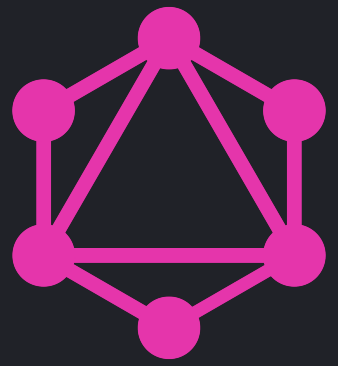


&

`dataloader`

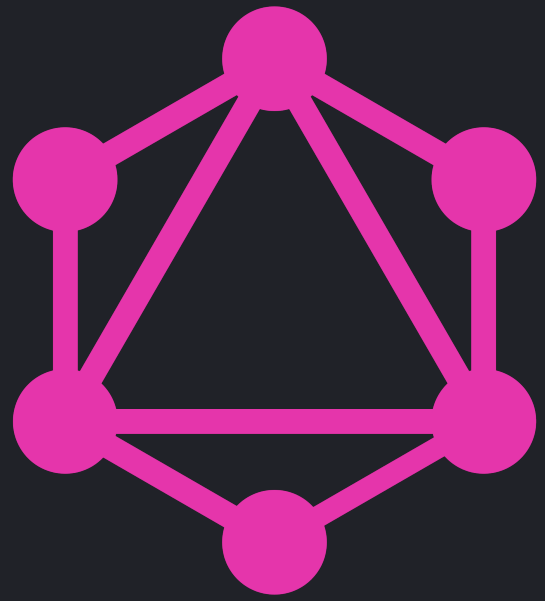
GraphQL & Dataloader

Tyler

Technical Architect @ Somo

Disclaimer

- This is just one way to build with GraphQL
- There are lots of ways
- Take my opinions for what you will



GraphQL

Here are some type definitions

```graphql
import { gql } from 'apollo-server-express'

export const typeDefs = gql`
  type Post {
    title: String!
    text: String!
    author: Author!
    comments: [Comment]!
  }

  type Comment {
    text: String!
    author: Author!
  }

  type Author {
    id: Int!
    name: String!
  }

  type Query {
    posts: [Post]!
  }
`
```

```
export const resolver = {
  Query: {
    posts: (_, args, { postsApi }, info) => postsApi.findAll()
  }
}
```

And here is a resolver

GraphQL Resolver

"A function that resolves a value for a type or field in a schema"

Resolver Rules

- Resolvers are executed breadth-firstly
  - Siblings are executed in parallel
  - A Child is excuted only after it's parent resolves
- If an object is returned, then execution continues to the next child field
- If a scalar is returned, execution completes

GraphQL query execution *always* ends when resolvers return a scalar or null value

`dataloader`

```
function batchLoadingFn (ids) {
  return Promise.all(
    ids.map(id => findById(id))
  )
}

const dataloader = new Dataloader(batchLoadingFn)

dataloader.load('id1').then(...)
dataloader.load('id2').then(...)
dataloader.load('id1').then(...)

// process.nextTick: batchLoadingFn(['id1', 'id2'])
```

- Accepts an Array
- Returns a Promise that resolves to an Array
- Array orders must match each other!

Code Time

May the demo gods be with us

The Default Resolver

```
const resolvers = {
  SomeTypeWithNoResolversDefined: {
    // GraphQL.js adds all of these for you
    id: ({ id }) => id,
    name: ({ name }) => name,
    age: ({ age }) => age
    ...
  }
}
```

Scenario 1: Eagely load the Tags

Scenario 1: Eagely load the Tags

- Pros
  - No Extra Resolver
  - One Query
  - Quick
- Cons
  - Potentially loading unecessary data!
  - One Query (could bog down the datasource)
  - Sets a bad precendent

Scenario 2: Load Tags in field lvl resolver

Scenario 2: Load Tags in field lvl resolver

- Pros
  - Only load Tags, when queried
- Cons
  - Could fetch the same Tag multiple times
  - Still loading unnecessary (though arguably not as bad as #1)

`dataloader` use case #1: Dedupe the Tags

Array of Post Ids => Array of Arrays of Tags per Post

`[1, 2, 3]` => `[ [Tag], [Tag, Tag, Tag], [Tag] ]`

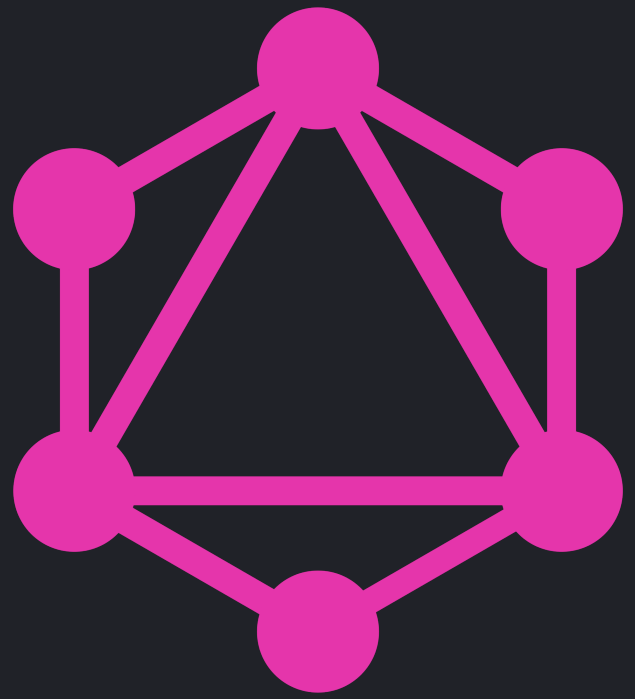
1. Waterfall type resolution
2. Multiple ways to resolve the same type

What we will do

- Each field has a resolver
- Each field fetches it's own data
- If the field:
  - is a scalar, return the scalar
  - is another type, return the **identifier** of that type
    - The parent arg in all child resolvers!

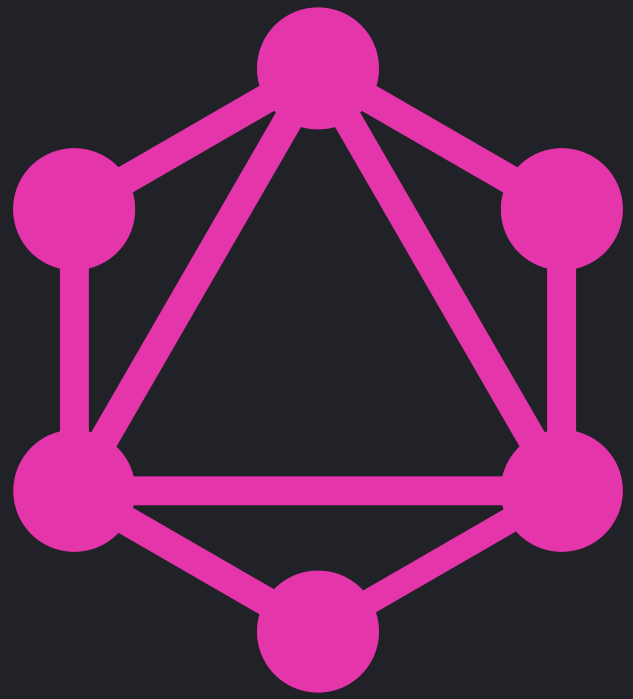
Scenario 3: fetch data at field lvl resolvers

`dataloader` use case #2: Dedupe all the things



Become One With The Graph

(by using `dataloader`)



End