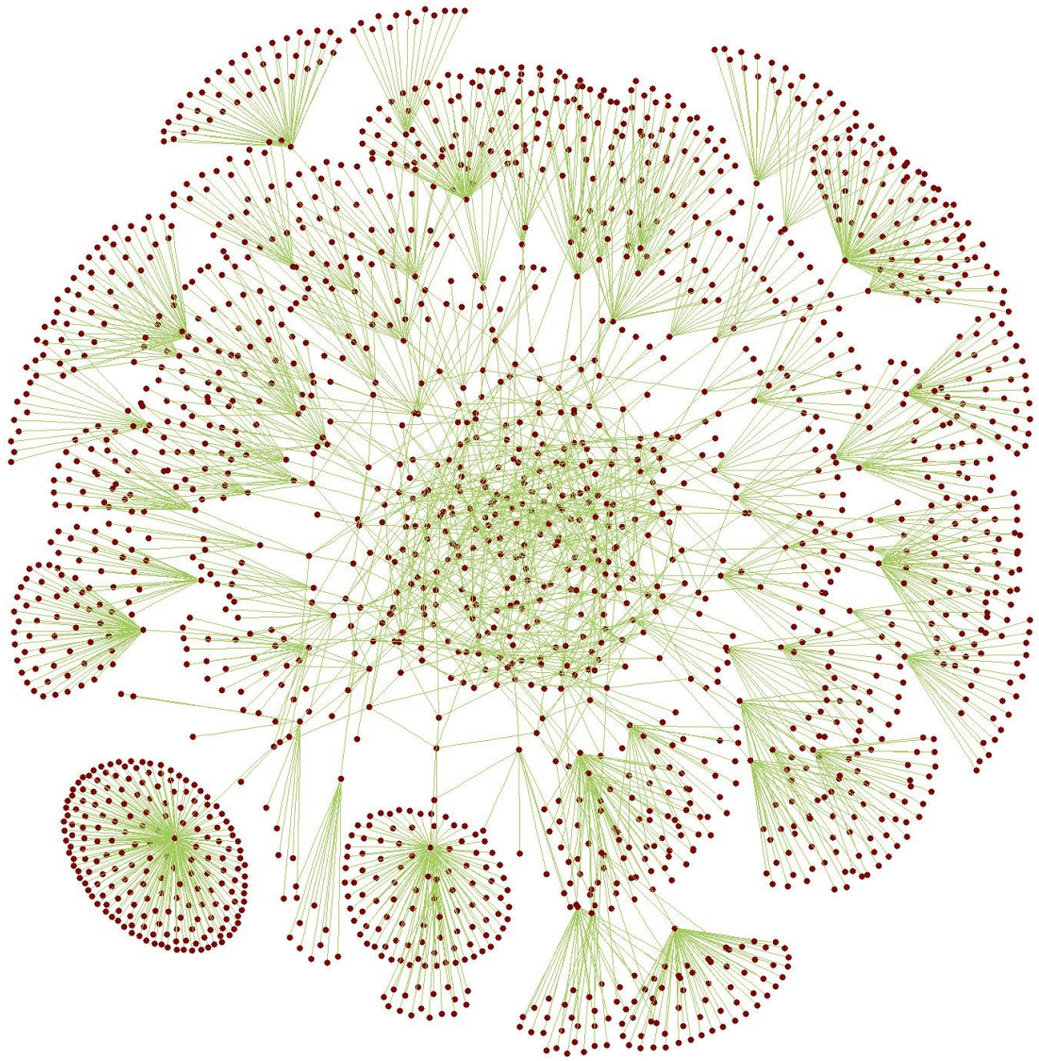
Bases de données orientées graphe

Introduction

- Concept de connexion omniprésent
  - Réseaux sociaux, internet, biologie, transports…

- Modèle d'analyse des bases de données NoSQL basé sur les agrégats

- Stocker des entités connectées est un challenge non adressé ni par les SGBD relationnels, ni par les bases NoSQL



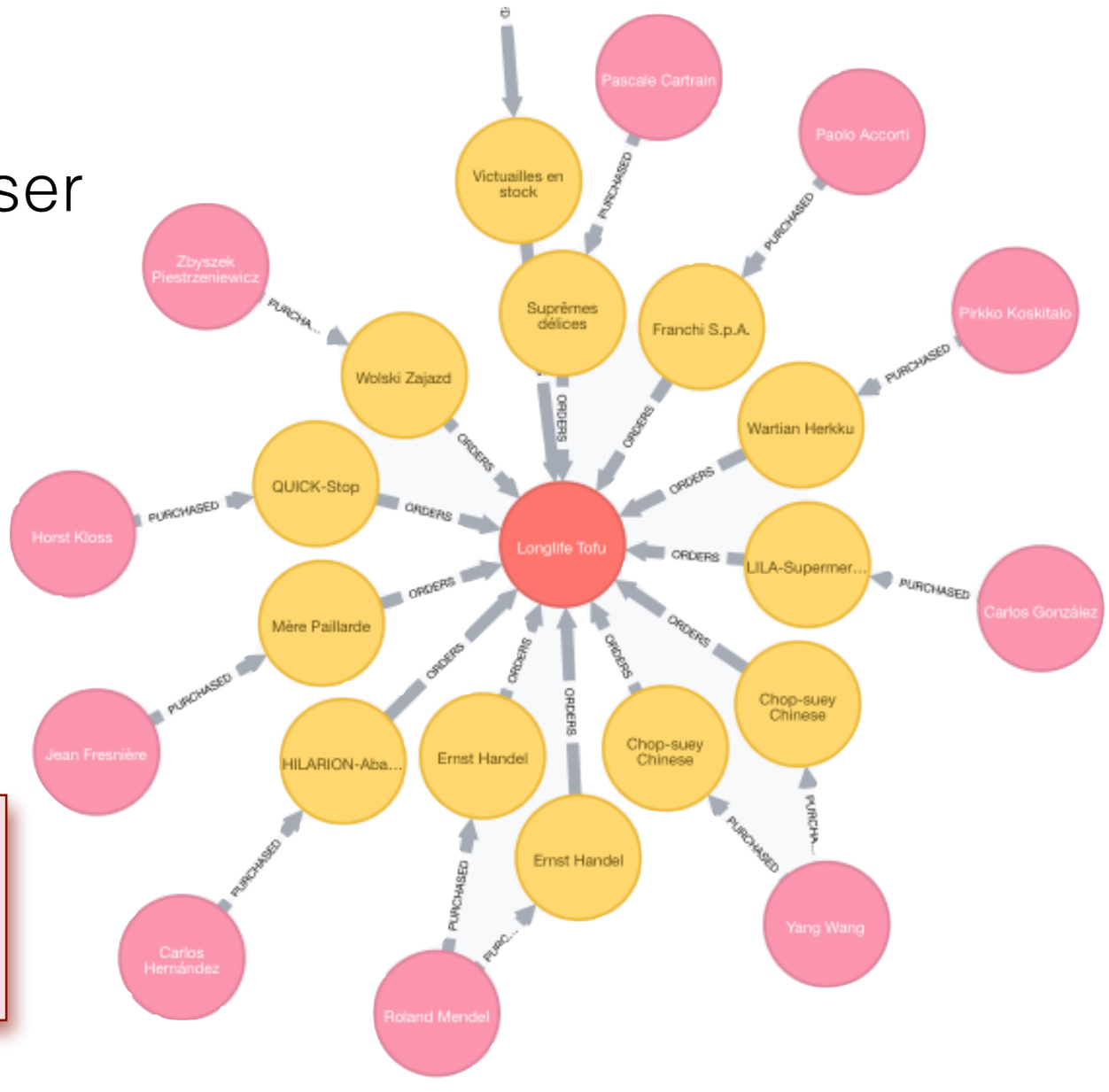
Source : http://www.math.cornell.edu/~mec/

Introduction

Modeliser les relations

Relations difficiles à modéliser par :

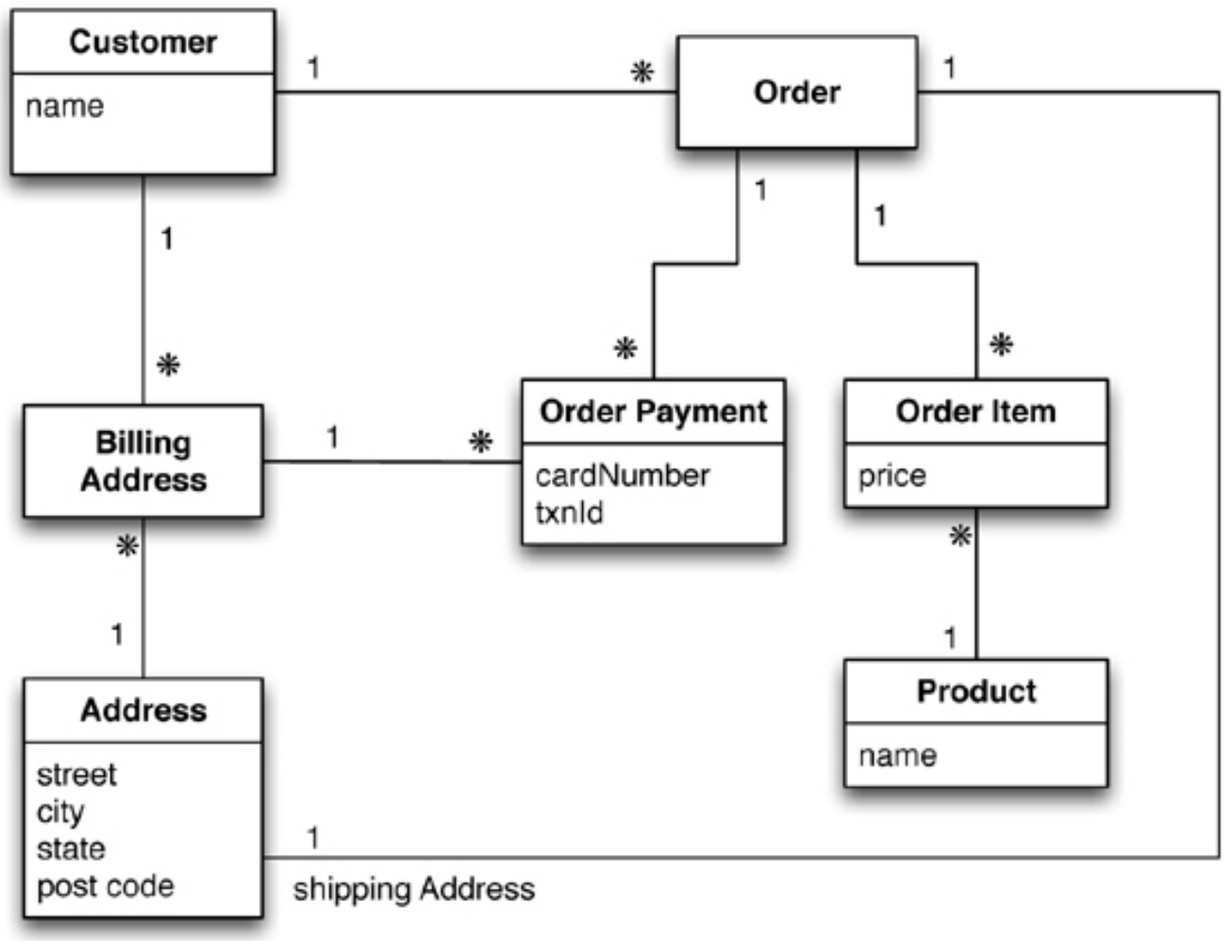
- Les BD relationnelles
- Les BD NoSQL



Les graphes permettent une modélisation naturelle des relations entre entités

Introduction

De la difficulté des BDR pour modéliser les relations



Quels produits ont été achetés par un utilisateur ?
Requête coûteuse en jointures…

Introduction

De la difficulté des BD NoSQL pour modéliser les relations

- Stockage indépendant des documents/valeurs/colonnes
- Ajout de relations par l'imbrication
- Requiert des jointures au niveau application

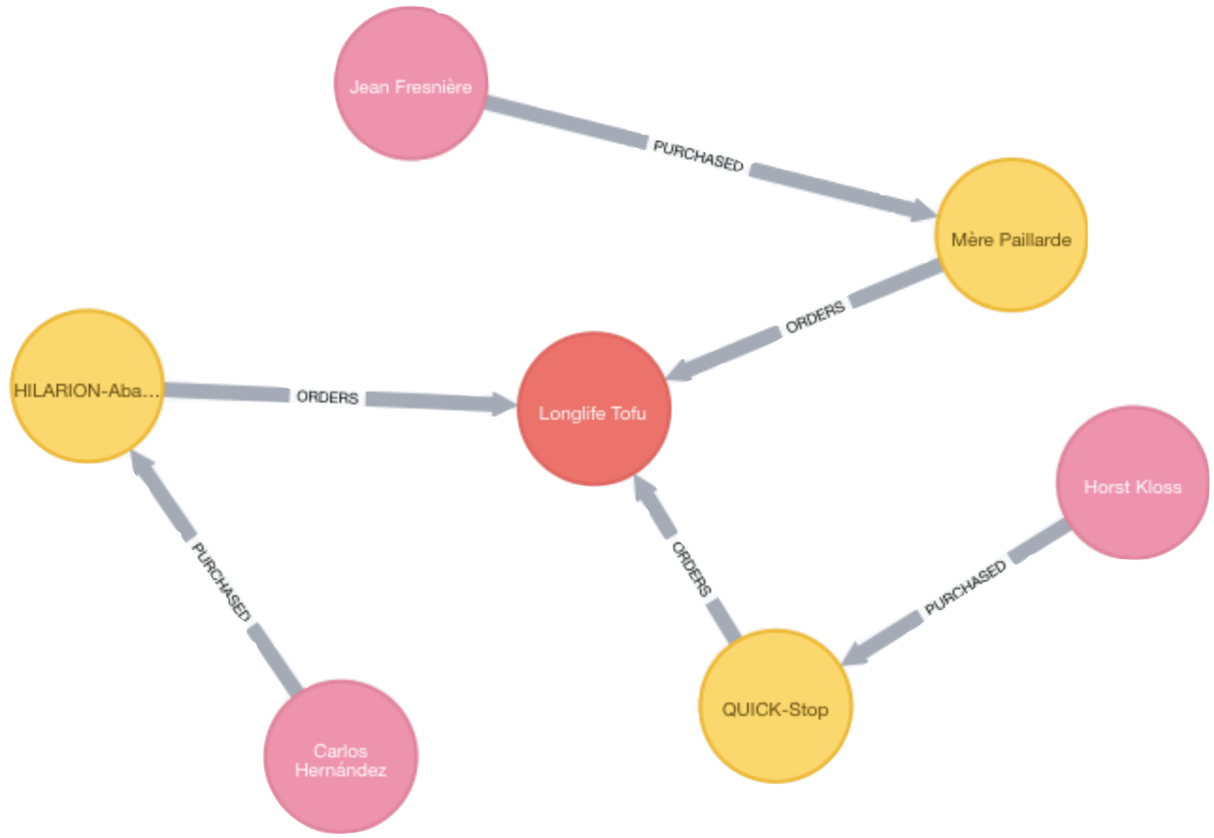
Quels produits ont été achetés ensemble ?
Requête coûteuse…


```
{
    "customers": {
        "customer": [
            {
                "id": "1",
                "firstName": "Jean",
                "lastName": "Serrien",
                "orders": [
                    {
                        "id": 1,
                        "date": "12/06/2016",
                        "products": [
                            {
                                "id": 1,
                                "name": "Beer",
                                "quantity": 10,
                                "unitPrice": 2
                            },
                            {
                                "id": 3,
                                "name": "Red wine",
                                "quantity": 5,
                                "unitPrice": 5
                            }
                        ]
                    }
                ]
            }
        ]
    }
}
```

Introduction

Pertinence des graphes pour modéliser les relations

- Les graphes représentent une voie naturelle pour modéliser des relations
- Possibilité de :
  - Ajouter une sémantique aux relations
  - Typer les noeuds
  - Définir des attributs sur les noeuds et les liens



Grande flexibilité

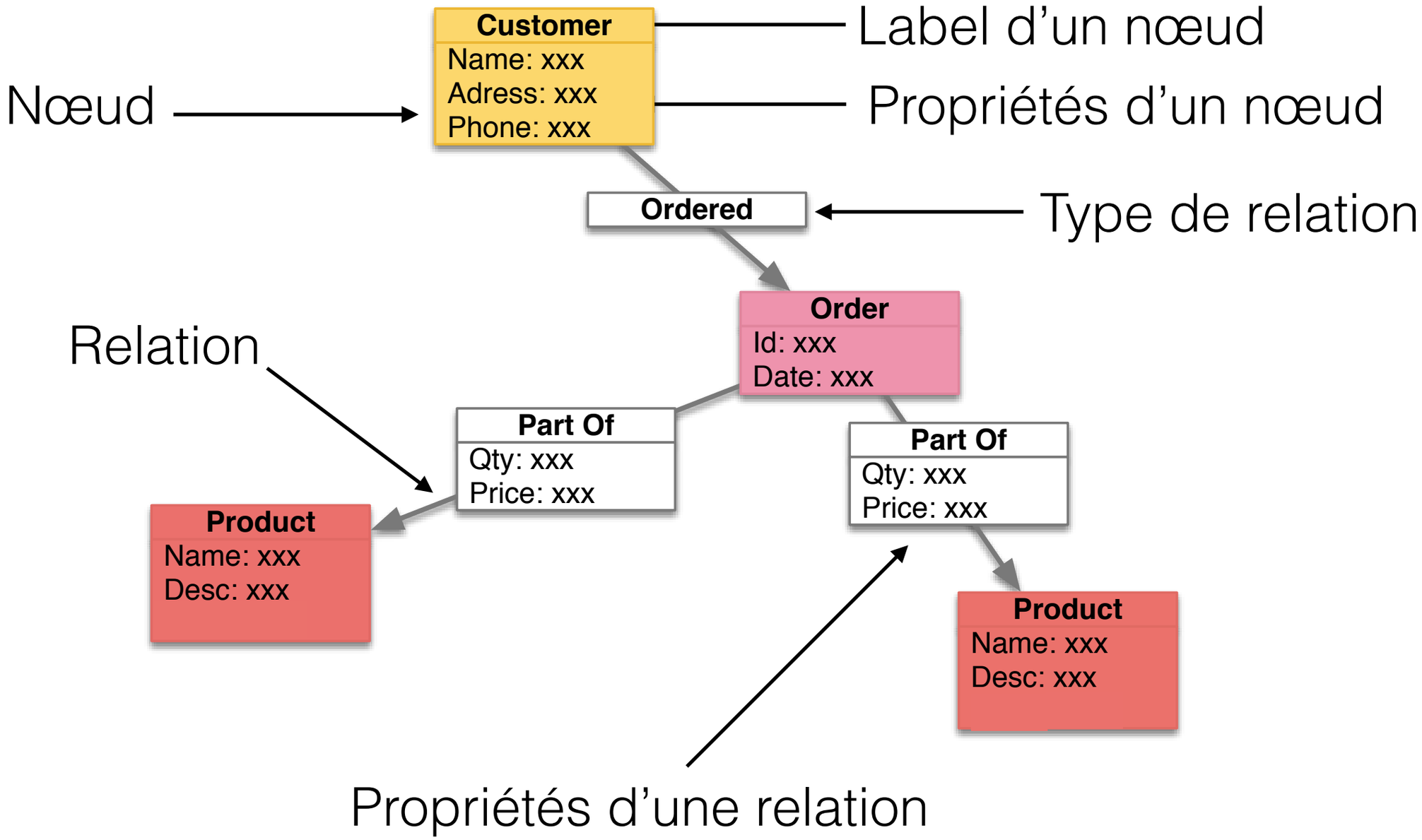
Concepts généraux

Les BD orientées graphe introduisent généralement les concepts de :

- **Noeud :** une entité
- **Label :** permet de regrouper des nœuds entre eux
- **Relation :** matérialise un lien (dirigé) entre deux nœuds, possède un type
- **Propriétés :** un nœud ou une relation peuvent disposer de propriétés (numériques, chaînes de caractères, booléens ou une liste des précédents types)

Concepts généraux

Exemple



Bases de données graphe

Caractéristiques

- Stockage optimisé pour données de type graphe
- Permet de traverser le graphe aisément
- Optimisé pour les requêtes orientées voisinage
- Modèle de données flexible

Les entités doivent forcément être liées!

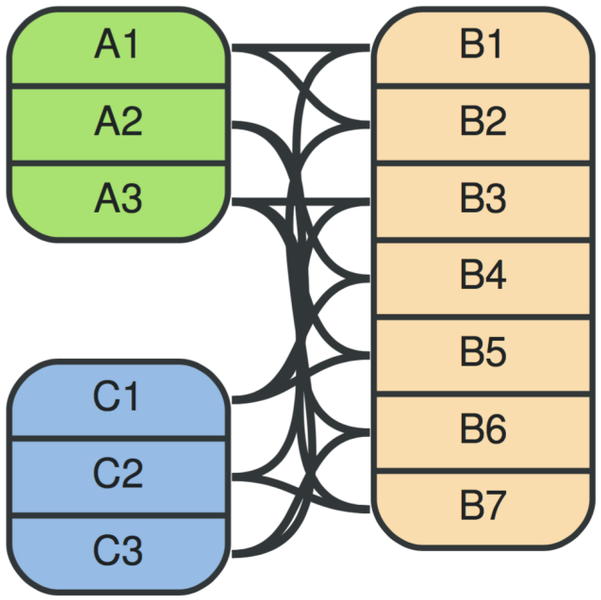
Principaux domaines applicatifs

- Recommandation
- Réseaux sociaux
- Monitoring
- Détection de fraudes

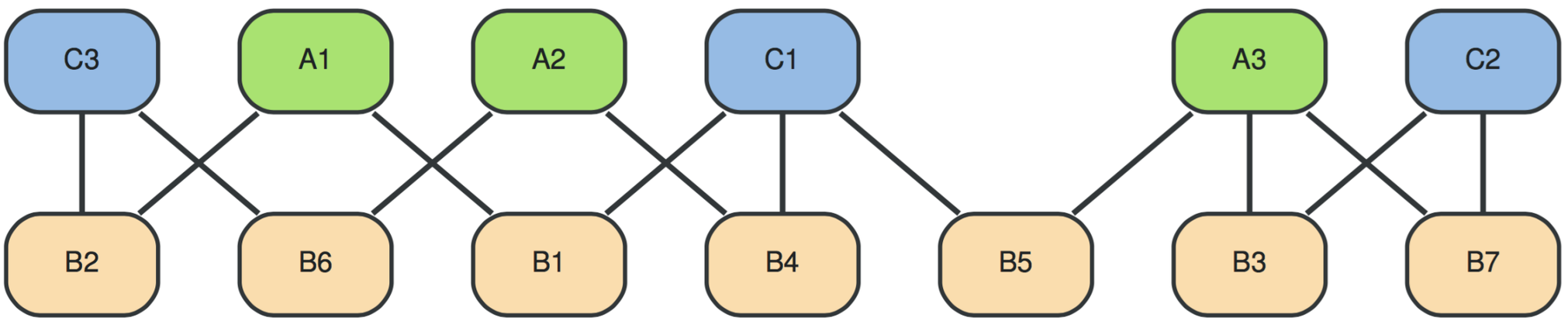
Bases de données graphe

Du relationnel au graphe

Relationnel



Graphe

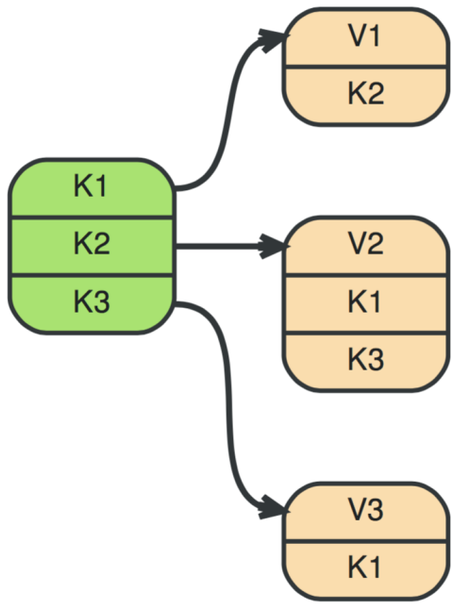


Source : le manuel Neo4j

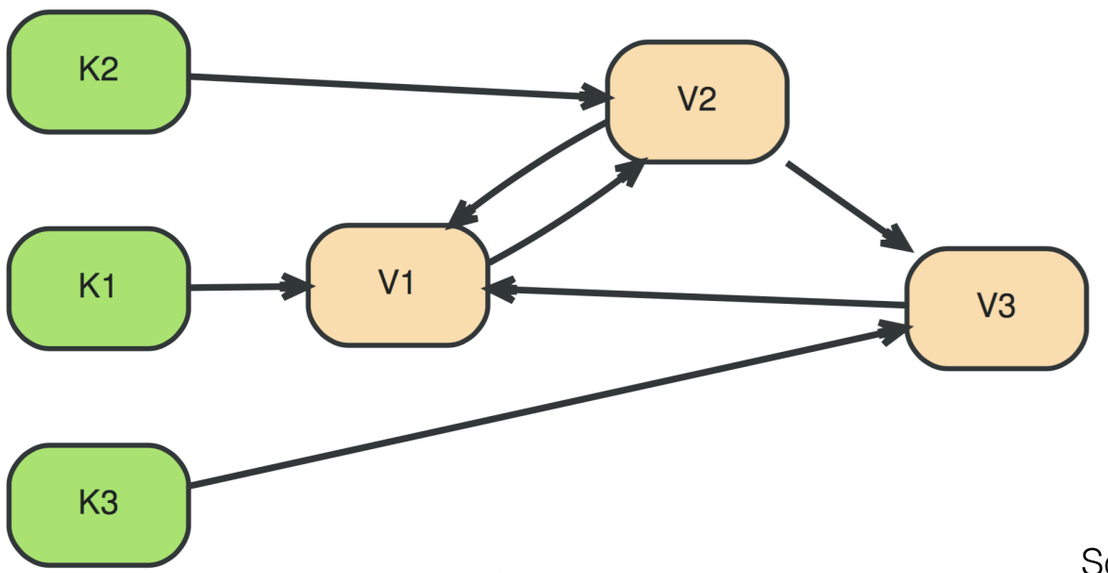
Bases de données graphe

De clé-valeur au graphe

Clé-valeur



Graphe

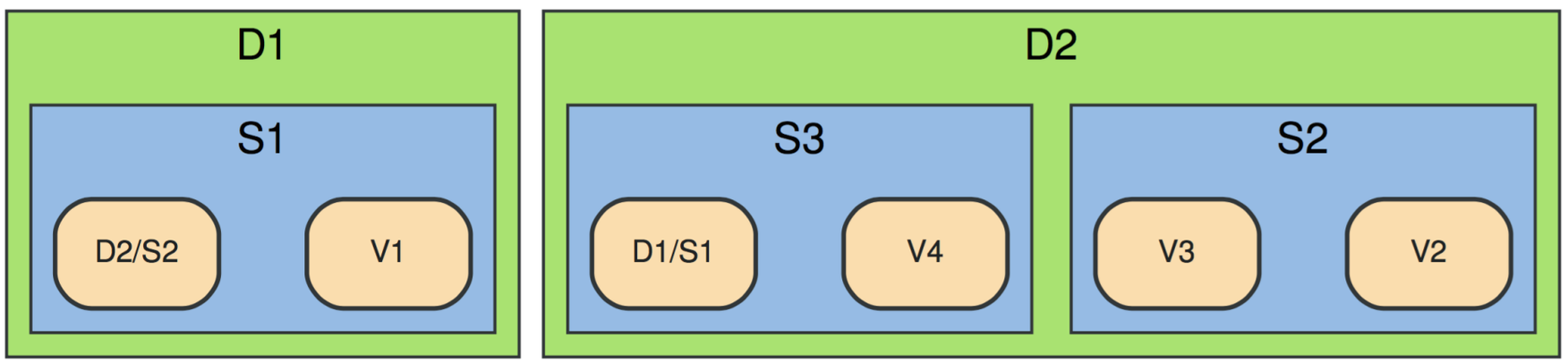


Source : le manuel Neo4j

Bases de données graphe

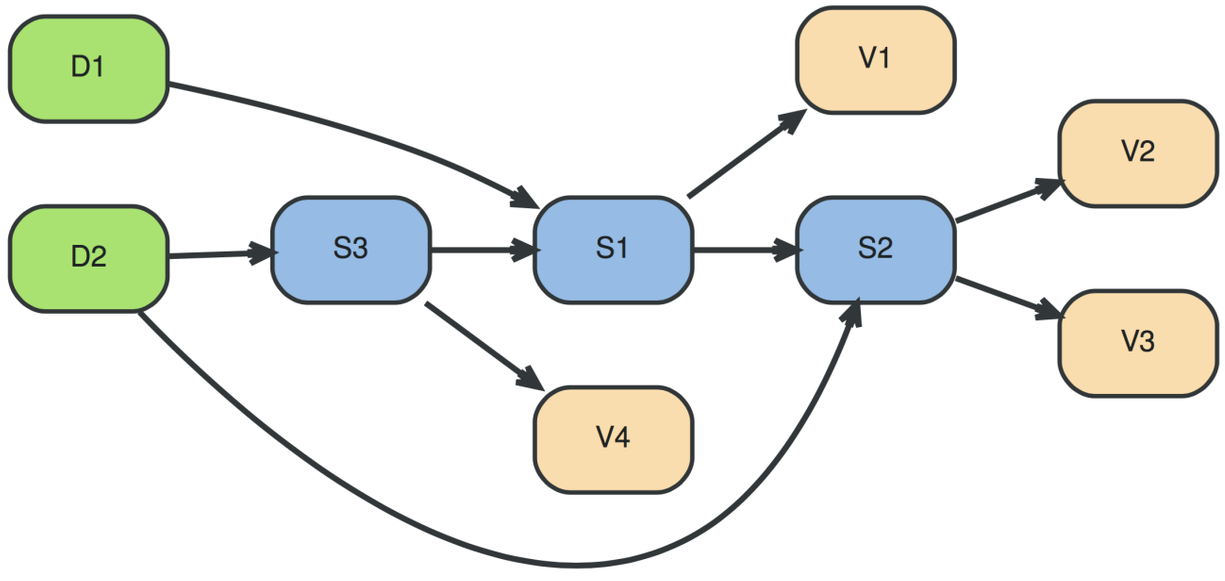
Du document au graphe

Document



D : document
S : sous-document
X/Y : référence d'un sous-document dans un document

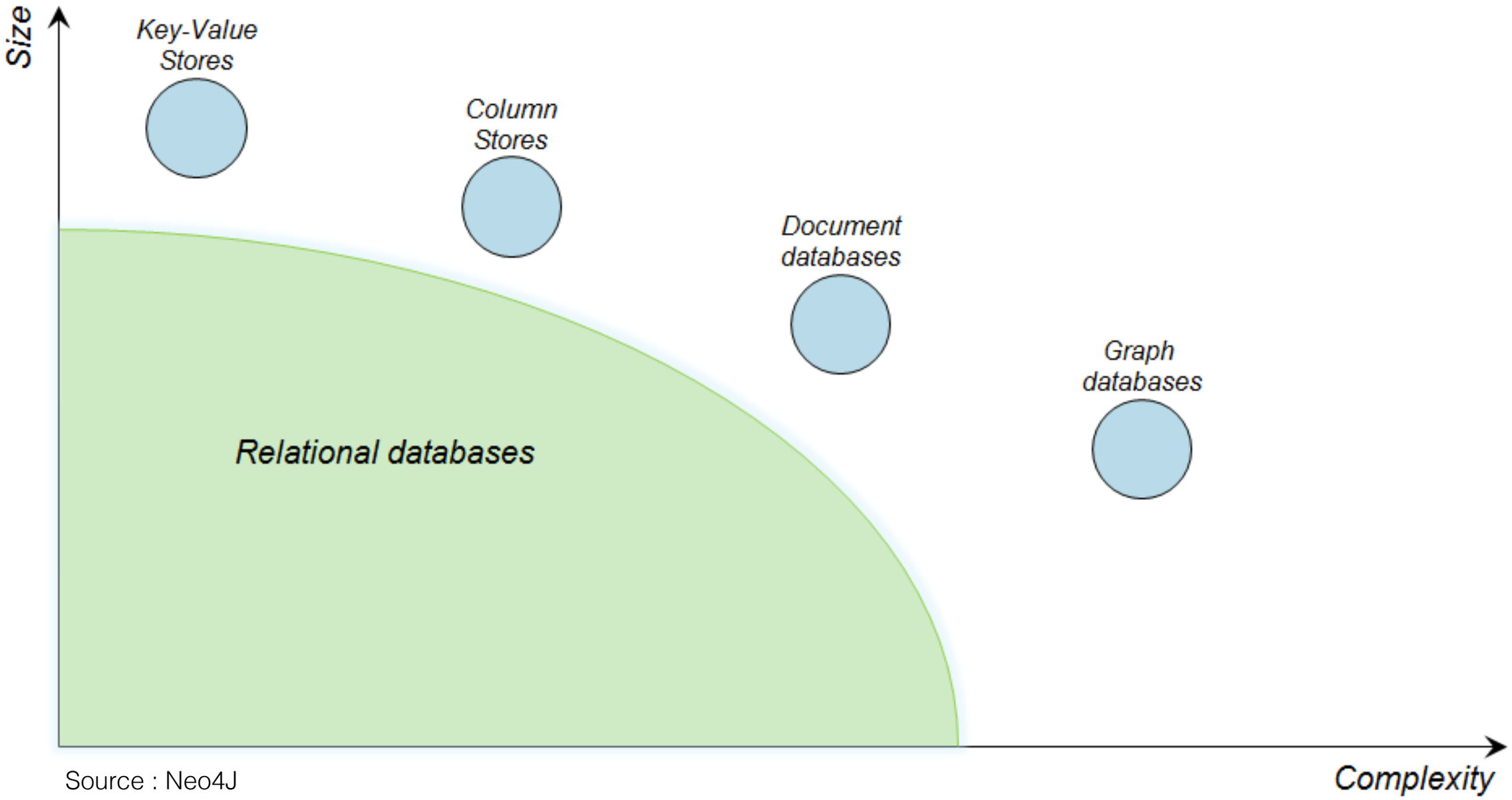
Graphe



Source : manuel Neo4j

Bases de données graphe

Position dans l'écosystème NoSQL



Source : Neo4J

Bases de données graphe

Etat des lieux

Neo4J

- Licence GPL, ACID compliant, basé sur Java

Infinite Graph

- Propriétaire (Objectivity), passage à l'échelle virtuellement illimité, utilisé à la CIA et au département de la défense américain

AllegroGraph

- Propriétaire (Franz Inc.), linked data et Web sémantique, supporte SPARQL, RDFS++ et Prolog

FlockDB

- Créé par Twitter, pas de version stable, requêtes limitées

Bases de données graphe

Etat des lieux

Neo4J

- Licence GPL, ACID compliant, basé sur Java

Infinite Graph

- Propriétaire (Objectivity), passage à l'échelle virtuellement illimité, utilisé à la CIA et au département de la défense américain

AllegroGraph

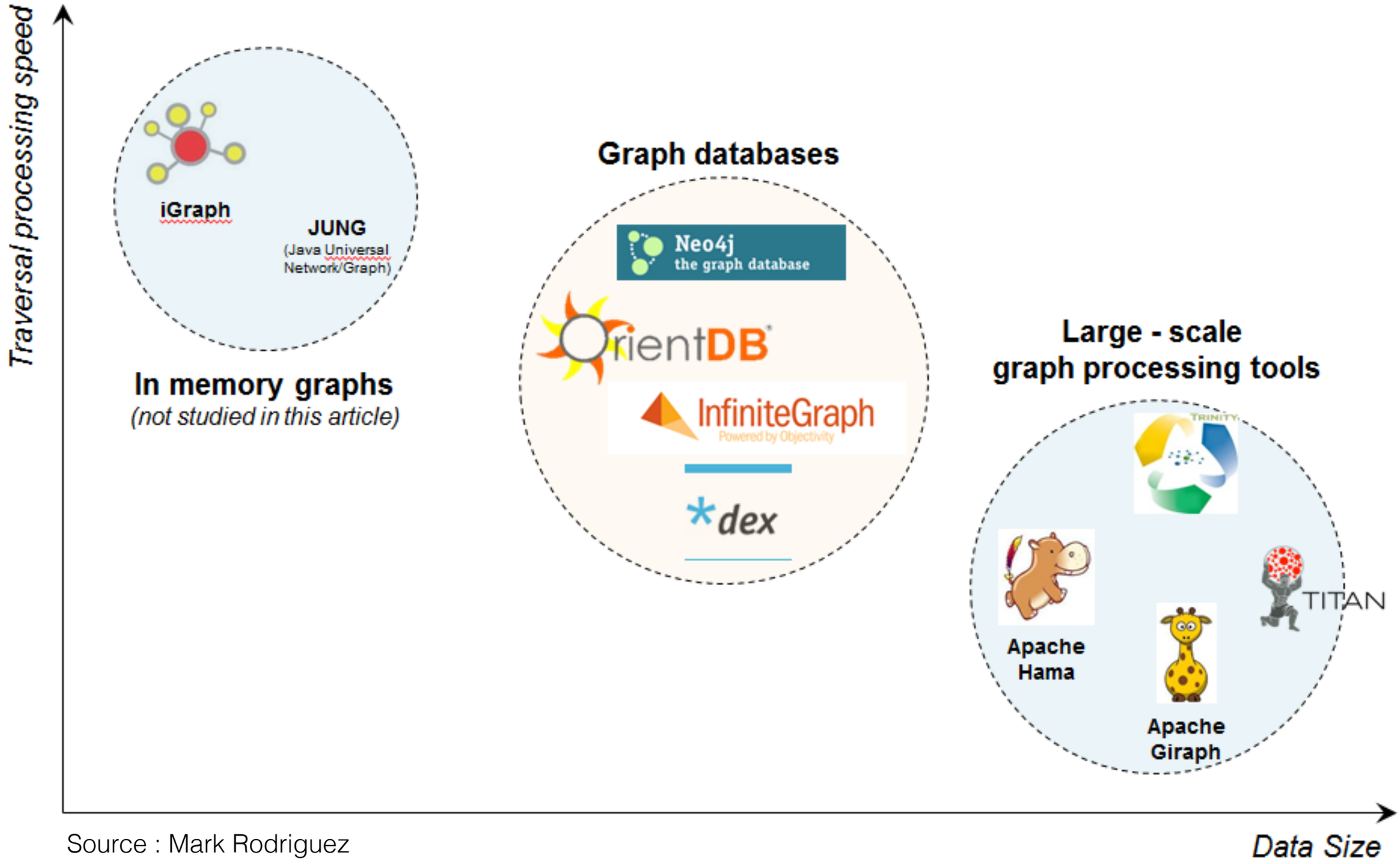
- Propriétaire (Franz Inc.), linked data et Web sémantique, supporte SPARQL, RDFS++ et Prolog

FlockDB

- Créé par Twitter, pas de version stable, requêtes limitées

Bases de données graphe

Etat des lieux



Source : Mark Rodriguez

Bases de données graphe

Quand les utiliser ?

Pourquoi ?

- Problèmes avec des jointures
- Evolution constante du jeu de données
- Modélisation naturelle des données sous forme de graphes
- Pour des développements rapides et itératifs

Qui (quelques bigs names) ?

- Microsoft (Microsoft Graph)
- Twitter
- Facebook (Facebook Graph)
- Google (Knowledge Graph)

Cas d'étude

Gestion des commandes

Relationnel

Customer

Id	Name
1	Martin

Product

Id	Name
27	NoSQL Distilled

Orders

Id	CustomerId	ShippingAdressId
99	1	77

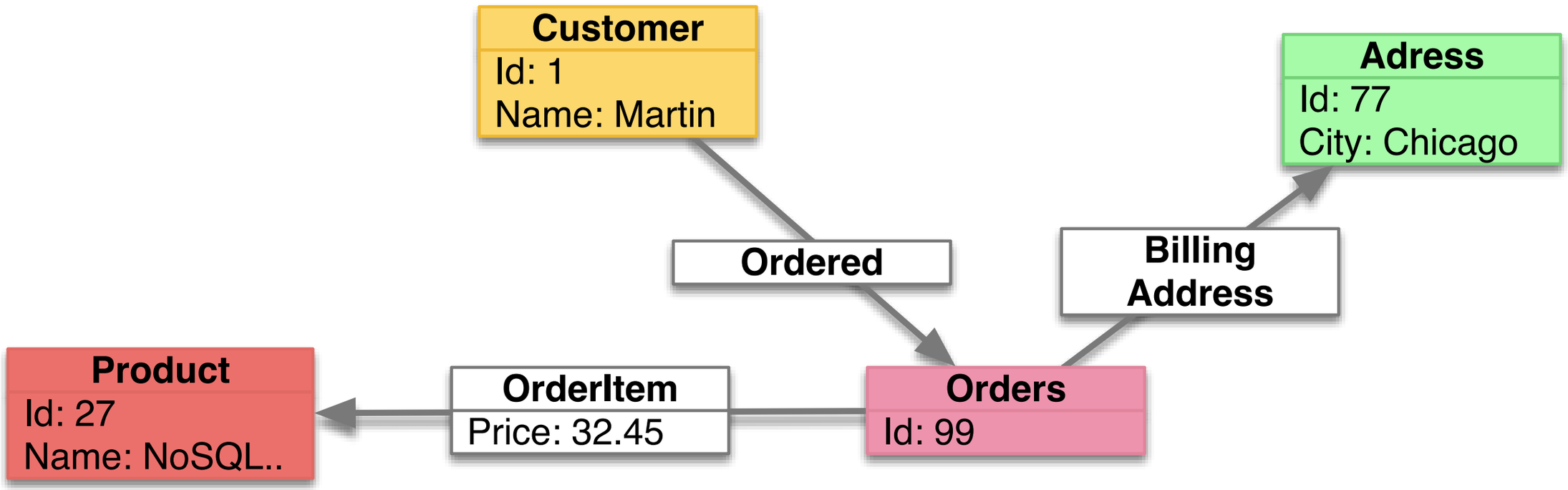
Address

Id	City
77	Chicago

OrderItem

Id	OrderId	ProductId	Price
100	99	27	32.45

Graphe



Focus

Neo4j

Quelques faits

Logiciel libre (licence GLPv3), Java, projet initié en 2000 (version 1.0 en 2010), une des BD graphes les plus évoluées et les plus robustes, version courante : 4.4

Principales caractéristiques

- Transaction : c'est une base de données transactionnelle, respectueuse des principes ACID
- Haute disponibilité : via la mise en place d'un cluster
- Volumétrie : stocker et requêter des milliards de nœuds et de relations
- Cypher : un langage de requête graphe déclaratif, simple et efficace
- Schemaless : pas de schéma préétabli

Bases de données orientées graphe

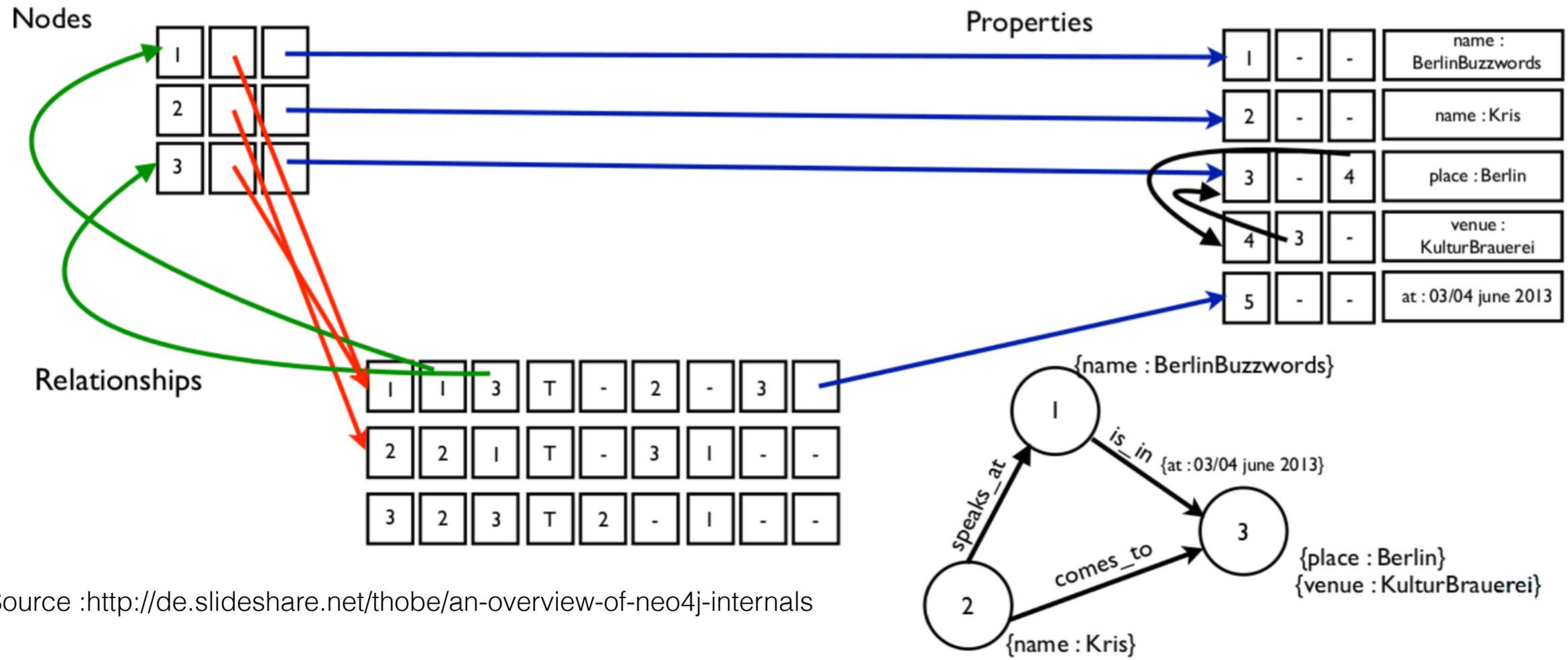
Adoption de Neo4J dans les entreprises

Source : http://fr.slideshare.net/maxdemarzi/graph-database-use-cases

Neo4j

Architecture

- Plusieurs fichiers sont physiquement utilisés pour le stockage
- Données stockées comme des listes d'enregistrements liées
- Nœuds, propriétés et relations sont stockés séparément



Source :http://de.slideshare.net/thobe/an-overview-of-neo4j-internals

Neo4j

Cohérence

Support des transactions ACID

- Isolation des opérations concurrentes jusqu'à complétion
- Tri des opérations « écriture » pour assurer un ordre de mise à jour prévisible
- Ecritures stockées dans le log de transaction de manière ordonnée
- Application des modifications dans les fichiers de données
- Changements stockés jusqu'à la fin de la transaction
- Processus de récupération : ré-application du log de transaction

Neo4j

High Availability (HA)

- Réplication des données à travers différents serveurs
- Architecture maître / esclaves :
  - Redondance des données
  - Tolérance à la faute
- Protocole d'élection du maître
  - Une majorité de serveurs doivent être opérationnels pour effectuer une écriture
- Transactions d'abord sur le maître
  - Génération d'un identifiant
  - Appliquées ensuite aux esclaves
  - Mise à jour entraîne un retard (eventual consistency)
- Transactions chez les esclaves:
  - Verrous coordonnés par le maître
  - Même identifiant que pour le maître

Neo4j

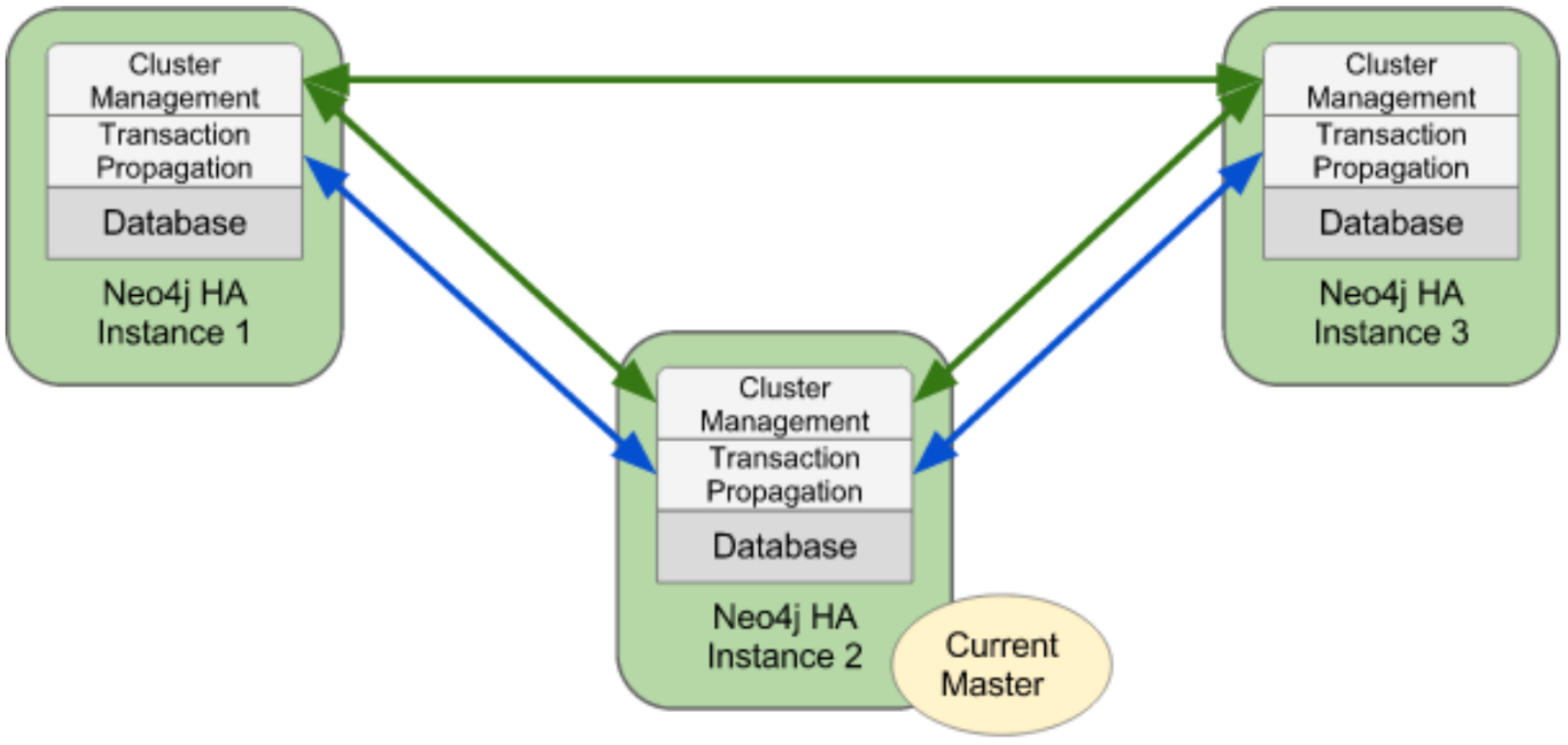
High Availability (HA)

- Réplication des données à travers différents serveurs
- Architecture maitre / esclaves :
  - Redondance des données
  - Tolérance à la faute
- Protocole d'élection du maître
  - Une majorité de serveurs doivent être opérationnel pour effectuer une écriture
- Transactions d'abord sur le maître
  - Génération d'un identifiant
  - Appliquées en
  - Mise à jour ent
- Transactions chez les esclaves:
  - Verrous coordonnées par le maître
  - Même identifiant que pour le maître

Passage à l'échelle difficile en écriture

Neo4j

High Availability (HA)



Source : https://neo4j.com/docs/ogm-manual/current/

Neo4j

Réplication et performances en lecture

Réplication

- Graphe répliqué entièrement sur chaque serveur
- Aucune limitation sur la taille des graphes
- Lecture possible sur tous les nœuds du cluster

Performances en lecture

- Capacité de lecture augmente linéairement avec la taille du cluster
- Taille du graphe n'impacte pas sur les performances (≠BDR)
- **Cache-based sharing:**
  - Routage consistant des requêtes pour optimiser utilisation de la RAM

Requêtes pour un nœud A envoyées systématiquement au serveur 1
Requêtes pour un nœud B envoyées systématiquement au serveur 2
…

Neo4j

Réplication et performances en lecture

Réplication

- Graphe répliqué entièrement sur chaque serveur
- Aucune limitation sur la taille des graphes
- Lecture possible sur tous les nœuds du cluster

Performances en lecture

- Capacité de lecture augmente linéairement avec la taille du cluster
- Taille du graphe n'impacte pas sur les performances (≠BDR)
- **Cache-based sharing:**
  - Routage consistent des requêtes pour optimiser utilisation de la RAM

Bon passage à l'échelle en lecture

Neo4j

Sharding

Principe

Eclatement des données sur plusieurs serveurs

Constat

- Concept très apprécié dans les BDs clé-valeur et orientées documents
- Stockage d'enregistrements individuels

Sharding et Neo4j

- Stockage des relations central dans les BDs graphe
- Sharding optimal: problème NP-complet
- Très difficile mais Neo4j travaille sur ce point

Neo4j

Interrogation - Cypher

- Langage déclaratif pour formuler des requêtes sur des graphes
- Permet d'interroger et/ou mettre à jour le graphe
  - Chaque partie de la requête doit être « écriture » ou « lecture » seulement
  - Une requête = plusieurs clauses
- Transactions : plusieurs requêtes possibles
- Variables, expressions, opérateurs, commentaires
- Collections (liste, dictionnaire)
- Ensemble de fonctions natives (collections, agrégation, chaînes de caractères, maths)

Neo4j

Cypher

Spécification de structures de graphes via des motifs

Nœud

- Nœud anonyme : ()
- Nœud nommé : (x)
- Nœud avec un label spécifique : (:label)

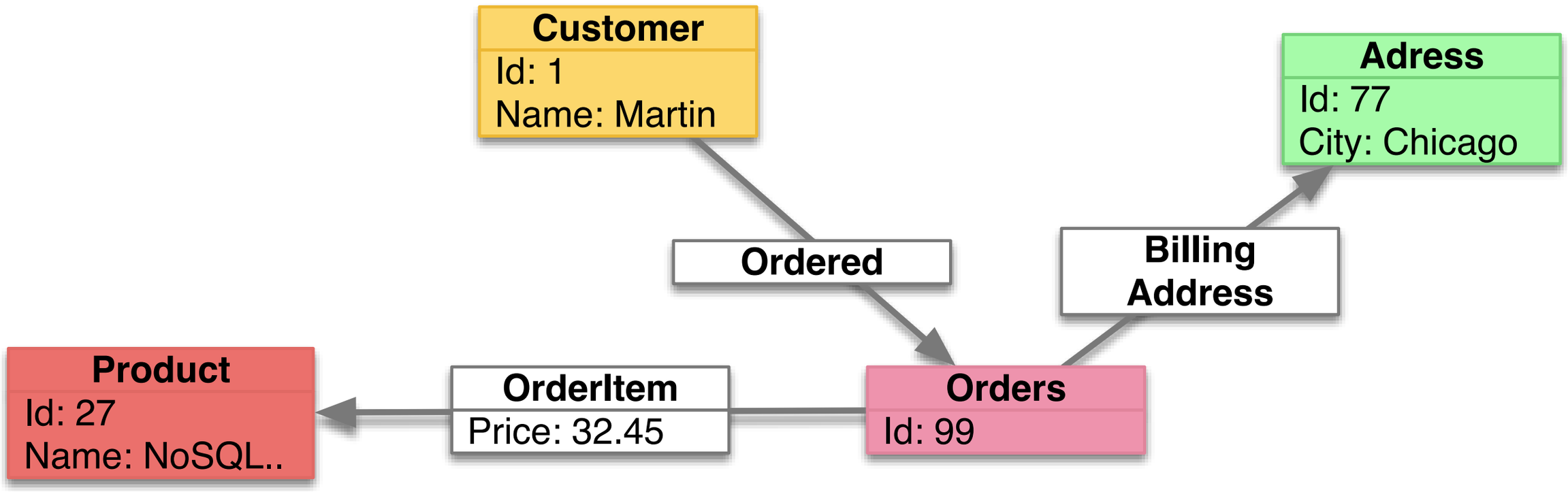
Relations

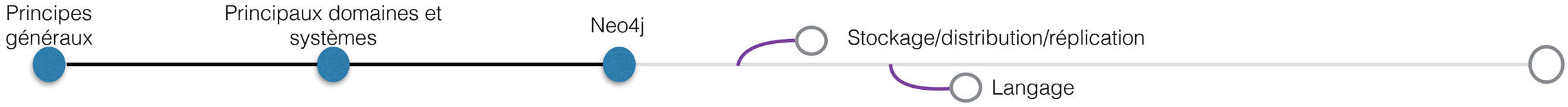
- Relation anonyme: -[]->
- Relation nommée : -[r]->
- Relation nommée avec un label spécifique : -[r:t]->
- 2 nœuds avec une relation : (a)-[r]->(b)

- Les propriétés peuvent être spécifiées via {}, e.g., (x {city: ‘’Chicago’’})
- Un motif peut combiner plusieurs nœuds et relations

Neo4j

Rappel du cas d'étude





Neo4j

Cypher - Clauses générales

- **MATCH :** recherche un élément et retourne une table ou un sous-graphe
  - **DISTINCT :** élimine la redondance
  - **OPTIONAL MATCH :** relation optionnelle (jointure externe en SQL)

- **WHERE :** sélection
  - Supporte les expressions régulières sur les chaînes

- **RETURN :** retourne le sous-graphe ou la table

Neo4j

Cypher - Exemples interrogations simples

```
// Renvoie tous les noeuds d'une base
MATCH (n) RETURN n

// Renvoie tous les noeuds de label Product
MATCH  (n:Product)  RETURN n

// Renvoie tous les noeuds de label Product
// qui ont pour propriété name : Neo4j in a nutshell
MATCH  (n:Product{name : "Neo4j in a nutshell"})
RETURN n

// Idem
MATCH  (n:Product)
WHERE n.name = "Neo4j in a nutshell"
RETURN n

// Retourne un produit et certaines de ses propriétes
MATCH (prod:Product { name: "NoSQL Distilled" })
RETURN prod.name;
```

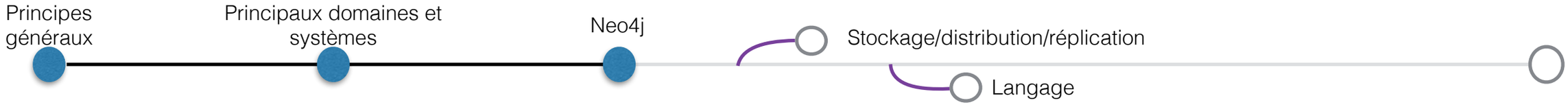
Neo4j

Cypher - Exemples interrogations simples

```
// Retourne les commandes de Martin
MATCH (a:Customer { name: 'Martin' })-[:]->(x)
RETURN x


// Retourne les clients triés selon leur nom
MATCH (customer:Customer)
RETURN customer ORDER BY customer.name;

// Tous les clients dont le nom termine par in
MATCH (customer:Customer)
WHERE customer.name =~ ".*in$"
RETURN customer.name;
```



Neo4j

Cypher - Clauses d'Écriture

- **CREATE :** crée un nœud ou une relation
- **MERGE** : crée un noeud ou une relation si elle n'existe pas déjà
- **SET :** modifie/ajoute des données/labels
- **REMOVE :** supprime des labels et des propriétés
- **DELETE :** supprime des éléments du graphe

Neo4j

Cypher - Exemples création

```
// Peuple une base
CREATE (prod1:Product {name : "NoSQL Distilled"})
CREATE (ord1:Orders)
CREATE (ord1)-[:ORDERITEM { Price : 32.45 }]->(prod1)

// OU
CREATE  (ord1:Orders) -[:ORDERITEM { Price : 32.45 }] ->
                       (prod1:Product {name : "NoSQL Distilled"})


// Charge une table depuis un CSV volumineux
USING PERIODIC COMMIT
LOAD CSV WITH HEADERS FROM 'http://neo4j.com/docs/2.2.5/csv/
artists-with-headers.csv' AS line
CREATE (:Artist { name: line.Name, year: toInt(line.Year)})
```

Neo4j

Cypher - Exemples mise à jour / suppression

```
// Fixe la propriété name des noeuds dont la propreté name valait "NoSQL
Distilled" à "NoSQL Distilled v1"
MATCH (n) WHERE n.name = "NoSQL Distilled" SET n.name = "NoSQL Distilled v1";

// Suppression d'une propriété
MATCH (n) -[r:ORDERITEM]-> (m)
REMOVE r.Price

// Suppression d'une relation
MATCH (m) -[r:ORDERITEM]-> (n:Product {name : "NoSQL Distilled"})
DELETE r

// Suppression d'un noeud et de ses relations
MATCH (n:Product {name : "NoSQL Distilled"}) DETACH
DELETE n

// Nettoyage complet de la base
MATCH (n) DETACH  DELETE n;
```

Neo4j

Cypher - Fonctions générales utiles

- **id() :** identifiant du nœud
- **labels ():** le(s) label(s) du nœud
- **type()** : le « type de la relation (son label)
- **length(path), relationships(path), nodes(path)**
- **timestamp() :** une estampille temporelle
- **upper(), lower() :** modification de la casse
- **range(l,u):** retourne une liste entre l et u

Neo4j

Cypher - Clauses générales utiles

- **FOREACH :** pour mettre à jour les composant d'une collection (chemin ou résultat d'une agrégation)
- **UNWIND:** crée une séquence de lignes à partir d'une collection
- **WITH** : permet de chainer les parties d'une requête (pipeline) en utilisant les résultats d'un prédicat précédent en entrée d'un nouveau (similaire à RETURN)
  - Retrouver les top nœuds d'après un critère **PUIS** faire une jointure
  - Pour combiner des opérations de lecture et d'écriture
  - Indispensable en cas d'agrégation
- **ORDER BY x (ASC I DESC) :** tri
- **SKIP, LIMIT X :** pagination
- **UNION :** composition d'expressions

Neo4j

Cypher - Lecture

- **Fonctions d'agrégation :**
  - Regroupement automatique sur toutes les colonnes non concernées par la fonction d'agrégation
  - **SUM**, **AVG**, **COUNT**
    - COUNT(*), COUNT(DISTINCT X)
  - **COLLECT(X) :** crée une liste de toutes les valeurs

Neo4j

Cypher - Exemples

```
// Création graphe étoile
CREATE (c) FOREACH (x IN range(1,6)| CREATE (l),(c)-[:Rel]->(l)) RETURN id(c);
id
0
Updated the graph - created 7 nodes and 6 relationships

// Nombre de nœuds
MATCH (n) RETURN count(n); # since we have not defined any restriction, all nodes
count(n)
7

// Nombre de relations basées sur le type
MATCH ()-[r]->() RETURN type(r), count(*);
type(r) count(*)
Rel       6

// Suppression de la redondance dans une liste
WITH [1,1,2,2] AS coll UNWIND coll AS x WITH DISTINCT x RETURN collect(x)
[1,2]
```

Neo4j

Cypher - Exemples

```
// Liste tous les nœuds et leurs relations
MATCH (n)-[r]->(m) RETURN n AS De , r AS '->', m AS Vers;

// Retourne les commandes des produits apparaissant dans la commande 99
MATCH (:Order { id: 99 })-[:ORDERITEM]->(product)<-[:ORDERITEM]-(order)
RETURN order.id;
```

```
// Filtre
MATCH (o:Order)-[r:ORDERITEM]->(p:Product)
WHERE p.name =~ "N.+" OR r.price > 12.5
RETURN p,r,o

// Filtre basé sur la structure du graphe
MATCH (c:Customer)-[:ORDERED]->(o) WHERE NOT (o)-[:BILLINGADDRESS]->(:Address
{City:"Chicago"})
RETURN c,o

// Combien de fois les produits ont été achetés par ville
MATCH (p:Product)<-[:ORDERITEM]-(o:Order)-[:BILLINGADDRESS]->(a:Address)
RETURN p.name,a.city,count(*) AS achats
```

Neo4j

Cypher - Exemples

```
// Utilisation de l'union
MATCH (o1:Orders)-[r:BILLINGADDRESS]->(a1:Address {City:'Toulouse'}) RETURN o1, a1
UNION
MATCH (o2:Orders)-[r:BILLINGADDRESS]->(a2:Address {City:'Chicago'}) RETURN o2, a2

// Retourne 5 produits par commande
MATCH (p:Product)<-[:ORDERITEM]-(o:Order)
RETURN id(o) AS commande, collect(p.name)[0..5] AS cinq_produits



MATCH (o)-[:ORDERITEM]->(product)
WITH o, count(product) as nombreProduits
WHERE nombreProduits > 3
SET o.productCount = nombreProduits
RETURN o, nombreProduits

// Mise à jour de tous les chemins possibles entre A et D
MATCH p =(begin)-[*]->(end)
WHERE begin.name="A" AND end.name="D"
FOREACH (n IN nodes(p)| SET n.marked = timestamp() )
```

Neo4j

Cypher - Contraintes, index et debug

Contraintes

```
CREATE CONSTRAINT ON (p:Product) ASSERT p.name IS UNIQUE
DROP CONSTRAINT ON (p:Product) ASSERT p.name IS UNIQUE
```

Index

```
CREATE INDEX ON :Product(name)
DROP INDEX ON :Product(name)
```

Debug

- **EXPLAIN :** montre le plan d'exécution
- **PROFILE :** exécute les opérations et indique le temps passé pour chaque

Neo4j

Offre

Cloud

Self-managed

Graph database

AuraDB

Community Edition
Enterprise Edition

Graph datascience

AuraDS

Graph datascience Community
Graph datascience Enterprise

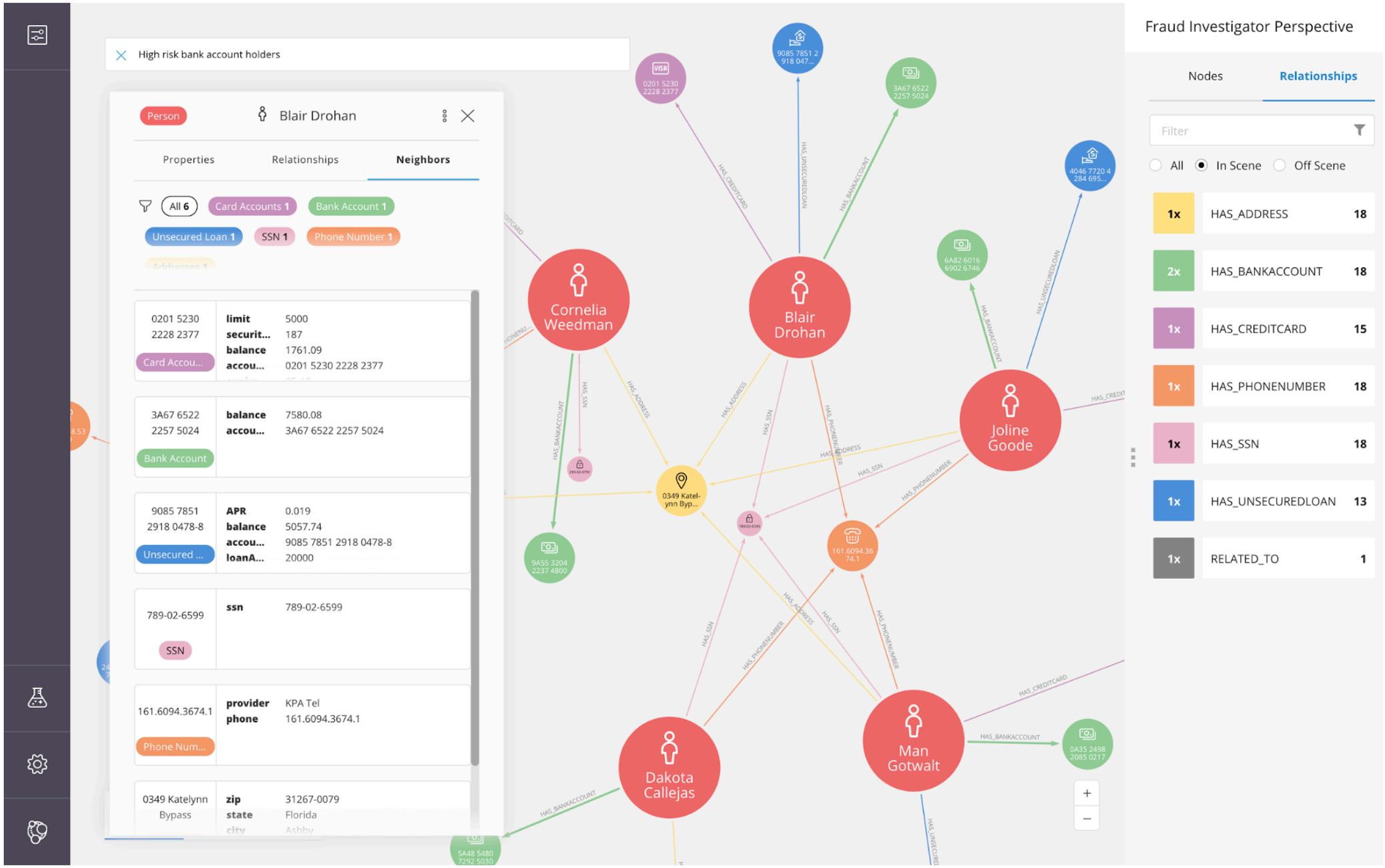
Neo4j

Interrogation - Interfaces

- **Bloom** : outil de navigation « pour les nuls »
- **Shell Neo4j**
  - Création, import, export, executer code Cypher
  - Résultats sous forme tabulaire ASCII
- **Neo4J Desktop**
  - Shell Cypher
  - Visualisation de résultats
  - Monitoring de performances
  - Support HTTPS
- API Python, NodeJS, Go, .Net, Java, PHP, Perl, etc.

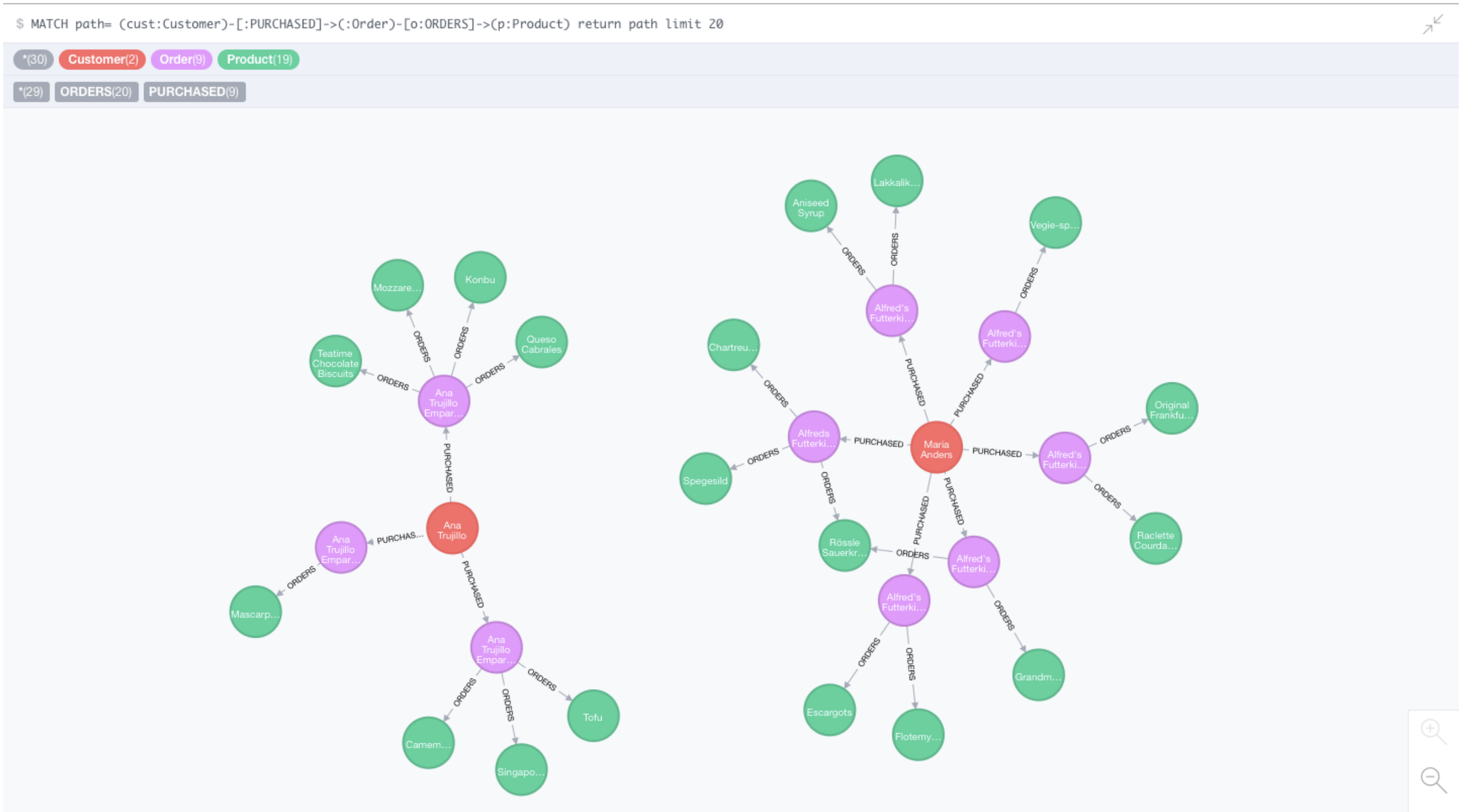
Neo4j

Bloom


High risk bank account holders
Person
Blair Drohan
Properties
Relationships
Neighbors
All 6
Card Accounts 1
Bank Account 1
Unsecured Loan 1
SSN 1
Phone Number 1
0201 5230 2228 2377
limit 5000
securit... 187
balance 1761.09
accou... 0201 5230 2228 2377
Card Accou...
3A67 6522 2257 5024
balance 7580.08
accou... 3A67 6522 2257 5024
Bank Account
9085 7851 2918 0478-8
APR 0.019
balance 5057.74
accou... 9085 7851 2918 0478-8
loanA... 20000
Unsecured ...
789-02-6599
ssn 789-02-6599
SSN
161.6094.3674.1
provider KPA Tel
phone 161.6094.3674.1
Phone Num...
0349 Katelynn Bypass
zip 31267-0079
state Florida
city Ashby
Cornelia Weedman
Blair Drohan
Joline Goode
Dakota Callejas
Man Gotwalt
Fraud Investigator Perspective
Nodes
Relationships
Filter
All
In Scene
Off Scene
1x HAS_ADDRESS 18
2x HAS_BANKACCOUNT 18
1x HAS_CREDITCARD 15
1x HAS_PHONENUMBER 18
1x HAS_SSN 18
1x HAS_UNSECUREDLOAN 13
1x RELATED_TO 1

Neo4j

Interface Web (Neo4j Desktop)



Neo4j

API Python

```
# import the neo4j driver for Python
from neo4j import GraphDatabase


# Database Credentials
uri             = "bolt://localhost:7687"
userName        = "neo4j"
password        = "test"

# Connect to the neo4j database server
graphDB_Driver  = GraphDatabase.driver(uri, auth=(userName, password))
cql             = "MATCH (x:film) RETURN x"

# Execute the CQL query
with graphDB_Driver.session() as graphDB_Session:
    nodes = graphDB_Session.run(cql)

    for node in nodes:
        print(node)
```

Neo4j

Modélisation - Méthode

1. Identifier l'application et les objectifs
2. Formuler les questions types
3. Identifier les entités dans chaque question
4. Identifier les relations dans chaque question
5. Convertir les entités et les relations en chemin

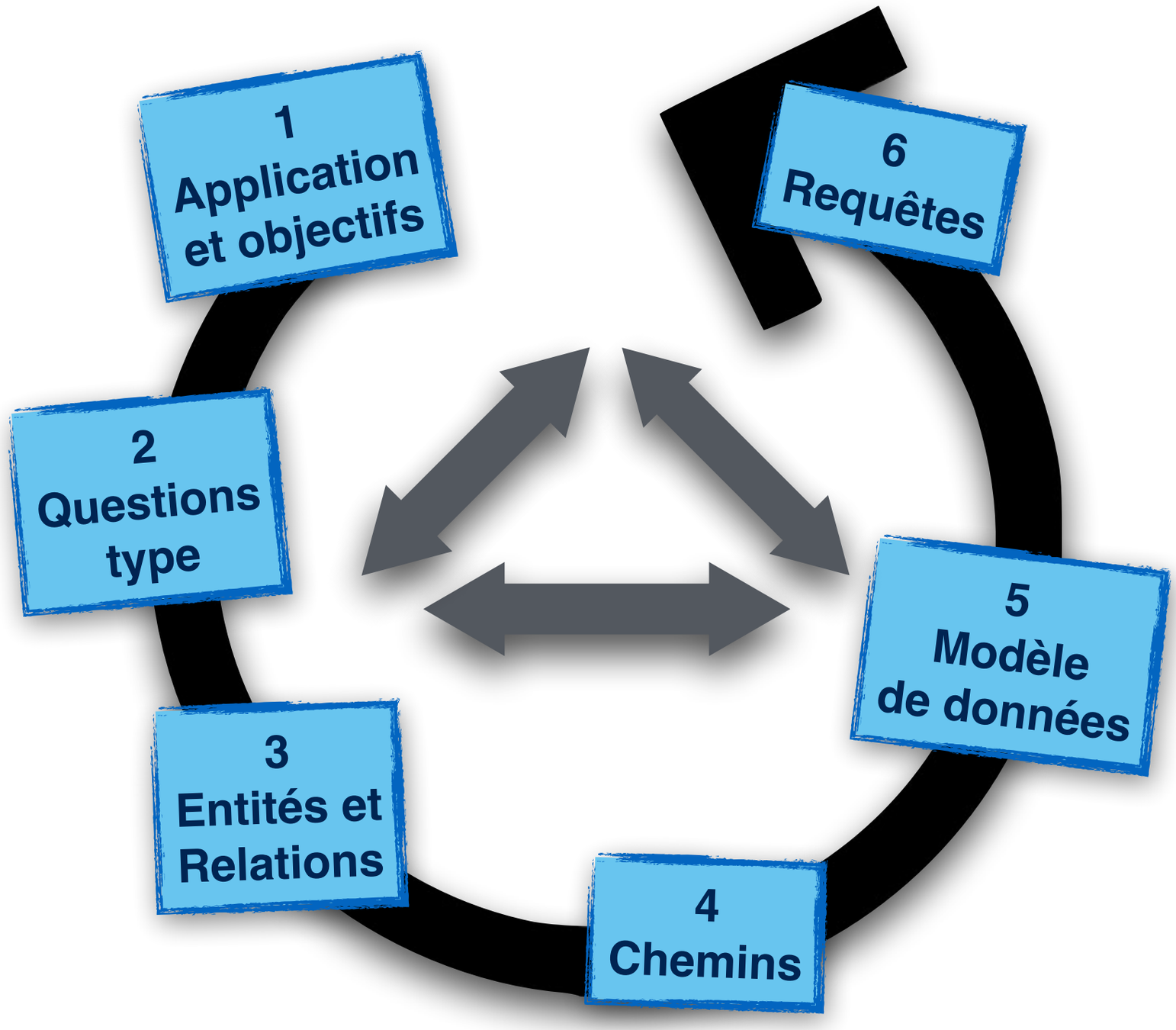
Eléments centraux du modèle de données

6. Exprimer les question sous forme de motifs de graphe

Eléments centraux des requêtes

Neo4j

Modélisation - Méthode



Neo4j

Modélisation - Application

En tant que gestionnaire des stocks

Je veux connaître :

- Qui achète quel produit
- En quelles quantités
- A quelle date
- Et où

De telle sorte à gérer mes stocks et réfléchir à une stratégie commerciale

Neo4j

Modélisation - Questions

En tant que gestionnaire des stocks

Je veux connaître :

- Qui achète quel produit
- En quelles quantités
- A quelle date
- Et où

De telle sorte à gérer mes stocks et réfléchir à une stratégie commerciale

Quels produits sont commandés pour chaque commande en quelles quantités et à quel prix unitaire ?

Qui effectue chaque commande et à quelle adresse la commande est livrée ?

Neo4j

Modélisation - Entités

Quels **produits** sont commandés pour chaque **commande** en quelles **quantités** et à quel **prix unitaire** ?

Quel **client** effectue chaque **commande** et à quelle **adresse** la commande est livrée ?

- Produit
- Commande
- Client
- Adresse

Neo4j

Modélisation - Relations

Quels produits **sont commandés** pour chaque commande en quelles quantités et à quel prix unitaire ?

Quel client **effectue** chaque commande et à quelle adresse la commande **est livrée** ?

- Produit APPARTIENT_A Commande
- Client EFFECTUE Commande
- Commande EST_LIVREE Adresse

Neo4j

Modélisation - Chemins

Produit **APPARTIENT_A** Commande

Client **EFFECTUE** Commande

Commande **EST_LIVREE** Adresse

Legende
Label
Relations

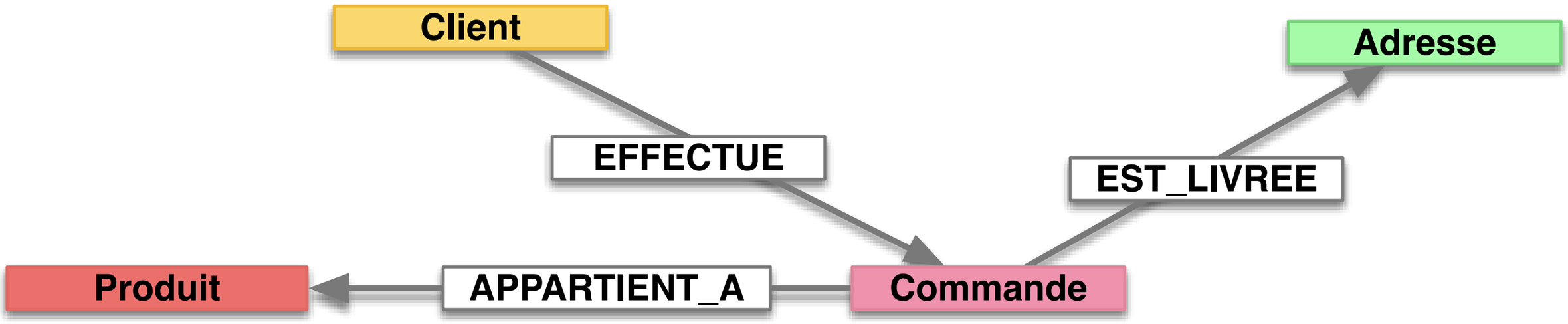
(:Produit)-[:APPARTIENT_A]->(:Commande)
(:Client)-[:EFFECTUE]->(:Commande)
(:Commande)-[:EST_LIVREE]->(:Adresse)

Neo4j

Modélisation - Consolidation des chemins

(:Produit)-[:APPARTIENT_A]->(:Commande)
(:Client)-[:EFFECTUE]->(:Commande)
(:Commande)-[:EST_LIVREE]->(:Adresse)

(:Produit)-[:APPARTIENT_A]->(:Commande)<-[:EFFECTUE]-(:Client)
(:Commande)-[:EST_LIVREE]->(:Adresse)



Neo4j

Modélisation - Requêtes

Quels produits sont commandés pour chaque commande ?

```
MATCH (p:Produit)-[:APPARTIENT_A]->(c:Commande)
RETURN id(c) as IdCommande, collect(p)
```

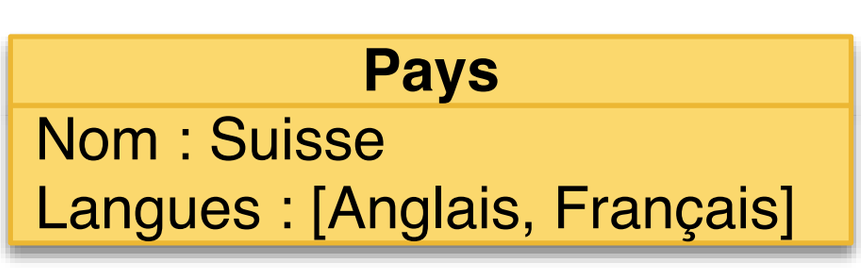
Quel client effectue chaque commande et à quelle adresse la commande est livrée ?

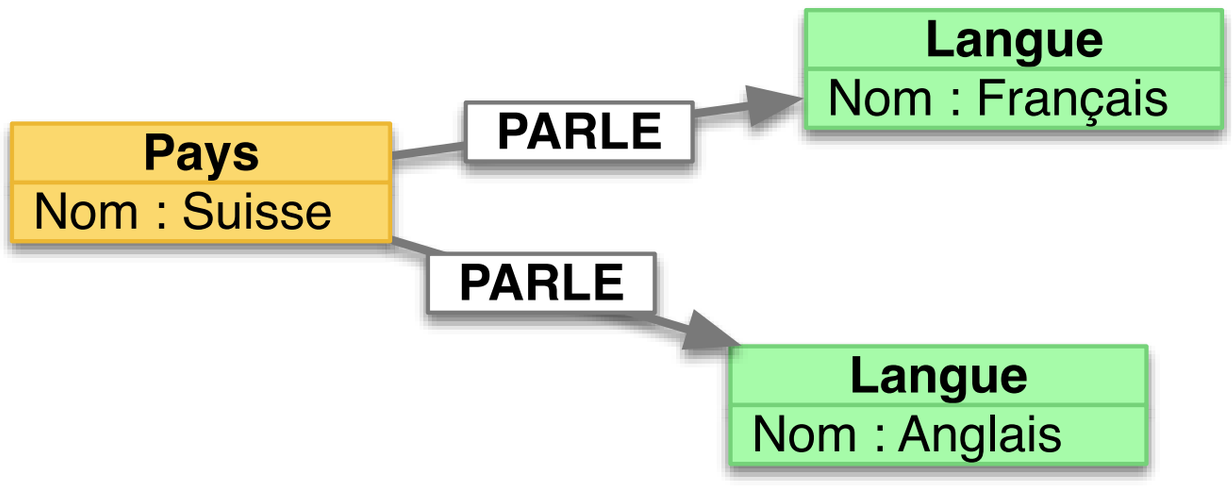
```
MATCH p=(:Client)-[:EFFECTUE]->(:Commande)-[:EST_LIVREE]->(:Adresse)
RETURN p
```

Neo4j

Modélisation - Mauvaises pratiques

Propriétés complexes

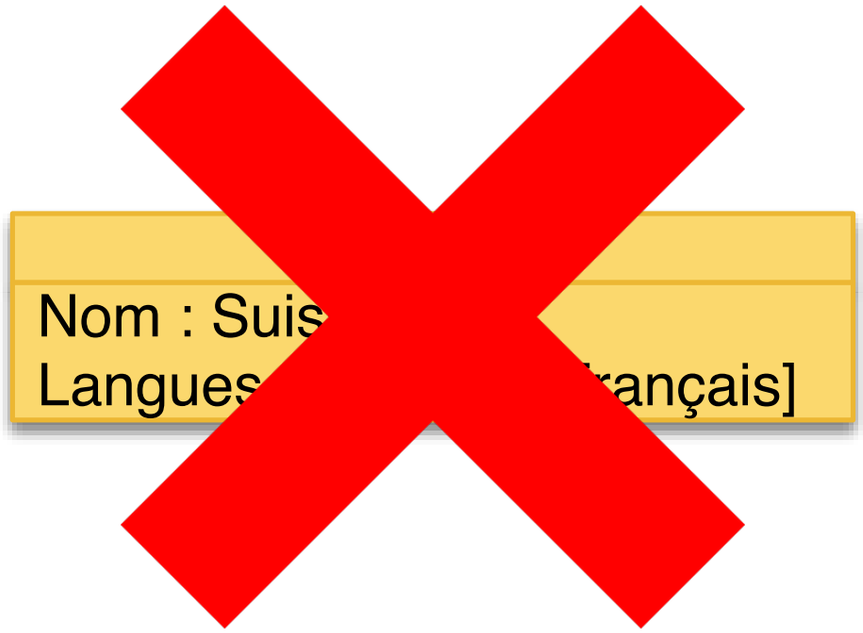


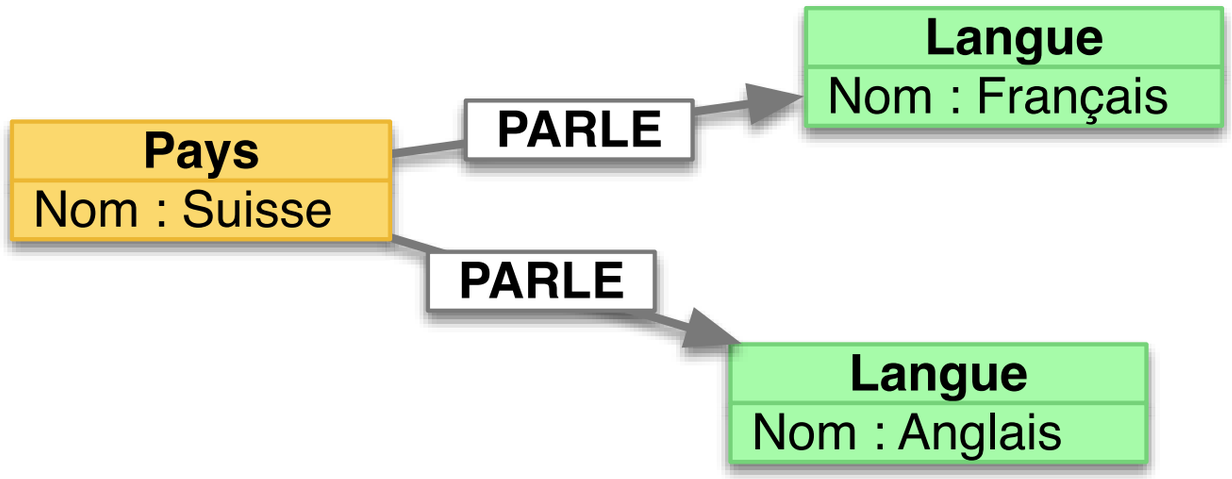


Neo4j

Modélisation - Mauvaises pratiques

Propriétés complexes

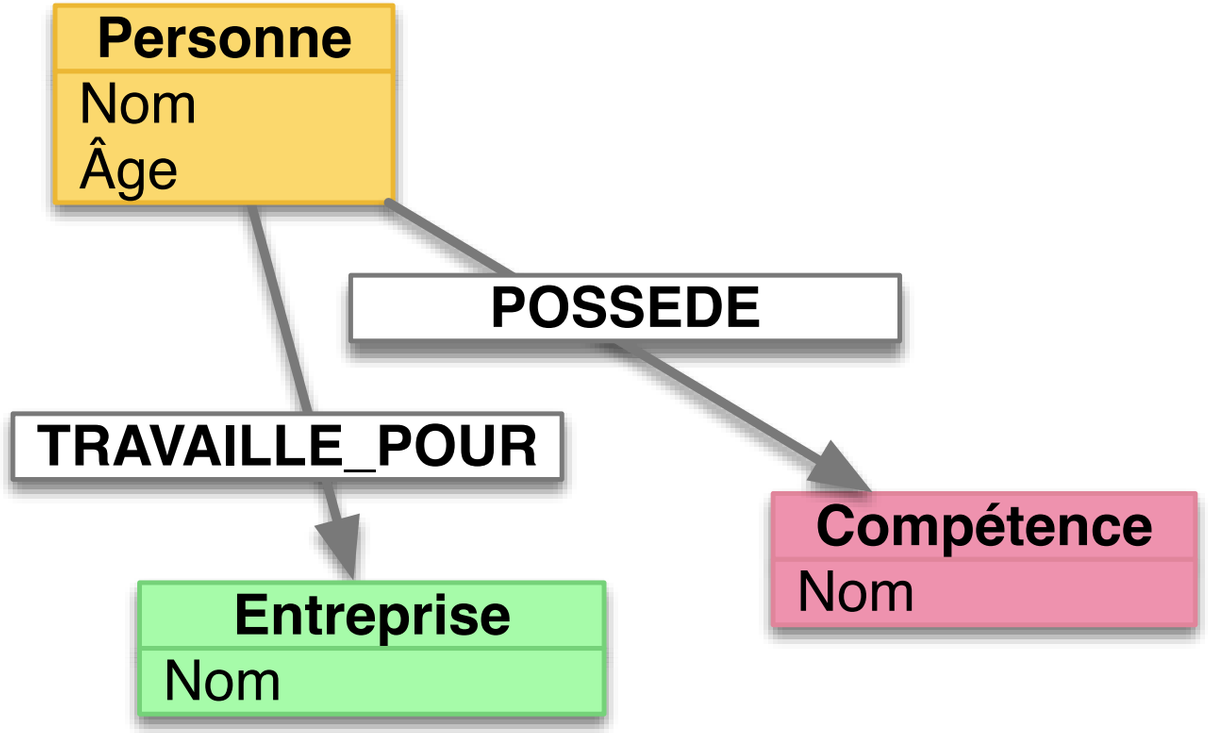




Neo4j

Modélisation - Mauvaises pratiques

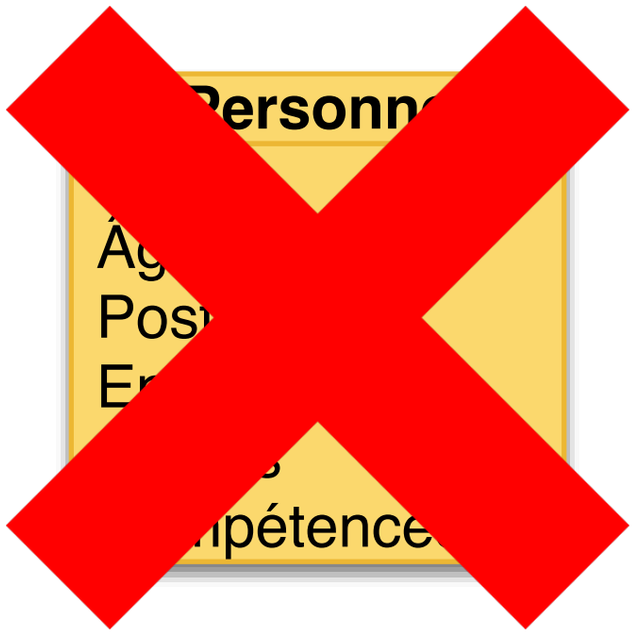
Nœuds représentants plusieurs concepts

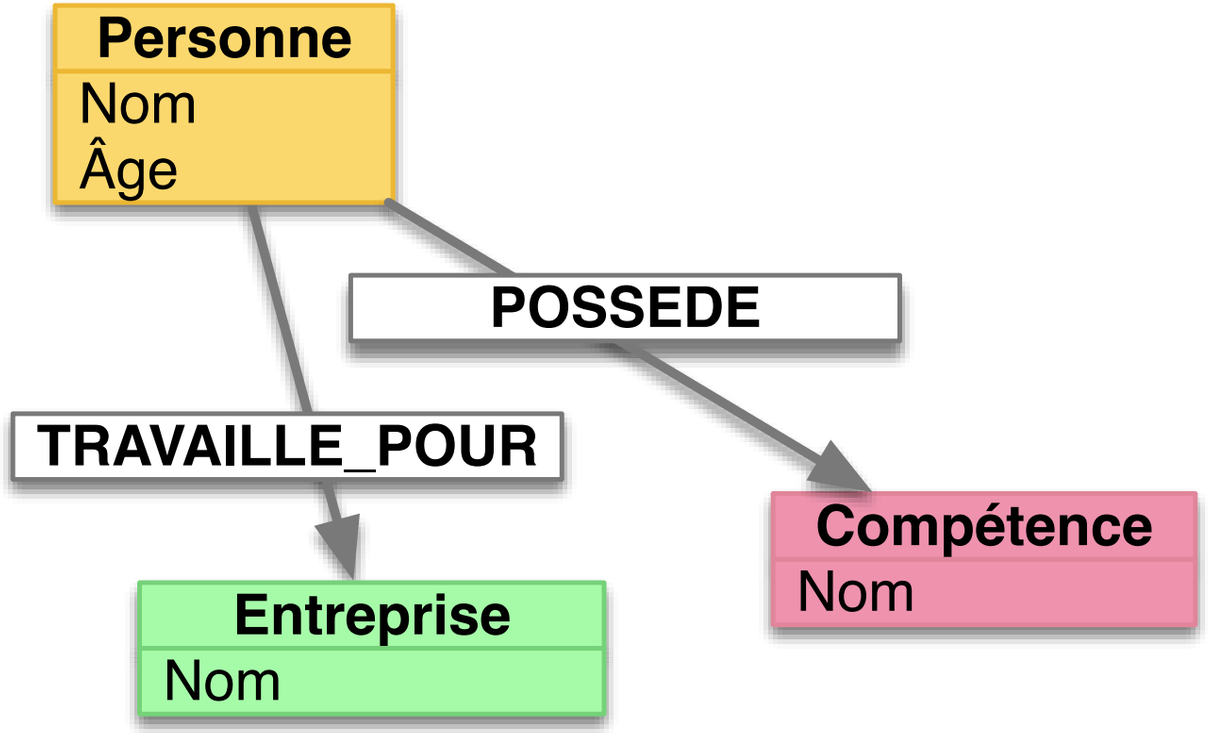


Neo4j

Modélisation - Mauvaises pratiques

Nœuds représentants plusieurs concepts

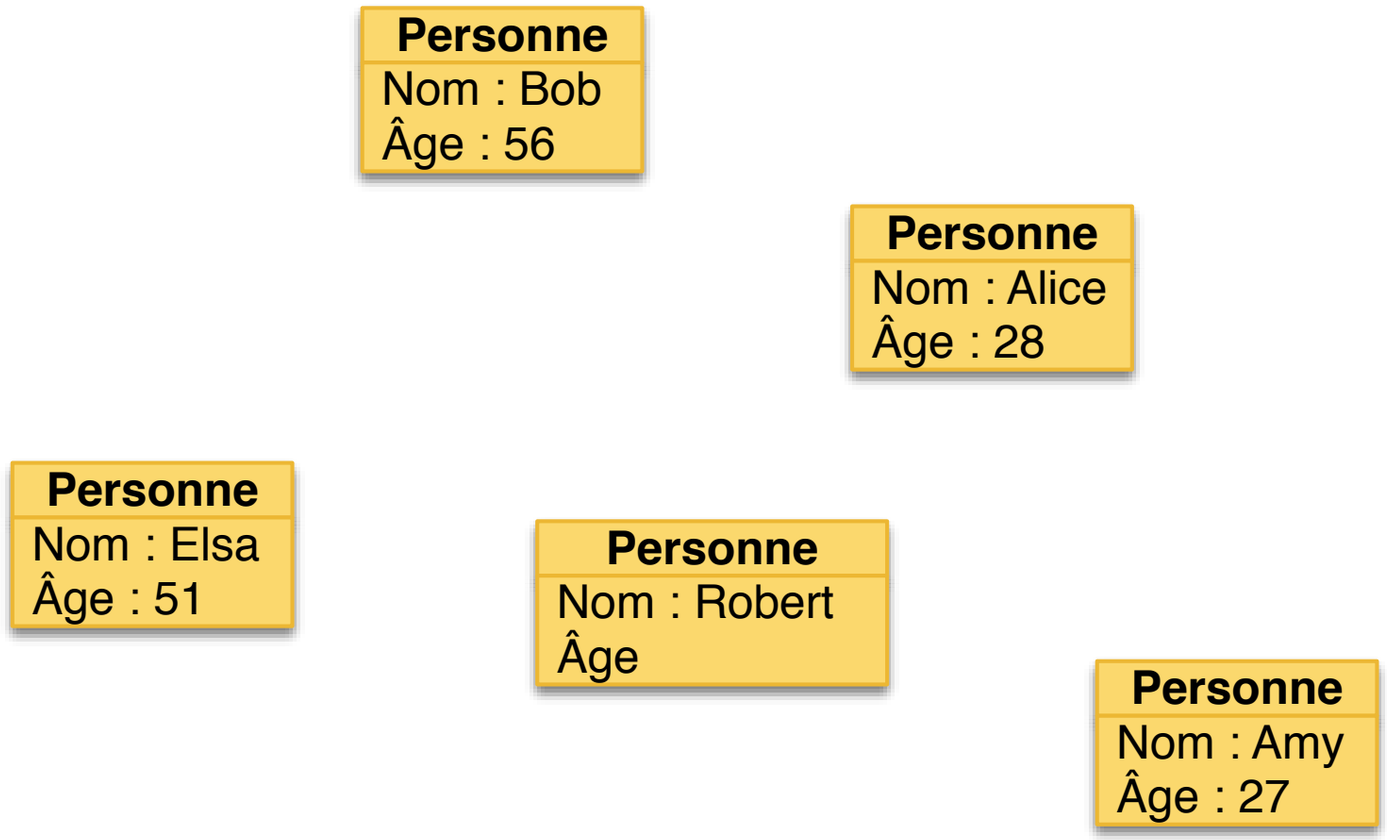




Neo4j

Modélisation - Mauvaises pratiques

Graphes non connectés

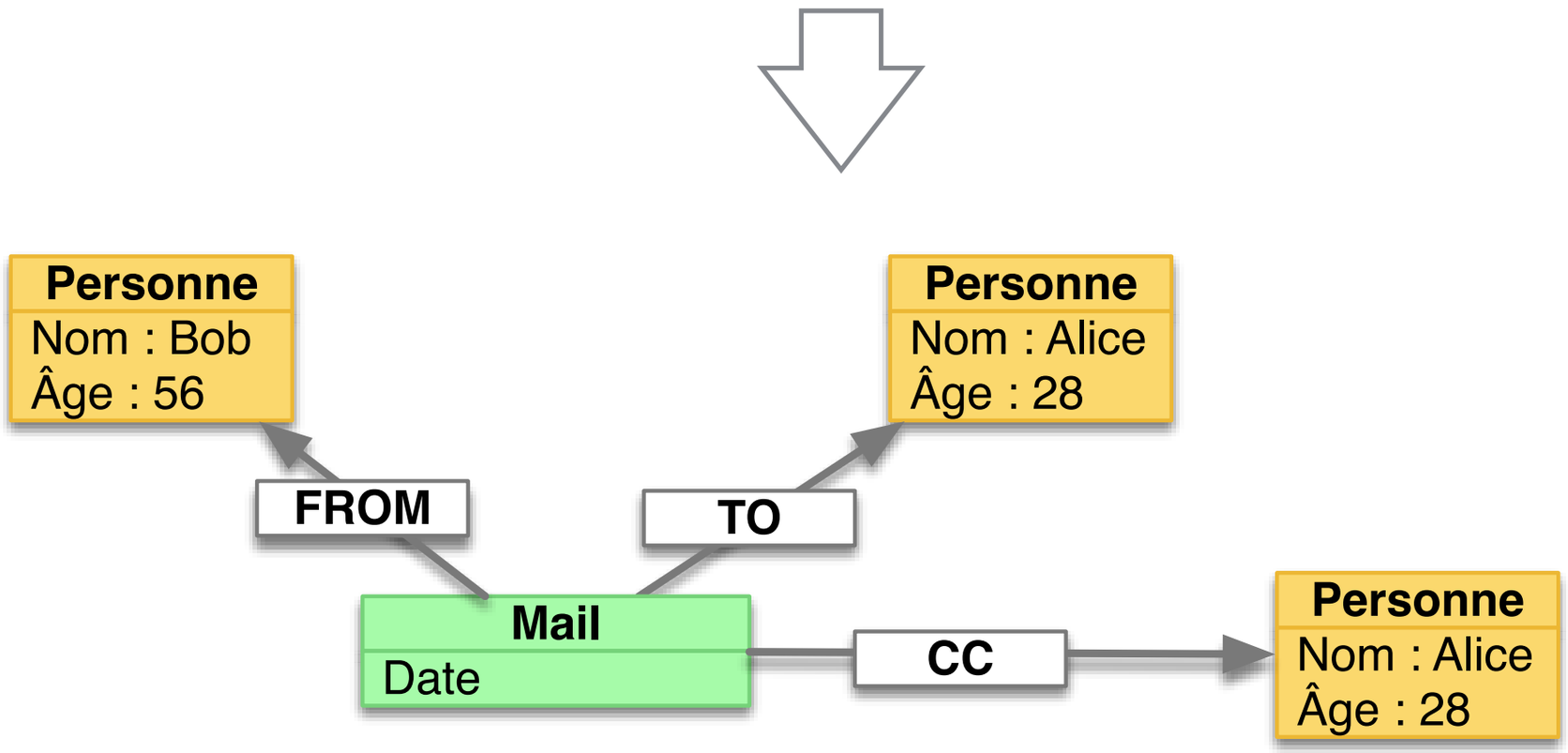


Les BD graphes sont faites pour les objets connectés

Neo4j

Evolution d'un graphe

D'une relation à des nœuds



Neo4j

Combinaison de domaines

- Démarrer avec un seul domaine
- Ajouter des domaines connectés en fonction de l'évolution du système et des besoins
- Permet de répondre à différentes requêtes
- Très facile d'ajouter des relations, des nœuds et des labels

Facebook Graph Search encode :

- Le graphe social
- Le graphe des localisations
- Le graphe des activités
- Le graphe des favoris
- …

Bases de données graphe

Forces et Faiblesses

Quand utiliser ?

- Données connectées

Quand ne pas utiliser ?

- Données peu connectées
- Mises à jour fréquentes sur une grande partie du graphe
- Analyse globale du graphe
- Quand on ne connait pas le point d'entrée d'une requête

Des questions ?

Neo4j
Resources

- Wikipedia
- Interactive Online Course http://neo4j.com/graphacademy/online- training/
- http://de.slideshare.net/thobe/an-overview-of-neo4j-internals
- The Neo4j Manual https://neo4j.com/docs/developer-manual/current/
- https://neo4j.com/developer/cypher/
- D. Montag. Understanding Neo4j Scalability.
- https://neo4j.com/docs/developer-manual/current/cypher/
- http://neo4j.com/use-cases/
- http://neo4j.com/docs/stable/tutorials-java-embedded-hello-world.html
- http://neo4j.com/docs/stable/tools.html
- https://neo4j.com/docs/cypher-refcard/current/