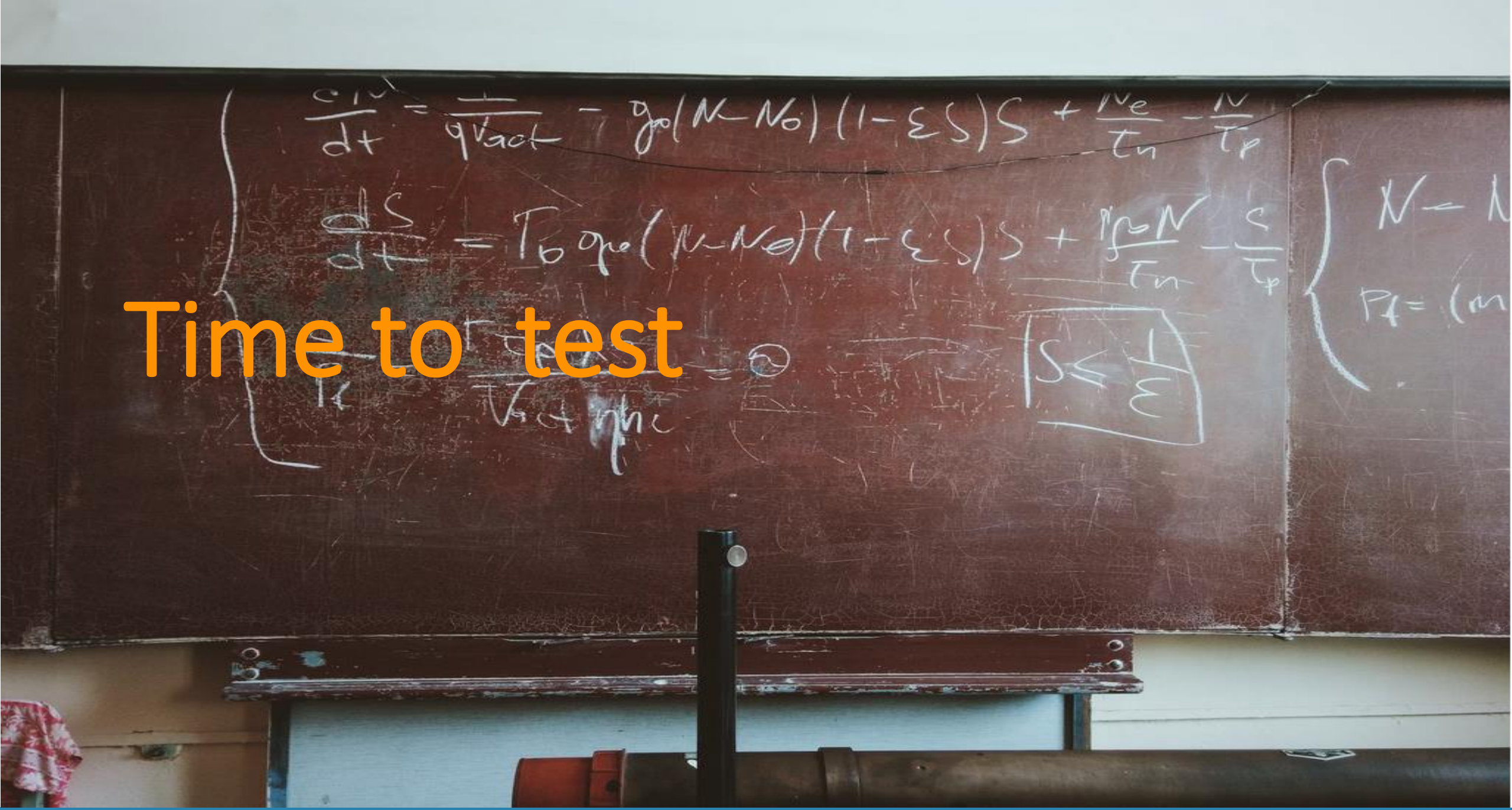
The reason is this:

```
postgres=# show track_io_timing;
 track_io_timing
-----------------
 off
(1 row)
```

This parameter is off by default,

> "as it will repeatedly query the operating system for the current time,

> which may cause significant overhead on some platforms"


Time to test

… but how?

pg_test_timing
Tell me the truth

Say "hello" to pg_test_timing

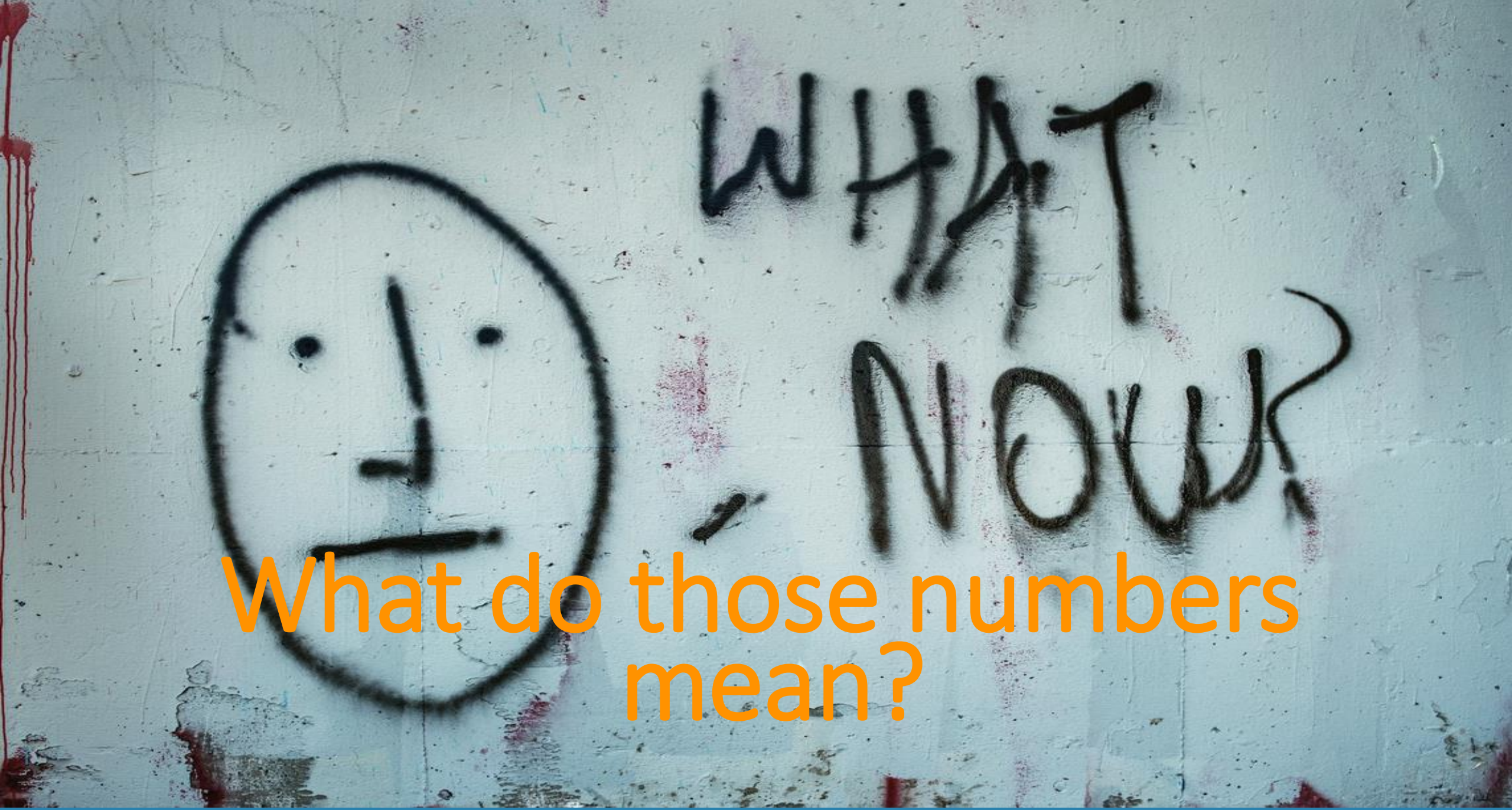
```
$ pg_test_timing --help
Usage: pg_test_timing [-d DURATION]
```

pg_test_timing is a tool to measure the timing overhead on your system

> and confirm that the system time never moves backwards

> systems that are slow to collect timing data can give less accurate EXPLAIN ANALYZE results

```
$ pg_test_timing -d 5
Testing timing overhead for 5 seconds.
Per loop time including overhead: 53.75 ns
Histogram of timing durations:
  < us   % of total      count
     1     94.64253   88038542
     2      5.35433    4980712
     4      0.00016        145
     8      0.00269       2506
…
```

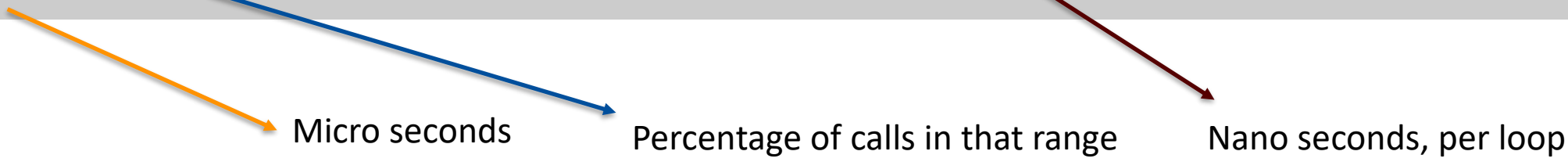


What do those numbers mean?

pg_test_timing
Tell me the truth

What do those numbers mean?

```
$ pg_test_timing -d 5
Testing timing overhead for 5 seconds.
Per loop time including overhead: 53.75 ns
Histogram of timing durations:
  < us   % of total      count
     1     94.64253   88038542
     2      5.35433    4980712
     4      0.00016        145
     8      0.00269       2506
...
```



pg_test_timing
Tell me the truth

The recommendation (from the docs)

```
$ pg_test_timing -d 5
Testing timing overhead for 5 seconds.
Per loop time including overhead: 53.75 ns
Histogram of timing durations:
  < us   % of total      count
     1     94.64253   88038542
     2      5.35433    4980712
     4      0.00016        145
     8      0.00269       2506
…
```

> "Good results will show most (>90%) individual timing calls take less than one microsecond"


How does this relate to …
pg_stat_io ?

pg_stat_io
The same story

Some columns will be empty (0) by default there as well

```
postgres=# \o | grep \_time
postgres=# \d pg_stat_io
 read_time       | double precision         |           |          |
 write_time      | double precision         |           |          |
 writeback_time | double precision         |           |          |
 extend_time     | double precision         |           |          |
 fsync_time      | double precision         |           |          |
```

- The same story here
  - Without track_io_timing = true/1/on/yes, there will be no I/O related statistics

Ready to get started?

(1) - Extends

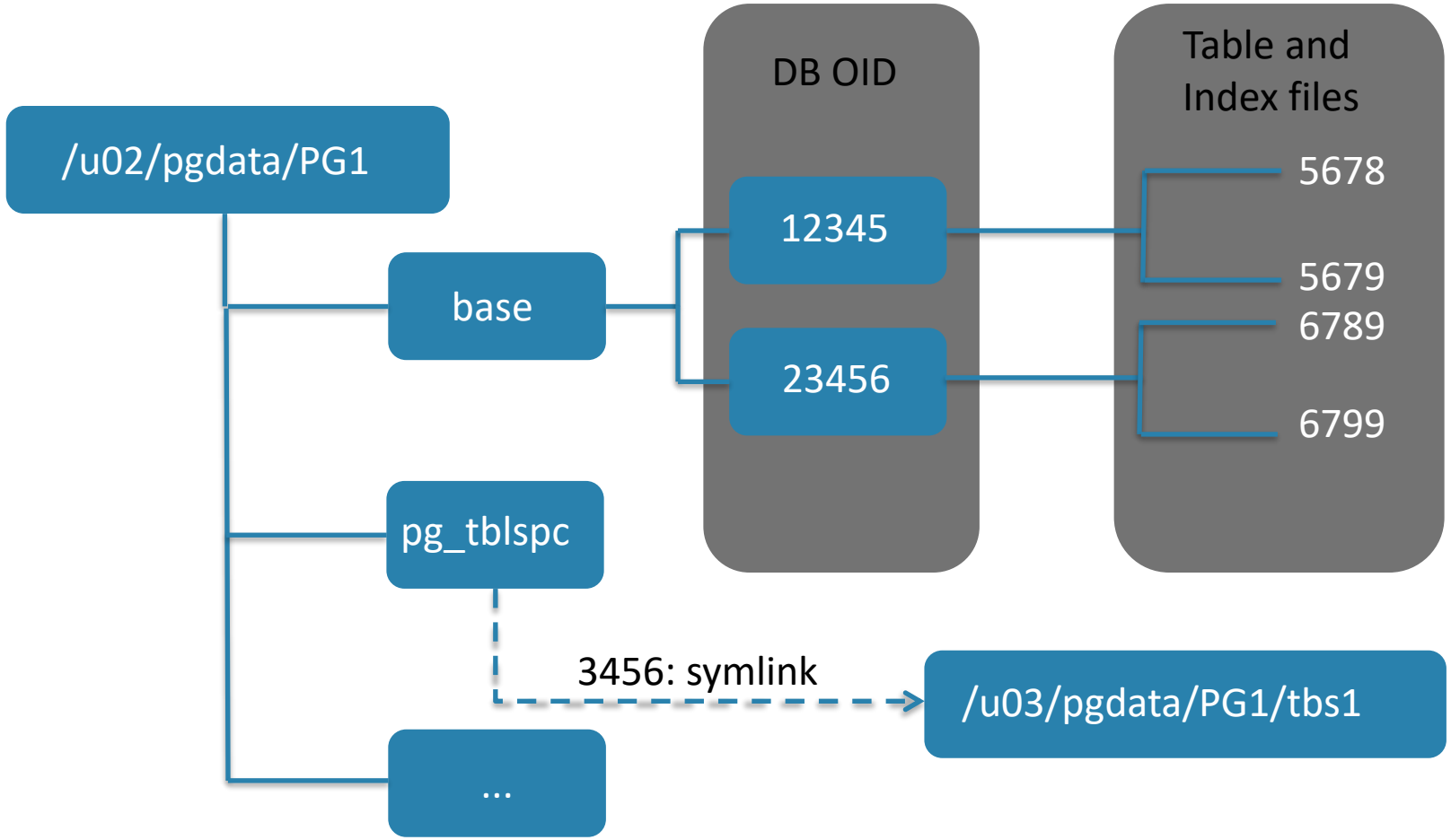
pg_stat_io
Extends

How does PostgreSQL organize data files on disk?

```
$ pg_config | grep -i segsize
CONFIGURE =  '--prefix=/u01/app/postgres/product/16/db_0/' '--exec-
prefix=/u01/app/postgres/product/16/db_0/' '--
bindir=/u01/app/postgres/product/16/db_0//bin' '--
libdir=/u01/app/postgres/product/16/db_0//lib' '--
sysconfdir=/u01/app/postgres/product/16/db_0//etc' '--
includedir=/u01/app/postgres/product/16/db_0//include' '--
datarootdir=/u01/app/postgres/product/16/db_0//share' '--
datadir=/u01/app/postgres/product/16/db_0//share' '--with-pgport=5432' '--with-perl' '--
with-python' '--with-openssl' '--with-pam' '--with-ldap' '--with-libxml' '--with-libxslt'
'--with-segsize=1' '--with-blocksize=8' '--with-llvm' 'LLVM_CONFIG=/usr/bin/llvm-config'
'--with-uuid=ossp' '--with-lz4' '--with-zstd' '--with-gssapi' '--with-systemd' '--with-
icu' '--with-system-tzdata=/usr/share/zoneinfo' '--with-extra-version= dbi services build'
```

- The default segment size is 1GB
  - This can be changed at compilation time

Overview of the data directory



pg_stat_io
Extends

How does PostgreSQL organize data files on disk?

```
$ cd $PGDATA
$ oid2name
All databases:
  Oid  Database Name  Tablespace
----------------------------------
    5       postgres  pg_default
    4      template0  pg_default
    1      template1  pg_default
$ psql -c "create table t1 ( a int )"
CREATE TABLE
$ psql -c "select pg_relation_filepath('t1')"
 pg_relation_filepath
----------------------
 base/5/16388
(1 row)
$ ls -l base/5/16388
-rw------- 1 postgres postgres 0 Nov  3 17:05 base/5/16388
```

pg_stat_io
Extends

We do start from scratch, resetting all pg_stat_io related statistics

```
$ psql -c "select pg_stat_reset_shared('io')"
 pg_stat_reset_shared
----------------------

(1 row)
$ psql -c "select backend_type
                , object
                , context
                , extends
             from pg_stat_io
            where extends > 0;"
 backend_type | object | context | extends
--------------+--------+---------+---------
(0 rows)
```

pg_stat_io
Extends

The arguments for pg_stat_reset_shared are:

Argument	Meaning
bgwriter	reset all the counters shown in the pg_stat_bgwriter view
archiver	reset all the counters shown in the pg_stat_archiver view
io	reset all the counters shown in the pg_stat_io view
wal	reset all the counters shown in the pg_stat_wal view
recovery_prefetch	reset all the counters shown in the pg_stat_recovery_prefetch view

- Starting with PostgreSQL 17
  - Calling the function without an argument will reset all shared statistics listed above

pg_stat_io
Extends

What happens if we start to insert data into our small test table?

```
$ psql -c "insert into t1 values(1)"
INSERT 0 1
$ ls -l base/5/16388
-rw------- 1 postgres postgres 8192 Dec  8 20:15 base/5/16388
```

> 8192 bytes is 8kB, which is the default block size of PostgreSQL

```
$ pg_config | grep -i blocksize
CONFIGURE =  '--prefix=/u01/app/postgres/product/16/db_0/' '--exec-
…datarootdir=/u01/app/postgres/product/16/db_0//share' '--
datadir=/u01/app/postgres/product/16/db_0//share' '--with-pgport=5432' '--with-perl' '--
with-python' '--with-openssl' '--with-pam' '--with-ldap' '--with-libxml' '--with-libxslt'
'--with-segsize=1' '--with-blocksize=8' '--with-llvm' 'LLVM_CONFIG=/usr/bin/llvm-config'
'--with-uuid=ossp' '--with-lz4' '--with-zstd' '--with-gssapi' '--with-systemd' '--with-
icu' '--with-system-tzdata=/usr/share/zoneinfo' '--with-extra-version= dbi services build'
```

pg_stat_io
Extends

What we see here is exactly one extend

```
$ psql -c "select backend_type
                , object
                , context
                , extends
                , op_bytes
             from pg_stat_io
            where extends > 0"
 backend_type    |  object   | context | extends | op_bytes
-----------------+----------+---------+---------+----------
 client backend | relation | normal  |       1 | 8192
(1 row)
```

> PostgreSQL will extend (increase) the data file(s) by 8kB when it is required

> Required means: If new space is needed, an additional 8kb is added to the data file

pg_stat_io
Extends

The meaning of those columns

Column	Description
backend_type	The same as in pg_stat_activity (client backend, ….)
object	Relation or temp(orary) relation
context	normal - reads and writes from/to shared buffers vacuum - I/O operations performed outside of shared buffers (vacuuming and analyzing) blkread - certain large read I/O operations done outside of shared buffers, e.g a large seq scan blkwrite - certain large write I/O operations done outside of shared buffers, such as COPY.
extends	Number of relation extend operations, each of the size specified in op_bytes (usually 8kB).

pg_stat_io
Extends

Generating more extends

```
$ psql -c "insert into t1 select *
                          from generate_series(2,1000)"
INSERT 0 999
$ ls -l base/5/16388
-rw------- 1 postgres postgres 40960 Nov 29 11:10 base/5/16388
$ psql -c "select backend_type
                , object
                , context
                , extends
                , op_bytes
             from pg_stat_io
            where extends > 0"
 backend_type   |  object  | context | extends | op_bytes
----------------+----------+---------+---------+----------
 client backend | relation | normal  |       8 |     8192
```

pg_stat_io
Extends

What looks strange here?

```
 backend_type    |  object   | context | extends | op_bytes
-----------------+----------+---------+---------+----------
 client backend | relation | normal  |       8 |     8192
```

Let's do the math

```
$ psql -c "select 8*8192 as bytes"
  bytes
----------
    65536
(1 row)
$ ls -la base/5/24588
-rw------- 1 postgres postgres 40960 Dec 10 10:32 base/5/16388
```

> This doesn't match


Ideas?

pg_stat_io
Extends

Not only the data files need to be extended

```
$ ls -la base/5/24588*
-rw------- 1 postgres postgres 40960 Dec 10 10:32 base/5/24588
-rw------- 1 postgres postgres 24576 Dec 10 10:32 base/5/24588_fsm
```

> There is the free space map

> ... and

```
$ psql -c "vacuum t1"
VACUUM
$ ls -la base/5/24588*
-rw------- 1 postgres postgres 40960 Dec 10 10:32 base/5/24588
-rw------- 1 postgres postgres 24576 Dec 10 10:32 base/5/24588_fsm
-rw------- 1 postgres postgres  8192 Dec 10 10:43 base/5/24588_vm
```

> There is the visibility map

pg_stat_io
Extends

Finally: What is this metric good for?

> If you combine that with extend_time (if track_io_timing is on)

> This gives you an idea how much time was spend for extending relation files

```
$ psql -c "alter system set track_io_timing=on" -c "select pg_reload_conf()"
VACUUM
 pg_reload_conf
----------------
 t
(1 row)
$ psql -c "insert into t1 select * from generate_series(1001,2000)"
INSERT 0 1000
$ psql -c "select extends
                 , extend_time
              from pg_stat_io where extends > 0"
 extends | extend_time
---------+-------------
      13 |       0.045
(1 row)
```

pg_stat_io
Extends

Finally: What is this metric good for?

- In a long running cluster / instance
  - When there are much more extends than writes

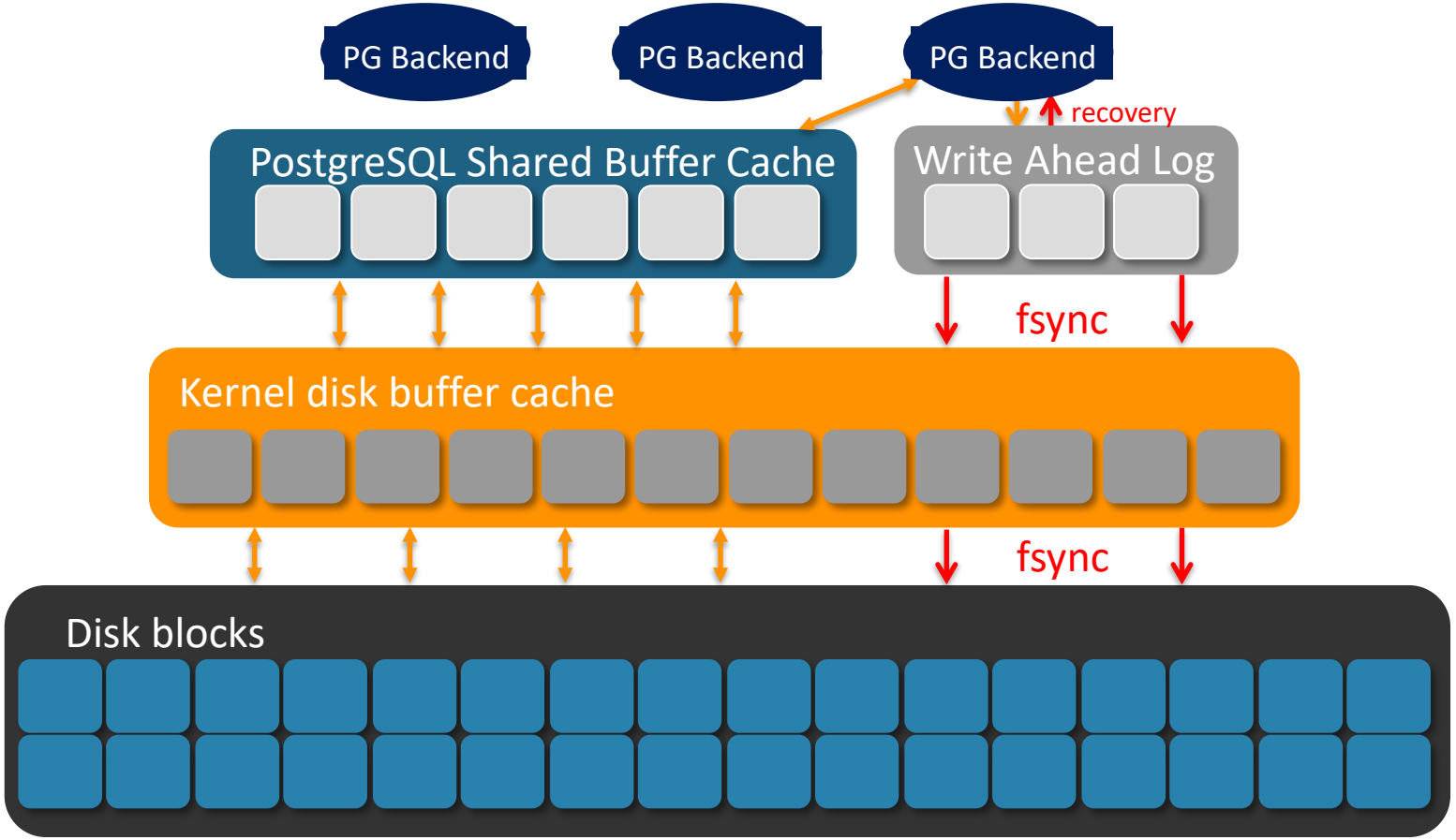
```
$ psql -c "select backend_type, writes,extends
             from pg_stat_io
            where writes > 0 or extends > 0"
  backend_type   | writes | extends
-----------------+--------+---------
 client backend |      0 |      13
 checkpointer   |     87 |
(2 rows)
```

  - This might mean, that autovacuum is not able to keep up
  - Usually autovacuum/vacuum is freeing space in the relation files and usually writes do not require additional extends

(2) - Evictions

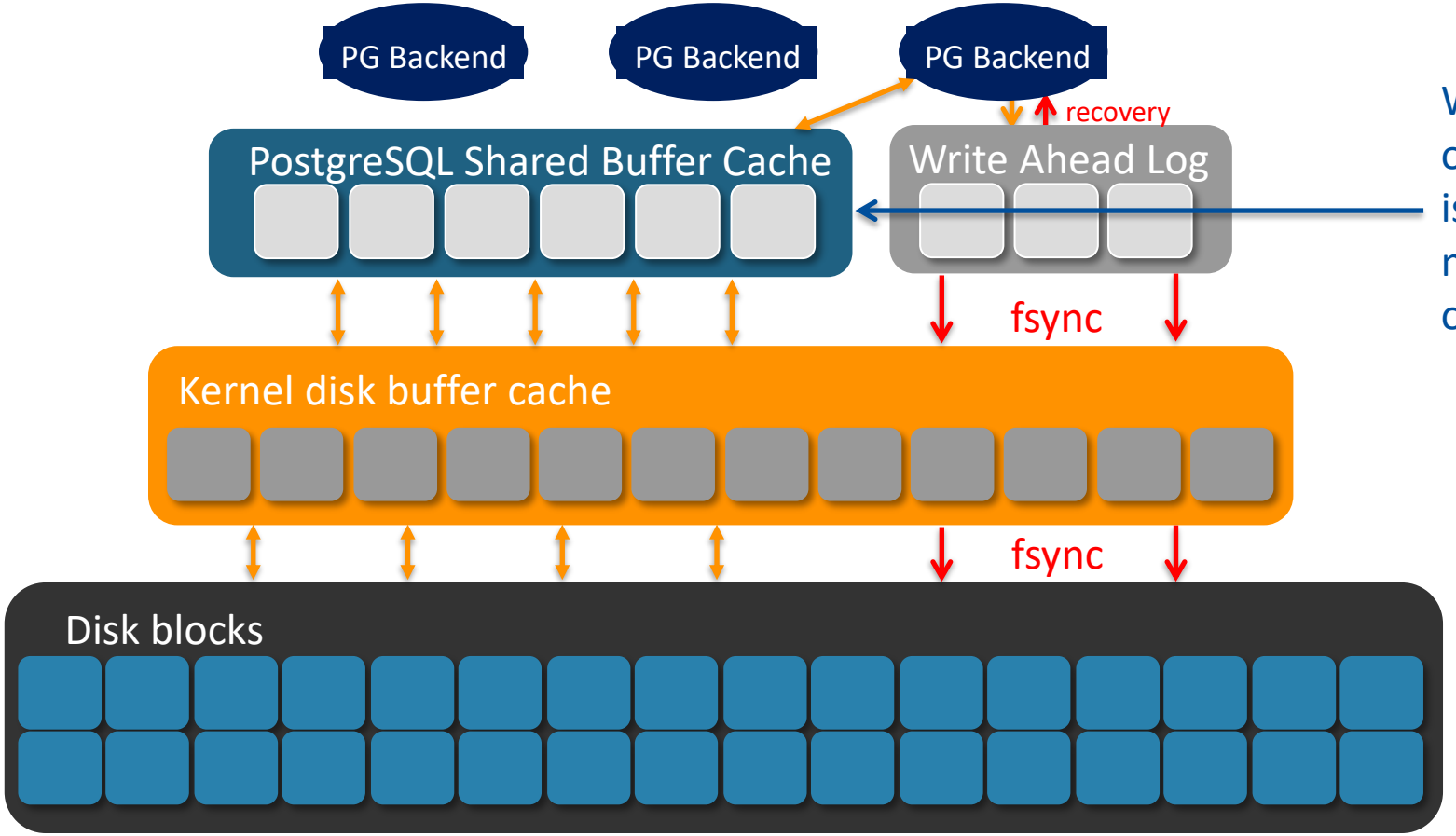
pg_stat_io
Evictions

What does "evictions" mean?



pg_stat_io
Evictions

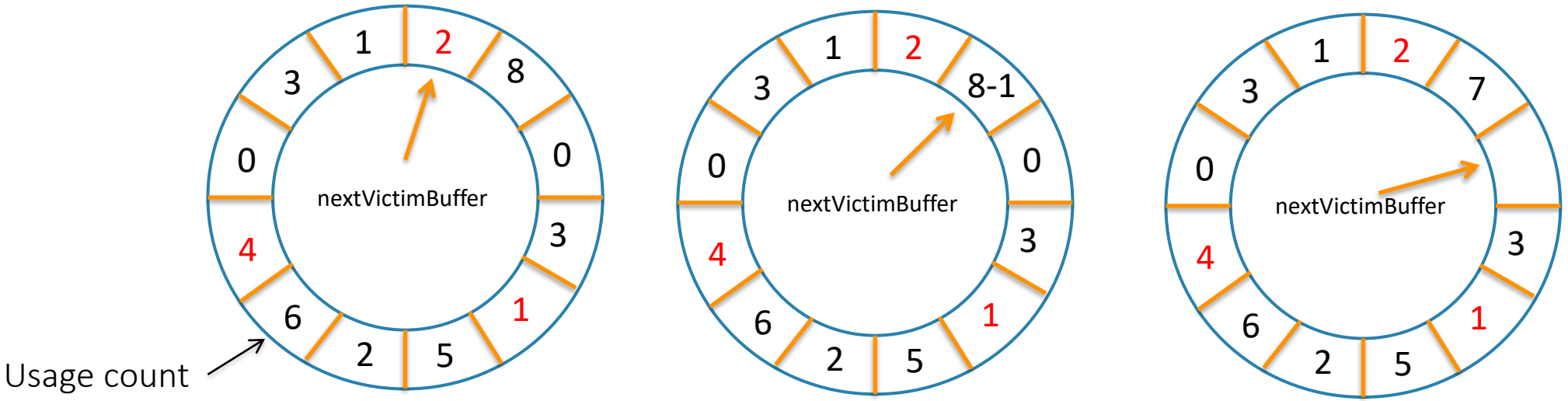
What does "evictions" mean?



What happens,
once the buffer cache
is full and new blocks
need to be read from disk
or OS cache?

The clock-sweep algorithm, clockwise rotation

> If there are unpinned buffers in the cache, clock-sweep can always find a victim



pinned - someone is doing something with the buffer

pg_stat_io
Evictions

There is a very good description of this in the source code

```
$ grep -ir "clock sweep" * | awk -F ":" '{print $1}' | uniq
src/backend/storage/buffer/README
src/backend/storage/buffer/localbuf.c
src/backend/storage/buffer/bufmgr.c
src/backend/storage/buffer/freelist.c
src/include/storage/buf_internals.h
$ vi src/backend/storage/buffer/README
…
To choose a victim buffer to recycle when there are no free
buffers available, we use a simple clock-sweep algorithm, which avoids the
need to take system-wide locks during common operations.  It works like
this:
…
```

pg_stat_io
Evictions

What does "evictions" mean, summary:

- Once the PostgreSQL buffer cache is full
  - Blocks/pages needs to be evicted from the cache
    - To make room for new blocks to be read from disk
  - Kicking blocks/pages out of the buffer cache is called "evictions"
- PostgreSQL uses a simple clock-sweep algorithm to implement this
  - A victim is found by decreasing the usage count
    - Only for buffers which are not currently pinned
  - The usage count gets increased whenever a buffer is pinned

pg_stat_io
Evictions

pg_stat_io comes with metrics for evictions

```
$ psql -c "select backend_type
                , object
                , context
                , evictions
             from pg_stat_io
            where evictions > 0"
 backend_type | object | context | evictions
--------------+--------+---------+-----------
(0 rows)
```

> This means that there wasn't any evictions in my instance

pg_stat_io
Evictions

To get any numbers for this you need to know the size of the buffer cache

```
$ psql -c "show shared_buffers"
 shared_buffers
----------------
 128MB
(1 row)
```

- 128MB is the default on Linux systems
- To see any evictions we need to fill the cache
  - So PostgreSQL is forced to kick out buffers from the cache to make room for new buffers

pg_stat_io
Evictions

Before we do that, let's look at the current content of the cache

> The standard extension "pg_buffercache" can be used for that

```
$ psql -c "create extension pg_buffercache"
CREATE EXTENSION
$ psql -c "select count(*) as \"8k\" from pg_buffercache;"
  8k
-------
 16384
(1 row)
$ psql -c "select (16384*8)/1024 as MB"
 mb
-----
 128
(1 row)
```

> This is exactly the size of our cache (shared_buffers)

pg_stat_io
Evictions

How does the clock sweep usage/access count look like currently?

```
$ psql -c "select count(*) from pg_buffercache where usagecount > 0"
 count
-------
   422
(1 row)
```

> This means only 422 buffers out of the 16384 buffershave been used right now, mostly for internal relations

```
$ psql -c "select c.relname, count(*) AS buffers
            from pg_class c inner join pg_buffercache b ON b.relfilenode=c.relfilenode
           inner join pg_database d on (b.reldatabase=d.oid
             and d.datname=current_database())
           group by c.relname
           oderby 2 desc limit 10;
 relname                | buffers
--------------------------------+---------
 pg_statistic                   |      14
 pg_operator                    |      14
…
```

pg_stat_io
Evictions

Let's force some evictions

```
$ pgbench -i -s 100
dropping old tables...
creating tables...
generating data (client-side)...
10000000 of 10000000 tuples (100%) done (elapsed 6.63 s, remaining 0.00 s)
vacuuming...
creating primary keys...
done in 9.24 s (drop tables 0.00 s, create tables 0.01 s, client-side generate 6.65 s,
vacuum 0.14 s, primary keys 2.44 s).

$ psql -c "select backend_type,object,context,evictions
             from pg_stat_io
            where evictions > 0"
 backend_type | object | context | evictions
--------------+--------+---------+-----------
(0 rows)
```

> Not a single buffer was evicted


Ideas?

pg_stat_io
Evictions

When reading or writing a huge relation

> PostgreSQL uses a ring buffer instead of the buffer pool
> The ring buffer is a temporary buffer in shared memory
> The allocated ring buffer is released immediately after use

The conditions for using a ring buffer

> The relation size exceeds ¼ of the buffer pool size, ring buffer = 256 KB
> Execution of the following commands, ring buffer = 16 MB
  > COPY FROM
  > CREATE TABLE AS SELECT
  > CREATE/REFRESH MATERIALIZED VIEW
  > ALTER TABLE
> Vacuum, ring buffer = 256 KB
  > Autovacuum performs vacuum

pg_stat_io
Evictions

Forcing evictions by pre-loading pgbench_accounts

```
$ psql -c "create extension pg_prewarm"
CREATE EXTENSION
$ psql -c "select pg_prewarm ( 'pgbench_accounts'::regclass
                             , 'buffer'
                             , 'main' , null, null )"
 pg_prewarm
------------
     163935
(1 row)
$ psql -c "select pg_size_pretty(pg_relation_size('pgbench_accounts'))"
 pg_size_pretty
----------------
 1281 MB
(1 row)
```

> The size of pgench_accounts is much larger than the buffer cache, which is 128MB

pg_stat_io
Evictions

Forcing evictions by pre-loading pgbench_accounts

```
$ psql -c "select backend_type
                , object
                , context
                , evictions
             from pg_stat_io
            where evictions > 0"
   backend_type    |  object  | context | evictions
-------------------+----------+---------+-----------
 autovacuum worker | relation | normal  |        36
 client backend    | relation | normal  |    150232
(2 rows)
```

> We can also see that an autovacuum worker process caused some evictions

pg_stat_io
Evictions

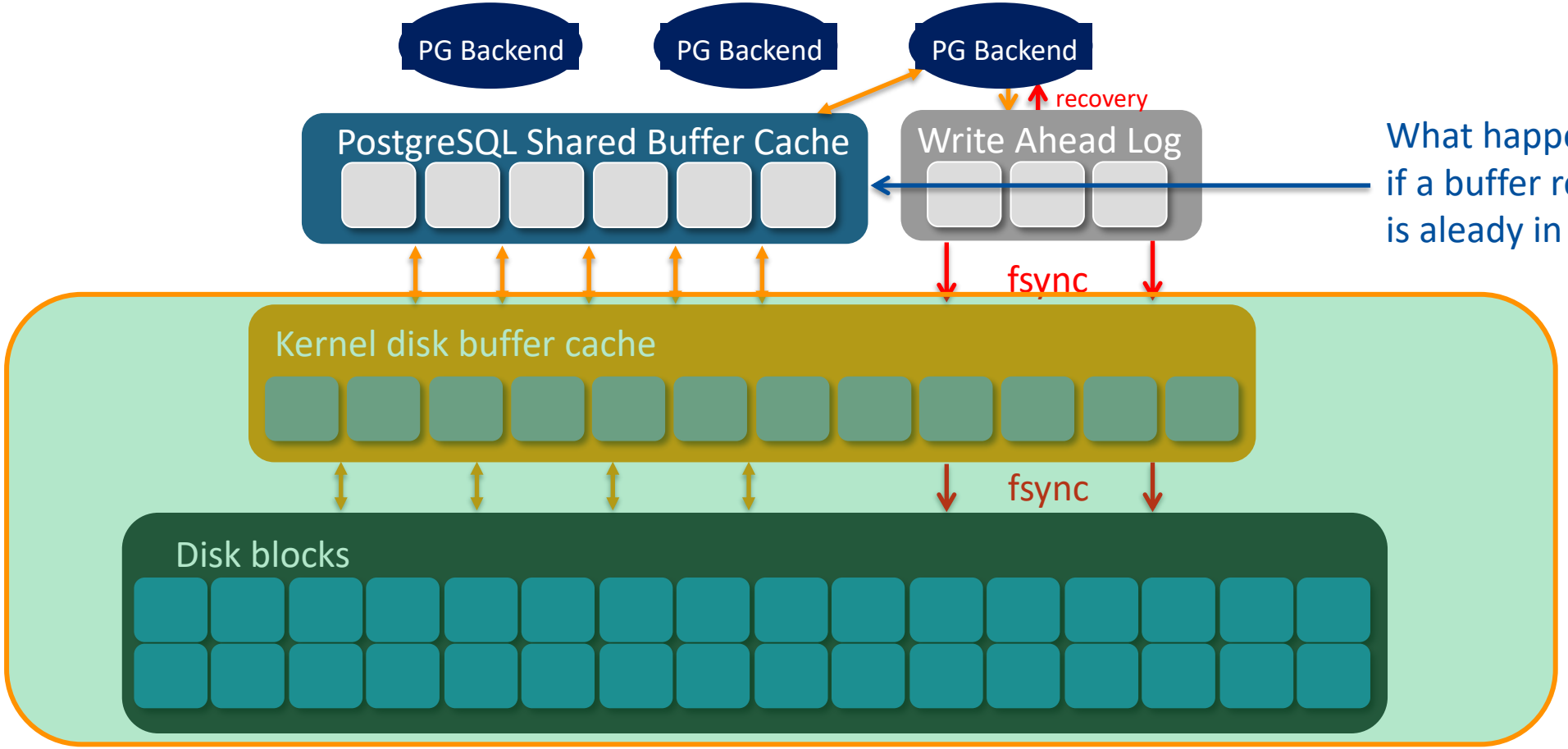
Finally: What is this metric good for?

- A high number of evictions can mean
  - The current size of the buffer cache is too small
  - or, in other words
  - It might be a good idea to increase shared_buffers
- When PostgreSQL needs to make free space in the buffer cache constantly
  - This also causes cache contention
  - Many processes / sessions compete against the same resource

(3) - hits

pg_stat_io
Hits

When there are evictions, there also must be hits



What happens,
if a buffer requested
is aleady in the cache?

No need to read
from either the
disk or the OS
cache

pg_stat_io
Hits

When there are evictions, there also must be hits

> "hits" mean: A requested buffer is already in the cache
  > It can be used immediately without requesting the buffer from disk or OS cache
  > The more hits you'll see, the less reads against relation data files you'll see

```
$ psql -c "select backend_type, object, context, hits
             from pg_stat_io
            where hits > 0"
  backend_type     |  object  |  context  |  hits
-------------------+----------+-----------+--------
 autovacuum worker | relation | normal    |  17125
 autovacuum worker | relation | vacuum    |     14
 client backend    | relation | bulkread  |   1355
 client backend    | relation | bulkwrite | 161371
 client backend    | relation | vacuum    |    429
 background worker | relation | bulkread  |    688
 background worker | relation | normal    |    139
(8 rows)
```

pg_stat_io
Hits

What does bulk* mean?

```
$ psql -c "select backend_type, object, context, hits
             from pg_stat_io
            where hits > 0"
  backend_type      |  object  |  context   |  hits
--------------------+----------+------------+--------
 autovacuum worker  | relation | normal     |  17125
 autovacuum worker  | relation | vacuum     |     14
 client backend     | relation | bulkread   |   1355
 client backend     | relation | bulkwrite  | 161371
 client backend     | relation | vacuum     |    429
 background worker  | relation | bulkread   |    688
 background worker  | relation | normal     |    139
(8 rows)
```

- bulkread: Certain large read I/O operations done outside of shared buffers
  - e.g., a sequential scan of a large table
- bulkwrite: Certain large write I/O operations done outside of shared buffers
  - such as COPY

pg_stat_io
Hits

Starting from scratch once more

```
$ psql -c "select pg_stat_reset_shared('io')"
 pg_stat_reset_shared
----------------------

(1 row)

$ psql -c "select backend_type
                , object
                , context
                , hits
             from pg_stat_io
            where hits > 0"
  backend_type   |  object   | context | hits
-----------------+----------+---------+------
 client backend | relation | normal  |   80
```

> This already produced some hits

pg_stat_io
Hits

To watch this in action: In one session, always touch the same buffers

```
$ psql
psql (16.0)
Type "help" for help.

postgres=# select count(*) from t1;
 count
-------
  2000
(1 row)

postgres=# \watch
Sun 10 Dec 2023 01:57:55 PM CET (every 2s)

 count
-------
  2000
(1 row)
…
```

pg_stat_io
Hits

In another session, monitor the "hits"

```
postgres=# select backend_type, object, context, hits from pg_stat_io where hits > 0;
   backend_type    |  object  | context | hits
-------------------+----------+---------+------
 autovacuum worker | relation | normal  |  537
 client backend    | relation | normal  | 1090
(2 rows)

postgres=# \watch
   backend_type    |  object  | context | hits
-------------------+----------+---------+------
 autovacuum worker | relation | normal  |  627
 client backend    | relation | normal  | 1306
…
```

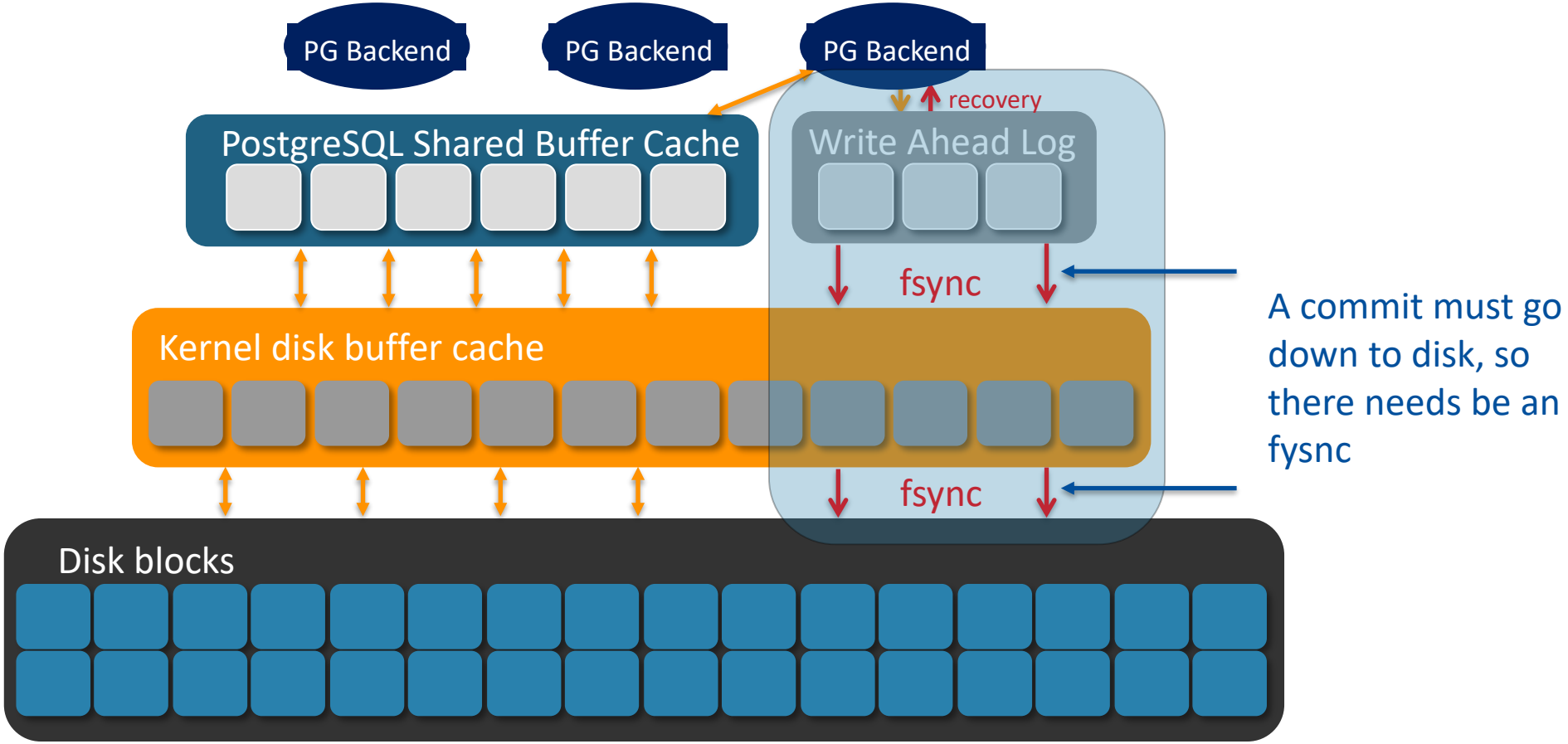
Finally: What is this metric good for?

- Without "evictions" there is probably not much you can make out of this
- A high number of "hits" compared to a low number of "evictions"
  - Most probably means your buffer cache is either fine or too large
- A low number of "hits" compared to a high number of "evictions"
  - Most probably means your buffer cache is sized too small

(4) - fsyncs

pg_stat_io
fsyncs

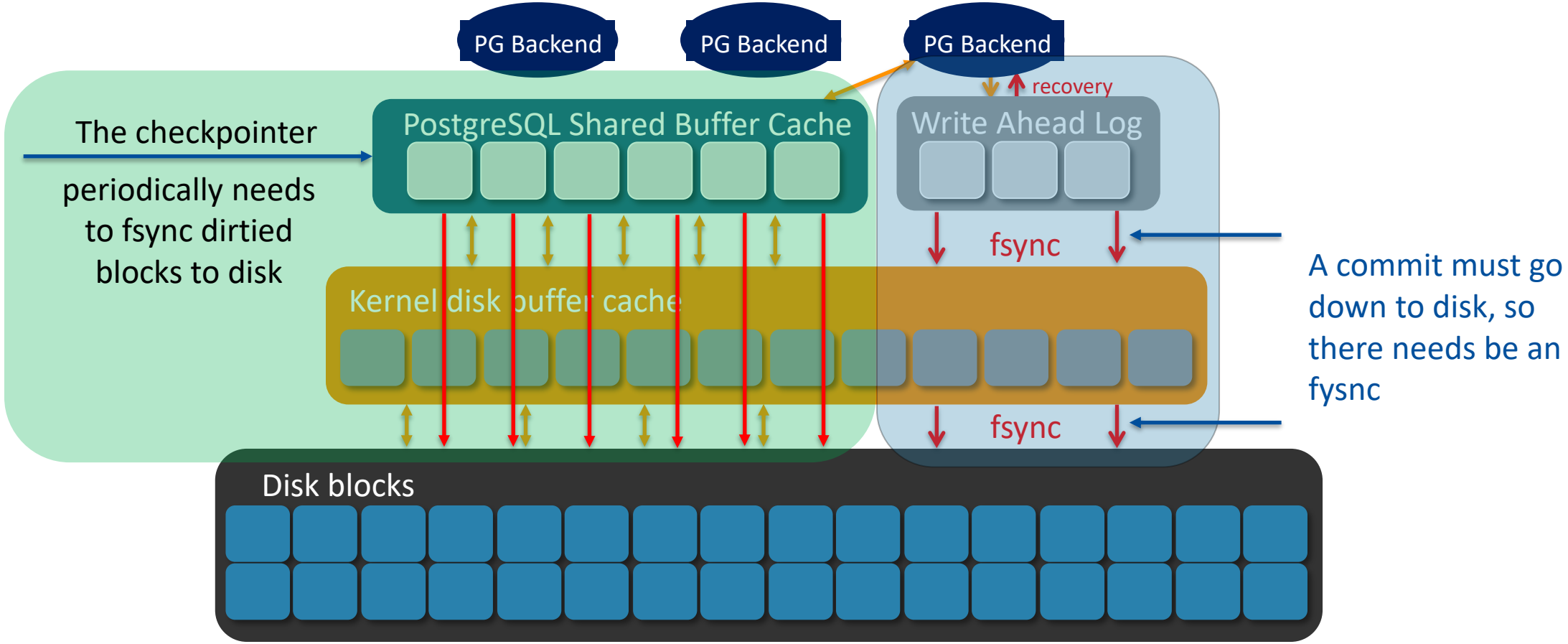
What does "fsyncs" mean?



A commit must go down to disk, so there needs be an fysnc

pg_stat_io
fsyncs

What does "fsyncs" mean?



pg_stat_io
fsyncs

fsync calls are only tracked in context normal

```
postgres=# psql -c "select context
                          , fsyncs
                          , fsync_time
                       from pg_stat_io
                      where fsyncs > 0"
 context | fsyncs | fsync_time
---------+--------+------------
 normal  |      2 |      5.771
(1 row)
```

Finally: What is this metric good for?

- It is the task of the checkpointer and the backround writer to write dirtied buffers to disk
  - Client backends should be able to rely on those
    - But can issue fsyncs as well, if required
- If you see many fsyncs by client backends
  - Either shared_buffers is misconfigured
  - or
  - The checkpointer is not configured correctly

(5) - Remaining metrics

pg_stat_io
Remaining metrics

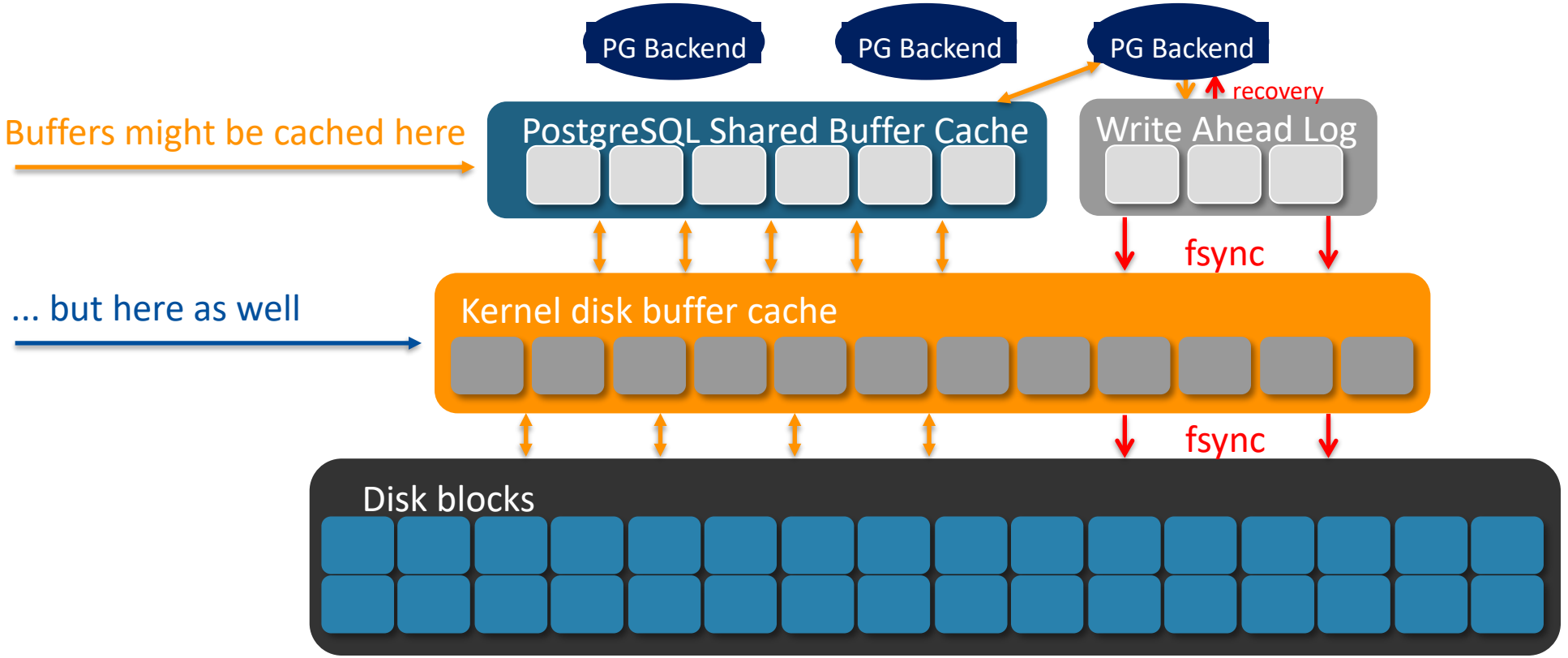
Other metrics we do not have time to talk about

Metric	Meaning
reads	Number of read operations, each of the size specified in op_bytes
read_time	Time spent in read operations in milliseconds
writes	Number of write operations, each of the size specified in op_bytes
write_time	Time spent in write operations in milliseconds
writebacks	Number of units of size op_bytes which the process requested the kernel write out to permanent storage
writeback_time	Time spent in writeback operations in milliseconds
reuses	The number of times an existing buffer in a size-limited ring buffer outside of shared buffers was reused as part of an I/O operation in the bulkread, bulkwrite, or vacuum contexts

(6) - direct I/O

Again, the same picture

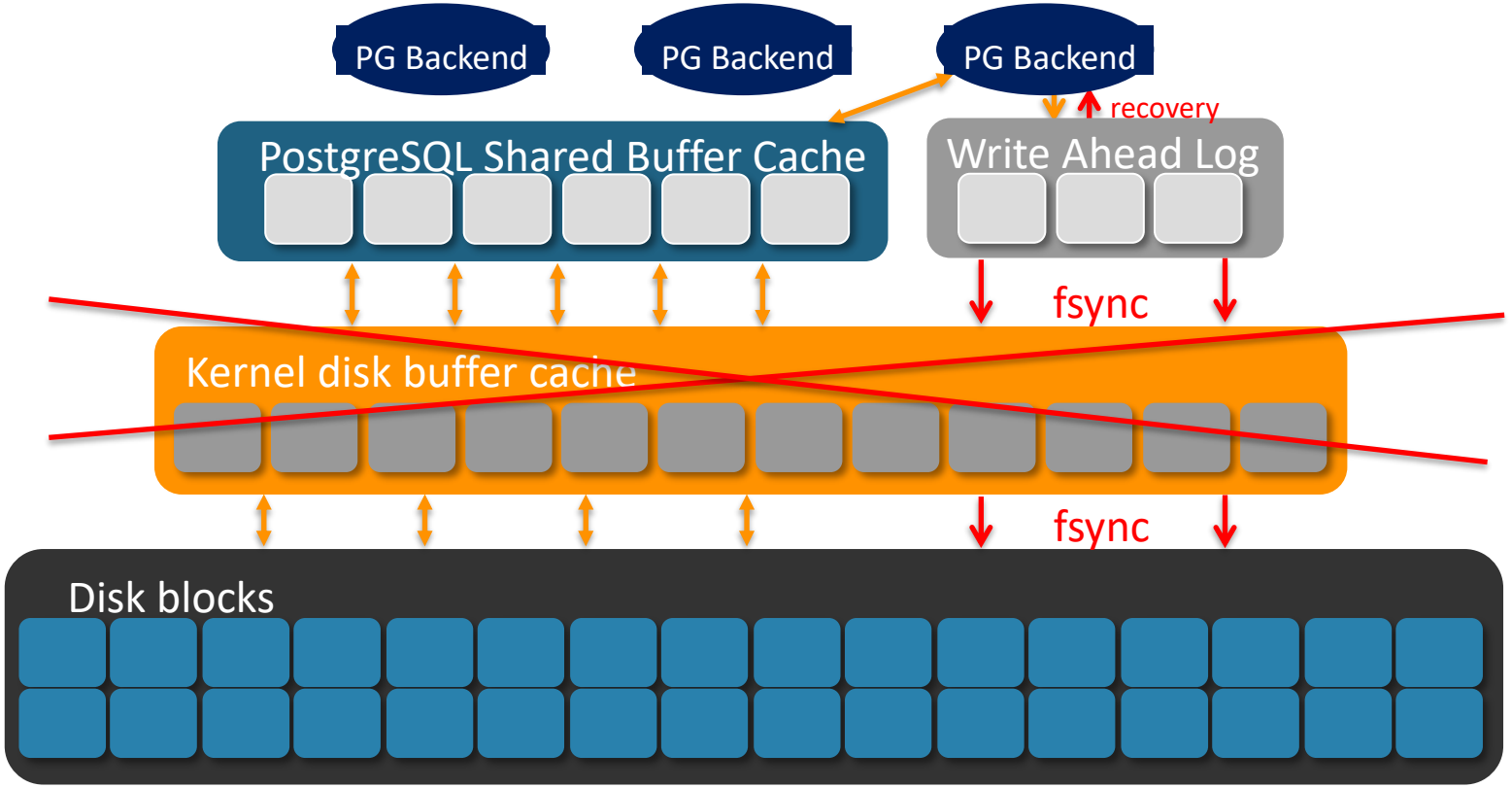
> This architecture comes with (potential) double buffering



Again, the same picture

> Direct I/O bypasses the OS file cache

> Files must be opened with the O_DIRECT flag (on Unix/Linux systems)

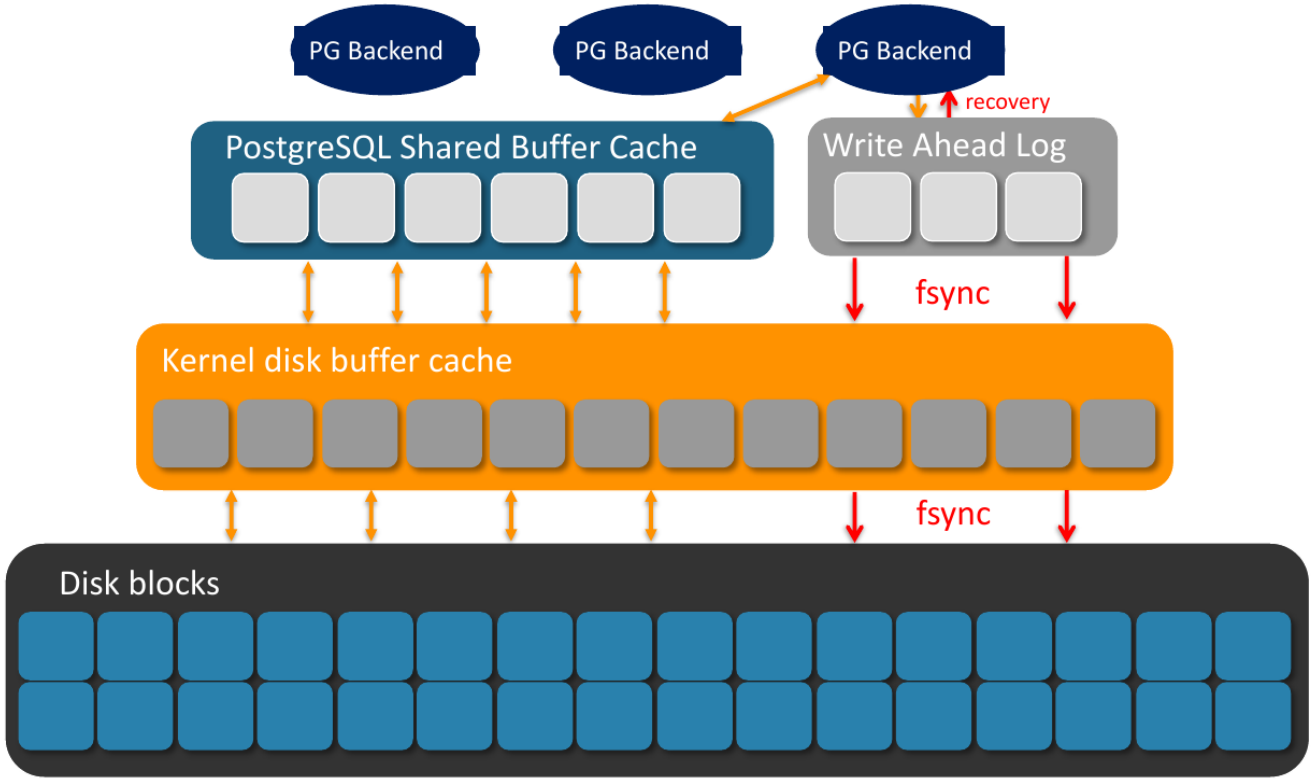


pg_stat_io
Direct I/O

PostgreSQL 16 comes with a new developer option: debug_io_direct

- Can be set to either
  - "data" for main data files
  - "wal" for WAL files
  - "wal_init" for initializing WAL files

- Asks the Kernel to minize caching effects
  - O_DIRECT (most Unix systems)
  - F_NOCACHE (maxOS)
  - FILE_FLAG_NO_BUFFERING (Windows)



pg_stat_io
Direct I/O, help to test

Testing a workload without direct I/O

```
$ psql -c "show debug_io_direct;
 debug_io_direct
-----------------

(1 row)

$ time pgbench -i -s 10
dropping old tables...
…
real    0m4.747s
user    0m0.208s
sys     0m0.012s
$ pgbench --client=2 --time=10 --progress=1
number of transactions actually processed: 2342
number of failed transactions: 0 (0.000%)
latency average = 8.535 ms
latency stddev = 10.716 ms
initial connection time = 8.949 ms
tps = 234.208970 (without initial connection time)
```

pg_stat_io
Direct I/O, help to test

Enabling direct I/O

```
$ psql -c "alter system set debug_io_direct
                          to 'data','wal','wal_init';"
ALTER SYSTEM
$ pg_ctl stop
$ pg_ctl start
$ psql -c "show debug_io_direct"
   debug_io_direct
---------------------
 data, wal, wal_init
(1 row)
```

pg_stat_io
Direct I/O, help to test

Repeating the same workload

```
postgres@debian12:[160] time pgbench -i -s 10
dropping old tables...
NOTICE:  table "pgbench_accounts" does not exist, skipping
…
real    0m5.147s
user    0m0.189s
sys     0m0.010s
postgres@debian12:[160] pgbench --client=2 --time=10 --progress=1
number of transactions actually processed: 2342
number of failed transactions: 0 (0.000%)
latency average = 8.535 ms
latency stddev = 10.716 ms
initial connection time = 8.949 ms
tps = 260.860923 (without initial connection time)
```


Done
... almost

A big "Thank you, to … "

commit a9c70b46dbe152e094f137f7e6ba9cd3a638ee25
Author: Andres Freund <andres@anarazel.de>
Date: Sat Feb 11 09:51:58 2023 -0800

  Add pg_stat_io view, providing more detailed IO statistics

  Builds on 28e626bde00 and f30d62c2fc6. See the former for motivation.
…
Bumps catversion.

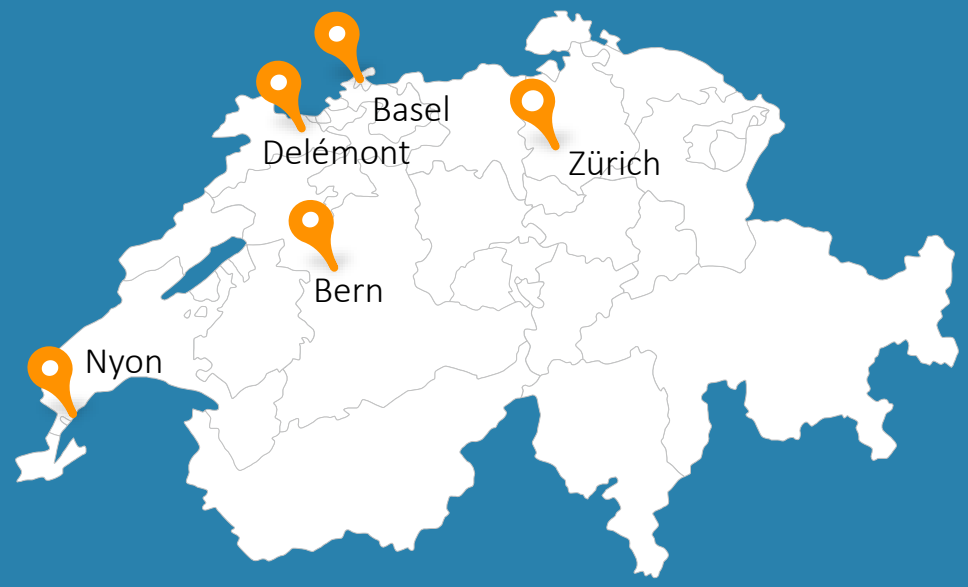
  Author: **Melanie Plageman** <melanieplageman@gmail.com>
  Author: **Samay Sharma** <smilingsamay@gmail.com>
  Reviewed-by: **Maciek Sakrejda** <m.sakrejda@gmail.com>
  Reviewed-by: **Lukas Fittl** <lukas@fittl.com>
  Reviewed-by: **Andres Freund** <andres@anarazel.de>
  Reviewed-by: **Justin Pryzby** <pryzby@telsasoft.com>
  Discussion: https://postgr.es/m/20200124195226.lth52iydq2n2uilq@alap3.anarazel.de

Want to travel to Munich in April 2024?

https://2024.pgconf.de

Any questions?

Please do ask!



We would love to boost your IT-Infrastructure

How about you?