A SHARED TRACING LAYER FOR POSTGRESQL EXTENSIONS

Distributed tracing through Postgres and the extensions you run

PGDU · 30 October 2026

Craig Ringer, EDB

EDB POSTGRES AI

©EDB 2026 — ALL RIGHTS RESERVED.

I WORK ON OBSERVABILITY FOR THE DATABASE AND THE LAYERS UNDER IT

Craig Ringer

EDB
Cloud database management

Observability for the database, and for Istio, Kubernetes and the other layers under it.

In the Postgres community since 2008; first contribution committed 2010.

2ndQuadrant -> EDB; BDR & logical replication

©EDB 2026 — ALL RIGHTS RESERVED.

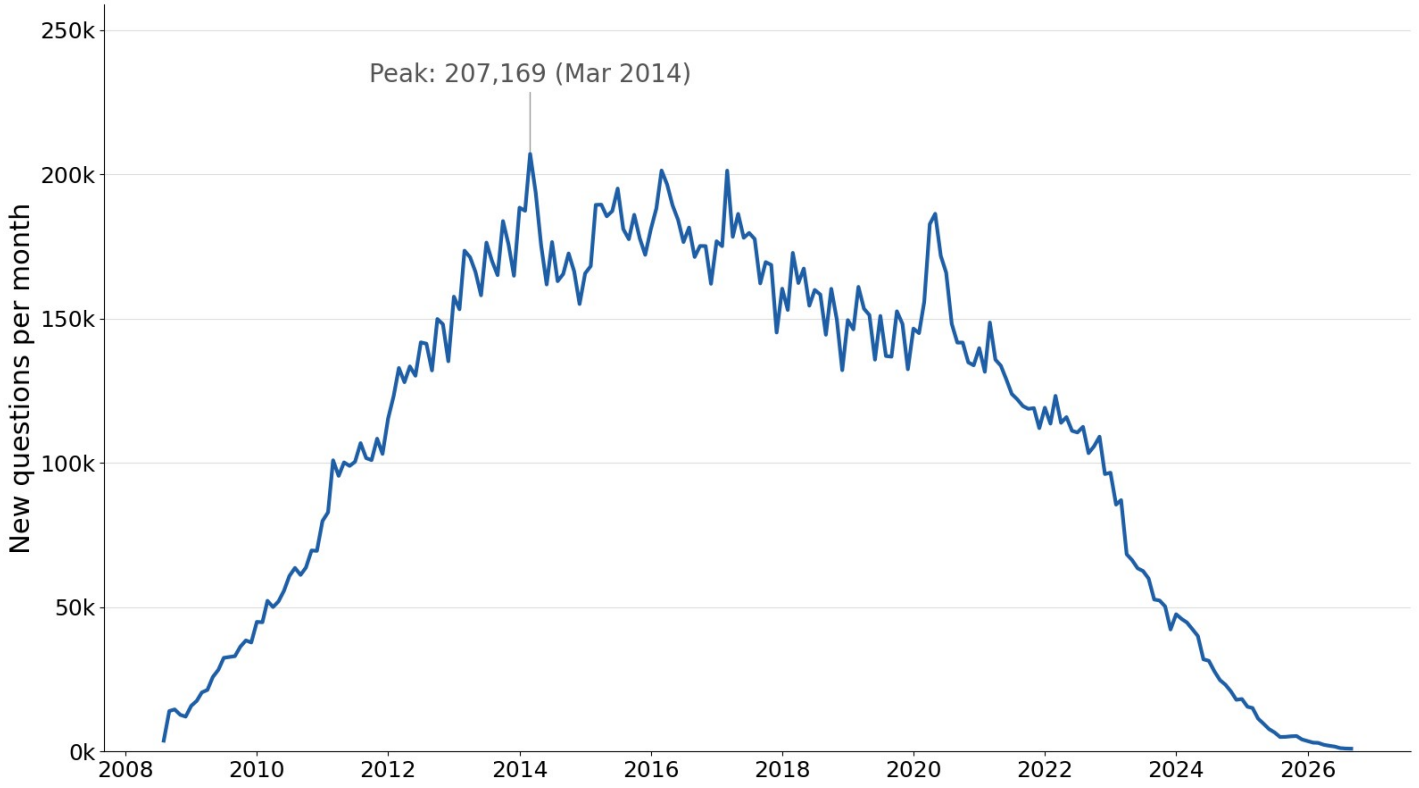
YOU'VE PROBABLY USED MY STACKOVERFLOW ANSWERS

They even sent me free stuff

©EDB 2026 — ALL RIGHTS RESERVED.

STACK OVERFLOW NEW QUESTIONS PER MONTH

Pity about that; anyway...



Stack Overflow new questions per month. Data: Stack Exchange API, CC BY-SA 4.0; chart by Craig Ringer.

©EDB 2026 — ALL RIGHTS RESERVED.

A LOT OF MY WORK HAS BEEN ON DISTRIBUTED SYSTEMS

BDR, now EDB Postgres Distributed

Logical replication features in Postgres

The first version of libpq's pipeline mode

Lately: Contributing to ClickHouse

©EDB 2026 — ALL RIGHTS RESERVED.

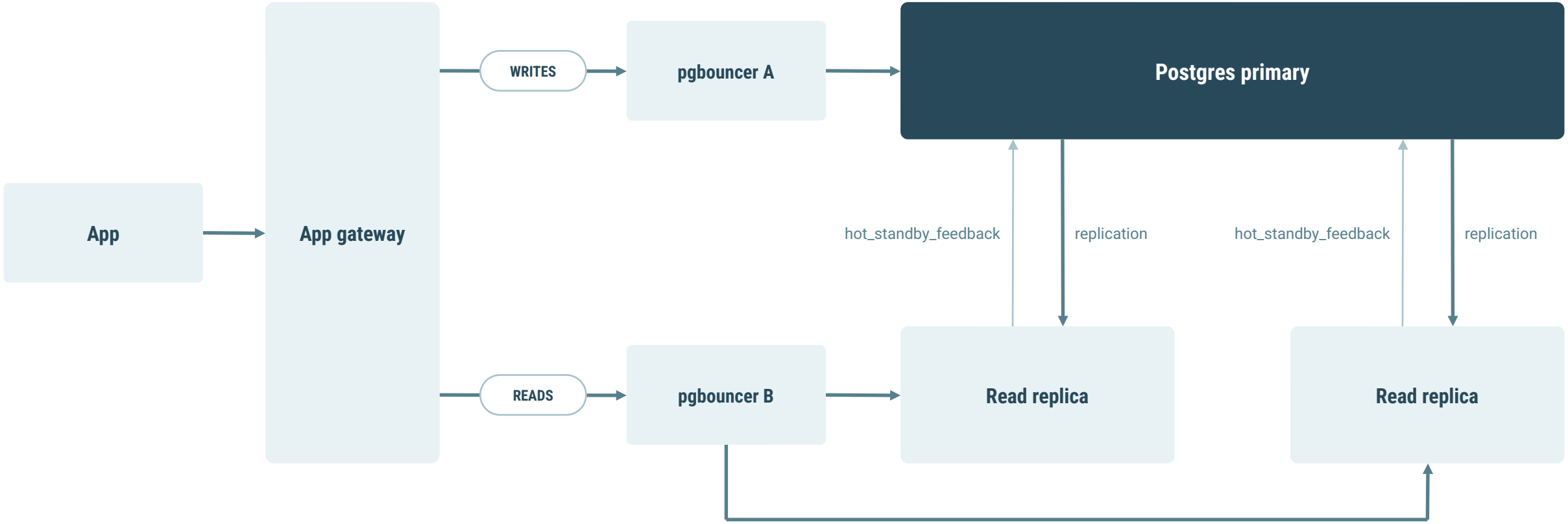
WHO HERE WORKS ON DISTRIBUTED SYSTEMS?

Who here works on distributed systems?

©EDB 2026 — ALL RIGHTS RESERVED.

MOST OF YOU DO

Most apps are now distributed systems.



EDB POSTGRES AI

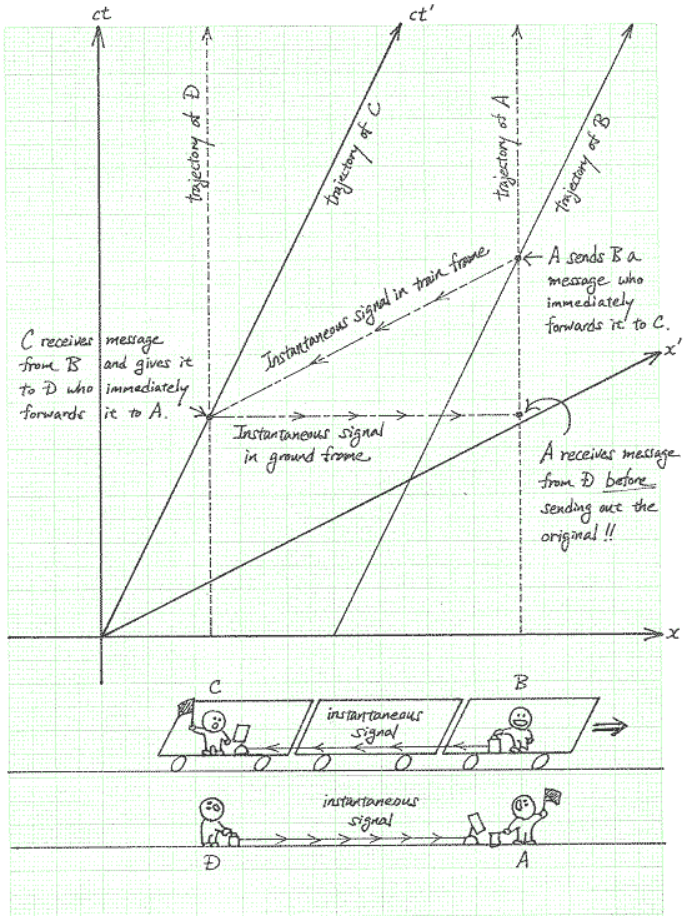
©EDB 2026 — ALL RIGHTS RESERVED.

DISTRIBUTED SYSTEMS YOU ALREADY RUN

Read replicas	**Synchronous replication**	**Multi-region HA and failover**	**Memcached or Redis cache**	**Logical replication**
Foreign data wrappers postgres_fdw	**Cloud DBs with off-node storage**	**Object-store table access methods** pg_ducklake, EDB PGAA	**Extensions with heavy network calls** pg_duckdb	**The client/server connection**

©EDB 2026 — ALL RIGHTS RESERVED.

DISTRIBUTED SYSTEMS ARE COMPLICATED



Tatsu Takeuchi, Virginia Tech, © 2005 Tatsu Takeuchi.

Is there even a “now”?

Speed of light = speed of information

CAP, PACELC, letter-salad

Raft and Paxos

Even more things you don't want to care about

EDB POSTGRES AI

©EDB 2026 — ALL RIGHTS RESERVED.

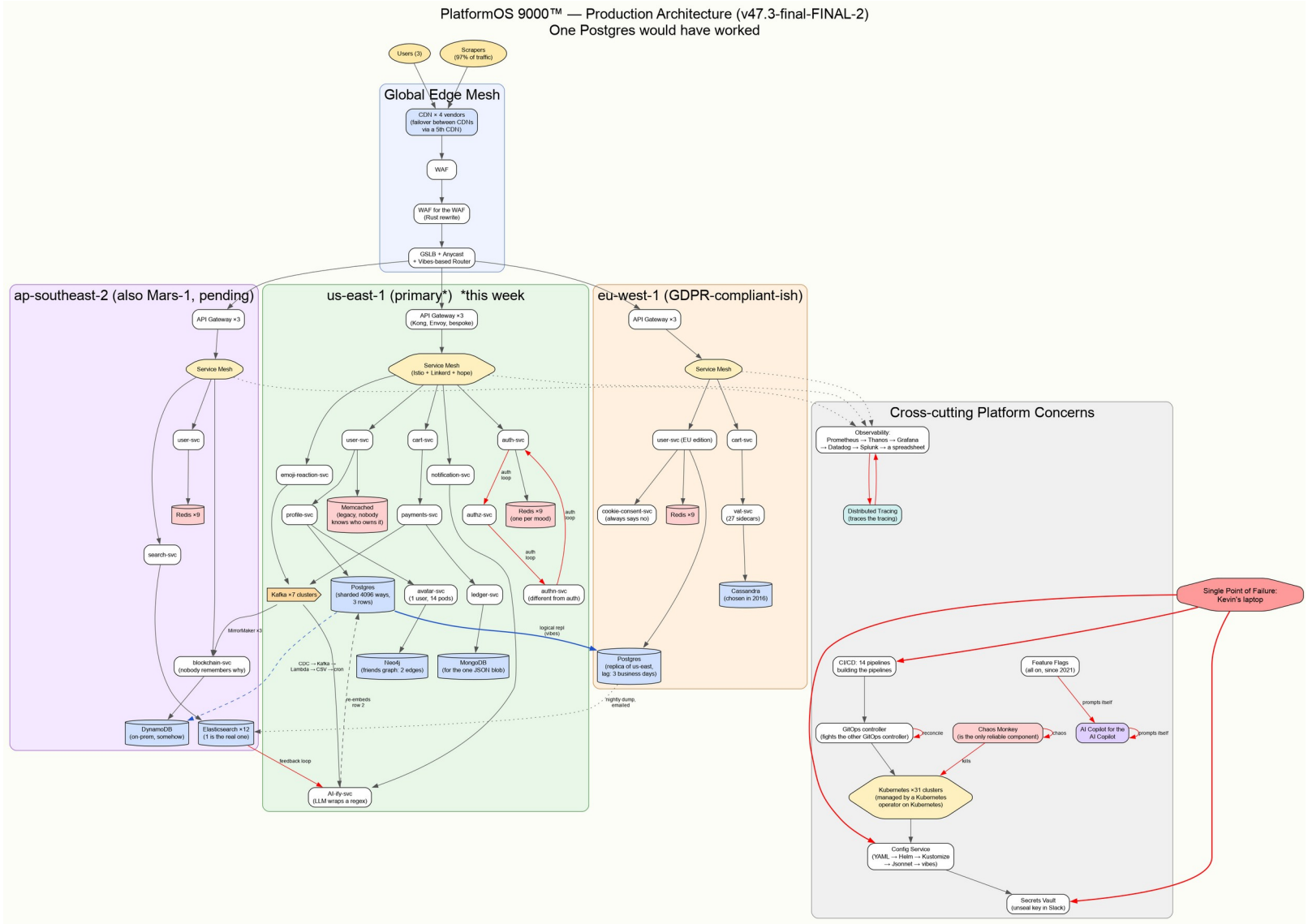
ON-CALL

2am, or halfway through cooking dinner.

“The invoices page is really slow today.”

©EDB 2026 — ALL RIGHTS RESERVED.

EVERYBODY LOVES MICROSERVICES, RIGHT?



©EDB 2026 — ALL RIGHTS RESERVED.

EVERY LAYER NAMES THE SAME REQUEST DIFFERENTLY

```
# gateway (nginx)
10.2.4.17 - - [27/Oct/2026:02:13:07 +0000] "PUT /api/orders/42 HTTP/1.1" 200 412 rt=1.231 req_id=7f3a9c

# orders service (JSON)
{"ts":"2026-10-27T02:13:07.114Z","level":"info","svc":"orders","requestId":"7f3a9c","msg":"update order","order":42}

# pgbouncer
2026-10-27 02:13:07.118 UTC [1] LOG C-0x55d1c0: app/orders@10.2.5.3:51432 login attempt: db=app user=orders tls=no

# postgres
2026-10-27 02:13:08.341 UTC [26560] LOG:  duration: 1117.204 ms  statement: UPDATE orders SET status = $1 WHERE id = $2
```

Illustrative, not captured

Many layers and hops, and every layer logs a different request identifier.

©EDB 2026 — ALL RIGHTS RESERVED.

EVERY LAYER NAMES THE SAME REQUEST DIFFERENTLY

```
# gateway (nginx)
10.2.4.17 - - [27/Oct/2026:02:13:07 +0000] "PUT /api/orders/42 HTTP/1.1" 200 412 rt=1.231 req_id=7f3a9c

# orders service (JSON)
{"ts":"2026-10-27T02:13:07.114Z","level":"info","svc":"orders","requestId":"7f3a9c","msg":"update order","order":42}

# pgbouncer
2026-10-27 02:13:07.118 UTC [1] LOG C-0x55d1c0: app/orders@10.2.5.3:51432 login attempt: db=app user=orders tls=no

# postgres
2026-10-27 02:13:08.341 UTC [26560] LOG:  duration: 1117.204 ms  statement: UPDATE orders SET status = $1 WHERE id = $2
```

Illustrative, not captured

req_id, then requestId, then nothing, then a pid.

©EDB 2026 — ALL RIGHTS RESERVED.

LOGS DON'T GET YOU THERE

FAN-OUT

Requests fan out to several systems at once, so the logs interleave.

COST

Detailed logging is expensive, so it is off until after the incident.

POINTS IN TIME

A log line has no start and end, and pairing them up is a chore.

©EDB 2026 — ALL RIGHTS RESERVED.

NO IDENTIFIER FOLLOWS A REQUEST THROUGH POSTGRES

```
02:13:07.118 [26560] LOG:  connection received: host=10.2.5.3 port=51432
02:13:07.121 [26560] LOG:  connection authenticated: user="orders" method=scram-sha-256
02:13:07.121 [26560] LOG:  connection authorized: user=orders database=app application_name=orders-svc
02:13:07.122 [26560] LOG:  connection ready: setup total=4.210 ms
02:13:07.137 [26571] LOG:  statement: SELECT * FROM orders WHERE id = 42 FOR UPDATE
02:13:07.140 [26560] LOG:  statement: BEGIN
02:13:07.141 [26560] LOG:  statement: INSERT INTO audit (order_id, action) VALUES (42, 'update')
02:13:07.173 [26560] LOG:  statement: UPDATE orders SET status = 'shipped' WHERE id = 42
02:13:08.174 [26560] LOG:  process 26560 still waiting for ShareLock on transaction 756 after 1000.044 ms
02:13:08.174 [26560] DETAIL:  Process holding the lock: 26571. Wait queue: 26560.
02:13:08.174 [26560] CONTEXT:  while updating tuple (0,42) in relation "orders"
02:13:08.290 [26560] LOG:  process 26560 acquired ShareLock on transaction 756 after 1116.871 ms
```

Illustrative, PG18 format

pid, backend id, vxid, xid, client address, application_name: none of them follows every path.
With verbose logging on, the signal is buried.

©EDB 2026 — ALL RIGHTS RESERVED.

VIEWS AND METRICS TELL YOU HOW MANY, NOT WHICH

```
  pid  |        state        | wait_event_type |  wait_event   |                    query
-------+---------------------+-----------------+---------------+-----------------------------------------------
 26560 | active              | Lock            | transactionid | UPDATE orders SET status = 'shipped' WHE...
 26571 | idle in transaction | Client          | ClientRead    | SELECT * FROM orders WHERE id = 42 FOR...
```

Illustrative

Catalog views are snapshots of one node.
Metrics built from them are aggregates.

©EDB 2026 — ALL RIGHTS RESERVED.

VIEWS AND METRICS TELL YOU HOW MANY, NOT WHICH

```
  pid  |        state        | wait_event_type |  wait_event   |                    query
-------+---------------------+-----------------+---------------+-----------------------------------------------
 26560 | active              | Lock            | transactionid | UPDATE orders SET status = 'shipped' WHE...
 26571 | idle in transaction | Client          | ClientRead    | SELECT * FROM orders WHERE id = 42 FOR...
```

Illustrative

Catalog views are snapshots of one node.
Metrics built from them are aggregates.

P99 LATENCY



Illustrative

which requests?

©EDB 2026 — ALL RIGHTS RESERVED.

THE LOGGING I WISH I HAD

01 **Switch it on for one customer, one feature or one request.**

©EDB 2026 — ALL RIGHTS RESERVED.

THE LOGGING I WISH I HAD

01 Switch it on for one customer, one feature or one request.

02 **Outbound and inbound hops recorded as matching pairs.**

©EDB 2026 — ALL RIGHTS RESERVED.

THE LOGGING I WISH I HAD

01 Switch it on for one customer, one feature or one request.

02 Outbound and inbound hops recorded as matching pairs.

03 **Reliable start and end times.**

©EDB 2026 — ALL RIGHTS RESERVED.

THE LOGGING I WISH I HAD

01 Switch it on for one customer, one feature or one request.

02 Outbound and inbound hops recorded as matching pairs.

03 Reliable start and end times.

04 **Rich annotations: client address, user, cache hit or miss, I/O cost.**

©EDB 2026 — ALL RIGHTS RESERVED.

THE LOGGING I WISH I HAD

01 Switch it on for one customer, one feature or one request.

02 Outbound and inbound hops recorded as matching pairs.

03 Reliable start and end times.

04 Rich annotations: client address, user, cache hit or miss, I/O cost.

05 **Everything in one place.**

©EDB 2026 — ALL RIGHTS RESERVED.

THE LOGGING I WISH I HAD

01 Switch it on for one customer, one feature or one request.

02 Outbound and inbound hops recorded as matching pairs.

03 Reliable start and end times.

04 Rich annotations: client address, user, cache hit or miss, I/O cost.

05 Everything in one place.

06 **Tools that correlate, flag errors and find trends for you.**

©EDB 2026 — ALL RIGHTS RESERVED.

THE LOGGING I WISH I HAD

01 Switch it on for one customer, one feature or one request.

02 Outbound and inbound hops recorded as matching pairs.

03 Reliable start and end times.

04 Rich annotations: client address, user, cache hit or miss, I/O cost.

05 Everything in one place.

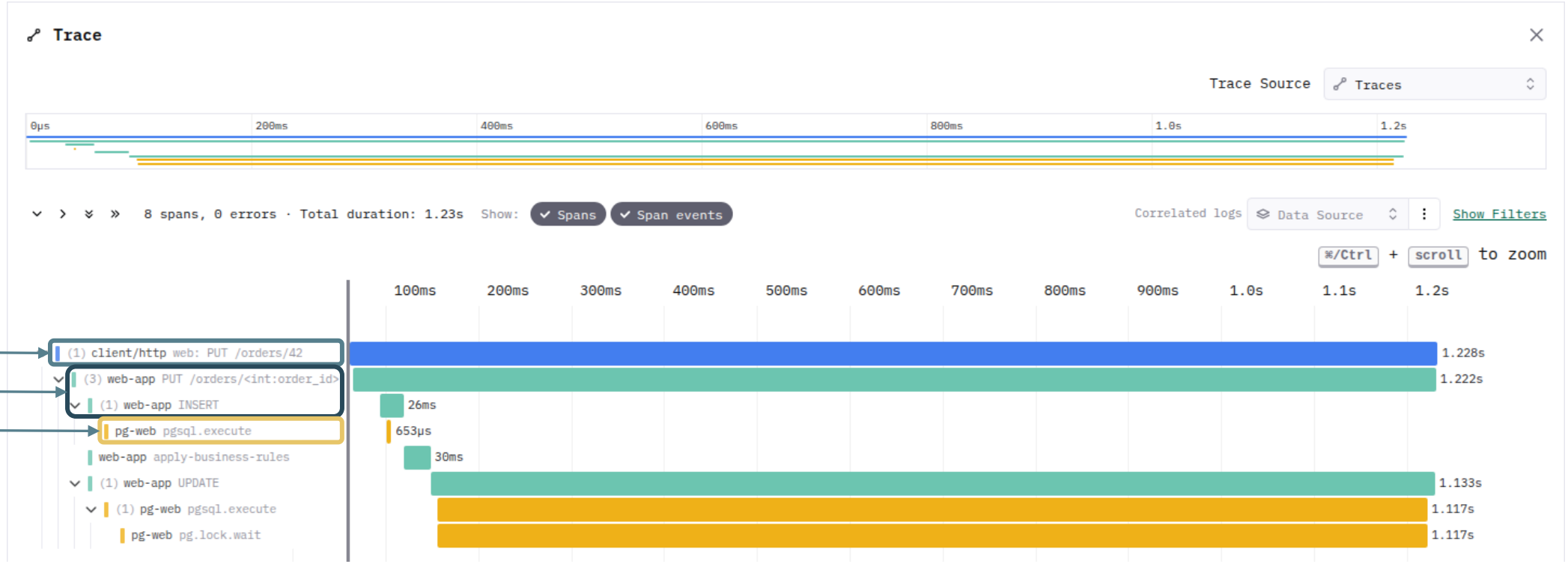
06 Tools that correlate, flag errors and find trends for you.

That's what distributed tracing is supposed to provide.

©EDB 2026 — ALL RIGHTS RESERVED.

A TRACE FOLLOWS ONE REQUEST THROUGH THE WHOLE STACK

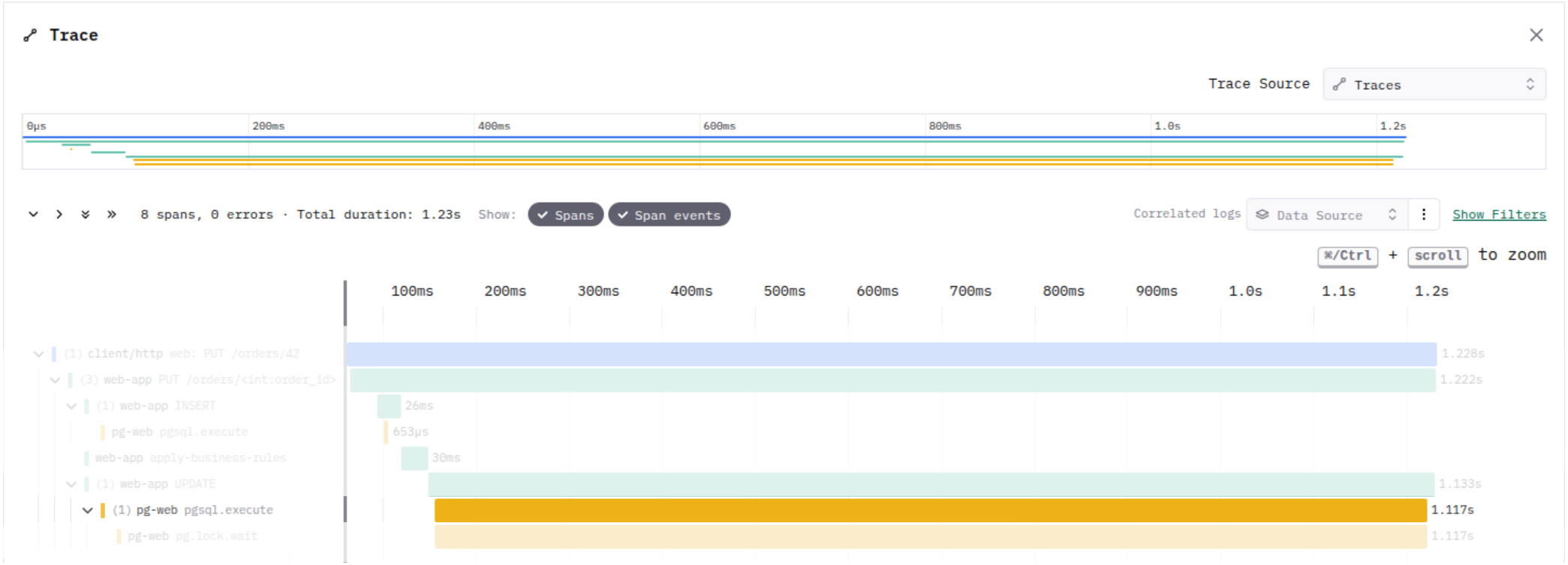
PATCHED CORE



HyperDX

©EDB 2026 — ALL RIGHTS RESERVED.

A SPAN IS ONE TIMED STEP, HERE A 1.1 S UPDATE



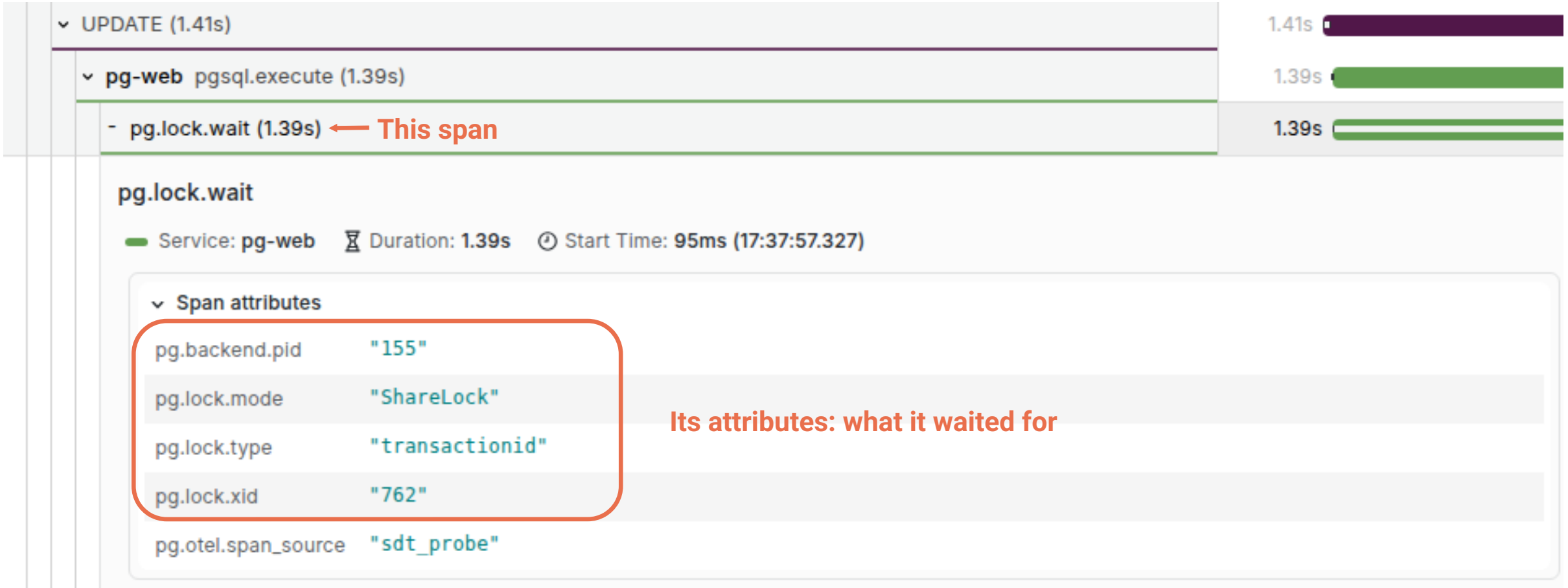
HyperDX

©EDB 2026 — ALL RIGHTS RESERVED.

SPANS CARRY CONTEXTUAL ATTRIBUTES

PATCHED CORE

E.g. details on a lock wait



Grafana

©EDB 2026 — ALL RIGHTS RESERVED.

A LOG LINE LEADS YOU TO ITS TRACE

NEEDS NO PATCHES

With a trace active, log lines carry the trace and span ID.

```
2026-10-25 02:59:40.425 UTC [26563] ERROR:  duplicate key value violates unique constraint "orders_pkey"
2026-10-25 02:59:40.425 UTC [26563] DETAIL:  Key (id)=(42) already exists.
2026-10-25 02:59:40.425 UTC [26563] CONTEXT:  trace_id=756841cca92ab4cd826206caaffaf24d span_id=83c89bd22c936622 trace_flags=01
2026-10-25 02:59:40.425 UTC [26563] STATEMENT:  INSERT INTO orders (id, status) VALUES ($1, $2) /*…traceparent='00-756841cc…'*/
```

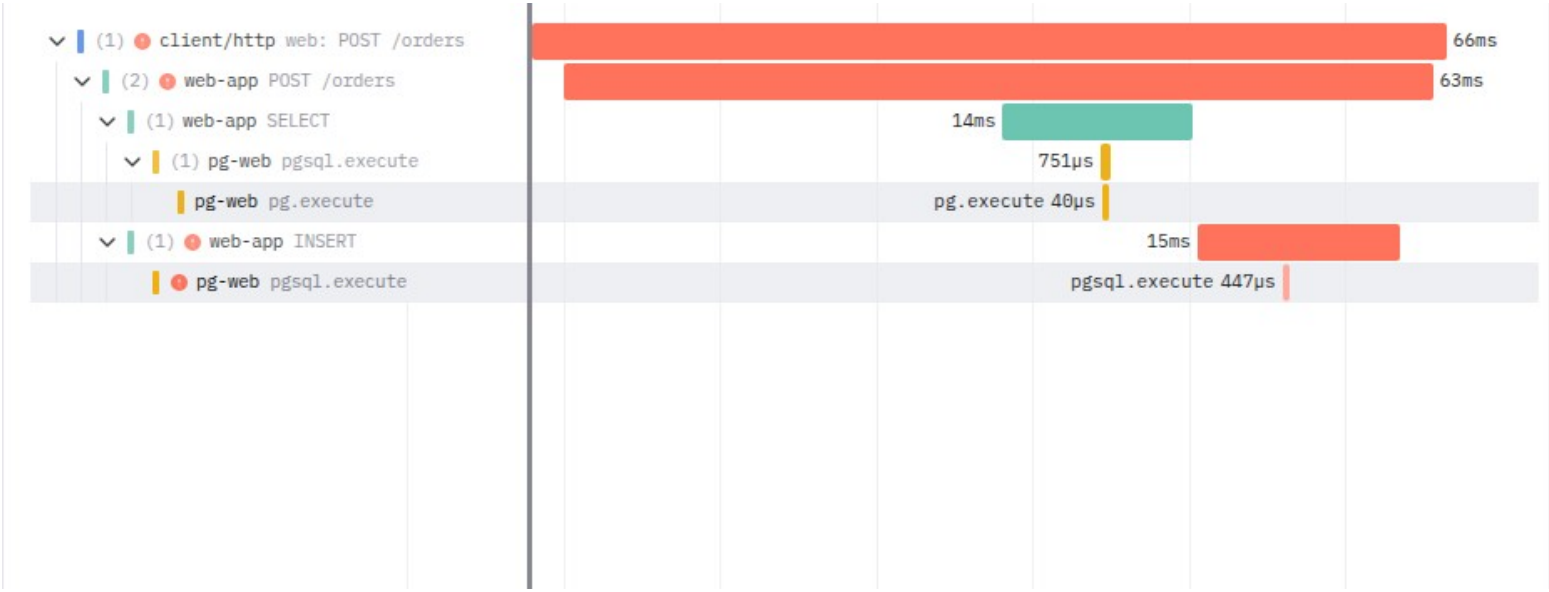
An optional core patch gives the trace ID its own field.

©EDB 2026 — ALL RIGHTS RESERVED.

AN ERROR IS RECORDED ON ITS SPAN, AND ON EVERY SPAN ABOVE IT

NEEDS NO PATCHES

SQLSTATE, message and detail sit on the span that raised the error.





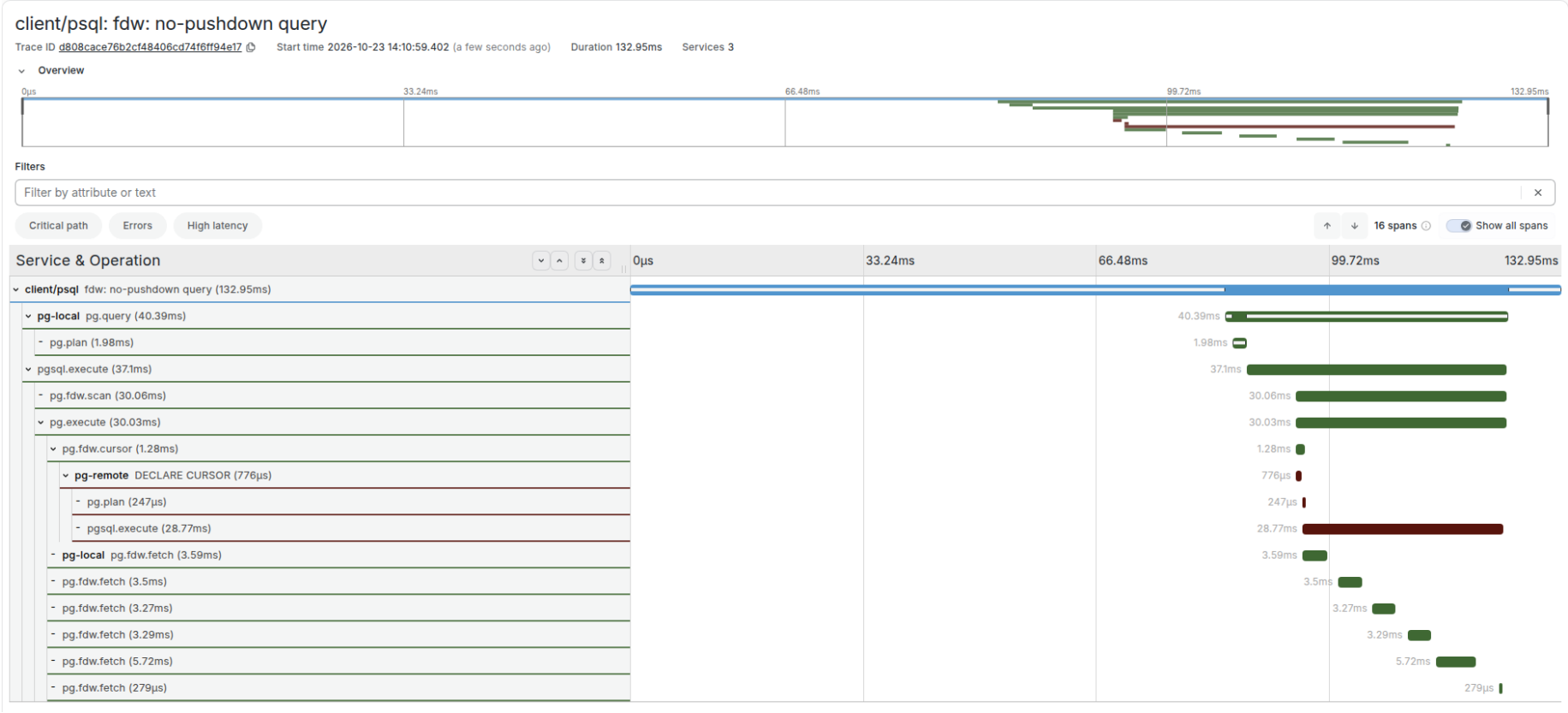
HyperDX

©EDB 2026 — ALL RIGHTS RESERVED.

THE TRACE SHOWS THE CRITICAL TIMING PATH

PATCHED CORE + PATCHED CONTRIB

A white line through a span marks it as on the critical timing path (Grafana).



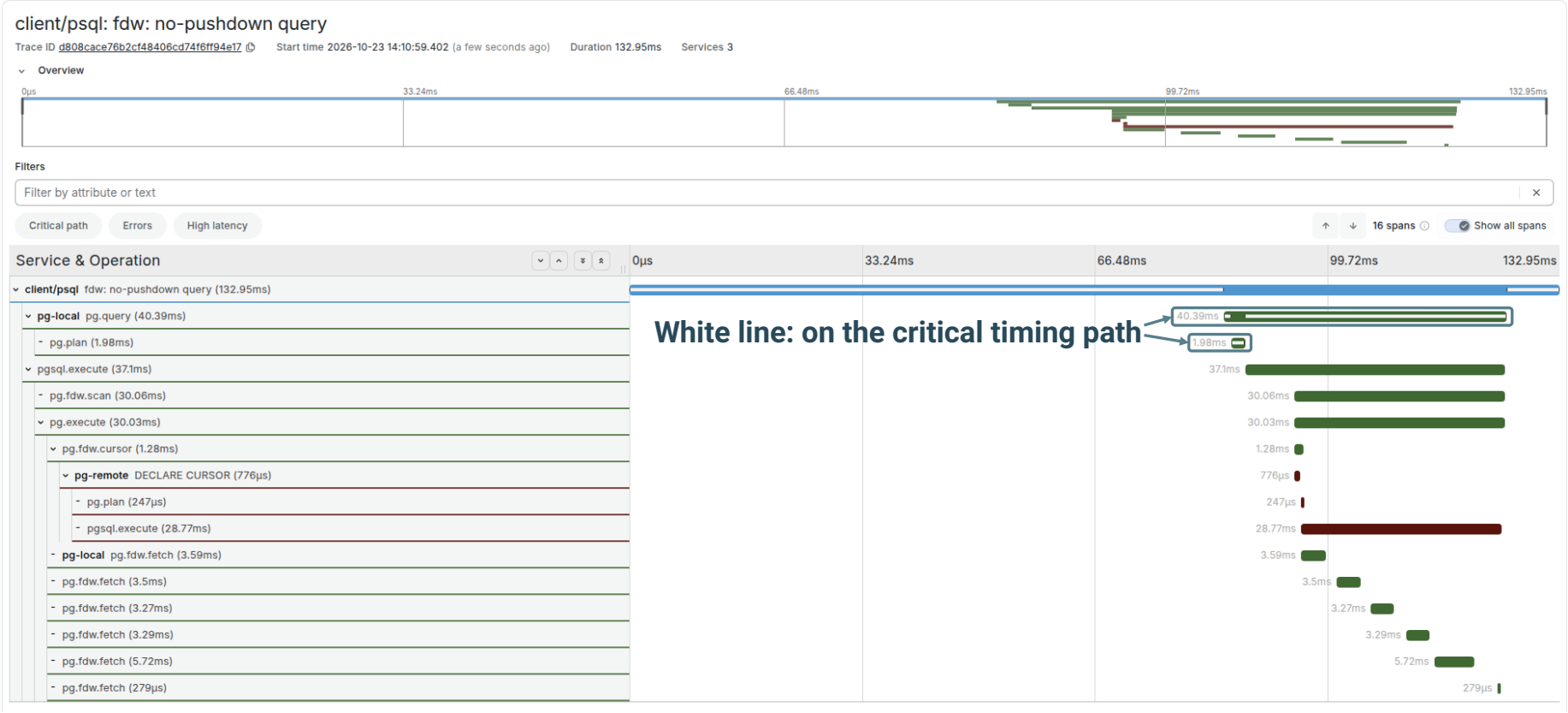
Grafana

©EDB 2026 — ALL RIGHTS RESERVED.

THE TRACE SHOWS THE CRITICAL TIMING PATH

PATCHED CORE + PATCHED CONTRIB

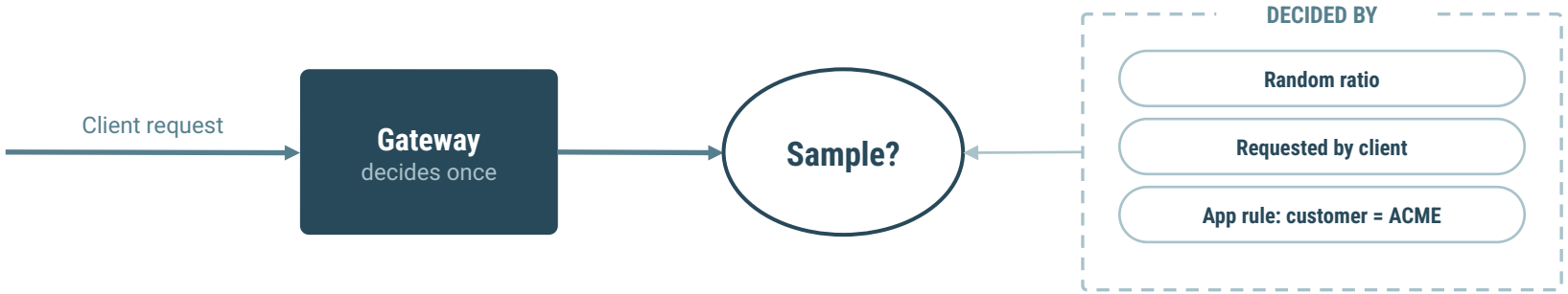
A white line through a span marks it as on the critical timing path (Grafana).



Grafana

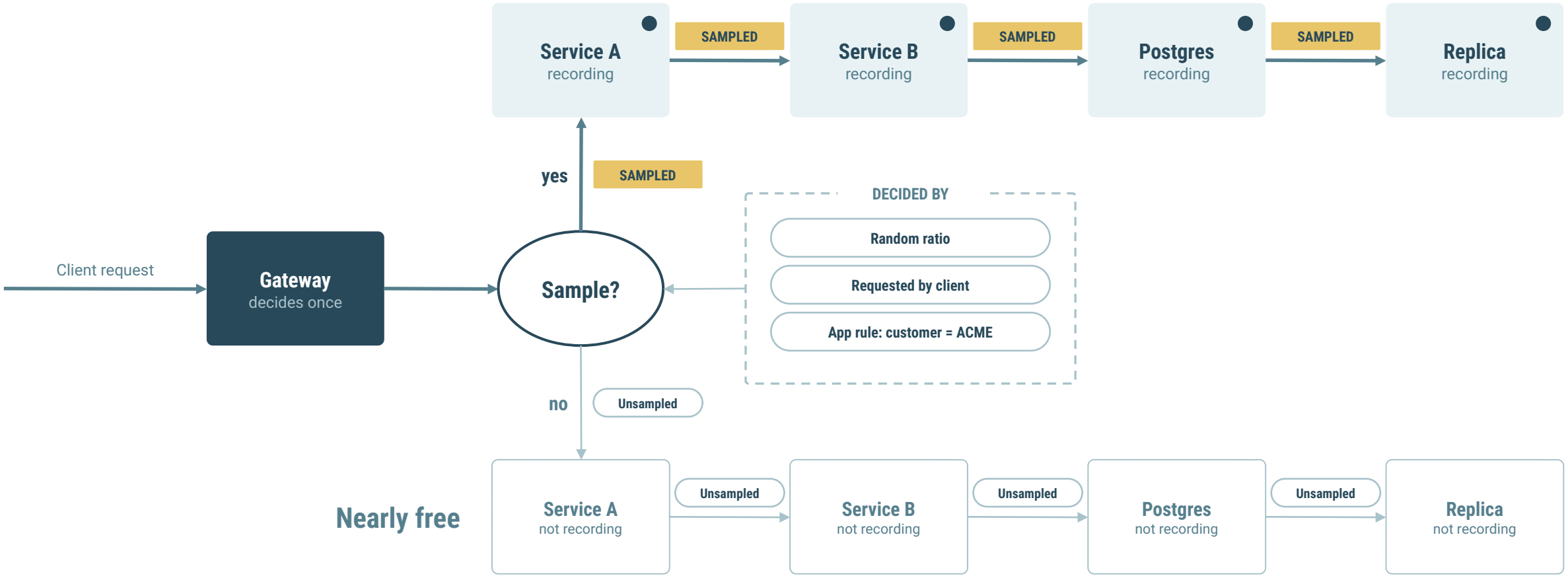
©EDB 2026 — ALL RIGHTS RESERVED.

TRACING IS SAMPLED, NOT ALWAYS ON



EDB POSTGRES AI
©EDB 2026 — ALL RIGHTS RESERVED.

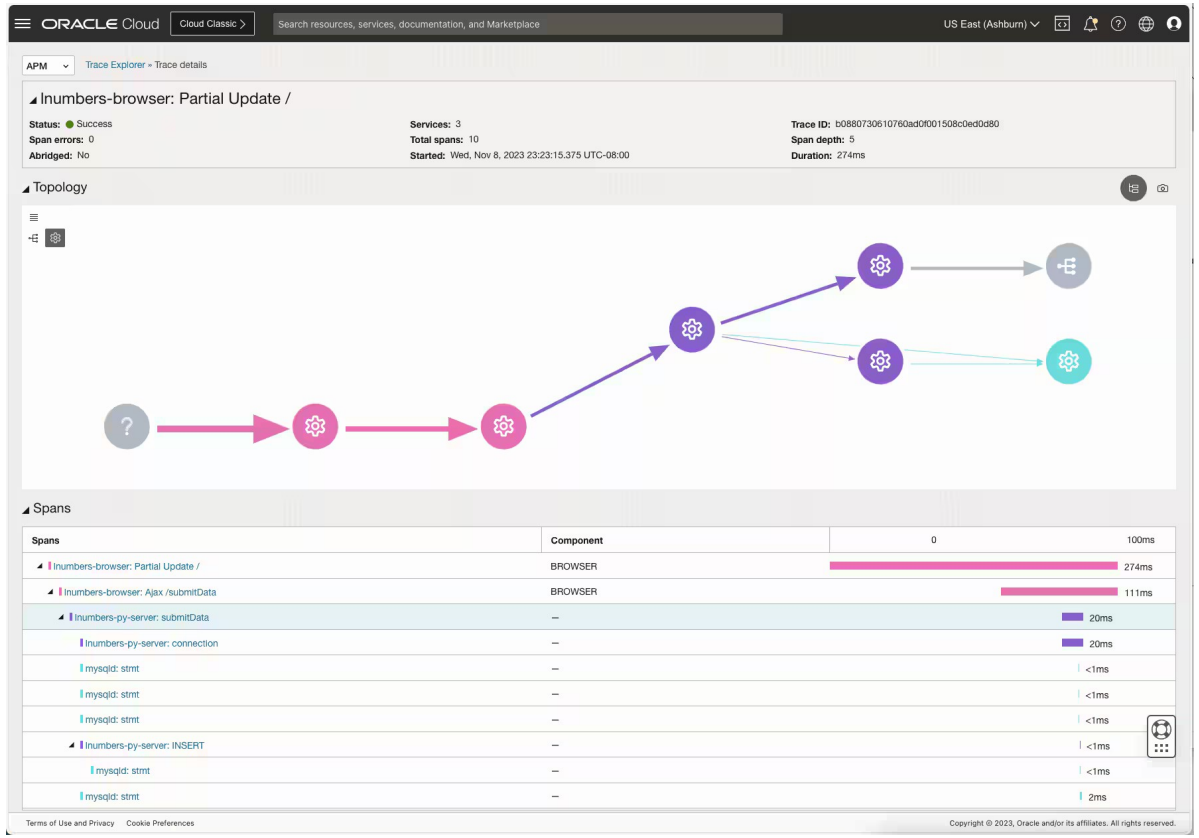
THE SAMPLING DECISION TRAVELS WITH THE REQUEST



©EDB 2026 — ALL RIGHTS RESERVED.

TRACING ISN'T NEW

MySQL Enterprise, traced in OCI APM

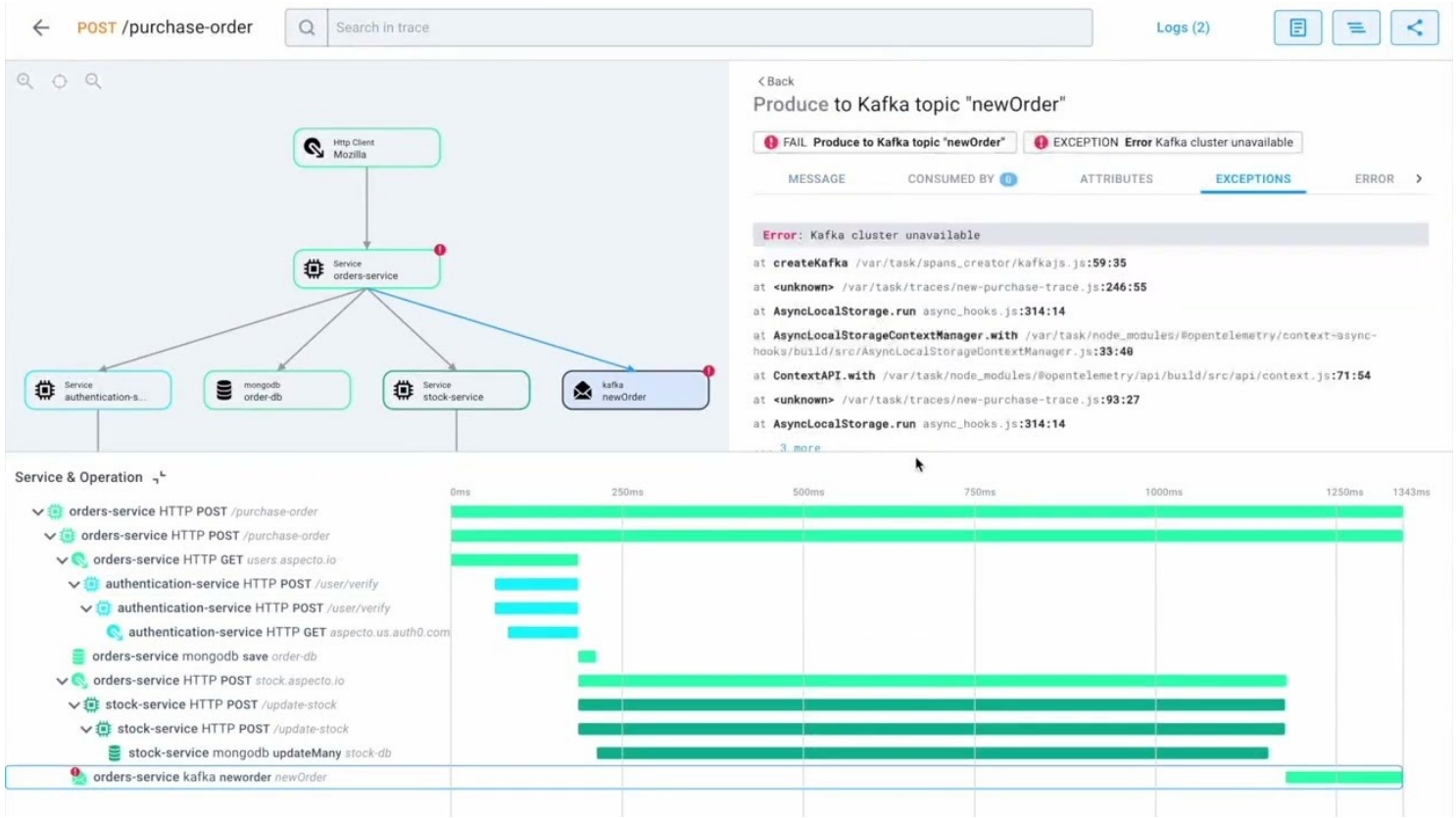


Oracle MySQL blog, Mike Frank and Miroslav Rajčić, © 2023

©EDB 2026 — ALL RIGHTS RESERVED.

TRACING ISN'T NEW

Kafka and MongoDB, traced in Aspecto

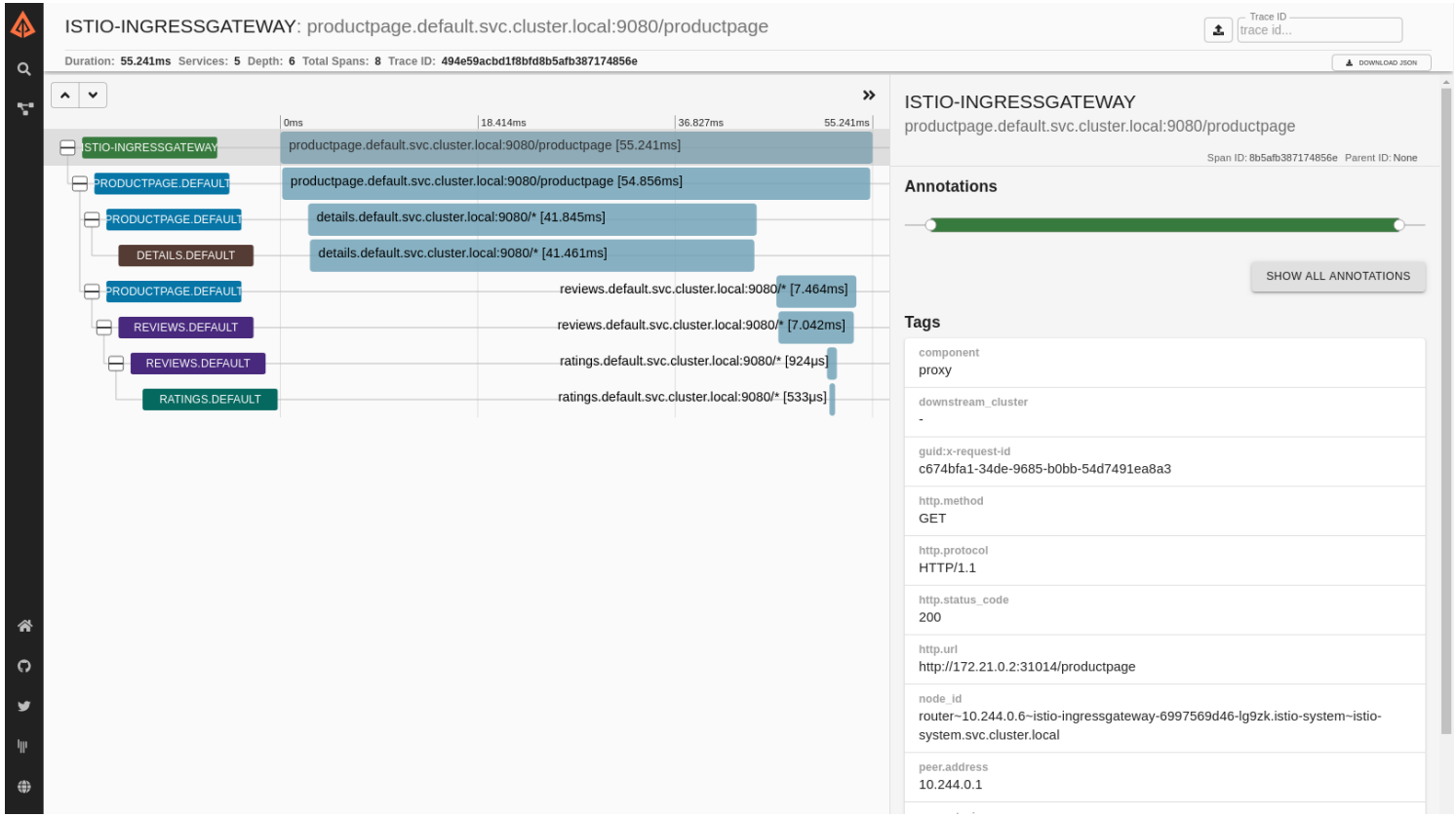


dev.to/aspecto, Motti Bechhofer at Aspecto, © 2023

©EDB 2026 — ALL RIGHTS RESERVED.

TRACING ISN'T NEW

The Istio service mesh, traced in Zipkin

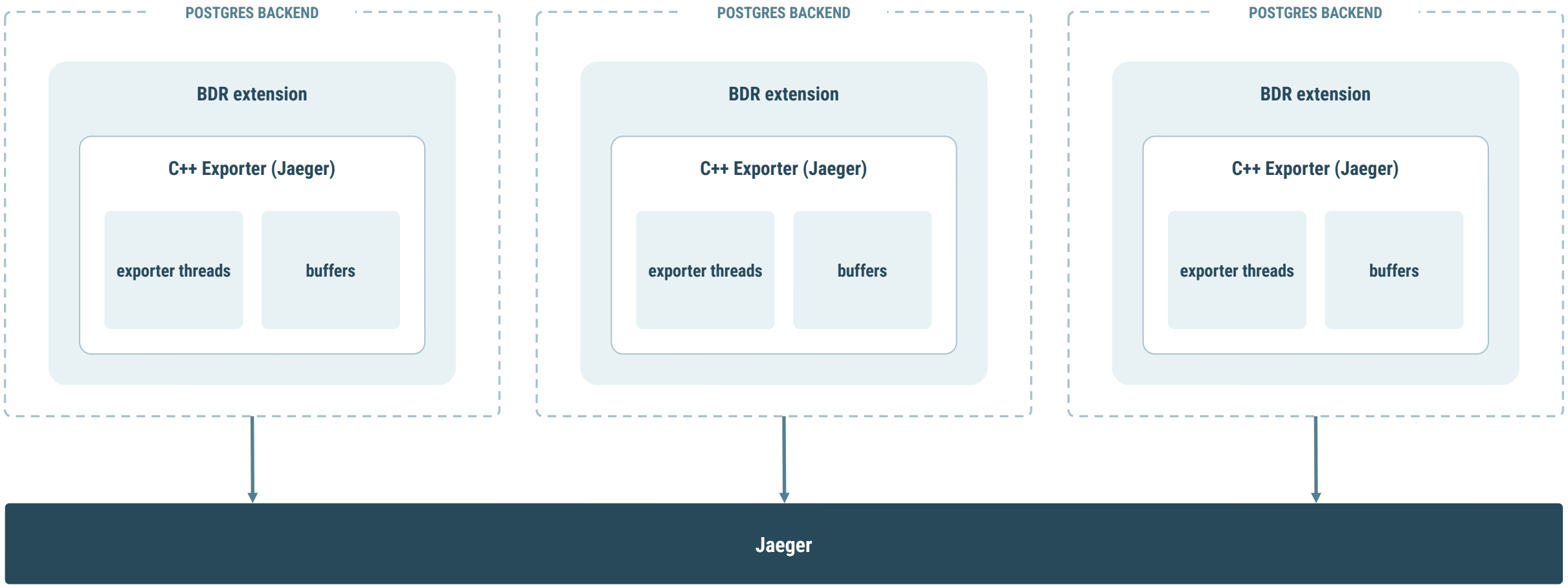


istio.io, Istio Authors, © 2026

©EDB 2026 — ALL RIGHTS RESERVED.

DISTRIBUTED TRACING OF POSTGRES IN BDR (2018)

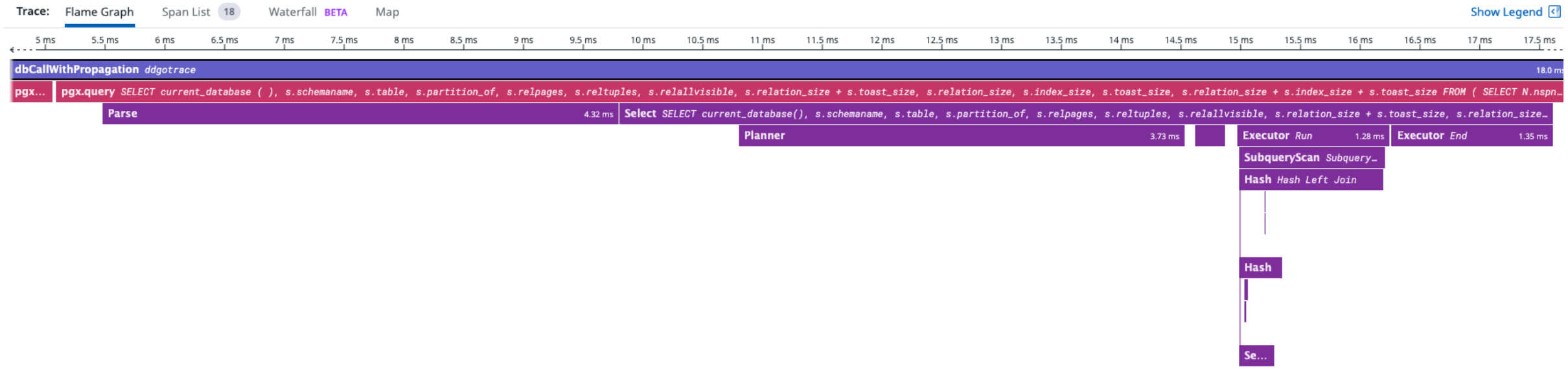
Every backend carried its own threaded C++ SDK



EDB POSTGRES AI

©EDB 2026 — ALL RIGHTS RESERVED.

PG_TRACING EXISTS, AND IT IS GOOD



pgsql-hackers post, Anthonin Bonnefoy at Datadog, © 2023

pg_tracing, by Anthonin Bonnefoy and others.
(pg_stat_ch is closer to stats logging, with some proto-tracing.)

©EDB 2026 — ALL RIGHTS RESERVED.

WHAT I WANTED TO SEE

Trace context from any client - efficiently

Long lock waits
mapped to the request

Sync-rep waits
mapped to the request

PL/pgSQL
nested queries and calls

TRACES THROUGH OTHER EXTENSIONS

See through FDWs
into the remote system

NOT YET

Object store / lakehouse
access time for lakehouse queries

NOT YET

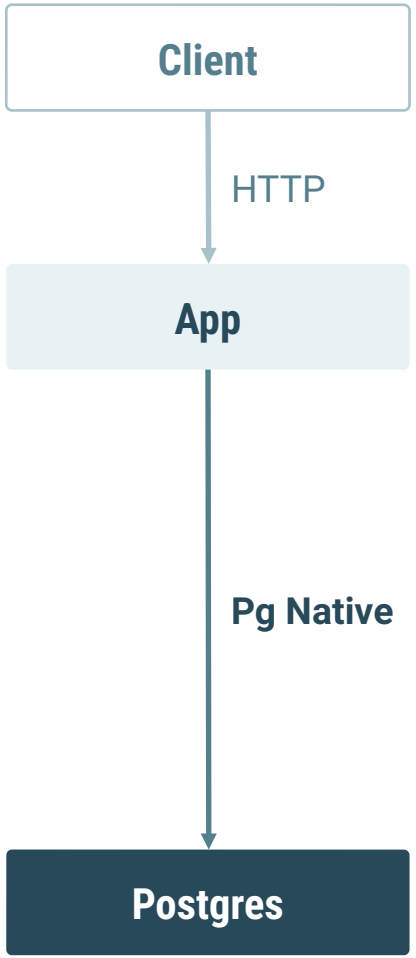
PGD / BDR flows
a request through a PGD cluster

©EDB 2026 — ALL RIGHTS RESERVED.

CONTEXT PROPAGATION

NEEDS NO PATCHES

The app's trace continues into Postgres



HTTP HEADER (W3C TRACEPARENT STANDARD)

```
PUT /orders/42 HTTP/1.1
traceparent: 00-4bf92f3577b34da6a3ce929d0e0e4736-00f067aa0ba902b7-01
```

trace id

parent span id (for nesting)

SQL COMMENT ON THE QUERY

```
UPDATE orders SET status = $1 WHERE id = $2
  /*traceparent='00-4bf92f3577b34da6a3ce929d0e0e4736-b7ad6b7169203331-01'*/
```

— OR —

A SETTING BEFORE THE QUERY

```
SET pg_tracing.trace_context =
  'traceparent=''00-4bf92f3577b34da6a3ce929d0e0e4736-b7ad6b7169203331-01''';
```

Same trace ID at every hop; the parent span ID is the caller's span.

EDB POSTGRES AI

©EDB 2026 — ALL RIGHTS RESERVED.

CONTEXT PROPAGATION IS AWKWARD

sqlcommenter fights client query caches and prepared statements.

```
INSERT INTO orders (id, status) VALUES ($1, $2)
/*controller='create_order',db_driver='psycopg%3A3.2.10',framework='flask%3A3.1.2',
  route='/orders',traceparent='00-756841cca92ab4cd826206caaffaf24d-87fd120092488e21-01'*/
```

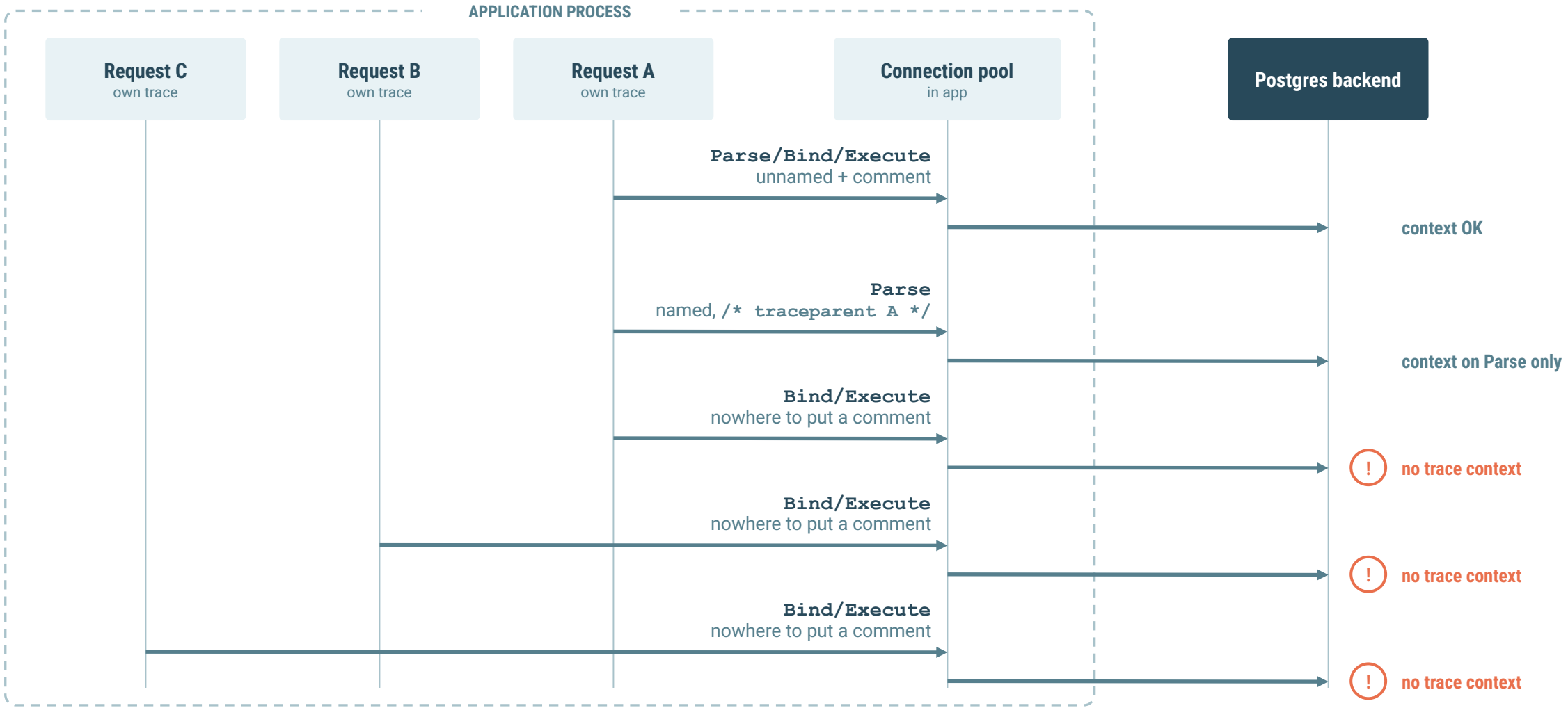
`SET pg_tracing.traceparent = '...'` adds round trips.

Other protocols carry trace context in a header.

©EDB 2026 — ALL RIGHTS RESERVED.

ONE PARSE, MANY EXECUTIONS, ONE COMMENT

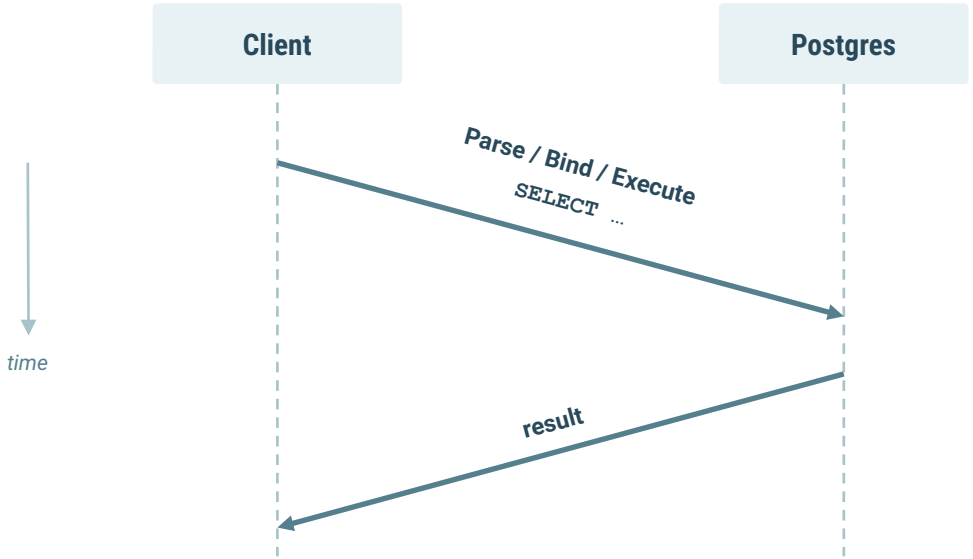
SQLCOMMENTER



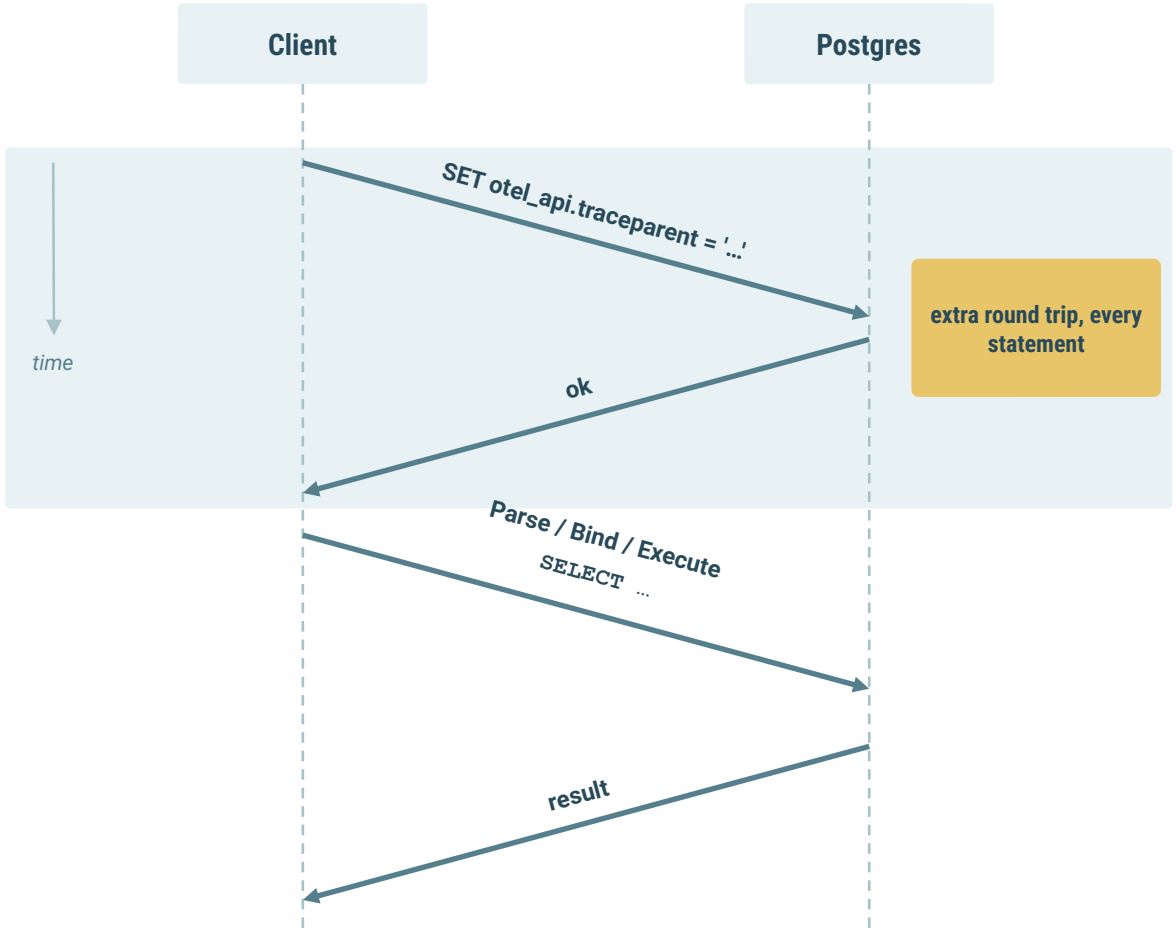
©EDB 2026 — ALL RIGHTS RESERVED.

SET COSTS A ROUND TRIP PER STATEMENT

NO TRACE CONTEXT



CONTEXT VIA SET



©EDB 2026 — ALL RIGHTS RESERVED.

THE SET ROUND TRIP SHOWS UP IN THROUGHPUT

POSTGRES_FDW SENDS CONTEXT BY	POINT-QUERY THROUGHPUT
SET before each query	about 13% fewer TPS, at any sample rate
SQL comment on the query	no measurable cost

From the otel_api benchmarks

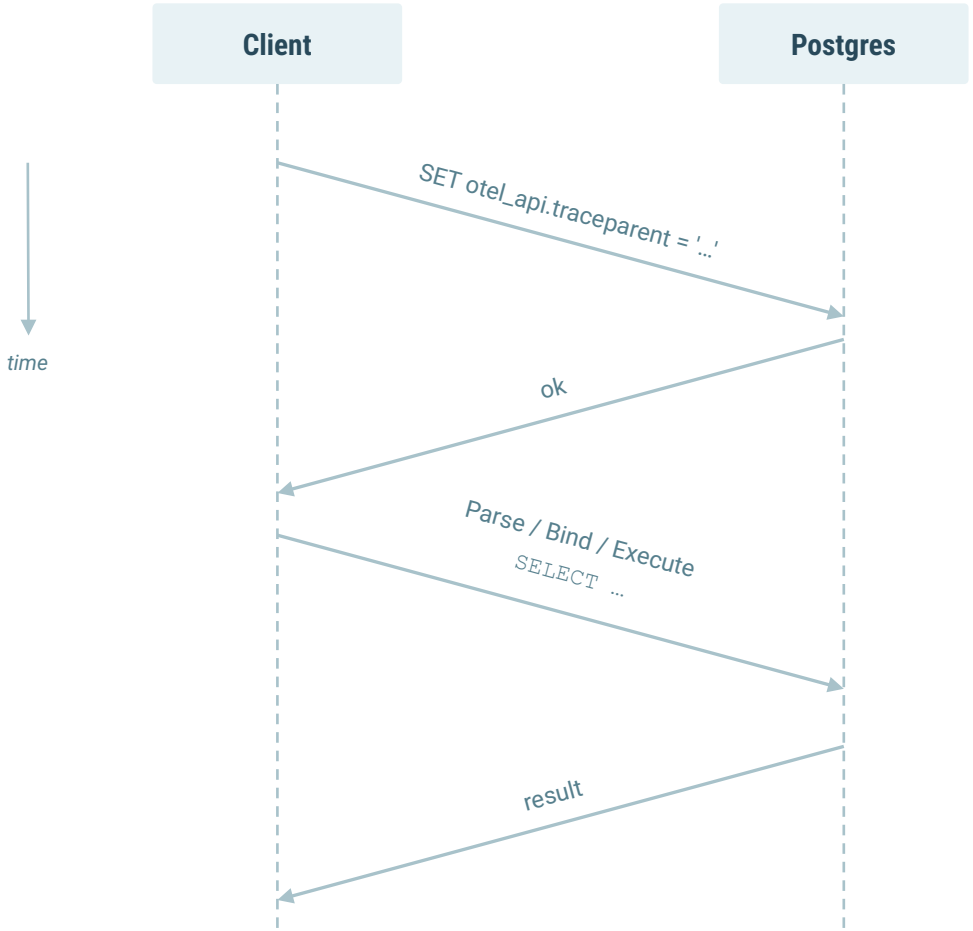
OTHER PROTOCOLS USE HEADERS FOR THIS

HTTP: traceparent header · gRPC: metadata

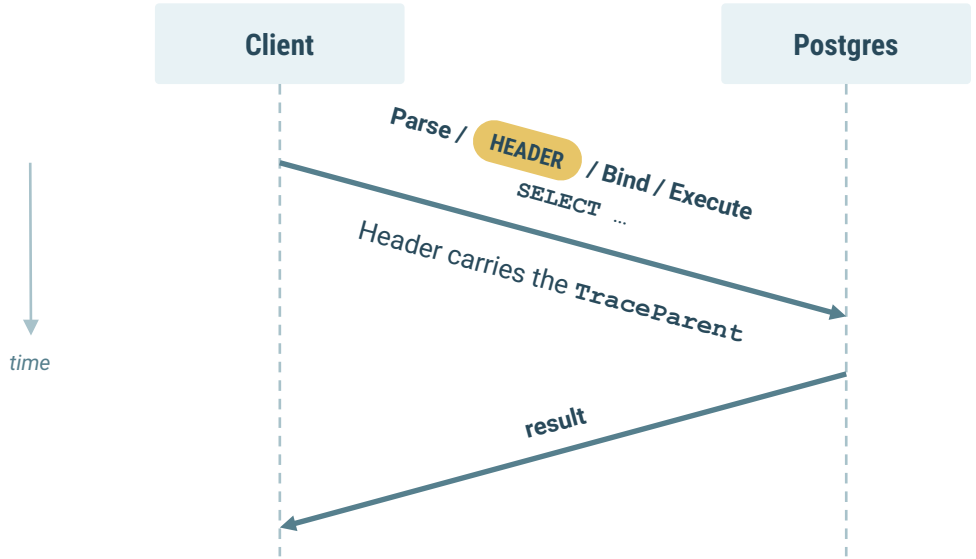
©EDB 2026 — ALL RIGHTS RESERVED.

THE WIRE PROTOCOL NEEDS A PLACE FOR TRACE CONTEXT

BEFORE: CONTEXT VIA SET



CONTEXT IN A PROTOCOL HEADER (OPTIONAL CORE PATCH)



Patches:

traceparent message: protocol-traceparent

generalised headers message: protocol-headers-generic

©EDB 2026 — ALL RIGHTS RESERVED.

CONTEXT ONLY MATTERS IF SOMETHING IS LISTENING

pg_tracing listens - if it's available

©EDB 2026 — ALL RIGHTS RESERVED.

EXTENSIONS AND PG_TRACING ARE MUTUALLY OBLIVIOUS

Other extensions can't emit spans for pg_tracing to export.
It sees them only through hooks in core.

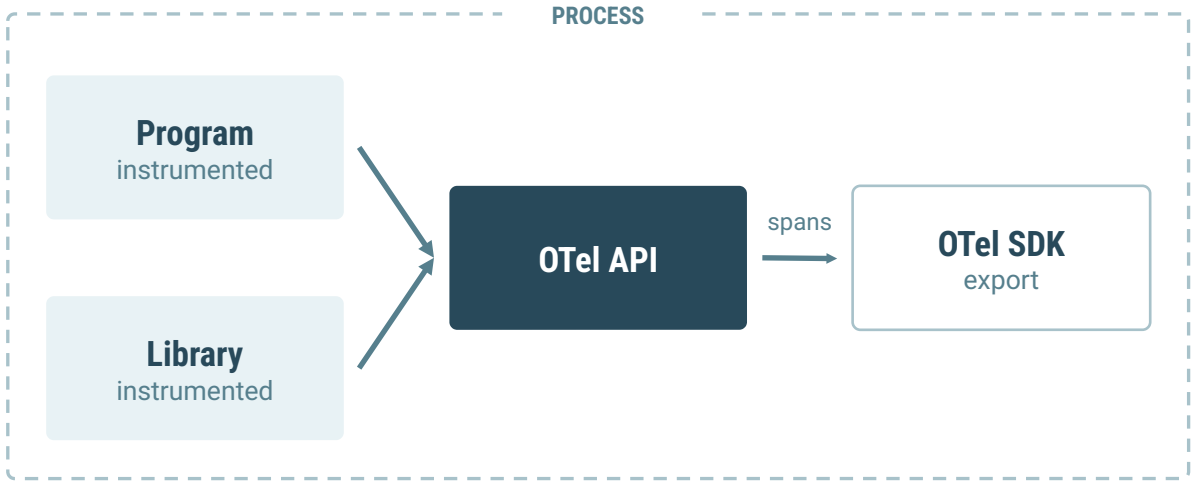
Carrying code for other extensions is a maintenance burden for it.

A direct dependency between them is a build and distribution problem.

©EDB 2026 — ALL RIGHTS RESERVED.

NO OPENTELEMETRY SDK FITS POSTGRES

OpenTelemetry standard design



The API is what instrumented code calls.

The SDK implements it and exports the data.

Postgres architecture incompatible
(pure C, multi-processing, no threads)

©EDB 2026 — ALL RIGHTS RESERVED.

TRACING SHOULD COME FROM POSTGRES, LIKE `ereport(...)` DOES

Extensions don't bring their own ereport or log_destination. Tracing should work the same way.

Nobody adopts tracing that needs core patches.



Core won't take a tracing API without users.

So: an out-of-tree API extension.

©EDB 2026 — ALL RIGHTS RESERVED.

`otel_api`: SPANS IN, COMMON SERVICES, NO TRACING OF ITS OWN

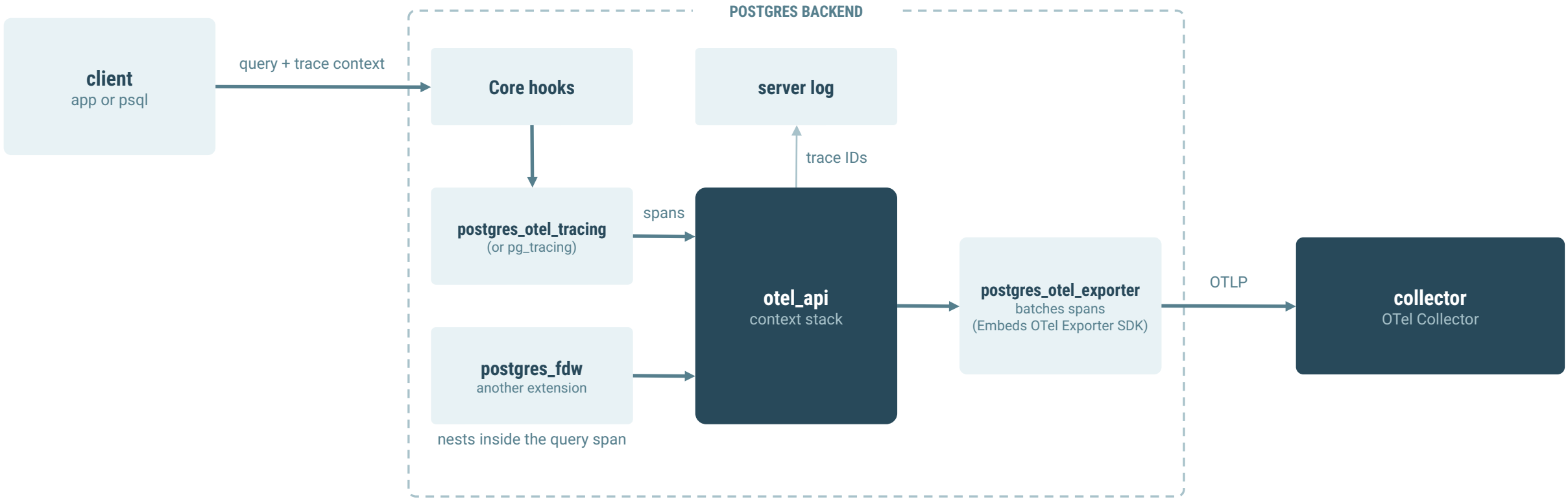
The OpenTelemetry API/SDK split, mapped to PostgreSQL

Trace* emitter API	Does no tracing itself
Exporter plugin API	Decouples heavy exporters
Common services	Context propagation, trace stack, ...
Stable, versioned API	Safe to depend on
Lightweight	Minimal, no external dependencies

* Metrics coming soon

©EDB 2026 — ALL RIGHTS RESERVED.

FROM CLIENT CONTEXT TO A COLLECTOR, THROUGH ONE API



©EDB 2026 — ALL RIGHTS RESERVED.

NO BUILD OR RUN-TIME DEPENDENCY ON `otel_api`

```
#include "otel_api/otel_producer.h"

static OtelTracer my_tracer = {.name = "my_ext"};

OtelSpanRef s = otel_span_start(.tracer = &my_tracer,
                                .name   = "my_ext.work");
/* ... work ... */
otel_span_end(s);
```

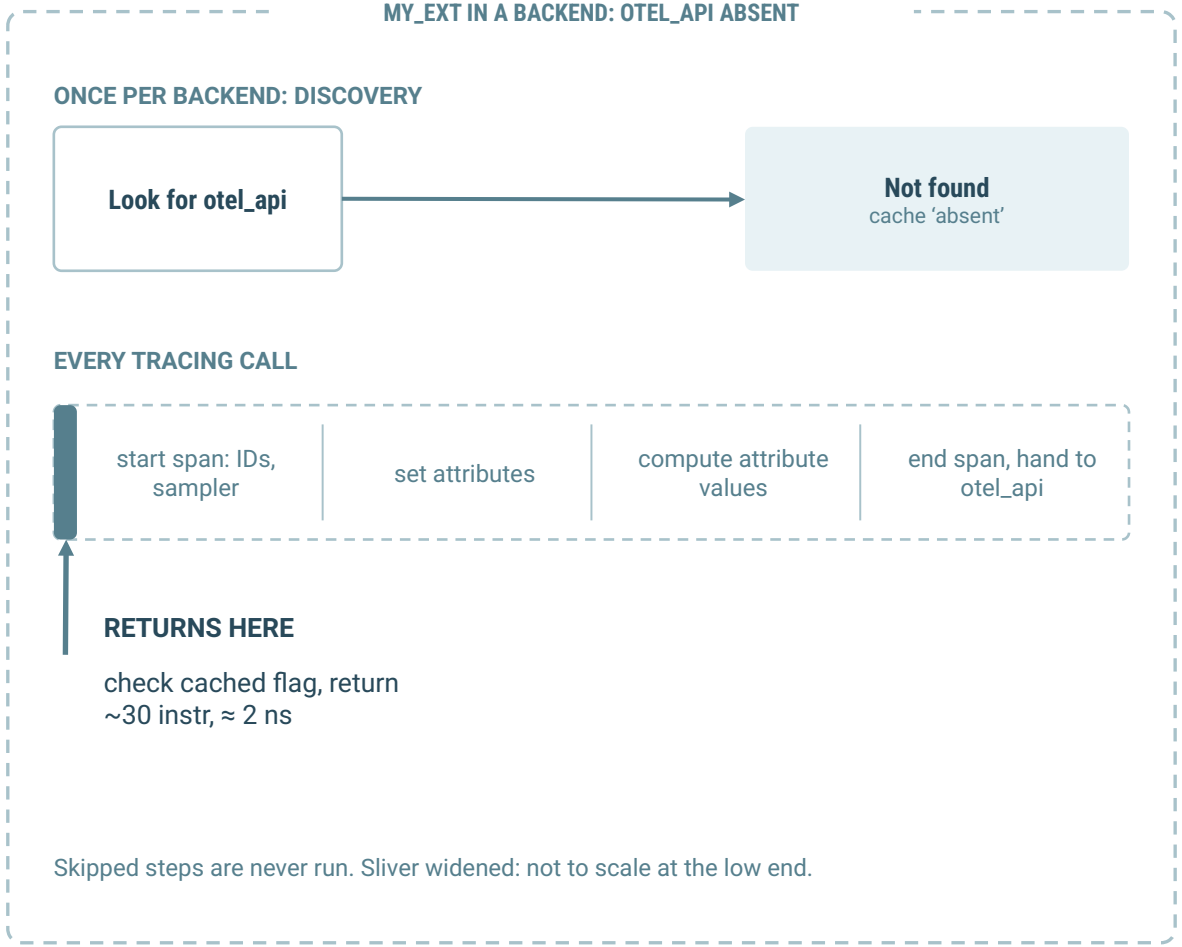
Vendorable header-only API

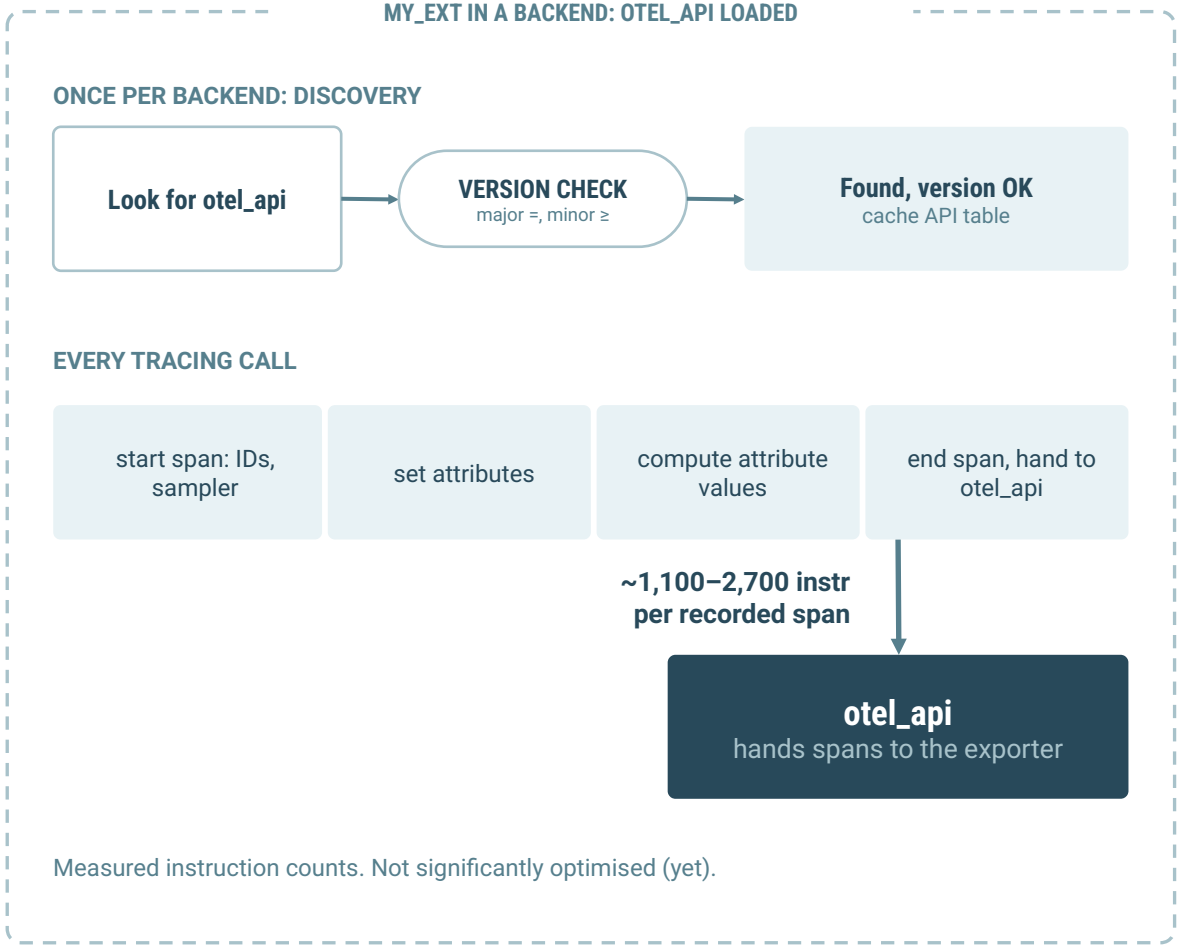
No build, link or run-time dependency

Or copy the no-op stub header: every call compiles to nothing.

©EDB 2026 — ALL RIGHTS RESERVED.

BUILT AGAINST `otel_api`, AN EXTENSION LOADS FINE WITHOUT IT





EDB POSTGRES AI

©EDB 2026 — ALL RIGHTS RESERVED.

`otel_api` RUNS ON UNMODIFIED POSTGRESQL 18 TODAY

Some demos use optional core or contrib patches that add more. Each demo slide says which.

NEEDS NO PATCHES	Unmodified PG18 plus the otel_api extensions.
PATCHED CORE	Needs the core patch that bridges tracepoints to spans.
PATCHED CONTRIB	Needs the instrumented postgres_fdw.

©EDB 2026 — ALL RIGHTS RESERVED.

`otel_api` NEEDS NO CORE CHANGES, BUT TWO WOULD HELP

Context propagation from any client, so tracing is a setting, not an app change.

Other protocols carry trace context in a header - easier for middleware/drivers.

Trace and span IDs in log lines, to correlate logs with traces.

©EDB 2026 — ALL RIGHTS RESERVED.

WHERE DID THAT SECOND GO? A ROW LOCK

PATCHED CORE

The UPDATE takes 1.1 s, and nearly all of it is pg.lock.wait.

✓ Long lock waits



HyperDX

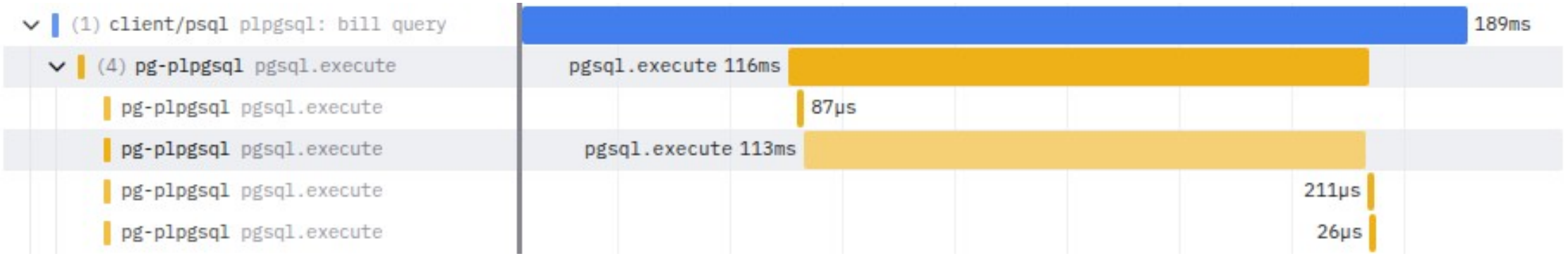
©EDB 2026 — ALL RIGHTS RESERVED.

PL/PGSQL: STATEMENT-LEVEL TIMING

NEEDS NO PATCHES

Every statement bill_customer runs gets its own span, with its text.

✓ PL/pgSQL




```
db.query.text          "SELECT coalesce(sum(total), 0)
                       FROM orders
                           WHERE customer_id = p_customer
                             AND placed_at >= p_period
                             AND placed_at < p_period +
                       interval '1 month'"
```

HyperDX

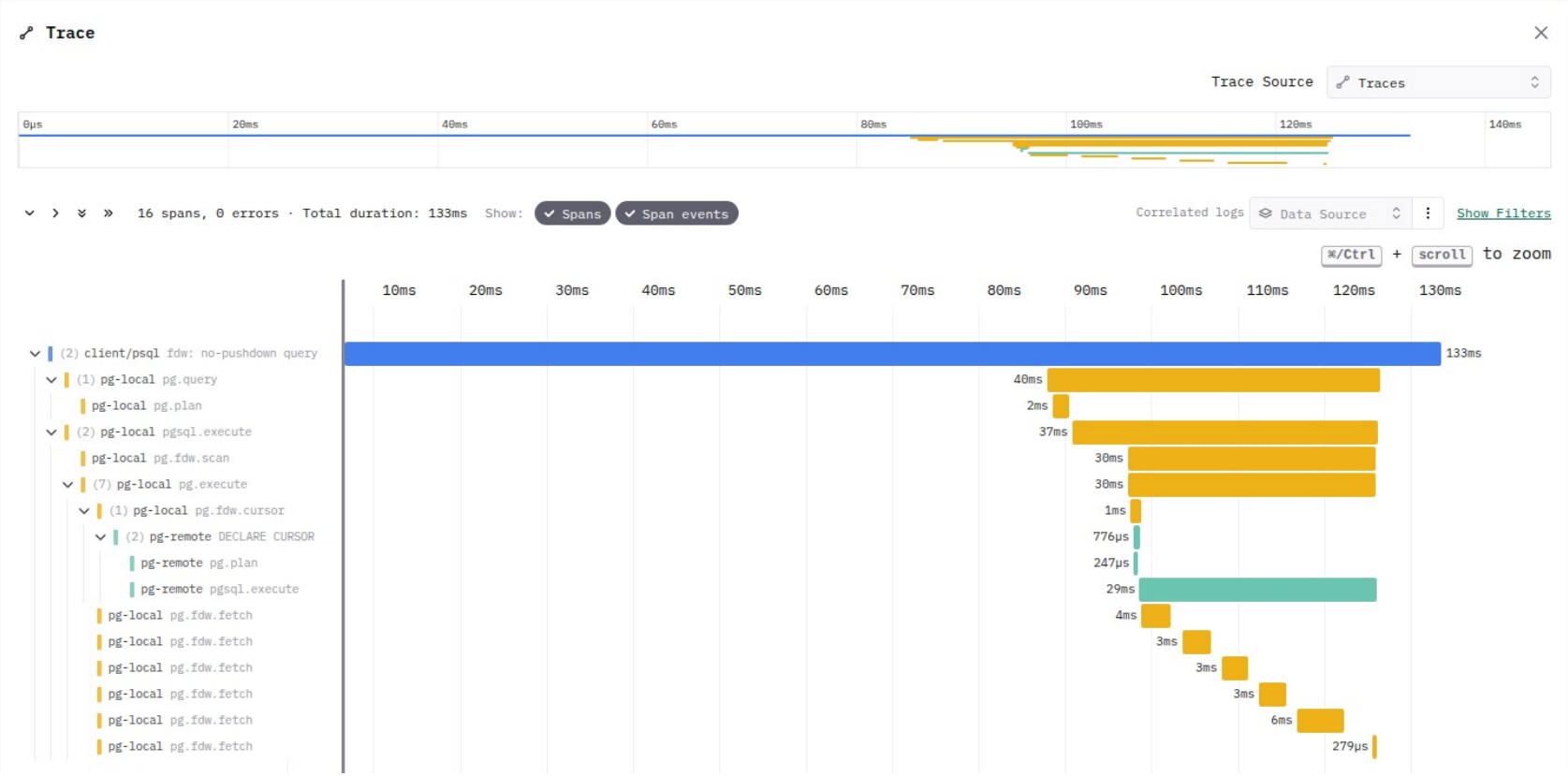
©EDB 2026 — ALL RIGHTS RESERVED.

`postgres_fdw`: SEE THROUGH TO THE FOREIGN SERVER

PATCHED CORE + PATCHED CONTRIB

A join that can't be pushed down: local scan and fetches, then the remote query.

✓ See through FDW



HyperDX

©EDB 2026 — ALL RIGHTS RESERVED.

OTEL_API IS AN INTERIM STEP, NOT THE GOAL

Ideally core would have a tracing API, like ereport(...).

This is a bridge, a way to break the chicken-and-egg problem.

It builds interest and a base of enthusiastic users.

They then instrument the extensions they use, and core.

©EDB 2026 — ALL RIGHTS RESERVED.

I'D LIKE PG_TRACING TO ADOPT THE API, NOT BE REPLACED

Not a replacement for pg_tracing.

I'd like `pg_tracing`, `pg_stat_ch` and others to adopt the API.

Interoperate with other extensions' spans.

Get driver and app context propagation.

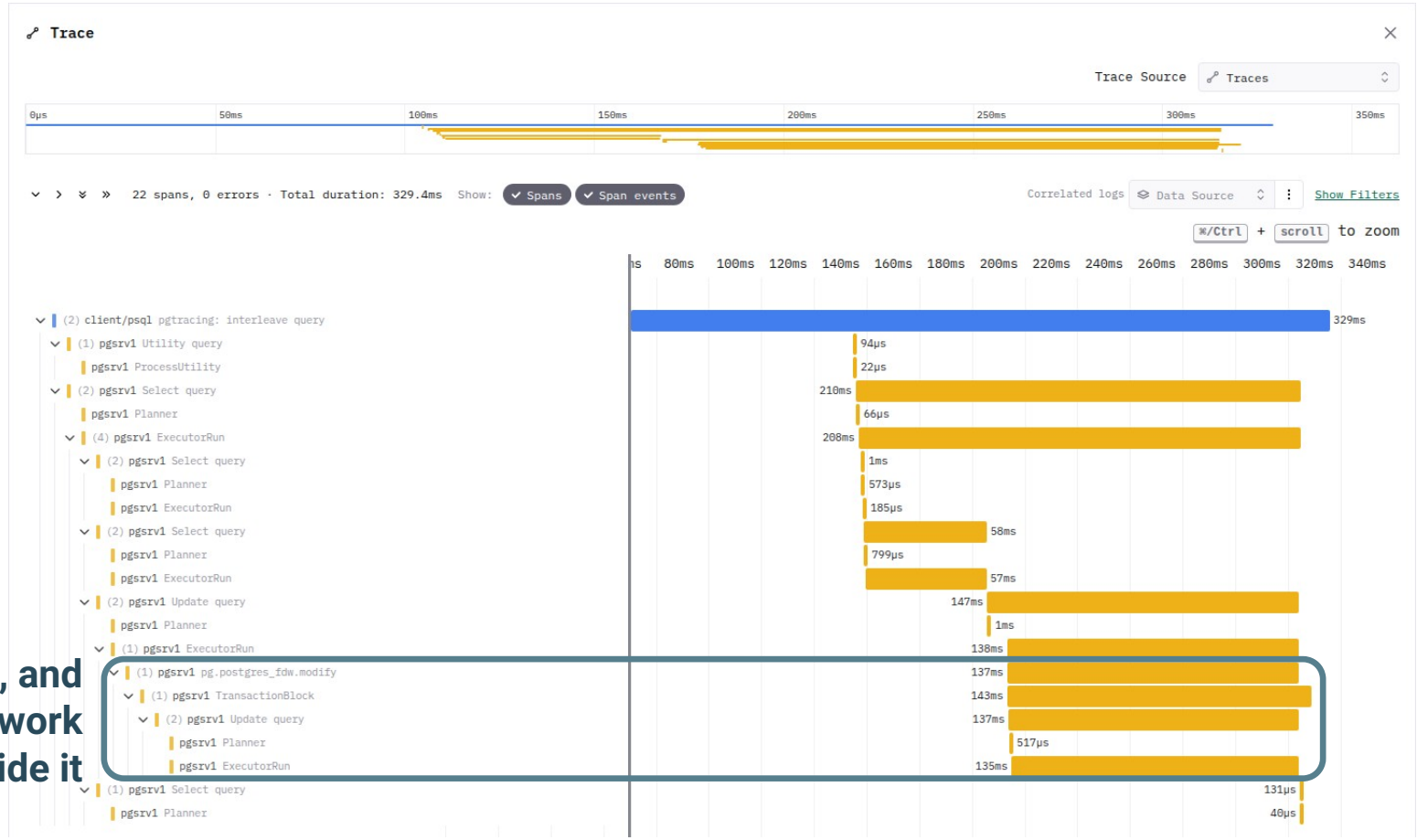
Use a pluggable exporter, and focus on tracing.

©EDB 2026 — ALL RIGHTS RESERVED.

pg_tracing WITH NESTED postgres_fdw SPANS

PATCHED PG_TRACING + PATCHED CONTRIB

postgres_fdw, ported to otel_api



postgres_fdw's span, and the remote backend's work inside it

HyperDX

©EDB 2026 — ALL RIGHTS RESERVED.

BUT POSTGRES ALREADY HAS TRACEPOINTS

So why do we need a new kind?

USDTs (Userspace statically-defined trace points)

```
$ grep -r 'TRACE_POSTGRESQL_' .
./src/backend/storage/buffer/bufmgr.c:  TRACE_POSTGRESQL_BUFFER_SYNC_DONE(NBuffers, num_written, num_to_scan);
./src/backend/storage/buffer/bufmgr.c:  TRACE_POSTGRESQL_BUFFER_FLUSH_START(BufTagGetForkNum(&buf->tag),
./src/backend/storage/buffer/bufmgr.c:  TRACE_POSTGRESQL_BUFFER_READ_DONE(tag.forkNum,
./src/backend/storage/smgr/md.c:                    TRACE_POSTGRESQL_SMGR_MD_READ_START(forknum, blocknum,
./src/backend/storage/smgr/md.c:                    TRACE_POSTGRESQL_SMGR_MD_WRITE_DONE(forknum, blocknum,
./src/backend/storage/lmgr/lwlock.c:            if (TRACE_POSTGRESQL_LWLOCK_WAIT_START_ENABLED())
./src/backend/storage/lmgr/lwlock.c:                TRACE_POSTGRESQL_LWLOCK_WAIT_START(T_NAME(lock), mode);
./src/backend/storage/lmgr/lwlock.c:                TRACE_POSTGRESQL_LWLOCK_WAIT_DONE(T_NAME(lock), mode);
```

Used by eBPF, DTrace, perf, SystemTap

©EDB 2026 — ALL RIGHTS RESERVED.

USDTS ARE CHEAP, BUT CAN'T DO DISTRIBUTED TRACING

Very cheap, fine on hot paths.
But:

- No parent/child relationships or trace context.
- No way to carry a sampling decision.
- Privileged kernel access on Linux.
- They need an external tracer.

©EDB 2026 — ALL RIGHTS RESERVED.

A CURATED SET OF TRACEPOINTS BECOMES SPANS

Experimental core patch repurposes some USDT probes for OTel tracing

The demo tracing extension turns start/done pairs into spans on the active trace.

Only in sessions that already have a trace context.

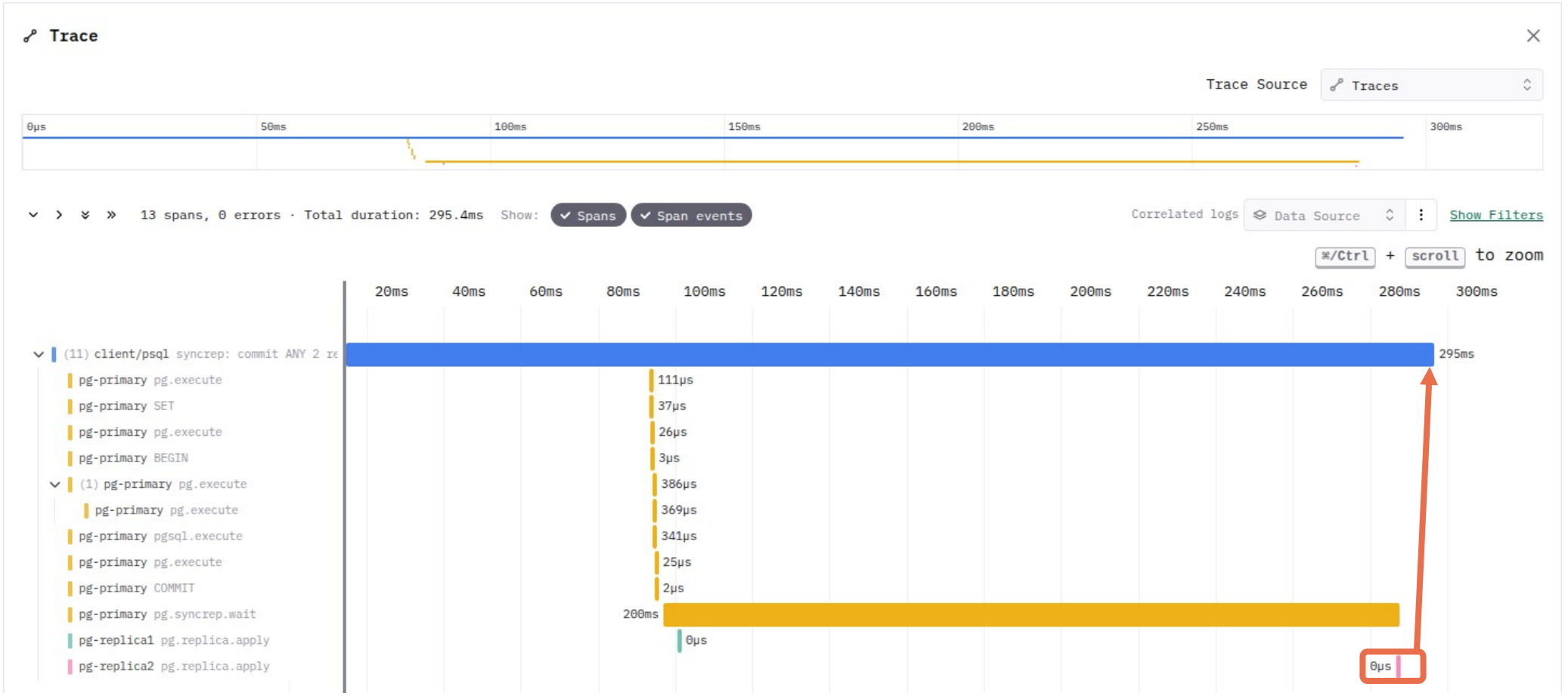
Unsuitable for upstream as-is.

©EDB 2026 — ALL RIGHTS RESERVED.

SYNCHRONOUS COMMIT, ANY 2: THE COMMIT WAITS 200 MS

PATCHED CORE

1 primary, 2 synchronous replicas · quorum commit: ANY 2 (replica1, replica2)

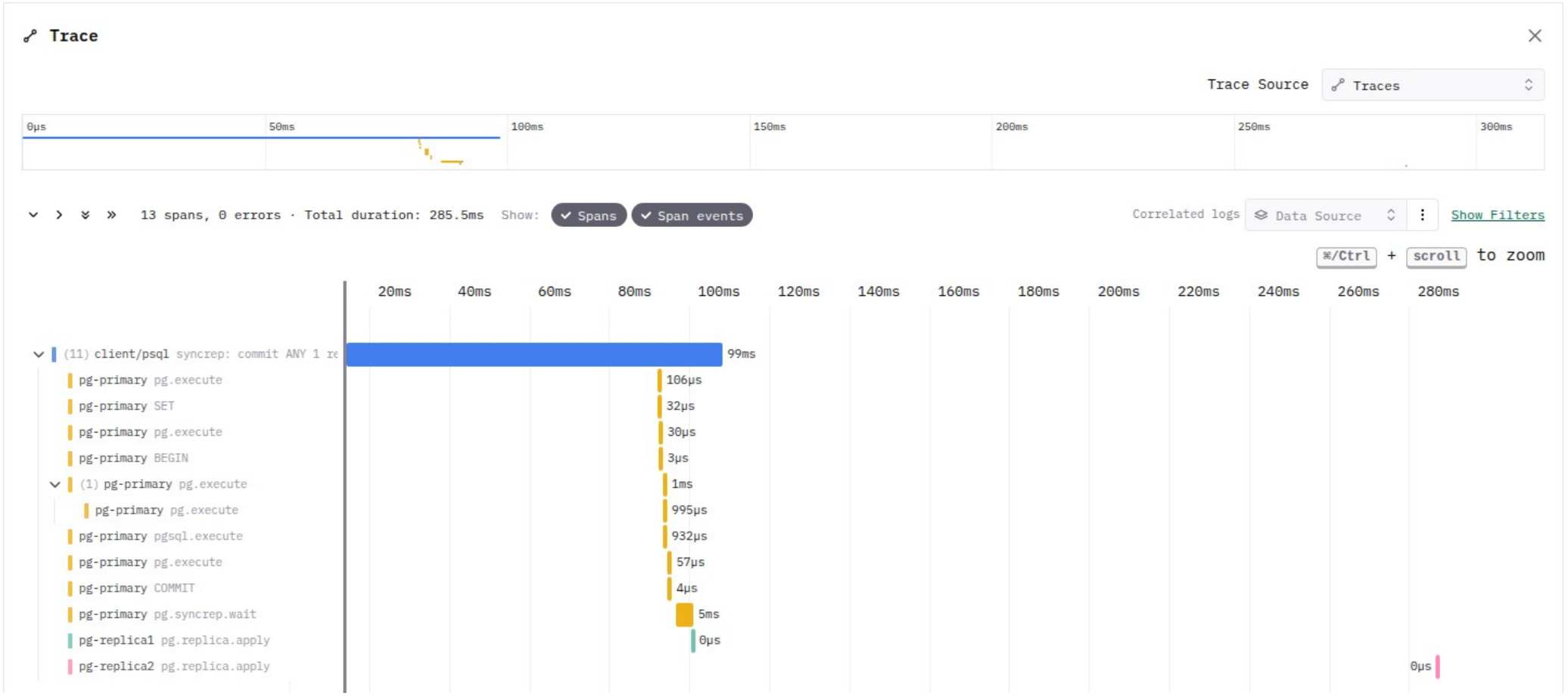


HyperDX

©EDB 2026 — ALL RIGHTS RESERVED.

SYNCHRONOUS COMMIT, ANY 1: THE SAME COMMIT RETURNS AFTER 5 MS

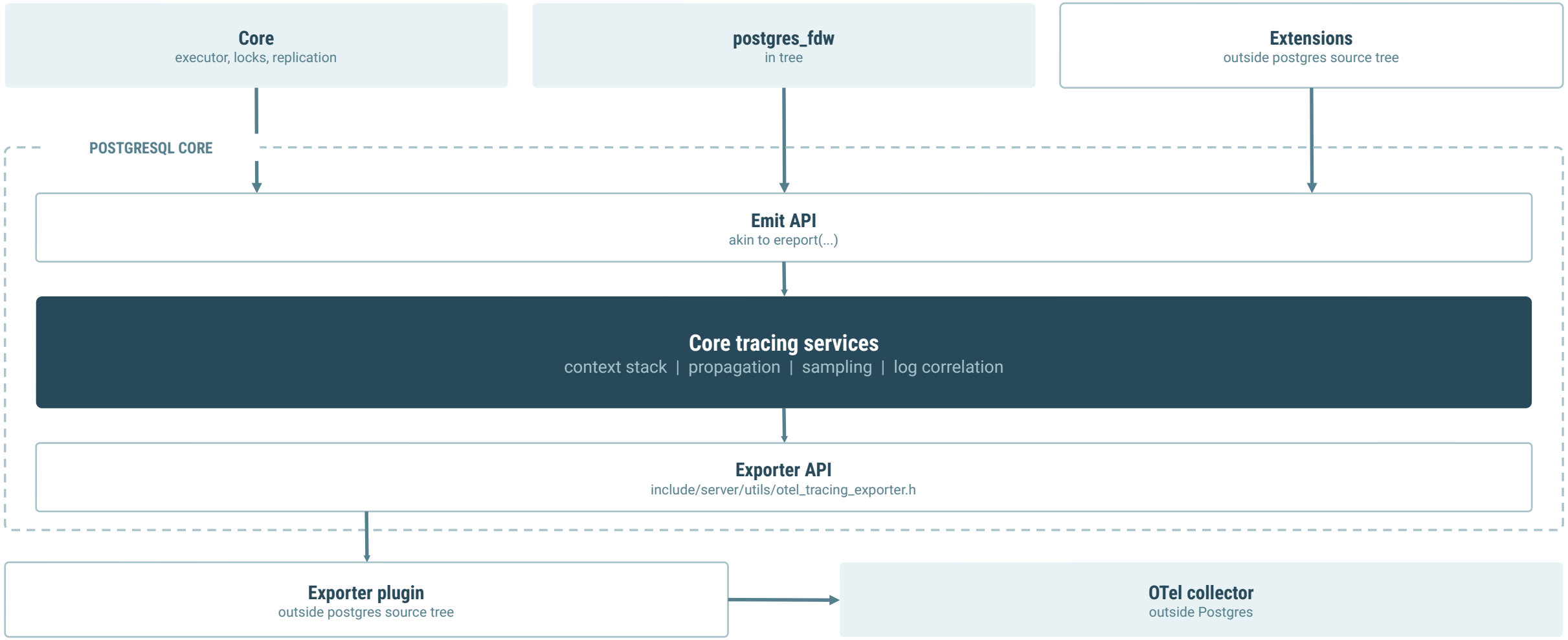
PATCHED CORE



HyperDX

©EDB 2026 — ALL RIGHTS RESERVED.

A PATH TO CORE: ONE POSSIBLE END STATE



©EDB 2026 — ALL RIGHTS RESERVED.

OPEN QUESTIONS REMAIN

Is the approach acceptable to extension authors?

Which hooks can core accept for observing core from tracing extensions?

How should tracing be exposed to Rust extensions?

How do we trace into threaded extensions? (Pg's APIs aren't thread-safe)

Is OpenTelemetry too heavy for a database?

©EDB 2026 — ALL RIGHTS RESERVED.

IS OTEL OVERHEAD ACCEPTABLE?

It has overhead. Sampling keeps it manageable.

Today's cost is mostly the exporter.
Unoptimised, and replaceable.

Collector cost is offloaded.
Out-of-process or off-node.

Negligible cost unless active.

©EDB 2026 — ALL RIGHTS RESERVED.

TRACING THAT IS OFF COSTS (ALMOST) NOTHING

CPU instructions per span site. A trivial SELECT 1 is about 45k. Nothing is optimised yet.

STATE	INSTRUCTIONS PER SPAN	% OF A SELECT 1
otel_api not loaded	~30	< 0.1%
Loaded, not tracing	~33	< 0.1%
Tracing on, not sampled	~400	~1%
Sampled, recorded (null exporter)	~1,100; ~2,700 with 4 attributes	~2.4%; ~6% with 4 attributes

©EDB 2026 — ALL RIGHTS RESERVED.

SAMPLE, AND THE COST SHRINKS

postgres_fdw and PL/pgSQL, with an **unoptimised demo OTLP exporter**.

SAMPLING	EFFECT
100% sampled	TPS −7% (fdw) to −34% (PL/pgSQL, 23 spans/txn)
1% sampled	≤ 1.5% instructions, TPS −1 to −3%
Off or not sampled	Within noise (≤ 1%)

©EDB 2026 — ALL RIGHTS RESERVED.

HELP MAKE TRACING UBIQUITOUS

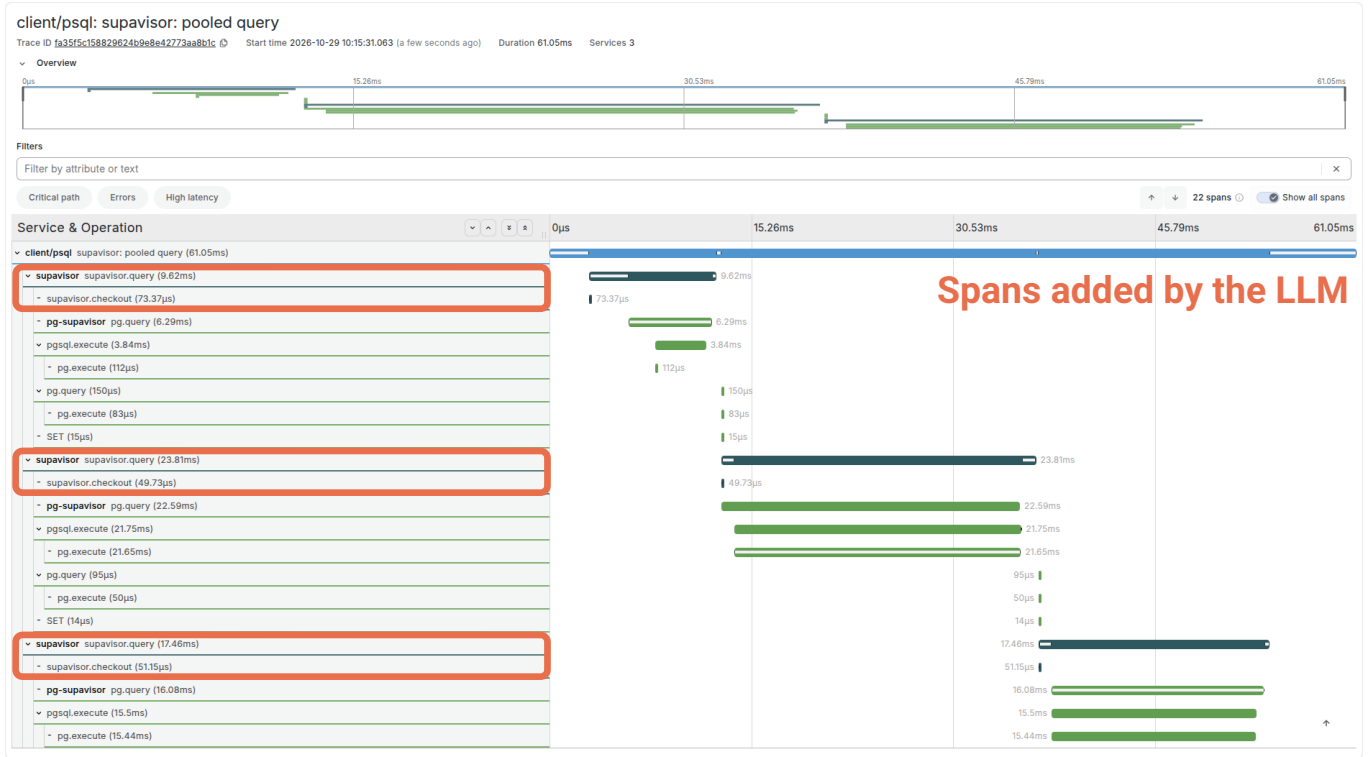
EXTENSION AUTHORS	Try `otel_api`, tell me what's missing.
CORE & CONTRIBUTORS	Help land the trace-context protocol message.
EVERYONE	Try it out.
NOT INSTRUMENTED?	LLMs can help.

Sending LLM patches upstream? Vet carefully, prominently disclose, and ask the maintainer's permission.

©EDB 2026 — ALL RIGHTS RESERVED.

AN LLM INSTRUMENTED THIS IN HALF AN HOUR

Supavisor instrumented by an LLM (demo grade, simple queries only).



Grafana

Quality instrumentation takes more time and effort.

github.com/ringerc/supavisor-patches/tree/otel-tracing-demo

©EDB 2026 — ALL RIGHTS RESERVED.

RESOURCES AND CREDITS

IMAGE CREDITS

Distributed systems	Tatsu Takeuchi, Virginia Tech, © 2005 Tatsu Takeuchi.
Question volume	Stack Overflow new questions per month. Data: Stack Exchange API, CC BY-SA 4.0; chart by Craig Ringer.
MySQL in OCI APM	Oracle MySQL blog, Mike Frank and Miroslav Rajčić, © 2023.
Kafka and MongoDB	dev.to/aspecto, Motti Bechhofer at Aspecto, © 2023.
Istio with Zipkin	istio.io, Istio Authors, © 2026.
pg_tracing	pgsql-hackers post, Anthonin Bonnefoy at Datadog, © 2023.

TOOLS IN THE SCREENSHOTS INCLUDE:

HyperDX

Grafana

ClickHouse

OpenTelemetry Collector

Supavisor

©EDB 2026 — ALL RIGHTS RESERVED.

QUESTIONS

Resources:

OTEL_API AND ITS EXTENSIONS

github.com/ringerc/postgres_otel_api

PATCHED PG18 CORE USED BY THE DEMOS

github.com/ringerc/postgres/tree/postgres-otel-tracing-pg18

TRACEPARENT PROTOCOL MESSAGE

github.com/ringerc/postgres/tree/protocol-traceparent

GENERALISED HEADERS PROTOCOL MESSAGE

github.com/ringerc/postgres/tree/protocol-headers-generic

The demo ports, experiments and test cases were built with heavy LLM assistance.

Craig Ringer

craig.ringer@enterprisedb.com

ringerc@ringerc.id.au

This deck:

EDB POSTGRES AI

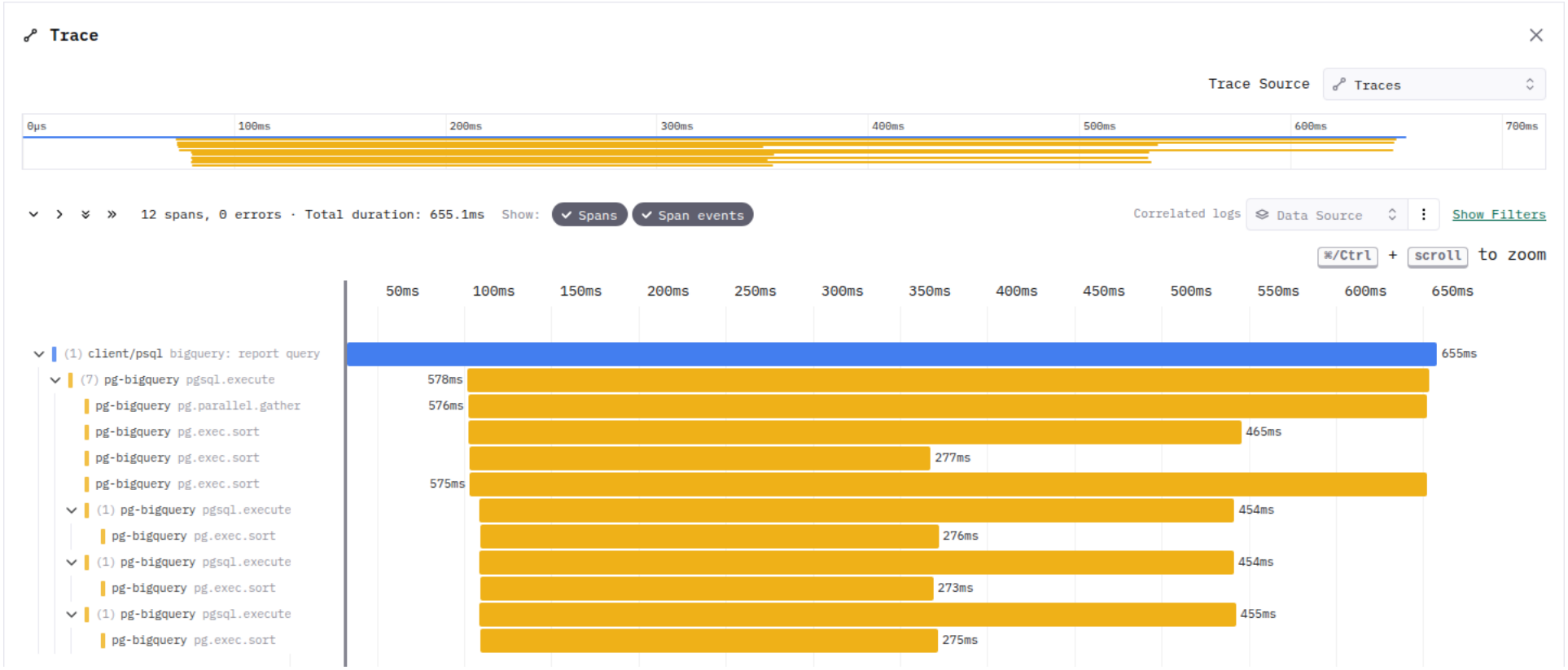
©EDB 2026 — ALL RIGHTS RESERVED.

©EDB 2026 — ALL RIGHTS RESERVED.

BACKUP: PARALLEL WORKERS GET THEIR OWN SPANS

NEEDS NO PATCHES

Each worker is its own span in the same trace, with its own sort.



HyperDX

©EDB 2026 — ALL RIGHTS RESERVED.

BACKUP: PG_TRACING NATIVE VS OTEL_API

ROUGH, ONE CLIENT

WORKLOAD	VARIANT	VS NO PG_TRACING	SPANS/TXN DELIVERED
select-only	native	−23%	almost none
select-only	otel_api, no exporter	−18%	-
select-only	otel_api, OTLP exporter	−62%	3 (all)
tpcb-like	native	−22%	almost none
tpcb-like	otel_api, no exporter	−16%	-
tpcb-like	otel_api, OTLP exporter	−65%	about 25 (all)

Rough, one client. Throughput change against no pg_tracing, 2026-10-27.

©EDB 2026 — ALL RIGHTS RESERVED.

BACKUP: JSONLOG CARRIES THE TRACE ID TOO

PATCHED CORE

The same error as a jsonlog record, with trace_id and span_id.

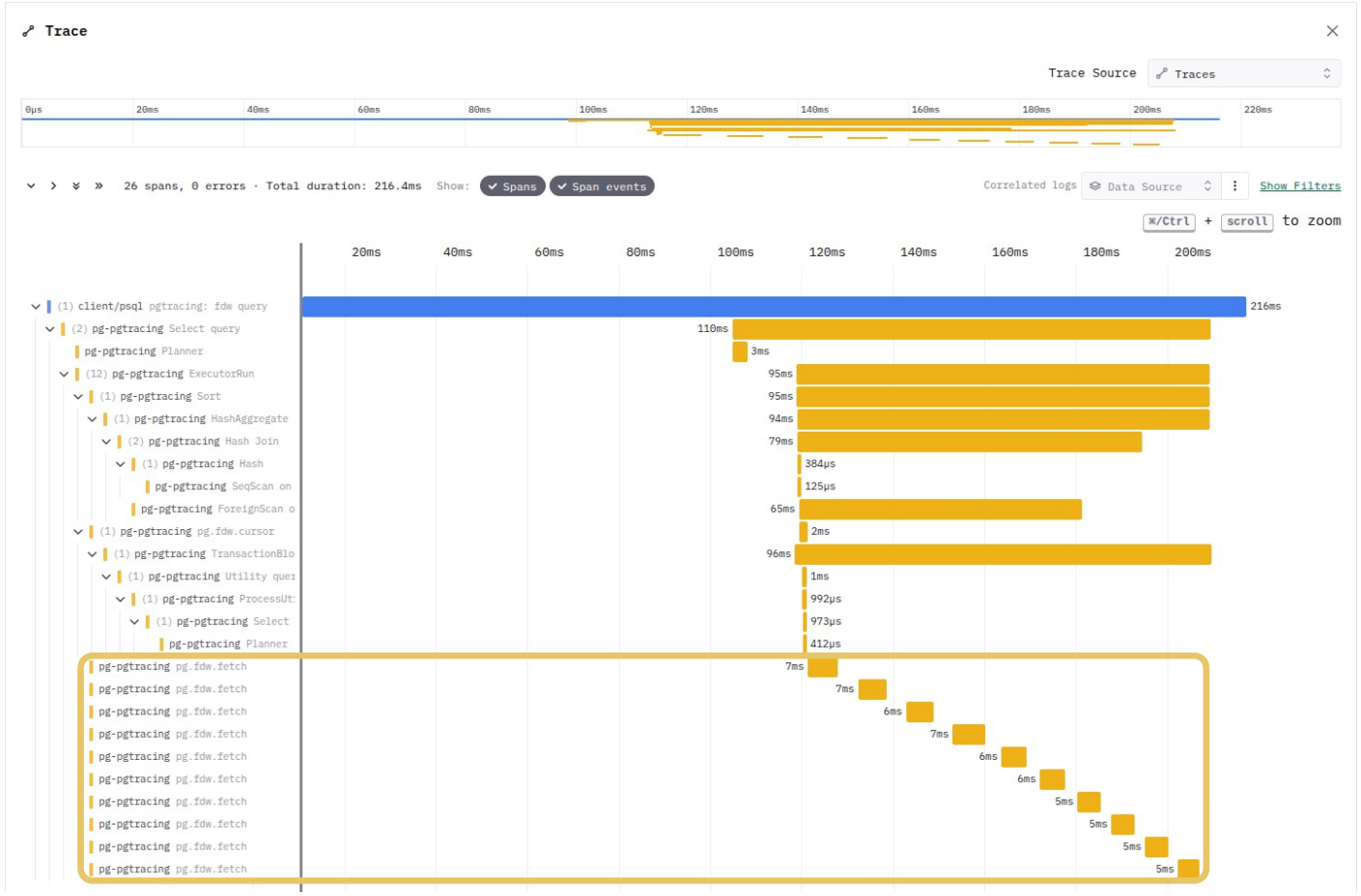

```
{"timestamp":"2026-10-25 02:59:40.425 UTC","user":"postgres","dbname":"postgres",
 "pid":26563,"error_severity":"ERROR","state_code":"23505",
 "message":"duplicate key value violates unique constraint \"orders_pkey\"",
 "detail":"Key (id)=(42) already exists.",
 "trace_id":"756841cca92ab4cd826206caaffaf24d","span_id":"83c89bd22c936622"}
```


Simulated: the demo image lacks the core log-annotation patch.

©EDB 2026 — ALL RIGHTS RESERVED.

BACKUP: PG_TRACING AND POSTGRES_FDW IN ONE TRACE

PATCHED CONTRIB



HyperDX

The pg.fdw.fetch spans sit beside Sort, not under the ForeignScan that caused them.

pg_tracing builds its node spans after execution. postgres_fdw's spans are live.

©EDB 2026 — ALL RIGHTS RESERVED.

BACKUP: THE SAMPLING DECISION RIDES IN THE TRACEPARENT

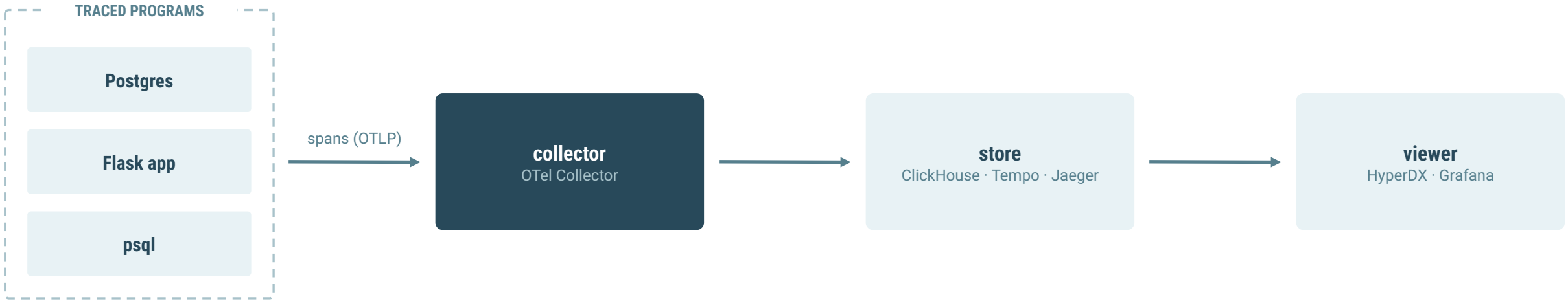
```
00-<trace-id>-<span-id>-<flags>
```



01: the sampled bit is set, so every service downstream records.

©EDB 2026 — ALL RIGHTS RESERVED.

BACKUP: A TRACE REACHES YOUR SCREEN IN FOUR STEPS



©EDB 2026 — ALL RIGHTS RESERVED.

BACKUP: SPAN EVENTS ARE LOG LINES ATTACHED TO A SPAN

NEEDS NO PATCHES


```
{
  "name": "exception",
  "time": "2026-10-25 02:59:40.425106+00",
  "attributes": {
    "exception.type":    "23505",
    "exception.message": "duplicate key value violates unique constraint \"orders_pkey\"",
    "pg.error.detail":   "Key (id)=(42) already exists.",
    "code.function.name": "_bt_check_unique",
    "code.file.path":    "nbtinsert.c",
    "code.line.number":  666
  }
}
```


©EDB 2026 — ALL RIGHTS RESERVED.

BACKUP: THE PLANNER WAS OFF BY 6000×, AND THE SPAN SAYS SO

NEEDS NO PATCHES



Grafana

©EDB 2026 — ALL RIGHTS RESERVED.