

# GraphQL

forget (the) REST?



Lakeside HackFest 2019  
Christian Schwendtner





## REST – Feelings



I'm lovin' it



Well, ...



*What else?*

# The Plan ...

REST Challenges - Why GraphQL?

GraphQL Language Features

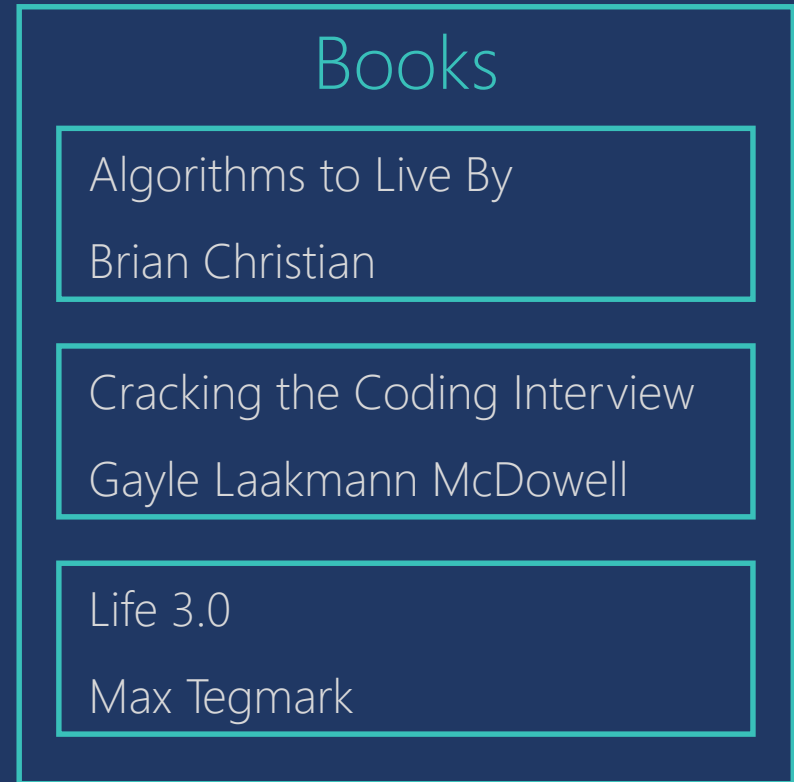
GraphQL Backend (using Node.js & **Apollo Server**)

Performance (DataLoader)

TypeScript



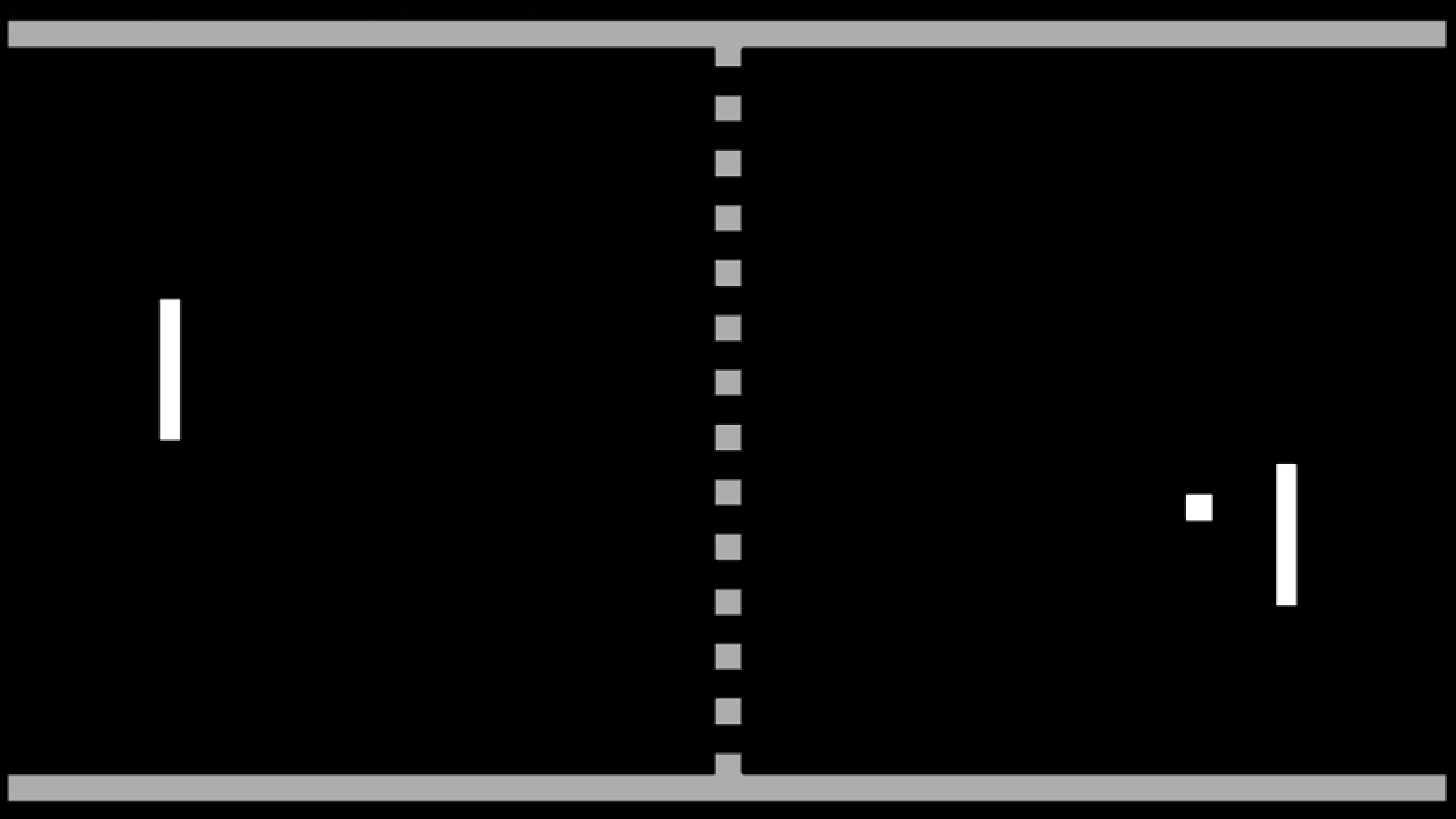
# The BookStore aka "Amazon (very) light"



# Interaction









<https://www.pexels.com/photo/adult-alcohol-blur-celebrate-290316/>



<https://www.pexels.com/photo/time-lapse-photography-of-water-bobbling-beside-lemon-fruit-161425/>

## “Custom Endpoints”?

GET /books\_overview

GET /books?dataset=books\_with\_authors

GET /books?dataset=books\_with\_authors\_with\_ratings

GET /books?dataset=books\_with\_authors\_with\_all\_i\_will\_ever\_need

GET /books?dataset=books\_with\_authors\_with\_all\_i\_will\_ever\_need2



# GraphQL

---

A query language for your API





# GraphQL is ...

Specification

<https://graphql.org>

Reference implementation

Schema (domain specific)

Just for once, let me look on you with my own eyes.  
*Anakin Skywalker*







# Fields

```
query {  
  books {  
    title  
    author {  
      firstName  
      lastName  
    }  
  }  
}
```

# Arguments

```
query {  
  books(first: 2) {  
    title  
    author {  
      firstName  
      lastName  
    }  
  }  
}
```

# Aliases

```
query {  
  books {  
    title  
    author {  
      firstName  
      lastName  
      smallPic: profilePic(size: SMALL)  
      largePic: profilePic(size: LARGE)  
    }  
  }  
}
```

```
"books": [  
  {  
    "title": "Algorithms to Live By: The Computer Science ...",  
    "author": {  
      "firstName": "Brian",  
      "lastName": "Christian",  
      "smallPic": "https://robohash.org/Christian.png?set=set2&size=100x100",  
      "largePic": "https://robohash.org/Christian.png?set=set2&size=200x200"  
    }  
  },  
]
```

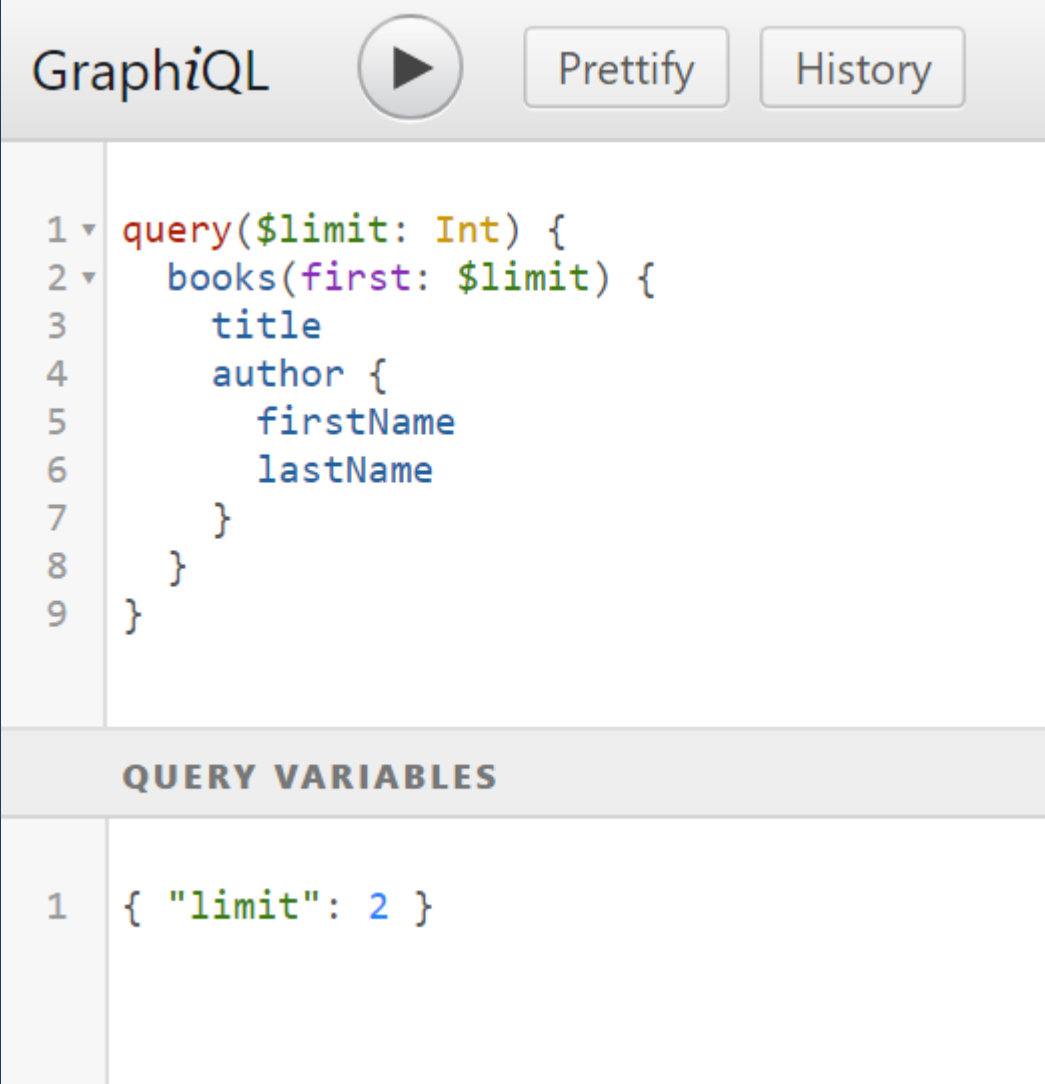
# Fragments

```
fragment authorFragment on Author {  
  firstName  
  lastName  
}
```

```
query {  
  books {  
    title  
    author {  
      ...authorFragment  
    }  
  }  
}
```

# Variables

```
query($limit: Int) {  
  books(first: $limit) {  
    title  
    author {  
      firstName  
      lastName  
    }  
  }  
}
```



The screenshot shows the GraphQL IDE interface. At the top, there is a header bar with the text "GraphiQL", a play button icon, and two buttons labeled "Prettify" and "History". Below the header, the main area is divided into two sections. The top section contains a query written in a monospaced font with syntax highlighting. The bottom section, titled "QUERY VARIABLES", contains a JSON object representing the variables for the query.

```
1 query($limit: Int) {  
2   books(first: $limit) {  
3     title  
4     author {  
5       firstName  
6       lastName  
7     }  
8   }  
9 }
```

**QUERY VARIABLES**

```
1 { "limit": 2 }
```

# Validation / Errors

GraphiQL



Prettify

History

< Docs

```
1 query {  
2   books {  
3     title  
4     author {  
5       fristName  
6       lastName  
7     }  
8   }  
9 }
```

```
{  
  "errors": [  
    {  
      "message": "Cannot query field \"fristName\" on type \"Author\".  
Did you mean \"firstName\" or \"lastName\"?",  
      "locations": [  
        {  
          "line": 5,  
          "column": 7  
        }  
      ]  
    }  
  ]  
}
```

# Introspection

GraphiQL

▶ Prettify History

```
1 query {
2   __schema {
3     types {
4       name
5     }
6   }
7 }
```

```
{
  "data": {
    "__schema": {
      "types": [
        {
          "name": "Query"
        },
        {
          "name": "Author"
        },
        {
          "name": "Int"
        },
        {
          "name": "String"
        },
        {
          "name": "Book"
        }
      ]
    }
  }
}
```

GraphiQL

▶ Prettify History

```
1 query {
2   __type(name: "Author") {
3     name
4     description
5     fields {
6       name
7       description
8       |
9     }
10  }
11 }
```

name
description
args
type
isDeprecated
deprecationReason
Self descriptive.

```
{
  "data": {
    "__type": {
      "name": "Author",
      "description": "an author",
      "fields": [
        {
          "name": "id",
          "description": "the id of the author"
        },
        {
          "name": "firstName",
          "description": "the firstname of the author"
        },
        {
          "name": "lastName",
          "description": "the lastname of the author"
        }
      ]
    }
  }
}
```



# Mutations

```
mutation {  
  createAuthor(firstName: "Hugo", lastName: "Meier") {  
    id  
    firstName  
    lastName  
  }  
}
```

*Schema*

May the force be with you



# Schema

```
type Book {  
  id: Int!  
  title: String  
  author: Author  
}
```

```
type Author {  
  id: Int!  
  firstName: String  
  lastName: String  
  books: [Book]  
}
```

```
type Query {  
  books(first: Int): [Book]  
  book(id: Int!): Book  
  authors: [Author]  
}
```

```
type Mutation {  
  createAuthor(  
    firstName: String!,  
    lastName: String!): Book  
}
```

# Schema

## Type Definitions

`type`

`interface`

`union`

`enum`

`scalar`

`input`

## Type Modifiers

`String`

`String!`

`[String]`

`[String]!`

`[String!]!`

## Built-in Scalar Types

`Int`

`Float`

`String`

`Boolean`

`ID`

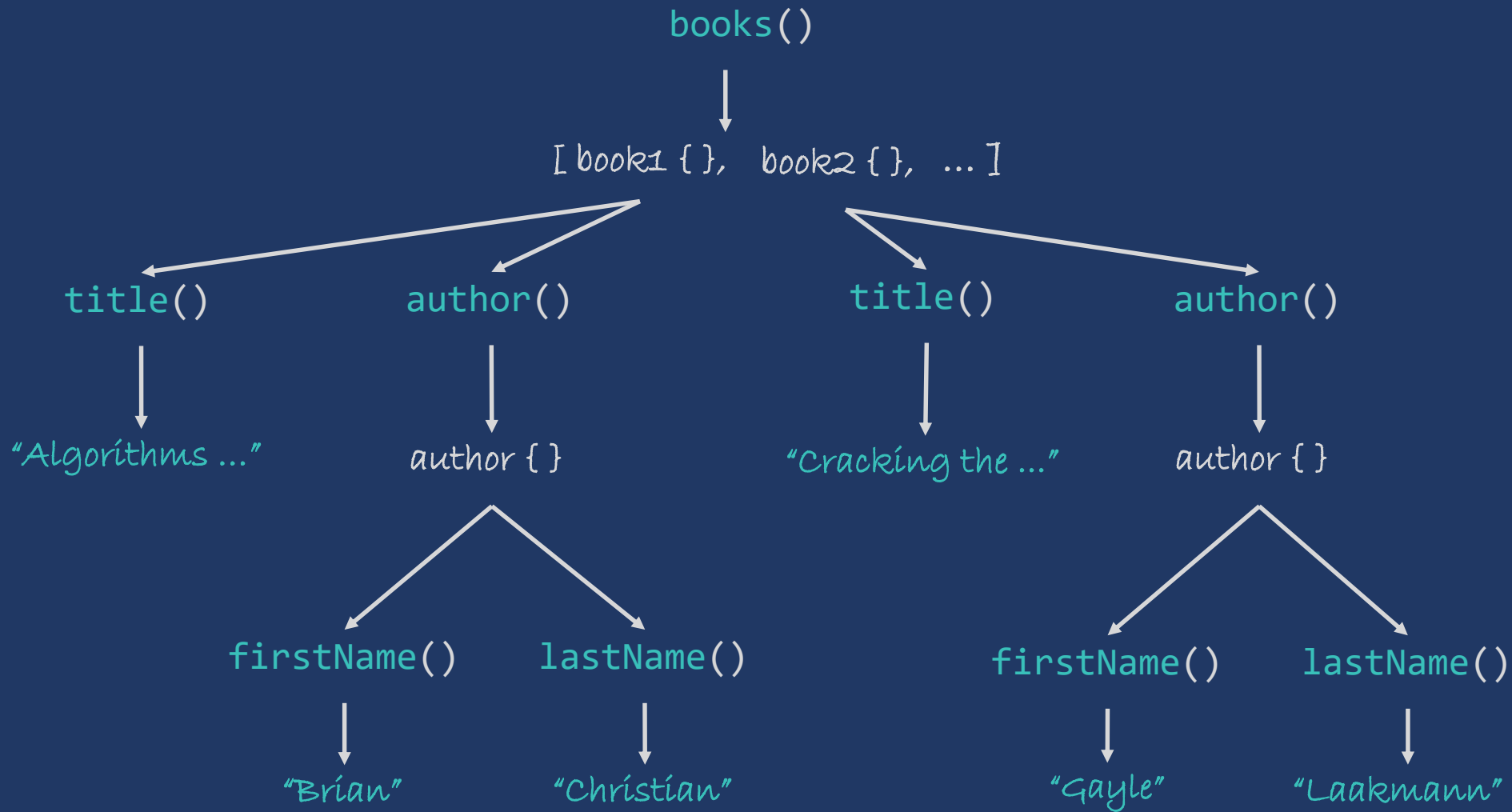
<https://github.com/sogko/graphql-schema-language-cheat-sheet>

# Resolvers

QUERY

BOOK

AUTHOR



```
query {  
  books {
```

```
    title  
    author {
```

```
      firstName  
      lastName
```

```
    }  
  }  
}
```

Let`s start ...

Apollo Server

<https://www.apollographql.com/docs/apollo-server/>

```
npm install apollo-server graphql
```

# Execution

```
query {  
  books {  
    title  
    author {  
      firstName  
      lastName  
    }  
  }  
}
```

```
db.getAllBooks()
```

```
db.getAuthorForBook(1)
```

```
db.getAuthorForBook(2)
```

```
db.getAuthorForBook(3)
```

```
db.getAuthorForBook(4)
```

```
db.getAuthorForBook(5)
```

```
db.getAuthorForBook(6)
```

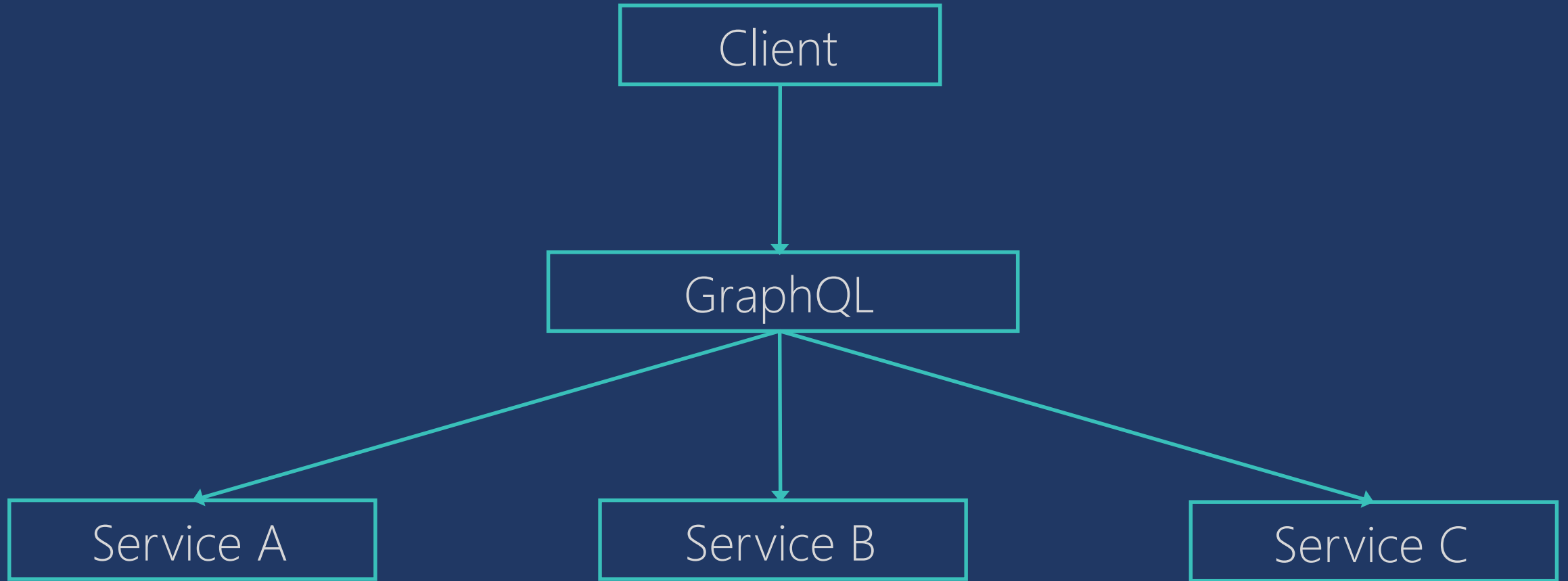
Caching

Batching

DataLoader

<https://github.com/facebook/dataloader>

# GraphQL as Gateway





REST

GraphQL







Christian Schwendtner



 @CSchwendtner



@CSchwendtner