PRODUCT ARCHITECTURE

How this should be built — what connects to what, and why each piece earns its place.

A 13-page architecture brief for the Visit Operations Platform — a scheduling tool that sits between Science 37's existing Web App (study + participant data) and Patient Mobile App, and pushes committed visits out to provider calendars and Zoom. Covers services, data, the scheduling engine, integrations, and deployment.

1+2
modular monolith
+ 2 extracted services

9
domain modules
inside the monolith

3
engine modes
one shared ranker

9
external integrations
with explicit contracts

Harun Tuncelli
Sr. Product Designer

01

EXECUTIVE SUMMARY · THE ARCHITECTURAL BET

One modular monolith. Two extracted services. Boring choices everywhere else.

THE BET

Monolithic core, two surgical extractions, two upstream Sci 37 systems. The Visit Operations Platform doesn't own studies, appointment types, or the participant list — those sync in from the Science 37 Web App. Patient preferences (preferred dates, address, contact) sync in from the Patient Mobile App. We own the scheduling engine, the booking transaction, and the calendar UI. After commit, visits fan out to the Patient Mobile App, provider Google Cal / Outlook, and Zoom.

- **Don't re-own data Science 37 already owns. Own the scheduling decision and the booking transaction.**

PG

Postgres for everything operational. Strong constraints. Real transactions.

Audit

Append-only event log to a dedicated store. 21 CFR Part 11 posture.

SOC 2

PHI in PG; field-level encryption; HIPAA + SOC 2 from day one.

Interactive

latency budget for calendar / search reads — sub-second end-to-end as a design target.

THREE PRINCIPLES THAT DRIVE THE DESIGN

1. **Slices are first-class data — not UI sugar.** Every appointment is a parent record with N child role-slice rows. The calendar, the optimizer, the audit log, and the sponsor export all read the same model.
2. **License safety is enforced at the database level.** Booking constraints check eligibility against the staff/license join in a single transaction. No code path can produce an illegal assignment.
3. **Audit is a write-once primitive, not a feature.** Every mutation in the booking domain emits an event to the audit log before the transaction commits. Bulk operations get a 10-second reversal window backed by a soft-delete flag, not a re-create dance.

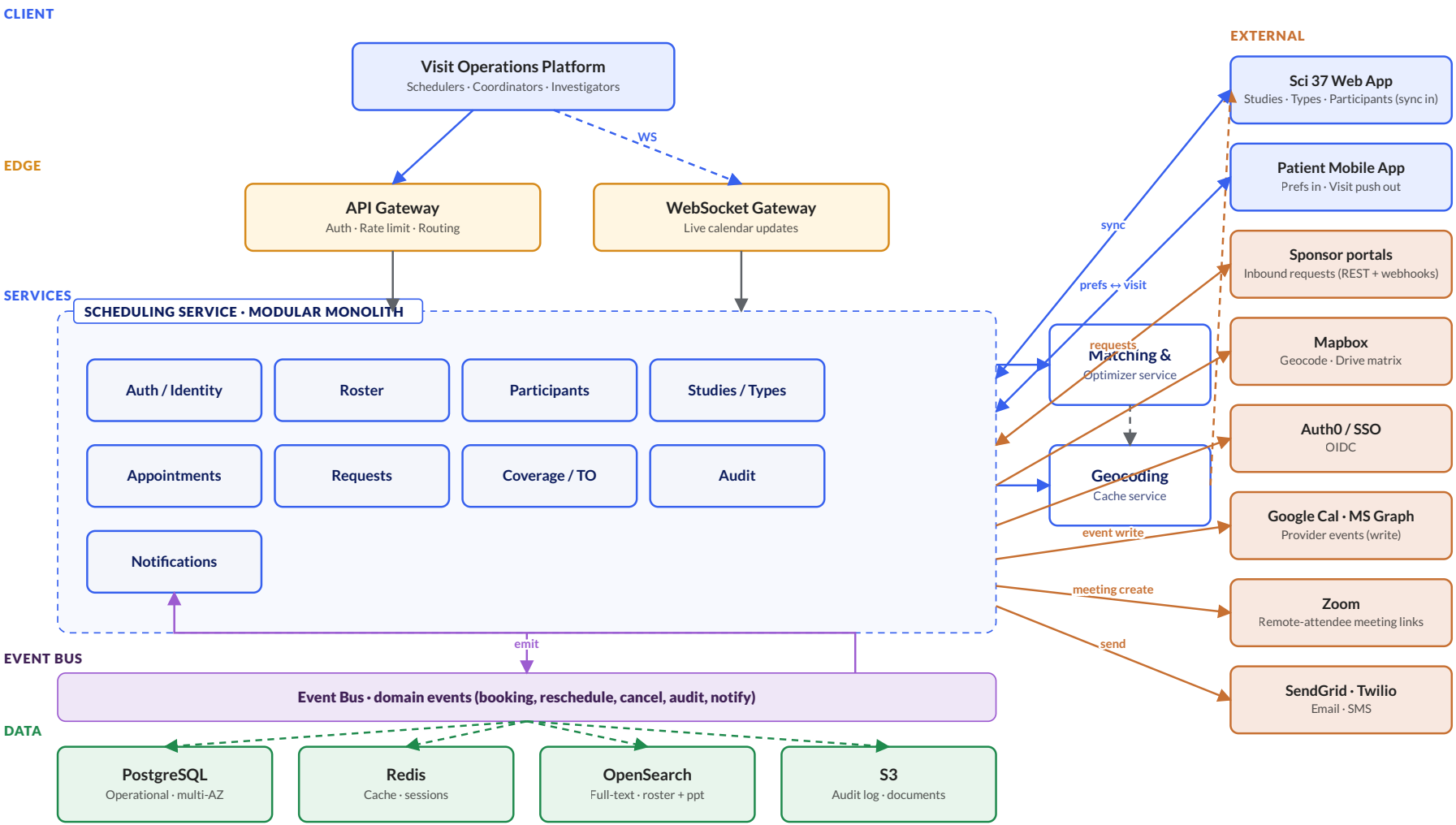
WHAT WE EXPLICITLY DO *NOT* BUILD (YET)

- → No re-implementation of Studies, Appointment Types, or Participant identity — the Science 37 Web App owns these; we read replicas, never write.
- → No homegrown ML — matching is deterministic and rule-based (license + study + distance + workload). ML only enters as a ranker on top of the rules-eligible pool, when we have labeled data.
- → No microservices per domain — premature for current scale; cost exceeds benefit.
- → No multi-region active-active — RPO/RTO targets are hours, not seconds. Multi-AZ primary + warm DR.

02

SYSTEM OVERVIEW

The Visit Operations Platform sits between two existing Science 37 systems and the sponsor request feed — and pushes scheduled visits out to provider calendars and Zoom.

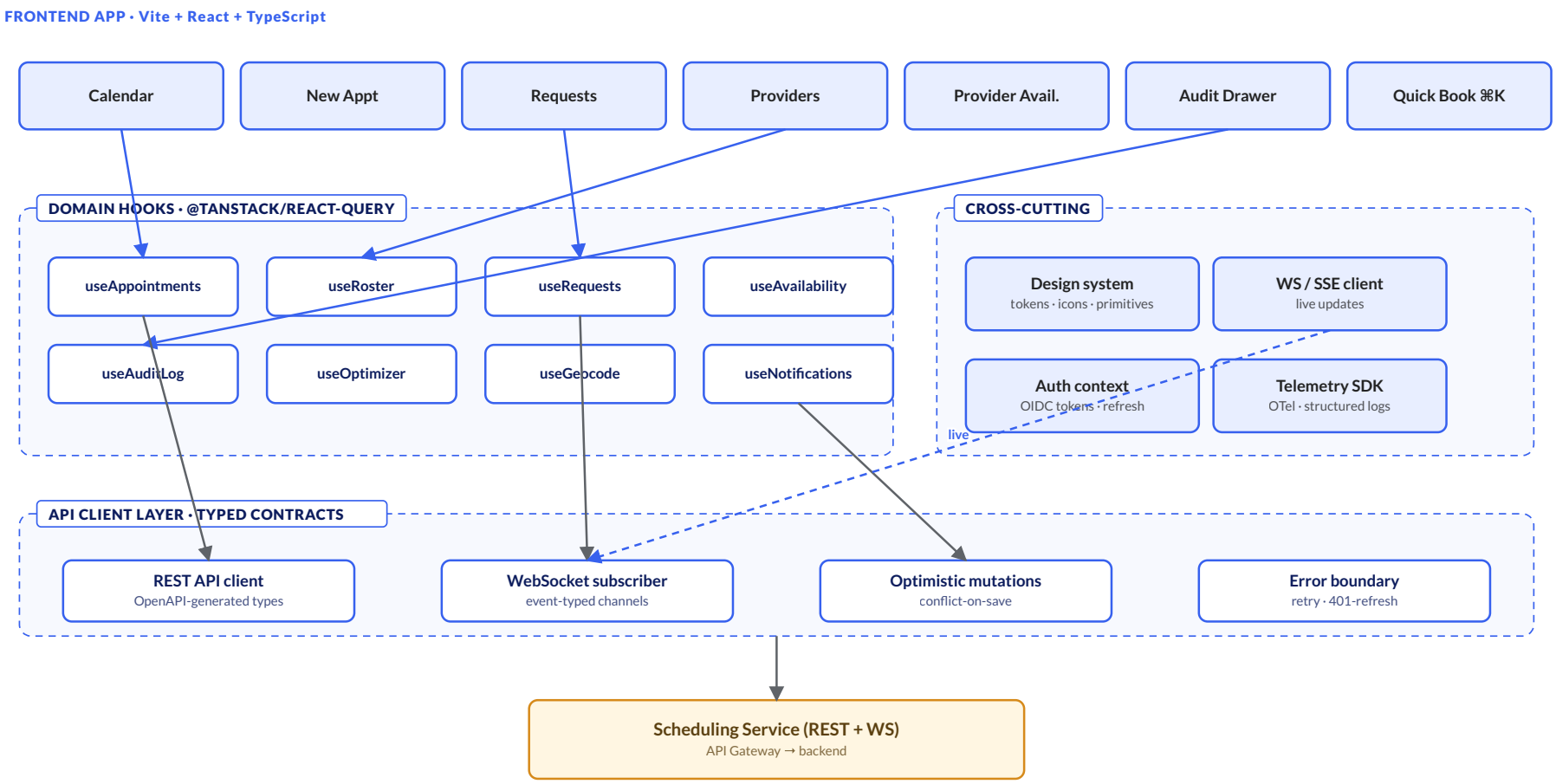


Upstream: Studies, Appointment Types, and the Participant list flow in from the Science 37 Web App; participant preferences (name, address, preferred dates/times) come from the Patient Mobile App; sponsor visit requests arrive via signed REST + webhooks from each sponsor portal. **Downstream:** when a scheduler commits an appointment, the Visit Operations Platform pushes the visit to the Patient Mobile App, writes a calendar event into each assigned provider's Google Calendar / Outlook, and creates a Zoom meeting link for any remote attendees (CRC, Investigator). Sync API · WebSocket · Domain event · Data read/write · External integration

03

FRONTEND ARCHITECTURE

Vite + React + TypeScript. Domain hooks for state. Typed API contracts.

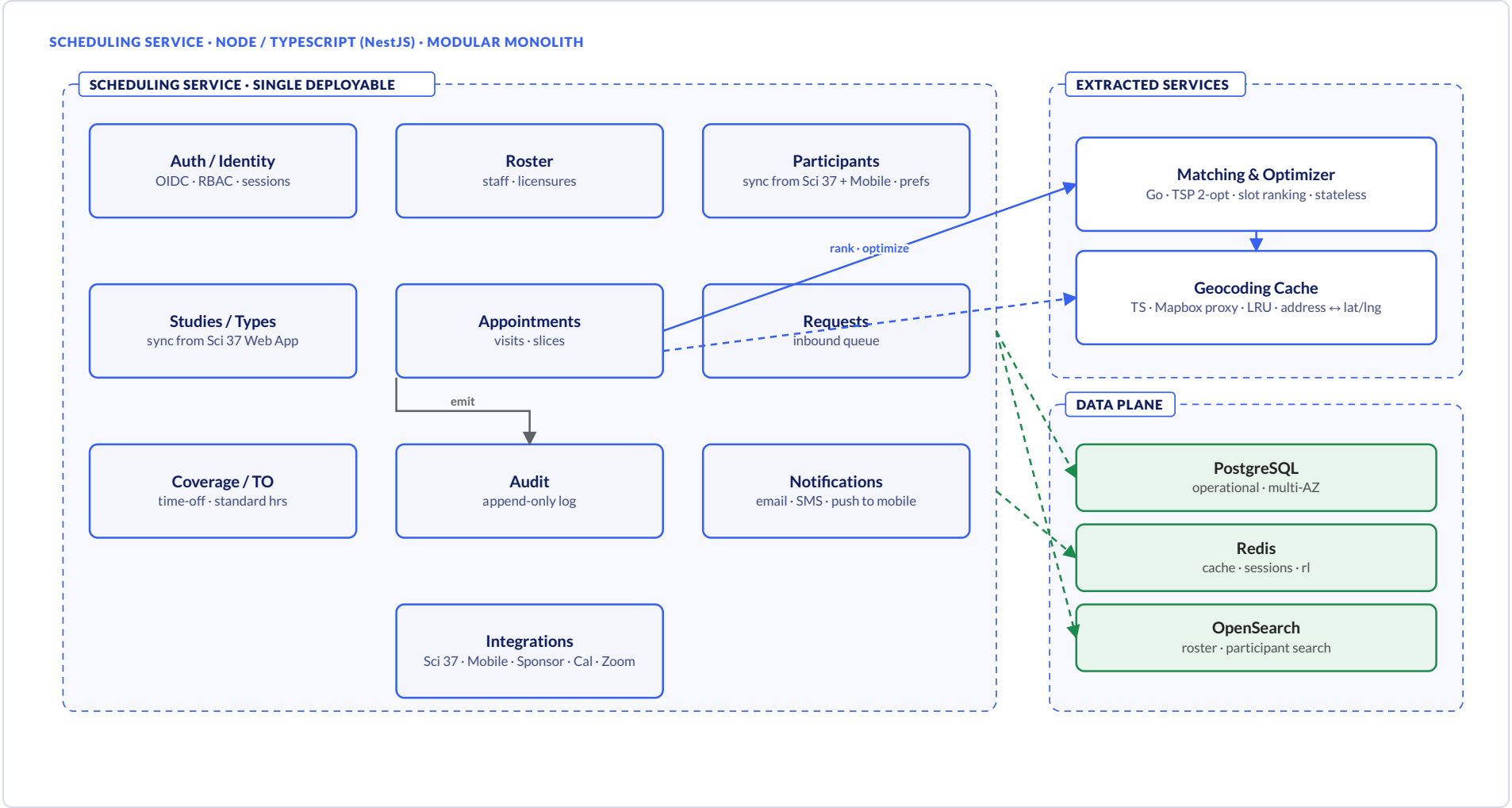


Each top-level route is a shell with its own data dependencies. **Domain hooks** wrap TanStack Query mutations and reads — they're the only place that knows about the API surface. **Optimistic mutations** with conflict-on-save replace pessimistic locking. **WebSocket / SSE** pushes calendar deltas so two schedulers see the same state without polling.

04

BACKEND SERVICES

9 domain modules in one deployable. Optimizer + geocoder extracted on principle, not premature instinct.



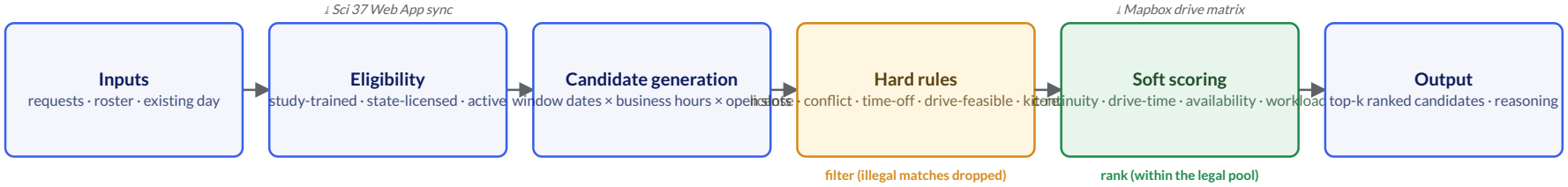
Data ownership: Studies, Appointment Types, and the Participant list are *synced from the Science 37 Web App* — we cache locally for read latency and join-friendliness, but the Web App is the source of truth. Participant preferences (preferred date/time windows, address, contact info) flow in from the Patient Mobile App. **Why monolith:** the booking transaction (license check + conflict check + commit + audit) crosses 4 modules and has to be ACID. Splitting into services would force distributed transactions or eventually-consistent compromises that the regulated context can't absorb. **Why extract optimizer:** CPU-bound, batchable, separately scalable. **Why extract geocoder:** latency-sensitive, externally rate-limited (Mapbox), benefits from process-level caching.

05

SCHEDULING ENGINE

One ranker behind Auto Suggest, Optimize Day, and Bulk Schedule.

SHARED ENGINE PIPELINE · ONE RANKER, THREE MODES



MODE A Auto Suggest

INPUT 1 open request

PIPELINE eligibility → candidates within visit window → hard-rule filter → soft score → top-1 + 2 alternates

CONTINUITY RNs / CRCs / INVs the participant has already seen win the tie — Sci 37's "same nurse for most visits" principle in code

KIT-READY hard rule, not a label: any candidate date earlier than the kit ETA is dropped before scoring. Mirrors Sci 37's stated workflow — supplies arrive 1–2 weeks pre-visit, IMP 1–2 days before dosing

OUTPUT proposed (date · time · provider trio) with a reasoning chip; "Same as last visit" badge when continuity wins; reasoning surfaces when kit ETA was the binding constraint

LATENCY interactive — runs inline on the New Appointment screen; sized to feel instantaneous to a scheduler typing

Use when a scheduler is working a single request and wants the engine to do the legwork.

MODE B Optimize Day

INPUT one provider's existing day (≤ 12 visits)

PIPELINE reuse current candidates → greedy initial route → 2-opt swap pass minimizing total drive minutes

OUTPUT reordered same-day schedule with explicit "saved X minutes / Y miles" delta

LATENCY sub-second on a per-day workload — runs in the extracted optimizer service so it's separately scalable

Use after a day is roughed in and the scheduler wants the route tightened. No new bookings; only resequencing.

MODE C Bulk Schedule

INPUT M open requests (up to ~100 per run)

PIPELINE per-request match in parallel → cross-provider 2-opt swap pass on the placed set → atomic insert of all M

OUTPUT M visits in one transaction + 10-second undo + audit event with all M IDs in `subject_ids`

LATENCY minutes-not-seconds; backgrounded in the optimizer service and sharded by region; progress streams over WS

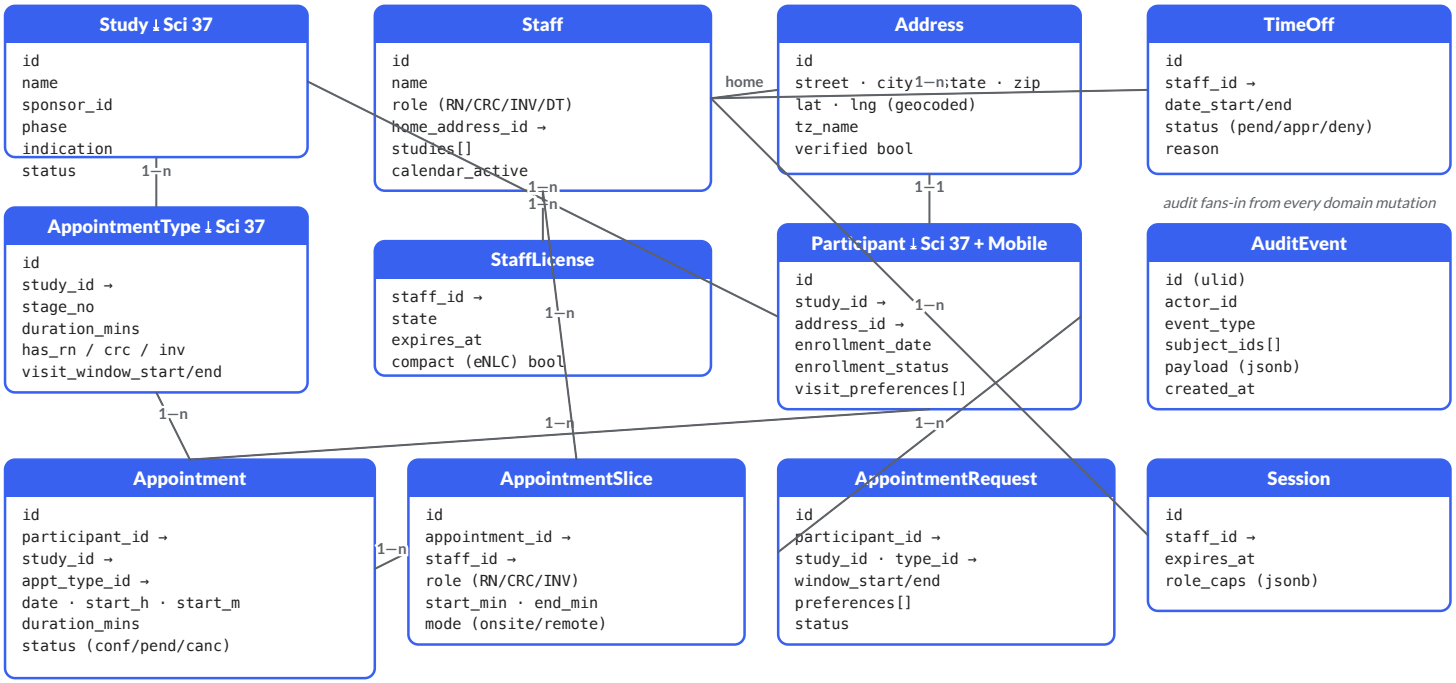
Use to clear an inbound queue. The cross-provider swap is what separates this from "loop Auto Suggest 87 times."

06

DATA MODEL

Eleven core entities. Slices live as their own table. Audit is append-only.

CORE ENTITIES · POSTGRESQL · 11 TABLES

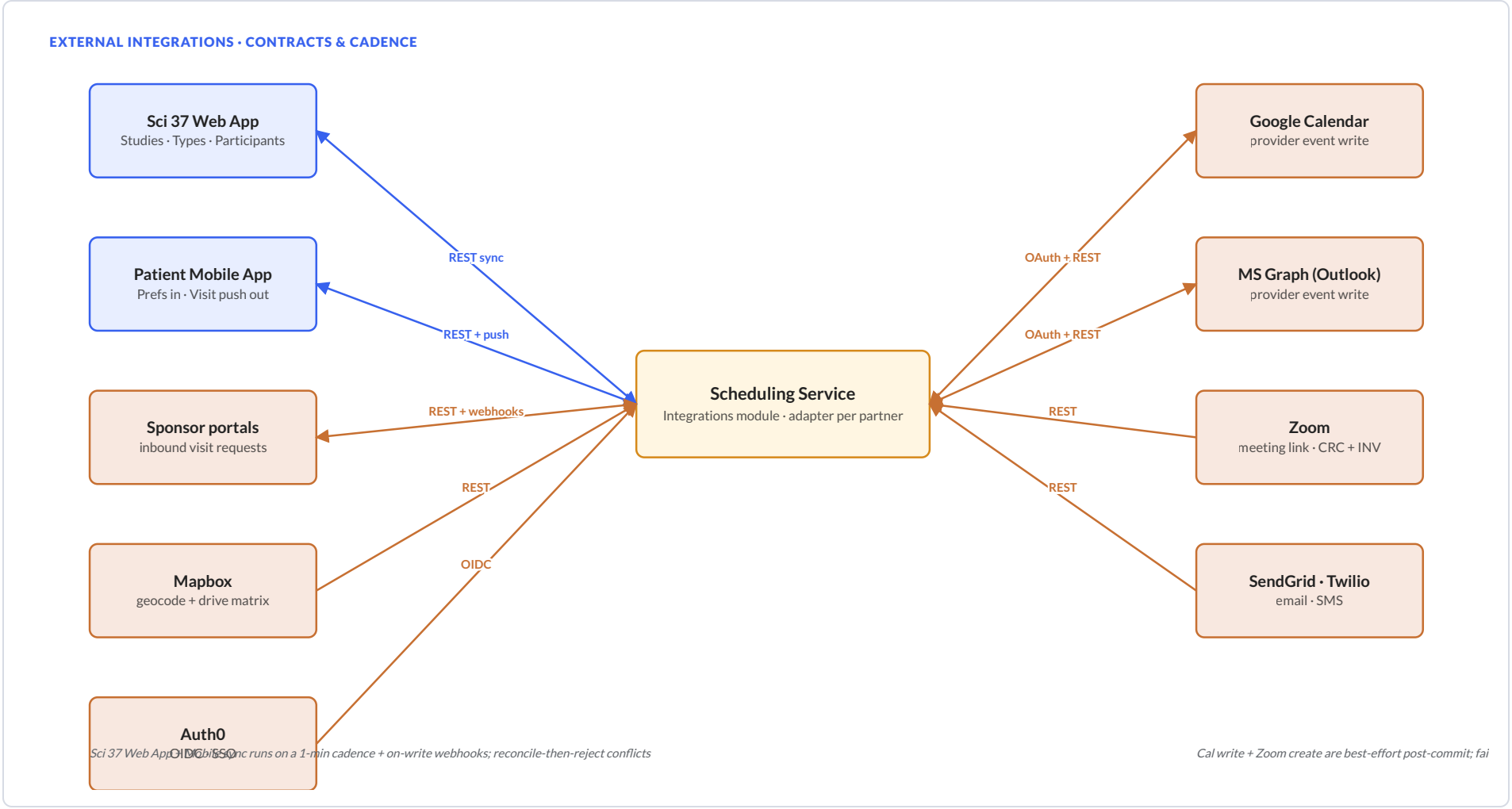


Entities marked **⤓ Sci 37** are read replicas of records owned by the Science 37 Web App — kept in PG for query locality, but reconciled on a sync cadence; we never update them directly. Participant rows merge an upstream identity payload (Sci 37) with preference data the patient enters in the Mobile App. **AppointmentSlice** is the load-bearing structural choice — it's why role-attendance reporting is a query, not a project. **Audit** is append-only, ulid-keyed; subject_ids is a Postgres array so a single event can reference all 87 appointments in a bulk auto-schedule.

07

EXTERNAL INTEGRATIONS

Two Science 37 first-party systems plus the sponsor request feed on the inbound side. Five third-party services on the delivery side.

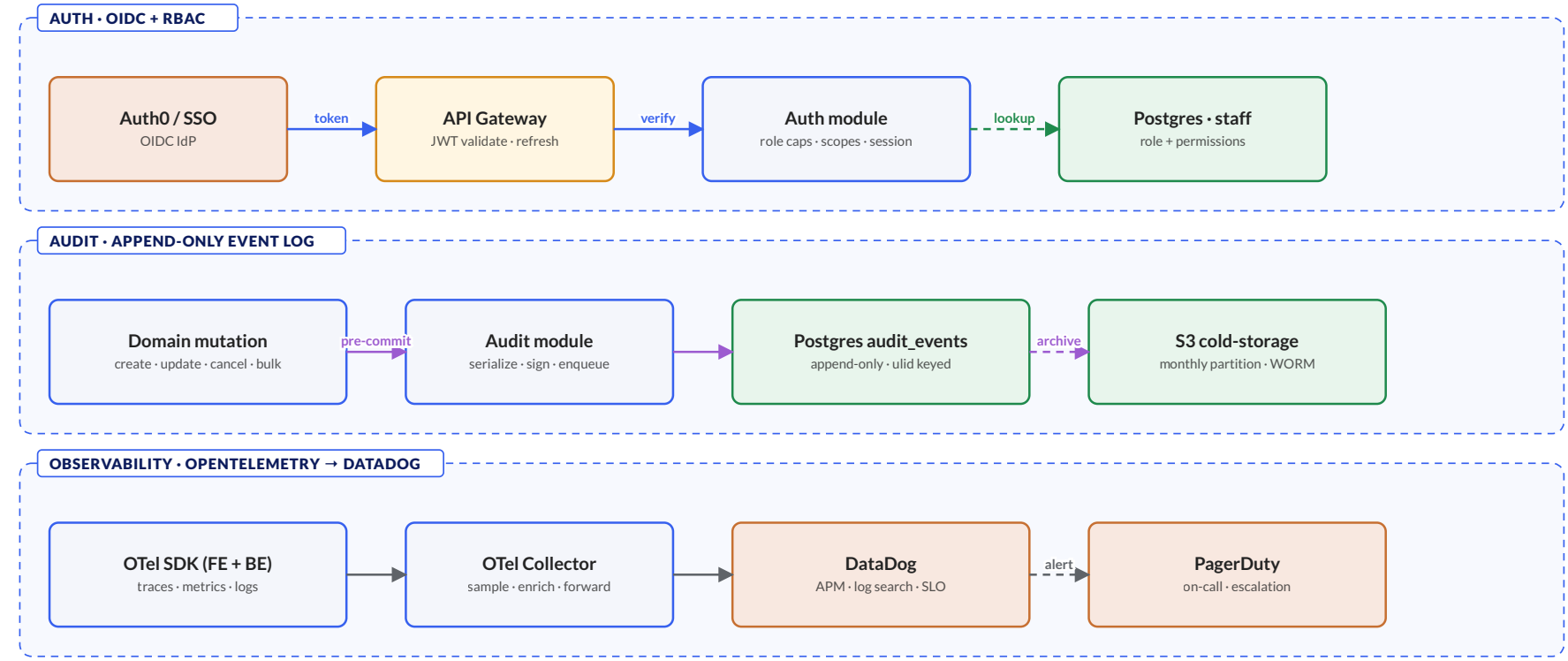


Adapter pattern: each integration lives behind a typed adapter interface inside the Integrations module. **Sci 37 Web App** is the source of truth for Studies, Appointment Types, and the Participant list — synced on a 1-min cadence + on-write webhooks. **Patient Mobile App** is the channel for participant-entered preferences (preferred dates, address, contact details) coming in, and for scheduled-visit notifications going out. **Sponsor portals** push new visit requests via signed webhooks (HMAC, idempotent by event_id) and accept status callbacks via REST. **Google Calendar / MS Graph** get a written event for every assigned provider when an appointment commits. **Zoom** generates a meeting link for any visit with remote attendees (CRC, Investigator); the link rides on the calendar event and the patient-app push.

08

CROSS-CUTTING CONCERNS

Auth, audit, and observability — the three planes that touch every request.

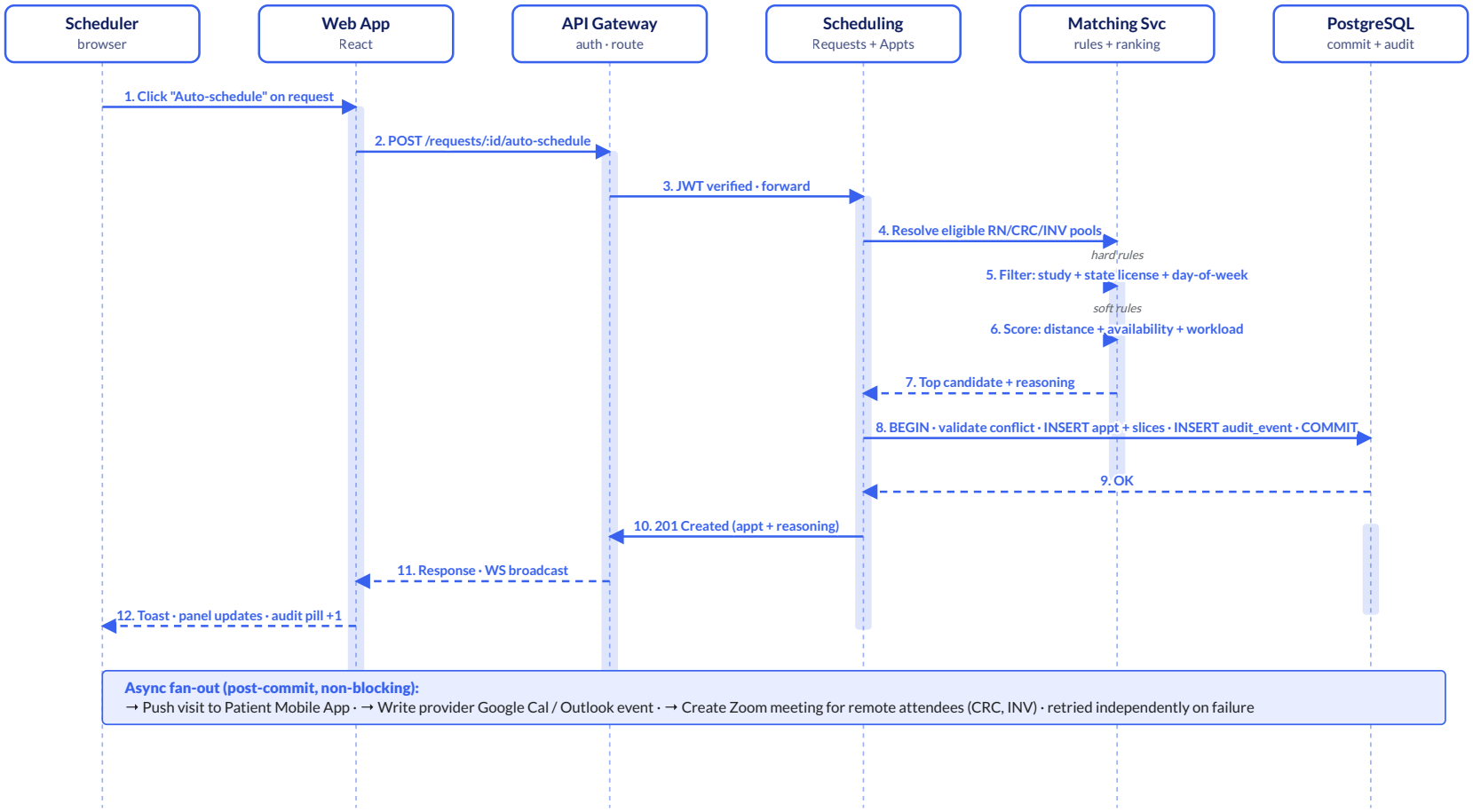


Auth: OIDC tokens validated at the gateway; role capabilities resolved at the auth module from the staff table. **Audit:** domain mutations write the audit event in the same transaction as the data change — no possible drift between "what happened" and "what was logged." Cold-storage partition rolls to S3 monthly with WORM lock. The audit posture is engineered to support an FDA inspection at the standard Science 37 has already cleared on five Phase-3 trials — **"No Action Indicated," no Form 483 issued**. **Observability:** single OTel pipeline; one ID propagates from browser click to database row.

09

SEQUENCE · AUTO-SCHEDULE A REQUEST

From "Auto-schedule" click to committed visit — the happy path.

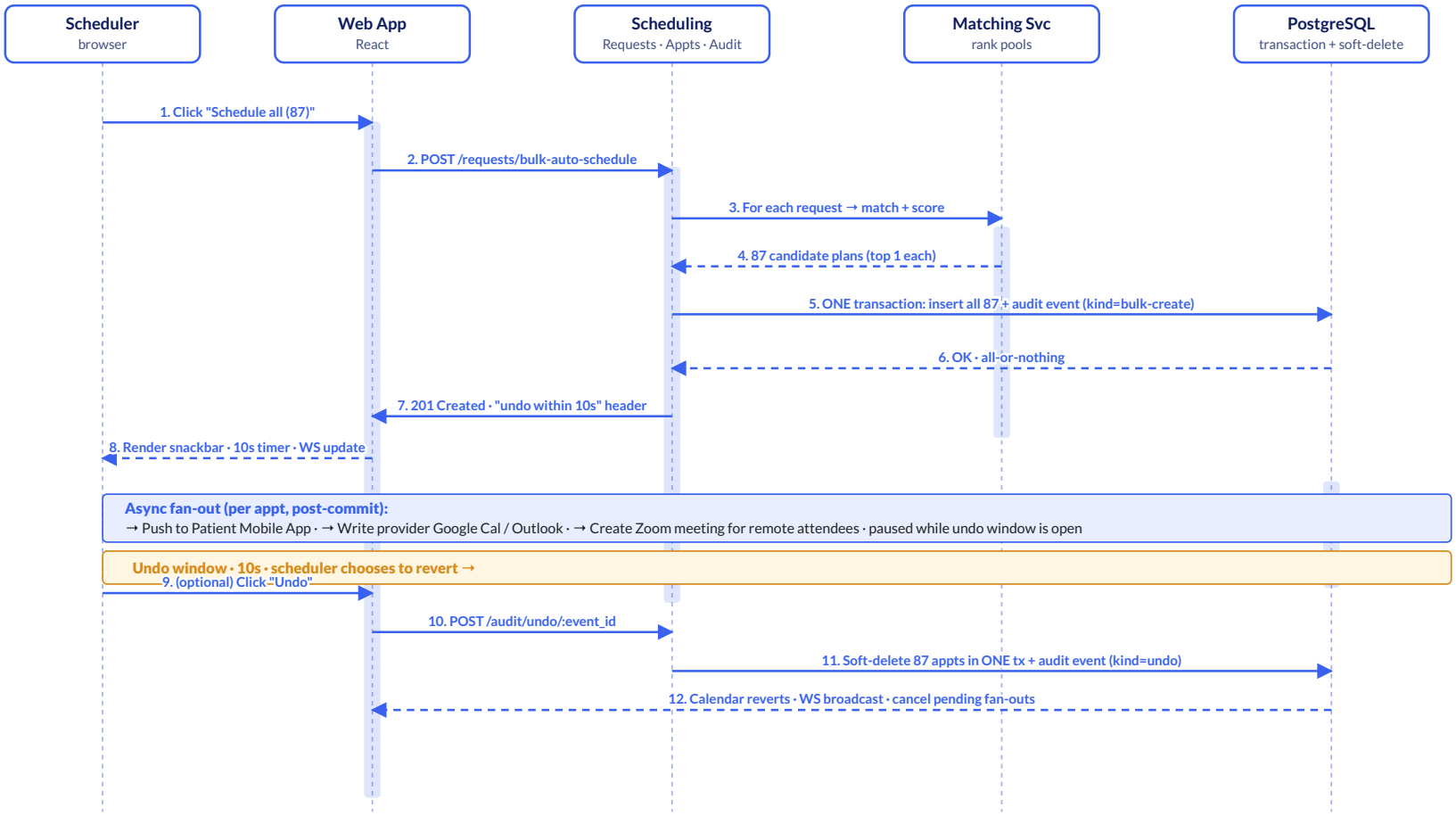


Hard rules (license, study eligibility, conflict) **filter** the pool — never propose an illegal match. Soft rules (distance, availability tier, recent workload) **rank** within the legal pool. The booking transaction is a single SQL transaction: validate → insert appointment + slices → insert audit_event → commit. **Post-commit fan-out** (the highlighted bar) is intentionally async: the patient-app push, provider calendar write, and Zoom meeting create all happen after the user sees the success toast, retried independently — a calendar API hiccup never blocks a booking.

10

SEQUENCE · BULK AUTO-SCHEDULE + UNDO

87 visits committed atomically — and reversible for 10 seconds.

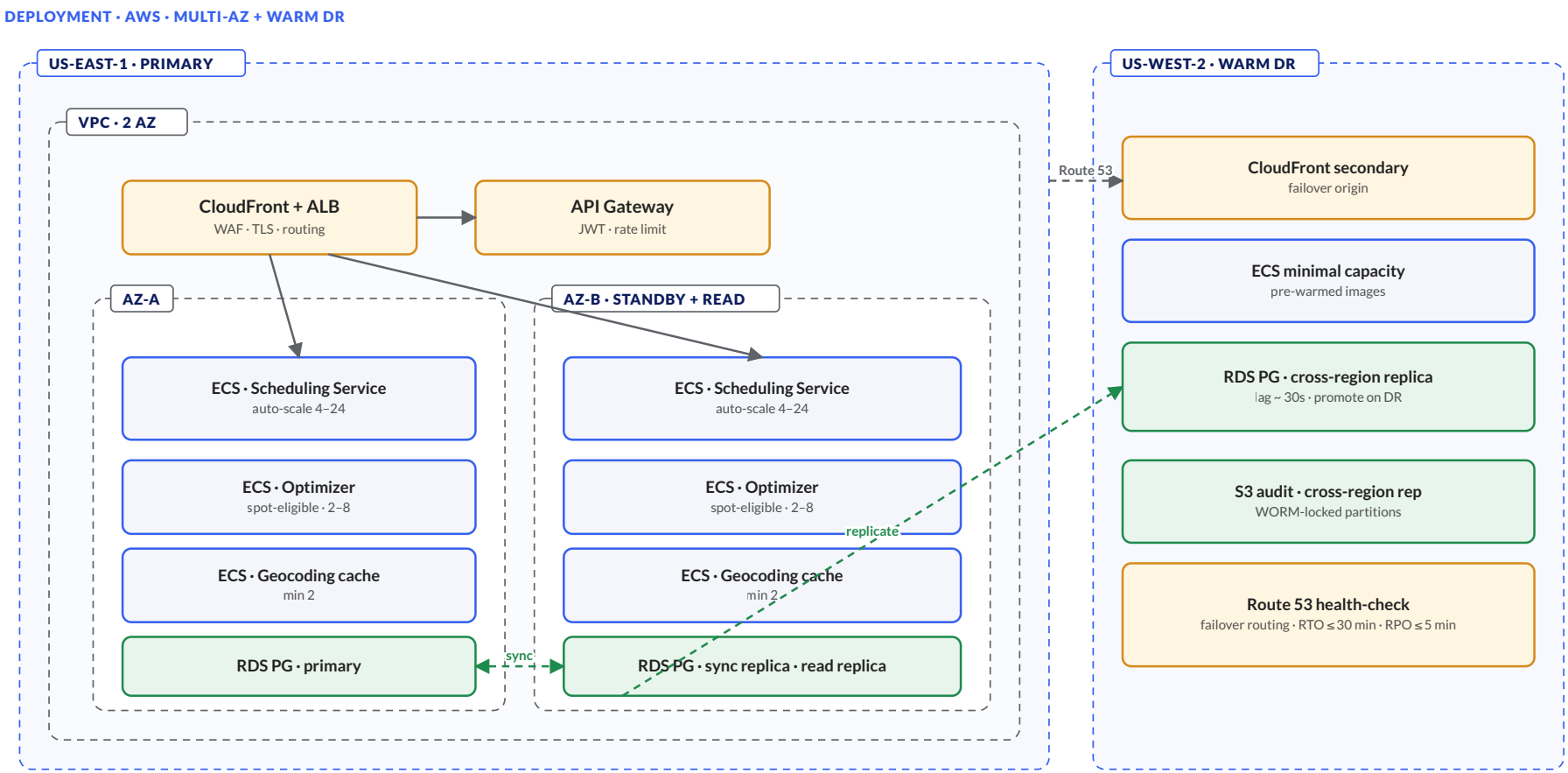


The bulk action commits as **one transaction** — either all 87 succeed or none do. **Post-commit fan-out** (Patient Mobile push, provider Cal write, Zoom create) is queued but **held during the 10-second undo window** — if the scheduler reverts, the queue is cancelled before any external surface sees the booking. After the window, the queue drains in parallel; per-appt failures retry independently. **Undo** is a soft-delete (status flip + tombstone), not a re-create — appointment IDs survive for audit linkage. After 10s the undo endpoint refuses; reversal becomes a manual cancel-and-rebook.

11

DEPLOYMENT TOPOLOGY

One primary region, two AZ, warm cross-region DR. RTO ≤ 30 min, RPO ≤ 5 min.



Primary: ECS-on-Fargate per service, multi-AZ, RDS Postgres with sync replica + read replica. **DR:** minimal warm capacity in us-west-2 with cross-region read-replica and S3 cross-region replication. **Failover:** Route 53 health-check flips traffic; RDS replica promotes; ECS scales out from warm pool. **Targets:** RTO ≤ 30 min, RPO ≤ 5 min — sufficient for a regulated operations product, simpler than active-active.

12

NON-FUNCTIONAL REQUIREMENTS & CLOSING PRINCIPLES

The targets the architecture is built to hit — and the trade-offs it accepts.

LATENCY BUDGETS (DESIGN INTENT — TO BE BENCHMARKED)

- Calendar / search reads: sized for an interactive feel; cached + indexed reads, server-side never blocks on Mapbox
- Booking write (single visit): completes inside a single interaction without spinner; audit + WS broadcast on the same transaction
- Optimize Day: sub-second on a typical provider-day; runs in the extracted optimizer so calendar reads don't degrade
- Geocoding: served from the cache layer for the steady state; falls back to Mapbox for new addresses, behind a circuit breaker

AVAILABILITY & RESILIENCE

- API SLO: 99.9% monthly target · multi-window burn-rate alerts
- Multi-AZ active/standby; warm cross-region DR (RTO 30m / RPO 5m)
- Booking transaction: serializable isolation; conflict-on-save UI
- External adapters: circuit-breaker; degraded mode keeps core read-only

COMPLIANCE & SECURITY

- HIPAA: PHI in PG with field-level encryption · BAAs with all subprocessors
- 21 CFR Part 11: append-only audit · e-signatures on critical actions
- FDA inspection-ready: holds the bar Science 37 has cleared on 5 Phase-3 trials (No Action Indicated, no Form 483)
- SOC 2 Type II: control library + continuous evidence collection
- Auth: OIDC + RBAC + per-action scoping; tokens rotate ≤ 1h

SCALE (YEAR 1, SINGLE-TENANT SCIENCE 37)

- ~5,000 staff in roster · ~30,000 active participants across active studies
- ~120k appointments / month · ~6× that in role-slices · ~480k audit events
- ~50k inbound webhook events / day from sponsor portals
- API throughput sized for a couple hundred concurrent schedulers; auto-scales 4 → 24 tasks under load

1 **Slices are first-class data, not UI.**
Every view (calendar, panel, audit, sponsor export) reads the same `appointment_slices` table. Role-attendance reporting is a query, not a project.

2 **License safety enforced at the database.**
Booking constraints check eligibility against `staff_licenses` in the same transaction. No code path can produce an illegal assignment.

3 **Audit is a transactional primitive.**
Audit events write in the same SQL transaction as the data change they describe. There is no possible drift between "what happened" and "what was recorded."

4 **Boring choices everywhere it doesn't matter.**
Postgres for everything operational. ECS instead of Kubernetes. Modular monolith instead of microservices. Reserve novelty for the optimizer and the slice model — the two places where it earns return.