GraphQLConf 2026

hosted by  GraphQL Foundation

GraphQLConf 2026

hosted by GraphQL Foundation

Instagram Server Migration via Federated GraphQL

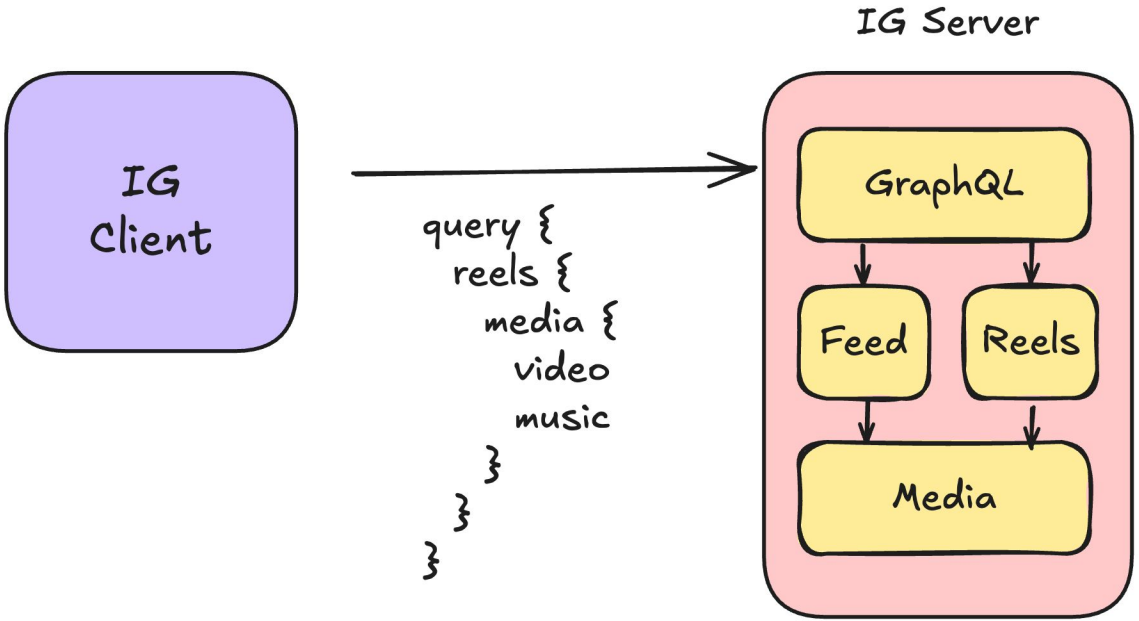
Anirudh Padmarao, Chi Chan, Johnson Han, Deepak Singh, Kristina Kamendova

GraphQLConf
2026

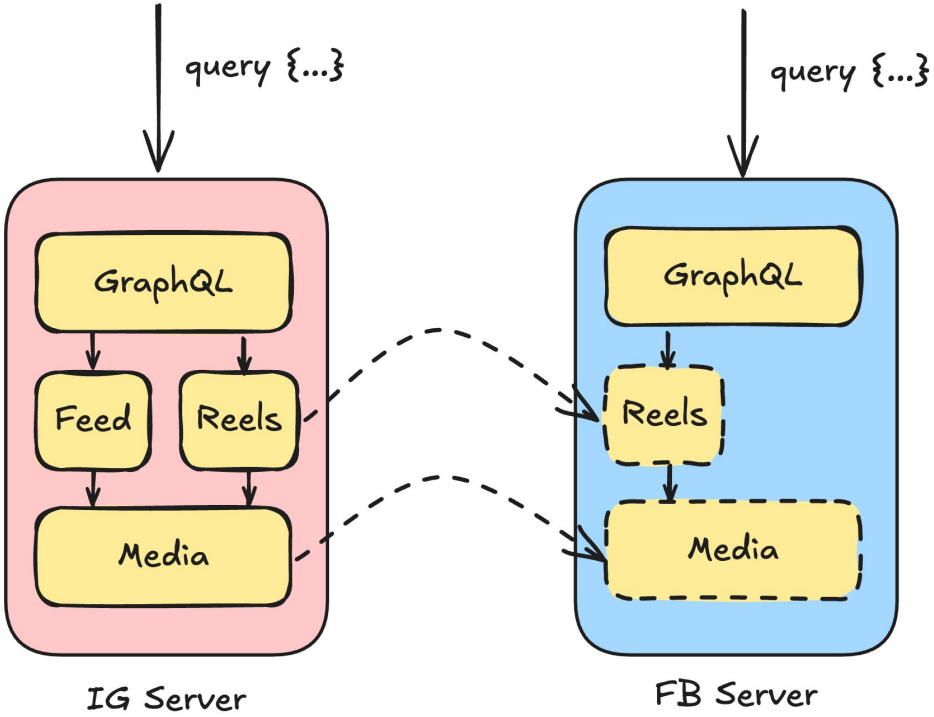
hosted by GraphQL Foundation

Architecture

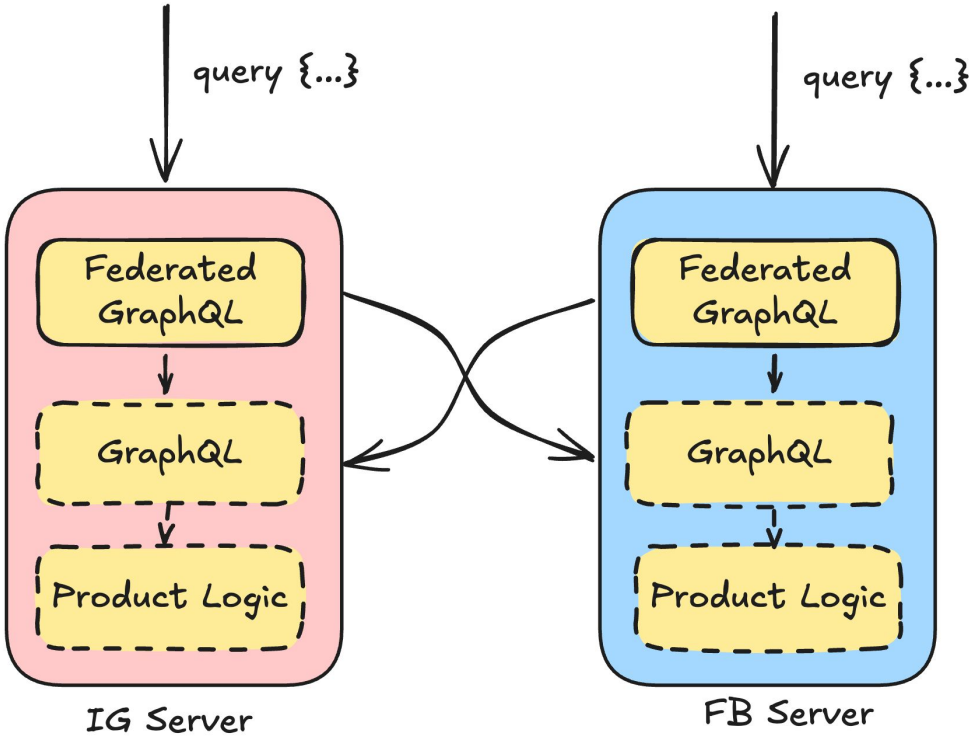
Instagram Architecture



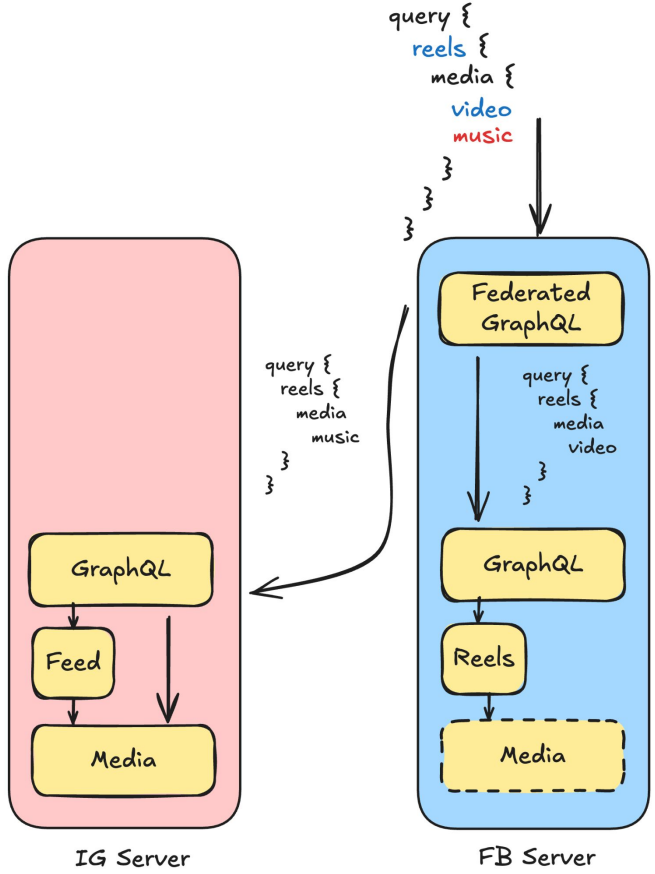
Meta Architecture



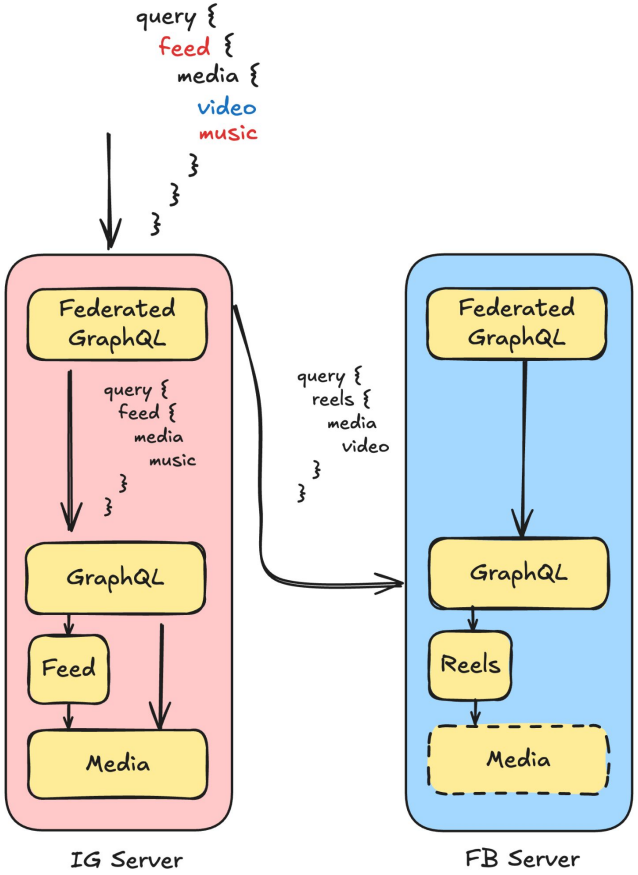
Federation Architecture



Federation Architecture: FB Server



Federation Architecture: IG Server

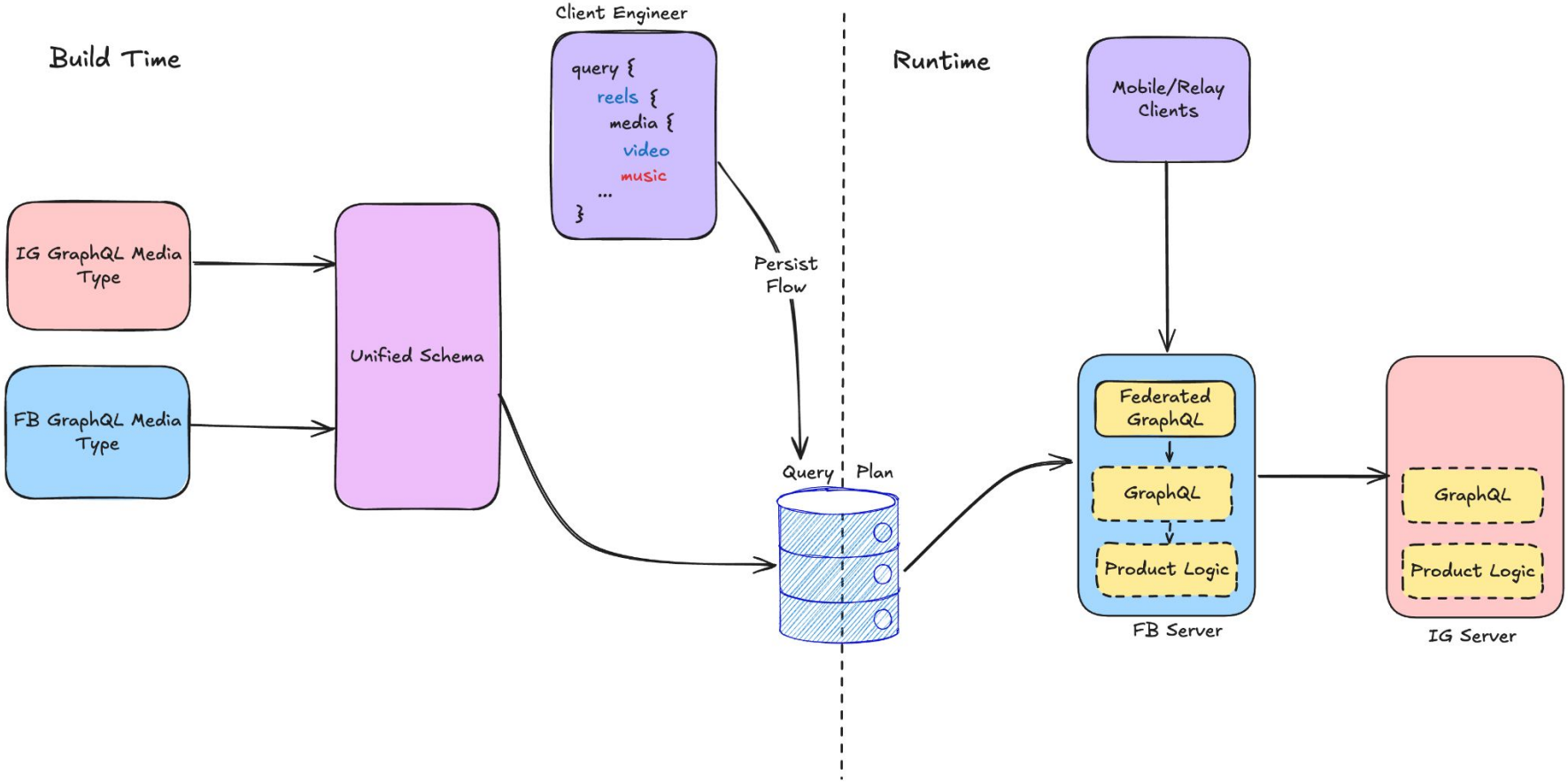


GraphQLConf
2026

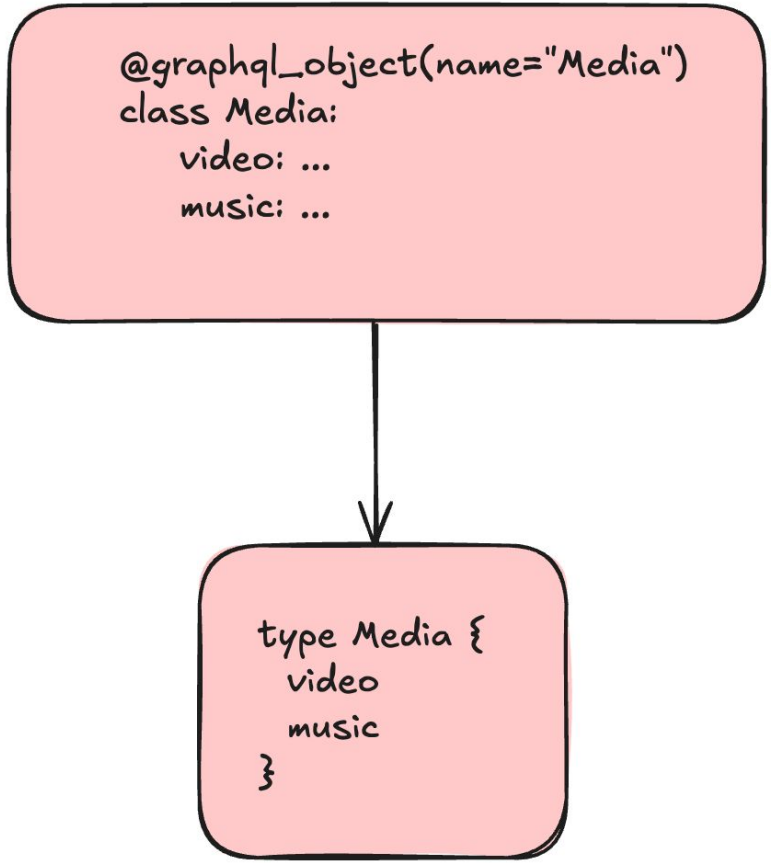
hosted by GraphQL Foundation

Federation

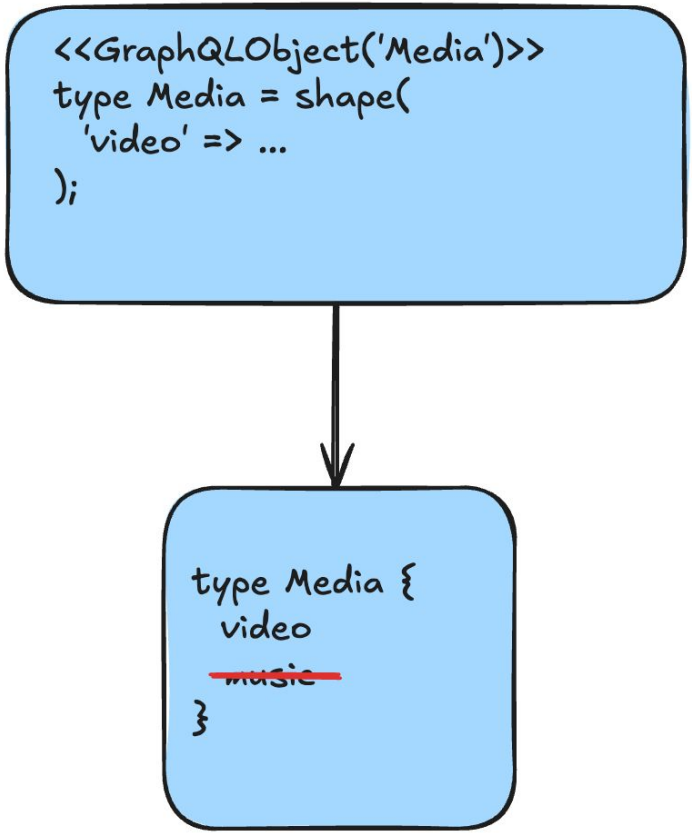
Federation Overview



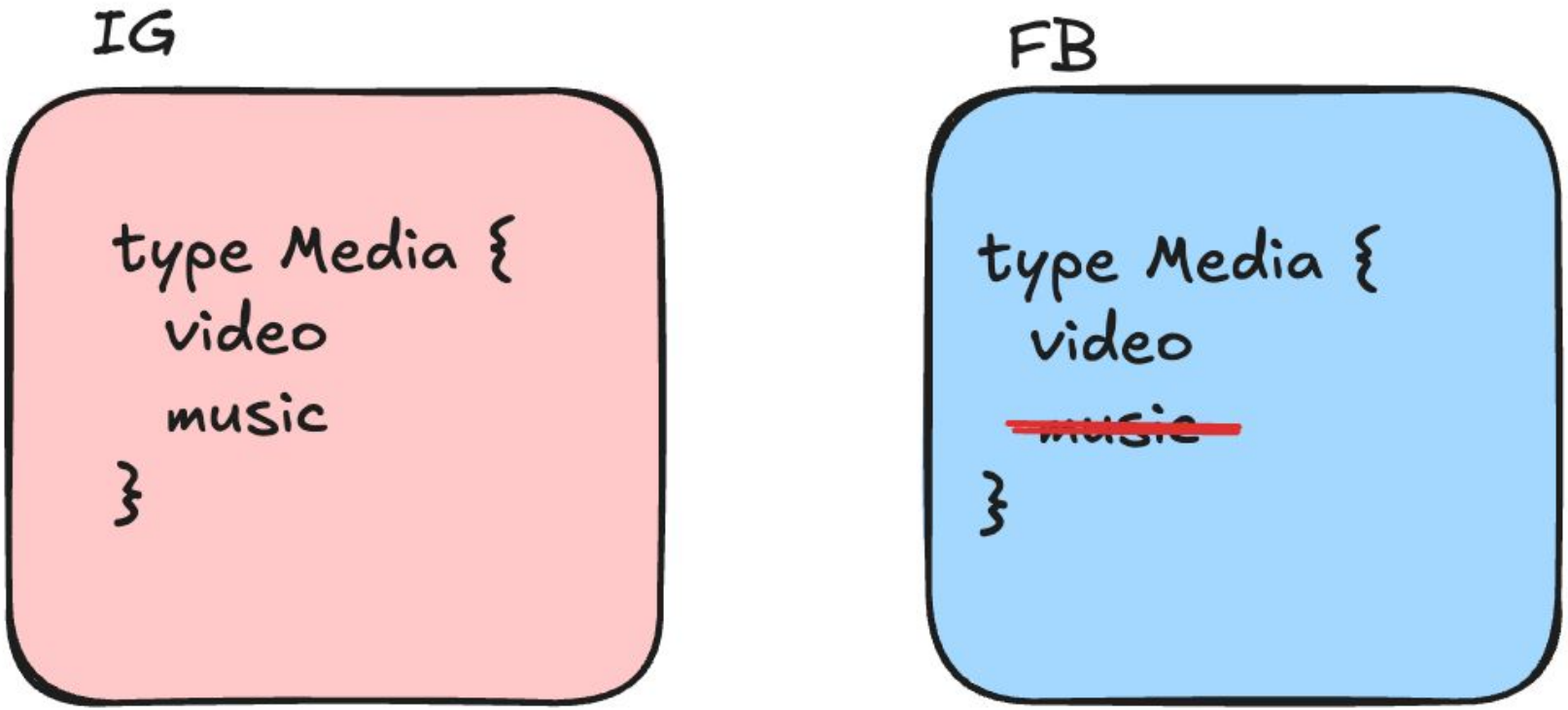
IG Media Server Definition



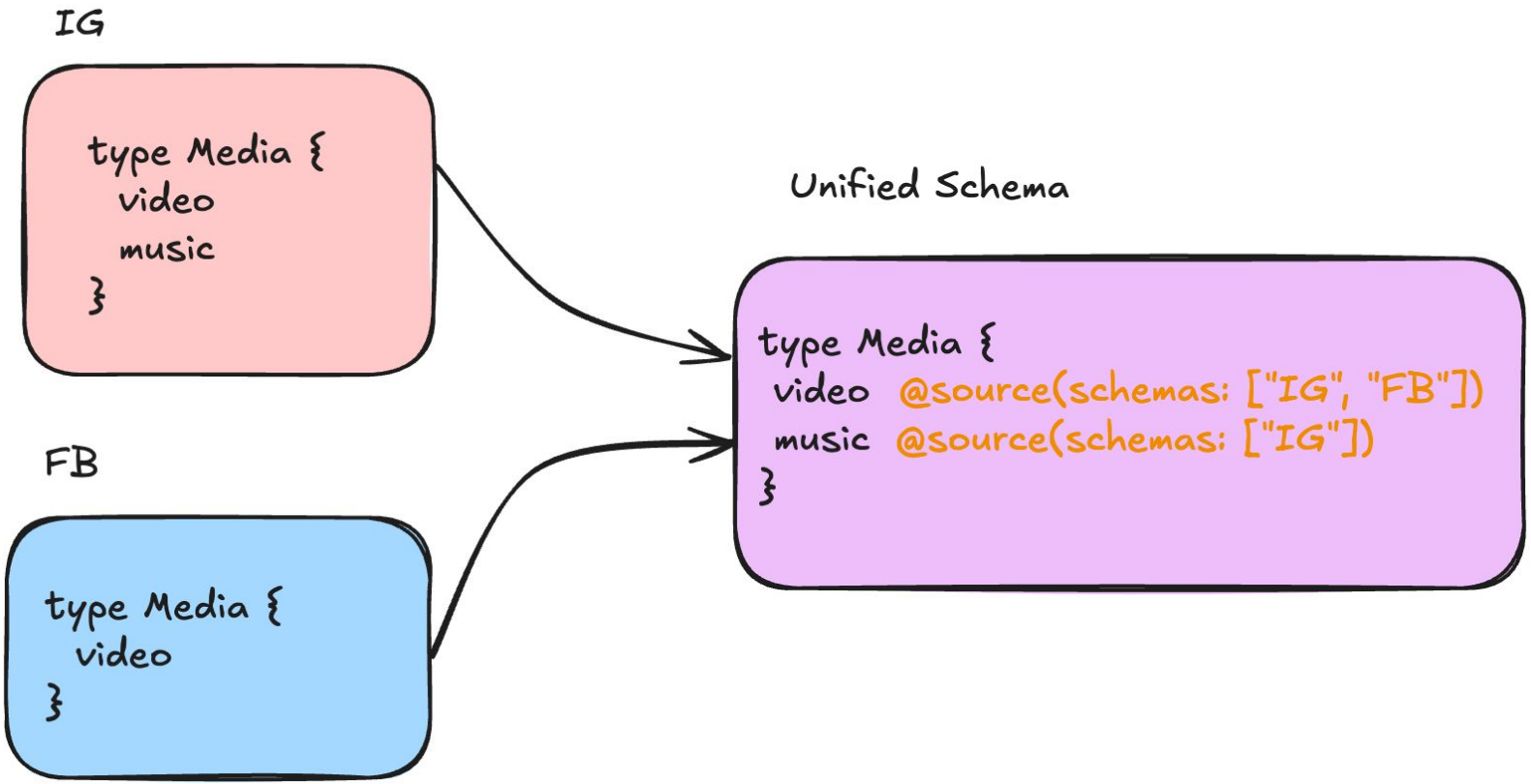
FB Server Media Definition



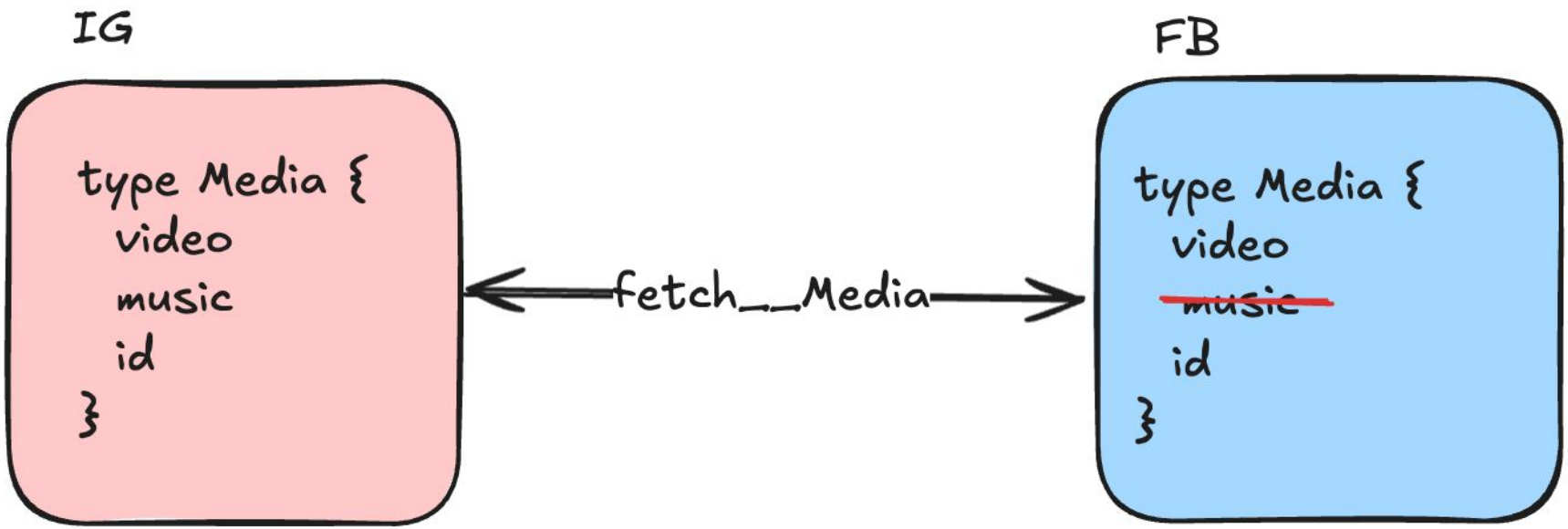
Two Medias, No Connection



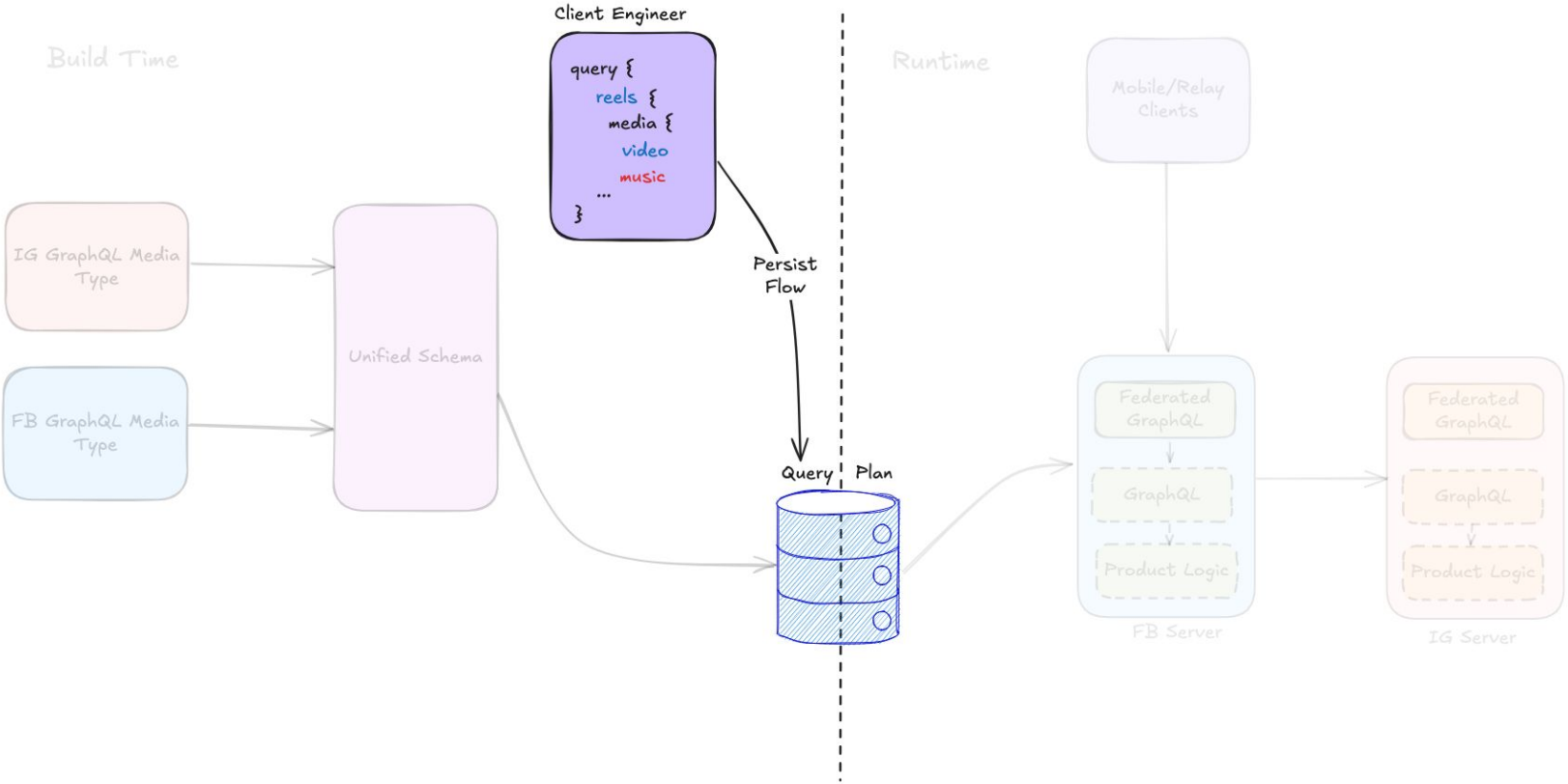
Unified SDL


IG
type Media {
video
music
}
FB
type Media {
video
}
Unified Schema
type Media {
video @source(schemas: ["IG", "FB"])
music @source(schemas: ["IG"])
}

One Media, Two Servers



Federated Query Persistence

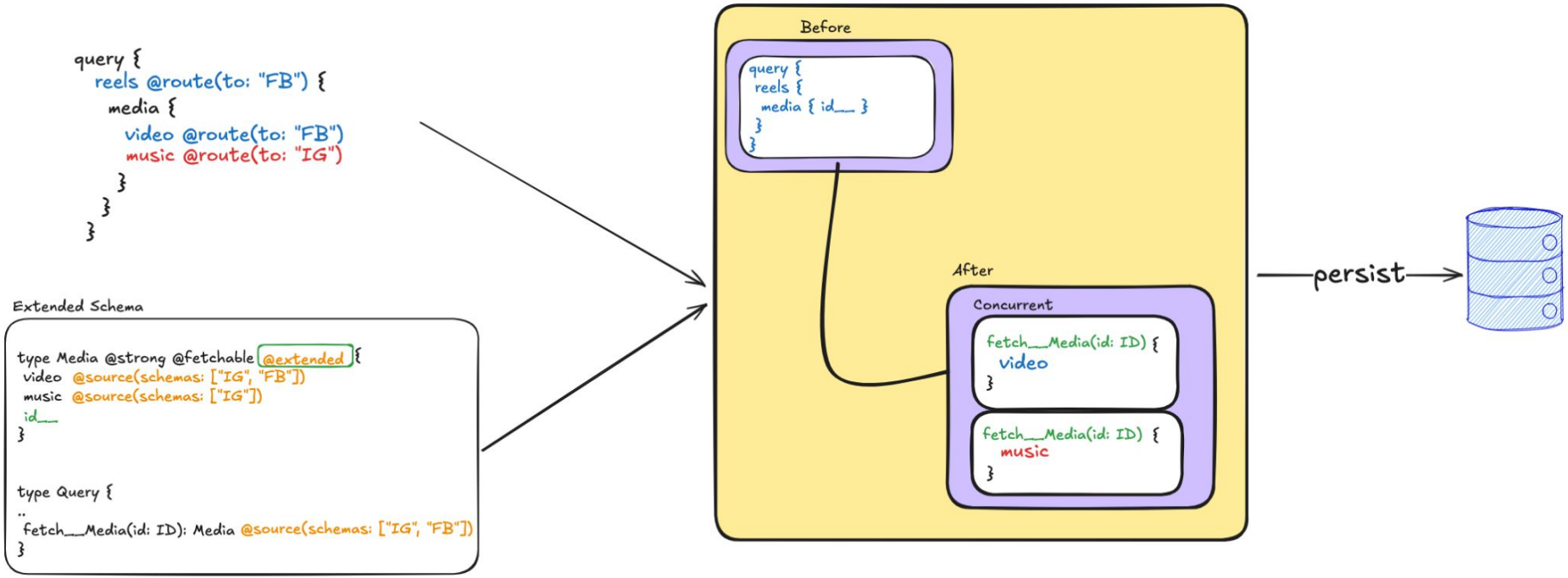


Federated Query Persistence: Routing

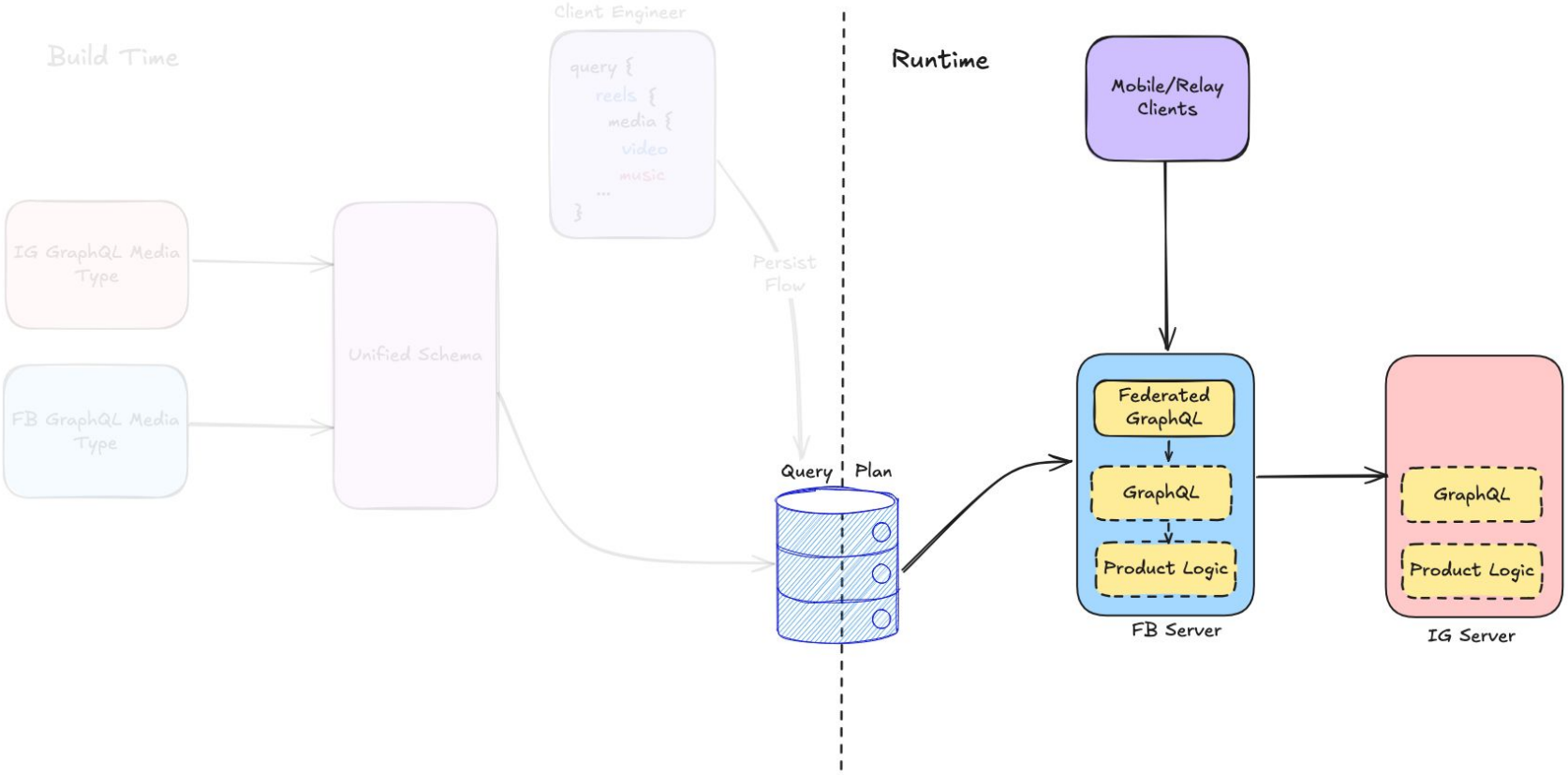
```
query {
  reels {
    media {
      video
      music
    }
  }
}
```

```
query {
  reels @route(to: "FB") {
    media {
      video @route(to: "FB")
      music @route(to: "IG")
    }
  }
}
```

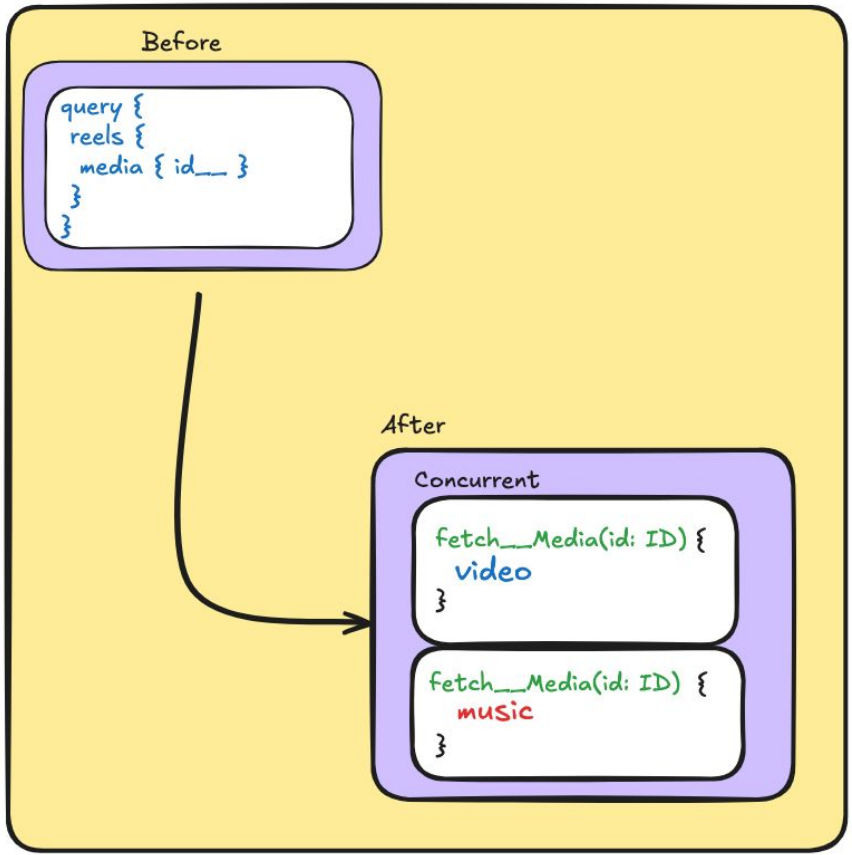
Federated Query Persistence: Building

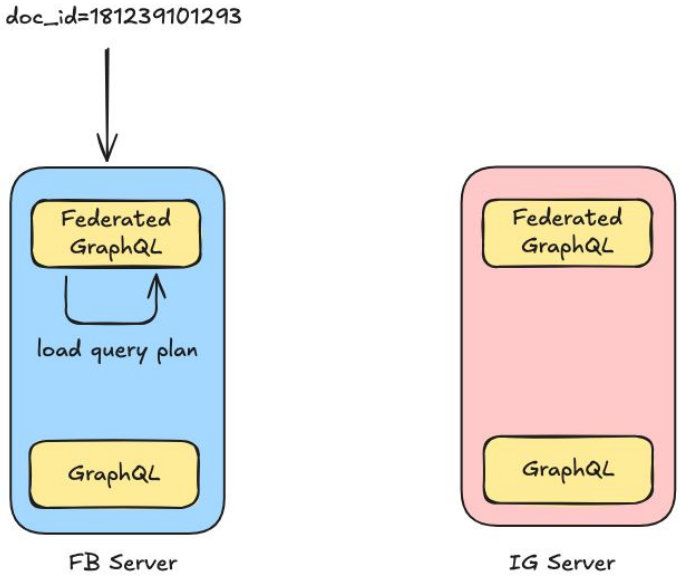


Federated Query Execution

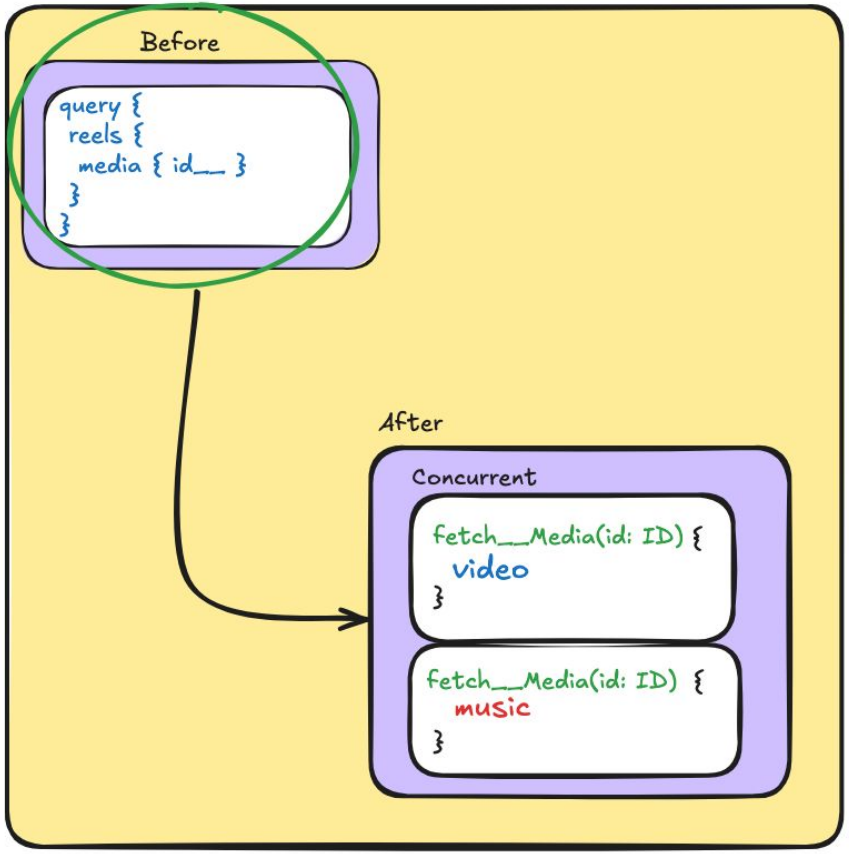


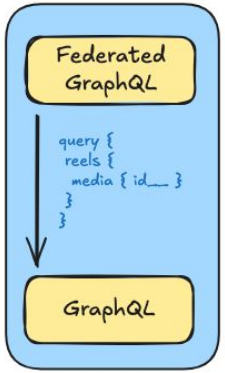
Federated Query Execution

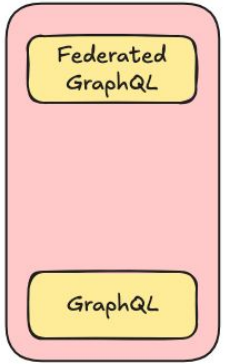




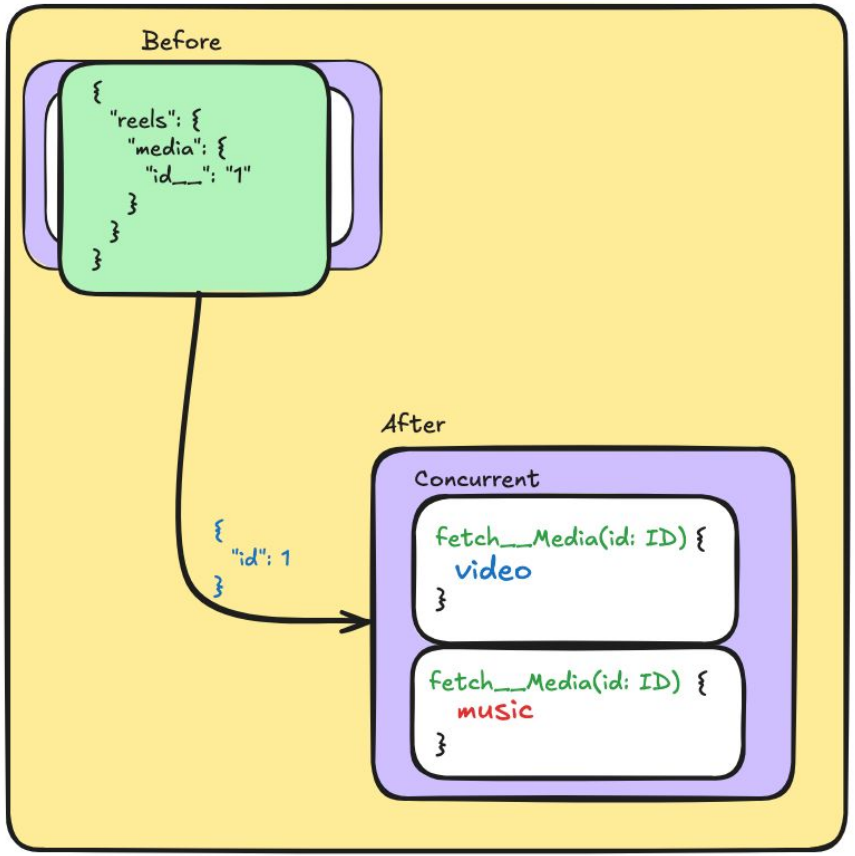
Federated Query Execution: Before

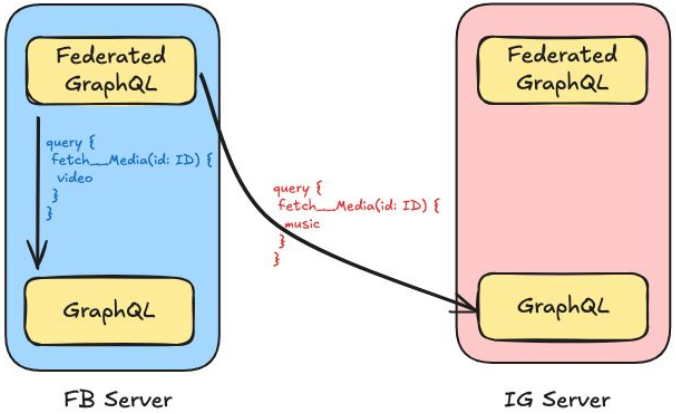




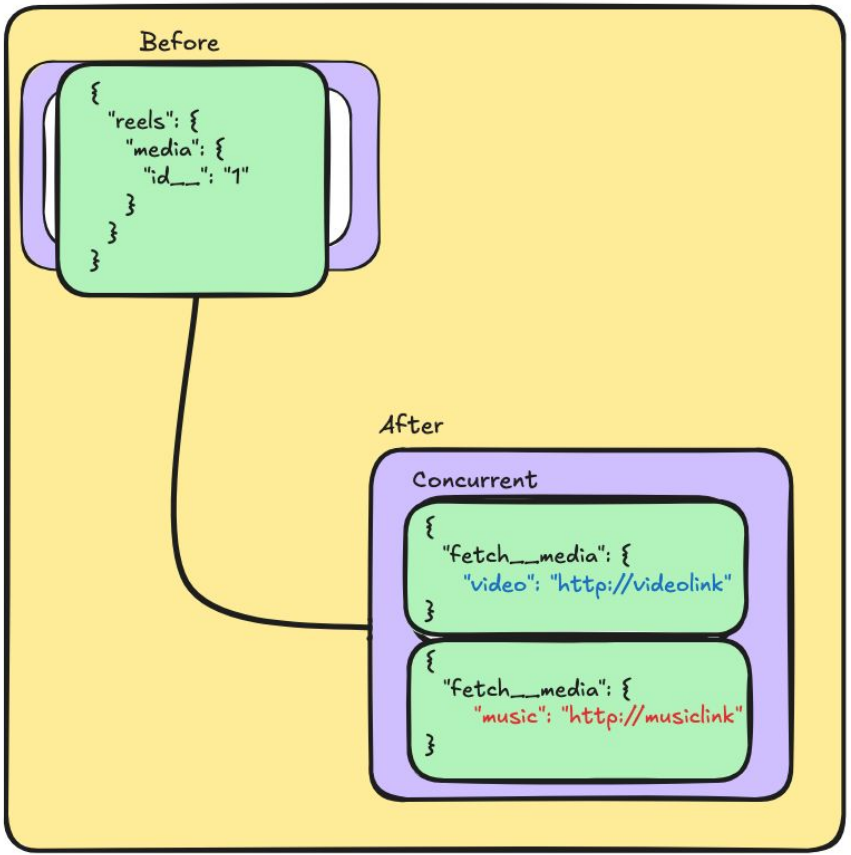


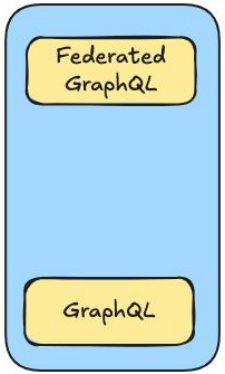
Federated Query Execution: After

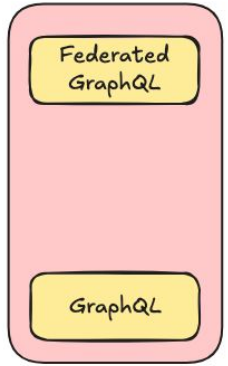




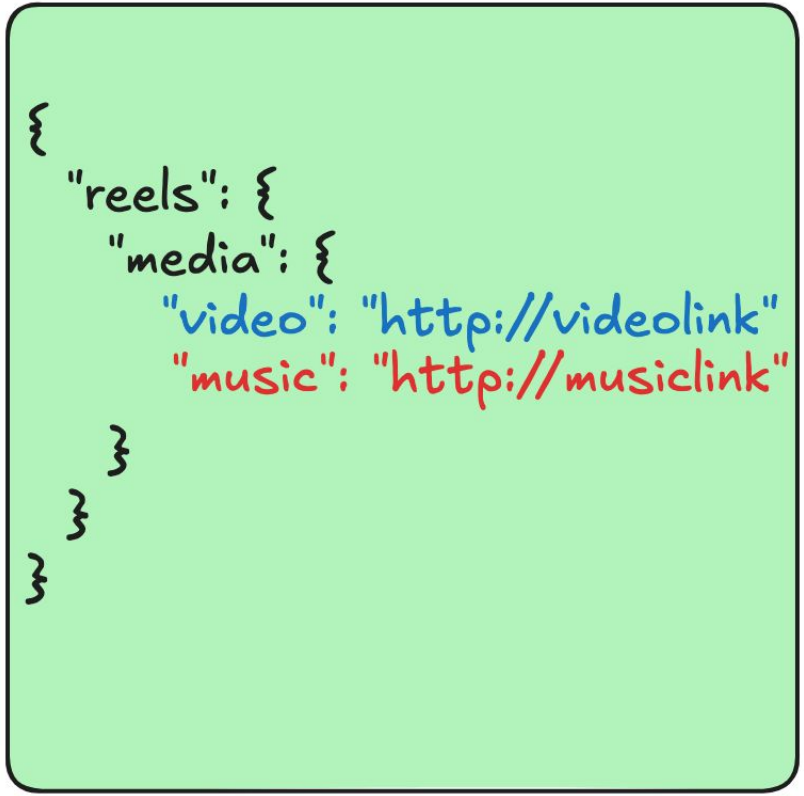
Federated Query Execution: Merging







Federated Query Execution: Response

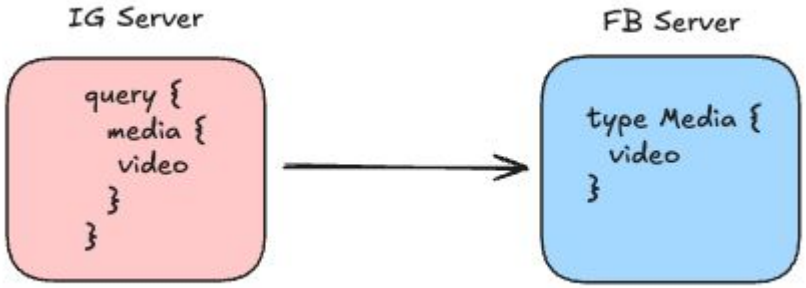


GraphQLConf
2026

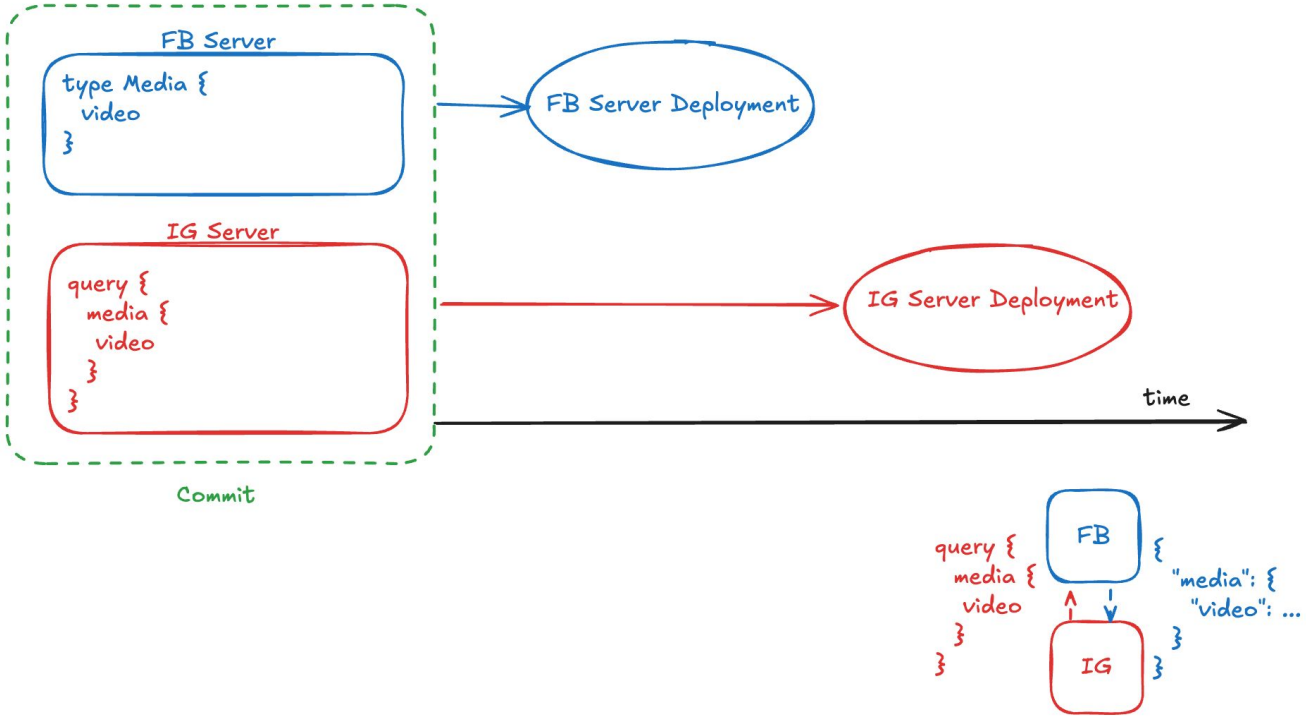
hosted by GraphQL Foundation

Push Safety & Performance

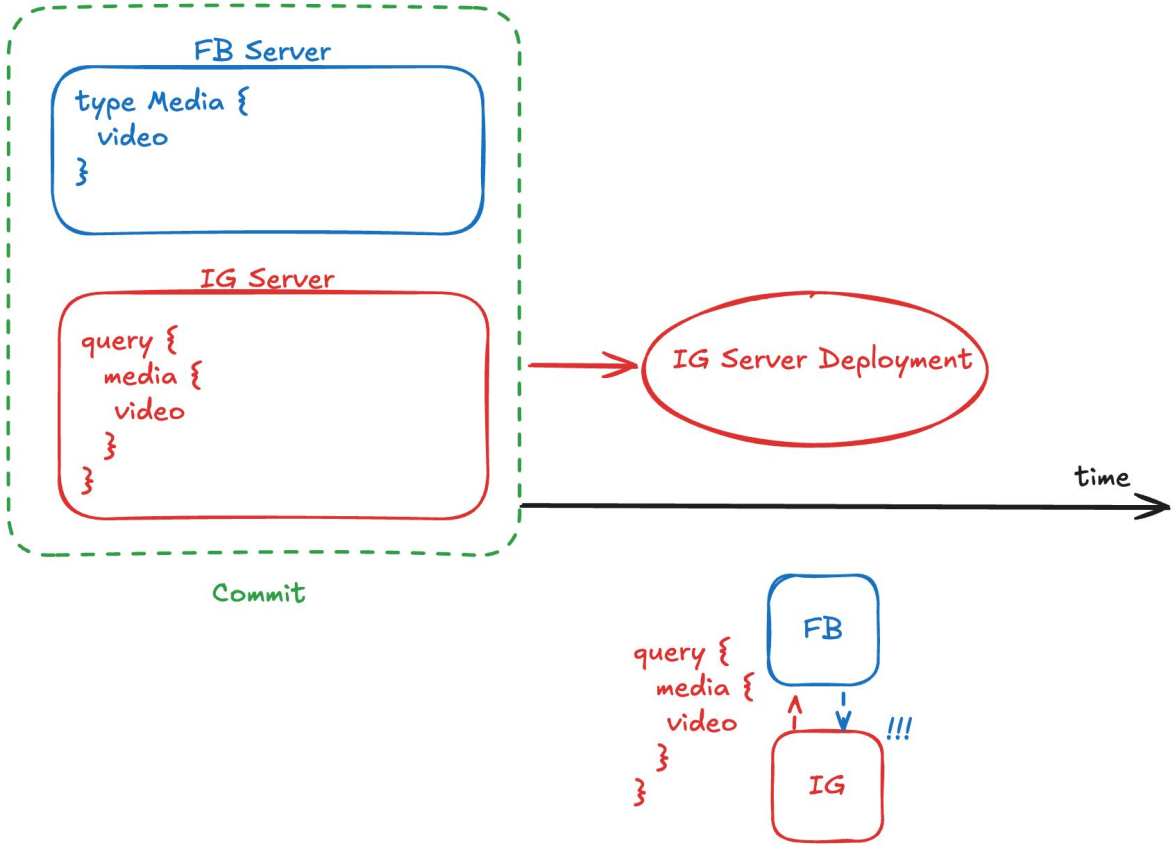
Cross Client-Server Change & Push Safety



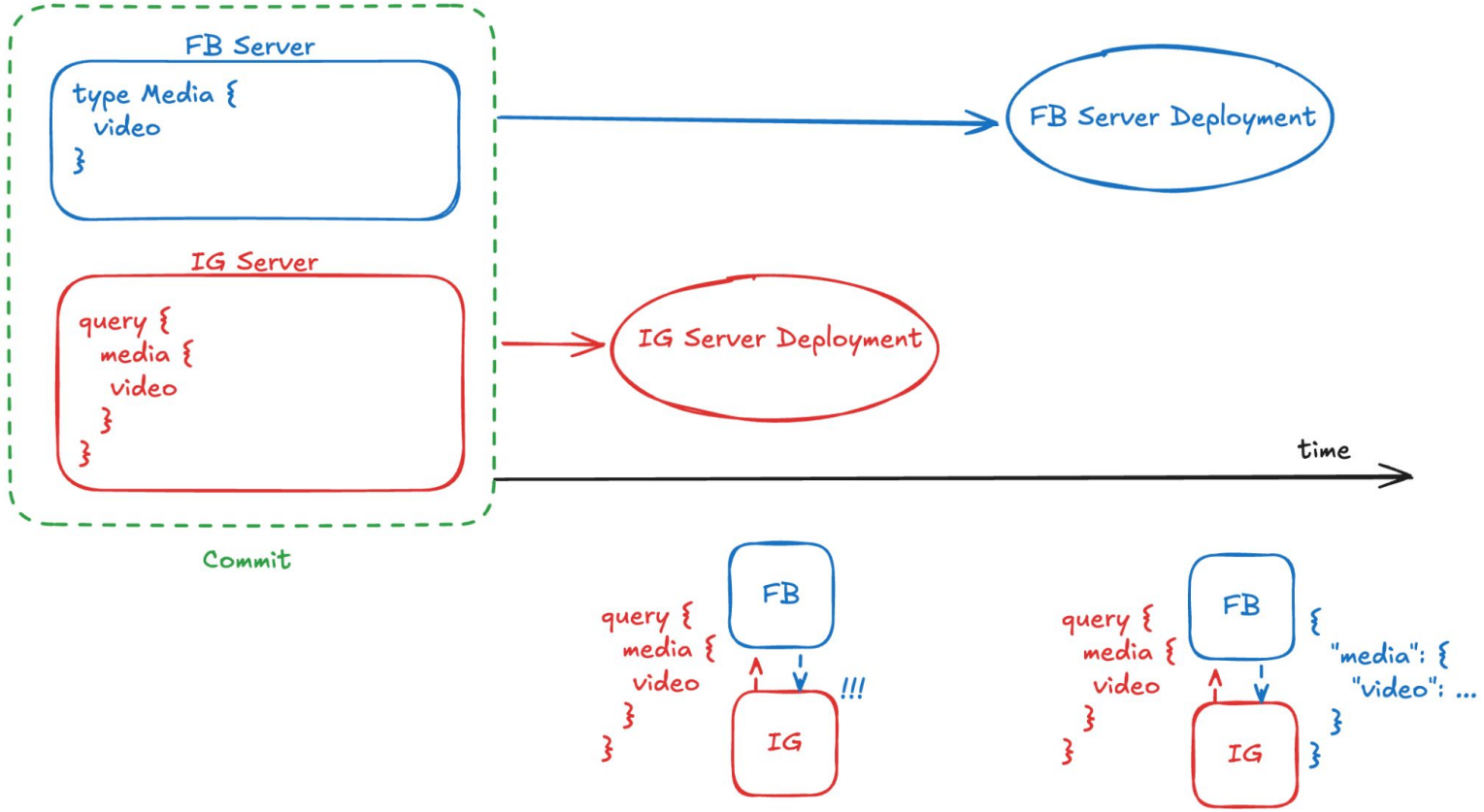
Cross Client-Server Change & Push Safety



Cross Client-Server Change & Push Safety



Cross Client-Server Change & Push Safety

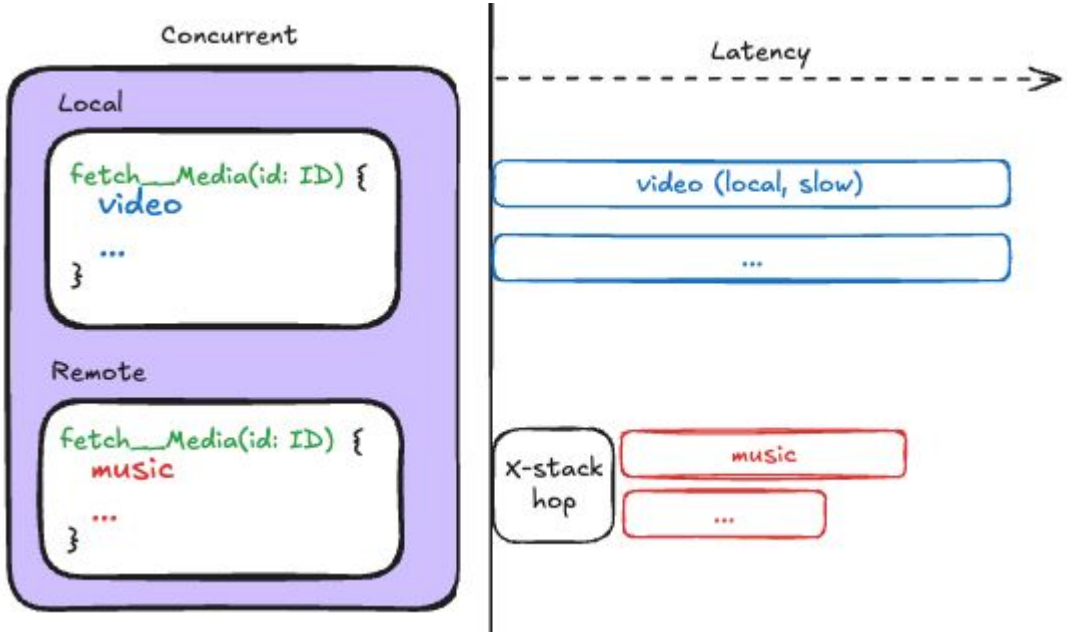


Cross Client-Server Change & Push Safety

What GraphQL Gives Us

- ✓ New nullable field → safe (old server returns null, client already handles it)
- ✗ New non-nullable field → unsafe (old server can't produce it)

Performance



GraphQLConf
2026

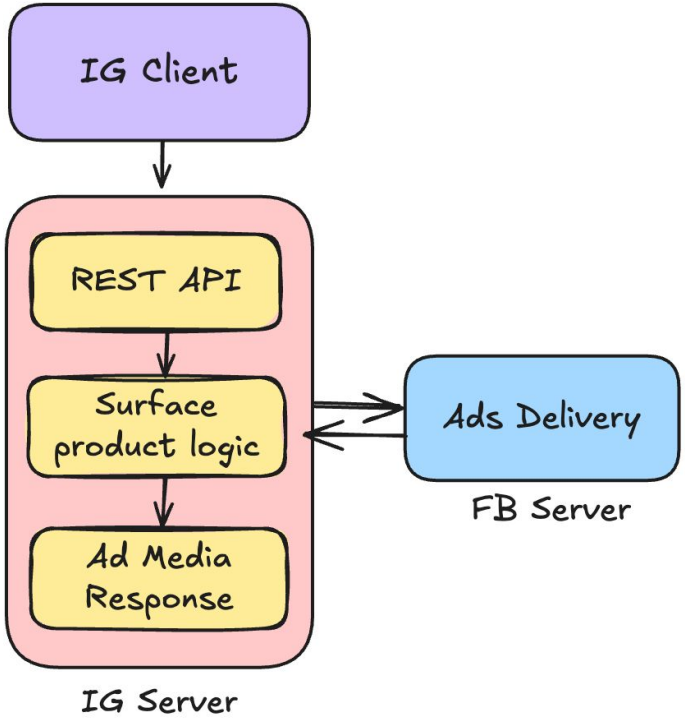
hosted by GraphQL Foundation

Case Study

IG Ads: Federated Schema Adoption

How It Started:

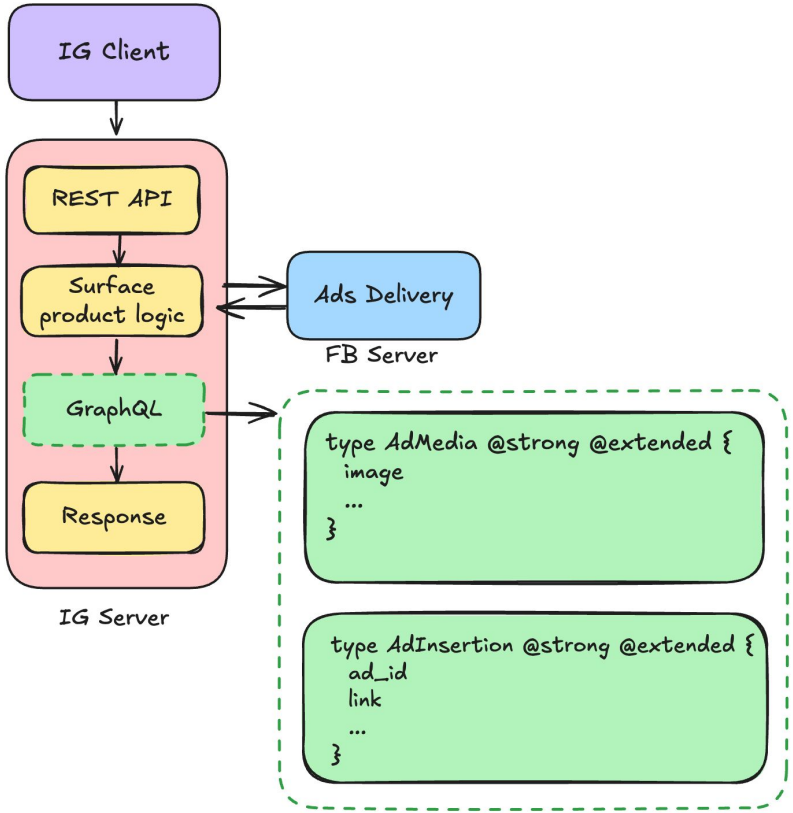
- IG Server serves as REST API entry point for ads – holds significant portion of business logic and computes 600+ ad media fields for the final client response.
- Fields logic tightly coupled and deeply intertwined.
- No GraphQL compatibility.



IG Ads: Federated Schema Adoption

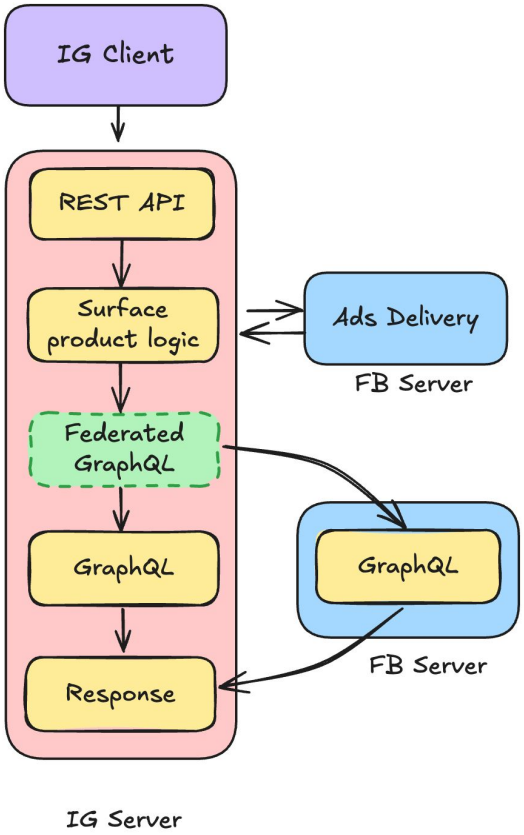
Switching to GraphQL Extended Types:

- Modular architecture — every field encapsulated as its own GraphQL function, decoupled from the monolithic rendering pipeline.
- Performance & efficiency gains — improved latency and capacity across all IG ad surfaces as ads rendering became independent, streamlined and more optimized.



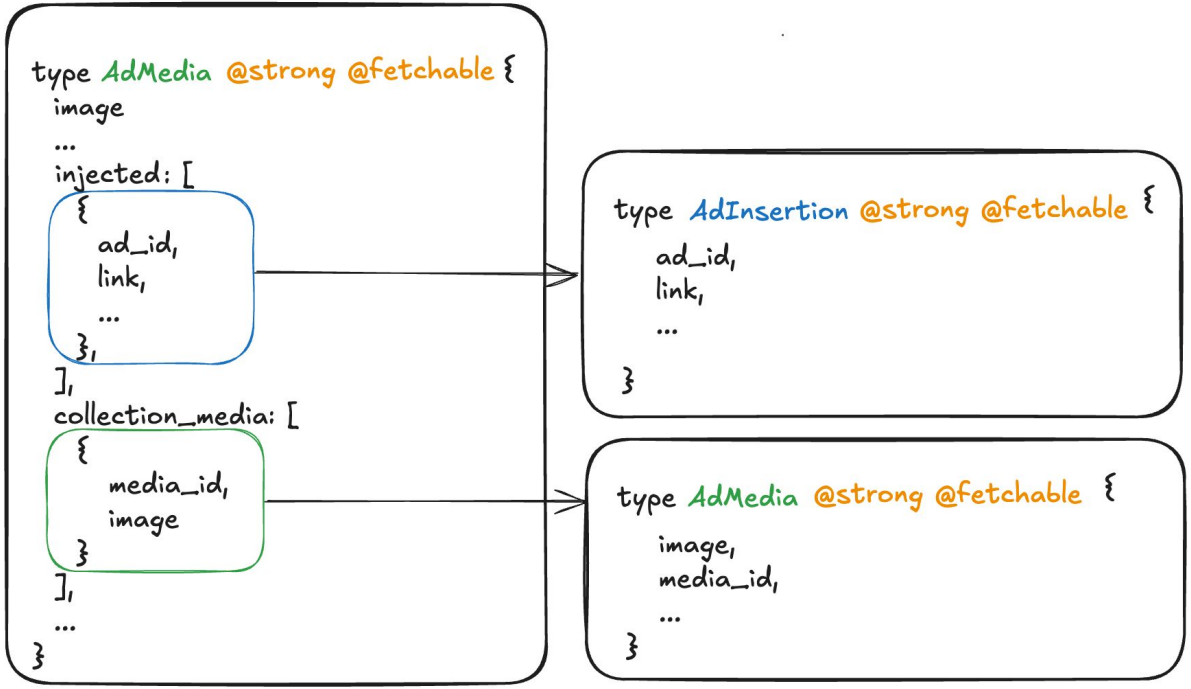
IG Ads: Federated Schema Adoption

- Federated Schema unblocked incremental ads fields migration to FB Server.
- New fields can be defined and implemented entirely in FB Server, zero IG Server code needed.



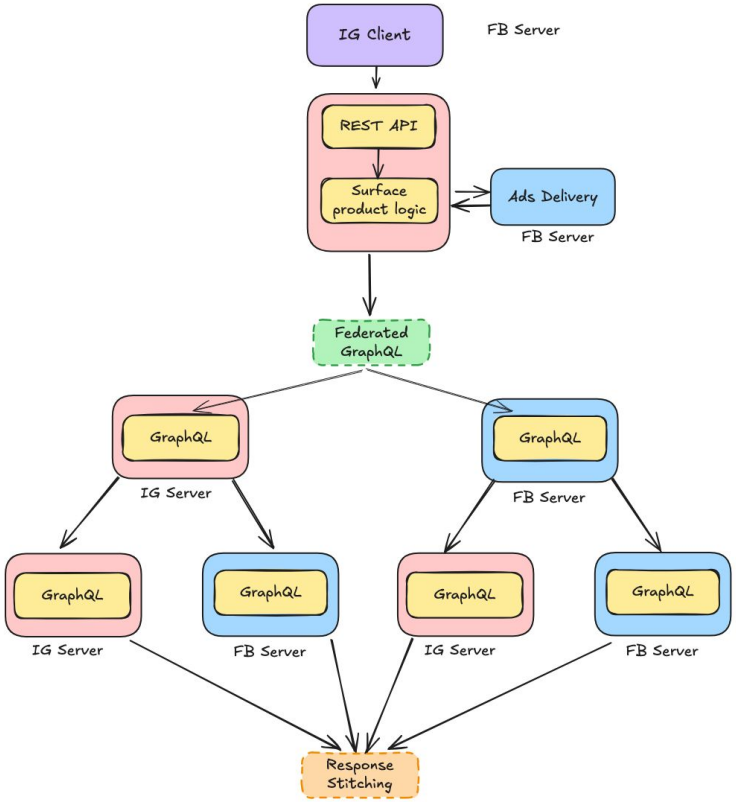
IG Ads: Federated Schema Adoption

Ads responses have multiple layers of extended types nested inside each other:



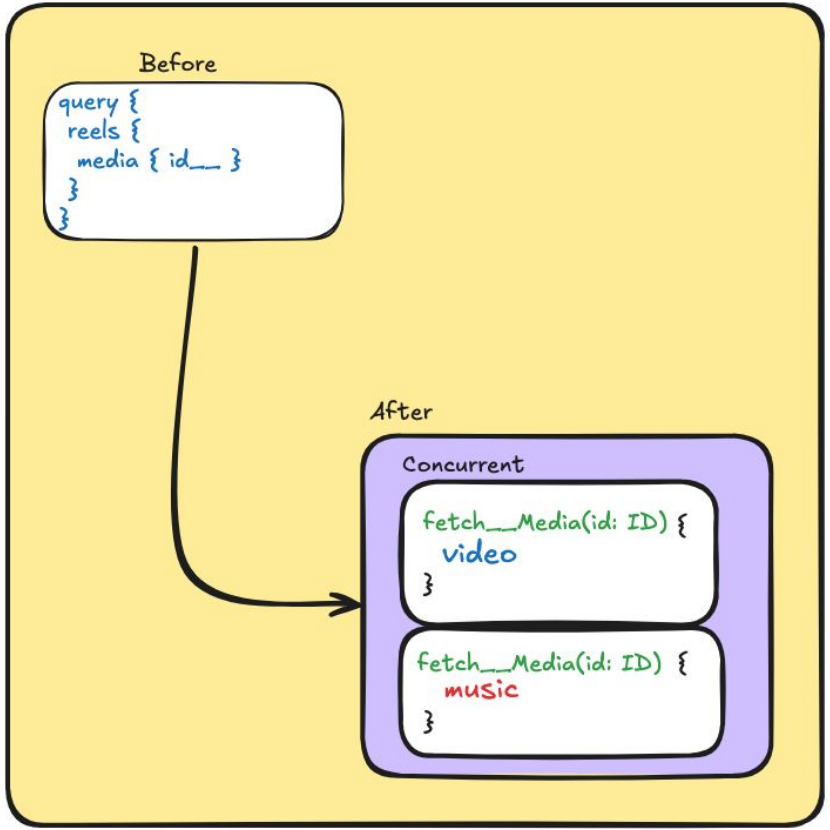
IG Ads: Federated Schema Adoption

- Separate cross-stack call for each nested type boundary.
- Each call = serialization + network hop + deserialization overhead.



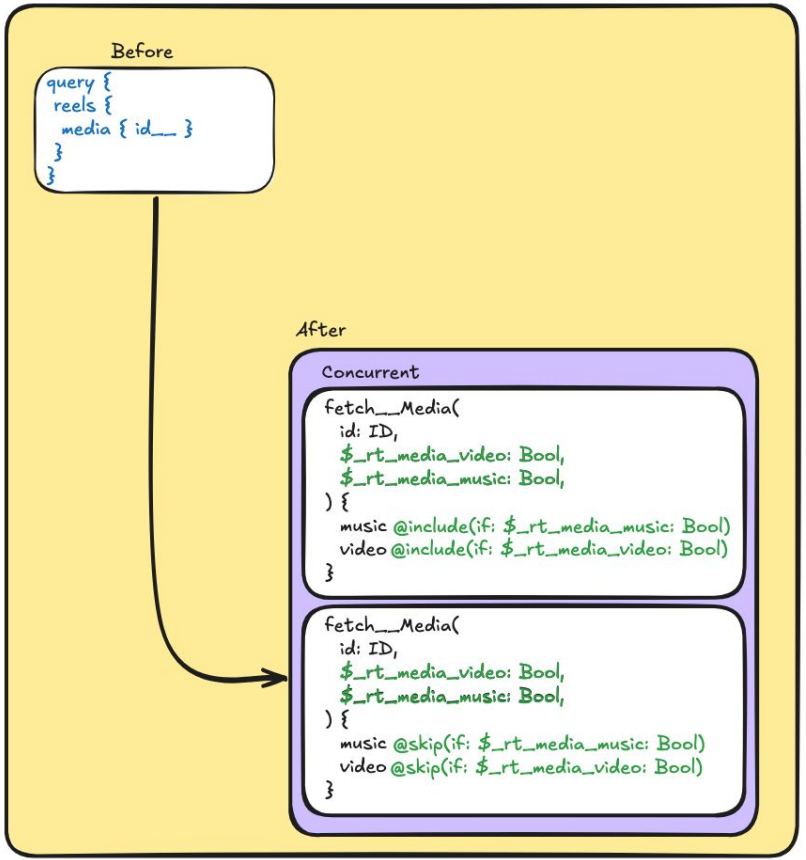
Evolving Routing: Static Query Plan

- Initial Design
- Encodes routing in the query plan itself.
- **Limitation**: Routing changes require rebuilding and repersisting the query itself.

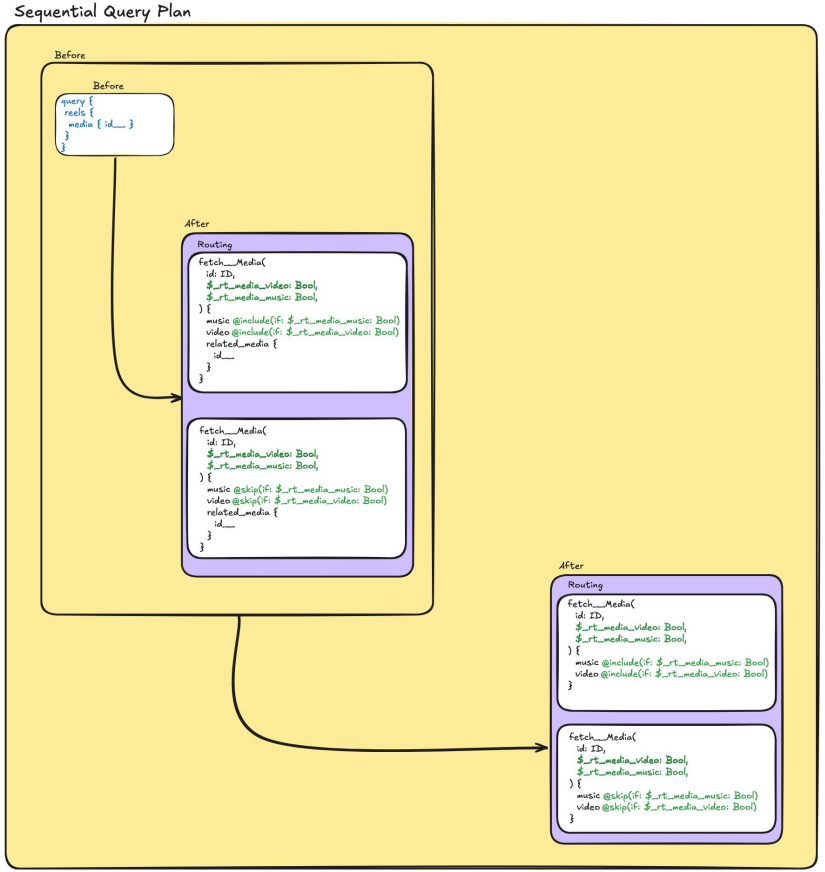


Evolving Routing: Dynamic Query Plan

- Enables dynamic routing
- `@skip` and `@include` directives allow dynamic routing.
- Routing is computed at execution time.
- **Limitation:** Nested Extended Types lead to deeply nested query plans.

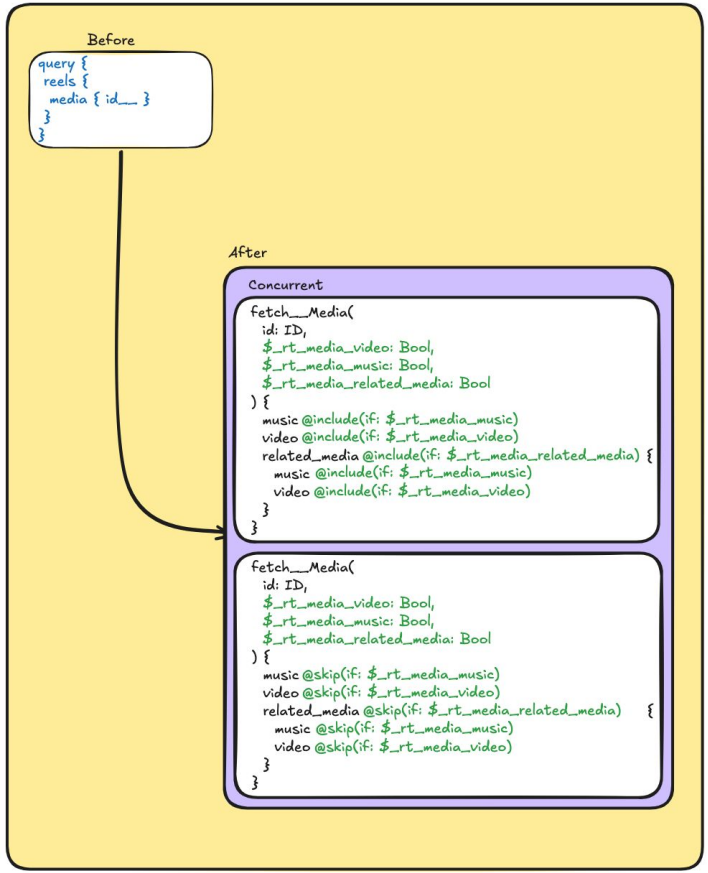


Evolving Routing: Dynamic Query Plan (cont)



Evolving Routing: Dynamic Query Plan V2

- Flattens the nestedness of Routing Query Plans.
- **Pros**
  - Simpler structure.
  - Eliminates latency overhead from V1.
- **Limitations:**
  - Higher CPU cost to re-execute paths in both stacks.



IG Ads - GraphQL & Monoschema Adoption

Product Correctness & Testing:

- DMS not supported in test environments — persisted query IDs don't exist in test tier.
- Cross-stack calls fail silently, tests fall back to hardcoded mocks.
- More fields migrated to WWW = less real test coverage (migration progress inversely correlated with testing confidence).
- Resolved with cross-stack testing framework — a framework that spins up a real WWW test server alongside Distillery, integration tests can make actual DMS calls E2E.

IG Ads - GraphQL & Monoschema Adoption

Key Learnings:

- Understand your type graph before migration, not after. A flat schema splits cleanly across servers. But the moment you have nested types, each boundary multiplies your cross-stack overhead. RQP is essential but requires planning upfront.
- Invest in cross-stack testing early. Don't let your testing infrastructure lag behind your migration.
- Budget before you bridge — splitting queries across servers costs real latency; earn that margin upfront so the migration is a trade-off, not a regression.

GraphQLConf
2026

hosted by GraphQL Foundation

Conclusion

GraphQLConf 2026

hosted by  GraphQL Foundation