

# W gąszczu grafów

czyli jak zostać bohaterem swoich danych w Neo4j

a code

# a code

---

```
public void m() {  
    n(this);  
}  
  
public static n(Object obj) {  
    if(obj==null) {  
        doSomething();  
    }  
}
```

---

# inlining

---

```
public void m() {  
    if(this==null) {  
        doSomething();  
    }  
}
```

---

# null check folding

---

```
public void m() {  
    if(false) {  
        doSomething();  
    }  
}
```

---

# dead code elimination

---

```
public void m() {  
}
```

---

and last but not least

-XX:+UseFastEmptyMethods



**WAT?**

# W gąszczu grafów

## czyli jak zostać bohaterem swoich danych w Neo4j

# O mnie czyli kim jestem

**Jarek Pałka**

- Allegro.tech, obecnie tajny projekt o którym nie mogę mówić
- JDD/4Developers, naczelný sprawca zamieszania czyli człowiek od kłopotów,
- a tak poza tym jeśli cokolwiek ma związek z JVM i HotSpot, bytecode'em, parserami, językami programowania na JVM oraz technikami JIT, wcześniej czy później się tym zainteresuje :)
- były architekt, manager, sysadmin i człowiek od wydajności,

# O mnie czyli kim chciałbym być

- Neo4j, obecnie modelowanie rzeczywistości w grafach, w przyszłości rozbudowa parsera Cypher, lepsze narzędzia do importu danych
- OpenJDK kontrybutor, może w przyszłym życiu
- Python3 na JVM, ale to już na emeryturze :)

Wstęp do grafów

# Odświeżający łyk teorii

graf  $G$  składa się z dwóch zbiorów –  $V$  oraz  $E$ , przy czym  $V$  jest niepustym zbiorem, którego elementy nazywane są wierzchołkami, a  $E$  jest rodziną dwuelementowych podzbiorów zbioru wierzchołków  $V$ , zwanych krawędziami

---

# Gatunki i podgatunki

- graf prosty
- graf skierowany
- graf mieszany
- graf z wagami
- hipergraf

# Ważne pojęcia

- gęstość grafu - stosunek ilości wierzchołków do krawędzi
- droga/ścieżka - kolekcja wierzchołków lub krawędzi
- cykl - kolekcja połączonych wierzchołków, gdzie pierwszy element jest taki sam jak ostatni
- klika - podzbiór wierzchołków, gdzie istnieje połączenie pomiędzy każdym z każdym
- stopień wierzchołka - ilość krawędzi wychodzących z wierzchołka

# Algorytmy i inne znane i lubiane pojęcia

- algorytmy A\*, Dijkstra, Bellman-Ford, Floyd-Warshall,
- przeszukiwania grafu wgłąb i wszerz,
- "clusters"
- "connected components"
- "small worlds"

Wstęp do Neo4j

# Wstęp do Neo4j

---

“A Graph – records data in > Nodes – which have > Properties”

---

“Nodes – are organized by > Relationships – which also have > Properties”

---

“Nodes – are grouped by > Labels – into > Sets”

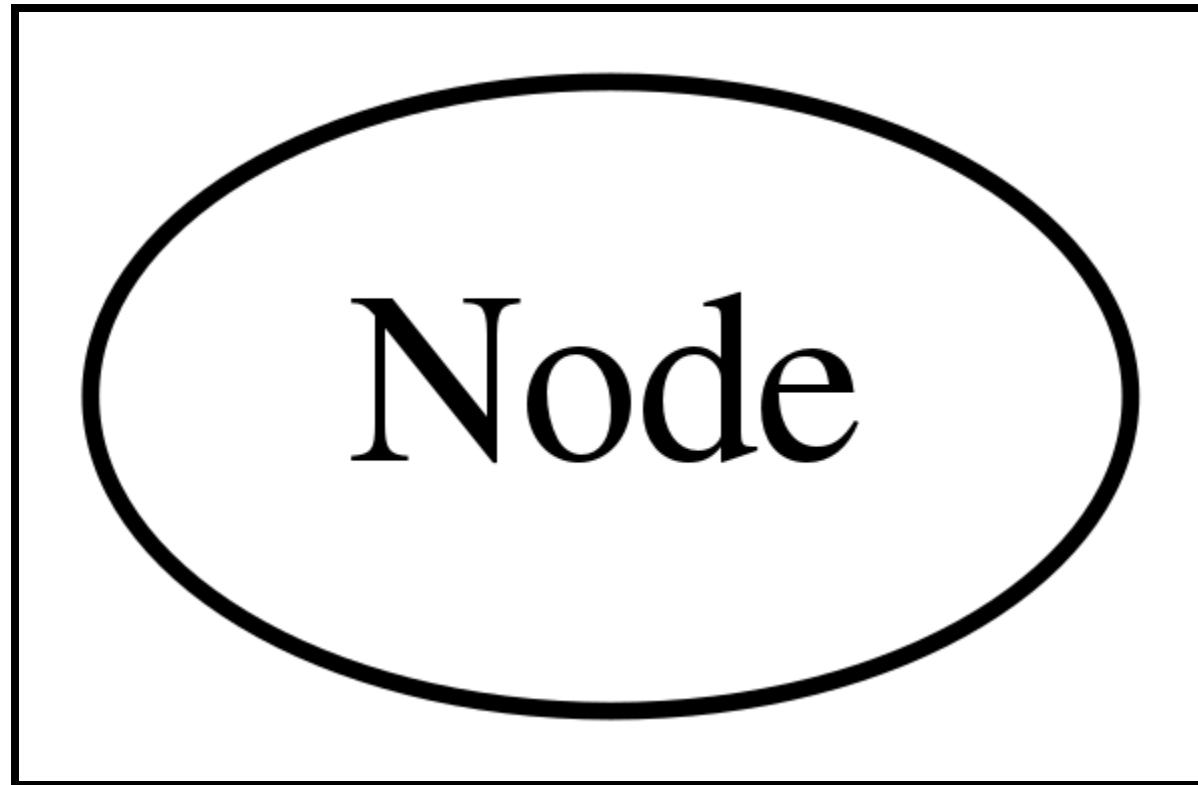
# Słowem wyjaśnienia

- Neo4j to baza danych a nie nakładka na inną bazę danych
- Własny mechanizm przechowywania danych, zoptymalizowany pod strukturę grafów
- Dostępne wsparcie dla następujących modeli:
  - embedded
  - serwer (REST)
  - serwer (binary protocol, prace ciągłe trwają)
  - HA (edycja "enterprise")

# Podstawowe elementy

- węzły ("node")
  - identyfikator
  - etykiety ("labels")
  - własności

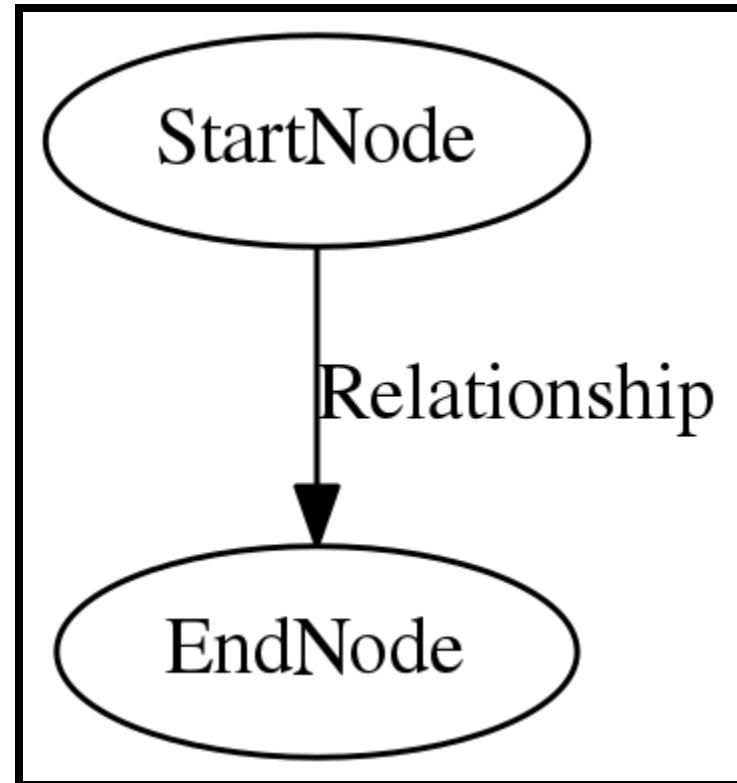
# Węzeł



# Podstawowe elementy

- relacje ("relationship")
  - typ relacji ("relationship type")
  - kierunek ("direction")
  - własności

# Relacje



# Podstawowe elementy

- schemat ("schema")
  - indeksy ("index")
  - ograniczenia ("constraints")

# Własności węzłów i relacji

- właściwości ("properties")
  - prymitywy Java (long, int, float, etc.)
  - łańcuchy znakowe
  - tablice powyższych

Podstawy Cypher

# Podstawy Cypher

---

\*Looks like SQL, feels like pattern-matching\*

---

```
MATCH (n:Label) RETURN n;  
MATCH (n) - [:LIKES] ->m return m;  
MATCH (n) - [r:LIKES] ->m return r;
```

---

# Dopasowywanie w grafach

---

(a) --> (b)  
(a) --> (b) <-- (c)  
(a:User) - -> (b)  
(a:User:Admin) - -> (b)  
(a) - [r:REL\_TYPE] -> (b)  
(a) - [\*2] -> (b)  
(a) - [\*3..5] -> (b)

---

# "Patterns are expressions too"

Wzorce dopasowania, są wyrażeniami, które zwracają kolekcję węzłów i relacji (ścieżek). Wszędzie tam gdzie możesz użyć wyrażenia (klauzula 'WHERE', 'RETURN', 'WITH', etc.), możesz też użyć wzorców dopasowań.

---

```
MATCH a-[:REL]-b WHERE NOT b-[:REL]-c RETURN b;  
MATCH a-[:REL]-b RETURN a, b-[:REL]-c;  
MATCH a-[:REL]-b WITH a, b-[:REL]-c RETURN a, count(distinct c);
```

---

# Praca z kolekcjami

---

```
RETURN [0,1,2,3,4,5,6,7,8,9] AS collection
RETURN range(0,10) [3]
RETURN length(range(0,10) [0..3])
RETURN [x IN range(0,10) WHERE x % 2 = 0 | x^3] AS result
```

---

# Predykaty i zawężanie list

---

```
RETURN [x IN range(0,10) WHERE x % 2 = 0 | x^3] AS result;

WHERE a.name='Alice' AND b.name='Daniel' AND
ALL (x IN nodes(p) WHERE x.age > 30);

WHERE a.name='Eskil' AND ANY (x IN a.array WHERE x = "one");

WHERE n.name='Alice' AND NONE (x IN nodes(p) WHERE x.age = 25);

WHERE n.name='Alice' AND
SINGLE (var IN nodes(p) WHERE var.eyes = "blue");

MATCH (n)
WHERE EXISTS(n.name)
RETURN n.name AS name, EXISTS((n)-[:MARRIED]->()) AS is_married
```

---

# Ścieżki

---

```
MATCH p = ((a)-[rels]-(b)) RETURN nodes(p)
MATCH p = ((a)-[rels]-(b)) RETURN relationships(p)
MATCH p = ((a)-[:RAILS]->(b)) UNWIND nodes(p) AS n return DISTINCT n;
```

---

# Potoki

Klauzula 'WITH' umożliwia przetwarzanie zapytań z w potokach, gdzie wynik poprzedniego zapytania jest przekazywany do następnego, myślmy o tym jak o '|' w \*NIX.

---

```
MATCH (david { name: "David" }) -> (otherPerson) -> ()
WITH otherPerson, count(*) AS foaf
WHERE foaf > 1
RETURN otherPerson
```

# Pułapki Cypher

Ponieważ Cypher jest składniowo, młodszym bratem SQL,  
wpadniesz w pułapkę pisania zapytań relacyjnych,  
które będą wykorzystywać iloczyn kartezjański.

---

To nie jest błogosławiona droga grafu.

---

# Przygotowanie warsztatu

Na nośniku USB lub też dla tych połączeniem:

- prezentacja
- dane do 3 modeli z którymi będziemy pracować
- dokumentacja neo4j
- neo4j-community-2.2.3

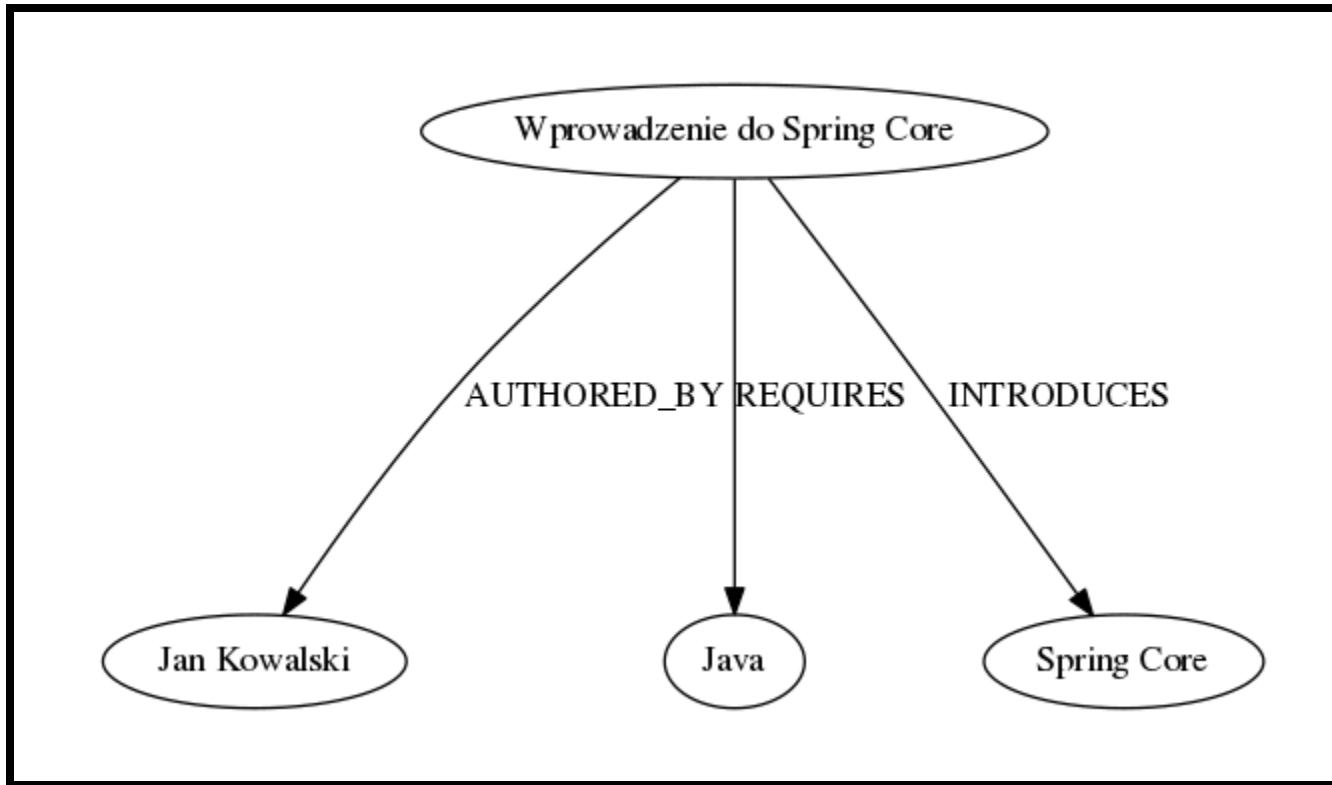
# Narzędzia pracy

- konsola neo4j-shell
- konsola interaktywna

# Model zerowy, czyli coś na rozgrzewkę

Zacznijmy od prostego modelu szkoleń, trenerów i umiejętności.

# Model



# Przygotowanie danych

---

## CREATE

```
(t0:Training {title : "Archeologia kodu a architektura" }) ,  
(n0:Trainer { name : "Jarosław Pałka"}),  
(s0:Skill { name : "Git" }) ,  
(s1:Skill { name : "SonarQube" }) ,  
t0 - [:AUTHORED_BY] ->n0 ,  
t0 - [:REQUIRES] ->s0 ,  
t0 - [:INTRODUCES] ->s1 ;
```

---

# Ćwiczenie 1

Znajdź wszystkie szkolenia przygotowane przez danego trenera.

# Ćwiczenie 2

Znajdź najszybszą ścieżkę szkoleń o danej umiejętności do innej docelowej umiejętności.

---

Podpowiedź:

Cypher oferuje funkcję 'shortestPath', która zwraca najkrótszą ścieżkę.

**Sorry, wrong answer.**



# reduce na ratunek

```
MATCH (a:Skill {name:"Java"})-[:REQUIRES]- (b:Training)
MATCH (c:Training) - [:INTRODUCES] -> (d:Skill {name:"REST"})
MATCH p=allShortestPaths (b)-[rels:REQUIRES|INTRODUCES*]-c)
WHERE length(p)%2=0 AND ALL(idx IN range(0, length(p)-2, 2) WHERE type(re
RETURN p
```

---

# Model pierwszy, czyli co kryje Twoja skrzynka pocztowa

W tym modelu, naszym grafem będzie skrzynka pocztowa.

Poszukamy węzłów i krawędzi pośród:

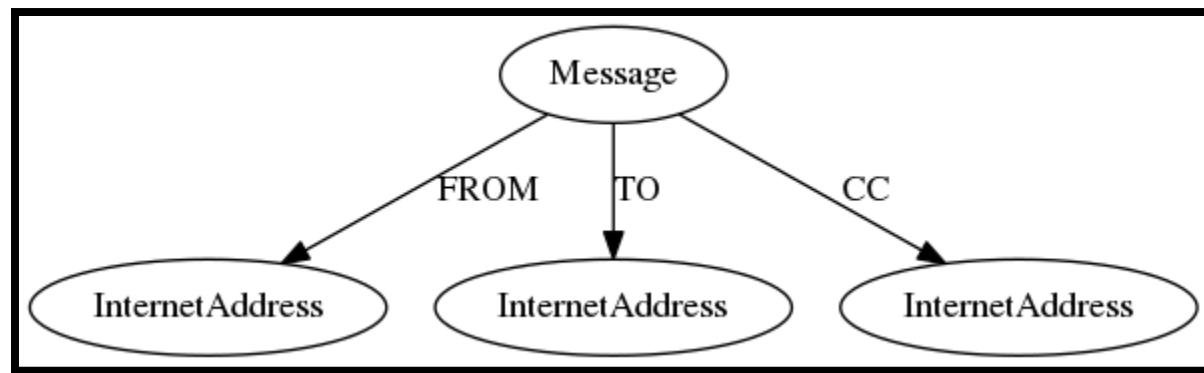
- wiadomości email
- osób wysyłających i odbierających

# Przygotowanie danych

Aby pracować z tym modelem, należy skopiować zawartość katalogu datasets-mails.workshop. Zawiera on zanimizowane maile z mojej skrzynki mailowej, celem nadanie ćwiczeniu więcej realizmu.

# Zaczniemy od modelu

A poniżej model grafu po zimportowaniu zawartości skrzynki pocztowej.



# Ćwiczenia 1

Wyszukajmy wszystkie maile które są odpowiedzią na innego maila.

---

Podpowiedź:

węzły z etykietą `Message` posiadają pola  
`Message-Id` oraz `In-Reply-To`.

---

# Ćwiczenie 2

Znajdźmy najdłuższy wątek mailowy.

---

Podpowiedź:

na początek spróbuj zidentyfikować te wiadomości,  
które są początkiem i końcem wątku

---

# Słów kilka o wydajności zapytań

Cypher umożliwia analizę planu zapytań, w celu ich optymalizacji.

```
EXPLAIN  
MATCH (p:Person { name:"Tom Hanks" })  
RETURN p
```

```
PROFILE  
MATCH (p:Person { name:"Tom Hanks" })  
RETURN p
```

---

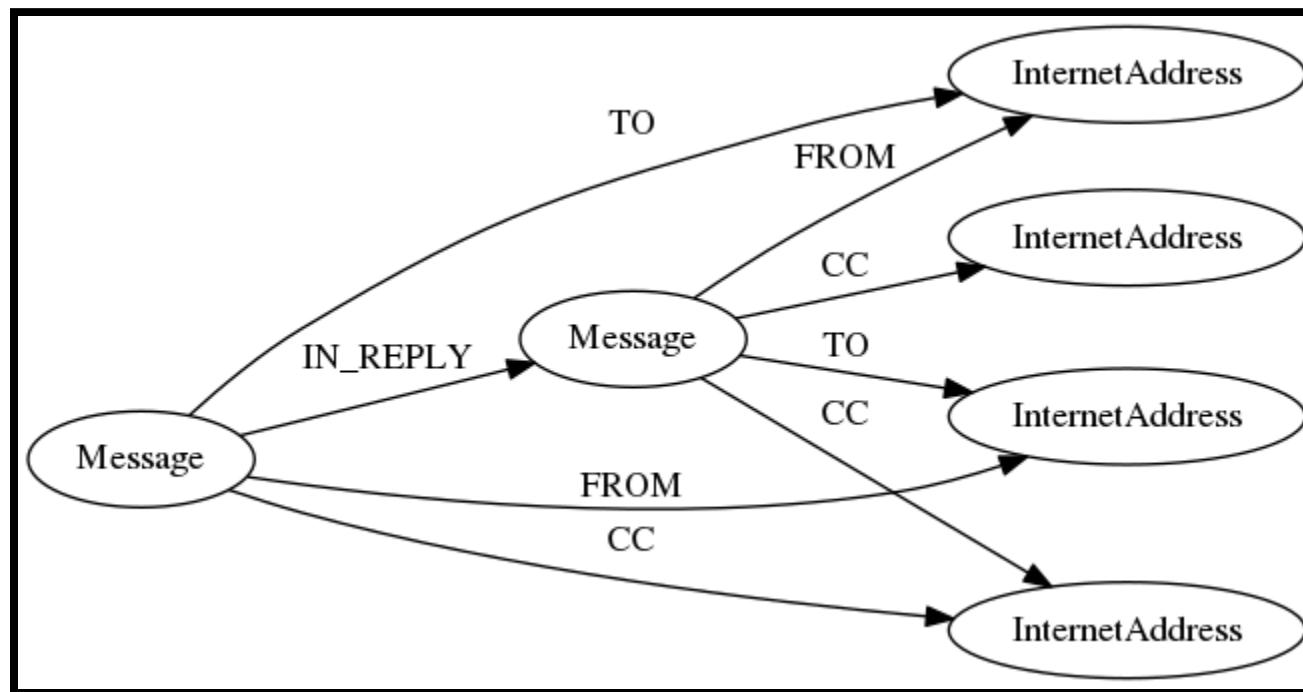
Sugestia:

Bądź jak najbardziej specyficzny w swych zapytaniach,  
wykorzystuj etykiety i typy relacji.

---

# Ćwiczenie 2b

Prawdopodobnie napisaliście to zapytanie myśląc SQLem i to jak najbardziej w porządku. Zobaczmy teraz co tak naprawdę potrafi Neo4j i Cypher, czyli czas na utworzenie nowych relacji.



# Indeksy ku chwale wydajności

---

```
CREATE INDEX ON :Message(`In-Reply-To`);  
CREATE INDEX ON :Message(`Message-Id`);
```

Sprawdźmy teraz status indeksu

```
neo4j-sh (?)$ schema  
Indexes  
  ON :Message(In-Reply-To) ONLINE  
  ON :Message(Message-Id) ONLINE  
No constraints
```

---

# Ćwiczenie 2c

Tym razem wyszukajmy najdłuższego wątku, korzystając z nowych krawędzi.

---

Podpowiedź:

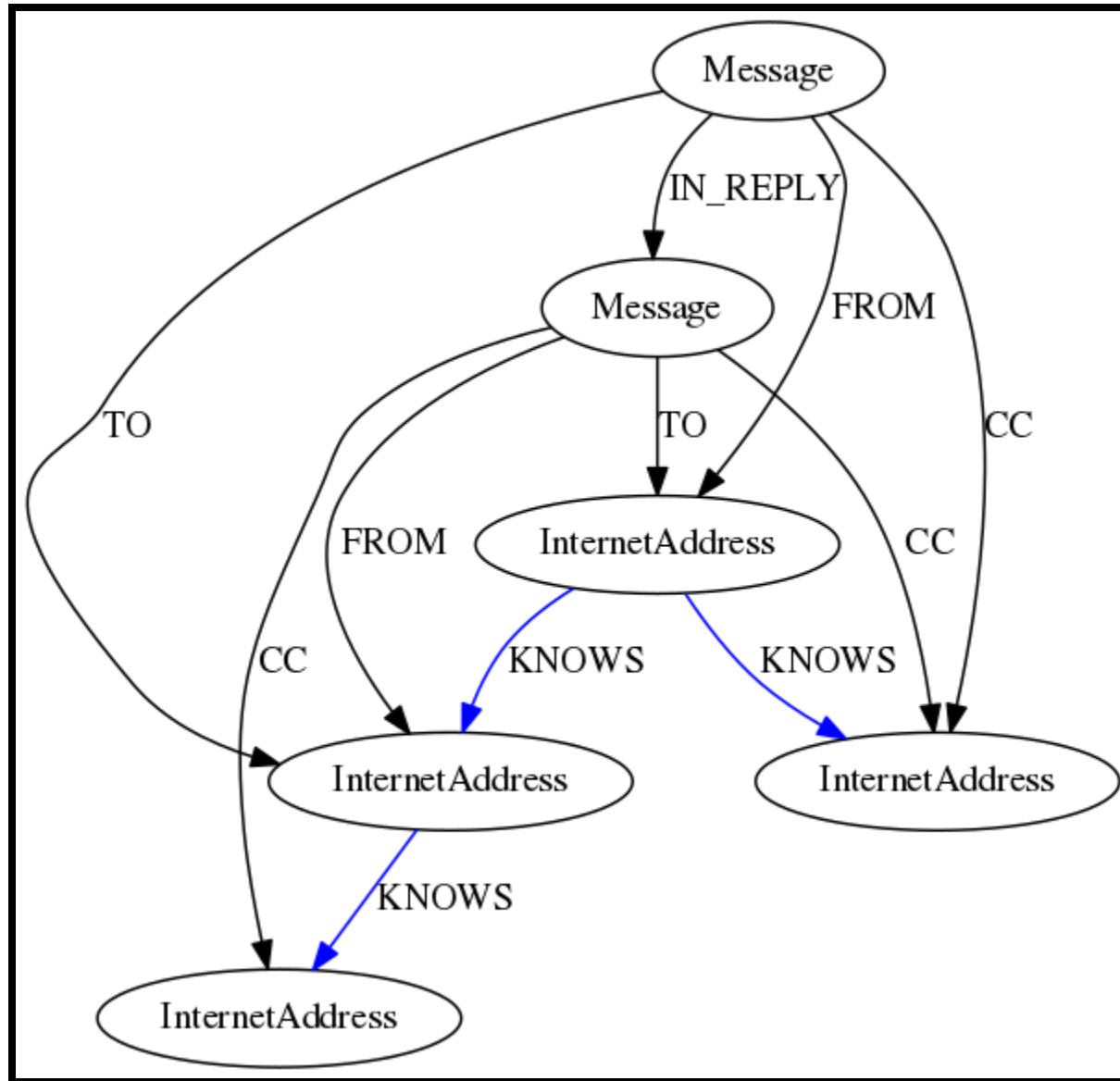
Wzorce w Cypher mogą być traktowane jak wyrażenia,  
nie pasujący wzorzec zwraca pusty rezultat

---

# Ćwiczenie 3

Spróbujmy odnaleźć znajomych w naszej skrzynce, czyli "let's go social".

# Ćwiczenie 3 / model



# Ćwiczenie 4

Poszukajmy grup znajomych w naszym grafie.

Podpowiedź:

Uwaga na węzeł, który jest znajomym wszystkich,  
czyli właściciel skrzynki pocztowej, węzeł o id=2.

Możemy go usunąć, lub dla zachowania struktury grafu,  
nadać mu inną etykietę.

---

Definicja:

clusters - tightly knit groups characterised by a relatively high density of ties; this likelihood tends to be greater than the average probability of a tie randomly established between two nodes

# Ćwiczenie 4a

Dwie podstawowe techniki wyznaczania klastrów to "Global clustering coefficient" oraz "Local clustering coefficient" ([https://en.wikipedia.org/wiki/Clustering\\_coefficient](https://en.wikipedia.org/wiki/Clustering_coefficient)).

Spróbujmy policzyć współczynnik techniką "local clustering coefficient".

# Kęs teorii

Współczynnik ten jest liczony jako prawdopodobieństwo że dwa sąsiadujące losowe wybrane węzły także są ze sobą połączone.

$$C_i = \frac{|\{e_{jk} : v_j, v_k \in N_i, e_{jk} \in E\}|}{k_i(k_i - 1)}$$

gdzie sąsiedztwo  $N_i$  określamy jako:

$$N_i = \{v_j : e_{ij} \in E \vee e_{ji} \in E\}$$

i  $k_i$  to liczba sąsiadów danego węzła.

# Wszystkie ręce na konsole

Wystarczy policzyć ilość bezpośrednich sąsiadów, i ilość relacji pomiędzy nimi.

# "Go deeper"

---

Definicja:

strongly connected components - a graph is said to be strongly connected if every vertex is reachable from every other vertex

---

Dwa podstawowe algorytmy to algorytm Kosaraju oraz algorytm Tarjan. Ciągle kombinuję jak zapisać to w Cypher :), dla wytrwałych nagroda :)

# Algorytm Tarjan

---

```
algorithm tarjan is
    input: graph G = (V, E)
    output: set of strongly connected components (sets of vertices)

    index := 0
    S := empty
    for each v in V do
        if (v.index is undefined) then
            strongconnect(v)
        end if
    end for

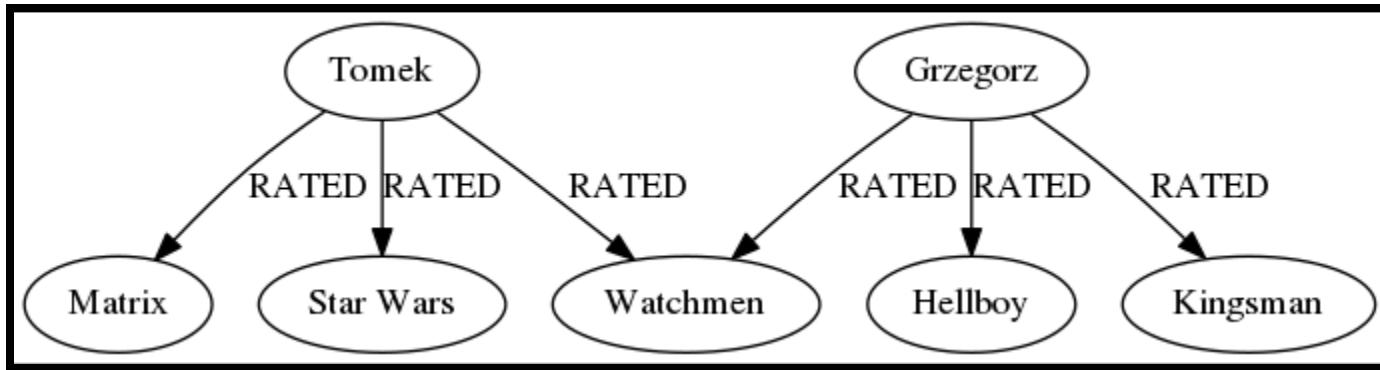
    function strongconnect(v)
        // Set the depth index for v to the smallest unused index
        v.index := index
        v.lowlink := index
        index := index + 1
        S.push(v)
```

---

# Model drugi, czyli chodźmy do kina na film

W tym celu wykorzystamy bazę MovieLens, tworzoną w ramach projektu GroupLens Research, przy Uniwersytecie Stanu Minnesota. Baza zawiera dane o filmach i rekomendacjach, wykorzystując system ocen punktowych w skali od 1-5.

# Zacznijmy od modelu



# Szybki kurs hodowania grafów

```
LOAD CSV FROM 'file:data.csv' AS line FIELDTERMINATOR '|'
WITH line[0] AS firstname, line[0] AS lastname
CREATE (:Person {firstname : firstname, lastname : lastname});
```

---

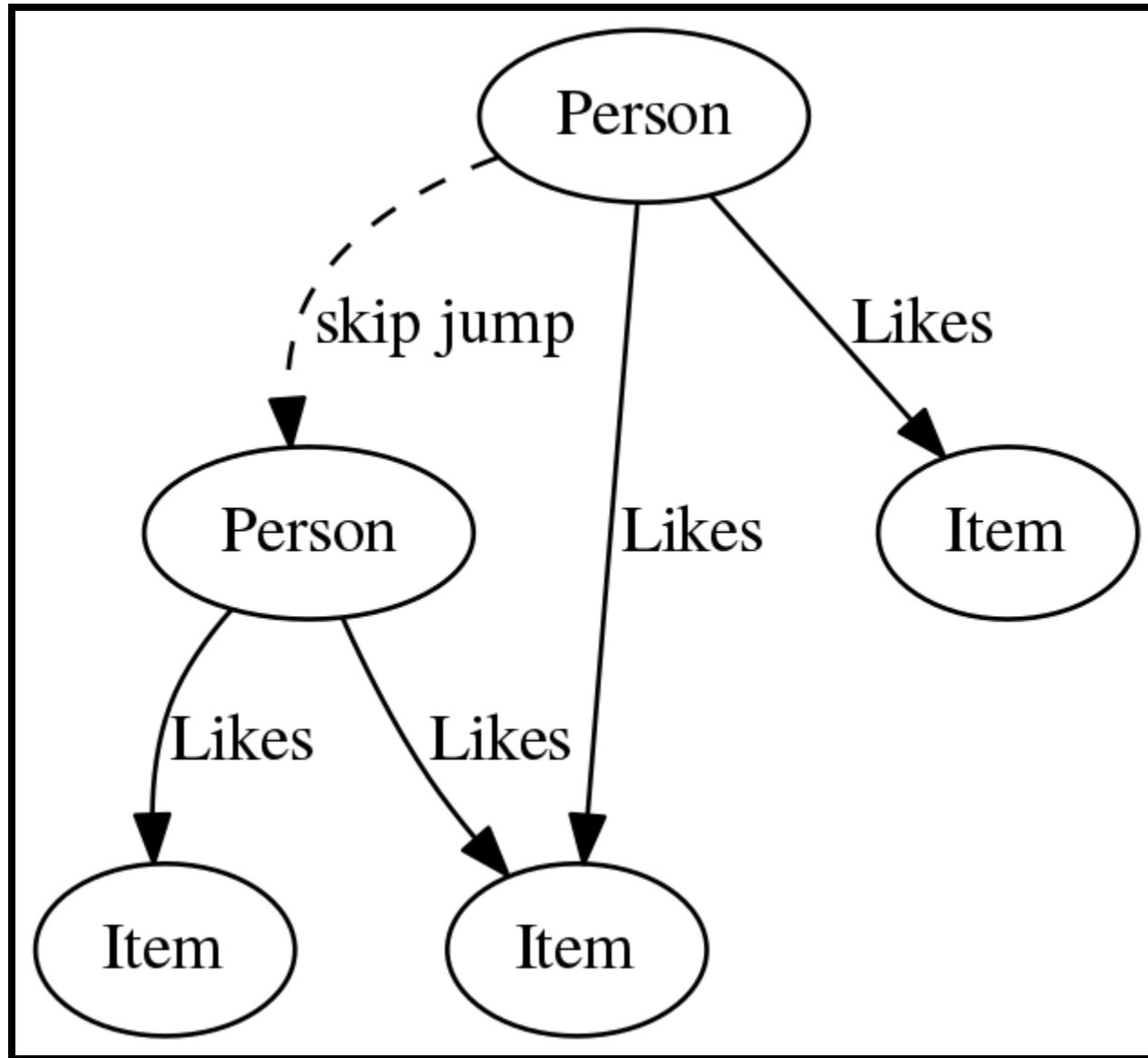
```
LOAD CSV WITH HEADERS FROM 'file:data.csv' AS line
CREATE (:Person {firstname : line.firstname, lastname : line.lastname});
```

---

# Ćwiczenie 1

Rekomendacje z wykorzystaniem algorytmu "skip jump", jest to technika oparta o prostą obserwację. Jeśli dwie osoby mają przynajmniej jedną relację do tego samego węzła, w tym przypadku filmu, to oznacza że istnieje wysokie prawdopodobieństwo, że osoba ta polubi także pozostałe filmy polubione przez te drugą osobę.

# Model



# Ćwiczenie 2

Rekomendacje z wykorzystaniem techniki "hammock", to rozszerzenie techniki "skip jump", która bazuje na więcej niż jednym elemencie wspólnym dla dwóch wybranych węzłów.

Kolejnym rozwinięciem tej techniki, jest szukanie rekomendacji, gdy dwa lub więcej elementów są tego samego typu/kategorii, np.  
gatunek filmowy.

---

Podpowiedź:

W pliku u.item zawarte są informacje o gatunku filmowym,  
spróbujmy je zaimportować do naszej bazy danych.

---

# Ćwiczenie 3

A teraz coś bardziej zakręconego, czyli collaborative filtering, k-nearest neighbors i cosine similarity.

$$\cos(\theta) = \frac{A \cdot B}{||A|| \cdot ||B||} = \frac{\sum_{i=1}^n A_i \times B_i}{\sqrt{\sum_{i=1}^n (A_i)^2} \cdot \sqrt{\sum_{i=1}^n (B_i)^2}}$$

# Materiały dodatkowe

- różnej maści rozszerzenia, geo, rekomendacje czy modelowanie czasu, znajdziecie je na [GraphAware extensions](#)
- teoria grafów, czyli od zera do bohatera, jedno z najlepszych i bardziej dostępnych opracowań to [Graph Theory by Keijo Ruohonen](#)
- algorytmy rekomendacji z wykorzystaniem grafów, zostały opisane w [Studying Recommendation Algorithms by Graph Analysis](#)
- czytać, testować, sprawdzać co inni wyczynią z grafami, czyli [GraphGist Project](#)

# dla prawdziwych "no life"

- raptem 857 strony "Social Network Analysis: Methods and Applications" autorstwa Stanley Wasserman (nie przebrnałem)