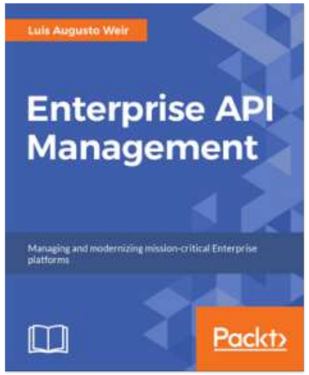
GraphQL as an alternative approach to REST

About me

Luis Weir
CTO at Capgemini UK Oracle
Ace Director & Groundbreaker Ambassador

luis.weir@capgemini.com
uk.linkedin.com/in/lweir
http://www.soa4u.co.uk



Luis Augusto Weir

Enterprise API Management

Managing and modernizing mission-critical Enterprise platforms

Packt>

tinyurl.com/eapim18
Goes to Print Q2 2019

EXPERT INSIGHT

Implementing Oracle API Platform Cloud Service

Foreword by:
Vikas Anand
Vice President Product Management Oracle Corporation

Design, deploy, and manage your APIs in Oracle's new API Platform

Andrew Bell
Luis Weir
Sander Rensen
Phil Wilkins

Packt>

apiplatform.cloud/
Released Q2 2018

Want the slides? I'll share via:

@luisw19

Latest articles:

- The 7 Deadly Sins of API Design
- Setting the vision, strategy and direction — the CTO's role
- How can you design, deploy and manage your APIs?
- The Spotify's Engineering Culture. My interpretation and summary
- A comparison of API Gateways communication styles
- Is BPM Dead, Long Live Microservices?
- Five Minutes with Luis Weir
- 2nd vs 3rd Generation API Platforms - A Comprehensive Comparison
- Podcast: Are Microservices and APIs Becoming SOA 2.0?
- 3rd-Generation API Management: From Proxies to Micro-Gateways
- Oracle API Platform Cloud Service Overview

GraphQL as an alternative appraoch to REST

01 GraphQL – context & key concepts

02 Demos

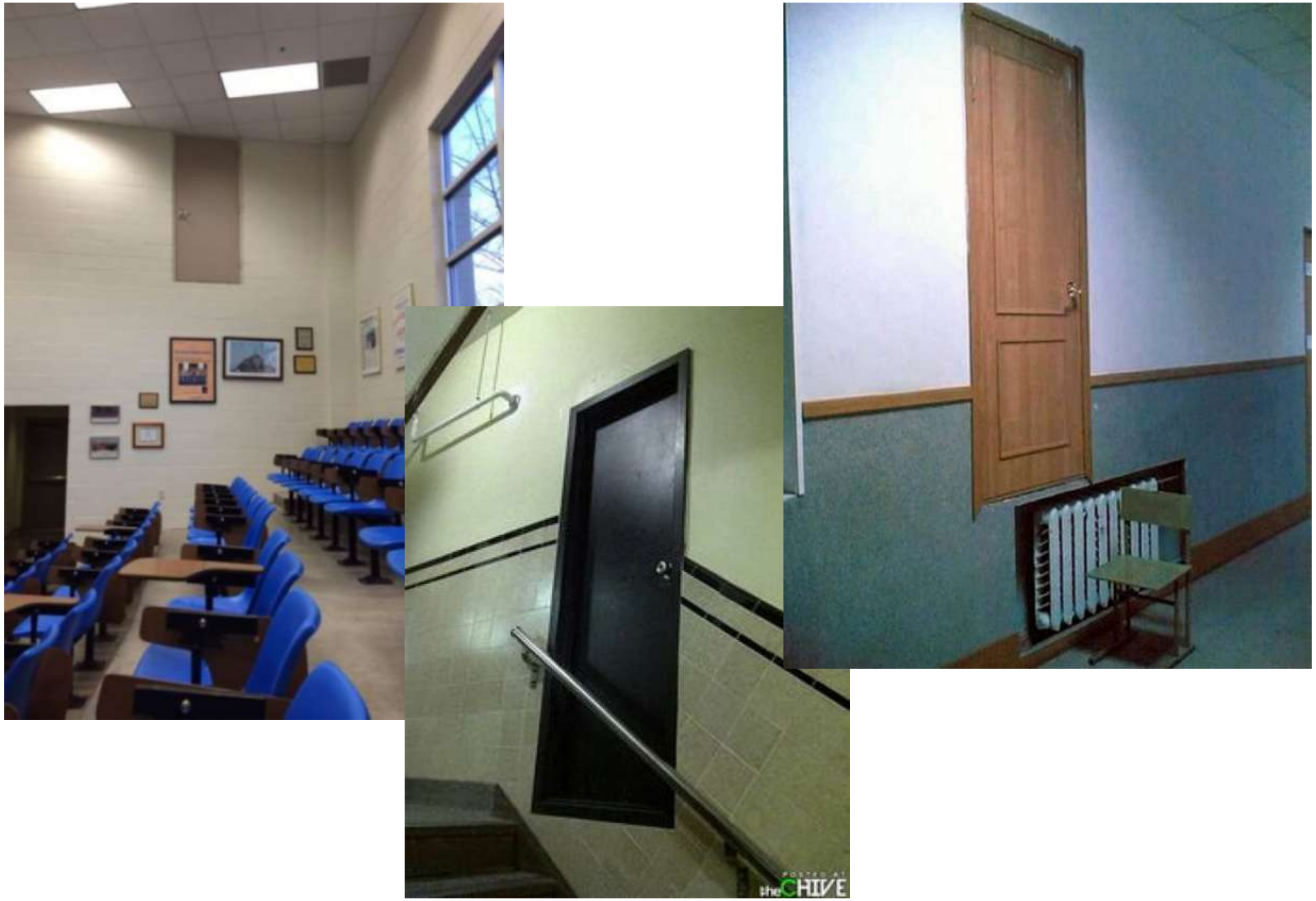
03 GraphQL vs REST PoV

04 Conclusions

@luisw19

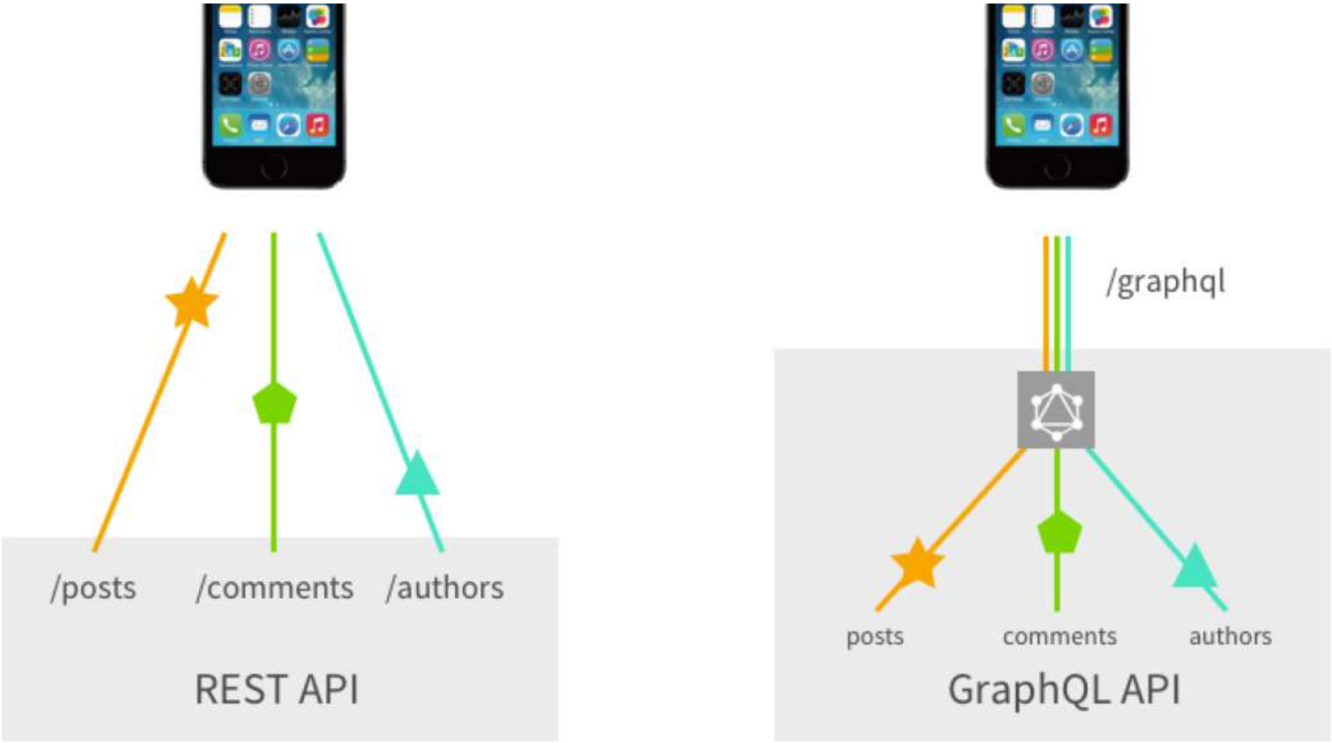
Application Programming Interfaces (**APIs**) are **doors** to **information** and **functionality.**

But some doors can be unfit for purpose…



source: https://imgur.com/a/J3ttg

Why GraphQL?



Source: https://dev-blog.apollodata.com/graphql-vs-rest-5d425123e34b by Sashko Stubailo

GraphQL - Background

- **Created by Facebook** in 2012 to get around a **common constraints** in the **REST** approach when **fetching data**
- Publicly **released** in **2015**
- GraphQL **Schema Definition Language (SDL)** added to spec in Feb’18

https://GraphQL.org

Latest release: http://facebook.github.io/graphql

Latest draft: http://facebook.github.io/graphql/draft/

GraphQL – What is it **NOT**?

in spite of its name, it has nothing to do with Graphs DBs

a query language for a databases

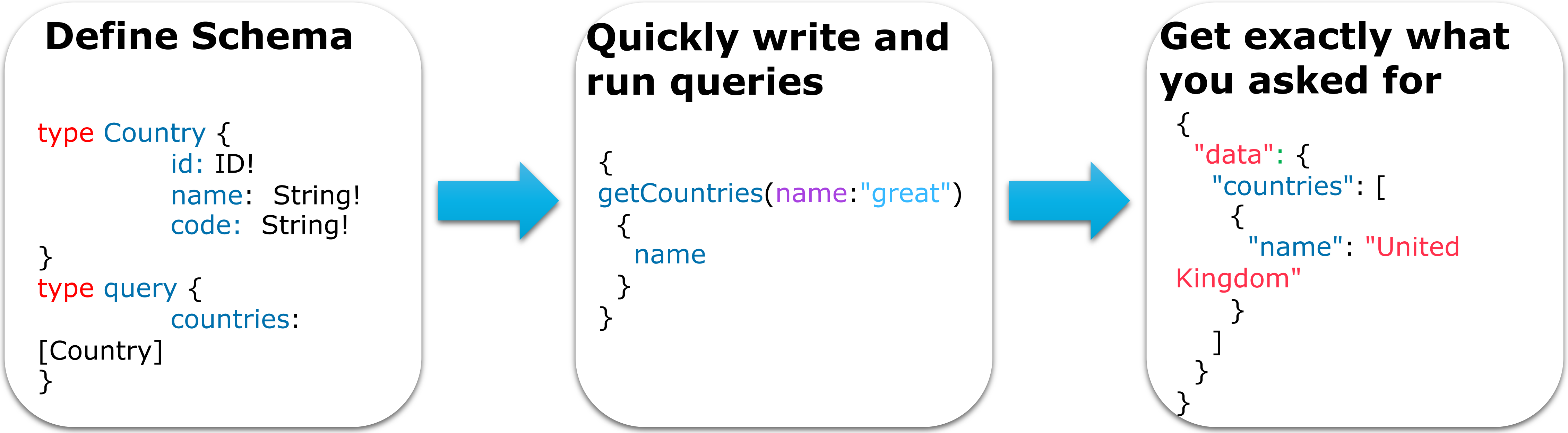
necessarily a replacement for REST. Both can work together

A silver Bullet

Roy from the IT Crowed →

GraphQL – What is it then?

A consumer oriented **query language,** a strongly typed **schema language** and a **runtime** to implement **GraphQL services**.



Define Schema

```
type Country {
             id: ID!
             name:  String!
             code:  String!
}
type query {
             countries:
[Country]
}
```

GraphQL Service

Quickly write and run queries

```
{
getCountries(name:"great")
 {
   name
 }
}
```

GraphQL Client

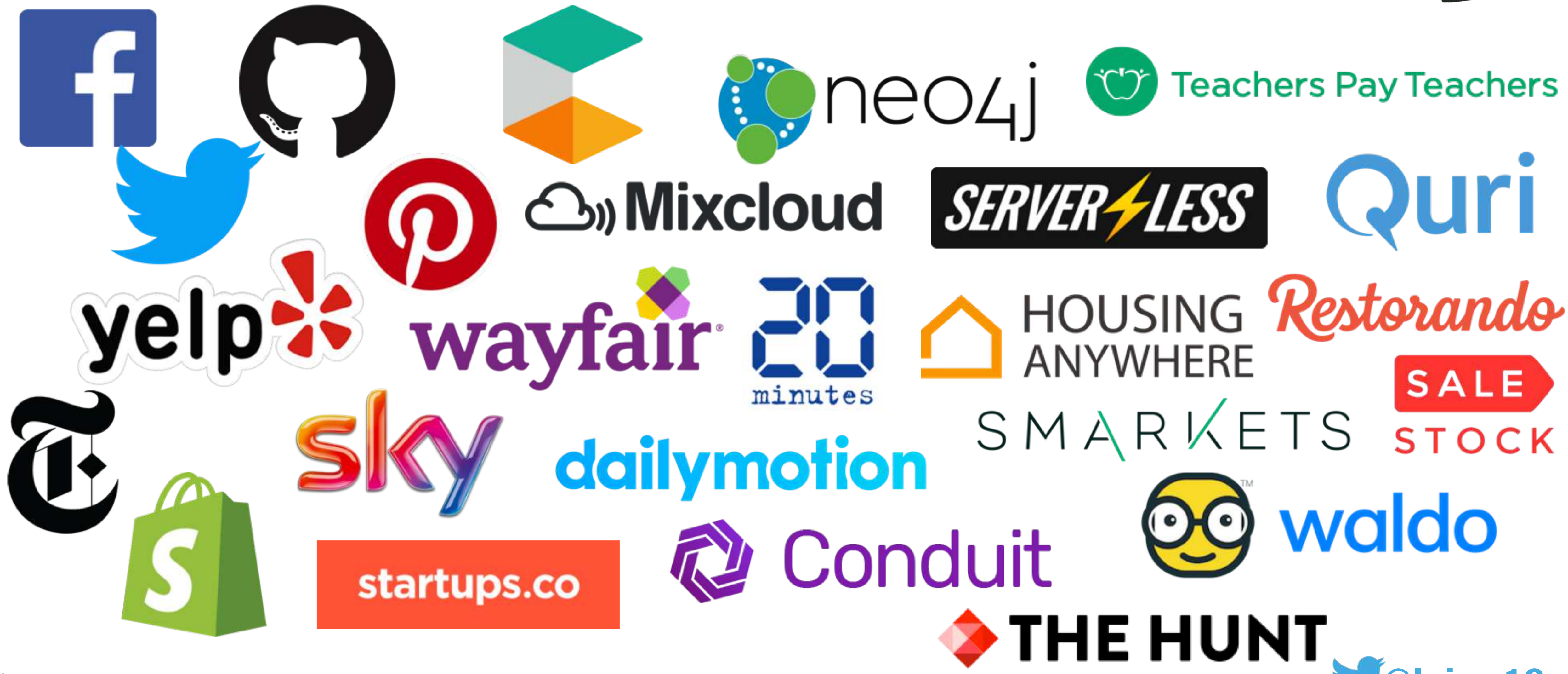
Get exactly what you asked for

```
{
  "data": {
    "countries": [
      {
        "name": "United
Kingdom"
      }
    ]
  }
}
```

GraphQL Client

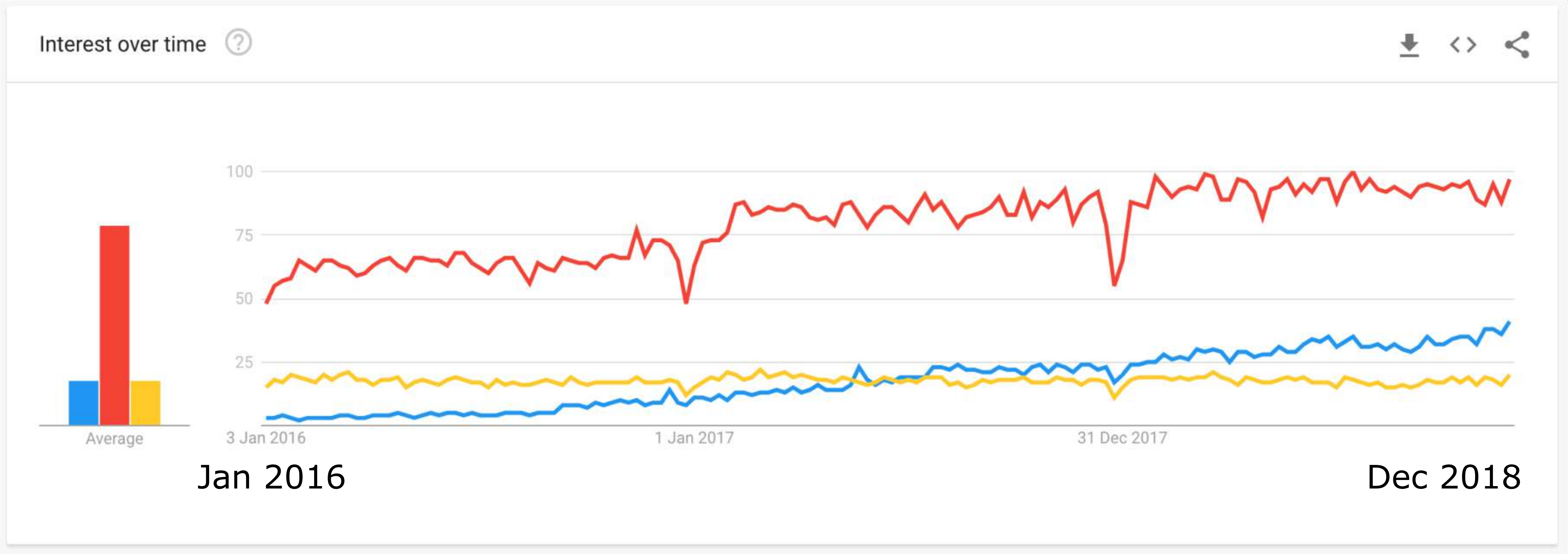
Who is using it?

Lots of organisations are embracing GraphQL: http://graphql.org/users



Increasing rapidly in Popularity

● GraphQL ● REST API ● OData

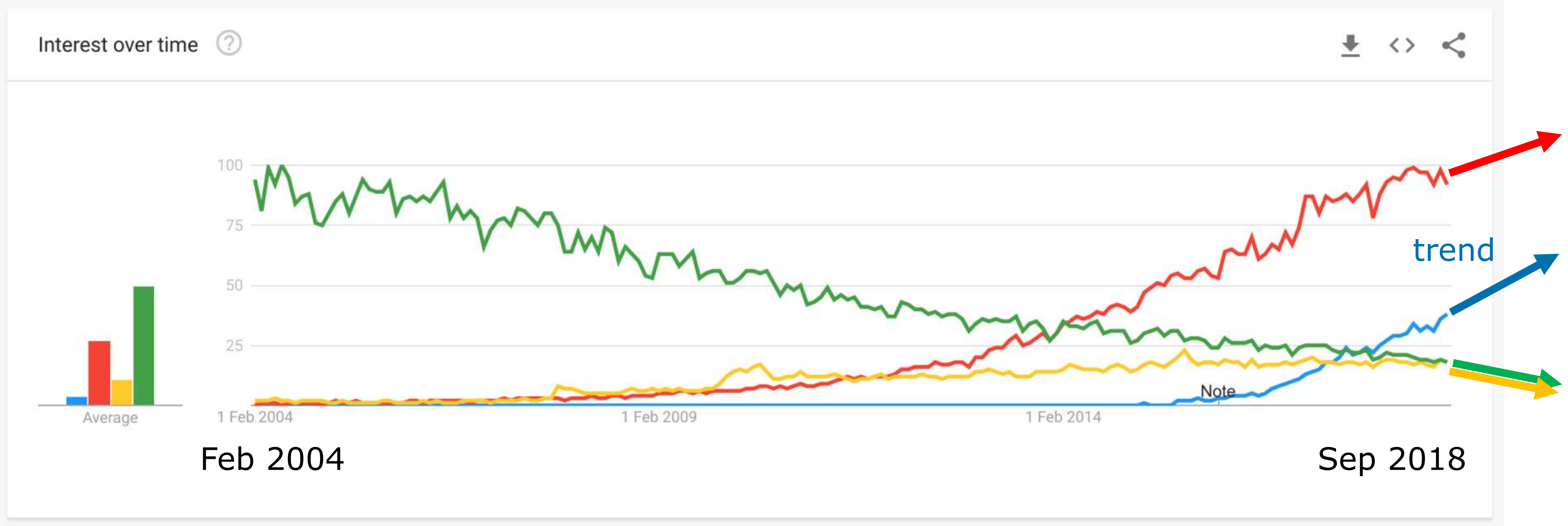


Jan 2016 Dec 2018

https://trends.google.com/trends/explore?date=2016-01-01%202018-12-01&q=GraphQL,REST%20API,OData

Let’s put it into perspective

● GraphQL ● REST API ● OData ● WSDL



trend

Feb 2004

Sep 2018

https://trends.google.com/trends/explore?date=2004-01-10%202018-11-30&q=GraphQL,REST%20API,OData,WSDL

GraphQL – Key Concepts

There are 5 key characteristics of GraphQL that are important to understand:

1

Hierarchical

Queries as hierarchies of data definitions, shaped just how data is expected to be retuned.

2

View-centric

By design built to satisfy frontend application requirements.

3

Strongly-typed

A GraphQL server defines a specific type system. Queries are executed within this context.

4

Introspective

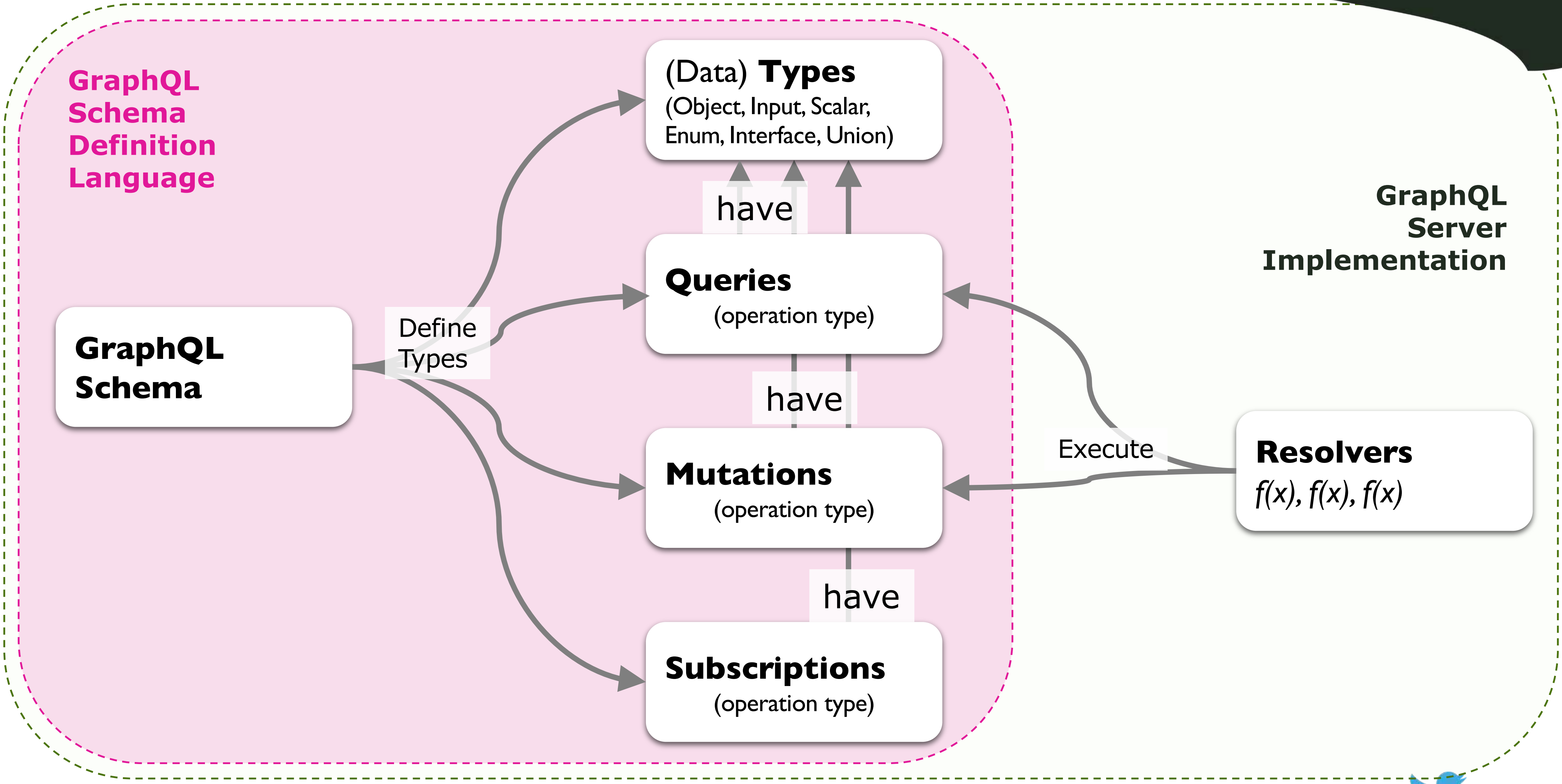
The type system itself is queryable. Tools are built around this capability.

5

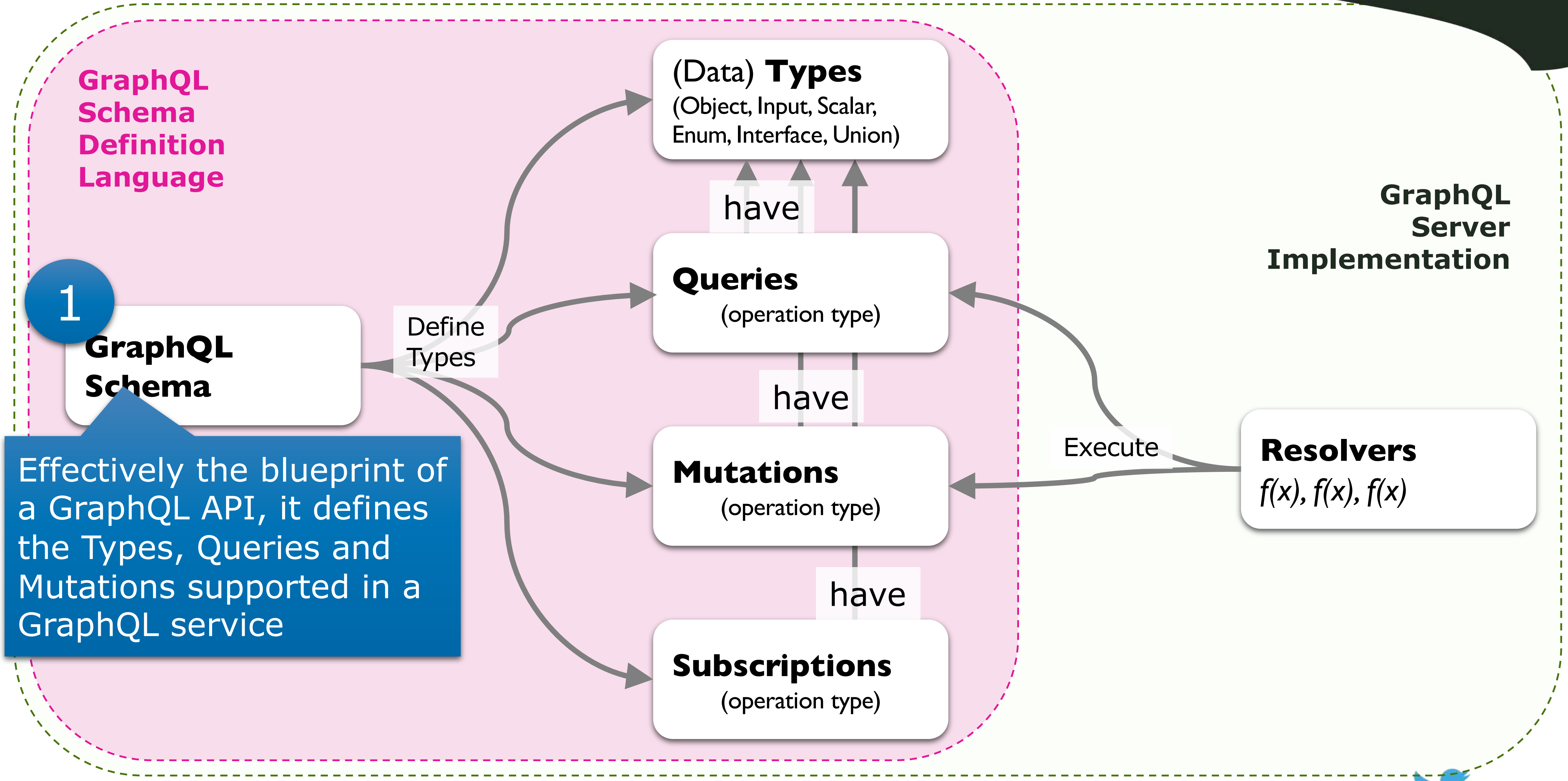
Version-Free

GraphQL takes a strong opinion on avoiding versioning by providing the tools for the continuous evolution.

GraphQL – Anatomy

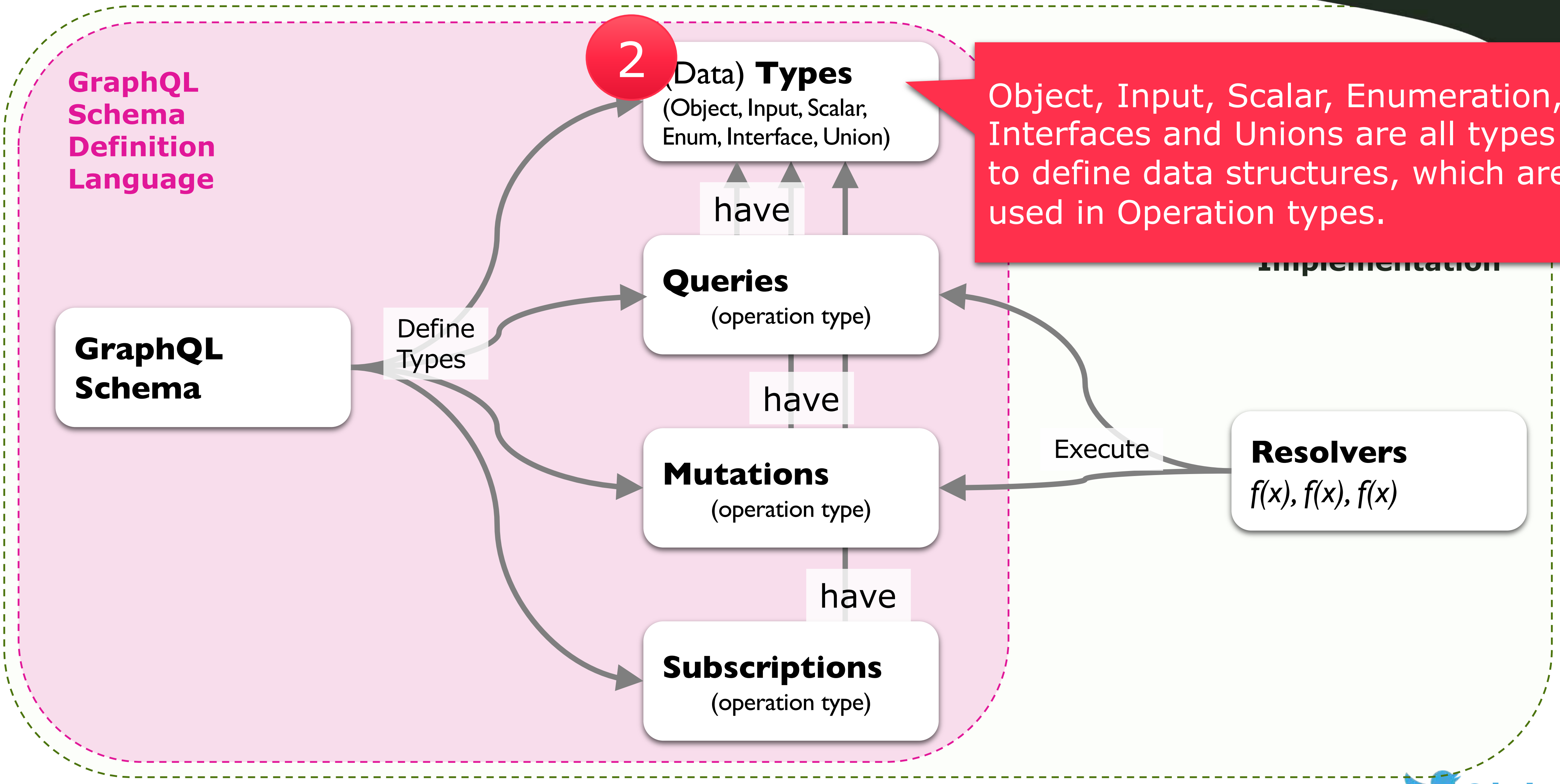


GraphQL – Anatomy



Effectively the blueprint of a GraphQL API, it defines the Types, Queries and Mutations supported in a GraphQL service

GraphQL – Anatomy



GraphQL Schema Definition Language

2

(Data) **Types**
(Object, Input, Scalar, Enum, Interface, Union)

have

Queries
(operation type)

have

Mutations
(operation type)

have

Subscriptions
(operation type)

GraphQL Schema

Define Types

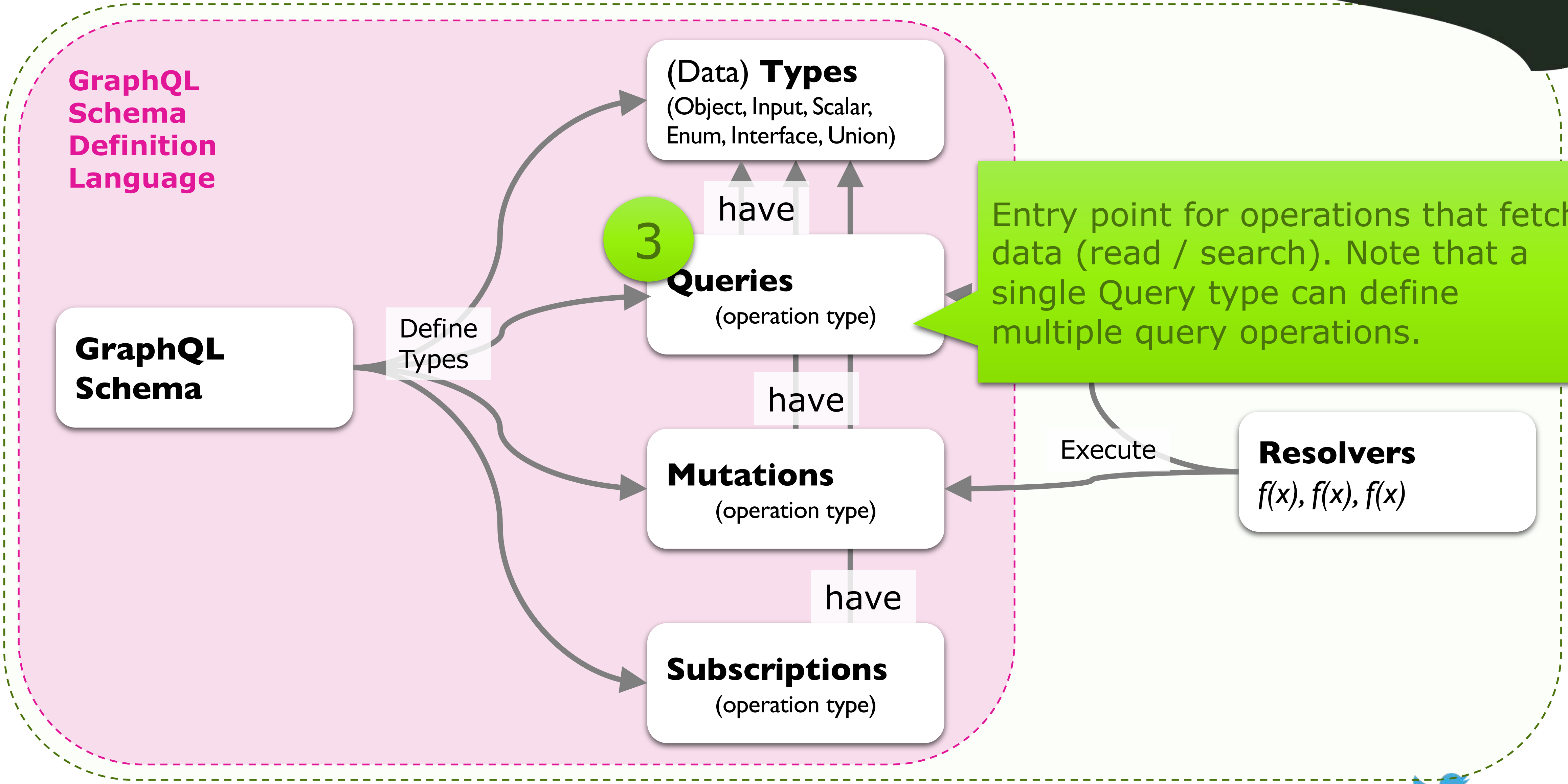
Implementation

Execute

Resolvers
f(*x*), *f*(*x*), *f*(*x*)

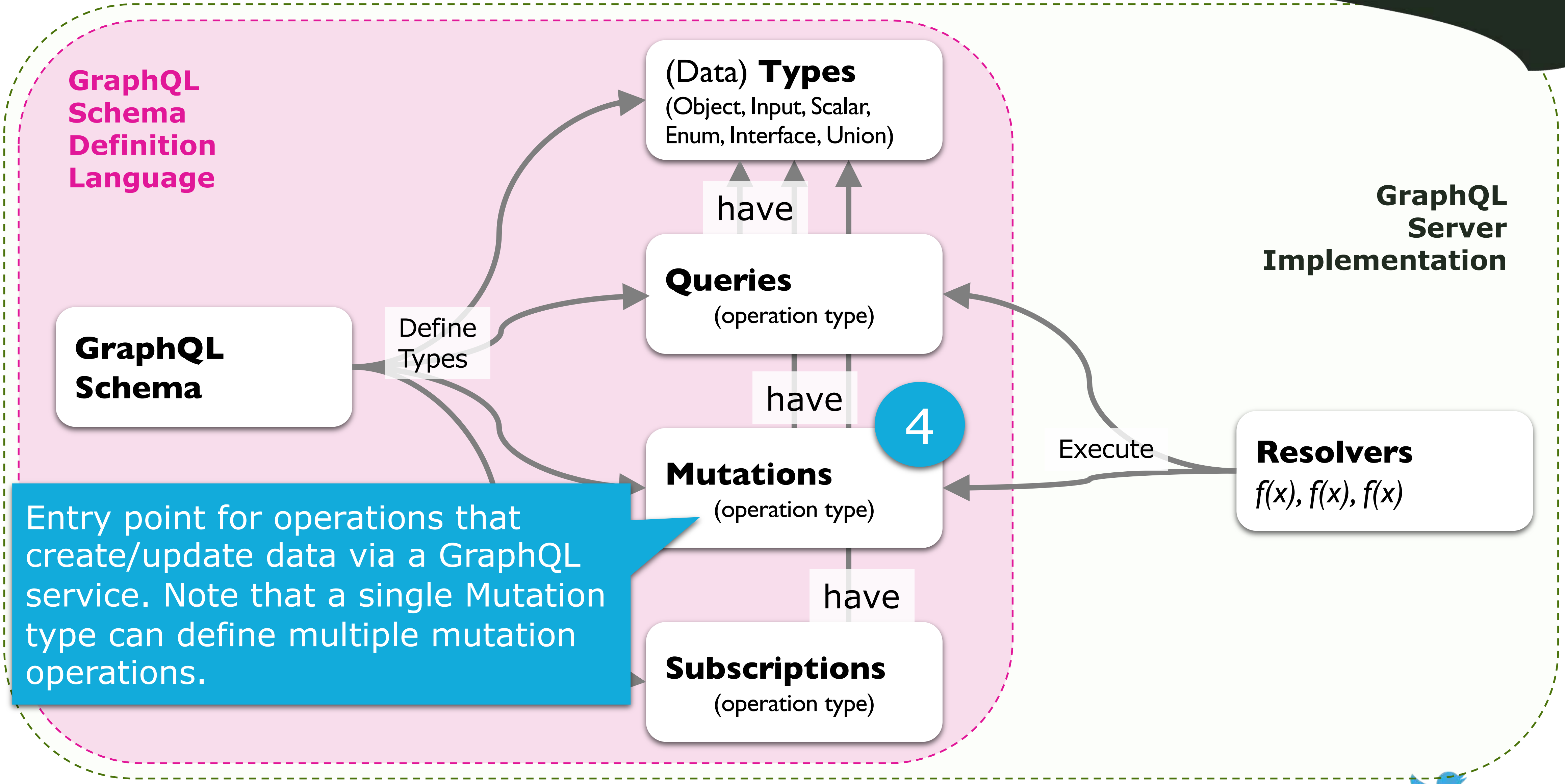
Object, Input, Scalar, Enumeration, Interfaces and Unions are all types to define data structures, which are used in Operation types.

GraphQL – Anatomy



Entry point for operations that fetch data (read / search). Note that a single Query type can define multiple query operations.

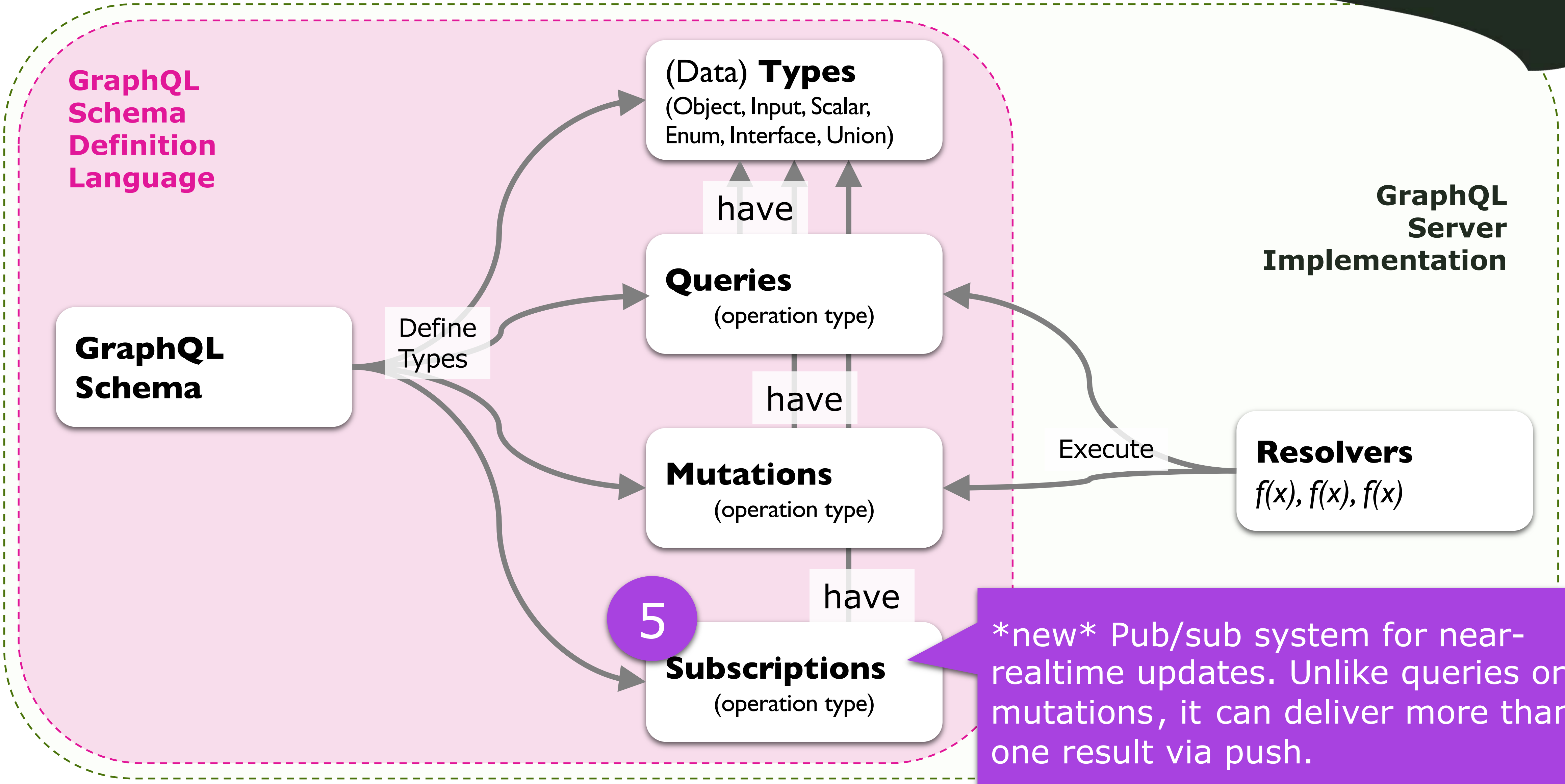
GraphQL – Key Concepts



Entry point for operations that create/update data via a GraphQL service. Note that a single Mutation type can define multiple mutation operations.

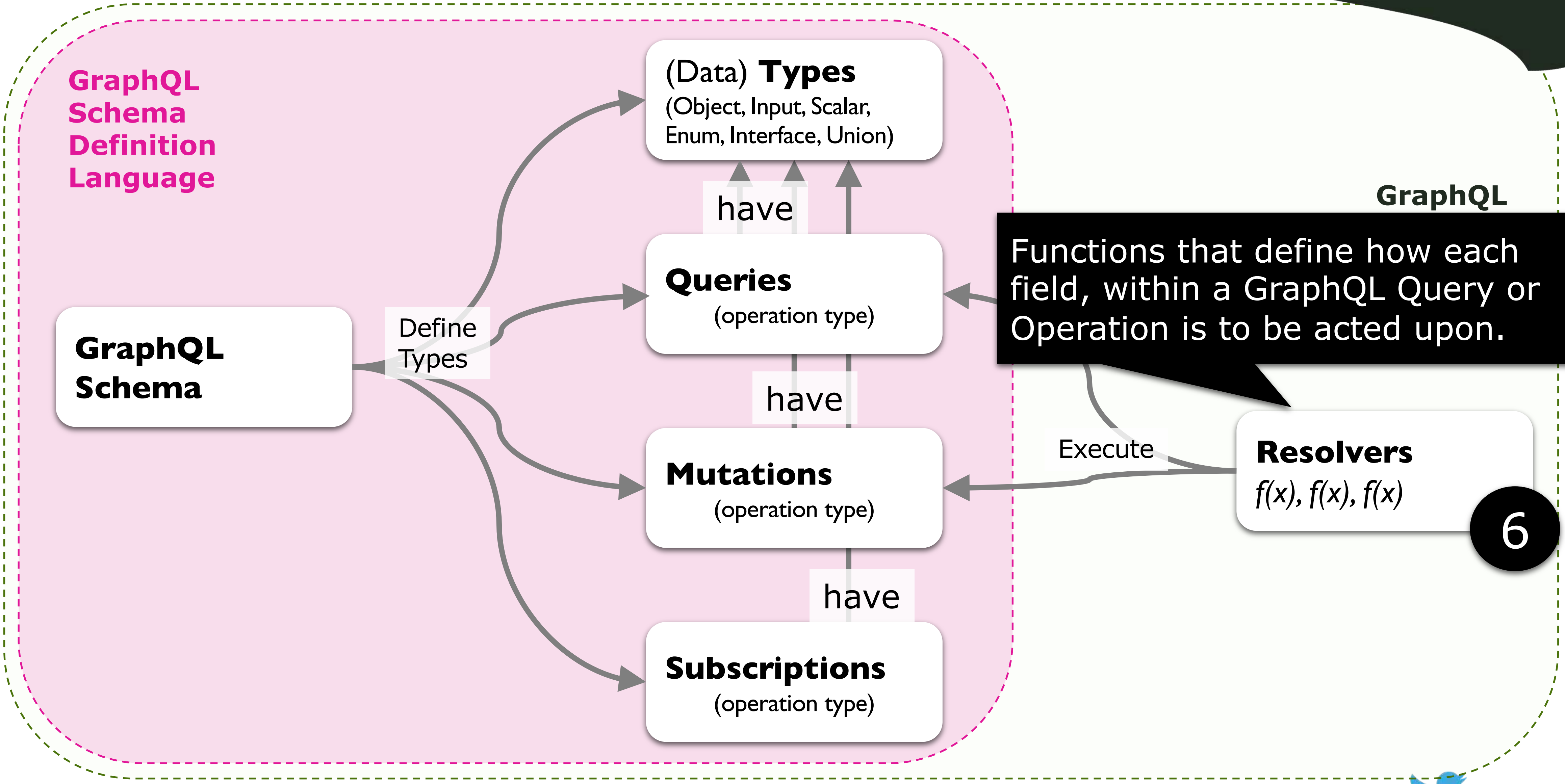
01

GraphQL – Anatomy



new Pub/sub system for near-realtime updates. Unlike queries or mutations, it can deliver more than one result via push.

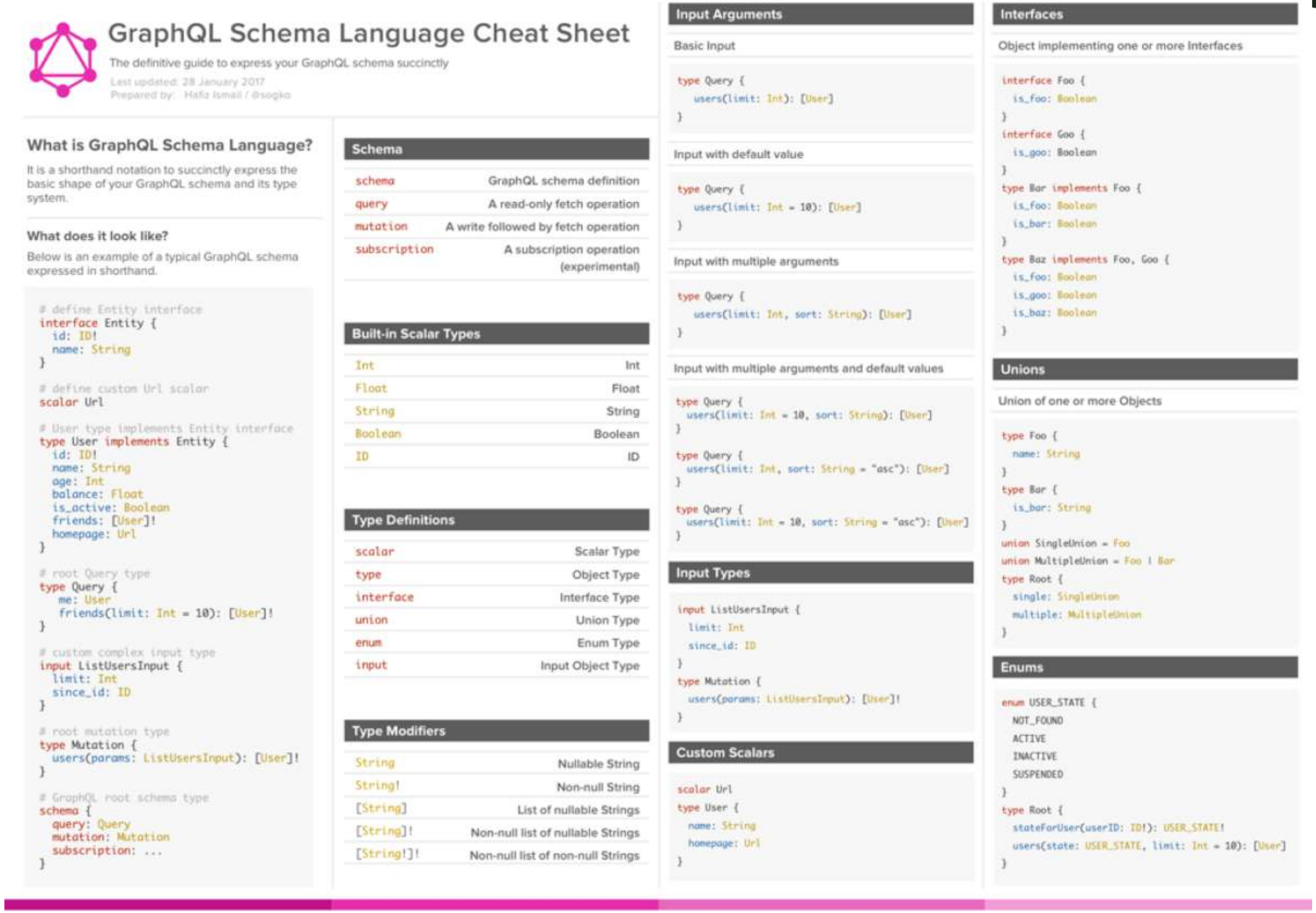
GraphQL – Anatomy



Functions that define how each field, within a GraphQL Query or Operation is to be acted upon.

@luisw19

GraphQL Schema Cheat Sheet



GraphQL Schema Language Cheat Sheet

The definitive guide to express your GraphQL schema succinctly

Last updated: 28 January 2017
Prepared by: Hafiz Ismail / @sogko

What is GraphQL Schema Language?

It is a shorthand notation to succinctly express the basic shape of your GraphQL schema and its type system.

What does it look like?

Below is an example of a typical GraphQL schema expressed in shorthand.

```
# define Entity interface
interface Entity {
  id: ID!
  name: String
}

# define custom Url scalar
scalar Url

# User type implements Entity interface
type User implements Entity {
  id: ID!
  name: String
  age: Int
  balance: Float
  is_active: Boolean
  friends: [User]!
  homepage: Url
}

# root Query type
type Query {
   me: User
   friends(limit: Int = 10): [User]!
}

# custom complex input type
input ListUsersInput {
  limit: Int
  since_id: ID
}

# root mutation type
type Mutation {
  users(params: ListUsersInput): [User]!
}

# GraphQL root schema type
schema {
  query: Query
  mutation: Mutation
  subscription: ...
}
```

Schema

schema	GraphQL schema definition
query	A read-only fetch operation
mutation	A write followed by fetch operation
subscription	A subscription operation (experimental)

Built-in Scalar Types

Int	Int
Float	Float
String	String
Boolean	Boolean
ID	ID

Type Definitions

scalar	Scalar Type
type	Object Type
interface	Interface Type
union	Union Type
enum	Enum Type
input	Input Object Type

Type Modifiers

String	Nullable String
String!	Non-null String
[String]	List of nullable Strings
[String]!	Non-null list of nullable Strings
[String!]!	Non-null list of non-null Strings

Input Arguments

Basic Input

```
type Query {
  users(limit: Int): [User]
}
```

Input with default value

```
type Query {
  users(limit: Int = 10): [User]
}
```

Input with multiple arguments

```
type Query {
  users(limit: Int, sort: String): [User]
}
```

Input with multiple arguments and default values

```
type Query {
  users(limit: Int = 10, sort: String): [User]
}
type Query {
  users(limit: Int, sort: String = "asc"): [User]
}
type Query {
  users(limit: Int = 10, sort: String = "asc"): [User]
}
```

Input Types

```
input ListUsersInput {
  limit: Int
  since_id: ID
}
type Mutation {
  users(params: ListUsersInput): [User]!
}
```

Custom Scalars

```
scalar Url
type User {
  name: String
  homepage: Url
}
```

Interfaces

Object implementing one or more Interfaces

```
interface Foo {
  is_foo: Boolean
}
interface Goo {
  is_goo: Boolean
}
type Bar implements Foo {
  is_foo: Boolean
  is_bar: Boolean
}
type Baz implements Foo, Goo {
  is_foo: Boolean
  is_goo: Boolean
  is_baz: Boolean
}
```

Unions

Union of one or more Objects

```
type Foo {
  name: String
}
type Bar {
  is_bar: String
}
union SingleUnion = Foo
union MultipleUnion = Foo | Bar
type Root {
  single: SingleUnion
  multiple: MultipleUnion
}
```

Enums

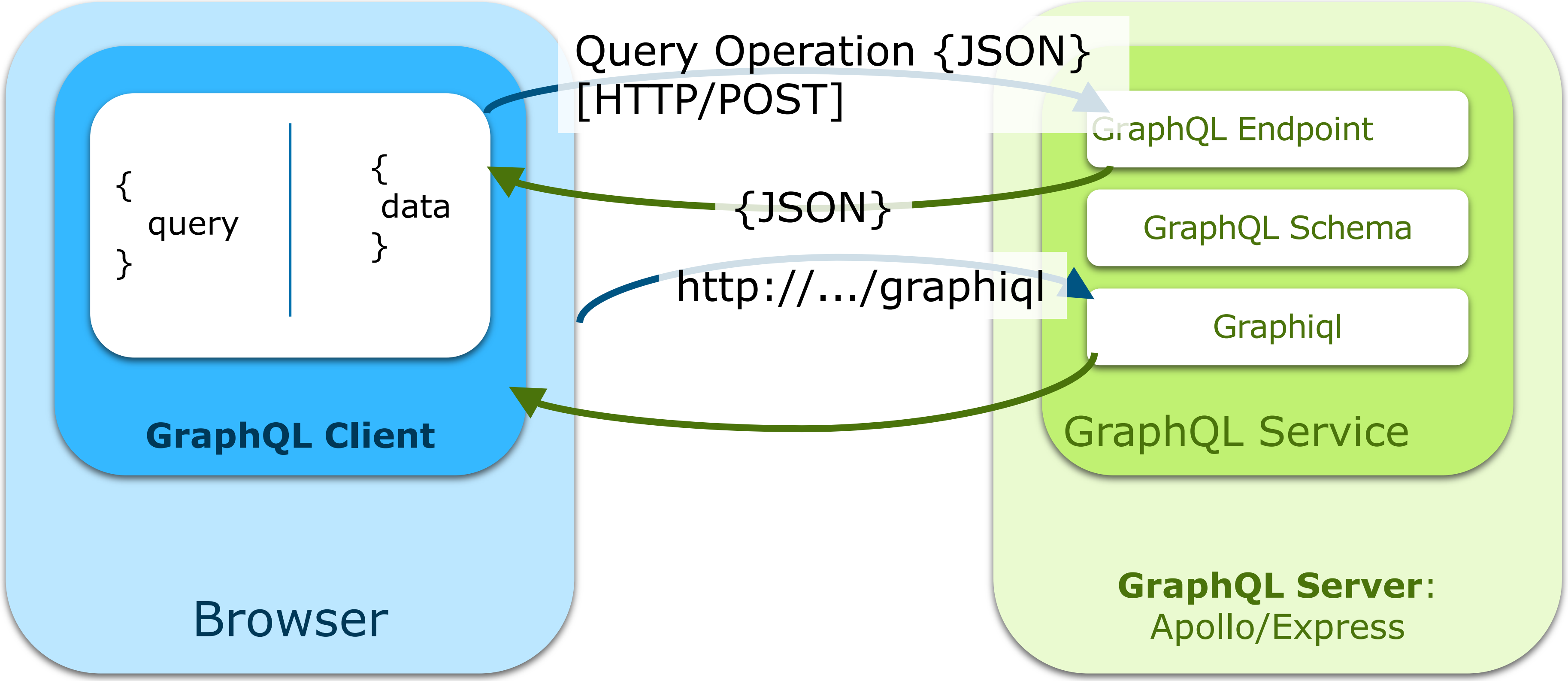
```
enum USER_STATE {
  NOT_FOUND
  ACTIVE
  INACTIVE
  SUSPENDED
}
type Root {
  stateForUser(userID: ID!): USER_STATE!
  users(state: USER_STATE, limit: Int = 10): [User]
}
```

https://github.com/sogko/graphql-schema-language-cheat-sheet

02

Simple GraphQL Query Demo (I) - Mock

https://github.com/luisw19/graphql-samples/tree/master/graphql-countries-part1

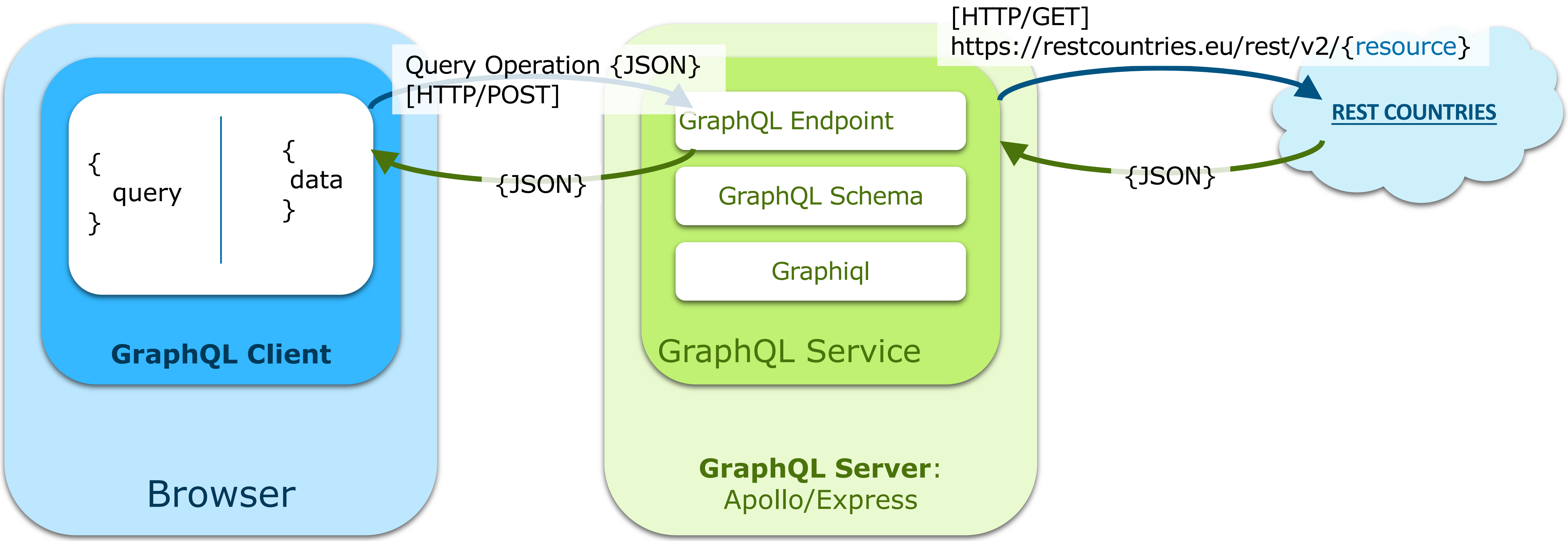


@luisw19

Apollo Express: https://github.com/apollographql/apollo-server

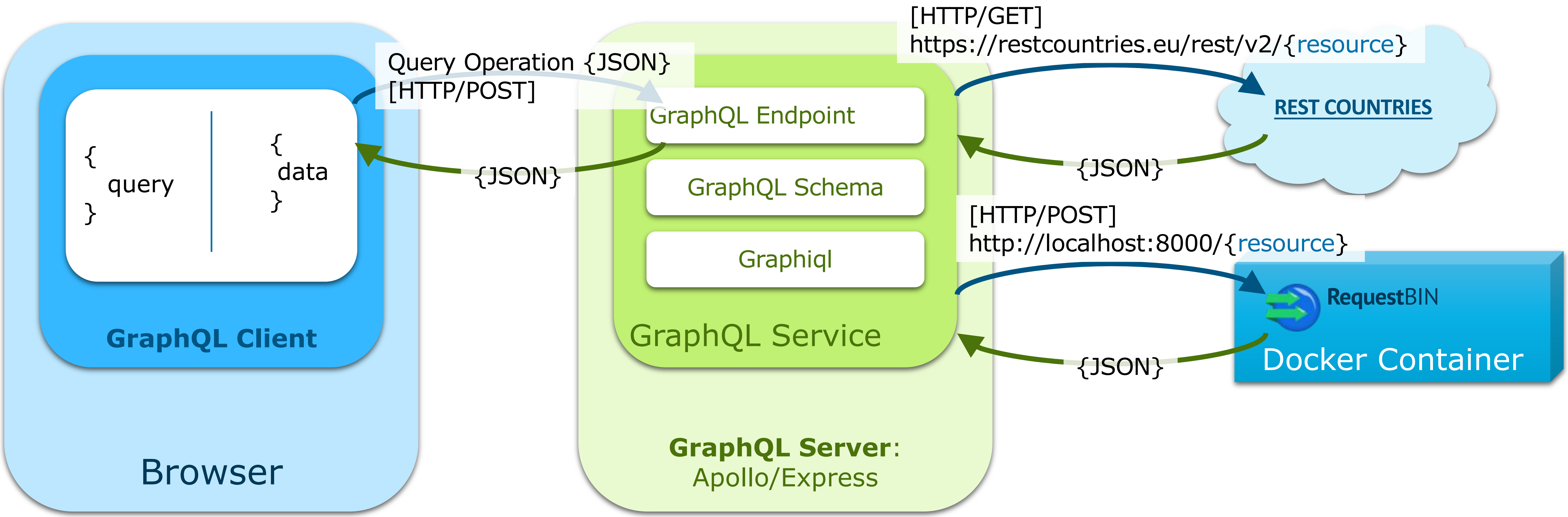
Simple GraphQL Query Demo (II) – REST

https://github.com/luisw19/graphql-samples/tree/master/graphql-countries-part2



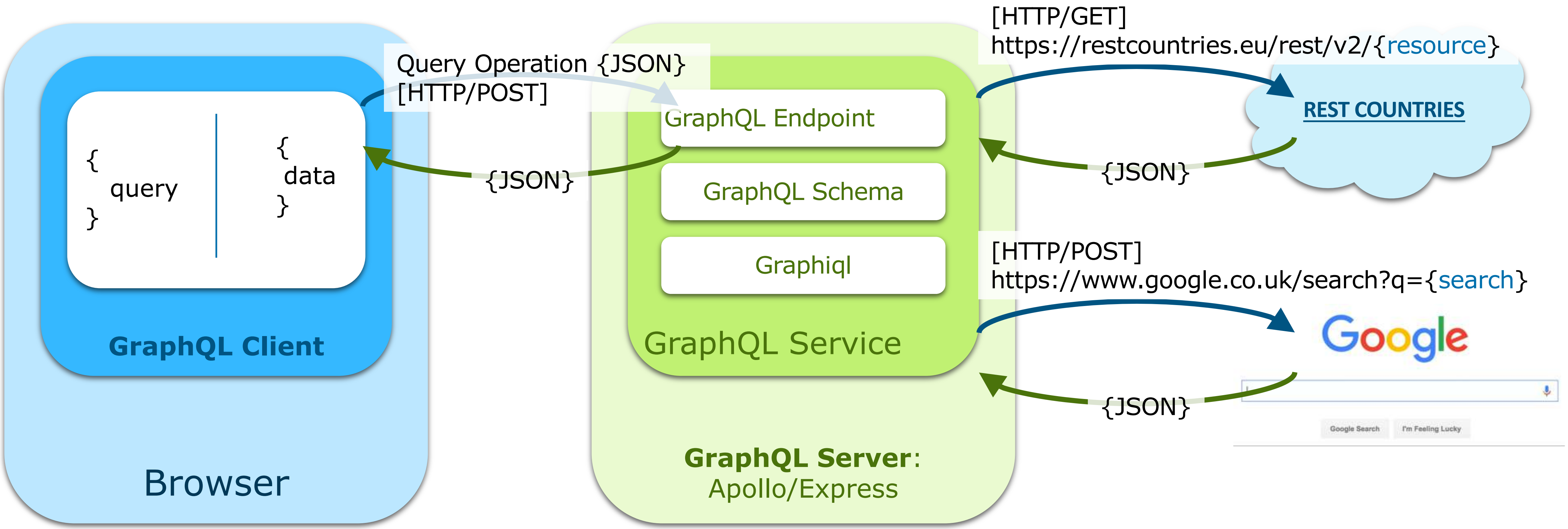
Simple GraphQL Query Demo (III) - Muta

https://github.com/luisw19/graphql-samples/tree/master/graphql-countries-part3



Simple GraphQL Query Demo (IV) – API C

Not merged yet… (soon)



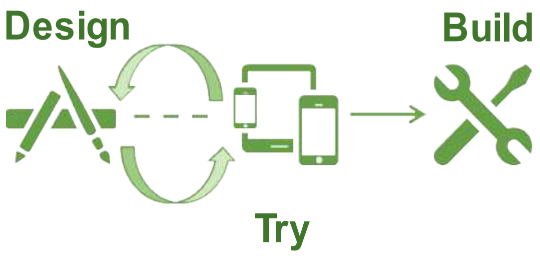
GraphQL vs REST

(++) Brilliant
(+) Good
(~) Neutral (it depends)
(-) Not very good
(--) It sucks!

GraphQL

REST

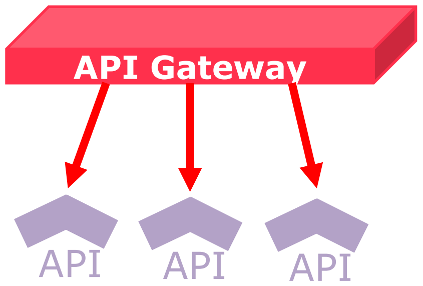
Developer Experience
(design and consume APIs)



(++) Usage: Best usage experience for developers (Graphiql is brilliant!)
(~) API-first design: Tooling evolving (build a service to mock).

(~) Usage: depends on the quality of API definition and documentation
(+) API-first design: good tools available (e.g. Apiary, Swagger Hub).

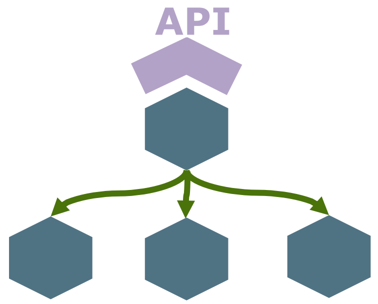
API Gateway
(API routing, security, policies)



(-) Existing Gateways have rich support for REST, not yet GraphQL - but could be used. Alternative is to use a GraphQL Service as API Gateway.

(++) API Gateways take away from REST endpoints common tasks (e.g. OAuth, API keys, throttling, security).

API Composition
(query data from multiple sources)



(++) Perfectly suited for API composition. Each field can be fetch (in parallel) from any source in a single query.

(--) The nature of REST makes it difficult to model resources that combine data structures from multiple sources. HATEOAS results in chattiness.

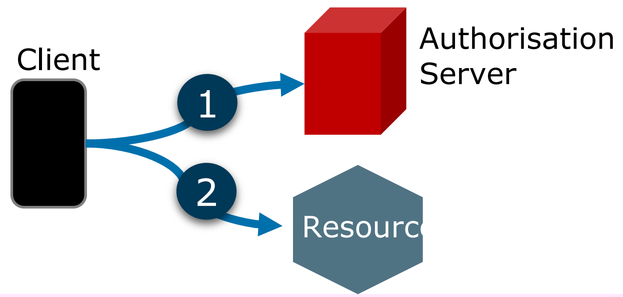
GraphQL vs REST

03

(++) Brilliant
(+) Good
(~) Neutral (it depends)
(-) Not very good
(--) It sucks!

GraphQL **REST**

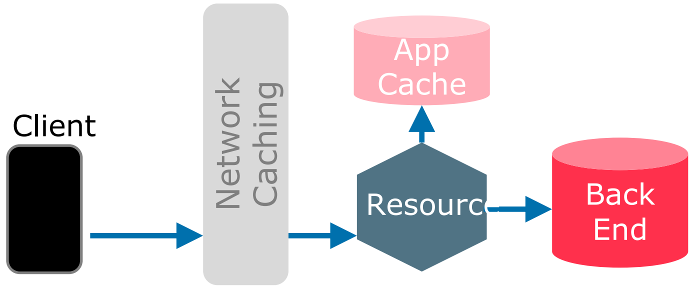
Authentication / Authorization



(-) Standards like OAuth, OpenID can be used however because all ops can be accessed by single URI, custom authorisation is typically required.

(++) Major standards (OAuth 2, OpenId) supported by API Gateways and frameworks.

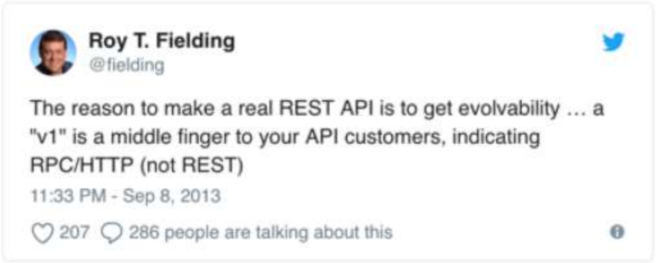
Caching



(--) Network: unsuitable as there is a common URL for all operations.
(+) Service: possible based on Object Type (even fields) and in mem cache like REDIS.

(++) Network: Caching is easy as each resource is a unique URI. Common tools can be used (e.g. CDNs).
(+) Service: It’s equally possible at service Level.

Versioning



Roy T. Fielding
@fielding

The reason to make a real REST API is to get evolvability … a "v1" is a middle finger to your API customers, indicating RPC/HTTP (not REST)

11:33 PM - Sep 8, 2013

207 286 people are talking about this

(++) Best practices are clear. Versioning should be avoided and tools are provided (e.g. deprecation of fields) for continuous evolution.

(-) Best practice less clear, in practice URI based versioning very popular although not encouraged.

@luisw19

GraphQL vs REST (completely subjective!)

GraphQL

+++++++ (7) → But it will only improve!

~ (1)

---- (4)

REST

++++++++ (8)

~~ (2)

--- (3)

Conclusions

1

Still early days but GraphQL has huge potential

GraphQL takes away many of the headaches of dealing with REST from a client side -specially around complex queries (against multiple sources). However tooling specially around API Design and API Gateways is still evolving. So bear this in mind when considering GraphQL.

2

GraphQL and REST can work nicely together

There are thousands of REST APIs (external and internal) and REST still is a viable and popular option. Instead of boiling the ocean, as Roy said, GraphQL is not necessarily a replacement for REST. As shown in this presentation both can be complementary and work together nicely.

3

There is no silver bullets –do your own research

There is tons of information available in the GraphQL Communities page. Explore it, learn about it and adopt it based on your own criteria and requirements. And hope this presentation helps in the process!

@luisw19

Resources

- GraphQL as an alternative approach to Rest recording at Devoxx’18 London

https://www.youtube.com/watch?v=hJOOdCPlXbU

- Github repository with the GraphQL tutorials

https://github.com/luisw19/graphql-samples

- Related articles:
  - GraphQL with Oracle Database and node-oracledb by Christopher Jones
    https://blogs.oracle.com/opal/demo%3a-graphql-with-node-oracledb
  - GraphQL+OracleDB by Steven B
    https://github.com/cloudsolutionhubs/oracledb-graphql-demo

People matter, results count.

About Capgemini

With more than 190,000 people, Capgemini is present in over 40 countries and celebrates its 50th Anniversary year in 2018. A global leader in consulting, technology and outsourcing services, the Group reported 2016 global revenues of EUR 12.5 billion. Together with its clients, Capgemini creates and delivers business, technology and digital solutions that fit their needs, enabling them to achieve innovation and competitiveness. A deeply multicultural organization, Capgemini has developed its own way of working, the Collaborative Business Experience™, and draws on Rightshore®, its worldwide delivery model.

Learn more about us at

www.capgemini.com