Client-Server Communication with GraphQL

Web Engineering, Guest lecture, 188.951 2VU SS20

Erik Wittern | erikwittern@gmail.com | @erikwittern | wittern.net

May 18th 2020

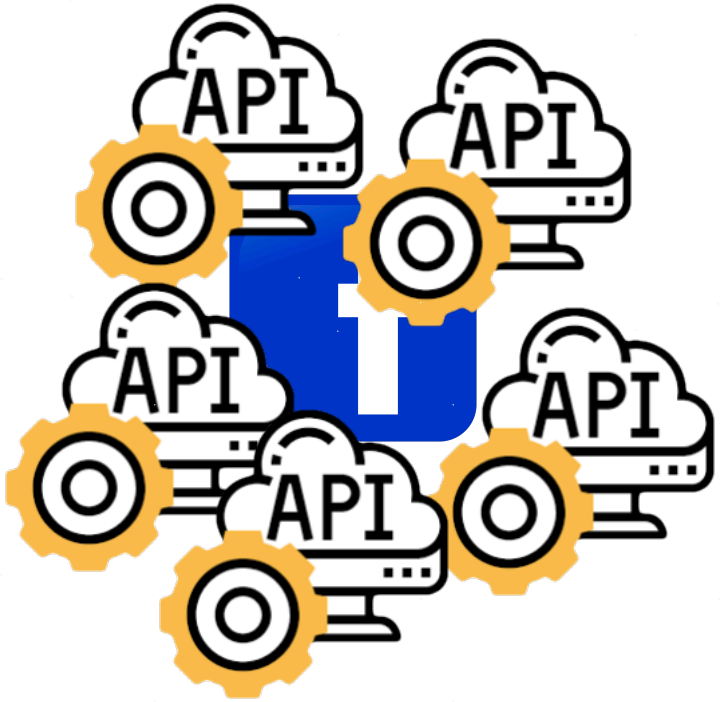
Learning goals

- What actually *is* GraphQL, and how came it about?
- How does GraphQL work?
- What are some benefits vs. challenges for GraphQL?

Background & Overview

In 2012, Facebook faced a problem

An increasing number of ever-evolving (mobile, native) clients…



…led to the creation of (and hence maintenance burden for) more and more, increasingly complex (ad hoc) API endpoints.

Source: https://reactjs.org/blog/2015/05/01/graphql-introduction.html

GraphQL shifts control over what data is returned (or mutated) to clients

Providers define their data types at design time

```
type User {
  name: String
  age: Int
}

type Query {
  me: User
}
```

Clients send queries at runtime

```
query {
  me {
    name
  }
}
```

Servers respond with data at runtime

```
{
  "data" : {
    "me" : {
      "name" : "Erik"
    }
  }
}
```

Demo

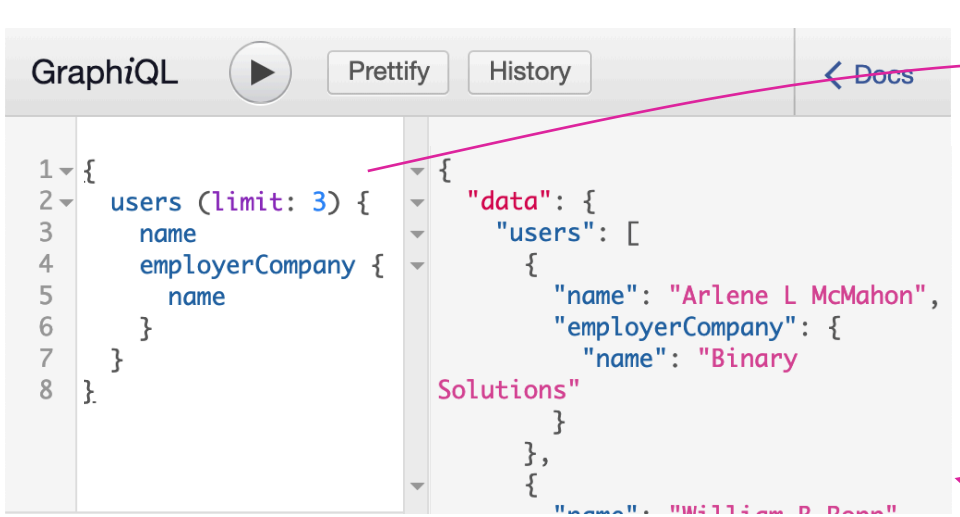
https://www.github.com/ErikWittern/graphql-demo

So, what is GraphQL?

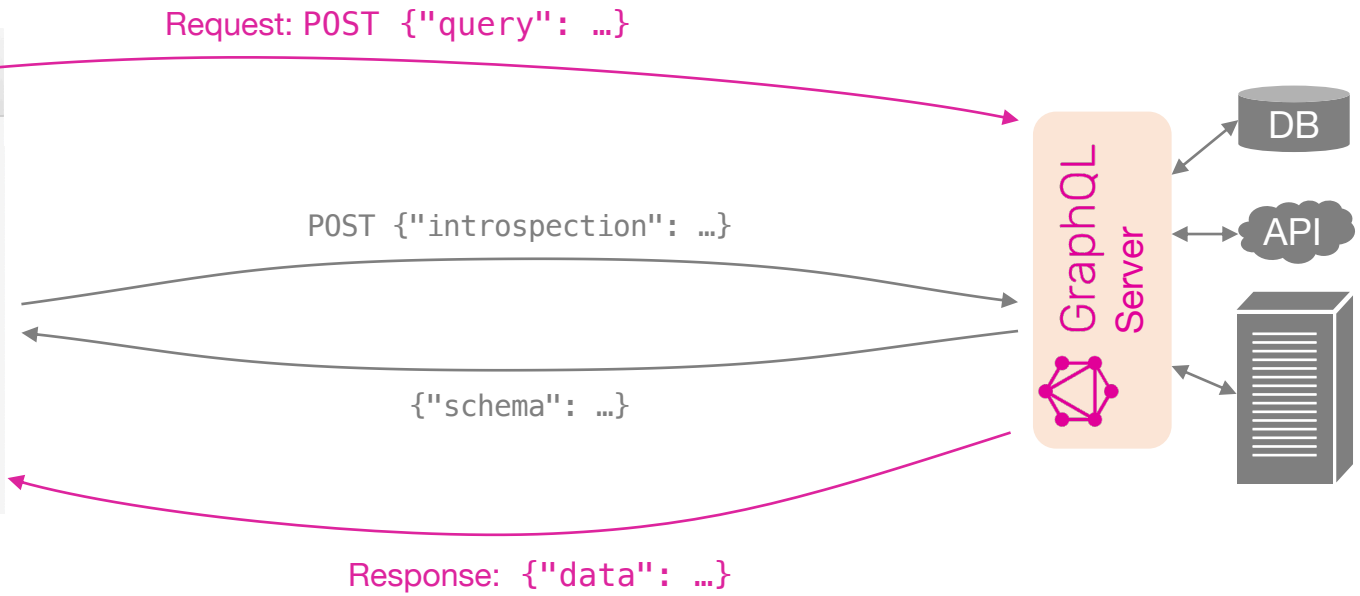
- A **query language** for networked APIs…
- …and a **runtime** for servers to fulfill queries

- Specification + reference implementation in JavaScript

- Clients send type-checked queries, servers respond with requested data:

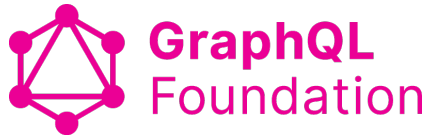


GraphQL client (here: Graph*i*QL)



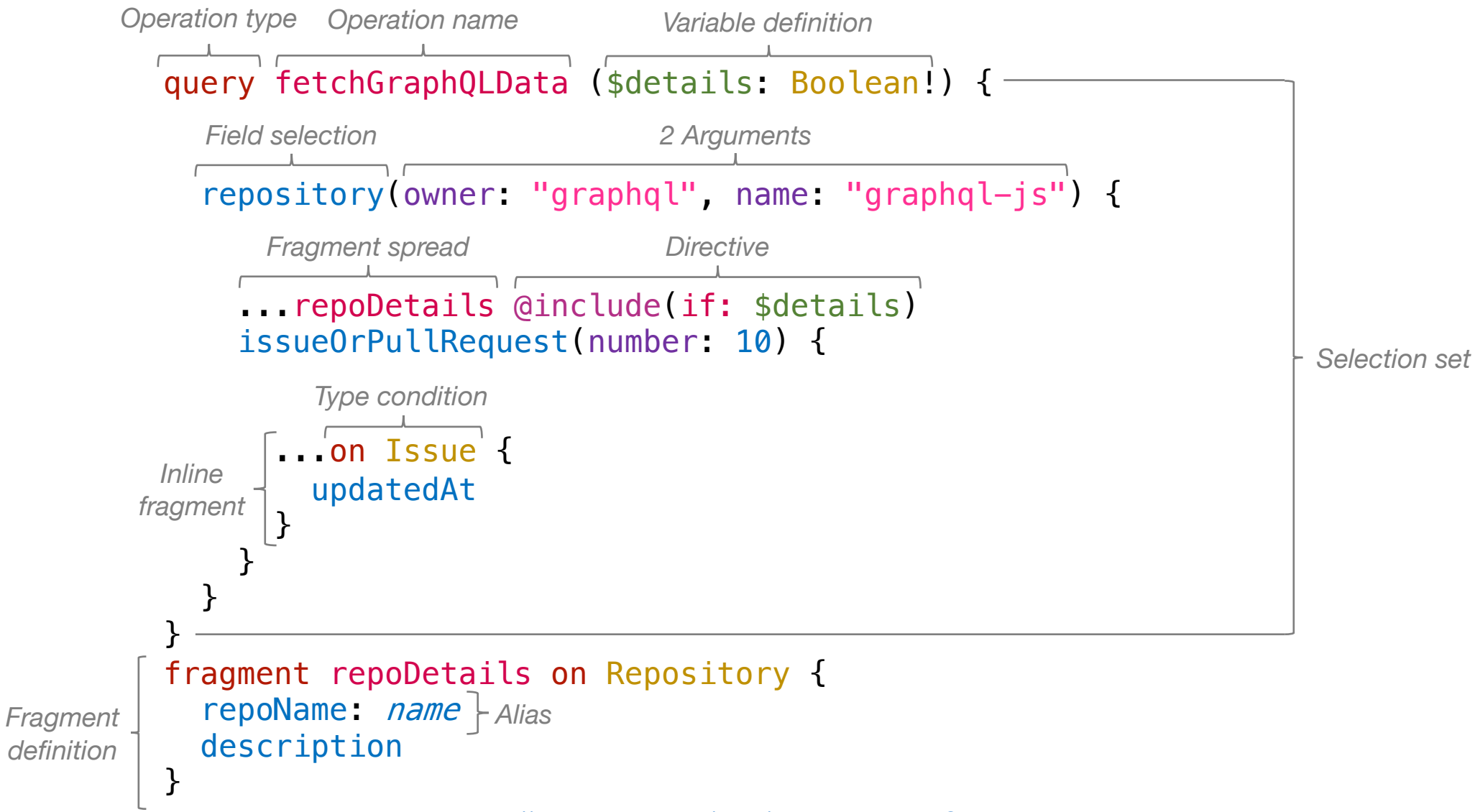
History of GraphQL

- 2012 – Originally developed and used by Facebook
  - …to serve increasing numbers of diverse clients
- Sep 2015 – Open sourcing
- Sep 2016 – Move from “technical preview” to “working draft”
- Nov 2018 – Announcement of GraphQL Foundation (part of The Linux Foundation)



Language & Runtime

Anatomy of a GraphQL query (selected concepts)



More details at: http://spec.graphql.org/draft/#sec-Document-Syntax

Defining schemas with the *schema definition language* (SDL)

Schema definition

```
schema {
  query: Query
}
```

Root operation type definition

Object type definition

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

Name *Argument definition* *Type* *Field definition*

Directive definition

```
directive @upperCase on FIELD_DEFINITION
```

Name *Directive locations*

Enum type definition

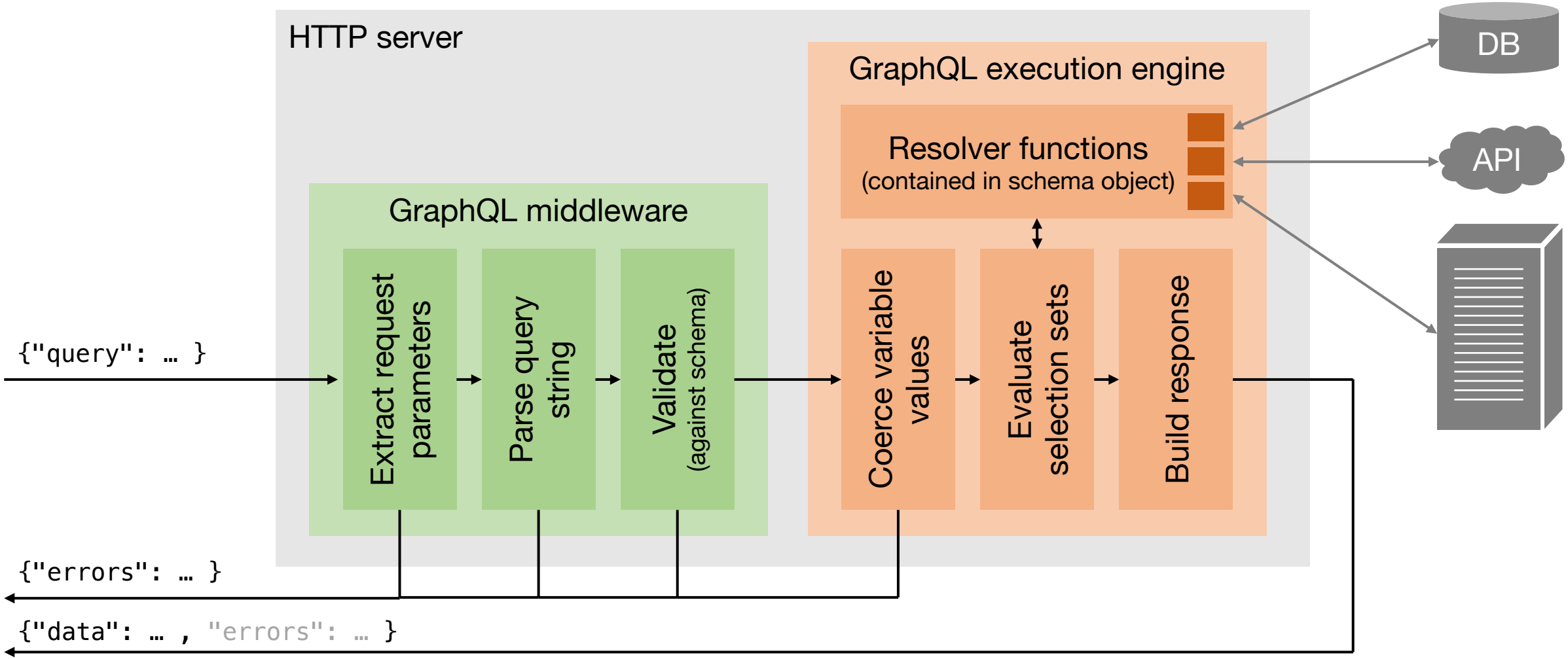
```
enum Status {
  ACTIVE
  INACTIVE
}
```

```
type User {
  name: String @upperCase
  """Description of field User.status"""
  status: Status
}
```

Directive

More details at: http://spec.graphql.org/draft/#sec-Document-Syntax

Query exeuction on a (HTTP) server



Advanced Query Concepts

Introspection

- Introspection is a mechanism for clients to learn (at runtime) about the data types and operations a GraphQL server offers

- An *introspection query* is a plain-old GraphQL query…
- …that happens to select *meta-fields* provided by *introspection types*

- Client-tools like GraphiQL rely on introspection for:
  - Showing documentation about types & operations
  - Client-side query validation
  - Auto-completion when typing queries
  - Etc.

```
query IntrospectionQuery {
  __schema {
    queryType { name }
    mutationType { name }
    subscriptionType { name }
    types {
      ...FullType
    }
    directives {
      name
      locations
      args {
        ...InputValue
      }
    }
  }
}

... Directive Definitions ...
```

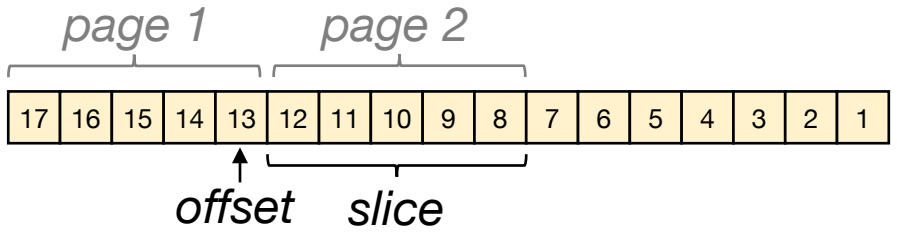
Demo

https://developer.github.com/v4/explorer/

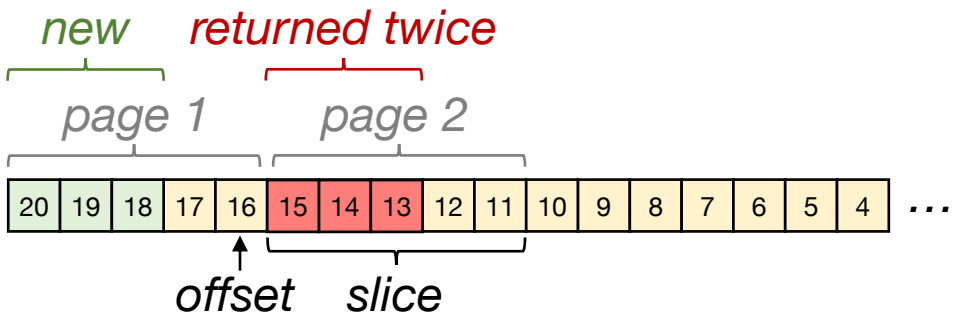
Pagination with slicing arguments and offset

- Pagination aims to return different parts (or *slices*) of long lists of data
- Slicing arguments (often named `max`, `limit` , `first`, or `last`) define length of slice to return
- Often combined with an “offset”

```
query fetchPage2 {
  user {
    name
    friends(last: 5, offset: 5) {
      name
} } }
```



- One problem: this approach may return items twice when list updates

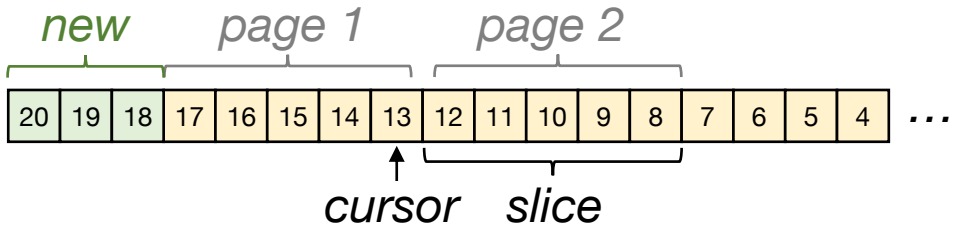


Pagination with Cursor Connections

- Cursor Connections rely on…
  - Fields using slicing arguments…
  - …return a `Connection` with fields `pageInfo` and `edges`…
  - …where each `Edge` has fields `cursor` and `node`, containing the actual object.

```
query fetchPage2 {
  user {
    name
    friends(last: 5, after: "opaqueCursor") {
      pageInfo {
        hasNextPage
      }
      edges {
        cursor
        node {
          name
        }
} } } }
```

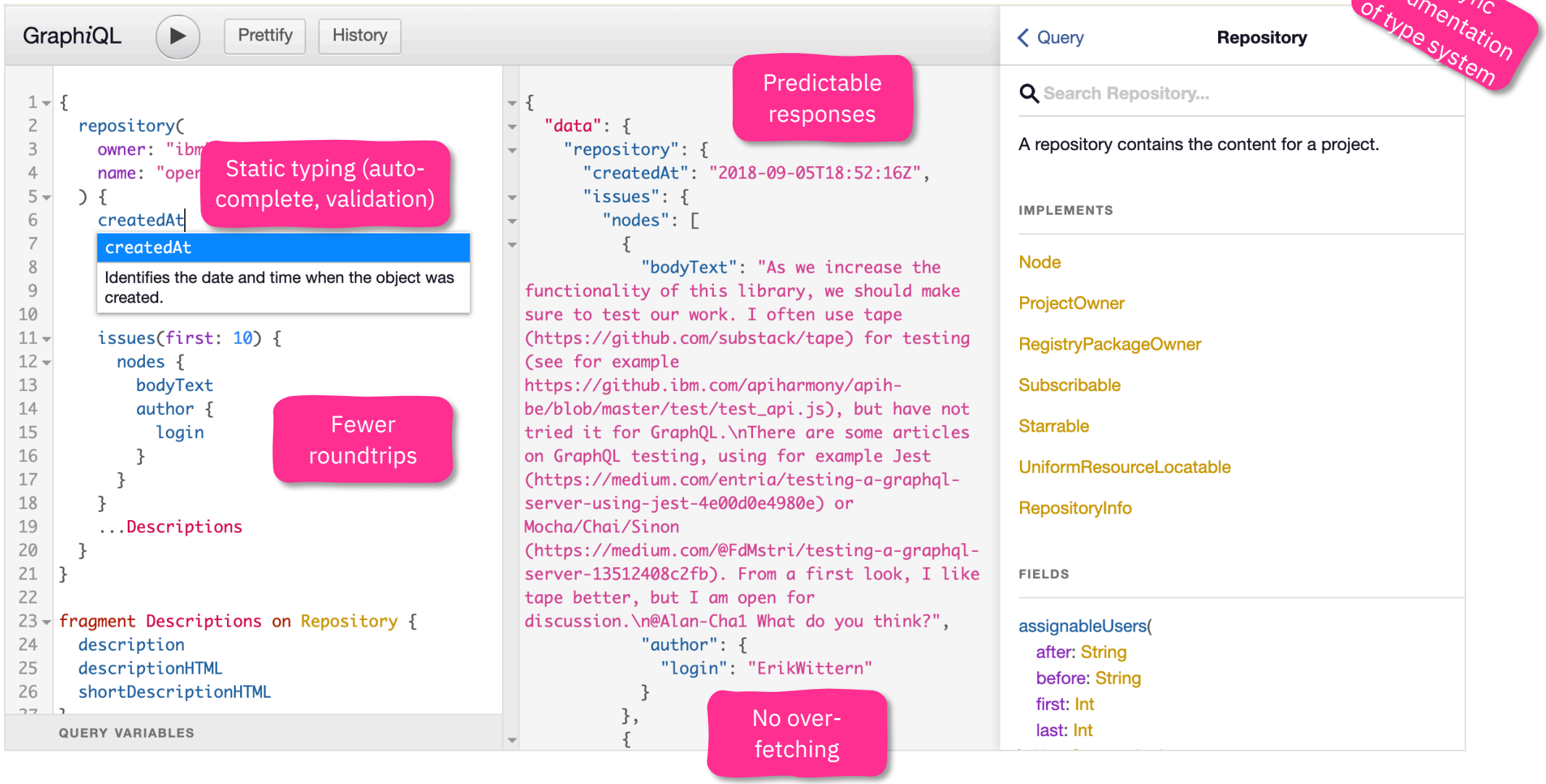
- Robust to list-updates outside the slice during paginating
- Think of a common Facebook's use-case: *news feed*, where mostly items are added



More details about cursor connections at: https://relay.dev/graphql/connections.htm (also source of the example above)

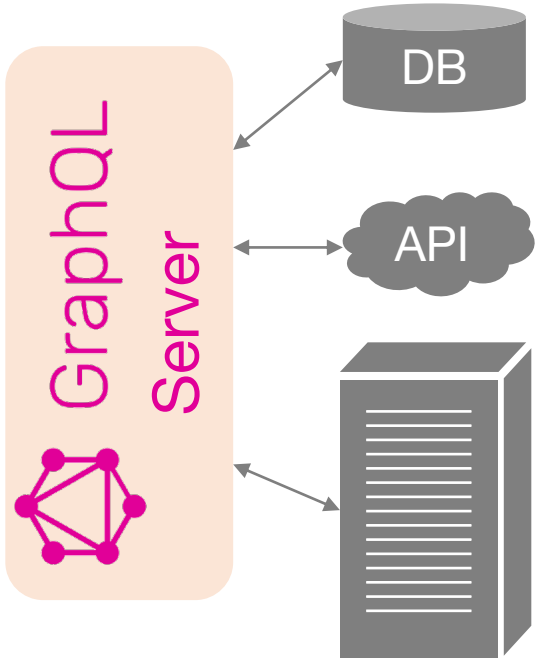
Pros & Cons

GraphQL benefits for clients



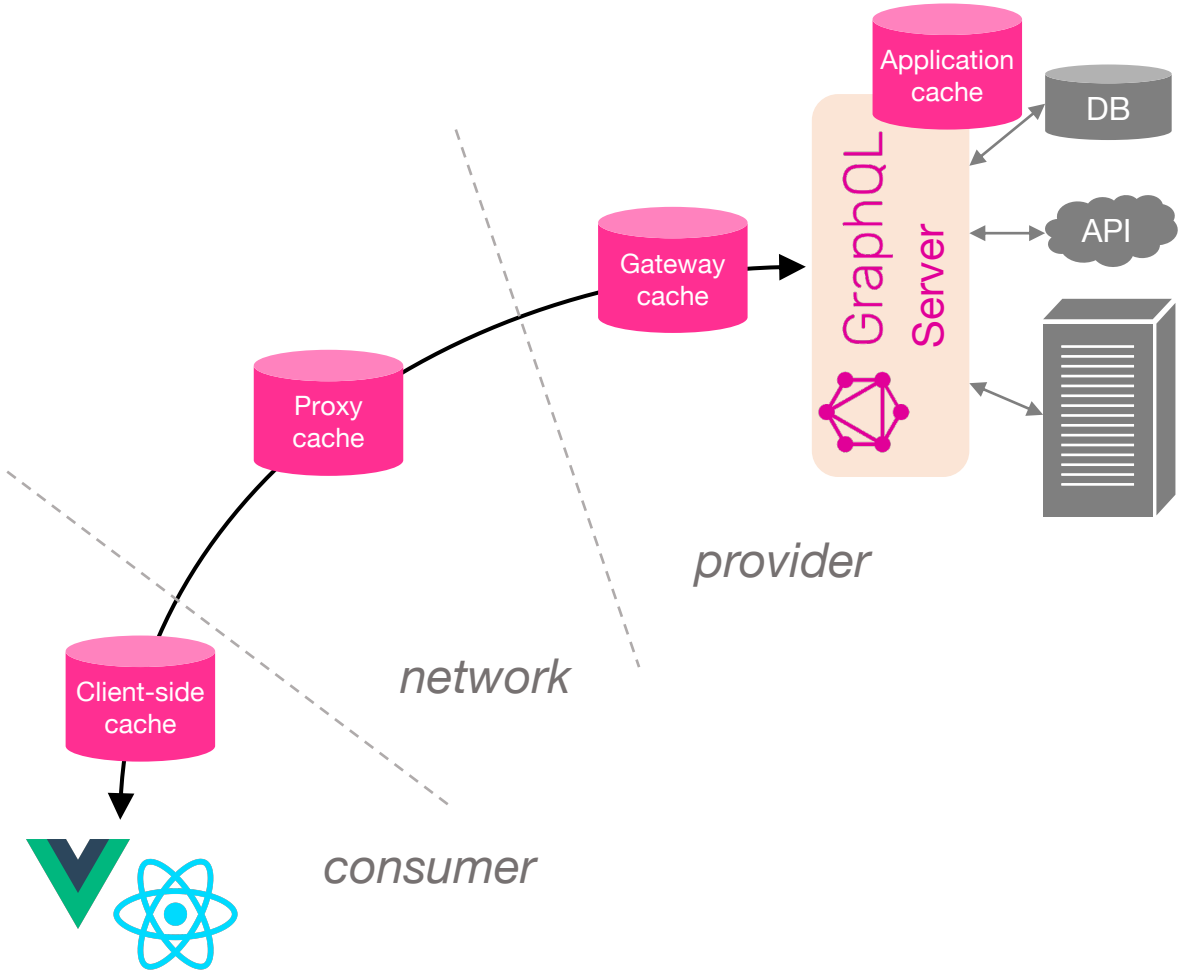
GraphQL benefits for providers

- Happy API consumers (!)
- Simplified maintenance
  - Serve clients with diverse, changing requirements with a single endpoint
  - GraphQL API self-documents types & operations
- Improved performance and operations
  - Avoid loading / caching / exposing unneeded data
  - Understand data-use on a per-field level
- Compose heterogenous backend resources



Challenge: HTTP caching of GraphQL requests

- Problems with typical HTTP proxy / gateway caches include:
  - Often, non-safe & non-idempotent `POST` is used to send (large) queries
  - Some queried fields may become stale sooner than others, making it hard to define `Cache-Control` / `Last-modified` headers
- Alternatives include:
  - Cache *persisted queries* in proxy or gateway
  - Client-side caching based on `ID` field
  - Application caches in the data-layer ("DataLoaders") or resolver functions



More details at: https://www.apollographql.com/blog/graphql-caching-the-elephant-in-the-room-11a3df0c23ad

Challenge: rate-limiting & threat prevention

- Servers may need to deal with excessive queries sent by clients
  - Rate-limiting - and not “x requests per time-interval”
  - Pricing requests
  - Blocking (inadvertently) threatening requests
- Options include:
  - Timeouts against threatening requests
  - Dynamic analysis
  - Static analysis
    - Query “depth” or “nesting”
    - Query “cost” or “complexity”

```
query fetchAllTheData {
  users (limit: 1000) {
    orders (first: 1000) {
      paymentDetails { # calls external API
        status
      }
    }
  }
}
```

= ~1000s of REST requests!

More details at: https://www.ibm.com/blogs/research/2019/02/graphql-api-management/

Wrap-up

Summary

- Remember: GraphQL was created to address specific problems with other API models
- Using GraphQL may or may not be beneficial
  - Who are API clients? Internal, external, both?
  - How is an API used?
  - How will the API (likely) evolve?
  - → consider the *trade-offs* (as with most technology choices)
- There is much more to learn about GraphQL !!
  - *Mutation* and *subscription* operations
  - (Automatic) mappings to REST APIs or databases
  - Schema stitching and federation
  - And more!

Additional resources

- **Web resources**
  - Official GraphQL website, incl. documentation (https://graphql.org/)
  - GraphQL specification (http://spec.graphql.org/)
  - Principled GraphQL (https://principledgraphql.com/)
- **Libraries**
  - GraphQL-js reference implementation (https://github.com/graphql/graphql-js)
  - OpenAPI-to-GraphQL (https://github.com/IBM/openapi-to-graphql)
  - Apollo Client (https://www.apollographql.com/client/)
  - …and many many more!!!
- **Videos**
  - “GraphQL – The Documentary” (https://www.youtube.com/watch?v=783ccP__No8)
  - “Zero to GraphQL in 30 Minutes” by Steven Luscher (https://www.youtube.com/watch?v=UBGzsb2UkeY)
- **Research papers & books**
  - “Semantics and Complexity of GraphQL” by Hartig and Perez (http://olafhartig.de/files/HartigPerez_WWW2018_Preprint.pdf)
  - “An Empirical Study of GraphQL Schemas” by Wittern et al. (http://people.cs.vt.edu/davisjam/downloads/publications/WitternChaDavisBaudartMandel-EmpiricalGraphQL-ICSOC19.pdf, summary at https://medium.com/swlh/empirical-study-graphql-icsoc19-29038c48da5)
  - ”Production ready GraphQL” by Marc-Andre Giroux (from GitHub, @__xuorig__) (https://book.productionreadygraphql.com/)

Thank you!