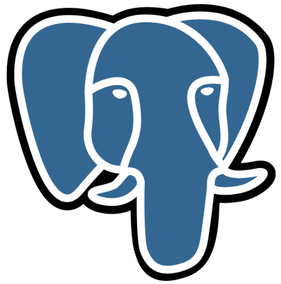
Electric SQL

Local first Architecture




App doesn’t WORK

A customer complaint that became an overhead

Let’s Debug

Talk with the customer

- Reached out to the customer personally
- Phone call + video call : ***Everything worked***
- Then they said: "*It doesn't work on the airplane.*"

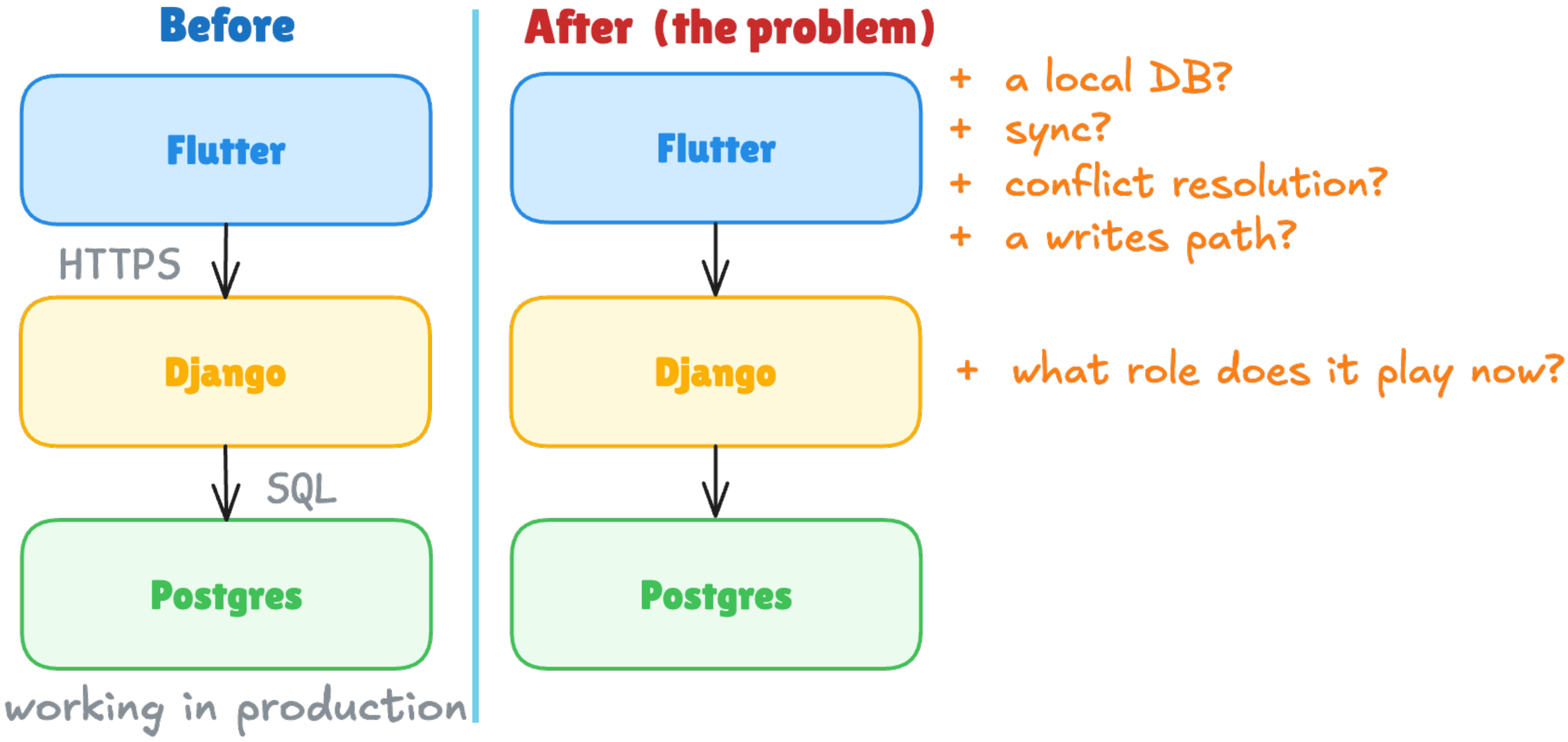
Not for ONE customer

Need evidence that it is a required feature.

Don't build local-first because one customer asked.

Build it because the data says you have to...

Our stack | The retrofit problem



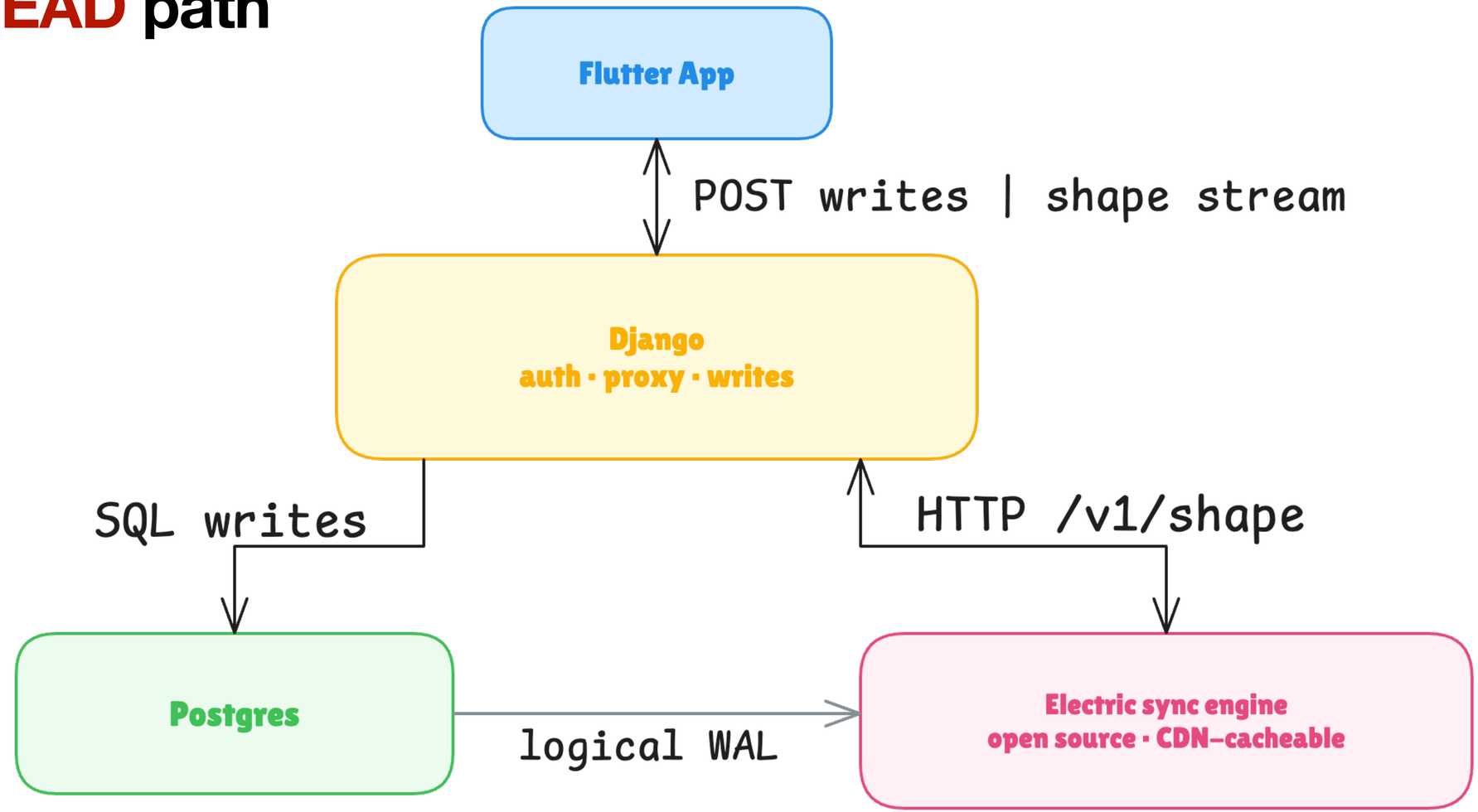
Choosing Electric

Why JSON over HTTP mattered

What that bought us	Why it mattered for retrofit
No new wire protocol	Our Django, our load balancer, our WAF, our observability, all already speak **HTTP**
No new socket framing	No **WebSocket** gateway, no message framing layer, no custom keep alive
Cacheable at every layer	CDN, reverse proxy, HTTP cache headers, Electric is a normal HTTP service
Proxy-able through Django	We could put auth, RLS, and rate-limiting in our existing Django middleware
Debuggable with curl	`curl https://{electric-api-server-url}/v1/shape?table=tasks`

Where does electric fits the stack ?

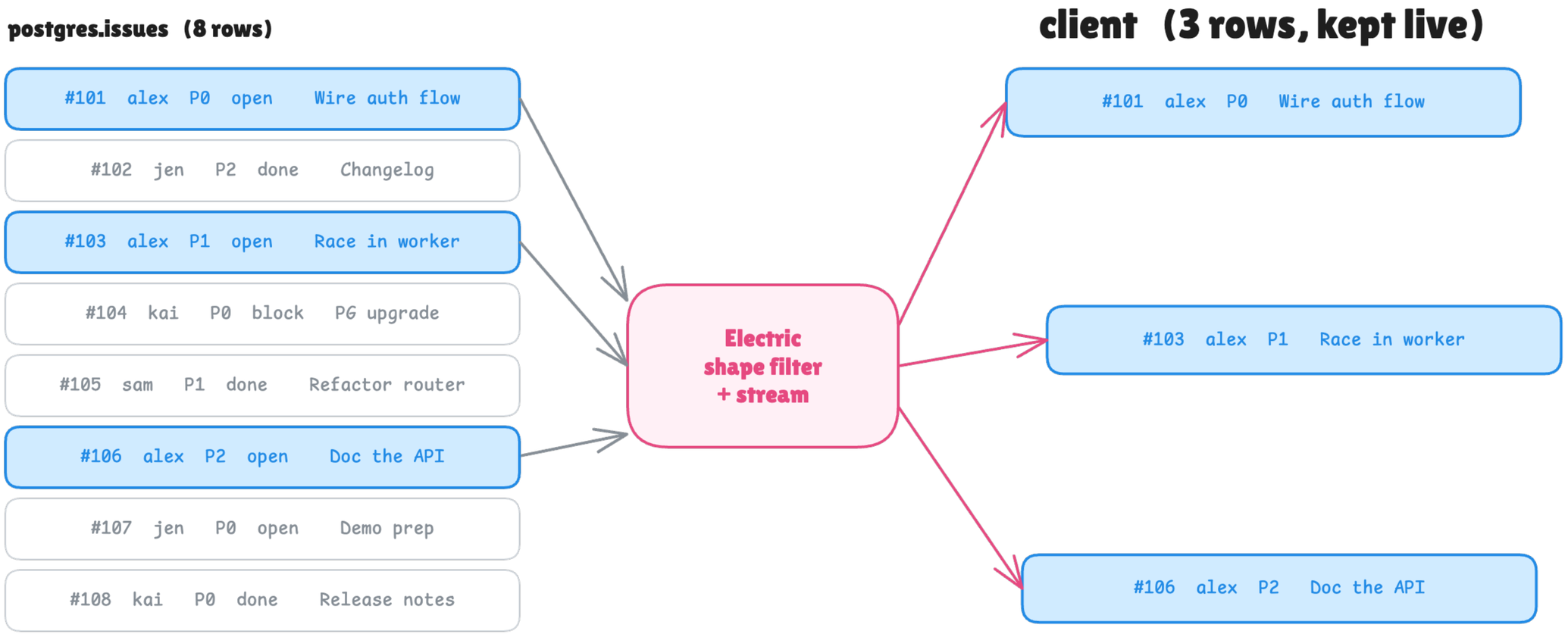
Only the READ path



Shape

A SQL query against Postgres.

```
SELECT * FROM issues WHERE assignee = 'alex' AND status = 'open'
```



https://electric.ax/docs/sync/guides/shapes

GET /v1/shape

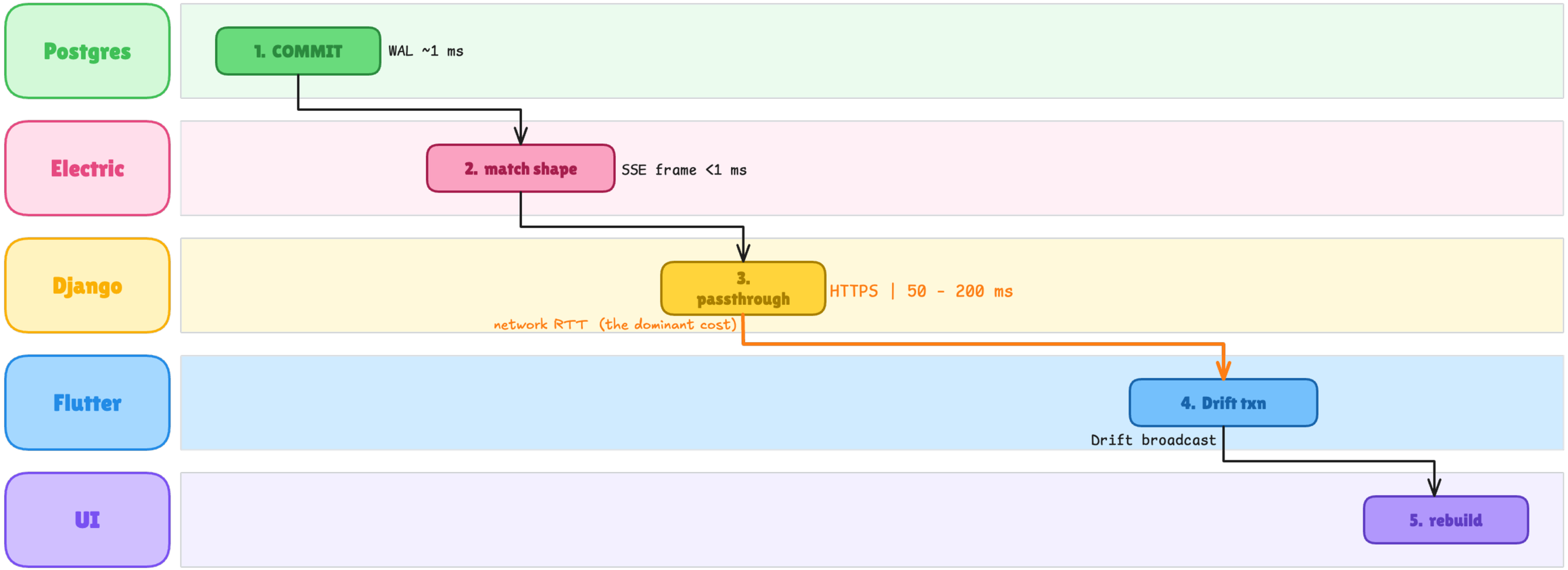
Response from Electric

- **Qualified table + primary key** (globally addressable)
- The actual Postgres row, **JSON-encoded**
- Every event is **typed**: insert, update, or delete

```
apoorvgarg@Apoorvs-MacBook-Pro ~ % curl -G "http://localhost:3000/v1/shape" \
    --data-urlencode "table=tasks" \
    --data-urlencode "offset=-1" | jq .
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100  1898    0  1898    0     0  27080      0 --:--:-- --:--:-- --:--:-- 27114
[
  {
    "key": "\"public\".\"tasks\"/\"t101\"",
    "value": {
      "created_at": "2026-06-05 00:50:59.002517+00",
      "id": "t101",
      "status": "open",
      "title": "Wire auth flow",
      "user_id": "alex"
    },
    "headers": {
      "relation": [
        "public",
        "tasks"
      ],
      "operation": "insert"
    }
  },
  {
    "key": "\"public\".\"tasks\"/\"t102\"",
    "value": {
      "created_at": "2026-06-05 00:50:59.002517+00",
      "id": "t102",
      "status": "done",
      "title": "Update changelog",
      "user_id": "jen"
    },
    "headers": {
      "relation": [
        "public",
```

When a row changes in Postgres

here's what happens -

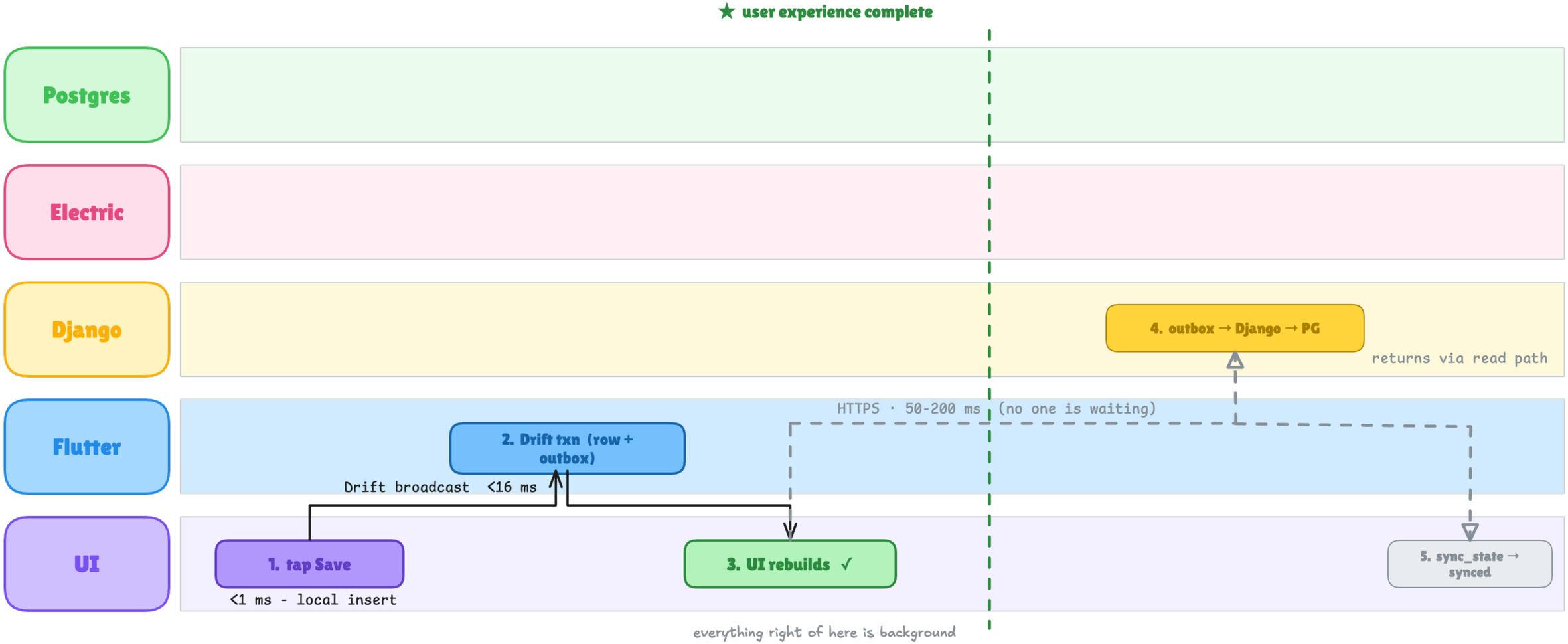


Total ≈ 80 – 250 ms

The SSE socket was already open and idle, waiting for a frame

Client Write Path

Electric Sync provides only read-only sync…



Total user-visible latency: <16 ms. Total sync latency: ~80-250 ms. Decoupled.

Read path and Write path are not the same path.

The user never waits.

Electric runs read-only against your database

smallest privilege grant that works

SQL setup

```
-- One dedicated Postgres role
CREATE ROLE electric WITH LOGIN
  REPLICATION
  PASSWORD '<secret>';

-- Scoped grants — only synced tables
GRANT USAGE  ON SCHEMA public  TO electric;
GRANT SELECT ON tasks          TO electric;
GRANT SELECT ON projects       TO electric;

-- Not granted:
--   INSERT · UPDATE · DELETE · TRUNCATE
--   CREATE · EXECUTE · superuser

-- Postgres-side prereq (postgresql.conf)
--   wal_level = logical
```

What this gives you

Read-only by contract

No INSERT, UPDATE, DELETE, TRUNCATE granted.
Enforced by Postgres, not by convention.

Scoped tables only

Other tables in your schema are invisible.
Electric can't even see them.

REPLICATION ≠ write

Authorizes tailing the WAL.
Does not authorize data modification.

Blast radius

Stolen credentials = SELECT on a few tables.
That is the entire blast radius.

Same in both modes

Self-host or Electric Cloud — identical role,
identical grants. Only the operator differs.

Deployment of Electric sync

Self-hosted

Electric runs inside your infrastructure

Electric sync service
Elixir, Docker, in your VPC

Postgres
your existing instance

Django proxy
your existing app

+ Data never leaves your VPC
+ Open source · fork-able
- You operate Elixir / BEAM
- You handle upgrades

Electric Cloud (managed)

ElectricSQL Inc. runs the sync service

Electric sync service
they run it

Postgres tunnel
your DB, secure conn

Django proxy
your existing app

+ No ops surface
+ Faster to ship
- DB connection from their infra
- Vendor in the data path

Scale

CDN fan-out

100k – 1M

concurrent users per
Electric instance

CDN does fan-out, Electric does work once

Live update latency

6 ms FLAT

with optimised where clauses
(3 ms PG + 3 ms Electric)

10 shapes or 10k shapes – same

Write throughput

5,000 / sec

changes per second,
any shape count (optimised)

WAL → shape log indexing

Non-optimised clauses

1400 → 140

changes/sec at 10 → 100 shapes
inversely proportional

ILIKE / regex / functions

https://electric.ax/docs/sync/reference/benchmarks

Conflict Resolution (bonus)

Last Write Wins

LWW

Server stamps updated_at.
Newer write overwrites older.
Audit log keeps the loser.

PRO

Trivial. One column.
Clock skew sidestepped
(server stamps).

CON

Whole-row LWW.
A's title edit and B's done
edit on same row =
one of them lost.

Per-field LWW

column-level timestamps

Each column has its own
updated_at. Different fields
on same row both survive.

PRO

Survives concurrent edits
to different columns.

CON

N timestamps per row.
Schema bloat.
Still LWW inside a column.

CRDTs

Yjs · Automerge · etc.

Data types whose operations
commute. Any order of
application = same result.

PRO

Mathematically convergent.
No data loss.
Best for text editors, lists.

CON

Metadata grows with edits.
Library footprint.
Doesn't map to all data types.

User-mediated

show both · let user choose

Detect the conflict. Show
both versions in the UI.
User picks the winner.

PRO

Always correct
(the human is authoritative).

CON

Bad UX if frequent.
Good as a fallback layer.
Legal docs · finance.