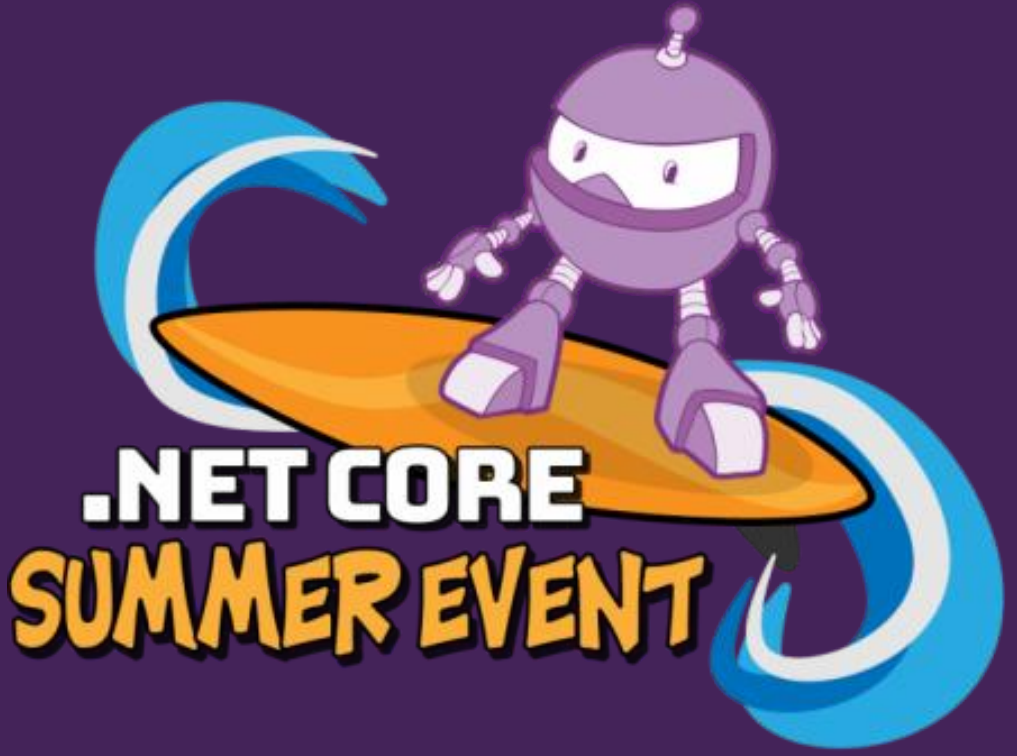


Beyond REST

with GraphQL in .NET Core

Irina Scurtu

@irina_scurtu

Irina Scurtu

@irina_scurtu

- **Romania Based**
- Software Architect @Endava
- Organizer of DotNetIasi user group
- I teach .NET
- Blog: https://Irina.codes

Agenda

- REST
- GraphQL
- GraphQL in .Net
- REST or GraphQL

REST Constraints

Client-Server

Stateless

Caching

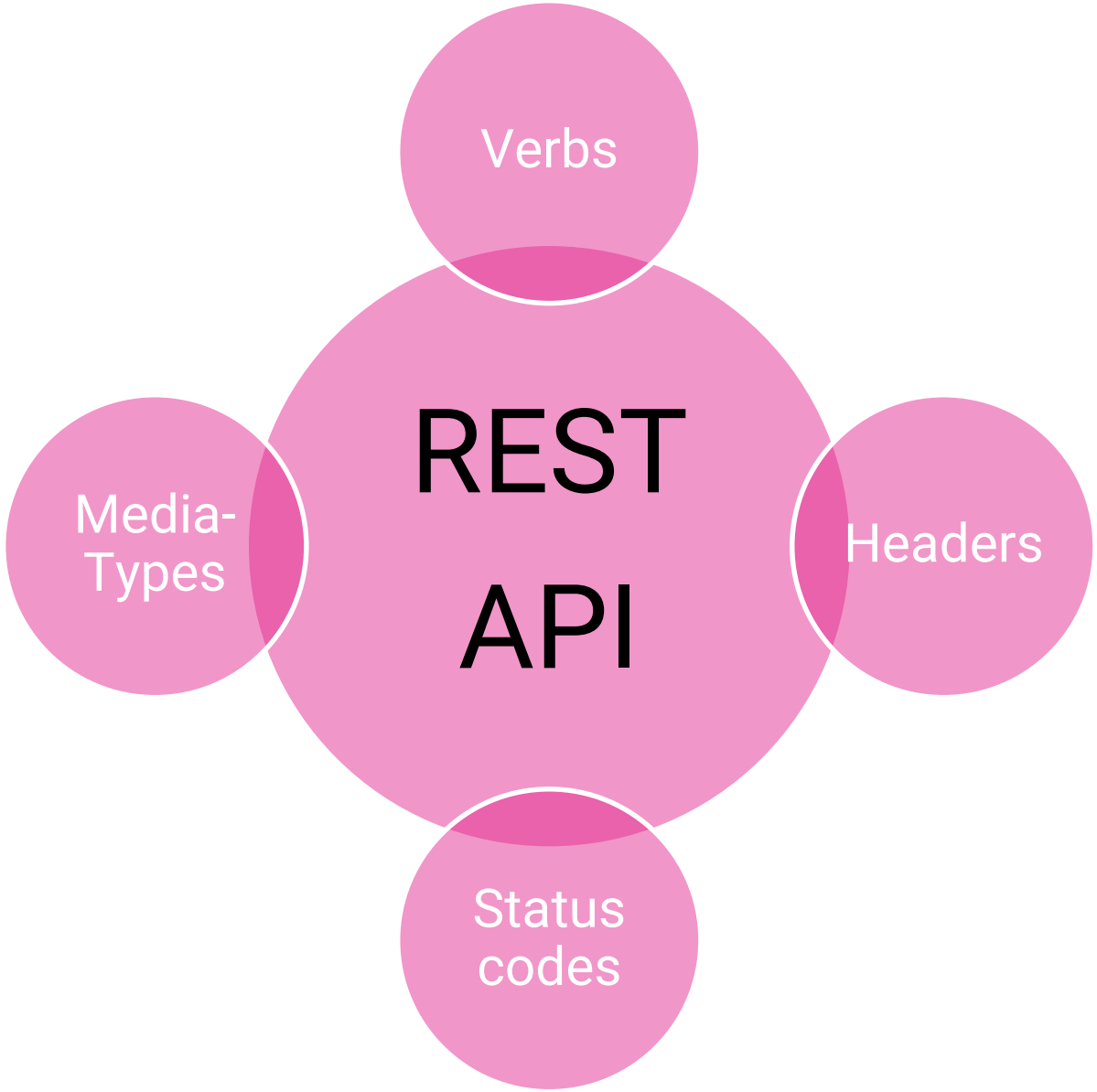
Uniform interface

REST Constraints

Uniform interface

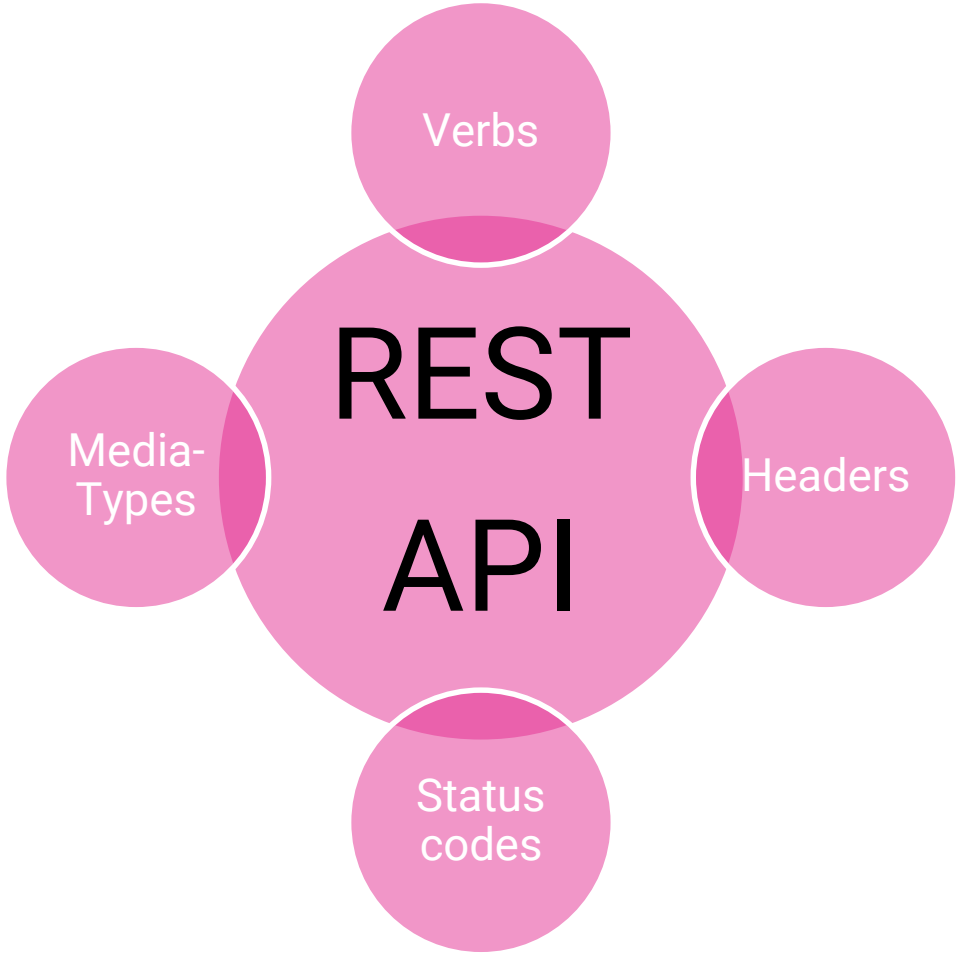
- Identification of resources
- Representation of resources
- Self descriptive messages
- HATEOAS

REST API



REST API

- Intuitive endpoints
- Meaning
- HTTP power
- Responds to different needs



Representations of the same resource

- Headers
- Content-Negotiation
- Status Codes

200

JSON

Representations of the same resource

Headers

Content-Negotiation

Status Codes

200

XML

Representations of the same resource

Headers

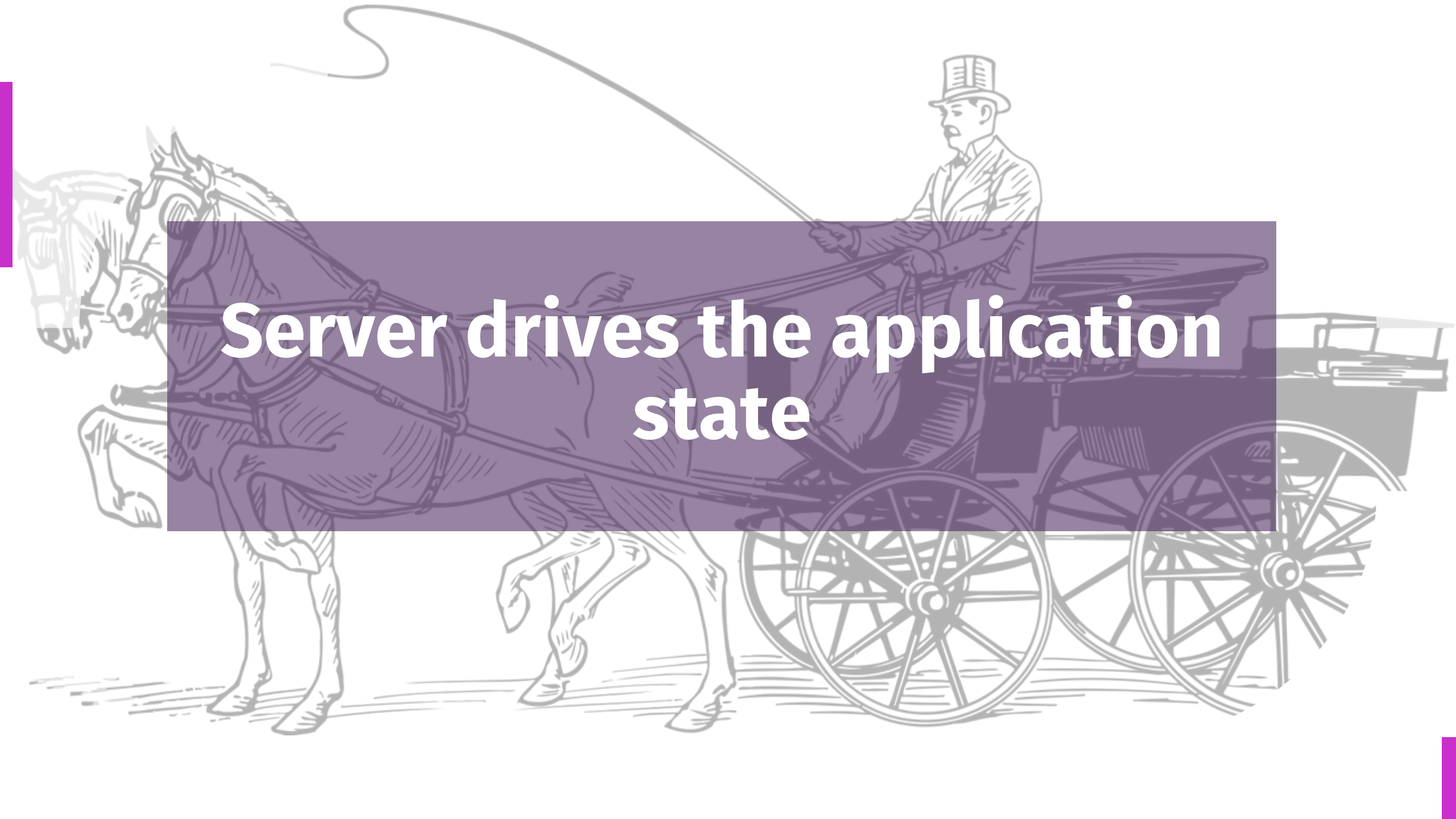
Content-Negotiation

Status Codes

415
no content

REST API benefits

- Scalable
- Server & client can evolve independently
- Discoverable
- Evolvable



Server drives the application state

Can evolve independently

Versioning

/v1

REST APIs....

Rare

Treated superficially

You need discipline

Constant design

Proven results

Evolvability

“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)”


Chatty

Exact fetching

overfetching

underfetching

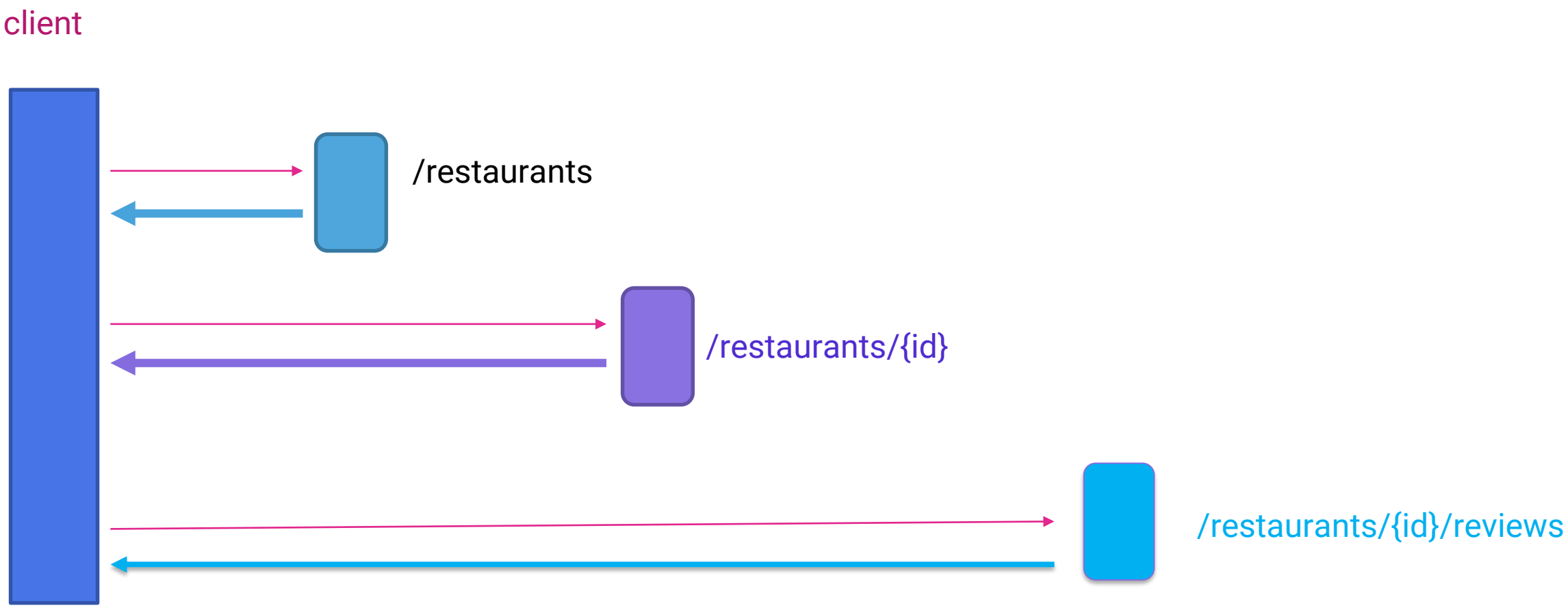
Overfetching

- Get more data than you use for your client
- Can be solved by continuously designing and trimming your API
  - IF YOU’re LUCKY

Underfetching

- Forces you to make another call to get some data
- “get that, and that, and also that”

REST


client
/restaurants
/restaurants/{id}
/restaurants/{id}/reviews

REST-ish Apis

Bad naming

Resource based?

Accept vs Content-Type

Most of us did this

Almost fully use HTTP

Status codes

usage of headers?

No HATEOAS

Self-descriptive messages

What is GraphQL?

“a query language that solves the issue with overfetching & underfetching”


GitHub


yelp


PayPal


KLM

Why this hype?

Exact fetching

overfetching

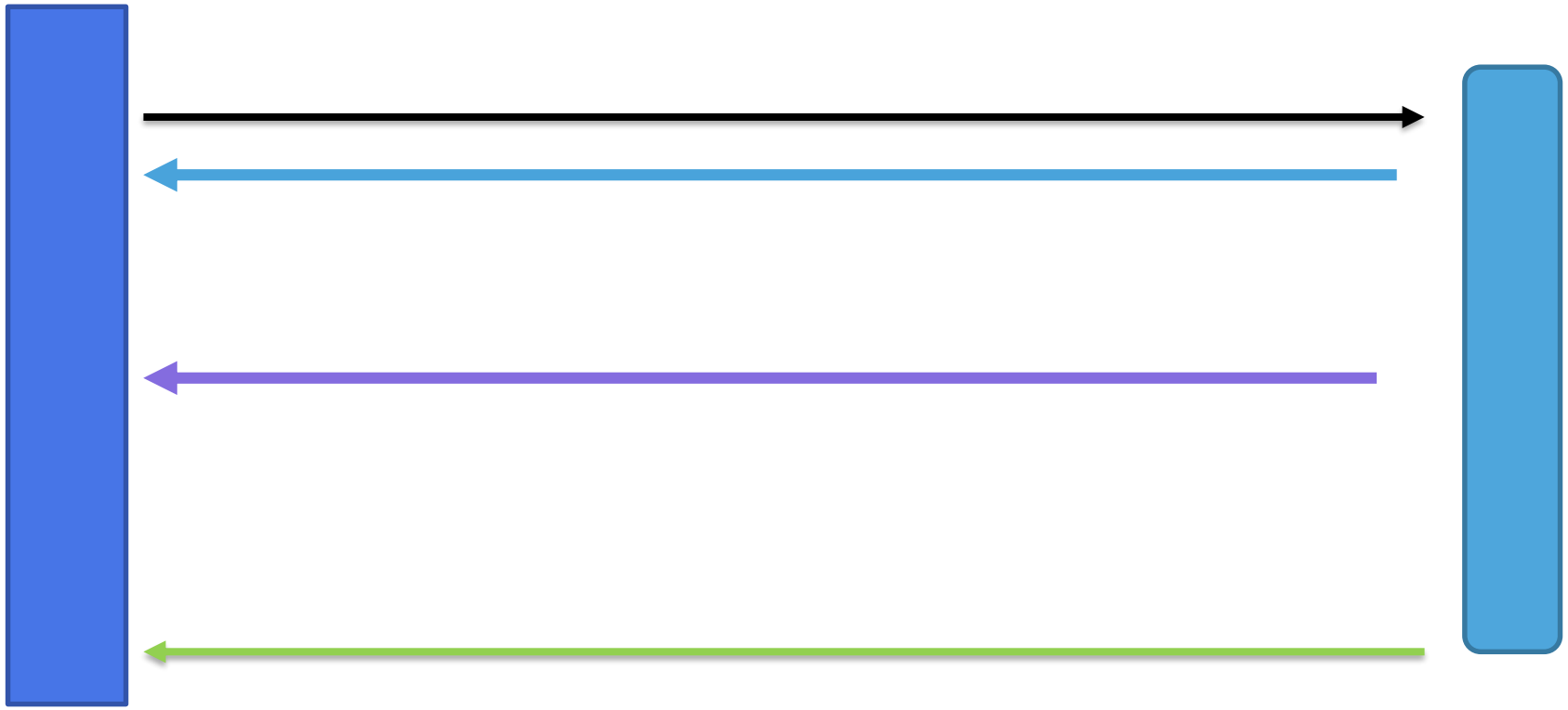
underfetching

Why this hype?

“It enables clients to **specify exactly what data is needed**, makes it easier to **aggregate data** from multiple sources, and uses a type system to describe data.”

GraphQL Request/Response

Client

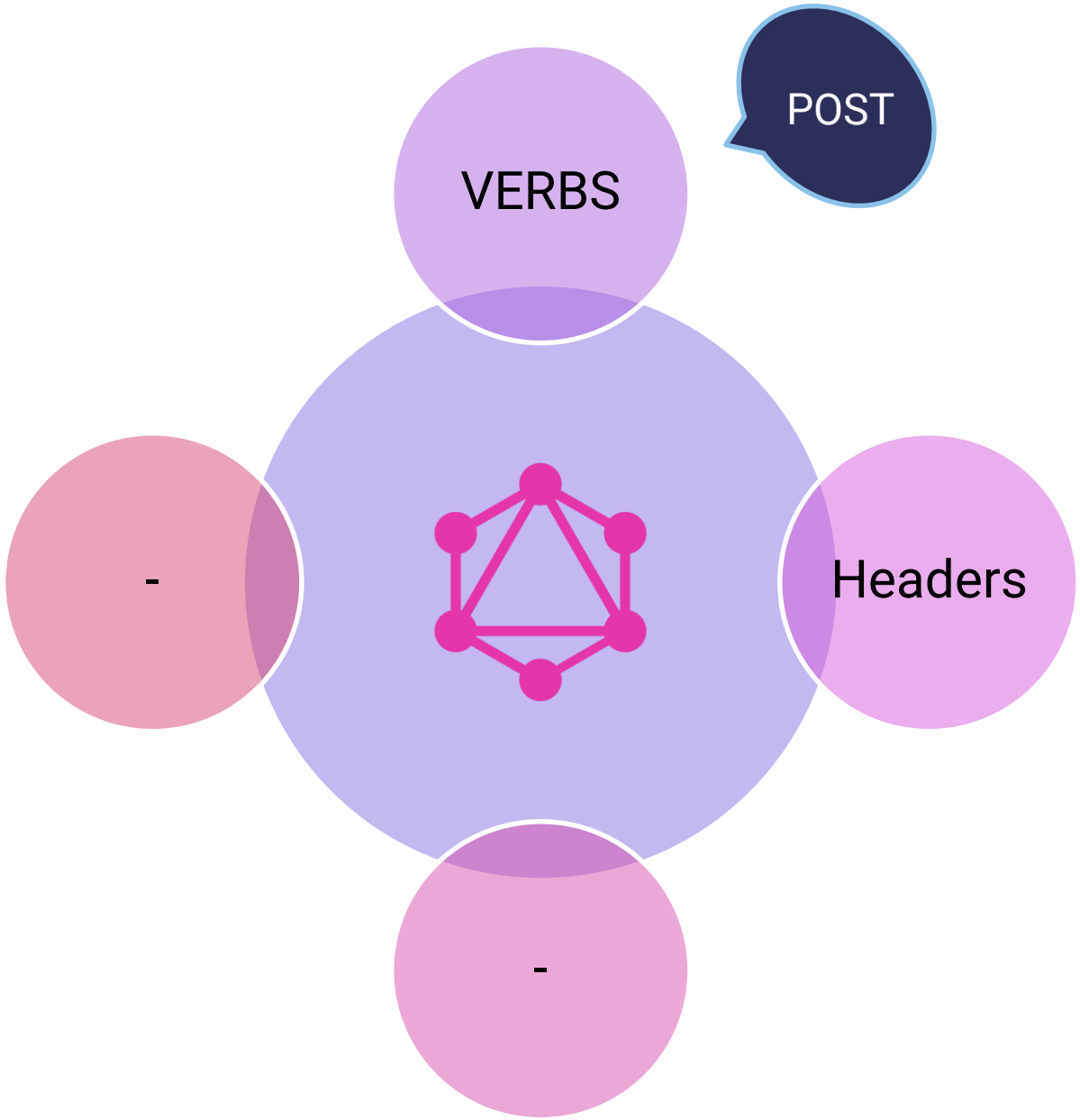


What it solves?

http://coolapi.com/speakers
http://coolapi.com/speakers/1
http://coolapi.com/speakers/1/talks
http://coolapi.com/talks

+

GraphQL



/speakers


```
[
    {
     "companyName": "Microsoft",
     "description": "Speaker, Teacher, Coder, Blogger",
     "id": 1,
     "lastName": "Hanselman",
     "firstName": "Scott",
     "position": "Program Manager“,
     “address” : { … },
     “twitter” :  “”,
     “github” : “”,
    “phoneNumber” : “”
    },
    …
]
```

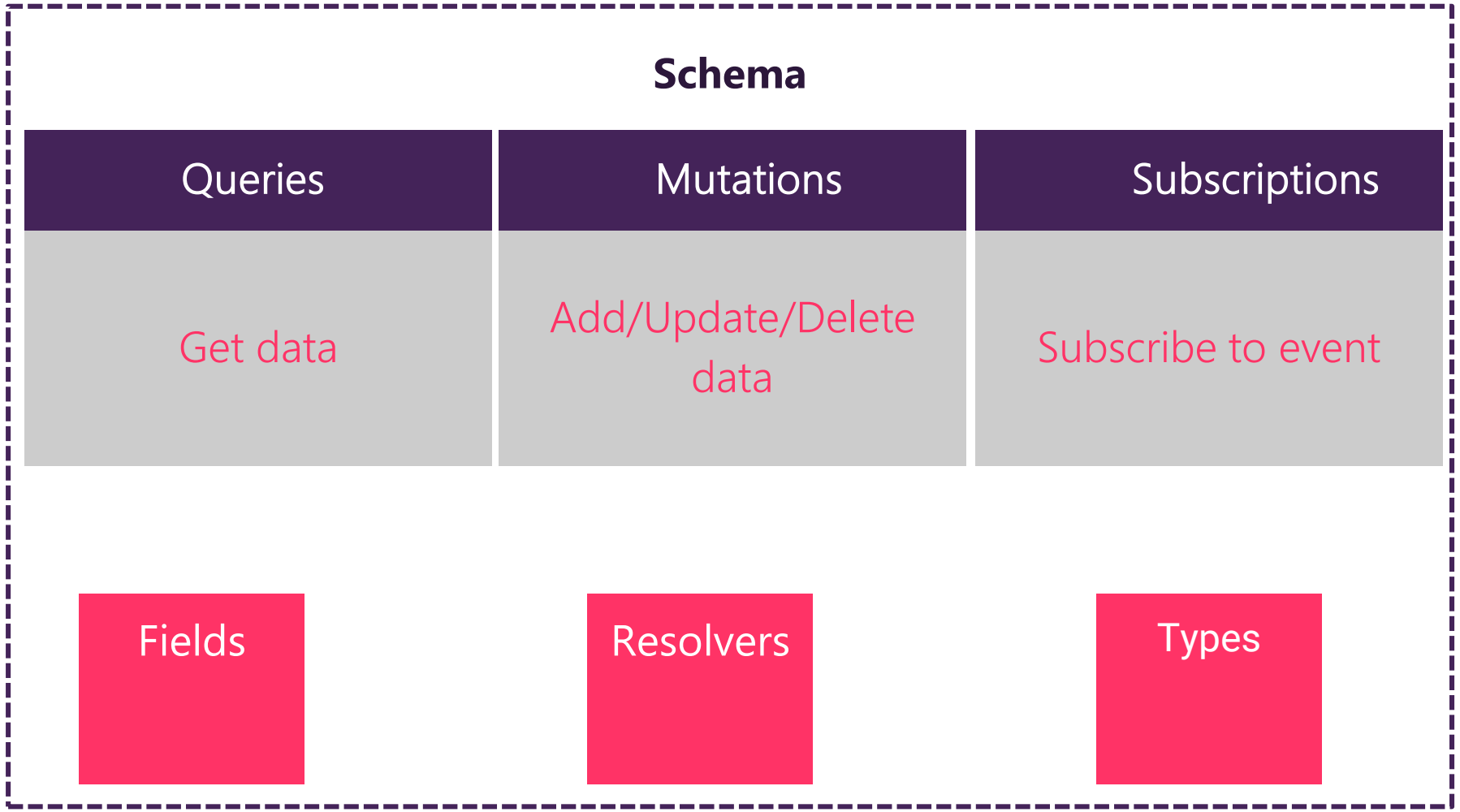
/speakers/1


```
[
    {
     "companyName": "Microsoft",
     "description": "Speaker, Teacher, Coder, Blogger",
     "id": 1,
     "lastName": "Hanselman",
     "firstName": "Scott",
     "position": "Program Manager“,
     “address” : { … },
     “twitter” :  “”,
     “github” : “”,
    “phoneNumber” : “”
    },
    …
]
```

/speakers/1/talks


```
{
  "data": {
    "talk": {
      "description": "There is an entire universe outside REST apis. You just need to fly there",
      "title": "GraphQL",
      "speaker": {
        “firstName": “Irina"
      }
    }
  }
}
```

Building blocks



Query

- A query is everything that can be ‘questioned’ from the outside
- Can have headers
- You can run multiple queries in parallel
- You need to define the query type

Querying in GraphQL

```
query {
    speakers {
        companyName
        lastName
        firstName
        twitter
    }
}
```


```
{
  "data": {
    "speakers": [
      {
        "companyName": "Microsoft",
        "lastName": "Hanselman",
        "firstName": "Scott",
        "twitter": “”
      },
          …
        ]
    }
}
```

/talks/2/speaker

```
query {
        talk(id: 2) {
                description
                title
                speaker {
                    lastName
                }
        }
}
```

```
{
  "data": {
    "talk": {
      "description": "There is an entire universe outside REST
API. You just need to fly there",
      "title": "GraphQL",
      "speaker": {
        "lastName": “Irina"
      }
    }
  }
}
```

Mutation

- A Mutation is a POST, UPDATE, DELETE
- Can have headers
- Run one by one
- You need to define the Input type

Mutation

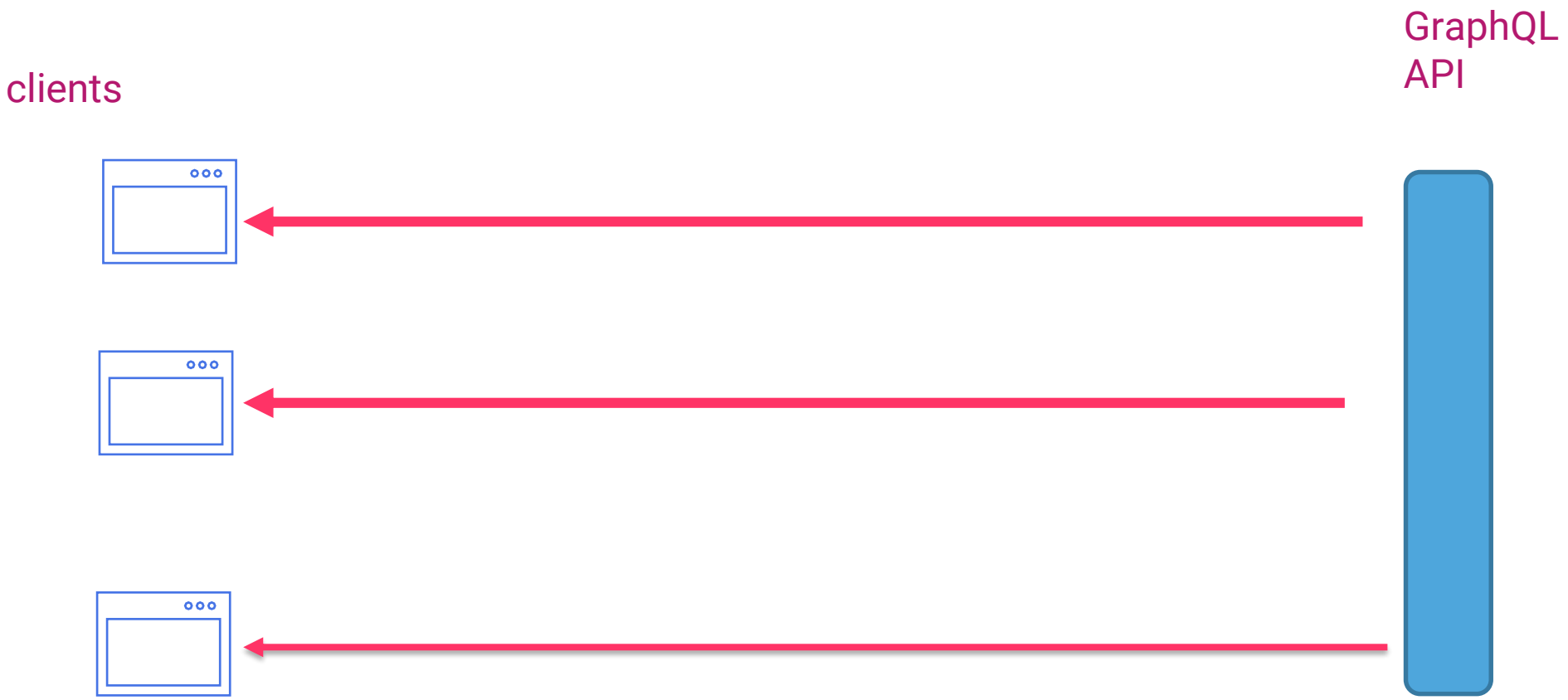
```
mutation($talk: talkInput!) {
      createTalk(talkInput: $talk) {
        title
        description
        speakerId
      }
}
```

Parameter

```
{
      "talk": {
      "title" :"Awesome .Net Core",
       "description" :"we'll talk about how cool it is .Net Core",
       "speakerId": 2
      }
}
```

Subscriptions

Subscriptions



GraphQL in .NET

Setup steps

- Install-Package GraphQL
- Install-Package GraphQL.Server.Transports.AspNetCore
- Install-Package GraphQL.Server.Ui.Playground
- Add middlewares
- Create Schema
- Resolve Query
- Resolve Mutations

#done
#NOTdone

talk | speakers

PRETTIFY | HISTORY | http://localhost:64803/graphql | COPY CURL

```
# Write your query or mutation here
query {
  speakers{
    companyName
    description
    id
    lastName
    firstName
    position
    twitter
  }
}
```


```
{
  "data": {
    "speakers": [
      {
        "companyName": "Microsoft",
        "description": "Speaker, Teacher, Coder, Blogger",
        "id": 1,
        "lastName": "Hanselman",
        "firstName": "Scott",
        "position": "Program Manager",
        "twitter": null
      },
      {
        "companyName": "Endava",
        "description": "lalalaalala",
        "id": 2,
        "lastName": "Scurtu",
        "firstName": "Irina",
        "position": "Design Lead",
        "twitter": null
      },
      {
        "companyName": "Google",
        "description": "lalalaalala",
        "id": 3,
        "lastName": "xyz",
        "firstName": "Irina",
        "position": "Devops",
        "twitter": null
      }
    ]
  },
  "errors": [
    {
      "message": "GraphQL.ExecutionError: Cannot return null for non-null type. Field: twitter, Type:
```


DOCS | SCHEMA

QUERY VARIABLES | HTTP HEADERS | TRACING

Need to define

- Your graph entities
- Every available query
- Every mutation
- Schema and mutations

Field by Field

Define the schema

```
  public class ConferenceSchema : Schema
    {
        public ConferenceSchema(IDependencyResolver resolver) :
base(resolver)
        {
            Query = resolver.Resolve<ConferenceQuery>();
            Mutation = resolver.Resolve<ConferenceMutation>();
        }
    }
```

Define the returned types

```
public class Speaker : ObjectGraphType<Data.Entities.Speaker>
    {

        public Speaker()
        {
            Field(t => t.Id);
            Field(t => t.FirstName);
            Field(t => t.LastName);
            Field(t => t.Position).Description("The position in the company");
            Field(t => t.Description).Description("Speaker Bio");
            Field(t => t.CompanyName);
            Field(t => t.LinkedIn);
            Field(t => t.Twitter).Description("Twitter username");
        }
    }
```

```
public ConferenceQuery(SpeakersRepository speakersRepo,
TalksRepository talksRepo, FeedbackService feedbackService)
{
  Field<ListGraphType<Types.Speaker>>(
      "speakers",
      Description = "will return all the speakers",
          resolve: context => speakersRepo.GetAll()
            );
}
```

```
public ConferenceMutation(TalksRepository talkRepository)
        {
            FieldAsync<Talk>(
              "createTalk",
              arguments: new QueryArguments(
                  new QueryArgument<NonNullGraphType<TalkInput>>
                  {
                      Name = "talk"
                  }
              ),
              resolve: async context =>
              {
                  var talk = context.GetArgument<Data.Entities.Talk>("talk");

                  return await context.TryAsyncResolve(async c => await talkRepository.Add(talk));
              });

      FieldAsync<Talk>(
              “updateTalk",
        . . .
        }
```

The awesome GraphQL

World

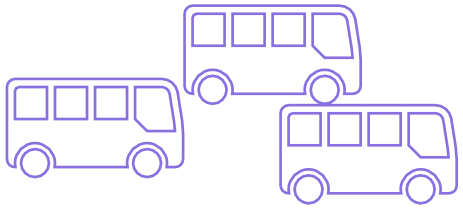
The Awesome GraphQL

- You want to defer understanding the user needs and how the client consumes your API
- Easy to get started
- Built-in introspection
- Friendly
- Is contract-driven
- wonderful playground(s)

Client performance First

Cool parts

- **Exact** fetching
- Less round-trips



- Wonderful experience for humans/api consumers

When?

- For small/complex projects
- Aggregation layer
- care about your consumer’s bandwidth
- Care about bandwidth
- you have no control over the client-app
- empower your consumer

Is GraphQL better than REST?

Application state is driven by the client


Querying the database?

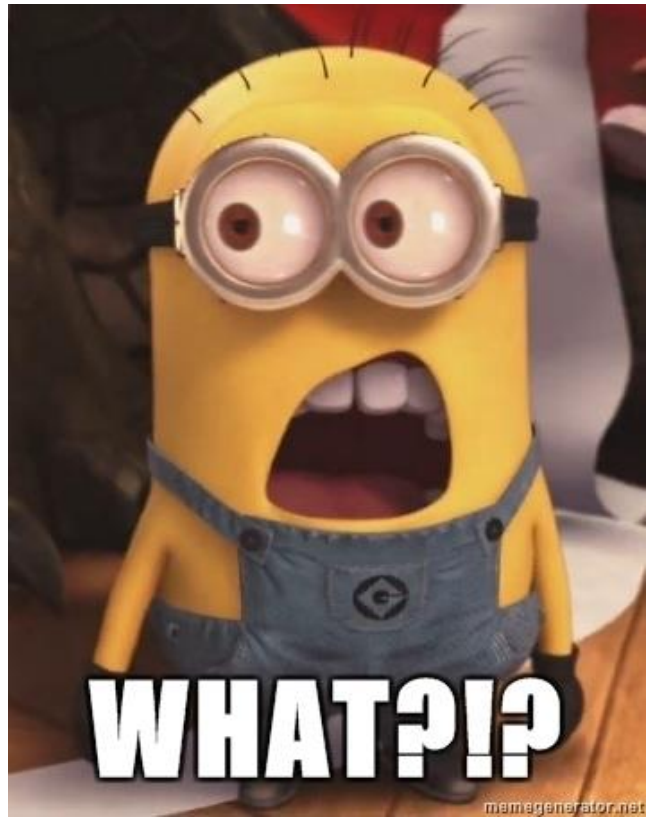
Problems?

- Single endpoint
- Only POST requests

Problems?

- Forget about all you know in HTTP

Caching



DataLoader

- Tries to solve the n+1 problem
- It can batch multiple requests into one
- The defined name is contextual to that ‘entity’
- Can ‘hold’ in memory the result of a query

In summary

- Is a new tool for our toolbox
- Anything that can issue a HTTP request, can consume a GraphQL API
- Leverage only POST as verb
- Single endpoint
- No caching using headers
- A lot of flexibility
- Suitable for aggregation layers

Resources

- https://graphql-dotnet.github.io
- https://graphql.org/

- https://hotchocolate.io/

- https://github.com/APIs-guru/graphql-over-http

There is no silver bullet

Choose the right tech

Q&A

@irina_scurtu

THANK YOU!

@irina_scurtu