GRAPHQL CONFERENCE 2026

Safely Merging Subgraphs in a Distributed World

How to consolidate federation boundaries with zero customer impact

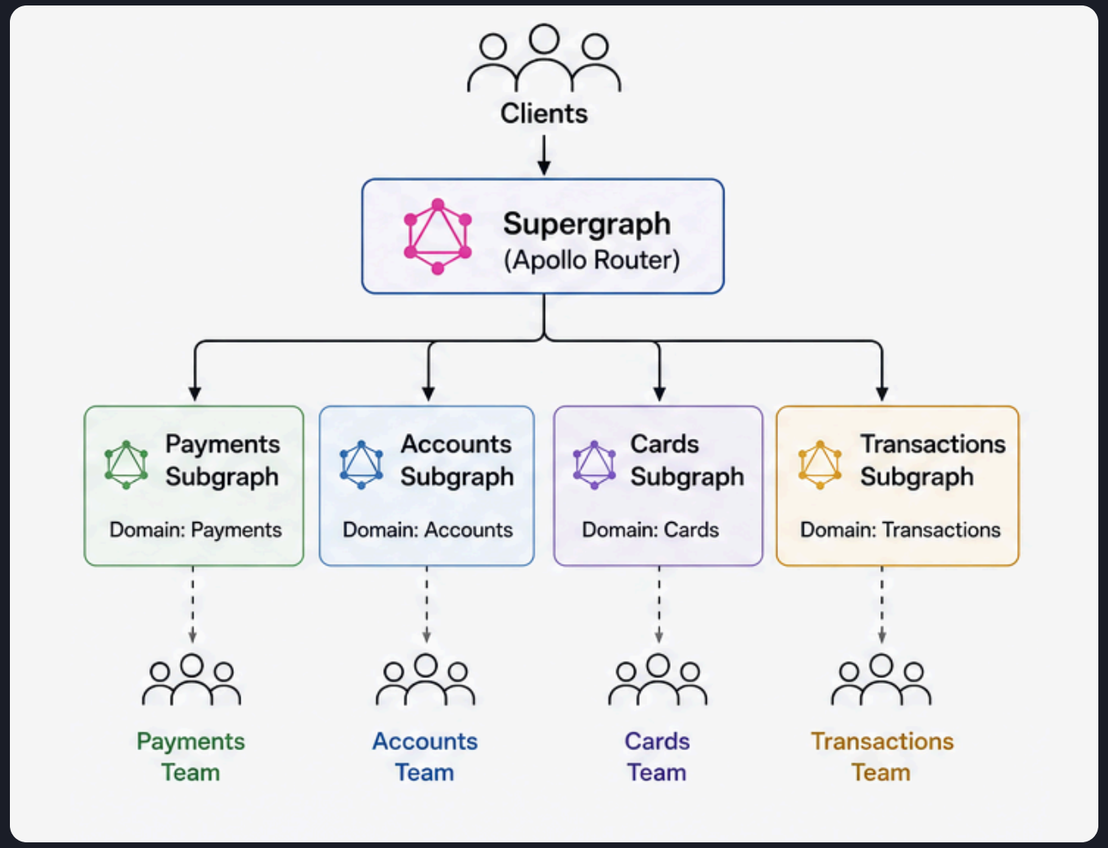
Clarice & Rodrigo
Brex

MOTIVATION

The Cost of Too Many Subgraphs

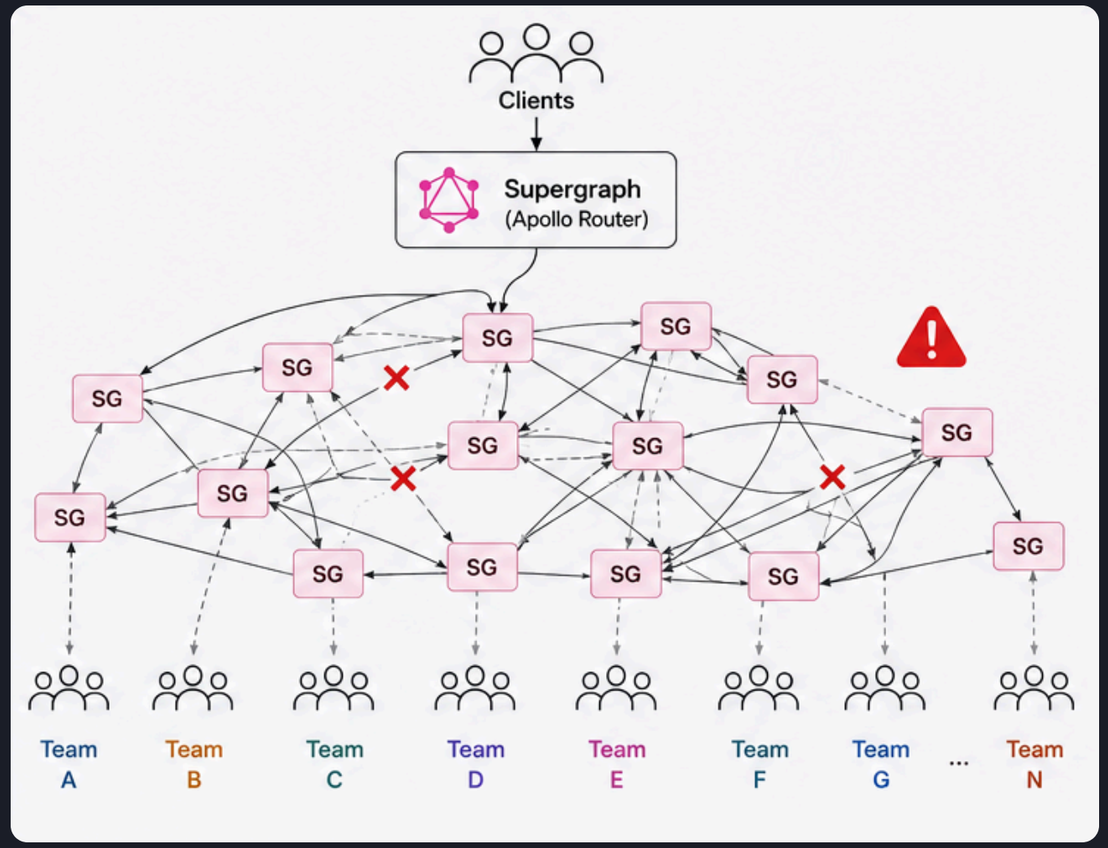
The Federation Promise

Team autonomy and flexibility



The Reality When You Go Too Far

Reliability risk, operational overhead, and degraded graph quality



HOW WE GOT HERE

How Did We Get Here?

Elixir monolith
One big GraphQL API. Worked great—until it became a bottleneck.

Federation + Kotlin
Breaking the monolith. Teams moved faster.

New feature, new subgraph
The default became “always split”.

Hidden costs compound
More boundaries, more context switching, more contractual overhead.

Domain consolidation
Boundaries follow domains—not teams, not features.

Federation is still the right abstraction for us. We just drew the boundaries around features instead of domains.

OUR ARCHITECTURE

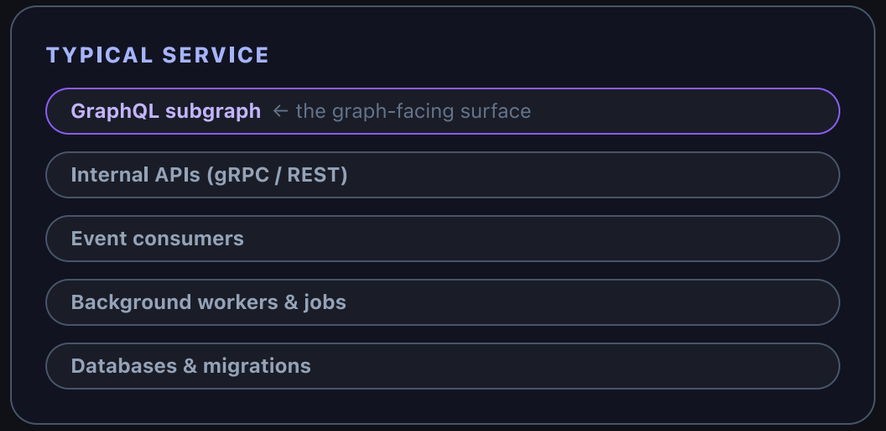
Anatomy of a Subgraph

Schema	SDL files that define the federated contract — types, queries, entities, directives.
Wiring	Connects schema types to resolver functions. The routing table between SDL and code.
Resolvers	Implement field-level logic. Each resolver knows how to fetch or compute its field.
DataLoaders	Batch and cache downstream calls to avoid N+1 problems across a request.
Domain Modules	Business logic that resolvers delegate to. Encapsulate rules, persistence, and integrations.

The subgraphs in this talk are Kotlin services backed by GraphQL Java. This layering is common across most GraphQL server implementations.

OUR ARCHITECTURE

A Subgraph Is Not the Complete Picture



Each service boundary also means

- Its own dashboards & alerts
- Its own on-call surface
- Its own config, secrets, and infrastructure

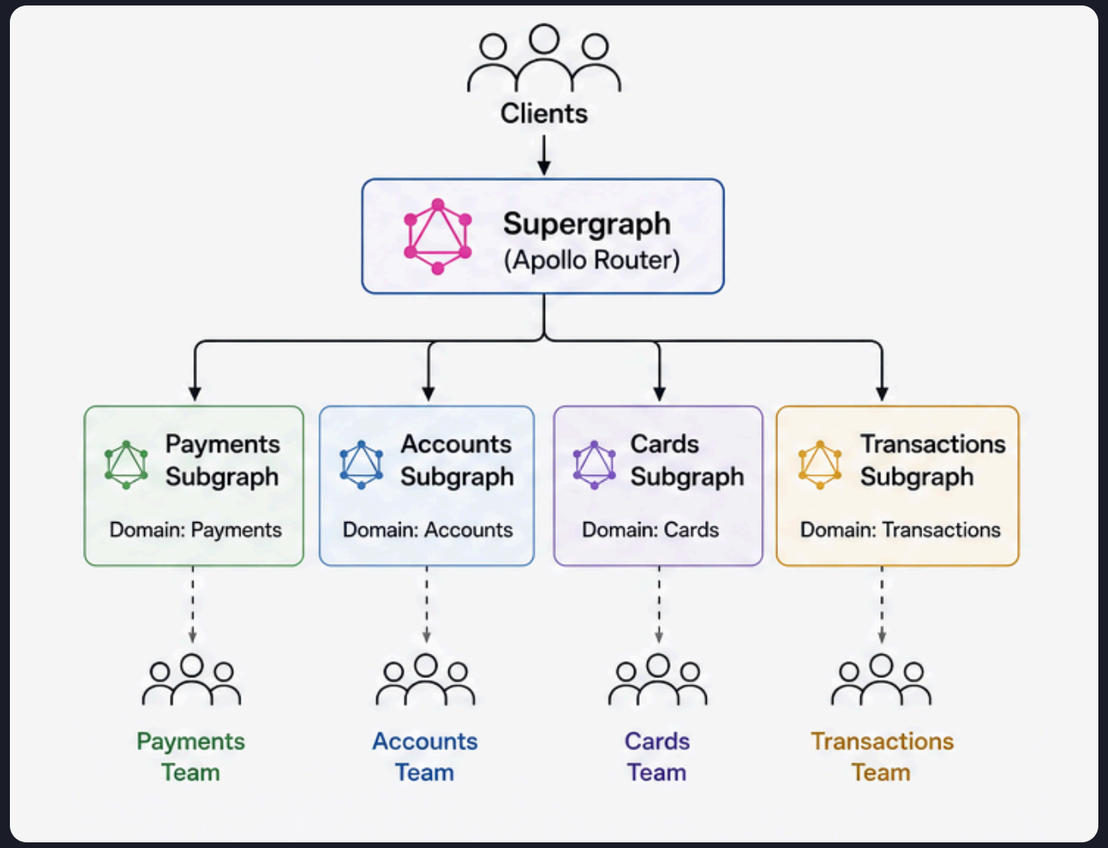
Merging subgraphs is not just moving schema files.
It means consolidating everything a service boundary drags with it.

MOTIVATION

The Cost of Too Many Subgraphs

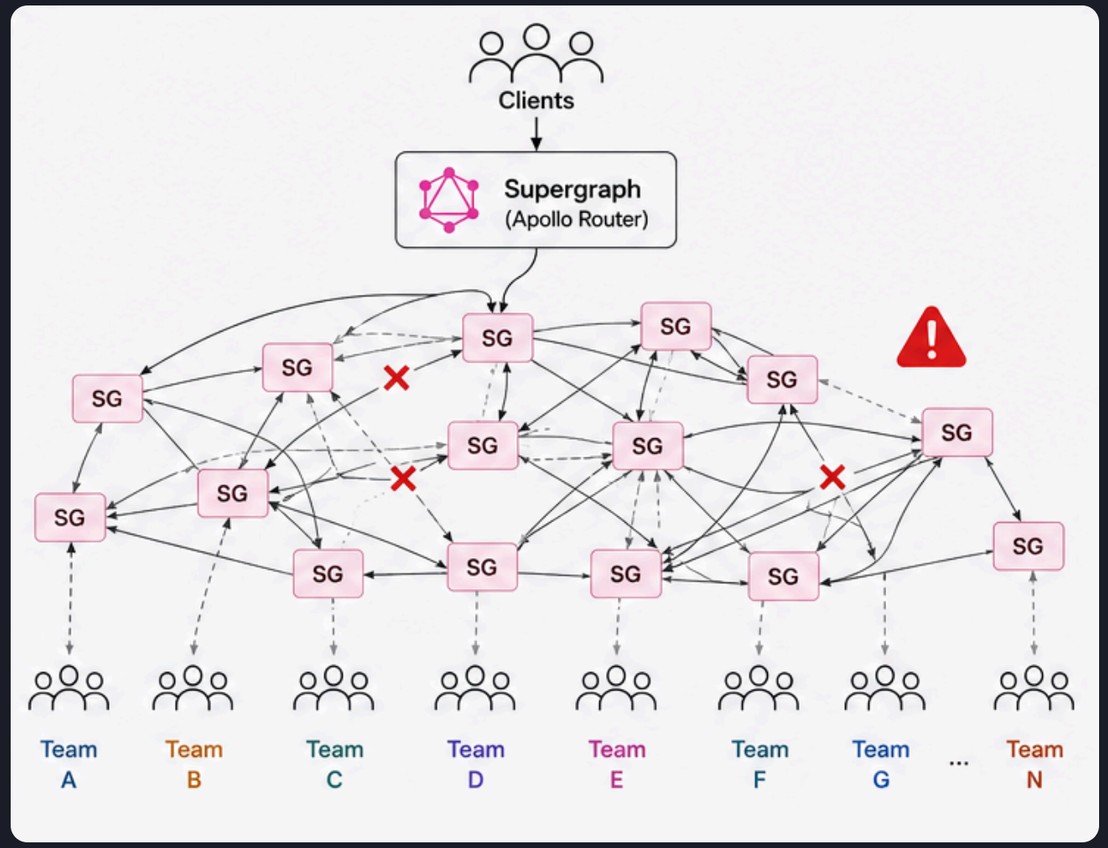
The Federation Promise

Team autonomy and flexibility



The Reality When You Go Too Far

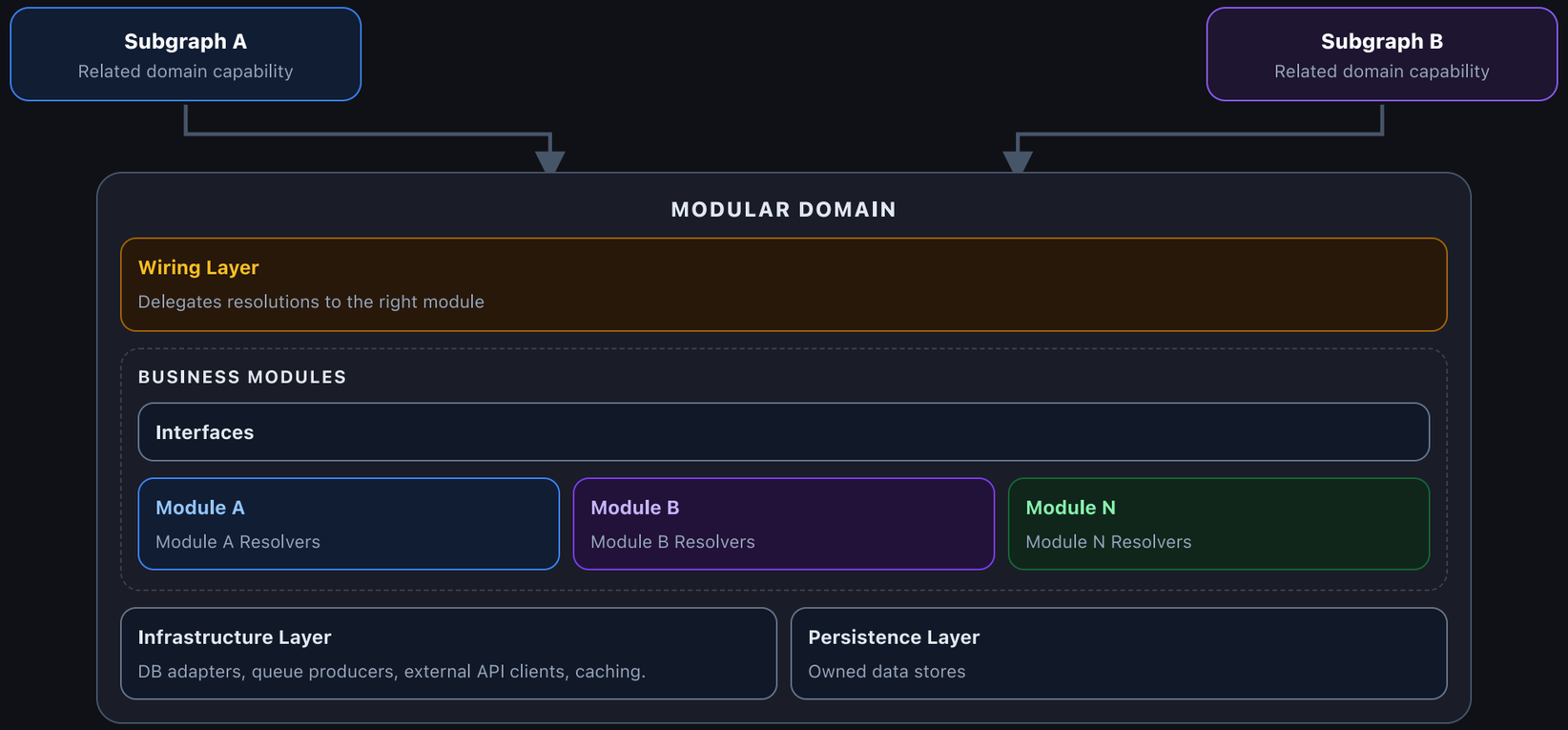
Reliability risk, operational overhead, and degraded graph quality



Which of these boundaries are still earning their cost?

GOAL

Target Architecture



PROCESS OVERVIEW

Timeline

Test
Integration tests prove the contract and ensure there's no client impact

Code Migration
New domain service can execute old operations

Traffic Shift
Traffic moves gradually while router still points to the old service

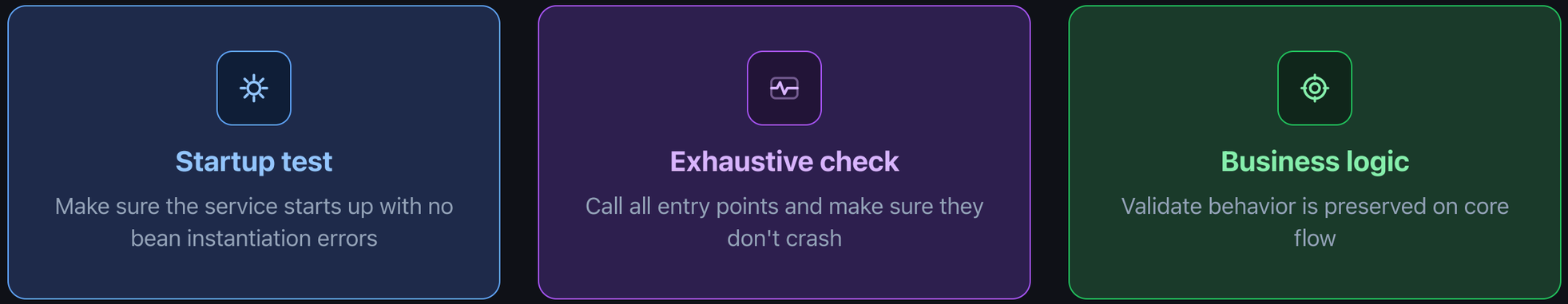
Ownership Handoff
New service publishes the combined schema and takes over ownership

Cleanup
Delete old code, clean up resources and make sure alerts are updated

Contracts Should Stay Stable
Migrations must be invisible to clients — zero downtime, zero breaking changes

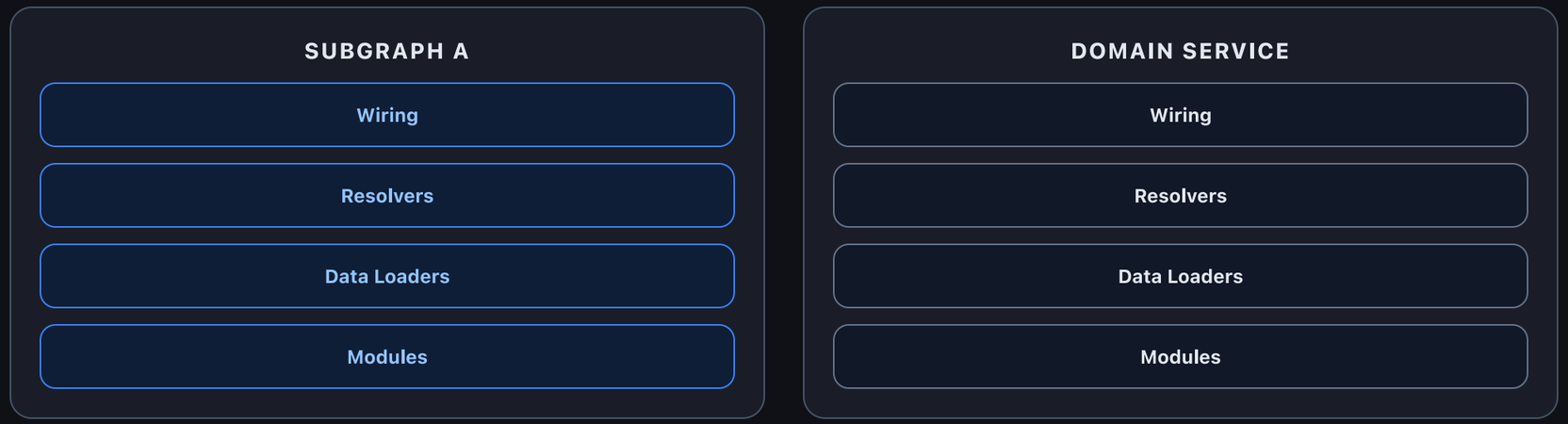
GUARDRAILS

Tests



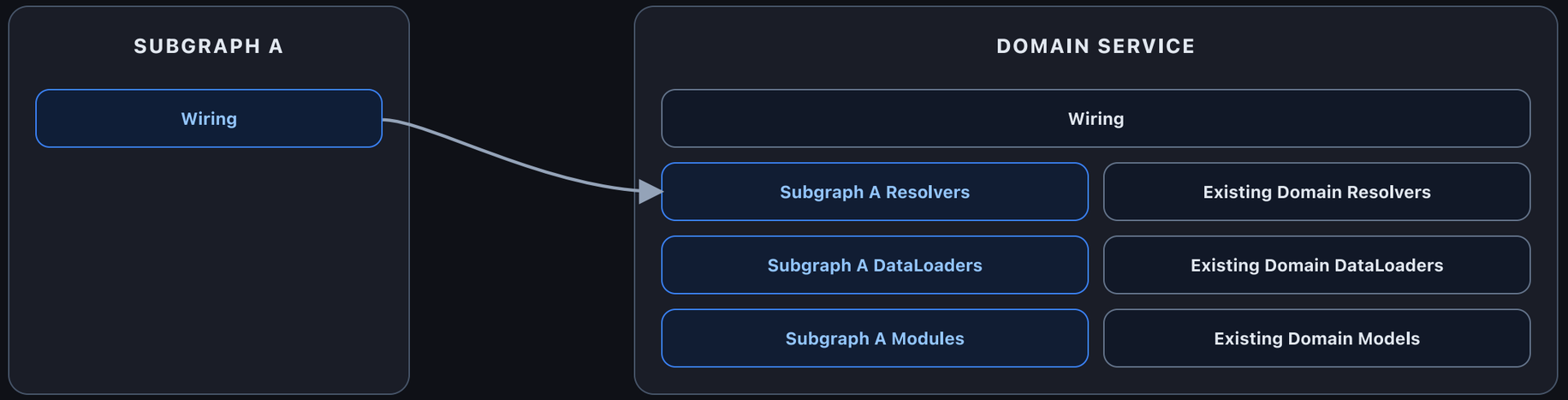
CODE MIGRATION

Before



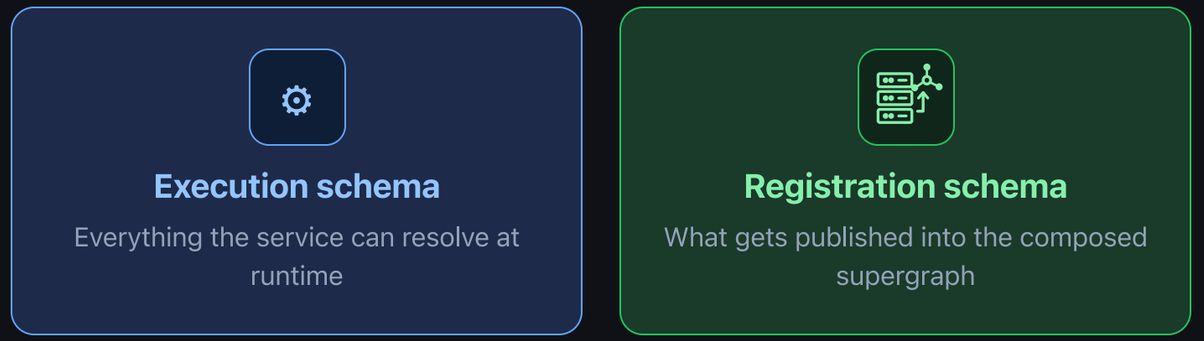
CODE MIGRATION

Intermediate Step



CODE MIGRATION

Schema Separation



CODE MIGRATION

Example

SUBGRAPH A

```
# execution & registration schema
extend type Query {
  fieldA(id: ID!): TypeA
}

type TypeA @key(fields: "id") {
  id: ID!
  status: TypeAStatus!
}
```

DOMAIN SERVICE

```
# execution schema
extend type Query {
  fieldA(id: ID!): TypeA
  domainObject(id: ID!): DomainObject
}

type TypeA @key(fields: "id") {
  id: ID!
  status: TypeAStatus!
}

type DomainObject @key(fields: "id") {
  id: ID!
}
```

```
# registration schema
extend type Query {
  domainObject(id: ID!): DomainObject
}

type DomainObject @key(fields: "id") {
  id: ID!
}
```

The Router Supergraph is unchanged

The router's ownership map stays unchanged. The domain service is already able to execute both paths.

Why not just use @override?

Overrides work well for migrating a few fields. They are not the right choice when you are migrating entire services with hundreds of fields, types, and enums.

TRAFFIC CUTOVER

How the Router Routes

What the Router sees

A subgraph name that maps to a hostname. It does not know or care which pod is behind that hostname.

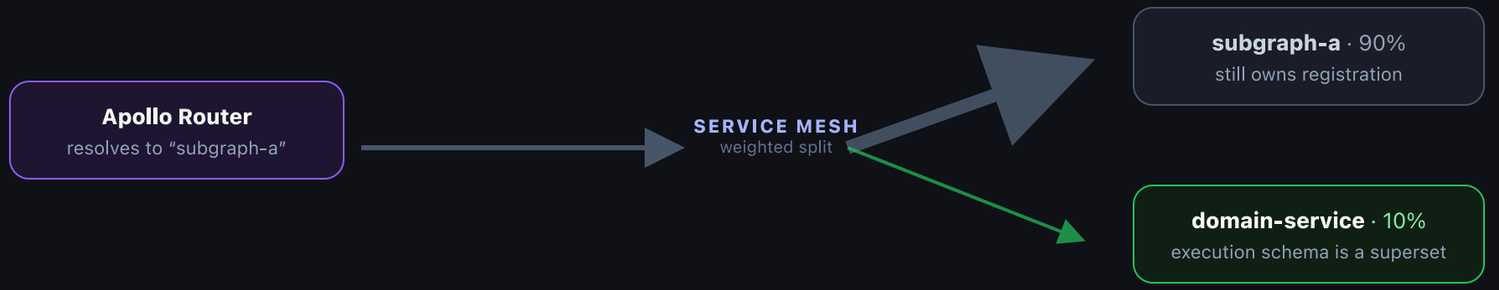
What this means for us

We can redirect traffic at the mesh layer — from subgraph-a's hostname to domain-service — **without the Router or clients knowing.**

This works because domain-service's execution schema is a superset. It can already answer requests the Router thinks belong to subgraph-a.

TRAFFIC CUTOVER

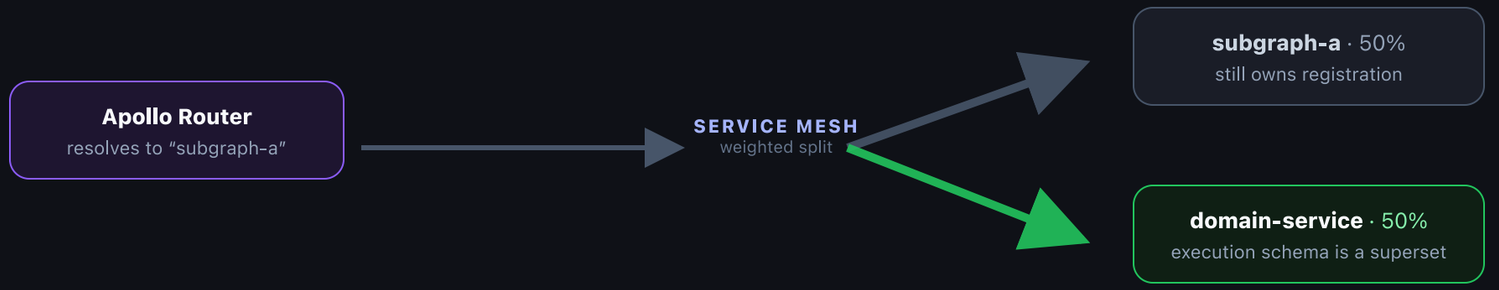
Move the Traffic. Leave the Schema Ownership Alone.




```
# service mesh virtual service
route:
  - destination:
      host: subgraph-a
    weight: 90
  - destination:
      host: domain-service
    weight: 10
```

TRAFFIC CUTOVER

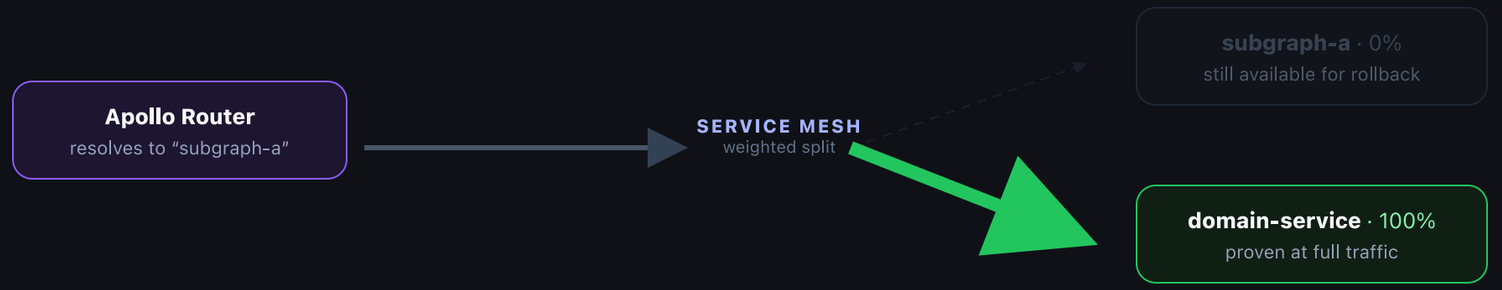
Move the Traffic. Leave the Schema Ownership Alone.




```
# service mesh virtual service
route:
  - destination:
      host: subgraph-a
    weight: 50
  - destination:
      host: domain-service
    weight: 50
```

TRAFFIC CUTOVER

Move the Traffic. Leave the Schema Ownership Alone.




```
# service mesh virtual service
route:
  - destination:
      host: subgraph-a
    weight: 0
  - destination:
      host: domain-service
    weight: 100
```

MERGING THE SUBGRAPHS

Ownership Handoff

1. **subgraph-a registers empty schema**
   Federation directives only, no owned types.
2. **domain-service registers full combined schema**
   Ownership now matches real execution capability.
3. **Publish only when final composition succeeds**
   On failure, supergraph A stays active.

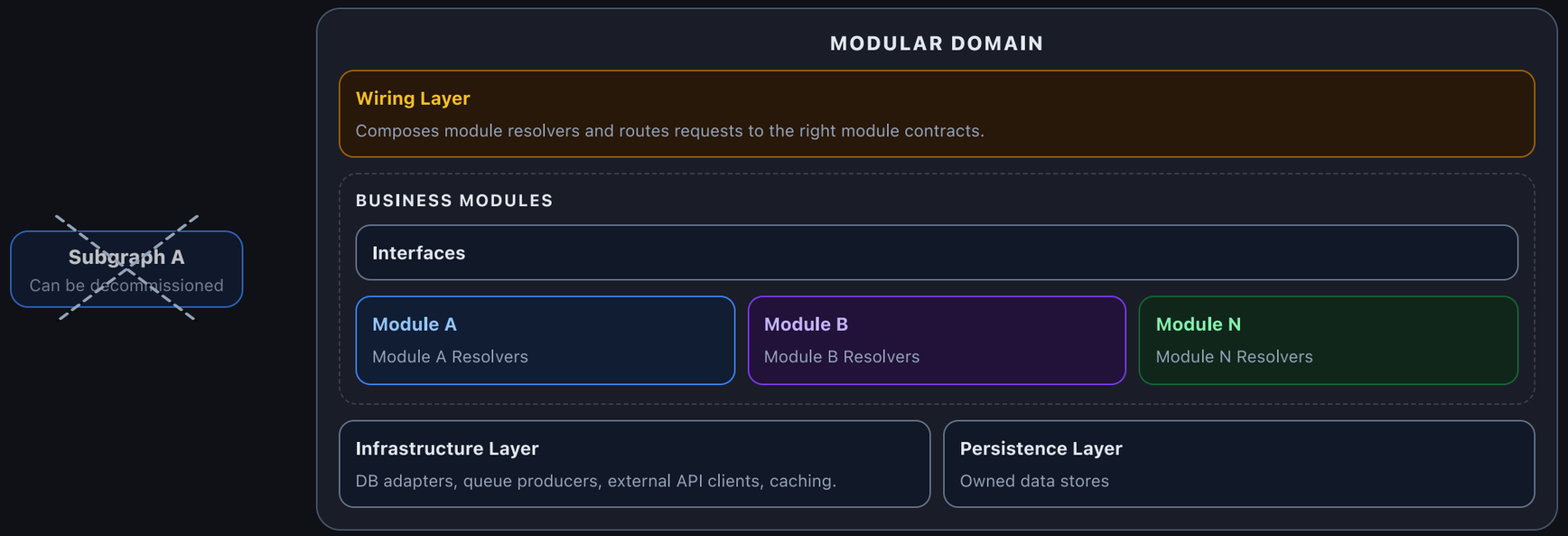
```
# subgraph-a.graphqls (registration)
extend schema
 @link(url: "https://specs.apollo.dev/federation/v2.0",
   import: ["@key", "@shareable"])
# no types
# no queries
# no mutations
# no subscriptions
```

```
# domain-subgraph.graphqls (registration)
extend type Query {
  fieldA(id: ID!): TypeA
  domainObject(id: ID!): DomainObject
}

type TypeA @key(fields: "id") {
  id: ID!
  status: TypeAStatus!
}

type DomainObject @key(fields: "id") {
  id: ID!
}
```

Atomicity Is Required
Removing ownership before reassignment would break composition and temporarily hide types.

OUTCOME

Final State



Thank you!

Questions?

Clarice Abreu
linkedin.com/in/clarice-abreu

Rodrigo Jesus
linkedin.com/in/rodrigoj42