# Movie Score

Web Mining

Steven Liatti & Jeremy Favre

# Sommaire

- Introduction
- Présentation des données
- Algorithmes appliqués
- Architecture
- Implémentation
  - Crawler
  - Parser
  - Frontend
- Demo
- Conclusion

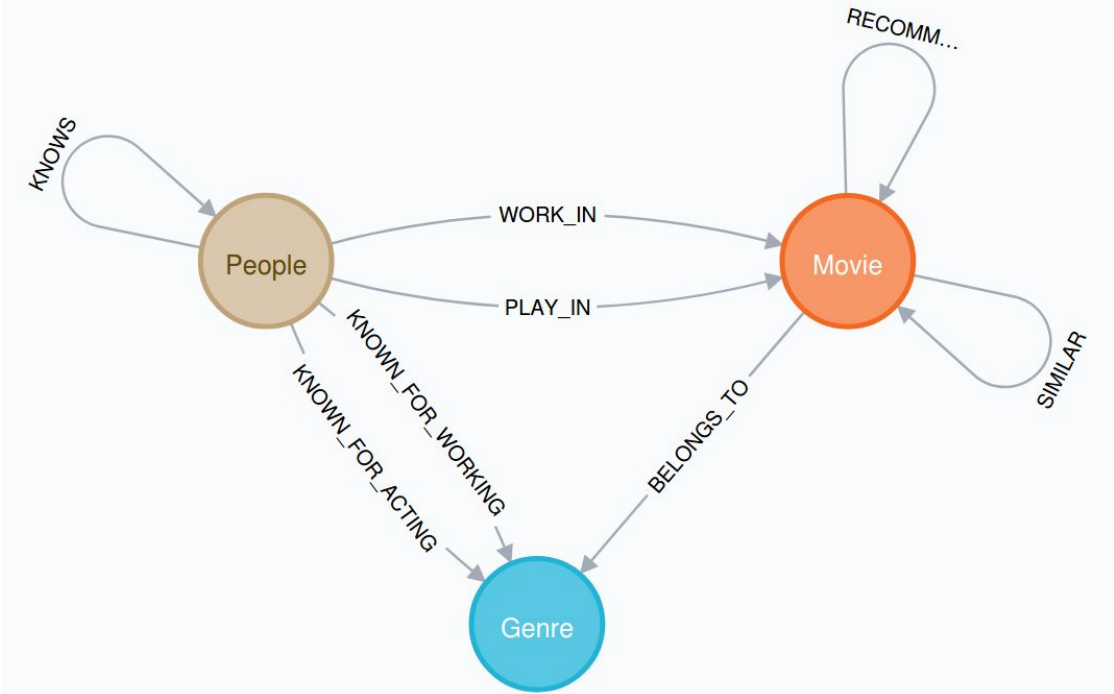
# Introduction

- Données en provenance de TMDb
  - Récupération des données via l'API ⇒ Crawler
  - Insertion dans une base de données orientée graphe
- Calcul du score pour chaque film
  - Score = revenu / budget
- Application d'algorithme de graphe
  - Page rank, shortest path, centrality etc.
- Réalisation d'un client Web pour visualiser les données

# Présentation des données

- Export de la base TMDb
- Format JSON

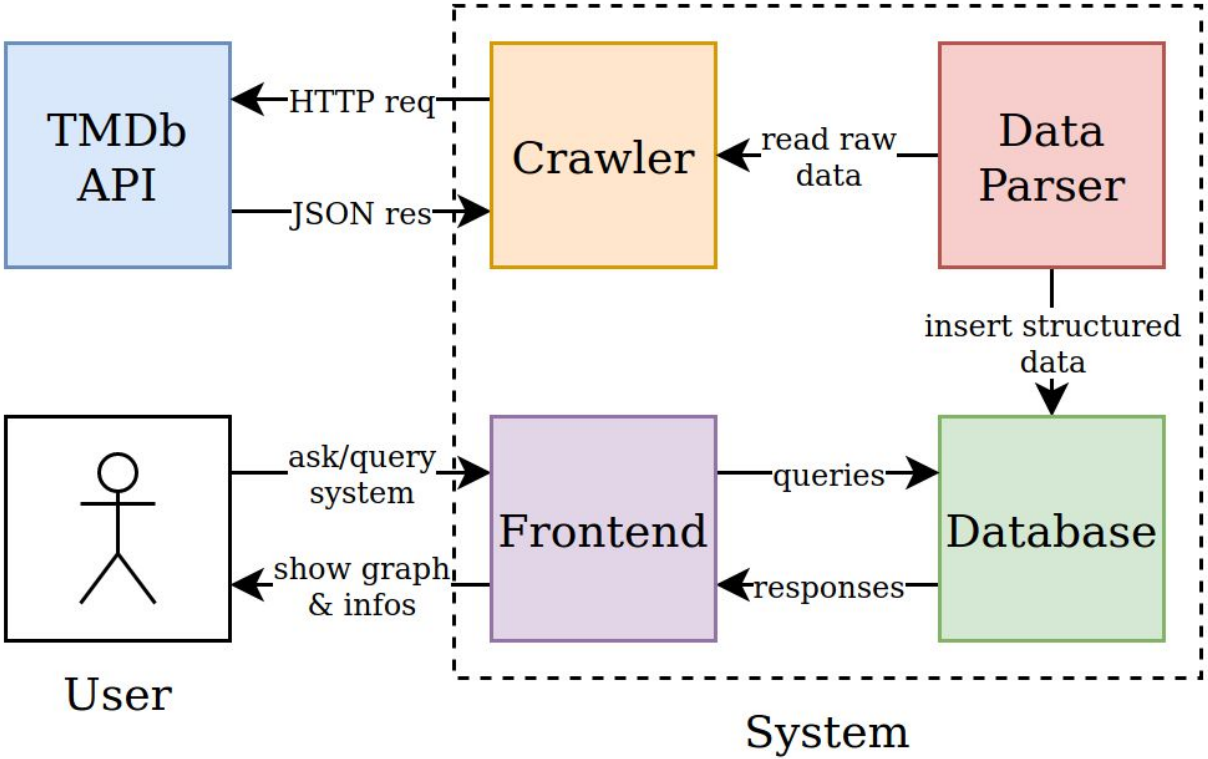

```
{
    "budget": 140000000,
    "revenue": 655011224,
    "genres": [
        { "id": 12, "name": "Adventure" },...
    ],
    "id": 22,
    "title": "Pirates of the Caribbean: The Curse of the Black Pearl",
    "original_language": "en",
    "overview": "Jack Sparrow, a freewheeling ...",
    "release_date": "2003-07-09",
    "runtime": 143,
    "spoken_languages": [
        { "iso_639_1": "en", "name": "English"}
    ],
    "status": "Released",
    "popularity": 54.847,
    "vote_average": 7.7,
    "vote_count": 14020,
    "credits": {
        ...
    },
    "keywords": {
        ...
    },
    "similar": {
        "results": [
            ...
        ]
    }
}
```

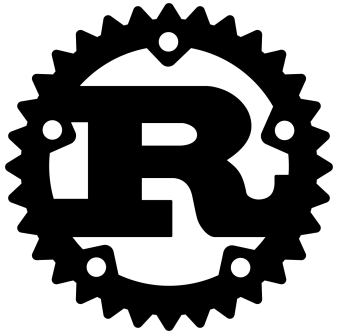
# Algorithmes appliqués

- Algorithmes de centralité
  - Page Rank
  - Degré de centralité
- Algorithme de communautés
  - Algorithme de Louvain
- Algorithmes de similarité entre noeuds
  - Node Similarity ⇒ films & peoples
- Algorithmes de plus court chemin
  - Shortest Path

# Architecture



# Implémentation - Crawler

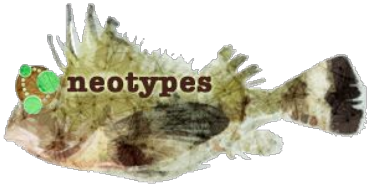


- Approche naïve avec bash
  - Quelques films par seconde
  - 500'000 requêtes ≈ 2-3 semaines
- Crawler parallèle en Rust
  - 70 machines accessibles via ssh
  - Chaque machine a lancé le crawler sur 20 threads avec un fichier d'II
  - Equivalent de 70 * 20 = 1400 scripts bash initiaux
- Données récupérées en quelques minutes

# Implémentation - Parser



- Parser réalisé en Scala
  - Sérialisation des données JSON en cases classes
  - Langage à typage statique fort
- Insertion des données dans une base Neo4j
  - Orientée graphe
  - Propose de nombreuses features (algorithmes + visualisation)
- Insertion dans l'ordre
  - Insertion des noeuds
  - Insertion des relations
  - Calcul des algorithmes
- Utilisation du driver neotypes
  - Basé sur le driver java



# Frontend


- Neovis.js
  - Wrapper entre le driver Neo4j et code JS
- Javascript
  - Logique
  - Neo4j Driver for JavaScript ⇒ Autocompletion & textual request
- Bootstrap
  - Design & Autocompletion
  - Responsive web design
- HTML/CSS
  - Rendu de la page web


Demo

# Conclusion

- Crawling de TMDb + insertion dans neo4j
- Application d'algorithmes de graphe sur les données
- Réalisation d'un client JS permettant la visualisation
- Etat final du projet
  - Planification respectée
  - Features prévues réalisées (et + encore)
  - Satisfaits du rendu visuel
- Améliorations
  - Performances parallèles du parser
  - Amélioration du client avec librairie graphique qui propose des effets encore plus poussé