Building a Truly Compatible Postgres Proxy

What "speaks Postgres" actually costs

September 2026

Haritabh Gupta

Engineer · Multigres

github.com/haritabh17

haritabh.gupta@supabase.io

supabase

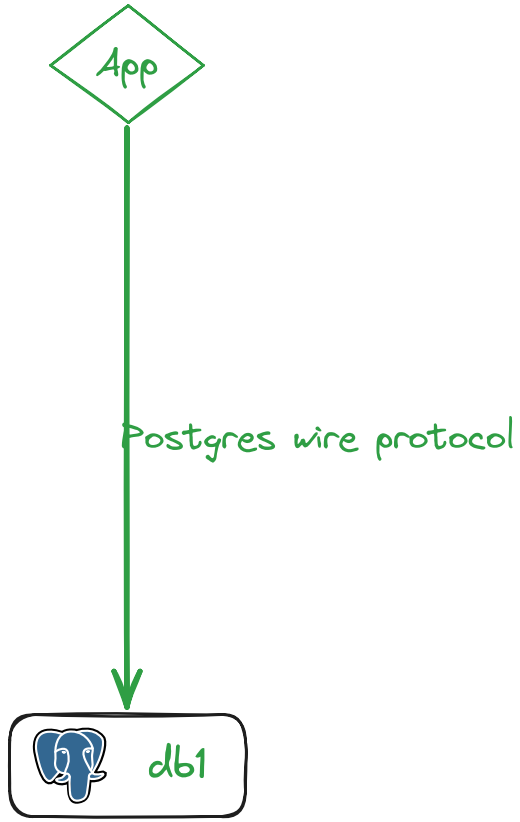
What is Multigres?

An ecosystem for PostgreSQL.

- Open Source
- Sharding
- High Availability
- Backups & Restore
- Connection Pooling
- Query Analytics

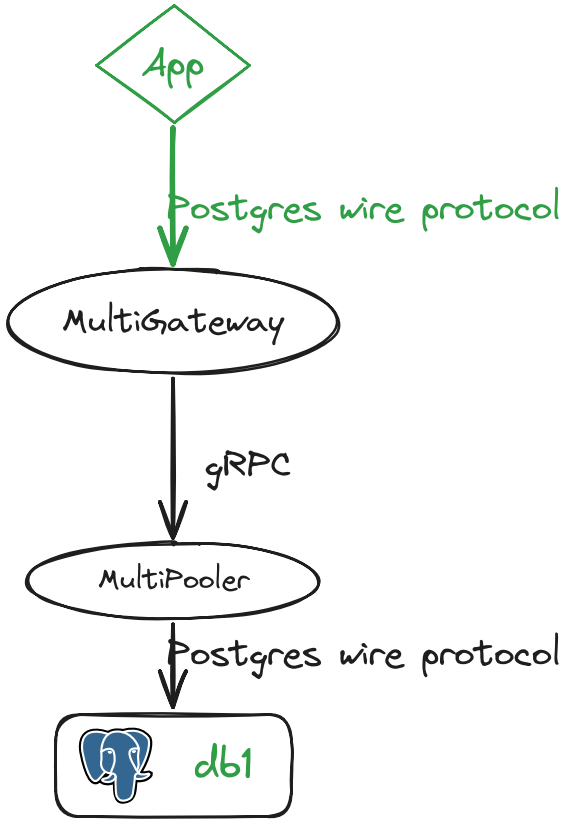
The shape of Multigres

Two services. **Multigateway** speaks the Postgres wire protocol to the client. **Multipooler** sits in front of each backend, owns transactions, talks to Postgres. Between them: gRPC.



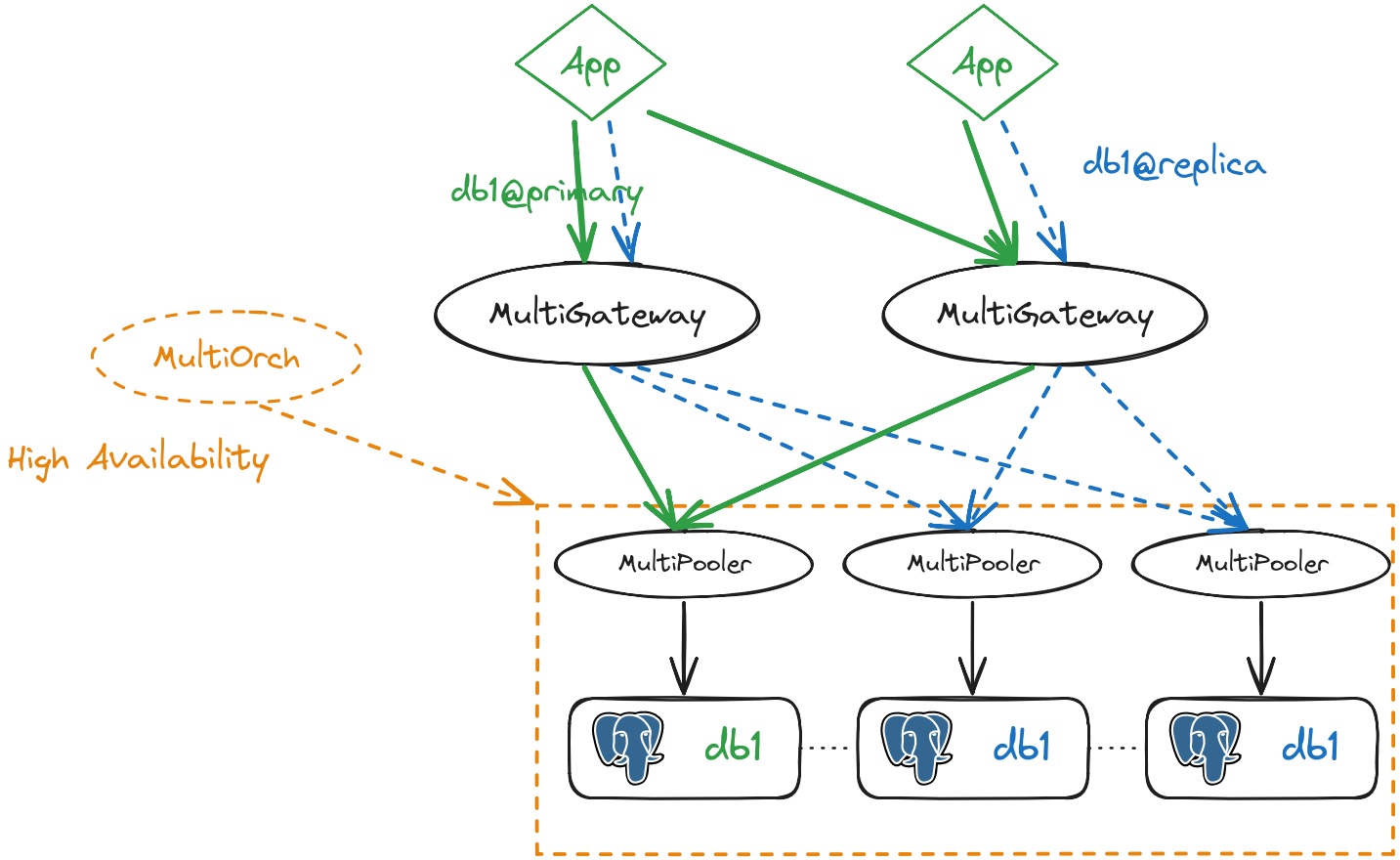
The shape of Multigres

Two services. **Multigateway** speaks the Postgres wire protocol to the client. **Multipooler** sits in front of each backend, owns transactions, talks to Postgres. Between them: gRPC.



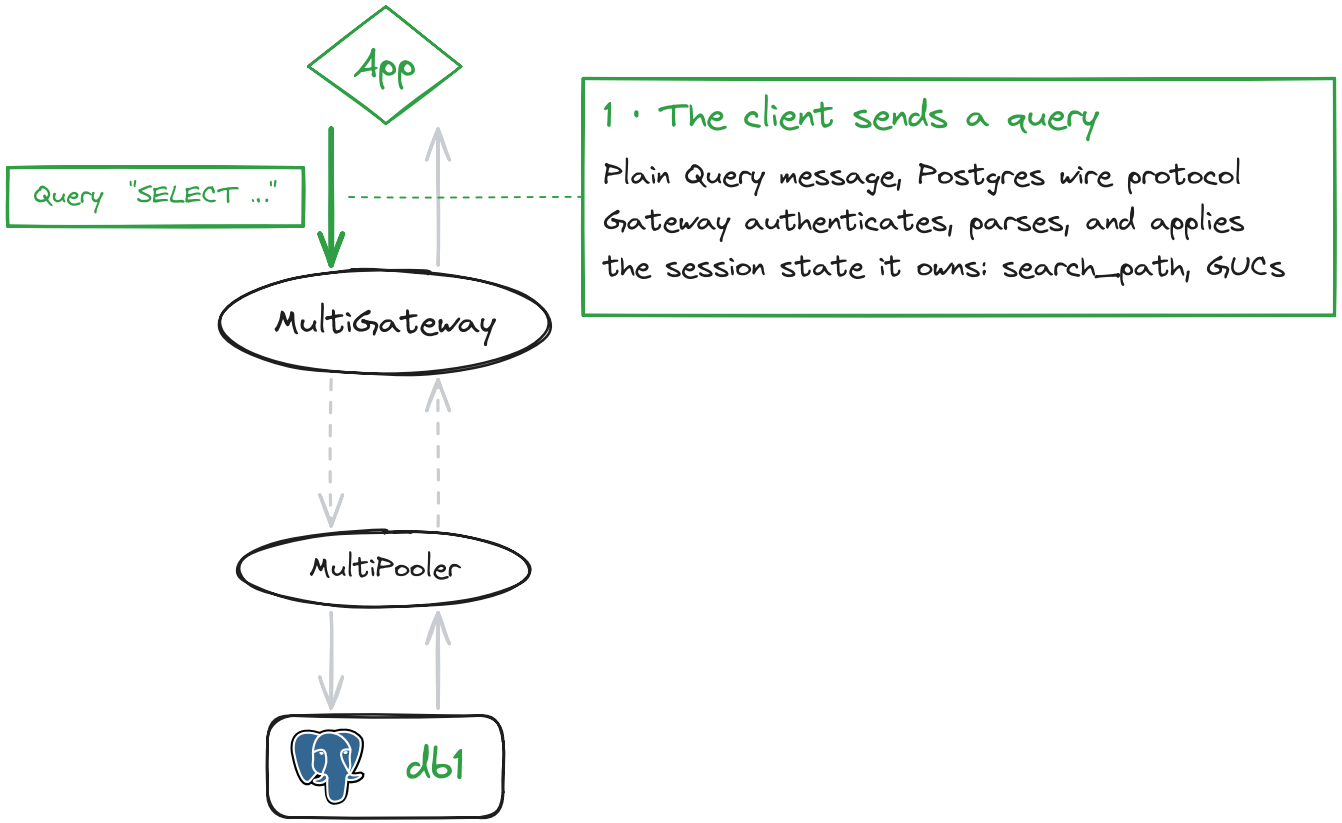
The shape of Multigres

Two services. **Multigateway** speaks the Postgres wire protocol to the client. **Multipooler** sits in front of each backend, owns transactions, talks to Postgres. Between them: gRPC.



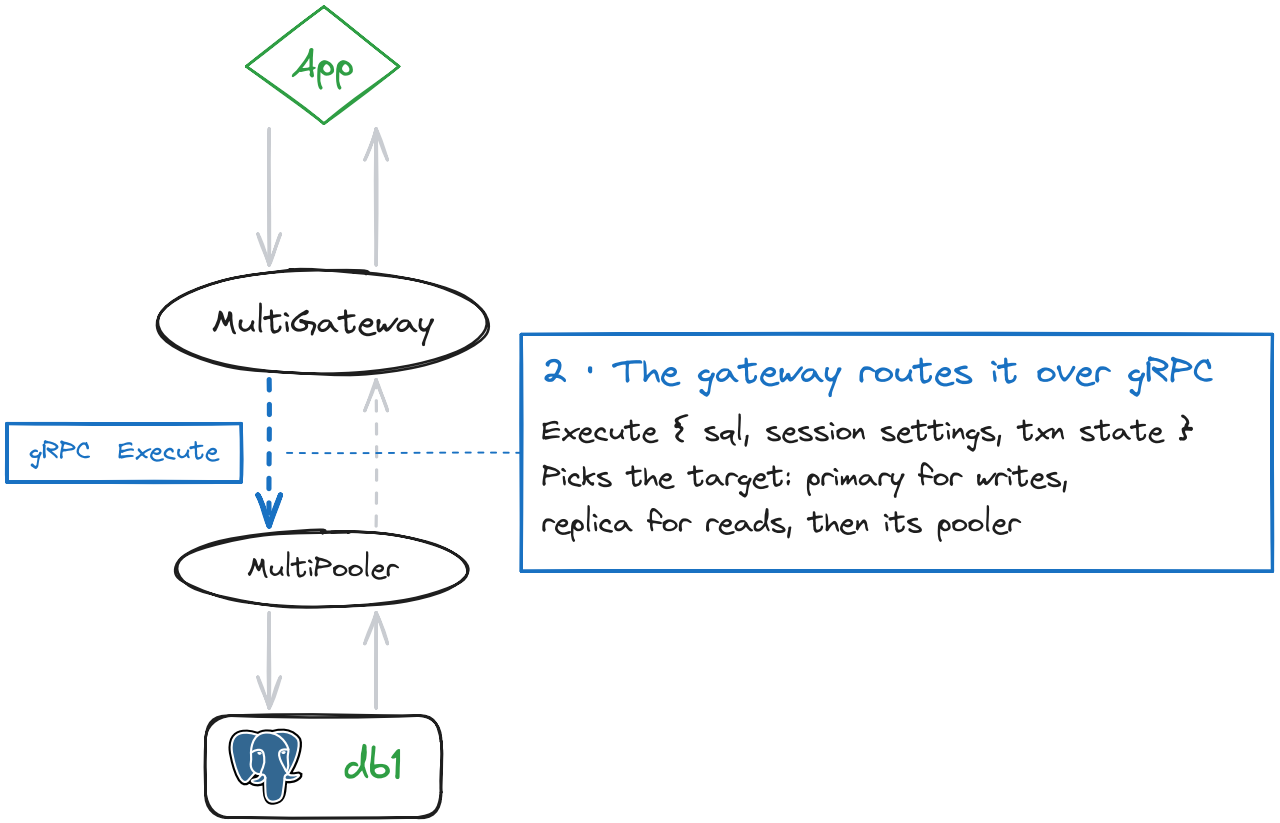
Life of a query

One SELECT, six hops. Postgres wire protocol in green, gRPC in **blue** — every message has to survive the translation and come back byte-for-byte.



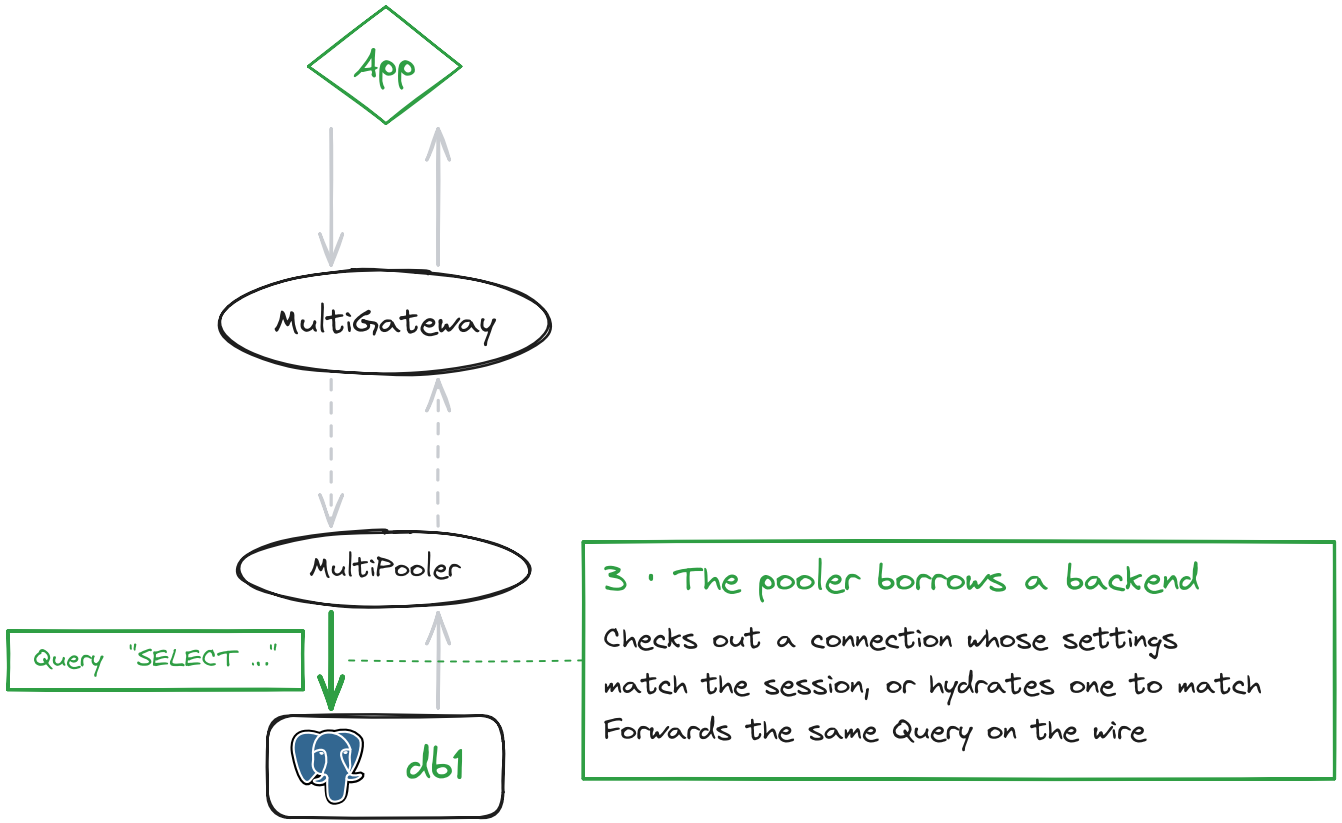
Life of a query

One SELECT, six hops. Postgres wire protocol in green, gRPC in **blue** — every message has to survive the translation and come back byte-for-byte.



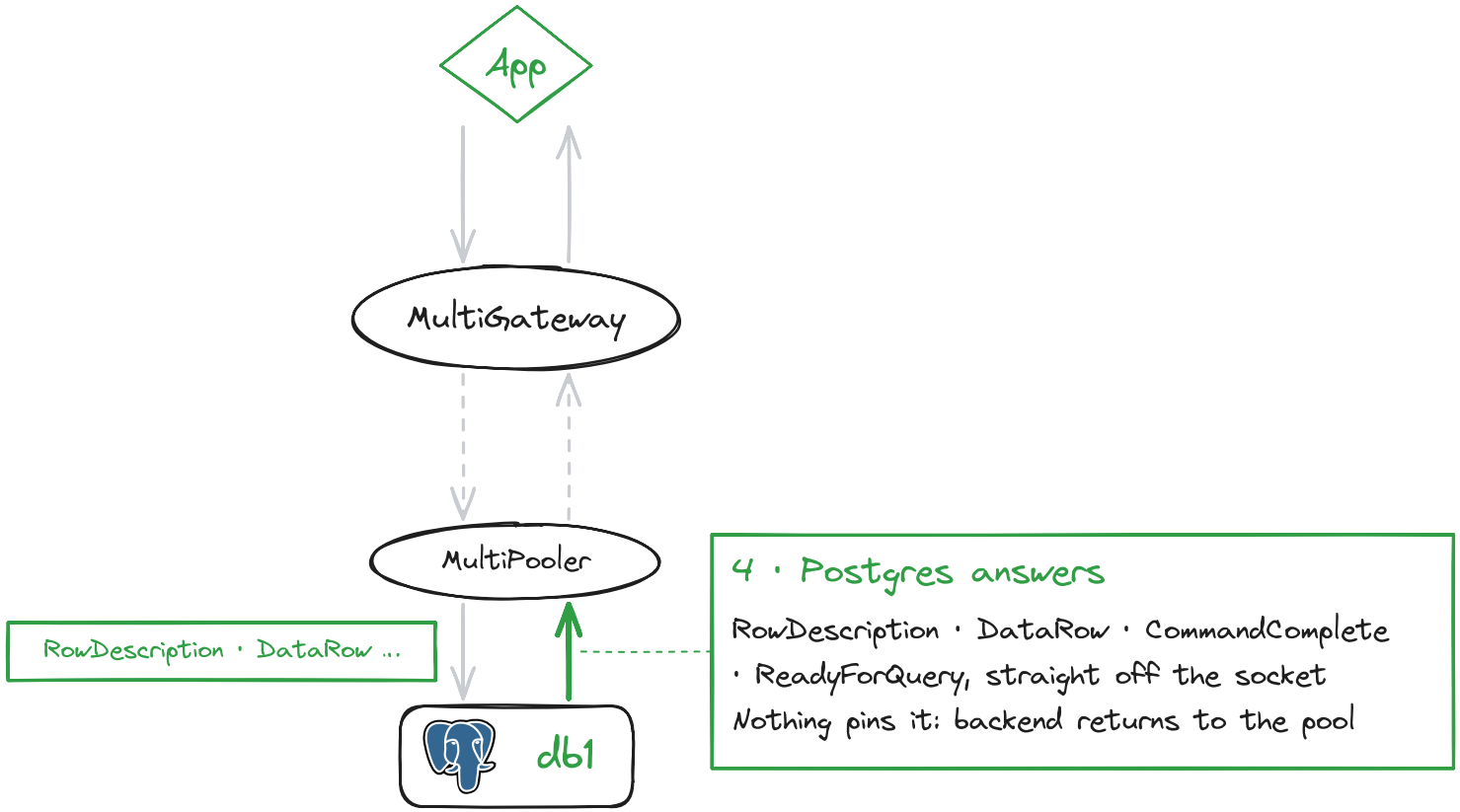
Life of a query

One SELECT, six hops. Postgres wire protocol in green, gRPC in **blue** — every message has to survive the translation and come back byte-for-byte.



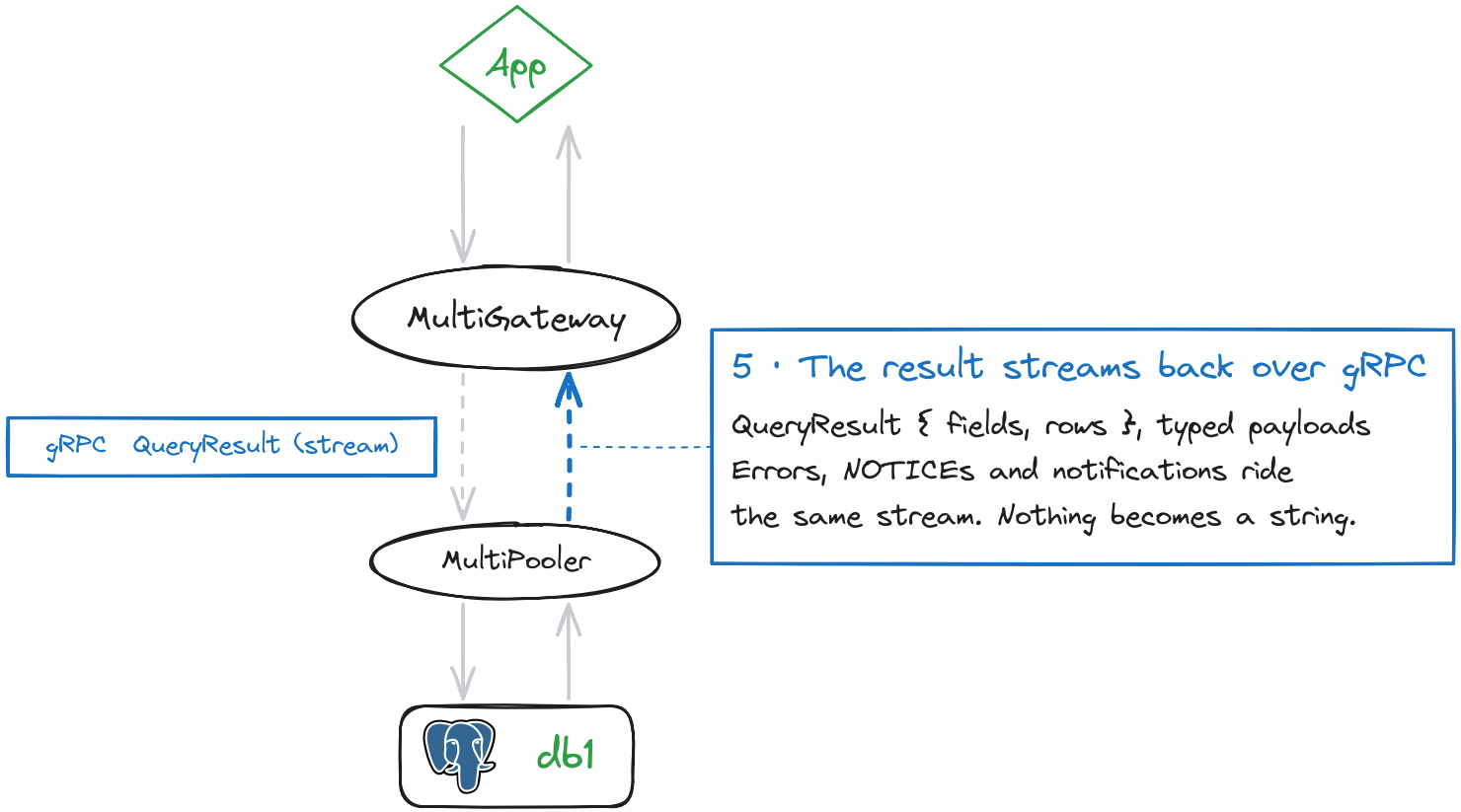
Life of a query

One SELECT, six hops. Postgres wire protocol in green, gRPC in **blue** — every message has to survive the translation and come back byte-for-byte.



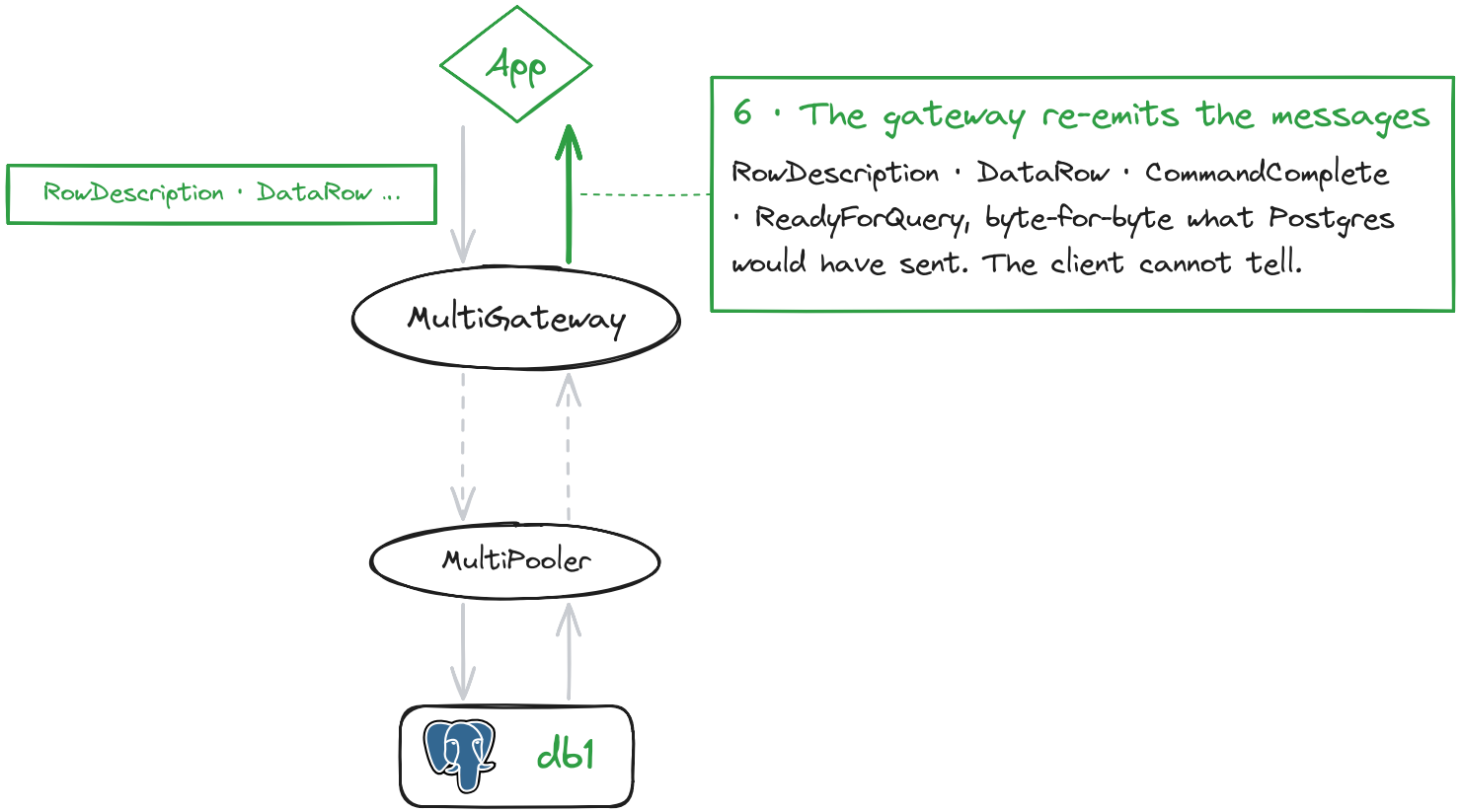
Life of a query

One SELECT, six hops. Postgres wire protocol in green, gRPC in **blue** — every message has to survive the translation and come back byte-for-byte.



Life of a query

One SELECT, six hops. Postgres wire protocol in green, gRPC in **blue** — every message has to survive the translation and come back byte-for-byte.



Every proxy says it speaks Postgres. Why isn't that enough?

Parsing StartupMessage and Query ≠ behaving like Postgres.

What "transparent" means

An application connected to Multigres cannot observe a difference from a direct Postgres connection — along any axis it can probe.

- Every wire message Postgres would send — and only those
- Every diagnostic field on errors and notices
- Every state transition, every async message, at the right moment
- Every startup parameter, every TLS handshake quirk

THE LONG TAIL

The compatibility surface

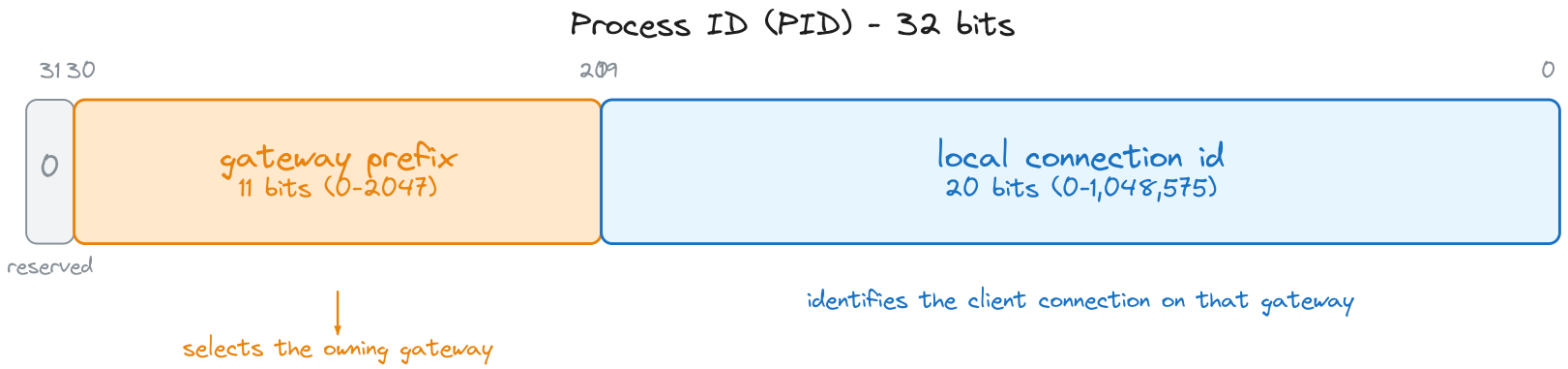
- Query cancellation
- COPY sub-protocol
- Error diagnostics
- Async NOTICE
- . . .

Each sounded niche. Together they decide whether applications can trust the proxy.

Cancel arrives on a different connection

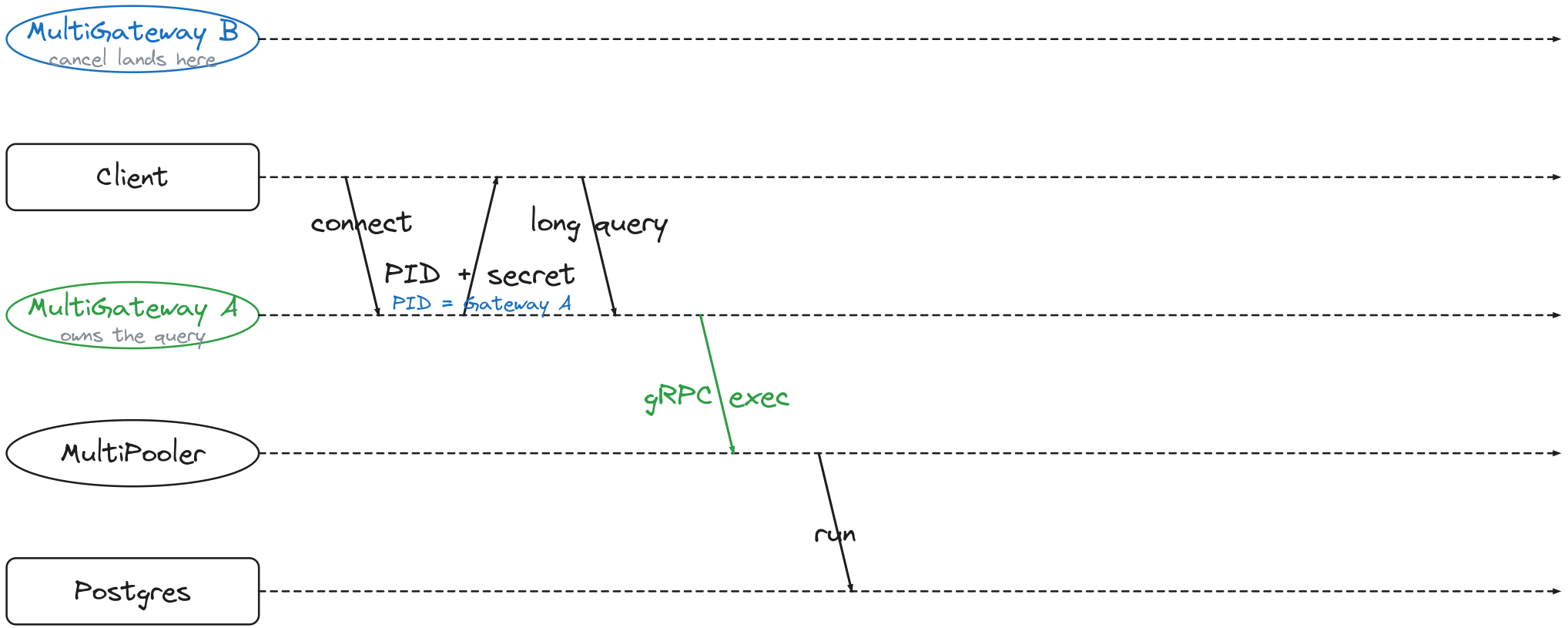
- Postgres hands the client a `PID` + secret key at startup (`BackendKeyData`).
- To cancel, the client opens a **new TCP connection** and sends a `CancelRequest` carrying that PID + secret.
- The server matches them to the running backend and cancels it — best-effort; the connection closes immediately.

Multigres keeps the exact same protocol, but encodes the owning gateway into the PID it issues:



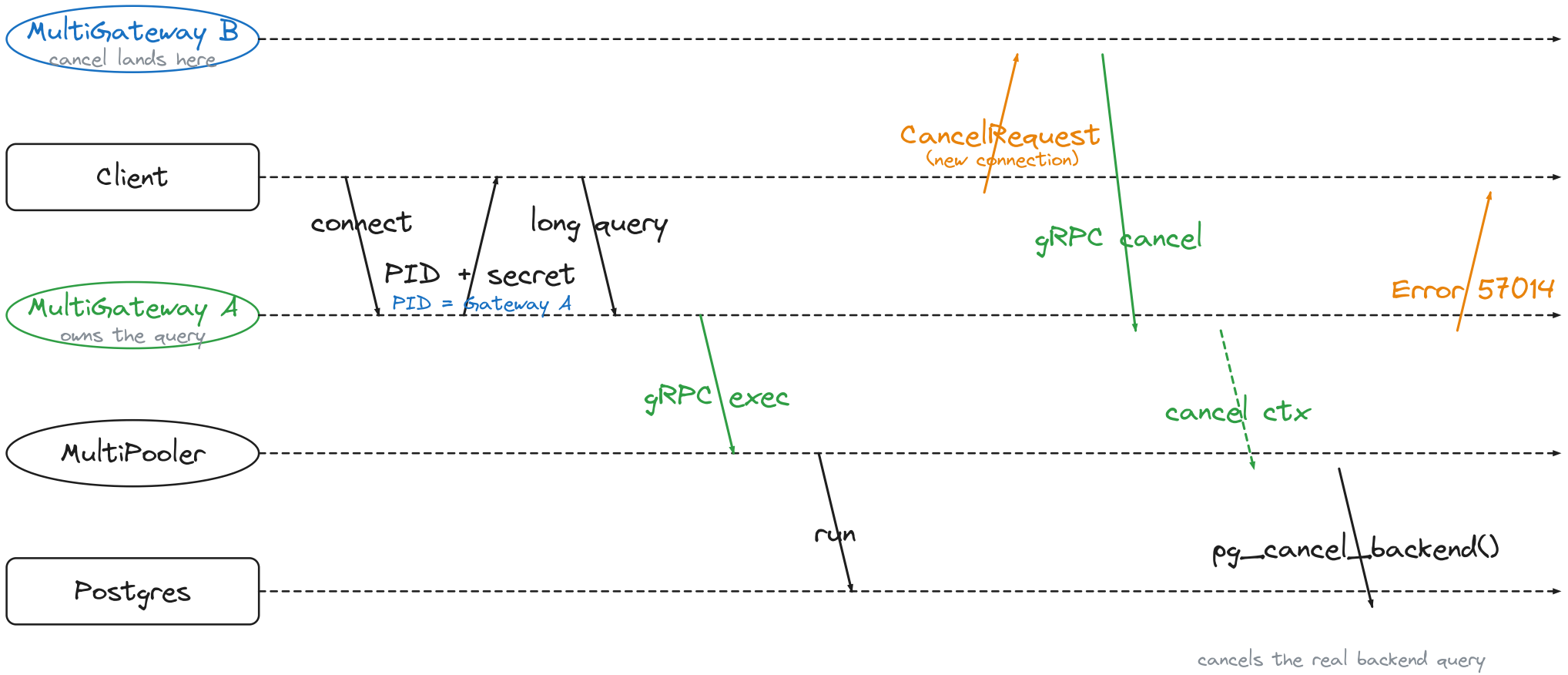
Routing a cancel to the right gateway

The cancel can land on **any** gateway. The PID's prefix routes it to the owner over gRPC; the owner verifies the secret and cancels the in-flight query — the cancelled context propagates down to the pooler.



Routing a cancel to the right gateway

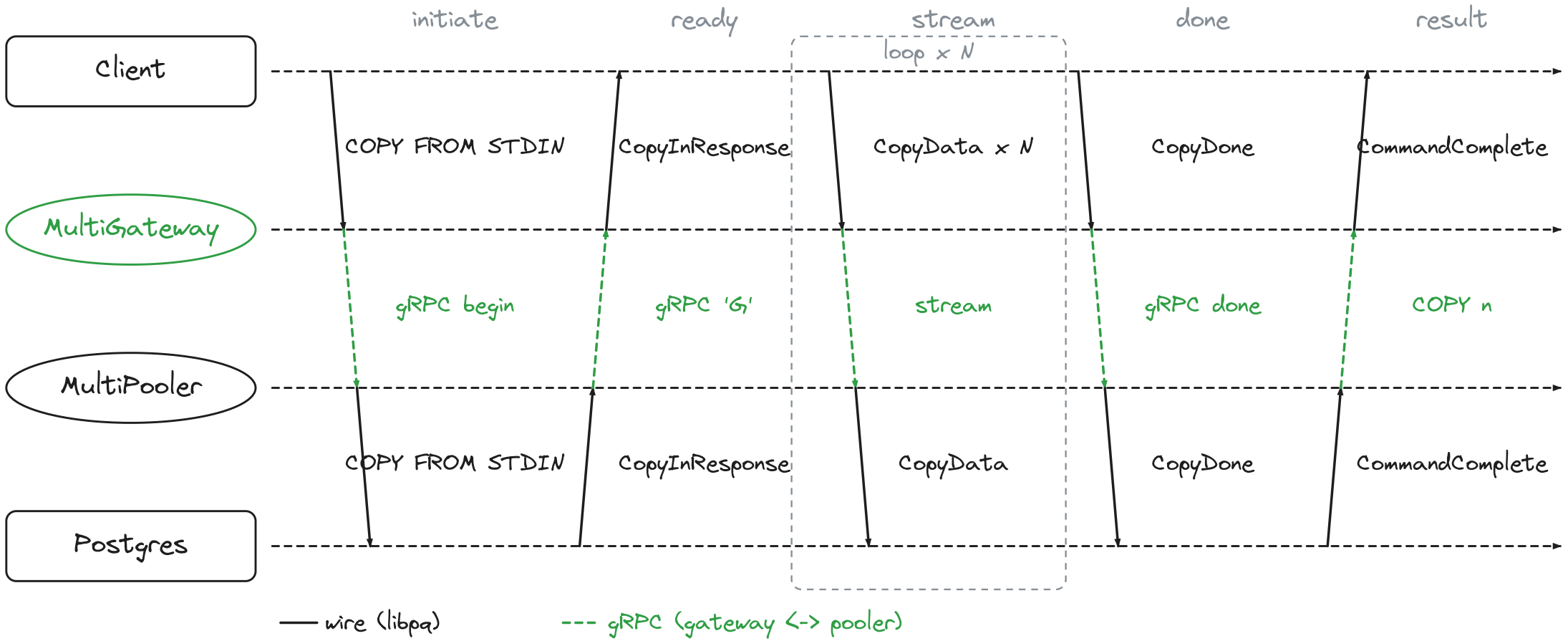
The cancel can land on **any** gateway. The PID's prefix routes it to the owner over gRPC; the owner verifies the secret and cancels the in-flight query — the cancelled context propagates down to the pooler.



COPY FROM STDIN is its own sub-protocol

- `COPY t FROM STDIN` flips the connection into a sub-protocol — no longer request/response
- Server sends `CopyInResponse`; the client then streams unbounded `CopyData` frames
- Client ends with `CopyDone` (or `CopyFail`) → `CommandComplete` + `ReadyForQuery`
- A 10 GB import can't be buffered; re-fragmenting the stream loses ordering

Streaming COPY through the proxy



Each `CopyData` chunk is forwarded over the gRPC stream and written straight to Postgres — **never staged in memory**. COPY TO STDOUT (COPY OUT) is the same machinery, data arrows reversed.

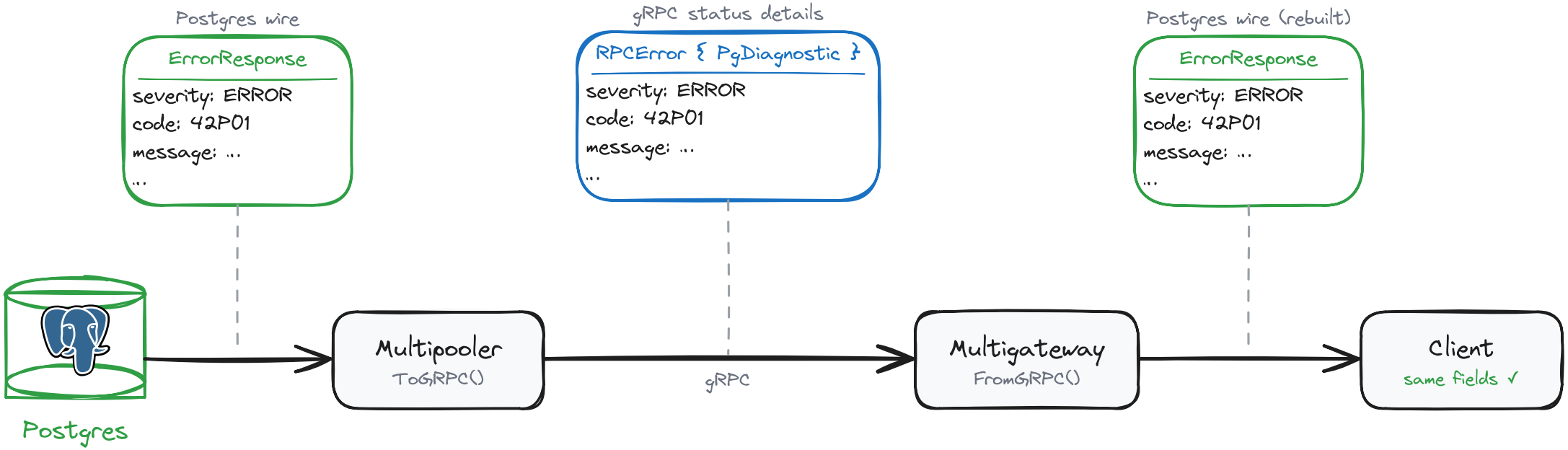
Error diagnostics: fourteen fields, not one

A Postgres `ErrorResponse` is one wire message built from typed fields.

- **14 typed fields** — severity, SQLSTATE, message, detail, hint, position, schema, table, column, constraint, and more.
- **250+ SQLSTATE codes** — ORMs branch on them; observability groups by them.
- gRPC has **one code and one message**. A naive bridge flattens everything else to a string.

Lossless errors over gRPC

Every field rides inside the gRPC status — `RPCError { PgDiagnostic }` attached with `status.WithDetails` — then re-emitted byte-faithfully to the client.



Naive bridge: gRPC code + message only → 13 fields lost (ORMs string-parse, observability mis-groups)

Lossless both directions — a round-trip test guards every field.

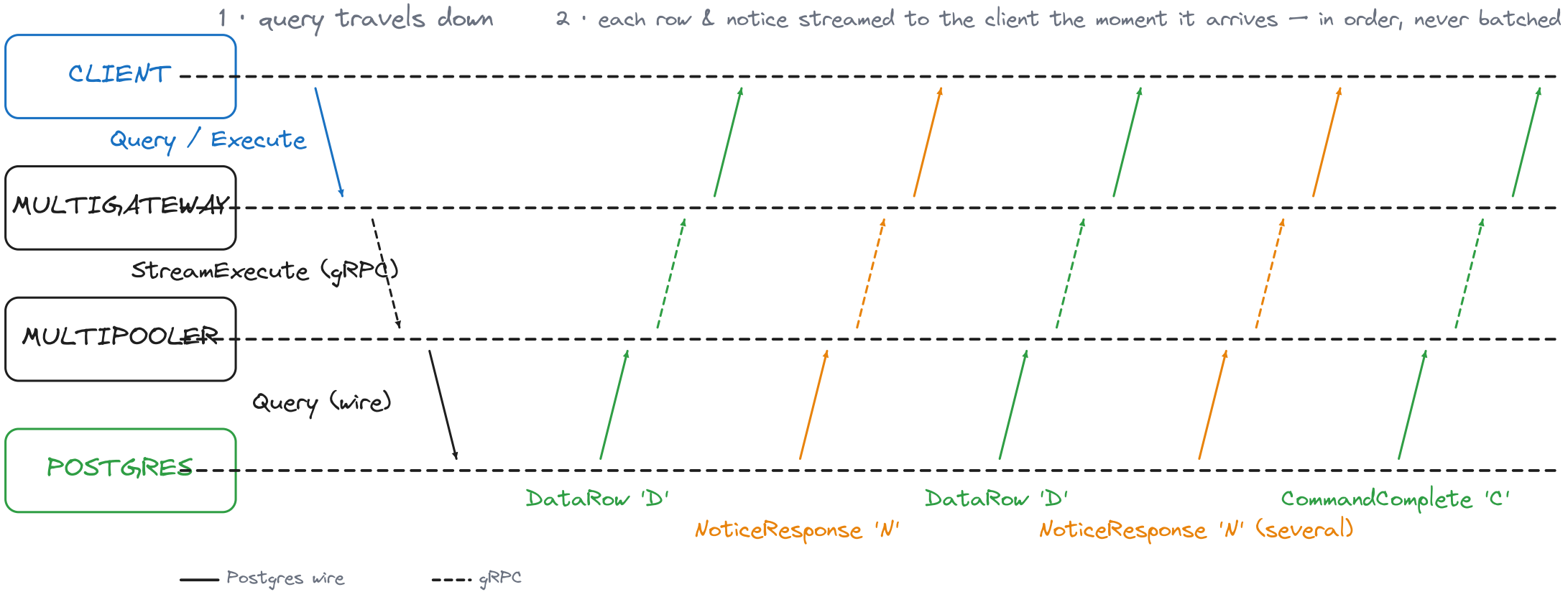
NOTICE can arrive at any time

Postgres pushes `NoticeResponse` on its own — the client never asked for it.

- **Unsolicited, any time** — between `DataRow`s, before `CommandComplete`, several in a row, even while idle.
- **Same wire shape as errors** — `'N'` vs `'E'`, the same 14 diagnostic fields — but **not** an error: no transaction abort, no `ReadyForQuery`.
- **Apps depend on them** — `RAISE NOTICE`, deprecation hints, monitoring; `client_min_messages` filters which arrive.
- **Position is information** — must stay inline, in arrival order. Buffer or reorder and you silently corrupt the stream.

How Multigres forwards a NOTICE

Each backend message becomes a typed payload in one gRPC stream — `QueryResult | PgDiagnostic | PgNotification` — so notices ride alongside rows and the gateway re-emits each one the instant it arrives.



One payload class — rows, notices and LISTEN notifications share the path; forwarded in arrival order, no buffering

There were others

- **TLS pre-startup handshake** — a CVE-class footgun around buffered plaintext before the handshake completes
- **Startup parameters** (`client_encoding`, `application_name`, `search_path`, `DateStyle`) must influence the whole session over pooled backends
- **Extended-query `Describe`** — two forms (statement vs portal) that libpq breaks on if you confuse them

The list keeps growing as we find new corners.

PROVING IT

How do we know any of it works?

`pg_regress` + `pg_isolation_regress`

Postgres's own suites. Byte-for-byte output diff. The tests that gate every Postgres release.

A hundred failures that weren't bugs

Our Postgres was provisioned with production-tuned GUCs — small `work_mem`, low `random_page_cost`. The regression suite's expected output was captured on vanilla defaults.

Same answer, different EXPLAIN. Reset to defaults for the test run, and a whole batch of failures vanished at once.

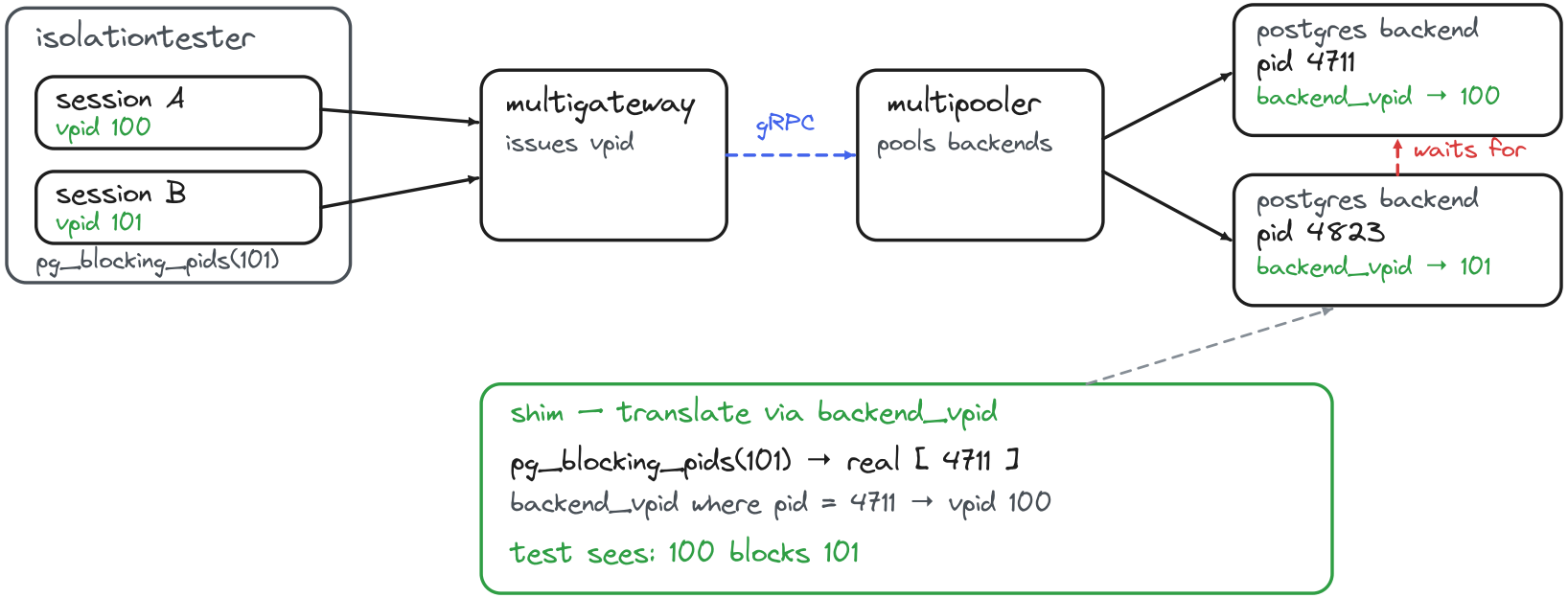
Some diffs are on purpose

Multigres reports parser error positions through the gateway with a different anchoring than direct Postgres. We'd rather not pretend otherwise.

We capture those as **patches** alongside the expected output, and require the actual output to match expected + patch.

Concurrency, through a pool

- Goal: run Postgres's own concurrency suite — `pg_isolation_regress` — through the proxy
- Specs use `pg_blocking_pids()`; through a pool those are the **wrong PIDs** — the lock-holder sits on a hidden backend
- Fix: multipooler records each backend's virtual PID in an unlogged `multigres.backend_vpid` table; a PL/pgSQL shim maps vpid → real backend PID at query time
- Result: 117 / 117 isolation specs pass, spec files untouched — only the tester's blocking probe is pointed at the shim



RESULTS

Measure compatibility. Don't assert it.

Baseline: PgBouncer — the de-facto standard — in its two common pool modes.

pg_regress, head to head

Multigres	PgBouncer (transaction)	PgBouncer (session)
222 / 222	**155 / 222**	**221 / 222**

pg_regress pass counts on PostgreSQL's own suite. Per-test breakdown lives alongside the harness in the repo.

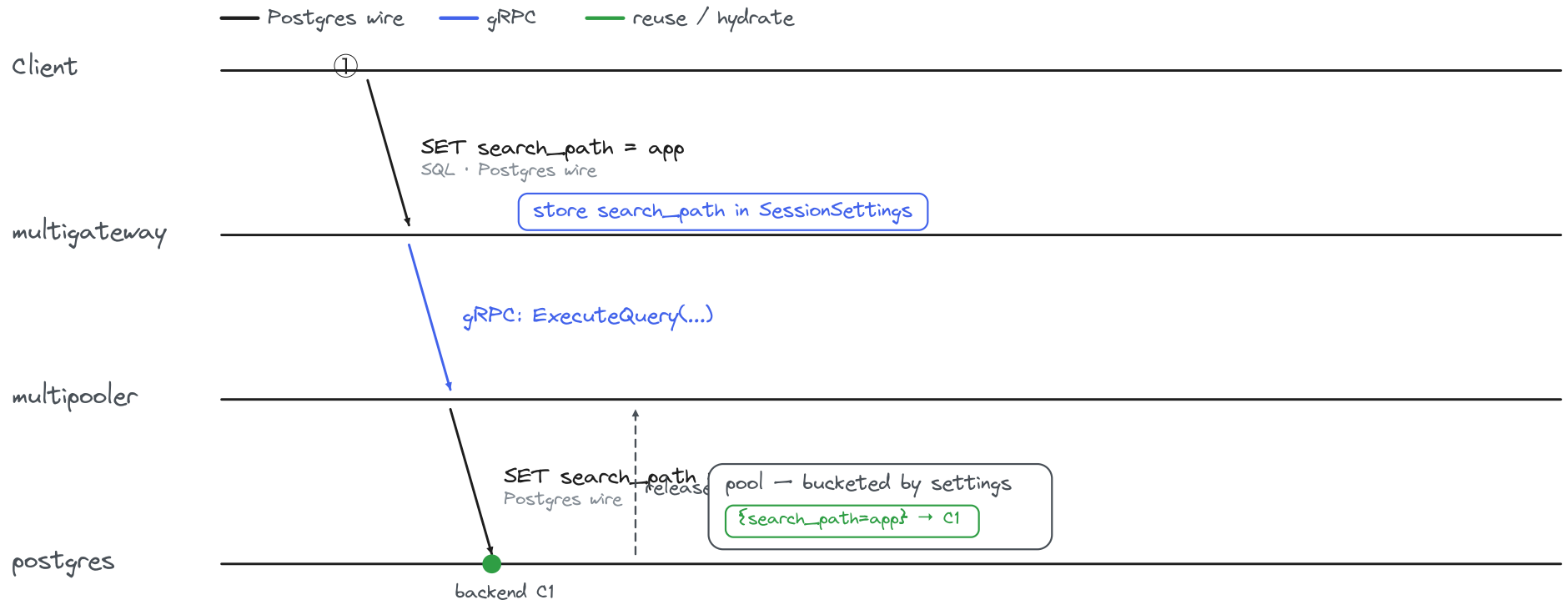
PgBouncer's knobs — and their limits

Recovers some: `max_prepared_statements` (protocol-level prepared statements) · `track_extra_parameters` (DateStyle, TimeZone, client_encoding).

No knob at all: SQL-level `PREPARE` · temp tables · `LISTEN/NOTIFY` · session advisory locks · `WITH HOLD`. Tied to a single backend transaction pooling won't pin.

Why Multigres keeps them

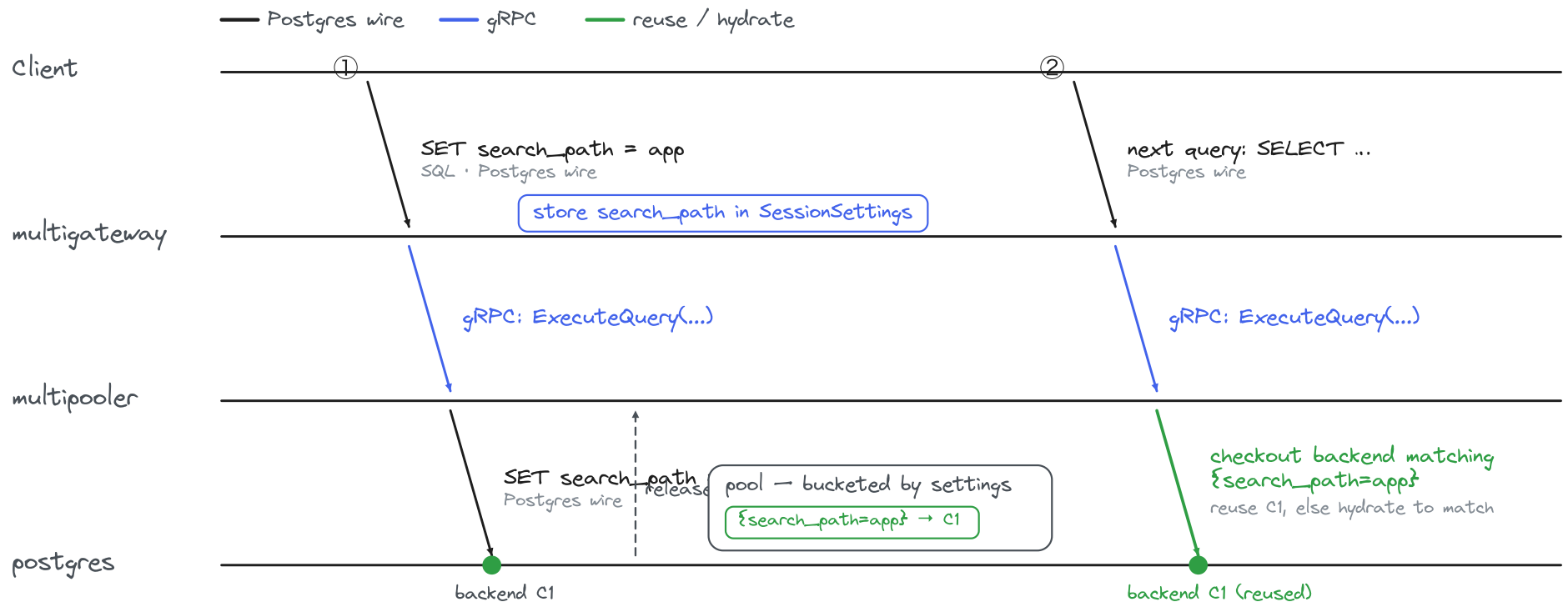
Each client connection carries its **own session state** — GUCs, prepared statements, role, open portals.



Reservation bitmask — `TXN · TEMP · PORTAL · COPY · LISTEN · LOGICAL REPL · ADVISORY LOCK · SETSEED · UNSAFE CONN` — pins a backend only as long as any reason holds, then returns it to the pool.

Why Multigres keeps them

Each client connection carries its **own session state** — GUCs, prepared statements, role, open portals.



Reservation bitmask — `TXN` · `TEMP` · `PORTAL` · `COPY` · `LISTEN` · `LOGICAL REPL` · `ADVISORY LOCK` · `SETSEED` · `UNSAFE CONN` — pins a backend only as long as any reason holds, then returns it to the pool.

Feature support across the modes

FEATURE	MULTIGRES	PGBOUNCER (TRANSACTION)	PGBOUNCER (SESSION)
`SET` / `RESET` (incl. `SET ROLE`)	✓	✗	✓
`PREPARE` / `DEALLOCATE`	✓	✗	✓
Temp tables (`ON COMMIT PRESERVE/DELETE ROWS`)	✓	✗	✓
`LISTEN` / `NOTIFY`	✓	✗	✓
`WITH HOLD` cursors	✓	✗	✓
`CREATE SUBSCRIPTION` (logical replication)	✗	✓	✓
`postgres_fdw` / foreign tables	✗	✗	✓
`CREATE` / `DROP DATABASE`	✗	✓	✓

Tradeoffs we accepted

gRPC isn't free for protocol bridges.

- It gives you streaming, structured errors, codegen
- It takes back schema-on-the-wire flexibility and Postgres's error model
- Plan for what you'll have to rebuild on top

Takeaways

- **Compatibility is a spectrum, not a checkbox.** The long tail is what separates adoptable from quietly broken.
- **Test with the database's own tests.** `pg_regress` was already written.
- **Diagnostic fidelity is non-negotiable.** Every error field exists because some tool relies on it.
- **gRPC isn't free for bridges.** Plan the rebuild.
- **Measure, don't assert.**

READ MORE

Building a Truly Compatible Postgres Proxy

Thank you

Questions?

Haritabh Gupta · haritabh.gupta@supabase.io

github.com/multigres/multigres · multigres.com