DB Memory Management

Shared (shared_buffers) & Private (work_mem)

Agenda

Intro & why I want to focus on work_mem today

How this typically works out fine (the good)

Some sub-optimal behavior (the bad)

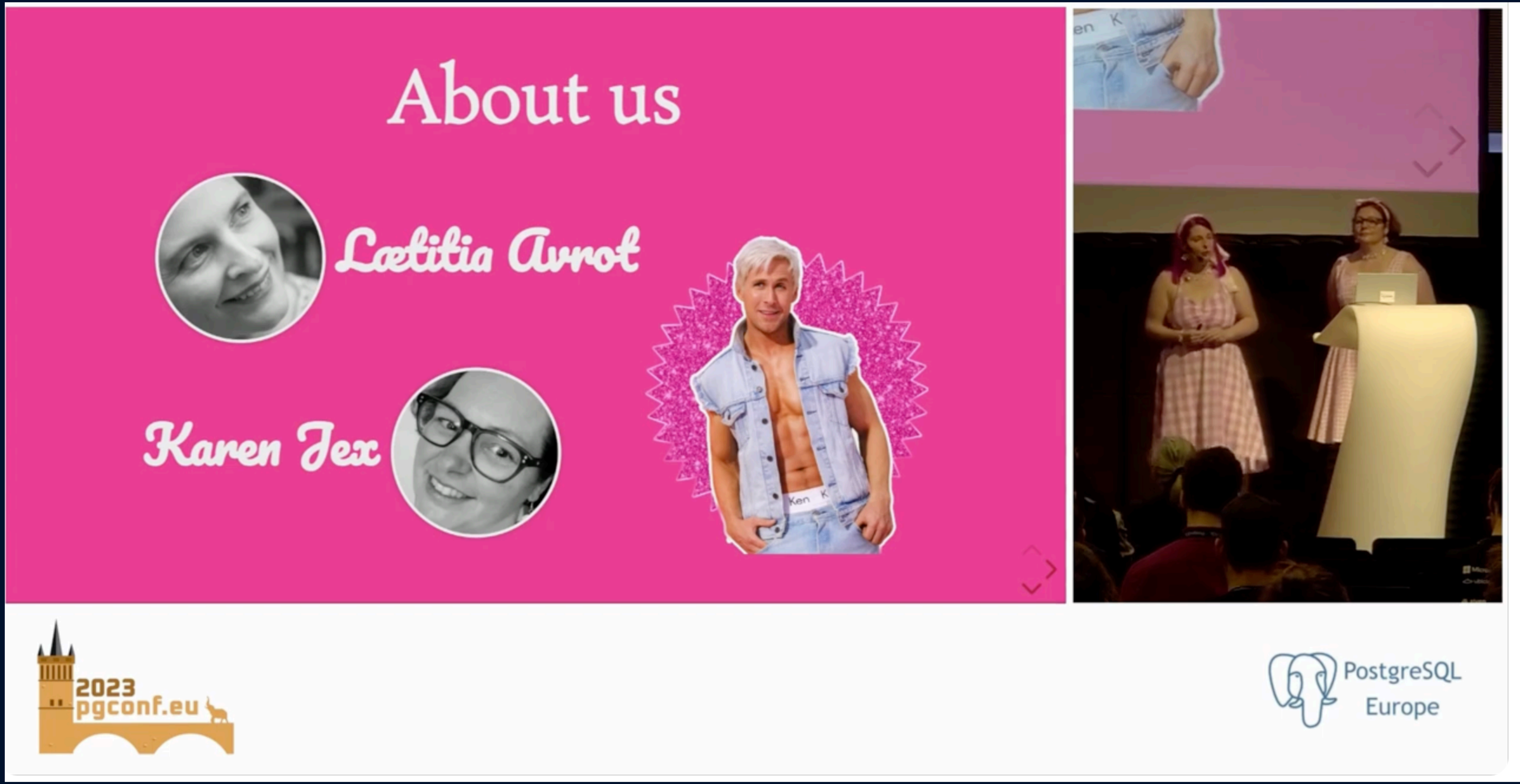
Ocassionally gets ugly (DBaaS and OOM Killer)

Conclusions

Who am I?

Dave Pitts - Database Engineer - Adyen

DBA Dream Jobs?



https://www.youtube.com/watch?v=nuqpL1LFCCE

Sizing PG to avoid OOM killer

If PostgreSQL itself is the cause of the system running out of memory … it may help to lower memory-related configuration parameters, particularly shared_buffers, work_mem, and hash_mem_multiplier.

In many cases, it may be better to reduce max_connections and instead make use of external connection-pooling software.

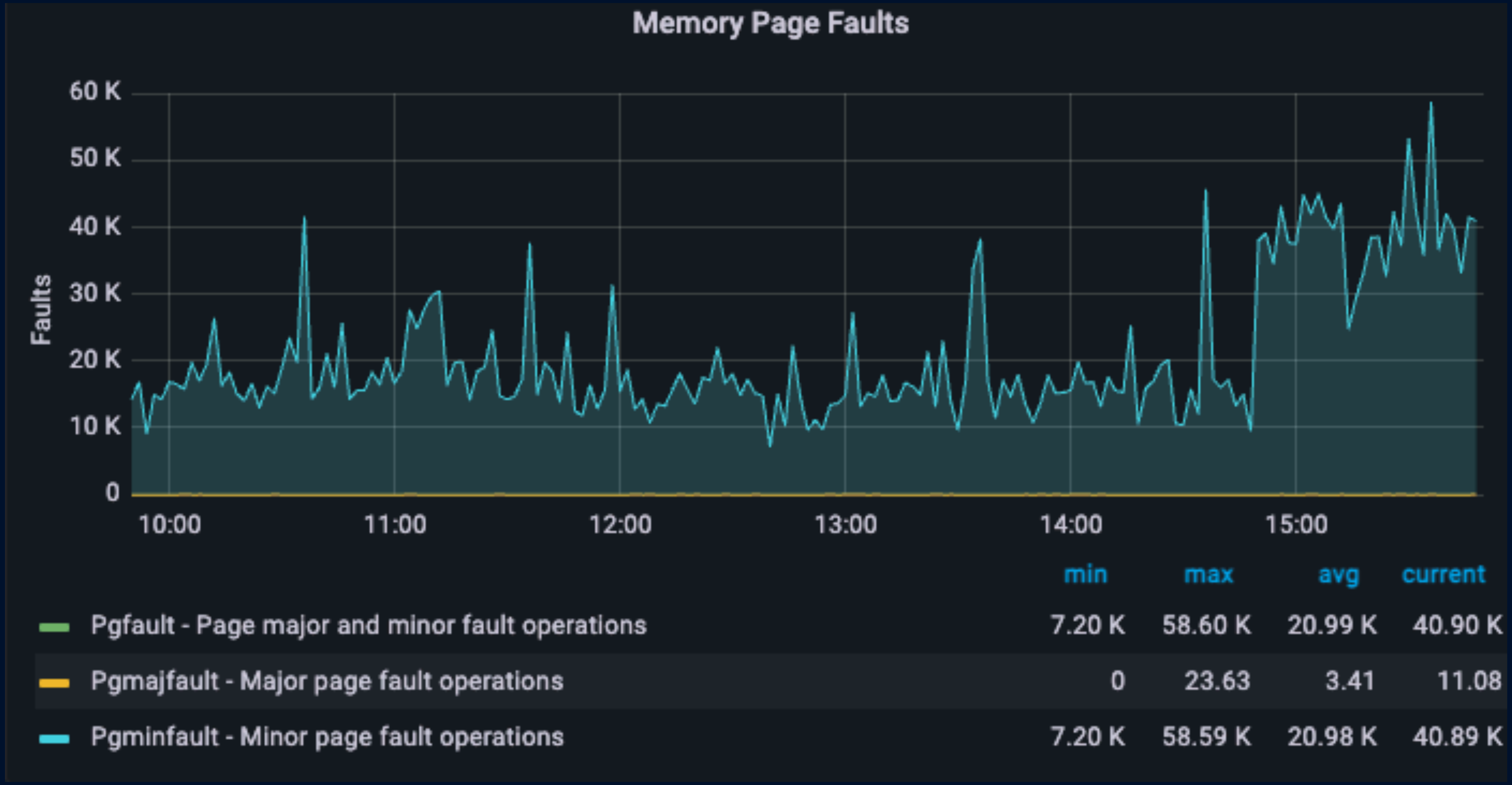
https://www.postgresql.org/docs/current/kernel-resources.html#LINUX-MEMORY-OVERCOMMIT

Show of hands

When the DB goes pop …

- Who’s seen OOM Killer (in production)
- Who’s seen DBaaS Failover (memory exhaustion?)

Any ideas - minor page faults?



DB Memory Management

Private Memory
- work_mem
- maintenance_work_mem

Shared Memory
- shared_buffers (don’t forget “double buffering”)
- other “smaller stuff“ (e.g. WAL buffers)

OS Memory

pg13 further complications

Private
- work_mem
- maintenance_work_mem
- hash_mem_multiplier (pg13+)

Shared
- shared_buffers (don't forget “double buffering”)
- other “smaller stuff“ (e.g. WAL buffers)

OS Memory

PostgreSQL worst practices

Apparently running with default shared_buffers and work_mem is surprising common

https://www.postgresql.eu/events/
pgdayparis2024/schedule/session/5333-postgresql-worst-practices/

A good starting point

For example, Christophe Pettus suggests that 16MB is a good starting point for most people.

Apparently running with default shared_buffers and work_mem is surprising common.

https://www.pgmustard.com/blog/work-mem

https://www.youtube.com/watch?v=XUkTUMZRBE8&t=304s

work_mem an after-taught?

“It's December 29th, so it's holiday season. And let's have some small episode about work_mem.”

postgres.fm/episodes/work_mem

When work_mem matters

Two common use cases :

- Large **Sort** Operations (relatively easy to predict)

- Large **Hashing** Operations (maybe hard to predict)

Sizing work_mem gotcha pg15+?

Multipler for just Hash Operations :

- Default work_mem=4096Kb

- pg13 introduced hash_mem_multiplier default of 1

- Default hash_mem_multiplier value of 2 in pg15+

- Double memory for hash ops only (8192Kb?)

increase hash_mem_multiplier thread

- PG - Peter Geoghegan (pg at bowt.ie)
- JN - John Naylor (john.naylor at postgresql.org)
- Both major Postgres contributors

https://www.postgresql.org/community/contributors/

open discussion

- PG - default is 1.0, which is a fairly conservative default: it preserves the historic behavior, which is that hash-based executor nodes receive the same work_mem budget as sort-based nodes …
- JN - on a couple occasions recommend clients to raise hash_mem_multiplier to 2.0 to fix performance problems

sorts should not affected?

- PG - sort-based nodes have very predictable performance characteristics, and the possible upside of allowing a sort node to use more memory is quite bounded
- JN During this cycle, we also got a small speedup in the external sorting code

We will review sorts again later…

pg15+ default x2

- The default hash_mem_multiplier value is now 2.0 from pg15+ (it was 1.0 in pg13 and pg14)

- Sounds like a potentially breaking change to me. Are there any edge cases around high hash operations workloads?

- Let's review some execution plans and run some custom pgbench tests

Citus Simple OLAP Cube

OLAP Table with 100 different columns (all integer)

```
# CREATE TABLE perf_row(
 c00 int8, c01 int8, c02 int8, c03 int8, c04 int8, c05 int8, c06 int8, c07 int8, c08 int8, c09 int8,
 c10 int8, c11 int8, c12 int8, c13 int8, c14 int8, c15 int8, c16 int8, c17 int8, c18 int8, c19 int8,
 c20 int8, c21 int8, c22 int8, c23 int8, c24 int8, c25 int8, c26 int8, c27 int8, c28 int8, c29 int8,
 c30 int8, c31 int8, c32 int8, c33 int8, c34 int8, c35 int8, c36 int8, c37 int8, c38 int8, c39 int8,
 c40 int8, c41 int8, c42 int8, c43 int8, c44 int8, c45 int8, c46 int8, c47 int8, c48 int8, c49 int8,
 c50 int8, c51 int8, c52 int8, c53 int8, c54 int8, c55 int8, c56 int8, c57 int8, c58 int8, c59 int8,
 c60 int8, c61 int8, c62 int8, c63 int8, c64 int8, c65 int8, c66 int8, c67 int8, c68 int8, c69 int8,
 c70 int8, c71 int8, c72 int8, c73 int8, c74 int8, c75 int8, c76 int8, c77 int8, c78 int8, c79 int8,
 c80 int8, c81 int8, c82 int8, c83 int8, c84 int8, c85 int8, c86 int8, c87 int8, c88 int8, c89 int8,
 c90 int8, c91 int8, c92 int8, c93 int8, c94 int8, c95 int8, c96 int8, c97 int8, c98 int8, c99 int8
);
```

https://github.com/dgapitts/pgday-munich-work_mem (Demo 01)

Citus Simple OLAP Cube

Distrinct values C00:500, C70:35500, C99:50000

```
# INSERT INTO perf_row
  SELECT
    g % 00500, g % 01000, g % 01500, g % 02000, g % 02500, g % 03000, g % 03500, g % 04000, g % 04500, g % 05000,
    g % 05500, g % 06000, g % 06500, g % 07000, g % 07500, g % 08000, g % 08500, g % 09000, g % 09500, g % 10000,
    g % 10500, g % 11000, g % 11500, g % 12000, g % 12500, g % 13000, g % 13500, g % 14000, g % 14500, g % 15000,
    g % 15500, g % 16000, g % 16500, g % 17000, g % 17500, g % 18000, g % 18500, g % 19000, g % 19500, g % 20000,
    g % 20500, g % 21000, g % 21500, g % 22000, g % 22500, g % 23000, g % 23500, g % 24000, g % 24500, g % 25000,
    g % 25500, g % 26000, g % 26500, g % 27000, g % 27500, g % 28000, g % 28500, g % 29000, g % 29500, g % 30000,
    g % 30500, g % 31000, g % 31500, g % 32000, g % 32500, g % 33000, g % 33500, g % 34000, g % 34500, g % 35000,
    g % 35500, g % 36000, g % 36500, g % 37000, g % 37500, g % 38000, g % 38500, g % 39000, g % 39500, g % 40000,
    g % 40500, g % 41000, g % 41500, g % 42000, g % 42500, g % 43000, g % 43500, g % 44000, g % 44500, g % 45000,
    g % 45500, g % 46000, g % 46500, g % 47000, g % 47500, g % 48000, g % 48500, g % 49000, g % 49500, g % 50000
  FROM generate_series(1,500000) g;
```

https://github.com/dgapitts/pgday-munich-work_mem (Demo 01)

Before pg15 - hash_mem_multiplier=1

Aggr/Group 50K values: very high Disc Usage and high IOPs

```
# EXPLAIN (ANALYZE, BUFFERS) SELECT c99, SUM(c29), AVG(c71) FROM perf_row GROUP BY c99;
                                        QUERY PLAN
------------------------------------------------------------------------------------------------------------
 HashAggregate  (cost=97743.84..104357.10 rows=50256 width=72) (actual time=1697.361..2050.372 rows=50000 loops=1)
   Group Key: c99
   Planned Partitions: 4  Batches: 5  Memory Usage: 4145kB  Disk Usage: 23496kB
   Buffers: shared hit=15688 read=39868, temp read=2641 written=4909
   ->  Seq Scan on perf_row  (cost=0.00..60556.04 rows=500004 width=24) (actual time=2.007..833.247 rows=500000 loops=1)
         Buffers: shared hit=15688 read=39868
```

https://github.com/dgapitts/pgday-munich-work_mem (Demo 01)

pg15+ hash_mem_multiplier=2

Aggr/Group 50K values: high Disc Usage and high IOPs

```
# EXPLAIN (ANALYZE, BUFFERS) SELECT c99, SUM(c29), AVG(c71) FROM perf_row GROUP BY c99;
                                          QUERY PLAN
--------------------------------------------------------------------------------------------------------------
 HashAggregate  (cost=97743.84..104357.10 rows=50256 width=72) (actual time=1480.648..1689.421 rows=50000 loops=1)
   Group Key: c99
   Planned Partitions: 4  Batches: 5  Memory Usage: 8241kB  Disk Usage: 14104kB
   Buffers: shared hit=15688 read=39868, temp read=1525 written=2896
   ->  Seq Scan on perf_row  (cost=0.00..60556.04 rows=500004 width=24) (actual time=0.700..825.849 rows=500000 loops=1)
         Buffers: shared hit=15688 read=39868
```

https://github.com/dgapitts/pgday-munich-work_mem (Demo 01)

Custom hash_mem_multiplier=4

Aggr/Group 50K values - with custom setting (pg13+)

```
# set hash_mem_multiplier=4;
SET
# EXPLAIN (ANALYZE, BUFFERS) SELECT c99, SUM(c29), AVG(c71) FROM perf_row GROUP BY c99;
                                            QUERY PLAN
------------------------------------------------------------------------------------------------------------
 HashAggregate  (cost=64306.07..65059.91 rows=50256 width=72) (actual time=247.716..257.823 rows=50000 loops=1)
   Group Key: c99
   Batches: 1  Memory Usage: 12561kB
   Buffers: shared hit=15822 read=39734
   ->  Seq Scan on perf_row  (cost=0.00..60556.04 rows=500004 width=24) (actual time=0.226..88.872 rows=500000 loops=1)
         Buffers: shared hit=15822 read=39734
```

https://github.com/dgapitts/pgday-munich-work_mem (Demo 01)

Why hash_mem_multiplier?

Why not just run with bigger work_mem?

How much do we need reduce average work_mem to accommodate higher hash_mem_multiplier?

Wasn't sizing Postgres Private Memory already hard enough?

Checking Sort Behavior

hash_mem_multiplier 4 > temp (read=4963 written=4973)

```
# EXPLAIN (ANALYZE, BUFFERS) SELECT * FROM perf_row where c00 < 50 order by c50;

 Gather Merge  (cost=67677.98..72370.17 rows=40216 width=800) (actual time=481.396..519.861 rows=50000 loops=1)
   Workers Planned: 2
   Workers Launched: 2
   Buffers: shared hit=15654 read=40016, temp read=4963 written=4973
   ->  Sort  (cost=66677.96..66728.23 rows=20108 width=800) (actual time=402.314..409.534 rows=16667 loops=3)
         Sort Key: c50
         Sort Method: external merge  Disk: 16552kB
         Buffers: shared hit=15654 read=40016, temp read=4963 written=4973
         Worker 0:  Sort Method: external merge  Disk: 11584kB
         Worker 1:  Sort Method: external merge  Disk: 11568kB
         ->  Parallel Seq Scan on perf_row  (cost=0.00..58160.19 rows=20108 width=800) (actual time=4.001..242.583 rows=16667 loops=3)
               Filter: (c00 < 50)
               Rows Removed by Filter: 150000
               Buffers: shared hit=15540 read=40016
```

https://github.com/dgapitts/pgday-munich-work_mem (Demo 02)

no change (as expected)

hash_mem_multiplier 2 > temp (read=4962 written=4970)

```
# EXPLAIN (ANALYZE, BUFFERS) SELECT * FROM perf_row where c00 < 50 order by c50;

 Gather Merge  (cost=67677.98..72370.17 rows=40216 width=800) (actual time=359.359..397.040 rows=50000 loops=1)
   Workers Planned: 2
   Workers Launched: 2
   Buffers: shared hit=15667 read=40003, temp read=4962 written=4970
   ->  Sort  (cost=66677.96..66728.23 rows=20108 width=800) (actual time=338.722..343.132 rows=16667 loops=3)
         Sort Key: c50
         Sort Method: external merge  Disk: 14360kB
         Buffers: shared hit=15667 read=40003, temp read=4962 written=4970
         Worker 0:  Sort Method: external merge  Disk: 10016kB
         Worker 1:  Sort Method: external merge  Disk: 15320kB
         ->  Parallel Seq Scan on perf_row  (cost=0.00..58160.19 rows=20108 width=800) (actual time=9.280..214.408 rows=16667 loops=3)
               Filter: (c00 < 50)
               Rows Removed by Filter: 150000
               Buffers: shared hit=15553 read=40003
```

https://github.com/dgapitts/pgday-munich-work_mem (Demo 02)

ORA-PG migration headache

Oracle uses a simpler bucket approach - at least from a DBA perspective

A bucket for total private memory e.g. 4000Mb on 16G VM

Typically queries use only 1Mb or 2Mb (introspection is also easier)

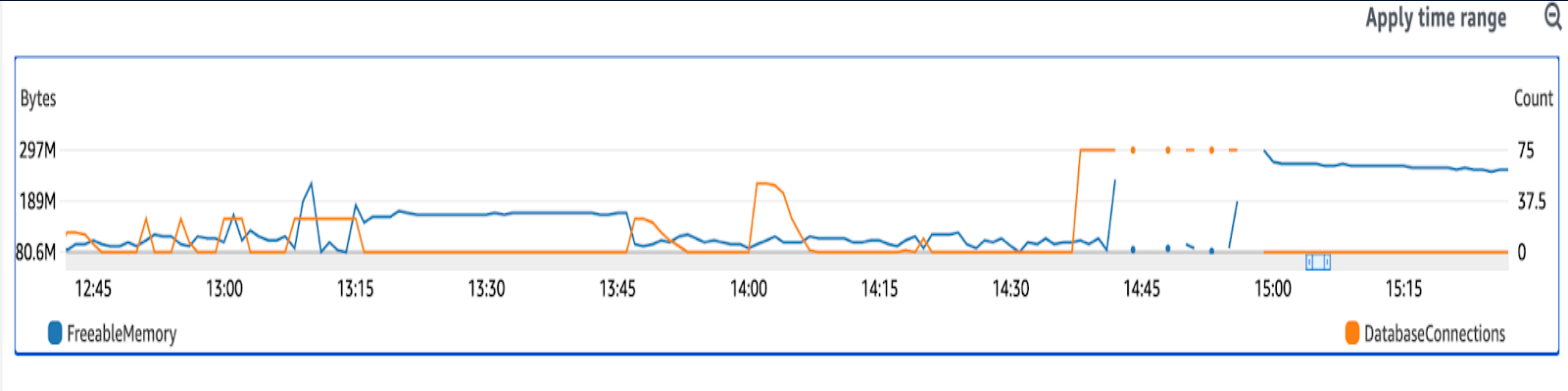
Exceptional processes with complex queries able to grow up 10% e.g. 400Mb

pgbench & hash_mem_multiplier

- testing high hash_mem_multiplier (4) with 10, 15, 25, 50 & 75 concurrent processes (DBConnections)
- monitoring via AWS RDS server FreeableMemory

https://github.com/dgapitts/pgday-munich-work_mem (Demo 03)

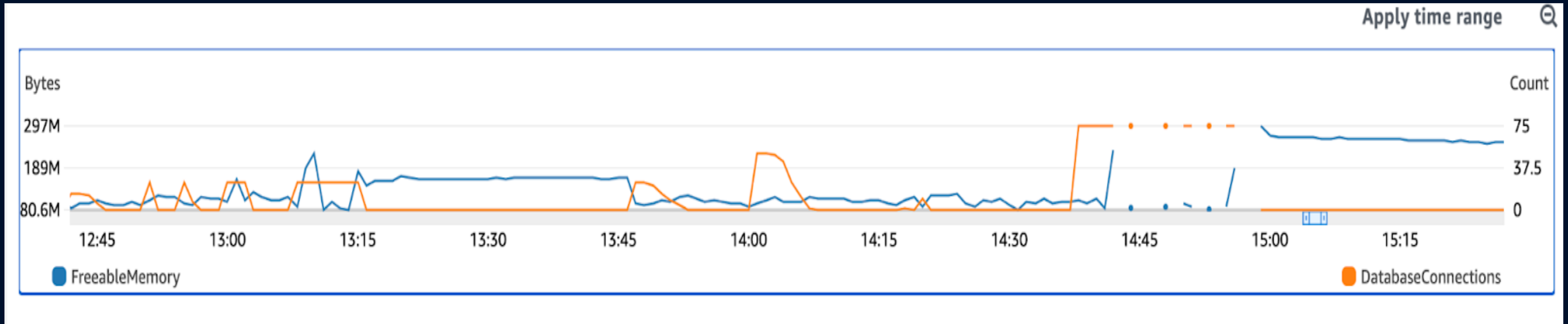
DBConn sizes 10, 15, 25, 50 & 70

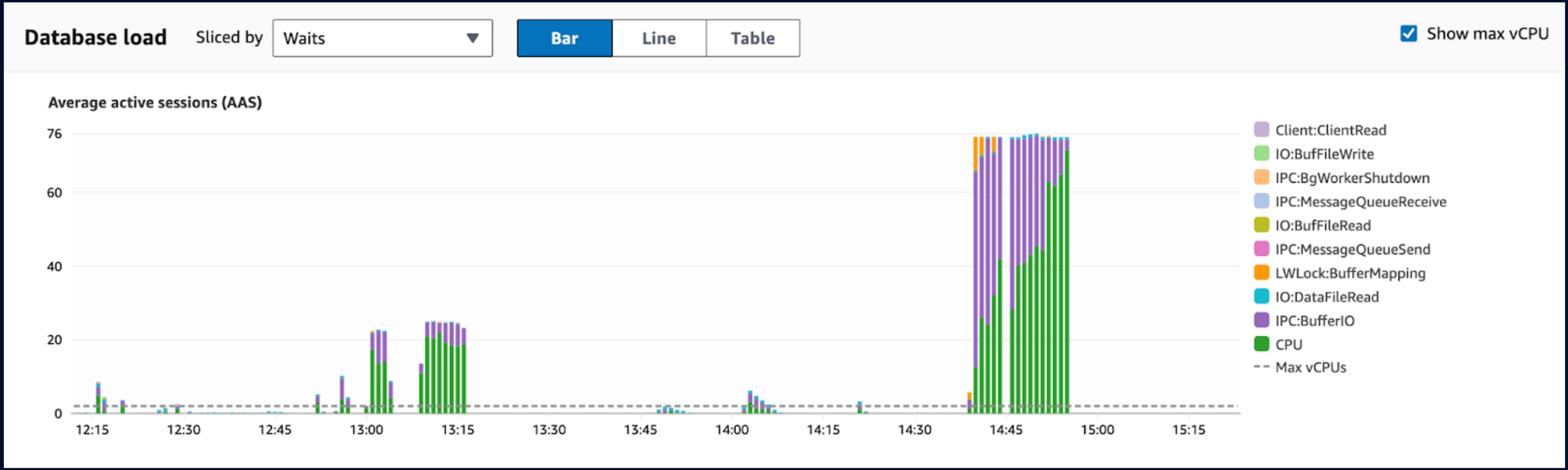


- FreeableMemory - 80M to 297M left-side
- DatabaseConnections - 0 to 75 right-side
- no correlation which is good

https://github.com/dgapitts/pgday-munich-work_mem (Demo 03)

Tests eventually hit CPU limits





pgbench & hash_mem_multiplier

- Expensive hash operations over 75 concurrent connections: minimal affect on Overall Memory usage

- Conclusion - individual execution plans clearly show higher memory usage but this is short lived/duration

https://github.com/dgapitts/pgday-munich-work_mem (Demo 03)

Expert Systems and AI?

- PGTune (simple webpage/expert system)
- DBtune (complex AI/ML model)

PGTune

- Simple heuristic model but a good starting point
- Definitely better than pg defaults (laptop only)

PGTune & hash_mem_multiplier

PG14 - PGTune - Inputs

```
# DB Version: 14
# OS Type: linux
# DB Type: oltp
# Total Memory (RAM): 16 GB
# CPUs num: 4
# Connections num: 500
# Data Storage: ssd
```

PG15 - PGTune - Inputs

```
# DB Version: 15
# OS Type: linux
# DB Type: oltp
# Total Memory (RAM): 16 GB
# CPUs num: 4
# Connections num: 500
# Data Storage: ssd
```

PG14 - PGTune

```
max_connections = 500
shared_buffers = 4GB
effective_cache_size = 12GB
maintenance_work_mem = 1GB
checkpoint_completion_target = 0.9
wal_buffers = 16MB
default_statistics_target = 100
random_page_cost = 1.1
effective_io_concurrency = 200
work_mem = 4194kB
huge_pages = off
min_wal_size = 2GB
max_wal_size = 8GB
max_worker_processes = 4
max_parallel_workers_per_gather = 2
max_parallel_workers = 4
max_parallel_maintenance_workers = 2
```

PG15 - PGTune - Output (identical)

```
max_connections = 500
shared_buffers = 4GB
effective_cache_size = 12GB
maintenance_work_mem = 1GB
checkpoint_completion_target = 0.9
wal_buffers = 16MB
default_statistics_target = 100
random_page_cost = 1.1
effective_io_concurrency = 200
work_mem = 4194kB
huge_pages = off
min_wal_size = 2GB
max_wal_size = 8GB
max_worker_processes = 4
max_parallel_workers_per_gather = 2
max_parallel_workers = 4
max_parallel_maintenance_workers = 2
```

PGTune: CPUs & work_mem

PG15 with 4 CPUs - PGTune (OLTP)

```
# CPUs num: 4
...
work_mem = 4194kB
max_worker_processes = 4
max_parallel_workers_per_gather = 2
max_parallel_workers = 4
max_parallel_maintenance_workers = 2
```

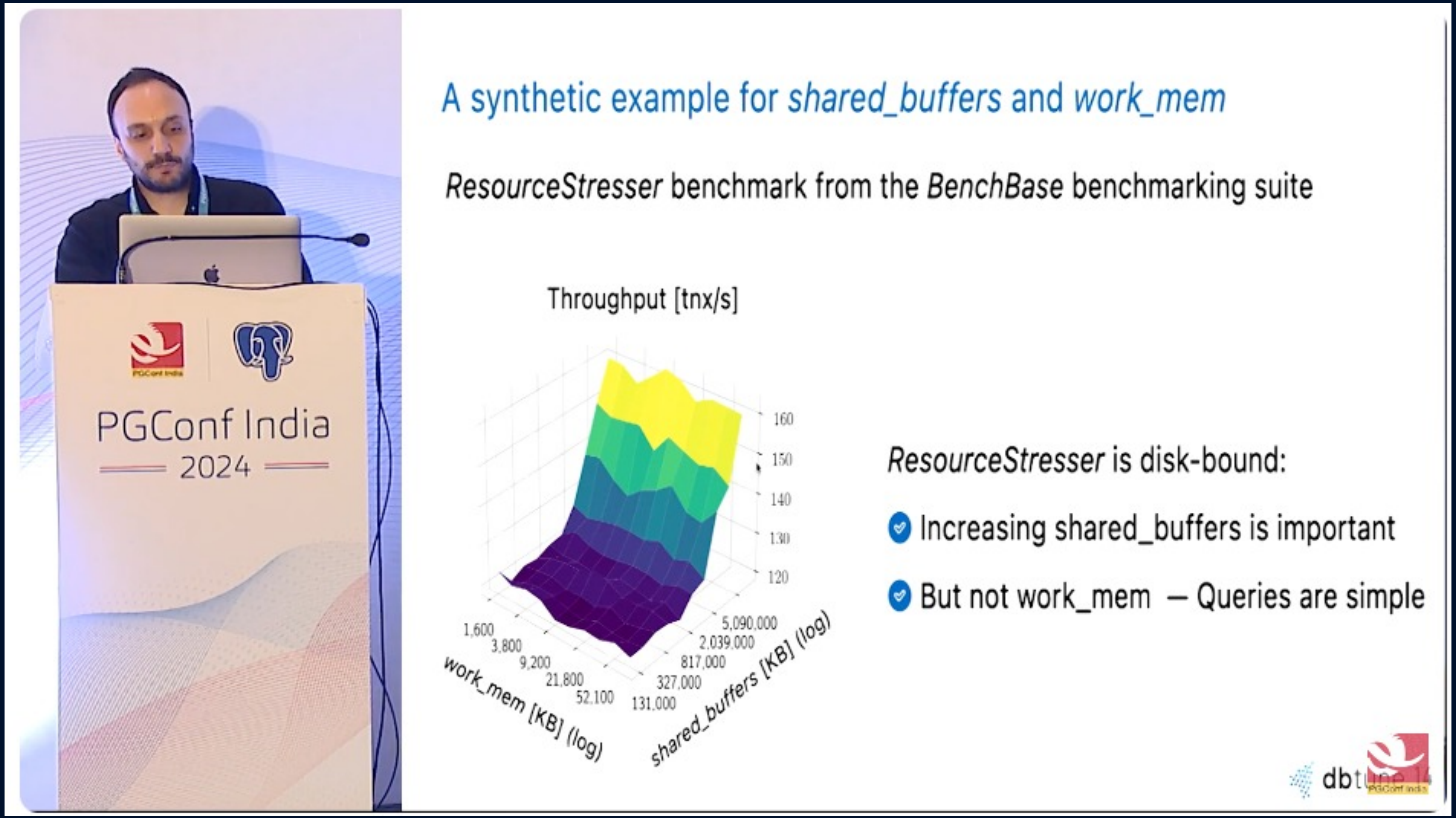
PG15 with 4 CPUs - PGTune (OLTP)

```
# CPUs num: 8
...
work_mem = 2097kB
max_worker_processes = 8
max_parallel_workers_per_gather = 4
max_parallel_workers = 8
max_parallel_maintenance_workers = 4
```

DBtune

- Interesting AI/ML approach
- Some impressive results (using BenchBase)
- Lots of DB restarts (optional?)
- oh yes and they are a sponsor for pgDay.DE 2024!

Tuning work_mem is workload specific!



https://www.youtube.com/watch?v=qNoiyqHdZlo

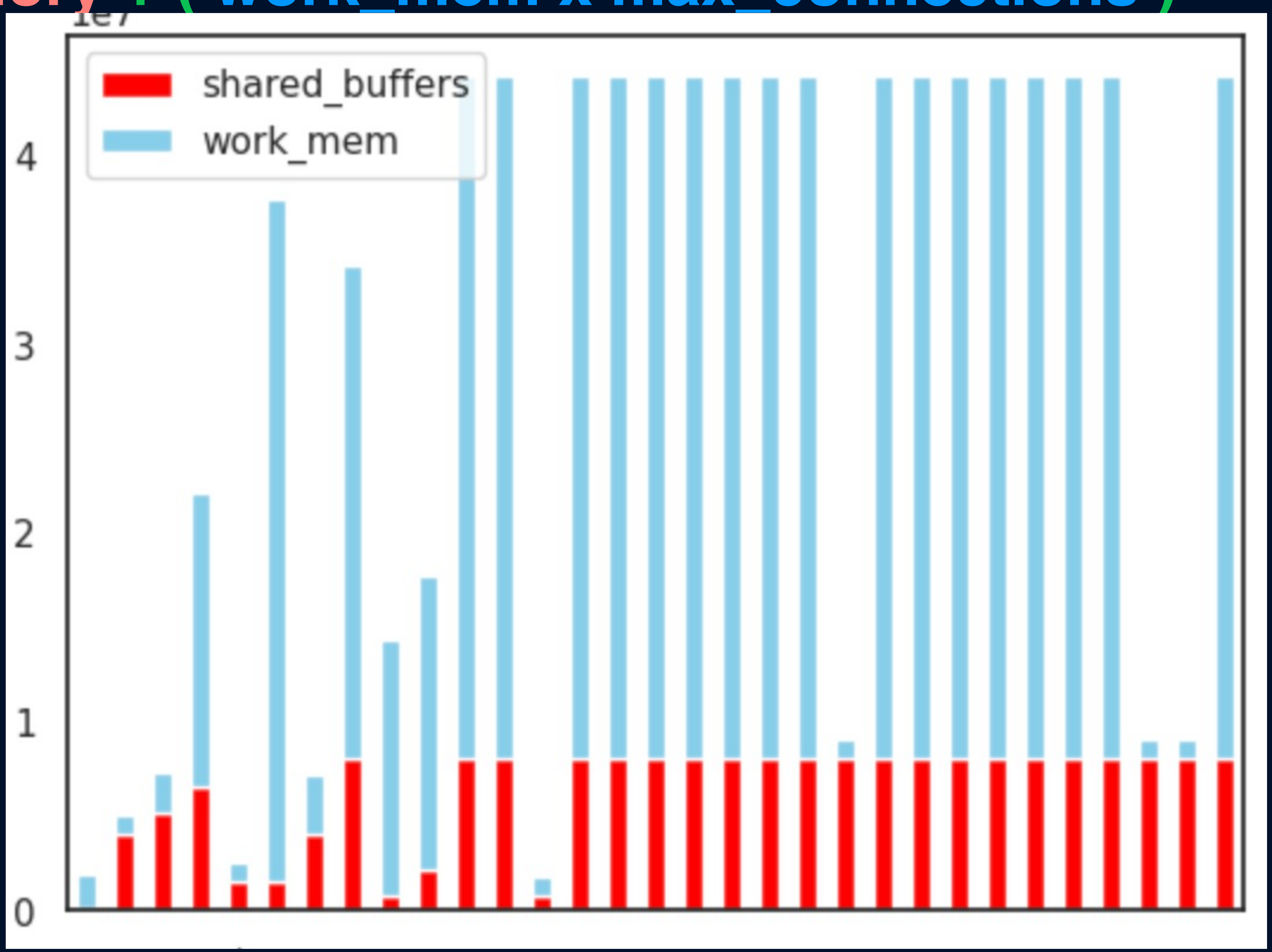
DBtune & BenchBase 30x5mins

shared_buffers,work_mem
131072,4096
4017552,2231
5142464,4859
6588784,34598
1526672,2231
1526672,80351
4017552,7146
8035096,58237
803512,30023
2249824,34598
8035096,80351
8035096,80351
803512,2231
8035096,80351
8035096,80351
8035096,80351

8035096,80351
8035096,80351
8035096,80351
8035096,80351
8035096,2231
8035096,80351
8035096,80351
8035096,80351
8035096,80351
8035096,80351
8035096,80351
8035096,80351
8035096,2231
8035096,2231
8035096,80351

Visualising Memory

Shared Memory + (work_mem x max_connections)



https://github.com/dgapitts/pgday-munich-work_mem (Demo 04)

Tuning work_mem is workload specific!

- OLTP - short atomic queries (low work_mem)
- OLAP - reports (high work_mem & fewer connections)
- HTAP - mixed (? work_mem & ? connections)

My Observations

- DBtune with restarts (default) seems like a natural fit for tools like pgReplay

- Could we seed solutions a bit more like a Genetic Algorithm? I asked Luigi …

- ML approach is more complex and exciting (faster / few iterations)

- Luigi “non-linear and non-obvious relationship between the different parameters”

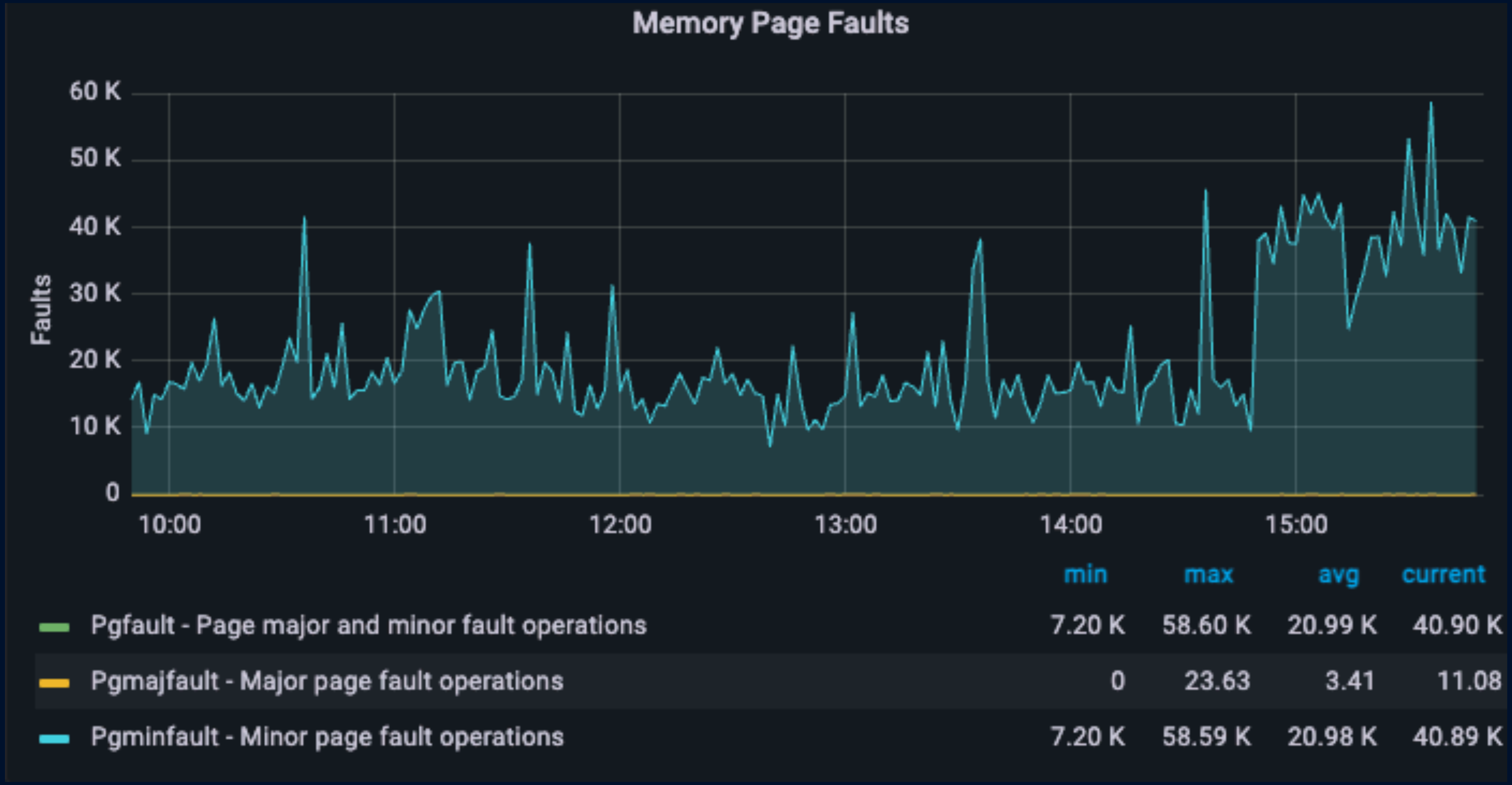
and pause before ...

The Ugly

- Hard to reproduce edge cases
- DBaaS appears more difficult/vulnerable
- Your not 100% safe on bare metal

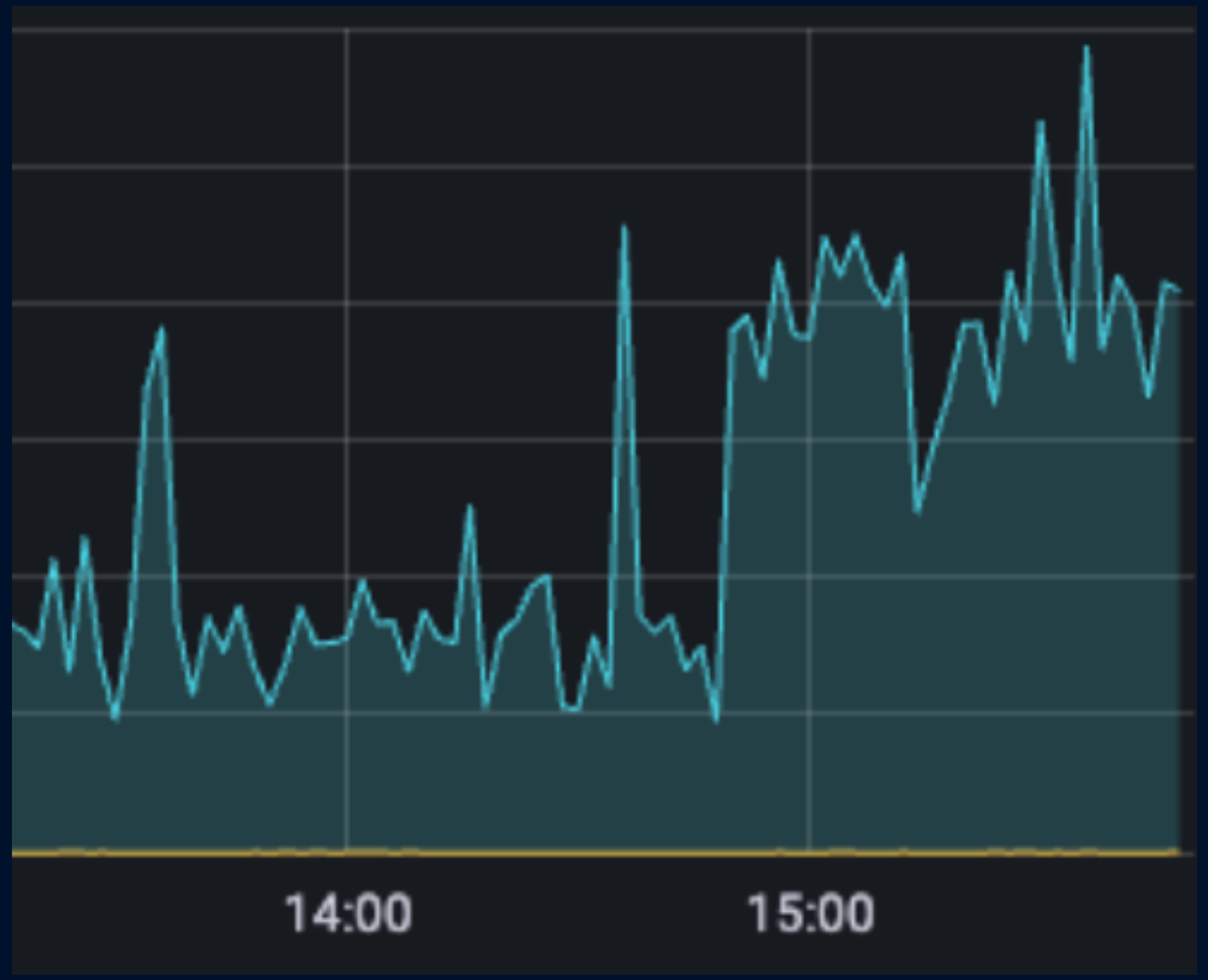
https://github.com/dgapitts/pgday-munich-work_mem (Demo 05)

Remember these page faults?



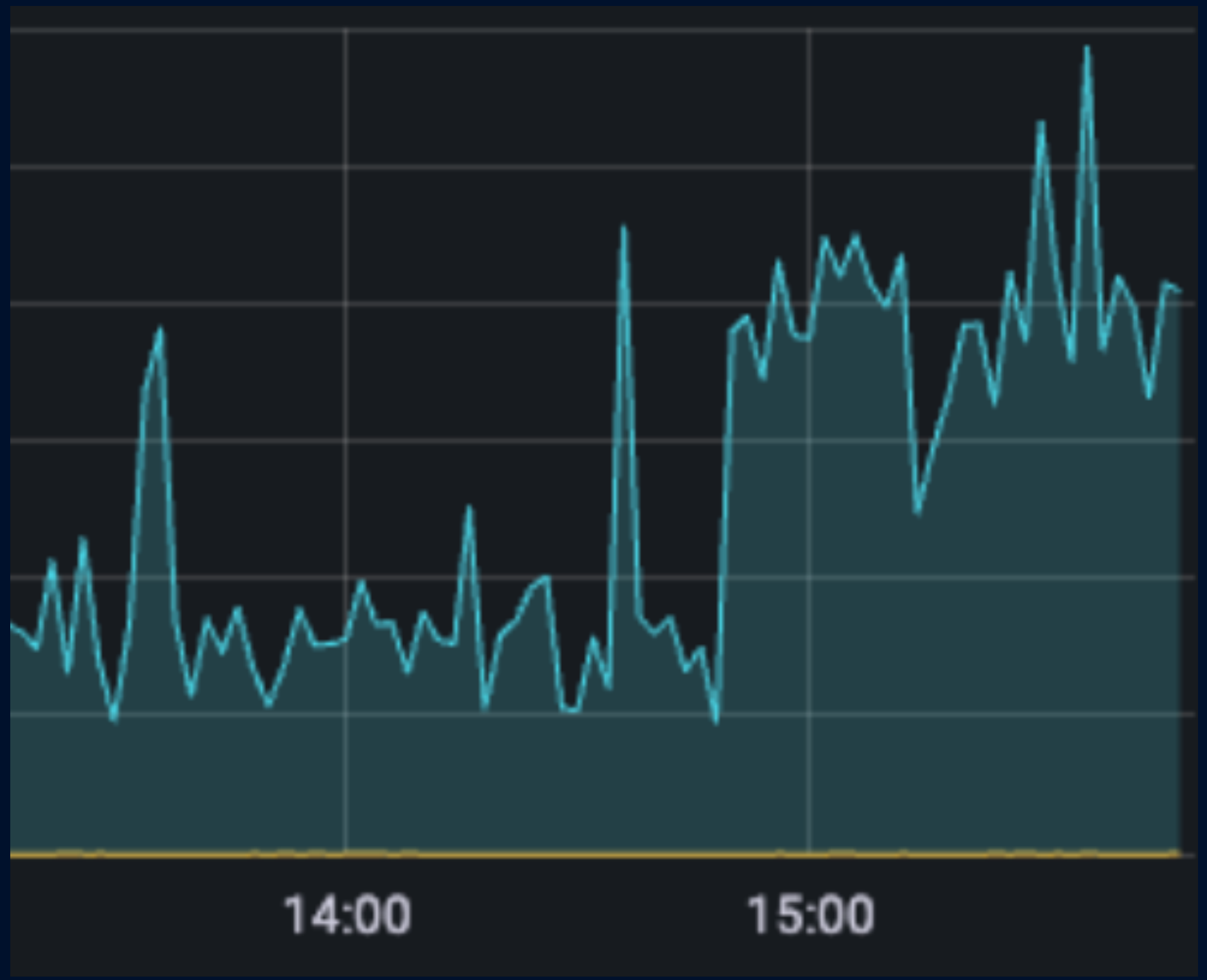
First Vulnerability

- Dynamic SQL is everywhere
- Loop / edge cases bugs
- PG 1K or 10K or 100K bind variable



Page Fault Incidents

- One dominant query (30%) with very high executions and buffer reads per exec (but no change)
- Multiple other queries (5%)
- New query extreme bind variables



Second Vulnerability

- Extreme SQL Statement
- No (apparent) limit of SQL size
- Invisible failovers (SAAS)

https://github.com/dgapitts/pgday-munich-work_mem (Demo 05)

Conclusion

16MB is a good starting point

Avoid high max_connections if possible use connection pooling layer (pg_bouncer)

For HTAP set work_mem at session level or custom conn pool

Sanitise your queries - Postgres does not do this form you.

PG Usergroups BCN & MAD

Andrea Cucciniello (Barcellona Lead)

Dave Pitts (Madrid Lead)

https://www.linkedin.com/company/102283719/admin/feed/posts/

Q & A

https://github.com/dgapitts/pgday-munich-work_mem

https://www.linkedin.com/company/102283719/
BCN & MAD Postgres Usergroups