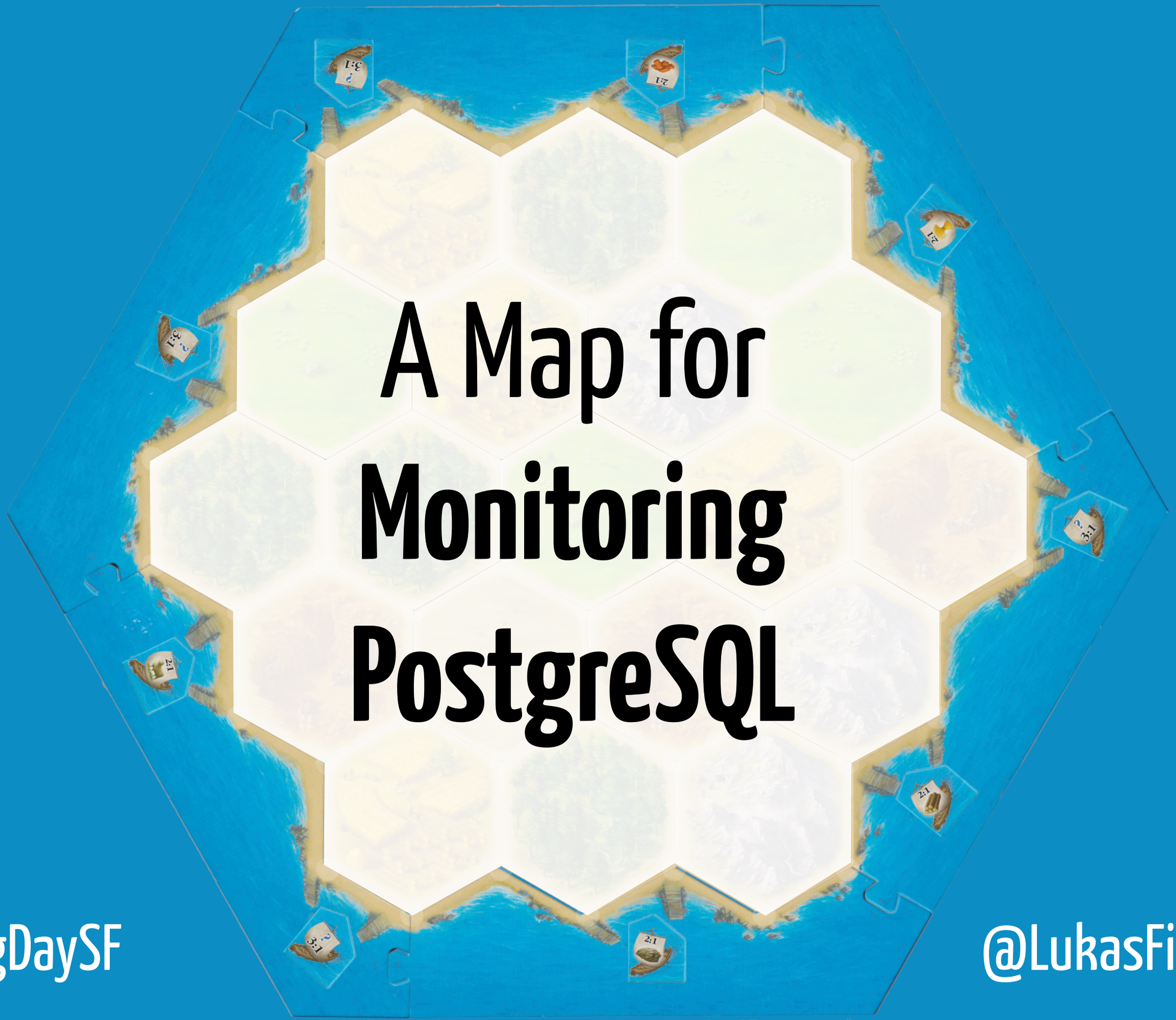


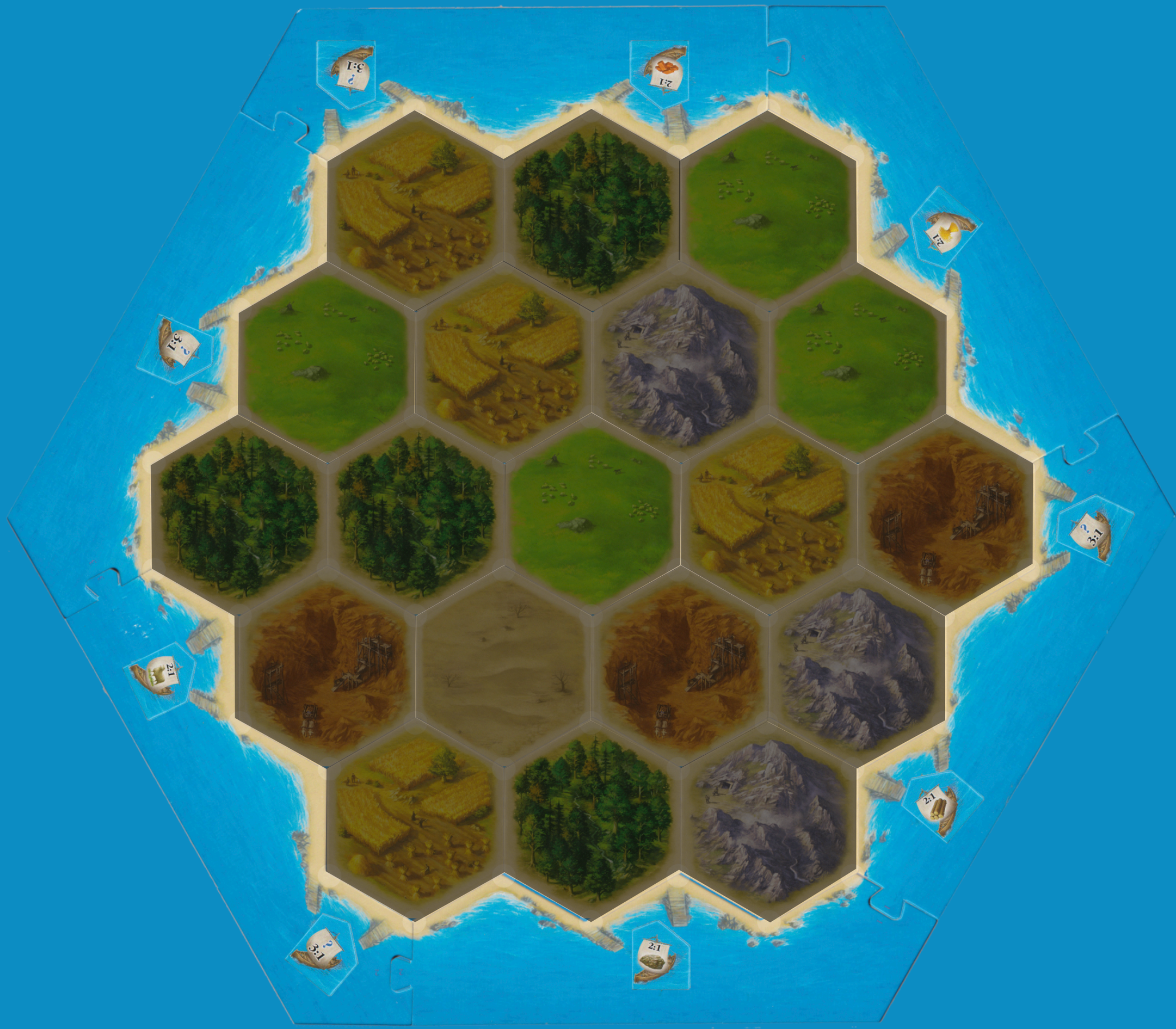
A Map for Monitoring PostgreSQL

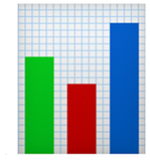
#PgDaySF

@LukasFittl

@LukasFittl

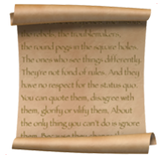
> 100 Metrics We Could Talk About


3:1
2:1
3:1
2:1
2:1
3:1
2:1
2:1



Historic Metrics

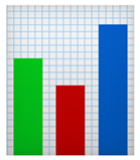
Current Activity



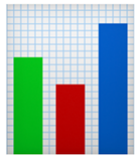
Logs

Tuning Actions


Query
Workload



pg_stat_statements



Enabling pg_stat_statements

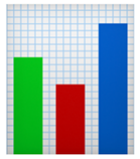
1. Install postgresql contrib package (if not installed)
2. Enable in postgresql.conf

```
shared_preload_libraries = 'pg_stat_statements'
```

3. Restart your database
4. Create the extension

```
CREATE EXTENSION pg_stat_statements;
```

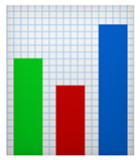
Enabled By Default On Most Cloud Platforms



pg_stat_statements

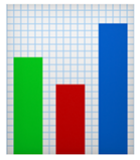
```
SELECT * FROM pg_stat_statements;

userid             | 10
dbid               | 1397527
query              | SELECT * FROM x WHERE
calls              | 5
total_time         | 15.249
rows               | 0
shared_blks_hit    | 451
shared_blks_read   | 41
shared_blks_dirtied | 26
shared_blks_written | 0
local_blks_hit     | 0
```



```
queryid      | 1720234670
query        | SELECT * FROM x WHERE y = ?
calls        | 567
total_time   | 56063.6489
```

Avg Runtime = 98.87 ms

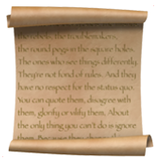


```
queryid       | 1720234670
query         | SELECT * FROM x WHERE y = ?
calls         | 567
total_time    | 56063.6489
min_time      | 0.0949
max_time      | 902.545
mean_time     | 98.87687654321
stddev_time   | 203.19222186271
```

Mean Runtime = **98.87 ms**

95th Percentile = **505.45 ms**

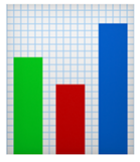
Max Runtime = **902.54 ms**



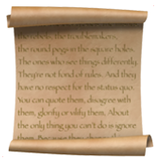
Slow Queries

```
log_min_duration_statement
        = 1000 ms
```

```
LOG: duration: 4079.697 ms execute <unnamed>:
SELECT * FROM x WHERE y = $1 LIMIT $2
DETAIL: parameters: $1 = 'long string', $2 = '1'
```



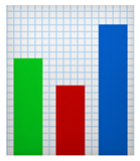
```
UPDATE "backends"
SET seen_at_range = tstzrange(LOWER(seen_at_range), ?::timestamptz)
WHERE "backends"."server_id" = ?
AND ("backends"."backend_id" NOT IN (?) )
AND (seen_at_range @> ?::timestamptz)
```



Query Sample

Mar 21, 2018 10:16 PM PDT

```
UPDATE "backends"
SET seen_at_range = tstzrange(lower(seen_at_range), '2018-03-22 05:16:20 UTC'::timestamptz)
WHERE "backends"."server_id" = 'fe86cc41-ff76-46c6-851d-7f585bc1c346'
AND ("backends"."backend_id" NOT IN ('8630f3d1-9037-413e-87ea-66b2aad3cb88', '4bc00bb0-fd34-4c8a-a511-afda2ed4bd84', '2c247b9b-85ce-4eb4-b995-bf
840c7387c9', 'a080899f-f59f-4059-8ca8-b1590c76e3bb', '611ca351-a691-4884-8ba6-02f28ab325c4', 'ed280708-56eb-4741-819a-4f5dcd88b221', 'd5bf8fc3-e
530-4e71-a6b2-74cbfb5ed7ad', '319e8988-8fd4-4794-80b0-9b9dc4068b3b', 'c8956d94-13c4-4759-99b5-1984ce23c9ef', 'e339dfcb-960f-451f-8cb5-5810221772
b7', 'd96989a7-8f28-4264-adbc-80430c14501b', '0c3de58f-4dd0-4581-a6e9-29861d06b0c2', '894937d1-a49d-41b1-bf76-390ed606645b', '3c624496-84c5-411c
-a3bf-6a841ac90373', '65c276de-42bd-438d-93ea-06ae4627bcb0', 'bc3d9652-1e65-4ae1-8d5b-c0526c0dbfb8', '95c2d4b0-fdf7-456c-914a-833ef6e448af', '1c
352037-1d62-4e8f-b158-4b89e7af5834', 'fec59cd4-c7b6-4810-904e-ae8d11b876c8', 'c946fb65-222b-42fc-ac40-51053bc5946b', '4acd24cd-82cd-4dc4-a063-28
7e537459ea', 'f933626e-a337-453d-9f7a-03494a126c04', 'd198f737-2a6e-4a93-b91a-f865ef9893fd', 'd62a950a-6493-488f-bb70-1311c2f68d39', '7a7c22cd-7
59e-4e43-a7d3-554dd47d97c5', 'f6c832cb-cfb8-4940-98f2-a483b6422cb0', '0ce20fce-aa10-45a0-9fad-b8db820d8e8b', 'adab4b7c-335a-4ded-aa08-a43a5a9852
b7', 'd544fe1d-7938-4acb-a139-2a4b3eb3adf7', '65f1f7cc-781f-4dad-9f55-f938b3ed8744', '6d013a17-efbe-421b-bdad-f46bd9698726', 'a3017438-0bea-47a2
-bacc-d115700720d0', '4cd0bb9c-3c70-4cd1-9440-192e225b28bf', '6a476c68-cfef-49fa-9d6f-dbfaf7db6bf5', '56153417-c52e-421b-8a7c-18890a2575a3', '1a
70c019-820c-4ade-8a59-be8c3ca1c9de', '356321b4-9455-4d9a-abc0-81d94cb0a201', '49616461-5995-4930-9bbe-391f9b2a5c9e', 'ab0b3fa1-d86f-44c7-8046-d6
a185fc872b', '47347373-7211-4312-b68d-dfa1e9648f40', 'dce4eed4-d452-459b-93ba-e5778c3e6678', 'cdd2111d-65ca-4d20-84fd-3bde7842a9db', '23f692b4-7
dc0-4937-8083-020425fc5afa', 'a65b9e2b-bc4f-4a44-b7eb-1f33393583a4', '67b2a0d3-1b05-4f4f-ba35-4e07ba9b7466', 'e45e0d1a-2477-426f-a576-a1a427d48f
ef', 'cc759f3e-ab65-42d7-8e90-d3c17215047d', '1f541b9a-7943-4afc-b4d9-18cb29a0cd0c', '5eb31d51-c338-489f-80b6-5fcfa34ea0ca', '1939b071-11a6-4c1f
-8d1b-f4b54bd9f302', '26648c41-fe0a-4e47-8d0f-0f1768d177a2', 'd0e49353-b441-423b-af3e-e9c9f98dfc93', '4ff4f0f6-c5cd-43ac-a26a-9c2108e8e14d', '38
44c6fb-f10d-417a-8df4-ccc4c8bde06d', 'eac88dfc-9ca5-4f74-b3f7-43e8354d4bd2', 'e62e7d61-e974-4280-a10d-cb88c5a627d5', '012f215a-81b5-40b7-b559-d7
f98e0c9bb7', '19217067-0734-4925-beb4-063971554c7a', '2f86e9a1-dcbd-4acd-97c6-c83da1e01cad', 'c5fd77f1-f237-41ab-8bce-4321cabd17e1', 'a7e4a033-2
0f9-46dc-a949-9ecc3900550f', '072972f7-2755-4148-b84d-1733a534b312', '0b49e424-8236-4c96-9205-ed22a8bc7a9f', 'ef57f5de-aadb-448c-8a61-92cf988102
b9', 'd7719a38-f46d-448b-815b-69998a2ec4df', '541085d2-8305-430c-90cb-f375c8e3b33f', 'a38386a6-dc56-4e38-ade5-34f07920ac63', '40b23eb2-9d31-4168
-8f35-4306aec06137', 'e3acb49a-b9da-4850-bf9b-5336342682a5', 'f22a0772-c744-439a-89f8-76ef68be8797', '30278ece-6605-41ab-94e5-a414a7c8b3ef', '54
a1ad82-a4af-4ae4-841a-a95fb2dd1bbe', '1a0fa601-dc1e-4a52-b01a-a129878265c0', '277b747a-c324-4dde-82e6-391c003bd4c5', '5503b7ed-d7f5-41fa-9d13-f7
826d348e9a', 'eccd0d94-00f5-40a9-b50d-35e9e636e6fd', 'cb66f8ee-e17a-4586-a438-00d770ed79dd', '75567845-946c-4be2-8bb2-a4c03243df09', '59208377-5
9a6-4afd-8e71-b6ad832f00c7', 'd9c8b212-69fa-4266-abea-872a7aa892c4', 'e9d1ecd2-6753-416c-b4c2-6afbba14b0b0', 'b77e6556-3158-45af-bfb5-a1467b1d50
34', '8e7f0a6c-2500-4b43-b07b-5eedf4347d45', '0cb1e7e4-669c-4cc7-bc87-86ffd3d54651', '9cff3bbe-cbdc-4d5e-aa6c-86bb44ea2b40', 'f996cc84-eb0a-497b
-b440-d4df0448fe46', '800cadb7-7a98-4d2a-9516-78663e941dad', '97fc580a-de21-4b66-afb2-9a766e5a31ae', 'edcadeca-e84b-4a68-b0b9-5584ca93375e', '73
20ae94-2333-437d-9b44-1ba2d8381d94', 'a2ff4c69-5a6e-4ca2-a273-add4891f30f6', 'a4837fcb-ed47-41fd-bc9c-79df07a63ae1', '8dc23b9d-8959-4ac9-a789-6a
b11812c6ca', 'e91fb2e4-9a56-4dae-b2c3-65e569c01b97', 'c9334914-5af5-40bf-afda-3021216564d3', 'fab1a1d8-0af6-403f-b771-bd23c71c87f0', 'a25441c9-1
9be-4982-82ac-0628e6da02d6', '31d0d8f8-751a-46fd-8902-d9569b134fc2', '293f7dd3-bf20-4dbe-a065-b4e89dc6b03c', '64f19dc9-32f1-42eb-9d24-b49251af42
fb', 'f52e20dd-4dd9-4b72-b1bb-bbaba9b1fd9b', 'fc66e2ed-46e6-48c5-b47b-6fe33abe87b3', '4af4efe4-3636-47ae-ae28-fc9b02073ae8', '7c5d74fb-b248-4da0
-a8ae-a388442266d6', '63f9112b-5a7e-4d12-8079-1a19d4b87d3c', '22f52c2d-8717-4473-9392-453c63b0c348', '7c31ee83-5195-40a4-9e8c-a46a89e4b54b', '30
adf624-6d1c-4ab1-9f9b-c1b968b90974', '2cc696b7-eb09-49ec-a109-f20028f0e49f', '96b929bb-4583-4429-b96f-5d28a2bf1681', 'f73ee3a8-6697-4b6b-8283-05
8b0b680683', '490e6950-2f97-419d-b9f7-996c1b93fdd4', '35770933-e13f-4794-ad83-a5822d2bd886', '38f6013a-c35c-4b26-a199-597b23cbf450', 'bd4d48d1-4
f09-4bec-9960-ae46991fac18', '9179a431-779f-4992-a080-3c042fe5235c', '6c0779a2-eae9-42df-a9f1-658ab06846bd', 'fc216aa1-bee0-491f-992d-eadb821055
22', 'f961a354-c39b-4b13-813b-41fe956d5546', '08940c58-3a2c-4b80-b339-472fa7a4eea0', 'bf49a7f4-a458-4582-abb7-eaad53e0bbde', '231f8e7d-5aea-4ce8
-b1be-8852fc286a07', '180e1719-2ae8-432c-a077-83024286147c', 'ac858ddf-16b9-4644-9494-ae9c4d4320d7', '1c69b200-b5cd-4ba0-9f64-dde4b5f9832f', '7a
```



pg_stat_database

xact_commit:

Committed Transactions Per Second

tup_*:

Rows Updated/etc Per Second

Optimize Indices,

Tune Postgres or

Rewrite/Change Your Queries


Index
Optimization

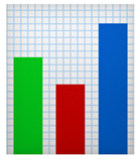
Important Questions For Indices

Should I add an index?

Do I need to REINDEX?

Should I remove an index?

Should I add an index?

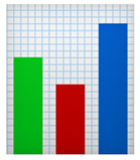


Should I add an index?

Measuring Sequential Scans - Per Table

```
pg_stat_all_tables

seq_scan:          # of Sequential Scans
seq_tup_read:      # of rows read by
                   # Sequential Scans
```



Index Hit Rate

```
SELECT relname, seq_scan + idx_scan,
       100 * idx_scan / (seq_scan + idx_scan)
  FROM pg_stat_user_tables
ORDER BY n_live_tup DESC
```

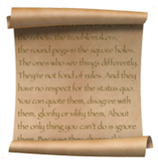
Target: >= 95% on large, active tables

Should I add an index?

For a Specific Query?
Can I use pg_stat_statements?

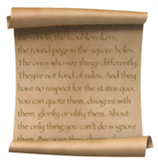
Doesn't know about what indices get used / what plan is being executed.

Doesn't have enough details to EXPLAIN a query, because text is normalized.

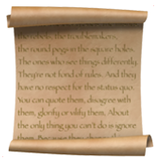


auto_explain

logs the query plan
for specific slow queries



```
2018-03-11 01:00:03 UTC:10.40.29.136(48110):demo_pgbench@demo_pgbench:[31321]:LOG:  duration: 2334.085 ms  plan:
        {
          "Query Text": "SELECT abalance FROM pgbench_accounts WHERE aid = 2262632;",
          "Plan": {
            "Node Type": "Index Scan",
            "Parallel Aware": false,
            "Scan Direction": "Forward",
            "Index Name": "pgbench_accounts_pkey",
            "Relation Name": "pgbench_accounts",
            "Schema": "public",
            "Alias": "pgbench_accounts",
            "Startup Cost": 0.43,
            "Total Cost": 8.45,
            "Plan Rows": 1,
            "Plan Width": 4,
            "Actual Rows": 1,
            "Actual Loops": 1,
            "Output": ["abalance"],
            "Index Cond": "(pgbench_accounts.aid = 2262632)",
            "Rows Removed by Index Recheck": 0,
            "Shared Hit Blocks": 4,
            "Shared Read Blocks": 0,
            "Shared Dirtied Blocks": 0,
            "Shared Written Blocks": 0,
            "Local Hit Blocks": 0,
            "Local Read Blocks": 0,
            "Local Dirtied Blocks": 0,
            "Local Written Blocks": 0,
            "Temp Read Blocks": 0,
            "Temp Written Blocks": 0,
            "I/O Read Time": 0.000,
            "I/O Write Time": 0.000
          },
          "Triggers": [
          ]
        }
```



EXPLAIN Plan

2020-01-21 09:39:12am UTC · a3e4875

1344.49ms	1344.42ms	8 kB	8
Runtime	I/O Read Time	Read From Disk	Total Est. Cost

✔ Analyze ✖ Verbose ✔ Costs ✔ Buffers ✖ Timing ✖ Summary

Compare Query Plans View EXPLAIN Source

EXPLAIN Insights

Plan node 2 was estimated to be expensive (cost 8 vs avg cost 4)

```
UPDATE pgbench_accounts SET abalance = abalance + -3370 WHERE aid = 3541771;
```

ModifyTable (Update) 1

on public.pgbench_accounts

Est. Cost:	8
Est. Rows:	1
Actual Rows:	0

Index Scan (Forward) 2

on public.pgbench_accounts
using pgbench_accounts_pkey
expensive

Est. Cost:	8
Est. Rows:	1
Actual Rows:	1

Index Scan (Forward) 2

on public.pgbench_accounts
using pgbench_accounts_pkey

Overview | I/O & Buffers | Output | Source

EXPLAIN Insights

expensive was estimated to be expensive (cost 8 vs avg cost 4)

Index Scan

Scans through the index to fetch a single value or a range of values in index order from the table. Learn more

Index Cond

```
(pgbench_accounts.aid = 3541771)
```

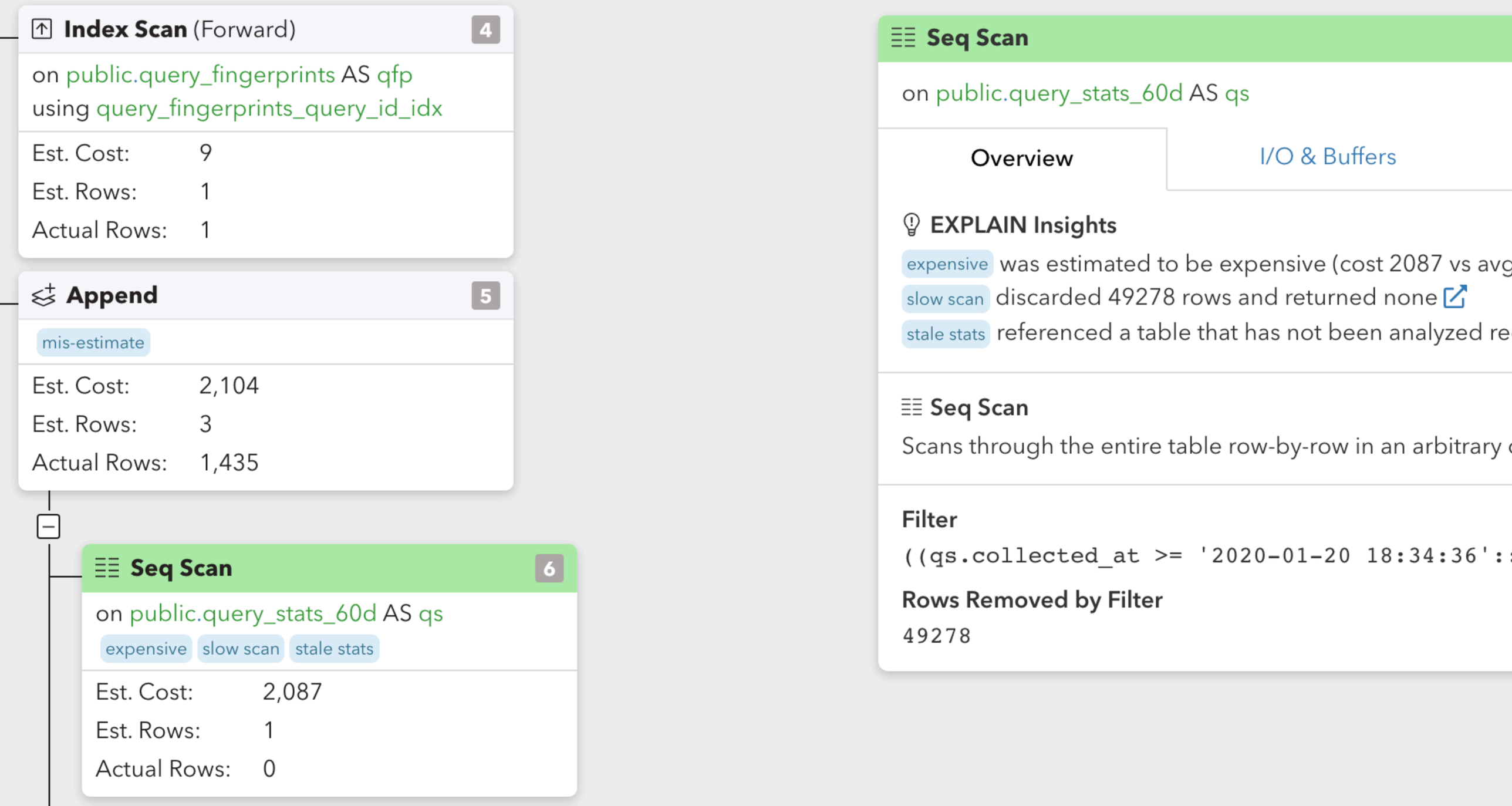
Rows Removed by Index Recheck

0

Scan Direction

Forward

“Discarded 49278 rows and **returned none.**"



Create Indices When There Are Frequent Sequential Scans on Large Tables

Measure CREATE INDEX Progress

`pg_stat_progress_create_index`

```
# SELECT index_relid::regclass, phase, blocks_done, blocks_total
    FROM pg_stat_progress_create_index;
   index_relid    |              phase               | blocks_done | blocks_total
------------------+----------------------------------+-------------+--------------
  index_tab_pkey  | building index: scanning table |       27719 |        44248
(1 row)
```

Postgres 12+

Do I need to REINDEX?

Do I need to REINDEX?

```
# SELECT relname, pg_table_size(oid) as index_size,
  100-pgstatindex(relname).avg_leaf_density AS leaf_density
FROM pg_class;
```

relname	index_size	leaf_density
test_inventory_id_idx	376832	**89.75**
test_pkey	376832	89.75
test_rental_date_inventory_id_customer_id_idx	524288	89.27

pgstatindex(relname).avg_leaf_density

Density of ~90% = Optimal for B-Tree

Do I need to REINDEX?

```
# SELECT relname, pg_table_size(oid) as index_size,
  100-pgstatindex(relname).avg_leaf_density AS leaf_density
FROM pg_class;
```

relname	index_size	leaf_density
test_inventory_id_idx	376832	**89.75**
test_pkey	376832	89.75
test_rental_date_inventory_id_customer_id_idx	524288	89.27

UPDATE 50% of Rows in Table:

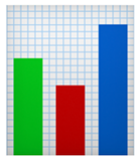
relname	index_size	leaf_density
test_inventory_id_idx	745472	**45.52**
test_pkey	737280	46.02
test_rental_date_inventory_id_customer_id_idx	925696	51.04

Index Size Doubled, 50% Bloated

When Indices Have Low Density

REINDEX CONCURRENTLY

for better performance

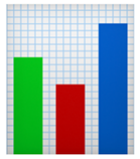


Should I remove an index?

Measuring Index Scans - Per Index

pg_stat_all_indices

```
idx_scan:          # of Index Scans
```



Should I remove an index?

relname	n_live_tup	scans	index_hit_rate
query_fingerprints	347746140	513262821	99
queries	346575911	22379253	99
schema_table_events	100746488	1459	99
queries_schema_tables	62194571	7754	99
log_lines	46629937	2	0
issue_states	31861134	3	0
schema_columns	31849719	6688381553	99
query_overview_stats	26029247	13831	99
schema_index_stats_2d_20170329	18274023	1592	99
schema_index_stats_2d_20170328	18164132	6917	99
snapshot_benchmarks	13094945	2315069	99
schema_index_stats_60d_20170329	9818030	69	20
schema_index_stats_60d_20170328	9749146	110	30
schema_index_stats_60d_20170323	9709723	103	40
schema_index_stats_60d_20170327	9702565	103	33
schema_index_stats_60d_20170324	9672853	64	48
schema_index_stats_60d_20170322	9651125	141	46
schema_index_stats_60d_20170325	9647832	23	69
schema_index_stats_60d_20170326	9636532	39	53
schema_index_stats_60d_20170303	9538898	174	63
schema_index_stats_60d_20170321	9522712	170	49
schema_index_stats_60d_20170309	9492844	126	57
schema_index_stats_60d_20170304	9491850	64	82
schema_index_stats_60d_20170320	9486945	104	56
schema_index_stats_60d_20170319	9466378	47	74

Remove Indices When There Are **No Index Scans**

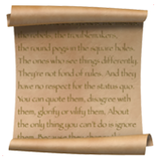
(But watch out for **Replicas**)

Unused Indices:

- Make Writes Slower
- Cause VACUUM to take longer

Index Scans

Read From The Table Too!



Bitmap Heap Scan 3

on public.queries

expensive

Est. Cost:	25,774,595
Est. Rows:	13,943,446
Actual Rows:	2,178,416

Bitmap Index Scan 4

using index_queries_on_last_occurred_at

Est. Cost:	973,280
Est. Rows:	13,943,446
Actual Rows:	10,607,067

Exact Heap Blocks

1548352

Lossy Heap Blocks

0

Recheck Cond

(queries.last_oc

Rows Removed by In

0

Index Scan (Forward) 14

on public.query_table_associations

using index_query_table_associations_on…

Est. Cost:	221
Est. Rows:	54
Actual Rows:	0

Looped 2178416 times

Index Only Scan (Forward) 10

on group_user AS gu

using uk_groupid_userid

Est. Cost:	115
Est. Rows:	2
Actual Rows:	1

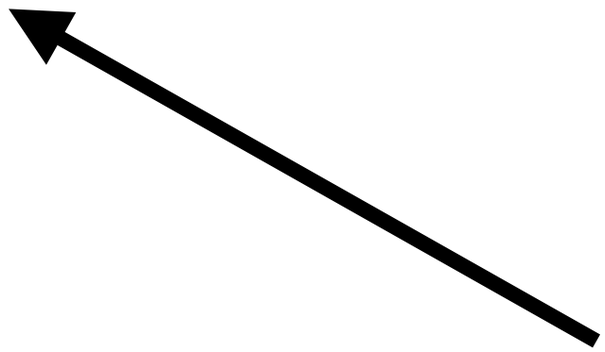
Looped 785 times

pg_stat_all_tables
- idx_tup_fetch

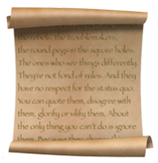
Bitmap Heap Scan

pg_stat_all_indices
- idx_tup_fetch

Index Scan



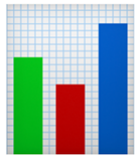
Index-Only Scan



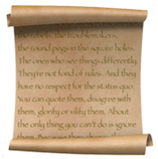
```
 QUERY PLAN
 ──────

  Aggregate  (cost=12.53..12.54 rows=1 width=0) (actual
time=0.046..0.046 rows=1 loops=1)
    ->  Index Only Scan using categories_pkey on
categories  (cost=0.00..12.49 rows=16 width=0) (actual
time=0.018..0.038 rows=16 loops=1)
          Heap Fetches: 16
  Total runtime: 0.108 ms
 (4 rows)
```

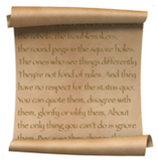

Query Tags



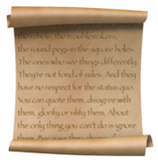
```
SELECT SUM("log_files"."byte_size")
  FROM "log_files"
 WHERE ("log_files"."collected_at" BETWEEN $1 AND $2)
       AND "log_files"."server_id" IN (
         SELECT "servers"."id"
           FROM "servers"
          WHERE "servers"."organization_id" = $3
          AND "servers"."deleted_at" IS NULL
       )
```



```
SELECT SUM("log_files"."byte_size")
  FROM "log_files"
 WHERE ("log_files"."collected_at" BETWEEN $1 AND $2)
       AND "log_files"."server_id" IN (
         SELECT "servers"."id"
           FROM "servers"
          WHERE "servers"."organization_id" = $3
                AND "servers"."deleted_at" IS NULL
         )
/*application:pganalyze,controller:graphql,action:graphql,line:/app/graphql/organization_t
 in <top (required)>',graphql:getOrganizationDetails.logVolume24h,request_id:44bd562e-0f53
```



```
application: pganalyze
 controller: graphql
     action: graphql
       line: /app/graphql/organization_type.rb …
    graphql: getOrganizationDetails.logVolume24h
 request_id: 44bd562e-0f53-453f-831f-498e61ab6db5
```



github.com/basecamp/**marginalia**

Automatic
Query Tags For Ruby on Rails

When A Web Request Is Slow,
Find The Slow Queries
By Tagging Them In Your App


Connection
Pooling

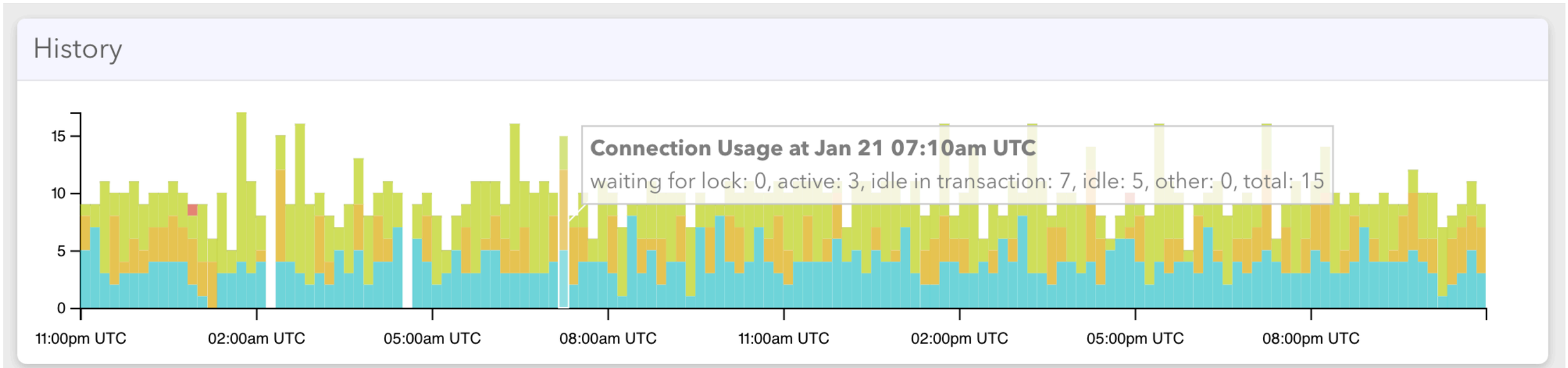
pg_stat_activity

pid: process ID
backend_type: “client backend” vs internal processes
state: idle/active/idle in transaction
state_change: time of state change
query: current/last running query
backend_start: process start time
xact_start: TX start time
query_start: query start time
wait_event: what backend is waiting for (e.g. Lock, I/O, etc)

…

of Connections By State

```
SELECT state,
       backend_type,
       COUNT(*)
  FROM pg_stat_activity
 GROUP BY 1, 2
```



High Number of Idle Connections

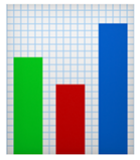
=> Add a connection pooler

work_mem Tuning

Out Of Memory

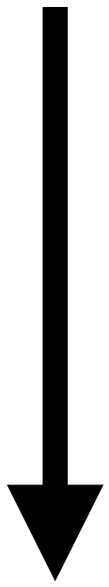
VS

Operations Spill To Disk

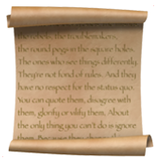


Temporary Files Written

pg_stat_database.temp_bytes



pg_stat_statements.temp_blks_written



Temporary Files Written (Per Query)

log_temp_files = 0

```
Jan 20 09:18:58pm PST 28847 LOG: temporary file: path
"base/pgsql_tmp/pgsql_tmp28847.9", size 50658332

Jan 20 09:18:58pm PST 28847 STATEMENT:  WITH servers AS
( SELECT …
```

When Sorts Spill To Disk,

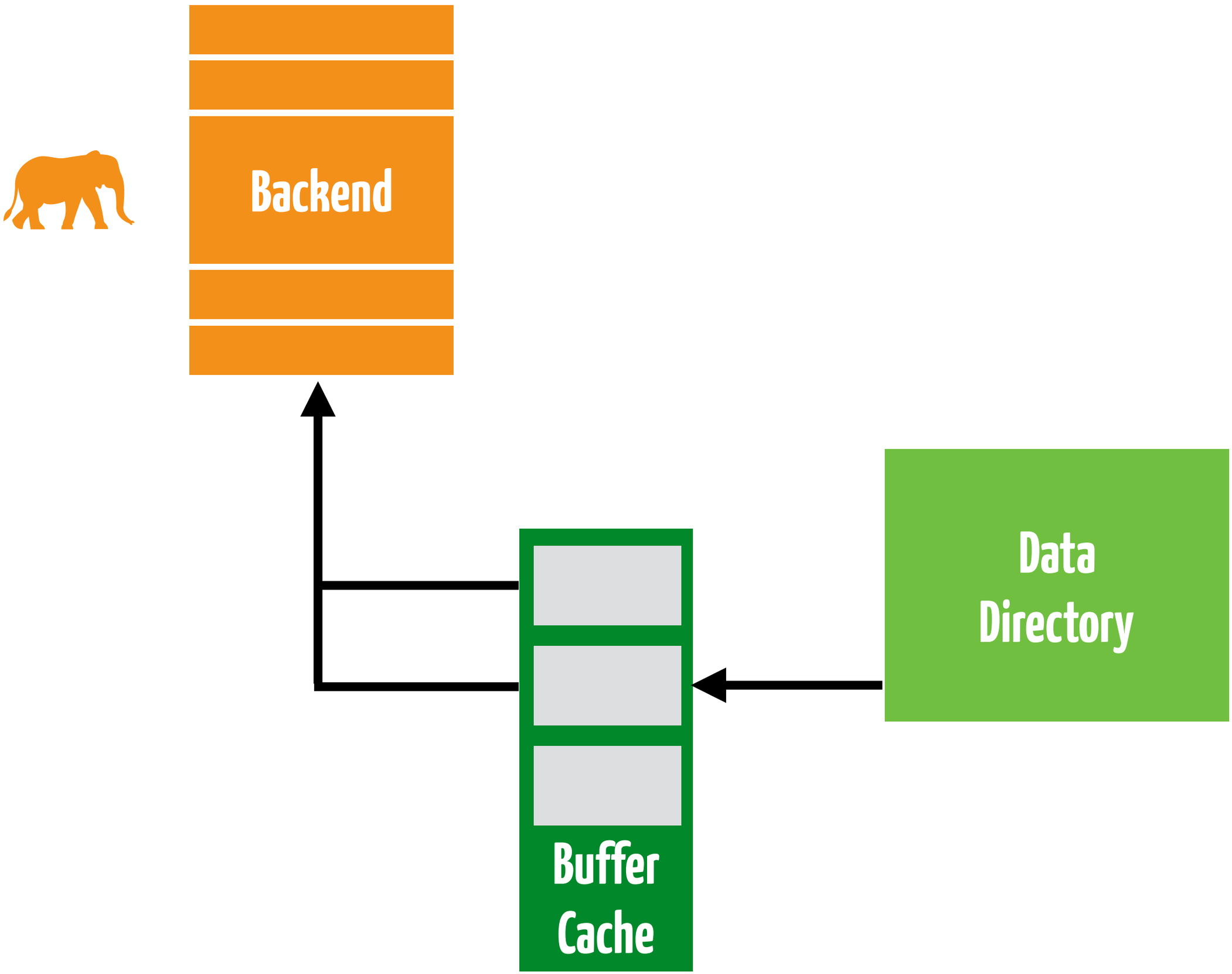
Increase work_mem

However, be aware of OOMs!

When you get a lot of Out of Memory Errors

Reduce work_mem!

Buffer Cache Hit Ratio


Backend
Data
Directory
Buffer
Cache

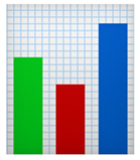
Cache Hit Ratio %

pg_stat_database.**blks_hit**

pg_stat_database.**blks_read**

pg_stat_statements.**shared_blks_hit**

pg_stat_statements.**shared_blks_read**

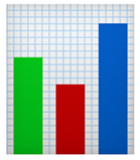


Cache Hit Ratio % (Per Query)

```
shared_blks_hit       | 2447215
shared_blks_read      | 55335
```

```
hit_rate = shared_blks_hit /
           (shared_blks_hit + shared_blks_read)
```

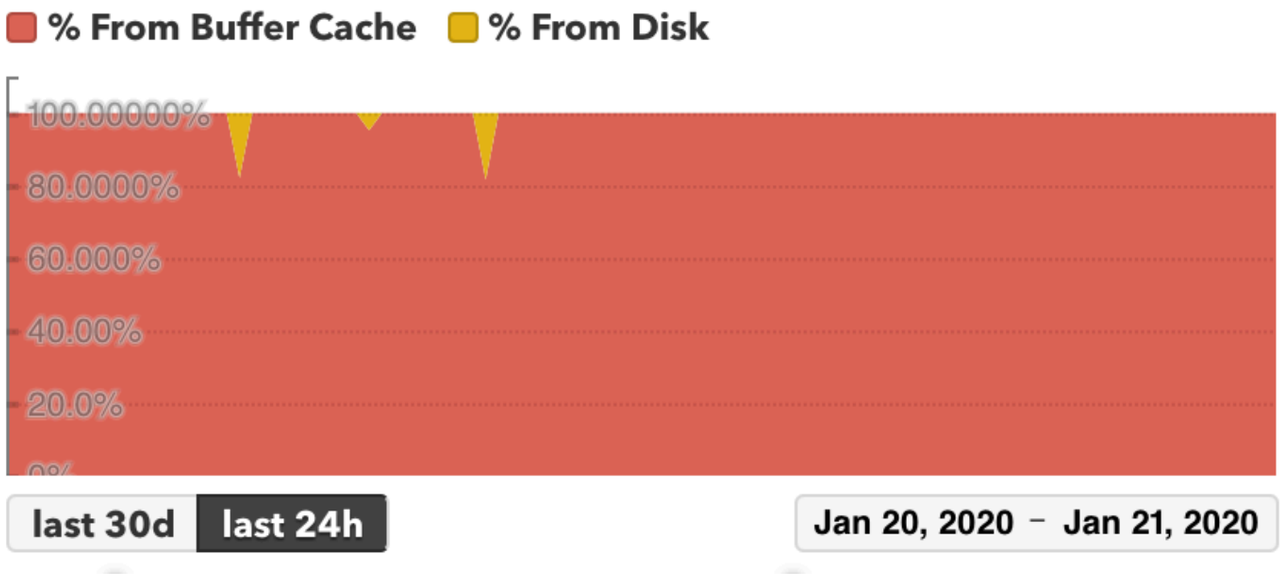
97.78% Cache Hit Rate



Cache Hit Ratio % (Per Table/Index)

```
SELECT sum(heap_blks_hit) /
  nullif(sum(heap_blks_hit + heap_blks_read),0)
  FROM pg_statio_user_tables

SELECT sum(idx_blks_hit) /
  nullif(sum(idx_blks_hit + idx_blks_read),0)
  FROM pg_statio_user_indexes
```



pg_buffercache

Buffer Cache Report

Last Updated: 19 hours ago

Refresh

Report auto-refreshes every 24 hours

Entries

DATABASE	OBJECT TYPE	OBJECT	CACHED BYTES	% OF BUFFER CACHE
pgaweb	table	schema_columns	4.5 GB	58.06%
pgaweb	table	schema_constraints	1 GB	13.10%
pgaweb	table	query_stats_60d	567.1 MB	7.09%
pgaweb	table	schema_indices	430.7 MB	5.39%
pgaweb	index	schema_columns_table_id_name_idx	192.6 MB	2.41%
pgaweb	index	query_fingerprints_unique_index	151.7 MB	1.90%
pgaweb	index	schema_index_stats_60d_20170329_pkey	115.7 MB	1.45%
pgaweb	index	schema_index_stats_2d_20170329_pkey	111.9 MB	1.40%
pgaweb	index	query_stats_60d_20170329_pkey	101.5 MB	1.27%
pgaweb	index	query_stats_2d_20170329_pkey	100 MB	1.25%
pgaweb	table	schema_tables	95.9 MB	1.20%
pgaweb	index	schema_indices_table_id_name_idx	64.5 MB	0.81%
pgaweb	index	schema_columns_pkey	52.3 MB	0.65%
pgaweb	index	schema_table_stats_2d_20170329_pkey	43.7 MB	0.55%
pgaweb	index	schema_table_stats_60d_20170329_pkey	41.6 MB	0.52%

Andres Freund (Tech)
@AndresFreundTec

The often repeated recommendation to not exceed 8GB/25% of RAM for shared buffers in postgres is wrong.

For pgbench scale 1500, on laptop with fast SSD and 32GB of RAM, parallelism of 16.

s_b of 1GB, 8GB, 16GB, using huge pages.

r/o: 115k, 100k, 185k
r/w: 16300, 15100, 21500

1:15 PM · Sep 30, 2019 · TweetDeck

Benchmark with higher shared_buffers

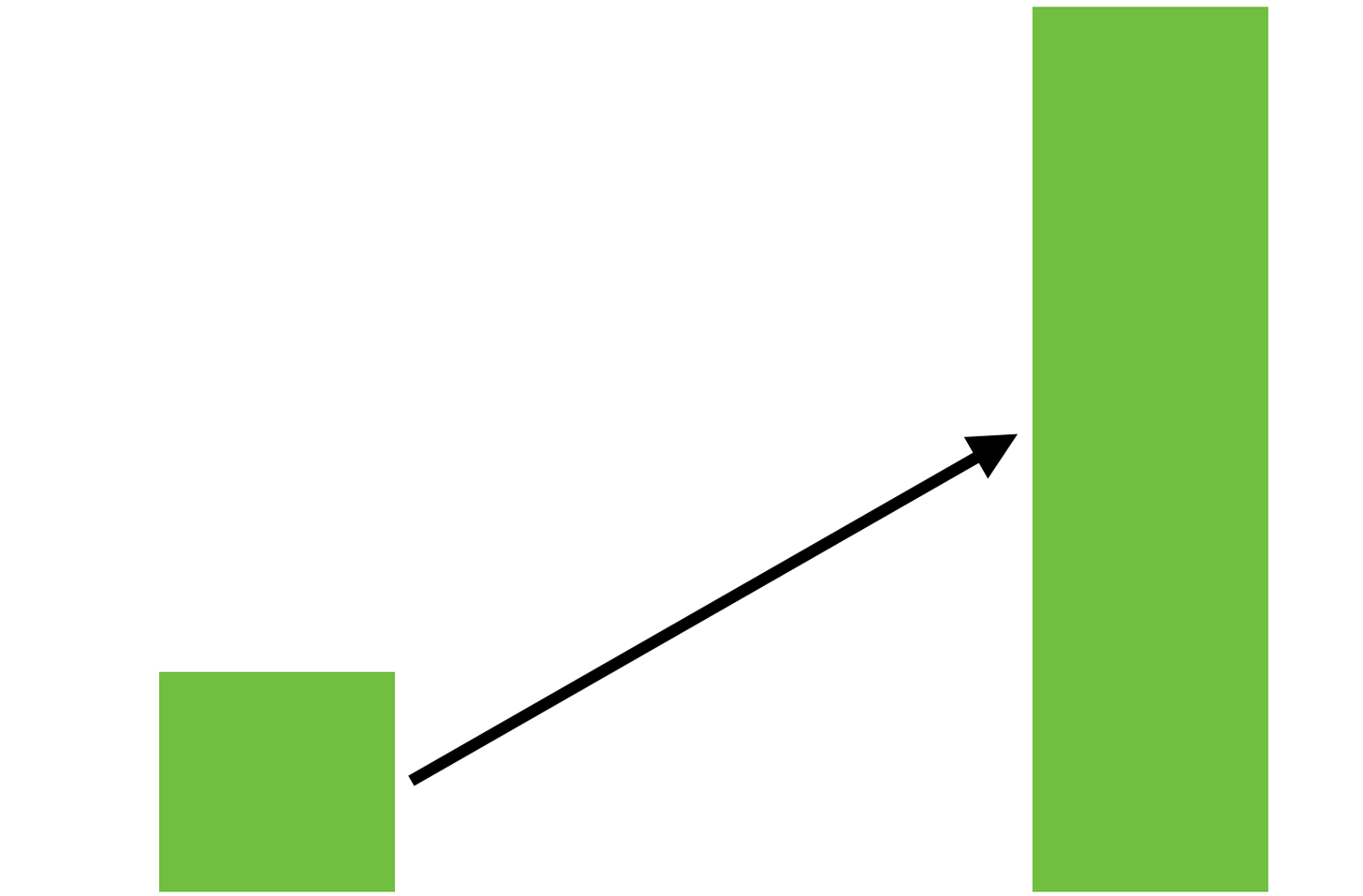
When Your Workload Doesn't Fit In Memory

Change Your Workload

Or

Add More Memory

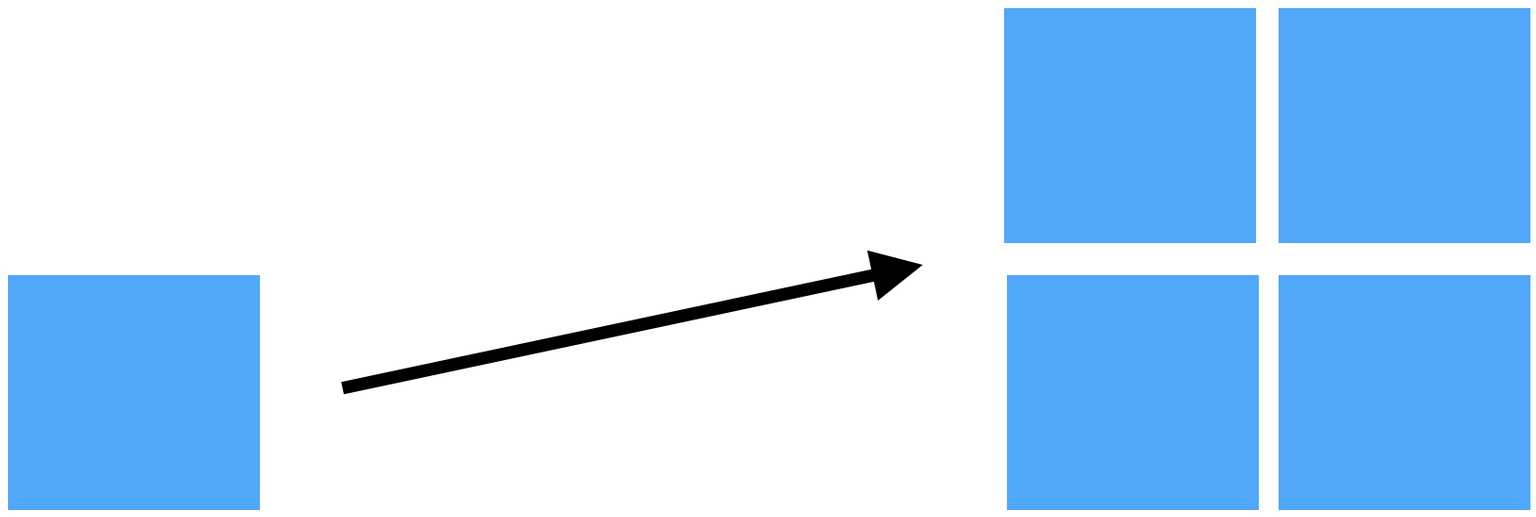
Scaling Up



Scaling Out

Shard in your application

Use a sharded database (e.g. Citus)




3:1
2:1
2:1
3:1
2:1
3:1
2:1
Lock
Contention

Locks Held/Waited On

`pg_locks`

`pid`: process ID
(JOIN to pg stat activity.pid!)
`locktype`: type of object being locked
`mode`: locking type (e.g. AccessExclusive)
`granted`: Lock Granted vs Being Waited For
…

Locks Waited On

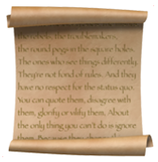
pg_locks

```
SELECT *
  FROM pg_locks
 WHERE NOT granted
```

Locks Held

pg_locks

```
SELECT locktype,
       mode,
       COUNT(*)
  FROM pg_locks
WHERE granted
GROUP BY 1, 2
```



Locks Held/Waited On

`log_lock_waits = on`

```
LOG: process 123 still waiting for ShareLock on transaction 12345678
after 1000.606 ms
STATEMENT: SELECT table WHERE id = 1 FOR UPDATE;
CONTEXT: while updating tuple (1,3) in relation “table”
DETAIL: Process holding the lock: 456. Wait queue: 123.
```

Rewrite Transactions To Hold Locks Shorter To **Reduce Lock Contention**

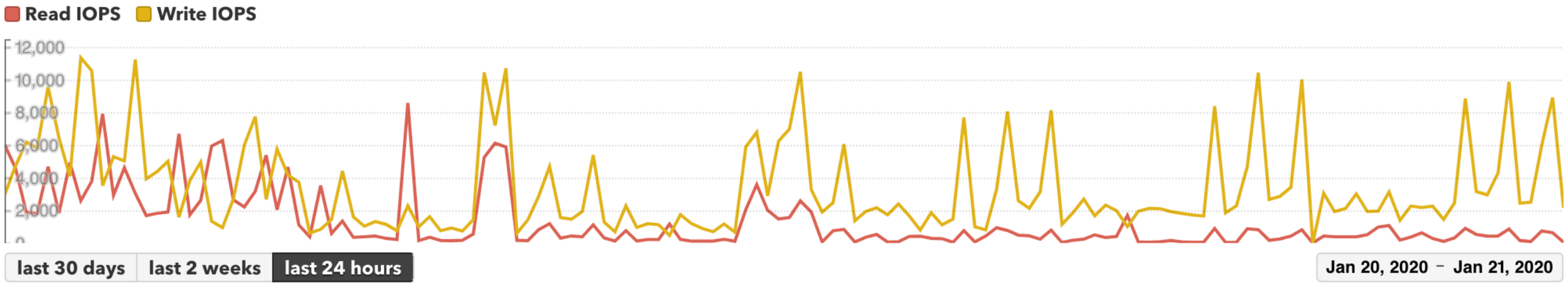
Long Held Locks in Transactions

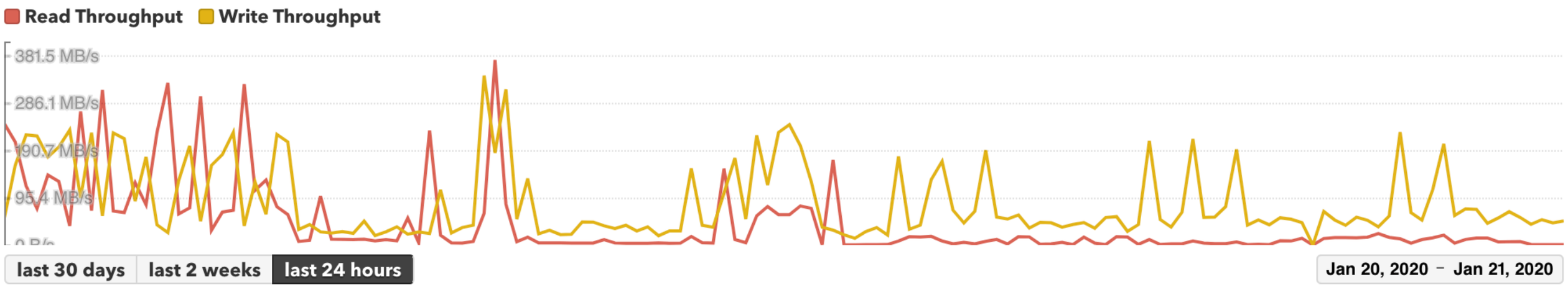
Rails Counter Cache & Timestamps

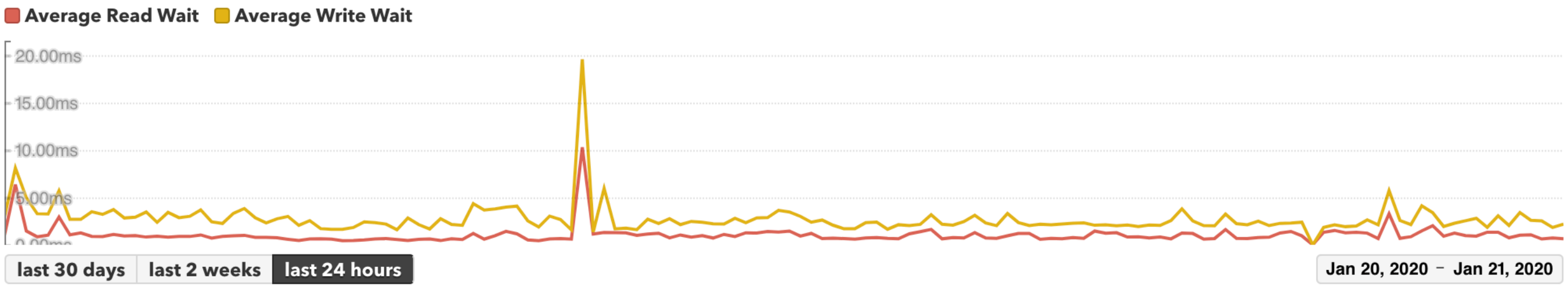
```
BEGIN
SELECT 1 AS one FROM "post_votes"  WHERE (…) LIMIT 1
SELECT "posts".* FROM "posts"  WHERE "posts"."id" = $1 LIMIT 1
INSERT INTO "notifications" (…) VALUES (…) RETURNING "id"
SELECT "users".* FROM "users"  WHERE "users"."id" = $1 LIMIT 1
UPDATE "users" SET "updated_at" = ? WHERE "users"."id" = ?
INSERT INTO "post_votes" (…) VALUES (…) RETURNING "id"
UPDATE "posts" SET "votes" = COALESCE("votes", 0) + 1 WHERE "posts"."id" = ?
UPDATE "posts" SET "credible_post_votes_count" = … WHERE "posts"."id" = ?
UPDATE "users" SET "updated_at" = ? WHERE "users"."id" = ?
UPDATE "posts" SET "updated_at" = ? WHERE "posts"."id" = ?
COMMIT
```

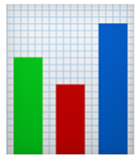

3:1
2:1
2:1
3:1
2:1
3:1
2:1
3:1
I/O Workload

System-Level I/O Metrics









Data Read from Disk / OS Cache

pg_stat_database.**blks_read**

↓

pg_statio_all_tables.**heap_blks_read**

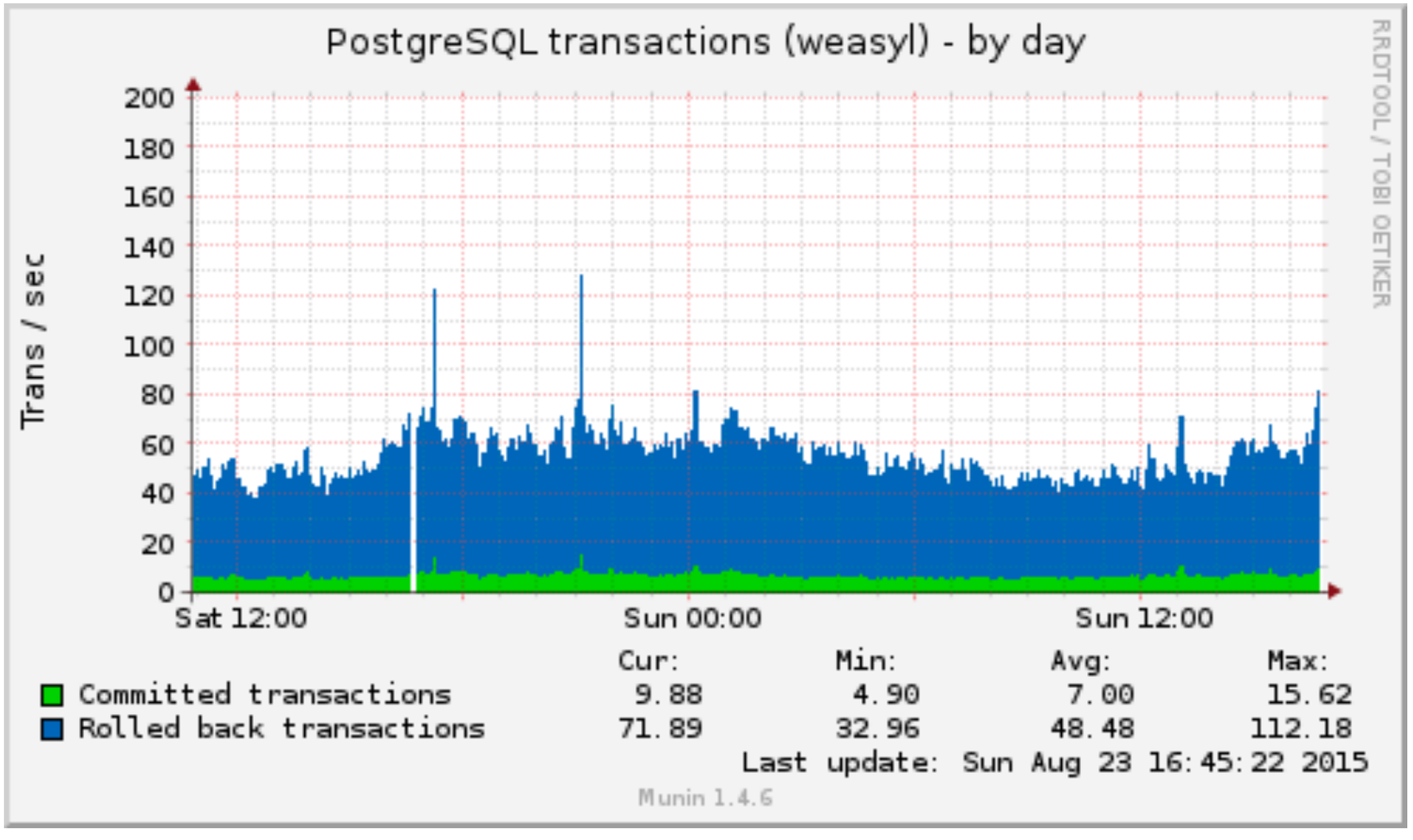
pg_statio_all_indexes.**idx_blks_read**

↓

pg_stat_statements.**shared_blks_read**

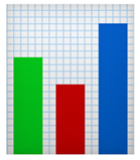
Transactions Per Second

pg_stat_database.**xact_commit**
pg_stat_database.**xact_rollback**

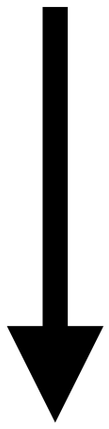


Time spent reading/writing to disk

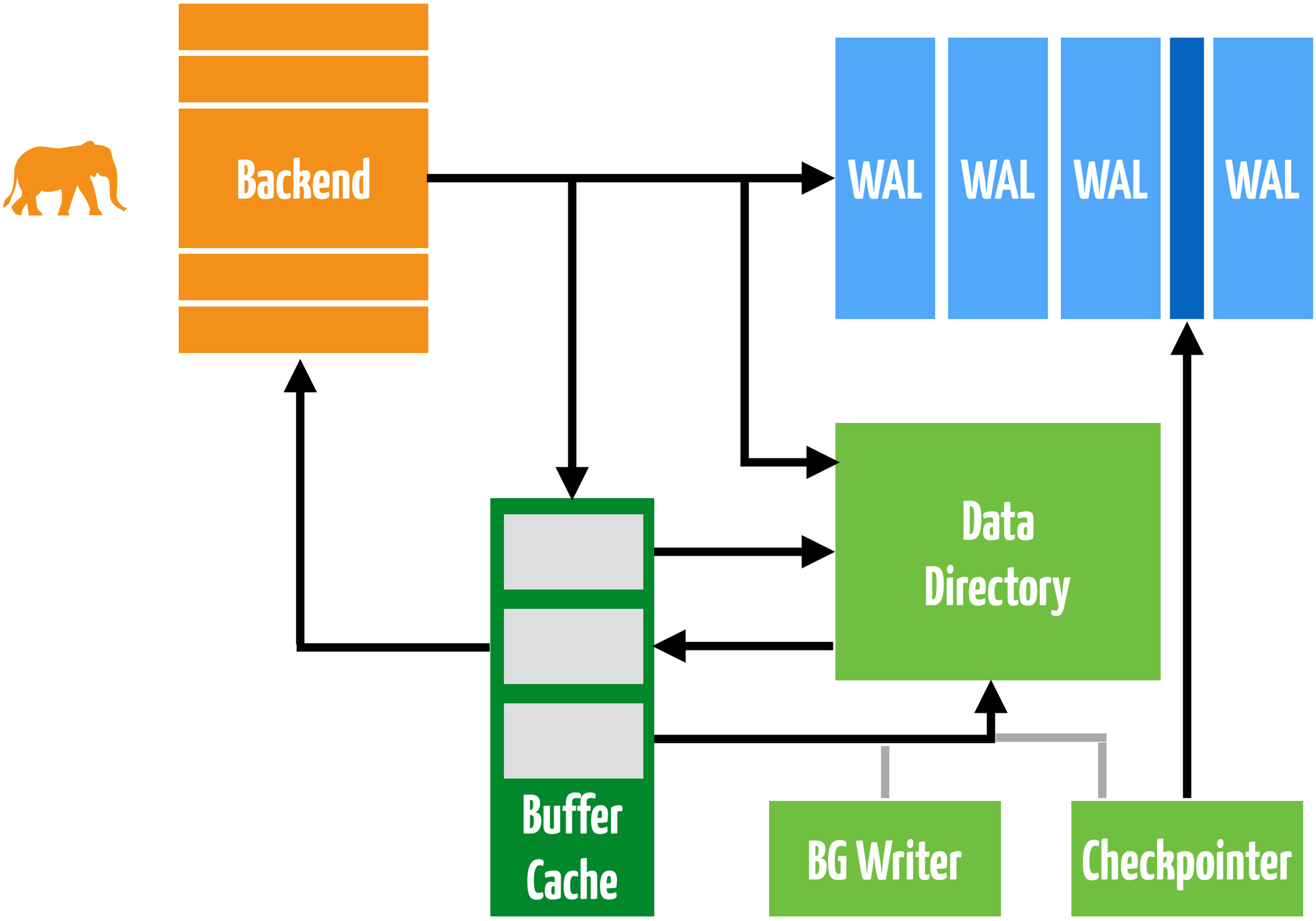
```
track_io_timing = on
```

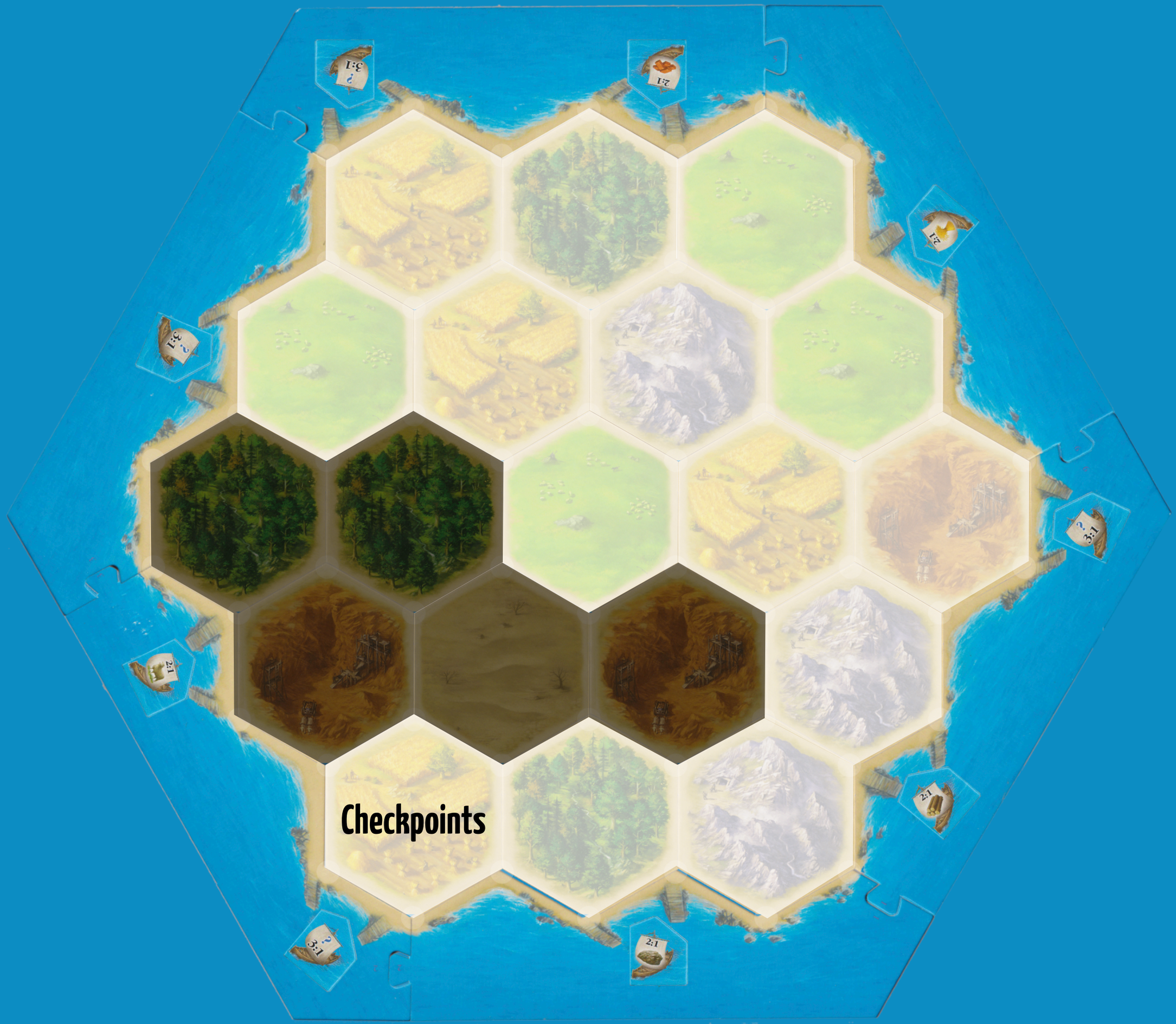


pg_stat_database.**blk_read_time**
pg_stat_database.**blk_write_time**



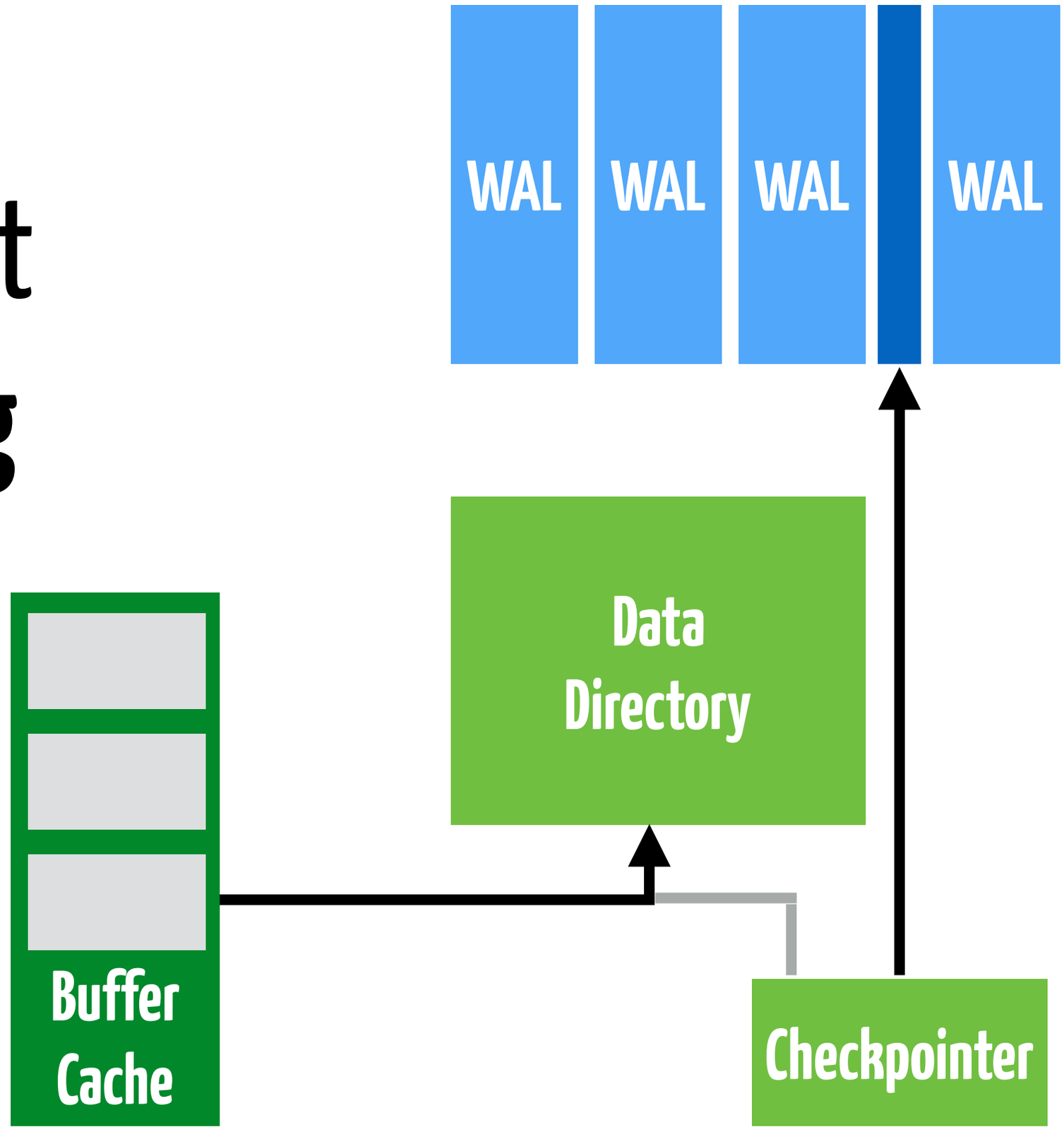
pg_stat_statements.**blk_read_time**
pg_stat_statements.**blk_write_time**

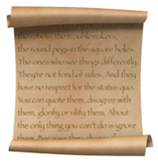

Backend
WAL
WAL
WAL
WAL
Data
Directory
Buffer
Cache
BG Writer
Checkpointer



Checkpoints

Checkpoints Are Important **For I/O Tuning**

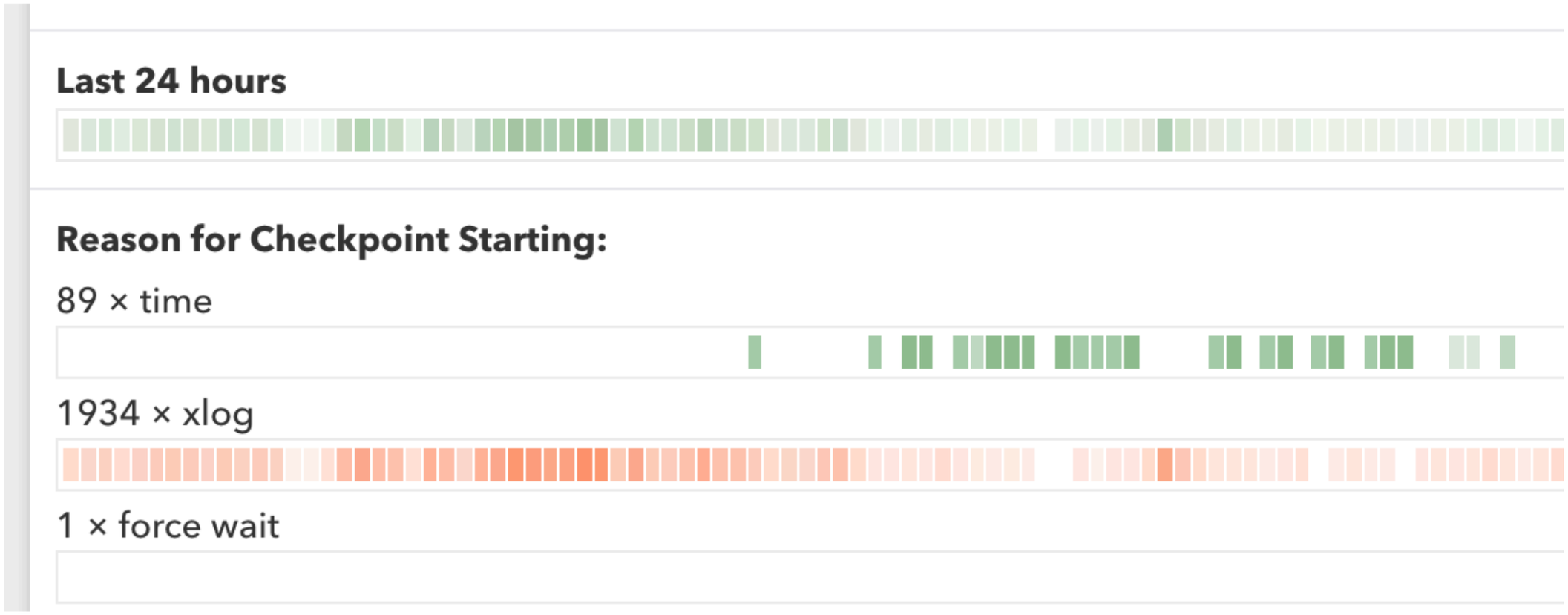


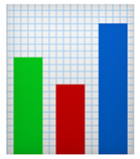


```
16688 LOG: checkpoint starting: xlog
```

xlog = WAL exceeded max_wal_size, checkpoint has to happen quickly

time = checkpoint_timeout reached, checkpoint impact spread over time





Checkpoint Statistics

pg_stat_bgwriter

```
checkpoints_timed: # of scheduled checkpoints
  checkpoints_req: # of requested checkpoints
```

1. Time Between Checkpoints
2. % of Timed Checkpoints

Increase max_wal_size / Reduce checkpoint_timeout

To Have More Timed Checkpoints

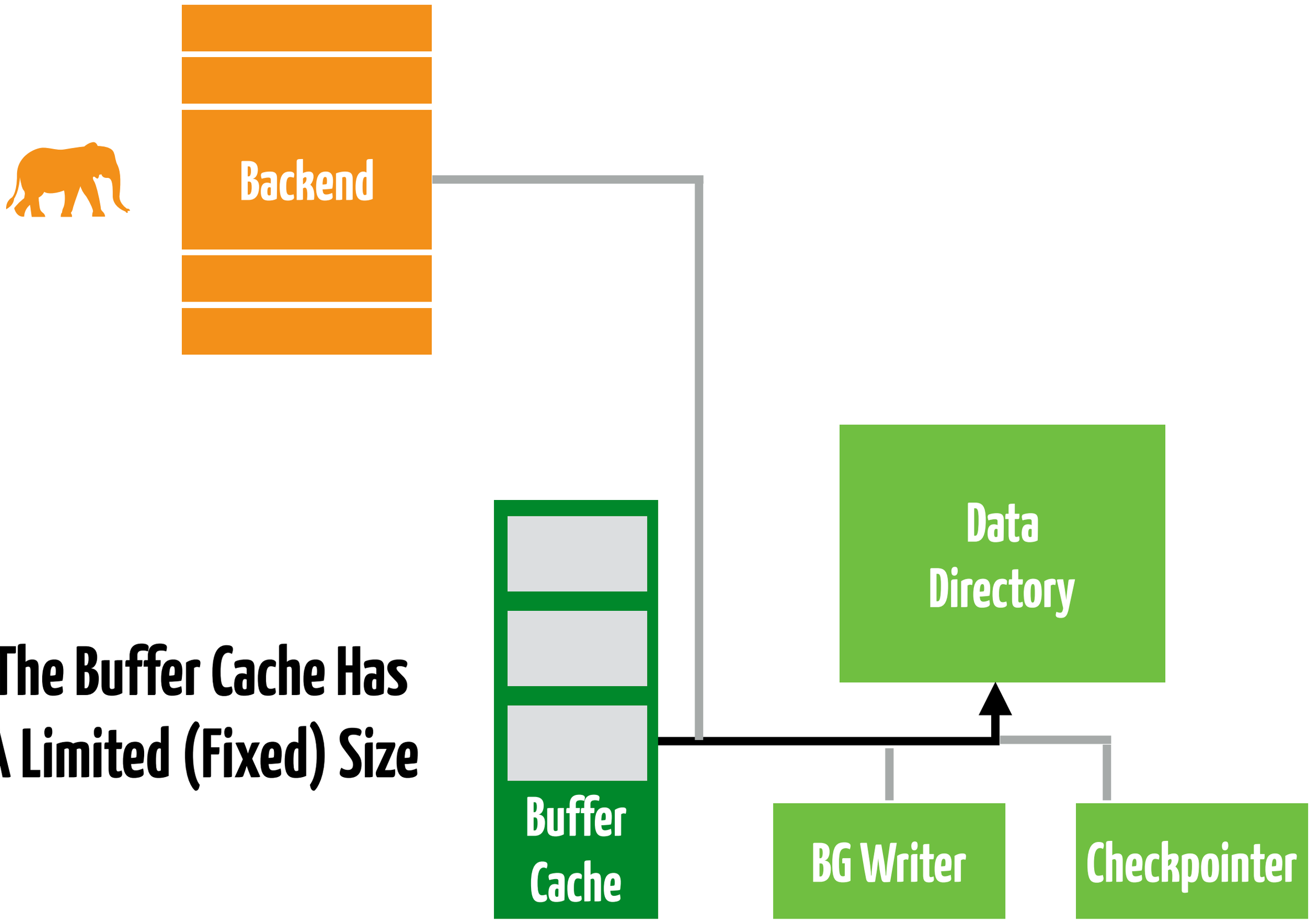
(but be careful with recovery times)

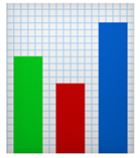
Tune checkpoint_completion_target To Control I/O Impact of Timed Checkpoints

(Often **0.9** is a good value, but depends on I/O Subsystem & Workload)

Background Writer

The Buffer Cache Has A Limited (Fixed) Size





Who wrote the Buffers?

pg_stat_bgwriter

BG Writer — `buffers_clean`

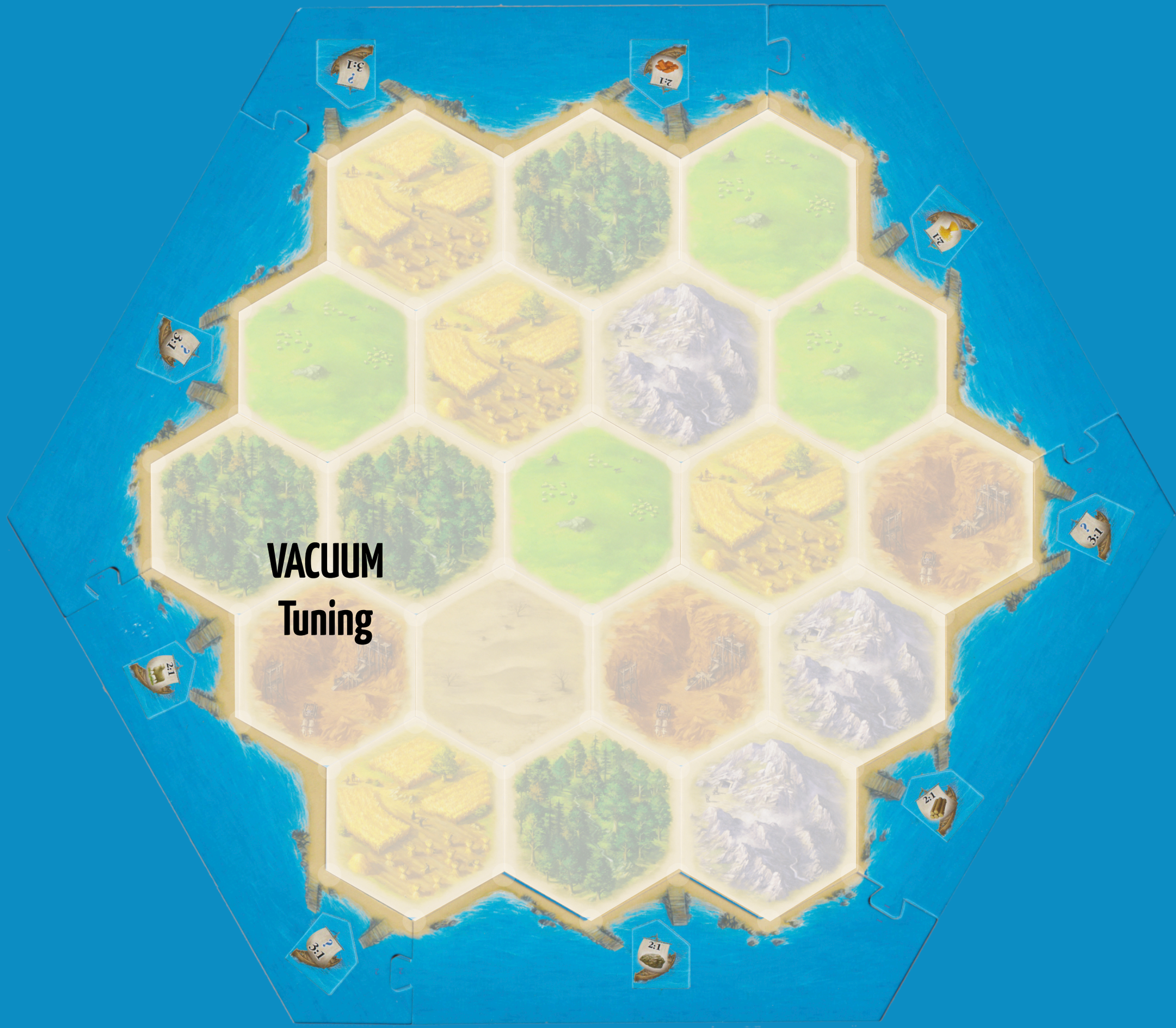
Checkpointer — `buffers_checkpoint`

Backend — `buffers_backend`

Reduce bgwriter_delay & Raise bgwriter_lru_maxpages

To Have More Buffers Written By The BG Writer

VACUUM
Tuning



autovacuum

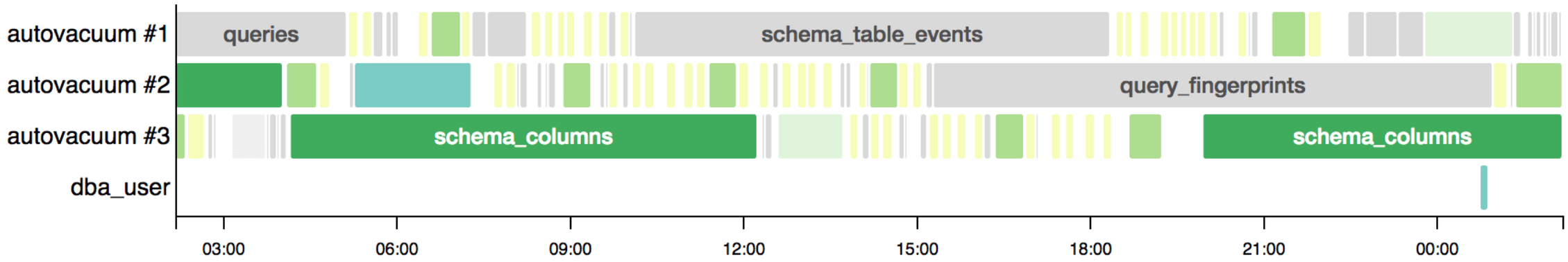
pg_stat_activity

```
=> SELECT pid, query FROM pg_stat_activity
     WHERE query LIKE 'autovacuum: %';

 10469 | autovacuum: VACUUM ANALYZE public.schema_columns
 12848 | autovacuum: VACUUM public.replication_follower_stats
 28626 | autovacuum: VACUUM public.schema_index_stats
       | (to prevent wraparound)
(3 rows)
```

autovacuum

pg_stat_activity



ID	Start	End	Table	Role	Autovacuum
1511841418010469	7:56:58 PM PST	currently running	public.schema_columns	rdsadmin	Yes
1511860908007950	1:21:48 AM PST	currently running	public.schema_indices	rdsadmin	Yes
1511863592012848	2:06:31 AM PST	2:10:00 AM PST	public.postgres_settings	rdsadmin	Yes

autovacuum

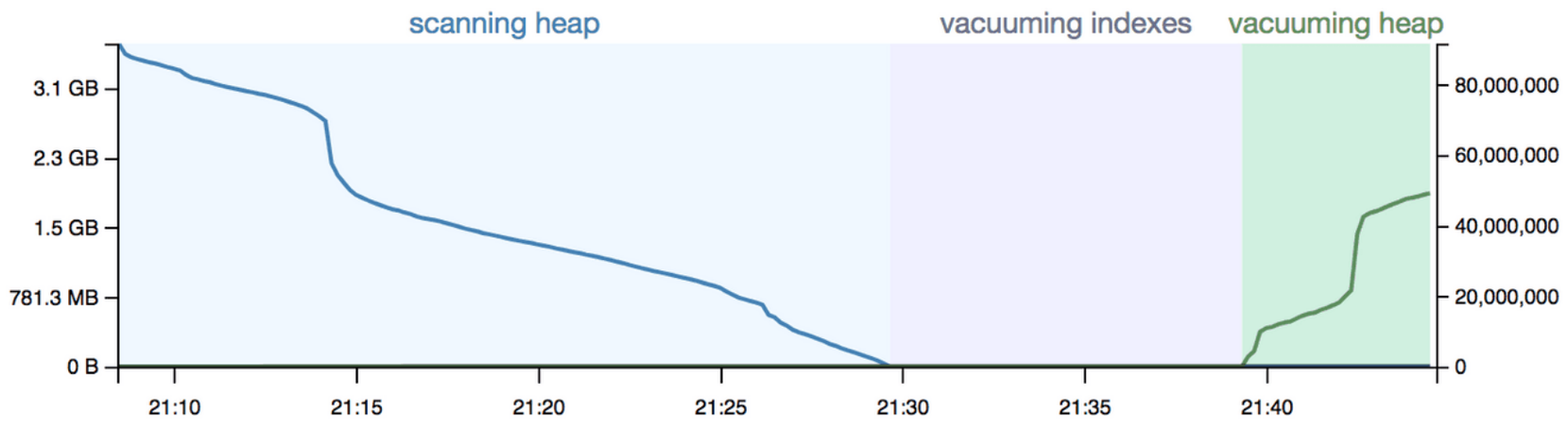
pg_stat_progress_vacuum

relid: OID of the table
phase: current VACUUM phase
heap_blks_total: Heap Blocks Total
heap_blks_scanned: Heap Blocks Scanned
heap_blks_vacuumed: Heap Blocks Vacuumed
…

autovacuum

pg_stat_progress_vacuum

Table Name	public.schema_indices	**Start Time**	Nov 27, 2017 9:08:27 PM PST
Postgres Role	postgres	**End Time**	Nov 27, 2017 9:44:40 PM PST
Heap Blocks Total	3.5 GB · 464,038 blocks	**Max Dead Tuples / Phase**	91,575,637



Reduce autovacuum_vacuum_cost_delay

To Increase VACUUM Speed

Older PG Default

8 MB/s

(20ms)

PG 12+

80 MB/s

(2ms)

OS / Disk Reads

Use Table Partitioning For Append-Only + Delete Workloads (e.g. Timeseries)


Buffer Cache Hit Ratio
Index Optimization
work_mem Tuning
Connection Pooling
Query Tags
Query Workload
VACUUM Tuning
Background Writer
Lock Contention
Checkpoints
I/O Workload
3:1
2:1
2:1
3:1
2:1
3:1
2:1
3:1
2:1

Thanks!

@LukasFittl