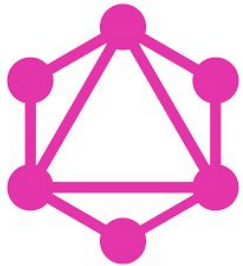
:iTechArt

Software
Engineering
Services



GraphQL.

Production experience and pitfalls

Anton Chernov

Hi there.
I’m Anton Chernov

D1 G1

Contact Information:

Email: chernov.anton.dev@gmail.com

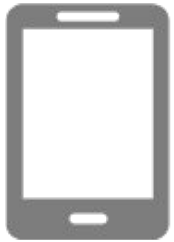
AGENDA

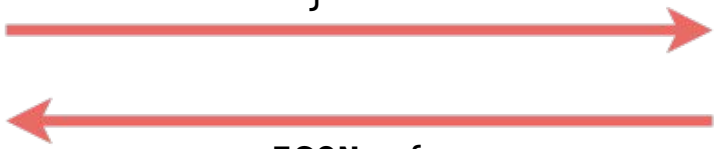
1. GraphQL. What is it?
2. History
3. Design
4. How to start
5. Production experience
6. Summarize

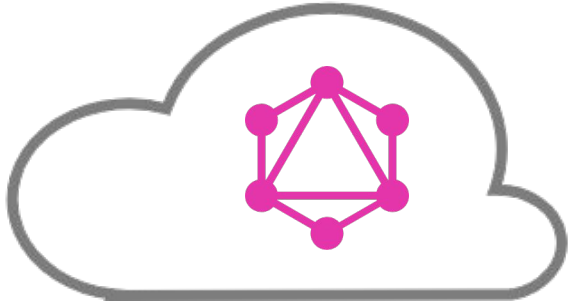
What is GraphQL?

It’s a query language for client-server communication

```
query {
  user {
    id
    name
  }
}
```







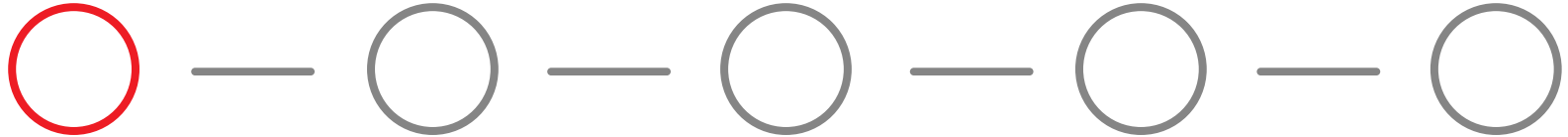
```
JSON: {
  "data": {
    "user": {
      "id": "2001",
      "name": "Tom"
    }
  }
}
```

HISTORY

→ First was Facebook

HISTORY

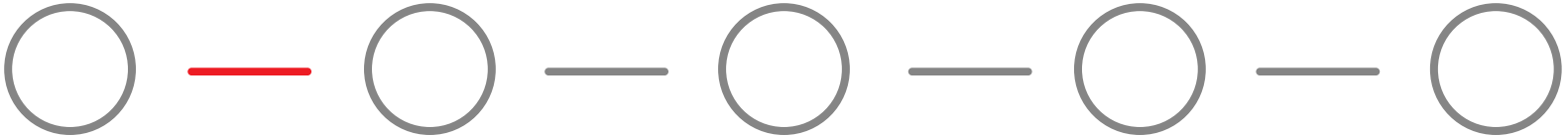
Prototype



Feb 2012

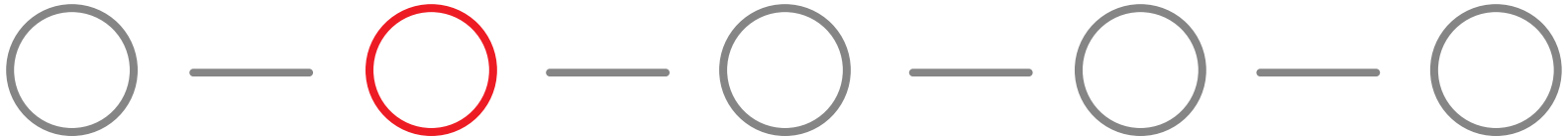
HISTORY

Initial development



HISTORY

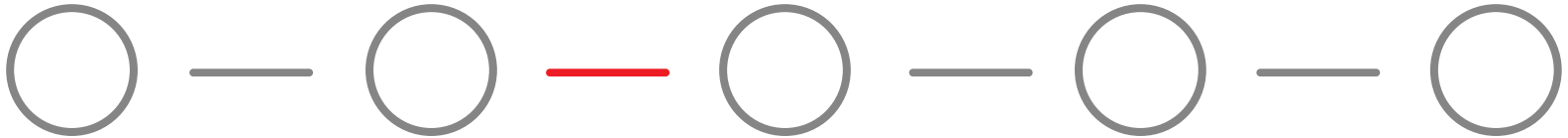
Production Use



Aug 2012

HISTORY

Evolution



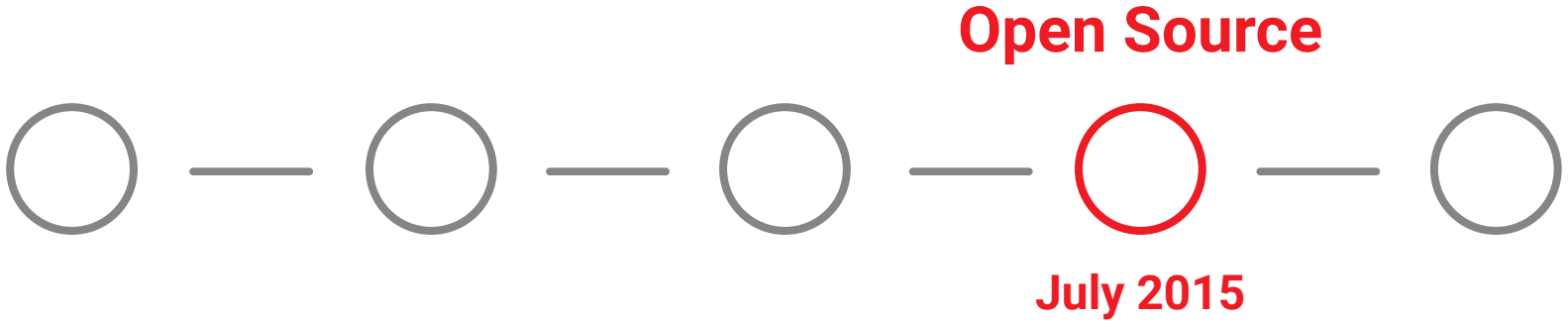
HISTORY

Discussed publicly

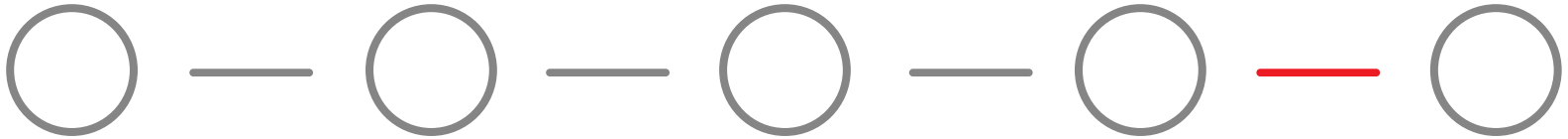
Jan 2015

HISTORY

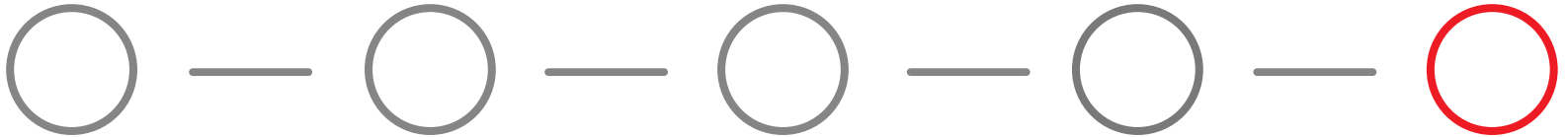
HISTORY



HISTORY



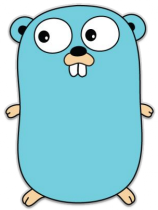
HISTORY



Production Ready

Sept 2016

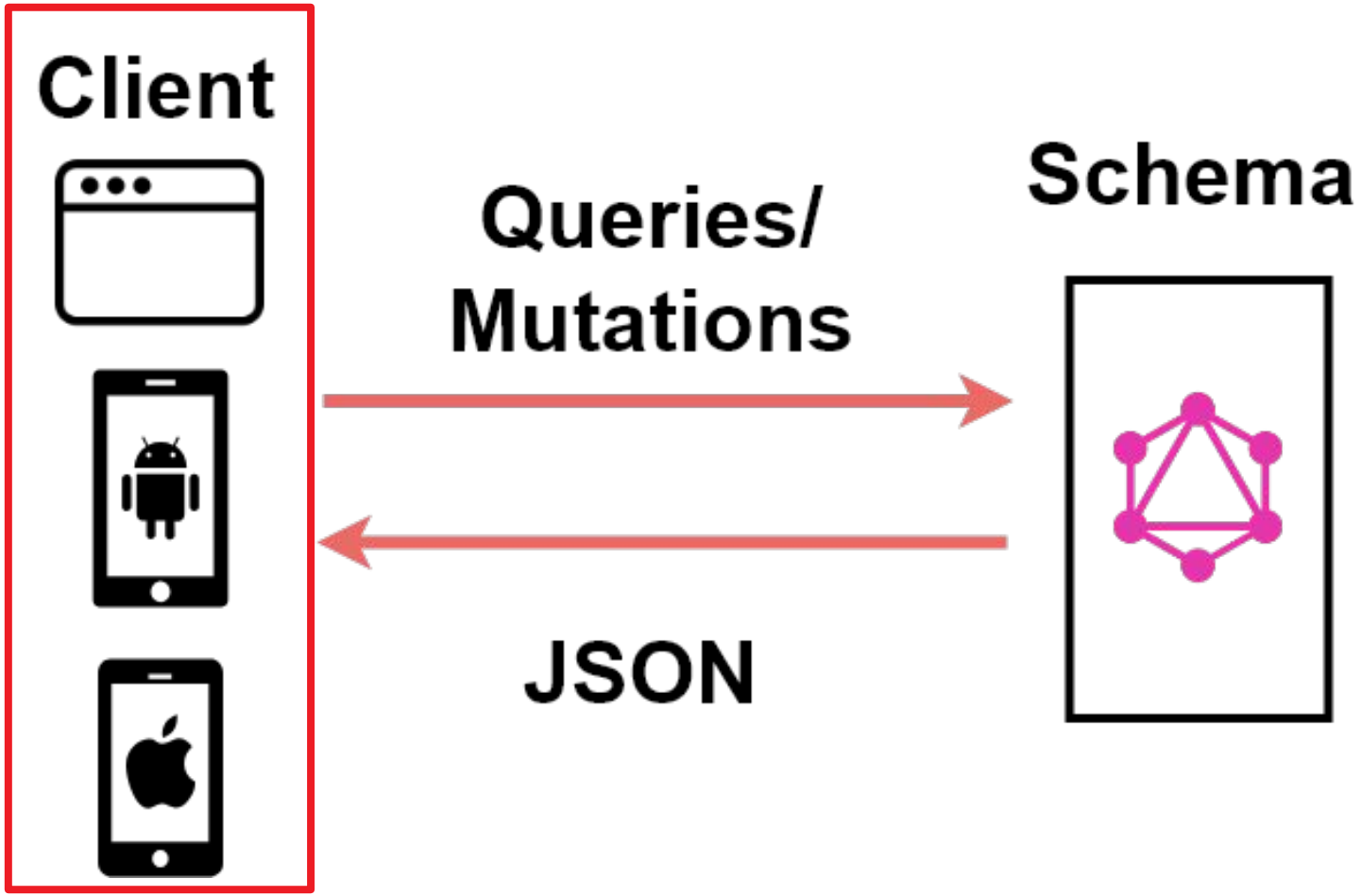
IMPLEMENTATIONS



DESIGN

→ Big picture

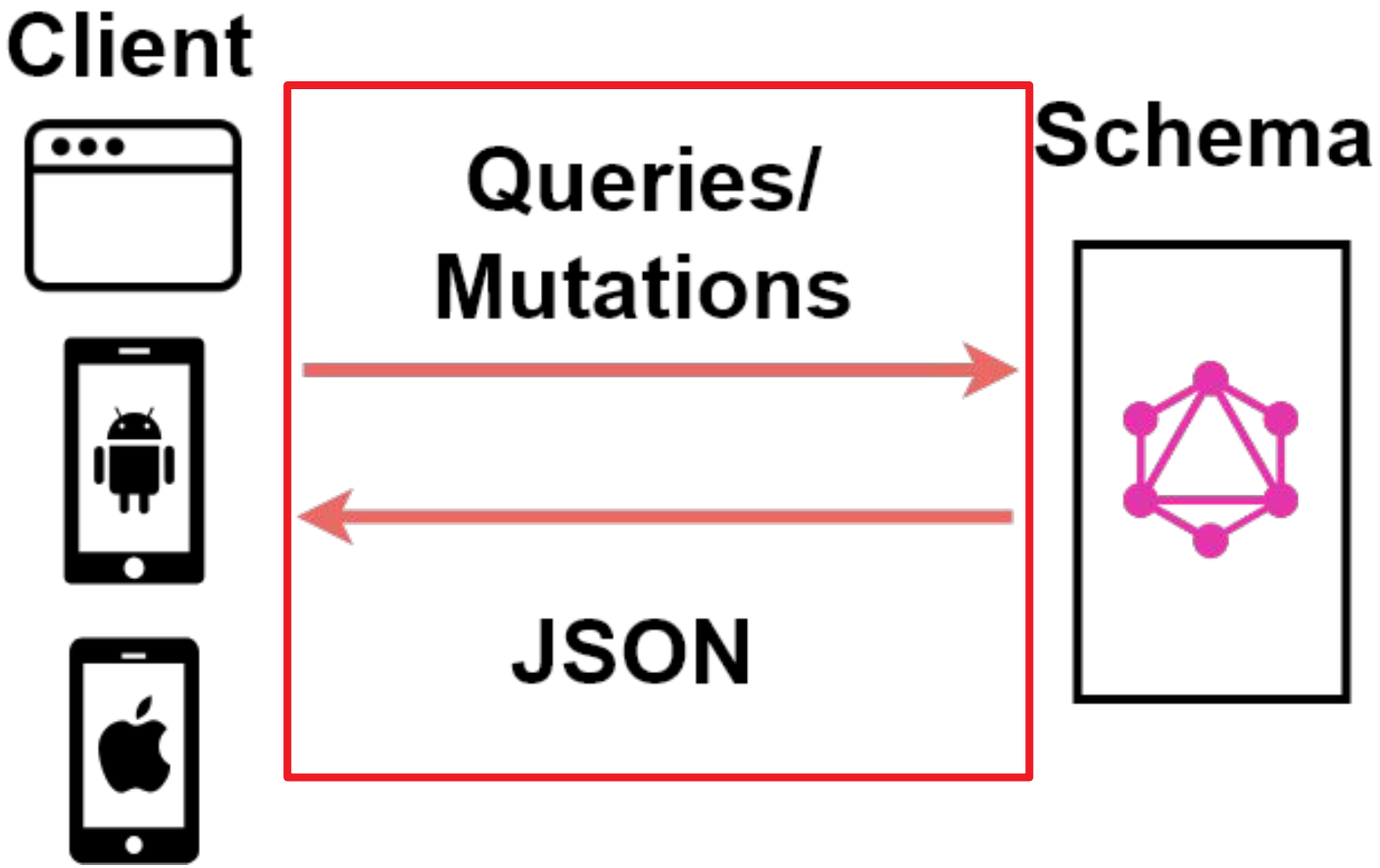
CLIENT


Client
Queries/
Mutations
Schema
JSON

CLIENT

- apollo-client
- apollo-ios
- apollo-android

- relay (relay modern)


Client
Queries/
Mutations
Schema
JSON

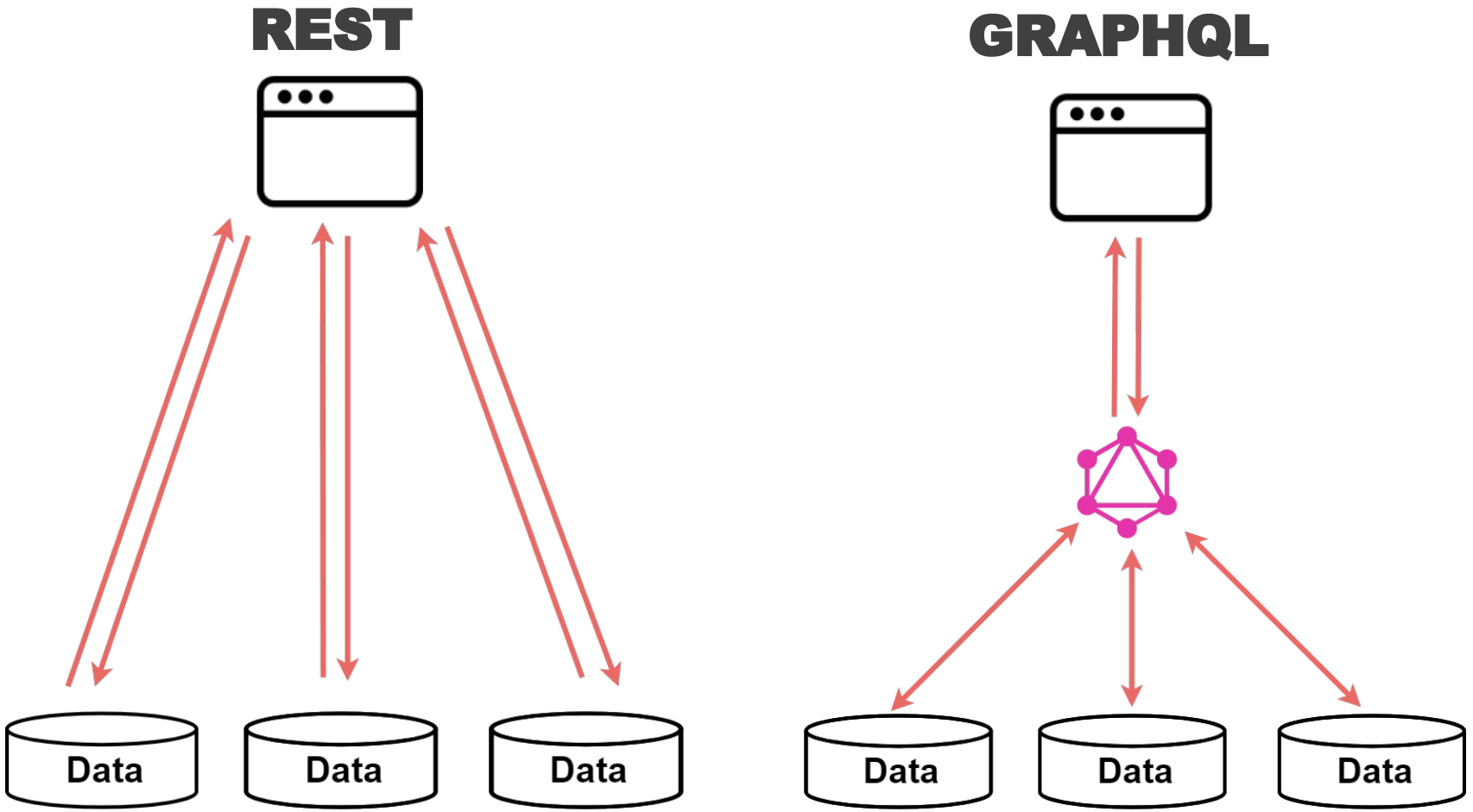
ENDPOINT

GRAPHQL

```
http://myapi/graphql?query={me{name}}
```

REST

```
http://myapi/hero
http://myapi/friend
http://myapi/what_else_i_need_to_provide_to_client
```


REST
Data
Data
Data
GRAPHQL
Data
Data
Data

QUERIES

Request

```
query {
    user {
        id
        name
        createdAt
    }
}
```

Response

```
"data": {
    "author": {
        "id": 9722695,
        "name": "Tom",
        "createdAt": "2014-11"
    }
}
```

Request

```
query {
    user {
        id
        name
        createdAt
        posts(first: 2) {
            title
        }
    }
}
```

Response

```
"data": {
    "author": {
        "id": 9722695,
        "name": "Tom",
        "createdAt": "2014-11-13T16",
        "posts": [
            { "title": "Graphql" },
            { "title": "Awesome" }
        ]
    }
```

MUTATIONS

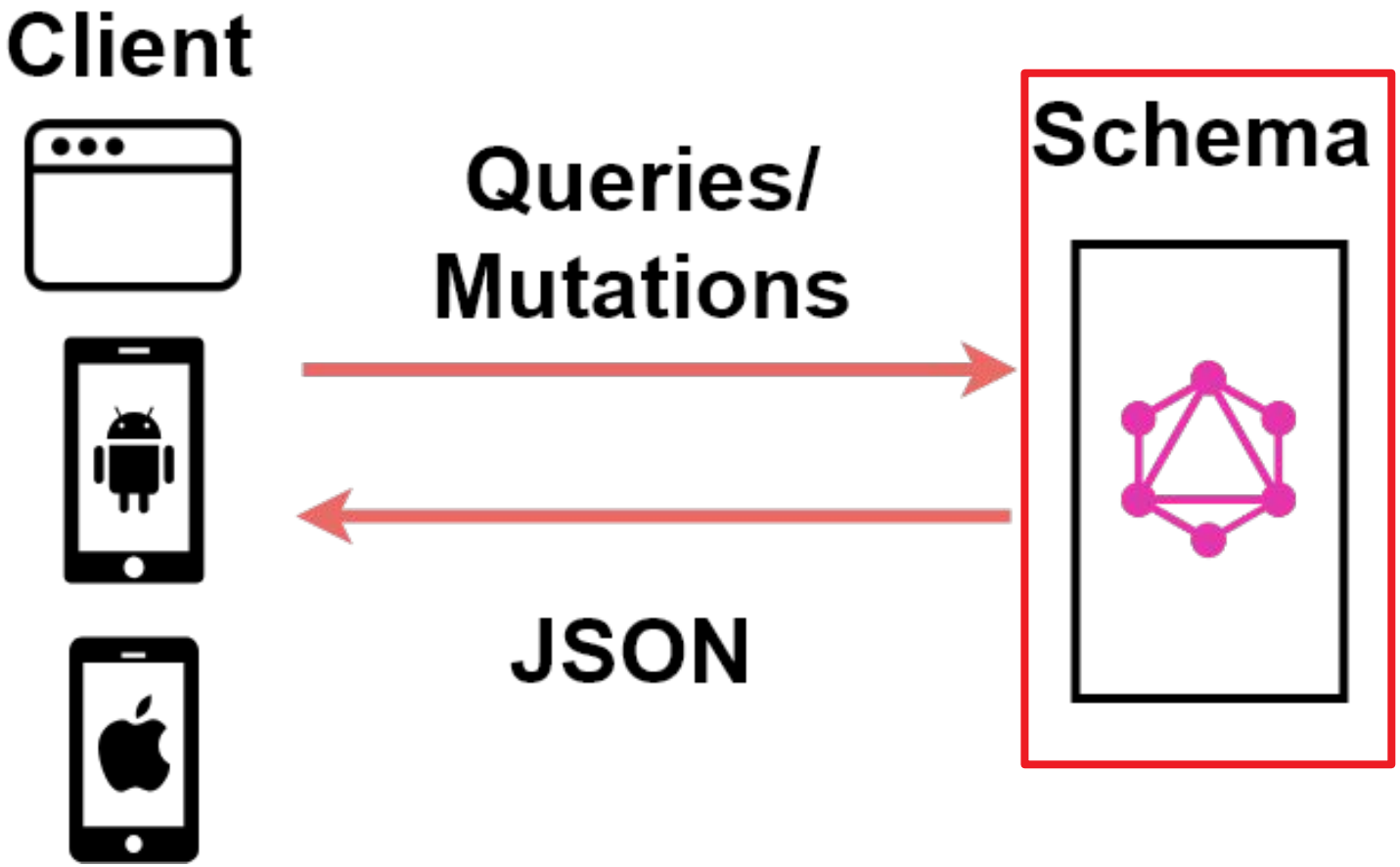
Mutation

```
mutation ($post: PostInput!) {
    addPost(post: $post) {
        post {
            id
            title
        }
    }
}
```

Response

```
"data": {
    "post": {
        "id": "1234",
        "title": "New post!"
    }
}
```

SCHEMA


Client
Queries/
Mutations
Schema
JSON

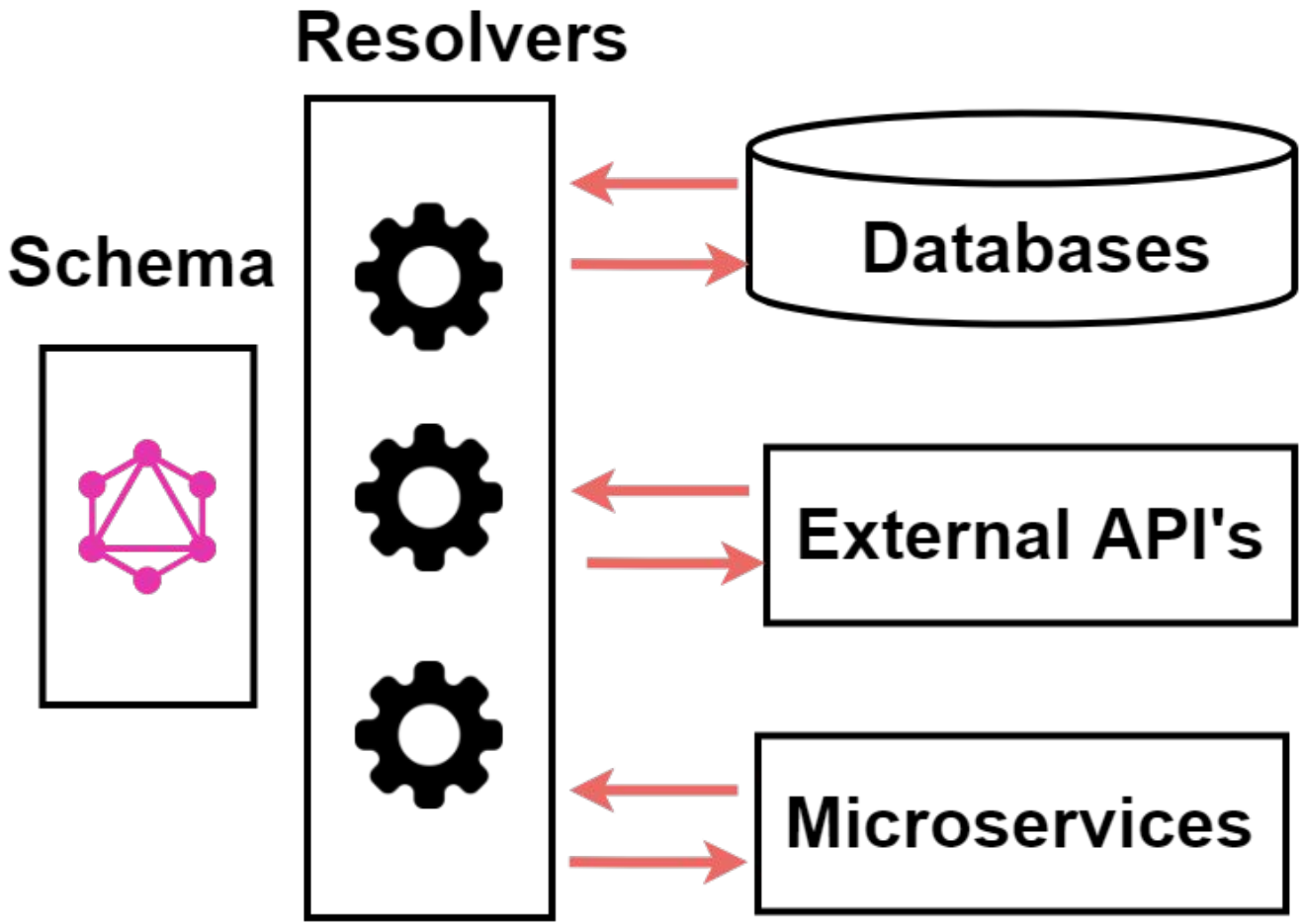
Root types

```
type Query {
    posts: [Post]
    author(id: Int!): Author
}

type Mutation {
    upvotePost(postId: Int!): Post
}
```

Types

```
type Author {
    id: Int!
    firstName: String
    lastName: String
    posts: [Post]
}

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

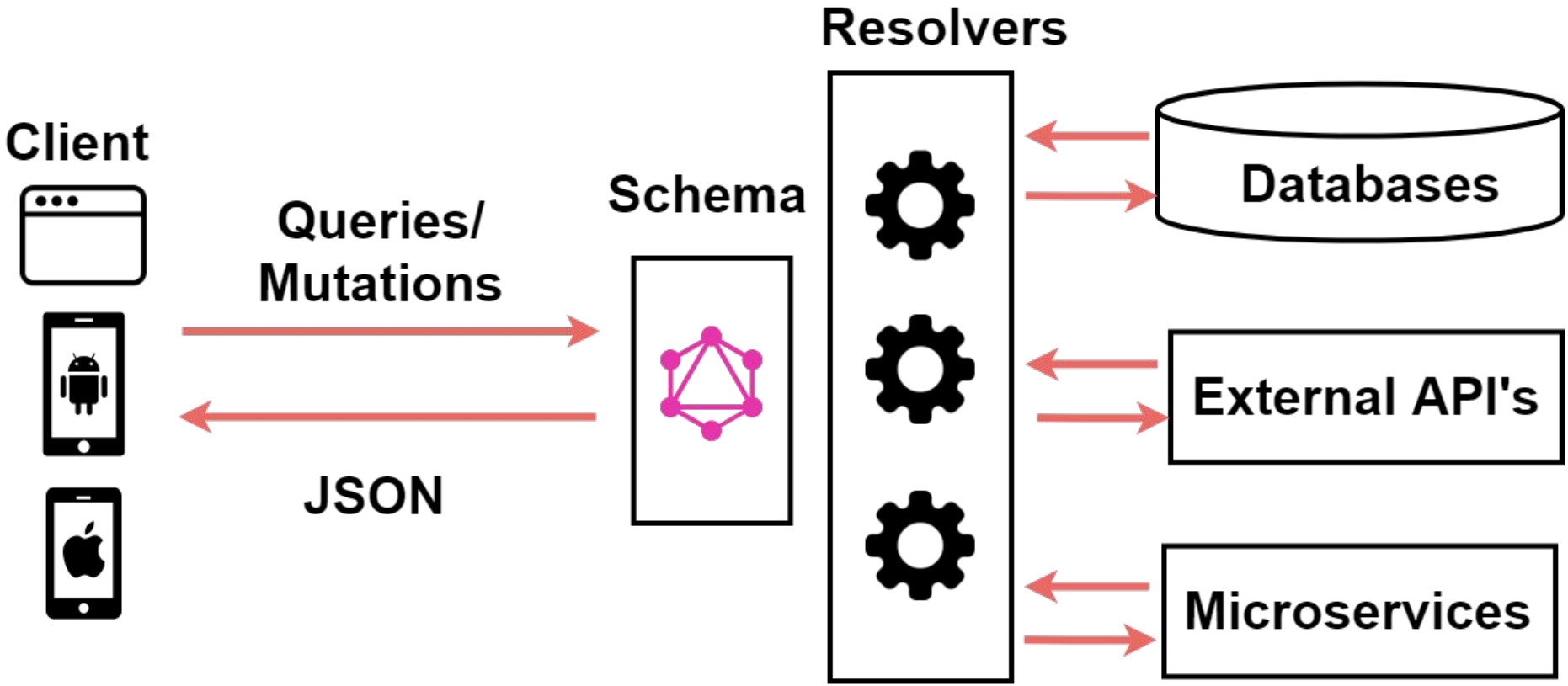
RESOLVERS


Resolvers
Schema
Databases
External API's
Microservices

Resolvers

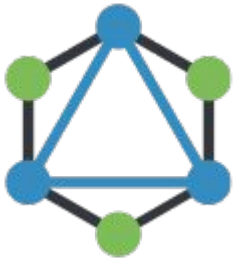

```
Query: {
    posts: () =>
        Post.findAll(),
    author: (_, {id}) =>
        Author.find(id),
},
Mutation: {
    upvotePost: (_, {postId}) => {
        const post = Post.upvotePost(postId);
        return post;
    },
}
```

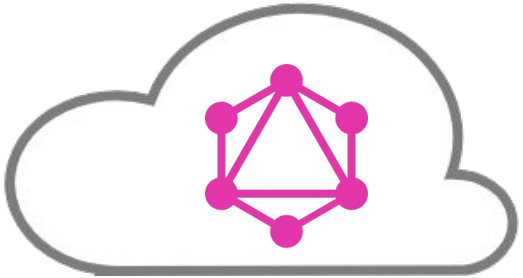
```
Author: {
    posts: author =>
        Post.findByAuthor(author.id),
},

Post: {
    author: post =>
        Author.findByPost(post.id),
},
```


Resolvers
Client
Schema
Queries/
Mutations
JSON
Databases
External API's
Microservices

DOCUMENTATION

```
query {
    __schema {
        ...schema
    }
}
```

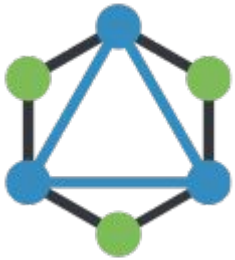


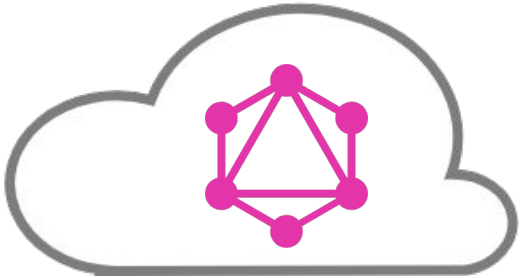


```
full schema
...types
...queries
...mutations
```

```
query {
    __schema {
        ...schema
    }
}
```

altair.sirmuel.design





```
full schema
...types
...queries
...mutations
```

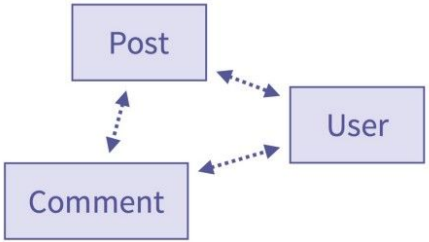
HOW TO START

→ Some advice

TOO MUCH FEATURES

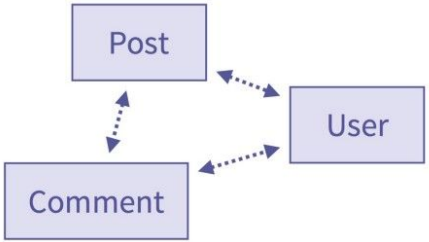
START GRADUALLY

GRAPHQL DEVELOPMENT

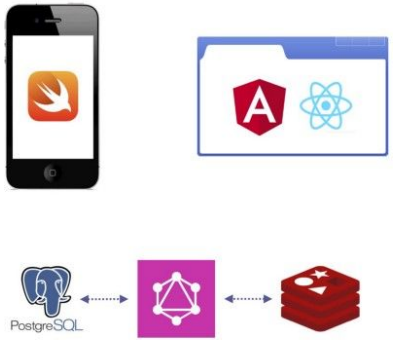


1. Design API Schema

GRAPHQL DEVELOPMENT

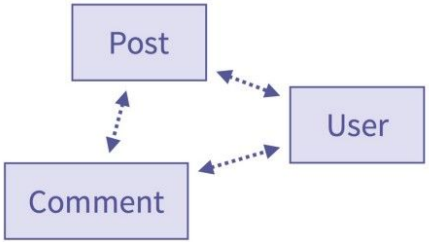


1. Design API Schema

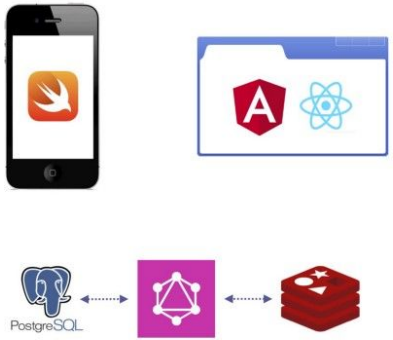


2. Build UI and Backend

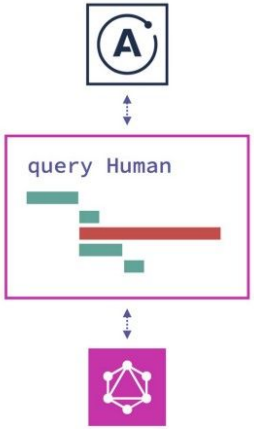
GRAPHQL DEVELOPMENT



1. Design API Schema



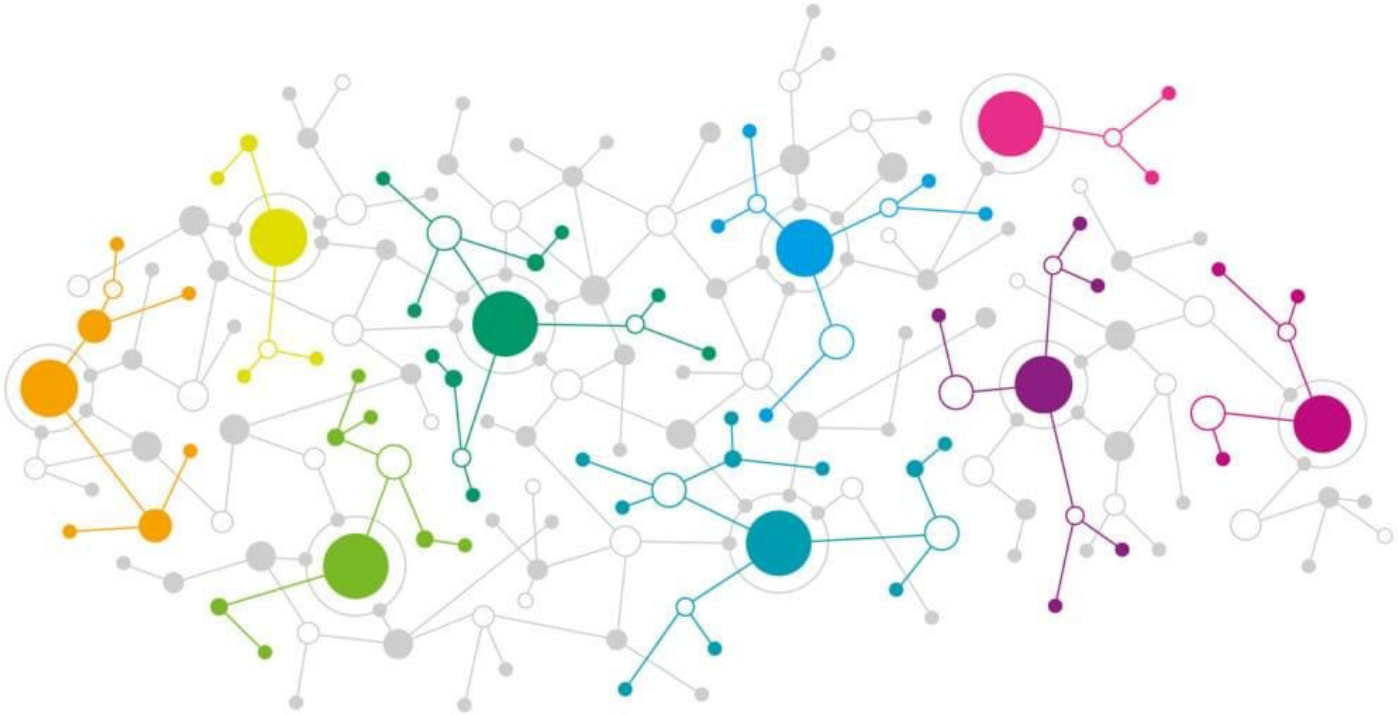
2. Build UI and Backend



3. Run in production

Think in graphs, not endpoints

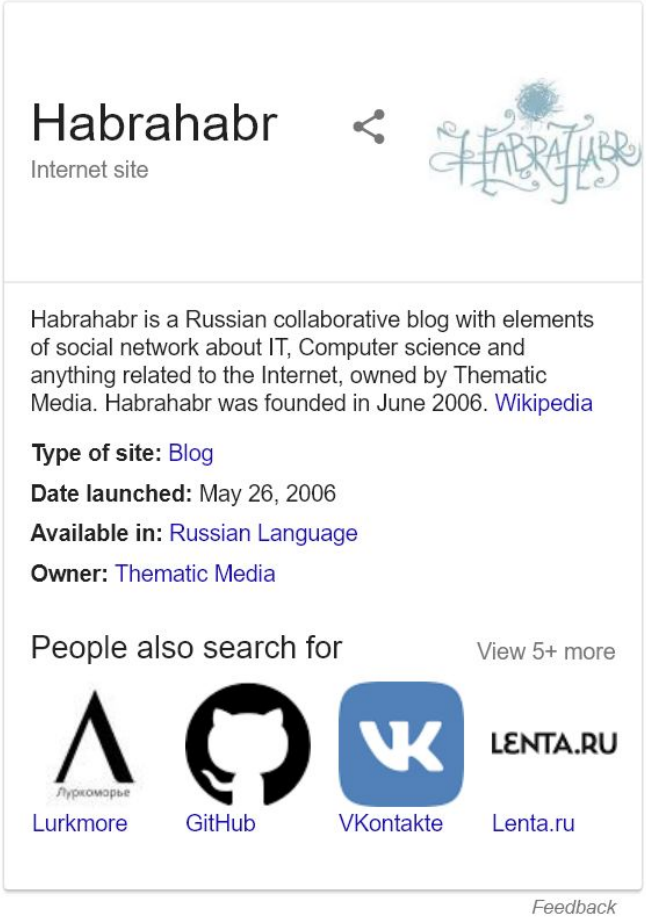
ONE BIG GRAPH



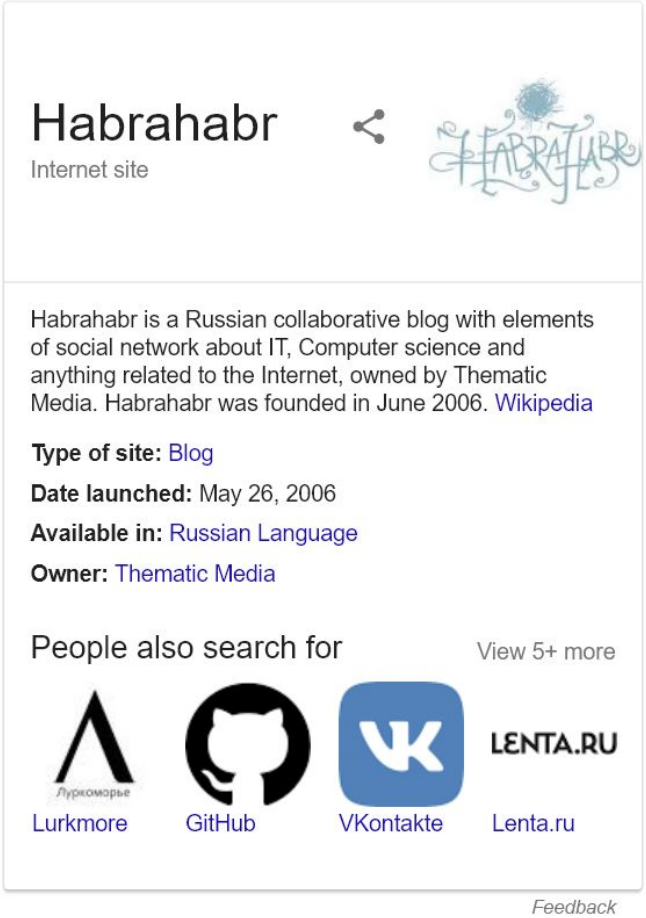
ONE BIG GRAPH


```
type Query {
    search(query: Sring): Search
}
type Search {
    id: ID
    title: String,
    results(first: Int, after: ID): [Node]
    suggestedSearches(first: Int, after: ID): [Search]
}
```

DESCRIBE DATA, NOT VIEW

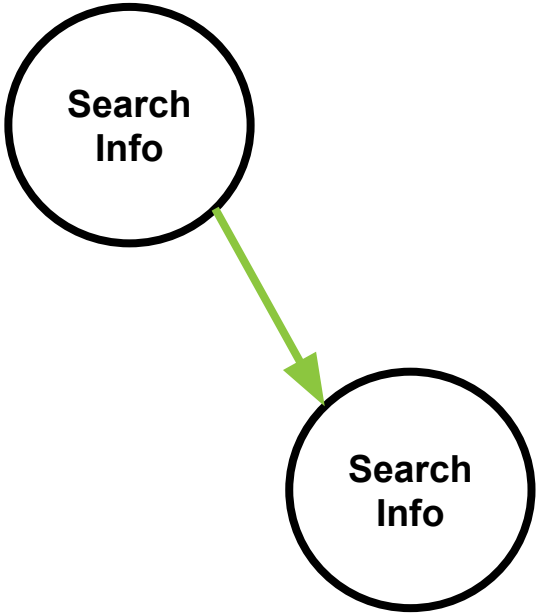



```
search {
    url
    title
    subTitle
    imageUrl
    description
    metaInfo {
        label
        value
    }
    relatedSearches ...
```

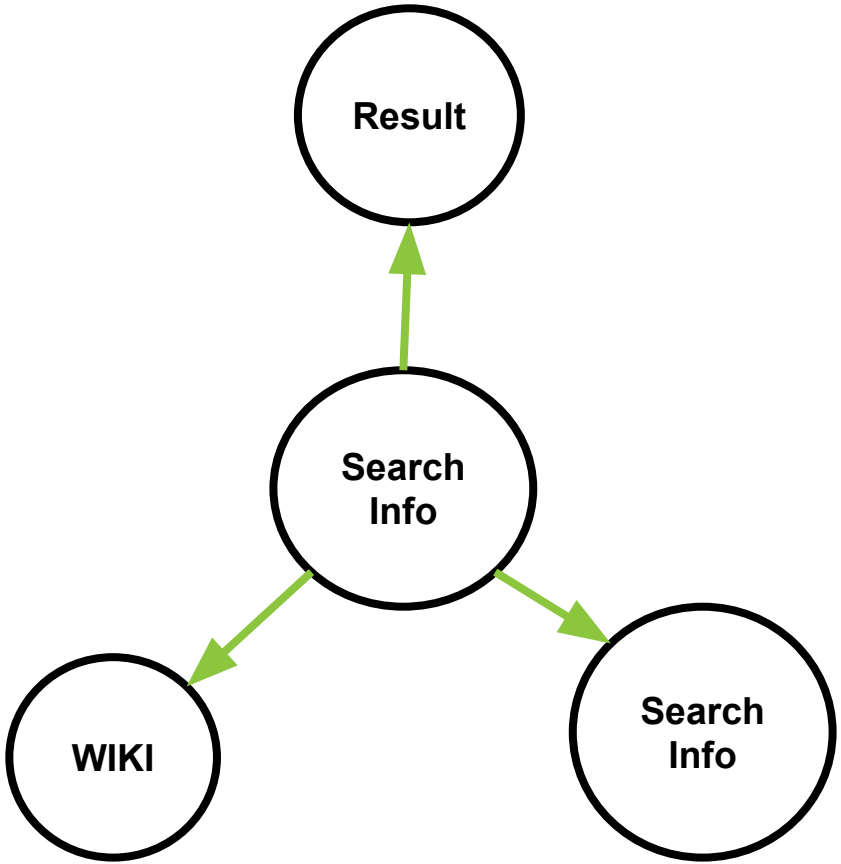



```
search {
    info {
        url
        title
        subTitle
        imageUrl(size: "MEDIUM")
    }
    wiki {
        description
        metaInfo {
            label
            value
        }
    }
    relastedSearches(first: 4) ...
```

BAD


Search
Info
Search
Info

GOOD


Result
Search
Info
WIKI
Search
Info

DESCRIBE DATA

1. How version 2 will look like?

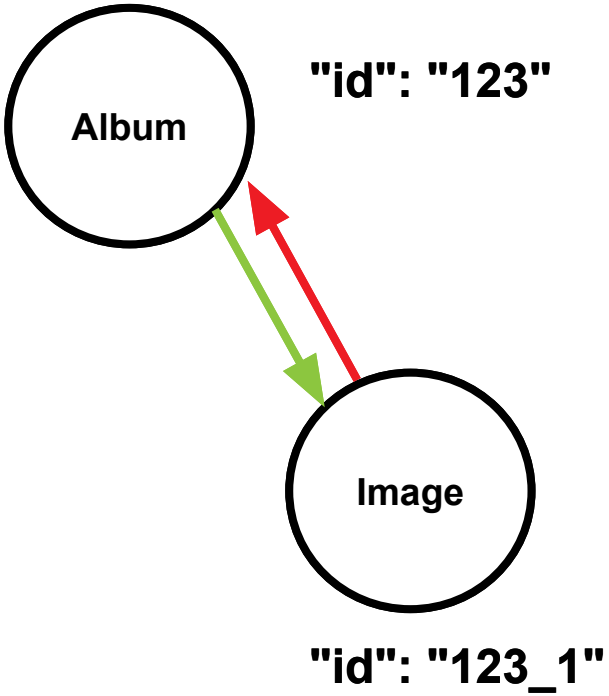
DESCRIBE DATA

1. How version 2 will look like?
2. **Will it work for new clients?**

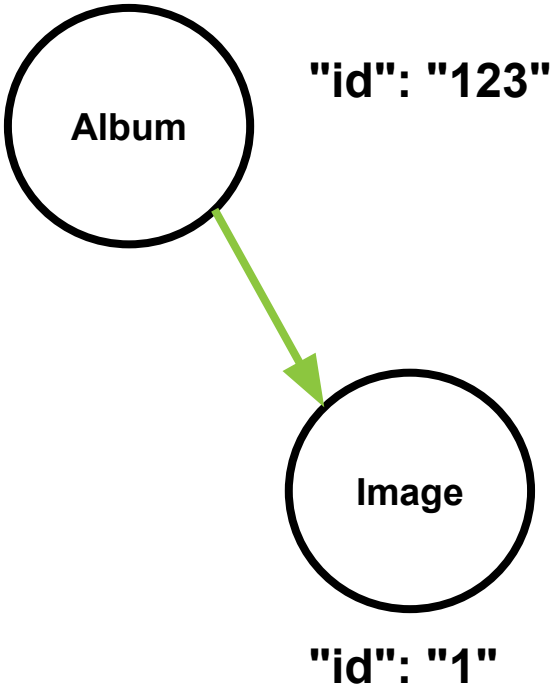
DESCRIBE DATA

1. How version 2 will look like?
2. Will it work for new clients?
3. **What if implementation changes?**

BAD



GOOD



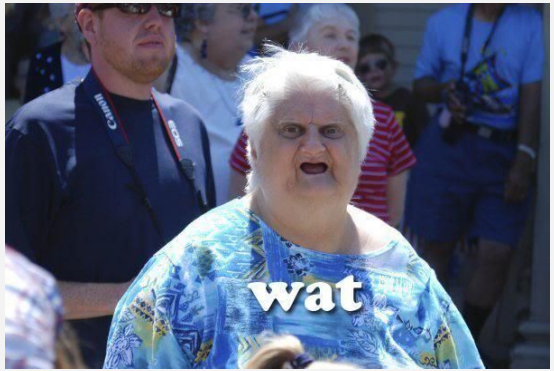
NAME MATTERS

BAD NAMES


```
type Mutation {
    saveJob(saveJob: SaveJobInputObject!): SaveJobPayload!
    updateJob(updateJob: UpdateJobInputObject!): UpdateJobPayload!
    createJob(createJob: CreateJobInputObject!): CreateJobPayload!
}
```

BAD NAMES


```
type Mutation {
    saveJob(saveJob: SaveJobInputObject!): SaveJobPayload!
    updateJob(updateJob: UpdateJobInputObject!): UpdateJobPayload!
    createJob(createJob: CreateJobInputObject!): CreateJobPayload!
}
```



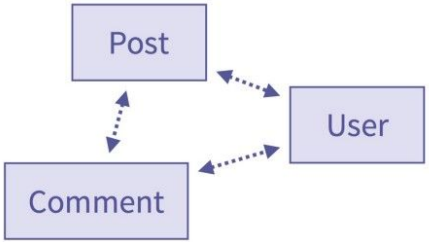
GOOD

```
type Mutation {
    likeJob(jobId: ID!): JobPayload!
    updateJob(job: jobInput!): JobPayload!
    createJob(job: jobInput!): JobPayload!
}
```

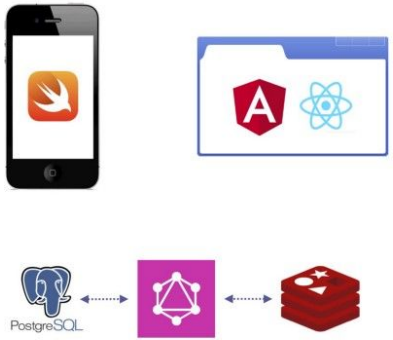
BAD

```
type Mutation {
    saveJob(saveJob: SaveJobInputObject!): SaveJobPayload!
    updateJob(updateJob: UpdateJobInputObject!): UpdateJobPayload!
    createJob(createJob: CreateJobInputObject!): CreateJobPayload!
}
```

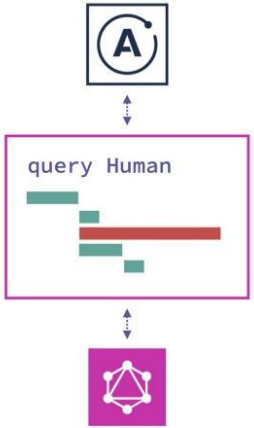
GRAPHQL DEVELOPMENT



1. Design API Schema



2. Build UI and Backend



3. Run in production

THIN LAYER

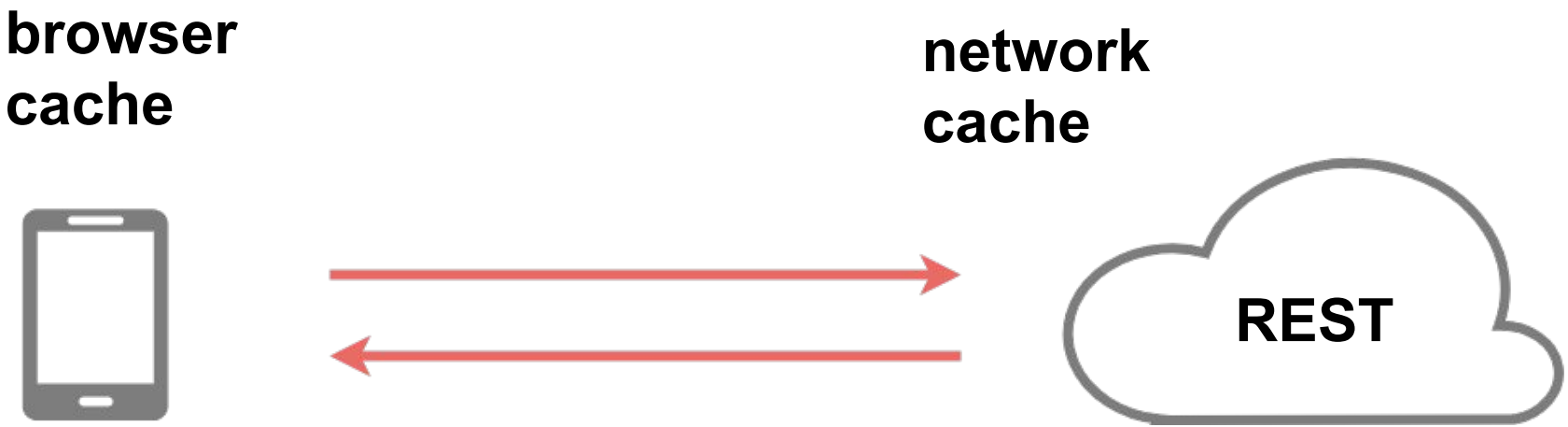
IN THEORY

- No caching
- No authentication
- No authorization
- One dummy endpoint

REALITY

CACHING

HTTP CACHING



HTTP CACHING

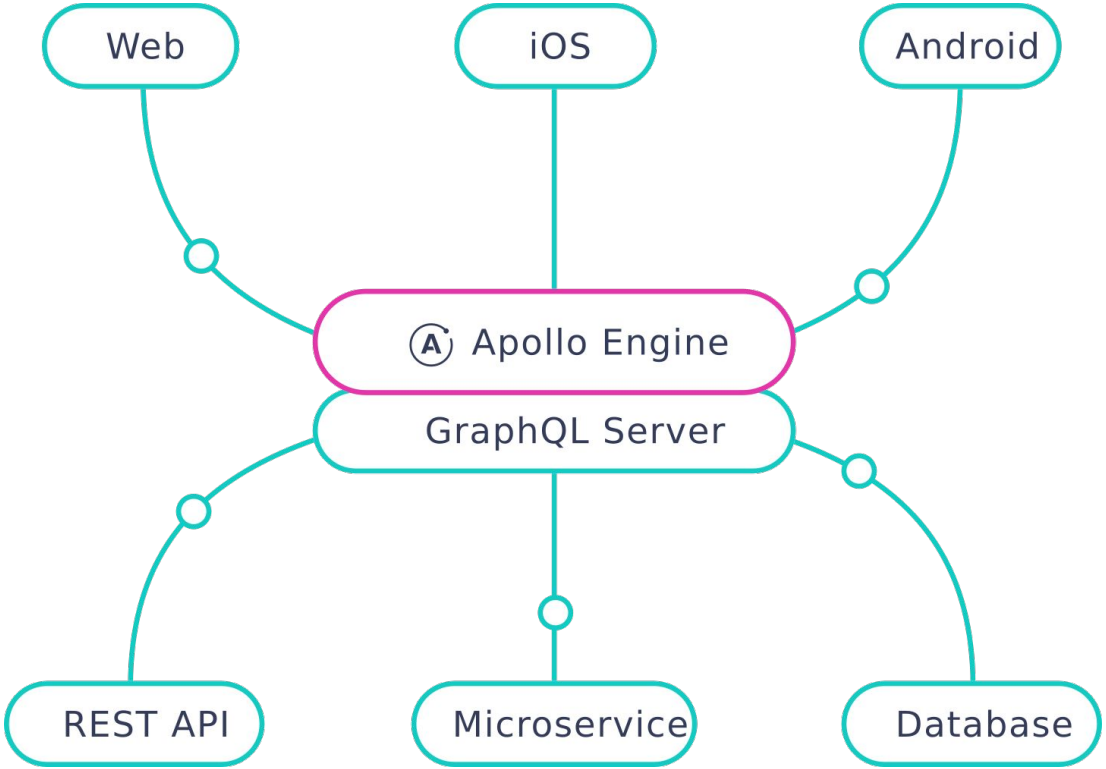
~~browser cache~~

~~network cache~~

CACHE OPTIONS

- Apollo Engine

Apollo Engine



Apollo Engine

- Error tracking
- Query caching
- Field-level schema analysis
- Improved tracing

CACHE OPTIONS

- Apollo Engine
- **In memory stores**

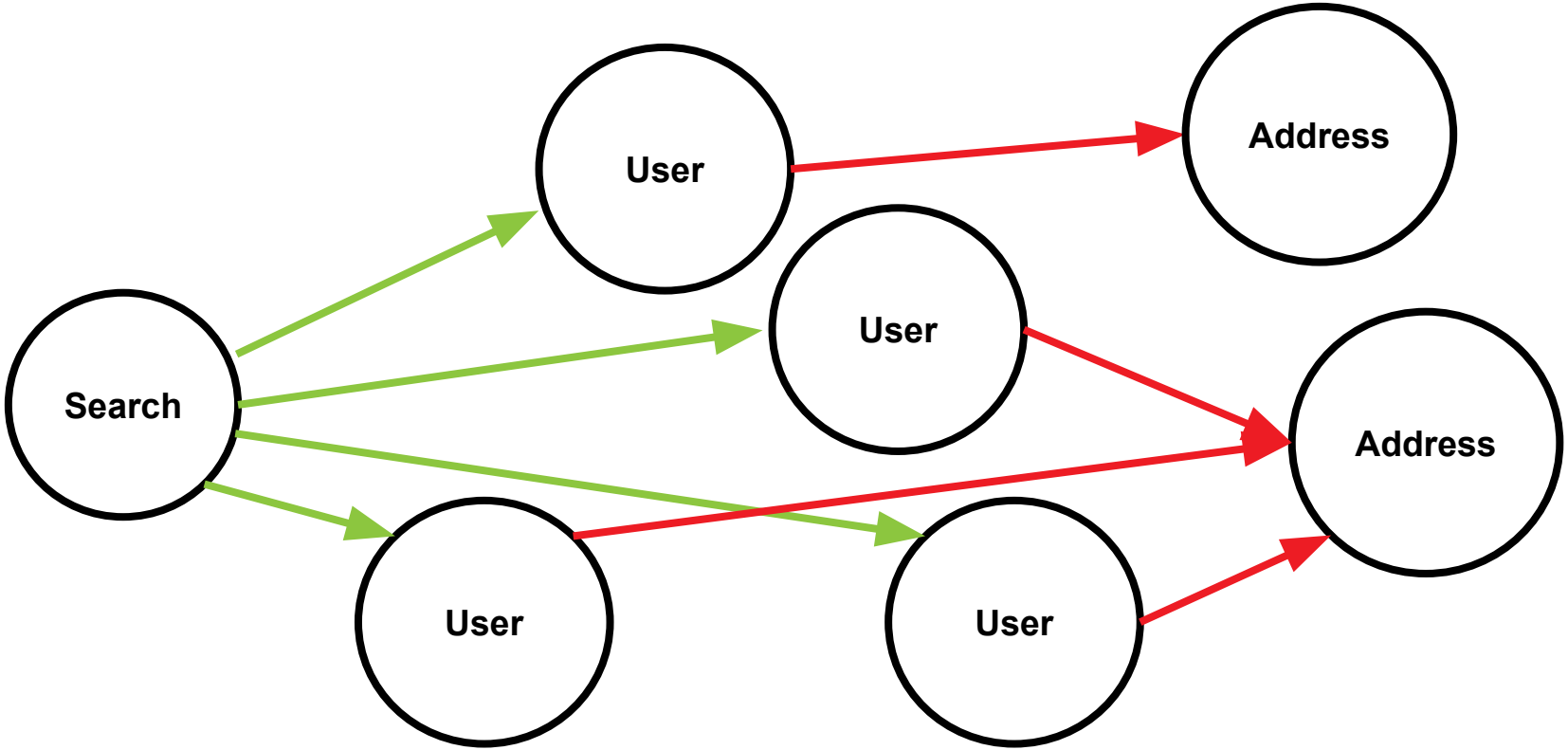
CACHE OPTIONS

- Apollo Engine
- **In memory stores**

mongoDB

N + 1 PROBLEM

N+1 PROBLEM



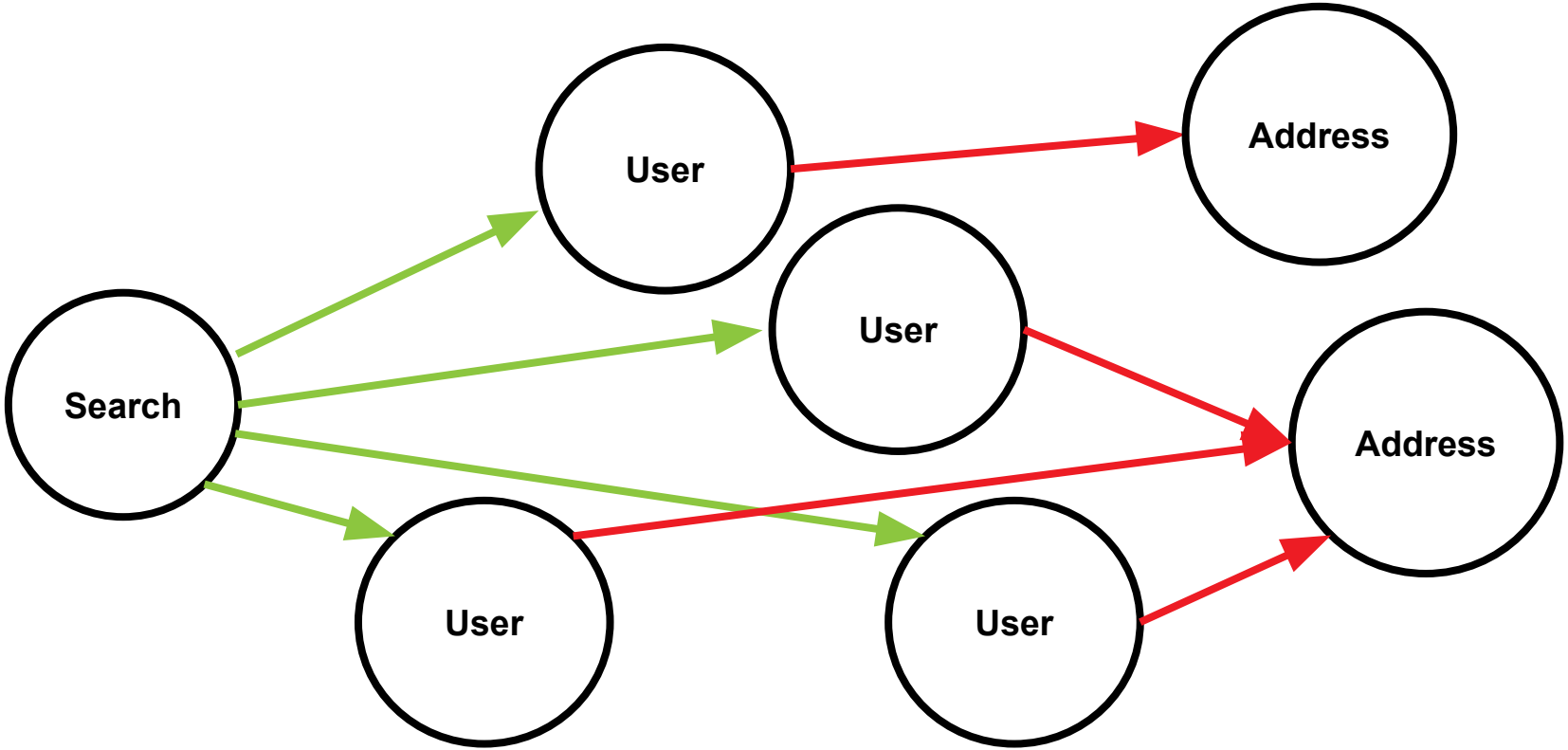
Query

```
query GetUsersList {
    userList {
        id
        address {
            id
            streetName
        }
    }
}
```

Resolvers

```
Query: {
    userList: (root) => {
        return User.all()
    }
},

User: {
    address: (user) => {
        return Address.fromId(user.addressId)
    }
}
```

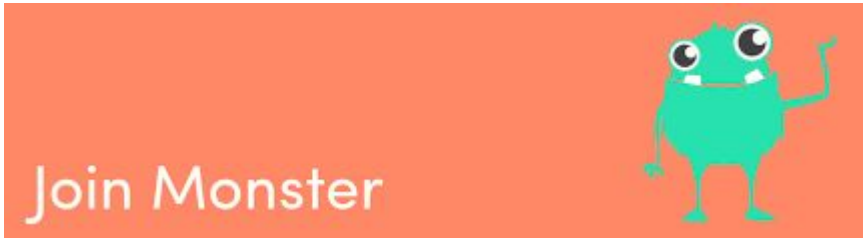
N+1 PROBLEM



CACHE OPTIONS

- Apollo Engine
- In memory stores
- **GraphQL to SQL**

PRISMA



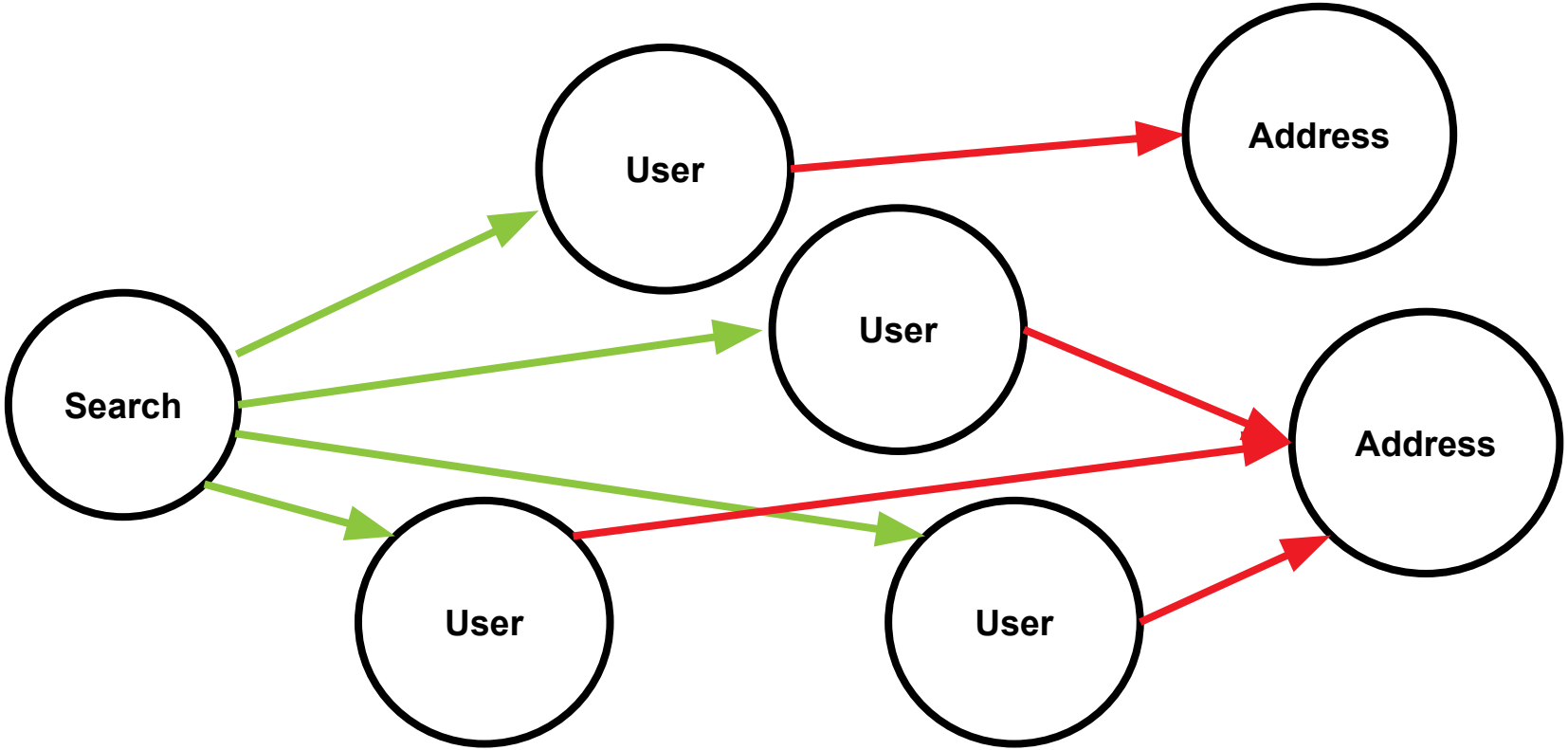
CACHE OPTIONS

- Apollo Engine
- In memory stores
- GraphQL to SQL
- **Data Loaders**

DATA LOADER

- Caching of similar requests
- Batching of single item request to grouped requests

N+1 PROBLEM



GRAPHQL IS NULL FIRST

DEALING WITH NULLS

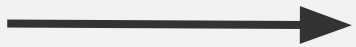
- Null is default

DEALING WITH NULLS

- Null is default
- **Hard to evolve your schema**

NULL IS DEFAULT


```
type User {
    ...
    name: String
    ...
}
```



```
type User {
    ...
    name: String!
    ...
}
```

NULL IS DEFAULT


```
type User {
    ...
    name: String!
    ...
}
```

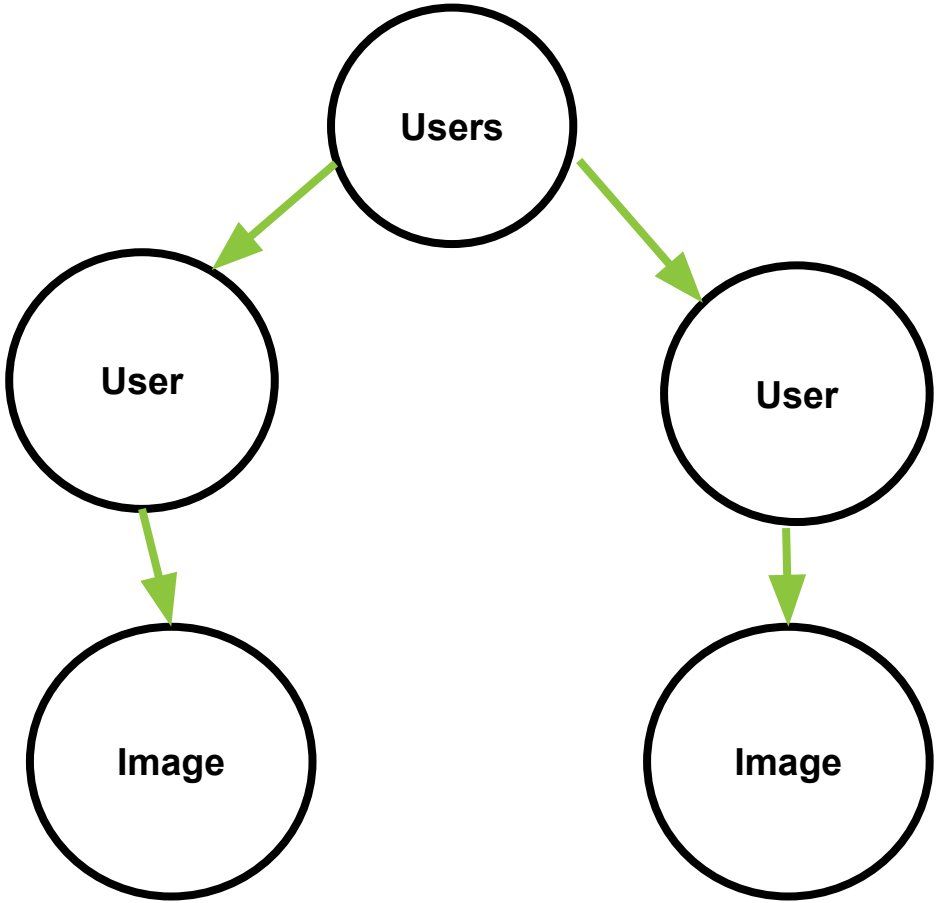
```
type User {
    ...
    name: String
    ...
}
```

DEALING WITH NULLS

- Null is default
- Hard to evolve your schema
- **Small failures have an outsized impact**

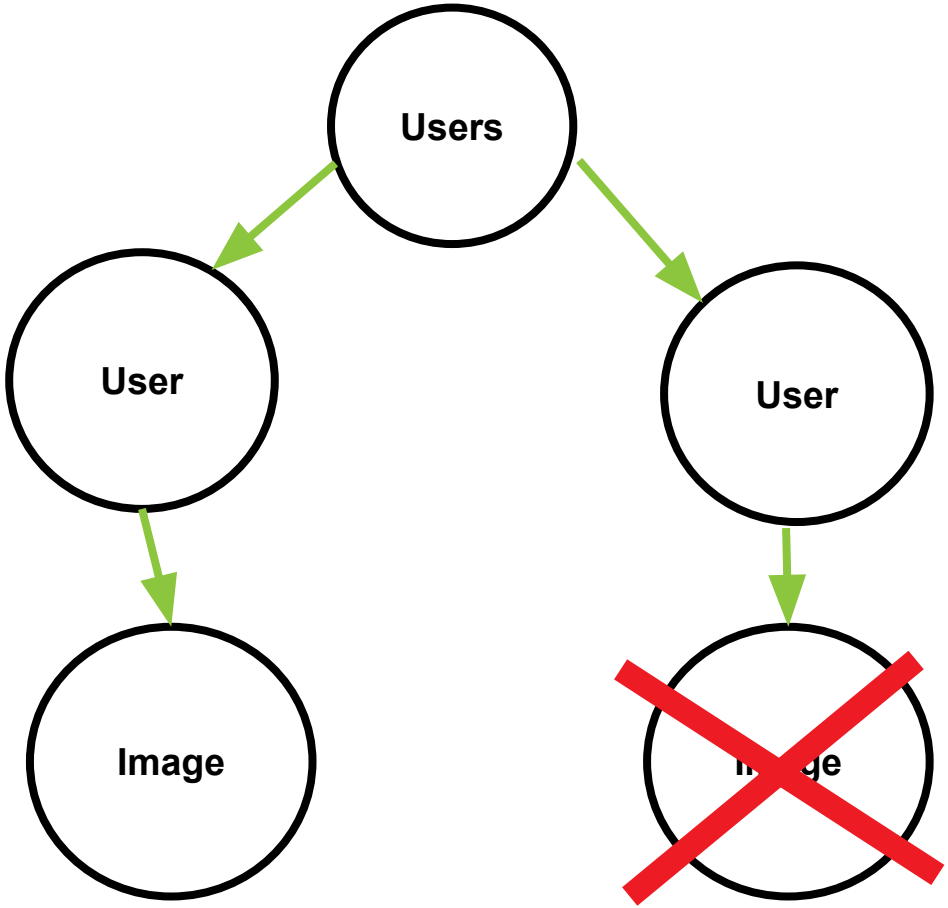
Schema


```
type User {
    id: Int!
    name: String
    imageURL: String
}

type Query {
    users: [User]
}
```



Schema


```
type User {
    id: Int!
    name: String
    imageURL: String
}

type Query {
    users: [User]
}
```



Schema

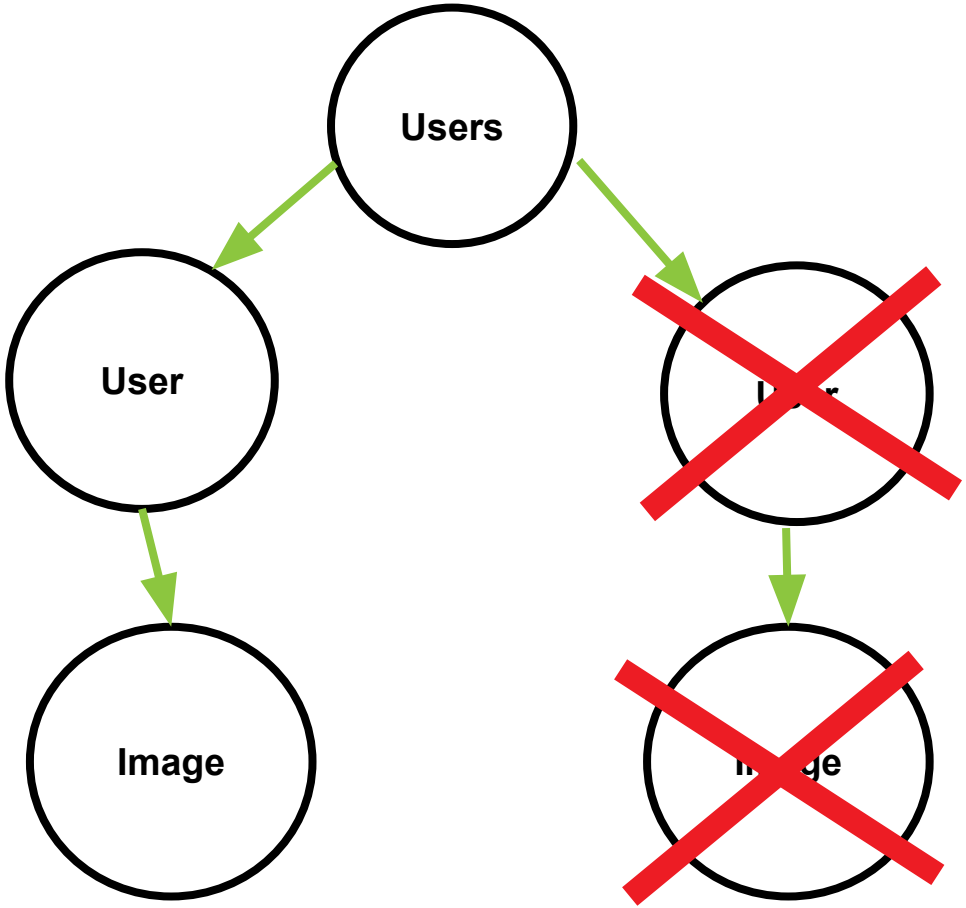
```
type User {
    id: Int!
    name: String
    imageURL: String
}

type Query {
    users: [User]
}
```

Response

```
"users": [
    {
        "id": 1,
        "name": "Sara Smith",
        "imageURL": null
    },
    {
        "id": 2,
        "name": "John Smith",
        "imageURL": "image-2.jpeg"
    }
]
```

Schema


```
type User {
    id: Int!
    name: String
    imageURL: String!
}

type Query {
    users: [User]
}
```



Schema

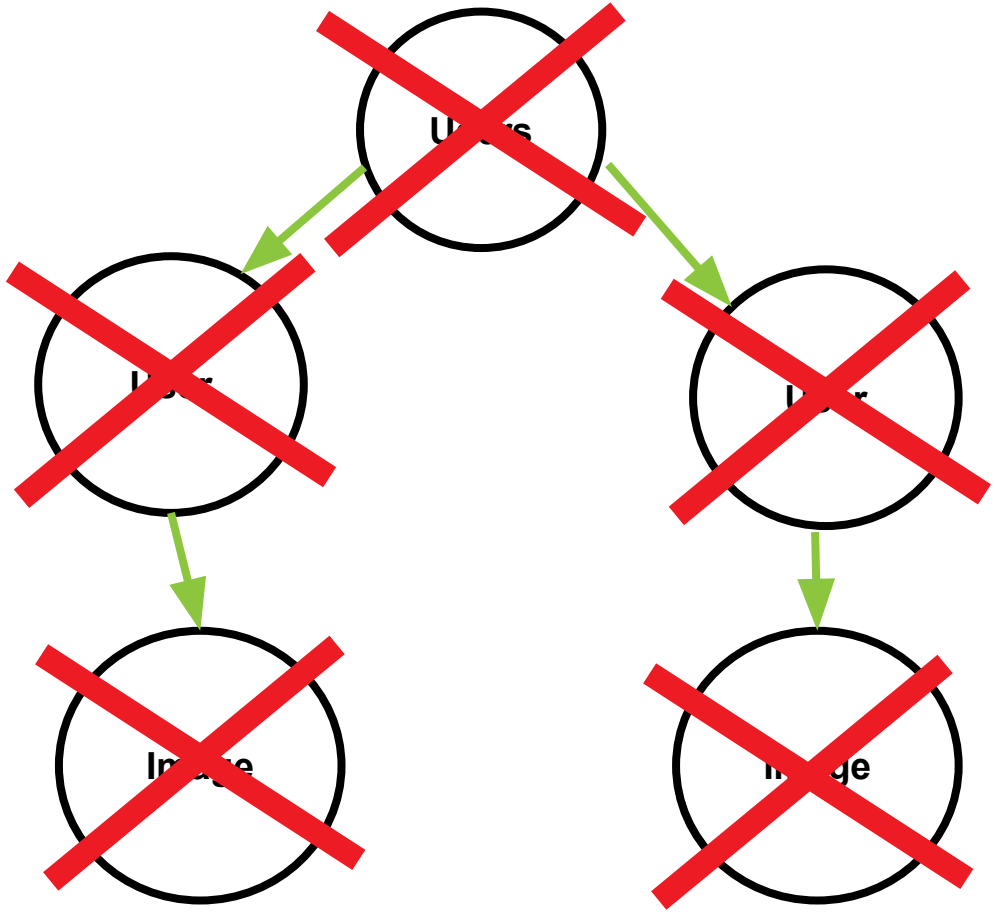
```
type User {
    id: Int!
    name: String
    imageURL: String!
}

type Query {
    users: [User]
}
```

Response

```
"users": [
    null,
    {
        "id": 2,
        "name": "John Smith",
        "imageURL": "image-2.jpeg"
    }
]
```

Schema


```
type User {
    id: Int!
    name: String
    imageURL: String!
}

type Query {
    users: [User!]
}
```



Schema

```
type User {
    id: Int!
    name: String
    imageURL: String!
}

type Query {
    users: [User!]
}
```

Response

```
"users": null
```

ERROR HANDLING

ERROR HANDLING

- 401 is null

Schema

```
type User {
    id: Int!
    name: String
    imageURL: String!
    myComments: [Comment]
}
```

Response

```
"user": {
    "id": 9722695,
    "name": "Tom",
    "imageUrl": "image-2.jpeg",
    "myComments": [...]
}
```

Schema

```
type User {
    id: Int!
    name: String
    imageURL: String!
    myComments: [Comment]
}
```

Response

```
"user": {
    "id": 9722695,
    "name": "Tom",
    "imageUrl": "image-2.jpeg",
    "myComments": null
}
```

ERROR HANDLING

- 401 is null
- **Client errors as part of payload**

BAD

```
data {
    user {  #succeeds
        name
    }
    jobSearch { #fails
        results
    }
}
errors {
    message
}
```

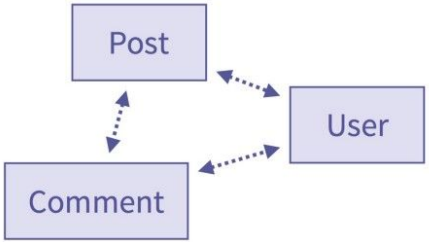
GOOD

```
data {
    user {  #succeeds
        name
    }
    jobSearchPayload { #fails
        error {
            message
        }
        jobSearch {
            results
        }
    }
}
```

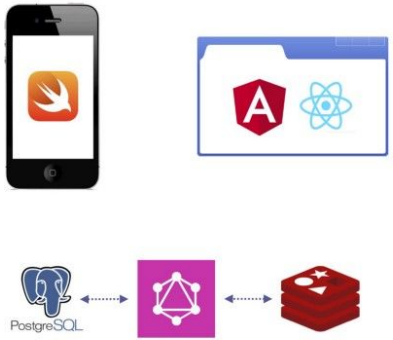
ERROR HANDLING

- 401 is null
- Client errors as part of payload
- **Other errors should be sanitized**

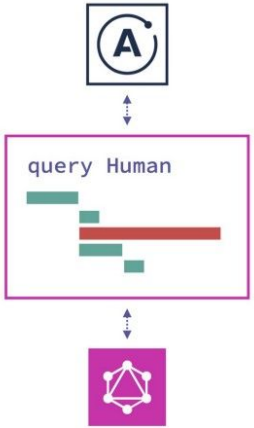
GRAPHQL DEVELOPMENT



1. Design API Schema



2. Build UI and Backend



3. Run in production

PROTECT GRAPHQL SCHEMA

SECURITY

- Timeout

SECURITY

- Timeout
- **Maximum Query Depth**

MAXIMUM QUERY DEPTH


```
query IAmEvil {
   author(id: "abc") {
       posts {
           author {
               posts {
                   author {
                       posts {
                           author {
                               # that could go on as deep as the client wants!
```

SECURITY

- Timeout
- Maximum Query Depth
- **Query Complexity**

QUERY COMPLEXITY


```
query {
   author(id: "abc") {  # complexity: 1
       posts(first: 5) {# complexity: 5
           title        # complexity: 1
       }
   }
}
```

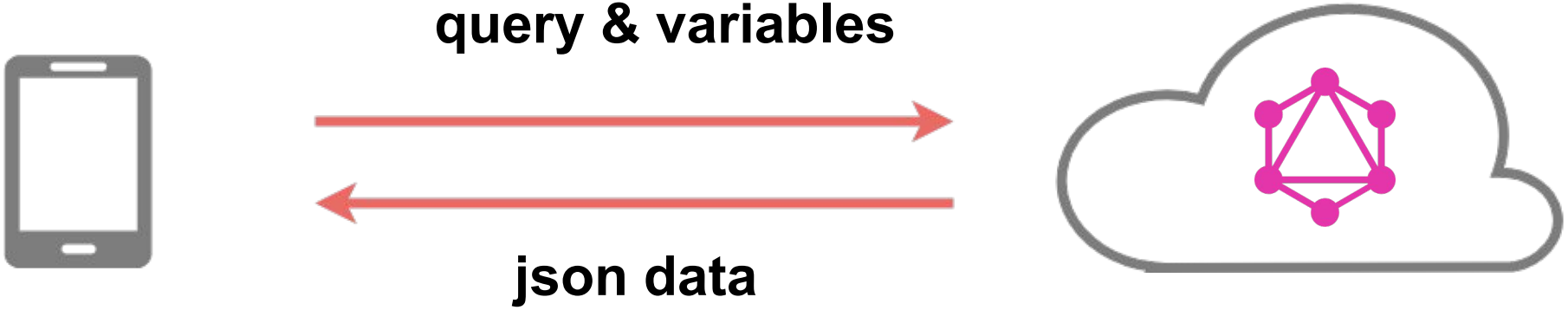
```
query {
   author(id: "abc") {  # complexity: 1
       posts {          # complexity: ?
           title        # complexity: 1
       }
   }
}
```

SECURITY

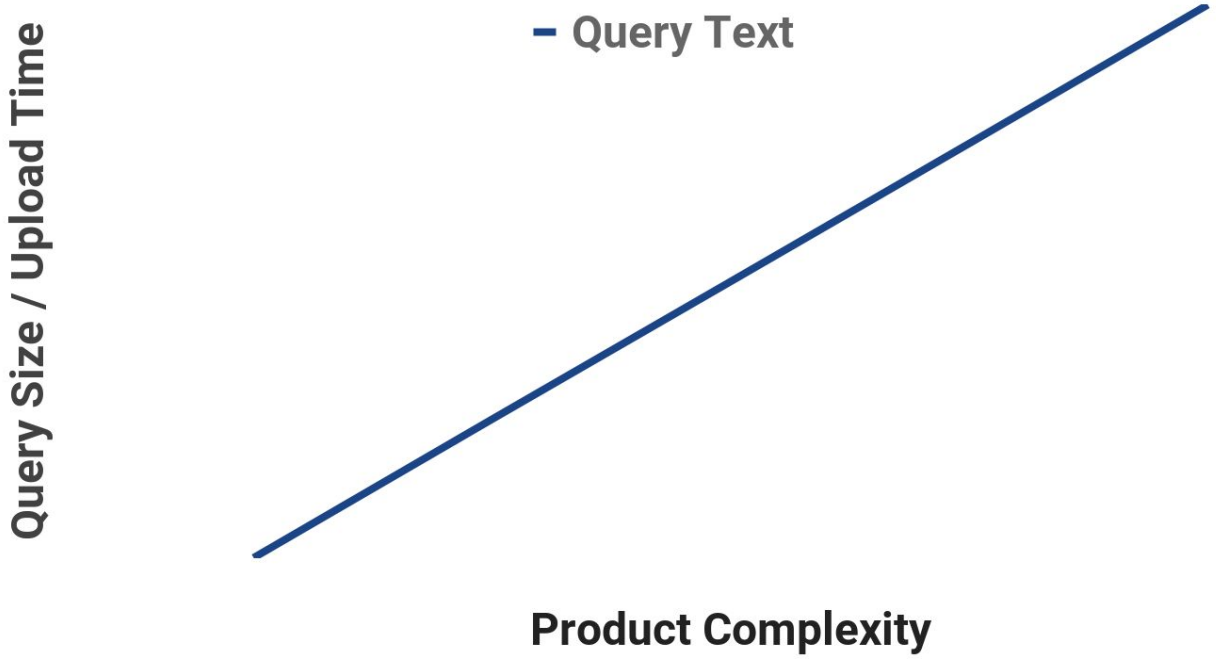
- Timeout
- Maximum Query Depth
- Query Complexity
- **Throttling (requests per minute)**

PERSISTED QUERIES

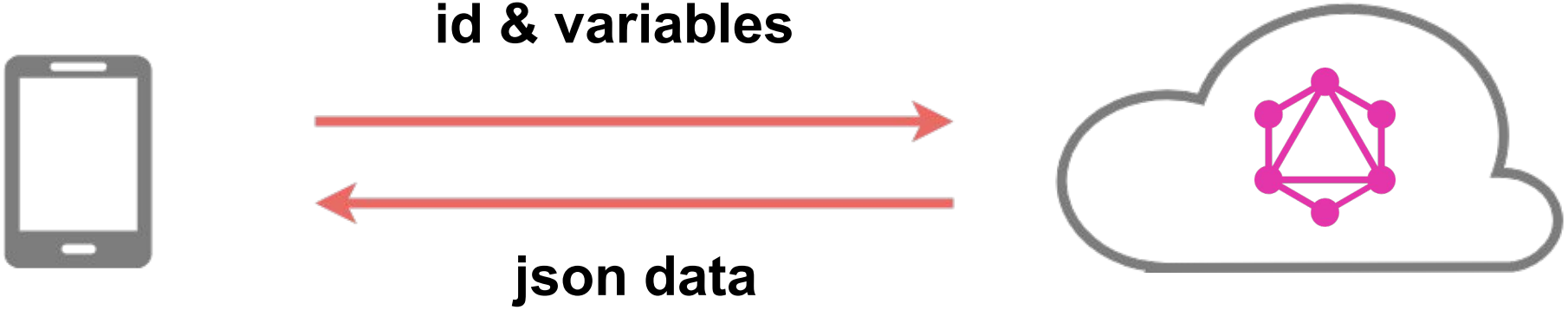
PERSISTED QUERIES



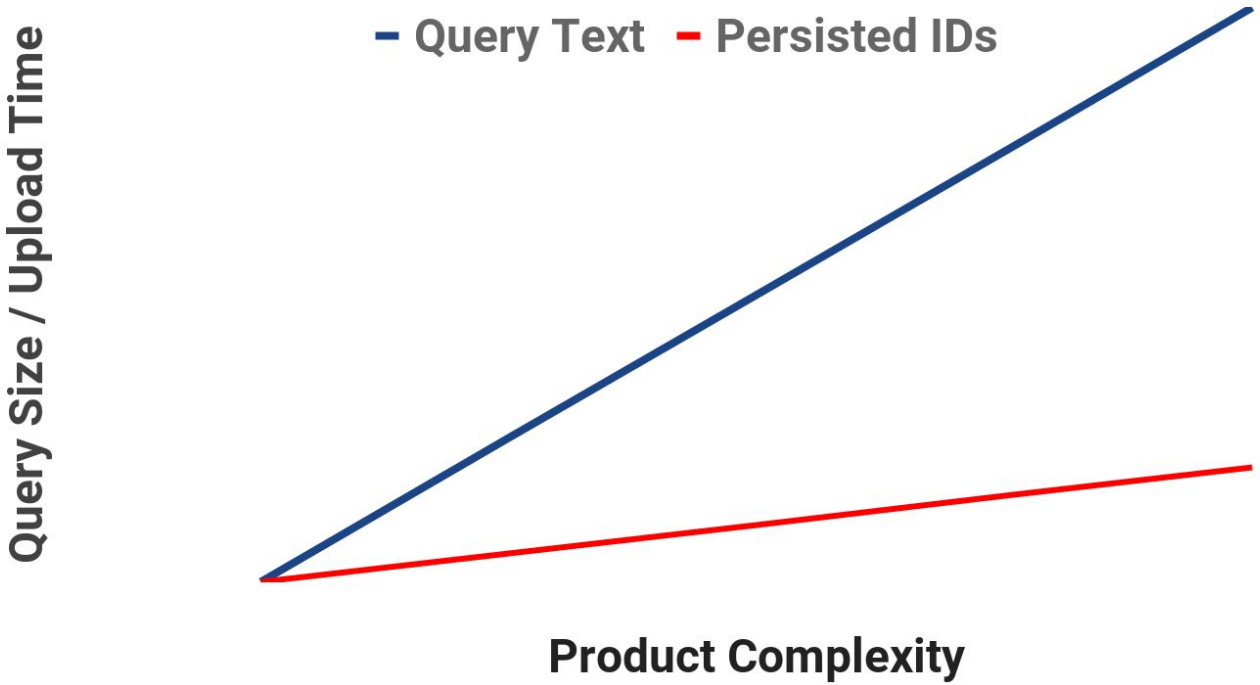
PERSISTED QUERIES



PERSISTED QUERIES



PERSISTED QUERIES



PERSISTED QUERIES

1. Resolve security problems

PERSISTED QUERIES

1. Resolve security problems
2. **Optimize upload time**

PERSISTED QUERIES

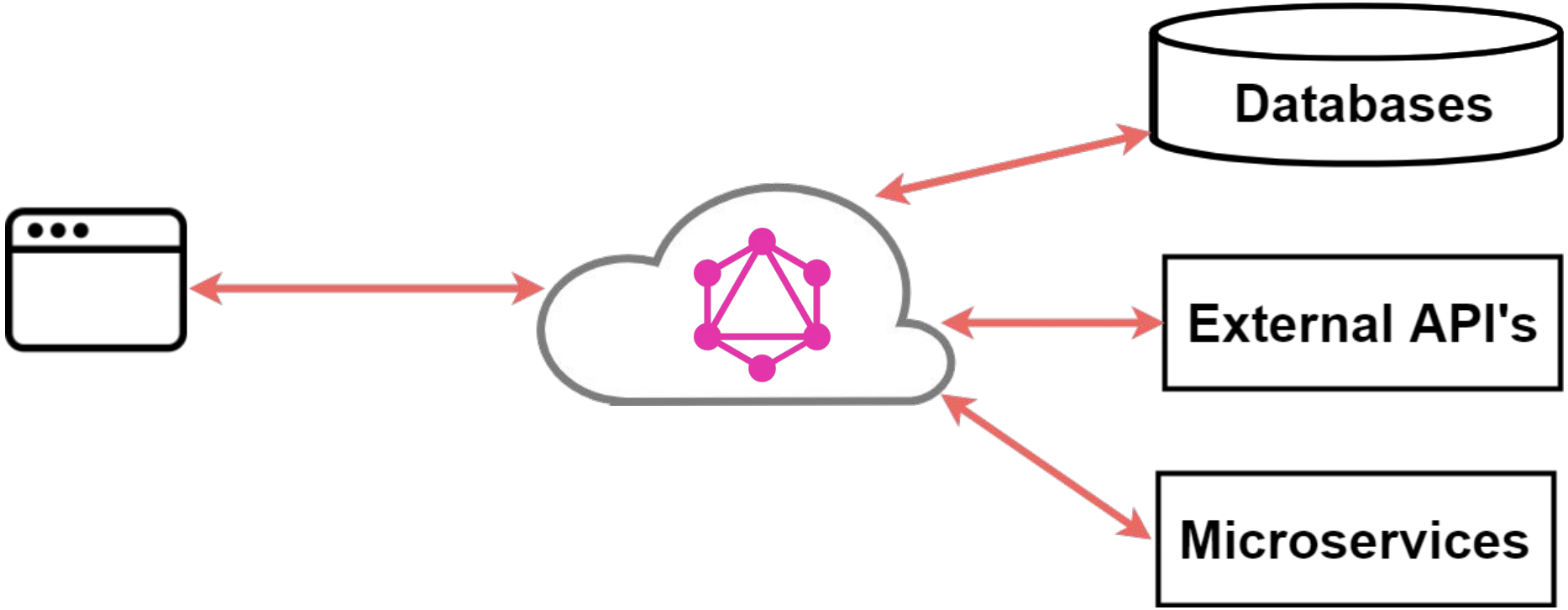
1. Resolve security problems
2. Optimize upload time
3. **Debugging**

PERSISTED QUERIES

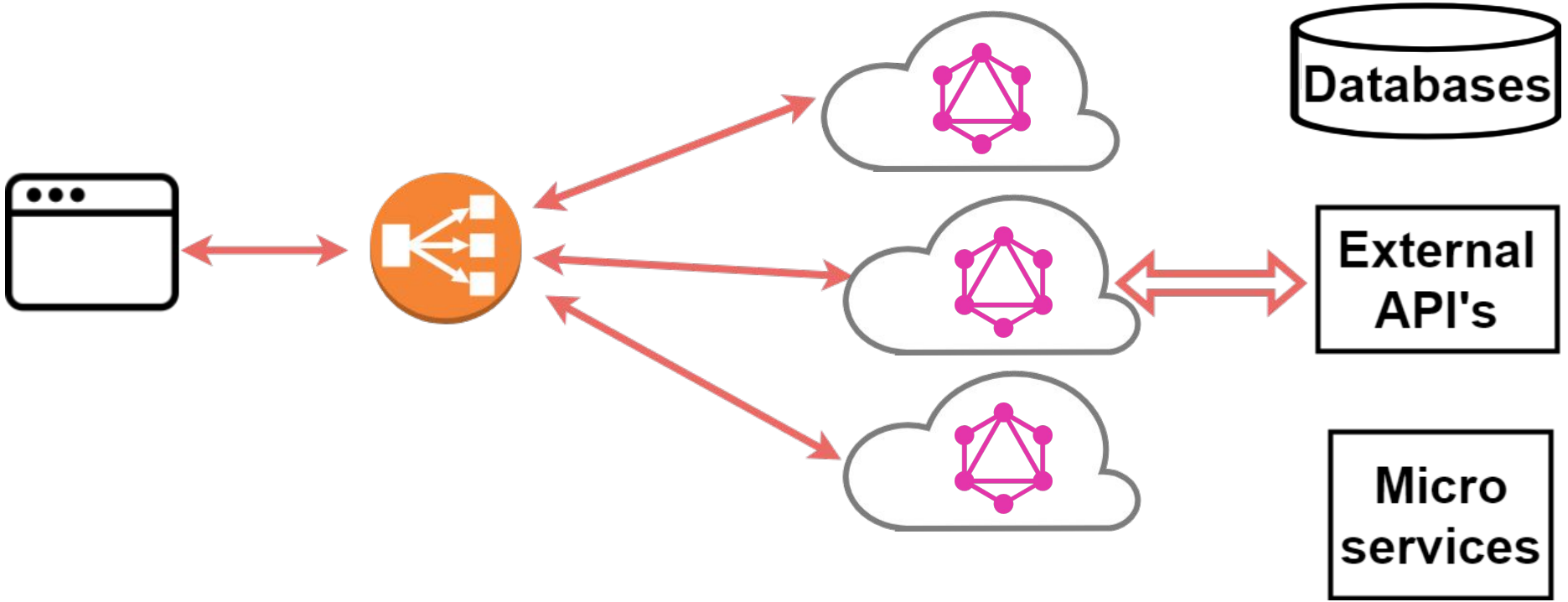
1. Resolve security problems
2. Optimize upload time
3. Debugging
4. **Supported by Apollo**

MICROSERVICES & SCALABILITY

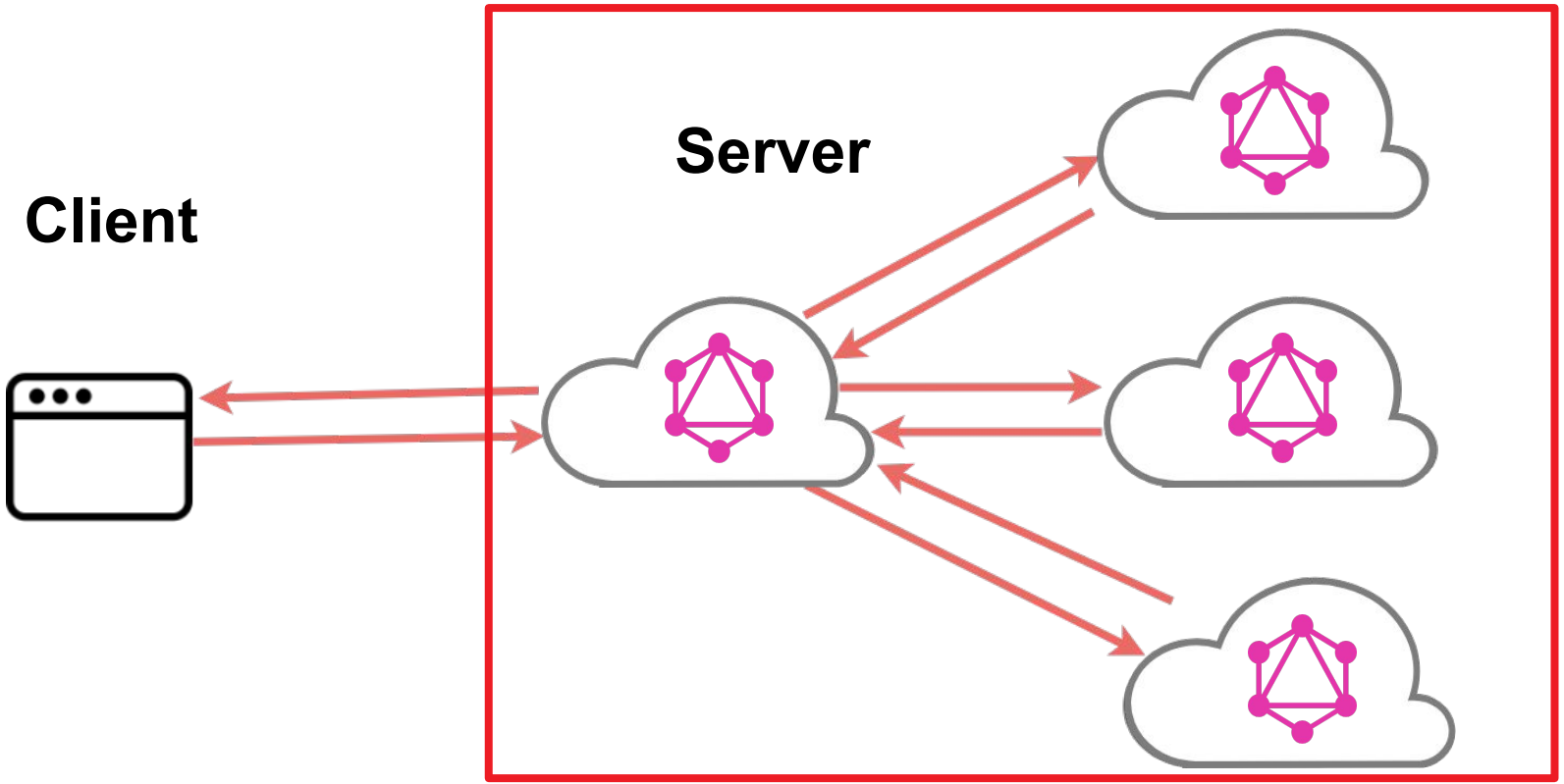
MICROSERVICES



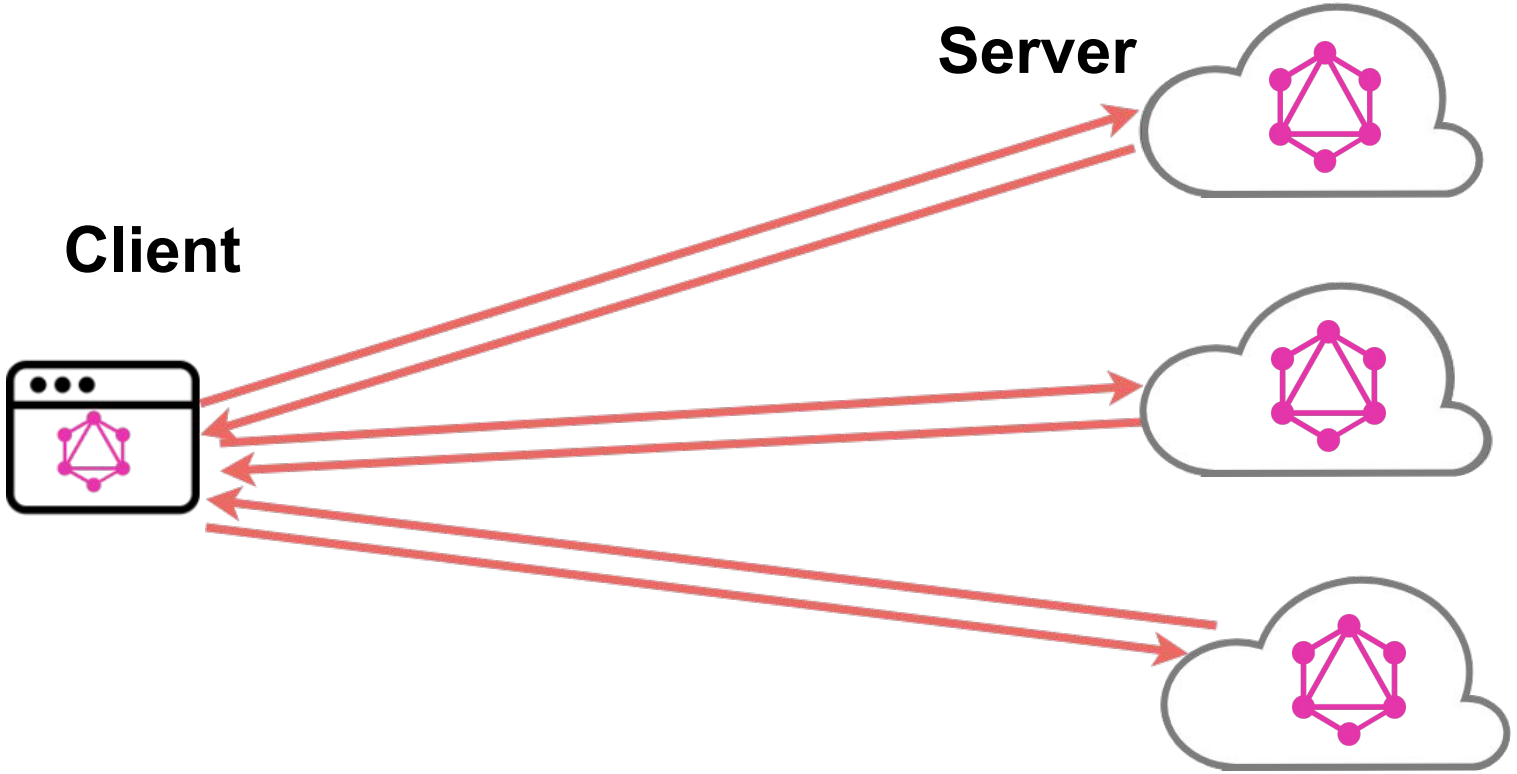
SCALABILITY



GOOD ARCHITECTURE



COMPROMISE ARCHITECTURE



GRAPHQL is not replacement

GRAPHQL is alternative

DO YOU REALLY NEED IT?

→ Ask yourself

REST IS STILL FINE

1. REST is great for service communication

REST IS STILL FINE

1. REST is great for service communication
2. **Versionizing**

REST IS STILL FINE

1. Rest is great for service communication
2. Versionizing
3. **Pulling part of fields**

PART OF FIELDS

FIELDS

```
GET /articles?fields[articles]=title,body
GET /frankcarter?fields=id,name,picture
GET /users/uploads?fields=entry(title,gd:comments,yt:statistics)
```

CUSTOM QUERIES

```
GET /Airports?$filter=contains(Location/Address, 'San Francisco')
```

REST IS STILL FINE

1. Rest is great for service communication
2. Versionizing
3. Pulling part of fields
4. **Resources**

WHEN USE GRAPHQL

1. Medium or big command
2. Two or more clients
3. Reduction in data volume
4. Standard way to expose data and operations
5. Community and ecosystem of libraries

Users


coursera


SERVERLESS


Product Hunt


amazon
web services


IBM


yelp


METEOR


FT
FINANCIAL
TIMES

END OF THE STATUS QUO

We finally have alternative to REST

Resources

- graphql.org
- www.apollographql.com
- facebook.github.io/relay
- github.com/facebook/dataloader
- join-monster.readthedocs.io
- altair.sirmuel.design

Thank you!

→ Questions?