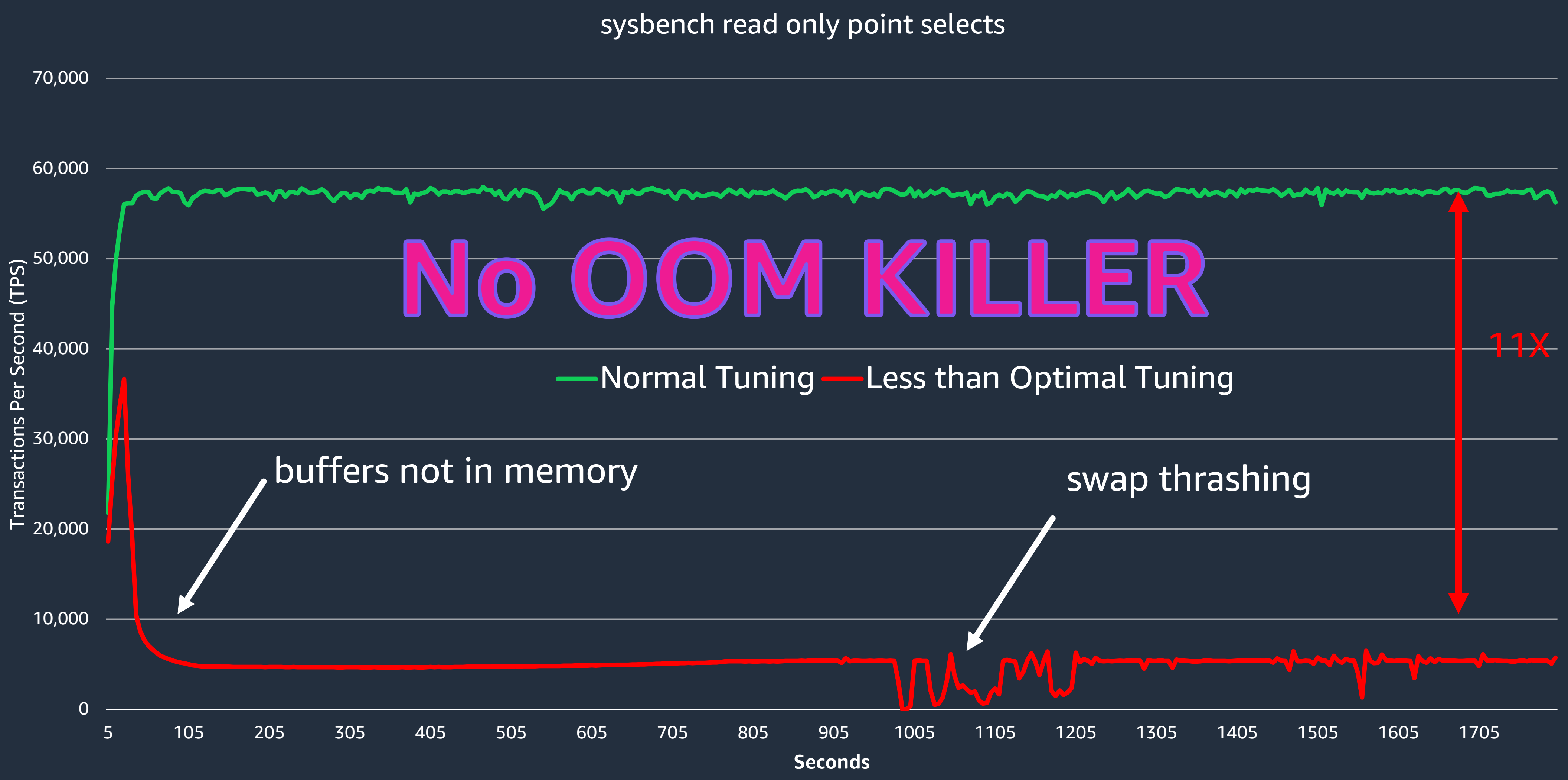
PRACTICAL MEMORY TUNING FOR POSTGRESQL

Grant McAlister

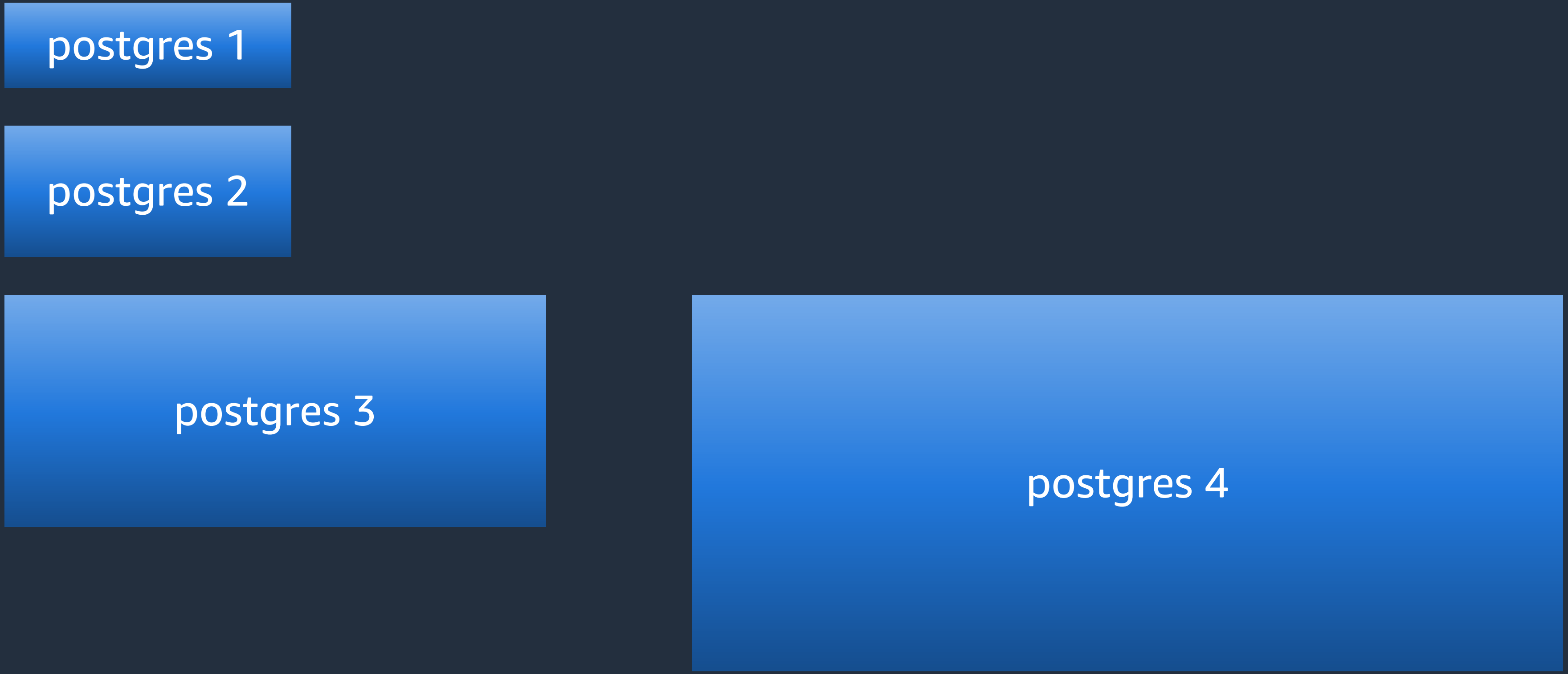
Senior Principal Engineer

Why you should care



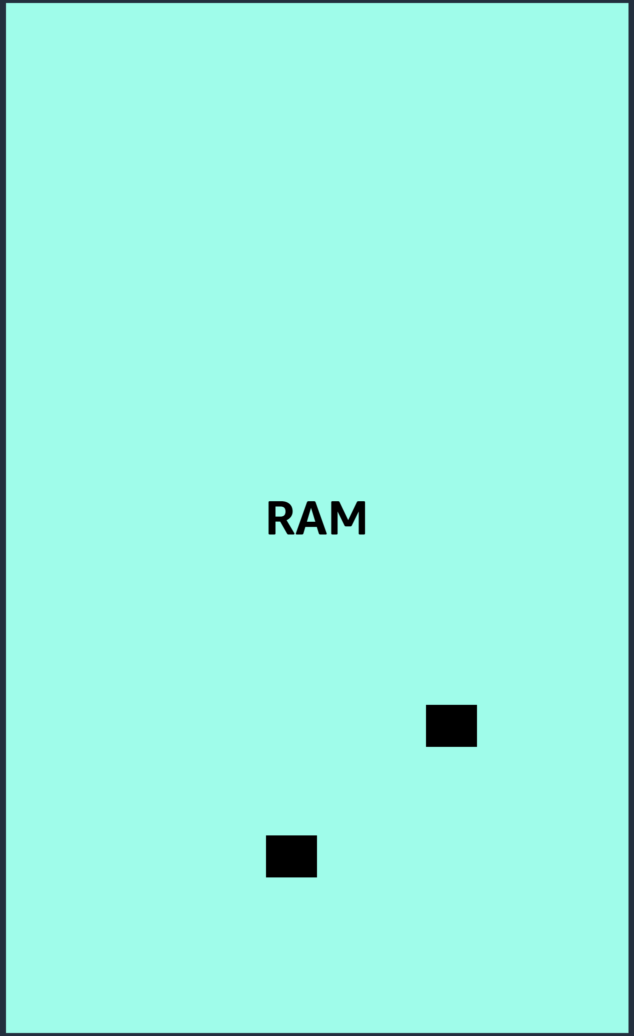
OOM & Swap

What is the Out of Memory (OOM) Killer

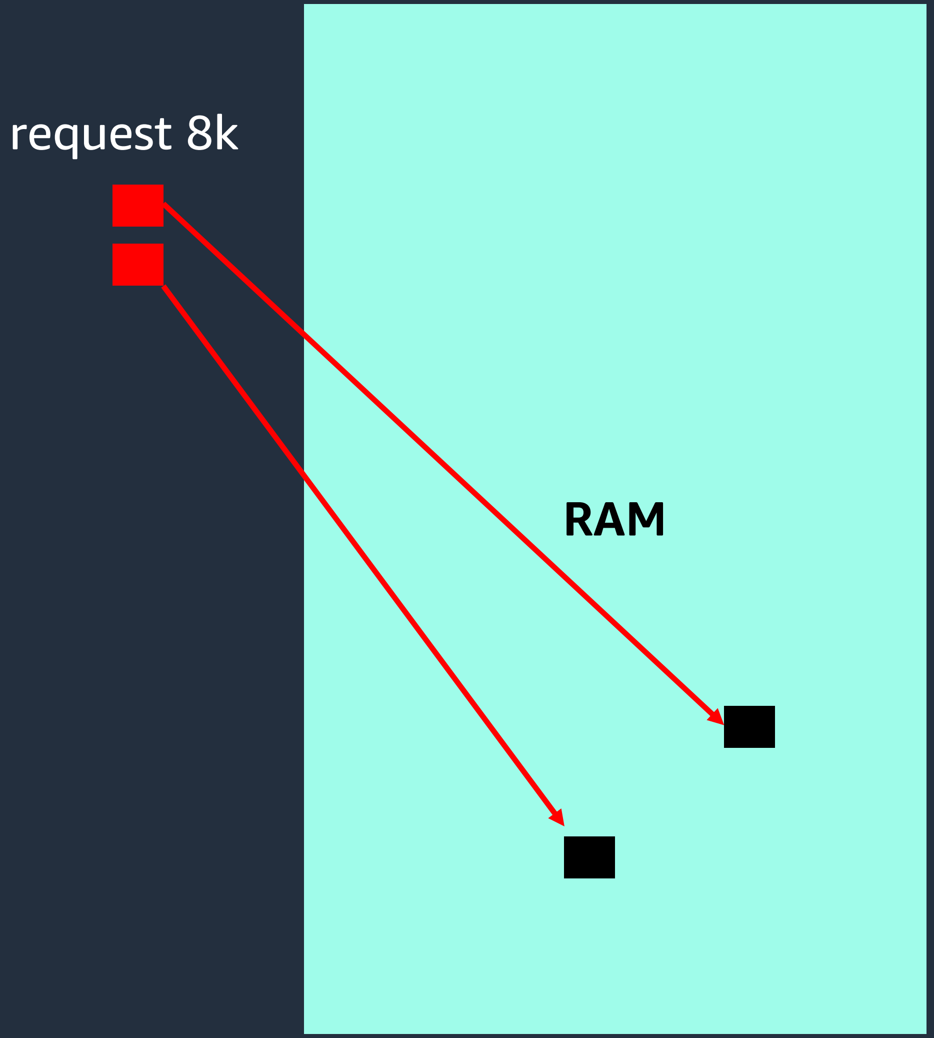


badness_for_task = total_vm_for_task / (sqrt(cpu_time_in_seconds) * sqrt(sqrt(cpu_time_in_minutes)))

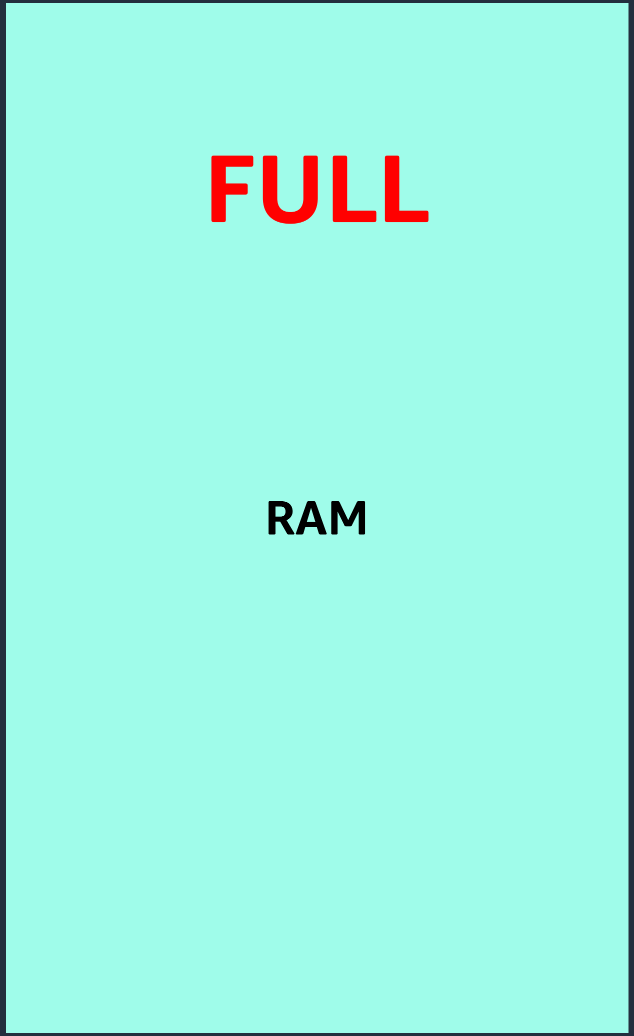
Why swap?



Why swap?



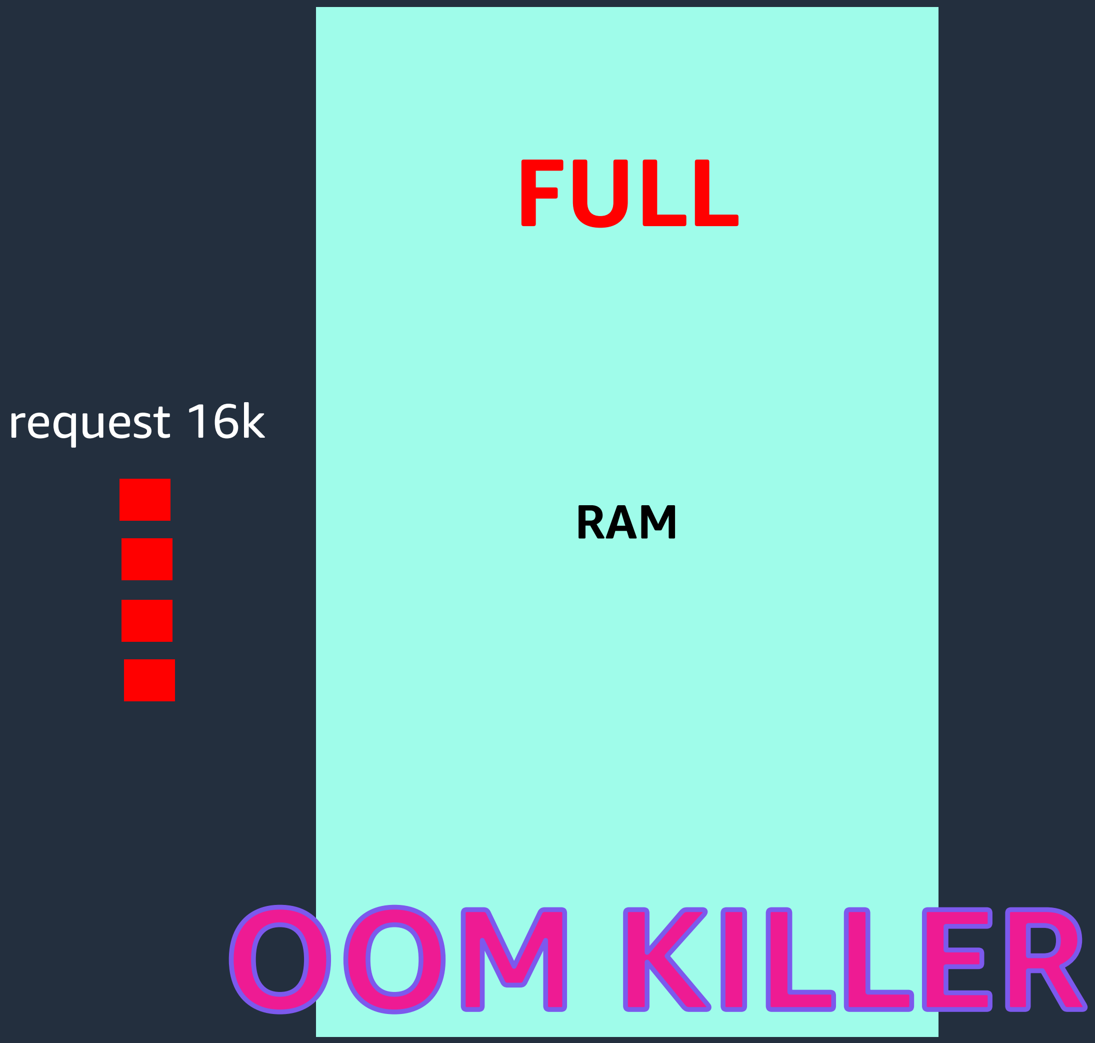
Why swap?



FULL

RAM

Why swap?



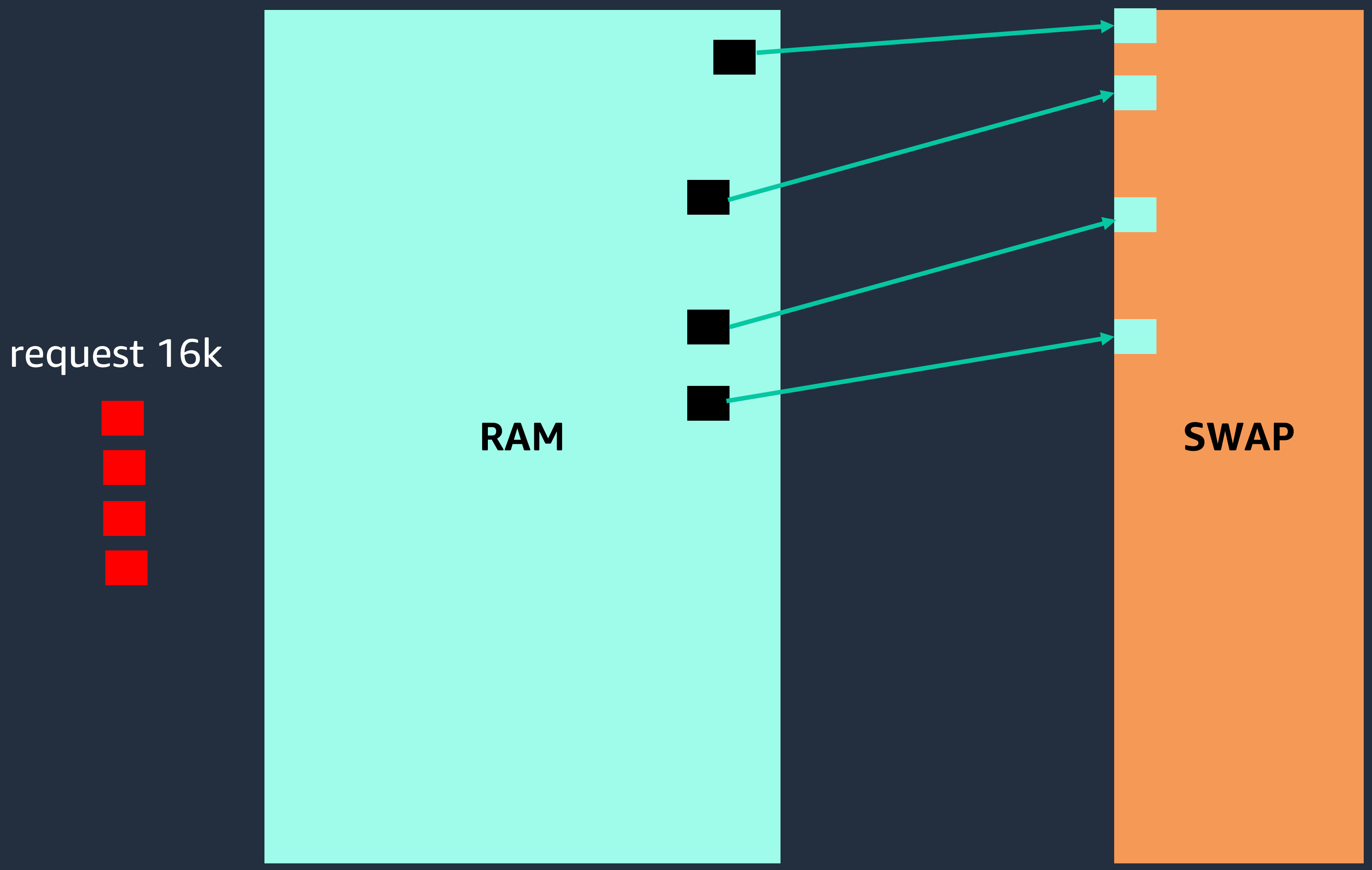
request 16k

FULL

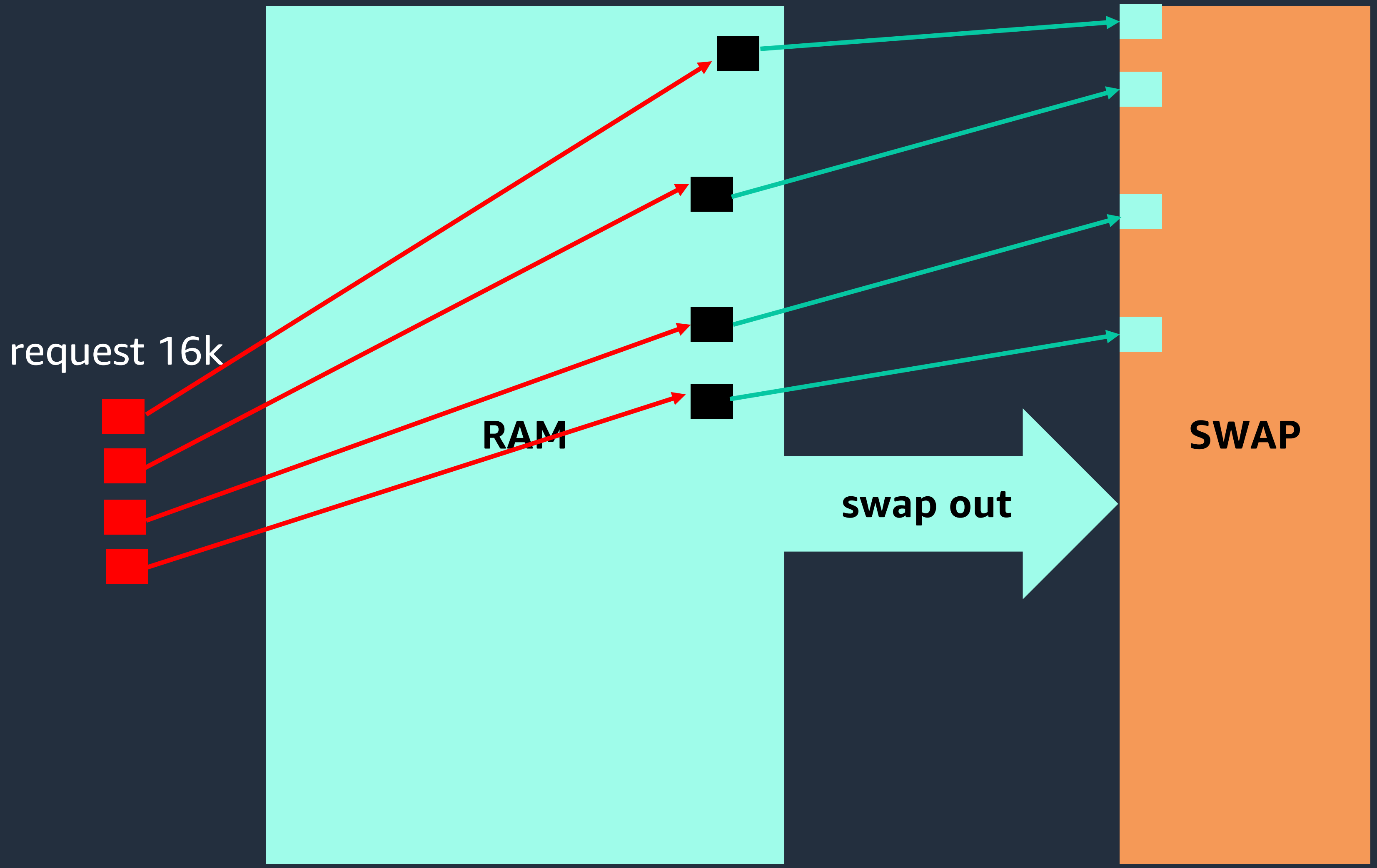
RAM

OOM KILLER

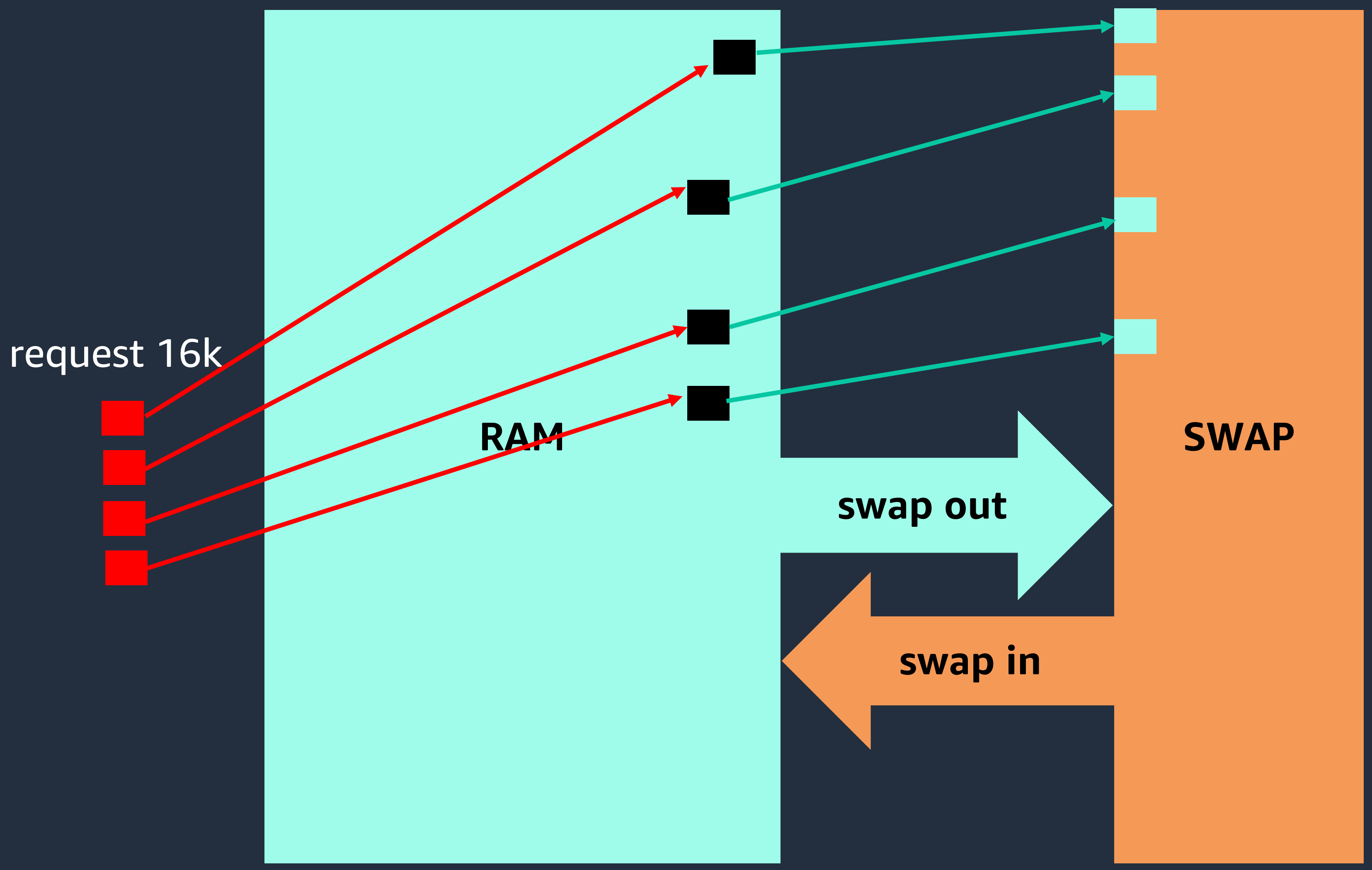
Why swap?



Why swap?



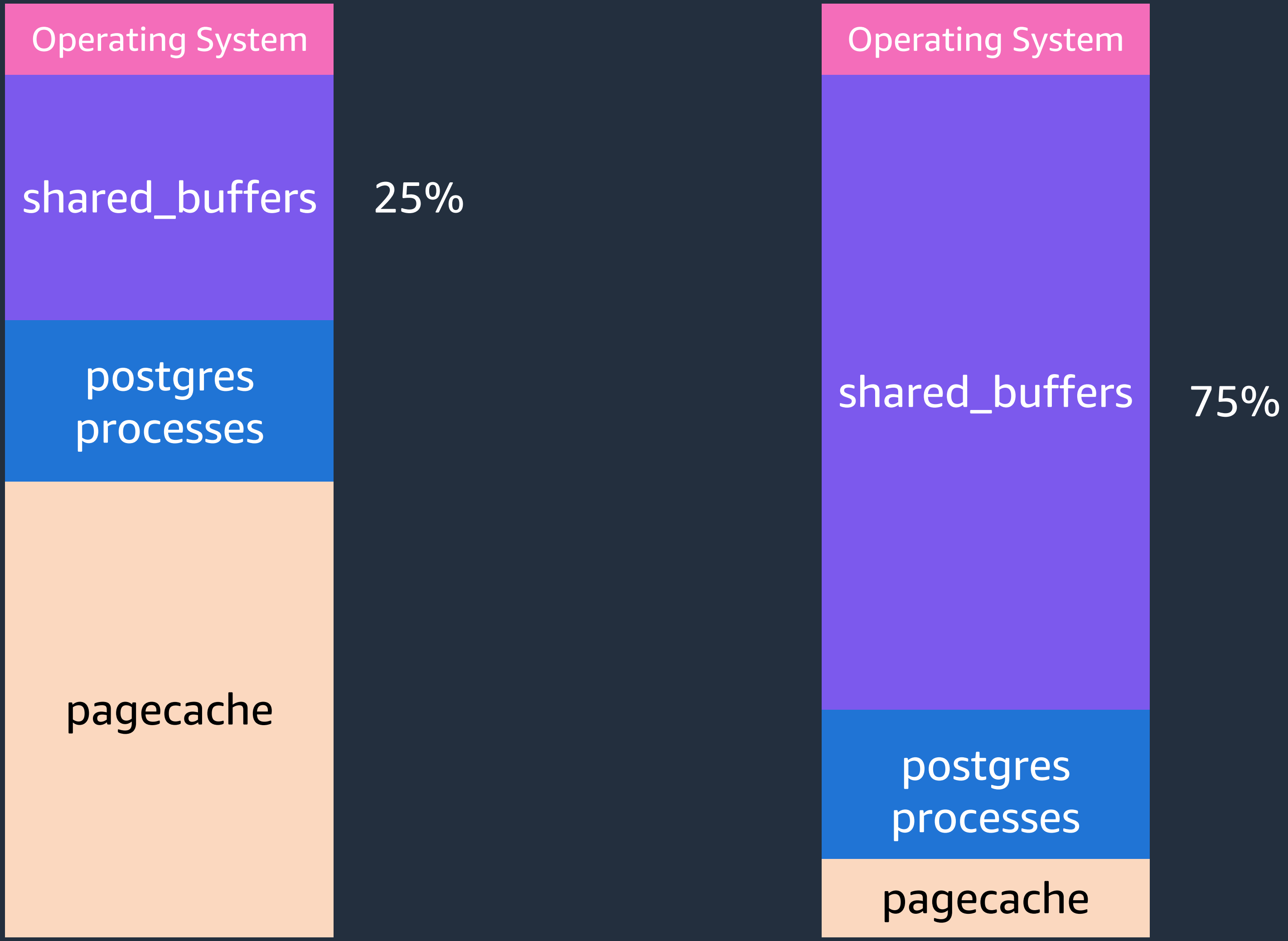
Why swap?



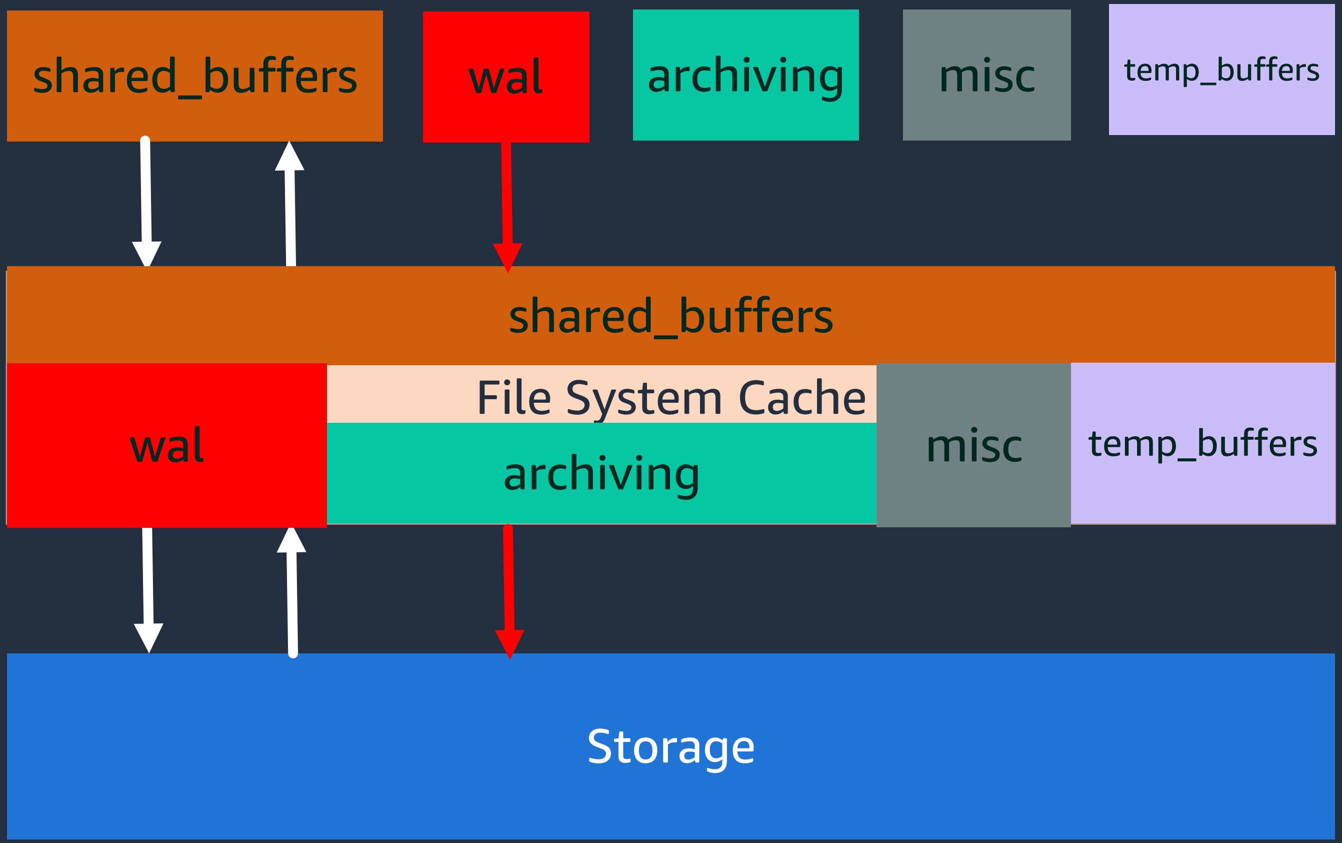
THRASHING

Overview

Memory Overview



Pagecache



shared_buffers

wal

archiving

misc

temp_buffers

shared_buffers

wal

File System Cache

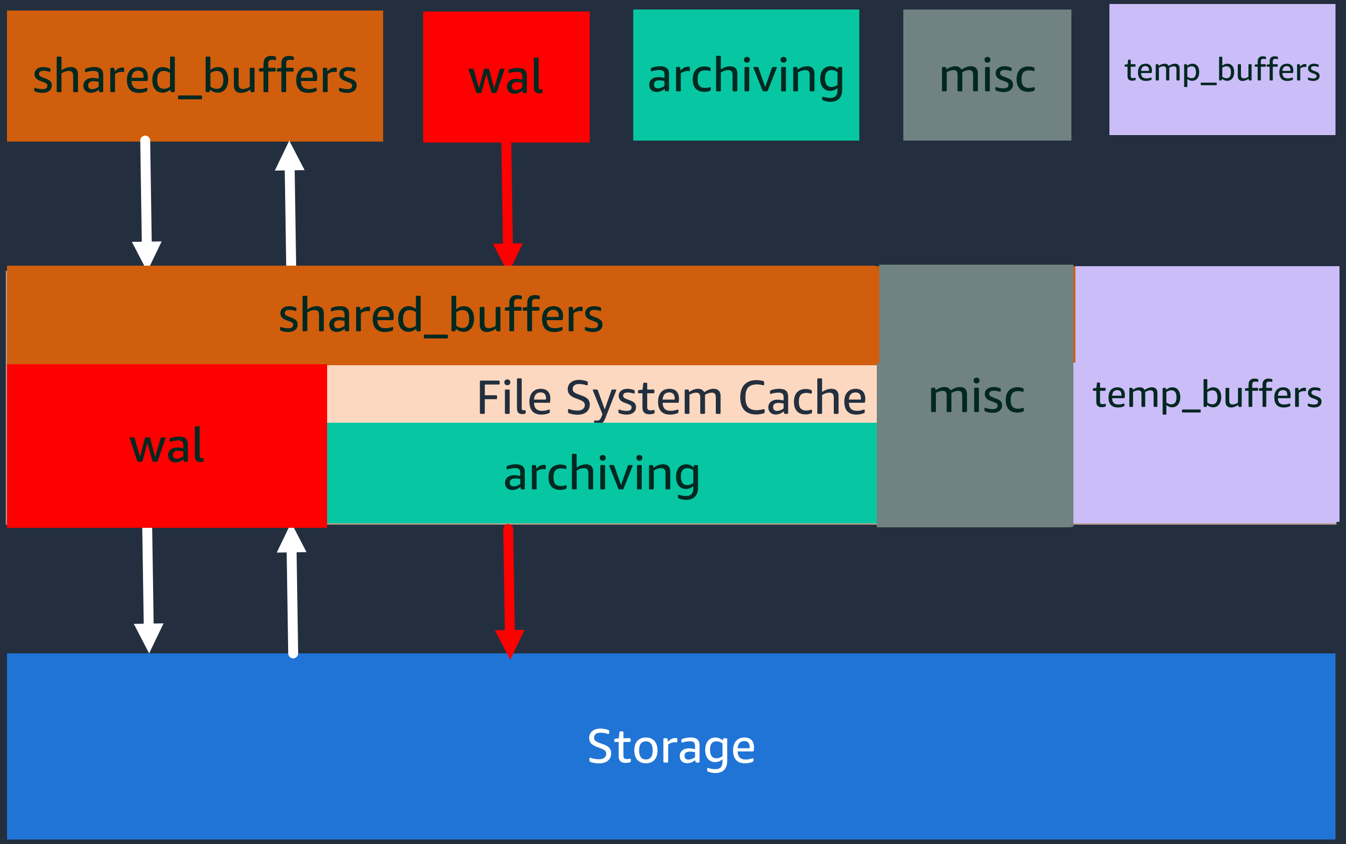
archiving

misc

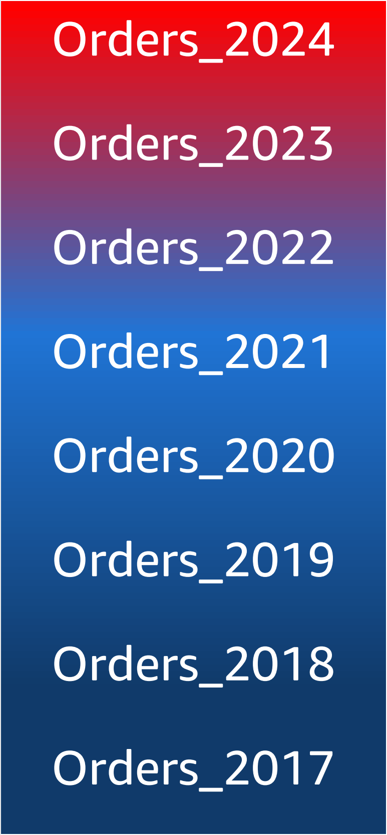
temp_buffers

Storage

Pagecache


shared_buffers
wal
archiving
misc
temp_buffers
shared_buffers
File System Cache
wal
archiving
misc
temp_buffers
Storage

Working Set Size - Heat



Orders_2024

Orders_2023

Orders_2022

Orders_2021

Orders_2020

Orders_2019

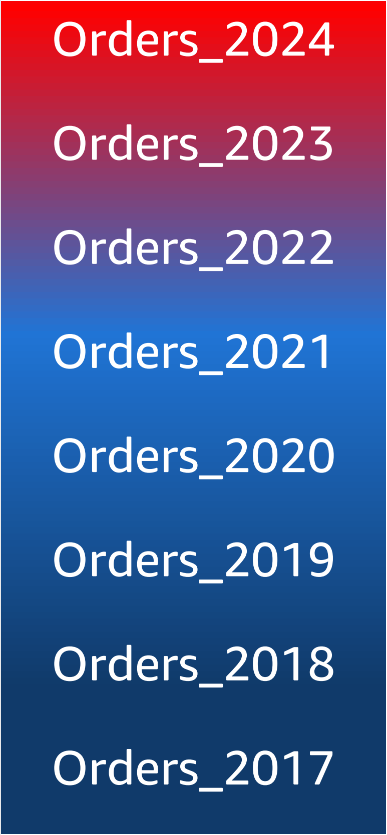
Orders_2018

Orders_2017

800GB

80-100GB hot working set

Working Set Size - Heat



Orders_2024

Orders_2023

Orders_2022

Orders_2021

Orders_2020

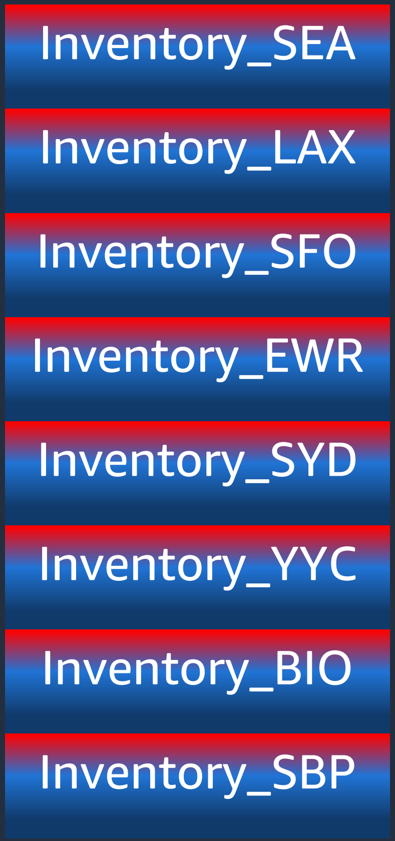
Orders_2019

Orders_2018

Orders_2017

800GB

80-100GB hot working set



Inventory_SEA

Inventory_LAX

Inventory_SFO

Inventory_EWR

Inventory_SYD

Inventory_YYC

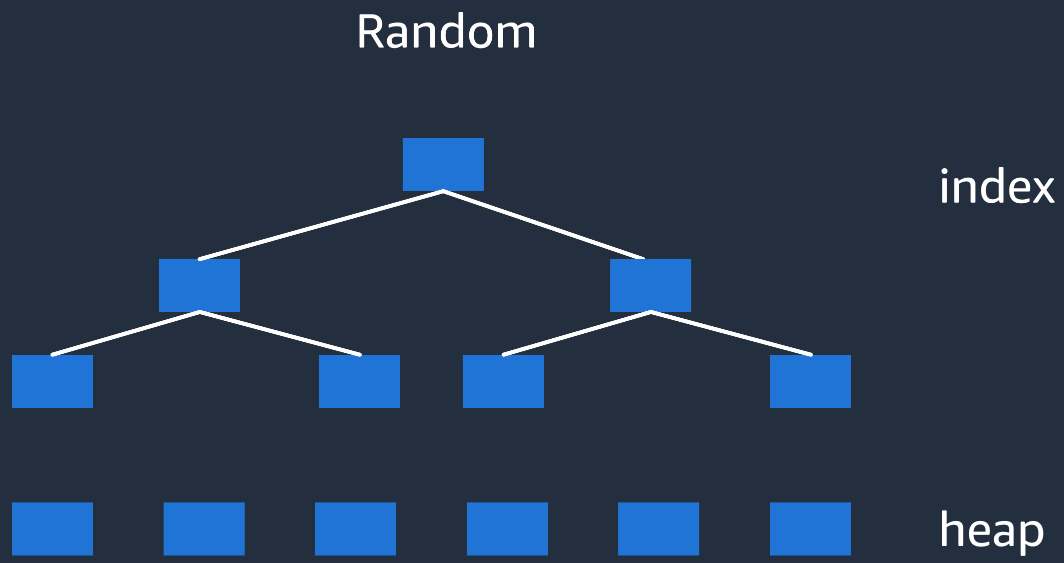
Inventory_BIO

Inventory_SBP

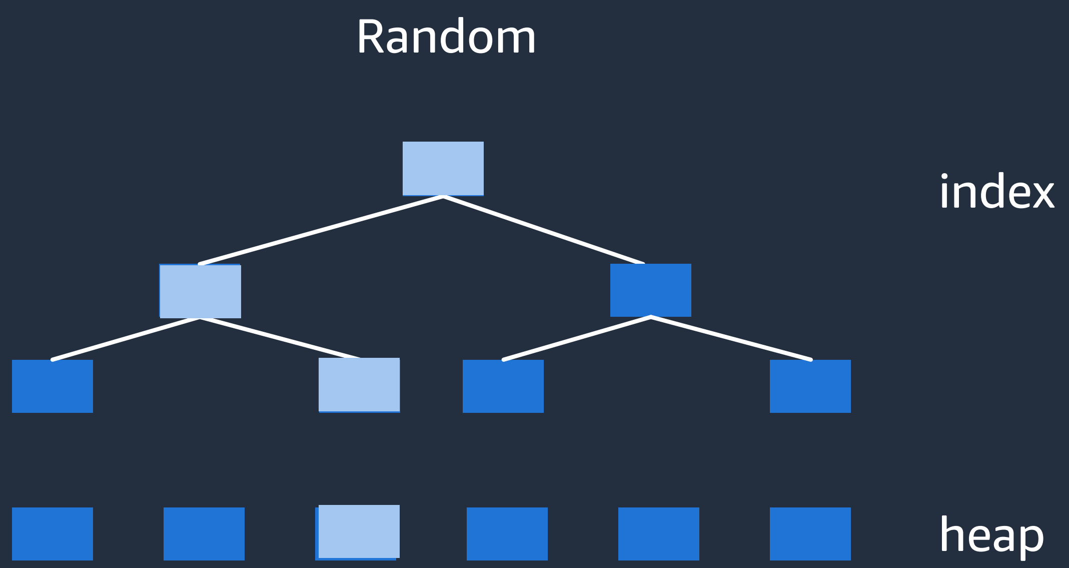
800GB

200-400GB hot working set

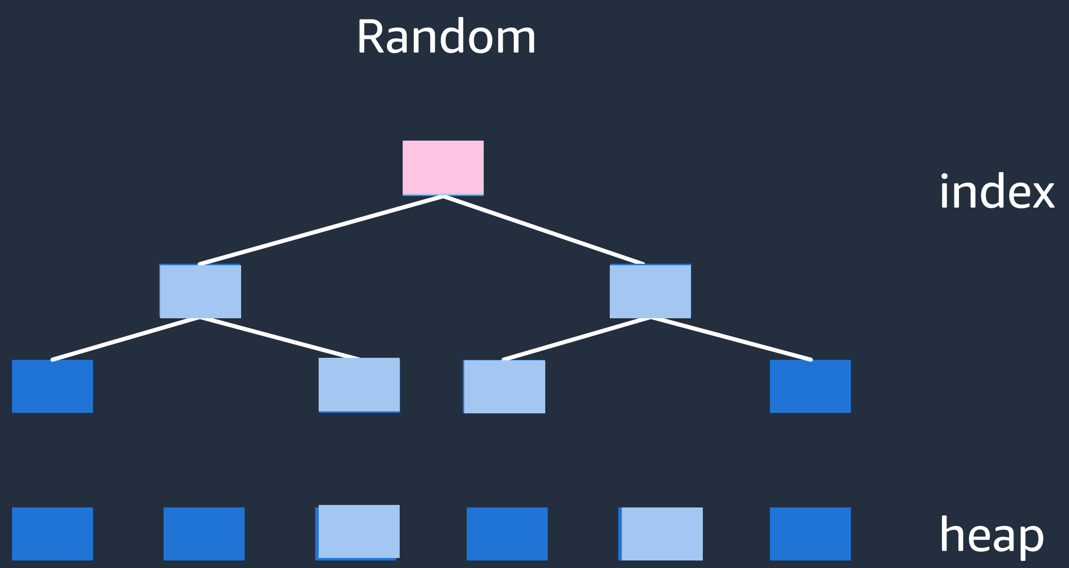
Working Set Size – Indexes and Data



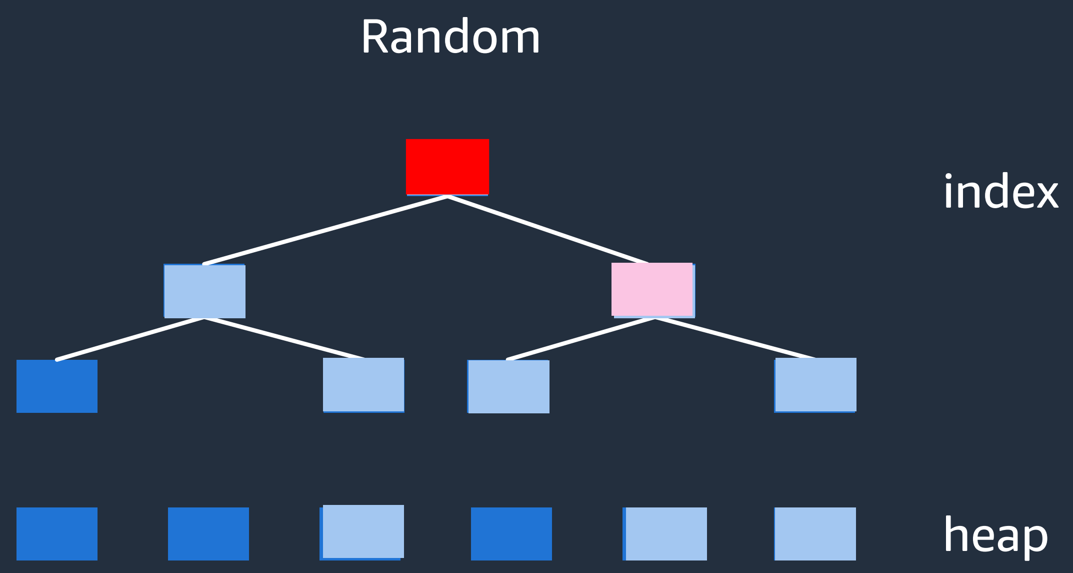
Working Set Size – Indexes and Data



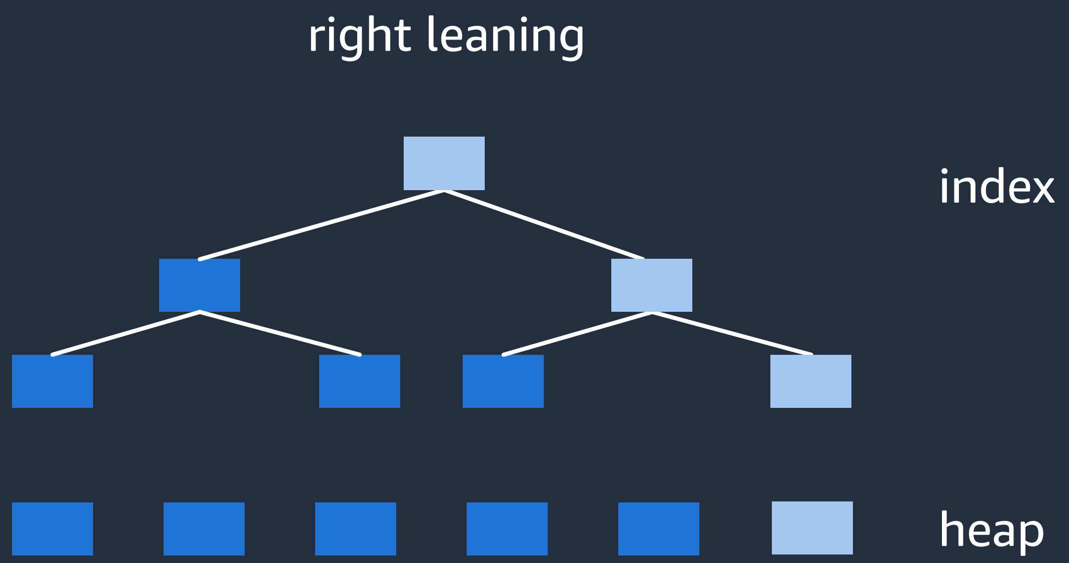
Working Set Size – Indexes and Data



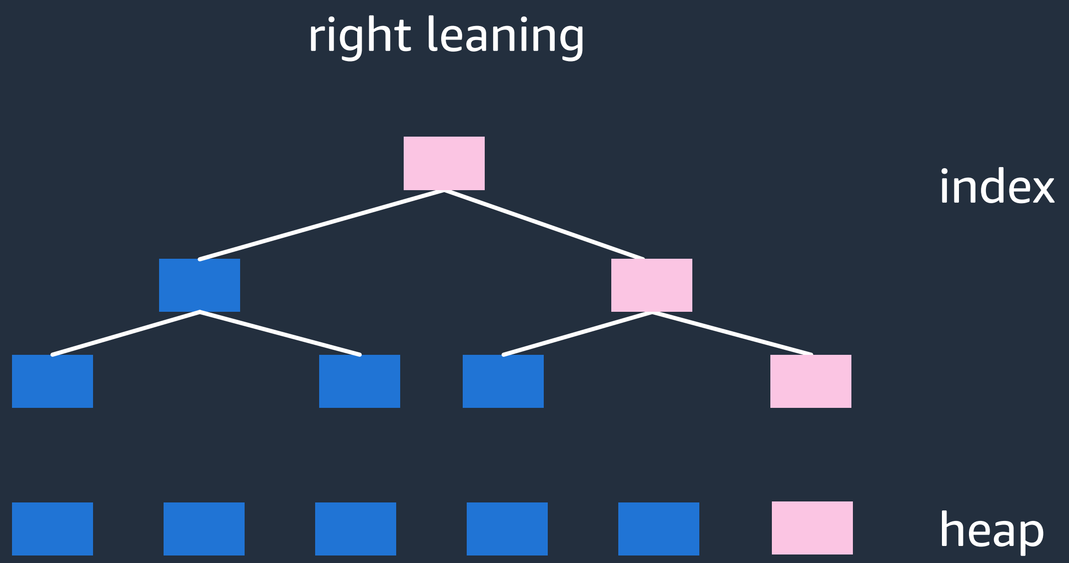
Working Set Size – Indexes and Data



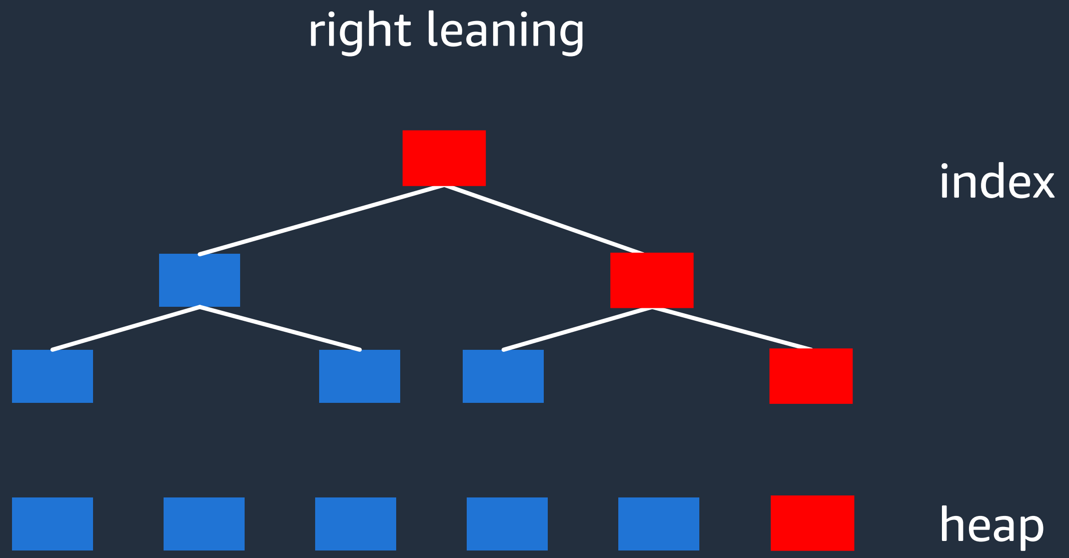
Working Set Size – Indexes and Data



Working Set Size – Indexes and Data

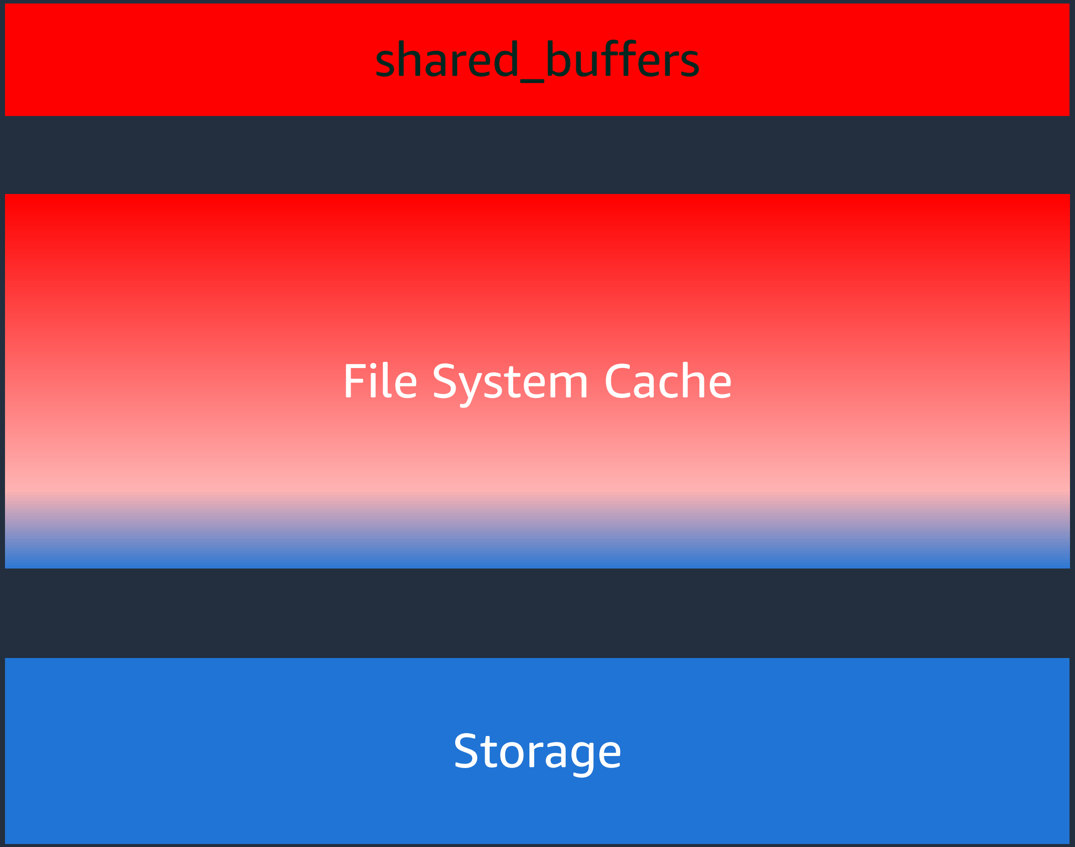


Working Set Size – Indexes and Data



Shared Buffers

Small Cache

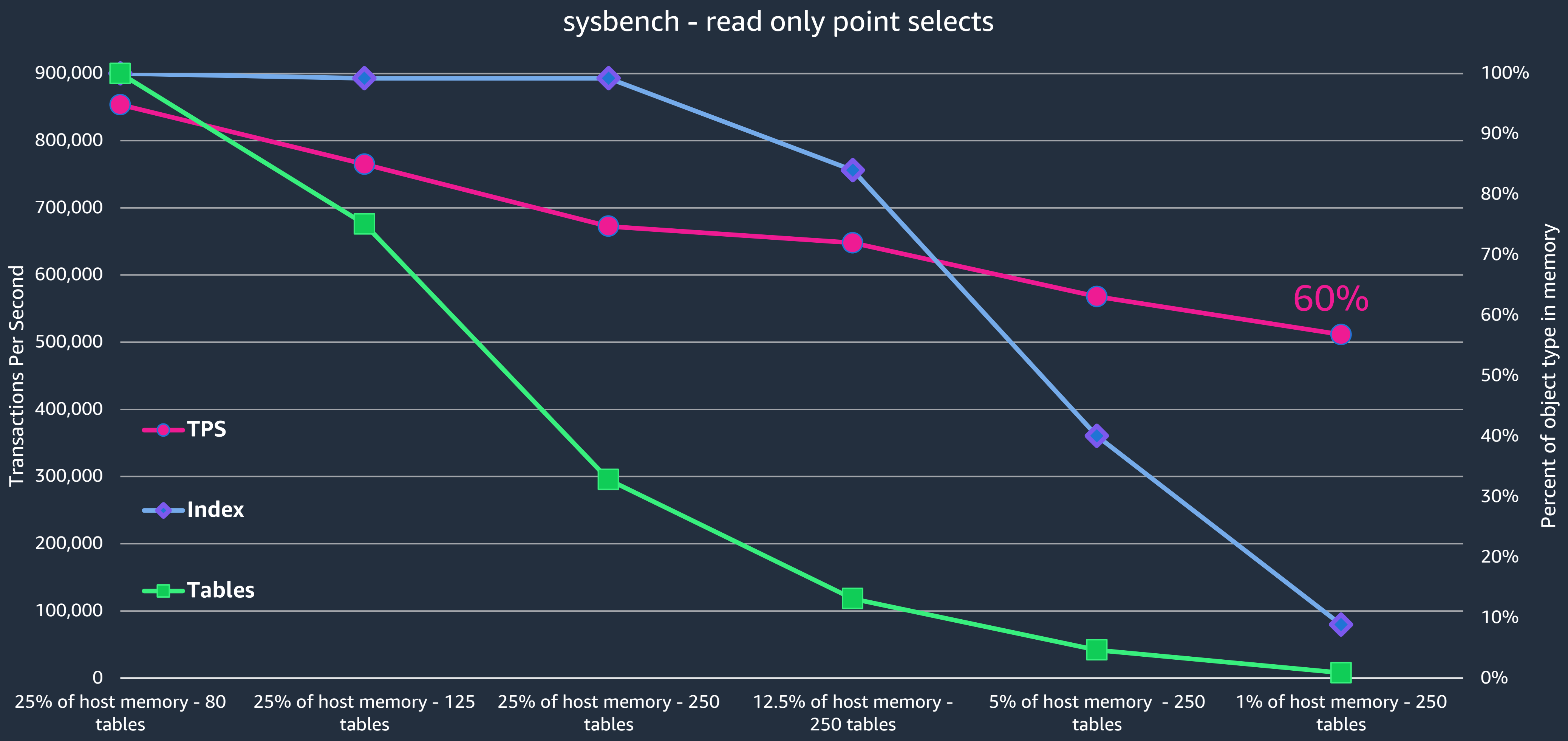


shared_buffers

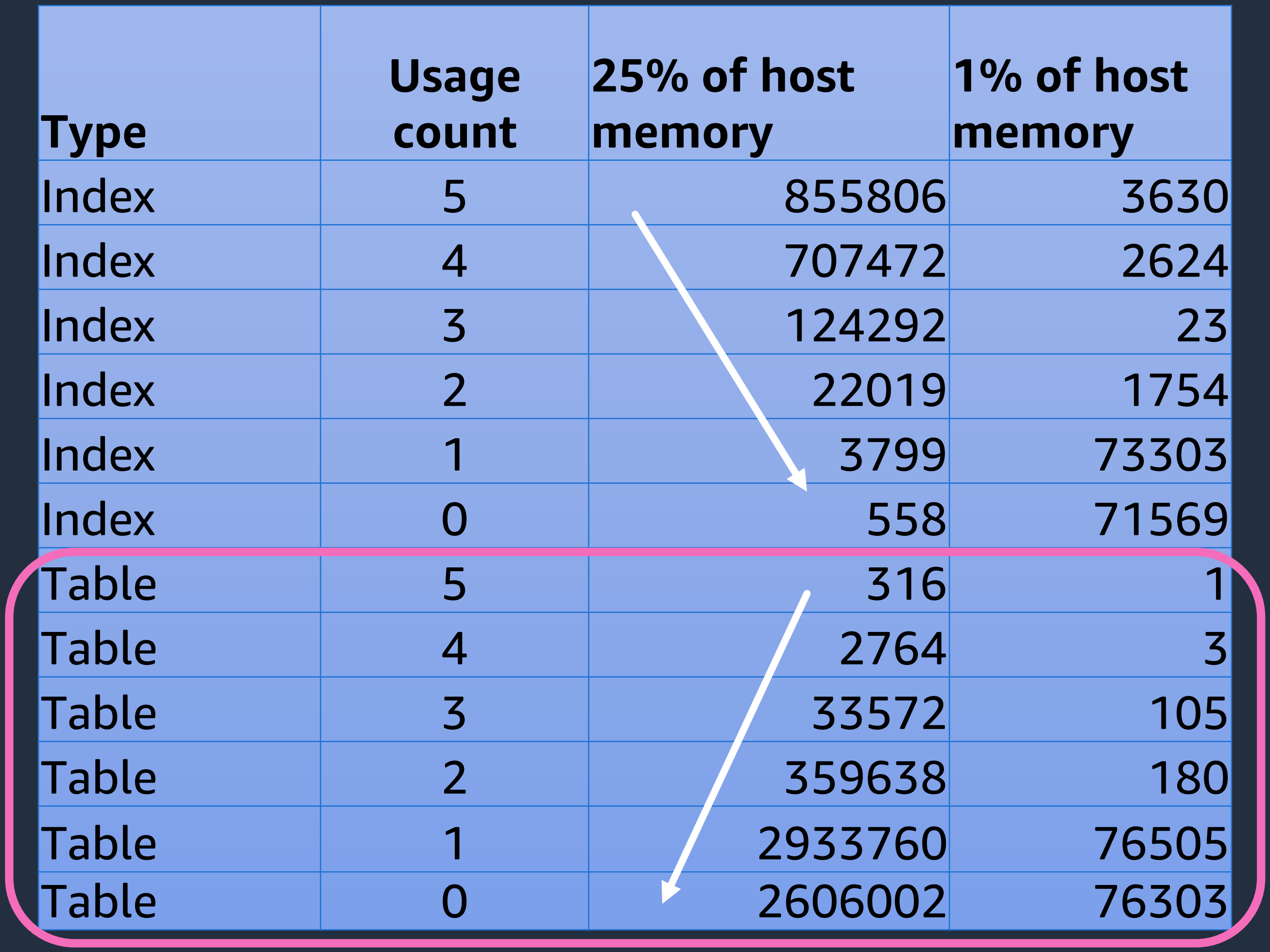
File System Cache

Storage

shared_buffers comparison

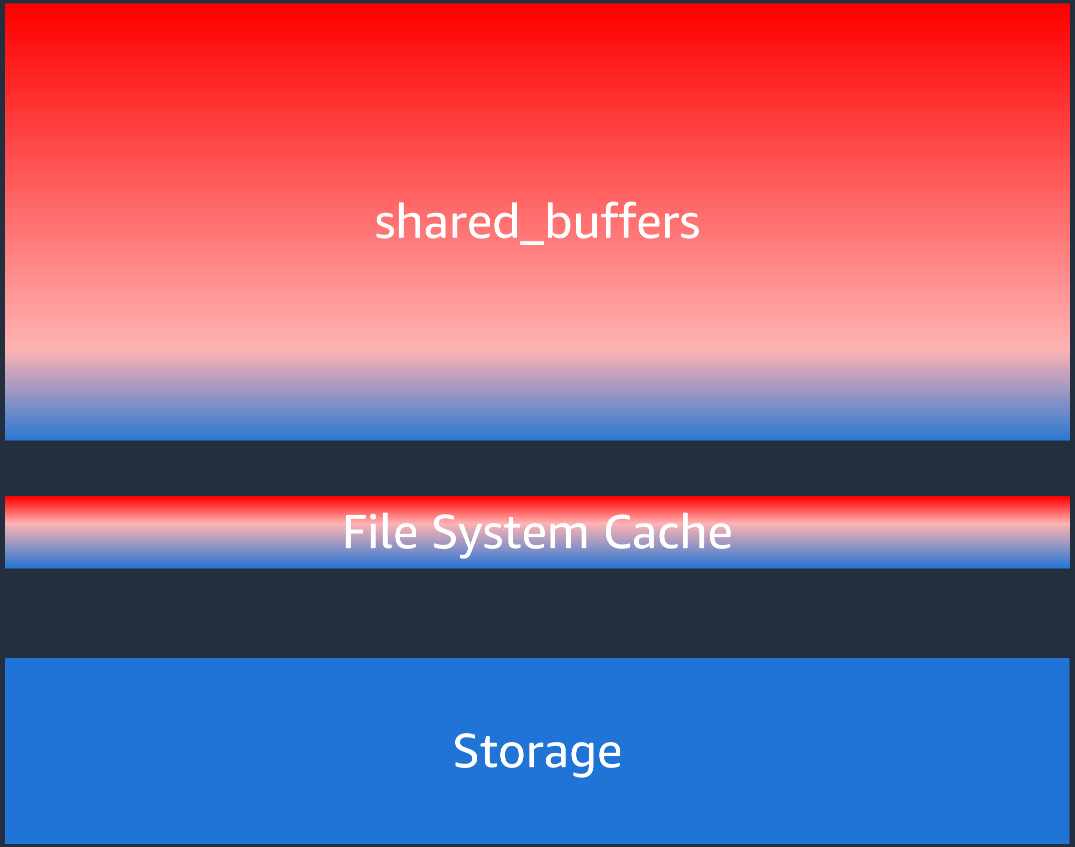


shared_buffers comparison – pg_buffercache



Type	Usage count	25% of host memory	1% of host memory
Index	5	855806	3630
Index	4	707472	2624
Index	3	124292	23
Index	2	22019	1754
Index	1	3799	73303
Index	0	558	71569
Table	5	316	1
Table	4	2764	3
Table	3	33572	105
Table	2	359638	180
Table	1	2933760	76505
Table	0	2606002	76303

Big Cache

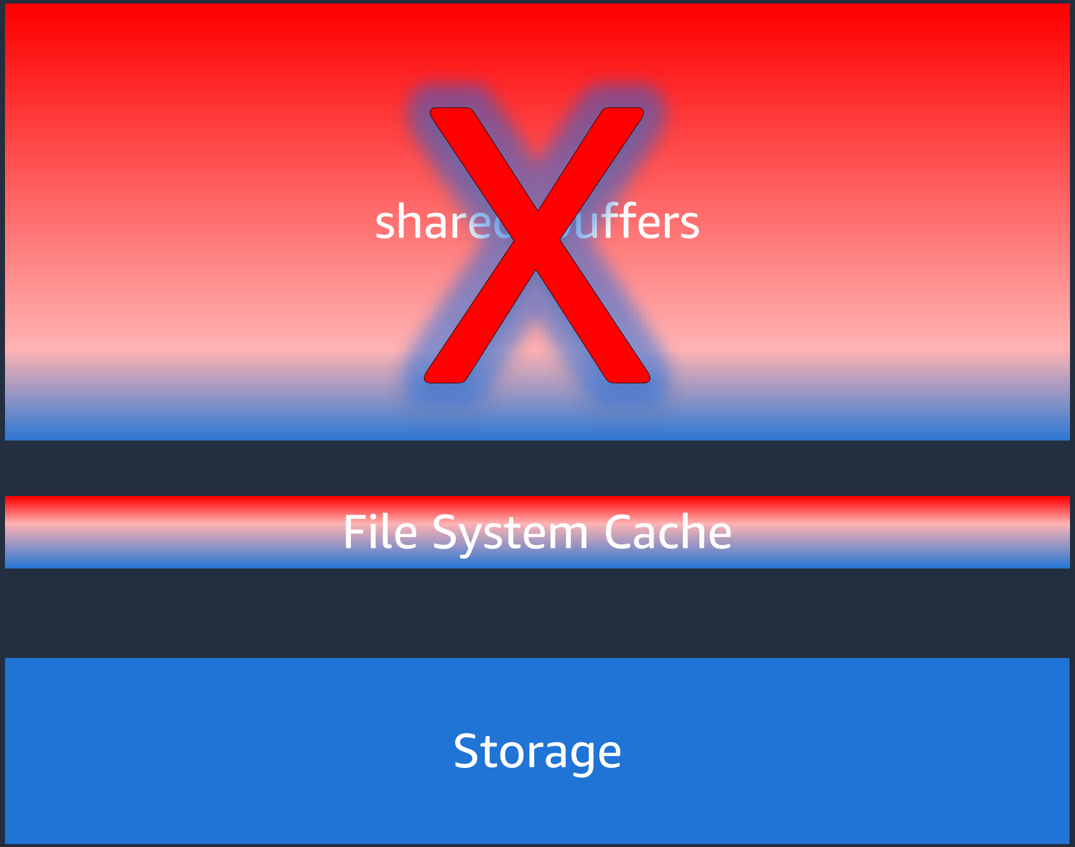


shared_buffers

File System Cache

Storage

Big Cache

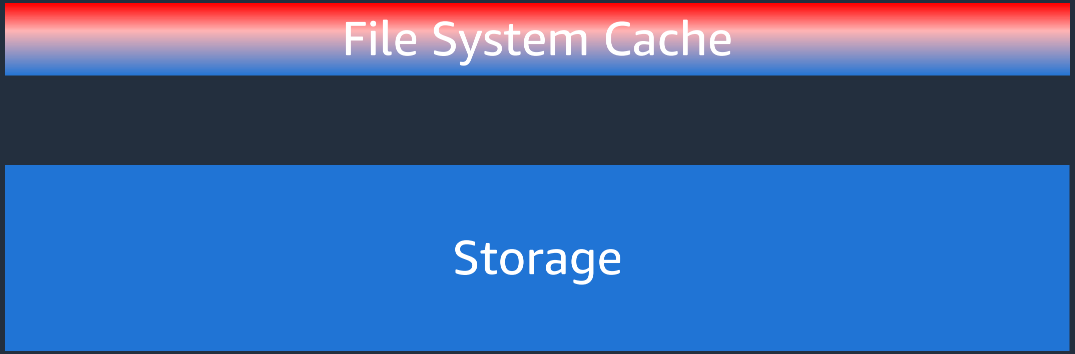


~~shared_buffers~~

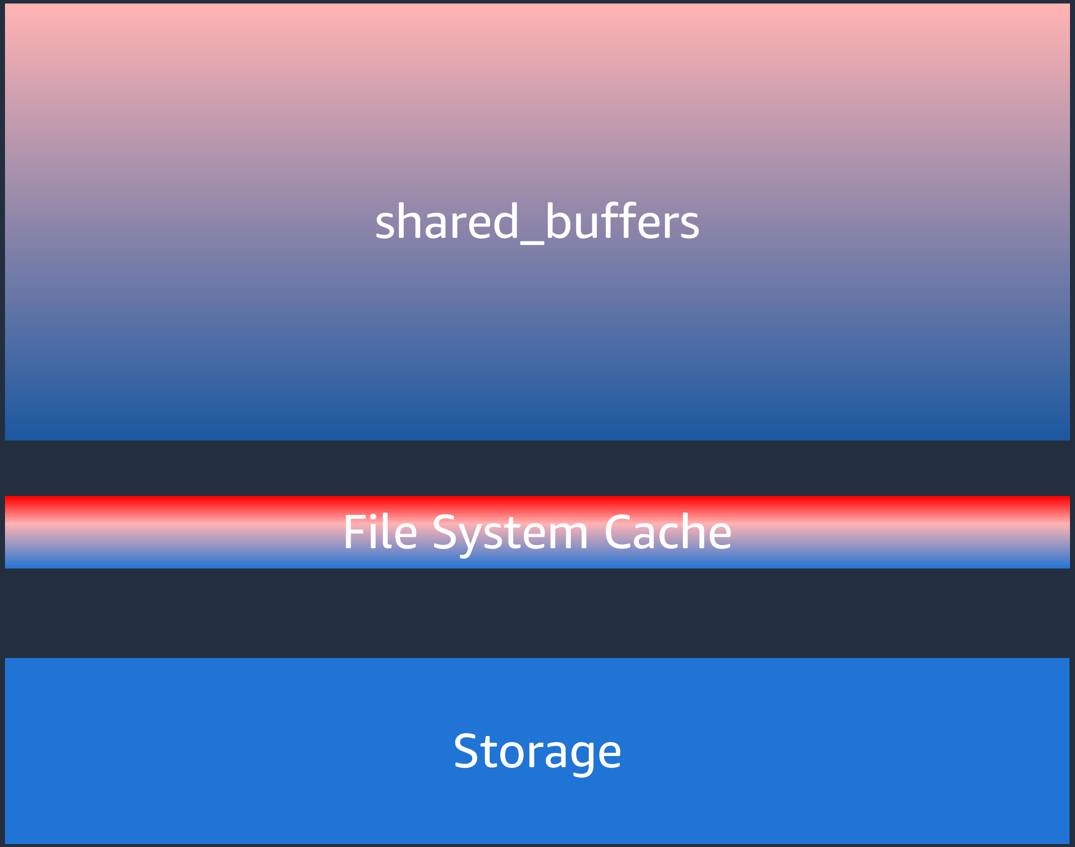
File System Cache

Storage

Big Cache



Big Cache



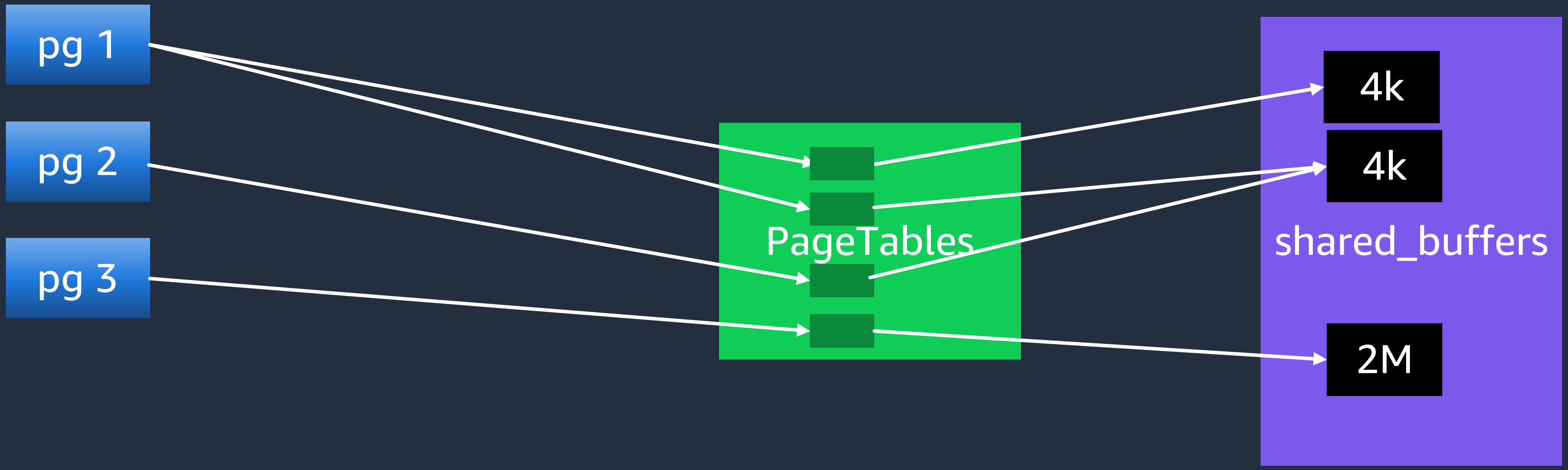
shared_buffers

File System Cache

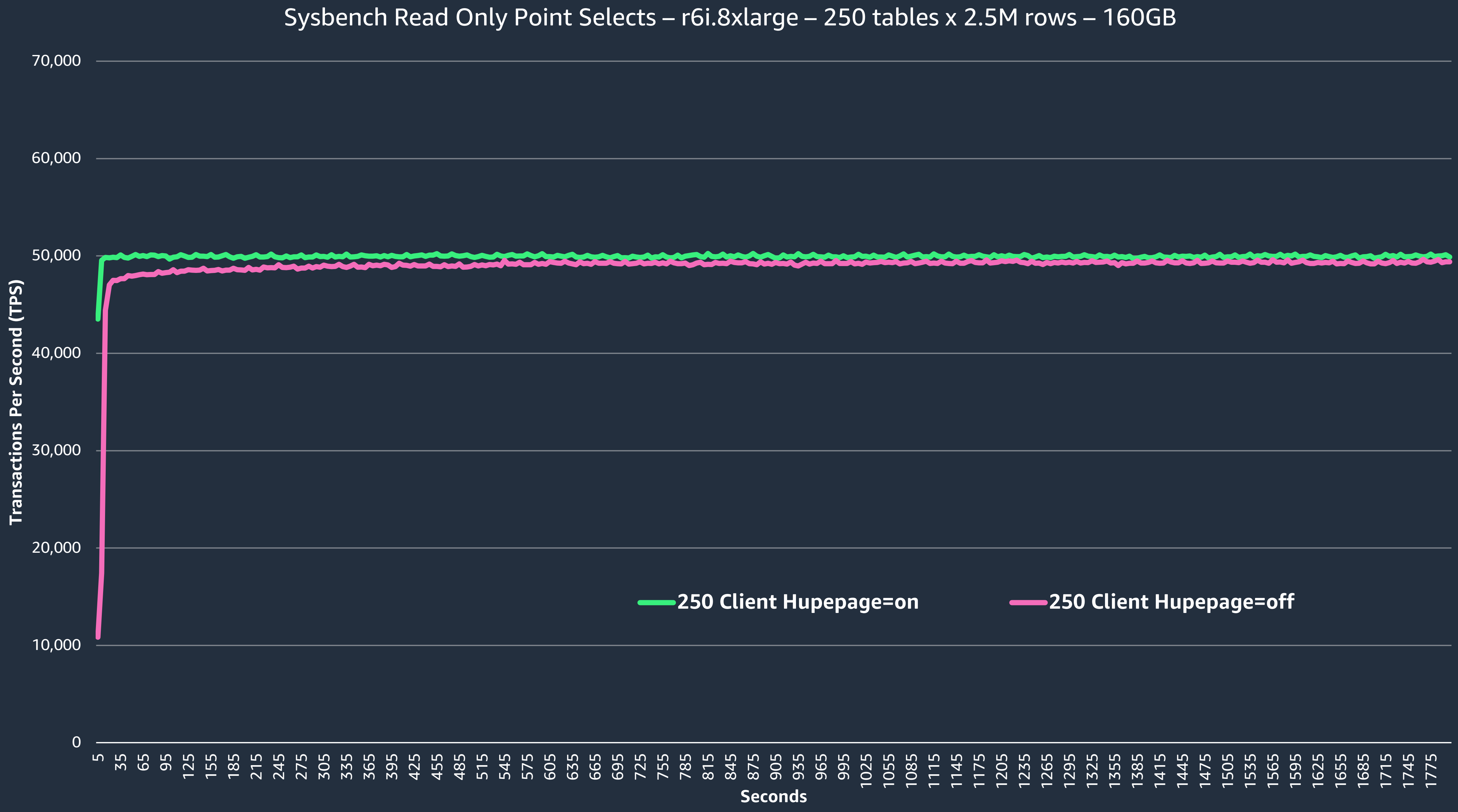
Storage

HugePages

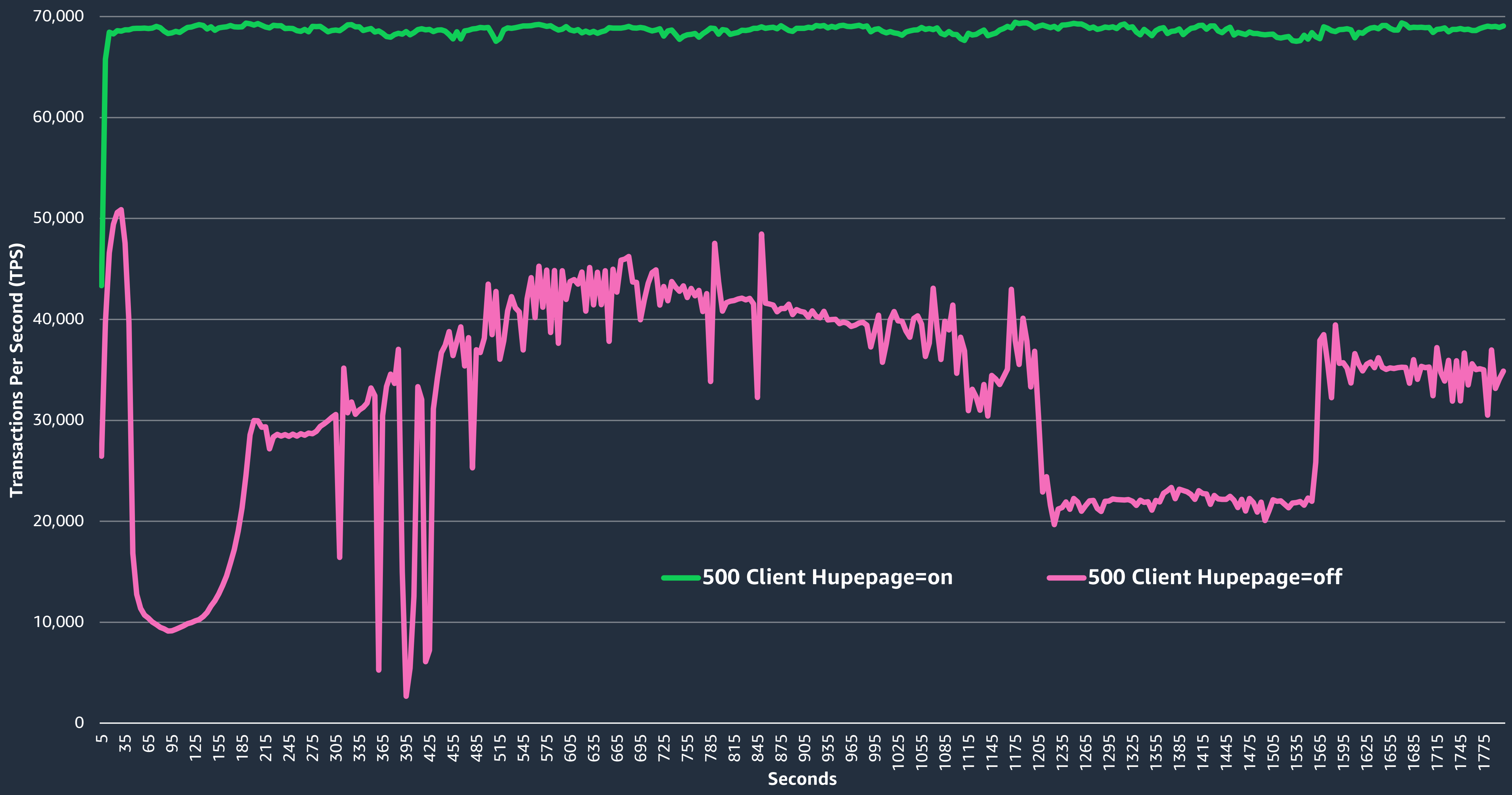
Why HugePages – page mapping



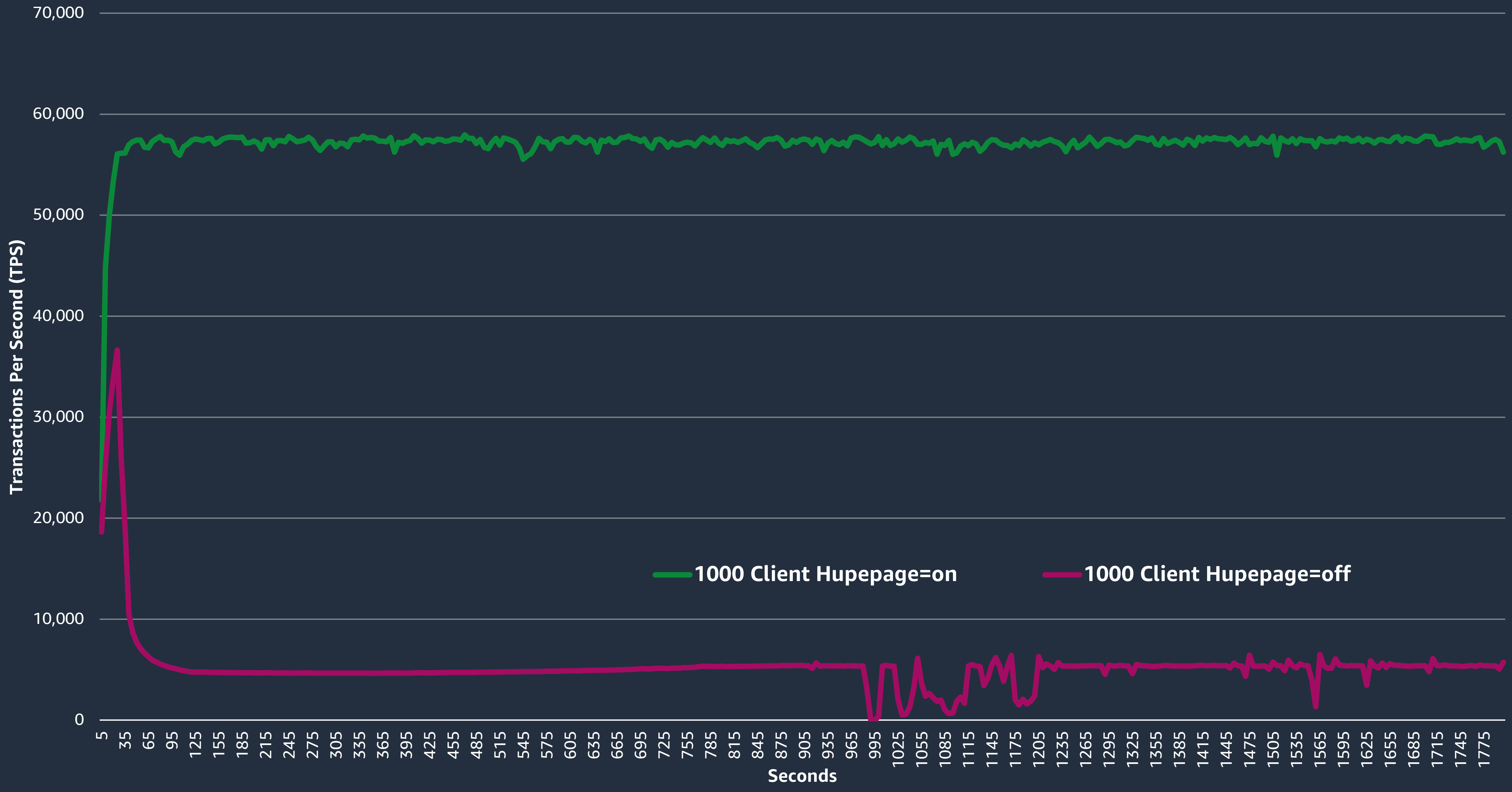
size of page table = # of PostgreSQL process **X** amount of shared buffers accessed


Sysbench Read Only Point Selects – r6i.8xlarge – 250 tables x 2.5M rows – 160GB
Transactions Per Second (TPS)
70,000
60,000
50,000
40,000
30,000
20,000
10,000
0
250 Client Hupepage=on
250 Client Hupepage=off
5
35
65
95
125
155
185
215
245
275
305
335
365
395
425
455
485
515
545
575
605
635
665
695
725
755
785
815
845
875
905
935
965
995
1025
1055
1085
1115
1145
1175
1205
1235
1265
1295
1325
1355
1385
1415
1445
1475
1505
1535
1565
1595
1625
1655
1685
1715
1745
1775
Seconds

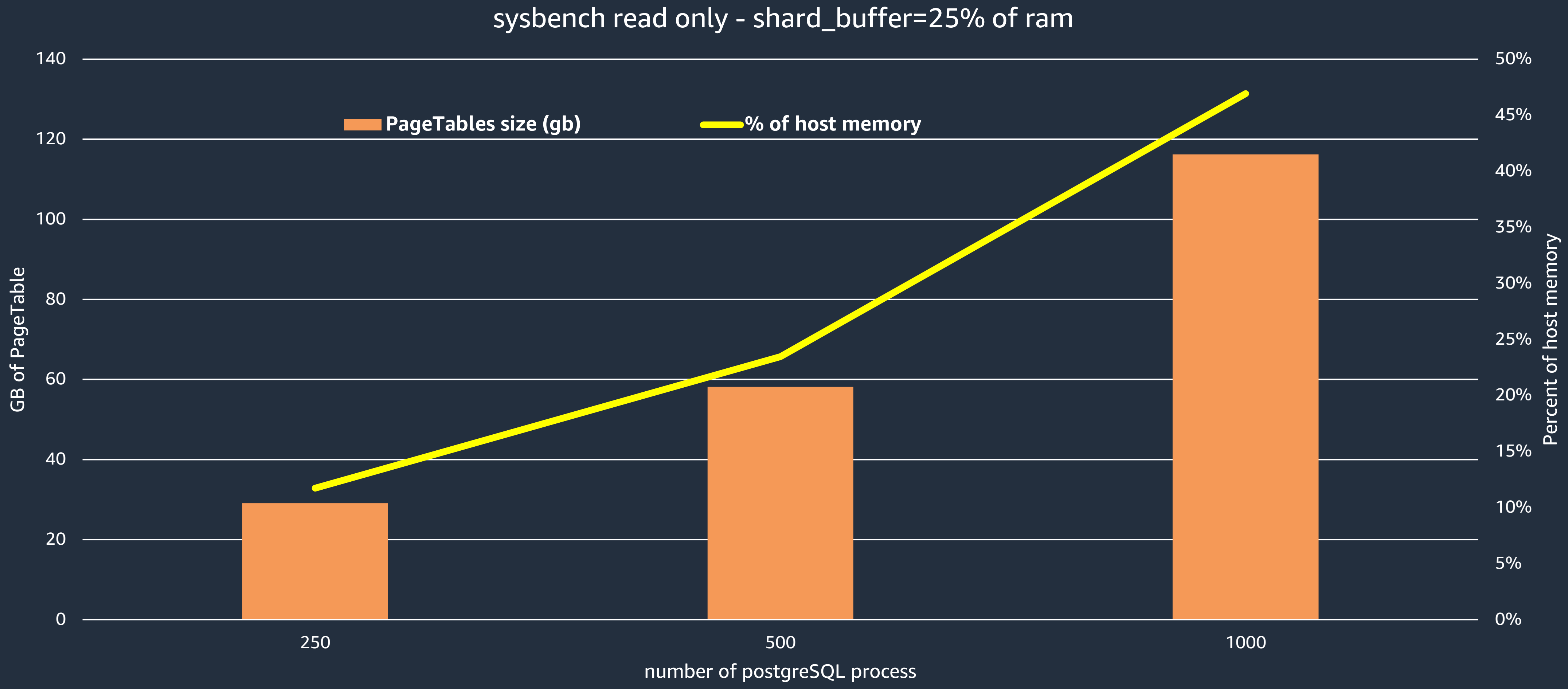
Sysbench Read Only Point Selects – r6i.8xlarge – 250 tables x 2.5M rows – 160GB



Sysbench Read Only Point Selects – r6i.8xlarge – 250 tables x 2.5M rows – 160GB



Cost of not setting HugePages



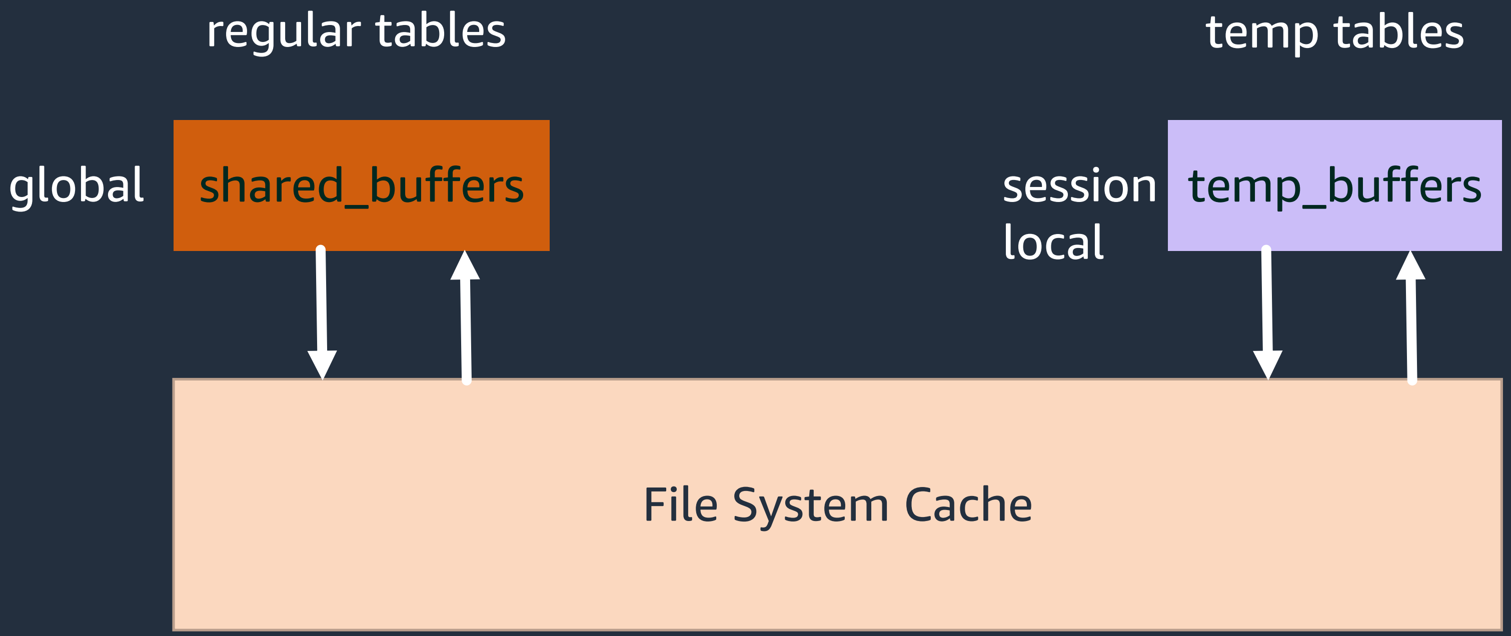
Other Cluster Wide Memory Parameters

Cluster wide Parameters

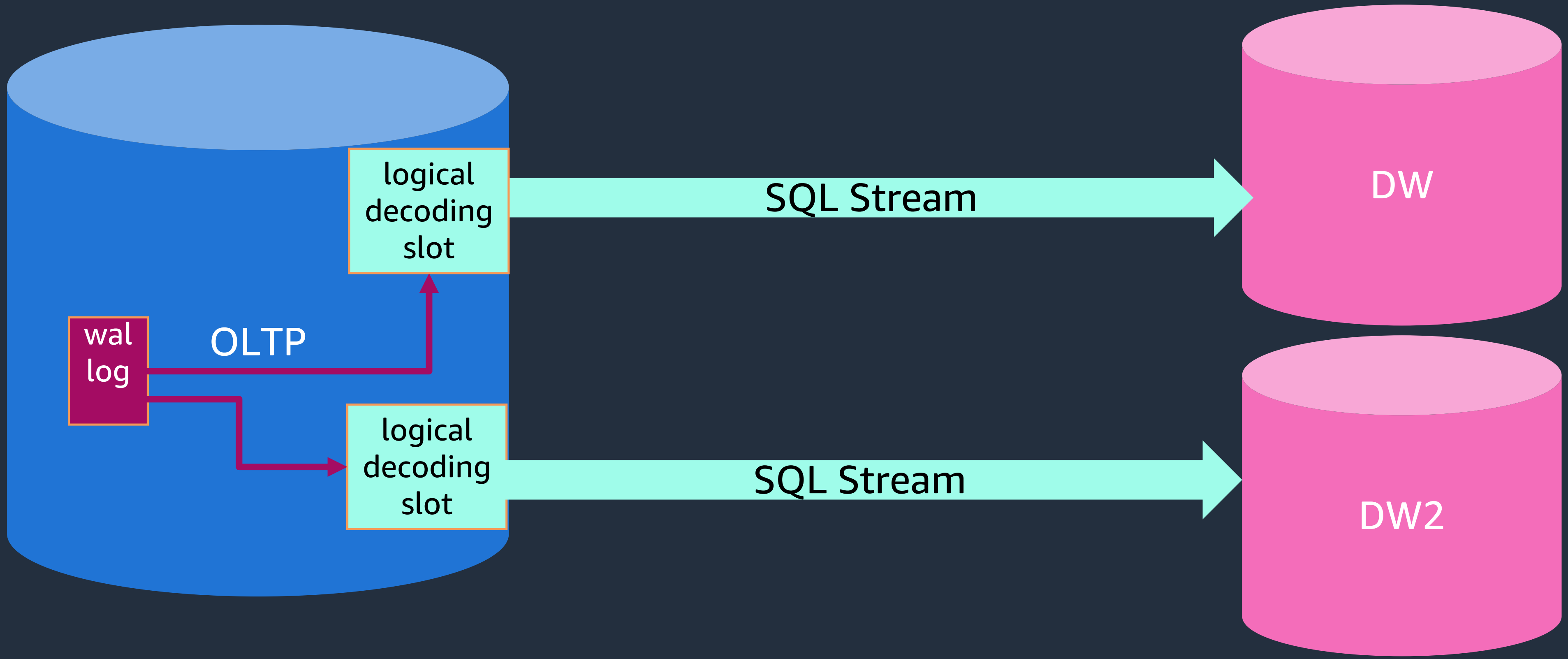
Name	Default
commit_timestamp_buffers	shared_buffers/512 up to 1024 blocks, but not fewer than 16 block
multixact_member_buffers	32 * 8KB
multixact_offset_buffers	16 * 8KB
notify_buffers	16 * 8KB
serializable_buffers	32 * 8KB
subtransaction_buffers	shared_buffers/512 up to 1024 blocks, but not fewer than 16 block
transaction_buffers	shared_buffers/512 up to 1024 blocks, but not fewer than 16 block
max_prepared_transactions	for XA (please don't use XA)

Per Session

temp_buffers



logical_decoding_work_mem



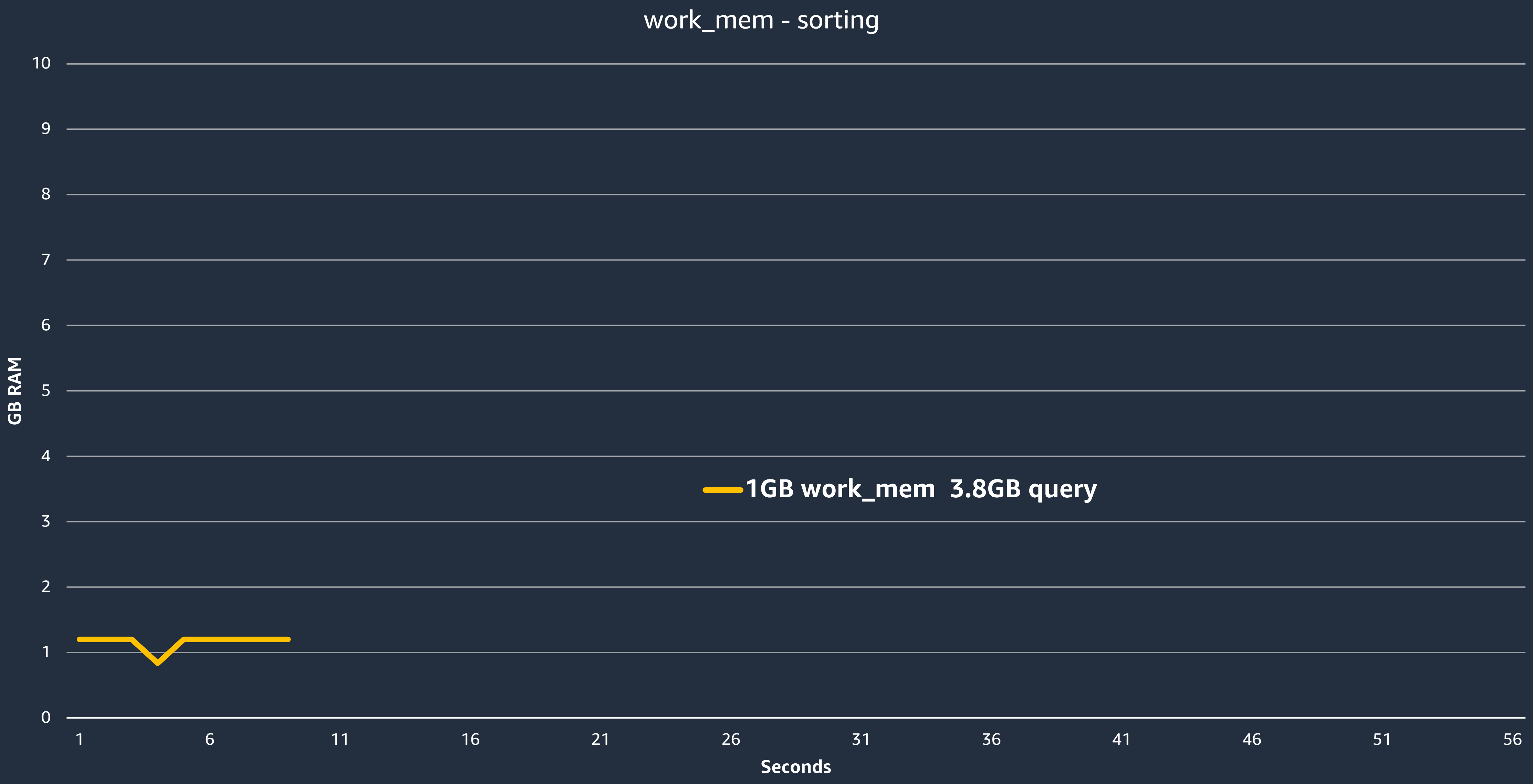
Per Session and Operation

work_mem

```
postgres=# set work_mem TO '1 GB’;

postgres=# explain analyze
select mykey::bigint, (random()*10000000000)::bigint as scratch ,
repeat('X', 1024)::char(1024) filler from generate_series(1,3800000)
as mykey order by scratch;
```

~3.8GB

```
                                                                   QUERY PLAN
------------------------------------------------------------------------------------------------------------------------------------------------
-
 Sort  (cost=7219224.79..7228724.79 rows=3800000 width=4116) (actual time=7083.506..8358.093 rows=3800000 loops=1)
   Sort Key: (((random() * '10000000000'::double precision))::bigint)
   Sort Method: external merge  Disk: 3919000kB
   ->  Function Scan on generate_series mykey  (cost=0.00..76000.00 rows=3800000 width=4116) (actual time=194.053..493.117 rows=3800000 loops=1)
 Planning Time: 0.049 ms
 Execution Time: 9117.344 ms
(6 rows)
```

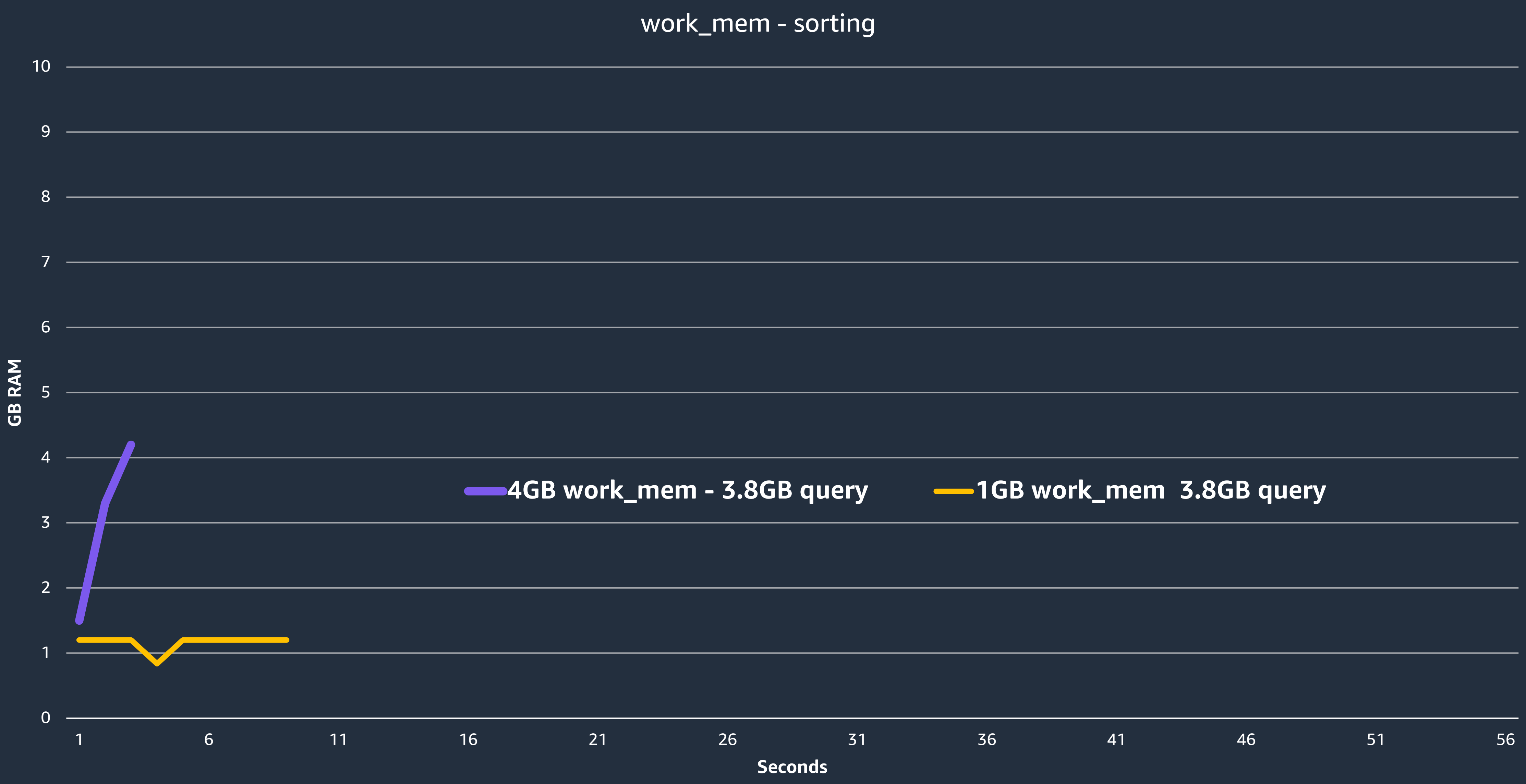

work_mem - sorting
GB RAM
10
9
8
7
6
5
4
3
2
1
0
1
6
11
16
21
26
31
36
41
46
51
56
Seconds
1GB work_mem 3.8GB query

work_mem

```
postgres=# set work_mem TO '4 GB’;

postgres=# explain analyze
select mykey::bigint, (random()*10000000000)::bigint as scratch ,
repeat('X', 1024)::char(1024) filler from generate_series(1,3800000)
as mykey order by scratch;
```

~3.8GB

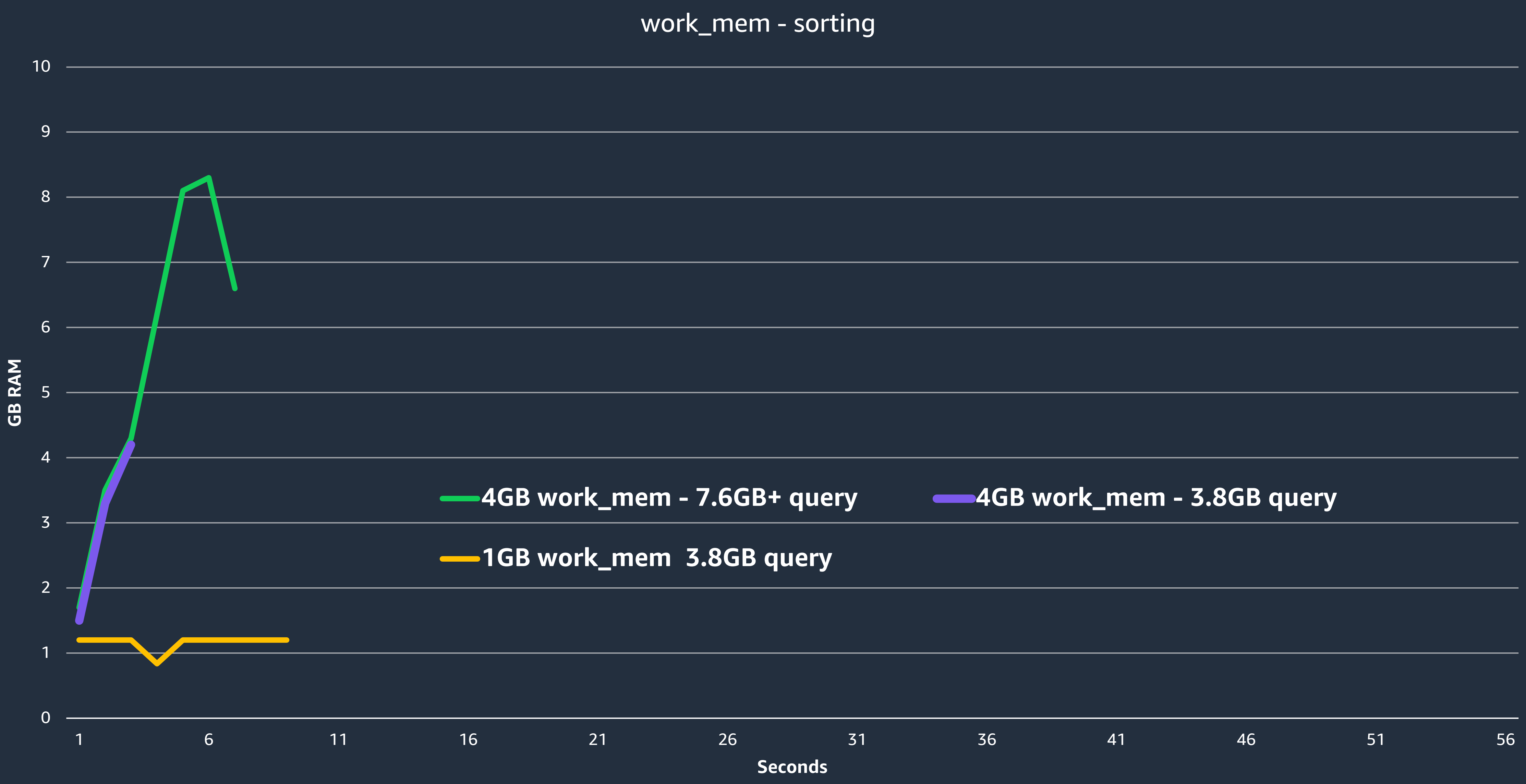
```
                                                                   QUERY PLAN
------------------------------------------------------------------------------------------------------------------------------------------------
-
 Sort  (cost=7219224.79..7228724.79 rows=3800000 width=4116) (actual time=2766.682..3408.706 rows=3800000 loops=1)
   Sort Key: (((random() * '10000000000'::double precision))::bigint)
   Sort Method: quicksort  Memory: 4128025kB
   ->  Function Scan on generate_series mykey  (cost=0.00..76000.00 rows=3800000 width=4116) (actual time=194.365..506.965 rows=3800000 loops=1)
 Planning Time: 0.047 ms
 Execution Time: 3825.965 ms
(6 rows)
```


work_mem - sorting
GB RAM
10
9
8
7
6
5
4
3
2
1
0
4GB work_mem - 3.8GB query
1GB work_mem 3.8GB query
1
6
11
16
21
26
31
36
41
46
51
56
Seconds

work_mem

```
postgres=# set work_mem TO '4 GB’;

explain analyze select s1.filler, s2.filler,s1.mykey, s2.mykey, s1.scratch, s2.scratch from
(select mykey::bigint, (random()*1000000000)::bigint as scratch ,
repeat('X', 1024)::char(1024) filler from generate_series(1,3800000)
as mykey order by scratch) s1,
(select mykey::bigint, (random()*10000000000)::bigint as scratch ,
repeat('X', 1024)::char(1024) filler from generate_series(1,3800000)
as mykey order by scratch) s2
where s1.scratch=s2.scratch order by s1.mykey;


                                                                             QUERY PLAN
------------------------------------------------------------------------------------------------------------------------------------------------
---------------------
 Sort  (cost=523467712452.33..523648212452.33 rows=72200000000 width=8232) (actual time=6851.977..6852.172 rows=1494 loops=1)
   Sort Key: ((mykey.mykey)::bigint)
   Sort Method: quicksort  Memory: 3153kB
   ->  Merge Join  (cost=14438449.59..1097542949.59 rows=72200000000 width=8232) (actual time=5558.904..6850.847 rows=1494 loops=1)
         Merge Cond: ((((random() * '1000000000'::double precision))::bigint) = (((random() * '10000000000'::double precision))::bigint))
         ->  Sort  (cost=7219224.79..7228724.79 rows=3800000 width=4116) (actual time=2785.745..3524.489 rows=3800000 loops=1)
               Sort Key: (((random() * '1000000000'::double precision))::bigint)
               Sort Method: quicksort  Memory: 4128025kB
               ->  Function Scan on generate_series mykey  (cost=0.00..76000.00 rows=3800000 width=4116) (actual time=194.222..515.986
rows=3800000 loops=1)
         ->  Materialize  (cost=7219224.79..7276224.79 rows=3800000 width=4116) (actual time=2772.383..2961.007 rows=380357 loops=1)
               ->  Sort  (cost=7219224.79..7228724.79 rows=3800000 width=4116) (actual time=2772.377..2867.394 rows=380353 loops=1)
                     Sort Key: (((random() * '10000000000'::double precision))::bigint)
                     Sort Method: quicksort  Memory: 4128025kB
                     ->  Function Scan on generate_series mykey_1  (cost=0.00..76000.00 rows=3800000 width=4116) (actual time=198.688..511.083
rows=3800000 loops=1)
 Planning Time: 0.098 ms
 Execution Time: 7511.377 ms
```

~3.8GB


work_mem - sorting
GB RAM
10
9
8
7
6
5
4
3
2
1
0
1
6
11
16
21
26
31
36
41
46
51
56
Seconds
4GB work_mem - 7.6GB+ query
4GB work_mem - 3.8GB query
1GB work_mem 3.8GB query

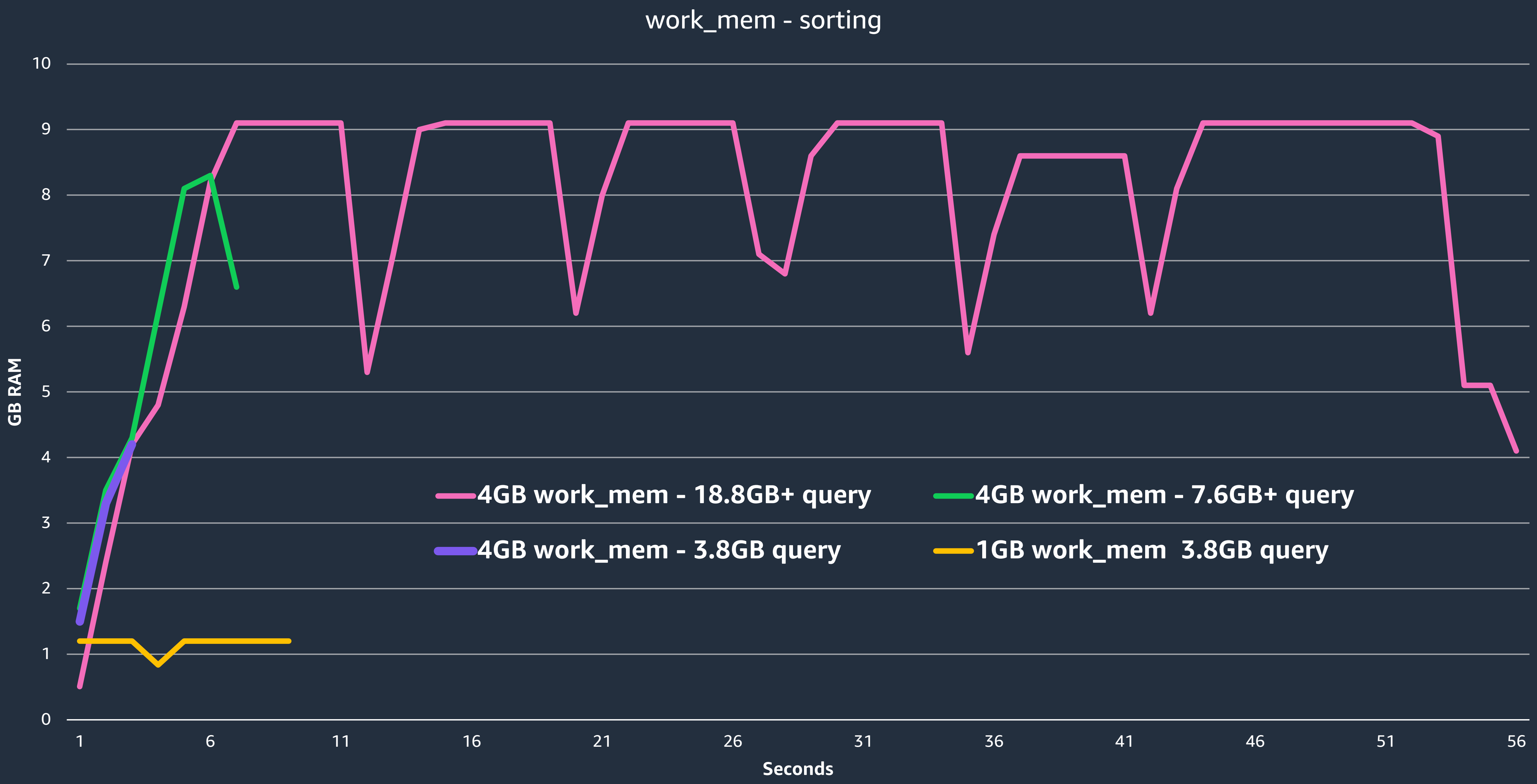
work_mem

```
postgres=# set work_mem TO '4 GB’;

postgres=# explain analyze
select s1.filler, s2.filler,s1.mykey, s2.mykey, s1.scratch, s2.scratch from
(select mykey::bigint, (random()*1000000000)::bigint as scratch ,
repeat('X', 1024)::char(1024) filler from generate_series(1,18800000)
as mykey order by scratch) s1,
(select mykey::bigint, (random()*10000000000)::bigint as scratch ,
repeat('X', 1024)::char(1024) filler from generate_series(1,3800000)
as mykey order by scratch) s2
where s1.scratch=s2.scratch order by s1.mykey;
```

~18.8GB

~3.8GB

```
                                                                                 QUERY PLAN
-------------------------------------------------------------------------------------------------------------------------------------------------
-------------------------
 Sort  (cost=3853846009559.71..3854739009559.71 rows=357200000000 width=8232) (actual time=52689.953..52690.952 rows=7184 loops=1)
   Sort Key: ((mykey_1.mykey)::bigint)
   Sort Method: quicksort  Memory: 15122kB
   ->  Merge Join  (cost=43152211.85..5401444211.85 rows=357200000000 width=8232) (actual time=42991.198..52679.295 rows=7184 loops=1)
         Merge Cond: ((((random() * '10000000000'::double precision))::bigint) = (((random() * '1000000000'::double precision))::bigint))
         ->  Sort  (cost=7219224.79..7228724.79 rows=3800000 width=4116) (actual time=2771.496..2877.777 rows=380382 loops=1)
               Sort Key: (((random() * '10000000000'::double precision))::bigint)
               Sort Method: quicksort  Memory: 4128025kB
               ->  Function Scan on generate_series mykey  (cost=0.00..76000.00 rows=3800000 width=4116) (actual time=194.404..516.400
rows=3800000 loops=1)
         ->  Materialize  (cost=35932987.06..36214987.06 rows=18800000 width=4116) (actual time=40218.833..48614.865 rows=18800001 loops=1)
               ->  Sort  (cost=35932987.06..35979987.06 rows=18800000 width=4116) (actual time=40218.824..46815.209 rows=18800000 loops=1)
                     Sort Key: (((random() * '1000000000'::double precision))::bigint)
                     Sort Method: external merge  Disk: 19388680kB
                     ->  Function Scan on generate_series mykey_1  (cost=0.00..376000.00 rows=18800000 width=4116) (actual
time=960.638..2593.178 rows=18800000 loops=1)
 Planning Time: 0.102 ms
 Execution Time: 55724.412 ms
```


work_mem - sorting
GB RAM
10
9
8
7
6
5
4
3
2
1
0
1
6
11
16
21
26
31
36
41
46
51
56
Seconds
4GB work_mem - 18.8GB+ query
4GB work_mem - 7.6GB+ query
4GB work_mem - 3.8GB query
1GB work_mem 3.8GB query

hash_mem_multiplier

```
postgres=# set hash_mem_multiplier = 2 ; set work_mem='1 GB';

QUERY PLAN
--------------------------------------------------------------------------------------------------------------------------------------------------------------
 Hash Join  (cost=2083766.00..5865195158.01 rows=167200000000 width=8232) (actual time=5393.748..27494.124 rows=3338 loops=1)
   Hash Cond: (s1.scratch = s2.scratch)
   ->  Subquery Scan on s1  (cost=0.00..264000.00 rows=8800000 width=4116) (actual time=363.214..2759.362 rows=8800000 loops=1)
         ->  Function Scan on generate_series mykey  (cost=0.00..176000.00 rows=8800000 width=4116) (actual time=363.213..1686.157 rows=8800000 loops=1)
   ->  Hash  (cost=114000.00..114000.00 rows=3800000 width=4116) (actual time=5023.109..5023.111 rows=3800000 loops=1)
         Buckets: 524288  Batches: 8  Memory Usage: 505770kB
         ->  Subquery Scan on s2  (cost=0.00..114000.00 rows=3800000 width=4116) (actual time=161.237..1005.102 rows=3800000 loops=1)
               ->  Function Scan on generate_series mykey_1  (cost=0.00..76000.00 rows=3800000 width=4116) (actual time=161.236..625.366 rows=3800000 loops=1)(10 rows)
 Execution Time: 27597.763 ms

postgres=# set hash_mem_multiplier = 1 ; set work_mem=‘2 GB';
QUERY PLAN
--------------------------------------------------------------------------------------------------------------------------------------------------------------
 Hash Join  (cost=2083766.00..5865195158.01 rows=167200000000 width=8232) (actual time=4941.337..27460.203 rows=3330 loops=1)
   Hash Cond: (s1.scratch = s2.scratch)
   ->  Subquery Scan on s1  (cost=0.00..264000.00 rows=8800000 width=4116) (actual time=308.103..2682.171 rows=8800000 loops=1)
         ->  Function Scan on generate_series mykey  (cost=0.00..176000.00 rows=8800000 width=4116) (actual time=308.102..1627.632 rows=8800000 loops=1)
   ->  Hash  (cost=114000.00..114000.00 rows=3800000 width=4116) (actual time=4632.871..4632.873 rows=3800000 loops=1)
         Buckets: 524288  Batches: 8  Memory Usage: 506081kB
         ->  Subquery Scan on s2  (cost=0.00..114000.00 rows=3800000 width=4116) (actual time=150.922..917.699 rows=3800000 loops=1)
               ->  Function Scan on generate_series mykey_1  (cost=0.00..76000.00 rows=3800000 width=4116) (actual time=150.921..573.278 rows=3800000 loops=1)
Execution Time: 27571.422 ms

 postgres=# set hash_mem_multiplier = 8 ; set work_mem=‘2 GB';
                                                                          QUERY PLAN
--------------------------------------------------------------------------------------------------------------------------------------------------------------
 Hash Join  (cost=161500.01..5852447500.01 rows=167200000000 width=8232) (actual time=3166.564..6799.798 rows=3367 loops=1)
   Hash Cond: ((((random() * '1000000000'::double precision))::bigint) = s2.scratch)
   ->  Function Scan on generate_series mykey  (cost=0.00..176000.00 rows=8800000 width=4116) (actual time=312.749..1225.873 rows=8800000 loops=1)
   ->  Hash  (cost=114000.00..114000.00 rows=3800000 width=4116) (actual time=2846.645..2846.647 rows=3800000 loops=1)
         Buckets: 4194304  Batches: 1  Memory Usage: 4040581kB
         ->  Subquery Scan on s2  (cost=0.00..114000.00 rows=3800000 width=4116) (actual time=162.493..905.717 rows=3800000 loops=1)
               ->  Function Scan on generate_series mykey_1  (cost=0.00..76000.00 rows=3800000 width=4116) (actual time=162.491..566.310 rows=3800000 loops=1)
Execution Time: 7234.249 ms
```

Almost a 75% reduction in execution time

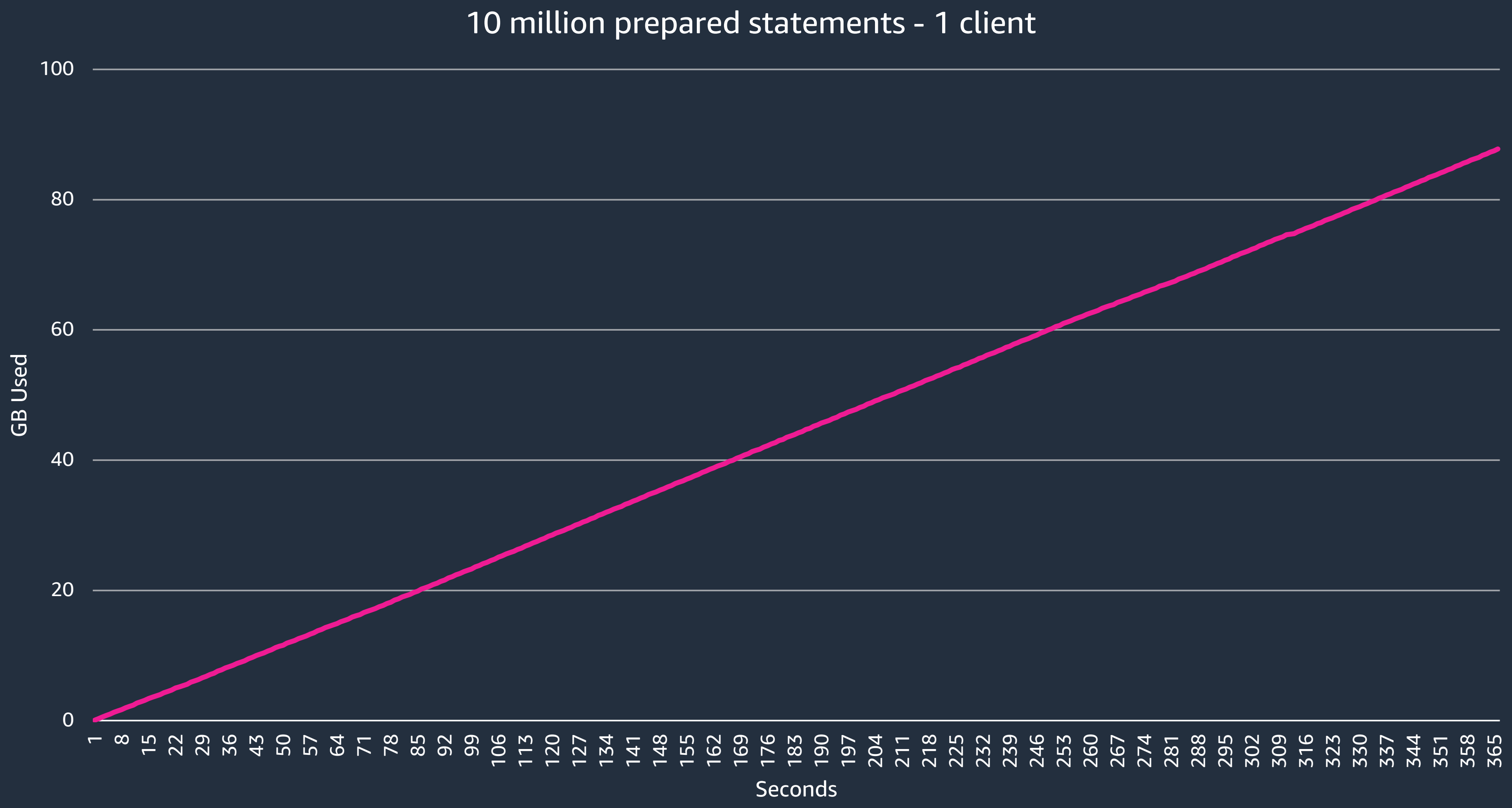
Per Session Item

prepared statements

```
PREPARE sbtestplan (int) AS
    SELECT * FROM sbtest1 WHERE id=$1 ;
EXECUTE sbtestplan(1);
```

```
sbtest=# SELECT name, ident, level, total_bytes
      FROM pg_backend_memory_contexts where name ='CachedPlanSource';
       name       |                  ident                  | level | total_bytes
------------------+-----------------------------------------+-------+-------------
 CachedPlanSource | PREPARE sbtestplan3 (int) AS           +|     2 |        4096
                  |     SELECT * FROM sbtest3 WHERE id=$1 ; |       |
 CachedPlanSource | PREPARE sbtestplan2 (int) AS           +|     2 |        4096
                  |     SELECT * FROM sbtest2 WHERE id=$1 ; |       |
 CachedPlanSource | PREPARE sbtestplan (int) AS            +|     2 |        4096
                  |     SELECT * FROM sbtest1 WHERE id=$1 ; |       |
```

prepared statements – 10 Million

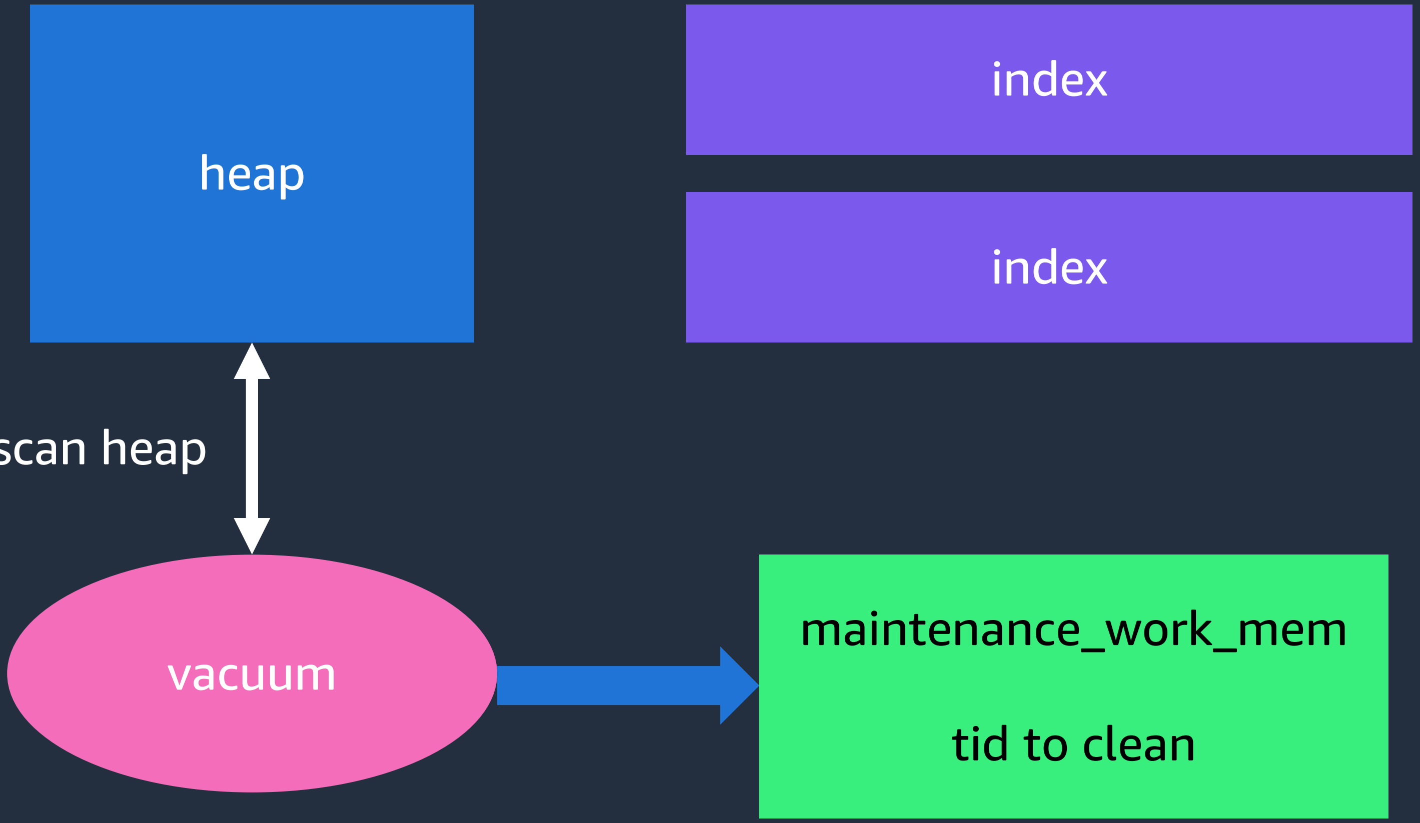


Maintenance

maintenance_work_mem – Index builds

```
sbtest=# set maintenance_work_mem = '1 GB';
                                                 ^
sbtest=# create index foo on sbtest1 (id,c,pad,k desc) ;    ~1GB
CREATE INDEX
Time: 45803.577 ms (00:45.804)

sbtest=# set maintenance_work_mem = '2 GB';

sbtest=# create index foo on sbtest1 (id,c,pad,k desc) ;    ~2GB
CREATE INDEX
Time: 45904.760 ms (00:45.905)


sbtest=# set maintenance_work_mem = '4 GB';

sbtest=# create index foo on sbtest1 (id,c,pad,k desc) ;    ~3GB
CREATE INDEX
Time: 46081.414 ms (00:46.081)
```

maintenance_work_mem – how vacuum works



heap

index

index

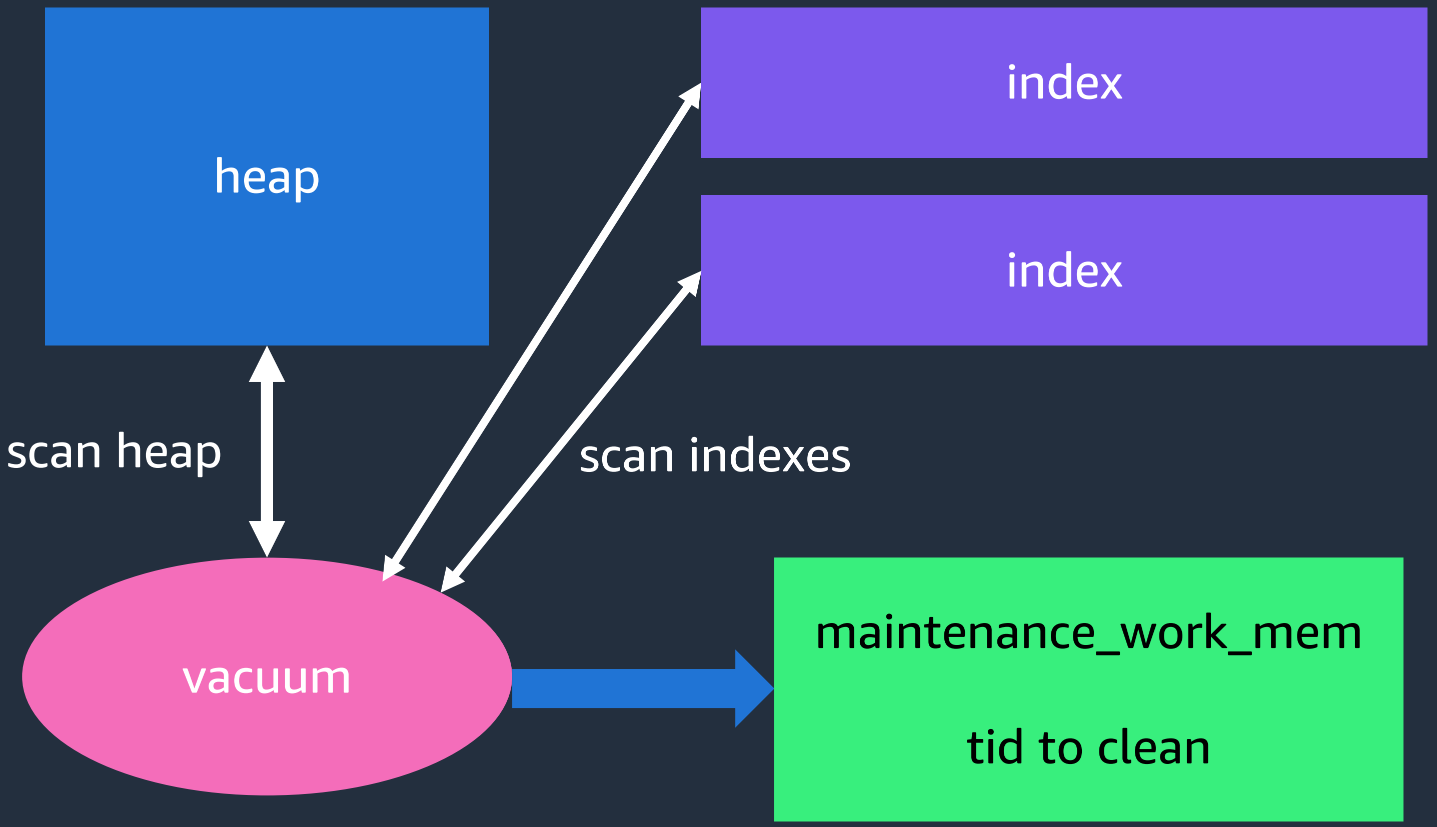
scan heap

vacuum

maintenance_work_mem

tid to clean

maintenance_work_mem – how vacuum works



maintenance_work_mem - vacuum (Version 12-16)

```
update sbtest1 set k=k+1; UPDATE all 500 M rows in the table
UPDATE 500000000

sbtest=# set maintenance_work_mem='200 MB';

benchdb=> vacuum (verbose) sbtest1;
INFO:  vacuuming "benchdb.public.sbtest1"
INFO:  launched 1 parallel vacuum worker for index vacuuming (planned: 1)
… 13 more lines of index vacuuming
INFO:  launched 1 parallel vacuum worker for index vacuuming (planned: 1)
INFO:  finished vacuuming "benchdb.public.sbtest1": index scans: 15
pages: 0 removed, 27375047 remain, 27375047 scanned (100.00% of total)
tuples: 500000000 removed, 500000000 remain, 0 are dead but not yet removable
removable cutoff: 119558641, which was 91 XIDs old when operation ended
new relfrozenxid: 119558641, which is 50002327 XIDs ahead of previous value
frozen: 27194801 pages from table (99.34% of total) had 500000000 tuples frozen
index scan needed: 15300869 pages from table (55.89% of total) had 500000000 dead item identifiers removed
index "sbtest1_pkey": pages: 2741864 in total, 0 newly deleted, 0 currently deleted, 0 reusable
index "k_1": pages: 2405158 in total, 0 newly deleted, 0 currently deleted, 0 reusable
I/O timings: read: 22363.650 ms, write: 76810.339 ms
avg read rate: 49.223 MB/s, avg write rate: 194.936 MB/s
buffer usage: 118567797 hits, 28698356 misses, 113651873 dirtied
WAL usage: 183521738 records, 113651894 full page images, 313681897335 bytes
system usage: CPU: user: 3756.62 s, system: 99.85 s, elapsed: 4554.85 s
```

Memory use – 6 bytes per dead item in heap (4 for block, 2 for offset)

500,000,000 x 6 ≈ 2861 MB needed

maintenance_work_mem - vacuum (Version 12-16)

```
update sbtest1 set k=k+1; UPDATE all 500 M rows in the table
UPDATE 500000000

sbtest=# set maintenance_work_mem=‘3 GB’;
```

Only use max of 1GB

```
benchdb=> vacuum (verbose) sbtest1;
INFO:  vacuuming "benchdb.public.sbtest1"
INFO:  launched 1 parallel vacuum worker for index vacuuming (planned: 1)
INFO:  launched 1 parallel vacuum worker for index vacuuming (planned: 1)
INFO:  launched 1 parallel vacuum worker for index vacuuming (planned: 1)
INFO:  finished vacuuming "benchdb.public.sbtest1": index scans: 3
pages: 0 removed, 27375047 remain, 27367689 scanned (99.97% of total)
tuples: 500000000 removed, 500134706 remain, 0 are dead but not yet removable
removable cutoff: 119560957, which was 89 XIDs old when operation ended
new relfrozenxid: 119560957, which is 568 XIDs ahead of previous value
frozen: 26637843 pages from table (97.31% of total) had 500000000 tuples frozen
index scan needed: 27197885 pages from table (99.35% of total) had 500000000 dead item identifiers removed
index "sbtest1_pkey": pages: 2741864 in total, 0 newly deleted, 0 currently deleted, 0 reusable
index "k_1": pages: 2405158 in total, 0 newly deleted, 0 currently deleted, 0 reusable
I/O timings: read: 29066.179 ms, write: 79510.303 ms
avg read rate: 56.429 MB/s, avg write rate: 121.372 MB/s
buffer usage: 64896769 hits, 32488492 misses, 69879342 dirtied
WAL usage: 151000537 records, 69879380 full page images, 209552080351 bytes
system usage: CPU: user: 3656.90 s, system: 117.19 s, elapsed: 4498.00 s
```

Memory use – 6 bytes per dead item in heap (4 for block, 2 for offset)

`500,000,000 x 6 ≈ 2861 MB needed`

maintenance_work_mem - vacuum (Version 17+)

```
update sbtest1 set k=k+1; UPDATE all 500 M rows in the table
UPDATE 500000000

sbtest=# set maintenance_work_mem=‘200 MB';

benchdb=> vacuum (verbose) sbtest1;
INFO:  vacuuming "benchdb.public.sbtest1"
INFO:  launched 1 parallel vacuum worker for index vacuuming (planned: 1)
INFO:  launched 1 parallel vacuum worker for index vacuuming (planned: 1)
INFO:  launched 1 parallel vacuum worker for index vacuuming (planned: 1)
INFO:  finished vacuuming "benchdb.public.sbtest1": index scans: 3
pages: 0 removed, 27281773 remain, 27281773 scanned (100.00% of total)
tuples: 500000000 removed, 500000000 remain, 0 are dead but not yet removable
removable cutoff: 81359906, which was 60 XIDs old when operation ended
new relfrozenxid: 81359906, which is 50000640 XIDs ahead of previous value
frozen: 27094240 pages from table (99.31% of total) had 500000000 tuples frozen
index scan needed: 15247188 pages from table (55.89% of total) had 500000000 dead item identifiers removed
index "sbtest1_pkey": pages: 2741864 in total, 0 newly deleted, 0 currently deleted, 0 reusable
index "k_1": pages: 2405158 in total, 0 newly deleted, 0 currently deleted, 0 reusable
I/O timings: read: 22918.753 ms, write: 65940.679 ms
avg read rate: 73.603 MB/s, avg write rate: 150.222 MB/s
buffer usage: 57030696 hits, 28230433 misses, 57617361 dirtied
WAL usage: 112180070 records, 57617381 full page images, 163701031507 bytes
system usage: CPU: user: 2692.02 s, system: 97.26 s, elapsed: 2996.46 s
```

Memory use – 4 bytes per block with dead tuple plus bitmap for offset

`Used between 400 and 600 MB (5+X reduction in memory)`

maintenance_work_mem - vacuum (Version 12-16)

```
update sbtest1 set k=k+1  where (ctid::text::point)[1]::bigint < 5; UPDATE all blocks but only some rows
UPDATE 58634114

sbtest=# set maintenance_work_mem=‘200 MB’;

benchdb=> vacuum (verbose) sbtest1;
INFO:  vacuuming "benchdb.public.sbtest1"
INFO:  launched 1 parallel vacuum worker for index vacuuming (planned: 1)
INFO:  launched 1 parallel vacuum worker for index vacuuming (planned: 1)
INFO:  finished vacuuming "benchdb.public.sbtest1": index scans: 2
pages: 0 removed, 27375047 remain, 27310183 scanned (99.76% of total)
tuples: 58634114 removed, 501167394 remain, 0 are dead but not yet removable
removable cutoff: 119558905, which was 55 XIDs old when operation ended
new relfrozenxid: 119558905, which is 264 XIDs ahead of previous value
frozen: 8737533 pages from table (31.92% of total) had 58634114 tuples frozen
index scan needed: 20162834 pages from table (73.65% of total) had 58634114 dead item identifiers removed
index "sbtest1_pkey": pages: 2741864 in total, 0 newly deleted, 0 currently deleted, 0 reusable
index "k_1": pages: 2405158 in total, 0 newly deleted, 0 currently deleted, 0 reusable
I/O timings: read: 20886.445 ms, write: 45927.957 ms
avg read rate: 73.424 MB/s, avg write rate: 144.178 MB/s
buffer usage: 59121573 hits, 25966600 misses, 50988868 dirtied
WAL usage: 100700060 records, 50988886 full page images, 138744674856 bytes
system usage: CPU: user: 2428.40 s, system: 77.22 s, elapsed: 2762.91 s
```

Memory use – 6 bytes per dead item in heap (4 for block, 2 for offset)

`58,634,114 x 6 ≈ 224 MB needed`

maintenance_work_mem - vacuum (Version 17+)

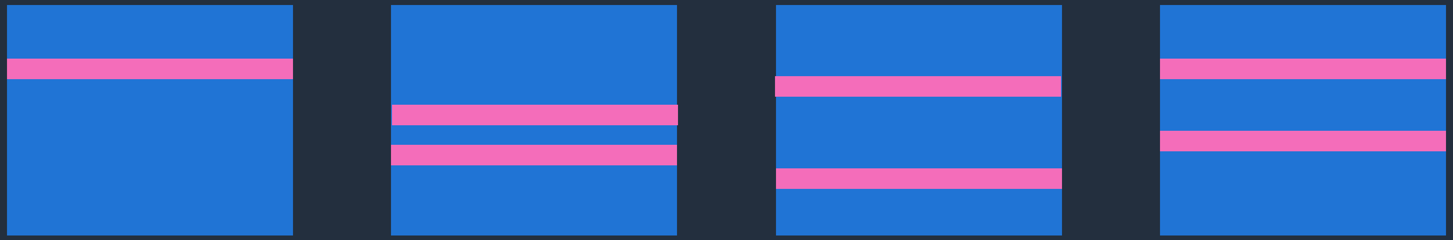
```
update sbtest1 set k=k+1  where (ctid::text::point)[1]::bigint < 5; UPDATE all blocks but only some rows
UPDATE 59461887

sbtest=# set maintenance_work_mem='200 MB';

benchdb=> vacuum (verbose) sbtest1;
INFO:  vacuuming "benchdb.public.sbtest1"
INFO:  launched 1 parallel vacuum worker for index vacuuming (planned: 1)
INFO:  launched 1 parallel vacuum worker for index vacuuming (planned: 1)
INFO:  launched 1 parallel vacuum worker for index vacuuming (planned: 1)
INFO:  finished vacuuming "benchdb.public.sbtest1": index scans: 3
pages: 0 removed, 27281773 remain, 27203678 scanned (99.71% of total)
tuples: 59461887 removed, 501403118 remain, 0 are dead but not yet removable
removable cutoff: 81360196, which was 53 XIDs old when operation ended
new relfrozenxid: 81360196, which is 290 XIDs ahead of previous value
frozen: 9163577 pages from table (33.59% of total) had 59461887 tuples frozen
index scan needed: 20599698 pages from table (75.51% of total) had 59461887 dead item identifiers removed
index "sbtest1_pkey": pages: 2741864 in total, 0 newly deleted, 0 currently deleted, 0 reusable
index "k_1": pages: 2405158 in total, 0 newly deleted, 0 currently deleted, 0 reusable
I/O timings: read: 17411.841 ms, write: 43079.573 ms
avg read rate: 77.004 MB/s, avg write rate: 161.922 MB/s
buffer usage: 64207953 hits, 26250232 misses, 55198523 dirtied
WAL usage: 97593961 records, 55198539 full page images, 145589683624 bytes
system usage: CPU: user: 2385.56 s, system: 75.04 s, elapsed: 2663.24 s
```

Memory use – 4 bytes per block with dead item plus bitmap for offset

Used between 400 and 600 MB (i.e. the same as last run with less updates)

Memory Use – Version 16 vs 17

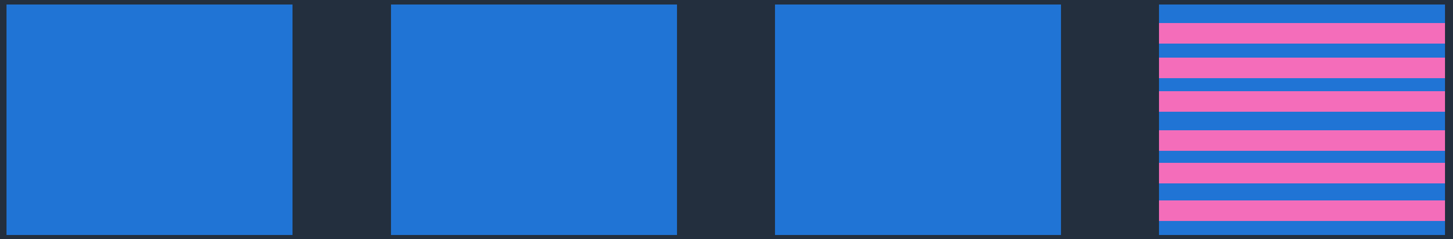
Workload A



V16 – 6 x 6bytes

V17 – 4 x 4bytes + bitmap for offset

Workload B



V16 – 6 x 6bytes

V17 – 1 x 4bytes + bitmap for offset

4 blocks
6 dead tuples

maintenance_work_mem - vacuum (Version 17+)

```
benchdb=> update sbtest1 set k = k + 1 where (ctid::text::point)[1]::bigint < 5;
UPDATE 166666668

sbtest=# set maintenance_work_mem=‘1 GB’;

benchdb=> vacuum (verbose) sbtest1;
INFO:  vacuuming "benchdb.public.sbtest1"
INFO:  launched 1 parallel vacuum worker for index vacuuming (planned: 1)
… 3 more lines of index vacuuming
INFO:  launched 1 parallel vacuum worker for index vacuuming (planned: 1)
INFO:  finished vacuuming "benchdb.public.sbtest1": index scans: 5
pages: 0 removed, 166666667 remain, 166666667 scanned (100.00% of total)
tuples: 166666668 removed, 500000000 remain, 0 are dead but not yet removable
removable cutoff: 81363963, which was 166 XIDs old when operation ended
new relfrozenxid: 81363963, which is 1294 XIDs ahead of previous value
frozen: 166666667 pages from table (100.00% of total) had 500000000 tuples frozen
index scan needed: 166666667 pages from table (100.00% of total) had 693018132 dead item identifiers removed
index "sbtest1_pkey": pages: 2741898 in total, 0 newly deleted, 0 currently deleted, 0 reusable
index "k_1": pages: 2405132 in total, 0 newly deleted, 0 currently deleted, 0 reusable
I/O timings: read: 1904853.515 ms, write: 694137.509 ms
avg read rate: 296.413 MB/s, avg write rate: 336.211 MB/s
buffer usage: 211128150 hits, 314622418 misses, 356864387 dirtied
WAL usage: 690192617 records, 356864414 full page images, 241152922879 bytes
system usage: CPU: user: 5389.92 s, system: 1597.28 s, elapsed: 8292.43 s
```

Memory use – 4 bytes per block with dead tuple plus bitmap for offset

`Used between 4 and 5 GB (10 % fillfactor – a lot more block)`

maintenance_work_mem - vacuum (Version 17+)

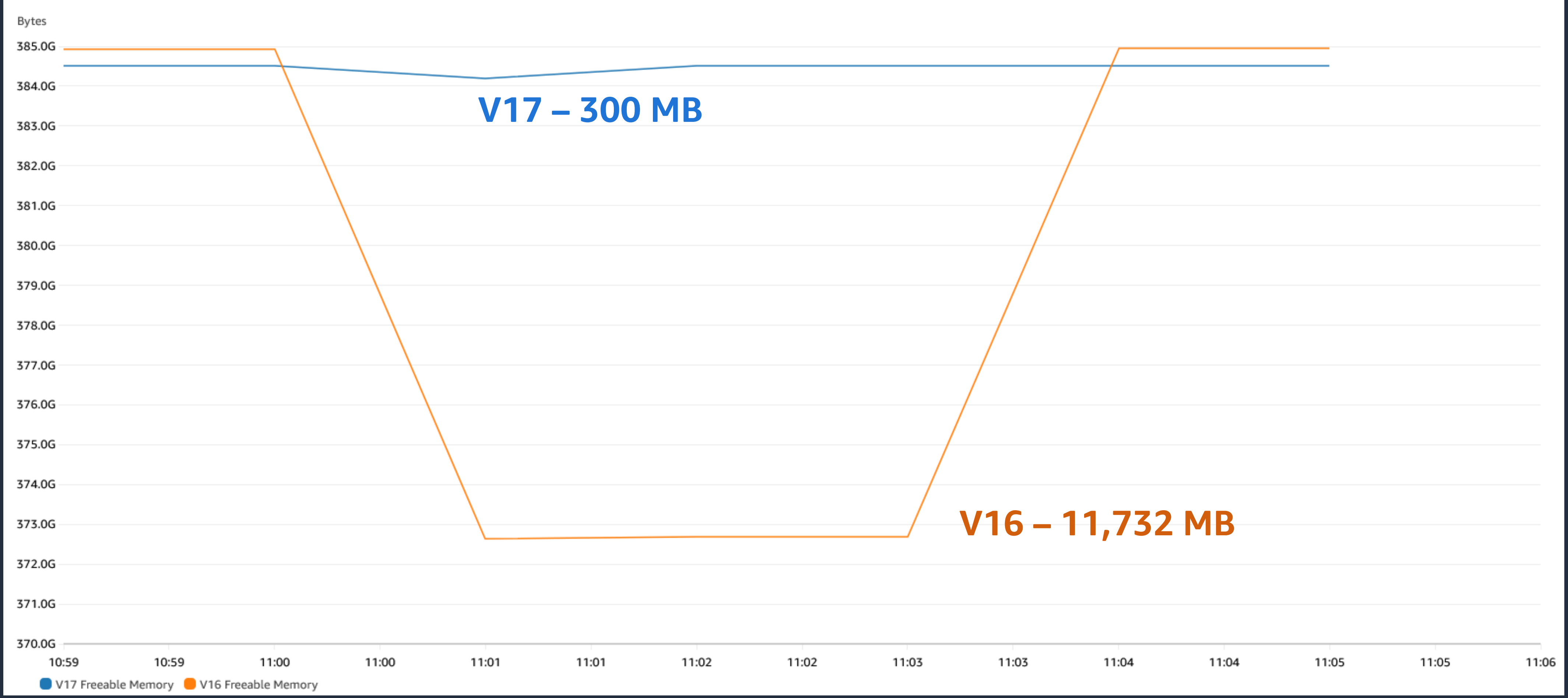
```
benchdb=> update sbtest1 set k = k + 1 where (ctid::text::point)[1]::bigint < 5; Fillfactor of 10% so lots many blocks
UPDATE 166666668

sbtest=# set maintenance_work_mem='5 GB';

benchdb=>  vacuum (verbose) sbtest1;
INFO:  vacuuming "benchdb.public.sbtest1"
INFO:  launched 1 parallel vacuum worker for index vacuuming (planned: 1)
INFO:  finished vacuuming "benchdb.public.sbtest1": index scans: 1
pages: 0 removed, 166666667 remain, 166666667 scanned (100.00% of total)
tuples: 166666667 removed, 500000000 remain, 0 are dead but not yet removable
removable cutoff: 81368431, which was 151 XIDs old when operation ended
new relfrozenxid: 81368431, which is 258 XIDs ahead of previous value
frozen: 166666667 pages from table (100.00% of total) had 166666667 tuples frozen
index scan needed: 166666667 pages from table (100.00% of total) had 166666667 dead item identifiers removed
index "sbtest1_pkey": pages: 2741898 in total, 0 newly deleted, 0 currently deleted, 0 reusable
index "k_1": pages: 2405132 in total, 0 newly deleted, 0 currently deleted, 0 reusable
I/O timings: read: 1650457.241 ms, write: 688257.723 ms
avg read rate: 321.438 MB/s, avg write rate: 349.887 MB/s
buffer usage: 194261510 hits, 310946909 misses, 338467339 dirtied
WAL usage: 671795571 records, 338467370 full page images, 198971096165 bytes
system usage: CPU: user: 4940.17 s, system: 1591.97 s, elapsed: 7557.50 s
```

Memory use – 4 bytes per block with dead tuple plus bitmap for offset

Used between 4 and 5 GB (10 % fillfactor - a lot more block)

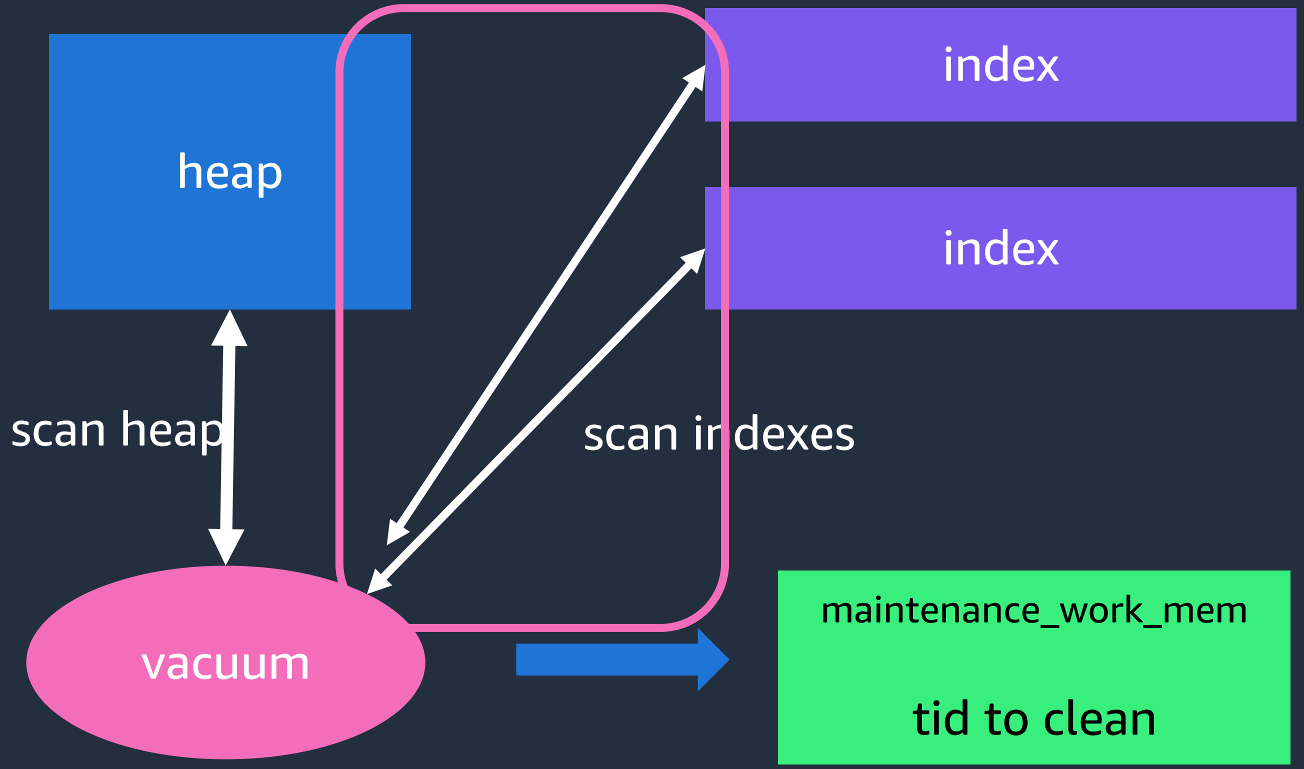
Vacuum memory allocation Version 16 vs 17

Running 50 table vacuums in parallel with 1 GB maintance_work_mem – 5M Row table



maintenance_work_mem - vacuum parallel

```
sbtest=# set maintenance_work_mem='1 GB’;


sbtest=# vacuum (verbose) sbtest1;
INFO:  vacuuming "sbtest.public.sbtest1"
INFO:  launched 2 parallel vacuum workers for index vacuuming (planned: 2)
```



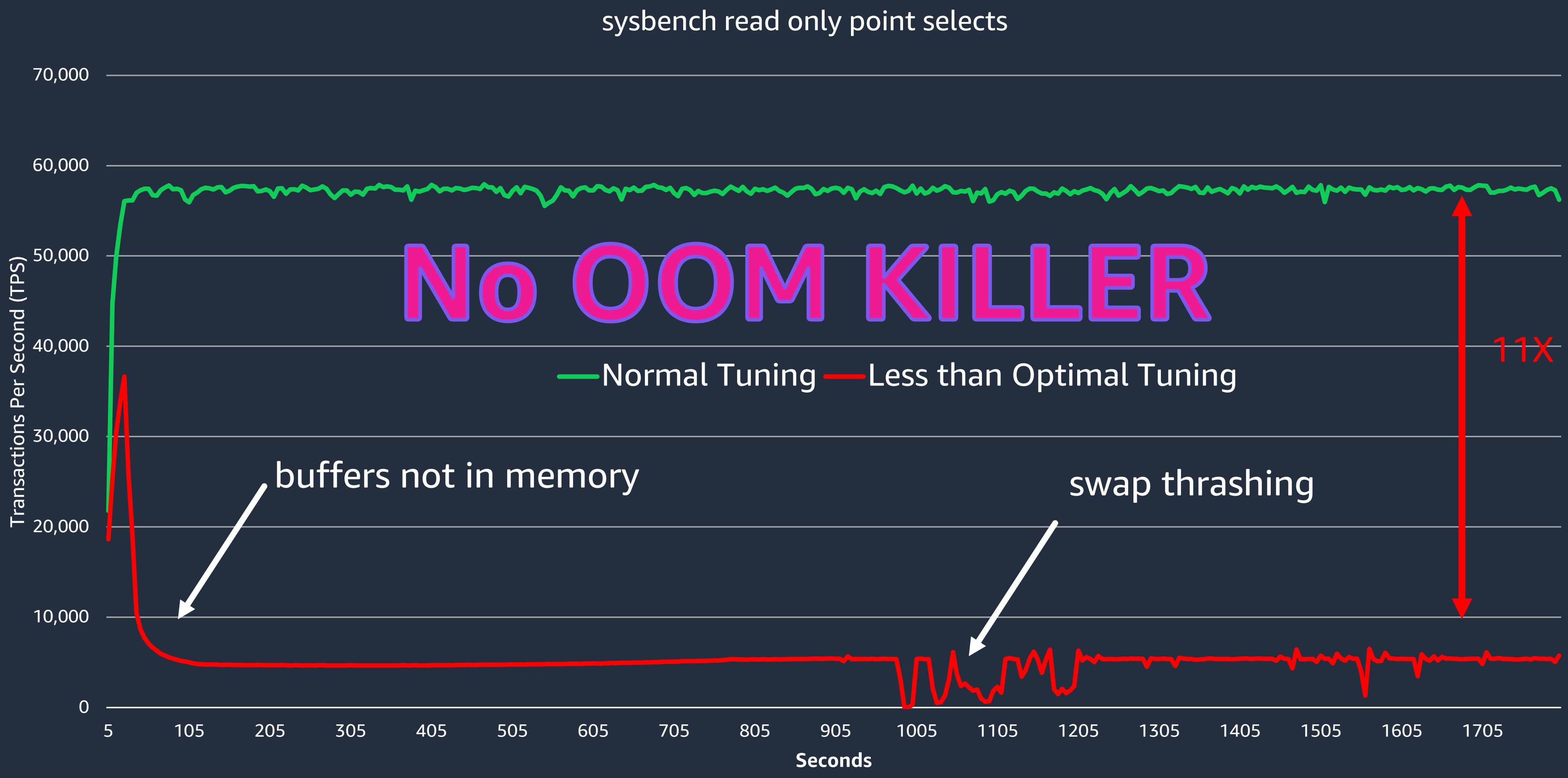
autovacuum_work_mem

autovacuum_work_mem X autovacuum_max_workers = possible memory used

Example

1 GB X 30 workers = 30 GB possible memory used by autovacuum

Why you should care



No OOM KILLER

buffers not in memory

swap thrashing

Thank you!

Grant McAlister