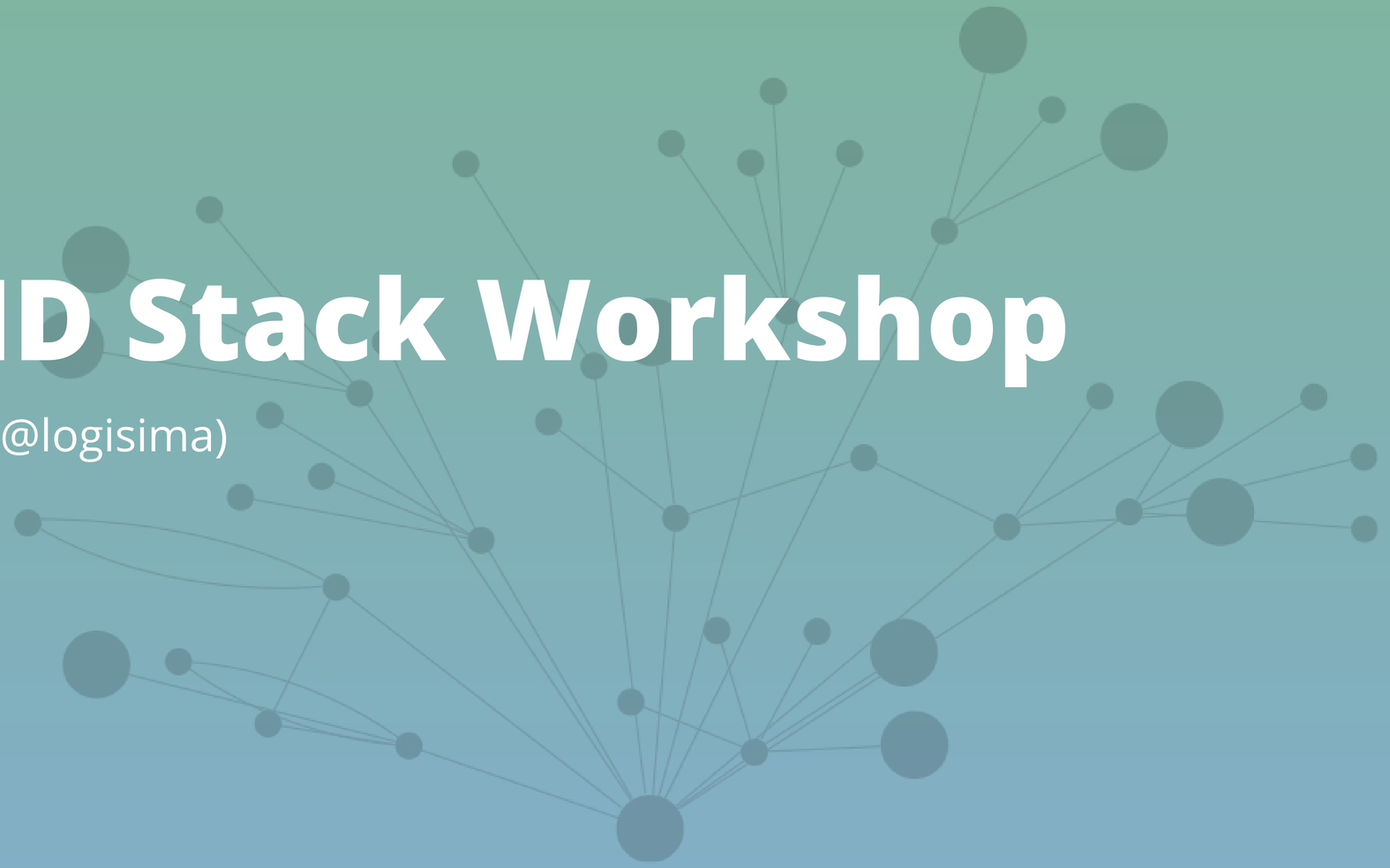
# GRAND Stack Workshop

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**

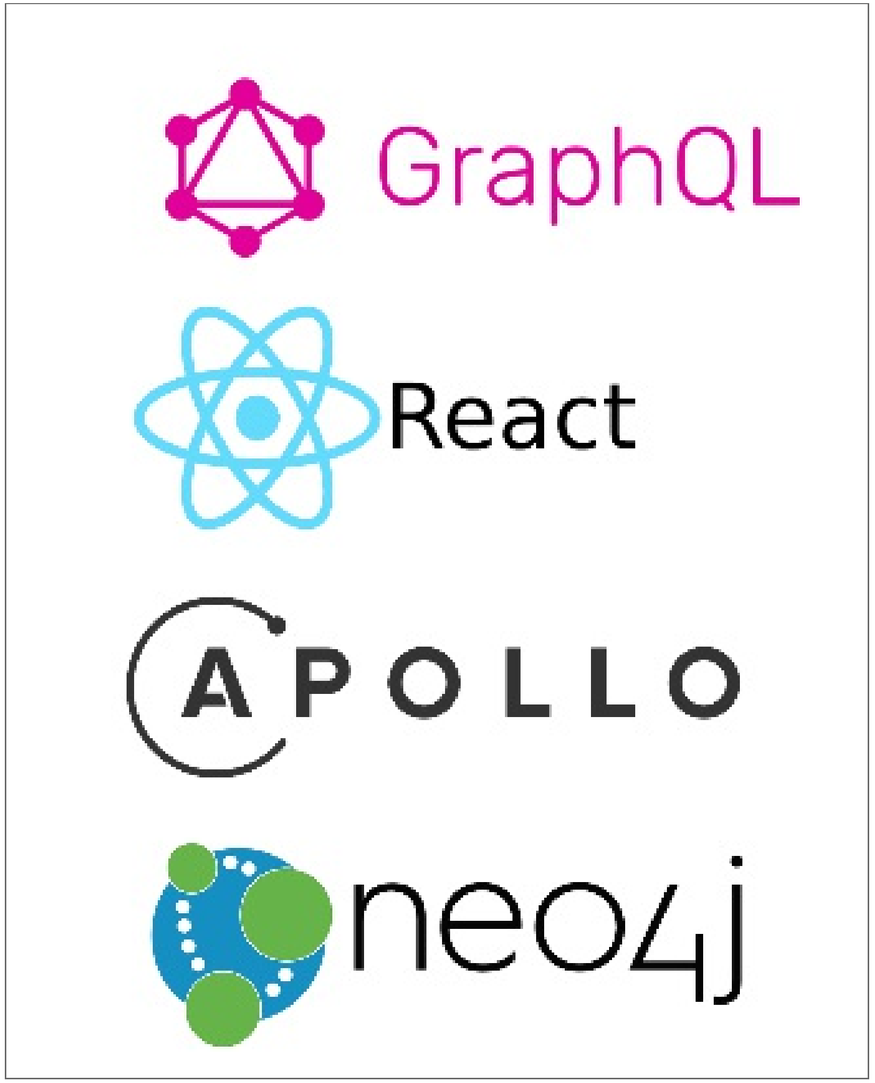
- Github repo : **https://bit.ly/grandstack**
- Requirement : Neo4j (3.2), Node & NPM

# 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

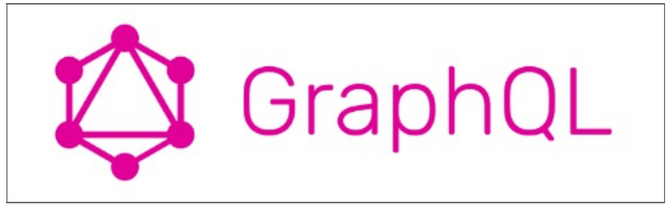
# La GRAND stack

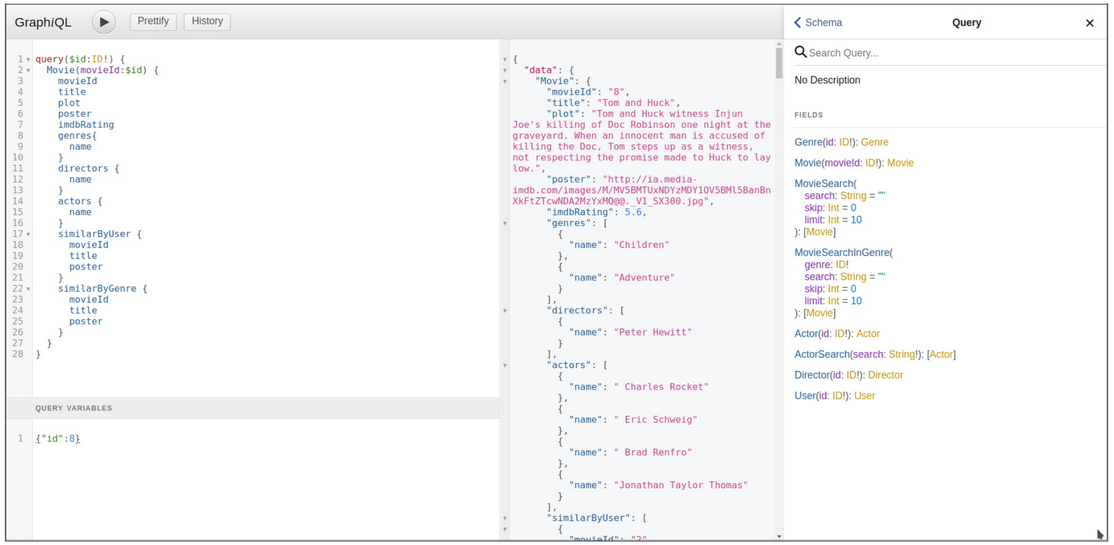
**http://grandstack.io**


GraphQL
React
APOLLO
neo4j

# A quick overview

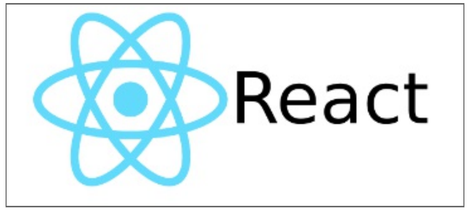
# GraphQL




- 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

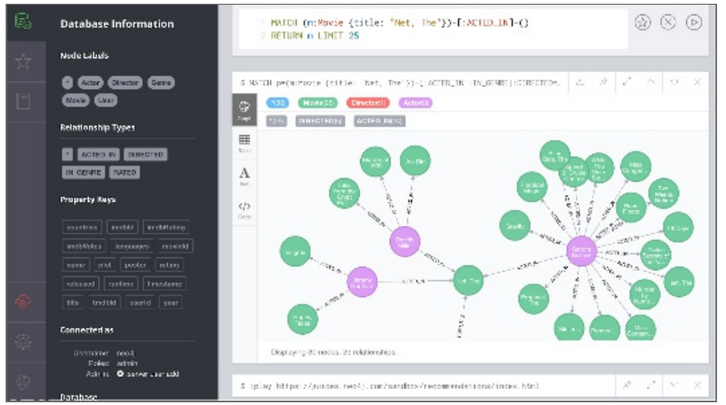
- Frontend framework integrations
- Caching
- Code generation

## 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

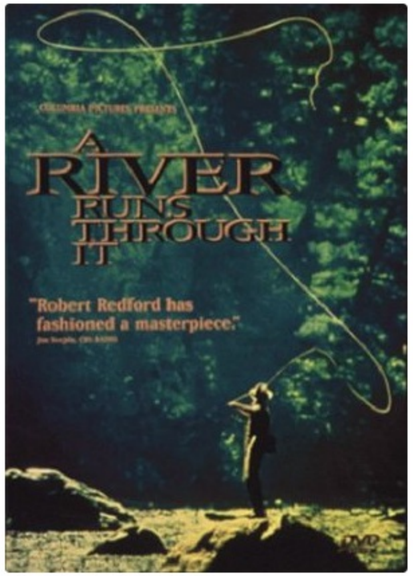


# A movie application

https://1vn07jo3j3.codesandbox.io/

River Runs Through It

Search




## River Runs Through It, A

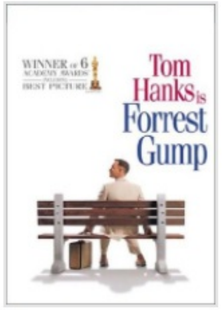
Year: 1992

Rating: 7.3

The story about two sons of a stern minister -- one reserved, one rebellious -- growing up in rural Montana while devoted to fly fishing.

Drama

You might also like:




Forrest Gump


Titanic

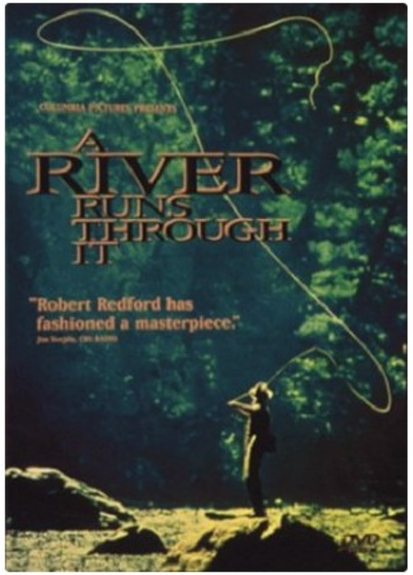

Shawshank Redemption, The

# A movie application

River Runs Through It

Search

Search




## River Runs Through It, A

Movie Details

Year: 1992

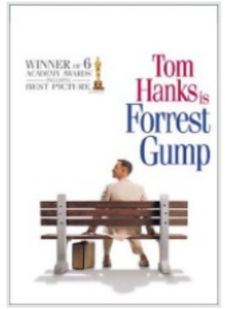
Rating: 7.3

The story about two sons of a stern minister -- one reserved, one rebellious -- growing up in rural Montana while devoted to fly fishing.

Drama

You might also like:

Recommendations

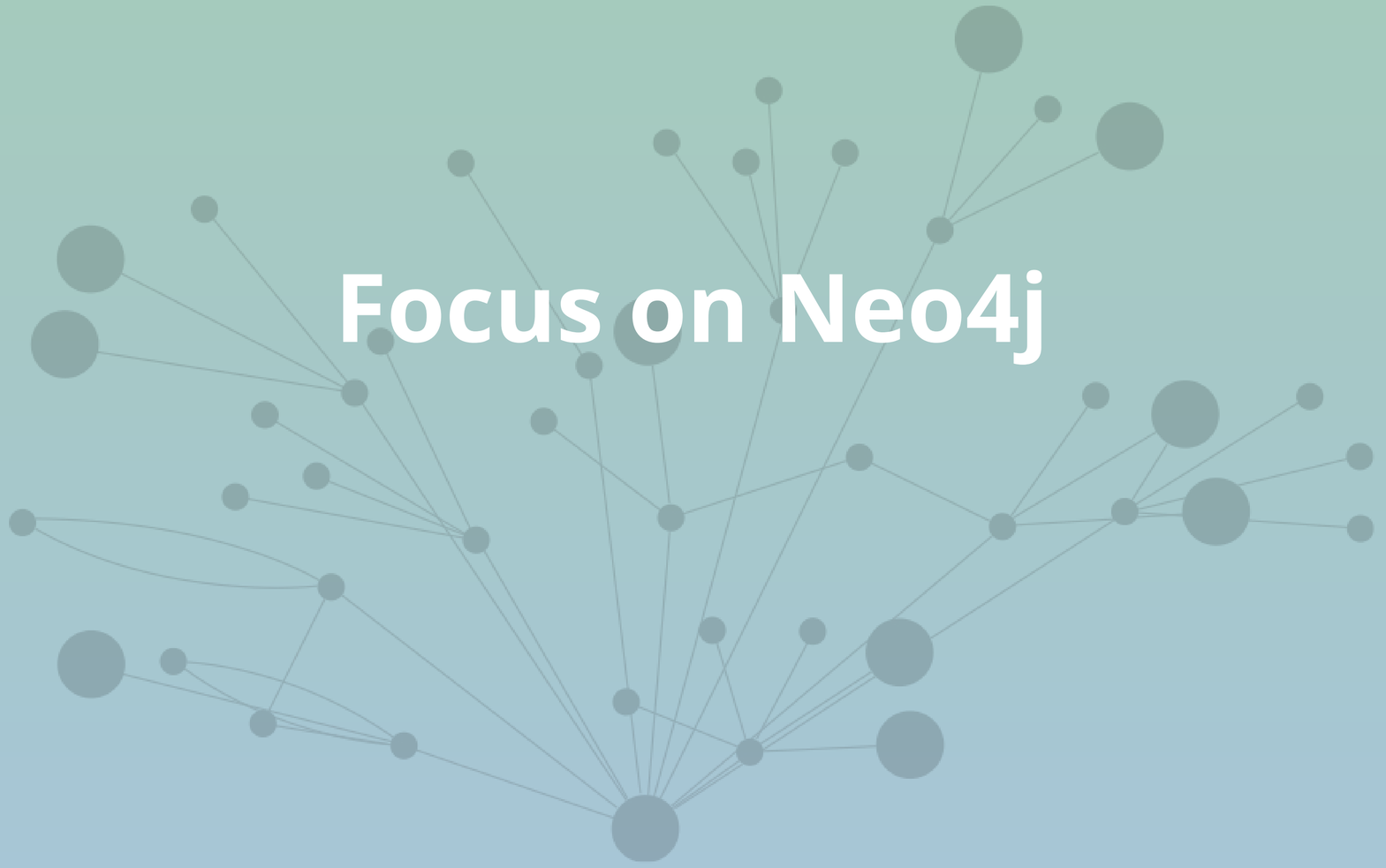



Forrest Gump


Titanic


Shawshank Redemption, The

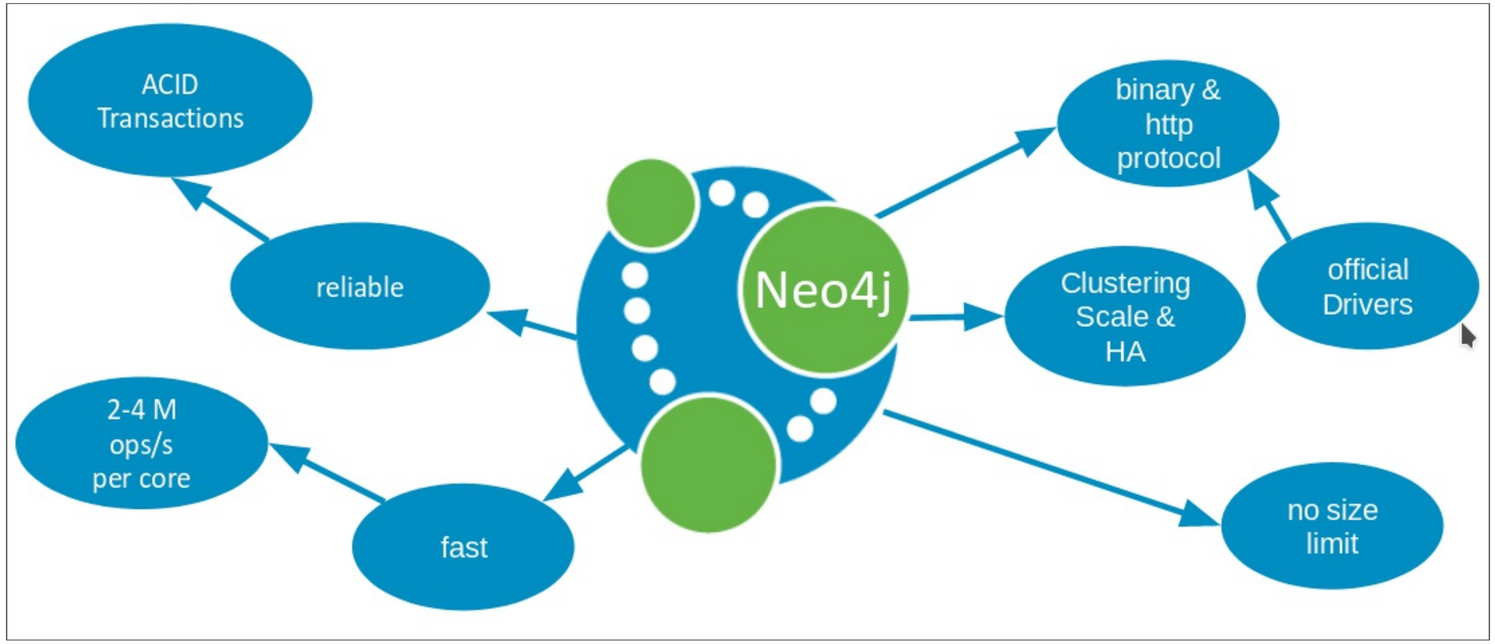
# Focus on Neo4j



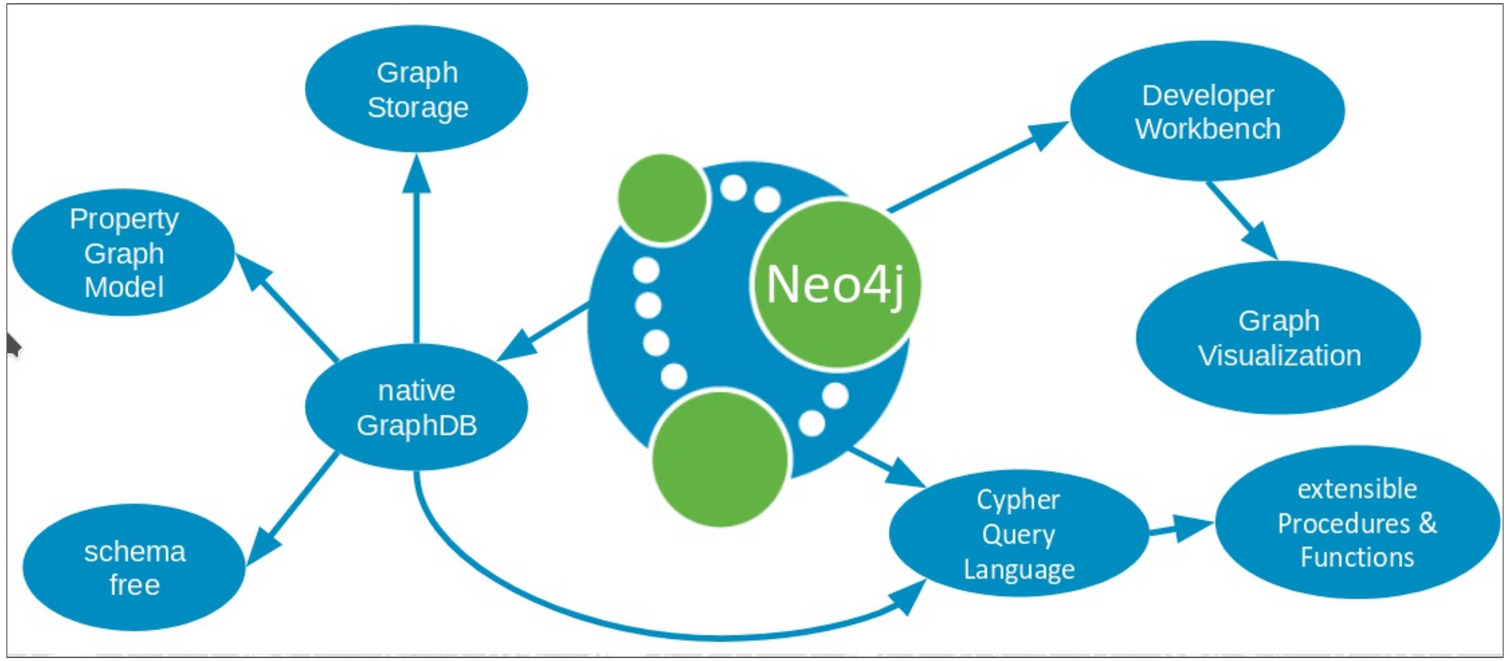
# Neo4j

**http://www.neo4j.com/developper**

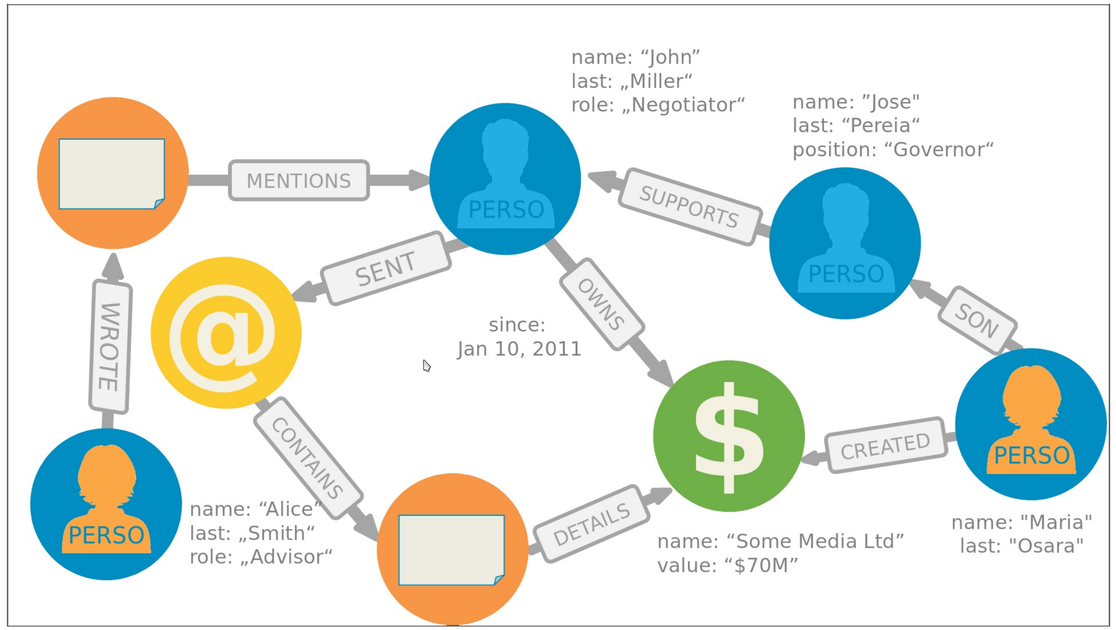
# Neo4j is a database



# Neo4j is a native graph database



# A graph of properties



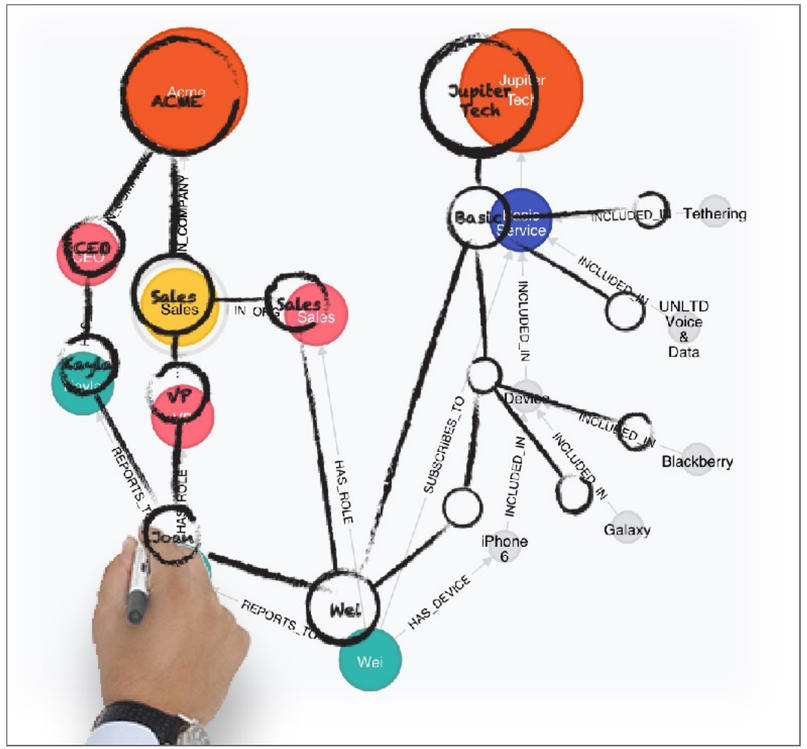

## Nodes

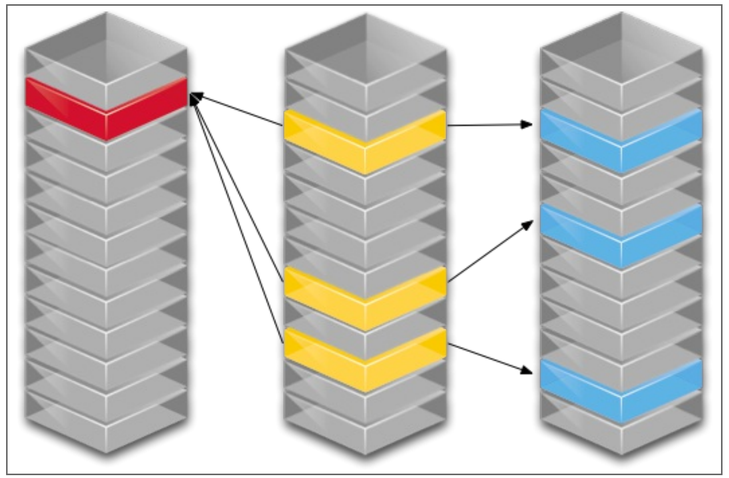
- The entity of your model
- Can have labels
- Can have properties

## Relationships

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

# A local approach

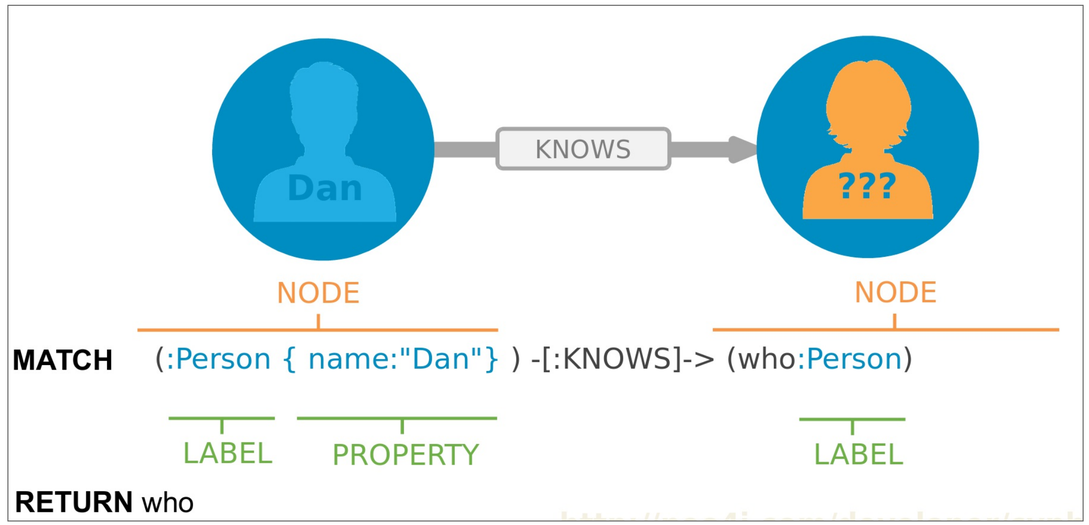


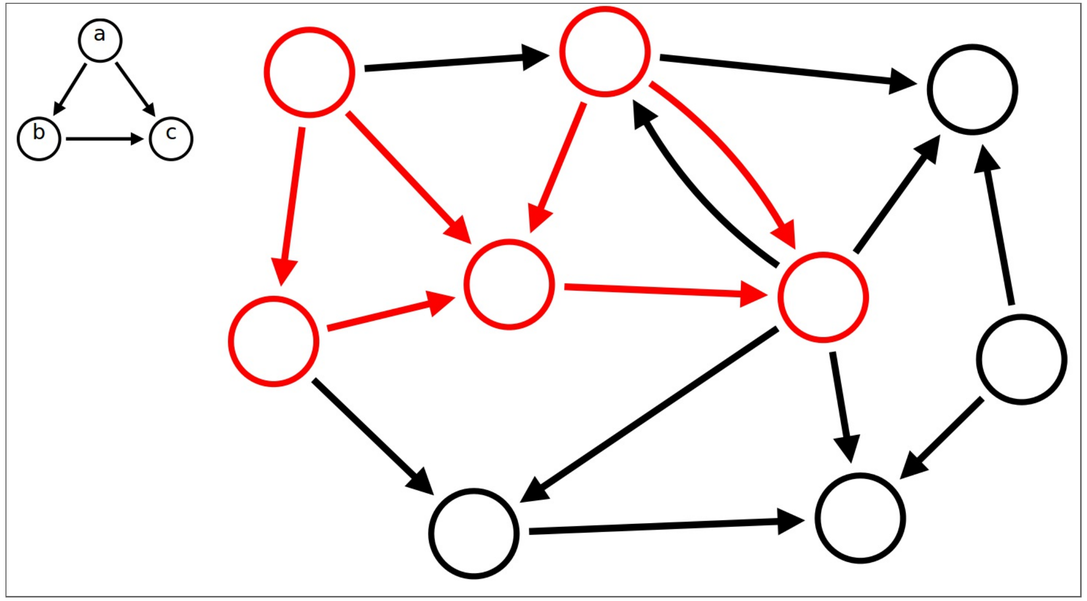



# Cypher

## 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

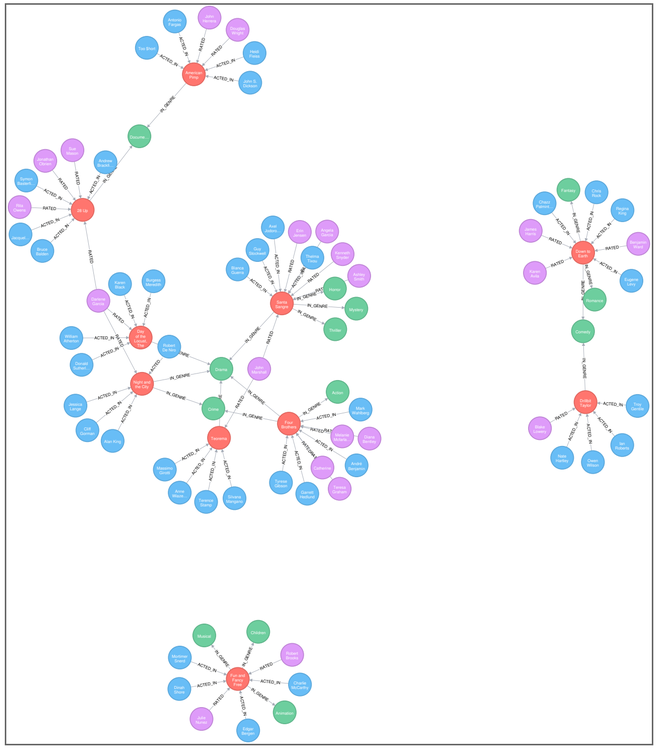
- **`-→` or `-[r:TYPE]→`**
  - Wrapped with hyphens & square brackets
  - Like labels, a relationship type starts with a colon `:`
- **`<` `>` Specify the direction of the relationship `-[:KNOWS {since: 2010}]→`**
  - Relationships can have properties too!

# 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

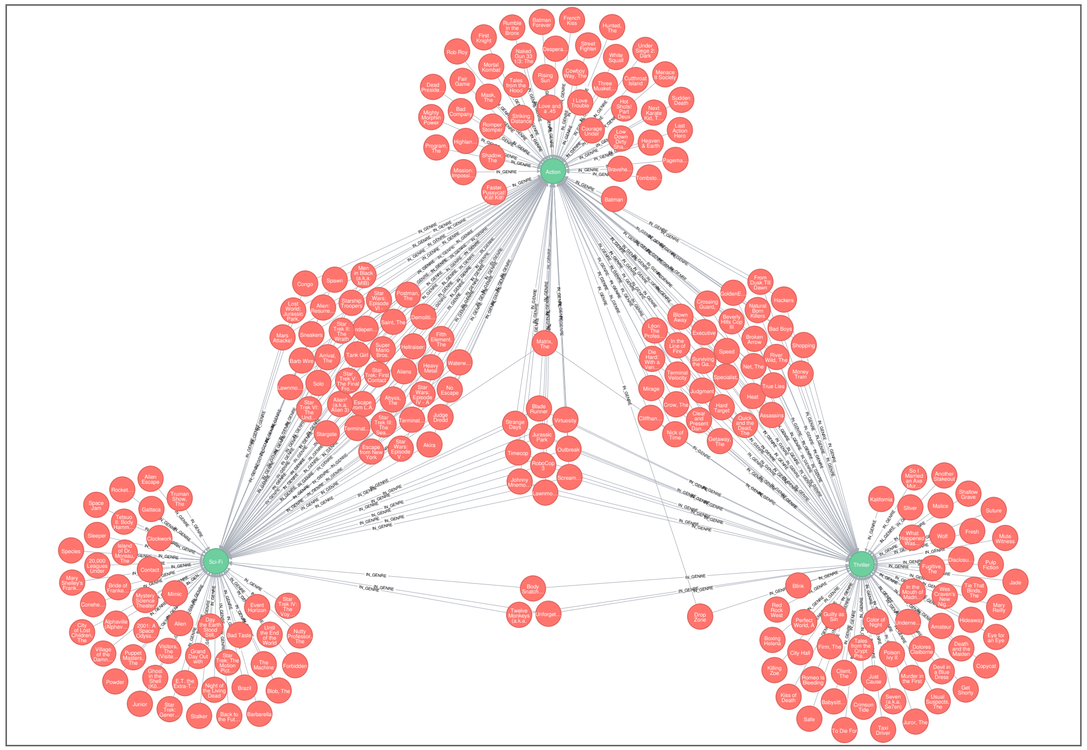
# Exercice 1: answer

- Search for a Movie by title substring

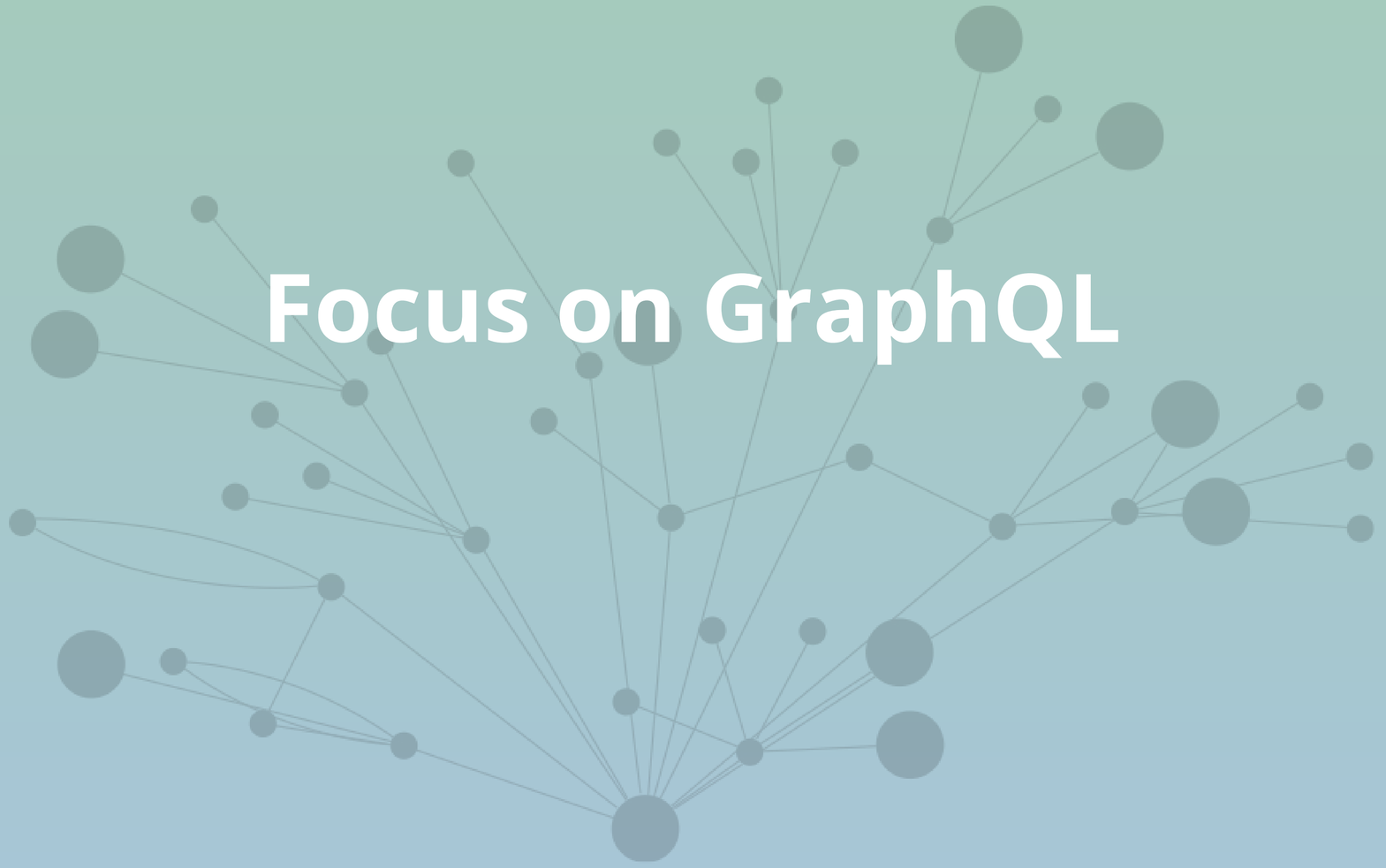
- For a given movie, find recommended movies

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

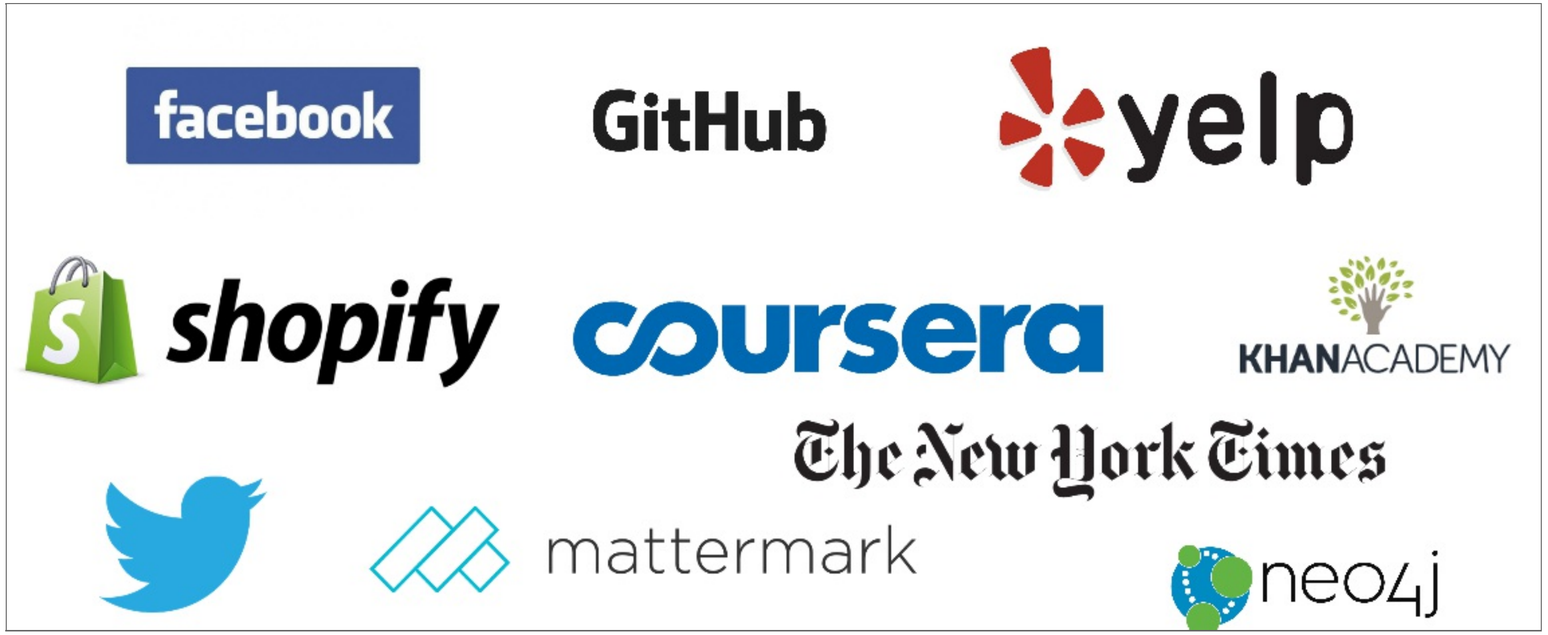
# Recommendation



# Focus on GraphQL



# A large adoption



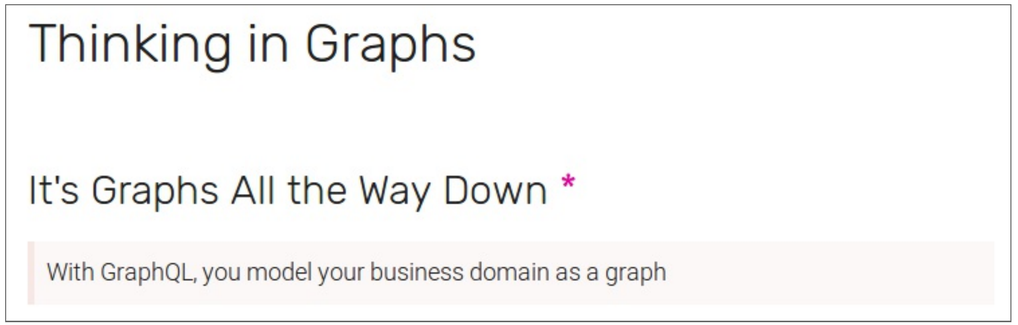
# What is GraphQl

**“A query language for your API”**

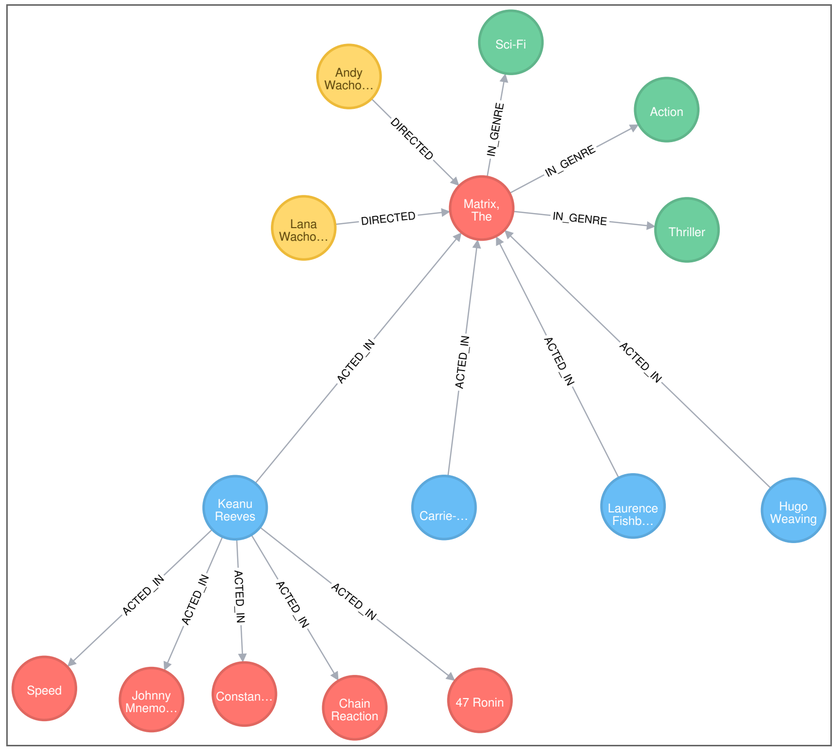
- Developed by Facebook iOS team for iOS app
  - Reduce number of round trip requests in face of low latency
- Declarative, state what fields you want Alternative to REST Self documenting (schema and types) Limited support for “queries” Logic is implemented in server

# 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”

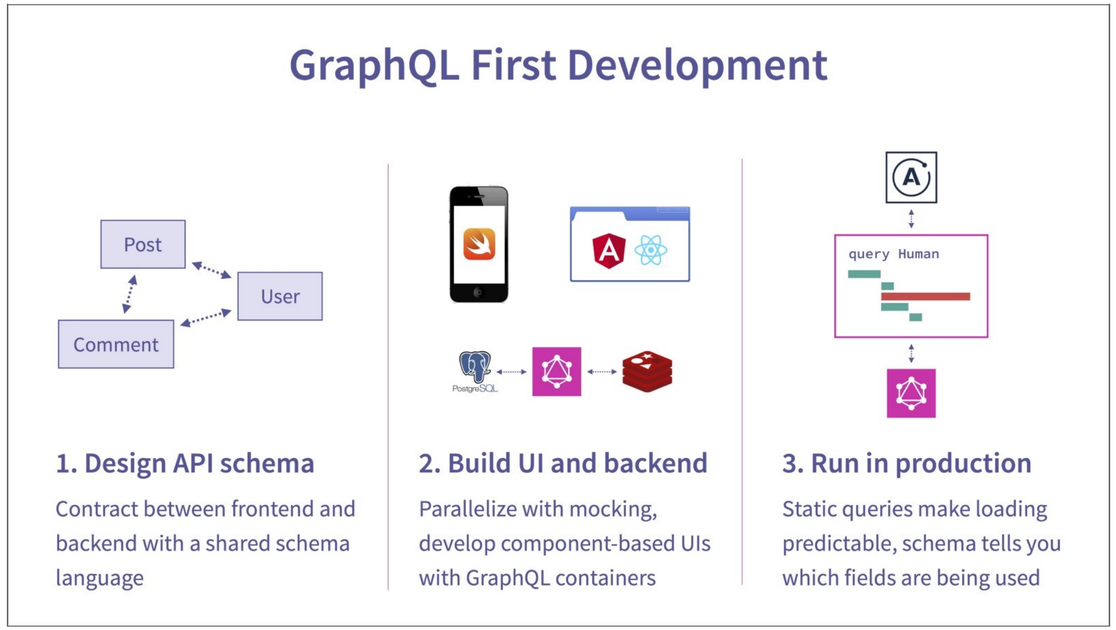


# Your application model is a Graph !



# Your application model is a Graph !

# 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

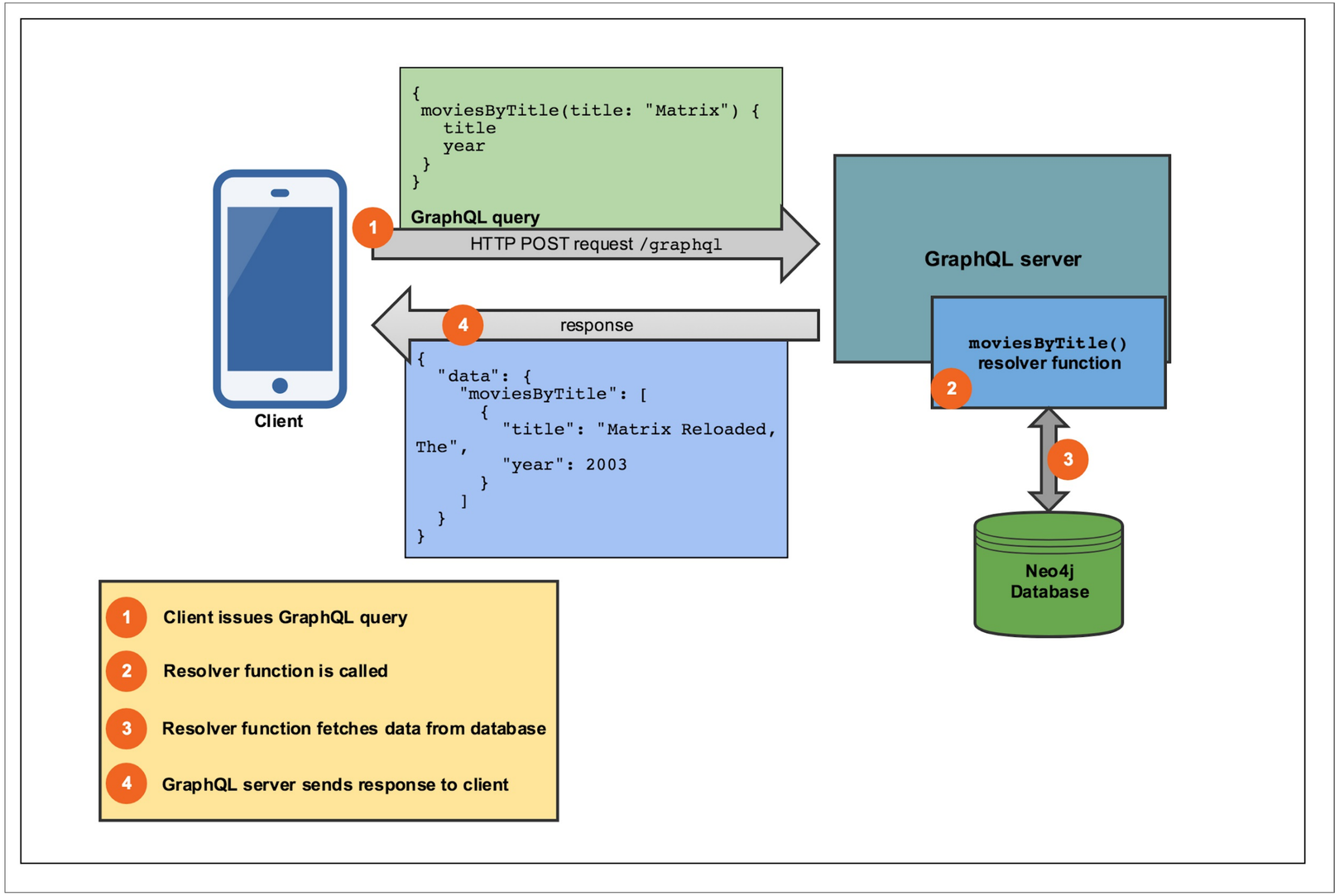
## Actor

# Queries / Mutation

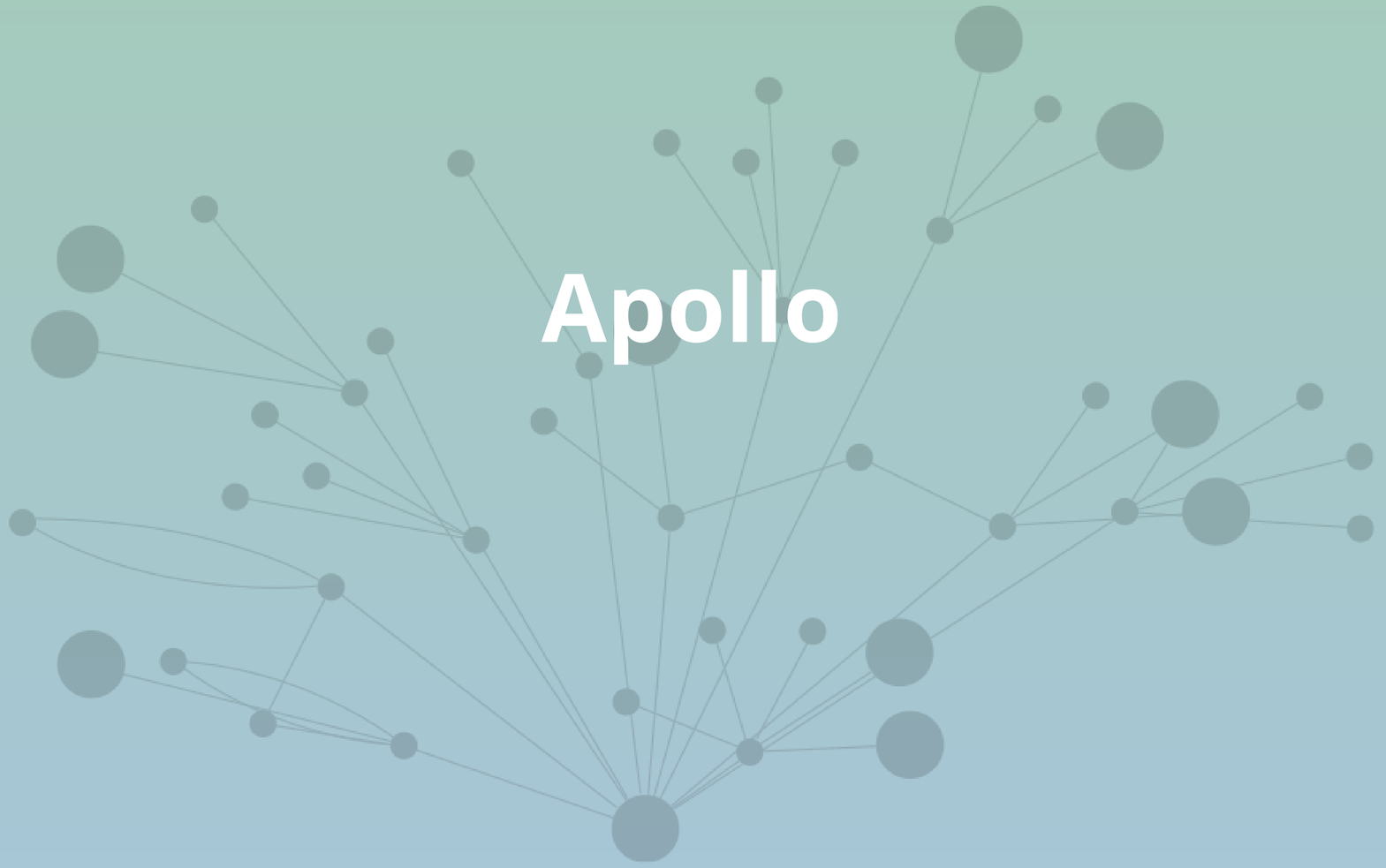
## Movie

## Actor

# API Workflow



# Apollo



neo4j

# Resolvers

**How to fetch the data**

# Apollo Launchpad

**https://launchpad.graphql.com/lkvl73r1xq**

Launchpad | New | Save | Download | Fork | GRAND Stack workshop - end | johnymontana

```
// GRAND Stack workshop - end state

// The goal of this section of the workshop is to complete our GraphQL server
// We start with a
// We need to query our Neo4j Database to ensure that we're

// Welcome to Launchpad!
// Log in to edit and save pads, run queries in GraphiQL on the right.

// graphql-tools combines a schema string with resolvers.
import { makeExecutableSchema } from 'graphql-tools';
import {v1 as neo4j} from 'neo4j-driver';

// Construct a schema, using GraphQL schema language
const typeDefs = `
  type Movie {
  movieId: ID!
  title: String
  year: Int
  plot: String
  poster: String
  imdbRating: Float
  genres: [String]
  similar(first: Int=3, offset:Int=0): [Movie]
}

type Query {
  moviesByTitle(subString: String!, first: Int=3, offset: Int=0): [Movie]
}
`;

// Provide resolver functions for your schema fields
const resolvers = {
  Query: {
    moviesByTitle: (root, args, context) => {
      let session = context.driver.session();
      let query = "MATCH (movie:Movie) WHERE movie.title CONTAINS $subString RETURN movie LIMIT $firs
      return session.run(query, args)
        .then( result => { return result.records.map(record => { return record.get("movie").propertie
    },
  },
  Movie: {
    genres: (movie, _, context) => {
      let session = context.driver.session();
      let params = {movieId: movie.movieId};
      let query = `
        MATCH(m:Movie)-[:IN_GENRE]->(g:Genre)
        WHERE m.movieId = $movieId
        RETURN g.name AS genre
      `;

      return session.run(query, params)
        .then( result => { return result.records.map(record => {return record.get("genre")})})

    },
```

GraphiQL | Prettify | Save | Reset | Headers

```
# Welcome to GraphiQL

query MovieListQuery($title: String!) {
      moviesByTitle(subString: $title, first: 30) {
        title
        movieId
        imdbRating
        plot
        poster
        year
        genres
        similar {
          movieId
          poster
          title
        }
      }
    }
```

QUERY VARIABLES

```
{
  "title": "Matrix"
}
```


```
{
 "data": {
   "moviesByTitle": [
     {
       "title": "Matrix, The",
       "movieId": "2571",
       "imdbRating": 8.7,
       "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/MV5BMTkxNDYxOTA4M15BMl5BanBnXkFtZTgwNTk0NzQx
MTE@._V1_SX300.jpg",
       "year": 1999,
       "genres": [
         "Thriller",
         "Sci-Fi",
         "Action"
       ],
       "similar": [
         {
           "movieId": "260",
           "poster": "http://ia.media-
imdb.com/images/M/MV5BOTIyMDY2NGQtOGJjNi00OTk4LWFhMDgtYmE3M2Ni
YzM0YTVmXkEyXkFqcGdeQXVyNTU1NTcwOTk@._V1_SX300.jpg",
           "title": "Star Wars: Episode IV - A New Hope"
         },
         {
           "movieId": "1196",
           "poster": "http://ia.media-
imdb.com/images/M/MV5BMjE2MzQwMTgxN15BMl5BanBnXkFtZTcwMDQzNjk2
OQ@@._V1_SX300.jpg",
           "title": "Star Wars: Episode V - The Empire
Strikes Back"
         },
         {
           "movieId": "480",
           "poster": "http://ia.media-
imdb.com/images/M/MV5BMjM2MDgxMDg0Nl5BMl5BanBnXkFtZTgwNTM2OTM5
NDE@._V1_SX300.jpg",
           "title": "Jurassic Park"
         }
       ]
     },
     {
       "title": "Matrix Reloaded, The",
       "movieId": "6365",
       "imdbRating": 7.2,
       "plot": "Neo and the rebel leaders estimate that they
have 72 hours until 250,000 probes discover Zion and destroy
it and its inhabitants. During this, Neo must decide how he
can save Trinity from a dark fate in his dreams.",
       "poster": "http://ia.media-
```


Secrets | npm Deps | Logs | GraphQL Endpoint: https://x57134qwl.lp.gql.zone/graphql | Docs | Powered by Auth0 Extend

# GraphQl Server

Just an express.js server with some custom endpoints.

# GraphQl Server - II

# 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**

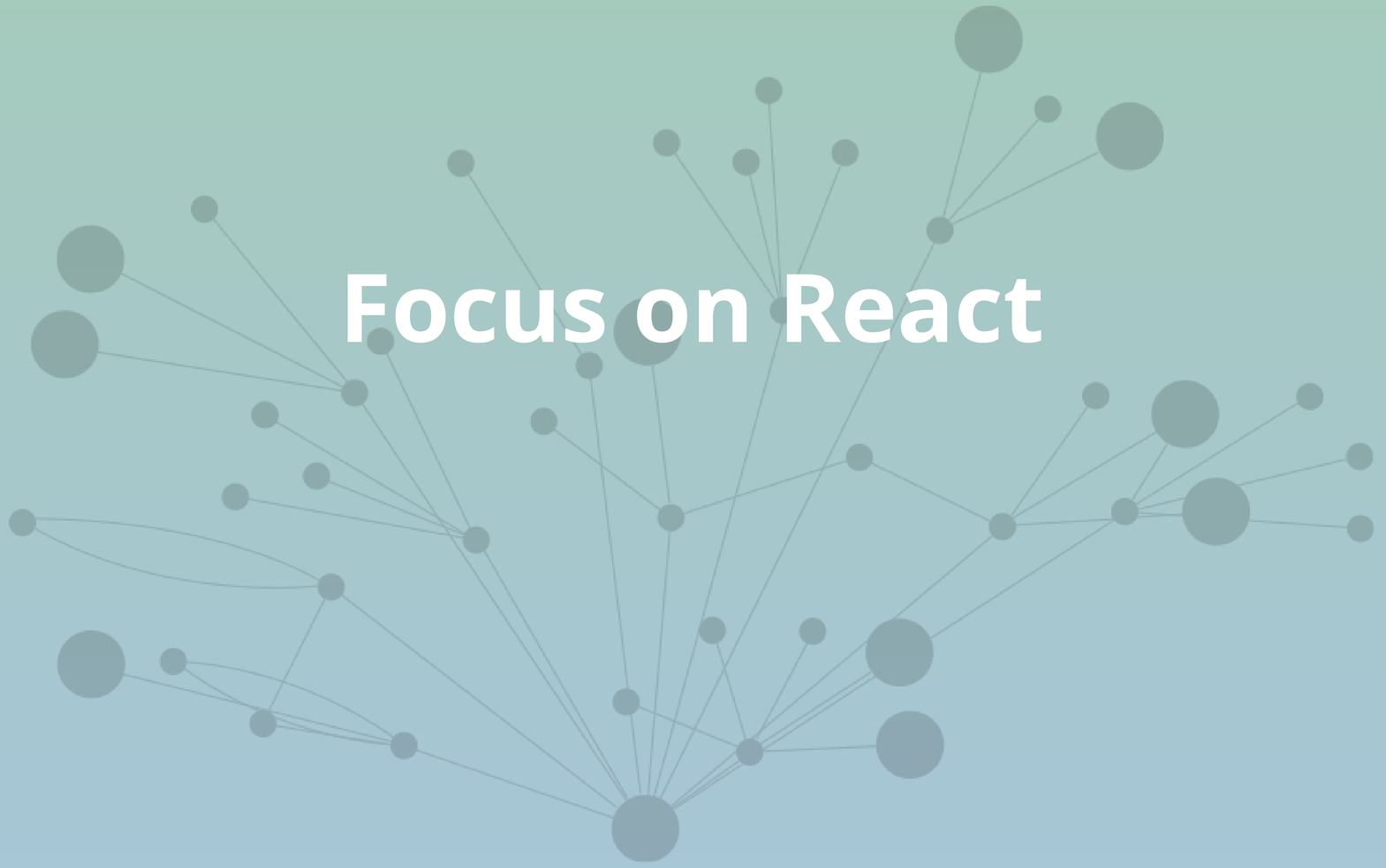
- Start with skeleton Launchpad: launchpad.graphql.com/3x984k8mv
- Fork it, then add your Neo4j Sandbox credentials to secrets
- Complete the GraphQL service implementation by adding your Cypher queries from Exercise 1
- [optional] Use the JS GraphQL→ Cypher integrations instead of Cypher in resolver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

# React Apollo : wrap application

Wrap your react application with `ApolloProvider`

# React Apollo : component

====!

====!

- Using React’s Higher Order Component feature : we can populate a component’s props to a GraphQL query
- graphql() enhancer function
- Now whenever TodoApp component is rendered, the GraphQL query is executed

# React Apollo : Use the query component

# 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