Scaling GraphQL for 500,000,000 req/min

whoami

Tushar Mathur

- GraphQL Enthusiast
- Open Source Contributor
- Passionate about DevEx

Part 1

GraphQL Journey

2016 Dream11

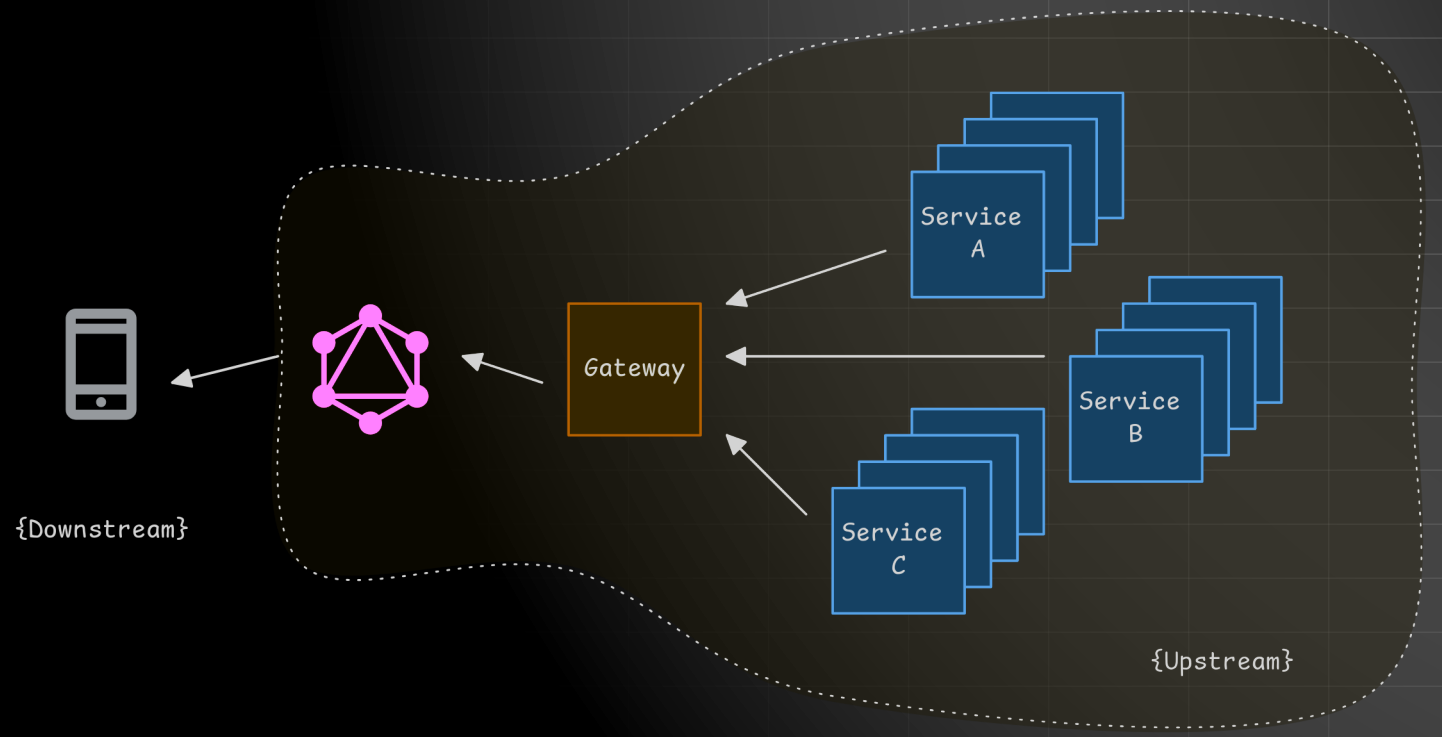
- Fantasy Gaming Platform
- Early Stage
- Monolith to Microservices Considerable Scale

2016 Dream11

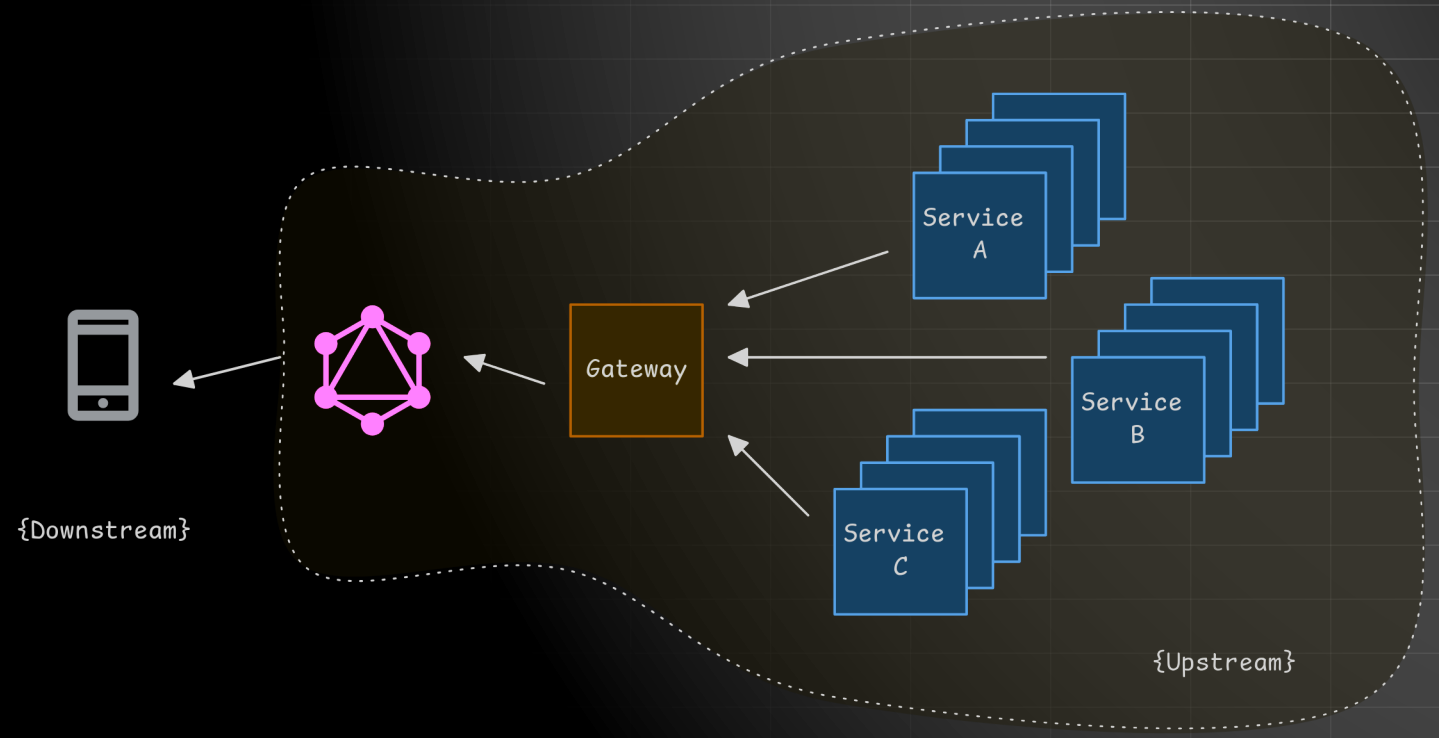
- Fantasy Gaming Platform
- Early Stage
- Monolith to Microservices Considerable Scale

 GraphQL?

Architecture



Architecture



GraphQL as client-side abstraction

2022 Dream11

500
ENGINEERS

150
MICROSERVICES

10,000,000
CONCURRENCY

200,000,000
USERS

2022 Dream11

500 ENGINEERS	150 MICROSERVICES
10,000,000 CONCURRENCY	200,000,000 USERS

GraphQL

Saved the day...

Not so fast!

Part 2

Challenges

1. Performance

1. Performance

👍 Latency & Throughput

1. Performance

👍 Latency & Throughput

👎 Infrastructure

1. Performance

👍 Latency & Throughput

👎 Infrastructure

50,000

CORES

Things that worked for us

Things that worked for us

👉 Micro Optimizations

Things that worked for us

👉 Micro Optimizations

👉 Infrastructure Tuning

Things that worked for us

👉 Micro Optimizations

👉 Infrastructure Tuning

👉 Benchmarking

Things that worked for us

- 👉 Micro Optimizations
- 👉 Infrastructure Tuning
- 👉 Benchmarking
- 👉 Caching on Upstream

Things that worked for us

- 👉 Micro Optimizations
- 👉 Infrastructure Tuning
- 👉 Benchmarking
- 👉 Caching on Upstream

6,400
CORES

Things that worked for us

- 👉 Micro Optimizations
- 👉 Infrastructure Tuning
- 👉 Benchmarking
- 👉 Caching on Upstream

6,400
CORES

High Priority

2. Reliability

2. Reliability

👎 Fragility

2. Reliability

👎 Fragility

2. Reliability

👎 Fragility

👎 Resiliency

Things that worked for us

Things that worked for us

👉 Redefining Processes

Things that worked for us

👉 Redefining Processes

👉 Micro-optimizing Cold Paths

Things that worked for us

👉 Redefining Processes

👉 Micro-optimizing Cold Paths

👉 Rate Limiting & Circuit Breaking

Things that worked for us

👉 Redefining Processes

👉 Micro-optimizing Cold Paths

👉 Rate Limiting & Circuit Breaking

❌ Query Cost

3. Maintainability

3. Maintainability

- ❓ Library Upgrades
- ❓ Unused Nodes
- ❓ Data Loaders
- ❓ Unimplemented Resolver
- ❓ Code Duplication
- ❓ ...

3. Maintainability

- ❓ Library Upgrades
- ❓ Unused Nodes
- ❓ Data Loaders
- ❓ Unimplemented Resolver
- ❓ Code Duplication
- ❓ ...

Summary

1. Performance ⭐⭐☆
2. Reliability ⭐⭐☆
3. Maintenance ☆☆☆

Part 3

Learnings of 8 Years

🕊️ Liberties Constraint and Constraints Liberate🕊️

Liberties

Constraints

GraphQL

1. Schema

2. Query

3. Resolver

🕊️ Resolver 🕊️

Conformance requirements expressed as algorithms can be fulfilled by an implementation of this specification in any way as long as the perceived result is equivalent. Algorithms described in this document are written to be easy to understand. Implementers are encouraged to include equivalent but optimized implementations.

Step 1

✋ Avoid Business Logic

👍 Allow only Orchestration Logic

Step 2

🙅‍♂️ No Handwritten Resolver

🤔 Configuration Based

Step 3

🤔 Learn from SQL

Part 4

Generalized Runtime for GraphQL


tailcall

Built with 💗 using

Apache 2.0 License

High Performance

How does it work?

GraphQL Schema 👉

```
schema {
  query: Query
}

type Query {
  posts: [Post]
}

type Post {
  id: ID!
  title: String!
  body: String
  userId: ID!
  user: User
}

type User {
  id: ID!
  name: String!
  email: String!
}
```

Annotate 👉

```
schema @upstream(baseURL: "https://api.d11.local") {
  query: Query
}

type Query {
  posts: [Post] @http(path: "/posts")
}

type Post {
  id: ID!
  title: String!
  body: String
  userId: ID!
  user: User @http(path: "/users/{{value.userId}}")
}

type User {
  id: ID!
  name: String!
  email: String!
}
```

Start the server

```
❯ tailcall start config.graphql
  INFO File read: config.graphql ... ok
  INFO 🚀 Tailcall launched at [127.0.0.1:8000] over HTTP/1.1
```

First Class Support

- REST
- gRPC
- GraphQL

First Class Support

- REST
- gRPC
- GraphQL

```
@addField
@call
@graphQL
@http
@modify
@rest
@telemetry
@cache
@expr
@grpc
@link
@omit
@server
@upstream
```

First Class Support

- REST
- gRPC
- GraphQL

```
@addField    @cache
@call        @expr
@graphQL     @grpc
@http        @link
@modify      @omit
@rest        @server
@telemetry   @upstream
```

Router

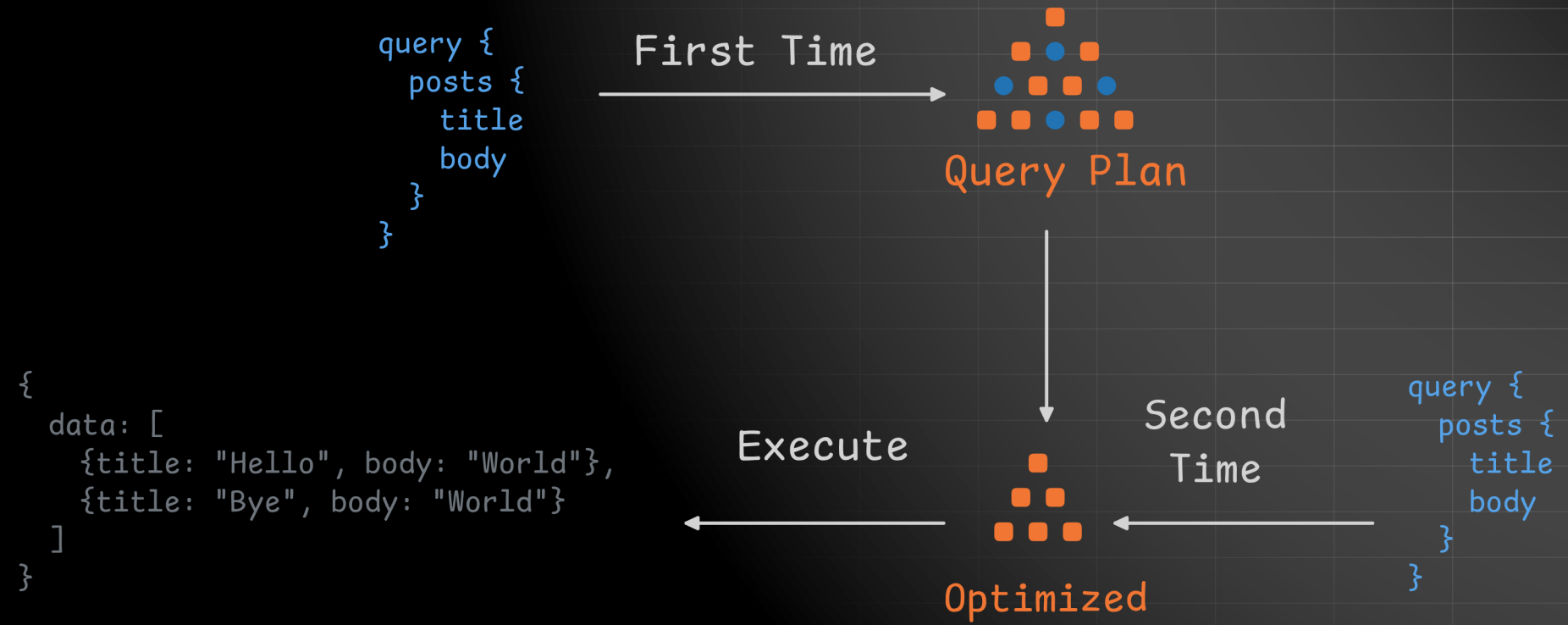
Constraints that Liberate

2 Key Capabilities

AOT Analyzer

```
❯ tailcall start config.graphql
  INFO File read: config.graphql ... ok
  INFO N + 1 detected: 1
  ERROR Invalid Configuration
  Caused by:
  • argument 'id' is a nullable type [at Query.args.id]
```

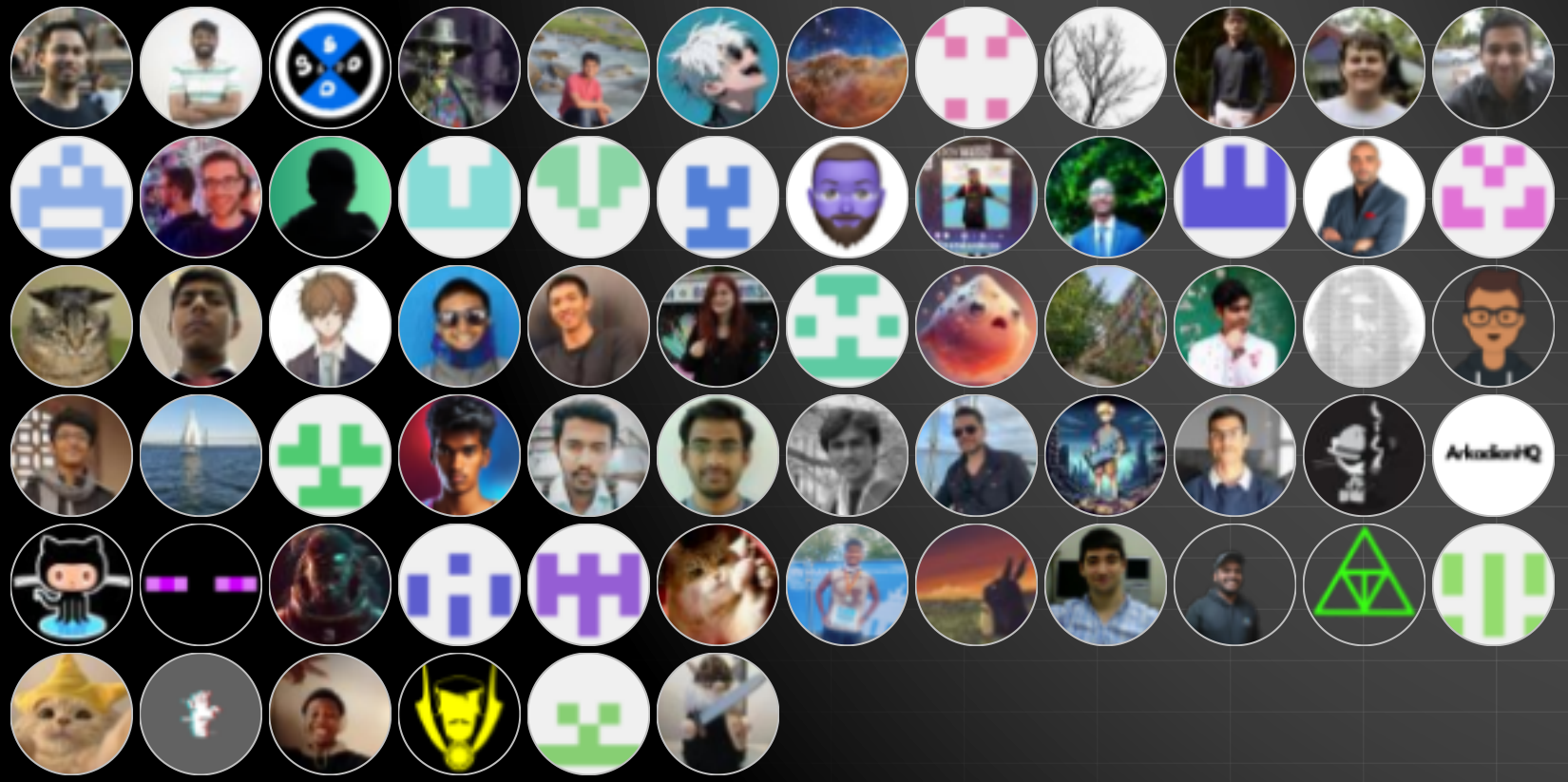
JIT Optimizer



Revisiting the Challenges

	Before	tailcall
1. Performance	★★☆	★★★
2. Reliability	★★☆	★★★
3. Maintenance	☆☆☆	★★★
4. Flexibility	★★★	★☆☆

Awesome Community 🙌



Part 5

Takeaway

Takeaway

1. Innovation on GraphQL performance is necessary.
2. Handwritten GraphQL is difficult to maintain.
3. Library authors should take inspiration from SQL engines.

#TailcallHack

- Build the fastest GraphQL Server
- Checkout - tailcallhq/hackathon

$ 5,000

Thank You! ❤️

Tushar Mathur

1. Founder of Tailcall
2. Ex VP of Engineering at Dream11 (2016-2022)

@tusharmath @tusharmath @tusharmath