Fast-path locking improvements in PG18

Tomas Vondra <tomas@vondra.me> / https://vondra.me

pgconf.eu 2025, October 21-24, Riga

Microsoft

Agenda

- Why this improvement?
- A bit of history (PG 9.2)
- PG 18 improvements
- Trade-offs
- Challenges
- Future

Why was I looking into this?

- end of 2023 (?)
- poor performance reported by a customer
- partitioned table (handful of partitions)
- upgraded to CPU with more cores
  - slower cores but ~2x the core count
  - expected better performance
- the opposite happened
  - much slower (with concurrency)

example workload

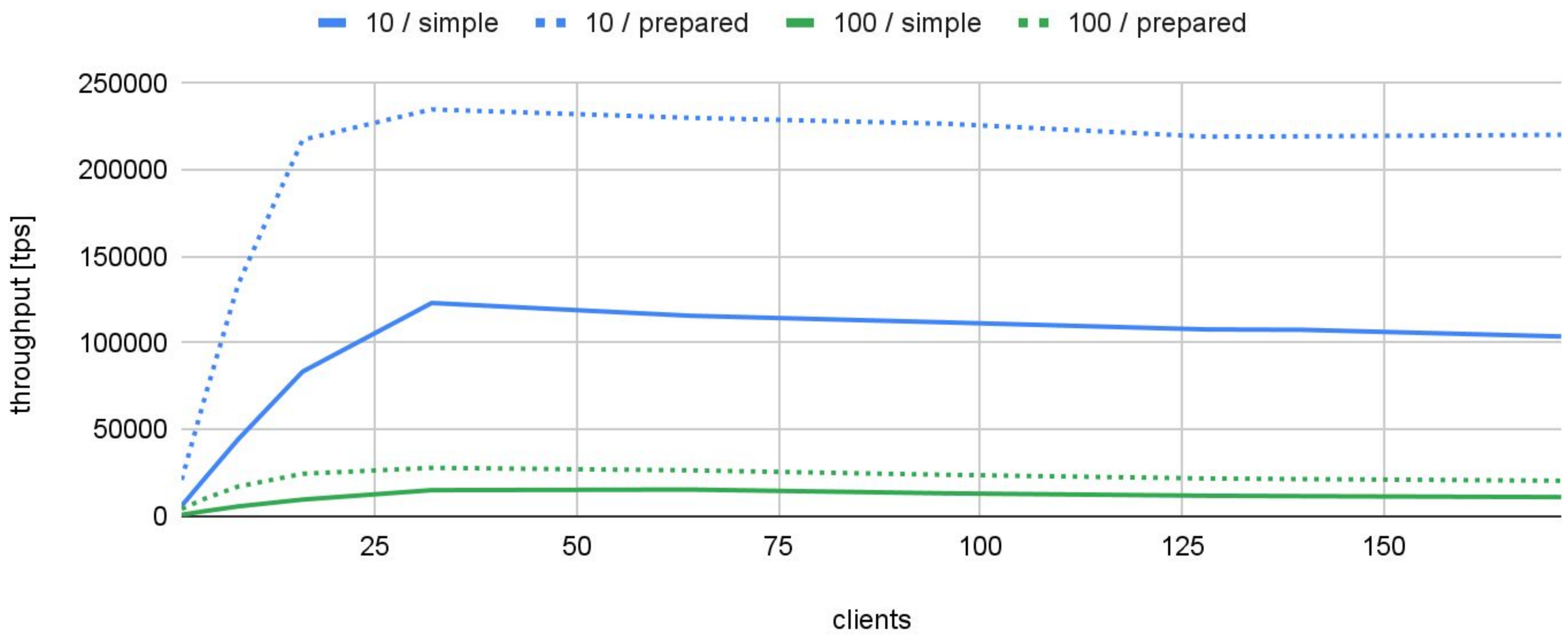
```
# pgbench -i -s 1 --partitions 10

ALTER TABLE pgbench_accounts ADD COLUMN aid_new INT;
UPDATE pgbench_accounts SET aid_new = aid;
CREATE INDEX ON pgbench_accounts (aid_new);
VACUUM FULL pgbench_accounts;

\set aid random(1, 100000 * :scale)
SELECT * FROM pgbench_accounts pa
         JOIN pgbench_branches pb ON (pa.bid = pb.bid)
 WHERE pa.aid_new = :aid
```

EXPLAIN

```
                                    QUERY PLAN
-------------------------------------------------------------------------------------------------------------------------------
 Hash Join  (cost=1.52..34.41 rows=10 width=465)
   Hash Cond: (pa.bid = pb.bid)
   ->  Append  (cost=0.29..33.15 rows=10 width=101)
       ->  Index Scan using pgbench_accounts_1_aid_parent_idx on pgbench_accounts_1 pa_1 (cost=0.29..3.31 rows=1 width=101)
             Index Cond: (aid_new = 3489734)
       ->  Index Scan using pgbench_accounts_2_aid_parent_idx on pgbench_accounts_2 pa_2 (cost=0.29..3.31 rows=1 width=101)
             Index Cond: (aid_new = 3489734)
       ->  Index Scan using pgbench_accounts_3_aid_parent_idx on pgbench_accounts_3 pa_3 (cost=0.29..3.31 rows=1 width=101)
             Index Cond: (aid_new = 3489734)
       ->  Index Scan using pgbench_accounts_4_aid_parent_idx on pgbench_accounts_4 pa_4 (cost=0.29..3.31 rows=1 width=101)
             Index Cond: (aid_new = 3489734)
       -> ...
   ->  Hash  (cost=1.10..1.10 rows=10 width=364)
       ->  Seq Scan on pgbench_branches pb  (cost=0.00..1.10 rows=10 width=364)
```

throughput with partitions

AMD EPYC 9V74 80-Core Processor



What could be causing this?

- Clearly a concurrency issue.
- Something is contended, but what?
- Let's jump to "obvious" conclusions!

```
/* lwlock.h */
#define LOG2_NUM_LOCK_PARTITIONS  4
#define NUM_LOCK_PARTITIONS  (1 << LOG2_NUM_LOCK_PARTITIONS)
```

16

What could be causing this?

- Clearly a concurrency issue.
- Something is contended, but what?
- Let's jump to "obvious" conclusions!

```
/* lwlock.h */
#define LOG2_NUM_LOCK_PARTITIONS  4
#define NUM_LOCK_PARTITIONS  (1 << LOG2_NUM_LOCK_PARTITIONS)
```

16

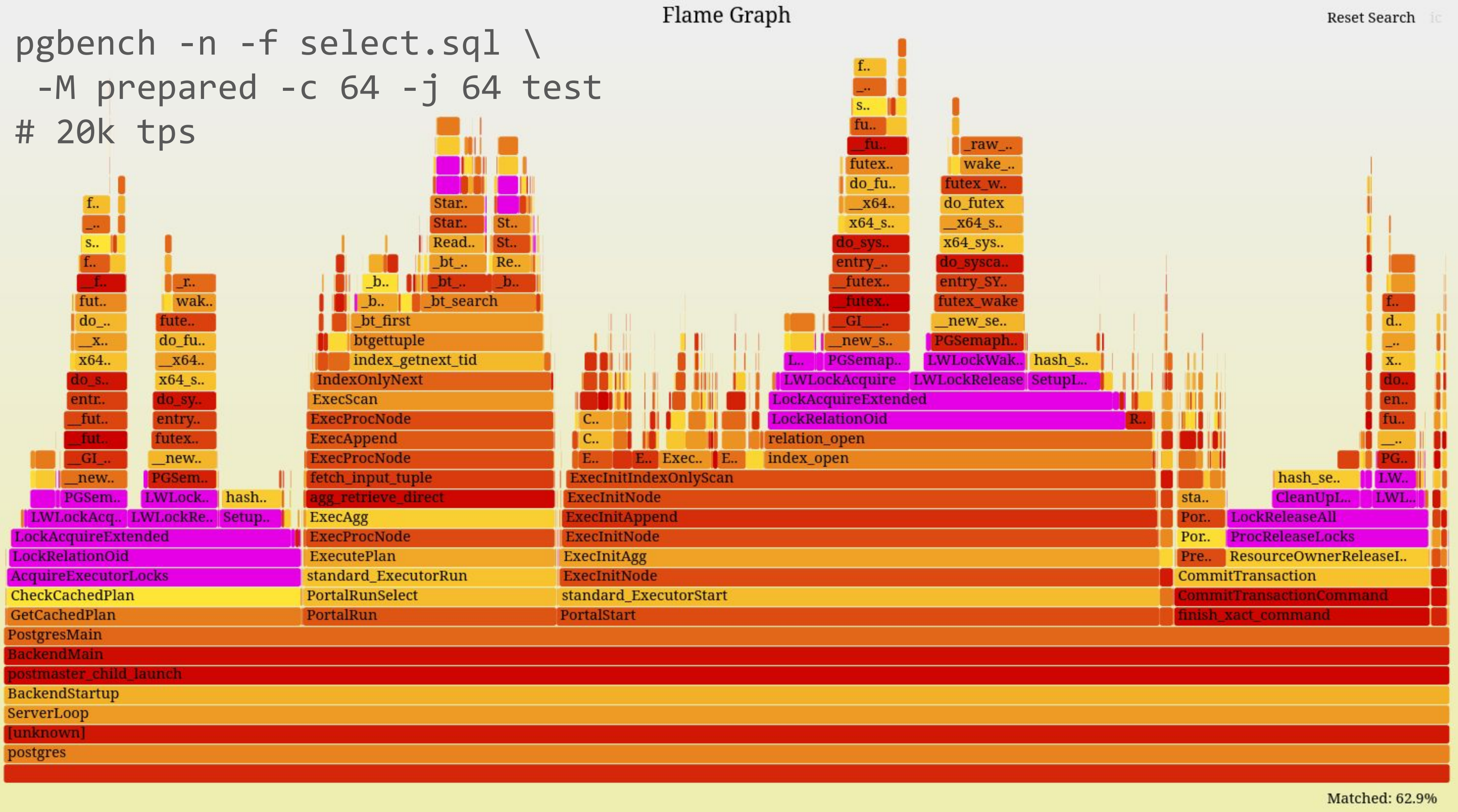
This is not it. Increasing to 64 makes no difference.

Time for crazy ideas ...

- Could be power management / thermal throttling?
  - seen that before, was "fun" to investigate (invisible from a VM)
- Worse with SMT / hyper threading.
  - kinda sad to run with cores disabled
- Could it be malloc contention?
  - maybe a little bit, but a separate issue

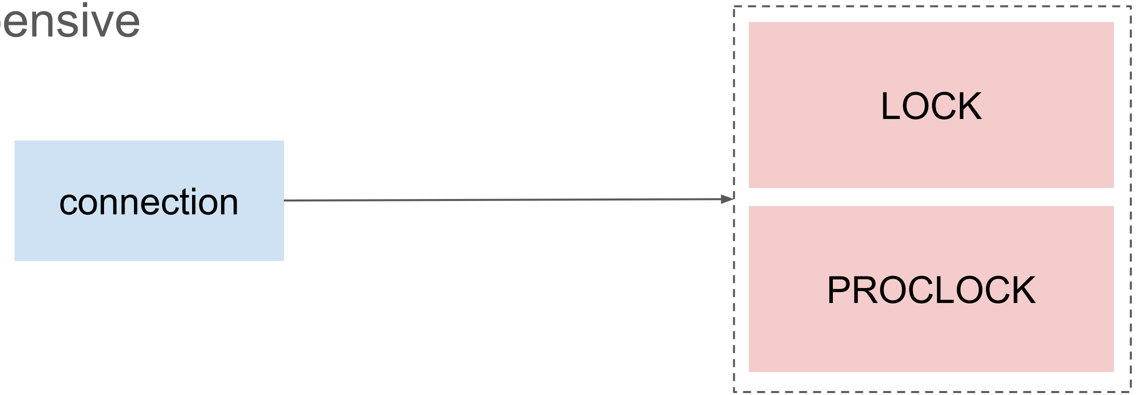
```
Samples: 20K of event 'task-clock:ppp', Event count (approx.): 5181500000
  Children      Self  Command   Shared Object      Symbol
+   99.99%     0.00%  postgres  [unknown]          [.] 0xffffffffffffff
+   99.98%     0.00%  postgres  postgres           [.] ServerLoop
+   99.98%     0.00%  postgres  postgres           [.] BackendStartup (inlined)
+   99.98%     0.00%  postgres  postgres           [.] postmaster_child_launch
+   99.98%     0.00%  postgres  postgres           [.] BackendMain
+   99.98%     0.09%  postgres  postgres           [.] PostgresMain
+   44.91%     0.28%  postgres  postgres           [.] LockRelationOid
+   43.17%     1.45%  postgres  postgres           [.] LockAcquireExtended
+   41.73%     0.03%  postgres  postgres           [.] PortalStart
+   41.60%     0.03%  postgres  postgres           [.] standard_ExecutorStart
+   41.50%     0.22%  postgres  postgres           [.] ExecInitNode
+   41.49%     0.06%  postgres  postgres           [.] ExecInitAgg
+   41.16%     0.11%  postgres  postgres           [.] ExecInitAppend
+   40.77%     0.41%  postgres  postgres           [.] ExecInitIndexOnlyScan
+   28.50%     0.55%  postgres  postgres           [.] relation_open
+   26.86%     0.01%  postgres  postgres           [.] index_open
+   24.69%     0.00%  postgres  [kernel.kallsyms]  [k] entry_SYSCALL_64_after_hwframe
+   24.58%     1.26%  postgres  [kernel.kallsyms]  [k] do_syscall_64
+   22.42%     0.10%  postgres  [kernel.kallsyms]  [k] x64_sys_call
+   20.67%     0.12%  postgres  [kernel.kallsyms]  [k] __x64_sys_futex
+   20.52%     0.04%  postgres  [kernel.kallsyms]  [k] do_futex
+   20.36%     0.01%  postgres  postgres           [.] GetCachedPlan
+   20.31%     0.00%  postgres  postgres           [.] CheckCachedPlan (inlined)
+   20.31%     0.14%  postgres  postgres           [.] AcquireExecutorLocks
+   20.07%     8.12%  postgres  postgres           [.] LWLockAcquire
+   18.25%    16.02%  postgres  postgres           [.] hash_search_with_hash_value
+   17.68%     3.82%  postgres  postgres           [.] LWLockRelease
+   17.67%     0.00%  postgres  postgres           [.] finish_xact_command (inlined)
+   17.67%     0.01%  postgres  postgres           [.] CommitTransactionCommand
+   17.66%     0.05%  postgres  postgres           [.] CommitTransaction
+   17.52%     0.00%  postgres  postgres           [.] PortalRun
```

```
pgbench -n -f select.sql \
 -M prepared -c 64 -j 64 test
# 20k tps
```



locking

- shared lock table
- partitioned (N=16)
- but still expensive



max_connections *
max_locks_per_transaction

Fast-path locking (9.2)

Microsoft

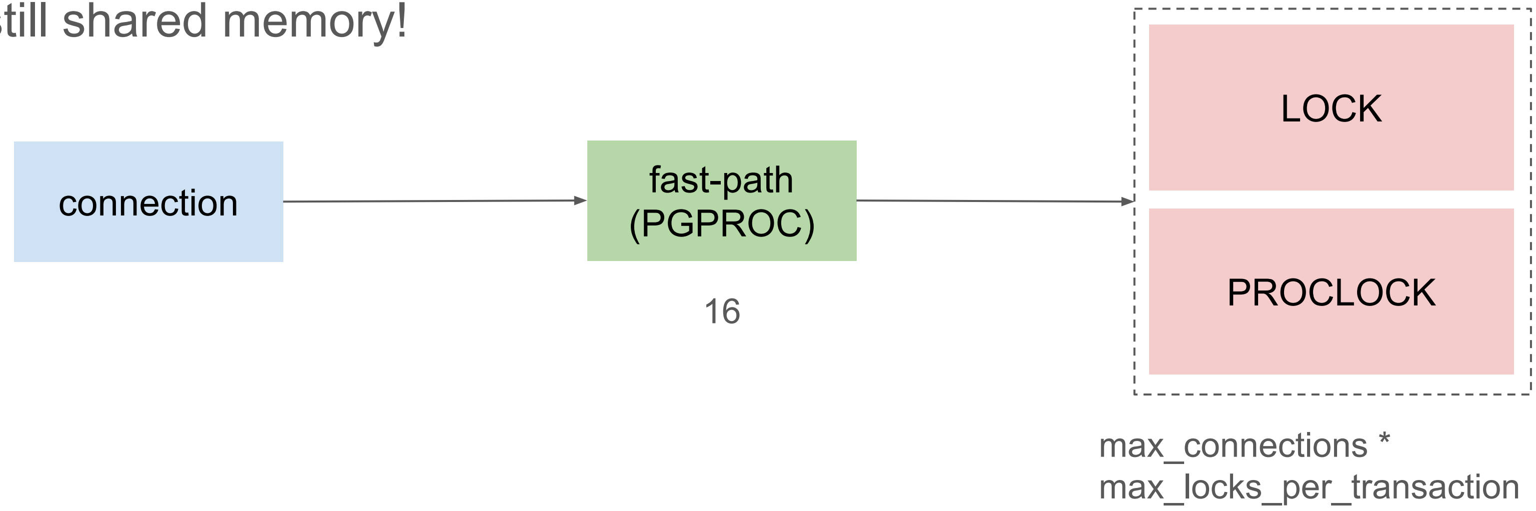
Table 13.2. Conflicting Lock Modes

Requested Lock Mode	Existing Lock Mode							
	ACCESS SHARE	ROW SHARE	ROW EXCL.	SHARE UPDATE EXCL.	SHARE	SHARE ROW EXCL.	EXCL.	ACCESS EXCL.
ACCESS SHARE								X
ROW SHARE							X	X
ROW EXCL.					X	X	X	X
SHARE UPDATE EXCL.				X	X	X	X	X
SHARE			X	X		X	X	X
SHARE ROW EXCL.			X	X	X	X	X	X
EXCL.		X	X	X	X	X	X	X
ACCESS EXCL.	X	X	X	X	X	X	X	X

https://www.postgresql.org/docs/current/explicit-locking.html

fast-path locking

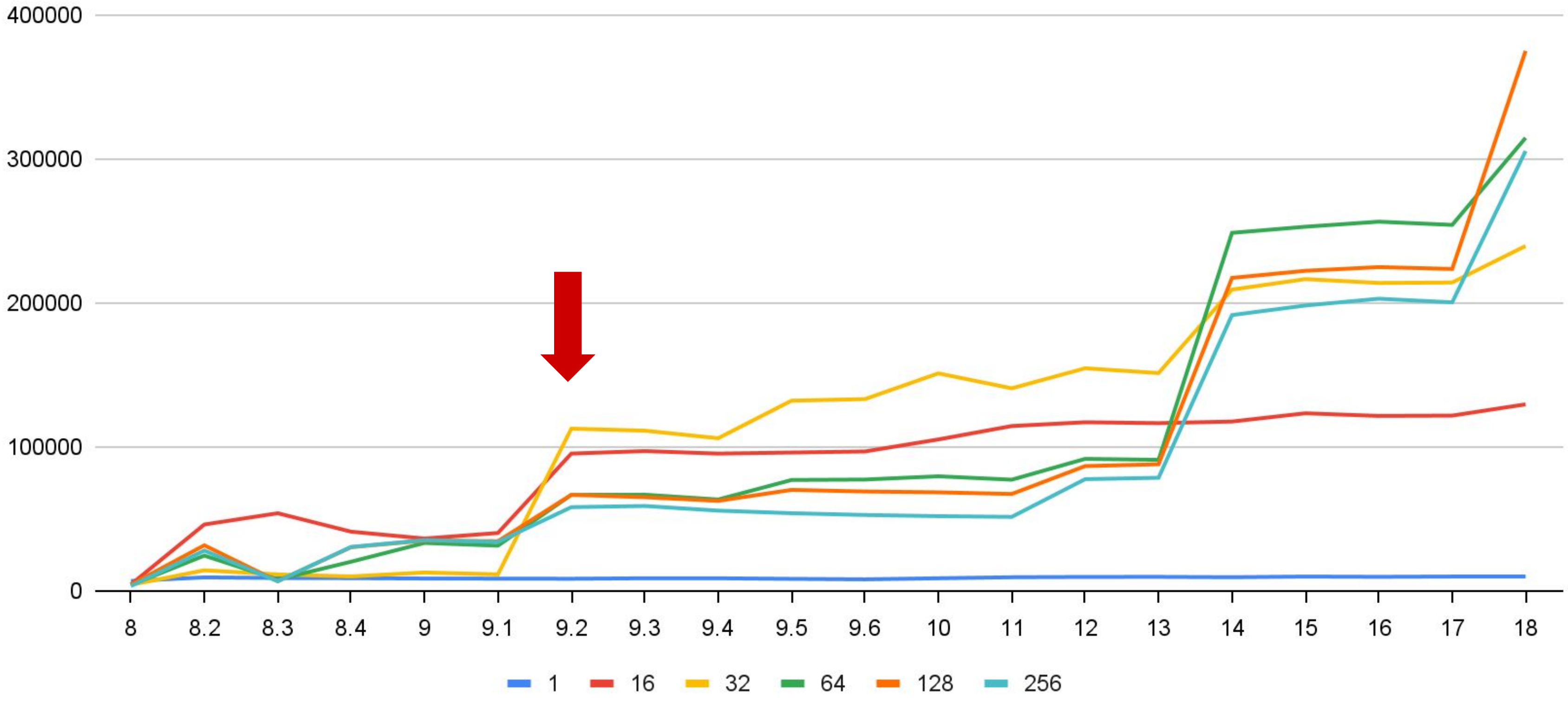
- shared lock table
- local "fast-path" buffer
- still shared memory!



Fast-path locking (9.2)

- fast-path array in PGPROC
  - "local cache" - the point is to not use shared hash table often
  - still in shared memory, but has a separate lock (per process)
- fast-path protocol (lock.c, LockAcquireExtended)
  - fast-path if no one holds a conflicting lock + there's space in PGPROC
  - obtaining conflicting lock -> transfer locks to shared hash table
- capacity for 16 OIDs - that's not very many
  - tables + indexes + ...
  - trivial to hit the limit, especially with partitioning

OLTP starjoin / -M prepared



https://vondra.me/pdf/performance-archaeology-pgconfeu-2024.pdf

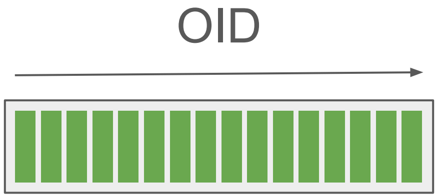
Making it larger ...

- also, make it configurable
  - so that people can adjust that by a GUC
- can't keep it in PGPROC anymore
  - a "struct" needs to be of a fixed-size
  - still has to be shared memory, but as a separate "chunk"
- fast-path locking protocol
  - no change [src/backend/storage/lmgr/README]

But what should be the data structure?

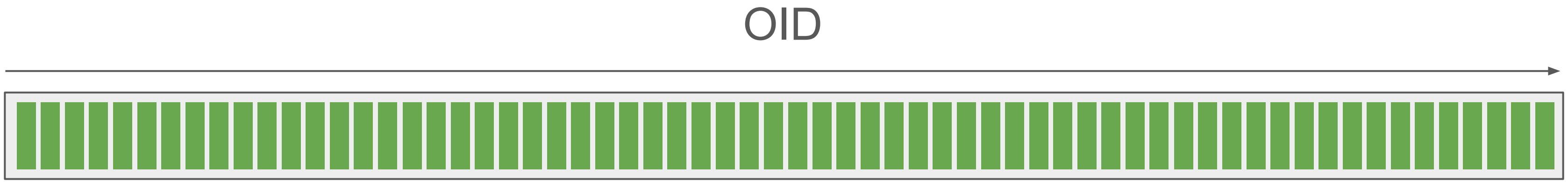
how it used to work

- linear search, 16 slots
- good: simple, fast, cheap, efficient
- bad: limited capacity



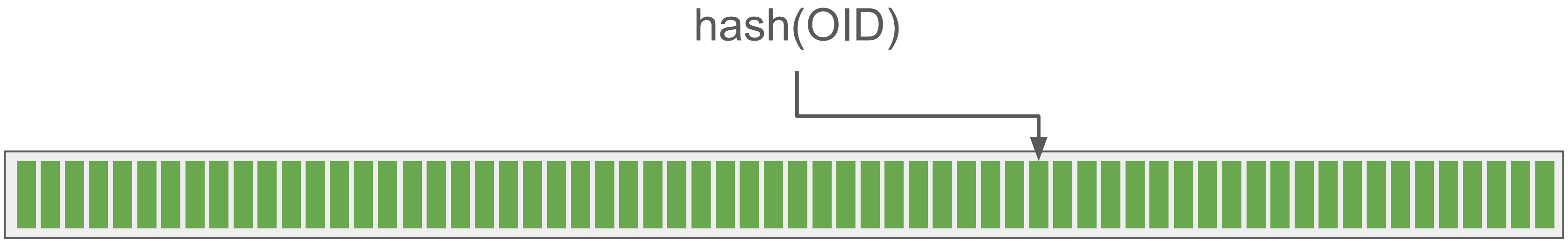
how to improve?

- **increase the number of slots + linear search**
- good: trivial extension (but naive)
- bad: expensive linear search (worst case)



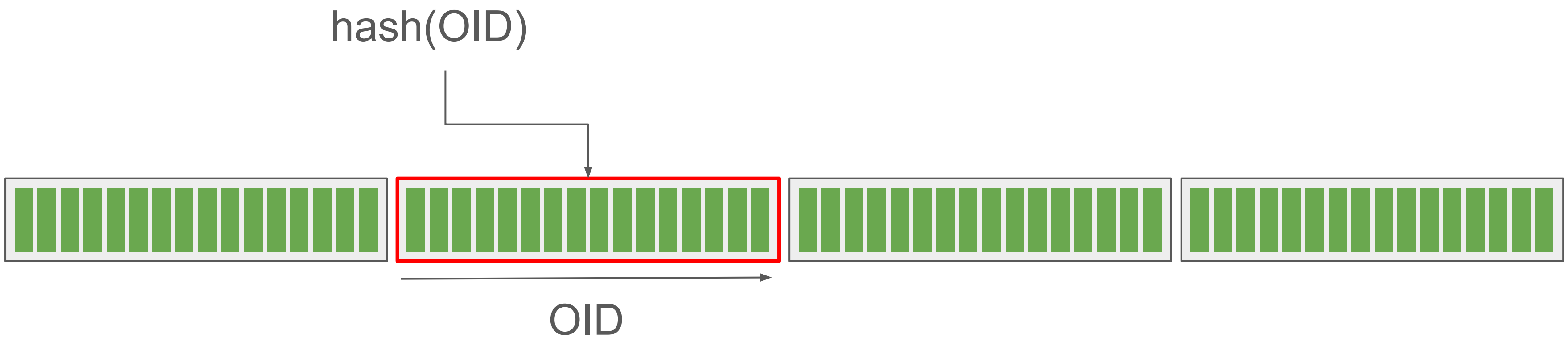
how to improve?

- **use a traditional hash table**
- good: well understood
- bad: not great with high load factor (can't resize!)
- bad: long runs or random access not great (even for RAM)



16-way set-associative cache

- clone the original approach + hash partition



https://en.algorithmica.org/hpc/cpu-cache/associativity/

https://developer.arm.com/documentation/den0013/d/Caches/Cache-architecture/Set-associative-caches

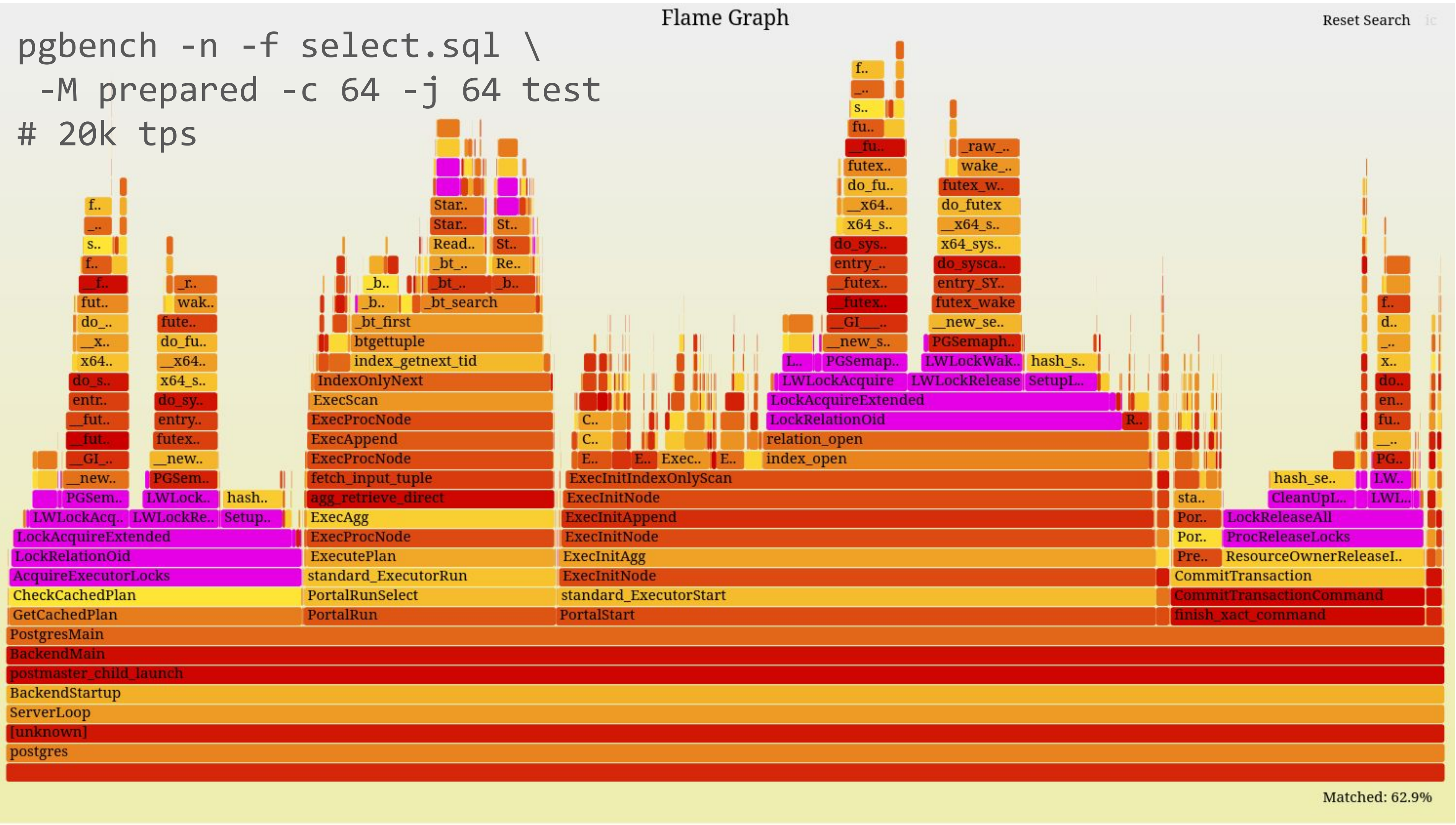
Data structure

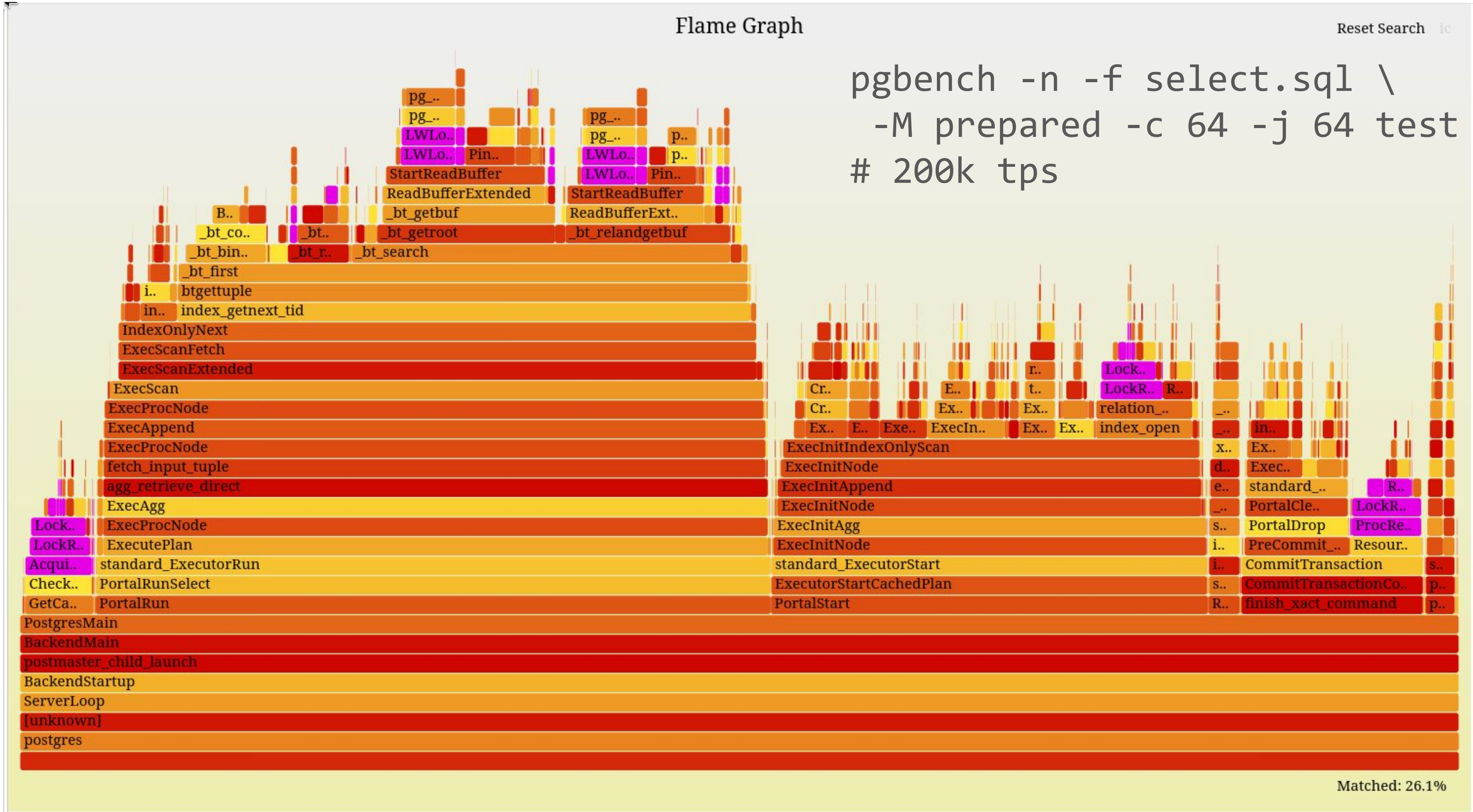
- array + linear search
  - worked great for 16 items, linear search wins here
  - probably not beyond 32/64 items, we're aiming for 1024+
- hash table (open addressing)
  - we'd need to limit load factor (e.g. 75%) to keep it fast
  - random access is not great (cacheline 64B)
- 16-way set-associative cache
  - hash table of arrays
  - ingenious product of my laziness

16-way set-associative cache

- simple concept
  - hash + array
- nice sequential access
  - regular hash tables are much more random
  - not great, even for RAM
  - cache friendly (cachelines)
- no problem with limited capacity
  - can always promote to shared lock table

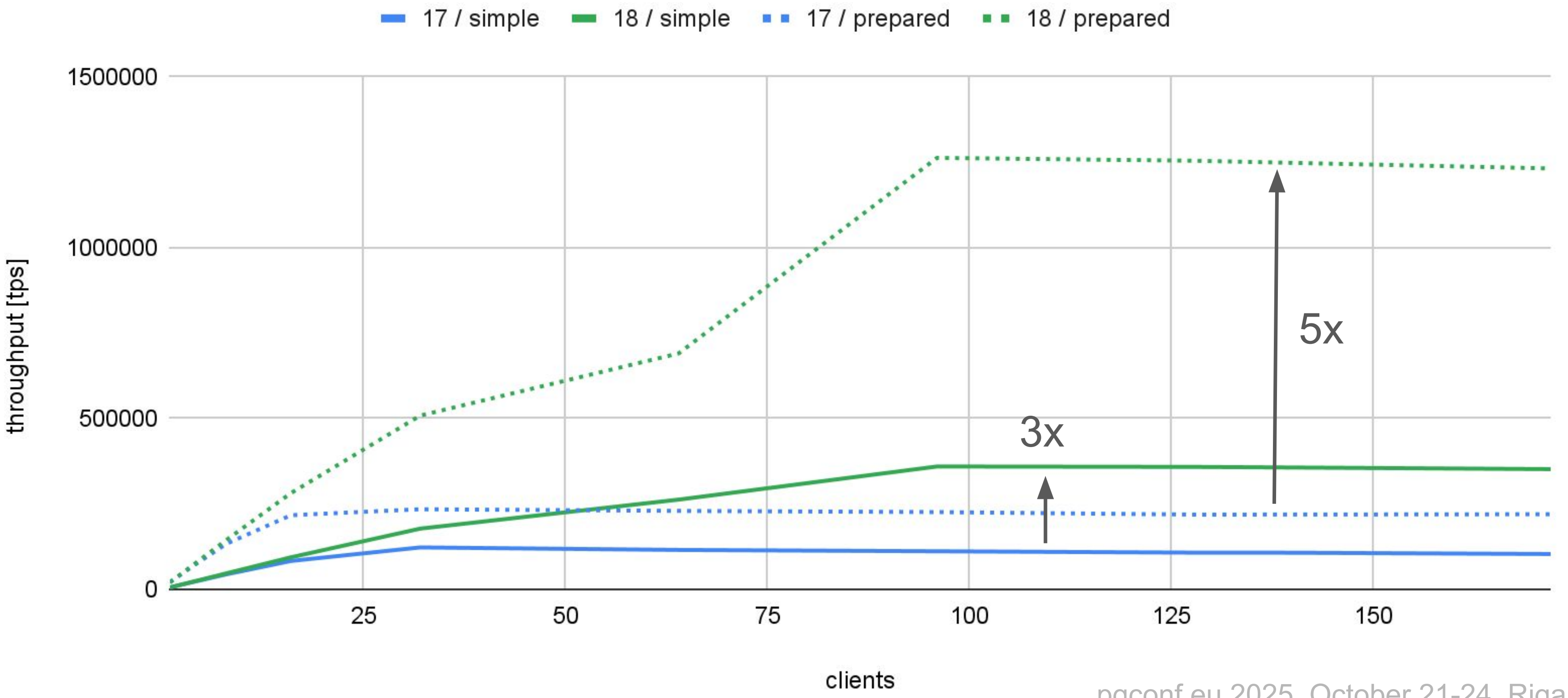
```
pgbench -n -f select.sql \
 -M prepared -c 64 -j 64 test
# 20k tps
```




Flame Graph
Reset Search
pgbench -n -f select.sql \
 -M prepared -c 64 -j 64 test
200k tps
pg_..
pg_..
LWLo..
LWLo..
Pin..
StartReadBuffer
ReadBufferExtended
_bt_getbuf
_bt_getroot
pg_..
pg_..
p..
LWLo..
p..
LWLo..
Pin..
StartReadBuffer
ReadBufferExt..
_bt_relandgetbuf
B..
_bt_co..
_bt..
_bt_bin..
_bt_r..
_bt_search
_bt_first
i..
btgettuple
in..
index_getnext_tid
IndexOnlyNext
ExecScanFetch
ExecScanExtended
ExecScan
ExecProcNode
ExecAppend
ExecProcNode
fetch_input_tuple
agg_retrieve_direct
ExecAgg
ExecProcNode
ExecutePlan
standard_ExecutorRun
PortalRunSelect
PortalRun
Lock..
LockR..
Acqui..
Check..
GetCa..
Lock..
LockR..
R..
relation_..
index_open
ExecInitIndexOnlyScan
ExecInitNode
ExecInitAppend
ExecInitNode
ExecInitAgg
ExecInitNode
standard_ExecutorStart
ExecutorStartCachedPlan
PortalStart
Exec..
standard_..
PortalCle..
PortalDrop
PreCommit_..
CommitTransaction
CommitTransactionCo..
finish_xact_command
LockR..
ProcRe..
Resour..
PostgresMain
BackendMain
postmaster_child_launch
BackendStartup
ServerLoop
[unknown]
postgres
Matched: 26.1%

10 partitions, max_locks_per_transaction = 64

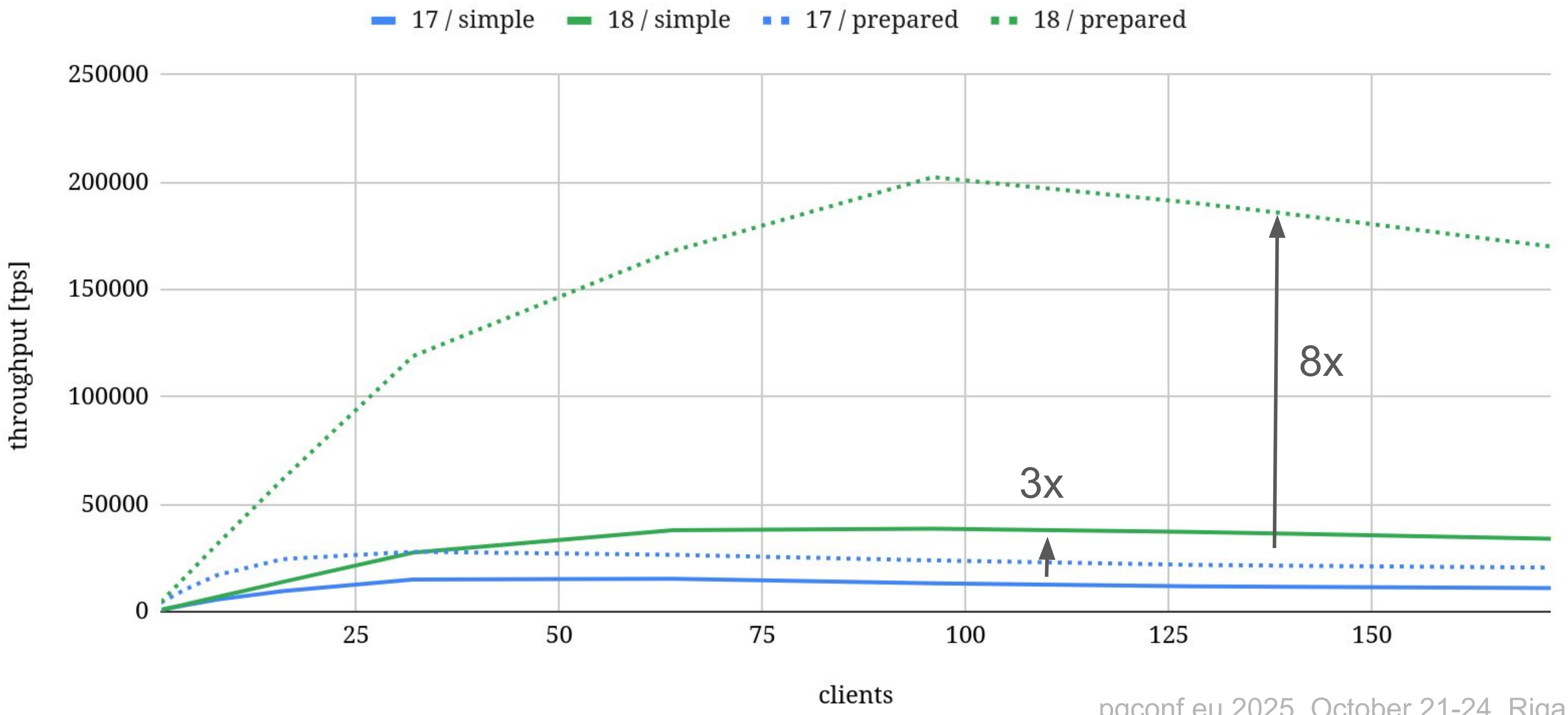
AMD EPYC 9V74 80-Core Processor



100 partitions, max_locks_per_transaction = 1024

AMD EPYC 9V74 80-Core Processor

Microsoft



Trade-offs

- tied to max_locks_per_transaction
  - ease of tuning vs. configurability (too many GUCs)
  - best idea about how many locks to expect
  - per-backend limit (max_locks_per_transaction was not that)
- what's a good value?
  - no "optimal" value, depends on workload
  - fast-path locks are cheaper (smaller) than shared lock table entries
- max_locks_per_transaction = 64
  - sensible, maybe not ideal for "unbalanced" clusters?
  - should be enough for ~10 tables

What's next?

Future

- we need better monitoring
    - how do you pick the max_locks_per_transaction value?
- pg_locks is not great for this
    - snapshot of current state
- probably some cumulative counters
    - number of locks
    - number of fast-path overflows
    - can we track "peak lock count"?

Future

- could we use the same idea elsewhere?
  - pins for "hot" buffers - maybe a "fast-path pinning"?
  - Problem #4 - Buffer Lock Contention (https://youtu.be/V75KpACdI6E?t=2120)
- consider hotness
  - now first come, first served
  - Maybe consider how often an OID is locked? Has to be cheap.
- NUMA effects
  - maybe should be NUMA partitioned
  - same NUMA node as PGPROC?
- make shared lock table cheaper
  - lock less often / keep locks, maybe smaller entries, ...

Other bottlenecks

- glibc malloc vs. concurrency
  - btbeginscan() allocates ~30kB, can't be cached, always malloc
  - MALLOC_TOP_PAD_ (see mallopt)
  - two "connected" bottlenecks - have to address both
  - jemalloc/tmalloc do not have this issue
- join order planning
  - OLTP starjoin
  - other bottleneck swamping the results
- multiple bottlenecks can be hit simultaneously
  - and compose in non-linear way (50% vs. 10x speedup)

Shout-out

- Robert Haas
  - wrote the fast-path locking in 9.2
  - it was extremely easy to build on his code
  - first PoC patch in ~ ½ day, worked on 1st try

- Jakub Wartak
  - support engineer / hacker in EDB investigating this
  - provided a lot of great insights and expertise
  - super-fun collaboration

feedback

Prague events

Prague PostgreSQL Developer Day 2026

January 27-28

CfP (closes November 14)
https://cfp.p2d2.cz/2026/

looking for sponsors & partners

Prague PostgreSQL Meetup

https://www.meetup.com/prague-postgresql-meetup

Tomas Vondra

- Postgres engineer @ Microsoft
- https://vondra.me
- vondratomas@microsoft.com
- tomas@vondra.me
- office hours
- ...

Q & A