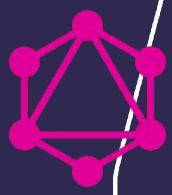
Walmart Java Guild Presents:

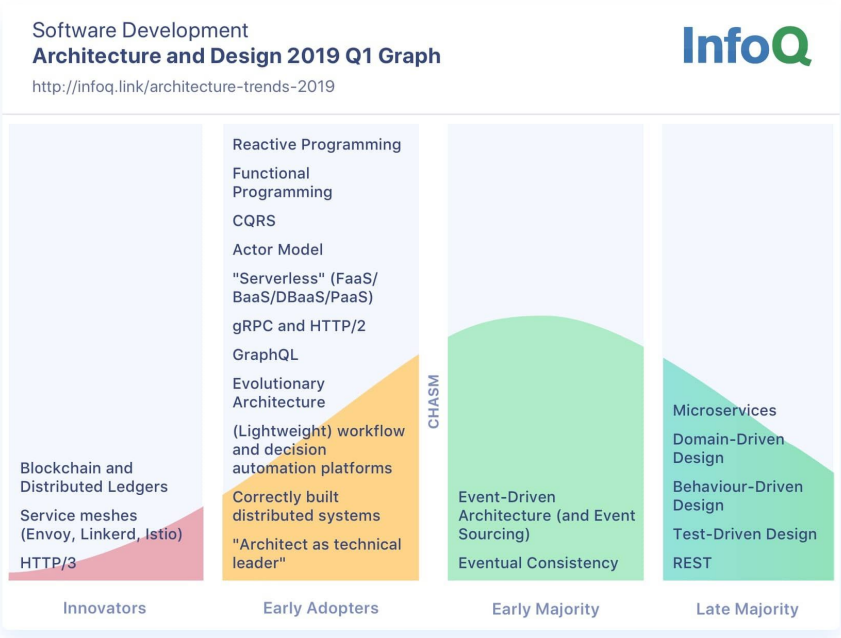
GraphQL



By John McCaulley

Quick Intro – About me!

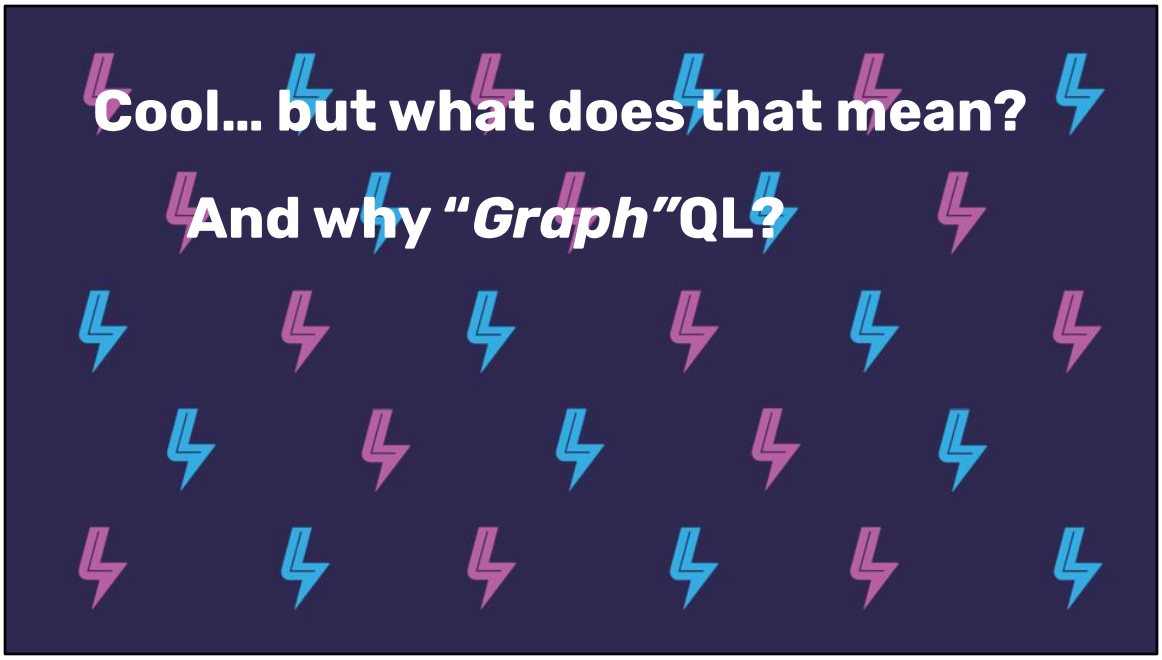
Don’t forget to pace based on how familiar everyone is.
Going to cover basics with the goal of giving everyone a good understanding of why data loaders are necessary and why we need to deal with the problems they create in java/schema design.


Software Development
Architecture and Design 2019 Q1 Graph
http://infoq.link/architecture-trends-2019
InfoQ
Blockchain and Distributed Ledgers
Service meshes (Envoy, Linkerd, Istio)
HTTP/3
Reactive Programming
Functional Programming
CQRS
Actor Model
"Serverless" (FaaS/ BaaS/DBaaS/PaaS)
gRPC and HTTP/2
GraphQL
Evolutionary Architecture
(Lightweight) workflow and decision automation platforms
Correctly built distributed systems
"Architect as technical leader"
CHASM
Event-Driven Architecture (and Event Sourcing)
Eventual Consistency
Microservices
Domain-Driven Design
Behaviour-Driven Design
Test-Driven Design
REST
Innovators
Early Adopters
Early Majority
Late Majority

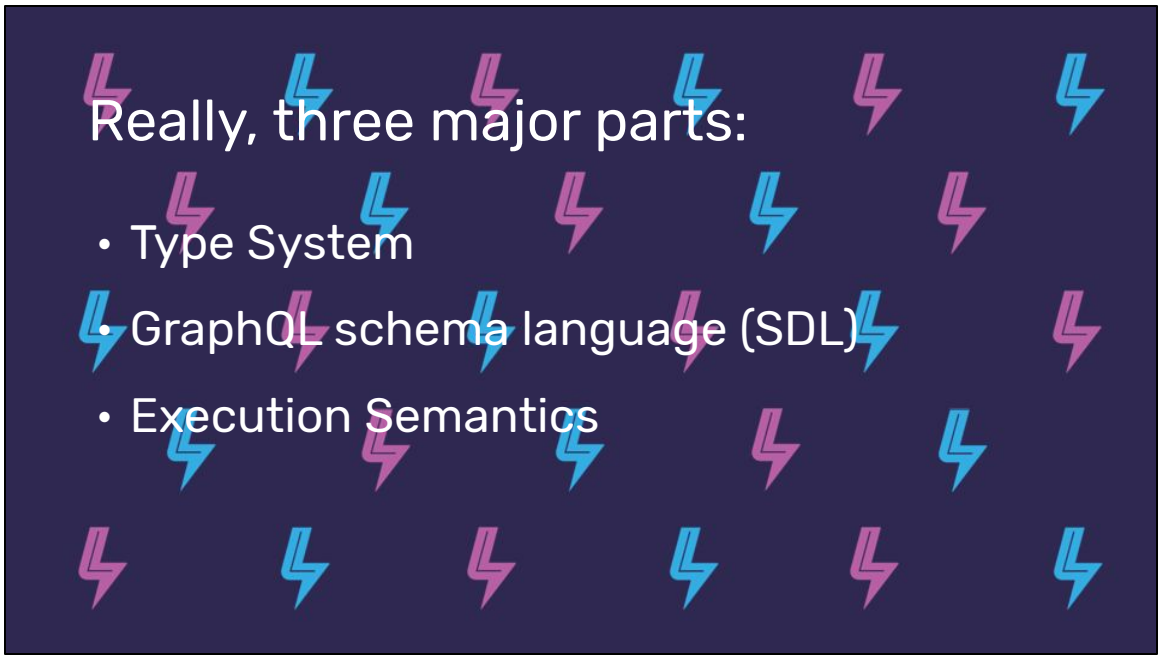
What is GraphQL?

“GraphQL is a query language for APIs and a runtime for fulfilling those queries with your existing data. GraphQL provides a complete and understandable description of the data in your API, gives clients the power to ask for exactly what they need and nothing more, makes it easier to evolve APIs over time, and enables powerful developer tools.”

First ask the crowd

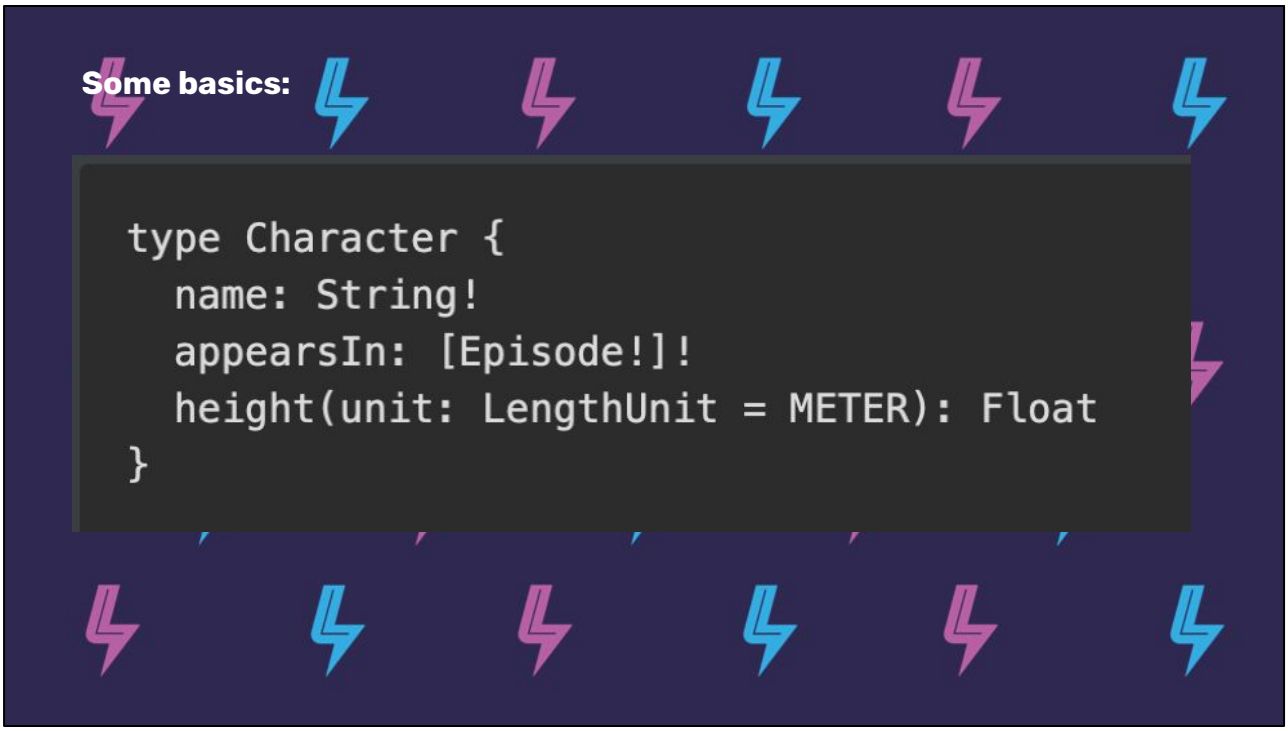


Like REST, GraphQl sits on top of http and defines an “API design architecture”. One important part is "gives clients the power to ask for exactly what they need”. This is in direct contrast to REST. Instead of resources, everything is a graph of relationships, aka why the graph. The other important part is the Runtime - I would argue is much more structured than REST and it has to be because GET, POST, UPDATE, DELETE are not enough.



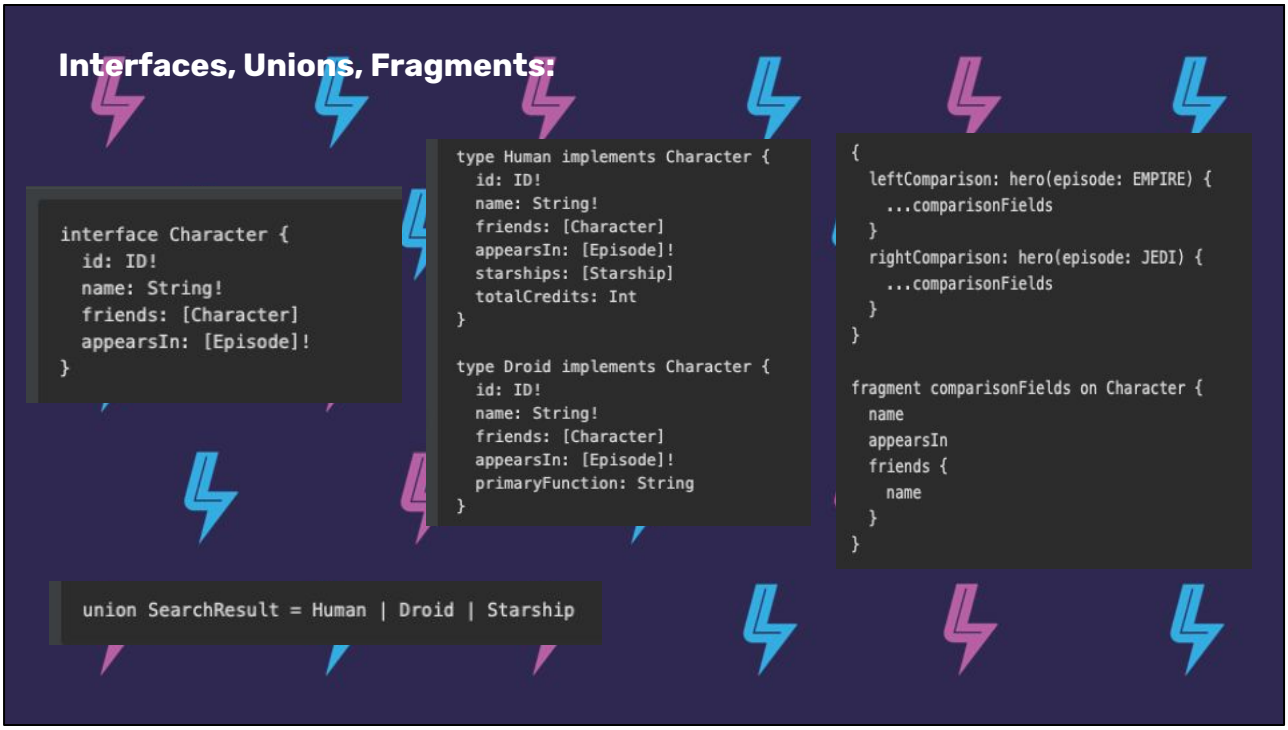
Type systems all start with the special root, service specific set of possible relations to query. Defines why returned data can look like and allows validation of queries
SDL, schema definition language, programming language agnostic way to talk about type systems
Execution semantics - Graphql documents, differences between query and mutation, response map format (data, errors, extensions)

Some basics:

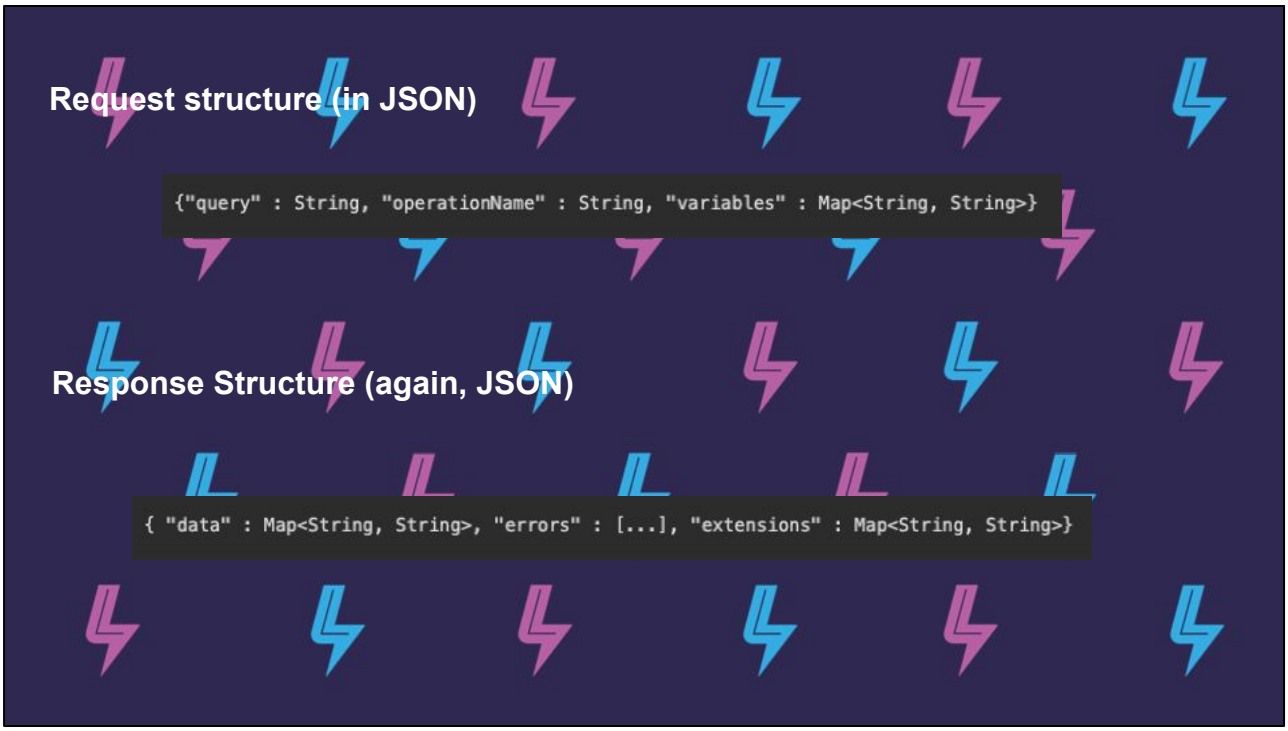


Cover the basics here as much as possible:
Everything is just a graphql object type - meaning it is a type with fields
Everything starts with the special root object and ends in scalars
Scalar types: String, Int, Float, Boolean, ID - same as a string but signifies may not be human readable
Enums are special type of scalar - language dependent in implementation.
Modifiers - Lists and Nullity (return type even has nullity)
2 special types -> query/mutation. Query is entry point and at least one required, mutation optional
Fields can have parameters
Parameters can have defaults

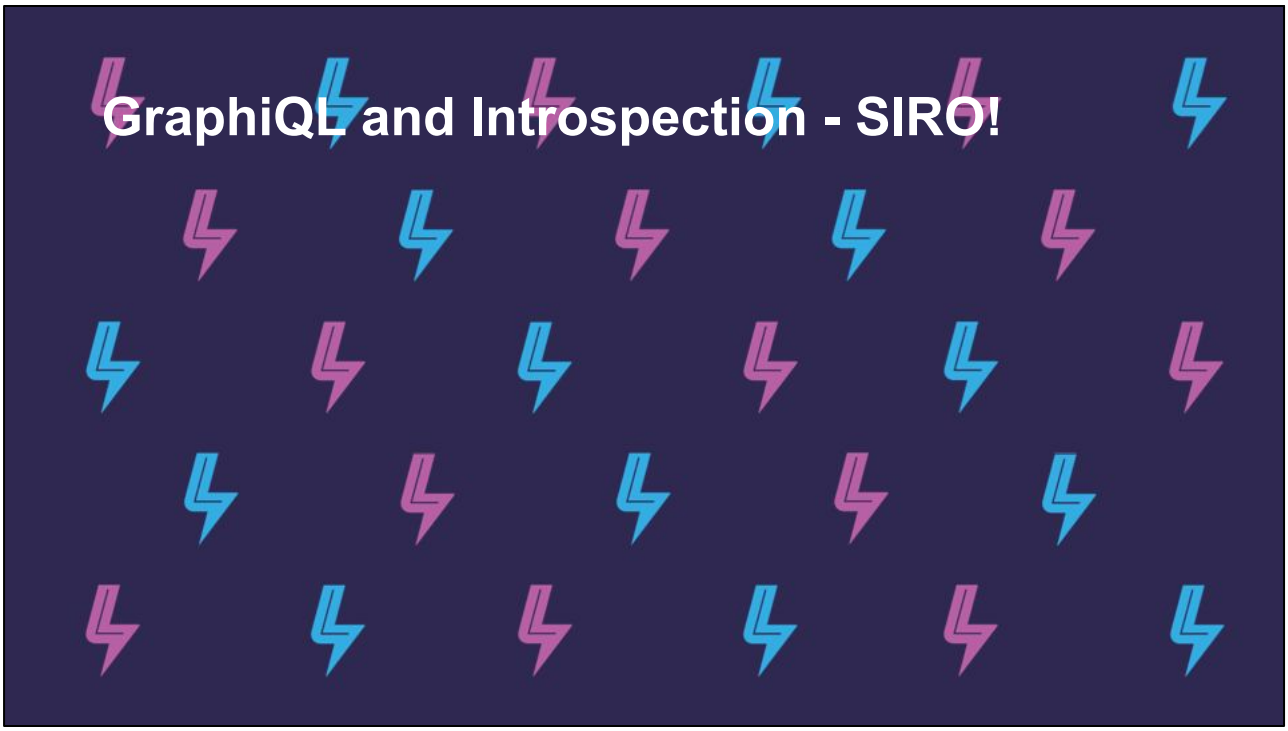
Interfaces, Unions, Fragments:



Interfaces are pretty self explanatory, unions are groupings that have nothing in common, fragments can save space or be inline - they are conditional
We haven't used either and from what I've read interfaces are the only ones people find really useful.



Interesting points - this is designed with lightweight browsers in mind. Query isn't really a query, it is a static set of operations, op name lets you choose which one - avoiding string parsing etc on the client
Query, Mutate are by convention similar to get, post in REST. also have subcriptions
Subscriptions are not durable like kafka etc might be, but more for things like news feeds. Java implementations are socket based.



Show siro’s graphiql - maybe demonstrate variables etc...

Execution

- Parse > Validate > Execute
- Everything is a type, which is some hierarchy from the root
  - So really, everything is just a field of the enclosing type
- Each field is associated with a function to obtain the type - a *resolver*
- Once a resolver completes, arguments and the results (usually called source) supplied to next level in the hierarchy

Validation can occur at any point with the type system - ie, after introspection a client can validate their own queries before sending
We are not getting into details of the “Runtime”. Resolving happens down until leaf nodes or scalars.
Resolvers can be anything - arbitrarily complex, many though are as simple as a property - these are usually supplied by the implementation by default - as we will see with java-graphql.
This is in large contrast to REST, you could view a rest endpoint as a single resolver.
Can anyone see how this is beneficial? How might it suck?
You must have a resolver per field because you don’t know what will be requested

Resolving sounds simple right? Any obvious problems?

A naive resolver example:

```
query {
  authors {
    name
    address {
      country
    }
  }
}
```

1st trip to the db gets the authors

2nd trip gets the address for each author
= n + 1 authors

For this example assume the author backing object has just a name/id field and the address resolver must make another db call.

Data Loaders to the rescue!

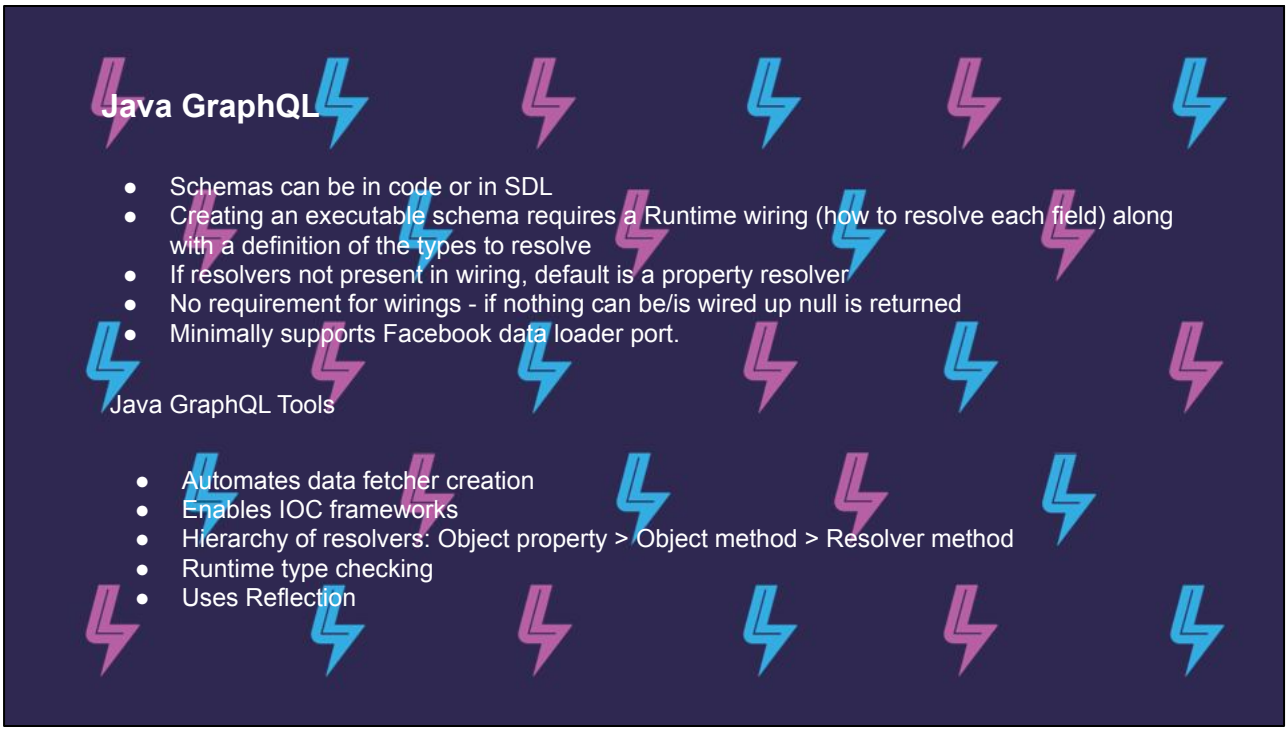
- Developed by Facebook
- Asynchronous - revolves around futures/promises
- Utilizes a cache
- Supports batching
- Created specifically for javascript
- Dataloader<T, S>.load(T t)

Important to note that you want the batching/caching to be request scoped
Can anyone guess why it starting as a javascript library might impact us?

Current Java Frameworks

- Graphql-Java: Core framework. Port of a js reference implementation.
- Graphql-java-tools: A schema-first tool for graphql-java inspired by graphql-tools for JS
- Graphql-java-servlet: Servlet endpoint for GraphQL Java
- Graphql-spring-boot: GraphQL and GraphiQL Spring Framework Boot Starters
- Java-dataloader: A Java 8 port of Facebook DataLoader

These are the ones we are currently using and have experience with
Notice everything is a port….

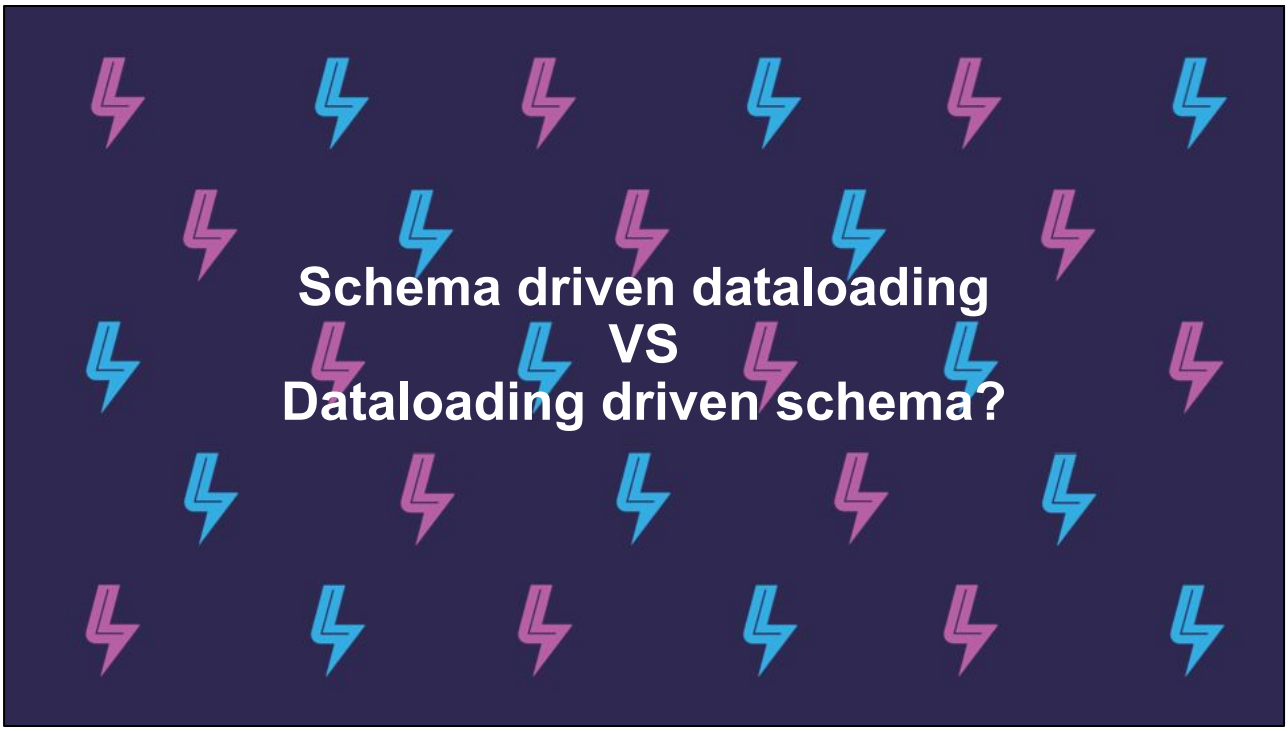


After reviewing this step into code examples leading up to the data loader dispatch problem
Not going to spend a lot of time covering the other frameworks libraries - they are kinda self explanatory
The tools project does classpath scanning to create data fetchers based on schema files (.graphqls)
Relate GOG example to BI and XREF

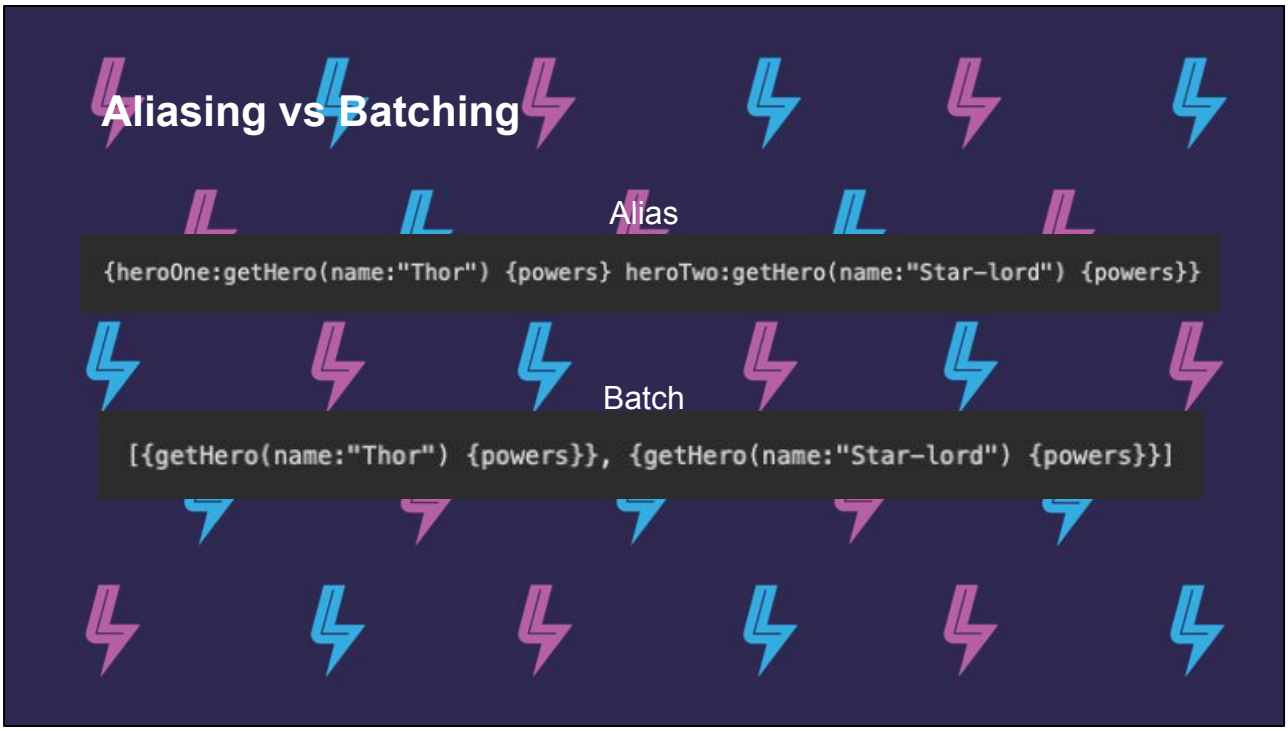
Dataloader Pains

- Javascript version uses the event loop
- Java Graphql's default is through instrumentation
  - Instrumentation build into graphql java aop like
- Dispatching is about competing concerns - being async vs lazy vs sequential relationships.
- Problem even more complex when request batching
- Is this a problem of the framework or a modeling problem?

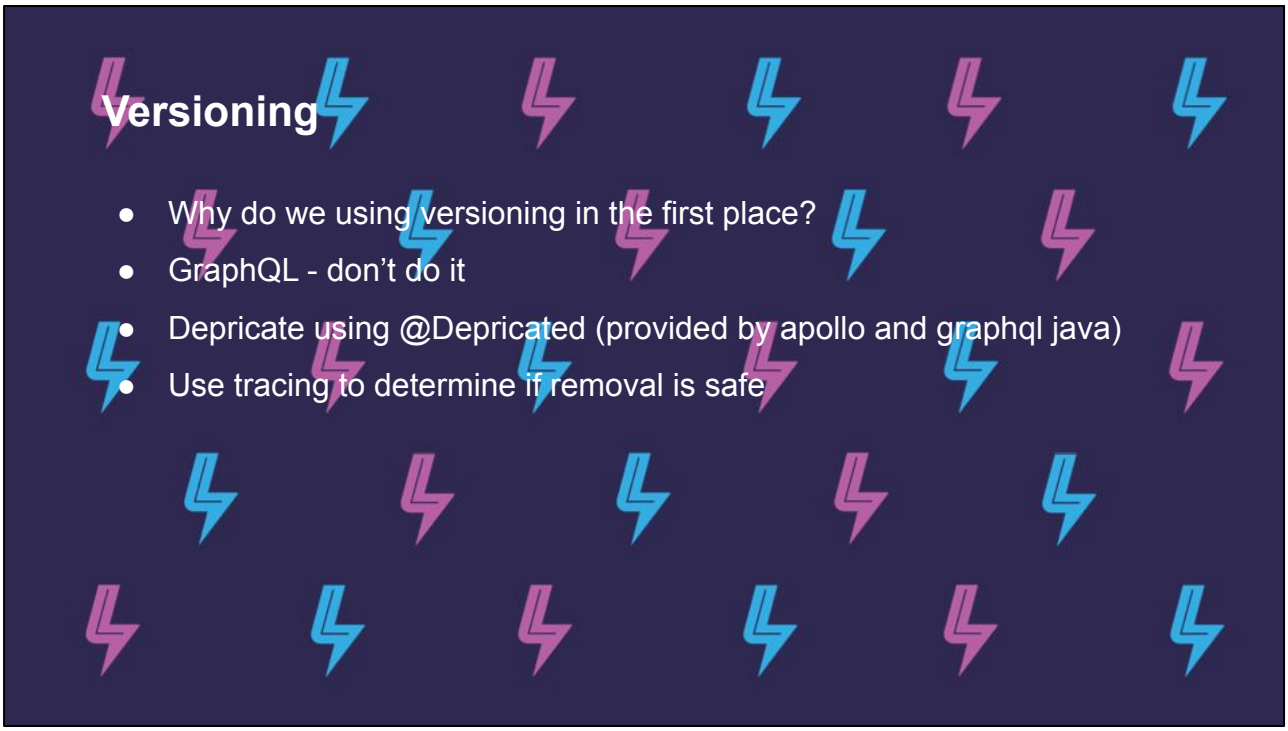
It is important to note - JS does not have any of these problems
There is no current solution in java - this could be a good place for contributions
Further explain request batching - and how this isn't an issue with REST
Is the problem the framework is a port? Is it data modeling? Leading to the first "controversial" topic...



Big point of contention, SIRO did the schema first. What is business logic within the API orchestration layer? Why would writing the db first be an anti pattern and writing the schema first not be? Best argument against so far is how to handle if you change the loading pattern but how often do you change data from one place to another, that changes the loading pattern (hint we never have)? In that case why wouldn't you simply treat the new one as a new field and the old one as deprecated?



If we didn't mention aliasing earlier explain here.
Aliasing is nice, but it destroys caching - can no longer cache document parsing, and can your cdn handle it?
You no longer get metrics the way you think you would
The client now has to maintain/create names
This sounds good right? Well not in Java - highlight how the instrumentation wasn't allowing batching to really happen.
Is the answer to add a new method that accepts a list instead (like REST would require)?

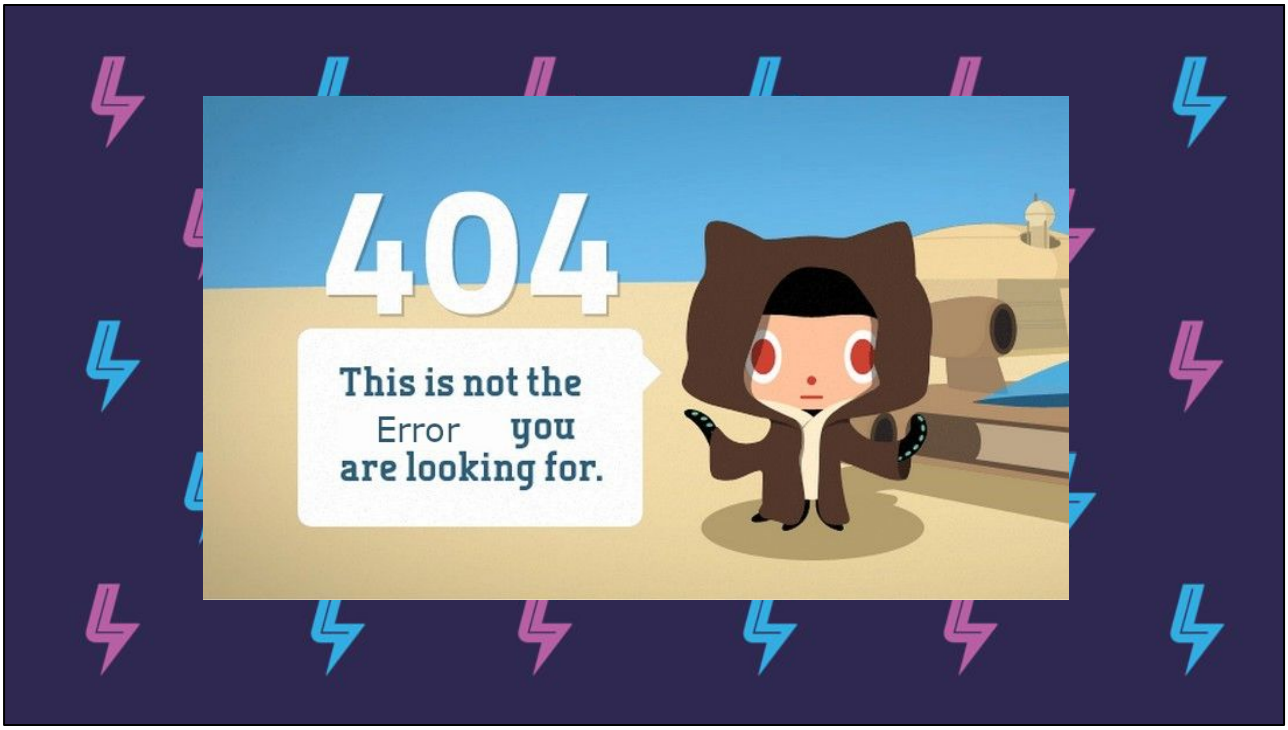


We version apis to gain control over the shape of data
GraphQL is highly opinionated about not versioning, you shouldn’t but you can if you really want
Additions are non breaking.
Let this lead into directives

Directives

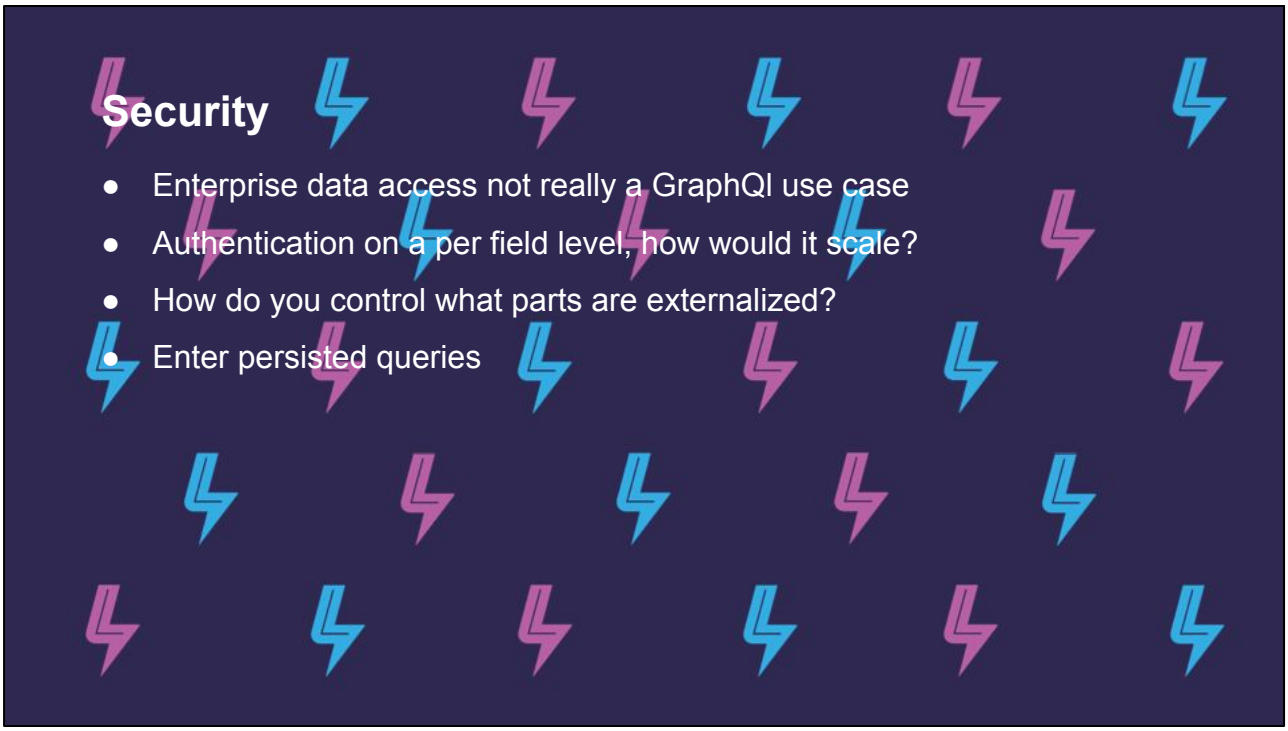
- @include(if: Boolean)
- @skip(if: Boolean)
- Others provided by frameworks (depreciated, cacheControl, deferred etc)
- Mostly useless for Java/Walmart use case

They seem powerful. Explain experience with chainRestriction and cacheControl and how they don’t really work with ccm etc.

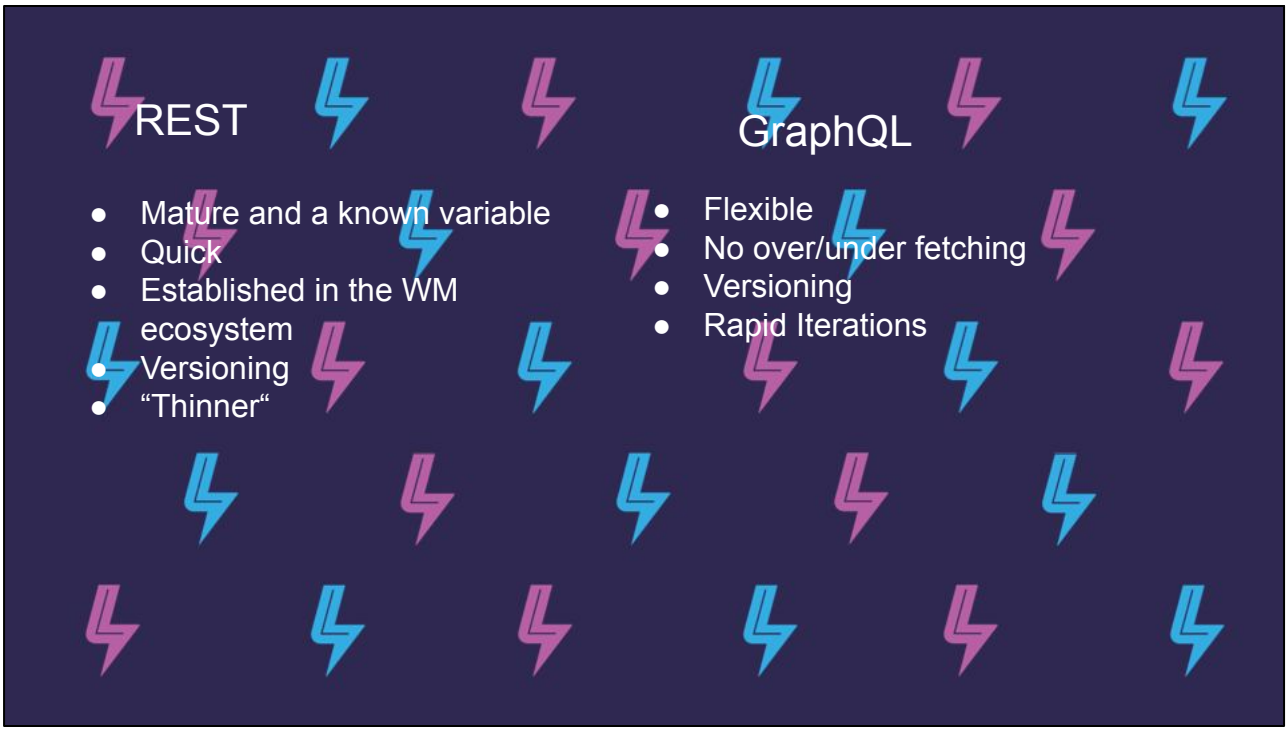


GraphQL errors are a huge growing pain, 200 only really means request got to the server and parsed.
Error handling is on the top of everyones “5 things I learned doing graphql”
Have to keep in mind - graphql supports both errors and data at the same time
If you want the client to use it, think about making it a property or flag allowing general errors to use the built in error mechanisms

This is one of our old errors that we used to send back to clients.
We are simplifying our error responses, but we still do not have a good way to communicate what errors are fatal etc.



Graphql is really designed with browsers in mind, you can authenticate but do things like what fields to externalize belong in the business logic layer?
How about when a client changes their query without notice? How do you handle neighbor effects?
This problem doesn't exist in rest because you would do a different endpoint. We ended up putting a rest like cover over graphql in APQs.
Movement to APQ has really just made SIRO "a set of configurable on demand rest endpoints".



When would I use REST over graphQL? Something one off, something that needs to be done quickly and will not necessarily iterate much.
In the general walmart ecosystem suggest evaluating using apollo and just biting the bullet on JS
Other complains, like REST endpoints that to really implement REST have analogs in GraphQL - this may be mitigated with apollo but Java has a long way to go