What's Missing For Postgres Monitoring

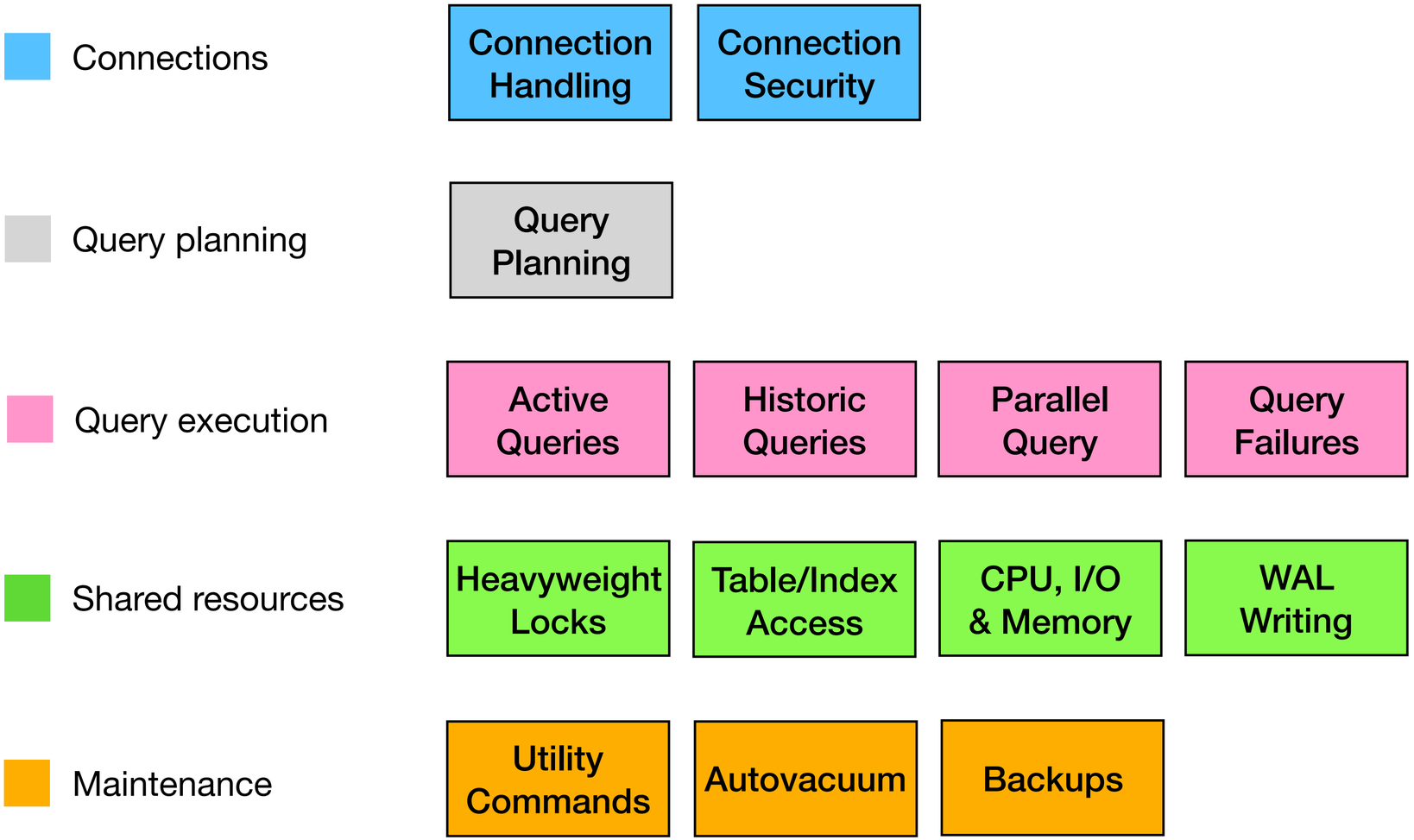
@LukasFittl

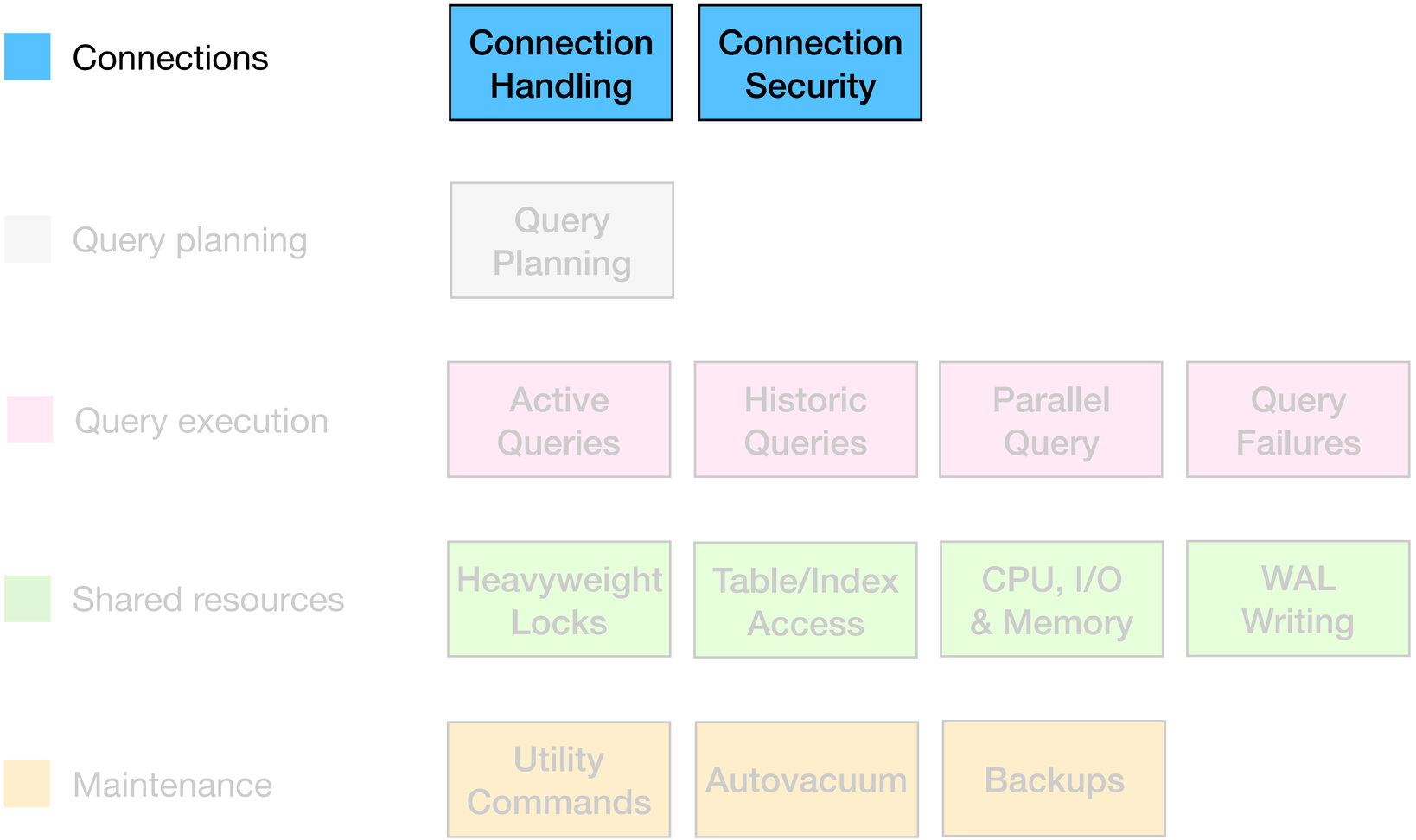
@LukasFittl

What are the problems with Postgres monitoring?

It’s incomplete.
It’s hard to access & understand.
It contains sensitive information.

It’s incomplete.
It’s hard to access & understand.
It contains sensitive information.


Connections
Connection Handling
Connection Security
Query planning
Query Planning
Query execution
Active Queries
Historic Queries
Parallel Query
Query Failures
Shared resources
Heavyweight Locks
Table/Index Access
CPU, I/O & Memory
WAL Writing
Maintenance
Utility Commands
Autovacuum
Backups


Connections
Connection Handling
Connection Security
Query planning
Query Planning
Query execution
Active Queries
Historic Queries
Parallel Query
Query Failures
Shared resources
Heavyweight Locks
Table/Index Access
CPU, I/O & Memory
WAL Writing
Maintenance
Utility Commands
Autovacuum
Backups

Connection Handling

pg_stat_activity

Log events

Connection received
Disconnection
Incomplete startup packet (client failed to connect)
Could not receive data from client / connection to client lost
EOF on client connection with an open transaction
Terminating connection due to administrator command
Remaining connection slots are reserved for superuser (out of connections)
Too many connections for role
Could not accept SSL connection
Unsupported frontend protocol
Incomplete message from client

What's Missing

Client-side connection latency

Hard to track from the Postgres server side

- could libpq have built-in measurements here?
- should **\timing** in psql give connection time and planning/execution time separately?

Connection Security

pg_stat_ssl

pg_stat_gssapi

Log events

Connection authorized

Authentication failed / pg_hba.conf rejects connection

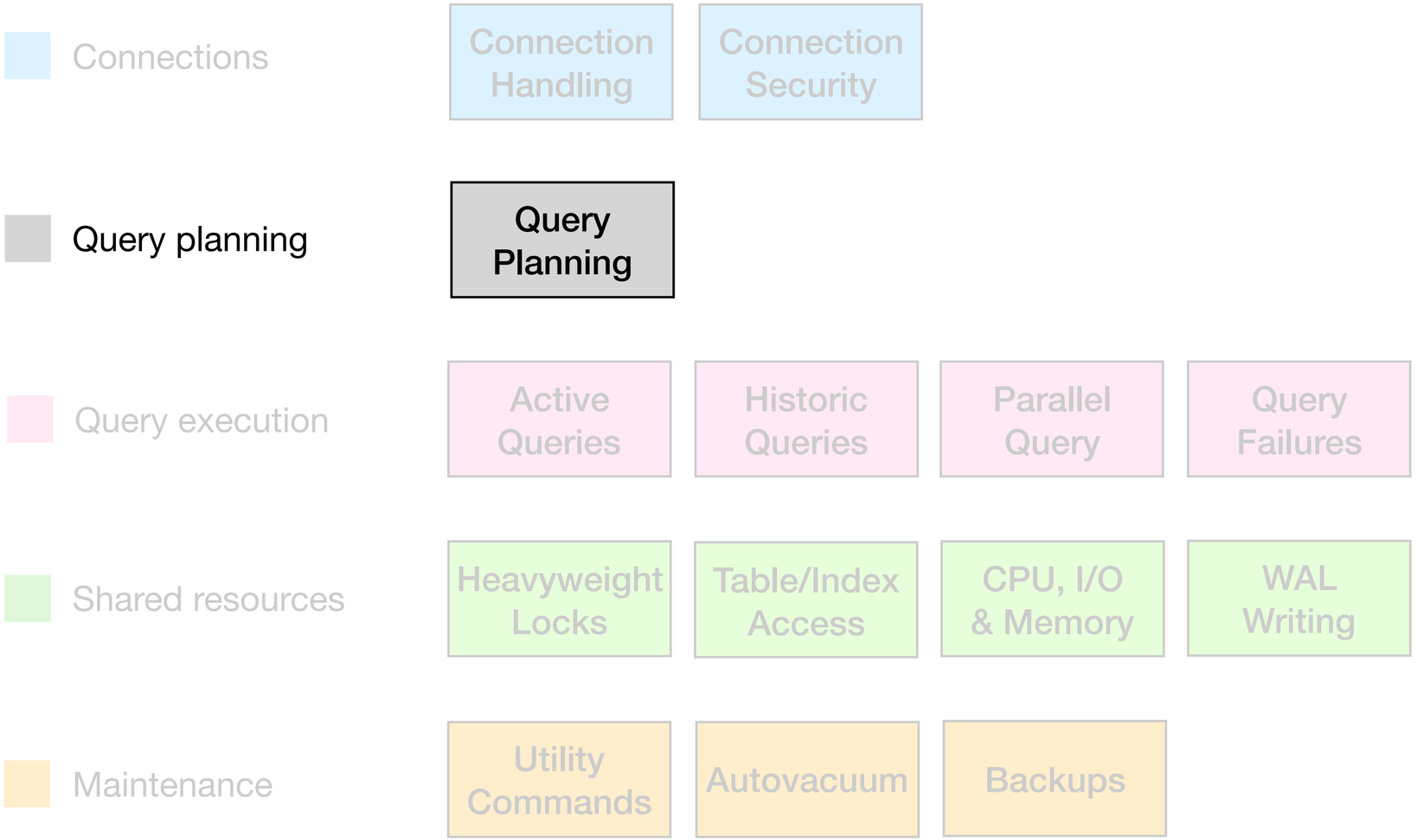
What's Missing

Aggregation of security-relevant Postgres events

Which IPs logged in as superuser?

How many login failures occurred recently?

Which of my pg_hba lines are matching?


Connections
Connection Handling
Connection Security
Query planning
Query Planning
Query execution
Active Queries
Historic Queries
Parallel Query
Query Failures
Shared resources
Heavyweight Locks
Table/Index Access
CPU, I/O & Memory
WAL Writing
Maintenance
Utility Commands
Autovacuum
Backups

Query Planning

EXPLAIN

New EXPLAIN: Buffers for Planning

New pg_stat_statements planning time

Log events

auto_explain

New in Postgres 13

EXPLAIN: Buffers for Planning

```
                                                   QUERY PLAN
----------------------------------------------------------------------------------------------------------------------------------
 Limit  (cost=0.00..0.03 rows=1 width=86) (actual time=0.446..0.446 rows=1 loops=1)
   Buffers: shared read=1
   ->  Append  (cost=0.00..103559.88 rows=3406392 width=86) (actual time=0.445..0.446 rows=1 loops=1)
         Buffers: shared read=1
         ->  Seq Scan on query_stats_hourlies_60d_20200127 query_stats_hourlies_60d_1  (cost=0.00..527.90 rows=20790 width=86) (actual time=0.440..0.440 rows=1 loops=1)
               Buffers: shared read=1
         ->  Seq Scan on query_stats_hourlies_60d_20200128 query_stats_hourlies_60d_2  (cost=0.00..723.93 rows=28493 width=86) (never executed)
         ...
 Planning Time: 45.882 ms
   Buffers: shared hit=8306 read=435 dirtied=10
 Execution Time: 0.446 ms
(128 rows)
```

New in Postgres 13

pg_stat_statements: Planning Time

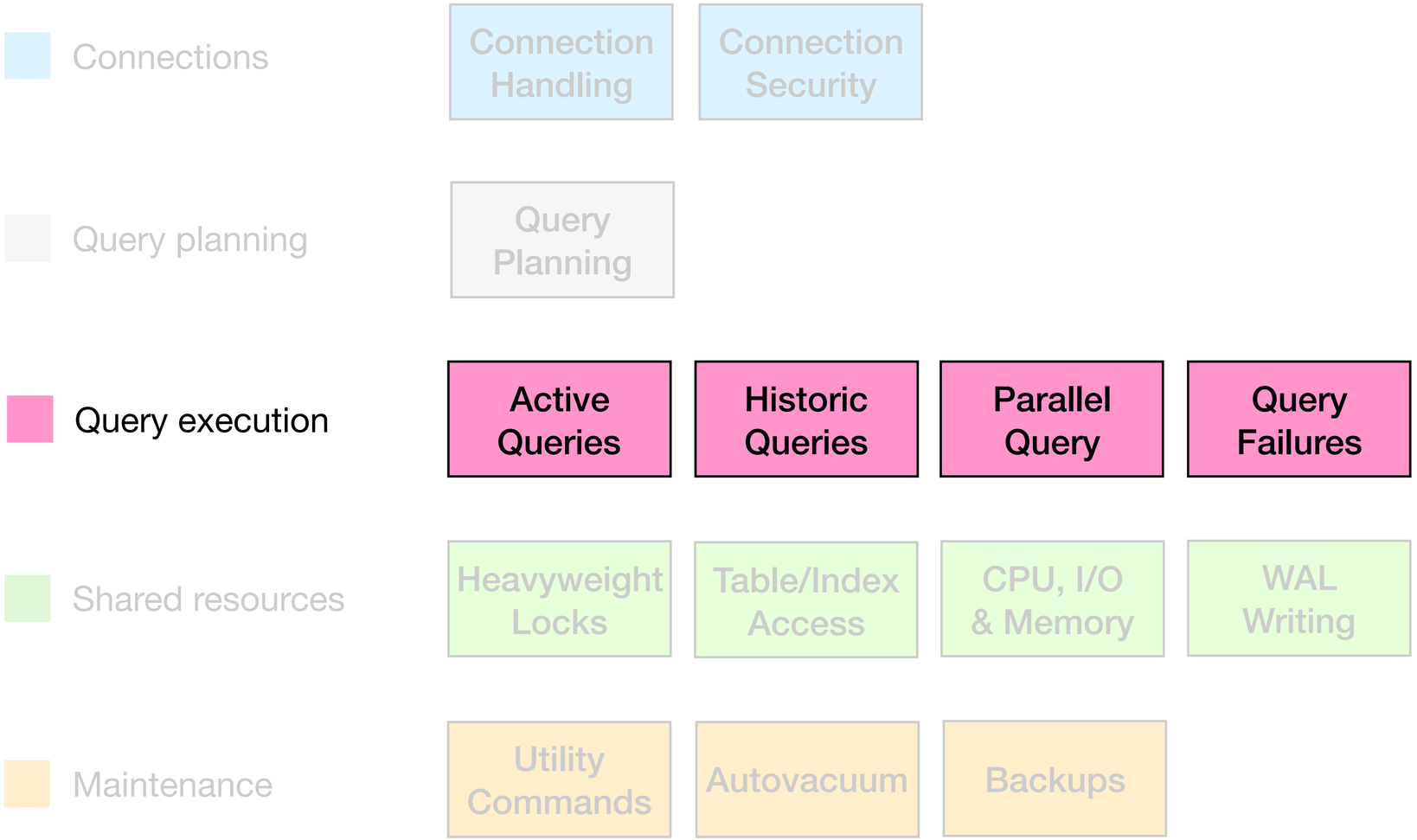
```
=# SELECT queryid, substring(query for 40), mean_exec_time, mean_plan_time, max_plan_time FROM pg_stat_statements ORDER BY mean_plan_time
LIMIT 5;
       queryid        |                substring                 |   mean_exec_time   |  mean_plan_time   | max_plan_time
----------------------+------------------------------------------+--------------------+-------------------+---------------
   586048399314747810 | WITH upsert(backend_id, server_id, ident |           0.440361 |          5.890649 |      5.890649
  5426874022189006220 | WITH data(table_id, name, first_snapshot |          18.846979 |          5.452164 |      5.452164
  3576712877697568576 | WITH data(table_id, name, first_snapshot |  17.85431479746835 | 5.032493797468352 |     12.714722
 -1758450264182311255 | WITH data(table_id, name, first_snapshot | 17.870344956521738 | 4.544071499999999 |      6.236185
 -1076182304104233502 | WITH data(table_id, name, first_snapshot | 15.446047395348836 | 3.378207406976743 |      5.378551
(5 rows)
```

What's Missing

Aggregate Plan Statistics

Many experimental Postgres extensions (pg_stat_plans, pg_store_plans, pg_stat_sql_plans, etc)

Not production ready, or merge-able into Postgres core


Connections
Connection Handling
Connection Security
Query planning
Query Planning
Query execution
Active Queries
Historic Queries
Parallel Query
Query Failures
Shared resources
Heavyweight Locks
Table/Index Access
CPU, I/O & Memory
WAL Writing
Maintenance
Utility Commands
Autovacuum
Backups

Active Queries

pg_stat_activity

(state, query_start, xact_start, wait events)

New in Postgres 13

Additional & renamed wait events

Report wait event for cost-based vacuum delay.
Add description about LogicalRewriteTruncate wait event into document.
Add description about GSSOpenServer wait event into document.
Correct the descriptions of recovery-related wait events in docs.
Rename the recovery-related wait events.
Add wait events for WAL archive and recovery pause.
Add wait events for recovery conflicts.
Report missing wait event for timeline history file.
Report time spent in posix_fallocate() as a wait event.
Drop the redundant "Lock" suffix from LWLock wait event names.
Mop-up for wait event naming issues.

What's Missing

Breakdown of non-waiting active state

```
postgres=# SELECT state, wait_event_type, wait_event, substring(query for 100) FROM pg_stat_activity WHERE backend_type = 'client backend';
 state  | wait_event_type | wait_event |                                                  substring
--------+-----------------+------------+-------------------------------------------------------------------------------------------------
 active |                 |            | COPY public.log_lines_30d_20200516 (log_line_id, server_id, backend_pid, occurred_at, log_file_id, l
 active |                 |            | COPY public.log_lines_30d_20200514 (log_line_id, server_id, backend_pid, occurred_at, log_file_id, l
 active |                 |            | COPY public.log_lines_30d_20200517 (log_line_id, server_id, backend_pid, occurred_at, log_file_id, l
 active |                 |            | COPY public.log_lines_30d_20200515 (log_line_id, server_id, backend_pid, occurred_at, log_file_id, l
 active |                 |            | SELECT state, wait_event_type, wait_event, substring(query for 100) FROM pg_stat_activity WHERE back
 idle   | Client          | ClientRead |
(6 rows)
```

perf top -g

```
Samples: 379K of event 'cpu-clock:pppH', 4000 Hz, Event count (approx.): 55672843733 lost: 0/0 drop: 15165/199698
  Children      Self  Shared Object             Symbol
+   58.01%     0.91%  postgres                  [.] CopyFrom
+   46.72%     1.54%  postgres                  [.] NextCopyFrom
+   23.68%     0.98%  postgres                  [.] InputFunctionCall
+   20.72%     5.71%  postgres                  [.] NextCopyFromRawFields
+   15.13%     0.03%  perf                      [.] __ordered_events__flush.part.0
+   15.08%     0.03%  perf                      [.] deliver_event
+   14.73%     0.02%  perf                      [.] hist_entry_iter__add
+   11.81%     0.82%  perf                      [.] iter_add_next_cumulative_entry
+   11.60%     0.45%  postgres                  [.] timestamp_in
+    8.77%     0.99%  postgres                  [.] DecodeDateTime
+    8.36%     0.22%  [kernel]                  [k] do_syscall_64
+    7.29%     0.27%  [kernel]                  [k] __softirqentry_text_start
+    6.97%     0.01%  [kernel]                  [k] net_rx_action
+    6.54%     0.01%  [kernel]                  [k] ena_io_poll
+    6.13%     0.00%  libc-2.31.so              [.] __libc_start_main
```

perf top -g

```
Samples: 379K of event 'cpu-clock:pppH', 4000 Hz, Event count (approx.): 55672843733 lost: 0/0 drop: 15165/199698
  Children      Self  Shared Object             Symbol
-   58.01%     0.91%  postgres                  [.] CopyFrom
   - 7.70% CopyFrom
      - 11.96% NextCopyFrom
         + 15.40% NextCopyFromRawFields
         - 11.65% InputFunctionCall
              4.84% uuid_in
           + 4.26% timestamp_in
      + 1.70% heap_multi_insert
   + 0.91% __libc_start_main
+   46.72%     1.54%  postgres                  [.] NextCopyFrom
+   23.68%     0.98%  postgres                  [.] InputFunctionCall
+   20.72%     5.71%  postgres                  [.] NextCopyFromRawFields
+   15.13%     0.03%  perf                      [.] __ordered_events__flush.part.0
+   15.08%     0.03%  perf                      [.] deliver_event
+   14.73%     0.02%  perf                      [.] hist_entry_iter__add
+   11.81%     0.83%  perf                      [.] iter_add_next_cumulative_entry
```

What's Missing

Query Progress Monitoring

Historic Queries

pg_stat_statements

Log Events

Slow query (log_min_duration_statement)

Statement notice (log_statement)

auto_explain

What's Missing

Better handling of IN(…) lists & other ORM patterns

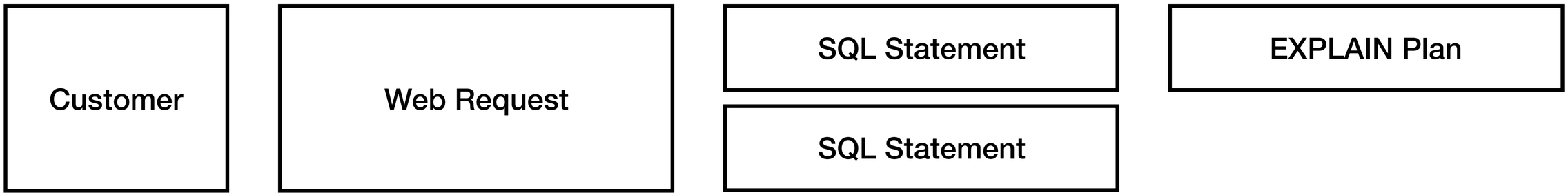
What's Missing

Linking pg_stat_statements with other views & logs

What's Missing

Finding queries based on application requests/customers

pg_stat_statements has no way of differentiating queries beyond the queried



Which customers were affected by a slow query?

What was the EXPLAIN plan for a SQL query involved in a particular slow web request?

Solution for per-customer analysis:

“citus_stat_statements” in Citus extension

```
SELECT partition_key as tenant_id,
       count(*) as tenant_unique_queries,
       sum(calls) as tenant_total_queries,
       sum(total_time) as total_query_time
FROM citus_stat_statements
WHERE partition_key is not null
GROUP BY tenant_id
ORDER BY tenant_total_queries DESC
LIMIT 5;

 tenant_id | tenant_unique_queries | tenant_total_queries | total_query_time
-----------+-----------------------+----------------------+------------------
 12        |                   148 |               159295 |        753142.54
 2         |                  2045 |                23846 |         12957.83
 1         |                    74 |                 9572 |          8492.05
 634       |                   175 |                12753 |          6876.11
 361       |                   375 |                 3653 |          6422.93
(5 rows)
```

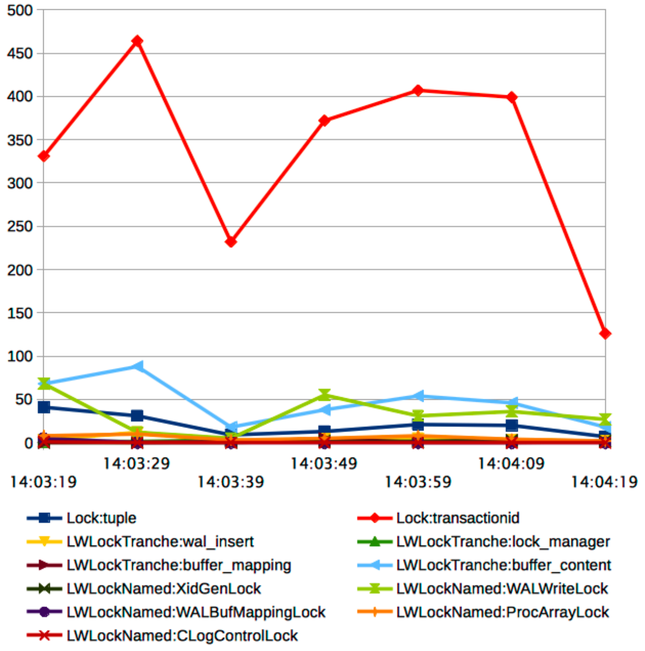
Solution for finding particular web requests:

Application adds comments to locate specific queries + auto_explain/log_min_duration_statement

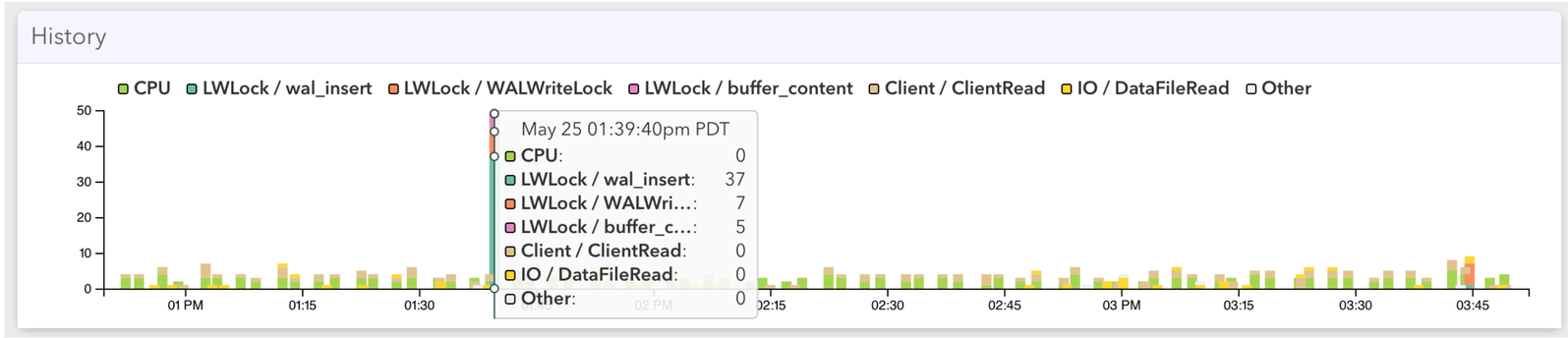
```
/*
application:pganalyze,
controller:graphql,
action:graphql,
line:/app/services/dataload.rb:39:in `select_rows',
graphql:getQueryDetailStats,
request_id:55a6fa2d-9ffe-4374-a535-f5d1ee64ba84
*/
```

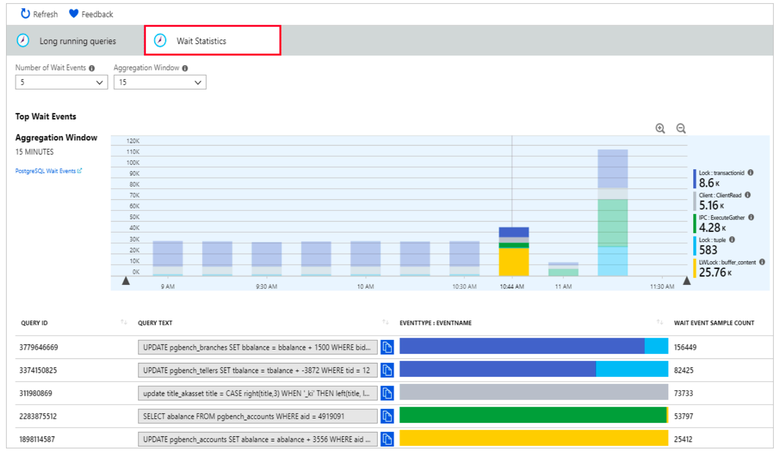
What's Missing

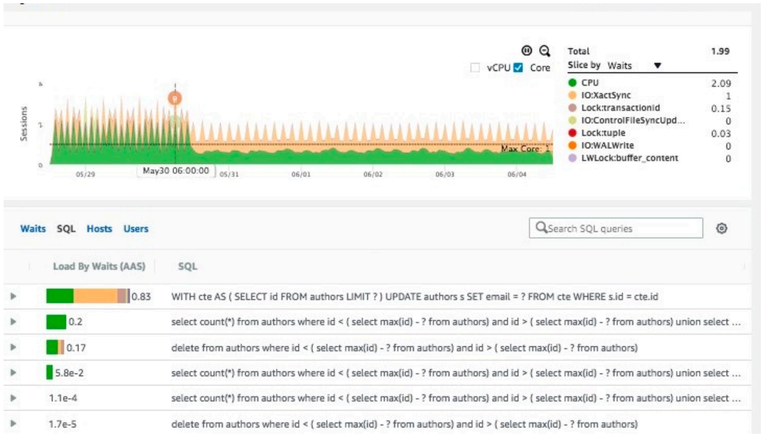
Built-in Wait Event Aggregation



pg_wait_sampling







Parallel Query

pg_stat_activity
(backend_type = parallel worker)

New pg_stat_activity
(leader_pid)

New EXPLAIN improvements

New in Postgres 13

pg_stat_activity: leader_pid for Parallel Query

```
# SELECT backend_type, leader_pid, state, wait_event, wait_event_type, query FROM pg_stat_activity WHERE state <> 'idle';
  backend_type   | leader_pid | state  |  wait_event  | wait_event_type |                                 query
-----------------+------------+--------+--------------+-----------------+-------------------------------------------------------------
 client backend  |      36936 | active | DataFileRead | IO              | SELECT * FROM log_lines_30d ORDER BY occurred_at DESC LIMIT 10;
 parallel worker |      36936 | active | DataFileRead | IO              | SELECT * FROM log_lines_30d ORDER BY occurred_at DESC LIMIT 10;
 parallel worker |      36936 | active | DataFileRead | IO              | SELECT * FROM log_lines_30d ORDER BY occurred_at DESC LIMIT 10;
(3 rows)
```

New in Postgres 13

EXPLAIN improvements for parallel workers

```
                                                    QUERY PLAN
-----------------------------------------------------------------------------------------------------------------------------------
 Limit  (cost=2333874.04..2333876.38 rows=20 width=206) (actual time=29049.924..29049.934 rows=20 loops=1)
   ->  Gather Merge  (cost=2333874.04..7549291.60 rows=44700458 width=206) (actual time=29046.525..29217.937 rows=20 loops=1)
         Workers Planned: 2
         Workers Launched: 2
         ->  Sort  (cost=2332874.02..2388749.59 rows=22350229 width=206) (actual time=28998.140..28998.143 rows=20 loops=3)
               Sort Key: log_lines_30d.occurred_at DESC
               Sort Method: top-N heapsort  Memory: 34kB
               Worker 0:  Sort Method: top-N heapsort  Memory: 35kB
               Worker 1:  Sort Method: top-N heapsort  Memory: 35kB
                 ...
```

+ JIT Information

+ JSON format fixes

What's Missing

Aggregate information about Effectiveness of Parallel Query

Are my queries using parallel query?

Are there sufficient workers for parallel query?

Query Failures

Log Events

Canceling statement due to statement timeout

Canceling statement due to user request

New CONTEXT for failure of parameterized queries

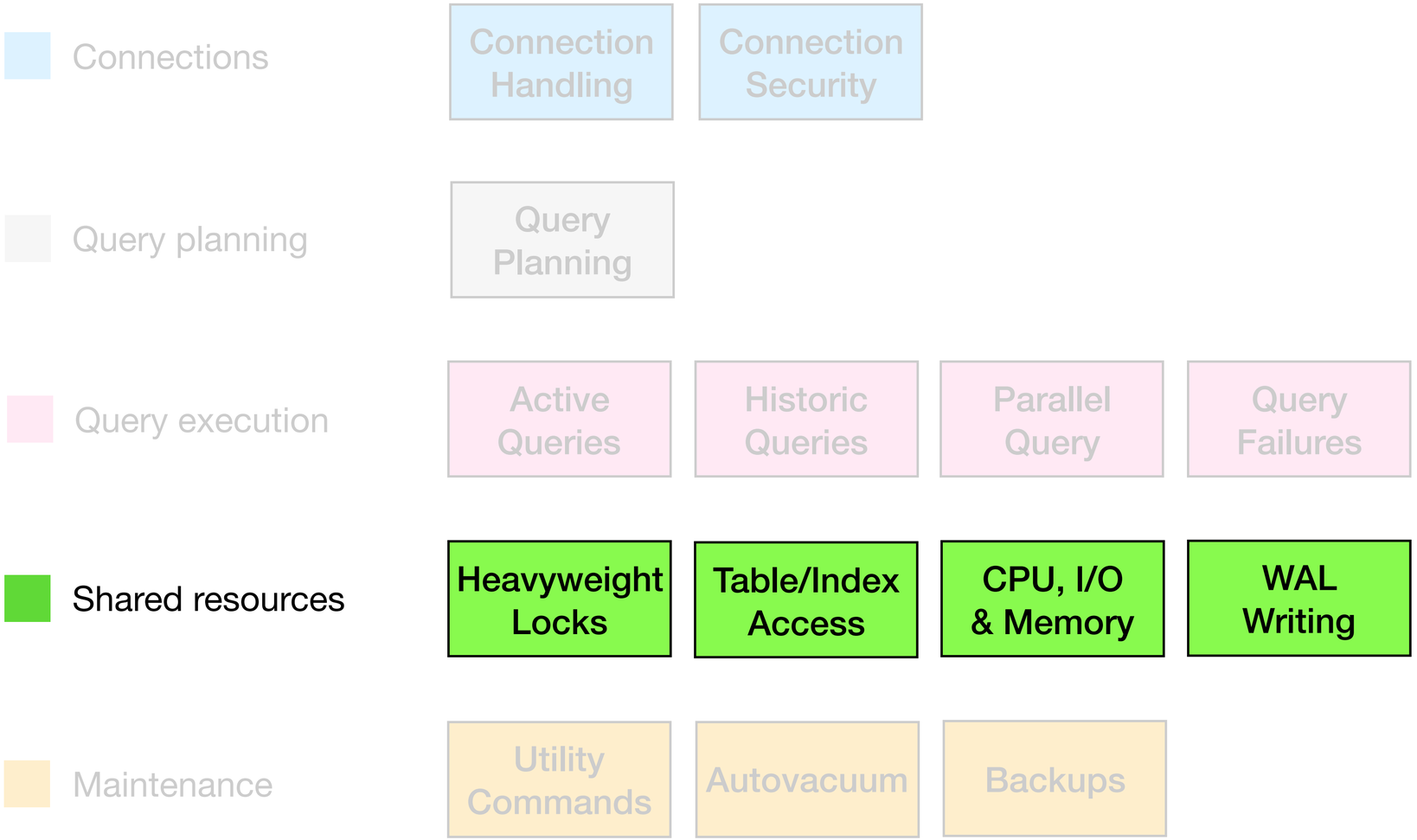
New in Postgres 13

CONTEXT for failure of parameterized queries

```
ERROR:  division by zero
STATEMENT:  SELECT 1/$1
```

```
SET log_parameter_max_length_on_error = 1024
```

```
ERROR:  division by zero
CONTEXT:  extended query with parameters: $1 = '0'
STATEMENT:  SELECT 1/$1
```


Connections
Connection Handling
Connection Security
Query planning
Query Planning
Query execution
Active Queries
Historic Queries
Parallel Query
Query Failures
Shared resources
Heavyweight Locks
Table/Index Access
CPU, I/O & Memory
WAL Writing
Maintenance
Utility Commands
Autovacuum
Backups

Heavyweight Locks

pg_locks

Log Events

Process acquired lock on tuple / relation / object

Process still waiting for lock on tuple / relation / object

Canceling statement due to lock timeout

Deadlock detected (transaction rolled back)

Process avoided deadlock by rearranging queue order

What's Missing

Aggregate Lock Statistics

Difficult to use pg_locks for
historic data

(e.g. pg_stat_statements lock_wait_time column)

Table/Index access

pg_stat_all_tables

pg_statio_all_tables

pg_stat_all_indexes

pg_statio_all_indexes

What's Missing

Per-statement index scan/seq scan counters

pg_stat_statements should have idx_scan and seq_scan counters

CPU, I/O and Memory

System metrics

pg_statio_*.

New pg_shmem_allocations

What's Missing

Connection memory usage statistics

New in Postgres 13

pg_shmem_allocations

```
=# SELECT * FROM pg_shmem_allocations  ORDER BY allocated_size DESC;
                name                |    off     |    size    | allocated_size
------------------------------------+------------+------------+----------------
 Buffer Blocks                      |   86739584 | 8589934592 |     8589934592
 <anonymous>                        |            |   91191424 |       91191424
 Buffer Descriptors                 |   19630720 |   67108864 |       67108864
 Buffer IO Locks                    | 8676674176 |   33554432 |       33554432
 Checkpointer Data                  | 8808573696 |   25165888 |       25165952
 Checkpoint BufferIds               | 8710228608 |   20971520 |       20971520
 XLOG Ctl                           |     104832 |   16803472 |       16803584
                                    | 8849116416 |    8033024 |        8033024
 Xact                               |   16908800 |    2116320 |        2116352
 Backend Activity Buffer            | 8807698304 |     541696 |         541696
 Subtrans                           |   19158912 |     267008 |         267008
 Backend Status Array               | 8807406080 |     224296 |         224384
```

<= shared_buffers

WAL Writing

pg_current_wal_lsn

New Per-statement WAL statistics

New autovacuum WAL statistics

New EXPLAIN WAL statistics

New in Postgres 13

Per-statement WAL statistics

```
=# SELECT substring(query for 70), wal_records, wal_fpi, wal_bytes FROM pg_stat_statements ORDER BY wal_records DESC;
                               substring                                | wal_records | wal_fpi |  wal_bytes
------------------------------------------------------------------------+-------------+---------+-------------
 COPY public.log_lines_30d_20200525 (log_line_id, server_id, log_line_p |    13637990 | 1690272 | 11920182549
 CREATE TEMPORARY TABLE upsert_data (server_id uuid NOT NULL, backend_i |     8568987 |    5429 |   881673525
 COPY activity.query_origins_7d_20200525 (backend_query_id, database_id |     7507811 |  789923 |  5796656931
 COPY activity.backend_snapshots_1d_20200525 (collected_at, state, wait |     6909068 |  802625 |  5241077274
 CREATE TEMPORARY TABLE upsert_data (server_id uuid NOT NULL, identity  |     6541995 |   11087 |   705700102
 CREATE TEMPORARY TABLE upsert_data (server_id uuid NOT NULL, identity  |     6418566 |   31123 |   771654086
 COPY public.log_line_stats_30d_20200525 (log_line_id, server_id, occur |     5338310 |  722396 |  4800621926
 DROP TABLE upsert_data                                                 |     5056385 |      13 |   269723603
 COPY public.log_lines_30d_20200524 (log_line_id, server_id, log_line_p |     3162220 |  496058 |  3261770520
 DROP TABLE upsert_data                                                 |     2134608 |       6 |   113901008
…
```

New in Postgres 13

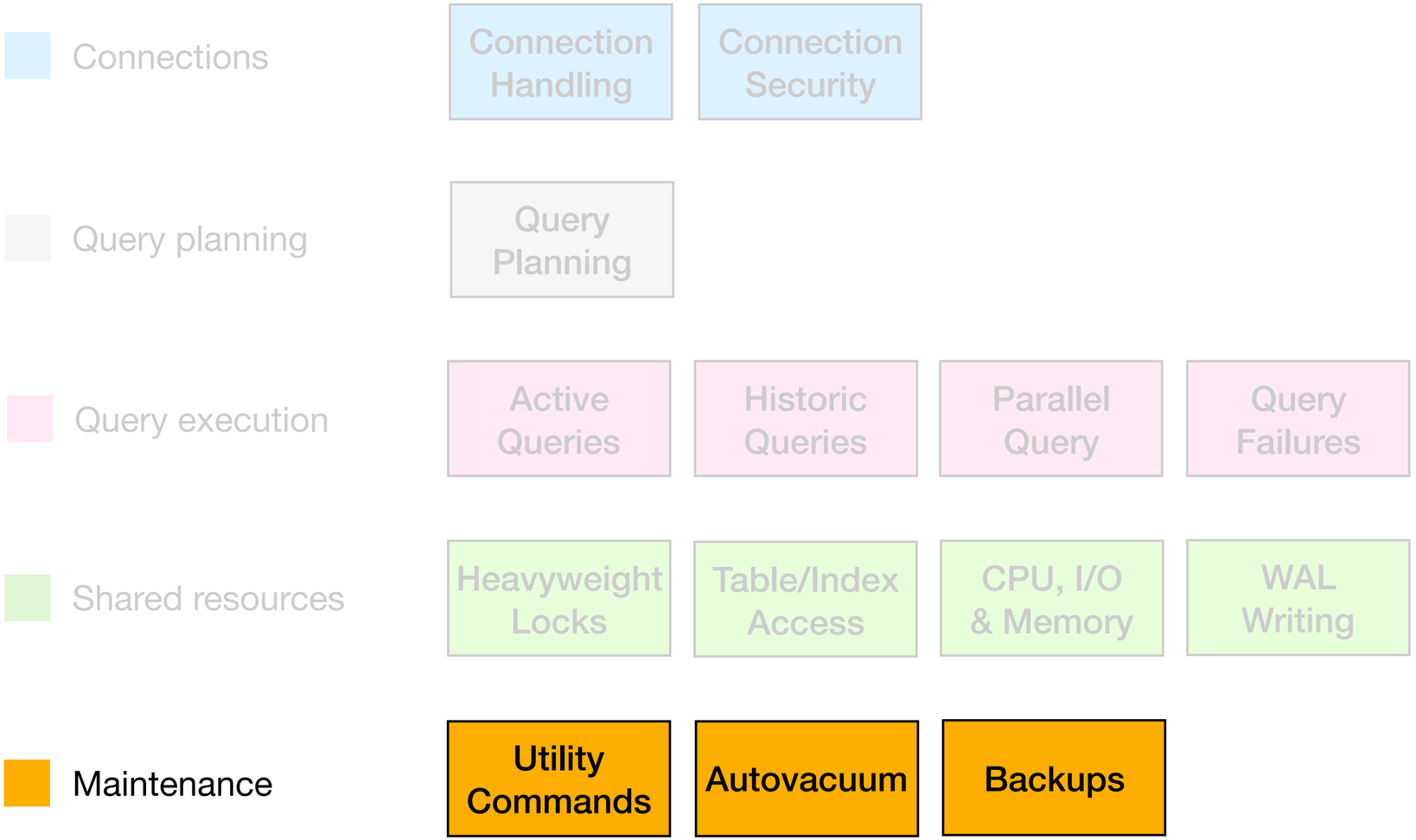
autovacuum WAL statistics

```
LOG:  automatic vacuum of table "…": index scans: 1
  pages: 0 removed, 75444 remain, 3 skipped due to pins, 0 skipped frozen
  tuples: 996760 removed, 4210912 remain, 0 are dead but not yet removable, oldest xmin: 1871789
  buffer usage: 114171 hits, 1 misses, 21614 dirtied
  avg read rate: 0.001 MB/s, avg write rate: 20.434 MB/s
  system usage: CPU: user: 2.42 s, system: 0.03 s, elapsed: 8.26 s
  WAL usage: 94064 records, 17930 full page images, 34394711 bytes
```

New in Postgres 13

EXPLAIN WAL statistics

```
=# BEGIN;
BEGIN
=*# EXPLAIN (ANALYZE, WAL) UPDATE backend_counts SET state = state WHERE backend_count_id IN (SELECT backend_count_id FROM backend_counts LIMIT 100);
                                                                            QUERY PLAN
------------------------------------------------------------------------------------------------------------------------------------------------------
 Update on backend_counts  (cost=4.47..850.04 rows=100 width=139) (actual time=1.049..1.049 rows=0 loops=1)
   WAL: records=168 fpi=5 bytes=39013
   ->  Nested Loop  (cost=4.47..850.04 rows=100 width=139) (actual time=0.239..0.734 rows=100 loops=1)
         WAL: records=2 bytes=416
         ->  HashAggregate  (cost=4.04..5.04 rows=100 width=56) (actual time=0.229..0.246 rows=100 loops=1)
               Group Key: "ANY_subquery".backend_count_id
               Peak Memory Usage: 45 kB
               ->  Subquery Scan on "ANY_subquery"  (cost=0.00..3.79 rows=100 width=56) (actual time=0.014..0.202 rows=100 loops=1)
                     ->  Limit  (cost=0.00..2.79 rows=100 width=16) (actual time=0.011..0.184 rows=100 loops=1)
                           ->  Seq Scan on backend_counts backend_counts_1  (cost=0.00..119801.53 rows=4291453 width=16) (actual time=0.010..0.177 rows=
         ->  Index Scan using backend_counts_pkey on backend_counts  (cost=0.43..8.45 rows=1 width=99) (actual time=0.005..0.005 rows=1 loops=100)
               Index Cond: (backend_count_id = "ANY_subquery".backend_count_id)
               WAL: records=2 bytes=416
```


Connections
Connection Handling
Connection Security
Query planning
Query Planning
Query execution
Active Queries
Historic Queries
Parallel Query
Query Failures
Shared resources
Heavyweight Locks
Table/Index Access
CPU, I/O & Memory
WAL Writing
Maintenance
Utility Commands
Autovacuum
Backups

Utility Commands

pg_stat_progress_vacuum

New pg_stat_progress_analyze

pg_stat_progress_cluster

pg_stat_progress_create_index

New in Postgres 13

pg_stat_progress_analyze

```
=# SELECT * FROM pg_stat_progress_analyze ;
  pid  | datid |      datname      | relic  |         phase          | sample_blks_total | sample_blks_scanned | ext_stats
-------+-------+-------------------+--------+------------------------+-------------------+---------------------+----------
 36936 | 16400 | pganalyze_staging | 115537 | acquiring sample rows |             30000 |               26756 |
(1 row)
```

autovacuum

pg_stat_progress_vacuum

Log Events

Canceling autovacuum task
Database must be vacuumed within N transactions (TXID Wraparound Warning)
Database is not accepting commands to avoid wraparound data loss
Autovacuum launcher started
Autovacuum launcher shutting down
Automatic vacuum of table completed
Skipping vacuum - lock not available

What's Missing

Aggregate autovacuum stats (only available in logs)

How often a table is being vacuumed
Avg runtime of a vacuum
Tuples that couldn't be removed

Backups

New pg_stat_progress_basebackup

New in Postgres 13

pg_stat_progress_basebackup

```
=# SELECT * FROM pg_stat_progress_basebackup ;
  pid  |              phase               | backup_total | backup_streamed | tablespaces_total | tablespaces_streamed
-------+----------------------------------+--------------+-----------------+-------------------+----------------------
 35397 | waiting for checkpoint to finish |            0 |               0 |                 0 |                    0
(1 row)
```

```
=# SELECT *, backup_streamed / backup_total::float * 100 AS  pct_done FROM pg_stat_progress_basebackup ;
  pid  |          phase           | backup_total | backup_streamed | tablespaces_total | tablespaces_streamed |     pct_done
-------+--------------------------+--------------+-----------------+-------------------+----------------------+------------------
 35397 | streaming database files |  63006018048 |     52671390720 |                 1 |                    0 | 83.59739648976586
(1 row)
```

What's Missing

1. Client-side connection latency
2. Aggregation of security-relevant Postgres events
3. Aggregate Plan Statistics
4. Breakdown of non-waiting active state
5. Query Progress Monitoring
6. pgss: Better handling of IN(…) lists & other ORM patterns
7. Linking pg_stat_statements with other views & logs
8. Finding queries based on application requests/customers
9. Built-in Wait Event Aggregation
10. Aggregate information about effectiveness of Parallel Query
11. Aggregate Lock Statistics
12. Per-statement index scan/seq scan counters
13. Connection memory usage statistics
14. Aggregate autovacuum stats

New in Postgres 13

1. EXPLAIN: Buffers for Planning
2. pg_stat_statements: Planning Time
3. Additional & renamed wait events
4. pg_stat_activity: leader_pid for Parallel Query
5. EXPLAIN improvements for parallel workers
6. CONTEXT for failure of parameterized queries
7. pg_shmem_allocations
8. Per-statement WAL statistics
9. autovacuum WAL statistics
10. EXPLAIN WAL statistics
11. pg_stat_progress_analyze
12. pg_stat_progress_basebackup

Thank you!

lukas@fittl.com

@LukasFittl