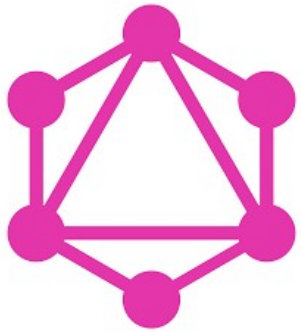
Introduction to GraphQL

Introduction to GraphQL

This lesson covers

- Understanding GraphQL and the design concepts behind it
- How GraphQL differs from alternatives like REST APIs
- Understanding the language used by GraphQL clients and services
- Understanding the advantages and disadvantages of GraphQL

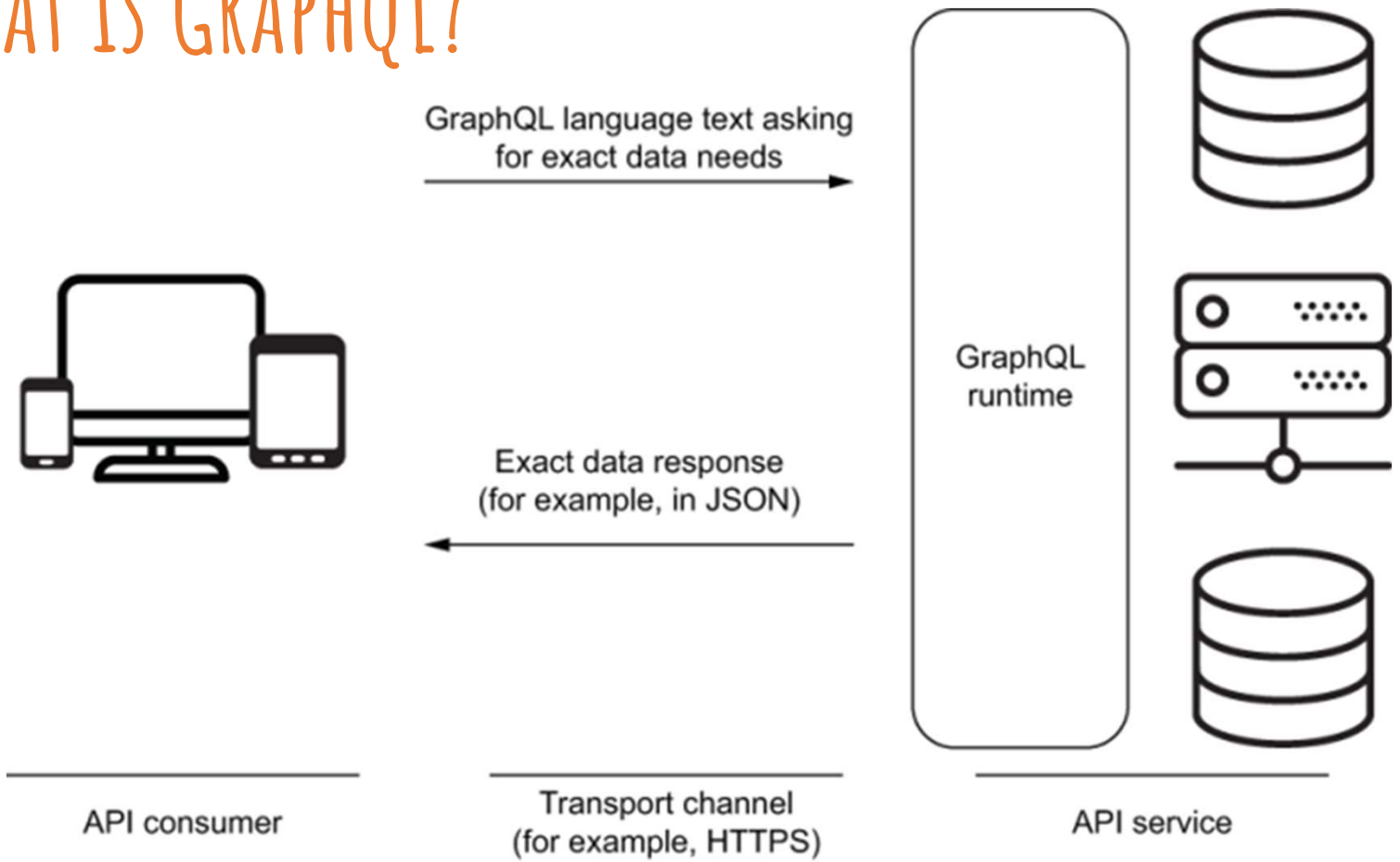
What is GraphQL?



- The word graph in GraphQL comes from the fact that the best way to represent data in the real world is with a graph-like data structure.

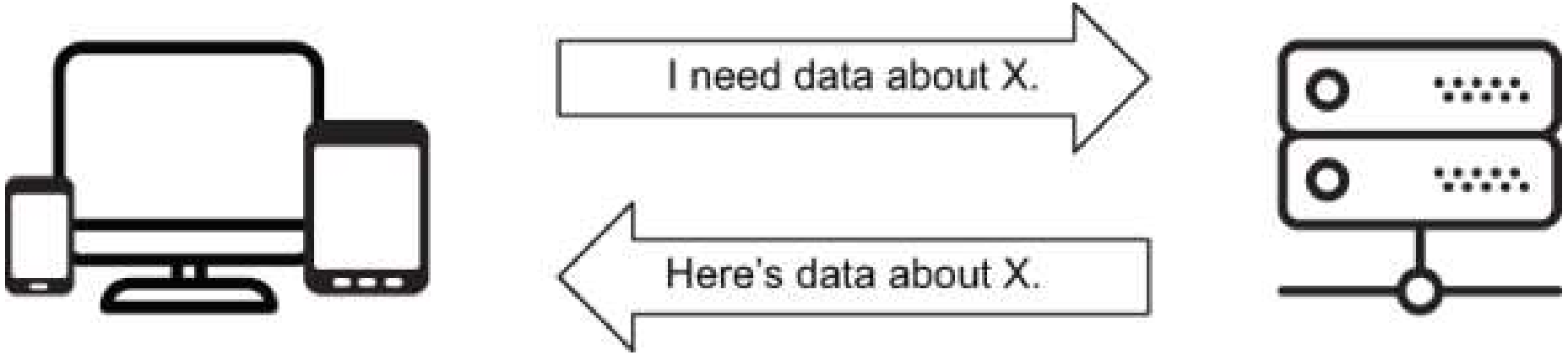
- If you analyze any data model, big or small, you'll always find it to be a graph of objects with many relations between them.

What is GraphQL?



The big picture

- For example, an API can enable the communication that needs to happen between a web client and a database server.



The big picture

- For example, if we have a table of data about a company's employees, the following is an example SQL statement to read data about the employees in one department.

```
SELECT id, first_name, last_name, email,
birth_date, hire_date
FROM employees
WHERE department = 'ENGINEERING'
```

The big picture

- Here is another example SQL statement that inserts data for a new employee.

```
INSERT INTO employees (first_name, last_name,
email, birth_date, hire_date)
VALUES ('Jane', 'Doe', 'jane@doe.name',
'01/01/1990', '01/01/2020')
```

The big picture

```
{
  "data": {
    "employee":{
      "id": 42,
      "name": "Jane Doe",
      "email": "jane@doe.name",
      "birthDate": "01/01/1990",
      "hireDate": "01/01/2020"
    }
  }
}
```

The big picture

```
{
  "select": {
    "fields": ["name", "email", "birthDate",
"hireDate"],
    "from": "employees",
    "where": {
      "id": {
        "equals": 42
      }
    }
  }
}
```

The big picture

```
{
  employee(id: 42) {
    name
    email
    birthDate
    hireDate
  }
}
```

GraphQL is a language

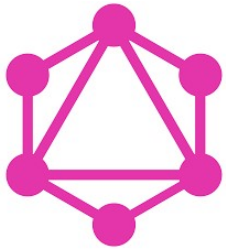
- A query language like GraphQL (or SQL) is different from programming languages like JavaScript and Python.
- You cannot use the GraphQL language to create user interfaces or perform complex computations.
- Query languages have more specific use cases, and they often require the use of programming languages to make them work.

GraphQL is a language

- For example, here's a hypothetical single GraphQL query that represents both of John's questions to Jane.

```
{
  timeLightNeedsToTravel(toPlanet: "Earth") {
    fromTheSun: from(star: "Sun")
    fromTheMoon: from(moon: "Moon")
  }
}
```

GraphQL is a service



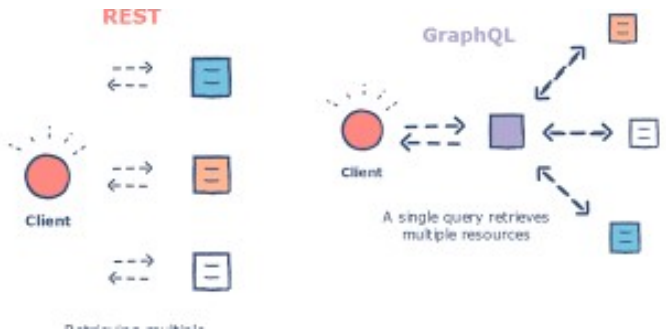
A GraphQL service can be written in any programming language, and it can be conceptually split into two major parts, structure and behavior:

- The structure is defined with a strongly typed schema.
- The behavior is naturally implemented with functions that in the GraphQL world are called resolver functions.

GraphQL is a service

- To understand how resolvers work, let's take the query in listing 1.5 (simplified) and assume a client sent it to a GraphQL service.

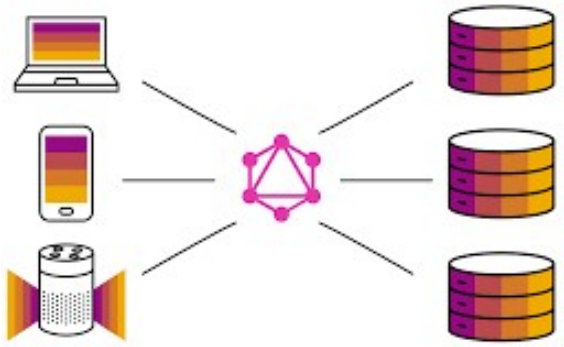
```
query {
  employee(id: 42) {
    name
    email
  }
}
```



GraphQL is a service

- Here's an example to represent the Employee type using GraphQL's schema language.

```
type Employee(id: Int!) {
  name: String!
  email: String!
}
```



GraphQL is a service

- The employee field's resolver function for employee #42 might return an object like the following.

```
{
  "id": 42,
  "first_name": "Jane",
  "last_name": "Doe",
  "email": "jane@doe.name",
  "birth_date": "01/01/1990",
  "hire_date": "01/01/2020"
}
```

GraphQL is a service

- Let's say we have the following (JavaScript) functions representing the server resolver functions for the name and email fields:

```
// Resolver functions
const name => (source) => `${source.first_name}
${source.last_name}`;
const email => (source) => source.email;
```

GraphQL is a service

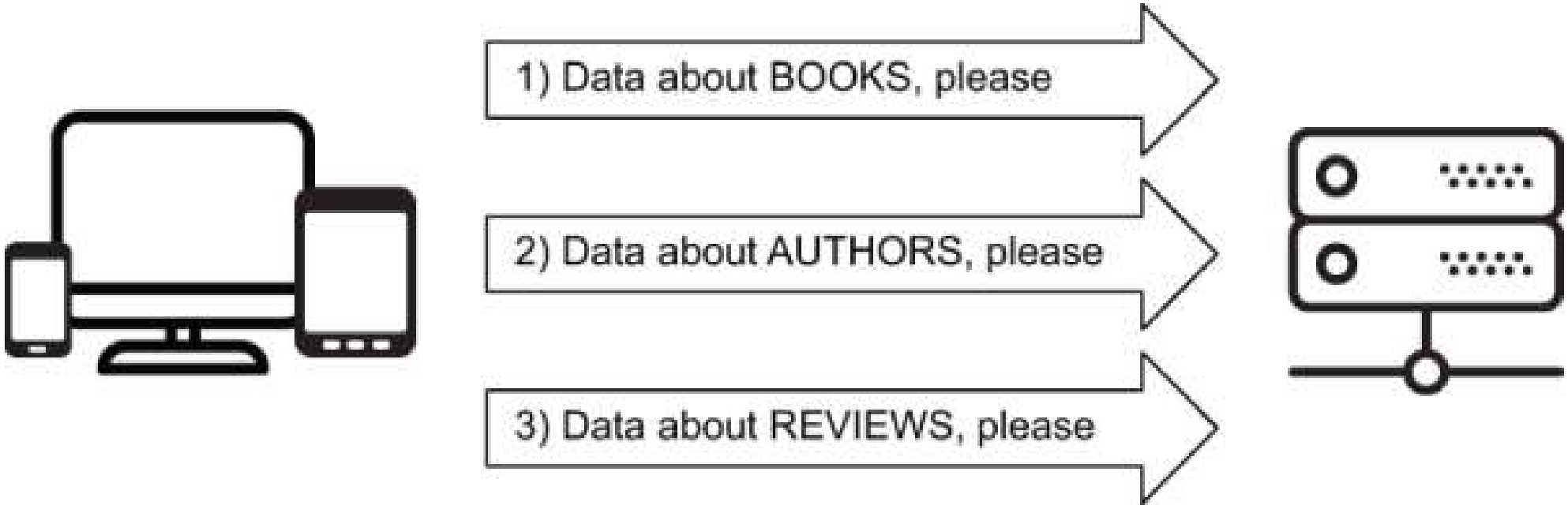
- The GraphQL service uses all the responses of these three resolver functions to put together the following single response for the query in listing 1.7.

```
{
  data: {
    employee: {
      name: 'Jane Doe',
      email: 'jane@doe.name'
    }
  }
}
```

Why GraphQL?

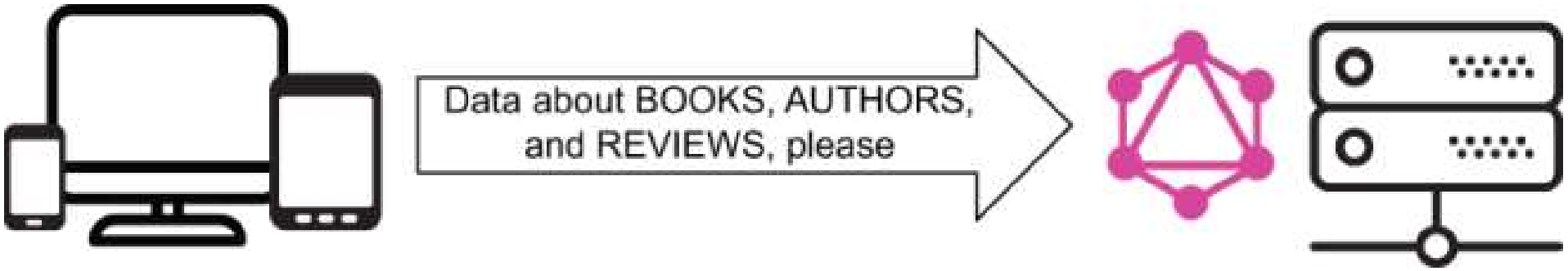
- GraphQL is not the only-or even the first-technology to encourage creating efficient data APIs.
- You can use a JSON-based API with a custom query language or implement the Open Data Protocol (OData) on top of a REST API.
- GraphQL provides comprehensive standards and structures to implement API features in maintainable and scalable ways.

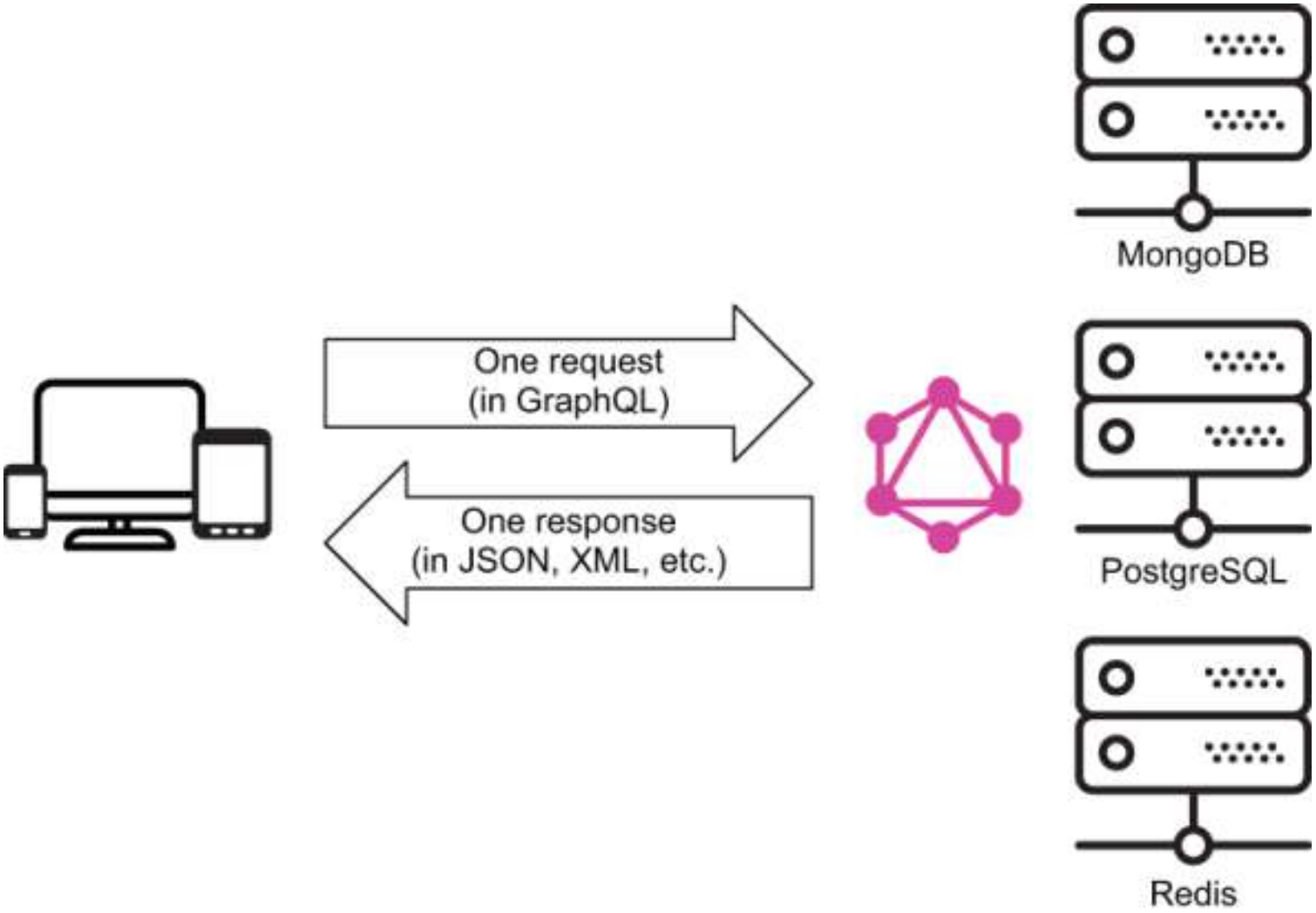
Why GraphQL?


1) Data about BOOKS, please
2) Data about AUTHORS, please
3) Data about REVIEWS, please

Why GraphQL?

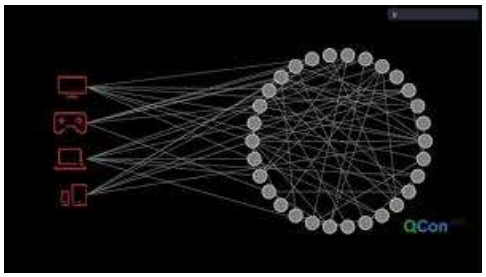
- You can customize a REST-based API to provide one exact endpoint per view, but that's not the norm.
- You will have to implement it without a standard guide.




One request
(in GraphQL)
One response
(in JSON, XML, etc.)
MongoDB
PostgreSQL
Redis

What about REST APIs?

- GraphQL APIs are often compared to REST APIs because the latter have been the most popular choice for data APIs demanded by web and mobile applications.
- GraphQL provides a technological alternative to REST APIS



The GraphQL way

To see the GraphQL way of solving the REST API problems we have talked about, you need to understand the concepts and design decisions behind GraphQL. Let's review the major ones.

- The typed Graph schema
- The declarative language
- The single endpoint and client language
- The simple versioning

REST APIs and GraphQL APIs in action

- The JSON data for this view could be something like the following.


```
{
  "data": {
    "person": {
      "name": "Darth Vader",
      "birthYear": "41.9BBY",
      "planet": {
        "name": "Tatooine"
      },
      "films": [
        { "title": "A New Hope" },
        { "title": "The Empire Strikes Back" },
        { "title": "Return of the Jedi" },
        { "title": "Revenge of the Sith" }
      ]
    }
  }
}
```

REST APIs and GraphQL APIs in action

```
// The Container Component:
<PersonProfile
person={data.person}></PersonProfile>
// The PersonProfile Component:
Name: {data.person.name}
Birth Year: {data.person.birthYear}
Planet: {data.person.planet.name}
Films: {data.person.films.map(film =>
film.title)}
```

REST APIs and GraphQL APIs in action

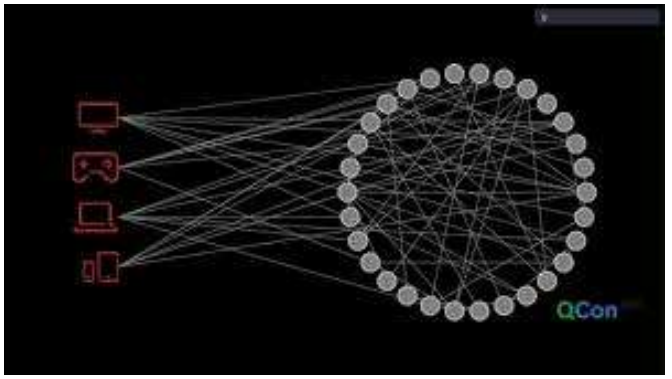
- You need a Star Wars character's information.
- Assuming that you know that character's ID, a REST API is expected to expose that information with an endpoint like this:

```
GET - /people/{id}
```

REST APIs and GraphQL APIs in action

- The JSON response for this request could be something like the following:

```
{
  "name": "Darth Vader",
  "birthYear": "41.9BBY",
  "planetId": 1
  "filmIds": [1, 2, 3, 6],
   .-.-.
}
```



In English	In GraphQL

The view needs:
 	
a person's name,
 	
birth year,
 	
planet's name,
 	
and the titles of all their films.
	{
	  person(ID: ·····) {
	  name
	  birthYear
	  planet {
	  name
	  }
	  films {
	  title
	  }
	  }
	}

GraphQL query (question)	Needed JSON (answer)

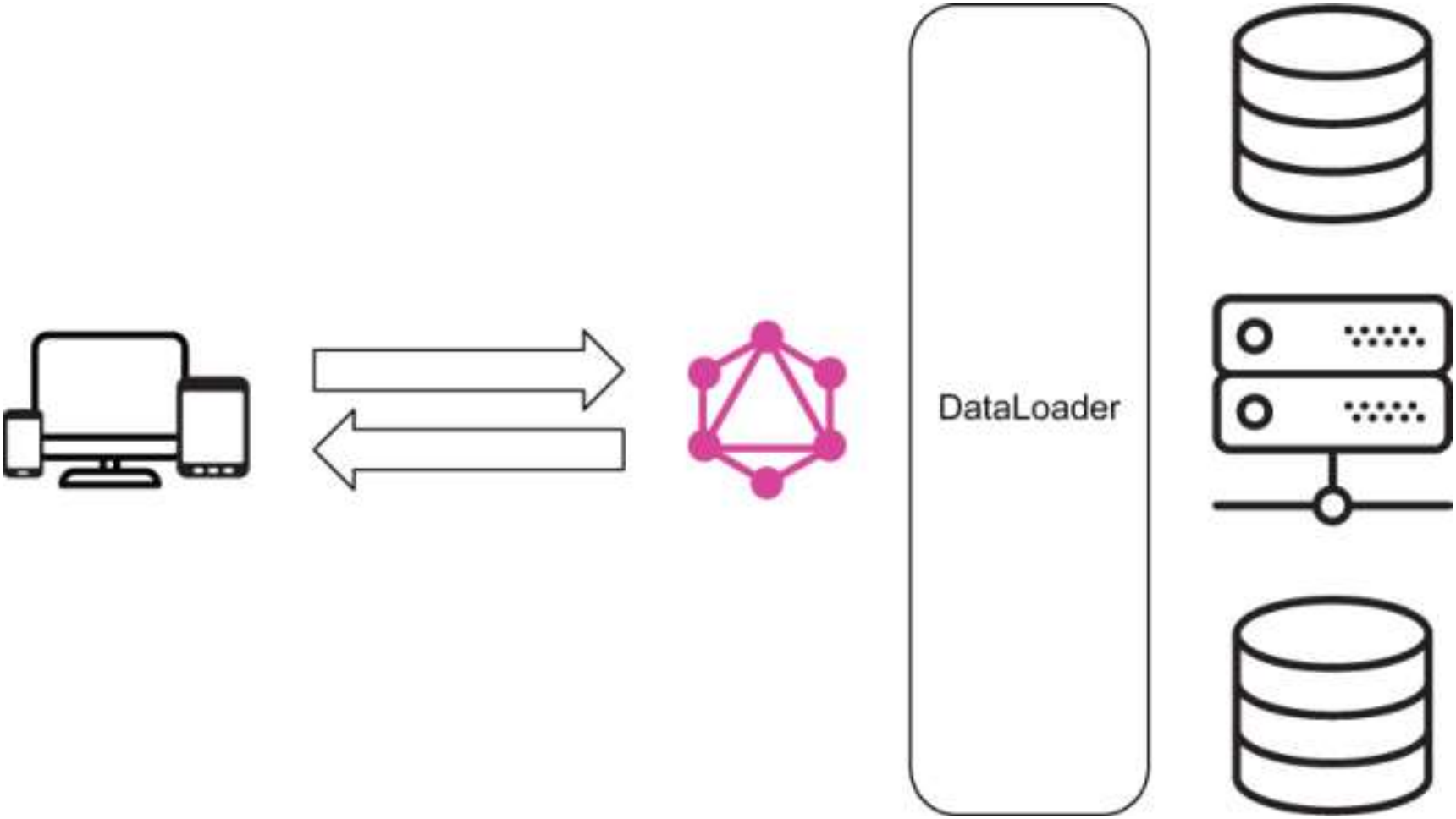
`{`
`  person(ID: -----) {`
`  name`
`  birthYear`
`  planet {`
`  name`
`  }`
`  films {`
`  title`
`  }`
`  }`
`}`
	`{`
	`  "data": {`
	`  "person": {`
	`  "name": "Darth Vader",`
	`  "birthYear": "41.9BBY",`
	`  "planet": {`
	`  "name": "Tatooine"`
	`  },`
	`  "films": [`
	`  { "title": "A New Hope" },`
	`  { "title": "The Empire Strikes Back" },`
	`  { "title": "Return of the Jedi" },`
	`  { "title": "Revenge of the Sith" }`
	`  ]`
	`  }`
	`  }`
	`}`

```
{
  person(personID: 4) {
    name
    birthYear
    homeworld {
      name
    }
    filmConnection {
      films {
        title
      }
    }
  }
}
```

GraphQL problems

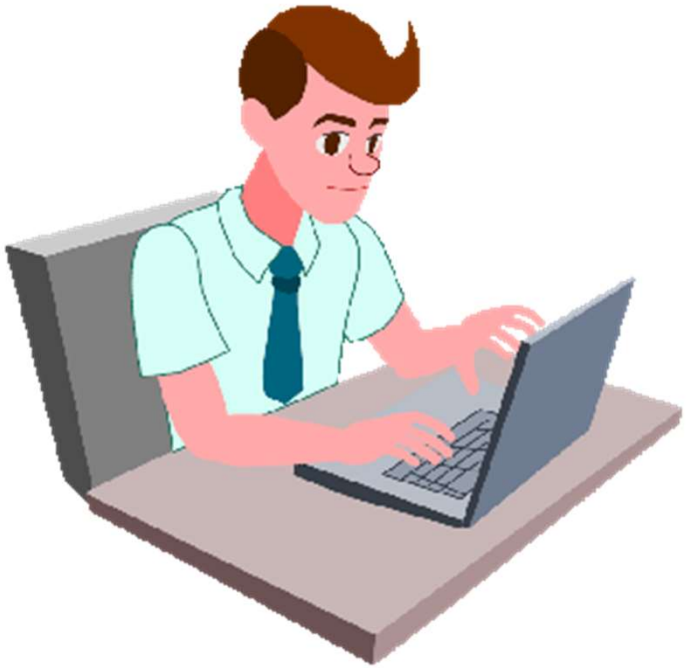
Perfect solutions are fairy tales. The flexibility that GraphQL introduces opens a door to some clear issues and concerns.

- Security
- Caching and optimizing
- Learning curve


DataLoader

Summary

- The best way to represent data in the real world is with a graph data structure. A data model is a graph of related objects. GraphQL embraces this fact.
- A GraphQL system has two primary components: the query language, which can be used by consumers of data APIs to request their exact data needs; and the runtime layer on the backend, which publishes a public schema describing the capabilities and requirements of data models



"Complete Lab"