GRAND Stack Workshop

Benoit Simard (@logisima)

neo4j

Welcome !

neo4j

Welcome everyone

Please say hi to each other.

We'll spend the afternoon together.

Benoit Simard

- **Graph addict**
- Data liberator
- Web developer
- Works at Neo4j
- benoit@neo4j.com
- @logisima

Logistic

- **WIFI :** network: `SKEMA_GUEST` | login: `event2018` | password : `Riviera1`
- **Working together today :** Pair on the exercises | Ask relevant questions immediately, raise your hand if you have problems
- **1 break** of 15 min exactly
- **Slides :** https://bit.ly/2k168IY

Exercices

Use hosted online demo services

- Neo4j Sandbox : https://neo4jsandbox.com
- Apollo Launchpad : https://launchpad.graphql.com
- CodeSandbox : https://codesandbox.io

Locally

- Github repo : https://bit.ly/grandstack
- Requirement : Neo4j (3.2), Node & NPM

Agenda

- What is this Stack ?
- Why a new stack?
- Build a simple Movie application
  - A graph model with Neo4j
  - How to query it ?
  - GraphQL Schema definition
  - Implement GraphQl schema with Apollo
  - React'ing

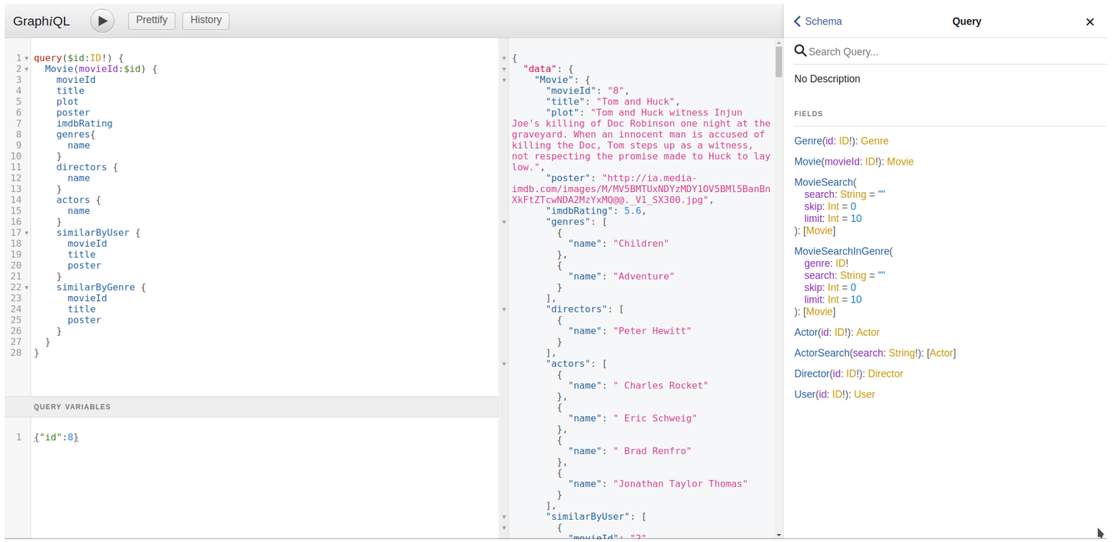
La GRAND stack

http://grandstack.io

A quick overview

neo4j

GraphQL



- A new paradigm for **building APIs**
- **Schema** definition
- Query language for APIs
- Community of tools

React

- JavaScript library for building user interfaces
  - Web, mobile (React Native)
- Component based
  - Reusable
  - Composable

```
import React, { Component } from 'react';
import MovieSearch from './MovieSearch';
import MovieList from './MovieList';


class App extends Component {

  constructor(props) {
    super(props);
    this.state = {
      title: 'River Runs Through It'
    }
  }

  setSearchTerm = (title) => {
    this.setState({ title });
    console.log("setSearchTerm called");
  };

  render() {
    const { title } = this.state;
    return (
      <div>
        <MovieSearch setSearchTerm={this.setSearchTerm} title={title} />
        <MovieList title={title} />
      </div>
    );
  }
}

export default App;
```

Apollo

"A set of tools designed to leverage GraphQL and work together to create a great workflow"

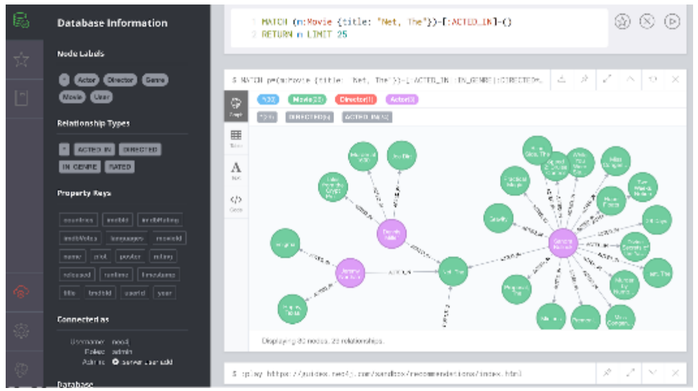
Client-side tooling

- Frontend framework integrations
- Caching
- Code generation

Server-side tooling

- Schema creation
- Mocking
- Schema stitching
- Performance monitoring

Neo4j

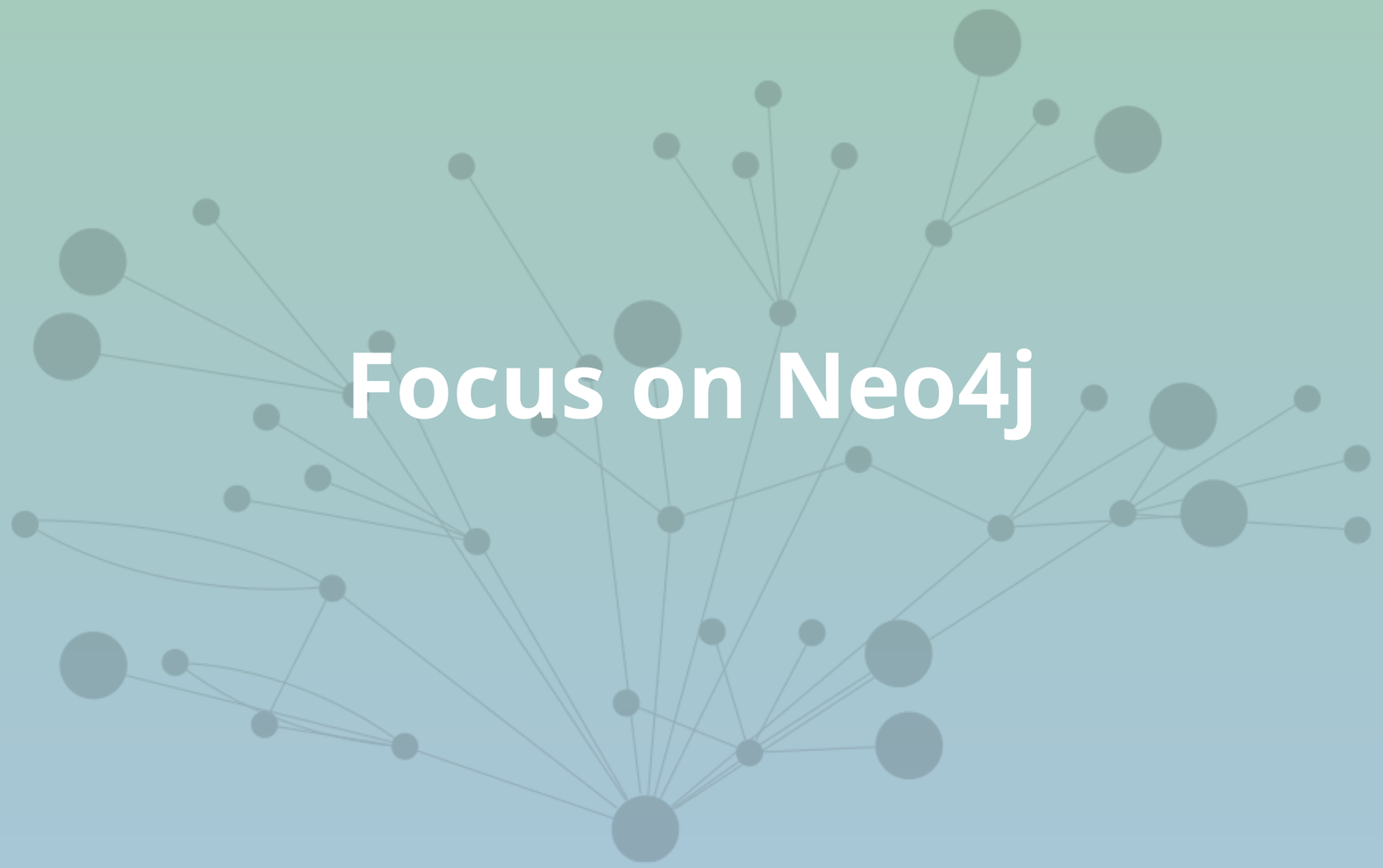


- Is a graph **database** (ACID compliant)
- Is a **graph** database (Native)
- Schema less
- Exists since 2010
- Open-source

A movie application

https://1vn07jo3j3.codesandbox.io/

A movie application

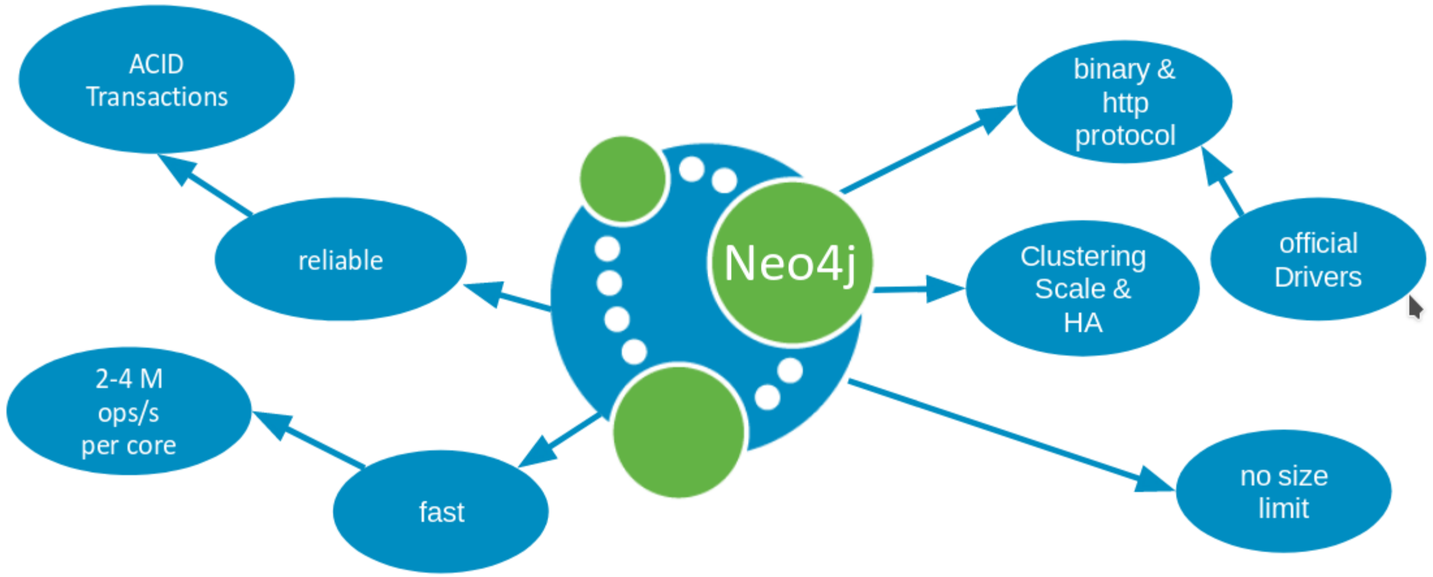


Focus on Neo4j

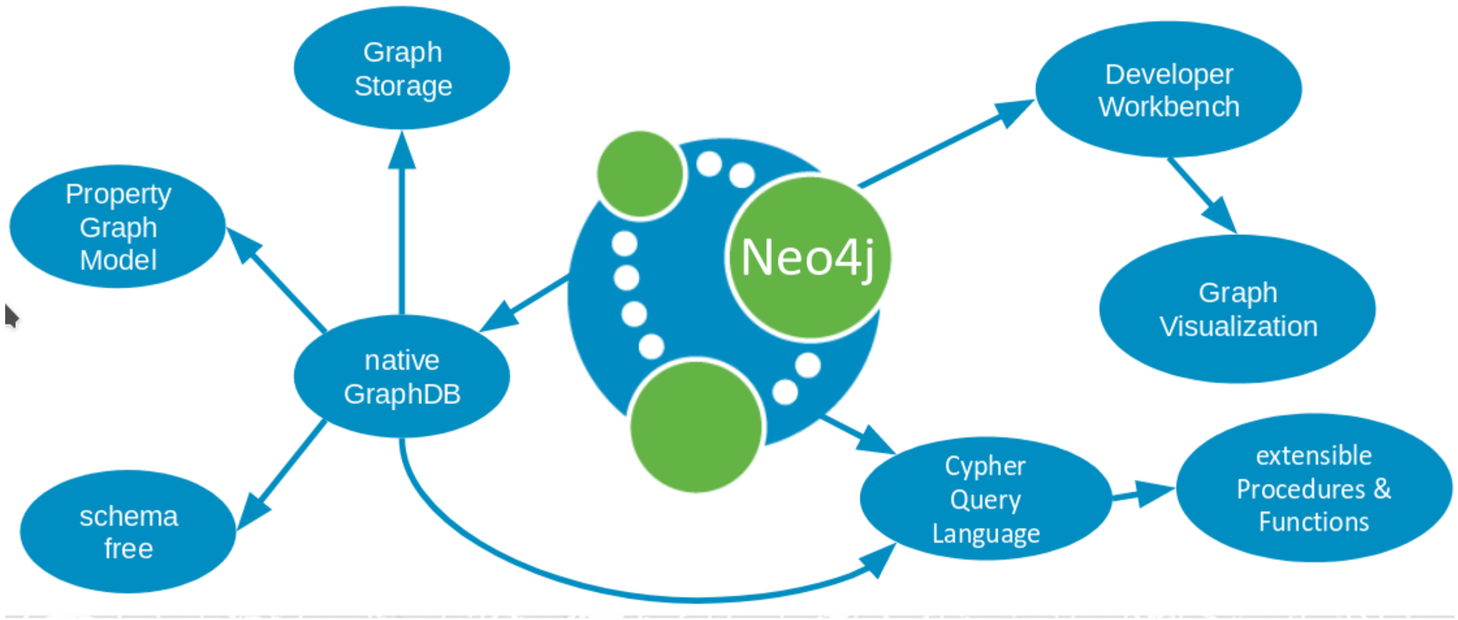
Neo4j

http://www.neo4j.com/developper

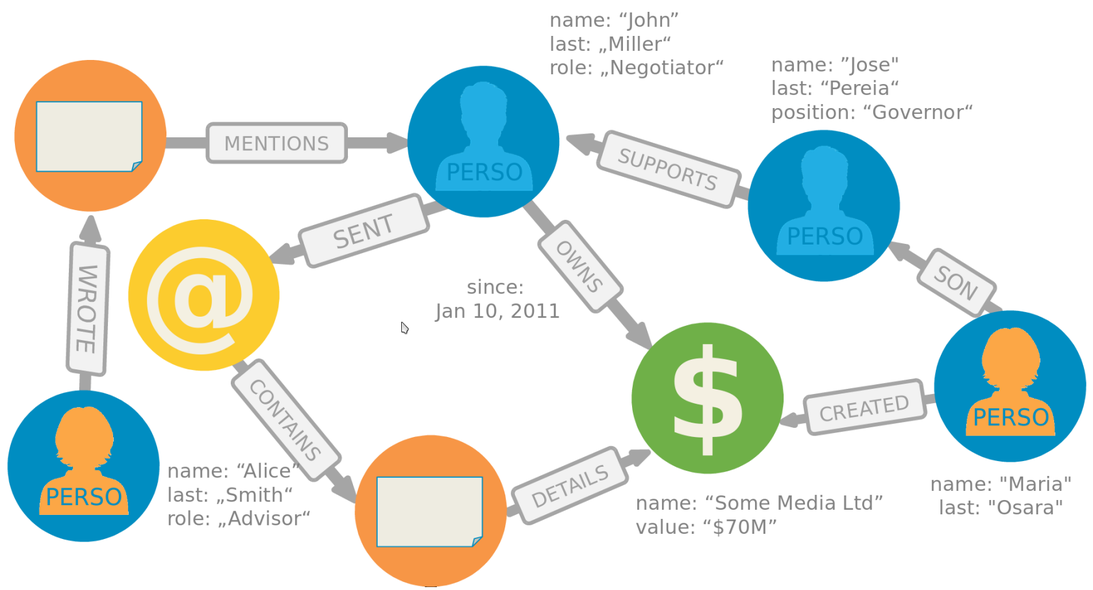
Neo4j is a database



Neo4j is a native graph database



A graph of properties



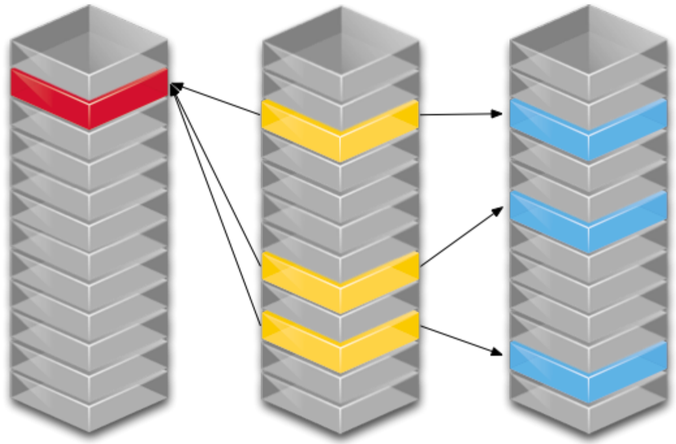
Nodes

- The entity of your model
- Can have labels
- Can have properties

Relationships

- Links two nodes, with a **direction** and a **type**
- Can have properties

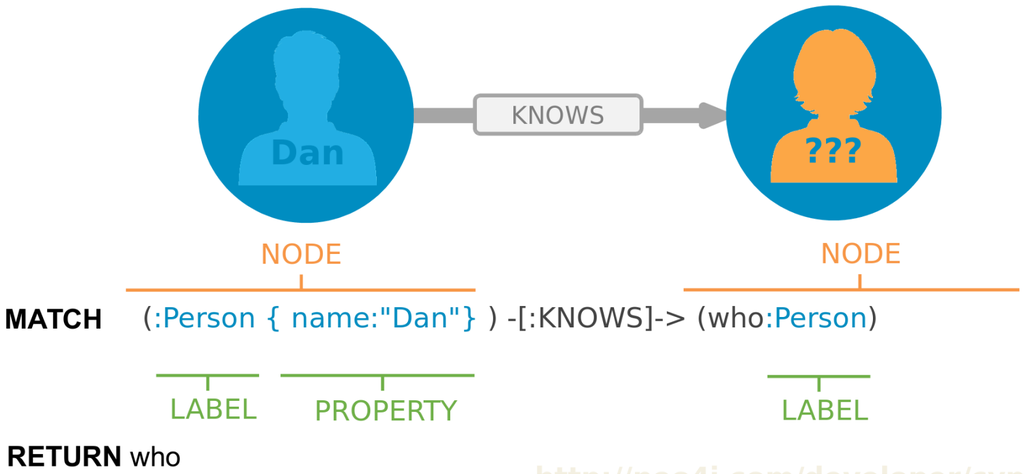
A local approach

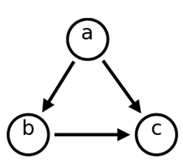


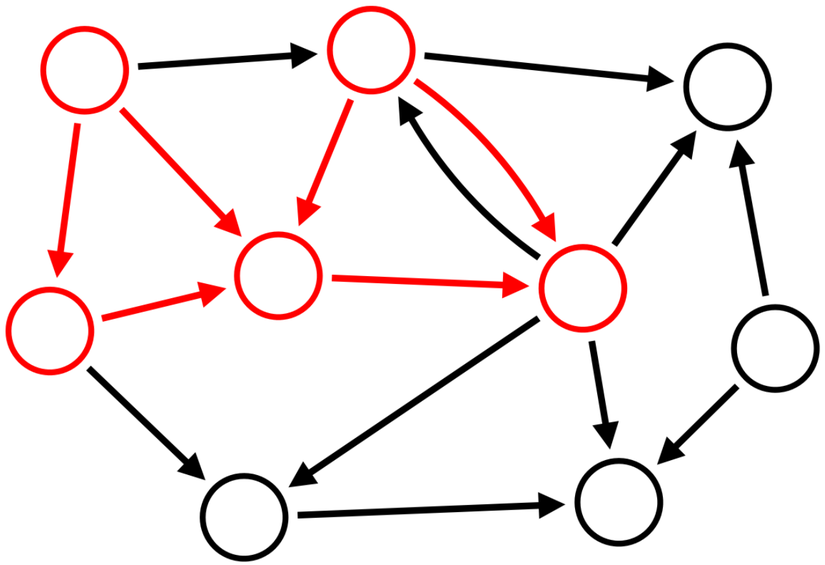
Cypher

OpenCypher.org

Aims to deliver a full and open specification of the industry's most widely adopted graph database query langage.







Ascii Art : Node

- **`()` or `(n)`**
  - Surrounded with parentheses
  - Use an alias to refer to our node later
- **`(n:Label1:Label2)`**
  - Specify a Label, starts with a colon `:`
  - Group nodes by roles or types, think of labels as tags
- **`(n:Label {prop: 'value'})`**
  - Nodes can have properties

Ascii Art : Relationship

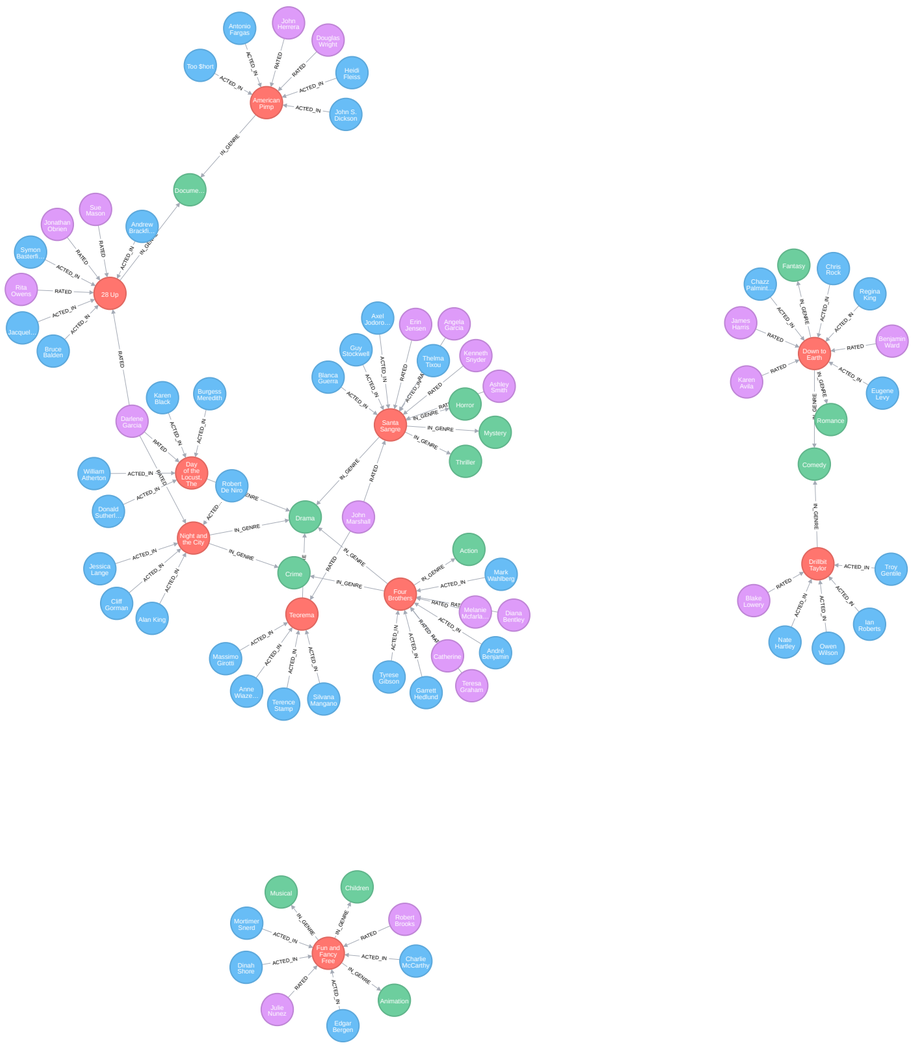
- **`-→` or `-[r:TYPE]→`**
  - Wrapped with hyphens & square brackets
  - Like labels, a relationship type starts with a colon `:`
- **`<` `>` Specify the direction of the relationship `-[:KNOWS {since: 2010}]→`**
  - Relationships can have properties too!

The movie dataset

https://neo4j.com/sandbox-v2/

Recommendations

Generate personalized real-time recommendations using a dataset of movie reviews.



Exercice 1

Goal: Query for Movie by title and find recommended movies

- Sign in to Neo4j Sandbox: neo4jsandbox.com
- Create a “Recommendations” sandbox
- Explore the data in Neo4j Browser
- Write two Cypher queries:
- Search for a Movie by title substring
- For a given movie, find recommended movies
  - *Hint: explore the browser guide for ideas*
  - *Save these queries, we’ll use them in the next exercise*

For those that want to do it locally, you can download the dataset here : https://bit.ly/2wNbhgI

And use `:play` `https://guides.neo4j.com/sandbox/recommendations/index.html` to launch the guide

Exercice 1: answer

- Search for a Movie by title substring

```
MATCH (movie:Movie) WHERE toLower(movie.title) CONTAINS toLower($subString) RETURN movie LIMIT $limit;
```

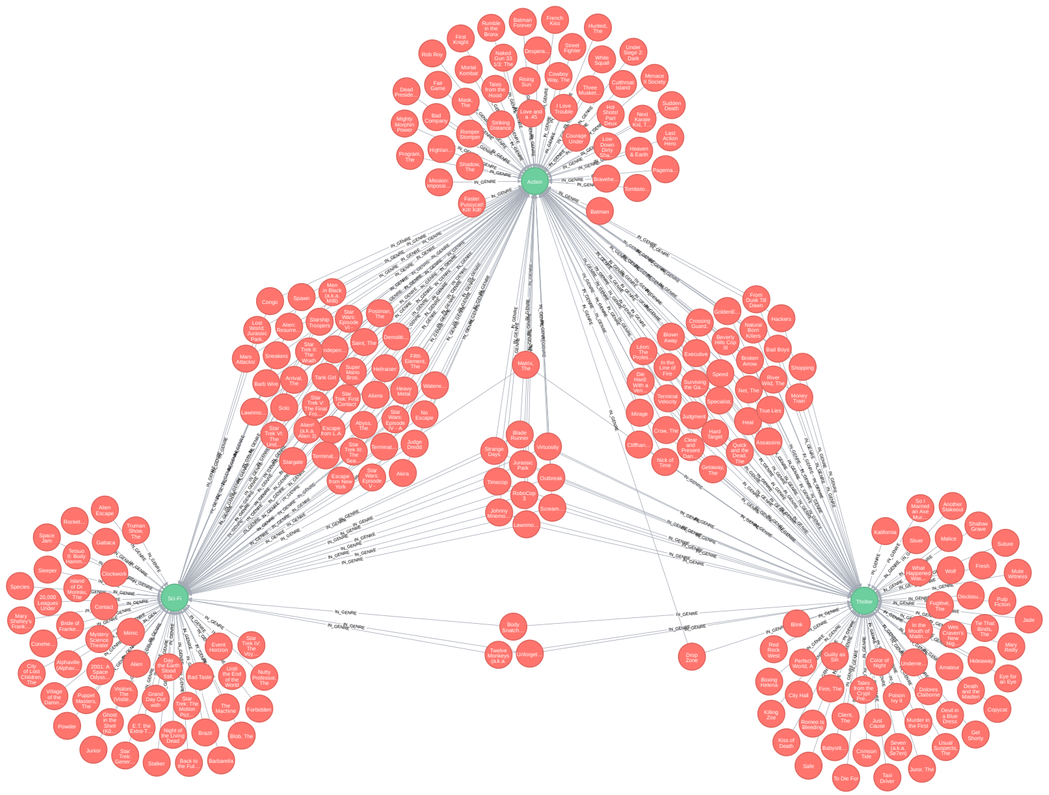
- For a given movie, find recommended movies

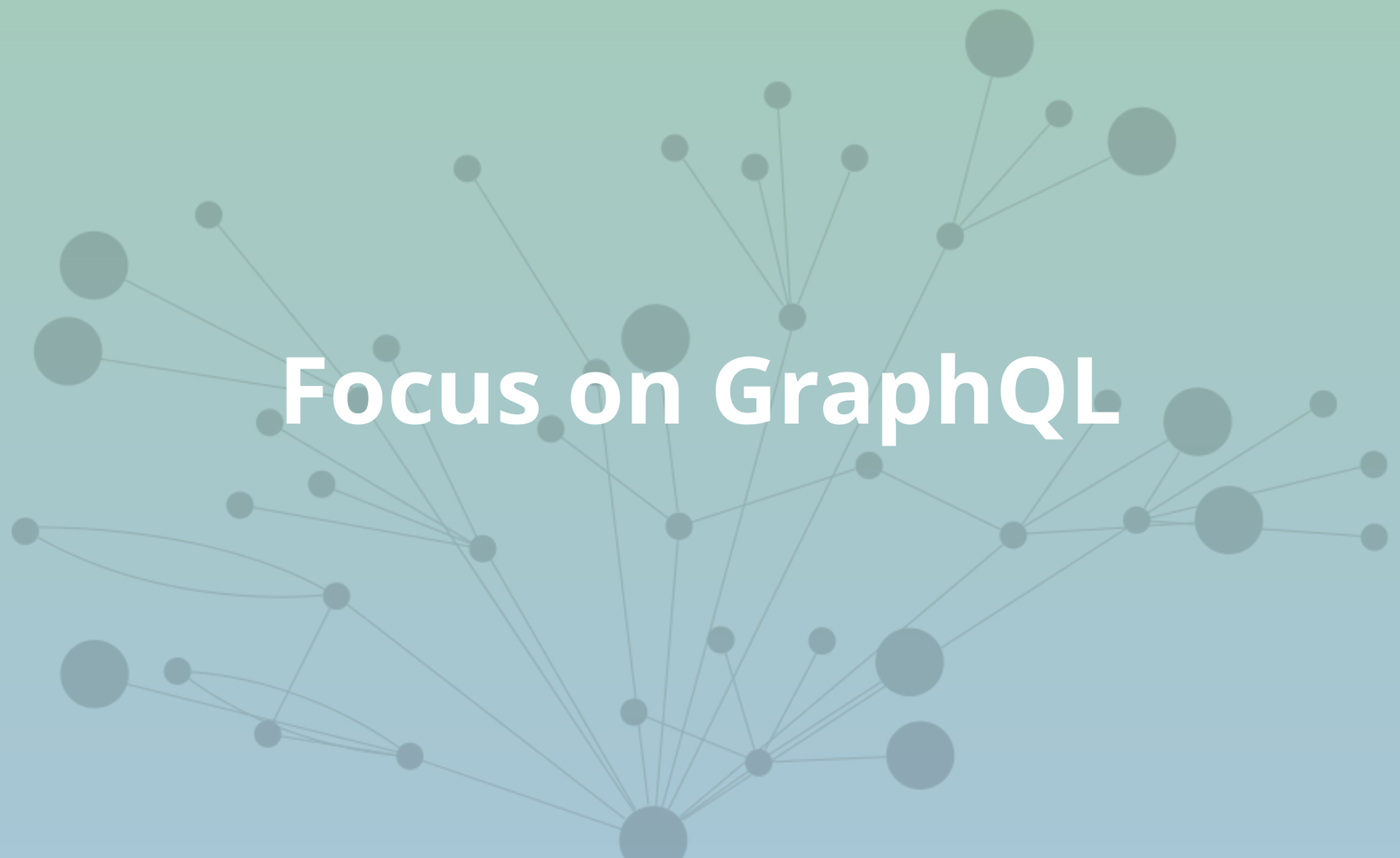
```
MATCH (m:Movie)-[:IN_GENRE]->(:Genre)<-[:IN_GENRE]-(movie:Movie)
WHERE m.movieId= $movieIds
WITH m, movie, COUNT(*) AS genreOverlap

  MATCH (m)<-[:RATED]-(:User)-[:RATED]->(movie)
  WITH movie, genreOverlap, COUNT(*) AS userRatedScore
  RETURN movie
  ORDER BY
    (0.9 * genreOverlap) + (0.1 * userRatedScore)
    DESC
  LIMIT 3
```

https://github.com/grand-stack/grand-stack-movies-workshop/blob/master/neo4j-database/answers.md

Recommendation

```
MATCH
  (m:Movie {title:'Matrix, The'}),
  (m)-[:IN_GENRE]->(:Genre)<-[:IN_GENRE]-(movie:Movie)
WITH m, movie, COUNT(*) AS genreOverlap

  MATCH (m)<-[:RATED]-(:User)-[:RATED]->(movie)
  WITH movie, genreOverlap, COUNT(*) AS userRatedScore
  RETURN movie
  ORDER BY
    (0.9 * genreOverlap) + (0.1 * userRatedScore)
    DESC
  LIMIT 3
```





Focus on GraphQL

neo4j

A large adoption

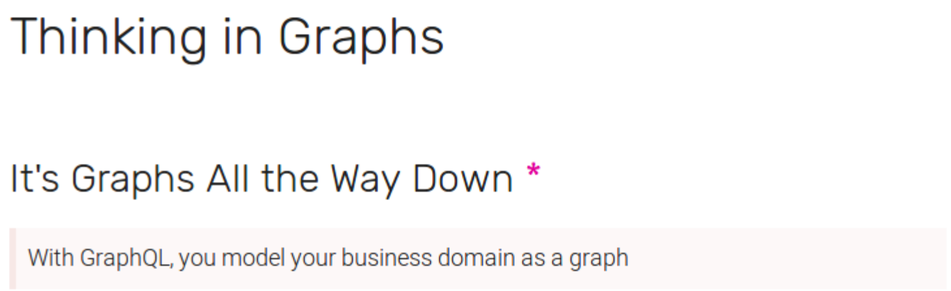
What is GraphQl

“A query language for your API”

- Developed by Facebook iOS team for iOS app
  - Reduce number of round trip requests in face of low latency
- Declarative, state what fields you want Alternative to REST Self documenting (schema and types) Limited support for “queries” Logic is implemented in server

GraphQL

- “A query language for your API, and a server-side runtime for executing queries by using a type system you define for your data”
- “GraphQL isn’t tied to any specific database or storage engine”
- “A GraphQL service is created by defining types and fields on those types, then providing resolver functions for each field on each type”



Your application model is a Graph !

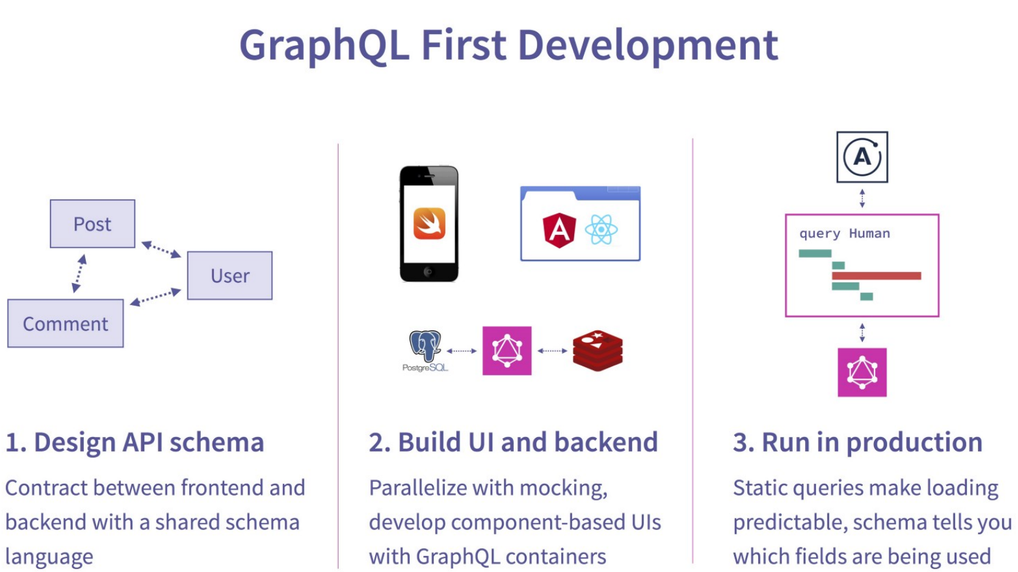
```
query($id:ID = 2571) {
  Movie(movieId:$id) {
    movieId
    title
    plot
    poster
    imdbRating
    genres{
      name
    }
    directors {
      name
    }
    actors {
      name
      actedIn {
        title
      }
    }
  }
}
```

Your application model is a Graph !

```
query($id:ID = 2571) {
  Movie(movieId:$id) {
    movieId
    title
    plot
    poster
    imdbRating
    genres{
      name
    }
    directors {
      name
    }
    actors {
      name
      actedIn {
        title
      }
    }
  }
}
```

```
{
  "Movie": {
    "movieId": "2571",
    "title": "Matrix, The",
    "plot": "A computer hacker learns from mysterious rebels about the true nature of his reality and his role in the war against its controllers.",
    "poster": "http://ia.media-imdb.com/images/M/MV5BMTkxNDYxOTA4M15BMl5BanBnXkFtZTgwNTk0NzQxMTE@._V1_SX300.jpg",
    "imdbRating": 8.7,
    "genres": [ { "name": "Thriller" }, { "name": "Sci-Fi" }, { "name": "Action" } ],
    "directors": [ { "name": " Lana Wachowski" }, { "name": "Andy Wachowski" } ],
    "actors": [
      {
        "name": "Keanu Reeves",
        "actedIn": [ { "title": "John Wick" }, { "title": "47 Ronin" }, ...]
      },
      ...
    ]
  }
}
```

GraphQl First Development



Schema is your friend, and GraphQl schema is the API spec

- **Design API schema**
- **Build UI and backend**
- **Deploy !**

Schema Definition Language

GraphQL Cheat Sheet

- **Type :** The graph model definition
- **Query :** What queries you can do
- **Mutations :** What changes you can do

Exercice 2

Define the GraphQl schema of our web application

- Take a look at the Neo4j model
- Think about `entities` and `queries` we need

Type

Movie

```
type Movie {
  movieId: ID!
  title: String
  released: Int
  plot: String
  poster: String
  imdbRating: Float
  actors: [Actor]
  recommendations(skip: Int = 0, limit: Int =
5): [Movie]
}
```

Actor

```
interface Person {
  name: ID!
}

type Actor implements Person {
  name: ID!
  actedIn(skip: Int = 0, limit: Int = 5):
[Movie]
}
```

Queries / Mutation

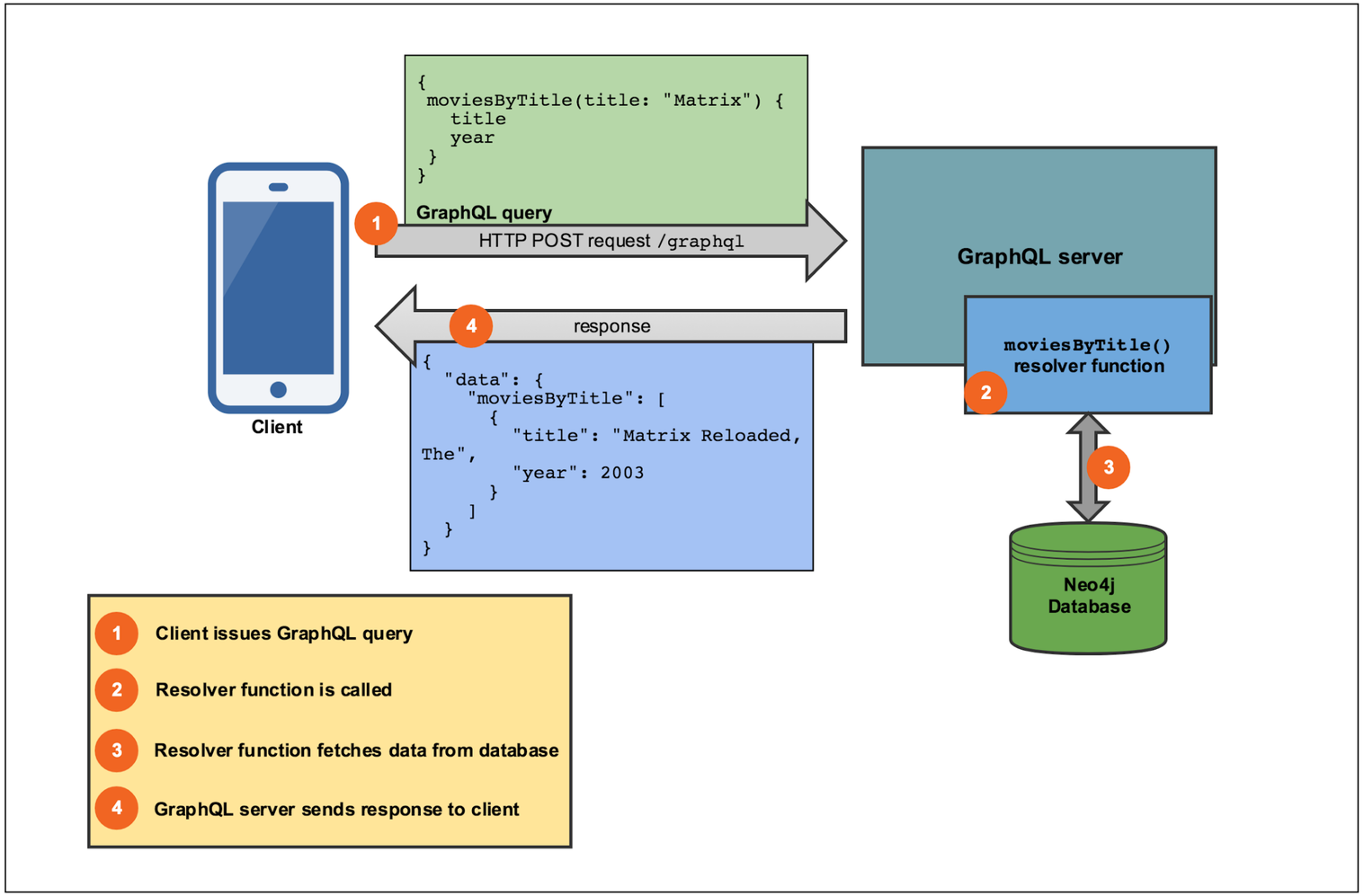
Movie

```
type Query {
  movieById(movieId: ID!): Movie
  movieSearch(search: String = "", skip: Int =
0, limit: Int = 10): [Movie]
}

type Mutation {
  movieMerge(movieId: ID, title: String!,
actors:[String], ...): Movie
}
```

Actor

```
type Query {
  actorById(id: ID!): Actor
  actorSearch(search:String!): [Actor]
}

type Mutation {
  actorMerge(name:String!): Actor
}
```

API Workflow



Apollo

Resolvers

How to fetch the data

```
Movie {

  recommendations: async ( current, _, context ) => {
    let result = await Neo4j.run( context.driver.session(), queries.RECO, current, Neo4j.mappingNodeN );
    return result;
  }

},

Query: {

  movieById( root, params, context, resolveInfo ) {
    let result = await Neo4j.run( context.driver.session(), queries.GET, params, Neo4j.mappingNodeN );
    return result;
  },

  movieSearch: async ( root, params, context, resolveInfo  ) => {
    let session = context.driver.session();
    let query = "MATCH (movie:Movie) WHERE movie.title CONTAINS $search RETURN movie LIMIT $first;"
    return session.run(query, params)
      .then( result => { return result.records.map(record => { return record.get("movie").properties })})
  },

}
```

Apollo Launchpad

https://launchpad.graphql.com/lkvl73r1xq

GraphQl Server

Just an express.js server with some custom endpoints.

```
import express from 'express';
import { graphqlExpress, graphiqlExpress } from 'apollo-server-express';
import bodyParser from 'body-parser';

const PORT = 3000;
const server = express();

/**
 * GraphQL endpoint
 */
server.use( '/graphql', bodyParser.json(), graphqlExpress( async ( request ) => {  return {
  // your graphql schema
  schema: schema,
  // some context for the endpoint (here the Neo4j driver)
  context: Neo4j.context( request.headers, process.env )
}}));

/**
 * Create the graphiql endpoint.
 */
server.use( '/graphiql', graphiqlExpress( { endpointURL: '/graphql', query:'' } ));

/**
 * Run the server.
 */
server.listen( PORT, () => { console.log( `GraphQL Server is now running on http://localhost:${PORT}/graphql` ) } );
```

GraphQl Server - II

```
/**
 * Enable CORS + OPTIONS request for the graphql endpoint.
 */
server.use(
  '/graphql',
  ( req, res, next ) => {
    res.header( 'Access-Control-Allow-Credentials', true );
    res.header( 'Access-Control-Allow-Headers', 'content-type, authorization, content-length, x-requested-with, accept, origin' );
    res.header( 'Access-Control-Allow-Methods', 'POST, GET, OPTIONS' );
    res.header( 'Allow', 'POST, GET, OPTIONS' );
    res.header( 'Access-Control-Allow-Origin', '*' );

    if ( req.method === 'OPTIONS' ) {
      res.sendStatus( 200 );
    } else {
      next();
    }
  }
);
```

Use The Schema, Luke

neo4j-graphql-js & neo4j-graphql plugin

Translate your schema directly in Cypher, thanks to drirectives.

```
{
  Movie(title: "River Runs Through It, A") {
    title
    year
    imdbRating
    actors {
      name
    }
  }
}
```

```
MATCH (movie:Movie {title:"River Runs Through It, A"})
RETURN movie { .title , .year , .imdbRating,
  actors: [(movie)<-[ACTED_IN]-(movie_actors:Actor) | movie_actors { .name }] }
AS movie
SKIP 0
```

openCypher

```
type Movie {
    movieId: ID!
    title: String
    year: Int
    plot: String
    poster: String
    imdbRating: Float
    genres: [String]
    similar(first: Int = 3, offset: Int = 0): [Movie] @cypher(statement: "WITH {this} AS this MATCH (this)--(:Genre)--(o:Movie) RETURN o")
    mostSimilar: Movie @cypher(statement: "WITH {this} AS this RETURN this")
    degree: Int @cypher(statement: "WITH {this} AS this RETURN SIZE((this)--())")
    actors(first: Int = 3, offset: Int = 0): [Actor] @relation(name: "ACTED_IN", direction:"IN")
    avgStars: Float
    filmedIn: State @relation(name: "FILMED_IN", direction: "OUT")
}
```

https://launchpad.graphql.com/xnx0k35lrl

Exercice 3

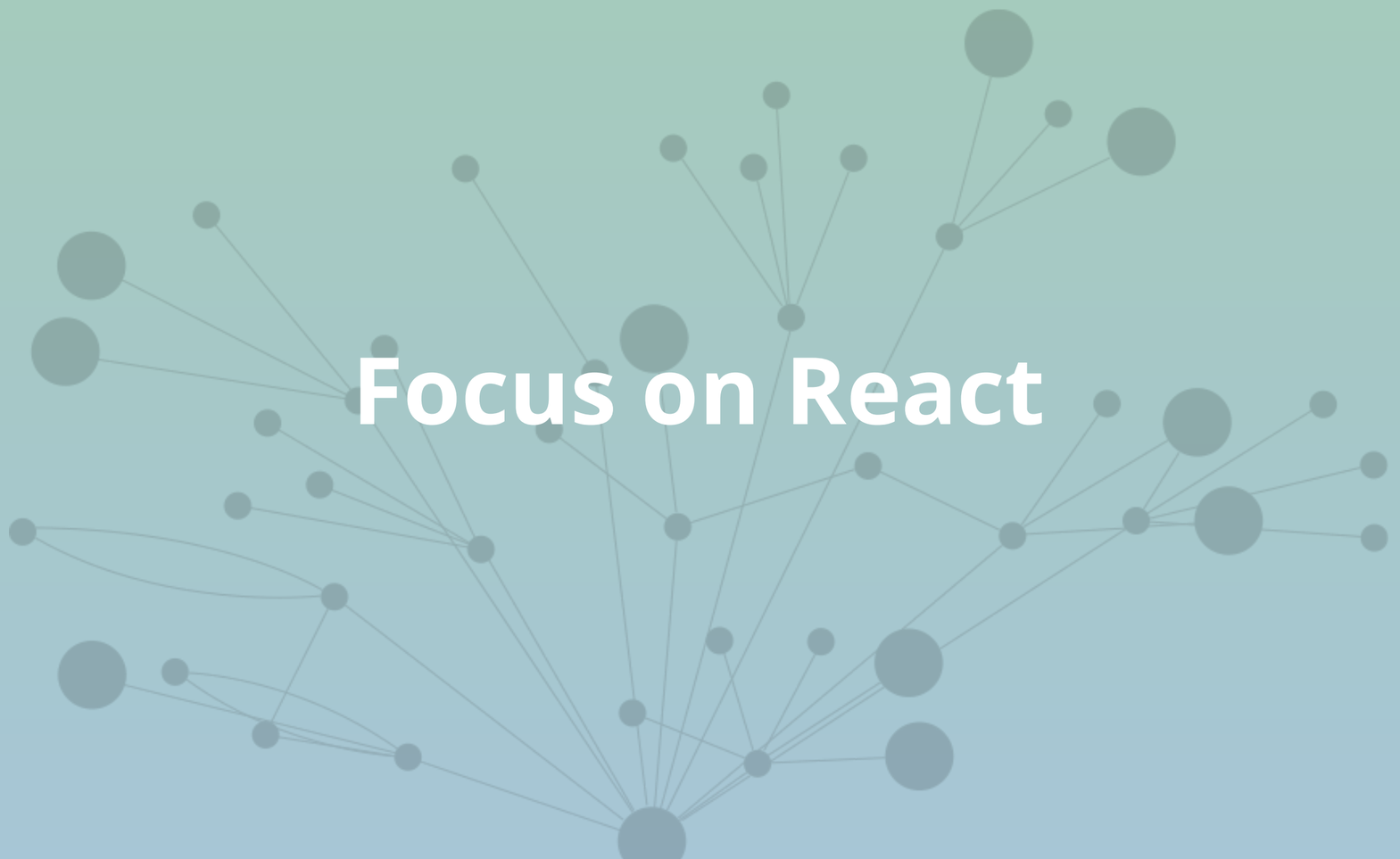
Goal: Build a GraphQL service that connects to Neo4j, allow for querying movies and recommended movies via GraphQL

- Start with skeleton Launchpad: launchpad.graphql.com/3x984k8mv
- Fork it, then add your Neo4j Sandbox credentials to secrets
- Complete the GraphQL service implementation by adding your Cypher queries from Exercise 1
- [optional] Use the JS GraphQL→ Cypher integrations instead of Cypher in resolver functions.

Exercice 3 : answer

https://launchpad.graphql.com/4rpj783r59

Note the GraphQl endpoint, we will use it later



Focus on React

Everything is component

React Apollo : client

https://github.com/apollographql/react-apollo

Create an Apollo client by providing the GraphQl endpoint

```
1 import { ApolloClient } from 'apollo-boost';
2
3 const client = new ApolloClient({
4   uri: 'https://mpjk0plp9.lp.gql.zone/graphql',
5 });
```

React Apollo : wrap application

Wrap your react application with `ApolloProvider`

```
import React from 'react';
import { render } from 'react-dom';
import { ApolloProvider } from 'react-apollo';

const WrappedApp = (
  <ApolloProvider client={client}>
    <App />
  </ApolloProvider>
);

render(WrappedApp, document.getElementById('root'));
```

React Apollo : component

====!

```
import {Component} from "react";
import gql from "graphql-tag";
import { graphql} from "react-apollo";

class MovieList extends Component {

  render() {
    const {data} = this.props;
    if (data.loading) return <div>Loading...</div>;
    if (data.movies.length === 0) return <div>No movies!</div>;

    ...

  }
}

export default graphql(gql`${GRAPHQL_QUERY}`)(MovieList);
```

====!

React Apollo : Use the query component

```
import {Component} from "react";
import gql from "graphql-tag";
import { Query } from "react-apollo";

class MovieList extends Component {

  render() {
    <Query query={GRAPHQL_QUERY}>
      {
        ({ loading, data: { movies } }) => {
          if (loading)  return <div>loading....</div>
          if (movies.length === 0) return <div>No movies!</div>;

          ...
        }
      }
    </Query>
  }
}

export default MovieList;
```

Codesandbox

https://codesandbox.io/s/pk4219zyp0

Exercice 4

Goal: Complete the skeleton React app by adding Apollo Client and fetch data from the GraphQL API you created in Exercise #2

- Start with skeleton CodeSandbox: https://codesandbox.io/s/pk4219zyp0
- Fork it
- Complete the app by adding Apollo Client as a higher order component and connecting to your GraphQL API, using GraphQL to render movie data and recommendations
- See: https://github.com/apollographql/react-apollo
- You'll need to add react-apollo and apollo-client NPM dependencies

Exercice 4 : answer

https://codesandbox.io/s/1vn07jo3j3

Any questions ?

THANKS