

Stop Guessing!

Unlocking **Wait Events** at Scale
with Existing and **New** Tools

Dmitry Fomin
PGConf.EU 2026.
Valencia

Why do I care about **latency** and **performance**

- Postgres performance and observability practitioner at Adyen
- Oracle performance certified expert
- Off-duty: play blues harmonica



~~You~~ Many people are troubleshooting
PostgreSQL wrong

A veterinarian goes to the doctor

The doctor asks: "So, what seems to be the problem?"

The vet: "Oh, come on! **Anyone can do it *that way!***"

Animals can't tell you where it hurts.

PostgreSQL *can*, if you **ask** it the *right* question

Canonical PostgreSQL troubleshooting

- `pg_stat_*` views
- Logs / pgbadger
- `pg_locks`
- `EXPLAIN`, `auto_explain`
- OS metrics
- Educated guess

Everything is green but still degraded

● ● ● prod-db-01 · last 1 hour

● All systems operational

CPU

● OK

31 % of 64 cores



Memory

● OK

118 / 512 GB · swap 0



Disk I/O

● OK

12 % busy · 0.4 ms



Network

● OK

1.2 / 25 Gb/s



Connections

● OK

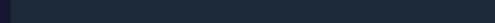
212 / 1000



Buffer cache hit

● OK

99.8 %



Replication lag

● OK

0 s



Blocked locks

● OK

0 pg_locks



Errors in log

● OK

0 last hour



Slow query log

● OK

0 statements > 5 s



Throughput

● DOWN

6.1k TPS · was 18k



Latency p99

● UP

4.8 s · was 12 ms



Waiting on what? No panel for that.

Problematic queries are almost invisible

- pg_stat_statements - when it finishes
- auto_explain - when it finishes
- log_min_duration_statement - when it finishes
- **Outliers** are dissolved by averages

Run it again and it's **faster or slower**

Plan shows how data processed and time spent in each node but it does not say

WHY

Guess. Try. Repeat.

Plan flip?

Bloat?

autovacuum?

Locks?

Disk?

Concurrency?

Where did the time go?

response time = service time + wait time

OEM



“I was busy” is not a timesheet

Contractor 10 hours: "I was **busy**."

- **waiting** for the cement truck 5h - IO
- **waiting** for the scaffolding 3h - Lock
- laying bricks 2h - CPU

PostgreSQL has had wait events since 9.6

Lock:transactionid IO:DataFileRead

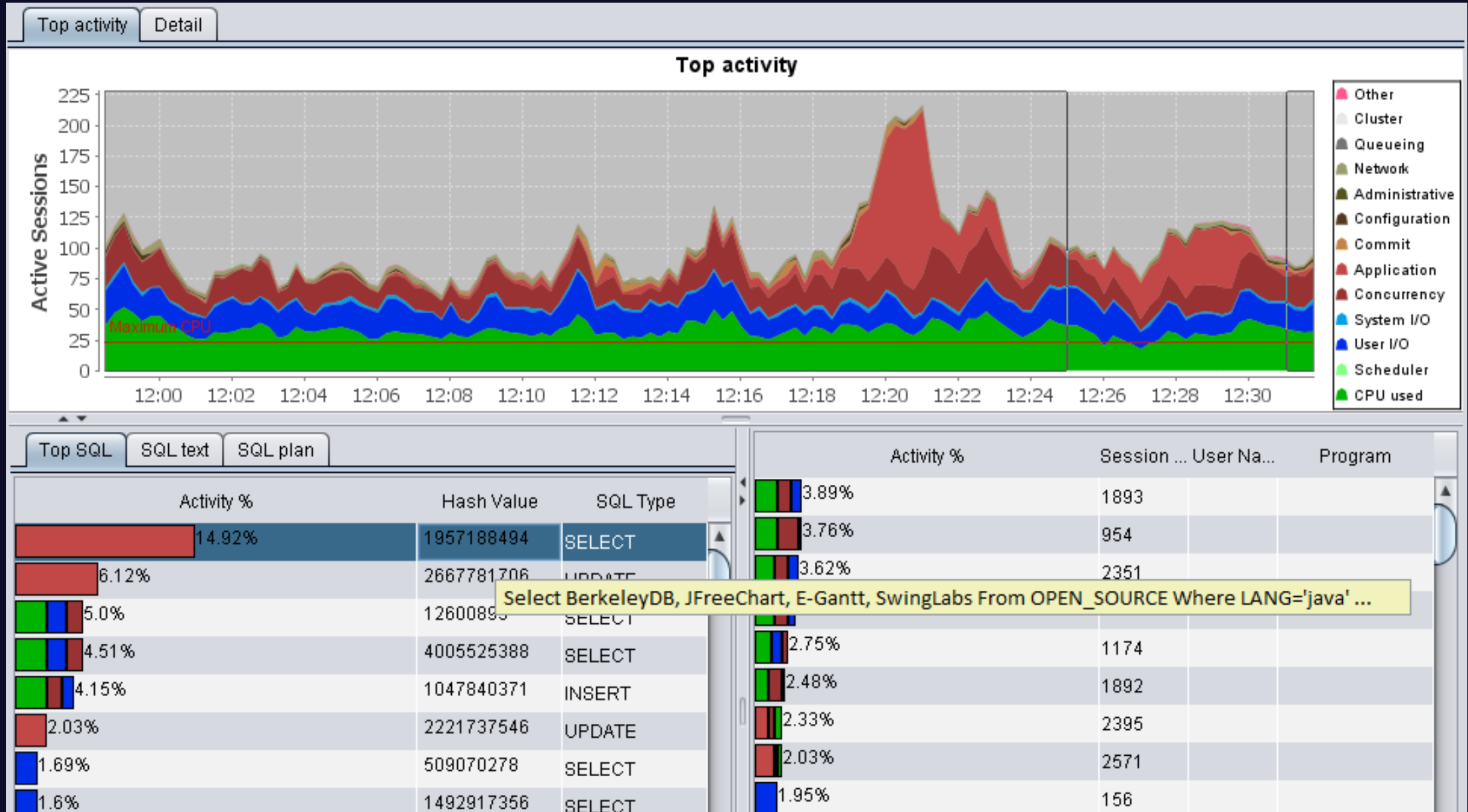
LWLock:LockManager

Client:ClientRead IPC:SyncRep

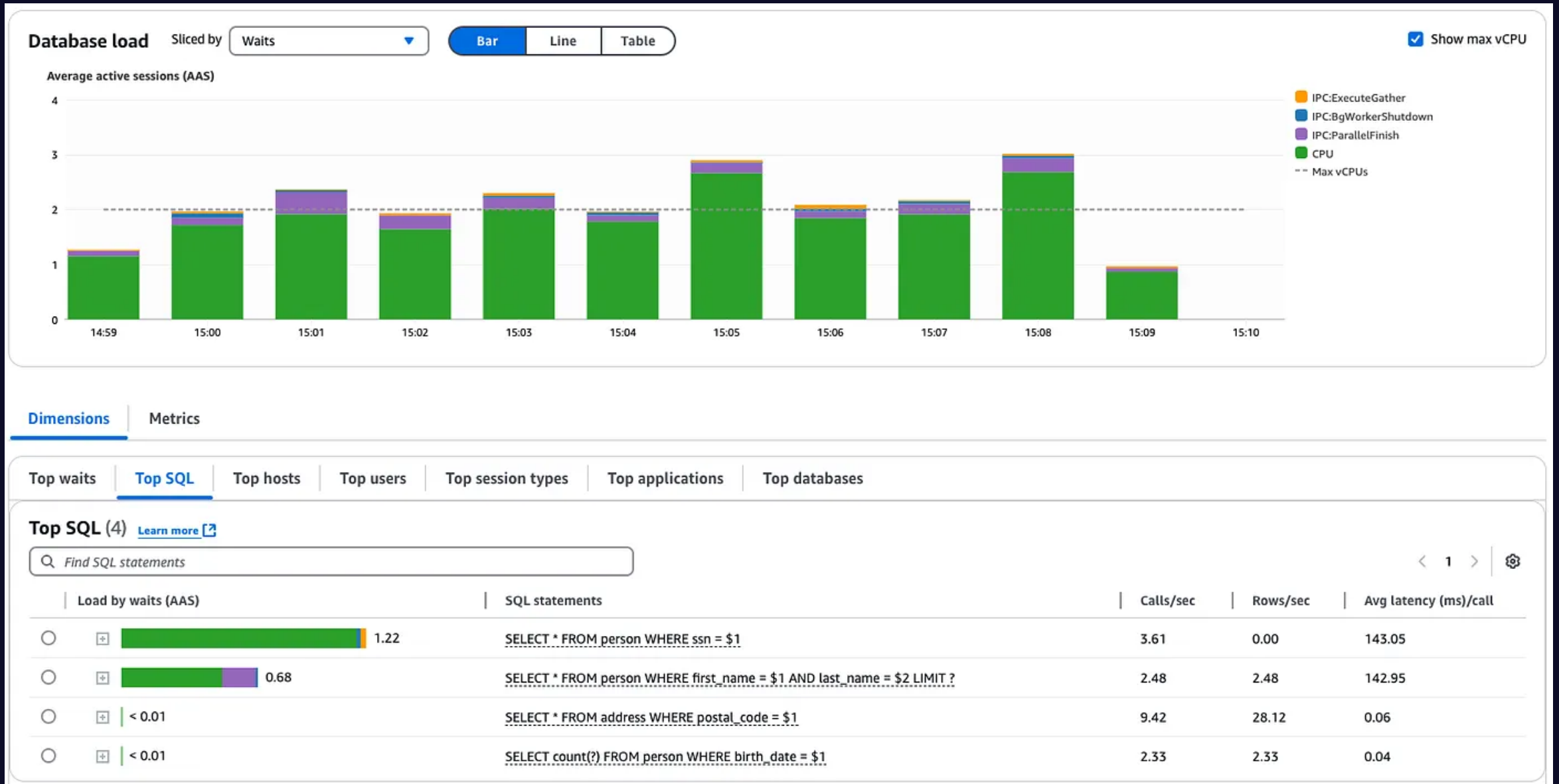
Extensions and others:

- pgsentinel
- pg_wait_sampling
- PASH-Viewer
- pg_ash
- Performance Insights and friends

PASH-Viewer



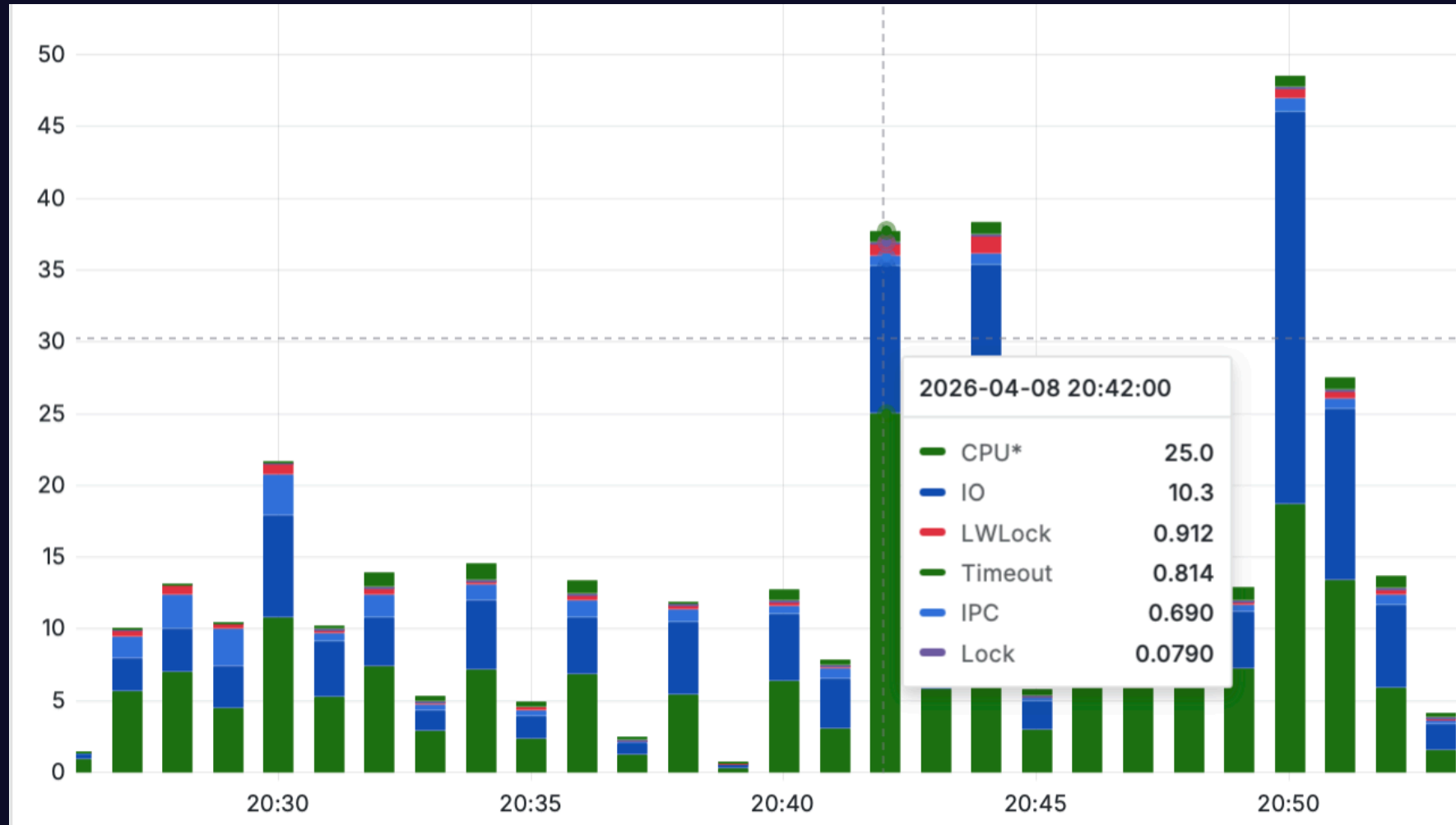
CloudWatch Database Insights



pg_ash

wait_event	samples	pct	bar	
Lock:transactionid	821	25.63		25.63%
CPU*	753	23.51		23.51%
Client:ClientRead	664	20.73		20.73%
Lock:tuple	284	8.87		8.87%
IO:WalSync	205	6.40		6.40%
IO:AioIoCompletion	157	4.90		4.90%
LWLock:WALWrite	152	4.75		4.75%
IdleTx	123	3.84		3.84%
IO:DataFileRead	17	0.53		0.53%
IPC:BgworkerShutdown	11	0.34		0.34%
Other	16	0.50		0.50%
(11 rows)				

Waits became the first panel

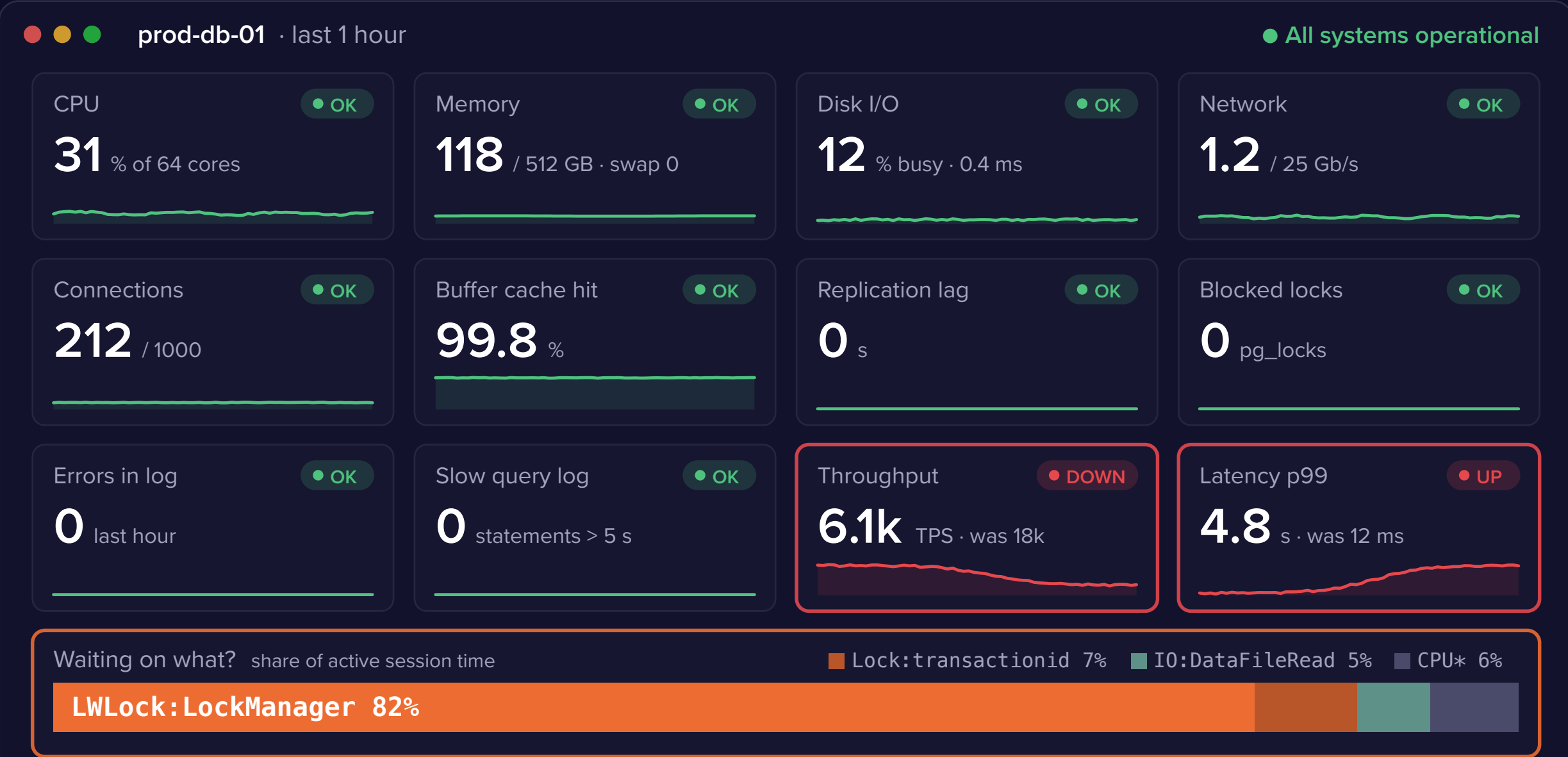


Lab environment

Case 1: New HW. Faster CPUs. Slower PostgreSQL.

- Changed silicon
- Postgres got slower
- Usual guess Plan flip? Config drift? Saturation?

Case 1: New HW. Faster CPUs. Slower PostgreSQL.



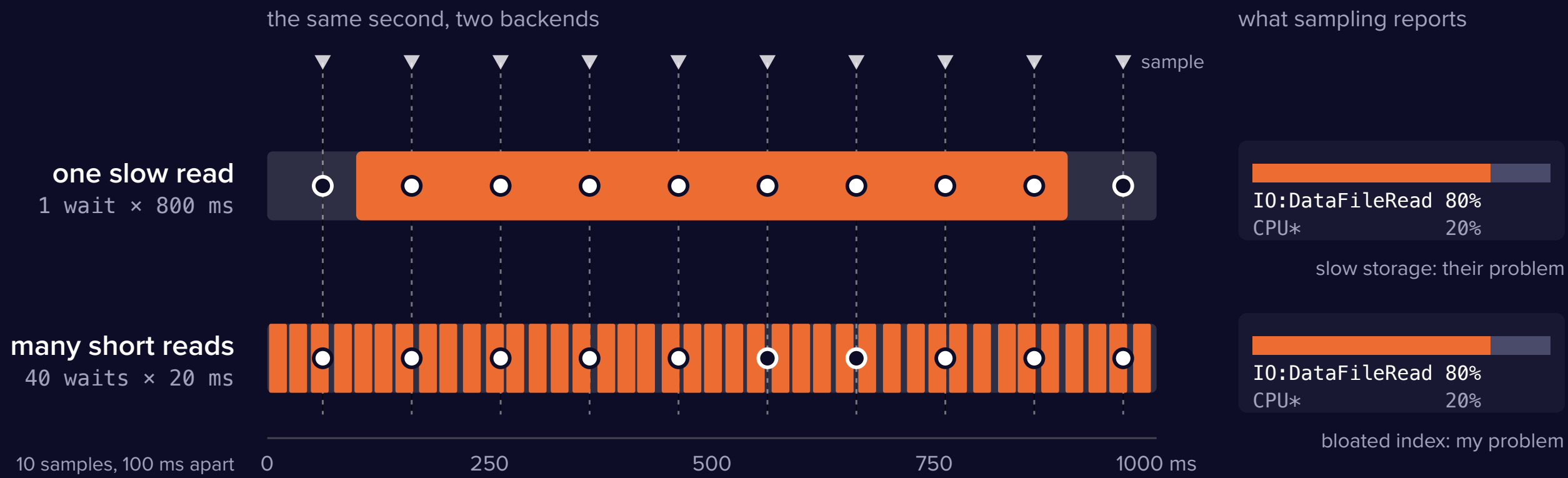
Case 2: Insert into table 1 ms → 40+ s

- Everything is green, clean logs, nothing changed except flapping query latency.
- Usual guess Plan flip? Busy disk? Saturation?
- Wait event **IO:DataFileRead**
- **INSERT** reading the TOAST index for seconds → **OID** exhaustion

Case 3: A primary-key lookup that sometimes took a second.

- PK SELECT, usually milliseconds. Sometimes the 1s `statement_timeout`
- Usual guess Lock contention? Plan flip? Disk issues?
- Wait event `IO:DataFileRead`
- too many reads → bloated index? → REINDEX (the DBA)
- slow reads → storage? → escalate (the storage team)

One long wait or tons of short ones? The profile says: yes.



Same samples. Same profile. Different problem.

What a sample can't tell you

- how many waits there were
- how long each one lasted
- what happened between two samples
- order of waits
- when the box is overloaded, the sampler runs late too

The idea: watch the four bytes

MyProc->wait_event_info



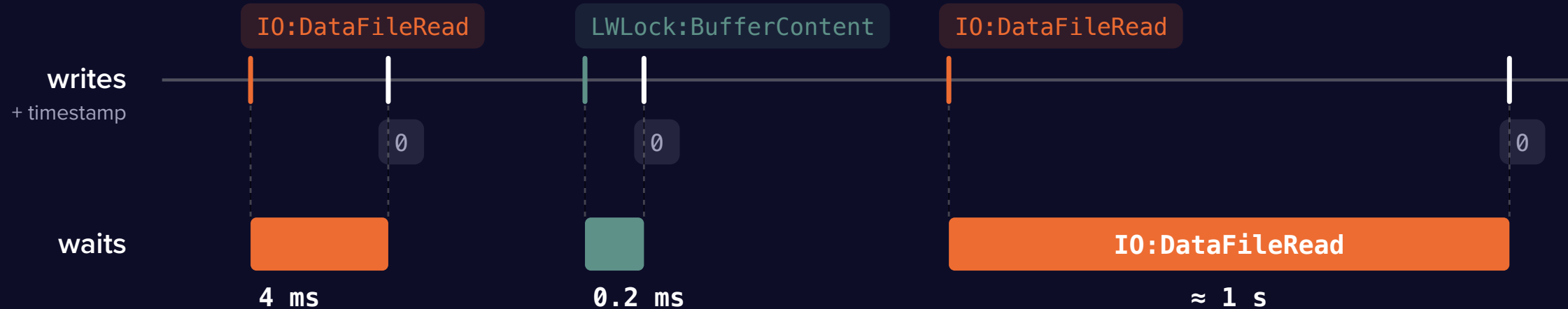
class

event

4 bytes in shared memory, one per backend

Every wait starts with a write here,
and ends with a write of zero.

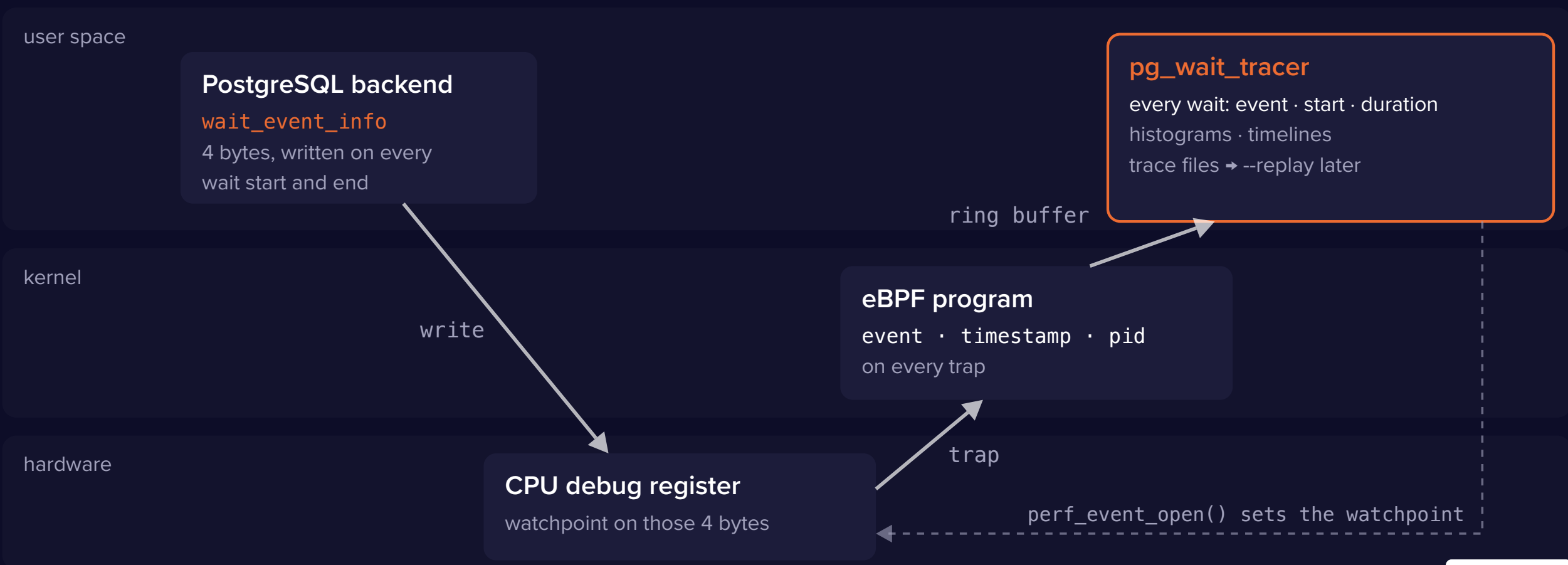
Catch every write, with a timestamp, and you have every wait.



Not a photo. The whole film.

illustration, not to scale

pg_wait_tracer



No patch to PostgreSQL.

Linux, root or CAP_BPF · CAP_PERFMON · CAP_SYS_PTRACE



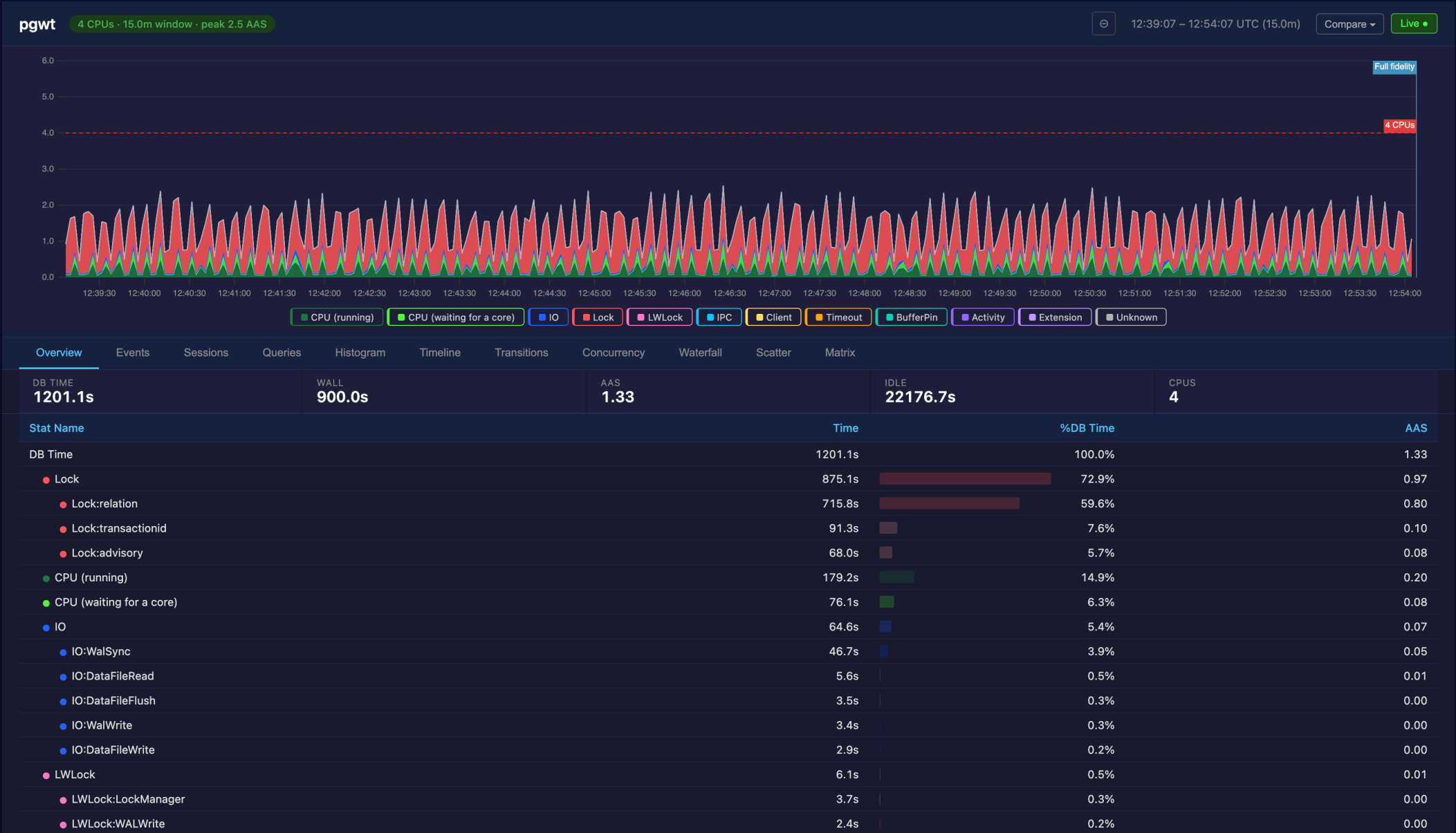
Case 3: A primary-key lookup that sometimes took a second

- **DataFileRead** usually takes $\sim 1\text{ms}$
- DataFileRead sometimes takes up to $\sim 120\text{ms}$
- one lookup $\rightarrow 9+$ slow reads $\times 120\text{ ms} > 1\text{ s timeout}$

What a trace opens up

- real counts and durations, per wait event
- histograms instead of averages
- the exact timeline of one execution
- several backends side by side
- **next:** Markov chains, queueing theory
application, process mining

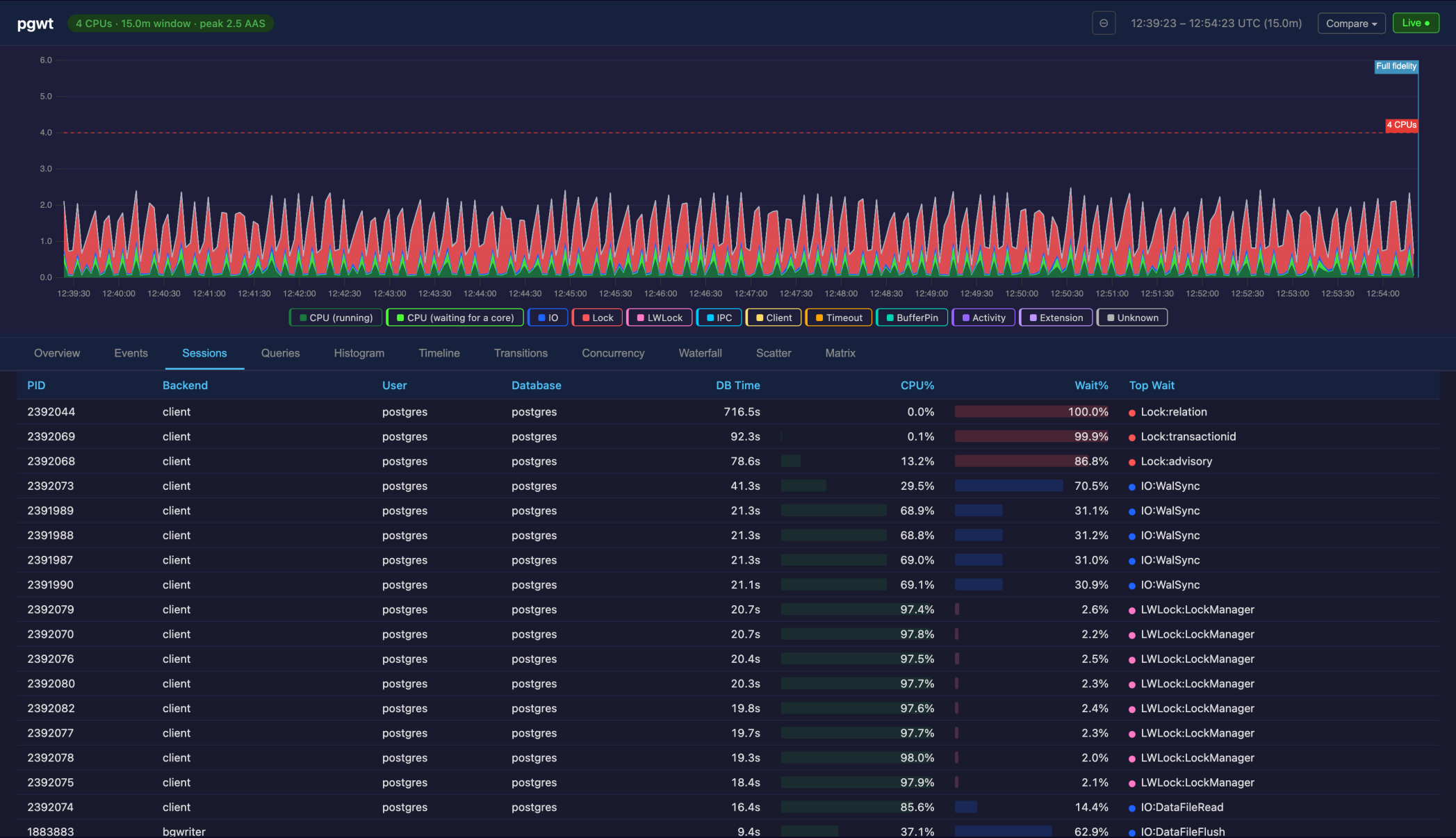
PGWT: Overview, DB time



PGWT: Queries



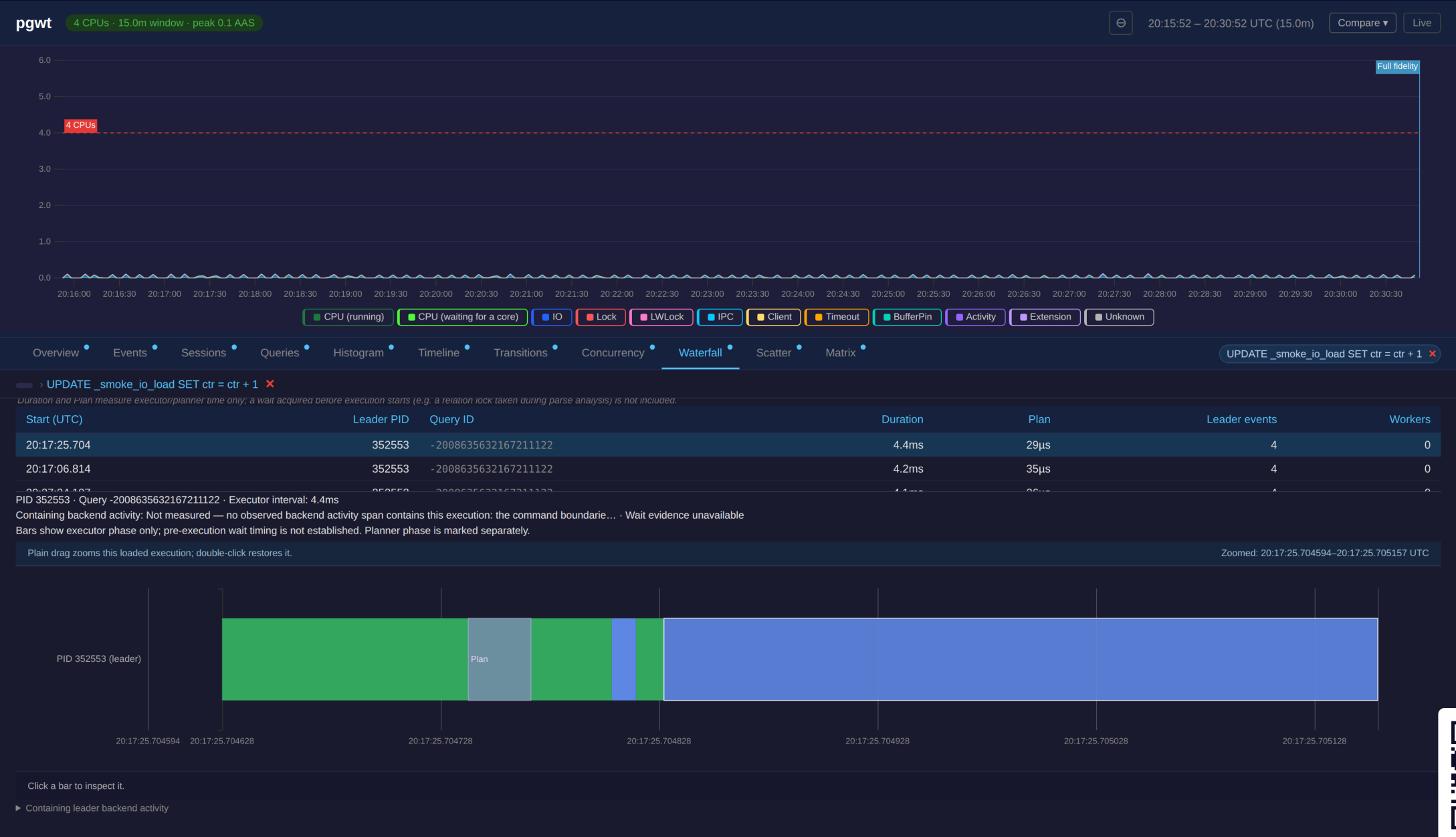
PGWT: Backends



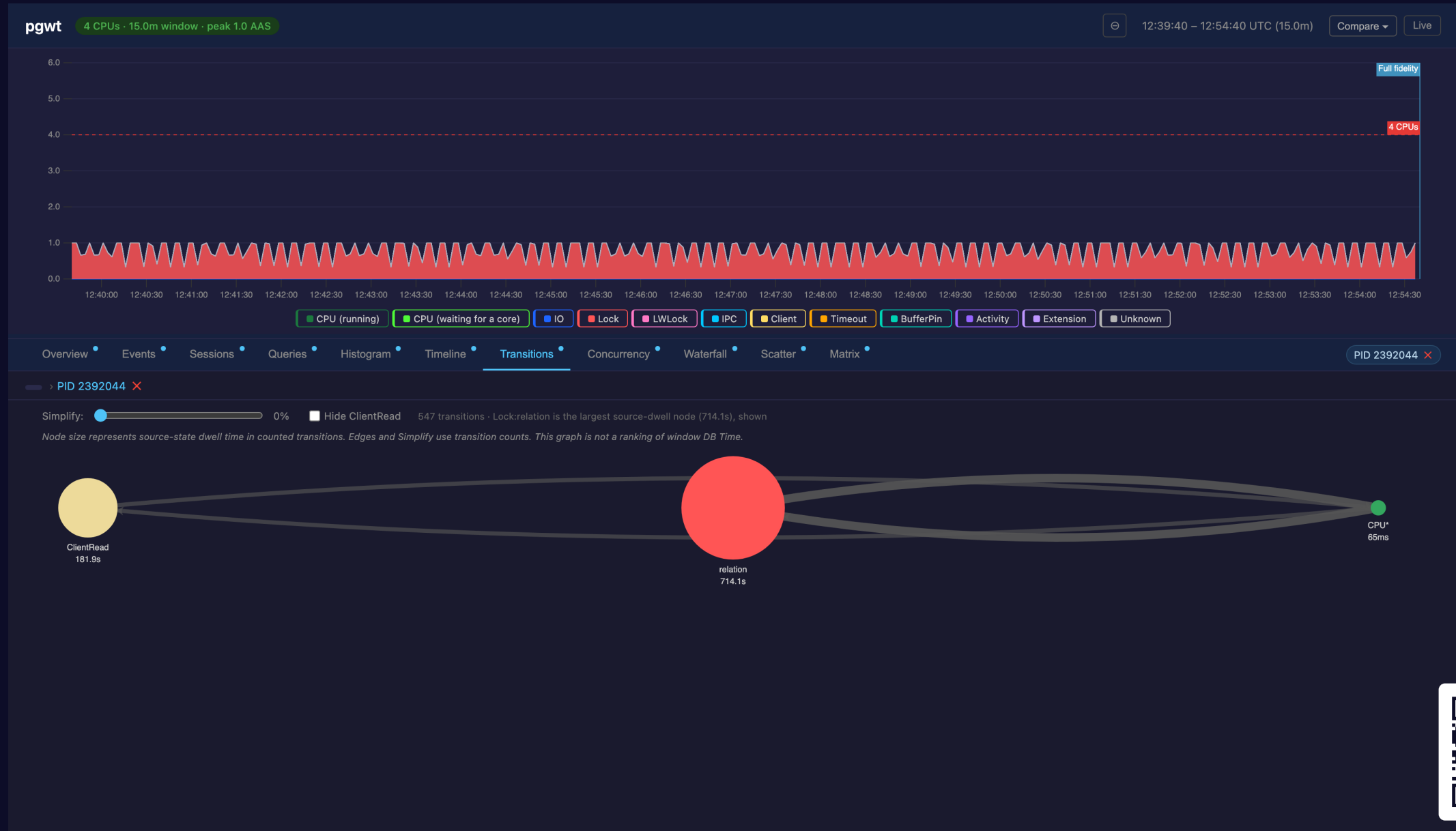
PGWT: histogram



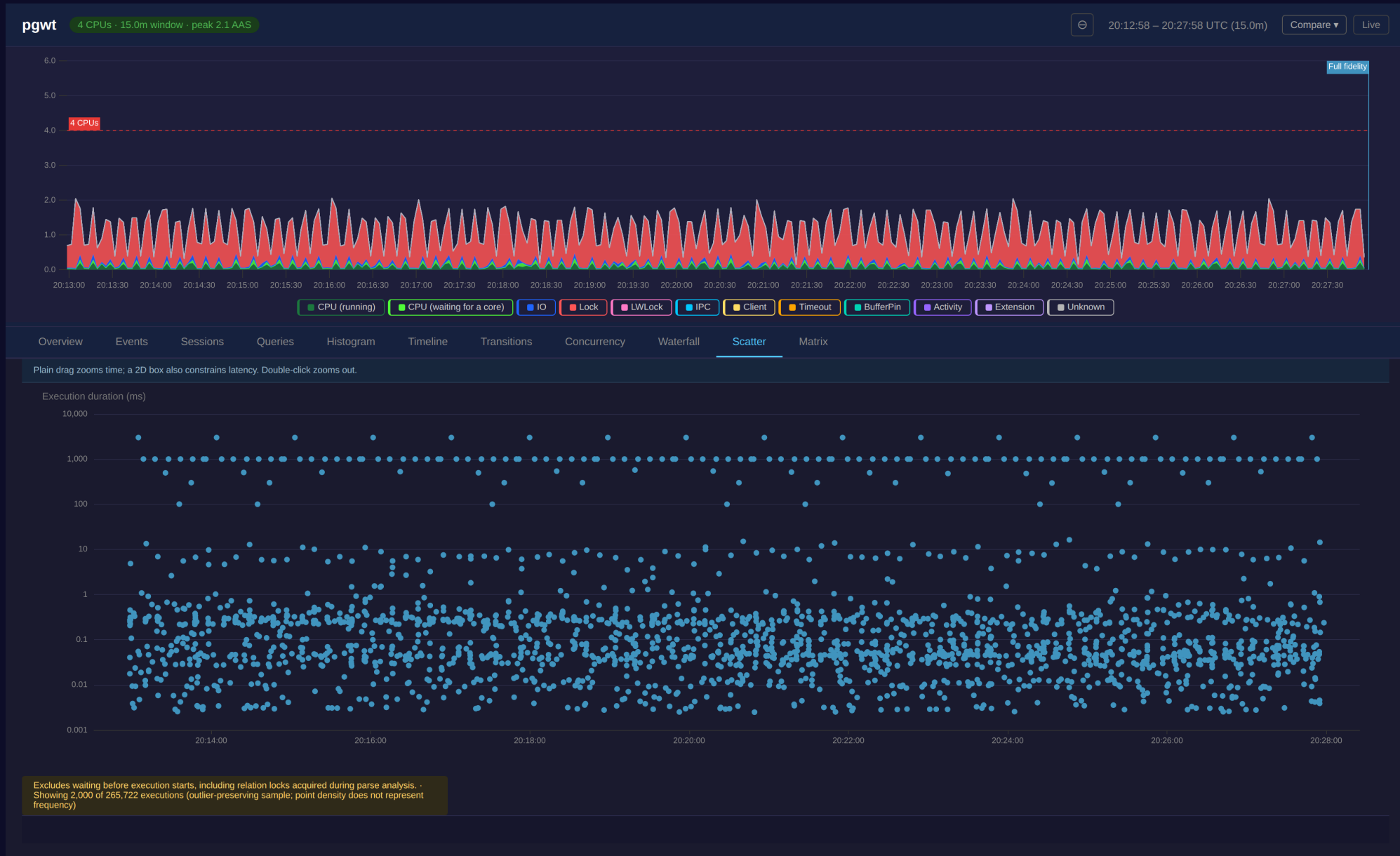
PGWT: microsecond precision



PGWT: transitions



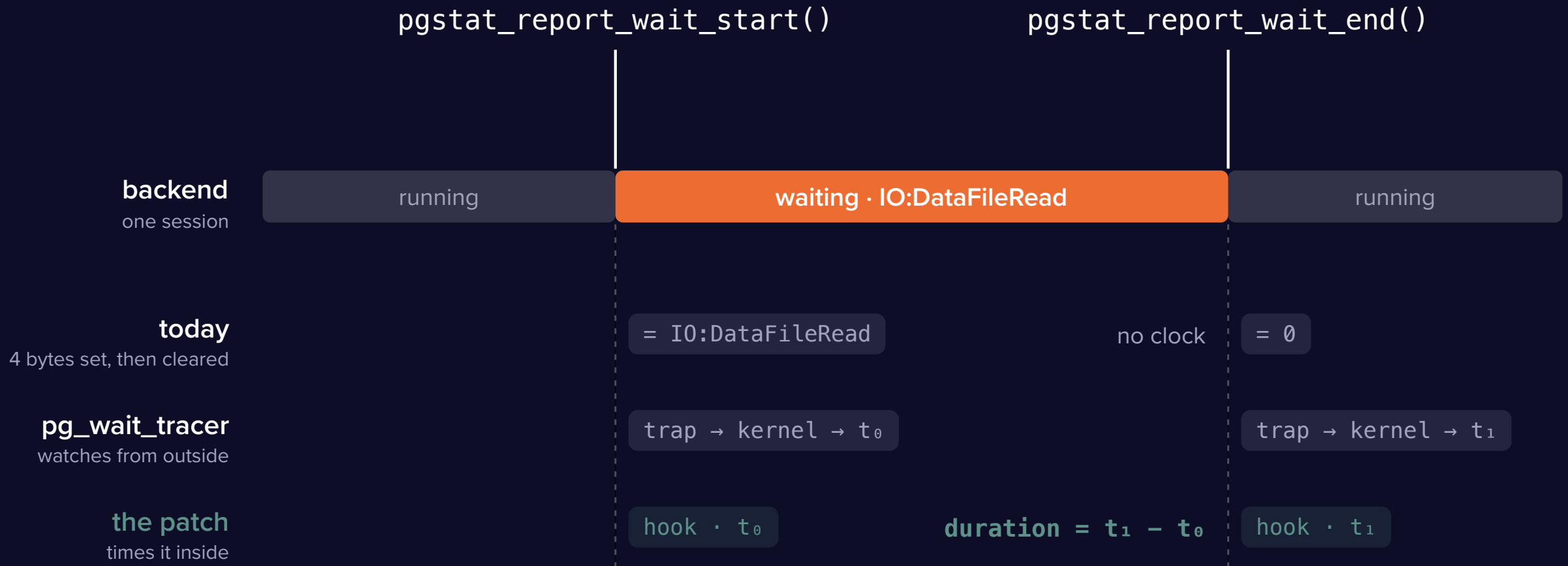
PGWT: scatter



The **exact** tier is a **diagnostic**. The **always-on** tier is **sampling** again

- every wait change is a trip through the kernel: the cost grows with the wait rate
- needs the host: Linux, root, eBPF
- managed PostgreSQL: no host, no tool
- Turn it on. Answer the question. Turn it off.

PostgreSQL already knows when every wait starts and ends



Why watch from outside? Time it inside.

The **patch**: timed hooks, a recorder, SQL

0001-0002

Hooks

at the wait call sites

```
pgstat_report_wait_start_timed()  
pgstat_report_wait_end_timed()
```

Nothing installed:

one pointer test per side

0003

test module: the hook contract

core PostgreSQL

pg_wait_event_tracing

a module in contrib

0004

stats

per wait event: count, total time,
max, histogram

0005

trace

+ every wait, in a ring per backend

capture

off

stats

trace

per session, role or database

contrib/pg_wait_event_tracing

SQL

read it like any statistics view

```
pg_stat_wait_event_timing
```

```
pg_get_backend_wait_event_trace()
```

```
pg_wait_event_trace_by_statement()
```

granted to pg_read_all_stats

No root. No eBPF.

No debug registers.



800 waits, counted and timed

```
SELECT wait_event, calls, total_time_ms,  
       max_time_us / 1000.0 AS max_time_ms  
FROM pg_stat_get_wait_event_timing(pg_backend_pid())  
WHERE wait_event = 'PgSleep';
```

calls

800

asked for 800 sleeps of 4 ms

total_time_ms

3,983.52 ms

mean 4.98 ms per wait

max_time_ms

39.22 ms

one wait, $\approx 8\times$ the mean

Similar means. Different distributions

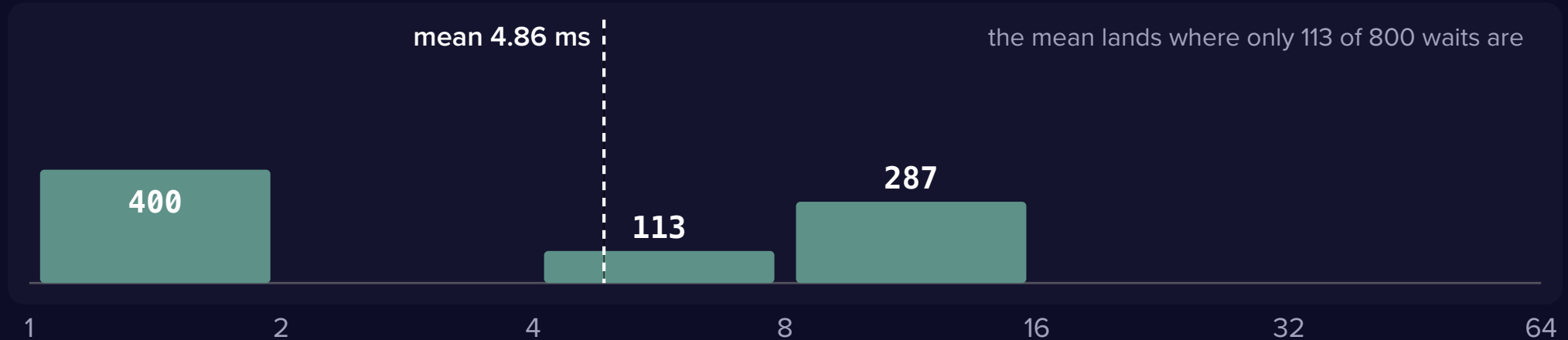
requested 4 ms

800 waits



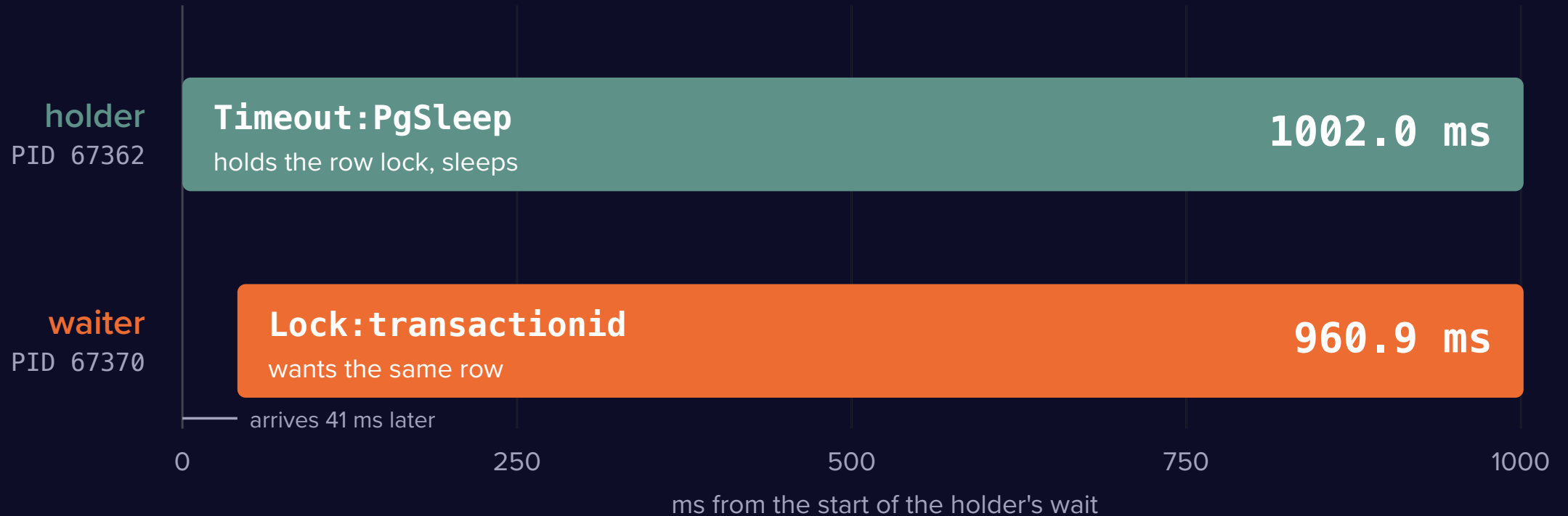
requested 1 / 7 ms

800 waits



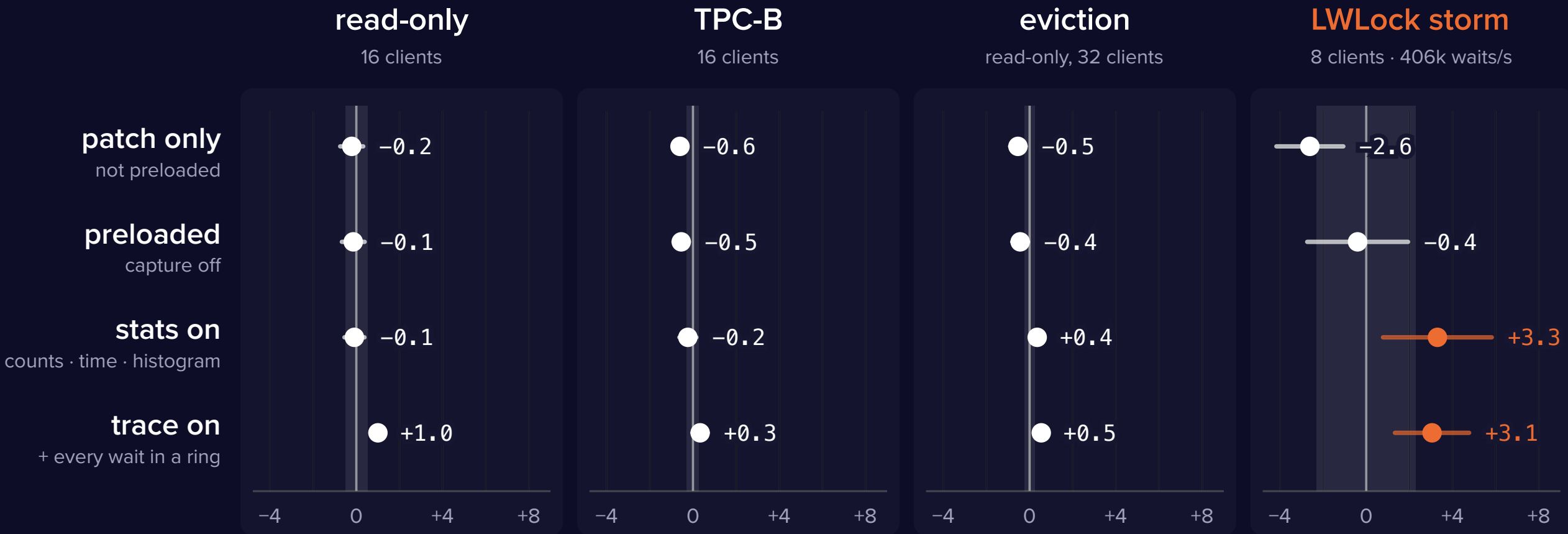
wait duration, ms (power-of-two buckets, labels approximate)

Put the waiter beside the holder



Both backends had exited. Their traces were still readable: 36 and 29 records.

Off: **no** measurable cost. On: about **1%** or less. Worst case: **3%**



Slowdown vs vanilla master, % TPS (right = slower) · dot = mean, bar = 95% CI · shaded = noise between two vanilla builds

v11 · two-socket Xeon 8462Y+, server and pgbench on 8 cores each, one NUMA node, SMT and turbo off · GCC 8.5 · 16 randomized blocks · paired 95% CI

What to do on Monday

- Put waits on the first panel of your dashboard
- For counts, tails or order, trace.
`pg_wait_tracer`, if you own the host
- Review and `support the patch`, so one day this works everywhere.

Ask what PostgreSQL is waiting for

- Don't just ask: "Which query is slow?"
- Ask: "What is PostgreSQL waiting for?"



Connect

[linkedin.com/in/dmitry-fomin-a88b7b45](https://www.linkedin.com/in/dmitry-fomin-a88b7b45)



pg_wait_tracer

github.com/DmitryNFomin/pg_wait_tracer

Thanks!
Questions?



Review the patch

commitfest.postgresql.org/patch/6984



Rate this talk

postgresql.eu/feedback 7907