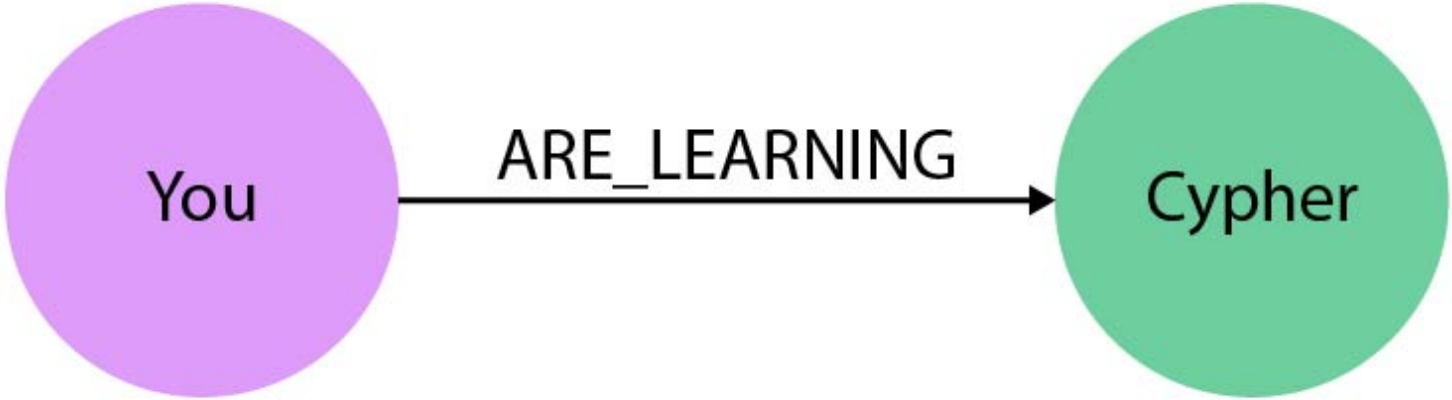
Bases de données spécialisées

Partie 2 : Bases de données orientées graphes
neo4j et Cypher



Cristina Sirangelo
IRIF, Université Paris Cité
amelie@irif.fr

Transparent de cours de Amélie Gheerbrant IRIF, Université Paris Cité amelie@irif.fr

Neo4j et Cypher

- Neo4j est le sgbd graphe le plus utilisé :

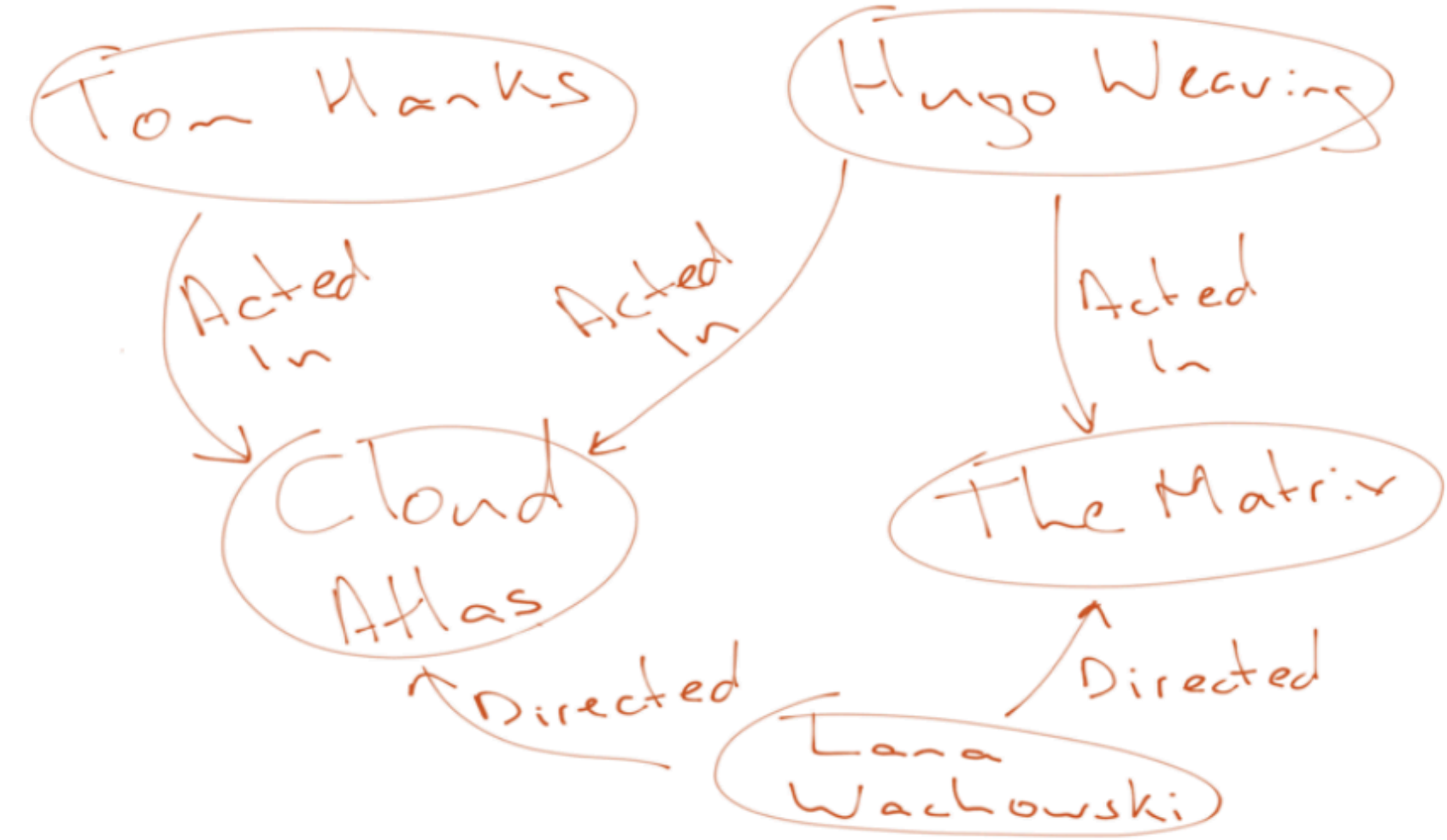
https://db-engines.com/en/ranking/graph+dbms

- Modèle : graphe de propriété (graphe dont les noeuds et les arrêtes comportent des paires nom-valeur de valeurs de données)
- Language de requête : Cypher
  - ASCII-art pattern matching + fonctionnalités BD usuelles

The #1 Database for Connected Data

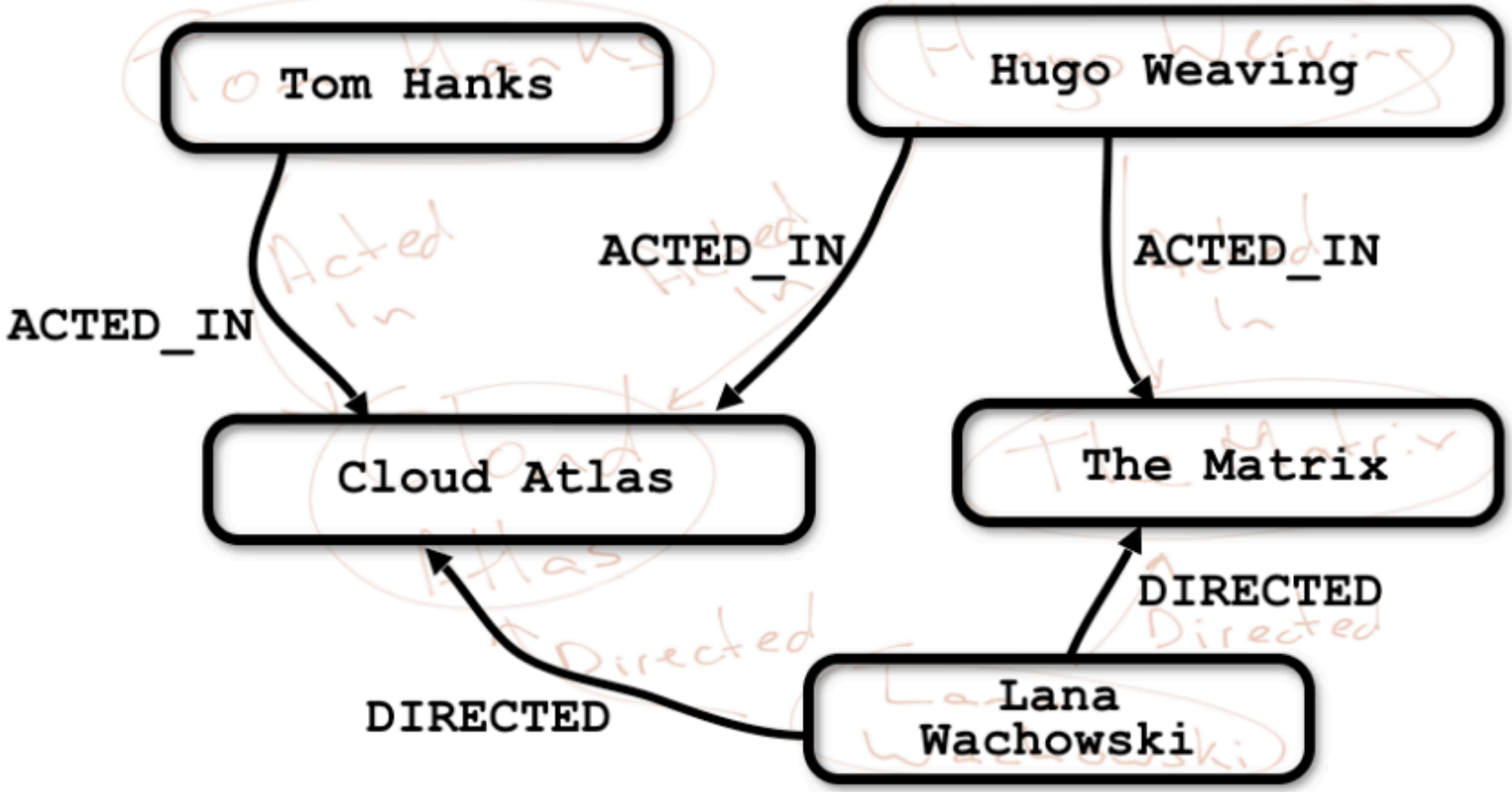
Whiteboard friendliness

Matrix - whiteboard model

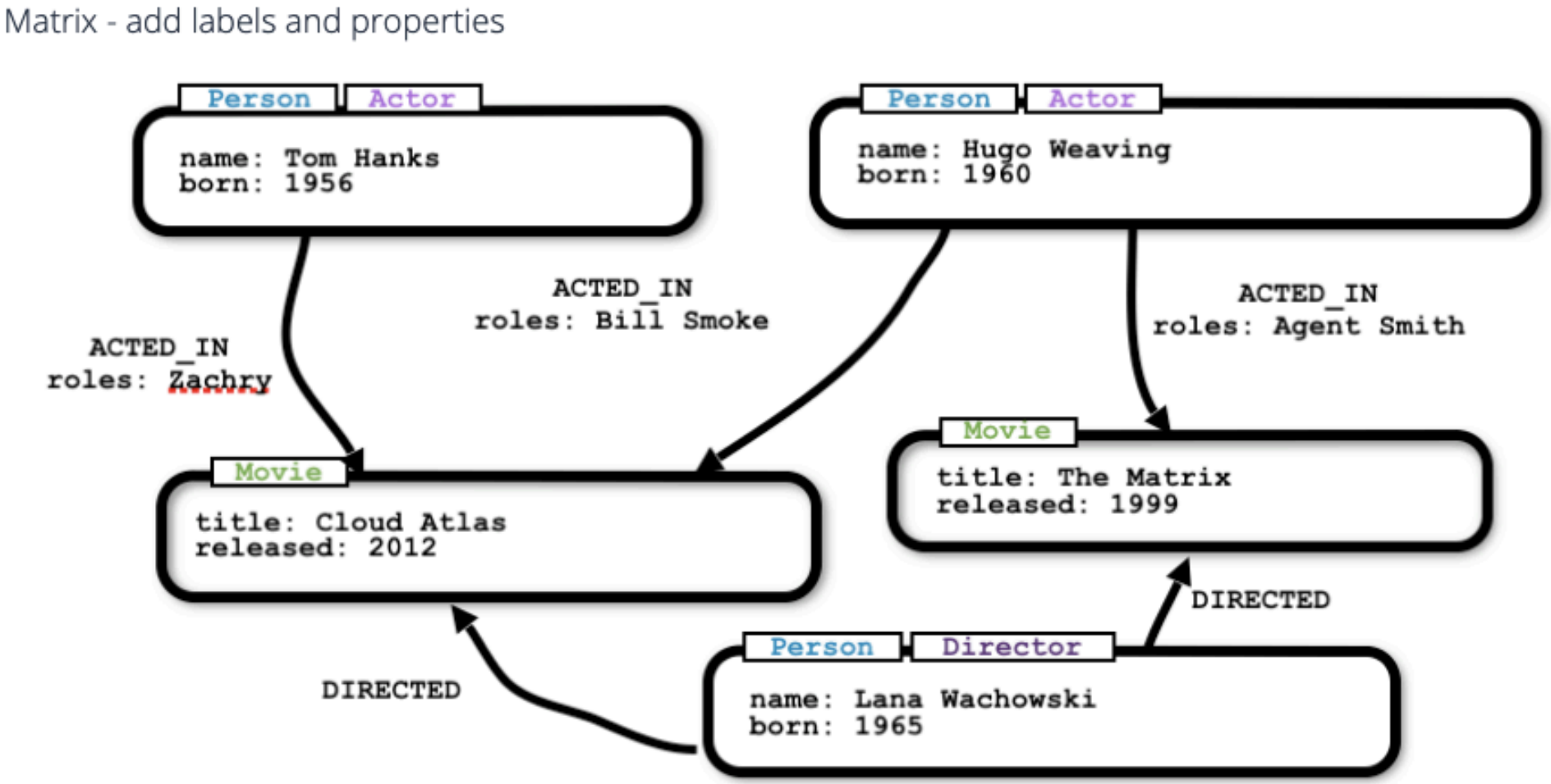


Whiteboard friendliness

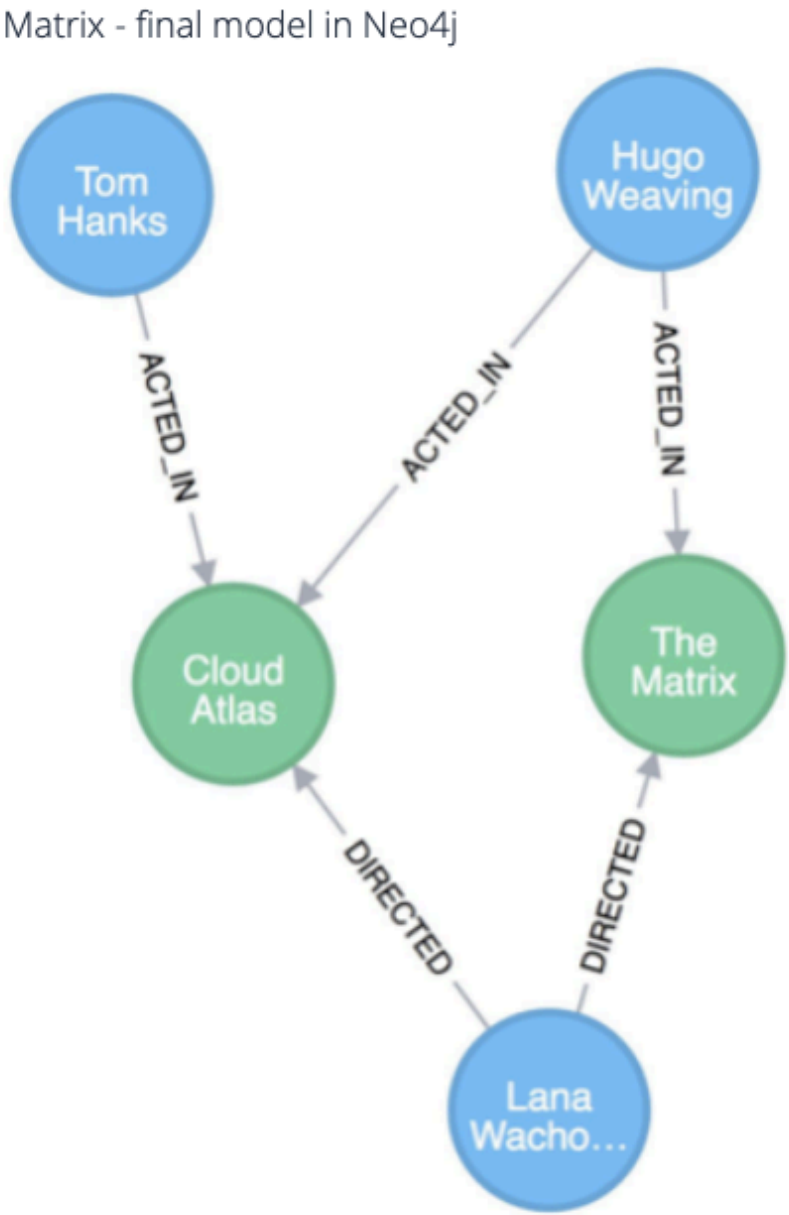
Matrix - match node and relationship format of property graph model



Whiteboard friendliness

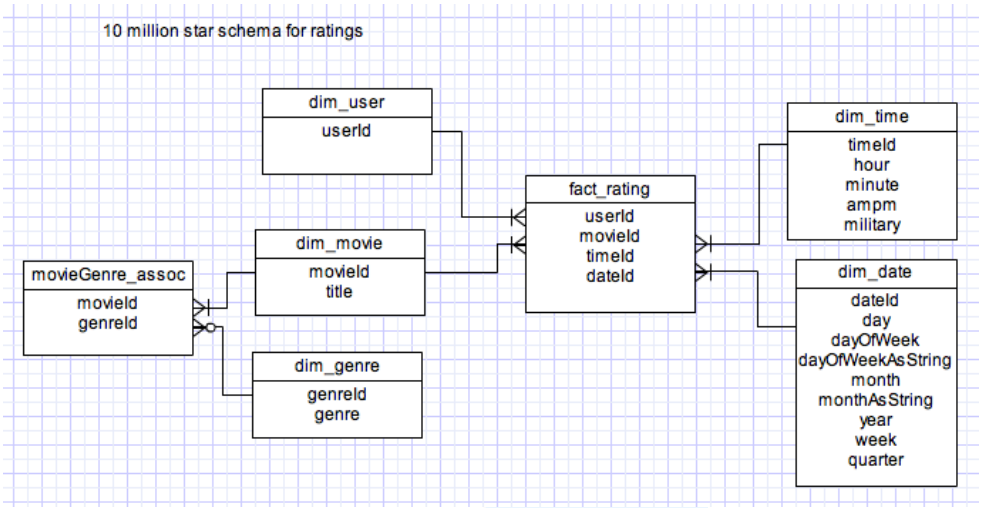


Whiteboard friendliness



L' étape pénible de génération de diagramme entité relation (cas relationnel) : on oublie !!!

oui, ça



Neo4j : les fondamentaux

- Les noeuds
- Les relations
- Les propriétés
- Les labels

The #1 Database for Connected Data

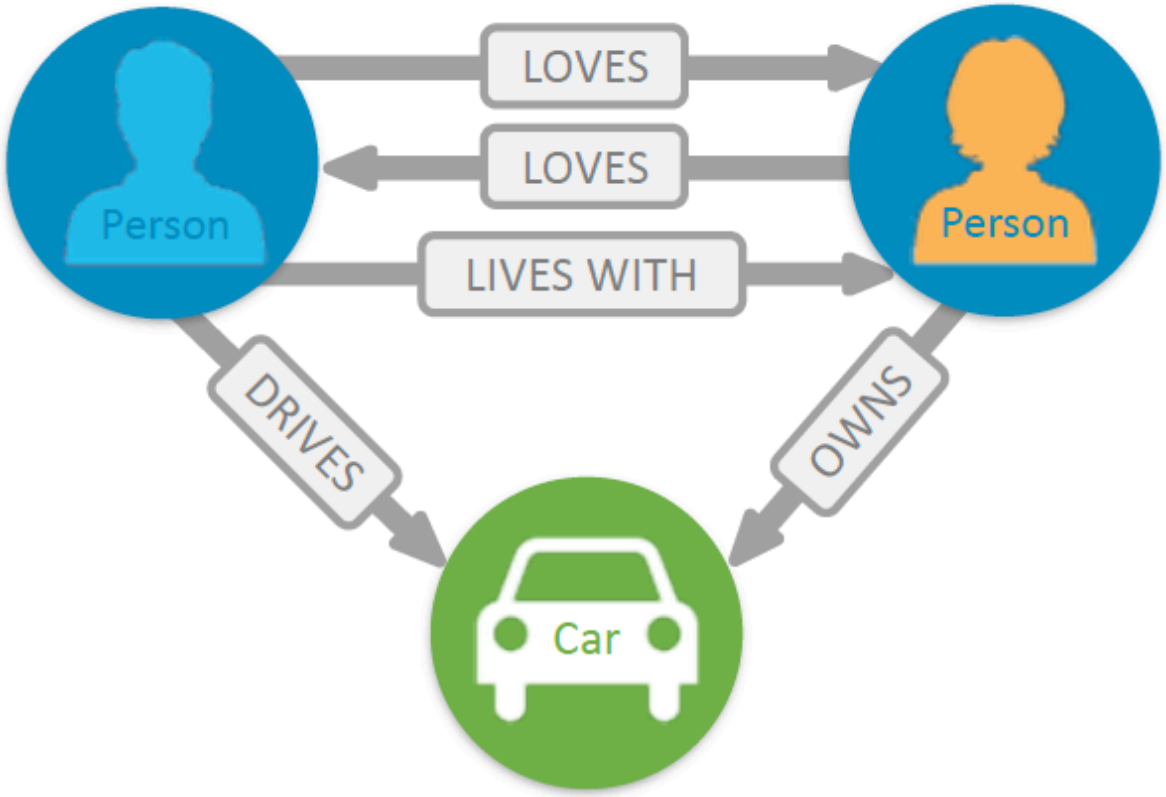
Les composants du modèle de graphe de propriété

- Les noeuds
  - représentent les objets dans le graphe (~tuples)
  - peuvent comporter un ou des labels / types / étiquettes

Les composants du modèle de graphe de propriété

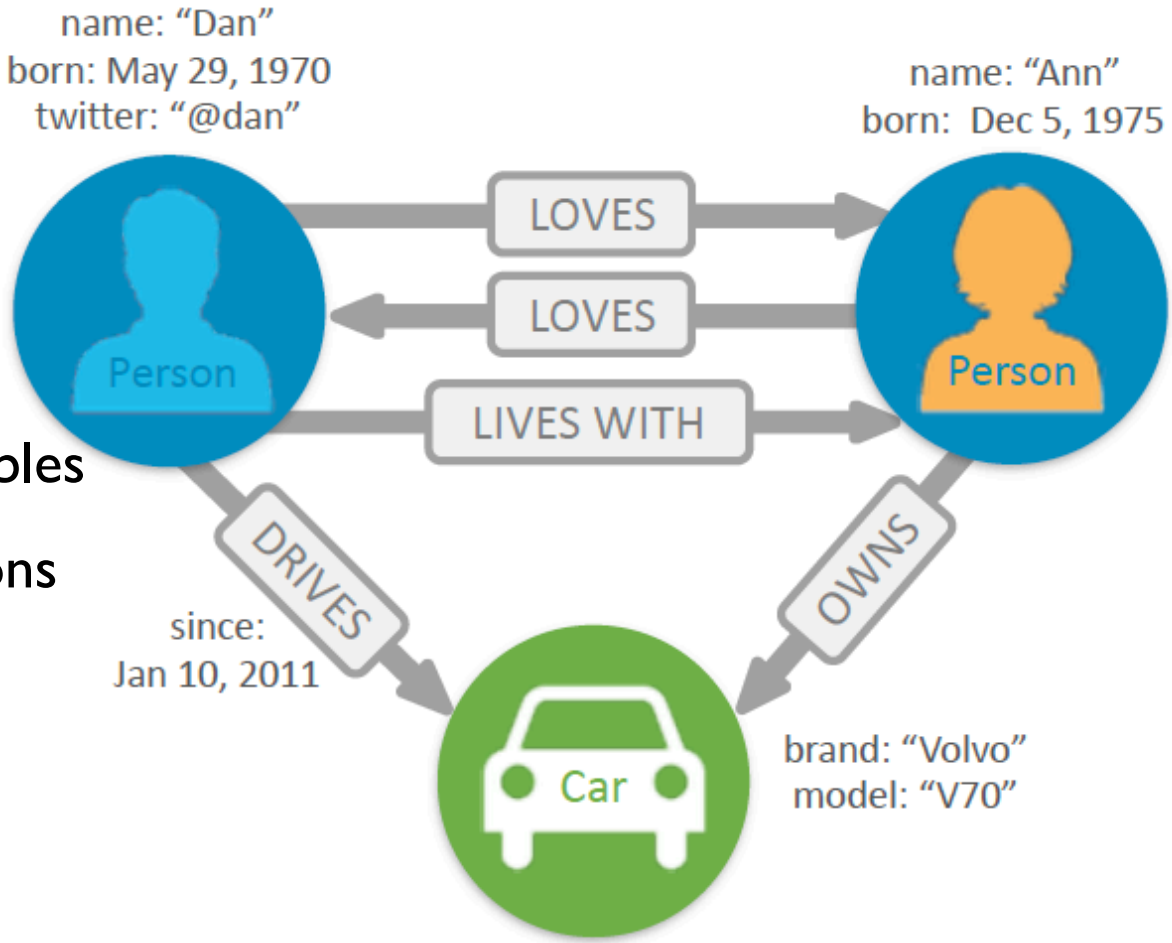
- Les noeuds
  - représentent les objets dans le graphe (~tuples)
  - peuvent comporter un ou des labels / types / étiquettes
- Les relations
  - relient les noeuds par

    type et direction



Les composants du modèle de graphe de propriété

- Les noeuds
  - représentent les objets dans le graphe (~tuples)
  - peuvent comporter un ou des labels / types / étiquettes
- Les relations
  - relient les noeuds par type et direction
- Les propriétés
  - paires nom-valeur possibles sur les noeuds et relations



Briques de base du graphe : résumé

- Les noeuds - entités et types de valeur complexes
- Les relations - connectent les entités et structure le domaine
- Les propriétés - attributs d'entité, qualités de relations, métadonnées
- Les labels - groupent les noeuds par role

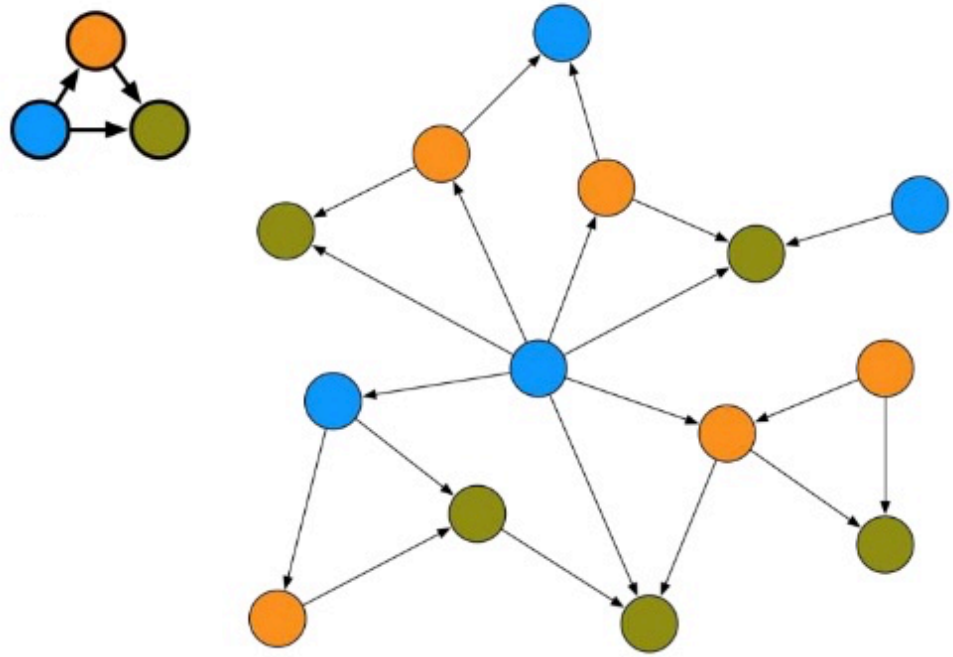
Langage de requête pour les graphes

Pourquoi pas SQL ?

- SQL inefficace pour exprimer des patterns de graphe complexes
- En particulier pour les requêtes
  - récursives
  - pouvant accepter des chemins de longueurs différentes
- Les patterns de graphes sont plus intuitifs et déclaratifs que les jointures
- SQL ne peut pas retourner de valeurs sur les chemins

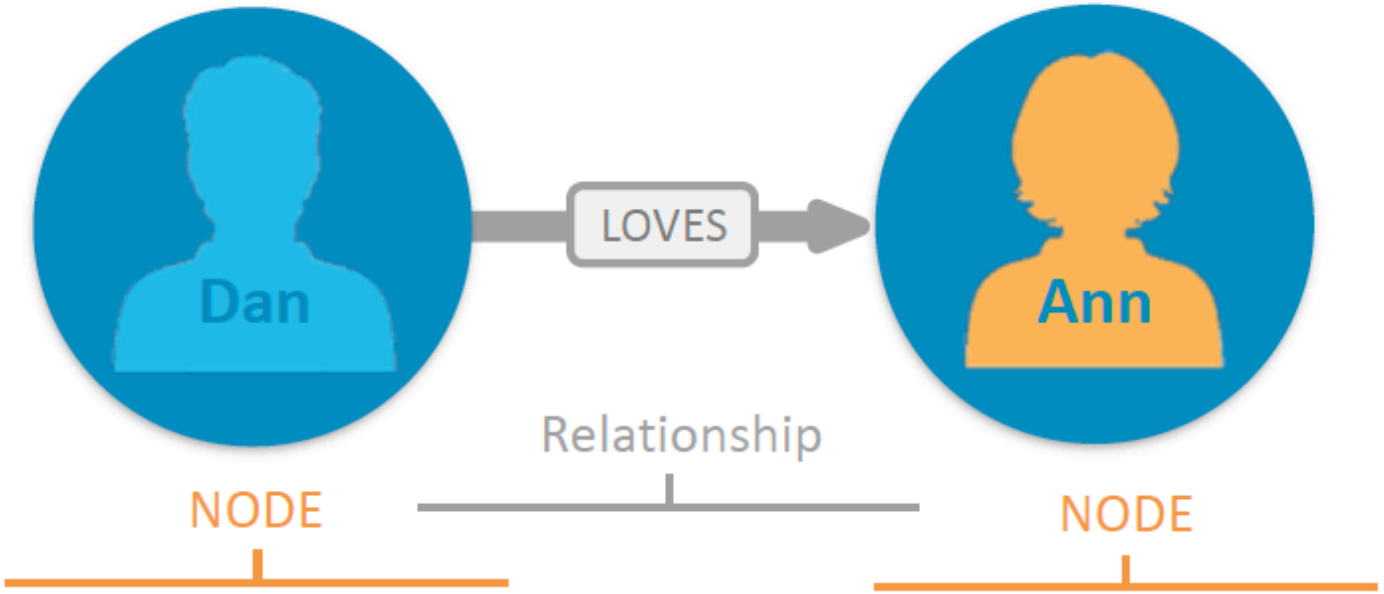
Cypher

- Un langage de requête fait pour les graphes basé sur les patterns de graphe
  - Déclaratif
  - Expressif
  - Pattern matching

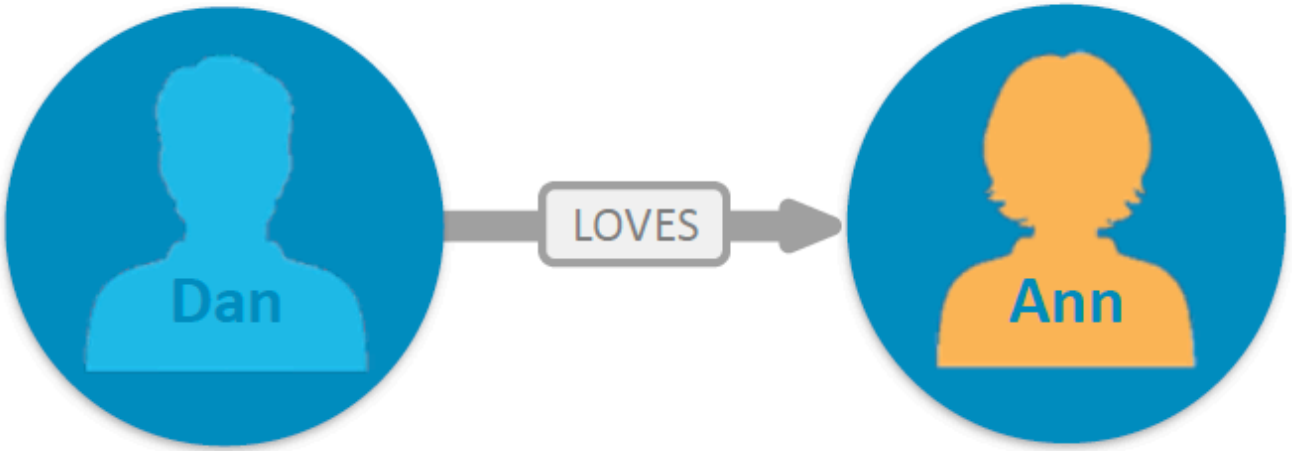


(Pattern matching = « *recherche de motif* » en français)

Les patterns dans notre modèle de graphe



Cypher : expression des patterns de graphe

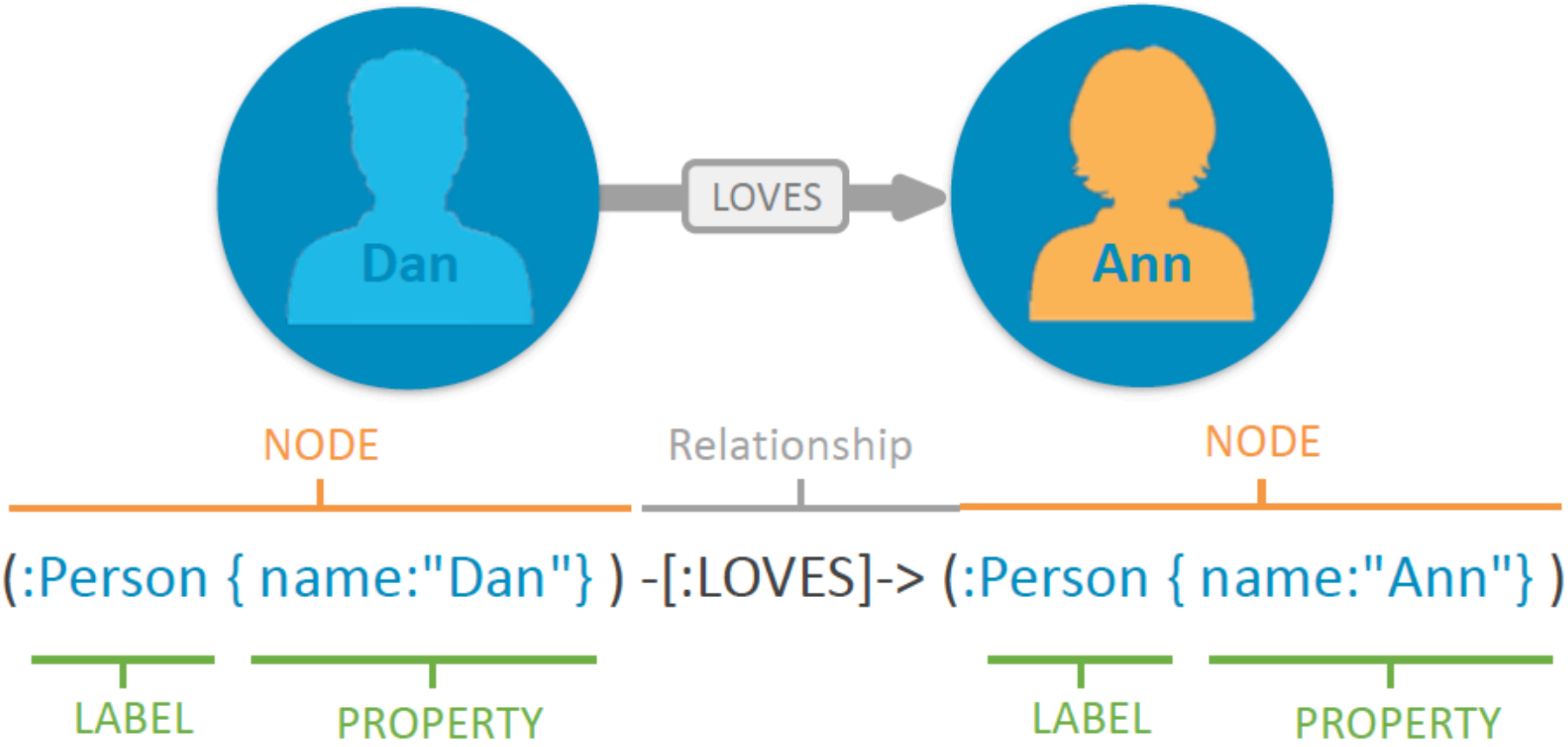


Relationship

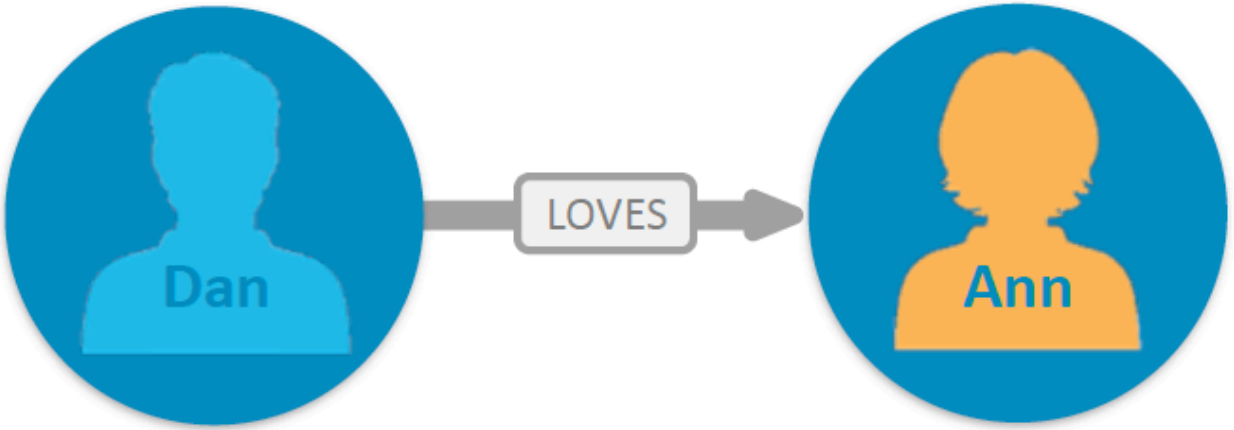
```
(:Person { name:"Dan"} ) -[:LOVES]-> (:Person { name:"Ann"} )
```

LABEL PROPERTY LABEL PROPERTY

Cypher : expression des patterns de graphe



Cypher : création des patterns de graphe

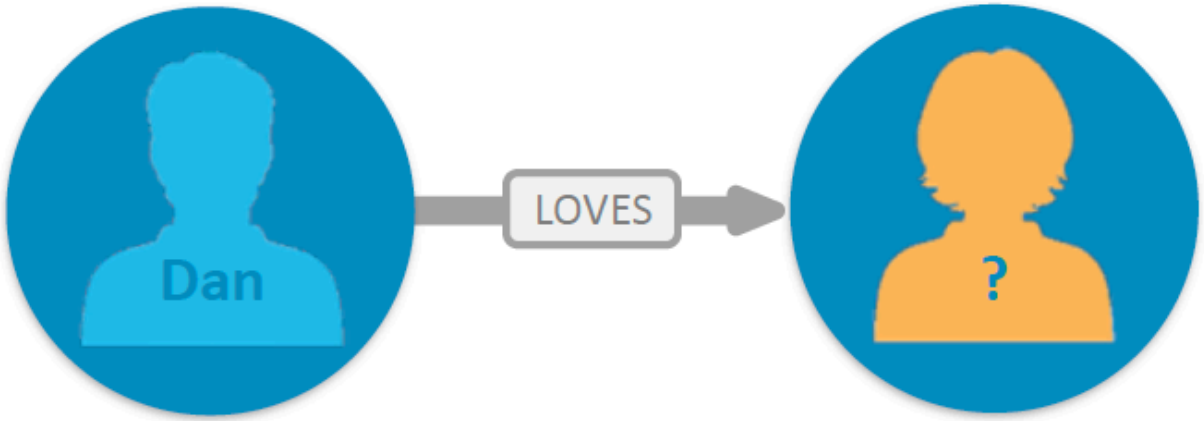


NODE Relationship NODE

```
CREATE (:Person { name:"Dan"} ) -[:LOVES]-> (:Person { name:"Ann"} )
```

LABEL PROPERTY LABEL PROPERTY

Cypher : match des patterns de graphe



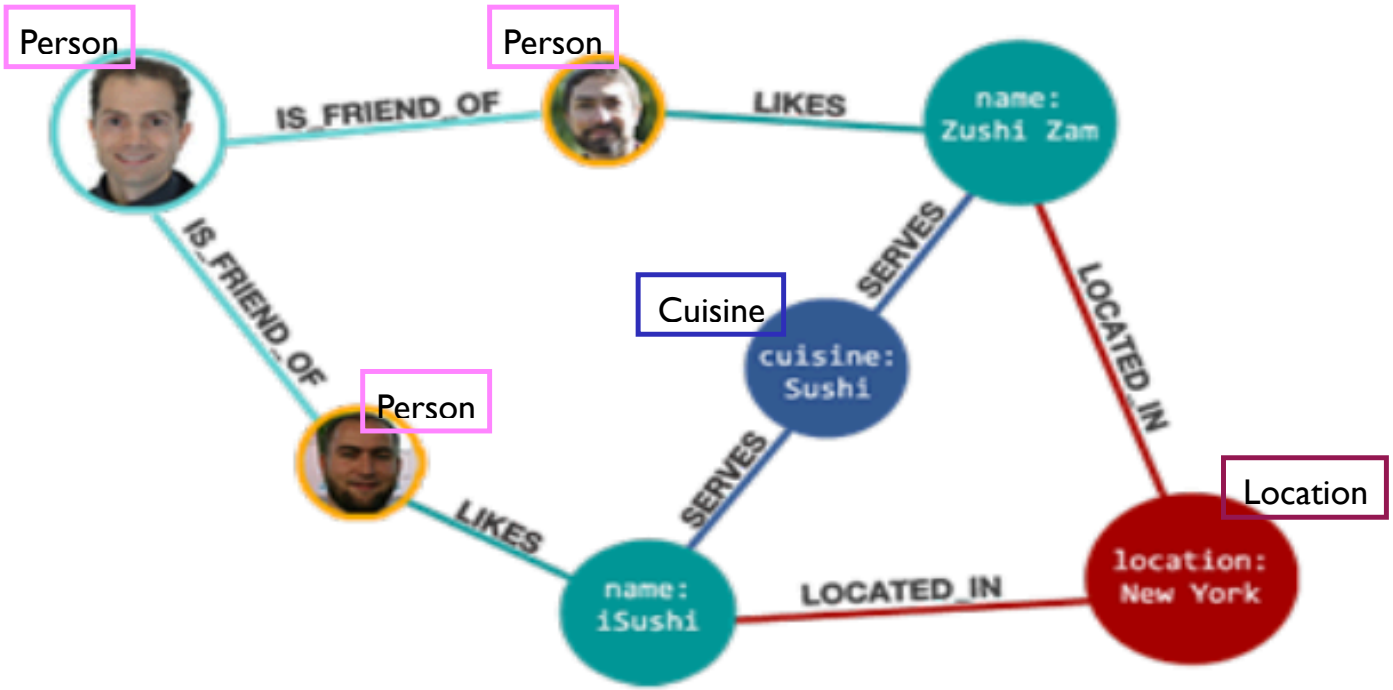
NODE — Relationship — NODE

```
MATCH (:Person { name:"Dan"} ) -[:LOVES]-> ( whom ) RETURN whom
```

LABEL — PROPERTY — VARIABLE

Un exemple de requête de graphe

Recommandation et réseaux sociaux



Recommandation et réseaux sociaux


```
MATCH (person:Person)-[:IS_FRIEND_OF]->(friend),
      (friend)-[:LIKES]->(restaurant),
      (restaurant)-[:LOCATED_IN]->(loc:Location),
      (restaurant)-[:SERVES]->(type:Cuisine)
WHERE person.name = 'Philip'
AND loc.location='New York'
AND type.cuisine='Sushi'
RETURN restaurant.name
```

La Syntaxe de Cypher

Les motifs de noeuds

- Les noeuds
  - dessinés avec des parenthèses

( )

Les motifs de noeuds

- Les noeuds
  - dessinés avec des parenthèses, on peut spécifier un type (label)

(:Type)

Les motifs de noeuds

- Les noeuds
  - dessinés avec des parenthèses, on peut spécifier un type
    - Expression de type autorisées :

(:(A&B)&!(B&C))

    - Opérateurs logiques pour les expressions de type :
      - & | !
    - Wildcard (au moins une etiquette) : %

Remarque : avant Neo4J 5 seulement “and” autorisé pour les labels de noeuds et avec une autre syntaxe (:A:B)

Les motifs de relation

- Les relations
  - dessinés avec des flèches, détails entre crochets

```
- - >

-[]->

- [:DIRECTED] ->
```

  - Matchent une seule relation du graphe. Entre crochets les propriétés pour filtrer les relations matchées

Les motifs de relation

- Les relations
  - dessinés avec des flèches, détails entre crochets
  - expressions de type et wildcard autorisés

$$- [:(!R\&!S)|T] ->$$

  - Les expressions de type ont la même syntaxe que l'expressions d'etiquette (pour le noeuds)
    - (mais à difference d'un noeud, une relation a exactement une etiquette)

Les motifs de chemin (simple path patterns)

- Simple path patterns
  - dessinés en connectant les noeuds et les relations avec des tirets,

    la spécification de directions avec > et < est optionnelle

```
( ) - - ( )
( ) - [ ] - ( )
( ) - [ ] - >( )
( ) <- [ ] - ( )
( ) <- [ ] -> ( )
```

Les motifs de chemin simples (simple path patterns)

- Un simple path pattern commence et termine avec un noeud et alterne strictement noeuds et relations
- Il doit avoir au mons un noeud

( ) - [ ] - >( ) - - ( )<- [] - ( ) <- [ ] - ( )

Les variables

- Un pattern cypher peut utiliser des variables pour nommer noeuds, relations et chemins

(n :A) - [r:Directed] - >(m)

- n, r et m sont les variables du patterns, elles precedent les éventuelles etiquettes
  - Elles servent à obtenir une reference aux noeuds et relations matchées
- :A est un label de noeud
- :Directed est un label de relation

Les propriétés

- Les propriétés de noeuds et relations sont décrites entre { }

(n :P {name :'John', office: 34}) - [r:Directed {since :1992}] - >(m)

- n, r et m sont les variables du pattern, elles precedent les éventuelles etiquettes
- :P est un label de noeud
- :Directed est un label de relation
- "name" et "office" sont des propriétés du noeud n, "since" une propriété de la relation r

Les motifs de graphe (graph patterns)

- Un motif de graphe est formé par plusieurs path patterns séparés par ‘,’
- Ils peuvent utiliser des variables en commun

(p : Person) - [r : ACTED_IN] -> (m : Movie)<- [: ACTED_IN] - (c : Person) ,

(m) <- [:DIRECTED]- (d : Person)

Intuitivement : ce motif matche toutes les parties du graphe décrivant un film, un couple d’acteurs de ce film ainsi que le réalisateur de ce film

Les composants d'une requête Cypher

MATCH (m : Movie)

RETURN m

MATCH est suivi d'un graph pattern (le motif à chercher dans le graphe)

RETURN est suivi des parties du pattern (noeuds, relations, chemins et/ou propriétés) à retourner

MATCH et **RETURN** sont des mot clefs Cypher

Les composants d'une requête Cypher

Exemples :

MATCH (p : Person) - [r : ACTED_IN] -> (m : Movie)

RETURN p, r, m.title

MATCH (p : Person) - [r1 : ACTED_IN] -> (m : Movie) ,
(m) <- [r2 : ACTED_IN] - (c : Person)

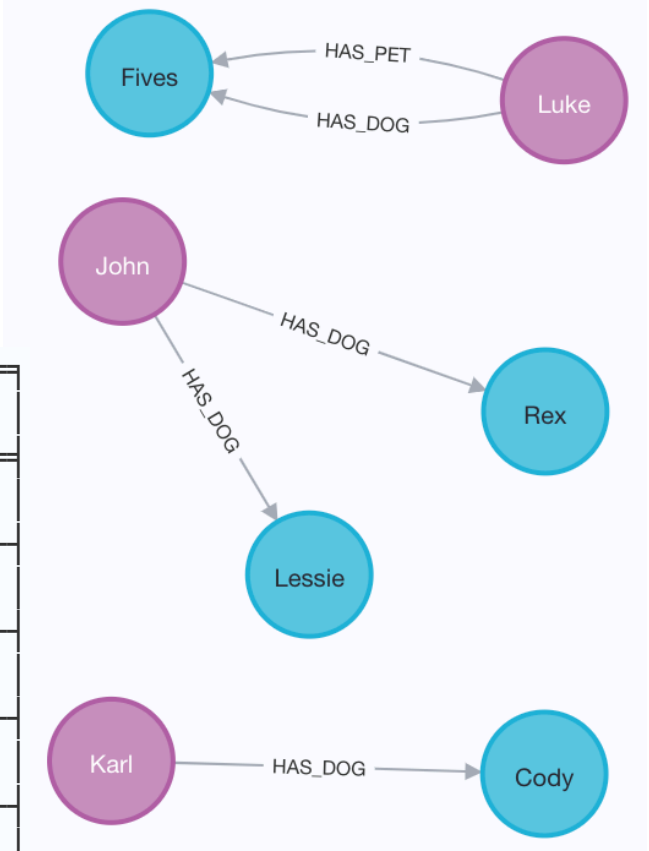
RETURN p, r1, c

MATCH
(p : Person) - [r1 : ACTED_IN] -> (m : Movie) <- [r2 : ACTED_IN] - (c : Person), (m) <- [:DIRECTED]- (d : Person)

RETURN p.name, c.name, r1.fee, d

Matchs

- Un match d'un graph pattern P (dont les path patterns sont simples) sur un graphe G est une affectation v des variables de P tel que v(P) est dans G. De plus v(r1)≠v(r2) pout toute paire r1≠r2 de relations
- MATCH P calcule tous les matchs de P dans G



MATCH (person :Person)-[r]->(d:Dog)

d	person	r
(:Dog {name: "Lessie"})	(:Person {name: "John"})	[:HAS_DOG]
(:Dog {name: "Rex"})	(:Person {name: "John"})	[:HAS_DOG]
(:Dog {name: "Cody"})	(:Person {name: "Karl"})	[:HAS_DOG]
(:Dog {name: "Fives"})	(:Person {name: "Luke"})	[:HAS_DOG]
(:Dog {name: "Fives"})	(:Person {name: "Luke"})	[:HAS_PET]

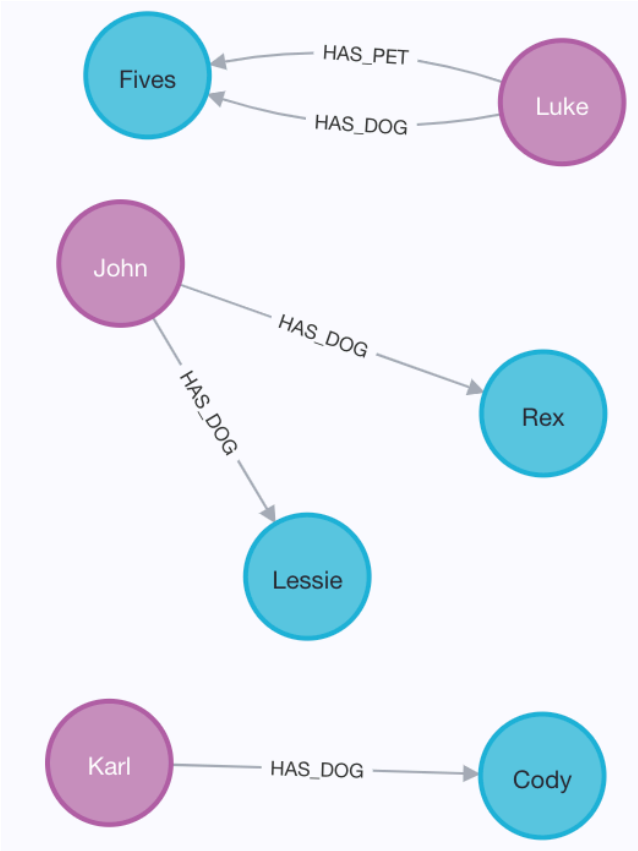
Matchs : variables anonymes

- Les variables sans nom se comportent comme les variables avec nom (toutes distinctes)

MATCH (person :Person)-[]->(:Dog)

RETURN *

person
(:Person {name: "John"})
(:Person {name: "John"})
(:Person {name: "Karl"})
(:Person {name: "Luke"})
(:Person {name: "Luke"})



Remarques : doublons pas éliminés dans les résultats retournés

* retourne toutes les variables

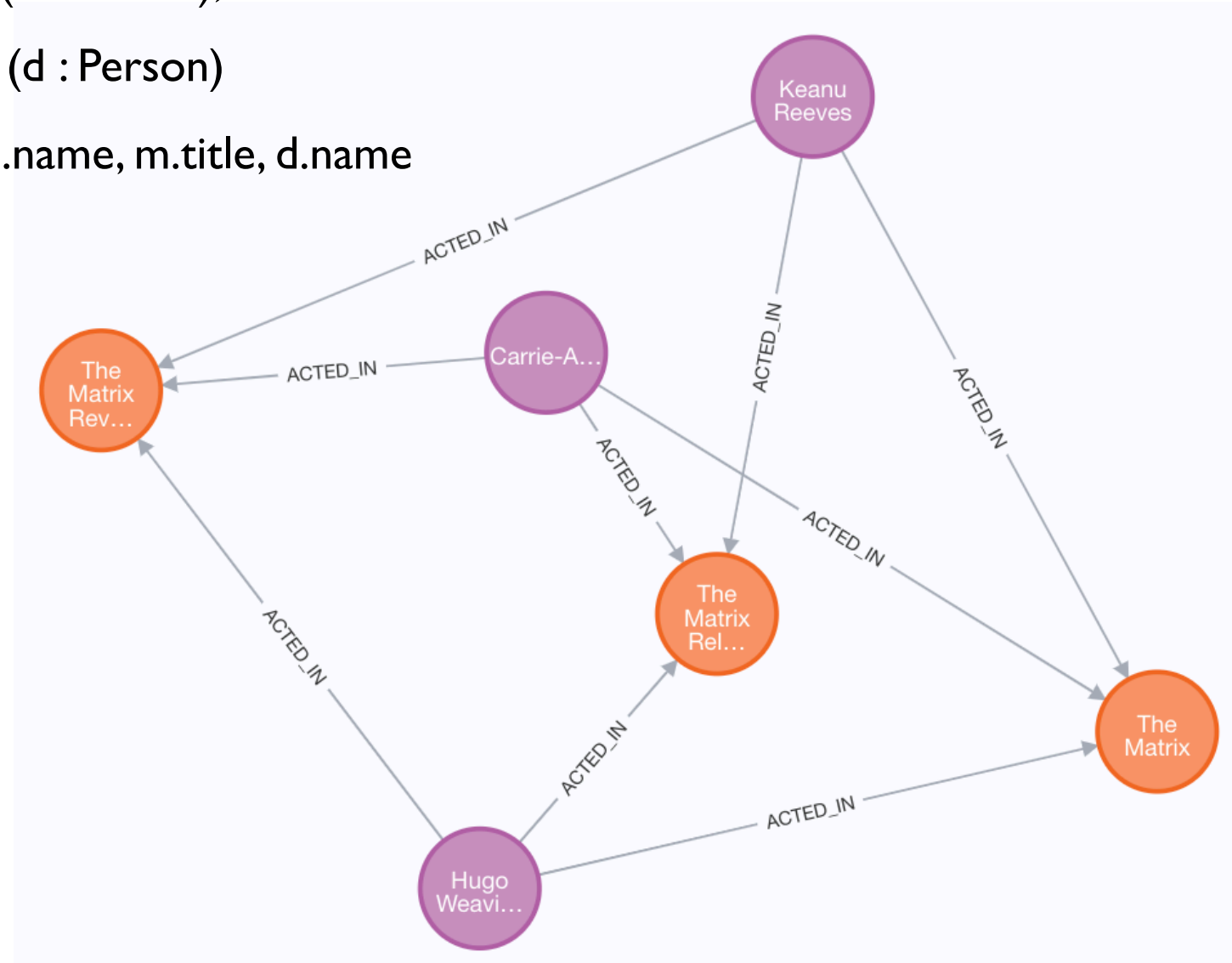
Matchs : sémantique “Trail”

MATCH (p : Person) - [r1 :ACTED_IN] -> (m : Movie)

<- [r2 :ACTED_IN] - (c : Person),

(m) <- [:ACTED_IN]- (d : Person)

RETURN p.name, c.name, m.title, d.name



Matchs : sémantique “Trail”

MATCH (p : Person) - [r1 :ACTED_IN] -> (m : Movie)

<- [r2 :ACTED_IN] - (c : Person),

(m) <- [:ACTED_IN]- (d : Person)

RETURN p.name, c.name, m.title, d.name

p.name	c.name	m.title	d.name
"Keanu Reeves"	"Carrie-Anne Moss"	"The Matrix"	"Hugo Weaving"
"Carrie-Anne Moss"	"Keanu Reeves"	"The Matrix"	"Hugo Weaving"
"Keanu Reeves"	"Hugo Weaving"	"The Matrix"	"Carrie-Anne Moss"
"Hugo Weaving"	"Keanu Reeves"	"The Matrix"	"Carrie-Anne Moss"
"Carrie-Anne Moss"	"Hugo Weaving"	"The Matrix"	"Keanu Reeves"
"Hugo Weaving"	"Carrie-Anne Moss"	"The Matrix"	"Keanu Reeves"

Matchs : sémantique “Trail”

Remarquer qu’on n’obtient pas la ligne

"Keanu Reeves"	"Carrie-Anne Moss"	"The Matrix”	"Keanu Reeves"

Ni la ligne

"Keanu Reeves"	"Keanu Reeves”	"The Matrix”	"Hugo Weaving"

Car ces matchs nécessiteraient d’utiliser plusieurs fois la même relation (pour matcher différentes parties du graph pattern)

Matchs : sémantique “Trail”

Donc par défaut dans Cypher les matchs où la même relation est incluse plusieurs fois ne sont pas retenus (*relationships isomorphism*)

Raisons :

- réduction de la taille des résultats
- élimination des traversées infinies

D’autres sémantiques existent. (c.f., manuel Cypher p8)

Les composants d'une requête Cypher

MATCH path = (p : Person) - [: HAS_DOG] - > ()

RETURN path

path est une variable de chemin

path
(:Person {name: "John"})-[:HAS_DOG]->(:Dog {name: "Lessie"})
(:Person {name: "John"})-[:HAS_DOG]->(:Dog {name: "Rex"})
(:Person {name: "Karl"})-[:HAS_DOG]->(:Dog {name: "Cody"})
(:Person {name: "Luke"})-[:HAS_DOG]->(:Dog {name: "Fives"})

Valeur retournés

- On peut retourner des noeuds

```
MATCH (m : Movie)
RETURN m
```

- Des relations

```
MATCH  (person :Person )-[r:HAS_DOG]->(d:Dog)
RETURN r
```

- Des propriétés

```
MATCH (m : Movie)
RETURN m.title, m.released
```

On accède aux propriétés avec {variable}.{property_key}

Valeur retournés

- Des chemins

```
MATCH path = (p : Person) - [] - > (d : Dog)
RETURN path
```

- Toutes les variables (de noeuds, relation et chemin) de la requête

```
MATCH path = (p : Person) - [] - > ()
RETURN *
```

Valeur retournés

```
MATCH path = (p : Person) - [] - > ()
RETURN *
```

p	path
(:Person {name: "John"})	(:Person {name: "John"})-[:HAS_DOG]->(:Dog {name: "Lessie"})
(:Person {name: "John"})	(:Person {name: "John"})-[:HAS_DOG]->(:Dog {name: "Rex"})
(:Person {name: "Karl"})	(:Person {name: "Karl"})-[:HAS_DOG]->(:Dog {name: "Cody"})
(:Person {name: "Luke"})	(:Person {name: "Luke"})-[:HAS_DOG]->(:Dog {name: "Fives"})
(:Person {name: "Luke"})	(:Person {name: "Luke"})-[:HAS_PET]->(:Dog {name: "Fives"})

Valeur retournés

- Des fonctions peuvent être appliquées aux valeurs retournés (cf. https://neo4j.com/docs/cypher-manual/current/functions/)

```
MATCH path = (p : Person) - [r] - > ()
RETURN length(path), type(r)
```

- Plus en general n'importe quelle expression peut être retournée
  - Expression : constantes, variables, propriétés, appel de fonction, expression arithmétique, motifs de chemin, etc…

```
MATCH (m:Movie)
RETURN "Early release : ", m.released < 2012
```

  - cf. https://neo4j.com/docs/cypher-manual/current/queries/expressions/

Sensibilité à la casse

Sensible à la casse

- les labels de noeud
- Les types de relation
- Les clefs de propriété

Insensible à la casse

- les mots clefs Cypher

Sensibilité à la casse

Sensible à la casse

- : Person
- :ACTED_IN
- name

Insensible à la casse

- MaTcH
- return

L’écriture de requêtes

La sous-clause WHERE

```
MATCH (m : Person)
WHERE m.age > 60 XOR m.country = 'France'
RETURN m.name
```

Fait partie de MATCH, (ainsi que de OPTIONAL MATCH et WITH, cf. plus loin) :

- ajoute des contraintes au pattern
- seulement les matchs qui satisfont le WHERE sont retenus

Connecteurs logiques : NOT, OR, XOR, AND

Tests booléens : =, >,<, <=, >=, expressions régulières (=~ 'regexp'), fonctions booléennes (e.g. STARTS, CONTAINS, ENDS WITH,…), prédicat de label ("x : Label"), IS (NOT) NULL, prédicat de chemin, etc…

La sous-clause WHERE

Fonctions booléennes (sur les chaînes de caractères)

```
MATCH (m : Person)
WHERE m.name STARTS WITH 'A'
RETURN m.name
```

La sous clause WHERE

Fonctions booléennes (sur les chaînes de caractères)

```
MATCH (m : Person)
WHERE m.name ENDS WITH 'li'
RETURN m.name
```

La sous clause WHERE

Fonctions booléennes (sur les chaînes de caractères)

```
MATCH (m : Person)
WHERE m.name CONTAINS ‘li’
RETURN m.name as nom
```

La sous clause WHERE

Expressions régulières

```
MATCH (m : Person)
WHERE m.name =~ 'Am.*'
RETURN m.name
```

Hérité de la syntaxe des expressions régulières Java

Flags inclus, par exemple (?i) insensibilité à la casse

```
MATCH (m : Person)
WHERE m.name =~ '(?i)Am.*'
RETURN m.name
```

La sous-clause WHERE

prédicat de label (“:”) :

```
MATCH (m)
WHERE m : Person
RETURN m.name
```

```
MATCH (m)
WHERE m : Employee  : Student
RETURN m.name
```

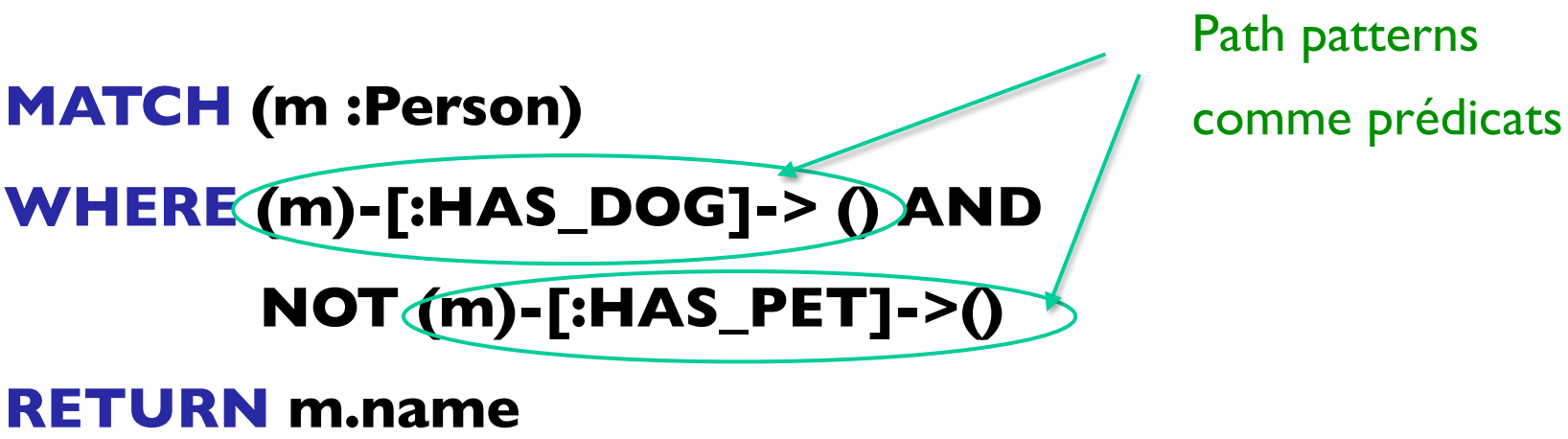
La sous-clause WHERE

- IS (NOT) NULL vérifie si une propriété existe (ou pas)

```
MATCH (m :Client)
WHERE m.email IS NOT NULL
RETURN m.email
```

La sous-clause WHERE

- **Prédicat de chemin** : un path pattern dans la clause WHERE est considéré un prédicat booléen :



- Il ne peut pas introduire de nouvelles variables : (m)-[:HAS_DOG]-> (x)
- Si p est un path pattern qui utilise les variables $m_1 \ldots m_k$
  p retourne vrai pour $m_1 = v_1, \ldots, m_k = v_k$ si $m_1 = v_1 \ldots m_k = v_k$ est un match pour p dans le graphe

La sous-clause WHERE

- WHERE peut aussi être utilisé dans un élément (noeud ou relation) d'un pattern :

```
MATCH (a:Person)-[r:HAS_DOG  WHERE r.since < 2009]->(b:Dog)
RETURN r.since
```

Dans ce cas la condition de WHERE peut seulement mentionner les variables du noud/relation dans lequel il se trouve

Les sous clauses SKIP et LIMIT

```
MATCH (m : Person)
WHERE m.name CONTAINS 'li'
RETURN m.name as nom
LIMIT 5
```

Sous-clauses de RETURN (ou de WITH, cf. plus loin)

5 premiers résultats seulement

Limite le branchement d'un chemin de recherche

Les sous clauses SKIP et LIMIT

```
MATCH (m : Person)
WHERE m.name CONTAINS 'li'
RETURN m.name as nom
SKIP 5
```

A partir du 6ème résultat

Utile en combinaison avec LIMIT pour l'affichage
de résultats page par page

Les sous clauses SKIP et LIMIT

```
MATCH (m : Person)
WHERE m.name CONTAINS 'li'
RETURN m.name as nom
SKIP 5
LIMIT 2
```

Le 6ème et 7ème résultat

La sous clause ORDER BY

```
MATCH (m : Person)
WHERE m.name CONTAINS 'li'
RETURN m.name as nom
ORDER BY m.name DESC
```

Suit RETURN (ou WITH), trie les résultats avant de les retourner

Modificateur par défaut : ASC (ordre croissant)

DESC : ordre décroissant

Souvent utile en combinaison avec LIMIT

La clause CREATE

```
CREATE (m : Movie {title : 'Mystic River', released : 2003})
RETURN m
```

- Ajoute au graphe les noeuds et relations spécifiées
- Les variables servent à garder une référence aux elements créés, pour les utiliser plus loin
- Meme syntaxe que la clause MATCH mais pas de relations / path pattern quantifiés, ni de relations de longueur variable (cf plus loin)

La clause CREATE

```
MATCH (m : Movie {title : 'Mystic River'}),
      (p : Person {name : 'Kevin Bacon'})
CREATE (p) - [r : ACTED_IN {roles: ['Sean']}] -> (m)
RETURN p, r, m
```

La clause DELETE

Attention ! A n'utiliser que si le noeud n'intervient dans aucune relation.

```
MATCH (p : Person {name : 'Your Name'})
DELETE p
```

Pour supprimer également toutes les relations associées :

```
MATCH (p : Person {name : 'Your Name'})
DETACH DELETE p
```

La clause DELETE

Pour supprimer une relation (mais pas les noeuds y participant) :

```
MATCH (p {name : 'Andy'}) - [r: KNOWS]-> ()
DELETE r
```

Pour supprimer tout le contenu de la base de données :

```
MATCH (p)
DETACH DELETE p
```

La clause SET

MATCH (m : Movie {title : 'Mystic River'})

SET m.tagline = 'We bury our sins here, Dave. We wash them clean.'

RETURN m

- Modifie chaque match comme spécifié
- Peut modifier
  - sur les noeud : etiquettes et propriétés
  - sur les relations : propriétés

La clause SET

- Ajouter des etiquettes de noeud :

```
match (n:Person {name:"John"} )
SET n:User
return labels(n)
```

labels (n)
["Person", "User"]

- Ajouter des propriétés :

```
match (n:Person {name:"John"} )
SET n += {age: 38}
return n
```

n
(:Person:User {name: "John",age: 38})

- Supprimer des propriétés

```
match (n:Person {name:"John"} )
SET n.age = null
return n
```

n
(:Person:User {name: "John"})

La clause SET

- Remplacer toutes les des propriétés :

```
match (n:Person {name:”John"} )
SET n = {firstName: ”John"}
return n
```

n
(:Person:User {firstName: "John"})

La clause REMOVE

```
MATCH (p : Person {name : 'Your Name'})
REMOVE p.age
RETURN p.name, p.age
```

Pour supprimer seulement la propriété âge,

la valeur retournée pour p.age sera alors nulle

La clause REMOVE

Attention, REMOVE ne peut pas être utilisé pour supprimer toutes les propriétés, à la place :

```
MATCH (p : Person {name : 'Your Name'})
SET p={}
RETURN p.name, p.age
```

La clause REMOVE

```
MATCH (p : Person {name : 'Your Name'})
REMOVE p:German:Swedish
RETURN p.name, labels(p)
```

Pour supprimer des labels sur un noeud.

L'ensemble de labels retourné sera vide [ ]