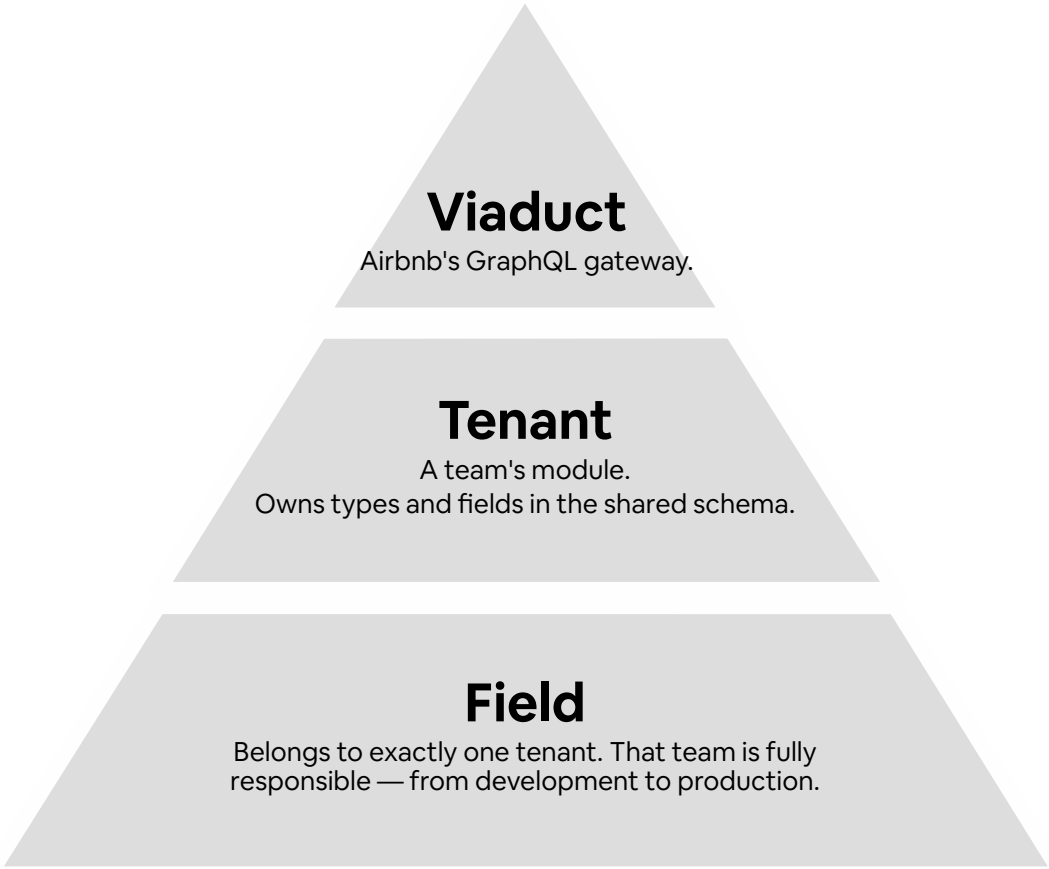
Observability for a Multi-Tenant GraphQL Gateway at Scale

Vickey Yeh

👋 I'm Vickey Yeh

Senior Engineer @ Airbnb

One process. 600 tenants.


Viaduct
Airbnb's GraphQL gateway.
Tenant
A team's module.
Owns types and fields in the shared schema.
Field
Belongs to exactly one tenant. That team is fully responsible — from development to production.

All operating as a single service.

~600	200+	1.5M
tenant modules	changes merged daily	lines of application code

Who's responsible?

```
query StaysSearch {
  listing(id: $id) {
    price { total }          #  ← pricing
    availability { nights } #  ← availability
    reviews { rating }       #  ← reviews
    photos { url }           #  ← media
  }
}
```

Attribution

Every field is owned by exactly one tenant.

System

Gateway-wide health. Owned by the Viaduct core team.

Operation

One named GraphQL operation, e.g. StaysSearch.

Tenant

One module's contribution to an operation.

Field

One specific schema field. The finest grain.

Every downstream call is instrumented. Automatically.

Tenants don't wire up tracing. The framework does.

It's a Tuesday afternoon.
StaysSearch p95 latency just spiked.

A path to a name

Step 1

Operation dashboard

p95 spike, all platforms affected

A path to a name

Step 1

Operation dashboard

p95 spike, all platforms affected

Step 2

Tenant overview

one outlier with spiking latency

A path to a name

Step 1

Operation dashboard

p95 spike, all platforms affected

Step 2

Tenant overview

one outlier with spiking latency

Step 3

Tenant dashboard

TimeoutException elevated in one field

Simple to navigate. Genuinely hard to build.

The Hard Parts

Alert Ownership

Cardinality

Latency Attribution

Schema-Driven Alerting

600 tenants. Who owns the alerts?

The schema is the control plane.

Field & type alerts

```
@alert(owners: [...], latencyP95: "200ms")
type Listing {
  photos: [Photo]

  @alert(owners: [...], errorRate: "5%")
  reviews: [Review]
}
```

Operation alerts

```
// operation_alerts.json
"StaysSearch": {
  "owners": ["search-oncall"],
  "warningLatencyRate": "500ms",
  "errorRate": "5%"
}
```

Ownership is in the code.

Alerts as code — Viaduct generates the rules

No PromQL. No per-team alert config.

Cardinality

The label space explodes fast.

29K	8K	~600
schema types	operations	tenants

Which fields deserve metrics?

```
type Listing @resolver {
  price: Money @resolver
  photos: [Photo]
  availability: Availability @privacy(check:
"availability")
  status: String @alert(owners: ["listings"])
  rating: Float @deprecated(reason: "use
reviews")
}
```

Behavioral directives

```
type Listing @resolver {
  price: Money @resolver
  photos: [Photo]
  availability: Availability @privacy(check:
"availability")
  status: String @alert(owners: ["listings"])
  rating: Float @deprecated(reason: "use
reviews")
}
```

Let the schema decide what to measure

All fields

count

errors

Behavioral directives

count

errors

distribution

Opt-in

count

errors

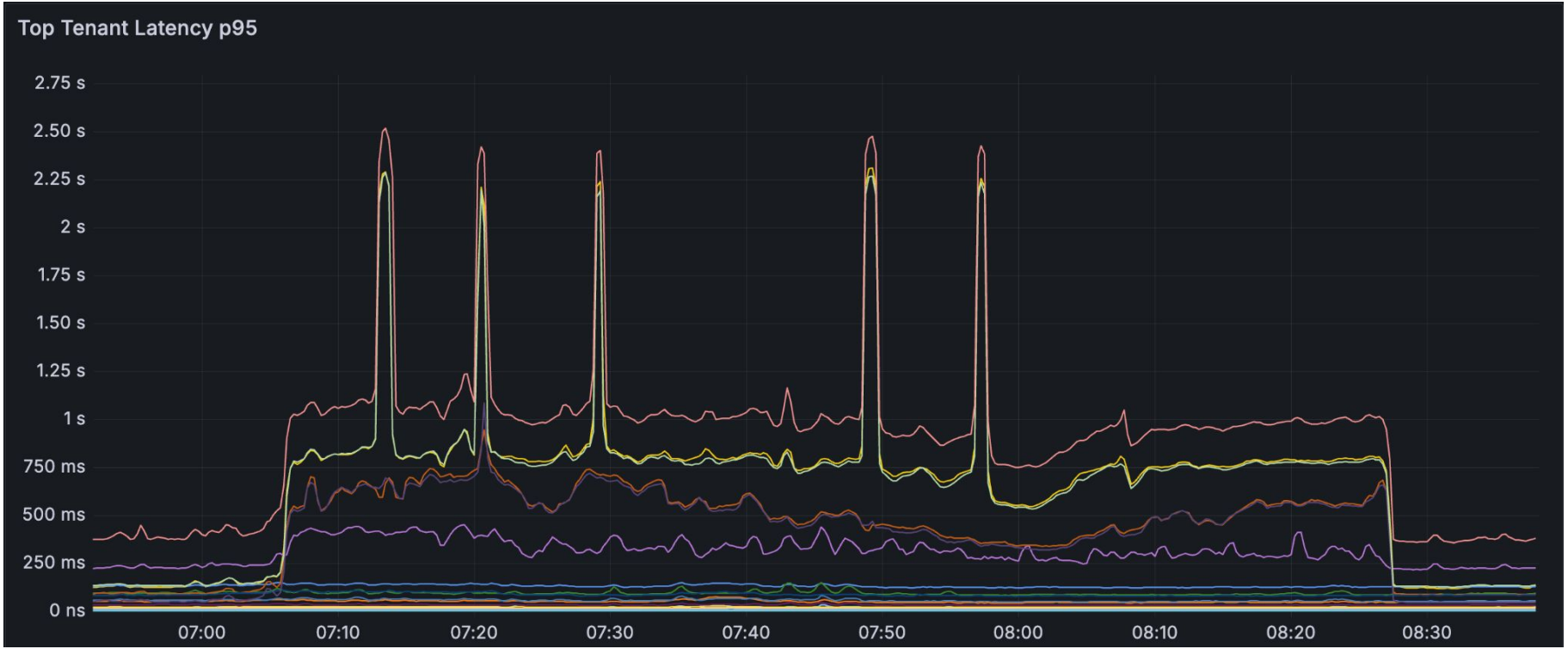
distribution

detailed field metrics

Latency Attribution

Your tenant called another tenant. Their latency is now yours.

Wall clock: who looks guilty?



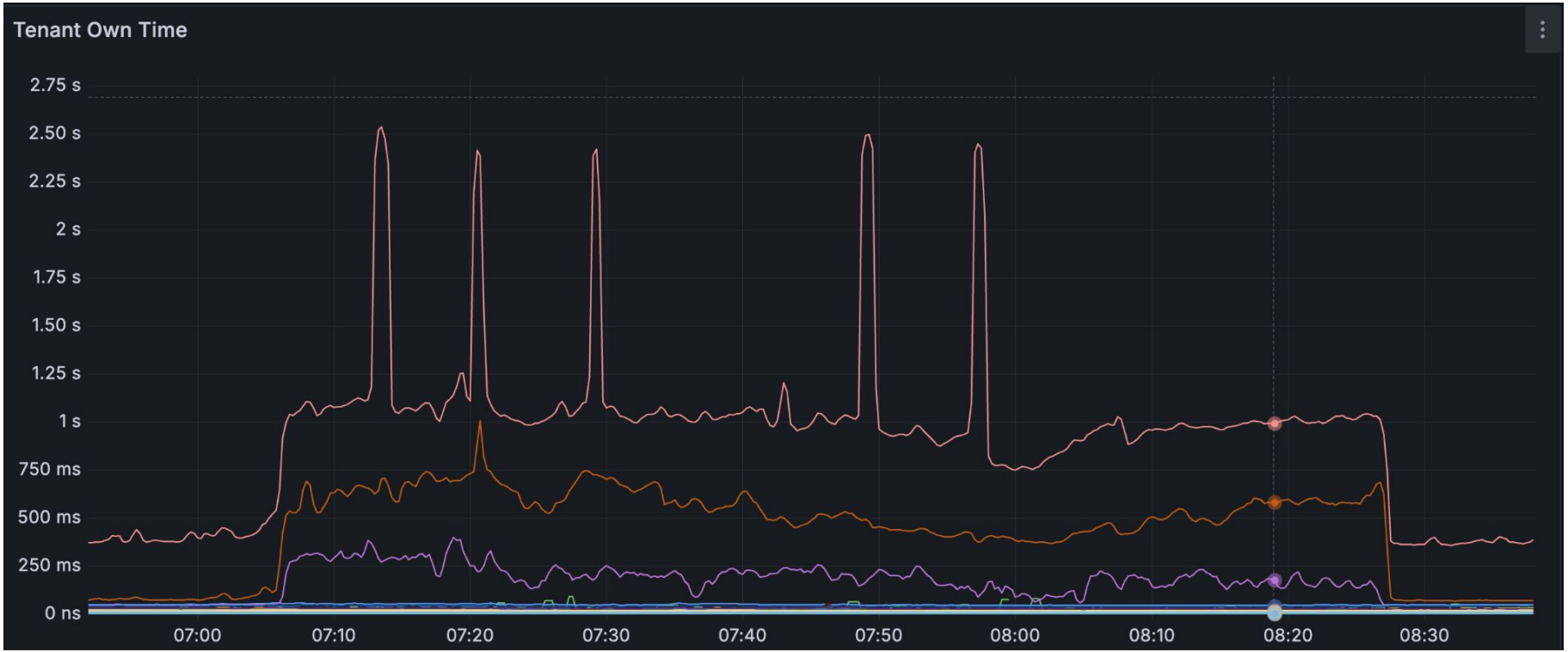
Own time = Wall clock - downstream

Own time = wall clock − downstream

own time = resolver − merged(A, B) — work done, not time spent waiting

Read more: https://medium.com/airbnb-engineering/measuring-latency-overhead-with-own-time-f4373f586ca

Own time: who's actually responsible?



Traces

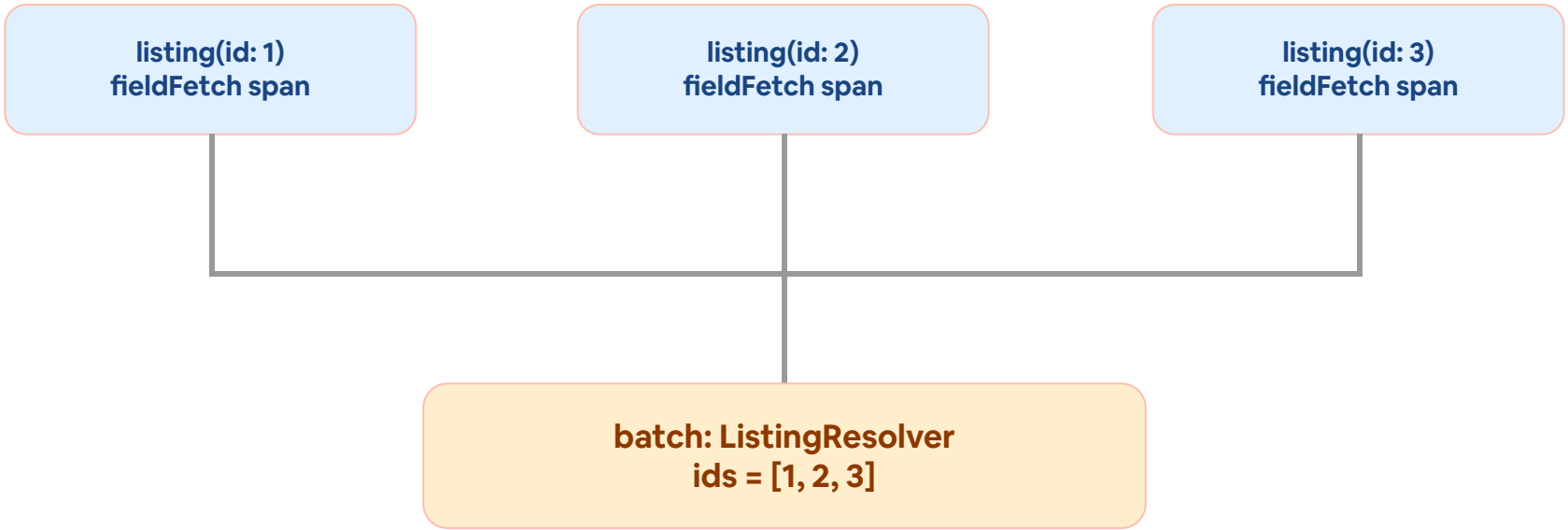
What if metrics don’t tell the whole story?

Distributed tracing

A batch has N causes. A trace has one parent.

The obvious answer — and why it fails

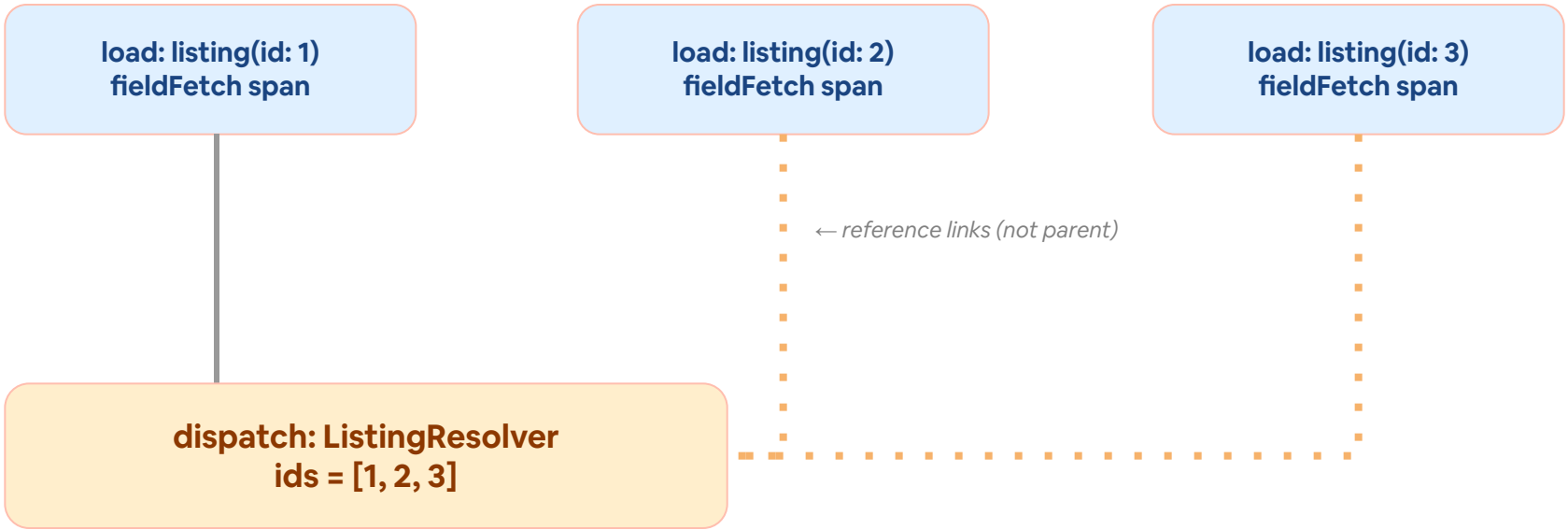
Three fields request listings — DataLoader batches them into one call



Three callers. One batch. Only one span can be the parent — the others disappear from the trace.

Separate the load span from the dispatch span

Give each field its own span — link it to the batch via a reference, not a parent



Three observable load spans, one per field. One dispatch span with one parent — the batch, honestly represented.

Takeaways

The schema is the control plane.

Ownership, alerting, cardinality — one place drives all of it.

Bake attribution into the framework, not the tenant.

If it requires per-team effort, it will be inconsistent.

Questions?